wrapblock

An image beside its own text, which wraps around it and runs below

Antonin Marchand

Version 0.1.5 — 26 November 2026

Abstract

The wrapblock package provides one environment, which sets an image beside a block of text. The text is narrowed along the image, then takes the full width again below it. The block is closed: the text that follows the environment never runs beside the image.

The package gives the image no size of its own. The room it takes from the text is the width and the height of the image as it is given; only the width can be forced, with a key.

Four properties set it apart from the other wrapping packages:

1. the text beside the image may hold lists, nested to any depth, and a whole block may be the first thing in a list item, right after `\item`;
2. the image and the lines beside it stay on the same page;
3. it works with pdfLaTeX and with LuaLaTeX;
4. in a tagged PDF, the structure is what it would be without the package.

Contents

1 What the package does **2**
- 1.1 An example . . . 2
- 1.2 Near the bottom of a page . . . 3
- 1.3 What it does not do . . . 3

2 Three things to know **3**
- 2.1 The text should be at least as long as the image . . . 3
- 2.2 Near the bottom of a page, the text is typeset twice . . . 3
- 2.3 No verbatim in the block . . . 4

3 Usage **4**
- 3.1 Requirements . . . 4
- 3.2 The environment . . . 5
- 3.3 Keys . . . 5
- 3.4 Searching the width: `width=fit` . . . 6
- 3.5 What the image may be . . . 7
- 3.6 Image on the left . . . 7

4 What the block may contain **8**
- 4.1 Lists . . . 8
- 4.2 Displayed equations . . . 9
- 4.3 Footnotes, inline formulas . . . 9
- 4.4 Centred or ragged text . . . 9
- 4.5 Boxes and wide material . . . 9
- 4.6 Floats . . . 9

5 Where the environment may be used **10**

6 Tagged PDF **10**

7 Messages **10**

8 How it works, in short **11**

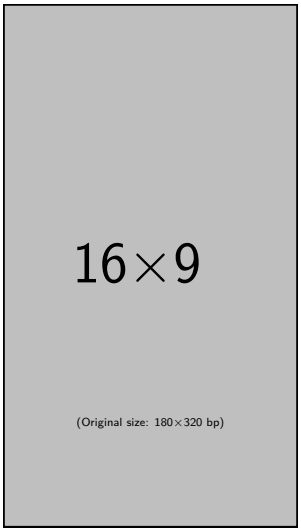
9 Other packages **12**

10 History **12**

11 Implementation **14**
11.1 The principle 14
11.2 Identification and state 16
11.3 Keys 18
11.4 The search of `width=fit` 19
11.5 The environment 22
11.6 Is there room for the image? 25
11.7 Each paragraph of the block 29
11.8 Setting the image 35
11.9 Keeping the zone on one page 39
11.10 Displayed equations 39
11.11 The deferred image 40
11.12 Boxes 47

1 What the package does

1.1 An example



This paragraph is typeset by wrapblock. The text runs along the image, on its left, then takes the full width again once it is past it.

- A list beside the image is narrowed too: each of its lines stops where the room of the image starts, like the lines of a paragraph.
- Its labels stay in their place, and a long item such as this one is broken at the same limit as the text around it.

The text after the list goes on wrapping, down to the bottom of the image and beyond. A ball of mass m bounces on the floor; we look for its velocity and angular velocity after the impact, assuming a point contact where the ball may slip. A ball of mass m bounces on the floor; we look for its velocity and angular velocity after the impact, assuming a point contact where the ball may slip. A ball of mass m bounces on the floor; we look for its velocity and angular velocity after the impact, assuming a point contact where the ball may slip.

This paragraph comes after the environment. It is full width, however long the block is: when the text of the block is shorter than the image, vertical space is added down to the bottom of the image (section 2).

In this documentation, the *block* is the contents of the environment, the *image* is whatever is set beside it (a picture, a diagram, a table), and the image is *deferred* when it is set at the top of the next page.

1.2 Near the bottom of a page

The image and the lines beside it are never split by a page break. When a block starts near the bottom of a page, one of three things happens:

1. *The image fits.* The text wraps around it, and what is left of the text runs at full width on the next page.

2. *The image does not fit, and the text is long.* The text runs at full width to the bottom of the page. The image is deferred: it goes to the top of the next page, where the text wraps around it.

3. *The image does not fit, and the text is short.* At full width, the text would end on this page and leave the image alone on the next one. The whole block moves to the next page instead, together with the heading that precedes it, if any. The bottom of the page is left empty, and a warning in the `.log` says so: the author may want to change something.

Below the image, the text of the block breaks like any other text. An example that comes with the package shows the three cases with long, medium and short texts: `wrapblock-mwe-pagebreak.tex`.

1.3 What it does not do

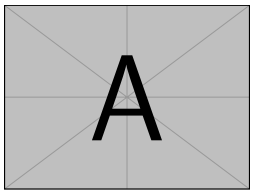
The package does only this. It does not wrap an image in the middle of a paragraph, flow text around a shape, or float the image: for those, see wrapfig2, wrapstuff, cutwin or picinpar. Section 9 explains why none of them has the four properties above.

2 Three things to know

2.1 The text should be at least as long as the image

The block is closed: the text after it never runs beside the image. When the text of the block is shorter than the image, the block ends with vertical space, down to the bottom of the image:

A short text, two lines long, beside a taller image: the block ends below the image, and leaves this blank.



This paragraph comes after the block. Lengthening the text of the block, taking the next paragraph into it or making the image smaller removes the blank.

wrapfig, wrapfig2 and wrapstuff leave no blank: their image is not closed in a block, so the paragraphs that follow run beside it in turn, until they are past it (checked with both engines, November 2026). The price is that whatever follows is narrowed: a heading, or a list, which wrapfig cannot narrow. With wrapblock, the text that runs beside the image is always the text of the block, and nothing else.

2.2 Near the bottom of a page, the text is typeset twice

When the image does not fit at the bottom of the page, the package must know how long the text is, to choose between cases 2 and 3 of section 1.2. It typesets the text of the block a first time, at full width, in a box that it then throws away. Everywhere else on the page, the text is typeset once.

This first pass leaves almost no trace:

- LaTeX counters, those made by `\newcounter` — equations, footnotes, list items, theorems, sections — are put back after the first pass: the numbers are right.
- Labels, footnotes, index entries, table-of-contents lines, floats and marginal notes vanish with the box: they come out once.
- But anything else the text does globally is done twice: a `\gdef`, an `\immediate\write`, a TeX count stepped globally, a global expl3 integer. The first use of an acronym is a common case, checked with glossaries and with acro: the first use, which prints the long form, is spent on the first pass, and the text shows the short form.

When the text holds such a command, there are two ways out. The key `defer=false` turns the first pass off: a block whose image does not fit then always moves to the next page (case 3), and its text is typeset once. Or the command can be moved out of the block: the first use of an acronym, for instance.

The image, for its part, is typeset twice at every block, wherever it is on the page: once to measure it, once for good. The LaTeX counters it steps (a numbered equation, a caption) are put back after the first time; anything else it does globally is done twice.

2.3 No verbatim in the block

The text of the block is read as an argument before it is typeset, so that it can be measured. `\verb`, `verbatim`, `lstlisting` or `minted` cannot be used in it: they give an error. Verbatim material can be typeset beforehand in a box, which the block then uses:

```
\newsavebox\code
\begin{lrbox}{\code}\verb|\int_gincr:N|\end{lrbox}
\begin{wrapblock}{\includegraphics{diagram}}
  The function \usebox{\code} steps an integer.
\end{wrapblock}
```

3 Usage

```
\usepackage{wrapblock}
...
\begin{wrapblock}[side=l, hsep=1.5em]{\includegraphics[width=4cm]{diagram}}
  The text of the block: paragraphs, lists, equations.
\end{wrapblock}
```

3.1 Requirements

- A LaTeX kernel of June 2023 or later; the package checks it. It was tested with TeX Live 2026 only.
- pdfLaTeX or LuaLaTeX: the tests run with both. With XeLaTeX, two of the examples (`wrapblock-mwe-pagebreak`, `wrapblock-mwe-lists`) were compiled once and gave the expected pages, but the tests do not run it.
- One correction needs a kernel of June 2025 or later: the second pass that makes room for a footnote before a deferred image (message “The page broke too early”, section 7). An older kernel does not have the hook it uses: the correction is then never made.

3.2 The environment

`\begin{wrapblock}[`⟨*keys*⟩`]{`⟨*image*⟩`}`
⟨*environment content*⟩
`\end{wrapblock}`

Sets ⟨*image*⟩ beside the contents of the environment. ⟨*image*⟩ is any material in horizontal mode: an `\includegraphics`, an `\includestandalone`, a `tikzpicture`, a `tabular`. Its top is aligned with the top of the first line of the block. The package does not scale it: the image has the size it is given, with the `width` or `height` of `\includegraphics` for instance, and the text is narrowed by that width, over that height. The `width` key below can only reserve a wider room, in which the image is centred; the image is then typeset as a `minipage` of that width, and may hold a caption. The environment starts a new paragraph, and so does the text after it.

The contents are read as an argument, before they are typeset: they cannot hold verbatim material (`\verb`, `verbatim`); see section 2.

`\wrapblockset{`⟨*keys*⟩`}`

Sets the keys for all the following blocks, within the current group.

3.3 Keys

Key	Default	Meaning
`side`	`r`	side of the image: `r` (right) or `l` (left); `l` and `r` alone are short forms
`width`	`natural`	width reserved for the image: `natural`, the width of the image itself, a length, or `fit`, searched
`fit min`, `fit max`	`0.1`, `0.9`	bounds of the search of `width=fit`, as fractions of the line
`hsep`	`1em`	horizontal space between the image and the text
`vsep`	`0.5\baselineskip`	vertical space below the image
`defer`	`true`	whether the image may go to the top of the next page

`/wrapblock/side=l|r` (default `r`)

The side of the image: right (`r`) or left (`l`) of the text.

`/wrapblock/l`

Short for `side=l`. Likewise `r` for `side=r`.

`/wrapblock/width=natural|`⟨*length*⟩`|fit` (default `natural`)

The width reserved for the image. With `natural`, it is the width of the image itself. With a ⟨*length*⟩, the image is typeset as a `minipage` of that width, centred: this aligns several blocks whose images differ in width, and lets the image hold paragraphs or a caption below it (`\captionof`). The length is taken as it is: a room narrower than the image lets the image stick out on the right, with an `Overfull \hbox` from TEX.

In that `minipage`, `\linewidth` and `\textwidth` are the width of the block, not of the page: an image of width `.5\linewidth` is half as wide as the block, and changes with it. For a fraction of the full line, whatever the width of the block, use `\wrapblocklinewidth` (below).

With `fit`, the package searches the width itself (section 3.4). It is never the default.

`/wrapblock/fit min=`⟨*number*⟩ (default `0.1`)

The narrowest width tried by `width=fit`, as a fraction of the line.

`/wrapblock/fit max=`⟨*number*⟩ (default `0.9`)

The widest width tried by `width=fit`, as a fraction of the line.

`\wrapblocklinewidth` The full line of the block: `\linewidth` where the block starts, before the image takes its share. It is meant for the image, where `\linewidth` and `\textwidth` are the width of the block once the key `width` is given. With

```
\includegraphics[width=.33\wrapblocklinewidth]{...}
```

the image is a third of the line in any block, whatever its width; this holds for a deferred image as well.

`/wrapblock/hsep`=⟨*length*⟩ (default `1em`)

The horizontal space between the image and the text.

`/wrapblock/vsep`=⟨*length*⟩ (default `0.5\baselineskip`)

The vertical space below the image, before the text takes the full width again.

`/wrapblock/defer=true|false` (default `true`)

Whether the image may go to the top of the next page while the text fills the bottom of this one (case 2 of section 1.2). With `false`, a block whose image does not fit at the bottom of the page always moves to the next page as a whole (case 3), and its text is not typeset a first time to be measured (section 2).

The defaults of `hsep` and `vsep` are computed once, when the package is loaded, with the font and the `\baselineskip` of that moment: those of the normal size of the document. A length given in the document is computed where it is given.

3.4 Searching the width: `width=fit`

An image with a caption below it is a trade: the wider the block, the fewer lines of caption, but the narrower the text beside the image, and the taller. An image whose width is a fraction of `\linewidth` grows with the block instead. `width=fit` looks for the width where the image and the text beside it are as tall as each other: the block is then as low as it can be, with no blank below the shorter of the two.

At a given width, the image is built as it will be, in its `minipage`, and the text is set in the room left beside it, both in boxes that are thrown away, like the measure of section 2. The width is first tried at `fit min` and at `fit max`. If the image is the taller at one and the shorter at the other, the two heights cross between them, and a bisection closes on the crossing to 2 % of the line: six more trials with the default bounds. If the two bounds agree, the middle is tried as well: an image that grows with the block may be the taller there, between a text that is the taller beside a small image and one that is the taller in a column that has all but vanished. With no crossing found, the package says so (warning `fit`) and the image keeps its natural width. The width found is written to the `.log` (information `fit-width`), to be copied into the key `width` once the text is final.

The cost is that of the measure, eight or nine times: the text and the image are typeset at each trial, with the same limits (no verbatim; counters put back, but not a global count or a `\write`). Narrow trials do not report their `Overfull` boxes, since an image of fixed width is wider than a block at 0.1 of the line. Choose `fit min` no narrower than such an image: a crossing below its width would leave it sticking out of the block.

```
\begin{wrapblock}[width=fit, fit max=.5]{%
  \centering
  \includegraphics[width=.33\wrapblocklinewidth]{moseley}%
  \captionof{figure}{A long caption, on as few lines as the text allows.}}
Text...
\end{wrapblock}
```

3.5 What the image may be

The image is typeset the way `\mbox` typesets its contents: on one line, without line breaks. Anything that can go in an `\mbox` can be an image. These were tried, with both engines:

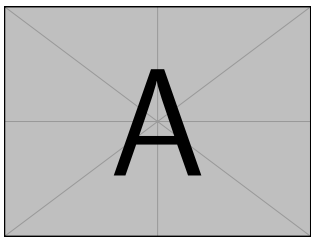
Given as the image	What happens
`\includegraphics`, `tikzpicture`, `tabular`	as expected
`minipage`, `\parbox`	as expected; this is the way to give paragraphs, a list or a displayed equation as the image
a caption (`\captionof`)	as expected with the `width` key, which makes the image a `minipage`; without it, the errors of a caption in an `\mbox` ("You can't use \hrule here")
`tcolorbox`	as expected; a `breakable` one does not break: near the bottom of a page it is deferred or moved as a whole, like any image
a few words	set on one line beside the text
nothing	the text is narrowed by `hsep` on its first line only
a long text, without `\parbox`	one very long line: see "too wide" below
a list or a displayed equation, without `minipage`	errors of LaTeX ("Not allowed in LR mode", "Bad math environment delimiter"): they need a `minipage`
`\verb`, `verbatim`	errors: the image is read as an argument. Typeset the verbatim material beforehand in a box, and give `\usebox` as the image (section 2)
a `figure`, a `\marginpar`	the error "Not in outer par mode": a float cannot be the image; for a caption, use `\captionof` with the `width` key
a footnote in a box (`\fbox`, `tabular`)	its text is lost, as in any box: use `\footnotemark` in the image and `\footnotetext` in the text. In a `minipage`, the footnote is set at the foot of the `minipage`
another `wrapblock`	the error "blocks cannot be nested"

Two sizes are special:

- *An image too wide.* When less than 3 em would be left for the text beside the image, the text is not wrapped: the image is set on a line of its own, above the text, and a warning says so.
- *An image taller than the page.* It fits nowhere: the block starts a new page, the image runs out of the page at the bottom, and a warning says so.

3.6 Image on the left

```
\begin{wrapblock}[l]{\includegraphics[width=0.25\linewidth]{example-image-a}}
```



A ball of mass m bounces on the floor; we look for its velocity and angular velocity after the impact, assuming a point contact where the ball may slip. A ball of mass m bounces on the floor; we look for its velocity and angular velocity after the impact, assuming a point contact where the ball may slip. A ball of mass m bounces on the floor; we look for its velocity and angular velocity after the impact, assuming a point contact where the ball may slip. A ball of mass m bounces on the floor; we look for its velocity and angular velocity after the impact, assuming a point contact where the ball may slip.

4 What the block may contain

4.1 Lists

A list in the block is narrowed along the image, labels included, at any depth; it gets its normal width back below the image.

A block may also be the first thing in an item of a list, right after `\item`, as the second question below: the image is then aligned with the margin of the item and, on the left, the label stays in the margin, before the image.

```
\begin{enumerate}
\item A question without an image.
\item \begin{wrapblock}[l]{\includegraphics[width=3cm]{diagram}}
  A question with its diagram, on the left: ...
\end{wrapblock}
\item The next question, after the block.
\end{enumerate}
```

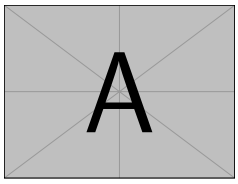
A block may also hold several items of a list that is opened before it and closed after it, with the image on the right:

```
\begin{enumerate}
\item A question without an image.
\begin{wrapblock}{\includegraphics[width=3cm]{diagram}}
\item The first question beside the diagram.
\item The second one.
\end{wrapblock}
\item The next question, after the block.
\end{enumerate}
```

With the image on the left, this does not work: only the label of the first item is kept out of the image, and the labels of the following items are set over it.

What cannot be done is to open a list in the block and close it after the block, or the reverse: the block is an environment, and LaTeX wants environments nested.

1. A question without an image. A ball of mass m bounces on the floor; we look for its velocity and angular velocity after the impact, assuming a point contact where the ball may slip.

2. 

   A question with its diagram, on the left:

   (a) a sub-question A ball of mass m bounces on the floor; we look for its velocity and angular velocity after the impact, assuming a point contact where the ball may slip.

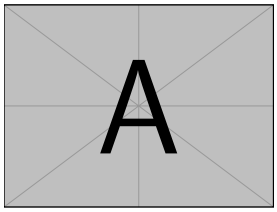
   (b) another one.

A ball of mass m bounces on the floor; we look for its velocity and angular velocity after the impact, assuming a point contact where the ball may slip.

3. The next question, after the block.

4.2 Displayed equations

A displayed equation (`\[ \]`, `equation`, `align`…) is centred in the width left beside the image. After it, the paragraph goes on wrapping from where the equation ended.



The conservation of angular momentum about the contact point gives

$$J\omega' + mRv' = J\omega + mRv,$$

and Coulomb's law, while the contact slips,

$$\|\vec{T}\| = f\|\vec{N}\|. \qquad (1)$$

The text goes on below the equation, still along the image as long as it has not passed it. A ball of mass m bounces on the floor; we look for its velocity and angular velocity after the impact, assuming a point contact where the ball may slip.

4.3 Footnotes, inline formulas

A footnote called in the block goes to the foot of the page, as anywhere else. A line taller than the others, with an inline fraction for instance, pushes the next ones down; since the package counts lines in `\baselineskip`, the wrapping may then last one line longer than needed, never one line shorter.

4.4 Centred or ragged text

`\centering`, `\raggedright` and `\raggedleft` work in the block, and so does a block in a `center`, `flushleft` or `flushright` environment: the lines are centred or ragged in the width left beside the image.

4.5 Boxes and wide material

The text of a box — `\parbox`, `minipage`, `p` column of a table — keeps the width it was given and is not wrapped. A box wider than the room left beside the image therefore overlaps the image; the package says so with a warning. A box set below the image raises no problem.

A `tcolorbox` is the exception: one that starts beside the image, between two paragraphs, takes the width left there, breakable or not, as it would in a list. Its text is not wrapped either: a box that starts beside the image and ends below it keeps that width. A box that opens the block is beside the image, but its line is taller than a line of text, and the image, aligned on the baseline of that line, comes down by as much: open the block with a line of text.

Other material that cannot be broken — a wide `tabular` without `p` column, a wide picture, a long formula — also overlaps the image when it is wider than the room left beside it. The package does not warn about it, but TEX does, with its usual `Overfull \hbox`.

4.6 Floats

A `figure` or a `table` may be declared in the block; it floats as usual. When a float lands at the top of the page that receives a deferred image, the image is set below it, beside its text.

5 Where the environment may be used

In running text and in lists All of the above.

In `twocolumn` mode and in `multicols` The block wraps its text within its column, and does not break from one column to the next while it runs along the image. The image is never deferred there: when it does not fit at the bottom of the column, the whole block moves to the next one.

In a box In a `minipage`, a `\parbox`, a `tcolorbox`…, the package cannot tell where it stands, and no page breaks there anyway. The text is then set in a `minipage` beside the image, aligned on its first line, without running below the image; an information message in the `.log` says so.

In another block Blocks do not nest: this is an error.

6 Tagged PDF

With `\DocumentMetadata{tagging=on}`, the text of the block is typeset in the main galley: paragraphs, lists and formulas keep the structure LaTeX gives them. The image is typeset at the start of the first paragraph of the block, as if it had been written there: an `\includegraphics` becomes a `Figure` element there, before the text it goes with, and its alternative text is given as usual, with the `alt` key:

```
\begin{wrapblock}{\includegraphics[width=4cm, alt={The ball}]{ball}}
```

A deferred image, set at the top of the next page, keeps its place in the reading order: the package makes an empty structure element where the block starts, and puts the image into it, in a `P` element, when its page is shipped out. It therefore comes before the text it goes with, as when it is not deferred.

An image of another kind — `tikzpicture`, `tabular`, `\fbox` — is tagged exactly as it would be at the start of a paragraph without the package; describing it is left to the author, as it is without the package.

The package was tried with both engines, `tagging=on` and `pdfstandard=ua-2`, with an `\includegraphics`, a `tabular`, a `tikzpicture`, a `\parbox`, a `minipage` and an `\fbox` as image: tagpdf gives no more warnings than with the same image at the start of an ordinary paragraph, and the structure tree is the same with pdfLaTeX and LuaLaTeX. veraPDF 1.30 finds the tagged example `wrapblock-mwe-tagged.tex` conformant to PDF/UA-2 with both engines, and finds nothing in the tests that the same document does not give without the package; with pdfLaTeX, a footnote breaks clause 8.2.2, with or without the package. The PDF/UA conformance of a whole document does not depend on this package only.

7 Messages

Each message names the input line where the block *ends*, the line of its `\end{wrapblock}`: the text of the block is read as a whole before anything is typeset, and TeX is on that line by then. Here is what each message means, and what to do about it.

`wrapblock blocks cannot be nested.`

Error. A block was opened in the text of another one, or in its image. Close the first block before opening the second.

`The image of the block that ends on input line … does not fit at the bottom of the page, and its text is too short …`

Warning. This is case 3 of section 1.2: the whole block moved to the next page, and the bottom of this page is empty. Shorten the image, lengthen the text, or move the block in the source.

`The image of the block that ends on input line … leaves only … for the text beside it: it is set above the text, which is not wrapped.`

Warning. Less than 3 em would be left for the text beside the image (section 3.5). Make the image narrower; a long text given as the image needs a `\parbox`.

`The image of the block that ends on input line … is taller than the page.`

Warning. The image runs out of the page at the bottom. Make it smaller.

`A box of width …, in the block that ends on input line …, sits beside the image, where only … is left.`

Warning. A `\parbox`, a `minipage` or a `p` cell beside the image is wider than the room left there, and overlaps the image. The warning is given once per block, for the first such box. Make the box narrower, or put it below the image.

`The page broke too early in the text of the block that ends on input line …`

Warning. The image is deferred, and a footnote in the lines that run at full width at the bottom of the page took room that was not counted: a page with only a few lines is left before the image. Rerun LaTeX: the room the footnote took is written to the `.aux` file, and the next run counts the lines without it (this needs a kernel of June 2025, section 3.1).

`The text of the block that ends on input line … ended before the bottom of the page: its image is set after it.`

Warning. The image was deferred because the text was measured long enough to reach the next page; but the text ended on this page, and the image comes after it. This should not happen; a float or a very tall line in the text can cause it. Use `defer=false` for this block.

`The block that ends on input line … moved to a new page while its text was set.`

Warning. The lines beside the image moved to another page after their number was counted, which only a very large footnote can cause. The wrapping may be wrong by a few lines. The tests could not make this happen: TeX moves the whole block before it comes to that.

`The block that ends on input line … is inside a box …`

Information, in the `.log` only. The block is in a `minipage`, a `\parbox` or the like: its text is set beside the image, without wrapping under it (section 5).

8 How it works, in short

The text of the block is typeset where it stands, in the main galley, as ordinary paragraphs. Nothing is put in a box: this is what leaves page breaks, lists and tagging untouched. The package only changes the shape of the paragraphs.

- *The image* is typeset at the start of the first line of the block, in a box with no width, height or depth.
- *Each paragraph* of the block is narrowed at its start, by a hook: the package reads on the page where the paragraph starts, works out how many of its lines run beside the image, and gives those lines a shorter length with `\parshape`.
- *No page break* may separate the lines beside the image: penalties forbid it inside the paragraphs and between them.

- *When the image does not fit* at the bottom of the page, the text is typeset once at full width, in a box that is thrown away, to measure it. A short text moves to the next page with its image. A long text stays: the package counts the lines that still fit on the page, forces the page break after the last of them, and narrows the lines that follow; the image is drawn at the top of the next page when that page is shipped out.

Section 11, at the end of this documentation, prints the whole code and explains it.

9 Other packages

These packages were tried on the same scenes as wrapblock (short text, list beside the image, image in a list, equations, image set lower and lower on the page), with both engines, in November 2026.

Package	What was observed
wrapfig 3.6	Does not work in lists, as its documentation says; at the bottom of the page, the image overflows the text area.
wrapfig2 7.0.2	An image in a list breaks the list; at the bottom of the page, the image overflows as with wrapfig.
wrapstuff 0.3	Handles page breaks, by moving the block on the next run; but loses text when a list or a displayed equation runs along the image.
sidefig-latex	Wraps lists well; at the bottom of the page, the image goes out of the text area. Not in TeX Live.
side-by-side `minipages`	The text does not run below the image, and the block never breaks.

cutwin, picinpar and floatflt were not tried: according to their documentation, the first two cut a window in a paragraph, and the third one floats the image, which is a different need.

10 History

0.1 (15 November 2026) First version.

0.1.1 (18 November 2026) Corrections:

- in centred or ragged-left text, an image on the left was moved over the text;
- a deferred image was drawn over a float at the top of its page; it is now set below it;
- a float or a `\marginpar` in a block near the bottom of a page gave an error;
- a float in a block gave the warning about a box that is too wide;
- the messages now say that the line they give is that of the end of the block.

The documentation was rewritten, and now ends with the whole code, explained.

0.1.2 (18 November 2026) The image:

- the default of the `width` key is now `natural`, the width of the image itself, instead of `0pt`; a length, `0pt` included, is always taken as it is;
- an image that leaves less than 3 em beside it is set above the text, which is not wrapped;
- an image taller than the page no longer leaves an empty page;
- the counters stepped by the image (an equation, a caption) are no longer stepped twice;
- a block given as the image of another is an error.

Lists: a block may hold items of a list opened before it and closed after it. A block moved to the next page no longer starts with a blank when its text starts with an `\item`, a list or a heading, and it keeps the indentation of the list around it.

The documentation says what the image may be (section 3.5).

0.1.3 (18 November 2026) No change in the code. The README says where to report a bug, and the documentation reads the code of the package wherever TEX finds it, so that it compiles from one folder.

0.1.4 (19 November 2026) Loading the package no longer slows down a document: the foreground of a page is asked for only when a deferred image waits for it. A document without any block compiled 20 % slower with pdfLATEX and 40 % slower with LuaLATEX; it now takes the same time as without the package.

0.1.5 (26 November 2026) The examples are renamed `wrapblock-mwe-…`. New: `width=fit`, with `fit min` and `fit max`, which searches the width where the image and the text beside it are as tall as each other; `\wrapblocklinewidth`, the full line, for an image whose width is a fraction of the line in a block of any width. Corrections:

- a paragraph that starts inside a group, such as `{\bfseries Bold} text` or a breakable `tcolorbox`, lost its shape when the group ended, and ran over the image;
- a `tcolorbox` between two paragraphs, beside the image, was as wide as the line and overlapped the image: it now takes the width left beside the image;
- with the `width` key, the image is a `minipage` of that width: it may hold a caption (`\captionof`), which gave errors;
- a breakable `tcolorbox` below the image, right after one beside it with no blank line between them, kept the shape of the first: it was moved right by the width of the image, and ran out of the page;
- a block right after a heading, when the paragraph after the heading is not indented, lost its image (0.1.4 did too);
- a block right after a deep line, such as a `tabular` centred on its axis, took its first line for higher than it was, by up to the depth of that line: TEX puts `\lineskip` below such a line, not `\baselineskip`. The image came lower than the room kept for it, and a heading after the block ran into it;
- a `tcolorbox` right after a paragraph, with no blank line between them, came while that paragraph was still open: with a deferred image not yet committed, or with a key such as `center`, which fixes the width when it is read, it kept the full line, overlapped the image and gave an overfull box;
- a block that opens with a `tcolorbox` put its image one line too high, over the end of the paragraph before, when the box is breakable, and below the box when it is not;
- a block whose text is shorter than its image, followed by a heading that goes to the next page, left its image out of the page: the space kept below the image went to the next page with the heading. The heading after such a block is now as far from the image as from a line of text: the larger of its space and `vsep`.

11 Implementation

This section prints the whole of `wrapblock.sty` and explains it. It is meant for someone who wants to check, repair or extend the package; using the package does not require reading it.

How the code is shown. The code is read from the file itself when this documentation is compiled, so that the two cannot drift apart. The line numbers are those of the file. The pieces are shown in the order in which they are best read, from the environment down to the details, not in the order of the file, where a function is declared before it is used. In the file, each piece is framed by two comment lines, `% <name>` and `% </name>`, which are only there for this documentation.

Names. The package is written in expl3. Its functions are named `\__wrapblock_...`, its global variables `\g__wrapblock_...` and its local ones `\l__wrapblock_...`. A few LaTeX 2_ε internals are used under their own names: `\@labels`, `\@totalleftmargin`, `\@itempenalty`, `\cl@@ckpt`, `\@arrayparboxrestore`, `\@toplist`, and the switches `@inlabel`, `@nobreak`, `@twoside`, `@twocolumn`.

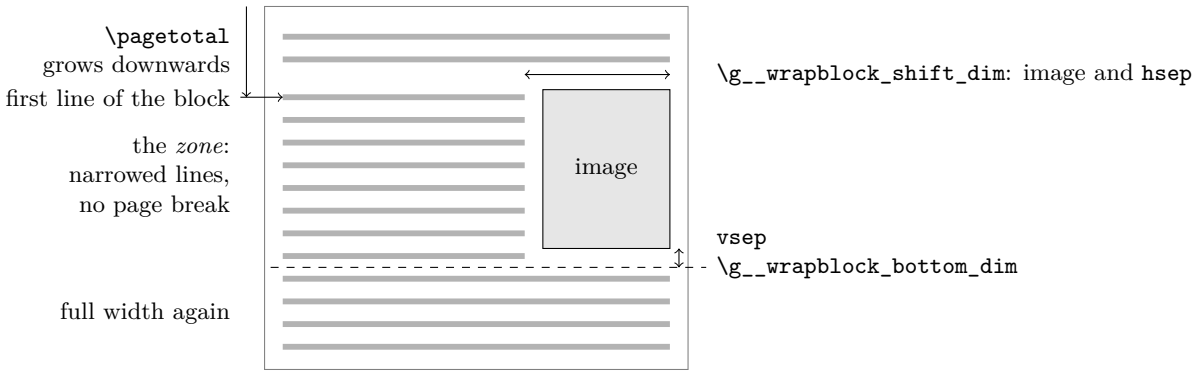
11.1 The principle

The text stays in the galley. Most packages that wrap text around an image set the text in boxes of their own. wrapblock does not: the text of the block is typeset where it stands, in the main vertical list, as if the package were not there. The package only acts at the start and at the end of each paragraph, through the paragraph hooks of LaTeX, and changes three things there:

- the `\parshape` of the paragraph, to narrow the lines that run beside the image;
- its `\interlinepenalties`, to forbid a page break between those lines;
- the penalties of lists and displayed equations, to forbid a page break between the paragraphs beside the image.

LaTeX sees ordinary paragraphs: this is why lists, page breaks and tagging work in the block as they do elsewhere.

The picture to keep in mind.



Positions on the page. To know which lines run beside the image, the package needs the position of each line on the page. It reads it from `\pagetotal`, the height of the material already on the current page. Every position it keeps is expressed the same way: the value `\pagetotal` would have at that point, measured downwards from the top of the text area. The main one is `\g__wrapblock_bottom_dim`, the bottom of the area taken by the image, `vsep` included. A line whose top lies above `\g__wrapblock_bottom_dim` is narrowed; the others keep their width.

Vocabulary. The code and this commentary use these words:

zone the lines beside the image. It starts with the first line of the block and ends with the first paragraph that goes below the image. No page break may happen inside it. `\g__wrapblock_active_bool` is true while it runs.

pending the image is not typeset yet; the first paragraph of the block will place it (`\g__wrapblock_pending_bool`).

deferred the image does not fit at the bottom of the page and will be drawn at the top of the next one, while the text starts on this page at full width.

waiting a deferred image whose page has not broken yet (`\g__wrapblock_wait_bool`).

committed a deferred image whose page is known; it is drawn when that page is shipped out.

The life of a block.

1. The environment reads its arguments, typesets the image once to measure it, and compares its height with the room left on the page (`\__wrapblock_begin:n`).
2. If the room is too small, `\__wrapblock_decide:n` chooses: either the whole block moves to the next page (`\__wrapblock_need_space:`), or, when the text is long enough, the block stays and the image is deferred.
3. At the start of each paragraph, the `para/begin` hook computes how many lines of the paragraph run beside the image, and sets `\parshape` (`\__wrapblock_para_begin:`, `\__wrapblock_shape:Nnnn`). In the first paragraph, it also puts the image at the start of the first line (`\__wrapblock_place_image:`).
4. At the end of each paragraph, the `para/after` hook decides whether the zone goes on; if it does, it forbids a page break before the next paragraph (`\__wrapblock_para_after:`).
5. At the end of the environment, vertical space is added if the text is shorter than the image (`\__wrapblock_end:`).

A block inside a box (a `minipage`, a `\parbox`) takes another, much simpler road (section 11.12).

Where the package hooks in.

Hook	What the package does there
`para/before`	keeps `\prevdepth`, needed to place the first line
`para/begin`	sets the shape of the paragraph; places the image in the first one
`para/end`	removes a shape left over by an earlier paragraph; clears three penalties in the paragraph that ends the page (deferred image)
`para/after`	ends the zone or keeps it together; commits a deferred image
`\everydisplay`	sets the shape again after a displayed equation
`\@arrayparboxrestore`	at the start of a box in the block: warns if it is too wide, and turns the package off inside
`shipout/before`	asks for the foreground of the page, only when an image waits
`shipout/after`	counts the pages shipped out
`shipout/foreground`	draws the deferred images of the page
`build/column/before`	measures the floats at the top of the page; notices a page that broke too early
`enddocument/afterlastpage`	writes to the `.aux` file the room footnotes took

11.2 Identification and state

```
\NeedsTeXFormat{LaTeX2e}[2023-06-01]
\ProvidesExplPackage{wrapblock}{2026-11-26}{0.1.5}
  {Closed block with an image beside its text}
```

The package asks for a kernel of June 2023: an older format stops with a clear message. Older kernels were not tried, and the package was tested with TeX Live 2026 only. The hook `build/column/before` exists since June 2025 (section 3.1).

```
% ---------------------------------------------------------------- state
% All positions are in page coordinates: the value \pagetotal would have
% at that point, from the top of the text area.
\box_new:N  \g__wrapblock_img_box
\tl_new:N   \g__wrapblock_img_tl        % the image, as given
\dim_new:N  \g__wrapblock_img_width_dim % its width key (-\maxdimen: natural)
\dim_new:N  \l__wrapblock_width_dim
\bool_new:N \g__wrapblock_active_bool   % a region is running
\bool_new:N \g__wrapblock_pending_bool  % the image is not placed yet
\dim_new:N  \g__wrapblock_bottom_dim    % bottom of the reserved area
\dim_new:N  \g__wrapblock_height_dim    % image height + vsep
\dim_new:N  \g__wrapblock_shift_dim     % room taken by image + hsep
\dim_new:N  \g__wrapblock_left_dim      % left edge of the block
\dim_new:N  \g__wrapblock_width_dim     % width of the block
\dim_new:N  \g__wrapblock_prevdepth_dim % \prevdepth before the paragraph
\int_new:N  \g__wrapblock_ship_int      % pages shipped so far
\int_new:N  \g__wrapblock_origin_ship_int
\tl_new:N   \g__wrapblock_side_tl
\tl_new:N   \l__wrapblock_side_tl
\int_new:N  \g__wrapblock_save_pre_int
\int_new:N  \g__wrapblock_save_post_int
\bool_new:N \g__wrapblock_global_bool
\bool_new:N \l__wrapblock_outer_bool
\bool_set_true:N \l__wrapblock_outer_bool
\int_new:N  \g__wrapblock_save_item_int
\int_new:N  \g__wrapblock_save_beginpar_int
\int_new:N  \g__wrapblock_save_endpar_int
\int_new:N  \l__wrapblock_lines_int
\dim_new:N  \l__wrapblock_base_dim
\dim_new:N  \l__wrapblock_rule_dim
\dim_new:N  \l__wrapblock_i_dim
\dim_new:N  \l__wrapblock_l_dim
\dim_new:N  \l__wrapblock_ni_dim
\dim_new:N  \l__wrapblock_nl_dim
\tl_new:N   \g__wrapblock_parshape_tl  % the shape of the paragraph being set,
\tl_new:N   \g__wrapblock_pen_tl       % and its penalties
\bool_new:N \g__wrapblock_inpar_bool   % a narrowed paragraph is being set
\dim_new:N  \g__wrapblock_avail_dim     % line width left beside the image
\tl_new:N   \g__wrapblock_warned_tl     % input line of the last box warning
\seq_new:N  \g__wrapblock_shapes_seq    % signatures of the shapes set
\int_new:N  \g__wrapblock_narrowed_int  % lines narrowed in the paragraph
\bool_new:N \l__wrapblock_inside_bool   % inside a wrapblock environment
\bool_new:N \l__wrapblock_inbox_bool    % this block sits in a box
\bool_new:N \l__wrapblock_inimage_bool  % inside the image of a block
\bool_new:N \l__wrapblock_alone_bool    % the image is too wide: no wrapping
\tl_new:N   \l__wrapblock_shape_tl
\tl_new:N   \l__wrapblock_pen_tl
```

The variables of a block are global, because a block runs over several paragraphs, and a paragraph often ends in another group than the one the block started in (an item, an environment). They describe:

- *the image*: the material given (`\g__wrapblock_img_tl`), the box built from it

(\g__wrapblock_img_box), its width key, its height plus vsep (\g__wrapblock_height_dim), and the width it takes from the lines, hsep included (\g__wrapblock_shift_dim);

- *the block*: its left edge and its width (\g__wrapblock_left_dim, \g__wrapblock_width_dim), the side of the image, and the bottom of the image area (\g__wrapblock_bottom_dim);
- *the zone*: running or not (\g__wrapblock_active_bool), image placed or not yet (\g__wrapblock_pending_bool), the number of lines narrowed in the current paragraph (\g__wrapblock_narrowed_int), the line width left beside the image (\g__wrapblock_avail_dim), the shapes set so far (\g__wrapblock_shapes_seq, see __wrapblock_unstale:), and the shape of the narrowed paragraph being typeset (\g__wrapblock_parshape_tl, \g__wrapblock_pen_tl, \g__wrapblock_inpar_bool, see __wrapblock_reshape:);
- *the pages*: the number of pages shipped out so far (\g__wrapblock_ship_int), and its value where the zone started (\g__wrapblock_origin_ship_int); when the two differ, the zone has moved to another page;
- *the penalties the block changes*, saved to be put back: \predisplaypenalty, \postdisplaypenalty, \@itempenalty, \@beginparpenalty, \@endparpenalty.

Three local switches say where the code stands:

- \l__wrapblock_outer_bool is true in the text of the block itself. It is false in a box inside the block (a \parbox, a footnote, a float) and in the box where the text is measured: the hooks then leave the paragraphs alone.
- \l__wrapblock_inside_bool is true within the environment.
- \l__wrapblock_inbox_bool is true when the block itself sits in a box.
- \l__wrapblock_inimage_bool is true while the image is typeset, to catch a block given as the image of another; \l__wrapblock_alone_bool is true when the image is too wide to have text beside it.

The remaining local variables are scratch space for the computations.

```
% A deferred image: it did not fit at the bottom of the page, and the text
% is long enough to fill that bottom at full width. It goes to the top of
% the next page, and the lines there are narrowed.
\bool_new:N \g__wrapblock_wait_bool     % deferred, the page has not broken yet
\int_new:N  \g__wrapblock_full_end_int  % last full-width line before the break
\int_new:N  \l__wrapblock_full_int
\int_new:N  \g__wrapblock_def_int       % deferred images so far
\seq_new:N  \g__wrapblock_def_seq       % deferred images waiting for their page
\int_new:N  \g__wrapblock_struct_int    % structure that holds the image
\tl_new:N   \g__wrapblock_counters_tl
\box_new:N  \l__wrapblock_measure_box
\box_new:N  \l__wrapblock_labels_box
\box_new:N  \g__wrapblock_float_box     % a float of the text, while it is measured
\box_new:N  \g__wrapblock_mar_box       % a \marginpar, likewise
\tl_new:N   \l__wrapblock_freelist_tl   % the lists of float boxes of LaTeX,
\tl_new:N   \l__wrapblock_currlist_tl   % kept while the text is measured
\dim_new:N  \g__wrapblock_text_dim      % height of the text at full width
\int_new:N  \g__wrapblock_block_int     % blocks so far, to name them in the .aux
\dim_new:N  \g__wrapblock_goal_dim      % \pagegoal where a deferred block starts
\dim_new:N  \g__wrapblock_short_dim     % room the footnotes will take there
\prop_new:N \g__wrapblock_short_prop    % that room, read from the .aux
\prop_new:N \g__wrapblock_short_new_prop % and as found in this run
\bool_new:N \l__wrapblock_defer_bool
\bool_new:N \l__wrapblock_allow_defer_bool
```

```
\bool_new:N \g__wrapblock_breakpar_bool  % this paragraph ends the page
\seq_new:N \l__wrapblock_done_seq
\dim_new:N  \g__wrapblock_top_dim       % height of the floats at the top of the page
\dim_new:N  \wrapblocklinewidth         % the full line of the block, for the image
\bool_new:N \l__wrapblock_fit_bool      % width=fit
\fp_new:N   \l__wrapblock_fit_min_fp    % the bounds of the search, fractions
\fp_new:N   \l__wrapblock_fit_max_fp    % of the line
\fp_new:N   \l__wrapblock_fit_lo_fp     % the bracket of the search
\fp_new:N   \l__wrapblock_fit_hi_fp
\fp_new:N   \l__wrapblock_fit_mid_fp
\bool_new:N \l__wrapblock_fit_lo_bool   % at lo, the image is the taller
\bool_new:N \l__wrapblock_fit_mid_bool
\dim_new:N  \l__wrapblock_fit_lo_dim    % height of the block at lo, at hi
\dim_new:N  \l__wrapblock_fit_hi_dim
\dim_new:N  \l__wrapblock_fit_w_dim     % width tried
\dim_new:N  \l__wrapblock_fit_col_dim   % width left for the text
\dim_new:N  \l__wrapblock_fit_img_dim   % height of the image at that width
\dim_new:N  \l__wrapblock_fit_txt_dim   % height of the text beside it
\int_new:N  \l__wrapblock_fit_int       % steps of the search
\box_new:N  \g__wrapblock_fit_box
```

These variables serve the deferred image and the measure of the text:

- `\g__wrapblock_wait_bool`: a deferred image is waiting for its page break; `\g__wrapblock_full_end_int`: the number of the last full-width line before that break; `\g__wrapblock_breakpar_bool`: the current paragraph is the one that ends the page;
- `\g__wrapblock_def_seq`, `\g__wrapblock_def_int`: the deferred images committed and not yet drawn, and their numbering; `\g__wrapblock_struct_int`: the structure element that will hold the image in a tagged PDF; `\g__wrapblock_top_dim`: the height of the floats at the top of the page being built;
- `\g__wrapblock_block_int`, `\g__wrapblock_goal_dim`, `\g__wrapblock_short_dim` and the two property lists: the room footnotes take, kept from one run to the next (see the chunk `aux`);
- `\g__wrapblock_text_dim`, `\g__wrapblock_counters_tl`, `\l__wrapblock_measure_box`, `\l__wrapblock_labels_box`, the two float boxes and the two lists of float boxes: the measure of the text (`\__wrapblock_measure:n`).

```
\hook_gput_code:nnn { shipout / after } { wrapblock }
  { \int_gincr:N \g__wrapblock_ship_int }
```

`\g__wrapblock_ship_int` counts the pages shipped out. The page counter would not do: the document can reset it. The count serves two purposes: to notice that the zone has moved to another page (`\__wrapblock_check_ship:`), and to know on which page a deferred image must be drawn (`\__wrapblock_overlay:nnnnnnn`).

11.3 Keys

```
% ------------------------------------------------------------------ keys
\keys_define:nn { wrapblock }
  {
    side  .choices:nn = { l , r } { \tl_set:Nn \l__wrapblock_side_tl {#1} } ,
    side  .initial:n  = r ,
    l     .meta:n     = { side = l } ,
    r     .meta:n     = { side = r } ,
    % natural: the width of the image itself, kept as -\maxdimen, which no
    % length can be; 0pt is a room of no width, like any other length
```

```
    % fit: searched between fit min and fit max (\__wrapblock_fit:n);
    % natural until then, and if the search finds nothing
    width .code:n     =
      {
        \bool_set_false:N \l__wrapblock_fit_bool
        \str_case:nnF {#1}
          {
            { natural } { \dim_set:Nn \l__wrapblock_width_dim { - \c_max_dim } }
            { fit }
              {
                \bool_set_true:N \l__wrapblock_fit_bool
                \dim_set:Nn \l__wrapblock_width_dim { - \c_max_dim }
              }
          }
          { \dim_set:Nn \l__wrapblock_width_dim {#1} }
      } ,
    width .initial:n  = natural ,
    width .value_required:n = true ,
    hsep  .dim_set:N  = \l__wrapblock_hsep_dim ,
    hsep  .initial:n  = 1em ,
    vsep  .dim_set:N  = \l__wrapblock_vsep_dim ,
    vsep  .initial:n  = 0.5\baselineskip ,
    defer .bool_set:N = \l__wrapblock_allow_defer_bool ,
    defer .initial:n  = true ,
    defer .default:n  = true ,
    fit~min .fp_set:N = \l__wrapblock_fit_min_fp ,
    fit~min .initial:n = 0.1 ,
    fit~max .fp_set:N = \l__wrapblock_fit_max_fp ,
    fit~max .initial:n = 0.9 ,
  }
\NewDocumentCommand \wrapblockset { m } { \keys_set:nn { wrapblock } {#1} }
```

The keys are local: `\wrapblockset` sets them for the current group, and the optional argument of the environment for one block. `width=natural` is kept as a width of minus `\maxdimen`, which no length can be; the code reads it as "the width of the image". `width=fit` is kept the same way, with `\l__wrapblock_fit_bool` true: the search (`\__wrapblock_fit:n`) replaces it by a length, or leaves it natural when it finds nothing. The initial values of `hsep` and `vsep` are computed when the package is loaded, with the font and the `\baselineskip` of that moment.

11.4 The search of `width=fit`

```
% width=fit. The block is tried at widths between fit min and fit max,
% fractions of the line: at each, the image is built in a minipage of that
% width, as with the key width, and the text set in the room left beside
% it. The image may get shorter as the block widens (a caption on fewer
% lines) or taller (an image whose width is a fraction of \linewidth);
% the text gets taller. The block is lowest where the two cross: if the
% image is the taller at one bound and the shorter at the other, a
% bisection closes on the crossing, to 2% of the line (6 steps from the
% default bounds). Otherwise there is
% no crossing between the bounds, and the block keeps its natural width.
% #1 is the text.
\cs_new_protected:Npn \__wrapblock_fit:n #1
  {
    \fp_set_eq:NN \l__wrapblock_fit_lo_fp \l__wrapblock_fit_min_fp
    \fp_set_eq:NN \l__wrapblock_fit_hi_fp \l__wrapblock_fit_max_fp
    \__wrapblock_fit_eval:Nn \l__wrapblock_fit_lo_fp {#1}
    \bool_set_eq:NN \l__wrapblock_fit_lo_bool \l__wrapblock_fit_mid_bool
    \dim_set:Nn \l__wrapblock_fit_lo_dim
      { \dim_max:nn { \l__wrapblock_fit_img_dim } { \l__wrapblock_fit_txt_dim } }
    \__wrapblock_fit_eval:Nn \l__wrapblock_fit_hi_fp {#1}
    \dim_set:Nn \l__wrapblock_fit_hi_dim
```

```
      { \dim_max:nn { \l__wrapblock_fit_img_dim } { \l__wrapblock_fit_txt_dim } }
    % The bounds agree. An image that grows with the block may still be the
    % taller in the middle: at the bounds, the text is the taller, beside a
    % small image or in a column that has all but vanished. One more trial,
    % in the middle, which brackets the crossing if it differs.
    \bool_if:NT
      {
        \bool_lazy_or_p:nn
          { \bool_lazy_and_p:nn { \l__wrapblock_fit_lo_bool } { \l__wrapblock_fit_mid_bool } }
          { \bool_lazy_and_p:nn { ! \l__wrapblock_fit_lo_bool } { ! \l__wrapblock_fit_mid_bool } }
      }
      {
        \fp_set:Nn \l__wrapblock_fit_mid_fp
          { ( \l__wrapblock_fit_lo_fp + \l__wrapblock_fit_hi_fp ) / 2 }
        \__wrapblock_fit_eval:Nn \l__wrapblock_fit_mid_fp {#1}
        \bool_if:NT
          {
            \bool_lazy_or_p:nn
              { \bool_lazy_and_p:nn { \l__wrapblock_fit_lo_bool } { ! \l__wrapblock_fit_mid_bool } }
              { \bool_lazy_and_p:nn { ! \l__wrapblock_fit_lo_bool } { \l__wrapblock_fit_mid_bool } }
          }
          {
            \fp_set_eq:NN \l__wrapblock_fit_hi_fp \l__wrapblock_fit_mid_fp
            \dim_set:Nn \l__wrapblock_fit_hi_dim
              { \dim_max:nn { \l__wrapblock_fit_img_dim } { \l__wrapblock_fit_txt_dim } }
          }
      }
    \bool_if:NTF \l__wrapblock_fit_lo_bool
      { \bool_if:NTF \l__wrapblock_fit_mid_bool { \use_i:nn } { \use_ii:nn } }
      { \bool_if:NTF \l__wrapblock_fit_mid_bool { \use_ii:nn } { \use_i:nn } }
      {
        \msg_warning:nne { wrapblock } { fit }
          { \bool_if:NTF \l__wrapblock_fit_lo_bool { taller } { shorter } }
      }
      {
        \int_zero:N \l__wrapblock_fit_int
        \__wrapblock_fit_step:n {#1}
        \dim_set:Nn \l__wrapblock_width_dim
          {
            \fp_to_dim:n
              {
                ( \dim_compare:nNnTF { \l__wrapblock_fit_lo_dim } > { \l__wrapblock_fit_hi_dim }
                    { \l__wrapblock_fit_hi_fp } { \l__wrapblock_fit_lo_fp } )
                * \linewidth
              }
          }
        \msg_info:nnee { wrapblock } { fit-width }
          { \dim_use:N \l__wrapblock_width_dim }
          { \fp_eval:n { round ( \l__wrapblock_width_dim / \linewidth , 3 ) } }
      }
  }
\cs_new_protected:Npn \__wrapblock_fit_step:n #1
  {
    \bool_lazy_and:nnT
      {
        \fp_compare_p:n
          { \l__wrapblock_fit_hi_fp - \l__wrapblock_fit_lo_fp > 0.02 }
      }
      { \int_compare_p:nNn { \l__wrapblock_fit_int } < { 20 } }
      {
        \int_incr:N \l__wrapblock_fit_int
        \fp_set:Nn \l__wrapblock_fit_mid_fp
```

```
          { ( \l__wrapblock_fit_lo_fp + \l__wrapblock_fit_hi_fp ) / 2 }
        \__wrapblock_fit_eval:Nn \l__wrapblock_fit_mid_fp {#1}
        \bool_if:NTF
          {
            \bool_lazy_or_p:nn
              { \bool_lazy_and_p:nn { \l__wrapblock_fit_lo_bool } { \l__wrapblock_fit_mid_bool } }
              { \bool_lazy_and_p:nn { ! \l__wrapblock_fit_lo_bool } { ! \l__wrapblock_fit_mid_bool
      } }
          }
          {
            \fp_set_eq:NN \l__wrapblock_fit_lo_fp \l__wrapblock_fit_mid_fp
            \dim_set:Nn \l__wrapblock_fit_lo_dim
              { \dim_max:nn { \l__wrapblock_fit_img_dim } { \l__wrapblock_fit_txt_dim } }
          }
          {
            \fp_set_eq:NN \l__wrapblock_fit_hi_fp \l__wrapblock_fit_mid_fp
            \dim_set:Nn \l__wrapblock_fit_hi_dim
              { \dim_max:nn { \l__wrapblock_fit_img_dim } { \l__wrapblock_fit_txt_dim } }
          }
        \__wrapblock_fit_step:n {#1}
      }
  }
% The block at the width #1 (a fraction of the line), its text #2: the
% heights of the image and of the text beside it, and whether the image is
% the taller (\l__wrapblock_fit_mid_bool). The image is built as it will
% be, its counters put back; with less than 3em left, the text is taken
% as endless.
\cs_new_protected:Npn \__wrapblock_fit_eval:Nn #1#2
  {
    \dim_set:Nn \l__wrapblock_fit_w_dim { \fp_to_dim:n { #1 * \linewidth } }
    \__wrapblock_counters_save:
    \group_begin:
      \cs_if_exist_use:NT \tag_suspend:n { { wrapblock } }
      % a trial: an image wider than the block, a caption badly broken,
      % are not reported. The minipage and a caption run \sloppy (through
      % \@parboxrestore), which sets \hfuzz again: it is made quiet too.
      \__wrapblock_quiet:
      \tl_put_right:Nn \sloppy { \__wrapblock_quiet: }
      \hbox_gset:Nn \g__wrapblock_fit_box
        { \__wrapblock_image:nv { \l__wrapblock_fit_w_dim } { g__wrapblock_img_tl } }
      \cs_if_exist_use:NT \tag_resume:n { { wrapblock } }
    \group_end:
    \tl_use:N \g__wrapblock_counters_tl
    \dim_set:Nn \l__wrapblock_fit_img_dim
      { \box_ht:N \g__wrapblock_fit_box + \box_dp:N \g__wrapblock_fit_box }
    \dim_set:Nn \l__wrapblock_fit_col_dim
      { \linewidth - \box_wd:N \g__wrapblock_fit_box - \l__wrapblock_hsep_dim }
    \dim_compare:nNnTF { \l__wrapblock_fit_col_dim } < { 3em }
      { \dim_set_eq:NN \l__wrapblock_fit_txt_dim \c_max_dim }
      {
        \__wrapblock_measure:nn { \l__wrapblock_fit_col_dim } {#2}
        \dim_set_eq:NN \l__wrapblock_fit_txt_dim \g__wrapblock_text_dim
      }
    \bool_set:Nn \l__wrapblock_fit_mid_bool
      { \dim_compare_p:nNn { \l__wrapblock_fit_img_dim } > { \l__wrapblock_fit_txt_dim } }
  }
\cs_new_protected:Npn \__wrapblock_quiet:
  {
    \int_set:Nn \hbadness { 10000 }
    \int_set:Nn \vbadness { 10000 }
    \dim_set_eq:NN \hfuzz \c_max_dim
    \dim_set_eq:NN \vfuzz \c_max_dim
  }
\msg_new:nnn { wrapblock } { fit }
```

```
  { The~image~of~the~block~that~ends~on~input~line~\msg_line_number:\
    is~#1~than~its~text~at~every~width~between~fit~min~and~fit~max:~
    width=fit~finds~no~crossing,~and~the~image~keeps~its~natural~width. }
\msg_new:nnn { wrapblock } { fit-width }
  { The~block~that~ends~on~input~line~\msg_line_number:\ is~#1~wide,~#2~of~
    the~line~(width=fit). }
```

`\__wrapblock_fit_eval:Nn` tries one width, a fraction of the line held in an fp variable: it builds the image in its `minipage` of that width, counters put back as in the environment, and measures the text in the room left, `\linewidth` less the width of the image and `hsep`, with `\__wrapblock_measure:nn`. A room under 3 em counts as an endless text. `\l__wrapblock_fit_mid_bool` says whether the image is the taller.

The trial must not fill the log with the `Overfull` boxes of an image too wide for a narrow block. Setting `\hfuzz` and `\hbadness` is not enough: the `minipage`, and a caption, run `\@parboxrestore`, whose `\sloppy` sets `\hfuzz` again. `\sloppy` is therefore extended, in the group of the trial, with `\__wrapblock_quiet:`.

`\__wrapblock_fit:n` tries the two bounds, and the middle if they agree. A bracket where the image is the taller at one end and the shorter at the other is closed by `\__wrapblock_fit_step:n`, which halves it while it is wider than 0.02 of the line. Of the two ends, the one where the block, the taller of image and text, is the lower gives the width.

11.5 The environment

```
% ------------------------------------------------------------------ user
\NewDocumentEnvironment { wrapblock } { O{} +m +b }
  {
    \par
    \bool_if:NT \l__wrapblock_inimage_bool { \msg_error:nn { wrapblock } { nested } }
    \int_gincr:N \g__wrapblock_block_int
    \keys_set:nn { wrapblock } {#1}
    \tl_gset:Nn \g__wrapblock_img_tl {#2}
    \dim_gset_eq:NN \wrapblocklinewidth \linewidth
    \bool_if:NT \l__wrapblock_fit_bool { \__wrapblock_fit:n {#3} }
    \dim_gset_eq:NN \g__wrapblock_img_width_dim \l__wrapblock_width_dim
    % measured here, with tagging suspended; built for good where it is
    % placed, inside the first paragraph (see \__wrapblock_build:)
    % the counters it steps (an equation, a caption) are put back
    \cs_if_exist_use:NT \tag_suspend:n { { wrapblock } }
    \__wrapblock_counters_save:
    \__wrapblock_build:
    \tl_use:N \g__wrapblock_counters_tl
    \cs_if_exist_use:NT \tag_resume:n { { wrapblock } }
    \dim_gset:Nn \g__wrapblock_height_dim
      { \box_ht:N \g__wrapblock_img_box + \box_dp:N \g__wrapblock_img_box
        + \l__wrapblock_vsep_dim }
    \dim_gset:Nn \g__wrapblock_shift_dim
      { \box_wd:N \g__wrapblock_img_box + \l__wrapblock_hsep_dim }
    \dim_compare:nNnT { \g__wrapblock_height_dim } > { \textheight }
      { \msg_warning:nn { wrapblock } { tall } }
    % less than 3em left beside the image: no wrapping, the image is set
    % on a line of its own, above the text
    \dim_compare:nNnTF { \linewidth - \g__wrapblock_shift_dim } < { 3em }
      {
        \msg_warning:nne { wrapblock } { wide }
          { \dim_eval:n { \linewidth - \g__wrapblock_shift_dim } }
        \bool_set_true:N \l__wrapblock_alone_bool
        \noindent \__wrapblock_build: \box_use:N \g__wrapblock_img_box
        \par \nobreak
      }
      {
```

```
        \mode_if_inner:TF
          {
            \bool_set_true:N \l__wrapblock_inbox_bool
            \__wrapblock_inbox_begin:
          }
          { \__wrapblock_begin:n {#3} }
      }
    #3
  }
  {
    \bool_if:NTF \l__wrapblock_alone_bool
      { \par }
      {
        \bool_if:NTF \l__wrapblock_inbox_bool
          { \__wrapblock_inbox_end: }
          { \__wrapblock_end: }
      }
  }
\msg_new:nnn { wrapblock } { wide }
  { The~image~of~the~block~that~ends~on~input~line~\msg_line_number:\
    leaves~only~#1~for~the~text~beside~it:~it~is~set~above~the~text,~
    which~is~not~wrapped. }
\msg_new:nnn { wrapblock } { tall }
  { The~image~of~the~block~that~ends~on~input~line~\msg_line_number:\
    is~taller~than~the~page. }
```

The environment takes its text as an argument (`+b`), because the text may have to be measured before it is typeset (`\__wrapblock_measure:n`). This is what forbids verbatim material in it.

In order, the environment:

1. ends the current paragraph, numbers the block and sets its keys; keeps the line in `\wrapblocklinewidth`, and searches the width with `width=fit` (`\__wrapblock_fit:n`);
2. builds the image once, with tagging suspended, only to measure it: `\g__wrapblock_height_dim` is its height and depth plus `vsep`, `\g__wrapblock_shift_dim` its width plus `hsep`. The LaTeX counters are kept before and put back after (`\__wrapblock_counters_save:`), so that a numbered equation or a caption in the image is not counted twice. The image is built for good later, inside the first paragraph;
3. warns about an image taller than the page. An image that leaves less than 3 em beside it is not wrapped at all: it is typeset on a line of its own, the text follows at full width, and nothing of the rest of the package is used (`\l__wrapblock_alone_bool`);
4. tells a block in a box from a block on the page. After `\par`, TeX is in internal vertical mode in a `minipage` or a `\parbox`, and in outer vertical mode on the page: `\mode_if_inner:TF` tests it. On the page, `\__wrapblock_begin:n` prepares the zone;
5. typesets the text, as it is.

```
\cs_new_protected:Npn \__wrapblock_begin:n #1
  {
    \bool_if:NT \g__wrapblock_active_bool { \msg_error:nn { wrapblock } { nested } }
    \dim_gset_eq:NN \g__wrapblock_left_dim  \@totalleftmargin
    \dim_gset_eq:NN \g__wrapblock_width_dim \linewidth
    \bool_set_false:N \l__wrapblock_defer_bool
    \dim_compare:nNnF { \pagegoal } = { \c_max_dim }
      {
        \dim_compare:nNnT { \__wrapblock_room: }
          < { \g__wrapblock_height_dim + \baselineskip }
          { \__wrapblock_decide:n {#1} }
      }
```

```
    \tl_gset_eq:NN \g__wrapblock_side_tl \l__wrapblock_side_tl
    \int_gset_eq:NN \g__wrapblock_save_pre_int  \predisplaypenalty
    \int_gset_eq:NN \g__wrapblock_save_post_int \postdisplaypenalty
    \int_gset_eq:NN \g__wrapblock_save_item_int     \@itempenalty
    \int_gset_eq:NN \g__wrapblock_save_beginpar_int \@beginparpenalty
    \int_gset_eq:NN \g__wrapblock_save_endpar_int   \@endparpenalty
    \tl_put_left:Nn \@arrayparboxrestore
      { \__wrapblock_box_check: \bool_set_false:N \l__wrapblock_outer_bool }
    \bool_set_true:N \l__wrapblock_inside_bool
    \seq_gclear:N \g__wrapblock_shapes_seq
    \bool_if:NTF \l__wrapblock_defer_bool
      {
        \bool_gset_false:N \g__wrapblock_pending_bool
        \bool_gset_true:N \g__wrapblock_wait_bool
        \int_gset_eq:NN \g__wrapblock_origin_ship_int \g__wrapblock_ship_int
        \dim_gset:Nn \g__wrapblock_bottom_dim
          { \topskip - \box_ht:N \strutbox + \g__wrapblock_height_dim }
        \dim_gset_eq:NN \g__wrapblock_goal_dim \pagegoal
        \prop_get:NeNTF \g__wrapblock_short_prop { \int_use:N \g__wrapblock_block_int } \l_tmpa_tl
          { \dim_gset:Nn \g__wrapblock_short_dim { \l_tmpa_tl } }
          { \dim_gzero:N \g__wrapblock_short_dim }
        \__wrapblock_placeholder:
      }
      {
        \bool_gset_true:N \g__wrapblock_pending_bool
        \bool_gset_false:N \g__wrapblock_wait_bool
      }
    \bool_gset_true:N \g__wrapblock_active_bool
  }
```

`\__wrapblock_begin:n` receives the text only to measure it if needed. In order:

1. A block in the text of another block is an error: the state of a block is global, and two blocks would share it.
2. The left edge and the width of the block are read from `\@totalleftmargin` and `\linewidth`: in a list, they are those of the current item.
3. The room left on the page is compared with what the image needs: its height and one line more. On an empty page there is always room (`\pagegoal` is `\maxdimen` until something is put on the page). When the room is too small, `\__wrapblock_decide:n` finds a way out (section 11.6).
4. The penalties are saved, to be put back at the end of the zone.
5. `\__wrapblock_box_check:` is put at the front of `\@arrayparboxrestore`, which LaTeX runs at the start of every `\parbox`, `minipage`, `p` cell, footnote and float. It warns about a box too wide for the room beside the image, then turns `\l__wrapblock_outer_bool` off inside the box (section 11.12). The change is local to the environment.
6. The state of the image is set. Normally the image is pending: the first paragraph will place it. When `\__wrapblock_decide:n` has deferred it, the zone is prepared on the next page instead: its bottom is computed from `\topskip`, the first baseline of a page; `\g__wrapblock_goal_dim` keeps the `\pagegoal` of this point and `\g__wrapblock_short_dim` the room footnotes took in the previous run (see the chunk `aux`); and an empty structure element is made for the tagged PDF (`\__wrapblock_placeholder:`).

```
\cs_new_protected:Npn \__wrapblock_end:
  {
    \par
    % the measure was wrong: the text ended on the page it started on
```

```
    \bool_if:NT \g__wrapblock_wait_bool
      {
        \msg_warning:nn { wrapblock } { alone }
        \bool_gset_false:N \g__wrapblock_wait_bool
        \bool_gset_true:N \g__wrapblock_pending_bool
        \dim_compare:nNnT { \__wrapblock_room: }
          < { \g__wrapblock_height_dim + \baselineskip }
          { \__wrapblock_need_space: }
      }
    \bool_if:NT \g__wrapblock_pending_bool { \noindent \strut \par }
    \bool_if:NT \g__wrapblock_active_bool
      {
        \__wrapblock_check_ship:
        % The room down to the bottom of the image is a kern, and only the
        % space below the image (vsep) a skip. \addpenalty (a heading, a
        % list) takes the last skip back and puts its penalty before it:
        % were the whole room a skip, the page could break there, and the
        % image run out of the page. \addvspace still takes the larger of
        % its space and vsep, as after a line of text. The penalties keep
        % the kern and the skip from being breakpoints.
        \dim_compare:nNnT { \pagetotal + \pagedepth } < { \g__wrapblock_bottom_dim }
          {
            \dim_set:Nn \l_tmpa_dim
              { \g__wrapblock_bottom_dim - \pagetotal - \pagedepth }
            \dim_set:Nn \l_tmpb_dim
              { \dim_min:nn { \l__wrapblock_vsep_dim } { \l_tmpa_dim } }
            \nobreak
            \tex_kern:D \dim_eval:n { \l_tmpa_dim - \l_tmpb_dim }
            \nobreak
            \skip_vertical:N \l_tmpb_dim
          }
        \__wrapblock_stop:
      }
  }
```

At the end of the text, `\__wrapblock_end:` deals with three cases, in order:

1. *A deferred image is still waiting.* The text was measured long enough to reach the next page, but it ended on this one. This should not happen; a tall line can cause it. The image goes back to the pending state, and the page is broken first if the image does not fit after the text.

2. *The image is still pending*, because the block is empty or because of the case above. A paragraph made of one empty line is typeset to carry it.

3. *The text ended above the bottom of the image.* Vertical space is added down to it. The next text thus starts below the image: this is what closes the block. The room down to the bottom of the image is a kern, and only the space below the image, `vsep`, is a skip; `\nobreak` comes before each. Were the whole room a skip, `\addpenalty`, which a heading and a list use, would take it back and put its penalty before it: the page could then break there, the room would go to the next page and be discarded, and the image would run out of the page, the more so under `\flushbottom`, which stretches a page that ends too early. With the kern, only `vsep` can go; and `\addvspace` still takes the larger of its own space and `vsep`, as after the last line of a paragraph.

`\__wrapblock_stop:` then ends the zone and puts the penalties back.

11.6 Is there room for the image?

```
% The room left on the page. Right after a heading, the space below it is
```

```
% not on the page yet: \pagetotal counts it only when the next paragraph
% starts. It is the last item of the list, and \lastskip gives it. When
% the page itself ends with glue, it is subtracted twice: the error only
% makes the image less likely to fit.
\cs_new:Npn \__wrapblock_room: { ( \pagegoal - \pagetotal - \lastskip ) }
```

The room left on the page is `\pagegoal` minus `\pagetotal`, with one correction right after a heading. A heading puts a space below itself, but the page builder only moves that space to the page when the next box comes: `\pagetotal` does not count it yet. That space is the last item of the vertical list, and `\lastskip` gives it.

When the block does not follow a heading and the page ends with some other glue, that glue is subtracted although `\pagetotal` already counts it. The room is then found a little smaller than it is. The error is on the safe side: the block may move when its image would just have fitted, but the image never overflows.

```
% The image does not fit at the bottom of the page. With the text long
% enough to fill that bottom at full width and run on to the next page,
% the image is deferred to the top of that page. With a shorter text, the
% image would end up alone: the whole block moves to the next page.
% Columns (twocolumn, multicols) are not handled: the block moves.
\cs_new_protected:Npn \__wrapblock_decide:n #1
  {
    \bool_set_false:N \l__wrapblock_defer_bool
    \bool_lazy_or:nnTF
      { \legacy_if_p:n { @twocolumn } }
      {
        \bool_lazy_and_p:nn
          { \cs_if_exist_p:N \col@number }
          { \int_compare_p:nNn { \col@number } > { 1 } }
      }
      { \__wrapblock_need_space: }
      {
        % defer=false: the body is not measured, the whole block moves
        \bool_lazy_or:nnTF
          { ! \l__wrapblock_allow_defer_bool }
          { \dim_compare_p:nNn { \__wrapblock_room: } < { 2 \baselineskip } }
          { \__wrapblock_need_space: }
          {
            \__wrapblock_measure:n {#1}
            \dim_compare:nNnTF
              { \g__wrapblock_text_dim } < { \__wrapblock_room: + 2 \baselineskip }
              {
                \msg_warning:nn { wrapblock } { short }
                \__wrapblock_need_space:
              }
              { \bool_set_true:N \l__wrapblock_defer_bool }
          }
      }
  }
\msg_new:nnn { wrapblock } { short }
  { The~image~of~the~block~that~ends~on~input~line~\msg_line_number:\ does~not~fit~
    at~the~bottom~of~the~page,~and~its~text~is~too~short~to~run~on~to~
    the~next~page:~the~whole~block~moves~to~the~next~page,~and~leaves~
    room~at~the~bottom~of~this~one. }
```

`\__wrapblock_decide:n` is called when the image does not fit. It ends in one of two states: `\l__wrapblock_defer_bool` is true, and the block stays; or `\__wrapblock_need_space:` has broken the page, and the block is at the top of the next one. The tests come in this order:

1. *Two columns*, or a `multicols` environment with more than one column: the block moves. A deferred image is drawn at the top of the page, and the package does not know the column.
2. *`defer=false`, or less than two lines of room*: the block moves, and the text is not measured.

3. *Otherwise the text is measured* at full width. A deferred image needs a text that fills the room left on this page, and two lines of the next page. A shorter text would leave the image alone on the next page: the block moves, with the warning `short`. A longer one stays, and the image is deferred.

```
% Not enough room left for the image, or the page already holds more than
% it can (TeX cuts it only at the next legal break). An invisible rule as
% tall as the image makes the page overflow, and a breakpoint after it,
% which TeX cannot take on a full page, makes TeX cut now at its best
% earlier break: before a heading, which forbids a break after itself,
% or before the previous paragraph. The rule is taller than the image by
% the shrink of the page: otherwise, when the image is only a little too
% tall, the page still fits by shrinking its glue, TeX takes the
% breakpoint after the rule, and a heading is left alone at the bottom.
% A negative skip then gives the room back, and \pagetotal measures the
% page the block starts on.
\cs_new_protected:Npn \__wrapblock_need_space:
  {
    % An image taller than the page fits nowhere: the rule below would
    % leave an empty page. A plain page break, and the image overflows.
    \dim_compare:nNnTF
      { \g__wrapblock_height_dim + \baselineskip } > { \tex_vsize:D }
      { \newpage }
      { \__wrapblock_need_space_aux: }
  }
\cs_new_protected:Npn \__wrapblock_need_space_aux:
  {
    \dim_set_eq:NN \l__wrapblock_base_dim \prevdepth
    \legacy_if:nF { @nobreak } { \tex_penalty:D 0 ~ }
    \dim_set:Nn \l__wrapblock_rule_dim
      { \g__wrapblock_height_dim + \baselineskip + \tex_pageshrink:D + 1sp }
    \tex_hrule:D height \l__wrapblock_rule_dim depth 0pt width 0pt \scan_stop:
    \tex_penalty:D 9999 ~
    % in a group: the primitive \par clears \parshape, which a list needs
    \group_begin: \tex_par:D \group_end:
    \tex_penalty:D 10000 ~
    \skip_vertical:n { - \l__wrapblock_rule_dim }
    % An \item, a list or a heading that starts the block calls \addvspace,
    % which would cancel a negative skip it finds last: hide it.
    \tex_penalty:D 10000 ~
    \tex_prevdepth:D \l__wrapblock_base_dim
  }
```

`\__wrapblock_need_space:` moves the block to the next page. An image taller than the page is a case apart: the block gets a plain `\newpage`, since the rule described below would itself not fit on the next page, and would leave it empty. In every other case, `\__wrapblock_need_space_aux:` does the work, and `\newpage` would not do: a block that follows a heading must take the heading with it. The package lets the page builder of TEX choose the break, since it knows the best breakpoint before this point. In order:

1. `\prevdepth` is kept, to be put back at the end.

2. A `\penalty` 0 offers a breakpoint just before the block; not after a heading (`@nobreak` is then true), which must stay with the block.

3. An invisible rule is added, too tall for the page: as tall as the image area plus a line, plus the shrink of the page (`\pageshrink`) and 1 sp.

4. A `\penalty` 9999 offers a breakpoint after the rule, and the primitive `\par`, in vertical mode, runs the page builder. The `\par` is in a group, because it clears `\parshape`, which carries the indentation of a list. The page cannot hold the rule, so TEX breaks at the best breakpoint it

has seen: before the block, or before the heading, since a heading forbids a break after itself. The rule goes to the next page.

5. There, a negative skip gives the height of the rule back, and `\prevdepth` is put back: `\pagetotal` and the interline glue are as if the rule had never been there. The skip sits between two `\penalty` 10000: the first keeps a page break away from it; the second hides it from `\addvspace`, which an `\item`, a list or a heading at the start of the block calls, and which cancels a negative skip it finds last.

The rule must be taller than the room by the shrink of the page. Otherwise, when the image is only a little too tall, the page could still hold the rule by shrinking its glue: TEX would break after the rule, and leave the heading alone at the bottom of the page.

```
% The text of the block, set at full width in a box that is thrown away.
% Counters are put back; labels, footnotes and index entries vanish with
% the box. A block that comes first in an \item sets its label in the box too: the
% label and the state of the list are put back as well.
\cs_new_protected:Npn \__wrapblock_counters_save:
  {
    \tl_gclear:N \g__wrapblock_counters_tl
    \group_begin:
      \cs_set:Npn \@elt ##1
        {
          \tl_gput_right:Ne \g__wrapblock_counters_tl
            { \int_gset:Nn \exp_not:c { c@##1 } { \int_use:c { c@##1 } } }
        }
      \cl@@ckpt
    \group_end:
  }
\cs_new_protected:Npn \__wrapblock_measure:n #1
  { \__wrapblock_measure:nn { \linewidth } {#1} }
% #1 the width of the text, #2 the text
\cs_new_protected:Npn \__wrapblock_measure:nn #1#2
  {
    \__wrapblock_counters_save:
    \hbox_set:Nn \l__wrapblock_labels_box { \box_use:N \@labels }
    \hbox_gset:Nn \@labels { \box_use:N \l__wrapblock_labels_box }
    \clist_map_inline:nn { @inlabel , @newlist , @noparitem , @minipage , @nobreak }
      {
        \tl_gput_right:Ne \g__wrapblock_counters_tl
          { \legacy_if:nTF {##1} { \exp_not:c { ##1true } } { \exp_not:c { ##1false } } }
      }
    \tl_set_eq:NN \l__wrapblock_freelist_tl \@freelist
    \tl_set_eq:NN \l__wrapblock_currlist_tl \@currlist
    \group_begin:
      \cs_if_exist_use:NT \tag_suspend:n { { wrapblock } }
      \vbox_set:Nn \l__wrapblock_measure_box
        {
          \bool_set_false:N \l__wrapblock_outer_bool
          % A float or a \marginpar cannot leave a box. Between two
          % paragraphs, LaTeX sees the box and gives an error: here they are
          % set in boxes of the package, without the error. Inside a
          % paragraph, LaTeX does not see it and takes a float box: the
          % lists of float boxes are put back after the measure.
          \cs_set_eq:NN \@parmoderr \prg_do_nothing:
          \cs_set_eq:NN \@currbox \g__wrapblock_float_box
          \cs_set_eq:NN \@marbox  \g__wrapblock_mar_box
          \dim_set:Nn \hsize {#1}
          \dim_set_eq:NN \linewidth \hsize
          \dim_zero:N \@totalleftmargin
          \tex_parshape:D 0 ~
          \int_set:Nn \hbadness { 10000 }
          \int_set:Nn \vbadness { 10000 }
```

```
          \dim_set_eq:NN \hfuzz \c_max_dim
          #2
          \par
        }
      \dim_gset:Nn \g__wrapblock_text_dim
        { \box_ht:N \l__wrapblock_measure_box + \box_dp:N \l__wrapblock_measure_box }
      \cs_if_exist_use:NT \tag_resume:n { { wrapblock } }
    \group_end:
    \tl_use:N \g__wrapblock_counters_tl
    \hbox_gset:Nn \@labels { \box_use:N \l__wrapblock_labels_box }
    \tl_gset_eq:NN \@freelist \l__wrapblock_freelist_tl
    \tl_gset_eq:NN \@currlist \l__wrapblock_currlist_tl
  }
```

The measure typesets the text at full width (`\__wrapblock_measure:n`), or at the width of a trial of `width=fit` (`\__wrapblock_measure:nn`), in a `\vbox` that is thrown away, and keeps its height and depth in `\g__wrapblock_text_dim`. Whatever the text writes to a file without `\immediate` (labels, table of contents, index) and its footnotes vanish with the box. Its other global effects are undone by hand:

- *The LaTeX counters*, by `\__wrapblock_counters_save:`. `\cl@@ckpt` lists them all, as `\@elt{`⟨*name*⟩`}`. `\@elt` is redefined so that running the list appends to `\g__wrapblock_counters_tl` one command per counter, which sets it back to its present value.
- *The label of an item.* `\@labels` is the box where `\item` leaves its label for the next paragraph. A block that comes first in an item would use the label in the measure, and leave none for the text. The box is copied, and set again from the copy after the measure.
- *The switches of lists and headings* (`@inlabel`, `@newlist`, `@noparitem`, `@minipage`, `@nobreak`), which the first paragraph of the text turns off. They are appended to `\g__wrapblock_counters_tl` as well. Without this, `@nobreak` would be left false after a heading, and the page could break between the heading and the block.
- *Floats and `\marginpar`.* They cannot leave a box. Between two paragraphs, LaTeX notices the box and gives the error "Not in outer par mode": the error is turned off, and the float is set in a box of the package. Inside a paragraph, LaTeX does not notice it and takes one of its float boxes, which would end as "Float(s) lost": its two lists of float boxes, `\@freelist` and `\@currlist`, are put back after the measure.

In the box, `\l__wrapblock_outer_bool` is false, so that the hooks of the package leave the text alone. The width is `\linewidth`, without the indentation of a list (`\@totalleftmargin` and `\parshape` are cleared). Overfull and underfull boxes are not reported, since the box is never seen. Tagging is suspended, so that nothing of the text goes into the structure tree twice.

What is not undone is done twice: an `\immediate\write`, a `\gdef`, a TeX or expl3 counter stepped globally (section 2).

11.7 Each paragraph of the block

```
% ------------------------------------------------------------- hooks
\hook_gput_code:nnn { para / before } { wrapblock }
  {
    \bool_if:NT \g__wrapblock_active_bool
      { \bool_if:NT \l__wrapblock_outer_bool
          { \dim_gset:Nn \g__wrapblock_prevdepth_dim { \prevdepth } } }
  }

\hook_gput_code:nnn { para / begin } { wrapblock }
```

```
  {
    \bool_if:NT \l__wrapblock_outer_bool { \__wrapblock_para_begin: }
  }
```

The two hooks act only while a zone runs, and only in the text of the block itself (`\l__wrapblock_outer_bool`).

- `para/before` runs in vertical mode. LaTeX has already added `\parskip` and run the page builder. The hook keeps `\prevdepth`, the depth of the line before the paragraph.
- `para/begin` runs in horizontal mode, before the first character of the paragraph, and before LaTeX puts the indentation box back. The `\parshape` of the paragraph can still be set there, and the image is placed there, at the very start of the first line.

```
% ------------------------------------------------------------------ helpers
% The messages give the line where the block ends: its text is read as an
% argument, and TeX is on the line of \end{wrapblock} when the code runs.
% Baseline of the first line of the paragraph being started.
\cs_new_protected:Npn \__wrapblock_first_baseline:N #1
  {
    \dim_compare:nNnTF { \pagegoal } = { \c_max_dim }
      { \dim_set:Nn #1 { \topskip } }     % empty page
      {
        % the interline glue TeX will put: \baselineskip less the depth
        % before and the height of the line, or \lineskip when that leaves
        % less than \lineskiplimit (a deep box before: a tabular, a minipage)
        \dim_compare:nNnTF { \g__wrapblock_prevdepth_dim } > { -1000pt }
          {
            \dim_set:Nn #1
              { \baselineskip - \g__wrapblock_prevdepth_dim - \box_ht:N \strutbox }
            \dim_compare:nNnT {#1} < { \lineskiplimit }
              { \dim_set:Nn #1 { \lineskip } }
            \dim_set:Nn #1 { \pagetotal + #1 + \box_ht:N \strutbox }
          }
          { \dim_set:Nn #1 { \pagetotal + \box_ht:N \strutbox } }
      }
  }
```

This function gives the baseline of the first line of the paragraph, in page coordinates. There are three cases:

- *An empty page*: the first baseline is at `\topskip`.
- *The usual case.* When `para/begin` runs, the page builder has taken the `\parskip` glue, and a glue on the page adds the depth of the line before it to `\pagetotal`. The baseline of that line is therefore at `\pagetotal` minus its depth, kept by `para/before`; the new baseline is one `\baselineskip` lower. Unless the line before is deep (a `tabular` centred on its axis, a `minipage`): when `\baselineskip` less that depth and the height of the new line, taken as a strut, leaves less than `\lineskiplimit`, TeX puts `\lineskip` between them, and the new line sits `\lineskip` below `\pagetotal`, its baseline a strut height lower.
- *No interline glue* (`\prevdepth` is $-1000\,\mathrm{pt}$ or less, after a rule or `\nointerlineskip`): the line sits right below `\pagetotal`, and its baseline is taken a strut height lower.

```
\cs_new_protected:Npn \__wrapblock_para_begin:
  {
    \bool_if:NT \g__wrapblock_active_bool
      {
        \__wrapblock_first_baseline:N \l__wrapblock_base_dim
        \bool_if:NTF \g__wrapblock_wait_bool
          { \__wrapblock_defer_para:N \prg_do_nothing: }
```

```
          { \__wrapblock_para_begin_aux: }
      }
  }
\cs_new_protected:Npn \__wrapblock_para_begin_aux:
  {
      \bool_if:NTF \g__wrapblock_pending_bool
        {
          \dim_gset:Nn \g__wrapblock_bottom_dim
            { \l__wrapblock_base_dim - \box_ht:N \strutbox + \g__wrapblock_height_dim }
          \int_gset_eq:NN \g__wrapblock_origin_ship_int \g__wrapblock_ship_int
        }
        { \__wrapblock_check_ship: }
      \__wrapblock_shape:nn { 0 } { \l__wrapblock_base_dim }
      \int_compare:nNnTF { \l__wrapblock_lines_int } > { 0 }
        {
          \int_set:Nn \predisplaypenalty  { 10000 }
          \int_set:Nn \postdisplaypenalty { 10000 }
          \bool_if:NT \g__wrapblock_pending_bool
            {
              \__wrapblock_labels_back:
              \__wrapblock_place_image:
            }
        }
        % past the image: the shape is the one LaTeX gave, keep it
        { \__wrapblock_stop: }
  }
```

`\__wrapblock_para_begin:` computes the first baseline, then sends a paragraph of a waiting deferred image to `\__wrapblock_defer_para:N` (section 11.11), and the others to `\__wrapblock_para_begin_aux:`. There:

1. The first paragraph of the zone, the one that finds the image pending, fixes the bottom of the image area: the top of its first line (its baseline minus the height of a strut) plus `\g__wrapblock_height_dim`. The following paragraphs only check that the zone has not moved to another page.

2. The shape is computed by `\__wrapblock_shape:nn`.

3. If some lines run beside the image, the penalties around displayed equations are set to 10 000, and the first paragraph places the image. If none does, the paragraph starts below the image: the zone is over.

```
% Set \parshape so that the lines from index #1 (0 = first line of the
% paragraph) whose top lies above the bottom of the image are narrowed.
% #2 is the baseline of line #1.
\cs_new_protected:Npn \__wrapblock_shape:nn { \__wrapblock_shape:Nnn \prg_do_nothing: }
% #1 is \prg_do_nothing: or \tex_global:D: after a display the paragraph
% may sit inside the group of an environment (amsmath), which would undo
% a local \parshape.
\cs_new_protected:Npn \__wrapblock_shape:Nnn #1#2#3
  { \__wrapblock_shape:Nnnn #1 {#2} { 0 } {#3} }
% With a deferred image, #3 lines keep their full width first, the last of
% them ends the page, and the narrowed lines start at the top of the next
% one; #4 is then the baseline of the first of them on that page.
\cs_new_protected:Npn \__wrapblock_shape:Nnnn #1#2#3#4
  {
    \int_set:Nn \l__wrapblock_full_int {#3}
    \int_set:Nn \l__wrapblock_lines_int
      {
        \fp_eval:n
          {
            ceil ( ( \dim_to_fp:n { \g__wrapblock_bottom_dim - #4 + \box_ht:N \strutbox } )
```

```
                  / \dim_to_fp:n { \baselineskip } )
        }
    }
  \int_compare:nNnF { \l__wrapblock_lines_int } > { 0 }
    { \int_set:Nn \l__wrapblock_lines_int { 0 } }
  % base line: what LaTeX gave this paragraph (lists set \parshape)
  \int_compare:nNnTF { \parshape } > { 0 }
    {
      \dim_set:Nn \l__wrapblock_i_dim { \parshapeindent \parshape }
      \dim_set:Nn \l__wrapblock_l_dim { \parshapelength \parshape }
    }
    {
      \dim_zero:N \l__wrapblock_i_dim
      \dim_set_eq:NN \l__wrapblock_l_dim \hsize
    }
  \str_if_eq:VnTF \g__wrapblock_side_tl { l }
    {
      \dim_set:Nn \l__wrapblock_ni_dim { \l__wrapblock_i_dim + \g__wrapblock_shift_dim }
      \dim_set:Nn \l__wrapblock_nl_dim { \l__wrapblock_l_dim - \g__wrapblock_shift_dim }
    }
    {
      \dim_set_eq:NN \l__wrapblock_ni_dim \l__wrapblock_i_dim
      \dim_set:Nn \l__wrapblock_nl_dim
        {
          \dim_min:nn { \l__wrapblock_l_dim }
            { \g__wrapblock_left_dim + \g__wrapblock_width_dim
              - \g__wrapblock_shift_dim - \l__wrapblock_i_dim }
        }
    }
  \dim_gset_eq:NN \g__wrapblock_avail_dim \l__wrapblock_nl_dim
  \tl_clear:N \l__wrapblock_shape_tl
  \tl_clear:N \l__wrapblock_pen_tl
  \int_step_inline:nn { #2 }
    {
      \tl_put_right:Nx \l__wrapblock_shape_tl
        { \dim_use:N \l__wrapblock_ni_dim \c_space_tl \dim_use:N \l__wrapblock_nl_dim \
c_space_tl }
      \tl_put_right:Nn \l__wrapblock_pen_tl { 10000 ~ }
    }
  \int_step_inline:nn { \l__wrapblock_full_int }
    {
      \tl_put_right:Nx \l__wrapblock_shape_tl
        { \dim_use:N \l__wrapblock_i_dim \c_space_tl \dim_use:N \l__wrapblock_l_dim \c_space_tl
}
      % the page breaks after the last full-width line (-10000, as \newpage:
      % LaTeX keeps lower values for its floats)
      \tl_put_right:Nx \l__wrapblock_pen_tl
        {
          \int_compare:nNnTF {##1} = { \l__wrapblock_full_int }
            { -10000 } { \int_use:N \interlinepenalty } ~
        }
    }
  \int_step_inline:nn { \l__wrapblock_lines_int }
    {
      \tl_put_right:Nx \l__wrapblock_shape_tl
        { \dim_use:N \l__wrapblock_ni_dim \c_space_tl \dim_use:N \l__wrapblock_nl_dim \
c_space_tl }
      \tl_put_right:Nn \l__wrapblock_pen_tl { 10000 ~ }
    }
  \int_set:Nn \l__wrapblock_lines_int
    { #2 + \l__wrapblock_full_int + \l__wrapblock_lines_int }
  \int_gset_eq:NN \g__wrapblock_narrowed_int \l__wrapblock_lines_int
  \int_compare:nNnT { \l__wrapblock_lines_int } > { 0 }
    {
```

```
      \seq_gput_right:Ne \g__wrapblock_shapes_seq
        {
          \int_eval:n { \l__wrapblock_lines_int + 1 } /
          \bool_lazy_and:nnTF
            { \int_compare_p:nNn {#2} = { 0 } }
            { \int_compare_p:nNn { \l__wrapblock_full_int } > { 0 } }
            { \dim_use:N \l__wrapblock_i_dim / \dim_use:N \l__wrapblock_l_dim }
            { \dim_use:N \l__wrapblock_ni_dim / \dim_use:N \l__wrapblock_nl_dim }
        }
    }
  \tl_gset:Ne \g__wrapblock_parshape_tl
    {
      \exp_not:N \tex_parshape:D
        \int_eval:n { \l__wrapblock_lines_int + 1 } \c_space_tl
        \l__wrapblock_shape_tl
        \dim_use:N \l__wrapblock_i_dim \c_space_tl \dim_use:N \l__wrapblock_l_dim
        \exp_not:N \scan_stop:
    }
  \tl_gset:Ne \g__wrapblock_pen_tl
    {
      \exp_not:N \tex_interlinepenalties:D
        \int_eval:n { \l__wrapblock_lines_int + 1 } \c_space_tl
        \l__wrapblock_pen_tl \int_use:N \interlinepenalty
        \exp_not:N \scan_stop:
    }
  #1 \g__wrapblock_parshape_tl
  #1 \g__wrapblock_pen_tl
  \int_compare:nNnT { \l__wrapblock_lines_int } > { 0 }
    {
      \bool_gset_true:N \g__wrapblock_inpar_bool
      \__wrapblock_reshape_after_group:
    }
}
```

`\__wrapblock_shape:Nnnn` is the heart of the package: it sets the `\parshape` and the `\interlinepenalties` of the paragraph. Its arguments are:

`#1` nothing, or `\global` after a displayed equation (section 11.10);

`#2` the number of lines of the paragraph already typeset, before a displayed equation; 0 at the start of a paragraph;

`#3` the number of lines kept at full width before the narrowed ones, for a deferred image; 0 otherwise;

`#4` the baseline of the first line to narrow.

`\__wrapblock_shape:nn` and `\__wrapblock_shape:Nnn` are short forms for the usual cases, where `#3` is 0.

How many lines are narrowed. The lines to narrow are those whose top lies above `\g__wrapblock_bottom_dim`. Line k, counted from 0, has its baseline at $\#4 + k$ `\baselineskip` and its top a strut height higher. It is narrowed when $k <$ (`\g__wrapblock_bottom_dim` $- \#4 + h_{\text{strut}}$)/`\baselineskip`: the count is the ceiling of that ratio. A line taller than a strut pushes the next ones down; the count is then one line too large, never too small. A line may be narrowed below the image, but no line runs over it.

How wide they are. The narrowed lines are cut from the shape LaTeX gave the paragraph, which is not always the full width: in a list, LaTeX sets a `\parshape` of one line, with the indentation and the width of the item. With the image on the left, the indentation of the line grows by `\g__wrapblock_shift_dim` and its width shrinks as much. With the image on the right, the line keeps its indentation and ends `\g__wrapblock_shift_dim` before the right edge of the block;

a nested list, which starts further right, loses less of its own width. The width left is kept in `\g__wrapblock_avail_dim`, for `\__wrapblock_box_check:`.

The two lists. The new `\parshape` lists, in order: `#2` entries for the lines already typeset (their values no longer matter, but `\parshape` counts from the first line of the paragraph); `#3` full-width lines; the narrowed lines; and a last entry, the shape LaTeX had given, which TeX repeats for all the lines that follow. `\interlinepenalties` follows the same order:

- after a narrowed line, 10 000: the zone cannot break across pages;
- after a full-width line, the usual `\interlinepenalty`;
- after the last full-width line, −10 000: this forces the page break of a deferred image. A lower value is not used: LaTeX keeps those for its own output routine.

For instance, in the usual case (`#2` and `#3` are 0), a paragraph of full width 345 pt with four lines beside an image that takes 100 pt on the right gets

```
\parshape 5  0pt 245pt  0pt 245pt  0pt 245pt  0pt 245pt  0pt 345pt
\interlinepenalties 5  10000 10000 10000 10000 0
```

The signature. The number of entries of each shape, with the indentation and the width of its first entry, is kept in `\g__wrapblock_shapes_seq`, for `\__wrapblock_unstale:`.

```
% TeX clears \parshape at the end of a paragraph, but locally. A paragraph
% ended inside a deeper group (an item that opens a nested list, a
% tcolorbox that starts in the group of the next one) leaves its narrowed
% shape behind, and a later paragraph gets it back, maybe only when some
% group ends after it has started. It is looked for at the end of a
% paragraph that was given no shape, just before the line breaking: it is
% recognised among the shapes set in the block; its last entry is what
% LaTeX had given.
\hook_gput_code:nnn { para / end } { wrapblock }
  {
    \bool_lazy_all:nT
      {
        { \l__wrapblock_outer_bool }
        { \l__wrapblock_inside_bool }
        { ! \g__wrapblock_inpar_bool }
      }
      { \__wrapblock_unstale: }
  }
\cs_new_protected:Npn \__wrapblock_unstale:
  {
    \int_compare:nNnT { \parshape } > { 1 }
      {
        \seq_if_in:NeT \g__wrapblock_shapes_seq
          {
            \int_eval:n { \parshape } /
            \dim_eval:n { \parshapeindent 1 } / \dim_eval:n { \parshapelength 1 }
          }
          {
            \tex_parshape:D 1 ~
              \dim_eval:n { \parshapeindent \parshape } \c_space_tl
              \dim_eval:n { \parshapelength \parshape } \scan_stop:
            \tex_interlinepenalties:D 0 ~
          }
      }
  }
```

TeX clears `\parshape` at the end of every paragraph, but only locally. Suppose a paragraph ends inside a deeper group than the one it started in: an item whose text opens a nested list, for instance. When that group ends, the narrowed shape comes back, and the next paragraph

of the outer level would be narrowed a second time. The leftover may also come back only after a later paragraph has started: a breakable `tcolorbox` beside the image, followed with no blank line by another one, starts its paragraph in a group, so that `\__wrapblock_reshape:` sets its shape again at the level of the block; that paragraph ends in the group of the second box, and the shape stays at the level of the block, where the paragraph of the second box, which starts deeper, finds it when it ends. `\__wrapblock_unstale:` is therefore called at `para/end`, just before the line breaking, in a paragraph of the block that was given no shape (`\g__wrapblock_inpar_bool` false): a `\parshape` of more than one line whose signature is among those the block has set is a leftover. It keeps only its last entry, which is the shape LaTeX had given. A paragraph that is given a shape needs nothing: `\__wrapblock_shape:Nnnn` reads its base line in the last entry, which is the same.

```
% \parshape is local. A paragraph that starts inside a group and goes on
% after it ({\bfseries Bold} text, a breakable tcolorbox) would lose
% its shape when the group ends, and run over the image. The shape is set
% again after every group that ends while the paragraph goes on.
\cs_new_protected:Npn \__wrapblock_reshape_after_group:
  {
    \int_compare:nNnT { \tex_currentgrouplevel:D } > { 0 }
      { \group_insert_after:N \__wrapblock_reshape: }
  }
\cs_new_protected:Npn \__wrapblock_reshape:
  {
    \bool_lazy_all:nT
      {
        { \g__wrapblock_inpar_bool }
        { \mode_if_horizontal_p: }
        { ! \mode_if_inner_p: }
      }
      {
        \g__wrapblock_parshape_tl
        \g__wrapblock_pen_tl
        \__wrapblock_reshape_after_group:
      }
  }
```

The opposite case: a paragraph that starts inside a group that ends before the paragraph does. `{\bfseries Bold} text` at the start of a paragraph is one; a breakable `tcolorbox`, which starts the paragraph of each of its parts in a group, is another. The shape set by `para/begin` is local to that group, and is gone when the group ends, before the paragraph is broken into lines. `\__wrapblock_shape:Nnnn` keeps the two assignments it made in `\g__wrapblock_parshape_tl` and `\g__wrapblock_pen_tl`, and, inside a group, asks `\aftergroup` for `\__wrapblock_reshape:`. That function sets the shape again if the same narrowed paragraph is still being typeset (`\g__wrapblock_inpar_bool`, cleared by `para/after`), in outer horizontal mode, and asks again for the group around, if any. When the paragraph ends inside the group, as usual, `\__wrapblock_reshape:` comes in vertical mode and does nothing.

11.8 Setting the image

```
% The image #2, in horizontal mode, with the width key #1. With a length,
% it is a minipage of that width, centred: it may hold paragraphs and a
% caption. Its bottom line is on the baseline, as that of a lone image.
% The image is not the text of the block: its boxes are not checked.
\cs_new_protected:Npn \__wrapblock_image:nn #1#2
  {
    \bool_set_true:N \l__wrapblock_inimage_bool
    \bool_set_false:N \l__wrapblock_outer_bool
    \dim_compare:nNnTF {#1} = { - \c_max_dim }
      {#2}
```

```
      { \begin{minipage}[b]{#1} \centering #2 \end{minipage} }
  }
\cs_generate_variant:Nn \__wrapblock_image:nn { nv }
% The image is built twice: once to be measured at the start of the
% block, once where it is placed. Built outside a paragraph, a box with
% text in it (\fbox, tabular, \parbox) would open its structure at the
% wrong place in a tagged PDF.
\cs_new_protected:Npn \__wrapblock_build:
  {
    \hbox_gset:Nn \g__wrapblock_img_box
      { \__wrapblock_image:nv { \g__wrapblock_img_width_dim } { g__wrapblock_img_tl } }
  }

% The image, flat (no width, height or depth), at the start of line 1.
\cs_new_protected:Npn \__wrapblock_place_image:
  {
    \__wrapblock_build:
    \hbox_set:Nn \l_tmpa_box
      {
        \tex_kern:D \dim_eval:n
          {
            \str_if_eq:VnTF \g__wrapblock_side_tl { l }
              { \g__wrapblock_left_dim }
              { \g__wrapblock_left_dim + \g__wrapblock_width_dim - \box_wd:N \g__wrapblock_img_box
     }
            - \l__wrapblock_ni_dim
          }
        \box_move_down:nn
          { \box_ht:N \g__wrapblock_img_box - \box_ht:N \strutbox }
          { \box_use:N \g__wrapblock_img_box }
      }
    \box_set_wd:Nn \l_tmpa_box { 0pt }
    \box_set_ht:Nn \l_tmpa_box { 0pt }
    \box_set_dp:Nn \l_tmpa_box { 0pt }
    % Centred or ragged-left text: \leftskip comes before the image and
    % would move it. The opposite glue takes it back to the start of the
    % line, and the same glue after it puts the text where it was.
    % After a heading with no indentation, \@afterheading removes the
    % \lastbox of the line, meant to be the indentation box: a kern after
    % the image keeps it from being taken instead.
    \skip_if_eq:nnTF { \leftskip } { 0pt }
      { \box_use_drop:N \l_tmpa_box \tex_kern:D 0pt \scan_stop: }
      {
        \tex_penalty:D 10000 ~
        \skip_horizontal:n { - \leftskip }
        \box_use_drop:N \l_tmpa_box
        \tex_penalty:D 10000 ~
        \skip_horizontal:N \leftskip
      }
    \bool_gset_false:N \g__wrapblock_pending_bool
  }
```

`\__wrapblock_image:nn` typesets the image `#2` as it is, or, when the `width` key `#1` gives a length, in a `minipage` of that width, centred. The `minipage` admits paragraphs and a caption, which an `\hbox` refuses; aligned on its bottom line (`[b]`), a single image keeps the height and depth it has on its own. `\l__wrapblock_outer_bool` is off in the image, which is not the text of the block: a box inside it is not checked. The deferred image of chunk `overlay` is built by the same function.

`\__wrapblock_build:` typesets the image in `\g__wrapblock_img_box`. It is called twice: by the environment, to measure the image, and by `\__wrapblock_place_image:`, inside the first paragraph. `\l__wrapblock_inimage_bool` is set inside the box: a `wrapblock` met there is an error. The second call is not a waste: in a tagged PDF, an `\fbox`, a `tabular` or a `\parbox` built

outside a paragraph would open its structure elements at the wrong place.

`\__wrapblock_place_image:` puts the image at the start of the first line of the block:

- The image goes in a box of zero width, height and depth, so that the line keeps its height and its natural width.
- The first line starts at its own indentation, the narrowed one. A kern moves the image from there to the left edge of the block, or to its right edge minus the width of the image.
- The image is lowered so that its top is at the top of the first line, a strut height above the baseline.
- In centred or ragged-left text, `\leftskip` comes before the image in the line, and would move it. The opposite glue is put before the image, and `\leftskip` again after it: the two cancel exactly, whatever their stretch, and the image is back at the start of the line. The two penalties forbid a line break at these glues.
- Otherwise, a kern of zero width follows the image. After a heading whose next paragraph is not indented (the LaTeX default in English), the `\everypar` of `\@afterheading` removes the `\lastbox` of the line, meant to be the indentation box. The image would be taken instead, and lost; the kern stops `\lastbox`, which then finds nothing to remove, or the indentation box when it is there.

```
% A block that comes first in an \item: the labels, which LaTeX sets at the start
% of the first line, would follow the text past a left image and land on
% it. Move them back to their place in the margin.
\cs_new_protected:Npn \__wrapblock_labels_back:
  {
    \str_if_eq:VnT \g__wrapblock_side_tl { l }
      {
        \legacy_if:nT { @inlabel }
          {
            \hbox_gset:Nn \@labels
              {
                \tex_kern:D - \g__wrapblock_shift_dim
                \hbox_unpack:N \@labels
                \tex_kern:D \g__wrapblock_shift_dim
              }
          }
      }
  }
```

When the block comes first in an `\item`, LaTeX typesets the label at the start of the first line, from the box `\@labels`; the label reaches into the margin by a negative kern. With the image on the left, the first line starts after the image, and the label would land on the image. The label box is wrapped in two kerns: the first moves the label back by `\g__wrapblock_shift_dim`, the second gives the width back.

11.9 Keeping the zone on one page

```
\hook_gput_code:nnn { para / after } { wrapblock }
  {
    \bool_if:NT \l__wrapblock_outer_bool
      {
        \bool_gset_false:N \g__wrapblock_inpar_bool
        \bool_if:NT \g__wrapblock_active_bool { \__wrapblock_para_after: }
      }
  }
\cs_new_protected:Npn \__wrapblock_para_after:
```

```
  {
    \tex_interlinepenalties:D 0 ~
    \bool_if:NT \g__wrapblock_global_bool
      {
        \tex_global:D \tex_parshape:D 0 ~
        \tex_global:D \tex_interlinepenalties:D 0 ~
        \bool_gset_false:N \g__wrapblock_global_bool
      }
    \int_set_eq:NN \predisplaypenalty  \g__wrapblock_save_pre_int
    \int_set_eq:NN \postdisplaypenalty \g__wrapblock_save_post_int
    % A deferred image waits until a paragraph goes past its last
    % full-width line, which ends the page.
    \bool_if:NTF \g__wrapblock_wait_bool
      {
        \int_compare:nNnT { \tex_prevgraf:D } > { \g__wrapblock_full_end_int }
          {
            % The \par that ended the paragraph ran the page builder, which
            % met the forced break: the page is shipped, and the image goes
            % to the page being built. It may come later than planned: a
            % footnote in the full-width lines shortens the page, which then
            % breaks twice.
            \int_compare:nNnT
              { \g__wrapblock_ship_int } > { \g__wrapblock_origin_ship_int + 1 }
              { \msg_warning:nn { wrapblock } { early } }
            \__wrapblock_commit:n { \g__wrapblock_ship_int }
            \__wrapblock_zone_after:
          }
      }
      { \__wrapblock_zone_after: }
  }
\cs_new_protected:Npn \__wrapblock_zone_after:
  {
    % A paragraph with more lines than were narrowed went on below the
    % image: the zone is over, whatever page it ended on. Otherwise the
    % zone may go on; it cannot break, so a new page means it moved.
    \int_compare:nNnTF { \tex_prevgraf:D } > { \g__wrapblock_narrowed_int }
      { \__wrapblock_stop: }
      {
        \__wrapblock_check_ship:
        \dim_compare:nNnTF { \pagetotal + \pagedepth } < { \g__wrapblock_bottom_dim }
          { \nobreak \__wrapblock_list_penalties_off: }
          { \__wrapblock_stop: }
      }
  }
```

At the end of each paragraph, `\__wrapblock_para_after:` first tidies up: it clears the `\interlinepenalties` of the paragraph (TEX keeps them, unlike `\parshape`) and puts back the penalties around displayed equations. After a displayed equation, shape and penalties were set globally, and are cleared globally.

The branch for a waiting deferred image is explained in section 11.11. Otherwise `\__wrapblock_zone_after:` decides whether the zone goes on:

- *The paragraph has more lines than were narrowed*: it went below the image, and the zone is over.
- *Otherwise*, the primitive `\par` has run the page builder, and `\pagetotal` plus `\pagedepth` is where the paragraph ended. If that is above the bottom of the image, the next paragraph starts beside the image too. `\nobreak` forbids a page break between the two paragraphs, and the list penalties do the same when the next paragraph is an item, or starts or ends a list. If the paragraph ended at or below the bottom of the image, the zone is over.

```
\cs_new_protected:Npn \__wrapblock_list_penalties_off:
  {
    \int_gset:Nn \@itempenalty     { 10000 }
    \int_gset:Nn \@beginparpenalty { 10000 }
    \int_gset:Nn \@endparpenalty   { 10000 }
  }
\cs_new_protected:Npn \__wrapblock_list_penalties_restore:
  {
    \int_gset_eq:NN \@itempenalty     \g__wrapblock_save_item_int
    \int_gset_eq:NN \@beginparpenalty \g__wrapblock_save_beginpar_int
    \int_gset_eq:NN \@endparpenalty   \g__wrapblock_save_endpar_int
  }
\cs_new_protected:Npn \__wrapblock_stop:
  {
    \bool_gset_false:N \g__wrapblock_active_bool
    \__wrapblock_list_penalties_restore:
  }
```

`\@itempenalty`, `\@beginparpenalty` and `\@endparpenalty` are the penalties LaTeX puts before an item, before a list and after a list. They are set globally to 10 000 while the zone runs. `\__wrapblock_stop:` ends the zone and puts them back.

```
% The region moved to a new page while it was running. This happens only
% when insertions (footnotes) shrank the page under it. Assume it now sits
% at the top of the page and say so.
\cs_new_protected:Npn \__wrapblock_check_ship:
  {
    \int_compare:nNnF { \g__wrapblock_ship_int } = { \g__wrapblock_origin_ship_int }
      {
        \msg_warning:nn { wrapblock } { moved }
        \int_gset_eq:NN \g__wrapblock_origin_ship_int \g__wrapblock_ship_int
        \dim_gset:Nn \g__wrapblock_bottom_dim
          { \topskip - \box_ht:N \strutbox + \g__wrapblock_height_dim }
      }
  }
\msg_new:nnn { wrapblock } { moved }
  { The~block~that~ends~on~input~line~\msg_line_number:\ moved~to~a~new~page~
    while~its~text~was~set.~Its~wrapping~may~be~off~by~some~lines. }
\msg_new:nnn { wrapblock } { nested }
  { wrapblock~blocks~cannot~be~nested. }
```

All these penalties forbid a page break in the zone. A footnote can still make the page too short for lines that were already counted, and TeX then breaks where it can. `\__wrapblock_check_ship:` notices it: the number of pages shipped is no longer what it was at the start of the zone. The zone is then assumed to go on at the top of the new page, and the warning `moved` says that the wrapping may be wrong by a few lines. The tests could not make this happen: TeX moves the whole block before it comes to that.

11.10 Displayed equations

```
% After a displayed equation the paragraph goes on; its remaining lines
% are re-measured from where the display actually ended.
\cs_new_protected:Npn \__wrapblock_after_display:
  {
    \bool_if:NT \g__wrapblock_active_bool
      {
        \dim_set:Nn \l__wrapblock_base_dim
          { \pagetotal + \baselineskip - \box_dp:N \strutbox }
        \bool_if:NTF \g__wrapblock_wait_bool
          { \__wrapblock_defer_para:N \tex_global:D }
          {
            \__wrapblock_check_ship:
```

```
            \__wrapblock_shape:Nnn \tex_global:D { \prevgraf } { \l__wrapblock_base_dim }
          }
        \bool_gset_true:N \g__wrapblock_global_bool
      }
  }
\cs_new_protected:Npn \__wrapblock_display_hook:
  {
    \bool_if:NT \g__wrapblock_active_bool
      { \bool_if:NT \l__wrapblock_outer_bool
          { \group_insert_after:N \__wrapblock_after_display: } }
  }
\hook_gput_code:nnn { begindocument } { wrapblock }
  {
    \tex_everydisplay:D \exp_after:wN
      { \tex_the:D \tex_everydisplay:D \__wrapblock_display_hook: }
  }
```

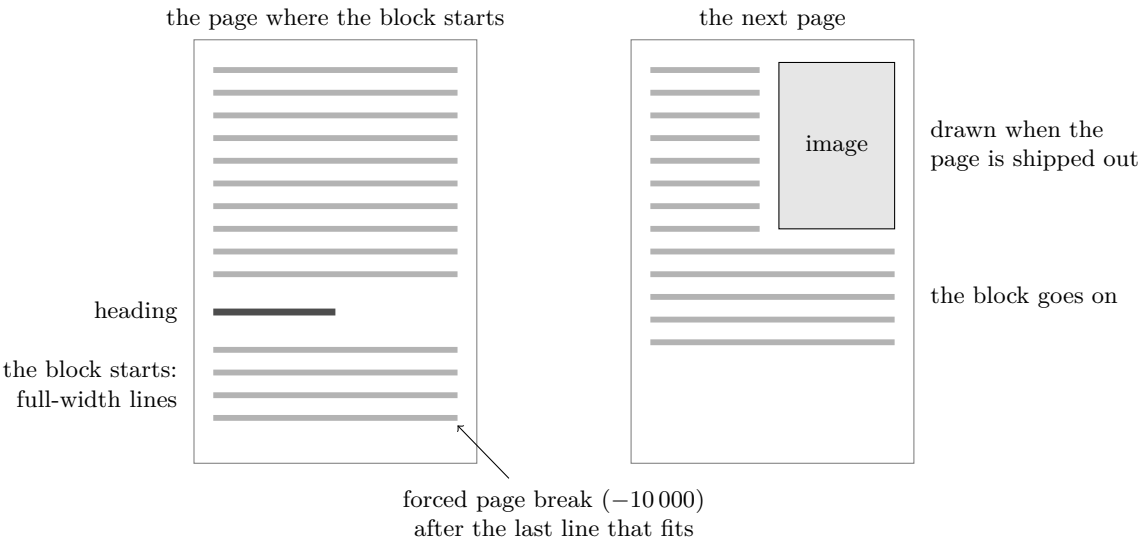
A displayed equation interrupts a paragraph without ending it. TEX breaks the lines before the equation, typesets the equation, then goes on with the same paragraph, whose `\parshape` still counts from its first line. But below the equation, the lines are no longer where `\__wrapblock_shape:Nnnn` expected them: the equation has its own height and its own spacing. The shape must be computed again.

- `\__wrapblock_display_hook:` is added to `\everydisplay`. It uses `\group_insert_after:N` to run `\__wrapblock_after_display:` right after the end of the equation.
- At that point, the equation is on the page and the page builder has run. `\prevgraf` is the number of lines typeset so far, the equation counting for three. The baseline of the next line is estimated from `\pagetotal`, taking the depth of a strut for that of the equation.
- The shape is computed again from line `\prevgraf` on: this is what the argument `#2` of `\__wrapblock_shape:Nnnn` is for.

The shape is set globally. With amsmath, the end of an `equation` or an `align` is followed by the end of the group of the environment, which would undo a local `\parshape`. `\g__wrapblock_global_bool` tells `\__wrapblock_para_after:` to clear it globally too.

11.11 The deferred image

When the image does not fit at the bottom of the page and the text is long enough, the text runs at full width to the bottom of the page, and the image goes to the top of the next one.



TEX does neither of two things by itself: break the page after a given line of a paragraph, and draw an image on a page where it was not typeset. The image goes through three states: *waiting*, from the start of the block until the page breaks; *committed*, once its page is known; drawn, when that page is shipped out.

```
% A deferred image, before the page break. #1 is \prg_do_nothing: at the
% start of a paragraph, \tex_global:D after a display. The baseline of the
% next line is in \l__wrapblock_base_dim, and \prevgraf lines are set.
% Count the full-width lines that still fit on the page: none, and the
% page breaks before the next line; some, and it breaks after the last.
\cs_new_protected:Npn \__wrapblock_defer_para:N #1
  {
    \int_compare:nNnTF { \g__wrapblock_ship_int } = { \g__wrapblock_origin_ship_int }
      {
        % the footnotes to come, as the last run found them
        \dim_set:Nn \l_tmpa_dim
          { \g__wrapblock_short_dim - \g__wrapblock_goal_dim + \pagegoal }
        \dim_compare:nNnT { \l_tmpa_dim } < { 0pt } { \dim_zero:N \l_tmpa_dim }
        \dim_set:Nn \l_tmpa_dim { \pagegoal - \l_tmpa_dim }
        \dim_compare:nNnTF
          { \l__wrapblock_base_dim + \box_dp:N \strutbox } > { \l_tmpa_dim }
          { \int_zero:N \l__wrapblock_full_int }
          {
            \int_set:Nn \l__wrapblock_full_int
              {
                \fp_eval:n
                  {
                    floor ( \dim_to_fp:n { \l_tmpa_dim - \l__wrapblock_base_dim - \box_dp:N \
    strutbox }
                            / \dim_to_fp:n { \baselineskip } ) + 1
                  }
              }
          }
        \int_compare:nNnT { \l__wrapblock_full_int } = { 0 }
          { \tex_vadjust:D pre { \tex_penalty:D -10000 ~ } }
      }
      % the page already broke before this paragraph
      { \int_zero:N \l__wrapblock_full_int }
    \int_compare:nNnTF { \l__wrapblock_full_int } = { 0 }
      {
        % the image goes to the page after the coming break, or to the
        % page being built if the page already broke
        \__wrapblock_commit:n
          {
            \g__wrapblock_ship_int
            \int_compare:nNnT { \g__wrapblock_ship_int } = { \g__wrapblock_origin_ship_int } { + 1
     }
          }
        \__wrapblock_shape:Nnnn #1 { \tex_prevgraf:D } { 0 } { \topskip }
      }
      {
        \int_gset:Nn \g__wrapblock_full_end_int
          { \tex_prevgraf:D + \l__wrapblock_full_int }
        \bool_gset_true:N \g__wrapblock_breakpar_bool
        \__wrapblock_shape:Nnnn #1 { \tex_prevgraf:D } { \l__wrapblock_full_int } { \topskip }
      }
    \int_set:Nn \predisplaypenalty  { 10000 }
    \int_set:Nn \postdisplaypenalty { 10000 }
  }
```

`\__wrapblock_defer_para:N` runs while the image is waiting: at the start of each paragraph of the block, and after each displayed equation in it. Its job is to count the full-width lines that still fit on the page.

The room. A line fits when its bottom, a strut depth below its baseline, is above the bottom of the page, `\pagegoal`. Footnotes that come later in these lines will take room that `\pagegoal` does not show yet. The previous run measured it: `\g__wrapblock_short_dim` is the room the footnotes of the block took (see the chunk `aux`). What they take already is the decrease of `\pagegoal` since the block started, `\g__wrapblock_goal_dim` minus `\pagegoal`. The difference is still to come, and is subtracted from `\pagegoal`.

The three cases.

- *Some lines fit.* The shape keeps them at full width, forces the break after the last of them, and narrows the lines that follow, which will be at the top of the next page: their first baseline is `\topskip`. `\g__wrapblock_full_end_int` keeps the number of that last full-width line. If the paragraph ends before it, nothing more happens, and the next paragraph counts again. If the paragraph goes past it, the page breaks inside the paragraph: `\g__wrapblock_breakpar_bool` is set for `para/end`, and `\__wrapblock_para_after:` will commit the image.
- *No line fits.* The page must break before the first line of the paragraph: a `\vadjust pre` puts a penalty of −10 000 above that line. The image is committed at once, to the next page, and the lines are narrowed from the first one.
- *The page has already broken*, between two paragraphs: the image is committed to the page being built, and the lines are narrowed from the first one.

```
% TeX adds the club, widow and broken penalties to the -10000 that ends
% the page, which then no longer forces the break. They are cleared at the
% end of the paragraph, just before it is broken into lines: at its start,
% the \everypar of LaTeX would set \clubpenalty again after the hook.
\hook_gput_code:nnn { para / end } { wrapblock }
  {
    \bool_lazy_and:nnT { \l__wrapblock_outer_bool } { \g__wrapblock_breakpar_bool }
      {
        \bool_gset_false:N \g__wrapblock_breakpar_bool
        \int_set:Nn \clubpenalty   { 0 }
        \int_set:Nn \widowpenalty  { 0 }
        \int_set:Nn \brokenpenalty { 0 }
        \tex_clubpenalties:D  0 ~
        \tex_widowpenalties:D 0 ~
      }
  }
```

The −10 000 that ends the page must reach the page builder unchanged: a penalty of −10 000 or less forces a break, anything above only suggests one. But TEX adds to the penalty between two lines the club penalty after the first line of a paragraph, the widow penalty before the last one, and the broken penalty after a hyphenated line. With a club penalty of 150, the forced break becomes −9 850, and TEX breaks the page a line earlier or later, as it pleases.

These penalties are therefore cleared in the paragraph that ends the page. This is done at `para/end`, just before TEX breaks the paragraph into lines. At `para/begin` it would be too early: the `\everypar` of LATEX, which runs afterwards (for `\@afterheading`, for `\item`), sets `\clubpenalty` again.

When the page has broken. This is the branch of `\__wrapblock_para_after:` for a waiting image (chunk `paraafter`, above). It acts when the paragraph has more lines than `\g__wrapblock_full_end_int`: the forced break is then inside the paragraph that just ended. The primitive `\par` that ended it has run the page builder, which met the −10 000 and ran the output routine: the page is shipped out, and `\g__wrapblock_ship_int` counts it. The image is committed to the page being built, the one that starts with the narrowed lines, and `\__wrapblock_zone_after:` takes over as for any zone.

One check comes first. If more than one page was shipped since the block started, the page broke twice: a footnote in the full-width lines shortened the page, which broke before the planned line. The image is still drawn on the page with the narrowed lines, but a page with a few lines is left before it. The warning `early` asks for another run, which will avoid it (chunk `aux`).

```
% The image goes to the top of the page shipped after #1 pages.
\cs_new_protected:Npn \__wrapblock_commit:n #1
  {
    \bool_gset_false:N \g__wrapblock_wait_bool
    \int_gset:Nn \g__wrapblock_origin_ship_int {#1}
    \int_gincr:N \g__wrapblock_def_int
    \tl_gset_eq:cN { g__wrapblock_def_ \int_use:N \g__wrapblock_def_int _tl }
      \g__wrapblock_img_tl
    \tl_gset:ce { g__wrapblock_def_ \int_use:N \g__wrapblock_def_int _data_tl }
      {
        { \int_use:N \g__wrapblock_origin_ship_int }
        { \dim_use:N \g__wrapblock_left_dim }
        { \dim_use:N \g__wrapblock_width_dim }
        { \dim_use:N \g__wrapblock_img_width_dim }
        { \g__wrapblock_side_tl }
        { \int_use:N \g__wrapblock_struct_int }
      }
    \seq_gput_right:Ne \g__wrapblock_def_seq { \int_use:N \g__wrapblock_def_int }
  }
```

`\__wrapblock_commit:n` records a deferred image for the page that will be shipped out after `#1` pages. The material of the image and its data (page, left edge and width of the block, `width` key, side, structure element) are kept in two token lists numbered by `\g__wrapblock_def_int`, and the number is added to `\g__wrapblock_def_seq`, the list of the images that wait for their page. The image is not typeset here: it is typeset when its page is shipped out.

```
% An empty structure element where the block starts: a deferred image is
% put in it when its page is shipped.
\cs_new_protected:Npn \__wrapblock_placeholder:
  {
    \int_gzero:N \g__wrapblock_struct_int
    \cs_if_exist:NT \tag_if_active:T
      {
        \tag_if_active:T
          {
            \tag_struct_begin:n { tag = NonStruct }
            \int_gset:Nn \g__wrapblock_struct_int { \tag_get:n { struct_num } }
            \tag_struct_end:
          }
      }
  }
```

In a tagged PDF, the structure follows the order in which things are typeset. An image drawn when the page is shipped out would come after everything else on its page, after the text it goes with. An empty structure element, of type `NonStruct`, is therefore made where the block starts; the image will be attached to it. Without tagging, `\g__wrapblock_struct_int` stays at 0.

```
% The floats LaTeX is about to put at the top of the page take room above
% the text: a deferred image is drawn below them. Their height is that of
% the boxes in \@toplist, \floatsep between them and \textfloatsep below.
\cs_new_protected:Npn \__wrapblock_top_floats:
  {
    \dim_gzero:N \g__wrapblock_top_dim
    \tl_if_empty:NF \@toplist
      {
        \group_begin:
          \cs_set:Npn \@elt ##1
            {
```

```
              \dim_gadd:Nn \g__wrapblock_top_dim
                { \box_ht:N ##1 + \box_dp:N ##1 + \floatsep }
            }
          \@toplist
        \group_end:
        \dim_gadd:Nn \g__wrapblock_top_dim { \textfloatsep - \floatsep }
      }
  }
```

LaTeX adds the floats of the top of a page when the page is finished, in the output routine: `\pagetotal` never counts them, and the narrowed lines, which follow the text, come out below them. The deferred image must be drawn below them too. The `build/column/before` hook runs just before LaTeX puts them in place, and they are still in the list `\@toplist`. `\__wrapblock_top_floats:` adds up their height, the way LaTeX stacks them: each box, `\floatsep` between two of them, `\textfloatsep` below the last. The natural lengths are used: on a page stretched by `\flushbottom`, the image may be off by the stretch of `\textfloatsep`, a few points.

```
% Deferred images are set on the page they belong to as it is shipped,
% at the top of the text area: the lines beside them are already narrowed.
% In a tagged PDF the image goes in a structure element made, empty, where
% the block starts, so that it keeps its place in the reading order.
% The foreground is asked for only when an image is waiting: once the hook
% holds code, LaTeX builds a picture on every page, which made a document
% without any block up to 40 % slower.
\hook_gput_code:nnn { shipout / before } { wrapblock }
  {
    \seq_if_empty:NF \g__wrapblock_def_seq
      { \hook_gput_next_code:nn { shipout / foreground } { \__wrapblock_overlays: } }
  }
\cs_new_protected:Npn \__wrapblock_overlays:
  {
    \seq_clear:N \l__wrapblock_done_seq
    \seq_map_inline:Nn \g__wrapblock_def_seq
      {
        \exp_last_unbraced:Nv \__wrapblock_overlay:nnnnnnn
          { g__wrapblock_def_ ##1 _data_tl } {##1}
      }
    \seq_map_inline:Nn \l__wrapblock_done_seq
      { \seq_gremove_all:Nn \g__wrapblock_def_seq {##1} }
  }
% #1 page (pages shipped before it), #2 left edge and #3 width of the
% block, #4 width key, #5 side, #6 structure, #7 number of the image
\cs_new_protected:Npn \__wrapblock_overlay:nnnnnnn #1#2#3#4#5#6#7
  {
    \int_compare:nNnT {#1} = { \g__wrapblock_ship_int }
      {
        \seq_put_right:Nn \l__wrapblock_done_seq {#7}
        \bool_lazy_and:nnT
          { \int_compare_p:nNn {#6} > { 0 } }
          { \cs_if_exist_p:N \tag_struct_begin:n }
          { \tag_struct_begin:n { tag = P , parent = #6 } }
        \hbox_set:Nn \l_tmpa_box
          {
            \cs_if_exist_use:NT \tag_mc_begin:n { { } }
            \dim_set:Nn \wrapblocklinewidth {#3}
            \__wrapblock_image:nv {#4} { g__wrapblock_def_ #7 _tl }
            \cs_if_exist_use:N \tag_mc_end:
          }
        \put
          (
            \dim_eval:n
              {
                1in + \hoffset
```

```
                + \legacy_if:nTF { @twoside }
                    { \int_if_odd:nTF { \c@page } { \oddsidemargin } { \evensidemargin } }
                    { \oddsidemargin }
                + #2
                + \str_if_eq:nnTF {#5} { l } { 0pt } { #3 - \box_wd:N \l_tmpa_box }
              } ,
            \dim_eval:n
              {
                - 1in - \voffset - \topmargin - \headheight - \headsep
                - \g__wrapblock_top_dim
                - \topskip + \box_ht:N \strutbox - \box_ht:N \l_tmpa_box
              }
          )
          { \box_use_drop:N \l_tmpa_box }
        \bool_lazy_and:nnT
          { \int_compare_p:nNn {#6} > { 0 } }
          { \cs_if_exist_p:N \tag_struct_begin:n }
          { \tag_struct_end: }
        \cs_undefine:c { g__wrapblock_def_ #7 _tl }
        \cs_undefine:c { g__wrapblock_def_ #7 _data_tl }
      }
  }
```

The deferred images are drawn by the `shipout/foreground` hook, in a `picture` whose origin is the top left corner of the paper. As soon as this hook holds code, LaTeX builds that picture on every page, and a document of a thousand pages without a single block compiled a third slower. The code is therefore given to the hook for the next page only, by the `shipout/before` hook, and only when the list of images that wait is not empty.

- `\__wrapblock_overlays:` goes through the images that wait. The images drawn are removed from the list afterwards, not during the loop that reads it.
- `\__wrapblock_overlay:nnnnnnn` draws an image if its page is the one being shipped: the image was committed for the page shipped after `#1` pages, and `\g__wrapblock_ship_int` pages have been shipped.
- *Vertically*, the image is put below the top of the text area (1 in, `\voffset`, `\topmargin`, `\headheight`, `\headsep`), below the floats of the top of the page, with its top at the top of the first line: `\topskip` minus the height of a strut. This is where an image that is not deferred sits.
- *Horizontally*, it starts from the left margin, which depends on the parity of the page in a two-sided document, then from the left edge of the block; on the right, it is flush with the right edge of the block.
- *In a tagged PDF*, the image is wrapped in a marked-content sequence and a `P` structure element, whose parent is the placeholder made where the block started.

```
% A footnote in the full-width lines of a deferred block shortens the page
% after the lines were counted: the page breaks before the planned break,
% at the end of a column whose \outputpenalty is not -10000. The room the
% footnotes took is then written to the .aux, and the next run counts the
% lines without it. A value found once is kept: the footnote may go below
% the break on the next run.
\cs_new_protected:cpn { wrapblock@short } #1#2
  { \prop_gput:Nnn \g__wrapblock_short_prop {#1} {#2} }
\hook_gput_code:nnn { build / column / before } { wrapblock }
  {
    \__wrapblock_top_floats:
    \bool_lazy_and:nnT
      { \g__wrapblock_wait_bool }
```

```
      { \int_compare_p:nNn { \tex_outputpenalty:D } > { -10000 } }
      {
        \dim_set:Nn \l_tmpa_dim
          {
            \g__wrapblock_goal_dim - \tex_vsize:D
            \box_if_empty:NF \footins
              { + \box_ht:N \footins + \box_dp:N \footins + \tex_skip:D \footins }
          }
        \dim_compare:nNnF { \l_tmpa_dim } > { 0pt }
          { \dim_set_eq:NN \l_tmpa_dim \baselineskip }
        \prop_gput:Nee \g__wrapblock_short_new_prop
          { \int_use:N \g__wrapblock_block_int }
          { \dim_eval:n { \g__wrapblock_short_dim + \l_tmpa_dim } }
      }
  }
\hook_gput_code:nnn { enddocument / afterlastpage } { wrapblock }
  {
    \iow_now:Nn \@auxout { \providecommand \wrapblock@short [2] { } }
    \prop_map_inline:Nn \g__wrapblock_short_prop
      {
        \prop_if_in:NnF \g__wrapblock_short_new_prop {#1}
          { \iow_now:Nn \@auxout { \wrapblock@short {#1} {#2} } }
      }
    \prop_map_inline:Nn \g__wrapblock_short_new_prop
      { \iow_now:Nn \@auxout { \wrapblock@short {#1} {#2} } }
  }
```

This chunk repairs, on the next run, a page that broke too early because of a footnote.

The problem. A footnote in the full-width lines takes room at the bottom of the page. That room is only known when the footnote is typeset, after the lines were counted. The page then breaks before the planned line, and breaks again at the forced break: a page with a few lines is left before the image.

Noticing it. The `build/column/before` hook runs each time a page is finished, and `\outputpenalty` is then the penalty of the break. While an image is waiting, a value other than $-10\,000$ means that the page broke elsewhere than planned.

Measuring the room. The room the footnotes of the block took is the height of all the footnotes of the page (the box `\footins` and the glue above it) minus what the footnotes before the block had taken, `\vsize` minus `\g__wrapblock_goal_dim`. If nothing is found, one line is assumed. (`\footins` is the number of an insertion: the glue above the footnotes is read with the primitive `\skip`, not with `\skip_use:N`.)

Keeping it. The room is kept under the number of the block, and written to the `.aux` file at the end of the document, as `\wrapblock@short{`⟨*block*⟩`}{`⟨*room*⟩`}`. The next run reads it into `\g__wrapblock_short_dim`, and `\__wrapblock_defer_para:N` counts the lines without that room.

Not forgetting it. On that next run, the page holds fewer lines, and the footnote may now be below the break: no early break is noticed. If the value were dropped then, the run after would break too early again, and the runs would alternate for ever. A value found once is therefore written again as long as no new one replaces it.

The `\providecommand` written first lets the `.aux` file be read by a document that no longer loads the package.

```
\msg_new:nnn { wrapblock } { early }
  { The~page~broke~too~early~in~the~text~of~the~block~that~ends~on~
    input~line~\msg_line_number:,~before~its~image~was~set:~a~footnote~in~the~
    full-width~lines~shortened~the~page,~and~a~page~with~a~few~lines~is~
    left~before~the~image.~Rerun~LaTeX~to~get~it~right. }
\msg_new:nnn { wrapblock } { alone }
  { The~text~of~the~block~that~ends~on~input~line~\msg_line_number:\ ended~before~
    the~bottom~of~the~page:~its~image~is~set~after~it. }
```

The two warnings of the deferred image: `early` for a page that broke too early (above), `alone` for a text that ended before the page broke (`\__wrapblock_end:`).

11.12 Boxes

```
% A \parbox, minipage or p cell is not wrapped: its text keeps the width
% it was given. Say so when that width does not fit beside the image.
\cs_new_protected:Npn \__wrapblock_box_check:
  {
    \bool_lazy_and:nnT { \l__wrapblock_outer_bool } { \g__wrapblock_active_bool }
      {
        % group 11: a footnote, set apart at the foot of the page;
        % \@captype: a float, which goes elsewhere on the page
        \bool_lazy_all:nT
          {
            { ! \g__wrapblock_pending_bool }
            { ! \g__wrapblock_wait_bool }
            { \int_compare_p:n { \tex_currentgrouptype:D != 11 } }
            { ! \cs_if_exist_p:N \@captype }
            { \dim_compare_p:nNn { \pagetotal } < { \g__wrapblock_bottom_dim } }
          }
          {
            \bool_lazy_and:nnT
              { \dim_compare_p:nNn { \hsize } > { \g__wrapblock_avail_dim + 0.5pt } }
              { ! \str_if_eq_p:ee { \msg_line_number: } { \g__wrapblock_warned_tl } }
              {
                \tl_gset:Ne \g__wrapblock_warned_tl { \msg_line_number: }
                \msg_warning:nnxx { wrapblock } { box }
                  { \dim_use:N \hsize } { \dim_use:N \g__wrapblock_avail_dim }
              }
          }
      }
  }
\msg_new:nnn { wrapblock } { box }
  { A~box~of~width~#1,~in~the~block~that~ends~on~input~line~
    \msg_line_number:,~sits~beside~the~image,~where~only~#2~is~left.~
    The~text~of~a~box~is~not~wrapped. }
```

The text of a `\parbox`, a `minipage` or a `p` cell is typeset with the width it was given: the package cannot narrow it. `\__wrapblock_box_check:` runs at the start of every such box, through `\@arrayparboxrestore`, where `\hsize` is already the width of the box. It warns when the box is wider than the room left beside the image, once per block: the input line it compares is that of the end of the block, the same for all its boxes. It stays silent when the box cannot be beside the image:

- before the image is placed, and while a deferred image waits;
- below the image (`\pagetotal` is past `\g__wrapblock_bottom_dim`);
- in a footnote, whose text is typeset in an insertion (group type 11) and goes to the foot of the page;
- in a float (`\@captype` is defined there), which goes elsewhere on the page.

```
% A tcolorbox reads \linewidth for its width before it starts its line,
% between two paragraphs, where \linewidth is the full line. One that
% starts beside the image is given the width left there, which its line
% gets. The width is set in the group of the box, and ends with it.
\cs_new_protected:Npn \__wrapblock_box_width:
  {
    % A box right after a paragraph, with no blank line between them, comes
```

```
    % while that paragraph is still open: end it first, so that the zone
    % knows where it stands (a deferred image is committed at its end).
    % Otherwise a key such as center, which fixes the width when it is
    % read, would keep the full line.
    \bool_lazy_all:nT
      {
        { \l__wrapblock_outer_bool }
        { \g__wrapblock_active_bool }
        { \mode_if_horizontal_p: }
      }
      { \par }
    % A box that opens the block: it starts its own paragraph after a
    % \nointerlineskip, on a line whose height is not that of a strut (an
    % empty line for a breakable box, the box itself otherwise), and the
    % image hung on it would go up or down. The image gets a line of its
    % own first, placed as a first line of text, and the box comes after,
    % with the height of a strut given back.
    \bool_lazy_all:nT
      {
        { \l__wrapblock_outer_bool }
        { \g__wrapblock_active_bool }
        { \g__wrapblock_pending_bool }
        { ! \g__wrapblock_wait_bool }
        { \mode_if_vertical_p: }
        { ! \mode_if_inner_p: }
      }
      {
        \noindent \par \nobreak
        \tex_kern:D - \box_ht:N \strutbox
      }
    \bool_lazy_all:nT
      {
        { \l__wrapblock_outer_bool }
        { \g__wrapblock_active_bool }
        { ! \g__wrapblock_wait_bool }
        {
          \bool_lazy_or_p:nn { \g__wrapblock_pending_bool }
            { \dim_compare_p:nNn { \pagetotal } < { \g__wrapblock_bottom_dim } }
        }
      }
      {
        \str_if_eq:VnTF \g__wrapblock_side_tl { l }
          { \dim_sub:Nn \linewidth { \g__wrapblock_shift_dim } }
          {
            \dim_set:Nn \linewidth
              {
                \dim_min:nn { \linewidth }
                  { \g__wrapblock_left_dim + \g__wrapblock_width_dim
                    - \g__wrapblock_shift_dim - \@totalleftmargin }
              }
          }
      }
  }
\hook_gput_code:nnn { cmd / tcolorbox / before } { wrapblock }
  { \__wrapblock_box_width: }
```

A `tcolorbox` computes its width from `\linewidth` when it starts, between two paragraphs, before it starts the line it sits on: `\linewidth` is then the full line, and the box would overlap the image even on a narrowed line. `\__wrapblock_box_width:` runs before `\tcolorbox`, through the hook `cmd/tcolorbox/before`, inside the group of the box. A box that comes right after a paragraph, with no blank line between them, finds that paragraph still open; `\__wrapblock_box_width:` ends it first, so that the zone knows where it stands — a deferred image is committed at the end of that paragraph — since a key such as `center` fixes the width of the box as soon as

it is read. A box that opens the block starts its own paragraph, after a `\nointerlineskip`: the image, hung on its first line as on a line of the height of a strut, would go up into the line before when the box is breakable (the line is empty, of height zero), and down below the box when it is not (the line is the box). `\__wrapblock_box_width:` then gives the image a line of its own first, an empty paragraph placed like a first line of text, takes the height of a strut back with a `\kern` (not a skip, which the `\addvspace` of the box would replace), and lets the box come after. When the box starts beside the image — the image is still to be placed, or `\pagetotal` has not reached `\g__wrapblock_bottom_dim` — `\linewidth` becomes the width of the narrowed line, computed as in `\__wrapblock_shape:Nnnn`. The line of the box is then narrowed like any other, and the box fits on it. Other boxes need no such help: a `minipage` or a `\parbox` is given its width by the author.

```
% In a box (minipage, \parbox, tcolorbox) \pagetotal says nothing of the
% position, and no page can break anyway: the text goes in a minipage and
% the image beside it, aligned on the first line.
\cs_new_protected:Npn \__wrapblock_inbox_image:
  {
    \__wrapblock_build:
    \box_move_down:nn
      { \box_ht:N \g__wrapblock_img_box - \box_ht:N \strutbox }
      { \box_use:N \g__wrapblock_img_box }
  }
\cs_new_protected:Npn \__wrapblock_inbox_begin:
  {
    \msg_info:nn { wrapblock } { inbox }
    \noindent
    \str_if_eq:VnT \l__wrapblock_side_tl { l }
      { \__wrapblock_inbox_image: \hfill }
    \begin{minipage}[t]{ \dim_eval:n { \linewidth - \g__wrapblock_shift_dim } }
  }
\cs_new_protected:Npn \__wrapblock_inbox_end:
  {
    \end{minipage}
    \str_if_eq:VnT \l__wrapblock_side_tl { r }
      { \hfill \__wrapblock_inbox_image: }
    \par
  }
\msg_new:nnn { wrapblock } { inbox }
  { The~block~that~ends~on~input~line~\msg_line_number:\ is~inside~a~box:~its~text~
    is~set~beside~the~image,~without~wrapping~under~it. }
```

A block that is itself in a box cannot use any of the above: in a box, `\pagetotal` says nothing of the position, and no page can break anyway. The block is then made of two boxes side by side: the text in a `minipage` as wide as the line minus `\g__wrapblock_shift_dim`, and the image, lowered so that its top is at the top of the first line. The text does not run below the image. An information message in the `.log` says so.