The `bigfoot` package
version 2.1

David Kastrup*

2015/08/30

Purpose of this package is to provide a one-stop solution to almost all problems related to footnotes. You can use it as a drop-in replacement of the `manyfoot` package, but without many of its shortcomings, and quite a few features of its own. It uses the existing document class layouts for footnotes, so you can usually use it without having to worry about the looks.

Features are:

- You can specify and use multiple footnote apparatus. Footnotes for an apparatus lower on the page[a] can be anchored in an apparatus[1] that is higher on the page.
- The last footnote in each apparatus may be broken to the next page.[2] Any subordinate footnote anchors that get moved to the next page will take the corresponding footnote with them.
- The order of footnotes in an apparatus is 'natural': it starts with any footnote that may have been broken from the next page, followed by footnotes from the current page in the order of the appearance of their footnote marks.[b] Where the order of appearance in the document differs from the order in the source code, you will usually want to use the `\MakeSorted` command from the `perpage` package to get the numbering fixed appropriately.

*dak@gnu.org [1]The plural of "apparatus" is actually "apparatus"[c]

[2]This will probably be interesting for footnotes that contain stuff like math equations[d] or lists[e].

[a]like this one [b]This footnote appears above notes on notes.

[c]Well, actually "apparatūs" with a long "u", but that's just obvious in spoken Latin.

[d]Like

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6} \tag{1}$$

[e]Like

- This, or
- this.

- Footnotes can be formatted in separate paragraphs, or be run into a single paragraph. The choice is made per footnote apparatus, but can be overridden for single footnotes.[3]
- If footnotes are run into one paragraph, a variety of criteria makes sure that this formatting is only chosen when it saves noticeable space and delivers visually attractive results.
- Parameters for footnote formatting can be specified globally, or separately for each footnote.
- The material in footnotes can contain `\verb`-like material without problems.[4]
- You can use color in footnotes. If a footnote gets broken across pages, the color at the point of the break will get resumed on the next page. Actually, the whole color stack will get reinstated.

As an example of how simple the usage can be, here is the documentation driver for this document:

```
⟨*driver⟩
\documentclass{ltxdoc}
\usepackage{bigfoot}
\usepackage{tabularx}
\usepackage{hyperref}
```

After loading the packages, we declare two footnote blocks. One is the default footnote block, another block is called `B` and is numbered with letters. The letters start new on each page. Both footnote blocks default to in-paragraph footnotes. Since the block `B` can get entries from both the main text as well as the default footnote block, the entries are not necessarily generated in page order. So we need to use a sorted counter to fix this (feel free to try what happens when using an unsorted counter).

```
\DeclareNewFootnote[para]{default}
\DeclareNewFootnote[para]{B}[alph]
\MakeSortedPerPage{footnoteB}
```

In addition, we add an alternate footnote sequence that can be interspersed with the normal footnotes by use of the `\footnote'` command which we effectively define here.

```
\newcounter{footalt}
\def\thefootalt{\fnsymbol{footalt}}
\MakeSortedPerPage[2]{footalt}
\WithSuffix\def\footnotedefault'{\refstepcounter{footalt}%
  \Footnotedefault{\thefootalt}}
```

[3] I.e., footnotes with display math[a] or list environments[b] have to be done in vertical mode.
[4] We wrote `|\verb|-like` above in the main text.[c]

[a] We had this already, right? [b] And this looks familiar, too.
[c] Well, this is not so impressive. But we wrote `\verb+|\verb|-like+` in the footnote then.

Actually, that already was all. We can now start the document. The following makes sure that we get the full documentation only by compiling the separate driver file:

```
\begin{document}
\OnlyDescription
⟨driver⟩ \AlsoImplementation
\DocInput{bigfoot.dtx}
\end{document}
⟨/driver⟩
```

In order to be useful without additional hassle, the normal footnote level will be called `default`. If no such style has been defined at the start of the document, it will get defined and used for ordinary footnotes, fixing quite a few problems of LaTeX's own footnote placement algorithms.

Apart from that, usage is very much like that of `manyfoot`, so for the customization possibilities of `bigfoot` with regards to multiple footnote blocks and rules between them, refer to `manyfoot`'s documentation.

`bigfoot` contains a lot of bells and whistles for defining your footnote formats and can use different formats for different footnote blocks. Those expert options are not documented separately yet: look through the code sections to see them explained.

1 The implementation

1.1 Startup code

We declare the package and several compatibility options supposed to make `bigfoot` a drop-in replacement for `manyfoot`.

```
⟨*style⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{bigfoot}[2015/08/30 2.1 makes footnotes work]

\DeclareOption{para}{\PackageInfo{bigfoot}{Compatibility option ‘para’
  has no effect:^^J%
  Spacing will be guessed from ‘\string\@makefntext’ unless^^J%
  ‘\string\@preparefnhtext’ is redefined}}

\DeclareOption{para*}{\PackageInfo{bigfoot}{Compatibility option
    ‘para*’:^^J%
    Redefining ‘\string\@preparefnhtext’}%
  \def\@preparefnhtext{\ifx\@thefnmark\@empty
    \else\@makefnmark\nobreak\fi}}

\DeclareOption{ruled}{\PassOptionsToPackage{ruled}{manyfoot}}
```

The normal processing makes footnote text macros allow verbatim and similar. We call this `robust` processing though it is not totally accurate. This is the default. There is also an option `fragile` which will not allow this, but may be required

for some definitions of `\@makefntext`. It turns out that most document classes work with the 'robust' option, but there might be some that fail, and there might be some footnote-modifying packages that also can cause failure.

```
\DeclareOption{robust}{\def\FN@makefncall{\FN@makefnrobust}}
\DeclareOption{fragile}{\def\FN@makefncall{\FN@makefnfragile}}
\ExecuteOptions{robust}
```

The `verbose` option talks about changed labels at the end of a run. It is for debugging instable configurations that fail to converge after a number of TEX runs. The output is probably obscure.

```
\DeclareOption{verbose}{\AtBeginDocument{%
    \def\@testdef #1#2#3{%
      \def\reserved@a{#3}%
      \expandafter \ifx \csname #1@#2\endcsname
        \reserved@a
      \else \@tempswatrue
        \typeout{Changed label #1/#2:
          \csname #1@#2\endcsname->#3}%
      \fi}}}
```

The `trace` option is only available if you used `docstrip` while explicitly requesting `trace` functionality. If you set the `trace` option, the next option specifies a bit map of trace bits.

The following bits can be set in tracing:

1	break decisions
2	horizontal box building
4	allocation stuff
8	output routine stuff
16	retained and kept boxes

```
⟨trace⟩\DeclareOption{trace}{%
⟨trace⟩\DeclareOption*{\ftflags=\CurrentOption\relax
⟨trace⟩   \DeclareOption*{\OptionNotUsed}}%
⟨trace⟩   \AtEndOfPackage{\RequirePackage{trace}\relax
⟨trace⟩   \errorcontextlines\maxdimen
⟨trace⟩   \showboxdepth4
⟨trace⟩   \showboxbreadth100
⟨trace⟩   \tracingonline=\@ne}%
⟨trace⟩}
```

The `tracepage` option is followed by another option specifying the page to be traced. If you use it, tracing happens only on the specified page. Only a single page can be specified.

```
⟨trace⟩\def\FN@tracepage{\c@page}
⟨trace⟩\DeclareOption{tracepage}{%
⟨trace⟩   \DeclareOption*{\edef\FN@tracepage{\CurrentOption}%
⟨trace⟩     \DeclareOption*{\OptionNotUsed}}}
⟨trace⟩\newcount\ftflags
⟨trace⟩\def\foottrace#1{\ifnum\numexpr(\ftflags+(#1))/(2*#1)*(2*#1)%
⟨trace⟩   =\numexpr(\ftflags+3*#1/2)/#1*#1\relax
⟨trace⟩   f\else t\fi\ifnum\FN@tracepage=\c@page t\else n\fi}
\ProcessOptions*
```

`hyperref`'s footnote support will just cause trouble. So if `hyperref` was already loaded or is going to be loaded, we turn off its footnote support. If you think you know what you are doing, you can use `\hypersetup` to turn it on again before the start of the document. Unfortunately, it appears like `\hypersetup` refuses to be called more than once, so this actually does not work unless you load `hyperref` last.

```
\ifx\hypersetup\@undefined
  \PassOptionsToPackage{hyperfootnotes=false}{hyperref}
\else
  \hypersetup{hyperfootnotes=false}
\fi
```

We require the `etex` package because

1. We need the facilities of the ε-TEX engine; and where they are not available, the error messages from not finding the `etex` package or from loading it into the wrong engine make much more sense than what would happen otherwise.

2. We allocate quite a few registers, and the danger of running out of them is smaller when the extra registers of ε-TEX are taken into account. Now unfortunately the LaTeX team has decided in 2015 to do its own extended allocation scheme incompatible with the `etex` package, so we need to guard against this load in case the new LaTeX allocation scheme is detected.

We need the `manyfoot` package to build on. The `suffix` and `perpage` package are needed for some small stuff.

```
\ifx\e@alloc\@undefined
\RequirePackage{etex}
\fi
\RequirePackage{manyfoot}
\RequirePackage{suffix}
\RequirePackage{perpage}
```

1.2 Fixes to the `manyfoot` package

While those fixes have been submitted once to the author of `manyfoot`, they have not made it into its distribution at the current point of time. In the interest of stability, it would probably be best just to incorporate the parts from `manyfoot` that get used by `bigfoot`. This has not yet been done.

`\MFL@reinsout` We need the appropriate splitting parameters set for the footnote again. `\MFL@realinsert` does that, but it has the disadvantage that it uses `\strutbox`, and that may be set to arbitrary values at the time the output routine is invoked. `manyfoot` already has this problem with minipages: the split sizes will be those of the font at the end of the minipage instead of those at the time the footnote body was set up. So we do this here, and see later for more info about how to do this right:

```
\def\MFL@reinsout#1#2{\ifvoid#2\else
    \ifnum\count\@currbox>\z@
```

```
      \advance\@pageht \ht#2%
      \advance\@pageht \skip#2%
      \advance\@pageht \dp#2%
    \fi
    \MFL@realinsert{#2}{\unvbox#2}%
  \fi
}
```

`\MFL@reins` Actually, I don't get the purpose of the following line in the first place. But if we do need it for some reason, it is rather certain that we don't want this empty insert to float. Use `\MFL@realinsert`, or set the floatingpenalty the hard way.

```
\def\MFL@reins#1#2{\ifvoid#2\else\insert#2{\floatingpenalty\@MM}\fi}
```

`\MFL@mpinsert` `\MFL@minipage` The structure of the `\MFL@mpinsert` box is overly complicated, and it is a bad idea to unpack the boxes put into it too early: the `\lastbox` command is pretty inefficient when the list before it is long due to unpacking. So we just leave everything packed in its own boxes, and unpack only at the moment when we are reinserting.

```
\long\def\MFL@mpinsert#1#2{%
  \global\setbox#1\vbox{%
    \unvbox#1%
    \nointerlineskip
    \vbox{#2}%
  }%
}

\def\FN@divert{%
  \let\MFL@mpinsertsave\MFL@insert
  \MFL@reinsert \let\MFL@insert\MFL@mpinsert}
\def\FN@enddivert{{\let\@elt\MFL@mpreinsert \MFL@list}}

\def\MFL@minipage{\ifinner\else \FN@divert\fi}
\def\MFL@endminipage{\ifinner\else \FN@enddivert\fi}
```

`\MFL@mpreinsert` When reinserting, we put all but the last insertion into one humongous blob. This is so that the last insertion can be split by TEX's paragraph splitting routines. The footnote types that `bigfoot` supports will never get split by TEX, anyhow, but it is conceivable that other extension packages for manyfoot work differently. There is one difference, though: we let a slave mark escape into the main vertical list.

```
\def\MFL@mpreinsert#1#2{%
  \ifvoid#2\else
    \setbox\@tempboxa\vbox\bgroup\unvbox#2%
      \global\setbox#2\lastbox
      \setbox\z@\lastbox
      \ifvoid\z@
        \egroup
        \setbox\z@\box#2%
```

```
      \else
        \MFL@removevboxes \unvbox\z@
        \egroup
        \setbox\z@\box#2%
        \MFL@mpinsertsave#2{\unvbox\@tempboxa}%
      \fi
    \ifvoid\z@\else
      \MFL@mpinsertsave#2{\unvbox\z@}%
    \fi
    \marks\FN@slave{\number\FN@id}%
  \fi}
```

`\MFL@removevboxes` This trick works like `\removehboxes` in the TEXbook's appendix D.

```
\def\MFL@removevboxes{{\setbox\z@\lastbox
    \ifvbox\z@ \MFL@removevboxes \unvbox\z@\fi}}
```

`\NCC@makefnmark` This provides the command in case it is not present (some versions did not have it).

```
\ifx\NCC@makemark\@undefined
  \ifx\NCC@makefnmark\@undefined \else
    \def\NCC@makemark{\NCC@makefnmark}
  \fi
\fi
```

While the above operations actually were fixes to `manyfoot`, now we actually patch it for our own purposes. When allocating a new footnote, we set its maximum dimension to `\maxdimen` (since no hard limit makes sense, given that we recalculate all respective sizes at output time) and allocate a cache box to go with it. We also add the insertion to the list of insertions in `\FN@nestlist`.

`\MFL@startplain` `\MFL@startpara`

```
\def\MFL@startplain#1{\global\dimen#1\maxdimen
  \@cons\FN@nestlist{{}#1}%
  \expandafter\expandafter\expandafter\newbox\FN@cache#1}

\let\MFL@startpara\MFL@startplain
```

`\RestyleFootnote` This macro gets two arguments: a footnote ⟨*type*⟩, and the style to be used for it. It works by redefining the corresponding `Footnotetext`⟨*type*⟩ macro.

```
\def\RestyleFootnote#1#2{\expandafter\xdef
  \csname Footnotetext#1\endcsname{\expandafter
    \noexpand\csname MFL@fnote#2\endcsname{\csname footins#1\endcsname}}}
```

`\FN@stripfootins` `\FN@restylefootnote` We need the same kind of functionality for a footnote specified by its footnote insertion. So we strip the footnote ⟨*type*⟩ from the insertion macro name. Kind of ugly.

```
\expandafter\def\expandafter\FN@stripfootins\string\footins{}

\def\FN@restylefootnote#1#2{{\edef\next{%
```

```
      \noexpand\RestyleFootnote{\expandafter\FN@stripfootins
        \string#1}{#2}}\next}}
```

1.3 Dealing with footnote-specific code

The formatting of footnotes is determined by macros such as `\@makefntext`. For several blocks of footnotes, we might want to have several different ways for formatting them. Whenever this is the case, we call them with

`\FN@specific{`⟨*insert#*⟩`}{`⟨*macroname*⟩`}`

This will use the default ⟨*macroname*⟩ unless a special macro has been defined with something like

`\FootnoteSpecific\marg{type}`...

A number of other defining commands and constructs are available: those are pretty much like the ones for the `\WithSuffix` command implemented by the `suffix` package.

`\FN@specific` We use `\romannumeral` here just for the purpose of sustaining expansion. It expands to nothing when followed by `\z@` eventually. Thus expanding the expansion of `\FN@specific` again delivers the (unexpanded) final token to use.

```
\def\FN@specific#1#2{\romannumeral
  \ifcsname FN\string#2\number#1\endcsname
  \expandafter
    \z@\csname FN\string#2\number#1\expandafter\endcsname
  \else\expandafter\z@
    \expandafter#2\fi}
```

`\FootnoteSpecific` `\FN@specific@ii` This is all a bit muddy, but quite similar to what the `suffix` package does, so you might want to look there for the explanation.

```
\def\FootnoteSpecific#1{\count@\csname footins#1\endcsname\toks@{}%
  \FN@specific@ii}

\long\def\FN@specific@ii#1#2{\toks@\expandafter{\the\toks@#1}%
  \the\expandafter\toks@
  \csname FN\string#2\number\count@\endcsname}

\WithSuffix\def\FN@specific@ii\long{\toks@\expandafter
  {\the\toks@\long}\FN@specific@ii}

\WithSuffix\def\FN@specific@ii\global{\toks@\expandafter
  {\the\toks@\global}\FN@specific@ii}

\WithSuffix\def\FN@specific@ii\expandafter{\expandafter
  \FN@specific@ii\expandafter}
```

1.4 Putting footnotes into insertions

1.4.1 Dealing with Ids

Since we have to store additional information for each footnote as long as it is not yet typeset, we allocate and deallocate numeric 'id's on an as-needed base, since we do not want to store this sort of information indefinitely, with a large toll on hash space. So we work with indirect ids, where the unique ids are just referenced indirectly. We do this with 'slots'.

`\FN@slotxdef` `\FN@slotget` New slots are assigned values with `\FN@slotxdef`, which can be retrieved again with `\FN@slotget`.

```
\def\FN@slotxdef#1{%
  \global\expandafter\xdef\csname FN@slot#1\endcsname}

\def\FN@slotget#1{\csname FN@slot#1\endcsname}
⟨trace⟩\def\FN@slotget#1{%
⟨trace⟩  \expandafter\FN@slotgetii\expandafter
⟨trace⟩    \FN@slotfreelist\expandafter
⟨trace⟩    {\number\number#1}}
⟨trace⟩\def\FN@slotgetii#1#2{%
⟨trace⟩  \ifx#1\@empty \csname FN@slot#2\endcsname\else
⟨trace⟩    \ifnum#1=#2 \errmessage{Use after freed: #1}\else
⟨trace⟩      \expandafter\FN@slotgetii
⟨trace⟩        \csname FN@slot#1\endcsname{#2}\fi\fi}
```

`\FN@slotfreelist` `\FN@nextslot` `\FN@slotfreelist` point to the first already allocated available id to be reused. If it is empty, none exist. In that case, `\N@nextslot` contains the next slot number to use.

```
\def\FN@slotfreelist{}
\def\FN@nextslot{1}
```

`\FN@newslot` This allocates a new slot by setting the given macro to a currently unused slot number in decimal form. If there is something left in the freelist, it is assigned, otherwise a new slot gets allocated.

```
\def\FN@newslot#1{%
  \ifx\FN@slotfreelist\@empty
    \edef#1{\FN@nextslot}%
    \xdef\FN@nextslot{\number\numexpr \FN@nextslot+\@ne}%
  \else
    \let#1\FN@slotfreelist
    \xdef\FN@slotfreelist{\csname FN@slot\FN@slotfreelist\endcsname}%
  \fi
⟨trace⟩  \if\foottrace4\message{^^JAllocated #1^^J}\fi
}
```

`\FN@freeslot` This frees a given slot (by number) again by adding it to the freelist.

```
\def\FN@freeslot#1{%
⟨trace⟩  \if\foottrace4\message{^^JFreeing #1^^J}\fi
```

```
  \global\expandafter\let\csname FN@slot#1\endcsname=\FN@slotfreelist
  \xdef\FN@slotfreelist{#1}}
```

1.4.2 Dealing with footnote stacks

Footnote stacks are used for paired footnotes that refer to a text range instead of a single text point. For example, you can use something like

```
Text \var<{was there}is here\var>
```

To have a text variant "was there" for the original passage "is here", and mark it, say, as "[a]is here[a]" by employing the `suffix` package suitably. This would anchor the footnote at the start of the passage. It would also be imaginable to implement the syntax

```
Text \var<is here\var>{was there}
```

for anchoring it at the end of the given passage.

```
\global\let\FN@stacklist\@empty
```

`\DefineFootnoteStack` This command is used for defining a footnote stack. It gets a single argument which is the name of the stack and should consist just of ordinary character tokens.

```
\def\DefineFootnoteStack#1{%
  \global\expandafter\let\csname FN@stack@#1\endcsname\@empty
  \@cons\FN@stacklist{{#1}}%
}
```

At the end of the document, all stacks are checked to make sure they have been used up completely.

```
\AtEndDocument{\FN@checkstacklist}

\def\FN@checkstacklist{{\let\@elt\FN@checkstack
  \FN@stacklist}}

\def\FN@checkstack#1{{\let\@elt\FN@checkstackentry
    \csname FN@stack#1@\endcsname}}

\def\FN@checkstackentry#1#2#3{%
  \PackageError{bigfoot}{Unfinished #1 #2 from line #3}%
  {The specified footnote range is uncomplete}}
```

`\PushFootnoteMark` This gets one argument, the name of the footnote stack. It pushes the current footnote mark name stored in `\@thefnmark` onto the footnote stack.

```
\def\PushFootnoteMark#1{{\let\@elt\relax
  \expandafter\unrestored@protected@xdef \csname FN@stack@#1\endcsname
  {\@elt{#1}{\@thefnmark}{\number\inputlineno}\csname
  FN@stack@#1\endcsname}}}
```

\PopFootnoteMark This gets one argument, the name of the footnote stack. It pops the value of \@thefnmark from the named footnote stack.

```
\def\PopFootnoteMark#1{\expandafter
  \ifx\csname FN@stack@#1\endcsname\@empty
    \PackageError{bigfoot}{Empty footnote stack #1}%
    {The specified footnote type has no uncompleted range}%
  \else
  {\let\@elt\FN@firstpop
    \iffalse{\fi\csname FN@stack@#1\endcsname}}\fi}

\def\FN@firstpop#1#2#3{\protected@xdef\@thefnmark{#2}%
  \let\@elt\relax
  \expandafter\protected@xdef\csname FN@stack@#1\endcsname{%
    \iffalse}\fi}
```

1.4.3 Continuation marks

We add a possibility of adding continuation marks. While the box is assembled, immediately before the break, \FN@beforebreak gets called, and \FN@afterbreak is called at the top of the continuing box.

```
\ifx\FN@beforebreak\@undefined
  \let\FN@beforebreak\@empty
\fi
\ifx\FN@afterbreak\@undefined
  \let\FN@afterbreak\@empty
\fi
```

1.4.4 The works

\FN@cache Cacheboxes cache the typeset forms of the insertion boxes for a certain configuration of footnotes.

```
\def\FN@cache#1{\csname FN@cache\number#1\endcsname}
```

\FN@sortlist takes the current vertical list and sorts the contained boxes according to their width (which is supposed to contain the sort key).

The algorithm is a pretty straightforward insertion sort with $O(n^2)$ steps. This is the best one can hope for without comparisons across non-adjacent list elements. For presorted lists, the performance will be $O(n)$, and that's what we expect to see for simple cases (and when there are no sortkeys yet). Any negative width will certainly hang the algorithm.

It also happens that TEX has a hardwired limit for grouping levels that hits at 255. Oops. We better not have a few hundred footnotes in a single block on one page...

```
\def\FN@sortlist{{%
  \setbox\z@\lastbox
  \ifvoid\z@ \else \FN@sortlist\FN@sortlistii \fi}}

\def\FN@sortlistii{%
```

```
  \setbox\tw@\lastbox
  \ifvoid\tw@\else
  \ifdim\wd\tw@<\wd\z@ {\FN@sortlistii}%
  \fi\nointerlineskip\box\tw@\fi\nointerlineskip\box\z@}
```

`\FN@sortinsert` This function is an `\@elt` function that will sort the given insertion if it is non-empty and if there is no cache box present (which would imply that the insertion had already been sorted previously).

```
\def\FN@sortinsert#1#2{\ifvoid\FN@cache#2%
  \ifvoid#2\else\global\setbox#2\vbox{\unvbox#2%
    \FN@sortlist}\fi\fi}
```

`\FN@maybeinvalidatecache` This is called after pulling in additional material from the page. If the material added an insertion, the cache is junk and must be regenerated.

```
\def\FN@maybeinvalidatecache#1#2{%
  \ifvoid#2\else\global\setbox\FN@cache#2=\box\voidb@x\fi}
```

`\FN@regeneratecache` This unconditionally regenerates one cache box. The structure of a cache box is basically a list of vertical boxes. All but the last such box are packed into a single vertical box which is then followed by the last vertical box.

```
\def\FN@regeneratecache#1#2{%
   \global\setbox\FN@cache#2=%
   \ifvoid#2%
     \box\voidb@x
   \else
     \vbox{\vbox{\unvcopy#2%
         \setbox\z@\lastbox
         \def\FN@masterinsert{#2}%
         \FN@assembleboxes
         \global\setbox\FN@cache#2\box\z@}%
       \nointerlineskip \box\FN@cache#2}%
    \fi}
```

`\FN@mayberegeneratecache` This regenerates the cache in case the cache box has been voided in order to mark it as invalid.

```
\def\FN@mayberegeneratecache#1#2{%
  \ifvoid\FN@cache#2%
    \FN@regeneratecache{}#2%
  \fi}
```

`\FN@cachesize` This calculates the size impact of a cache box on the current page as a term to be added into a `\glueexpr`-type of expression.

```
\def\FN@cachesize#1#2{%
  \ifvoid\FN@cache#2%
  \else
    +\skip#2+(\ht\FN@cache#2+\dp\FN@cache#2)*\count#2/\@m
  \fi}
```

`\FN@clearcache` This just completely voids a cache register.

```
\def\FN@clearcache#1#2{%
  \global\setbox\FN@cache#2=\box\voidb@x}
```

`\@makefnvtext` Ok, this is one of the parts putting together footnotes in para mode. The footnotes themselves have already been formatted into hboxes (placed there with `\@preparefnhtext` in order to cater for proper indentation). `\@makefnvtext` then formats a single footnote block from horizontal mode pieces (vertical mode pieces are kept as-is). This takes text and typesets it in a single block. To get correct indentation, it breaks before the first footnote and adjusts the clubpenalties to move them to one line lower effectively. `\@makefnvtext` is called in vertical mode, and its argument is typeset in horizontal mode right after a break, inside of `\@makefntext`.

```
\def\@makefnvtext#1{%
  \FN@specific\FN@masterinsert\@makefntext{%
    \clubpenalties\thr@@\@MM\clubpenalty\z@
    \vadjust{\nobreak\vskip-\baselineskip}\nobreak\hfill\break#1}}
```

`\@preparefnhtext` This creates appropriate skips to be put before the horizontal material to make the indentation correct with a breakpoint before the footnote as well as when in run-in text. This is run once at the start of each horizontal mode footnote when it is first being typeset, in horizontal mode.

```
\ifx\@preparefnhtext\@undefined
\def\@preparefnhtext{{%
  \setbox\z@\vbox{\FN@specific\FN@masterinsert\@makefntext{%
      \unskip\unpenalty\setbox\z@\lastbox
      \dimen@
        \ifnum\parshape>\z@
          \dimexpr\parshapeindent\tw@-\parshapeindent\@ne\relax
        \else \ifnum\hangafter=\@ne\hangindent \else
          \ifnum\hangafter=\m@ne -\hangindent
          \else \z@ \fi\fi\fi
      \dimen@ii\dimen@
      \ifhbox\z@ \advance\dimen@-\wd\z@
        \setbox\z@\hbox{\unhbox\z@}%
        \advance\dimen@\wd\z@
      \fi
      \xdef\FN@tempinfo{\hskip\the\dimen@
        \vadjust{}\nobreak\hskip-\the\dimen@ii\relax}}}%
  \FN@tempinfo}
\fi
```

Now we have in `\FN@tempinfo` the excess width of the label we don't want to preserve when doing in-paragraph footnote setting. A sequence of glue before a label now has to consist of stuff that vanishes at a breakpoint, followed by stuff that remains. We have to have two behaviors for the contents: behavior one is justification at the start of a line, behavior two is justification in the line. When we are at the start of the line, preceding interword space disappears swallowed

and so the natural criterion for distinguishing those cases is this initial line break. This means that we can't avoid articifially adding a line break at the start of such a box. We will back up its height again. Some packages specify a `\hangindent` (I know of no examples where they would actually set `\hangafter` to a value different from its default of 1, or set `\hangindent` to a negative value which would affect the right margin): due to our artifical line at the top, the indent will actually be active for the first line already. We back it out of the actual labels happening at the start of the line. Two-line parshapes have the same effect: the first line is not actually used, and we put the relevant info for the first line into the label. Different right indentation for the first line is something we can't simulate, but again, it should occur rarely. When `\parshape` is active, `\hangindent` is ignored.

```
\def\@makefnstartbox{%
  \ifdefined\setspace@singlespace
    \def\baselinestretch{\setspace@singlespace}%
  \fi
  \reset@font\footnotesize
  \hsize\MFL@columnwidth \@parboxrestore
  \interlinepenalty\FN@specific\FN@masterinsert\interfootnotelinepenalty
  \widowpenalty\FN@specific\FN@masterinsert\footnotewidowpenalty
  \clubpenalty\FN@specific\FN@masterinsert\footnoteclubpenalty
  \advance\linepenalty500\relax}

\def\@makefnendbox{%
  \widowpenalty\FN@specific\FN@masterinsert\finalfootnotewidowpenalty}

\newcount\footnotewidowpenalty
\footnotewidowpenalty=250
\newcount\footnoteclubpenalty
\footnoteclubpenalty=250
\newcount\finalfootnotewidowpenalty
\finalfootnotewidowpenalty=4000
```

`\@makefnvbox` This is the formatting code for a vertical mode footnote box from already set horizontal material. It uses `\@makefnstartbox` for setting up the initial widow/club penalties, and `\@makefnendbox` for preparing the final end. It results in a vbox.

```
\ifx\@makefnvbox\@undefined
  \def\@makefnvbox#1{\vbox{%
      \@makefnstartbox
      \clubpenalties\thr@@\@MM\clubpenalty\z@
      \let\@thefnmark\@empty
      \FN@specific\FN@masterinsert\@makefntext{\rule\z@\footnotesep
        \nobreak
        #1\@finalstrut\strutbox
      \@makefnendbox}}}
\fi
```

`\hfootfraction` `\vtypefraction` Those parameters govern when a footnote block is going to be set completely in vertical mode. If a footnote block does not shrink to less than `\hfootfraction`

its size when using in-paragraph notes or has at least `\vtypefraction` of forcedly vertical footnotes (specified as purely vertical, or vertical because of being large), it is set entirely in vertical mode.

```
\def\hfootfraction{0.9}
\def\vtypefraction{0.7}
```

`\FN@assembleboxes` This will produce the finished product, by generating all boxes and concatenating them except for the last vbox. It is assumed that have already set `\box\z@` to `\lastbox` before calling this routine (or, more likely, have already assembled and split the last box). The last, not yet unpacked `\vbox` is left in `\box\z@` on return. The original id of the last box of a block is properly transferred to it.

The last box might have come about by joining several horizontal boxes, so splitting it might separate footnotes. We deal with that problem at a different point of time by checking the respective Ids when breaking a vbox into pieces: if the split piece does not contain the last footnote beginning, we switch to a slow motion decomposal. `\FN@assembleboxes` is supposed to be entered and exited in vertical mode.

```
\def\FN@assembleboxes{%
⟨trace⟩ \ifhmode \PackageError{bigfoot}{Unexpected hmode}{}\fi
  \ifhbox\z@
    \dimen@\dp\z@
⟨trace⟩ \MFL@checksinglebox\z@\z@{}{}%
    \dimen@ii\z@
    \setbox\tw@\box\voidb@x
    \loop \advance\dimen@ii\dimexpr\ht\z@+\dp\z@\relax
      \setbox\tw@\hbox{\box\z@\unhbox\tw@}%
      \setbox\z@\lastbox
    \ifhbox\z@
    \repeat
    {\FN@assembleboxes\nointerlineskip\unvbox\z@}%
```

At this point of time, `\box\tw@` contains a plain hbox with nothing but the unadorned hboxes in horizontal mode to be joined into one footnote block. All preceding footnote blocks have been emptied into the current vertical list. We put the `\unvbox` operations in a group so that the paragraph shapes will not get reset over the break.

```
    \global\setbox\FN@tempbox\copy\tw@
    \setbox\z@\@makefnvbox{%
      {\unhbox\FN@tempbox}%
      \setbox\z@\lastbox\FN@joinhboxes}%
    \ifcase
      \ifdim\FN@vfound>\dimexpr\vtypefraction\p@*\FN@found\relax \@ne\fi
      \ifdim\dimexpr \ht\z@+\dp\z@>\hfootfraction\dimen@ii \@ne\fi \z@
    \or
      \global\setbox\FN@tempbox\box\tw@
      \setbox\z@\@makefnvbox{\let\@makefnbreak\FN@pseudofillbreak
          {\unhbox\FN@tempbox}\setbox\z@\lastbox\FN@joinhboxes}%
    \fi
```

```
    \setbox\tw@\box\voidb@x
    \ht\z@\dimexpr \ht\z@+\dp\z@-\dimen@\relax
    \dp\z@\dimen@
⟨trace⟩  \MFL@checksinglebox\z@\z@{}{}%
  \else
    \ifvbox\z@
⟨trace⟩  \MFL@checksinglebox\z@\z@{}{}%
      {\setbox\z@\lastbox
      \FN@assembleboxes\nointerlineskip\unvbox\z@}%
    \fi
  \fi}
```

Ok, now follow a lot of fuzzy calculation routines. When we are considering truth values, `\p@` (1pt) corresponds to a value of "true", and `\z@` corresponds to "false".

`\FN@fuzzyeval` This calculates a ratio, something with which you multiply. The first two arguments of the function define an interval, and the third argument is a value in that interval. If `#3` is equal to `#1`, the resulting ratio is 0, if the `#3` is equal to `#2`, the resulting ratio is 1. Values in between are linearly interpolated. Values outside of the interval are mapped to 0 and 1. If all three values are equal (hardly useful), 1 is returned.

```
\def\FN@fuzzyeval#1#2#3{%
  \ifdim\dimexpr(#3)<\dimexpr(#2)\relax
    \ifdim\dimexpr(#3)>\dimexpr(#1)\relax
      *(\dimexpr(#3)-(#1))%
      /(\dimexpr(#2)-(#1))%
    \else *\z@
    \fi
  \fi}
```

`\FN@fuzzyor` This returns probabilistic OR:

$$(\#1 + \#2 - \#1 \cdot \#2)$$

```
\def\FN@fuzzyor#1#2{(\p@-(\p@-(#1))*(\dimexpr\p@-(#2))/\p@)}
```

`\FN@magicclue` Ok, so here is the magic glue calculator. `#1` and `#2` give the range over which the preceding line changes from 'short' to 'long'. `#3` and `#4` give the range over which the current line changes from 'short' to 'long'. Both are combined with a probabilistic or function, and then a penalty is chosen which ranges from `#5` to `#6` for short to long.

```
\def\FN@magicglue#1#2#3#4#5#6{%
⟨trace⟩  \if\foottrace2\traceon\fi
  \dimen@\dimexpr\p@\FN@fuzzyeval{#1}{#2}\FN@lasthsize\relax
  \dimen@ii\dimexpr\p@\FN@fuzzyeval{#3}{#4}{\ht\z@+\dp\z@}\relax
  \dimen@\dimexpr\FN@fuzzyor\dimen@\dimen@ii
  \count@\numexpr((#6)-(#5))*\dimen@/\p@+(#5)\relax
  \xdef\FN@vfound{\the\dimexpr\FN@vfound+\dimen@}%
  \ifnum\count@>-\@M
```

```
    \penalty\count@
    \hskip\glueexpr -\parfillskip+1em minus 0.5em\relax
  \else
    \FN@pseudobreak
  \fi
  \xdef\FN@found{\number\numexpr\FN@found+\@ne}%
}
```

`\FN@pseudobreak` This ends a line, but without introducing parskip and similar. It also 'breaks in' the next line to get proper indentation. The main difference with regard to `\break` is that this restarts the reckoning of line numbers for the sake of `\clubpenalty` calculation.

```
\def\FN@pseudobreak{%
  {\parskip\z@skip\parfillskip\z@skip\parindent\z@\vadjust{}\par\noindent
    \vadjust{\nobreak\vskip-\baselineskip}\nobreak\hfill\break}}
```

`\FN@pseudofillbreak` This is basically just for separating paragraphs by force.

```
\def\FN@pseudofillbreak{\nobreak\hskip\parfillskip\FN@pseudobreak}
```

`\@makefnbreak` This calculates the glue for the standard horizontal footnotes.

```
\def\@makefnbreak{\FN@magicglue {\footnotesep+\dp\strutbox}%
  {\footnotesep+\dp\strutbox+\baselineskip}%
  {\footnotesep+\dp\strutbox+0.5\baselineskip}%
  {\footnotesep+\dp\strutbox+2\baselineskip}{-200}{-12000}}
```

`\FN@joinhboxes` is called with box 0 set to the next box to be appended to the current list (all preceding hboxes on the current vertical list will have to go in front). `\FN@joinhboxes` is entered in vertical mode, and will be exited in horizontal mode.

```
\def\FN@joinhboxes{%
⟨trace⟩ \ifvmode \errmessage{Unexpected vertical mode.}\fi
  \begingroup\setbox\z@\lastbox
    \ifhbox\z@ \FN@joinhboxes
⟨trace⟩ \ifvmode \errmessage{Unexpected vertical mode.}\fi
      \endgroup
      \nobreak\hskip\parfillskip
      \@makefnbreak
    \else
⟨trace⟩ \ifvbox\z@ \errmessage{Unexpected vbox.}\fi
      \endgroup
      \vadjust{\nobreak\vskip-\baselineskip}\nobreak\hfill\break
      \xdef\FN@vfound{\z@}%
      \xdef\FN@found{\z@}%
    \fi
  \xdef\FN@lasthsize{\the\dimexpr \ht\z@ +\dp\z@}%
  \unhbox\z@}
```

`\FN@par` `\FN@noindent` `\FN@indent` In-paragraph footnotes are collected in horizontal mode. So `\par`, `\noindent` and `\indent` simply don't work. We replace them with something having the same

effect when the boxes get unhboxed. Note that this does not admit the tracking of club/widow penalties: in a later version, it should get replaced by something that actually allows for separate paragraphs. One possibility would be to replace the current single hbox for an in-paragraph footnote by an hbox of hboxes and unbox all of them in separate paragraphs. But that glosses over the fact that a multi-paragraph footnote does not make sense in anything but vertical mode. So a saner way would probably be to close off the hbox altogether and reinsert it into a vbox, restarting the whole footnote in vertical mode. Both of those approaches would require that no groups have been opened since the start of the footnote by the time `\par` gets called. The below pseudosolution at least has the advantage of not depending on the grouping structure at all.

```
\def\FN@par{\unskip\nobreak\hskip\parfillskip
  \vadjust{\vskip\parskip}\break\null\kern\parindent\ignorespaces}
\def\FN@noindent{\unkern}
\def\FN@indent{\unkern{\setbox\z@\null\wd\z@\parindent\box\z@}}
```

`\MFL@fnoteplain` `\MFL@fnotepara` We redefine `manyfoot`'s basic footnote calls to use our own, versatile variant.

```
\def\MFL@fnoteplain{\FN@fnotenested{plain}}
\def\MFL@fnotepara{\FN@fnotenested{para}}
```

`\FN@fnotenested` This is somewhat contorted: we want `\footnote+` to be in `plain` style and `\footnote-` in `para` style regardless of the current footnote style. Adding a second `+` or `-` after the first will actually restyle all footnotes coming afterwards appropriately. This should work for all footnote commands getting footnote text.

```
\def\FN@fnotenested#1#2#3{%
  \edef\reserved@d{#1}%
  \FN@checkvariant{\edef\reserved@d}{%
    \FN@checkvariant{\FN@restylefootnote{#2}}%
    {\csname FN@fnote\reserved@d\endcsname{#2}{#3}}}}

\def\FN@checkvariant#1#2{\def\reserved@a{#1}%
  \def\reserved@b{#2}%
  \futurelet\reserved@c\FN@checkvariantii}

\def\FN@checkvariantii{%
  \ifx\reserved@c+%
    \reserved@a{plain}\expandafter\@firstoftwo
  \else\ifx\reserved@c-%
    \reserved@a{para}\expandafter\expandafter\expandafter
    \@firstoftwo
  \fi\fi
  \reserved@b}
```

In order to be able to sort footnotes according to the order of their reference points, we use a sorted counter.

```
\newcounter{FN@totalid}
\MakeSorted{FN@totalid}
```

\FN@fnoteplain
\FN@fnotepara

The actual commands are easy enough:

```
\def\FN@fnoteplain{\FN@fnotecommon\vbox}
\def\FN@fnotepara{\FN@fnotecommon\hbox}
```

\FN@masterinsert

This contains the insert number of the insert where the footnote mark appears. If it appears in the main text, 255 will be used.

```
\def\FN@masterinsert{\@cclv}
```

\FN@id
\FN@master
\FN@slave

Here is the deal with master and slave ids: each footnote has a unique master id. This master id is larger by one than the last id of its subordinate footnotes. It is recorded in the mark `\FN@master` in the footnote box at the start itself, although with an indirection through the `\FN@newslot` mechanism since the actual id can only become known after all subfootnotes have been typeset. The same id is recorded in `\FN@slave` at the ultimate end of the footnote.

At the point where a `\FN@master` mark is placed, a default `\FN@slave` mark is placed also with an id that is one less than the smallest id generated from a footnote that is a 'descendent' of the current one. This makes it possible to distinguish any split off subordinate footnotes. It must be noted that this sentinel slave id will be the valid id of a completely unrelated footnote! Since the value is only used for determining one end of an *open* interval of excluded ids, this is no problem. All subordinate footnotes are numbered sequentially in the order of *completion*, so that any subordinate footnotes have lower ids than their master.

```
\newcount\FN@id
\FN@id\@ne
\newmarks\FN@master
\newmarks\FN@slave
```

\FN@errorstack

This records the history of nested footnotes in order to deliver more useful error messages.

```
\let\FN@errorstack\@empty
```

\FN@fnotecommon

Well, this is the work horse if the footnote macro. Really bad thing. We start off by stepping our absolute counter and making a mark. `\leavevmode` is required so that the action of `perpage.sty` is done smoothly.

```
\def\FN@fnotecommon#1#2#3{%
  \leavevmode
  \stepcounter{FN@totalid}%
  \NCC@makemark{#3}%
```

It is an error if the footnote insert number of the current footnote does not correspond to a block below the current insertion level.

```
  \ifnum#2<\FN@masterinsert
    \FN@colorstackbgroup\FN@divert
    \FN@newslot\FN@masterslot
    \count@\FN@id
```

`\dimen@` is here set to a sorting criterion. This is designed to make the conversion of footnote blocks as reliable as possible. If we could guarantee convergence, just

using `\c@FN@totalid` would be sufficient for sorting. It turns out that this is too sensitive to footnotes of different blocks changing pages, so the number of the superior footnote block is allowed to take precedence by multiplying it with 4194304 which is unlikely to get exceeded by `\c@FN@totalid`.

```
    \dimen@=\dimexpr64\p@*\FN@masterinsert-\c@FN@totalid sp\relax
    \def\FN@masterinsert{#2}%
    \edef\FN@errorstack{\FN@errorstack^^J%
\FN@masterinsert\space entered in line \number\inputlineno}%
    \let\FN@boxtype=#1%
    \setbox\z@#1\bgroup
```

The following is for the likes of PDFTEX which has its own idea about how to restore a color stack.

```
      \let\current@color\default@color
      \FN@@color@begingroup
      \let\MFL@minipage\relax
      \let\MFL@endminipage\relax
      \@makefnstartbox
```

We reset the list parameters in footnotes. Strictly speaking, this is interfering with LATEX's standard operation, but the standard operation does not make sense.

```
      \let\@listdepth\@mplistdepth \@mplistdepth\z@
      \@itemdepth\z@ \@enumdepth\z@
      \protected@edef\@currentlabel{\csname p@footnote%
         \expandafter\FN@stripfootins\string#2\endcsname\@thefnmark}%
      \ifx\FN@boxtype\vbox \normalcolor\nobreak
      \else \FN@specific{#2}\@preparefnhtext \normalcolor
      \fi
```

Ok, now we do the call to `\@makefntext` which may occur in one of several ways, depending on whether the 'robust' or the 'fragile' package option got used.

```
      \expandafter \FN@makefncall
  \else
```

We still needed to cater for the error of badly anchored footnotes:

```
    \PackageError{bigfoot}{#2 forbidden in \FN@masterinsert.}%
    {Higher-placed footnotes can't be anchored in inferior ones.^^J%
    I am not putting this text in a footnote.  History:%
\FN@errorstack}%
    \rule{1em}{\ht\strutbox}%
\fi}
```

`\FN@makefnstart` This is called in the start of `\@makefntext`.

```
\providecommand{\FN@seitenobreak}{\nobreak}
\def\FN@makefnstart{%
```

Record the footnote specific dimensions. It is assumed that they don't change in the document, at least not before the footnote gets actually placed.

```
  \expandafter\xdef\csname FN@ht\number\FN@masterinsert\endcsname
    {\the\footnotesep}%
  \expandafter\xdef\csname FN@dp\number\FN@masterinsert\endcsname
```

```
    {\the\dp\strutbox}%
  \expandafter\xdef\csname FN@wd\number\FN@masterinsert\endcsname
    {\the\hsize}%
```

The footnote gets markers for identifying it and its starting block.

```
  \marks\FN@master{\FN@masterslot}%
  \marks\FN@slave{\number\FN@id}%
  \nobreak
```

`\FN@commonending` will intervene before any tokens that are shifted in due to switching back the color stack. Those will only be executed once we completely relinquish control.

```
  \ifx\FN@boxtype\vbox
    \rule\z@\footnotesep
  \else
    \ifx\FN@par\par\else
      \let\FN@@par\par
      \let\FN@@noindent\noindent
      \let\FN@@indent\indent
    \fi
     \everyvbox\expandafter{\expandafter\everyvbox
       \expandafter{\the\everyvbox}%
       \let\par\FN@@par
       \let\noindent\FN@@noindent
       \let\indent\FN@@indent
       \the\everyvbox}%
     \let\par\FN@par
     \let\noindent\FN@noindent
     \let\indent\FN@indent
  \fi
  \FN@seitenobreak
  \afterassignment\ignorespaces}
```

`\FN@makefnrobust` After preparation, we now do the big bad trick for making footnotes cooperate with `\verb` and other catcode changing things: we call `\@makefntext` with an argument of `\iffalse`. This kills off its expansion right at the point where it would choose to place its argument.

Furthermore, this swallows the opening brace of the footnote text and then lets the footnote text progress. The closing group will then trigger the processing via `\aftergroup`.

```
\def\FN@makefnrobust#{%
  \FN@specific\FN@masterinsert\@makefntext
  \iffalse\fi
  \bgroup
    \aftergroup\FN@robustending
    \FN@makefnstart
    \let\next}
```

`\FN@robustending` Here we put in the missing part of `\@makefntext`.

```
\def\FN@robustending{%
  \expandafter\expandafter\expandafter
  \expandafter\expandafter\expandafter\expandafter
  \iffalse  \FN@specific\FN@masterinsert\@makefntext\fi
  \FN@commonending}
```

`\FN@makefnfragile` This is the escape route when the robust variant does not work. In that case, `\verb` and similar won't work in footnotes.

```
\long\def\FN@makefnfragile#1{%
  \FN@specific\FN@masterinsert\@makefntext
  {\FN@makefnstart#1\FN@commonending}}
```

Ok, color handling is a nuisance, to say the least. Split footnotes need to close their color stack on the old page, and reopen it on the new one. So we record the color stack state at each time it changes in a marks register.

```
\newmarks\FN@color
\def\FN@colorstackbgroup{\let\FN@savecolorstack\FN@colorstack
  \global\let\FN@colorstack\@empty
  \bgroup
  \ifdefined\FN@savecolorstack\else
    \let\FN@@set@color\set@color
    \let\FN@@reset@color\reset@color
    \let\FN@@color@begingroup\color@begingroup
  \fi
  \let\set@color\FN@set@color
  \let\reset@color\FN@reset@color
  \let\color@begingroup\FN@color@begingroup}

\def\FN@colorstackegroup{\egroup
  \global\let\FN@colorstack\FN@savecolorstack}

\def\FN@colorstackfinish{\def\@elt##1##2{\FN@@reset@color##2}%
  \FN@colorstack
  \def\@elt##1##2{\noexpand\@elt{}{##2}}%
  \xdef\FN@colorstack{\FN@colorstack}%
  \let\@elt\relax
  \marks\FN@color{}}

\def\FN@reset@color{%
  \bgroup\def\@elt##1##2{\def\FN@next{##1}{\gdef\FN@colorstack{##2}}}%
    \let\FN@next\@empty
    \FN@colorstack
    \ifx\FN@next\@empty
      \FN@colorstackegroup
    \else \egroup
      \FN@@reset@color
      \marks\FN@color{\FN@colorstack}%
  \fi}

```

```
\def\FN@color@begingroup{%
  \let\reset@color\FN@@reset@color
  \let\color@begingroup\FN@@color@begingroup
  \let\set@color\FN@@set@color
  \color@begingroup}

\def\FN@set@color{\FN@@set@color
  \xdef\FN@colorstack{\@elt{\current@color}{\FN@colorstack}}%
  \marks\FN@color{\FN@colorstack}}

\def\FN@coloraftersplit#1{%
  \def\@elt##1##2{##2\def\current@color{##1}\set@color}%
  #1%
  \let\@elt\relax}
```

`\FN@commonending` We'll eventually arrive here at the end of the footnote. Now we again call `\@makefntext`, but this time pass it `\fi` as its argument, and place `\iffalse` before its expansion. This cuts away the start of the macro. If this start changes the tail of the macro when executed, the whole trickery will not work. It turns out that a large sampling of document classes (including the standard ones) happens to work.

```
\def\FN@commonending{%
  \@makefnendbox
  \ifx\FN@boxtype\vbox\@finalstrut\strutbox \else \unskip \fi
  \FN@colorstackfinish
  \color@endgroup
  \egroup
  \global\advance\FN@id\@ne
  \FN@slotxdef\FN@masterslot{\number\FN@id}%
```

Now we want to get an upper estimate of the size. In case of a horizontal box, we do this by creating a vertical box of it all alone, and measuring that. Measuring the hbox itself is plain out: TEX's maximal dimension of something like 5 m is already busted with about two pages of material. We put the master slot identification into the depth of the box, and arrange for the total of depth and height of the box to still give the total depth and height of its size on the page.

```
  \ifhbox\z@
    \global\setbox\FN@tempbox\copy\z@
    \setbox\tw@\@makefnvbox{\unhbox\FN@tempbox}%
    \ht\z@\dimexpr\ht\tw@+\dp\tw@-\FN@masterslot sp\relax
  \else
    \ht\z@\dimexpr\ht\z@+\dp\z@-\FN@masterslot sp\relax
  \fi
```

Now we put the sorting criterion into the width of the box, and then put the masterslot id into the depth.

```
  \wd\z@\dimen@
  \dp\z@\FN@masterslot sp\relax
⟨trace⟩       \ifnum\z@<0\FN@slotget{\FN@masterslot} %
```

```
⟨trace⟩       \else \errmessage{Inconsistent
⟨trace⟩         \string\FN@masterslot=\FN@masterslot}\fi
```

Now we just need to place the stuff into an insertion and record the possibly changed slave id in order to know what subordinate footnotes belong to this one.

```
  \MFL@insert\FN@masterinsert{\nointerlineskip\box\z@}%
  \ifdim\lastkern=\z@ \let\FN@next\@empty\else
    \edef\FN@next{\kern\the\lastkern\relax}\unkern
  \fi
  \marks\FN@slave{\number\FN@id}%
  \expandafter\FN@enddivert\expandafter\FN@colorstackegroup
  \FN@next
}
```

A lot of stuff follows. This should really be cleaned up and documented.

```
\dimen\footins\maxdimen
\gdef\FN@nestlist{}

\newdimen\FN@outervsize
\newskip\FN@vsize

\newbox\FN@insertions

⟨trace⟩\def\MFL@showone#1#2{\message{Box #2:}\showbox#2%
⟨trace⟩  \MFL@checkconsistency{#2}%
⟨trace⟩  \message{Cachebox #2:}\showbox\FN@cache#2}
⟨trace⟩
⟨trace⟩\def\MFL@checkconsistency#1{{%
⟨trace⟩  \setbox\z@\vbox{\unvcopy#1%
⟨trace⟩    \MFL@checkconsistencyi{#1}}}}
```

`\MFL@checksinglebox` Check box `#1` for consistency. If it is bad, output box `#2`. Execute `#3` if it was good, `#4` if it was bad.

```
⟨trace⟩\def\MFL@checksinglebox#1#2#3#4{%
⟨trace⟩    \ifvoid#1\else
⟨trace⟩      \ifnum\z@<0\FN@slotget{\number\dp#1} %
⟨trace⟩        #3%
⟨trace⟩      \else \errmessage{Inconsistent box #2}%
⟨trace⟩         \showboxdepth4\showboxbreadth100
⟨trace⟩         \showbox#2\relax
⟨trace⟩         #4%
⟨trace⟩      \fi\fi}

⟨trace⟩\def\MFL@checkconsistencyi#1{%
⟨trace⟩    \unpenalty\unskip\unkern
⟨trace⟩    \setbox\z@\lastbox
⟨trace⟩    \MFL@checksinglebox\z@{#1}{{\MFL@checkconsistencyi{#1}}}{}}
⟨trace⟩
⟨trace⟩\def\MFL@showall{{%
⟨trace⟩    \showboxbreadth=\maxdimen
```

```
⟨trace⟩     \showboxdepth=4
⟨trace⟩     \tracingonline=\@ne
⟨trace⟩     \FN@nest@iterate\MFL@showone}}
```

`\FN@retaindelayed` This is a complex macro that removes all boxes from the current list that are not to be kept for the next page. It works on the material from the original insertions, not the cache boxes. The slot specified by `\count@` is not freed when encountered, all others are freed upon removing the box. The last box is returned in box 0 if any is retained. The vertical list might have an unchecked part locked off in front by placing a `\nobreak` penalty there. This penalty is removed, and the list before it not touched.

```
\def\FN@retaindelayed{%
  \setbox\z@\lastbox
  \ifcase
    \ifvoid\z@\m@ne\fi \FN@config\z@
⟨trace⟩  \if\foottrace8\message{^^J\string\FN@retaindelayed:
⟨trace⟩     dropping Id \FN@slotget{\number\dp\z@}}\fi
⟨trace⟩  \if\foottrace{16}{\showboxdepth4 \showboxbreadth400
⟨trace⟩     \tracingonline=\@ne\showbox\z@}\fi
    \ifnum\dp\z@=\count@\else \FN@freeslot{\number\dp\z@}\fi
⟨trace⟩ \ifnum\dp\z@<\@ne \errmessage{Unidentified box}\fi
    \expandafter\FN@retaindelayed
  \or
⟨trace⟩ \if\foottrace8\message{^^J\string\FN@retaindelayed:
⟨trace⟩    retaining Id \FN@slotget{\number\dp\z@}}\fi
⟨trace⟩  \if\foottrace{16}{\showboxdepth4 \showboxbreadth400
⟨trace⟩     \tracingonline=\@ne\showbox\z@}\fi
    {\FN@retaindelayed \nointerlineskip \box\z@}%
  \else \unpenalty \setbox\z@\lastbox
⟨trace⟩  \ifnum\lastnodetype>\m@ne
⟨trace⟩    \errmessage{Unexpected node \number\lastnodetype}\fi
⟨trace⟩  \ifvoid\z@ \else
⟨trace⟩     \if\foottrace8\message{^^J\string\FN@retaindelayed:
⟨trace⟩        carrying split box \FN@slotget{\number\dp\z@}}\fi
⟨trace⟩     \if\foottrace{16}{\showboxdepth4 \showboxbreadth400
⟨trace⟩        \tracingonline=\@ne\showbox\z@}\fi\fi
  \fi}
```

`\MFL@processplain` This gets called for actually inserting the processed material into the footnote box. The current state of affairs is that `\FN@config` contains all footnotes that should get transferred to the next page completely. The cache boxes contain the collected and typeset footnotes for typesetting on the current page.

The structure of a cachebox is currently as follows: it is filled with vboxes containing the arranged material, optionally followed by another box to be carried over to the next page flagged with a `\nobreak` penalty.

```
\def\MFL@processplain#1{%
⟨trace⟩ \MFL@checkconsistency#1%
   \ifvoid\FN@cache#1%
```

Now if the cache box is void, nothing gets typeset on the current page. What we do, however, is to collect all boxes from the original insertion that did not make it on this page and reinsert them. `\count@` is cleared to zero to retain nothing special.

```
    \global\setbox\FN@tempbox\vbox\bgroup
      \unvbox#1%
      \count@\z@
      \let\@elt\FN@removecheck \FN@retaindelayed
      \ifvoid\z@ \egroup
      \else \nointerlineskip \box\z@ \egroup
        \MFL@realinsert{#1}{\unvbox\FN@tempbox}%
      \fi
```

The following stops in the insertion process within the `manyfoot` package.

```
    \expandafter\expandafter
  \fi\iffalse\fi
```

Ok, this is the case when we have a nonvoid cache box.

```
  \global\setbox#1\vbox\bgroup%
    \unvbox\FN@cache#1%
    \ifnum\lastpenalty>\z@
      \unpenalty
      \setbox\z@\lastbox
    \else
      \setbox\z@\box\voidb@x
    \fi
```

Ok, now box zero contains carryover material (if any). We initialize `\count@` to this so that we will keep this carryover material just once.

```
    \count@\dp\z@
    \global\setbox\FN@tempbox\vbox\bgroup
      \box\z@
      \nobreak
      \unvbox#1%
      \let\@elt\FN@removecheck \FN@retaindelayed
      \ifvoid\z@ \egroup \MFL@removevboxes\egroup
      \else \nointerlineskip \box\z@ \egroup
        \MFL@removevboxes \egroup
        \MFL@realinsert{#1}{\unvbox\FN@tempbox}%
      \fi}

\let\MFL@processpara\MFL@processplain
```

Ok, here is the bit about the caches: whenever we encounter a new configuration, we have to first update the caches since we don't know the sizes we are dealing with regarding the new configuration until we do so. The caches are kept up to date globally. When we are working at several levels in the recursion, we have a bottom active level where we may are looking for a way to find a best break and configuration. We will return at most one configuration once we are finished. While we are working with a returned configuration, adding more material on the

current list will not require another recursion as long as the totals stay underfull: the penalty difference between underfull configurations becomes smaller while the underfullness decreases, which means that smaller breaks that have not been chosen before might become eligible if the penalties allow for that. Only when the badness of underfullness remains infinite can't we have any improvement.

Ok, after we recurse for removing an underfull condition, the resulting configuration can't actually be used further for breaks with less remaining space. It is, however, clear that if less space remains, there is no better break with the same configuration leaving *more* space: if there were, it would already have been taken. That means that our goal height for the next break will be chosen in order to reach the exact size met on the last recursion. No break before that can be chosen on the next try, but a break after it might then be taken.

available, or an overfull one. If a deeper level at any point of time returns an overfull configuration, we are finished. The best configuration to be returned is the least underfull. If there is none, the least overfull. The case of no underfull at all can only happen if even splitting this and every subordinate level to minimal height and recursing does not yield an underfull. At every level, we need to maintain just a current split, and the previous best split at most.

When we change a configuration on recursing, we have to remember the configurations for the previous best split. We can manage that by sweeping the current cache values into a local box register before recursing with a different configuration: we have to rebuild the box registers for a different configuration, anyway. We don't save the configuration from an overfull setting: when we rework the list in slow motion mode, we can't help stopping the recursion by reaching an overfull setting that is at least as good as the initial one.

When we return to a caller, we leave the cache in the configuration of the best choice up to now: either we are returning an overfull configuration and if it is not the best so far, the caller can restore his better choice from his copy, or we are returning an underfull configuration in which case the caller might still want to improve upon it before returning to its caller in turn. New: If we return an underfull configuration, we also return an "optimal penalty estimate" that gives the best break point penalty under the assumption that additional stretchability is present on the page.

The purpose of this is to offer the possibility of avoiding widows and similar by moving more material in some footnotes to the next page in exchange for other material.

At the current grouping level we empty out our current cache and keep it for working purposes on the vertical list until we return (nobody references it while we are working on it). We always enter with an overfull configuration, meaning that `\FN@vsize` is negative. It is calculated with the current cache/config setting.

There is a danger of overflow involved with that: if we keep a swept complete configuration at each level of recursion, we need $O(n^2)$ of space here. The alternative would be to keep the history of how the configuration came about. Since that might involve some slow-motion splitting, this is also a speed issue. Since deep recursion with pending best data at each level is not really likely, and since we are not going to have that many footnote levels to go around, anyway, we just

rely on LaTeX having been started with sufficient memory.

A workable compromise would be to just store the split boxes from a configuration together with the configuration data for reconstructing the rest. After all, we don't need to reconsider such a configuration before actually typesetting anything. And whenever we find an acceptable fit (neither underfull/overfull), we could cut through all the hierarchy without having to restore anything. This has not been implemented yet: at the moment we go for the less complicated variation.

The algorithm we use here is a bit complicated. Whenever we recurse, we have one of the following situations:

1. An overfull/underfull dilemma: including a minimal amount of material at the current level will cause the page to become overfull. This can be the case in connection with zero (in case of interline penalties for larger blocks), one or more subordinate footnotes and related footnotes.

2. A pure overfull dilemma: the page was overfull to start with, we need to reduce it.

3. an underfull dilemma: some operation in the next level made the page become underfull, only too much so. We can't make it fuller on the current level, but we can make it even emptier, and let the next level fill it up again.

In the current implementation, we just ignore the slight probability that the optimum choice might lie with case 3. We don't recurse for making the page fuller again. If we have an overfull/underfull dilemma, the recursion will either give us a less awful overfull box, or an underfull one. An overfull box that occurs at the highest level of recursion can't be improved on any lower level. So we never need to locally return an overfull box: we can compare it to the best overfull box seen before, and if we turn out better than that, we overwrite the global best overfull value and return the best local underfull if there is such a one. The best local underfull will then be refilled as much as possible on the next level without changing the configuration. Actually, if we need to change the configuration, this would also be fine as long as we arrive at a better underfull eventually. But since a change of configuration renders our previous split completely useless, as the broken paragraph could look disastrously different under a changed configuration, we would need to recurse again. We repeat this recursing operation until we don't get an underfull solution returned anymore. We then return the best underfull, if any. The best overfull is stored globally, as mention before.

Does this sound complicated? Unfortunately, it does. It also sounds somewhat slow. For that reason, we do a few assumptions that will facilitate a good average-case behavior. The first assumption is that we will usually do fine by just splitting in the current level (if at all) and not at all in subordinate levels.

We do this assumption on the first pass used for gathering the size information and collect the corresponding boxes in nested lists. When the recursion tops out, it does so either with an overfull page, or an underfull page. If it does with an underfull page, we cache the current configuration for the next pass through the output routine, so that we won't need to retypeset and measure assembled boxes

that have not gathered any new material. If we top out with an overfull page, the previous underfull configuration is still worth keeping as well, as it might become the material actually chosen to be typeset.

Ok, the current best configuration of the next recursion level is gathered on the current vertical list, in a separate box. We use box 2 for this purpose. A saved configuration consists of the complete contents for the current cache box without the trailing penalty indicating material from a single split box carried over to the next page (boxes that are carried over completely to the next page are not maintained here but rather reinserted by `\MFL@processnested`). This penalty is added in case the box is actually disassembled and returned: there is no possibility for confusion since we only save such a configuration if indeed there is a split present.

```
\newtoks\FN@output
\FN@output\output

\newbox\FN@tempbox
\newinsert\FN@savebox
\count\FN@savebox\@m
\dimen\FN@savebox\maxdimen
\skip\FN@savebox\z@skip
\global\setbox\FN@savebox\box\voidb@x

\expandafter\expandafter\expandafter\let\FN@cache\FN@savebox=\@cclv

\expandafter\def\csname FN@ht\number\FN@savebox\endcsname{\z@skip}%
\expandafter\def\csname FN@dp\number\FN@savebox\endcsname{\maxdepth}
\expandafter\def\csname FN@wd\number\FN@savebox\endcsname{\columnwidth}

\def\FN@list{\MFL@list\@elt{}\footins}

\def\FN@sweepbox#1#2{\ifvoid#2\else
  \nointerlineskip\box#2\penalty#2\fi}

\def\FN@sweepcachebox#1#2{\nointerlineskip
  \box\FN@cache#2%
  \penalty\FN@cache#2}

\def\FN@copycachebox#1#2{\nointerlineskip
  \copy\FN@cache#2%
  \penalty\FN@cache#2}

\def\FN@restoreboxes{\count@\lastpenalty \unpenalty
  \ifnum\count@>\z@
    \global\setbox\count@\lastbox
    \expandafter\FN@restoreboxes
  \fi}
```

`\FN@removecheck` This returns `\@ne` if and only if the current slot master is strictly inside of the

specified open interval. In this case it is not to appear on the current page.

```
\def\FN@removecheck#1#2{%
  \ifnum#1<\FN@slotget{\number\dp\z@} %
  \ifnum#2>\FN@slotget{\number\dp\z@} %
    \@ne\fi\fi}
```

Parameter recording merely records the relevant value of the skip register and sets it to zero. The purpose is to avoid changes of the reserved page space when we collect additional material from a page where an insertion of the appropriate kind had already been encountered. This is used for filling up underfull pages.

```
\def\FN@recordinsertparam#1#2{\ifvoid#2\else
  \global\skip\number#2=\the\skip#2\relax\fi}

\def\FN@clearinsertparam#1#2{\ifvoid#2\else
  \global\skip#2=\z@skip\fi}
```

`\FN@insertouterspace` will sum the size of the inserts manually.

```
\def\FN@insertouterspace#1#2{\ifvoid#2\else
    +\skip#2+(\ht#2+\dp#2)*\count#2/\@m\fi}

\def\FN@list@iterate#1{\let\FN@eltsave\@elt
  \let\@elt#1%
  \FN@list
  \let\@elt\FN@eltsave}

\def\FN@nest@iterate#1{\let\FN@eltsave\@elt
  \let\@elt#1%
  \FN@nestlist
  \let\@elt\FN@eltsave}
```

1.5 The output routine stuff

Marks This is used for sweeping all marks up for reinsertion.

`\FN@allmarks`

```
\def\FN@allmarks#1{\@elt{#1}%
  \ifnum#1<\count266
    \expandafter\FN@allmarks\expandafter{\number\numexpr#1+\@ne}%
  \fi}
```

`\FN@sweeptopmarks` This sweeps the current topmarks and places them into the global box `\FN@topmarkbox`.

`\FN@topmarkbox`

```
\def\FN@sweeptopmarks{\global\setbox\FN@topmarkbox\vbox{%
    \def\@elt##1{\marks##1{\unexpanded\expandafter{\topmarks##1}}}%
    \FN@allmarks0}}
\newbox\FN@topmarkbox
```

`\FN@establishmarks` This sets marks from a marks sweep. The first argument is the mark number, the second is from the first mark on the first scan, the third argument from the bottom mark on the first scan, and the fourth argument from the bottom mark

on the second scan (with additional mark entries). If second and third arguments don't match, no mark gets placed.

```
\long\def\FN@establishmarks#1#2{\edef\reserved@a{\unexpanded{#2}}%
  \edef\reserved@b{\unexpanded\expandafter{\splitbotmarks#1}}%
  \ifx\reserved@a\reserved@b
    \marks#1{\unexpanded\expandafter{\splitfirstmarks#1}}%
    \marks#1{\unexpanded\expandafter{\reserved@b}}%
  \fi}
```

`\FN@markspassone` This constitutes the first pass for mark collection. We do this just to check whether there are any marks in the list.

```
\def\FN@markspassone#1{\noexpand\FN@establishmarks{#1}%
  {\unexpanded\expandafter{\splitbotmarks#1}}}
```

`\FN@insertmarks` This routine transfers first and bottom marks from the current `\box255` to the vertical list in order to get the marks right. This is quite a bother, since we must detect the special case where there are no marks at all in the list, and since we might require the use of several `\vsplit` commands in a row, since infinite stretch might make the optimal breakpoint lie before the end of the box in spite of its large size.

So we need to do the splitting in a loop, and do it twice, once with artificial marks at the start. If those artificial marks make it to `\splitbotmarks`, we don't place any actual marks.

```
\def\FN@pseudomarks#1{\marks#1{X}}
\def\FN@insertmarks{%
  {\setbox\z@\copy\@cclv
    \splittopskip-\maxdimen\relax
    \vbadness=\@M
    \vfuzz=\maxdimen
    \loop
    \ifvoid\z@\else
      {\let\@elt\FN@pseudomarks
        \setbox\z@\vbox{\FN@allmarks0\nobreak\unvcopy\z@}%
        \setbox\z@\vsplit\z@ to\maxdimen}%
      \let\@elt\FN@markspassone
      \edef\next{\FN@allmarks0}%
      \setbox\z@\vbox{\nobreak\unvbox\z@}%
      {\setbox\z@\vsplit\z@ to\maxdimen}%
      \next
    \repeat}}
```

Some stuff

`\FootnoteMainMinimum` This specifies the minimum amount of main text. You can make this a complicated expression if you want to, for example by checking the presence of particular footnotes.

```
\def\FootnoteMinimum{1sp}
\def\FootnoteMainMinimum{0pt}
```

```
\expandafter\def\csname\string\FootnoteMinimum\number\FN@savebox
\endcsname{\FootnoteMainMinimum}
```

The output routine itself This is our own output routine that does all the balancing stuff. If we receive a forced penalty here, we must not do any of our output processing on our own unless this is the choice of the underlying output routine. We do want to have the 'real' output routine to have a correct idea about the size that the insertions will take up. So the steps that we *will* actually perform in any case are sorting the insertions and calculating their real size. If we have not had a forced penalty, we are free to exit the output routine for gathering further material as there are no expections of the underlying output routine when it should get called. If we encountered a forced penalty, things are getting more complicated. If the current page happens to be overfull after adding the current material, we first need to ship out the material for a regular page (after splitting off the necessary material for the next page). We then reinsert the remaining split insertions, any possibly split off page material and the penalty.

TEX is rather monotonous in its page break processing. Increase the available page size, and the available page material will also increase. There is a singular exception to that rule, and that are split and floating insertions. However, we notice their presence by a non-zero setting of `\insertpenalties`, and we can just measure the material that they have taken up in a forced pass of the output routine, adding that much to our request size. However, this operation will change the penalties associated with the page breaks.

Unfortunately, this is not sufficient: the penalty might have been inserted with a box immediately preceding it. In that case the penalty would have been guaranteed to eventually turn up in the output routine. If we now reinsert merely all of the above stuff, the penalty will just disappear. If we protect the penalty by placing an empty box before it when none of it had been before it before, we will get an empty page. Since we don't know whether the penalty was supposed to disappear at the start of an empty page or not, we will do the following: if the rest of `\box255` is nonvoid, we just reinsert the split insertions followed by the rest of and the penalty and return. If it is void, we call the regular output routine, capturing its output in a `\vbox` of its own. If the regular output routine failed to ship out the prepared insertions, we just keep the original data either in their boxes or in a reinserted insertion.

It hides the relevant information from the 'real' output routine until such a time that we have enough material gathered to produce a full page. The exception to this is when we have a special penalty that gets passed through to the regular output routine.

If we are on a material collecting spree, `\FN@savebox` contains all boxes from the last output call time. At the point where we enter the output routine, `\FN@vsize` contains the amount of space available for mounting footnotes, after subtracting all insertions of footnote variety. At most times in our output routine, the variable will contain the amount of space left after everything is put to the page including footnotes.

```
\savingvdiscards=\@ne
```

We have the following situations that can cause us to enter the output routine:

1. The page has just filled up.
2. A magic output penalty has been encountered.
3. We are filling up a previously underfull page.
4. We are looking for missing insertions that may have floated.

We are trying to do bookkeeping on the effects of page size for insertions that fall into the footnote class. While we do basic bookkeeping for other insertions as well, this can only be incomplete since we don't reinsert material. In consequence, multiple material ending up in the same insertion might cause the corresponding skip register to be accounted for several times. LaTeX does not really reuse insertions in that manner except for footnotes, so we are mostly ok here.

`bigfoot` usually does some lookahead in the main list in order to obtain optimal breakpoints. It explicitly undoes the effect this has on marks, but insertions are a different matter here. So floats may appear on an earlier page than expected.

If the output routine is invoked with a penalty of -13750, then the page content is merely used for setting the `\topmarks` array. In that case, we just clear out the output box and resume. We don't fiddle with `\deadcycles` in order to catch foulups.

Also we don't touch insertion boxes. There is a particular situation where there *are* insertions, namely if we are collecting insertions after the last output routine has ended up with a non-zero value of `\insertpenalties`. In this case, all insertions we *do* get are floating insertions, meaning that they had a preceding insertion of the same class already on the last page, and thus we have zeroed its skip register already. We are assuming that a single pass with such a large `\vsize` is sufficient for pulling all insertions. If that happens to be incorrect, insertions need to get pulled in piecewise, but then we are probably in big dodo with regard to page size accounting, anyway.

```
\newcount\FN@outputflag
\FN@outputflag=3158345
\output{%
  \let\@elt\relax
  \ifvoid\@cclv \PackageError{bigfoot}{Empty box 255 in \output}\fi
⟨trace⟩  \if\foottrace8%
⟨trace⟩    \message{entering output with
⟨trace⟩       \outputpenalty=\the\outputpenalty:}%
⟨trace⟩    {\showboxdepth4\showboxbreadth\maxdimen\showbox\@cclv}\fi
  \ifnum\outputpenalty=-13750
⟨trace⟩  \if\foottrace8%
⟨trace⟩    \message{Discarding box 255.}%
⟨trace⟩  \fi
    \ifnum\insertpenalties>\z@
      \PackageError{bigfoot}{Too much insertion material}{%
```

```
798         This error means that the output routine was not able to^^J%
799         gather all floating insertions in a single pass.^^J%
800         Complain to the author if you consider this a bug}%
801     \fi
802     \global\advance\FN@outervsize\dimexpr\ht\@cclv-\vsize
803     \global\setbox\@cclv\box\voidb@x
804   \else
```

Note that a potential `\FN@vsadjustlist` will restore the previous value of `\outputpenalty`. So we need to save it.

```
805   \edef\FN@outputpenalty{\number\outputpenalty}%
806   \ifvoid\FN@savebox
807     \ifvoid\@holdpg
808       \FN@sweeptopmarks
809     \fi
810     \FN@nest@iterate{\FN@insertouterspace\global\FN@outervsize
811       \dimexpr\z@}%
812     \global\advance\FN@outervsize\ht\@cclv
813     \global\setbox\@cclv\vbox{\unvbox\@cclv\boxmaxdepth\maxdepth}%
814     \global\let\FN@vsadjustlist\@empty
```

`\FN@outervsize` now contains the value of `\pagegoal` at the time of output. It should be `\vsize` adjusted by the natural size of insertions. Note that `\FN@normaloutput` is not required to return with a sensible value of `\outputpenalty`.

```
815     \FN@normaloutput
816   \else
```

We now are in the situation that we already have collected material previously. We can't be sure that adding a special penalty does not take more than one output routine call before delivery. For that reason, we don't rely on special outputs being special and always subtract any additionally demanded `\vspace` from `\FN@outervsize` before calling a special output, so that we can afterwards compensate for it.

```
817     \FN@nest@iterate{\FN@insertouterspace\global\advance\FN@outervsize
818       \dimexpr\ht\@cclv}%
819     \global\setbox\@cclv\vbox{\unvbox\@cclv\boxmaxdepth\maxdepth}%
```

Now we invalidate the cache boxes for all insertions that had changed due to the recent additions to the page (this does *not* affect `\FN@vsize`).

```
820     \FN@nest@iterate\FN@maybeinvalidatecache
```

We now update all boxes by inserting the previously collected material in front of the boxes.

```
821     \vskip\z@skip
822     \unvbox\FN@savebox
823     \loop
824       \count@\lastpenalty
825     \ifnum\count@>\z@
826       \unpenalty
827       \setbox\z@\lastbox
```

```
      \global\setbox\count@\vbox{\unvbox\z@\unvbox\count@}%
    \repeat
    \ifcase
      \ifnum\FN@outputpenalty=-13749 \@ne\fi
      \ifnum\FN@outputpenalty=-13751 \@ne\fi \tw@
    \or
```

`\outputpenalty` is restored to the original value before the total page is glued together.

```
      \FN@vsadjustlist
⟨trace⟩ \if\foottrace8\message{receiving special penalty
⟨trace⟩    \FN@outputpenalty, dissing box 255:}%
⟨trace⟩      {\showboxdepth4 \showboxbreadth400
⟨trace⟩       \tracingonline=\@ne\showbox\@cclv}\fi
```

This special penalty means that we have been collecting floated insertions right now. So `\box255` is actually empty except for filler material. We restore the old box into it.

```
      \global\setbox\@cclv\lastbox
      \unskip
⟨trace⟩  \ifnum\lastnodetype>\m@ne
⟨trace⟩    \errmessage{Unexpected node \number\lastnodetype}\fi
      \ifnum\FN@outputpenalty=-13749
        \FN@normaloutput
      \else
        \the\FN@output
        \@pageht-\vsize
        \let\@currbox\footins
        \@reinserts
        \global\vsize-\@pageht
      \fi
    \else
      \dimen@\topskip
      \FN@vsadjustlist
      \setbox\z@\lastbox
      \unskip
⟨trace⟩  \ifnum\lastnodetype>\m@ne
⟨trace⟩    \errmessage{Unexpected node \number\lastnodetype}\fi
```

Ok, now we reconstruct the box from its parts. We add the material together, taking the previous output penalty and the current page discards (if it belongs between those boxes, otherwise we leave it on the list) for glueing the stuff together. The previous output penalty then is irrelevant for further purposes and we replace it again. `\FN@outervsize` has been adjusted by the accumulated contributions of insertions to the page size. Fiddling with it would not appear necessary or even prudent.

```
⟨trace⟩\if\foottrace8
⟨trace⟩   \message{Box 255 before reglue
⟨trace⟩     (outputpenalty=\the\outputpenalty):}%
⟨trace⟩   {\showboxdepth4\showboxbreadth100\showbox\@cclv}\fi
```

```
      \global\setbox\@cclv\vbox{%
```

Now we might have had a `\topskip` value designed for requesting a given number of lines. We need to remove anything of that kind. Splitting again achieves that. If the current page was empty except for insertions, this means that we gain a new breakpoint. But insertions with discardable material before them would be unusual.

The only exception to this may happen if the current page contained only insertions: in this case TEX has made a page break before the actually inserted `\topskip` glue (which will then arrive one page later).

Note that the pagediscards contain material corresponding to the last breakpoint chosen, so they will either start with a penalty of 10000 (which is what an actual outputpenalty gets replaced with) or will start with discardable material. We clean it for that reason.

```
        \unvbox\z@
        \global\setbox\@cclv\vbox{\break\unvbox\@cclv}%
        {\splittopskip-\maxdimen \setbox\z@\vsplit\@cclv to\z@}%
        \ifnum\outputpenalty=\@M
          \setbox\z@\vbox{\pagediscards
            \FN@cleanpagepenalty}%
          \unvbox\z@
        \else
          \penalty\outputpenalty
          \pagediscards
        \fi
        \unvbox\@cclv
        \boxmaxdepth\maxdepth}%
⟨trace⟩\if\foottrace8
⟨trace⟩   \message{Box 255 reglued (outputpenalty=\FN@outputpenalty):}%
⟨trace⟩   {\showboxdepth4\showboxbreadth100\showbox\@cclv}\fi
      \global\outputpenalty\FN@outputpenalty\relax
```

Ok, now if `\topskip` is actually *positive*, we have been collecting material tentatively without having proper marks. We then need to fill in the marks into the list and try again. Note that we are not reinserting anything in order to compensate for `\outputpenalty` being replaced by a nobreak: this is the job of `\FN@normaloutput` when it decides to place material back on the page. That is: when code is written that will make use of pagediscards, it has to cater for their proper structure.

```
      \ifdim\dimen@>\z@
⟨trace⟩  \if\foottrace8
⟨trace⟩    \message{recycling special penalty}
⟨trace⟩  \fi
        \hrule\@height\z@\@depth\z@
        \unvcopy\FN@topmarkbox
        \penalty-13750
        \penalty\FN@outputflag
        \hrule\@height\z@\@depth\z@
        \FN@insertmarks
```

```
        \penalty-13749
        \penalty\FN@outputflag
        \FN@prepareoutput
        \global\topskip-\maxdimen
        \global\vsize0.5\maxdimen
        \global\advance\FN@outervsize-\vsize
      \else
        \FN@normaloutput
      \fi
   \fi
  \fi
  \fi}
```

`\FN@normaloutput` This is the normal output routine we use. Now we have recovered a sensible state and glued everything together that has been necessary. All insertion parameters are at their standard values, and any insertions have been collected in the respective boxes.

```
\def\FN@normaloutput{%
⟨trace⟩ \if\foottrace8\message{^^JEntering \string\FN@normaloutput:^^J}\fi
```

`\FN@vsize` is now being set to the vertical size taken up by the insertions, according to TEX. Note that this does not include flexibility. This much amount of space gets available on the current page if we remove all insertions. This figures into `\FN@vsize` as a *positive* quantity since the insertion size was taken from `\pagegoal`, and we reconstitute it in this manner.

```
  \global\FN@vsize\FN@outervsize
  \global\advance\FN@vsize-\ht\@cclv\relax\relax
```

Now we sort the inserts and regenerate the cache.

```
  \FN@nest@iterate\FN@sortinsert
  \FN@nest@iterate\FN@clearcache
  \xdef\FN@config{\@elt{\number0\botmarks\FN@slave}%
    {\number\maxdimen}}%
⟨trace⟩ \if\foottrace8%
⟨trace⟩     \message{\noexpand\FN@normaloutput start config: \FN@config^^J}%
⟨trace⟩ \fi
  \FN@nest@iterate\FN@reconfig
```

Note that `\FN@reconfig` subtracts the *actual* size of all insertions (after paragraphs have been combined and too early insertions moved to the next page) and also subtracts the flexible glues associated with the insertions' skip registers, so this flexibility is typically negative. Since the cache registers have been explicitly cleared, `\FN@reconfig` starts from the state where indeed no insertions are present.

```
  \ifcase
    \ifnum\insertpenalties>\z@ \@ne\fi
```

If we have floating insertions, we need to catch up with them. This is done in case 1 which just places an immediate penalty and recurses.

Now here are a few cases that are only checked when we don't have a special

penalty:

```
    \ifnum\outputpenalty>-\@M
```

The first case is if the page is underfull. We need more material then.

```
      \ifdim\FN@vsize>-\gluestretch\FN@vsize \tw@ \fi
```

Second case is when there is not enough vertical minimum material.

```
      \ifdim\FootnoteMainMinimum>\ht\@cclv \tw@ \fi
    \fi
```

Case 3 means page is overfull. If there are no missing insertions, try to split.

```
    \ifdim\FN@vsize<\glueshrink\FN@vsize \thr@@ \fi
```

page has appropriate size or we have special penalty. If we have come here not the first time, we might have arrived at a non-optimal break. So we attempt a split.

```
    \ifx\FN@vsadjustlist\@empty \else \thr@@\fi\z@
```

Ok, now we get the default case in our big routine: case 0. We just pass the result onto the output routine.

```
    {\vbadness\@M
      \vfuzz\maxdimen
      \global\setbox\@cclv\vbox
      spread\FN@vsize{\unvbox\@cclv\boxmaxdepth\maxdepth}}%
    \the\FN@output
    \let\@currbox\footins
    \@pageht-\vsize
    \@reinserts
    \global\vsize-\@pageht
    \FN@nest@iterate\FN@clearcache
  \or
```

Case 1: We just pull in remaining insertions and are done. Note that the special penalty here will get turned into an explicit nobreak. So if we have no record of an actual outputpenalty, we need to insert an artificial penalty of 0 here.

```
    \FN@restartoutput
    \penalty -13749
    \penalty \FN@outputflag
  \or
```

Case 2: Now we want to gather additional material. This is somewhat weird. We first gather our material with a 'normal' setting of topskip, and then we'll have another go at the material using proper marks. We can't actually insert anything right now in order not to introduce a premature breakpoint.

```
    \dimen@=\dimexpr\FN@vsize-\glueshrink\FN@vsize\relax
    \FN@prepareoutput
    \global\topskip \normalbaselineskip
    \global\vsize \dimen@
    \global\advance\FN@outervsize-\vsize
    \global\deadcycles\z@
  \else
```

This is case 3: Fake our output box into something looking like a cache box and

do the optimal split routine. The output cache box has a few deficiencies: its inner box is *not* depth-extended to some default measurement. That means that where page size calculations are involved, one needs to *disregard* its actual depth and instead use `\maxdepth`. This is somewhat awkward and prone to problems. One alternative might be to mark the box as split, extend its depth in the split part and let it be followed by nothing as lower part of the split. But we still would need to account for the missing depth at the end.

```
    \edef\FN@masterid{\number\maxdimen}%
    \def\FN@masterslot{-1}%
    \global\setbox\@cclv\vbox{\box\@cclv}%
    \xdef\FN@config{\noexpand\@elt{\number0\botmarks\FN@slave}%
      {\number\maxdimen}}%
⟨trace⟩ \ifvoid\FN@savebox \else \PackageError{bigfoot}{\FN@savebox
⟨trace⟩   \space should be void!}{}\fi
    \global\setbox\FN@savebox\vbox{}%
    \gdef\FN@penalties{0}%
    \edef\FN@defaultpenalty{\ifnum\outputpenalty<\@M
        \number\outputpenalty
      \else
        0\fi}%
    \let\@elt\FN@newlevel
    \@elt{}\FN@savebox\FN@nestlist\FN@mainsplitreturn
    \let\@elt\relax
⟨trace⟩ \if\foottrace8{\showboxdepth4\showboxbreadth100\showbox\@cclv}\fi
    \global\setbox\FN@savebox\box\voidb@x
    {%
      \vbadness\@M
      \vfuzz\maxdimen
      \global\setbox\@cclv\vbox spread\FN@vsize{%
        \unvbox\@cclv
        \ifnum\lastpenalty>\z@
          \unpenalty
          \global\setbox\FN@tempbox\lastbox
        \else
          \global\setbox\FN@tempbox\box\voidb@x
        \fi
        \setbox\z@\lastbox
```

Now if a split has been done, `\box\FN@tempbox` contains the lower part of the split. In either case, `\box\z@` contains the upper part of the split (in a prepared form with the splitdiscards in a box of their own). This may be void if there is no main text but only footnotes. If we have carryover material, we add the current outputpenalty there and set outputpenalty to a value indicating that we have no outputpenalty to add at the end of the current list.

```
        \ifvoid\z@
          \ifvbox\FN@tempbox
            \ifnum\outputpenalty<\@M
% The output penalty originally from below the split box gets appended
% to the end of the split box.
```

```
            \global\setbox\FN@tempbox{\unvbox\FN@tempbox
              \penalty\outputpenalty}%
          \fi
          \global\outputpenalty=\@M
        \fi
      \else
```

Ok, we have material to go to the next page. We unpack it and fish out the break penalty from the last box. After checking it, put it in box 0. A prospective current break penalty gets appended to the carryover material. The fished-out break penalty becomes the new value of outputpenalty.

```
        \MFL@removevboxes
        \unvbox\z@
        \edef\FN@defaultpenalty{\number\@M}%
        \FN@getbreakpenalty
        \setbox\z@\lastbox
        \global\setbox\FN@tempbox\vbox\bgroup\unvbox\z@
          \unvbox\FN@tempbox
          \ifnum\lastnodetype<\z@
          \egroup\global\setbox\FN@tempbox\box\voidb@x
          \else
            \ifnum\outputpenalty<\@M
              \penalty\outputpenalty
            \fi
          \egroup
          \global\outputpenalty\FN@breakpenalty
        \fi
      \fi
      \boxmaxdepth\maxdepth}%
  }%
  \setbox\z@\box\FN@tempbox
  \let\@elt\relax
```

Ok, now we have in box 255 the split off stuff for the current output routine, and in box 0 stuff that is going to follow afterwards. If box 0 is not void, we were not able to make use of all of box 255. There is a slight probability that by taking even *more* material from the main list, we might get a better result (by being able to move footnote material to the next page instead), but we don't make use of this possibility here. In general, we assume that if box 0 is nonvoid, we take the resulting split. Otherwise, if the page appears underfull, we pull in more material. If the page is not underfull, we can pass it to the output routine. If box 0 is void, the break was chosen at the ultimate end of the vertical list. If it was not a forced break, and if it is not an overfull case already, we pull in more material in order to avoid widows in the main text.

```
  \dimen@=\dimexpr\FN@vsize-\glueshrink\FN@vsize\relax
  \ifcase
    \ifvoid\z@ \ifnum\outputpenalty>-\@M
                  \ifdim\dimen@<\z@ \else \@ne \fi
               \fi
```

```
      \else \thr@@
      \fi
      \ifdim\ht\@cclv<\normalbaselineskip \@ne\fi
      \ifdim\dimen@<\normalbaselineskip \tw@\fi \@ne
    \or
      \FN@prepareoutput
      \global\topskip \normalbaselineskip
      \ifdim\dimen@<\normalbaselineskip \dimen@=2\baselineskip\fi
      \global\vsize \dimen@
      \global\advance\FN@outervsize-\vsize
      \global\deadcycles\z@
    \or
⟨trace⟩ \if\foottrace8%
⟨trace⟩   \message{^^JOutput: config is \FN@config...}\fi
      \setbox\tw@\vbox{%
        \the\FN@output
⟨trace⟩ \if\foottrace8%
⟨trace⟩   \ifnum\lastnodetype=\m@ne
⟨trace⟩     \message{^^JOutput: end without carryover^^J}%
⟨trace⟩   \else
⟨trace⟩     \message{^^JOutput: end with carryover}}%
⟨trace⟩      {\showboxdepth5 \showboxbreadth400
⟨trace⟩      \tracingonline=\@ne\showbox\tw@
⟨trace⟩   \fi
⟨trace⟩ \fi
      }%
      \unvbox\tw@
      \unvbox\z@
      \let\@currbox\footins
      \@pageht-\vsize
      \@reinserts
      \global\vsize-\@pageht
      \FN@nest@iterate\FN@clearcache
    \or
      \FN@restartoutput
      \penalty -13751
      \penalty\FN@outputflag
      \unvbox\z@
      \ifnum\outputpenalty>\@M
      \else \penalty
            \ifnum\outputpenalty=\@M \z@ \else\outputpenalty\fi
      \fi
    \fi
  \fi
⟨trace⟩ \if\foottrace8\message{^^JExiting \string\FN@normaloutput^^J}\fi
}
```

`\FN@prepareoutput` This is a preparation for gathering more material. First sweep up all the information about `\vsize`, `\topskip` and `\outputpenalty`. After that, record the insertion skip parameter of all insertions that have already been started, and re-

set them to zero so that no additional space gets reserved for them in case more material accumulates. We don't reset `\topskip` here since the amount of newly requested material will typically be in total lines, and `\topskip` might be the only way to figure out the proper request size. If the current depth and following height would make for a non-standard line distance, we might have a problem here. There is no obvious way to avoid it, though.

```
\def\FN@prepareoutput{%
  {\let\@elt\FN@recordinsertparam
   \xdef\FN@vsadjustlist{%
     \global\vsize=\the\vsize
     \global\topskip=\the\topskip
     \global\outputpenalty=\the\outputpenalty\relax
     \FN@list}%
   \let\@elt\FN@clearinsertparam
   \FN@list}%
```

Now we collect all boxes in the save box.

```
⟨trace⟩ \ifvoid\FN@savebox \else \PackageError{bigfoot}{\FN@savebox
⟨trace⟩   \space should be void in \string\FN@prepareoutput}{}\fi
  \global\setbox\FN@savebox\vbox{%
    \box\@cclv
    \FN@list@iterate\FN@sweepbox}}
```

`\FN@restartoutput` This is for the case where we are requesting additional material and have to cater for sizes.

```
\def\FN@restartoutput{%
```

Calculate the remaining size on this page:

```
  \dimen@=\dimexpr\FN@vsize-\glueshrink\FN@vsize\relax
```

We just pull in remaining insertions and are done.

```
⟨trace⟩ \if\foottrace8\message{sending special penalty}\fi
  \hrule\@height\z@\@depth\z@
  \unvcopy\FN@topmarkbox
  \penalty -13750
  \penalty\FN@outputflag
  \hrule\@height\z@\@depth\z@
  \FN@insertmarks
  \FN@prepareoutput
  \global\topskip-\maxdimen\relax
  \global\vsize 0.5\maxdimen
  \global\advance\FN@outervsize-\vsize
  \global\deadcycles\z@
}
```

Ok, here is the deal. If the `\FN`@truevsize is negative, we have an overfull vbox at our hand. We then start the splitting action. We take the first non-split lowest footnote block and split it to size, removing subordinate footnotes that we would not be able to maintain. We do this recursively starting by the top footnote block. It must be noted that it would be even better to start with the highest-numbered

footnote (which corresponds to the latest finished footnote in *logical* order, that in the source code), but then we get the problem that we might have to remove boxes from a footnote block that has already been split, and that is troublesome (to put it mildly) in case where the footnote block is set in paragraph mode. It's bad enough backtracking in a fixed order across footnote blocks, going back and forward would be pretty tough.

So our recursion just walks the footnote blocks once top to bottom, splitting and removing boxes that are not needed. When we recurse, we have a dichotomy between current overfull and underfull boxes. At each recursion level, we enter with an overfull configuration that establishes the breakable section for the footnote block in question.

Suppose that we have already established a previous best configuration. When we are recursing, we can only increase the badness (a non-broken insertion box contributes nothing to the overall badness or penalties, breaking the box causes a badness of 10000, minus the break penalty, plus the break badness). So there is no point in recursing if entry badness and break penalty are as large as the previous best break penalty or more.

Ok, so we construct the footnote block and try splitting it to size. If this gives us a *good* underfull version, we return that (and break out of recursion altogether). Otherwise we remember the underfull version before the break and recurse on the overfull version. If this returns an overfull version again, we return the underfull version before the break. If it returns an underfull version, we fill up the underfull version as much as possible without a change of configuration, then select the best of the last underfull and this as new local underfull. We then take the first overfull combination (even allowing a change of configuration), throw away the previous split in the next recursion level and recurse on the now thoroughly overfull combination again.

When recursion tops out, we compare the current overfull with the previous one and record the best. We prefer keeping an older overfull, all other things being equal.

Ok, so what are the data structures we maintain when going through all this folderol?

We let the insertion boxes themselves remain untouched: that makes it only a bit more complicated to maintain and access the relevant boxes, but it might come handy at one time when somebody wants to implement recursion that is not strictly top-to-bottom.

Instead we return the relevant information in the cache boxes. The total size of the cache boxes may not correspond to their actual contents: in case a split box intended for the next page is stored within them, its height is deducted from the total height of the cache box (and, consequentially, from `\FN@vsize`).

`\FN@vsize`, the amount of free space on the current page, is only updated when changing levels of recursion. Instead we maintain score of the accumulated size in the current insertion in `\FN@myvsize`.

What about the penalties and badness we collect? An unsplit footnote block carries a penalty of 0 (so we need not take into account unsplit footnote blocks at all during our bookkeeping, as they are neutral), a split footnote block is prepe-

nalized with a penalty of 10000, plus the badness of the split, plus any penalties associated with the split (limited to the $[-10000 \ldots 10000]$ range). This means that no operation on other footnote blocks can lower an already accumulated score. This in turn means that we can prune any operations leading to a worse score than the preceding best score without having to actually recurse.

This strategy will usually buy us a minimum number of split footnotes (since the penalty of 10000 is not easy to compensate) and corresponds rather closely to TEX's own idea of footnote splitting.

The following routine will analyze the last box where the results from `\splitdiscards` are stored and return the penalty associated with the breakpoint in the macro `\FN@breakpenalty`.

`\FN@getbreakpenalty`

```
\def\FN@getbreakpenalty{{\setbox\z@\lastbox
    \nointerlineskip\copy\z@
    \setbox\z@
      \vbox{\unvbox\z@
        \count@\@M
        \FN@getbreakpenaltyii
        \xdef\FN@tempinfo{\edef\noexpand\FN@breakpenalty{%
            \number\ifnum\count@=\@M \FN@defaultpenalty \else \count@\fi
          }}}}%
  \FN@tempinfo}

\def\FN@getbreakpenaltyii{%
  \ifcase
    \ifnum\lastnodetype<\z@ \m@ne\fi
    \ifnum\lastnodetype<11 \@ne\fi
    \ifnum\lastnodetype>13 \@ne\fi
    \numexpr\lastnodetype-9\relax
  \or
    \PackageError{bigfoot}{Illegal node type}{This can't happen}%
  \or
    \count@\z@ \unskip \expandafter\FN@getbreakpenaltyii
  \or
    \count@\z@ \unkern \expandafter\FN@getbreakpenaltyii
  \or
    \count@\lastpenalty
    \unpenalty \expandafter\FN@getbreakpenaltyii
  \fi}
```

`\FN@cleanpagepenalty` This is used for removing initial infinite penalties from the pagediscards: those are artifacts of the page break routine.

```
\def\FN@cleanpagepenalty{%
  \ifcase
    \ifnum\lastnodetype<\z@ \m@ne\fi
    \ifnum\lastnodetype<11 \@ne\fi
    \ifnum\lastnodetype>13 \@ne\fi
    \numexpr\lastnodetype-9\relax
```

```
  \or
    \PackageError{bigfoot}{Illegal node type}{This can't happen}%
  \or
    \skip@=\lastskip \unskip
    \expandafter \FN@cleanpagepenalty \expandafter\vskip\the
    \expandafter\skip@
  \or
    \dimen@=\lastkern \unkern
    \expandafter \FN@cleanpagepenalty \expandafter\kern\the
    \expandafter\dimen@
  \or
    \count@\lastpenalty \unpenalty
    \ifnum\count@=\FN@outputflag
      \unpenalty\expandafter\expandafter\expandafter\FN@cleanpagepenalty
    \else
      \expandafter\FN@cleanpagepenalty\expandafter
      \penalty\the\expandafter\expandafter\expandafter\count@
    \fi
  \fi
  \relax}
```

`\FN@mainsplitreturn` This is merely an argument delimiting control sequence to make it possible to figure out which recursion levels still need visiting.

```
\def\FN@mainsplitreturn{}
```

`\FN@myvsize` This is the size currently taken by this insertion.

```
\newdimen\FN@myvsize
```

`\bigfoottolerance` This specifies what footnote arrangement penalty will be accepted without looking for a better solution.

```
\newcount\bigfoottolerance
\bigfoottolerance=100
```

`\FN@getbadness` This takes a skip value of remaining space and negative stretchability and shrinkability, and then calculates `\badness` depending on how good the stretching accommodates the remaining space.

```
\def\FN@getbadness#1{%
  {\hfuzz\maxdimen\hbadness\@M\setbox\z@\hbox to\z@{\hskip-#1}}}
```

`\FN@newlevel` This is the main workhorse of `bigfoot`. It splits a particular footnote level, recursing if necessary. The level list is delimited with `\FN@mainsplitreturn`. The whole thing is looped through while the splits are being optimized. While recursing, `\FN@penalties` contains the accumulated penalities of the current split configuration: a penalty of 10000 for any split (except the main list), plus the penalty at the split points plus a 'hangover' badness for the percentage of material carried over to the following pages. If nothing is carried over, this is 0, if more is carried over, we get a penalty according to the proportion of carryover material, raised to the third power.

\footnotecarryratio The fractional variable \footnotecarryratio is used for scaling the leftover material dimensions. After scaling with \footnotecarryratio, the carried material is treated like missing material in a glue calculation, while the stretchability for this calculation is given by the total size of material before breaking. So with a setting of 1, there should always be enough stretchability, causing at most a penalty of 100. That's not very effective, so we scale this up.

The default value of 2 seems to provide a reasonable penalty for leftover material. The actual purpose for this component of the scoring is to penalize footnote blocks that seem to carry over disproportionally much material to later pages.

```
\providecommand\footnotecarryratio{2}
```

\FN@ebadness is an augmented value, but also counting in the stretch badness for one particular configuration. Ebadness does not make sense to evaluate more than temporarily: it is not passed through the levels. Since \FN@penalties is globally tampered with, its value at entry is saved in \FN@entrypenalties. Whenever we recurse or return, \FN@vsize contains the full information about the available space on the page, even though locally we use \FN@myvsize, a local value, to keep track of the locally reserved space. The only time when we need to save \FN@myvsize should be when we temporarily leave boxes in order to save the current configuration.

```
\def\FN@newlevel#1#2#3\FN@mainsplitreturn{%
  \count@\FN@cache#2%
  \ifvoid\count@
⟨trace⟩ \if\foottrace1\message{Page=\thepage #2 is empty, recursing with
⟨trace⟩   \the\FN@vsize^^J}%
⟨trace⟩   \message{Config=\unexpanded\expandafter{\FN@config}^^J}\fi
      #3\FN@mainsplitreturn
⟨trace⟩ \if\foottrace1%
⟨trace⟩   \message{Page=\thepage #2 was empty,
⟨trace⟩     returning with \the\FN@vsize^^J}%
⟨trace⟩   \message{Config=\unexpanded\expandafter{\FN@config}^^J}\fi
  \else
⟨trace⟩ \if\foottrace1\message{Entering #2 with \FN@penalties,
⟨trace⟩   \FN@vsize=\the\FN@vsize,^^J%
⟨trace⟩   Config=\unexpanded\expandafter{\FN@config}^^J}\fi
    {\def\FN@currentinsertion{#2}%
     \def\FN@currentrecursion{#3}%
     \let\FN@entryconfig\FN@config
     \let\FN@entrypenalties\FN@penalties
     \splittopskip\csname FN@ht\number#2\endcsname\relax
     \splitmaxdepth\csname FN@dp\number#2\endcsname\relax
     \hsize\csname FN@wd\number#2\endcsname\relax
     \vbadness=\@M
     \vfuzz\maxdimen
     \let\@elt\relax
     \expandafter\FN@newleveli\expandafter}%
⟨trace⟩ \if\foottrace1\message{Exiting #2 with \FN@penalties,
⟨trace⟩   \FN@vsize=\the\FN@vsize,^^J%
```

```
⟨trace⟩    Config=\unexpanded\expandafter{\FN@config}^^J}\fi
  \fi}
```

`\FN@newleveli`

```
\def\FN@newleveli{%
```

`\FN@vsize` already includes the size of the complete unsplit insertion. When we recurse, it has to reflect the correct size at the time of recursion. Rounding error problems don't permit us to accumulate any sizes in `\FN@vsize` from processing our current insertion, so we just subtract the whole insertion-related content. We'll add stuff into it when recursing.

```
    \dimen@\dimexpr\ht\count@
      \ifnum\FN@currentinsertion=\FN@savebox
        +\maxdepth
      \else
        +\dp\count@
      \fi\relax
    \global\advance\FN@vsize\dimexpr \dimen@
      *\count\FN@currentinsertion/\@m\relax\relax
```

Ok, now we are typesetting and collecting the best box. Notice that we *don't* exit this `\setbox` command until we have found the best possible split. What we `\unvbox` here, stays dormant except for the last box. When we collect configurations from cache boxes, we don't collect anything from our current box that is being assembled. So the whole action is confined within the current list that will replace the cache box after splitting. The meaning of boxes on the various levels are:

0	box0:	where stuff gets collected as tentative material to be unboxed with `\FN@removevboxes` once the insertion gets readied for shipout
0	box2:	the previous best split that was found
1	box0:	the material that gets worked off, the tail of the split
1	box2:	where the current split is assigned

The structure of 0/box2 is the head of the split, followed by `\break` penalty, followed by the tail of the split, followed by `\penalty\FN@tempbox`, followed by pairs of cache boxes and penalties indicating their box number. That way, the tail can get restored immediately into `\FN@tempbox` when using `\FN@restoreboxes`.

```
    \global\setbox\count@\vbox\bgroup\unvbox\count@
```

We calculate `\FN@myvsize` as the total space taken up by this insertion. The size of the last box is excluded since it will be split now.

```
      \ifnum\lastpenalty=\z@
        \setbox\tw@\box\voidb@x
        \setbox\z@\lastbox
        \FN@myvsize=\ifnum\lastnodetype<\z@
          \z@
        \else
          \dimexpr\dimen@-\ht\z@-\dp\z@\relax
        \fi
      \else
```

Now if the box has been split previously, we glue it back together again. Since the lower part of the split has been *subtracted* from the total in `\dimen@`, we need to put it back into the equation here. `\dimen@` contains the size of the box after padding split material has been added.

```
⟨trace⟩ \if\foottrace1\message{Regluing box 2}\fi
        \unpenalty
        \setbox\tw@\lastbox
        \setbox\z@\lastbox
        \FN@myvsize=\dimexpr\dimen@-\ht\z@-\dp\z@\relax
        \dimen@\dp\z@
        \setbox\z@{\unvbox\z@
          \setbox\z@\lastbox
          \unvbox\z@
          \unvbox\tw@}%
        \ht\z@=\dimexpr\ht\z@+\dp\z@-\dimen@\relax
        \dp\z@\dimen@
      \fi
```

Ok, size is all accounted for. Go on with optimization. Note that these definitions here are made in *inner* level, so they can't make it outside as the result of the optimization. If we drop out of here without superceding them, something's completely rotten.

```
      \edef\FN@bestcost{\number\maxdimen}%
      \let\FN@bestbadness\FN@bestcost
      \let\FN@bestconfig\@undefined
      \def\FN@bestvsize{-\maxdimen}%
      \let\FN@splitcolors\@empty
```

Ok, first attempt. One interesting feature is that we will never have to rewind the boxes from a split: we can always just glue the box together again. And apart from tentative splits which we might revert if they cause a configuration change, we will not have to bother about contributing too much. What we put in the box here can stay.

First we split to the remaining size. Since we still have all subordinate footnotes considered fully, we need at least this split size (in case of a configuration change, we will need more). After having done the initial split, we continue splitting until we get the necessary mark of the last footnote into our grasp: we can't split before that.

```
      \ifnum\FN@currentinsertion=\FN@savebox
      \else
        \edef\FN@defaultpenalty{\number-\@M}%
        \edef\FN@masterslot{\number\dp\z@}%
        \edef\FN@masterid{\FN@slotget\FN@masterslot}%
      \fi
```

Now we are building one tentative candidate for returning in `\box\z@`. It will get discarded in case that a better candidate was already found before this box completes.

```
      \setbox\z@
```

```
    \vbox\bgroup
```

Ok, now stuff gets complicated: for the first, tentatively 'optimal' split, we want to have all available page stretchability properly taken into account. So we take the box, and add the available page stretchability at the top. Note that the stretchability is registered negatively. If we are on the main vertical list, an empty page can be an acceptable option, so we add a penalty of zero to account for that. Note that any prospective true penalties will already have disappeared into the page break.

```
      \let\FN@splitcolors\@empty
      \setbox\z@\vbox{\vskip-\glueexpr(\FN@vsize-\dimexpr\FN@vsize
        \relax\@minus\glueshrink\FN@vsize)%
        *\@m/\count\FN@currentinsertion
        \penalty\z@
        \unvbox\z@
        \ifnum\FN@defaultpenalty>-\@M
          \penalty\FN@defaultpenalty\relax\nointerlineskip
          \vbox to\maxdimen{}%
        \fi}%
```

Ok, now we have pushed the additional available stretch onto the top of box 0. Now we do the actual split to minimal size. That means that we don't consider any of the shrinkability available on the page: it might still be better employed in some recursive level.

```
      \setbox\tw@\vsplit\z@ to%
        \dimexpr\FN@vsize*\@m/\count\FN@currentinsertion
        -\FN@myvsize-\splitmaxdepth
      \relax
      \ifnum\FN@defaultpenalty>-\@M
        \setbox\z@\vbox\bgroup\unvbox\z@\setbox\z@\lastbox
          \unskip
          \unpenalty
          \ifnum\lastnodetype<\z@
            \egroup \setbox\z@\box\voidb@x
          \else
            \egroup
          \fi
      \fi
```

Ok, now the top of box 2 contains unwanted additional stretchability. The easiest way to get rid of it is by adding its negation.

```
      \setbox\tw@\vbox{%
        \vskip\glueexpr(\FN@vsize-\dimexpr\FN@vsize
        \relax\@minus\glueshrink\FN@vsize)%
        *\@m/\count\FN@currentinsertion
        \unvbox\tw@\boxmaxdepth\splitmaxdepth}%
```

Ok, now rinse and repeat if we haven't reached the last footnote in the block.

```
      \ifnum\FN@currentinsertion=\FN@savebox
        \edef\FN@slaveid{\splitbotmarks\FN@slave}%
```

```
          \FN@contribute@tw@
        \else
          \ifnum0\splitbotmarks\FN@master=\FN@masterslot \else
            \loop
              \FN@contribute@tw@
              \setbox\tw@\vsplit\z@ to\z@
            \ifnum0\splitbotmarks\FN@master=\FN@masterslot
            \else
            \repeat
          \fi
          \let\FN@splitcolors\@empty
          \edef\FN@slaveid{\splitbotmarks\FN@slave}%
          \FN@contribute@tw@
        \fi
```

All of the above was necessary to ensure that we actually have the beginning of the relevant footnote in our material. From now on, we are dealing with legal splits. Ok, now we have to check whether the subordinate configuration has changed.

```
        \ifx\FN@slaveid\@empty
⟨trace⟩ \ifnum\FN@currentinsertion=\FN@savebox\else
⟨trace⟩   \errmessage{Missing slaveid in \FN@currentinsertion}\fi
          \edef\FN@slaveid{\number0\topmarks\FN@slave}%
        \fi
        \ifnum\numexpr\FN@slaveid+\@ne<\FN@masterid
          \let\FN@next\FN@slaveid
        \else
          \let\FN@next\@empty
        \fi
```

At this point of time, we have `\FN@masterid` set properly for our purposes. It is to be used for returning any *tail* part of a box. `\FN@slaveid` is by necessity not empty. If any footnote has had its mark broken off, its id must be in the open range between `\FN@slaveid` and `\FN@masterid`. So a nonempty value of `\FN@next` at this point of time indicates that we have to cater for a different configuration rather than the currently cached one.

```
        \FN@splitfurther}
```

`\FN@vsizerecurse` This fixes the vertical size up and recurses once.

```
\def\FN@vsizerecurse{%
    \global\advance\FN@vsize
      -\dimexpr\FN@myvsize*\count\FN@currentinsertion/\@m \relax\relax
    \let\@elt\FN@newlevel
    \FN@currentrecursion\FN@mainsplitreturn
    \let\@elt\relax
    \global\advance\FN@vsize
      \dimexpr\FN@myvsize*\count\FN@currentinsertion/\@m \relax\relax}
```

1.5.1 Main label for reconsideration

When we are here, then there is not yet a split in the next footnote blocks scheduled. We might have to restitch stuff together here, though.

\FN@splitfurther

```
\def\FN@splitfurther{%
  \ifx\FN@next\@empty \else \let \FN@slaveid\FN@next \fi
```

Ok, if our configuration now differs from the last one for which we have cache boxes set up, we have to reconfigure. It it doesn't, we just stitch the boxes together again in order to have correct size info.

```
  \let\FN@next\FN@config
  \xdef\FN@config{%
    \@elt{\FN@slaveid}%
    {\FN@masterid}%
    \FN@entryconfig}%
  \ifx\FN@next\FN@config
    \let\@elt\FN@rejoin
  \else
    \let\@elt\FN@reconfig
  \fi
  \FN@currentrecursion
  \let\@elt\relax
```

Ok, now we check whether the current configuration is a match for the best previous one. Also we calculate the badness of the current situation.

```
  \xdef\FN@penalties{\number\FN@entrypenalties}%
  \FN@checkcurrent
```

No point in recursing if we can't beat the current best one. However, if we find a forced break, this is considered perfect as long as we are not overfull. Note that recursion can only increase the badness if we are still underfull here, so there is no point in using `\FN@penalties` as the deciding factor of whether there may be a point in recursing: the current ebadness (which is never less than the badness) already is minimal.

```
  \ifnum
    \ifdim\skip@>\z@ \FN@ebadness \else \FN@penalties \fi
    >\FN@bestcost\relax
⟨trace⟩    \if\foottrace1%
⟨trace⟩    \message{no recursion: \skip@=\the\skip@,
⟨trace⟩      \noexpand\FN@bestvsize=\FN@bestvsize,
⟨trace⟩      \noexpand\FN@ebadness=\FN@ebadness,
⟨trace⟩      \noexpand\FN@penalties=\FN@penalties,
⟨trace⟩      \noexpand\FN@bestcost=\FN@bestcost.}\fi
  \else
⟨trace⟩    \if\foottrace1%
⟨trace⟩    \message{before recursion: \skip@=\the\skip@,
⟨trace⟩      \noexpand\FN@bestvsize=\FN@bestvsize,
⟨trace⟩      \noexpand\FN@ebadness=\FN@ebadness,
```

```
⟨trace⟩       \noexpand\FN@penalties=\FN@penalties,
⟨trace⟩       \noexpand\FN@bestcost=\FN@bestcost.^^J
⟨trace⟩       recurse with \noexpand\FN@penalties=\number\FN@entrypenalties.}\fi
    \xdef\FN@penalties{\number\FN@entrypenalties}%
    \FN@vsizerecurse
    \FN@checkcurrent
⟨trace⟩     \if\foottrace1%
⟨trace⟩     \message{after recursion: \skip@=\the\skip@,
⟨trace⟩       \noexpand\FN@bestvsize=\FN@bestvsize,
⟨trace⟩       \noexpand\FN@ebadness=\FN@ebadness,
⟨trace⟩       \noexpand\FN@penalties=\FN@penalties,
⟨trace⟩       \noexpand\FN@bestcost=\FN@bestcost.}\fi
  \fi
```

Don't look further if we had a forced break or are overfull or are at the end of the list.

```
  \ifcase
    \ifnum\FN@breakpenalty>-\@M \else \@ne \fi
    \ifvoid\z@ \@ne \fi
    \ifnum\badness<\@MM \else \@ne \fi
    \tw@
  \or
    \expandafter \FN@returnbest
  \else
    \FN@mayberecordbest
    \setbox\tw@\vsplit\z@ to\z@
    \edef\FN@next{\splitbotmarks\FN@slave}%
    \FN@contribute@tw@
    \expandafter \FN@splitfurther
  \fi}
```

`\FN@checkcurrent` Check out the badness of the current configuration. The last box on the list is the material constituting the discardable material after a split.

```
\def\FN@checkcurrent{%
  \FN@getbreakpenalty
  \ifnum\FN@breakpenalty<-\@M
    \edef\FN@breakpenalty{\number-\@M}%
  \fi
  \ifnum\FN@currentinsertion=\FN@savebox
  \else
    \ifdim\FN@specific\FN@currentinsertion\footnotecarryratio\p@>\z@
      \skip@
      \ifdim\FN@specific\FN@currentinsertion\footnotecarryratio\p@>\p@
          \dimexpr\ht\z@+\dp\z@\relax
          \@plus-\dimexpr((\FN@myvsize+\ht\z@+\dp\z@)
                          *\p@/\dimexpr
                          \FN@specific\FN@currentinsertion
                          \footnotecarryratio\p@)\relax
      \else
          \FN@specific\FN@currentinsertion
```

```
          \footnotecarryratio
          \dimexpr\ht\z@+\dp\z@\relax
          \@plus-\dimexpr\FN@myvsize+\ht\z@+\dp\z@\relax
      \fi
      \relax
      \FN@getbadness\skip@
      \xdef\FN@penalties{\number\numexpr\FN@penalties+\badness}%
    \fi
  \fi
  \skip@\glueexpr\FN@vsize-\FN@myvsize
   *\count\FN@currentinsertion/\@m\relax
  \FN@getbadness\skip@
  \xdef\FN@penalties{\number\numexpr\FN@penalties+%
    \FN@breakpenalty+\@M}%
  \ifnum\badness>\@M
    \edef\FN@ebadness{\number\numexpr\maxdimen-\@ne}%
  \else
    \ifnum\badness=\@M
      \ifdim\skip@<\vsize
        \edef\FN@ebadness{\number\numexpr\maxdimen-\tw@}%
      \else
        \edef\FN@ebadness{\number\numexpr\maxdimen}%
      \fi
    \else
      \edef\FN@ebadness{\number\numexpr
        \FN@penalties+\badness
        \ifdim\FN@specific\FN@currentinsertion\FootnoteMinimum>\FN@myvsize
          1000000
        \fi
        }%
    \fi
  \fi
  \dimen@\glueexpr\FN@bestvsize\relax
}
```

`\FN@checkforbest` This generates 2 if the old stored best is better, 1 if the current variation is better.

```
\def\FN@checkforbest{%
    \ifnum\FN@breakpenalty>-\@M \else
      \ifnum\badness>\@M \else
         \@ne
      \fi
    \fi
```

Ok, so the split was not as good as to cause us to return immediately, and it also was not the last opportunity for a split (which again would make us return immediately). So we check if it is at least better than the last one, in which case we need to replace the previous best.

```
    \ifnum\FN@bestcost>\FN@ebadness \@ne\fi
    \ifnum\FN@bestcost<\FN@ebadness \tw@\fi
    \ifdim\skip@<\z@
```

```
      \ifdim\dimen@<\skip@ \@ne \fi \tw@
    \fi
    \ifdim\dimen@>\skip@ \@ne\fi \tw@}
```

`\FN@mayberecordbest` This checks whether the current configuration is better than a previously saved one. If it is, the previous configuration gets replaced. The cache boxes itself are copied, not voided in the process for efficiency reasons.

```
\def\FN@mayberecordbest{%
```

If the current break is forced and the page is not overfull, we take the break.

```
  \ifcase
    \FN@checkforbest
  \or
    \xdef\FN@tempinfo{\def\noexpand\FN@bestvsize{\the\skip@}%
      \def\noexpand\FN@bestcost{\FN@ebadness}%
      \def\noexpand\FN@bestbadness{\number\FN@penalties}%
      \def\noexpand\FN@bestconfig{\FN@config}%
      \def\noexpand\FN@bestslaveid{\FN@slaveid}%
      \def\noexpand\FN@bestsplitcolors{\FN@splitcolors}%
      \def\noexpand\FN@breakpenalty{\FN@breakpenalty}%
      \FN@myvsize=\the\FN@myvsize\relax}%
    \global\setbox\FN@tempbox\box\z@
    \egroup
    \FN@tempinfo
    \let\FN@splitcolors\FN@bestsplitcolors
    \let\FN@slaveid\FN@bestslaveid
```

Now all relevant info has been retrieved, and we collect the best box info in `\box\tw@`. The structure of the information is as follows: it starts with the current box in split form, first the tail, then the start of the current split box. This is then followed by a zero kern, and then by pairs of boxes and penalties indicating the swept box.

```
    \setbox\tw@\vbox{%
```

We don't need to place master/slave marks here: the necessary information is available outside in the `\FN@masterslot` and `\FN@slaveid` info and gets attached afterwards.

```
      \copy\z@\break\nointerlineskip
      \copy\FN@tempbox\penalty\FN@tempbox
      \let\@elt\FN@copycachebox
      \FN@currentrecursion}%
    \setbox\z@
    \vbox\bgroup
      \unvbox\z@
      \setbox\z@\box\FN@tempbox
  \fi}
```

`\FN@returnbest`

```
\def\FN@returnbest{%
  \ifcase\FN@checkforbest
```

```
\or
  \xdef\FN@tempinfo{\def\noexpand\FN@bestvsize{\the\skip@}%
    \def\noexpand\FN@bestcost{\FN@ebadness}%
    \def\noexpand\FN@bestbadness{\number\FN@penalties}%
    \def\noexpand\FN@bestconfig{\FN@config}%
    \def\noexpand\FN@bestslaveid{\FN@slaveid}%
    \def\noexpand\FN@bestsplitcolors{\FN@splitcolors}%
    \def\noexpand\FN@breakpenalty{\FN@breakpenalty}%
    \FN@myvsize=\the\FN@myvsize\relax}%
  \global\setbox\FN@tempbox\box\z@
  \egroup
  \FN@tempinfo
  \let\FN@splitcolors\FN@bestsplitcolors
  \let\FN@slaveid\FN@bestslaveid
  \global\FN@vsize\FN@bestvsize\relax
```

Now all relevant info has been retrieved, and we collect the best box info in `\box\tw@`. The structure of the information is as follows: it starts with the current box in split form, first the tail, then the start of the current split box. This is then followed by a zero kern, and then by pairs of boxes and penalties indicating the swept box.

```
\or
  \global\let\FN@config\FN@bestconfig
  \global\FN@vsize\FN@bestvsize
  \global\let\FN@penalties\FN@bestbadness
  \egroup
```

Restore the saved configuration.

```
  \setbox\z@\vbox{\unvbox\tw@ \FN@restoreboxes}%
  \let\FN@splitcolors\FN@bestsplitcolors
  \let\FN@slaveid\FN@bestslaveid
  \unvbox\z@
  \setbox\z@\lastbox
\fi

\ifnum\FN@currentinsertion=\FN@savebox
\else
  \setbox\z@\vbox{%
    \prevdepth\dp\z@
    \unvbox\z@
    \ifvoid\FN@tempbox
    \else
      \global\setbox\FN@tempbox\vbox{%
        \marks\FN@master{\FN@masterslot}%
        \marks\FN@slave{\FN@slaveid}%
        \FN@coloraftersplit\FN@splitcolors
        \FN@specific\FN@currentinsertion\FN@afterbreak
        \nobreak
        \unvbox\FN@tempbox}%
      \FN@specific\FN@currentinsertion\FN@beforebreak
      \ht\FN@tempbox
```

```
        \dimexpr\ht\FN@tempbox+\dp\FN@tempbox-\FN@masterslot sp\relax
        \dp\FN@tempbox\FN@masterslot sp\relax
        \wd\FN@tempbox\maxdimen
      \fi
    \ifdim\prevdepth<\splitmaxdepth
       \hrule\@height-\prevdepth \@width\z@
             \@depth \splitmaxdepth \relax \fi}%
    \ht\z@=\dimexpr\ht\z@+\dp\z@-\FN@masterslot sp\relax
    \dp\z@=\FN@masterslot sp
  \fi
  \nointerlineskip \box\z@
```

If nothing is to be carried over, we just finish our assignment to the cache box and return.

```
  \ifvoid\FN@tempbox \egroup
```

If not, we add the carried-over box to the list, flag it with a `\nobreak`, and subtract its size from the finished box. Please note that the `\expandafter` chain will expand just `\cmd\dimen@`, but everything following it will be evaluated only after `\egroup`, thus using the new height of the box.

```
  \else
    \dimen@-\dimexpr\ht\FN@tempbox+\dp\FN@tempbox\relax
    \nointerlineskip\box\FN@tempbox
    \nobreak
    \expandafter\egroup
    \expandafter\ht\expandafter\count@\expandafter\dimexpr
    \the\dimen@+\ht\count@\relax
  \fi
}
```

`\FN@contribute@tw@` This will go from the state where we have the previous `\splitdiscards` struttified on the current list some material split off from box 0 in box 2 to a state where box 2 is contributed to the current list.

```
\def\FN@contribute@tw@{%
```

First we change the current colors if we have any in our group. Not sure if this is entirely correct.

```
  \begingroup\edef\FN@next{\splitbotmarks\FN@color}%
  \ifx\FN@next\@empty \endgroup\else \endgroup
    \edef\FN@splitcolors{\splitbotmarks\FN@color}\fi
```

If the last box is void, there is no previous split to reconstitute.

```
  \setbox4\lastbox
  \ifvoid4 \setbox4\vbox{\splitdiscards}%
    \setbox\tw@\vbox{\unvbox\tw@\boxmaxdepth\splitmaxdepth}%
  \else
```

Now the last box is a strut. We remove its outer dimensions from the total account, and then add back its natural dimensions after which we pour it back into the current list.

```
    \advance\FN@myvsize-\dimexpr\ht4+\dp4\relax
```

```
    \setbox4\vbox{\unvbox4}%
    \advance\FN@myvsize\dimexpr\ht4+\dp4\relax
    \unvbox4
```

We want to contribute box 2 back without any topskip glue, so we manually remove any such glue by splitting an empty box off.

```
    \setbox4\vbox{\splitdiscards}%
    \setbox\tw@\vbox{\break\unvbox\tw@}%
    {\splittopskip-\maxdimen \setbox\tw@\vsplit\tw@ to\z@}%
```

Notice the effect of TEX's special box scope rules: box 2 assigned just right now will be affected by the split. The result of the split will be an empty box that will temporarily overwrite box 2 within the group, but will be restored back to the split result on exit. In this manner, any topskip glue will have disappeared. After the split, box 2 is set to the natural depth and height of its contents.

We now add a sort of strut by putting all the discarded material inside of a box that creates the proper size. If this split is taken, the box is adjusted to have a full depth of `\splitmaxdepth`, and we take this into account.

```
  \fi
  \ht4-\dp\tw@
  \dp4\ifdim\dp\tw@<\splitmaxdepth \splitmaxdepth \else \dp\tw@ \fi
  \advance\FN@myvsize\dimexpr \ht\tw@+\dp4\relax
  \unvbox\tw@
  \nointerlineskip
  \box4 }
```

`\FN@uncontribute@tw@` This is just the opposite: after a split, we revert its effects again.

```
\def\FN@uncontribute@tw@{%
  \ifvoid\tw@ \else
  \setbox\tw@\vbox{\unvbox\tw@\splitdiscards}%
  \setbox\z@\vbox{\break\unvbox\z@}%
  {\splittopskip-\maxdimen \setbox\z@\vsplit\z@ to\z@}%
  \setbox\z@\vbox{\unvbox\tw@\unvbox\z@}\fi}
```

`\FN@reconfig` This reconfigures the insertion cache to contain only the boxes that belong to this page. If the insertion box is empty, we can skip all the folderol. If it isn't, we empty the cache box (the number of which we place in `\count@`) and add its size back to `\FN@vsize`.

```
\def\FN@reconfig#1#2{\ifvoid#2%
⟨trace⟩  \ifvoid\FN@cache#2\else
⟨trace⟩    \errmessage{\FN@cache#2 should be void}\fi
  \else
  \count@\FN@cache#2%
  \ifvoid\count@\else
    \global\advance\FN@vsize
      \glueexpr(\ht\count@+\dp\count@)*\count#2/\@m+\skip#2\relax
    \global\setbox\count@ \box\voidb@x
  \fi
```

Ok, now we have emptied the cache and readjusted the size. We now fill the cache

by first copying the insertion into it.

```
  \global\setbox\count@\vbox\bgroup\vbox\bgroup\unvcopy#2%
    \let\@elt\FN@removecheck
    \FN@retainkept
```

Now if nothing was retained, we void the cachebox.

```
    \ifvoid\z@ \egroup\egroup \global\setbox\count@ \box\voidb@x
```

Otherwise, we combine all the boxes that remain on the page.

```
    \else \def\FN@masterinsert{#2}%
      \FN@assembleboxes\global\setbox\count@\box\z@\egroup
      \nointerlineskip\box\count@\egroup
```

Note that now all footnote boxes are collected into a single vbox, followed by the last footnote box as another vbox. Now we just need to reduce the available size on the page by the height of the assembled material:

```
    \global\advance\FN@vsize
      -\glueexpr(\ht\count@+\dp\count@)*\count#2/\@m+\skip#2\relax
  \fi\fi}
```

`\FN@rejoin` This glues together cache boxes that have been split, without regenerating them. This saves a lot of time as compared to `\FN@reconfig`.

```
\def\FN@rejoin#1#2{%
⟨trace⟩ \if\foottrace1\message{^^JRejoining #2}\fi
  \count@\FN@cache#2%
  \ifvoid\count@\else
    \global\advance\FN@vsize\dimexpr
       (\ht\count@+\dp\count@)*\count#2/\@m\relax
    \global\setbox\count@\vbox{%
      \unvbox\count@
      \ifnum\lastpenalty>\z@
        \unpenalty
        \setbox\tw@\lastbox
        \setbox\z@\lastbox
        \dimen@\dp\z@
        \setbox\z@\vbox{%
          \unvbox\z@
          \setbox\z@\lastbox
          \unvbox\z@
          \unvbox\tw@}%
        \ht\z@=\dimexpr\ht\z@+\dp\z@-\dimen@\relax
        \dp\z@=\dimen@
        \nointerlineskip
        \box\z@
      \fi}%
    \global\advance\FN@vsize-\dimexpr
        (\ht\count@+\dp\count@)*\count#2/\@m\relax
  \fi}}
```

`\FN@retainkept` This relies on `\@elt` being set to `\FN@removecheck` which expands to `\@ne` if `\box0` is strictly between the two values from an entry of `\FN@config`, which means that

it is material that should get moved to the next page. In that case, we recurse while dropping the box in question. Otherwise we keep it. Recursion bottoms out when there are no boxes left. The function leaves the last retained box in box 0; if there are no boxes to be retained, this will be void.

```
\def\FN@retainkept{%
  \setbox\z@\lastbox
  \ifcase
    \ifvoid\z@\m@ne\fi \FN@config\z@
⟨trace⟩  \if\foottrace8\message{^^J\string\FN@retainkept:
⟨trace⟩     retaining Id \FN@slotget{\number\dp\z@}}\fi
⟨trace⟩  \if\foottrace{16}{\showboxdepth4 \showboxbreadth400
⟨trace⟩     \tracingonline=\@ne\showbox\z@}\fi
    {\FN@retainkept \nointerlineskip \box\z@}%
  \or
⟨trace⟩  \if\foottrace8\message{^^J\string\FN@retainkept:
⟨trace⟩     dropping Id \FN@slotget{\number\dp\z@}}\fi
⟨trace⟩  \if\foottrace{16}{\showboxdepth4 \showboxbreadth400
⟨trace⟩     \tracingonline=\@ne\showbox\z@}\fi
    \FN@retainkept
⟨trace⟩  \else
⟨trace⟩  \ifnum\lastnodetype>\m@ne
⟨trace⟩    \errmessage{Unexpected node \number\lastnodetype}\fi
  \fi}
```

Well, as the last measure, we change the output routine to our new routine.

```
\let\output\FN@output
```

Ok, here is debugging code intercepting all calls of the regular output routine and reporting its entry and exit states.

```
⟨trace⟩ \newtoks\FN@tr@output
⟨trace⟩ \FN@tr@output\output
⟨trace⟩ \output{\if\foottrace8{%
⟨trace⟩   \setbox\z@\vbox{%
⟨trace⟩     \message{Calling regular output with
⟨trace⟩       \outputpenalty=\the\outputpenalty, box255 as}%
⟨trace⟩     \showbox\@cclv
⟨trace⟩     \the\FN@tr@output
⟨trace⟩     \message{Returning from regular output with
⟨trace⟩     \ifnum\lastnodetype<\z@
⟨trace⟩       empty vertical list.\else vlist:}}%
⟨trace⟩   \showbox\z@
⟨trace⟩   \unvbox\z@{{\fi}}}\else\the\FN@tr@output\fi}
⟨trace⟩ \let\output\FN@tr@output
```

If the footnote type “default” has not been declared by the time the document starts, we do so at the start of the document. Unfortunately, by this time the initialization code in `manyfoot`'s own `\AtBeginDocument` hook has already run, so we manually run the initialization hook just for the command we inserted ourselves.

```
\def\FN@maybestart#1#2#3{\ifx#3\relax
  \csname MFL@start#1\endcsname{#2}\fi#3}
\@onlypreamble\FN@maybestart
\AtBeginDocument{\@ifundefined{footinsdefault}%
  {\newfootnote[plain]{default}%
    {\let\@elt\FN@maybestart
      \MFL@list\relax}%
  }{}%
```

And since LaTeX's macros are inferior to our own (and would probably not match too well), we reroot them to the default footnote style.

```
  \def\@footnotetext{\Footnotetextdefault{}}%
  \def\p@footnotedefault{\p@footnote}%
}
```
⟨/style⟩

2 Various driver files

The installer, in case it is missing. If it is to be used via `make`, we don't specify an installation path, since

```
make install
```

is supposed to cater for the installation itself.

```
⟨installer⟩ \input docstrip
⟨installer & make⟩ \askforoverwritefalse \nopreamble
⟨installer⟩ \generate{
⟨installer⟩     \file{bigfoot.drv}{\from{bigfoot.dtx}{driver}}
⟨installer⟩     \file{perpage.drv}{\from{perpage.dtx}{driver}}
⟨installer⟩     \file{suffix.drv}{\from{suffix.dtx}{driver}}
⟨installer&!make⟩     \usedir{tex/latex/bigfoot}
⟨installer⟩     \file{bigfoot.sty}{\from{bigfoot.dtx}{style}}
⟨installer⟩     \file{perpage.sty}{\from{perpage.dtx}{style}}
⟨installer⟩     \file{suffix.sty}{\from{suffix.dtx}{style}}
⟨installer⟩ }
⟨installer⟩ \endbatchfile
```