RinohType

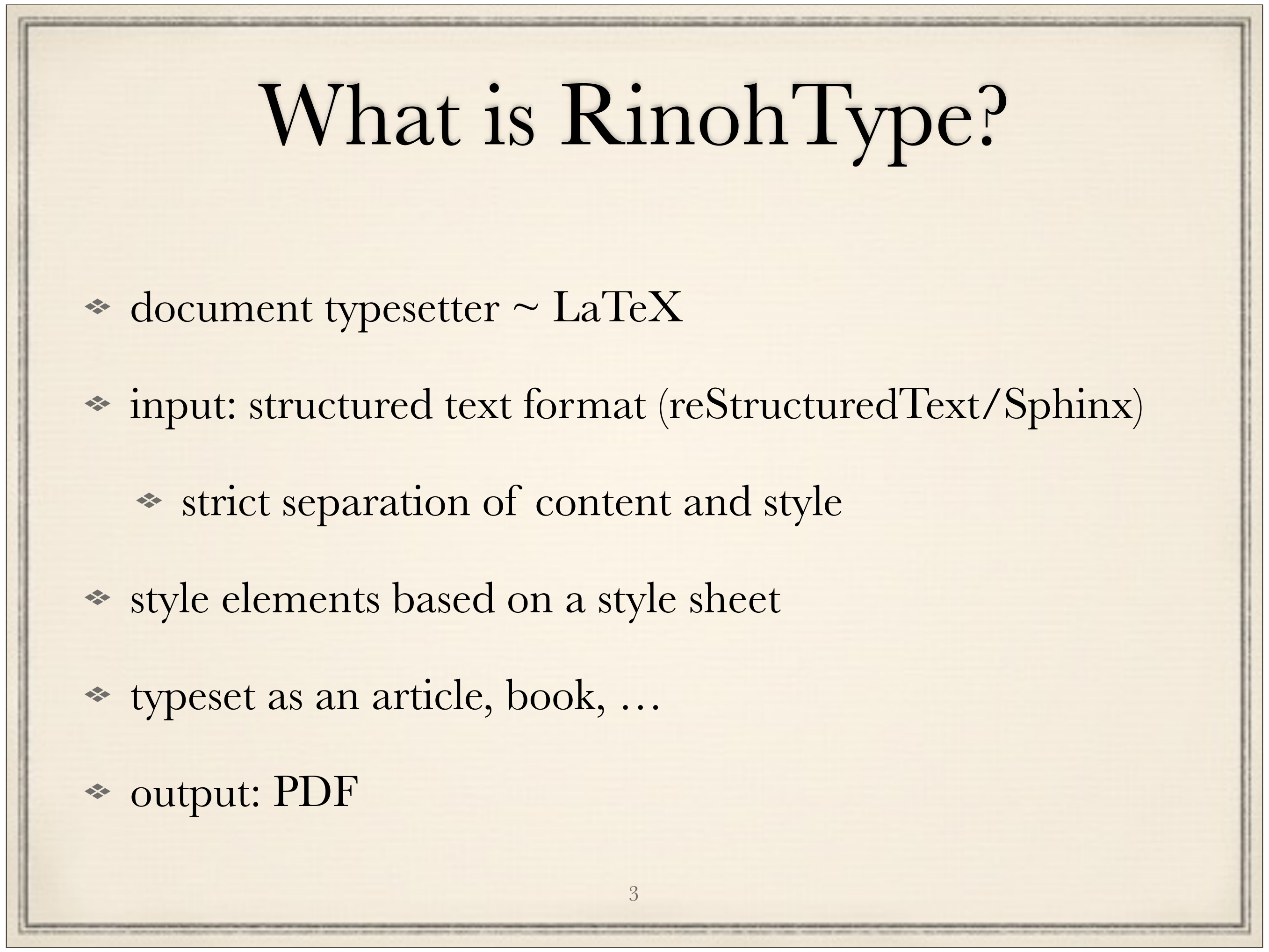
A Document Processor inspired by LaTeX

Brecht Machiels

EuroPython 2015

About the Speaker

- Brecht Machiels, *1982
- enjoying computers since about 1986
- Ph.D. in micro-electronics
- programming C, C++, Python professionally and for fun



What is RinohType?

- document typesetter ~ LaTeX
- input: structured text format (reStructuredText/Sphinx)
  - strict separation of content and style
- style elements based on a style sheet
- typeset as an article, book, …
- output: PDF

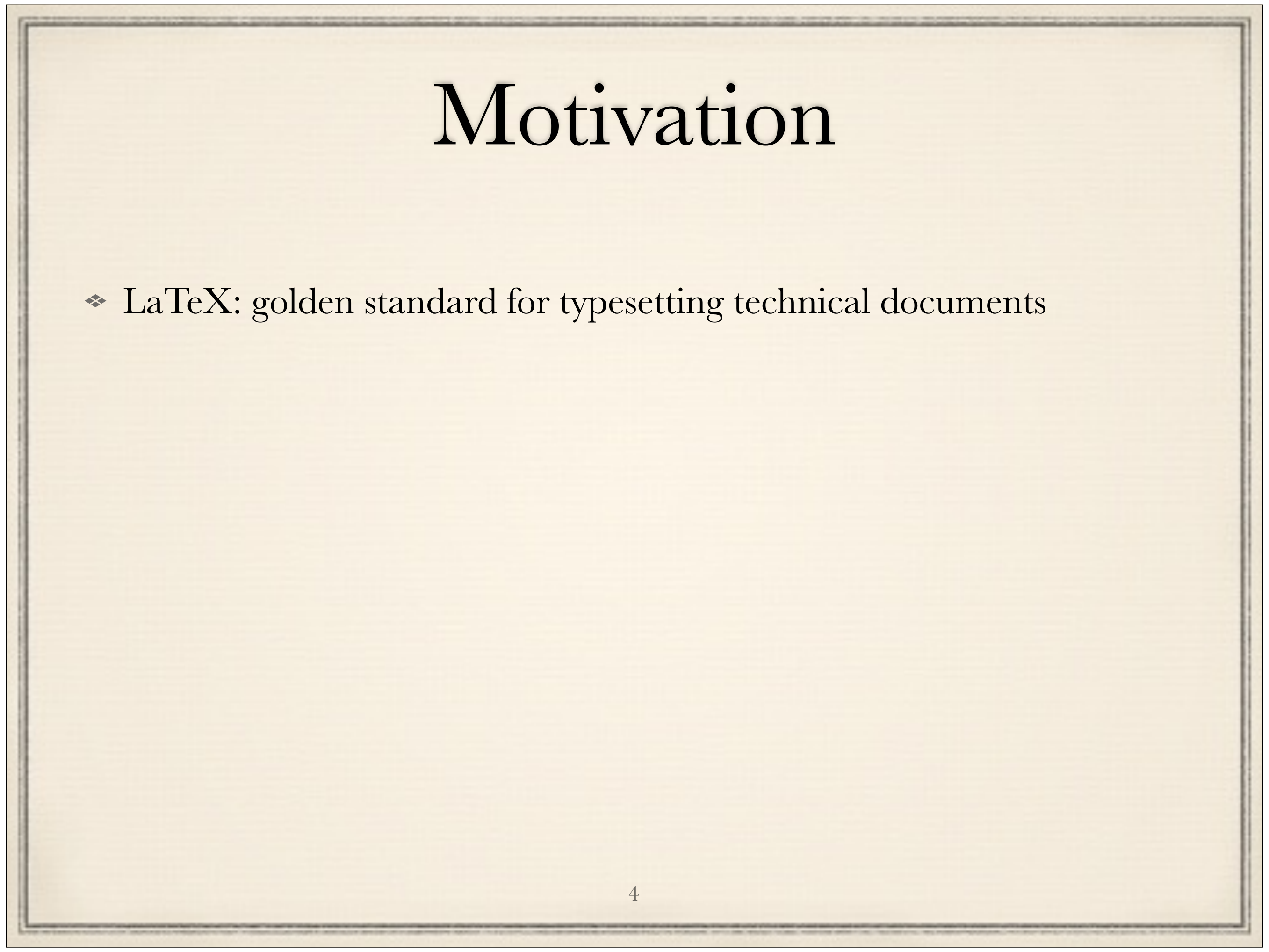
Q: who has used LaTeX?

Motivation

Who else hates LaTeX?

story?
LaTeX errors: http://www.cs.utexas.edu/~witchel/errorclasses.html
http://www-rohan.sdsu.edu/~aty/bibliog/latex/gripe.html
http://blog.computationalcomplexity.org/2011/07/problems-of-latex.html

MacTeX (TeX Live) download 2.5 GB (includes a lot)
MacTeX Basic TeX download 110 MB



Motivation

- LaTeX: golden standard for typesetting technical documents

4

Who else hates LaTeX?

story?

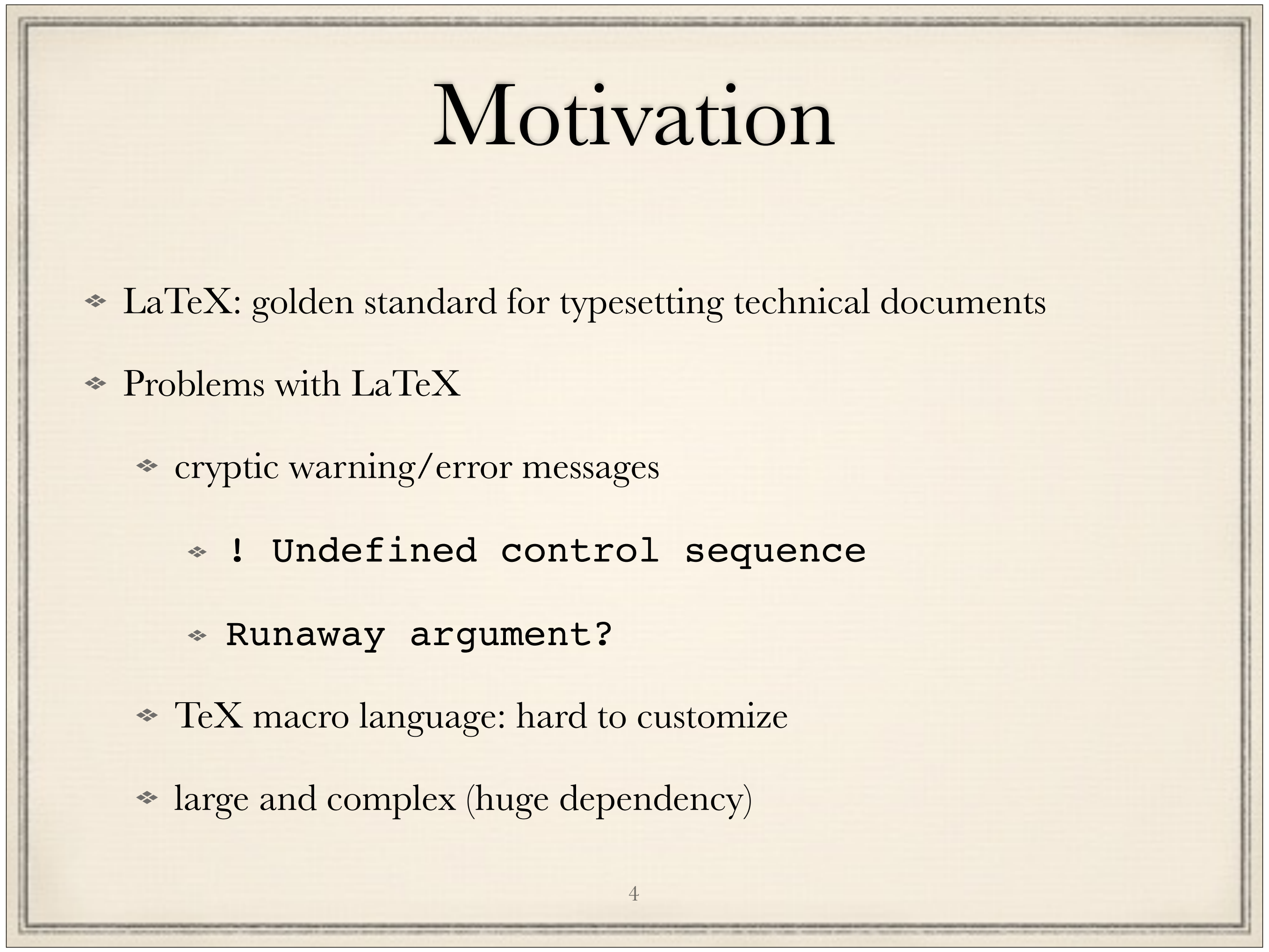
LaTeX errors: http://www.cs.utexas.edu/~witchel/errorclasses.html

http://www-rohan.sdsu.edu/~aty/bibliog/latex/gripe.html

http://blog.computationalcomplexity.org/2011/07/problems-of-latex.html

MacTeX (TeX Live) download 2.5 GB (includes a lot)

MacTeX Basic TeX download 110 MB



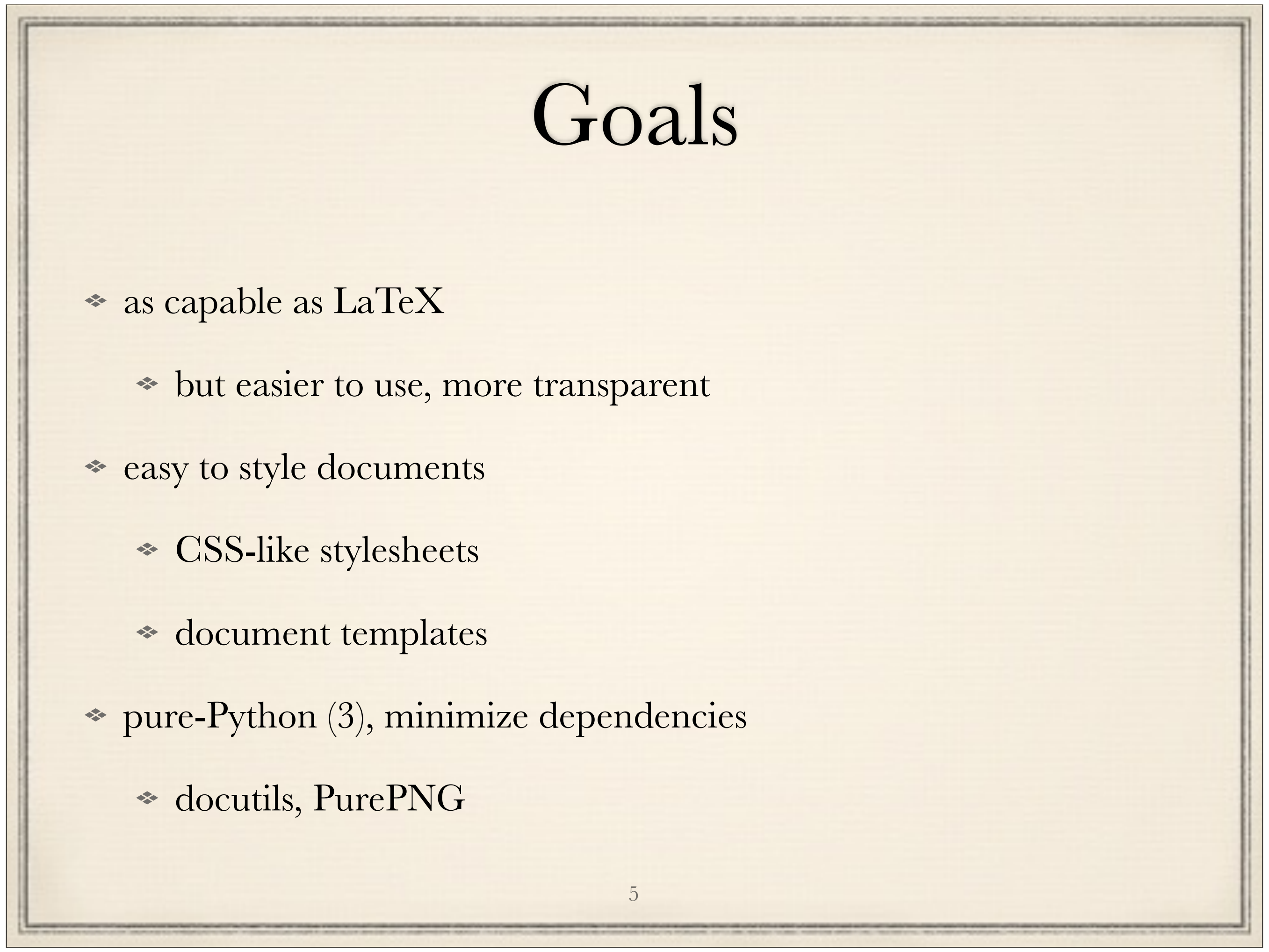
Motivation

- LaTeX: golden standard for typesetting technical documents
- Problems with LaTeX
  - cryptic warning/error messages
    - `! Undefined control sequence`
    - `Runaway argument?`
  - TeX macro language: hard to customize
  - large and complex (huge dependency)

Who else hates LaTeX?

story?
LaTeX errors: http://www.cs.utexas.edu/~witchel/errorclasses.html
http://www-rohan.sdsu.edu/~aty/bibliog/latex/gripe.html
http://blog.computationalcomplexity.org/2011/07/problems-of-latex.html

MacTeX (TeX Live) download 2.5 GB (includes a lot)
MacTeX Basic TeX download 110 MB



Goals

- as capable as LaTeX
  - but easier to use, more transparent
- easy to style documents
  - CSS-like stylesheets
  - document templates
- pure-Python (3), minimize dependencies
  - docutils, PurePNG

modern replacement for LaTeX

document templates:

- provide standard (configurable) templates
- easy to create new ones

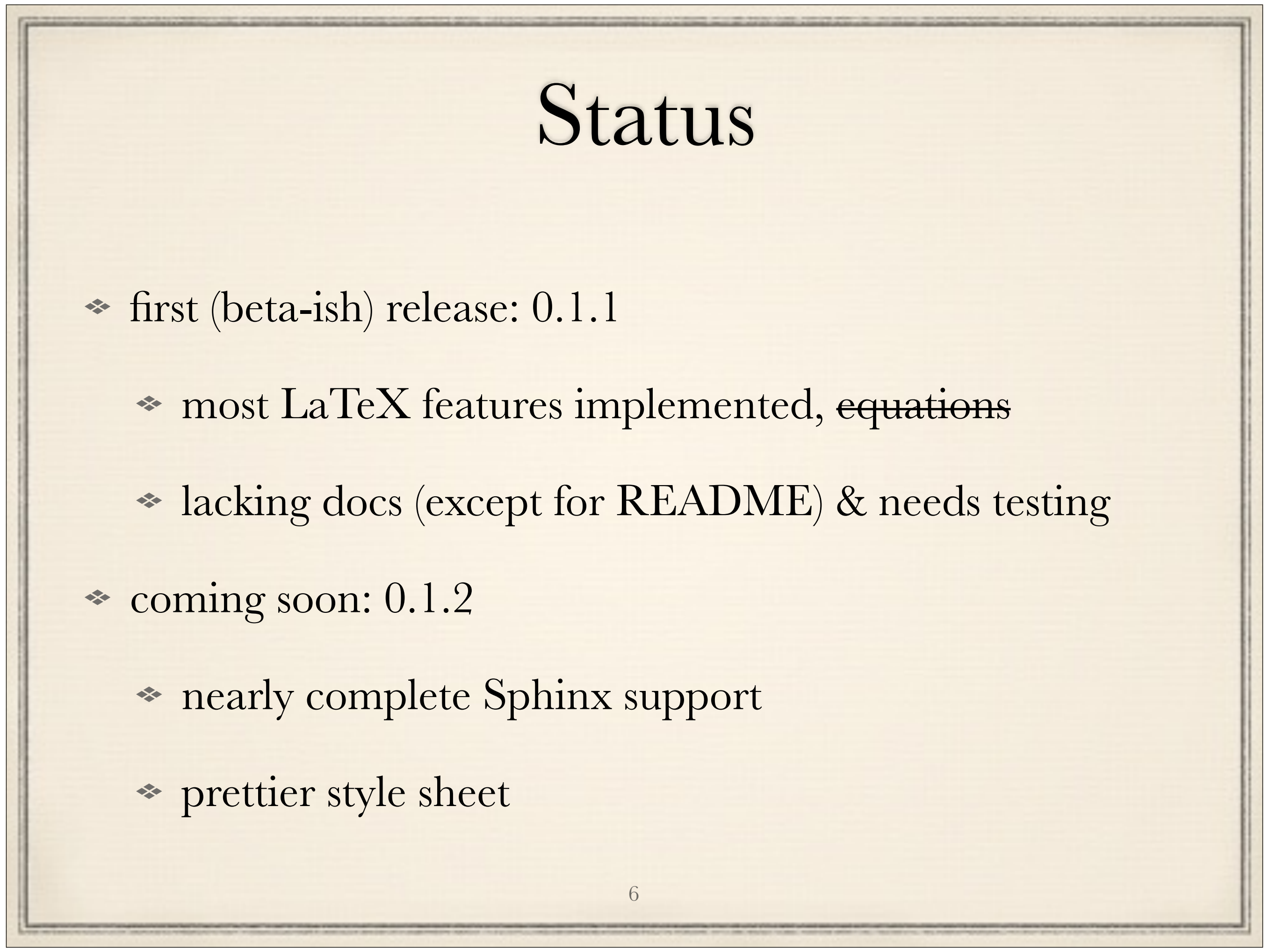
Status

beta: fairly feature-complete… needs testing, debugging, documenting

Status

- first (beta-ish) release: 0.1.1
  - most LaTeX features implemented, ~~equations~~
  - lacking docs (except for README) & needs testing

beta: fairly feature-complete… needs testing, debugging, documenting



Status

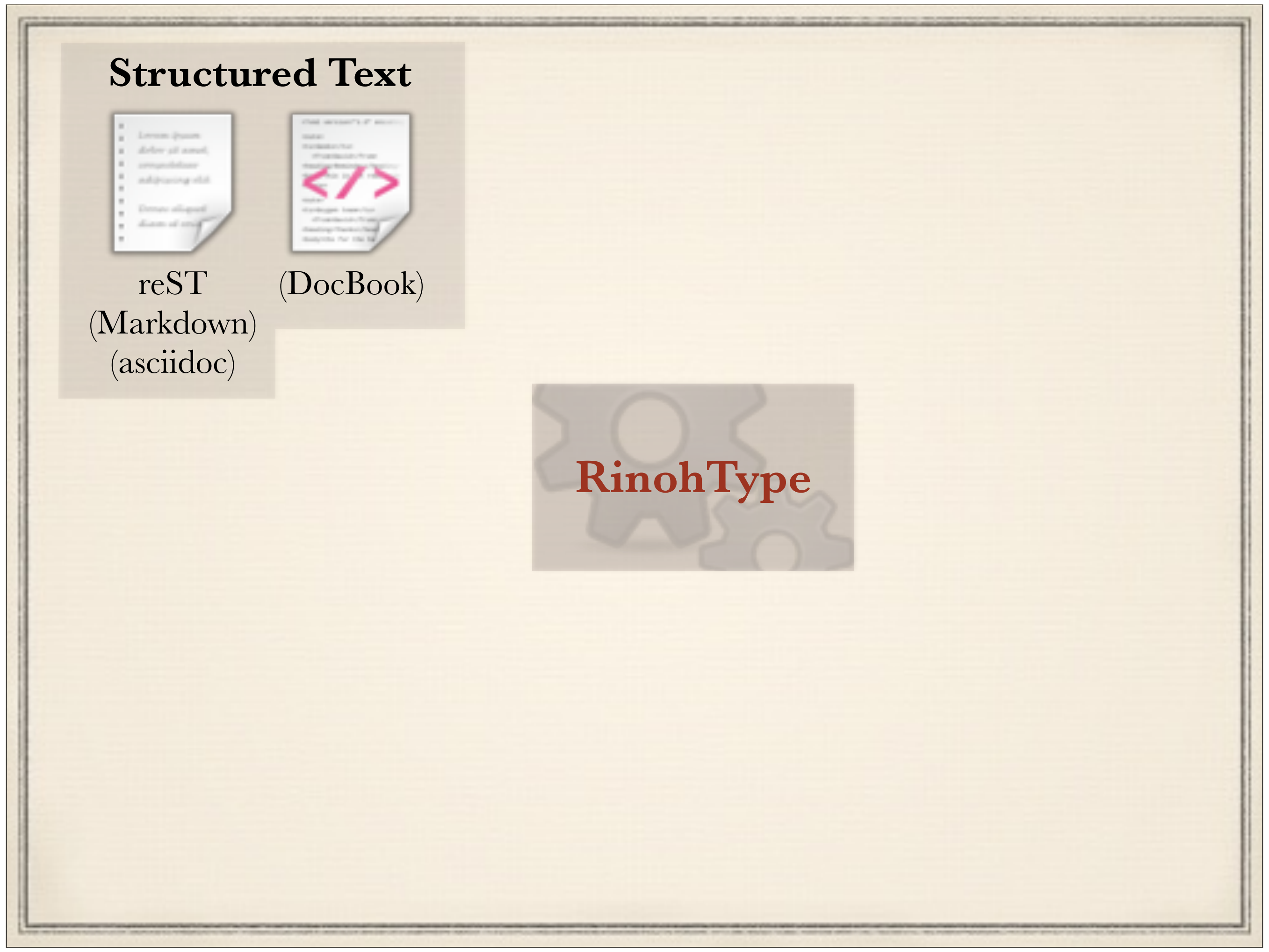
- first (beta-ish) release: 0.1.1
  - most LaTeX features implemented, ~~equations~~
  - lacking docs (except for README) & needs testing
- coming soon: 0.1.2
  - nearly complete Sphinx support
  - prettier style sheet

beta: fairly feature-complete… needs testing, debugging, documenting

How to Use

RinohType

input files (similar to LaTeX)
structured text references images
RinohType comes with some style sheets and fonts
common font formats supported



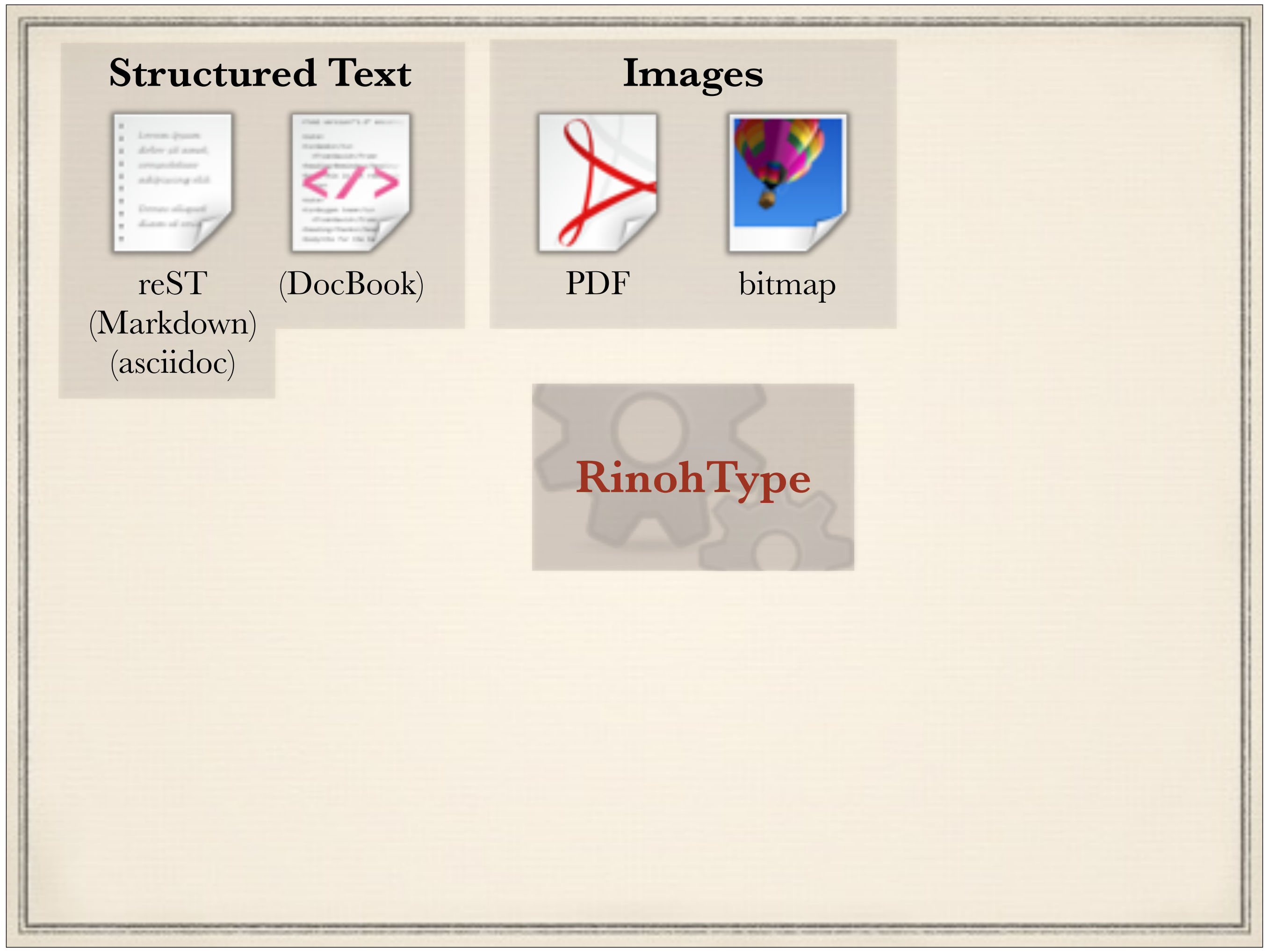
Structured Text

reST
(Markdown)
(asciidoc)

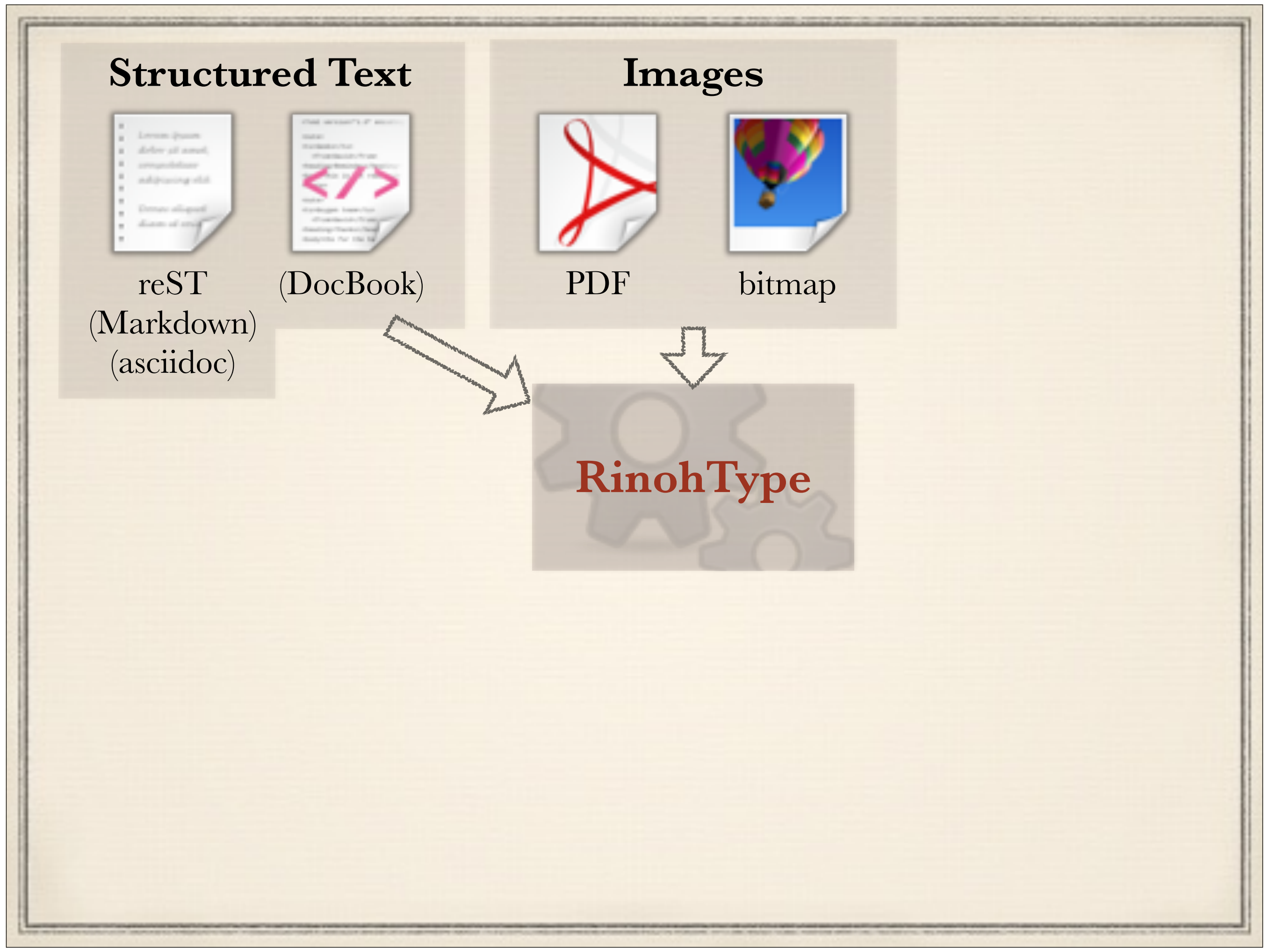
(DocBook)

RinohType

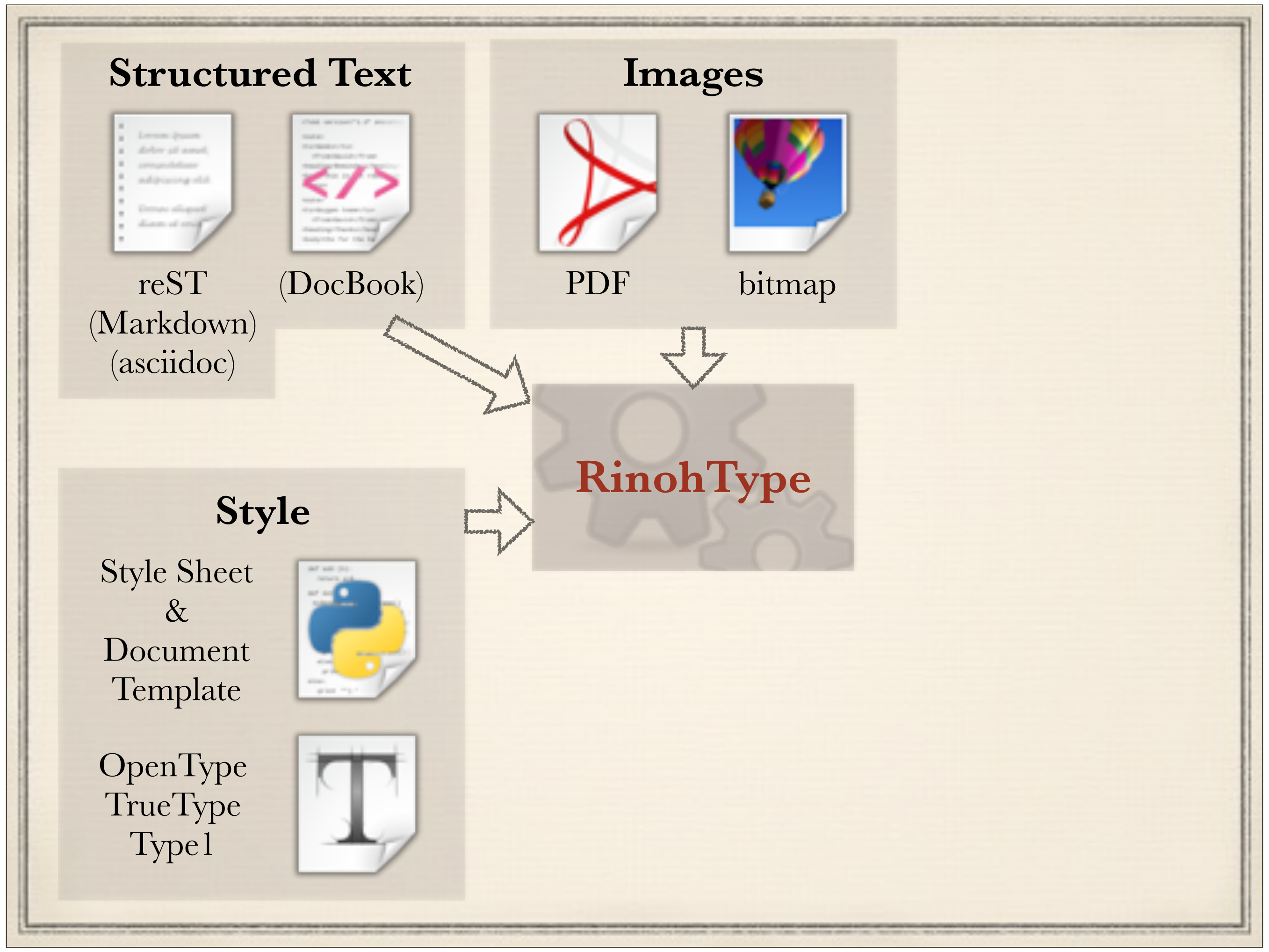
input files (similar to LaTeX)
structured text references images
RinohType comes with some style sheets and fonts
common font formats supported



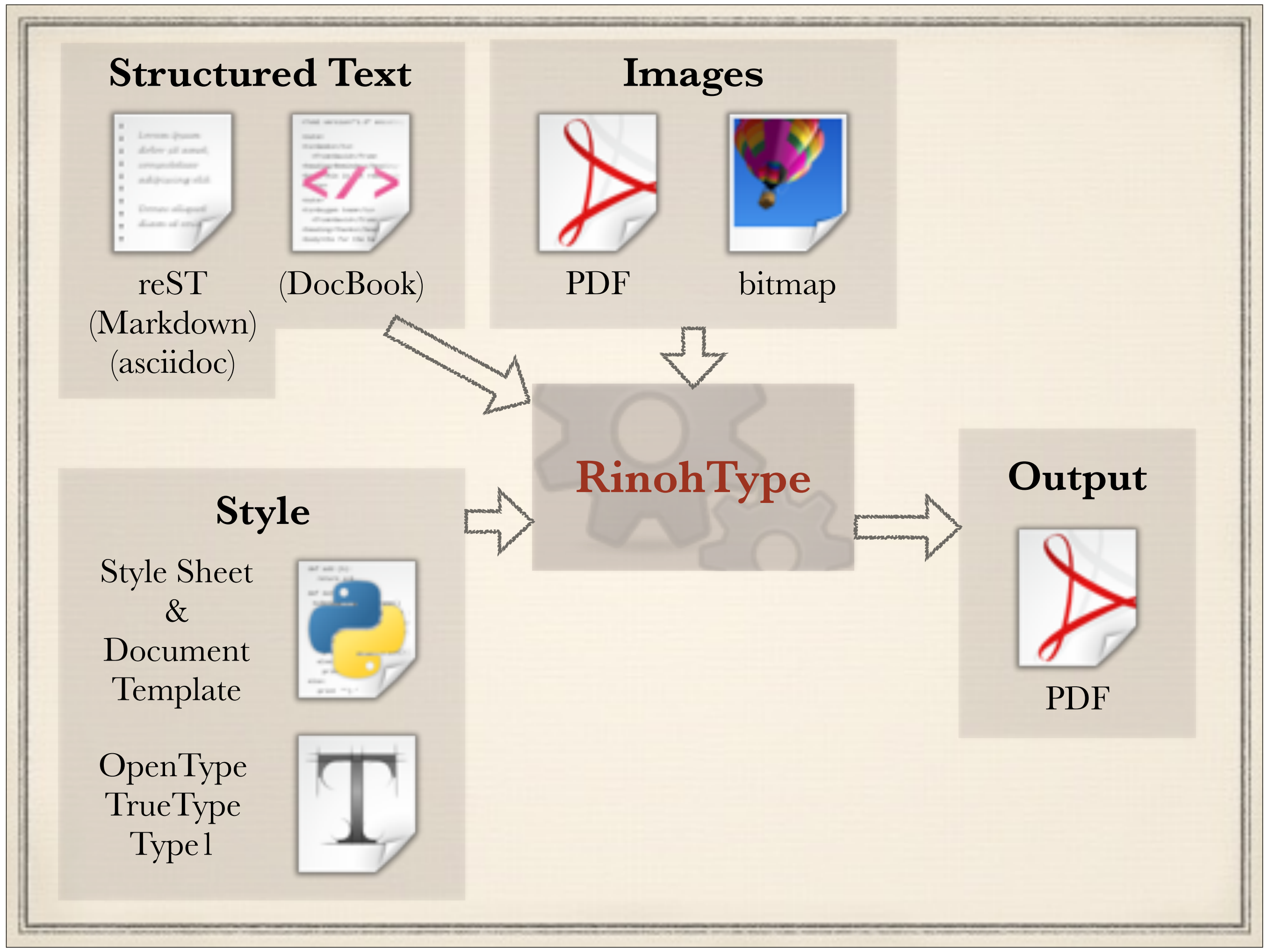
input files (similar to LaTeX)
structured text references images
RinohType comes with some style sheets and fonts
common font formats supported



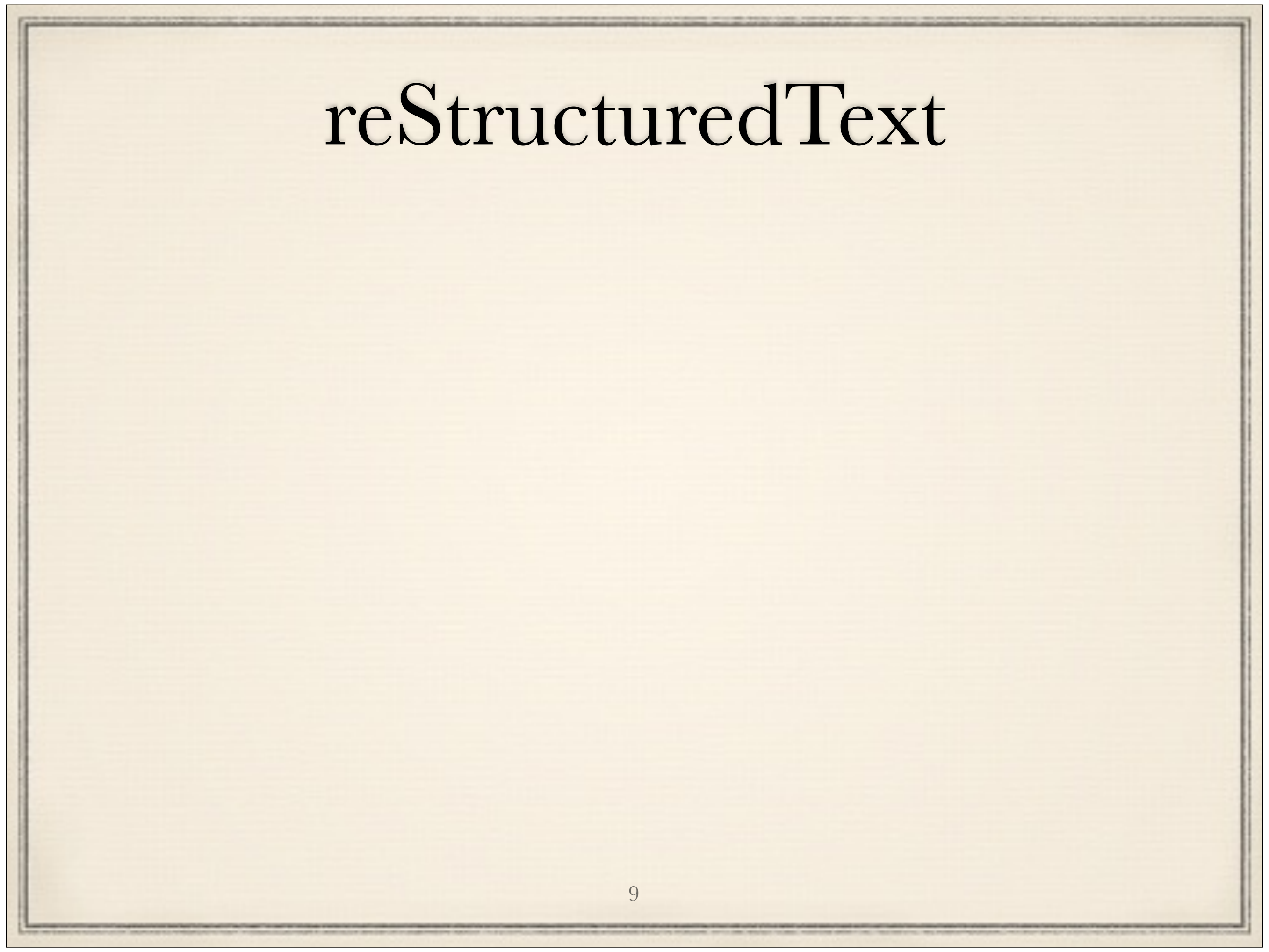
input files (similar to LaTeX)
structured text references images
RinohType comes with some style sheets and fonts
common font formats supported



input files (similar to LaTeX)
structured text references images
RinohType comes with some style sheets and fonts
common font formats supported



input files (similar to LaTeX)
structured text references images
RinohType comes with some style sheets and fonts
common font formats supported



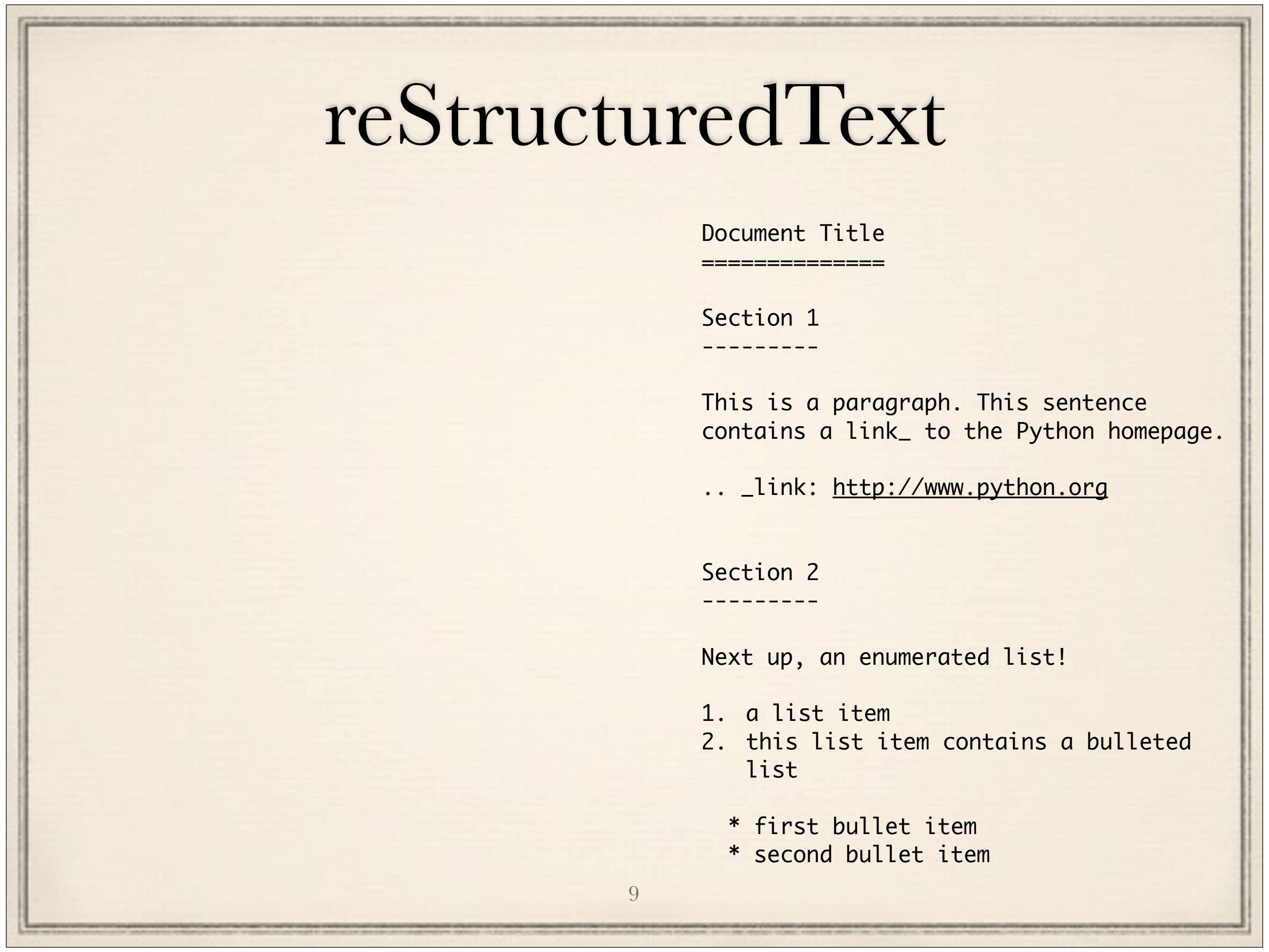
reStructuredText

Q: familiar with reStructuredText?

strict separation content/style - disadvantage?

- reST is extensible ~ TeX macros
- define new rST role/directive and corresponding RinohType Flowable/Inline
    - probably much quicker than figuring out how to do it in TeX
    - much cleaner

future: Sphinxtr for thesis?



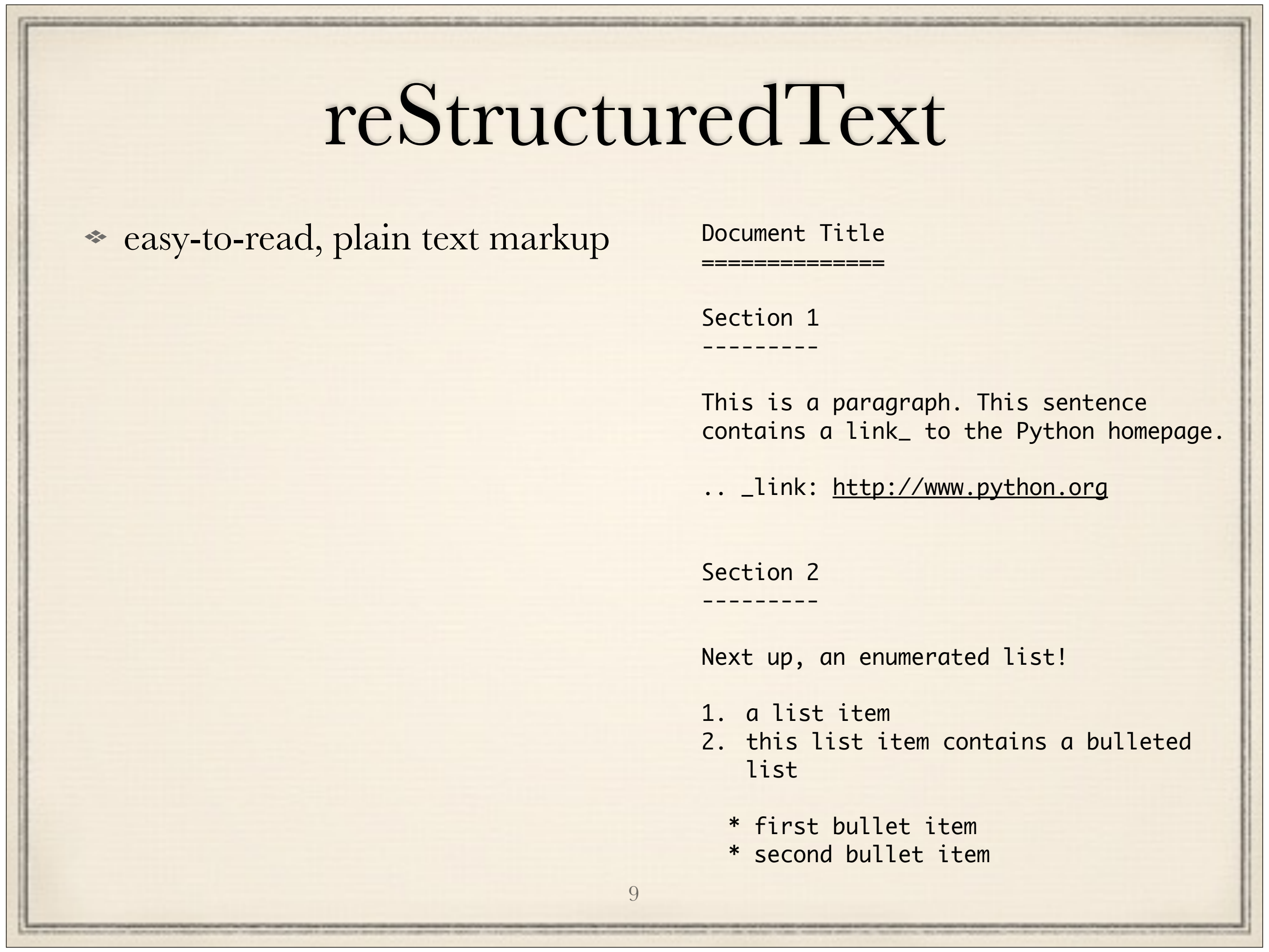
reStructuredText

```
Document Title
==============

Section 1
---------

This is a paragraph. This sentence
contains a link_ to the Python homepage.

.. _link: http://www.python.org


Section 2
---------

Next up, an enumerated list!

1. a list item
2. this list item contains a bulleted
   list

  * first bullet item
  * second bullet item
```

Q: familiar with reStructuredText?

strict separation content/style - disadvantage?
- reST is extensible ~ TeX macros
- define new rST role/directive and corresponding RinohType Flowable/Inline
    - probably much quicker than figuring out how to do it in TeX
    - much cleaner

future: Sphinxtr for thesis?



reStructuredText

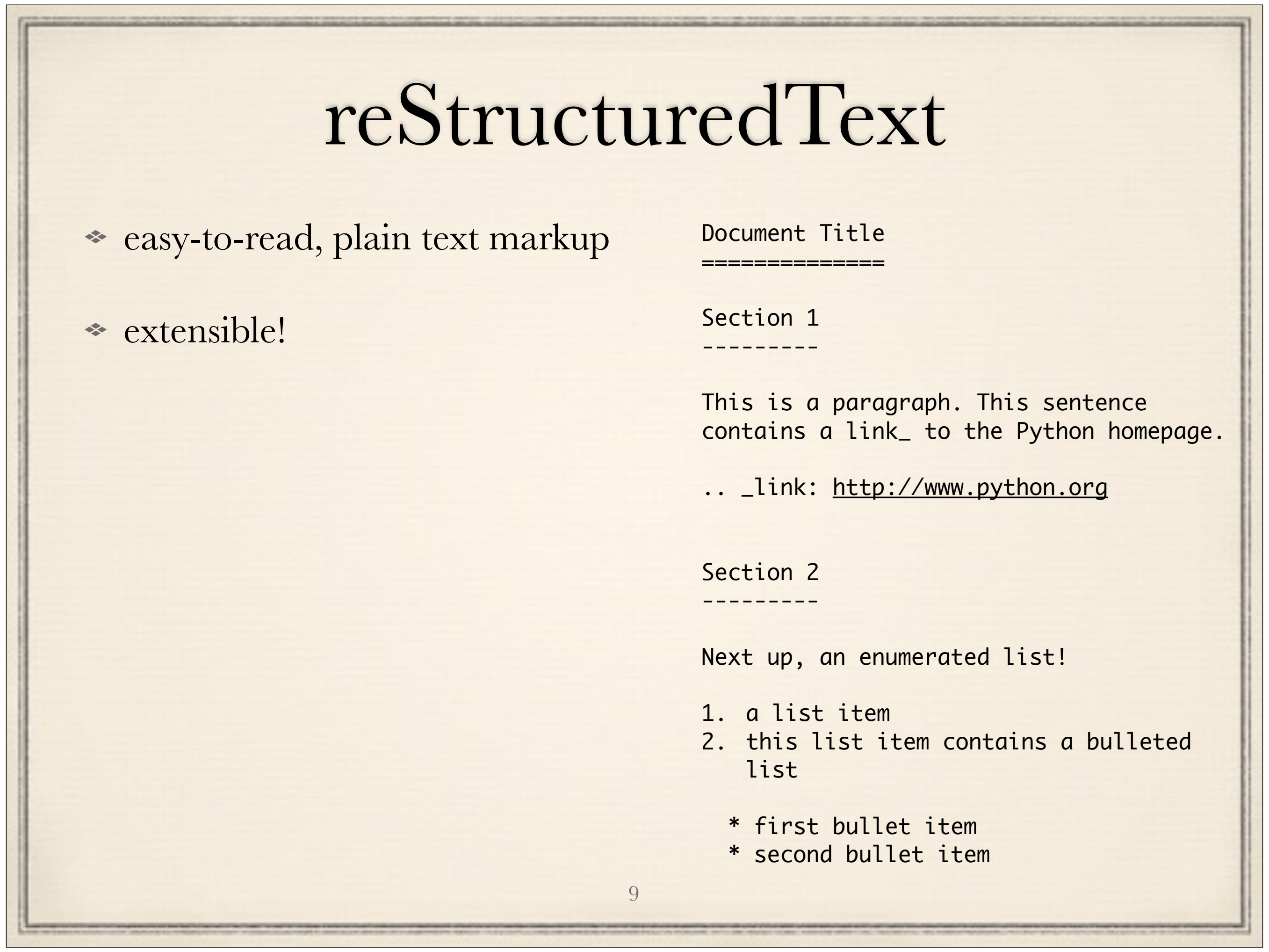
- easy-to-read, plain text markup

```
Document Title
==============

Section 1
---------

This is a paragraph. This sentence
contains a link_ to the Python homepage.

.. _link: http://www.python.org


Section 2
---------

Next up, an enumerated list!

1. a list item
2. this list item contains a bulleted
   list

  * first bullet item
  * second bullet item
```

Q: familiar with reStructuredText?

strict separation content/style - disadvantage?
- reST is extensible ~ TeX macros
- define new rST role/directive and corresponding RinohType Flowable/Inline
    - probably much quicker than figuring out how to do it in TeX
    - much cleaner

future: Sphinxtr for thesis?



reStructuredText

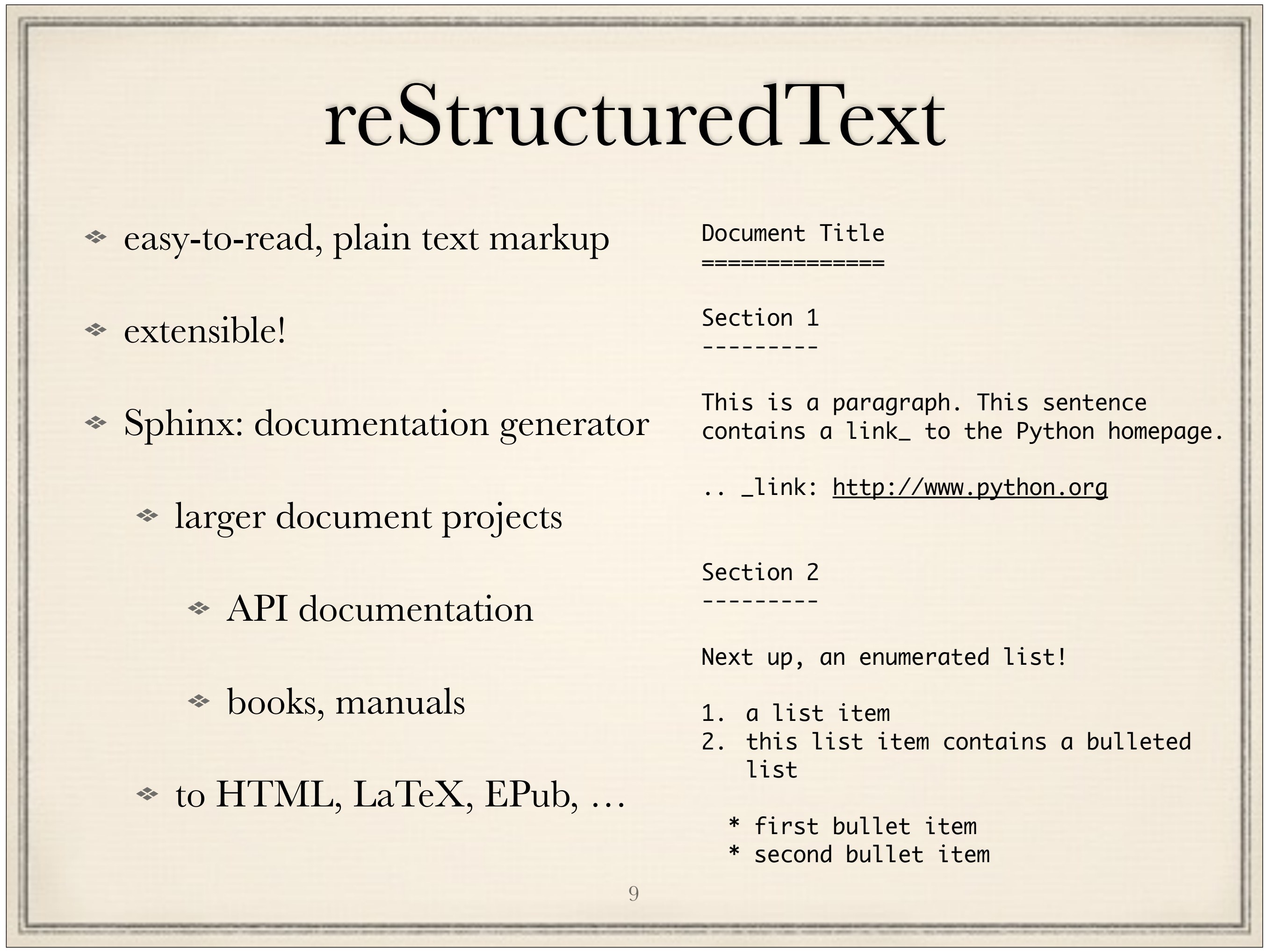
- easy-to-read, plain text markup
- extensible!

```
Document Title
==============

Section 1
---------

This is a paragraph. This sentence
contains a link_ to the Python homepage.

.. _link: http://www.python.org


Section 2
---------

Next up, an enumerated list!

1. a list item
2. this list item contains a bulleted
   list

  * first bullet item
  * second bullet item
```

Q: familiar with reStructuredText?

strict separation content/style - disadvantage?
- reST is extensible ~ TeX macros
- define new rST role/directive and corresponding RinohType Flowable/Inline
    - probably much quicker than figuring out how to do it in TeX
    - much cleaner

future: Sphinxtr for thesis?



reStructuredText

- easy-to-read, plain text markup
- extensible!
- Sphinx: documentation generator
  - larger document projects
    - API documentation
    - books, manuals
  - to HTML, LaTeX, EPub, …

```
Document Title
==============

Section 1
---------

This is a paragraph. This sentence
contains a link_ to the Python homepage.

.. _link: http://www.python.org


Section 2
---------

Next up, an enumerated list!

1. a list item
2. this list item contains a bulleted
   list

  * first bullet item
  * second bullet item
```

Q: familiar with reStructuredText?

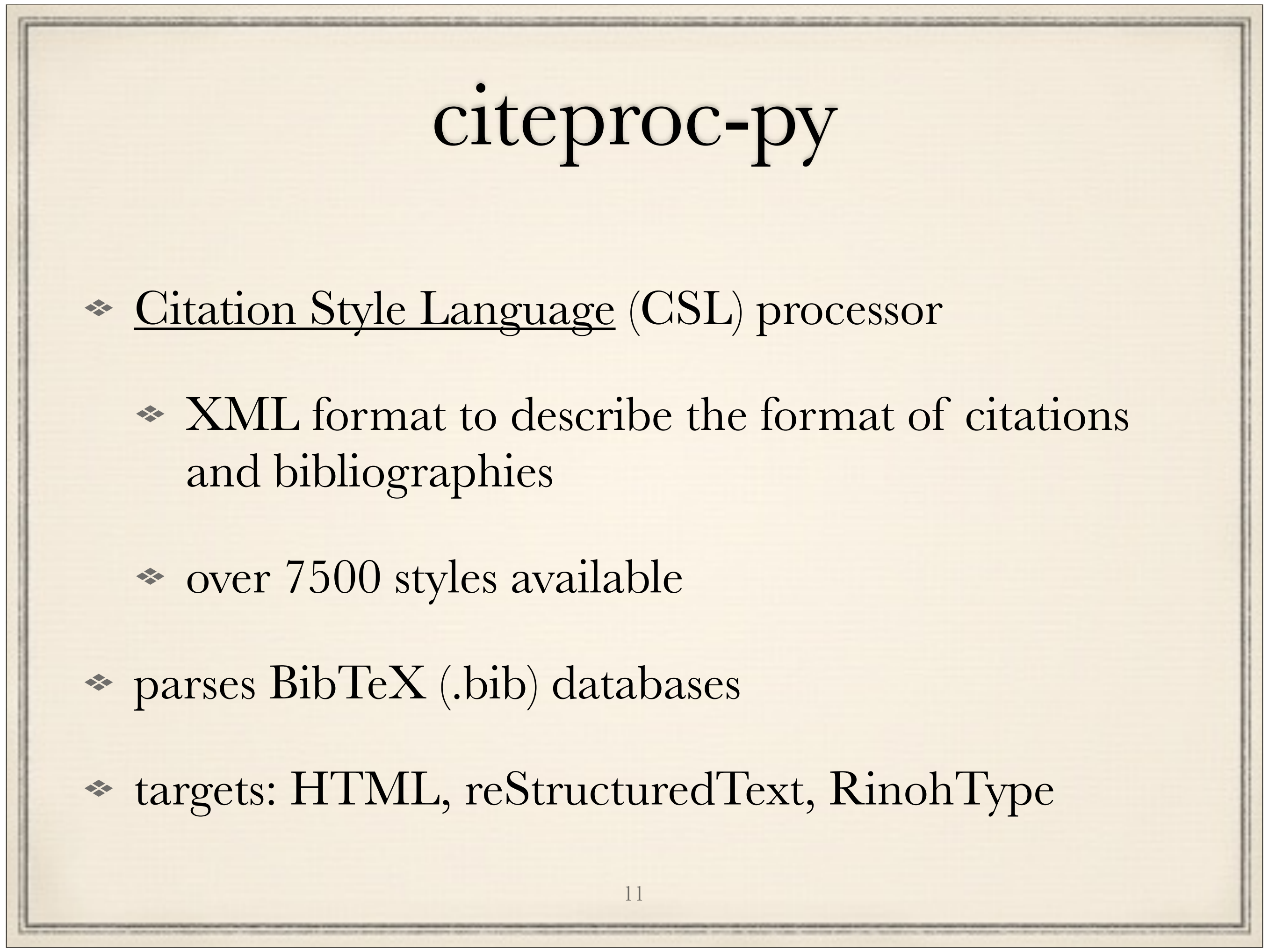
strict separation content/style - disadvantage?

- reST is extensible ~ TeX macros
- define new rST role/directive and corresponding RinohType Flowable/Inline
  - probably much quicker than figuring out how to do it in TeX
  - much cleaner

future: Sphinxtr for thesis?

Demonstration

- small reStructuredText file
    - **rinoh** cmd-line tool
- Sphinx documentation
    - conf.py changes



citeproc-py

- Citation Style Language (CSL) processor
  - XML format to describe the format of citations and bibliographies
  - over 7500 styles available
- parses BibTeX (.bib) databases
- targets: HTML, reStructuredText, RinohType

not yet useable from reST / Sphinx (requires custom role)

citeproc-py: Example

Citations

(Knuth 1968–1990; Missilany 1984)

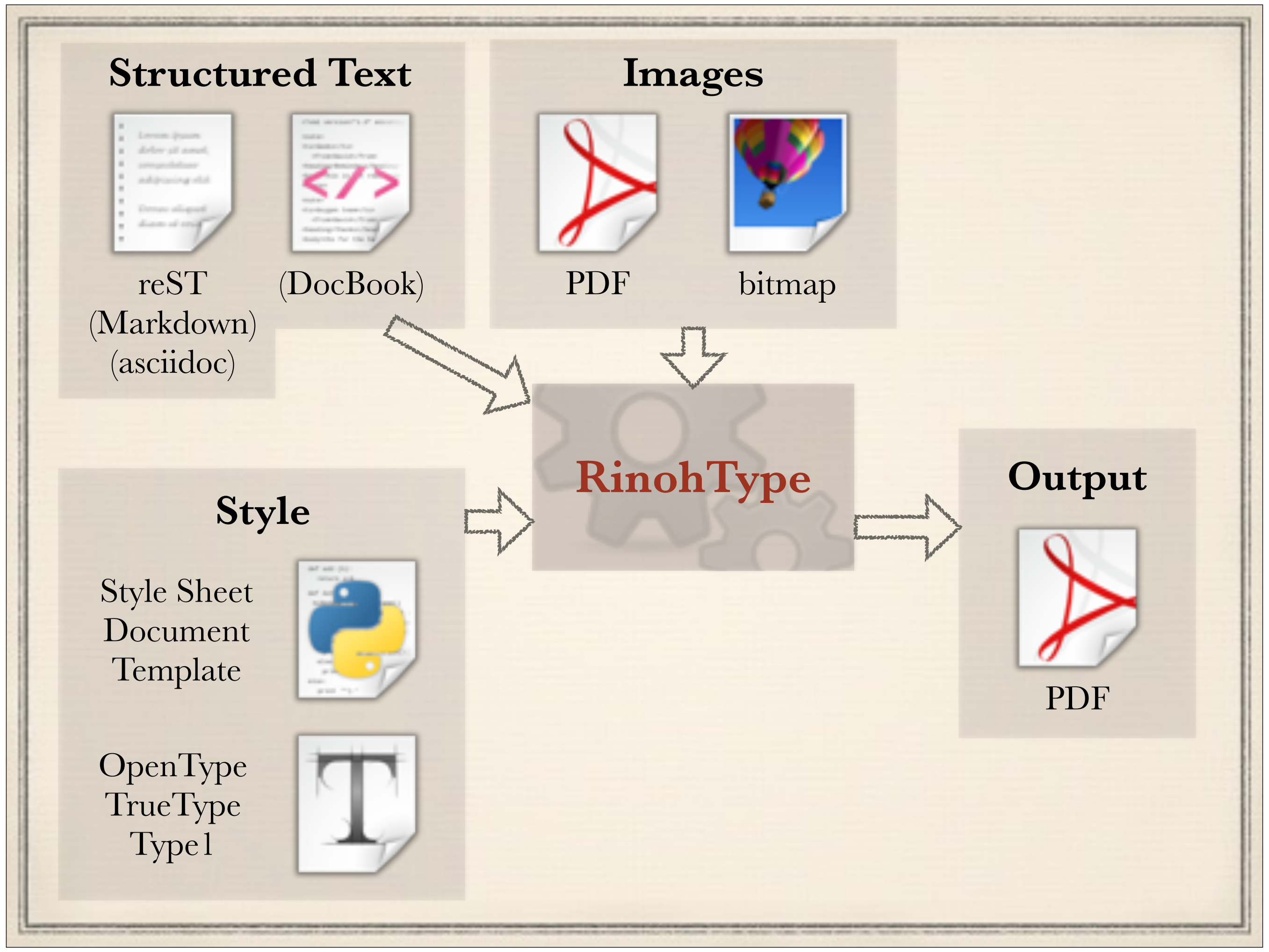
(Masterly 1988)

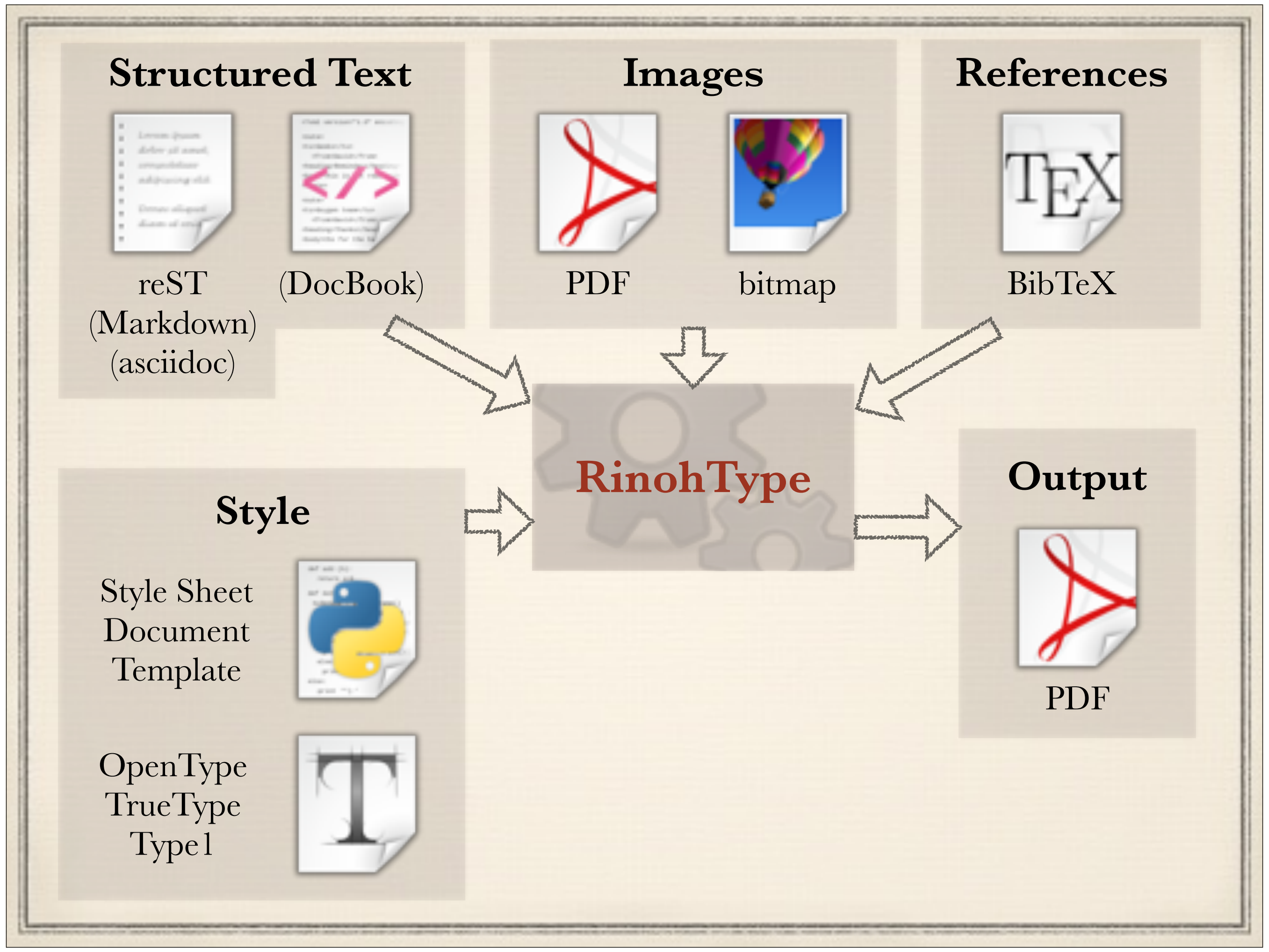
References

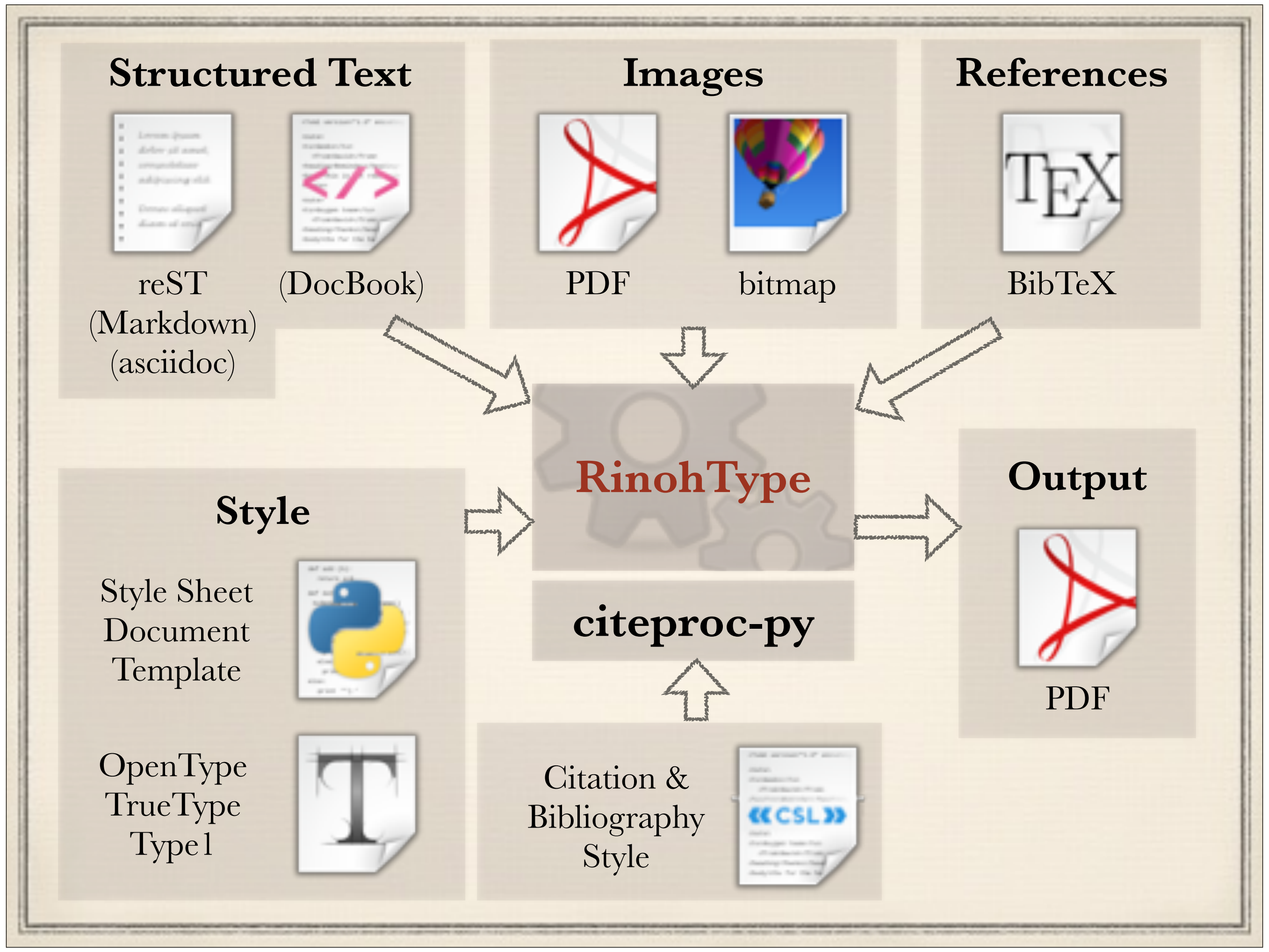
Knuth, *D.E.*, 1968–1990. *The Art of Computer Programming*, Addison-Wesley.

Missilany, *J.-B.*, 1984. *Handing out random pamphlets in airports*.

Masterly, *É.*, 1988. *Mastering Thesis Writing*.


Structured Text
reST
(Markdown)
(asciidoc)
(DocBook)
Images
PDF
bitmap
RinohType
Style
Style Sheet
Document
Template
OpenType
TrueType
Type1
Output
PDF


Structured Text
reST
(Markdown)
(asciidoc)
(DocBook)
Images
PDF
bitmap
References
BibTeX
RinohType
Style
Style Sheet
Document
Template
OpenType
TrueType
Type1
Output
PDF

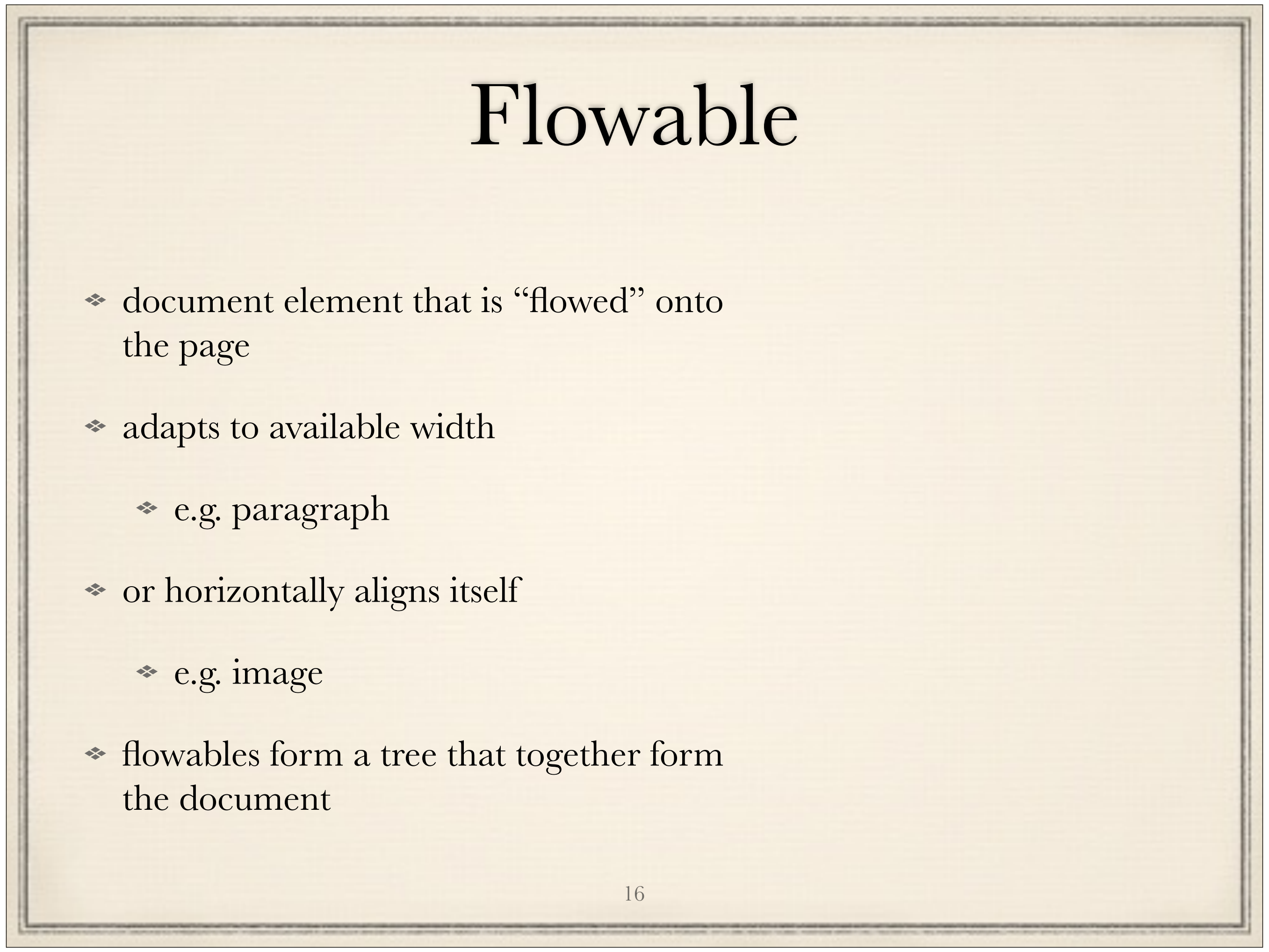

Structured Text
reST
(Markdown)
(asciidoc)
(DocBook)
Images
PDF
bitmap
References
BibTeX
RinohType
Style
Style Sheet
Document
Template
OpenType
TrueType
Type1
citeproc-py
Citation &
Bibliography
Style
Output
PDF

Internals

- Flowables & Style Sheets
- Frontend (reStructuredText)
- Page layout engine

Style Sheets

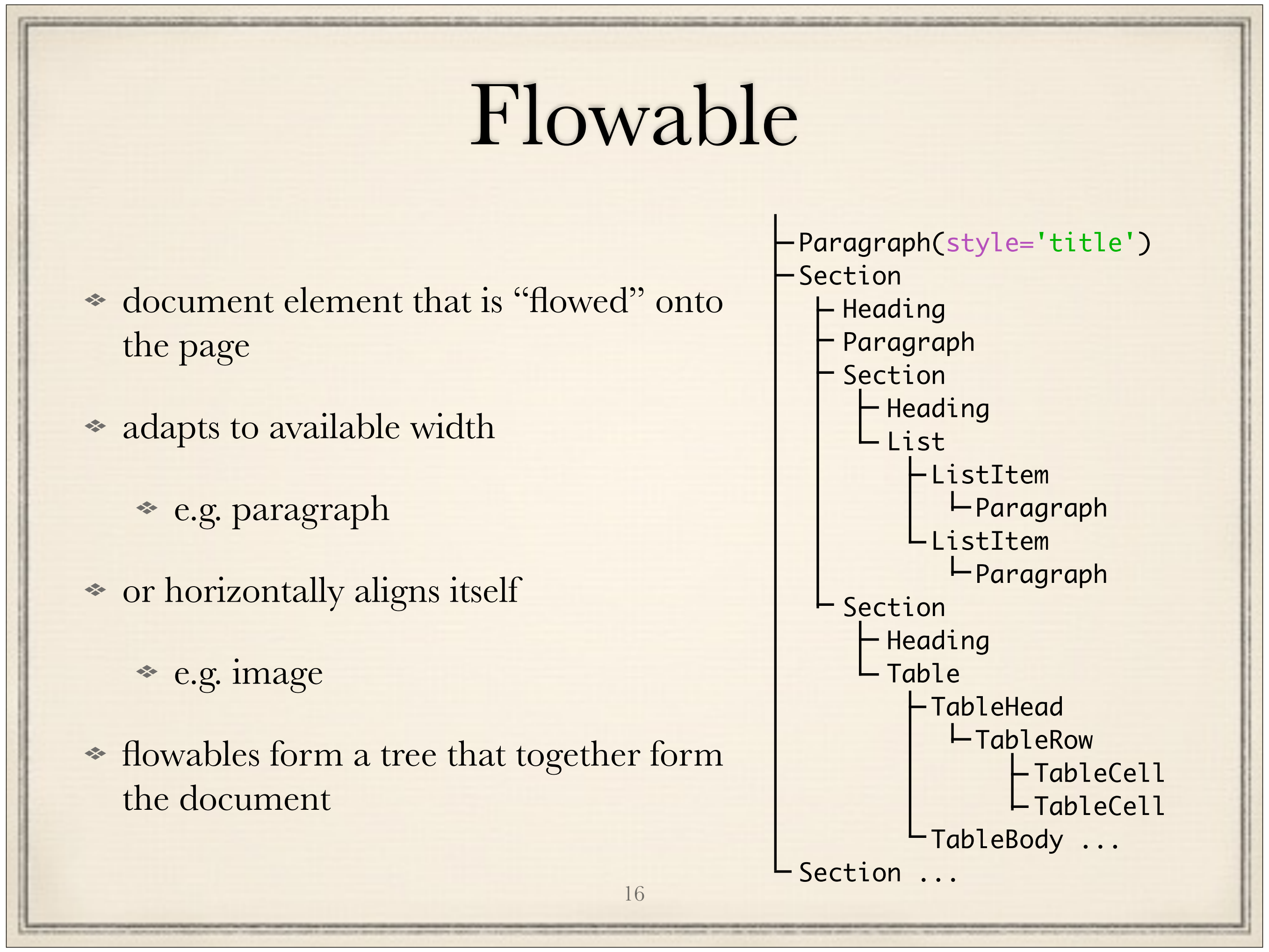
- document elements
  - Flowables (body elements)
  - Inline elements
- Style Sheets
  - link flowables to style definition



Flowable

- document element that is "flowed" onto the page
- adapts to available width
  - e.g. paragraph
- or horizontally aligns itself
  - e.g. image
- flowables form a tree that together form the document

flowables are **Python objects**



Flowable

- document element that is “flowed” onto the page
- adapts to available width
  - e.g. paragraph
- or horizontally aligns itself
  - e.g. image
- flowables form a tree that together form the document

```
┣Paragraph(style='title')
┣Section
┃ ┣Heading
┃ ┣Paragraph
┃ ┣Section
┃ ┃ ┣Heading
┃ ┃ ┗List
┃ ┃    ┣ListItem
┃ ┃    ┃  ┗Paragraph
┃ ┃    ┗ListItem
┃ ┃       ┗Paragraph
┃ ┗Section
┃    ┣Heading
┃    ┗Table
┃       ┣TableHead
┃       ┃  ┗TableRow
┃       ┃     ┣TableCell
┃       ┃     ┗TableCell
┃       ┗TableBody ...
┗Section ...
```

flowables are **Python objects**

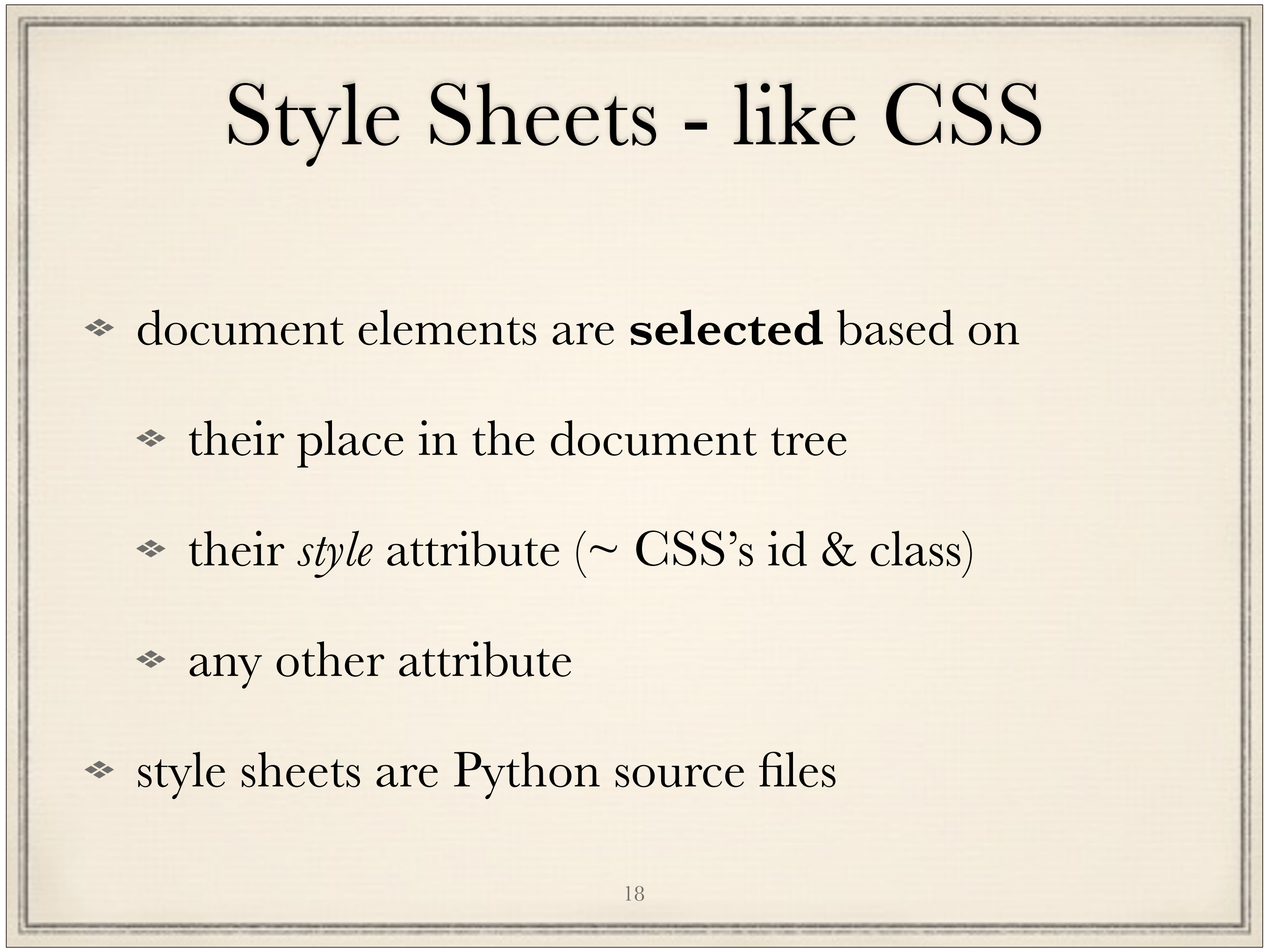
Inline Elements

Text with *multiple* ***nested STYLES***.

Inline Elements

Text with *multiple **nested STYLES***.

```
Paragraph
 ├─SingleStyledText("Text with ")
 ├─MixedStyledText(style='emphasis')
 │  ├─ SingleStyledText("multiple ")
 │  └─ MixedStyledText(style='strong')
 │     ├─SingleStyledText("nested ")
 │     └─SingleStyledText("styles", style='small caps')
 └─SingleStyledText(".")
```



Style Sheets - like CSS

- document elements are **selected** based on
  - their place in the document tree
  - their *style* attribute (~ CSS's id & class)
  - any other attribute
- style sheets are Python source files

Q: who is familiar with CSS?

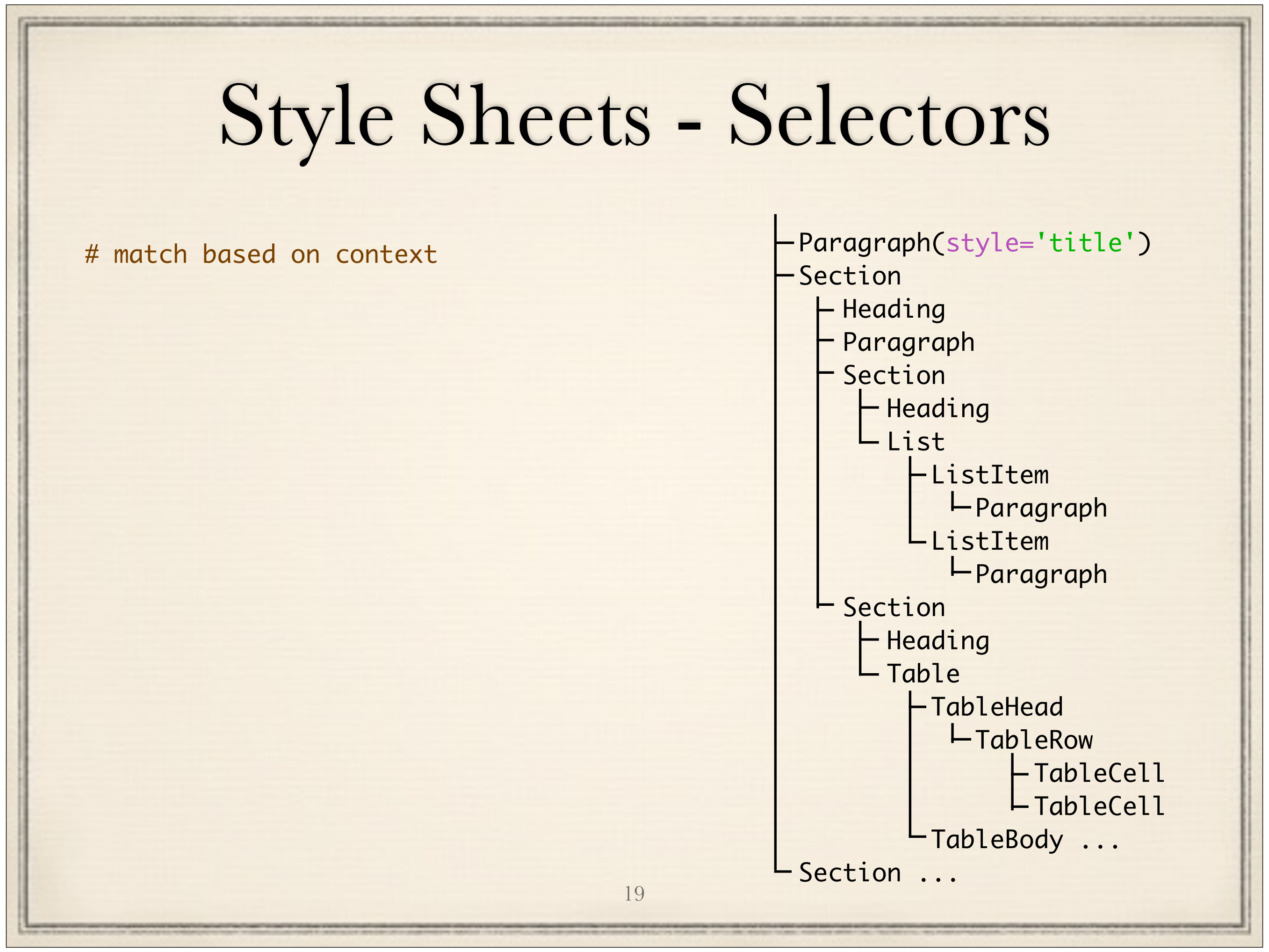
Python style sheets: later support text version

Style Sheets - Selectors

```
├─Paragraph(style='title')
├─Section
│  ├ Heading
│  ├ Paragraph
│  ├ Section
│  │  ├ Heading
│  │  └ List
│  │     ├─ListItem
│  │     │  └─Paragraph
│  │     └─ListItem
│  │        └─Paragraph
│  └ Section
│     ├ Heading
│     └ Table
│        ├─TableHead
│        │  └─TableRow
│        │       ├─TableCell
│        │       └─TableCell
│        └─TableBody ...
└─Section ...
```

flowables are **Python objects**, also used in selectors

row_index:
- not a simple integer, but
- a special object that has a custom __eq__ operator



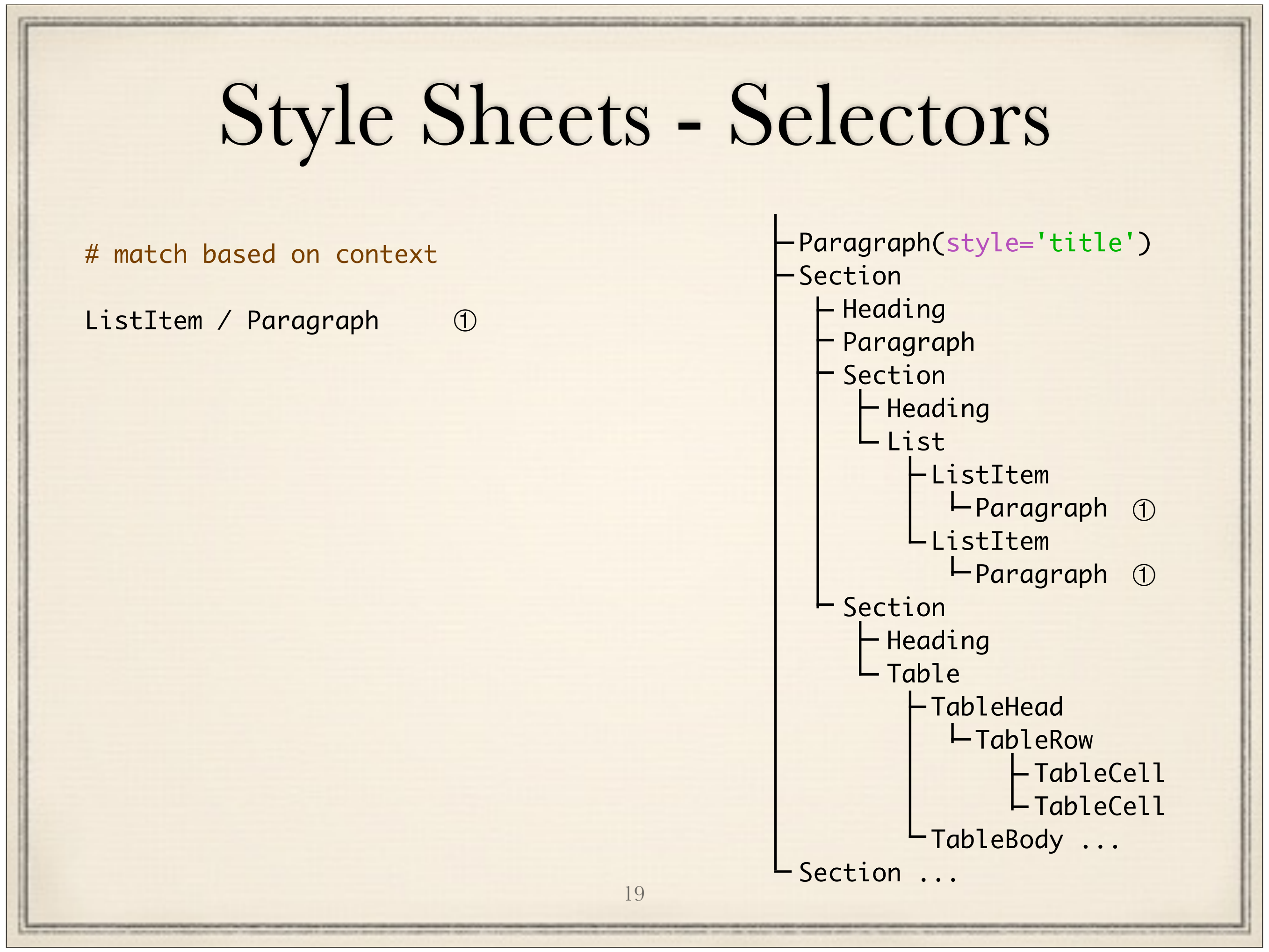
Style Sheets - Selectors

```
# match based on context
```

```
┣Paragraph(style='title')
┣Section
┃ ┣ Heading
┃ ┣ Paragraph
┃ ┣ Section
┃ ┃ ┣Heading
┃ ┃ ┗List
┃ ┃    ┣ListItem
┃ ┃    ┃  ┗Paragraph
┃ ┃    ┗ListItem
┃ ┃       ┗Paragraph
┃ ┗ Section
┃    ┣Heading
┃    ┗Table
┃       ┣TableHead
┃       ┃  ┗TableRow
┃       ┃      ┣TableCell
┃       ┃      ┗TableCell
┃       ┗TableBody ...
┗Section ...
```

flowables are **Python objects**, also used in selectors

row_index:
- not a simple integer, but
- a special object that has a custom __eq__ operator



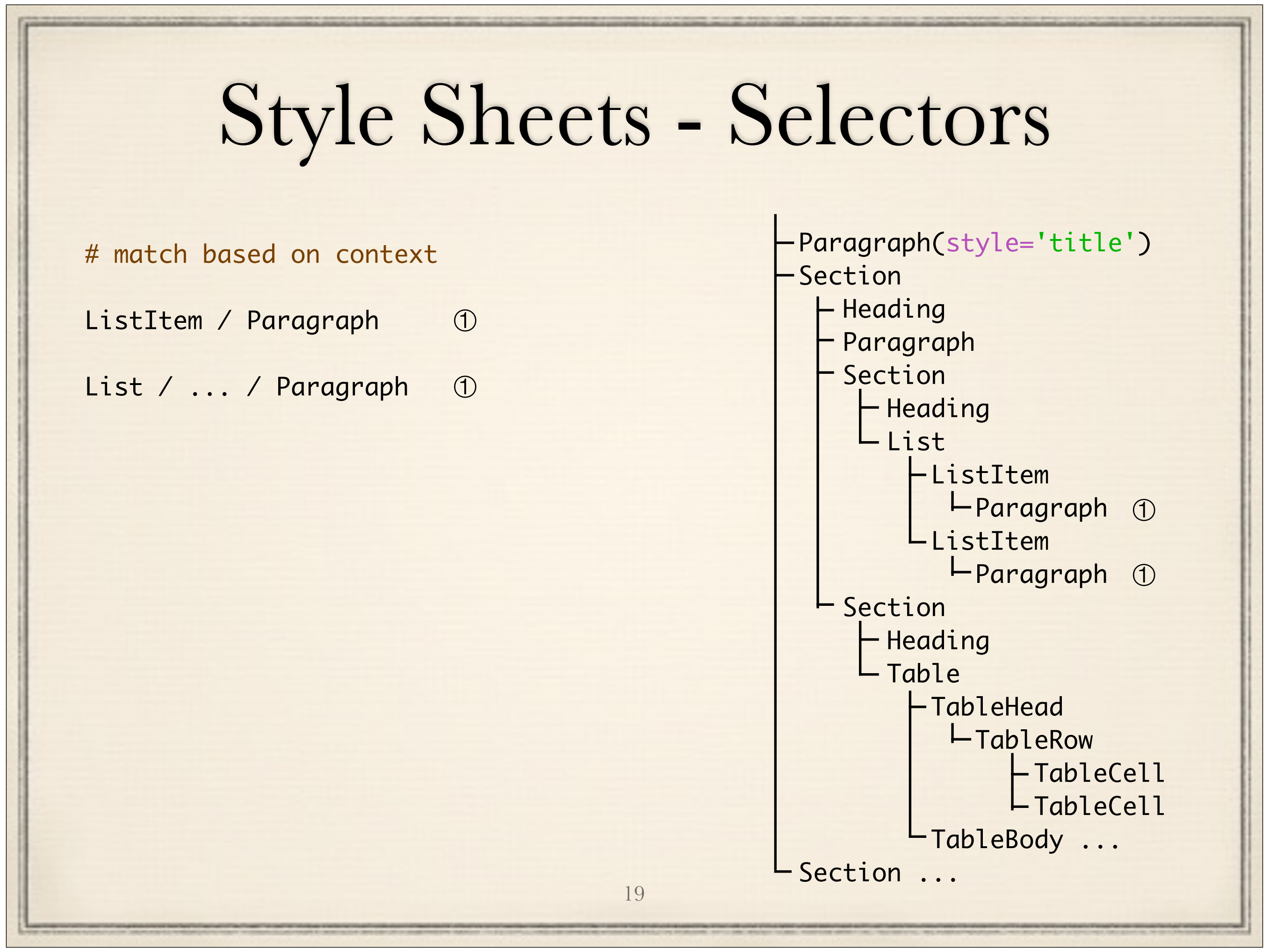
Style Sheets - Selectors

```
# match based on context

ListItem / Paragraph      ①
```

```
┣Paragraph(style='title')
┣Section
┃  ┣Heading
┃  ┣Paragraph
┃  ┣Section
┃  ┃  ┣Heading
┃  ┃  ┗List
┃  ┃     ┣ListItem
┃  ┃     ┃  ┗Paragraph  ①
┃  ┃     ┗ListItem
┃  ┃        ┗Paragraph  ①
┃  ┗Section
┃     ┣Heading
┃     ┗Table
┃        ┣TableHead
┃        ┃  ┗TableRow
┃        ┃     ┣TableCell
┃        ┃     ┗TableCell
┃        ┗TableBody ...
┗Section ...
```

flowables are **Python objects**, also used in selectors

row_index:
- not a simple integer, but
- a special object that has a custom __eq__ operator



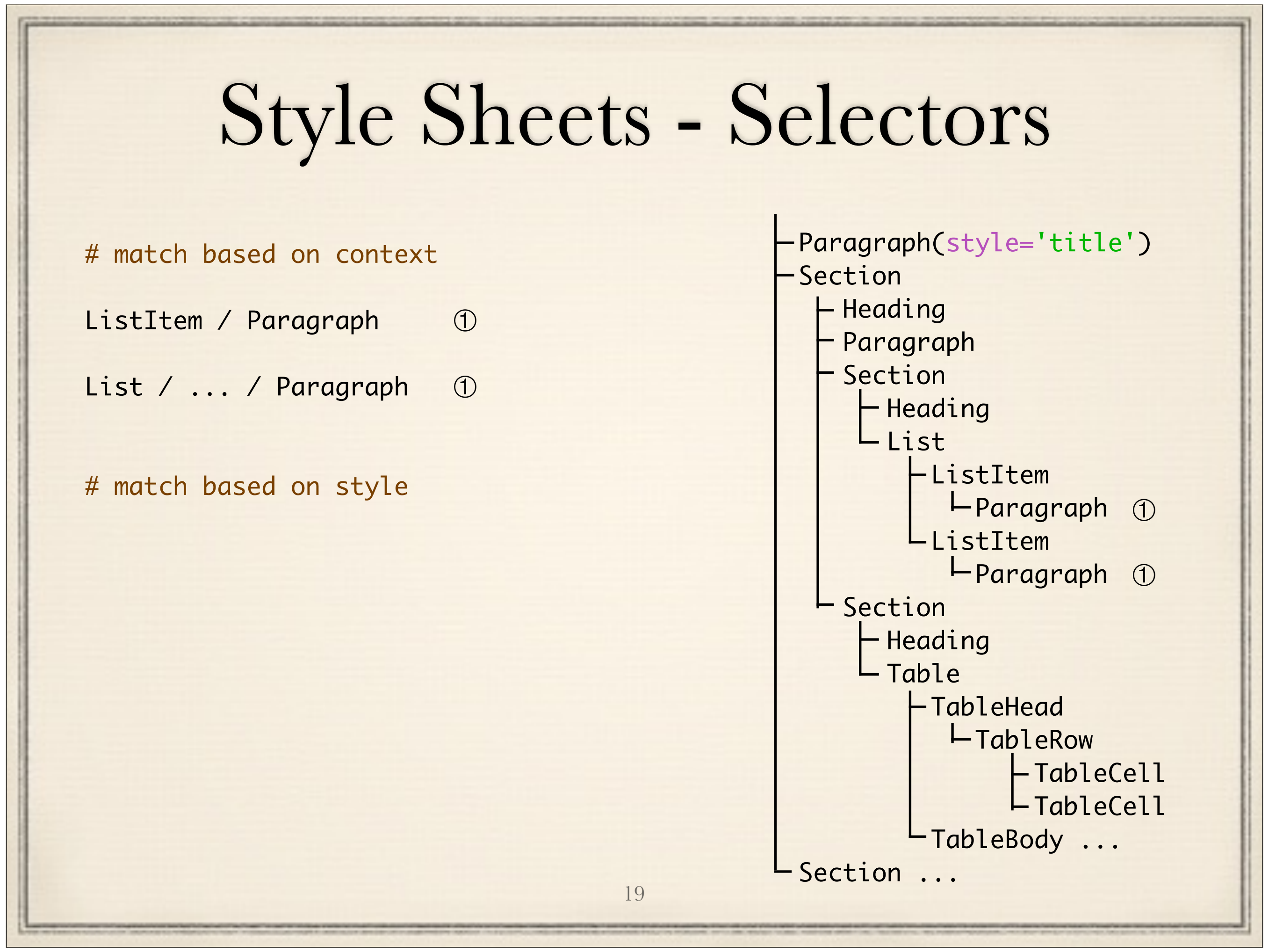
Style Sheets - Selectors

```
# match based on context

ListItem / Paragraph       ①

List / ... / Paragraph     ①
```

```
┣Paragraph(style='title')
┣Section
┃  ┣Heading
┃  ┣Paragraph
┃  ┣Section
┃  ┃  ┣Heading
┃  ┃  ┗List
┃  ┃     ┣ListItem
┃  ┃     ┃  ┗Paragraph  ①
┃  ┃     ┗ListItem
┃  ┃        ┗Paragraph  ①
┃  ┗Section
┃     ┣Heading
┃     ┗Table
┃        ┣TableHead
┃        ┃  ┗TableRow
┃        ┃      ┣TableCell
┃        ┃      ┗TableCell
┃        ┗TableBody ...
┗Section ...
```

flowables are **Python objects**, also used in selectors

row_index:
- not a simple integer, but
- a special object that has a custom __eq__ operator



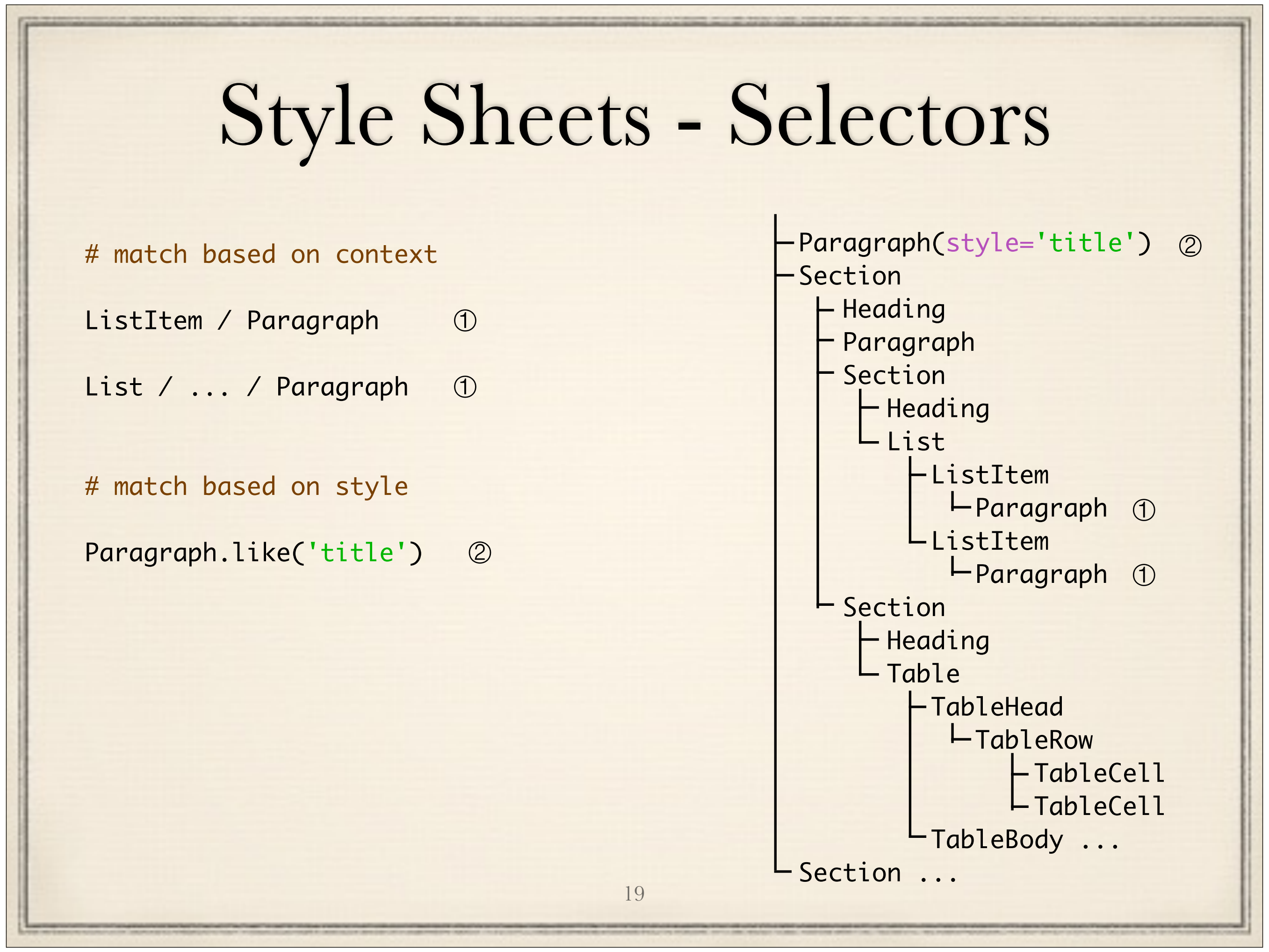
Style Sheets - Selectors

```
# match based on context

ListItem / Paragraph       ①

List / ... / Paragraph     ①


# match based on style
```

```
├Paragraph(style='title')
├Section
│ ├Heading
│ ├Paragraph
│ ├Section
│ │ ├Heading
│ │ └List
│ │   ├ListItem
│ │   │ └Paragraph  ①
│ │   └ListItem
│ │     └Paragraph  ①
│ └Section
│   ├Heading
│   └Table
│     ├TableHead
│     │ └TableRow
│     │     ├TableCell
│     │     └TableCell
│     └TableBody ...
└Section ...
```

flowables are **Python objects**, also used in selectors

row_index:
- not a simple integer, but
- a special object that has a custom __eq__ operator



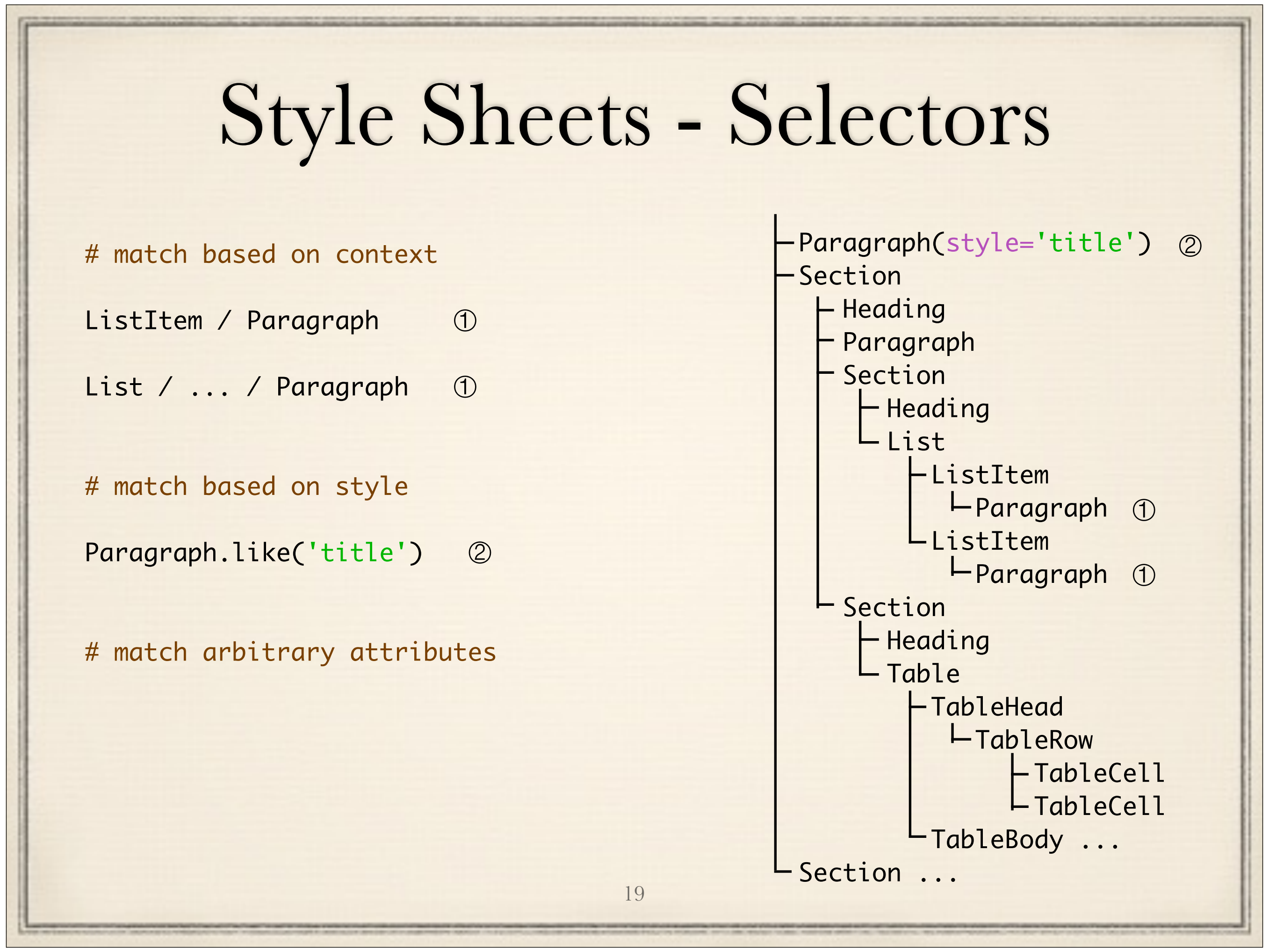
Style Sheets - Selectors

```
# match based on context

ListItem / Paragraph        ①

List / ... / Paragraph      ①


# match based on style

Paragraph.like('title')    ②
```

```
┣Paragraph(style='title')  ②
┣Section
┃  ┣ Heading
┃  ┣ Paragraph
┃  ┣ Section
┃  ┃  ┣Heading
┃  ┃  ┗List
┃  ┃     ┣ListItem
┃  ┃     ┃  ┗Paragraph  ①
┃  ┃     ┗ListItem
┃  ┃        ┗Paragraph  ①
┃  ┗ Section
┃     ┣Heading
┃     ┗Table
┃        ┣TableHead
┃        ┃  ┗TableRow
┃        ┃      ┣TableCell
┃        ┃      ┗TableCell
┃        ┗TableBody ...
┗Section ...
```

flowables are **Python objects**, also used in selectors

row_index:
- not a simple integer, but
- a special object that has a custom __eq__ operator



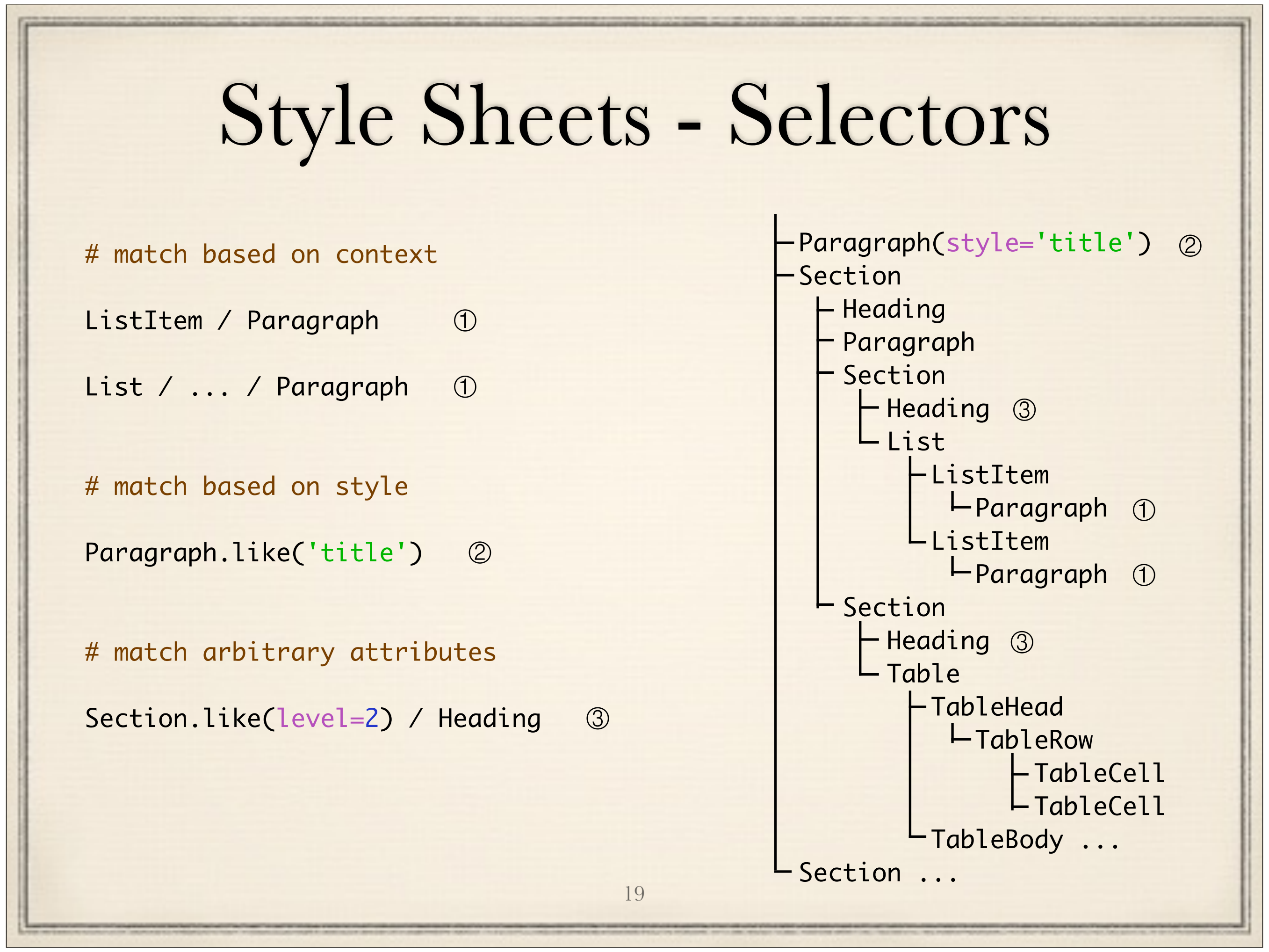
Style Sheets - Selectors

```
# match based on context

ListItem / Paragraph      ①

List / ... / Paragraph    ①


# match based on style

Paragraph.like('title')    ②


# match arbitrary attributes
```

```
┣Paragraph(style='title')  ②
┣Section
┃  ┣Heading
┃  ┣Paragraph
┃  ┣Section
┃  ┃  ┣Heading
┃  ┃  ┗List
┃  ┃     ┣ListItem
┃  ┃     ┃  ┗Paragraph  ①
┃  ┃     ┗ListItem
┃  ┃        ┗Paragraph  ①
┃  ┗Section
┃     ┣Heading
┃     ┗Table
┃        ┣TableHead
┃        ┃  ┗TableRow
┃        ┃      ┣TableCell
┃        ┃      ┗TableCell
┃        ┗TableBody ...
┗Section ...
```

flowables are **Python objects**, also used in selectors

row_index:
- not a simple integer, but
- a special object that has a custom __eq__ operator



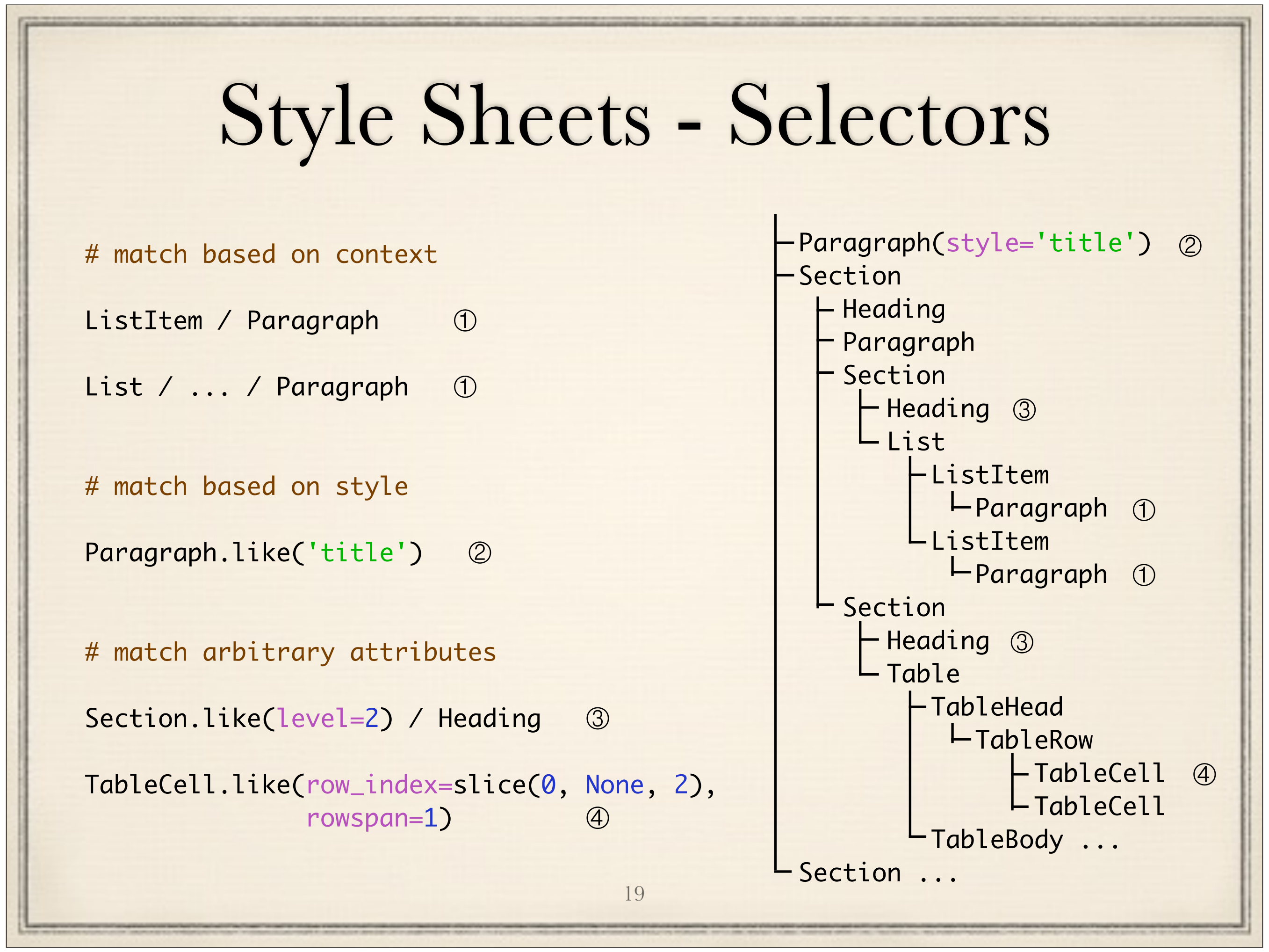
Style Sheets - Selectors

```
# match based on context

ListItem / Paragraph        ①

List / ... / Paragraph      ①


# match based on style

Paragraph.like('title')    ②


# match arbitrary attributes

Section.like(level=2) / Heading    ③
```

```
┣Paragraph(style='title')  ②
┣Section
┃  ┣Heading
┃  ┣Paragraph
┃  ┣Section
┃  ┃  ┣Heading  ③
┃  ┃  ┗List
┃  ┃     ┣ListItem
┃  ┃     ┃  ┗Paragraph  ①
┃  ┃     ┗ListItem
┃  ┃        ┗Paragraph  ①
┃  ┗Section
┃     ┣Heading  ③
┃     ┗Table
┃        ┣TableHead
┃        ┃  ┗TableRow
┃        ┃      ┣TableCell
┃        ┃      ┗TableCell
┃        ┗TableBody ...
┗Section ...
```

flowables are **Python objects**, also used in selectors

row_index:
- not a simple integer, but
- a special object that has a custom __eq__ operator



Style Sheets - Selectors

```
# match based on context

ListItem / Paragraph        ①

List / ... / Paragraph      ①


# match based on style

Paragraph.like('title')    ②


# match arbitrary attributes

Section.like(level=2) / Heading    ③

TableCell.like(row_index=slice(0, None, 2),
               rowspan=1)          ④
```

```
├Paragraph(style='title')  ②
├Section
│ ├Heading
│ ├Paragraph
│ ├Section
│ │ ├Heading  ③
│ │ └List
│ │   ├ListItem
│ │   │ └Paragraph  ①
│ │   └ListItem
│ │     └Paragraph  ①
│ └Section
│   ├Heading  ③
│   └Table
│     ├TableHead
│     │ └TableRow
│     │    ├TableCell  ④
│     │    └TableCell
│     └TableBody ...
└Section ...
```

flowables are **Python objects**, also used in selectors

row_index:
- not a simple integer, but
- a special object that has a custom __eq__ operator

Style Sheets - Beyond CSS

Style Sheets - Beyond CSS

- extra level of indirection
    - **style matcher**: map selectors to style names
    - **style sheet**: map style name to style definition

Style Sheets - Beyond CSS

- extra level of indirection
  - **style matcher**: map selectors to style names
  - **style sheet**: map style name to style definition
- **variables** avoid duplication

Style Sheets - Beyond CSS

- extra level of indirection
  - **style matcher**: map selectors to style names
  - **style sheet**: map style name to style definition
- **variables** avoid duplication
- a style can **inherit** from another

Style Matcher and Sheet

matcher can be used by multiple style sheets

Style Matcher and Sheet

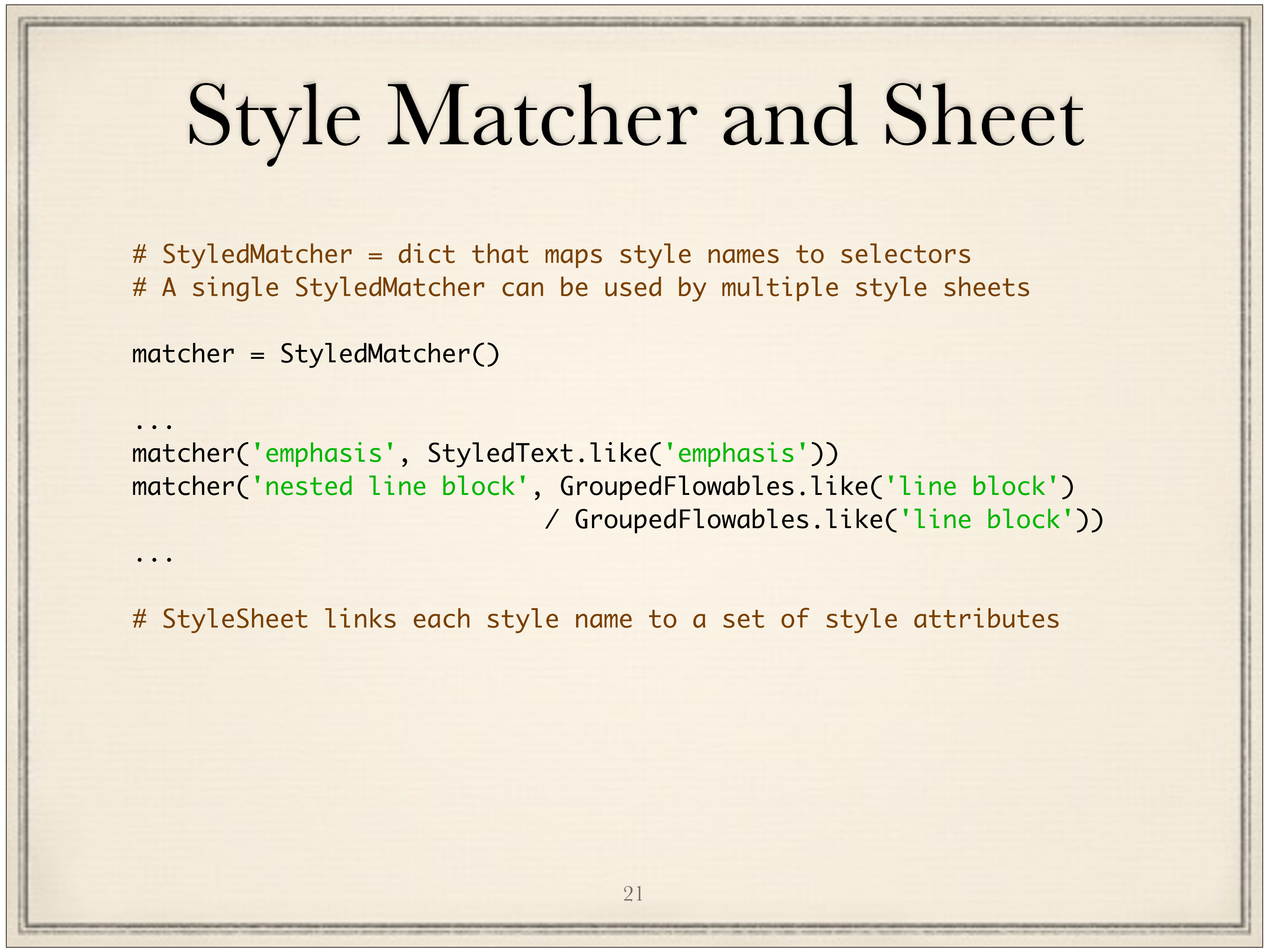
```
# StyledMatcher = dict that maps style names to selectors
# A single StyledMatcher can be used by multiple style sheets
```

matcher can be used by multiple style sheets

Style Matcher and Sheet

```
# StyledMatcher = dict that maps style names to selectors
# A single StyledMatcher can be used by multiple style sheets

matcher = StyledMatcher()
```

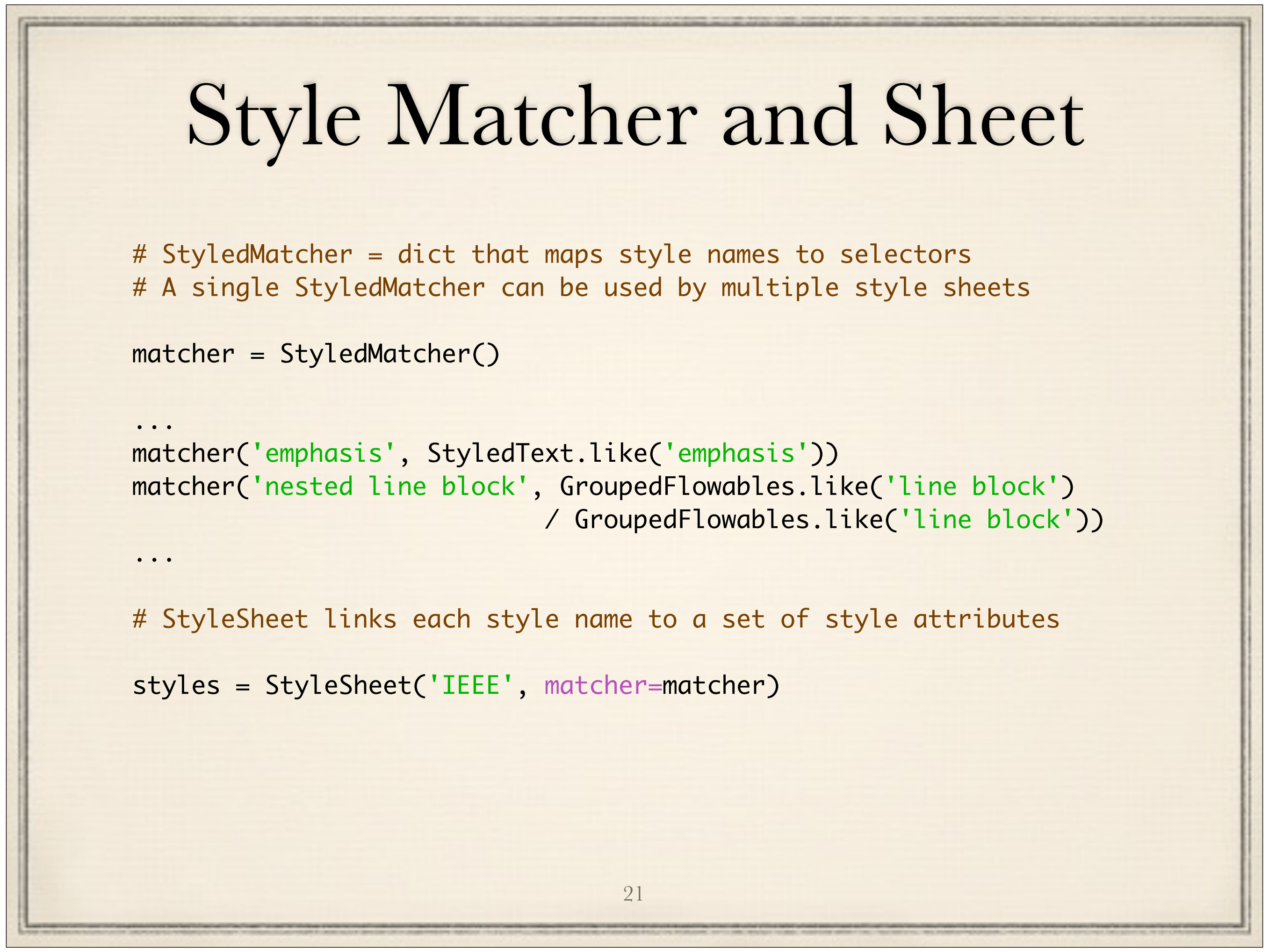
matcher can be used by multiple style sheets



Style Matcher and Sheet

```
# StyledMatcher = dict that maps style names to selectors
# A single StyledMatcher can be used by multiple style sheets

matcher = StyledMatcher()

...
matcher('emphasis', StyledText.like('emphasis'))
matcher('nested line block', GroupedFlowables.like('line block')
                             / GroupedFlowables.like('line block'))
...

# StyleSheet links each style name to a set of style attributes
```

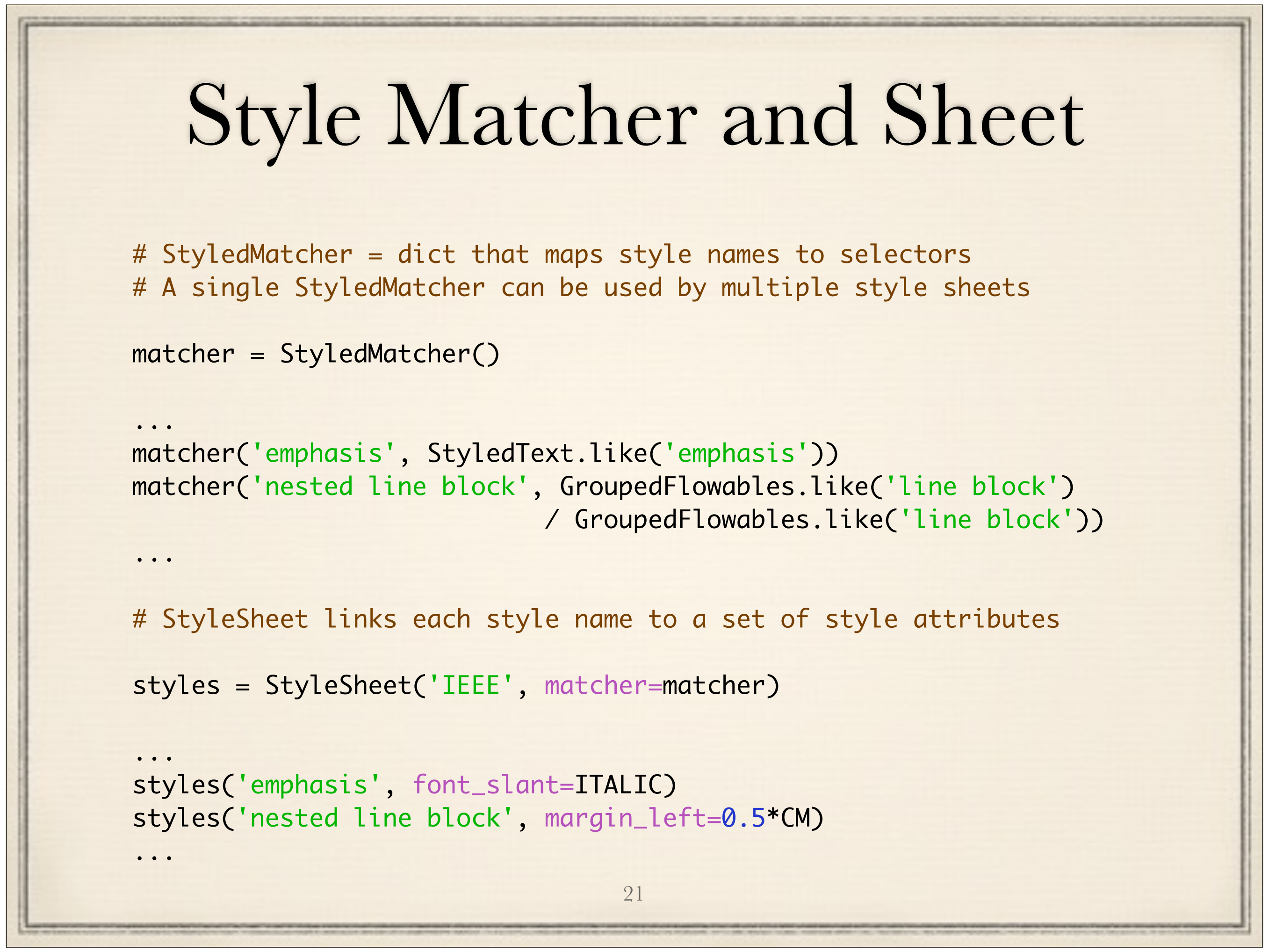
matcher can be used by multiple style sheets



Style Matcher and Sheet

```
# StyledMatcher = dict that maps style names to selectors
# A single StyledMatcher can be used by multiple style sheets

matcher = StyledMatcher()

...
matcher('emphasis', StyledText.like('emphasis'))
matcher('nested line block', GroupedFlowables.like('line block')
                             / GroupedFlowables.like('line block'))
...

# StyleSheet links each style name to a set of style attributes

styles = StyleSheet('IEEE', matcher=matcher)
```

matcher can be used by multiple style sheets



Style Matcher and Sheet

```
# StyledMatcher = dict that maps style names to selectors
# A single StyledMatcher can be used by multiple style sheets

matcher = StyledMatcher()

...
matcher('emphasis', StyledText.like('emphasis'))
matcher('nested line block', GroupedFlowables.like('line block')
                            / GroupedFlowables.like('line block'))
...

# StyleSheet links each style name to a set of style attributes

styles = StyleSheet('IEEE', matcher=matcher)

...
styles('emphasis', font_slant=ITALIC)
styles('nested line block', margin_left=0.5*CM)
...
```

matcher can be used by multiple style sheets

Style Sheets - Variables

Style Sheets - Variables

```
# Let’s make all fonts easily replaceable
```

Style Sheets - Variables

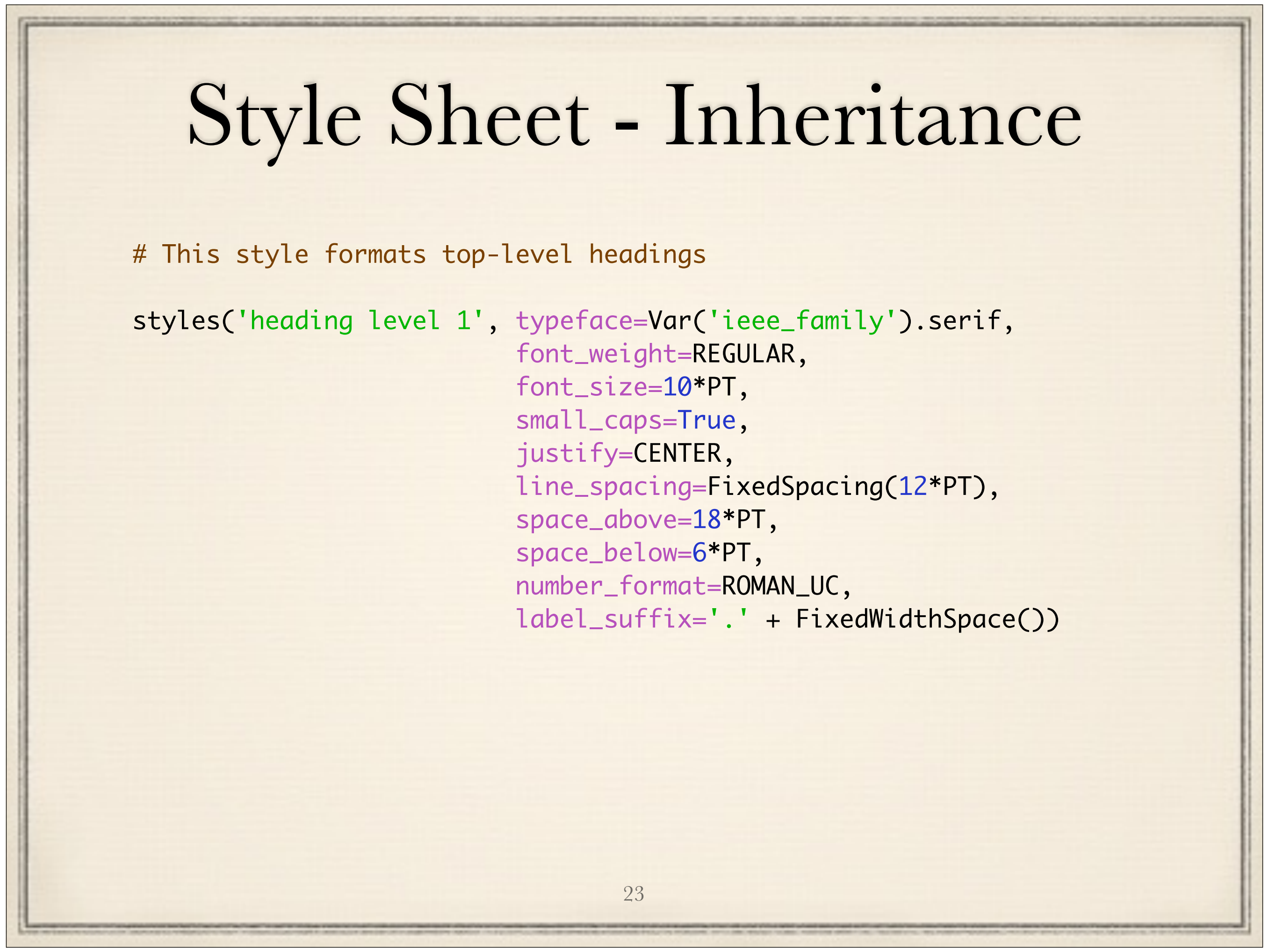
```
# Let’s make all fonts easily replaceable

styles.variables['ieee_family'] = TypeFamily(serif=times,
                                             sans=helvetica,
                                             mono=courier)
```

Style Sheets - Variables

```
# Let’s make all fonts easily replaceable

styles.variables['ieee_family'] = TypeFamily(serif=times,
                                             sans=helvetica,
                                             mono=courier)
...
styles('monospaced', typeface=Var('ieee_family').mono,
                     font_size=9*PT,
                     hyphenate=False,
                     ligatures=False)
...
```

Style Sheet - Inheritance

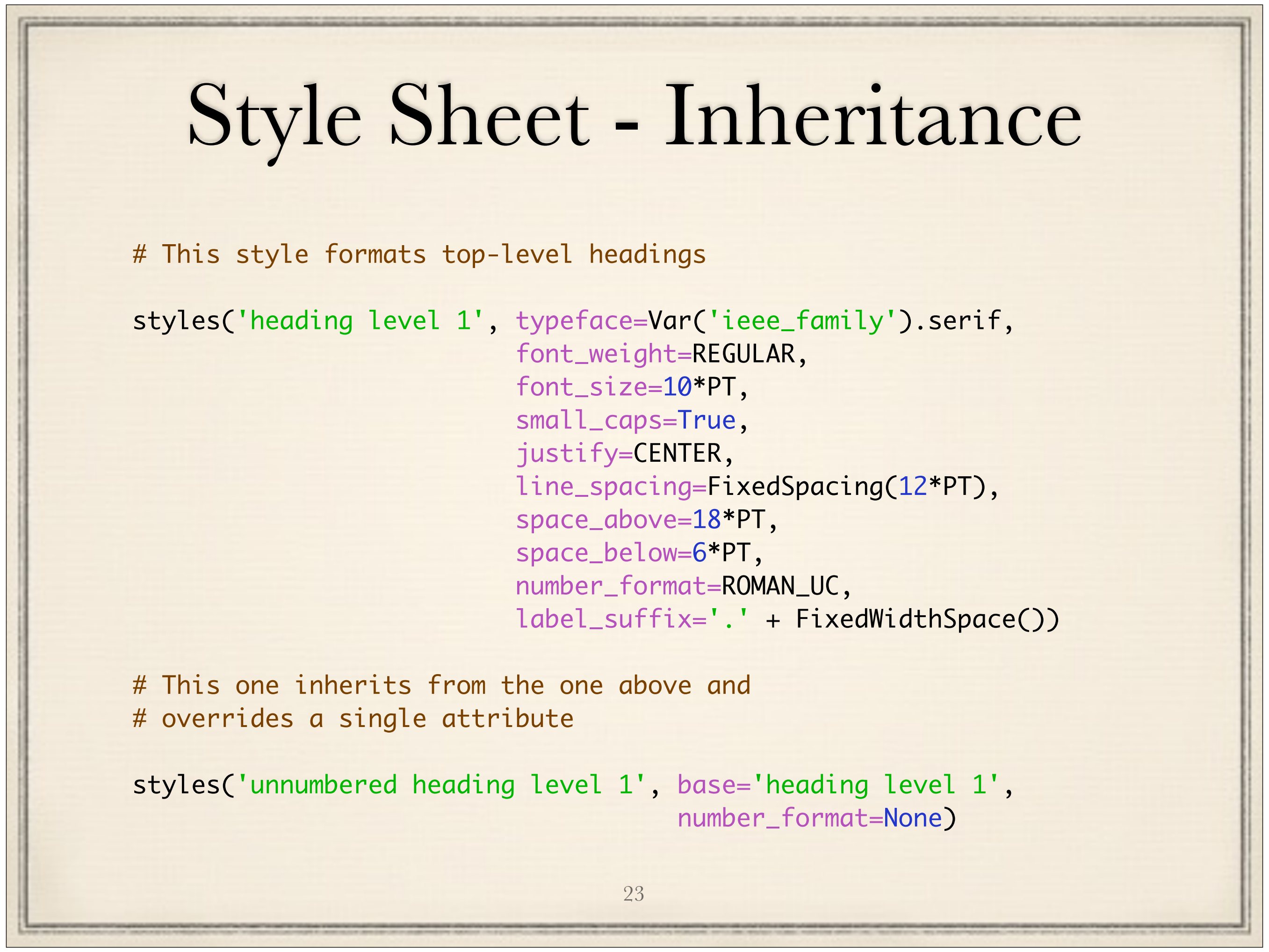
vs CSS - different elements can share a base style
avoids duplication



Style Sheet - Inheritance

```
# This style formats top-level headings

styles('heading level 1', typeface=Var('ieee_family').serif,
                          font_weight=REGULAR,
                          font_size=10*PT,
                          small_caps=True,
                          justify=CENTER,
                          line_spacing=FixedSpacing(12*PT),
                          space_above=18*PT,
                          space_below=6*PT,
                          number_format=ROMAN_UC,
                          label_suffix='.' + FixedWidthSpace())
```

vs CSS - different elements can share a base style
avoids duplication



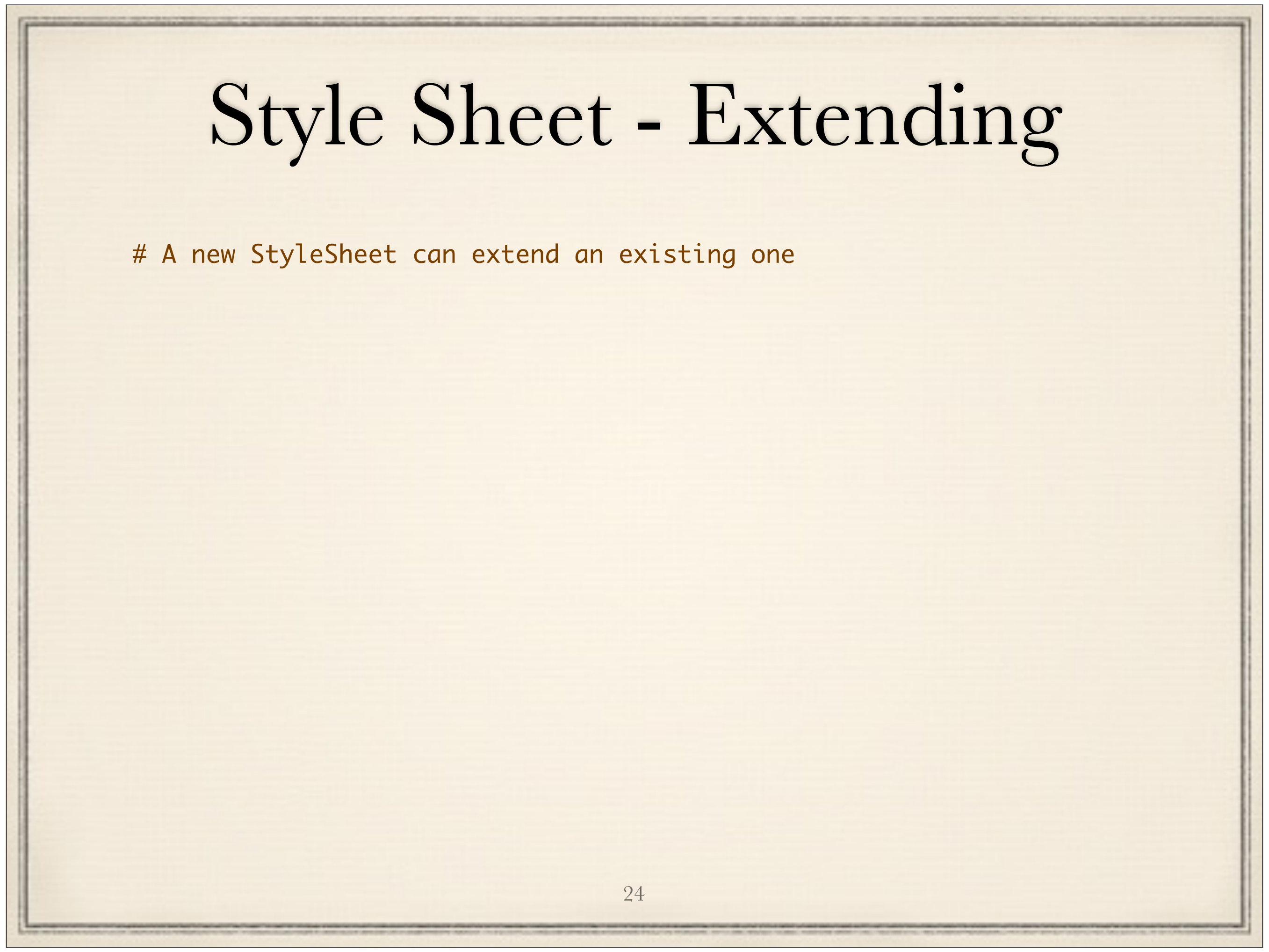
Style Sheet - Inheritance

```
# This style formats top-level headings

styles('heading level 1', typeface=Var('ieee_family').serif,
                          font_weight=REGULAR,
                          font_size=10*PT,
                          small_caps=True,
                          justify=CENTER,
                          line_spacing=FixedSpacing(12*PT),
                          space_above=18*PT,
                          space_below=6*PT,
                          number_format=ROMAN_UC,
                          label_suffix='.' + FixedWidthSpace())

# This one inherits from the one above and
# overrides a single attribute

styles('unnumbered heading level 1', base='heading level 1',
                                     number_format=None)
```

vs CSS - different elements can share a base style
avoids duplication

Style Sheet - Extending

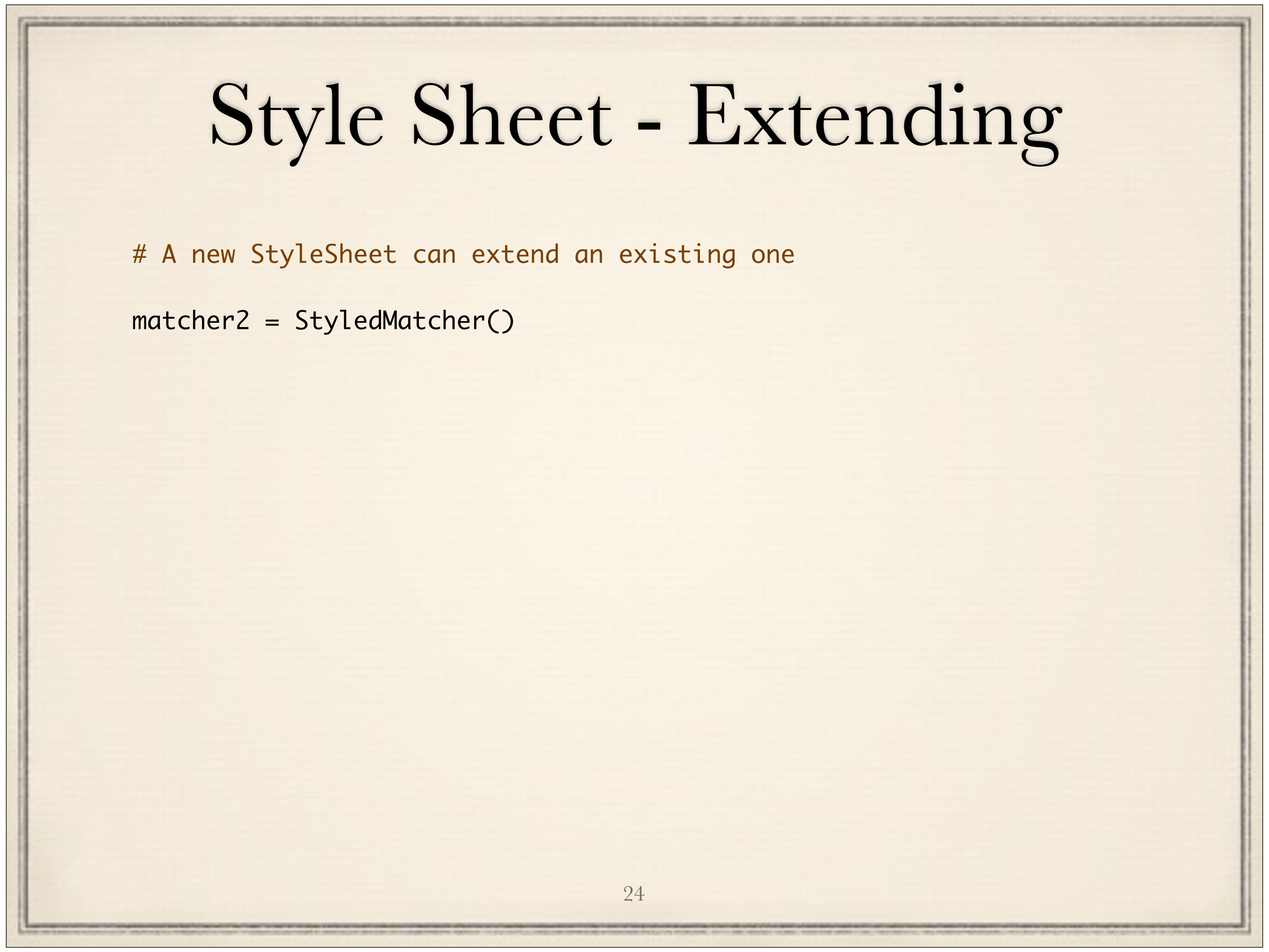
custom reStructuredText role
example :acronym:`RFC`



Style Sheet - Extending

```
# A new StyleSheet can extend an existing one
```

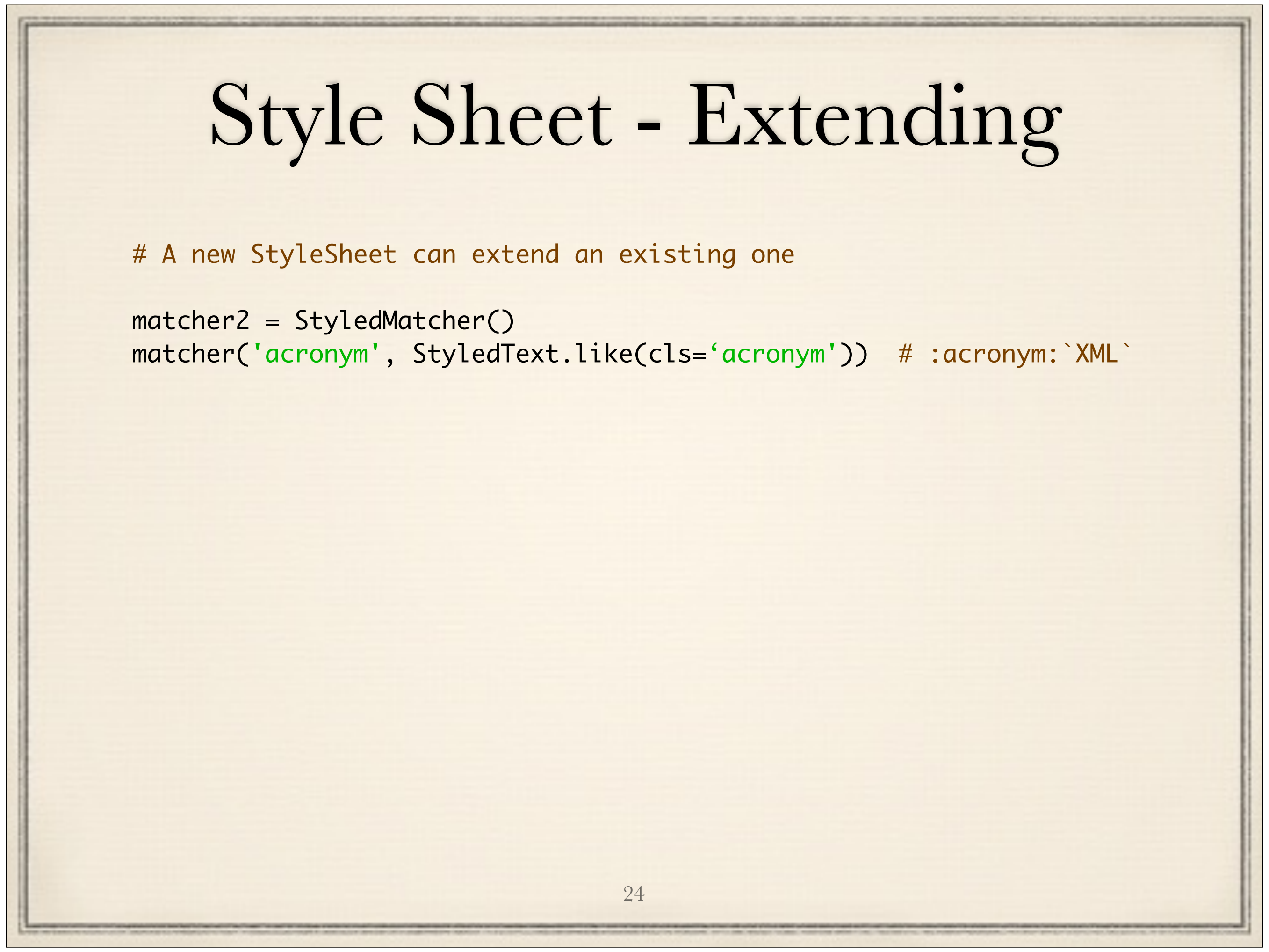
custom reStructuredText role
example :acronym:`RFC`



Style Sheet - Extending

```
# A new StyleSheet can extend an existing one

matcher2 = StyledMatcher()
```

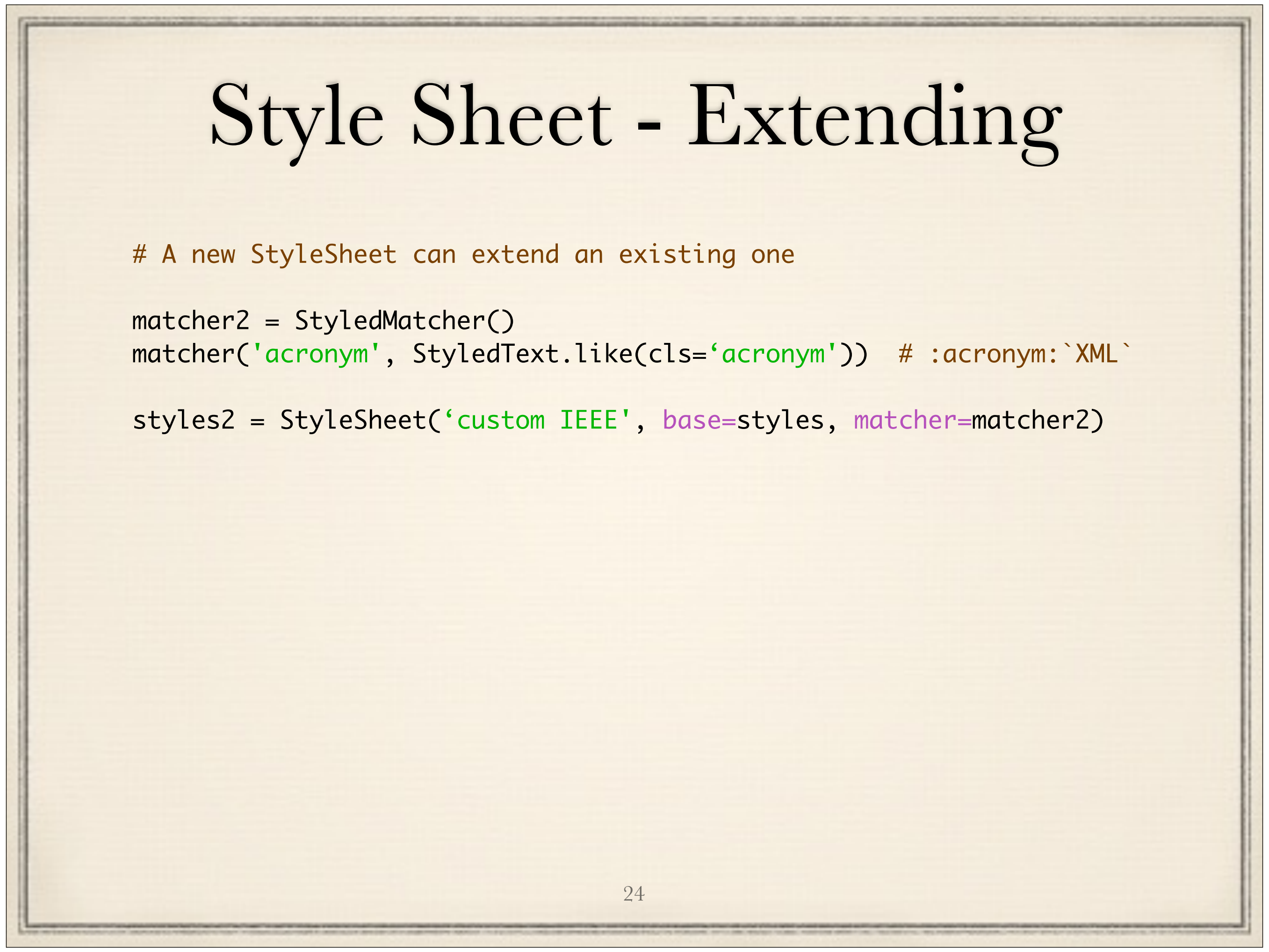
custom reStructuredText role
example :acronym:`RFC`



Style Sheet - Extending

```
# A new StyleSheet can extend an existing one

matcher2 = StyledMatcher()
matcher('acronym', StyledText.like(cls=‘acronym'))  # :acronym:`XML`
```

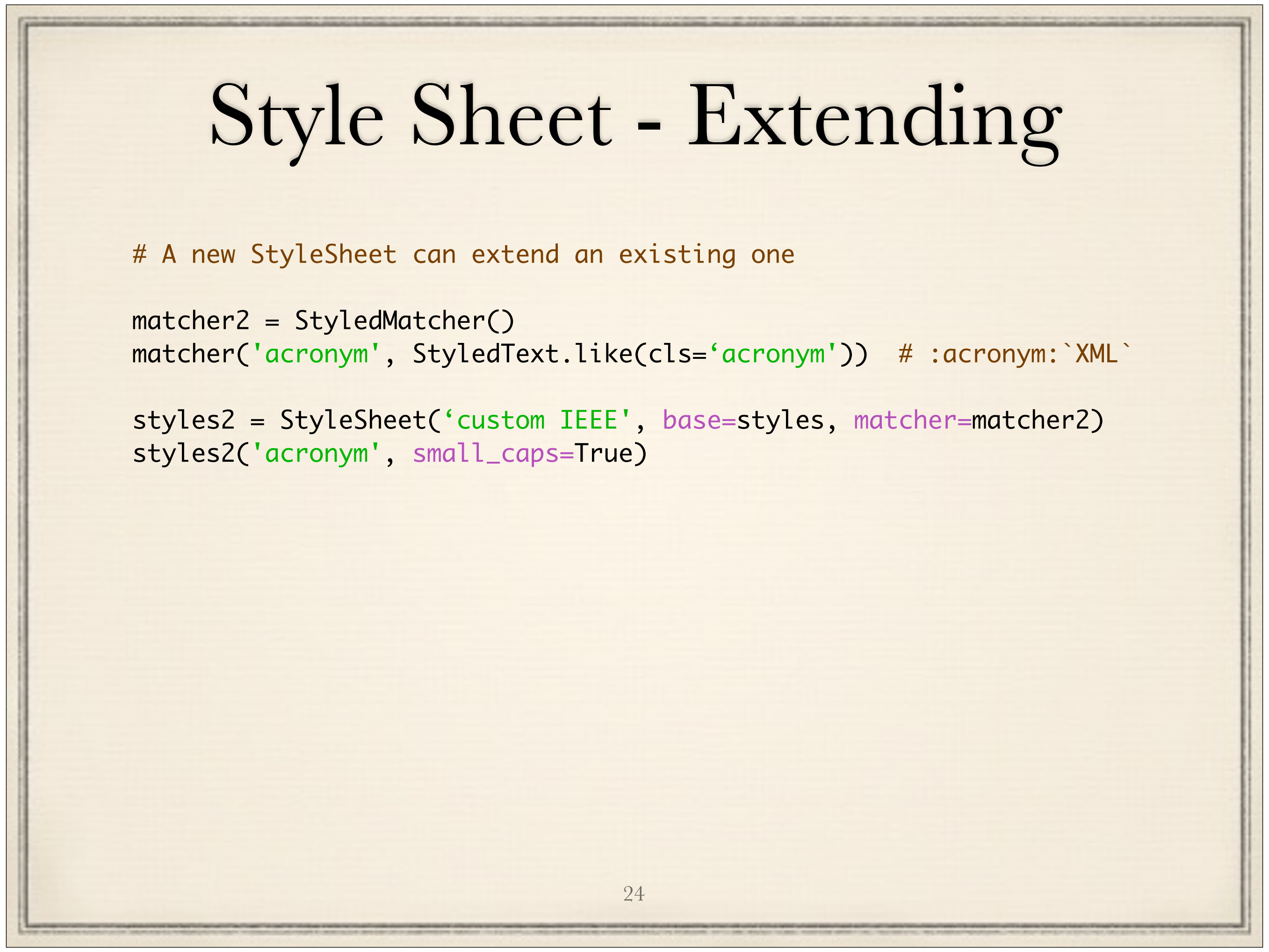
custom reStructuredText role
example :acronym:`RFC`



Style Sheet - Extending

```
# A new StyleSheet can extend an existing one

matcher2 = StyledMatcher()
matcher('acronym', StyledText.like(cls=‘acronym'))  # :acronym:`XML`

styles2 = StyleSheet(‘custom IEEE', base=styles, matcher=matcher2)
```

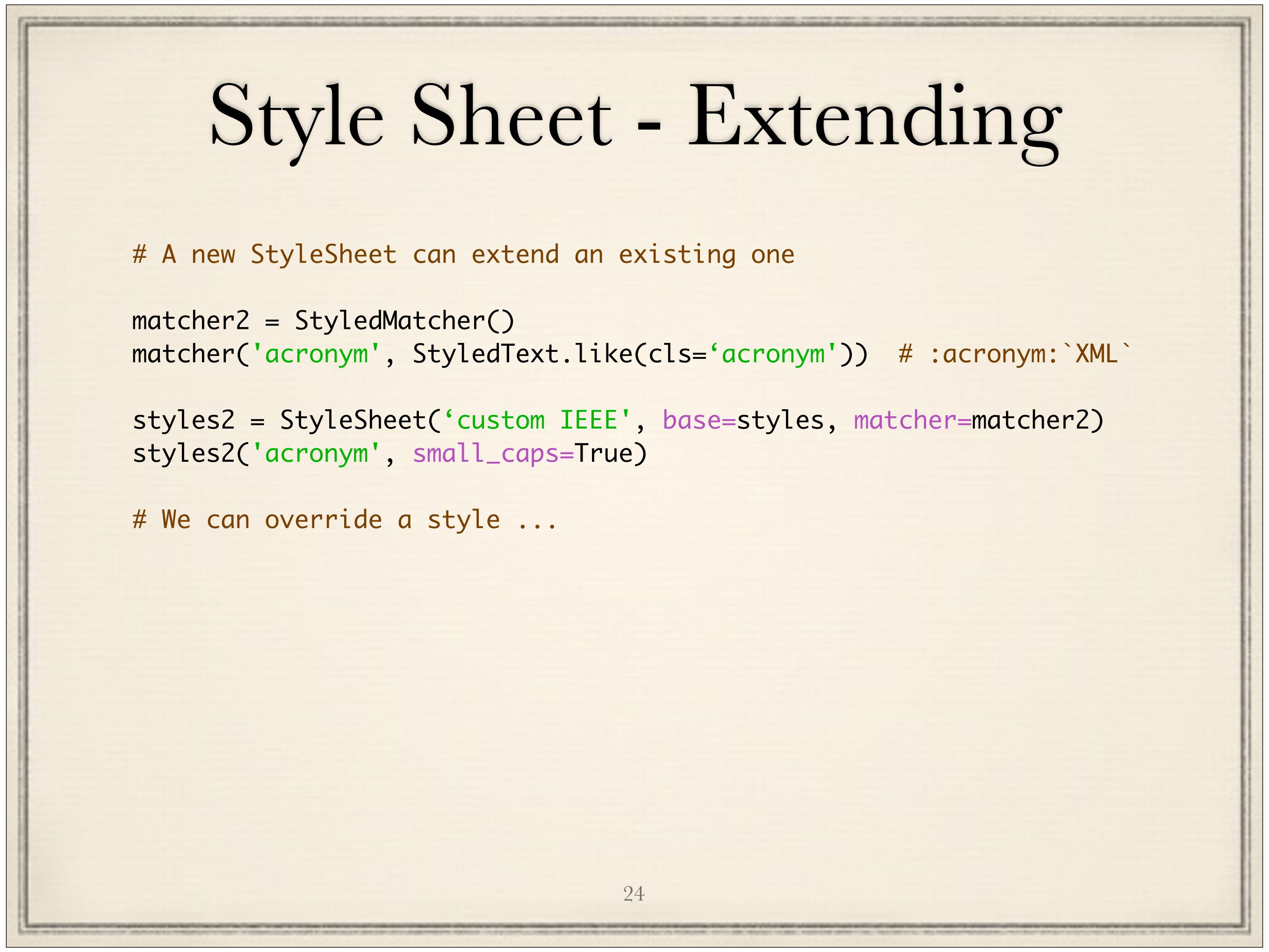
custom reStructuredText role
example :acronym:`RFC`



Style Sheet - Extending

```
# A new StyleSheet can extend an existing one

matcher2 = StyledMatcher()
matcher('acronym', StyledText.like(cls='acronym'))  # :acronym:`XML`

styles2 = StyleSheet('custom IEEE', base=styles, matcher=matcher2)
styles2('acronym', small_caps=True)
```

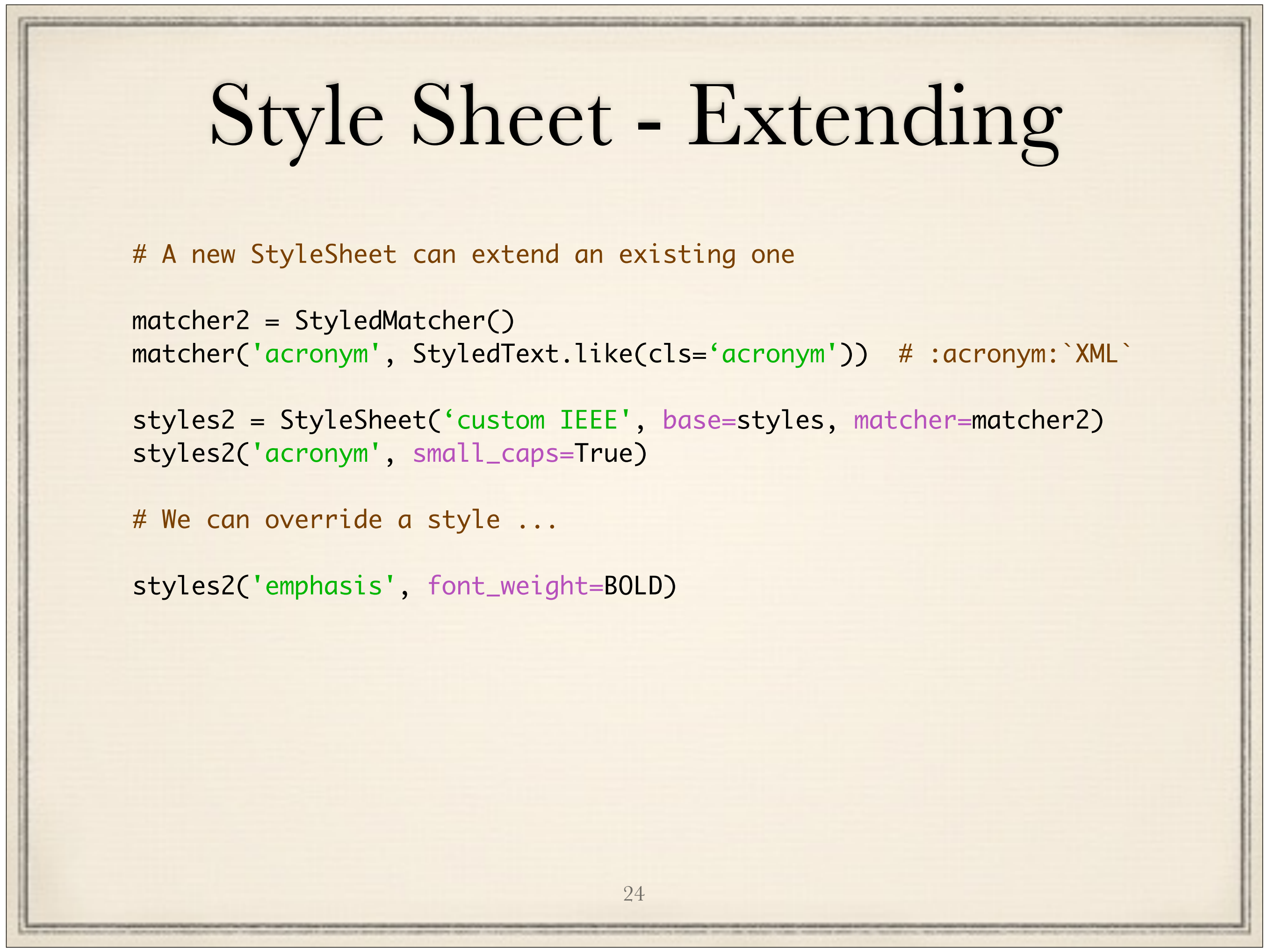
custom reStructuredText role
example :acronym:`RFC`



Style Sheet - Extending

```
# A new StyleSheet can extend an existing one

matcher2 = StyledMatcher()
matcher('acronym', StyledText.like(cls=‘acronym'))  # :acronym:`XML`

styles2 = StyleSheet(‘custom IEEE', base=styles, matcher=matcher2)
styles2('acronym', small_caps=True)

# We can override a style ...
```

custom reStructuredText role
example :acronym:`RFC`



Style Sheet - Extending

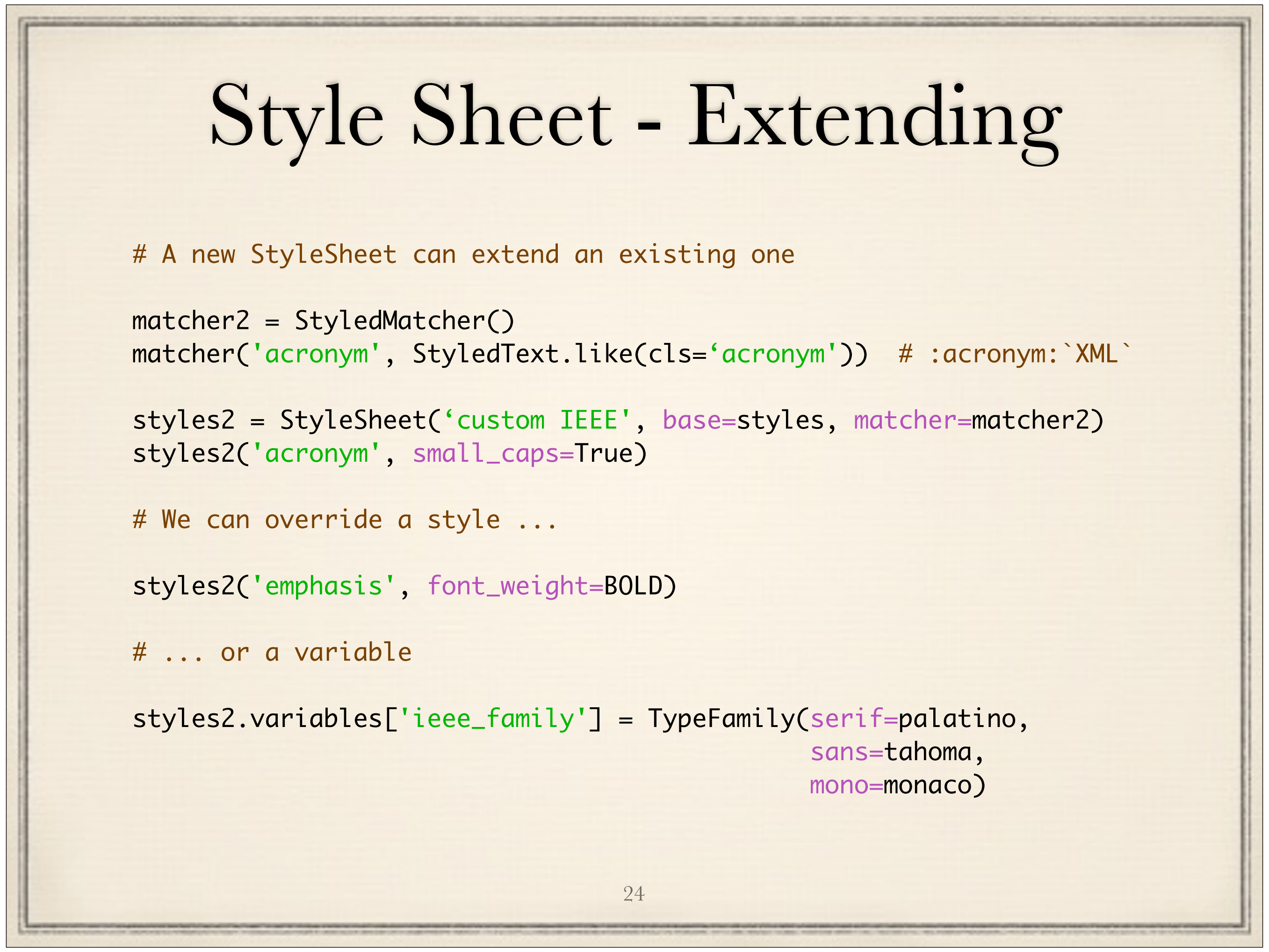
```
# A new StyleSheet can extend an existing one

matcher2 = StyledMatcher()
matcher('acronym', StyledText.like(cls=‘acronym')) # :acronym:`XML`

styles2 = StyleSheet(‘custom IEEE', base=styles, matcher=matcher2)
styles2('acronym', small_caps=True)

# We can override a style ...

styles2('emphasis', font_weight=BOLD)
```

custom reStructuredText role
example :acronym:`RFC`

Style Sheet - Extending

```
# A new StyleSheet can extend an existing one

matcher2 = StyledMatcher()
matcher('acronym', StyledText.like(cls=‘acronym'))  # :acronym:`XML`

styles2 = StyleSheet(‘custom IEEE', base=styles, matcher=matcher2)
styles2('acronym', small_caps=True)

# We can override a style ...

styles2('emphasis', font_weight=BOLD)

# ... or a variable
```

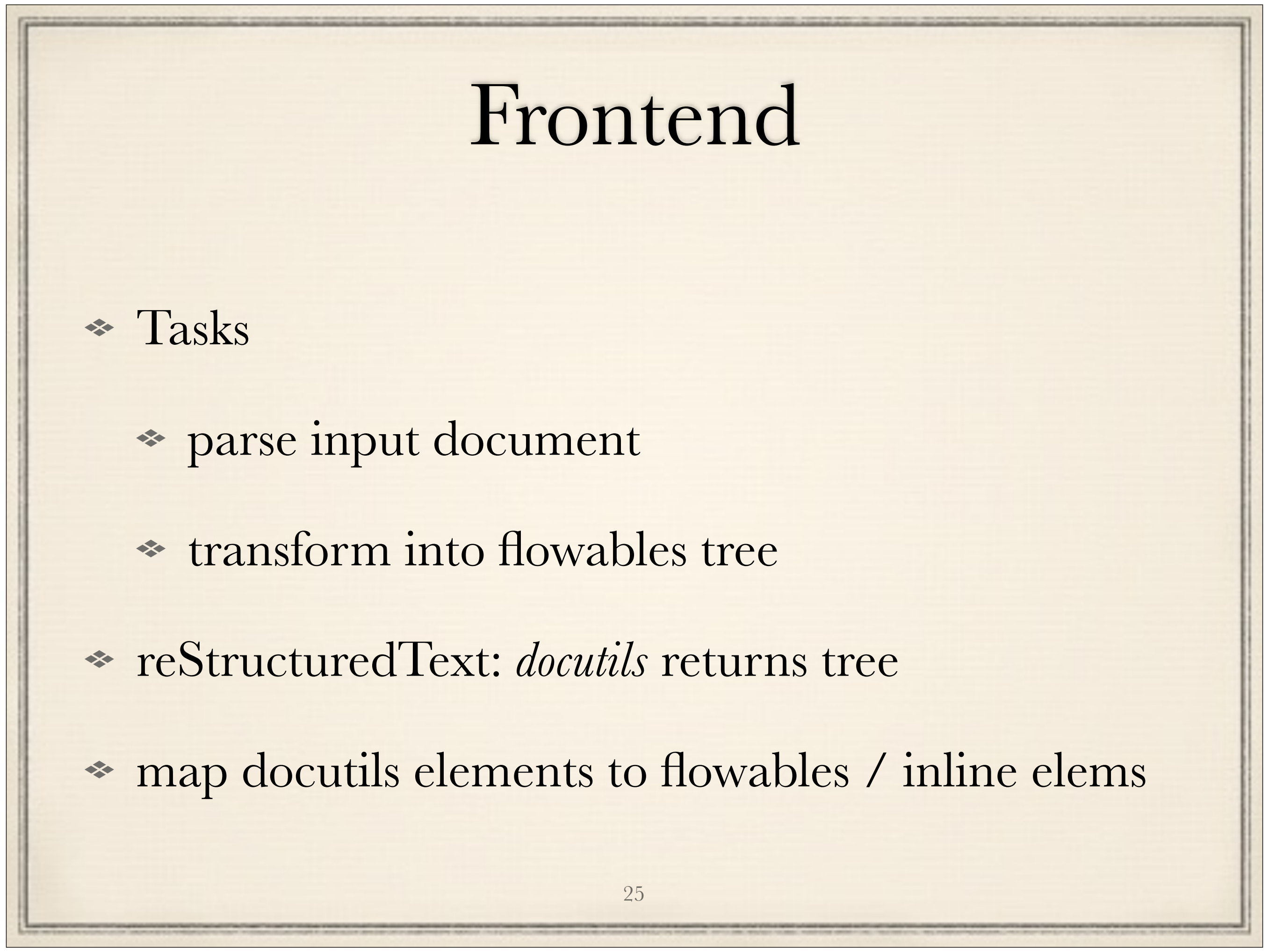
custom reStructuredText role
example :acronym:`RFC`



Style Sheet - Extending

```
# A new StyleSheet can extend an existing one

matcher2 = StyledMatcher()
matcher('acronym', StyledText.like(cls=‘acronym'))  # :acronym:`XML`

styles2 = StyleSheet(‘custom IEEE', base=styles, matcher=matcher2)
styles2('acronym', small_caps=True)

# We can override a style ...

styles2('emphasis', font_weight=BOLD)

# ... or a variable

styles2.variables['ieee_family'] = TypeFamily(serif=palatino,
                                              sans=tahoma,
                                              mono=monaco)
```

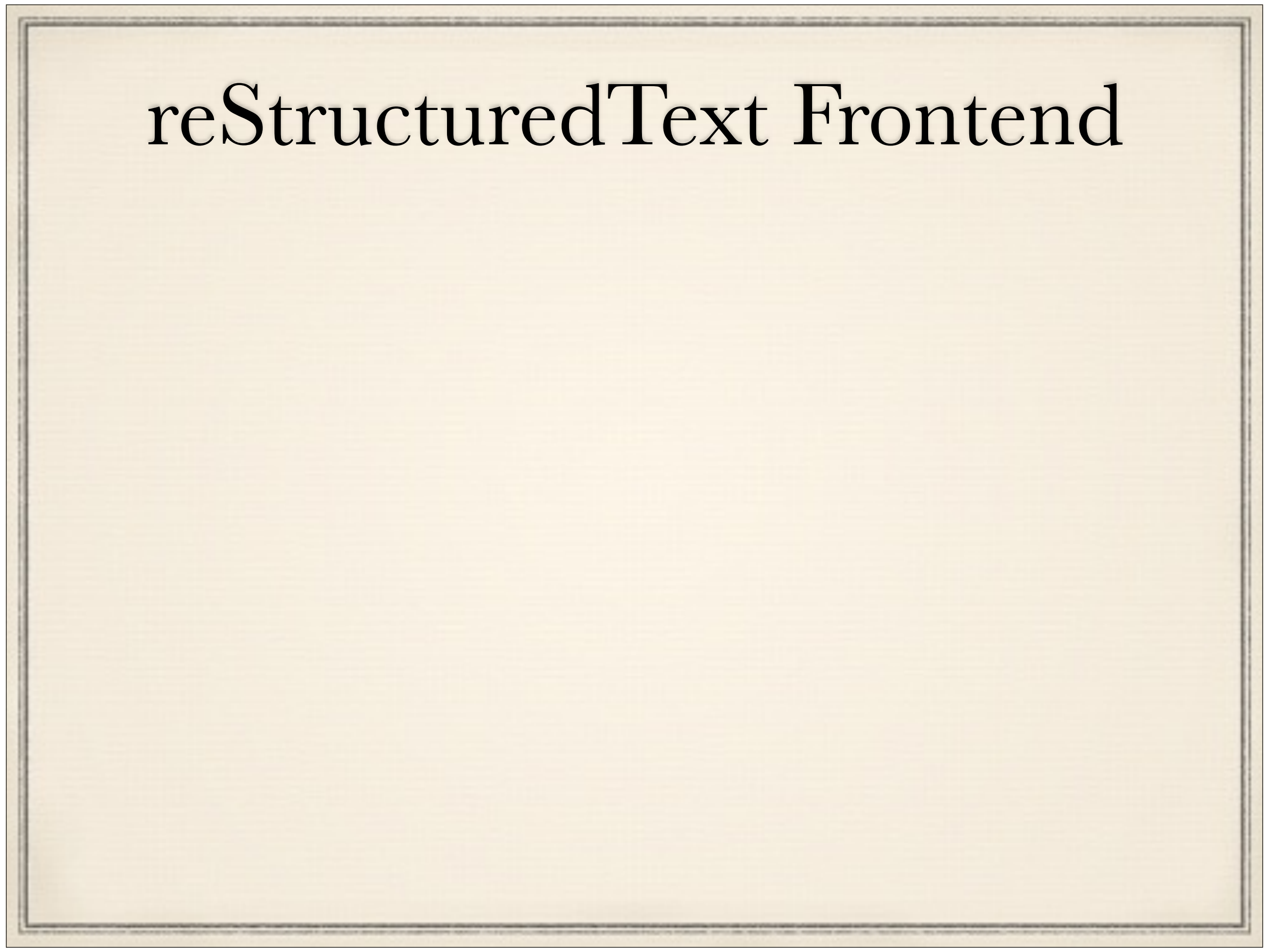
custom reStructuredText role
example :acronym:`RFC`



Frontend

- Tasks
  - parse input document
  - transform into flowables tree
- reStructuredText: *docutils* returns tree
- map docutils elements to flowables / inline elems

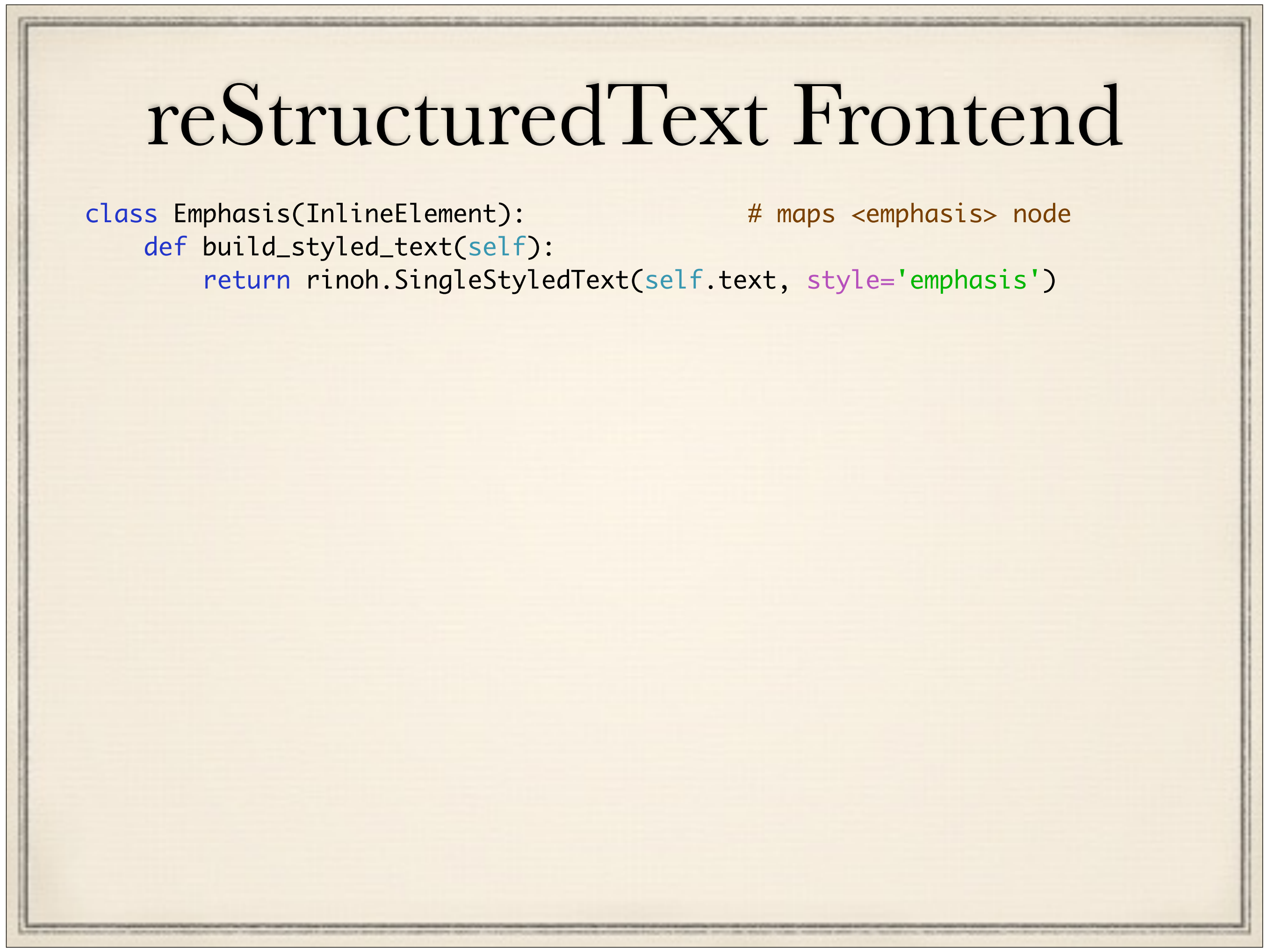
docutils = reference reStructuredText parser



reStructuredText Frontend

very short view of how input files are handled

- node names **map to** class names
- each element in the RinohType flowables tree has reference back to source element
    - warning/error messages can refer to it
- some nodes can represent both body and inline elements

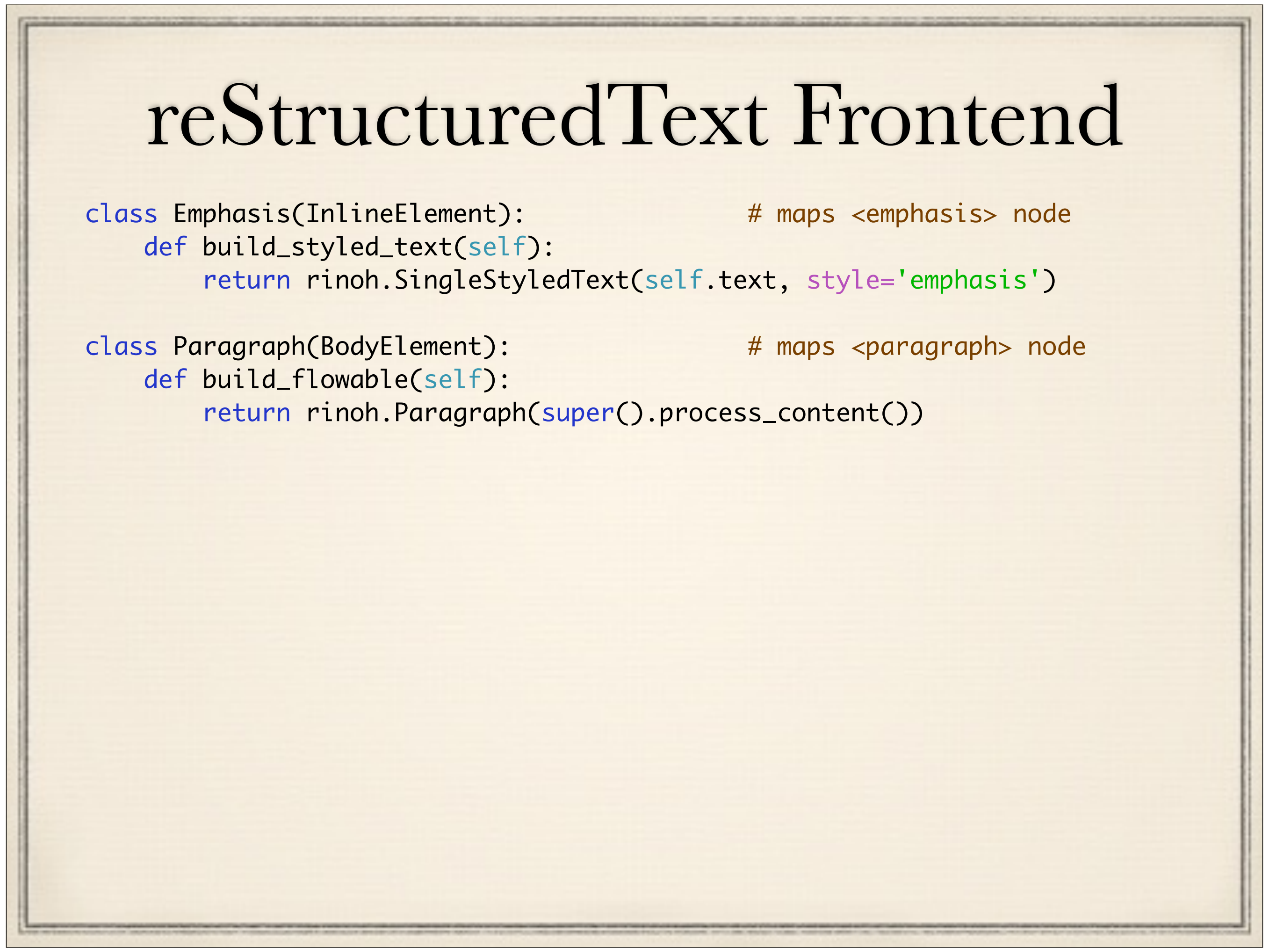


reStructuredText Frontend

```python
class Emphasis(InlineElement):               # maps <emphasis> node
    def build_styled_text(self):
        return rinoh.SingleStyledText(self.text, style='emphasis')
```

very short view of how input files are handled

- node names **map to** class names
- each element in the RinohType flowables tree has reference back to source element
    - warning/error messages can refer to it
- some nodes can represent both body and inline elements

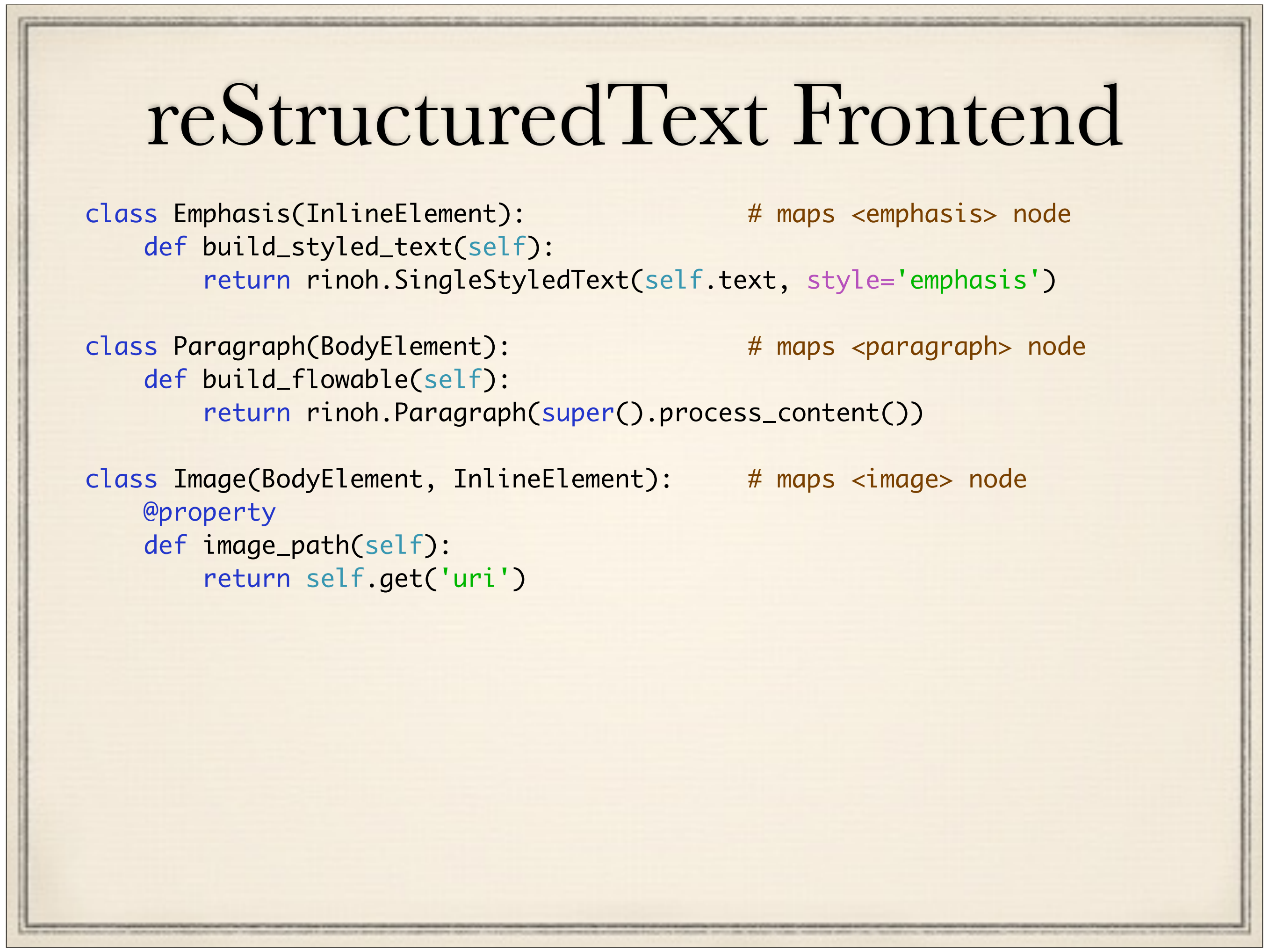


reStructuredText Frontend

```
class Emphasis(InlineElement):                # maps <emphasis> node
    def build_styled_text(self):
        return rinoh.SingleStyledText(self.text, style='emphasis')

class Paragraph(BodyElement):                 # maps <paragraph> node
    def build_flowable(self):
        return rinoh.Paragraph(super().process_content())
```

very short view of how input files are handled

- node names **map to** class names
- each element in the RinohType flowables tree has reference back to source element
    - warning/error messages can refer to it
- some nodes can represent both body and inline elements

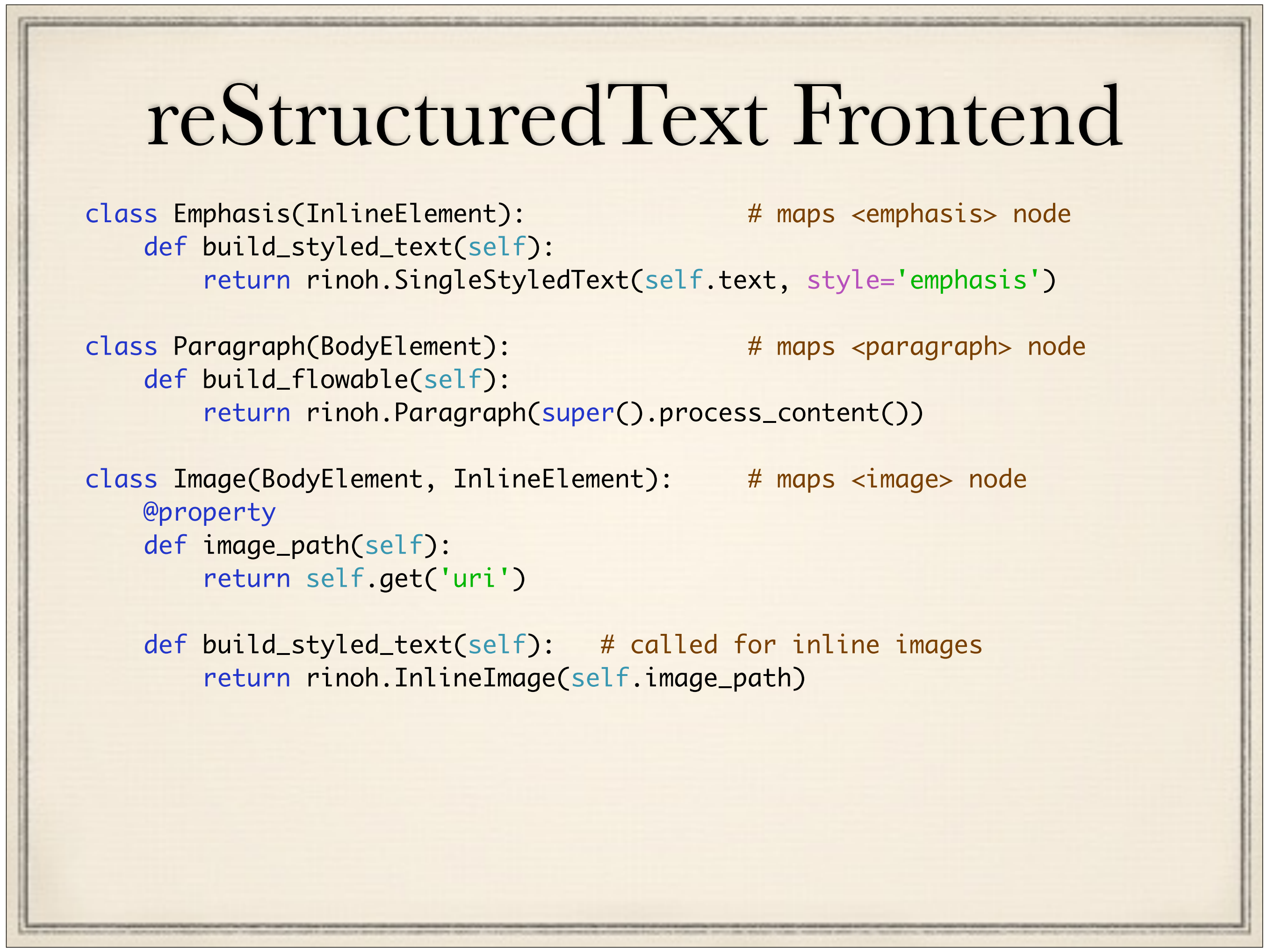


reStructuredText Frontend

```python
class Emphasis(InlineElement):                # maps <emphasis> node
    def build_styled_text(self):
        return rinoh.SingleStyledText(self.text, style='emphasis')

class Paragraph(BodyElement):                 # maps <paragraph> node
    def build_flowable(self):
        return rinoh.Paragraph(super().process_content())

class Image(BodyElement, InlineElement):      # maps <image> node
    @property
    def image_path(self):
        return self.get('uri')
```

very short view of how input files are handled

- node names **map to** class names
- each element in the RinohType flowables tree has reference back to source element
    - warning/error messages can refer to it
- some nodes can represent both body and inline elements

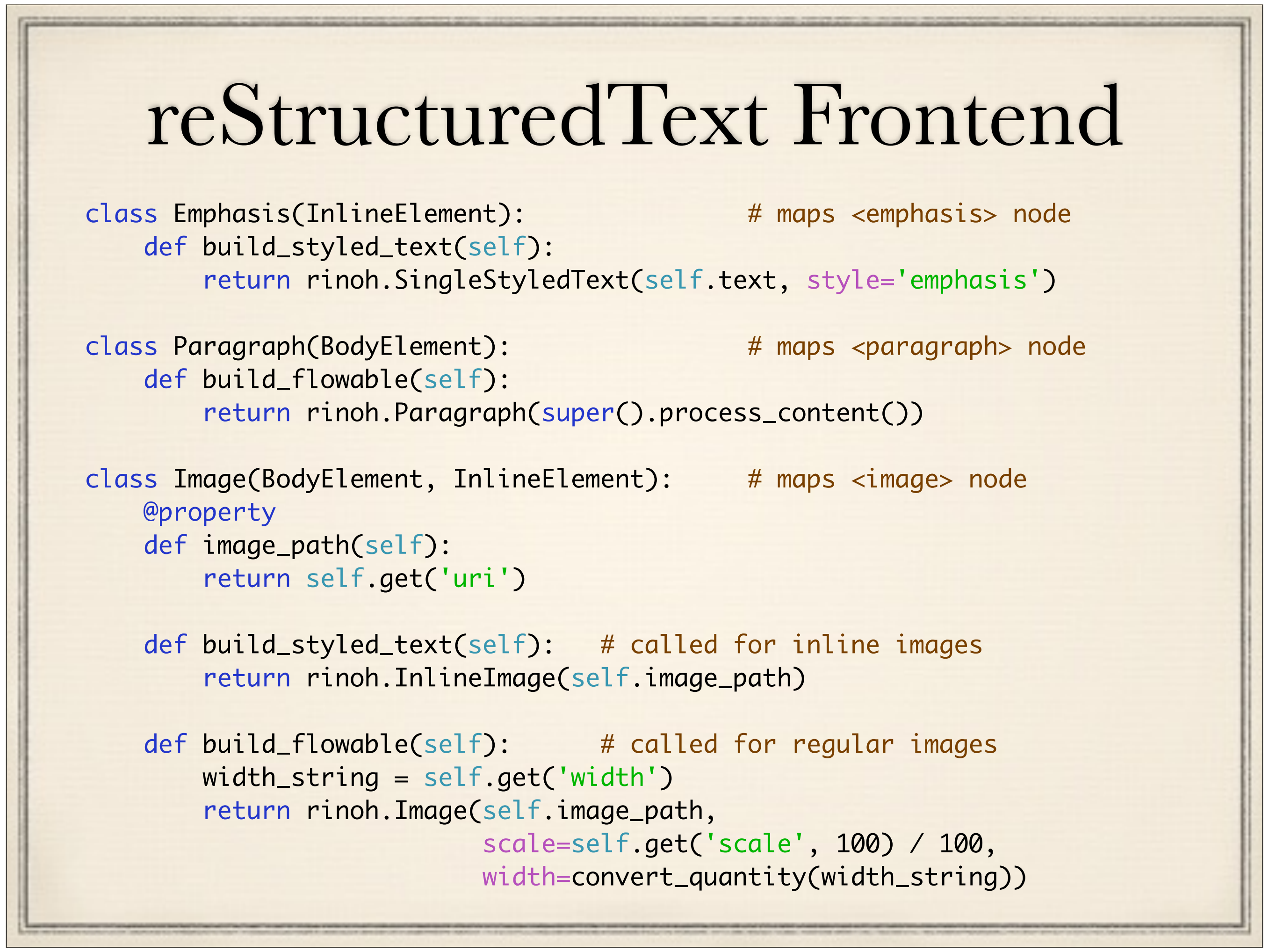


reStructuredText Frontend

```
class Emphasis(InlineElement):                # maps <emphasis> node
    def build_styled_text(self):
        return rinoh.SingleStyledText(self.text, style='emphasis')

class Paragraph(BodyElement):                 # maps <paragraph> node
    def build_flowable(self):
        return rinoh.Paragraph(super().process_content())

class Image(BodyElement, InlineElement):      # maps <image> node
    @property
    def image_path(self):
        return self.get('uri')

    def build_styled_text(self):   # called for inline images
        return rinoh.InlineImage(self.image_path)
```

very short view of how input files are handled

- node names **map to** class names
- each element in the RinohType flowables tree has reference back to source element
    - warning/error messages can refer to it
- some nodes can represent both body and inline elements



reStructuredText Frontend

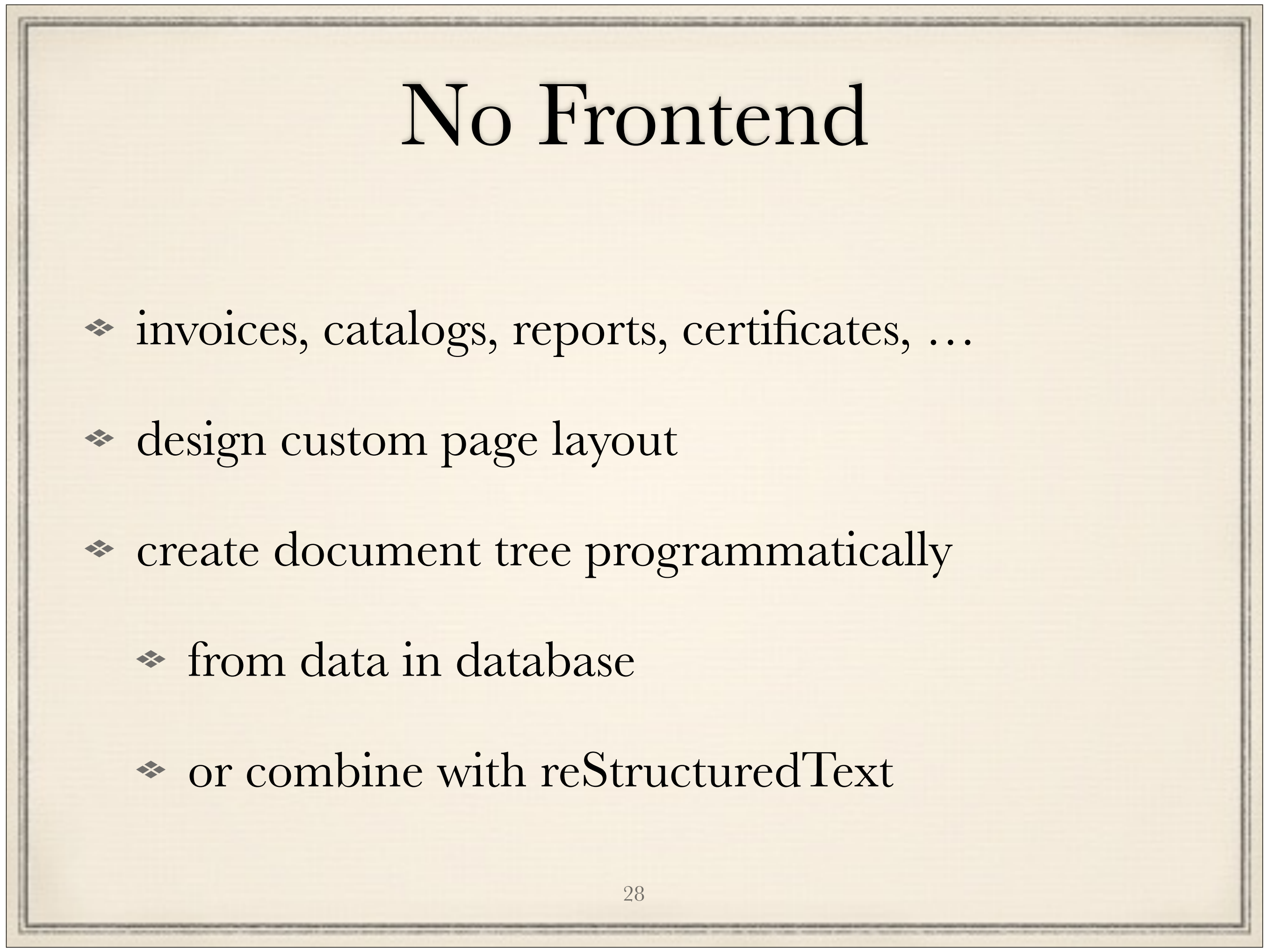
```
class Emphasis(InlineElement):               # maps <emphasis> node
    def build_styled_text(self):
        return rinoh.SingleStyledText(self.text, style='emphasis')

class Paragraph(BodyElement):                # maps <paragraph> node
    def build_flowable(self):
        return rinoh.Paragraph(super().process_content())

class Image(BodyElement, InlineElement):     # maps <image> node
    @property
    def image_path(self):
        return self.get('uri')

    def build_styled_text(self):   # called for inline images
        return rinoh.InlineImage(self.image_path)

    def build_flowable(self):      # called for regular images
        width_string = self.get('width')
        return rinoh.Image(self.image_path,
                           scale=self.get('scale', 100) / 100,
                           width=convert_quantity(width_string))
```

very short view of how input files are handled

- node names **map to** class names
- each element in the RinohType flowables tree has reference back to source element
    - warning/error messages can refer to it
- some nodes can represent both body and inline elements

Other Frontends

- same approach can be used for
  - Markdown (using Mistune)
  - DocBook & custom XML formats
  - HTML
  - LaTeX (using PlasTeX)



No Frontend

- invoices, catalogs, reports, certificates, …
- design custom page layout
- create document tree programmatically
  - from data in database
  - or combine with reStructuredText

not just limited to technical documents
folders: use PDFs as background (no examples yet)

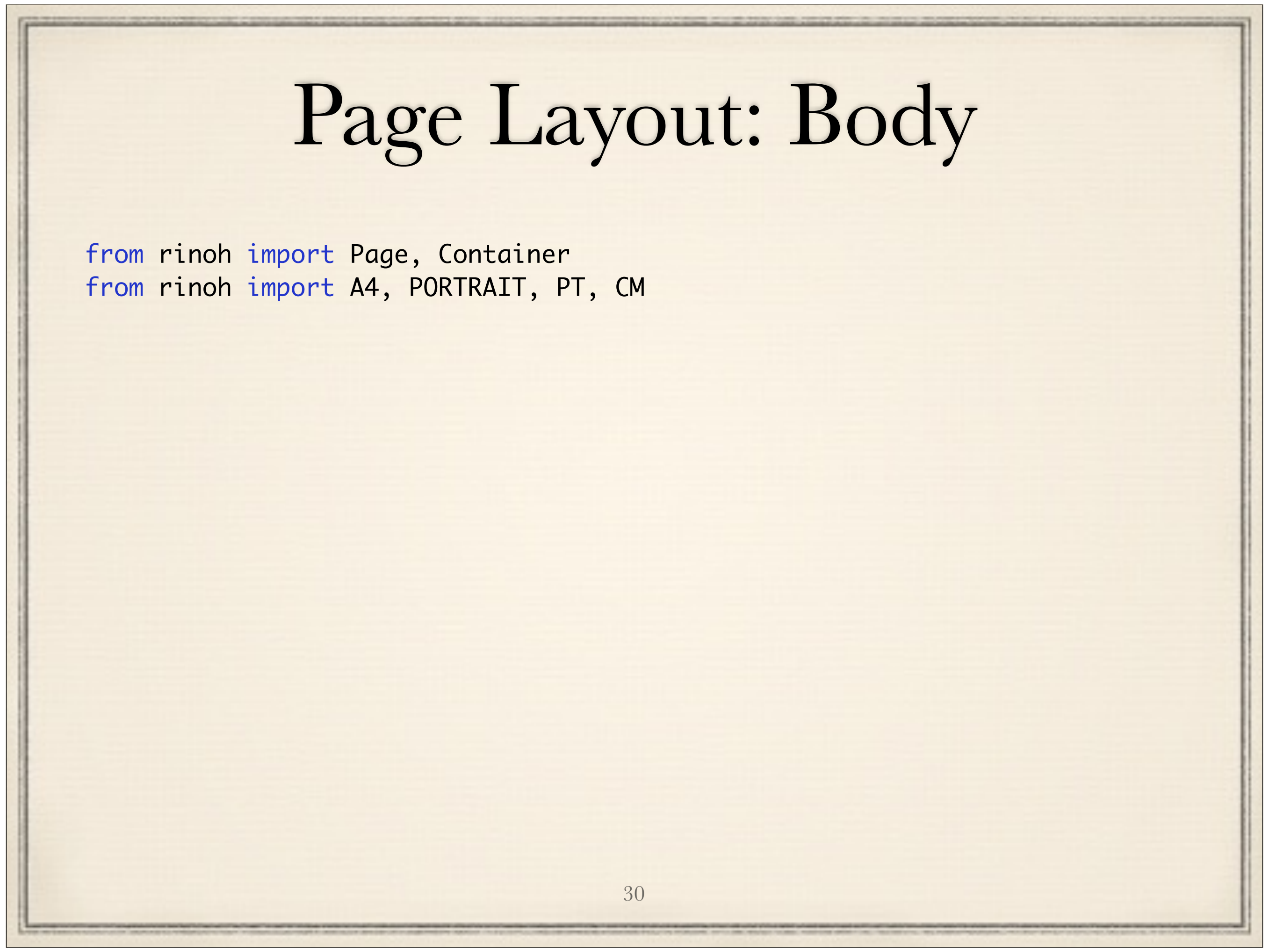
~ ReportLab

Page Layout

- Container: area on page where content is put
  - Page = top-level container
  - ExpandingContainer: enables footnotes, floats
- Chain: link containers (across pages)

Page Layout: Body

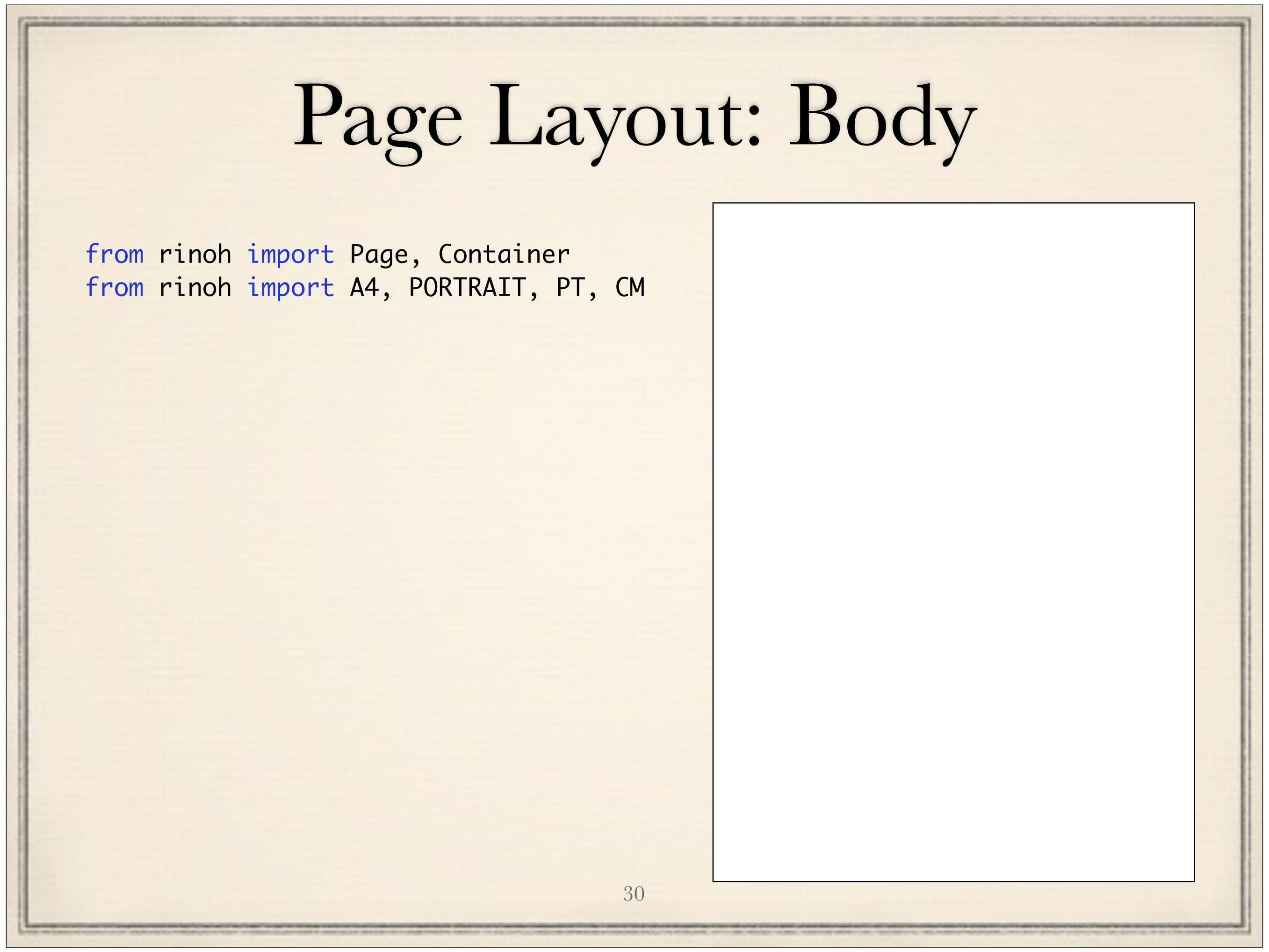
Page = top-level container
container's position is specified relative to parent



Page Layout: Body

```python
from rinoh import Page, Container
from rinoh import A4, PORTRAIT, PT, CM
```

Page = top-level container
container's position is specified relative to parent



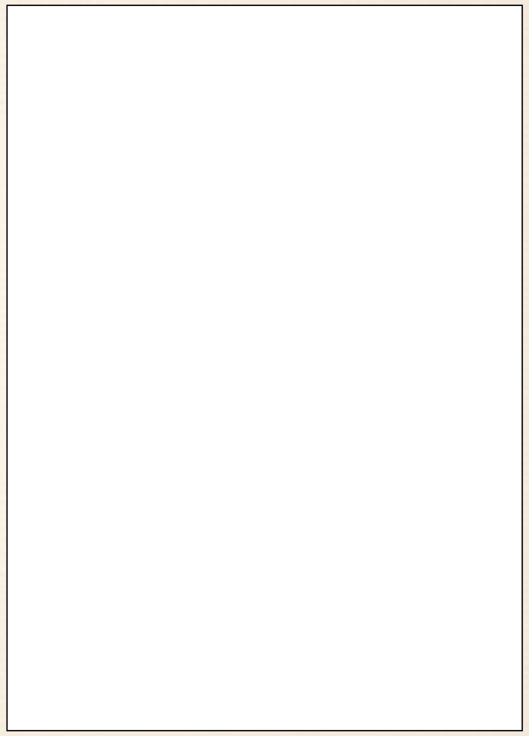
Page Layout: Body

```python
from rinoh import Page, Container
from rinoh import A4, PORTRAIT, PT, CM
```

Page = top-level container
container's position is specified relative to parent

Page Layout: Body

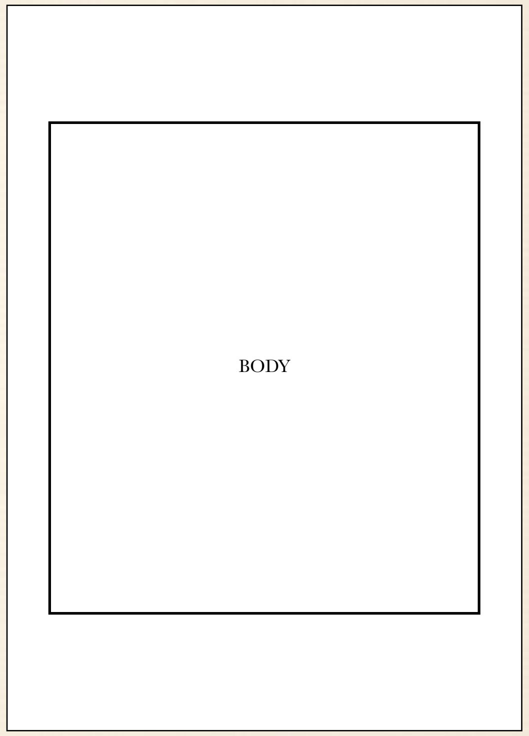
```
from rinoh import Page, Container
from rinoh import A4, PORTRAIT, PT, CM


page = Page(document_part, A4, PORTRAIT)
```



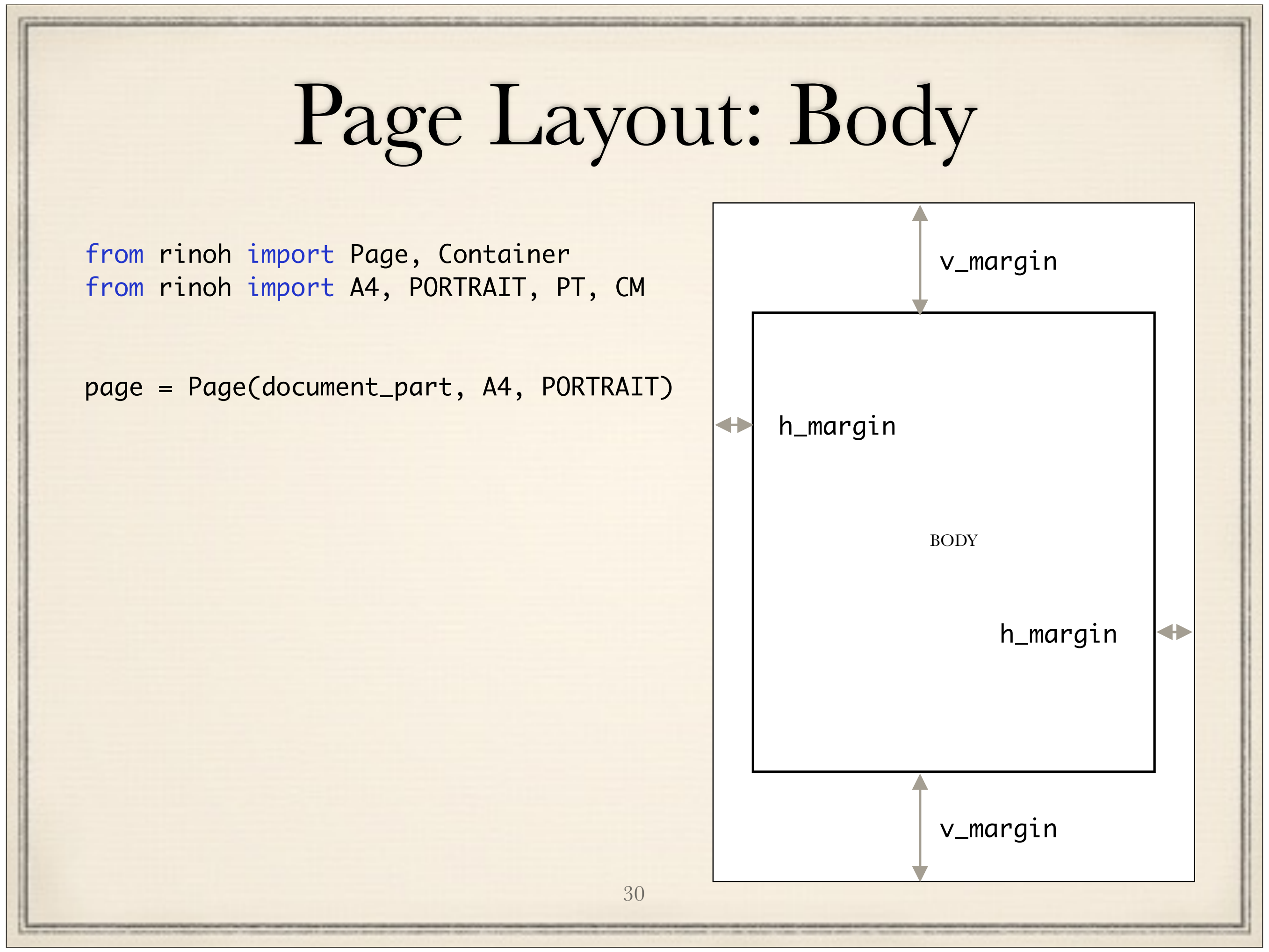
Page = top-level container
container's position is specified relative to parent

Page Layout: Body

```
from rinoh import Page, Container
from rinoh import A4, PORTRAIT, PT, CM


page = Page(document_part, A4, PORTRAIT)
```

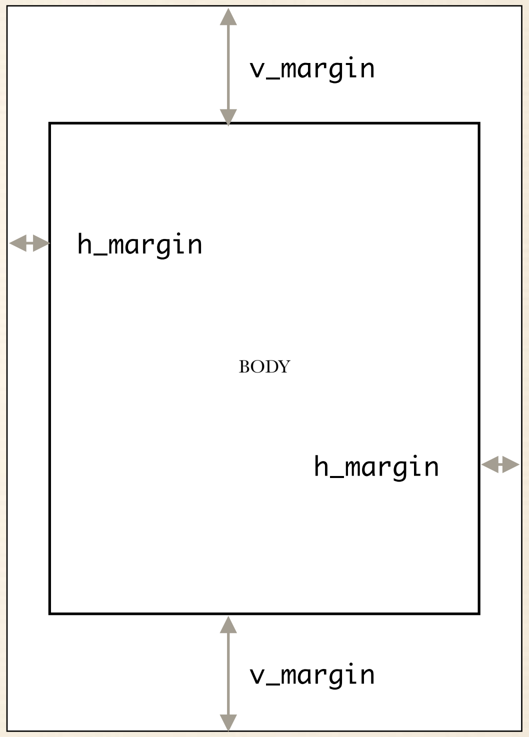


Page = top-level container
container's position is specified relative to parent



Page Layout: Body

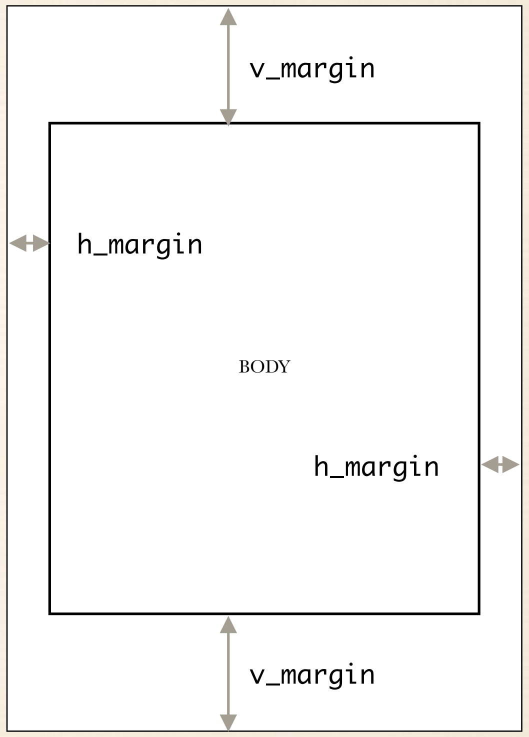
```
from rinoh import Page, Container
from rinoh import A4, PORTRAIT, PT, CM


page = Page(document_part, A4, PORTRAIT)
```



Page = top-level container
container's position is specified relative to parent

Page Layout: Body

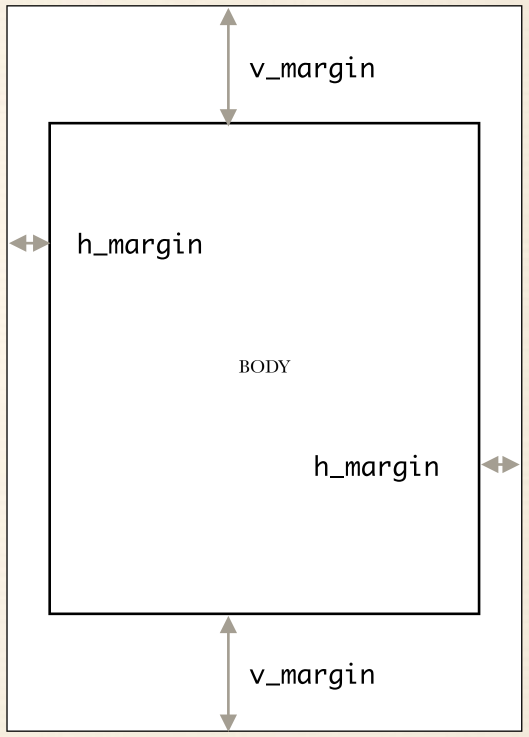
```
from rinoh import Page, Container
from rinoh import A4, PORTRAIT, PT, CM


page = Page(document_part, A4, PORTRAIT)

h_margin = 2*CM
v_margin = 4*CM
```



Page = top-level container
container's position is specified relative to parent

Page Layout: Body

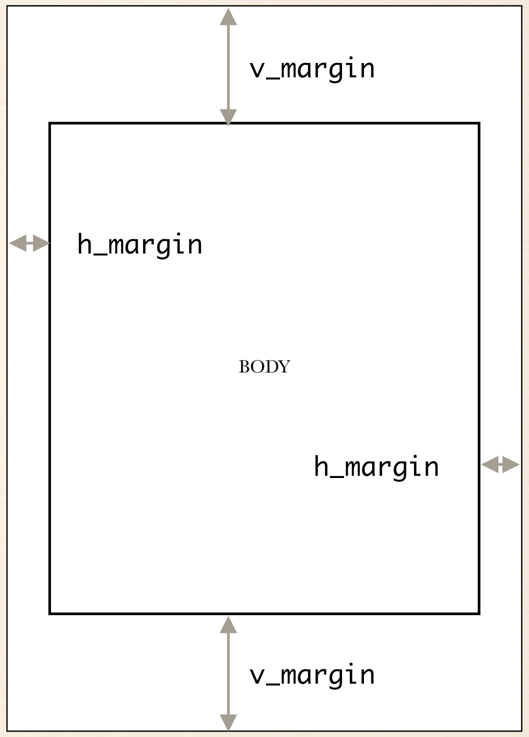
```
from rinoh import Page, Container
from rinoh import A4, PORTRAIT, PT, CM


page = Page(document_part, A4, PORTRAIT)

h_margin = 2*CM
v_margin = 4*CM
body_width = page.width - 2 * h_margin
body_height = page.height - 2 * v_margin
```



Page = top-level container
container's position is specified relative to parent

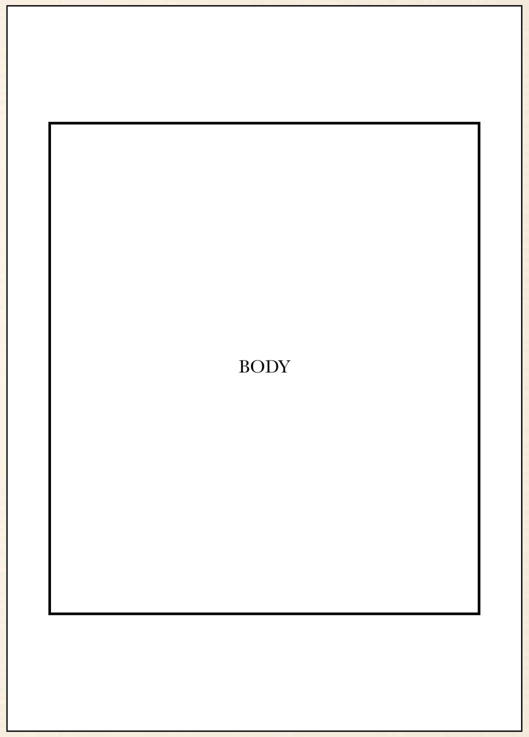
Page Layout: Body

```
from rinoh import Page, Container
from rinoh import A4, PORTRAIT, PT, CM


page = Page(document_part, A4, PORTRAIT)

h_margin = 2*CM
v_margin = 4*CM
body_width = page.width - 2 * h_margin
body_height = page.height - 2 * v_margin

body = Container('body',
                 parent=page,
                 left=h_margin,
                 top=v_margin,
                 width=body_width,
                 height=body_height)
```



Page = top-level container
container's position is specified relative to parent

Page Layout: Header & Footer

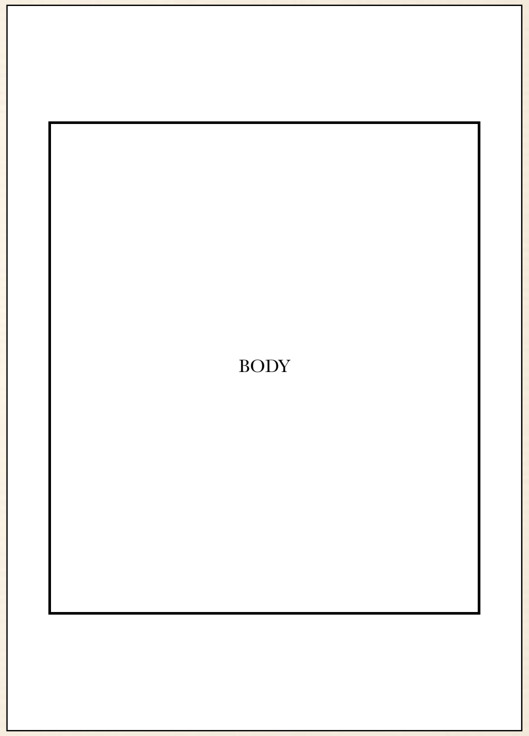


contents of footer have unknown height

DownExpandingContainer: initially 0 height, grows

Page Layout: Header & Footer

```python
from rinoh import DownExpandingContainer
from rinoh import UpExpandingContainer
from rinoh import PT
```

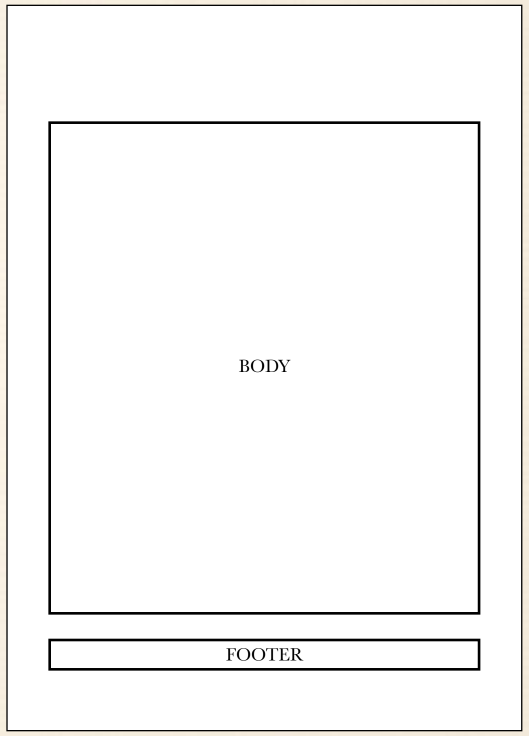


contents of footer have unknown height

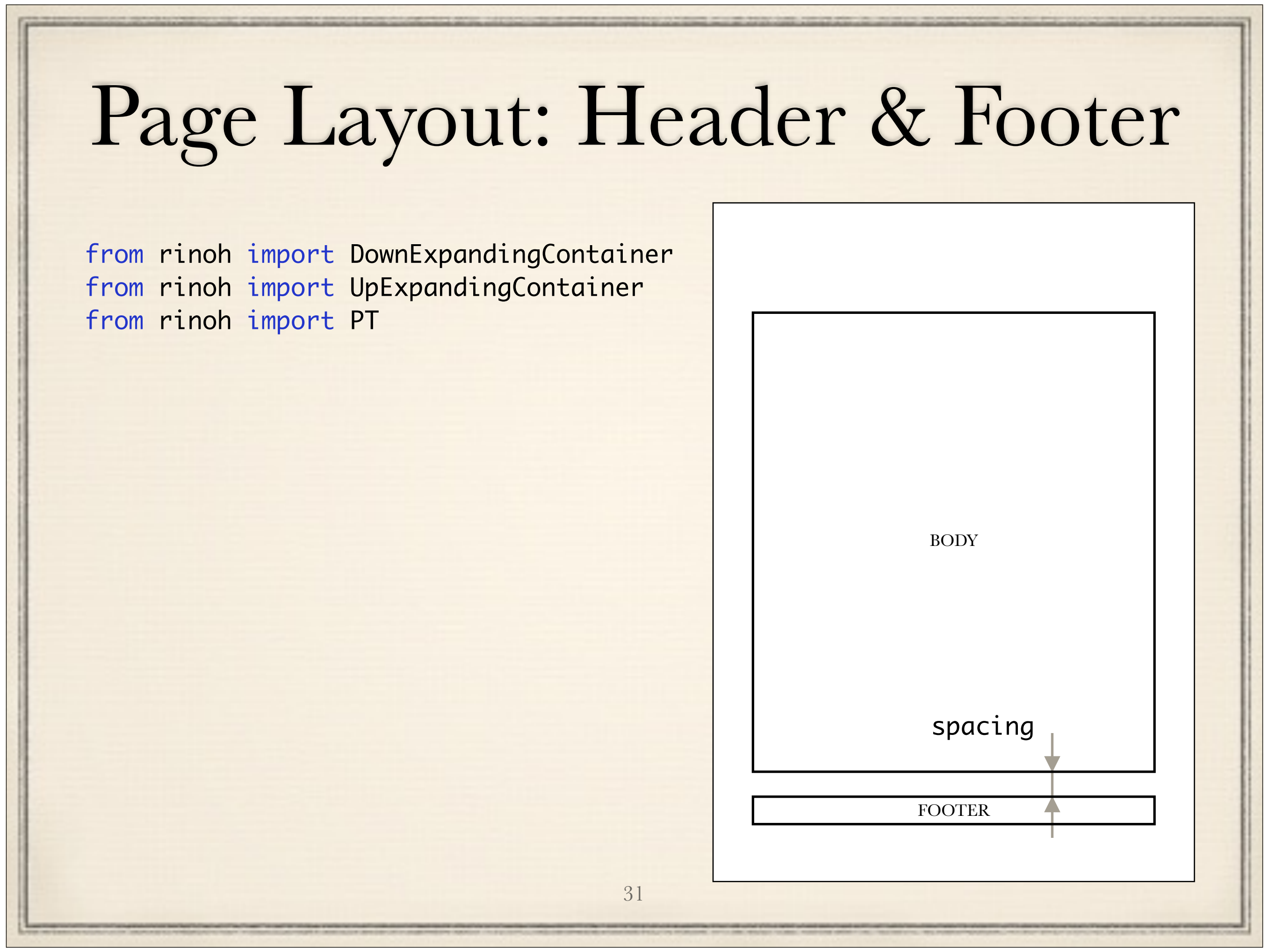
DownExpandingContainer: initially 0 height, grows

Page Layout: Header & Footer

```
from rinoh import DownExpandingContainer
from rinoh import UpExpandingContainer
from rinoh import PT
```

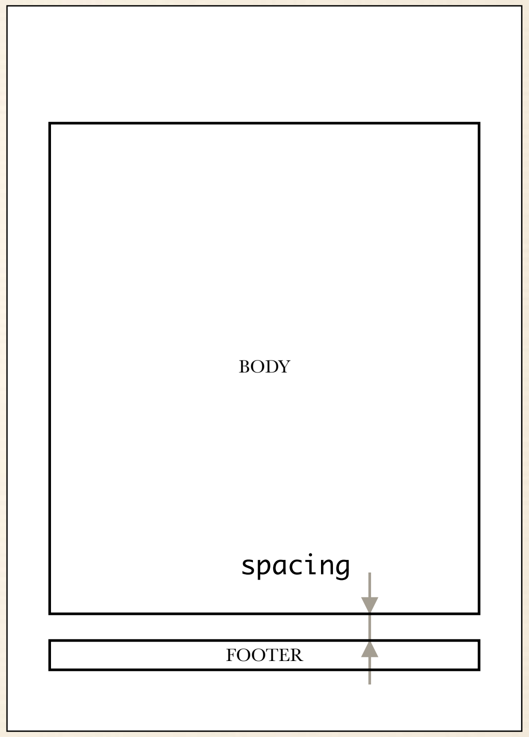


contents of footer have unknown height
DownExpandingContainer: initially 0 height, grows

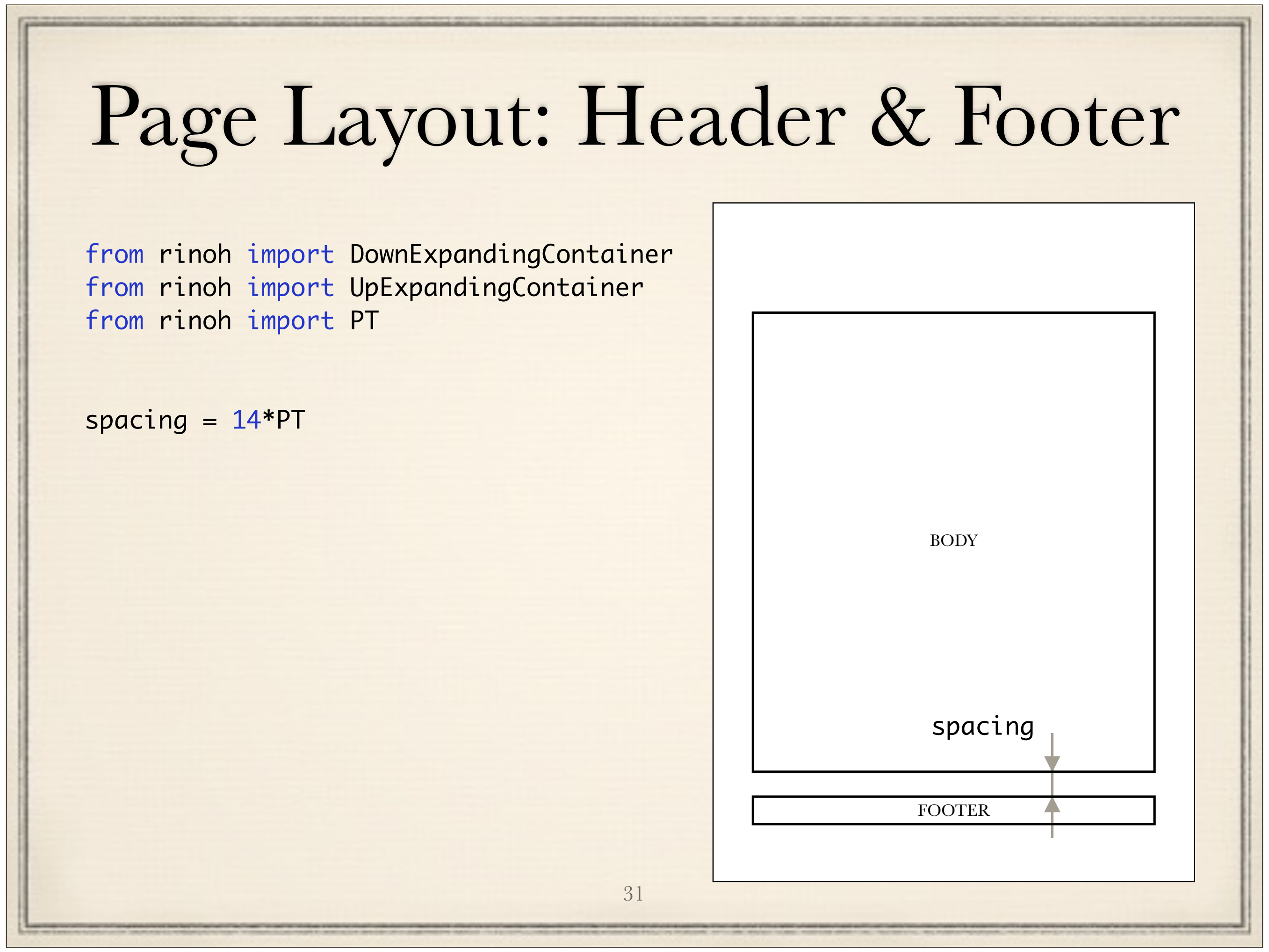


Page Layout: Header & Footer

```
from rinoh import DownExpandingContainer
from rinoh import UpExpandingContainer
from rinoh import PT
```

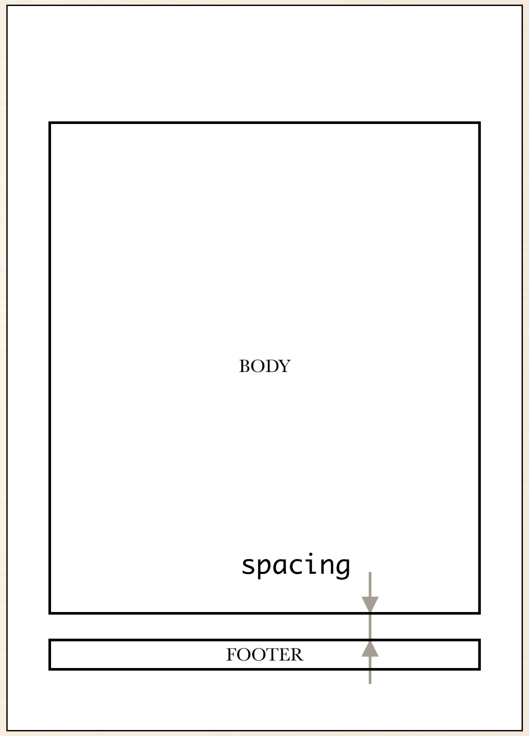


contents of footer have unknown height
DownExpandingContainer: initially 0 height, grows



Page Layout: Header & Footer

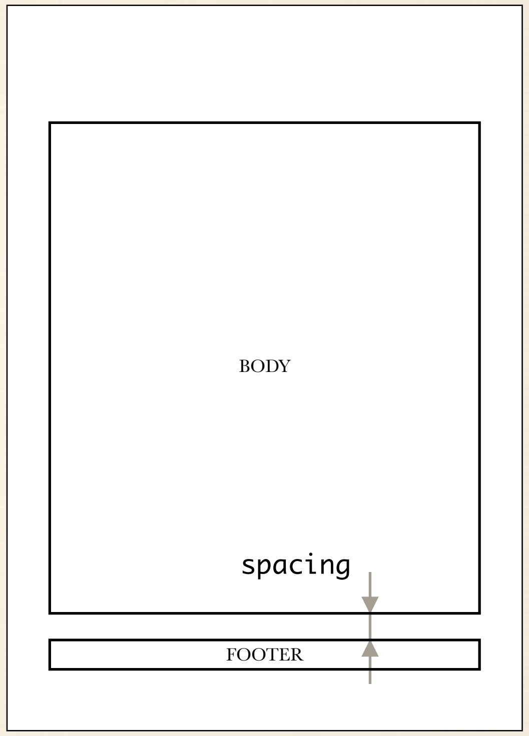
```
from rinoh import DownExpandingContainer
from rinoh import UpExpandingContainer
from rinoh import PT


spacing = 14*PT
```



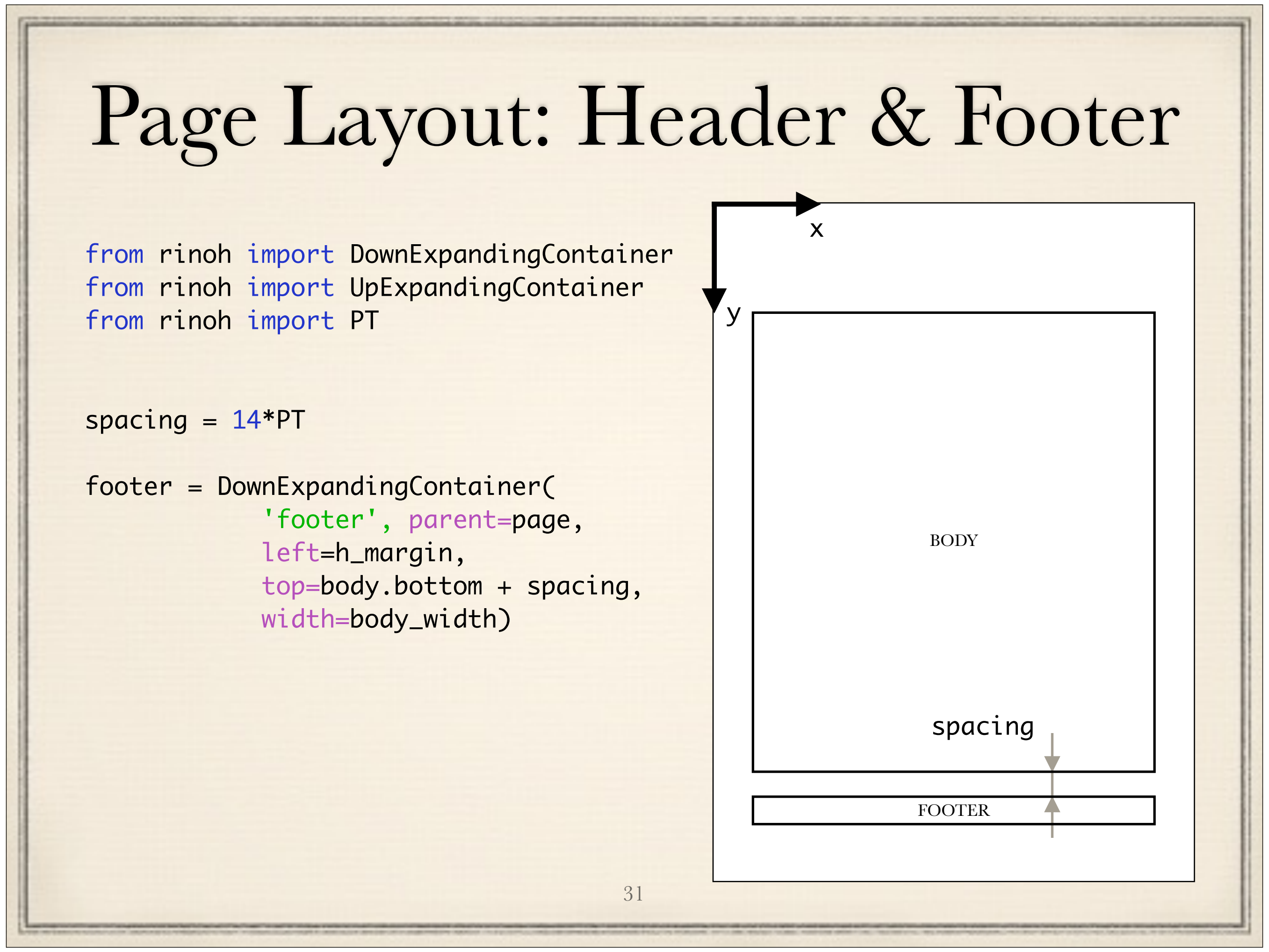
contents of footer have unknown height
DownExpandingContainer: initially 0 height, grows

Page Layout: Header & Footer

```
from rinoh import DownExpandingContainer
from rinoh import UpExpandingContainer
from rinoh import PT


spacing = 14*PT

footer = DownExpandingContainer(
            'footer', parent=page,
            left=h_margin,
            top=body.bottom + spacing,
            width=body_width)
```

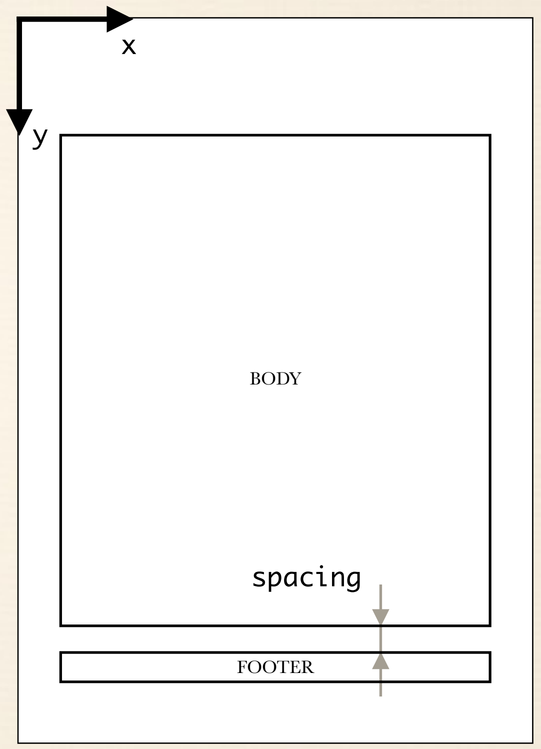


contents of footer have unknown height
DownExpandingContainer: initially 0 height, grows

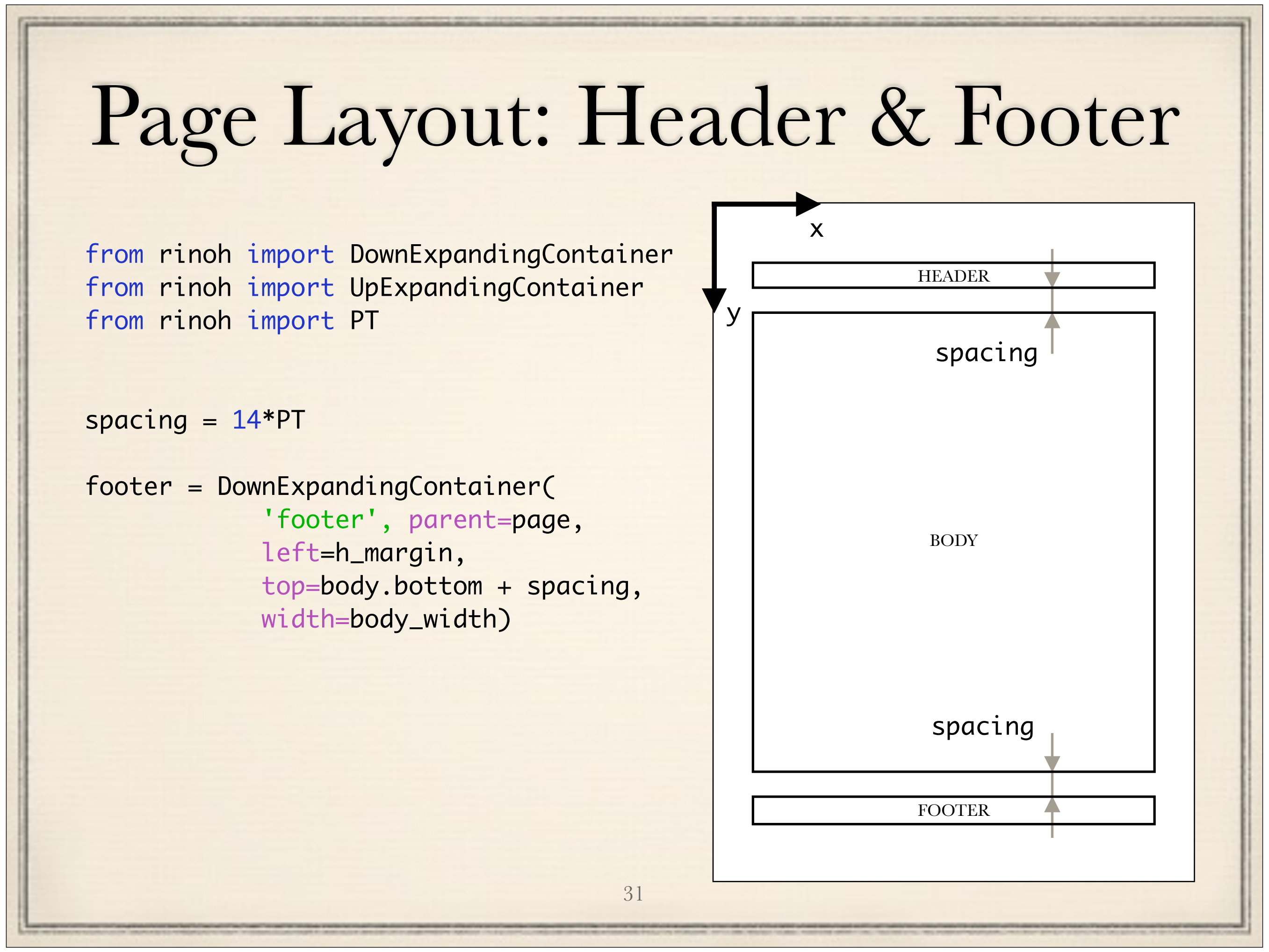


Page Layout: Header & Footer

```
from rinoh import DownExpandingContainer
from rinoh import UpExpandingContainer
from rinoh import PT


spacing = 14*PT

footer = DownExpandingContainer(
            'footer', parent=page,
            left=h_margin,
            top=body.bottom + spacing,
            width=body_width)
```

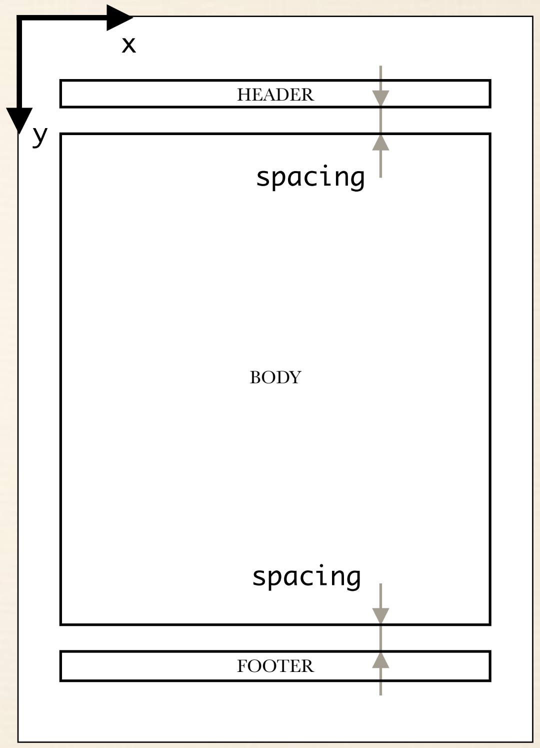


contents of footer have unknown height
DownExpandingContainer: initially 0 height, grows

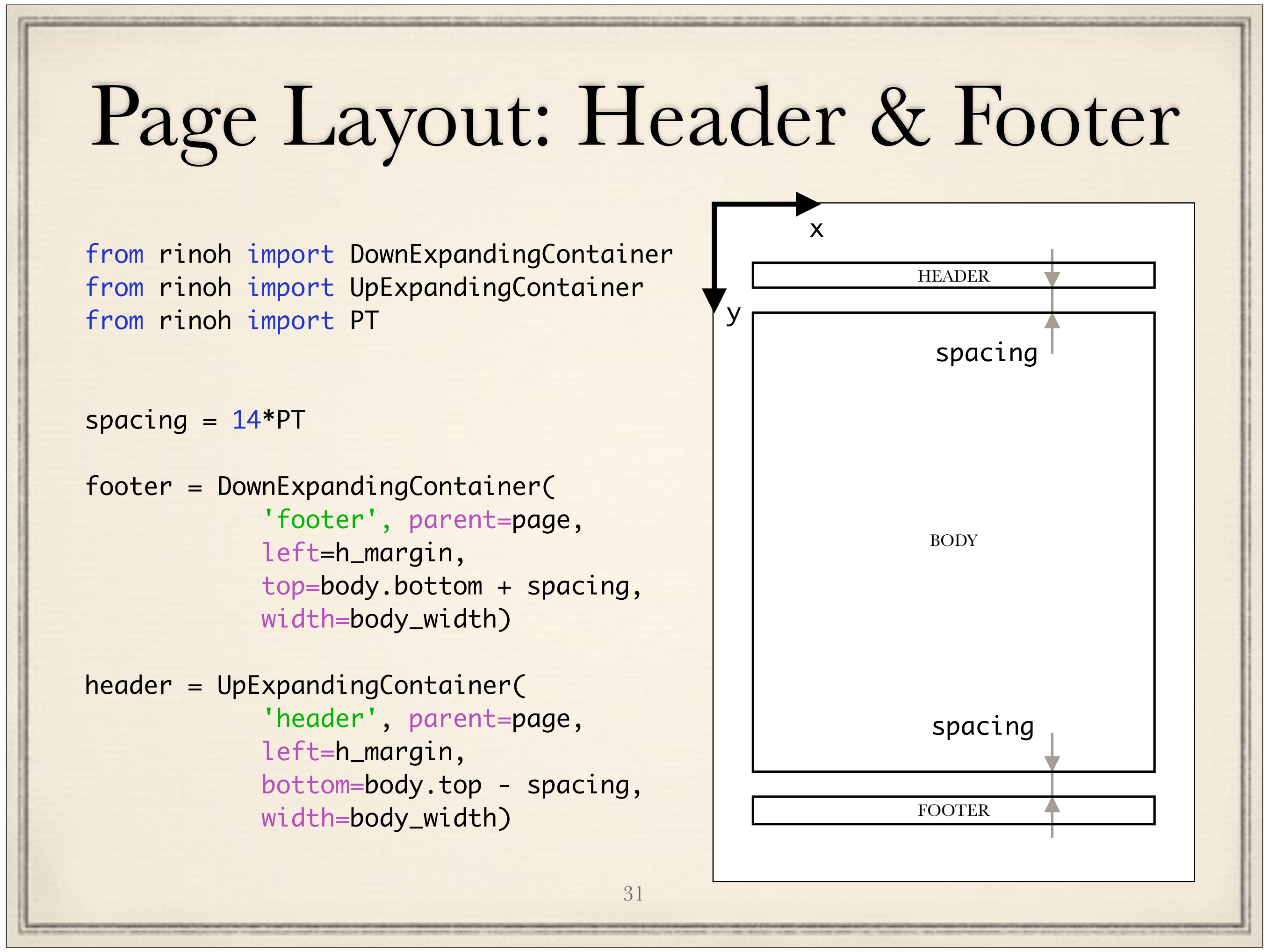


Page Layout: Header & Footer

```
from rinoh import DownExpandingContainer
from rinoh import UpExpandingContainer
from rinoh import PT


spacing = 14*PT

footer = DownExpandingContainer(
            'footer', parent=page,
            left=h_margin,
            top=body.bottom + spacing,
            width=body_width)
```

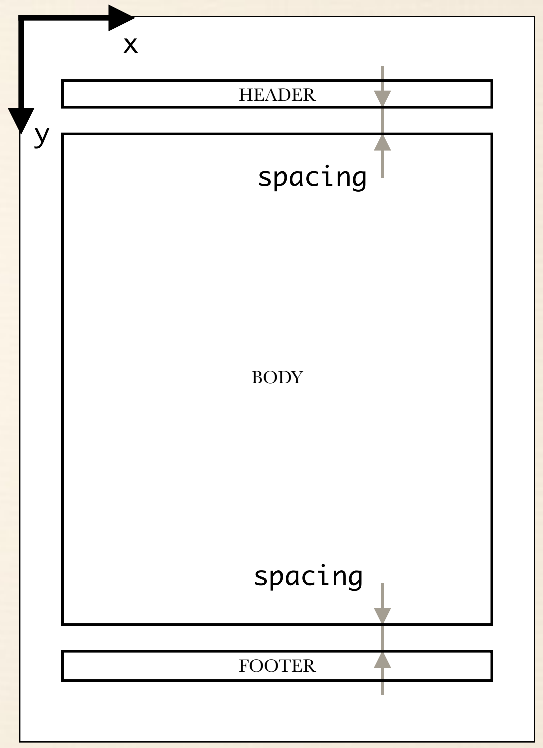


contents of footer have unknown height
DownExpandingContainer: initially 0 height, grows



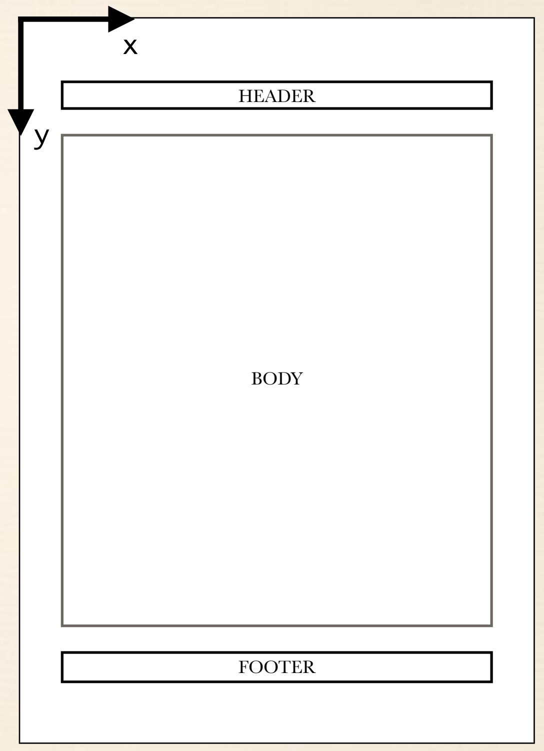
Page Layout: Header & Footer

```
from rinoh import DownExpandingContainer
from rinoh import UpExpandingContainer
from rinoh import PT


spacing = 14*PT

footer = DownExpandingContainer(
             'footer', parent=page,
             left=h_margin,
             top=body.bottom + spacing,
             width=body_width)

header = UpExpandingContainer(
             'header', parent=page,
             left=h_margin,
             bottom=body.top - spacing,
             width=body_width)
```



contents of footer have unknown height
DownExpandingContainer: initially 0 height, grows

Page Layout: Footnote Area

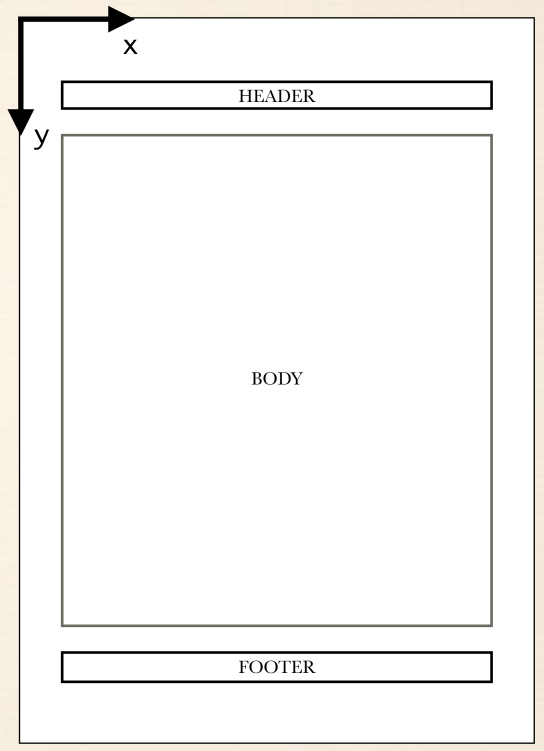


cannot use body.bottom!
footnotes are added, footnote area grows
as footnote area grows, text area should shrink

float area ~ footnote area (top and/or bottom)

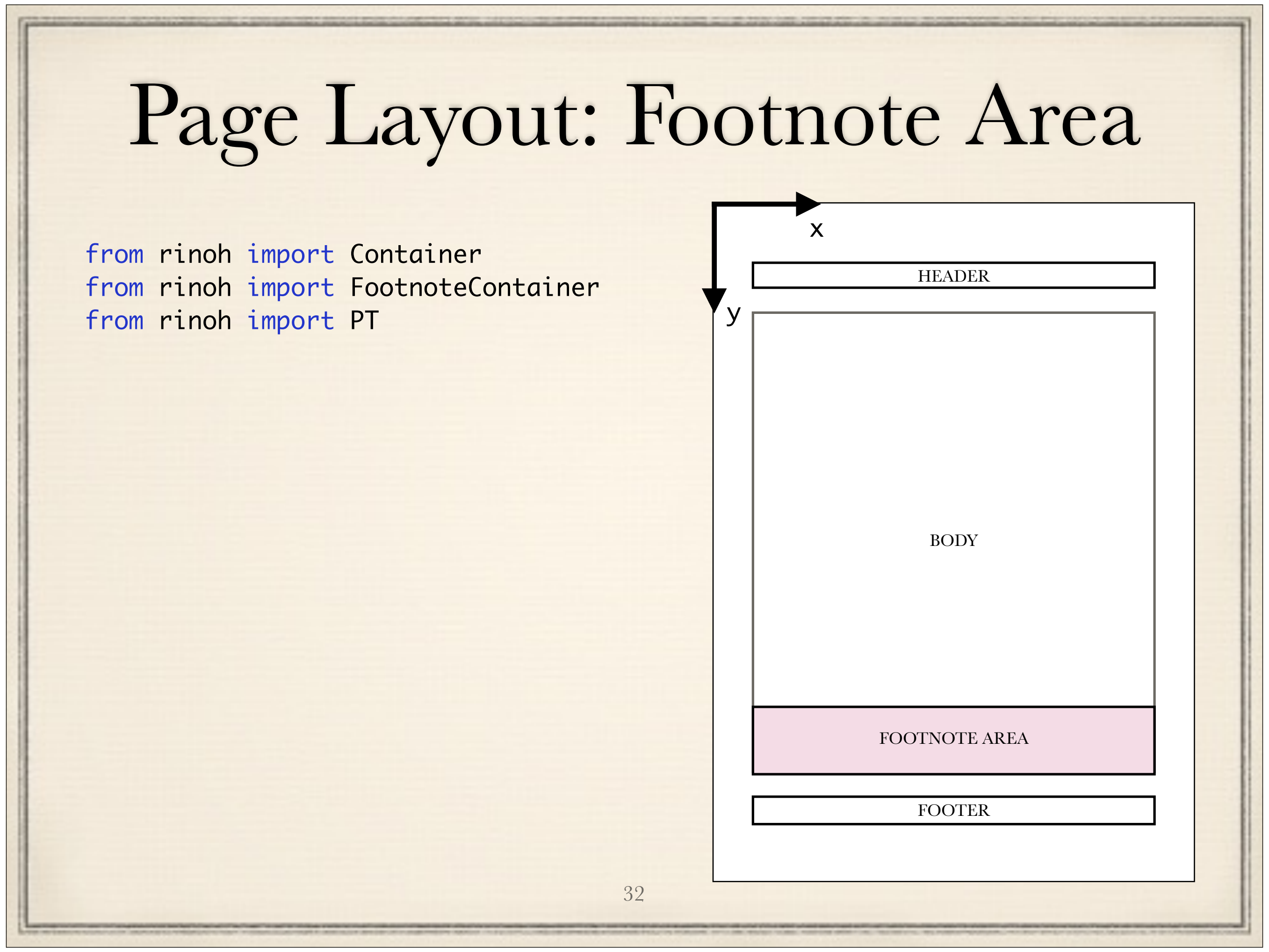
Page Layout: Footnote Area

```python
from rinoh import Container
from rinoh import FootnoteContainer
from rinoh import PT
```



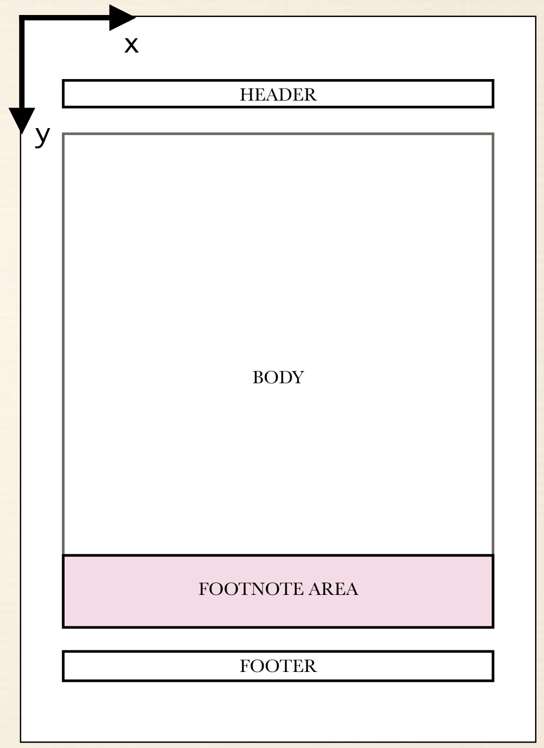
cannot use body.bottom!
footnotes are added, footnote area grows
as footnote area grows, text area should shrink

float area ~ footnote area (top and/or bottom)



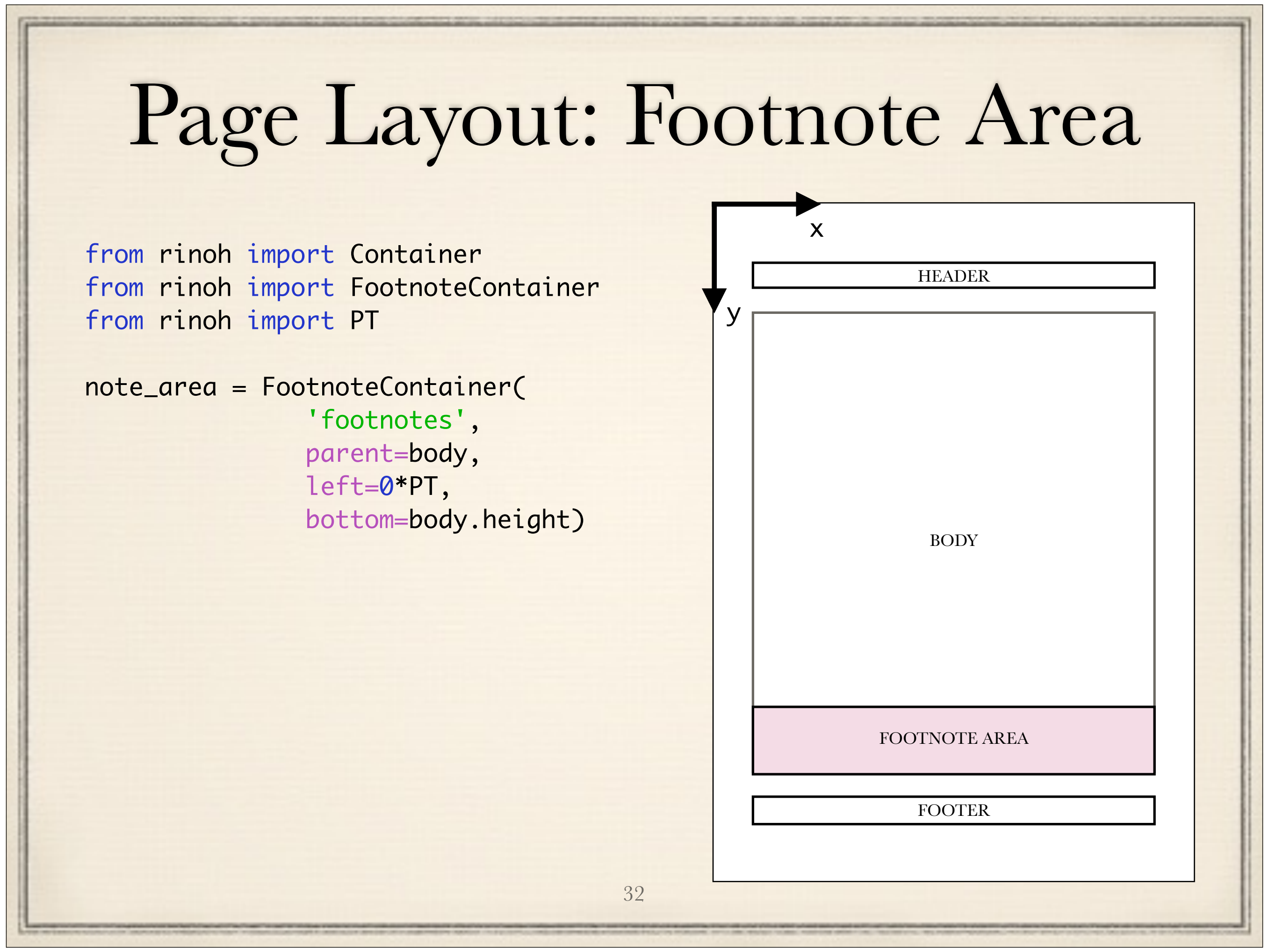
Page Layout: Footnote Area

```
from rinoh import Container
from rinoh import FootnoteContainer
from rinoh import PT
```



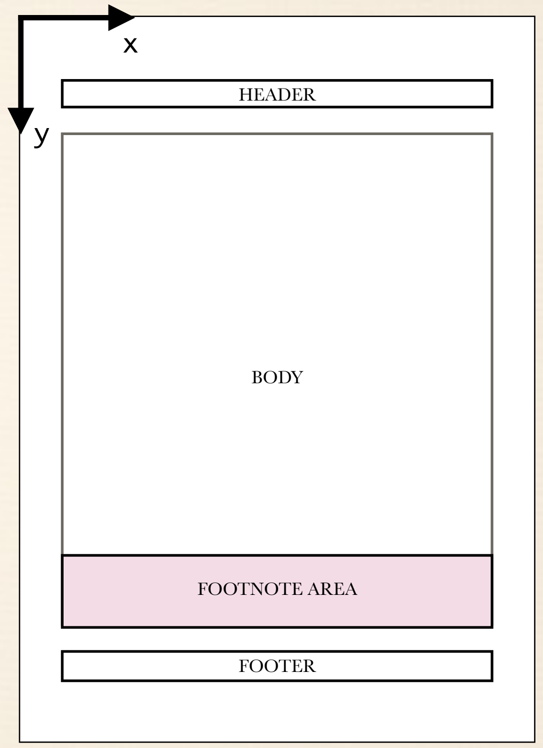
cannot use body.bottom!
footnotes are added, footnote area grows
as footnote area grows, text area should shrink

float area ~ footnote area (top and/or bottom)



Page Layout: Footnote Area

```
from rinoh import Container
from rinoh import FootnoteContainer
from rinoh import PT

note_area = FootnoteContainer(
                 'footnotes',
                 parent=body,
                 left=0*PT,
                 bottom=body.height)
```

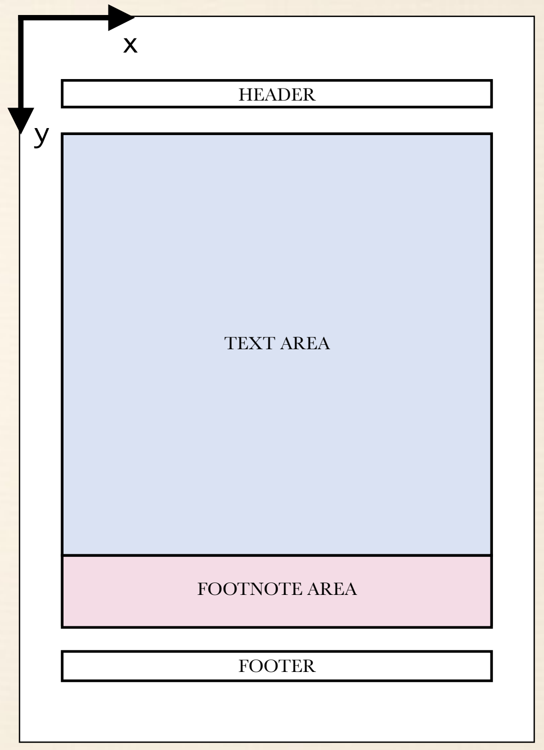


cannot use body.bottom!
footnotes are added, footnote area grows
as footnote area grows, text area should shrink

float area ~ footnote area (top and/or bottom)

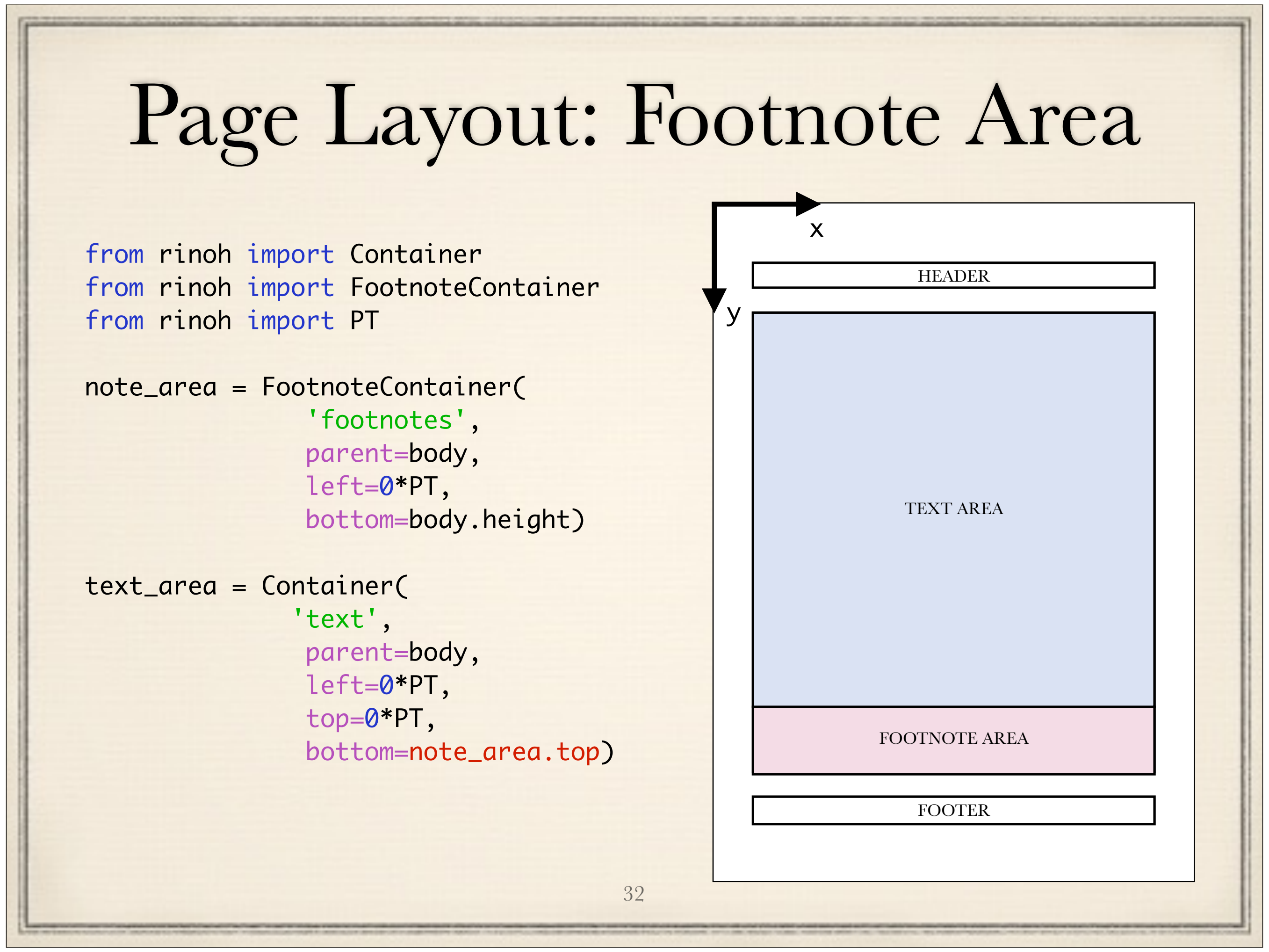
Page Layout: Footnote Area

```
from rinoh import Container
from rinoh import FootnoteContainer
from rinoh import PT

note_area = FootnoteContainer(
                'footnotes',
                parent=body,
                left=0*PT,
                bottom=body.height)
```



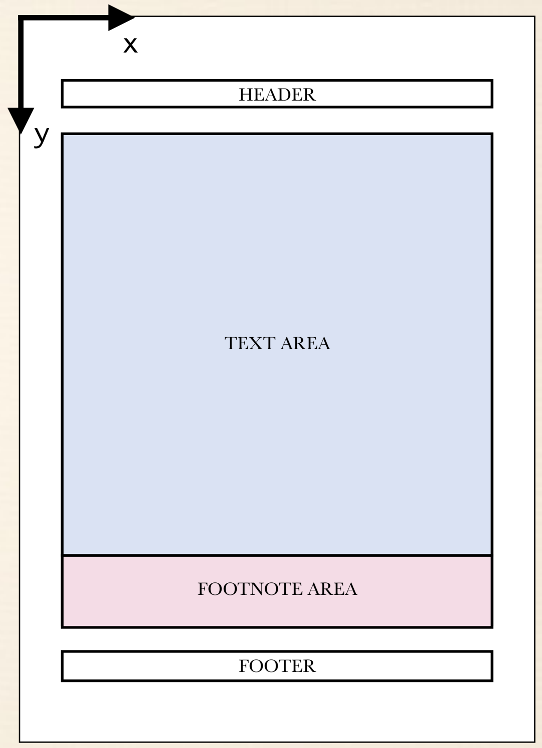
cannot use body.bottom!
footnotes are added, footnote area grows
as footnote area grows, text area should shrink

float area ~ footnote area (top and/or bottom)



Page Layout: Footnote Area

```
from rinoh import Container
from rinoh import FootnoteContainer
from rinoh import PT

note_area = FootnoteContainer(
                'footnotes',
                parent=body,
                left=0*PT,
                bottom=body.height)

text_area = Container(
               'text',
                parent=body,
                left=0*PT,
                top=0*PT,
                bottom=note_area.top)
```



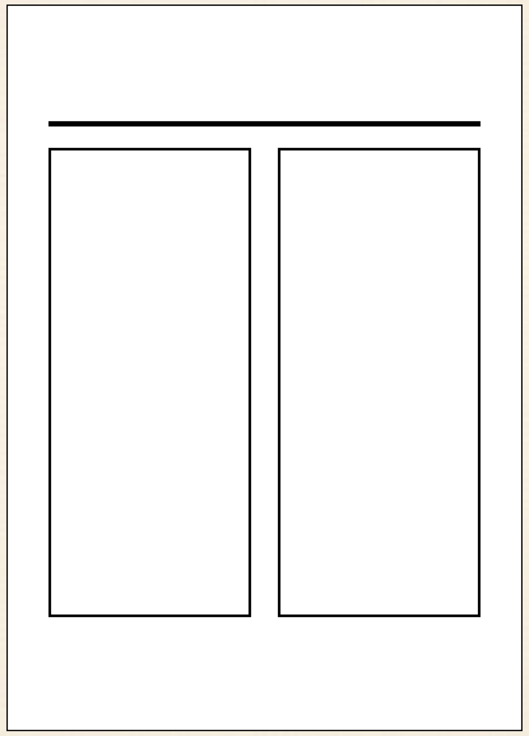
cannot use body.bottom!
footnotes are added, footnote area grows
as footnote area grows, text area should shrink

float area ~ footnote area (top and/or bottom)

Page Layout: Chains

- render contents directly into a container, or
- to a **chain** = list of containers
  - container full ➞ move to next container
  - no more containers ➞ new page

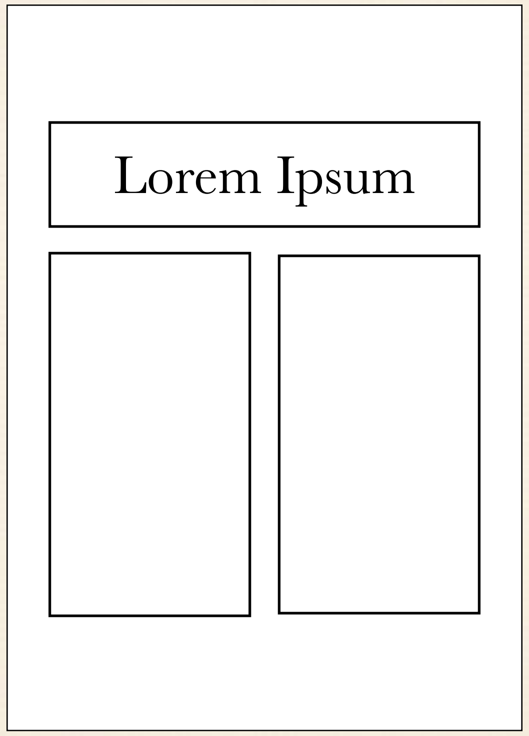
Page Layout: Chains



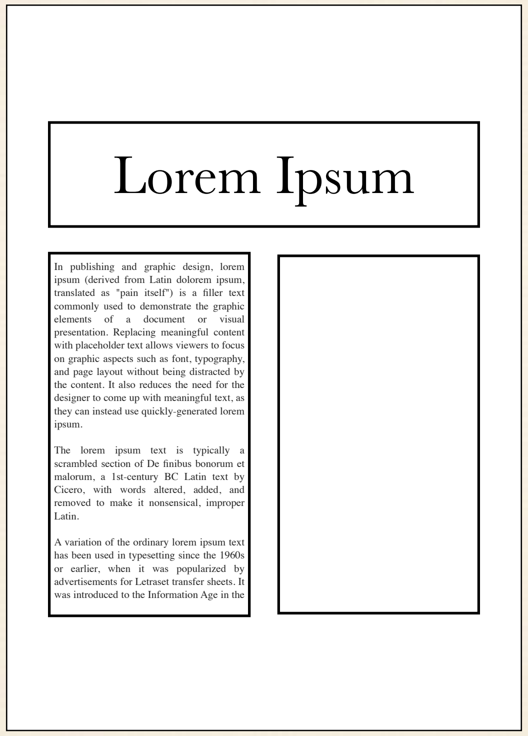
title container = DownExpandingContainer

column containers are placed relative to title container’s bottom attrib

Page Layout: Chains



Page Layout: Chains



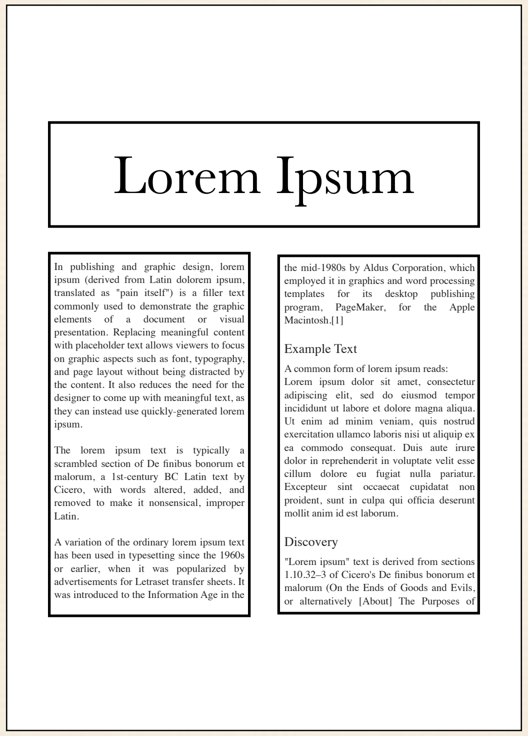
Lorem Ipsum

In publishing and graphic design, lorem ipsum (derived from Latin dolorem ipsum, translated as "pain itself") is a filler text commonly used to demonstrate the graphic elements of a document or visual presentation. Replacing meaningful content with placeholder text allows viewers to focus on graphic aspects such as font, typography, and page layout without being distracted by the content. It also reduces the need for the designer to come up with meaningful text, as they can instead use quickly-generated lorem ipsum.

The lorem ipsum text is typically a scrambled section of De finibus bonorum et malorum, a 1st-century BC Latin text by Cicero, with words altered, added, and removed to make it nonsensical, improper Latin.

A variation of the ordinary lorem ipsum text has been used in typesetting since the 1960s or earlier, when it was popularized by advertisements for Letraset transfer sheets. It was introduced to the Information Age in the

Page Layout: Chains



Lorem Ipsum

In publishing and graphic design, lorem ipsum (derived from Latin dolorem ipsum, translated as "pain itself") is a filler text commonly used to demonstrate the graphic elements of a document or visual presentation. Replacing meaningful content with placeholder text allows viewers to focus on graphic aspects such as font, typography, and page layout without being distracted by the content. It also reduces the need for the designer to come up with meaningful text, as they can instead use quickly-generated lorem ipsum.

The lorem ipsum text is typically a scrambled section of De finibus bonorum et malorum, a 1st-century BC Latin text by Cicero, with words altered, added, and removed to make it nonsensical, improper Latin.

A variation of the ordinary lorem ipsum text has been used in typesetting since the 1960s or earlier, when it was popularized by advertisements for Letraset transfer sheets. It was introduced to the Information Age in the mid-1980s by Aldus Corporation, which employed it in graphics and word processing templates for its desktop publishing program, PageMaker, for the Apple Macintosh.[1]

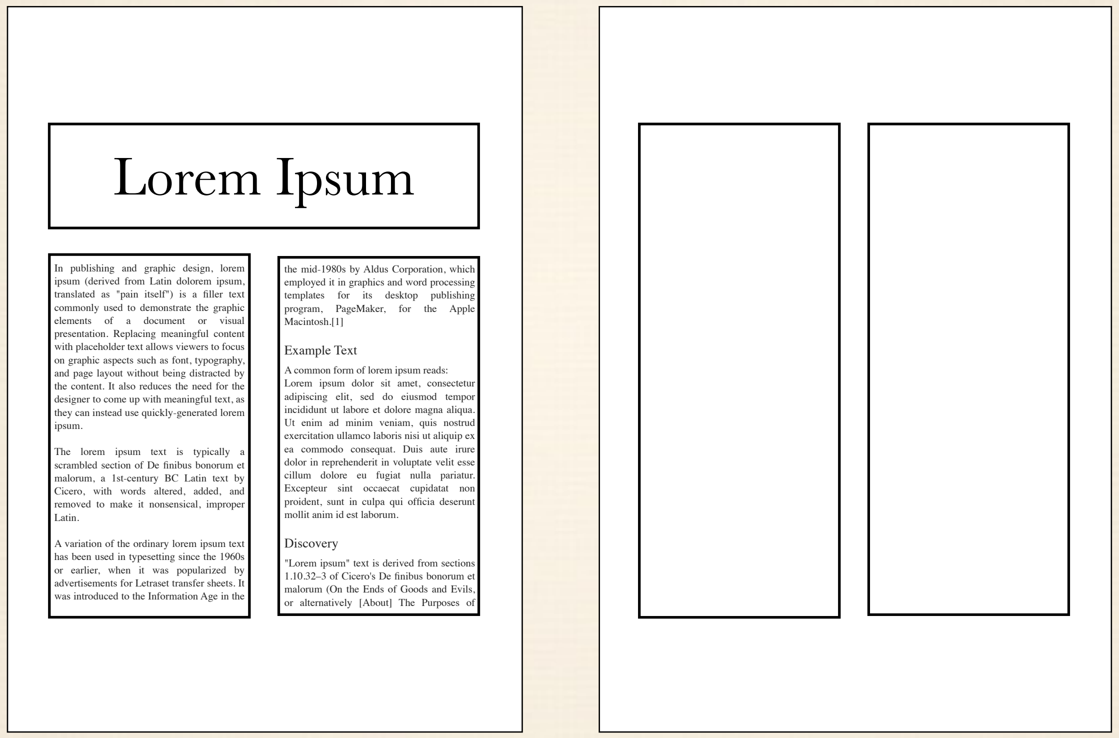
Example Text

A common form of lorem ipsum reads:
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Discovery

"Lorem ipsum" text is derived from sections 1.10.32–3 of Cicero's De finibus bonorum et malorum (On the Ends of Goods and Evils, or alternatively [About] The Purposes of

Page Layout: Chains



Lorem Ipsum

In publishing and graphic design, lorem ipsum (derived from Latin dolorem ipsum, translated as "pain itself") is a filler text commonly used to demonstrate the graphic elements of a document or visual presentation. Replacing meaningful content with placeholder text allows viewers to focus on graphic aspects such as font, typography, and page layout without being distracted by the content. It also reduces the need for the designer to come up with meaningful text, as they can instead use quickly-generated lorem ipsum.

The lorem ipsum text is typically a scrambled section of De finibus bonorum et malorum, a 1st-century BC Latin text by Cicero, with words altered, added, and removed to make it nonsensical, improper Latin.

A variation of the ordinary lorem ipsum text has been used in typesetting since the 1960s or earlier, when it was popularized by advertisements for Letraset transfer sheets. It was introduced to the Information Age in the mid-1980s by Aldus Corporation, which employed it in graphics and word processing templates for its desktop publishing program, PageMaker, for the Apple Macintosh.[1]

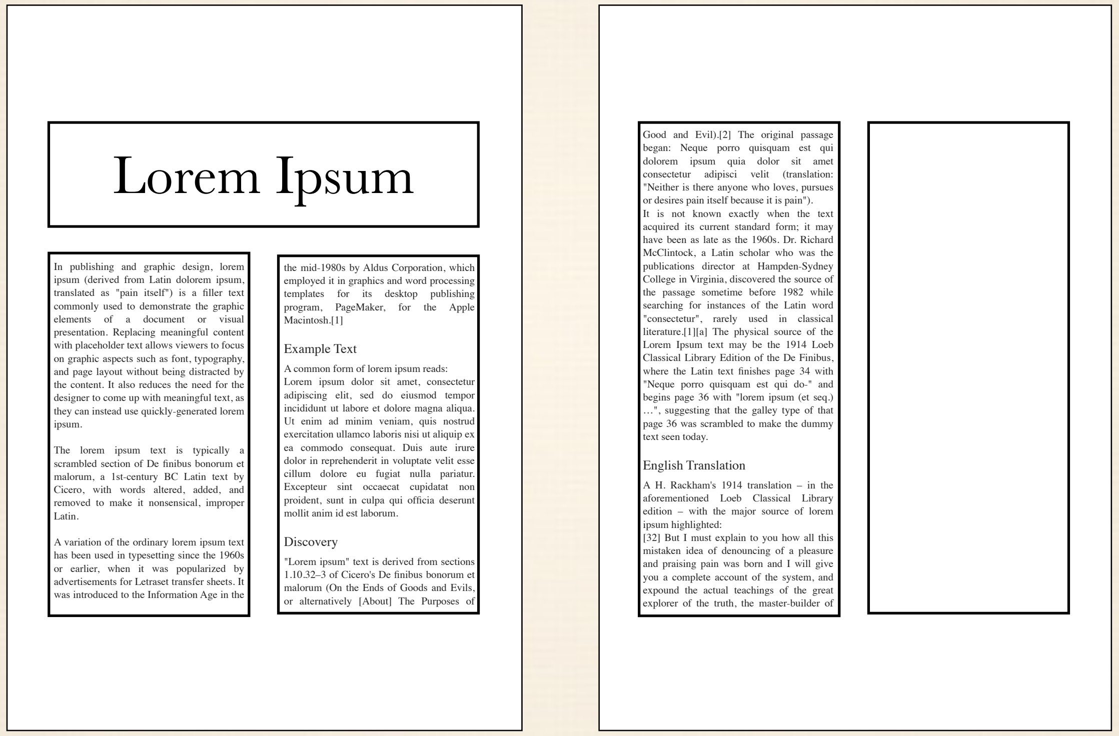
Example Text

A common form of lorem ipsum reads:
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Discovery

"Lorem ipsum" text is derived from sections 1.10.32–3 of Cicero's De finibus bonorum et malorum (On the Ends of Goods and Evils, or alternatively [About] The Purposes of

Page Layout: Chains



Lorem Ipsum

In publishing and graphic design, lorem ipsum (derived from Latin dolorem ipsum, translated as "pain itself") is a filler text commonly used to demonstrate the graphic elements of a document or visual presentation. Replacing meaningful content with placeholder text allows viewers to focus on graphic aspects such as font, typography, and page layout without being distracted by the content. It also reduces the need for the designer to come up with meaningful text, as they can instead use quickly-generated lorem ipsum.

The lorem ipsum text is typically a scrambled section of De finibus bonorum et malorum, a 1st-century BC Latin text by Cicero, with words altered, added, and removed to make it nonsensical, improper Latin.

A variation of the ordinary lorem ipsum text has been used in typesetting since the 1960s or earlier, when it was popularized by advertisements for Letraset transfer sheets. It was introduced to the Information Age in the mid-1980s by Aldus Corporation, which employed it in graphics and word processing templates for its desktop publishing program, PageMaker, for the Apple Macintosh.[1]

Example Text

A common form of lorem ipsum reads:
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

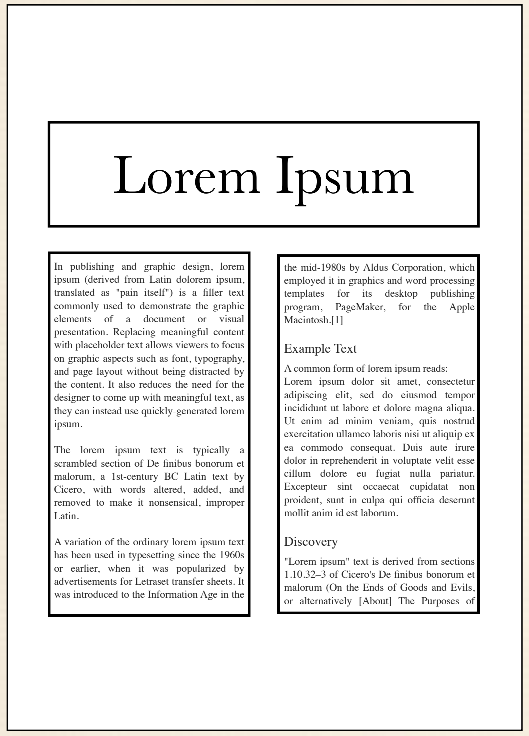
Discovery

"Lorem ipsum" text is derived from sections 1.10.32–3 of Cicero's De finibus bonorum et malorum (On the Ends of Goods and Evils, or alternatively [About] The Purposes of Good and Evil).[2] The original passage began: Neque porro quisquam est qui dolorem ipsum quia dolor sit amet consectetur adipisci velit (translation: "Neither is there anyone who loves, pursues or desires pain itself because it is pain").
It is not known exactly when the text acquired its current standard form; it may have been as late as the 1960s. Dr. Richard McClintock, a Latin scholar who was the publications director at Hampden-Sydney College in Virginia, discovered the source of the passage sometime before 1982 while searching for instances of the Latin word "consectetur", rarely used in classical literature.[1][a] The physical source of the Lorem Ipsum text may be the 1914 Loeb Classical Library Edition of the De Finibus, where the Latin text finishes page 34 with "Neque porro quisquam est qui do-" and begins page 36 with "lorem ipsum (et seq.) …", suggesting that the galley type of that page 36 was scrambled to make the dummy text seen today.

English Translation

A H. Rackham's 1914 translation – in the aforementioned Loeb Classical Library edition – with the major source of lorem ipsum highlighted:
[32] But I must explain to you how all this mistaken idea of denouncing of a pleasure and praising pain was born and I will give you a complete account of the system, and expound the actual teachings of the great explorer of the truth, the master-builder of

Page Layout: Chains


Lorem Ipsum

In publishing and graphic design, lorem ipsum (derived from Latin dolorem ipsum, translated as "pain itself") is a filler text commonly used to demonstrate the graphic elements of a document or visual presentation. Replacing meaningful content with placeholder text allows viewers to focus on graphic aspects such as font, typography, and page layout without being distracted by the content. It also reduces the need for the designer to come up with meaningful text, as they can instead use quickly-generated lorem ipsum.

The lorem ipsum text is typically a scrambled section of De finibus bonorum et malorum, a 1st-century BC Latin text by Cicero, with words altered, added, and removed to make it nonsensical, improper Latin.

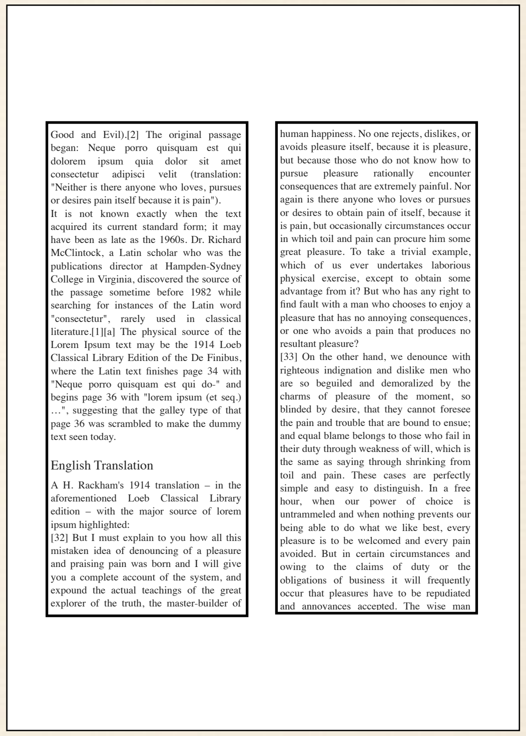
A variation of the ordinary lorem ipsum text has been used in typesetting since the 1960s or earlier, when it was popularized by advertisements for Letraset transfer sheets. It was introduced to the Information Age in the mid-1980s by Aldus Corporation, which employed it in graphics and word processing templates for its desktop publishing program, PageMaker, for the Apple Macintosh.[1]

Example Text

A common form of lorem ipsum reads:
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

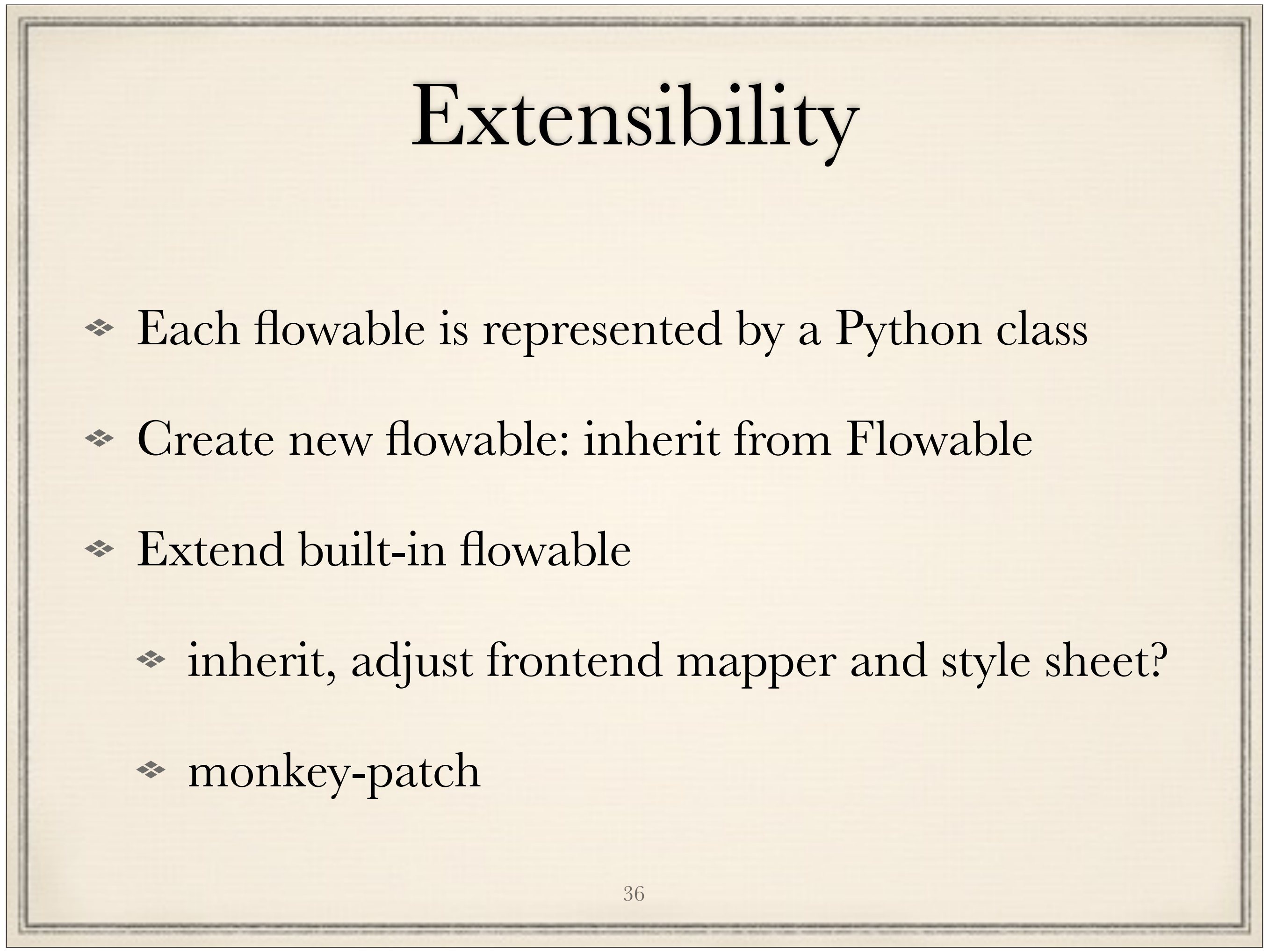
Discovery

"Lorem ipsum" text is derived from sections 1.10.32–3 of Cicero's De finibus bonorum et malorum (On the Ends of Goods and Evils, or alternatively [About] The Purposes of


Good and Evil).[2] The original passage began: Neque porro quisquam est qui dolorem ipsum quia dolor sit amet consectetur adipisci velit (translation: "Neither is there anyone who loves, pursues or desires pain itself because it is pain").
It is not known exactly when the text acquired its current standard form; it may have been as late as the 1960s. Dr. Richard McClintock, a Latin scholar who was the publications director at Hampden-Sydney College in Virginia, discovered the source of the passage sometime before 1982 while searching for instances of the Latin word "consectetur", rarely used in classical literature.[1][a] The physical source of the Lorem Ipsum text may be the 1914 Loeb Classical Library Edition of the De Finibus, where the Latin text finishes page 34 with "Neque porro quisquam est qui do-" and begins page 36 with "lorem ipsum (et seq.) …", suggesting that the galley type of that page 36 was scrambled to make the dummy text seen today.

English Translation

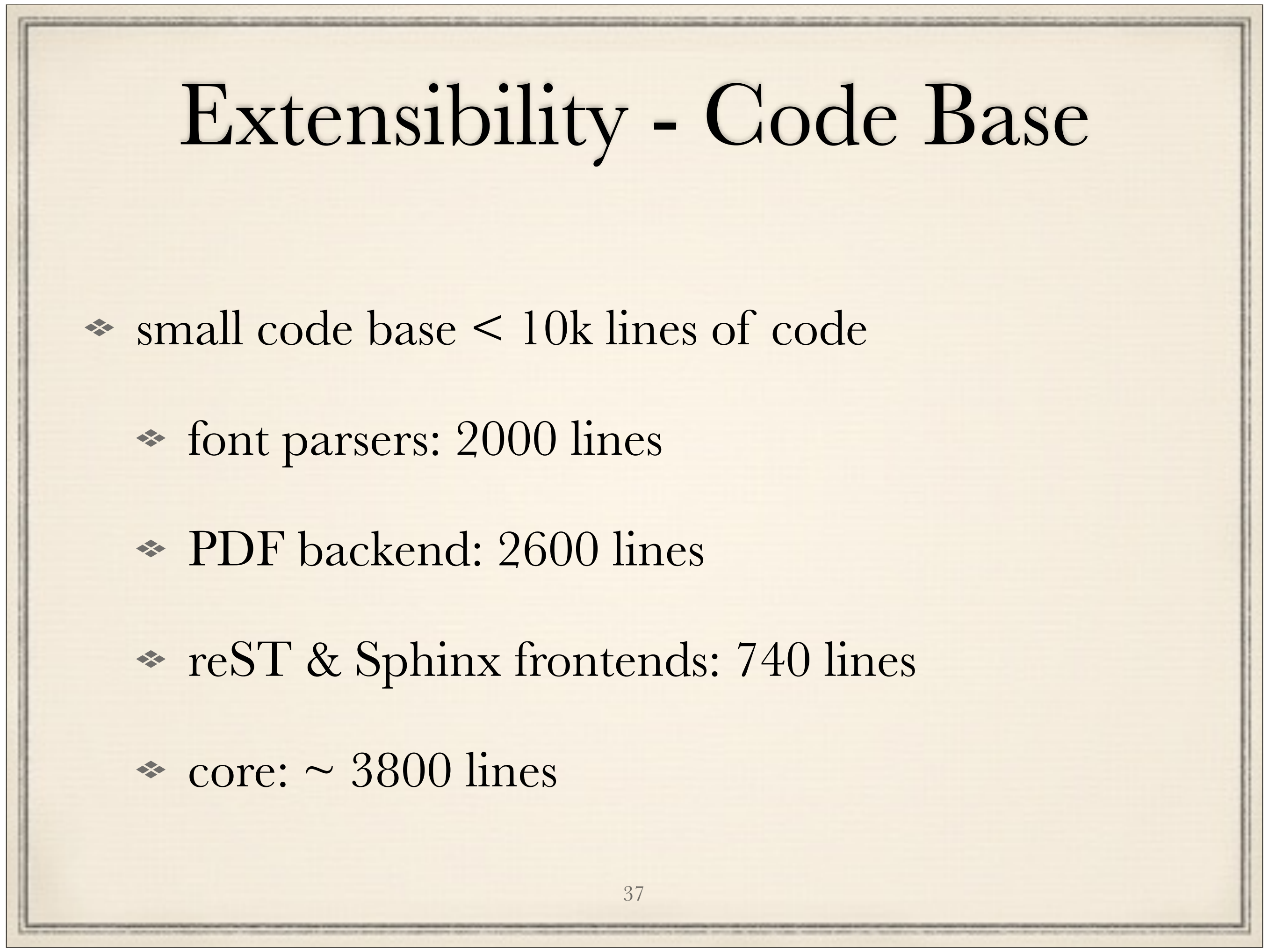
A H. Rackham's 1914 translation – in the aforementioned Loeb Classical Library edition – with the major source of lorem ipsum highlighted:
[32] But I must explain to you how all this mistaken idea of denouncing of a pleasure and praising pain was born and I will give you a complete account of the system, and expound the actual teachings of the great explorer of the truth, the master-builder of human happiness. No one rejects, dislikes, or avoids pleasure itself, because it is pleasure, but because those who do not know how to pursue pleasure rationally encounter consequences that are extremely painful. Nor again is there anyone who loves or pursues or desires to obtain pain of itself, because it is pain, but occasionally circumstances occur in which toil and pain can procure him some great pleasure. To take a trivial example, which of us ever undertakes laborious physical exercise, except to obtain some advantage from it? But who has any right to find fault with a man who chooses to enjoy a pleasure that has no annoying consequences, or one who avoids a pain that produces no resultant pleasure?
[33] On the other hand, we denounce with righteous indignation and dislike men who are so beguiled and demoralized by the charms of pleasure of the moment, so blinded by desire, that they cannot foresee the pain and trouble that are bound to ensue; and equal blame belongs to those who fail in their duty through weakness of will, which is the same as saying through shrinking from toil and pain. These cases are perfectly simple and easy to distinguish. In a free hour, when our power of choice is untrammeled and when nothing prevents our being able to do what we like best, every pleasure is to be welcomed and every pain avoided. But in certain circumstances and owing to the claims of duty or the obligations of business it will frequently occur that pleasures have to be repudiated and annoyances accepted. The wise man



Extensibility

- Each flowable is represented by a Python class
- Create new flowable: inherit from Flowable
- Extend built-in flowable
  - inherit, adjust frontend mapper and style sheet?
  - monkey-patch

extension API?



Extensibility - Code Base

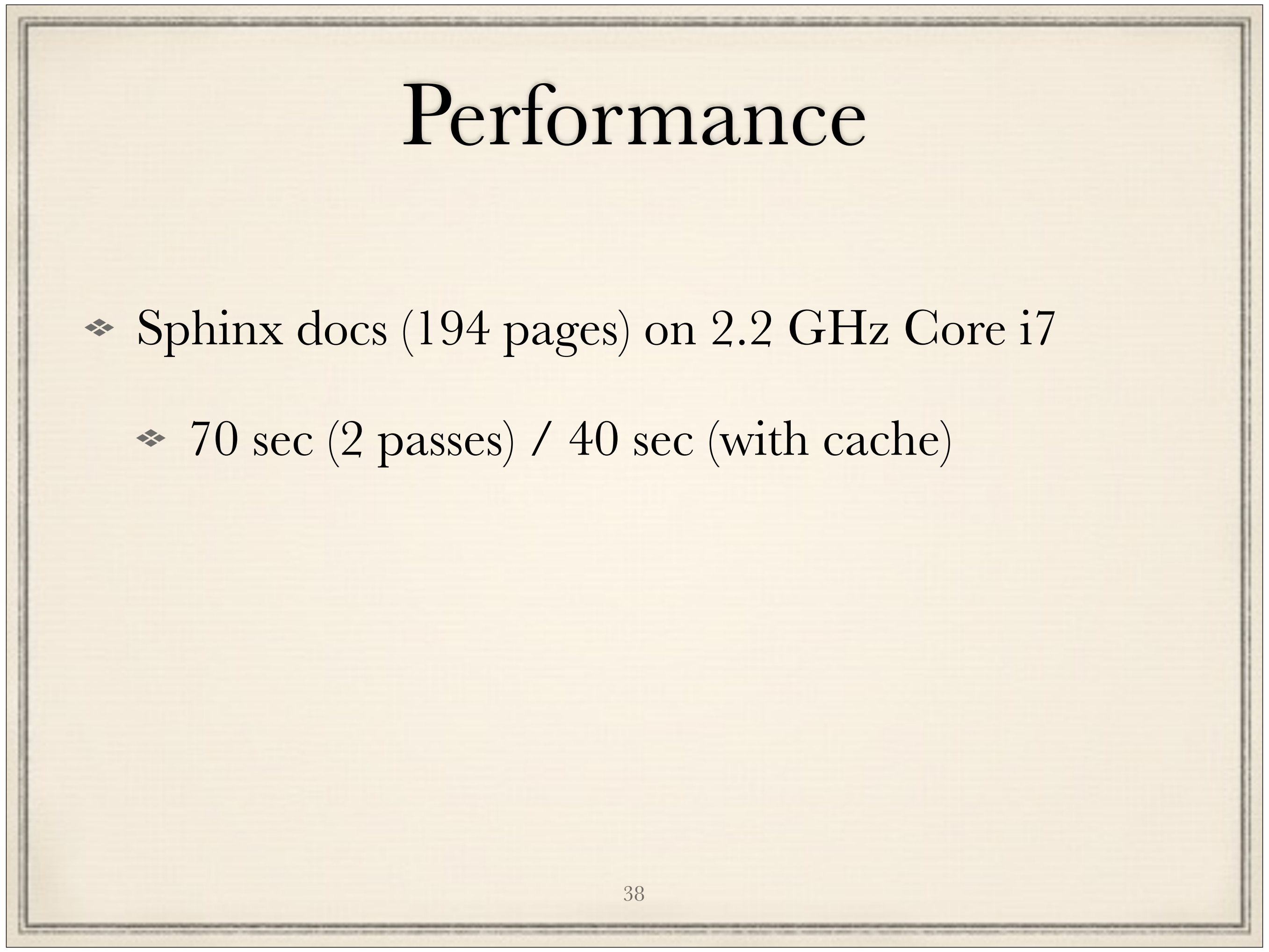
- small code base < 10k lines of code
  - font parsers: 2000 lines
  - PDF backend: 2600 lines
  - reST & Sphinx frontends: 740 lines
  - core: ~ 3800 lines

core is almost feature complete (except equations)
doesn’t have all the features offered by LaTeX extensions

Performance

Cython
- mostly container access instead of number crunching

PyPy
- not further investigated
- haven’t tried with newer releases



Performance

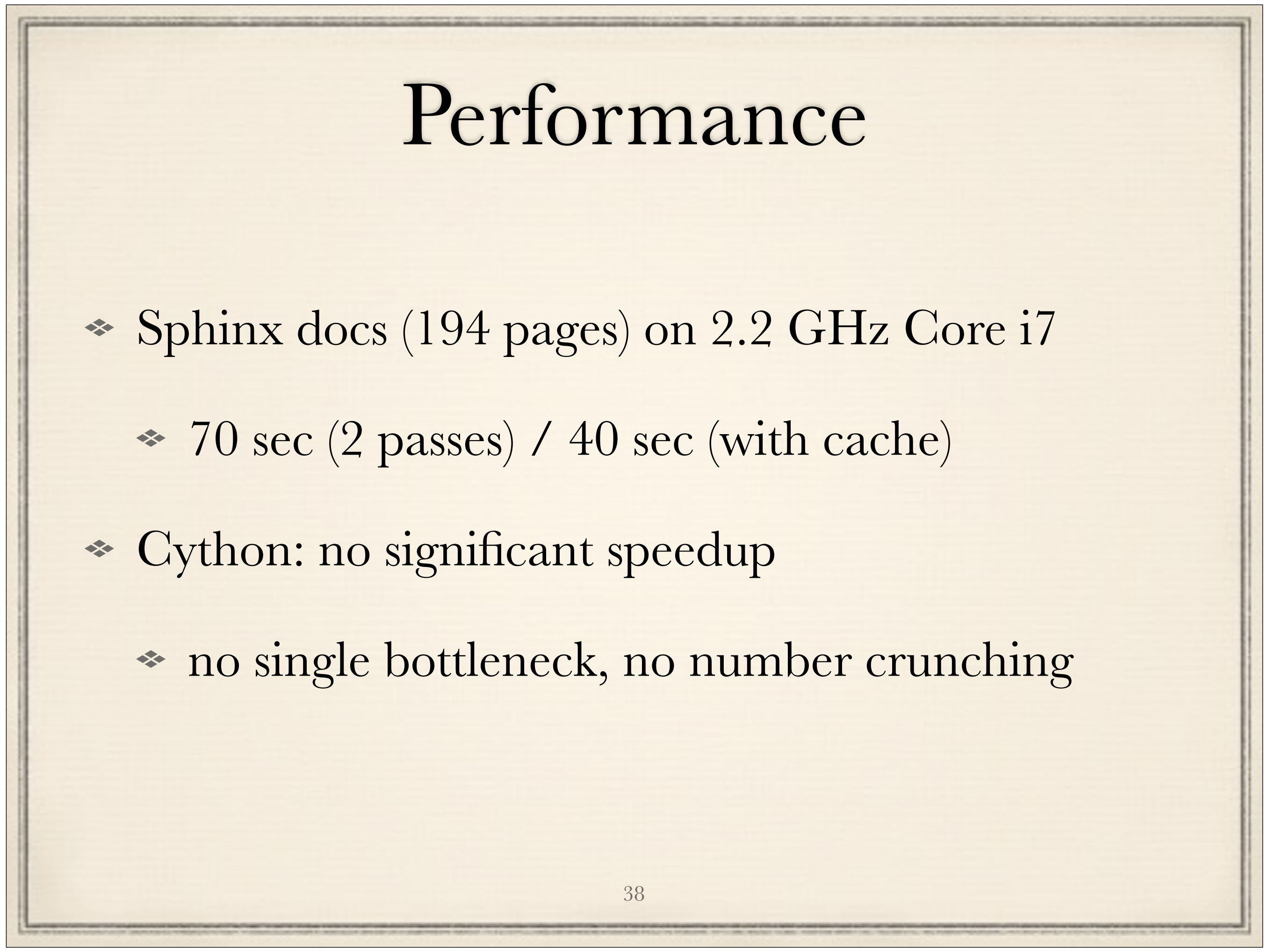
- Sphinx docs (194 pages) on 2.2 GHz Core i7
    - 70 sec (2 passes) / 40 sec (with cache)

Cython

- mostly container access instead of number crunching

PyPy

- not further investigated
- haven’t tried with newer releases

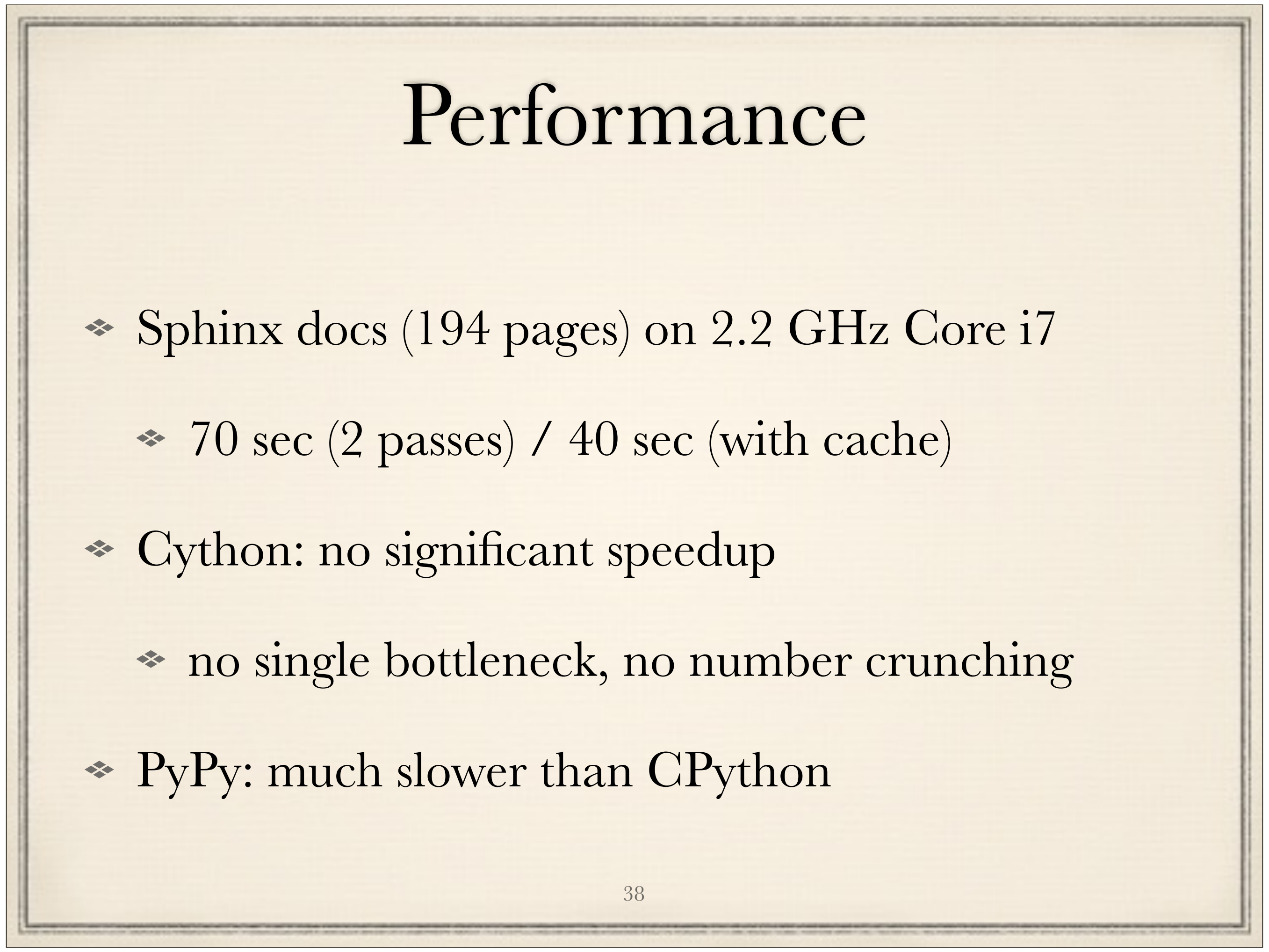


Performance

- Sphinx docs (194 pages) on 2.2 GHz Core i7
  - 70 sec (2 passes) / 40 sec (with cache)
- Cython: no significant speedup
  - no single bottleneck, no number crunching

Cython
- mostly container access instead of number crunching

PyPy
- not further investigated
- haven’t tried with newer releases

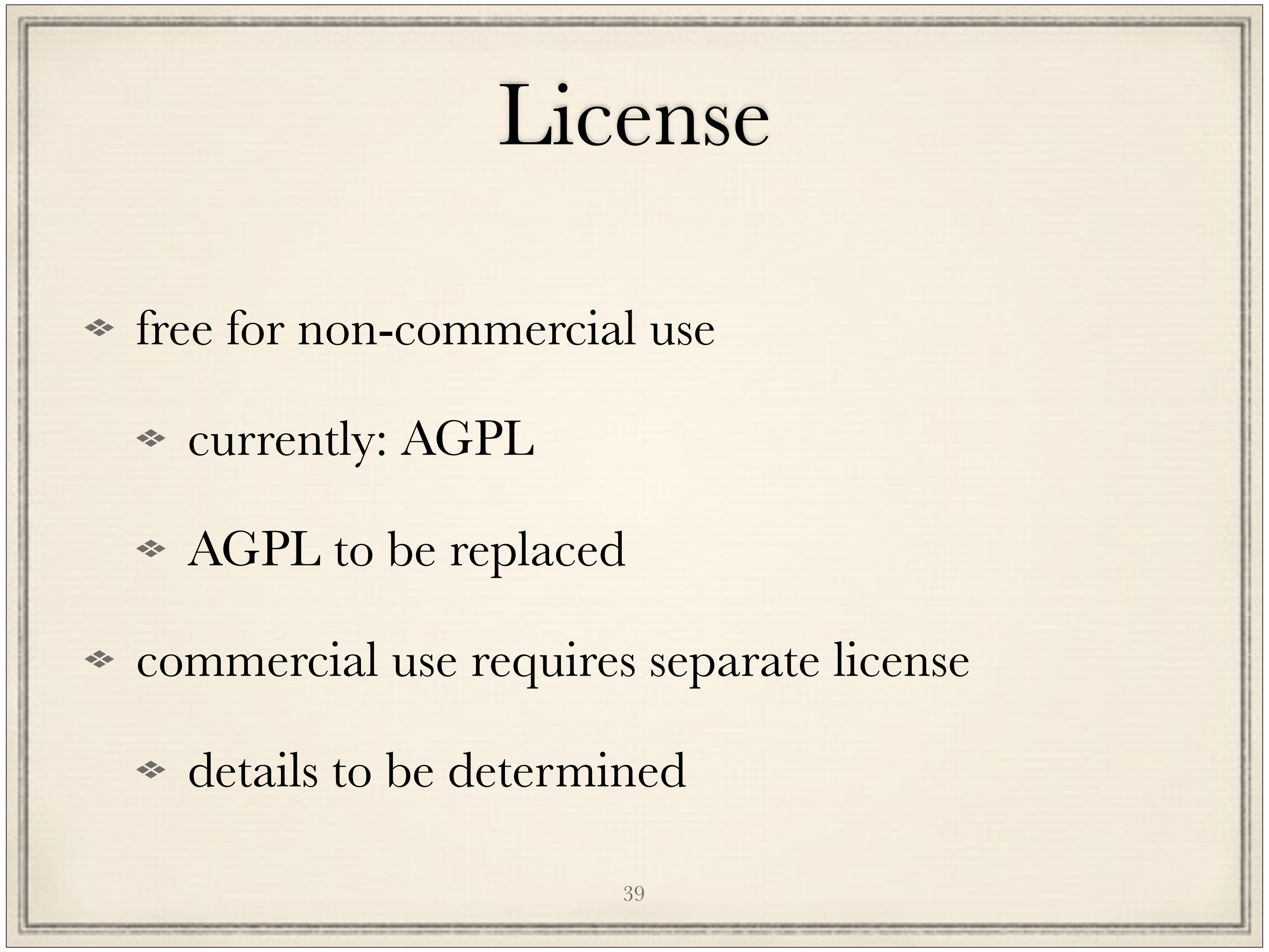


Performance

- Sphinx docs (194 pages) on 2.2 GHz Core i7
  - 70 sec (2 passes) / 40 sec (with cache)
- Cython: no significant speedup
  - no single bottleneck, no number crunching
- PyPy: much slower than CPython

Cython
- mostly container access instead of number crunching

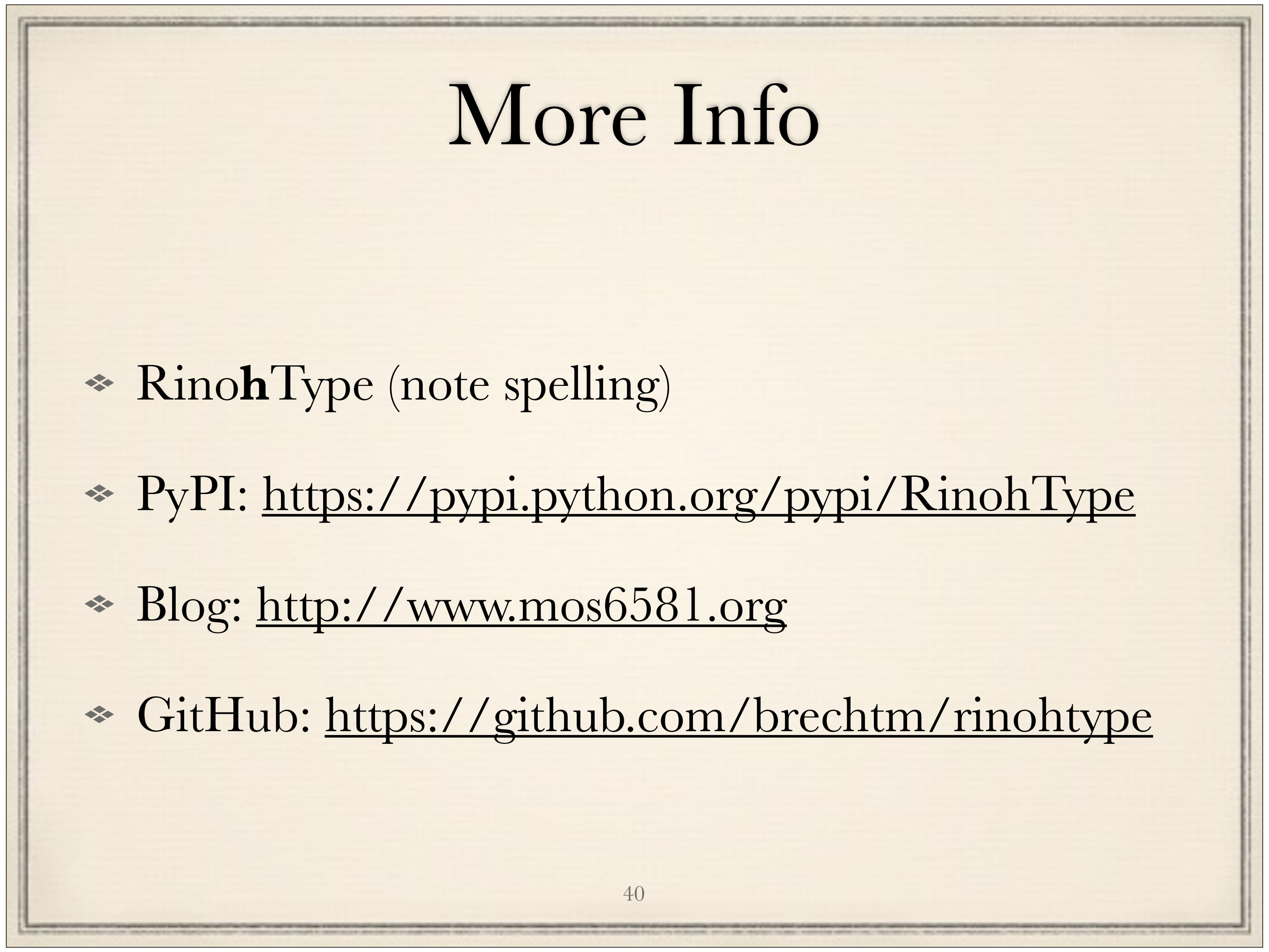
PyPy
- not further investigated
- haven’t tried with newer releases



License

- free for non-commercial use
  - currently: AGPL
  - AGPL to be replaced
- commercial use requires separate license
  - details to be determined

open-source projects can freely use RinohType for Sphinx PDF use
AGPL to be replaced, to make intentions more clear
complex matter



More Info

- Rino**h**Type (note spelling)
- PyPI: https://pypi.python.org/pypi/RinohType
- Blog: http://www.mos6581.org
- GitHub: https://github.com/brechtm/rinohtype

ideas, thoughts, suggestions -> talk to me

Questions?