A look at Parsoid internals

Subramanya Sastry (Subbu)
Gabriel Wicke

Wiki: http://www.mediawiki.org/wiki/Parsoid
Team: Arlo, Gabriel, Marc, Scott, Subbu
Alumnus: Mark Holmquist

April 15, 2014

Talk prepared using LaTeX, Beamer and InkScape

Outline

Introduction: What is Parsoid?

- Service that converts between wikitext and HTML5 + RDFa. Spec @ mediawiki.org/wiki/Parsoid/MediaWiki_DOM_spec
- Written in Javascript, running on node.js
- Provides a relatively simple API:

Introduction: What is Parsoid?

- Service that converts between wikitext and HTML5 + RDFa. Spec @ mediawiki.org/wiki/Parsoid/MediaWiki_DOM_spec
- Written in Javascript, running on node.js
- Provides a relatively simple API:

Wikitext to HTML

```
POST /enwiki/Main_Page
wt:  "foo"

<i>foo</i>
```

Introduction: What is Parsoid?

- Service that converts between wikitext and HTML5 + RDFa. Spec @ mediawiki.org/wiki/Parsoid/MediaWiki_DOM_spec
- Written in Javascript, running on node.js
- Provides a relatively simple API:

Wikitext to HTML

```
POST /enwiki/Main_Page
wt:  "foo"

<i>foo</i>
```

HTML to wikitext

```
POST /enwiki/Main_Page
HTML: <i>foo</i>

"foo"
```

Introduction: What is Parsoid?

- Service that converts between wikitext and HTML5 + RDFa. Spec @ mediawiki.org/wiki/Parsoid/MediaWiki_DOM_spec
- Written in Javascript, running on node.js
- Provides a relatively simple API:

  Fetch HTML for a page

  ```
  GET /enwiki/Main_Page

  <html ...> .. </html>
  ```

- Visit mediawiki.org/wiki/Parsoid#The_Parsoid_web_API

Introduction: What is Parsoid?

- Service that converts between wikitext and HTML5 + RDFa. Spec @ mediawiki.org/wiki/Parsoid/MediaWiki_DOM_spec
- Written in Javascript, running on node.js
- Provides a relatively simple API:
- Provides convenient command-line utilities

```
% node parse --wt2html < wikitext
% node parse --html2wt < html
% node parse --wt2wt < wikitext
% node parse --html2html < html
```

`% node parse --help` for more

Introduction: Who uses Parsoid?

- Visual Editor uses it both ways
- Flow uses it to support wikitext editing in html discussions.
- PDF rendering, Mobile, Kiwix use rendered html.
- Content translation uses it both ways to support translation between wikis.
- Gadgets, bots? ...
- Full list @ mediawiki.org/wiki/Parsoid/Users

Outline

What makes this challenging?

HTML should convert back to wikitext without "dirty diffs"

- `* foo` and `*foo` are different even though they map to the same HTML DOM.
- `<ref name='foo'>..</ref>`
  `<ref name = foo>..</ref>`
  `<REF name="foo">..</REF>`
- Requires Parsoid to serialize unmodified HTML to the exact same wikitext.

What makes this challenging?

Overloaded syntax makes for complex semantics

- Simple links:

  `[[Foo]], [[Foo|Foo]], [[Foo|bar]]`

- Link prefixes and suffixes/trails/tails:

  `[[Foo|bar]]s, Pre[[fix|Suf]]fix`

- Interwiki links:

  `[[fr:Interwiki]], [[wikt:fr:Blah]]`

- Templated links:

  `[[{{echo|Foo}}|{{echo|bar}}]]s`

- Images:

  `[[Image:Foo.jpg|caption]], [[Image:Foo.jpg|thumb|300px]],`
  `[[File.Foo.jpg|thumb|right|<table>..</table>]]`

- External links, URL links, Magicword links

  `[http://www.mediawiki.org MW], http://google.com,`
  `ISBN 0123456789, PMD something, RFC 1034`

What makes this challenging?

Wikitext templates are string-based: no DOM semantics

- Template output can have non-local effects on DOM structure.

  ```
  foo {{echo|<div>}} a lot of wikitext here </div>
  ```

- Requires all templates to be expanded first before DOM can be built.
- How do you edit this in a HTML editor like VisualEditor? Transclusion output cannot be mapped to any DOM node.

Some pages can have 100s of transclusions

- If expansions are done sequentially, parsing can be very inefficient.

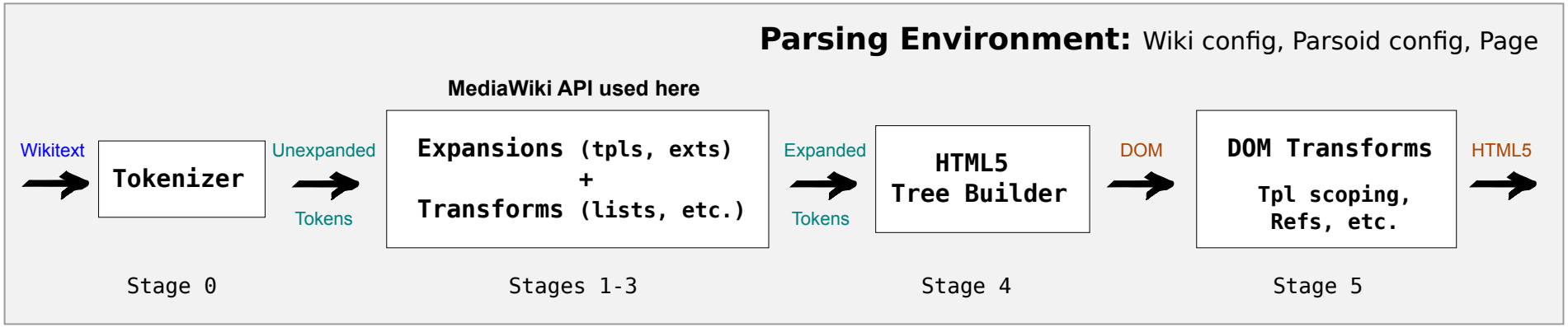
What makes this challenging?

There is no "invalid" wikitext

- `<div><small>foo</div>`
  `{|`
  `|- This text is dropped`
  `| 2.7183 || i || 3.1415 || 1`
  `|}`
- `<table>`
  `This text will move out of the table`
  `<tr><td>foo</td></table>`
- `<div title="foo'>Mismatched quotes</div>`
- `<i><b>overlapping</i>tags</b>`
- `[[Foo.jpg|thumb|caption 1|caption 2 will be lost]]`

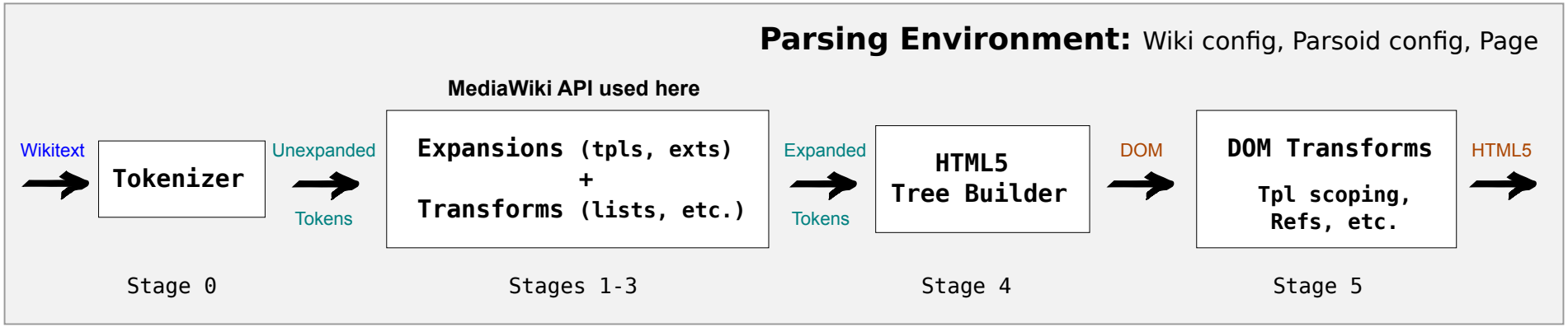
Outline

Parsoid pipeline: 10,000 feet view



Wikitext to HTML (wt2html) transformation

Parsoid pipeline: 10,000 feet view

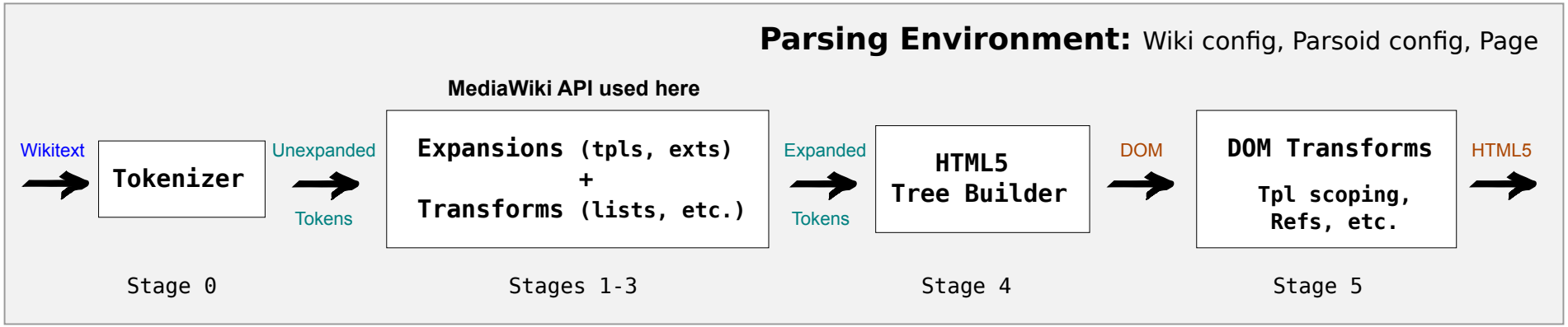


Wikitext to HTML (wt2html) transformation

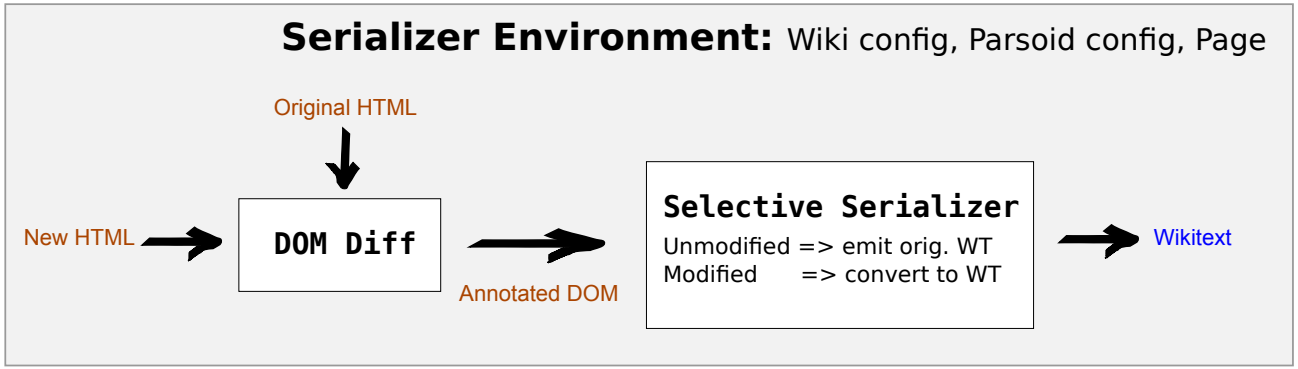
Pipeline	Stages	Input	Output
text/mediawiki/full	0-5	Wikitext	HTML
text/mediawiki	0-2	Wikitext	Expanded Tpls
tokens/mediawiki/expanded	3-5	Expanded Tpls	HTML

- Some pipeline types used by Parsoid
- Other pipelines can be constructed by hooking up modules and callbacks

Parsoid pipeline: 10,000 feet view



Wikitext to HTML (wt2html) transformation



HTML to Wikitext (html2wt) transformation

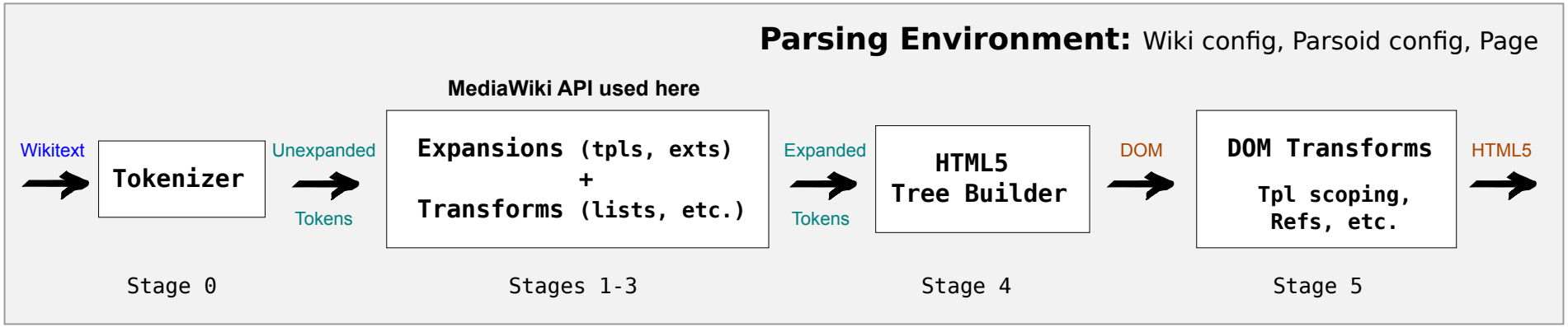
Outline

Tokenizer: Short overview

- Uses a PEG parser.
- Parses the context-free aspects of the syntax.
- Not possible to parse all wikitext constructs to final output because of context-sensitivity and transclusions.
- Token stream transformations and DOM passes (stages 1-5) handle context sensitive parts.

Outline

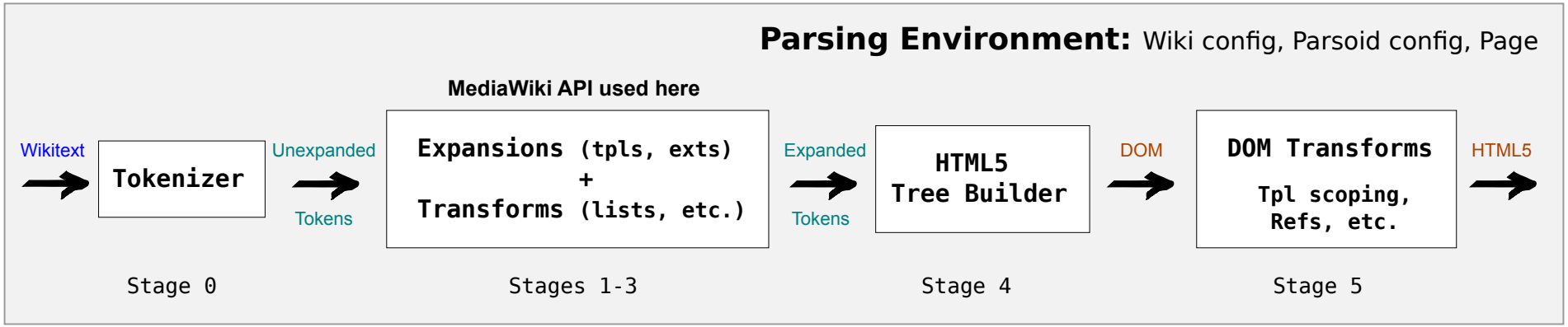
Example



Wikitext

```
a
*{{echo|b}}
```

Example



Wikitext

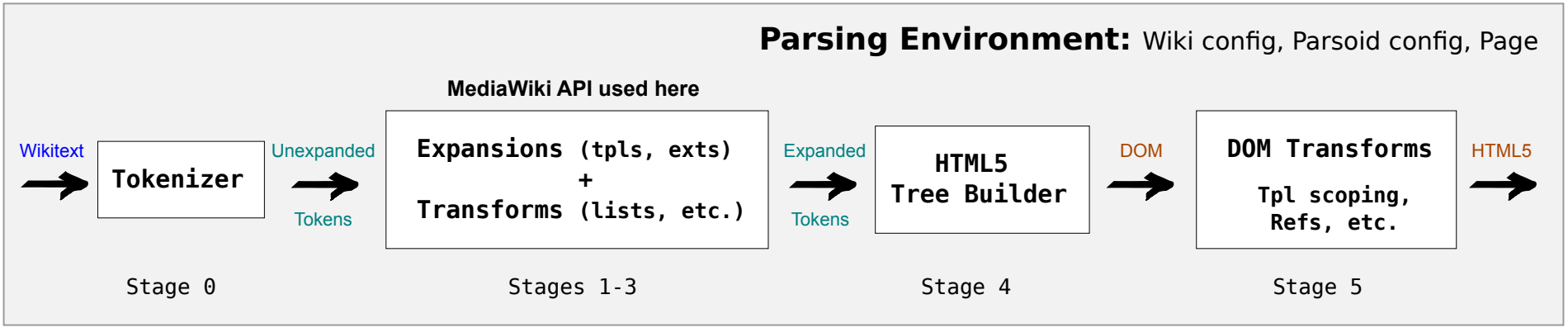
```
a
*{{echo|b}}
```

Tokenizer Output

```
% node parse --trace peg-tokens < wikitext*

"a", <NL>, <LI:*>, <TPL:echo:["b"]>, <NL>, <EOF>
```

* Trace output is simplified and reformatted.

Example



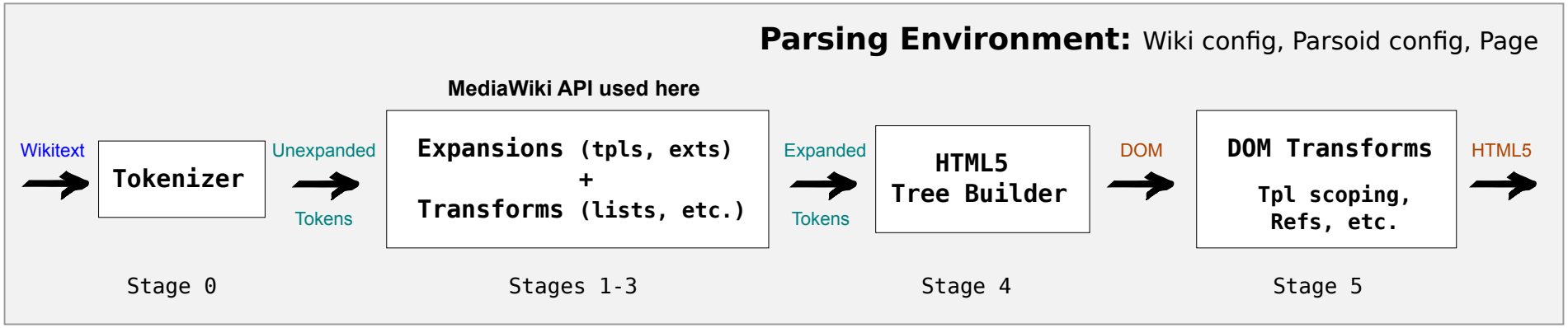
Wikitext

```
a
*{{echo|b}}
```

Expanded Tokens

```
% node parse --trace html < wikitext

<p>, "a", </p>, <NL>, <ul>, <li>, <tpl-start:1>, "b",
<tpl-end:1>, </li>, </ul>, <NL>, <EOF>
```

Example



Wikitext

```
a
*{{echo|b}}
```

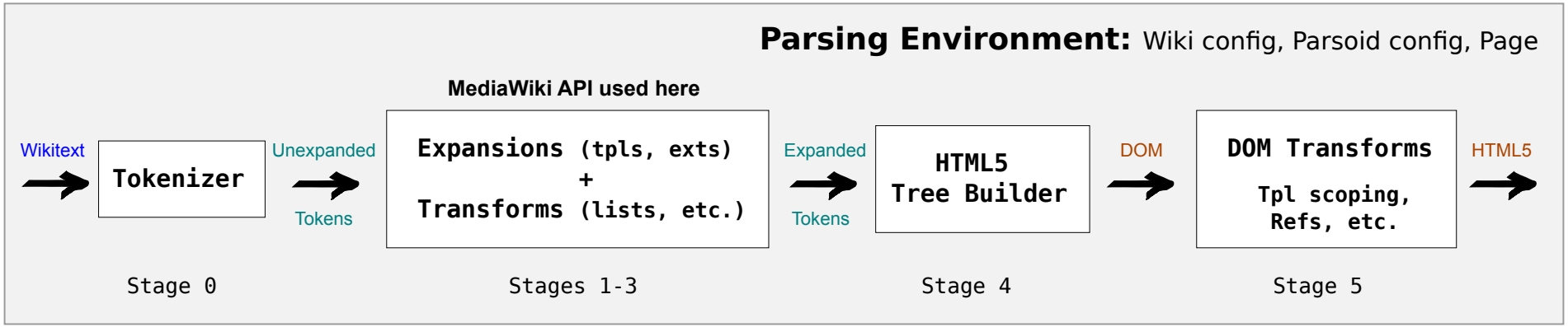
Simplified DOM* after some initial passes

```
% node parse --dump dom:pre-dsr < wikitext
```

```
<body><p>a</p>
<ul><li><meta typeof="mw:Transclusion" about="#mwt1">b<meta
typeof="mw:Transclusion/End" about="#mwt1"></li></ul>
</body>
```

* Nodes have addl. information in a data-parsoid attribute not shown here.

Example



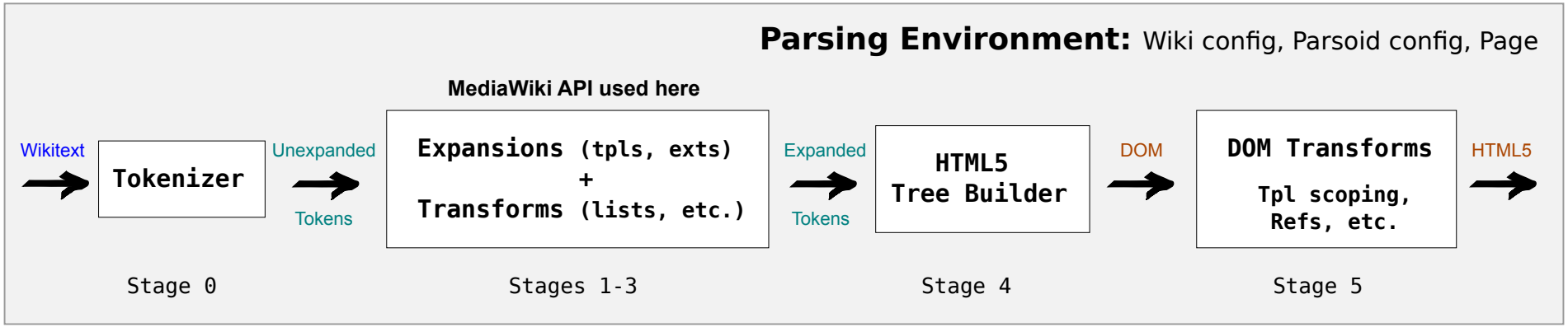
Wikitext

```
a
*{{echo|b}}
```

HTML (Reformatted)

```
<body data-parsoid='{"dsr":[0,14,0,0]}'>
<p data-parsoid='{"dsr":[0,1,0,0]}'>a</p>
<ul data-parsoid='{"dsr":[2,13,0,0]}'>
<li data-parsoid='{"dsr":[2,13,1,0]}'>
<span about="#mwt1" typeof="mw:Transclusion" data-mw=".."
data-parsoid="{..}">b</span></li></ul></body>
```

Example



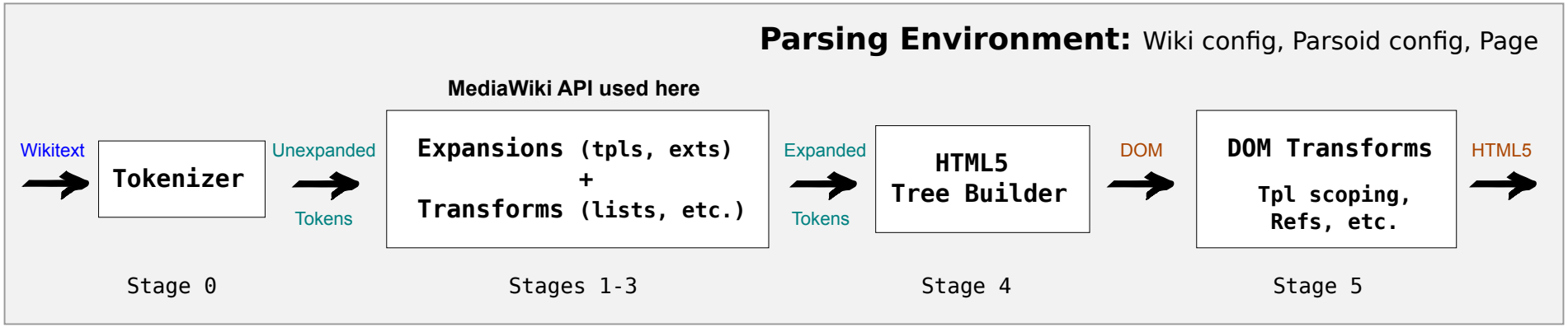
Wikitext

```
a
*{{echo|b}}
```

data-mw of the transclusion span

```
{"parts":[{
"template":{
"target":{"wt":"echo","href":"./Template:Echo"},
"params":{"1":{"wt":"b"}}
}}]}
```

Example



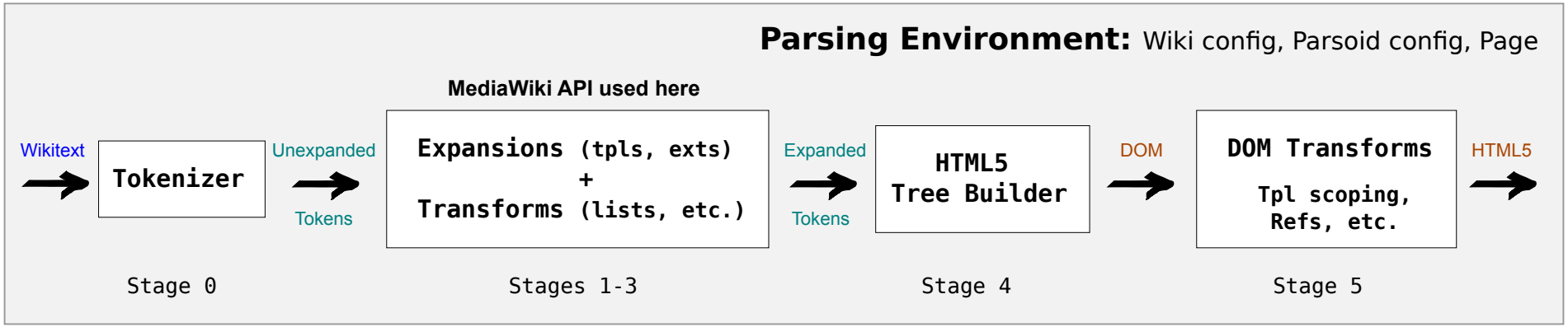
Wikitext

```
a
*{{echo|b}}
```

data-mw of the transclusion span

```
{"parts":[{
"template":{
"target":{"wt":"echo","href":"./Template:Echo"},
"params":{"1":{"wt":"b", "html": "b"}}
}}]}
```

Example



Wikitext

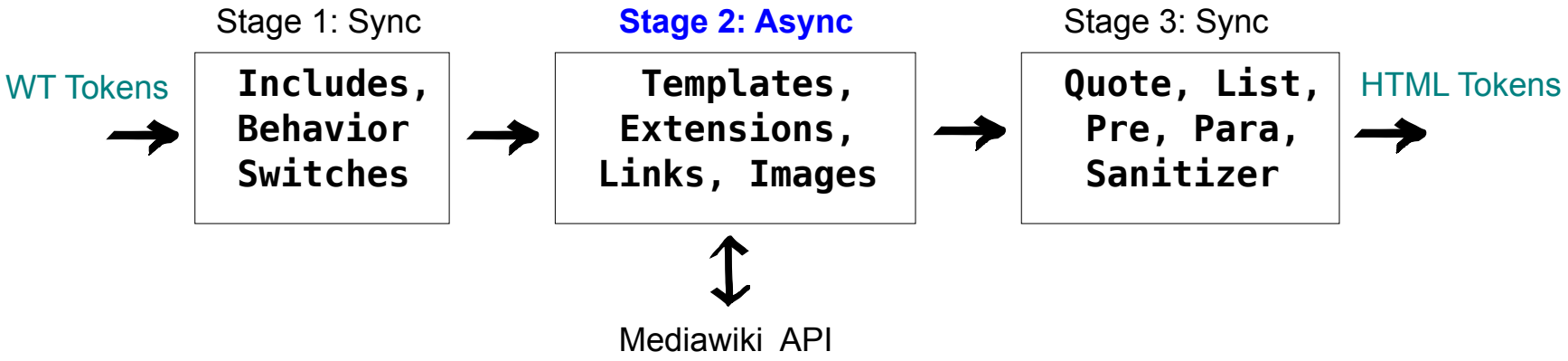
```
a
*{{echo|[[Foo]]}}
```

data-mw of the transclusion span

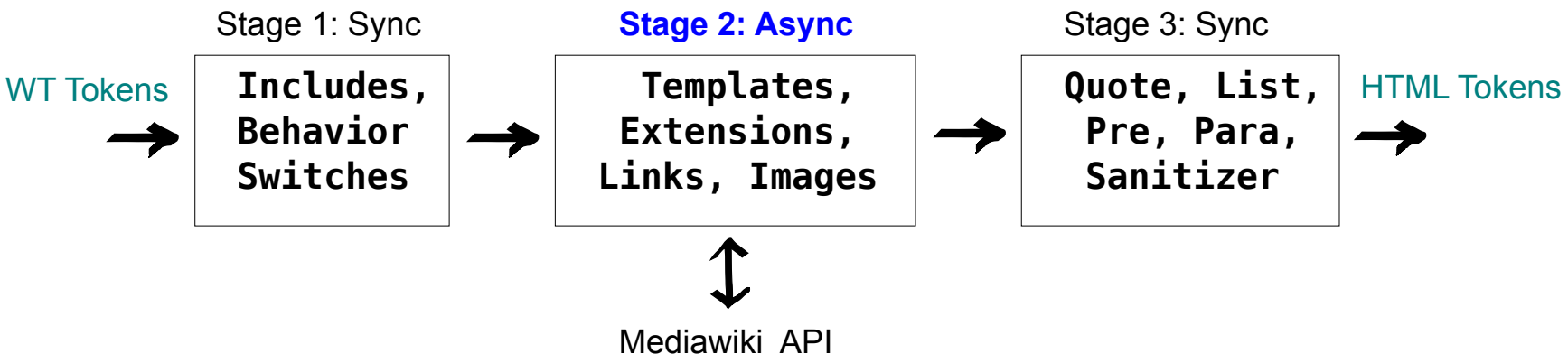
```
{"parts":[{
"template":{
"target":{"wt":"echo","href":"./Template:Echo"},
"params":{"1":{"wt":"[[Foo]]", "html": "<a href=..>Foo</a>"}}
}}]}
```

Outline

Zooming in: Stages 1-3: Template expansions, etc.

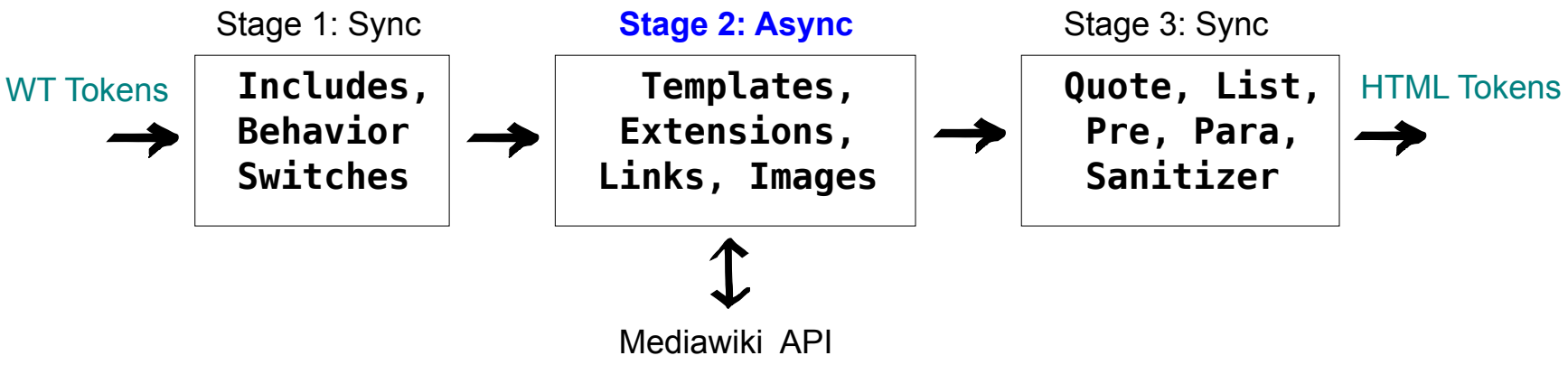


Zooming in: Stages 1-3: Template expansions, etc.



Wikitext	Unexpanded Tokens
a *{{echo|b}} <noinclude> {{Docs}} </noinclude>	.. <LI:*>, <TPL:echo:["b"]>, <NL>, <noinclude>, <TPL:Docs:[]>, </noinclude> ..

Zooming in: Stages 1-3: Template expansions, etc.



Wikitext

```
a
*{{echo|b}}
<noinclude>
{{Docs}}
</noinclude>
```

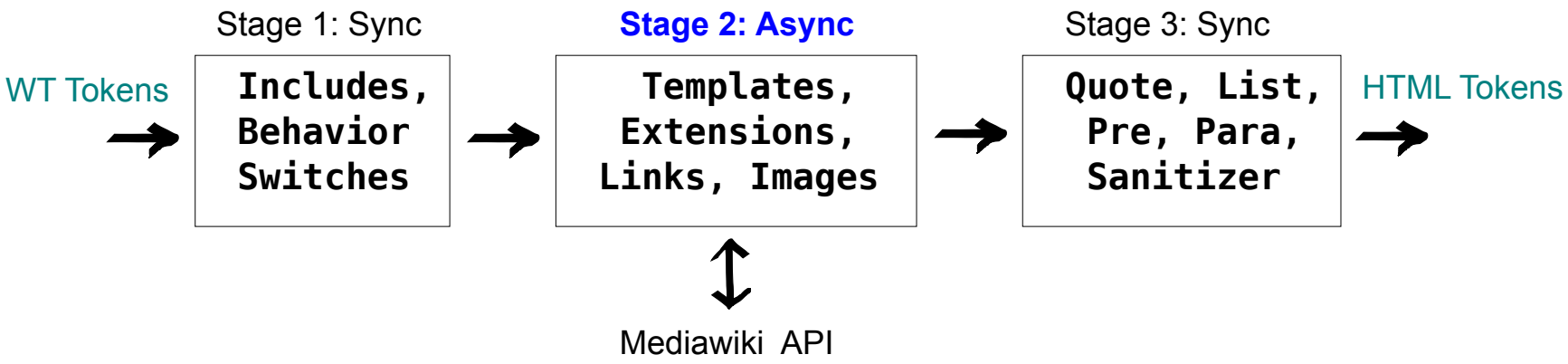
Unexpanded Tokens

```
.. <LI:*>, <TPL:echo:["b"]>, <NL>,
<noinclude>, <TPL:Docs:[]>, </noinclude> ..
```

Tokens after Stage 1

```
.. <LI:*>, <TPL:echo:["b"]>, <NL>,
<placeholder-for-RTing> ..
```

Zooming in: Stages 1-3: Template expansions, etc.



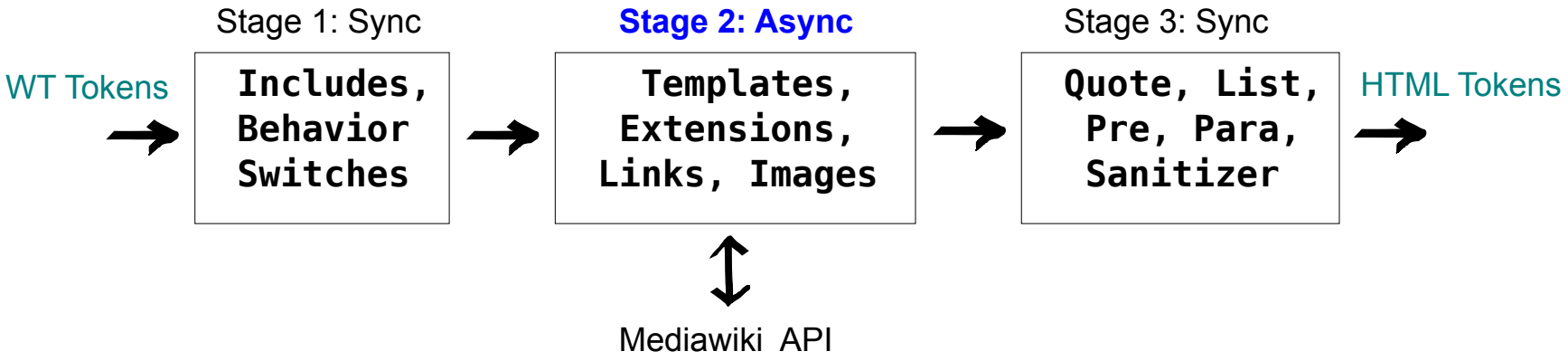
Wikitext

```
a
*{{echo|b}}
<noinclude>
{{Docs}}
</noinclude>
```

Expansion of template token: `<TPL:echo:["b"]>`

- Query mediawiki API for expanded wikitext: `b`
- Parse expanded wikitext in a **new pipeline**
  `"b", <EOF>`
- Wrap `tpl-start` and `tpl-end` tokens around it
  `<tpl-start:1>, "b", <tpl-end:1>`
- Splice tokens into the main token stream
  `.. <LI:*>, <tpl-start:1>, "b", <tpl-end:1>, <NL> ..`

Zooming in: Stages 1-3: Template expansions, etc.



Wikitext

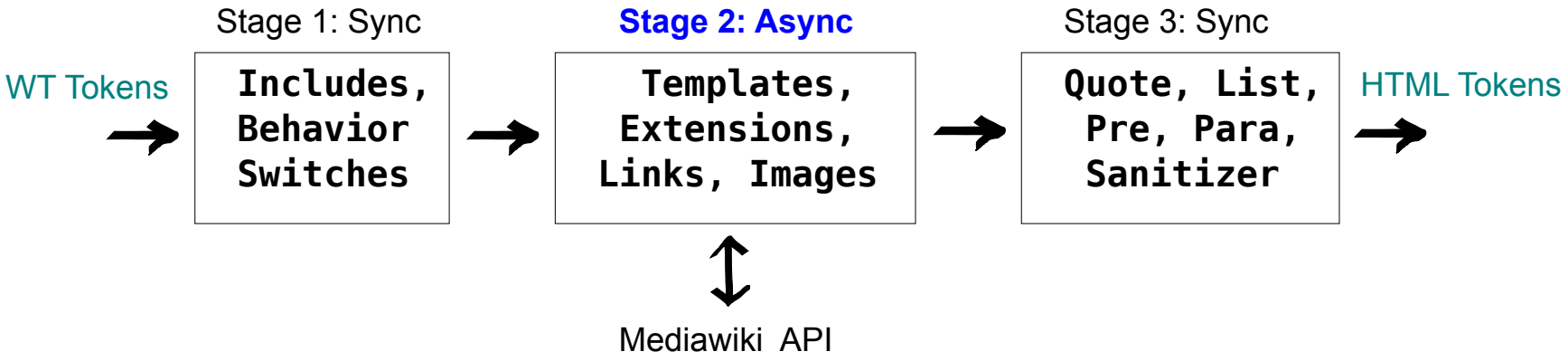
```
a
*{{echo|b}}
<noinclude>
{{Docs}}
</noinclude>

{{echo|[[Foo]]}}
{{Infobox|..}}
```

Expansion of template tokens

- Template tokens are processed asynchronously
- Multiple concurrent requests to MW API
- Token buffers ensure tokens are spliced in order

Outline

Stage 3: Transformations



- Stage 3 transformations run after all templates and extensions have been expanded.
- Quote Handler is more or less a straight port of the PHP parser's handler.
- Sanitizer is also a port of the PHP sanitizer.
- List, Indent Pre, and Paragraph transformers use state machines to transform the token stream.

Stage 3: Indent-Pre Handler

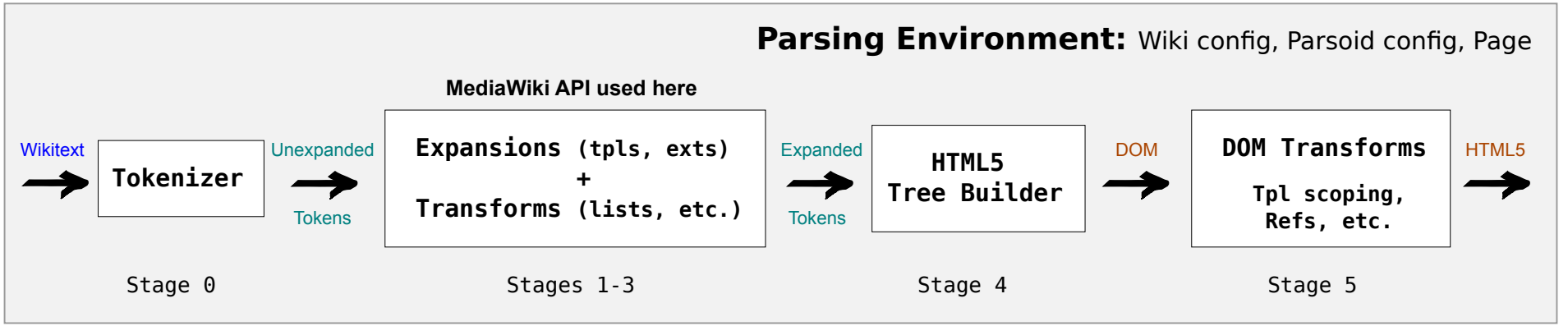
- Slice of the state machine shown below.

Start	Token	End	Action
SOL	nl/eof	SOL	Emit
	sol-tr	SOL	Buffer1
	ws	PRE	Buffer2
	other	IGNORE	Emit
PRE	nl/eof	SOL	Emit
	html-block-tag	IGNORE	Emit
	wt-table-tag	IGNORE	Emit
	sol-tr	PRE	Buffer3
	other	PRE-COLLECT	Buffer4
...	...	...	...

- SOL = Start of Line; WS = white space; Buffer1,2,3,4: buffering depending on state and token
- Handles single-line, multi-line pres, interaction with SOL-transparent tokens (comments, noinclude, categories), etc.

Outline

Stage 4: HTML Building

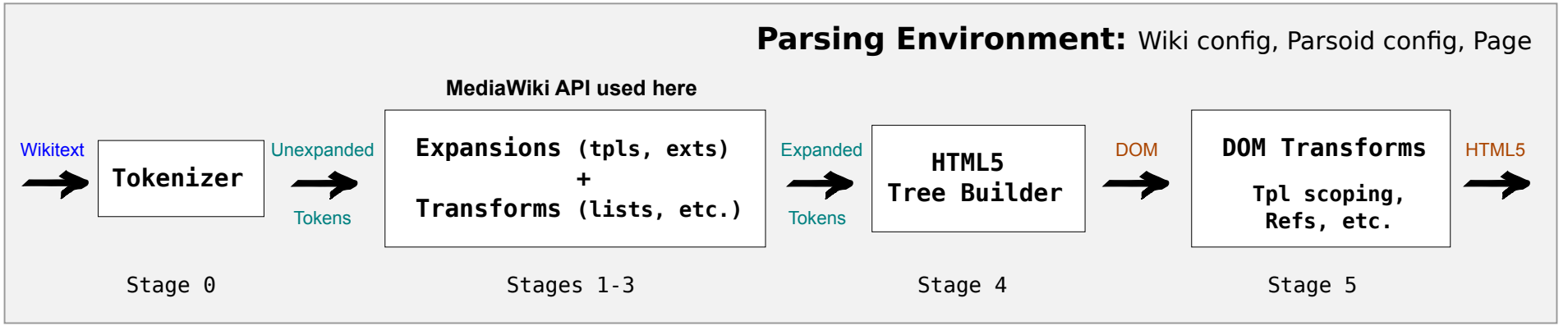


Wikitext to HTML (wt2html) transformation

- Use a standard HTML5 tree builder library.
- Use lot of tricks to detect fixup of misnested tags. `<i>X<b>Y</i></b>` to `<i>X<b>Y</b></i><b></b>`
  - Shadow tokens added for every node found in original stream.
  - Shadow tokens analyzed on constructed DOM to detect how misnested tags in HTML got fixed up – analysis necessary for accuracy of mapping wikitext to DOM.
- PHP parser relies on `Tidy` to fixup misnested tags – causes Parsoid and PHP parser output to occasionally differ.

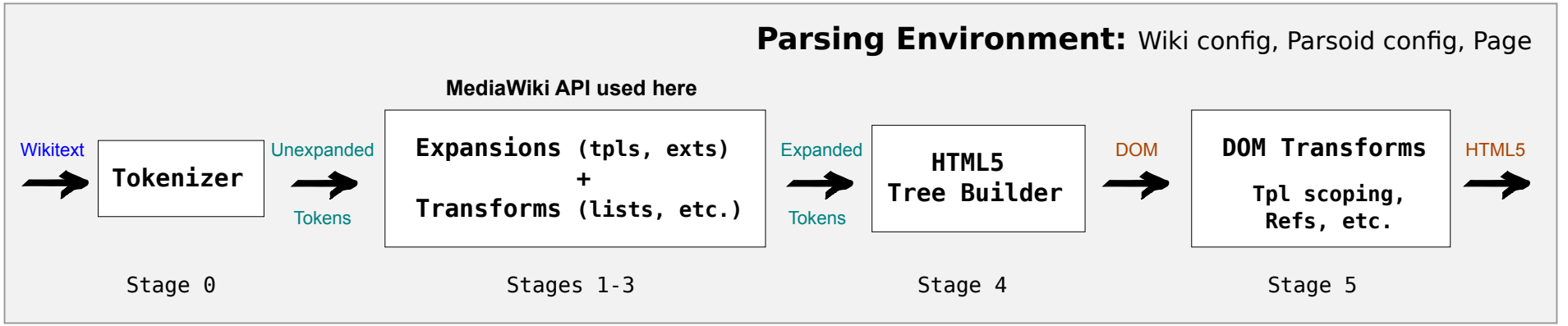
Outline

Stage 5: DOM transformations



Wikitext to HTML (wt2html) transformation

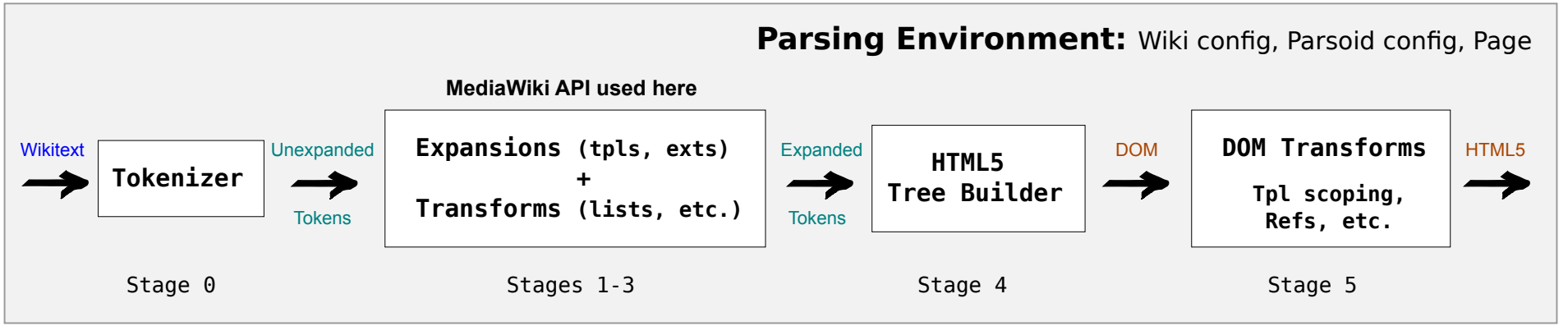
Stage 5: DOM transformations



Wikitext to HTML (wt2html) transformation

- DOM still needs lot of fixup (ex: template scoping).
- DOM's tree structure $\Rightarrow$ simpler & robust algorithms.
- Several passes transform the DOM.

Stage 5: DOM transformations



Wikitext to HTML (wt2html) transformation

1. Mark fostered content (Quite important!)
2. Mark HTML5 builder fixups
3. Map WT substrings to DOM subtrees (DSR computation)
4. Demarcate template scopes (Template encapsulation)
5. Handle link prefixes/trails
6. Generate references (Parsoid's native Cite impl)
7. Handle LI-hack, templated table cell attributes

Outline

DOM Pass: Marking fostered content

Fostered Content??

- Fostered content = content in `<table>`s that is badly nested and get adopted by a foster parent outside the table.
  Example:`<table>foo<tr><td>bar</td></table>`
  becomes:`foo<table><tr><td>bar</td></table>`
- This is part of the HTML5 spec – not something that Parsoid does.
- Sometimes, partial template content gets moved out.

DOM Pass: Marking fostered content

Q. Why is this a problem? A. Breaks content ordering

- Before this was fixed in Parsoid, this would not round-trip :

```
{|
|- fostered content
| foo
|}
```

round trips to

```
fostered content
{|
| foo
|}
```

- Interferes with ability to map wikitext strings to generated DOM nodes.
- When partial template content gets moved out, messes with template scoping.
- Used to cause more serious corruption (ex: content duplication) when these pages were serialized back.

DOM Pass: Marking fostered content

Basic idea* behind solving this:

- This pass adds markers before every table.
- This effectively creates a "foster box" between the marker and the table – i.e. content found between those two tags is fostered content.
- Later DOM passes now ignore fostered content in their analysis.
- Serializer relies on fostered content markers to avoid corruption.

* Caveat: Has to deal with other edge cases and fixups.

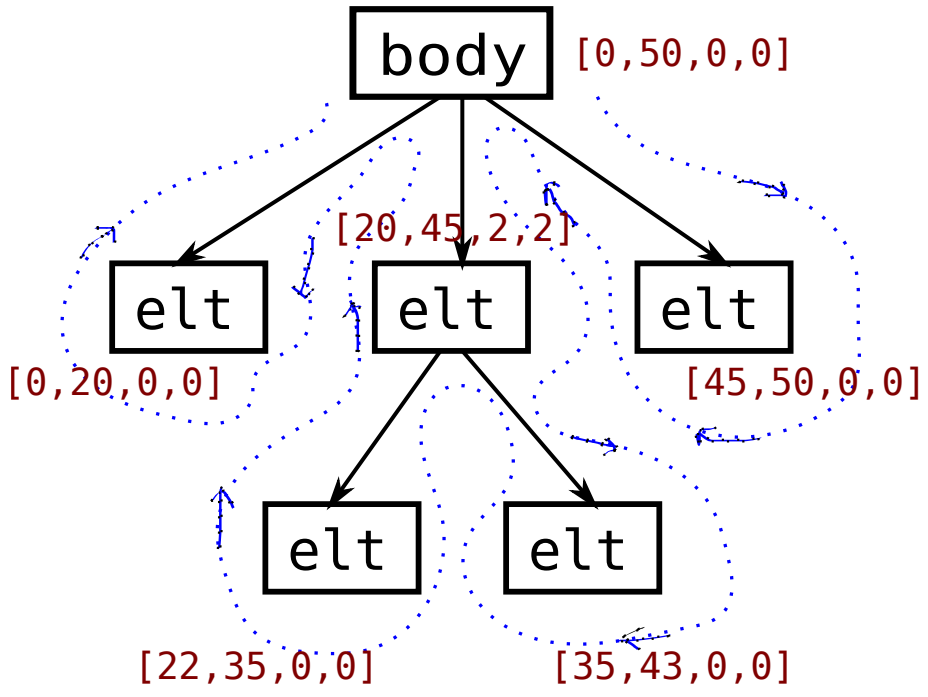
Outline

DOM Pass: DSR computation

DSR = DOM Source Range

- Assigned to every node in a DOM.
- 4-tuple: [start, end, start-tag-width, end-tag-width]
- Maps wikitext.substring(start,end) to a DOM node.
  "foo" will parse to foo and .dsr = [0,7,2,2]
  foo will parse to foo and .dsr = [0,10,3,4]
- Accuracy critical for selective serialization since it simply emits wikitext.substring(start,end) for unmodified DOM nodes.

DOM Pass: DSR computation



- DSR algo walks the DOM backward.
- Has a forward pass across siblings in addition at every level.
- Uses knowledge of wikitext and source wikitext position annotations from the tokenizer.
- Required us to fix other transformations, worry about newlines, etc.
- Uses redundant information to detect inconsistencies.

Outline

DOM Pass: Template Encapsulation/Scoping

- For clients like the VisualEditor, template output cannot be directly edited and should be edit-protected.
- For common transclusion scenarios, output of a single transclusion can be mapped to a forest of DOM trees. Example: {{echo|foo}} maps to <span..>foo
- Not true in the general case. Ex: Succession box templates ({{s-start}}, ... {{s-end}} and other such family of templates).

This pass associates a forest of adjacent DOM trees with a section of wikitext which includes one or more transclusions.

DOM Pass: Template Encapsulation/Scoping

- For clients like the VisualEditor, template output cannot be directly edited and should be edit-protected.
- For common transclusion scenarios, output of a single transclusion can be mapped to a forest of DOM trees. Example: {{echo|foo}} maps to <span..>foo
- Not true in the general case. Ex: Succession box templates ({{s-start}}, ... {{s-end}} and other such family of templates).

This pass associates a forest of adjacent DOM trees with a section of wikitext which includes one or more transclusions.

{{s-start}}
...
{{s-end}}

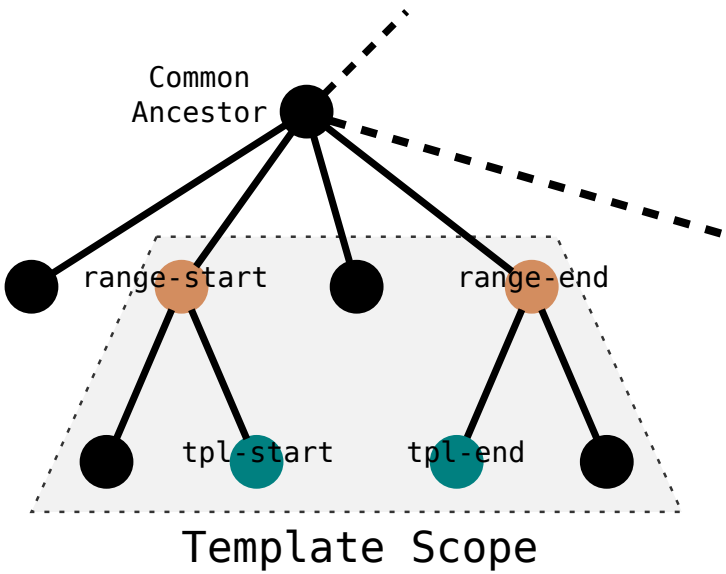
maps to

```
<table class="wikitable
succession-box"
about="#mwt1"
typeof="mw:Transclusion"
data-mw=".." ...>
...
</table>
```

DOM Pass: Template Encapsulation/Scoping

Algorithm outline

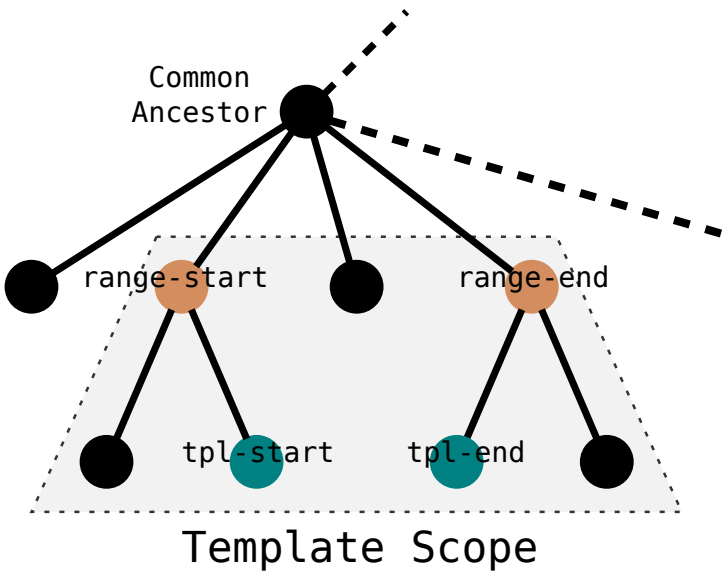
- Search for `<tpl-start>` and `<tpl-end>` markers and construct a set of DOM ranges (start and end DOM nodes that contain output of every transclusion).

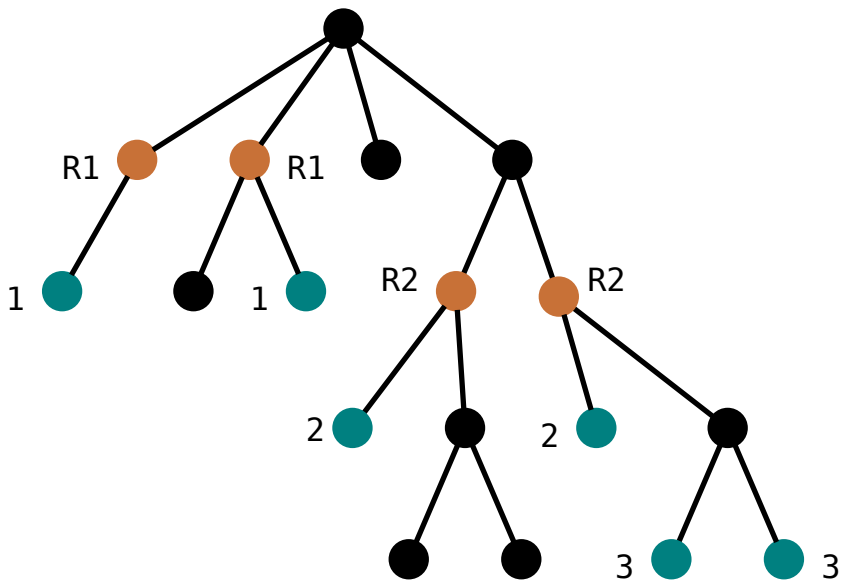


DOM Pass: Template Encapsulation/Scoping

Algorithm outline

- Search for `<tpl-start>` and `<tpl-end>` markers and construct a set of DOM ranges (start and end DOM nodes that contain output of every transclusion).
- Merge overlapping and nested ranges.
- For every range, set up about id, typeof, and data-mw.





Outline

Summary

- Generating html from wikitext is a fairly involved process.
- Roundtripping and editablity requirements are new when compared to the PHP parser and complicates parsing.
- Individual algorithms and solutions are mostly straightforward.
- But, lot of individual components and details to get right.

Thank you!
Questions?