Updates from the Wikitext Parsing world

C. Scott Ananian
Content Transform Team
SMWCon 2022, Oct 26, 2022

End goal

Use Parsoid for all wikitext processing

End goal

Use Parsoid for all wikitext processing

Then improve wikitext!

WIKIMEDIA
FOUNDATION

Why Parsoid?

- Core parser cannot support Parsoid clients (VE, etc.)
- Parsoid's annotated HTML provides more semantic information (for editing, bots, gadgets, etc)
- Two parsers not tenable and hamstrings future feature work
- Long-term transition to manipulation of balanced DOM trees & composition of rich typed fragments

Initial focus: Start transitioning the Wikimedia cluster

Some milestones

Rendering

✓ Migrate away from HTML4 Tidy to a HTML5 “tidier”

↣ Migrate core parser to use Parsoid HTML for media wikitext

☐ Use Parsoid for “read views”, starting with Discussion Tools

Functionality

✓ Equivalent ParserOutput, ParserOptions interfaces

↣ Provide a suitable Parsoid-compatible extension API

WIKIMEDIA
FOUNDATION

The future (briefly)

- Improvements to transclusion syntax and structure
  - Templates, parser functions, magic words, …
  - Typed/balanced results, typed/protected arguments
- Likely shifts in parsing model
  - Restructured caching, more postprocessing, more variants
  - Separate “parse” and “compose” steps
  - Multiple async “compose” steps (proposed)

Composition of typed fragments

- Treat transclusions as independently computed cacheable fragments
- Provide richer “output” types that plug together well
  - DOM tree, attribute pairs, structured data?
- Provide richer “parameter” types w/ good editor support
  - [[Extension:TemplateData]] is moving in this direction

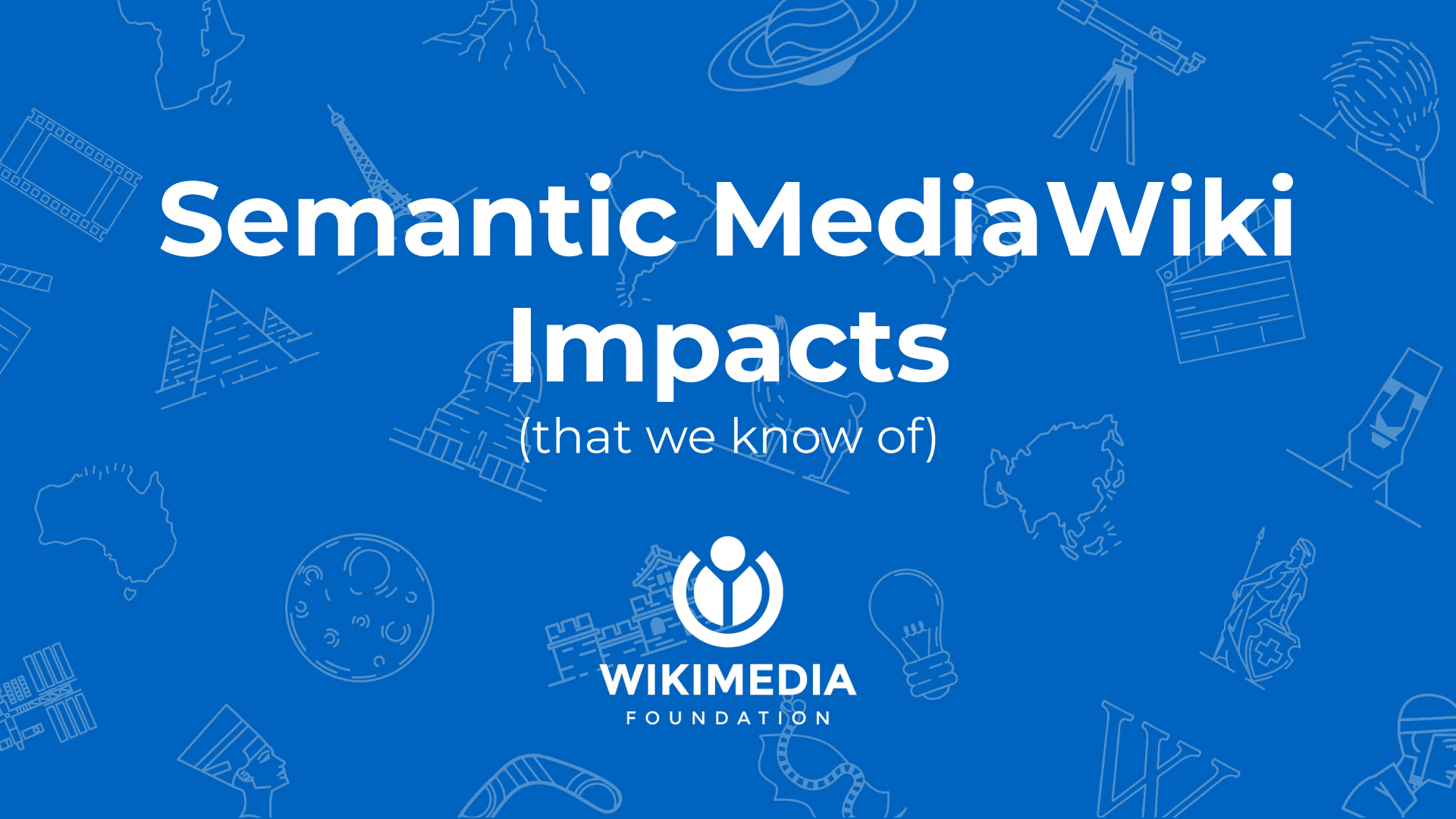
WIKIMEDIA FOUNDATION

Postprocessing

- Fragments are independent, but there is often a need to do some global reconciliation
  - Simple example: sequentially numbering references
- We have a final “post-composition” step for this
  - Try to keep it fast!
- Although a lot of “sequential” behavior can be emulated this way, the code to do so usually looks very different.

Semantic MediaWiki Impacts

(that we know of)



Semantic MediaWiki Impacts

- Depending on a linear parse order will be discouraged
  - Page-order dependencies in the Variables extension
  - Metadata composition in ParserOutput
- Extending the `[[....]]` link syntax will be deprecated
  - Use `<ext>` for API stability or `{{#parserFunction}}`
  - T204370: uniform `{{#...}}` syntax
- Parser Function API likely to be extended/improved
- Likely other updates needed? Help us find issues early.

WIKIMEDIA
FOUNDATION

Link syntax

- SMW hooks `InternalParseBeforeLinks` to replace `[[...]]` syntax “before” the MW legacy parser sees it
  - Parsoid does not have parser stages like this
- Two solutions: (T76278)
  - Introduce a specific hook to allow look-aside on `[[...]]` syntax specifically (no current plan to do this)
  - Use the alternative `{{#...}}` syntax (much preferred; I believe this is current community consensus)
- Is linter/migration help needed?

WIKIMEDIA FOUNDATION

Variables extension (T250963)

- The widely-used Variables extension uses the deprecated `InternalParseBeforeSanitize` hook.
  - Parsoid does not have parser stages like this
- It also introduces a central read/write store in the Parser, with constructs affecting all following wikitext
  - This conflicts with incremental/async parsing goals
  - But it's not unique in that: citations, LanguageConverter, etc, also have had that property (but we're trying to fix that)

Variables extension (T250963)

- Also: `#var_final`
- Some possible solutions:
  - Lint away `#var_final` – or make everything `#var_final`
  - Do all/most work in the final postprocessing phase
  - Forced linear parsing (T282499)
  - Come up with alternative/focused means to cache Cargo queries (is this the primary use case?)
    - https://www.mediawiki.org/wiki/Extension:Variables#Alternatives
    - Marijn van Wezel at Wikibase Solutions has a very interesting approach!

WIKIMEDIA FOUNDATION

Separation of Presentation and Data

- Instead of interleaving computation with wikitext and formatting, split off the computation into its own thing
- Evaluate the page as presentation “in the context of data”

https://gitlab.wikibase.nl/community/arrayfunctions/

WIKIMEDIA FOUNDATION

One page, three titles

In [[MainArticle]]:

```
{{MainArticlePresentation|{{#invoke:MainArticleData}}}}
```

In [[Template:MainArticlePresentation]]:

```
<table>
{{#af_foreach:{{{1}}} | key | value |
<tr><td>{{{key}}}</td><td>{{{value}}}</td></tr>
}}
</table>
```

In [[Module:MainArticleData]]:

```
return mw.af.export({
  “Row 1” = “First”,
  “Row 2” = “Second”,
})
```

WIKIMEDIA
FOUNDATION

Real-World Concerns

- Embedding certain wikitext constructs in transclusion arguments can be tricky.
  - Heredoc arguments should help (T114432)
- Memoizing computed data was necessary
  - Fragment composition framework tries to generalize this to allow broader memoization and reuse
- Passing structured data between transclusions
  - Richer types for parameters and results would help

WIKIMEDIA
FOUNDATION

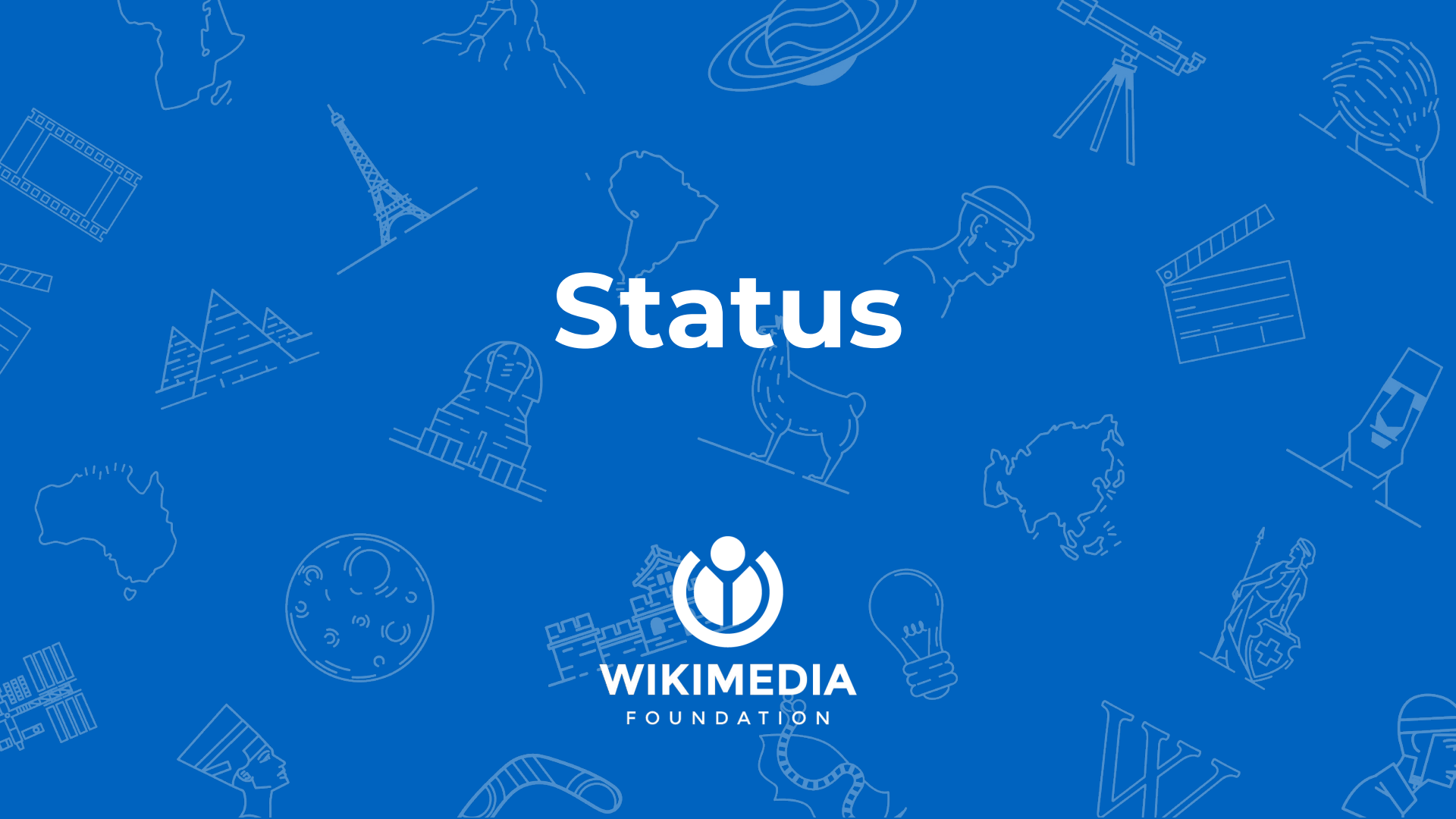
Looking forward: Parser Function API

- Parser function API is currently quite primitive
  - Eg, no support for named or numbered arguments!
- Currently area of active work, but is hoped that we can introduce “soon” a declarative API consistent with extension tags/magic words/template data, that can allow richer input and output types (at least: “raw text” input and “DOM tree” output)

Let’s discuss!

SKIP FROM HERE ON

Status



Semantic Media HTML

Pretty far along (**T51097**)

- Opt-in feature flag added to MediaWiki 1.37
  - Set `$wgParserEnableLegacyMediaDOM` to false to switch
- Mediawiki.org has this enabled since end-Sep 2021
- Tweaking CSS, JS, extensions, bots, gadgets, etc. in progress
- Latest step in broad project of unifying HTML w/ legacy parser

Please test your wikis and extensions!

WIKIMEDIA
FOUNDATION

Parsoid Extension API

Work in Progress

- Initial Focus: Tag extensions (`<ext ..>` … `</ext>`)
  - Mostly done, but some edge cases remain
  - See mw:Parsoid/Extension_API; August 2020 Tech Talk
- Work in Progress: Support for Parser Hooks
  - Partially done; working through some sticky issues
- Work in Progress: ParserOutput usage

WIKIMEDIA
FOUNDATION

ParserOutput equivalent

Preliminary work done

- Abstract class in Parsoid, extended by ParserOutput in core
  - Narrower interface than ParserOutput
  - Names & types & other hairy bits cleaned up
  - Write-only API

WIKIMEDIA
FOUNDATION

Parsoid & Extensions

The TLDR

Bad news:

- All parser extensions will need to be updated eventually

Good news:

- Most extensions are probably minimally affected
- WIP to reduce upfront work needed by extensions and continue to rely on current parser interfaces

The TLDR

Different pipeline stages in Parsoid

- Pipeline stage events NOT exposed to extensions
  - Legacy: `ParserBeforePreprocess`, `ParserBeforeInternalParse`, `InternalParseBeforeSanitize`, `InternalParseBeforeLinks`, `ParserAfterTidy`, `ParserAfterParse`
- Emphasis on transformation hooks
- Global DOM processors in Parsoid equivalent to some existing hooks

WIKIMEDIA
FOUNDATION

The TLDR

- No processing order guarantees
  - Cannot maintain global ordered state in extensions (ex: counters)
  - Allows future asynchronous and incremental rendering, caching
- Recommended approach
  - Treat each `<tag>..</tag>` instance as an independent document
  - Record information in the HTML / DOM for this instance
  - Register global DOM processor to process page-level DOM with info from all instances (Should be fast! No heavyweight processing here.)
  - Eliminates need to maintain global state within extensions

WIKIMEDIA
FOUNDATION

Extension porting status

- Tag extensions:
  - In (Wikimedia) production: Gallery, Nowiki, Pre, Cite, Poem, ImageMap
  - **Awaiting (Wikimedia) deploy: Translate**
- ContentHandler Extensions:
  - In (Wikimedia) production: JSON
- ParserTests extensions:
  - In use: RawHtml, StyleTag

NEW: annotation tags

Specialized tag extension type (experimental; details WIP)

Big picture: for “transparent” tags, which add information but **do not affect the rendering of the contained wikitext**.

- Tags stripped out from wikitext, replaced with marker tags in HTML with a data-mw blob
- Extension can inspect DOM and recover needed information
- Additional details are being fleshed out
- Ex: <translate>...</translate>

WIKIMEDIA FOUNDATION

For more details ...

- See Wikimedia Tech Talk from August 2020
- See mw:Tech_Talks for slides & video
- https://mediawiki.org/wiki/Parsoid/Extension_API
- Look at Parsoid's implementations for Poem, Pre, Cite, etc.
- Look at Parsoid docs for the Ext/ namespace @ https://doc.wikimedia.org/Parsoid-PHP/master/
- Parsoid HTML spec @ https://www.mediawiki.org/wiki/Specs/HTML

WIKIMEDIA
FOUNDATION

Why .... ?

- Core parser hooks refer to parsing stages
  - `ParserBeforeStrip, ParserAfterStrip`
  - `ParserBeforeTidy, ParserAfterTidy, ParserAfterParse`
  - `InternalParseBeforeLinks, BeforeParserFetchFileAndTitle`
- Parsoid's pipeline is different
  - It has very different pipeline stages
    - **wt** → **html**: tokenizer, 15+ token passes, DOM builder, 20+ DOM passes
    - **html** → **wt**: DOM builder, 2+ DOM passes, Serializer
  - Pipeline keeps changing over time ⇒ **not exposed to extensions**

WIKIMEDIA FOUNDATION

Why .... ?

- Core parser exposes sequential processing
- Parsoid does not expose processing order
  - Parsoid (JS) had out-of-order async processing ⇒ *your extension tags could be processed out-of-order*
  - While disabled now, Parsoid reused content from a previous parse ⇒ *your extension will not be invoked on that content*
  - Likely to reintroduce async processing into Parsoid for WikiFunctions

⇒ We cannot expose / guarantee specific processing order

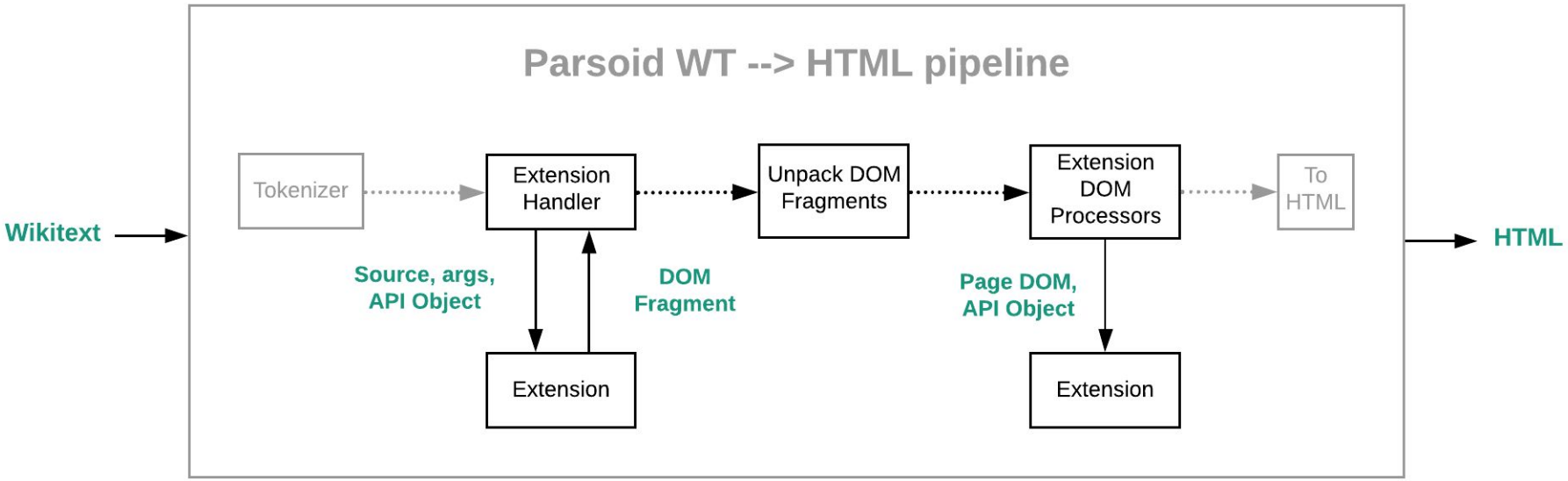
WIKIMEDIA
FOUNDATION

Why .... ?

- Core parser treats everything as strings
  - Parsoid treats extension output as documents
- Core parser exposes its methods & objects directly
  - Parsoid prefers keeping implementation details opaque
  - Parsoid provides an API object to extensions instead

WIKIMEDIA
FOUNDATION

Tag extensions: Overview



WIKIMEDIA FOUNDATION

Hooks

- Places to hook in wt → html direction
  - One localized transformation hook: (**source, args) → DOM**
  - One global DOM processing hook: **DOM → DOM**
  - **Not shown in diagram**: One wikitext linting hook (if your extension handles wikitext)
  - Maybe others in the future (depending on use case)

Extension Output

- DOM is annotated with `typeof` & `data-mw` attributes
  - `<div typeof="mw:Extension/Poem" data-mw="{attrs: { … }, … }">..</div>`

- DOM is tunneled through pipeline
  - Placeholder node represents the DOM output until it is unpacked ⇒ DOM is unmodified by intervening passes
    - Similar to strip-state mechanism that exts. explicitly manage currently
    - Extensions don't need to do anything special

WIKIMEDIA FOUNDATION

Extension registration

- extension.json adds **`ParsoidModules`** for config
  - One of 2 options
    - Inline JSON config
    - ObjectFactory declaration
  - ObjectFactory decl. should provide **`ExtensionModule`** interface impl.
    - Interface has one method: `getConfig()`

WIKIMEDIA
FOUNDATION

Example config

```
{
      'name' => 'Cite',
      'tags' => [
            [
                  'name' => 'ref', 'handler' =>Ref::class,
                  'options' => [ 'wt2html' => [ … ], 'html2wt' => [ … ]  ],
            ],
            [ … one for <references> as well with html2wt options … ]
      ],
      'domProcessors' => [ RefProcessor::class ],
      …
}
```

Extends **ExtensionTagHandler** class

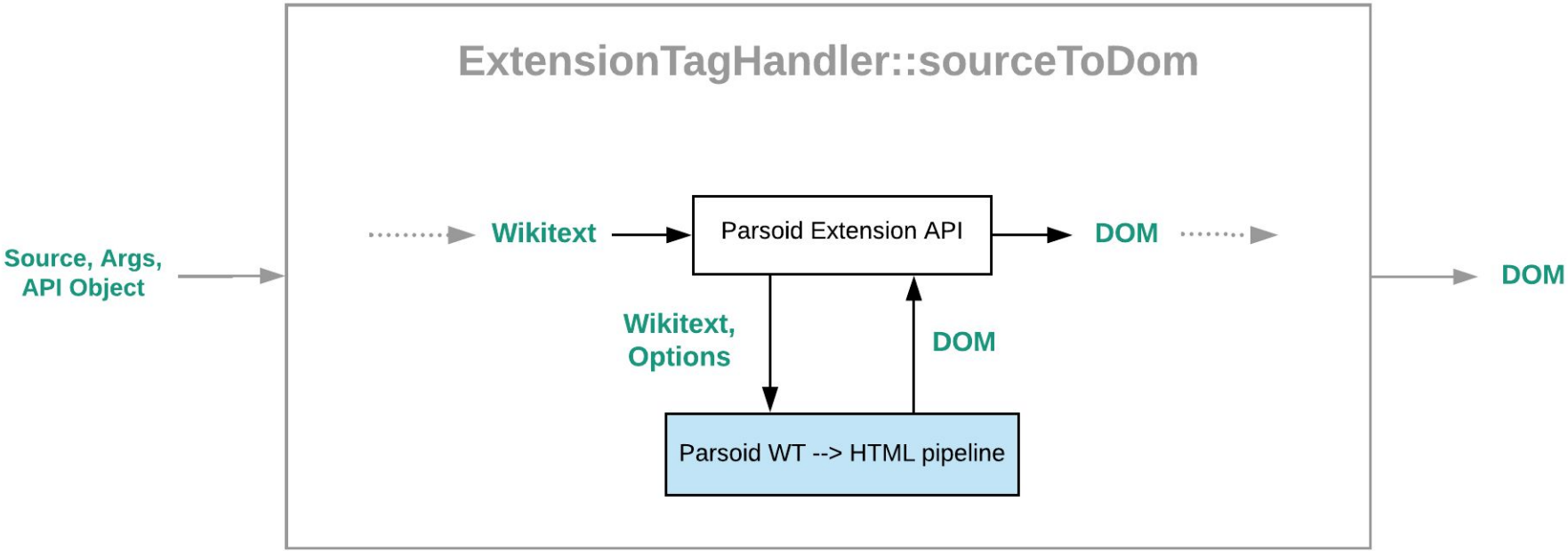
Extends **DOMProcessor** class

WIKIMEDIA FOUNDATION

ExtensionTagHandler

- Declares transformation hooks with dummy impls
  - `sourceToDom($api, string $src, array $args): DOM*`
  - `domToWikitext($api, DOMDocument $dom, …): string`
  - `lintHandler($api, $dom, $defaultLintHandler)`
  - `.....`
- Expect **`sourceToDom`** will be implemented
- **`domToWikitext`** optional
  - Parsoid provides default handling

WIKIMEDIA FOUNDATION



WIKIMEDIA FOUNDATION

Observations

- Use the API object to process wikitext
- Minimal control over parsing pipeline
  - You cannot run specific pipeline stages
  - You can specify output type / embedding context
    - Currently, **inline** or **block**
    - Additional output type / embedding contexts might be available in the future
- Output DOM has all applicable passes run
  - DOM passes that only apply to top-level Page DOM are skipped

Underlying principle

Wikitext should behave uniformly no matter where it shows up

- All deviations should have some conceptual grounding
  - Ex: embedding context type (CSS, HTML attribute, inline / phrasing content, table cell, etc.) introduces output constraints
  - No arbitrary subsets - https://phabricator.wikimedia.org/T192037

WIKIMEDIA FOUNDATION

ParsoidExtensionAPI

- Categories of API methods today:
  - Wikitext → DOM; DOM → wikitext (multiple methods)
  - **HTML → DOM; DOM → HTML** (vs. native DOM / library methods)
  - Methods that deal with extension args
  - get* methods (title, page URI, config objects, etc.)
  - ExtensionTag methods (query properties about `<ext … >` usage)
  - A few others (some transitional and may go away)

WIKIMEDIA FOUNDATION

Examples

Extension tag types

- Don’t wrap wikitext: `nowiki`, **`pre`**, `syntaxhighlight`, **`rawhtml`**
  - `$output = genDOM($input)`
- Thin wrapper over wikitext: **`ref`**
  - `$output = parseWT($input)`
- Process content as more-or-less-wikitext: **`poem`**
  - `$output = postProcessDOM(parseWT(mangle($input)))`
- Content has wikitext snippets that are processed separately: **`gallery`**
  - `$output = buildDOM(LOOP(parseWT(mangle($frag))))`

WIKIMEDIA
FOUNDATION

RawHTML extension

```
class RawHTML extends ExtensionTagHandler implements ExtensionModule {
    public function getConfig(): array {
        return [ 'name' => 'RawHTML',
            'tags' => [ [ 'name' => 'rawhtml', 'handler' => self::class ] ]
        ];
    }
    public function sourceToDom(ParsoidExtensionAPI $api, $src, $args) {
        return $api->htmlToDom($src); // returns DOM*
    }
}
```

WIKIMEDIA FOUNDATION


<pre>
WIKIMEDIA
FOUNDATION

Pseudocode

```
function sourceToDom(ParsoidExtensionAPI $api, $txt, $args): DOM* {
    $doc = $api->htmlToDom(''); // Empty doc
    $pre = $doc->createElement('pre');
    $api->sanitizeArgs($pre, $args);
    $txt = decodeWtEntities(trimLeadingNL(stripNoWikis($txt)));
    $pre->appendChild($doc->createTextNode($txt));
    DOMCompat::getBody($doc)->appendChild($pre);  // libxml fixes; T215000
    return $doc;
}
```

WIKIMEDIA
FOUNDATION


<ref>
WIKIMEDIA
FOUNDATION

<ref> Example

Sample Wikitext

```
Foo <ref>''AB'' and '''CD'''</ref>
  and bar and baz.
== References ==
<references />
```

Rendered Output

Foo [1] and bar and baz.

References

1. ^ *AB* and **CD**

WIKIMEDIA FOUNDATION

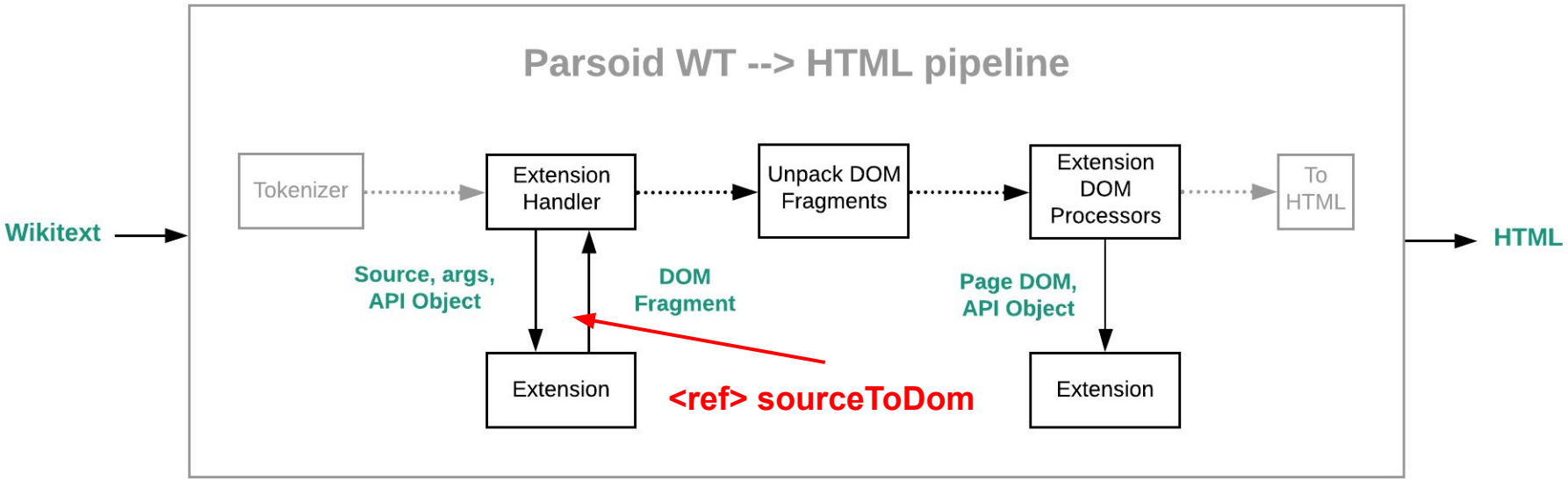
<ref> sourceToDom

```
function sourceToDom(ParsoidExtensionAPI $api, $txt, $args): DOM* {
    ... some checks to detect ref-in-ref scenarios ...
    return $api->extTagToDOM($args, $txt, [
        'wrapperTag' => 'sup', // DOM is wrapped in <sup> tag
        'parseOpts' => [
            'context' => 'inline', // No paragraphs, No "indent-pre"
            'extTag' => 'ref', 'extTagOpts' => [ 'allowNestedRef' => … ],
        ]
    ]);
}
```

WIKIMEDIA FOUNDATION

Wait a minute …

- That handler returned DOM of <ref>’s content
- How does that content migrate to the references section?
- What happened to numbered ref links?



WIKIMEDIA FOUNDATION

<ref> sourceToDom

- **Reminder**: this is the local transformation hook
- Cannot reliably count in the right order to generate links
- Does not have access to the final DOM

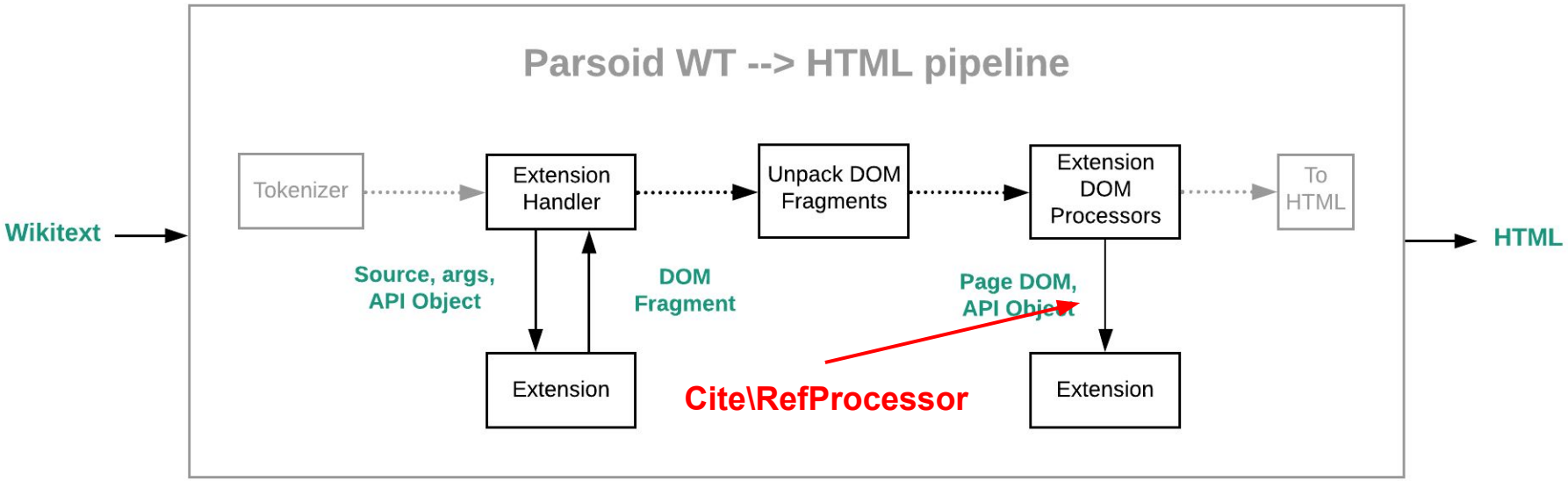
Cite config from before

```
{
	'name' => 'Cite',
	'tags' => [
		[
			'name' => 'ref', 'handler' =>Ref::class,
			...
		],
		...
	],
	'domProcessors' => [ RefProcessor::class ],
	...
}
```

Extends **ExtensionTagHandler** class;
Implements local transformation hook

Extends **DOMProcessor** class;
Implements global DOM processing hook

WIKIMEDIA
FOUNDATION



WIKIMEDIA FOUNDATION

Cite\RefProcessor

TIP: Parsoid adds ext. output to DOM. Page DOM is your global state object.

- `RefProcessor::wtPostProcess`
  - Has access to output of all `<ref>` tags via the Page DOM
  - Walks the tree (depth-first) **in-order** and harvests `<ref>` content
  - Generates the `<references>` section in required format
  - Migrates the `<sup>` content to the `<references>` section
  - Updates `<sup>` at `<ref>` sites with links to the `<references>` section

Effectively restructures the DOM

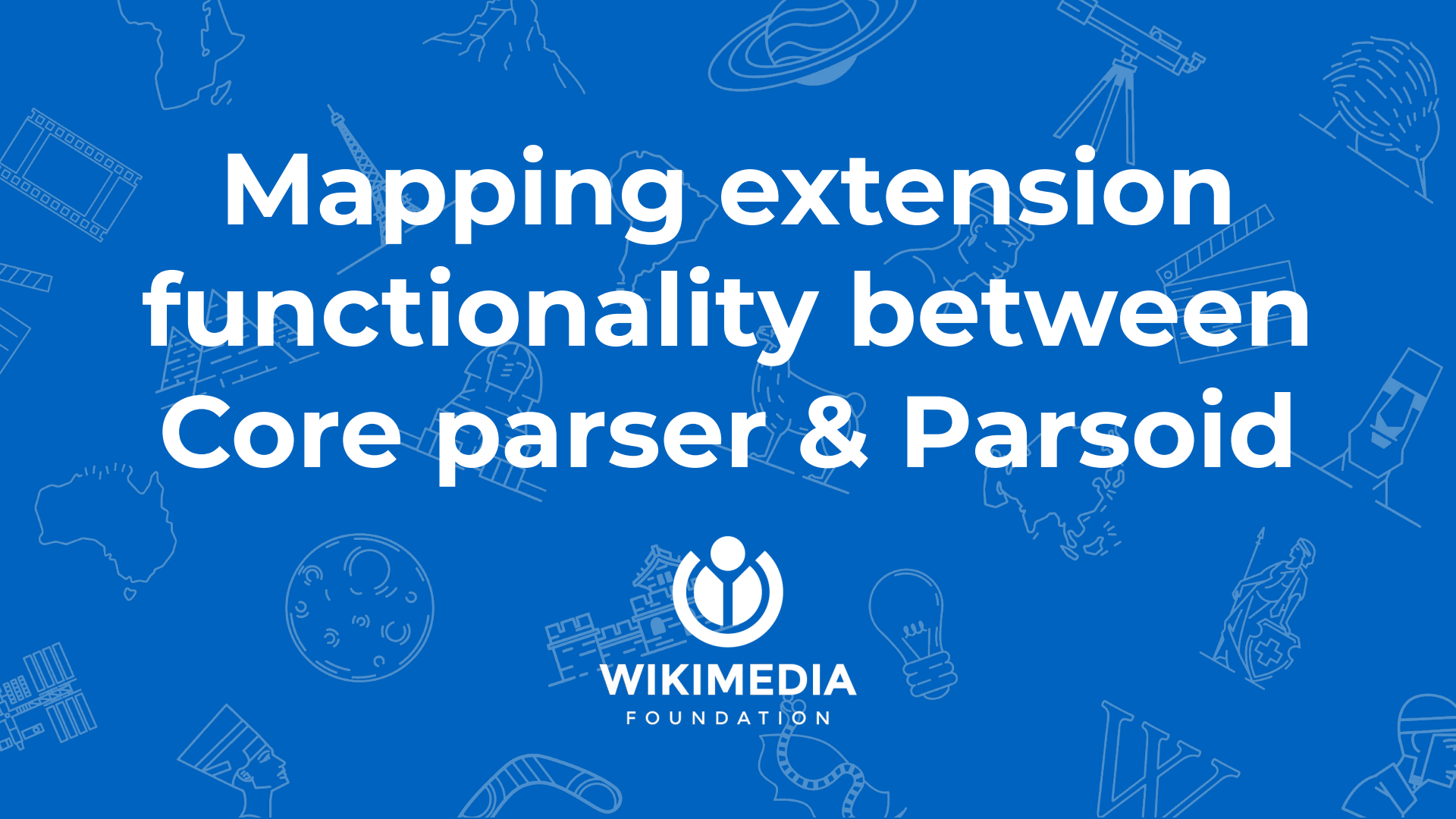
WIKIMEDIA
FOUNDATION

DOMProcessor class

- `wtPostProcess($api, $root, $opts, $atTopLevel)`
  - Invoked by Parsoid when the full page is constructed
  - Almost at “the end” when most, but not all, information is in the DOM
    - MediaInfo updates, Link annotation (external, red, disambig), LangConverter, Heading ids, other DOM fixups, section wrapping haven’t run yet
  - For extensions that might need all info, we might introduce a new DOM processor hook (maybe **`finalizeDoc($api, $root)`**)?
- `htmlPreProcessor($api, $root)`
- `Maybe others … ?`

WIKIMEDIA FOUNDATION

Mapping extension functionality between Core parser & Parsoid



Mapping: Parser hooks

- `ParserFirstCallInit:`
  - Register tag handlers directly in config
- `ParserBeforeTidy, ParserAfterTidy, ParserAfterParse:`
  - Use `wtPostProcess` DOMProcessor hook or if necessary, we can provide a `finalizeDoc` DOMProcessor hook
- `ParserClearState:`
  - Should not be needed - let us know if you have a use case for this
- `ParserBeforeStrip, ParserAfterStrip:`
  - Should not be needed - let us know if you have a use case for this

Mapping: Parser hooks

- `ParserLimit*:`
  - Unaffected. Will be refactored into meta-parsing functionality.
- `InternalParseBeforeLinks:`
  - Will not support (link syntax heavily overloaded in wikitext)
  - Use `wtPostProcess` DOM hook if you want to update links in any way
  - Alternatively, use different syntax (ex: parser functions)

Watch mw:Parsoid/Extension_API for complete mapping between hooks

WIKIMEDIA
FOUNDATION

Mapping: Parser API

Replacements for `parse`, `internalParse`, `startExternalParse`, `recursiveTagParse`, `recursiveTagParseFully`

- `extTagToDOM`: use when tag wraps wikitext (ex: <ref>)
- `extArgToDOM`: use when you need to process an arg as wikitext (ex: <gallery> caption)
- `renderMedia`: *what it says on the tin* (ex: <gallery>, <imagemap>)
- `wikitextToDOM`: use when none of the above meet your needs
  - **ParsoidExtensionAPI uses this internally for all the above 3 API methods.**
- May provide other flavours in the future

WIKIMEDIA FOUNDATION

To reiterate ...

- No control in Parsoid over how much parsing happens
  - `recursiveTagParse, internalParse` in current Parser API return "half-parsed HTML" whereas other methods return "fully-parsed HTML"
- wt2html options provide some semantic control
  - But, cannot turn on/off pipeline stages OR run stages selectively
- Dealing with special wikitext semantics
  - Mangle input as necessary (ex: `<poem>, <gallery>`)
  - Use DOM post processing as necessary (ex: `<poem>, <ref>`)

Mapping: Parser API

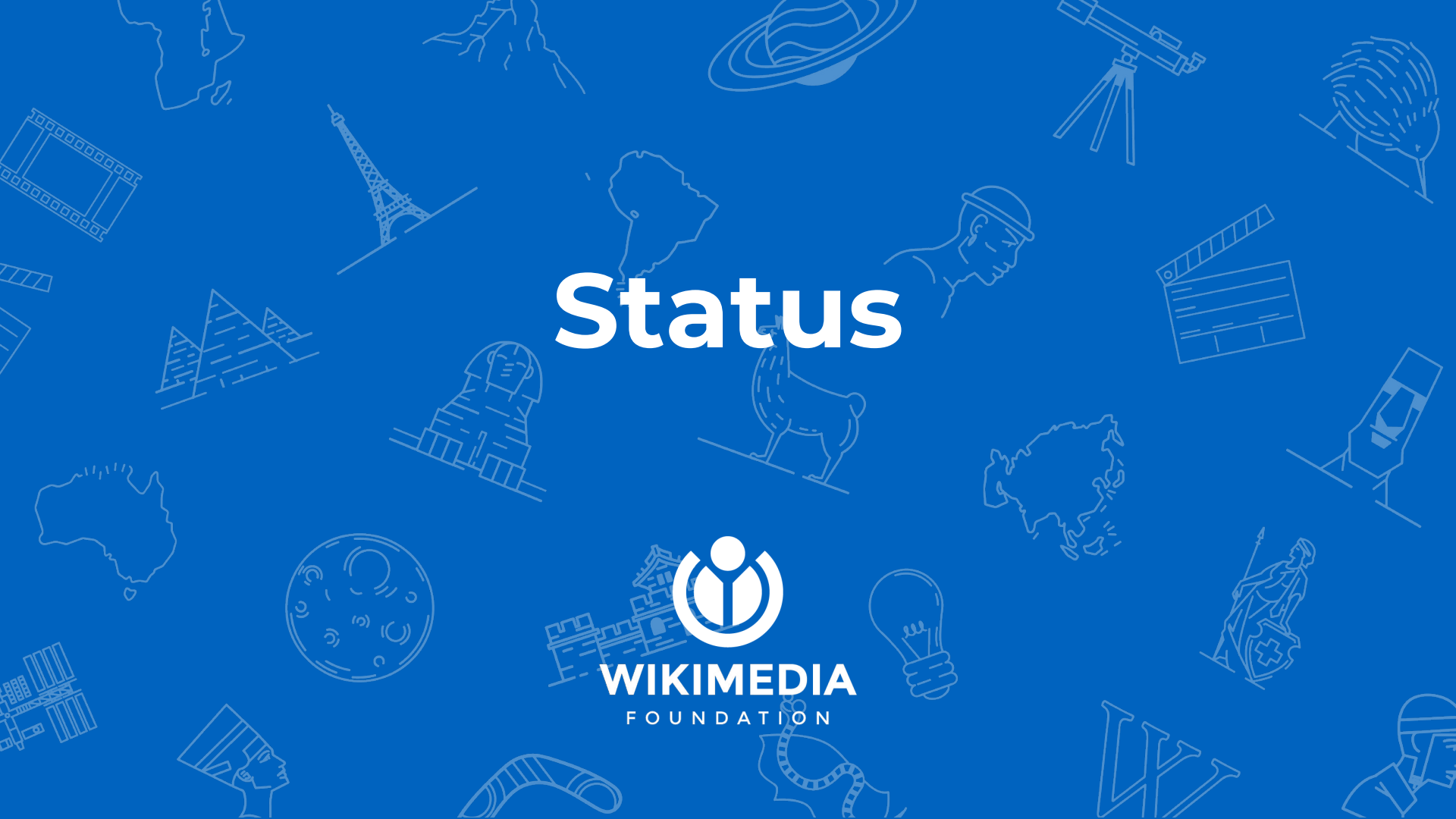
- Will expose `ParserOutput` object via the API
- Will expose `setFunctionHook` for declaring parser fns
  - Callback will get `ParsoidExtensionAPI`, not parser.
- Will augment `ParsoidExtensionAPI` with additional methods as necessary based on discovery and feedback

Watch mw:Parsoid/Extension_API for complete mapping between API methods

Mapping: Strip Markers

- Strip markers: used to tunnel output through parser stages
- Not needed in Parsoid
  - Extension output always tunneled through ⇒ output doesn't go through additional processing
  - StripState related hooks and methods don't exist in Parsoid
  - If found necessary, will introduce equivalent functionality

Status



Extensions

- Tag extensions:
  - In production: Cite, Poem, Gallery, Nowiki, Pre
  - In gerrit: ImageMap
  - Incomplete skeletons: LST, Translate
  - Next in line: `<indicator>`
- ContentHandler Extensions:
  - In production: JSON
- ParserTests extensions:
  - In use: RawHtml, StyleTag

WIKIMEDIA
FOUNDATION

Status: Hooks + API

- In late draft stage
  - As we discover unsupported uses, we will:
    - Add new hooks
    - Update ParsoidExtensionAPI with new functionality
  - Parser.php interface being narrowed
    - Make more methods private!
    - Deprecate lots of things!
- ParserTests & CI support will land in a couple weeks

WIKIMEDIA
FOUNDATION

Docs, next steps, ...

WIKIMEDIA
FOUNDATION

Next steps for us

- Will continue outreach and soliciting feedback
  - Presented early draft @ EMWCon in April 2020
  - Solicited feedback internally July 2020
  - This talk is next step in process
  - **Next:** wikitech-l, mediawiki-l, TechCom RFC

Will do our best to not break things unnecessarily

WIKIMEDIA
FOUNDATION

Next steps for you!

- Learn more, dive into the details, provide feedback
- Start updating your extensions now!
  - Best way to figure out what is missing, what is easy, what is hard

Help us migrate MediaWiki to Parsoid rendering for Wikimedia wikis next year!

WIKIMEDIA
FOUNDATION

Learn more: Look at code

- `Wikimedia\Parsoid\Ext:`
  - `ExtensionModule, ExtensionTagHandler, DOMProcessor, ParsoidExtensionAPI`
  - Helper classes
- `Wikimedia\Parsoid\Core:`
  - `DOMSourceRange`, various exception classes
- `Wikimedia\Parsoid\Utils\DOMCompat`
  - Work around broken PHP DOM API

WIKIMEDIA FOUNDATION

Learn more: docs, etc.

- https://mediawiki.org/wiki/Parsoid/Extension_API
  - **Discuss / leave questions on the Talk page**
- Look at Parsoid's implementations for Poem, Pre, Cite, etc.
- Look at Parsoid docs for the Ext/ namespace @ https://doc.wikimedia.org/Parsoid-PHP/master/
- Parsoid HTML spec @ https://www.mediawiki.org/wiki/Specs/HTML
- Find us at:
  - IRC: #mediawiki-parsoid
  - Email: parsing-team@wikimedia.org

WIKIMEDIA
FOUNDATION

Thanks!
Questions?

Backup slides

Extensions & Parser Tests

Parser Tests

- Add **`html/parsoid`** section w/ expected Parsoid output
  - Only needed if output differs
- ParserTests support multiple test modes per test
  - wt → HTML, wt → HTML → wt, HTML → wt, HTML → wt → HTML
  - Manual HTML edit tests (specify HTML edits, and expected wikitext)
  - Automated HTML edit tests
  - You can enable specific test modes per test
  - Use `{$ext}ParserTests-knownFailures.json` to track expected failures

WIKIMEDIA
FOUNDATION

Example

```
!! test
Poem with class
!! wikitext
<poem class="hiho">
hi ho
</poem>
!! html/php
<div class="poem hiho">
<p>hi ho
</p>
</div>
!! html/parsoid
<div class="poem hiho" typeof="mw:Extension/poem" about="#mwt3"
    data-mw='{"name":"poem","attrs":{"class":"hiho"},"body":{"extsrc":"\nhi ho\n"}}'><p>hi ho</p></div>
!! end
```

WIKIMEDIA
FOUNDATION

DOM Processor ordering

- How are DOM processors from multiple exts ordered?
  - Ordering problem not unique to Parsoid
  - Present wherever there are multiple listeners for the same event / hook that might operate on the same data
  - We have some ideas for Parsoid but nothing that is ready yet


<poem>
WIKIMEDIA
FOUNDATION

Pseudocode

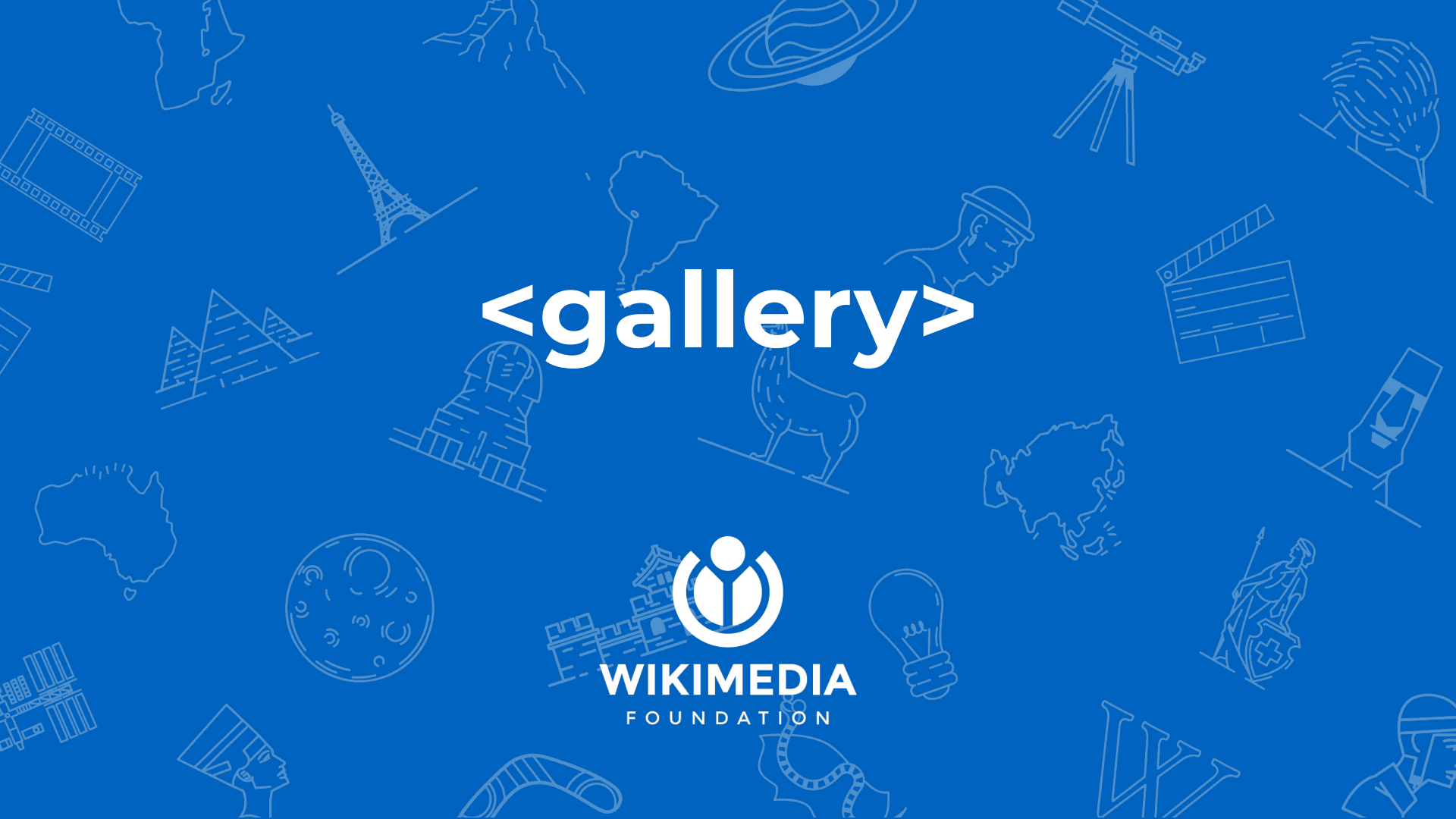
```
function sourceToDom(ParsoidExtensionAPI $api, $txt, $args): DOM* {
    $mTxt = $this->mangle($txt); // process :, newlines, ----, nowikis
    return $api->extTagToDom($args, $mTxt, [
        'wrapperTag' => 'div', // DOM is wrapped in <div> tag
        'parseOpts' => [ 'extTag' => 'poem' ],
        'processInNewFrame' => true, // mangled $mTxt is different from $txt
        'clearDSROffsets' => true    // mangled $mTxt => DSRoffsets incorrect
    ]);
}
```

WIKIMEDIA FOUNDATION

More ...

- Poem extension treats `<nowiki>` blocks differently
  - Unlike normal wikitext, newlines inside becomes `<br>`s
  - Changing newlines to `<br>` in `mangle(..)` won't work because `<nowiki>` will escape them!
  - Poem extension registers a DOM processor to deal with this
- Processor finds the `<nowiki>`s and fixes newlines
  - `typeof="mw:Extension/$extName"` attr. present on ext. wrappers
  - Processor looks for matching `typeof` to identify nowiki blocks
  - Replaces newlines inside them with `<br>` tags

WIKIMEDIA
FOUNDATION


<gallery>
WIKIMEDIA
FOUNDATION

Pseudocode

```
$doc = … ; // construct gallery scaffolding
foreach ($line in $txt) {
    $mLine = makeImageWikitext($line); // [[File:..|..|..]]
    $imgDOM = $api->wikitextToDom($mLine, [
        'parseOpts' => [ 'extTag' => 'poem', 'inlineContext' => true ],
        'processInNewFrame' => true,
        'shiftDSRFn' => function($dsr) { return updated $dsr; }
    ]);
    … Process $imgDOM and add to $doc …
}
```

WIKIMEDIA FOUNDATION