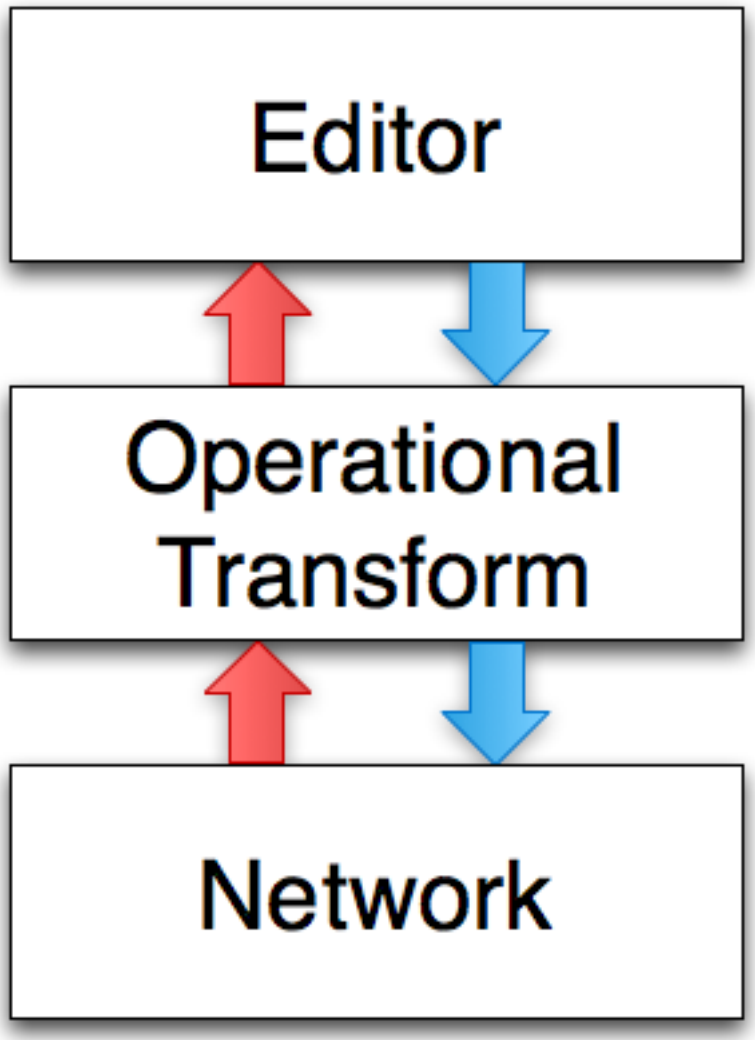
Wave Editor & Document Rendering

Daniel Danilatos, David Hearnden

Google

Context



Google

Design Goals

A toolkit for rich, extensible, real-time collaborative editing.

Enable:

- Edit rich, arbitrary document model
- Render to decent, semantic HTML

Provide:

- Stable editing core, abstracting browser quirks
- API to build composable editing components ("doodads")

Google

Design Goals

"The Editor" =

Editor core +

Document model +

Bundled feature A + Custom feature B +
Bundled feature C + Extended feature D etc.

Design Philosophy

Content-Editable: use or not?

- Hybrid approach

Careful balance

- Leverage what the browser can give us with contentEditable
  - Fast typing
  - Selection support
  - Clipboard
  - IME
- Explicitly handle events where it doesn't do what we want

"contentEditable on a leash"

Rendering the Document Model

Document Model

XML + Annotations

- Simple XML for structure
- Annotations for style and metadata

```
<body>
  <line/>Hey have you seen this website?
  <line/>It's sweet!
</body>
```

Image Thumbnail Example

```
<image attachment="...">
  <caption>
    A Thumbnail
  </caption>
</image>
```

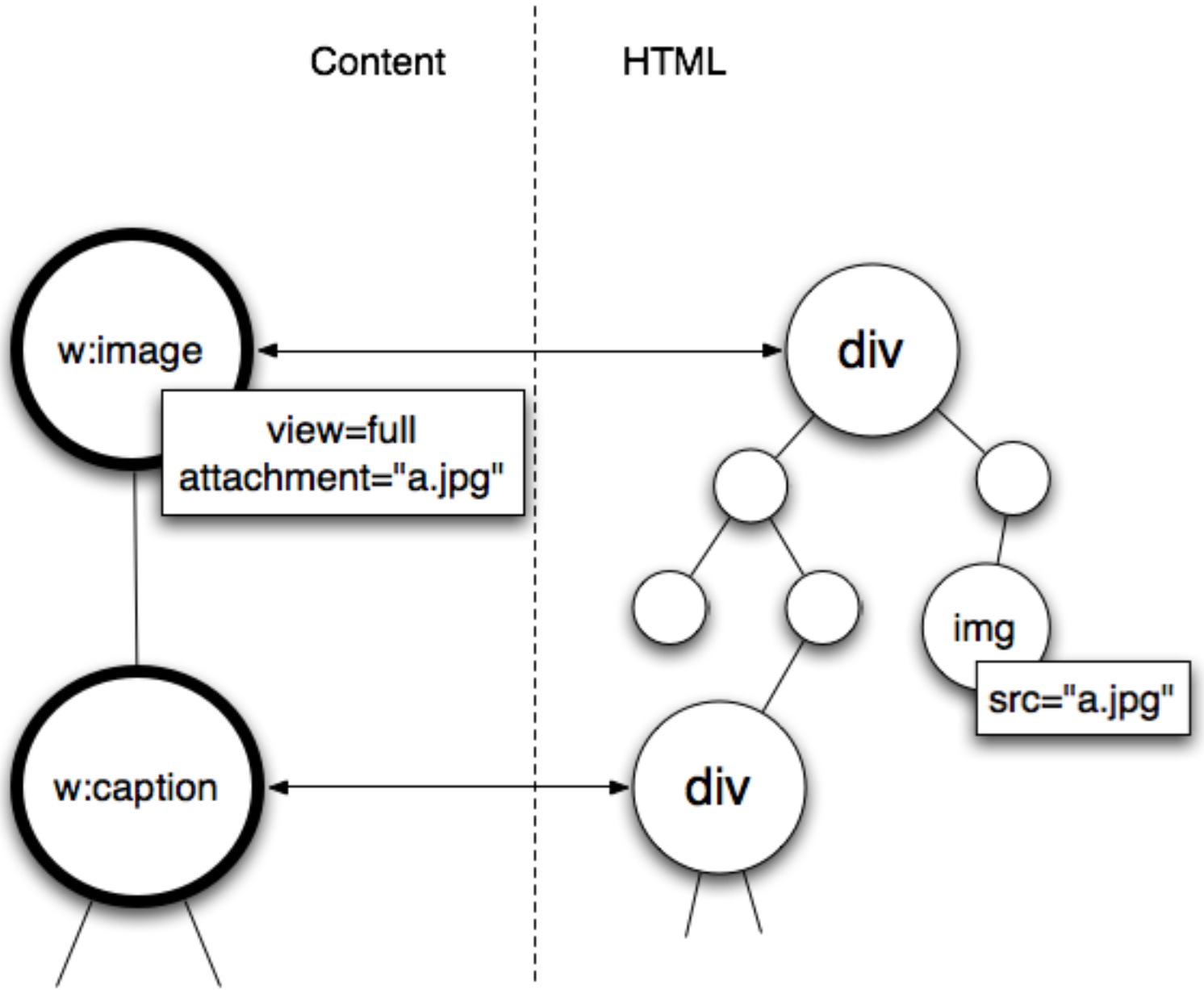
A Thumbnail

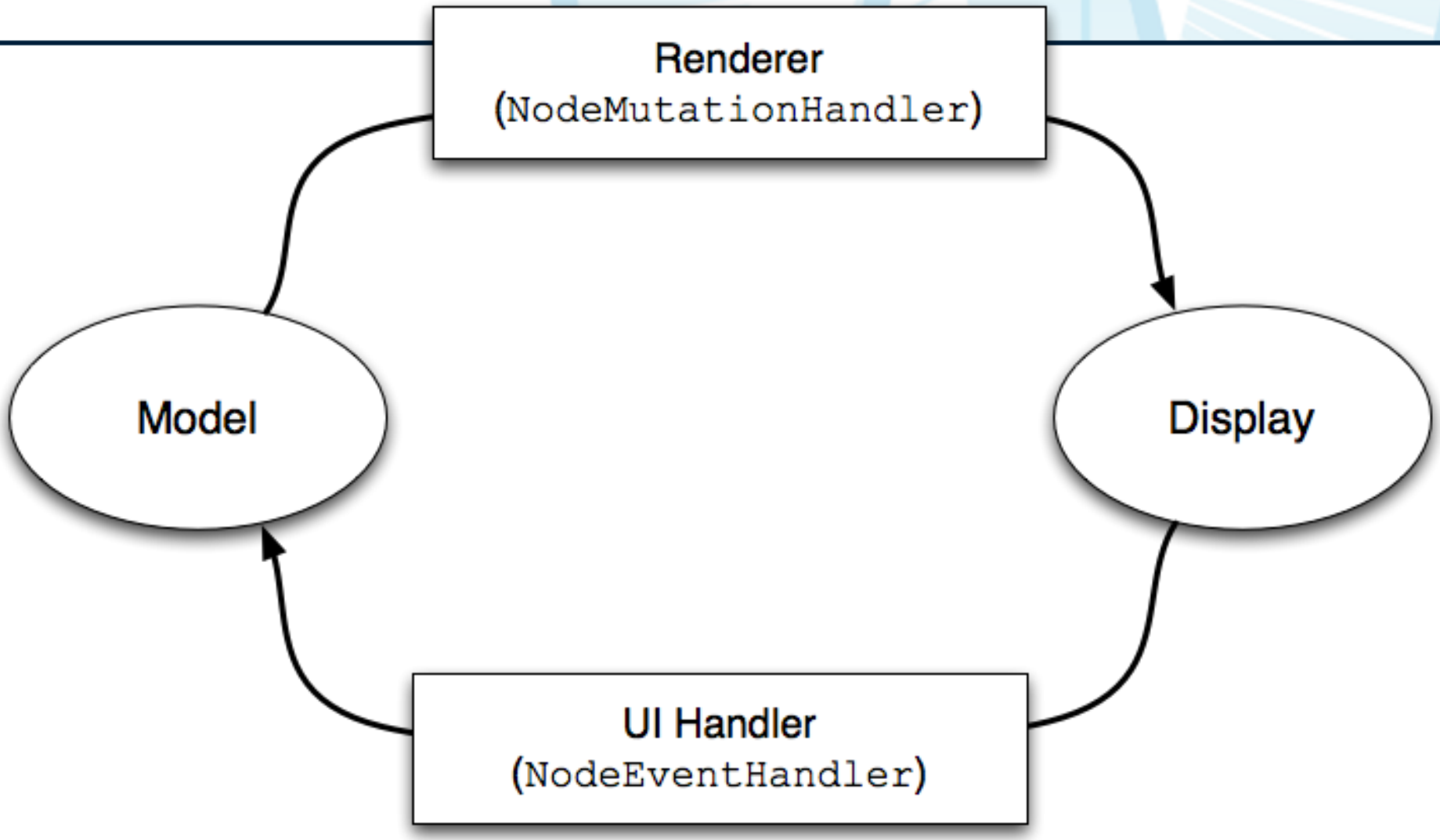
Google

Model XML to Rendered HTML

```
<image
attachment="...">
  <caption>
    A Thumbnail
  </caption>
</image>
```

```
<table unselectable='on' cellpadding='0' class='ittt' cellspacing='0'>
 <tbody>
  <tr unselectable='on'>
   <td unselectable='on' style='visibility: visible;' class='itco'>
    <div unselectable='on' style='width: 120px; height: 79px;' class='itci'>
     <a href='...' target='_blank'>
      <img style='width: 120px; height: 79px;' src='...' class='gwt-Image J' />
     </a>
     <div style='display: none;' class='itcc' />
     <div style='display: none;' class='pw'>
      <div style='width: 0%;' class='pwa' />
      <div style='width: 100%;' class='pwg' />
     </div>
    </div>
   </td>
  </tr>
  <tr unselectable='on'>
   <td unselectable='on' style='visibility: visible;'>
    <div contenteditable='true'>A Thumbnail<br /></div>
   </td>
  </tr>
 </tbody>
</table>
```


Content
HTML
w:image
view=full
attachment="a.jpg"
w:caption
div
img
src="a.jpg"
div


Renderer
(NodeMutationHandler)
Model
Display
UI Handler
(NodeEventHandler)

More complicated mappings

E.g. lines -> paragraphs

- We want "line tokens" in our model for OT reasons, but prefer , <li>, etc tags in our HTML rendering.

XML:

<line></line>hello world

HTML:

hello world

Concept: "Local Nodes"

- Additional nodes part of the document data structure
- Take up zero size
- Operations apply cleanly around them
  - Implementation that deals with conflicts: PersistentContent

Example: lines to paragraphs

```
<line/>
<l:p>
  abcdef
</l:p>
<line/>
<l:p>
  ghijkl
</l:p>
```

```
  -
<p>
  abcdef
</p>
  -
<p>
  ghijkl
</p>
```

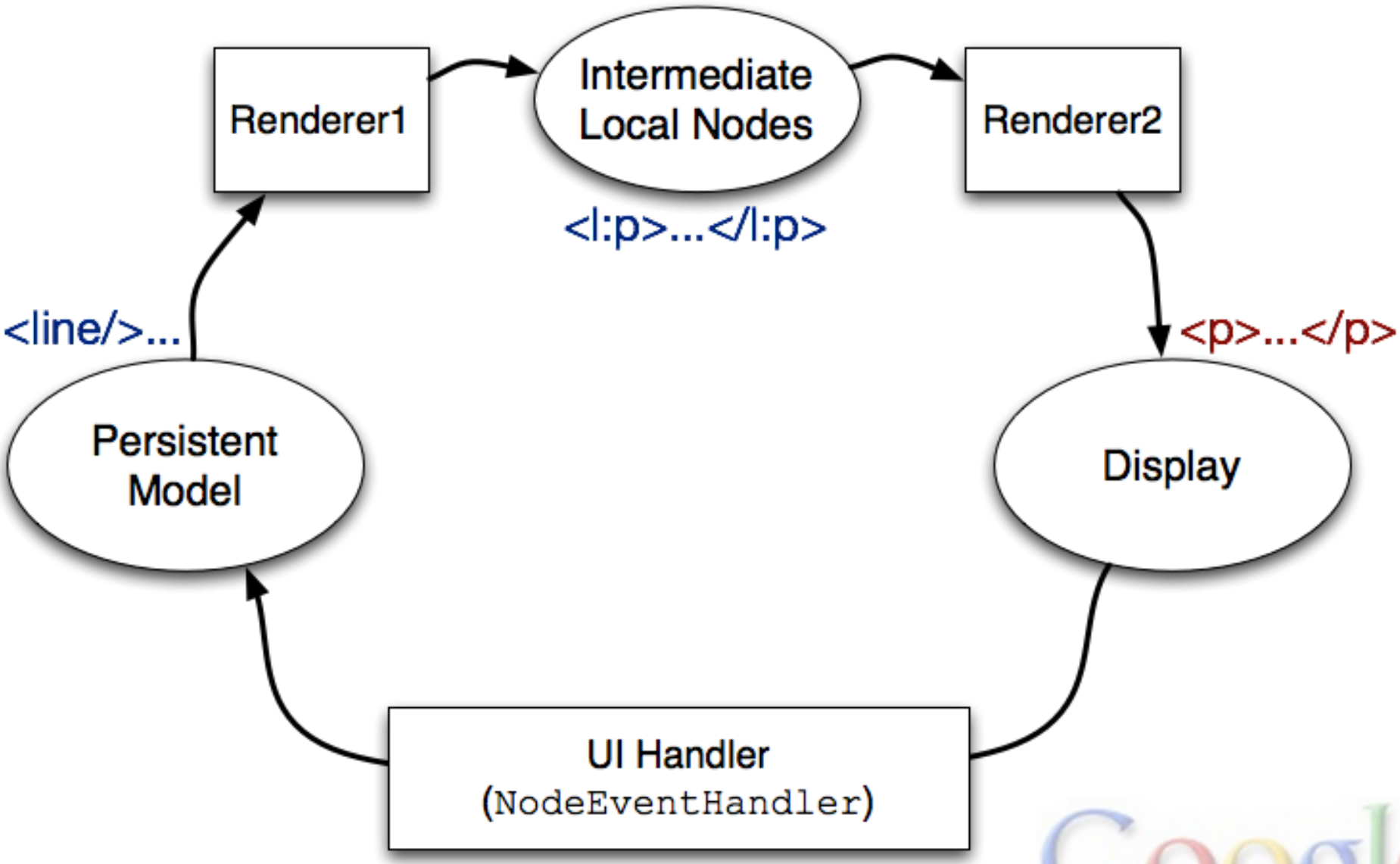
Local Nodes - uses

- Intermediate representation to make rendering convenient
- Miscellaneous book keeping

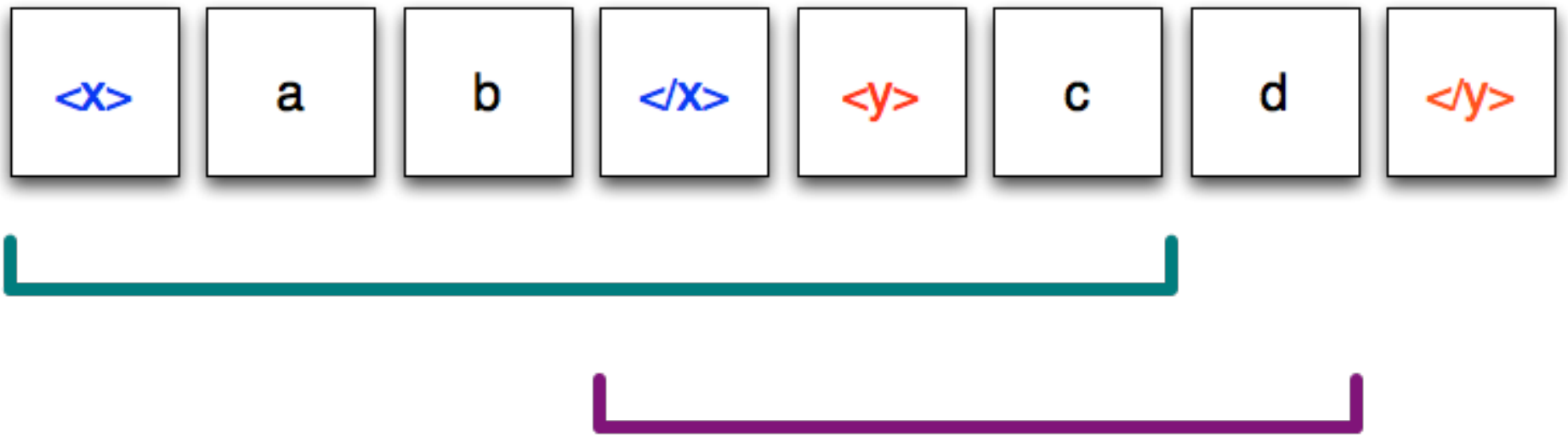
Local Nodes - accessing

Expose different "views" of the document, e.g.

- "Persistent view" - just the content the server would see
- "Full view" - everything, including extra nodes
- (Custom views exist for more specific purposes)


Renderer1
Intermediate
Local Nodes
Renderer2
<l:p>...</l:p>
<line/>...
...
Persistent
Model
Display
UI Handler
(NodeEventHandler)

Annotations



Properties of Annotations

- Don't affect the structural content or its operations
- Clients may choose to ignore some or all annotations

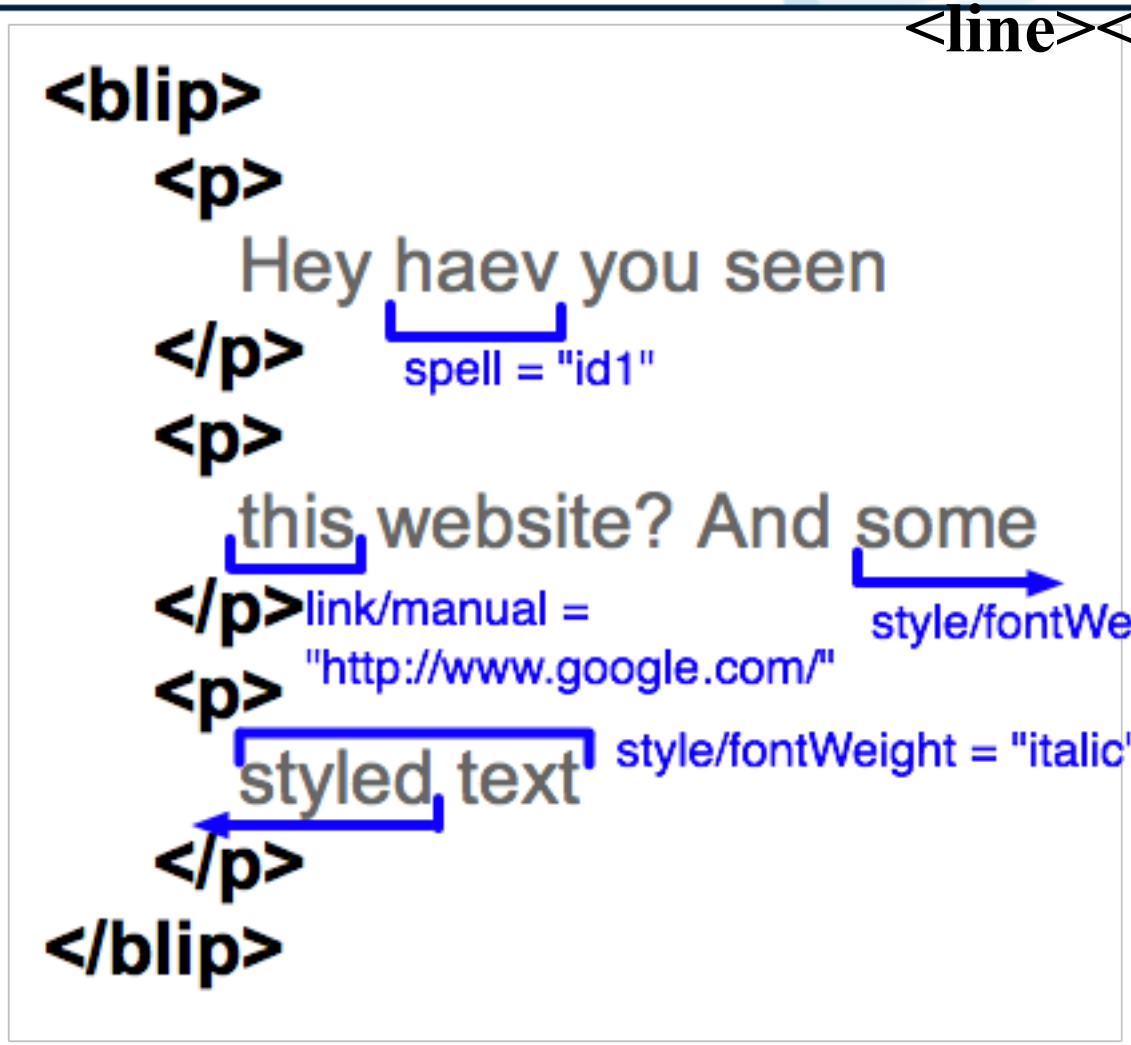
Some uses for Annotations

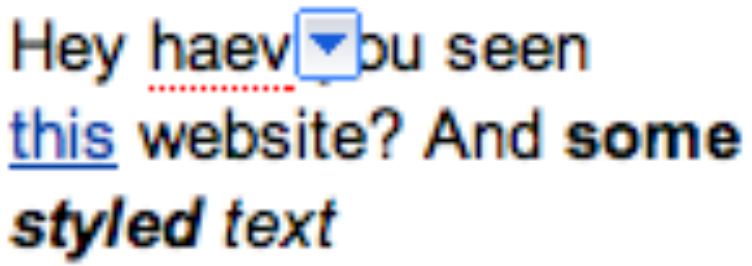
- Other users' cursor position, selection
- Rich link annotations
- Robot-specific data
- Diff highlighting (using local annotations)
- May reference structured data in another document
  - E.g. spelling suggestions and state

Rendering annotations

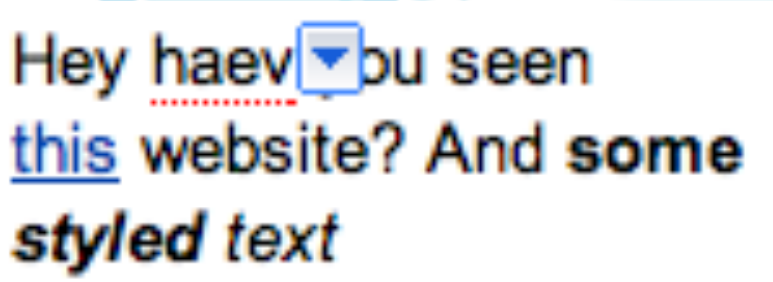
How convenient, we can use local nodes again!

NOTE: this is an old image,
** ...stuff... **
should be
<line></line>...stuff...





l:s - "spread"
l:xyz - "boundary"

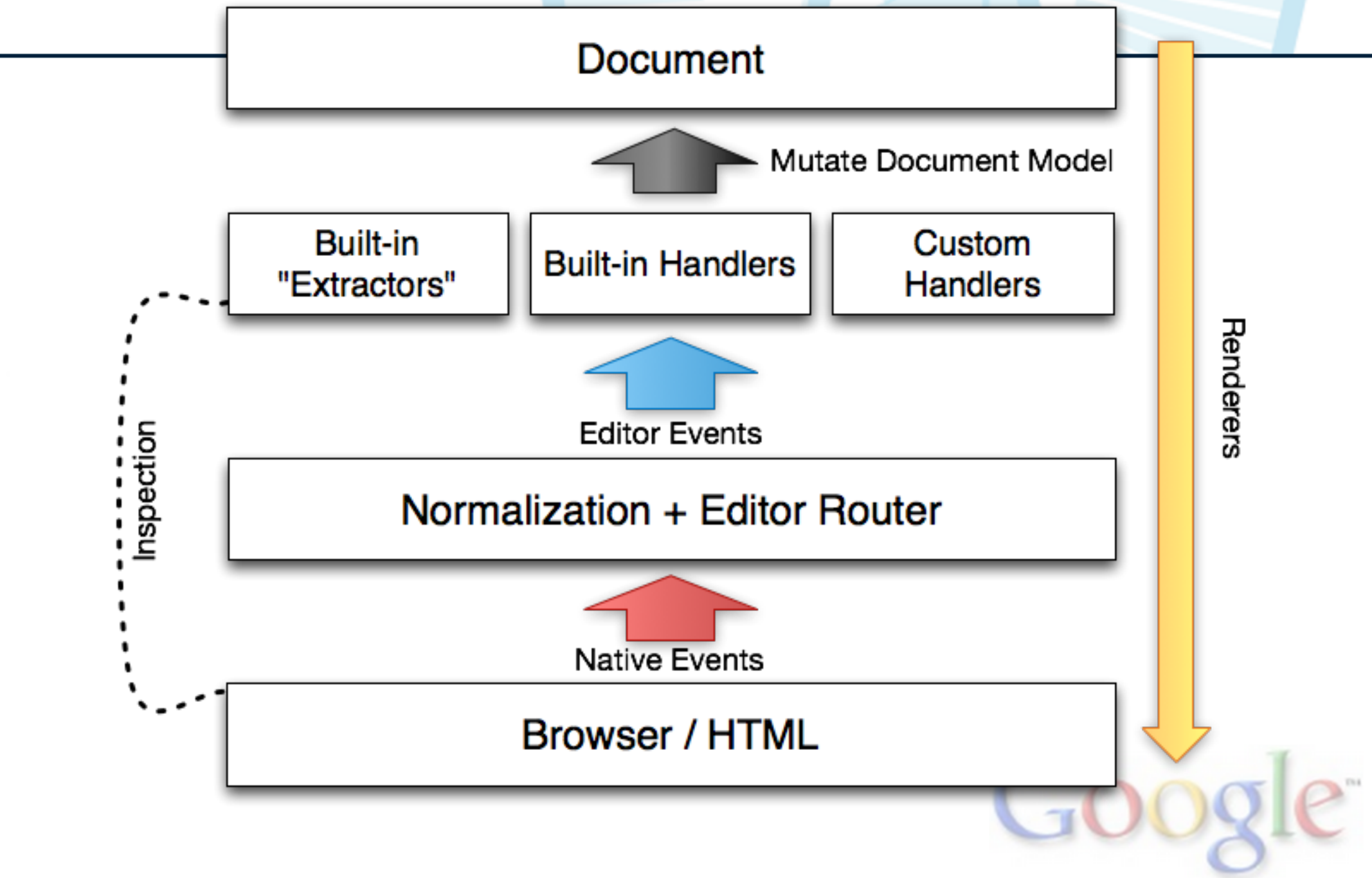



```
<body>
    <line/>
    <l:p>
      Hey <l:s hover="...">haev</l:s><l:spell/> you seen
    </l:p>
    <line/>
    <l:p>
      <l:s href="...">this</l:s> website?
      And <l:s fontWeight="bold">some</l:s>
    </l:p>
    <line/>
    <l:p>
      <l:s fontWeight="bold" fontStyle="italic">styled</l:s>
      <l:s fontStyle="italic">text</l:p>
</l:p>
</body>
```

Local annotations

- Similar to local nodes, but for annotations
- Behave just like regular annotations, but are not persisted
- Can have any `Object` as a value, not just `String`
- Uses
  - Diff highlighting
  - Very handy for book keeping

Event Handling


Document
Mutate Document Model
Built-in "Extractors"
Built-in Handlers
Custom Handlers
Editor Events
Normalization + Editor Router
Native Events
Browser / HTML
Inspection
Renderers

Event Layers

- **Normalization Layer**, converts "Event" into "Signal"
- **Routing Layer**, routes to:
  - *Typing* Extractor
  - *IME* Composition handler
  - *Clipboard* handler
  - *Key combo* registry
  - Custom node event handlers:
- **Custom Event Handlers**
  - Provide most of the high level functionality
  - Even things like the behavior of pressing enter in a paragraph, or backspace on bullet points, handled here.

Example: Typing

Goal: fast typing

- Pure-typing events routed to TypingExtractor
- Schedules a timeout, browser permitted to modify HTML
  - Multiple key strokes get caught at a time (this is a performance boost).
- Timeout returns, difference between HTML and XML intelligently calculated and extracted as an operation

- Note: Typing extractor is "flushed" before the timeout e.g. for network events, or non-typing user events

Example: Pressing Enter

- Event routed to custom logic registered for the "l:p" element
- Logic modifies the XML by inserting a new <line/> element
- As a result, first level renderer splits the <l:p> into two
- Second level rendering updates the HTML

- (More complicated logic exists for behavior with bullet points, headings, etc. The point is, it's application code, not the browser, that defines behavior).

Example: Paste

- "onpaste" event routed to clipboard handler
- Focus placed in hidden, editable div
- Timeout scheduled, paste permitted to happen there
- Pasted HTML converted to wave XML and inserted into old selection location, old selection restored
  - There is also a trick to allow pasting wave content into wave content, that preserves the original XML+annotations (rather than lossy conversion into HTML).

Example: Ctrl+B

- Editor KeyCombo event routed to handler registered in key combo registry
- "style/fontWeight=bold" applied or unapplied from model
  - (utilities exist to make this convenient)
- Annotation Painter applies registered PaintFunction for style/Fontweight to render to local nodes

Example: IME

- "oncompositionstart" event trapped
- special "extractor" created, cursor placed within
- IME composition happens inside span, fragile IME state safe from concurrent modifications to surrounding text
- "oncompositionend" event trapped, resulting text applied to model, extractor span removed

Q/A

Google

Caveats / TODOs

Some parts not yet fully customizable

- Clipboard behavior (interpreting pasted HTML)
- Undo granularity