Project 2: Persistence

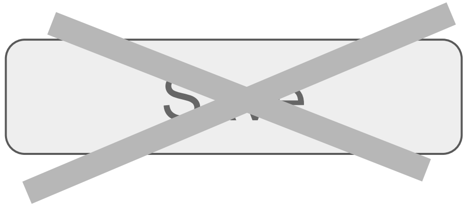
Software Design WiSe 15/16

Jan Lindemann, Daniel Stolpe

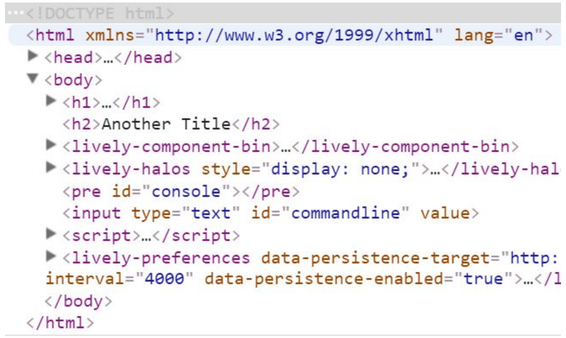
02.02.2016

Motivation

- **Auto-save**: Get rid of the save button
  - like Google Docs / OneNote / Etherpad

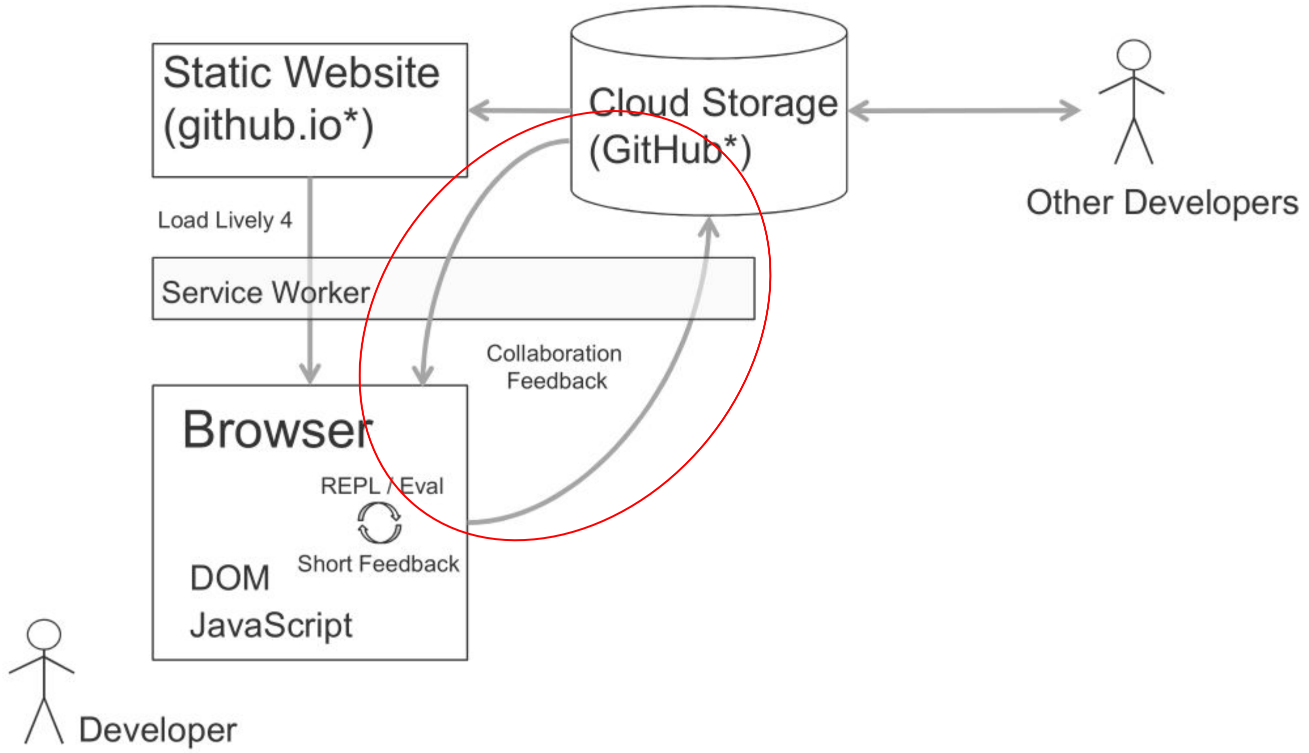


- **Readability**: User data shall be stored as HTML5

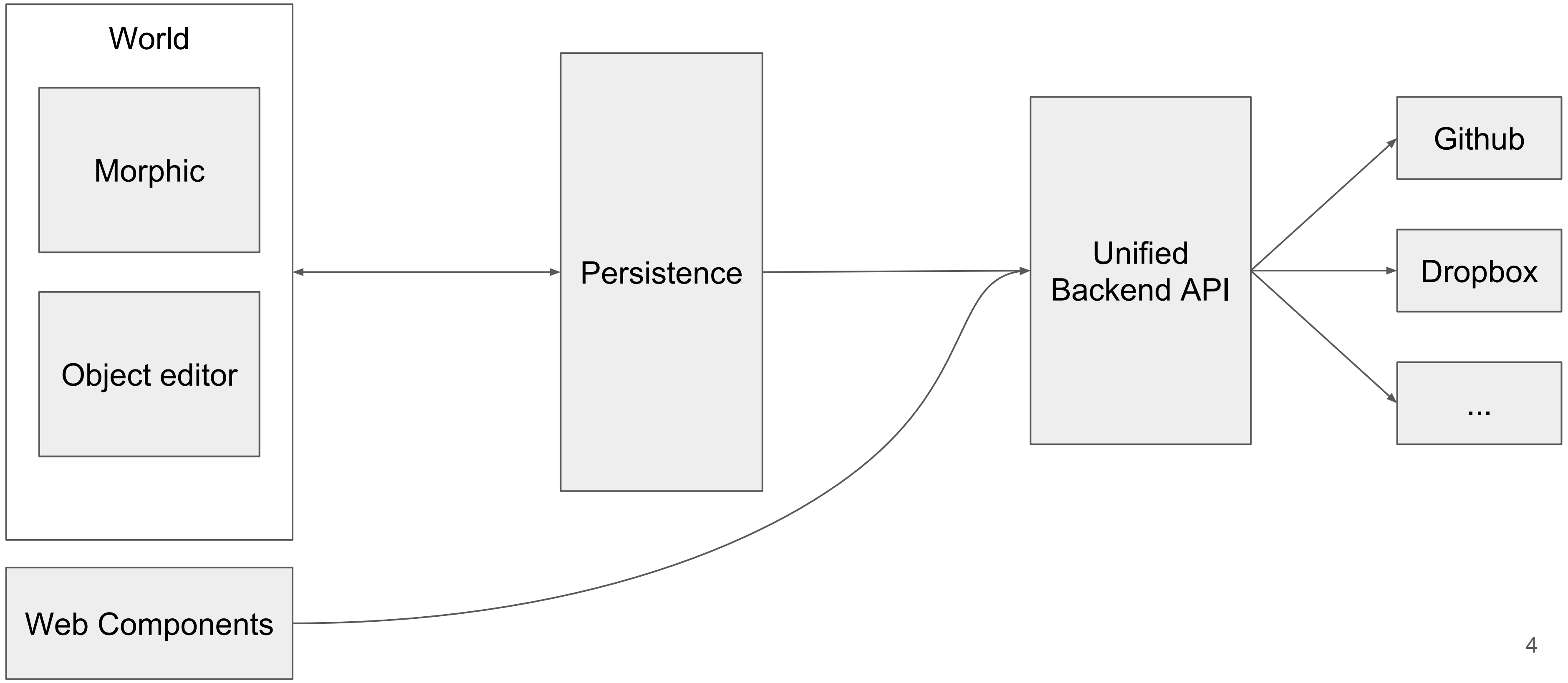


```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
 ▶<head>…</head>
 ▼<body>
   ▶<h1>…</h1>
    <h2>Another Title</h2>
   ▶<lively-component-bin>…</lively-component-bin>
   ▶<lively-halos style="display: none;">…</lively-hal
    <pre id="console"></pre>
    <input type="text" id="commandline" value>
   ▶<script>…</script>
   ▶<lively-preferences data-persistence-target="http:
  interval="4000" data-persistence-enabled="true">…</l
  </body>
</html>
```

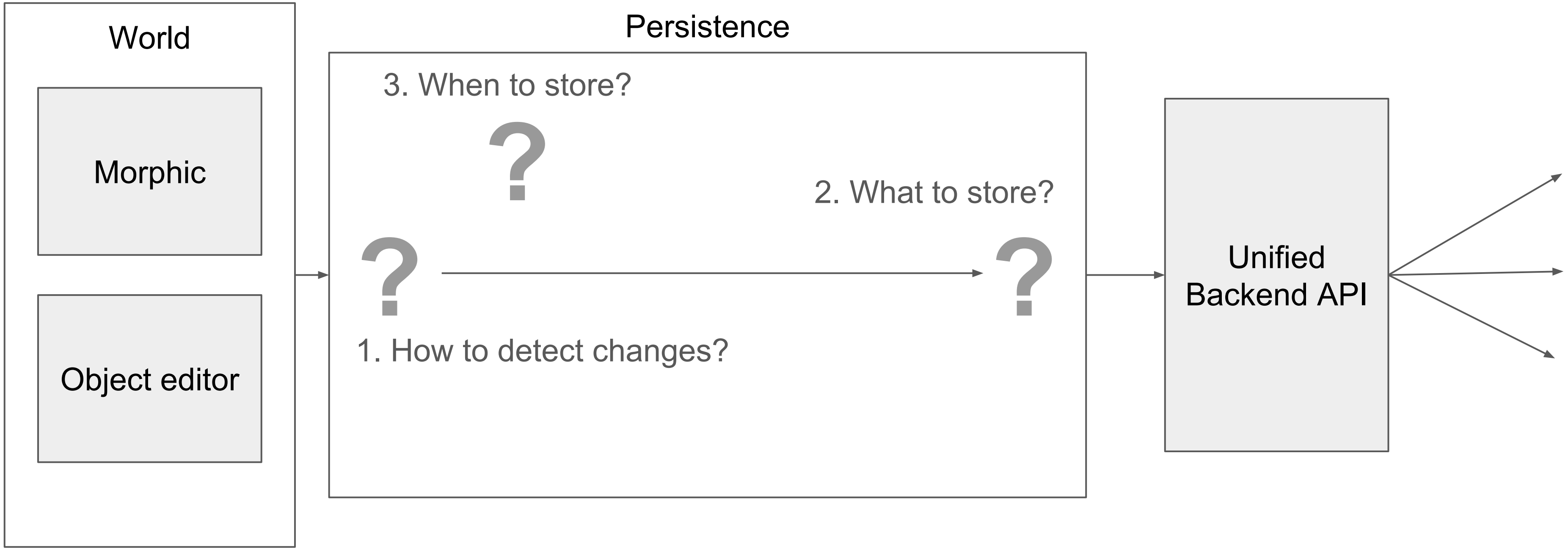
Background


Static Website
(github.io*)
Cloud Storage
(GitHub*)
Other Developers
Load Lively 4
Service Worker
Collaboration
Feedback
Browser
REPL / Eval
Short Feedback
DOM
JavaScript
Developer

Background


World
Morphic
Object editor
Persistence
Unified Backend API
Github
Dropbox
...
Web Components

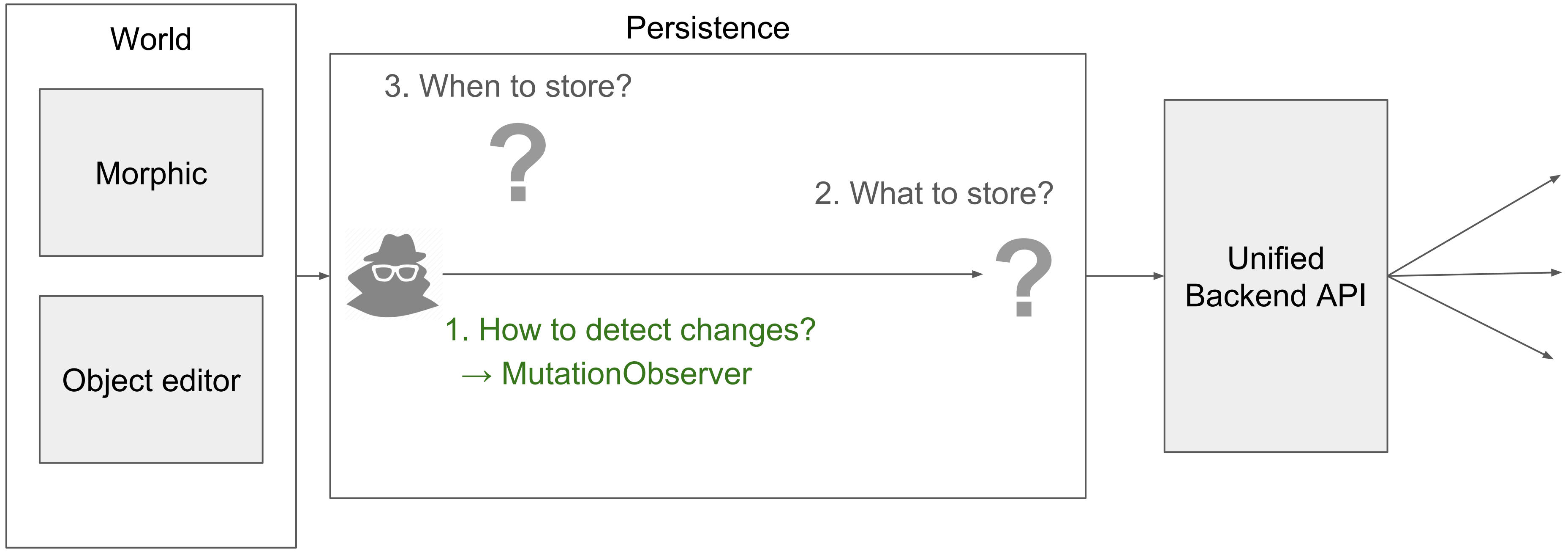
Central questions



1. How to detect changes?

Direct (API)	Mutation Observer
+ Less complex	+ No action required from objects
- No solution for non-lively4 objects	+ Already exists
- Error-prone	+ Low coupling
- High coupling	- Processes every DOM mutation
	- Decision, what has to be saved

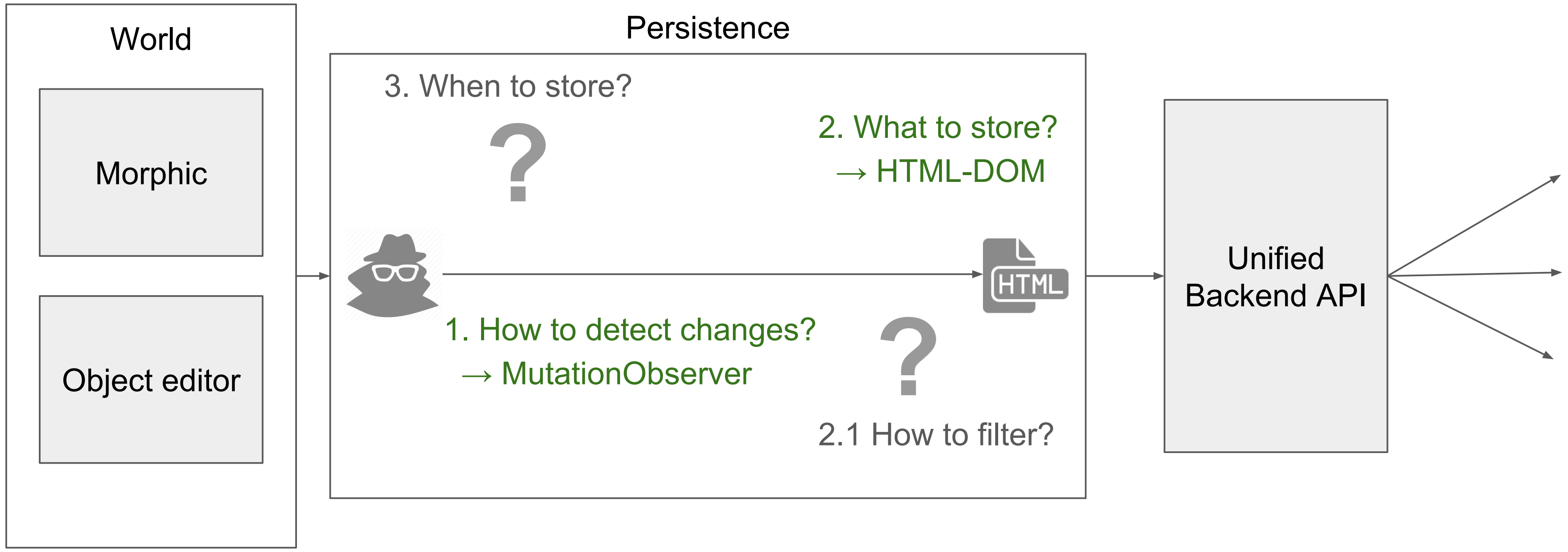
Central questions



2. What to store?

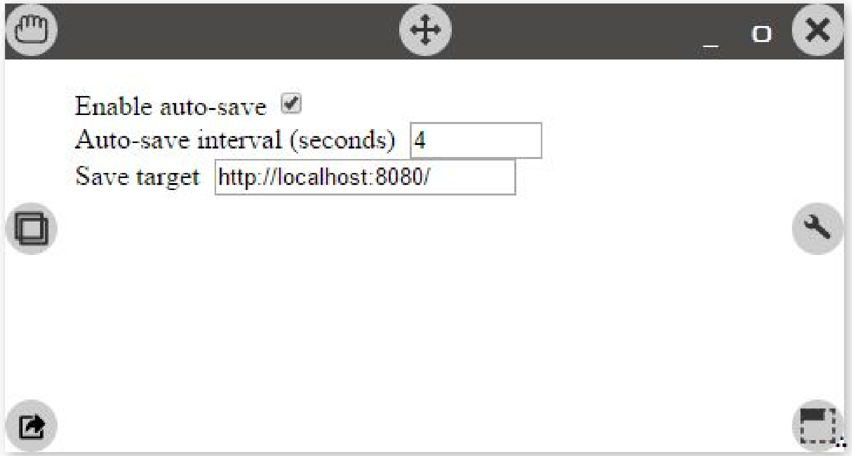
“History of changes”	Complete HTML-DOM
+ Only store changes + Undo/redo-implementation storage independent + Low network load - Not easily readable (reconstruction needed) - Git already does that - Server component neccessary	+ Readability + Fast access (no reconstruction) + Simplest thing that works - High network load - Undo/redo-implementation depends on support by storage

Central questions

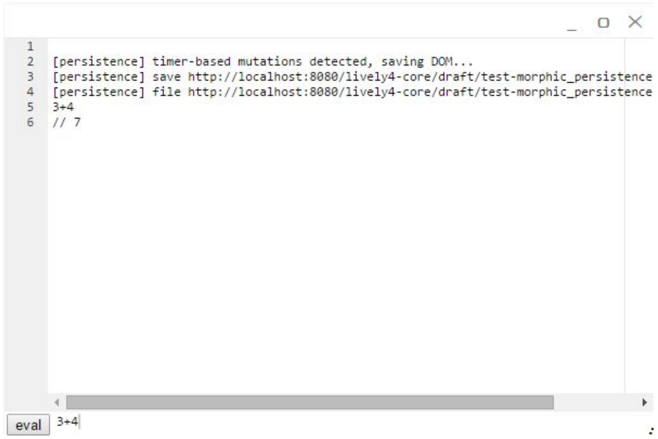


2.1 How to filter transient from persistent changes?

- Problem: We don’t want to store everything
- Persistent vs. transient Data
  - Halos
  - Console log messages



Enable auto-save ✓
Auto-save interval (seconds) 4
Save target http://localhost:8080/



```
[persistence] timer-based mutations detected, saving DOM...
[persistence] save http://localhost:8080/lively4-core/draft/test-morphic_persistence
[persistence] file http://localhost:8080/lively4-core/draft/test-morphic_persistence
3+4
// 7
```

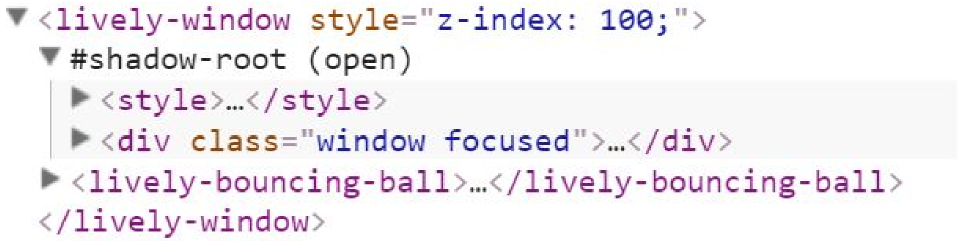
eval 3+4

2.1 How to filter transient from persistent changes?

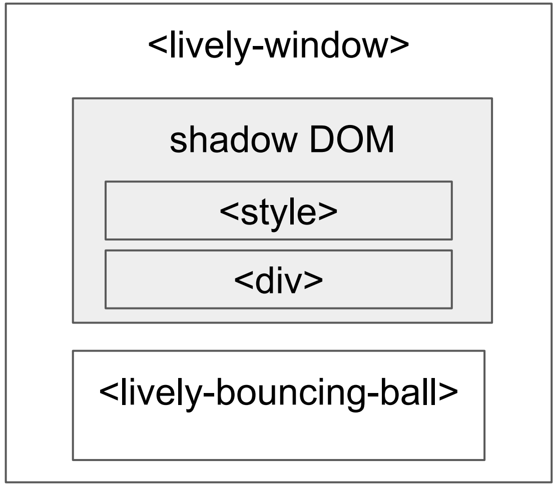
Blacklisting	Diff source HTML with loaded HTML-DOM	Web Components
+ Easy to implement - Needs maintenance	+ No maintenance - Difficult to say when page is “loaded”	+ Easy to use + Encapsulation - responsibility moved to developer

Web Components encapsulation

- Web Components
  - allow custom DOM elements with a shadow DOM



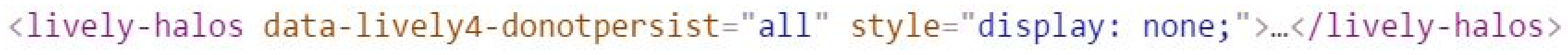
```
▼<lively-window style="z-index: 100;">
  ▼#shadow-root (open)
    ▶<style>…</style>
    ▶<div class="window focused">…</div>
  ▶<lively-bouncing-ball>…</lively-bouncing-ball>
 </lively-window>
```



➔ You want to persist information ? Put it in the DOM : Hide it in shadow DOM

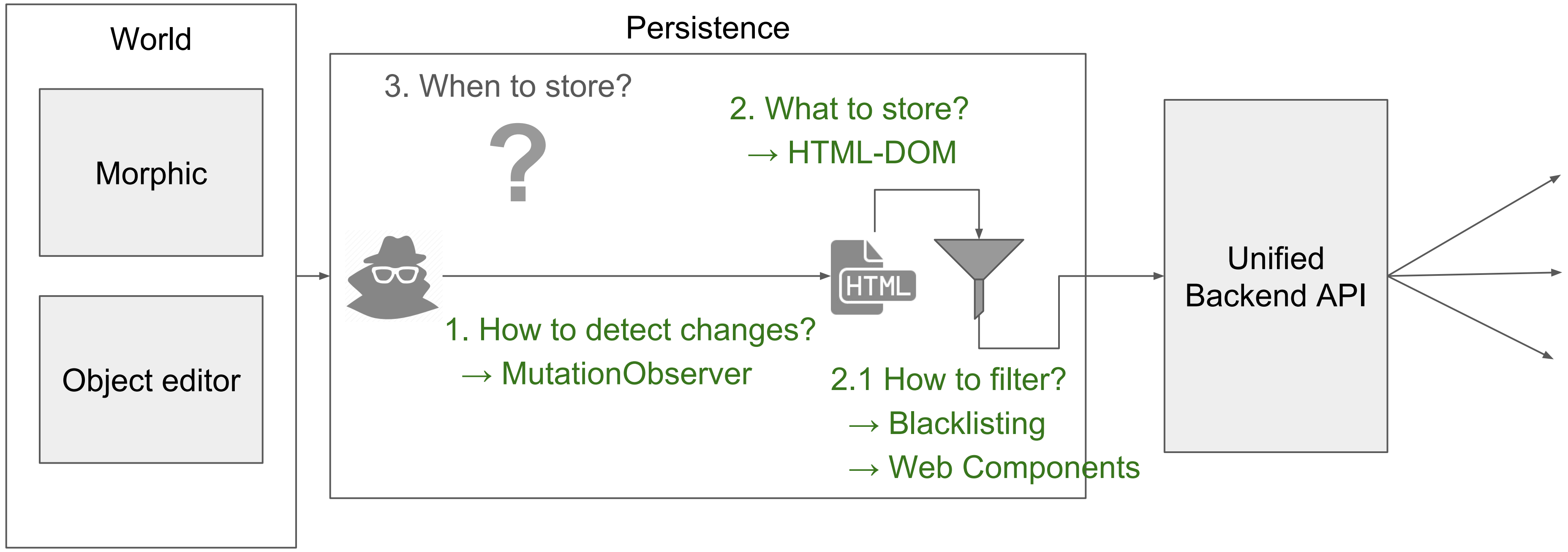
Blacklisting

- Tagging via “donotpersist”-flag
  - HTML5 allows custom attributes with prefix data-*



```
<lively-halos data-lively4-donotpersist="all" style="display: none;">…</lively-halos>
```

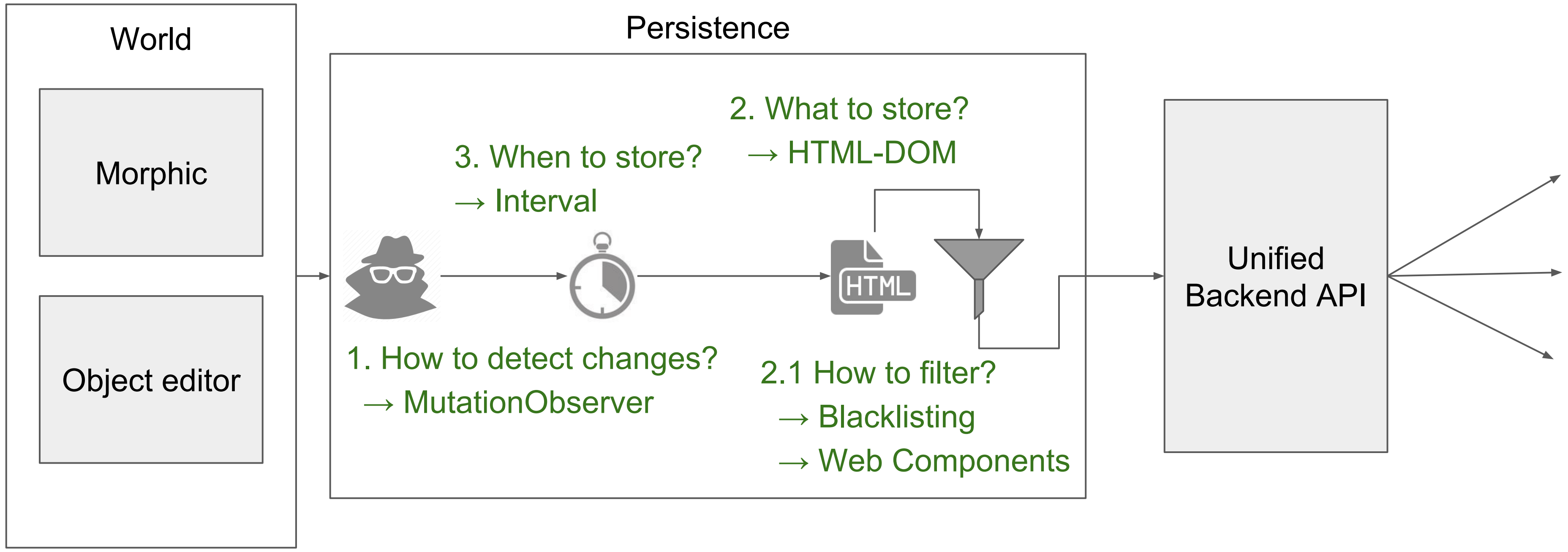
Central questions



3. When to store?

Explicit Save	On Every Change	Interval
+ Lowest network load - Manual	+ Automatic + Fault safety + Keeps semantics - High network load - Limitations of storage providers	+ Automatic + Reduced network load - Potential loss of last data - Current implementation looses semantics

Central questions



Problem: Saving triggers saving (Meta-circularity)

Discipline	Context switch	lively4-API	Custom HTML Serializer
+ easy for us - error-prone	+ easiest technical solution - explicit checks for context	+ highest control - needs discipline - high effort	+ no side-effects - high effort - many edge-cases

Demo

Limitations / Future Work

- Write custom HTML serializer instead of “DOM.clone() + DOM-clean-up + standard serialization”
- Automatic and manual changes not distinguishable → can defer saving to infinity by always resetting interval
- Impossible/hard to automatically decide whether an element should be saved or not by algorithm
- Undo/redo (git-dependent or -independent)
- Trigger event before save

Conclusion

Persistency:

- Readability ✓
- Get rid of save button ✓
- Network efficiency ×

DOM storage concept:

- Encapsulation ✓
- JavaScript object changes ×
- Easy to understand for HTML-aware user ✓