

Timelapse

Interactive record/replay for web apps

Brian Burg

Richard J.
Bailey

Andrew J. Ko

Michael D. Ernst

Computer Science and Engineering
University of Washington

“It’s hard to debug failures in the field”

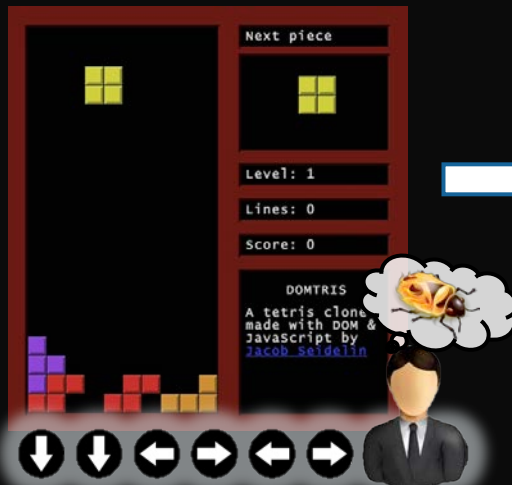


Distributed across time and space

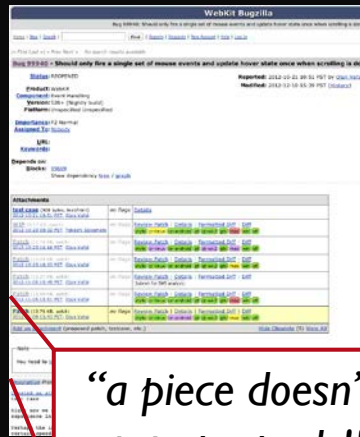
Hardware and software variation

Users are not software testers

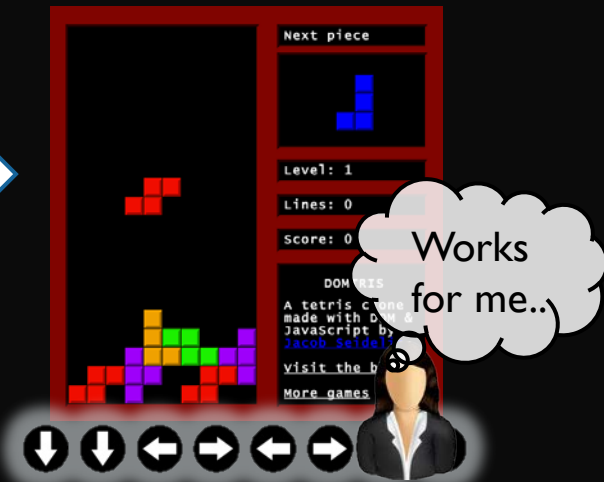
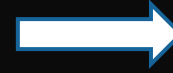
Users can't report failures accurately



end-user encounters
bug in production
code

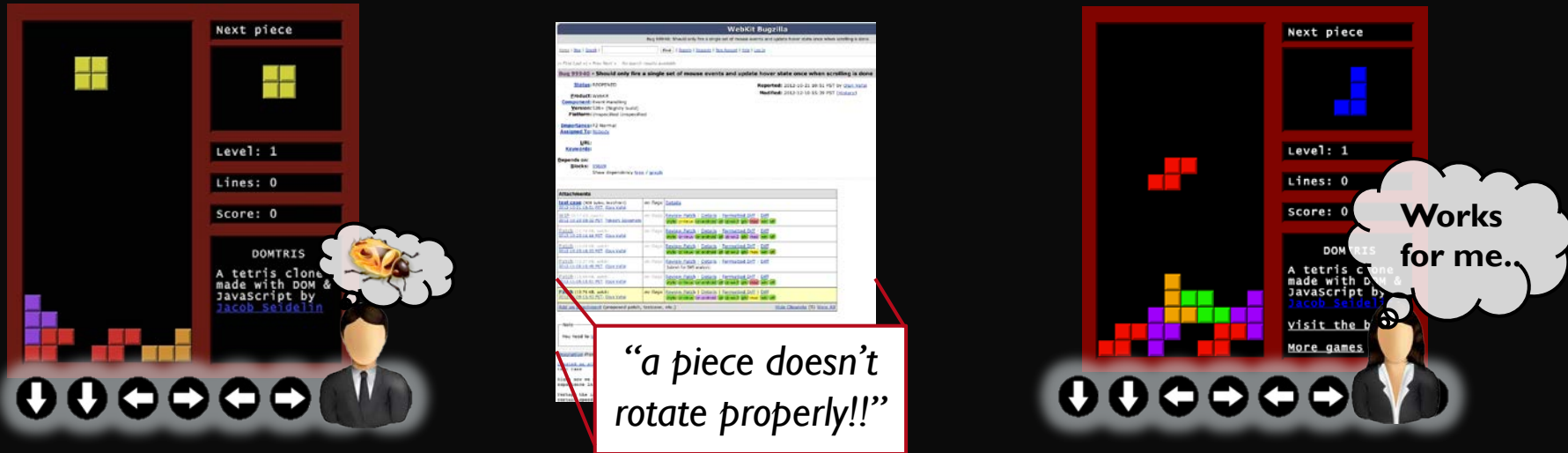


files bug report with
ad-hoc information



developer unable to
reproduce the bug

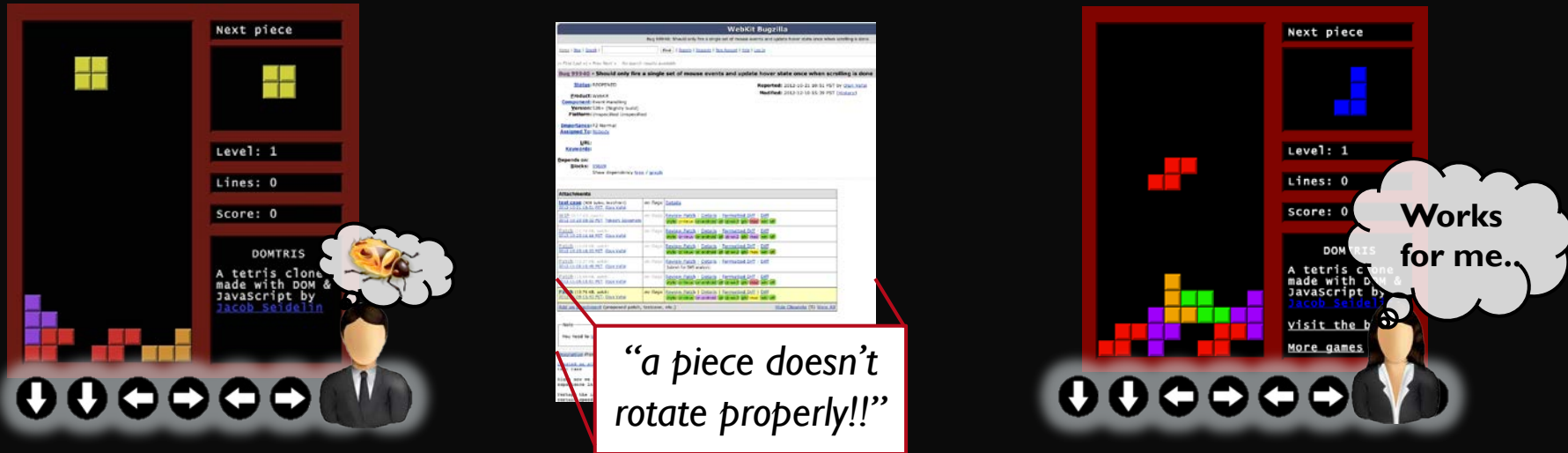
Users can't report failures accurately



“The most severe problems were errors in steps to reproduce and incomplete information.”

“What makes a good bug report”. Zimmerman et al. TSE Vol. 36, No. 5, 2010

Users can't report failures accurately

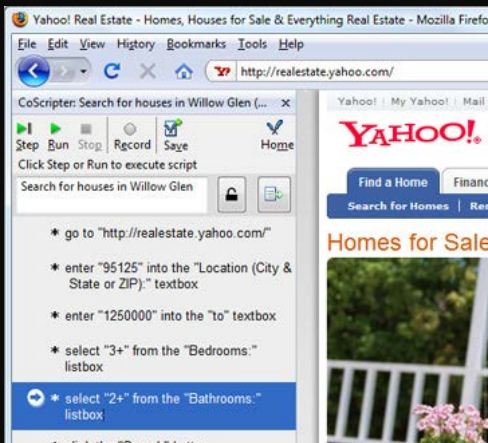


Bug reporters and developers want better tool support for reproducing buggy behavior.

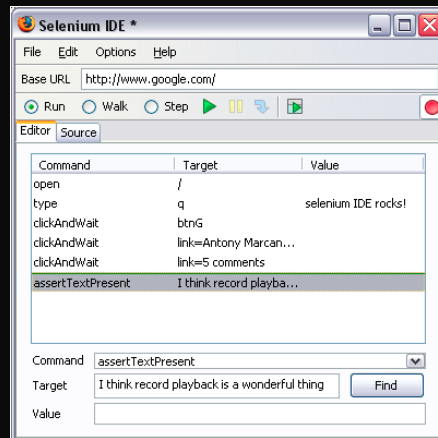
Existing tools are imprecise and hard to use

macro replay
(CoScripter, Selenium, Sikuli)

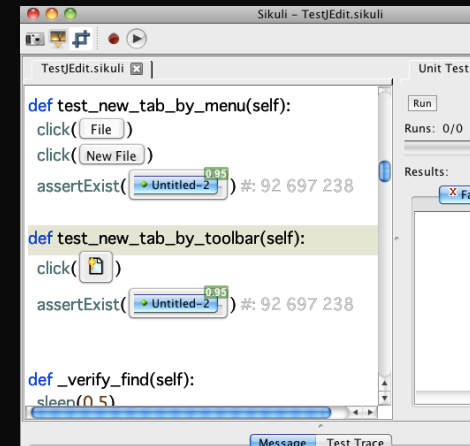
Capture and simulate user input.
Designed for test and task automation.



CoScripter
Leshed et al, CHI 2008



**Selenium/
WebDriver**



Sikuli Script
Yeh et al, UIST 2009

Existing tools are imprecise and hard to use

macro replay

(CoScripter, Selenium, Sikuli)

Capture and simulate user input. Designed for test and task automation.

Nondeterministic. Requires extra setup ahead of time. Can't use with a debugger.

deterministic replay

(Mugshot, WaRR)

Save and reuse nondeterministic inputs

to exactly recreate a specific

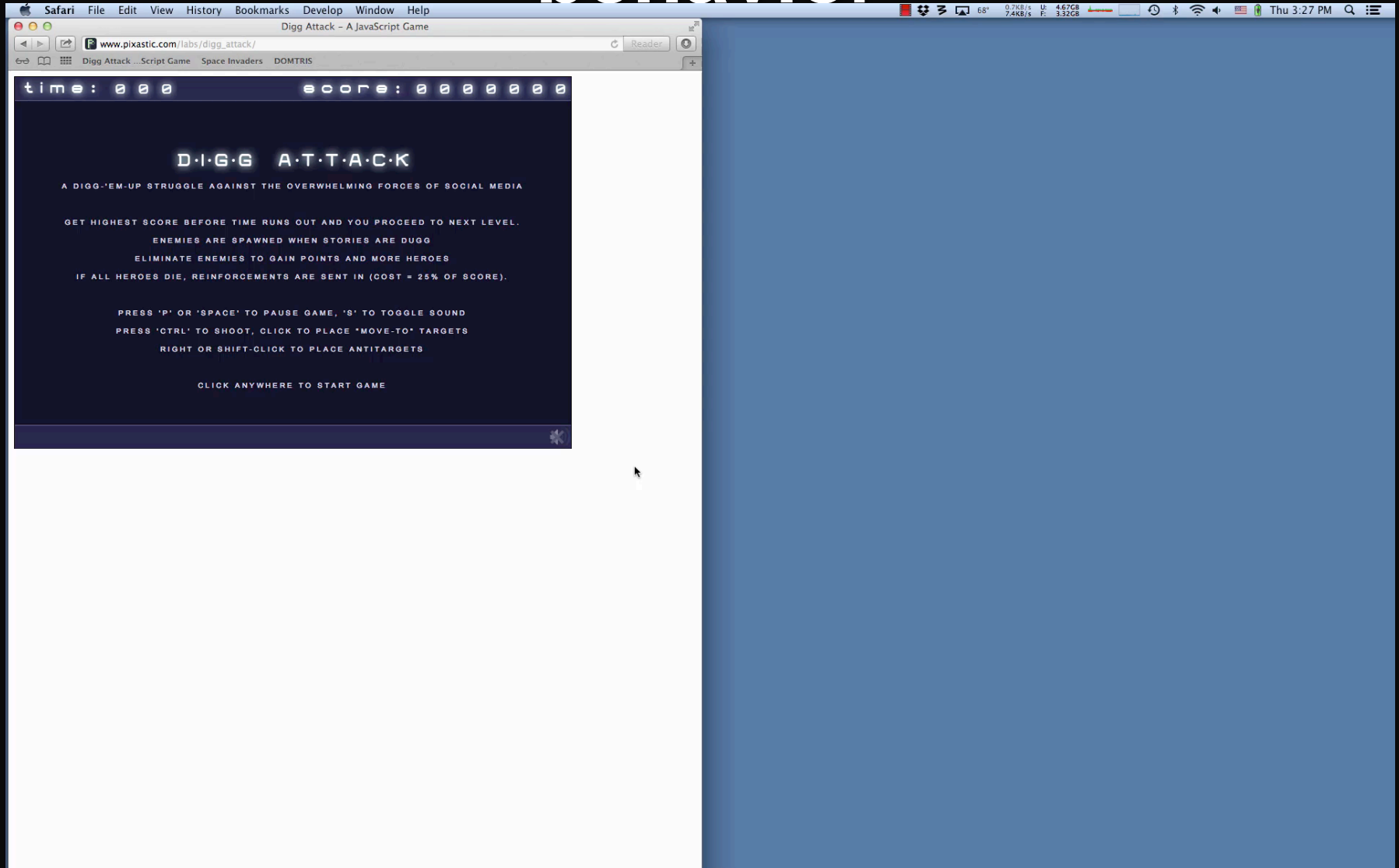
Play/pause buttons only. Slows down execution. Can't use with a debugger.

Timelapse: a precise, fast, integrated replay tool

This talk:

- An interface for capturing and replaying program behavior
- Techniques for cheap, precise record/replay in web browsers
- How developers use record/replay during debugging tasks

How to capture program behavior



How to navigate a recording

The screenshot shows the Safari Web Inspector interface. The top bar includes the Safari menu and system status. The main area displays a recording of a JavaScript game. The recording is paused at 8.44s. The recording shows three tracks: User (green), Network (yellow), and Timer (red). A red vertical line marks the current time. The recording is paused, and the preview window is empty.

time: 047 score: 0003902

oh no!

- CLICK MOUSE TO PLACE REINFORCEMENT HEROES -

Preview Window

Nothing to preview.
Select something from a timeline at left.

Paused

Using replay while debugging

The image shows a Safari browser window with the URL `matthaynes.net/playground/javascript/glow/spaceinvaders/`. The page content includes:

- JavaScript Space Invaders**: A heading followed by a paragraph: "Here's my take on Space Invaders written purely in javascript." and a link: "For more info [read the blog post](#)".
- Game Interface**: A black box containing:
 - SCORE: 0
 - LIVES: [Three green alien icons]
 - SPACE INVADERS JS
 - PLAY
 - CONTROLS
 - UP ARROW - FIRE
 - LEFT ARROW - MOVE LEFT
 - RIGHT ARROW - MOVE RIGHT
 - P - PAUSE
- Advertisement**: A box for "Javascript Fix" from "Javascript-Repair.SmartPCFixer.com" with a "How To Repair Javascript in 2 Min. Only 3 Steps (Recommended)" button.

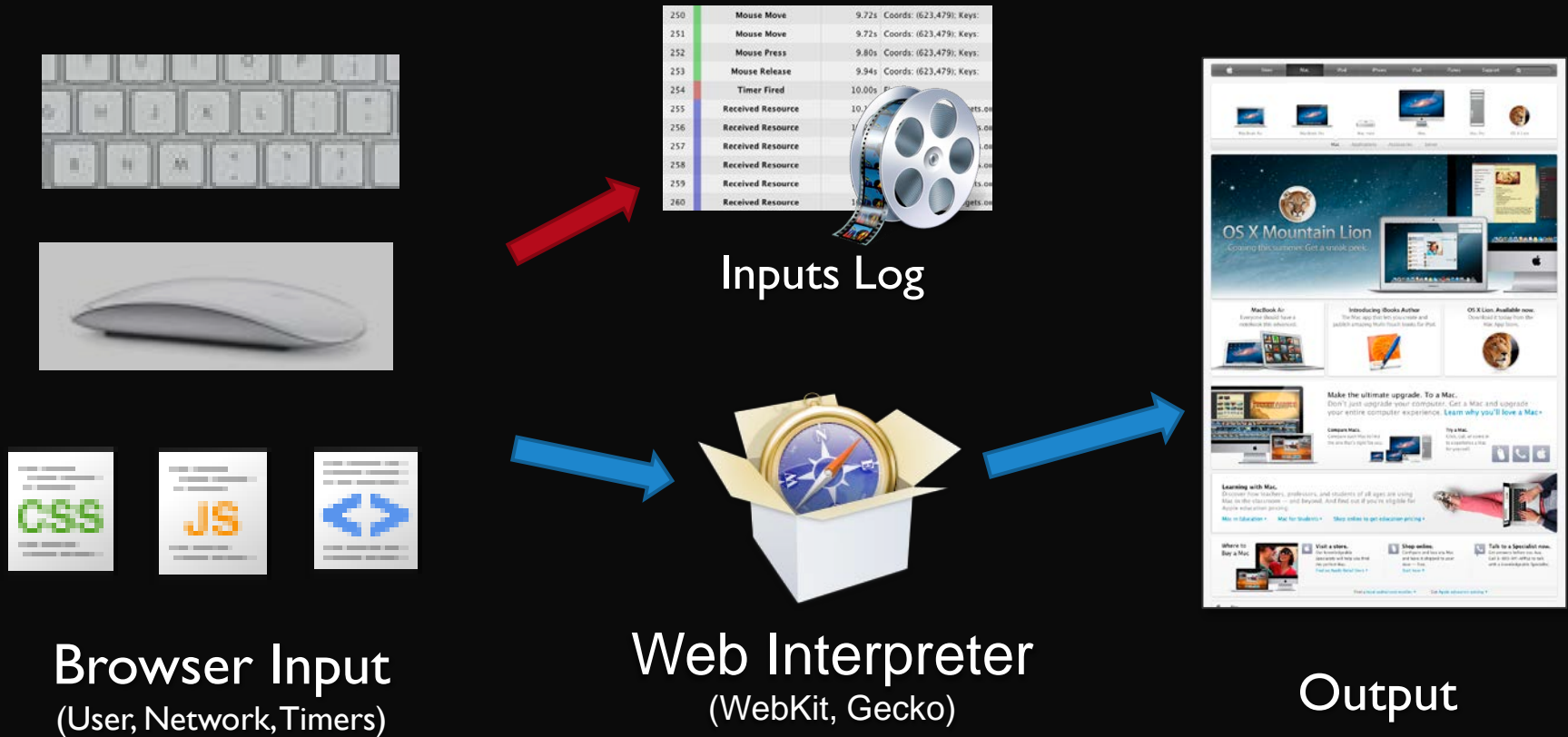
The Web Inspector on the right shows the DOM tree with the following structure:

```
<!DOCTYPE html PUBLIC "-//W3C/DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" lang="en-GB">
  <head profile="http://dublincore.org/documents/dcq-html/"></head>
  <body class="glow-basic">
    <div id="info"></div>
    <div id="board_holder"></div>
    <div id="adsenseUnit"></div>
  </body>
</html>
```

The Styles pane shows the following CSS rules:

```
element.style {
}
Matched CSS Rules
body {
  background-color: #000;
  text-align: center;
}
body {
  display: block;
  margin: 8px;
}
Metrics
Properties
DOM Breakpoints
Event Listeners
```


Timelapse captures a browser's inputs



Timelapse replays a browser's inputs

250	Mouse Move	9.72s	Coords: (623,479); Keys:
251	Mouse Move	9.72s	Coords: (623,479); Keys:
252	Mouse Press	9.80s	Coords: (623,479); Keys:
253	Mouse Release	9.94s	Coords: (623,479); Keys:
254	Timer Fired	10.00s	
255	Received Resource	10.00s	
256	Received Resource		
257	Received Resource		
258	Received Resource		
259	Received Resource		
260	Received Resource		

Inputs Log

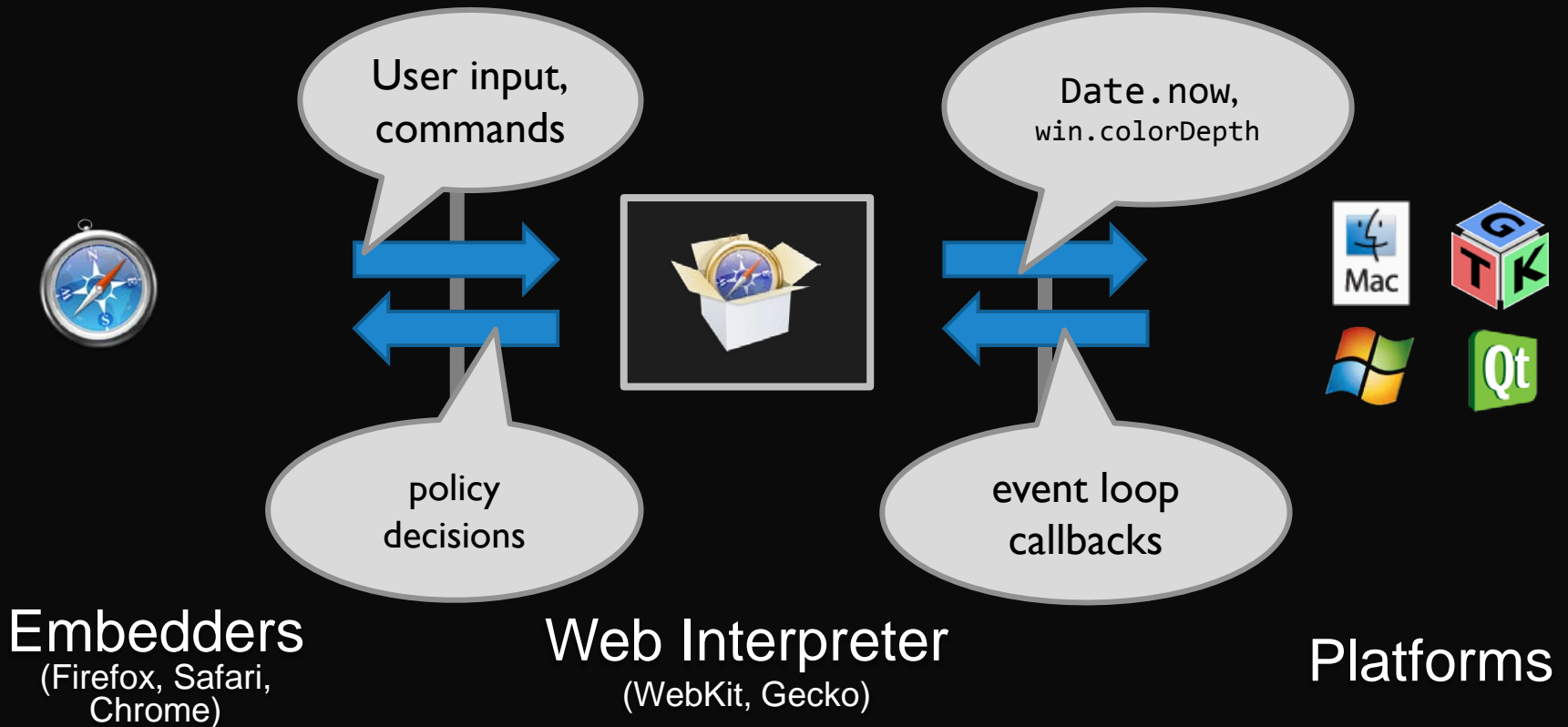


Web Interpreter
(WebKit, Gecko)

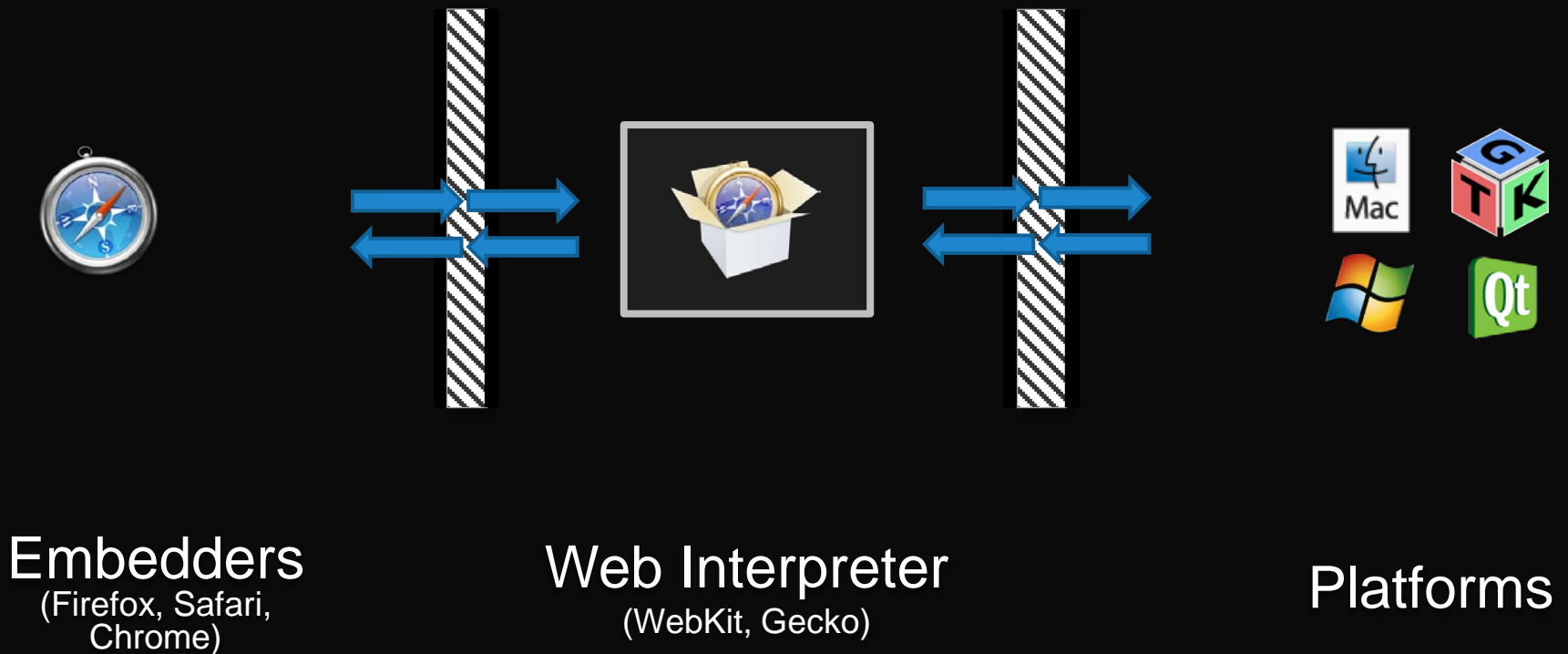


Output

Browsers have layered architectures



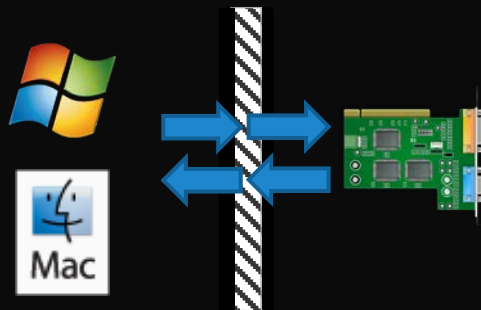
Timelapse intercepts input at layer boundaries



Inspired by VM record/replay

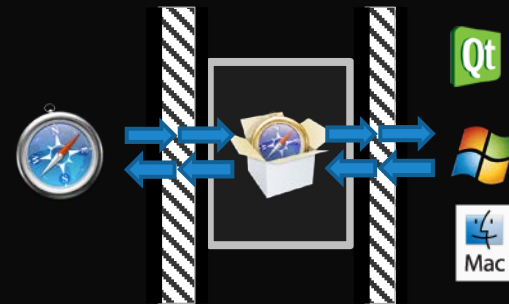
VM record/replay

- Hardware interrupts
- Nondeterministic instructions
- Instruction counts



Browser

- record/replay
Async callbacks
- Nondeterministic APIs
- DOM event counts

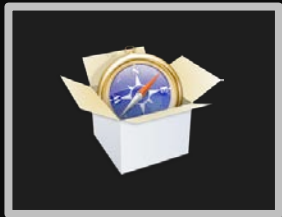


Memoizing nondeterministic APIs

During normal execution, `Date.now()` returns the current time.

```
oo(a,b) { return a + b; }  
(c) { return "now:" + Date.now(); }
```

```
/* file: Source/wtf/DateMath.h */  
inline double jsCurrentTime()  
{  
  return floor(WTF::currentTimeMS());  
}
```



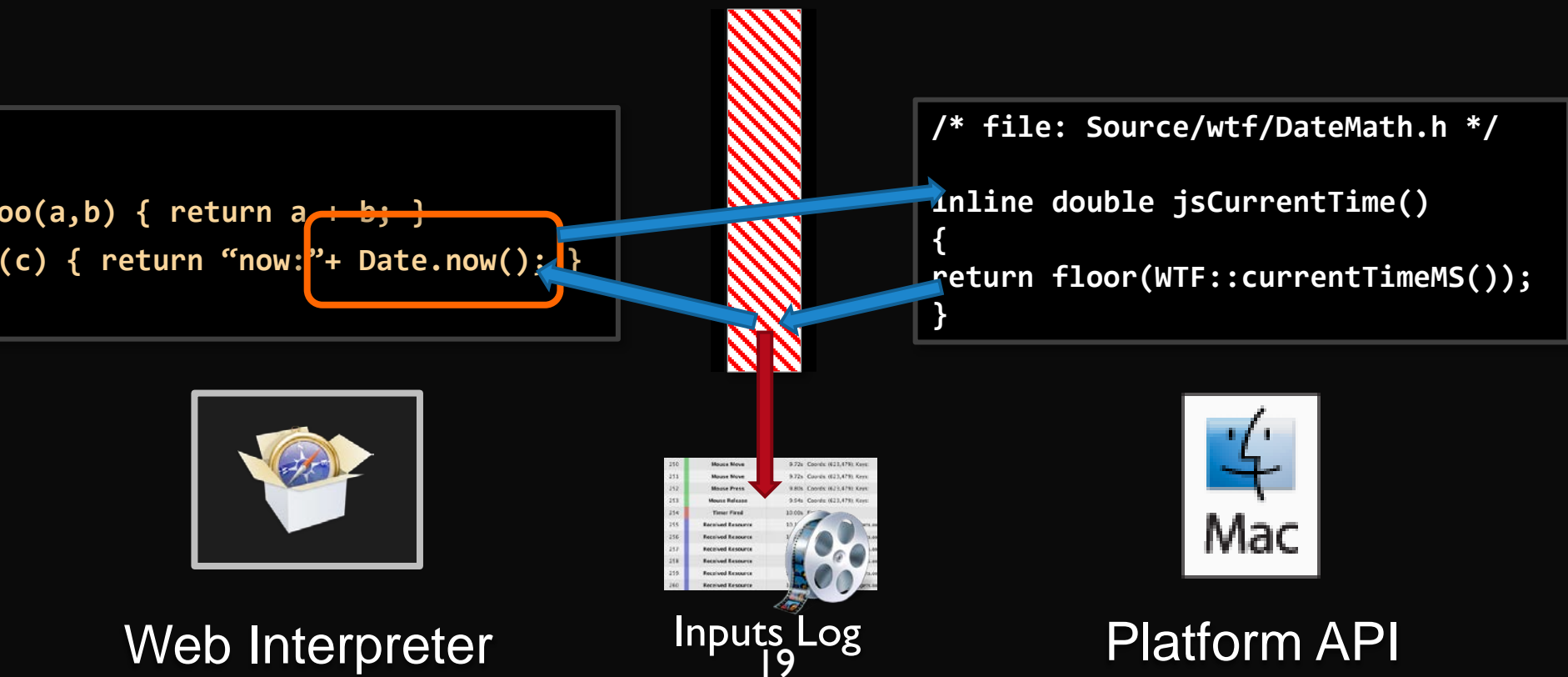
Web Interpreter



Platform API

Memoizing nondeterministic APIs

During recording, the return value of `Date.now()` is saved.



Web Interpreter

Inputs Log
19

Platform API

Memoizing nondeterministic APIs

On replay, the logged return value of `Date.now()` is used.

```
function foo(a,b) { return a + b; }  
function (c) { return "now:" + Date.now(); };
```



Web Interpreter

210	Mouse Move	9.720	Coords (623,476) Keys
211	Mouse Move	9.720	Coords (623,476) Keys
212	Mouse Press	9.800	Coords (623,476) Keys
213	Mouse Release	9.840	Coords (623,476) Keys
214	Timer Fired	10.000	F
215	Received Resource	10.0	
216	Received Resource		
217	Received Resource		
218	Received Resource		
219	Received Resource		
220	Received Resource		

Inputs Log

```
/* file: Source/wtf/DateMath.h */  
  
inline double jsCurrentTime()  
{  
  return floor(WTF::currentTimeMS());  
}
```



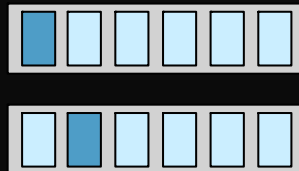
Platform API

Making callbacks deterministic



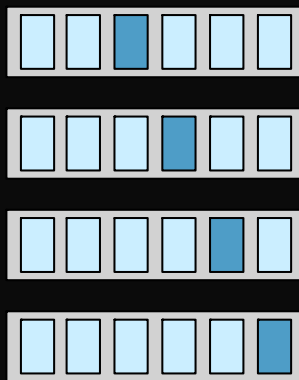
Callback registered

enqueue()



```
Event Loop
while (true) {
  var event = queue.pop();
  this.dispatchToListeners(event);
}
```

Problem: accurately capturing and simulating event loop dispatches.



timerFired()



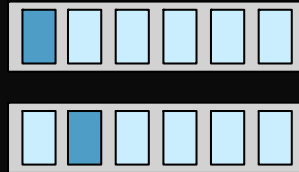
Callback executes

Making callbacks deterministic

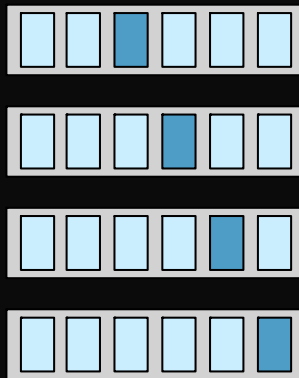


Callback registered

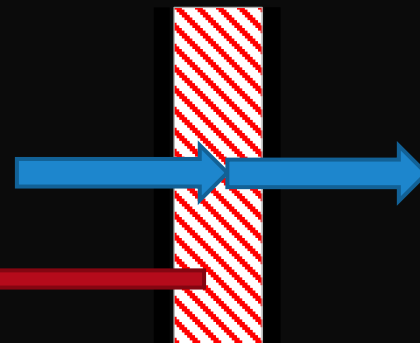
enqueue()



```
Event Loop
while (true) {
  var event = queue.pop();
  this.dispatch(event);
}
```



timerFired()



210	Mouse Move	0.72s	Coords (623,478) Key:
211	Mouse Move	0.72s	Coords (623,478) Key:
212	Mouse Press	0.80s	Coords (623,478) Key:
213	Mouse Release	0.84s	Coords (623,478) Key:
214	Timer Fired	10.00s	
215	Received Resource	10.1s	
216	Received Resource		
217	Received Resource		
218	Received Resource		
219	Received Resource		
220	Received Resource		

Inputs Log

timer 42,
34 DOM events



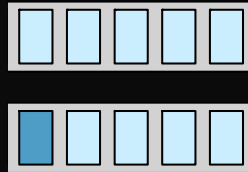
Callback executes

Making callbacks deterministic



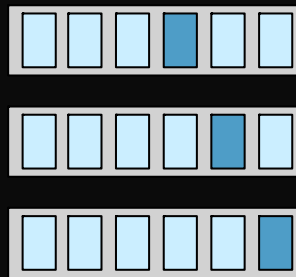
Callback registered

enqueue()



```
Event Loop
while (true) {
  var event = queue.pop();
  this.dispatchToListeners(event);
}
```

...



timerFired()



34 DOM events!



210	Mouse Move	9.72s	Coords: (623,476) Key:
211	Mouse Move	9.72s	Coords: (623,476) Key:
212	Mouse Press	9.80s	Coords: (623,476) Key:
213	Mouse Release	3.15s	Coords: (623,476) Key:
214	Timer Fired	35.00s	
215	Received Resource	0s	
216	Received Resource		
217	Received Resource		
218	Received Resource		
219	Received Resource		
220	Received Resource		

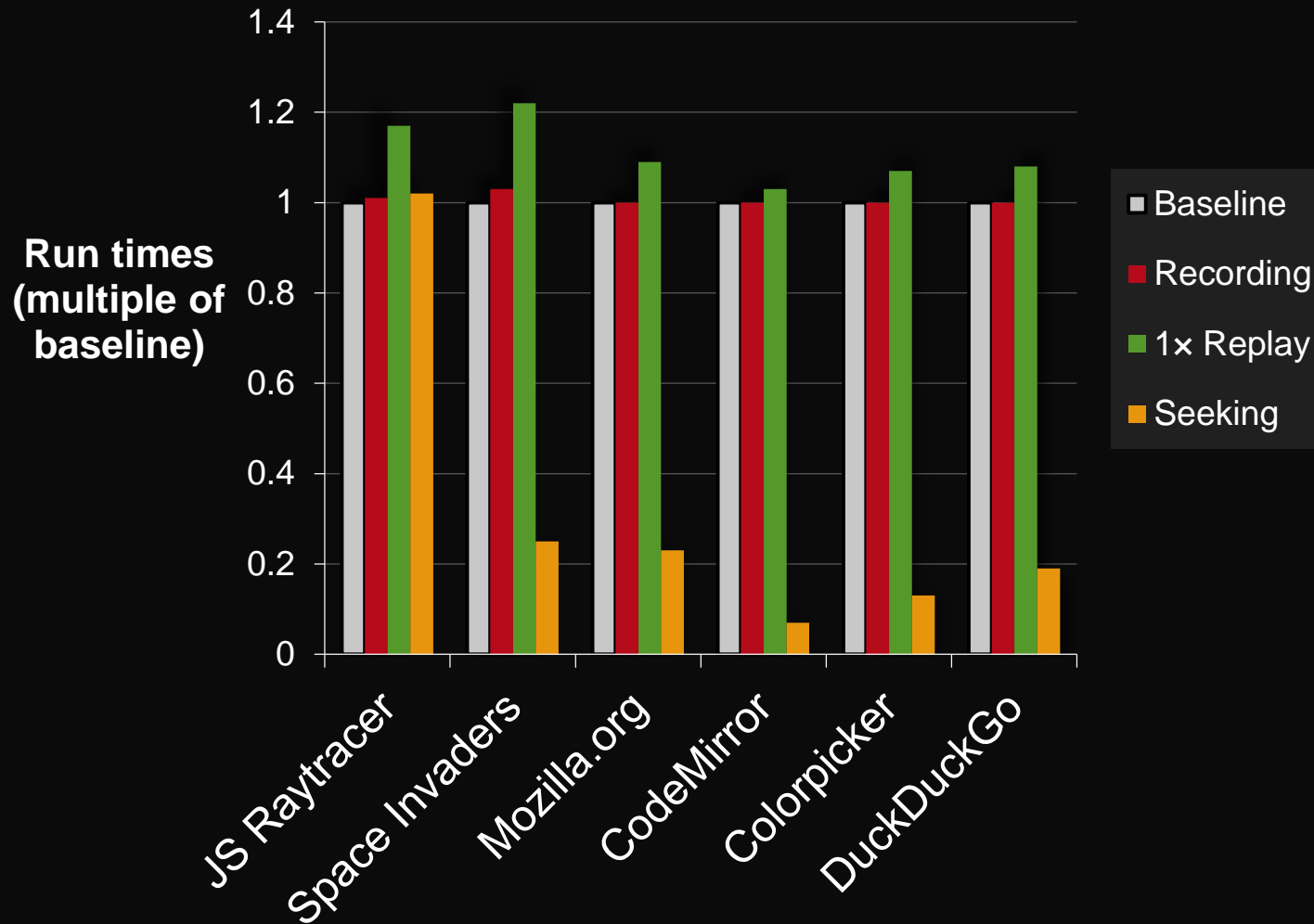


Inputs Log

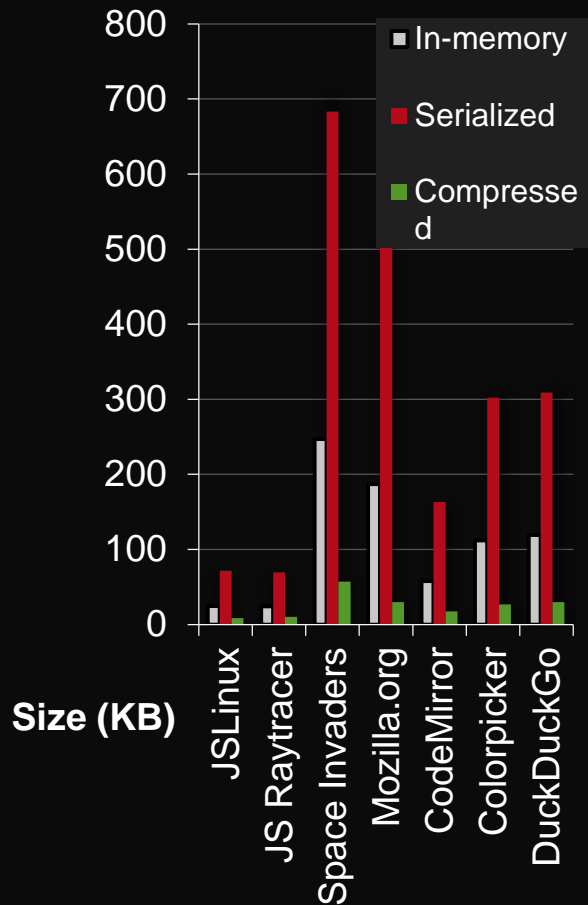


Callback executes

Runtime overheads are acceptable

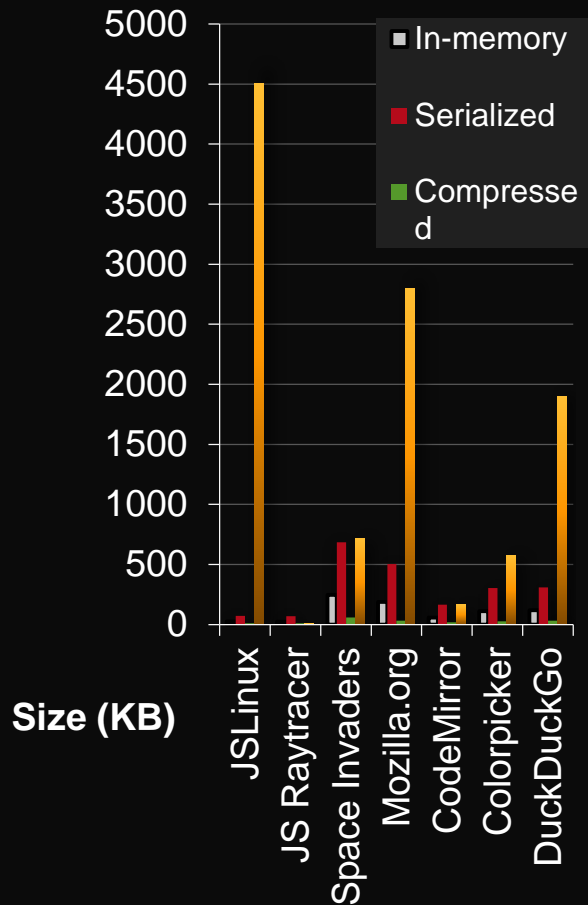


Recordings are small and compressible



Site	recording duration (s)	resources on page (KB)	log growth (KB/sec)
JSLinux	10.5	4500	0.8
JS Raytracer	6.3	5.9	1.6
Space Invaders	25.8	712	2.2
Mozilla.org	22.3	2800	1.3
CodeMirror	16.6	168	1.0
Colorpicker	15.3	577	1.7
DuckDuckGo	14.1	1900	2.1

Page resources dominate recording size



Site	recording duration (s)	resources on page (KB)	log growth (KB/sec)
JSLinux	10.5	4500	0.8
JS Raytracer	6.3	5.9	1.6
Space Invaders	25.8	712	2.2
Mozilla.org	22.3	2800	1.3
CodeMirror	16.6	168	1.0
Colorpicker	15.3	577	1.7
DuckDuckGo	14.1	1900	2.1

How would developers use it?

Study Design

20+ developers with industry experience
within-subjects, 2 tasks per person,
45 minutes per task, 4 treatments

Reproduction

*RQ: changes to
frequency/duration?*

Performance

*RQ: complete tasks more
quickly? more successfully?
Who₂₇ why?*

How did developers use it?

Study Design

20+ developers with industry experience
within-subjects, 2 tasks per person, 45 minutes per task, 4 treatments

Reproduction

Shorter and more frequent reproductions;
Time spent unchanged (max. 25%);
Successful developers quickly

Performance

integrated replay into their existing workflows.
Unsuccessful developers who used opportunistic strategies were distracted.

Current & Future Work

Visualizations

Interaction histories aid navigation, but not program understanding.

Passive capturing

Precision and low overhead don't matter if you forget to start capturing.

Post-hoc analysis

Developers can gather more runtime data without reproducing behavior:

Post-hoc logging, Post-hoc Whyline, Post-hoc SeeSS, Testcase extraction

Conclusion

Record/Replay

Virtual machine replay techniques work well when applied to web applications.

Visualizations

Interaction histories supported—but didn't reduce—reproduction of program state.

Infrastructure

Replay infrastructure enables new research, tools and workflows.



github.com/burg/timelapse

Replay fidelity and completeness

Divergence detection supports piecewise implementation.

Web interpreters expose a large and ever-changing API.

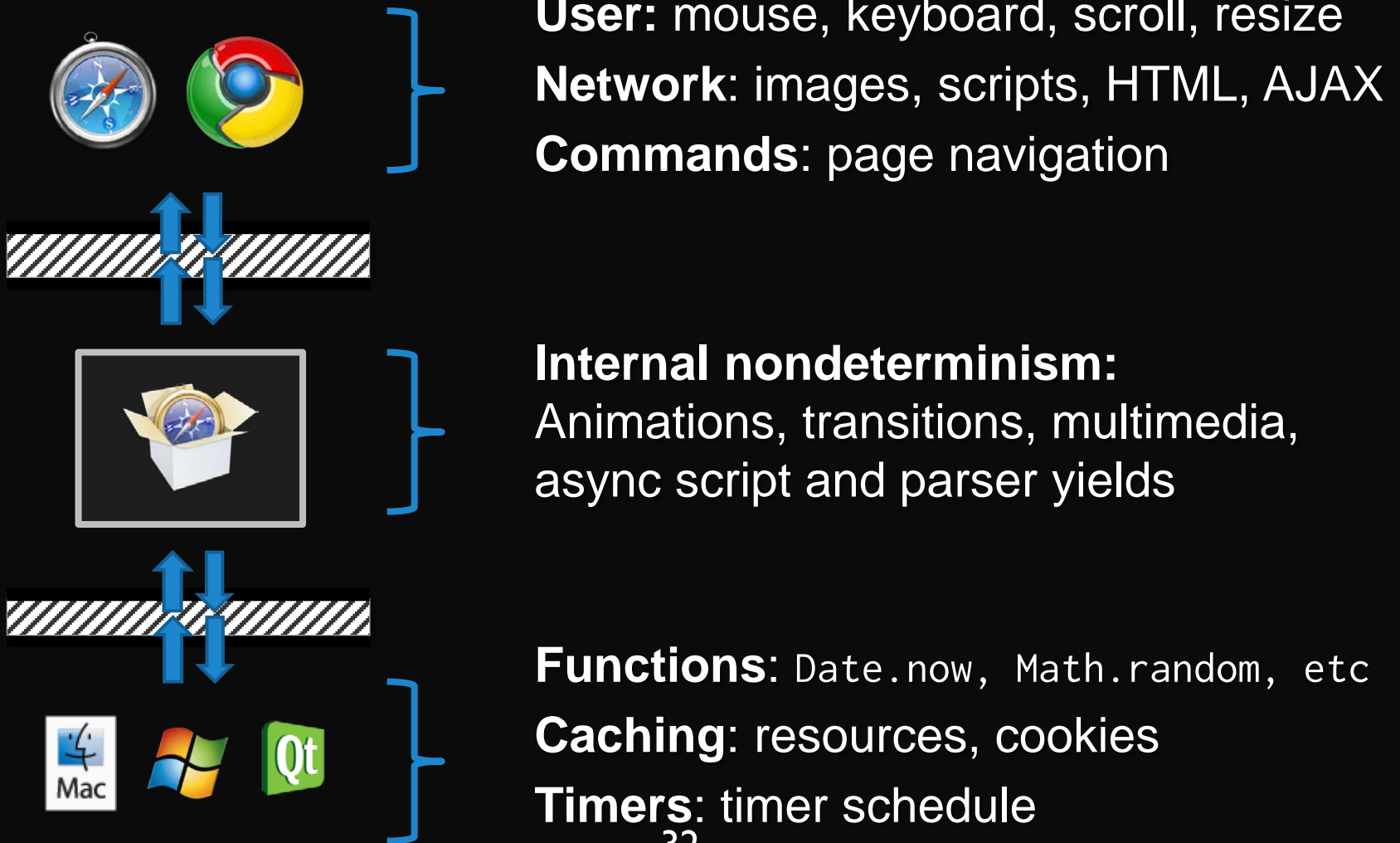
Timelapse doesn't tame all sources of nondeterminism (yet).

Excepting untamed sources, the DOM tree and JavaScript heap are identical for all recorded and replayed executions.

Possible divergence is automatically detected when:

- DOM event counts differ on capture and replay
- Memoized inputs are overused or unused
- Network request details differ unexpectedly
- Known-bad APIs are used by client code

Interpreter inputs by source



Shim: the thing in the middle

Shims are used to implement deterministic record/replay.

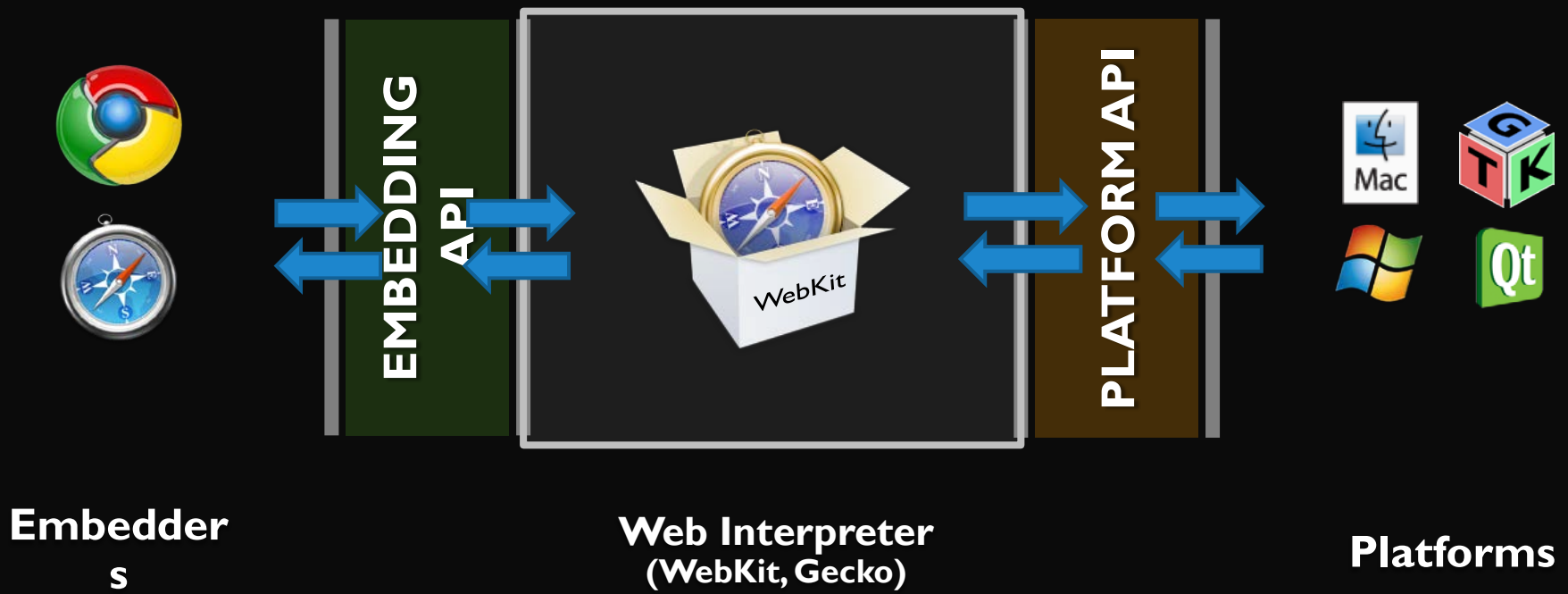


```
1
2 #if ENABLE(TIMELAPSE)
3 static double jsRiggedCurrentTime(JSGlobalObject* globalObject)
4 {
5     if (RefPtr<DeterminismLog> log = globalObject->determinismLog()) {
6         double currentTime;
7
8         if (log->capturing()) {
9             currentTime = jsCurrentTime();
10            log->append(new GetCurrentTime(currentTime));
11        } else {
12            ASSERT(log->replaying());
13            GetCurrentTime* action = log->currentAction<GetCurrentTime>();
14            currentTime = action->currentTime();
15        }
16        return currentTime;
17    }
18    |
19    //if no determinism shenanigans are going to happen
20    return jsCurrentTime();
21 }
22 #endif
```

The hard part of implementing record/replay is designing and placing shims.

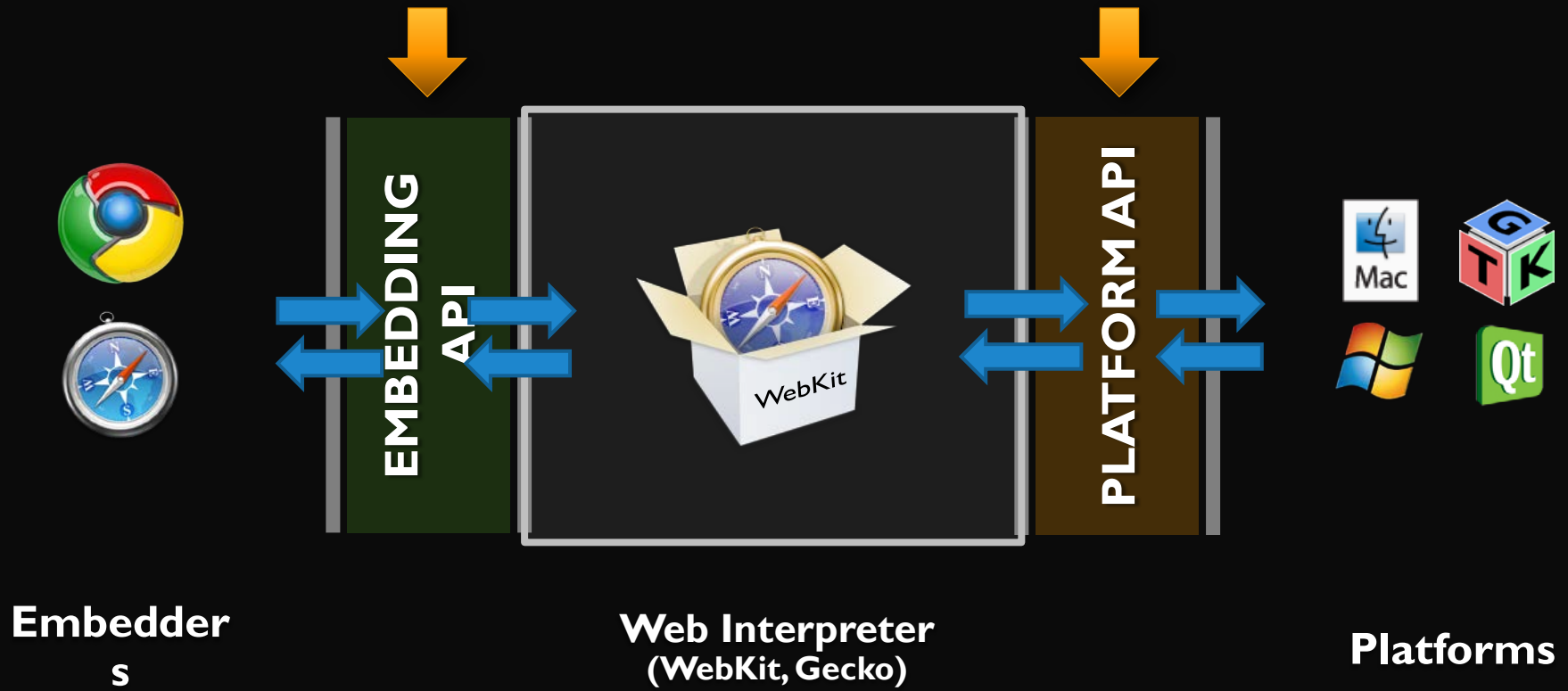
Embedding and platform APIs

Abstraction layers separate web interpreters from platforms/embedders.



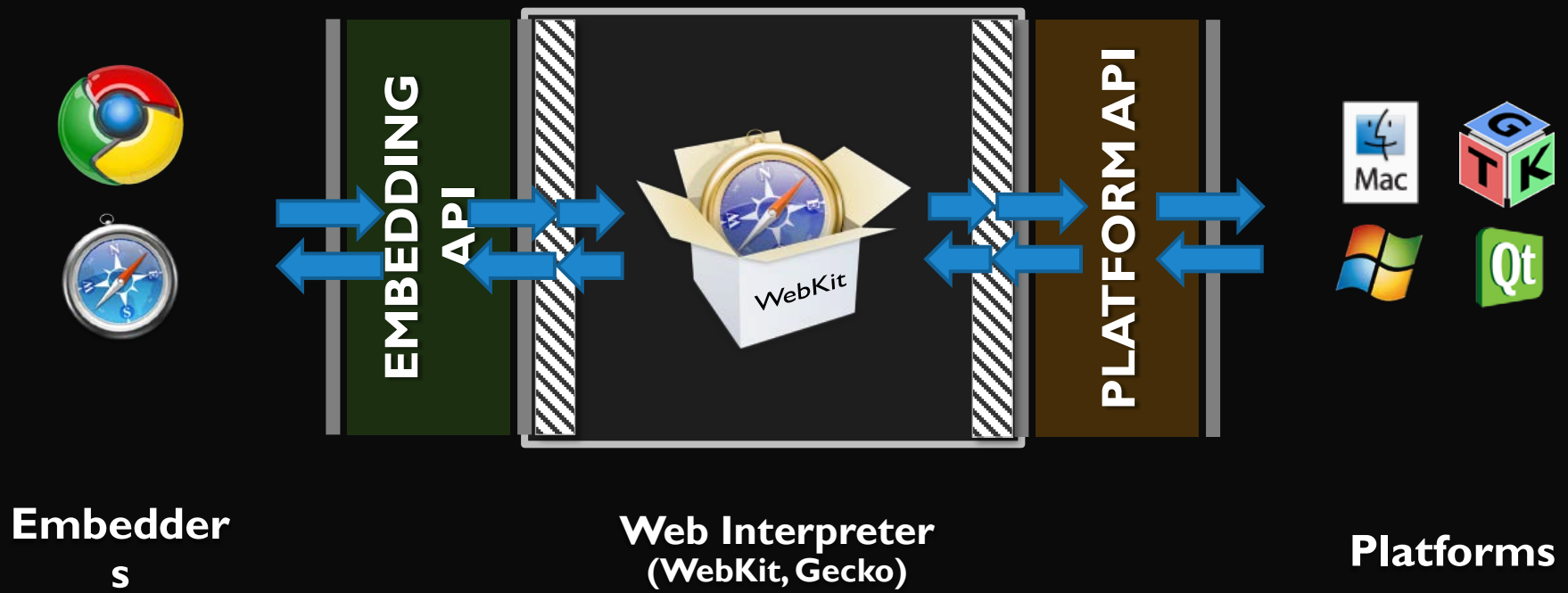
Embedding and platform APIs

Abstraction layers separate web interpreters from platforms/embedders.



Embedding and platform APIs

Shims sit between the web interpreter and abstraction layers.



Embedding and platform APIs

Shims sit between the web interpreter and abstraction layers.

