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

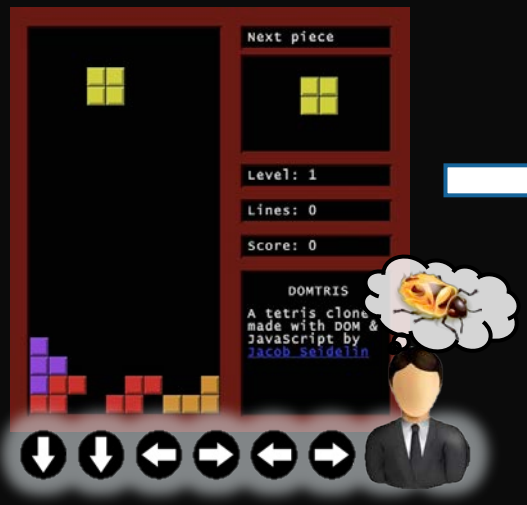
“It’s hard to debug failures in the field”

Distributed across time and space

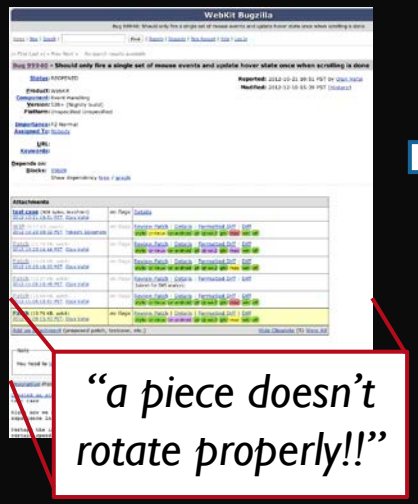
Hardware and software variation

Users are not software testers

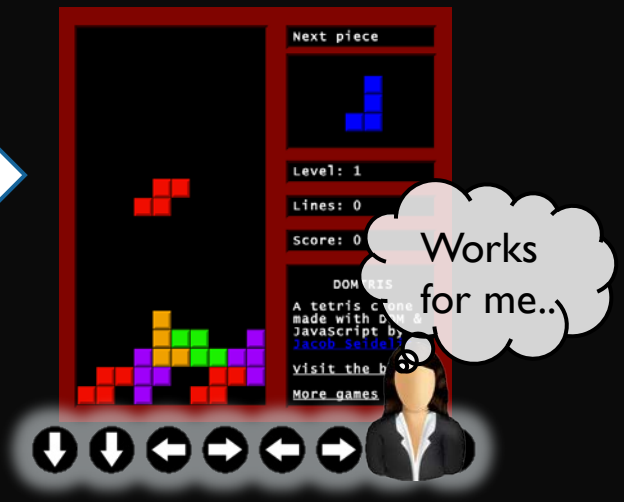
Users can’t report failures accurately



end-user encounters bug in production code

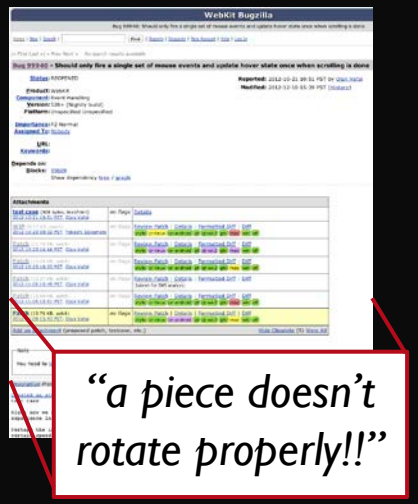


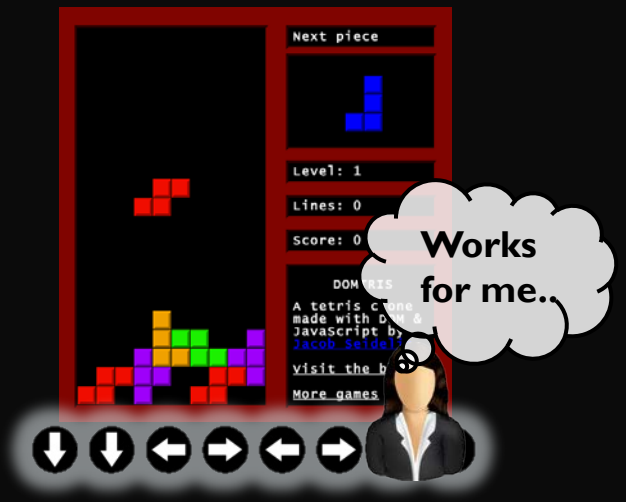
files bug report with ad-hoc information



developer unable to reproduce the bug

Users can’t report failures accurately

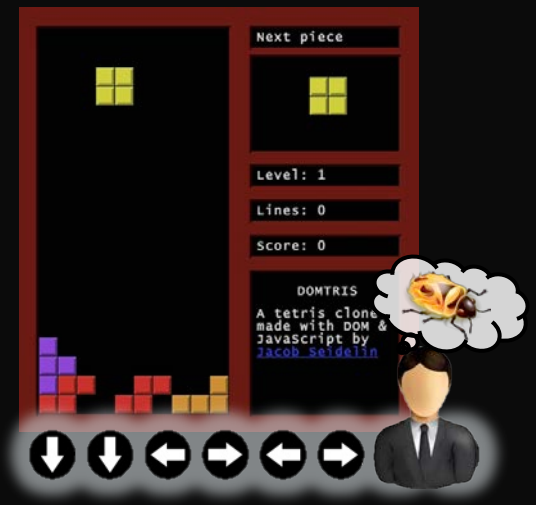


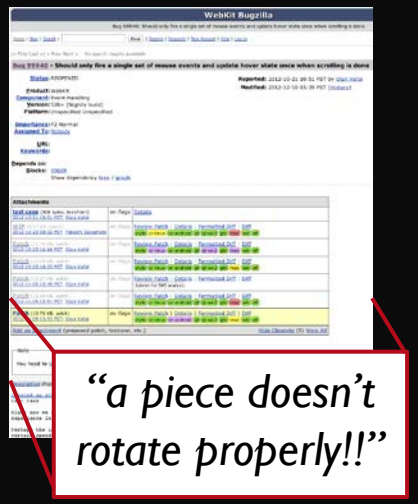


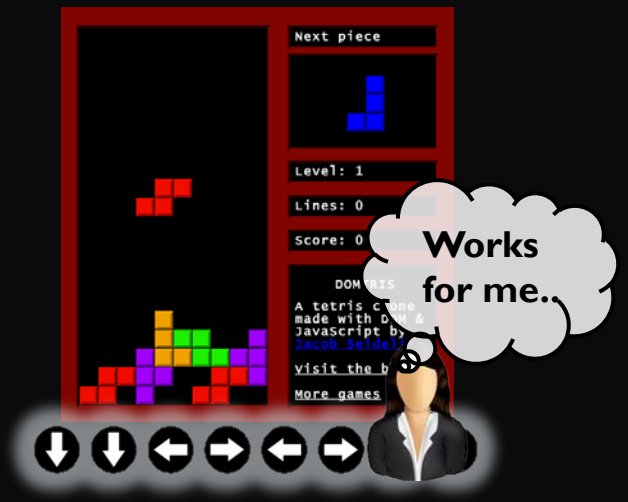
“The **most severe problems** were errors in *steps to reproduce* and *incomplete information*.”

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

Users can't report failures accurately







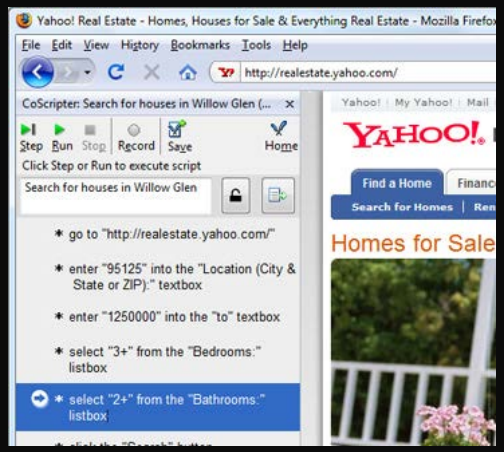
Bug reporters and developers want better tool support for reproducing buggy behavior.

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

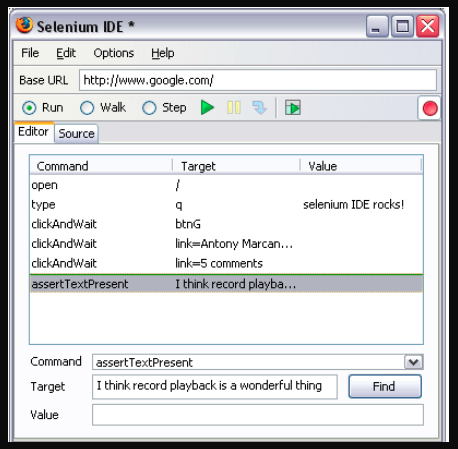
Existing tools are imprecise and hard to use

macro replay
(CoScripter, Selenium, Sikuli)

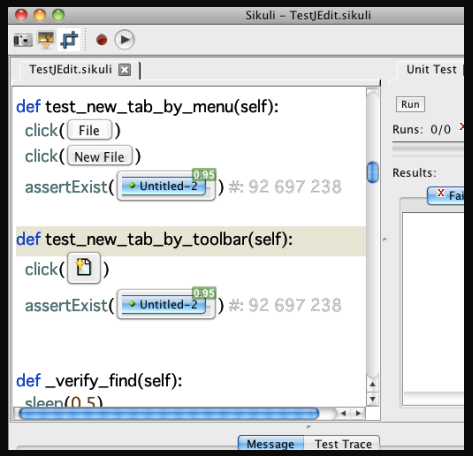
Capture and simulate user input. Designed for test and task automation.



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)

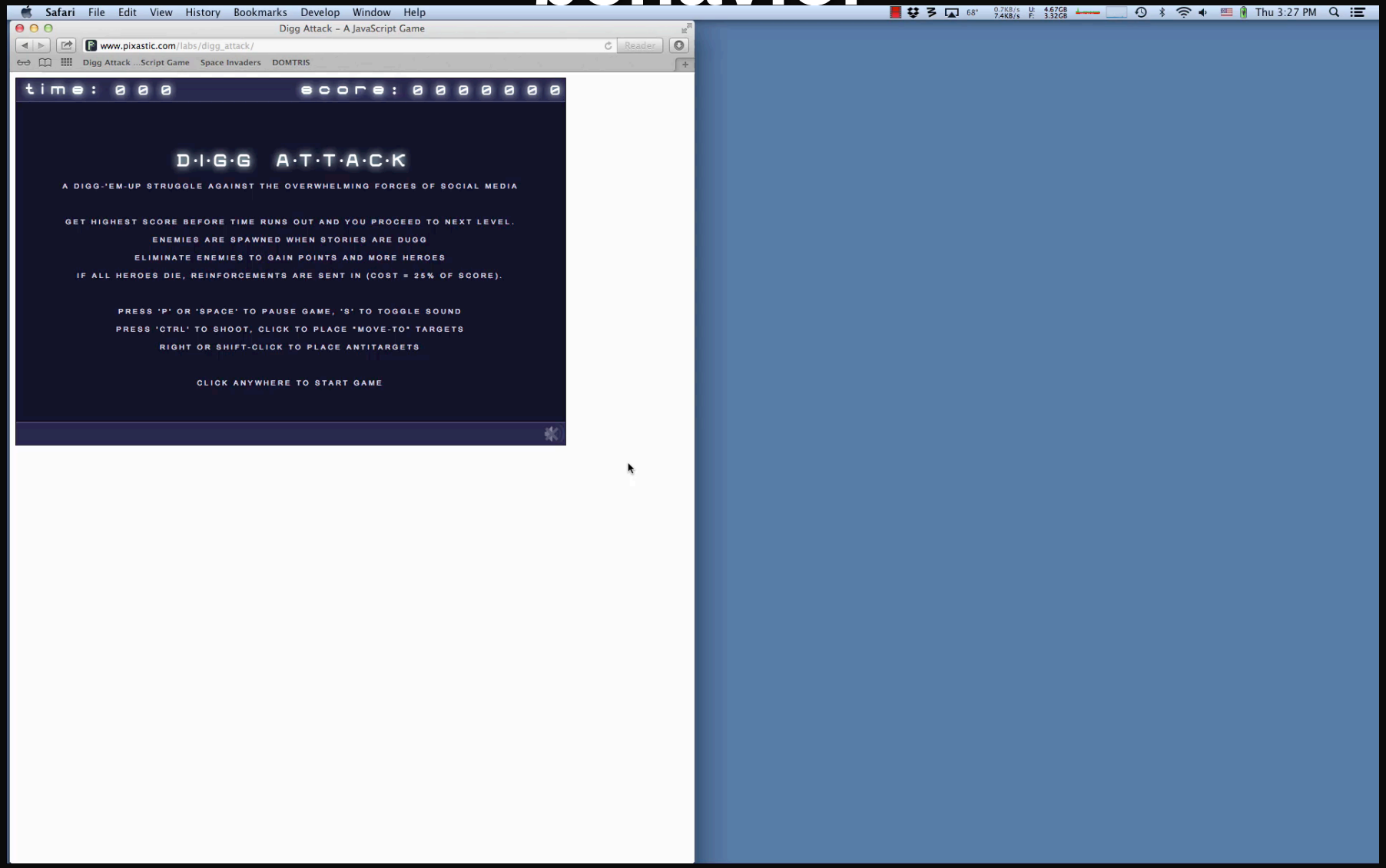
Save and reuse nondeterministic inputs
to exactly recreate a specific execution.
Play/pause buttons only. Slows down execution. Can't use with a debugger.

Timelapse: a precise, fast, integrated replay tool

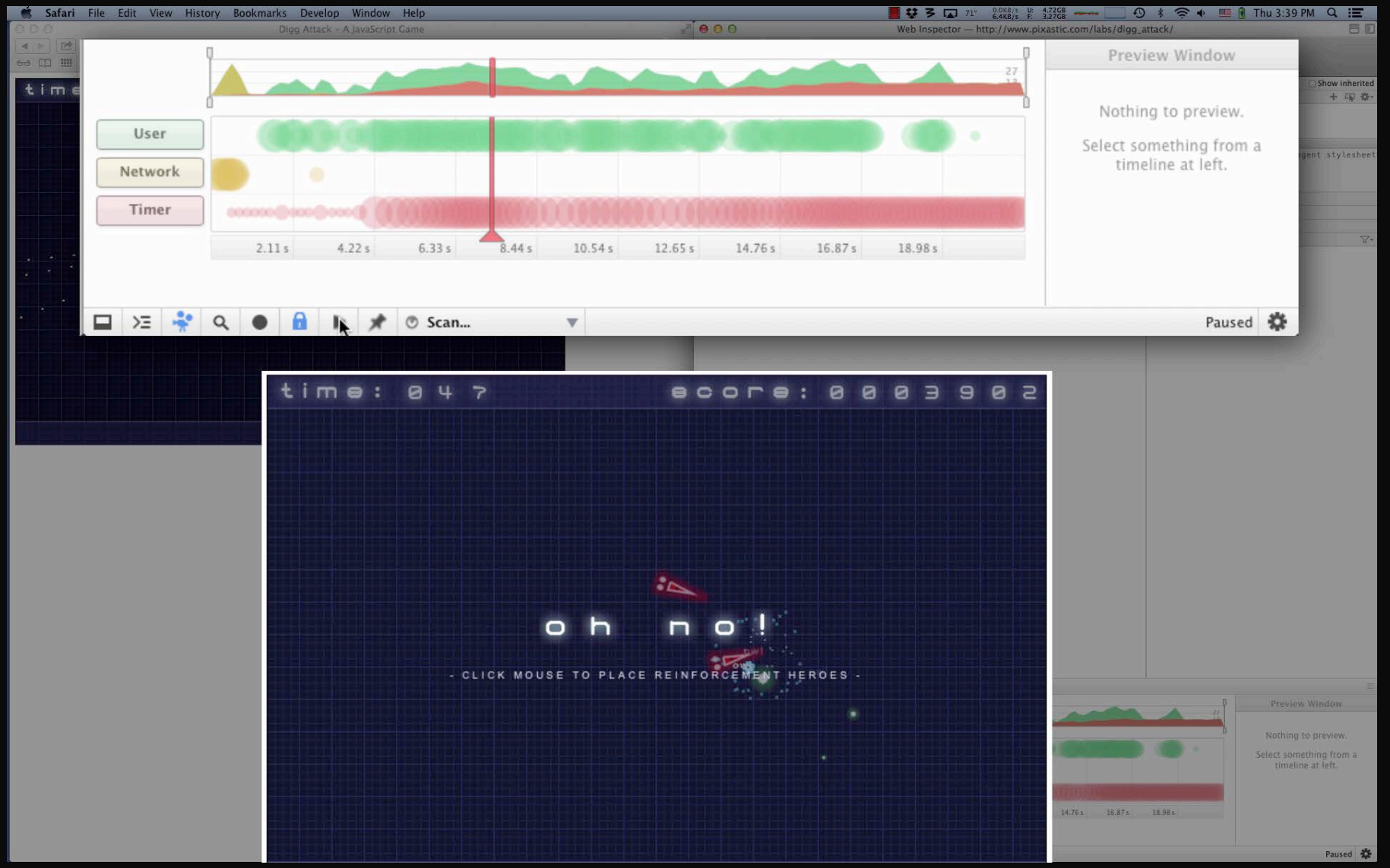
This talk:

- 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

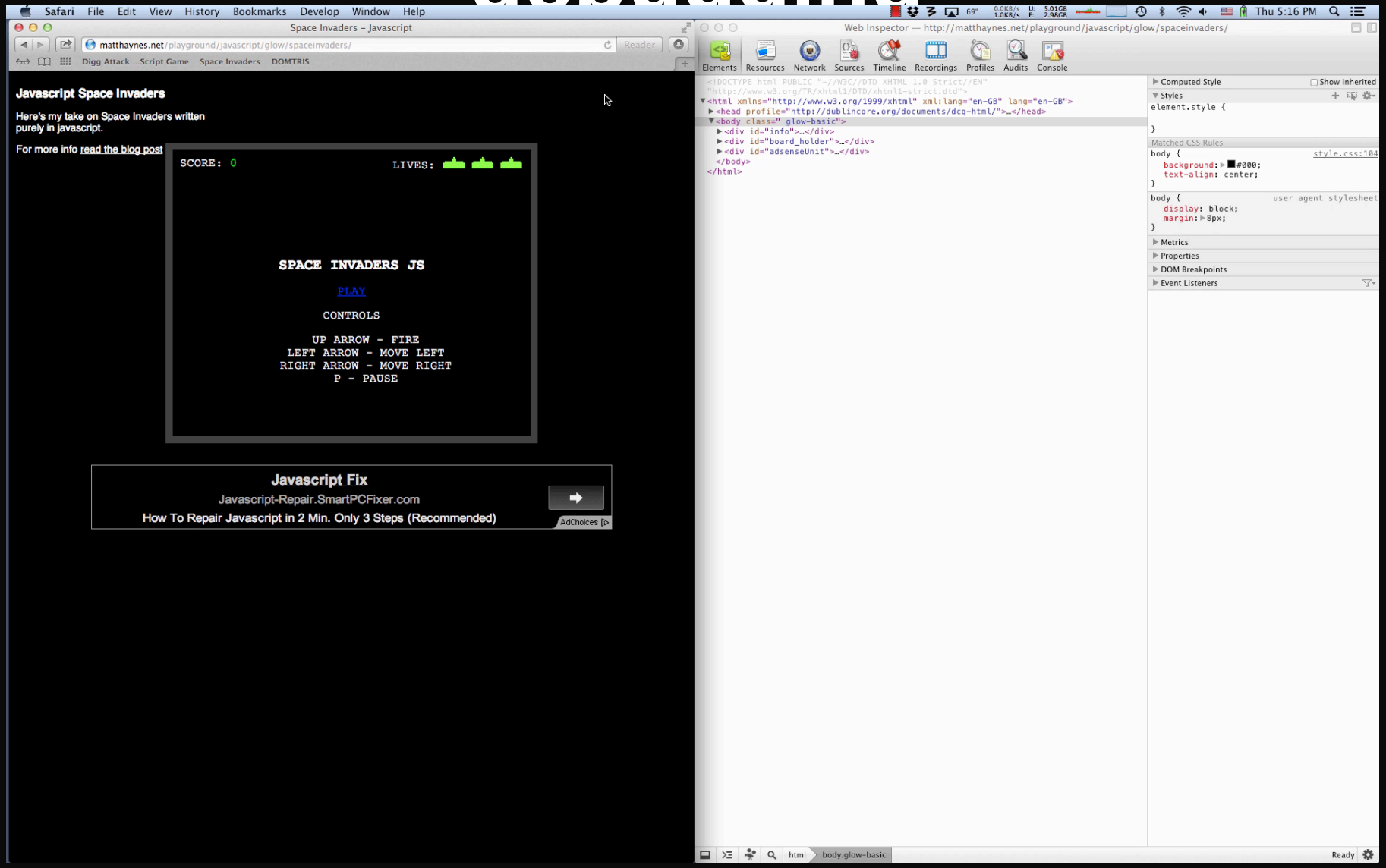
How to capture program behavior



How to navigate a recording



Using replay while debugging



Browsers interpret input, render output

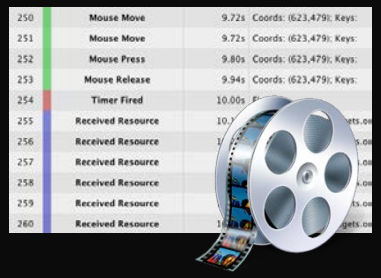
Browser Input
(User, Network, Timers)

Web Interpreter
(WebKit, Gecko)

Output

Timelapse captures a browser's inputs

Browser Input
(User, Network, Timers)

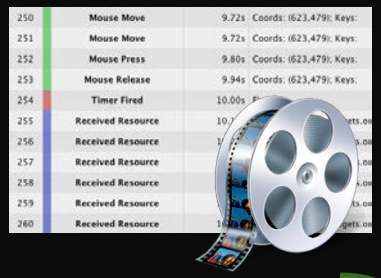


Inputs Log

Web Interpreter
(WebKit, Gecko)

Output

Timelapse replays a browser's inputs

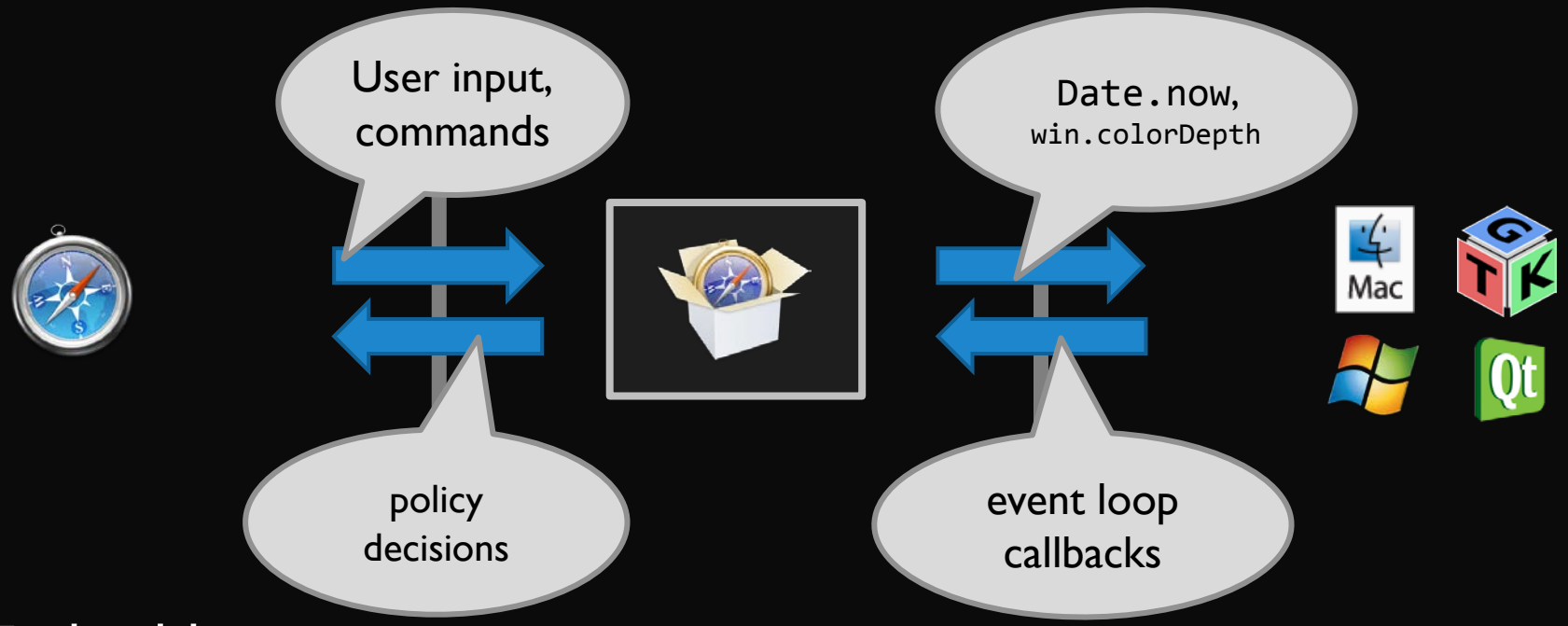


Inputs Log

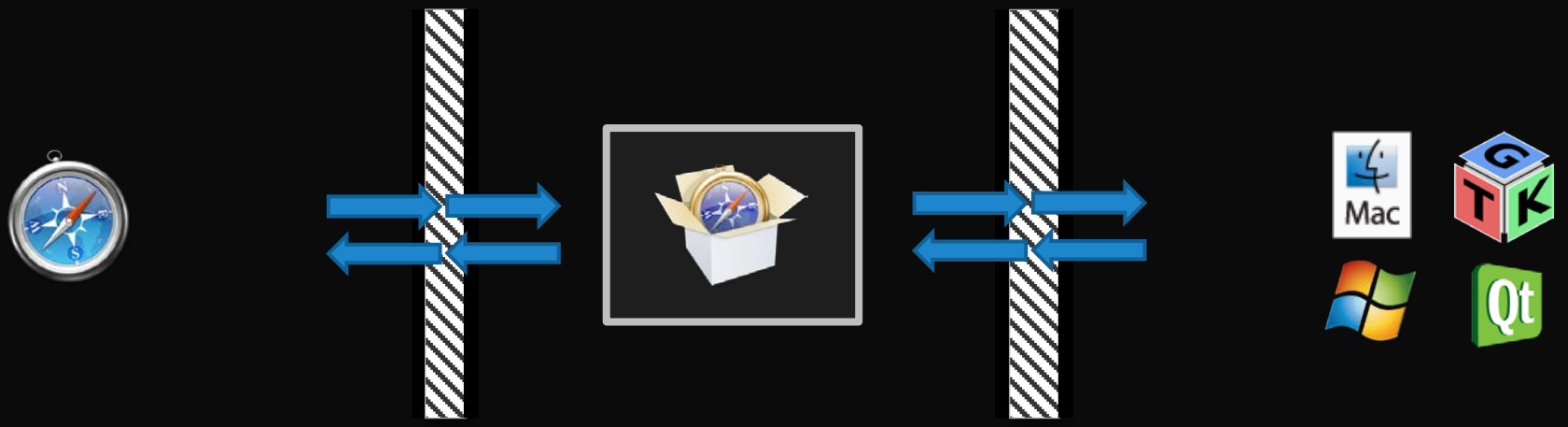
Web Interpreter
(WebKit, Gecko)

Output

Browsers have layered architectures



Timelapse intercepts input at layer boundaries



Embedders
(Firefox, Safari, Chrome)

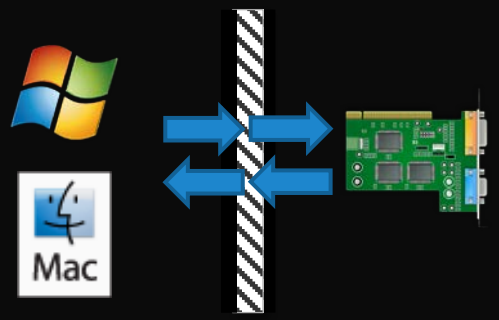
Web Interpreter
(WebKit, Gecko)

Platforms

Inspired by VM record/replay

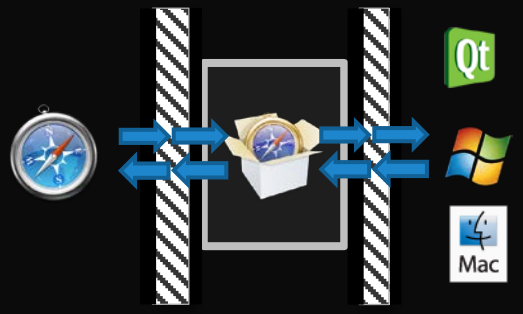
VM record/replay

- 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

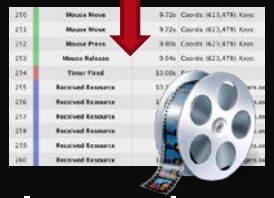
Platform API

Memoizing nondeterministic APIs

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

```
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

Inputs Log

Platform API

Memoizing nondeterministic APIs

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

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

Web Interpreter

Inputs Log

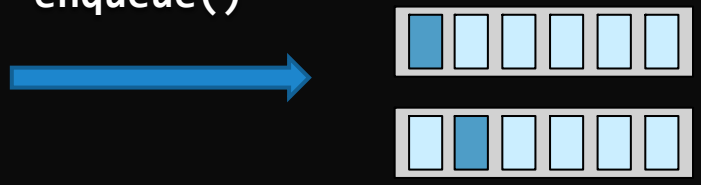
```
/* file: Source/wtf/DateMath.h */

inline double jsCurrentTime()
{
return floor(WTF::currentTimeMS());
}
```

Platform API

Making callbacks deterministic

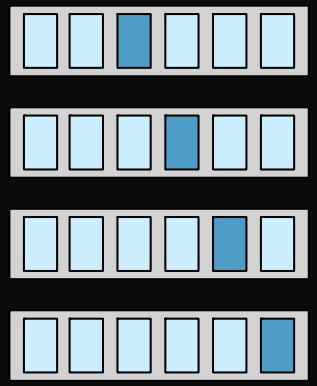
Callback registered

`enqueue()`



Event Loop

```
while (true) {
  var event = queue.pop();

this.dispatchToListeners(event);
}
```



`timerFired()`

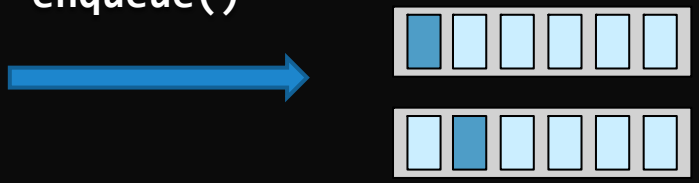
Callback executes

Problem: accurately capturing and simulating event loop dispatches.

Making callbacks deterministic

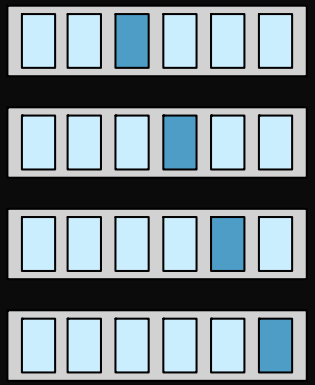
Callback registered

enqueue()

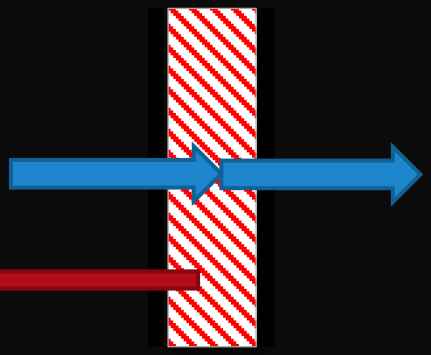


Event Loop

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

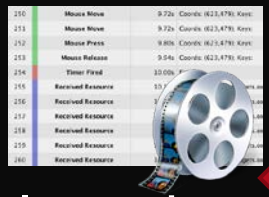


timerFired()



Callback executes

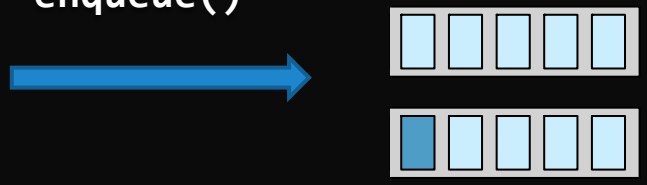
timer 42,
34 DOM events



Inputs Log

Making callbacks deterministic

enqueue()

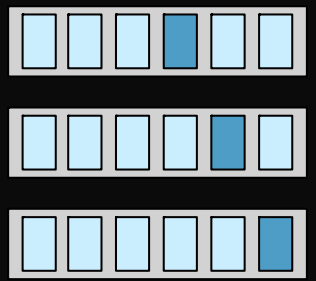


Callback registered

Event Loop

```
while (true) {
  var event = queue.pop();

this.dispatchToListeners(event);
}
```

...
...
...



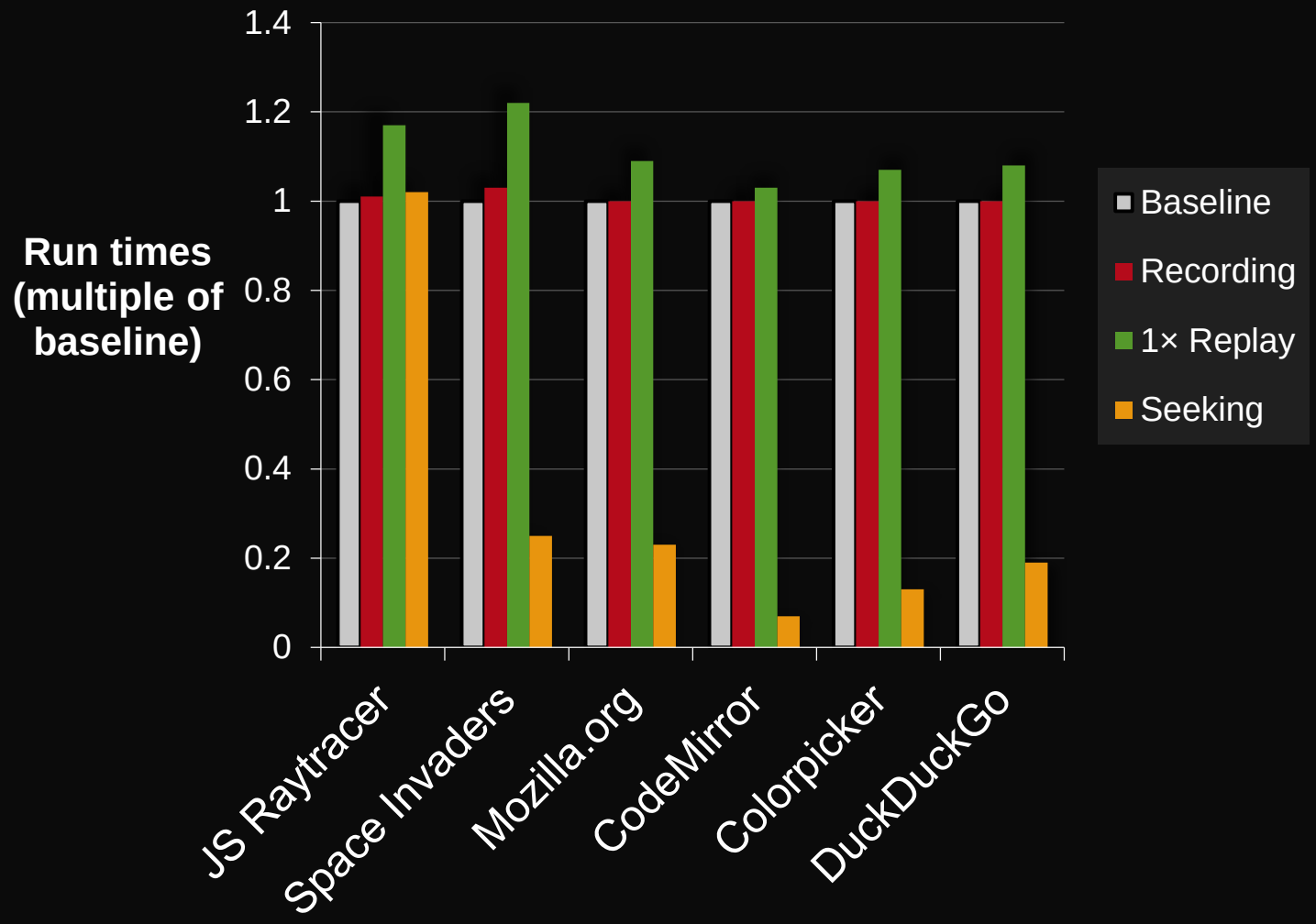
timerFired()

34 DOM events!

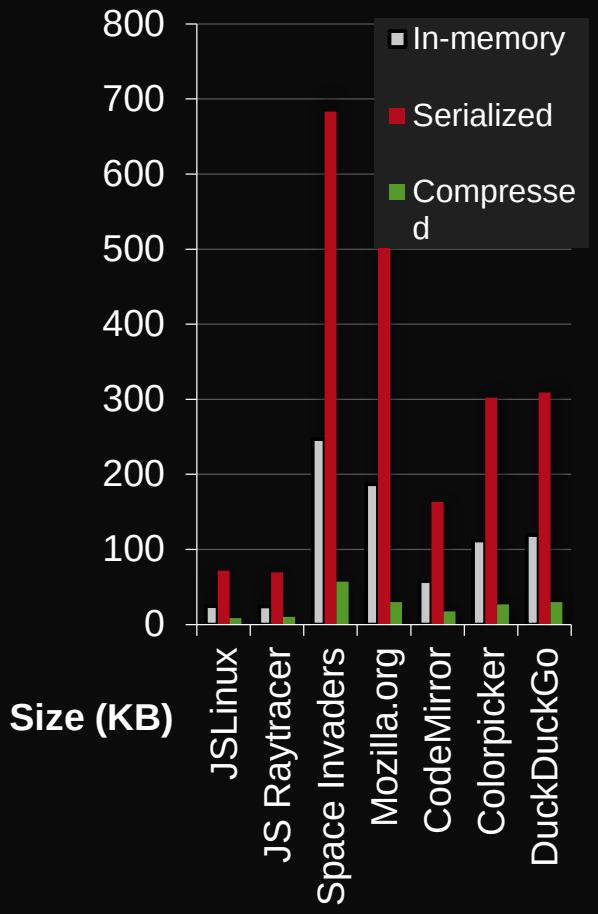
Inputs Log

Callback executes

Runtime overheads are acceptable

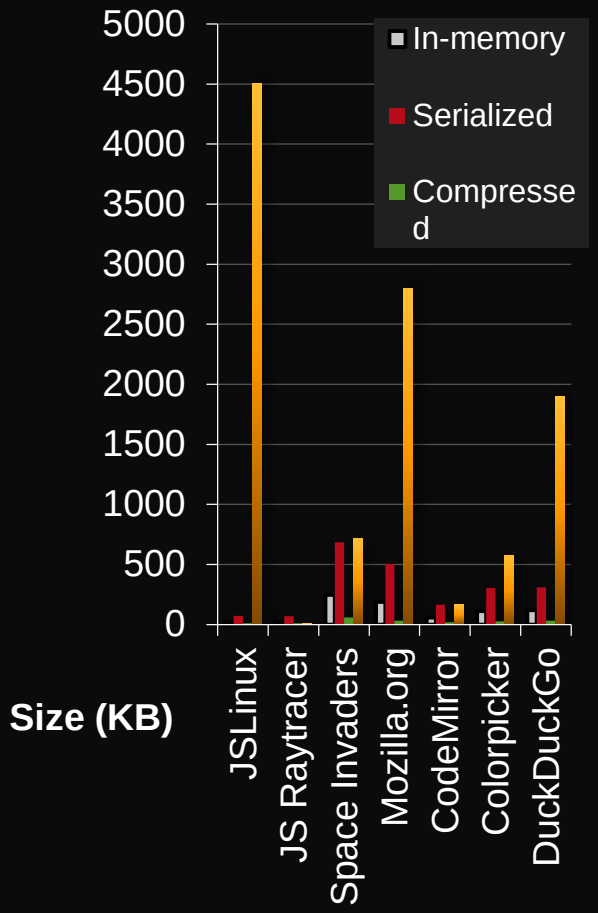


Recordings are small and compressible



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? Who, 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 repro actions;
Time spent unchanged (max. 25%; avg. 15%)

Performance

Successful developers quickly 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.

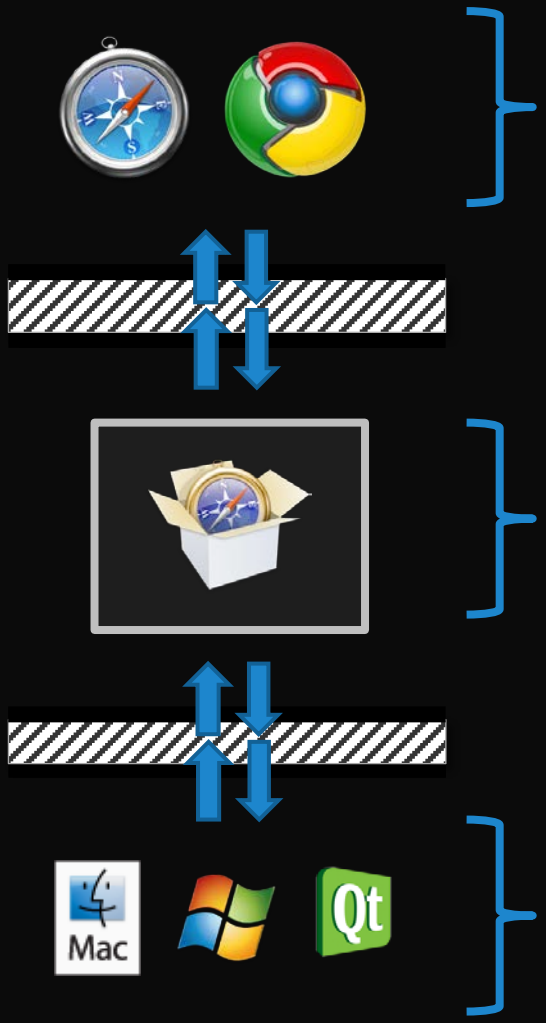
Timelapse doesn’t tame all sources of nondeterminism (yet).

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



User: mouse, keyboard, scroll, resize

Network: images, scripts, HTML, AJAX

Commands: page navigation

Internal nondeterminism:
Animations, transitions, multimedia, async script and parser yields

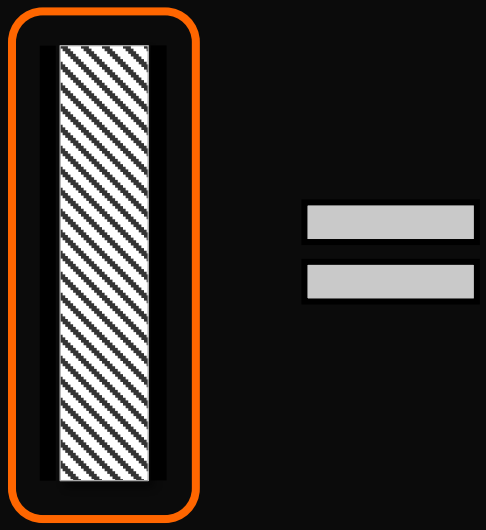
Functions: `Date.now, Math.random, etc`

Caching: resources, cookies

Timers: timer schedule

Shim: the thing in the middle

Shims are used to implement deterministic record/replay.



```
#if ENABLE(TIMELAPSE)
static double jsRiggedCurrentTime(JSGlobalObject* globalObject)
{
    if (RefPtr<DeterminismLog> log = globalObject->determinismLog()) {
        double currentTime;

        if (log->capturing()) {
            currentTime = jsCurrentTime();
            log->append(new GetCurrentTime(currentTime));
        } else {
            ASSERT(log->replaying());
            GetCurrentTime* action = log->currentAction<GetCurrentTime>();
            currentTime = action->currentTime();
        }
        return currentTime;
    }

    //if no determinism shenenigans are going to happen
    return jsCurrentTime();
}
#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.

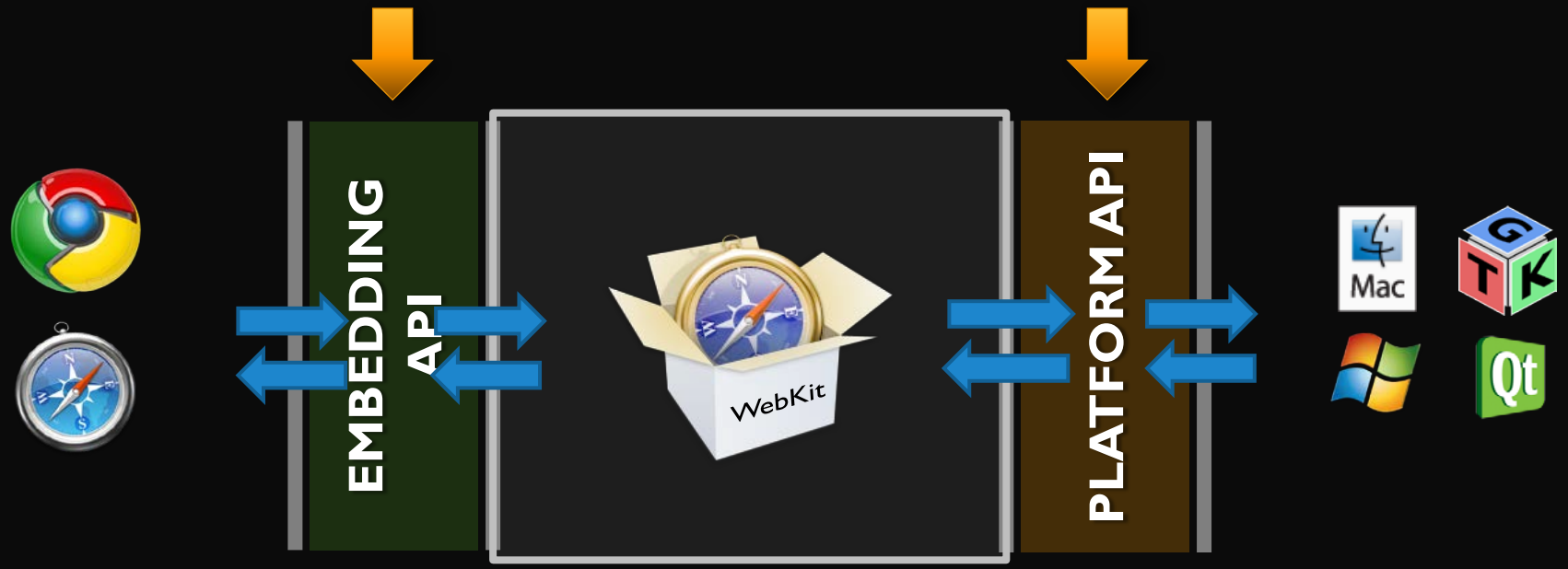
Embedders

Web Interpreter (WebKit, Gecko)

Platforms

Embedding and platform APIs

Abstraction layers separate web interpreters from platforms/embedders.



Embedders

Web Interpreter (WebKit, Gecko)

Platforms

Embedding and platform APIs

Shims sit between the web interpreter and abstraction layers.

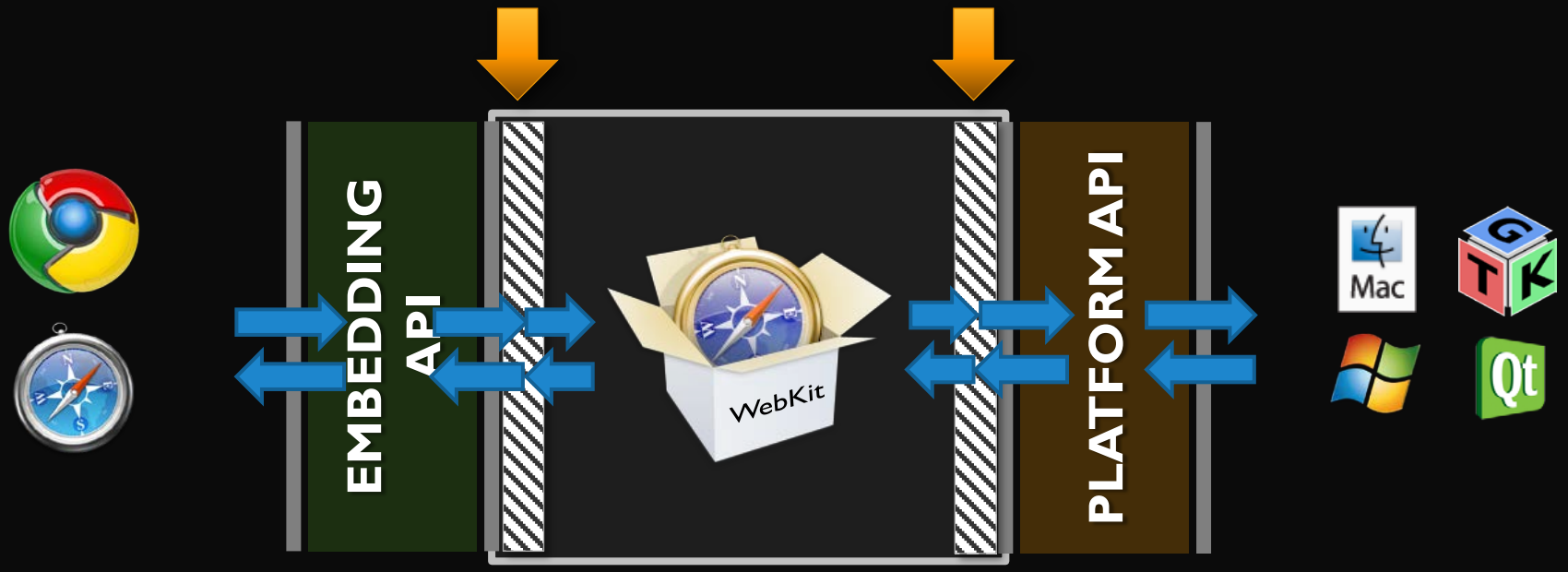
Embedders

Web Interpreter (WebKit, Gecko)

Platforms

Embedding and platform APIs

Shims sit between the web interpreter and abstraction layers.



Embedders

Web Interpreter (WebKit, Gecko)

Platforms