
TO WHOM
IT MAY
CONCERN:
James Mickens speaks the
TRUTH

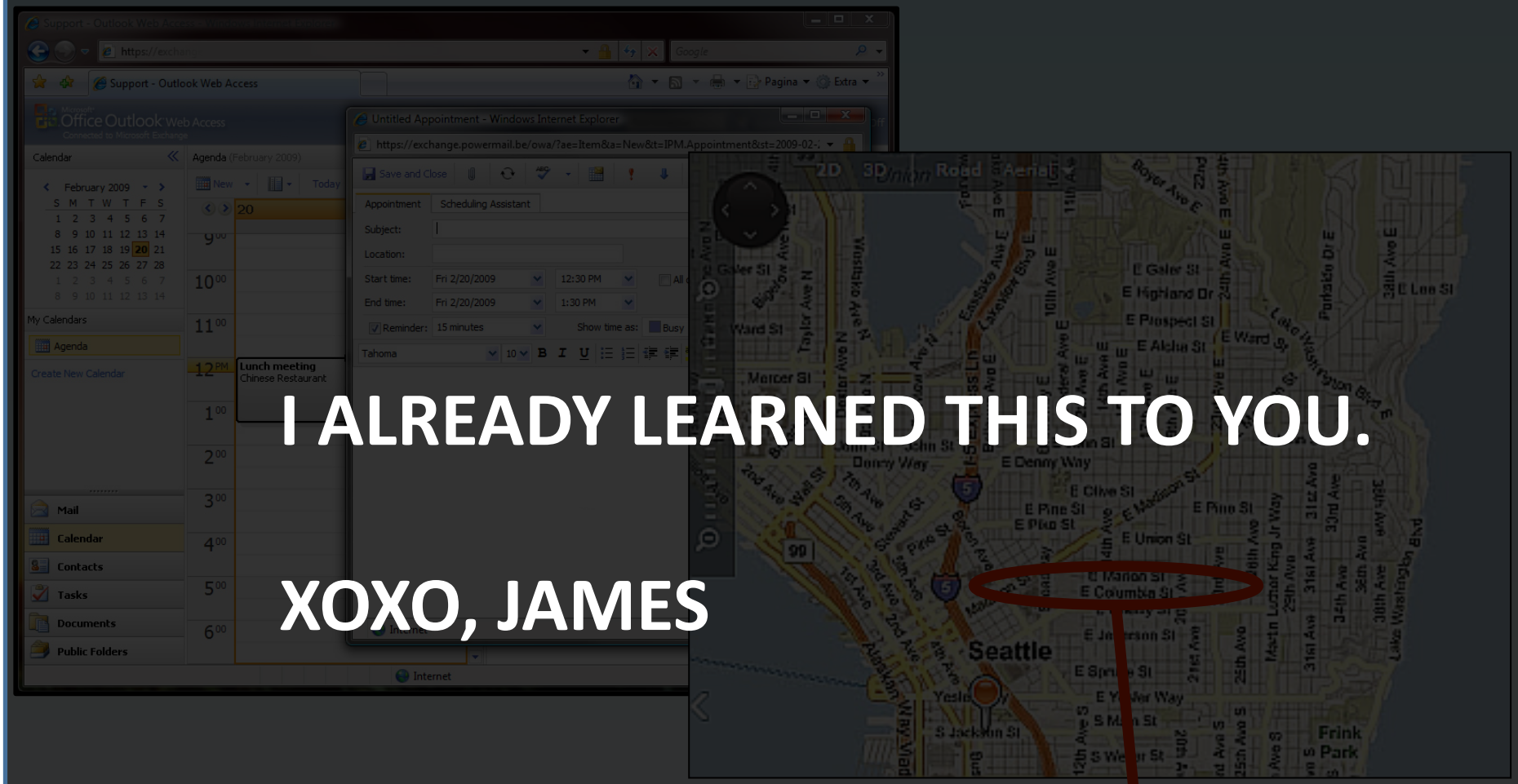
Mugshot: Recording and Replaying JavaScript Applications


James Mickens


Jeremy Elson


Jon Howell

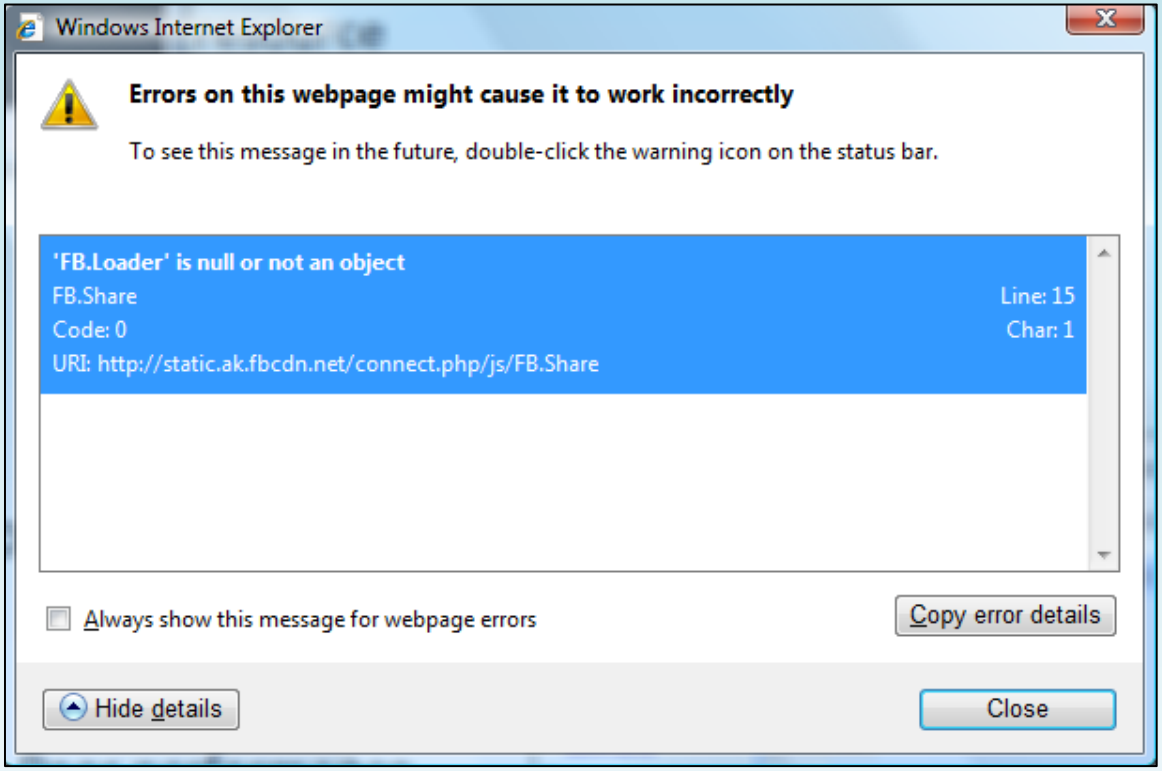
Microsoft®
Research

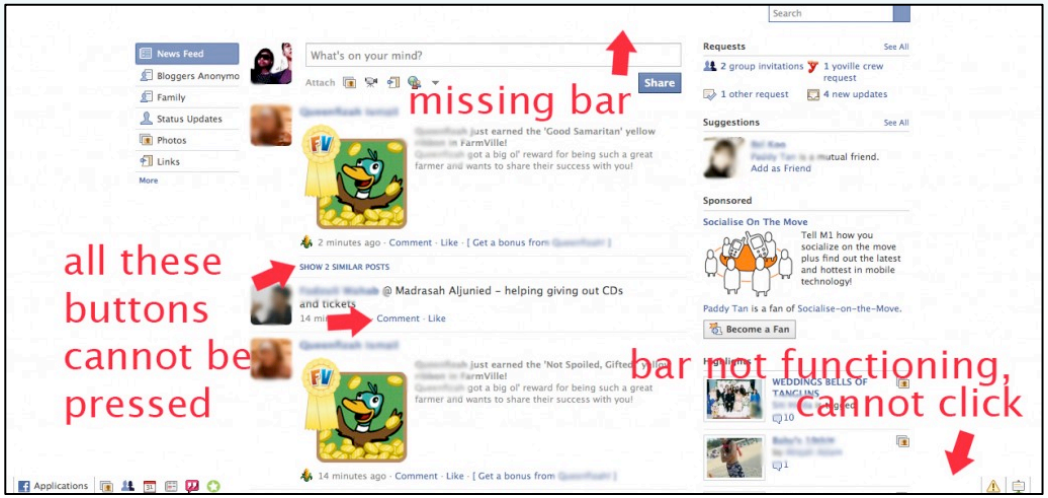


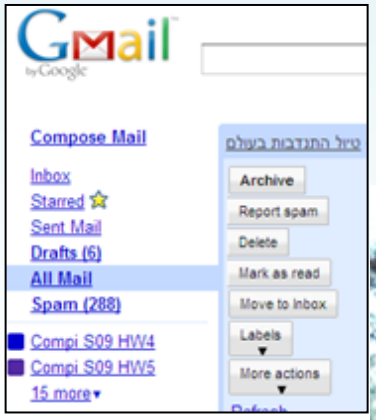
Modern web sites: event-driven functionality via JavaScript

```
mapItem.onclick = function(){
        mapControl.zoomLevel++;
        mapControl.fetchTiles();
        mapControl.displayTiles();
};
```

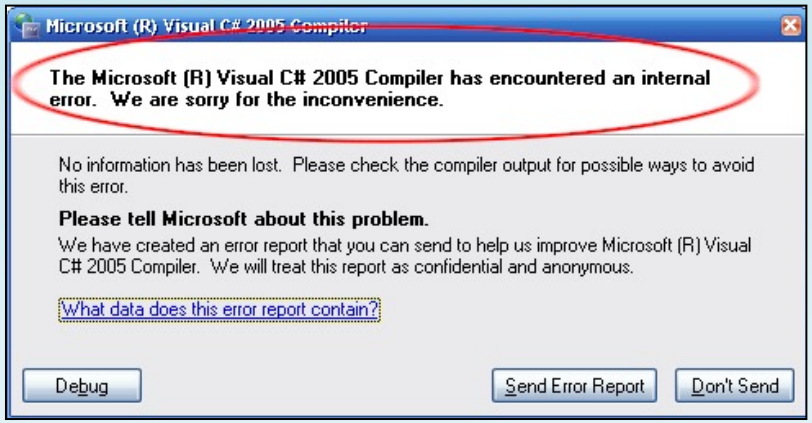
- "Hard" errors
  - Unexpected exception
  - Missing resource
- "Soft" errors
  - Layout glitch
  - Broken event handler
  - Poor performance





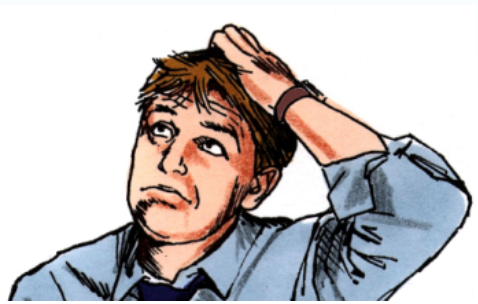


When Things Go Wrong



- Common post-mortems
  - Core dump
  - Stack trace
  - Error log

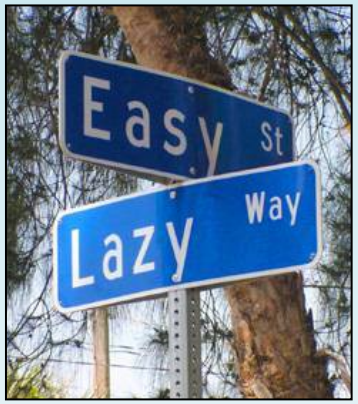
- In event-driven systems…
  - … interleavings are key!
  - Shouldn't rely on user to report nondeterministic events



Our Solution: Mugshot

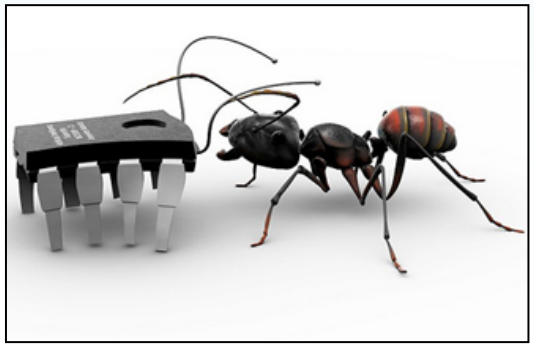
- Logs nondeterministic JavaScript events
  - Ex: Mouse clicks, date requests, random number generation
- On panic, upload event log to developer machine
- Developers replays the buggy program run
  - Single step or (near) real-time playback
  - Developer can leverage rich localhost debuggers . . .
  - . . . using buggy applications runs from the wild!

Why Mugshot Is Awesome



- Easy to deploy to end users
  - Logging/replay code is just a JavaScript library
  - Ship Mugshot infrastructure with the application:
    <script src="mugshot.js"></script>
  - Don't need special kernel/VM/browser!

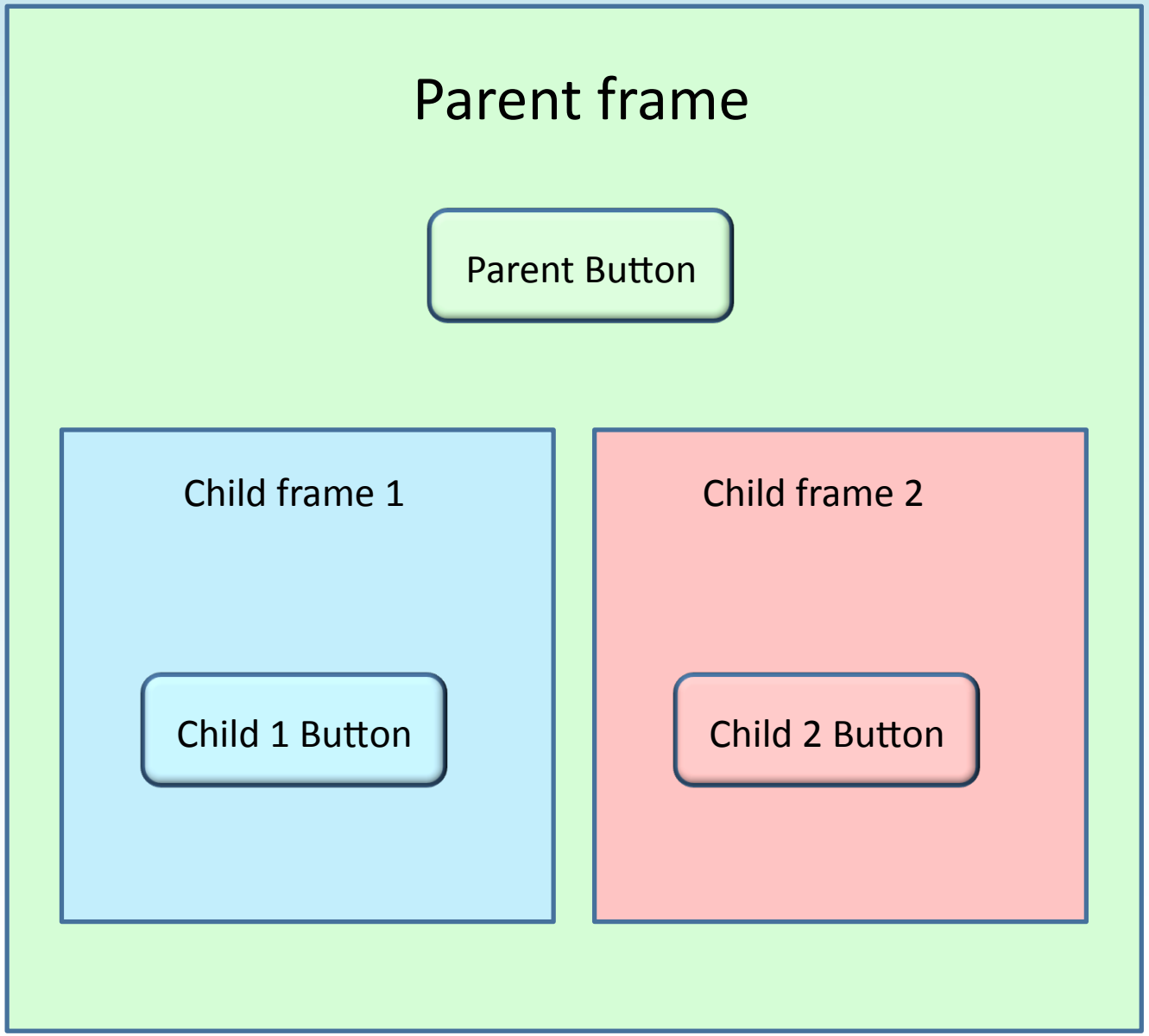
- Logging is lightweight: run in common case
  - Log size: Worst case 16 Kbps
  - CPU: Worst case 7% reduction in frame rate

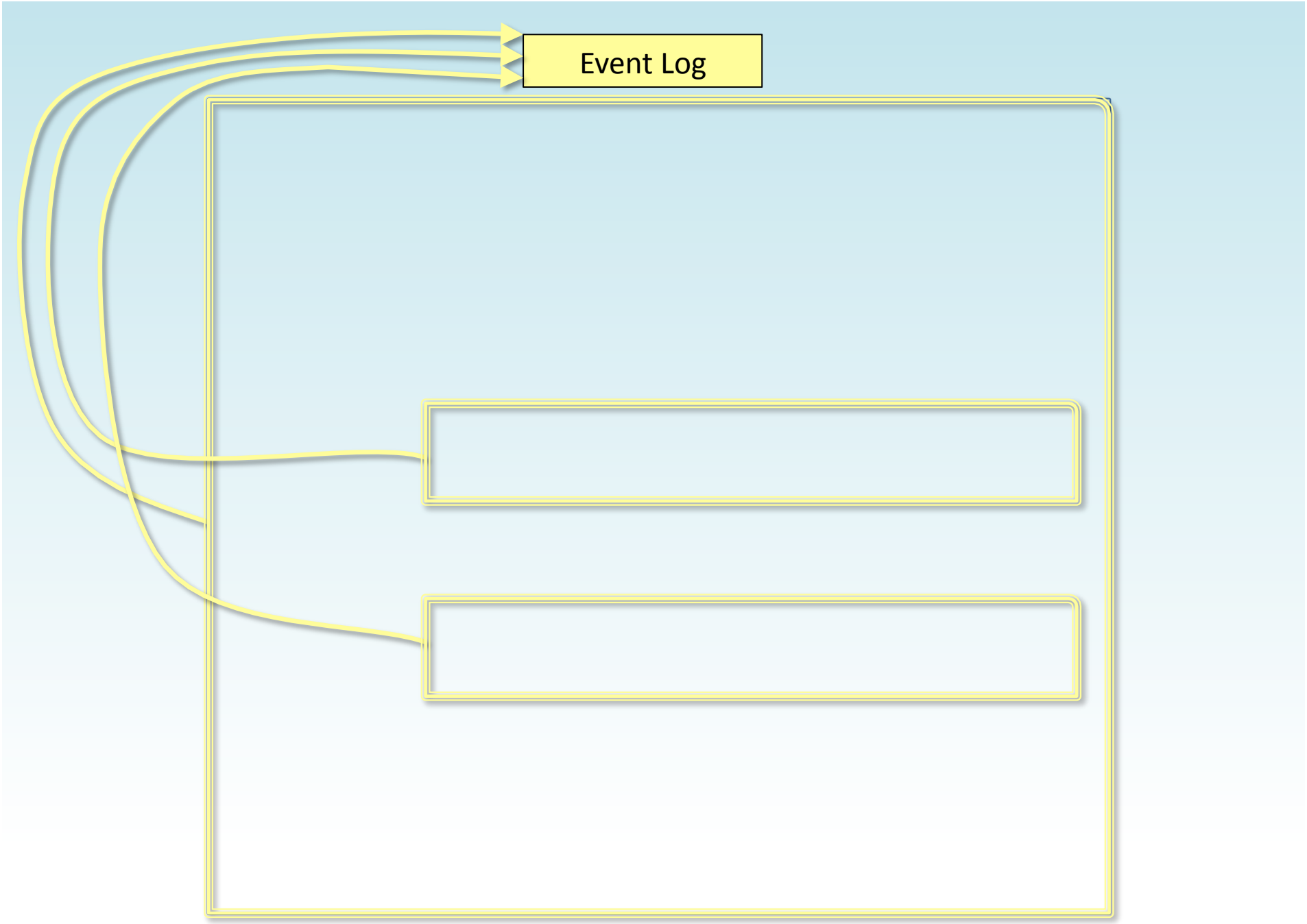


- Solves an important, practical problem
  - Increasingly complex apps migrating to the web
  - Remote bug repro is very important!

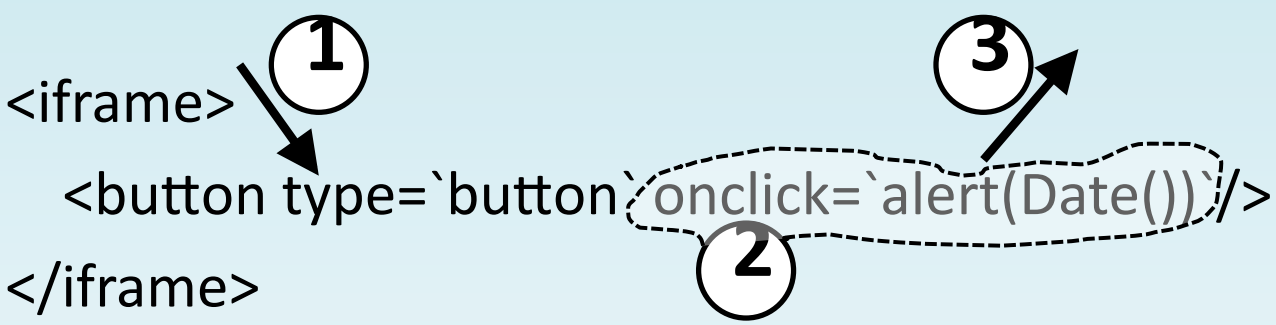
Outline

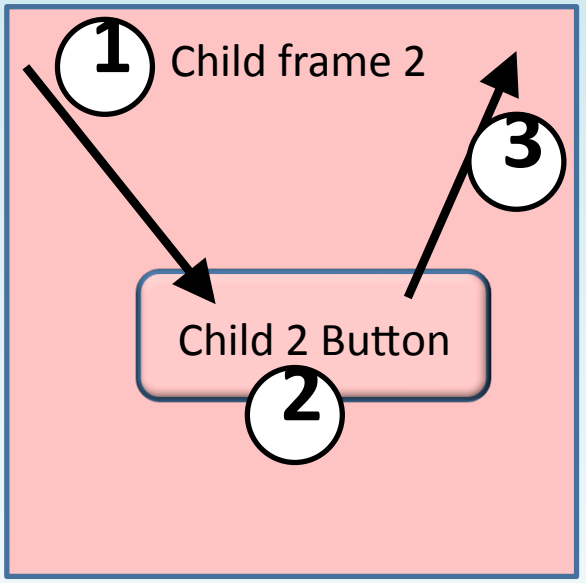
- Logging
- Replay
- Evaluation
- Conclusion


Parent frame
Parent Button
Child frame 1
Child 1 Button
Child frame 2
Child 2 Button


Event Log

“Official” W3C Event Model

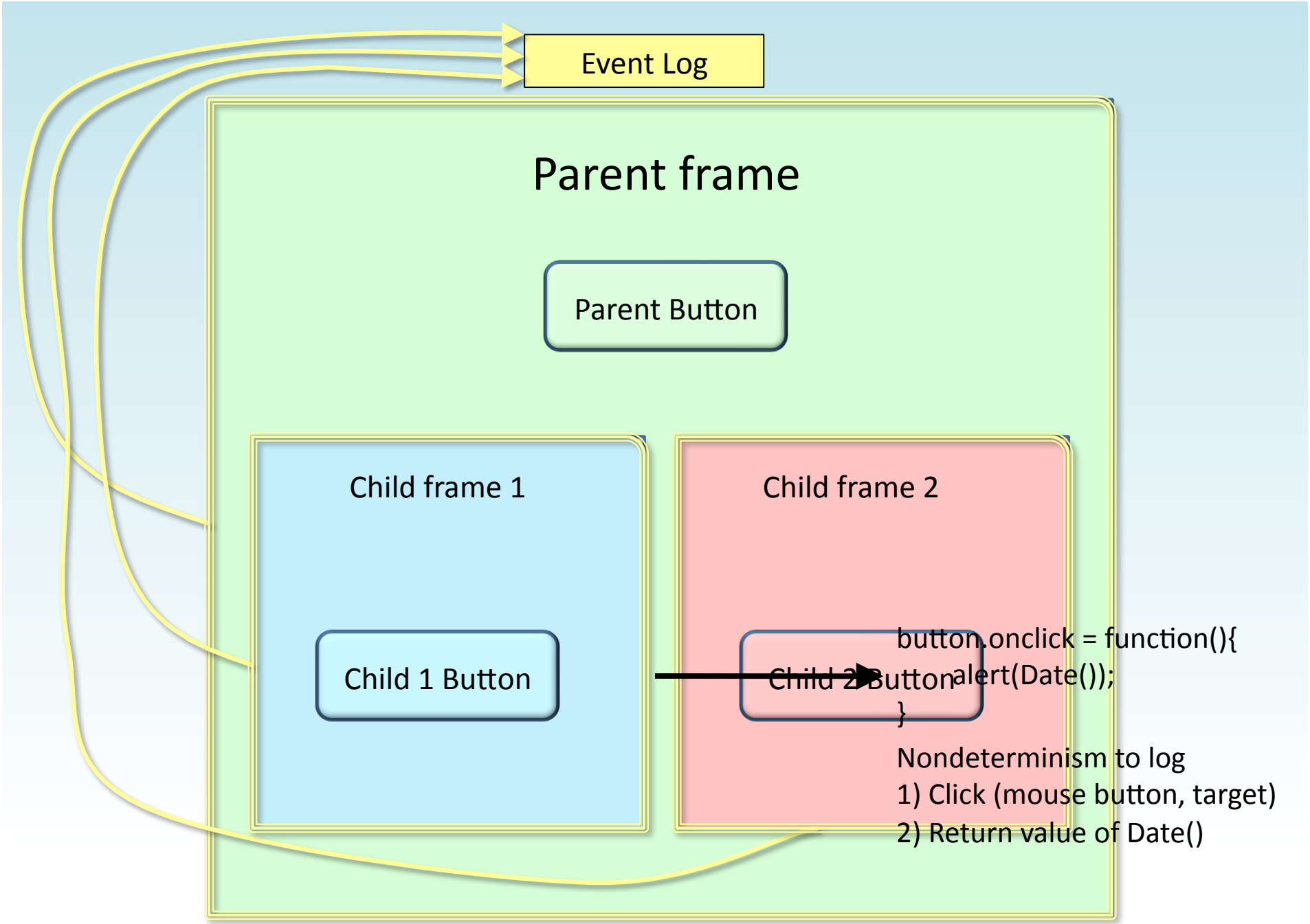




Phase 1: Capturing

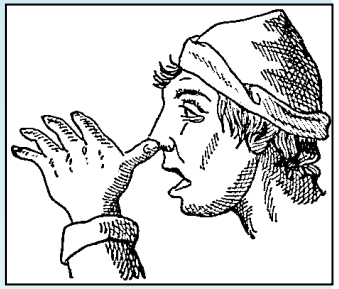
Phase 2: Target

Phase 3: Bubbling


Event Log
Parent frame
Parent Button
Child frame 1
Child 1 Button
Child frame 2
Child 2 Button
button.onclick = function(){
alert(Date());
}
Nondeterminism to log
1) Click (mouse button, target)
2) Return value of Date()

Logging Events on Firefox

- Logging Date() is straightforward . . .
  - . . . just enclose real Date() in logging wrapper
- Logging mouse click is “straightforward”



1) Capturing phase (iframe)

2) Target phase (button)

```
<iframe>onclick=“mugshotCapturingLogger()” >
        <script src=“logger.js”></script>
        <button type=“button” onclick=“alert(Date())”>
</iframe>
```

Event log

Event: Click
Time: 1000
Value: Child 2 button
Left-click
X=312, Y=209

Event: Date
Time: 2000

Seems simple, right?


IT'S
A
TRAP

DOM 0 versus DOM 2 Handlers

```
var f = document.getElementById("child2frame");
f.onclick = function(){alert("DOM 0 handler")};
f.addEventListener("click",
                   function(){alert("DOM 1 handler"),
                   true);
```

- For any DOM node/event name pair:
  - At most one DOM 0 handler
  - Arbitrary number of DOM 2 handlers

Life Is So Difficult

- Firefox calls DOM 0 handler *before* DOM 2 handlers
  - DOM 2 handlers called in order of registration
- Mugshot must ensure that its handler runs *before* any app-defined ones
  - App handler can cancel event . . .
  - . . . but we still need to log it!

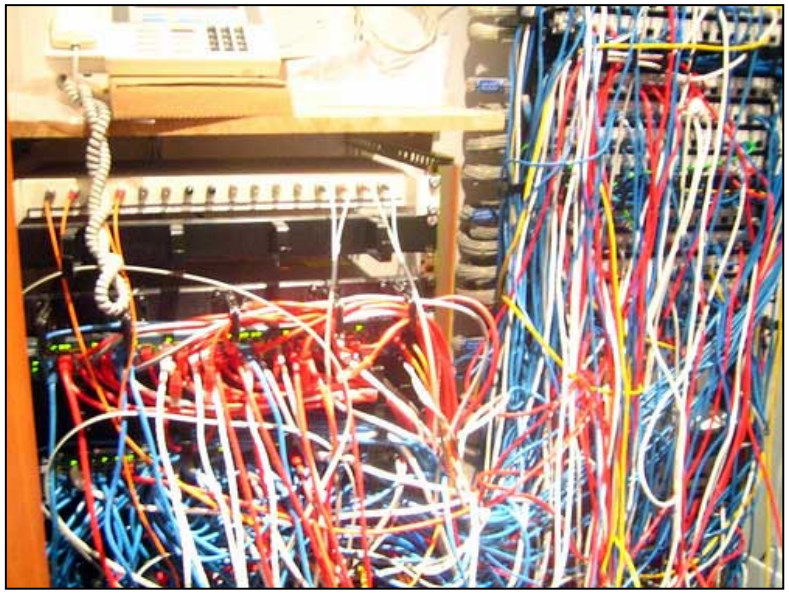
Life Is So Difficult

- We'd like to run *before* the app and . . .
  - Define DOM 2 logging handler for onclick
  - Use JavaScript setter shim to interpose on assignment to iframe.onclick
- This would let us:
  - Use DOM 2 logging func if no app-defined DOM 0 handler
  - Wrap app-defined DOM 0 handler in logging code

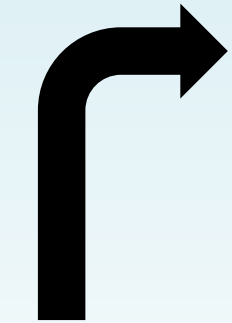
- The problem: Firefox setters are partially broken
  - Browser will not invoke DOM 0 handler for node with a shimmed DOM 0 event property

Life Is So Difficult

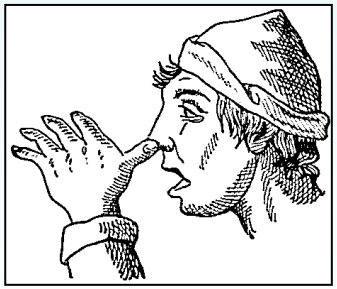
- Fortunately, setters for DOM 0 handlers don't keep browser from firing DOM 2 handlers
  - So, setter code registers DOM 0 app handler as DOM 2 handler too
  - Setter removes DOM 2 handler if "backing" DOM 0 handler is reset



Recap: Logging Events on Firefox



```
<iframe onclick="mugshotCapturingLogger()" >
        <script src="logger.js"></script>
        <button type="button" onclick="alert(Date())">
</iframe>
```



Strategery

Logging Events on IE

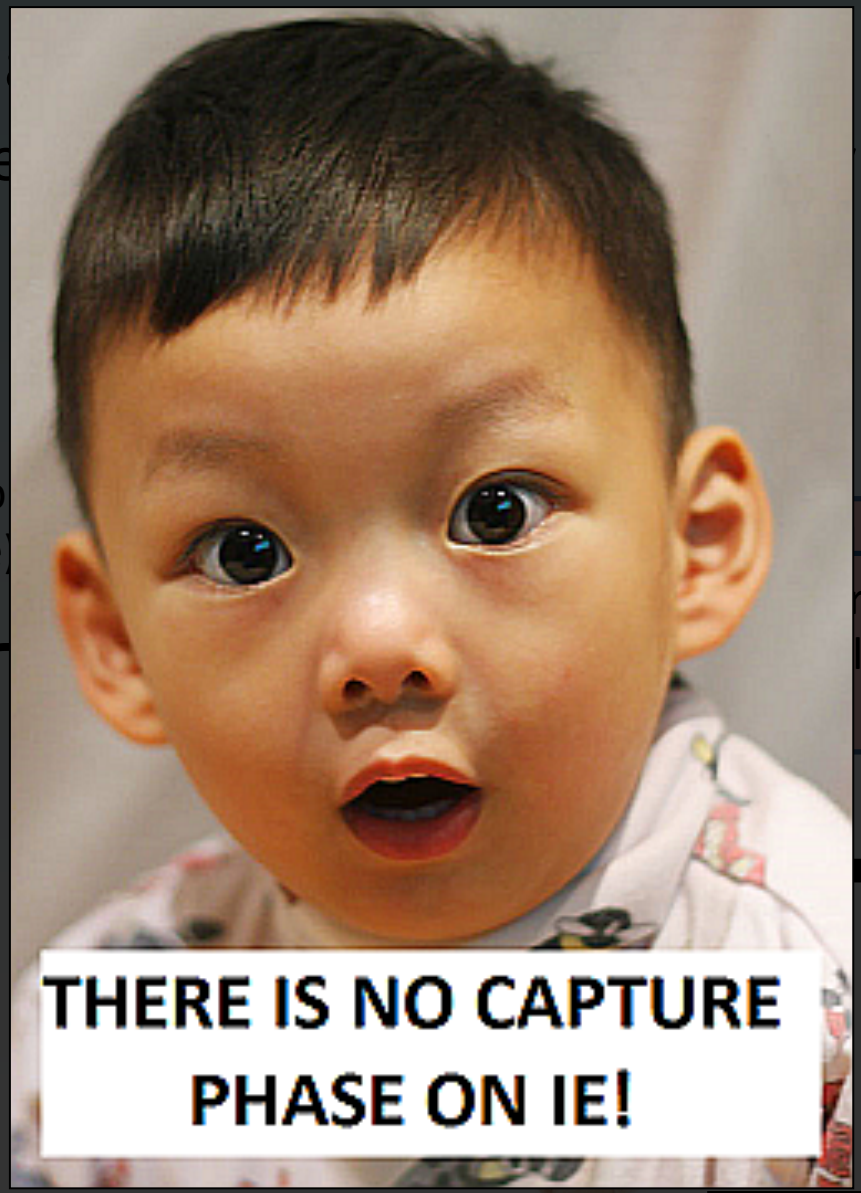
- Logging D
  - – . . . just e rapper
- Logging G
  - – There is

2) Bubbling p
(iframe)

1) Target phase
(button)

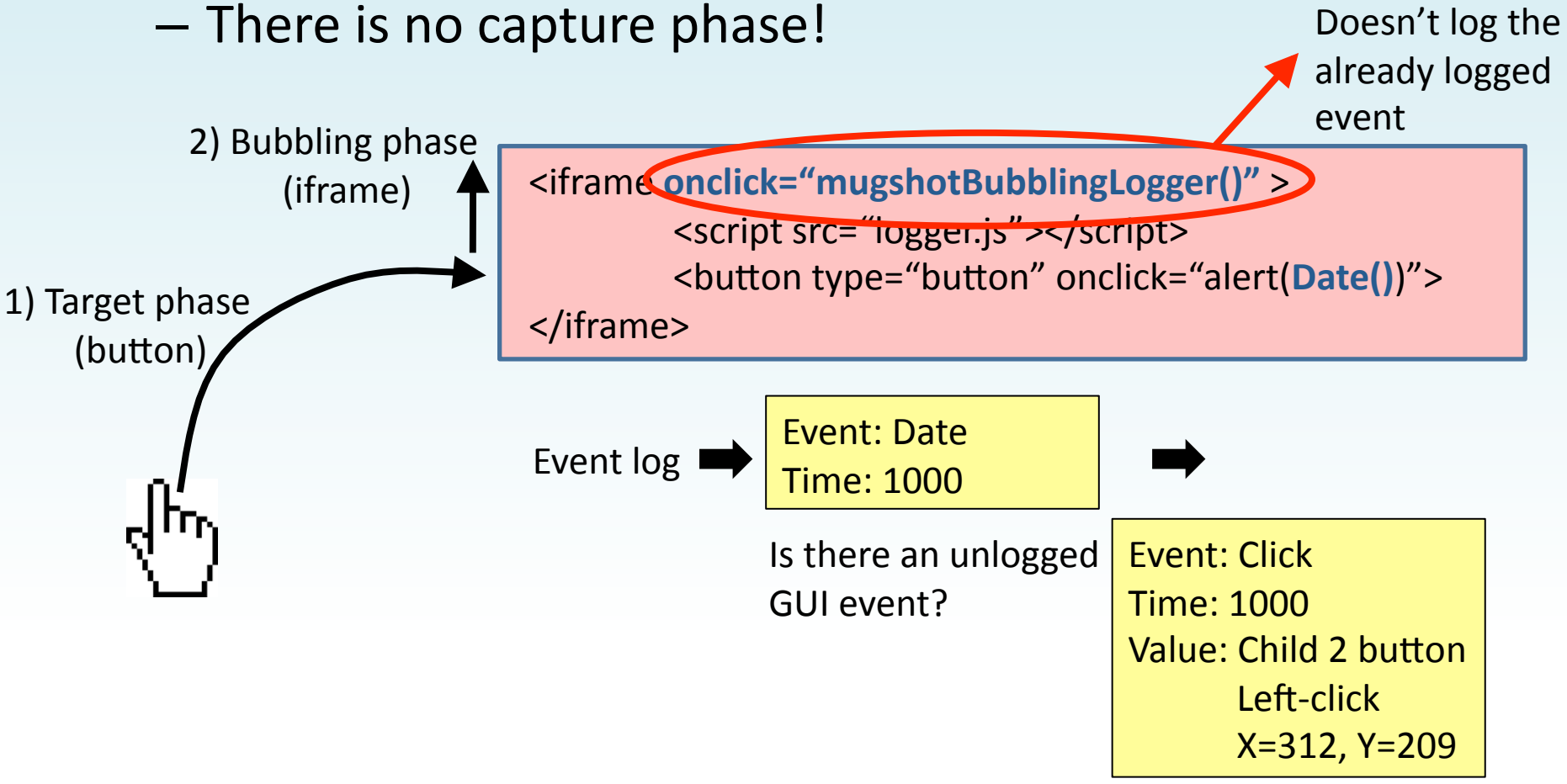
```
Logger()" >
ript>
lick="alert(Date())">
```

Event: Click
Time: 2000
Value: Child 2 button
Left-click
X=312, Y=209



Logging Events on IE

- Logging Date() is straightforward . . .
  - . . . just enclose real Date() in logging wrapper
- Logging GUI events is tricky in IE!
  - There is no capture phase!



Sources of Nondeterminism

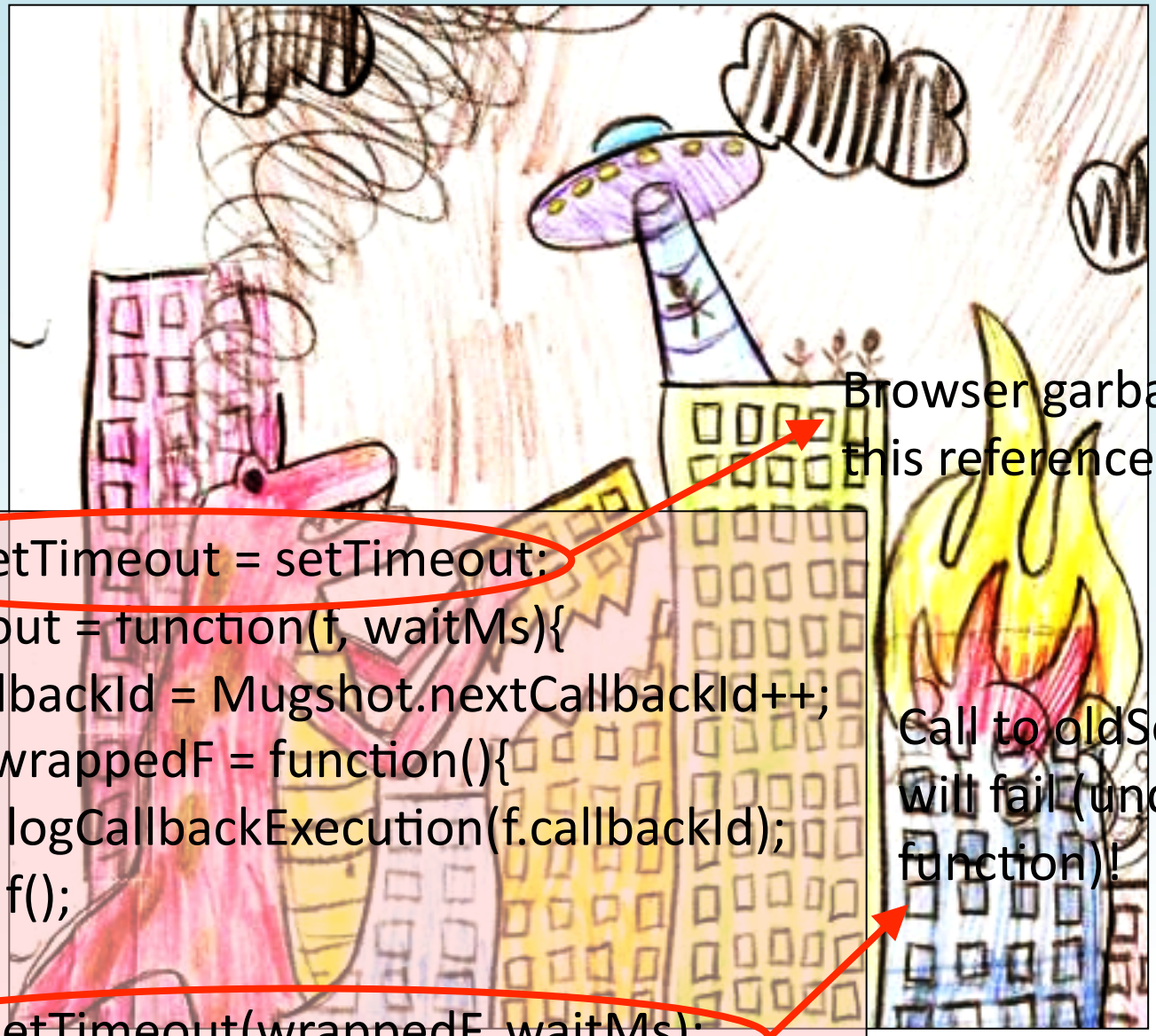
Category	Event Type	Example
DOM Events	Mouse Key Loads Form Body	click, mouseover keydown, keyup load focus, blur, select scroll, resize
Asynchronous callbacks	Set timer AJAX state change	setTimeout(f, 100) req.onchange = f
Nondeterministic functions	Get current time Get random number	(new Date()).getTime() Math.random()
Text selection	IE: document.selection FF: window.getSelection()	Highlight text w/mouse Highlight text w/mouse

How Do We Log “setTimeout(f, 50)”?

- Interpose on setTimeout()

```
var oldSetTimeout = setTimeout;
setTimeout = function(f, waitMs){
        f.callbackId = Mugshot.nextCallbackId++;
        var wrappedF = function(){
                logCallbackExecution(f.callbackId);
                f();
        };
        oldSetTimeout(wrappedF, waitMs);
};
```

- Easy, right?



Browser garbage-collects this reference!?!

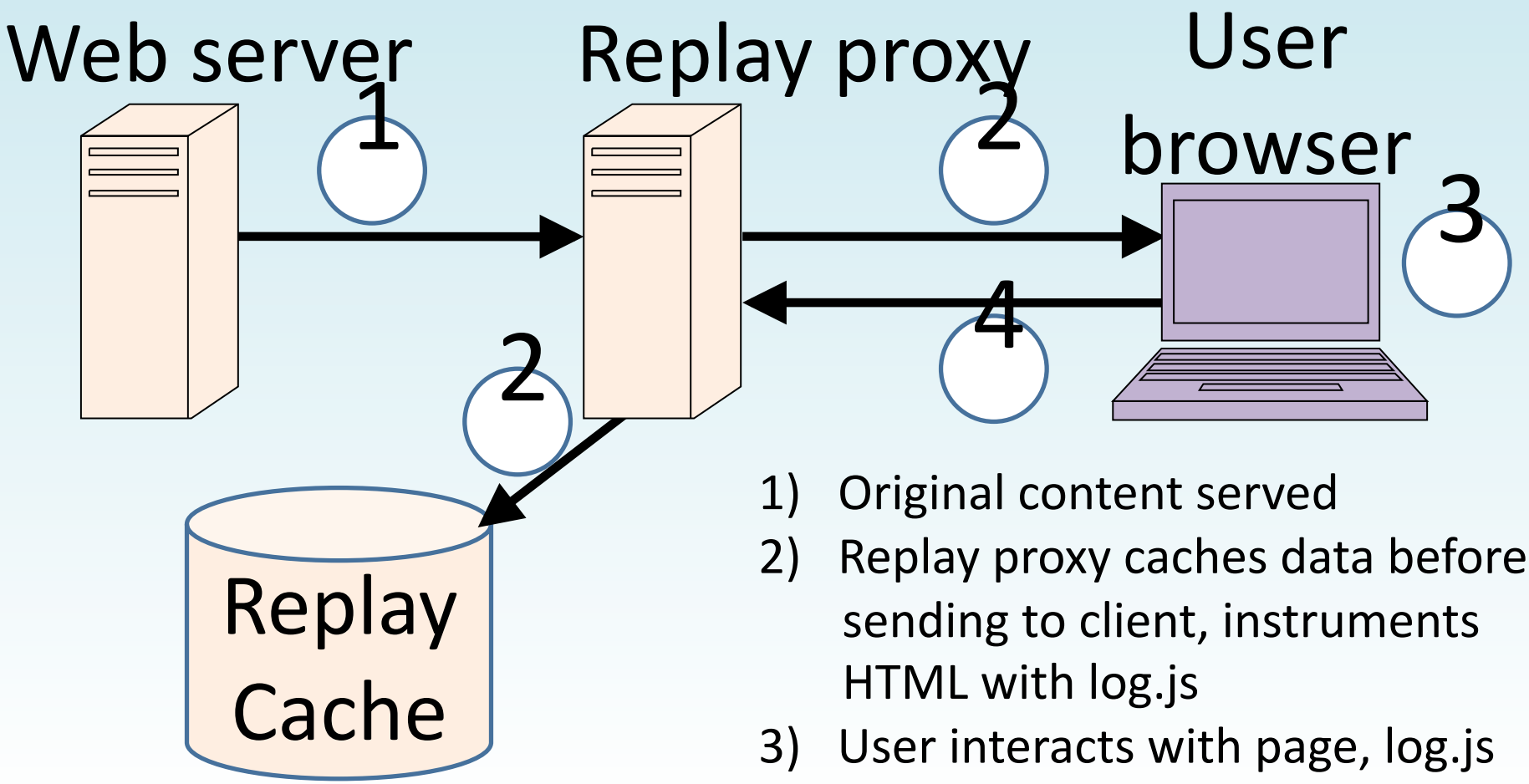
```
var oldSetTimeout = setTimeout;
setTimeout = function(f, waitMs){
        f.callbackId = Mugshot.nextCallbackId++;
        var wrappedF = function(){
                logCallbackExecution(f.callbackId);
                f();
        };
        oldSetTimeout(wrappedF, waitMs);
};
```

Call to oldSetTimeout() will fail (undefined function)!

I Hate Myself And I Want To Die

- Solution: Create an invisible iframe!
  - Save its reference to setTimeout() . . .
  - . . . and call it inside the wrapped callback
- Have to do this nonsense at replay time too
- Mugshot uses a variety of additional hacks
  - See the paper for details

Logging the Value of Loads

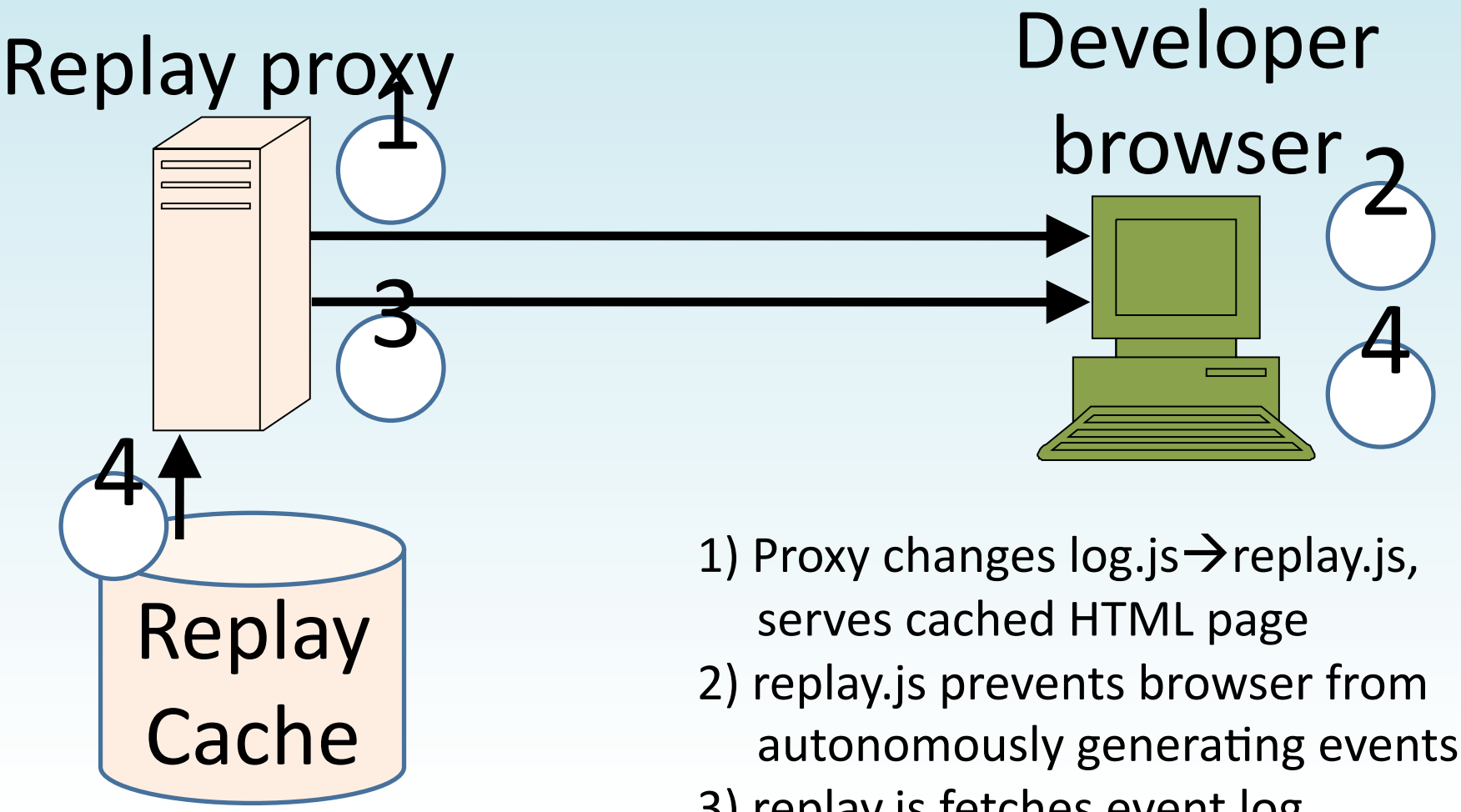


1) Original content served
2) Replay proxy caches data before sending to client, instruments HTML with log.js
3) User interacts with page, log.js records local events
4) On failure, log.js uploads event trace

Outline

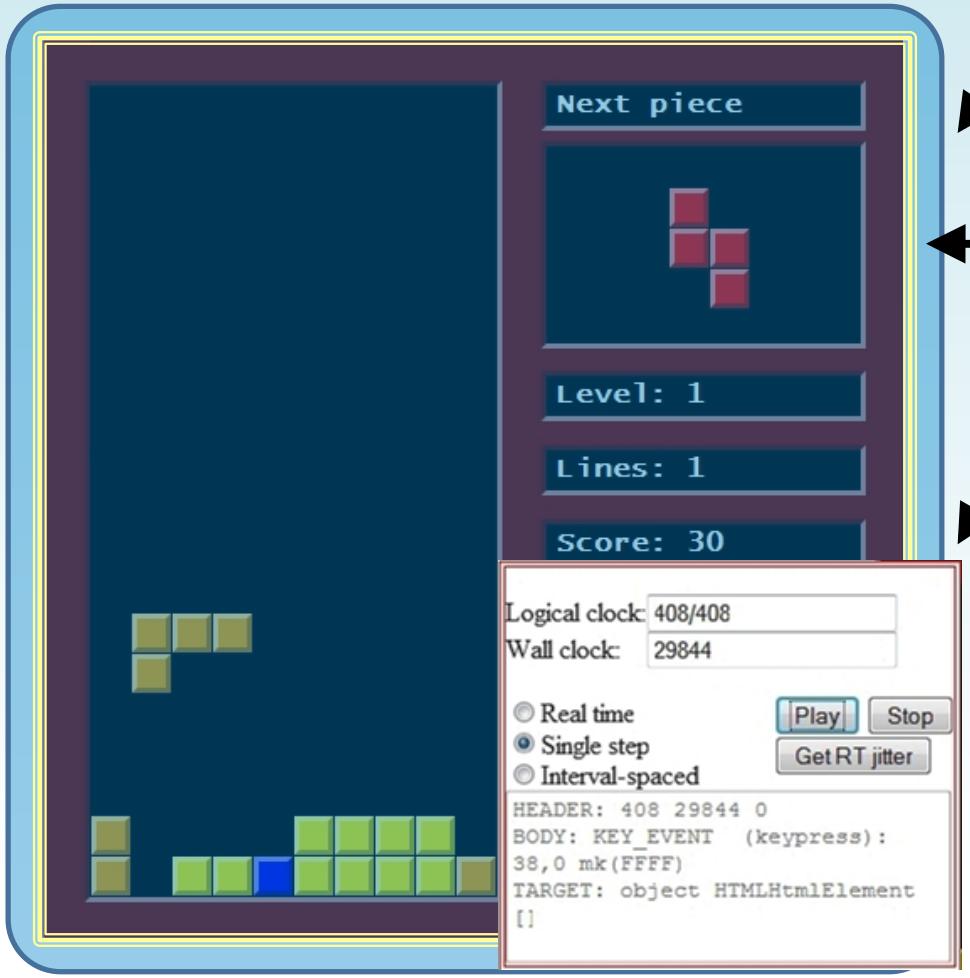
- Logging
- Replay
- Evaluation
- Conclusion

Using the Replay Proxy



1) Proxy changes log.js→replay.js, serves cached HTML page
2) replay.js prevents browser from autonomously generating events
3) replay.js fetches event log
4) replay.js replays events, fetching external content from replay proxy

On The Developer Machine: replay.js



1) Put transparent <iframe> on top of page

2) Interpose on Date(), Math.random(), setTimeout()…

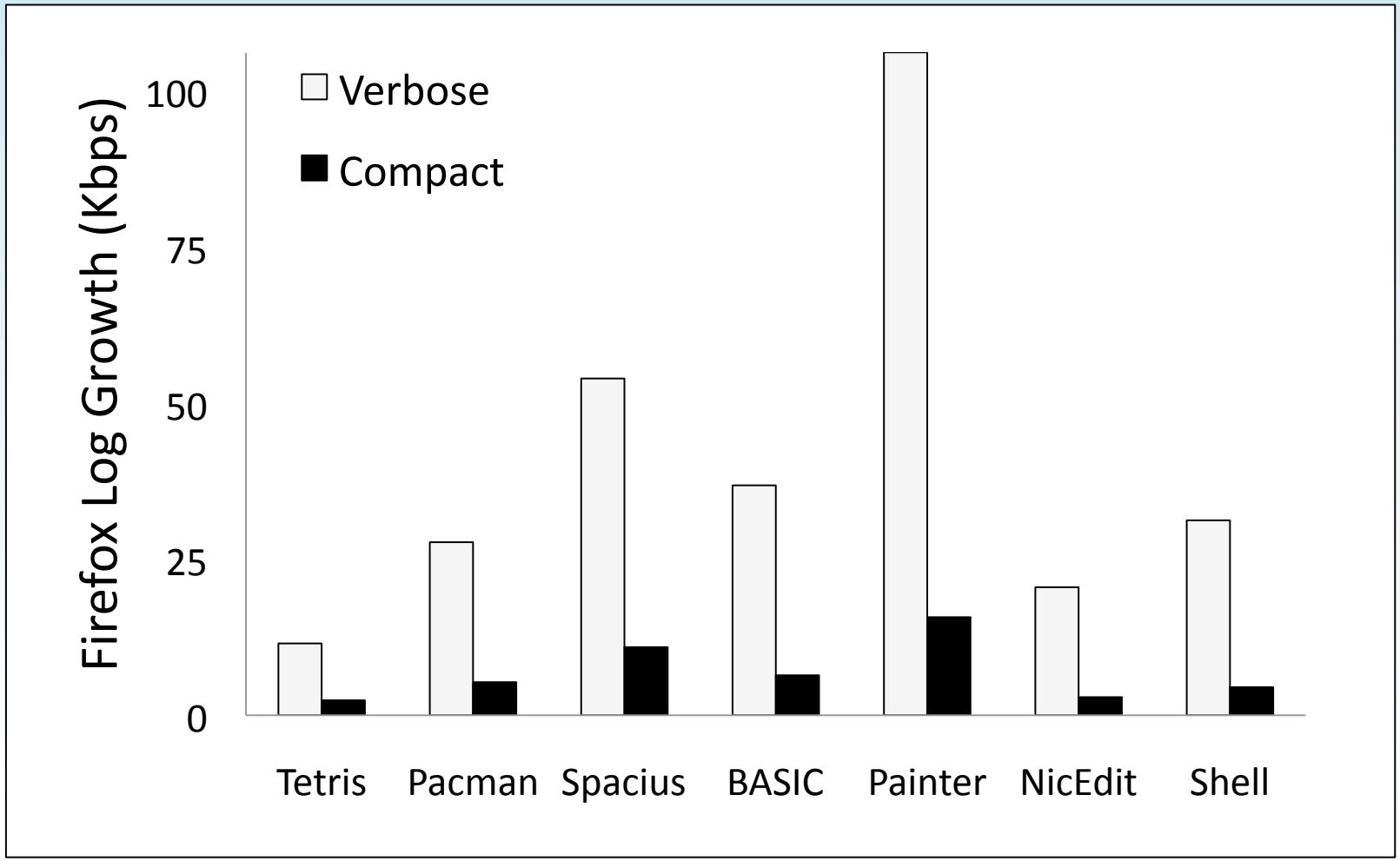
3) Fetch log and display VCR control

4) To replay, step through log . . .
   - Dispatch fake GUI events using fireEvent()/dispatchEvent()
   - Execute timer functions as they appear in log
   - As app code executes, pull return values of Date() and Math.random() from the log
   - When load arrives, signal replay proxy to release the data

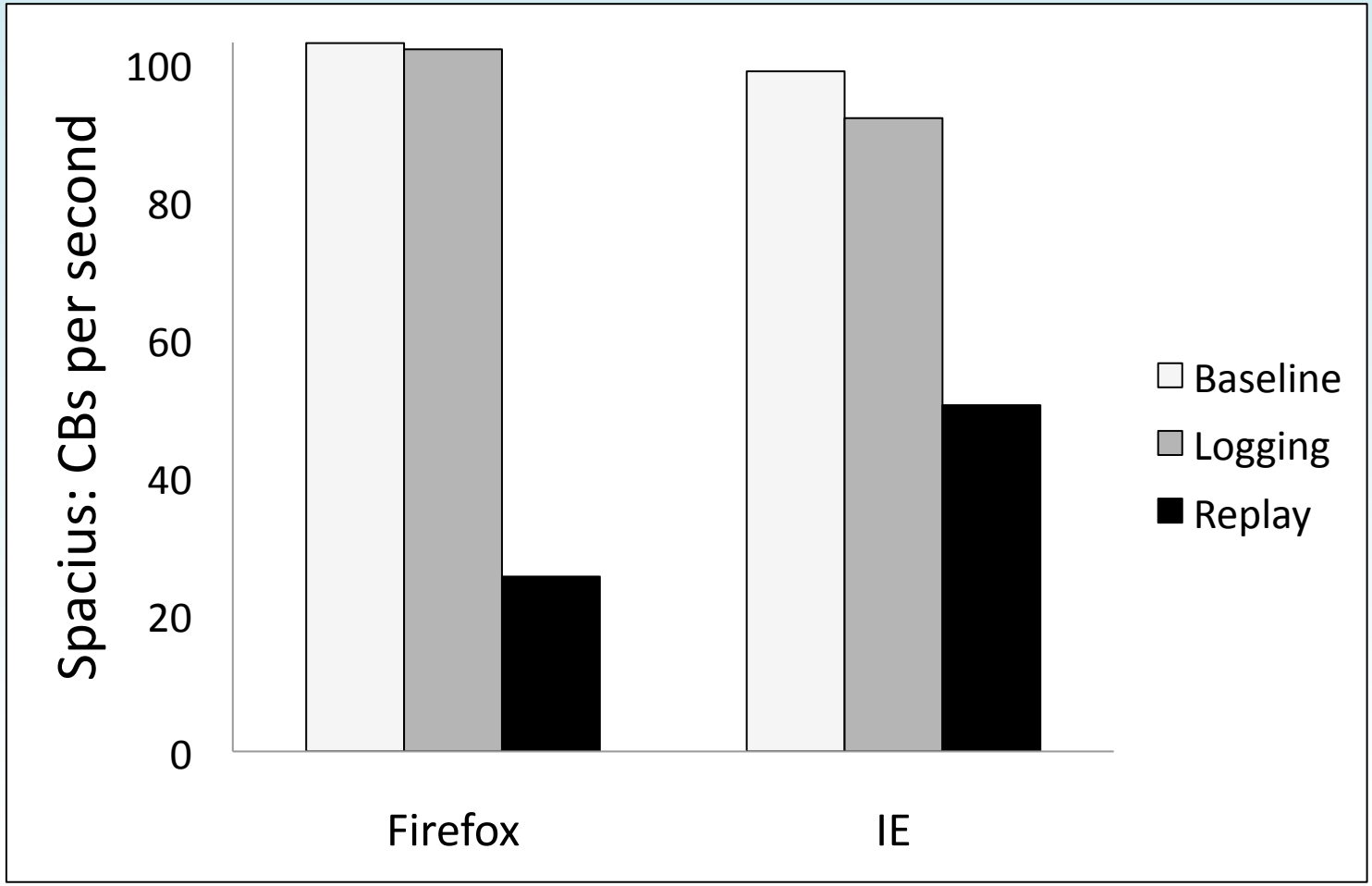
Outline

- Logging
- Replay
- Evaluation
- Conclusion

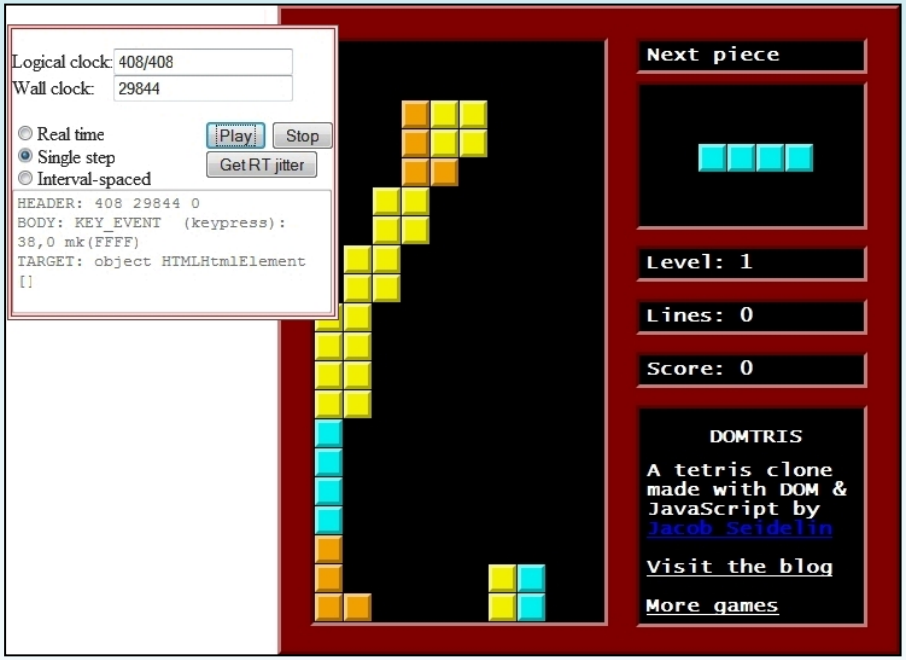
Log Growth

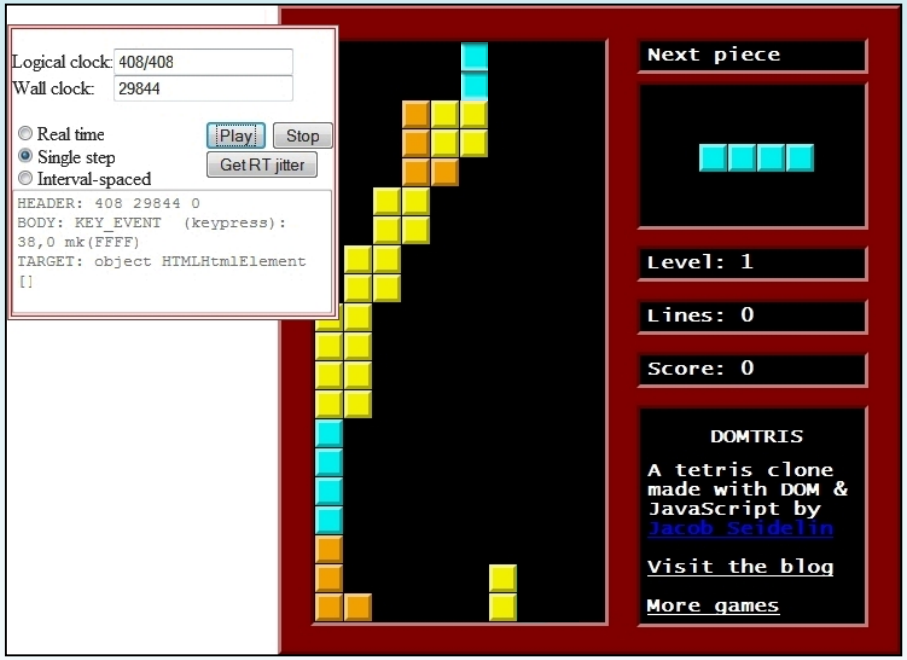

Firefox Log Growth (Kbps)
100
75
50
25
0
Verbose
Compact
Tetris
Pacman
Spacius
BASIC
Painter
NicEdit
Shell

Timer Callback Rates

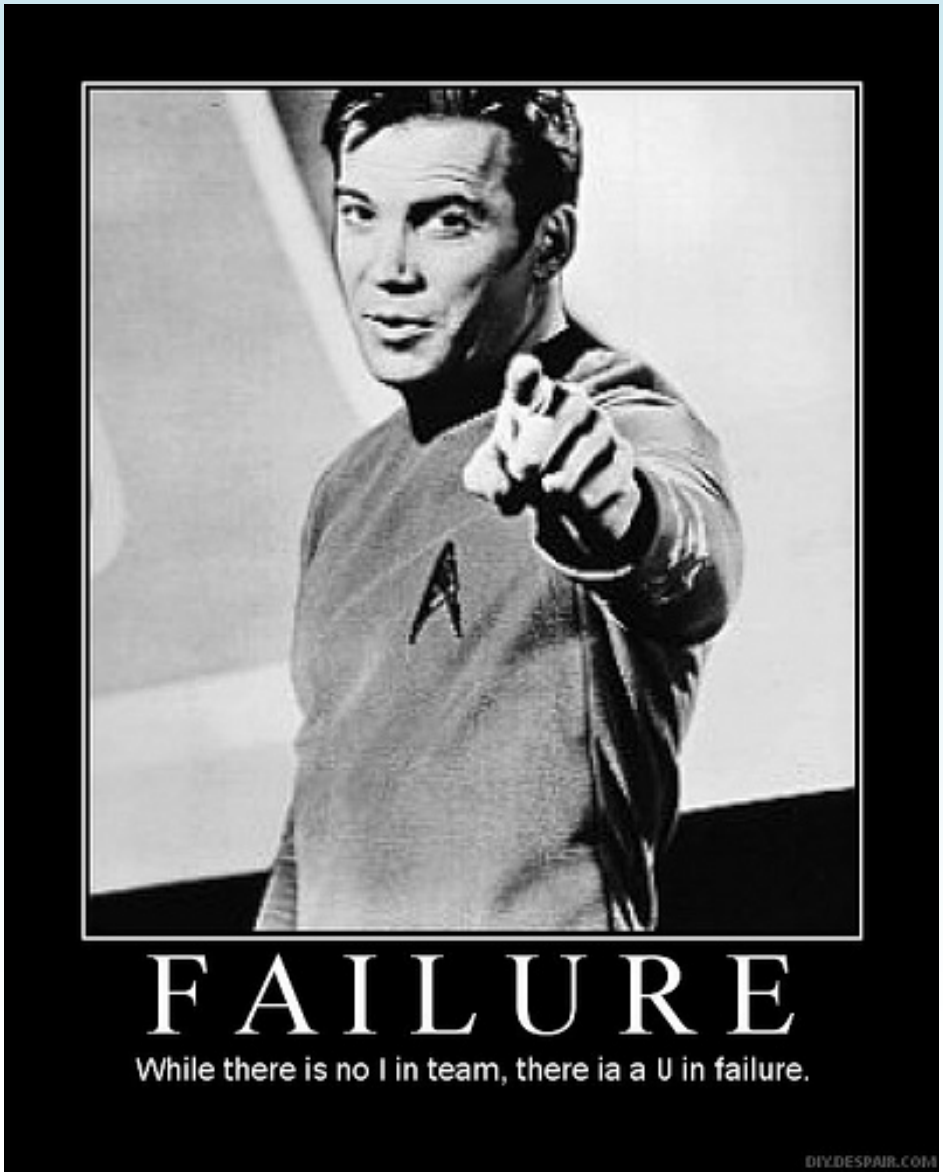


Reproducing Bugs





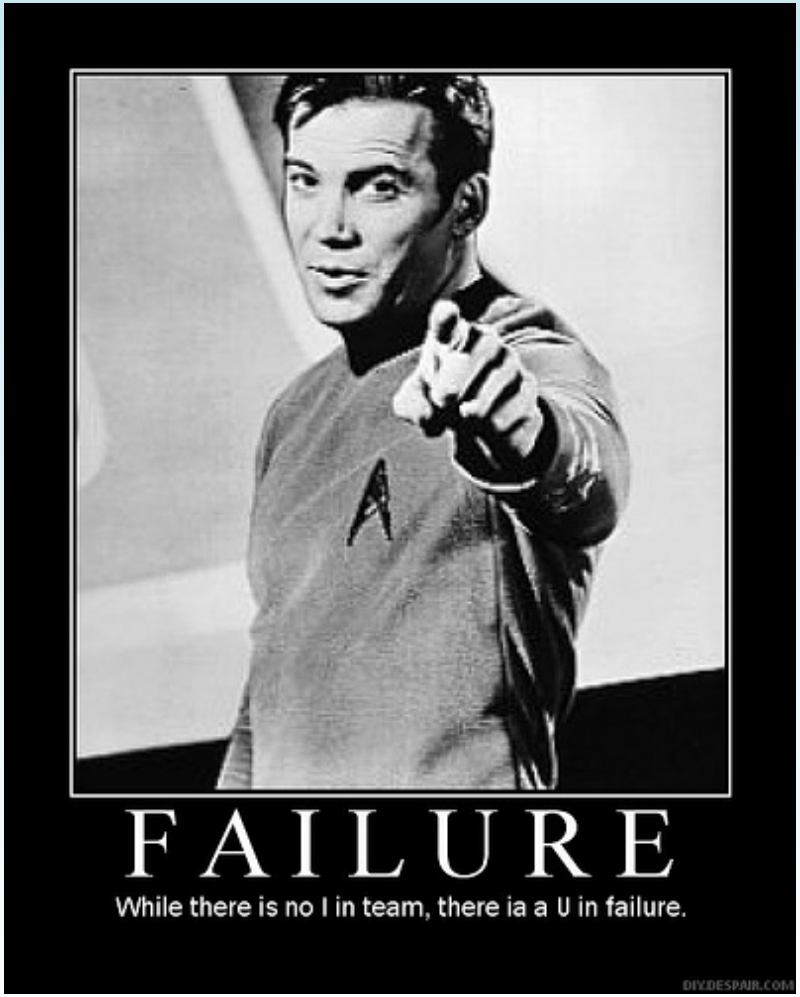
Conclusion

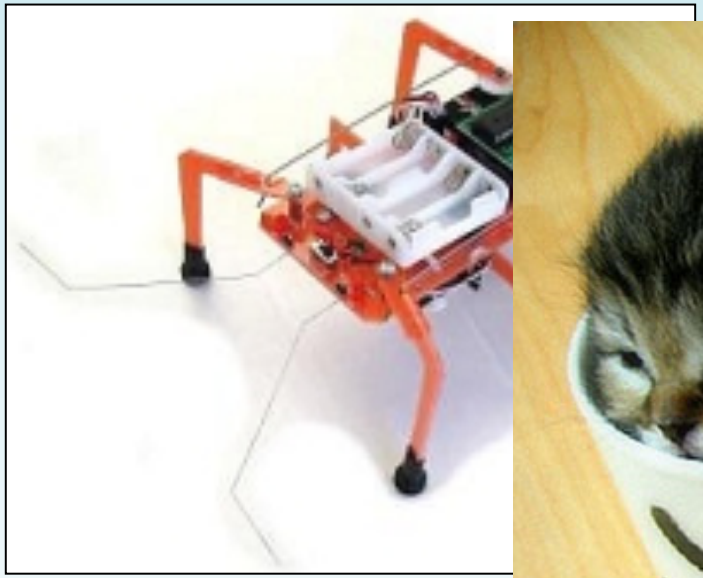

FAILURE
While there is no I in team, there ia a U in failure.
DIY.DESPAIR.COM

Conclusion

- Mugshot: trace+replay for JavaScript apps
  - Easy to deploy: run a script inside unmodified browser
  - Lightweight: 7% CPU overhead, 16 Kbps log growth
- Design is straightforward . . .
  - . . . but implementation is not!
  - Take my learnings, make them your own


LOGICAL AWESOME
My Codes Are Perfect


FAILURE
While there is no I in team, there ia a U in failure.
DIYDESPAIR.COM




BOLDT.US