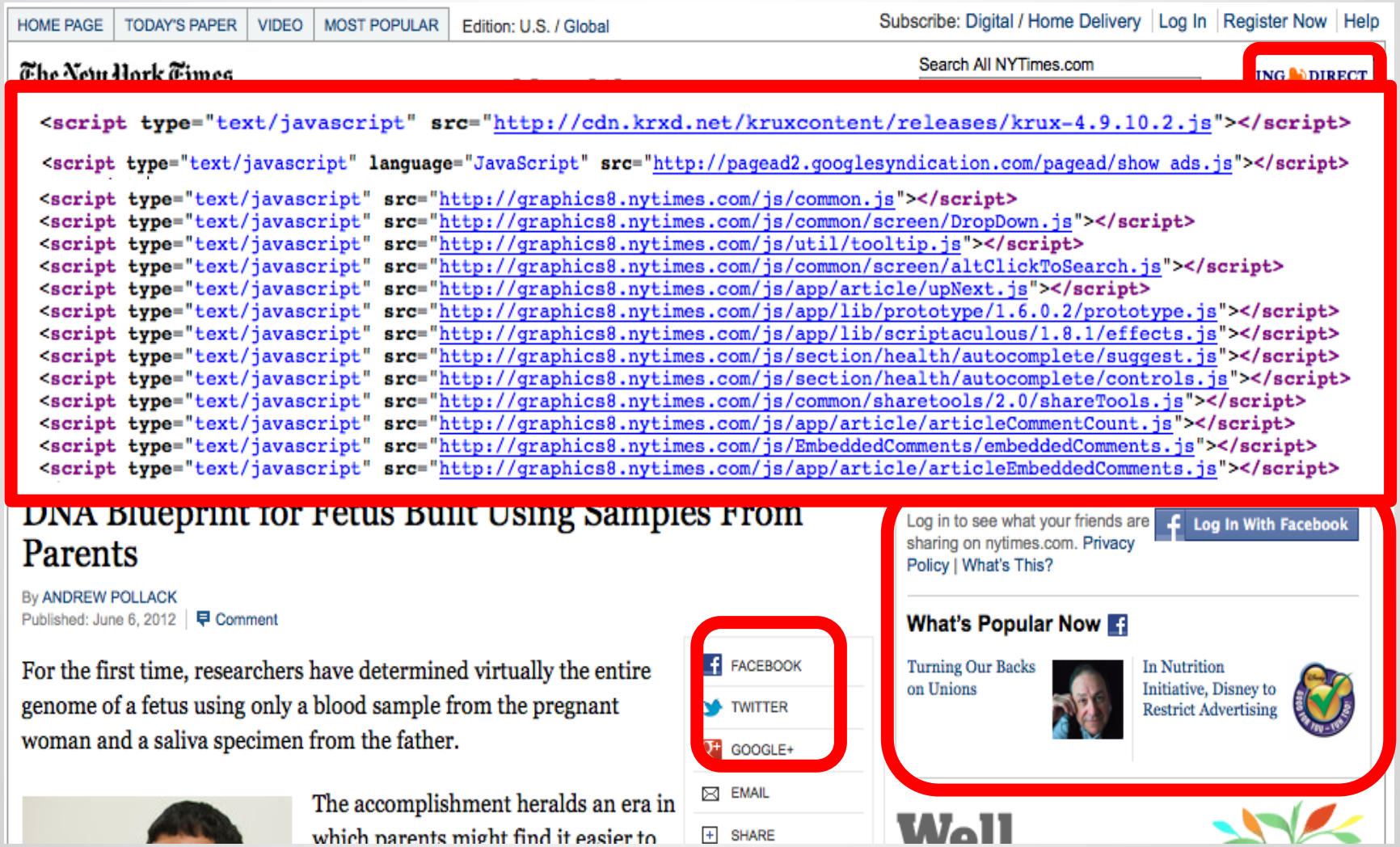
Jigsaw: Efficient, Low-effort Mashup Isolation

James Mickens and **Matthew Finifter**

Microsoft Research

UC Berkeley

The joy of creating a new webapp



Mashup Security

- Isolate third-party code
- Control what you share with it
- And make it easy for developers!

Talk Outline

- Previous approaches
- Goals
- Design
- Implementation
- Evaluation

Previous approaches: do nothing

+ Ease of development

- Zero isolation

Previous approaches: iframes plus postMessage

+ Standardized
+ Strong isolation
+ Simple string-based programming model

- Asynchronous programming model
- Need to layer on top of postMessage
- Performance overhead of object marshaling

Previous approaches: mashup isolation framework

Caja, Object Views, ConScript, ...

+ Built with security in mind

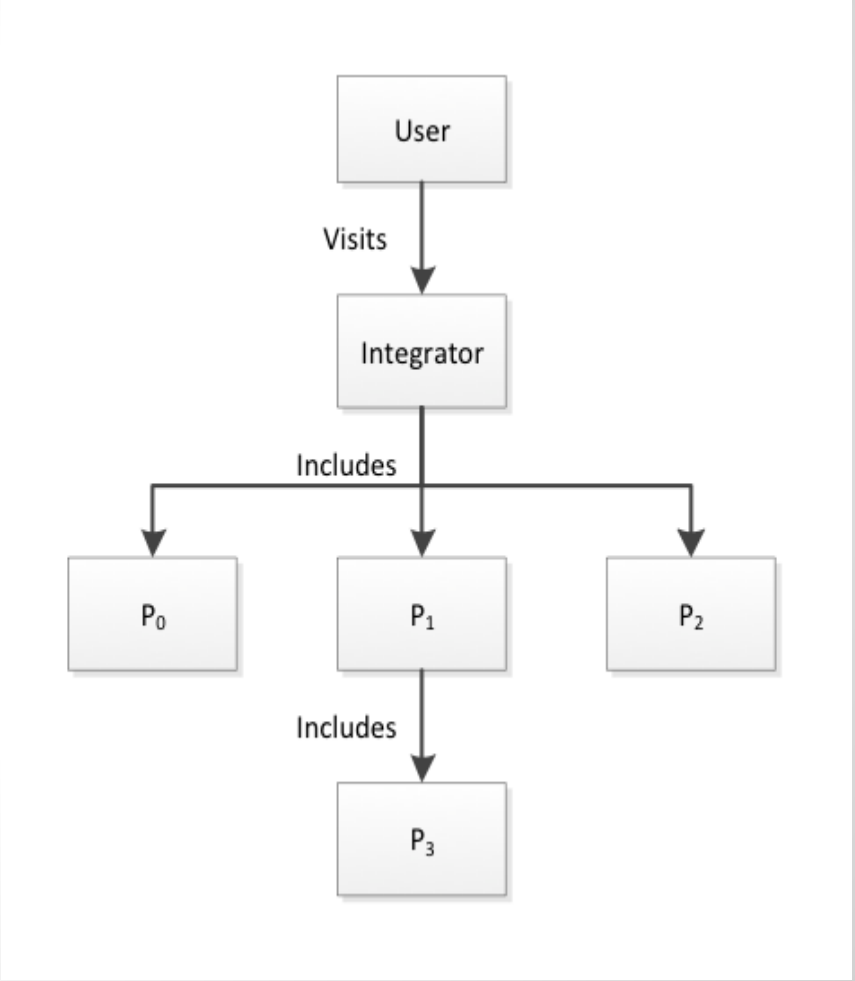
- Policies tend to be complex
- Varying performance and compatibility implications

Goals

- Isolation by default
- Simplicity
- Efficient, synchronous sharing
- Fail-safe legacy code

Design: terminology

- *Principal* is an instance of content
- May include HTML, CSS, JS
- Top-level is *integrator*
- Each principal is placed in a *box*, the unit of isolation



What's in a box?

- JavaScript namespace
- DOM tree
- Event loop
- Visual region
- Network connection
- Local storage area

box != iframe

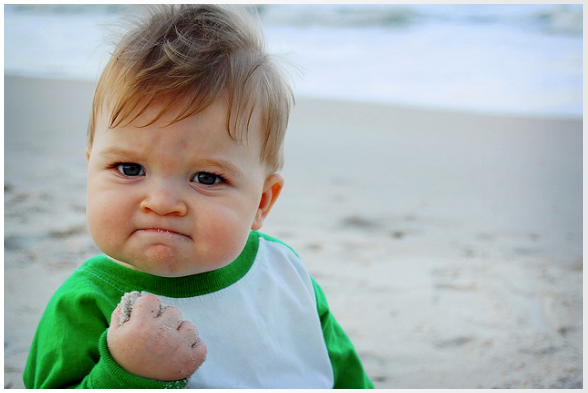
- Same-origin boxes isolated by default
  - Enables fault isolation, privilege separation
- Box permissions nest
  - e.g., monotonically decreasing network permission
- Synchronous communication

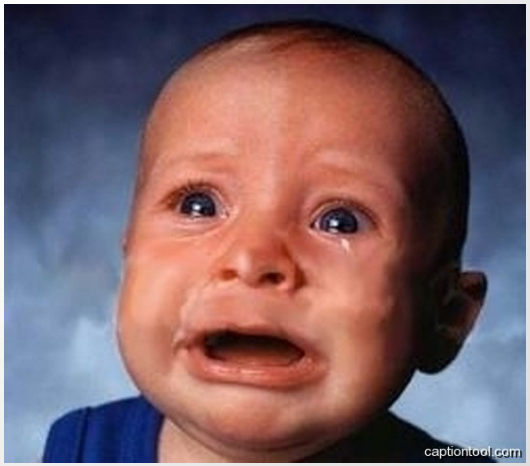
The horror of asynchrony

- N items to process with external library

```
for i in 0 .. N-1
  processed[i] =
    externalLibrary.process(data[i])
```

```
function process(data, i) {
  if (i < N) {
    externalLibrary.process(data[i],
      function(result) {
        processed[i] = result
        process(data, i+1)
      }
    )
  }
}
process(data, 0)
```





Design: principal objects

- Each principal has a *principal object*
  - Defines the public interface
- Jigsaw.getParentPrincipal()
- Jigsaw.principals

Design: DOM tree and visual field

- Each box can have a DOM tree and associated visual field.
- Visual field: width, height, location (within parent), z-order
- Granted using CSS-style syntax
- Parent can change child visual field
- Child changes validated by parent

Design: network access

- Granted from parent to child
- Specified as a whitelist of accessible domains
- Wildcards allowed
  - e.g., `*.foo.com` or `cache.*.bar.com`
- Monotonically decreasing

Design: JavaScript namespace

- public/private visibility modifiers
- Define the subset of an object graph that crosses an isolation boundary
- private by default

Principal X passes

```
{public p: "foo",
 private q: "bar"}
```

to principal Y

Principal Y sees

```
{p: "foo"}
```

Design: Surrogate objects

- *Surrogate* objects enforce private/public
- Jigsaw passes surrogate, not raw object, between boxes
  - Initially empty object, with public properties added

- Getter for public `p` of `obj` returns `createSurrogate(obj.p)`
- Setter for public `p` of `obj` executes `obj.p = createSurrogate(newVal)`

Design summary

- Isolation by default using boxes
- Principal object defines interface
- Only public properties traverse box boundary
- Resources (e.g., network, visual field) granted by parent to child

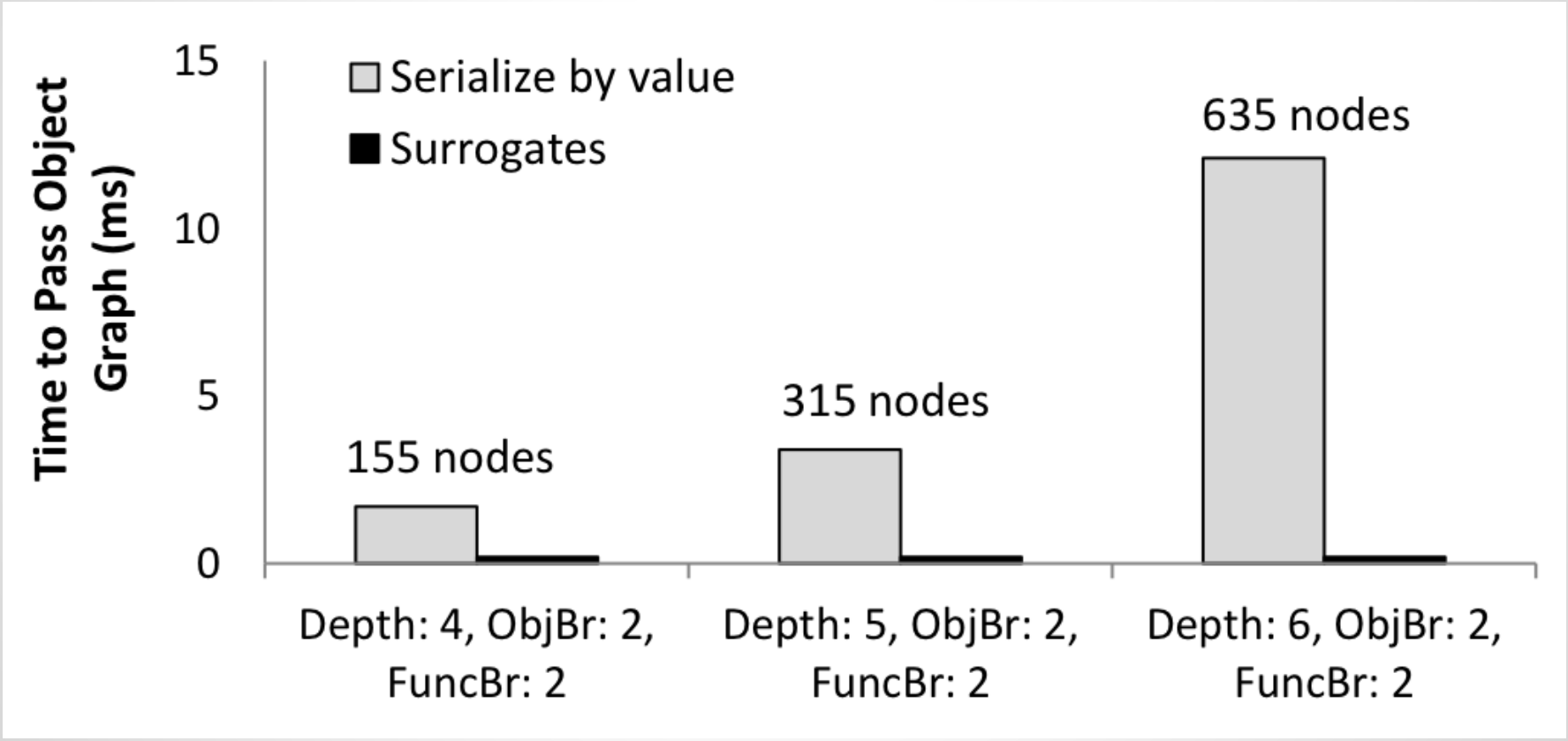
Implementation

- Jigsaw-to-JavaScript compiler
  - Translate private/public keywords into operations on per-object visibility metadata map
  - Adds calls to create surrogates
  - Maintain object ids and box ids
- Client-side JavaScript library
  - Defines management interface (e.g., `Jigsaw.createBox()`)
  - `eval`s box code in context with redefined globals
  - Redefined globals implement security checks
- Current prototype implements most (but not all) of the design

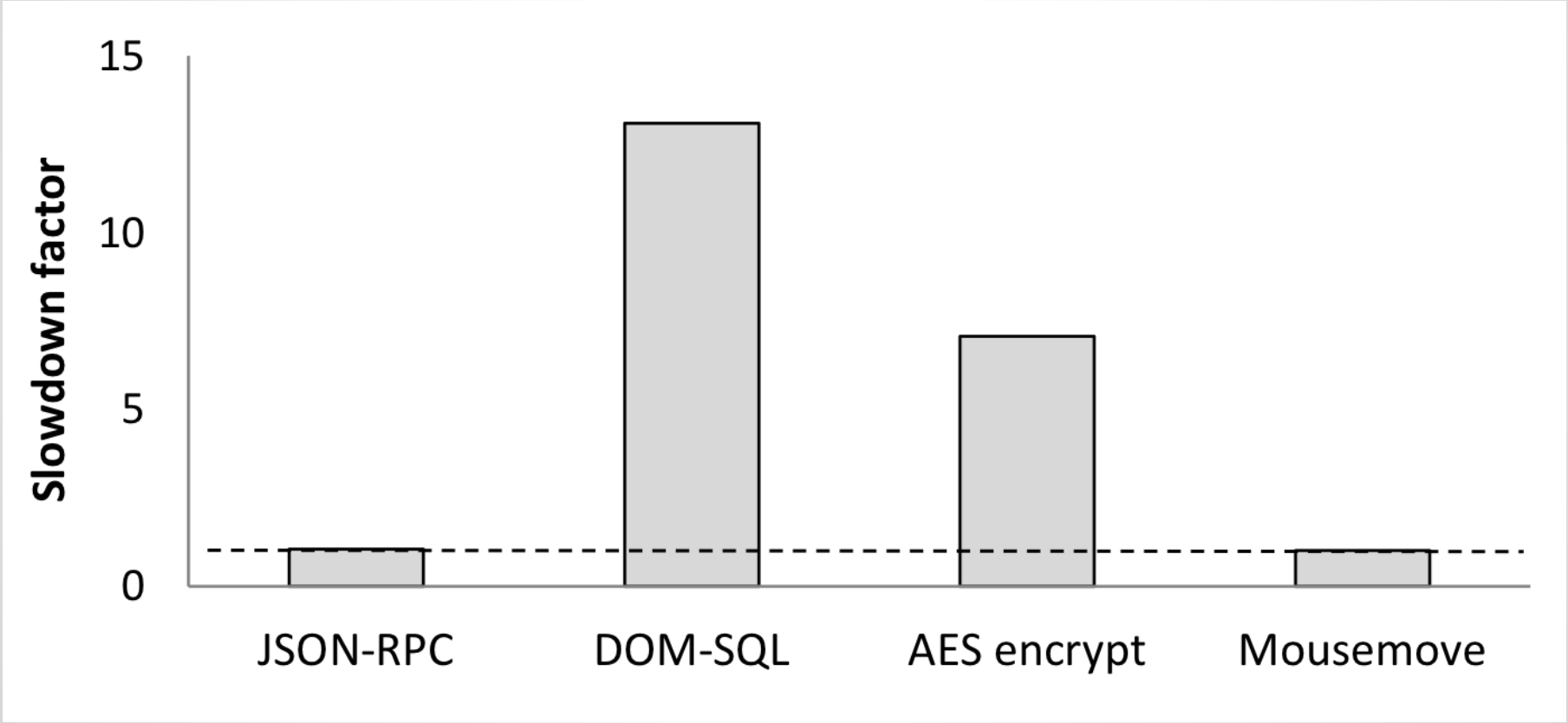
Evaluation: porting effort

- Many libraries already have a de facto principal object -- mark it as such
- Mark properties as public where appropriate
- Use a modified runtime to log private objects crossing boundaries, instead of disallowing them
- No explicit sanitization necessary

Evaluation: performance



Evaluation: performance



Related work

- ADsafe
- FBJS
- Dojo Secure
- Caja
- Secure ECMAScript
- PostMash
- Object Views
- ConScript

Conclusion

- Jigsaw: a new mashup isolation framework
- Policies are simple to write
  - public/private objects
  - high-level browser resources
- Synchronous programming model
- Automatic surrogates

Thank you

Matthew Finifter
finifter@cs.berkeley.edu

James Mickens
mickens@microsoft.com