Crom: Faster Web Browsing Using Speculative Execution

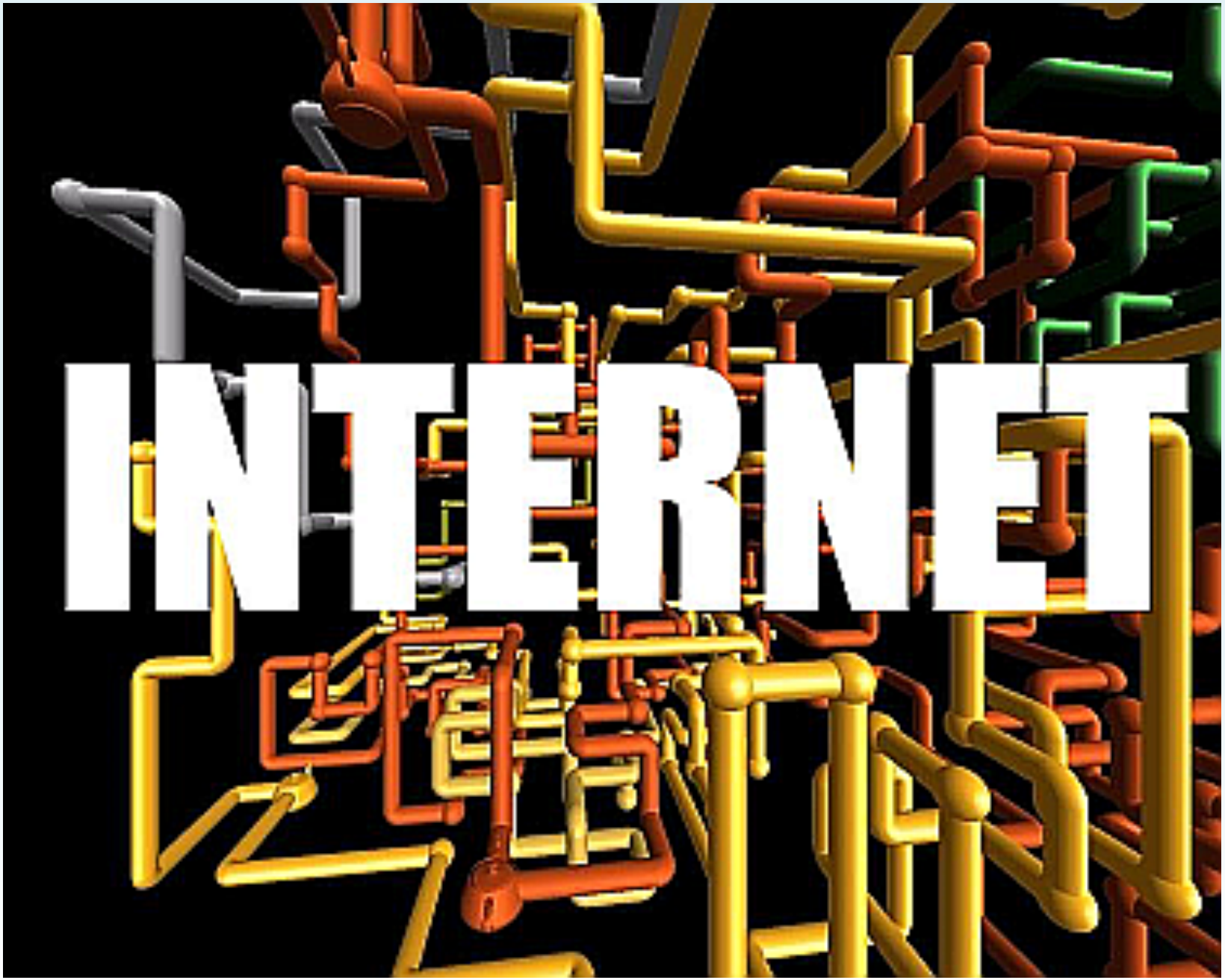

James Mickens


Jeremy Elson


Jon Howell


Jacob R. Lorch

Microsoft® Research

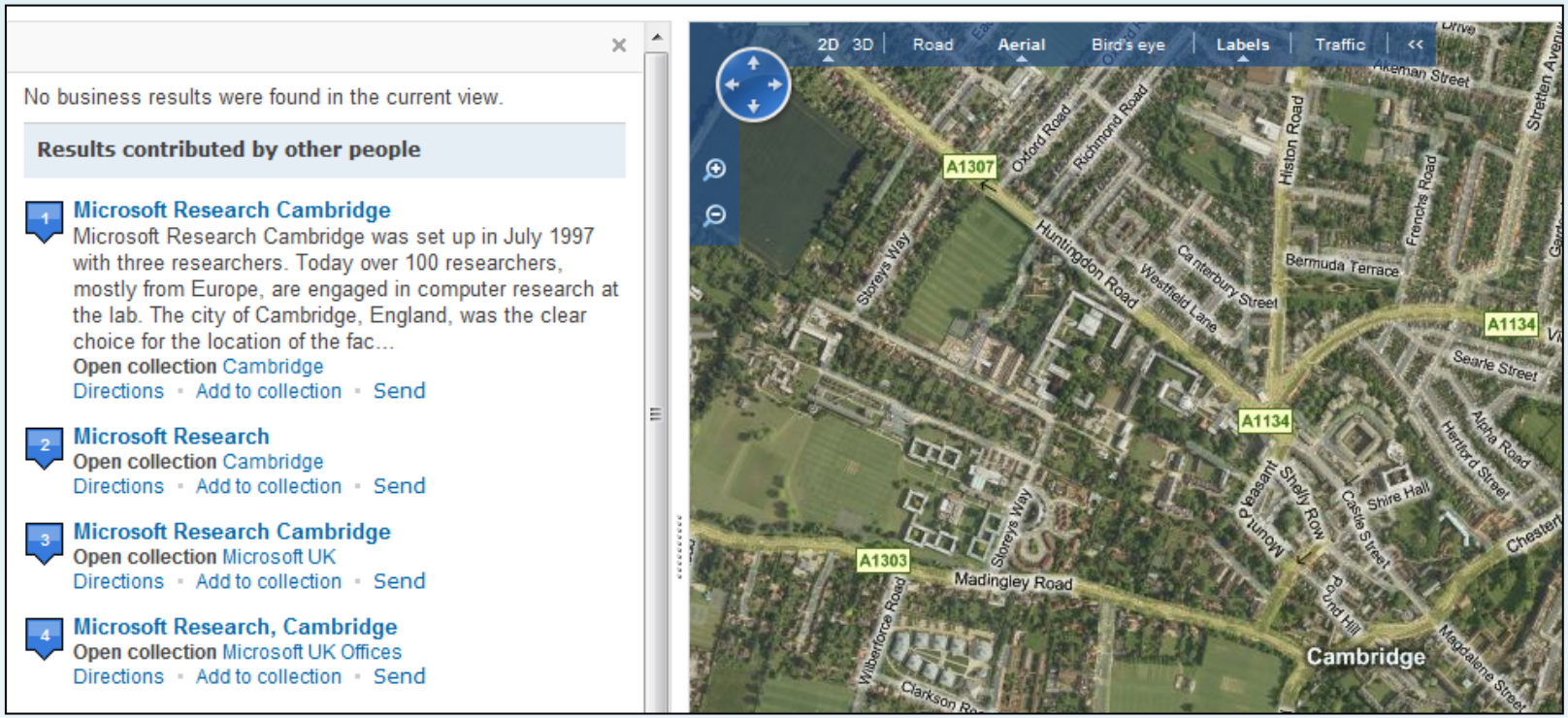

INTERNET

Prefetching: Web 1.0

- Static objects connected by declarative links
- Find prefetchable objects by traversing graph


```
<TABLE WIDTH='100%‘>
  <TR>
    <TD VALIGN='TOP' HEIGHT=60>
      <A HREF='/misc/winNews.htm'>
        Windows news
      </A>
    </TD>
```

The Brave New World: Web 2.0

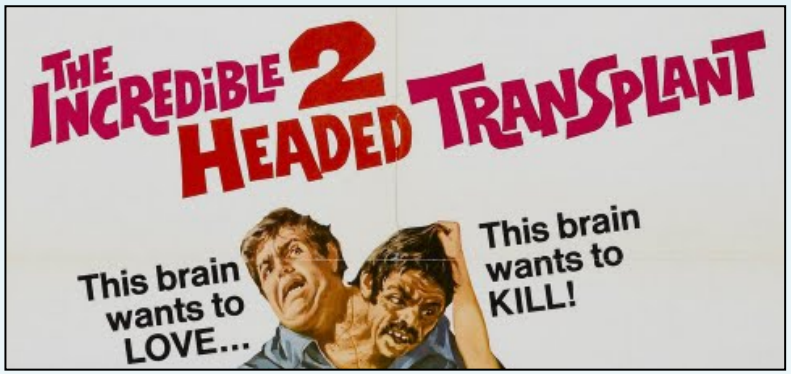



```
imgTile.ondblclick = function(){
        map.centerTile = imgTile;
        map.zoomLevel++;
        map.fetchAndDisplayNeighbors(imgTile, map.zoomLevel);
}
```

New Challenges to Prefetching

- Fetches triggered by imperative event handlers
  - Can't just "pre-execute" handlers to warm cache

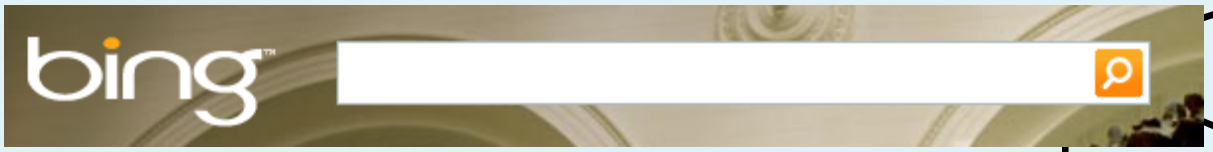
```
imgTile.ondblclick = function(){
        map.centerTile = imgTile;
        map.zoomLevel++;
        map.fetchAndDisplayNeighbors(imgTile,
                                     map.zoomLevel);
}
```

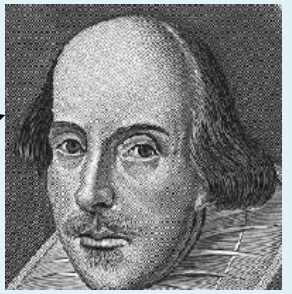


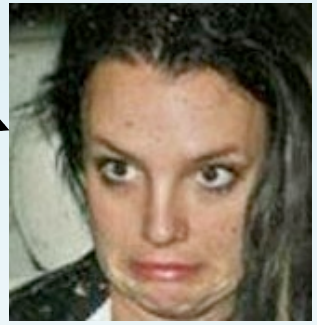
- Prefetcher must understand JavaScript
  - Hide side effects of speculation until commit time

New Challenges to Prefetching

- Fetches often driven by user-generated inputs
  - Binary: clicking a “download” button
  - Unconstrained: typing into a search form







Deleted Photo

- Infeasible to speculate on all possible user inputs!
  - Only speculate on *constrained* set of *reasonable* inputs

Prior Solutions: Custom Code

- Advantage: Exploit app-specific knowledge for . . .
  - Tight code
  - High performance

- Disadvantages:
  - Often difficult to write
  - Tightly integrated with application code base (can't be shared with other apps)

Our Solution: Crom

- A *generic* speculation engine for JavaScript
  - Implemented as regular JavaScript library
  - Requires no modification to browsers
- Applications define their *speculative intents*
  - Which event handlers should be speculative?
  - At what point should speculations occur?
  - Given an application state, how does Crom generate speculative user inputs that are reasonable?

Crom Handles The Rest ™

- Clones browser state
- Executes rewritten event handlers
- Commits shadow state if appropriate (fetch latency masked!)

- Crom provides other goodness:
  - AJAX cache
  - Speculative upload API
  - Speculative r+w ops on server-side

Outline

- Speculative Execution
  - Cloning the browser state
  - Rewriting event handlers
  - Committing speculative contexts
  - Optimizations
- Evaluation
- Related Work
- Conclusions

Adding Speculative Execution

```
<div id="textDiv">
    Click here to increment counter!
</div>
<script>
    var globalVar = 42;
    var f1 = function(){globalVar++;};
    var f2 = function(){globalVar++; f1();};
    var t = document.getElementById("textDiv");
    t.onclick = f2;
    Crom.makeSpeculative(t, "onclick");
    Crom.speculate();
</script>
```

Cloning the Browser State

- Application heap
  - All JavaScript objects reachable from the roots of the global namespace
  - Apps access global namespace through global "window" variable (e.g., window.X)
- DOM tree
  - JavaScript data structure representing page HTML
  - Each HTML tag has corresponding DOM object
  - App changes page visuals by modifying DOM tree

Cloning the Application Heap

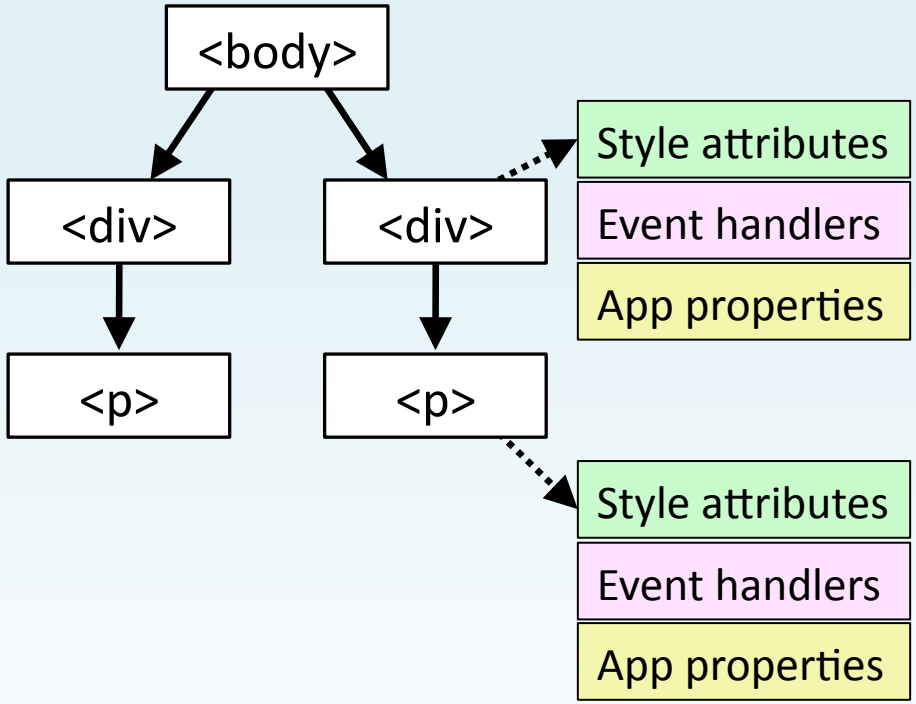
- Walk object graph and deep-copy everything

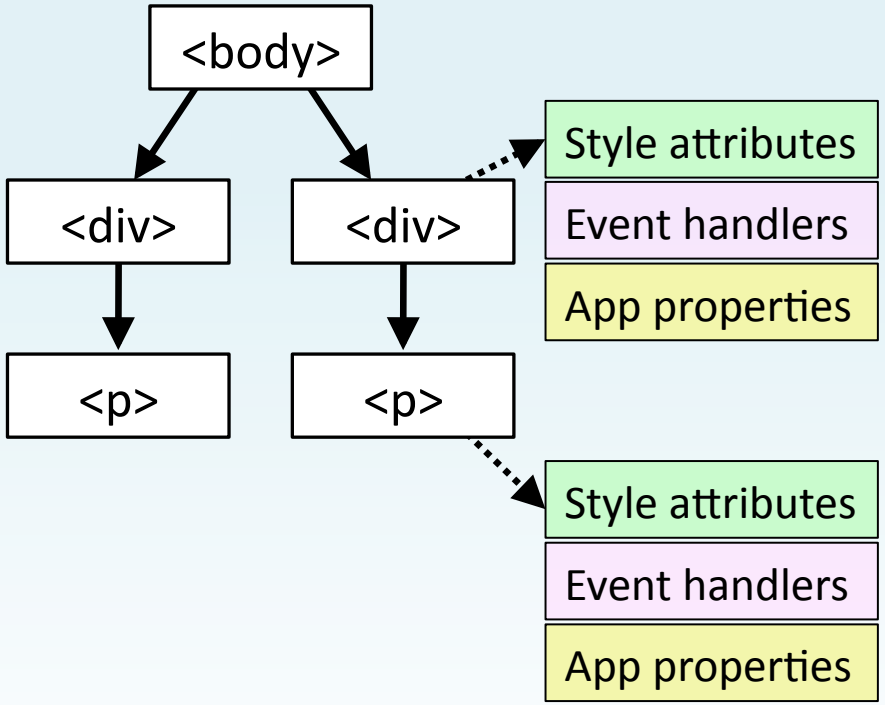
```
var specWindow = {};  //Our shadow global namespace
for(var globalProp in window)
    specWindow[globalProp] = Crom.deepCopy(window[globalProp]);
```

- Objects, primitives copied in obvious way . . .
- Functions
  - clonedF = eval(f.toString())
  - Deep copy any object properties

Cloning the DOM Tree

1) body.cloneNode()
   (Native code: FAST)
2) Crom fix-up traversal
   (Non-native code: SLOW)





Putting It All Together

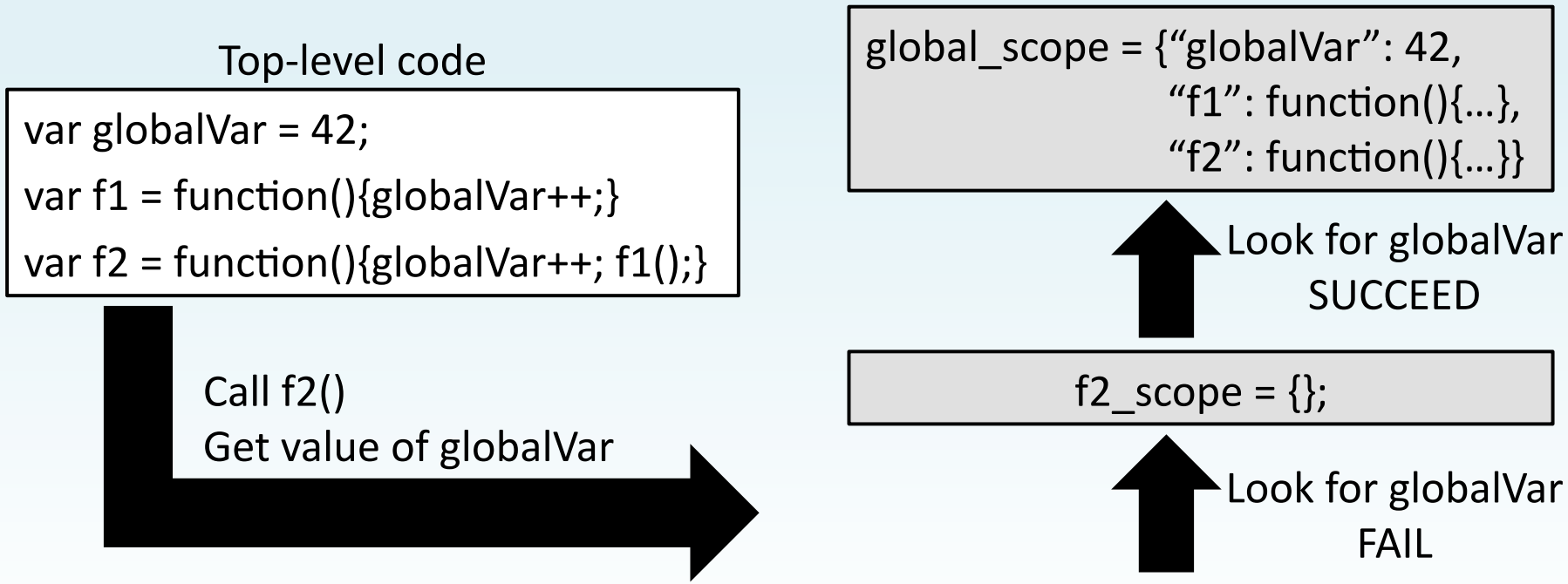
```
//Create a new DOM tree
var specFrame = document.createElement("IFRAME");
specFrame.document.body = Crom.cloneDOM();
```

```
//Create a new global JavaScript namespace
var specWindow = {};
for(var globalProp in window)
    specWindow[globalProp] = Crom.deepCopy(window[globalProp]);
specWindow.window = specWindow;
specWindow.document = specFrame.document;
```

Congratulations!

Rewriting Event Handlers

- JavaScript is lexically scoped
  - Activation records are objects (varName→varValue)
  - Resolve refs by following chain of scope objects

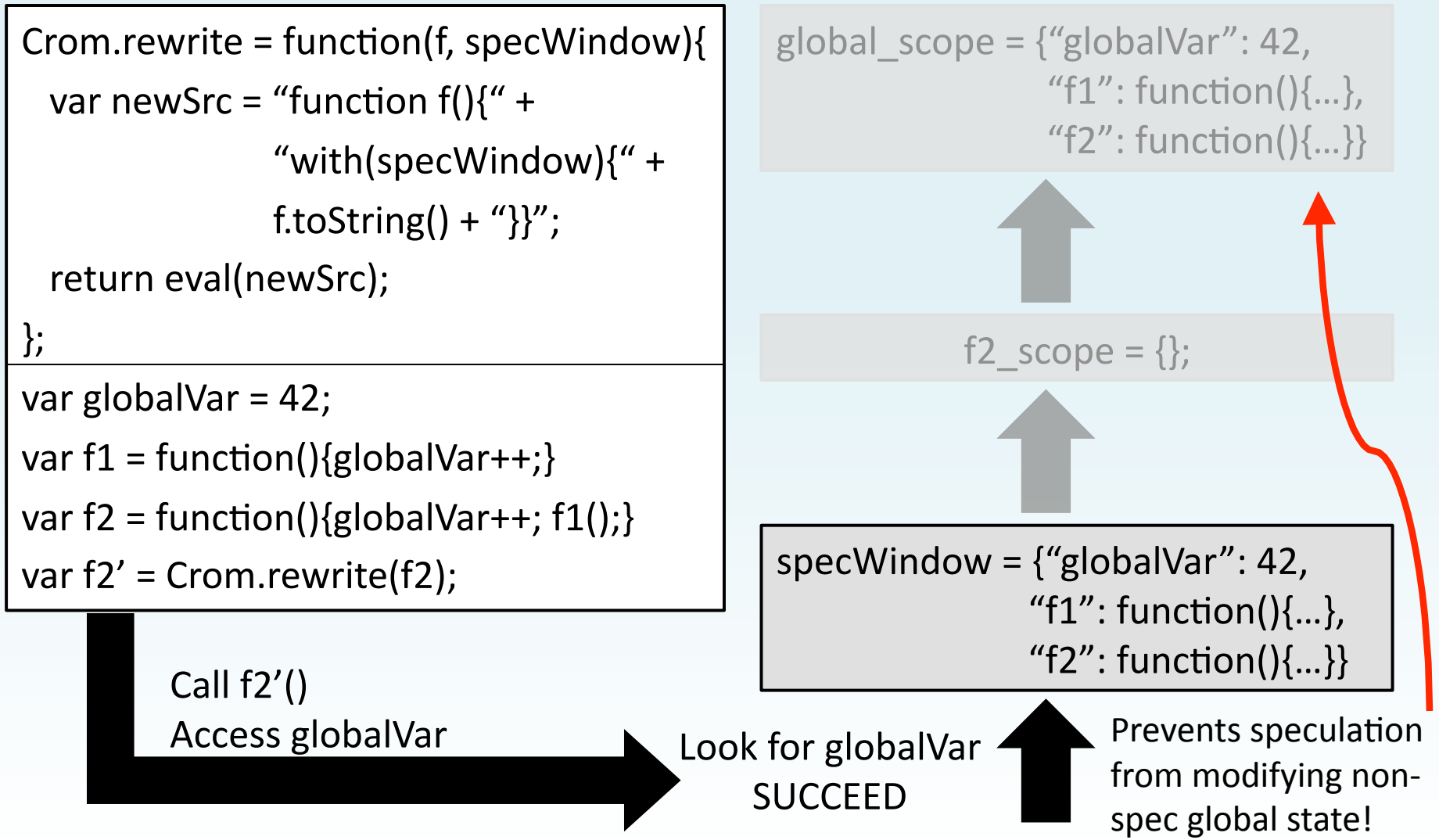


Rewriting Event Handlers

- "with(obj)" inserts obj to front of scope chain

```
Crom.rewrite = function(f, specWindow){
  var newSrc = "function f(){" +
               "with(specWindow){" +
               f.toString() + "}}";
  return eval(newSrc);
};
```

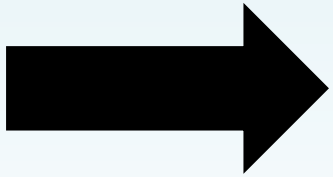
```
var globalVar = 42;
var f1 = function(){globalVar++;}
var f2 = function(){globalVar++; f1();}
var f2' = Crom.rewrite(f2);
```



Rewriting Event Handlers

- Various details (see the paper)
  - Lazily rewriting inner function calls
  - Addition/deletion of global variables
  - Rewriting closures
  - Local variables that shadow global ones

```
var f2 = function(){
   globalVar++;
   f1();
};
```



```
specWindow = Crom.newContext();
var f2' = function(){
   with(specWindow){
      var f1' = Crom.rewrite(f1, specWindow);
      globalVar++;
      f1'();
   }
};
```

Adding Speculative Execution

```
<div id="textDiv">
    Click here to increment counter!
</div>
<script>
    var globalVar = 42;
    var f1 = function(){globalVar++;}
    var f2 = function(){globalVar++; f1();}
    var t = document.getElementById("textDiv");
    t.onclick = f2;
    Crom.makeSpeculative(t, "onclick");
    Crom.speculate();
</script>
```

1) Clone browser state
2) Rewrite t.onclick()

3) Run t.onclick'()

Outline

- Speculative Execution
  - Cloning the browser state
  - Rewriting event handlers
  - Committing speculative contexts
  - Optimizations
- Evaluation
- Related Work
- Conclusions

Committing Speculative Contexts

- Suppose you know which one to commit . . .

```
//Commit the speculative DOM tree
document.body = specWindow.document.body;
```

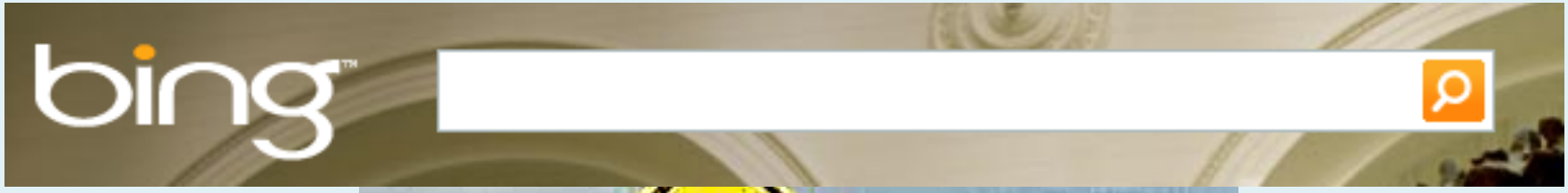
```
//Commit the application heap by committing global heap roots
for(var propName in specWindow)
        window[propName] = specWindow[propName];
```

```
//Clean-up globals deleted by committing speculation
for(var propName in window){
        if(!(propName in specWindow))
                delete window[propName];
}
```

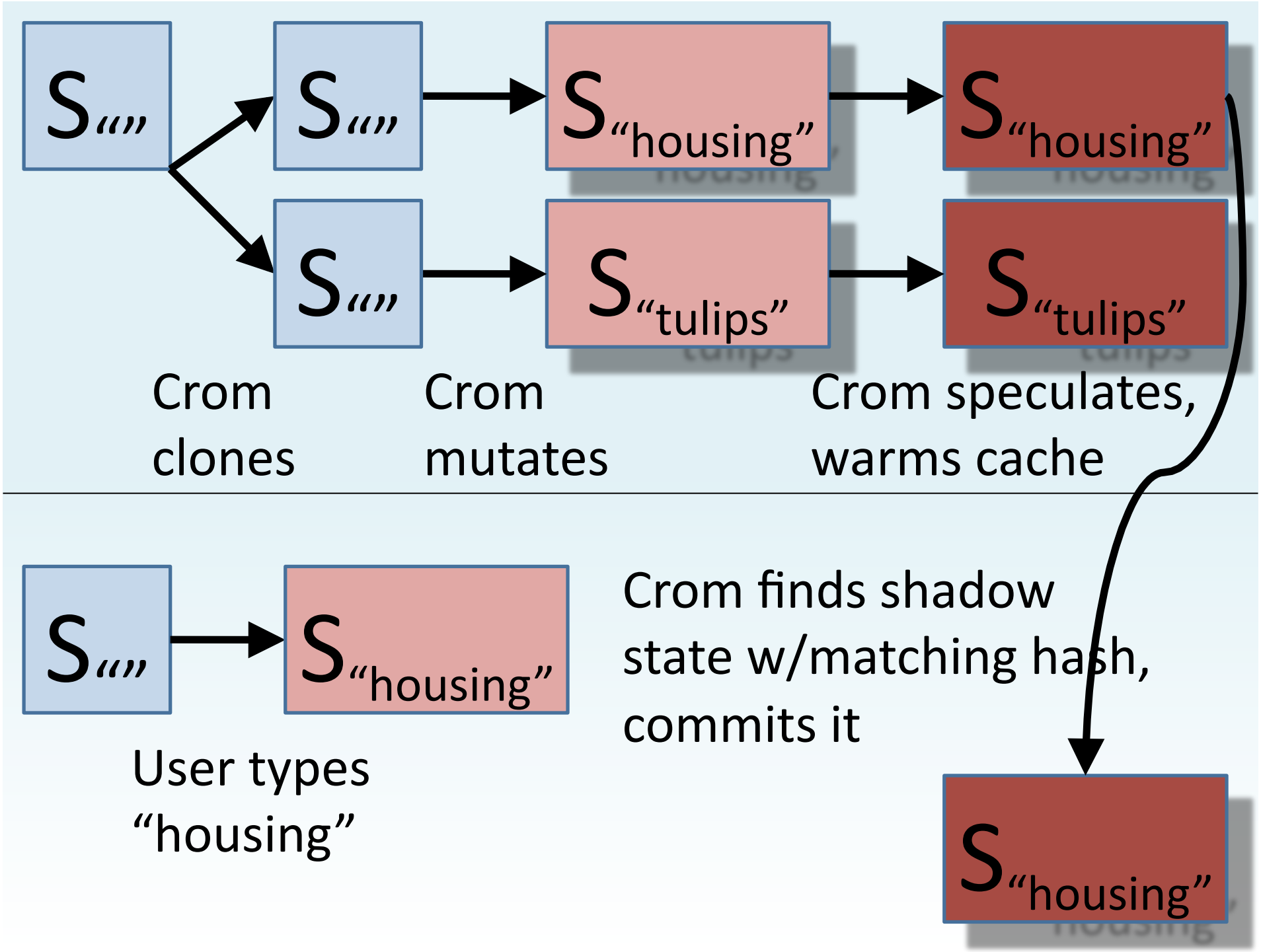
- . . . but how do you know?

Start-state Equivalence

- When is it safe to commit a speculative context?
  - Its *start state* must have been equivalent to application's *current state*
  - The speculative input that mutated it must be equivalent to the current (real) input
- Application defines equivalence function
  - Hash function over global namespace (real or speculative)
  - Speculative context can only commit if its hash matches that of current (real) namespace
- Application defines mutator function
  - Tells Crom how to change a new speculative context before running speculative event handler




```
Mutator:  function(specNamespace, specInput){
                specNamespace.searchText = specInput;
          }
State hash: function(globalNamespace){
                return globalNamespace.searchText;
          }
Crom.makeSpeculative(searchText, “onchange”,
                     mutator, stateHash,
                     [[“housing”], [“tulips”]]);
```


S""
S""
S"housing"
S"housing"
S""
S"tulips"
S"tulips"
Crom clones
Crom mutates
Crom speculates, warms cache
S""
S"housing"
User types "housing"
Crom finds shadow state w/matching hash, commits it
S"housing"

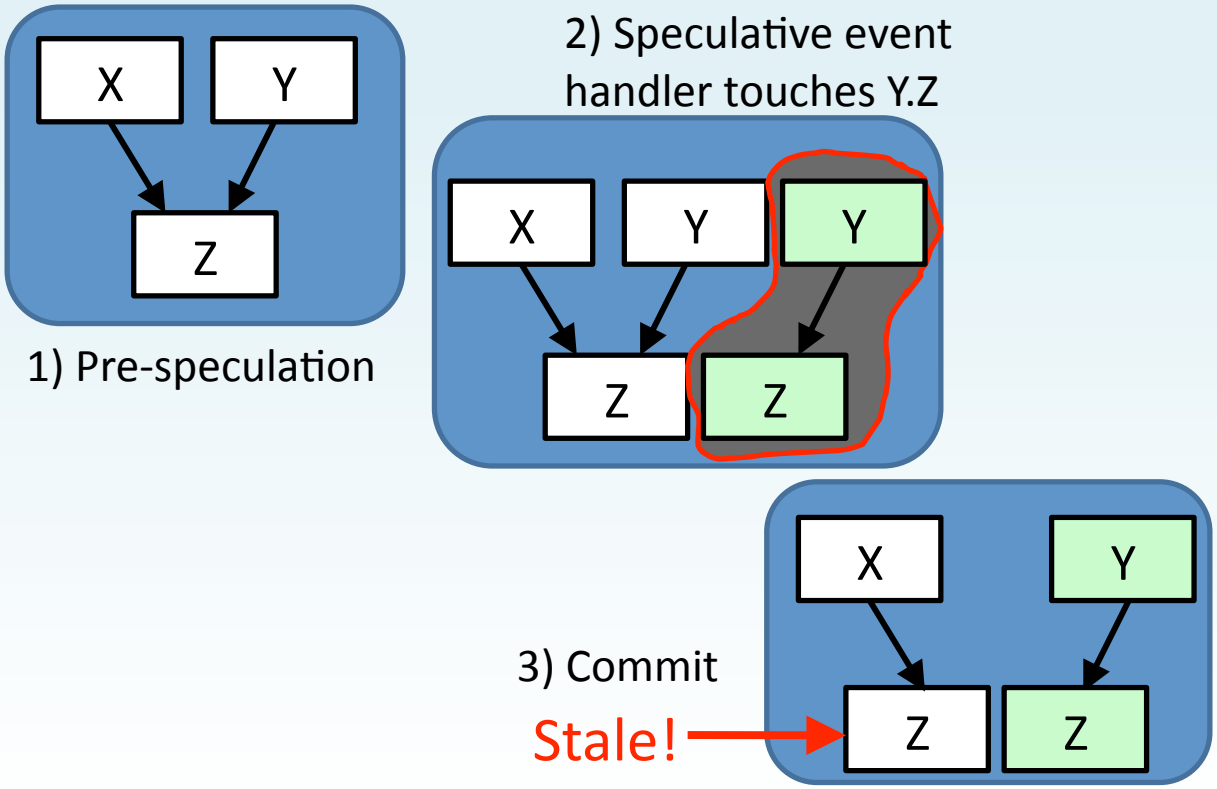
Start-state Equivalence

- What if app doesn't specify SSE data?
  - Crom throws away all speculations whenever any event handler executes, respeculates on everything
- Guarantees correctness for commits . . .
  - . . . but may lead to wasteful respeculation

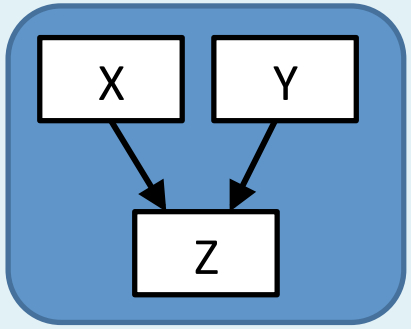
Outline

- Speculative Execution
  - Cloning the browser state
  - Rewriting event handlers
  - Committing speculative contexts
  - Optimizations
- Evaluation
- Related Work
- Conclusions

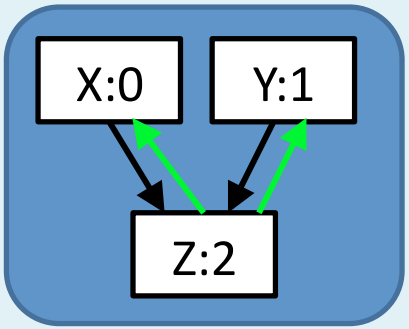
- Don't need to copy entire heap forest!
  - Only clone trees touched by speculation
  - Lazily clone them at rewrite time



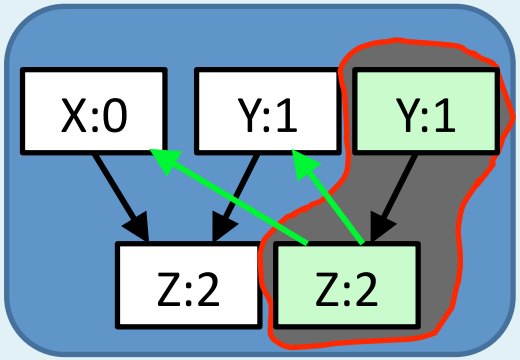
Tracking Parent References



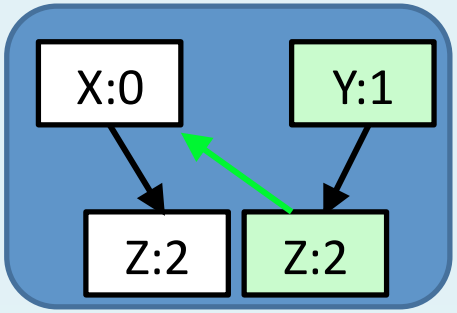
1) Pre-speculation



2) Parent mapping



3) Speculation



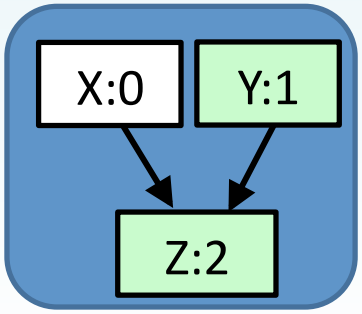
4) Commit: roots updated

Committed ids: 1,2

Ids of their parents: 0,1

X wasn't committed!

Warning: stale child ref!



5) Commit: child refs patched

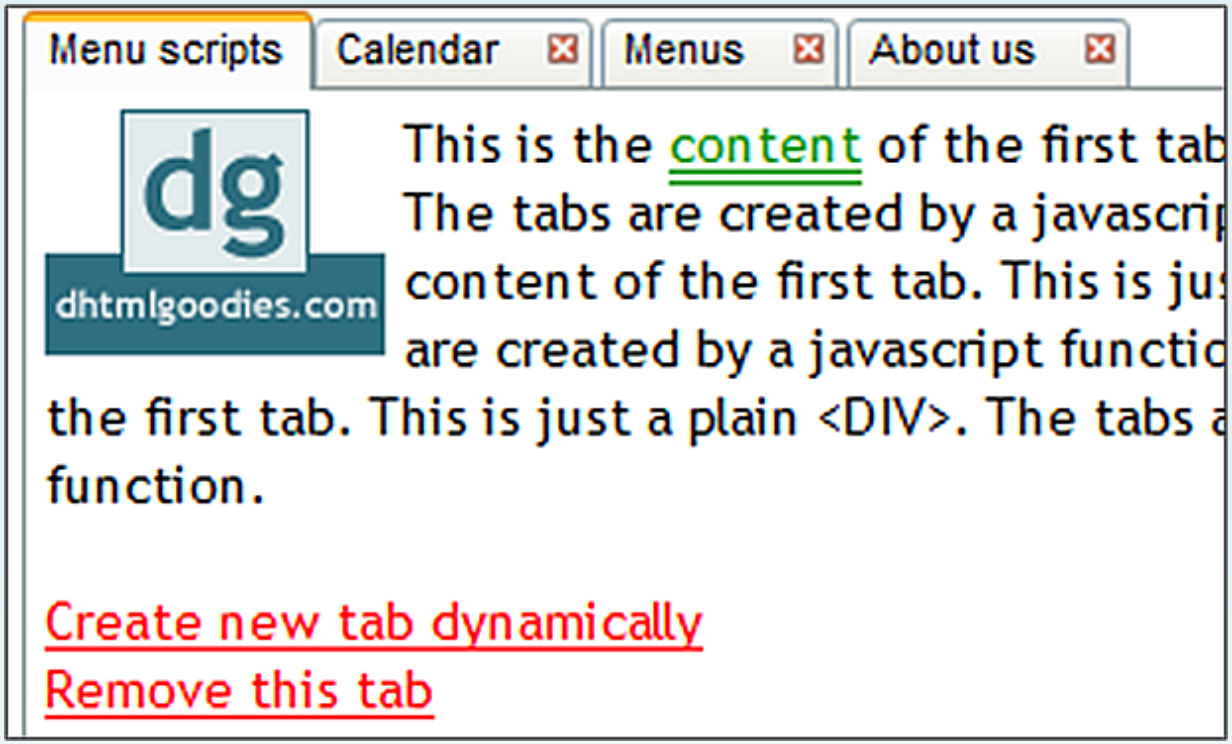
Three Speculation Modes

- Full copy: clone *entire* heap for *each* speculation
  - Always safe
  - May be slow
  - Non-amortizable costs
- Checked lazy mode: lazy copying+parent tracking
  - Always safe
  - Parent mapping costs amortizable across speculations
  - May be slow
- Unchecked lazy mode
  - Fast
  - Often safe in practice, but strictly speaking . . .
  - . . . unsafe without checked mode refactoring

Outline

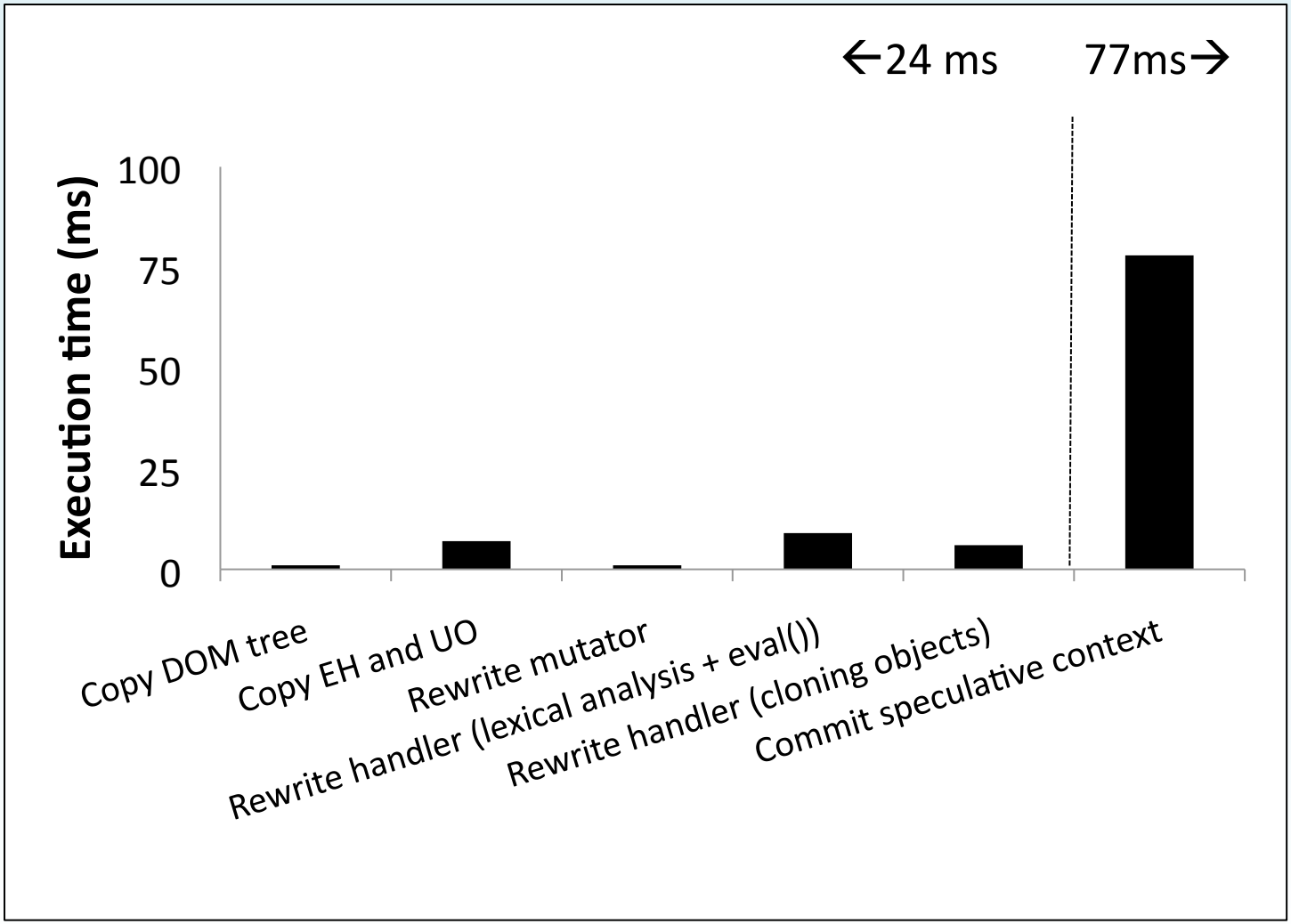
- Speculative Execution
  - Cloning the browser state
  - Rewriting event handlers
  - Committing speculative contexts
  - Optimizations
- Evaluation
- Related Work
- Conclusions

Evaluation Application

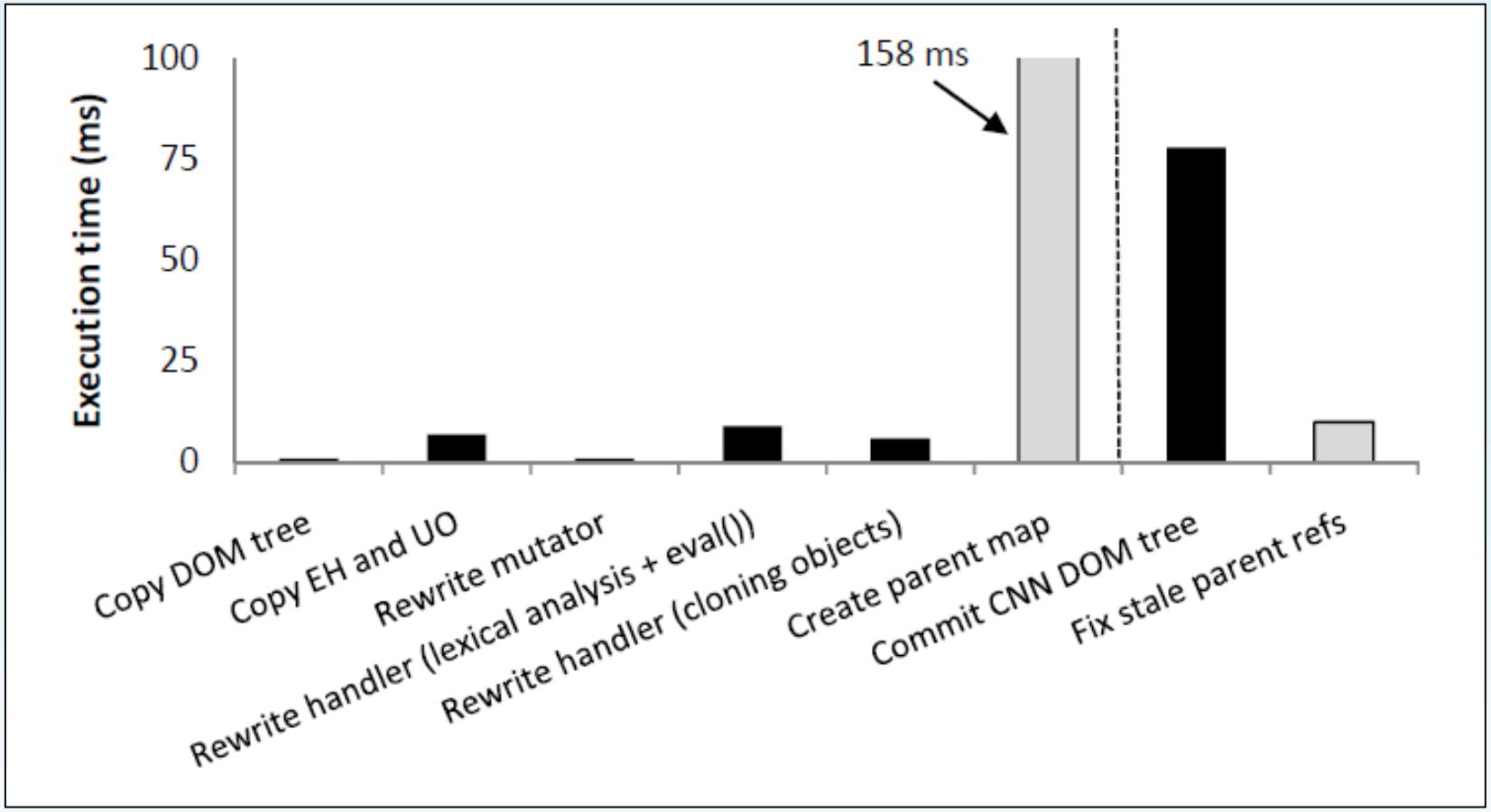


- DHTMLGoodies Tab Manager
  - Speculate on creating new tab (AJAX www.cnn.com)
  - Embed manager code within ESPN front page
- Can we hide Crom's overhead in user think time?

Speculation Costs (Unchecked Lazy Mode)



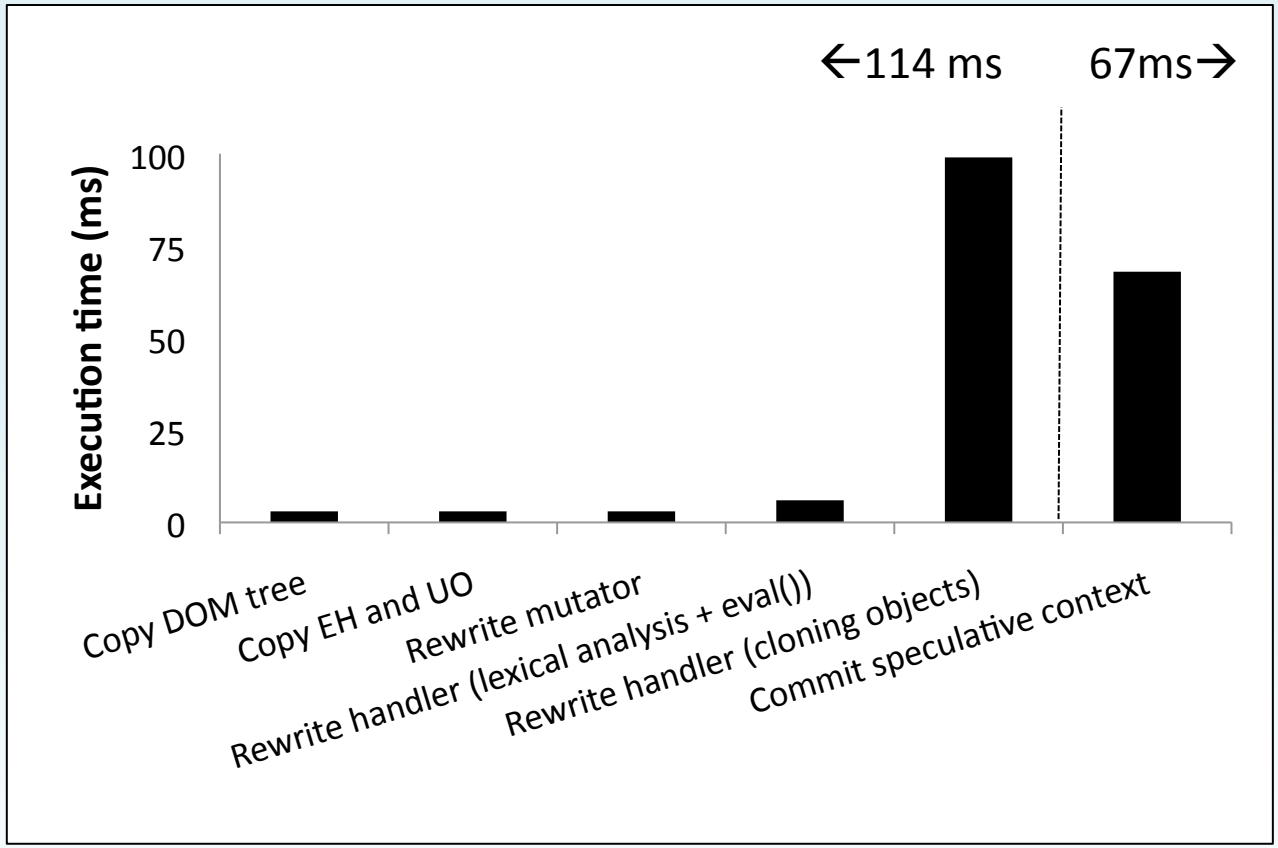
Speculation Costs (Checked Lazy Mode)



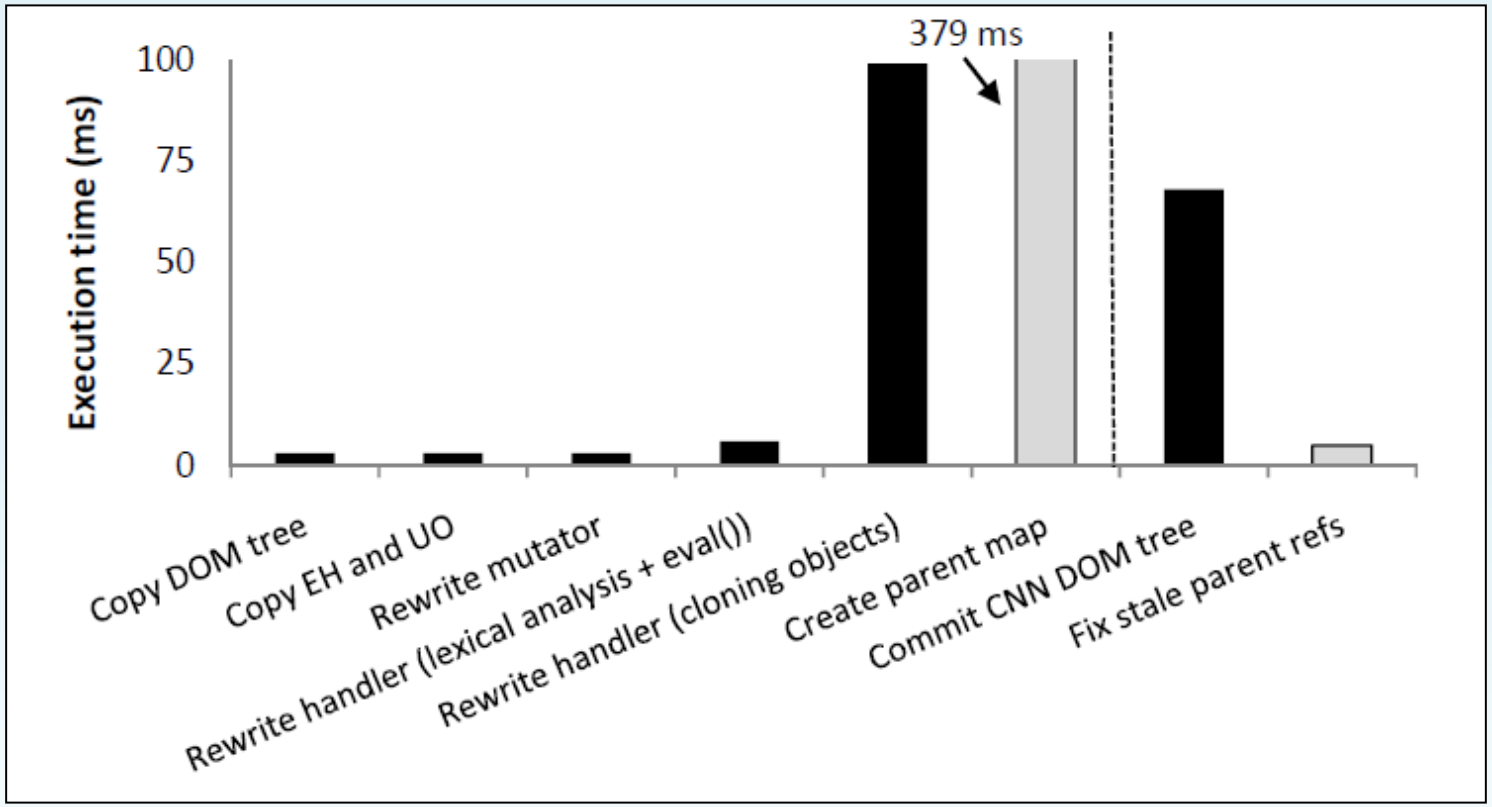
Pre-commit overhead: 182 ms

Commit overhead: 5 ms

Speculation Costs: Autocompletor



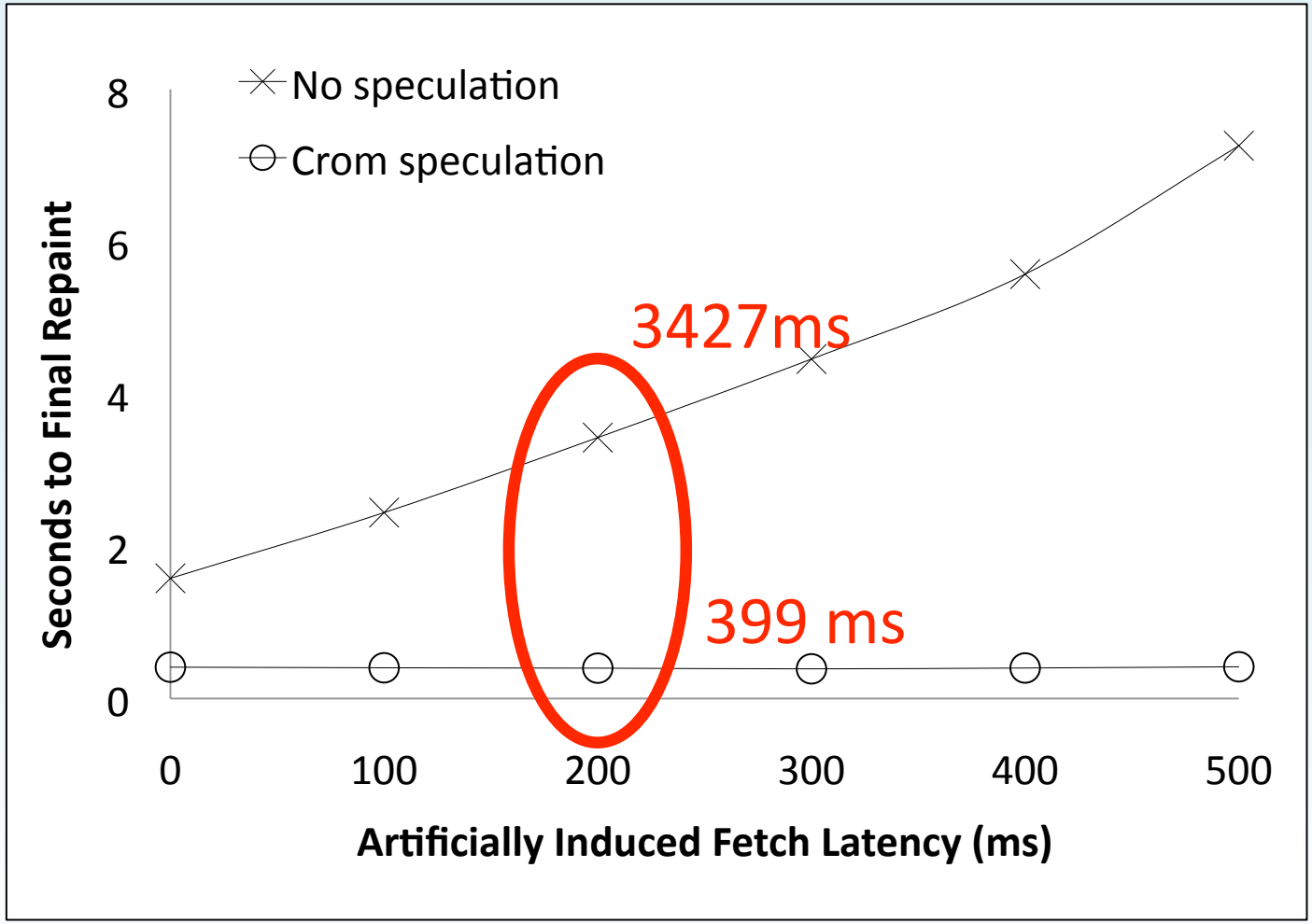
Speculation Costs: Autocompletor



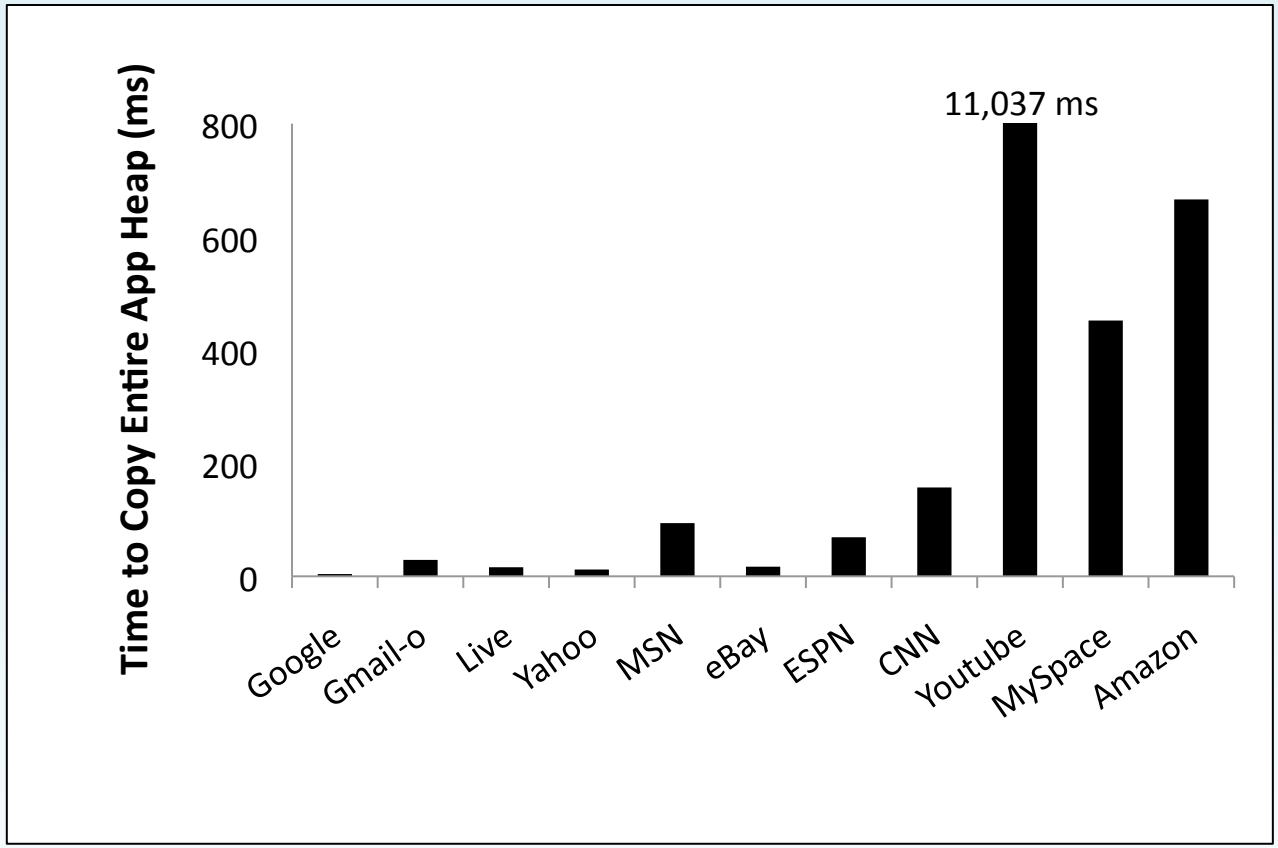
Pre-speculation overhead: 493 ms

Commit overhead: 7 ms

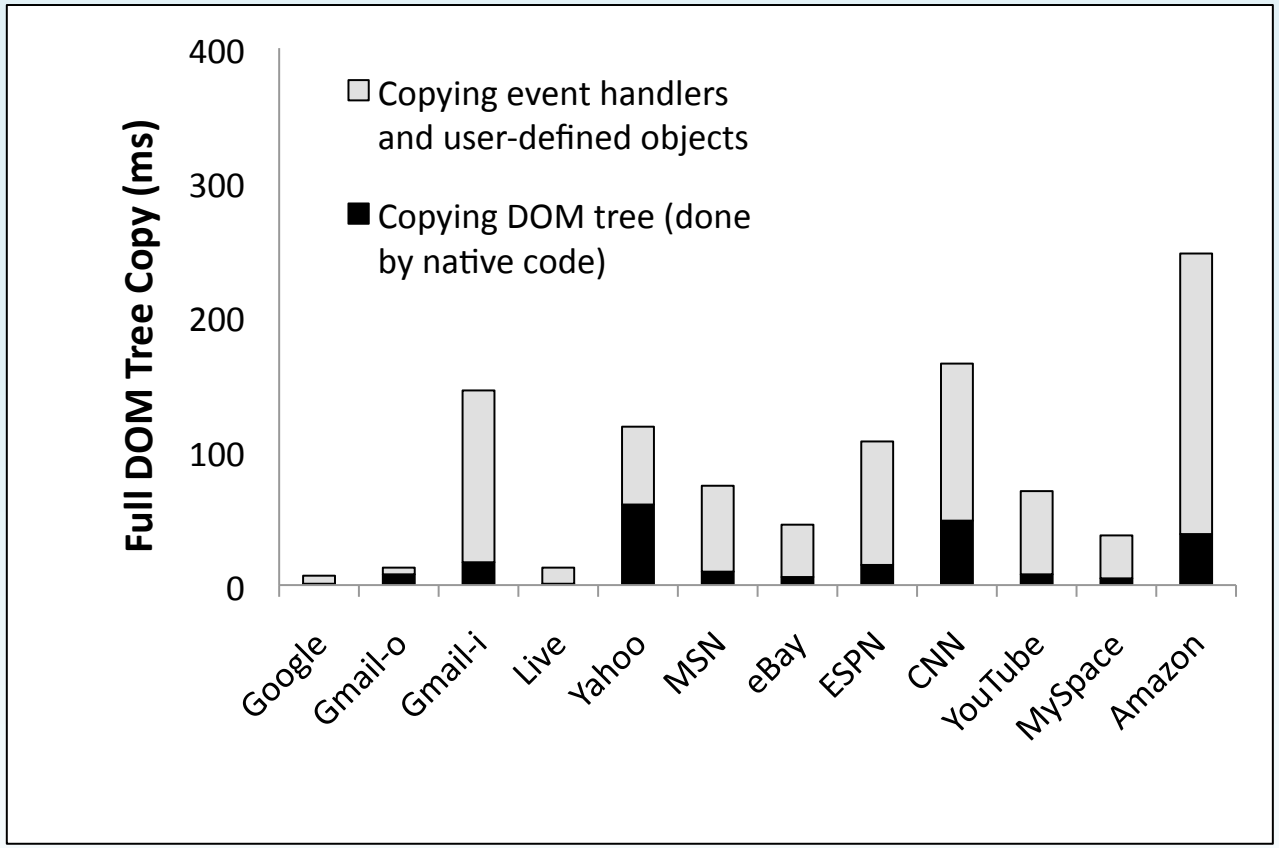
User-perceived Latency Reduction



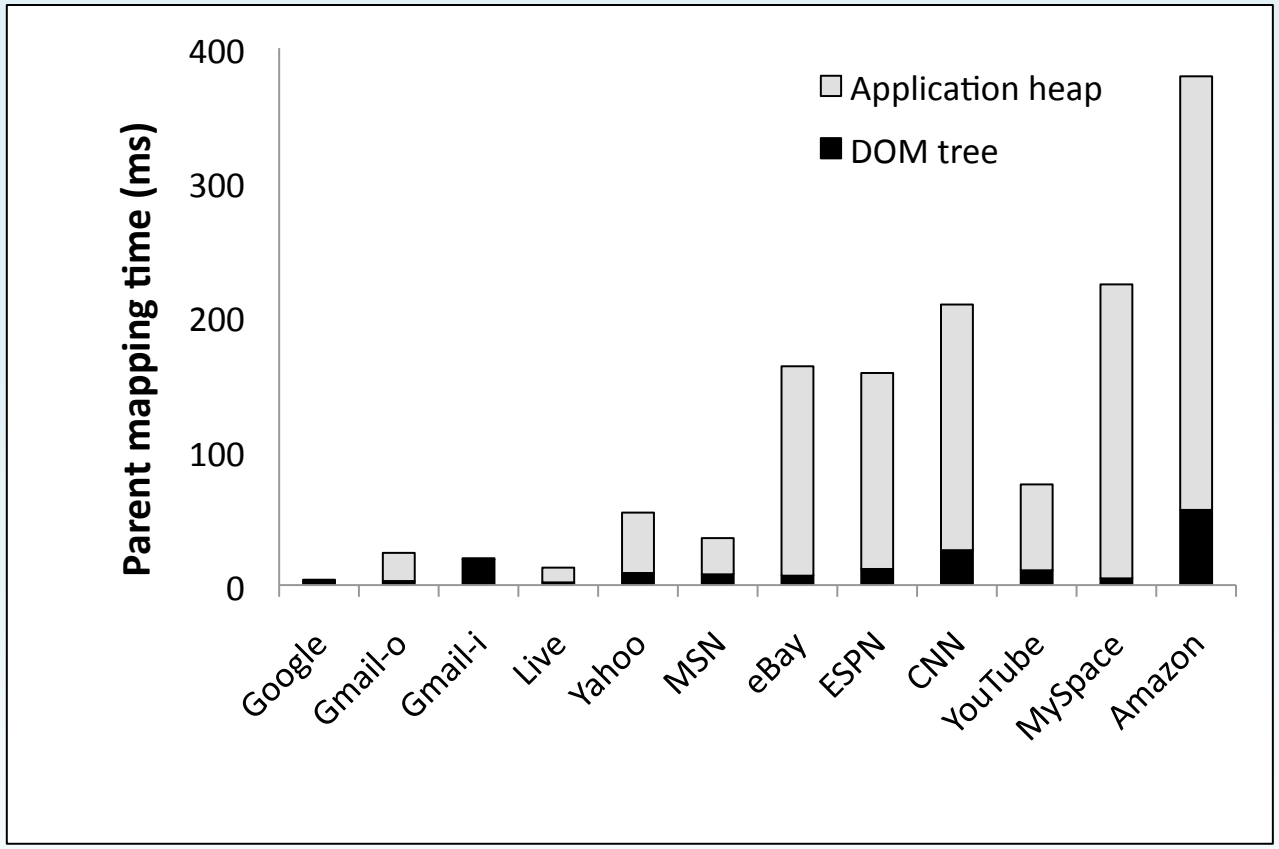
Copying the Full Heap



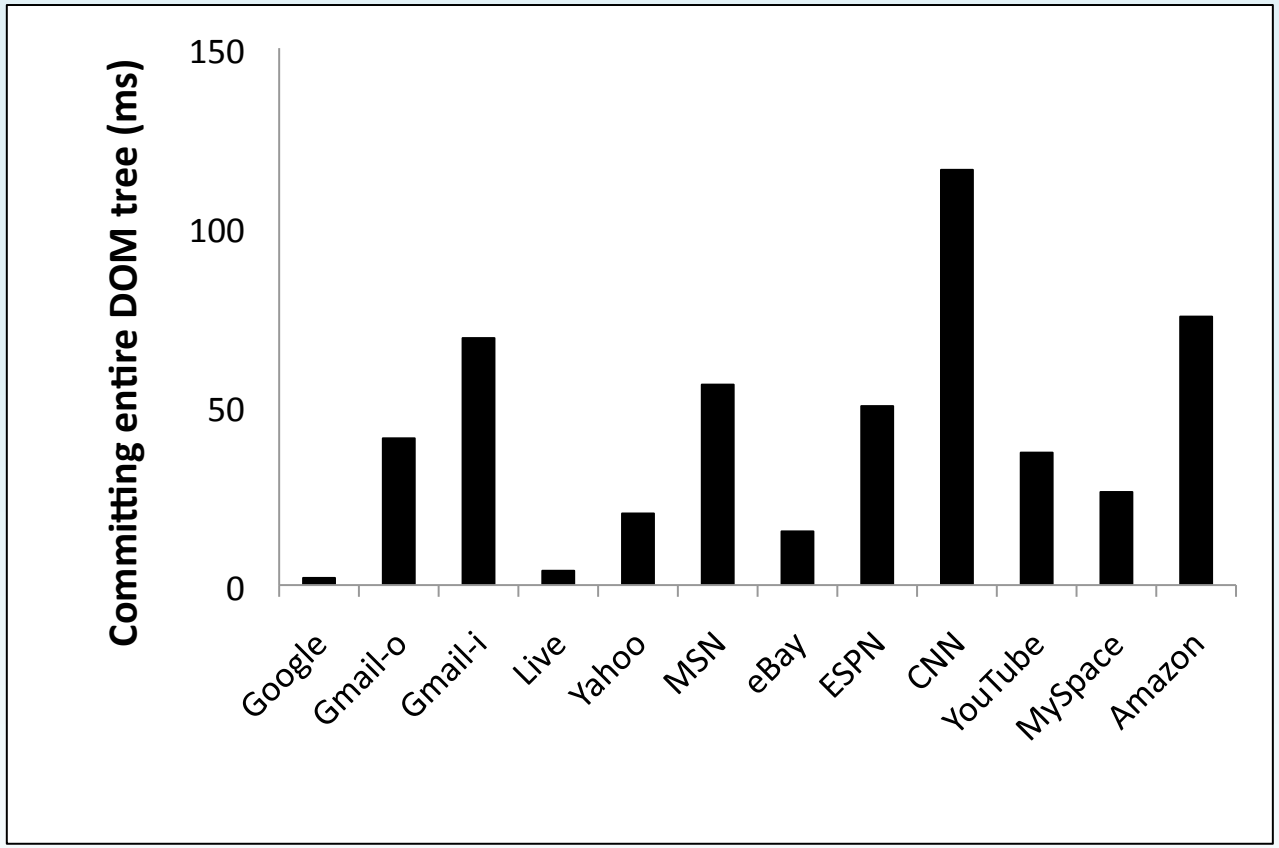
Copying the Full DOM Tree



Building the Parent Map



Committing the Entire DOM Tree



Outline

- Speculative Execution
  - Cloning the browser state
  - Rewriting event handlers
  - Committing speculative contexts
  - Optimizations
- Evaluation
- Related Work
- Conclusions

The Shoulders of Giants

- Speculation is a well-known optimization
  - File systems: Chang et al, Speculator
  - Static web data: Fasterfox, HTML 5 prefetch attribute
- Crom's contributions
  - Exploit *language introspection* to have apps self-modify
  - Explicitly reason about *user inputs*
  - Handle *dynamically-named* content

Conclusions

- Prefetching non-trivial in RIA
  - Must reason about JavaScript to get fetch targets!
  - Current speculative solutions use custom code

- Crom: generic JS speculation engine
  - Applications express speculative intents
  - Crom automates low-level tasks
  - Can reduce user-perceived latency by order of magnitude