RACE DETECTION FOR EVENT DRIVEN APPLICATIONS

Veselin Raychev
ETH Zurich

Martin Vechev
ETH Zurich

Manu Sridharan
IBM T. J. Watson

Event-Driven: Motivation

~ 1 trillion websites today

~ 1 billion smartphones by 2016

Reacts to events: user clicks, arrival of network requests

Event-Driven: Motivation

~ 1 trillion websites today

Wanted: fast response time

~ 1 billion smartphones by 2016

Reacts to events: user clicks, arrival of network requests

Event-Driven: Motivation

~ 1 trillion websites today

Wanted: fast response time

~ 1 billion smartphones by 2016

Reacts to events: user clicks, arrival of sensor data, ...

Highly Asynchronous, Complex control flow

Non-determinism: network latency

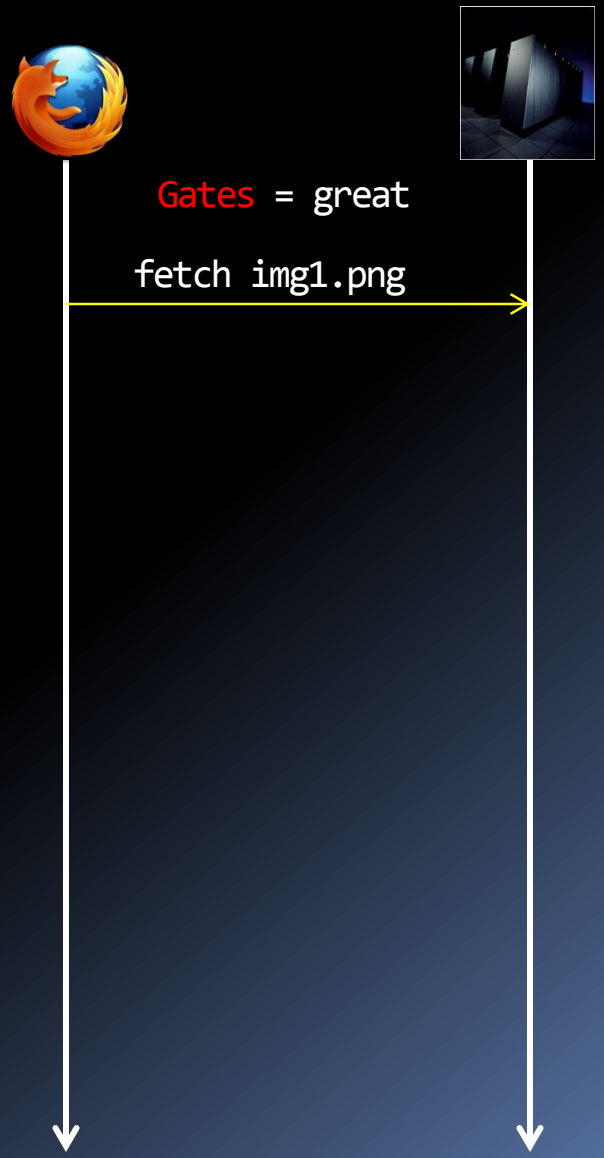
```
<html>
<head></head>

<body>
<script>
var Gates = “great”;</script>
</script>

<img src=“img1.png” onload=“Gates=‘poor’;”>
<img src=“img2.png” onload=“alert(Gates);”>

</body>
</html>
```

Non-determinism: network latency

```
<html>
<head></head>

<body>
<script>
var Gates = “great”;</script>
</script>

<img src=“img1.png” onload=“Gates=‘poor’;”>
<img src=“img2.png” onload=“alert(Gates);”>

</body>
</html>
```

Non-determinism: network latency

```
<html>
<head></head>

<body>
<script>
var Gates = “great”;</script>
</script>

<img src=“img1.png” onload=“Gates=‘poor’;”>
<img src=“img2.png” onload=“alert(Gates);”>

</body>
</html>
```

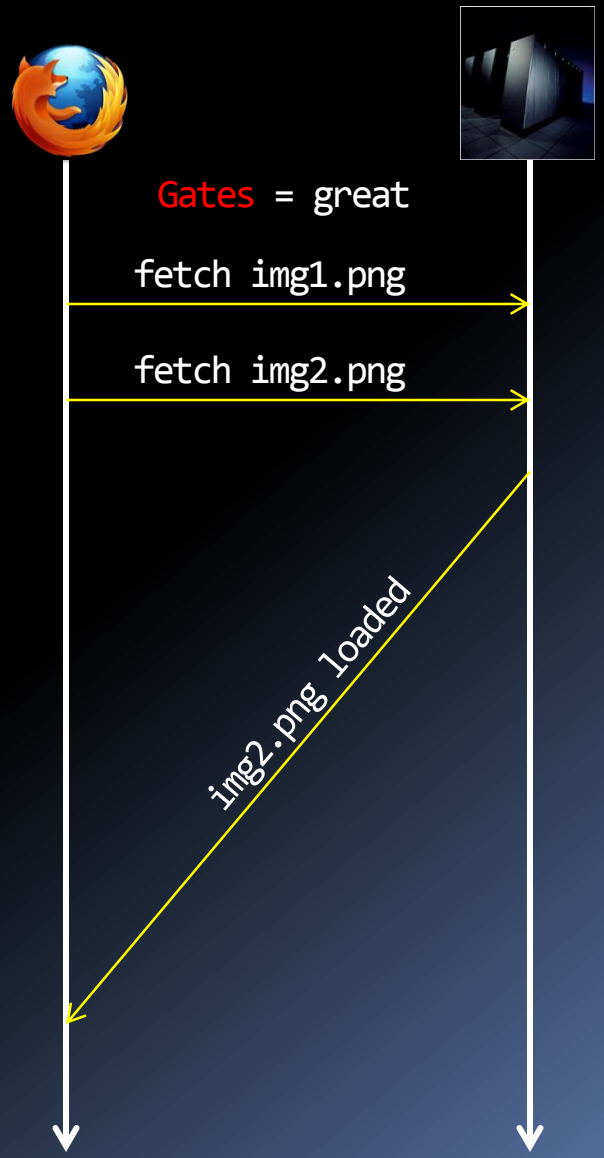


Non-determinism: network latency

```
<html>
<head></head>

<body>
<script>
var Gates = “great”;</script>
</script>

<img src=“img1.png” onload=“Gates=‘poor’;”>
<img src=“img2.png” onload=“alert(Gates);”>

</body>
</html>
```

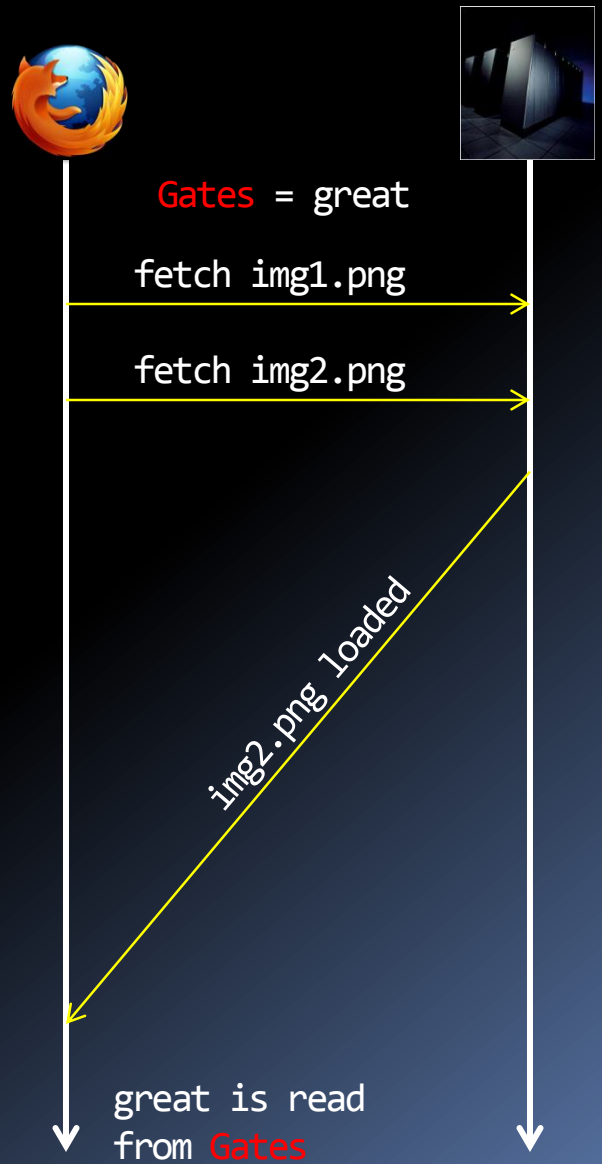
Non-determinism: network latency

```
<html>
<head></head>

<body>
<script>
var Gates = “great”;</script>
</script>

<img src=“img1.png” onload=“Gates=‘poor’;”>
<img src=“img2.png” onload=“alert(Gates);”>

</body>
</html>
```



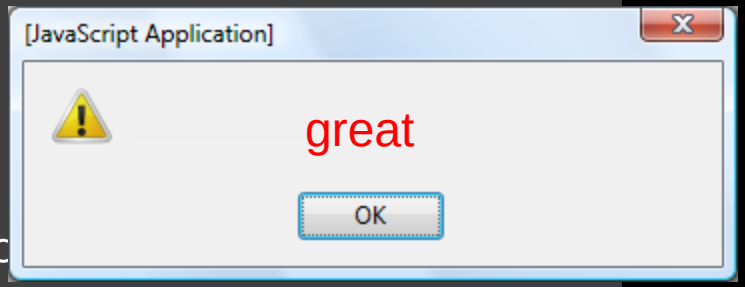
Non-determinism: network latency

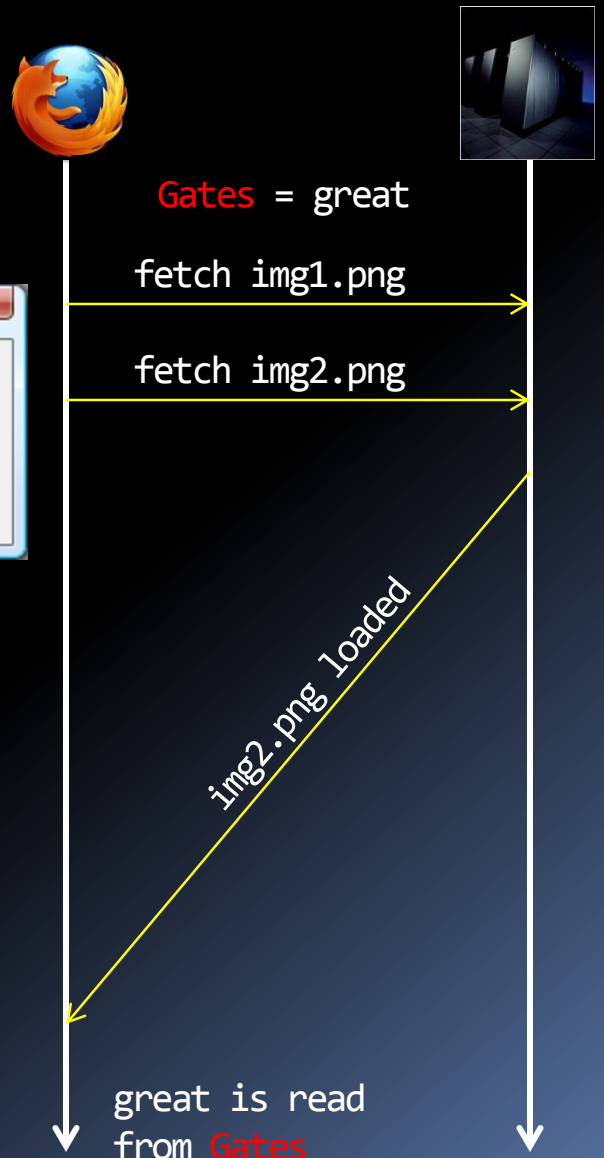
```
<html>
<head></head>

<body>
<script>
var Gates = “great”;</script>
</script>

<img src=“img1.png” onload=“Gates=‘poor’;”>
<img src=“img2.png” onload=“alert(Gates);”>

</body>
</html>
```



Non-determinism: network latency

```
<html>
<head></head>

<body>
<script>
var Gates = “great”;</sc
</script>

<img src=“img1.png” onload=“Gates=‘poor’;”>
<img src=“img2.png” onload=“alert(Gates);”>

</body>
</html>
```



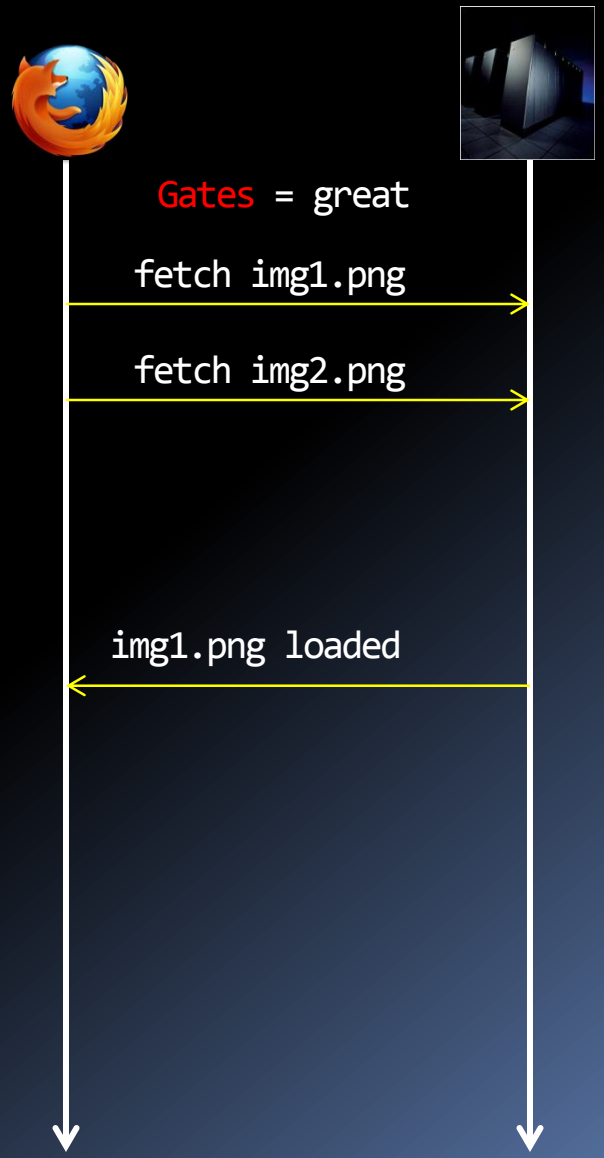


Non-determinism: network latency

```
<html>
<head></head>

<body>
<script>
var Gates = “great”;</script>
</script>

<img src=“img1.png” onload=“Gates=‘poor’;”>
<img src=“img2.png” onload=“alert(Gates);”>

</body>
</html>
```

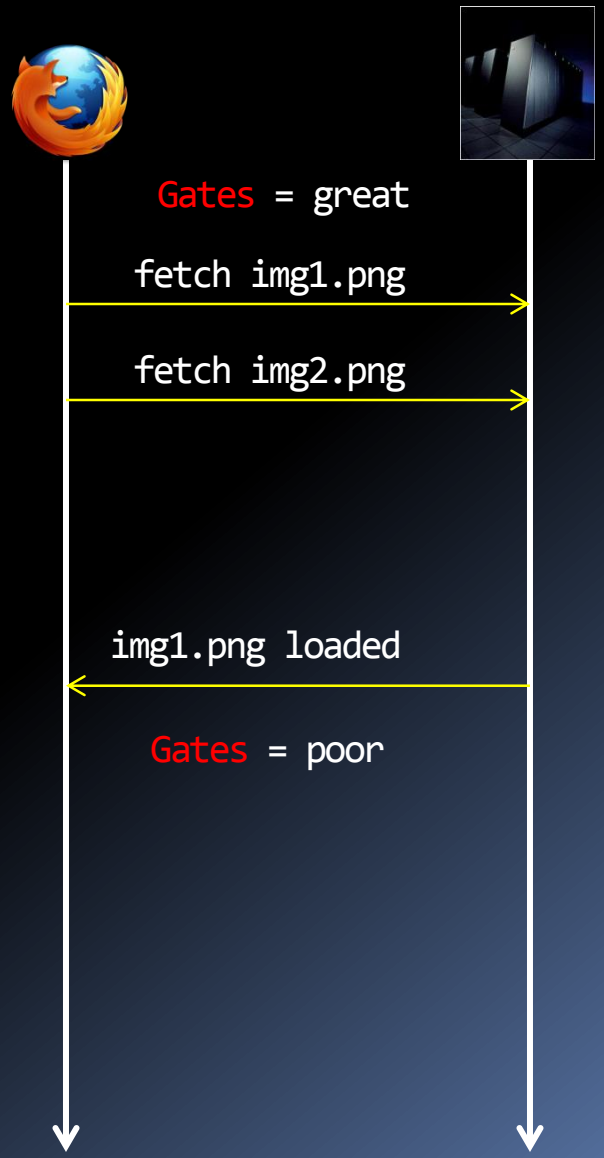
Non-determinism: network latency

```
<html>
<head></head>

<body>
<script>
var Gates = “great”;</script>
</script>

<img src=“img1.png” onload=“Gates=‘poor’;”>
<img src=“img2.png” onload=“alert(Gates);”>

</body>
</html>
```



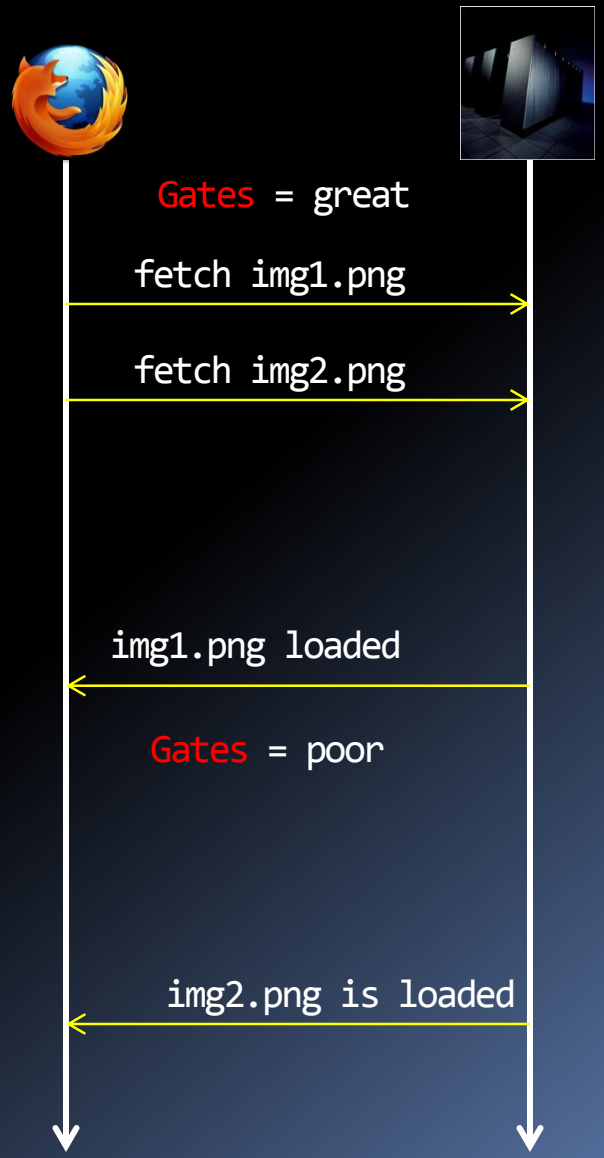
Non-determinism: network latency

```
<html>
<head></head>

<body>
<script>
var Gates = “great”;</script>
</script>

<img src=“img1.png” onload=“Gates=‘poor’;”>
<img src=“img2.png” onload=“alert(Gates);”>

</body>
</html>
```



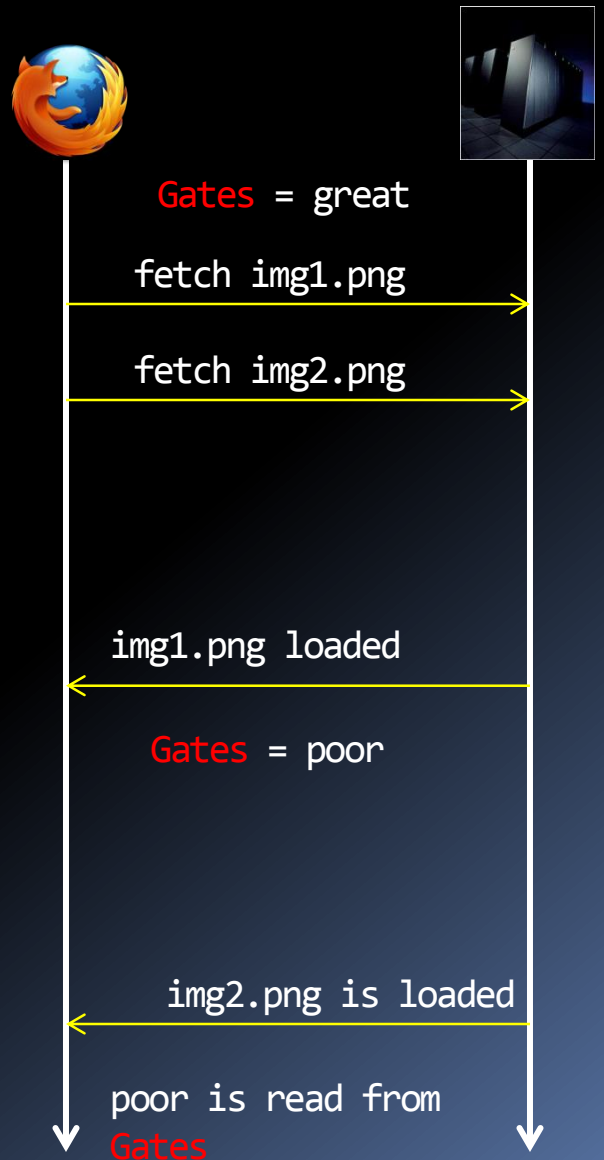
Non-determinism: network latency

```
<html>
<head></head>

<body>
<script>
var Gates = “great”;</script>
</script>

<img src=“img1.png” onload=“Gates=‘poor’;”>
<img src=“img2.png” onload=“alert(Gates);”>

</body>
</html>
```



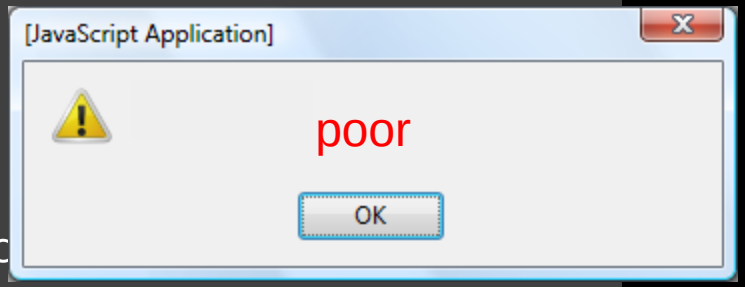
Non-determinism: network latency

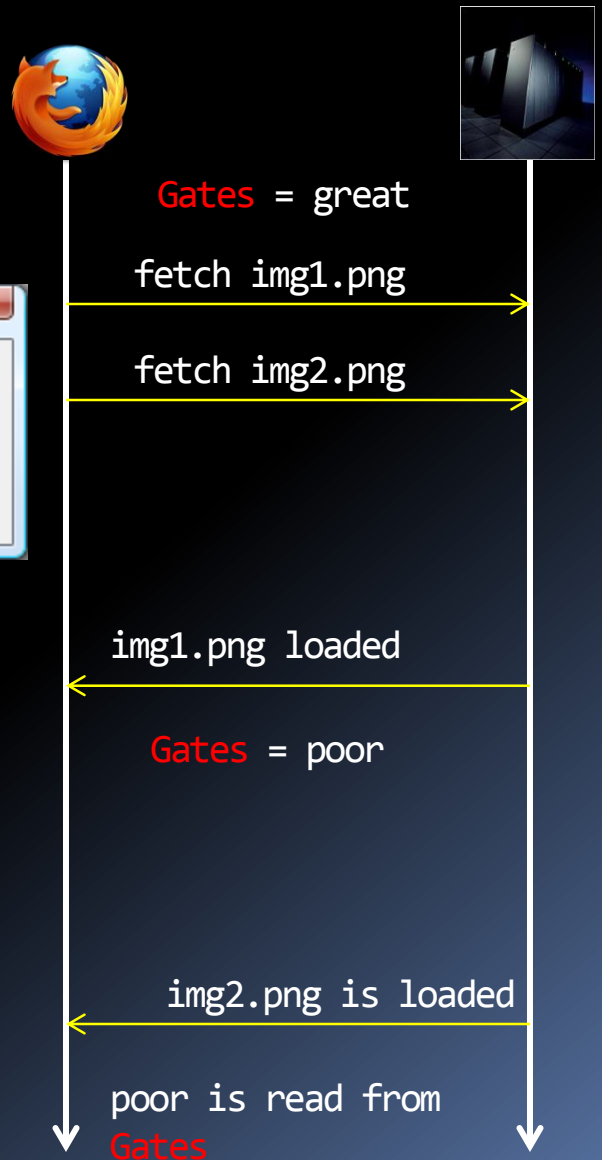
```
<html>
<head></head>

<body>
<script>
var Gates = “great”;</script>
</script>

<img src=“img1.png” onload=“Gates=‘poor’;”>
<img src=“img2.png” onload=“alert(Gates);”>

</body>
</html>
```



Non-determinism: network latency

```
<html>
<head></head>

<body>
<script>
var Gates = “great”;</sc
</script>

<img src=“img1.png” onload=“Gates=‘poor’;”>
<img src=“img2.png” onload=“alert(Gates);”>

</body>
</html>
```





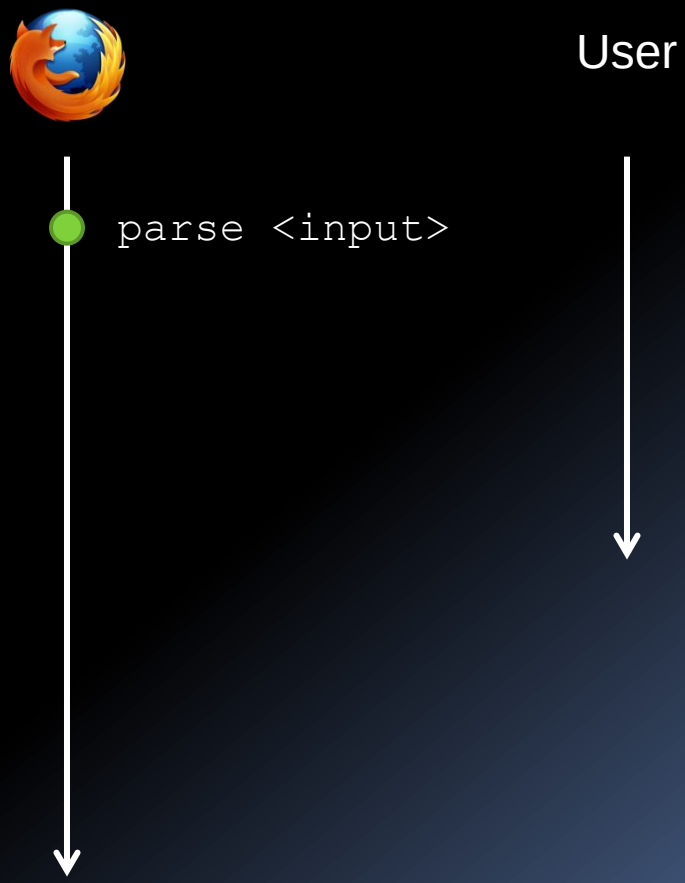
Non-determinism: user interaction

```
<html><body>

// Lots of code

 <input type="button" id="b1"
    onclick="javascript:f()">

// Lots of code

 <script>
  f = function() {
   alert("hello");
  }
 </script>

…
</body></html>
```

User

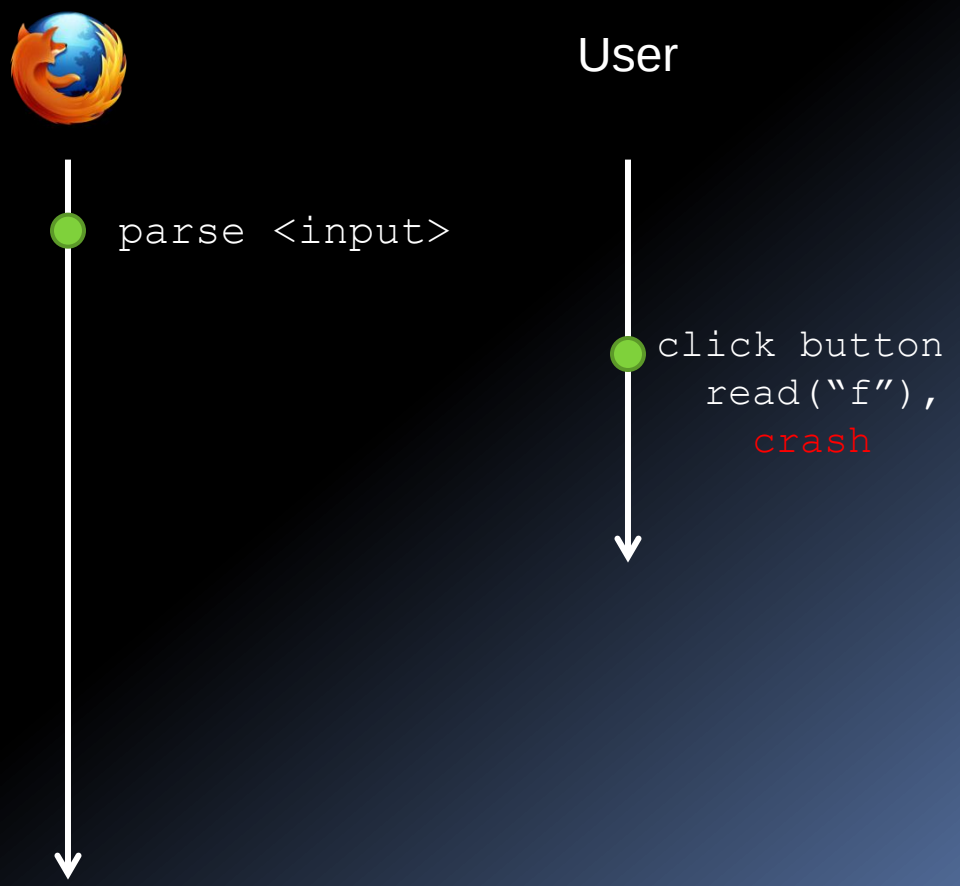
Non-determinism: user interaction

```
<html><body>

// Lots of code

 <input type="button" id="b1"
    onclick="javascript:f()">

// Lots of code

 <script>
  f = function() {
   alert("hello");
  }
 </script>

…
</body></html>
```



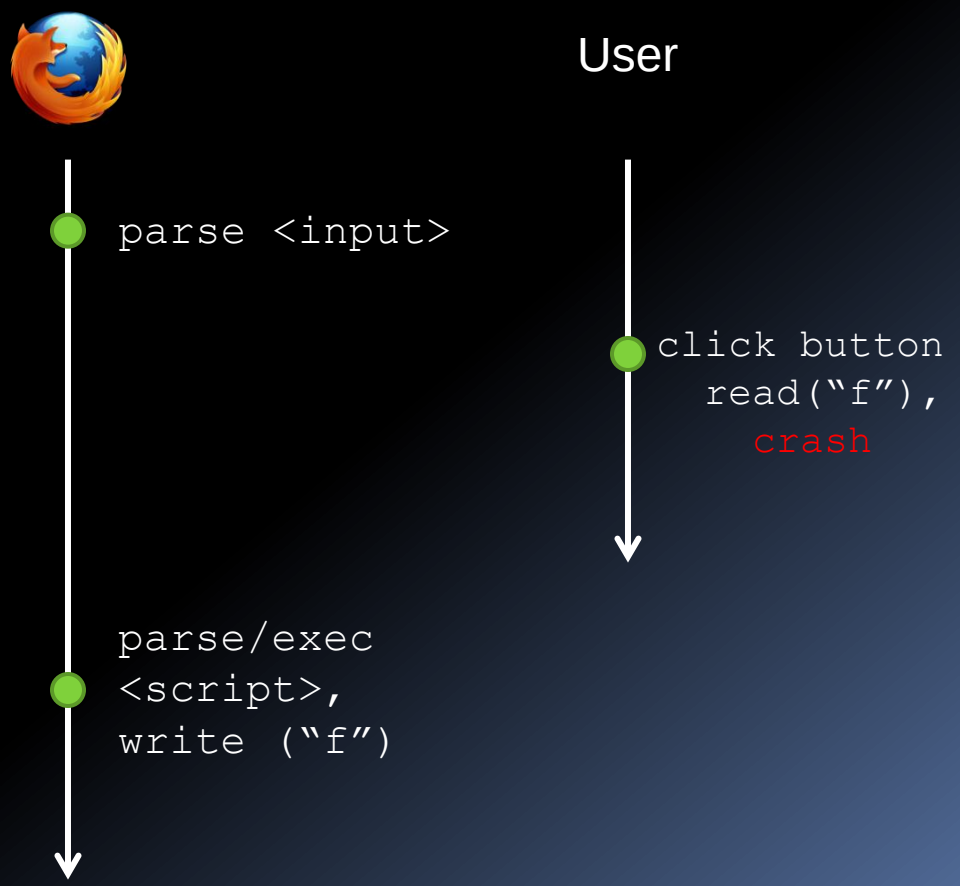
Non-determinism: user interaction

```
<html><body>

// Lots of code

 <input type="button" id="b1"
    onclick="javascript:f()">

// Lots of code

 <script>
  f = function() {
   alert("hello");
  }
 </script>

...
</body></html>
```



Non-determinism: user interaction

```
<html><body>

// Lots of code

 <input type="button" id="b1"
    onclick="javascript:f()">

// Lots of code

 <script>
  f = function() {
   alert("hello");
  }
 </script>

...
</body></html>
```

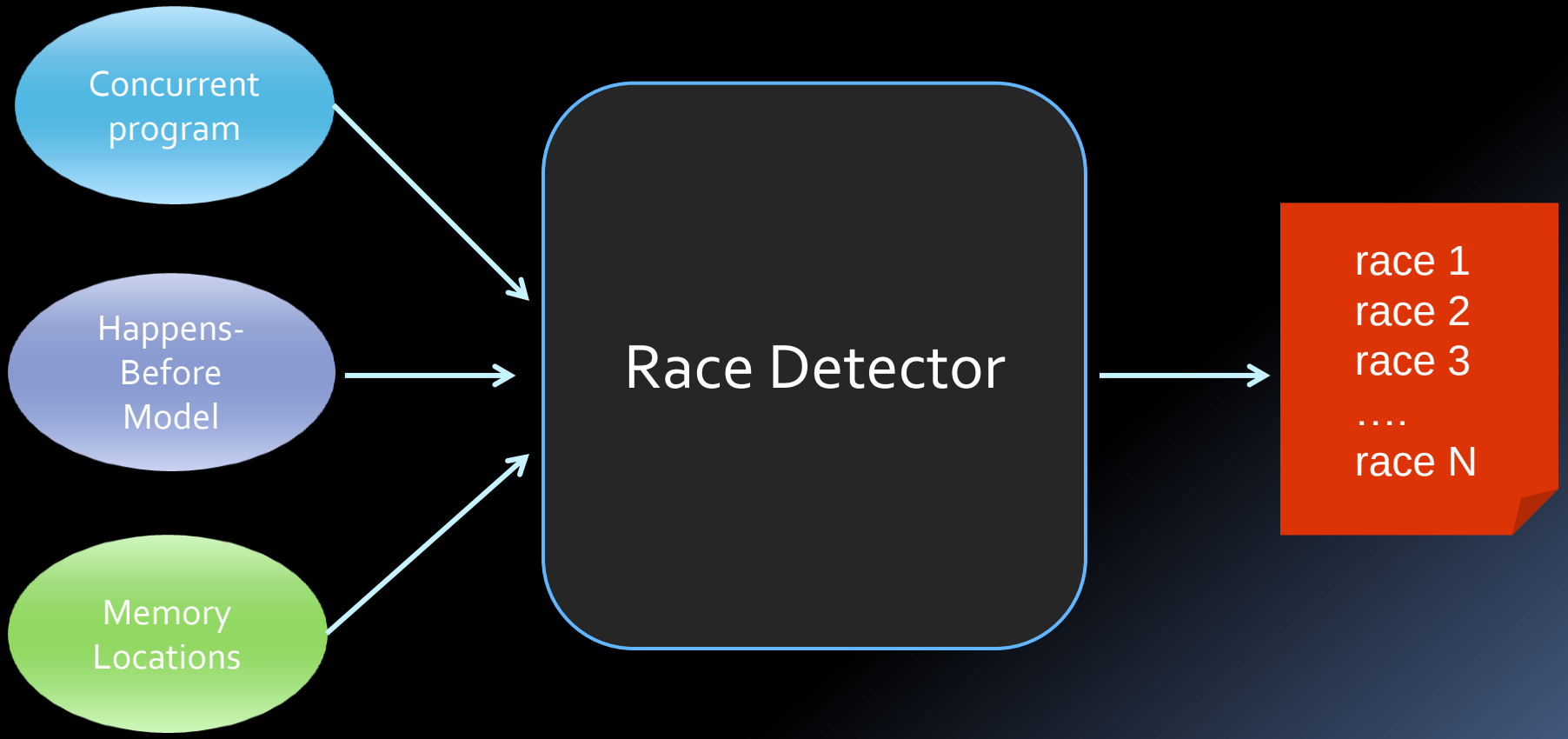


What do we learn from these?

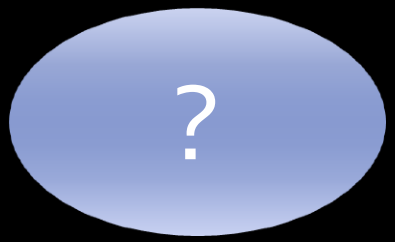
- Asynchrony causes non-determinism which may cause unwanted behavior
- Non-determinism is caused by interfering unordered accesses to shared locations
  - can be seen as data races

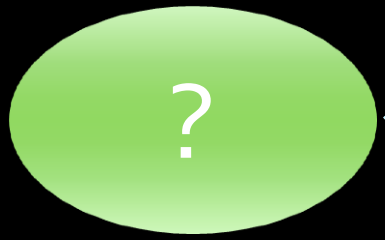
Can we detect such data races?

Race Detection Template



Race Detection: Web





Memory locations

- "Normal", C-like, memory locations for JavaScript variables
- Functions are treated like "normal" locations
- HTML DOM elements
- Event, event-target and event-handler tuple

Happens-Before: Ingredients

- What is an atomic action ?
  - E.g.: parsing a single HTML element, executing a script, processing an event handler
- How to order actions ?
  - E.g.: parsing of HTML elements of the page is ordered
- Laborious to define: go over HTML5 spec
  - Browser differences…

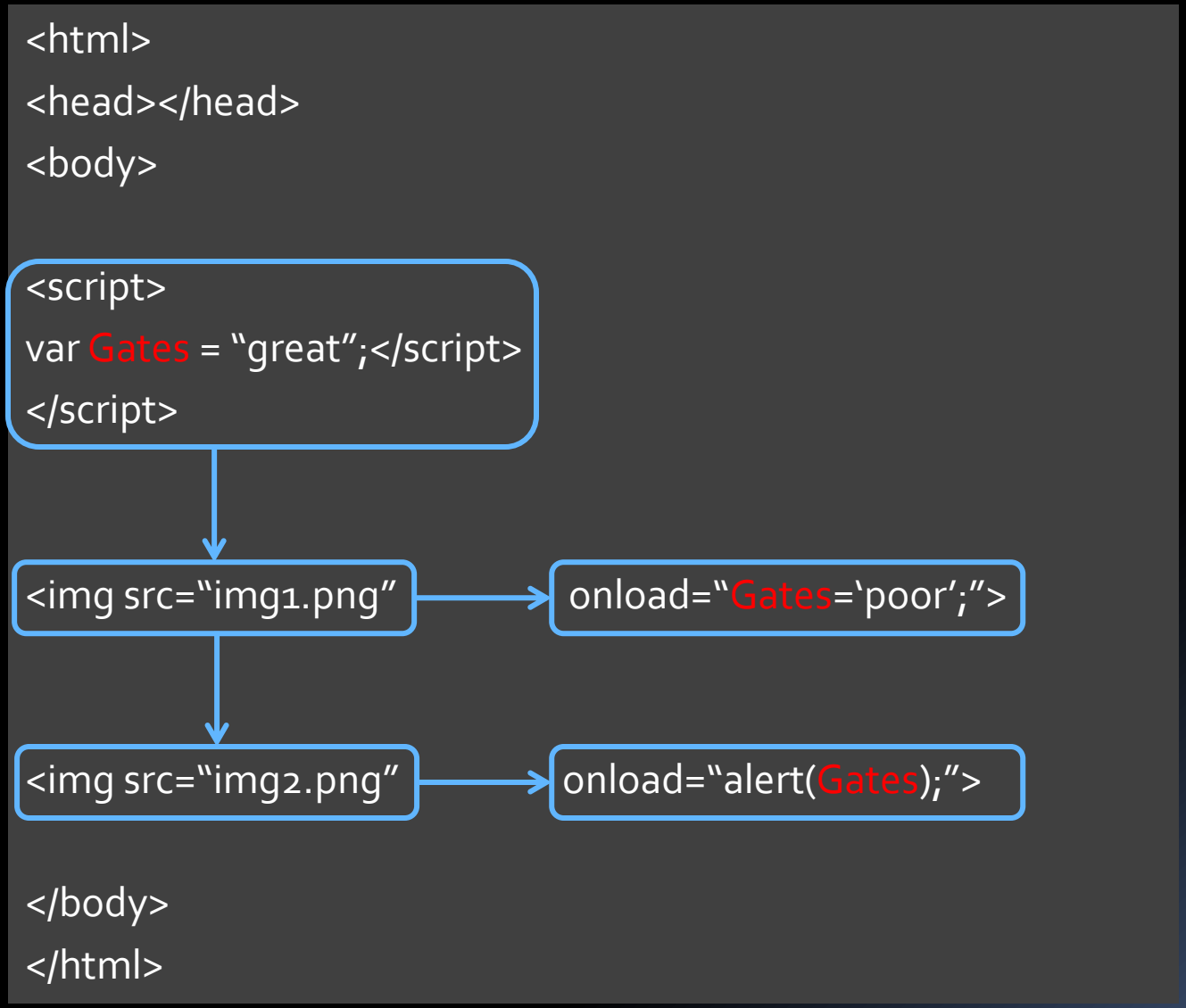
Example of Happens-Before

```
<html>
<head></head>
<body>

<script>
var Gates = “great”;</script>
</script>


<img src=“img1.png”              onload=“Gates=‘poor’;”>


<img src=“img2.png”              onload=“alert(Gates);”>

</body>
</html>
```

Example of Happens-Before

```
<html>
<head></head>
<body>

<script>
var Gates = "great";</script>
</script>

<img src="img1.png"        onload="Gates='poor';">

<img src="img2.png"        onload="alert(Gates);">

</body>
</html>
```

Example of Happens-Before



Race Detection Template

Happens-Before Model

Memory Locations

?

?

We will explore dynamic race detectors

Race Detection for Web: Challenges

- Precision: state-of-the-art detectors lead to too many false positives
  - caused by synchronization with read/writes, very common on the Web
- Scalability: state-of-the-art race detectors do not scale
  - blow-up in size of data structures caused by too many event handlers

Race Detection for Web: Challenges

- Precision: state-of-the-art detectors lead to too many false positives
  - caused by synchronization with read/writes, very common on the Web
- Scalability: state-of-the-art race detectors do not scale
  - blow-up in size of data structures caused by too many event handlers

Precision Issue: Example

```
<html><body>

<script>
 var init = false, y = null;
 function f() {
   if (init)
     alert(y.g);
   else
     alert("not ready");
 }
</script>

<input type="button" id="b1"
 onclick="javascript:f()">

<script>
  y = { g:42 };
  init = true;
</script>

</body></html>
```

- 3 variables with races:
  init
  y
  y.g
- some races are synchronization:
  init
- reports false races:
  y
  y.g

Wanted: “guaranteed” races

```
<html><body>

<script>
 var init = false, y = null;
 function f() {
   if (init)
     alert(y.g);
   else
     alert("not ready");
 }
</script>

<input type="button" id="b1"
 onclick="javascript:f()">

<script>
  y = { g:42 };
  init = true;
</script>

</body></html>
```

Intuition: identify races that are guaranteed to exist.

We want to report races on variable

init

But not on:

y

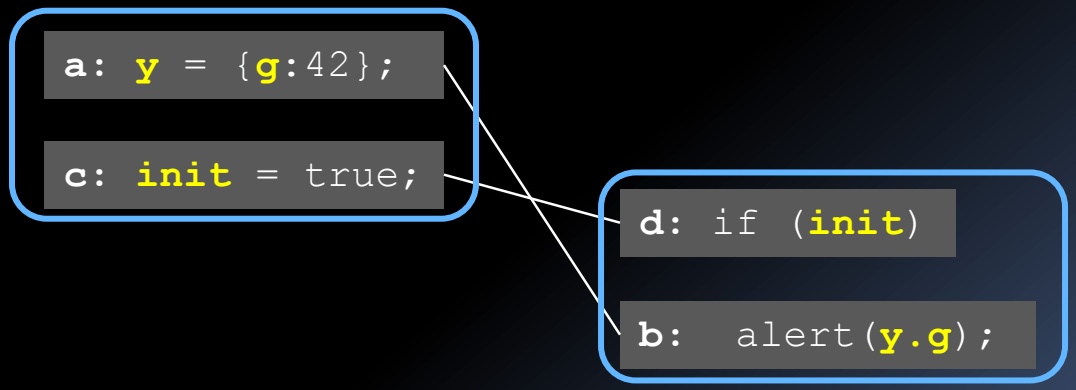
y.g

Because fixing the races on init will always remove all races on y and g (in this trace).

Wanted: “guaranteed” races

```
<html><body>

<script>
 var init = false, y = null;
 function f() {
   if (init)
     alert(y.g);
   else
     alert("not ready");
 }
</script>

<input type="button" id="b1"
 onclick="javascript:f()">

<script>
  y = { g:42 };
  init = true;
</script>

</body></html>
```

A race (c, d) is guaranteed if in one trace we see c … d and in another trace we see d … c

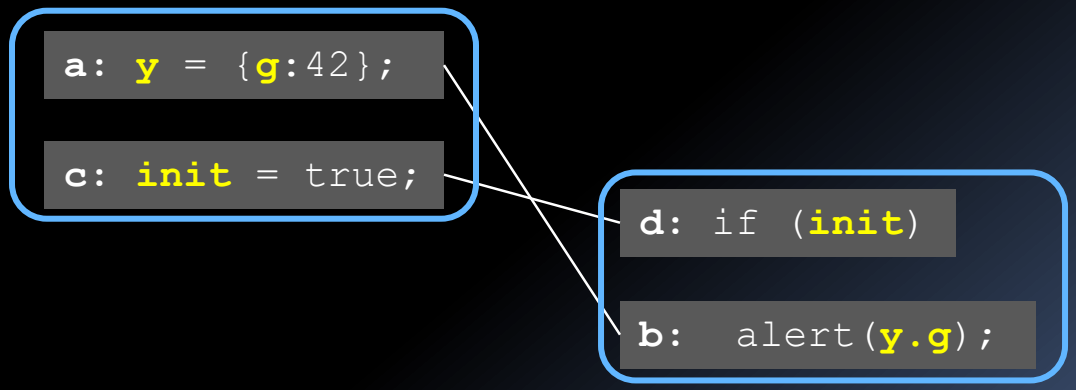


Here, race (c,d) is guaranteed

Wanted: “guaranteed” races

```
<html><body>

<script>
 var init = false, y = null;
 function f() {
   if (init)
     alert(y.g);
   else
     alert("not ready");
 }
</script>

<input type="button" id="b1"
 onclick="javascript:f()">

<script>
  y = { g:42 };
  init = true;
</script>

</body></html>
```

A race (c, d) is guaranteed if in one trace we see c … d and in another trace we see d … c



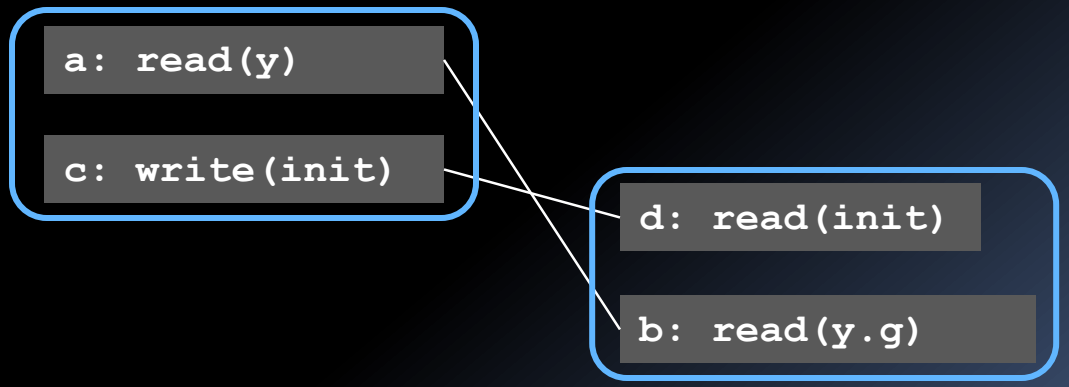
Approach: record the full program trace and then compute data-dependence, etc…

Expensive !

Wanted: “guaranteed” races

```
<html><body>

<script>
 var init = false, y = null;
 function f() {
   if (init)
     alert(y.g);
   else
     alert("not ready");
 }
</script>

<input type="button" id="b1"
 onclick="javascript:f()">

<script>
  y = { g:42 };
  init = true;
</script>

</body></html>
```

An abstraction of the program trace:

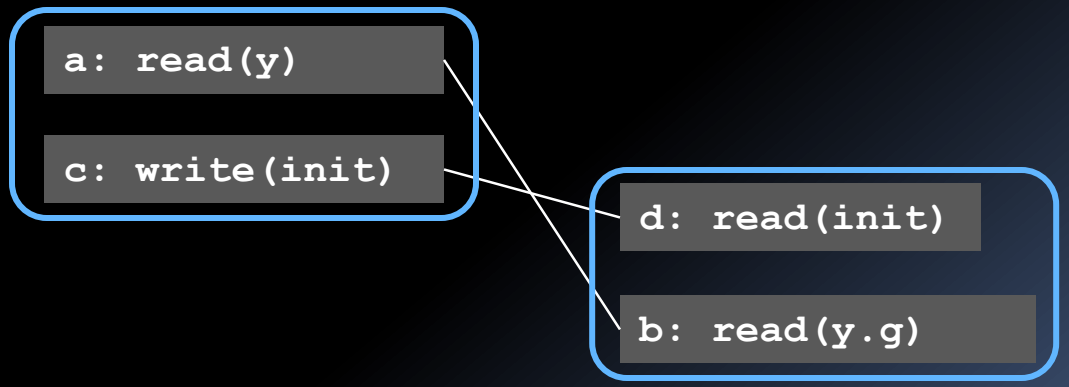


Common approach: record only shared reads and writes.

Wanted: “guaranteed” races

```
<html><body>

<script>
 var init = false, y = null;
 function f() {
   if (init)
     alert(y.g);
   else
     alert("not ready");
 }
</script>

<input type="button" id="b1"
 onclick="javascript:f()">

<script>
  y = { g:42 };
  init = true;
</script>

</body></html>
```

An abstraction of the program trace:

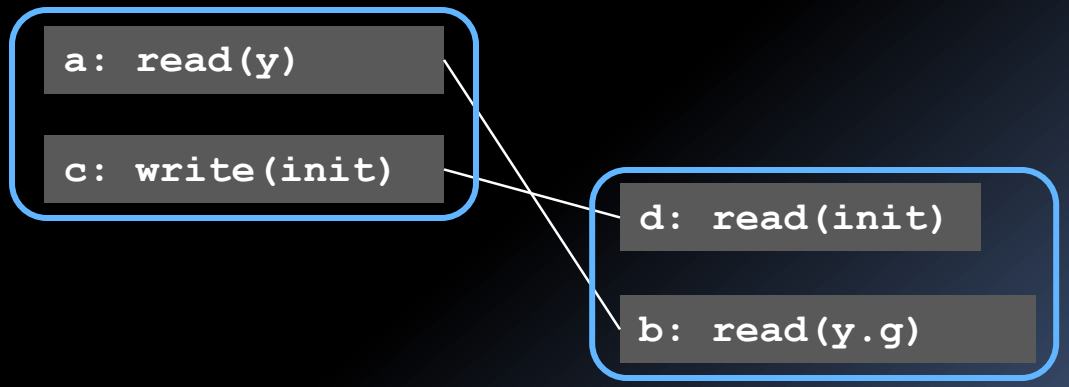


Common approach: record only shared reads and writes.

Wanted: “guaranteed” races

```
<html><body>

<script>
 var init = false, y = null;
 function f() {
   if (init)
     alert(y.g);
   else
     alert("not ready");
 }
</script>

<input type="button" id="b1"
 onclick="javascript:f()">

<script>
  y = { g:42 };
  init = true;
</script>

</body></html>
```

An abstraction of the program trace:



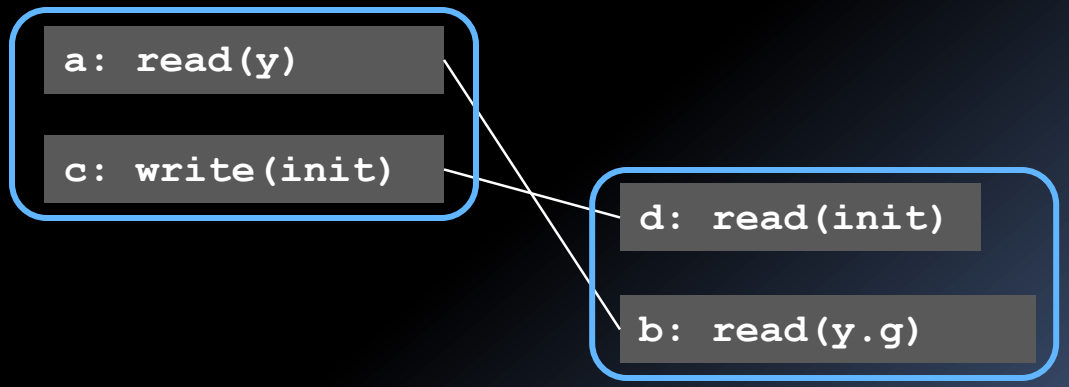
Common approach: record only shared reads and writes.

Wanted: “guaranteed” races

```
<html><body>

<script>
 var init = false, y = null;
 function f() {
   if (init)
     alert(y.g);
   else
     alert("not ready");
 }
</script>

<input type="button" id="b1"
 onclick="javascript:f()">

<script>
  y = { g:42 };
  init = true;
</script>

</body></html>
```

Which races are “guaranteed to exist” ?

An abstraction of the program trace:

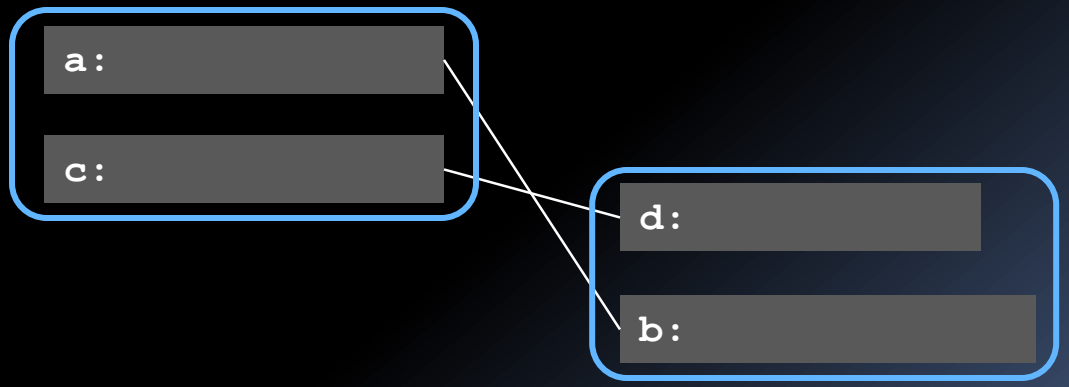


Common approach: record only shared reads and writes.

Wanted: “guaranteed” races

```
<html><body>

<script>
 var init = false, y = null;
 function f() {
   if (init)
     alert(y.g);
   else
     alert("not ready");
 }
</script>

<input type="button" id="b1"
 onclick="javascript:f()">

<script>
  y = { g:42 };
  init = true;
</script>

</body></html>
```

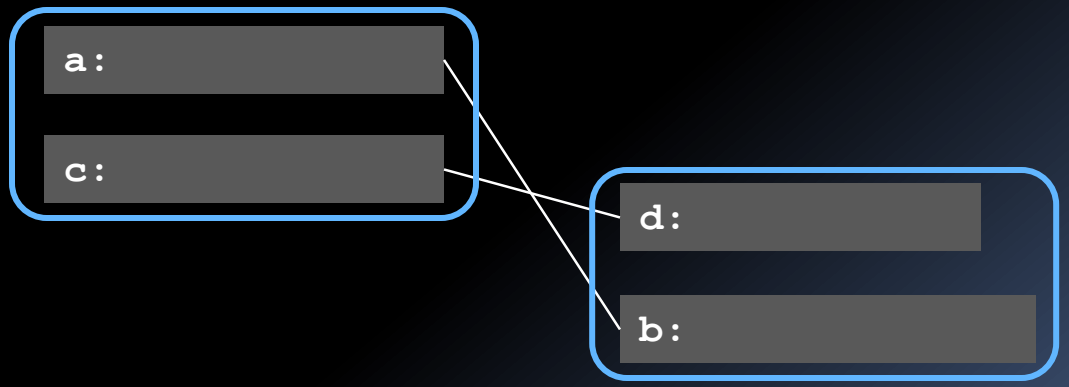
An abstraction of the program trace:



Wanted: “guaranteed” races

```
<html><body>

<script>
 var init = false, y = null;
 function f() {
   if (init)
     alert(y.g);
   else
     alert("not ready");
 }
</script>

<input type="button" id="b1"
 onclick="javascript:f()">

<script>
  y = { g:42 };
  init = true;
</script>

</body></html>
```

An abstraction of the program trace:

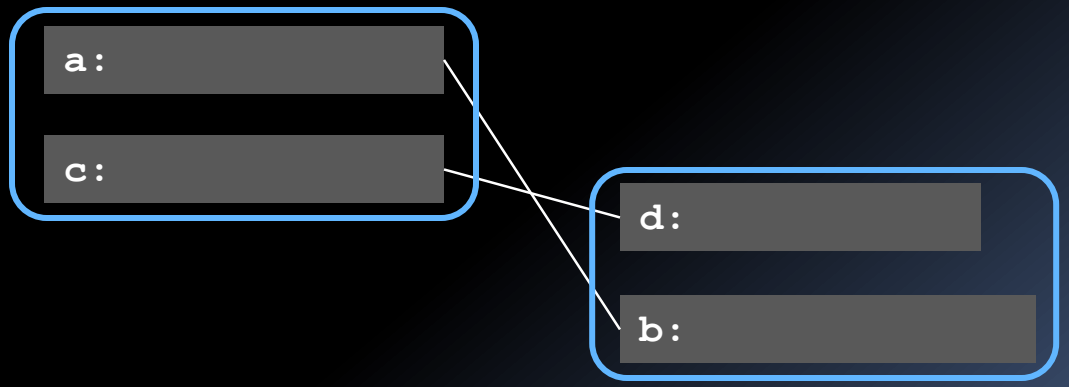


Wanted: “guaranteed” races

```
<html><body>

<script>
 var init = false, y = null;
 function f() {
   if (init)
     alert(y.g);
   else
     alert("not ready");
 }
</script>

<input type="button" id="b1"
 onclick="javascript:f()">

<script>
  y = { g:42 };
  init = true;
</script>

</body></html>
```

Which races are “guaranteed to exist” ?

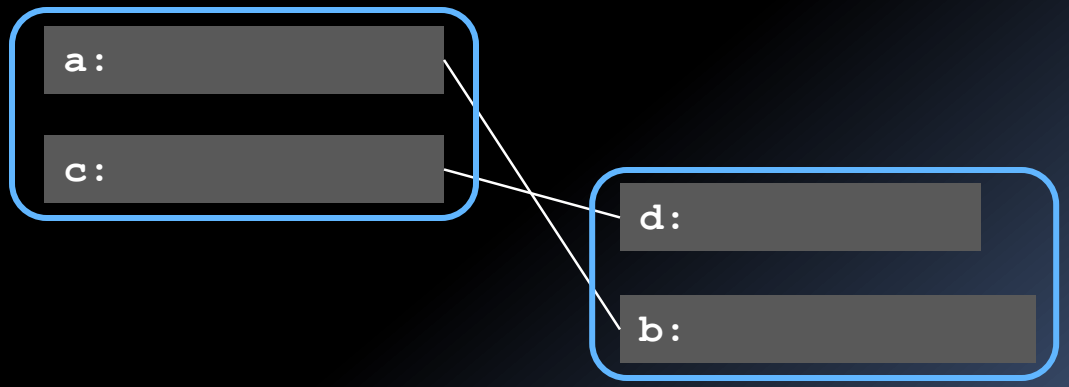
An abstraction of the program trace:



Wanted: “guaranteed” races

```
<html><body>

<script>
 var init = false, y = null;
 function f() {
   if (init)
     alert(y.g);
   else
     alert("not ready");
 }
</script>

<input type="button" id="b1"
 onclick="javascript:f()">

<script>
  y = { g:42 };
  init = true;
</script>

</body></html>
```

(c,d) is a guaranteed race if $\forall\ \pi' \in \gamma(\pi)$, c and d are a guaranteed race in the concrete trace π'



But how are we going to compute these In the abstract ?

Key Idea 1: Coverage

```
<html><body>

<script>
 var init = false, y = null;
 function f() {
   if (init)
     alert(y.g);
   else
     alert("not ready");
 }
</script>

<input type="button" id="b1"
 onclick="javascript:f()">

<script>
  y = { g:42 };
  init = true;
</script>

</body></html>
```

(c,d) is a guaranteed race if $\forall\ \pi' \in \gamma(\pi)$, c and d are a guaranteed race in the concrete trace π'

Definition of <R>

Theorem:

A <R> race is a guaranteed race

Key Idea 1: Coverage

```
<html><body>

<script>
 var init = false, y = null;
 function f() {
   if (init)
     alert(y.g);
   else
     alert("not ready");
 }
</script>

<input type="button" id="b1"
 onclick="javascript:f()">

<script>
  y = { g:42 };
  init = true;
</script>

</body></html>
```

(c,d) is a guaranteed race if $\forall\ \pi' \in \gamma(\pi)$, c and d are a guaranteed race in the concrete trace π'

Definition: race (c,d) covers race (a,b) if $a \preccurlyeq c$ (or a and c are in the same action), and $d \preccurlyeq b$.

Generalizes to coverage by multiple races

Theorem:

An uncovered race is a guaranteed race.

Race Detection for Web: Challenges

- Precision: state-of-the-art detectors lead to too many false positives
  - caused by synchronization with read/writes, very common on the Web
- Scalability: state-of-the-art race detectors do not scale
  - blow-up in size of data structures caused by too many event handlers

Race Detection for Web: Challenges

- Precision: state-of-the-art detectors lead to too many false positives
  - caused by synchronization with read/writes, very common on the Web
- Scalability: state-of-the-art race detectors do not scale
  - blow-up in size of data structures caused by too many event handlers

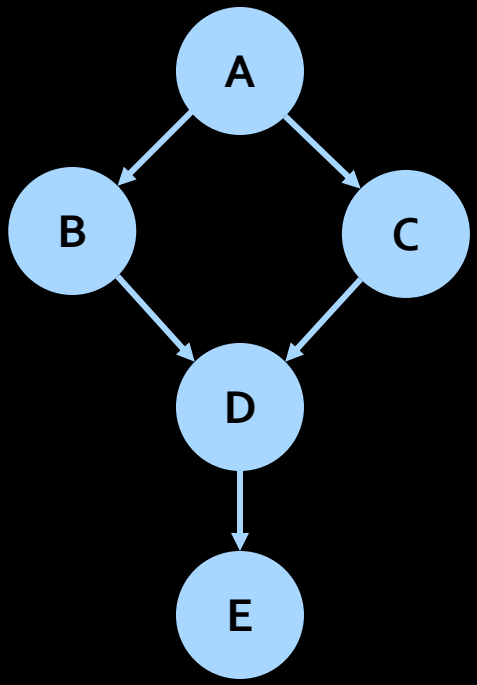
Computing Races

A race detector should compute races. The basic query is whether two operations a and b are ordered:

$$a \preccurlyeq b$$

Observation: represent $\preccurlyeq$ as a directed acyclic graph and perform graph connectivity queries to answer $a \preccurlyeq b$

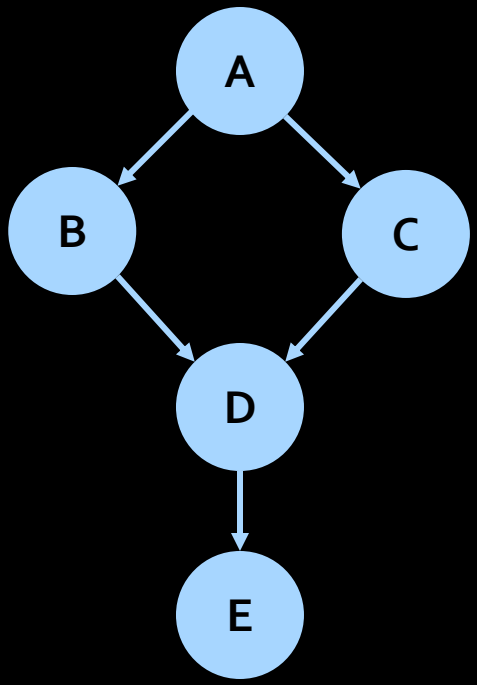
The happens-before graph



For this graph:

$A \prec B$
$A \prec C$
$B \prec D$
$C \prec D$
$D \prec E$
$A \prec D$
$A \prec E$
$C \prec E$
$B \prec E$
$B \not\prec C$

a $\overset{?}{\rightsquigarrow}$ b via BFS

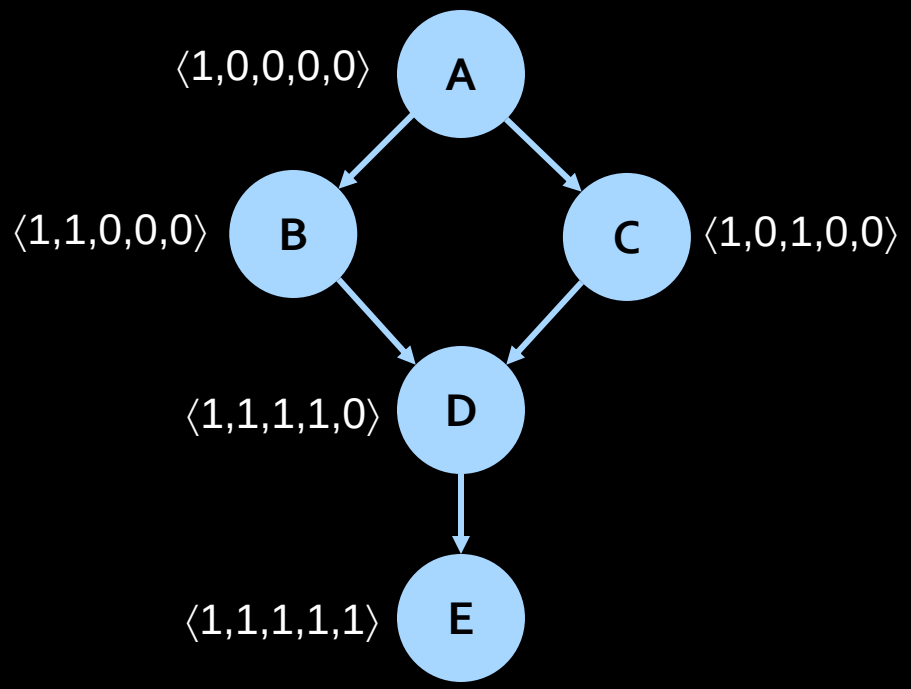


M - number of edges
N - number of nodes

Query Time: O(M)
Space : O(N)

$a \overset{?}{\preccurlyeq} b$ via vector clocks

(classic race detection)



A vector clock vc is a map:

$$vc \in TID \rightarrow N$$

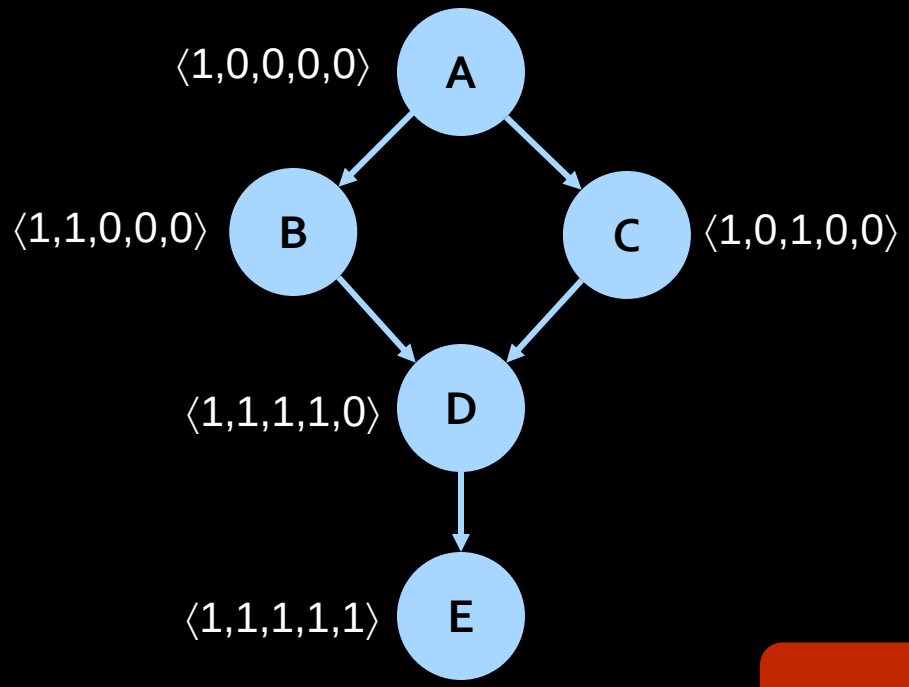
associate a vector clock with each node

<1,0,0,0,0> $\sqsubseteq$ <1,1,1,1,0>
it follows that A $\preccurlyeq$ D

<1,1,0,0,0> $\not\sqsubseteq$ <1,0,1,0,0>
it follows that B $\not\preccurlyeq$ C

$a \overset{?}{\preccurlyeq} b$ via vector clocks

(classic race detection)



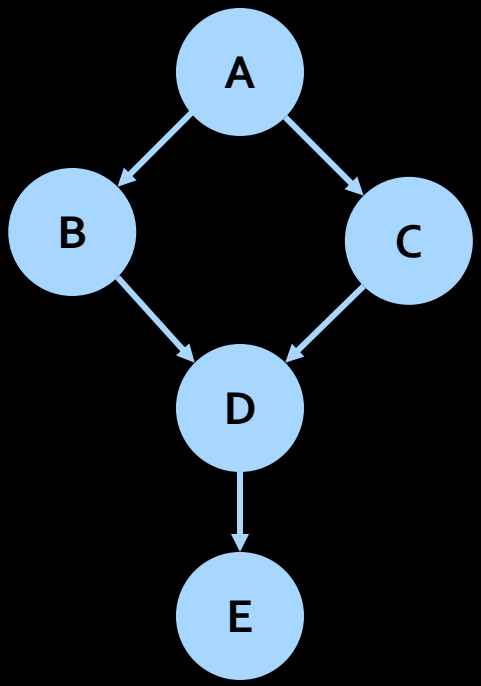
Pre-computation Time: $O(M \cdot N)$

Query Time: $O(1)$

Space: $O(N^2)$

Space Explosion

$a \overset{?}{\preccurlyeq} b$ via combining chain decomposition with vector clocks



Key idea: Re-discover threads by partitioning the nodes into chains.

due to:

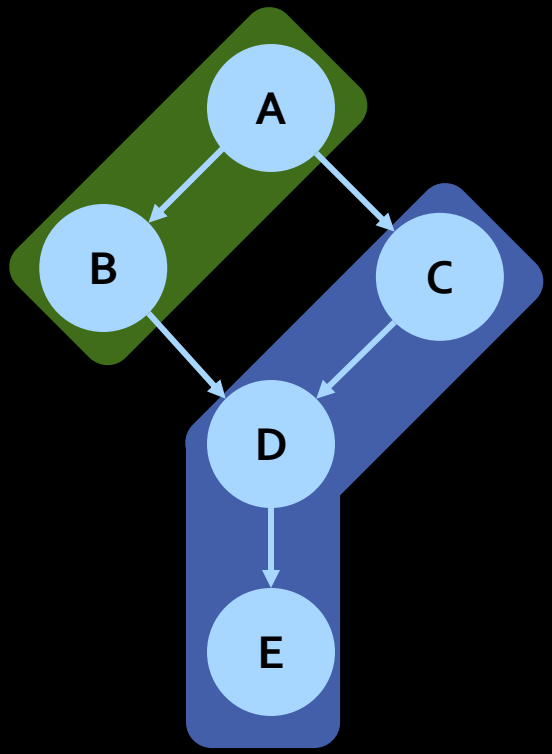
"A Compression Technique to Materialize Transitive Closure", 1990, H.V. Jagadish ACM Trans. Database Syst

computes a map:

$c \in \text{Nodes} \rightarrow \text{ChainIDs}$

associate a chain with each node

$a \overset{?}{\lesssim} b$ via combining chain decomposition with vector clocks



Key idea: Re-discover threads by partitioning the nodes into chains.

due to:

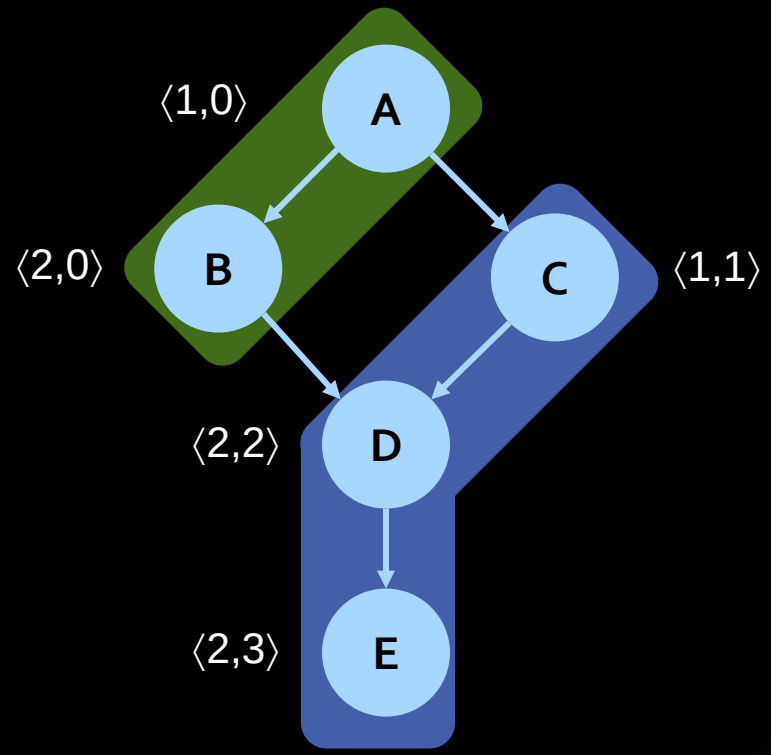
"A Compression Technique to Materialize Transitive Closure", 1990, H.V. Jagadish ACM Trans. Database Syst

computes a map:

$c \in \text{Nodes} \rightarrow \text{ChainIDs}$

associate a chain with each node

$a \overset{?}{\preccurlyeq} b$ via combining chain decomposition with vector clocks (optimal version)



C = number of chains

Chain Computation Time: $O(N^3 + C \cdot M)$

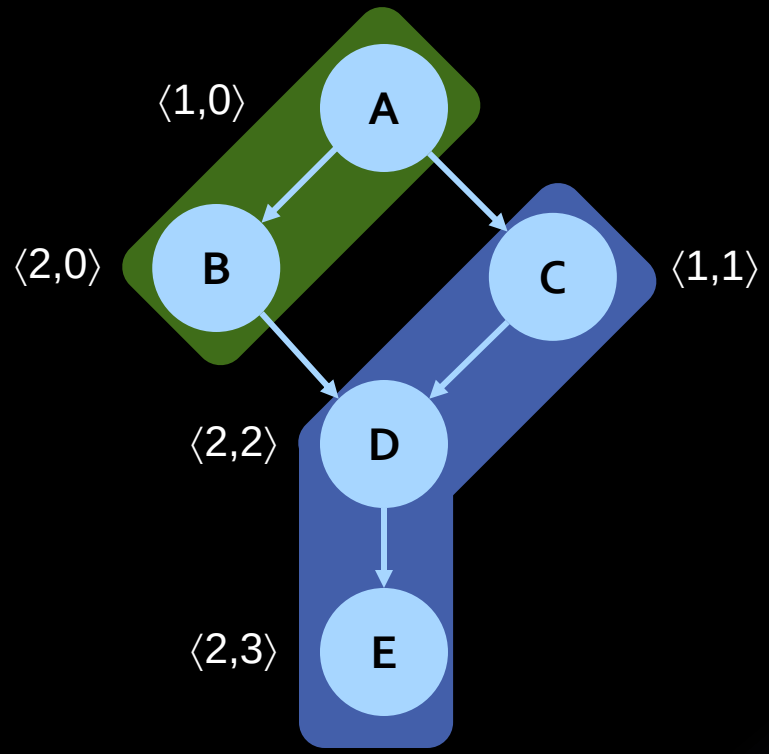
Vector clock computation: $O(C \cdot M)$

Query Time: $O(1)$

Space: $O(C \cdot N)$

Improved

$a \overset{?}{\lesssim} b$ via combining chain decomposition with vector clocks (greedy version)



C = number of chains

Chain Computation Time: $O(C \cdot M)$

Vector clock computation: $O(C \cdot M)$

Query Time: $O(1)$

Space: $O(C \cdot N)$

Race Detection: Web

Race Detector

Happens-Before Model

Memory Locations

race coverage

vector clocks

chain decomposition

?

Pre-computation Time: $O(C \cdot M)$

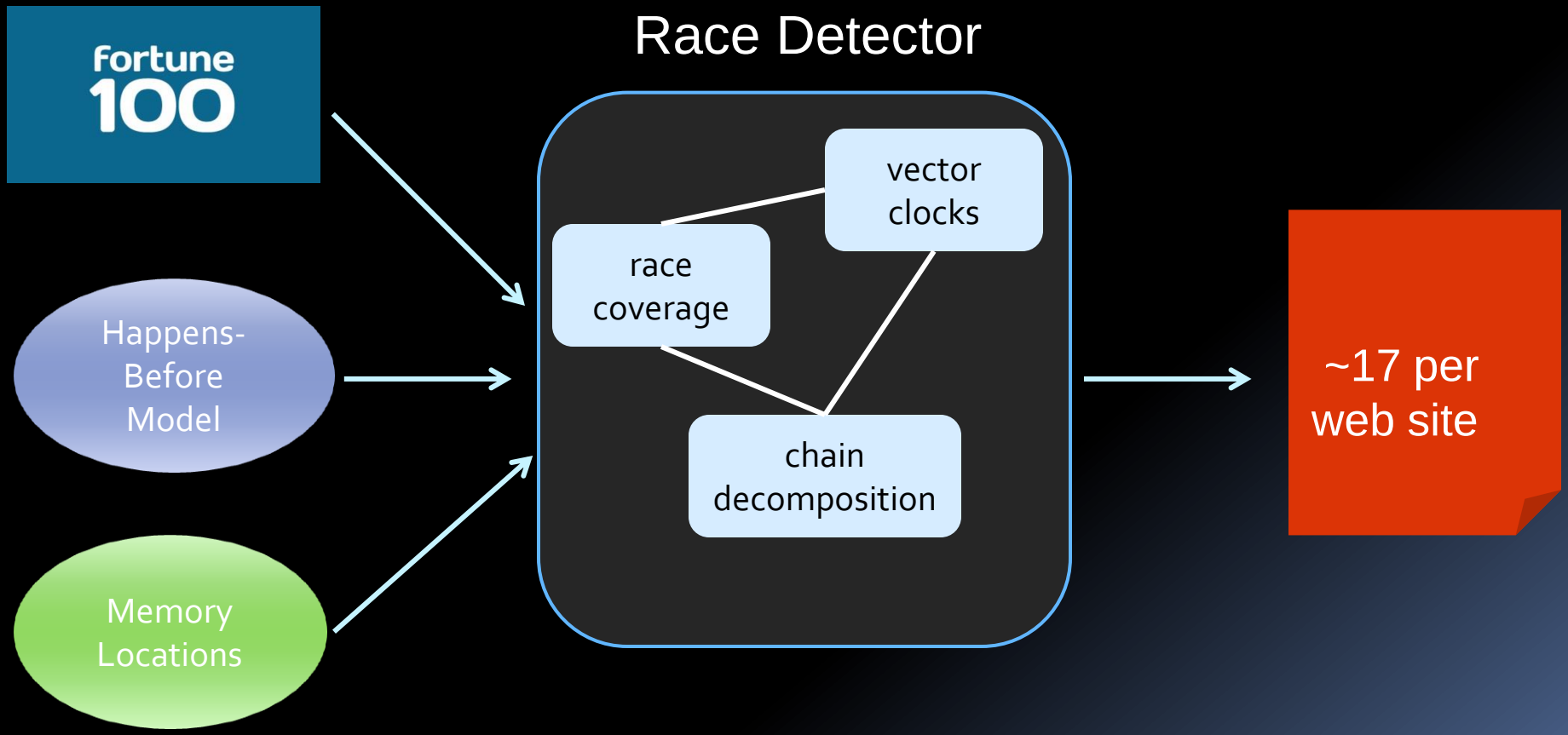
Query Time: $O(1)$

Space: $O(C \cdot N)$

Implementation

- Based on WebKit Browser
  - Used by Apple's Safari and Google's Chrome
- Quite robust, Demo:
  - http://www.eventracer.org

Experiments: Forune 100 web sites



Experiments: usability

Metric	Mean # race vars	Max # race vars
All	**634.6**	3460
Only uncovered races	**45.3**	331
Filtering methods		
Writing same value	0.75	12
Only local reads	3.42	43
Late attachment of event handler	16.7	117
Lazy initialization	4.3	61
Commuting operations - className, cookie	4.0	80
Race with unload	1.1	33
Remaining after all filters	**17.8**	261

Experiments: speed

Metric	Mean	Max
Number of event actions	5868	114900
Number of chains	175	792
Graph connectivity algorithm		
Vector clocks w/o chain decomposition	>0.1sec	OOM
Vector clocks + chain decomposition	0.04sec	2.4sec
Breadth-first search	>22sec	TIMEOUT

Experiments: space

Metric	Mean	Max
Number of event actions	5868	114900
Number of chains	175	792
Graph connectivity algorithm		
Vector clocks w/o chain decomposition	544MB	25181MB
Vector clocks + chain decomposition	5MB	171MB

Manual inspection of 314 races

- 57% are synchronization races
  - many idioms: conditionals, try-catch, looping over arrays
- 24% are harmful races
  - many cases of reading from undefined
  - new bugs:UI glitches, broken functionality after a race, needs page refresh, missing event handlers, broken analytics.
- 17% are harmless races

Future Work

- Race Detection as Abstract Interpretation
- Generalized Race Detection to Commutativity
- Synthesis of Repairs
- Reachability algorithms based on graph contraction
  - inspired by algorithms for road networks
- Stateless model checking
  - race-guided exploration of the web page

Summary

- Introduced Happens-Before model for web applications
  - useful for any concurrency analysis
- Race coverage: report only real races
- Efficient Analysis
  - combines vector clocks, chain decomposition and race coverage

Try it out

http://www.eventracer.org