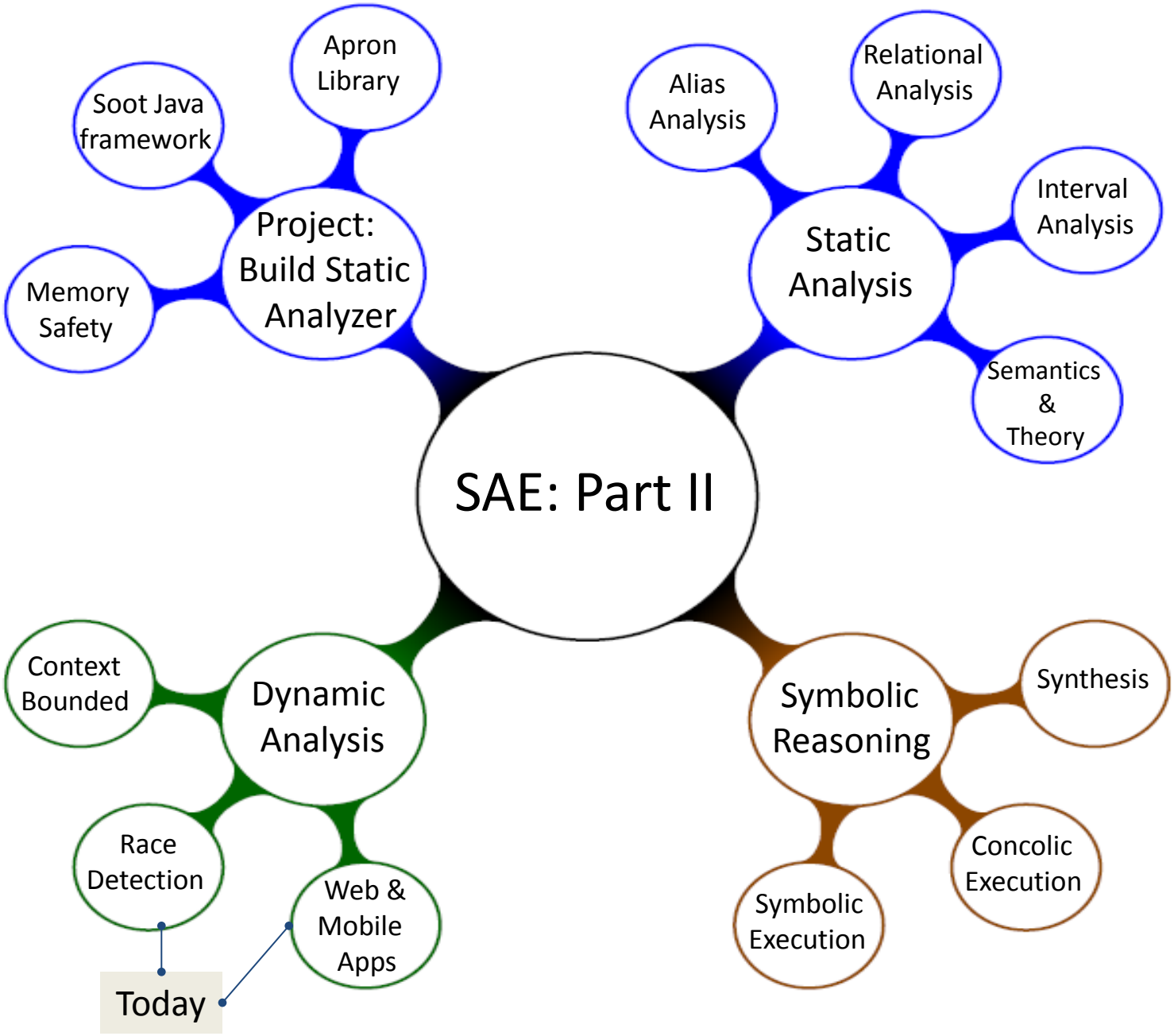
Software Architecture and Engineering: Part II

ETH Zurich, Spring 2014
Prof. Martin Vechev


SAE: Part II
Project: Build Static Analyzer
Apron Library
Soot Java framework
Memory Safety
Static Analysis
Alias Analysis
Relational Analysis
Interval Analysis
Semantics & Theory
Dynamic Analysis
Context Bounded
Race Detection
Web & Mobile Apps
Today
Symbolic Reasoning
Synthesis
Concolic Execution
Symbolic Execution

Dynamic Race Detection

- A popular kind of dynamic analysis
- Highly effective for finding concurrency bugs
- Many different variants
  - Trade-off between asymptotic complexity and precision of the analysis

Today

We will illustrate the key concepts of race detection on a rich application domain that is quite prevalent today, namely event-driven applications such as Web pages and Android

All concepts we study today apply to other settings: e.g. regular concurrent Java programs.

Motivation: Event-Driven Applications

~ 1 billion smartphones

~ 640 million web pages

Reacts to events: user clicks, arrival of network requests

Event-Driven Applications

Wanted: fast response time

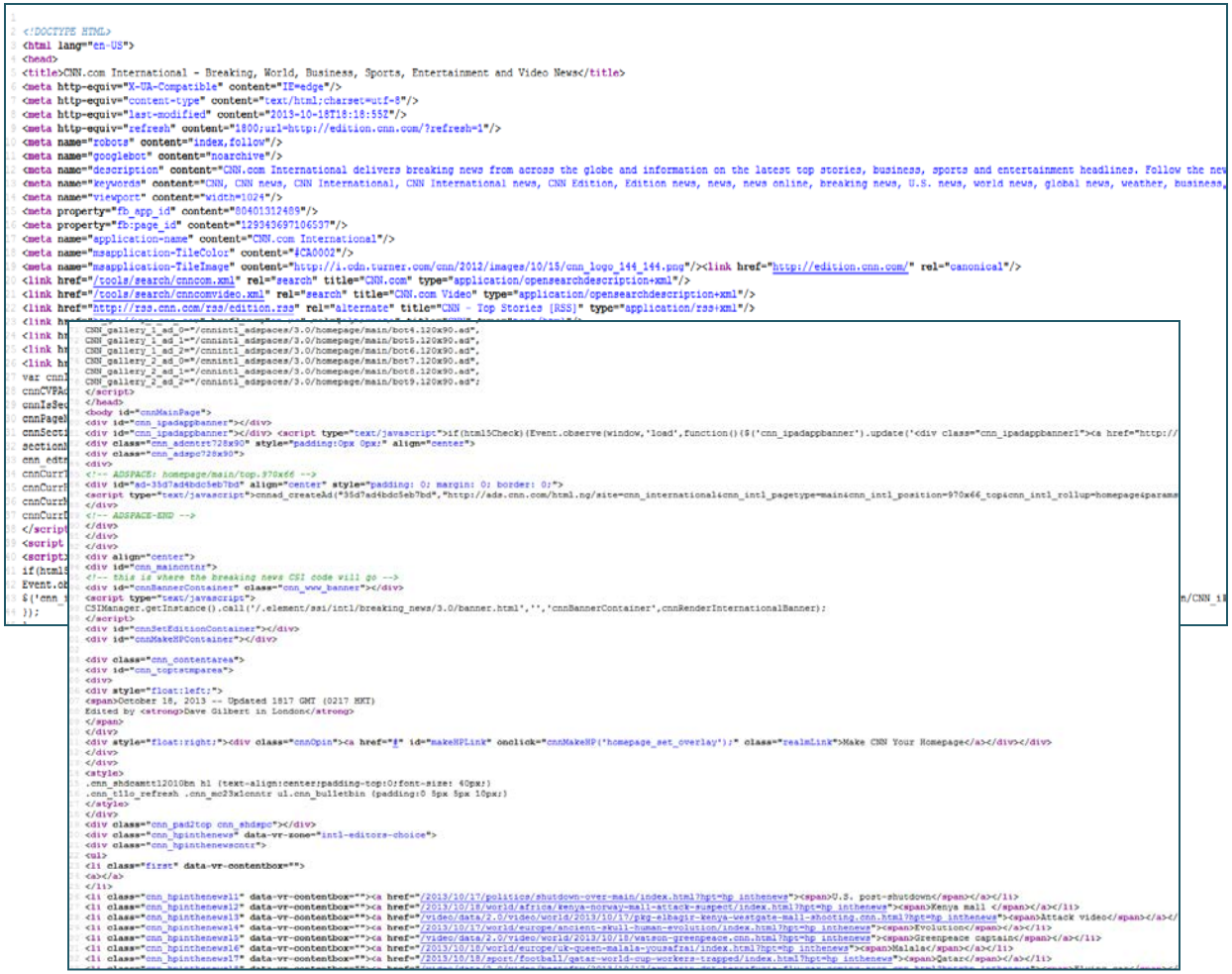
Highly Asynchronous,
Complex control flow

Looks Like This

This is what Runs

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
<title>CNN.com International - Breaking, World, Business, Sports, Entertainment and Video News</title>
<meta http-equiv="X-UA-Compatible" content="IE=edge"/>
<meta http-equiv="content-type" content="text/html;charset=utf-8"/>
<meta http-equiv="last-modified" content="2013-10-18T18:18:55Z"/>
<meta http-equiv="refresh" content="1800;url=http://edition.cnn.com/?refresh=1"/>
<meta name="robots" content="index,follow"/>
<meta name="googlebot" content="noarchive"/>
<meta name="description" content="CNN.com International delivers breaking news from across the globe and information on the latest top stories, business, sports and entertainment headlines. Follow the new
<meta name="keywords" content="CNN, CNN news, CNN International, CNN International news, CNN Edition, Edition news, news, news online, breaking news, U.S. news, world news, global news, weather, business,
<meta name="viewport" content="width=1024"/>
<meta property="fb_app_id" content="80401312489"/>
<meta property="fb:page_id" content="129943697106597"/>
<meta name="application-name" content="CNN.com International"/>
<meta name="msapplication-TileColor" content="#CA0002"/>
<meta name="msapplication-TileImage" content="http://i.cdn.turner.com/cnn/2012/images/10/15/cnn_logo_144_144.png"/><link href="http://edition.cnn.com/" rel="canonical"/>
<link href="/tools/search/cnncom.xml" rel="search" title="CNN.com" type="application/opensearchdescription+xml"/>
<link href="/tools/search/cnncomvideo.xml" rel="search" title="CNN.com Video" type="application/opensearchdescription+xml"/>
<link href="http://rss.cnn.com/rss/edition.rss" rel="alternate" title="CNN - Top Stories [RSS]" type="application/rss+xml"/>
<link href="http://www.cnn.com" hreflang="en-us" rel="alternate" title="CNN" type="text/html"/>
<link href="http://arabic.cnn.com" hreflang="ar" rel="alternate" title="CNN Arabic" type="text/html"/>
<link href="http://mexico.cnn.com" hreflang="es" rel="alternate" title="CNN Mexico" type="text/html"/>
<link href="http://z.cdn.turner.com/cnn/tmpl_asset/static/intl_homepage/1247/css/intlhplib-min.css" rel="stylesheet" type="text/css"/><script>
var cnnIsHomePage=true,
cnnCVPAdSection='edition.cnn.com_main_homepage',
cnnIsSectionPage=true,
cnnPageName="CNN International Home Page",
cnnSectionName="CNN International Home Page",
sectionName="homepage",
cnn_editnewsteherver="edition",
cnnCurrTime=new Date(1382121545000),
cnnCurrHour=14,
cnnCurrMin=39,
cnnCurrDay='Fri';
</script>
<script src="http://z.cdn.turner.com/cnn/tmpl_asset/static/intl_homepage/1247/js/intlhplib-min.js"></script>
<script>
if(html5Check){
Event.observe(window,'load',function(){
$('cnn_ipadappbanner').update('<div class="cnn_ipadappbanner1"><a href="http://itunes.apple.com/us/app/cnn-app-for-ipad/id407824176?mt=8"><img src="http://i.cdn.turner.com/cnn/.element/img/3.0/main/CNN_i
});
```

This is what Runs



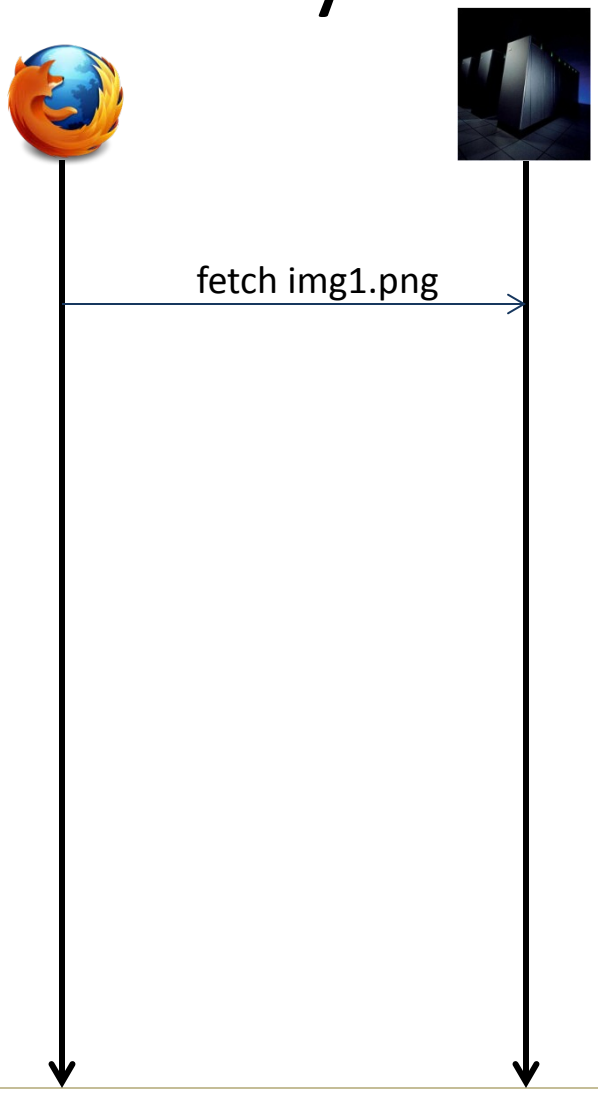
This is what Runs

Non-determinism: network latency

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

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

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

</body>
</html>
```

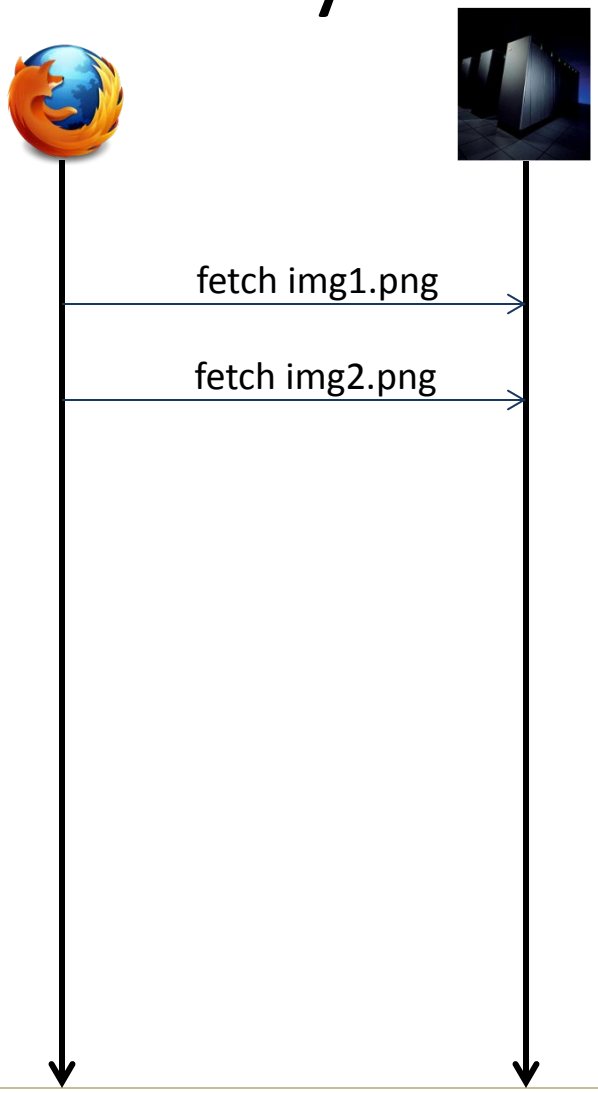
Non-determinism: network latency

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

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

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

</body>
</html>
```



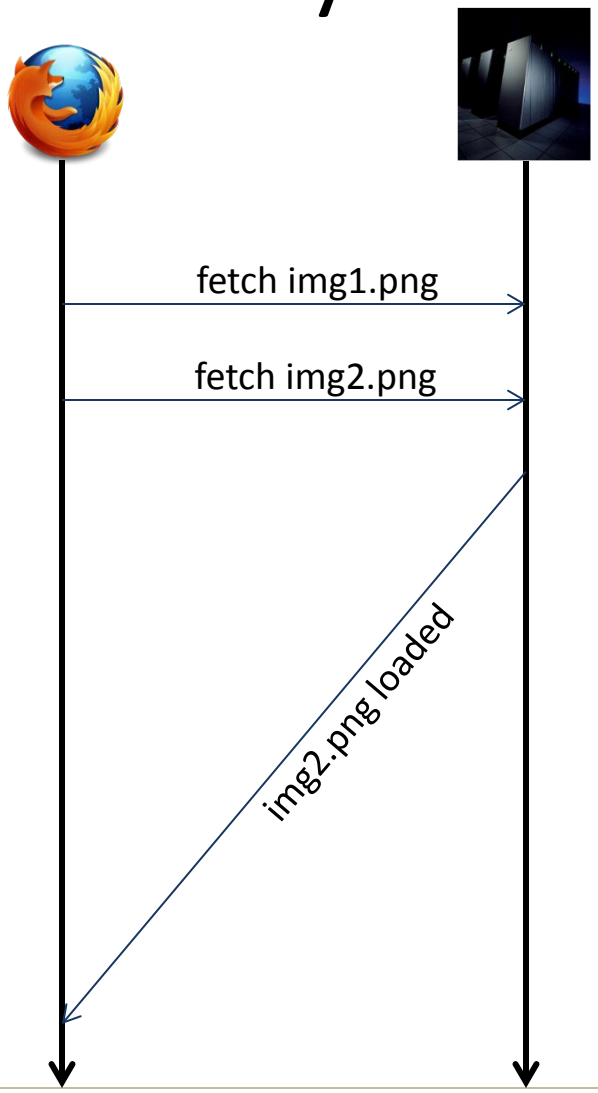
Non-determinism: network latency

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

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

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

</body>
</html>
```



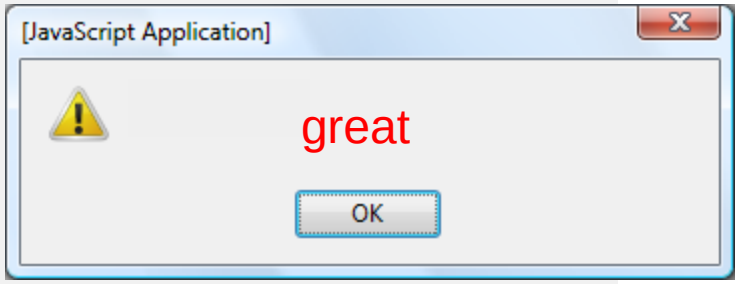
Non-determinism: network latency

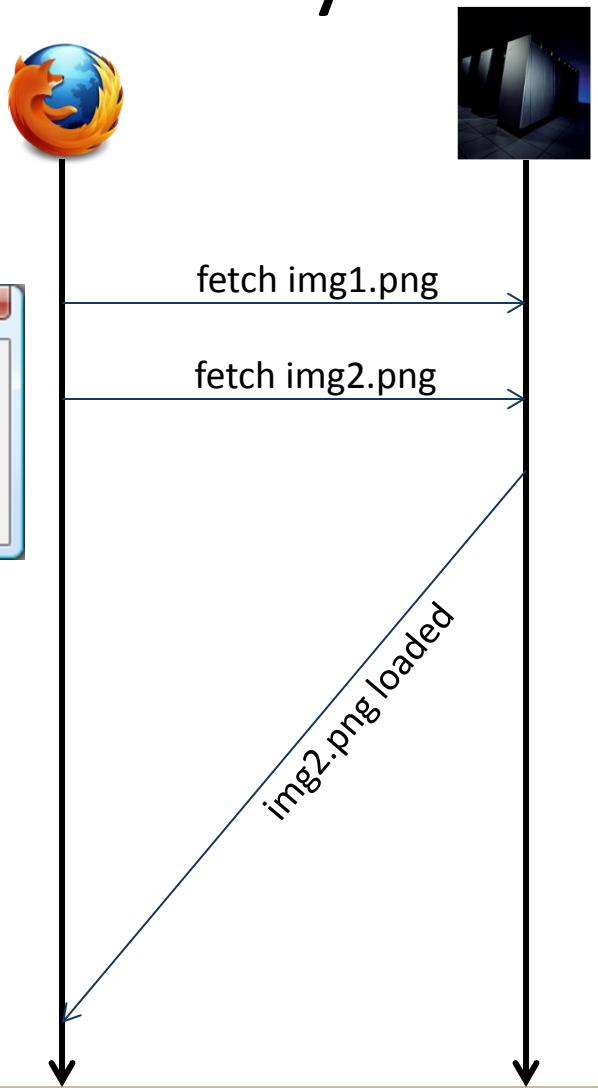
```
<html>
<head></head>
<body>

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

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

</body>
</html>
```



Non-determinism: network latency

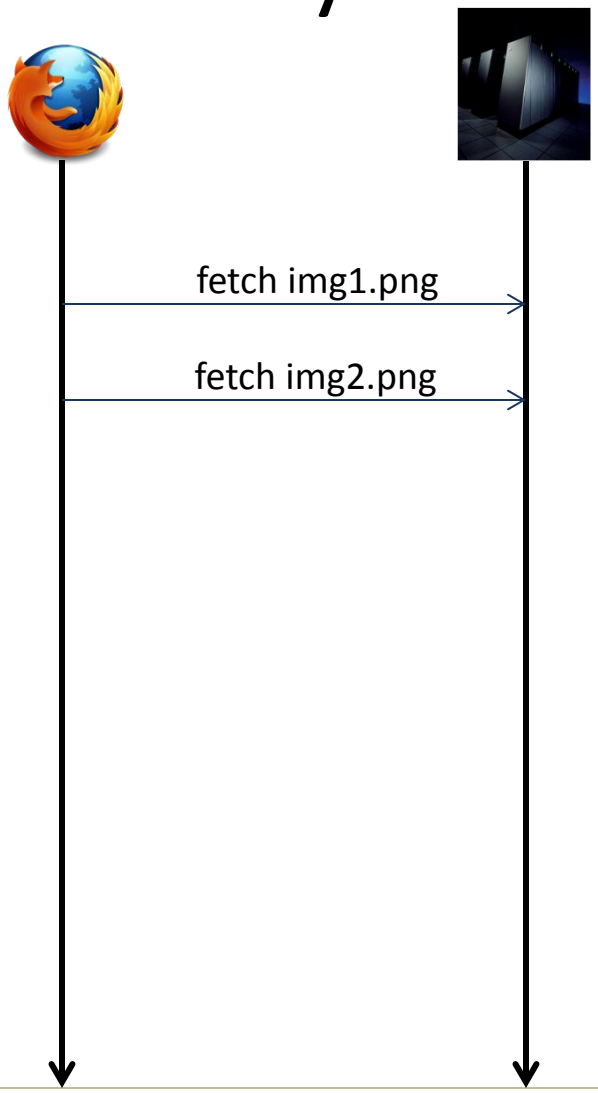
```
<html>
<head></head>
<body>

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

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

</body>
</html>
```





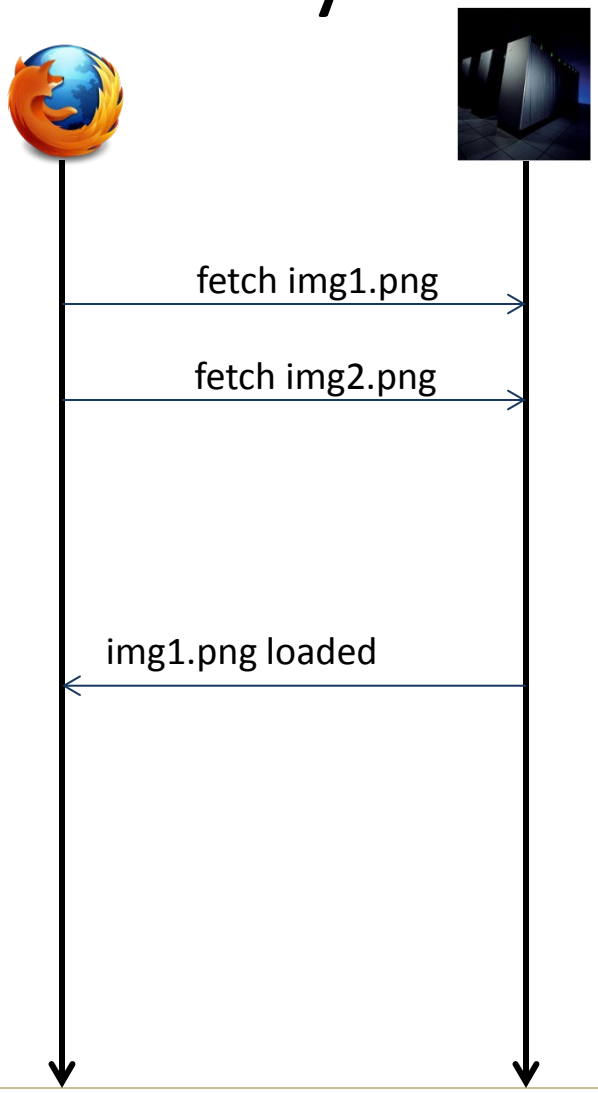
Non-determinism: network latency

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

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

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

</body>
</html>
```



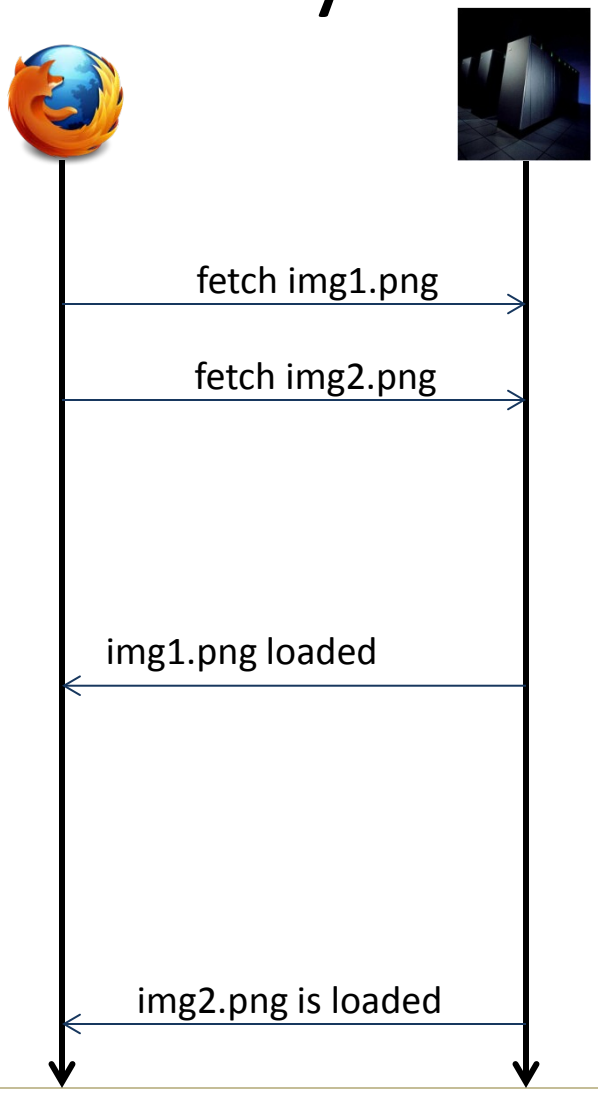
Non-determinism: network latency

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

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

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

</body>
</html>
```



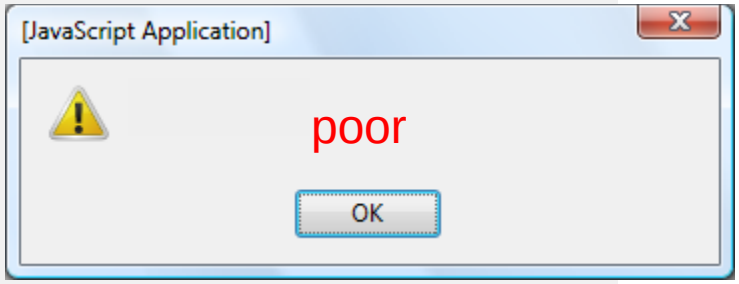
Non-determinism: network latency

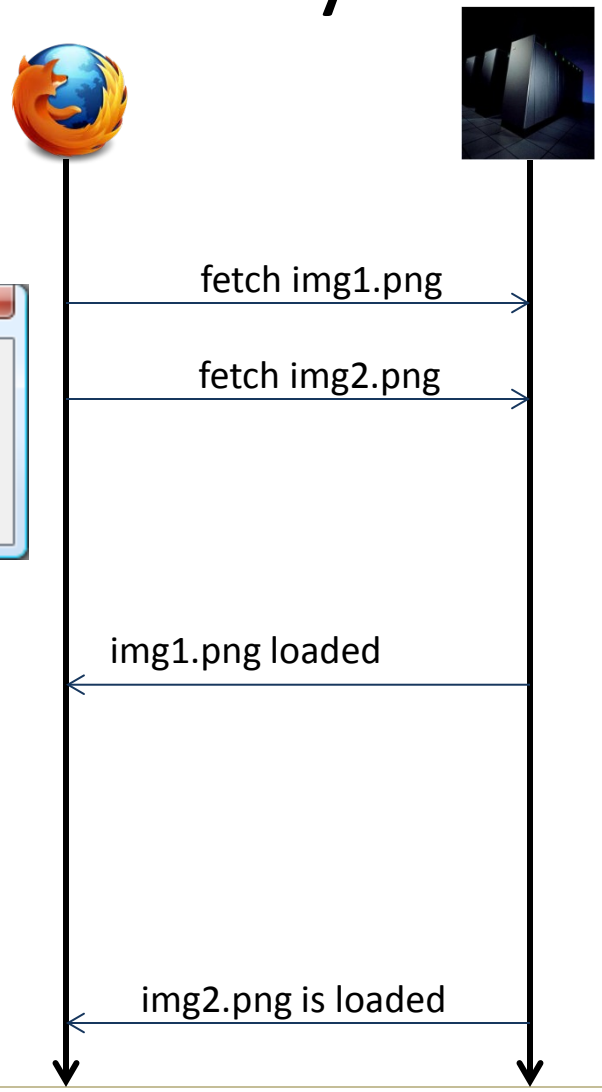
```
<html>
<head></head>
<body>

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

<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>

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

</body>
</html>
```





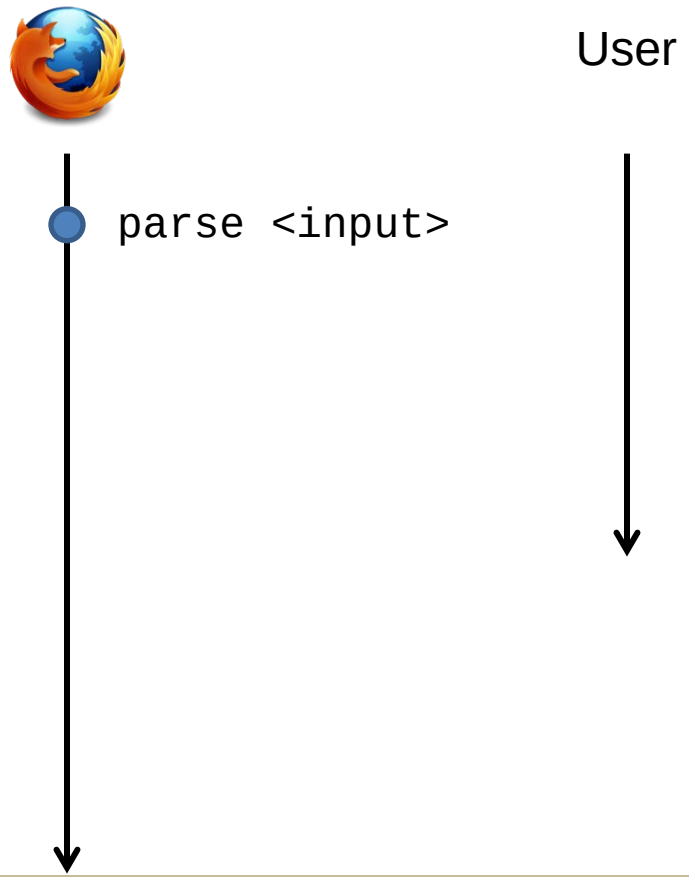
Non-determinism: user interaction

```
<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

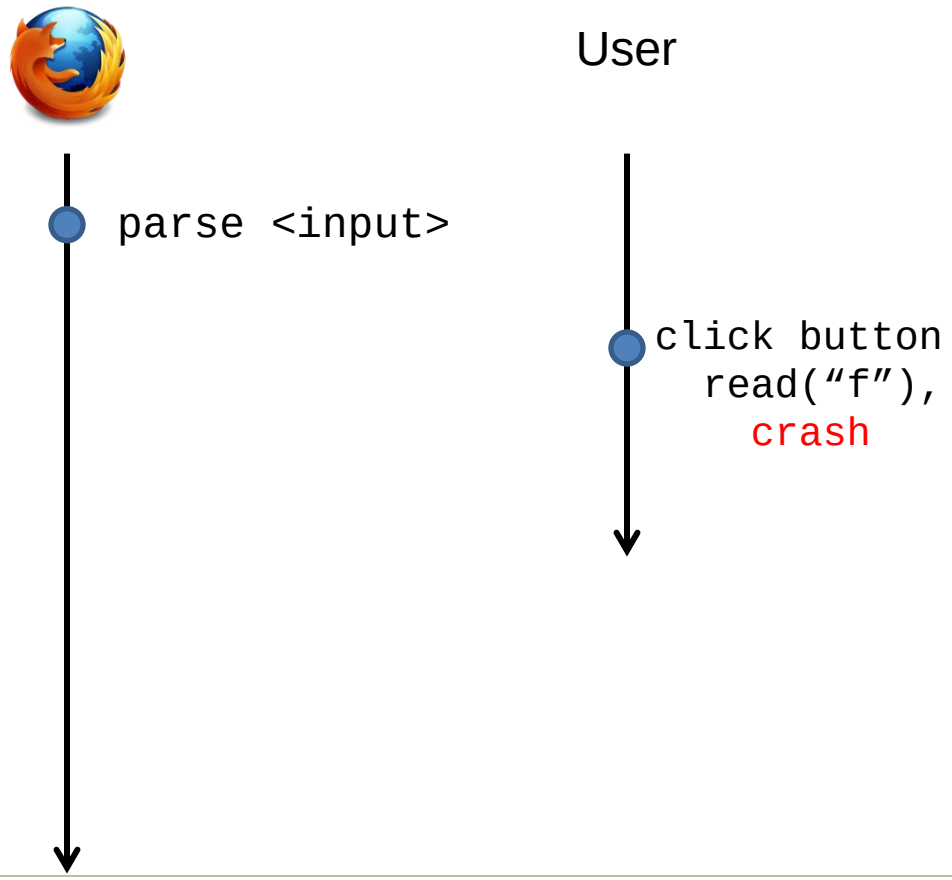
Non-determinism: user interaction

```
<html><body>

// Lots of code

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

// Lots of code

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

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



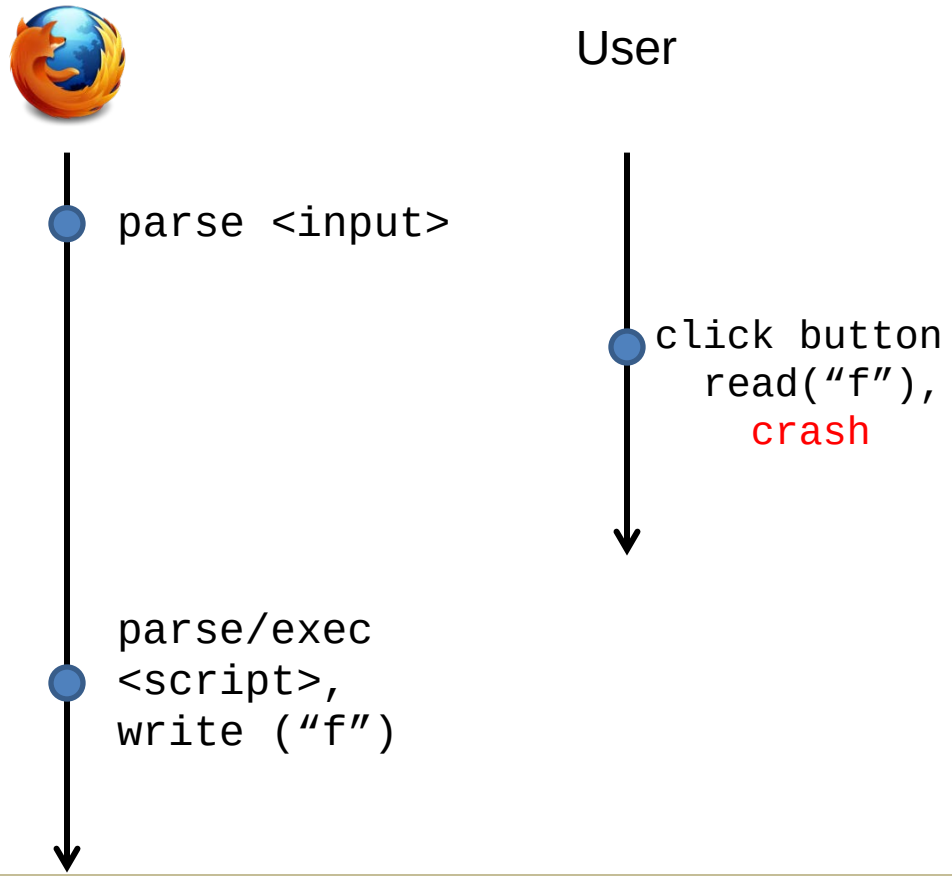
Non-determinism: user interaction

```
<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 + Shared Memory

Non-Determinism

Unwanted Behavior

What do we learn from these?

Asynchrony + Shared Memory

Non-Determinism

Unwanted Behavior

Can we phrase the problem as data race detection ?

What is a Data Race ?

What is a Data Race ?

Semantically, a data race occurs when we have a reachable program state where:

- we have two outgoing transitions by two different threads
- the two threads access the same memory location
- one of the accesses is a write

Examples

Data Race on X

Thread T_1:	Thread T_2:
fork T_2	
X = 1	X = 2

The program has a reachable state where both X = 1 and X = 2 are enabled

Program has No Data Races

Thread T_1:	Thread T_2:
X = 1	X = 2
fork T_2	

The program does not have a reachable state where both X = 1 and X = 2 are enabled

Wanted

Race Detector

race 1
race 2
race 3
….
race N

Naïve Algorithm

The definition of a data race suggests a naïve algorithm which finds all races of a program given some input states. The algorithm simply enumerates all reachable states of the concurrent program from the initial input states and checks the definition on each such reachable state.

Naïve Algorithm

The definition of a data race suggests a naïve algorithm which finds all races of a program given some input states. The algorithm simply enumerates all reachable states of the concurrent program from the initial input states and checks the definition on each such reachable state.

Does Not Scale to Real-World Programs

In Practice

In practice, algorithms aim to scale to large programs by being more efficient and not keeping program states around. To accomplish that, they weaken their guarantees.

We will see the guarantees they provide a little later, but at this point it is sufficient to mention that a typical guarantee is that the first race the algorithm repots is a real race, but any subsequent reported races after the first race are not guaranteed to exist, that is, they may be false positives, a major issue to deal with for any modern analyzer.

False positives exist because of user-defined synchronization.

Example of a False Positive Race (on variable X)

Initially: X = Y = 0

Thread T_1:	||	Thread T_2:
while(Y == 0);		X = 0
X = 1		Y = 1

A state of the art race detector may report a race on X and Y

Modern Dynamic Race Detection: 5 Steps

Step 1: Define Memory locations (on which races can happen)

Usually easy but there can be issues (framework vs. user-code)

Step 2: Define Happens-Before Model (how operations are ordered)

Can be tricky to get right due to subtleties of concurrency

Step 3: Come up with an Algorithm to detect races

Hard to get good asymptotic complexity + correctness

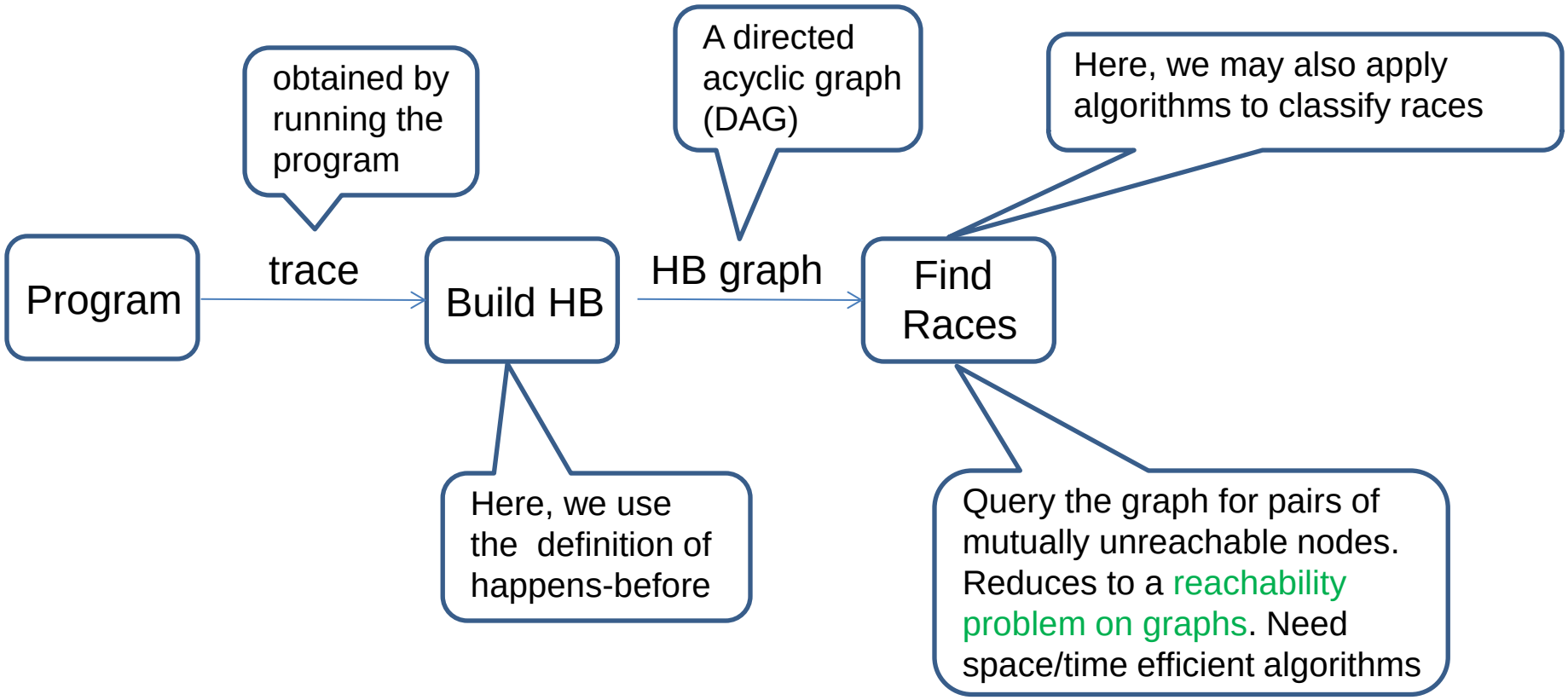
Step 4: Come up with techniques (algorithm, filters) to remove harmless races

Needs to answer what harmless means

Step 5: Implement Algorithm and Evaluate

Important to have low instrumentation overhead

Dynamic Race Detection: Flow



(some of these boxes will become clear later in the slides)

Let us now discuss these 5 steps in our example domain: event-driven applications

These 5 steps need to be taken for any other domain

Difficulties…

Requires going over the HTML5 specification…

…and experimenting with browsers…



Step 1:

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

Memory Locations: Example

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

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

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

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

</body>
</html>
```

Step 2:

… is a partial order $(A, \preccurlyeq)$

Step 2:

… is a partial order ($A, \preccurlyeq$)

First, define the contents of A, i.e. atomic action

- E.g.: parsing a single HTML element, executing a script, processing an event handler

Step 2:

… is a partial order ($A, \preccurlyeq$)

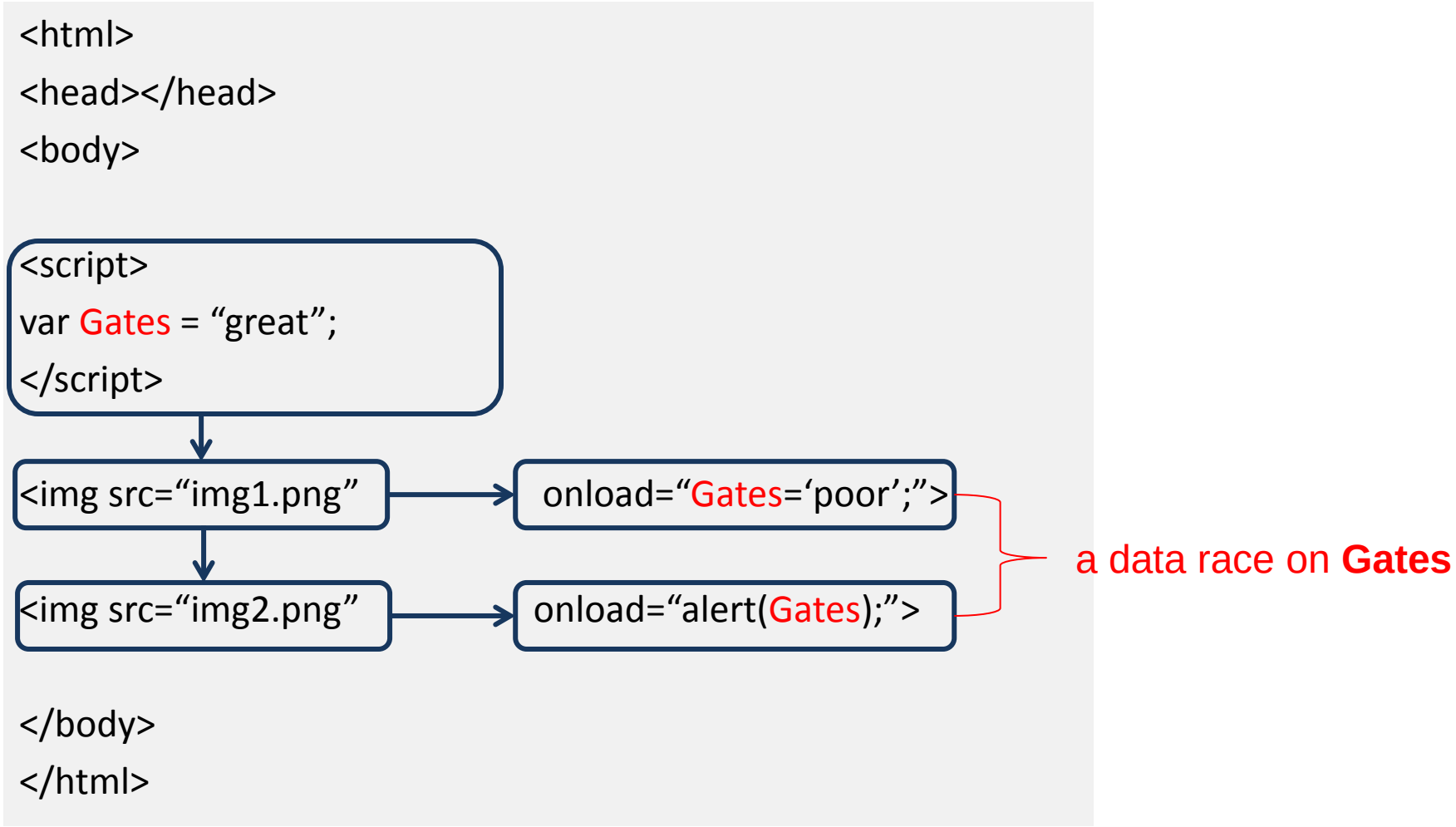
First, define the contents of A, i.e. atomic action

- E.g.: parsing a single HTML element, executing a script, processing an event handler

Then, define $\preccurlyeq$, i.e. how to order actions

- E.g.: parsing of HTML elements of the web page is ordered

Happens-Before: Example



Steps 3 and 4 : Define Race Detection Algorithm

Happens-Before Model

Memory Locations

?

?

Dynamic Race Detection: Theorems (that an analyzer should ensure)

No false negatives: if the Analysis reports no races on an execution, then the execution must not contain a race

No false positives: if the Analysis reports a race for a given execution then the execution for sure contains a race

Two Challenges Affecting Steps 3 and 4

Synchronization done with read/writes

quickly leads to thousands of false races

Massive number of event handlers

quickly causes space blow-up in analysis data structures

False Positives: 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 on variables:
  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 report races on variable

init

But not on:

y

y.g

Because races on **y** **and** **y.g** are covered by the race on init

Two Challenges Affecting Steps 3 and 4

Synchronization with read/writes

race coverage eliminates false races

Massive number of event handlers

quickly causes space blow-up in analysis data structures

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 $\preccurlyeq$ (the happens-before of an execution trace) as a directed acyclic graph and perform graph connectivity queries to answer $a \preccurlyeq b$

Report a race if a and b are not reachable from one another, they teach the same memory location and one is a write.

Example $\preccurlyeq$ built from a trace

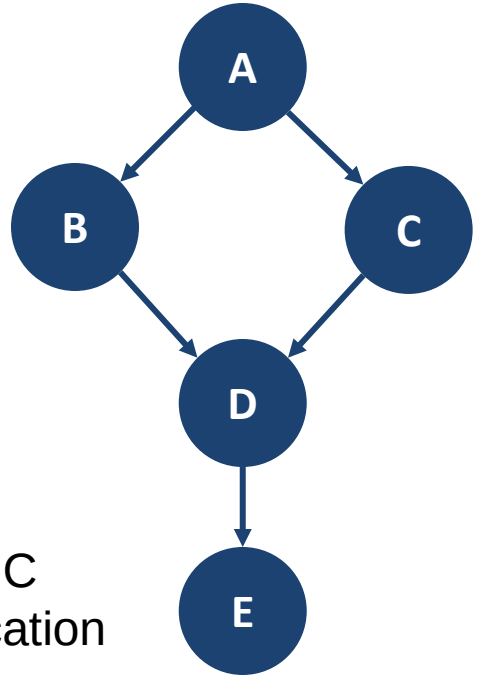
Lets take the trace: ABCDE.
If the happens-before tells us that B and C need not be ordered, but all others are ordered, then we obtain the following graph on the right, also written in text as:

$\preccurlyeq$ = { (A, B), (A, C), (B, D), (C, D), (D, E) }

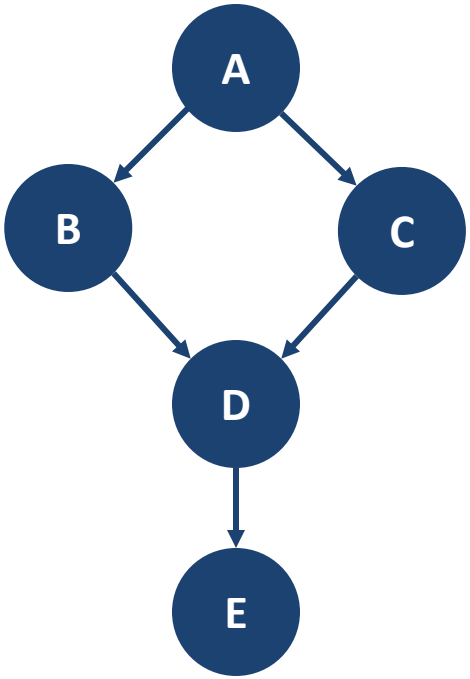
This graph captures that we not only have ABCDE as a trace but we also have ACBDE as a trace

In this example, we would have a race between B and C if actions B and C were touching the same memory location and one of them was writing to that location.

The DAG representing $\preccurlyeq$ (Hasse diagram)



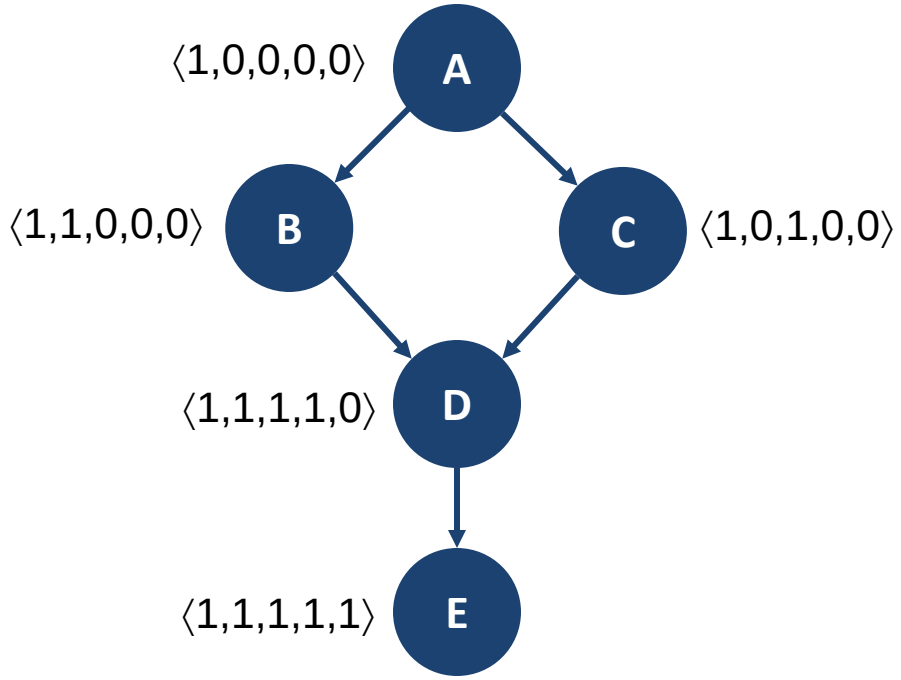
$a \overset{?}{\preccurlyeq} b$ via BFS



M - number of edges
N - number of nodes

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

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



A vector clock vc is a map:

$$vc \in Nodes \rightarrow Nat$$

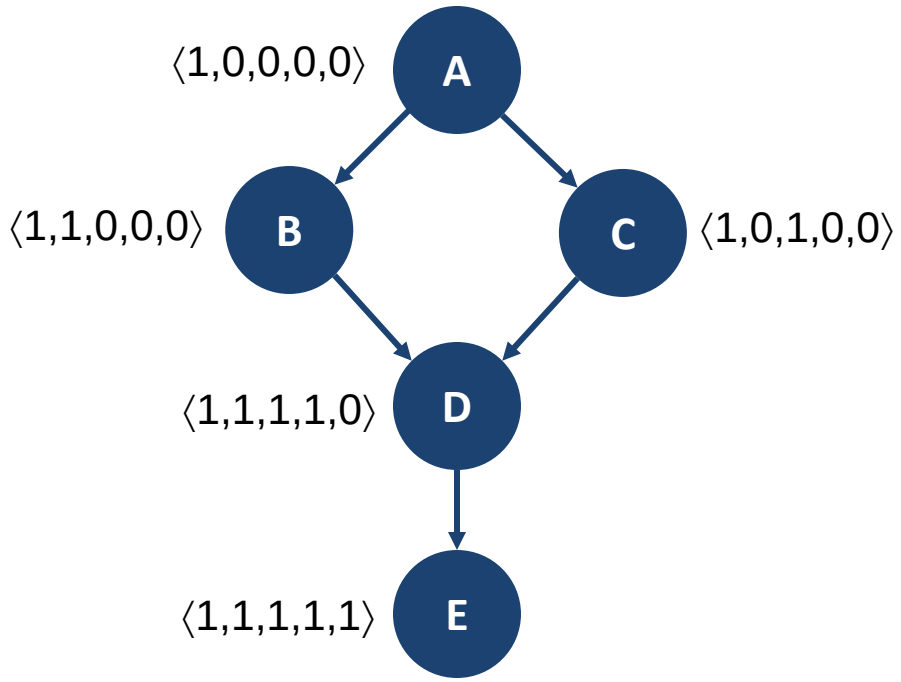
associate a vector clock with each node

$\langle 1,0,0,0,0 \rangle \sqsubseteq \langle 1,1,1,1,0 \rangle$
it follows that $A \preccurlyeq D$

$\langle 1,1,0,0,0 \rangle \not\sqsubseteq \langle 1,0,1,0,0 \rangle$
it follows that $B \not\preccurlyeq C$

In this example graph, Nodes = {A,B,C,D,E}

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



At a given node, its vector clock captures who can reach that node.

For example, for node C, its vector clock vc-C⟨1,0,1,0,0⟩ denotes that:

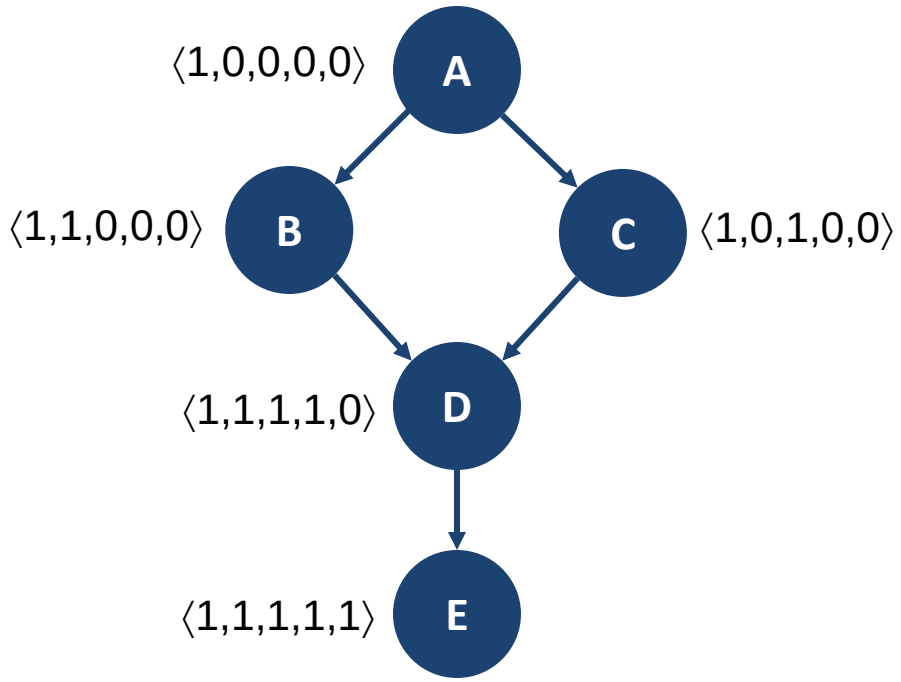
A can reach C: because vc-C(A) = 1
B cannot reach C: because vc-C (B) = 0
C can reach C: because vc-C (C) = 1
D cannot reach C: because vc-C (D) = 0
E cannot reach C: because vc-C (E) = 0

Given two nodes, say B and C, we can determine whether they are mutually unreachable by just checking:

whether vc-C(B) = 0 and vc-B(C) = 0

This is constant-time work.

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



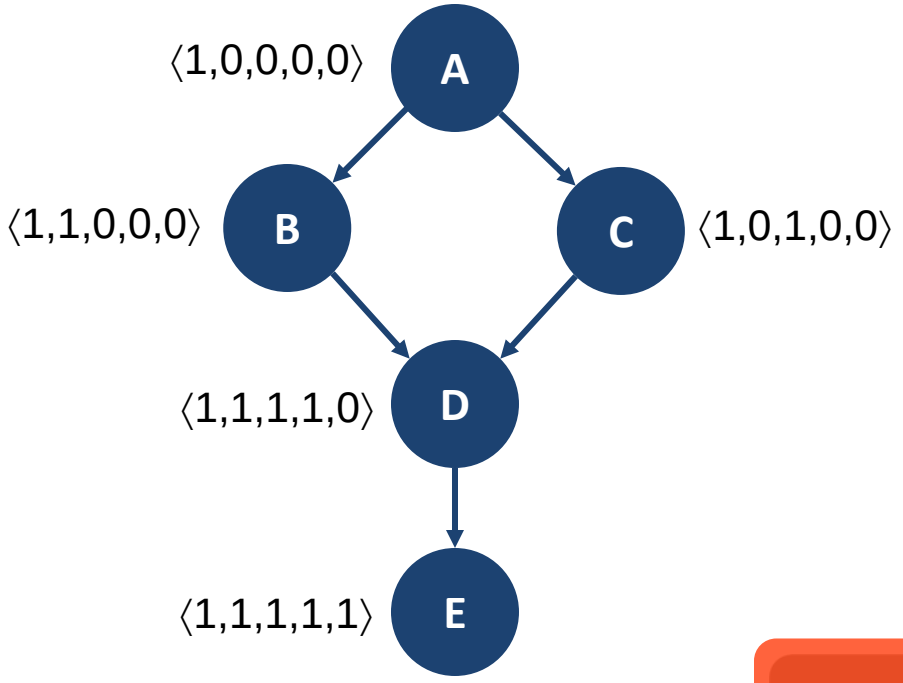
To compute the vector clocks, simply process each edge of the graph and join the vector clocks.

For instance, to compute the vector clock for node D, we may first process the edge from to B -> D, thereby copying the vector clock ⟨1,1,0,0,0⟩ from B to D.

Then, when we process the edge C -> D, we will join (take the max) of the current vector clock at D (⟨1,1,0,0,0⟩) and the vector clock coming from C (⟨1,0,1,0,0⟩).

That is, for each edge we process, we do O(N) work (as we need to iterate over each entry in the vector clock and the number of such entries is N).

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



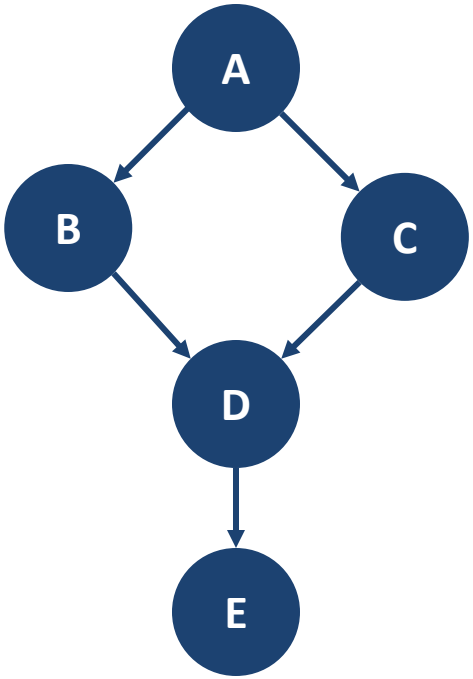
Pre-computation Time: $O(M \cdot N)$
(to obtain all vector clocks)

Query Time: $O(1)$
(for a pair of nodes)

Space: $O(N^2)$

Space Explosion

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



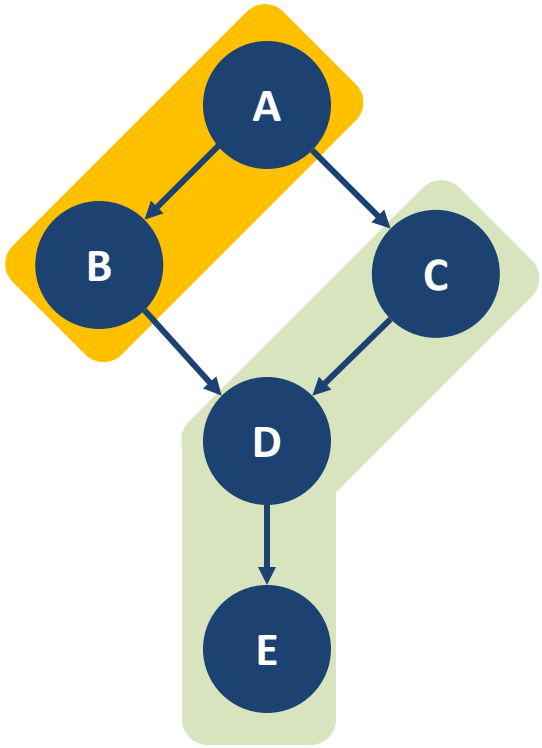
Key idea: Re-discover threads by partitioning the nodes into chains.

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



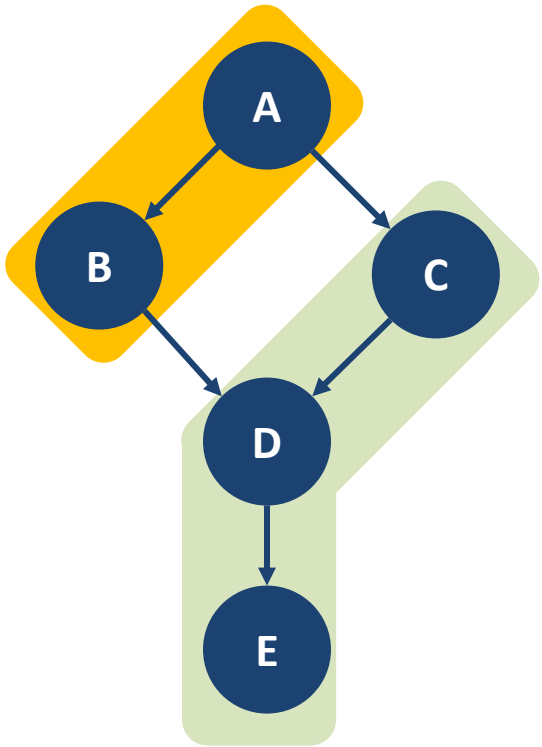
Key idea: Re-discover threads by partitioning the nodes into chains.

computes a map:

$c \in$ Nodes $\rightarrow$ ChainIDs

associate a chain with each node

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



C = number of chains

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

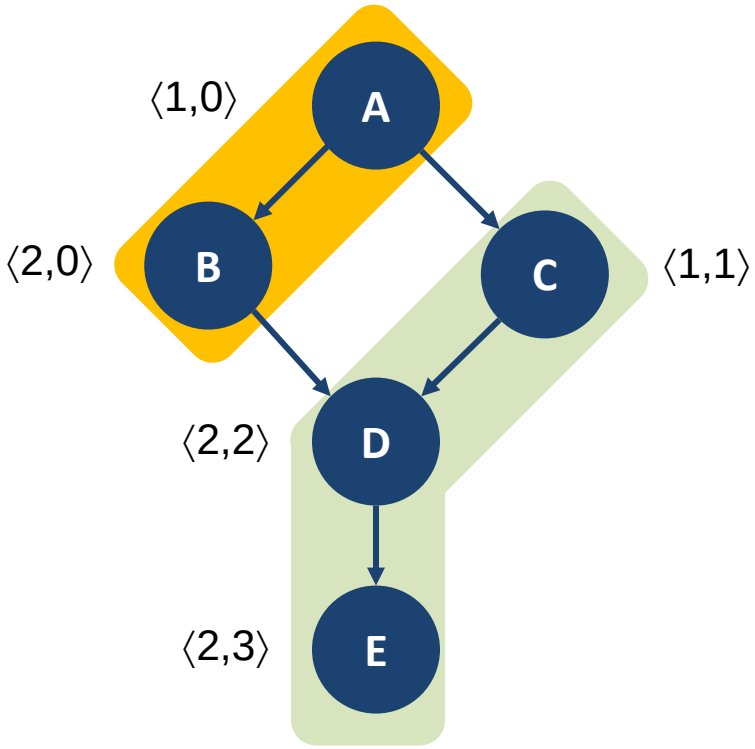
Vector clock computation: $O(C \cdot M)$

Query Time: $O(1)$

Space: $O(C \cdot N)$

Improved

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



C = number of chains

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

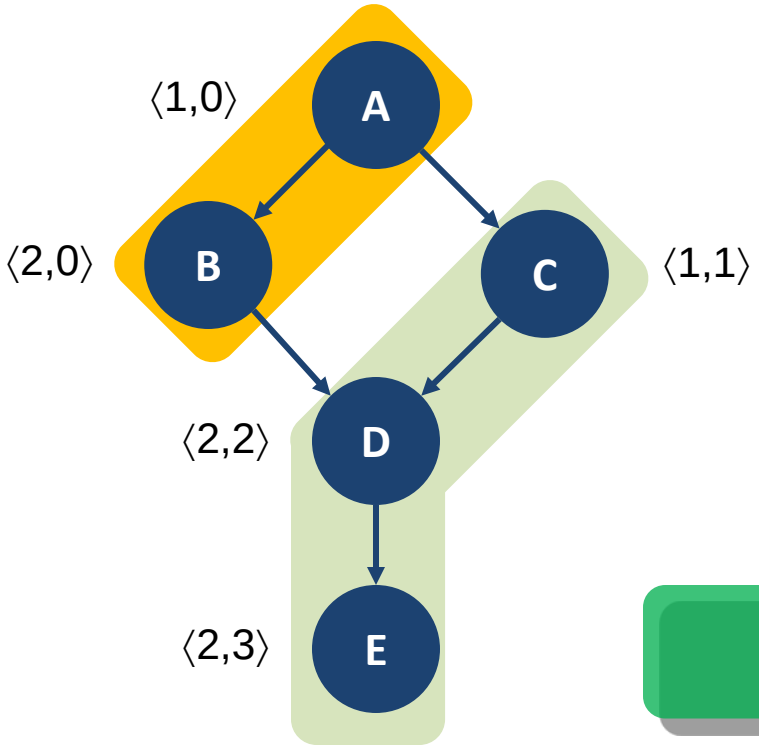
Vector clock computation: $O(C \cdot M)$

Query Time: $O(1)$

Space: $O(C \cdot N)$

Improved

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



Improved

C = number of chains

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

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

Query Time: $O(1)$

Space: $O(C \cdot N)$

Improved

Two Challenges Affecting Steps 3 and 4

Synchronization with read/writes

race coverage eliminates all false races

Massive number of event handlers

greedy chain decomposition + vector clocks

space: $O(C \cdot N)$ where $C << N$

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: $O(C \cdot N)$

Step 5: Implement and Evaluate

- Based on WebKit Browser
  - Used by Apple's Safari and Google's Chrome

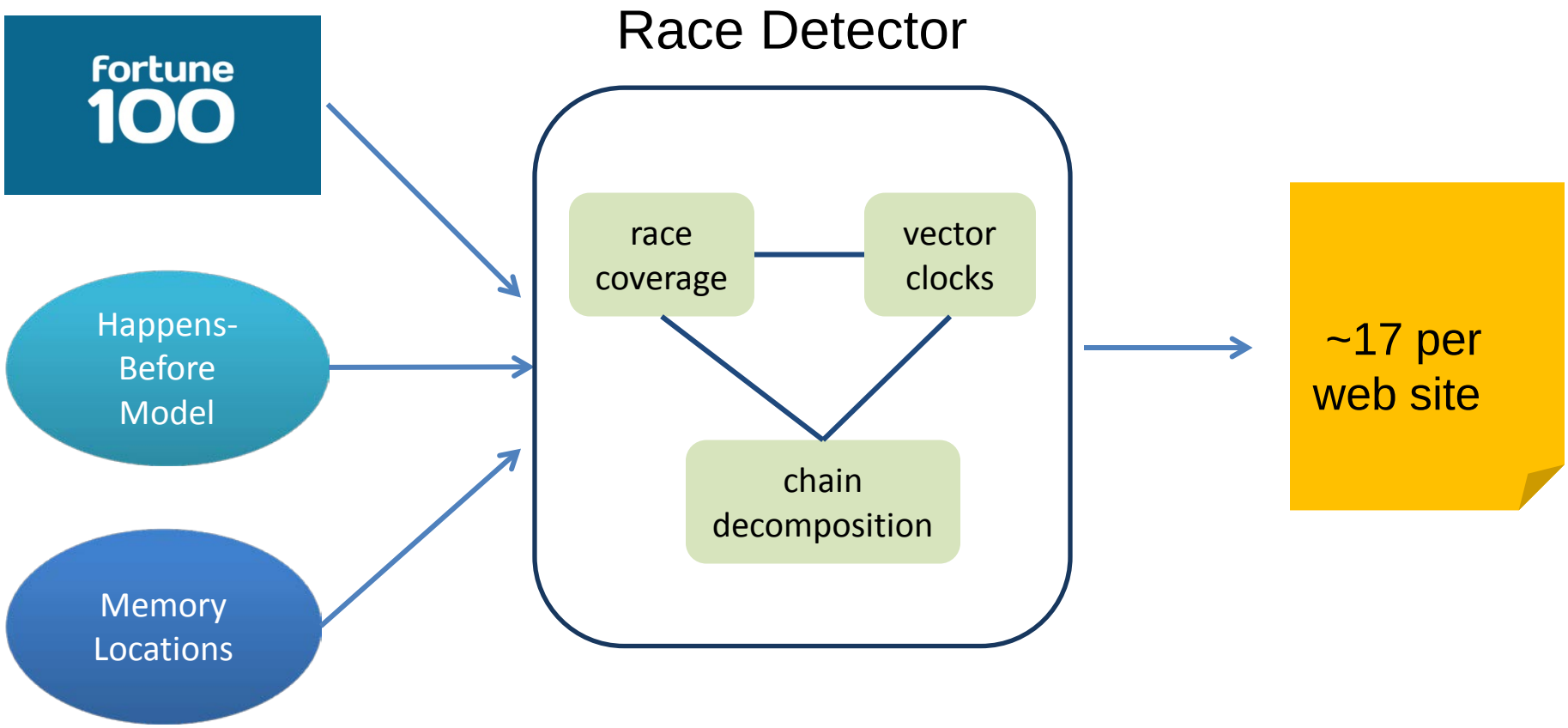
- Quite robust, Demo:
  - http://www.eventracer.org

Step 5: Implement and Evaluate

We evaluate algorithm performance and precision

Hopefully algorithm is fast and does not report too many false positives on a wide range of applications

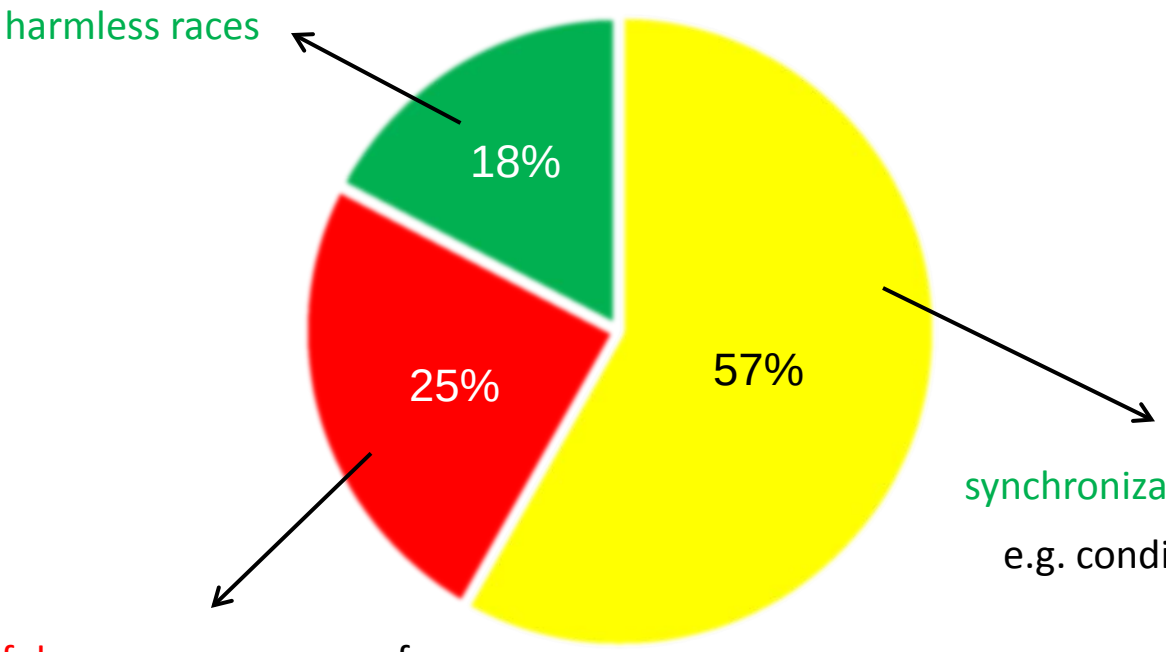
Experiments: Fortune 100 web sites



Race coverage: benefit

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

314 uncovered races: manual inspection



synchronization races: many idoms

e.g. conditionals, try-catch, …

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, …

Algorithm: 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

Algorithm: Time

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

Modern Dynamic Race Detection: 5 Steps

Step 1: Define Memory locations (on which races can happen)

Usually easy but there can be issues (framework vs. user-code)

Step 2: Define Happens-Before Model (how operations are ordered)

Can be tricky to get right due to subtleties of concurrency

Step 3: Come up with an Algorithm to detect races

Hard to get good asymptotic complexity + correctness

Step 4: Come up with techniques (algorithm, filters) to remove harmless races

Needs to answer what harmless means

Step 5: Implement Algorithm and Evaluate

Important to have low instrumentation overhead

Check it Out

Web: http://www.eventracer.org

Android: http://www.eventracer.org/android

All Open Source: https://github.com/eth-srl/