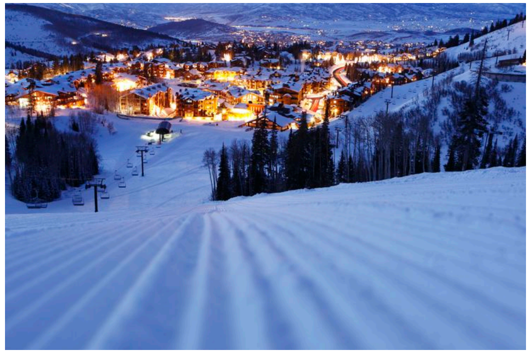
SWORD: A Bounded Memory-Overhead Detector of OpenMP Data Races in Production Runs

Simone Atzeni, Ganesh Gopalakrishnan, Zvonimir Rakamaric
School of Computing, University of Utah, **Salt Lake City**, UT 84112

Courtesy
Pinterest



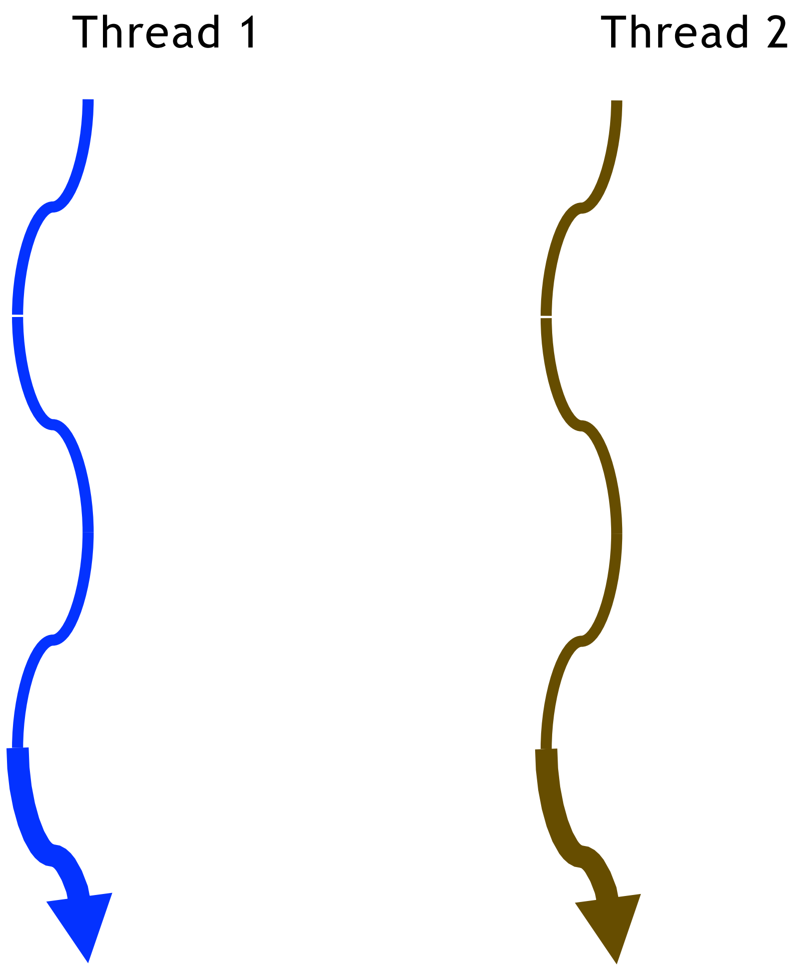
**Presented at
IPDPS 2018
See paper for details**

Ignacio Laguna, Greg L. Lee, Dong H. Ahn
Lawrence Livermore National Laboratory, Livermore, CA

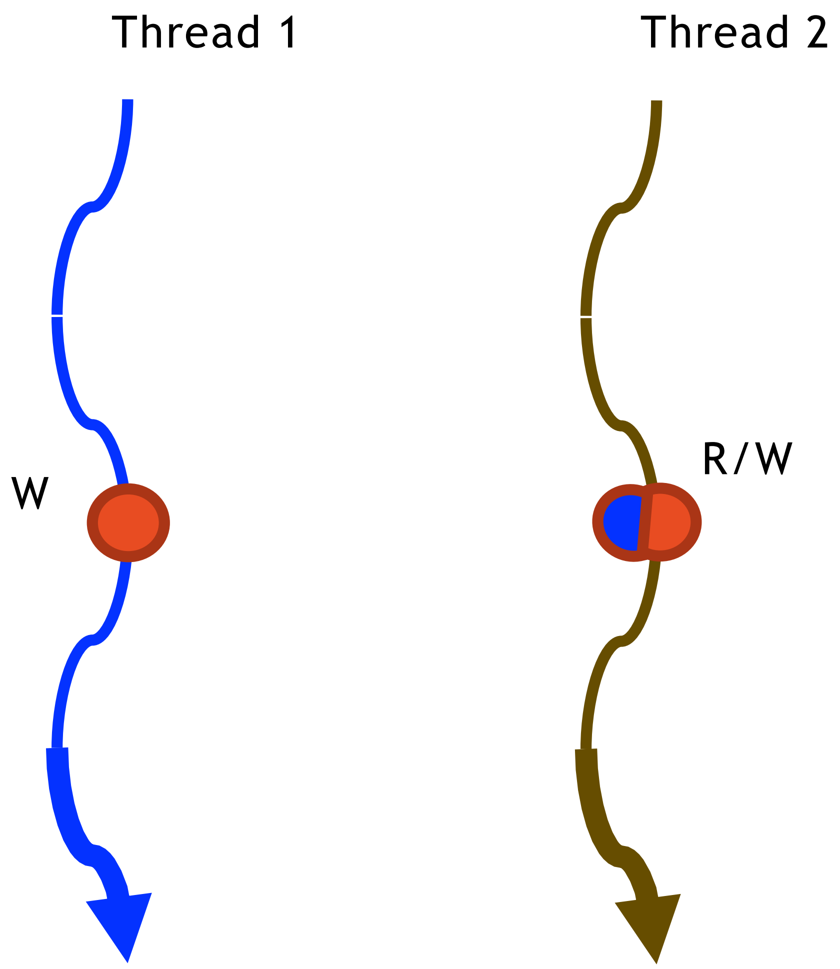
Github.com / PRUNERS

What is a data race?

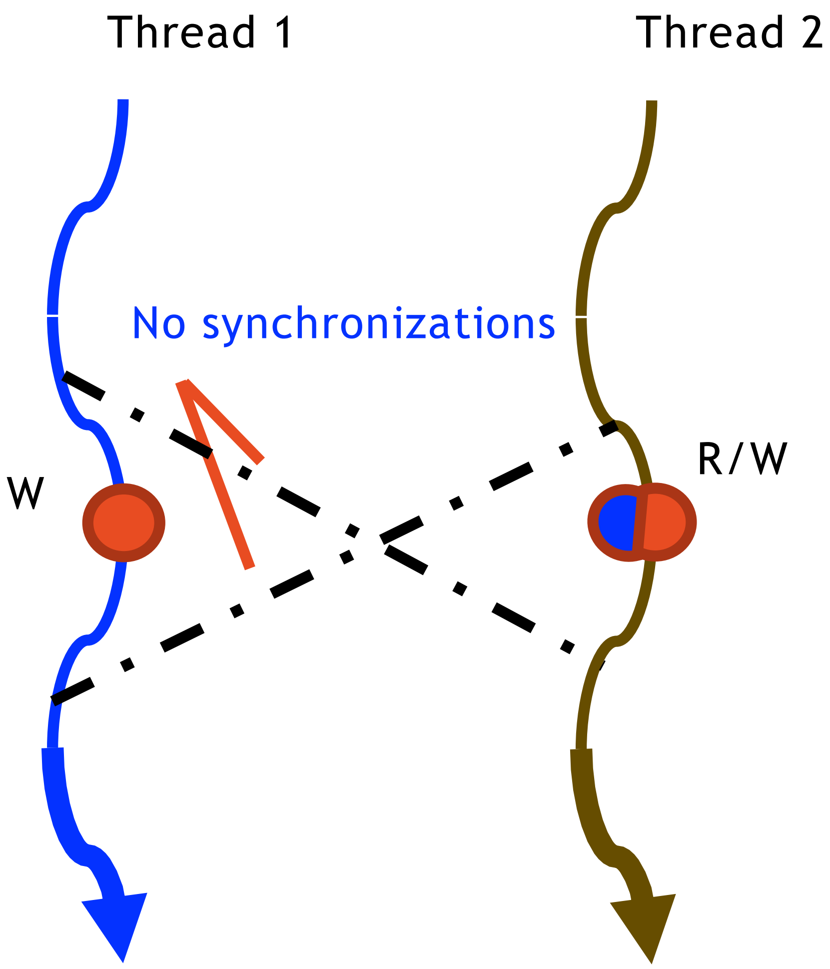
What is a data race?



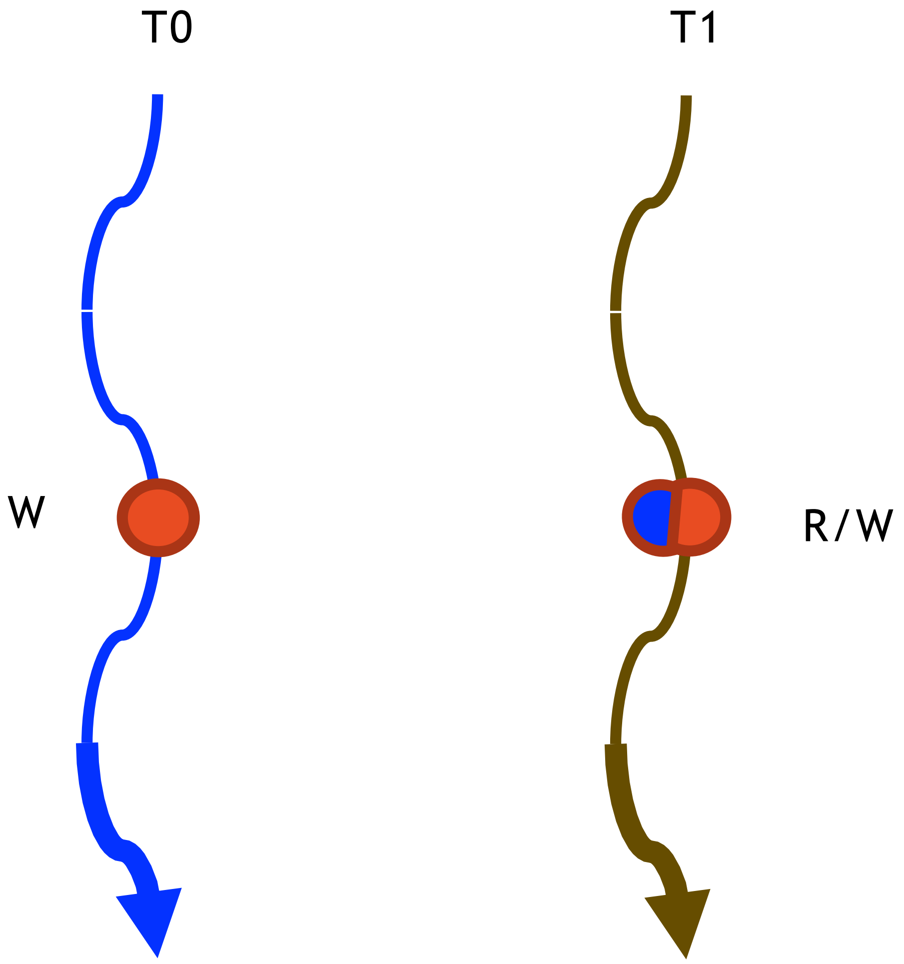
What is a data race?



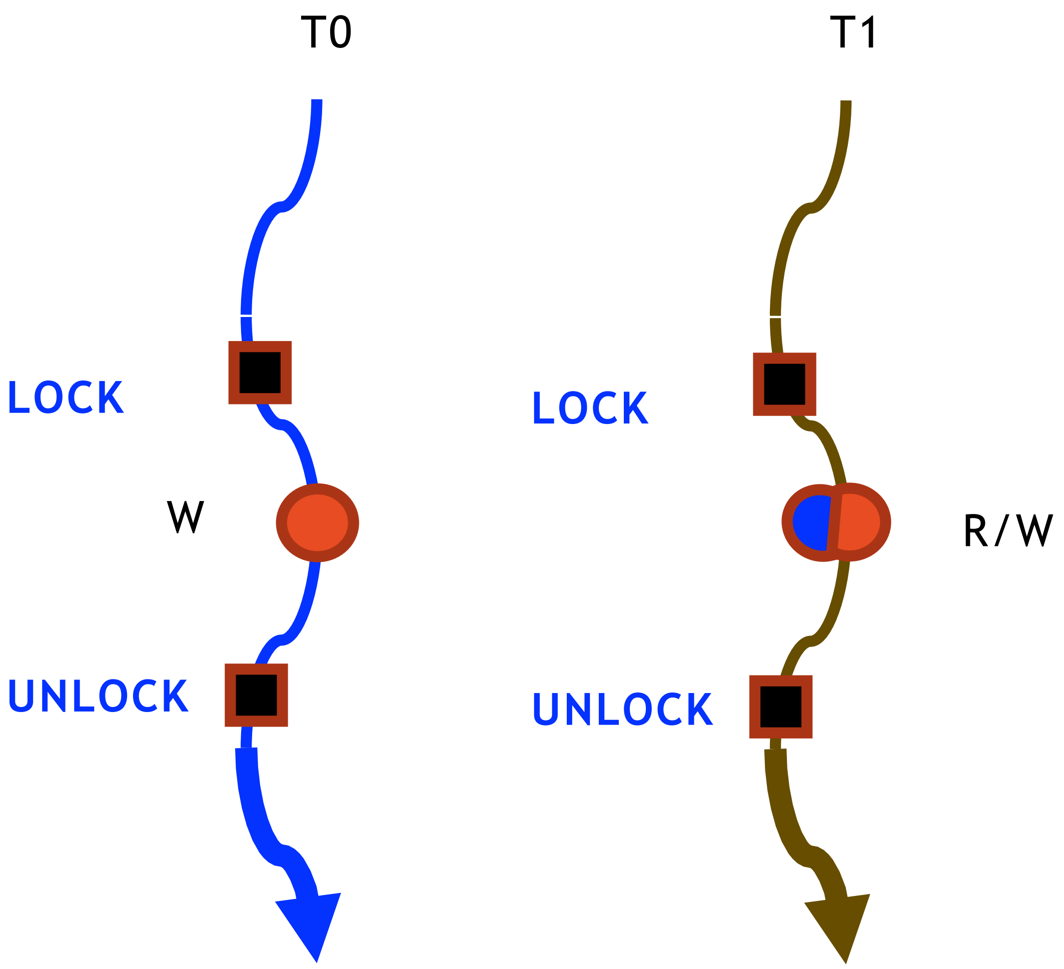
What is a data race?



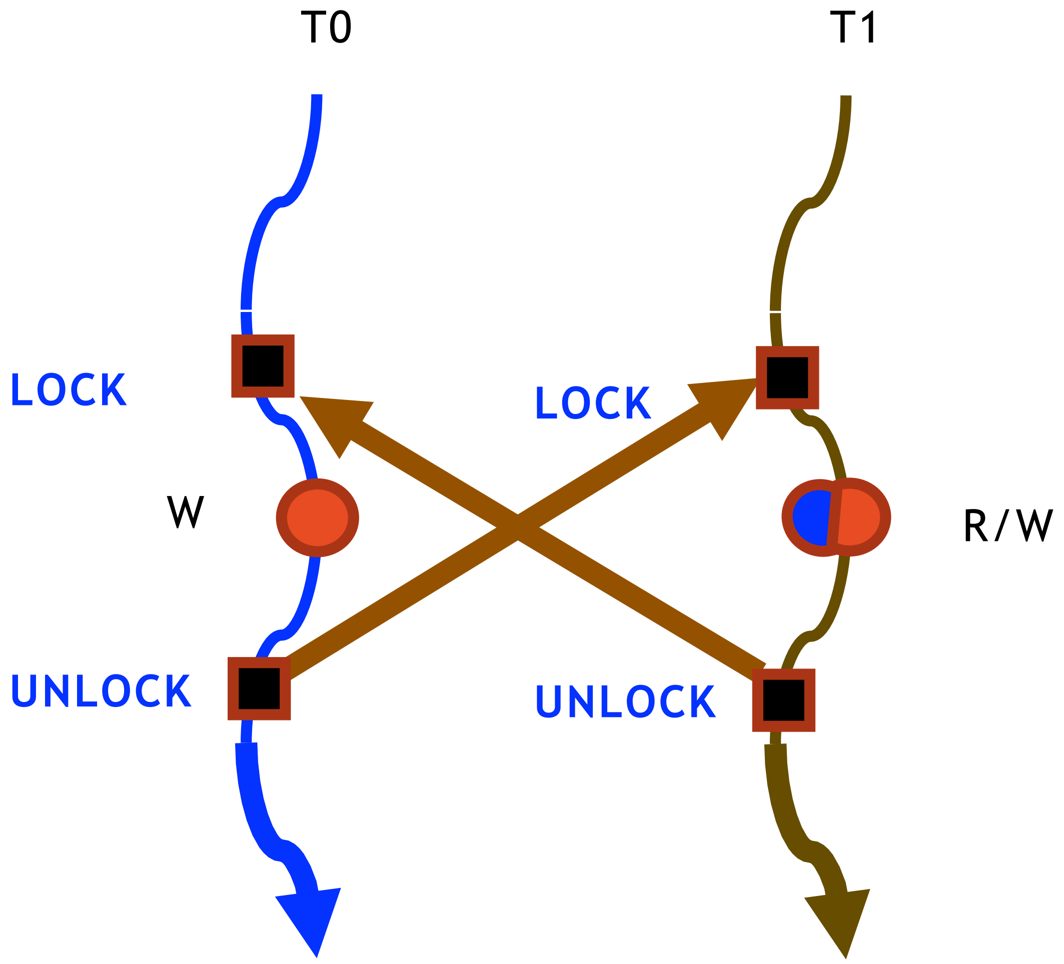
One way to eliminate this race



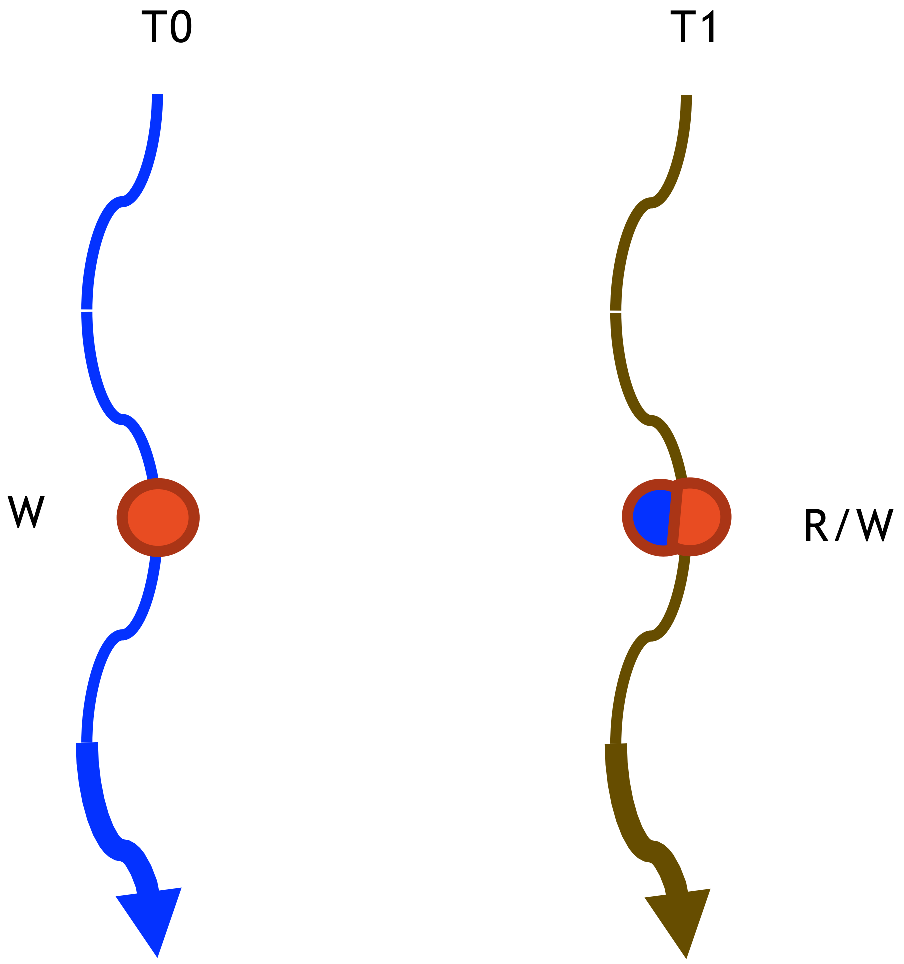
One way to eliminate this race



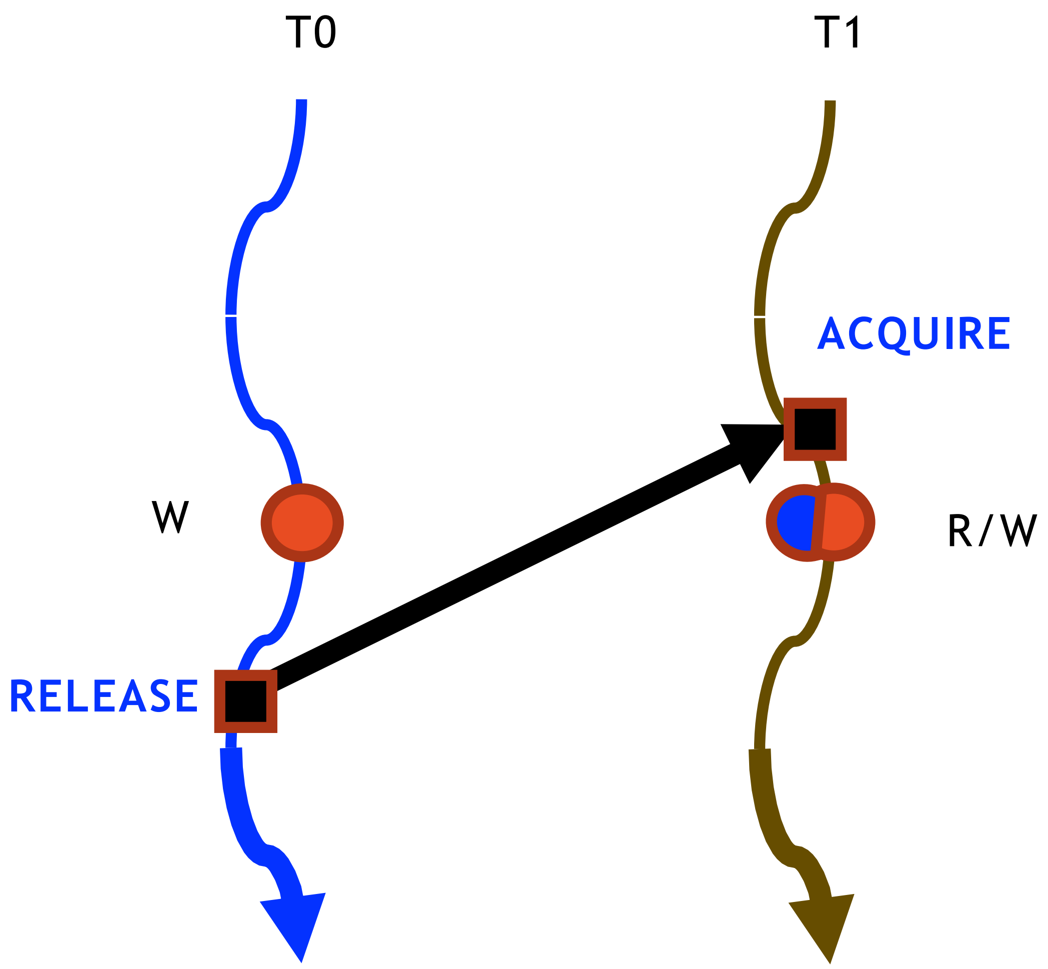
One way to eliminate this race



Another way to eliminate this race



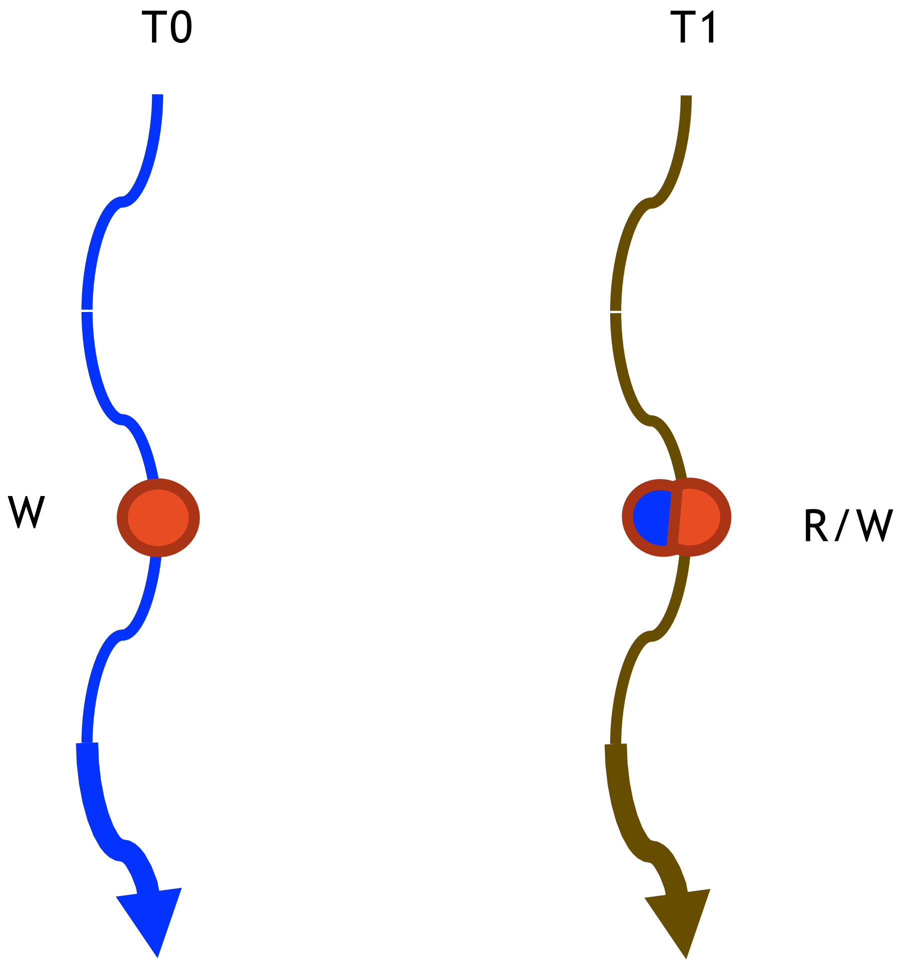
Another way to eliminate this race



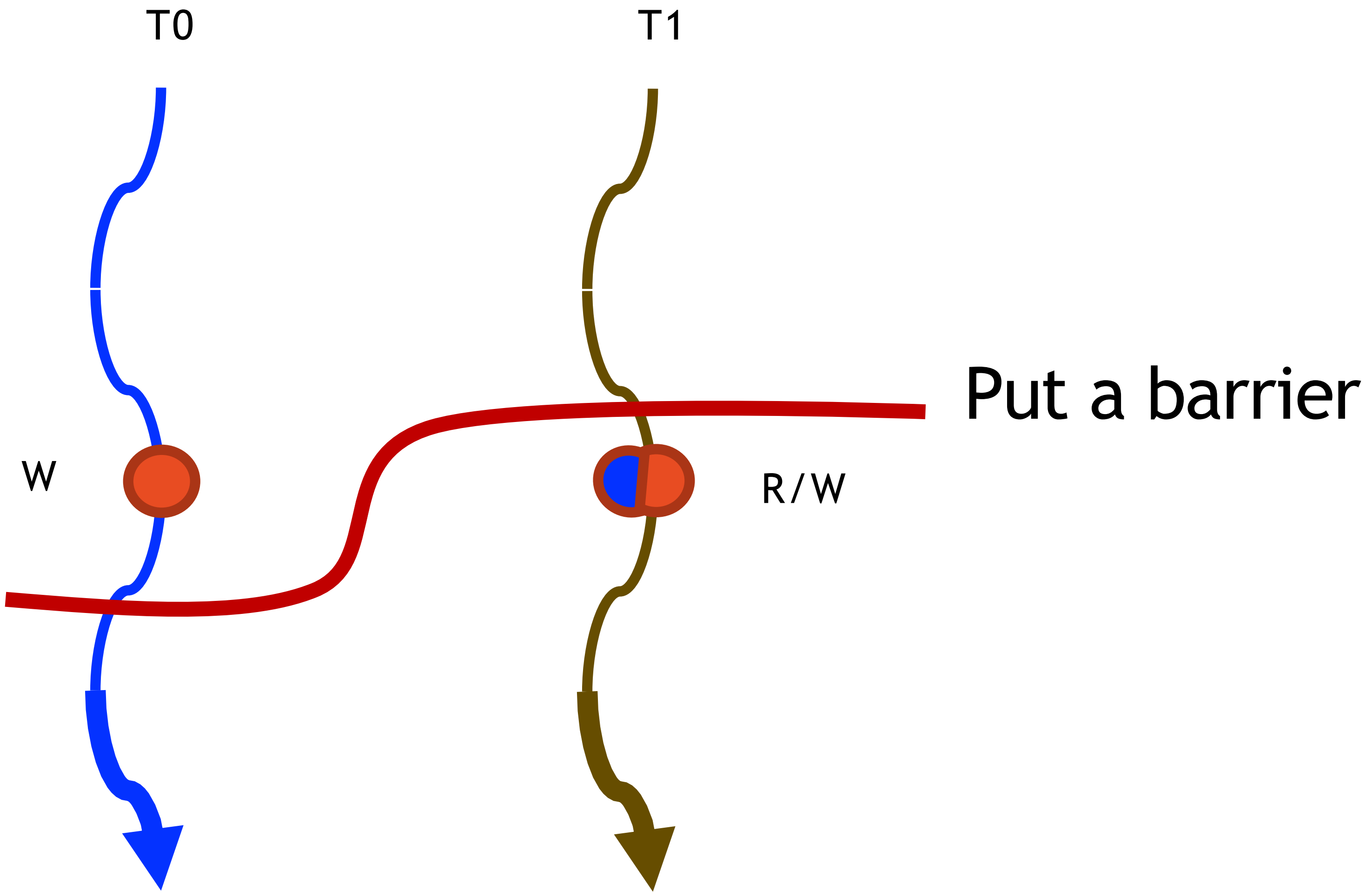
Signal using `special’ variables

- Java ‘volatile’ annotations
  - NOT C ‘volatiles’ ☹
- C++11 ’atomic’ annotations

A third way

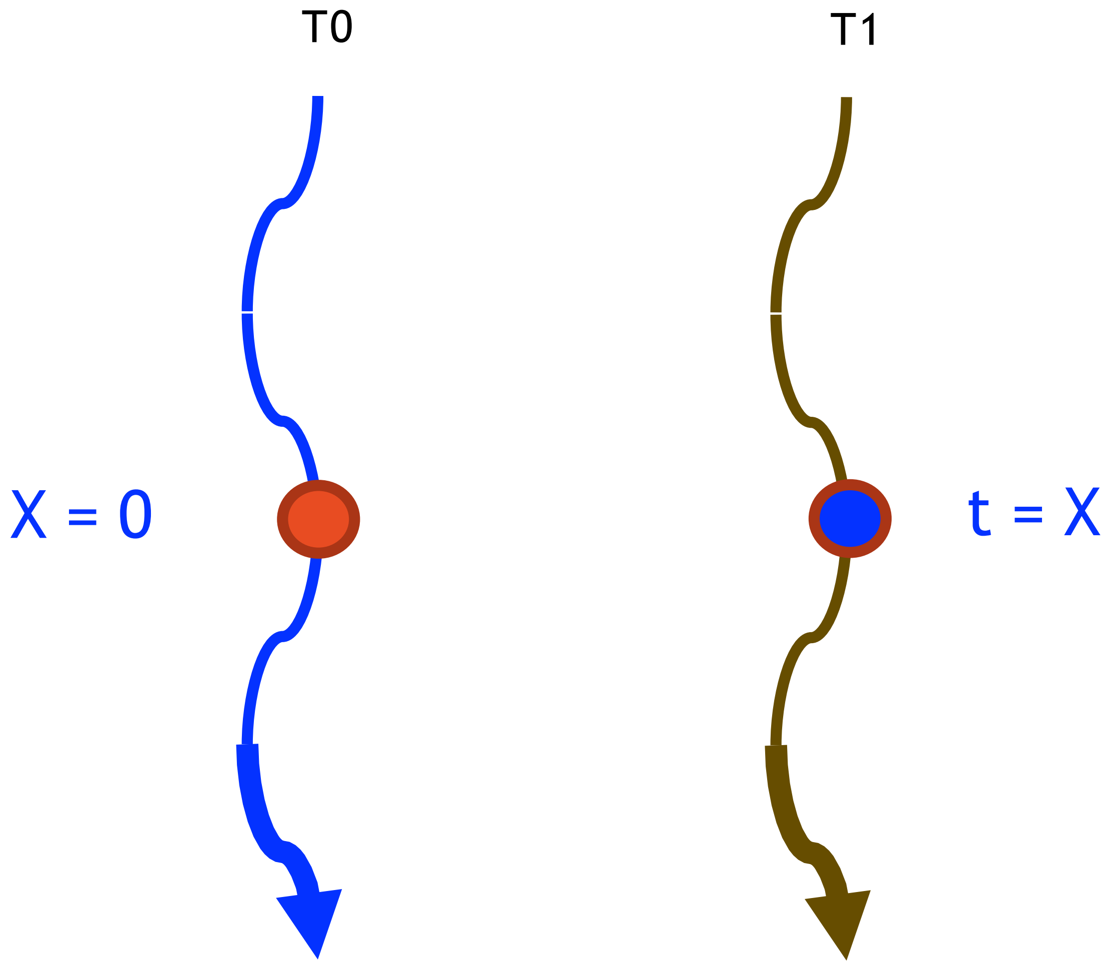


A third way



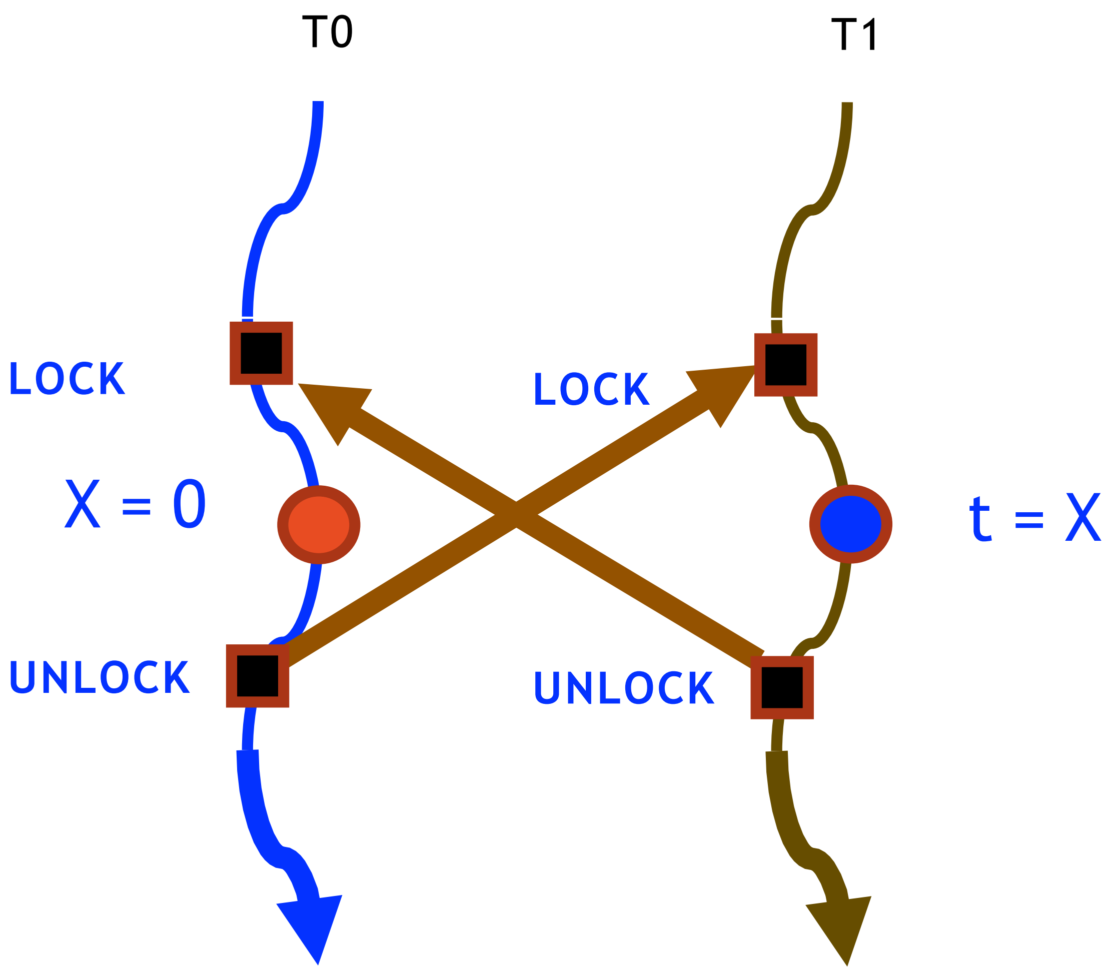
Why eliminate races?

Popular answer: “avoid nondeterminism”

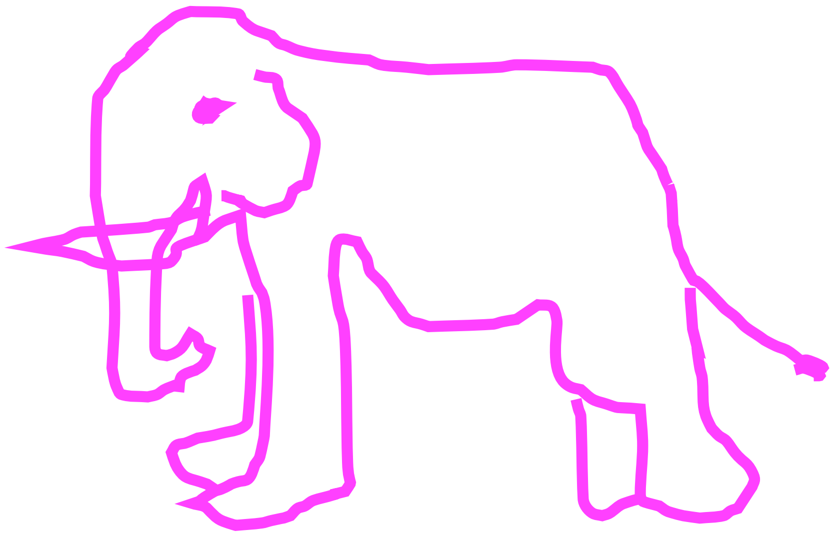


Unclear what “nondeterminism” means..

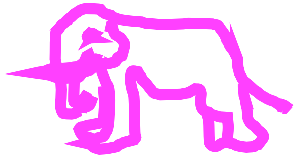
Execution Order is Still Nondeterministic



More relevant: Avoid “pink elephants” ☺



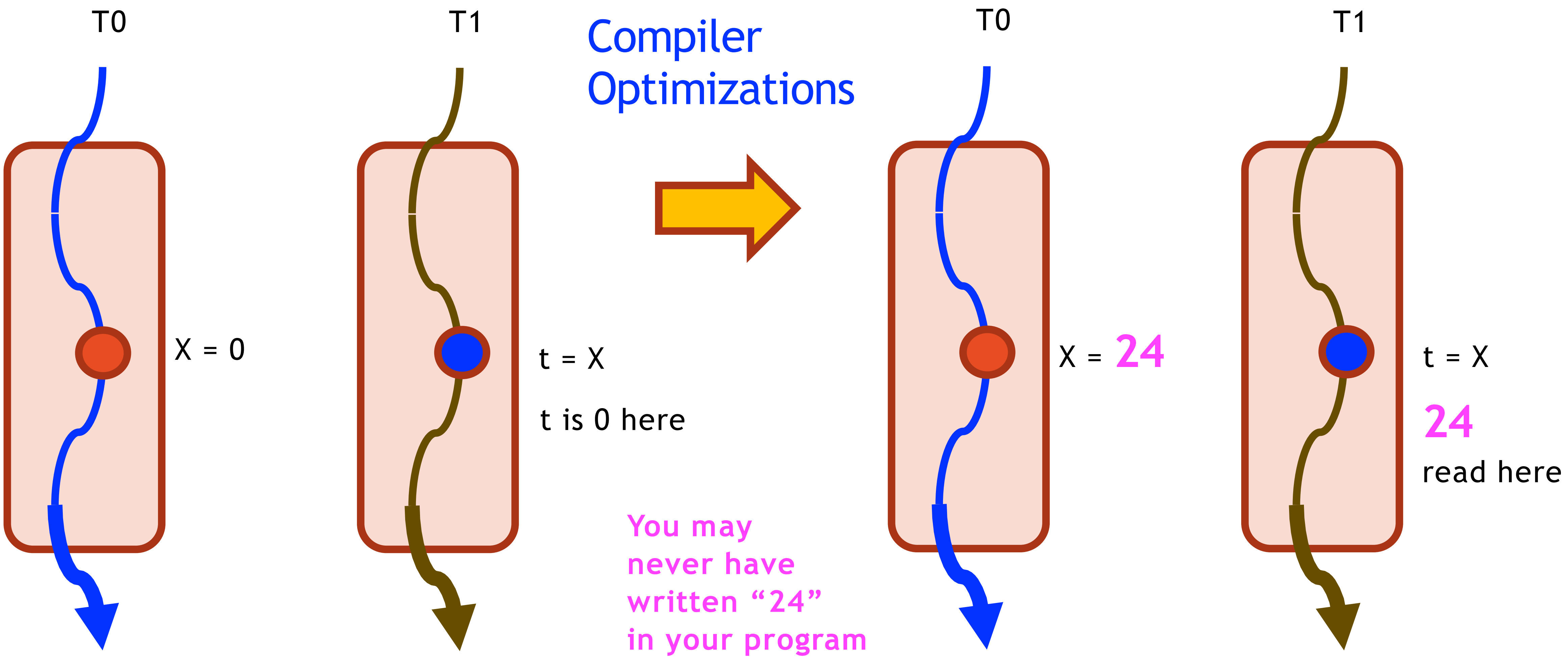
More relevant: Avoid “pink elephants” ☺



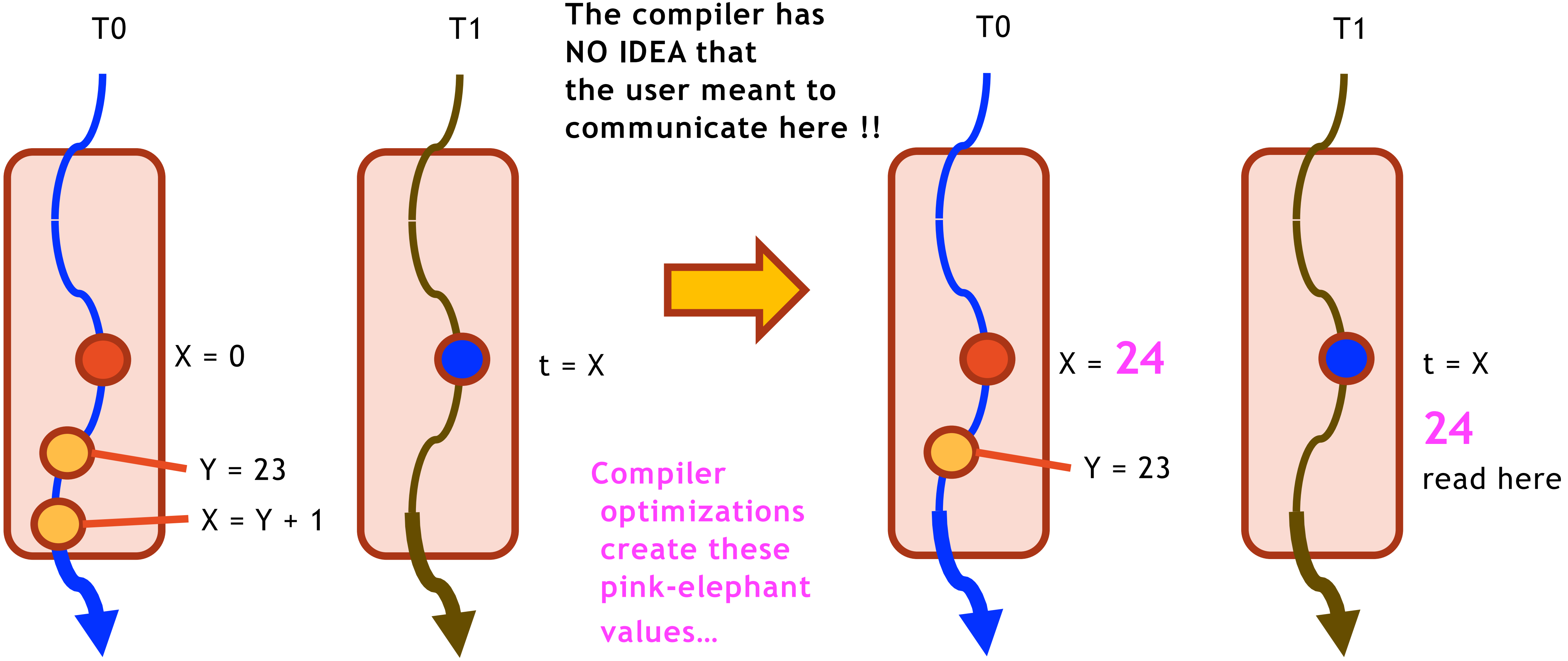
Pink elephant (Sutter) : “A value you never wrote but managed to read”

Aka ”out of thin air” value

The birth of a pink elephant...



Details of how a pink elephant is made!

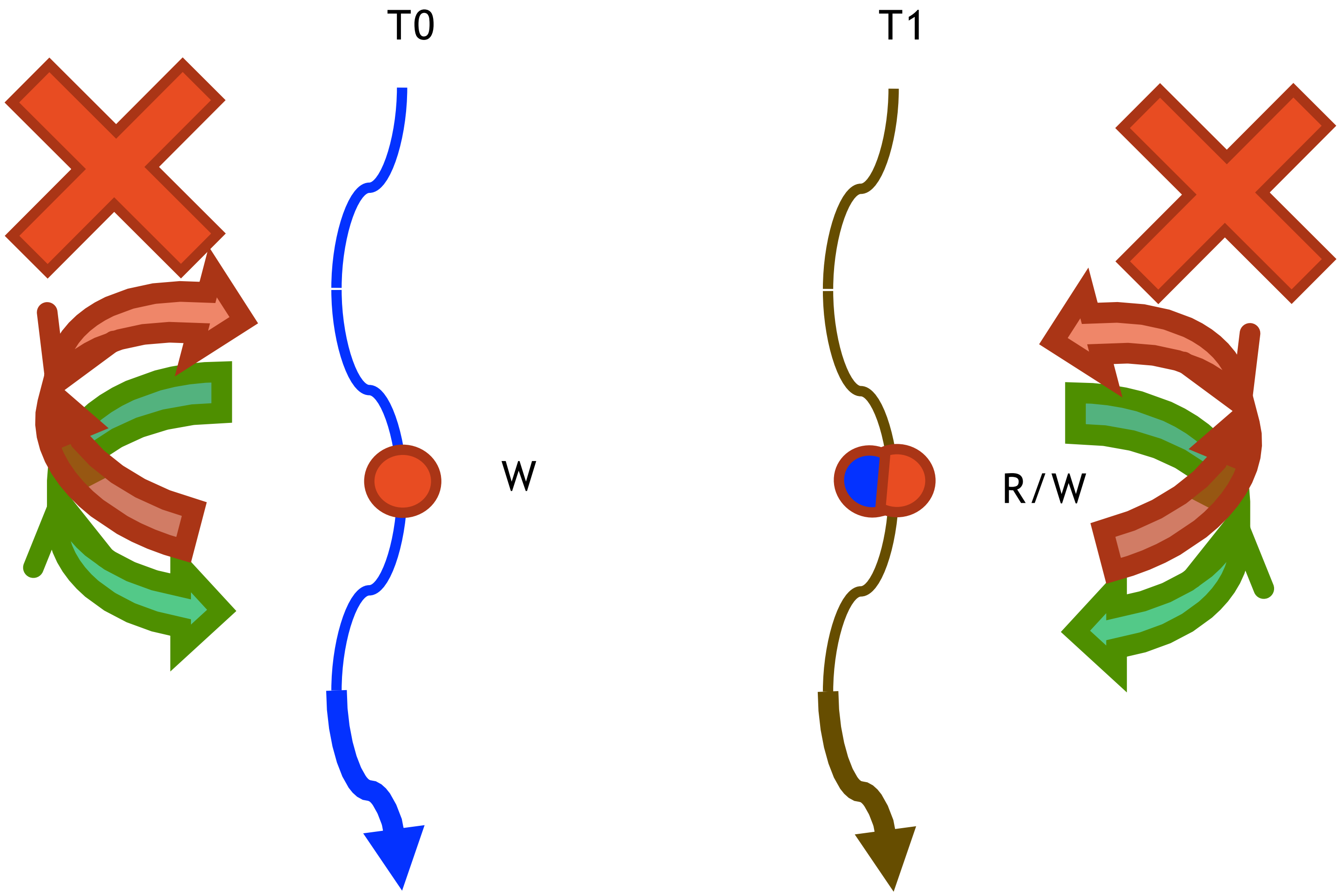


The compiler has
NO IDEA that
the user meant to
communicate here !!

Compiler
optimizations
create these
pink-elephant
values…

This is why code containing data races often fail (only) when optimized!

Race-freedom ensures intended communications



- *You don't observe "half baked" values*
- *Code does not reorder around sync. points*
- *No "word tearing"*
- *Pending writes flushed (fences inserted)*

Exploding a myth!

There is no such thing as a benign race !!

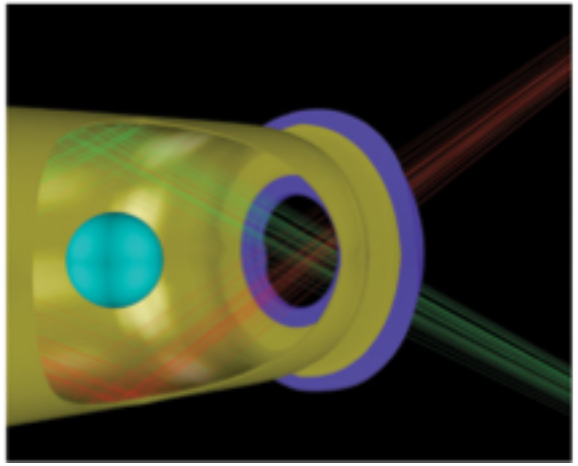
Races in OpenMP programs are hard to spot

- See tinyurl.com/ompRaces if you wish
  - but later ☺
- Static analysis tools never shown to work well
- First usable OpenMP dynamic race checker (afaik)
  - Archer [Atzeni, IPDPS'16]
  - More on that soon
- **This talk** will present the second usable dynamic race checker
  - Sword

This talk: Why and how of another OMP race checker

The Pink Elephant Actually Struck Us!

- HYDRA porting on Sequoia at LLNL
  - Large multiphysics MPI/OpenMP application
  - Non-deterministic crashes in OpenMP region
    - Only when the code was optimized!
  - Suspected data race
  - Emergency hack:
    - Disabled OpenMP in Hypre



Archer to the rescue!

Archer to the rescue!

Archer [IPDPS'16]

- Utah: Simone Atzeni, Ganesh Gopalakrishnan, Zvonimir Rakamaric
- LLNL: Dong H. Ahn, Ignacio Laguna, Martin Schulz, Gregory L. Lee
- RWTH: Joachim Protze, Matthias S. Muller

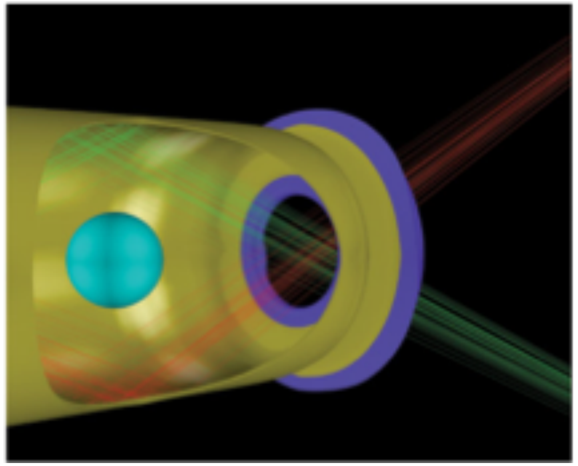
- In production use at LLNL

Part of the "PRUNERS" tool suite

PRUNERS was a finalist of the 2017 R&D 100 Award Selection

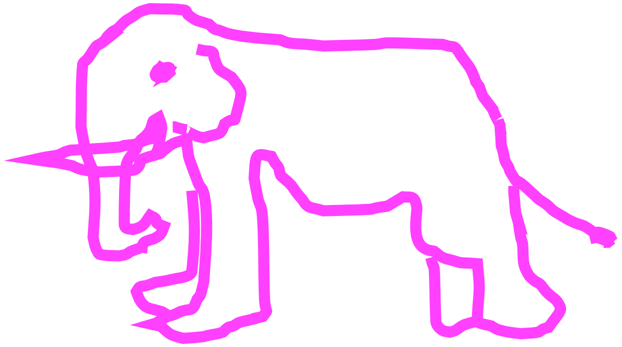
Archer’s “find”

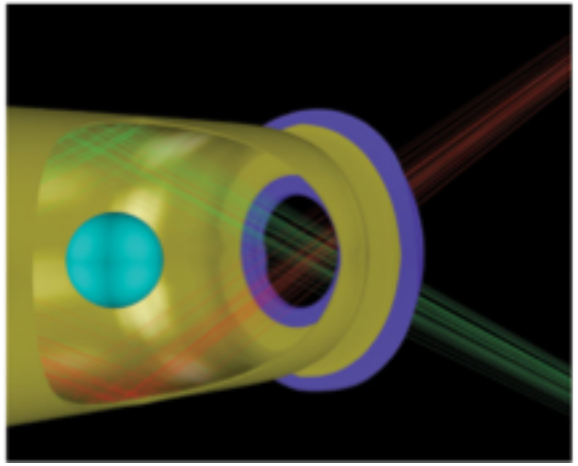
Two threads writing 0
to the same location
without synchronization



Archer's "find"

Two threads writing 0
to the same location
without synchronization





Did we live “happily ever after?”

No ☹

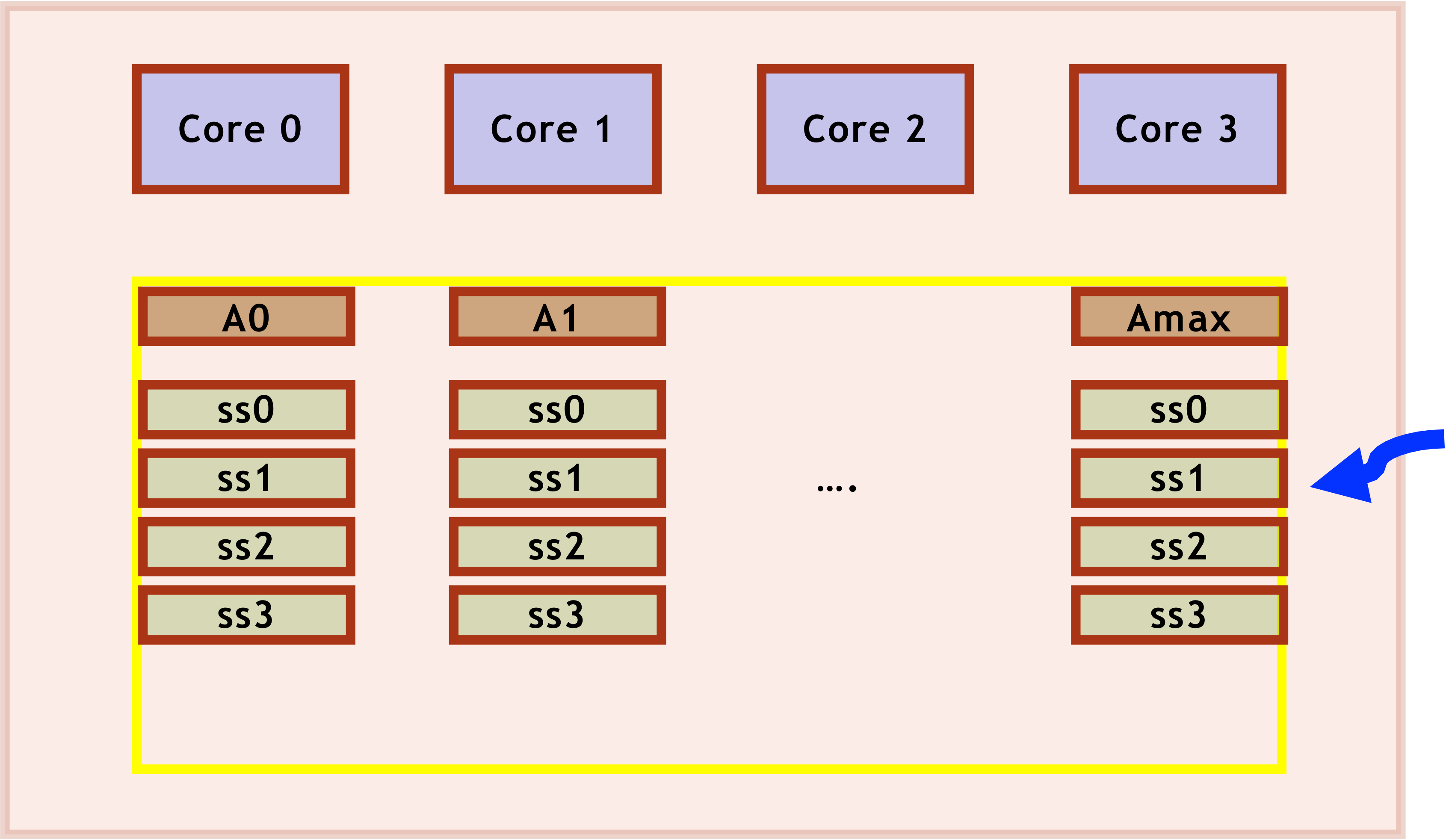
Archer has “memory-outs”; also misses races

Archer has “memory-outs”; also misses races

- Archer increases memory 500%
- It also misses races!

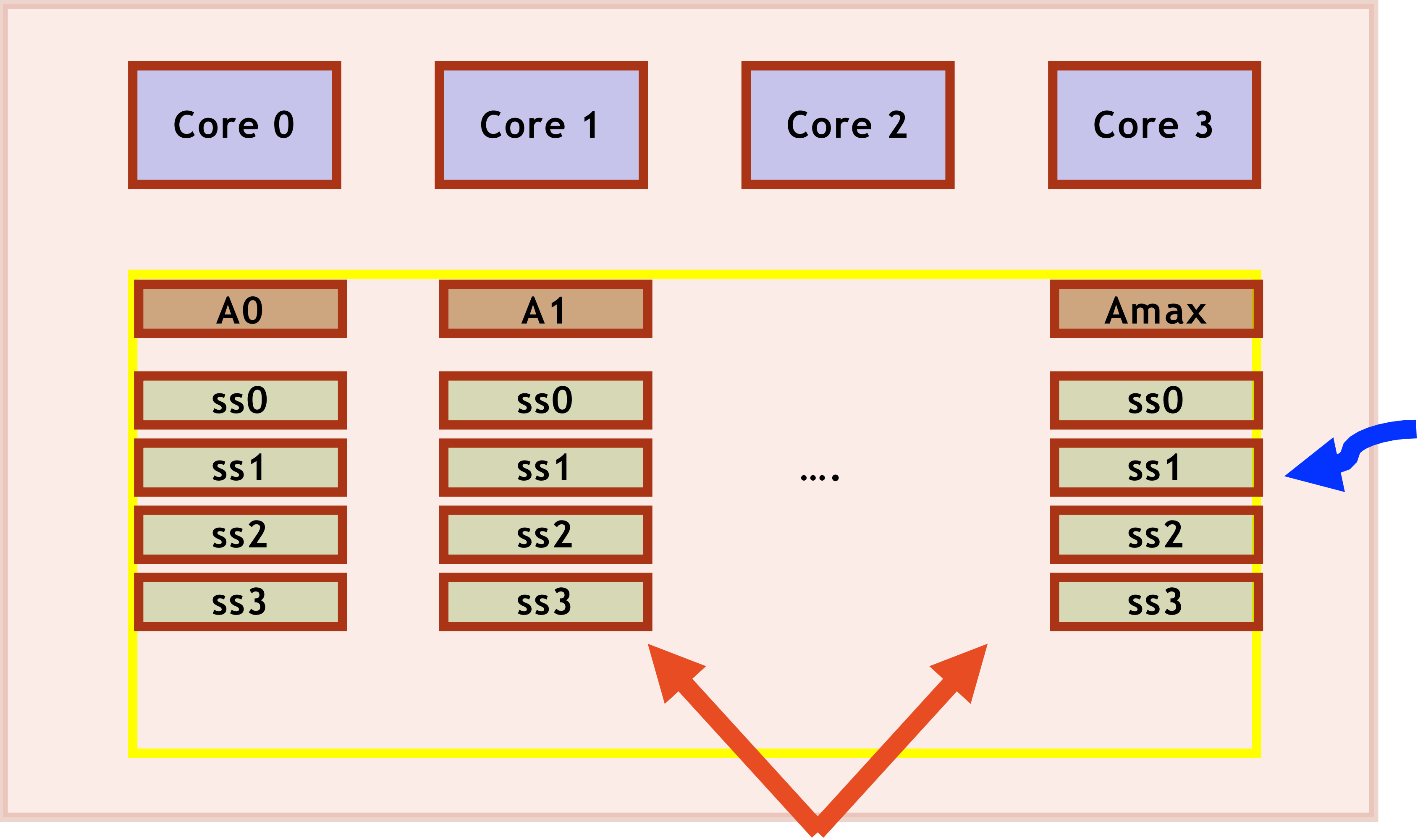
- These were known issues
  - **Finally surfaced with the ”right large example”**

Reason: Archer employs “shadow cells”



A programmable number of cells per address (4 shown, and is typical)

~4 shadow cells per application location

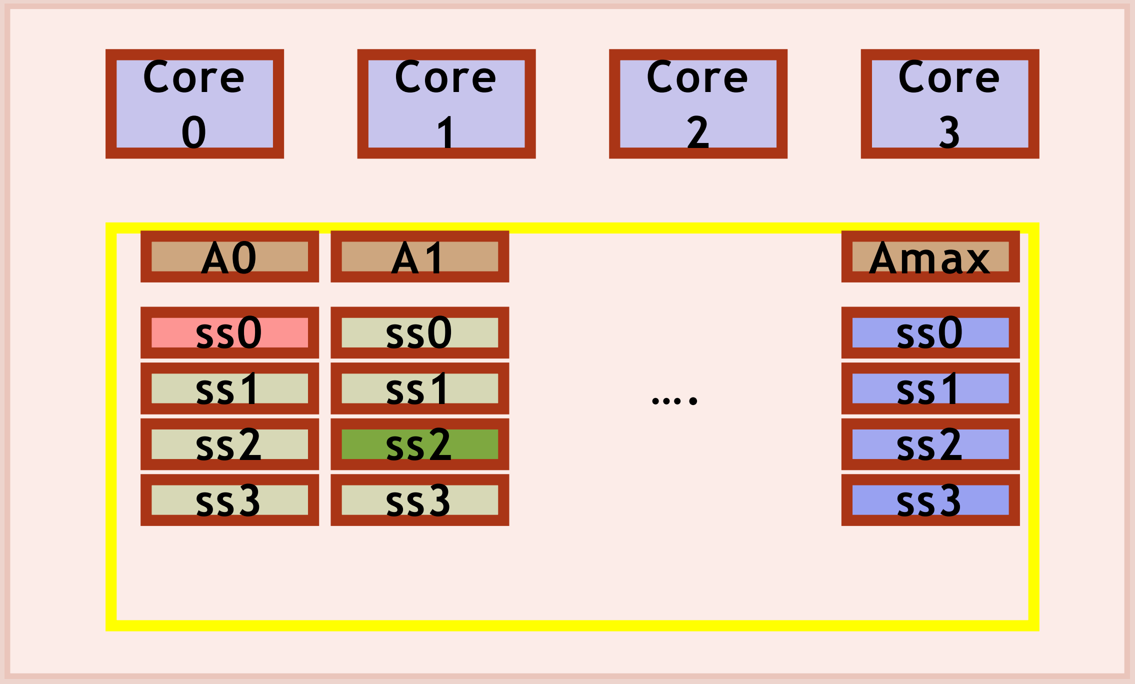


A programmable number of cells per address (4 shown, and is typical)

Shadow-cells immediately increase memory demand by a factor of four

Archer misses races due to shadow cell eviction

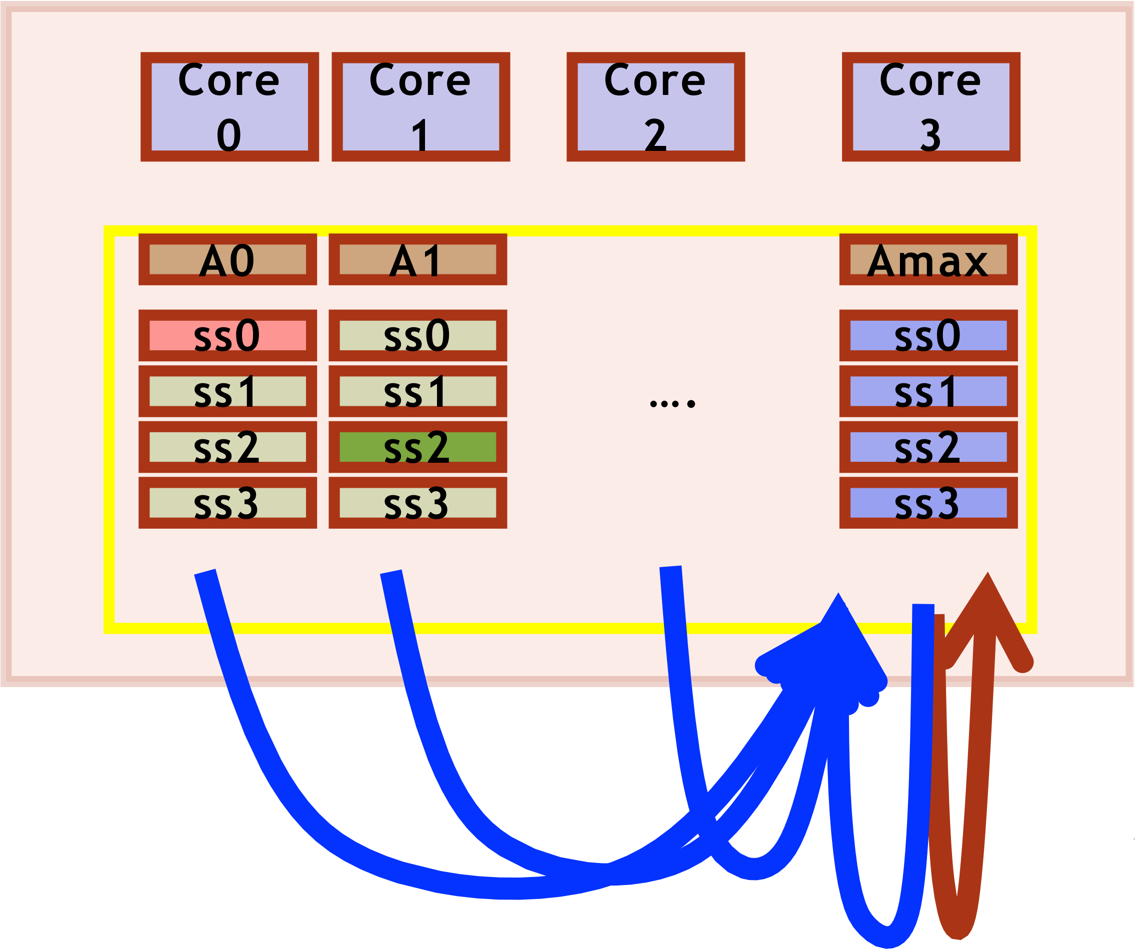
Archer misses races due to shadow cell eviction




```c
int a[N];

#pragma omp parallel for
for(int i = 0; i < N; i++) {
  a[i] = a[i] + a[3];
}
```

Archer misses races due to shadow cell eviction



All threads read A[3]
Thread 3 writes A[3]

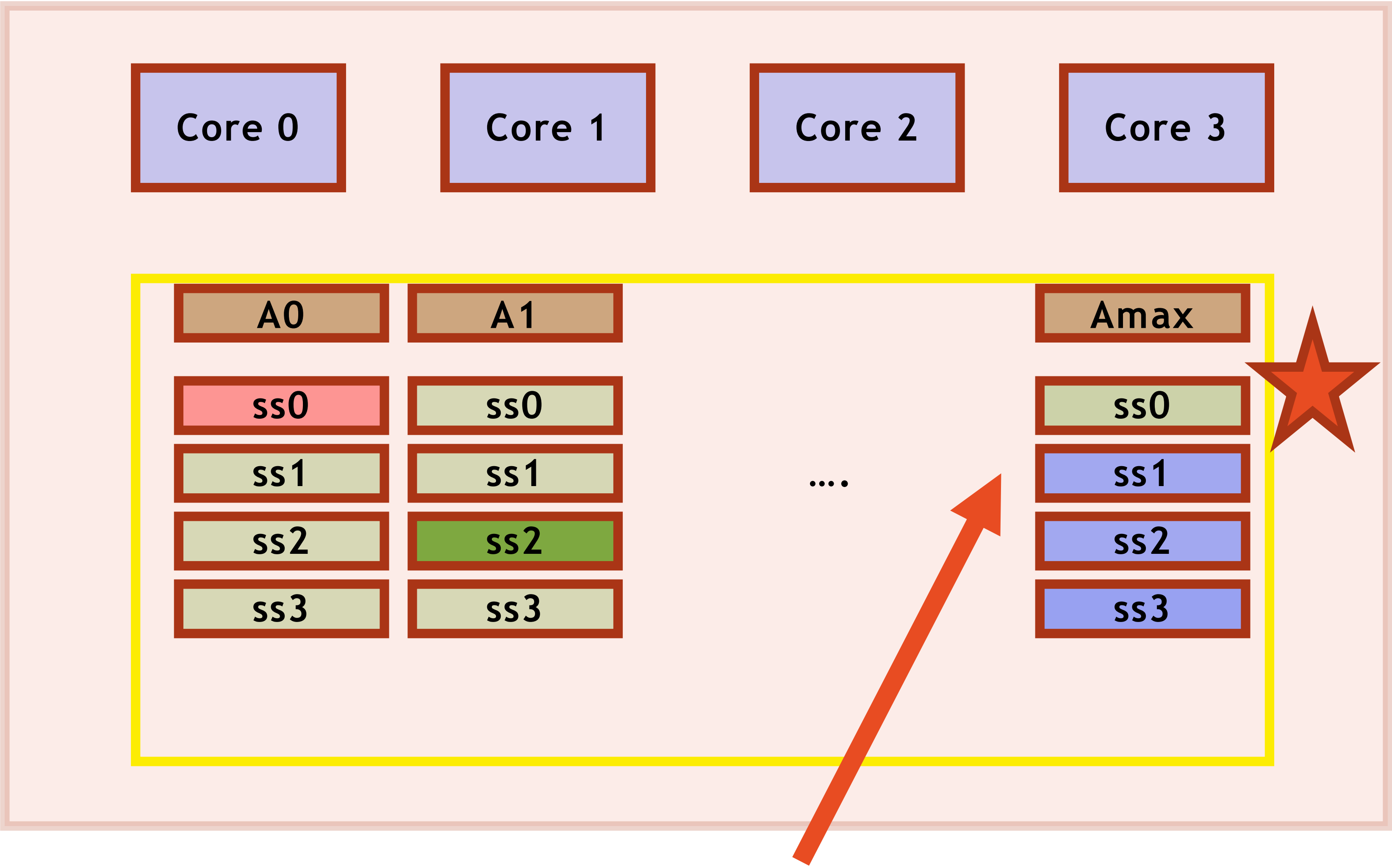
```
int a[N];

#pragma omp parallel for
for(int i = 0; i < N; i++) {
  a[i] = a[i] + a[3];
}
```

Thread 3 writes a[3]

All threads read a[3]

Capacity conflict → evict shadow cell

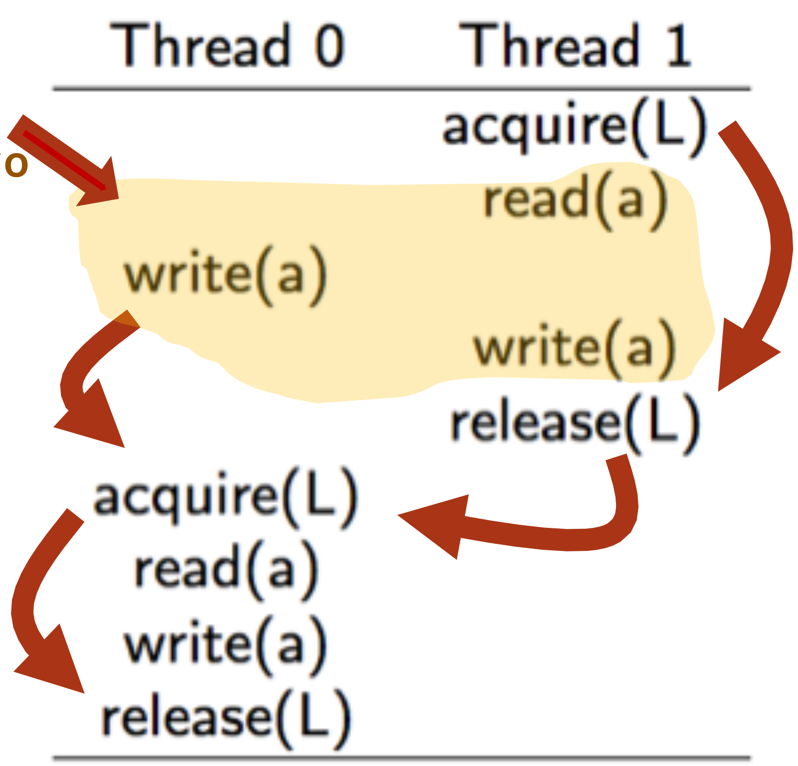


With shadow-cell evicted, races are missed

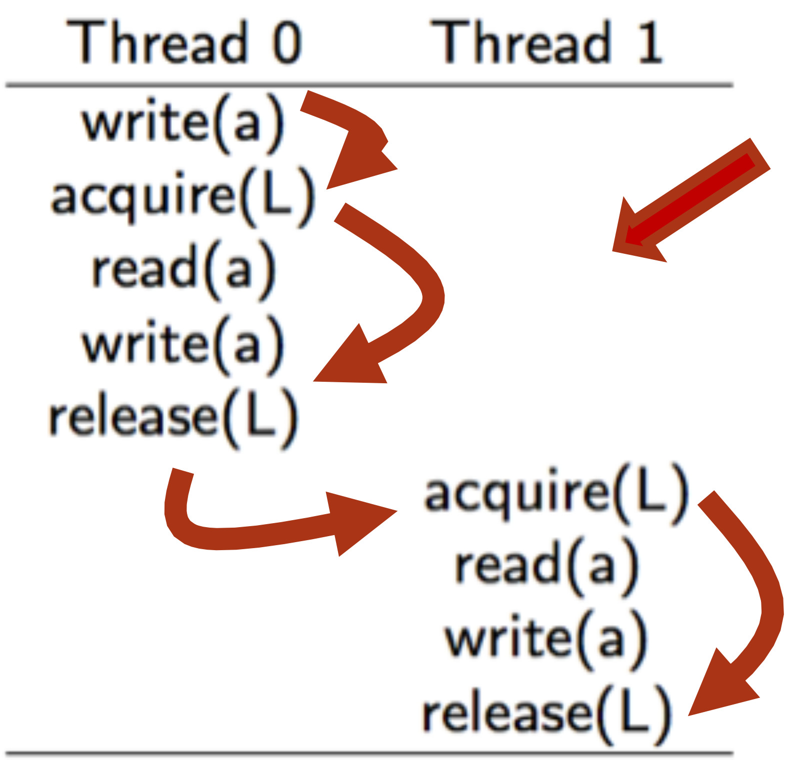
Archer misses races due to HB-masking

Archer misses races due to HB-masking

These are concurrent; there are two races here!



(a) No happens-before (race detected)

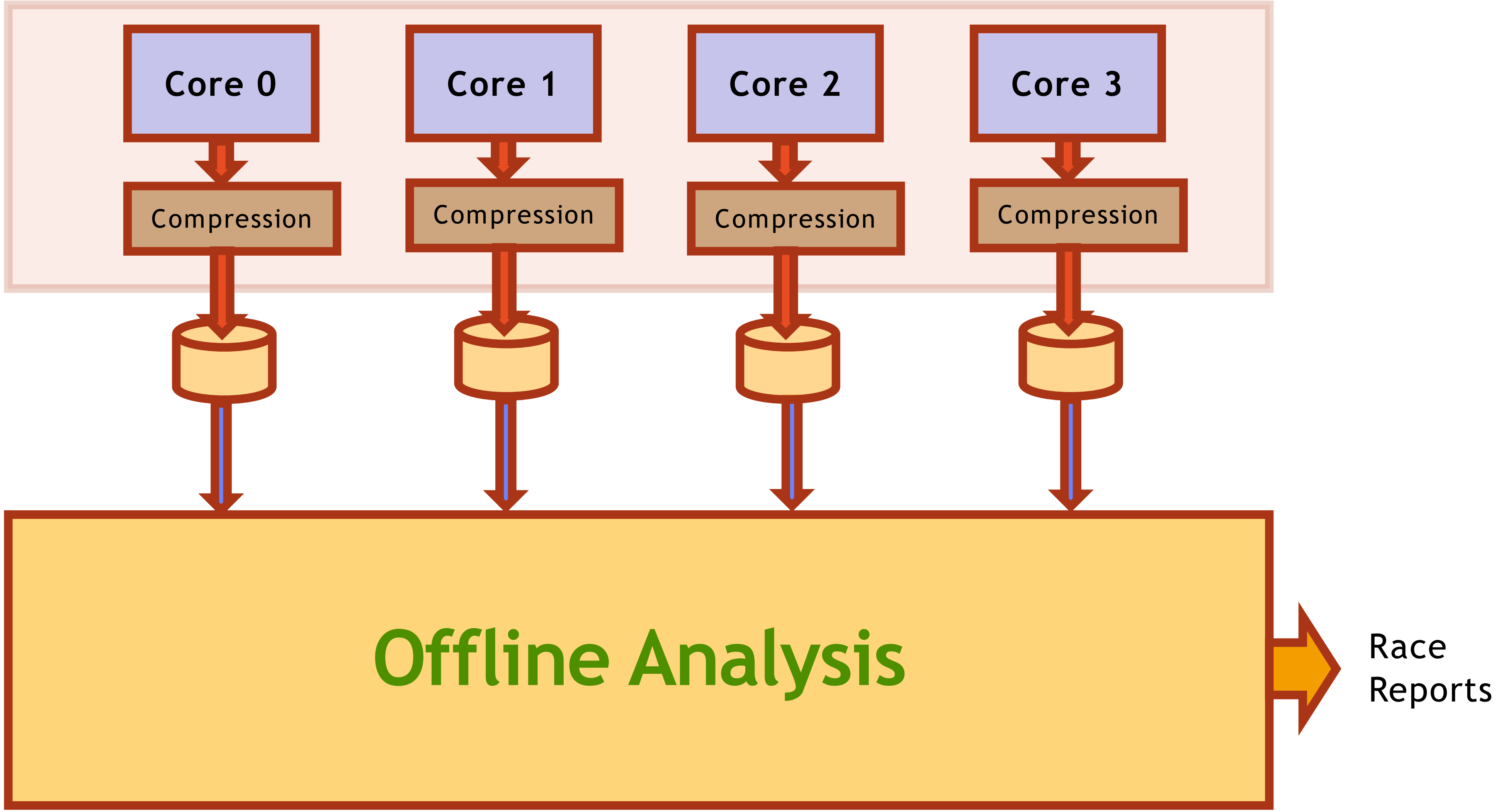


(b) Happens-before (no race detected)

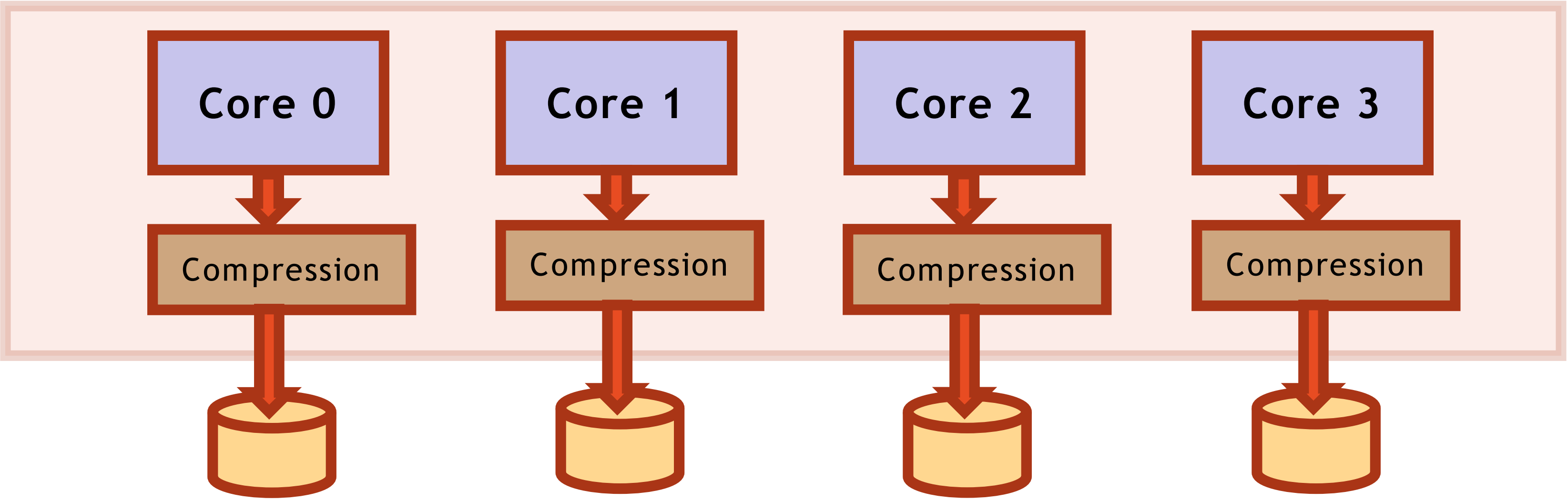
These races are missed in this interleaving!

Solution : Get rid of shadow cells !!

Need New Approach with Online/Offline split



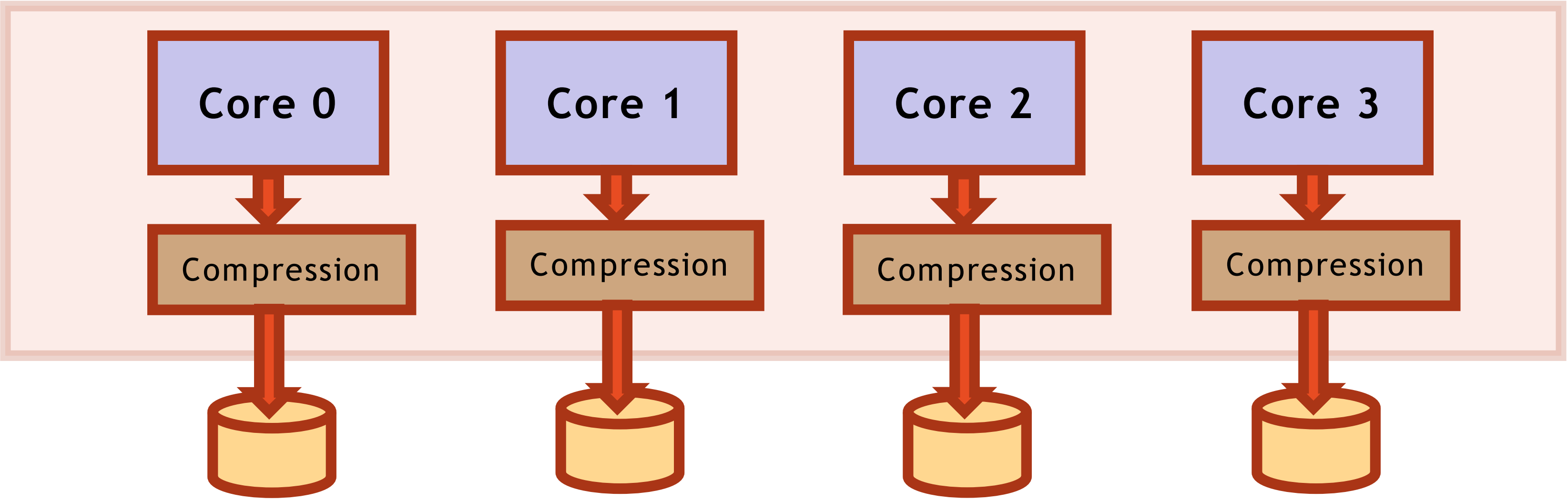
Details of the online phase



- Collect traces per core **un-coordinated**
  - **Trace collection speeds increased; we use the OMPT tracing method**

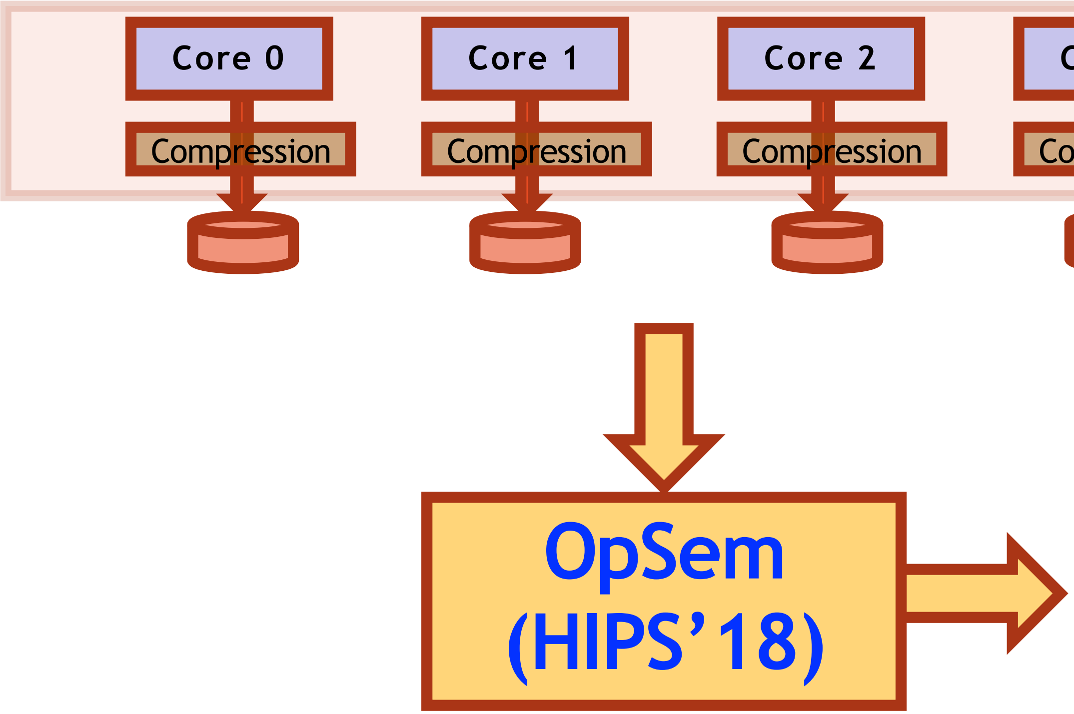
- **Employ data compression to bring FULL traces out**
  - Only 2.5 MB compression buffer per thread **(fits in L3 cache)**

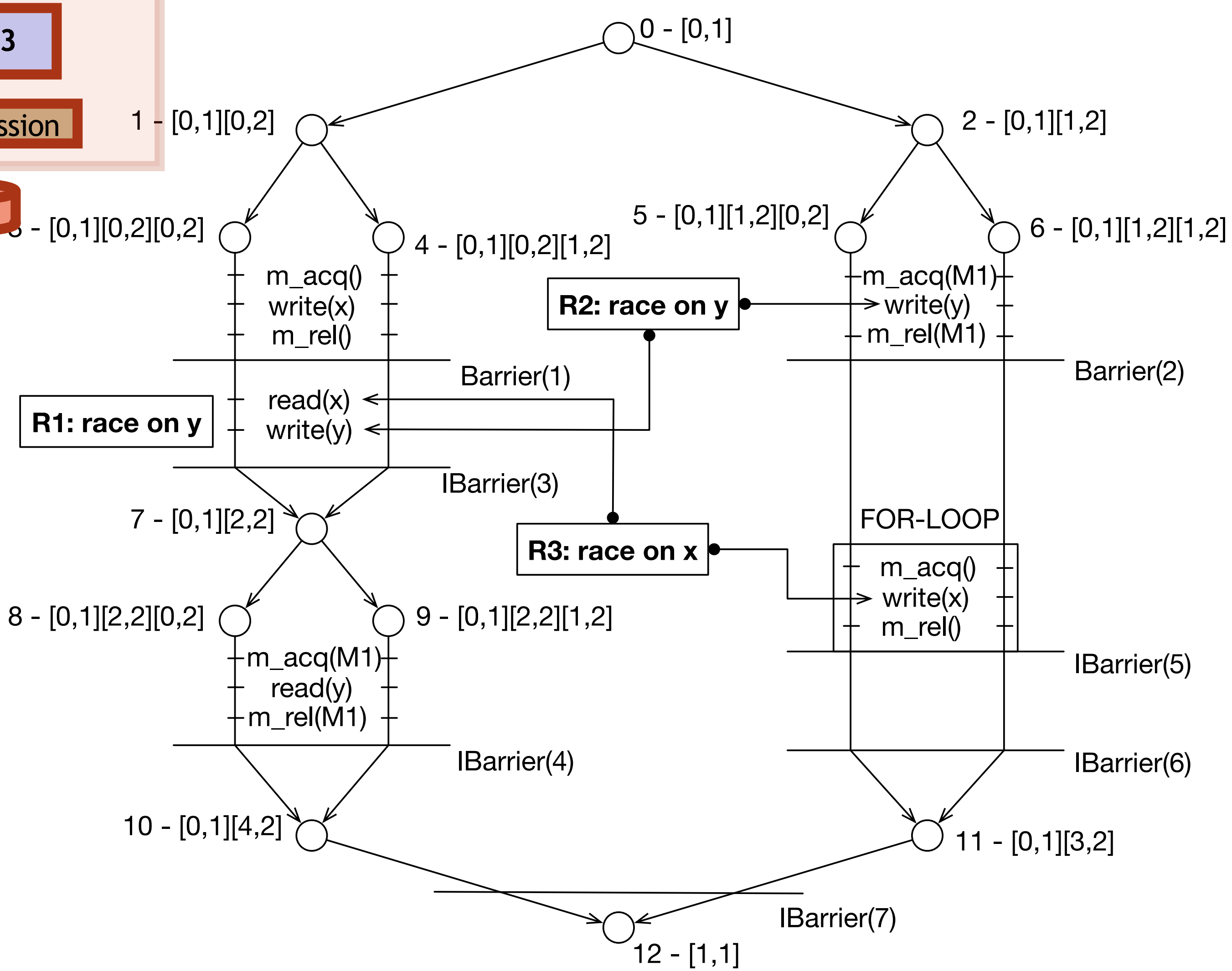
Consequences for the offline phase



- We would have lost all the synchronization information
  - We only know what each thread is doing
- We must recover the concurrency structure
  - And in the context of its happens-before order, detect races!

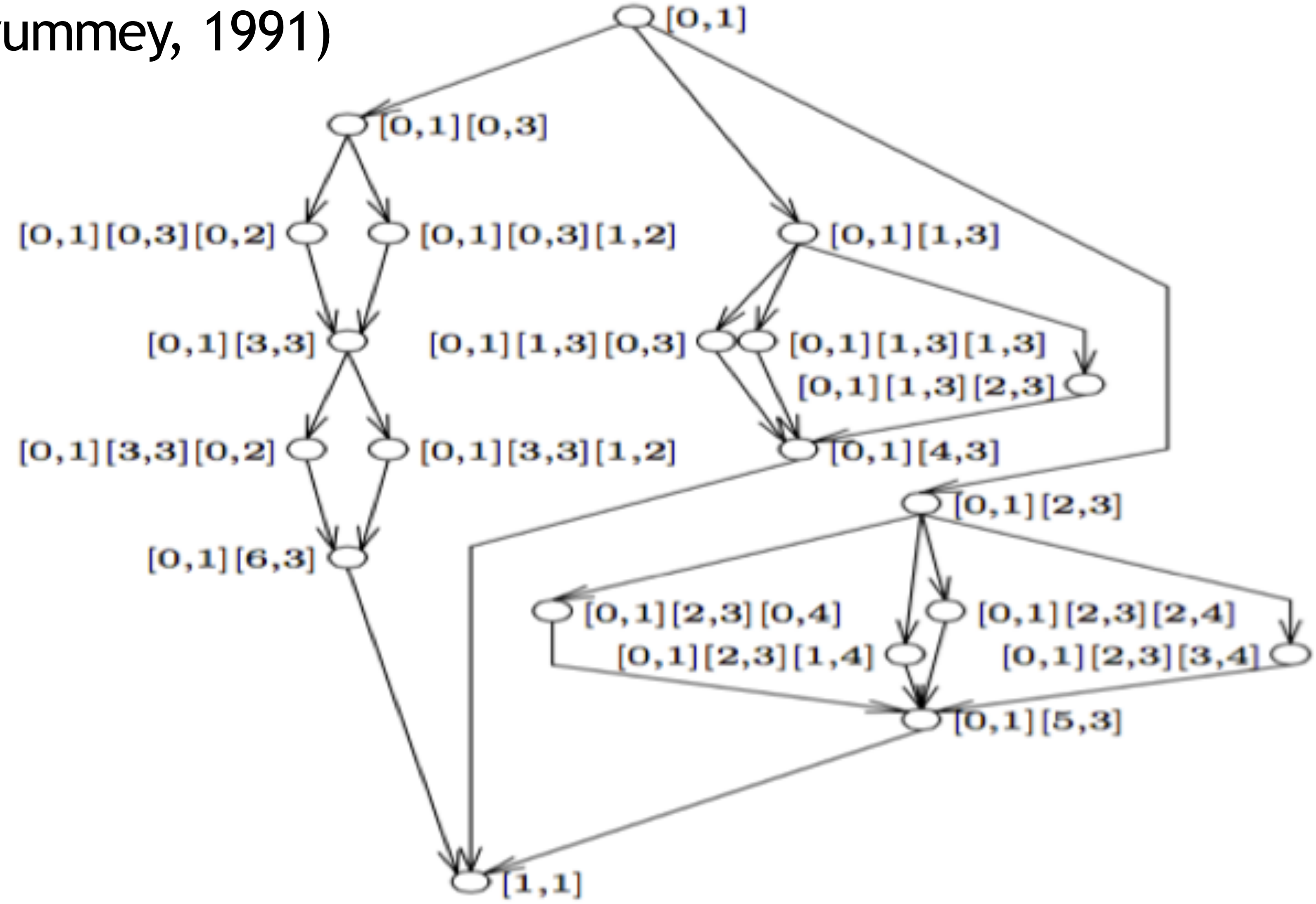
Offline synchronization recovery and analysis



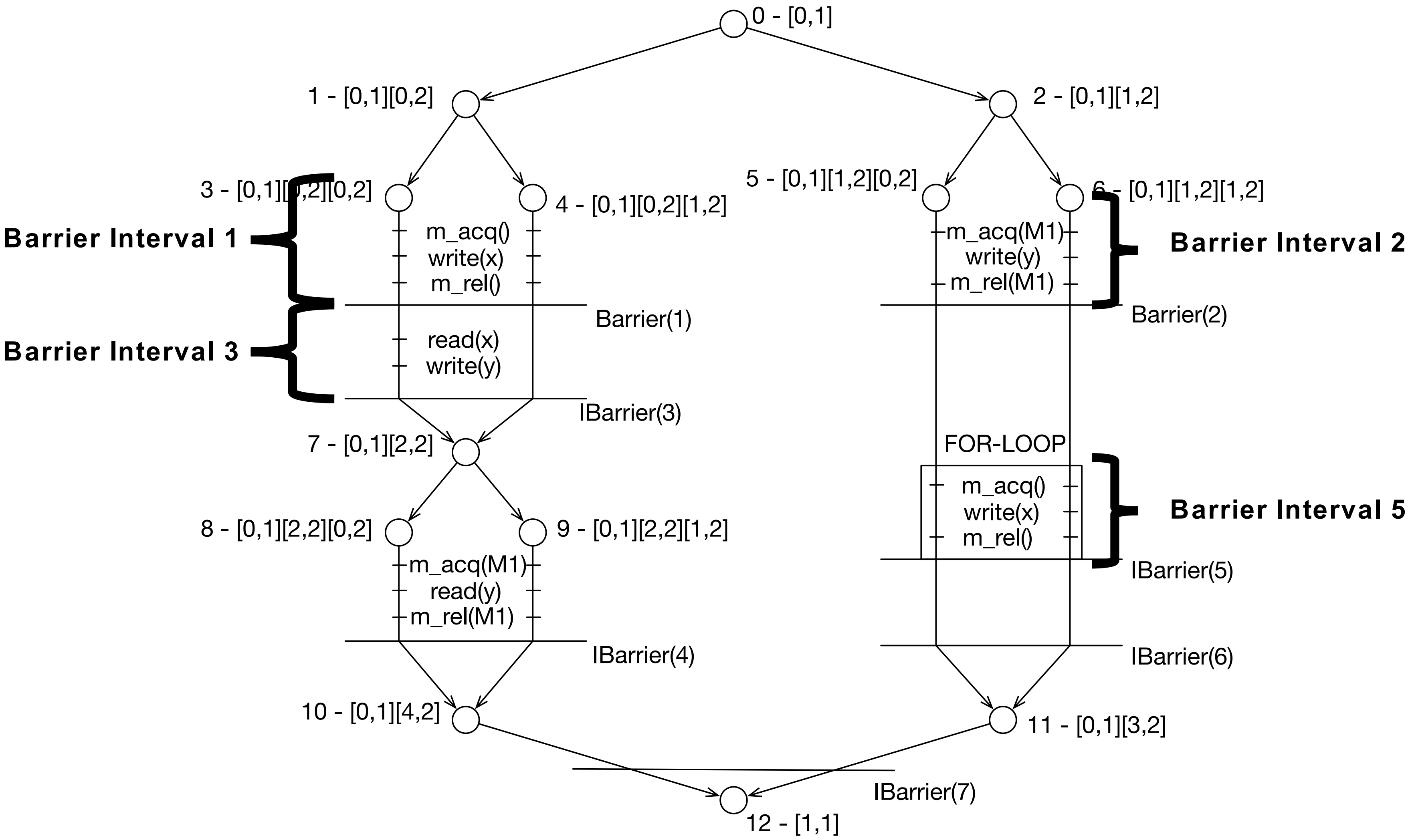


Offset-Span Labels: How we record concurrency

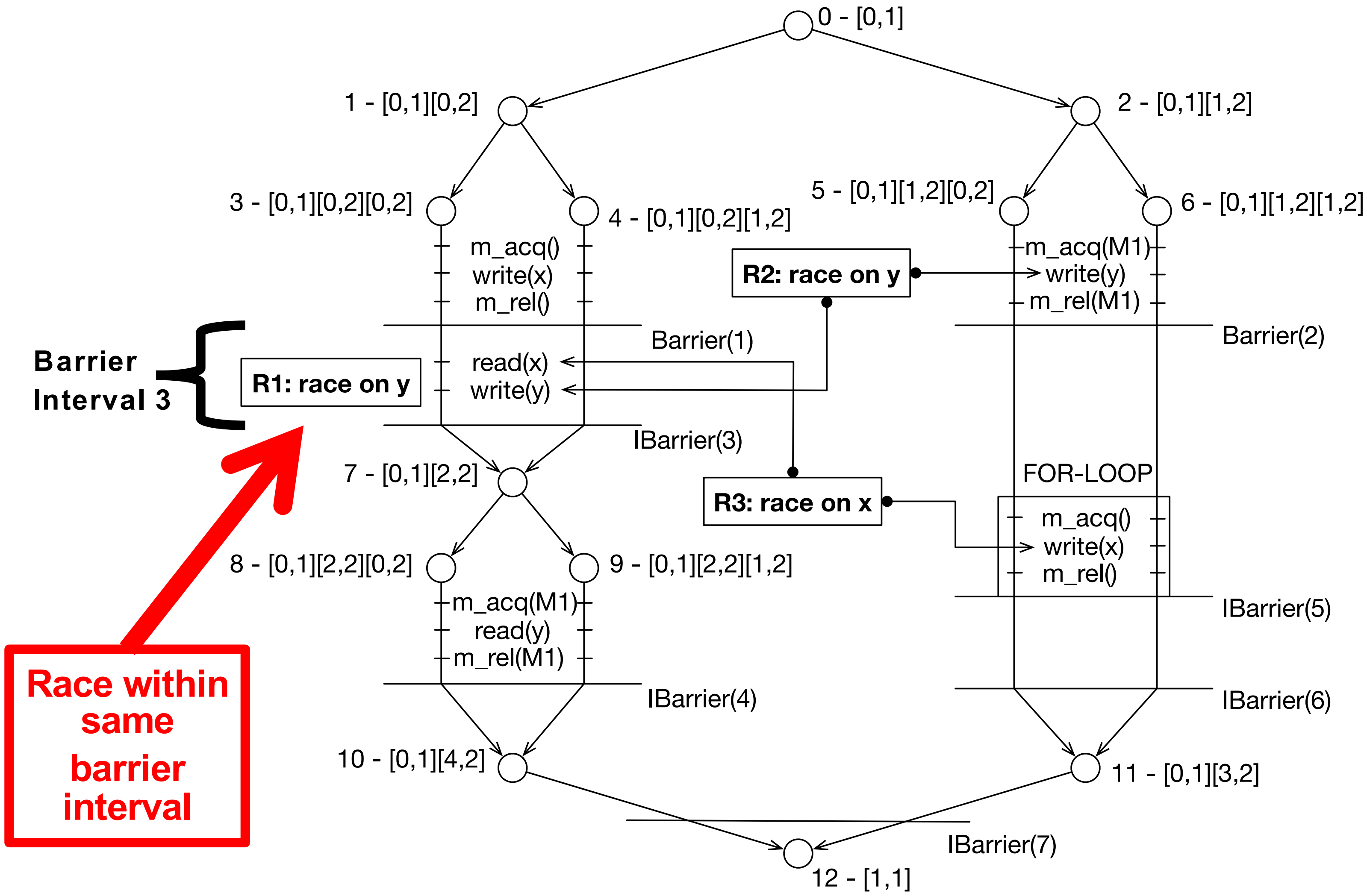
(Mellor-Crummey, 1991)



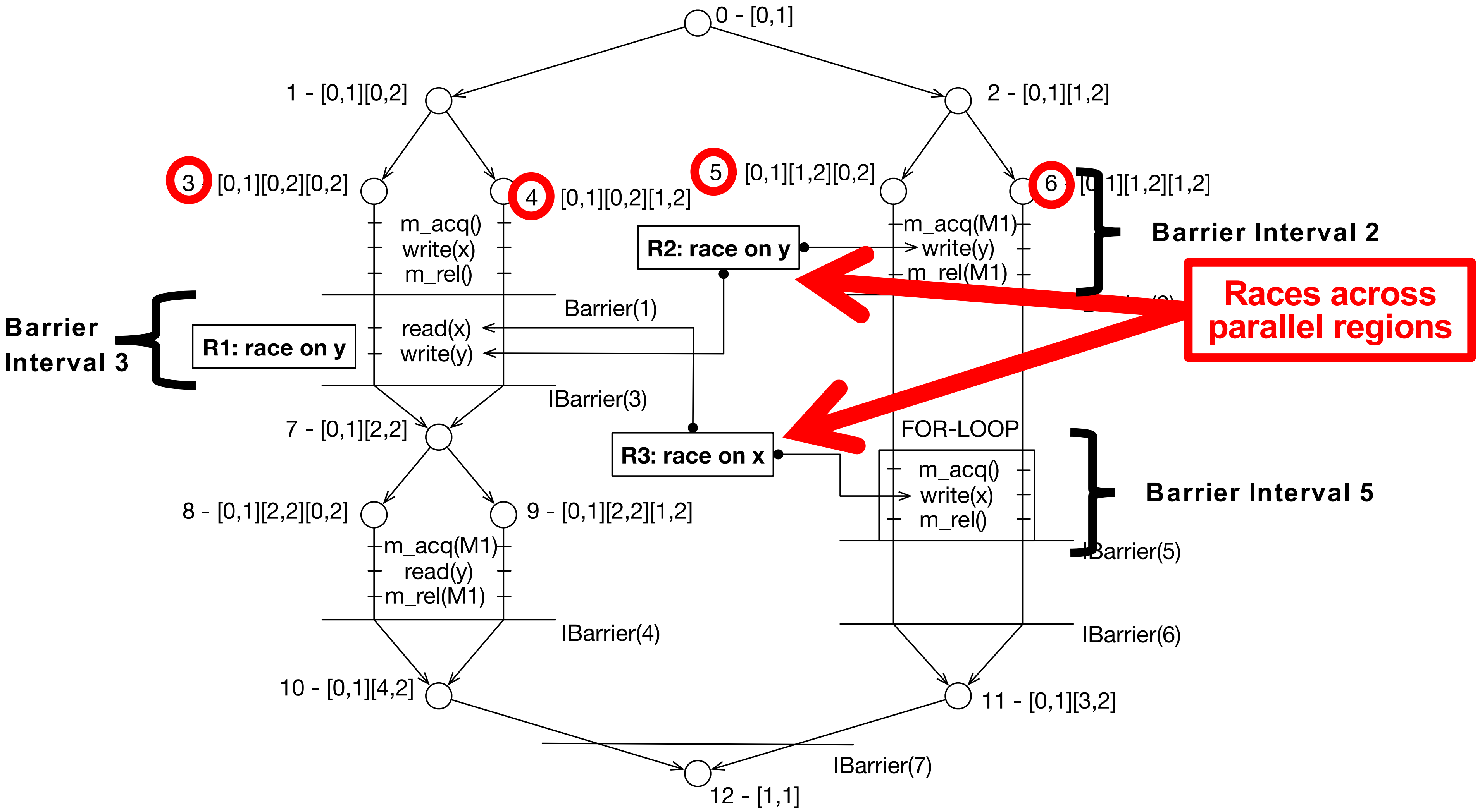
Key state in OpSem: Maintain Barrier Intervals



Examples of Races Reported



Examples of Races Reported



Good news

- Online analysis proved really good
  - **No memory pressure !!**

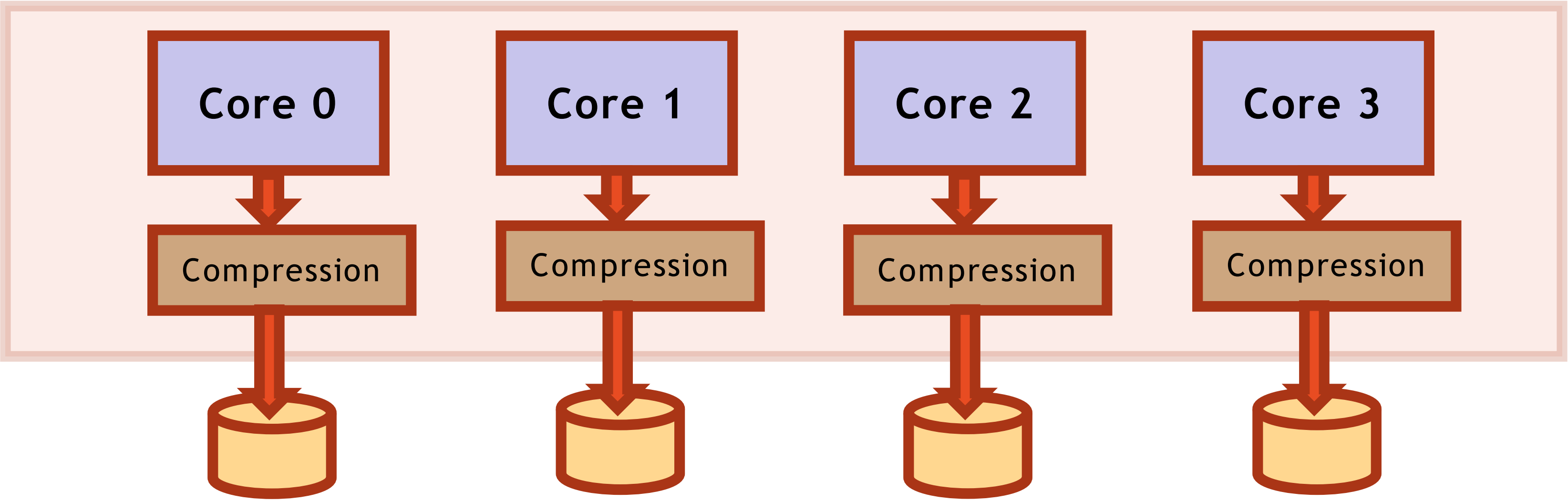
Bad news

Offline analysis ***took a day to finish on "medium sized" examples***

Two Key Innovations Saved the Approach

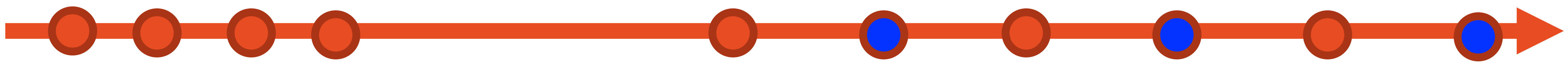
- Self-balancing **red-black** interval trees
- On-the-fly generation of Integer Linear Programs

Reducing “a day” to “under a minute”



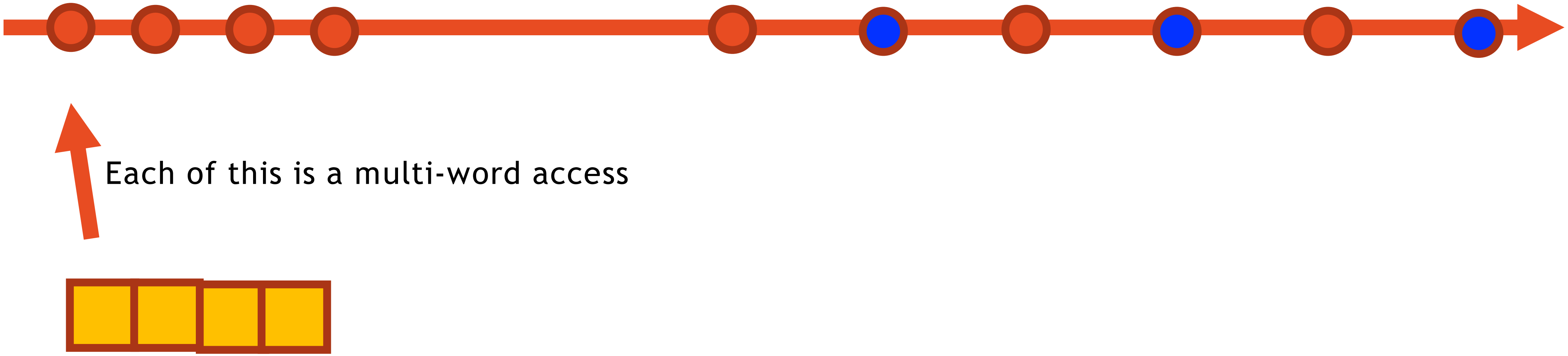
- Decompress, record **strided accesses** in self-balancing **red**-**black** interval trees
- Generate Integer Linear Programs on-the-fly, and check for overlaps
  - **Handles bursts of accesses efficiently**

OMP read/writes are bursty with strides!

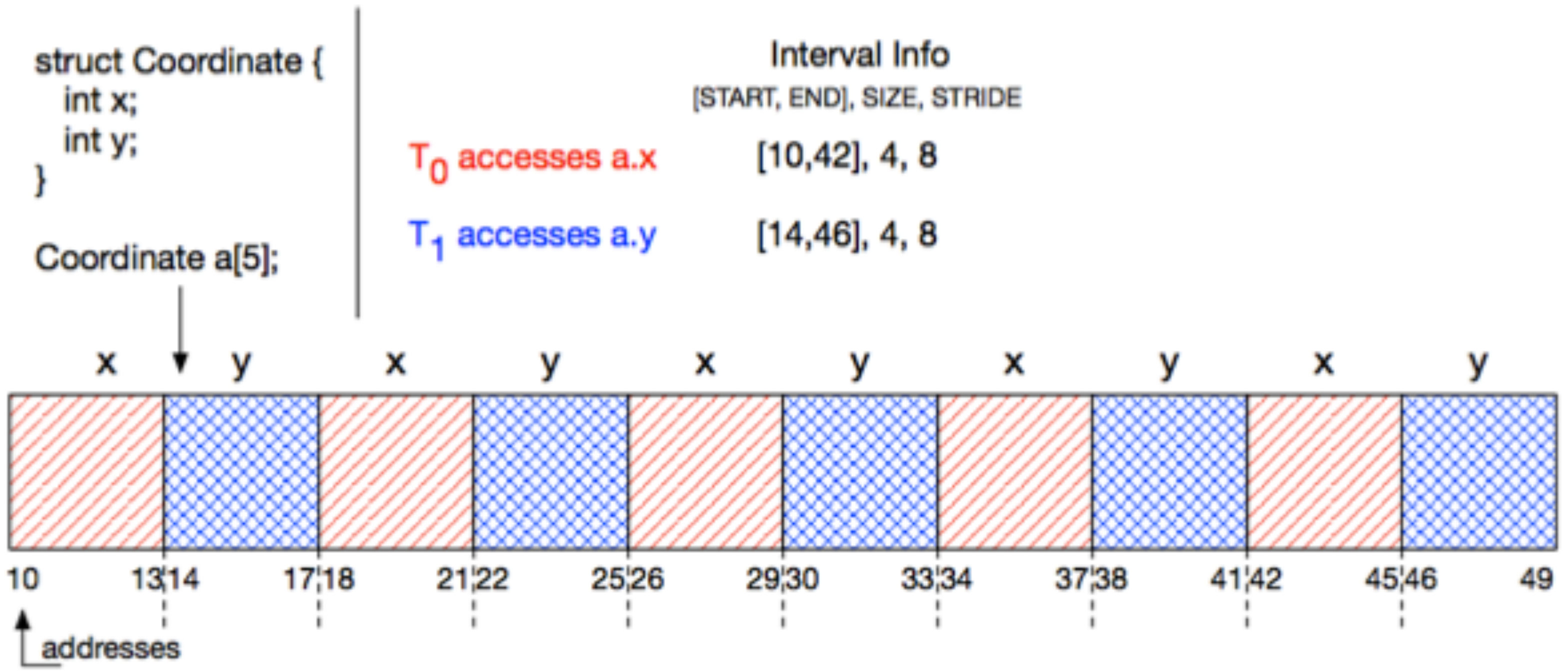


OMP read/writes are bursty with strides!

Build Integer Linear Programs for each constant-stride interval
ILP system encodes accessed byte-addresses in each “burst”

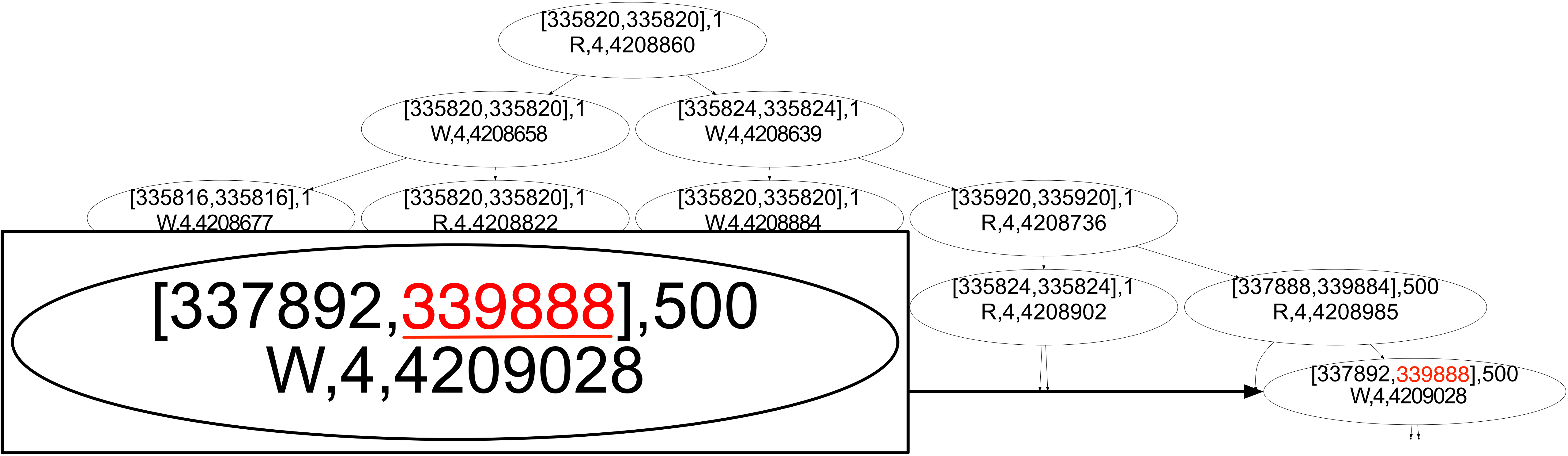


Overlap of Access Bursts: ILP Generation!



$$T_0: \quad 8 \cdot x_0 + 10 + s_0 = a \qquad\qquad T_1: \quad 8 \cdot x_1 + 14 + s_1 = a$$
$$\wedge \; 0 \leq x_0 \leq 4 \qquad\qquad \wedge \; 0 \leq x_1 \leq 4$$
$$\wedge \; 0 \leq s_0 < 4 \qquad\qquad \wedge \; 0 \leq s_1 < 4$$

Interval Trees to record accesses



- **Recorded info is:** [Begin, End], #Accesses, Kind, Stride, AtWhichPCValue
- Allows efficient comparison of ***access bursts*** across threads
- These Red-Black trees are highly tuned
  - Used within Linux to realize fair scheduling methods

Concluding Remarks: Sword is now practical!

Both Archer and Sword are available

Github.com / PRUNERS

Conclusions: Time for "Medium" Examples

	Online	Offline	Total	Efficacy
Archer	1	0	1	Misses races
Sword	1	10*	11	Finds all races within the execution**

* : can be brought down to 1 by using an MPI cluster

** : we define the formal semantics of OMP race checking [HIPS'18]

Conclusions: Time for Larger Examples

	Online	Offline	Total	Efficacy
Archer	Memory	0	1	Misses races
Sword	1	10*	11	Finds all races within the execution**

More Concluding Remarks

- Sword works well ; finds more races than Archer
  - Applied to realistic benchmarks
    - **Archer test suite**
    - **RaceBench from LLNL**
  - Offline analysis can be parallelized
  - Still “decent” on standard multicore platforms
- It took ***many ideas*** working together to realize Sword
  - **Formal semantics of OpenMP Concurrency**
  - **Online / Offline checking split**
  - **Data compression**
  - **Self-balancing interval trees**
  - **ILP-systems to compress traces**
  - **Employs standard tracing methods based on OMPT**

Future Work

- Continue to debug / tune Sword
- Incorporate ideas from upcoming pubs
- GPU race checking

Group Credits

Simone

Zvonimir

Dong

Ignacio

Greg

Extras

Data Races: Gist

- High-level code is just “fiction”
  - Code optimizations are done on a PER THREAD basis
  - Races occur if you don’t tell a compiler what’s shared

while(!f) {} → r = f; while (!r) {} : this is OK if “f” is purely local

while(!f) {} → r = f; while (!r) {} : not OK if f is shared and you don’t tell this to the compiler

- How to inform a compiler
  - Put the variables inside a mutex (or other synchronization block)
  - Declare them to be a Java volatile or C++11 atomic
    - C-volatiles won’t do (they don’t have a definite concurrency semantics)

Data Races: Gist

- High-level code is just "fiction"
  - **Code optimizations are done on a PER THREAD basis**
  - **Races occur <u>if you don't tell a compiler what's shared</u>**

`while(!f) {}` → `r = f;  while (!r) {}` : this is OK if "f" is purely local

`while(!f) {}` → `r = f;  while (!r) {}` : not OK if f is shared and you don't tell this to the compiler

Position paper: Nondeterminism is unavoidable, but data races are pure evil

Hans-J. Boehm
HP Laboratories
Hans.Boehm@hp.com

GPUs races also can lead to "pink-elephants"

Initially : x[i] == y[i] == i

Analogy due to Herb Sutter

Warp-size = 32

```
__global__ void kernel(int* x, int* y)
{
 int index = threadIdx.x;

 y[index] = x[index] + y[index];

 if (index != 63 && index != 31)
  y[index+1] = 1111;

}
```

The hardware schedules these instructions in "warps" (SIMD groups).

However, this "warp view" often appears to be lost

E.g. When compiling with optimizations

Expected Answer: 0, 1111, 1111, …, 1111, 64, 1111, …

New Answer: 0, 2, 4, 6, 8, …