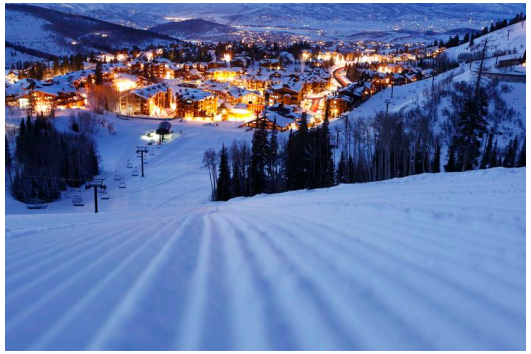
Making Formal Methods for HPC Disappear*

* Adapted from Rushby's http://www.csl.sri.com/papers/hase00/

Ganesh Gopalakrishnan
School of Computing, University of Utah, **Salt Lake City**, UT 84112



www. cs. utah. edu / fv

Pruners.Github.io

THE UNIVERSITY OF UTAH®

CISE

collaborations with Utah colleagues, and

subcontract

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

Group Credits (github.com/PRUNERS)

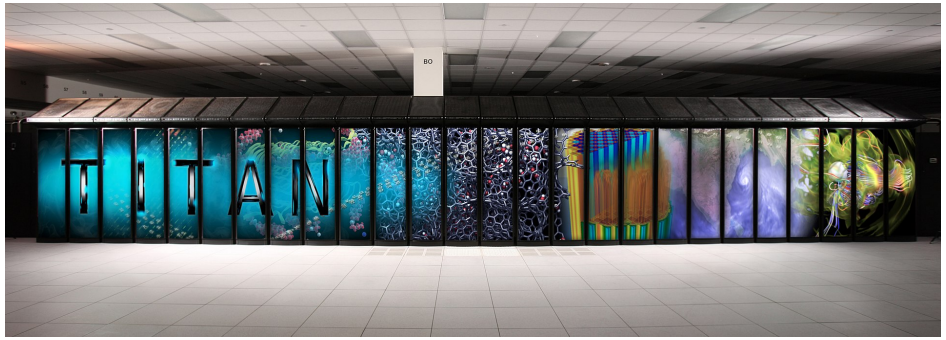
Utah: Ian, Zvonimir, Michael, Geof, Ganesh, Simone

LLNL: Dong, Ignacio, Greg

THE UNIVERSITY OF UTAH®

Growing Correctness Checking Needs in HPC

https://www.nvidia.com/en-us/data-center/tesla-v100/



https://en.wikipedia.org/wiki/Titan_(supercomputer)

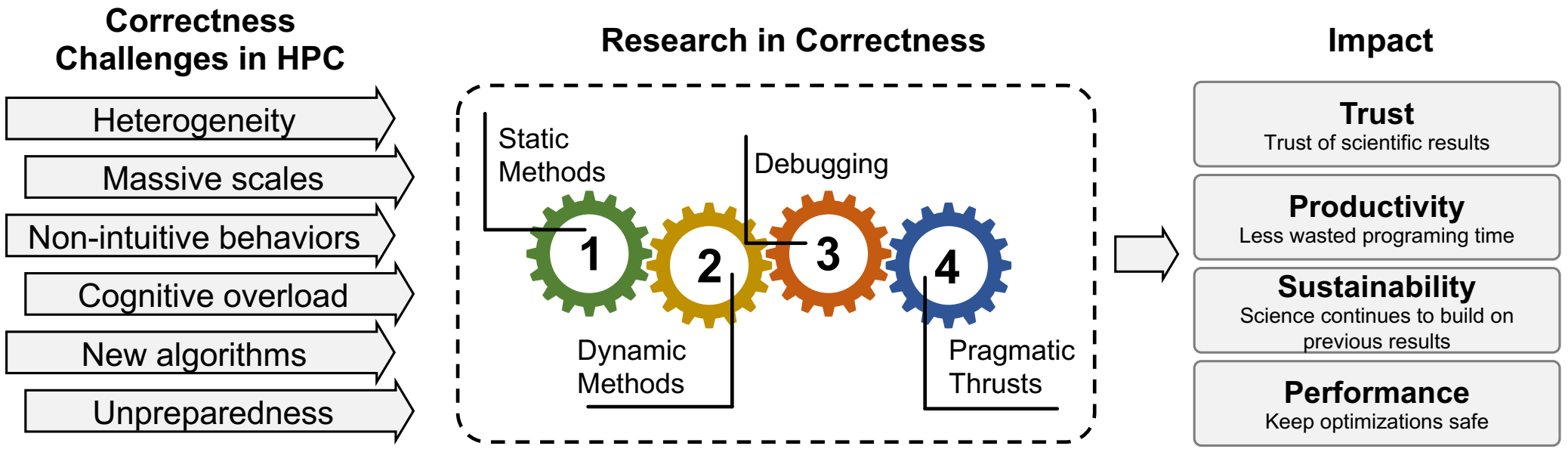
MPI
OpenMP
Pthreads
CUDA
TBB
Charm++
Uintah
Kokkos
RAJA
HYDRA
PETSc
HPX
OCR

By O01326 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45399546

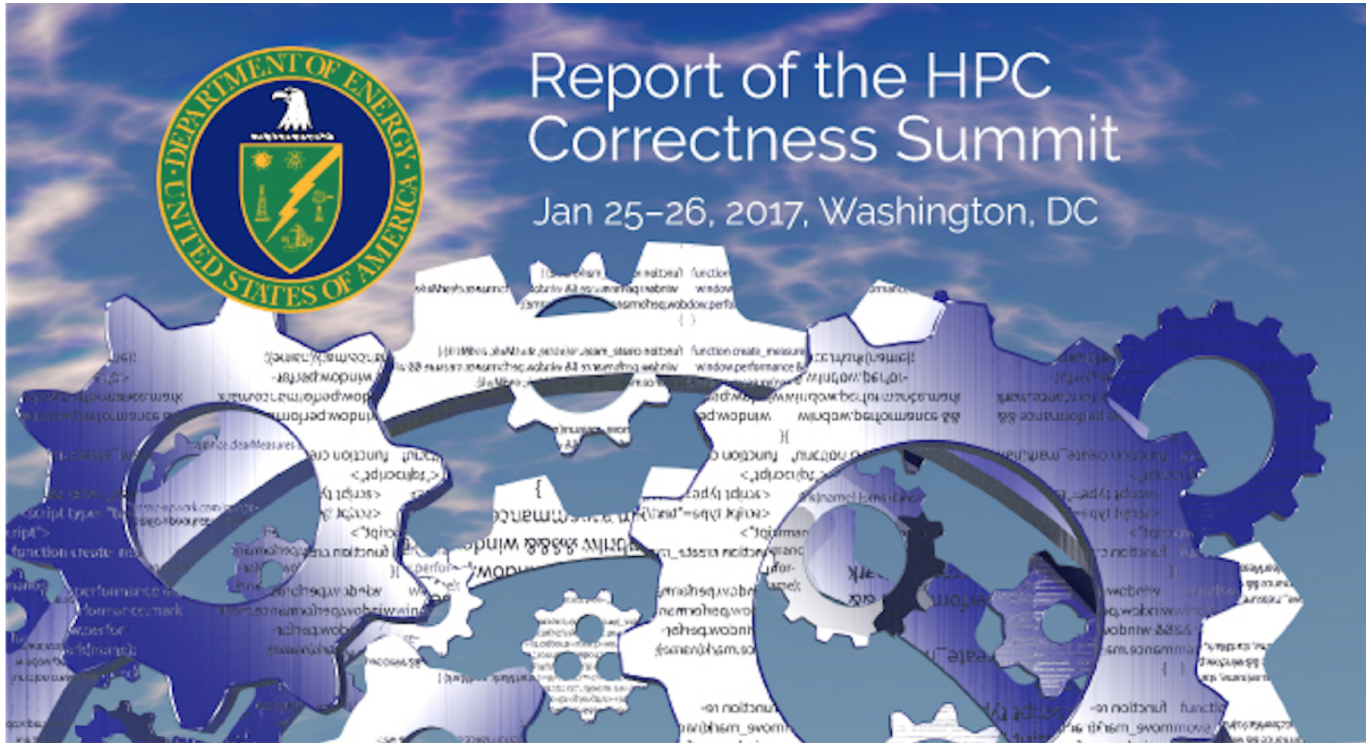
Store.Mellanox.com

THE UNIVERSITY OF UTAH®

Correctness: Central to Productivity



Recommended: DOE Correctness Summit Report



https://science.energy.gov/~/media/ascr/pdf/programdocuments/docs/2017/HPC_Correctness_Report.pdf

55 pages

Product of 2-day workshop organized by Dr. Sonia R. Sachs, Jan 25-26, 2017

HPC Bugs: “War Stories” Abound!

What is actionable?

Why this workshop?

Key Question

How to grow the ***HPC verification community?***

Growing the HPC Verification Community

- Growing through proper ***pedagogy***
  - **Teach students to *think about correctness* (not just performance)**
- Growing a critical mass of ***HPC correctness researchers***
  - **Share benchmarks, bug-repos, tools**
- Have a ***presence in non-HPC conferences***
  - **Develop buy-in and participation from "traditional CS"**

Correctness Projects in HPC at Utah

- Dynamic Analysis Tools for PThreads/MPI programs
- Symbolic Verification Methods for GPU Programs
- Rigorous Floating-Point Error Estimation and Precision Tuning
- Data Race Checking of OpenMP Programs 
- Reproducibility (checking + tuning) FP Compilations 
- System Resilience: Analysis, Protection, Manifesting Failures

This Talk: Three Parts

- Views shaped by research in Formal Methods for:
  - **Formal Hardware Modeling, Self-Timed Circuit Synthesis**
  - **Cache Coherence Protocol Verification**
  - **Processor Verification via Theorem Proving**
  - **Studying Weak Memory Consistency Models**
  - **Model-Checking**
- Two recent projects at Utah, including key collaborations
  - Archer and Sword: Two State-of-the-Art Race Checkers
  - FLiT: Root-causing Non-reproducible FP Results
- Conclusions

Part-1: Observations From a Formal Methods Perspective

Role of Pedagogy: Emphasize Enduring Ideas

- One Example
  - from Hoare's transistor mode (~1989)
  - PL and type-checking (co-/contra-variance)
  - …

Monotonicity

$$a \sqsubseteq b \quad \Rightarrow \quad C[a] \sqsubseteq C[b]$$

Congruence ("equals for equals" in a context) is a special case

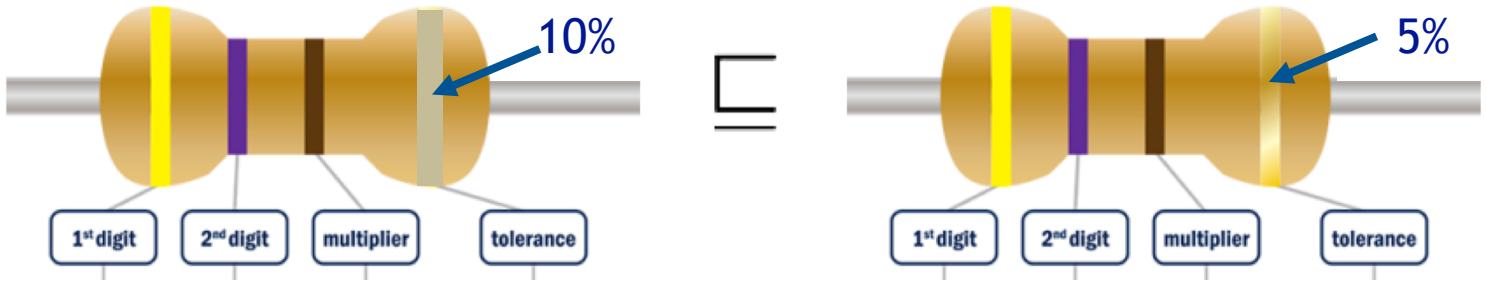
$$a \equiv b \quad \Rightarrow \quad C[a] \equiv C[b]$$

Role of Pedagogy: Emphasize Enduring Ideas

- One Example
  - from Hoare's transistor mode (~1989)
  - PL and type-checking (co-/contra-variance)
  - …

Monotonicity

$$a \sqsubseteq b \;\Rightarrow\; C[a] \sqsubseteq C[b]$$



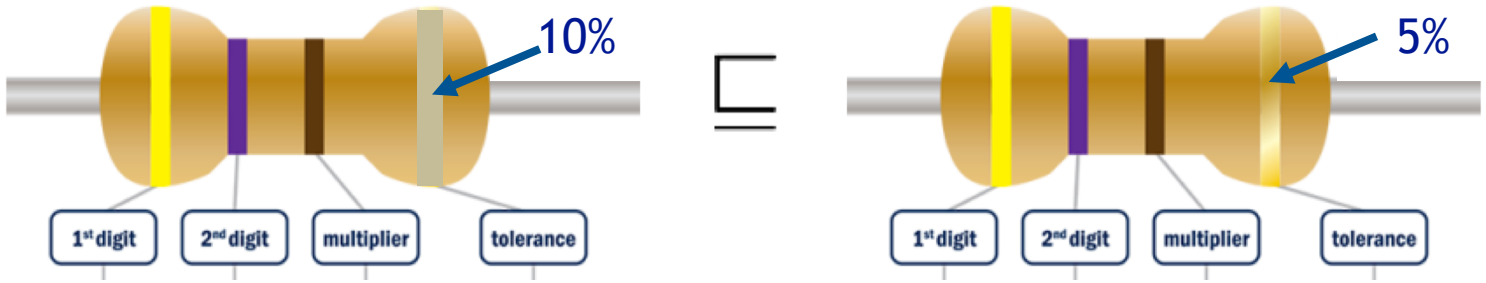
Role of Pedagogy: Emphasize Enduring Ideas

- One Example
  - from Hoare's transistor mode (~1989)
  - PL and type-checking (co-/contra-variance)
  - …

Monotonicity

$$a \sqsubseteq b \;\Rightarrow\; C[a] \sqsubseteq C[b]$$

Local improvements result in Global improvements



Non-monotonic behaviors are hard to debug

Let us see a few practical examples

Example-1: When in Doubt, Add Parentheses

- Else, the code does not port across languages

- Specifically,
  - Exponentiation can be left- or right-associative
    - **2 ** 2 ** 3** can be **64 or 256**
  - Negation-first or exponentiation-first is undefined
    - **-2 ** 2** can be **4 or -4**

- Is the *addition* of parentheses safe in all situations?

Not in Excel!

- (4/3-1)*3-1 produces 0
- ((4/3-1)*3-1) produces -2.22045E-16
  - They are bit-wise different too!
    - Branching behavior may be affected
- Adding 0 has the same effect as outer “(…)”

Example-2: When in Doubt, Add Buffering

- Buffering helps avoid head-to-head deadlocks
- Can improve performance

- Is buffer *addition* safe always?

Not in MPI!

- MPI deadlocks can increase with buffering

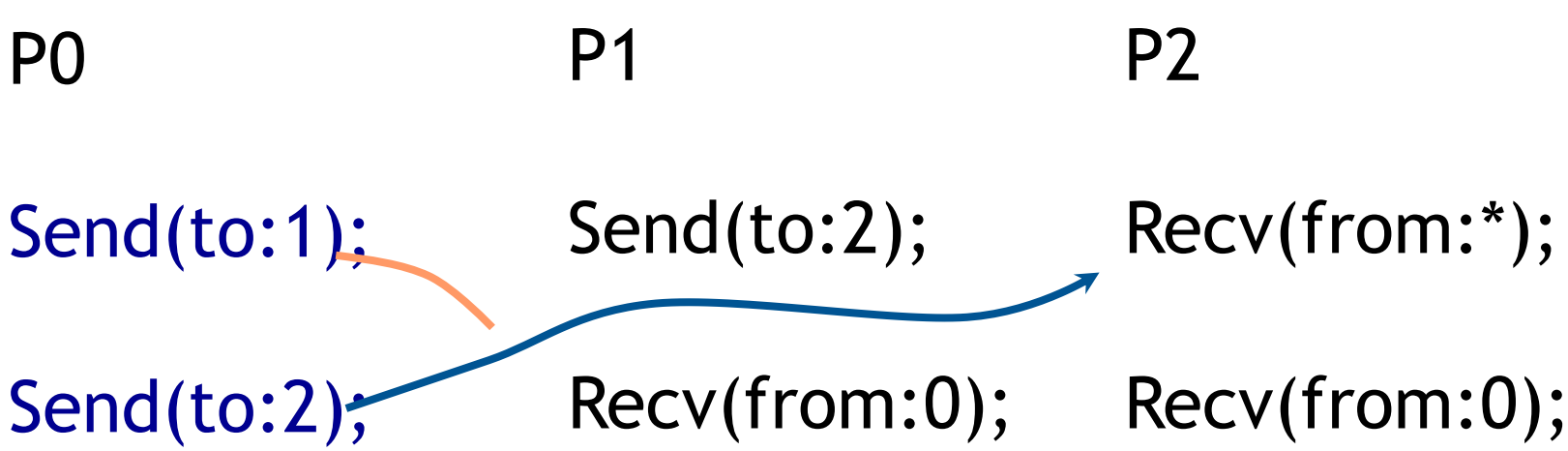
Slack-Inelastic Behavior (Manohar/Martin)

P0	P1	P2
Send(to:1);	Send(to:2);	Recv(from:*);
Send(to:2);	Recv(from:0);	Recv(from:0);

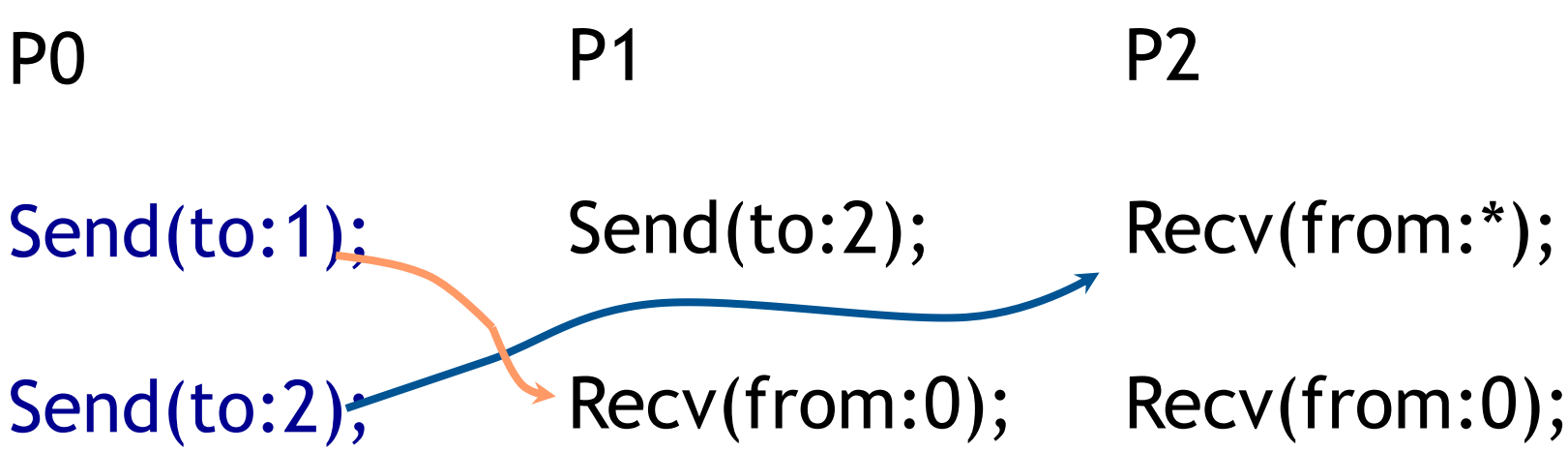
Slack-Inelastic Behavior (Manohar/Martin)

P0	P1	P2
Send(to:1);	Send(to:2);	Recv(from:*);
Send(to:2);	Recv(from:0);	Recv(from:0);

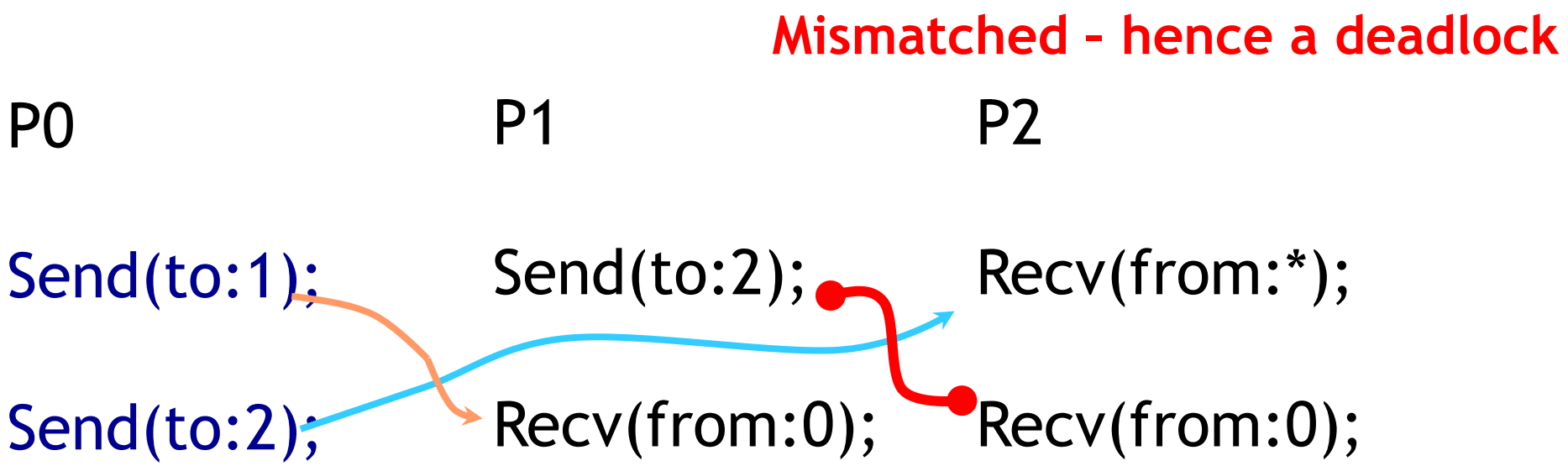
Slack-Inelastic Behavior (Manohar/Martin)



Slack-Inelastic Behavior (Manohar/Martin)

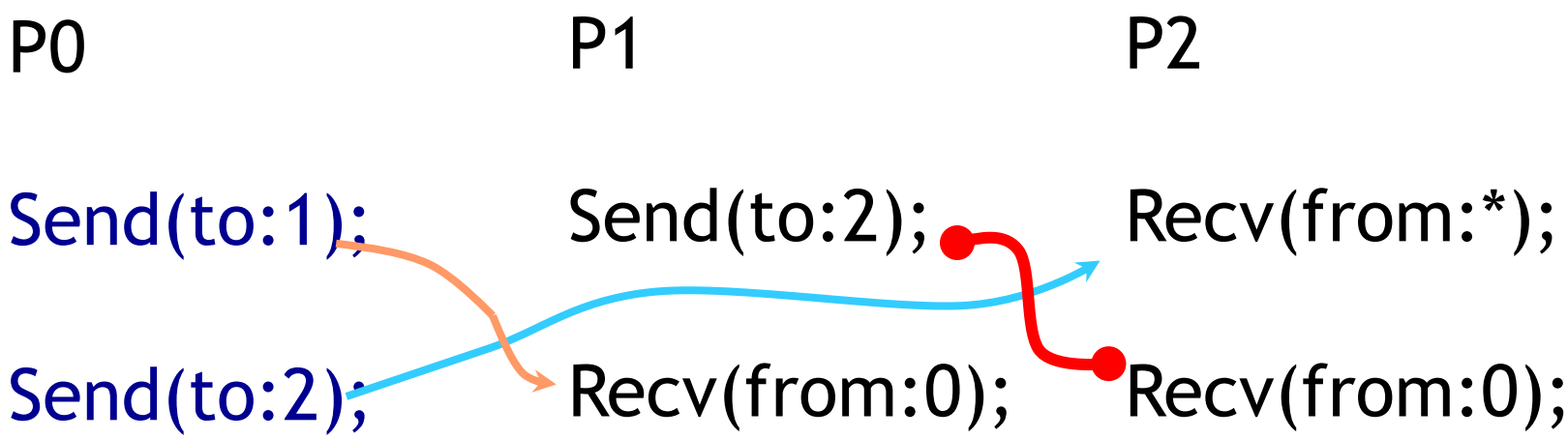


Slack-Inelastic Behavior (Manohar/Martin)



Slack-Inelastic Behavior (Manohar/Martin)

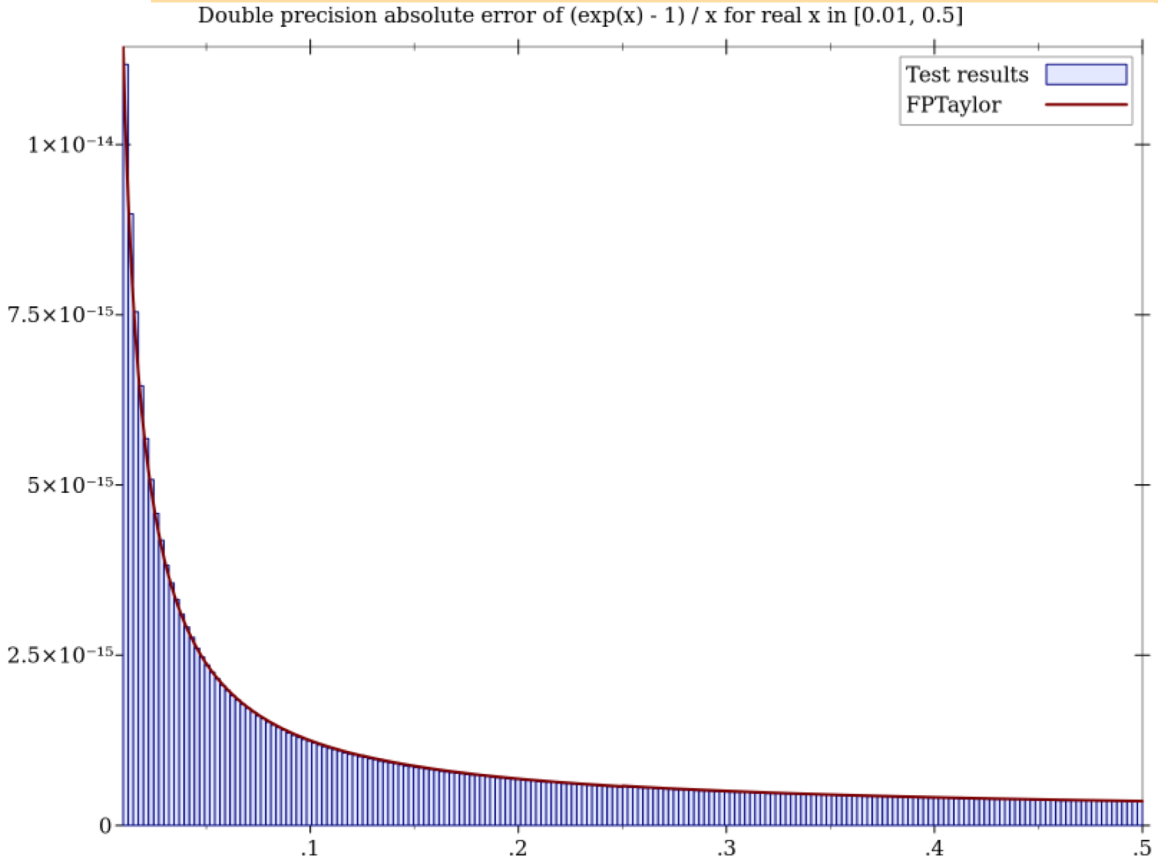
Mismatched - hence a deadlock



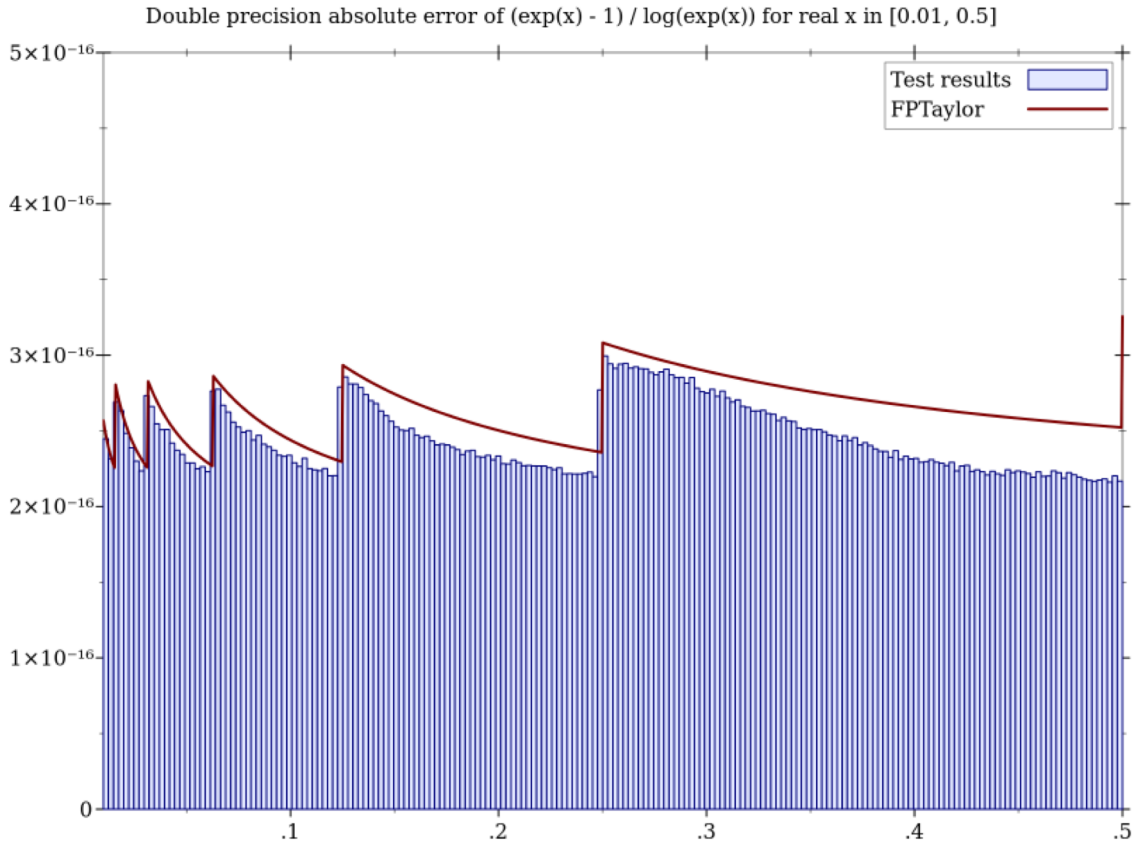
Deadlocks disappear under MPI EAGER LIMIT of zero !!

Example-3: When in Doubt, Increase FP Precision

Surprise! Non-Monotonic Behavior of FP

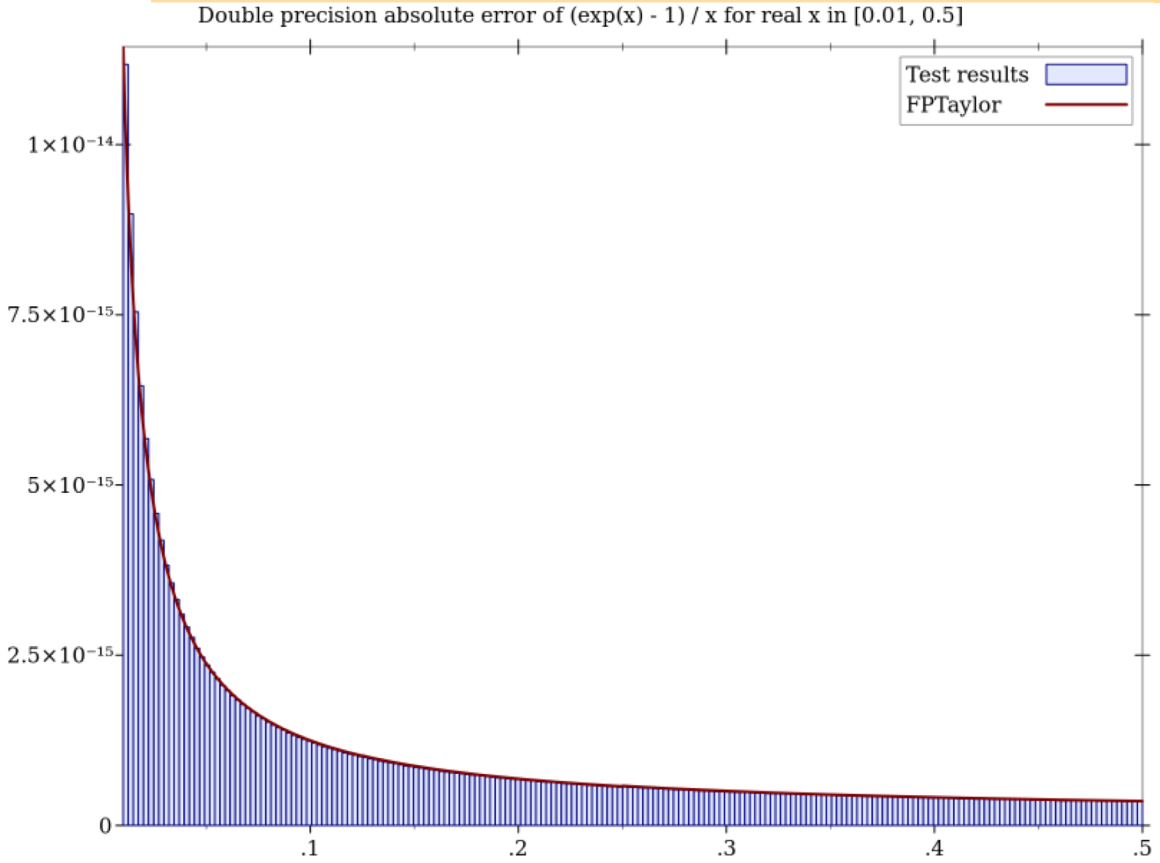


$(e^x - 1)/x$

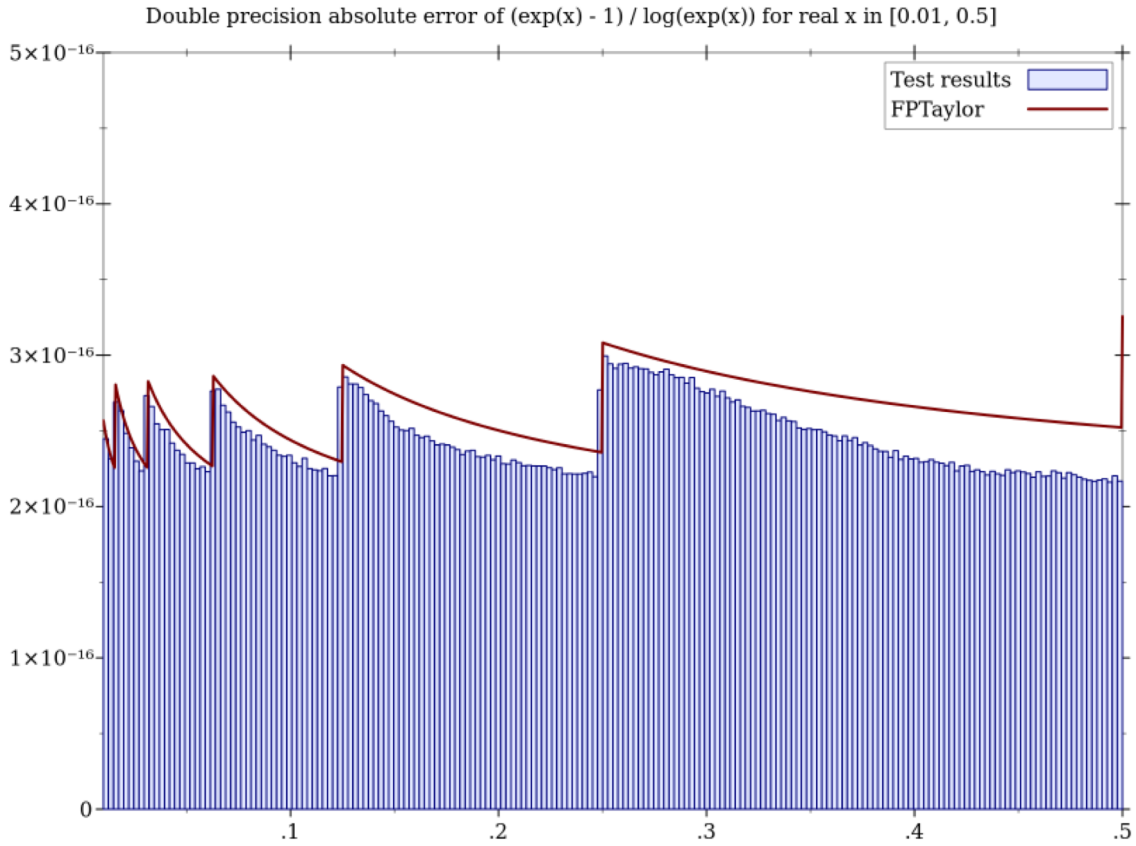


$(e^x - 1)/(log(e^x))$

Surprise! Non-Monotonic Behavior of FP



$(e^x - 1)/x$



$(e^x - 1)/(log(e^x))$

RHS is better for x in some ranges!

Another Non-Monotonic Behavior of FP

- Example from Baker et al's group
  - Climate code ***failed*** when FMA was ***introduced***

What are Formal Methods?

What are Formal Methods (for System Design)

- Those methods that help attain higher standards in
  - Specification
  - Testing
  - Verification
  - Synthesis

The "Higher Standards" through the use of decision procedures (SMT), program analysis methods, model-checking, …

Examples of Formal Methods

Our discussion of monotonicity

An example of what formal thinking helps obtain

Q: Example of Something Practical Through FM?

Q: Example of Something Practical Through FM?

A: Actionable language standards!

An example from OMP 5.0 now follows

English describing OpenMP's Happens-Before and Acq/Rel

1.4.5 Flush Synchronization and *Happens Before* 1.4.6 OpenMP Memory Consistency

If a thread has performed a write to its temporary view of a shared variable since its last strong flush of that variable, then when it executes another strong flush of the variable, the strong flush does not complete until the value of the variable has been written to the variable in memory. If a thread performs multiple writes to the same variable between two strong flushes of that variable, the strong flush ensures that the value of the last write is written to the variable in memory. A strong flush of a variable executed by a thread also causes its temporary view of the variable to be discarded, so that if its next memory operation for that variable is a read, then the thread will read from memory and capture the value in its temporary view. When a thread executes a strong flush, no later memory operation by that thread for a variable involved in that strong flush is allowed to start until the strong flush completes. The completion of a strong flush executed by a thread is defined as the point at which all writes to the flush-set performed by the thread before the strong flush are visible in memory to all other threads, and at which that thread's temporary view of the flush-set is discarded.

Producer/Consumer Example From a Tutorial

Atomics and synchronization flags

```
int main()
{   double *A, sum, runtime;
    int numthreads, flag = 0, flg_tmp;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
      #pragma omp section
      {  fill_rand(N, A);
         #pragma omp flush
         #pragma omp atomic write
                flag = 1;
         #pragma omp flush (flag)
      }
      #pragma omp section
      {  while (1){
            #pragma omp flush(flag)
            #pragma omp atomic read
                  flg_tmp= flag;
             if (flg_tmp==1) break;
          }
          #pragma omp flush
          sum = Sum_array(N, A);
      }
    }
}
```

This program is truly race free … the reads and writes of flag are protected so the two threads cannot conflict

Still painful and error prone due to all of the flushes that are required

Nothing “wrong”

It just is over-synchronized

Another Expert's Opinion

- *The #pragma omp flush (flag) directives are actually unnecessary, because a strong flush on flag is implied for the atomic write and read operations.*
- *The consumer thread should therefore eventually break out of the while loop*

Another Expert's Opinion

- *The #pragma omp flush (flag) directives are actually unnecessary, because a strong flush on flag is implied for the atomic write and read operations.*
- *The consumer thread should therefore eventually break out of the while loop.*

- We are working on formalizing OMP 5.0 concurrency
  - Work has just started in collaboration with Steve Siegel
- Siegel's CIVL model checker expected to be the "expert"
  - Helping answer queries about proposed litmus tests

Q: “Too Many Bug-types”: What to Focus on?

Q: “Too Many Bug-types”: What to Focus on?

Many “weird bug types abound” in HPC

Cannot be fighting ***every*** battle!

(-1 % N) = N-1 in Fortran, but -1 in C ➔ lead to a deadlock!

Answer: Prioritize Based On Principal Governing Semantics

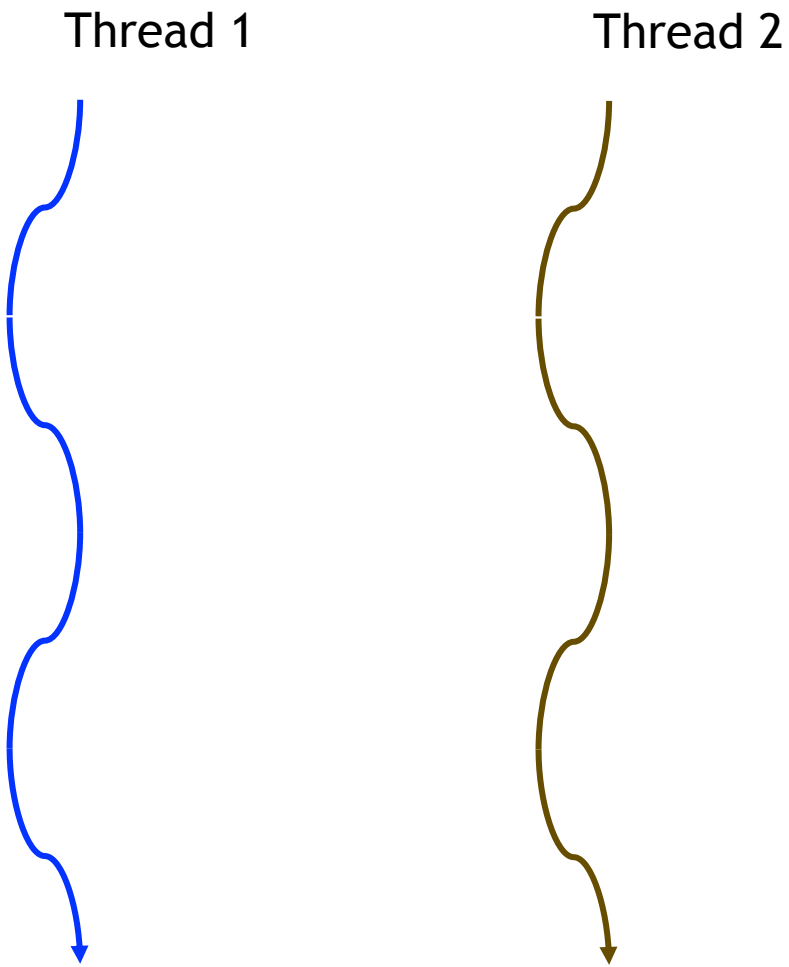
- Examples:
  - Processor Design
    - FPU Correctness, Cache Coherence
  - Message-Passing (MPI) Design
    - Happens-Before Model, Message Matching, Non-Overtaking, …
  - Distributed Systems
    - Linearizability
  - **Shared Memory Program Design**
    - **Data Races**

Focus of the Rest of This Talk

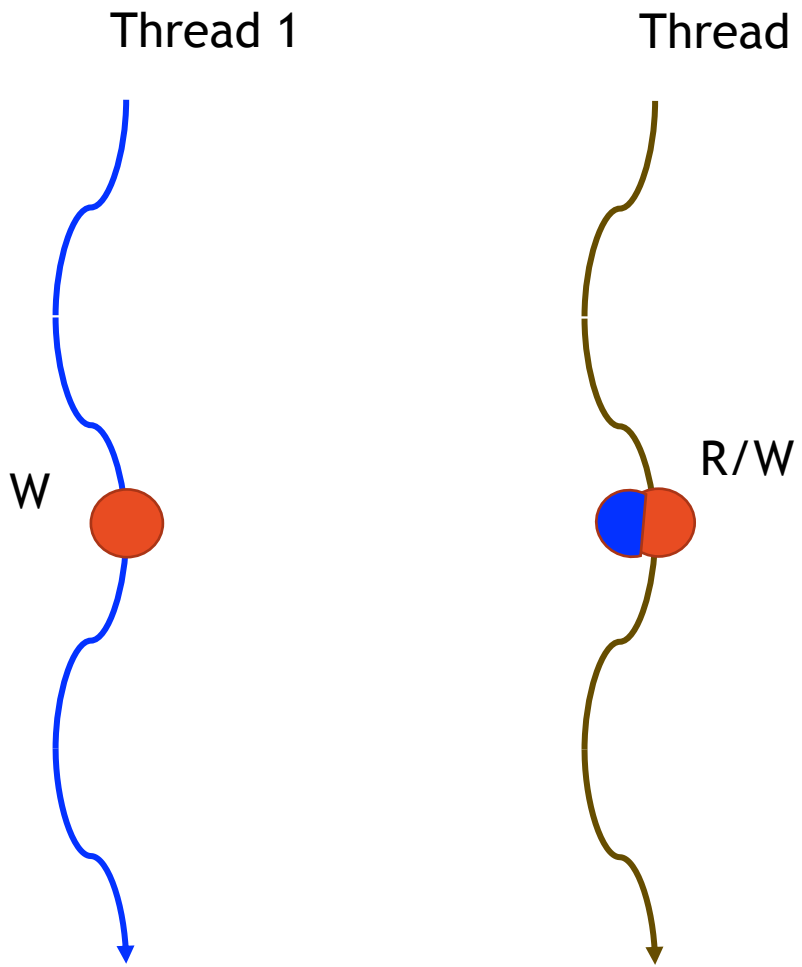
- Two race checkers
  - Archer (IPDPS'16)
  - Sword (IPDPS'18)
    - Commonality of influences
- A Floating-Point Non-Repro Root-Causing + Tuning Tool
  - FLiT (under submission)

What is a data race?

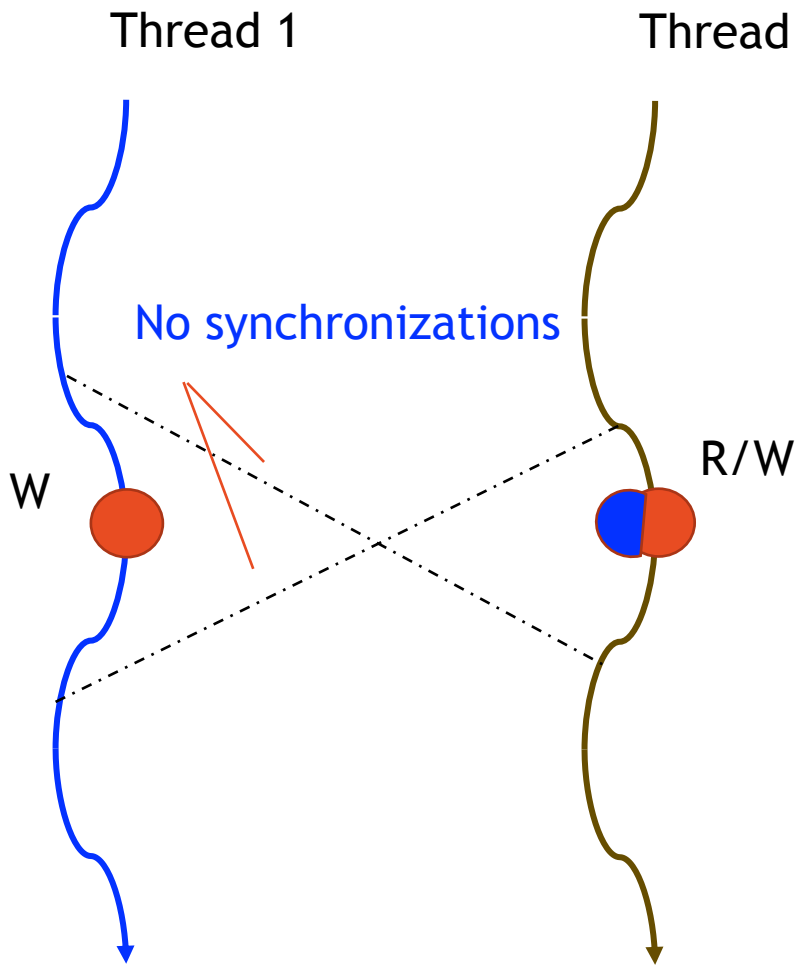
What is a data race?



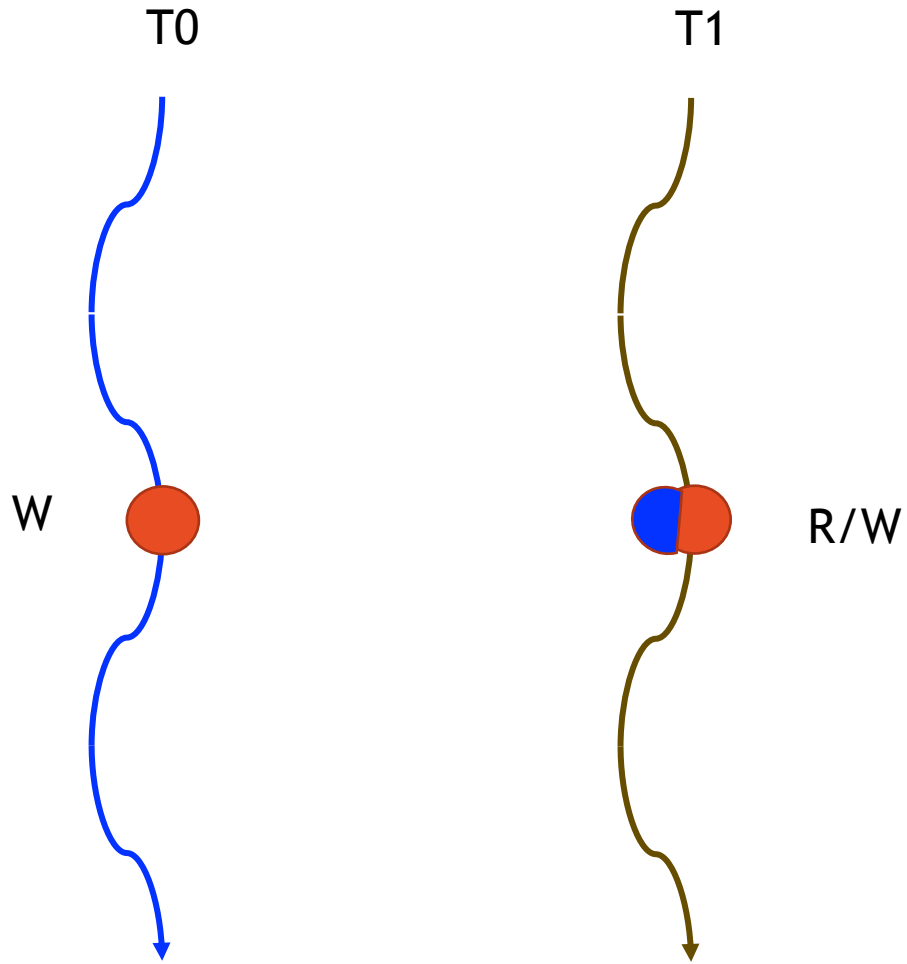
What is a data race?



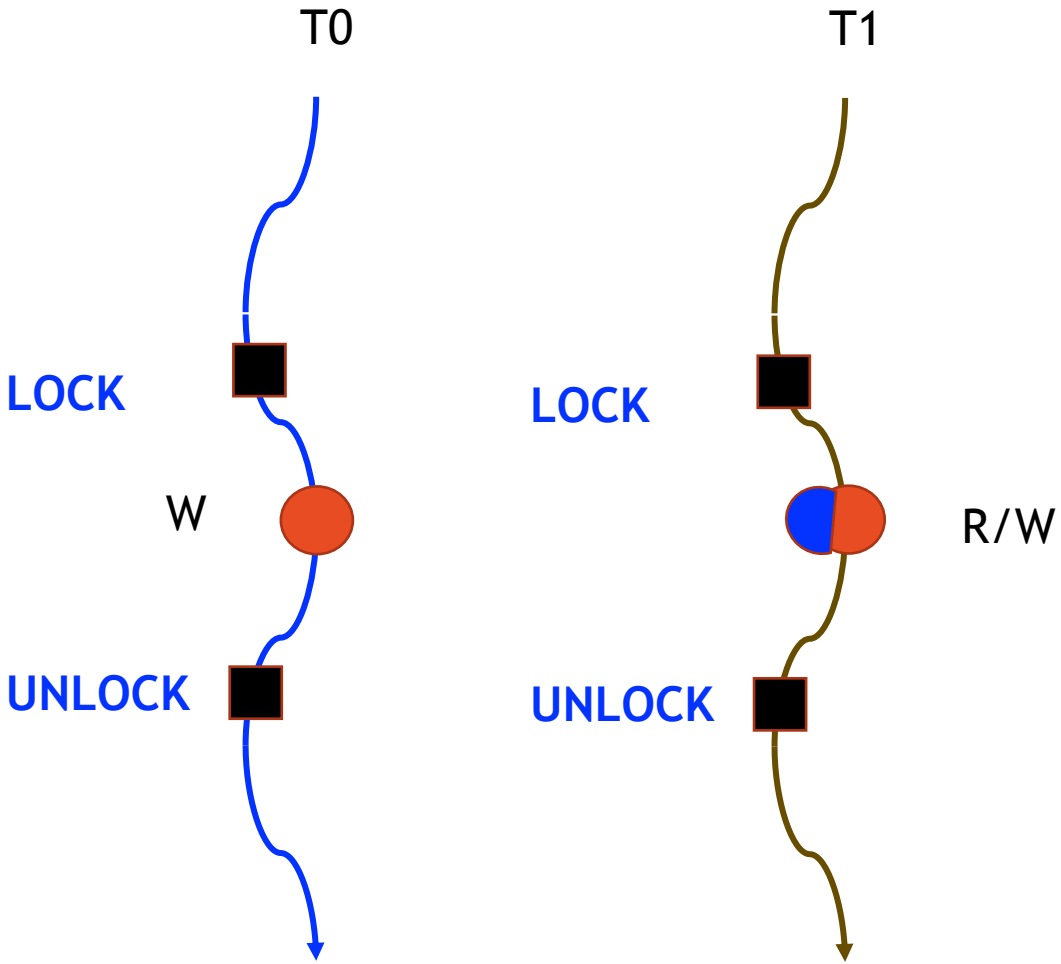
What is a data race?



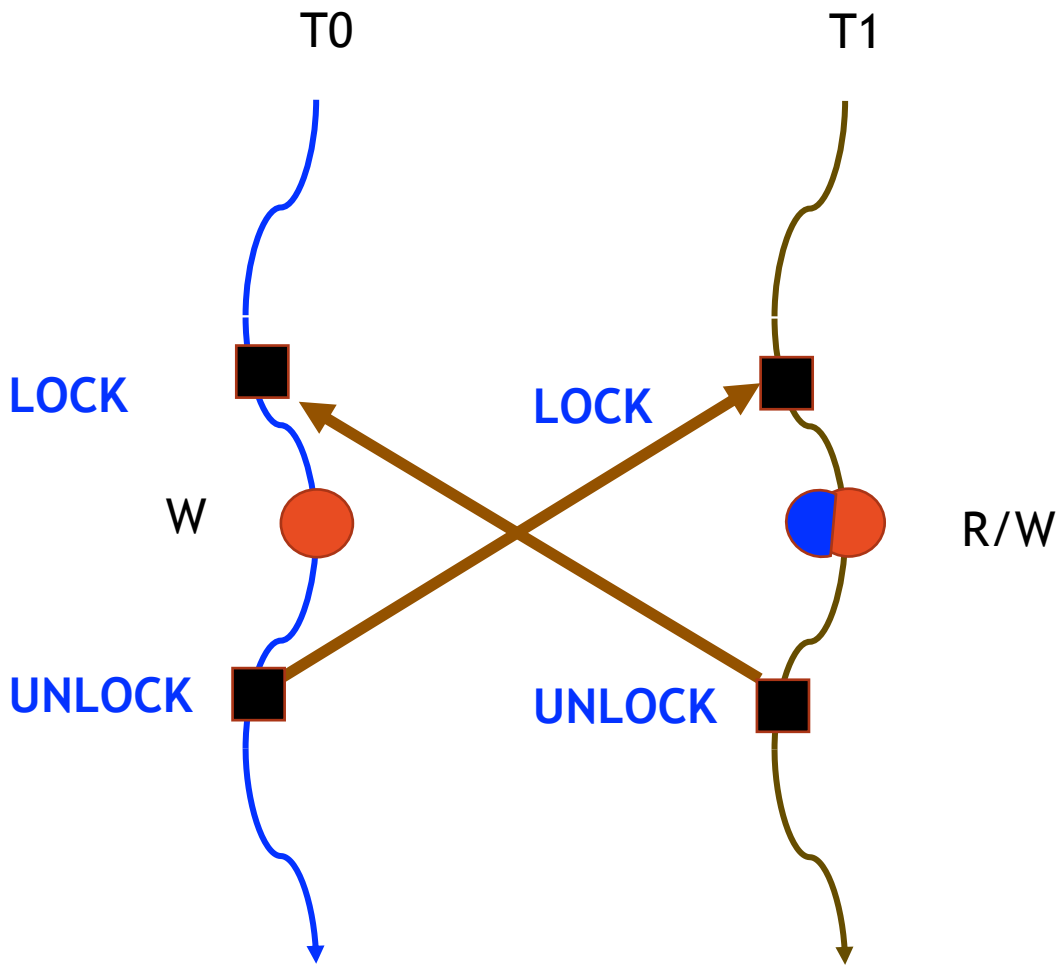
One way to eliminate this race



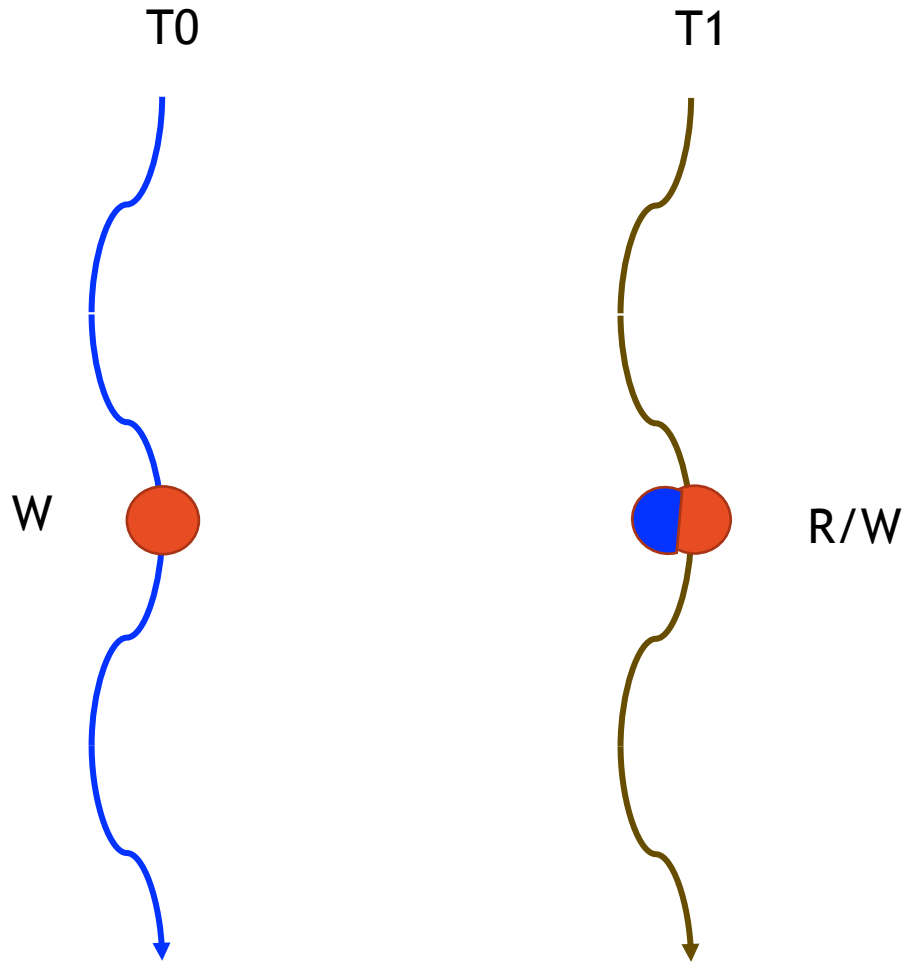
One way to eliminate this race



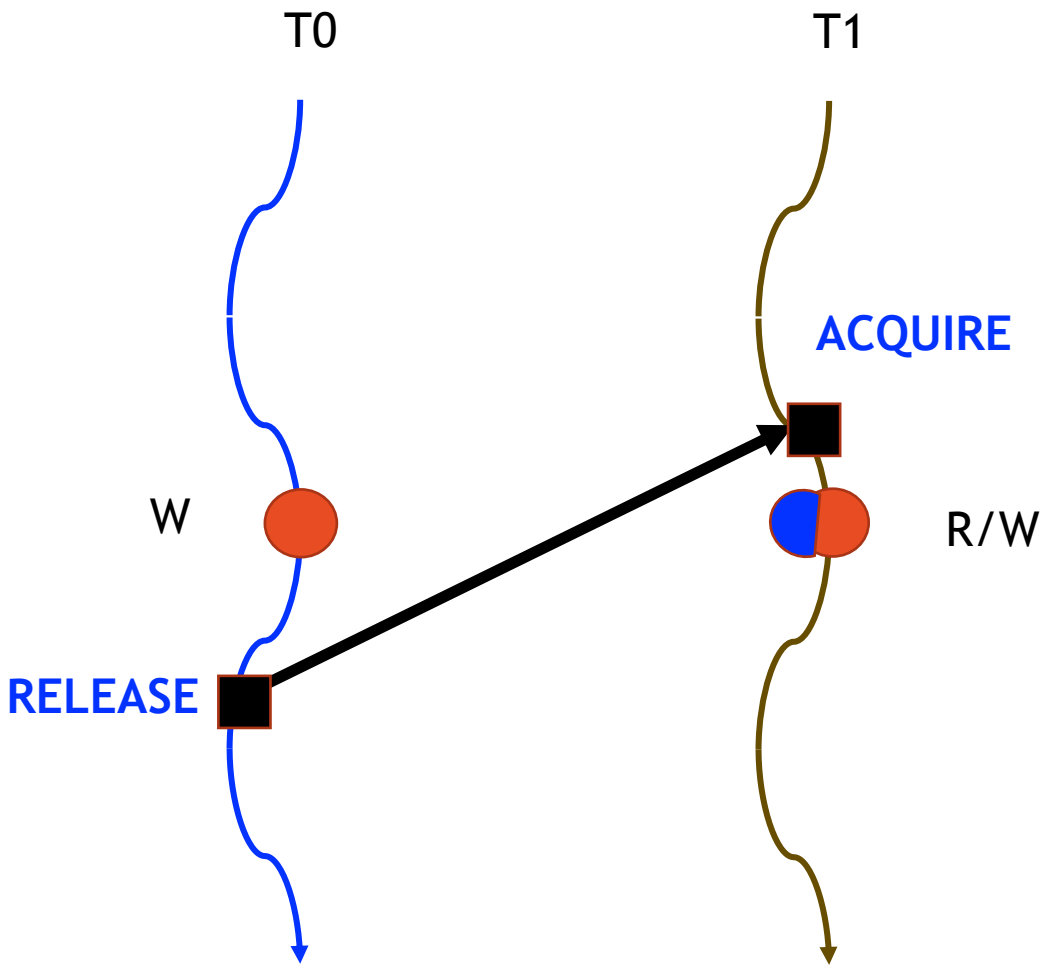
One way to eliminate this race



Another way



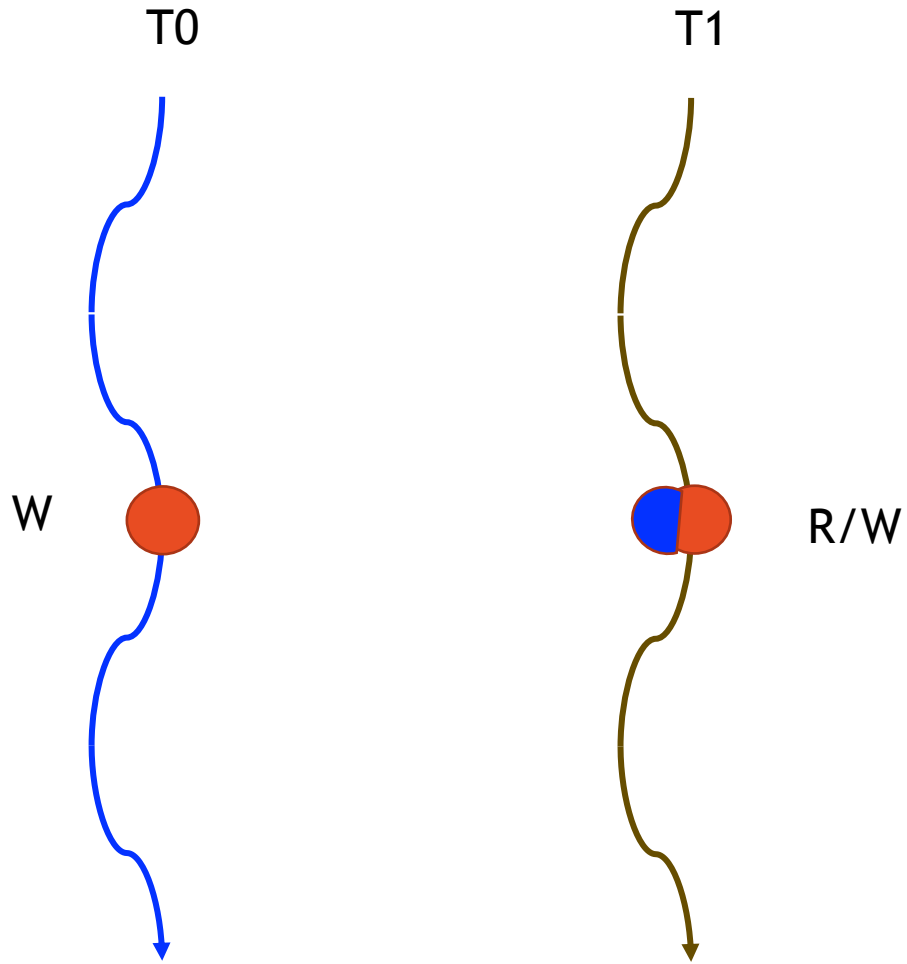
Another way



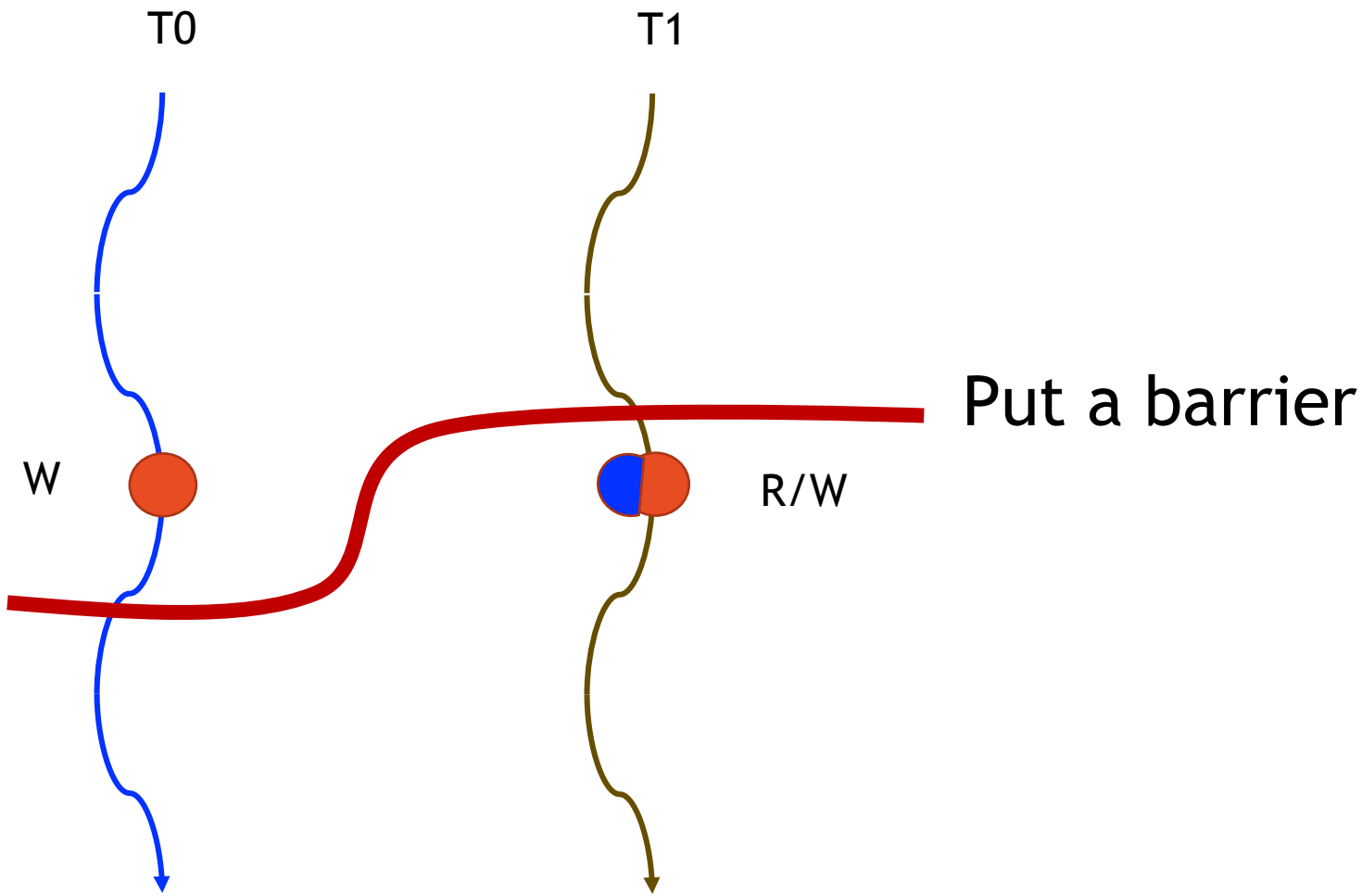
Signal using `special' variables

- Java 'volatile' annotations
  - NOT C 'volatiles' ☹
- C++11 'atomic' annotations

A third way

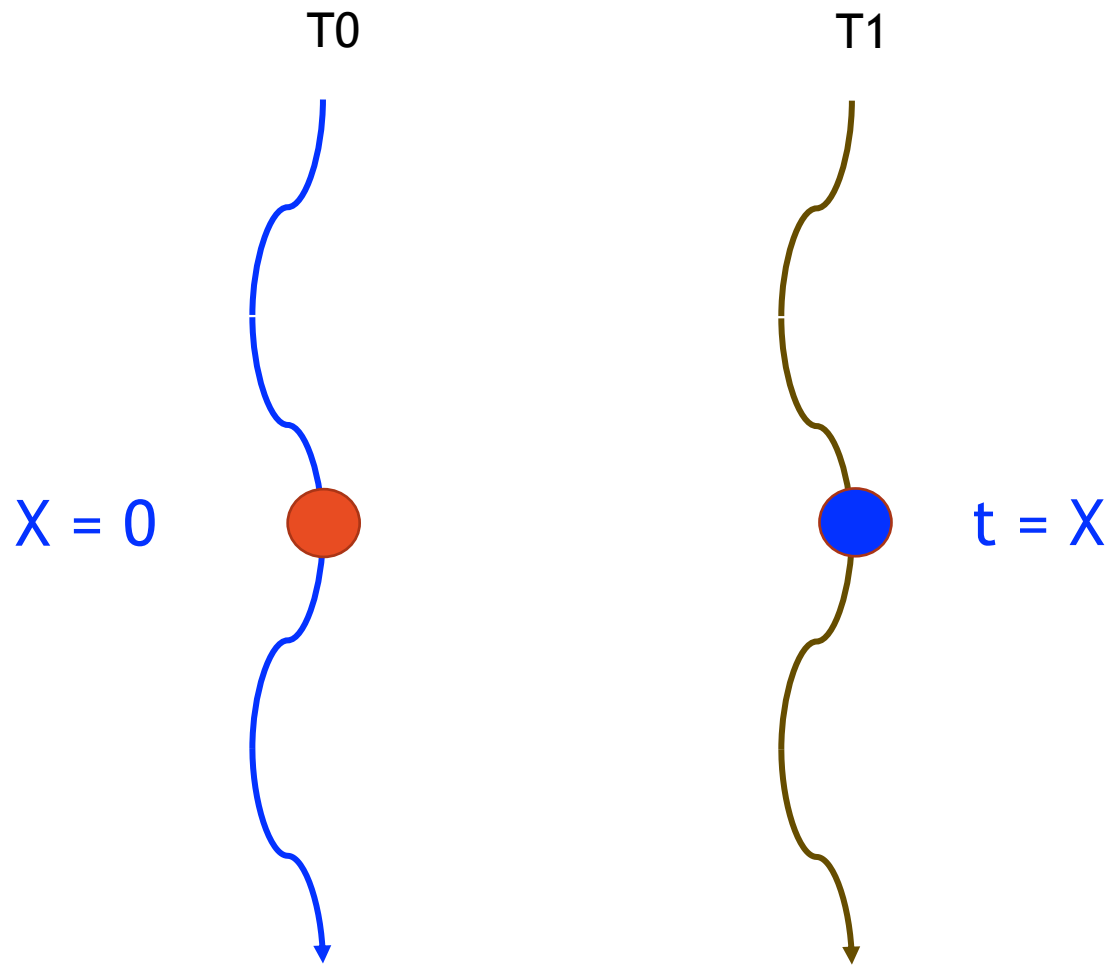


A third way



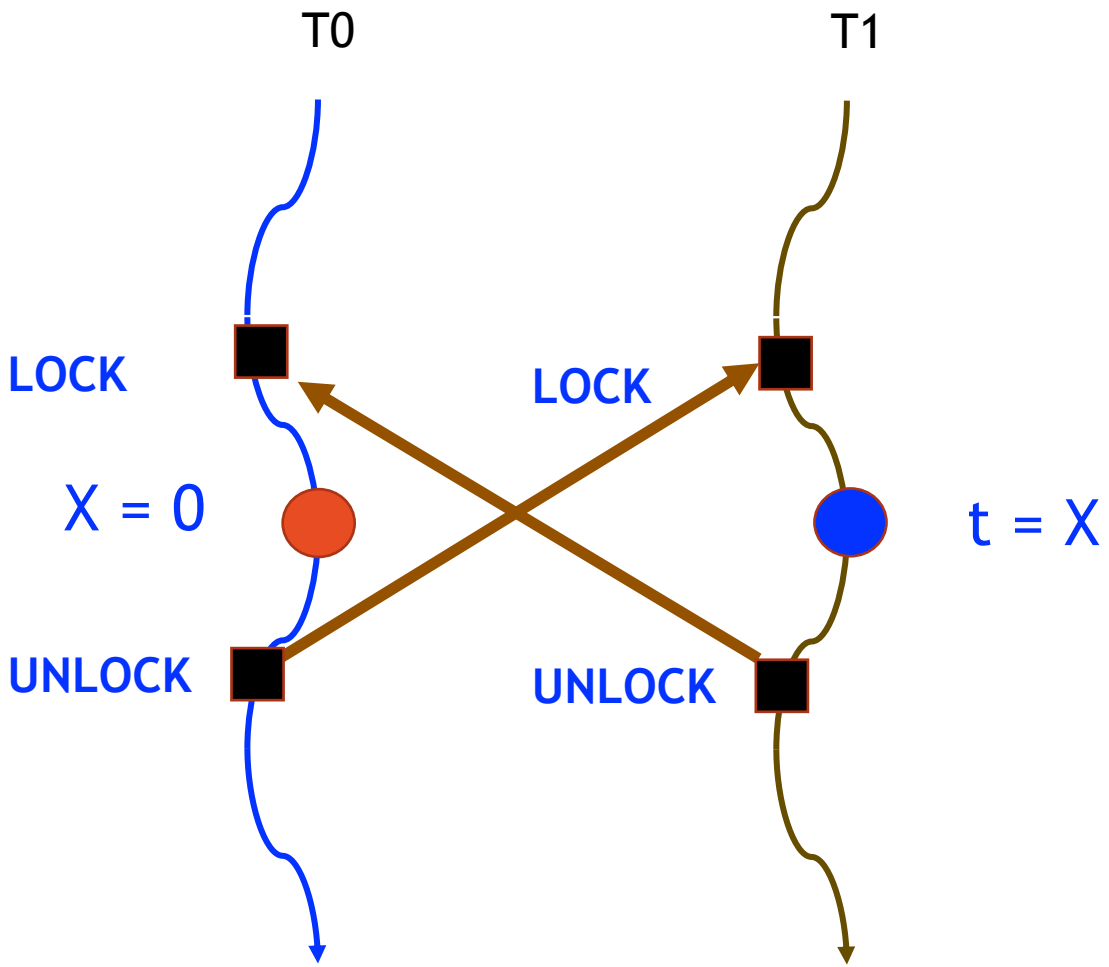
Why eliminate races?

Popular answer: "avoid nondeterminism"

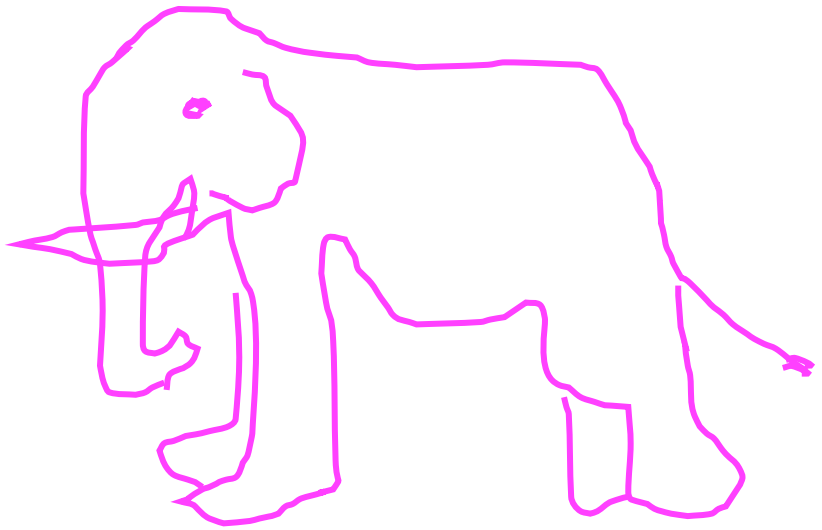


Unclear what “nondeterminism” means..

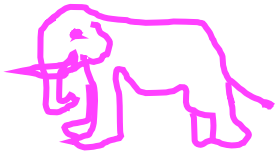
Execution Order is Still Nondeterministic



More relevant: Avoid “pink elephants” ☺



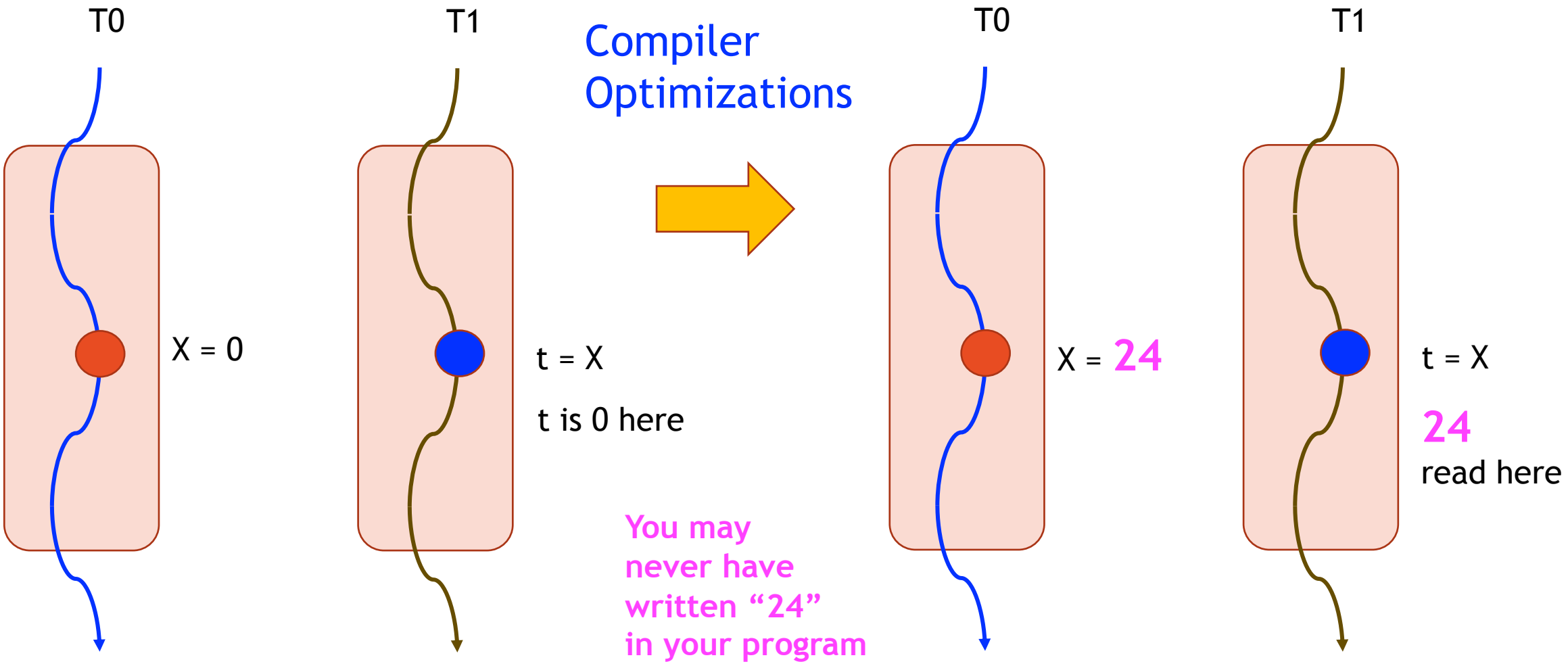
More relevant: Avoid “pink elephants” ☺



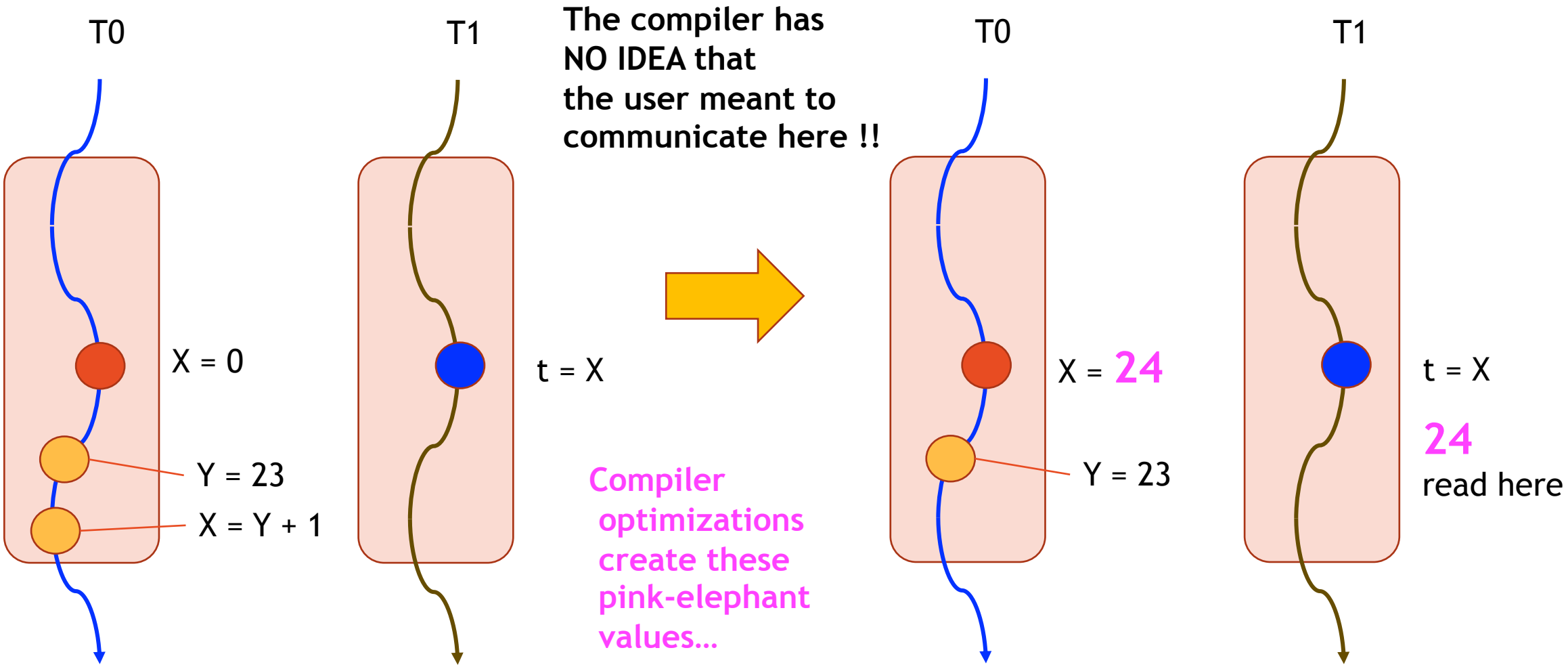
Pink elephant (Sutter) : “A value you never wrote but managed to read”

Aka ”out of thin air” value

The birth of a pink elephant…

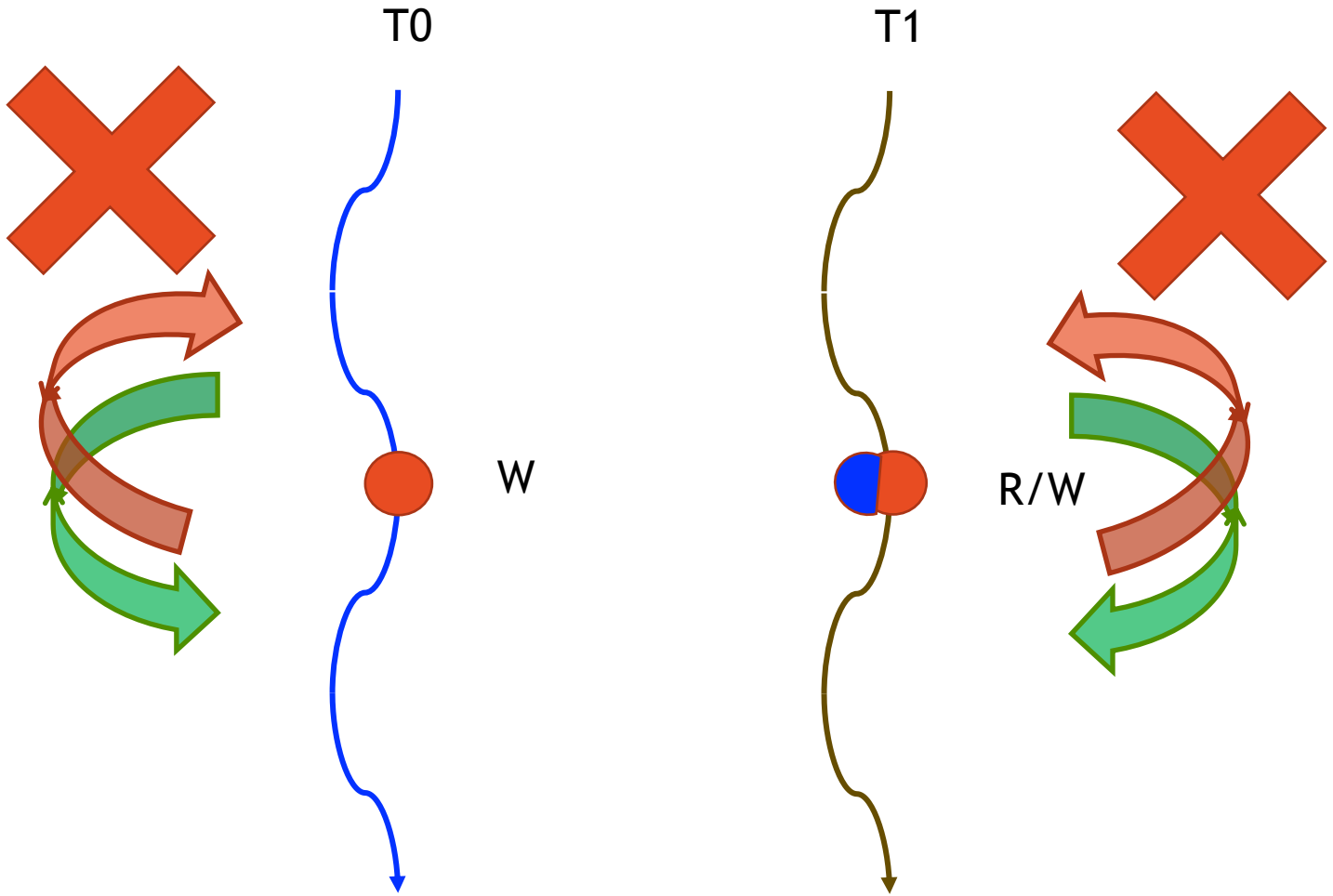


Details of how a pink elephant is made!



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)*

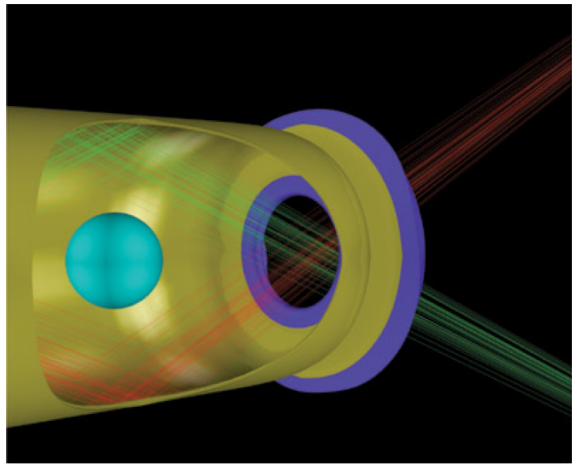
Exploding a myth!

There is no such thing as a benign race !!

THE UNIVERSITY OF UTAH®

Reality: Pink Elephant Strikes Again!

- 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!

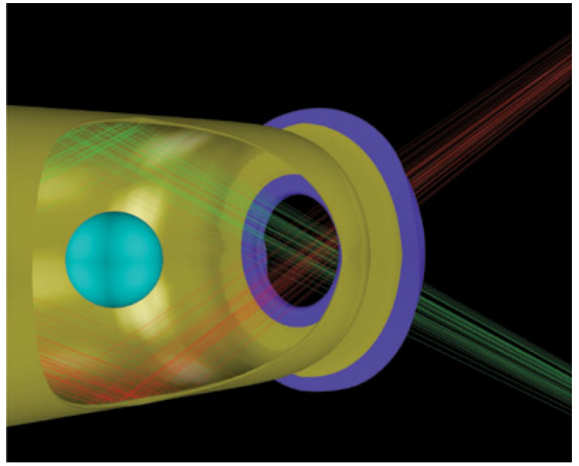
Archer [Atzeni et al., IPDPS'16] – in production use at LLNL

Part of the "PRUNERS" tool suite

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

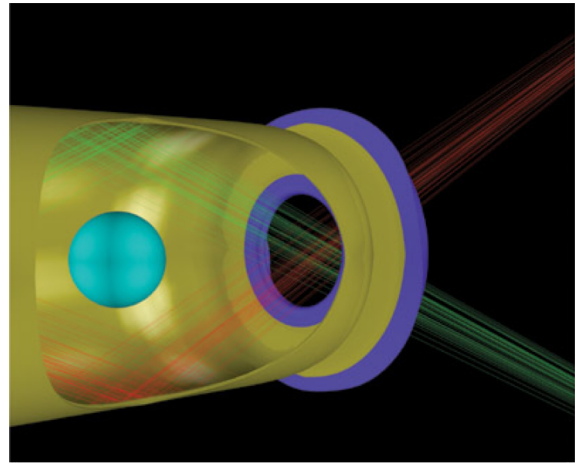
Archer's "find"

Two threads writing `0`
to the same location
without synchronization



Archer's "find"

Two threads writing 0
to the same location
without synchronization

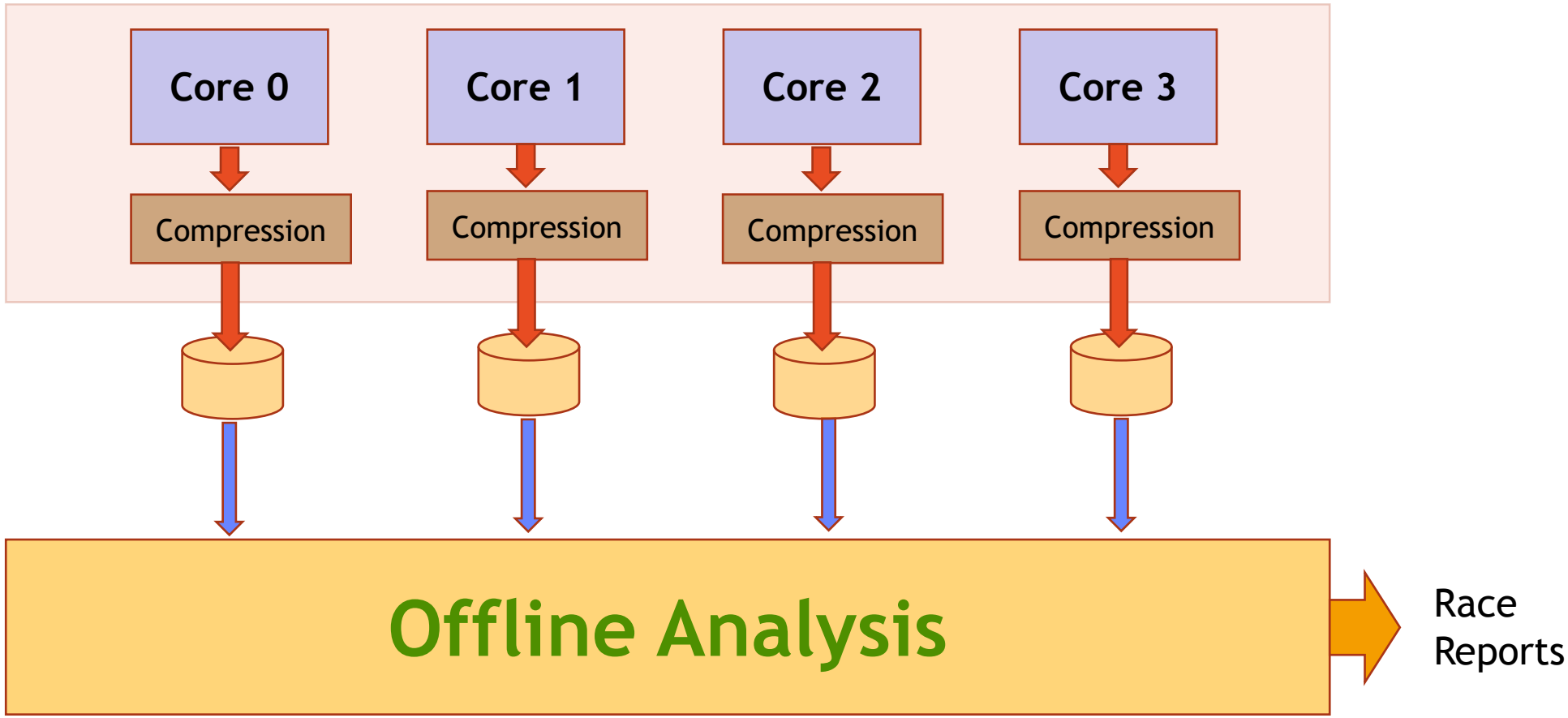


Did we live “happily ever after?”

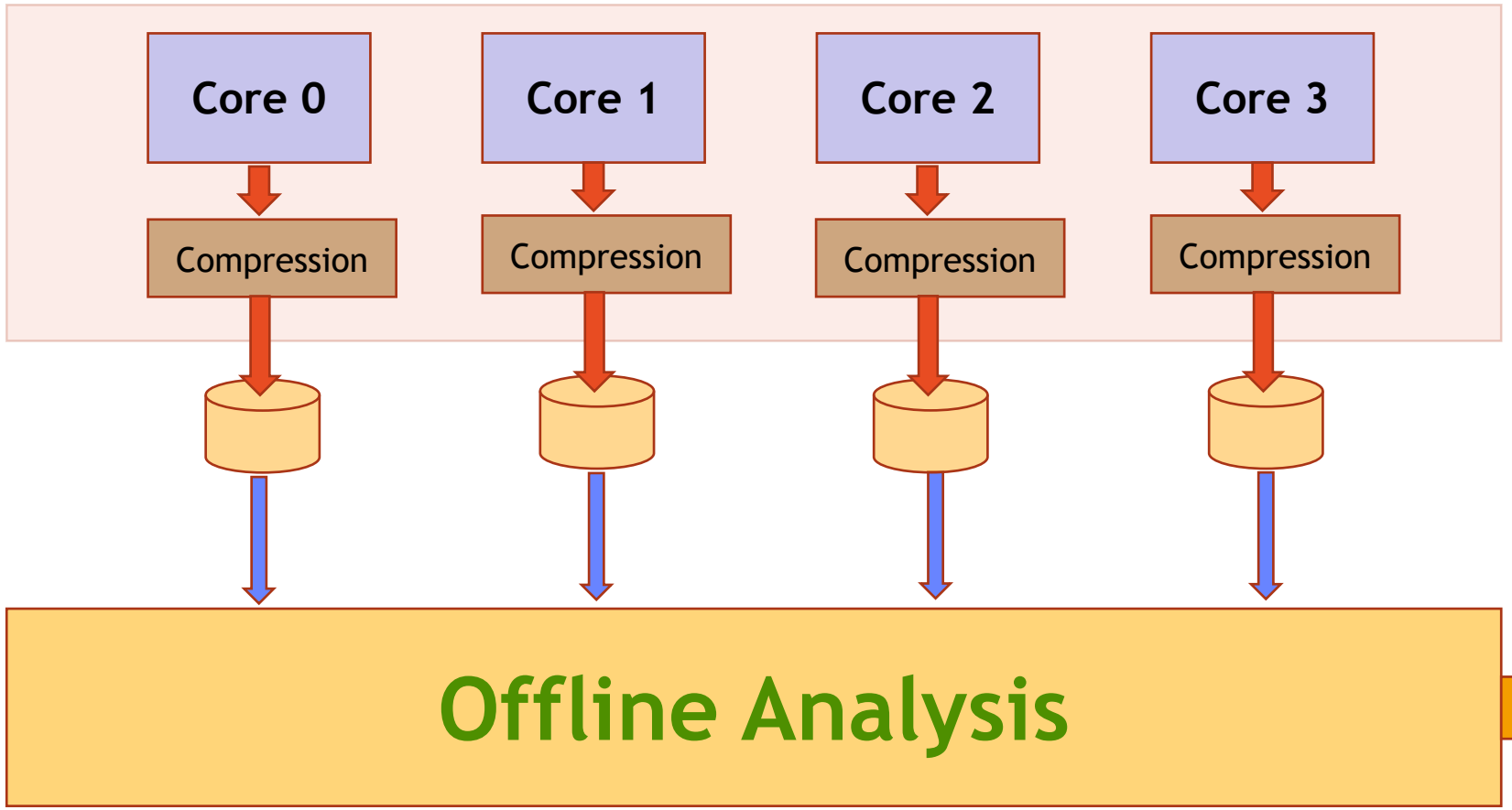
No ☹

Archer has “memory-outs”; also misses races

Sword [IPDPS'18]: Offline Checking



Sword [IPDPS'18]: Offline Checking

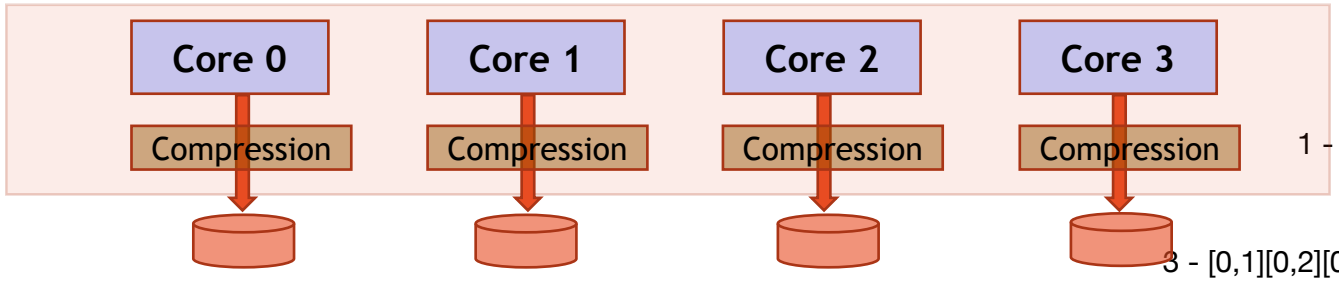


Memory Access Collection via Tsan Instrumentation

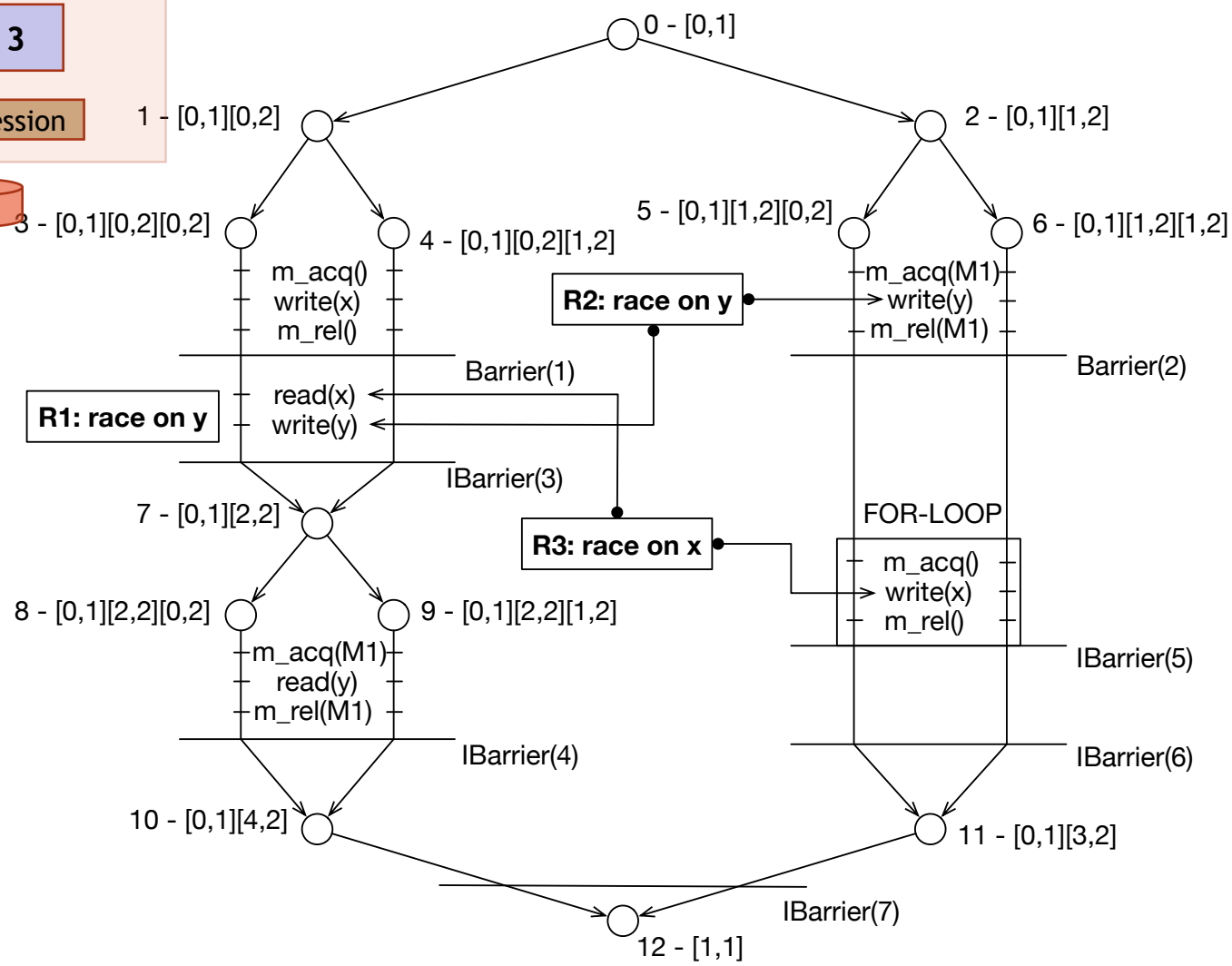
OMP Call Logging via OMPT Callbacks

Race Reports

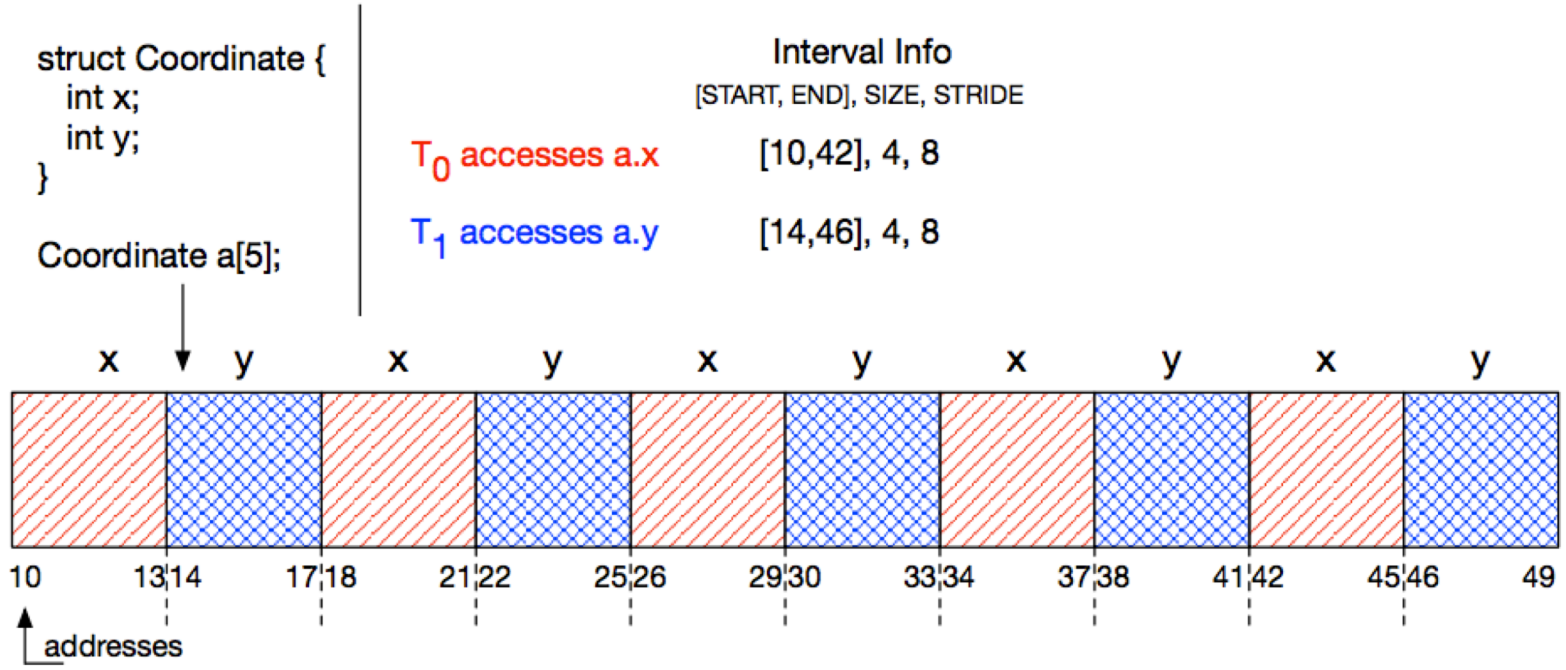
Offline Synchronization Recovery and Analysis



OpSem (HIPS'18)



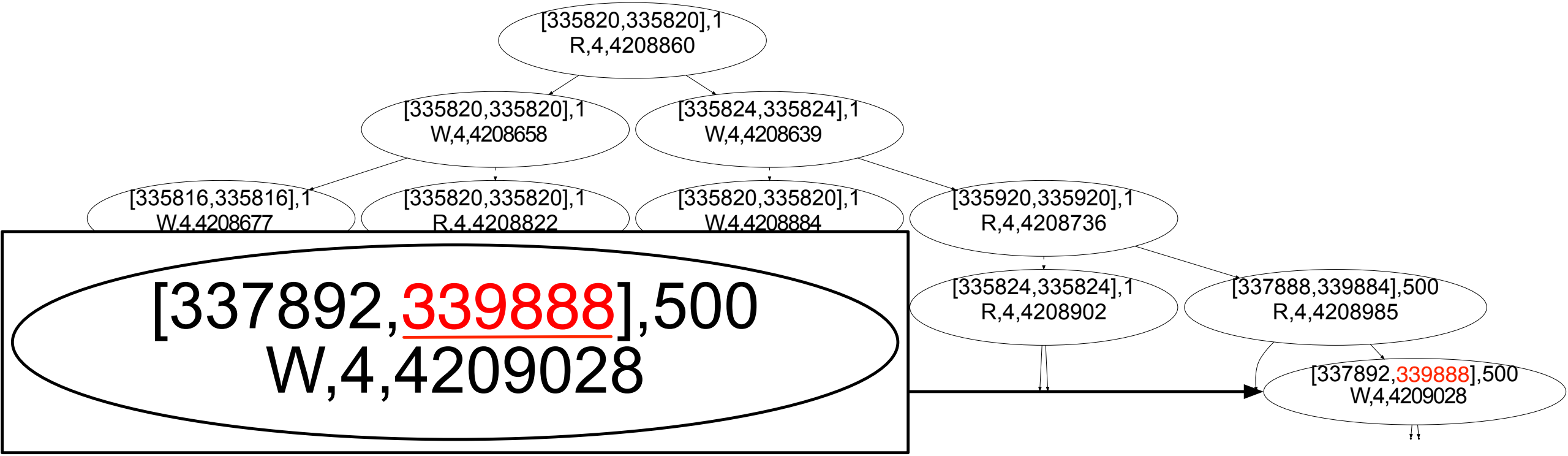
Overlap of Access Bursts: ILP Generation!



$$T_0: \quad 8 \cdot x_0 + 10 + s_0 = a \;\wedge\; 0 \leq x_0 \leq 4 \;\wedge\; 0 \leq s_0 < 4$$

$$T_1: \quad 8 \cdot x_1 + 14 + s_1 = a \;\wedge\; 0 \leq x_1 \leq 4 \;\wedge\; 0 \leq s_1 < 4$$

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

Group Credits

Central to the Archer and Sword Efforts were these:

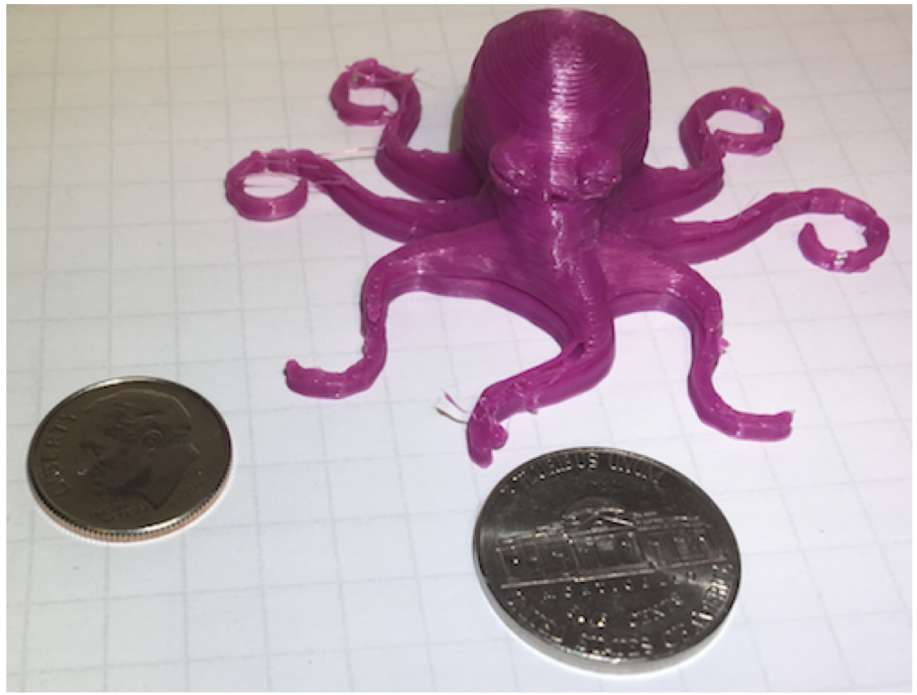
- OMPT Tool API [Eichenberger, Mellor-Crummey, Schulz, Protze, …]
- DataRaceBench [Liao, Lin, Asplund, Schordan, Karlin]

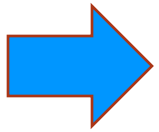
The ROMP race checker (SC'18) also relies on these!

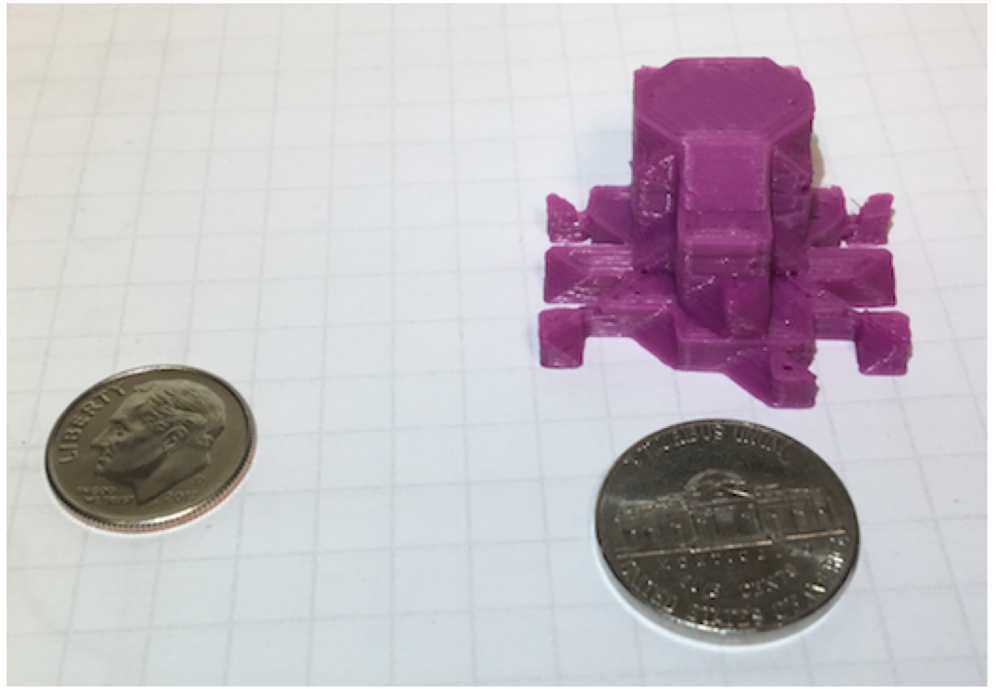
A powerful example of "verification community-building"

Research Thrust Behind FLiT:
Is your Science Hijacked by your compiler?

Floating-point is non-intuitive

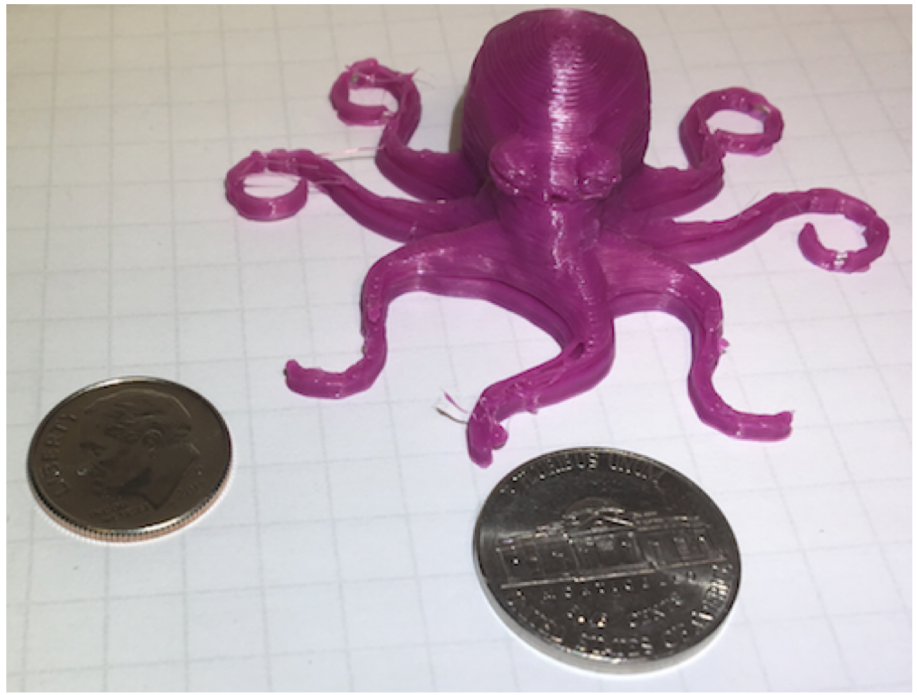


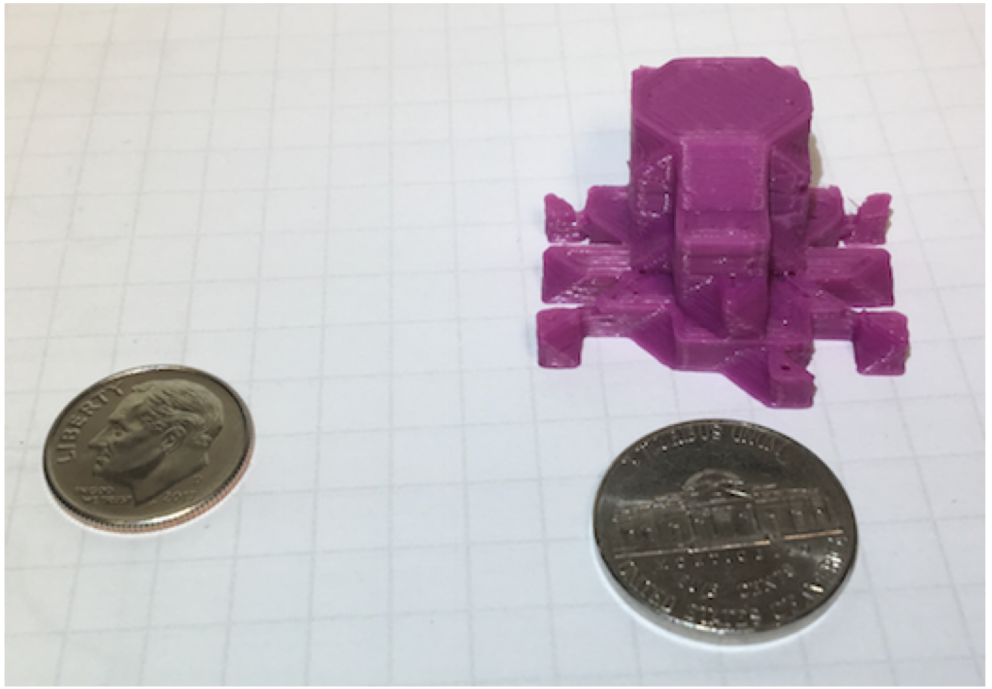




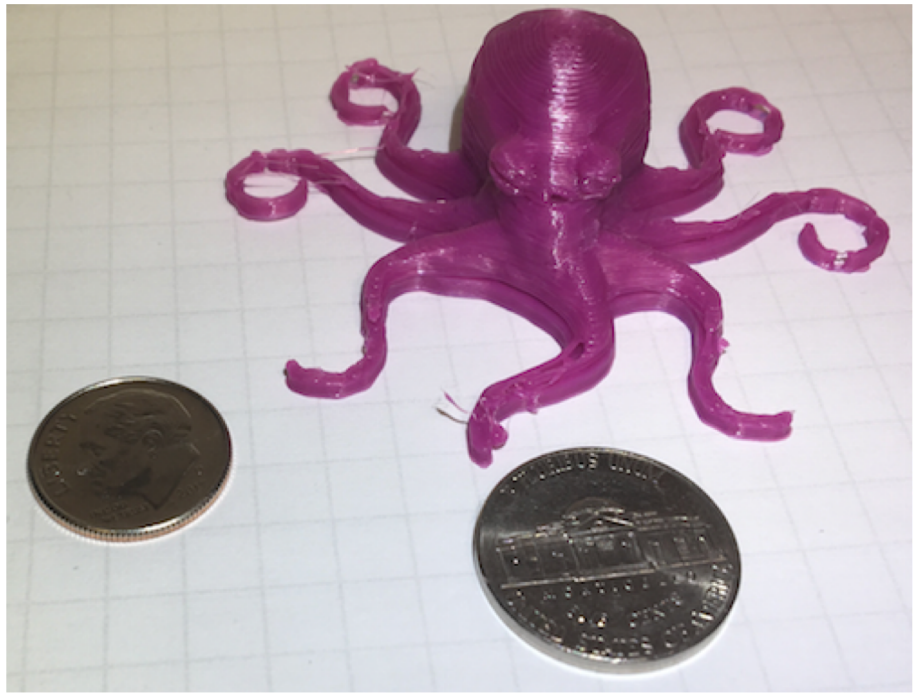
Floating-point is non-intuitive

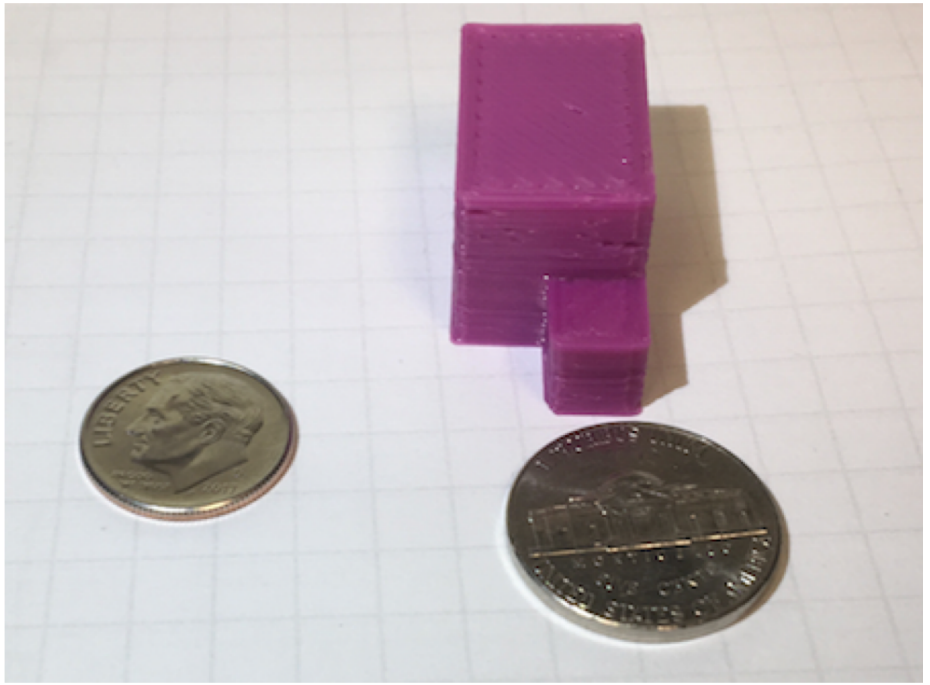
Fun diversion: Move 3D model by a large distance, then bring it back, and 3D-print!



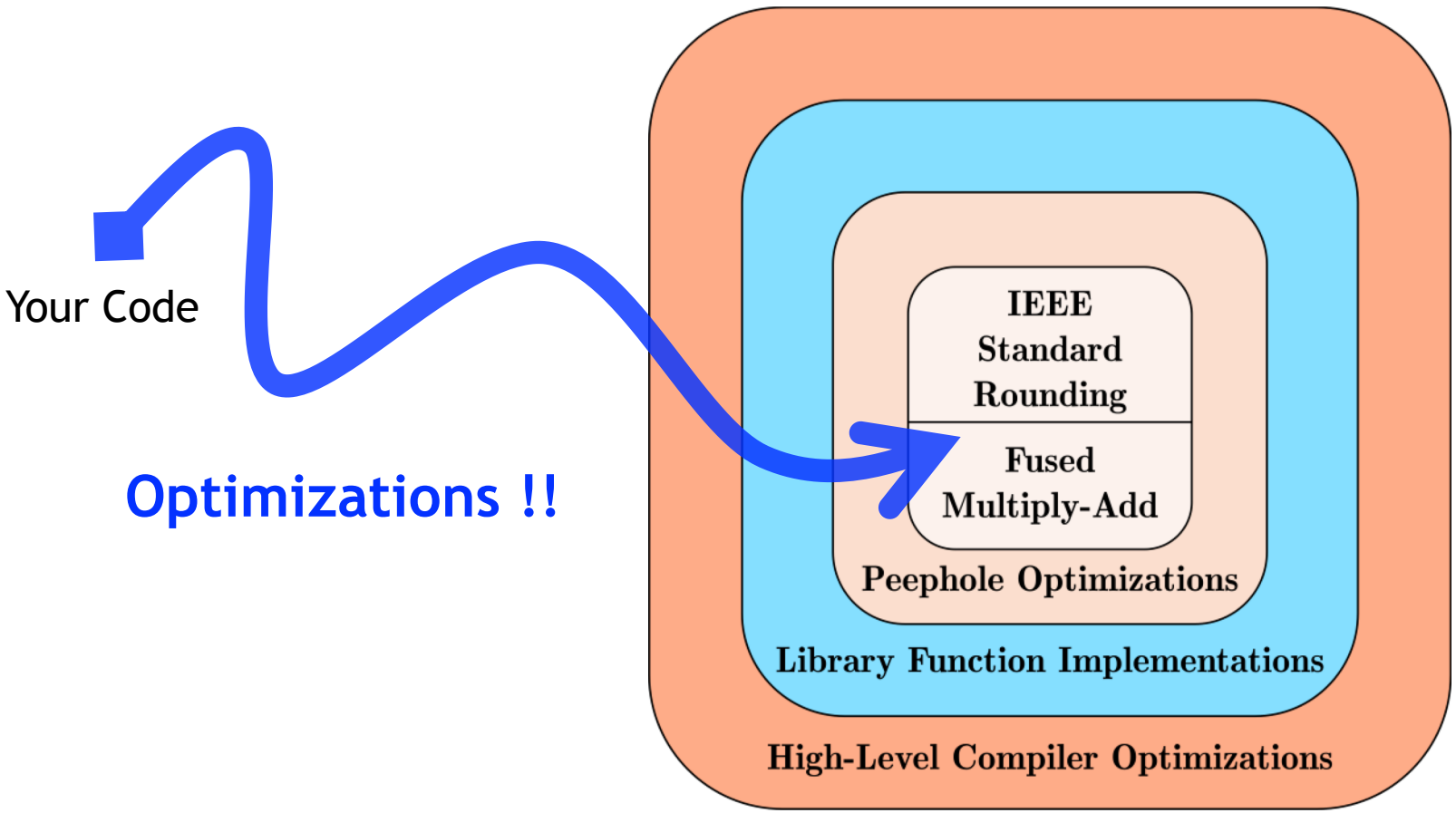


Floating-point is non-intuitive





The code you run is not what you ”see”



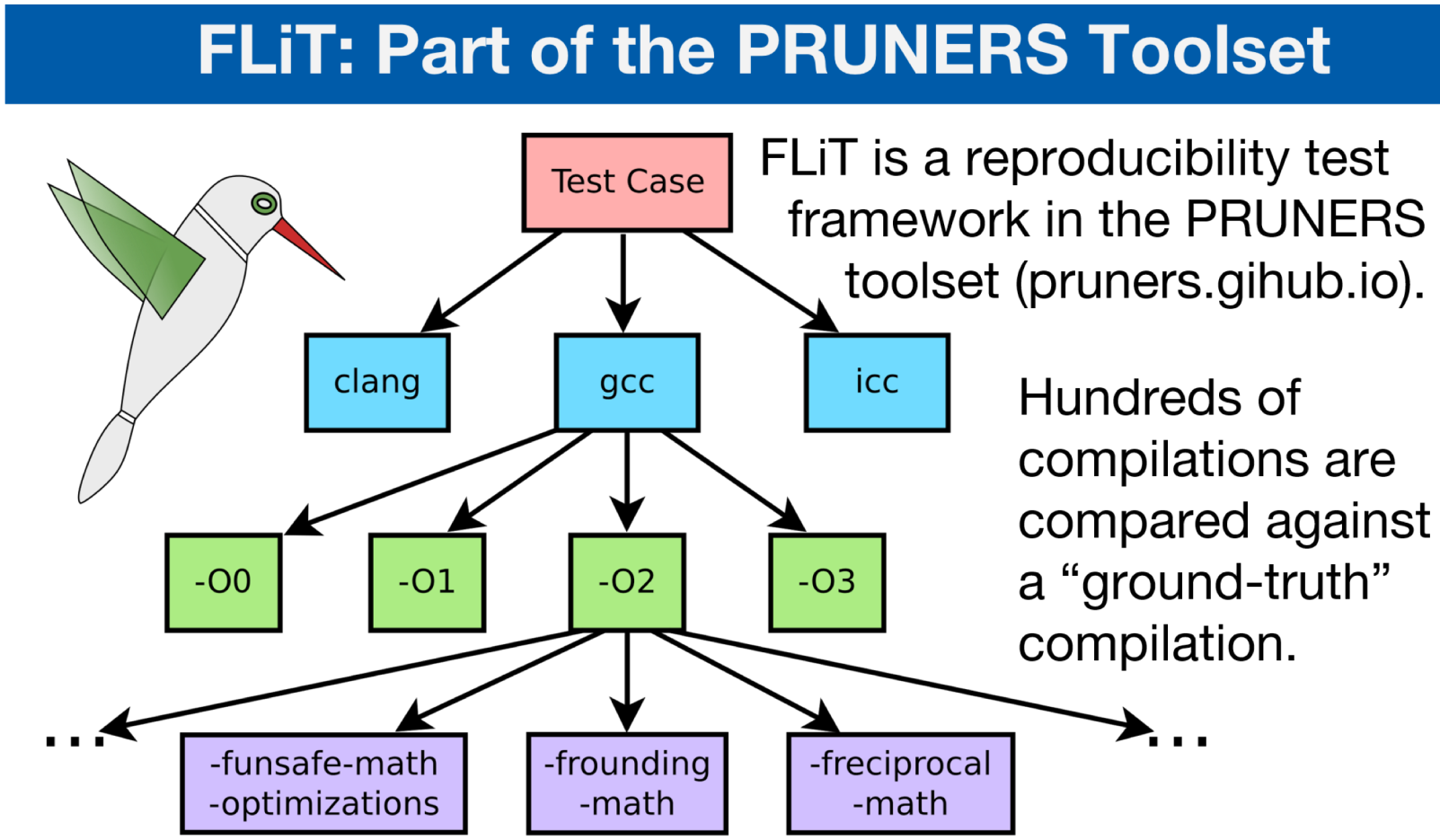
Challenge We Faced (Re-Enactment)

- Given a "large" 100K LOC ***schedule-reproducible*** application
  - **Written in C++/MPI, Frameworks (e.g. RAJA)**
  - **3K functions, 30 functions per file, a few 100 files**

- When one is hit with a "Floating-Point Repro" issue
  - **Optimization by a compiler that's not LLVM-based**

- **How does one:**
  - **Narrow root-cause down to file or function symbol?**
  - **Within modest execution budgets?**

We offer FLiT, a tool to study FP optimizations



THE UNIVERSITY OF UTAH®

One of FLiT's Inspirations

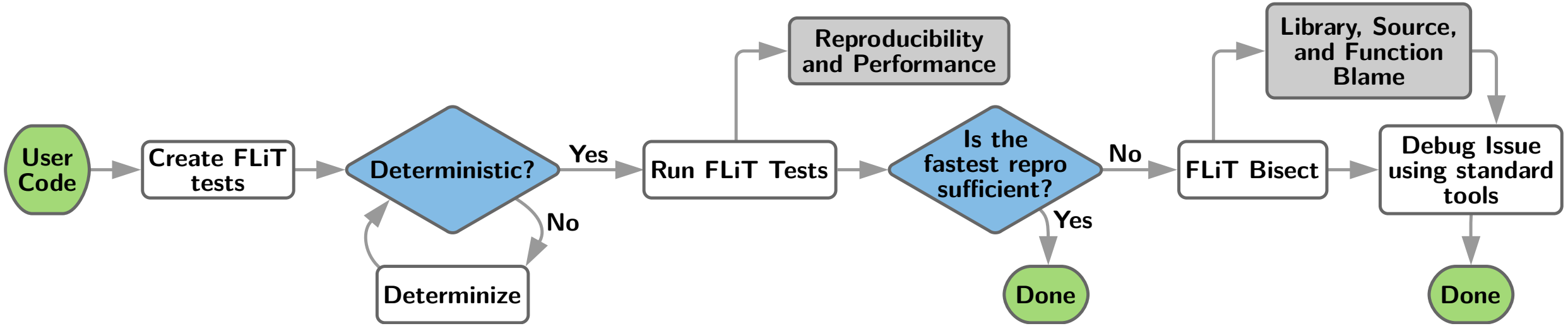
A new ensemble-based consistency test for the Community Earth System Model (pyCECT v1.0)

A. H. Baker, D. M. Hammerling, M. N. Levy, H. Xu, J. M. Dennis, B. E. Eaton, J. Edwards, C. Hannay, S. A. Mickelson, R. B. Neale, D. Nychka, J. Shollenberger, J. Tribbia, M. Vertenstein, and D. Williamson

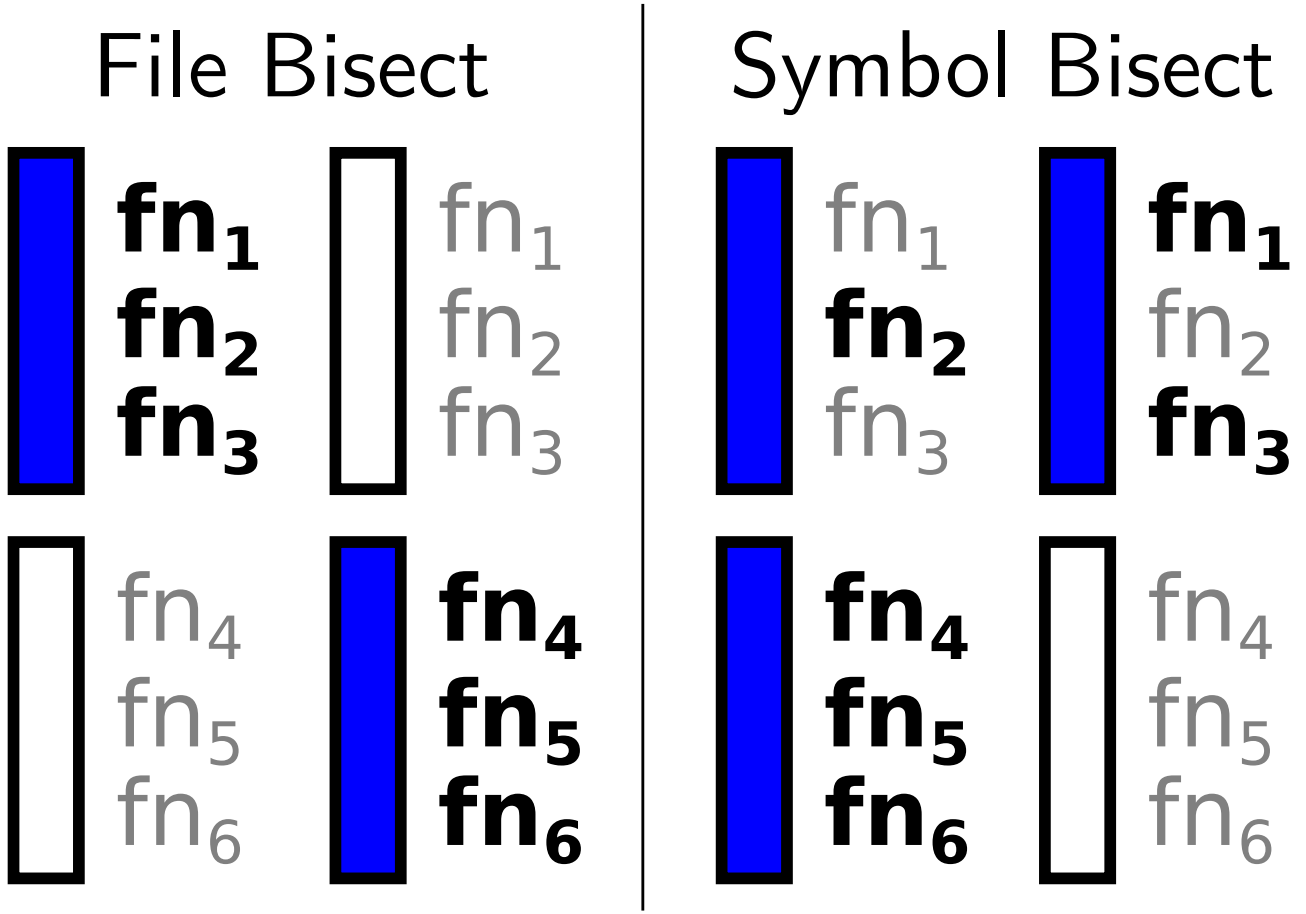
Making root cause analysis feasible for large code bases: a solution approach for a climate model

Daniel J. Milroy, Allison H. Baker, Dorit M. Hammerling, Youngsung Kim, Elizabeth R. Jessup, Thomas Hauser

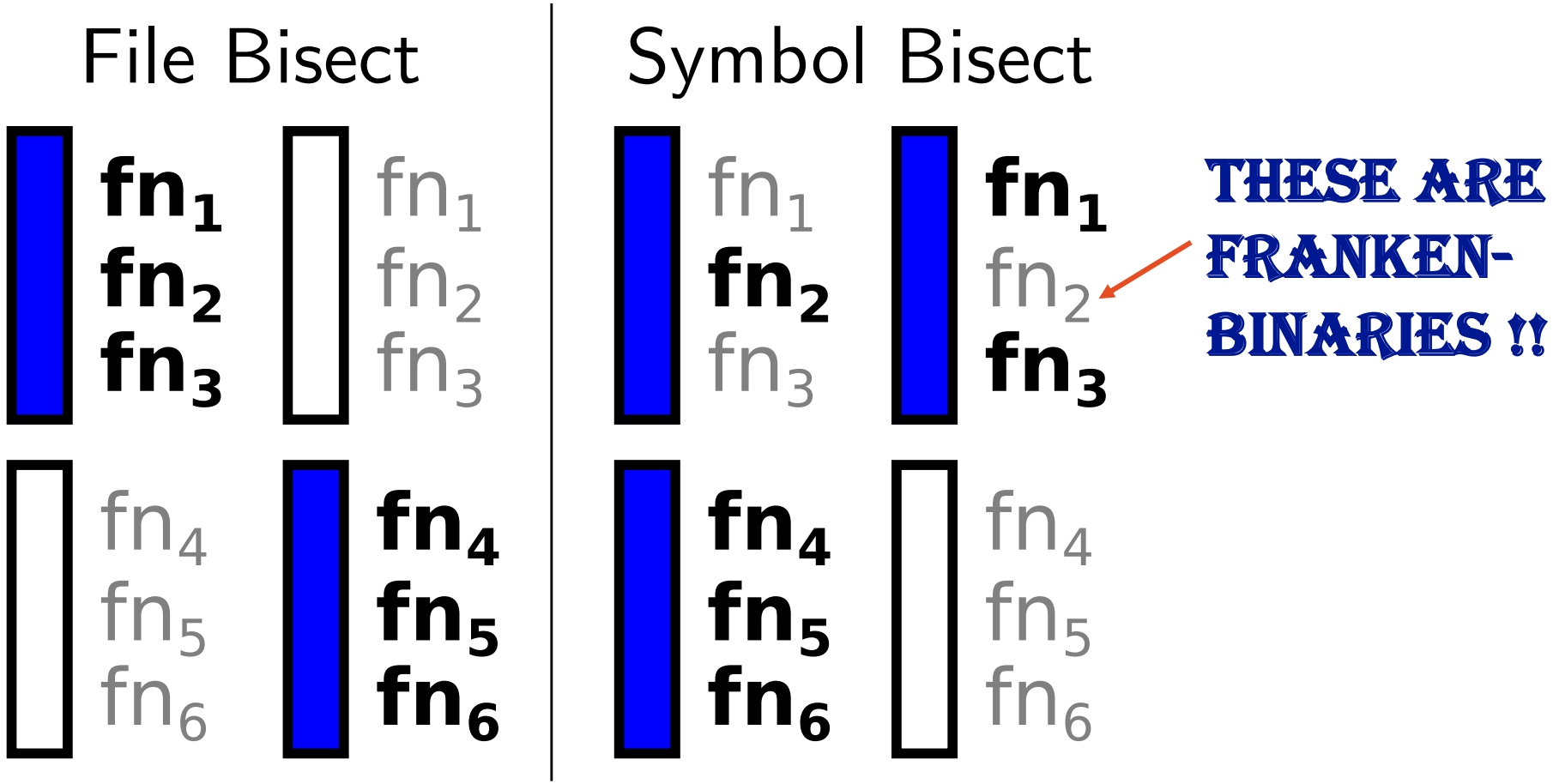
FLiT Workflow



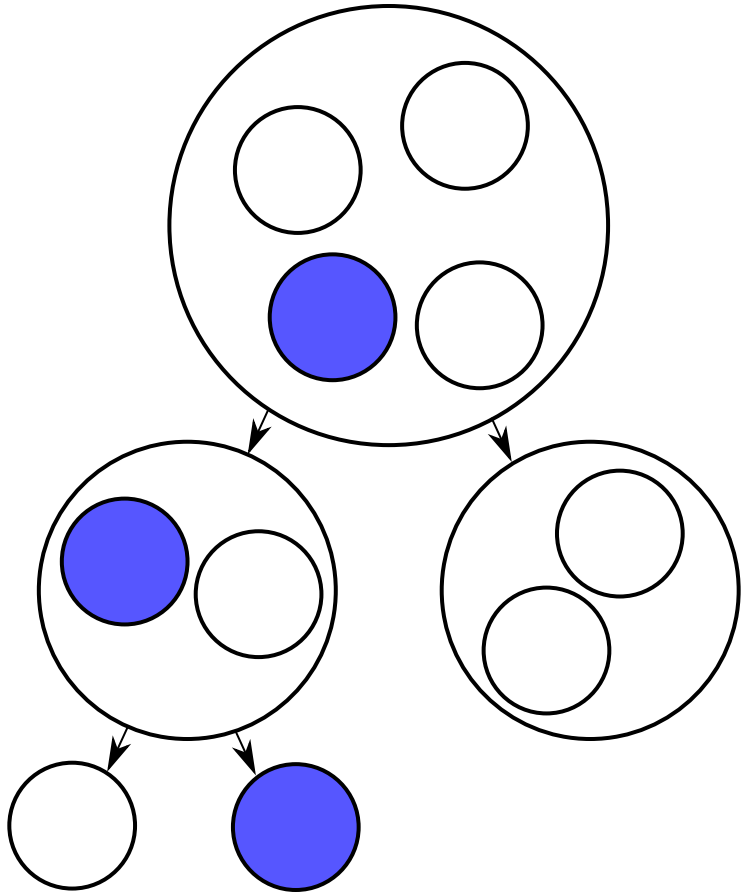
FLiT Bisect: File or Symbol



FLiT Bisect: File or Symbol



FLiT-Bisect under Singleton Minimal Set



FLiT Tool Capabilities

source files	97
average functions per file	31
total functions	2,998
source lines of code	103,205

Fig. 9. General statistics of code used by the MFEM examples.

	g++	clang++	icpc	total
average test executions	64	29	27	30
file bisect successes	78/78	24/24	778/984	880/1,086
symbol bisect successes	51/78	24/24	585/778	660/880

Fig. 8. Compiler characterization of bisect with MFEM.

Non-Portability Between IBM XLC and GCC

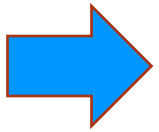
```
#define xsw(a,b) a^=b^=a^=b
```

Another Non-Portability Find

```
If (f(x,y) == 0)
  { … }
Else
  { … }
```

Rewrote Code To:

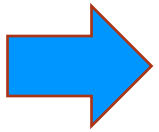
```
If (f(x,y) == 0)
  { … }
Else
  { … }
```



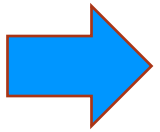
```
If (abs(f(x,y)) < EPS)
  { … }
Else
  { … }
```

More General Approach (Ericson)

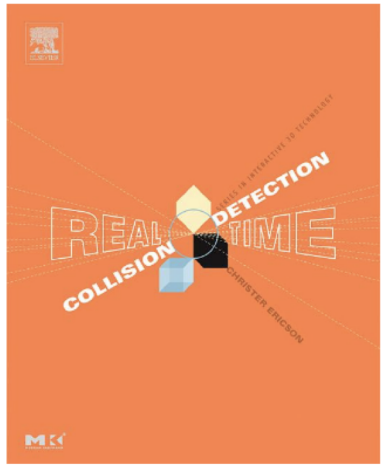
```
If (f(x,y) == 0)
  { … }
Else
  { … }
```



```
If (abs(f(x,y)) < EPS)
  { … }
Else
  { … }
```



Make EPS a function of x,y

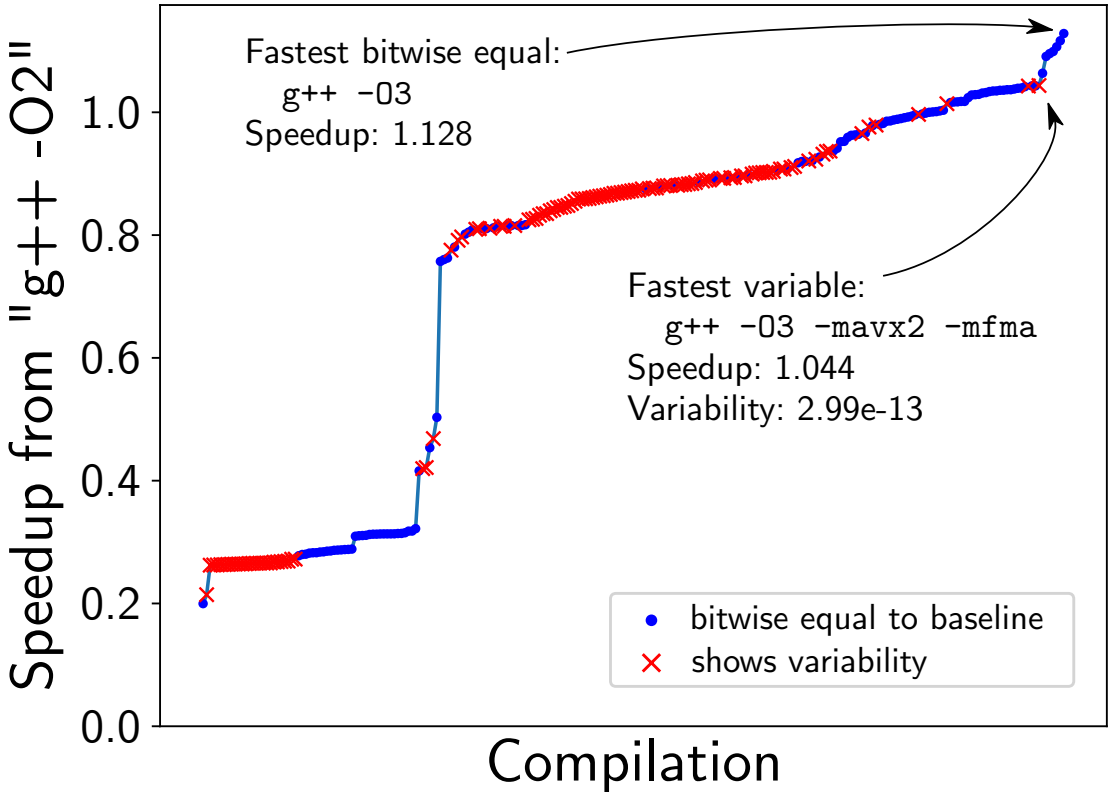


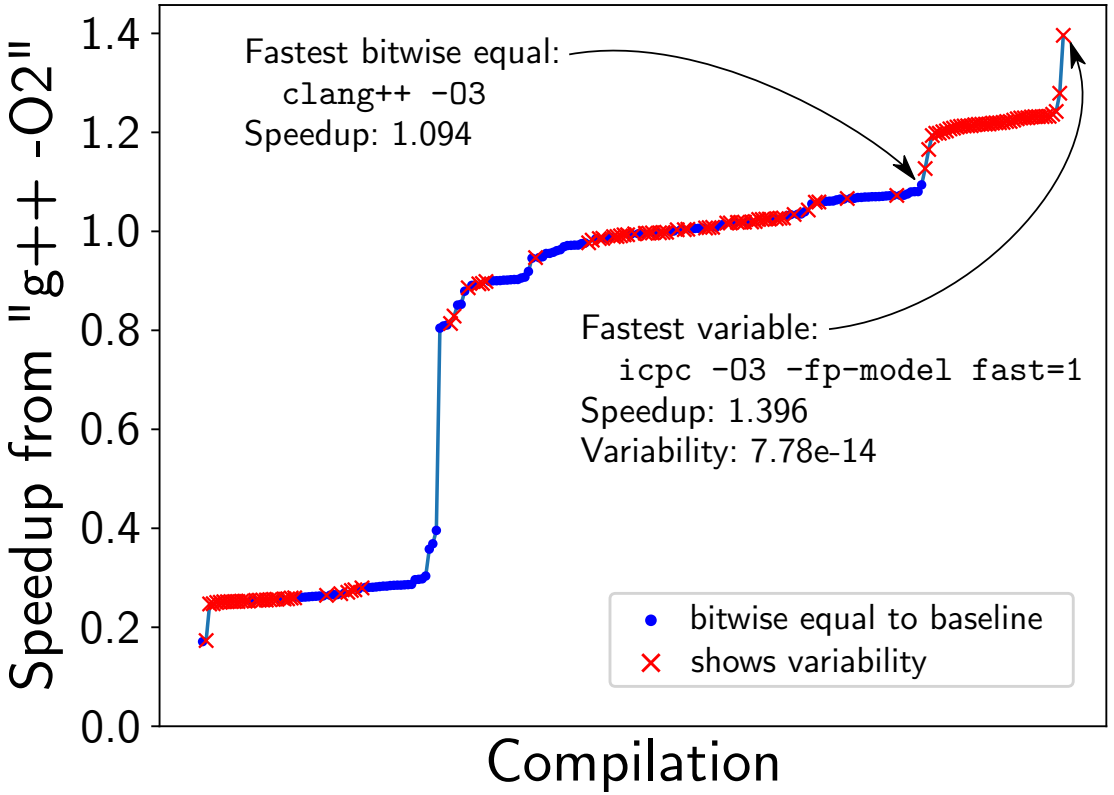
Real-Time Collision Detection
1st Edition
Christer Ericson

Hardback $84.00 | **eBook** $46.36 | **eBook Rental** from $28.98

CRC Press
Published December 22, 2004
Reference - 594 Pages
ISBN 9781558607323 - CAT# K16167

Compilation, Repro, Speedup (Two MFEM Tests)





Tools to Test FP Repro Checking Tools?

Tools to Test FP Repro Checking Tools?

FP “Fault Injector” (LLVM)

CONCLUDING REMARKS

Back to Formal Methods

Why can’t HPC achieve even a modicum of the success of Formal Methods in HW Design?

Stunning Success in CPU Design

- **Replacing Testing with Formal Verification in Intel® Core™ i7 Processor Execution Engine Validation,** Roope Kaivola **(CAV 2009)**

  Essentially, they let-go simulation / testing teams

- 2013 *Microsoft* Research Verified Software Milestone *Award*
- Formal Methods ***are why microprocessors work!***

  Hardware companies take “Formal” VERY SERIOUSLY !!

How Formal Methods came into HW Design: Intel FDIV

- A – B * (A/B) = 0 (math)
- A – B * (A/B) = 256 (Pentium)

- FROM: Dr. Thomas R. Nicely Professor of Mathematics Lynchburg College 1501 Lakeside Drive Lynchburg, Virginia
- TO: Whom it may concern RE: Bug in the Pentium FPU DATE: **30 October 1994**

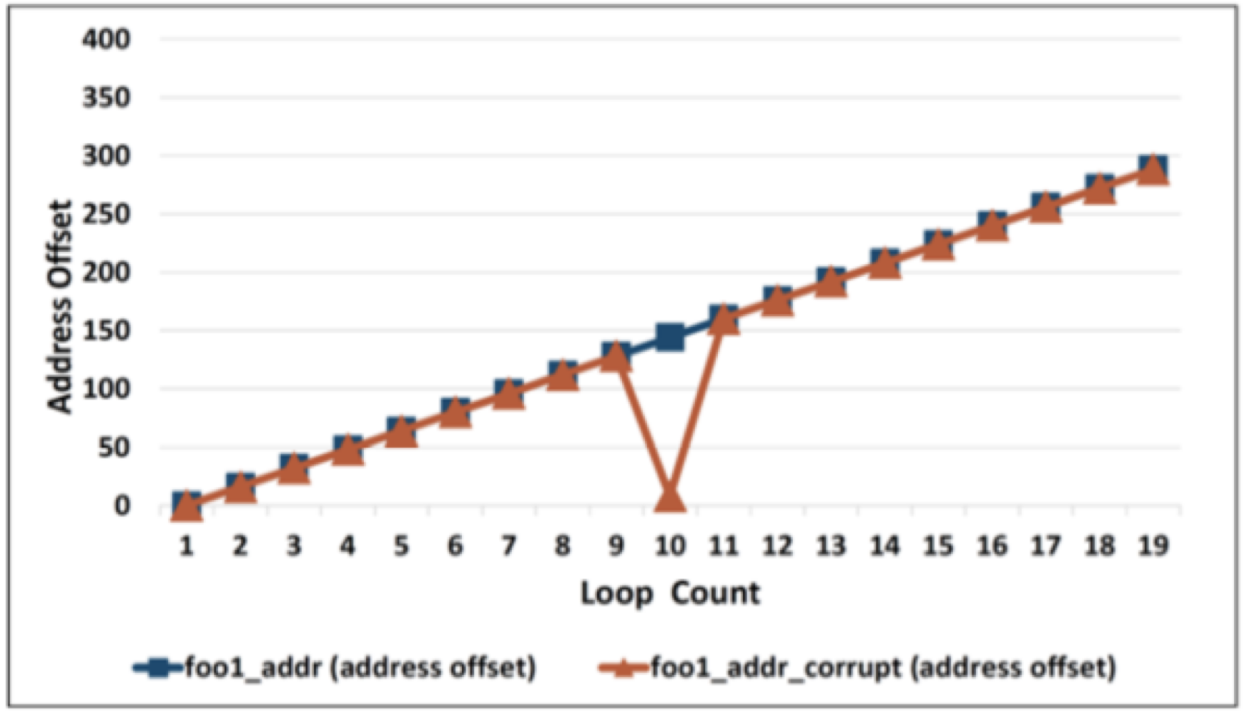
- 4195835.0 - 3145727.0*(4195835.0/3145727.0)

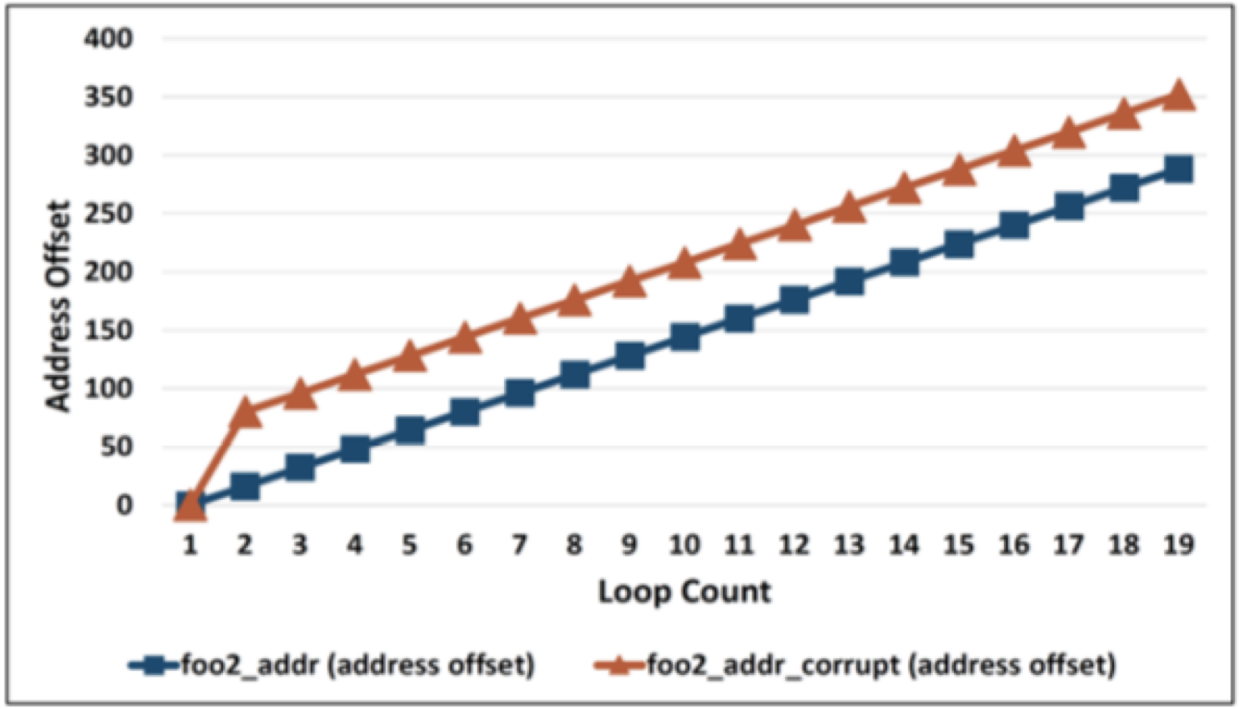
More Examples of Formal Methods

- A Formal Specification of the Itanium Memory Model
  - By Gil Neiger, a GaTech Professor who went to Intel
  - Gil's model was very helpful!
    - Lamport wrote at TLA+ model
    - See our CAV'04 paper that checks litmus tests using HOL -> SAT

FM Example: LLVM Transformation Verification

We applied SMACK (C/LLVM formal verifier) and found a bug in our address relativization algorithm used in ***System Resilience***





Formal Methods are not “All or Nothing”

They can take root in an incremental manner

Most such methods are ***semi-formal*** in any case

Formal Methods Help Have Fun with Bugs

Debugging is depressing

Making tight correctness statements is fun

Formal Methods Are Ego-Deflating

The most fun part of formal methods:

Counterexample generation!

Semi-Formal and Pragmatic Are Central!

Semi-Formal and Pragmatic Are Central!

Pedagogy:

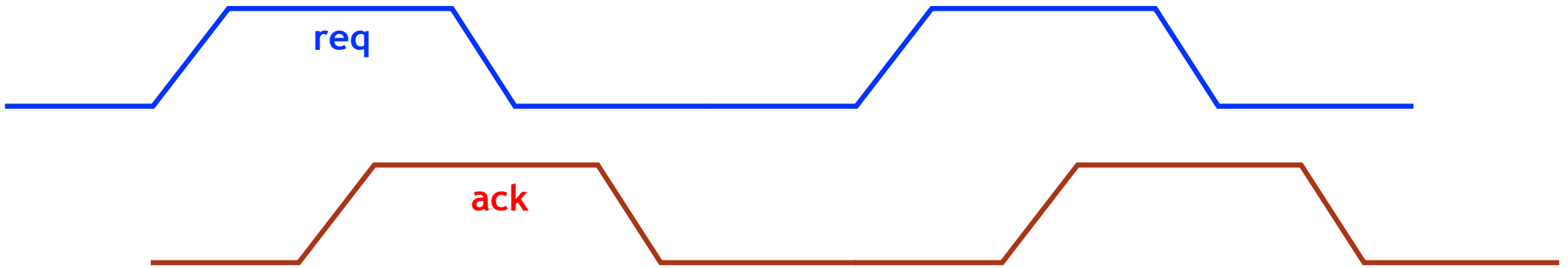
Do include the ***Address and Thread Sanitizer*** Tools!

We need ***peachy assignments*** to be successful

Peachy Assignment: Races and store buffers under a minute!

- T1 : do N times { req=1; while(!ack); req=0; while(ack); }
- T2 : do N times { while(!req); ack=1; while(req); ack=0; }

Java Volatiles Essential !!



Semi-Formal and Pragmatic Are Central!

Taught this class (not much HPC background) in a recent 1-week course:

ISP for MPI analysis

Data Race Detection illustration (Java Volatiles), using Archer

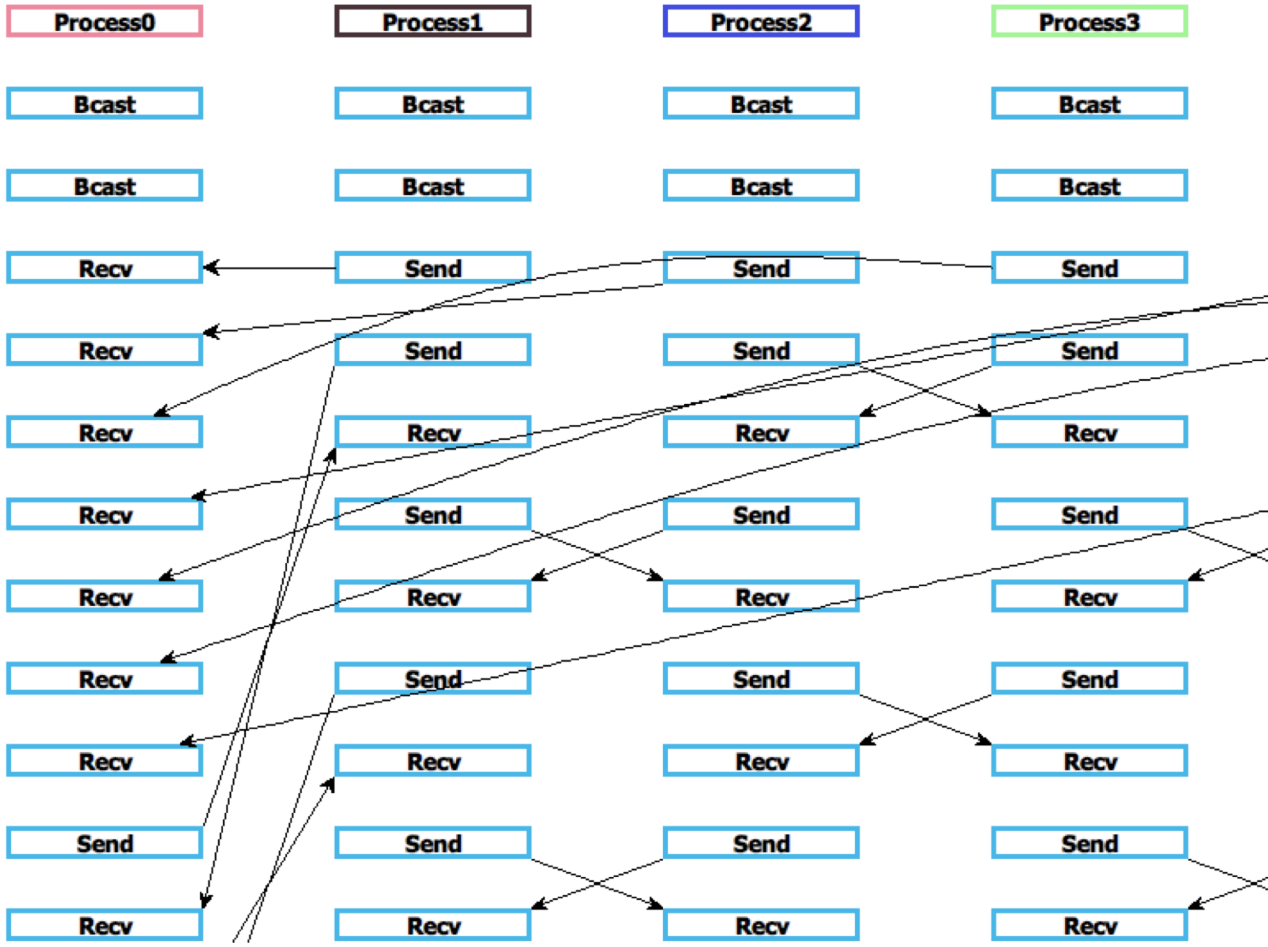
Linear versus Tree Sum of FP, Address and Thread Sanitizer

All these tools could be run without any trouble by my hosts

Audience-favorite at GIAN: code+message matches from ISP (decade-old)

```
void Odd_even_iter(int local_A[], int temp_B[], int temp_C[],
        int local_n, int phase, int even_partner, int odd_partner,
        int my_rank, int p, MPI_Comm comm) {
   MPI_Status status;

   if (phase % 2 == 0) {
      if (even_partner >= 0) {
         MPI_Sendrecv(local_A, local_n, MPI_INT, even_partner, 0,
            temp_B, local_n, MPI_INT, even_partner, 0, comm,
            &status);
         if (my_rank % 2 != 0)
            Merge_high(local_A, temp_B, temp_C, local_n);
         else
            Merge_low(local_A, temp_B, temp_C, local_n);
      }
   } else { /* odd phase */
      if (odd_partner >= 0) {
         MPI_Sendrecv(local_A, local_n, MPI_INT, odd_partner, 0,
            temp_B, local_n, MPI_INT, odd_partner, 0, comm,
            &status);
         if (my_rank % 2 != 0)
            Merge_low(local_A, temp_B, temp_C, local_n);
         else
            Merge_high(local_A, temp_B, temp_C, local_n);
      }
   }
}  /* Odd_even_iter */
```



Other two audience favorites

- FP linear-sum versus tree-sum
- Java volatiles and hang when you omit them

Conclusions

- Formal Methods *are* relevant for HPC
  - Hide them in tools and documents, but still derive benefit
- Tool construction and must be enabled
  - Important: Challenge problems, benchmarks, sharable tools
- Pedagogy as long-term hope
  - Books and tutorials must cover these topics:
    - Correctness **(rigorous techniques, coverage, testing)**
    - Tools **(where one can obtain usable/stable tools)**

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 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**

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

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;

}
```

Analogy due to Herb Sutter

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

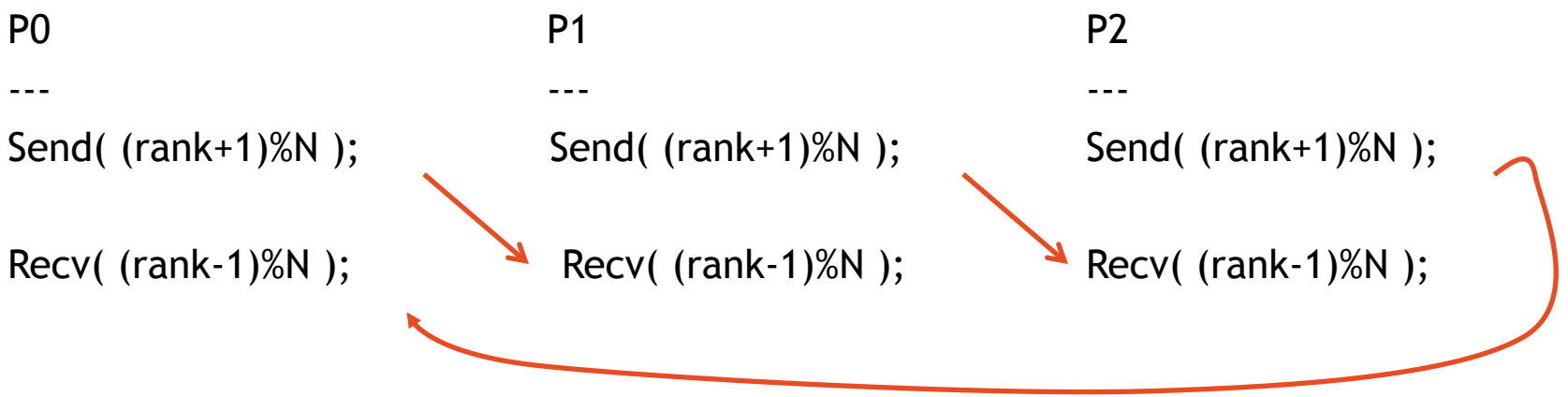
However, this “warp view” often appears to be lost

E.g. When compiling with optimizations

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

New Answer: 0, 2, 4, 6, 8, …

Counterexample: Be sure that the math ports



- Deadlocks when ported from Fortran to C

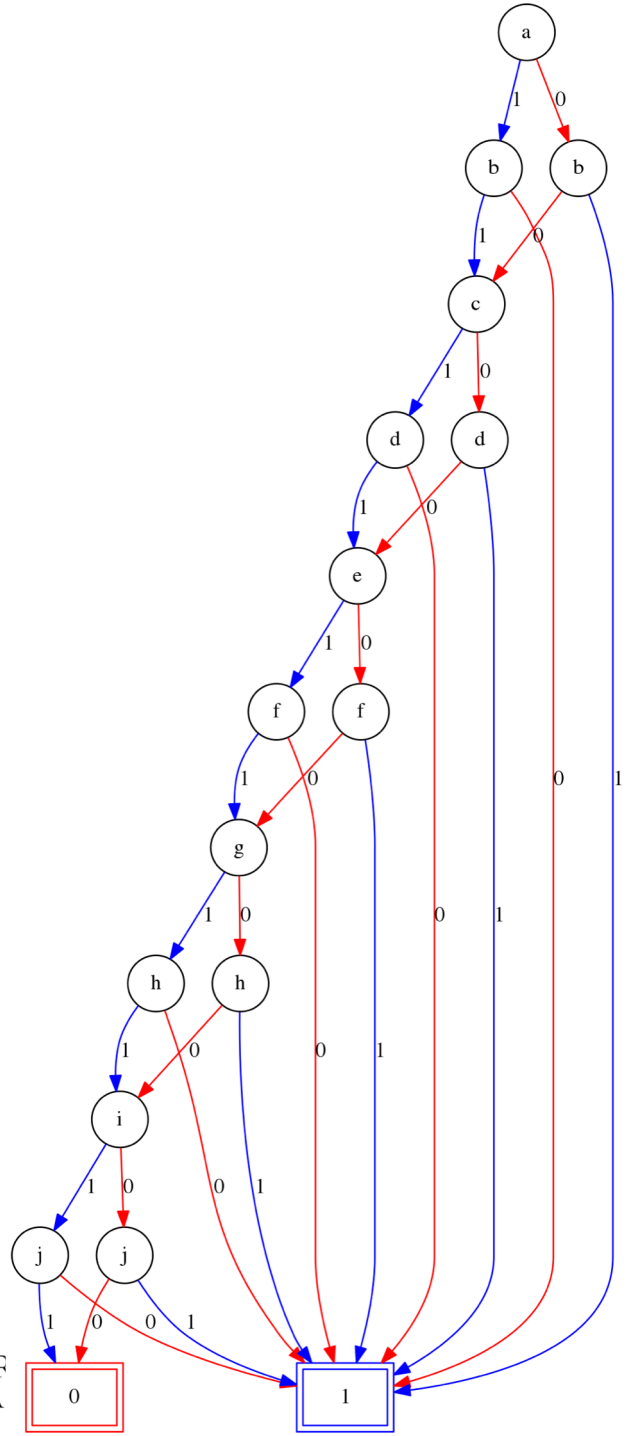
Not that cost-effective

- Where “extrapolation” works
  - **Where fuzz-testing**
  - **Highly observable systems (e.g. processor pipelines)**

Formal Methods that Came to the Rescue: BDD

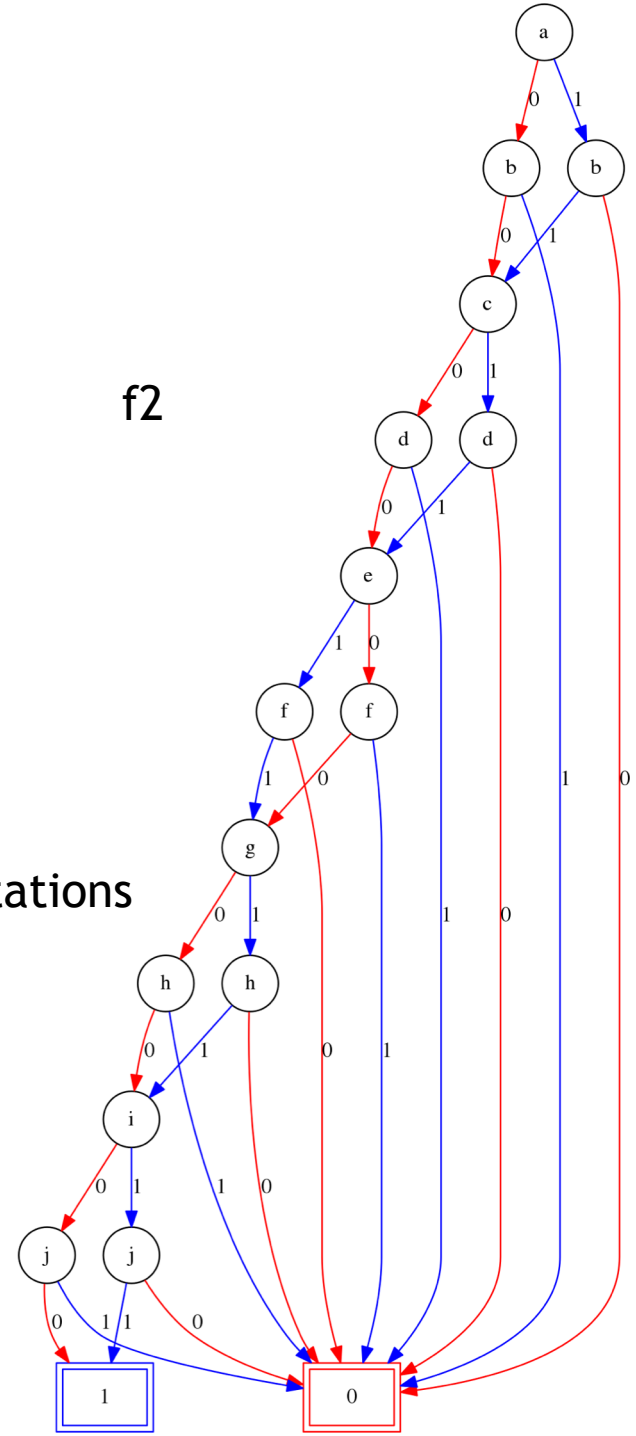
- Binary Decision Diagrams (BDD)
- A compact way to represent (and manipulate) large Boolean functions

f1



These functions have
Linearly-sized representations

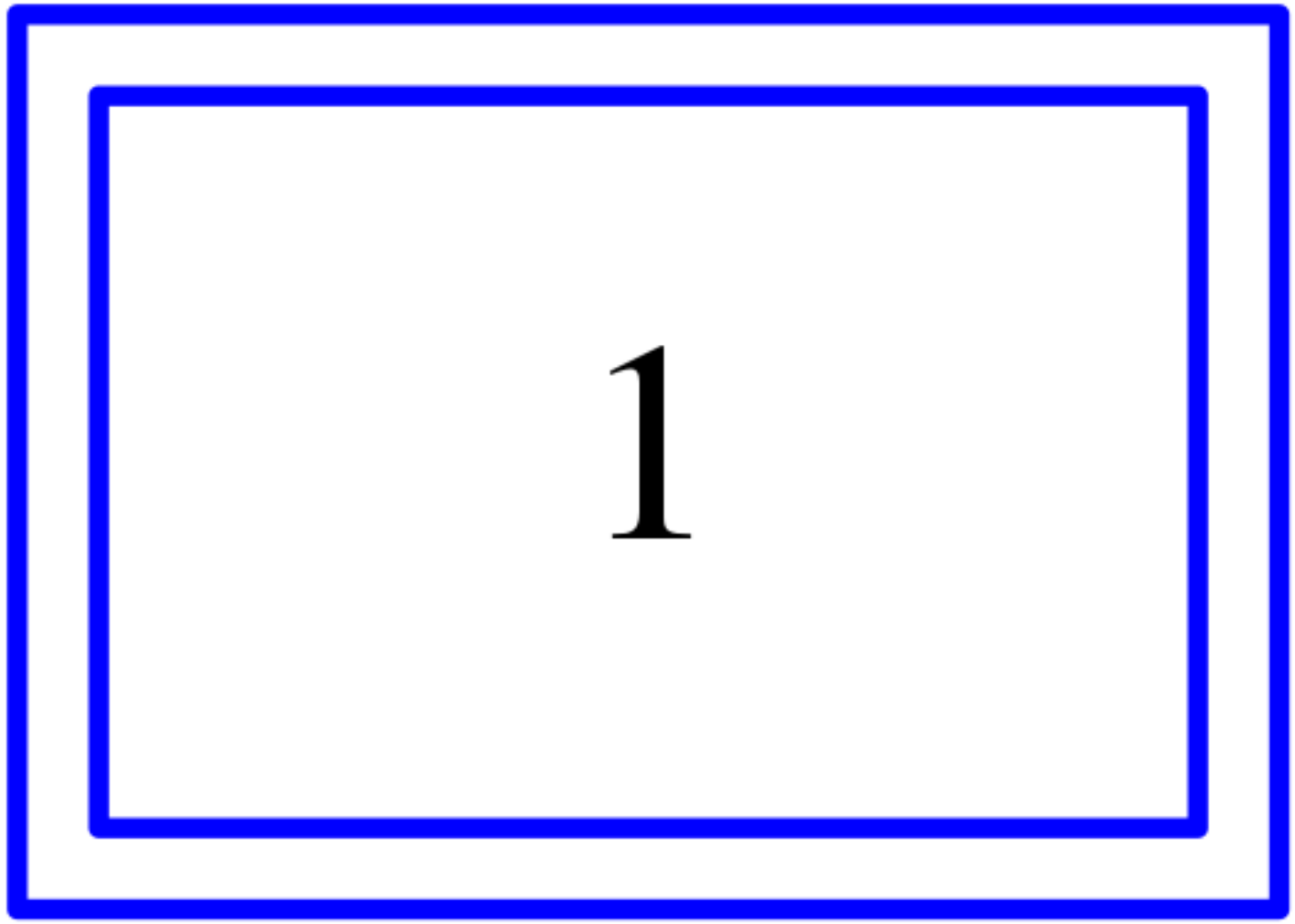
f2



THE UNIVERSITY OF UTAH

The tautology f1 ⇔ !f2 is established through “symbolic reasoning”

The BDD for f1 ⇔ !f2 is below (demo from http://formal.cs.utah.edu:8080/pbl/BDD.php)



One spectacular everyday use: Parsing!

- How olden-day compilers worked
  - E.g. turn 4 + 5 * 6 into 4))) + (((5)) * ((6
  - Then parenthesize: (((4))) + (((5)) * ((6)))
- Knuth on early 60's compilers :
    - "Phases jumbled together into a huge sprawling algorithm"

Power of SAT-Solving is Behind Modern FM

- **SAT = Boolean Satisfiability**
- **Supposed to be NP-complete**
- **Yet runs surprisingly fast**

Power of SMT-solving Powers Modern FM

- SMT = Satisfiability Modulo Theories
  - Like “SAT” except for a mix of theories
    - Int, List, String, Array, Functions, …
- Uses SAT underneath

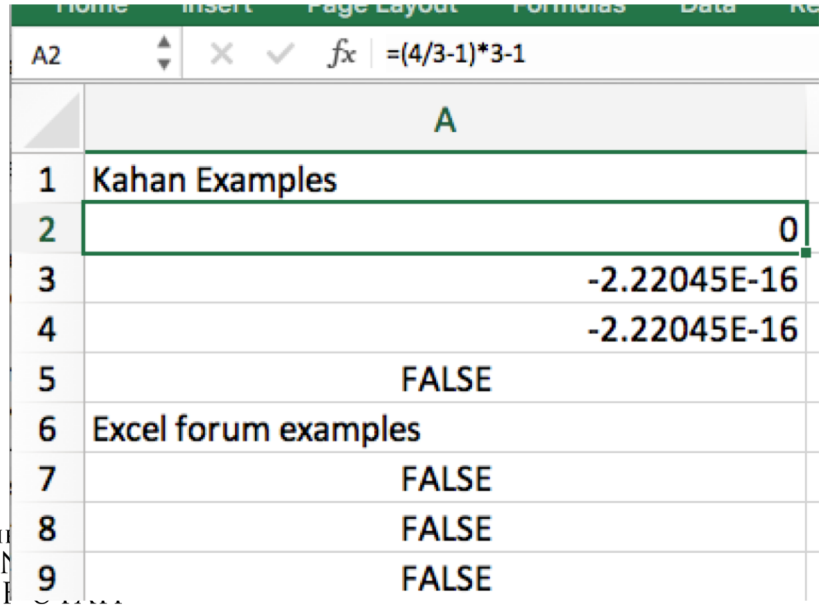
Power of SMT-solving Powers Modern FM

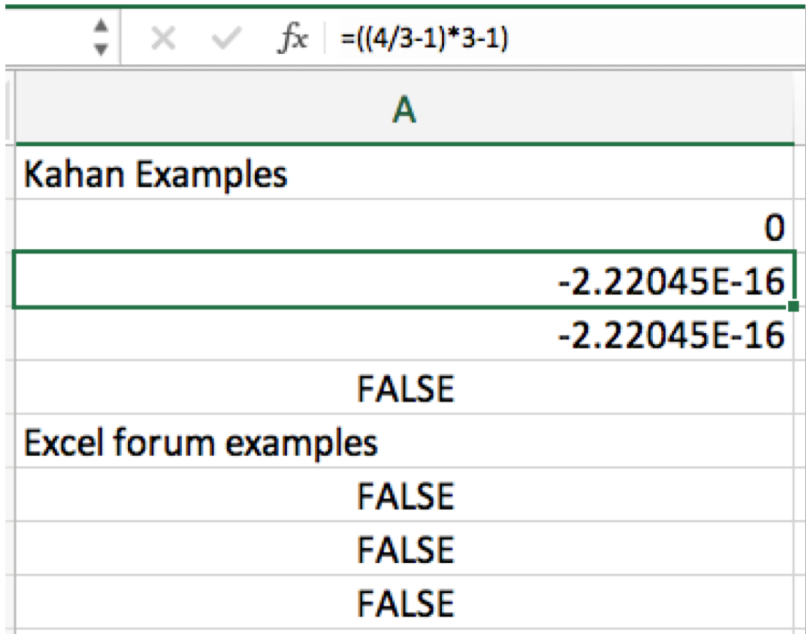
- SMT solvers power many FM engines in
  - Compiler analysis (safe compilation)
  - Security
  - Memory access overlap detection

  - HPC verification tools have begun incorporating SMT
    - Race checking
    - MPI analysis

Inexplicable variability adds to the confusion

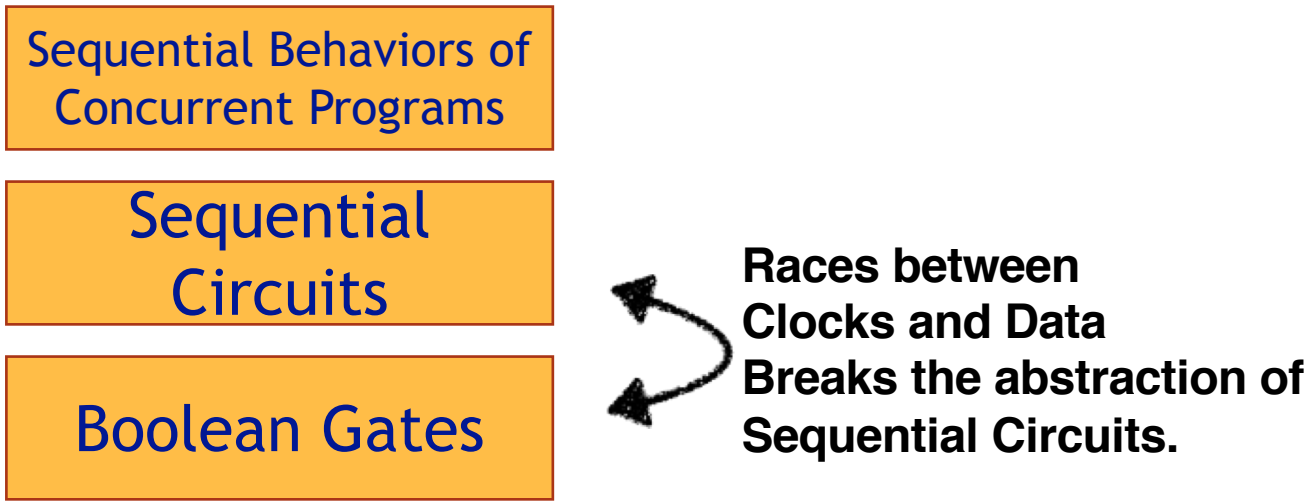
- Example: Excel's rounding is bizzare (First example by Kahan)
  - (4/3-1)*3-1 produces 0
  - ((4/3-1)*3-1) produces -2.22045E-16
  - Bitwise different
  - Adding 0 has the same effect as outer "(…)"
  - All this changes in an IF(….) context



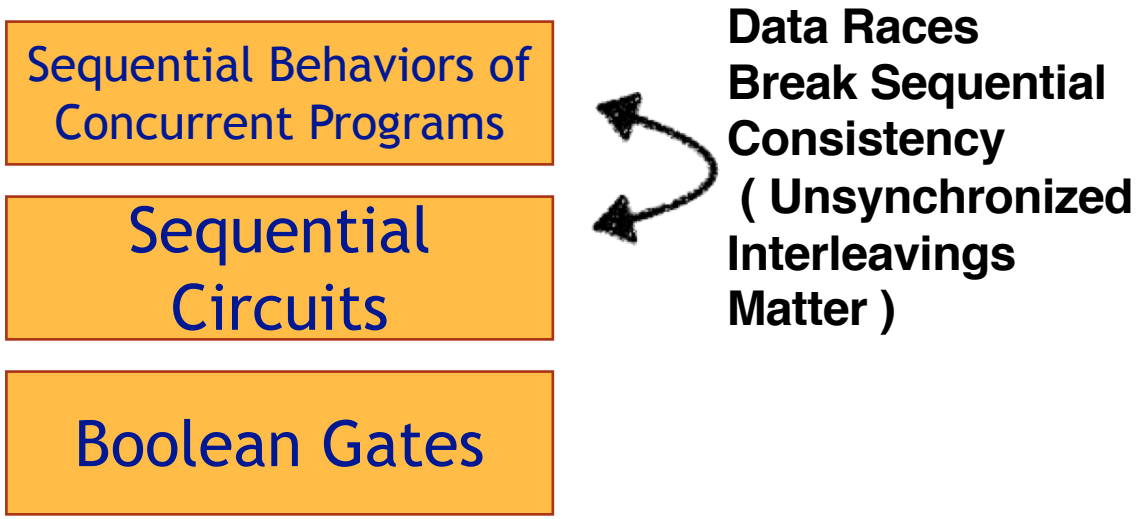


Most often, we “design by debugging” [McMillan, ‘98]

- When does debugging end?
  - When do we sign-off on a chip (i.e., “last bug” is caught?)


Sequential Behaviors of Concurrent Programs
Sequential Circuits
Boolean Gates
Races between Clocks and Data Breaks the abstraction of Sequential Circuits.

1
3
1


Sequential Behaviors of Concurrent Programs
Sequential Circuits
Boolean Gates
Data Races Break Sequential Consistency (Unsynchronized Interleavings Matter)