Verifying Concurrent Programs

Daniel Kroening

28 May – 1 June 2012

Outline

J. Alglave

A. Donaldson

A. Kaiser

T. Wahl

Soundness of Data Flow Analyses for Weak Memory Models, APLAS 2011

Symmetry-Aware Predicate Abstraction for Shared-Variable Concurrent Programs, CAV 2011

Dynamic Cutoff Detection in Parameterized Concurrent Programs, CAV 2010

Boom: Taking Boolean Program Model Checking One Step Further, TACAS 2010

Symbolic Counter Abstraction for Concurrent Software, CAV 2009

Forms of Concurrency

- Shared-variable concurrency
  (Linux pthread library, Win32 thread API, . . .)
  - Share all memory
  - Share OS API (e.g., file descriptors)

- Multiple processes on the same machine
  - Share file system
  - Can share memory via `mmap`

- Programs on different machines
  - Can communicate e.g. via UDP or TCP

Asynchronous vs. Synchronous Concurrency

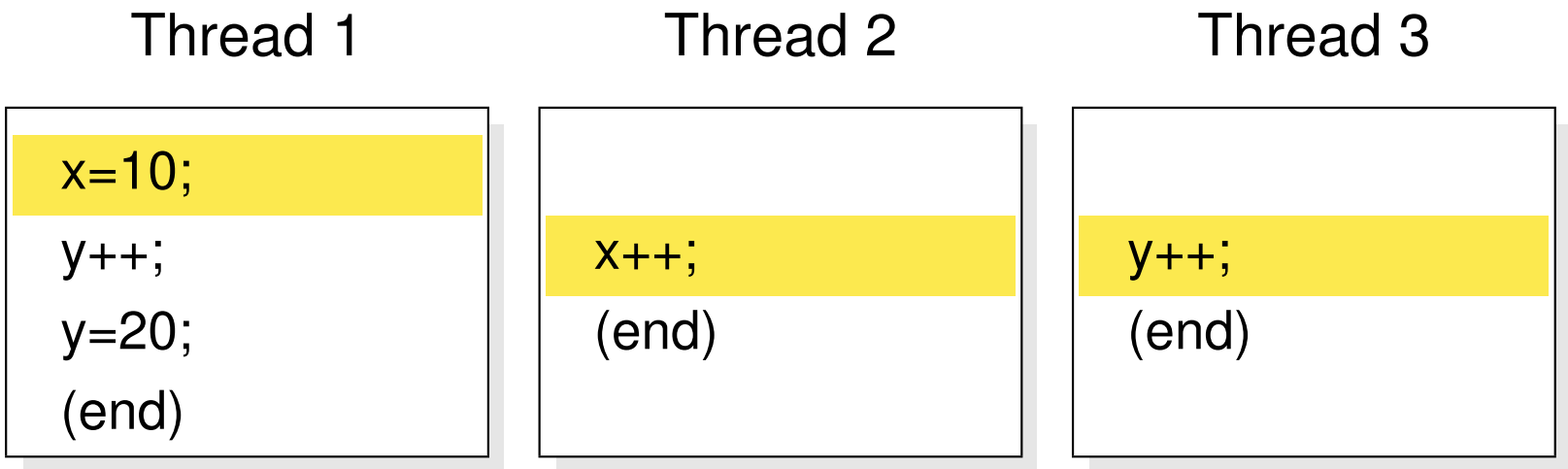
Synchronous	Asynchronous
Partition state: $S = S_1 \times \ldots \times S_n$	Transition relation for each process: $T_i : S \times S$
$T_i : S \times S_i$	
Overall system: $T(s, s') \iff \bigwedge_{i=0}^{n} T_i(s, s'_{(i)})$	Overall system: $T(s, s') \iff \exists i. T_i(s, s')$
Each process can perform a step in each transition	Only one process performs a step in a transition

Asynchronous Shared-Variable Concurrency

UNIVERSITY OF OXFORD

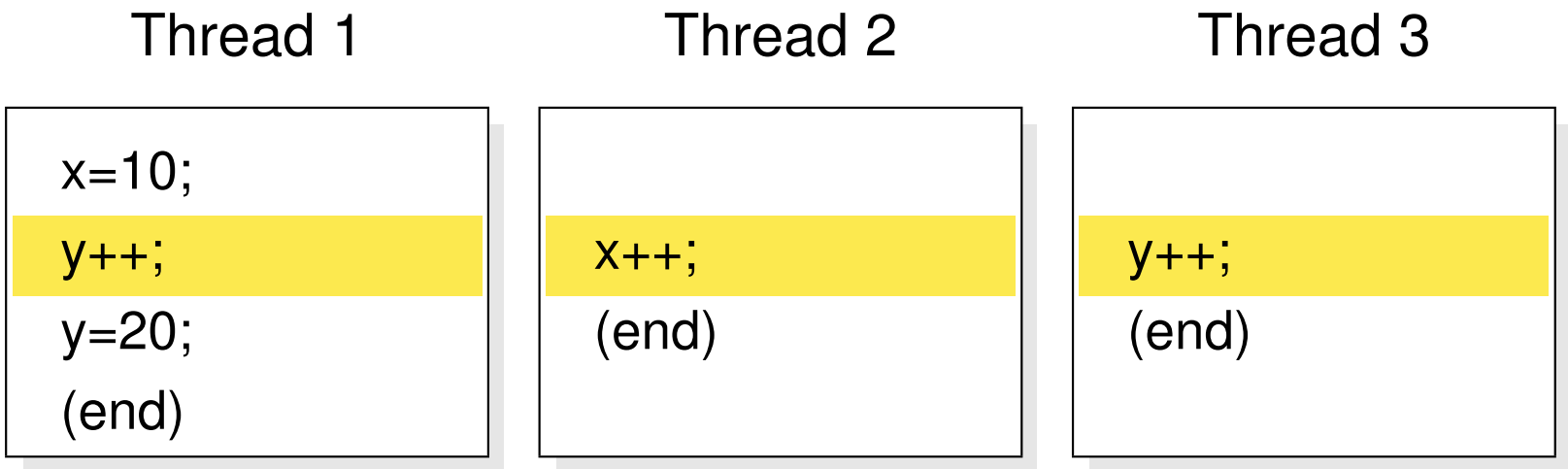
- We focus on asynchronous shared-variable concurrency
- Motivated by Intel's multi-core story
  - Doubling the gate count doubles the power consumption
  - Increasing the clock speed requires raising voltage, with manifold increase in power!
- Scaling micro-processors is easier by replicating CPU cores
- CPUs with 100 cores are around

Thread Interleavings: Example (1)



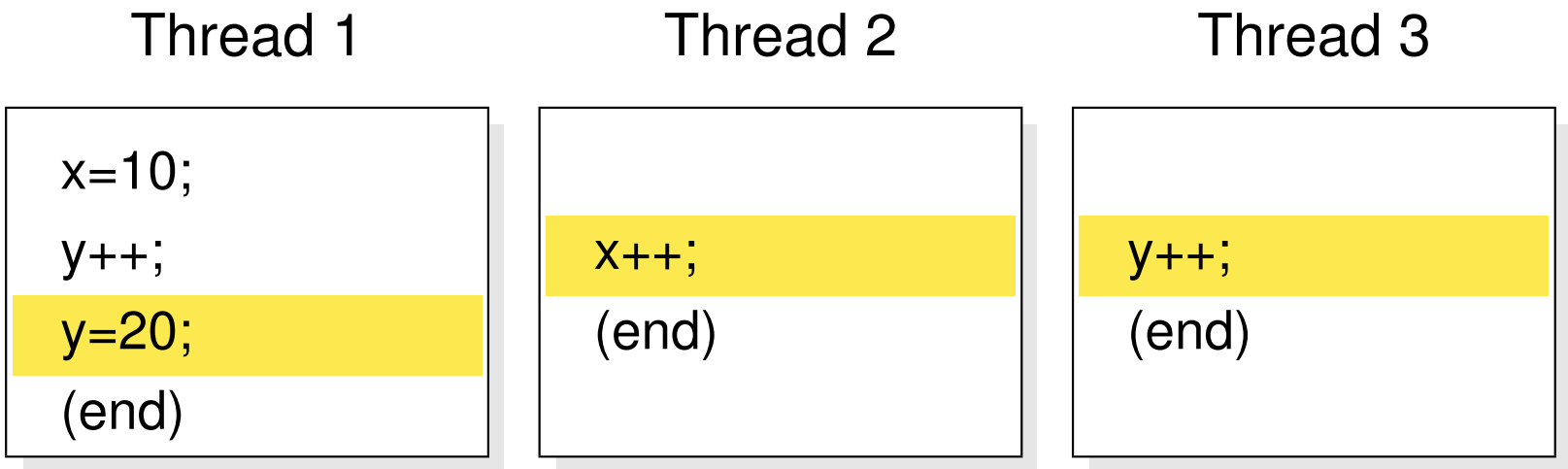
Current state: x=0, y=0

Thread Interleavings: Example (1)



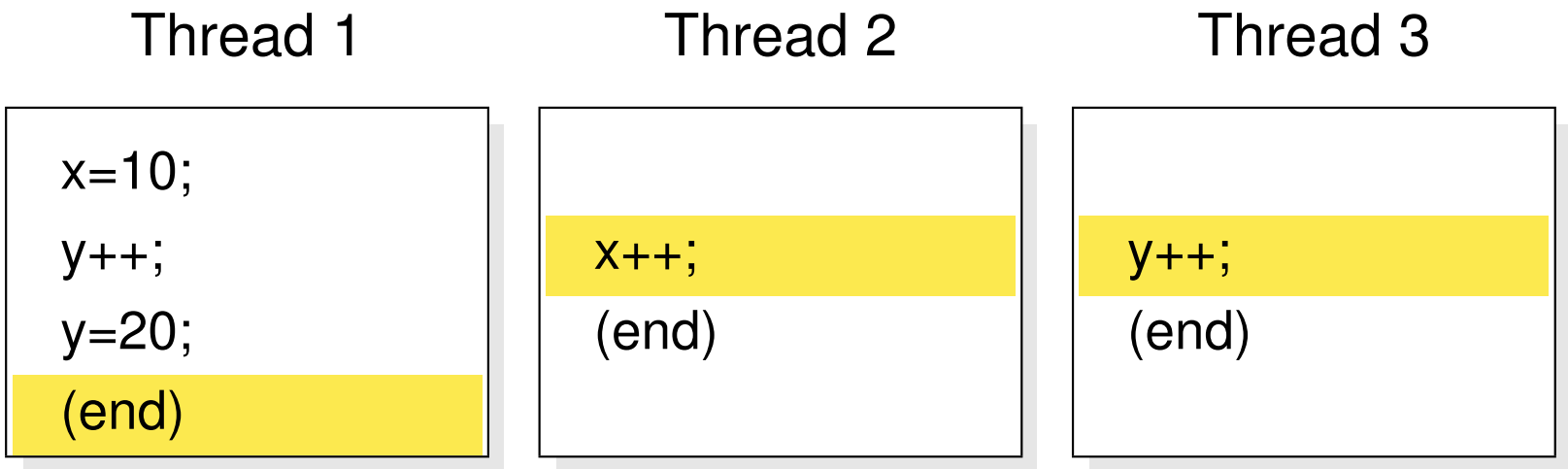
Current state: x=10, y=0

Thread Interleavings: Example (1)



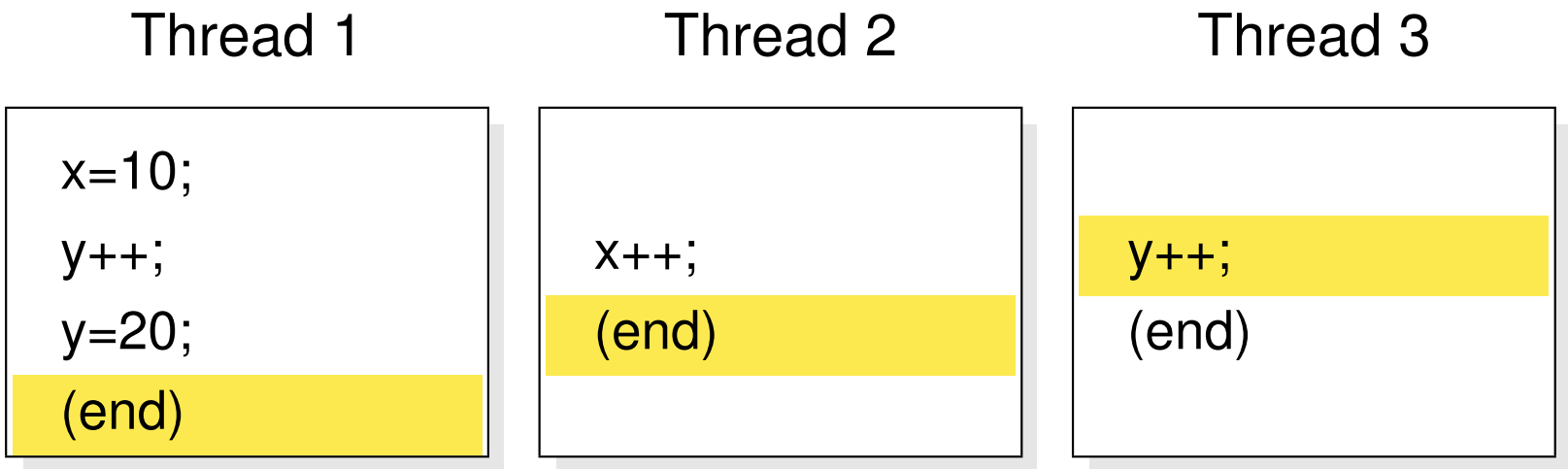
Current state: x=10, y=1

Thread Interleavings: Example (1)



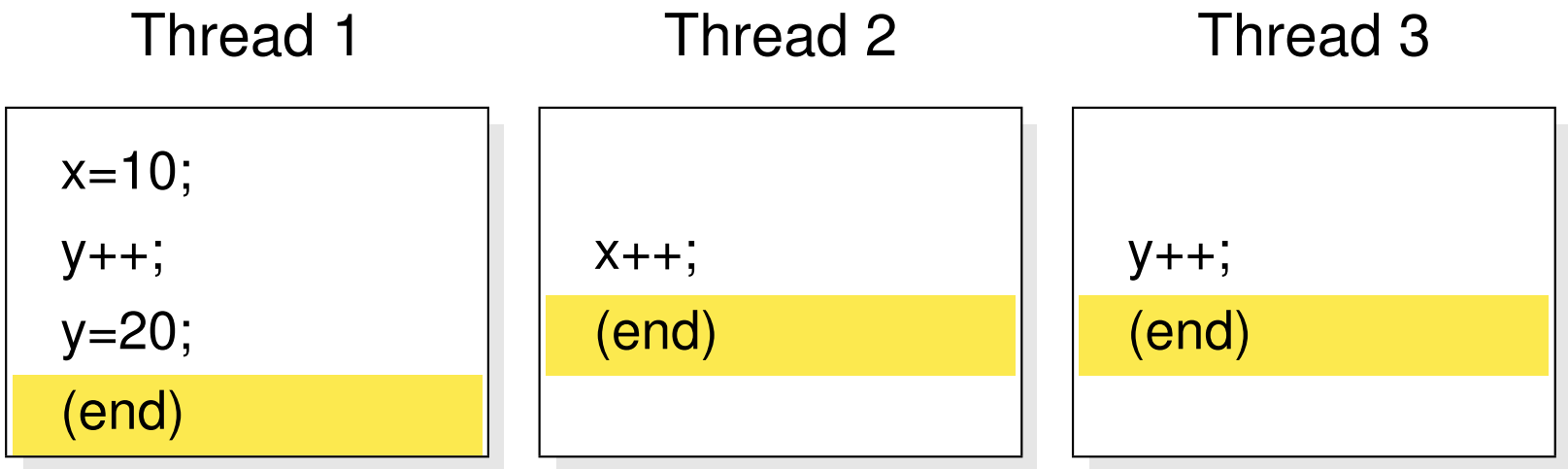
Current state: x=10, y=20

Thread Interleavings: Example (1)



Current state: x=11, y=20

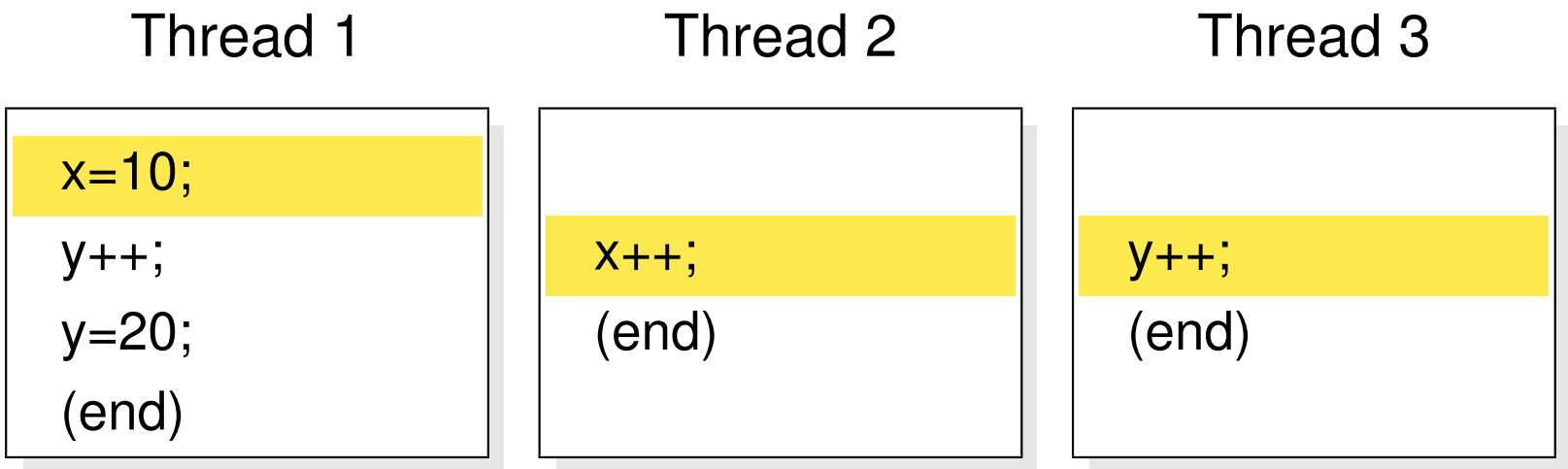
Thread Interleavings: Example (1)



Current state: x=11, y=21

Thread Interleavings: Example (2)

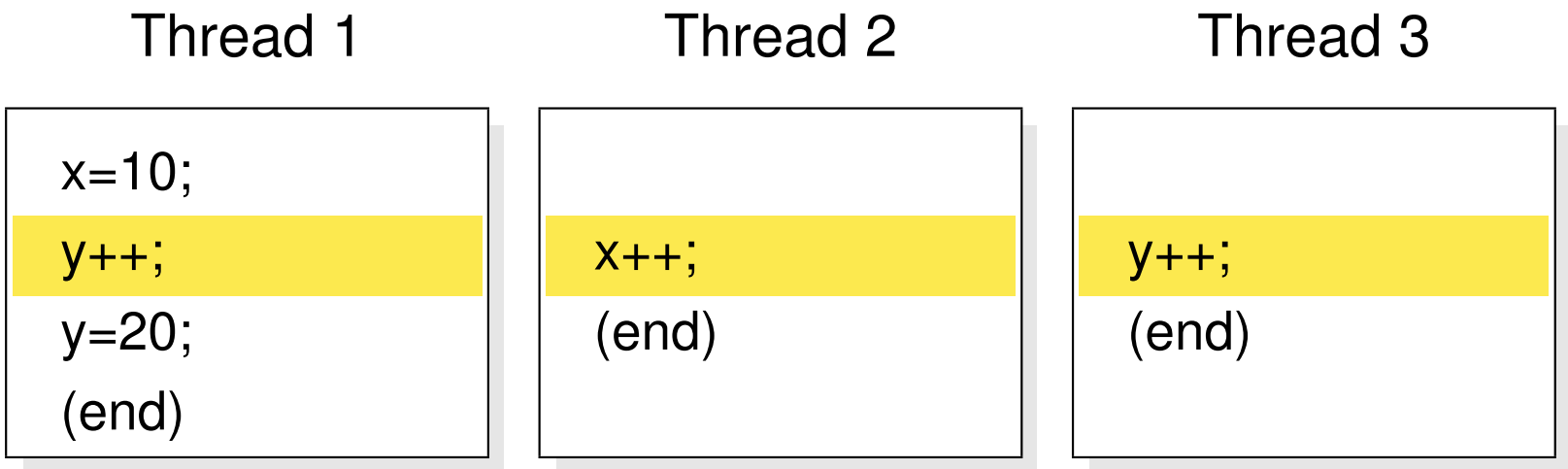
Alternative Schedule



Current state: x=0, y=0

Thread Interleavings: Example (2)

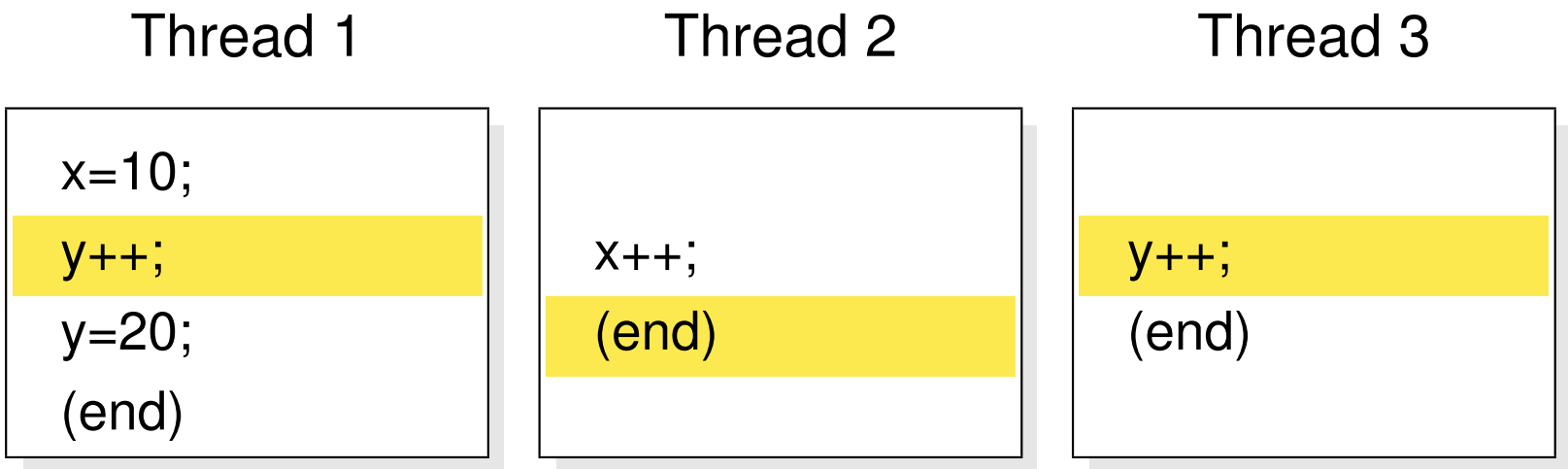
Alternative Schedule



Current state: x=10, y=0

Thread Interleavings: Example (2)

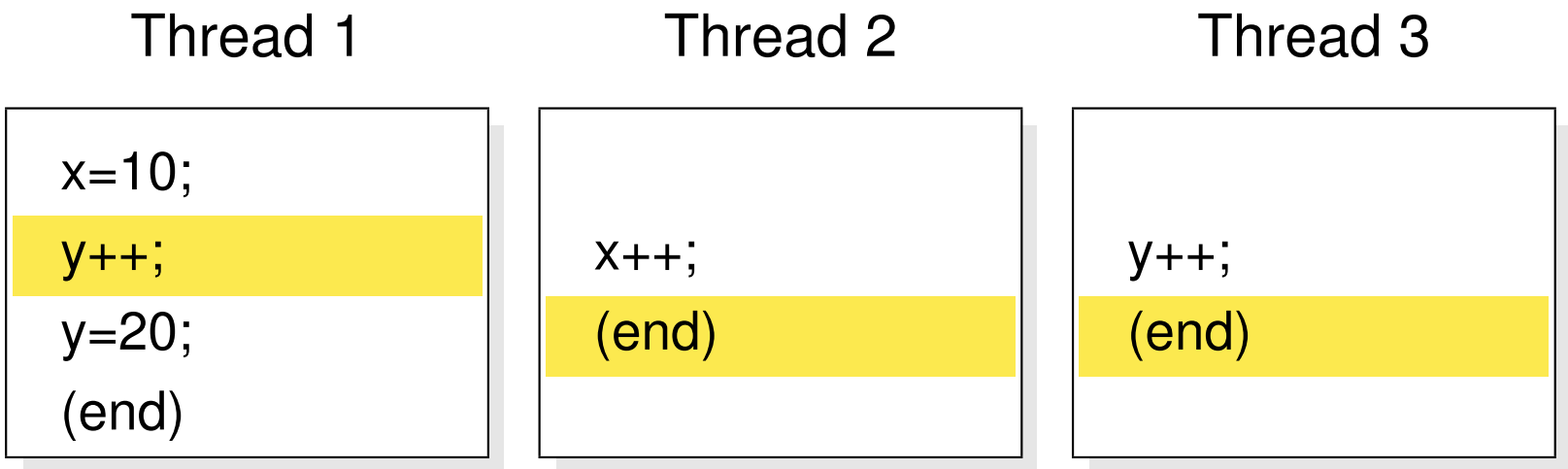
Alternative Schedule



Current state: x=11, y=0

Thread Interleavings: Example (2)

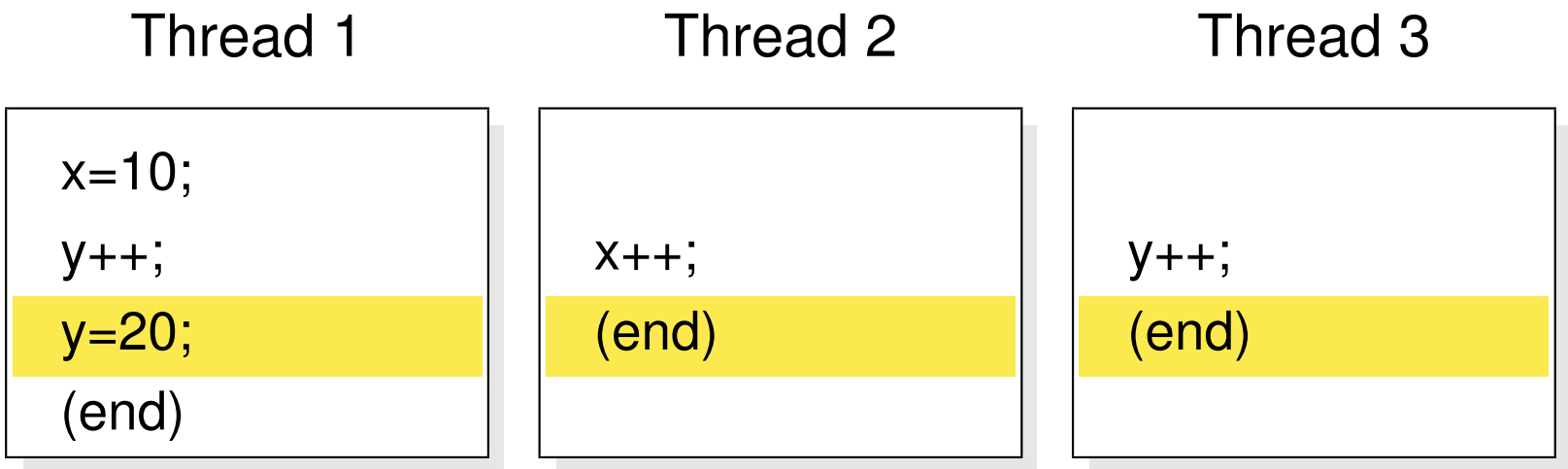
Alternative Schedule



Current state: x=11, y=1

Thread Interleavings: Example (2)

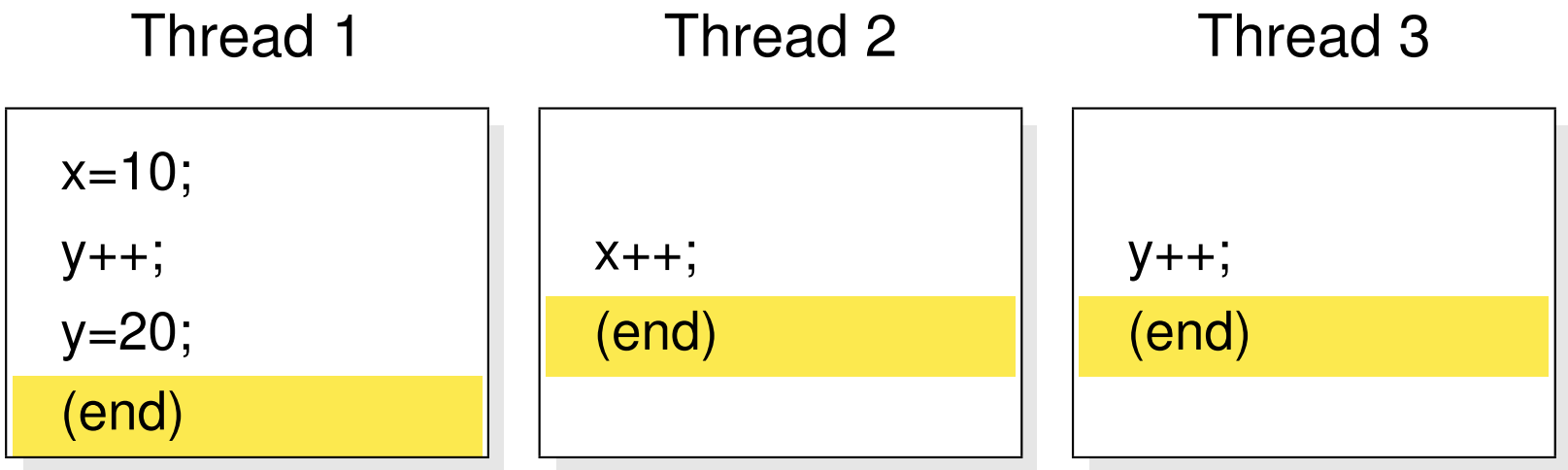
Alternative Schedule



Current state: x=11, y=2

Thread Interleavings: Example (2)

Alternative Schedule



Current state: x=11, y=20

Races

- The example program has a race, i.e., the result depends on the schedule

Races

- The example program has a race, i.e., the result depends on the schedule
- This may indicate a program bug, but need not
- Locks may help, but need not
- Enumerate? Exponential blowup!

Ordering of Loads/Stores

Thread 1	Thread 2
mov [x], 1	**mov** [y], 1
mov eax, [y]	**mov ebx**, [x]

- Suppose x and y are shared and initialized with 0

Ordering of Loads/Stores

Thread 1	Thread 2
mov [x], 1	**mov** [y], 1
mov eax, [y]	**mov ebx**, [x]

- Suppose x and y are shared and initialized with 0
- Unfortunately, the program may terminate with **eax**=**ebx**=0!

Shared Memory in Hardware

- ✘ Computer vendors do not guarantee atomicity of loads/stores

- ▶ Only very special commands are guaranteed to be atomic, e.g., the Compare-and-Swap (CAS) instruction

Shared Memory in Hardware

- ✘ Computer vendors do not guarantee atomicity of loads/stores
- Only very special commands are guaranteed to be atomic, e.g., the Compare-and-Swap (CAS) instruction
- Stores are not even guaranteed to be visible to other threads unless a memory barrier (memory fence) is inserted.
- ✘ But: memory fences are expensive ($>$100 cycles), so we cannot put them everywhere

Boolean Programs

Let's look at a Windows device driver example:

```
1  void DecrementIo(DEVICE_OBJECT * DeviceObject) {
2        EXT * ext = (EXT*)DeviceObject->DeviceExtension;
3
4        int  IoIsPending =
5          InterlockedDecrement (&ext->IoIsPending);
6
7        if  (!IoIsPending) {
8          KeSetEvent (&ext->event, IO_NO_INCREMENT, FALSE); }
9  }
```

Boolean Programs

The initial abstraction, without any predicates:

```
1  void DecrementIo() {
2        InterlockedDecrement();
3        goto L1,L2;
4        L1: KeSetEvent();
5        L2: return;
6  }
```

Adding Some Predicates

Some predicate refinement scheme computes a set of predicates:

$$
\begin{aligned}
\texttt{b1} &\triangleq \texttt{ext} == \&\texttt{envext} \\
\texttt{b2} &\triangleq \texttt{envext.IoIsPending} == 1 \\
\texttt{b3} &\triangleq \texttt{envext.IoIsPending} == 2 \\
\texttt{b4} &\triangleq \texttt{IoIsPending} == 2 \\
\texttt{b5} &\triangleq \texttt{IoIsPending} == 1 \\
\texttt{b6} &\triangleq (*\texttt{ext}).\texttt{IoIsPending} == 1 \\
\texttt{b7} &\triangleq (*\texttt{ext}).\texttt{IoIsPending} == 2
\end{aligned}
$$

Boolean Programs

This results in the following new abstract model:

```
1  bool b1,b2,b3; // global
2
3  void DecrementIo() {
4       bool b4,b5,b6,b7; // local
5       b1,b6,b7 = *,*,*
6        constrain((!(b1' && b2) || b6') &&
7                        (!( b1' && b3) || b7' ));
8       b4,b5 = InterlockedDecrement(b6,b7);
9       goto L1,L2;
10      L1: assume(!b4 && !b5);
11           KeSetEvent();
12      L2: return;
13 }
```

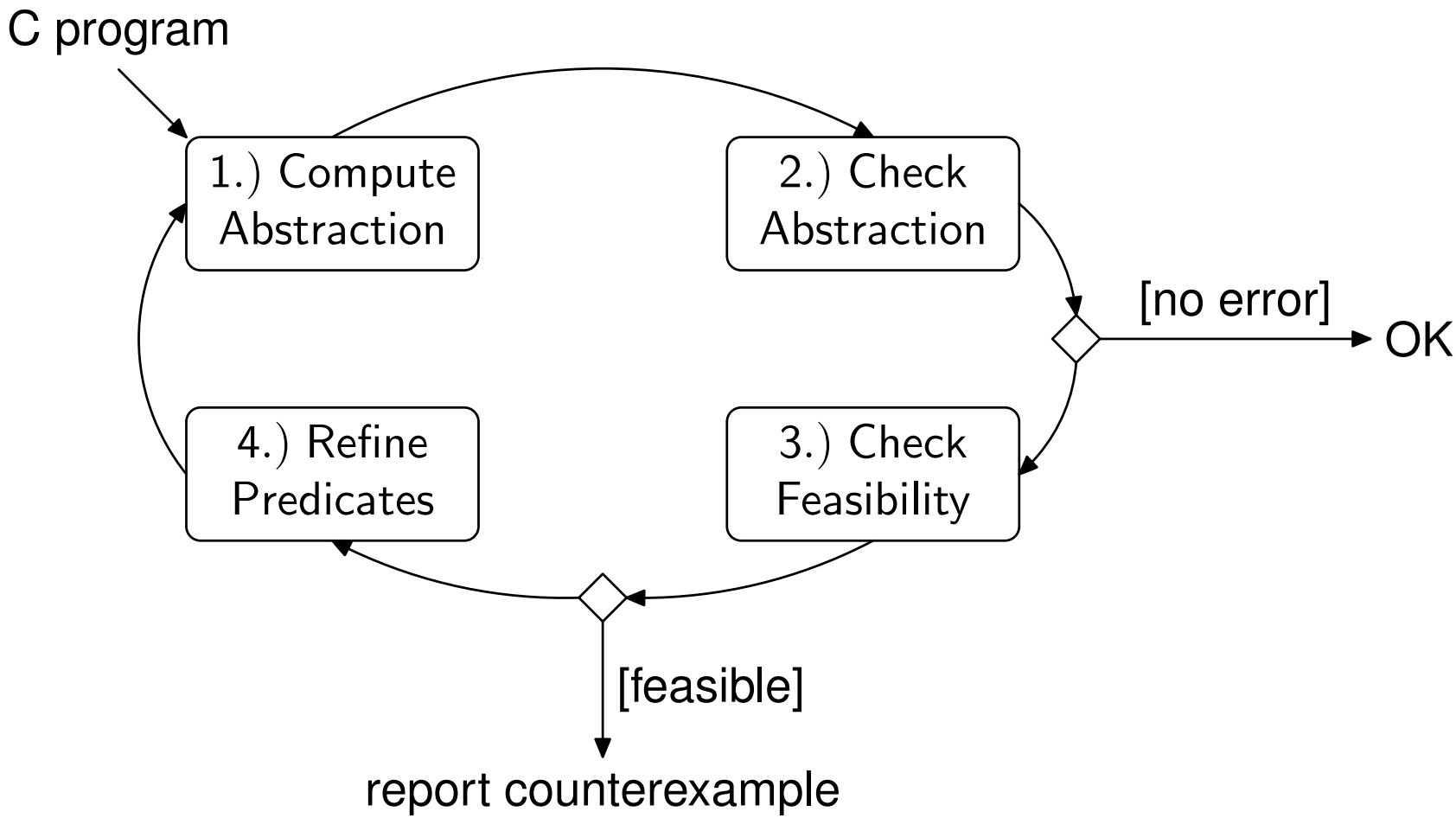
Concurrent Boolean Programs

Suppose we could dynamically create threads in Boolean Programs:

```
1  bool b1,b2,b3; // global
2
3  void DecrementIo() {
4        bool b4,b5,b6,b7; // local
5        b1,b6,b7 = *,*,*
6         constrain((!(b1' && b2) || b6') &&
7                        (!( b1' && b3) || b7' ));
8        b4,b5 = InterlockedDecrement(b6,b7);
9        start_thread L1, L2;
10       L1: assume(!b4 && !b5);
11           KeSetEvent();
12       L2: return;
13 }
```

Predicate Abstraction for Concurrent Programs

Can we apply our refinement loop to concurrent software?



Predicate Abstraction for Concurrent Programs

Claim: Yes, we can!

- Abstraction: as before!
- Checking $\hat{M}$: use Model Checker for concurrent Boolean programs
- Simulation: as before – use thread switches from abstract counterexample
- Refinement: as before!

Example

```
1  int g;

2  void *t1(void *arg) {
3    g=0;
4    if (g==1) {
5      g=2;
6      if (g==3)
7        g=4;
8    }
9  }

10 void *t2(void *arg) {
11   g=1;
12   if (g==2) {
13     g=3;
14     assert(g!=4);
15   }
16 }
```

Example

```
16 int main() {
17   pthread_t id1, id2;
18
19   pthread_create(&id1, NULL, t1, NULL);
20   pthread_create(&id2, NULL, t2, NULL);
21 }
```

Initial Abstraction

```
1  decl g_eq_4;

2  void t1 () begin
3     g_eq_4:=0;          //  g=0;
4     if * then begin     //  g==1
5        g_eq_4:=0;       //  g=2;
6        if * then        //  g==3
7           g_eq_4:=1;    //  g=4;
8     end
9  end

10 void t2 () begin
11    g_eq_4:=0;          //  g=1;
12    if * then begin     //  g==2
13       g_eq_4:=0;       //  g=3;
14       assert(!g_eq_4); //  g!=4
15    end
16 end

17 void main() begin
18    ...
19    g_eq_4:=0;
20    ...
21    start_thread ...
22 end
```

Abstract Counterexample

Thread	Line	Instruction
0	19	g_eq_4 := 0;
2	11	g_eq_4 := 0;
2	12	**if** $*$ **then** ...
2	13	g_eq_4 := 0;
1	3	g_eq_4 := 0;
1	4	**if** $*$ **then** ...
1	5	g_eq_4 := 0;
1	6	**if** $*$ **then** ...
1	7	g_eq_4 := 1;
2	14	**assert**(!g_eq_4);

Abstract Counterexample

Thread	Line	Instruction	Concrete Instr.	SSA-Constraint
0	19	g_eq_4 := 0;	g=0;	$g_0 = 0$
2	11	g_eq_4 := 0;	g=1;	$\wedge g_1 = 1$
2	12	**if** $*$ **then** ...	**if** (g==2)	$\wedge g_1 = 2$
2	13	g_eq_4 := 0;	g=3;	$\wedge g_2 = 3$
1	3	g_eq_4 := 0;	g=0;	$\wedge g_3 = 0$
1	4	**if** $*$ **then** ...	**if** (g==1)	$\wedge g_3 = 1$
1	5	g_eq_4 := 0;	g=2;	$\wedge g_4 = 2$
1	6	**if** $*$ **then** ...	**if** (g==3)	$\wedge g_4 = 3$
1	7	g_eq_4 := 1;	g=4;	$\wedge g_5 = 4$
2	14	**assert**(!g_eq_4);	**assert**(g!=4);	$\wedge g_5 = 4$

Example

- In total, 4 iterations, and 4 predicates:
  g==4, g==3, g==1, g==2

- Final counterexample has a thread switch
  after each assignment to g

Tricky Details

```
1  volatile  unsigned value;
2
3  unsigned NonblockingCounter__increment() {
4        unsigned v = 0;
5
6        lock ();
7        if (value == 0u-1) {
8              unlock();
9              return  0;
10       }else{
11             v = value;
12             value = v + 1;
13             unlock();
14             assert(value > v);
15             return  v + 1;
16       }
17 }
```

Tricky Details

```
1  volatile  unsigned value;
2
3  unsigned NonblockingCounter__increment() {
4        unsigned v = 0;
5
6        lock ();
7        if (value == 0u-1) {
8              unlock();
9              return  0;
10       }else{
11             v = value;
12             value = v + 1;
13             unlock();
14             assert(value > v);
15             return  v + 1;
16       }
17 }
```

Is "value>v" global or local?

Tricky Details

```
1  volatile  unsigned value;
2
3  unsigned NonblockingCounter__increment() {
4        unsigned v = 0;
5
6        lock ();
7        if (value == 0u-1) {
8              unlock();
9              return  0;
10       }else{
11             v = value;
12             value = v + 1;
13             unlock();
14             assert(value > v);
15             return  v + 1;
16       }
17 }
```

Is "value>v" global or local?

CAV 2011: neither. We need broadcast!

The Problem We Consider

UNIVERSITY OF OXFORD

Definition

Consider thread state (s, ℓ).

thread state reachability problem:

Is there a reachable global state $(s, \ell_1, \ldots, \ell_n)$ such that $\ell = \ell_i$ for some i?

Note:

- can express single-index properties of threads
- can also encode MutEx: using shared variable as monitor

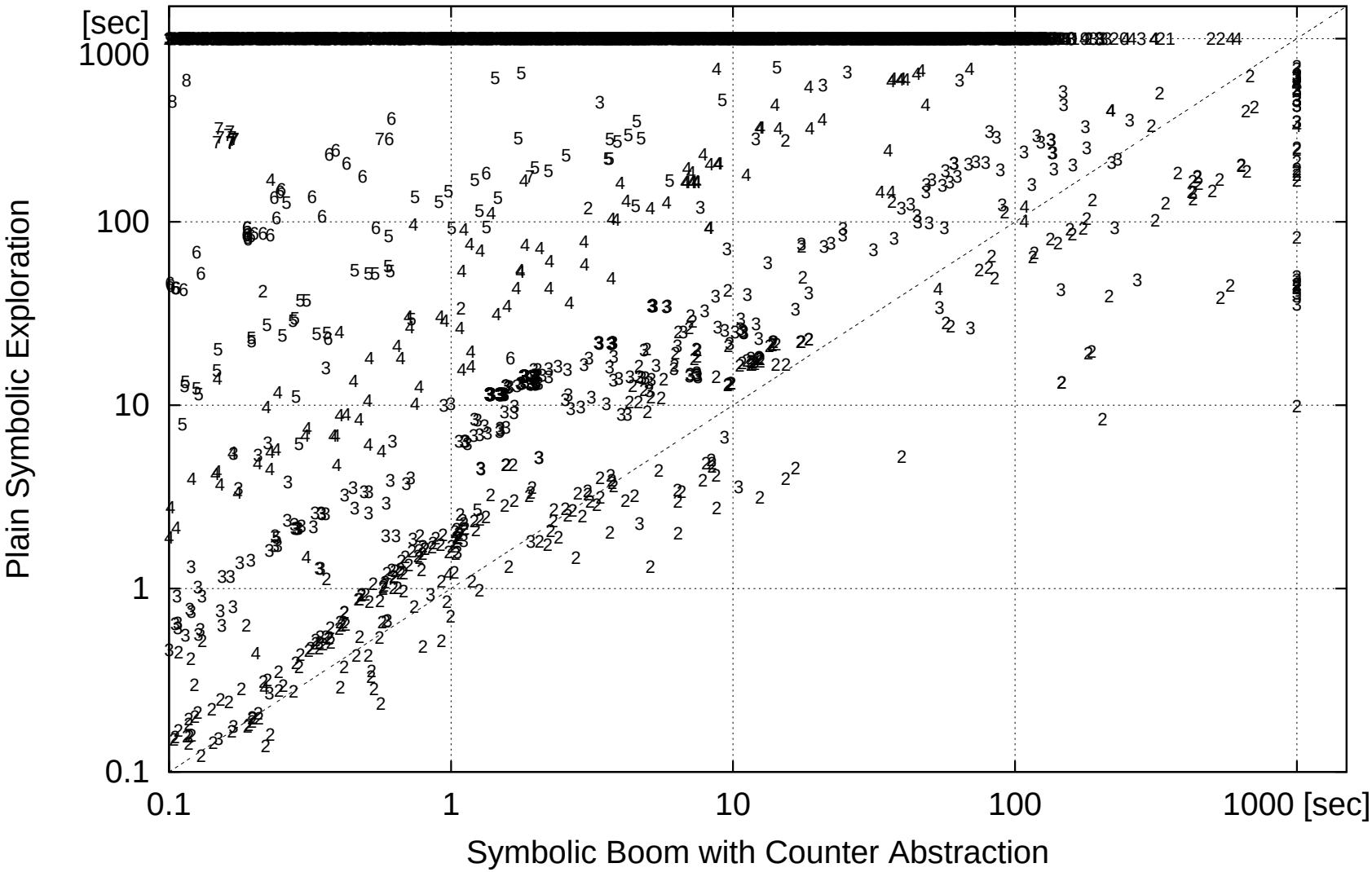
Boolean Programs with Bounded Replication

UNIVERSITY OF OXFORD

- We have an effective Model Checker
  for Boolean programs with a bounded number of threads
- BDD-based
- Exploits symmetry between threads

[CAV 2009, FMSD 2010]

Symbolic Experiments



Boolean Programs with Unounded Replication

But we really want an unbounded number of threads

- Programs with a large number of theads ($>$10)
- Server with dynamic thread spawning
  (say dependent on load)
- Or when thread-count is lost
  during predicate abstraction!

Our answer: reduction to Petri net coverability

Experimental Results

1. Boolean programs
   - Proper concurrent ones (extracted from Linux kernel)
   - “Pseudo” concurrent ones (SLAM)

2. Petri Net benchmarks
   - Bounded and unbounded
   - Bingham and Ganty/Begin/Delzanno/Raskin
   - Comparison with Lola (Karsten Wolf) and MIST

Experimental Results

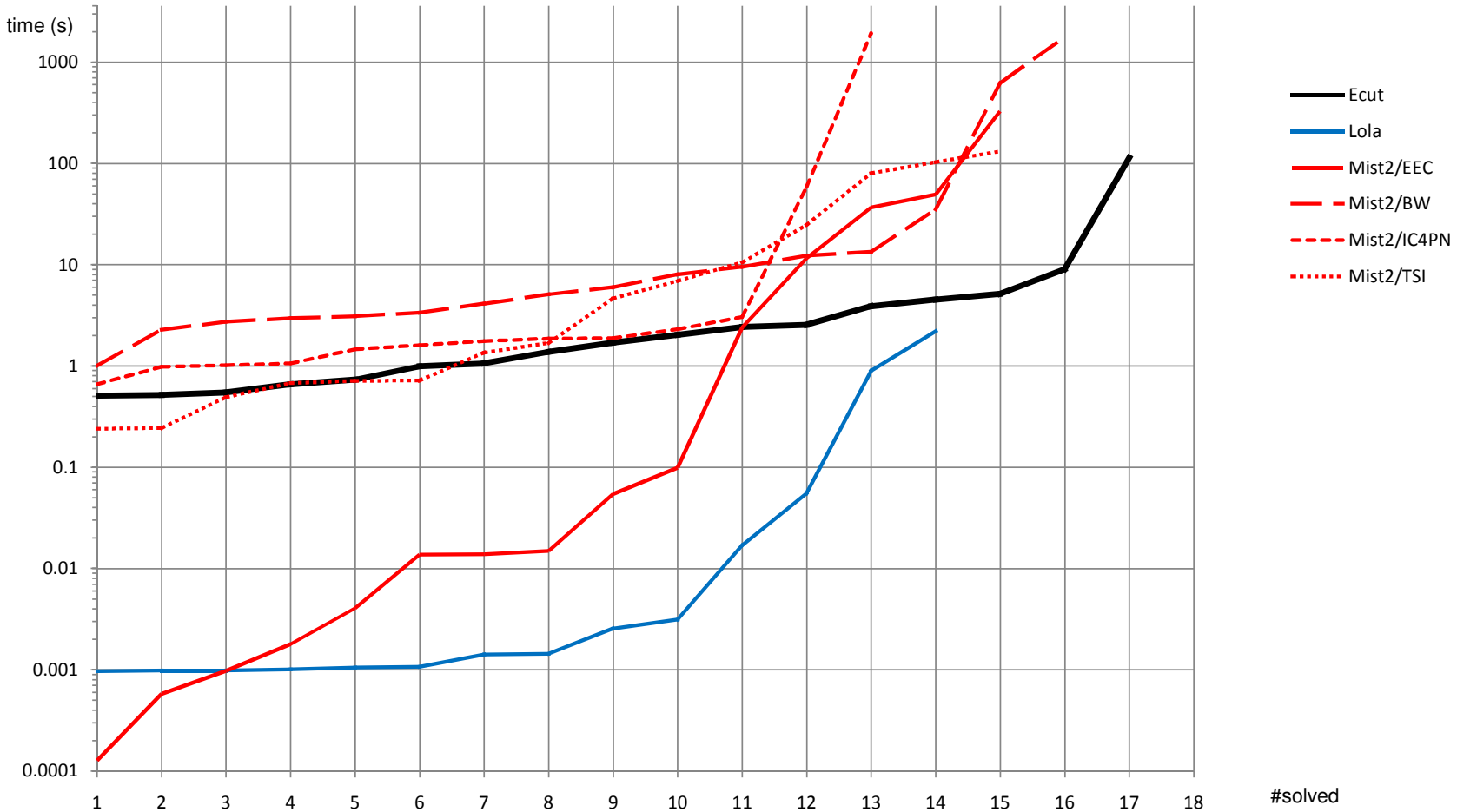
On Boolean Programs:

#P	Sh	Lcs	Loc	Time	c	Safe	Unsafe
773	17	8	1170	0.1	1	407	366
17	21	22	1139	0.8	2	3	14
8	13	26	1131	72.3	3	8	0
54	18	31	1267	874.0	?	–	–

We now need to make harder models!

Experimental Results

Checking coverability on standard Petri Net benchmarks:

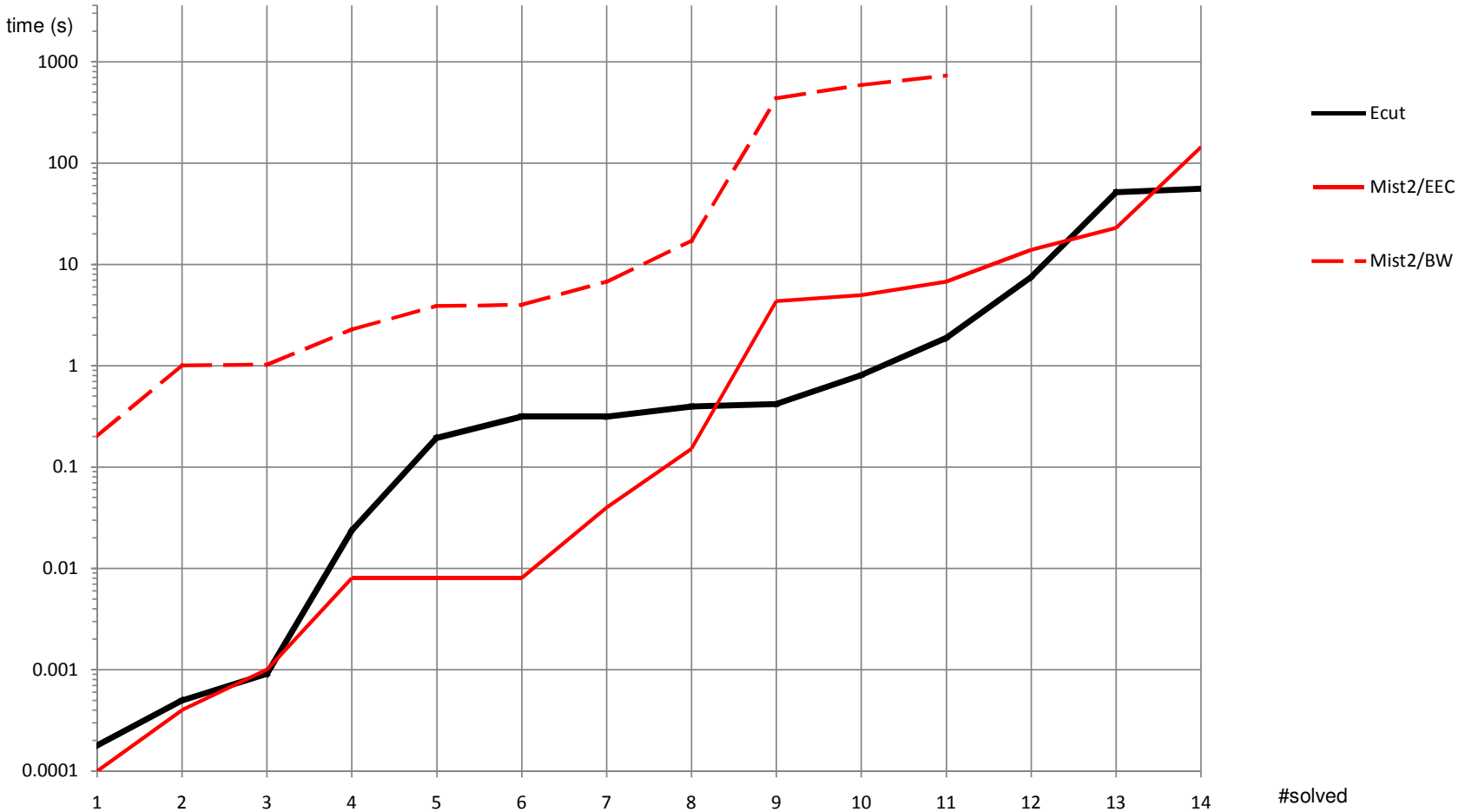


Number of Petri net instances solved by the different tools (hor.), runtime (vert., log. scale); timout: 1h, memory limit: 24g
Ecut: 17, Lola: 14, Mist2/EEC: 15, Mist2/BW: 16, Mist2/IC4PN: 13, Mist2/TSI: 15

Comparison with numerous algorithms implemented in MIST

Experimental Results

Checking coverability on Petri Nets *with transfer arcs*:



Number of Transfer Petri net instances solved by the different tools (hor.), runtime (vert., log. scale); timout: 1h, memory limit: 24g
Ecut: 14, Mist2/EEC: 14, Mist2/BW: 11

Comparison with two algorithms implemented in MIST

Conclusion

- Model Checking for C programs with unbounded threads
- Symmetry-aware abstraction to VASS/Petri net coverability (with transfer arcs)

✔ Enables verification of C programs with unbounded concurrency

Watch the video!

SATABS

Download me!

http://www.cprover.org/satabs/
http://www.cprover.org/SSFT12/