Model Checking with **Maximal Causality Reduction**

Jeff Huang

Assistant Professor

Parasol

COMPUTER SCIENCE & ENGINEERING
TEXAS A&M UNIVERSITY

A Real Bug – $12 million loss of equipment

https://stackoverflow.com/questions/16159203/why-does-this-java-program-terminate-despite-that-apparently-it-shouldnt-and-d

```
curPos = new Point(1,2);
class Point {  int x, y; }

Thread 1:
newPos = new
        Point(curPos.x+1, curPos.y+1);

Thread 2:
while (newPos != null)
if (newPos.x+1 !=
newPos.y)
    ERROR
```

A Real Bug – $12 million loss of equipment

https://stackoverflow.com/questions/16159203/why-does-this-java-program-terminate-despite-that-apparently-it-shouldnt-and-d

```
curPos = new Point(1,2);
class Point {  int x, y; }

Thread 1:
newPos = new
        Point(curPos.x+1, curPos.y+1);

Thread 2:
while (newPos != null)
if (newPos.x+1 != 
newPos.y)
    ERROR
```

```
x=0
y=0
x=curPos.x+1
y=curPos.y+1
newPos = object
```

statements are out of program order

Maximal Causality Reduction

- Open source: https://github.com/parasol-aser/JMCR
- Implementation
  - Java 8, multi-threading
  - The Z3 SMT solver
- Evaluation
  - Takes only two runs to find the error in <1s
  - **Orders of magnitude** more effective than partial order reduction and bounded model checking
  - Finding **new errors** (data races and NPEs) in extensively studied popular benchmarks

2007 Turing Award

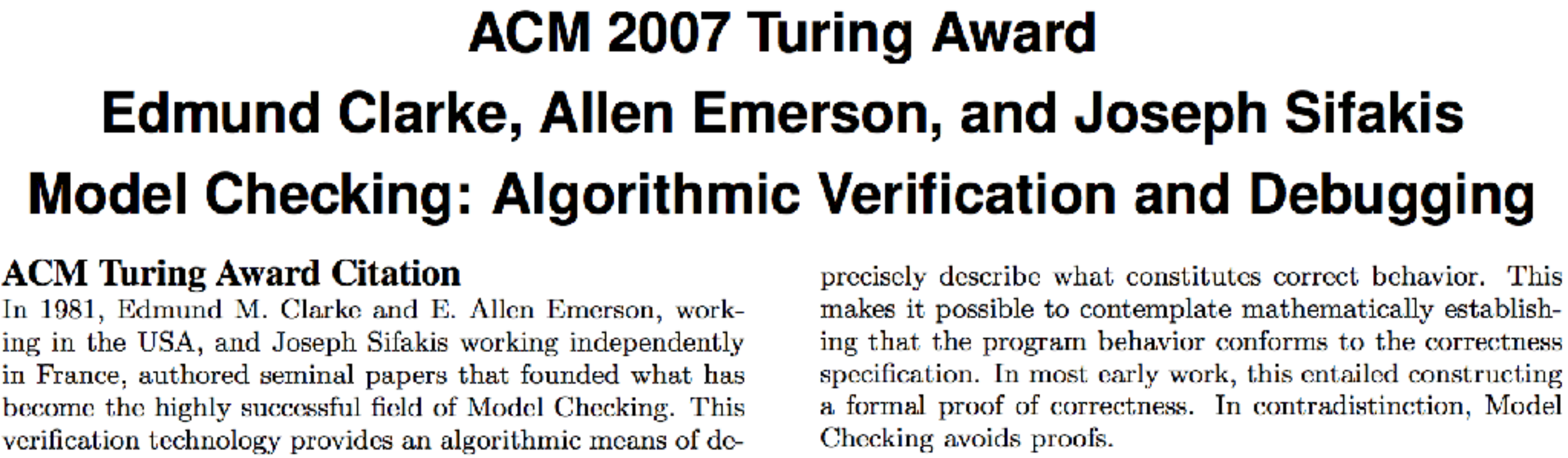
Edmund Clarke

Allen Emerson

Joseph Sifakis

For their role in "*developing Model-Checking into a highly effective verification technology …*"

The Key Challenge: State Explosion



ACM 2007 Turing Award

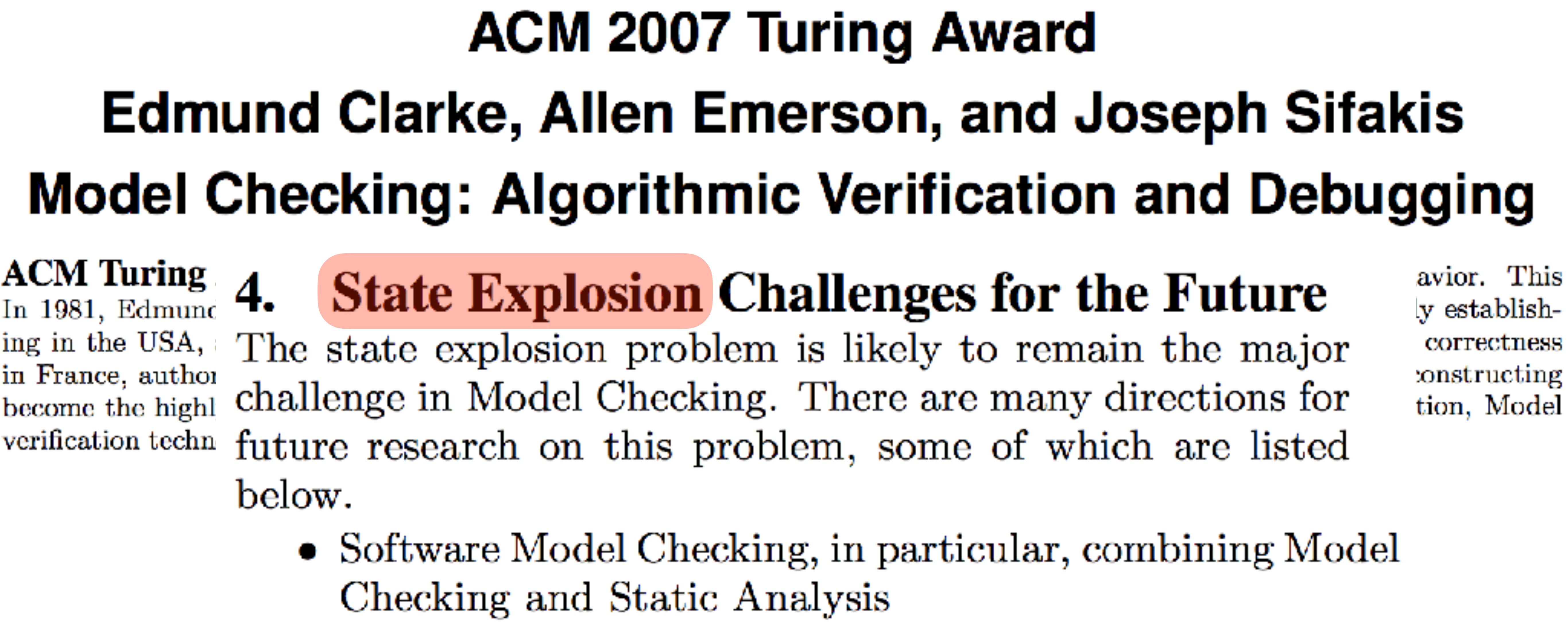
Edmund Clarke, Allen Emerson, and Joseph Sifakis

Model Checking: Algorithmic Verification and Debugging

ACM Turing Award Citation

In 1981, Edmund M. Clarke and E. Allen Emerson, working in the USA, and Joseph Sifakis working independently in France, authored seminal papers that founded what has become the highly successful field of Model Checking. This verification technology provides an algorithmic means of determining whether an abstract model--representing, for example, a hardware or software design--satisfies a formal specification expressed as a temporal logic formula. Moreover, if the property does not hold, the method identifies a counterexample execution that shows the source of the problem.

The Key Challenge: State Explosion



ACM 2007 Turing Award

Edmund Clarke, Allen Emerson, and Joseph Sifakis

Model Checking: Algorithmic Verification and Debugging

ACM Turing
In 1981, Edmunc
ing in the USA,
in France, authoi
become the highl
verification techn

4. State Explosion Challenges for the Future

The state explosion problem is likely to remain the major challenge in Model Checking. There are many directions for future research on this problem, some of which are listed below.

- Software Model Checking, in particular, combining Model Checking and Static Analysis

avior. This
y establish-
correctness
onstructing
tion, Model

Two Classical Approaches

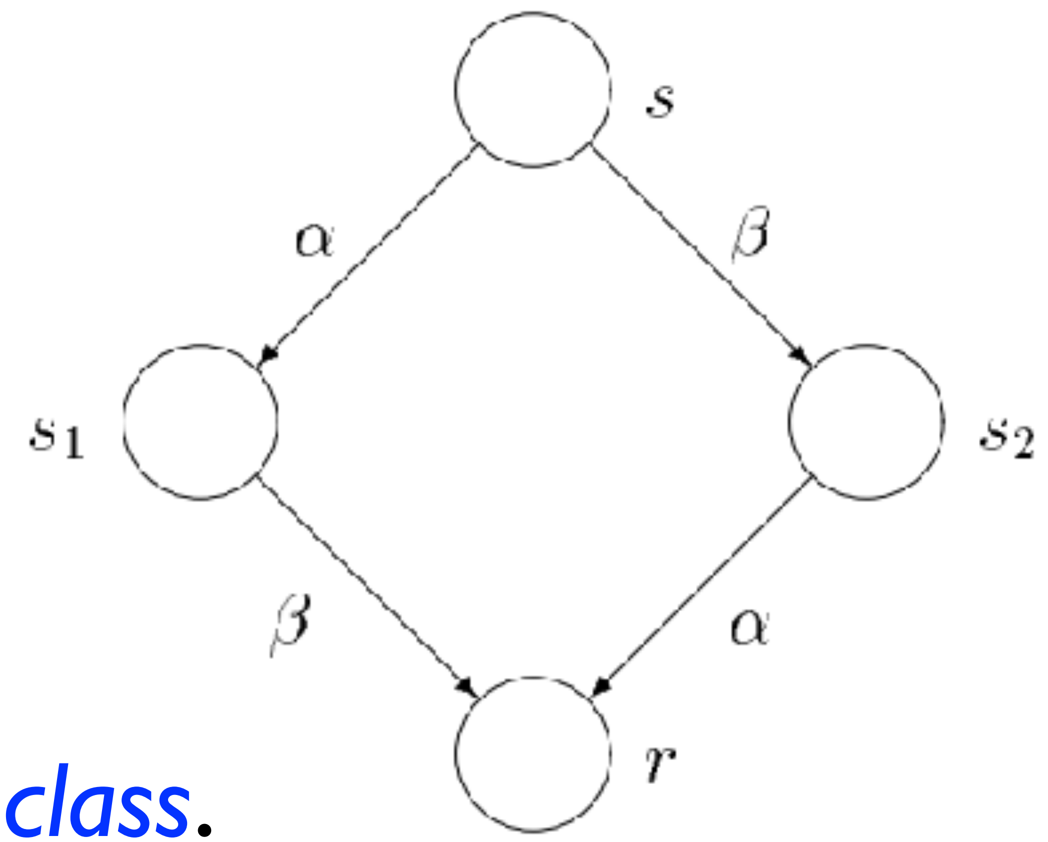
- Partial Order Reduction [2014 CAV Award] to Godefroid, Peled, Valmari, and Wolper
  - Reduce the size of the state space that needs to be searched
  - Exploit the *independence* between concurrently executed transitions, which result in the same state
- Bounded Model Checking Clarke, Biere, Raimi, Zhu (2001)
  - Limit the searched space to a certain bound

Partial Order Reduction

The two sequences

- s→α→s$_1$→β→r
- s→β→s$_2$→α→r

belong to the *same equivalent class*.



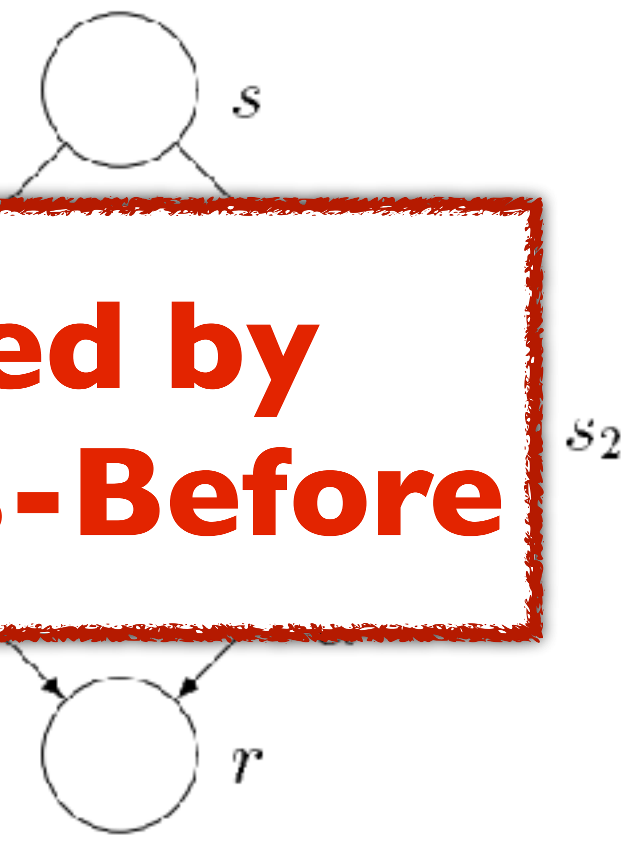
If the specification does not distinguish between these sequences, it is beneficial to consider only one with 2 + 1 states.

Partial Order Reduction

The two sequences

- s→α→s$_1$→β→r
- s→β→s$_2$→α→r

belong to the *same equivalent class*.

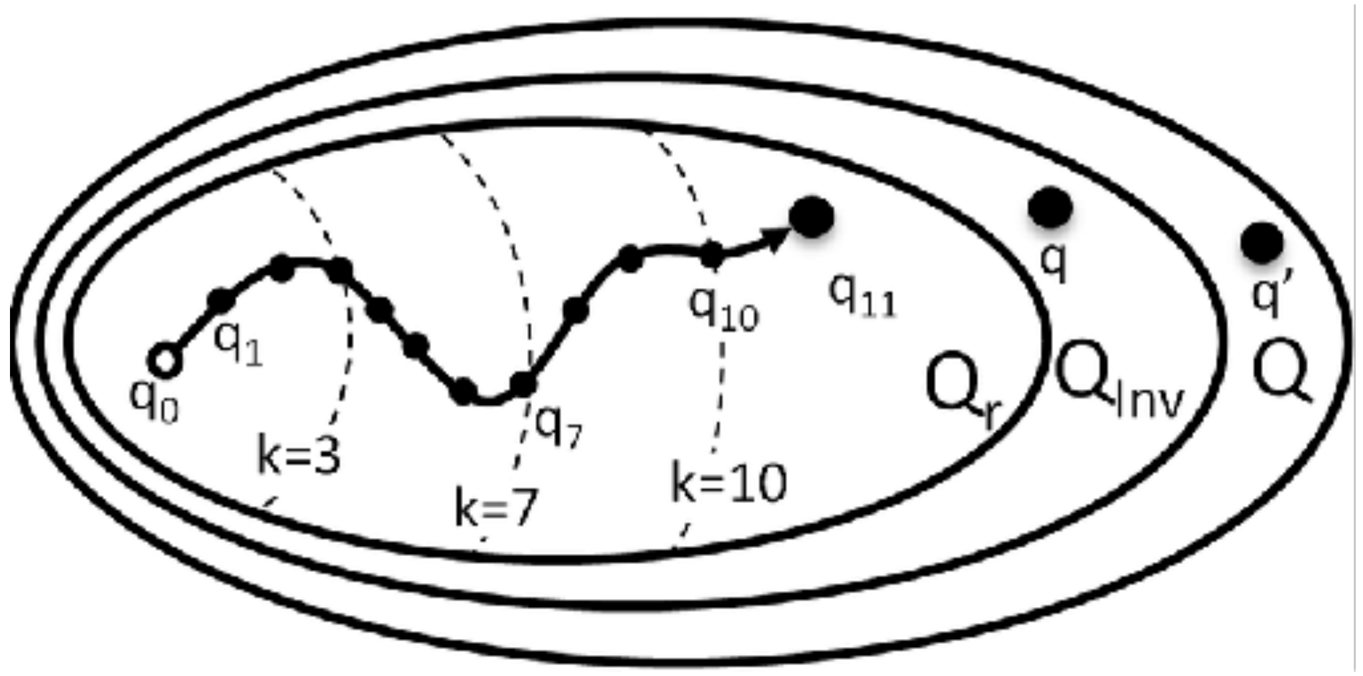


Limited by Happens-Before

If the specification does not distinguish between these sequences, it is beneficial to consider only one with 2 + 1 states.

Bounded Model Checking

- Restrict search to states that are reachable from initial state within fixed number k of transitions



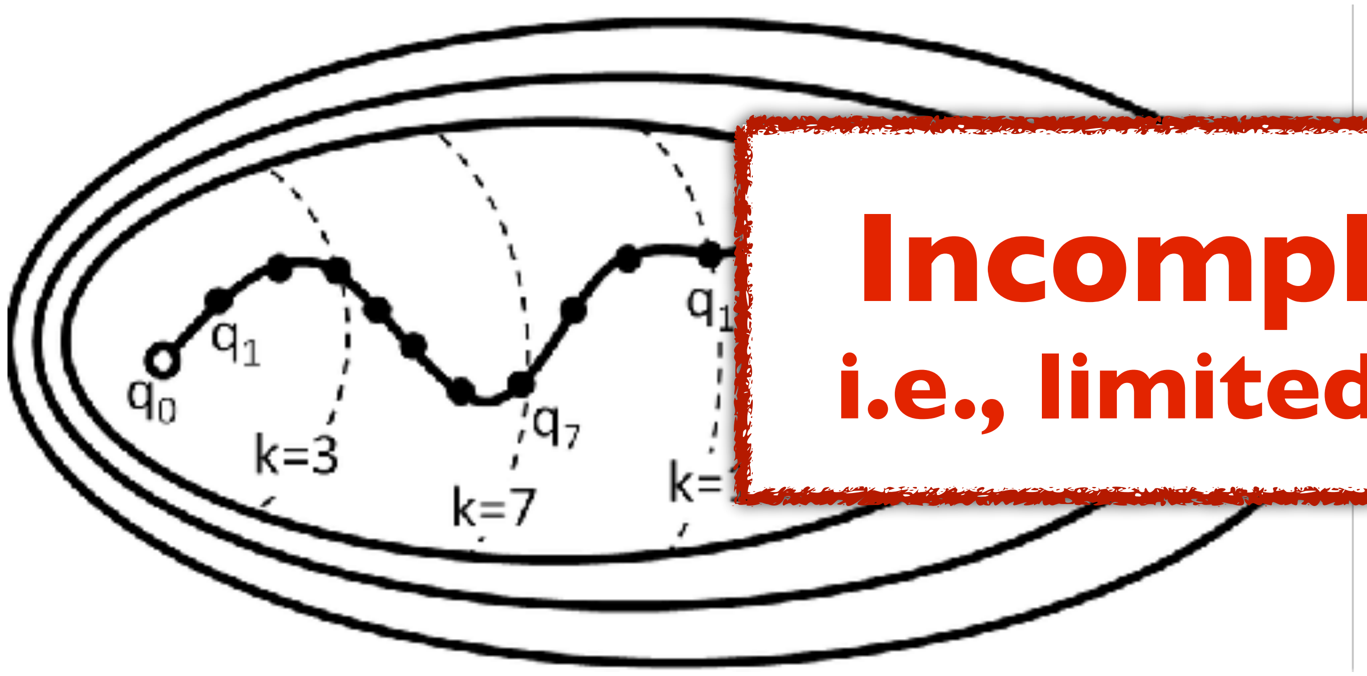
Can the given property fail in k-steps?

$I(V_0)$ Æ $T(V_0,V_1)$ Æ … Æ $T(V_{k-1},V_k)$ Æ (: $P(V_0)$ Ç…Ç: $P(V_k)$)

Initial state **k-steps** **Property fails in some step**

Bounded Model Checking

- Restrict search to states that are reachable from initial state within fixed number k of transitions



Incomplete
i.e., limited to k

Can the given property fail in k-steps?

I(V_0) Æ T(V_0,V_1) Æ … Æ T(V_{k-1},V_k) Æ (: P(V_0) Ç…Ç: P(V_k))

Initial state **k-steps** **Property fails in some step**

Example

initially **x**=**y**=0

T1

loop twice:

```
1:  lock(l)
2:  x=1
3:  y=1
4: unlock(l)
```

T2

loop twice:

```
5: lock(l)
6: x=0
7: unlock(l)
8: if(x>0)
9:     y++
10:    x=2
```

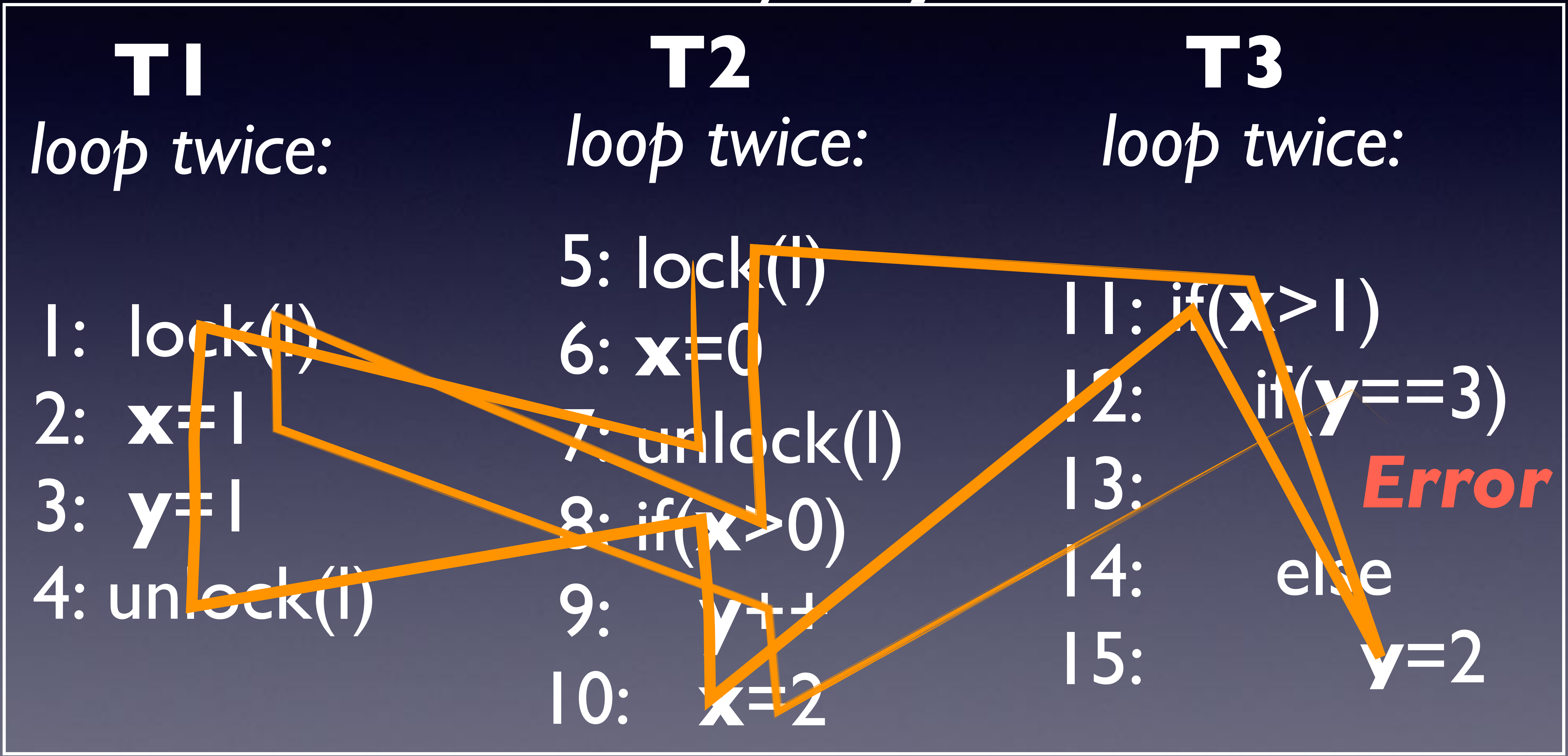
T3

loop twice:

```
11: if(x>1)
12:     if(y==3)
13:         Error
14:     else
15:         y=2
```

Example

initially **x**=**y**=0



T1

loop twice:

```
1:  lock(l)
2:  x=1
3:  y=1
4: unlock(l)
```

T2

loop twice:

```
5: lock(l)
6: x=0
7: unlock(l)
8: if(x>0)
9:     y++
10:    x=2
```

T3

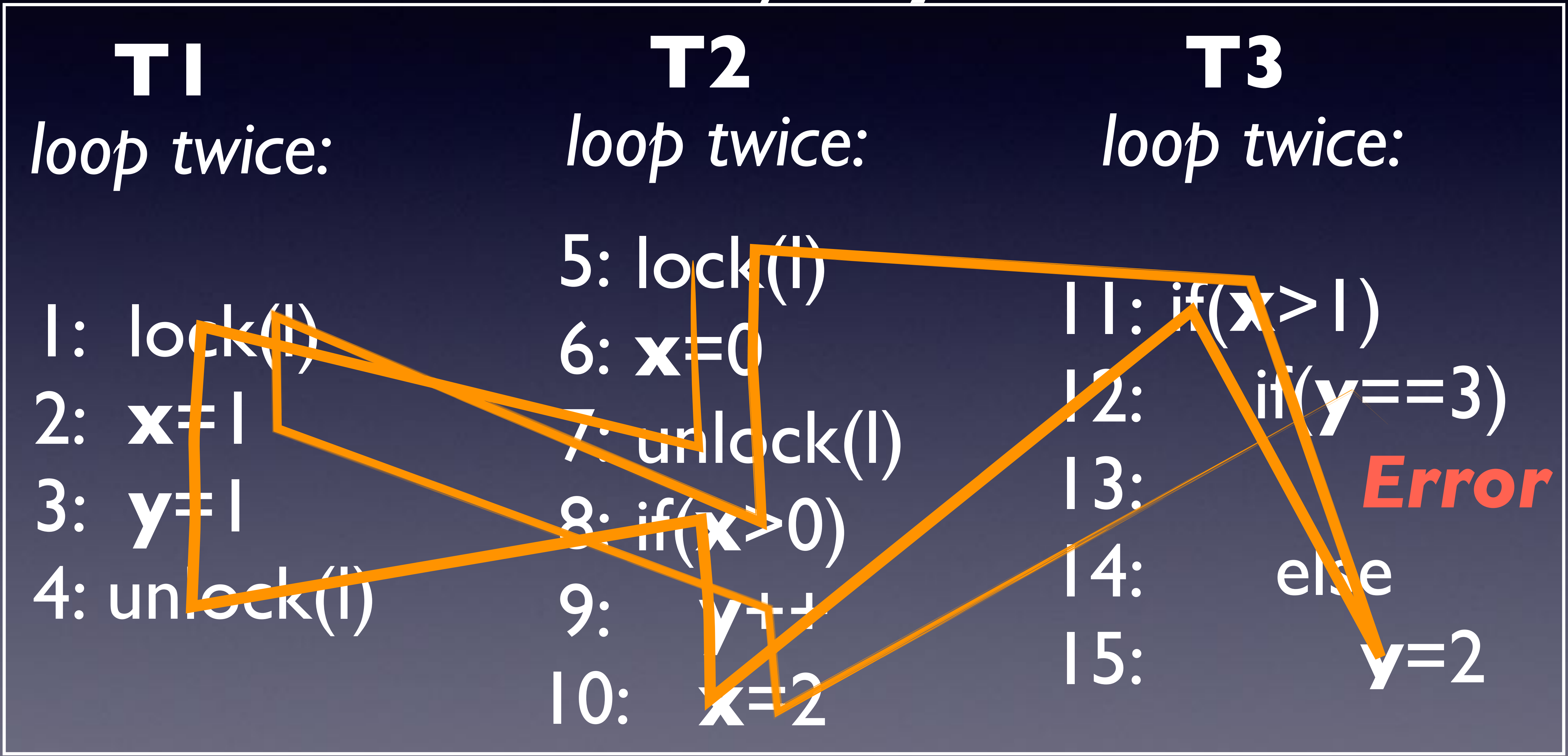
loop twice:

```
11: if(x>1)
12:     if(y==3)
13:         Error
14:     else
15:         y=2
```

T2T2T2 - T1T1T1T1 - T2T2T2T2 - T3T3T3 - T2T2T2T2 - T1T1 - T2T2T2T2 - T3T3

Example

initially **x**=**y**=0



T1

loop twice:

```
1:  lock(l)
2:  x=1
3:  y=1
4: unlock(l)
```

T2

loop twice:

```
5: lock(l)
6: x=0
7: unlock(l)
8: if(x>0)
9:     y++
10:    x=2
```

T3

loop twice:

```
11: if(x>1)
12:     if(y==3)
13:         Error
14:     else
15:         y=2
```

T2T2T2 - T1T1T1T1 - T2T2T2T2 - T3T3T3 - T2T2T2T2 - T1T1 - T2T2T2T2 - T3T3

7 thread context switches

Example

initially **x**=**y**=0

T1	**T2**	**T3**
loop twice:	*loop twice:*	*loop twice:*

DFS explores 3,293,931 runs

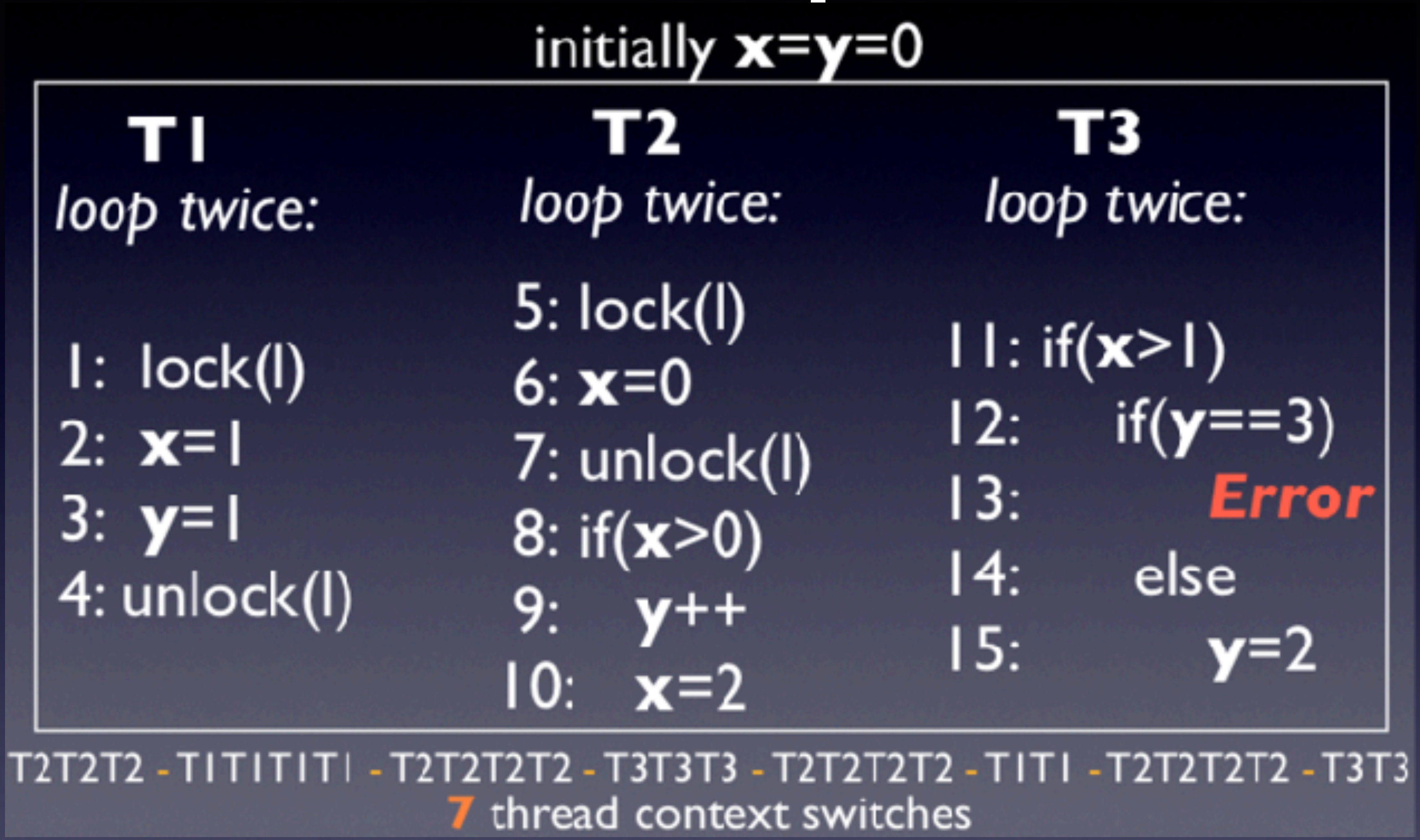
in an hour without finding the error

3: **y**=1	8: if(**x**>0)	13: Error
4: unlock(l)	9: **y**++	14: else
	10: **x**=2	15: **y**=2

T2T2T2 - T1T1T1T1T1 - T2T2T2T2 - T3T3T3 - T2T2T2T2 - T1T1 - T2T2T2T2 - T3T3

7 thread context switches

Example



initially **x**=**y**=0

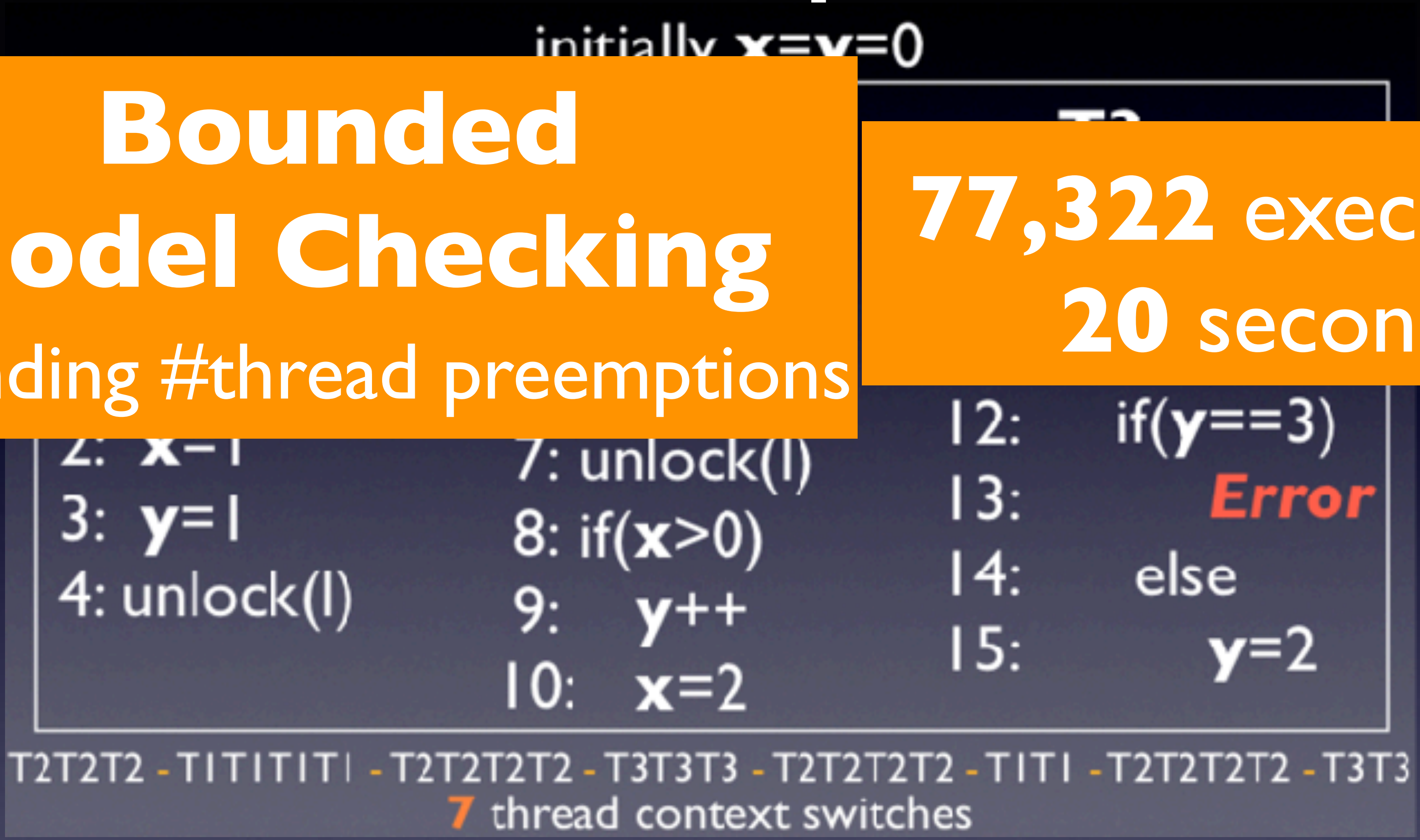
```
T1                  T2                  T3
loop twice:         loop twice:         loop twice:

                    5: lock(l)
                    6: x=0              11: if(x>1)
1:  lock(l)         7: unlock(l)        12:     if(y==3)
2:  x=1             8: if(x>0)          13:         Error
3:  y=1             9:    y++           14:     else
4: unlock(l)        10:   x=2           15:         y=2
```

T2T2T2 - T1T1T1T1 - T2T2T2T2 - T3T3T3 - T2T2T2T2 - T1T1 - T2T2T2T2 - T3T3

7 thread context switches

Example



initially **x**=**y**=0

```
T3
2:  x=1
3:  y=1            10:  x=2         15:        y=2
4: unlock(l)       9:   y++         14:     else
                   8: if(x>0)       13:          Error
                   7: unlock(l)     12:     if(y==3)
```

T2T2T2 - T1T1T1T1 - T2T2T2T2 - T3T3T3 - T2T2T2T2 - T1T1 - T2T2T2T2 - T3T3

7 thread context switches

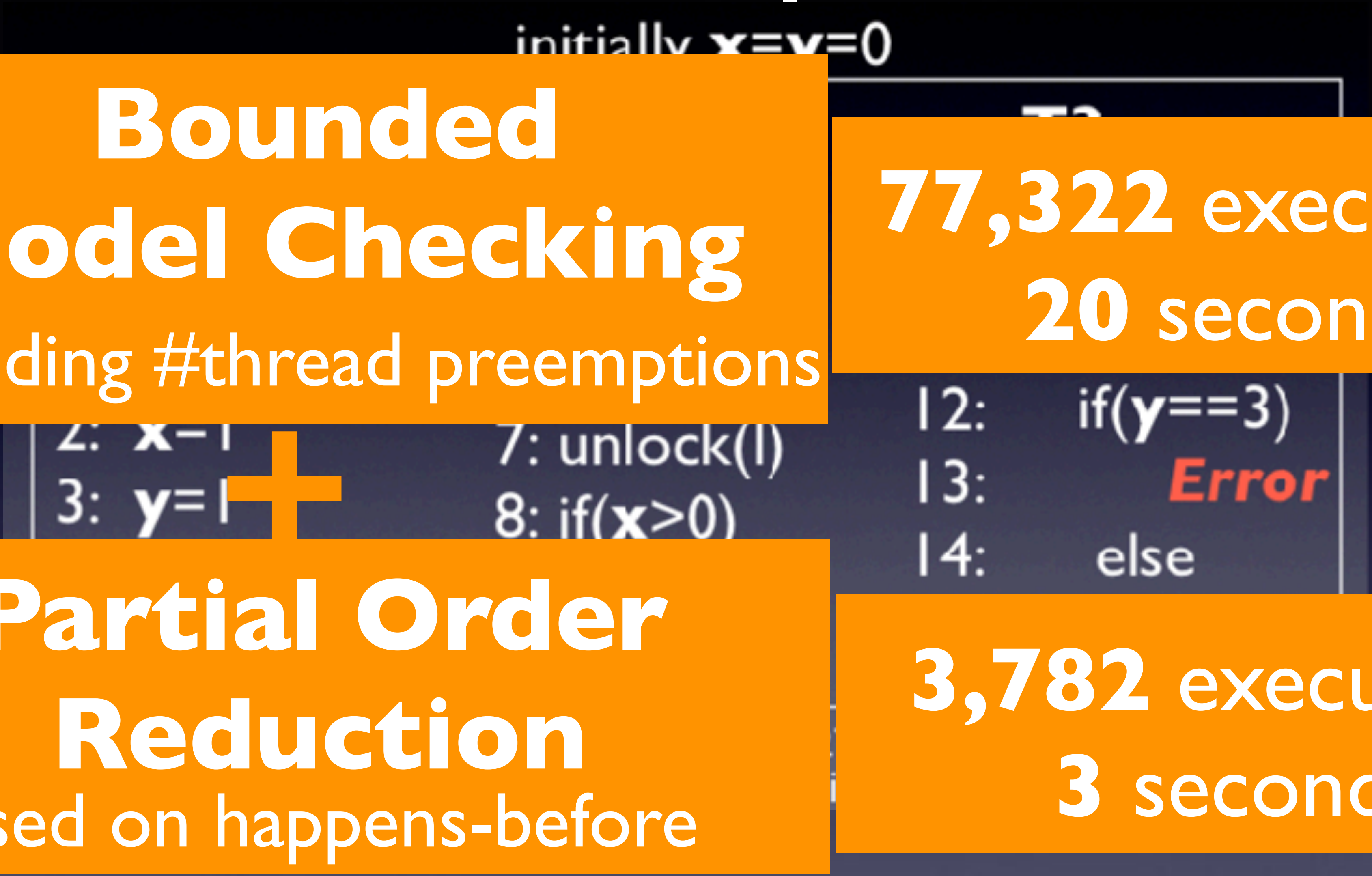
Bounded Model Checking

Bounding #thread preemptions

77,322 executions

20 seconds

Example



```
initially x=y=0
3:  y=1          7: unlock(l)
                 8: if(x>0)
12:     if(y==3)
13:           Error
14:     else
```

Bounded Model Checking

Bounding #thread preemptions

77,322 executions
20 seconds

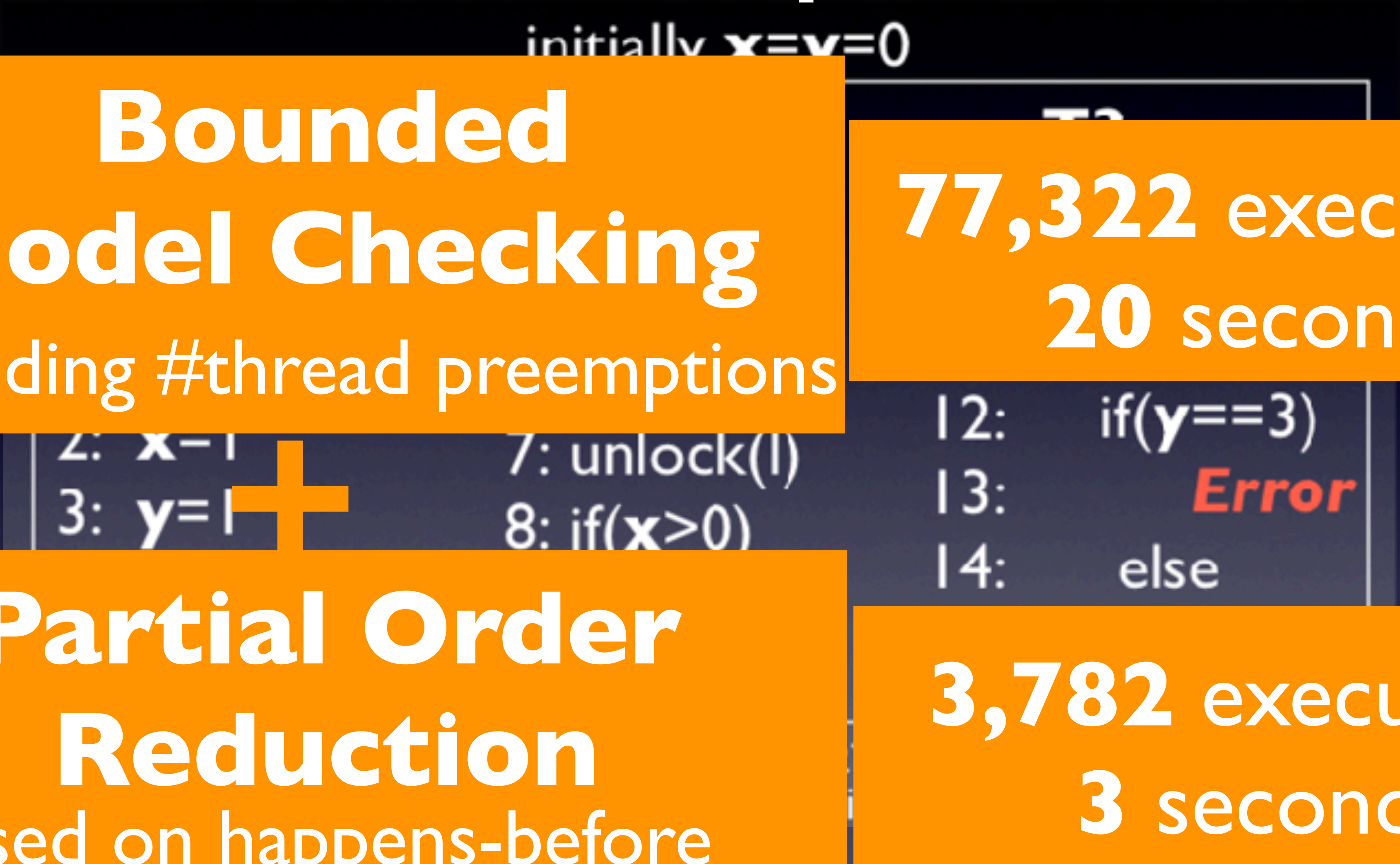
+

Partial Order Reduction

Based on happens-before

3,782 executions
3 seconds

Example



```
initially x=y=0
2:  x=1            7: unlock(l)      12:     if(y==3)
3:  y=1            8: if(x>0)        13:         Error
                                     14:     else
```

Bounded Model Checking

Bounding #thread preemptions

77,322 executions
20 seconds

+

Partial Order Reduction

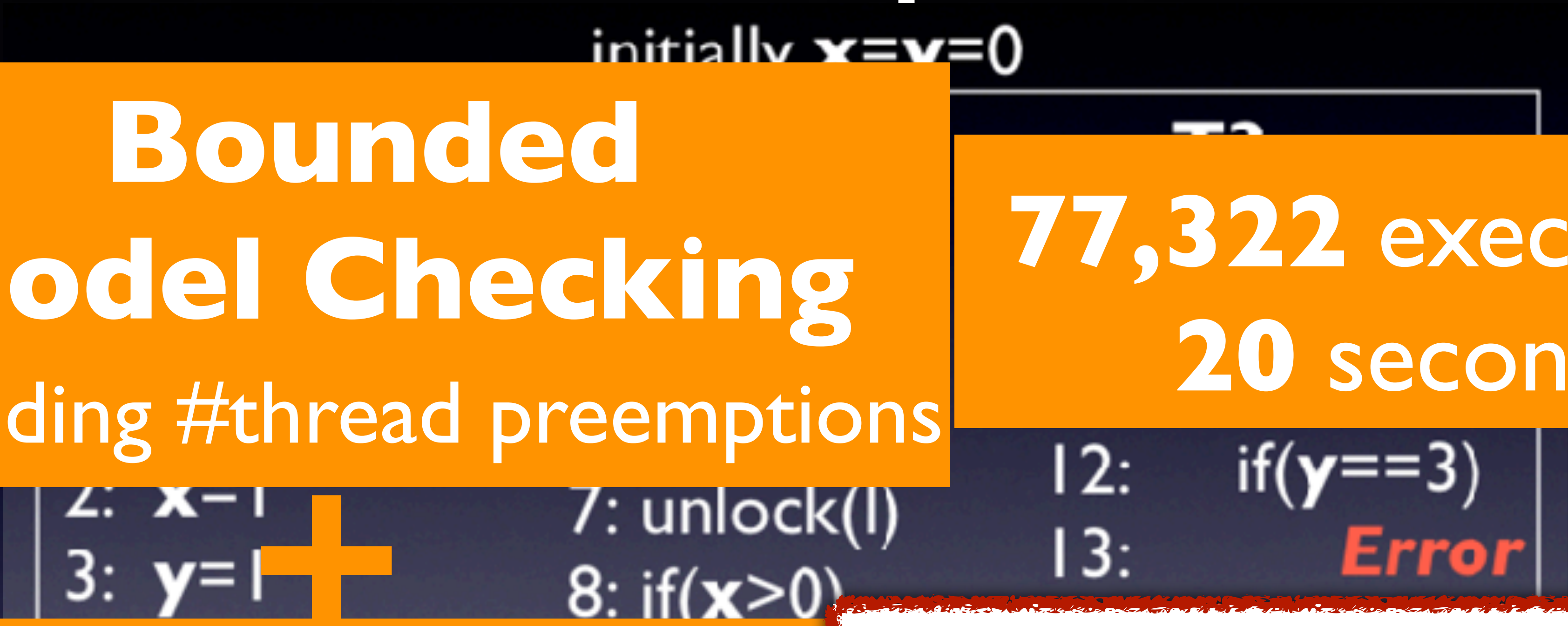
Based on happens-before

3,782 executions
3 seconds

Maximal Causality Reduction

46 executions
2 seconds

Example



initially x=y=0

T2

2: x=1

3: y=1

7: unlock(l)

8: if(x>0)

12: if(y==3)

13: Error

Bounded Model Checking

Bounding #thread preemptions

77,322 executions
20 seconds

+

Partial Order Reduction

Based on happens-before

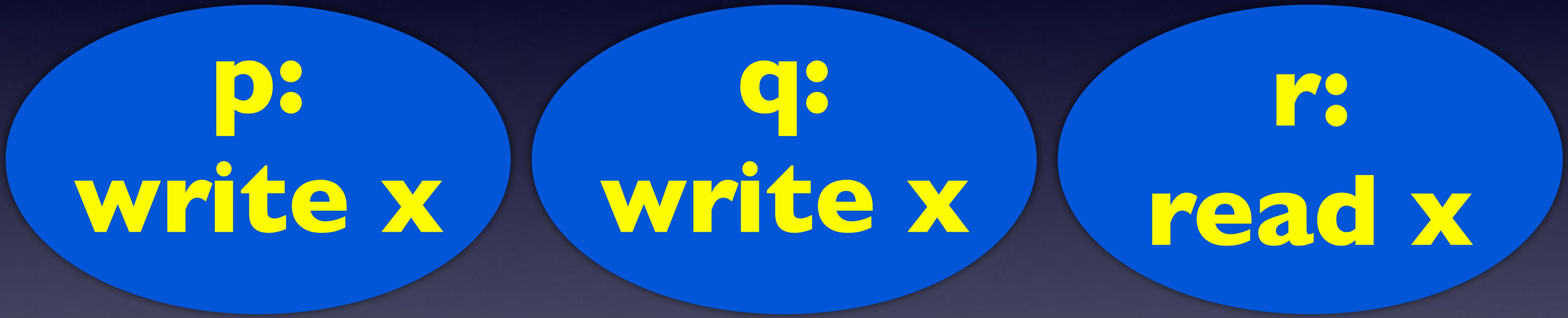
Happens-Before Limitation

Maximal Causality Reduction

46 executions
2 seconds

Happens-Before Limitation

Enforces dependence between conflicting reads and writes



Happens-before: **six** non-redundant transitions

p.q.r p.r.q q.p.r q.r.p r.p.q r.q.p

Happens-Before Limitation

Enforces dependence between conflicting reads and writes

p: write x **q: write x** **r: read x**

Happens-before: **six** non-redundant transitions

p.q.r p.r.q q.p.r q.r.p r.p.q r.q.p

In fact: only **four** are non-redundant

p.q.r == q.r.p r.q.p == r.p.q

Happens-Before Limitation

Enforces dependence between conflicting reads and writes

p: write x

q: write x

r: read x

Happens-before: **six** non-redundant transitions

p.q.r   p.r.q   q.p.r   q.r.p   r.p.q   r.q.p

In fact: only **four** are non-redu

r is the only read

p.q.r == q.r.p   r.q.p == r.p.q

Happens-Before Limitation

Enforces dependence between conflicting reads and writes

p: write x

q: write x

r: read x

Happens-before: **six** non-redundant transitions

If p and q write the same value, then only **two** non-redundant transitions:

p.q.r == q.p.r == q.r.p == p.r.q   r.q.p == r.p.q

Example

initially **x**=**y**=0

T1

loop twice:

```
1:  lock(l)
2:  x=1
3:  y=1
4: unlock(l)
```

*loop **N** times*

T2

loop twice:

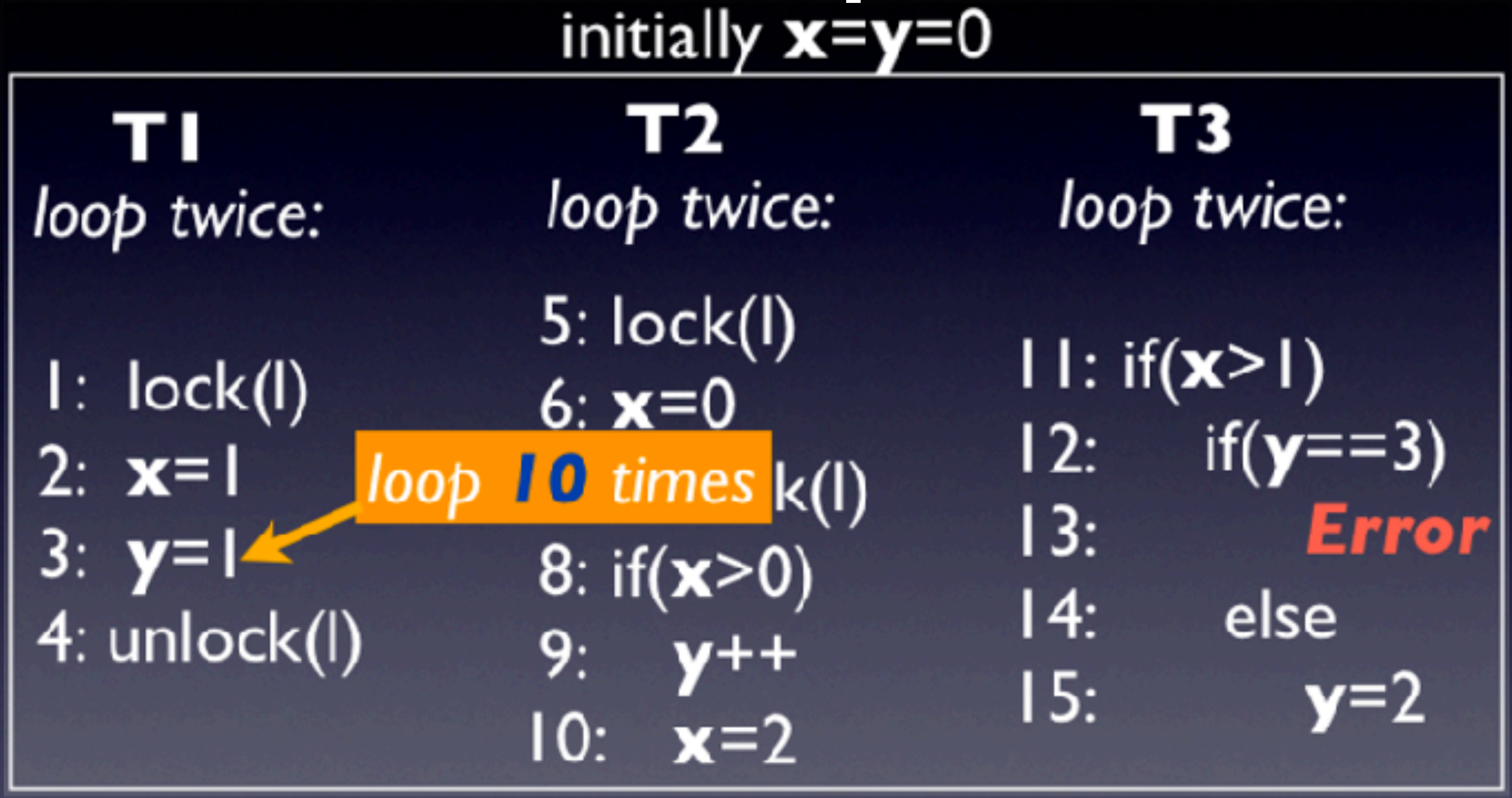
```
5: lock(l)
6: x=0
   k(l)
8: if(x>0)
9:     y++
10:    x=2
```

T3

loop twice:

```
11: if(x>1)
12:     if(y==3)
13:         Error
14:     else
15:         y=2
```

Example



initially **x**=**y**=0

```
T1
loop twice:

1:  lock(l)
2:    x=1
3:    y=1
4:  unlock(l)
```

loop ***10*** *times*

```
T2
loop twice:

5:  lock(l)
6:  x=0
            k(l)
8:  if(x>0)
9:      y++
10:     x=2
```

```
T3
loop twice:

11: if(x>1)
12:     if(y==3)
13:         Error
14:     else
15:         y=2
```

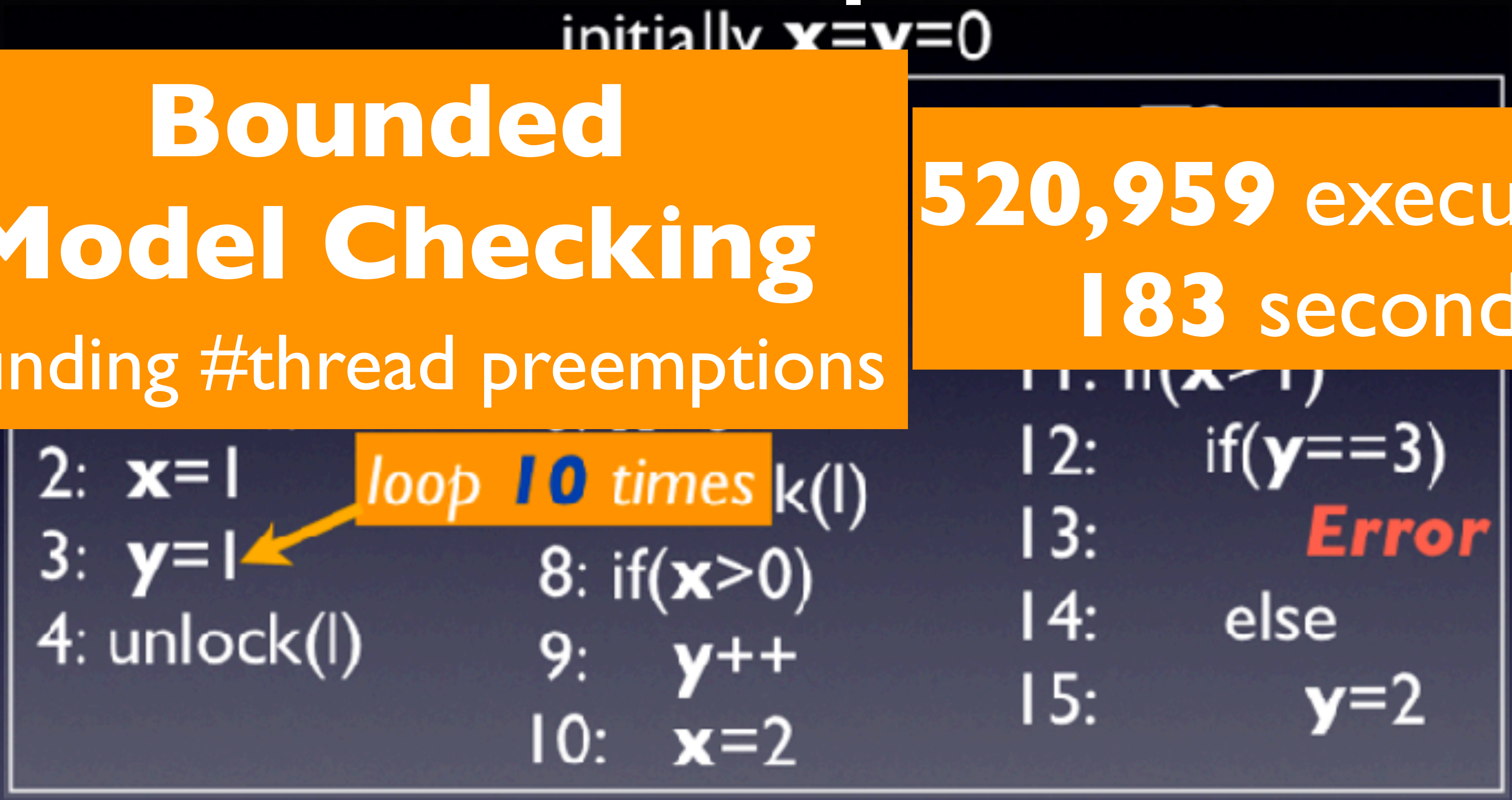
Example

Bounded Model Checking

Bounding #thread preemptions

520,959 executions

183 seconds

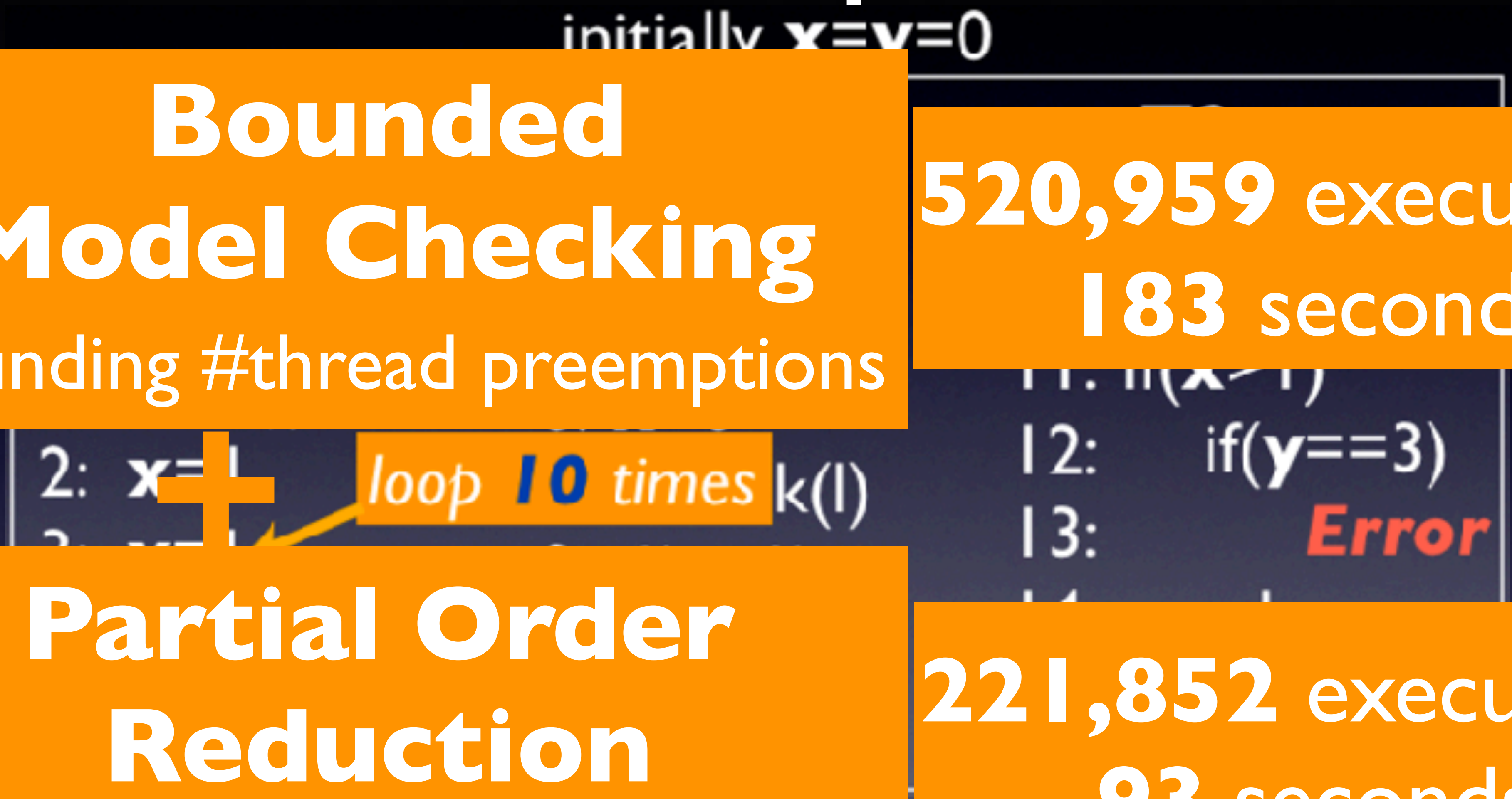


initially **x**=**y**=0

```
2:  x=1          k(l)            11: if(x>1)
3:  y=1          8: if(x>0)      12:     if(y==3)
4: unlock(l)     9:     y++      13:         Error
                 10:    x=2      14:     else
                                 15:         y=2
```

loop ***10*** *times*

Example


initially x=y=0

2: x=1

12: if(y==3)

13: Error

Bounded Model Checking

Bounding #thread preemptions

520,959 executions
183 seconds

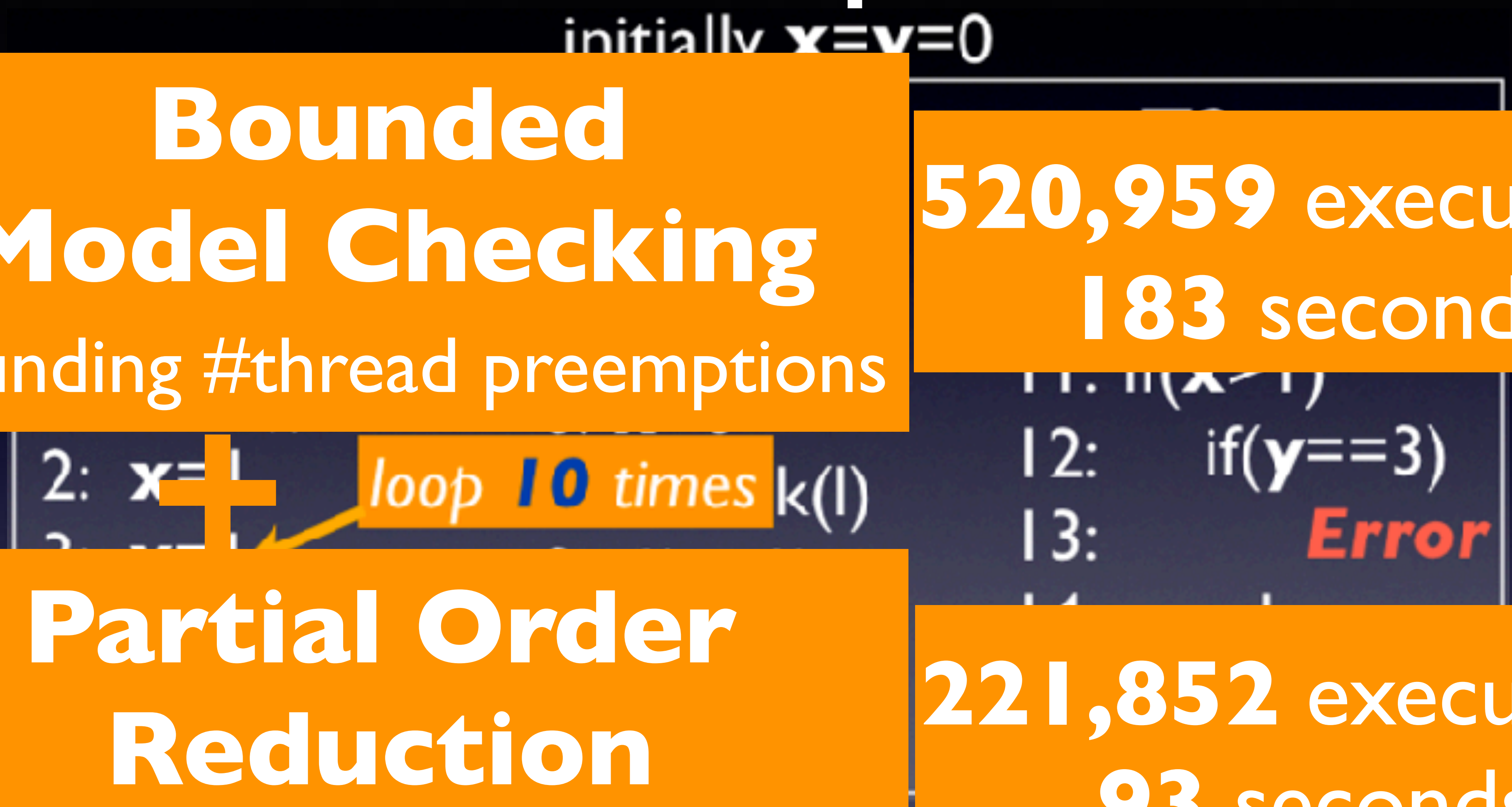
+

Partial Order Reduction

Based on happens-before

221,852 executions
93 seconds

Example



```
initially x=y=0
2:  x=1                        k(l)
                        11: if(x>1)
                        12:     if(y==3)
                        13:             Error
```

loop **10** times

Bounded Model Checking

Bounding #thread preemptions

520,959 executions
183 seconds

+

Partial Order Reduction

Based on happens-before

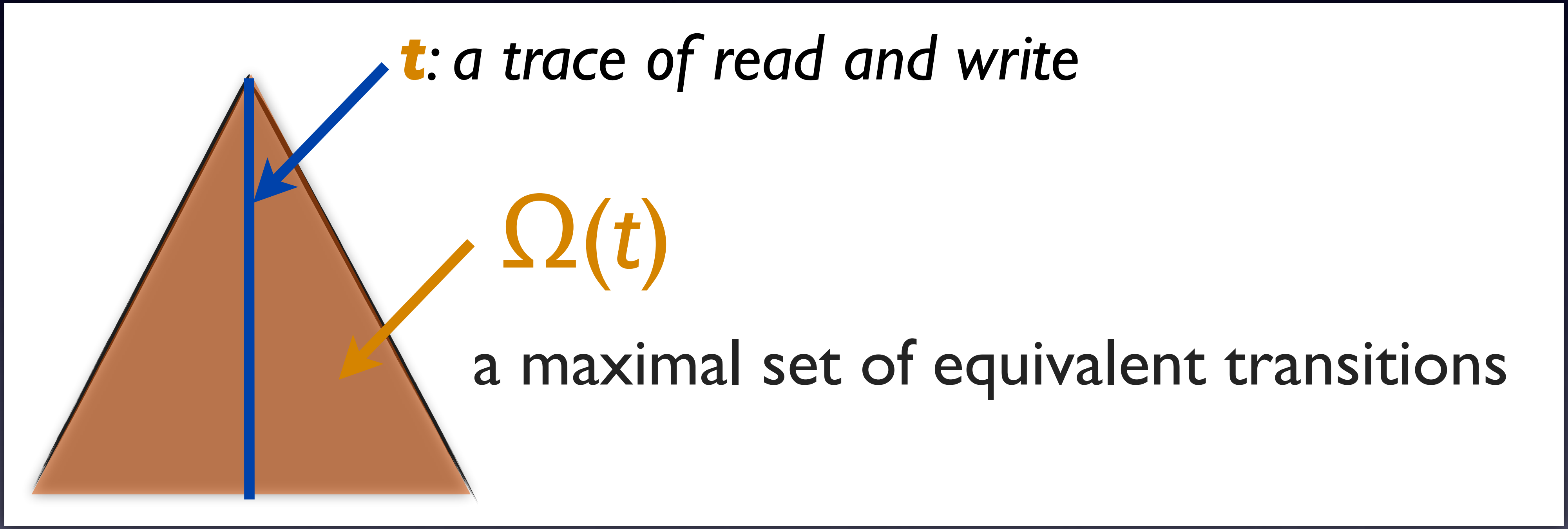
221,852 executions
93 seconds

Maximal Causality Reduction

50 executions
4 seconds

Maximal Causality Reduction

Key idea: characterizing redundant transitions with *maximal causality*

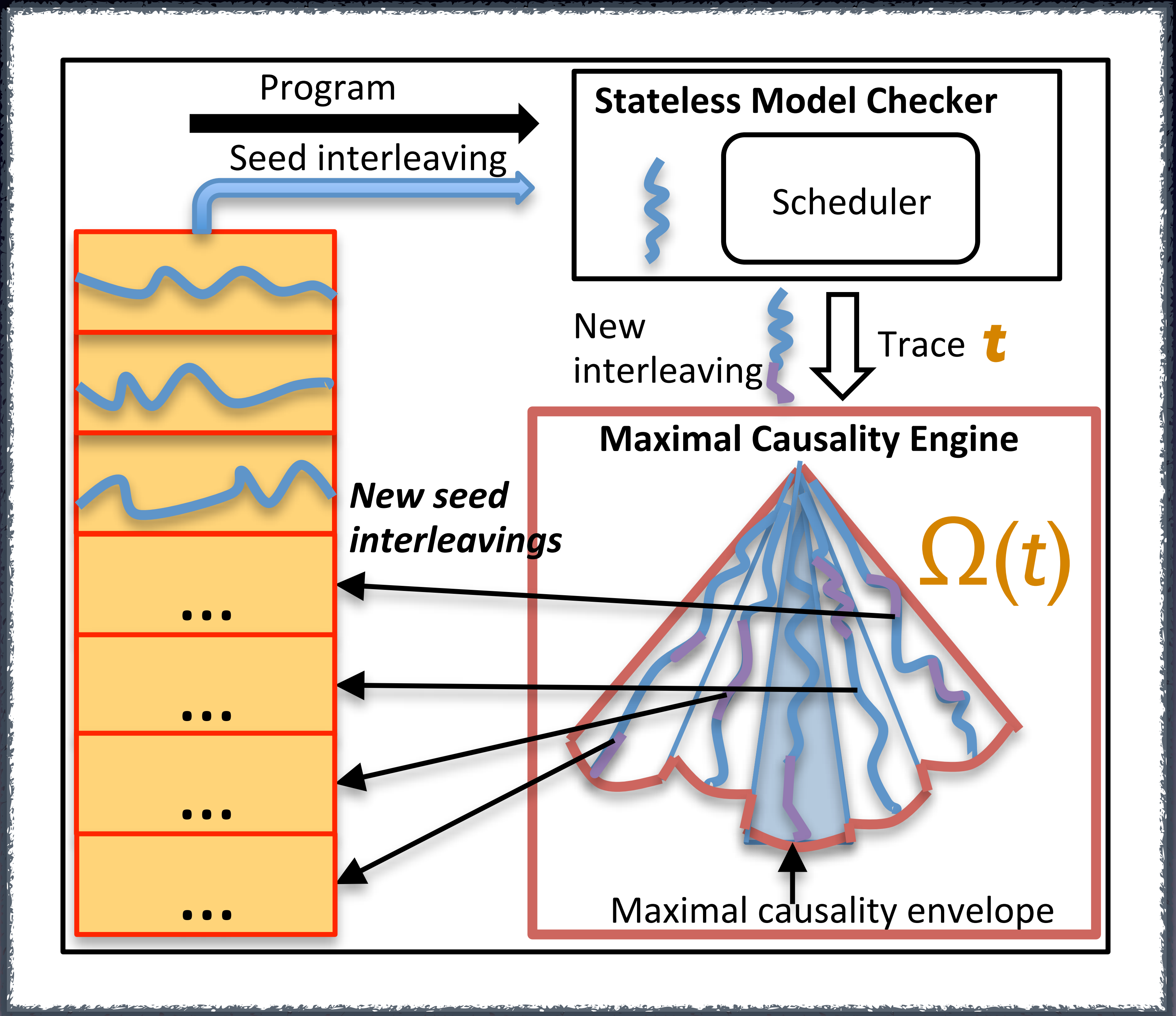


t: *a trace of read and write*
$\Omega(t)$
a maximal set of equivalent transitions

t*: takes the value of reads and writes into consideration*

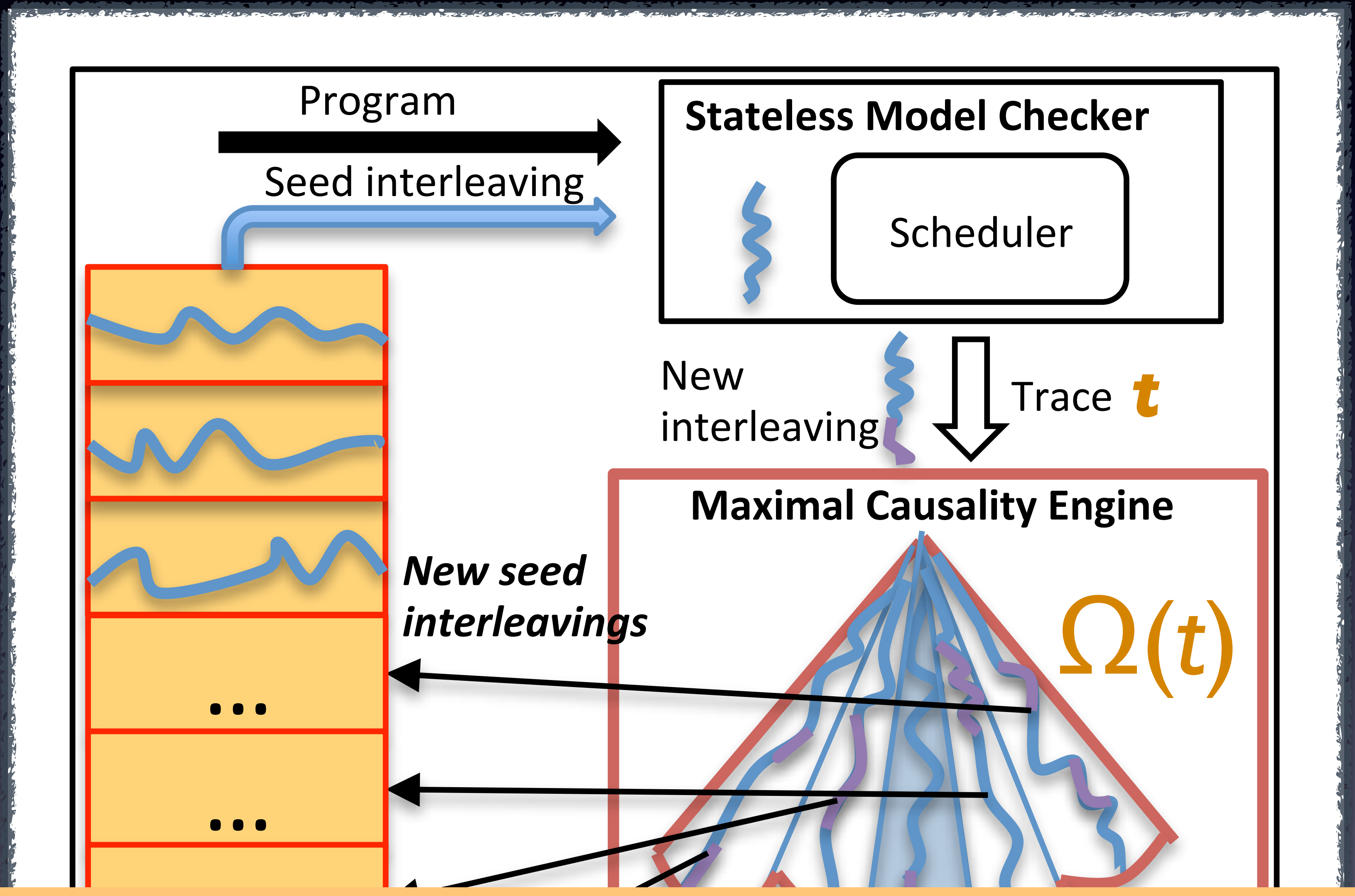
Ω(*t*)*: contains all transitions which all programs that can generate* ***t*** *can also generate*

Maximal Causality Reduction



1. Online tracing *t*
2. Construct $\Omega(t)$
3. Offline property checking with $\Omega(t)$
4. Generate new seed interleavings with $\Omega(t)$

Maximal Causality Reduction



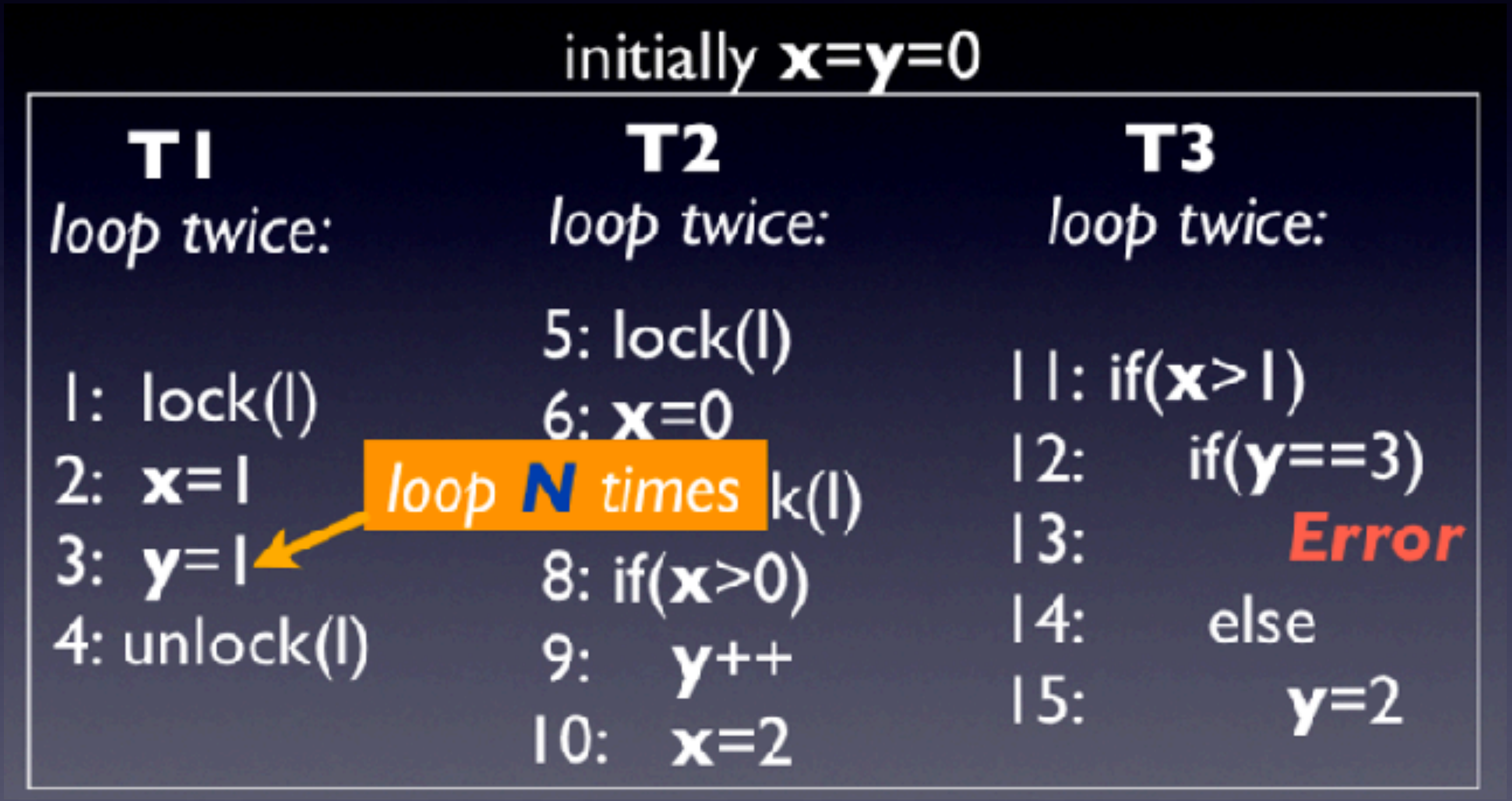
1. Online tracing *t*
2. Construct $\Omega(t)$
3. Offline property checking with $\Omega(t)$
4. Generate new

Seed interleaving: an interleaving in $\Omega(t)$ with at least *one read forced to see a different value*

Following a seed interleaving will produce a new state

Maximal Causality Reduction

N = 1, 2, ... , 10



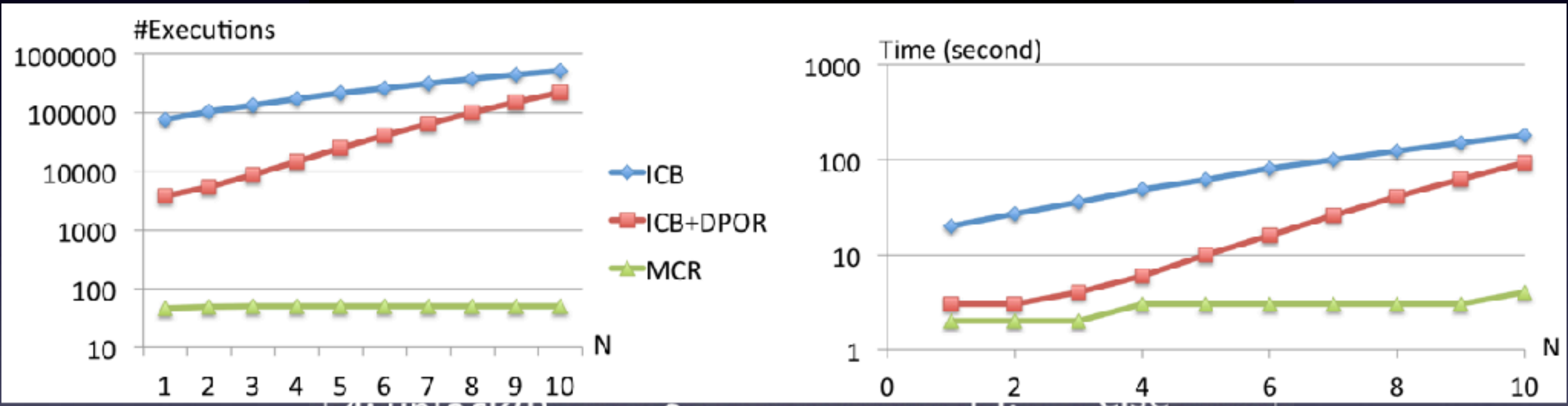
initially **x**=**y**=0

```
T1                    T2                    T3
loop twice:           loop twice:           loop twice:
                      5: lock(l)            11: if(x>1)
1:  lock(l)           6: x=0                12:     if(y==3)
2:  x=1                  k(l)               13:         Error
3:  y=1               8: if(x>0)            14:     else
4: unlock(l)          9:    y++             15:         y=2
                      10:   x=2
```

*loop **N** times*

Maximal Causality Reduction

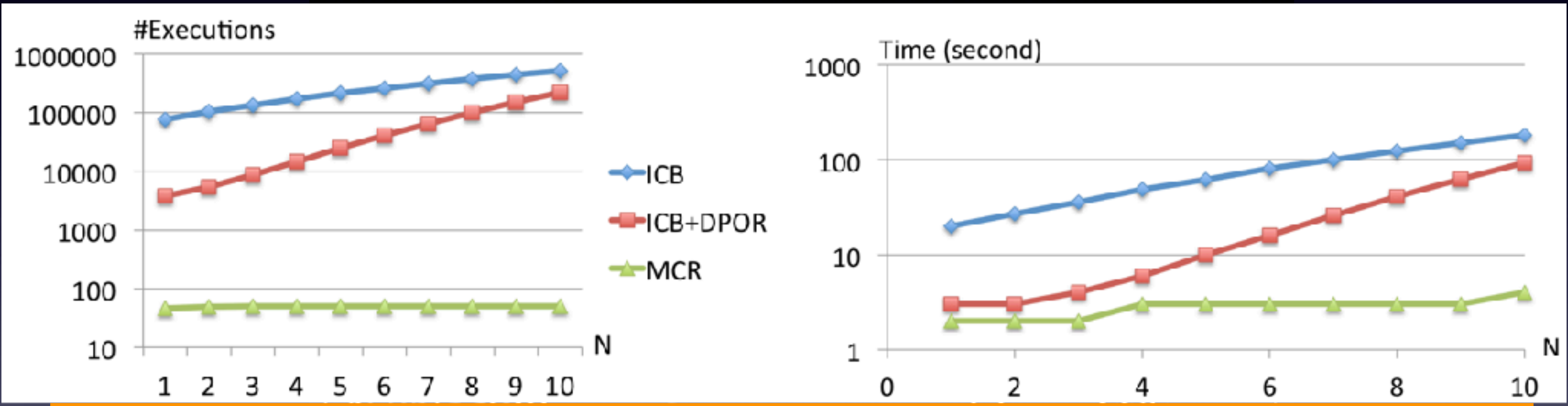
N = 1, 2, ... , 10



```
4: unlock(l)        9:    y++
                   10:    x=2        15:         y=2
```

Maximal Causality Reduction

N = 1, 2, ... , 10

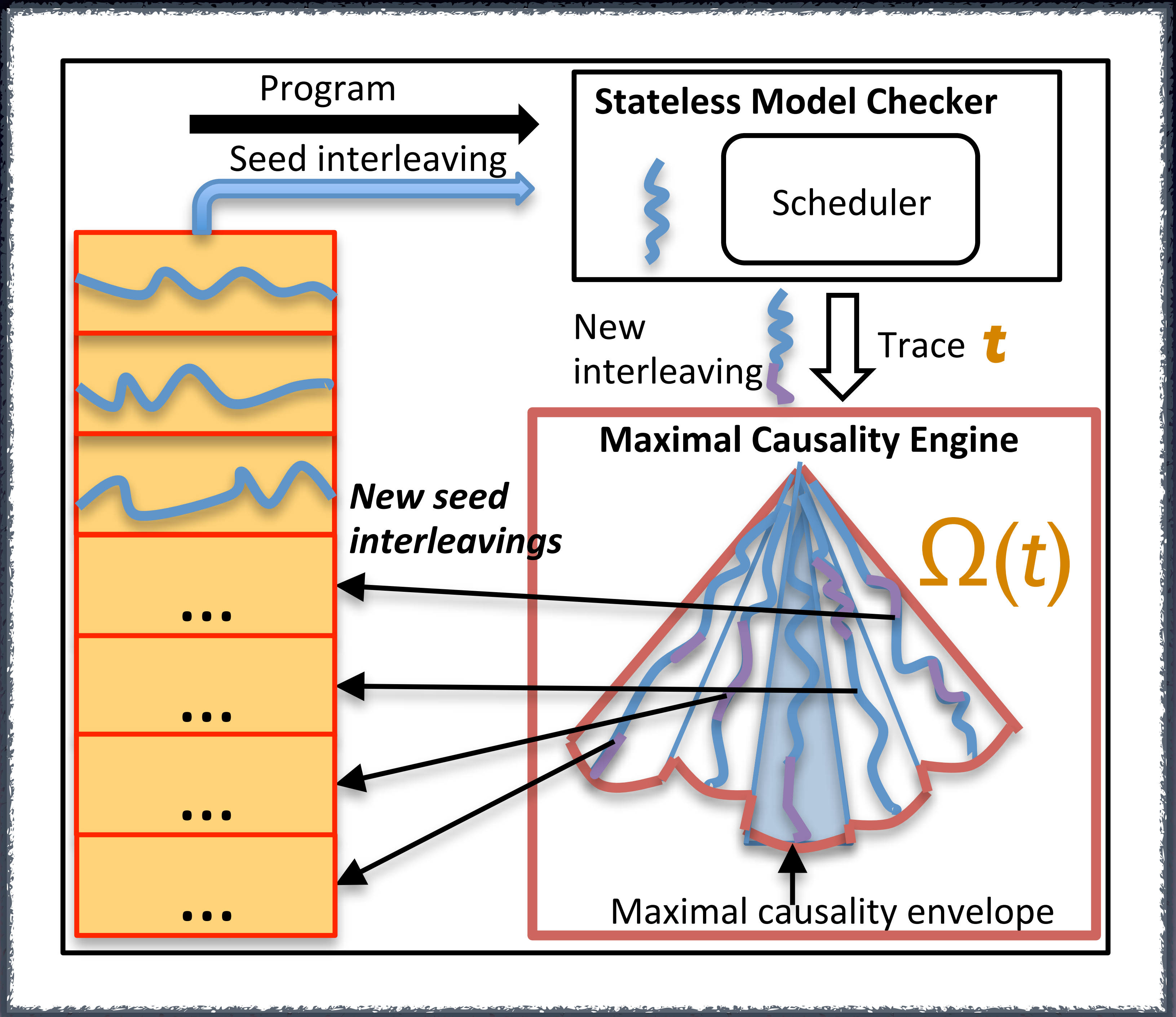


MCR is almost insensitive to N when N>3

Reduced #explorations by BMC+POR

by *two orders of magnitude*

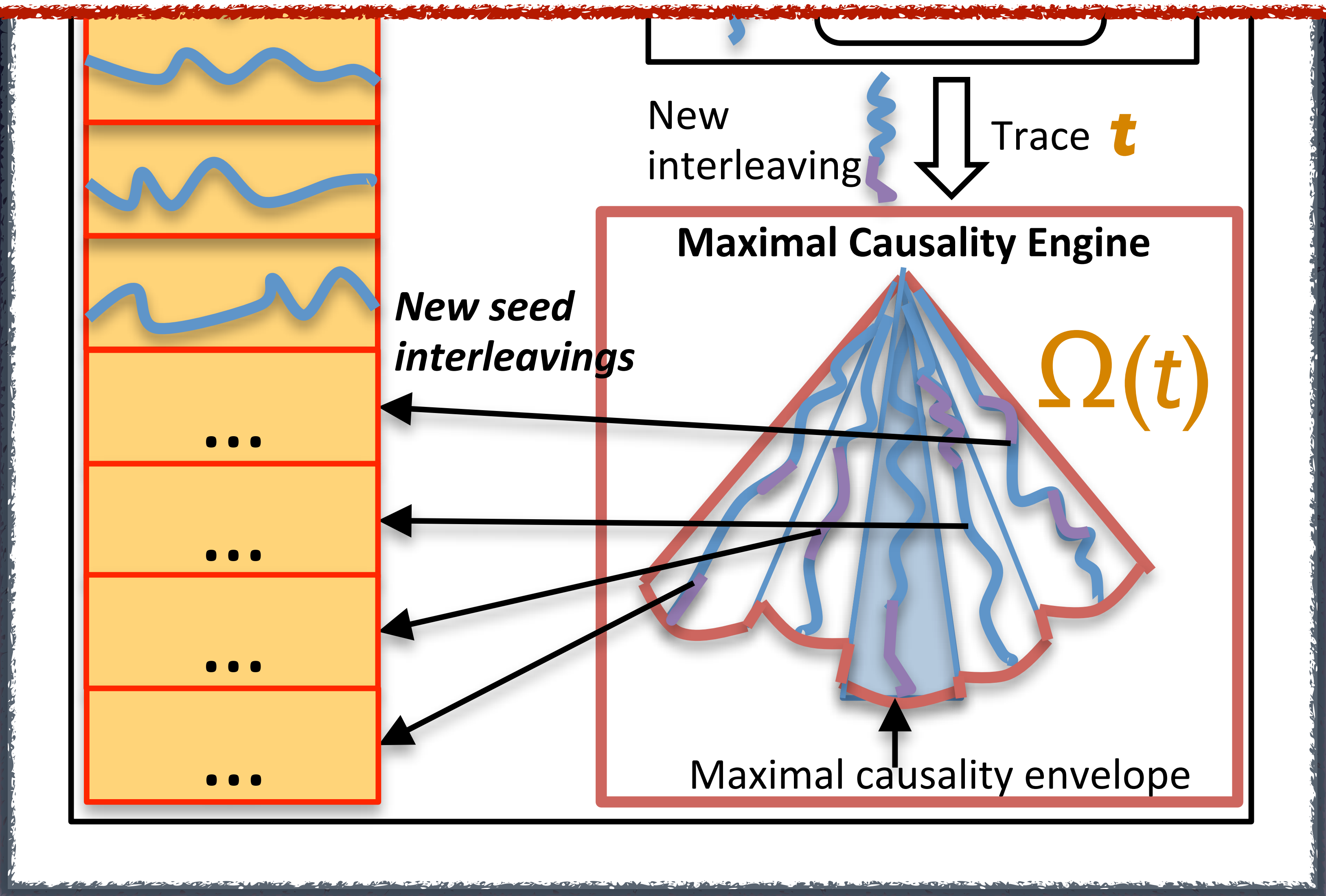
Maximal Causality Reduction



1. Online tracing *t*
2. Construct Ω(*t*)
3. Offline property checking with Ω(*t*)
4. Generate new seed interleavings with Ω(*t*)

Maximal Causality Reduction

A constraint-based approach



2. Construct Ω(*t*)

3. Offline property checking with Ω(*t*)

4. Generate new seed interleavings with Ω(*t*)

Constructing **Ω(*t*)**

A constraint-based approach

Introduce an ORDER variable for each event in the trace ***t***

Must-happen-before constraints **(Φ_{mhb})**

E.g., O1<O2 if events e1 and e2 are by the same thread, and e1 occurs before e2

Lock-mutual-exclusion constraints **(Φ_{lock})**

$$\bigwedge_{(e_a,e_b),(e_c,e_d)\in S_l} (O_{e_b} < O_{e_c} \vee O_{e_d} < O_{e_a})$$

Data-validity constraints **(Φ_{rw})**

$$\Phi_{rw}(e) \equiv \bigwedge_{r \in \prec_e} \Phi_{value}(r, value(r))$$

$$\Phi_{value}(r,v) \equiv \bigvee_{w \in W_v^x} (\Phi_{rw}(w) \wedge O_w < O_r \bigwedge_{w \neq w' \in W^x} (O_{w'} < O_w \vee O_r < O_{w'})))$$

$$\Phi_{rw} \equiv \bigvee_{e \in \tau} \Phi_{rw}(e)$$

Constructing Ω(***t***)

A constraint-based approach

Introduce an ORDER variable for each event in the trace ***t***

Must-happen-before constraints (Φ_{mhb})

E.g., O1<O2 if events e1 and e2 are by the same thread, and e1 occurs before e2

Lock-mutual-exclusion constraints (Φ_{lock})

$$\bigwedge_{(e_a,e_b),(e_c,e_d)\in S_l} (O_{e_b} < O_{e_c} \vee O_{e_d} < O_{e_a})$$

Data-validity constraints (Φ_{rw})

An event is feasible if every read that *must-happen-before* it in the trace ***t*** *returns the same value* as that in ***t***

$$\Phi_{rw} \equiv \bigvee_{e\in\tau} \Phi_{rw}(e)$$

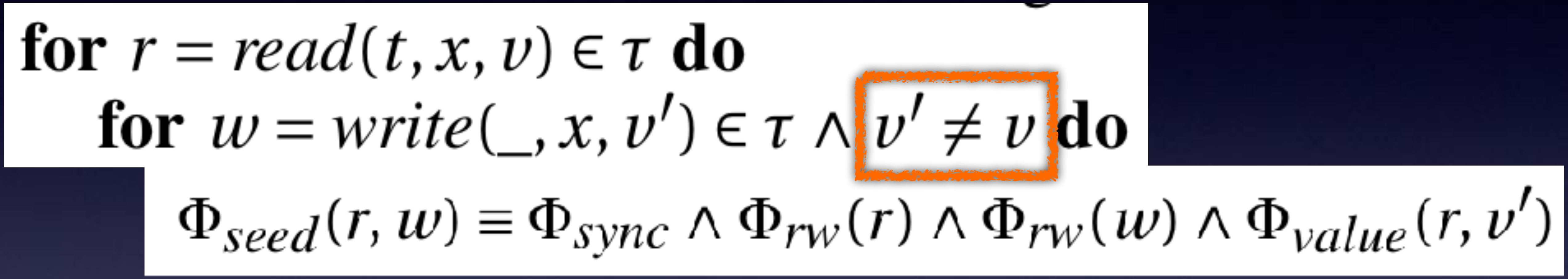
Generating Seed Interleavings

Main idea: enforce a read to see a new value

```
for $r = read(t, x, v) \in \tau$ do
    for $w = write(\_, x, v') \in \tau \wedge \boxed{v' \neq v}$ do
        $\Phi_{seed}(r, w) \equiv \Phi_{sync} \wedge \Phi_{rw}(r) \wedge \Phi_{rw}(w) \wedge \Phi_{value}(r, v')$
```

Generating Seed Interleavings

Main idea: enforce a read to see a new value



```
for $r = read(t, x, v) \in \tau$ do
    for $w = write(\_, x, v') \in \tau \land \boxed{v' \neq v}$ do
        $\Phi_{seed}(r, w) \equiv \Phi_{sync} \land \Phi_{rw}(r) \land \Phi_{rw}(w) \land \Phi_{value}(r, v')$
```

- Every seed interleaving is feasible and has at least one new event: *a read event that returns a new value*
- Termination: *when no new seed interleaving can be generated*

Generating Seed Interleavings

Main idea: enforce a read to see a new value

```
for $r = read(t, x, v) \in \tau$ do
    for $w = write(\_, x, v') \in \tau \wedge \boxed{v' \neq v}$ do
        $\Phi_{seed}(r, w) \equiv \Phi_{sync} \wedge \Phi_{rw}(r) \wedge \Phi_{rw}(w) \wedge \Phi_{value}(r, v')$
```

- Ever ew
  ever

No seed interleaving is redundant

- Termination: *when no new seed interleaving can be generated*

Generating Seed Interleavings

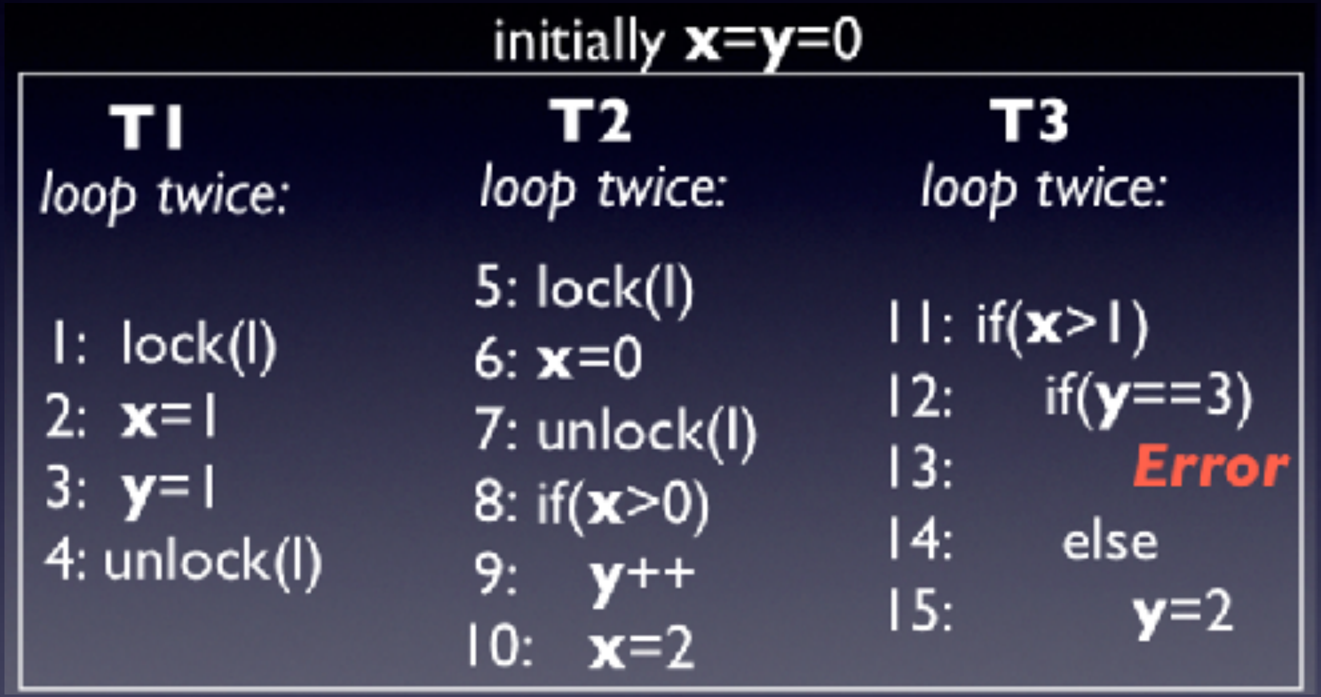
Main idea: enforce a read to see a new value

```
for $r = read(t, x, v) \in \tau$ do
    for $w = write(\_, x, v') \in \tau \wedge \boxed{v' \neq v}$ do
```

$$\Phi_{seed}(r, w) \equiv \Phi_{sync} \wedge \Phi_{rw}(r) \wedge \Phi_{rw}(w) \wedge \Phi_{value}(r, v')$$

- Ever **No seed interleaving is redundant** ew
  ever
- Tern **No seed interleaving is missed** *ed*

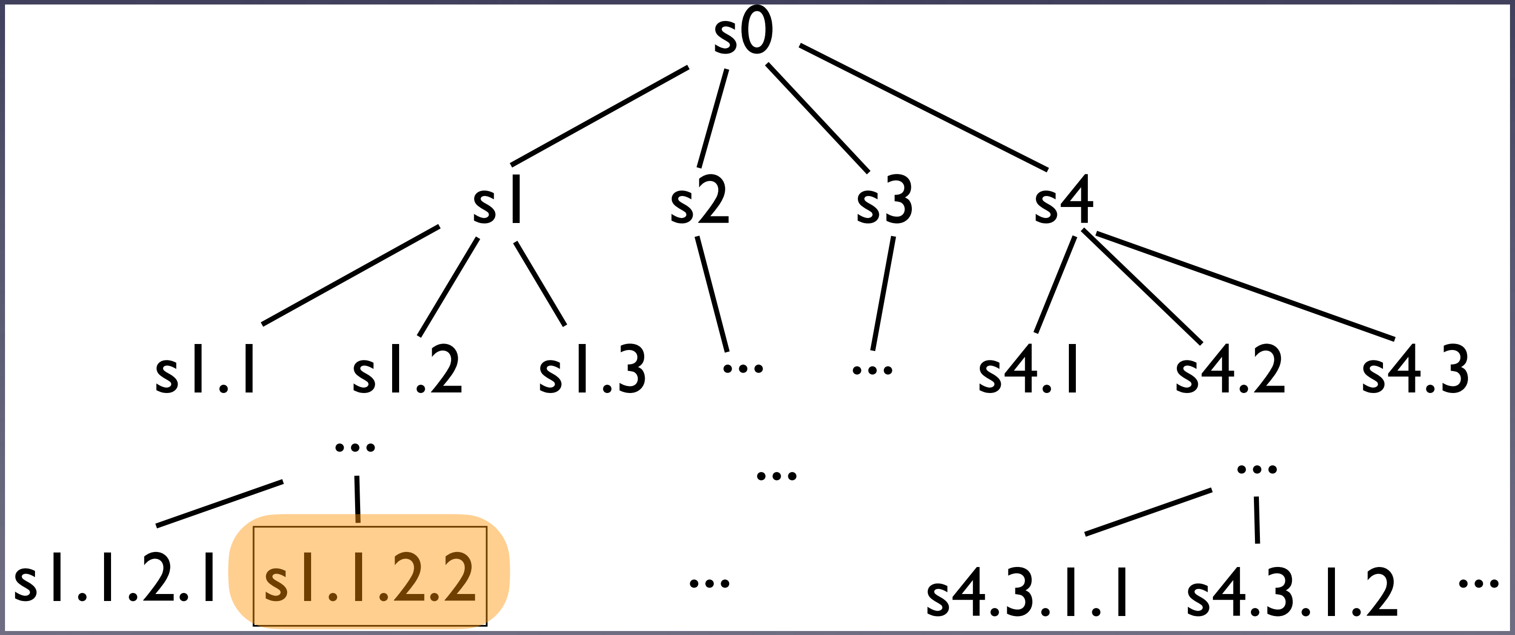
Seed Interleaving Exploration



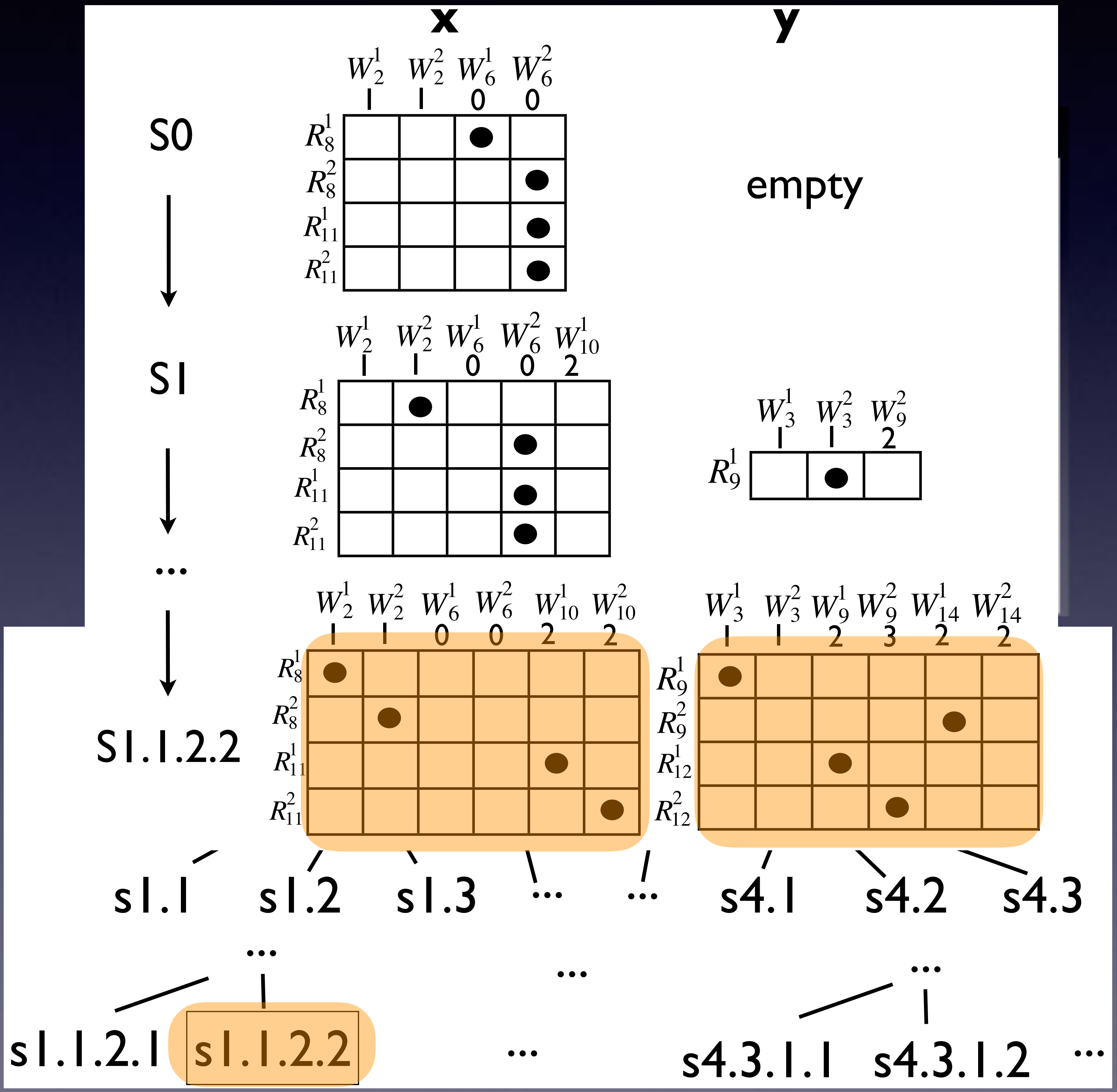
initially **x**=**y**=0

```
T1                 T2                 T3
loop twice:        loop twice:        loop twice:

                   5: lock(l)
1:  lock(l)        6: x=0             11: if(x>1)
2:  x=1            7: unlock(l)       12:     if(y==3)
3:  y=1            8: if(x>0)         13:         Error
4: unlock(l)       9:     y++         14:     else
                   10:    x=2         15:         y=2
```



Seed Interleaving Exploration



Checking Property Constraints

Checking assertions:

$$\underbrace{\Phi_{sync}}_{\text{synchronization constraints}} \wedge \Big(\bigwedge_{e\in R} \underbrace{\Phi_{rw}(e)}_{\text{data-validity constraints}} \wedge \overset{\text{value returned by event e}}{v(e)}\Big) \wedge \underbrace{\phi_{assert}(R)}_{\text{assertion formula over a set of reads}}$$

Checking Property Constraints

Checking assertions:

$$\underbrace{\Phi_{sync}}_{\text{synchronization constraints}} \wedge \Big(\bigwedge_{e \in R} \underbrace{\Phi_{rw}(e)}_{\text{data-validity constraints}} \wedge \overbrace{v(e)}^{\text{value returned by event e}} \Big) \wedge \underbrace{\phi_{assert}(R)}_{\text{assertion formula over a set of reads}}$$

E.g., Null Pointer Deferences:

$$\Phi_{sync} \wedge \Phi_{rw}(e) \wedge (value(e) = NULL)$$

Checking Property Constraints

Checking assertions:

$$\Phi_{sync} \land (\bigwedge_{e \in R} \Phi_{rw}(e) \land \nu(e)) \land \phi_{assert}(R)$$

value returned by event e

synchronization constraints

data-validity constraints

assertion formula over a set of reads

E.g., Null Pointer Deferences:

$$\Phi_{sync} \land \Phi_{rw}(e) \land (value(e) = NULL)$$

Checking data races:

$$\Phi_{sync} \land (O_{e_a} = O_{e_b}) \land \Phi_{rw}(e_a) \land \Phi_{rw}(e_b)$$

Relaxed Memory Models

Must-happen-before constraints **(Φ_{mhb})**

```
Init: x=y=0
thread 1:
  x = 1  //a1
  a = y  //a2

thread 2:
  y = 1  //b1
  b = x  //b2
```

Under SC:

$O_{a1} < O_{a2}$

$O_{b1} < O_{b2}$

Under TSO/PSO

$O_{a1}, O_{a2}, O_{b1}, O_{b2}$

A Real Bug – $12 million loss of equipment

https://stackoverflow.com/questions/16159203/why-does-this-java-program-terminate-despite-that-apparently-it-shouldnt-and-d

```
Init: x=1, y=2
     T1
1: T2.start()
2: z=0
3: x++
4: y++
5: z=1
6: T2.join()

     T2
7: if (z==1)
8:   ✗ assert(x+1==y)
```

A Real Bug – $12 million loss of equipment

https://stackoverflow.com/questions/16159203/why-does-this-java-program-terminate-despite-that-apparently-it-shouldnt-and-d

```
Init: x=1, y=2
     T1
1: T2.start()
2: z=0
3: x++
4: y++
5: z=1
6: T2.join()

     T2
7: if (z==1)
8:   ✗ assert(x+1==y)
```

Read-Write Constraints

$$(R_z^7 = 0 \ \land O_7 < O_2) \lor$$
$$(R_z^7 = W_z^5 \ \land O_5 < O_7 \land (O_2 < O_5 \lor O_7 < O_2))$$

Memory Order Constraints

SC

$$O_1 < O_2 < O_3^{R_x} < O_3^{W_x} < O_4^{R_x}$$
$$< O_4^{W_x} < O_5 < O_6$$
$$O_7 < O_8^x < O_8^y$$

PSO

$$O_1 < O_2 \quad O_5 < O_6$$
$$O_3^{R_x} < O_3^{W_x} \quad O_4^{R_x} < O_4^{W_x}$$
$$O_7 < O_8^x < O_8^y$$

Path Constraints

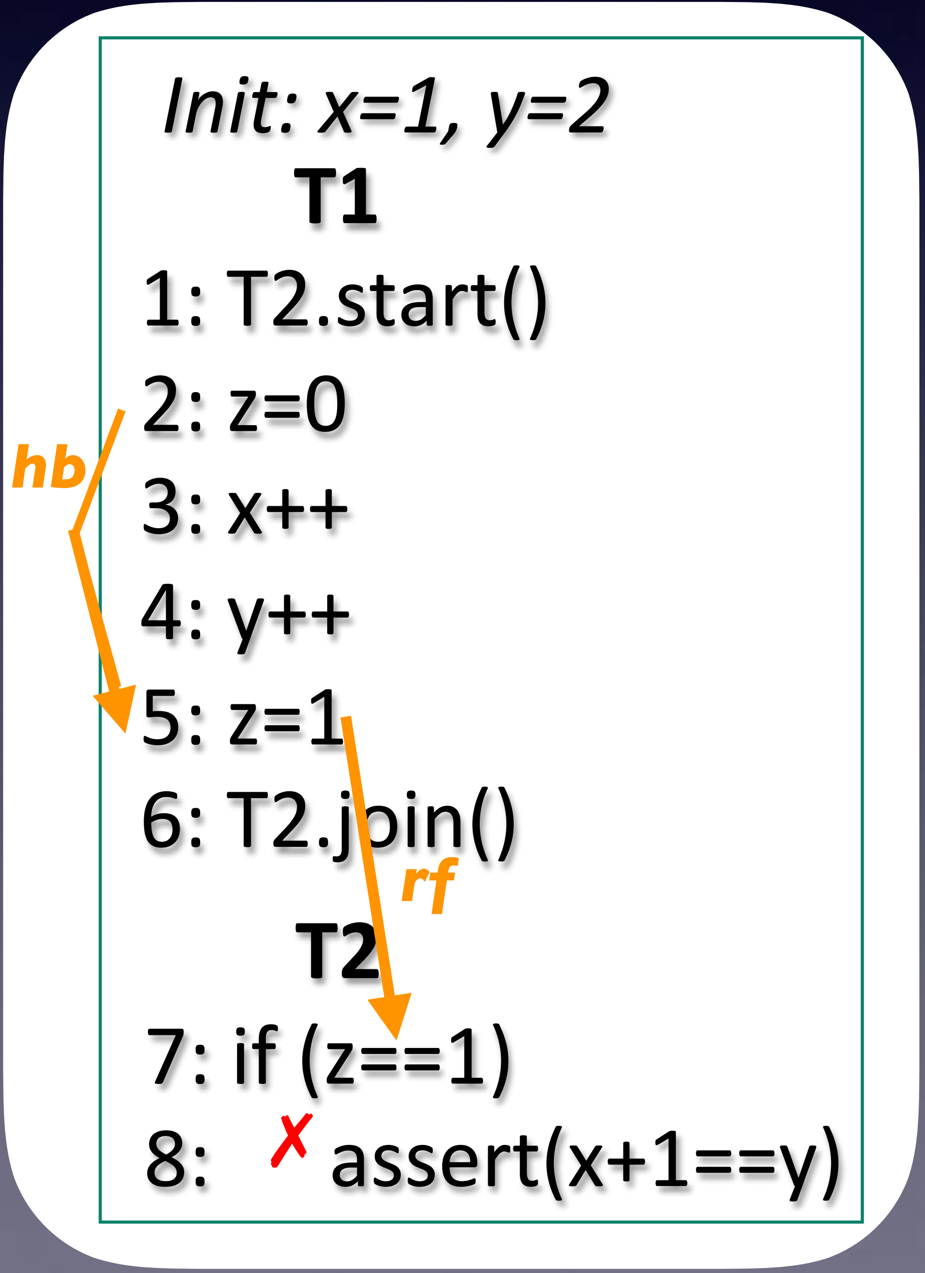
$$R_z^7 = 1$$

Failure Constraints

$$R_x^8 + 1 != R_y^8$$

A Real Bug – $12 million loss of equipment

https://stackoverflow.com/questions/16159203/why-does-this-java-program-terminate-despite-that-apparently-it-shouldnt-and-d



```
Init: x=1, y=2
      T1
1: T2.start()
2: z=0
3: x++
4: y++
5: z=1
6: T2.join()
      T2
7: if (z==1)
8:   ✗ assert(x+1==y)
```

Read-Write Constraints

$(R_z^7 = 0 \ \land O_7 < O_2) \lor$ match a read to a write

$(R_z^7 = W_z^5 \ \land O_5 < O_7 \land (O_2 < O_5 \lor O_7 < O_2))$

Memory Order Constraints

SC

$$O_1 < O_2 < O_3^{R_x} < O_3^{W_x} < O_4^{R_x} < O_4^{W_x} < O_5 < O_6$$
$$O_7 < O_8^x < O_8^y$$

PSO

$$O_1 < O_2 \quad O_5 < O_6$$
$$O_3^{R_x} < O_3^{W_x} \quad O_4^{R_x} < O_4^{W_x}$$
$$O_7 < O_8^x < O_8^y$$

Path Constraints

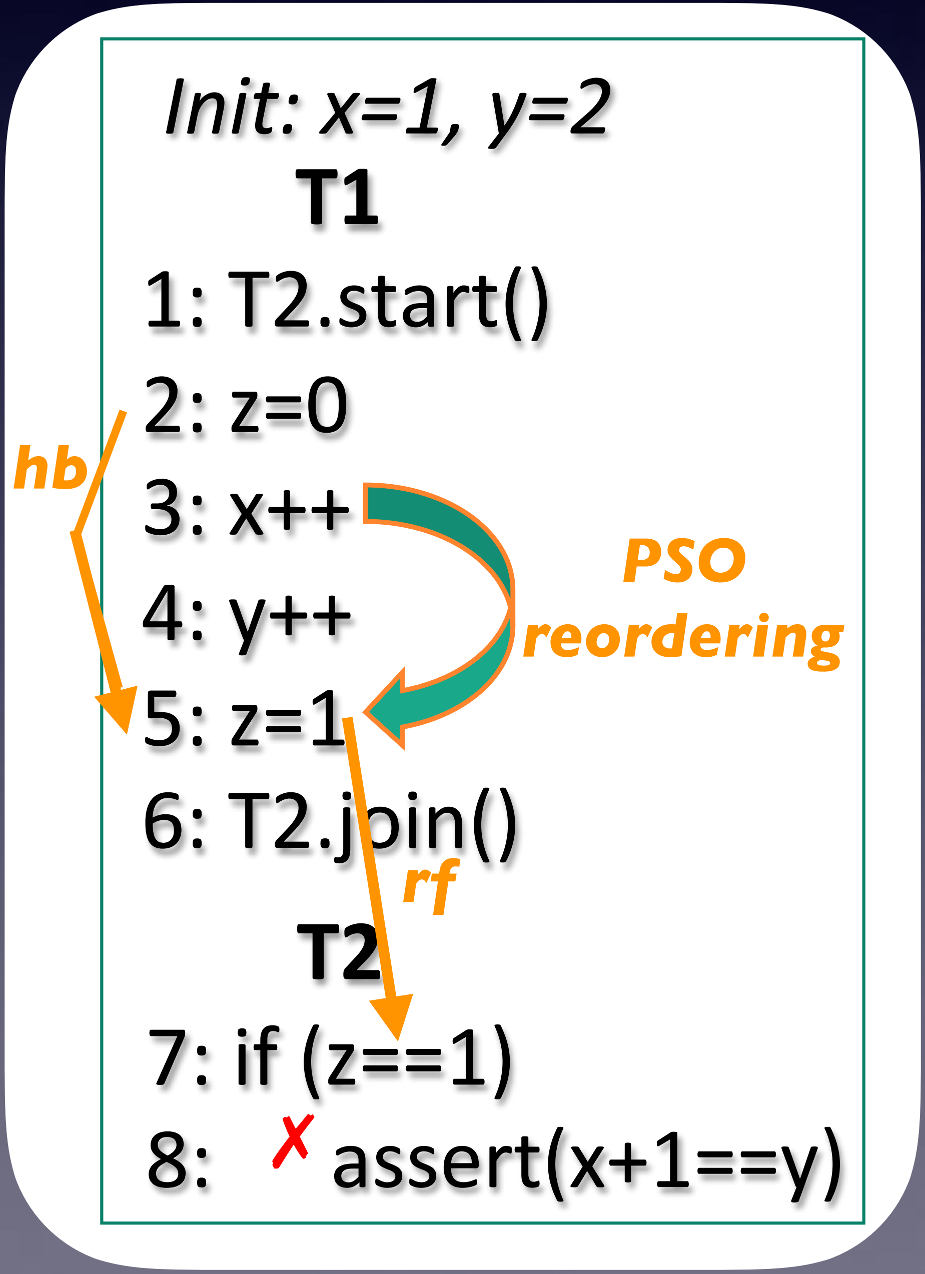
$$R_z^7 = 1$$

Failure Constraints

$$R_x^8 + 1 != R_y^8$$

A Real Bug – $12 million loss of equipment

https://stackoverflow.com/questions/16159203/why-does-this-java-program-terminate-despite-that-apparently-it-shouldnt-and-d



```
Init: x=1, y=2
      T1
1: T2.start()
2: z=0
3: x++
4: y++
5: z=1
6: T2.join()

      T2
7: if (z==1)
8:   ✗ assert(x+1==y)
```

Read-Write Constraints

$(R_z^7 = 0 \ \wedge O_7 < O_2) \vee$ match a read to a write

$(R_z^7 = W_z^5 \ \wedge O_5 < O_7 \wedge (O_2 < O_5 \vee O_7 < O_2))$

Memory Order Constraints

execution should be allowed by the memory model

SC

$O_1 < O_2 < O_3^{R_x} < O_3^{W_x} < O_4^{R_x}$
$< O_4^{W_x} < O_5 < O_6$
$O_7 < O_8^x < O_8^y$

PSO

$O_1 < O_2 \quad O_5 < O_6$
$O_3^{R_x} < O_3^{W_x} \quad O_4^{R_x} < O_4^{W_x}$
$O_7 < O_8^x < O_8^y$

Path Constraints

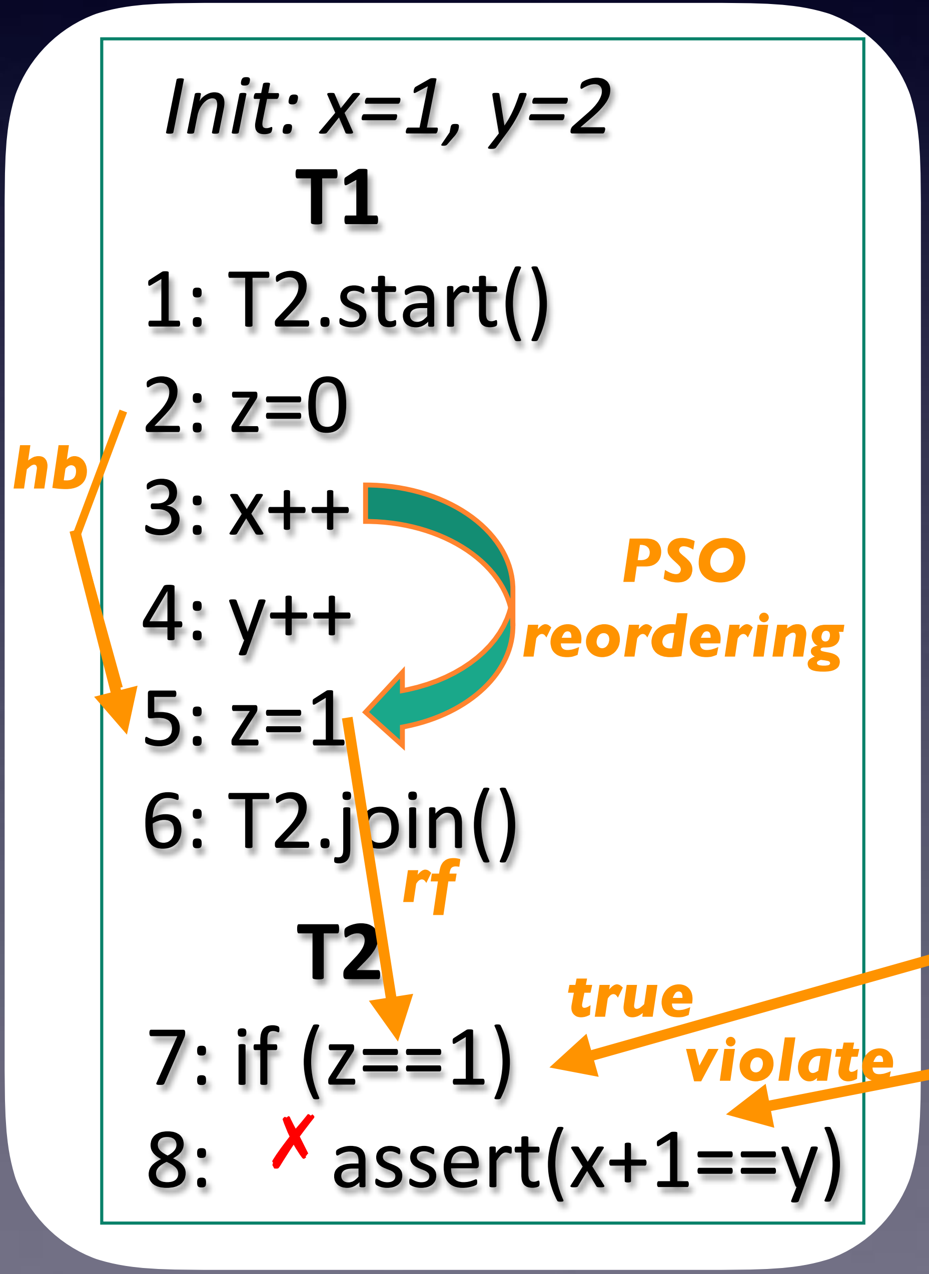
$R_z^7 = 1$

Failure Constraints

$R_x^8 + 1 != R_y^8$

A Real Bug – $12 million loss of equipment

https://stackoverflow.com/questions/16159203/why-does-this-java-program-terminate-despite-that-apparently-it-shouldnt-and-d



```
Init: x=1, y=2
      T1
1: T2.start()
2: z=0
3: x++
4: y++
5: z=1
6: T2.join()

      T2
7: if (z==1)
8:  ✗ assert(x+1==y)
```

Read-Write Constraints

match a read to a write

$$(R_z^7 = 0 \ \land O_7 < O_2) \lor$$
$$(R_z^7 = W_z^5 \ \land O_5 < O_7 \land (O_2 < O_5 \lor O_7 < O_2))$$

Memory Order Constraints

execution should be allowed by the memory model

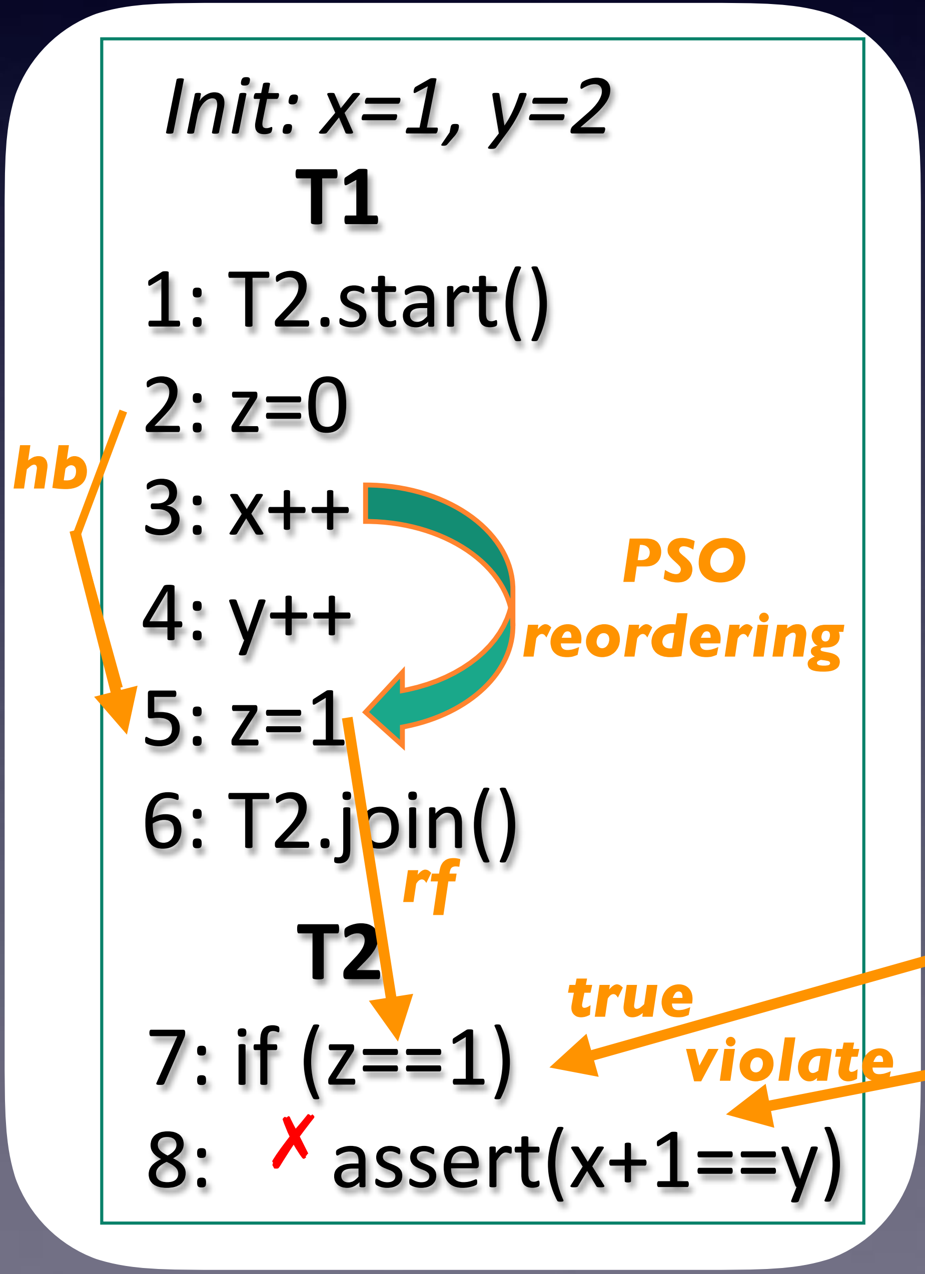
SC	PSO
$O_1 < O_2 < O_3^{R_x} < O_3^{W_x} < O_4^{R_x} < O_4^{W_x} < O_5 < O_6$	$O_1 < O_2 \quad O_5 < O_6$
	$O_3^{R_x} < O_3^{W_x} \quad O_4^{R_x} < O_4^{W_x}$
$O_7 < O_8^x < O_8^y$	$O_7 < O_8^x < O_8^y$

make the error happen

Path Constraints	Failure Constraints
$R_z^7 = 1$	$R_x^8 + 1 != R_y^8$

A Real Bug – $12 million loss of equipment

https://stackoverflow.com/questions/16159203/why-does-this-java-program-terminate-despite-that-apparently-it-shouldnt-and-d



```
Init: x=1, y=2
      T1
1: T2.start()
2: z=0
3: x++
4: y++
5: z=1
6: T2.join()

      T2
7: if (z==1)
8:   ✗ assert(x+1==y)
```

Read-Write Constraints

match a read to a write

$$(R_z^7 = 0 \ \wedge O_7 < O_2) \vee$$
$$(R_z^7 = W_z^5 \ \wedge O_5 < O_7 \wedge (O_2 < O_5 \vee O_7 < O_2))$$

Memory Order Constraints

execution should be allowed by the memory model

SC

$$O_1 < O_2 < O_3^{R_x} < O_3^{W_x} < O_4^{R_x} < O_4^{W_x} < O_5 < O_6$$
$$O_7 < O_8^x < O_8^y$$

PSO

$$O_1 < O_2 \quad O_5 < O_6$$
$$O_3^{R_x} < O_3^{W_x} \quad O_4^{R_x} < O_4^{W_x}$$
$$O_7 < O_8^x < O_8^y$$

make the error happen

Path Constraints

$$R_z^7 = 1$$

Failure Constraints

$$R_x^8 + 1 != R_y^8$$

Solution from the SMT solver:

O_1=1, O_2=2, O_3=3, O_5=4, O_7=5, O_8=6, O_4=7

Schedule: **1-2-3-5-7-8-4**

A Real Bug – $12 million loss of equipment

https://stackoverflow.com/questions/16159203/why-does-this-java-program-terminate-despite-that-apparently-it-shouldnt-and-d

```
Init: x=1, y=2
     T1
1: T2.start()
2: z=0
3: x++
4: y++
5: z=1
6: T2.join()

     T2
7: if (z==1)
8:  ✗ assert(x+1==y)
```

Read-Write Constraints

$$(R_z^7 = 0 \ \wedge O_7 < O_2) \vee$$
$$(R_z^7 = W_z^5 \ \wedge O_5 < O_7 \wedge (O_2 < O_5 \vee O_7 < O_2))$$

Memory Order Constraints

SC

$$O_1 < O_2 < O_3^{R_x} < O_3^{W_x} < O_4^{R_x}$$
$$< O_4^{W_x} < O_5 < O_6$$
$$O_7 < O_8^x < O_8^y$$

PSO

$$O_1 < O_2 \quad O_5 < O_6$$
$$O_3^{R_x} < O_3^{W_x} \quad O_4^{R_x} < O_4^{W_x}$$
$$O_7 < O_8^x < O_8^y$$

Path Constraints

$$R_z^7 = 1$$

Failure Constraints

$$R_x^8 + 1 != R_y^8$$

Solution from the SMT solver:

O_1=1, O_2=2, O_3=3, O_5=4, O_7=5, O_8=6, O_4=7

Schedule: **1-2-3-5-7-8-4**

A Real Bug – $12 million loss of equipment

https://stackoverflow.com/questions/16159203/why-does-this-java-program-terminate-despite-that-apparently-it-shouldnt-and-d

Init: x=1, y=2

```
      T1
1: T2.start()
2: z=0
3: x++
4: y++
5: z=1
6: T2.join()

      T2
7: if (z==1)
8:  ✗ assert(x+1==y)
```

Read-Write Constraints

$$(R_z^7 = 0 \ \wedge O_7 < O_2) \vee$$
$$(R_z^7 = W_z^5 \ \wedge O_5 < O_7 \wedge (O_2 < O_5 \vee O_7 < O_2))$$

Memory Order Constraints

SC

$$O_1 < O_2 < O_3^{Rx} < O_3^{Wx} < O_4^{Rx} < O_4^{Wx} < O_5 < O_6$$
$$O_7 < O_8^x < O_8^y$$

PSO

$$O_1 < O_2 \quad O_5 < O_6$$
$$O_3^{Rx} < O_3^{Wx} \quad O_4^{Rx} < O_4^{Wx}$$
$$O_7 < O_8^x < O_8^y$$

Path Constraints

$$R_z^7 = 1$$

Failure Constraints

$$R_x^8 + 1 != R_y^8$$

Solution from the SMT solver:

O_1=1, O_2=2, O_3=3, O_5=4, O_7=5, O_8=6, O_4=7

Schedule: **1-2-3-5-7-8-4**

Finding Known Errors

Program	LoC	#Threads	#Events	#Executions (Total Time)		
				ICB	ICB+DPOR	MCR
Example	79	3	32	77322(20s)	3782(3s)	**46(2s)**
Account	373	5	51	111(0.2s)	20(0.2s)	**2(0.3s)**
Airline	136	6	67	669(1.8s)	19(0.8s)	**9(3s)**
Allocation	348	3	125	15(0.1s)	8(0.3s)	**2(0.3s)**
BubbleSort	175	5	133	592(1.2s)	400(2.7s)	**4(4.8s)**
MTList	5759	27	685	**OOM**	5173(290s)	**8(97s)**
MTSet	7086	22	724	**OOM**	5480(267s)	**21(159s)**
PingPong	388	6	44	648(3s)	37(0.5s)	**2(0.7s)**
Pool	10K	3	170	24(0.3s)	6(0.3s)	**3(0.4s)**
StringBuffer	1339	3	70	12(0.1s)	10(0.5s)	**2(0.4s)**

Finding Known Errors

Program	LoC	#Threads	#Events	#Executions (Total Time)		
				ICB	ICB+DPOR	MCR
Example	79	3	32	77322(20s)	3782(3s)	**46(2s)**
Account	373	5	51	111(0.2s)	20(0.2s)	**2(0.3s)**
Airline	136	6	67	669(1.8s)	19(0.8s)	**9(3s)**
Pool	10K	3	170	24(0.3s)	6(0.3s)	**3(0.4s)**
StringBuffer	1339	3	70	12(0.1s)	10(0.5s)	**2(0.4s)**

MCR reduces #runs taken by BMC+POR by orders of magnitude!

MCR takes less time in half of the benchmarks

State-space Exploration

program	Finished✔ Timeout■ OOM✗			#Executions(Total Time)			#Race | #NPE		
	ICB	ICB+DPOR	MCR	ICB	ICB+DPOR	MCR	ICB	ICB+DPOR	MCR
Example	■	✔	✔	3.3M(1h)	26K(10s)	**50(2s)**	**7|0**	**10|0**	**10(0)**
Account	■	✔	✔	1.5M(1h)	875(2s)	**3(0.5s)**	**3(0)**	**3(0)**	**3(0)**
Airline	■	✔	✔	326K(1h)	3K(3.5s)	**8(4.5s)**	**0|0**	**0(0)**	**0(0)**
Allocation	✗	■	✔	-	1.4M(1h)	**30(5.6s)**	**0(0)**	**0(0)**	**0(0)**
BubbleSort	✗	■	■	-	327K(1h)	**14K(1h)**	**4(0)**	**6(0)**	**7|0**
MTList	✗	✗	■	-	-	**382(1h)**	**1|0**	**1|0**	**8|2***
MTSet	✗	✗	■	-	-	**457(1h)**	**5|0**	**5|0**	**6|5***
PingPong	■	■	✔	343K(1h)	973K(1h)	**413(13s)**	**6|1**	**7|1**	**7|1**
Pool	■	✔	✔	510K(1h)	1.5K(1.9s)	**3(0.9s)**	**0|0**	**0|0**	**0|0**
StringBuffer	■	✔	✔	1.3M(1h)	427(0.8s)	**3(0.4s)**	**0|0**	**0|0**	**0|0**

State-space Exploration

For most benchmarks, MCR finished in an hour
For half of the benchmarks, BMC+POR either out of memory or did not finish in an hour

program	Finished✔ Timeout■ OOM✗			#Executions(Total Time)			#Race | #NPE		
	ICB	ICB+DPOR	MCR	ICB	ICB+DPOR	MCR	ICB	ICB+DPOR	MCR
Allocation	✗	■	✔	-	1.4M(1h)	**30(5.6s)**	**0(0)**	**0(0)**	**0(0)**
BubbleSort	✗	■	■	-	327K(1h)	**14K(1h)**	**4(0)**	**6(0)**	**7|0**
MTList	✗	✗	■	-	-	**382(1h)**	**1|0**	**1|0**	**8|2***
MTSet	✗	✗	■	-	-	**457(1h)**	**5|0**	**5|0**	**6|5***
PingPong	■	■	✔	343K(1h)	973K(1h)	**413(13s)**	**6|1**	**7|1**	**7|1**
Pool	■	✔	✔	510K(1h)	1.5K(1.9s)	**3(0.9s)**	**0|0**	**0|0**	**0|0**
StringBuffer	■	✔	✔	1.3M(1h)	427(0.8s)	**3(0.4s)**	**0|0**	**0|0**	**0|0**

State-space Exploration

For most benchmarks, MCR finished in an hour
For half of the benchmarks, BMC+POR either out of memory or did not finish in an hour

MCR found 9 more data races and 7 more NPE than BMC+POR

program	Finished✔ Timeout■ OOM✗			#Executions(Total Time)			#Race | #NPE		
	ICB	ICB+DPOR	MCR	ICB	ICB+DPOR	MCR	ICB	ICB+DPOR	MCR
MTSet	✗	✗	■	-	-	437(1h)	3|0	3|0	0|3*
PingPong	■	■	✔	343K(1h)	973K(1h)	**413(13s)**	**6|1**	**7|1**	**7|1**
Pool	■	✔	✔	510K(1h)	1.5K(1.9s)	**3(0.9s)**	**0|0**	**0|0**	**0|0**
StringBuffer	■	✔	✔	1.3M(1h)	427(0.8s)	**3(0.4s)**	**0|0**	**0|0**	**0|0**

TSO and PSO Results

Program	DPOR (rInspect)			MCR			#Executions Reduction		
	SC	TSO	PSO	SC	TSO	PSO	SC	TSO	PSO
Dekker	248	252	508	62	98	155	4.0X	2.6X	3.3X
Lamport	128	208	2672	14	91	102	9.1X	2.3X	29.4X
Bakery	350	1164	2040	77	158	165	4.5X	7.1X	12.4X
Peterson	36	95	120	13	18	19	2.8X	5.3X	6.3X
StackUnsafe	252	252	252	29	46	108	8.7X	5.5X	2.3X
RVExample	1959	-	-	57	64	70	34.4X	-	-
Example (N=1 to 4)	4	4	-	2	2	10	2.0X	2.0X	-
	105	105	-	43	43	89	2.4X	2.4X	-
	4282	4282	-	296	296	819	14.5X	14.5X	-
	14840	14840	-	2767	2767	8420	5.4X	5.4X	-
Avg.	**435**	**394**	**1118**	**42**	**79**	**103**	**10.4X**	**5.0X**	**10.9X**

TSO and PSO Results

Program	DPOR (rInspect)			MCR			#Executions Reduction		
	SC	TSO	PSO	SC	TSO	PSO	SC	TSO	PSO
Dekker	248	252	508	62	98	155	4.0X	2.6X	3.3X
Lamport	128	208	2672	14	91	102	9.1X	2.3X	29.4X
Bakery	350	1164	2040	77	158	165	4.5X	7.1X	12.4X
Example (N=1 to 4)	[illegible]	[illegible]	-	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	-
	4282	4282	-	296	296	819	14.5X	14.5X	-
	14840	14840	-	2767	2767	8420	5.4X	5.4X	-
Avg.	**435**	**394**	**1118**	**42**	**79**	**103**	**10.4X**	**5.0X**	**10.9X**

MCR explores 5X-10X fewer executions than POR for TSO and PSO memory models

Maximal Causality Reduction Parallelization

- MCR is for massive parallelization
  - Online exploration with different seed interleavings is parallel
  - In each iteration, multiple seed interleavings can be generated in parallel

```
$\mathbf{parfor}\ r = read(t, x, v) \in \tau\ \mathbf{do}$
    $\mathbf{parfor}\ w = write(\_, x, v') \in \tau \wedge v' \neq v\ \mathbf{do}$
        $\Phi_{seed}(r, w) \equiv \Phi_{sync} \wedge \Phi_{rw}(r) \wedge \Phi_{rw}(w) \wedge \Phi_{value}(r, v')$
```

Results on Real Systems

program		ICB	ICB+DPOR	MCR	MCR-Parallel
Jigsaw	#Races	2	7	**20**	**38**
	#NPEs	1	2	**6**	**10**
	#Runs	307 **(OOM)**	425 **(OOM)**	**32**	**769**
Weblech	#Races	4	4	**6**	**7**
	#NPEs	0	0	**1**	**1**
	#Runs	1229 **(OOM)**	1072 **(OOM)**	**185**	**3311**

Results on Real Systems

program		ICB	ICB+DPOR	MCR	MCR-Parallel
	#Races	2	7	20	38
	#Runs	(OOM)	(OOM)	32	769
Weblech	#Races	4	4	6	7
	#NPEs	0	0	1	1
	#Runs	1229 (OOM)	1072 (OOM)	185	3311

Parallel-MCR explored many more states and detected many more data races and NPEs than MCR

Results on Real Systems

program		ICB	ICB+DP OR	MCR	MCR-Parallel
	#Races	2	7	**20**	**38**
Weblech	#Races	4	4	**6**	**7**
	#NPEs	0	0	**1**	**1**
	#Runs	1229 **(OOM)**	1072 **(OOM)**	**185**	**3311**

Parallel-MCR explored many more states and detected many more data races and NPEs than MCR

Found *five new bugs* (i.e., data races and NPEs)!

References

- ECOOP'17: Shiyou Huang and Jeff Huang, "Speeding Up Maximal Causality Reduction with Static Dependency Analysis"
- OOPSLA'16: Shiyou Huang and Jeff Huang,"Maximal Causality Reduction for TSO and PSO"
- PLDI'15: Jeff Huang,"Stateless Model Checking Concurrent Programs with Maximal Causality Reduction"
- PLDI'14: Jeff Huang, Patrick Meredith and Grigore Rosu "Maximal Sound Predictive Race Detection with Control Flow Abstraction"

Takeaway

- A new advance in Model-Checking
  - Maximal Causality Reduction (MCR)
- MCR dramatically improves scalability of BMC and POR
  - Minimal state-space exploration and embarrassingly parallel
- MCR open source
  - https://github.com/parasol-aser/JMCR