Lecture 17
Context Bounding Checkers

Zvonimir Rakamarić
University of Utah

Huge Number of Thread Schedules

- Concurrent program with *n* threads where each thread has *k* instructions has

$$(n*k)! \,/\, (k!)^n \geq (n!)^k$$

  interleavings
- Exponential in both n and k!
- Example: 5 threads with 5 instruction each

$$25! \,/\, 5!^5 = 6.2336074e{+}14$$

$$= 623 \text{ trillion interleavings}$$

Java Path Finder (JPF)

- Program checker for Java
- Properties to be verified
  - Program assertions
  - LTL properties
- Depth-first and breadth-first search, heuristics
  - Uses static analysis techniques to improve the efficiency of the search
- Requires a complete Java program
  - Cannot handle native code

Combating State Space Explosion

- Symmetry reduction
  - Search equivalent states only once
- Partial order reduction
  - Do not search thread interleavings that generate equivalent behavior
- Static analyses
  - Reduce state space using static analyses
- User-provided restrictions
  - Manually bound variable domains, array sizes,…

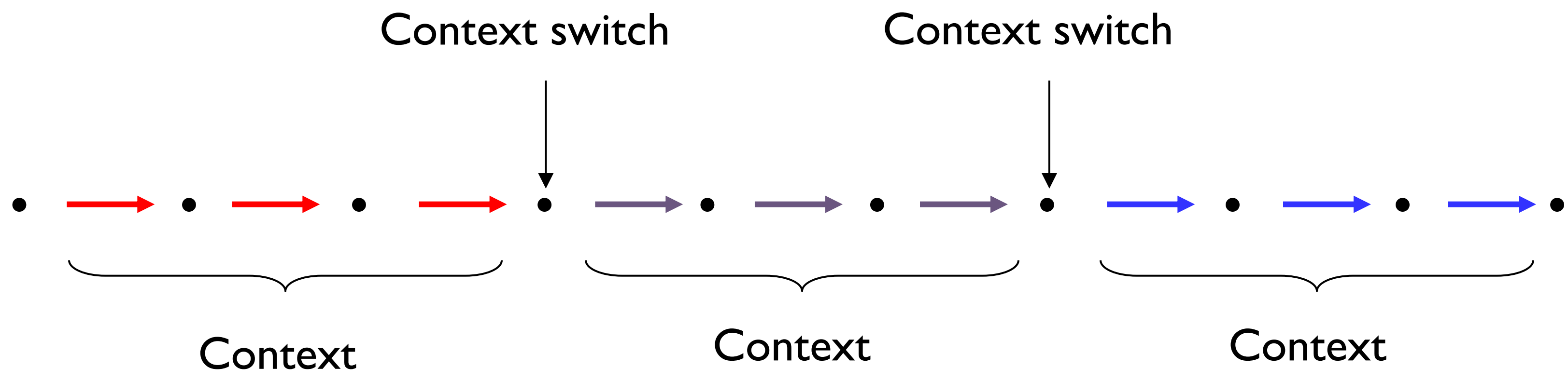
This Time

- Context-bounded verification of concurrent programs
- Sequentialization of concurrent programs

Context-Bounded Verification

slides acknowledgements: Shaz Qadeer, Madan Musuvathi

Context-Bounded Verification



- Many subtle concurrency errors are manifested in executions with few context switches
- Analyze all executions with few context switches

Context-Bounded Reachability Problem

- An execution is c-bounded if every thread has at most c contexts
- Does there exist a c-bounded execution from a state *S* to a state *E*?

CB Reachability is NP-Complete

- Membership in NP
  - Witness is an initial state and n*c sequences each of length at most $|L \times G|$

    n = # of threads, c = # of contexts

    L = # of program locations, G = # of global states
- NP-hardness
  - Reduction from the CIRCUIT-SAT problem

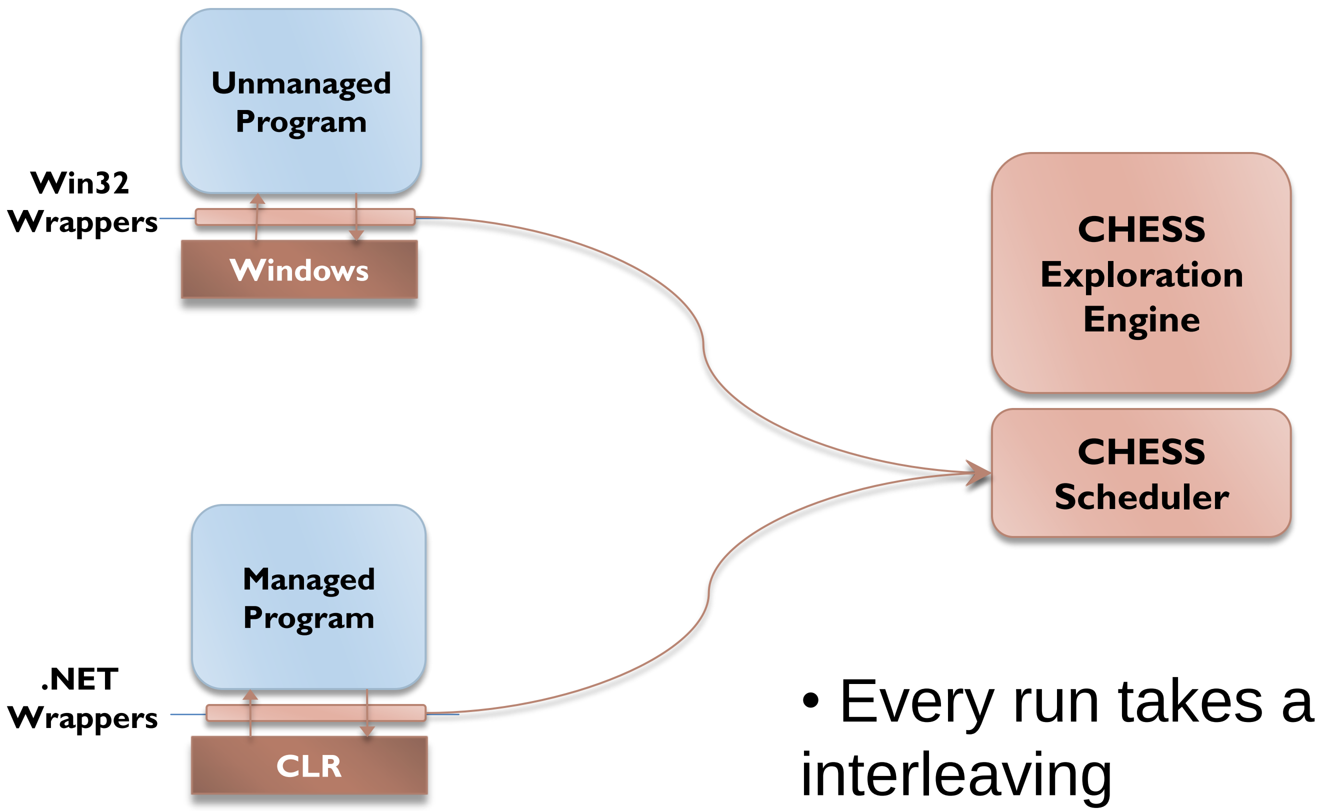
Complexity of Safety Verification

	Unbounded	Context-bounded
Finite-state systems	PSPACE complete	NP-complete
Pushdown systems	Undecidable	NP-complete

CHESS: Systematic Testing for Concurrency

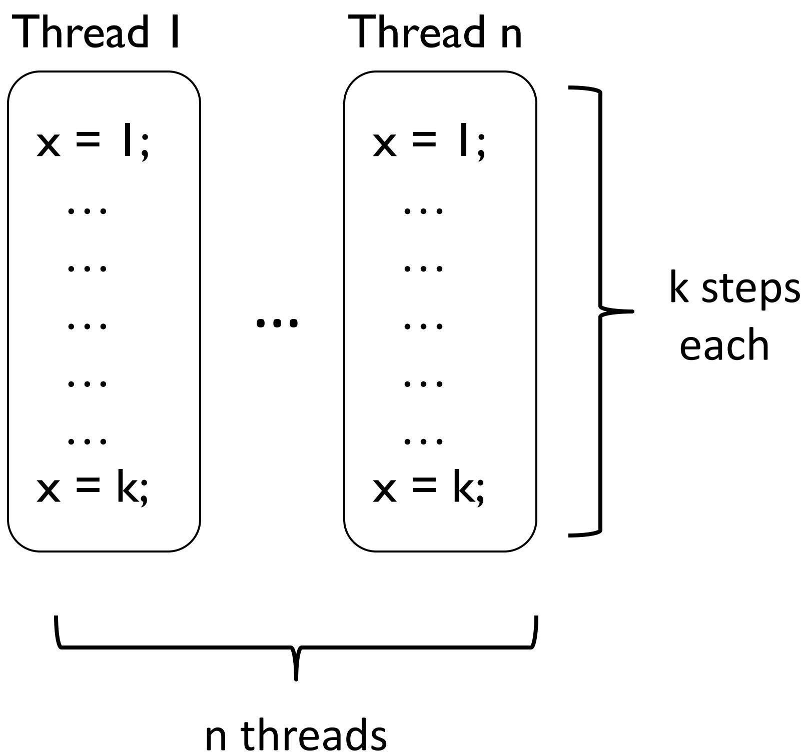
- CHESS is a user-mode scheduler
- Controls all scheduling nondeterminism
  - Replace the OS scheduler
- Guarantees:
  - Every program run takes a different thread interleaving
  - Reproduce the interleaving for every run

CHESS Architecture



- Every run takes a different interleaving
- Reproduce the interleaving for every run

State-Space Explosion

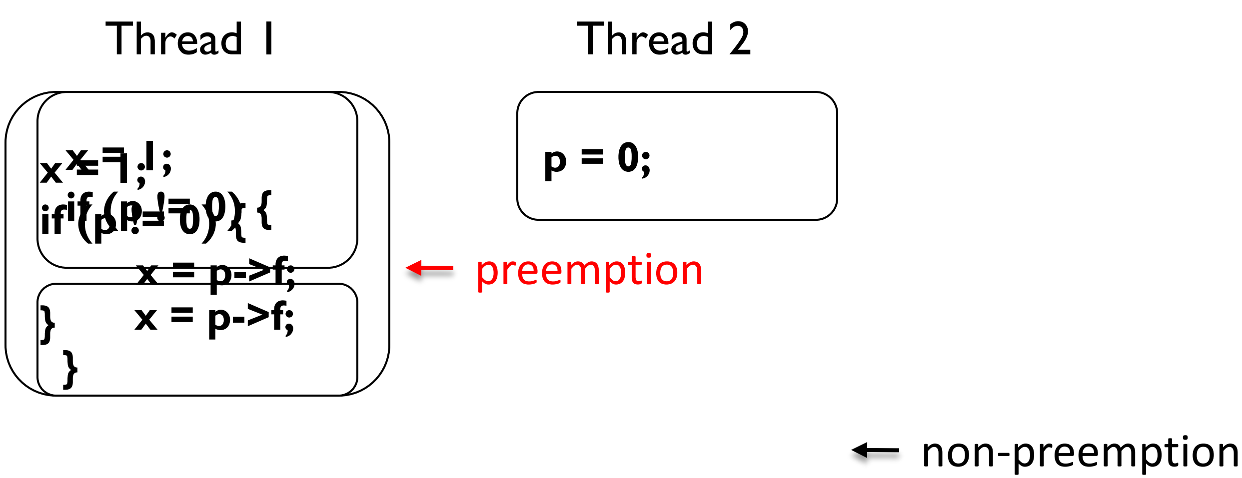


- Number of executions is $O(n^{nk})$
- Exponential in both n and k
  - Typically: n < 10, k > 1000
- Limits scalability to large programs

Goal: Scale CHESS to large programs (large k)

Preemption-Bounding

- Prioritize executions with small # of preemptions
- Two kinds of context switches:
  - Preemptions – forced by the scheduler
    - E.g., time-slice expiration
  - Non-preemptions – a thread voluntarily yields
    - E.g., blocking on an unavailable lock, thread end

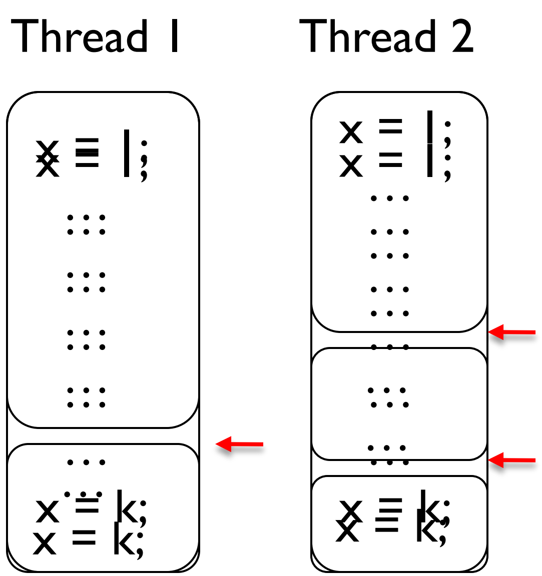


Preemption-Bounding in CHESS

- The scheduler has a budget of c preemptions
  - Nondeterministically choose the preemption points
- Resort to non-preemptive scheduling after c preemptions
- Once all executions explored with c preemptions
  - Try with c+1 preemptions

Property 1: Polynomial Bound

- Terminating program with fixed inputs and deterministic threads
  - n threads, k steps each, c preemptions
- Number of executions <= ${}_{nk}C_c$ * (n+c)!
  $= O((n^2k)^c * n!)$
- Exponential in n and c, but not in k!



- Choose c preemption points
- Permute n+c atomic blocks

Property 2: Simple Error Traces

- Finds smallest number of preemptions to the error
- Number of preemptions better metric of error complexity than execution length

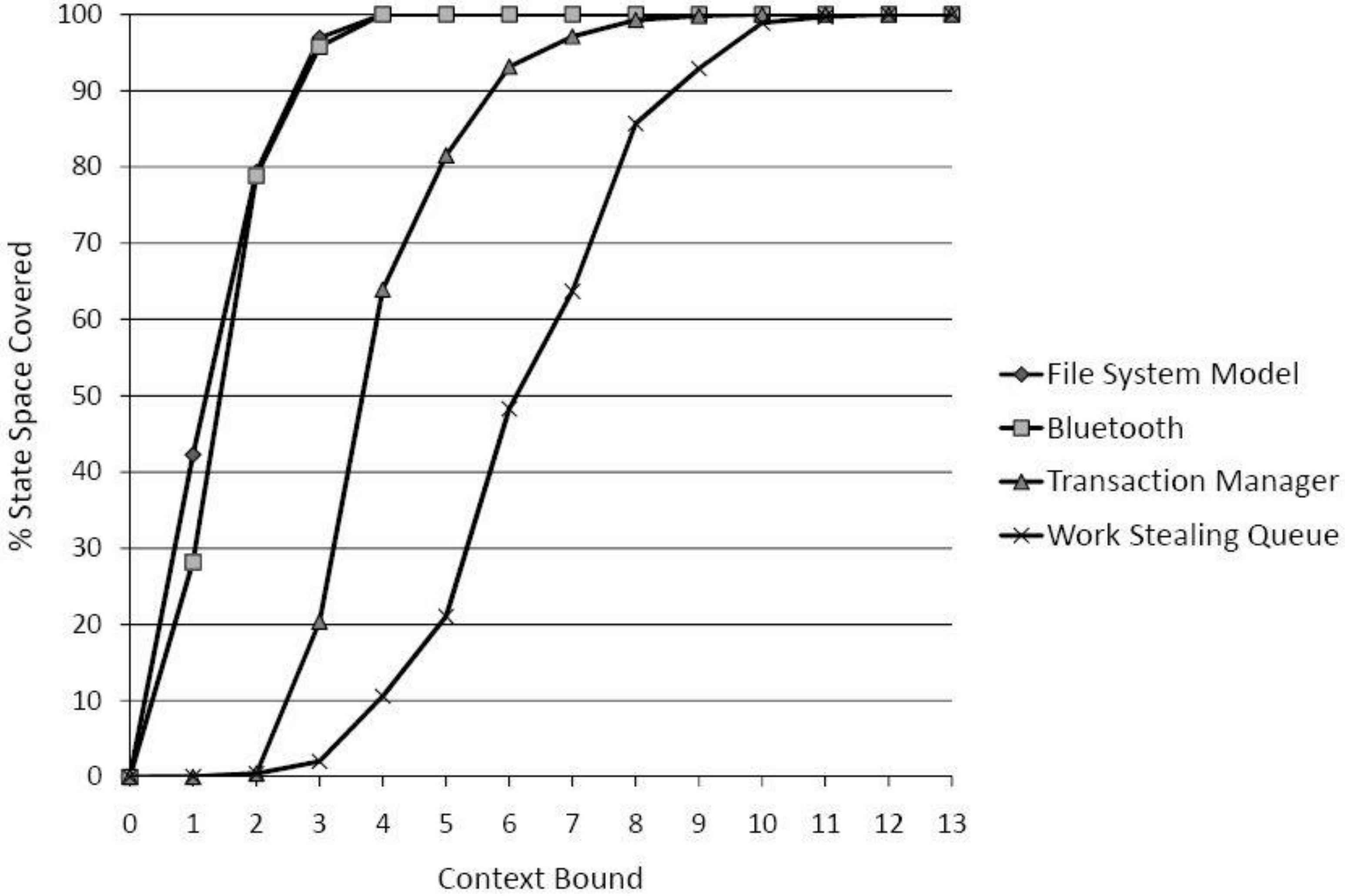
Property 3: Coverage Metric

- If search terminates with preemption-bound of c, then any remaining error must require at least c+1 preemptions
- Intuitive estimate for
  - The complexity of the bugs remaining in the program
  - The chance of their occurrence in practice

Property 4: Many Bugs with Few Preemptions

Program	kLOC	Threads	Preemptions	Bugs
Work-Stealing Queue	1.3	3	2	3
CDS	6.2	3	2	1
CCR	9.3	3	2	2
ConcRT	16.5	4	3	4
Dryad	18.1	25	2	7
APE	18.9	4	2	4
STM	20.2	2	2	2
PLINQ	23.8	8	2	1
TPL	24.1	8	2	9

Coverage vs Preemption-Bound



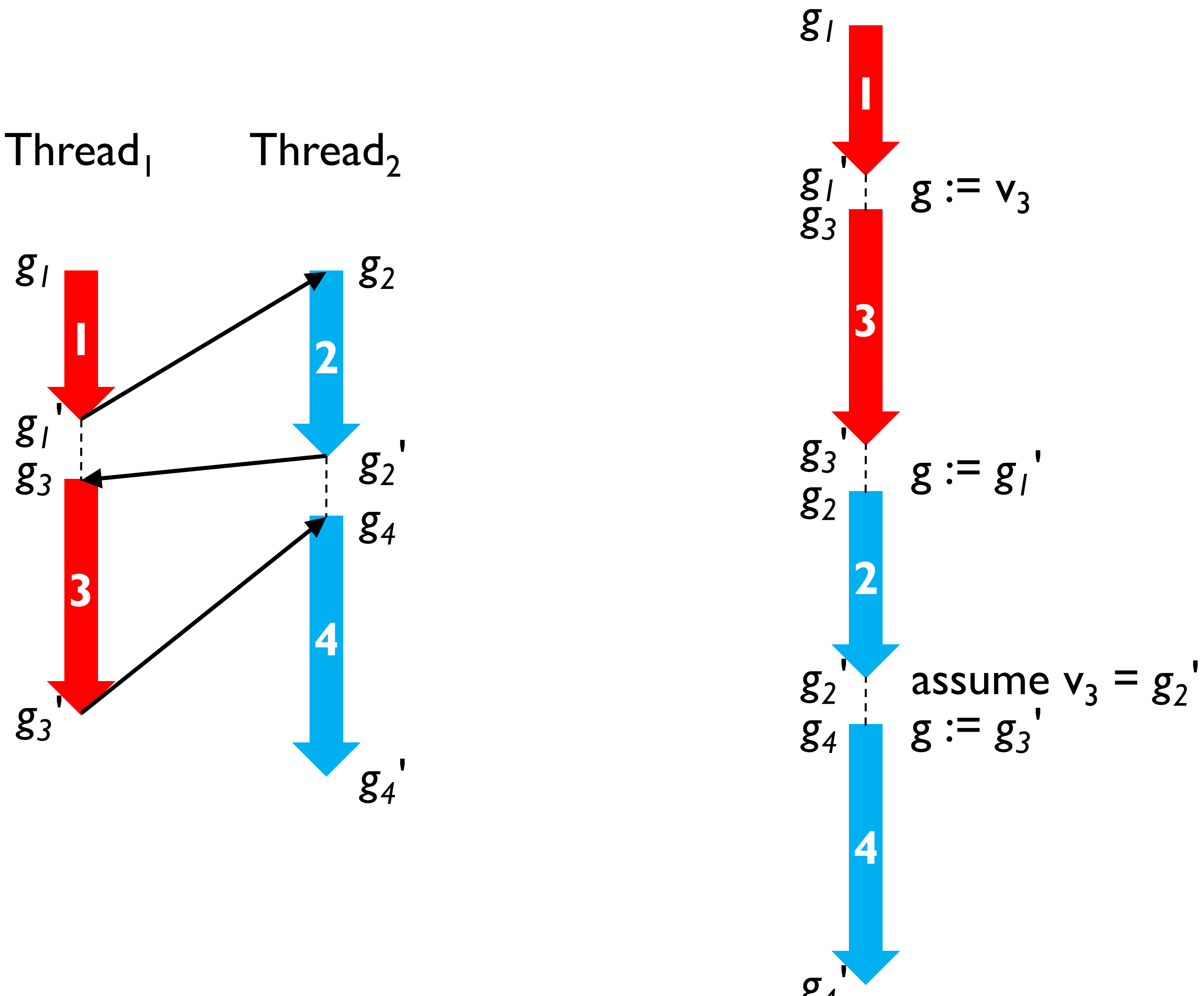
Sequentialization of Concurrent Programs

Concurrent Using Sequential

- Transform context bounded analysis of concurrent programs into analysis of sequential programs
- KISS [Qadeer, Wu, PLDI ‘04]
  - Only up to 2 context switches
- [Lal, Reps, CAV ‘08], [La Torre, Madhusudan, Parlato, CAV ‘09]
  - More general transformations, N context switches
  - Applied only on small, manually constructed Boolean programs

Simple Translation Example

- Translation of one concurrent trace
- Two threads: Thread_1, Thread_2
- One shared variable: g
- 3 context switches, 4 execution segments (or contexts)
- Main idea [Lal, Reps, CAV '08]
  - Avoid storing local state
  - Introduce unconstrained symbolic "prophecy" values instead of still unavailable "future" values
  - Constrain them when "future" values become available



Thread1
Thread2
g1
1
g2
2
g1'
g2'
g3
3
g4
4
g3'
g4'
g1
1
g1'
g := v3
g3
3
g3'
g := g1'
g2
2
g2'
assume v3 = g2'
g4
g := g3'
4
g4'

Lal-Reps Translation

- N contexts per thread, shared variable g

$T_1 \,||\, T_2$

assert F

int $g_1, g_2, \ldots g_N, v_2, \ldots v_N$;

int k := 1;

assume ($g_2 = v_2$ && $g_3 = v_3$ ... g

INIT;

L_1: T_1^s;

L_2: T_2^s;

L_3: END;

assert F

st → switch(k):

case 1: Schedule; st[g_1/g];

case 2: Schedule; st[g_2/g];

…

assume ($g_1 = v_2$ && $g_2 = v_3$ …);

<u>Schedule</u>

if (k <= N) {if (*) k++;}

else {k := 1; goto L_{i+1}};

Field Abstraction Example

- Fields = {`f`, `g`}
- Tracked fields = {`f`}

Before

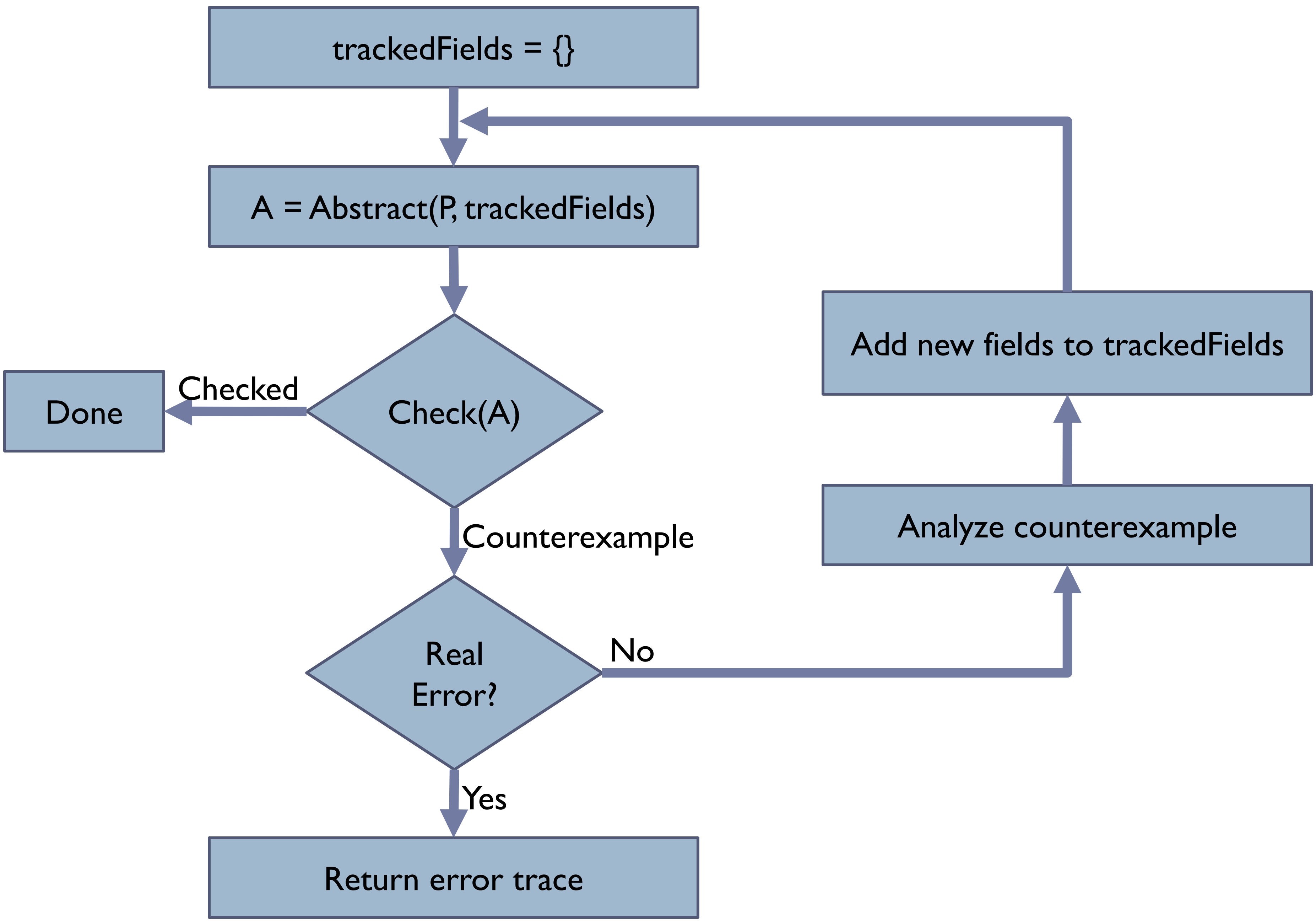
```
tmp = x->f;
tmp = x->g;
y->g = tmp;
```

Abstraction…

```
tmp = x->f;
tmp = nondet();
y->g = tmp;
```

Field Abstraction CEGAR

- How to discover tracked fields automatically?
- Algorithm based on CounterExample Guided Abstraction Refinement (CEGAR) framework


trackedFields = {}
A = Abstract(P, trackedFields)
Check(A)
Checked
Done
Counterexample
Real
Error?
No
Yes
Return error trace
Analyze counterexample
Add new fields to trackedFields

Experimental Results

- Prototype implementation: STORM
- 4 Windows Device Drivers
- Harness
  - Creates driver request that gets processed concurrently by multiple routines
    - Dispatch | Cancellation
    - Dispatch | Cancellation | Completion
    - Dispatch | Cancellation | Completion | DPC
- Checked property
  - Driver request cannot be used after it has been completed (i.e. use after free)

Varying Number of Contexts N

- Manually provided tracked fields

Driver	kLOC	#T	Routine	1	2	3	4	5
usbsamp Bug found!	4	3	read	17.9	37.7	65.8	66.8	85.2
			write	17.8	48.8	52.3	74.3	109.7
			ioctl	4.4	5.0	5.1	5.3	5.4
usbsamp_fix	4	3	read	16.9	28.2	38.6	46.7	47.5
			write	18.1	32.2	46.9	52.5	63.6
			ioctl	4.8	4.7	5.1	5.1	5.2
mqueue	14	4	read	62.1	161.5	236.2	173.0	212.4
			write	48.6	113.4	171.2	177.4	192.3
			ioctl	120.6	198.6	204.7	176.1	199.9
daytona	22	2	ioctl	3.4	3.8	4.2	4.5	5.6
serial	32	3	read	36.5	95.4	103.4	240.5	281.4
			write	37.3	164.3	100.8	233.0	649.8

Field Abstraction CEGAR

- N=2

Driver	Routine	#Fields Total	#TFieds Manual	#TFields CEGAR	#CEGAR Iterations	Time (s)
daytona	ioctl	53	3	3	3	244.3
mqueue	read	72	7	9	9	3446.3
	write			8	8	3010.0
	ioctl			9	9	3635.6
usbsamp_fix	read	113	1	3	3	4382.4
	write			4	4	2079.2
	ioctl			0	0	21.7
serial	read	214	5	5	5	3013.7
	write			4	3	1729.4

Bug Found (usbsamp)

- Sample driver in WinDDK
  - Example of how to write device drivers
    - Copy-pasted by driver vendors
  - Checked using existing tools
- Bug confirmed and fixed
- Requires 3 context switches
    - SLAM (SDV) – checks sequential code
    - KISS – only up to 2 context switches

→ Bug could not be found by other tools

Bug Found

Thread$_1$

Dispatch Routine

```
ReadRoutine(req) {
  ...
  WdfRequestMarkCancelable(
    req, CancelRoutine);
  ...
  WdfRequestComplete(req);
  ...
}
```

Thread$_2$

Cancellation Routine

```
CancelRoutine(req) {
  assume (CancelRoutineSet
   && !reqCompleted);
  ...
  GetRequestContext(req);
  ...
}
```