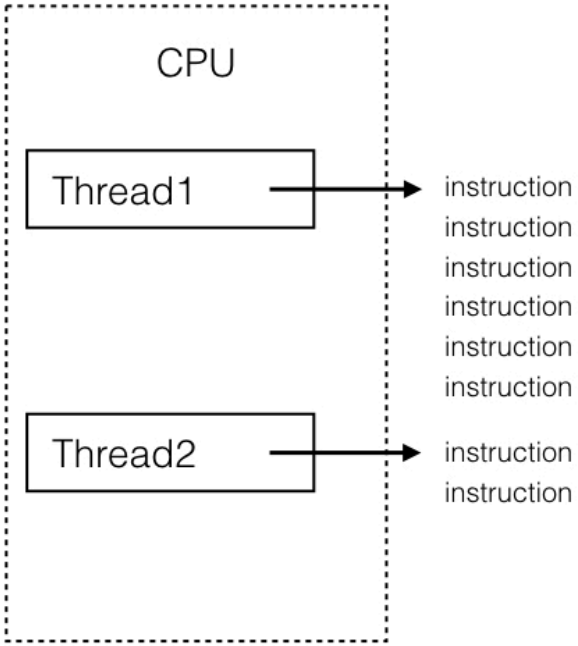
Verifying Multi-Threaded Programs

Susanne van den Elsen

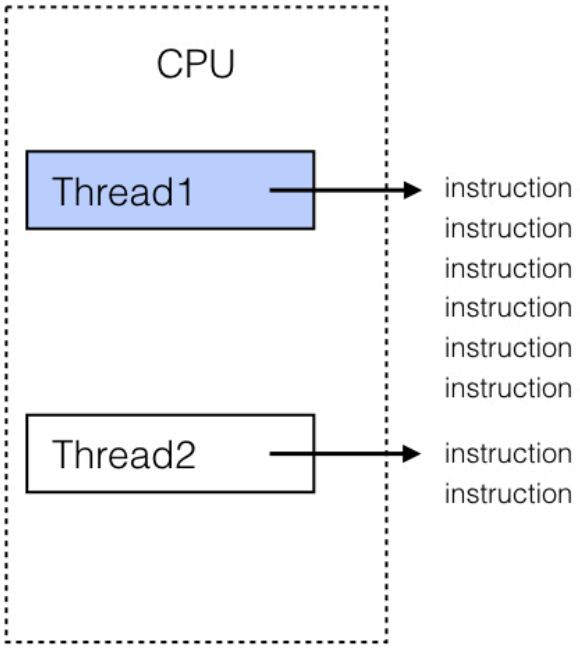
Native Instruments

February 21, 2017

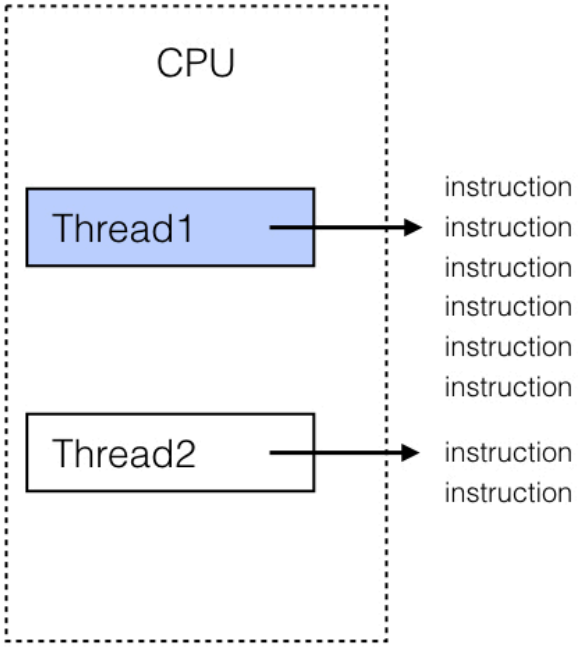
Threads


CPU
Thread1
instruction
instruction
instruction
instruction
instruction
instruction
Thread2
instruction
instruction

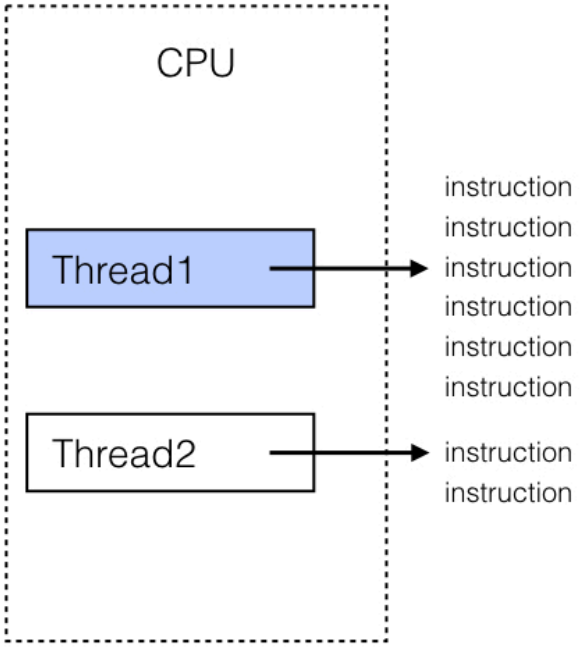
Threads


CPU
Thread1
instruction
instruction
instruction
instruction
instruction
instruction
Thread2
instruction
instruction

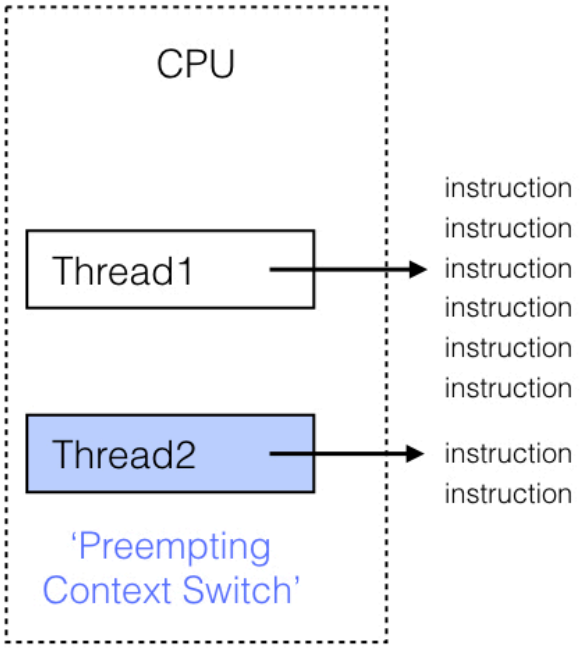
Threads


CPU
Thread1
Thread2
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction

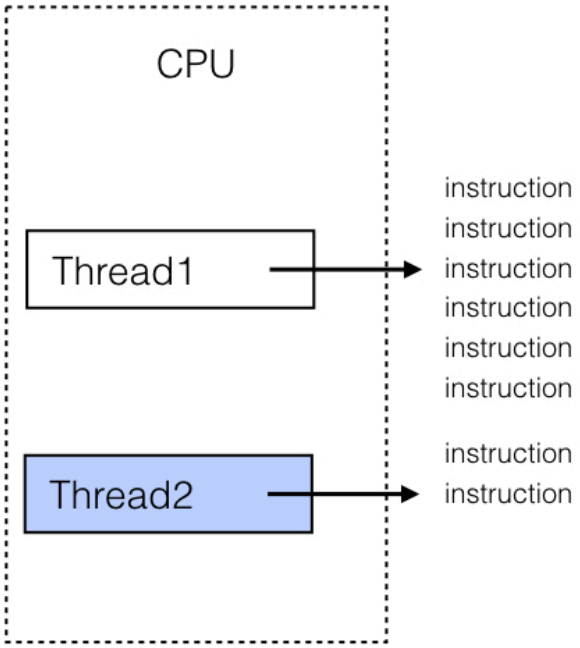
Threads


CPU
Thread1
Thread2
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction

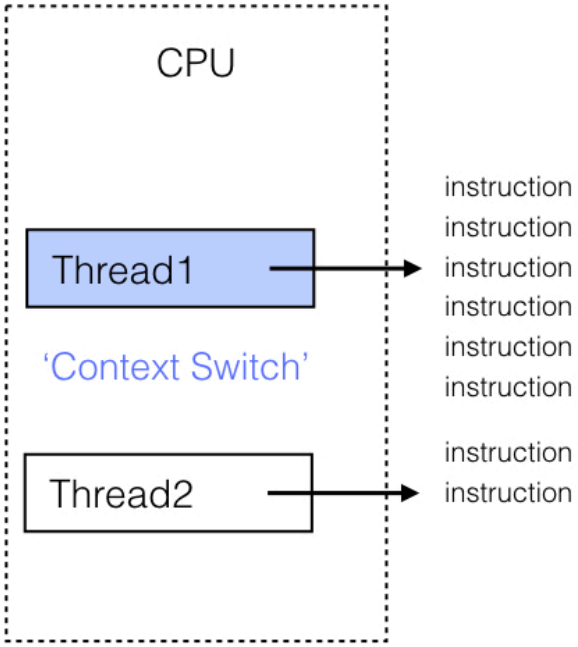
Threads


CPU
Thread1
Thread2
'Preempting
Context Switch'
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction

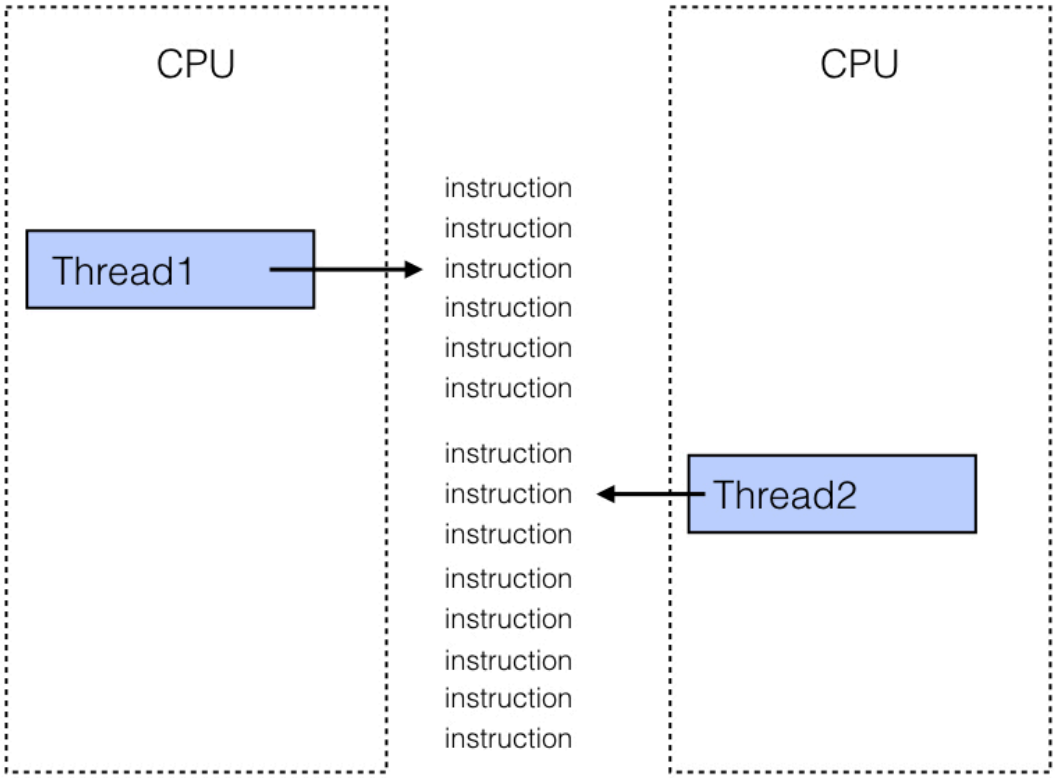
Threads


CPU
Thread1
Thread2
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction

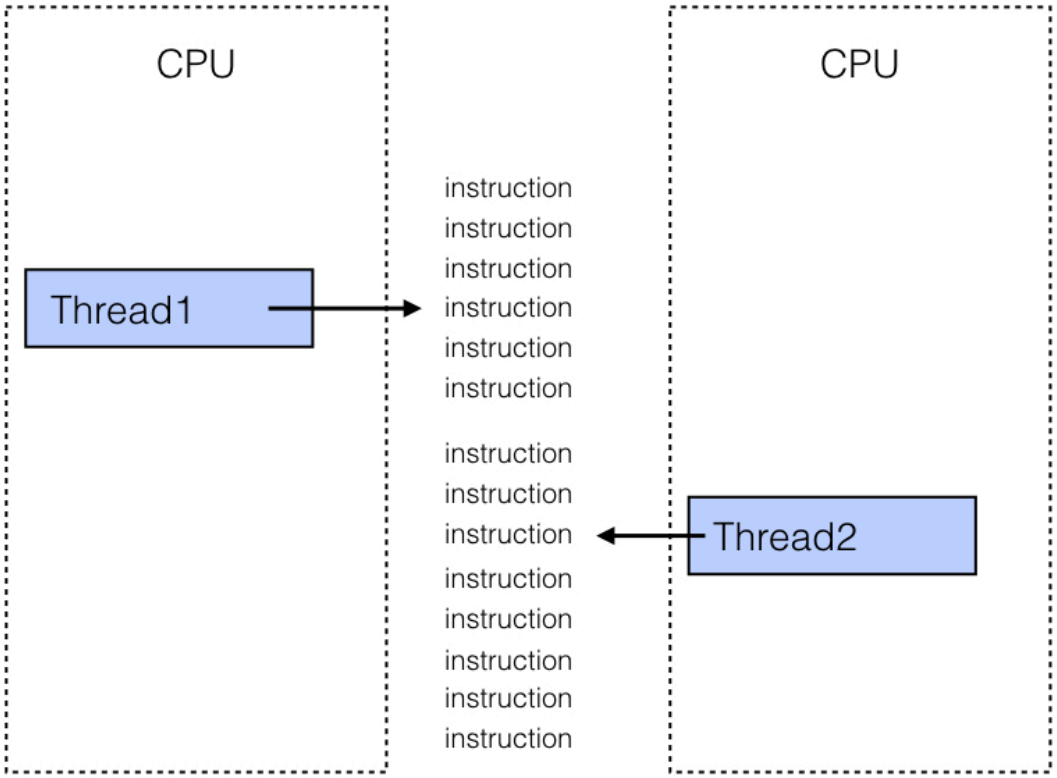
Threads


CPU
Thread1
'Context Switch'
Thread2
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction

Threads


CPU
CPU
Thread1
Thread2
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction

Threads


CPU
CPU
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction
instruction
Thread1
Thread2

What's Hard About Concurrency?

Shared Resources

```
int x, y, z = 0;

thread 0                    thread 1
--------                    --------

if (x == 1)                 x = 1;
{                           z = x + y;
   y = 1;                   std::cout << z << "\n";
}
```

What's Hard About Concurrency?

Shared Resources

```
int x, y, z = 0;

thread 0                    thread 1
--------                    --------

if (x == 1)                 x = 1;
{                           z = x + y;
   y = 1;                   std::cout << z << "\n";
}
```

Datarace:

Two (or more) threads access the same memory location *concurrently* and at least one of them is a *write*

What's Hard About Concurrency?

Synchronisation

```
array<int,2> nr_meals_eaten = {0, 0};
mutex fork1, fork2;

philosopher 0                philosopher 1
-------------                -------------

lock(fork1);                 lock(fork2);
lock(fork2);                 lock(fork1);
++nr_meals_eaten[0];         ++nr_meals_eaten[1];
unlock(fork2);               unlock(fork1);
unlock(fork1);               unlock(fork2);
```

What's Hard About Concurrency?

Synchronisation

```
array<int,2> nr_meals_eaten = {0, 0};
mutex fork1, fork2;

philosopher 0              philosopher 1
-------------              -------------

lock(fork1);
                           lock(fork2);
lock(fork2);
                           lock(fork1);
++nr_meals_eaten[0];       ++nr_meals_eaten[1];
unlock(fork2);             unlock(fork1);
unlock(fork1);             unlock(fork2);
```

What's Hard About Concurrency?

Synchronisation

```
array<int,2> nr_meals_eaten = {0, 0};
mutex fork1, fork2;

philosopher 0              philosopher 1
-------------              -------------

lock(fork1);
                           lock(fork2);
lock(fork2);
                           lock(fork1);
++nr_meals_eaten[0];       ++nr_meals_eaten[1];
unlock(fork2);             unlock(fork1);
unlock(fork1);             unlock(fork2);
```

Deadlock:

Two or more threads wait for each other indefinitely

What's Hard About Concurrency?

State Space Explosion

What's Hard About Concurrency?

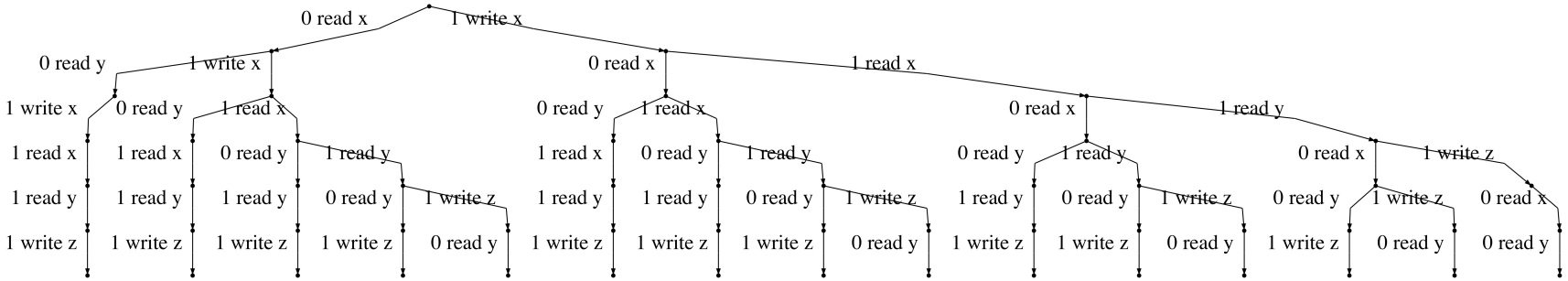
State Space Explosion

```
int x, y, z = 0;

thread 0                    thread 1
--------                    --------

if (x == 1)                 x = 1;
{                           z = x + y;
   y = 1;                   std::cout << z << "\n";
}
```

What's Hard About Concurrency?

State Space Explosion

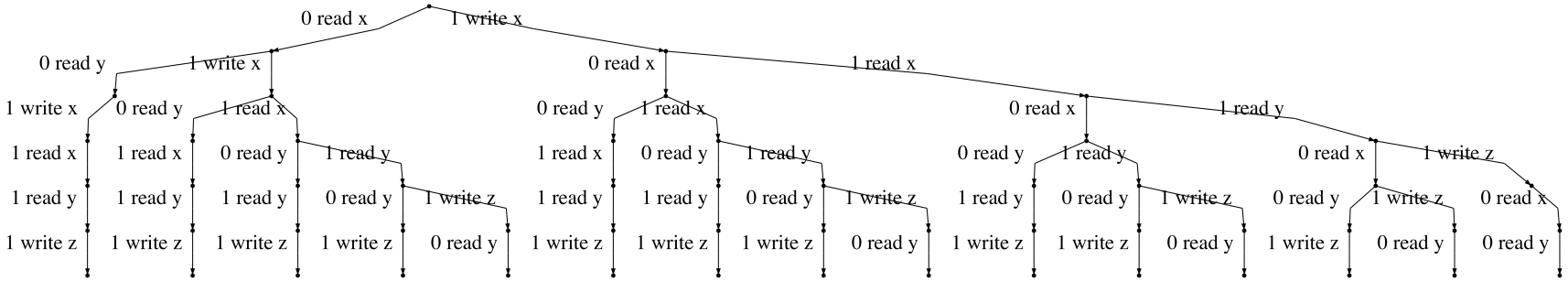
Number of possible thread interleavings is *exponential* in the number of threads and instructions!



What's Hard About Concurrency?

Nondeterminism

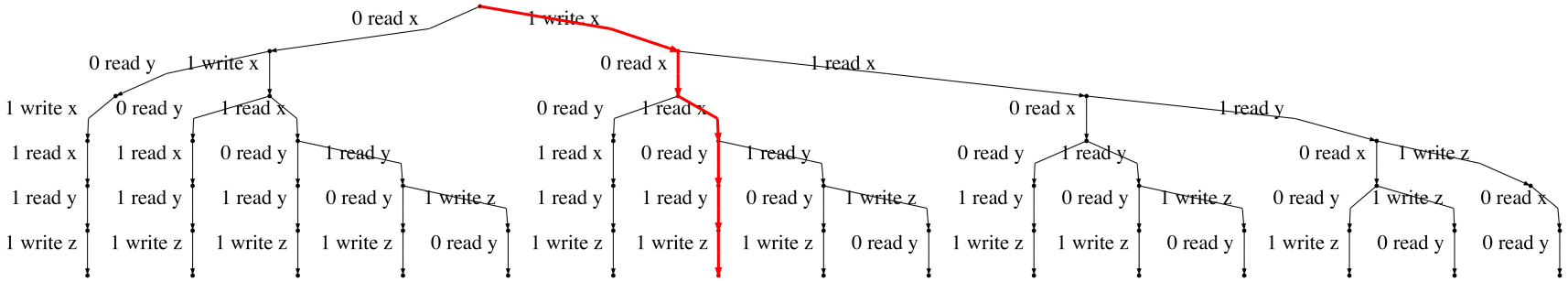
The interleaving of threads is out of the programmer's control



What's Hard About Concurrency?

Nondeterminism

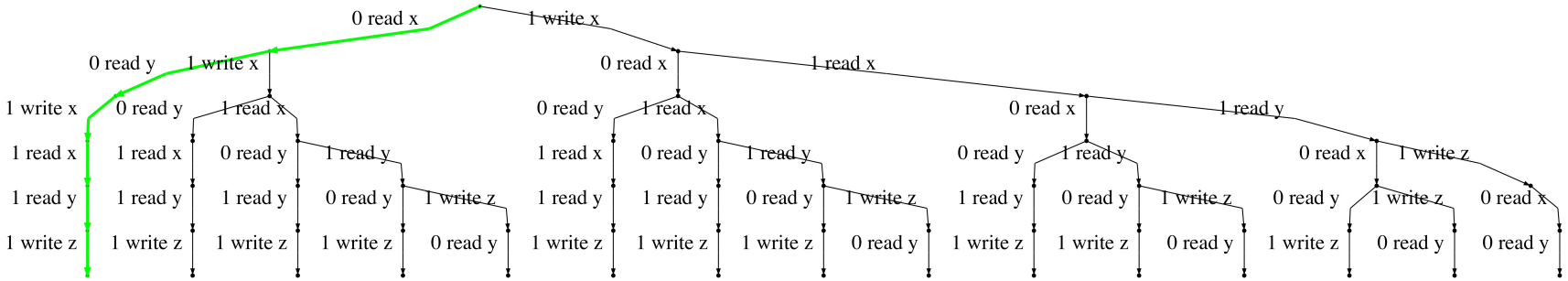
The interleaving of threads is out of the programmer's control



What's Hard About Concurrency?

Nondeterminism

The interleaving of threads is out of the programmer's control



Concurrency Error Detectors

Concurrency Error Detectors

ThreadSanitzer

```
clang++ program_under_test.cpp -fsanitize=thread -o program_under_test
./program_under_test
```

Concurrency Error Detectors

ThreadSanitzer

```
clang++ program_under_test.cpp -fsanitize=thread -o program_under_test
./program_under_test
```

Compiler Instrumentation (LLVM IR)

Instrument every potentially *visible* instruction (e.g. `load`, `store`)

Concurrency Error Detectors

ThreadSanitzer

```
clang++ program_under_test.cpp -fsanitize=thread -o program_under_test
./program_under_test
```

Compiler Instrumentation (LLVM IR)

Instrument every potentially *visible* instruction (e.g. `load`, `store`)

```
define void @function() {
__tsan_func_entry(caller)
%2 = alloca i32
__tsan_store(%2)
store i32 %0, i32* %2
__tsan_load(%2)
%4 = load i32, i32* %2
ret i32 %4
__tsan_func_exit()
}
```

Concurrency Error Detectors

ThreadSanitzer

Runtime library

The `__tsan_` functions are callbacks to a Runtime Library (part of clang-rt)

Maintains a state maschine using shadow state for analysis

Concurrency Error Detectors

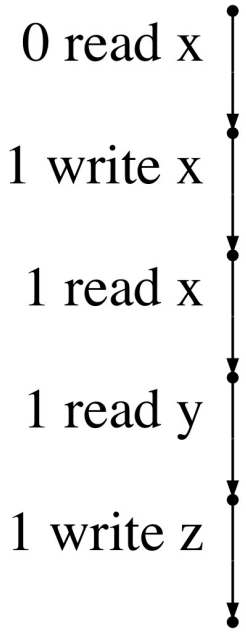
Helgrind

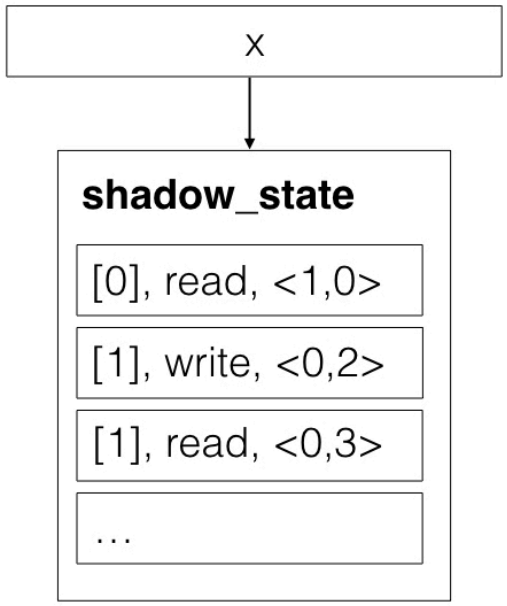
Binary Instrumentation

```
valgrind --tool=helgrind -v program
```

- Program is run on synthetic CPU provided by Valgrind Core
- Threads are fully serialzed (only single CPU used)

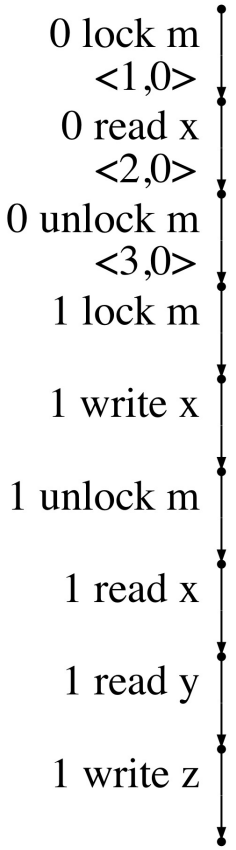
ThreadSanitizer & Helgrind: Data Races





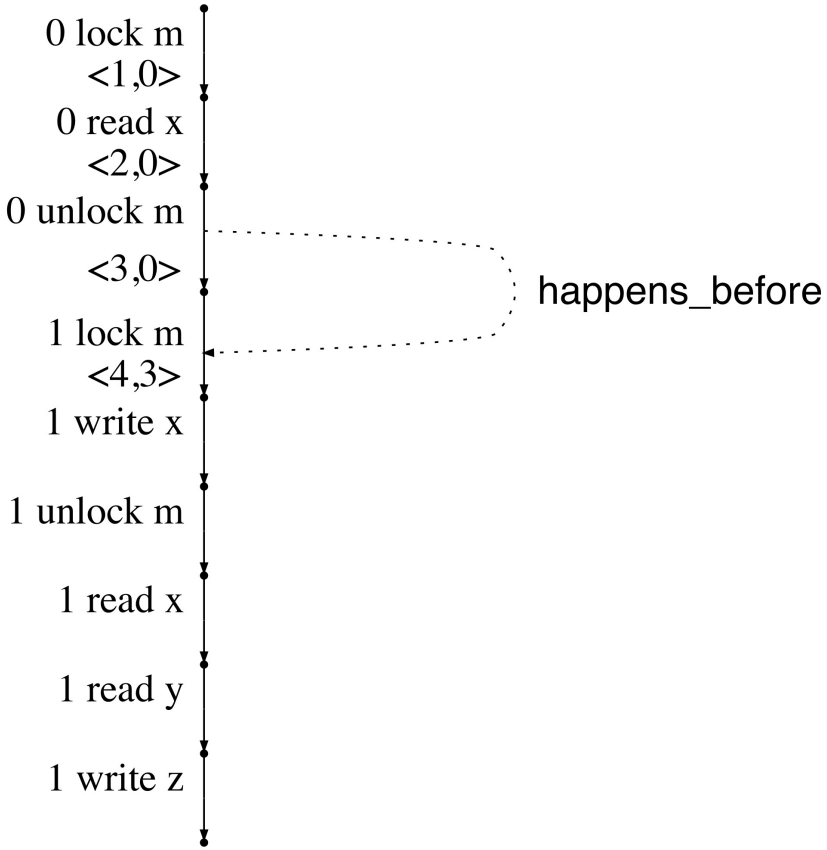
ThreadSanitizer & Helgrind: Data Races

Using the *happens-before* relation:



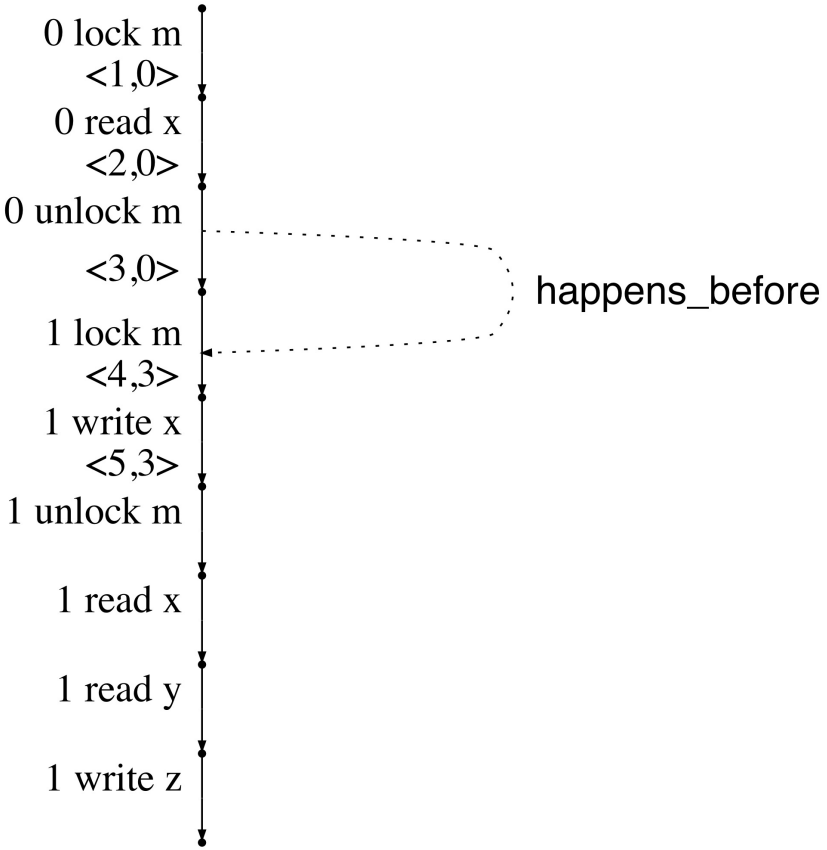
ThreadSanitizer & Helgrind: Data Races

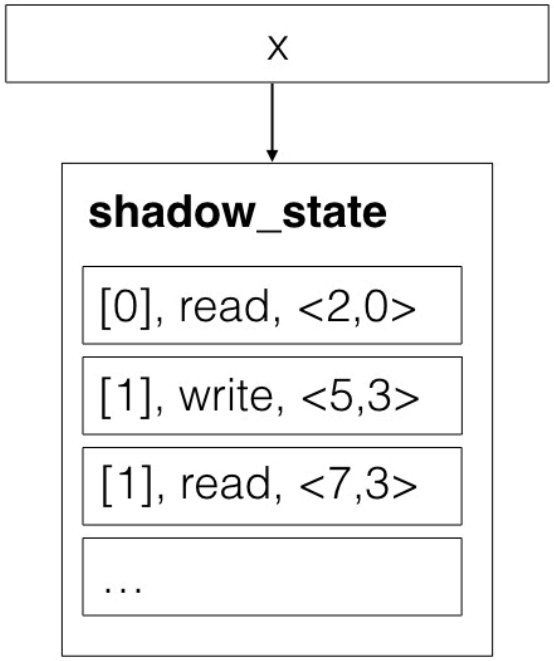
Using the *happens-before* relation:



ThreadSanitizer & Helgrind: Data Races

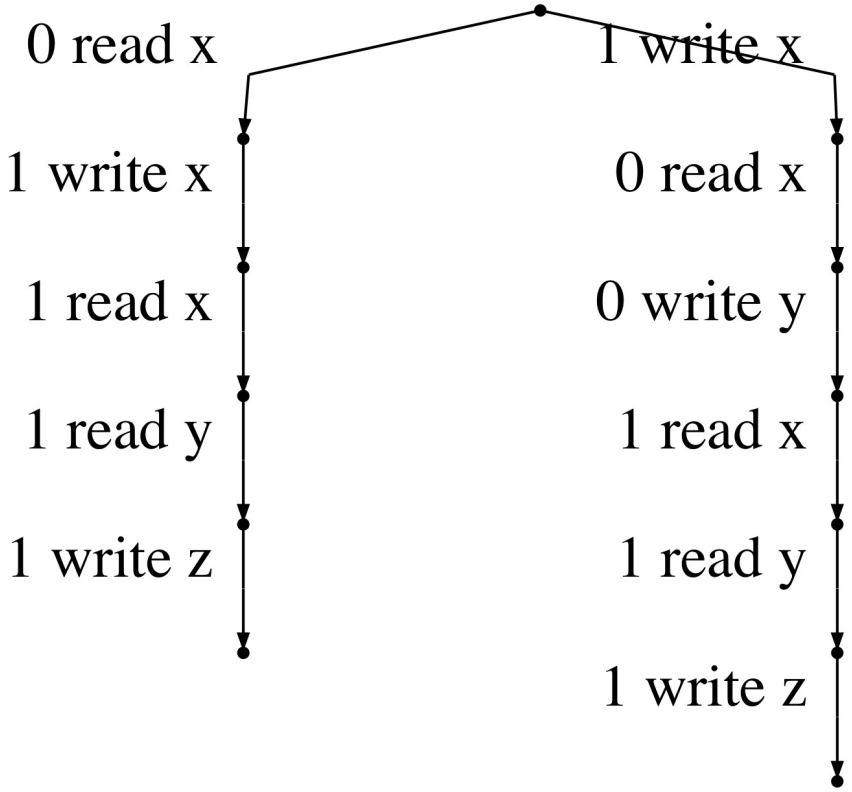
Using the *happens-before* relation:



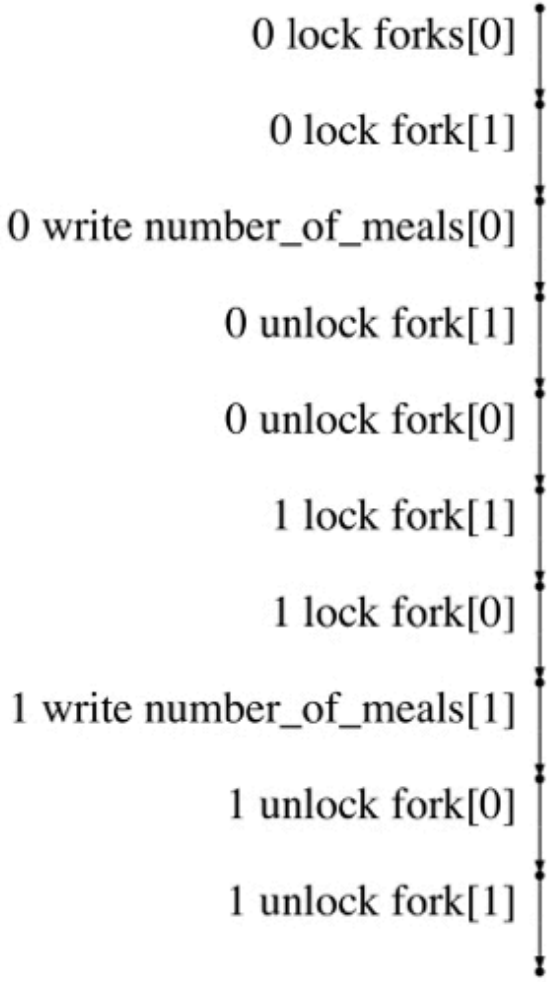


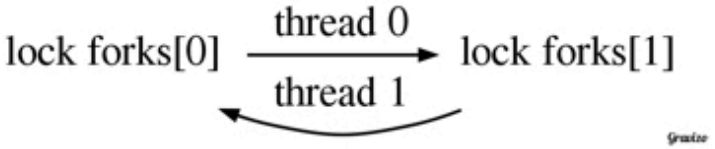
ThreadSanitizer & Helgrind: Data Races

Limited information in *single* execution:



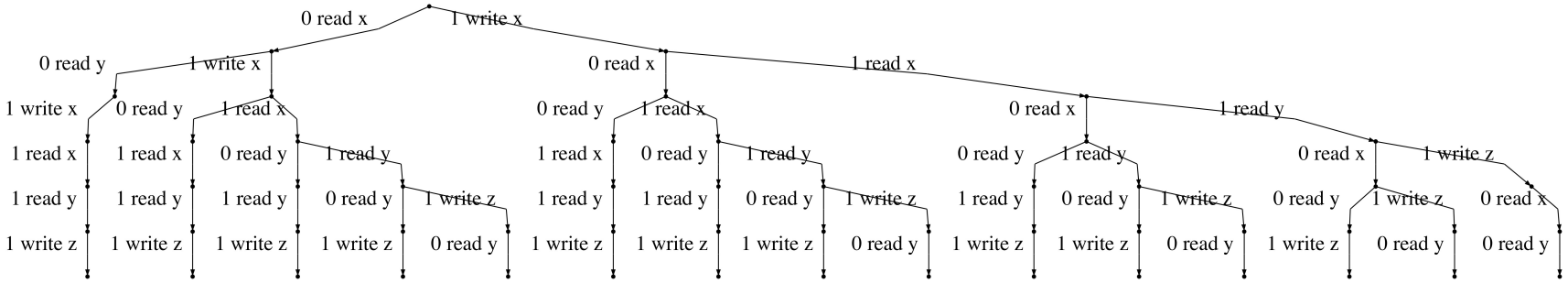
ThreadSanitizer: Deadlock





Systematic Exploration

Systematic Exploration



Taking Control Over the Thread Interleavings

LLVM IR Instumentation Pass

Replace

```
pthread_create(pid, attr, start_routine, args)
pthread_join(pid)
```

by

```
wrapper_spawn_thread(pid, attr, start_routine, args)
wrapper_pthread_join()
```

Scheduler

```
void wrapper_spawn_thread(pid, attr, start_routine, args)
{
   semaphores.insert({ pid, semaphore() })
   pthread_create(pid, attr, start_routine, args)
}
```

Taking Control Over the Thread Interleavings

LLVM IR Instumentation Pass

Insert call to potentially *visible* instructions

```
wrap_post_task(instruction(store, %2, is_atomic))
store i32 %0, i32* %2
```

Scheduler

```
void wrapper_post_task(instruction)
{
   task_pool.insert({ this_thread::id(), instruction });
   semaphores[this_thread::id()].wait();
   // wait for turn

   task_pool.remove(this_thread::id());
   // perform instructions
}
```

Taking Control Over the Thread Interleavings

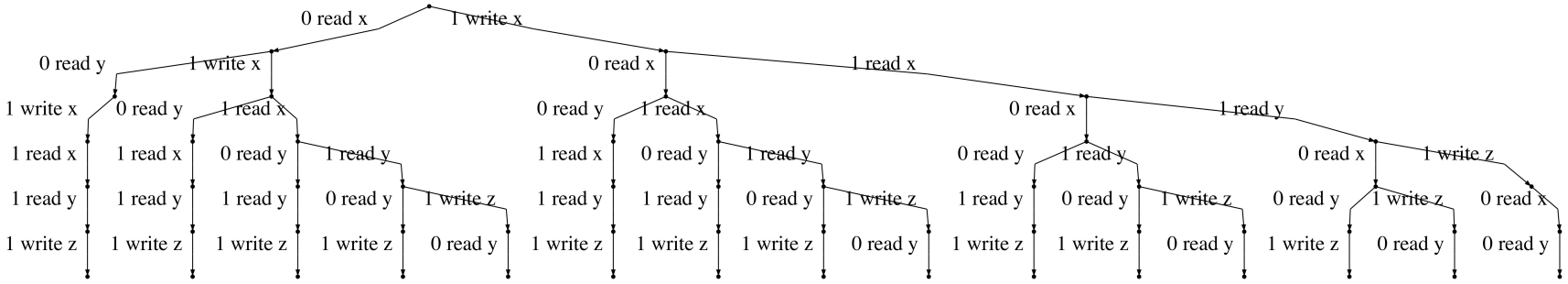
Scheduler

```
void scheduler_thread(schedule)
{
   for (thread_id : schedule)
   {
      wait_until(task_pool.contains_key(thread_id));
      current_thread_id = thread_id;
      semaphore[thread_id].post();
      // notify the waiting thread
   }
}
```

Systematic Exploration

Simple depth-first exploration

State space explosion: becomes infeasible for larger programs



State Space Reduction

- Explore only a subset of interleavings
- Provide coverage guarantees: quantify the class / number of interleavings seen

Bounded Search

```
bounded_search(program, bound_function, bound)
{
   for (interleaving : program.interleavings())
   {
      if (bound_function(interleaving) <= bound)
      {
         explore(interleaving);
      }
   }
}
```

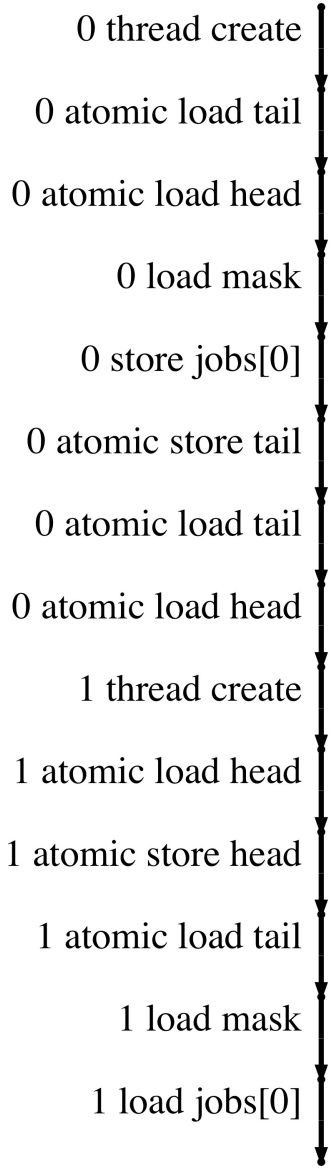
Bounded Search

```
bounded_search(program, bound_function, bound)
{
   for (interleaving : program.interleavings())
   {
      if (bound_function(interleaving) <= bound)
      {
         explore(interleaving);
      }
   }
}
```

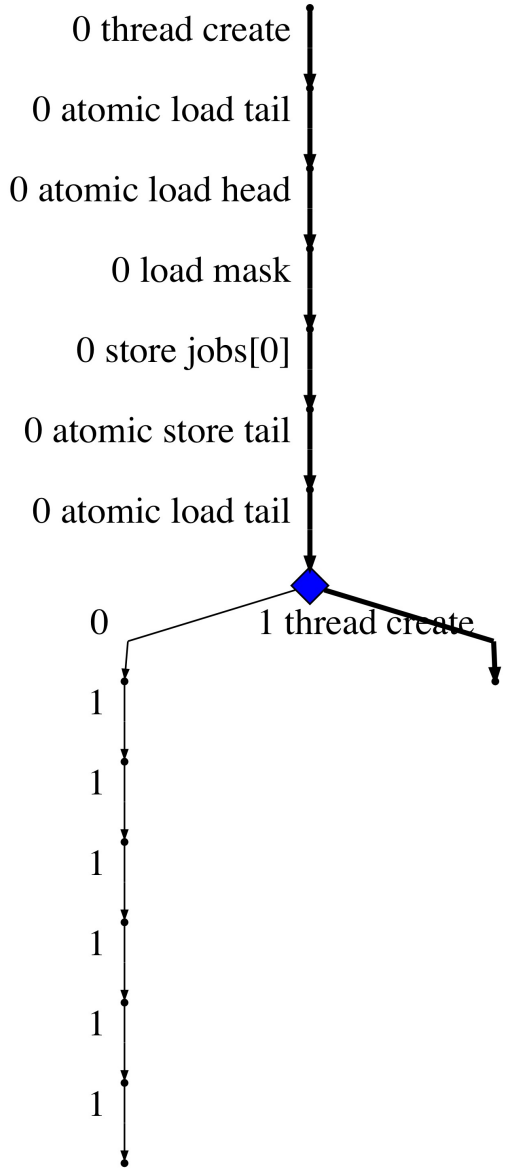
Bound Functions

- Number of context-switches
- Number of preemptions

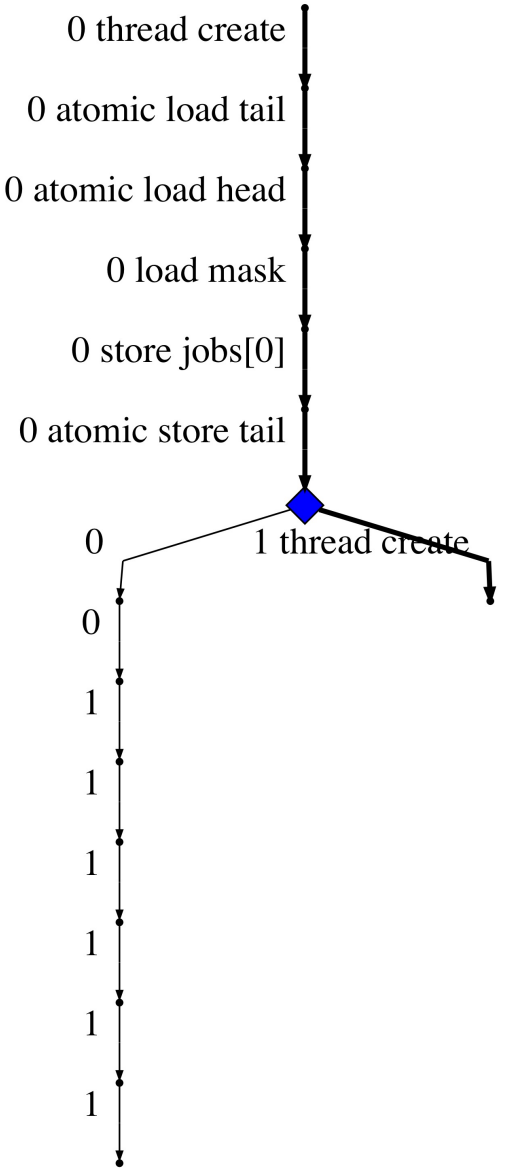
Bounded Search with Preemption Bound = 0



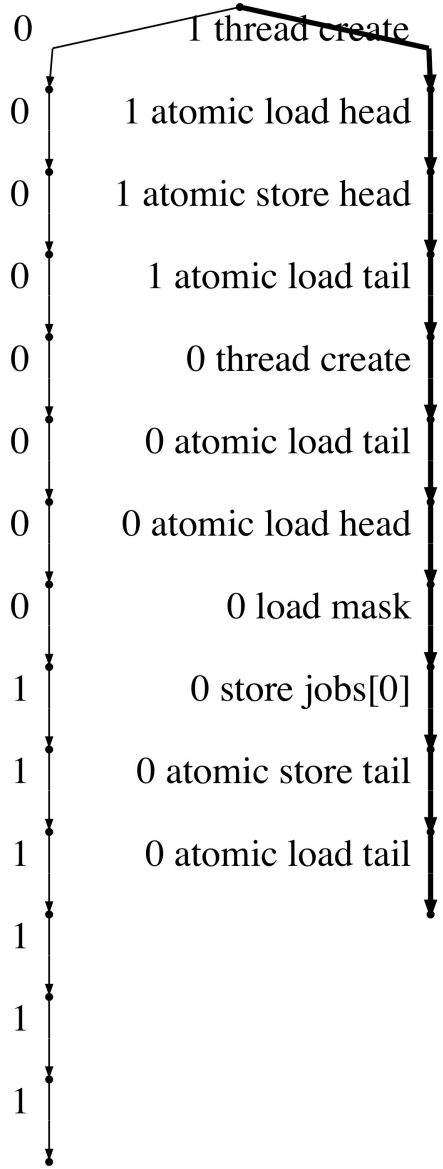
Bounded Search with Preemption Bound = 0



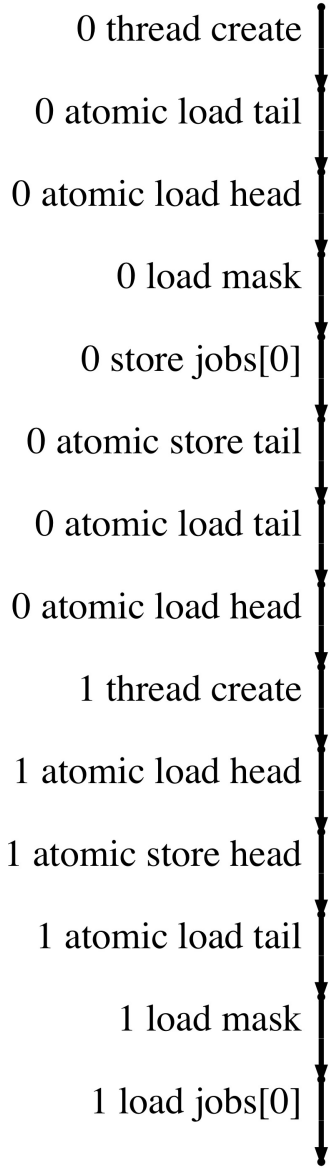
Bounded Search with Preemption Bound = 0



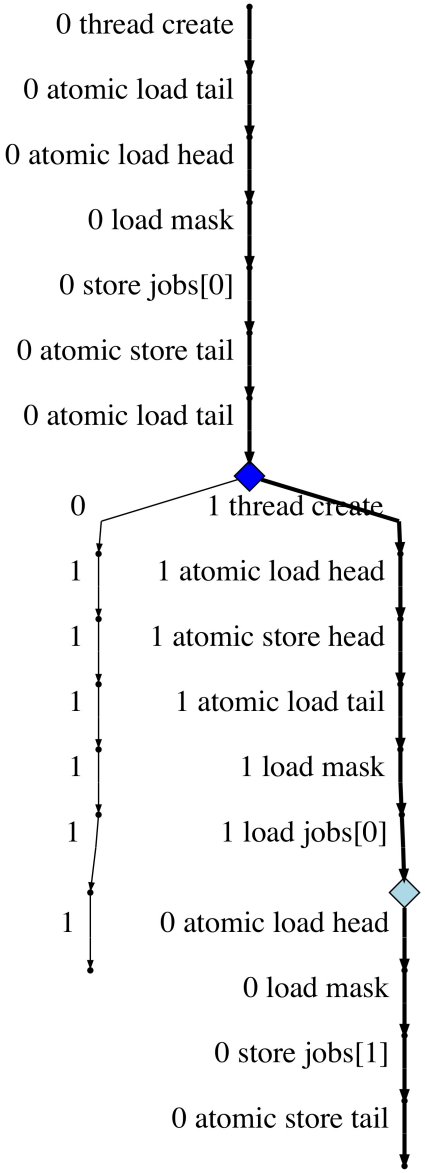
Bounded Search with Preemption Bound = 0



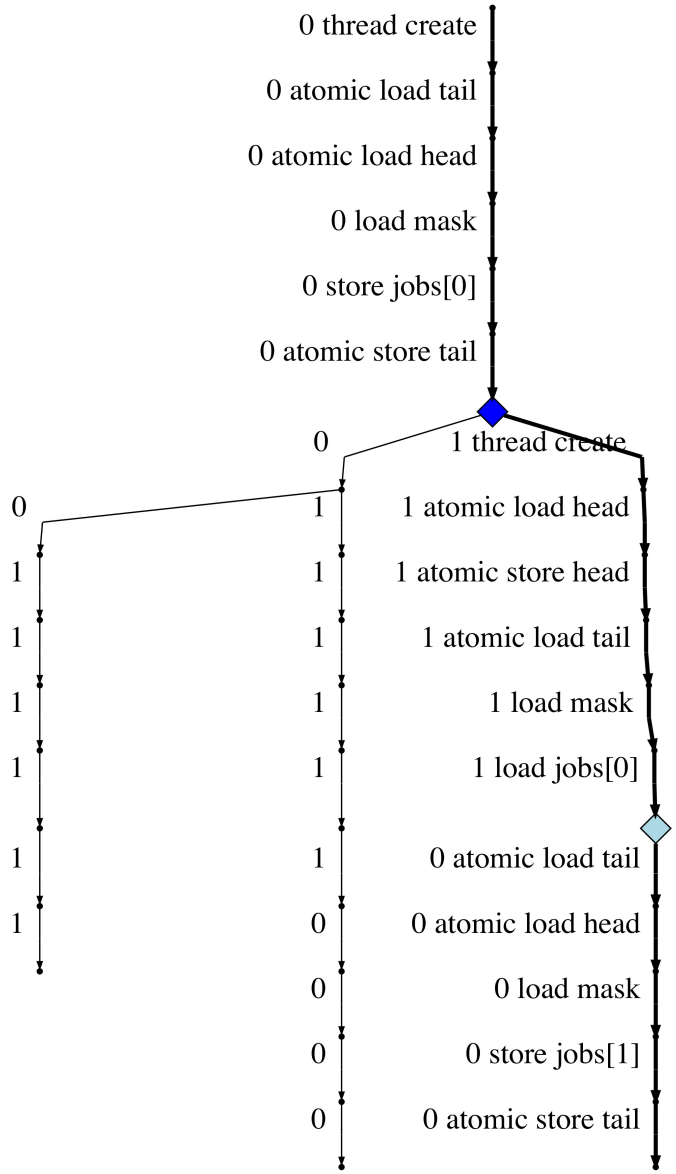
Bounded Search with Preemption Bound = 1



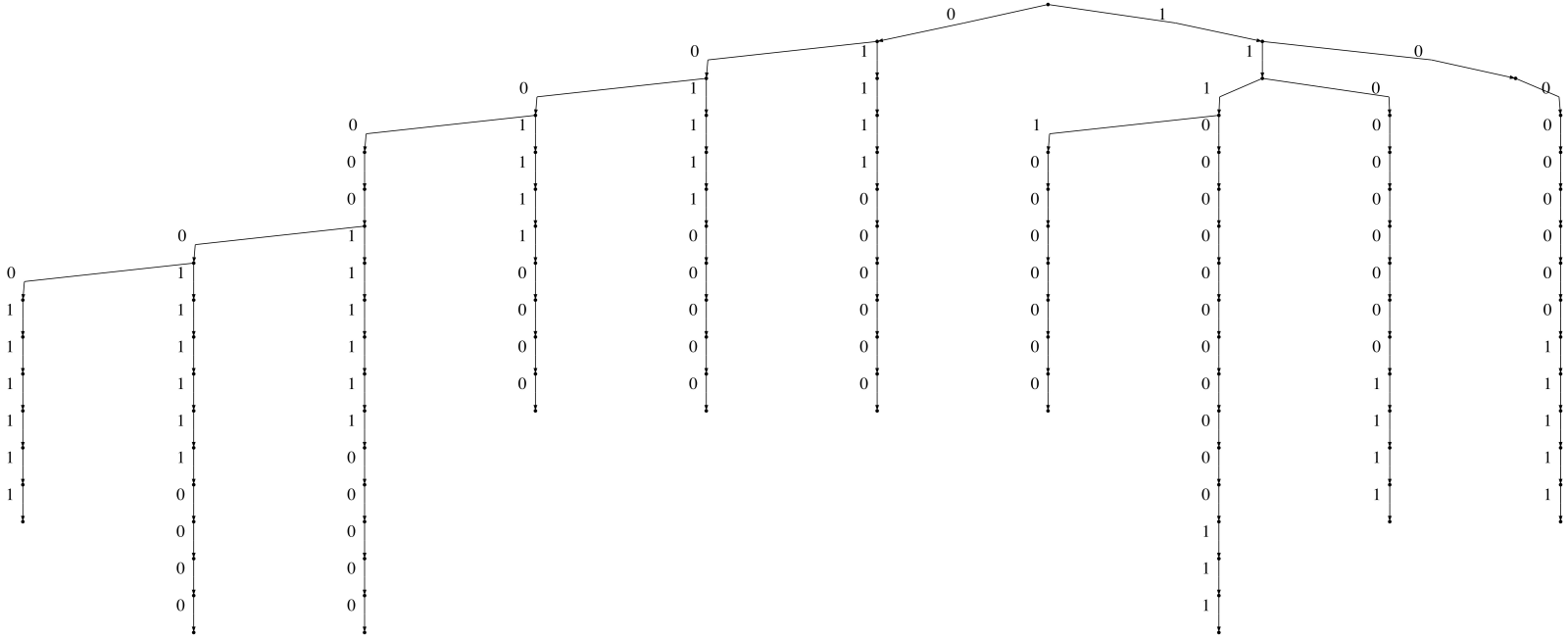
Bounded Search with Preemption Bound = 1



Bounded Search with Preemption Bound = 1



Bounded Search with Preemption Bound = 1



Bounded Search

Advantages:

- Very fast for low bounds
- Quantifiable coverage
- Incremental

Disadvantages:

- Incomplete
- For high bounds not better than depth first search

Partial Order Reduction

Different interleavings may yield *equivalent* executions: Ideally explore only a single one of those

Partial Order Reduction

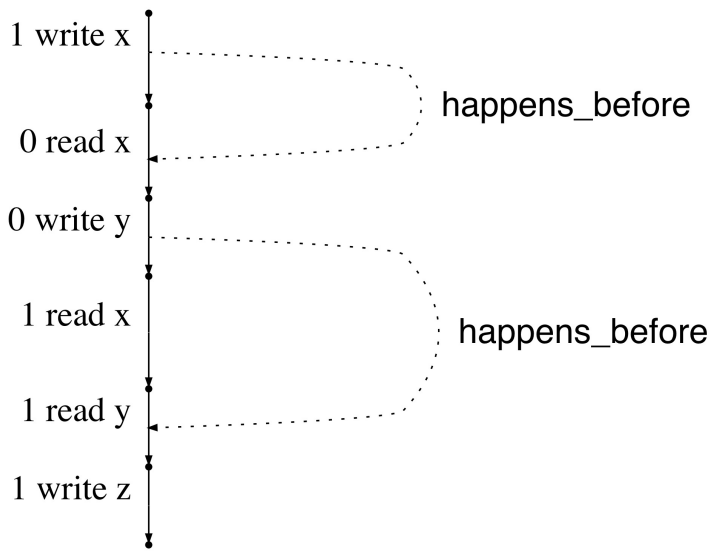
Different interleavings may yield *equivalent* executions: Ideally explore only a single one of those

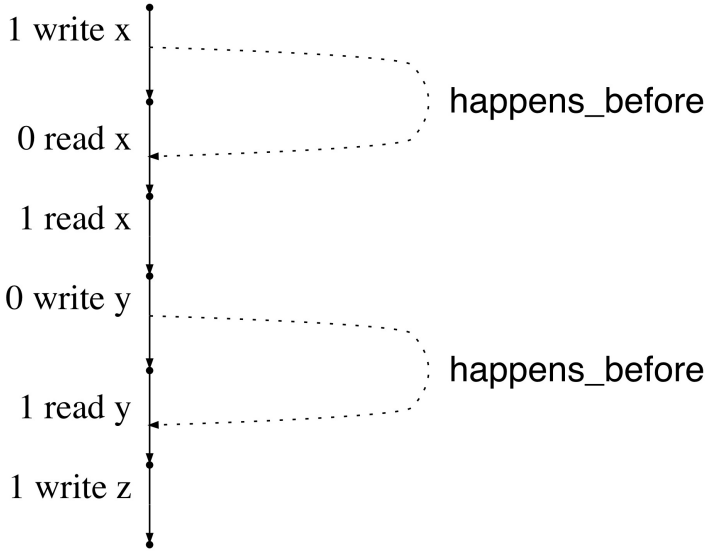
Dependence Relation

```
bool dependent(memory_instr1, memory_instr2)
{
   return same_thread(memory_instr1, memory_instr2) ||
          ( same_operand(memory_instr1, memory_instr2) &&
            memory_instr1.is_write() || memory_inst2.is_write() );
}
```

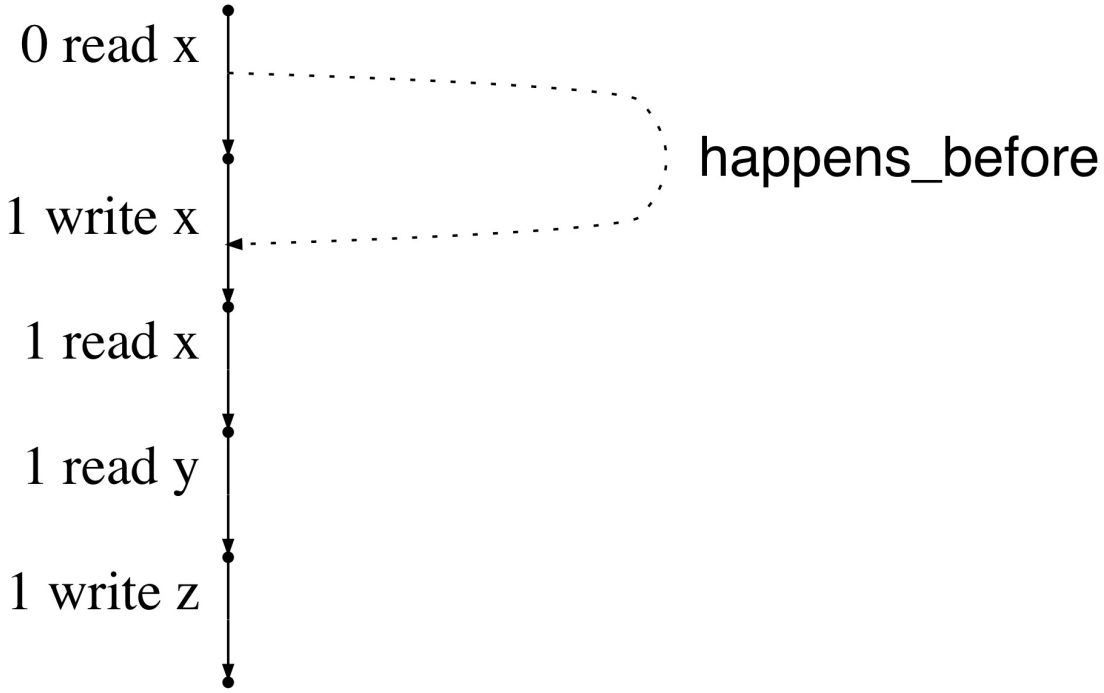
```
bool dependent(lock_instr1, lock_instr2)
{
   return same_thread(lock_instr1, lock_instr2) ||
          ( same_operand(lock_instr1, lock_instr2) &&
            lock_instr1.is_lock() && lock_inst2.is_lock() );
}
```

Partial Order Reduction

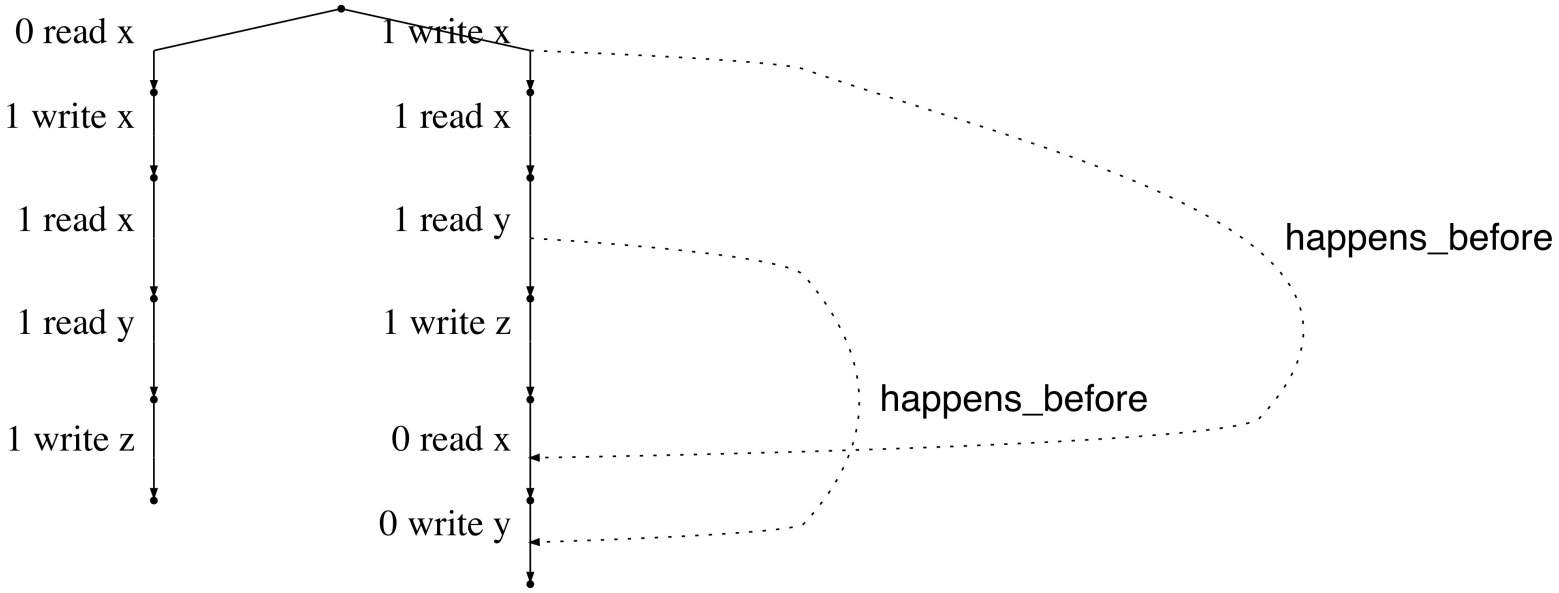




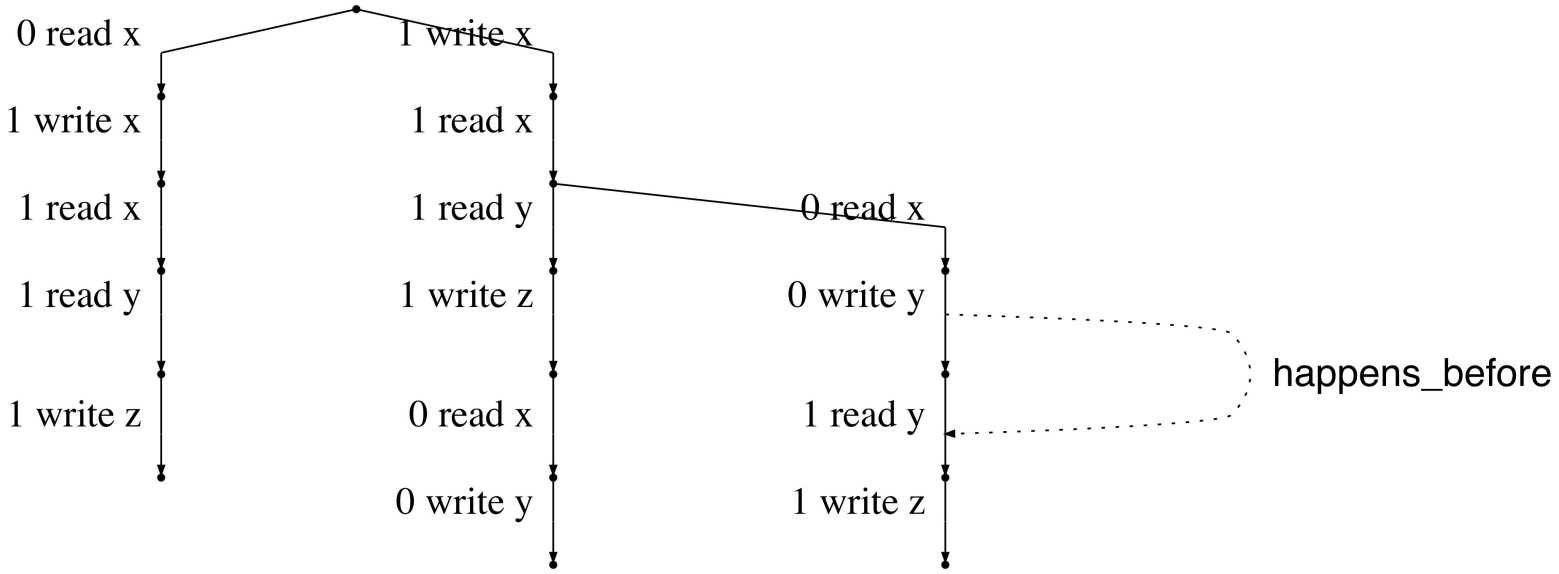
Partial Order Reduction



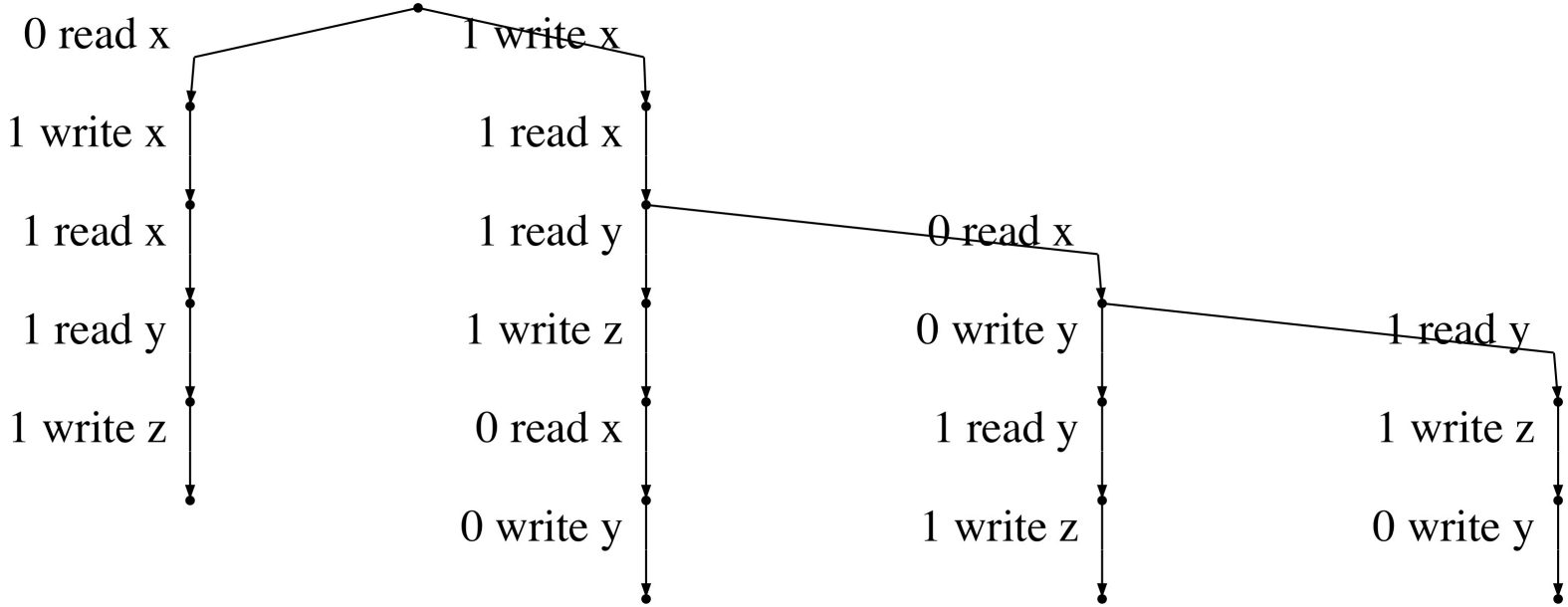
Partial Order Reduction



Partial Order Reduction



Partial Order Reduction



Partial Order Reduction

Advantages:

- Complete coverage

Disadvantages:

- Still infeasible for large state-spaces with many dependencies
- Not incremental

Thank you!

Links

Helgrind

http://valgrind.org/docs/manual/hg-manual.html

ThreadSanitizer:

https://github.com/google/sanitizers/wiki

My Project:

https://github.com/s-vde/record-replay

https://github.com/s-vde/state-space-explorer