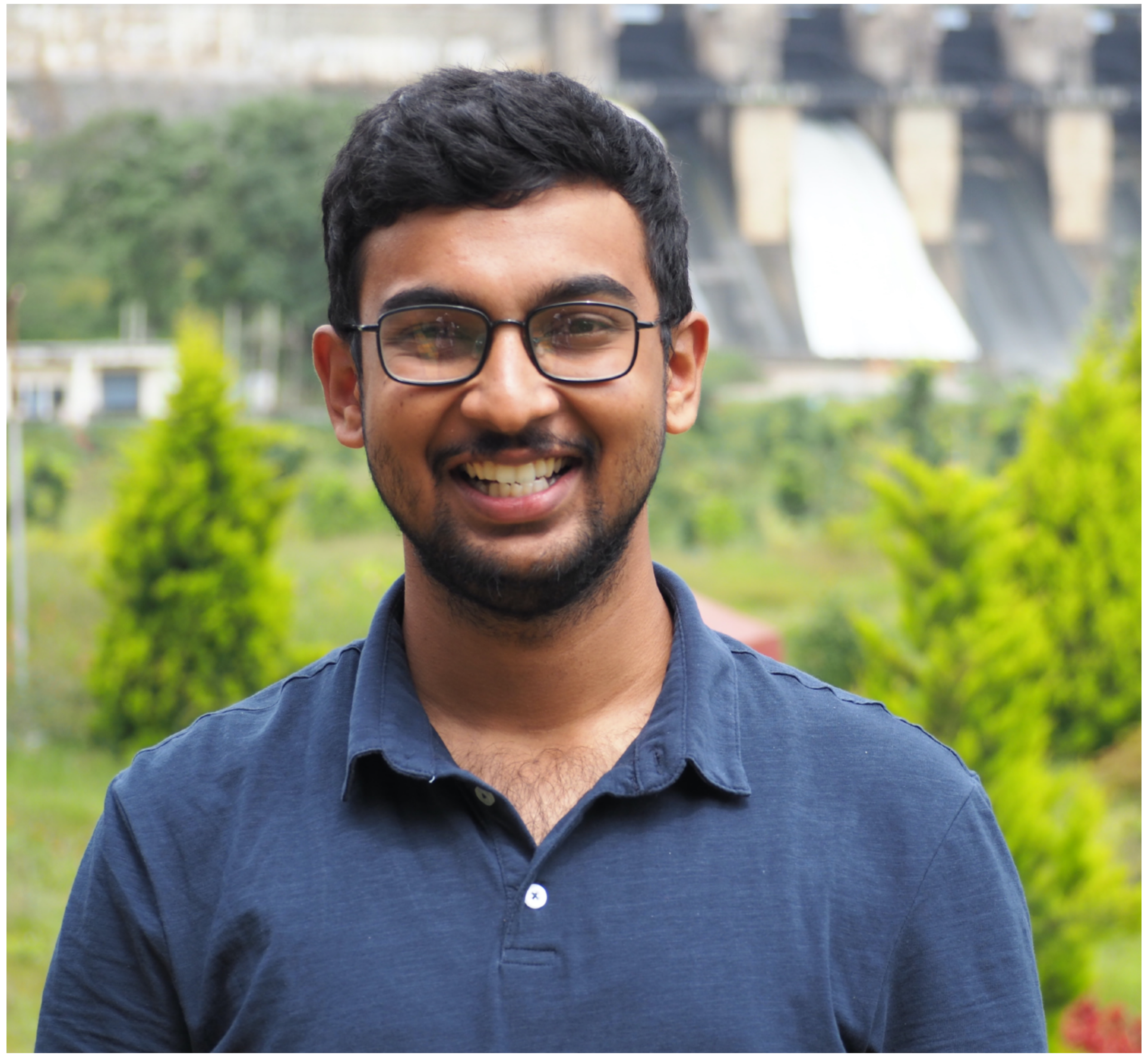
AI and Software Verification with JMC

Srinidhi Nagendra

Me

- I'm a PostDoc at MPI-SWS, Germany
- I completed PhD last December
  "Automated Testing of Distributed Protocol Implementations"
- Software Developer ~3 years



Outline

- Two ways of using AI
- The model checking problem.
- Efficient model checking - DPOR
- Tool - JMC
- Next steps

Using Agents

- Testing/checking the programs generated by Agents using JMC
- Write an Agent that annotates and runs JMC on the program generated.
- Give concrete bugs back to the agent to write better software

- Use Agents to make JMC better
- Write better strategies to test programs within JMC
- Guide using effectiveness to find bugs/coverage

What is Model Checking?

Model checking is a verification technique that explores all possible system states in a brute-force manner.

- Principles of Model Checking, MIT Press 2008

What's a *model*?

```
function factorial(n: int): (result: int)
requires n >= 0
    ensures result == if n == 0 then 1 else n * factorial(n - 1)
{
    if n == 0 then 1 else n * factorial(n - 1)
}
```

Spec/Model

Implementation

Dafny

```java
public static long factorial(int n) {
    long result = 1;
    for (int i = 2; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

Does it do it correctly?

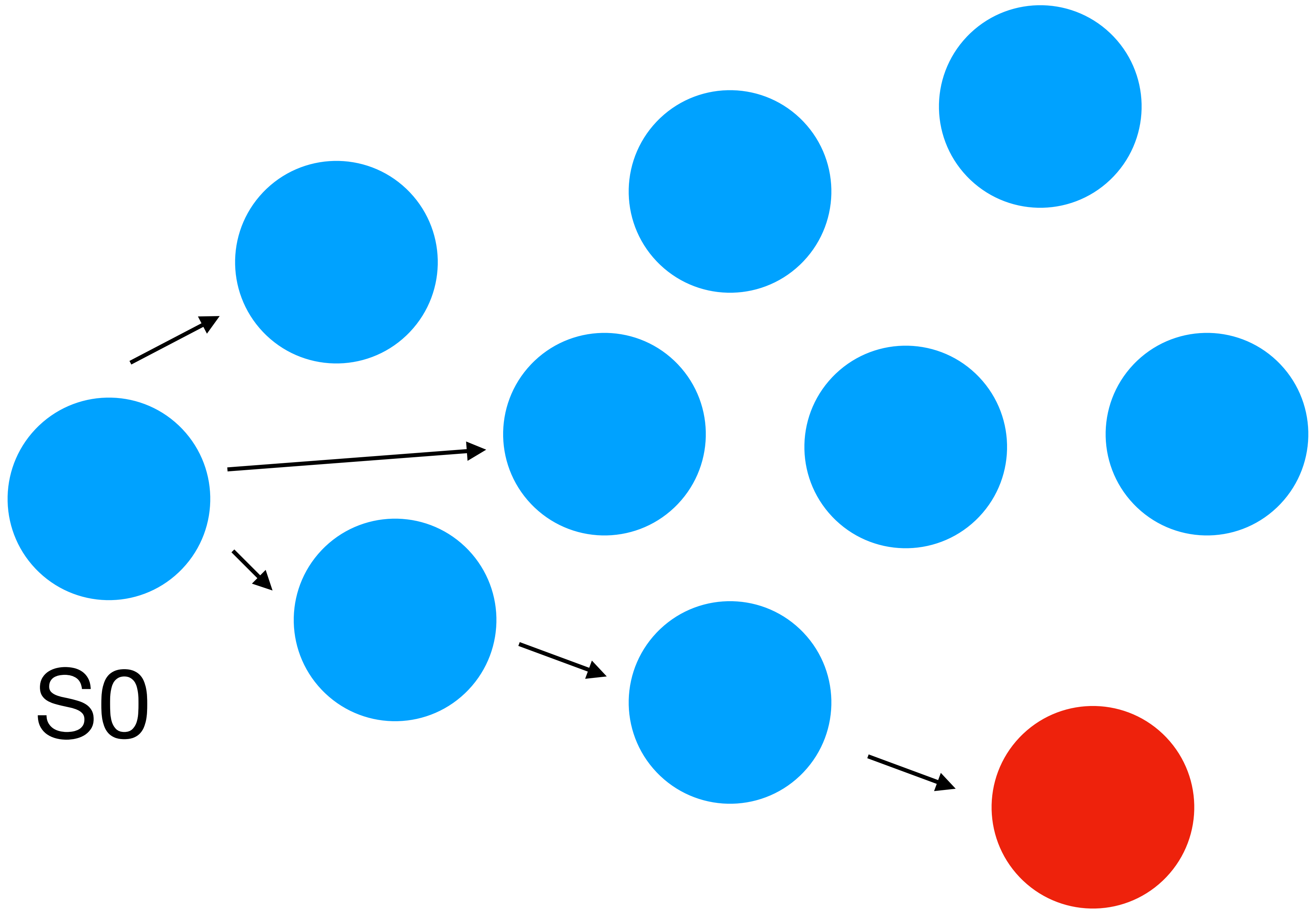
factorial(-1)?

```java
public static long factorial(int n) {
    if (n < 0) {
        throw new IllegalArgumentException("Negative numbers not allowed");
    }
    long result = 1;
    for (int i = 2; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

State

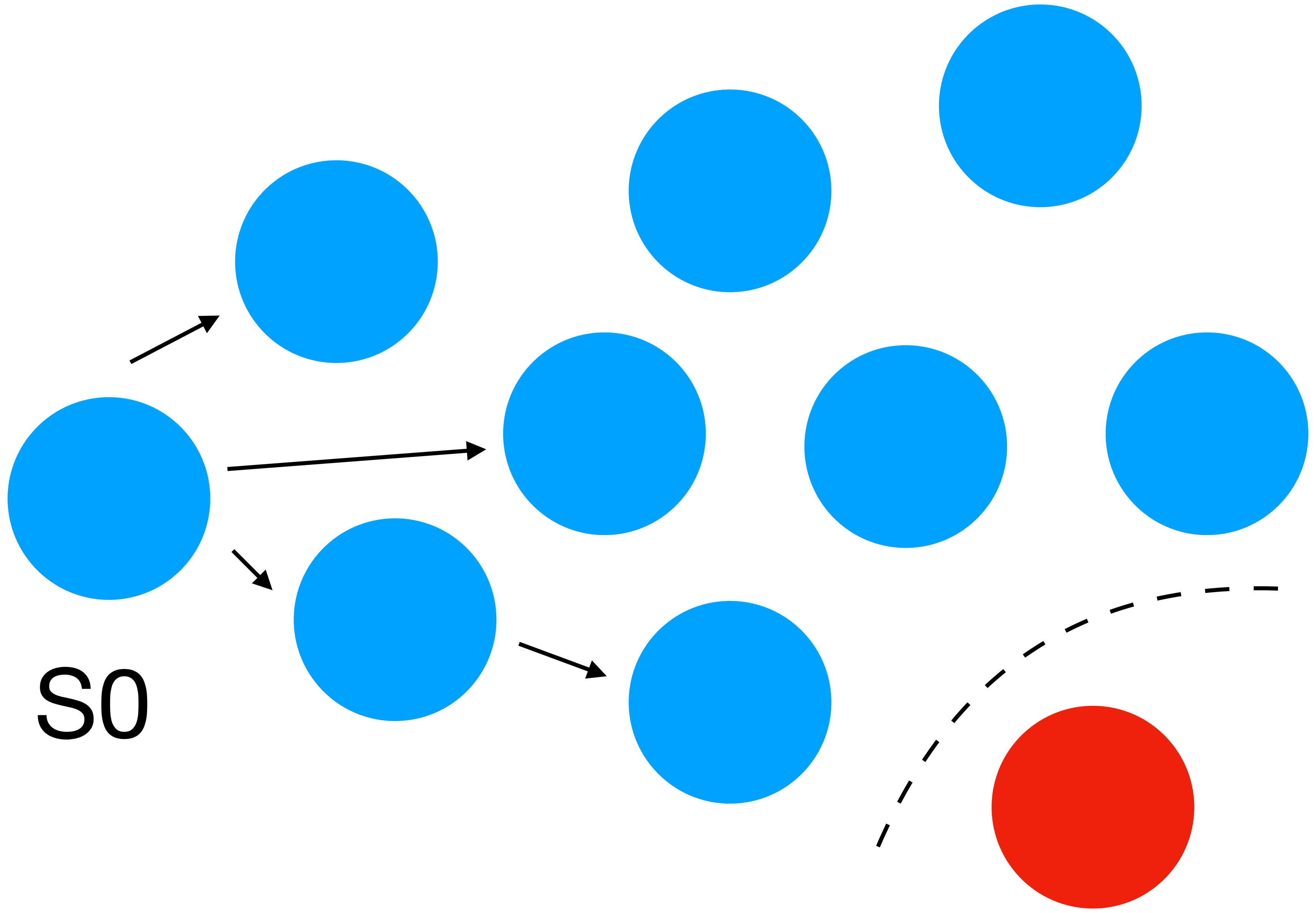
Is it still a model?

Model Checking



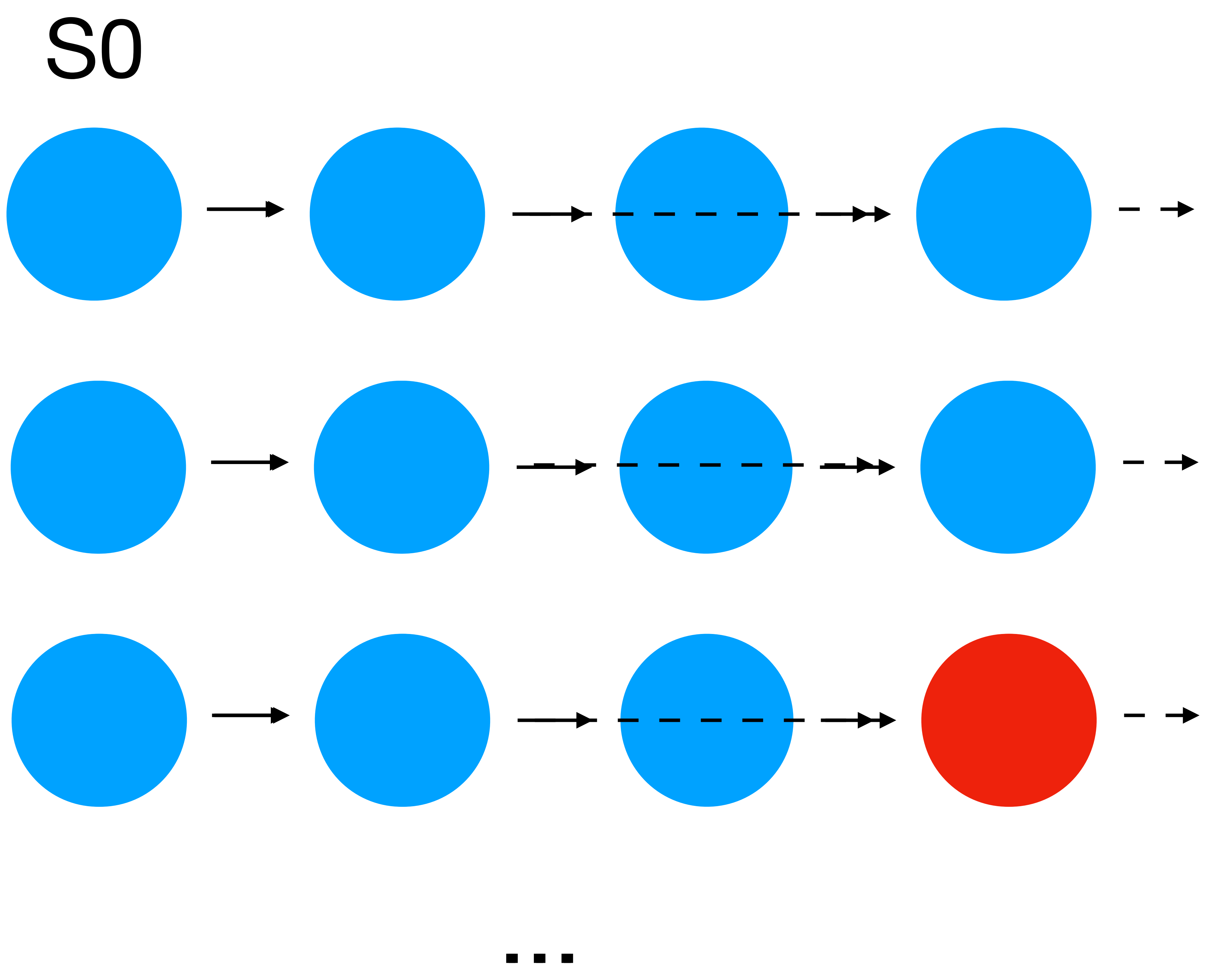
- Explore all states by enumerating them.
- E.g. Check that all values of state variables do not violate the property

Model Checking



- What about unreachable states?

Model Checking



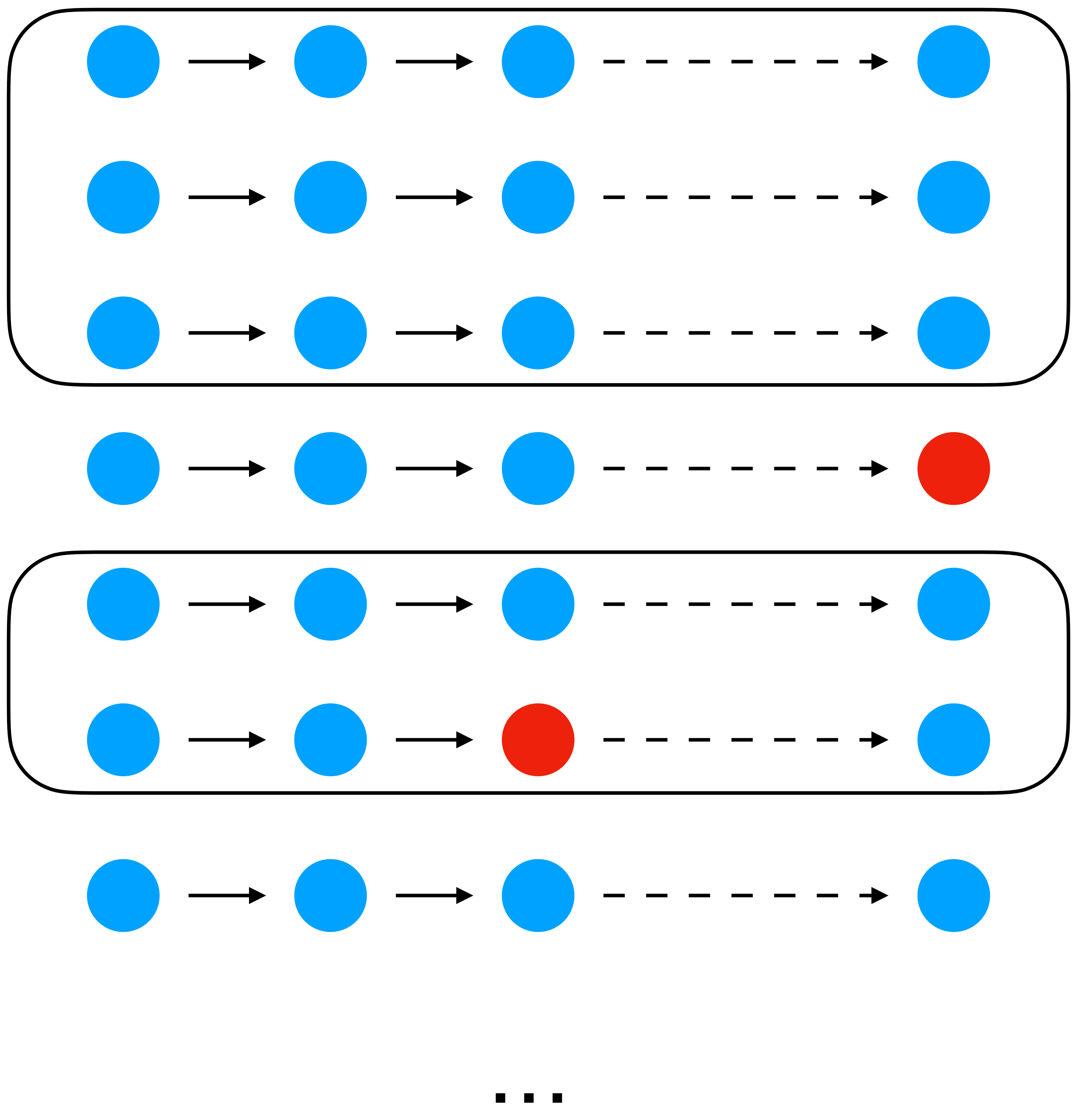
- Enumerate executions
- How long should each execution be?
    - Bounded model checking

Testing vs Model Checking

- Randomly sample executions.
- May/May not find bugs.
- Reproducibility is a challenge.

- Exhaustively enumerate all executions.
- Proof of bug or its absence.
- Easy to reproduce.

State explosion

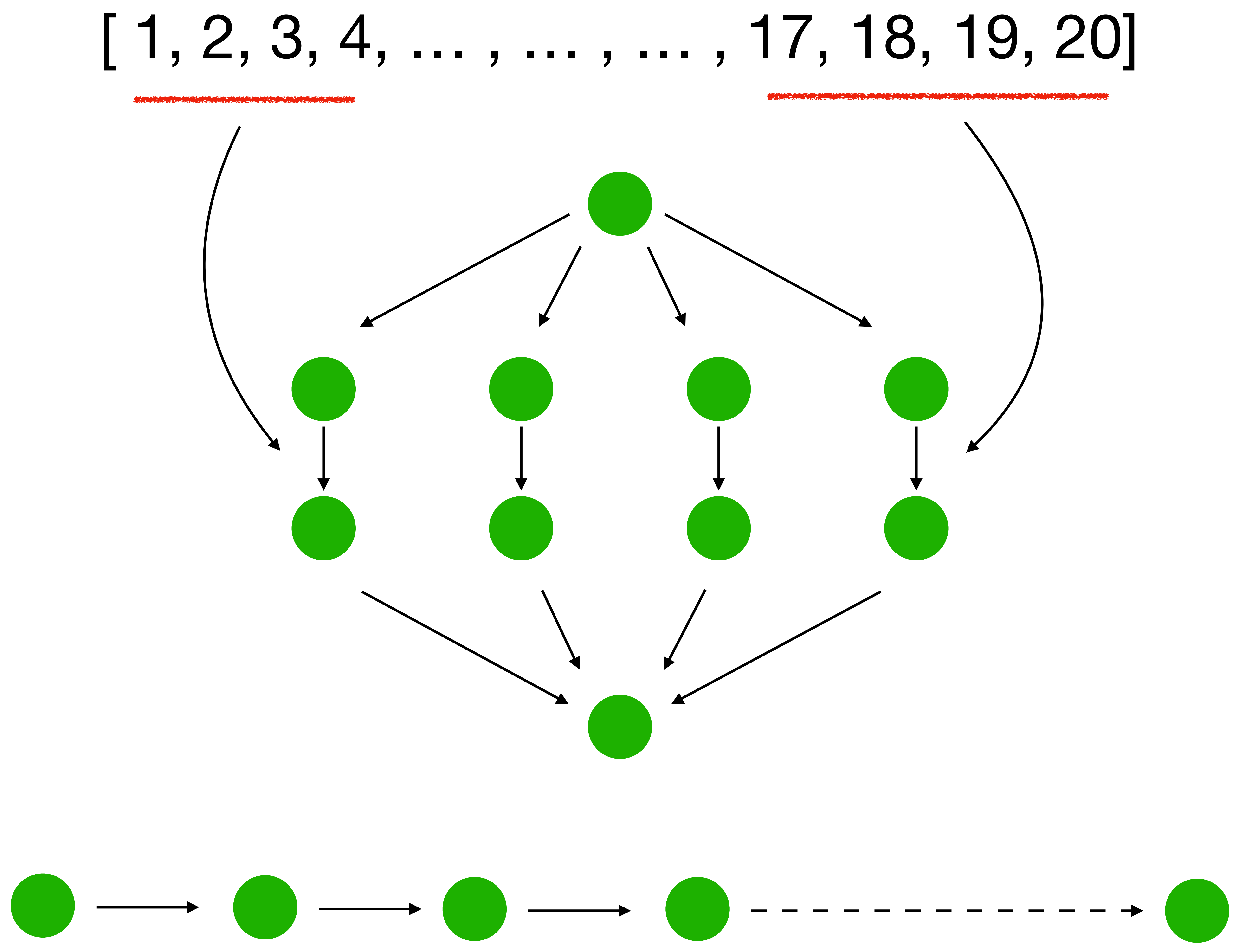


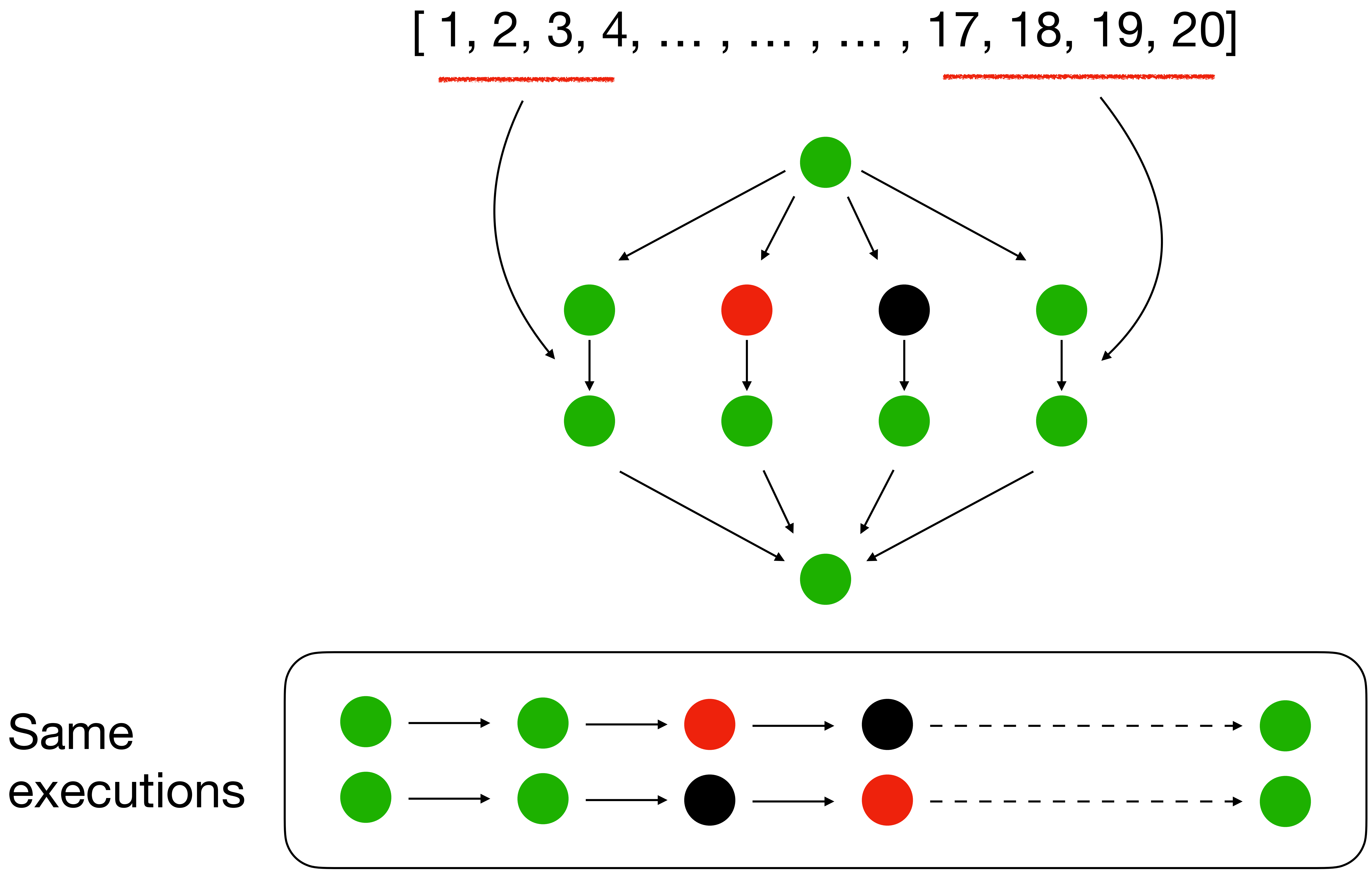
- Too many executions to explore.
    - Unrealistic to enumerate
- Testing becomes indistinguishable from model checking
- Solution: Partial order reduction

```java
public static BigInteger parallelFactorial(int n, int threadCount) {
    if (n < 0) throw new IllegalArgumentException("Factorial not defined for negative
    if (n == 0 || n == 1) return BigInteger.ONE;

    AtomicReference<BigInteger> result = new AtomicReference<>(BigInteger.ONE);
    ExecutorService executor = Executors.newFixedThreadPool(threadCount);

    ...

            // Update the shared result atomically
            result.getAndUpdate(current -> current.multiply(partial));

    ...

    return result.get();
}
```

Additional state

Faster implementation


[1, 2, 3, 4, … , … , … , 17, 18, 19, 20]


[1, 2, 3, 4, ... , ... , ... , 17, 18, 19, 20]
Same executions

Partial order reduction

- Merge executions that are equivalent under some relation.

- Dynamic: Do this by running the program.

    - (Static: Just analysing the code and not running it)

- Optimal: Explore each unique execution only once

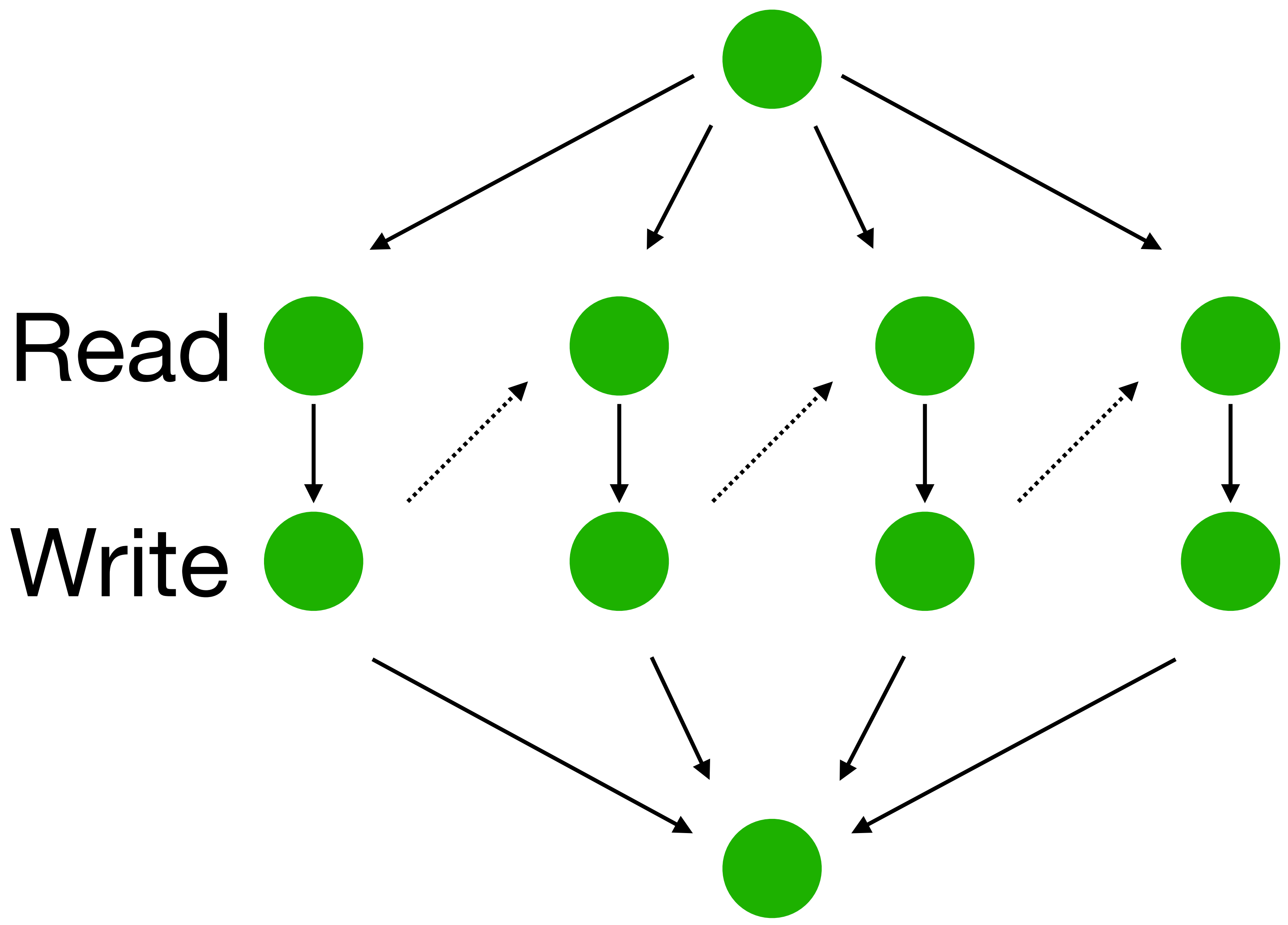
RESEARCH-ARTICLE | OPEN ACCESS |

Truly stateless, optimal dynamic partial order reduction

- POPL 2022

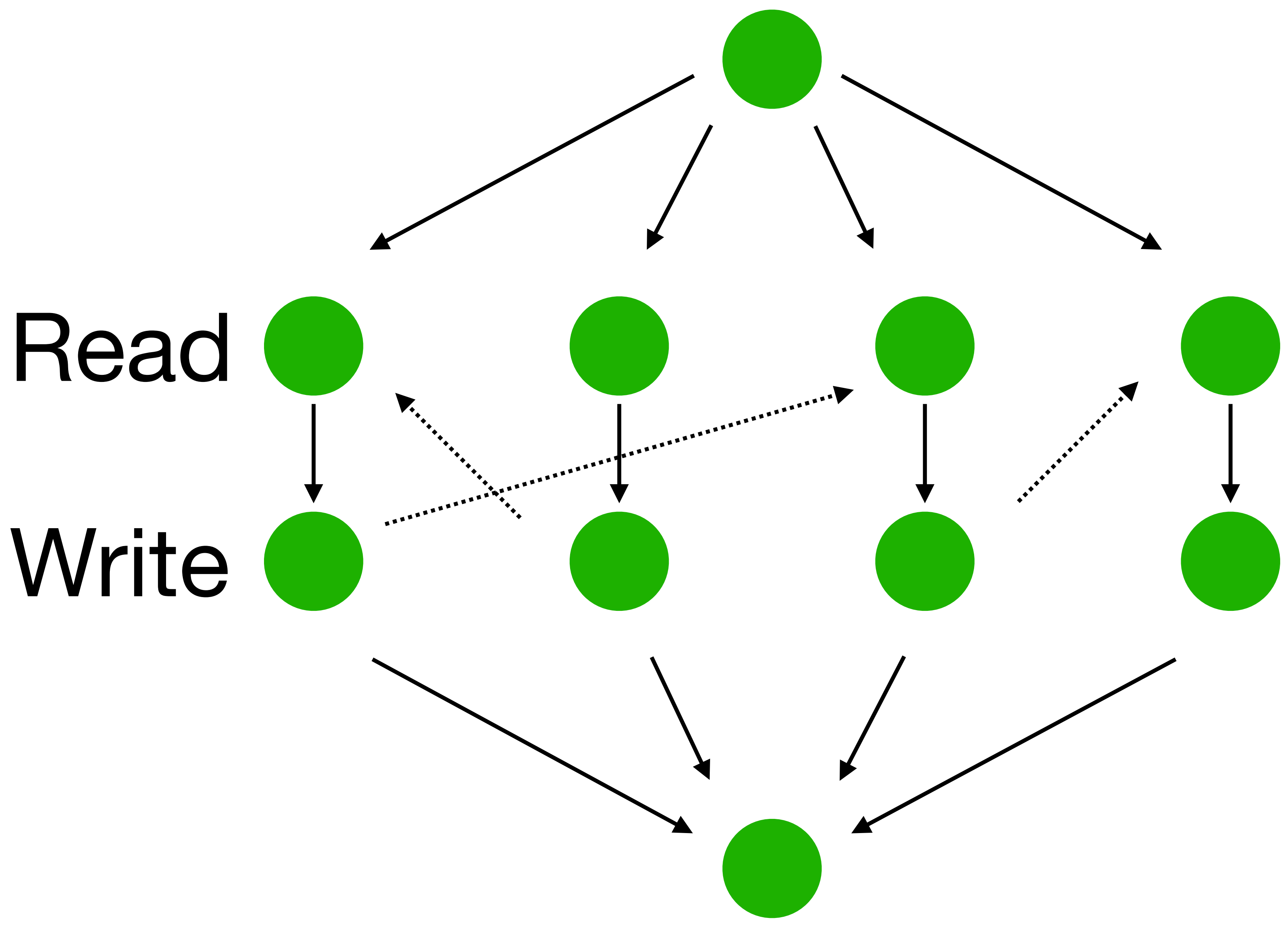
Counters

```java
public class Counter {
    private int count;

    public Counter() {
        this.count = 0;
    }

    public int get() {
        return count;
    }

    public void incr() {
        count++;
    }
}
```



Counters

```java
public class Counter {
    private int count;

    public Counter() {
        this.count = 0;
    }

    public int get() {
        return count;
    }

    public void incr() {
        count++;
    }
}
```



In general. *n* threads = *n!* Executions

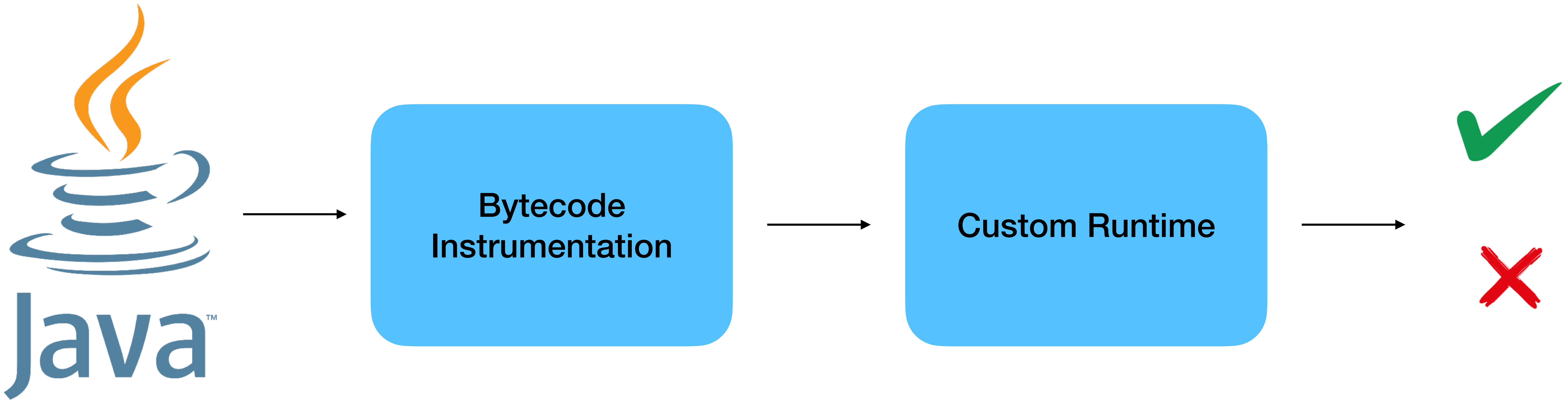
Demo

JMC working

```java
public class CounterTest {

    @JmcCheck
    @JmcCheckConfiguration(numIterations = 10)
    public void testRandomCounter() {
        ParametricCounter counter = new ParametricCounter(2);
        counter.run();
        assert counter.getCounterValue() == 2;
    }

    @JmcCheck
    @JmcCheckConfiguration(strategy = "trust", numIterations = 200)
    @JmcExpectExecutions(120)
    public void testTrustCounter() {
        ParametricCounter counter = new ParametricCounter(5);
        counter.run();
        assert counter.getCounterValue() == 5;
    }
}
```

- Write tests
- Annotate them
- Run the tests

JMC internals



Bytecode instrumentation

- Add stubs for every read and write events
  - GETFIELD and PUTFIELD bytecodes
- Add stubs for thread creation and deletion
  - This is a nightmare. Too many ways to create concurrency

Bytecode instrumentation

```java
public int get() {
    return count;
}
```

→

```java
public int get() {
    JmcRuntimeUtils.readEventWithoutYield(
        this, "org/mpi_sws/jmc/test/Counter", "count", "I");
    int var10000 = this.count;
    JmcRuntime.yield();
    return var10000;
}
```

Bytecode Instrumentation

Java Agent + ASM (asm.ow2.io)

- Java allows an agent to run with the JVM
- Agent code runs before every class is loaded

- ASM allows changing the class byte code
- Our philosophy: Do as little instrumentation as necessary

Threads

```java
public class CounterThread extends Thread {
    ...

    @Override
    public void run() {

    }
}
```

→

```java
public class CounterThread extends JmcThread {
    ...

    @Override
    public void run1() {

    }
}
```

- Why? We need to control every thread.
- Know when it starts and finishes and prevent deadlocks

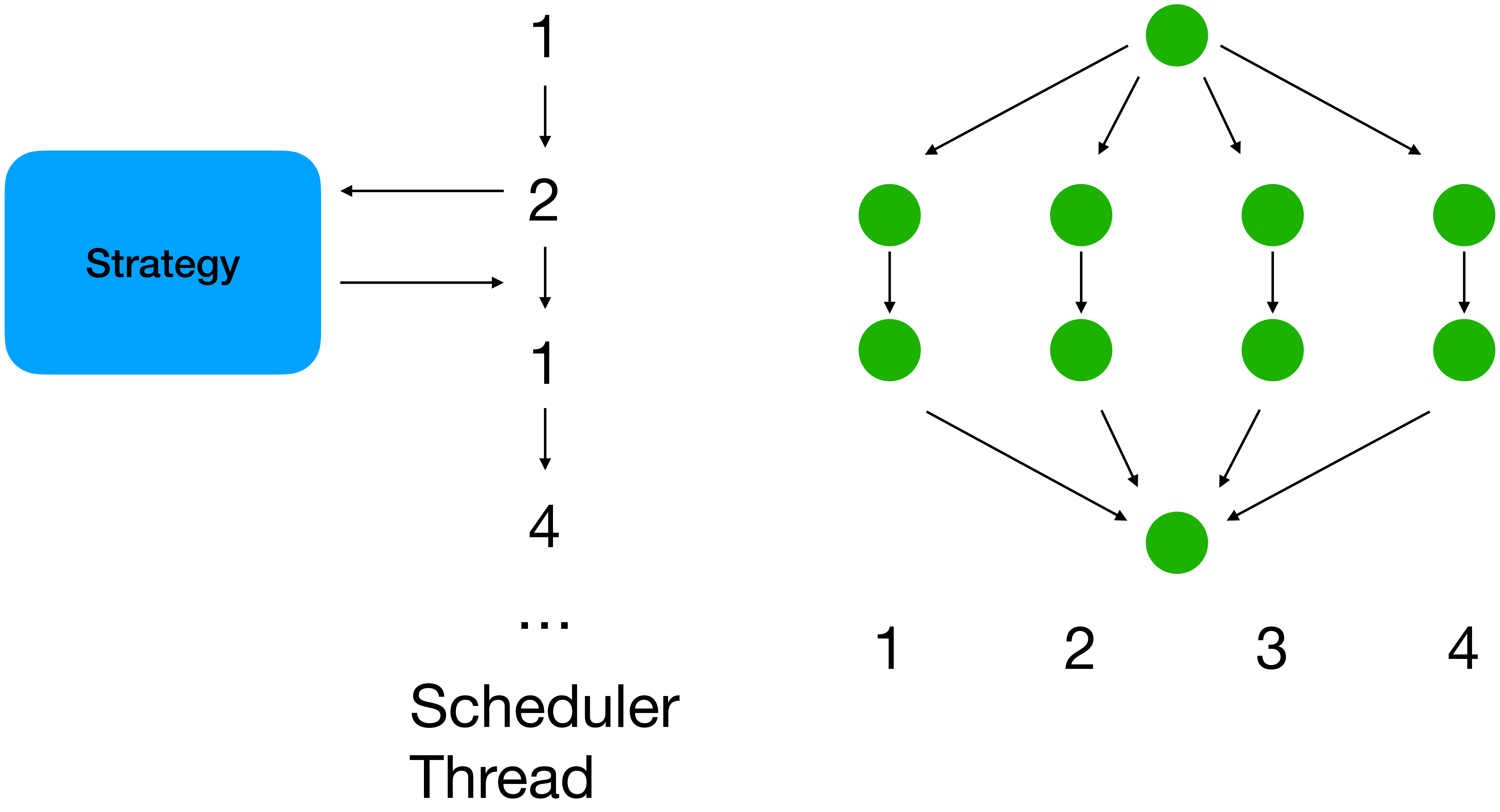
Thread Challenges

- Thread, Runnable
- Future, FutureTask, CompletableFuture
- ExecutorService
- ThreadPool
- Virtual Threads?

JMC Runtime

- Imposes a co-operative multi threading environment
- Each thread *yields* control when an event occurs
- Uses a Strategy to decide which thread to *resume*

Runtime working



Basic Strategy

- Cannot *resume* threads randomly. Leads to deadlock.
- E.g. Allowing a thread that wants a lock that is already acquired
- We implement a basic event tracker that marks threads as enabled/disabled
- Currently supports *synchronized* and *ReentrantLock.*
    - *Need to extend support*

Strategy class

JMC Summary

- A testing framework that verifies your code.
- Works by instrumenting byte code and running under a custom runtime
- Try it and let us know
  - Disclaimer - under development

What’s next?

Other strategies

- PCT - A Randomized Scheduler with Probabilistic Guarantees of Finding Bugs - ASPLOS ’10
- PCTCP - Randomized Testing of Distributed Systems with Probabilistic Guarantees OOPSLA ‘18
- (RA)POS - Effective random testing of concurrent programs ASE ’07
- SURW - Selectively Uniform Concurrency Testing ASPLOS ‘25
- QL (hard) - Learning-based controlled concurrency testing OOPSLA ‘20

Build an agent to write a strategy for you.

Evaluate based on how many executions/bugs it covers

Questions?