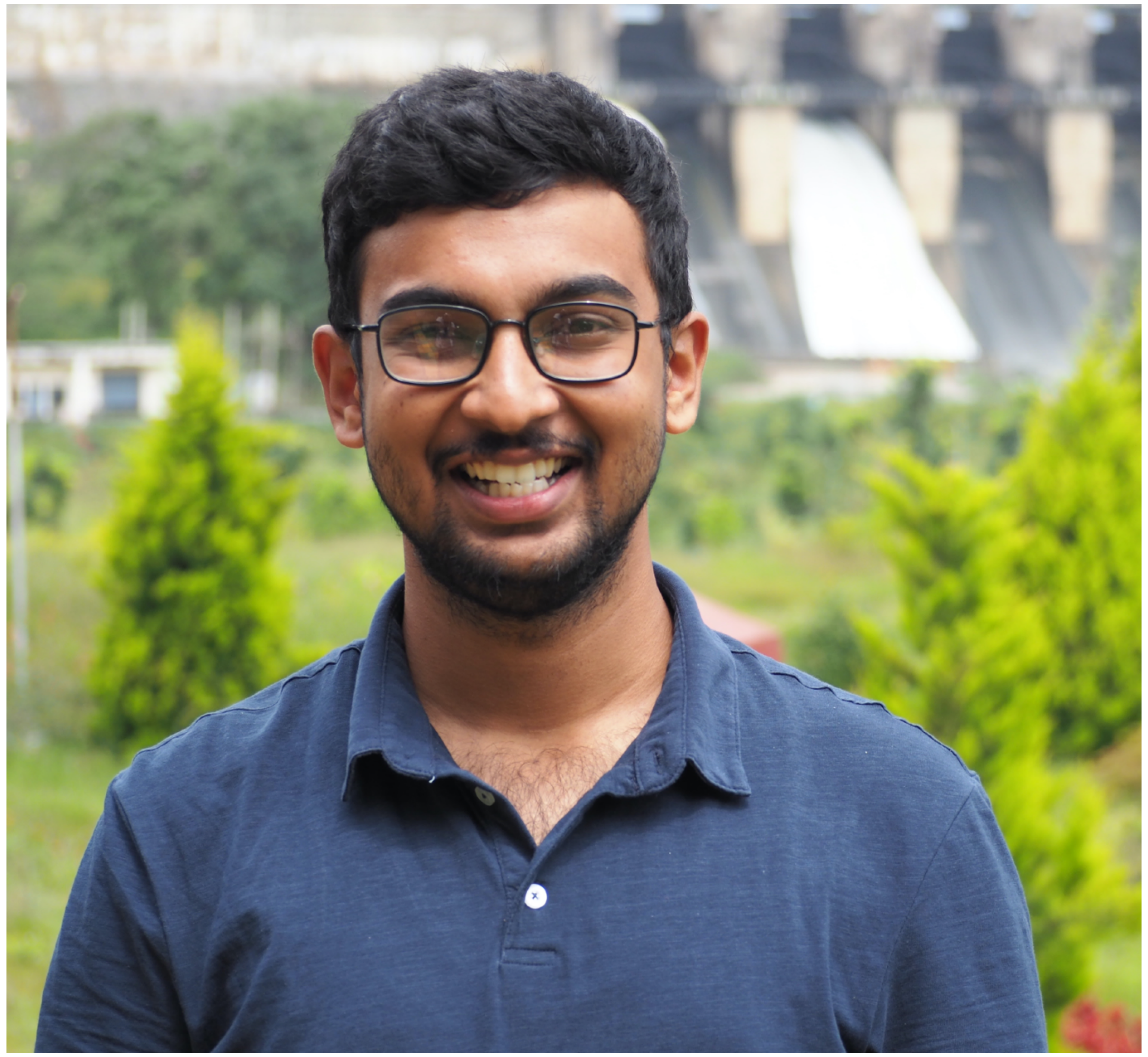
Model guided fuzzing of distributed systems

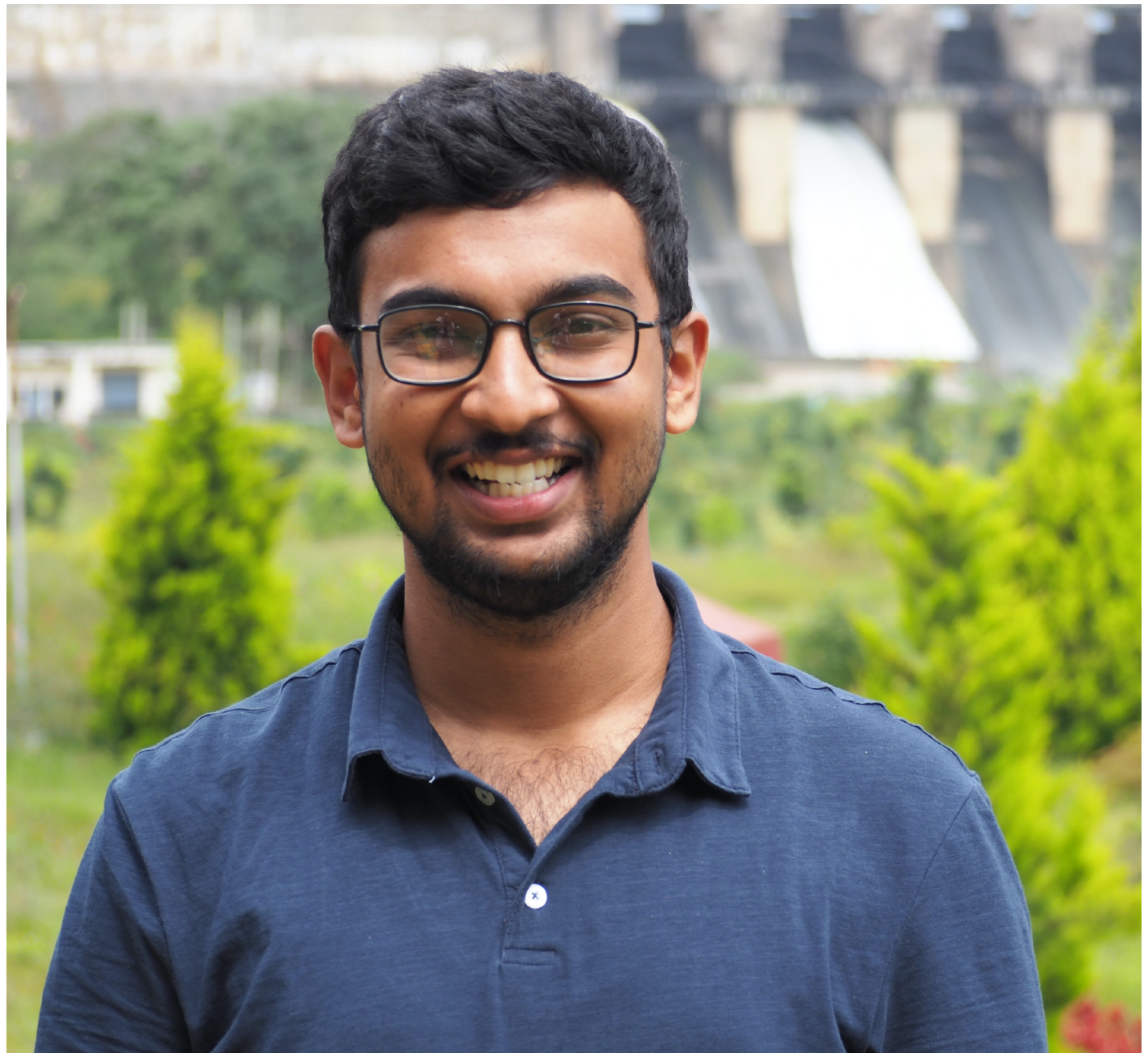
Ege Berkay, Burcu Özkan, Rupak Majumdar, **Srinidhi Nagendra**

Me



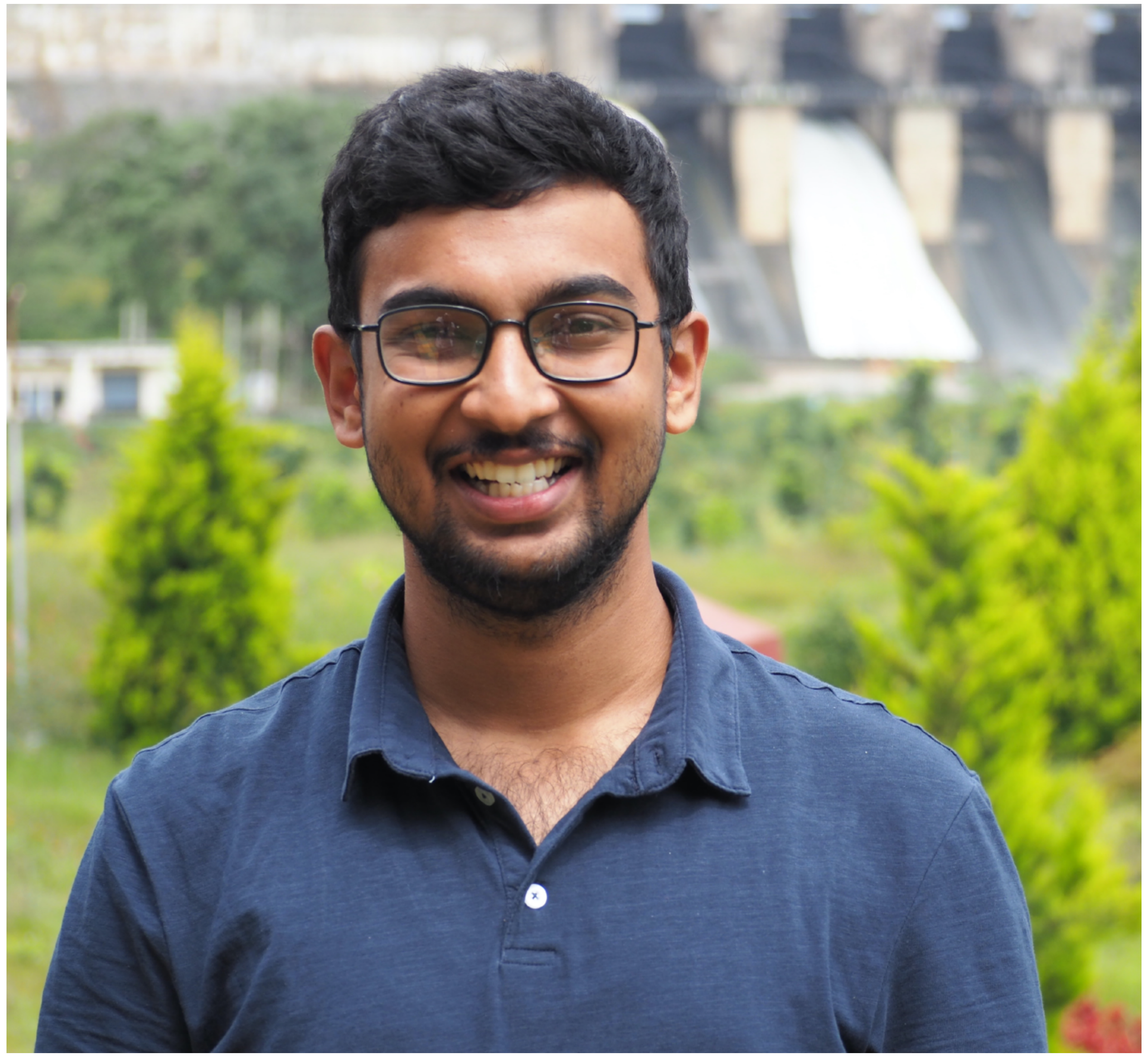
Me

- I’m a PostDoc at MPI-SWS



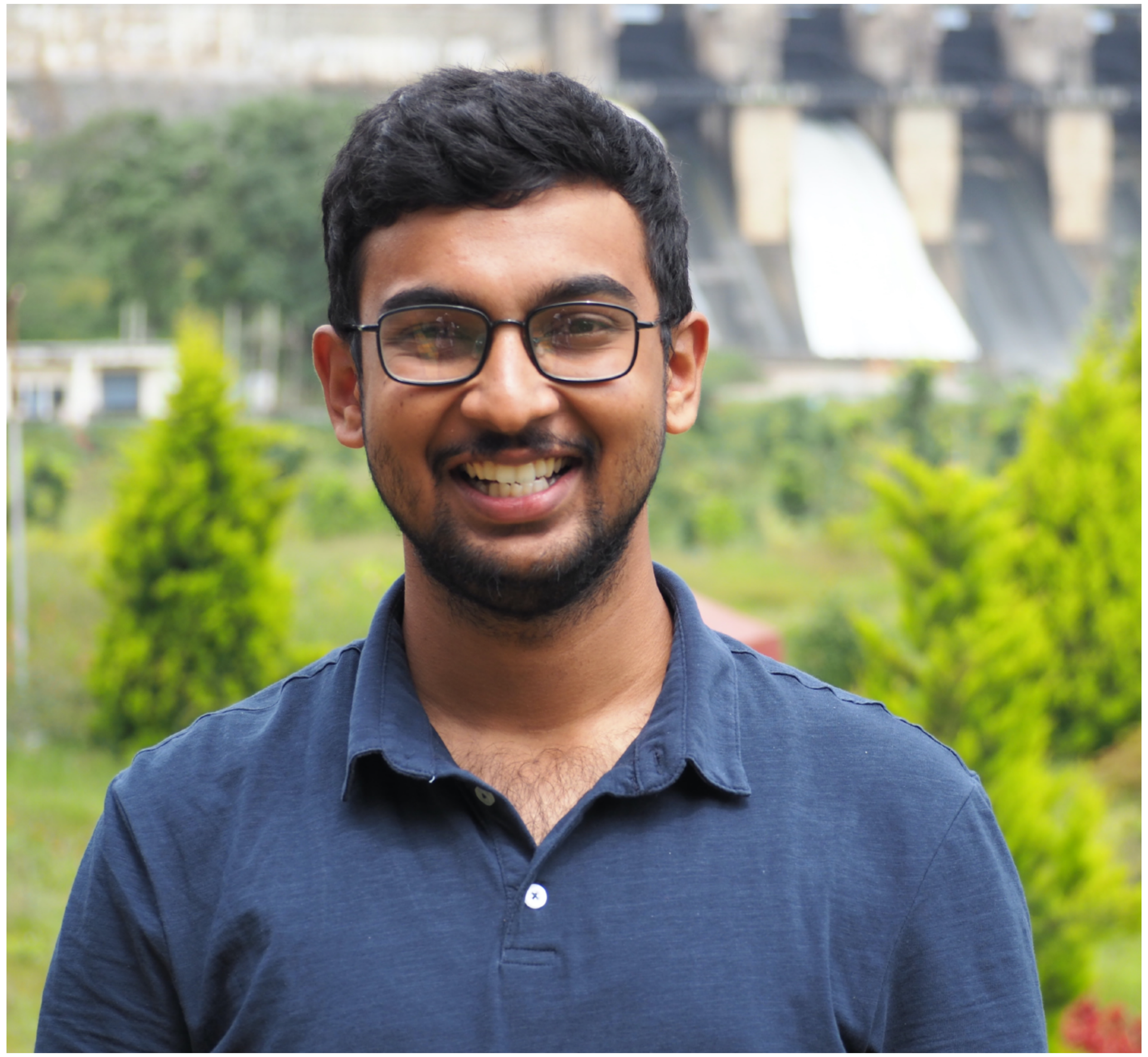
Me

- I'm a PostDoc at MPI-SWS
- I completed PhD last December
  "**Automated Testing of Distributed Protocol Implementations**"
  - Unit tests for Distributed Systems
  - Reinforcement learning guided exploration



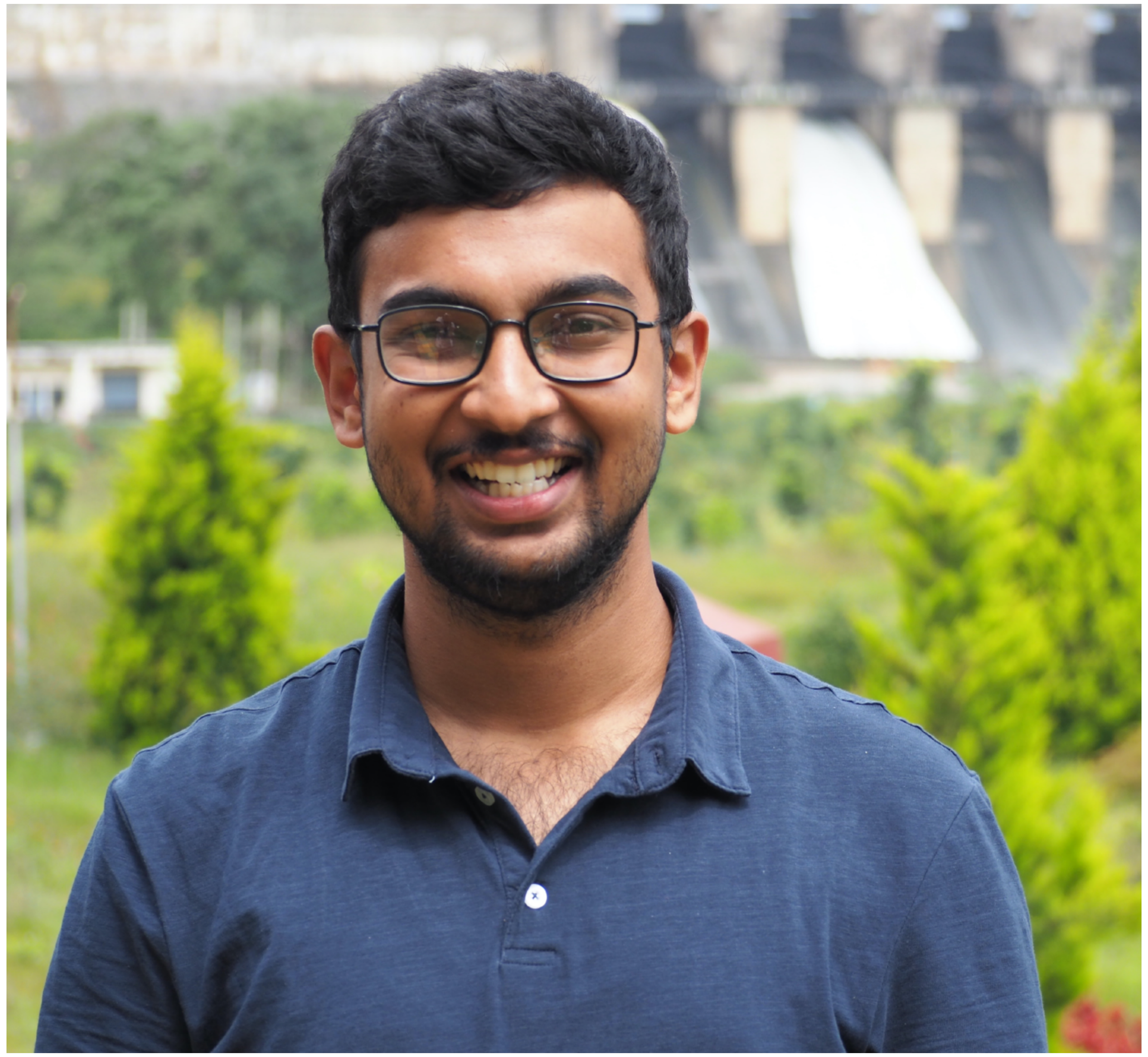
Me

- I’m a PostDoc at MPI-SWS
- I completed PhD last December
  “**Automated Testing of Distributed Protocol Implementations**”
  - Unit tests for Distributed Systems
  - Reinforcement learning guided exploration
- I discovered TLA+ in 2018 and have been an enthusiast since.



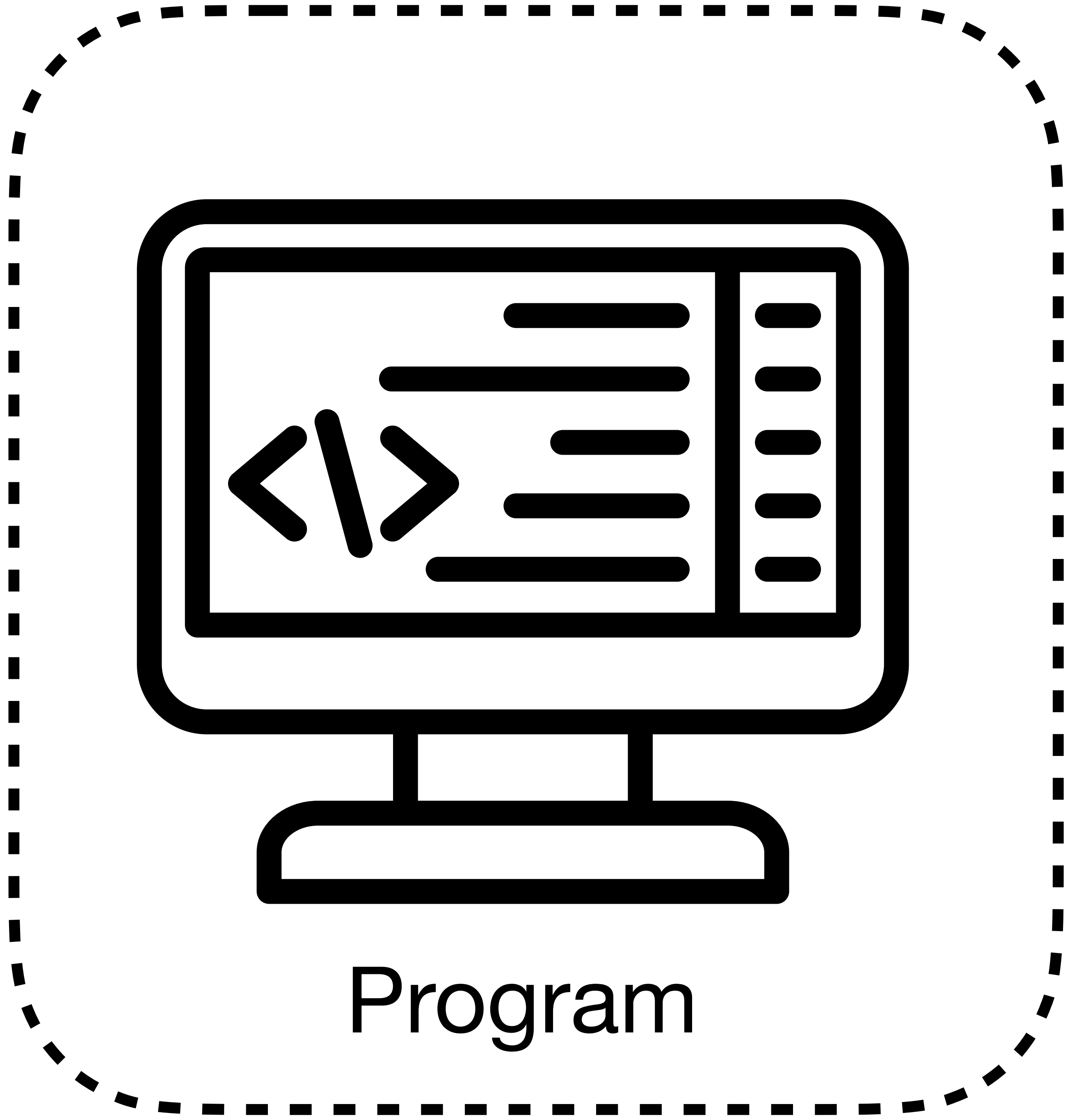
Me

- I’m a PostDoc at MPI-SWS
- I completed PhD last December
  “**Automated Testing of Distributed Protocol Implementations**”
  - Unit tests for Distributed Systems
  - Reinforcement learning guided exploration
- I discovered TLA+ in 2018 and have been an enthusiast since.
- I am on the job market looking for my next adventure!



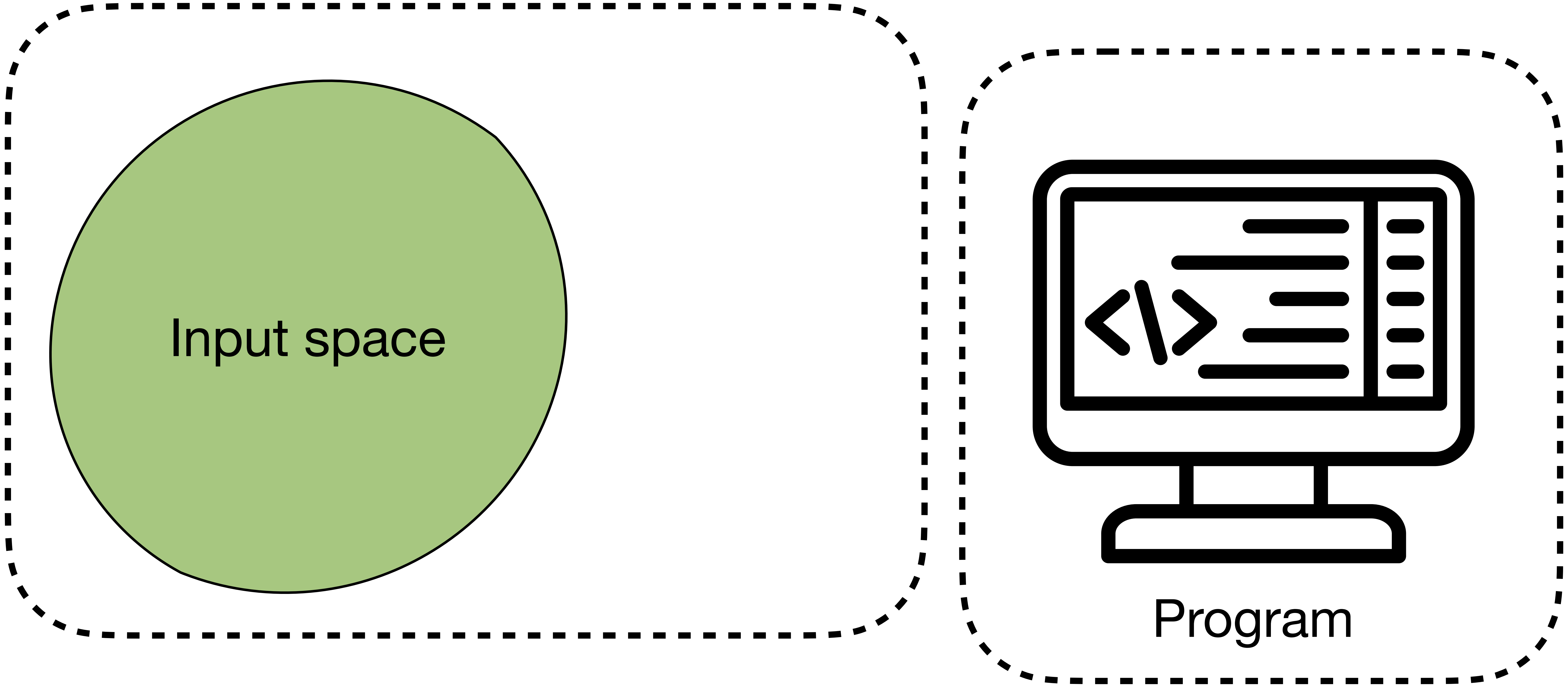
Traditional software testing

Traditional software testing



Program

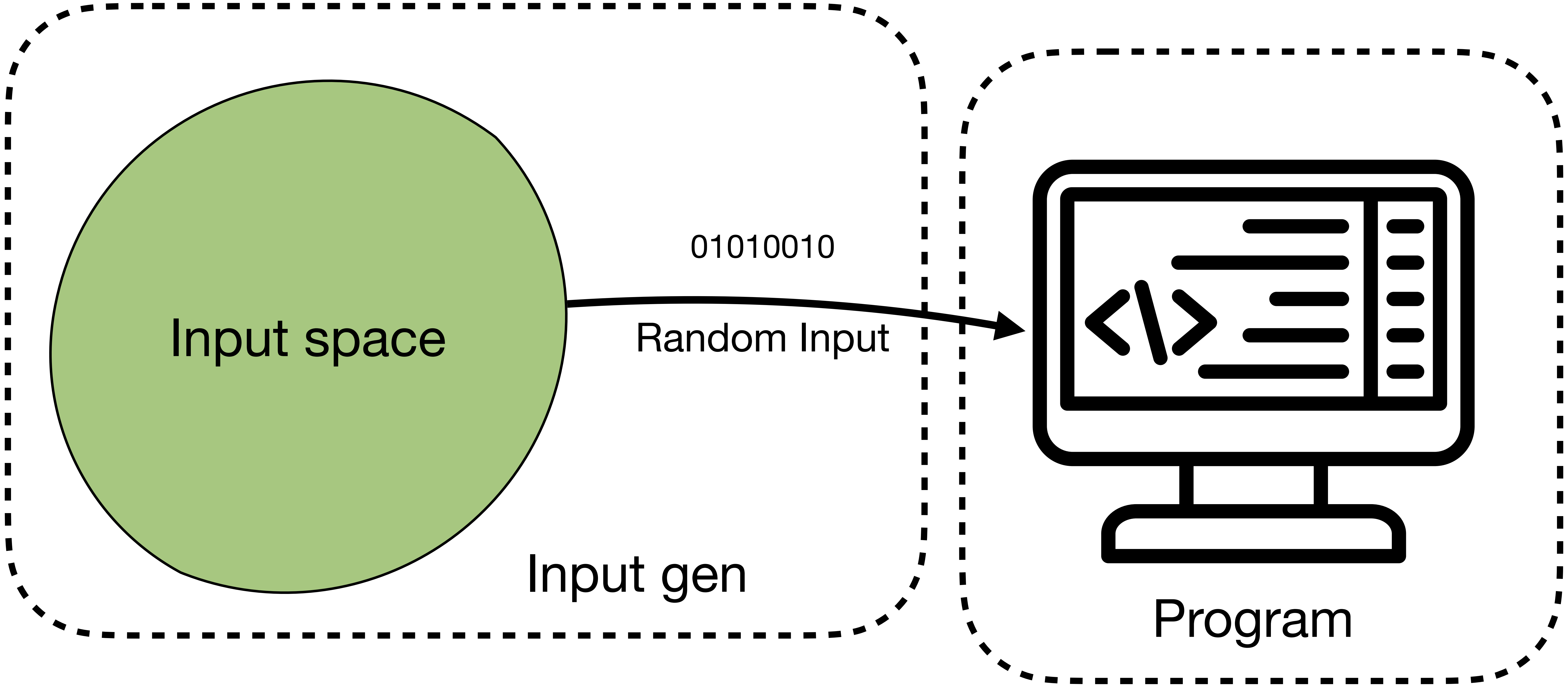
Traditional software testing



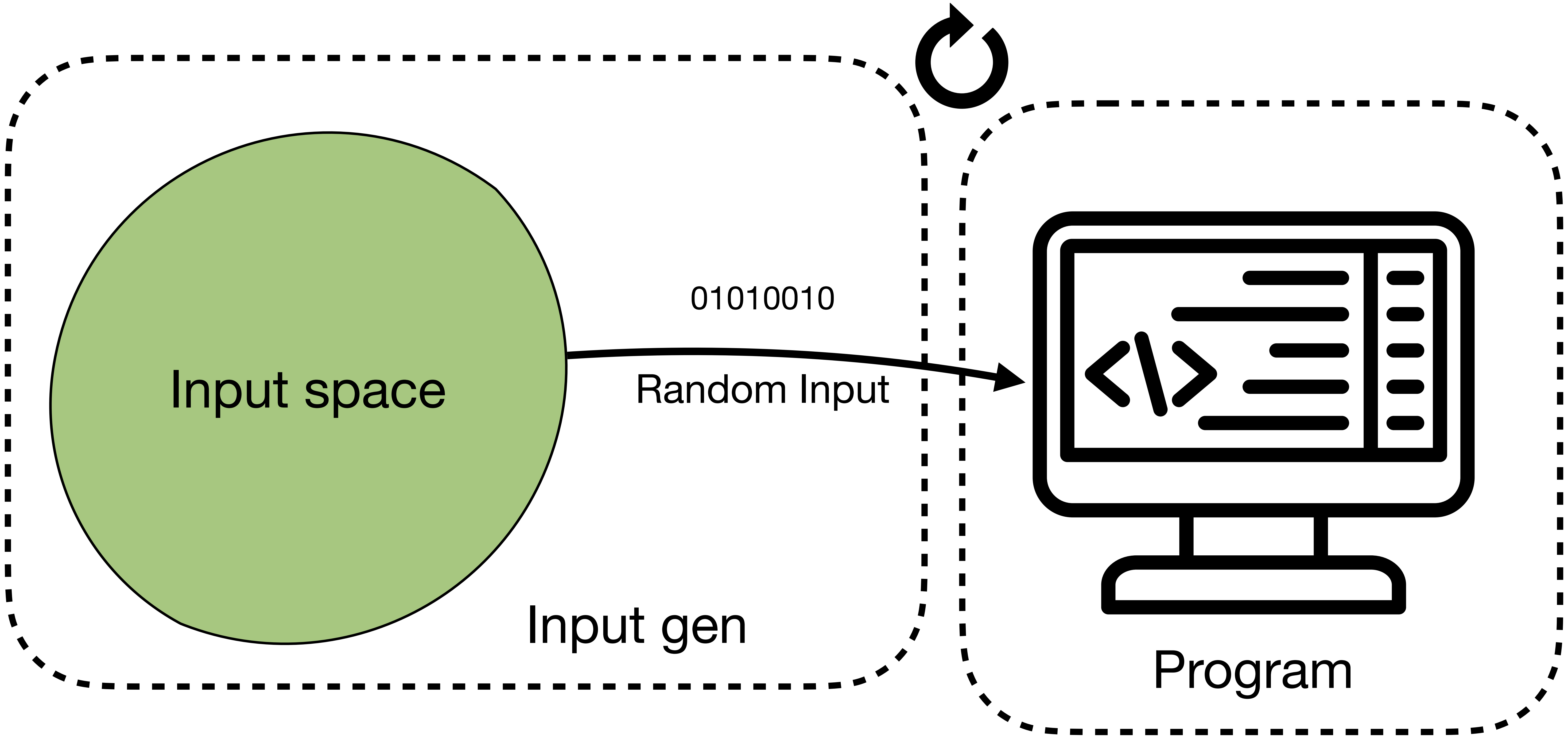
Input space

Program

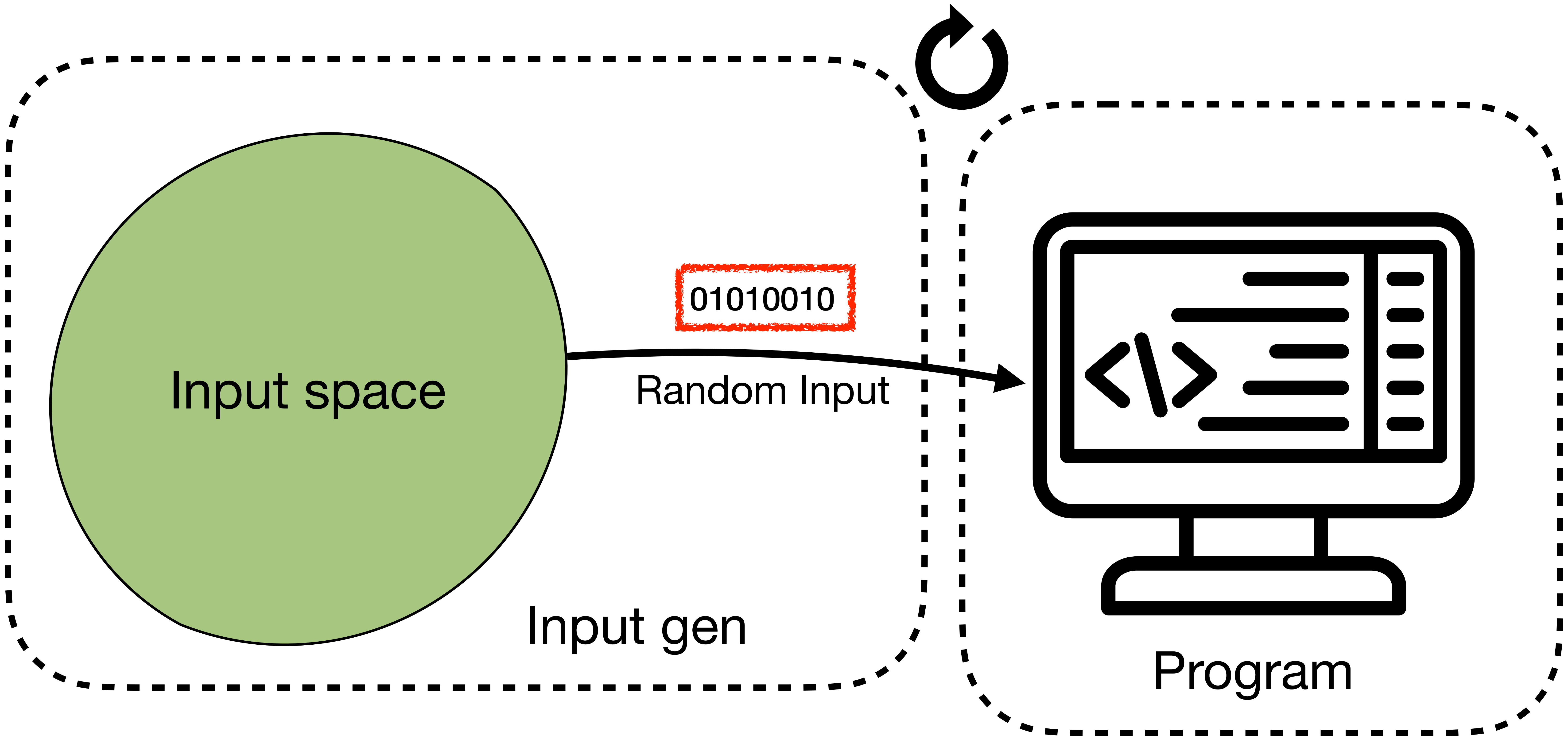
Traditional software testing



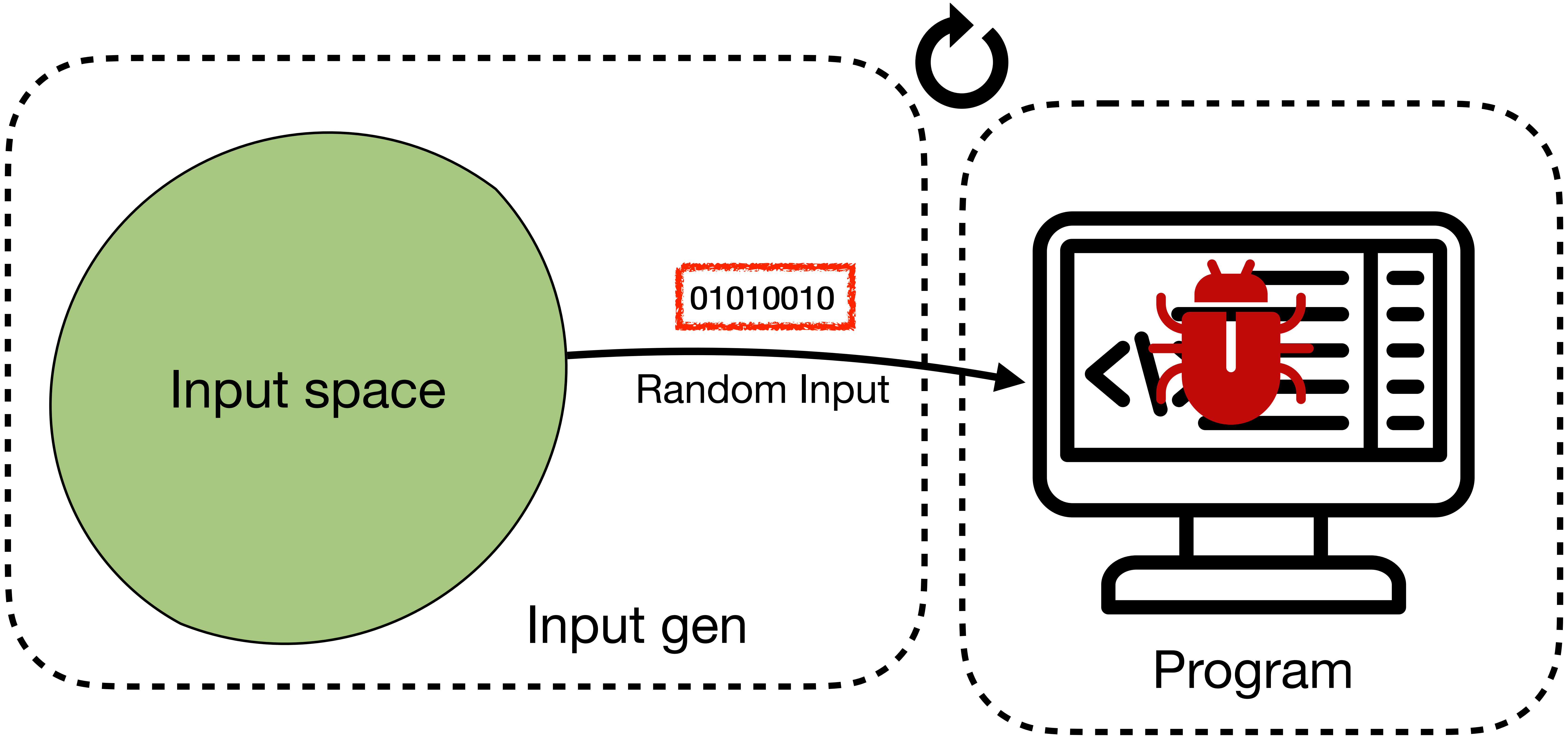
Traditional software testing



Traditional software testing

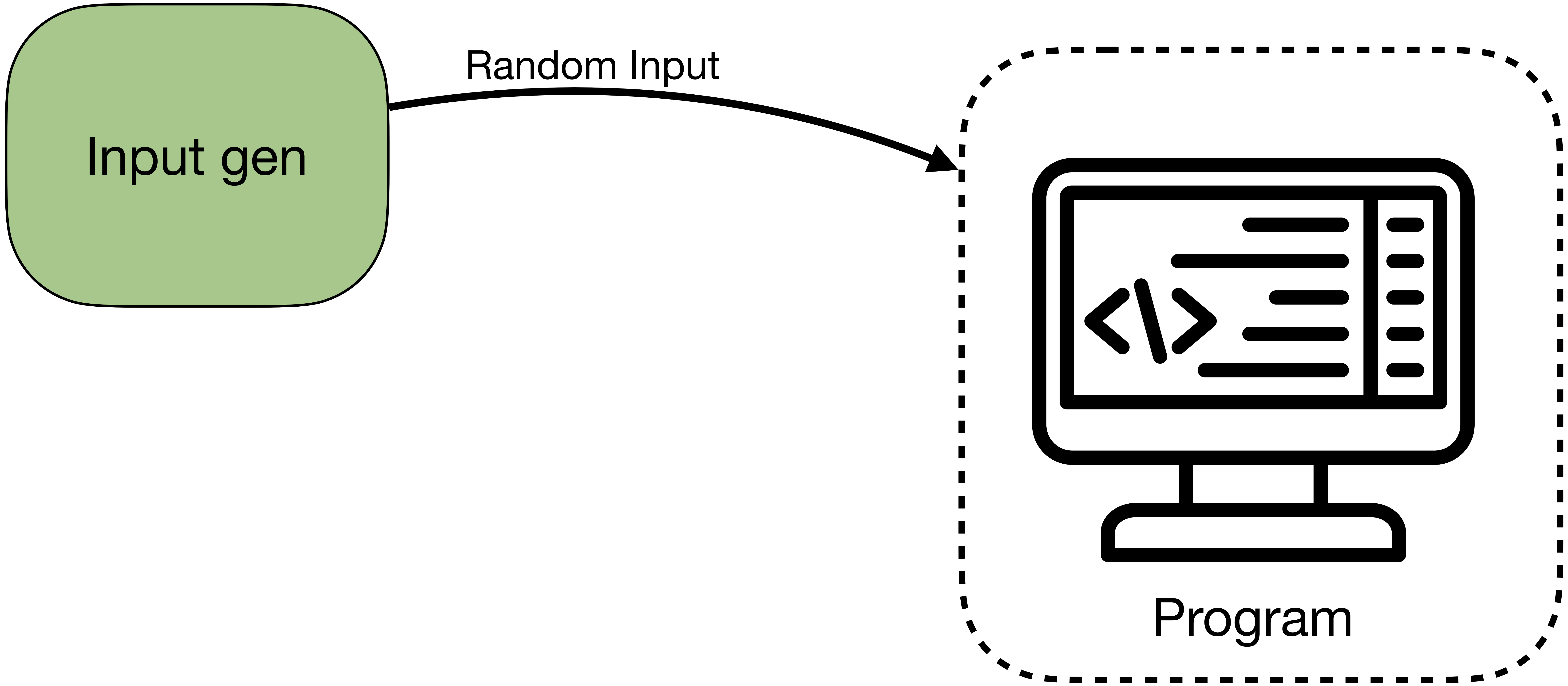


Traditional software testing

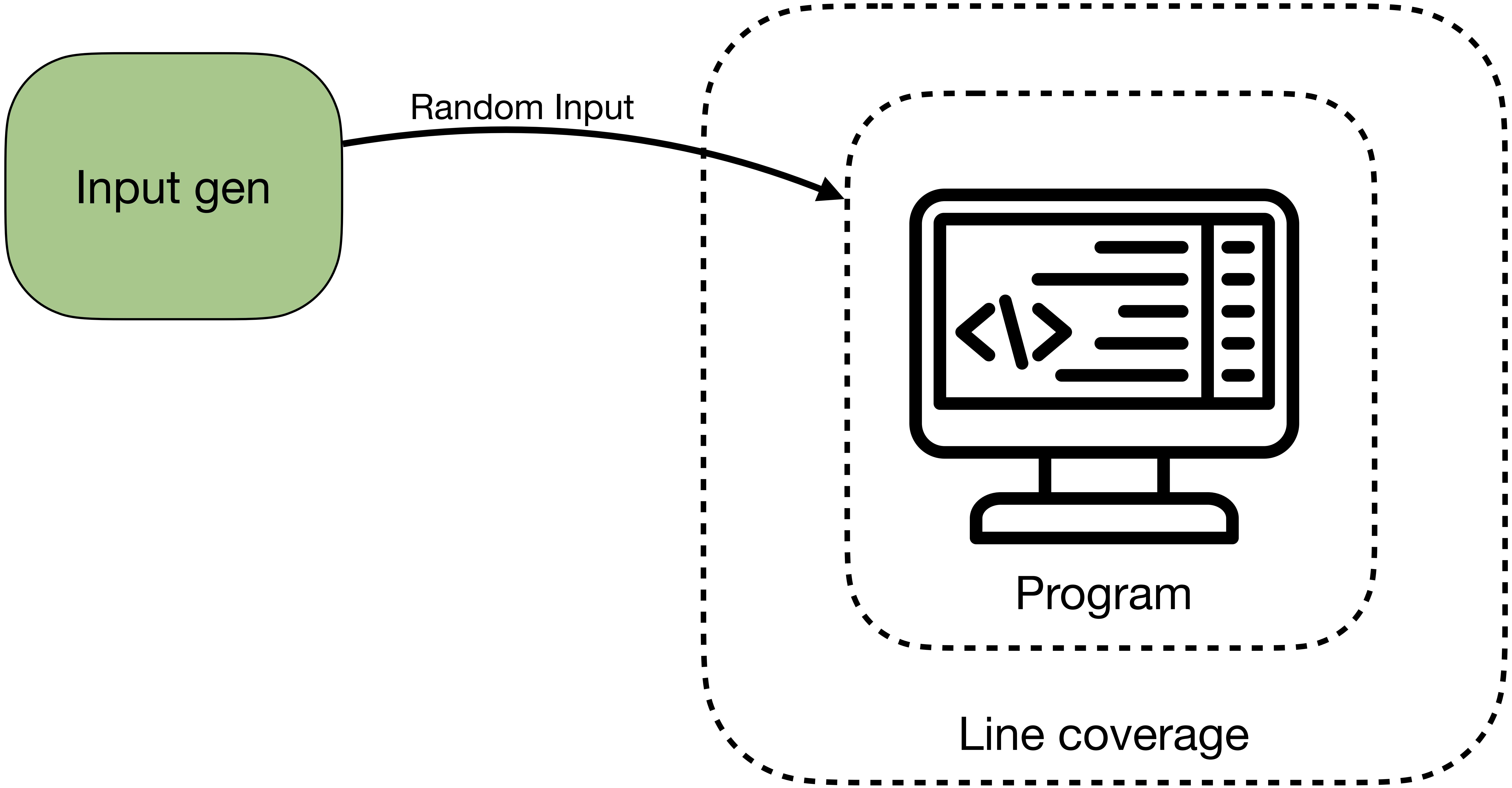


Guided software testing

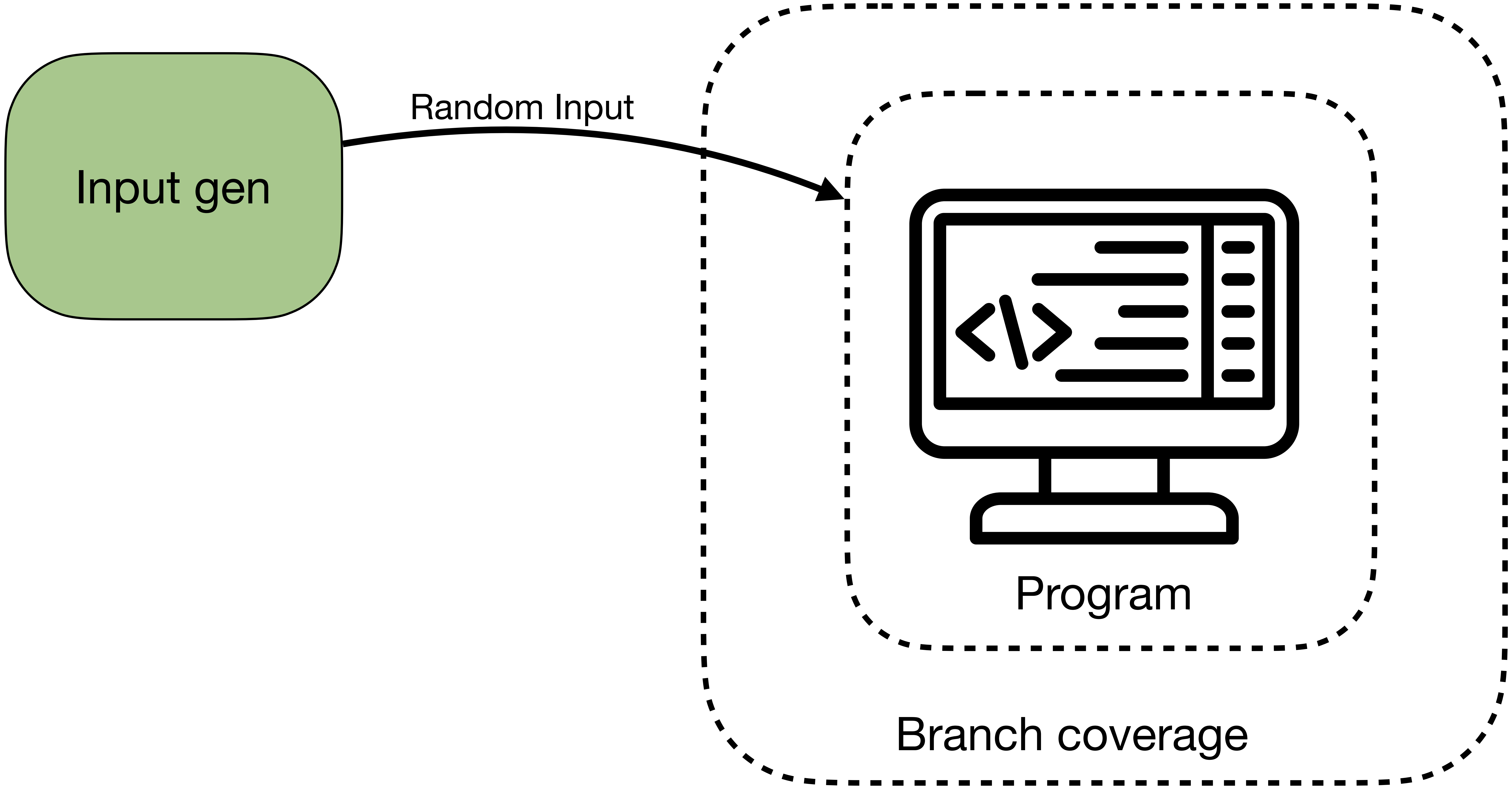
Guided software testing



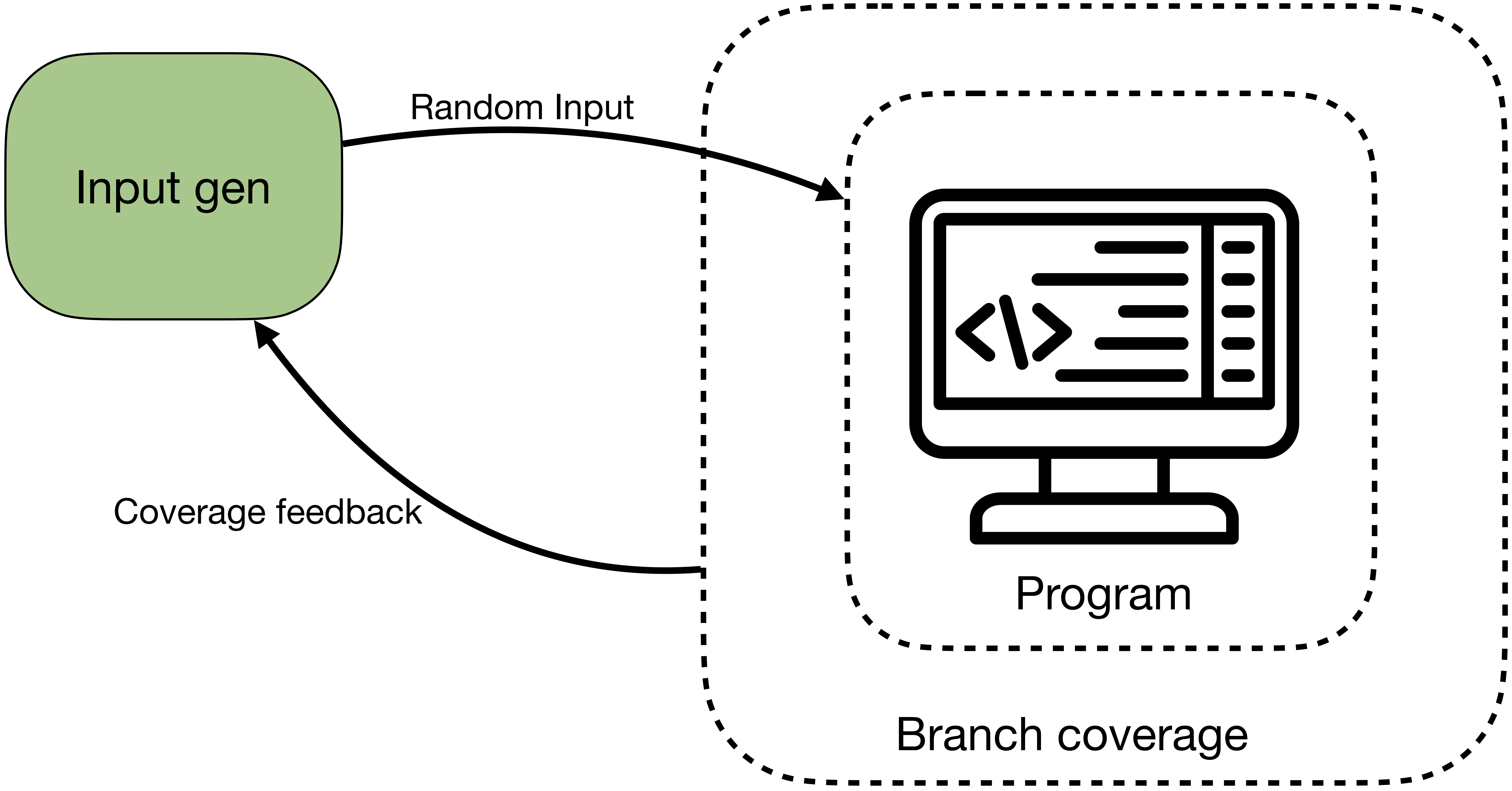
Guided software testing



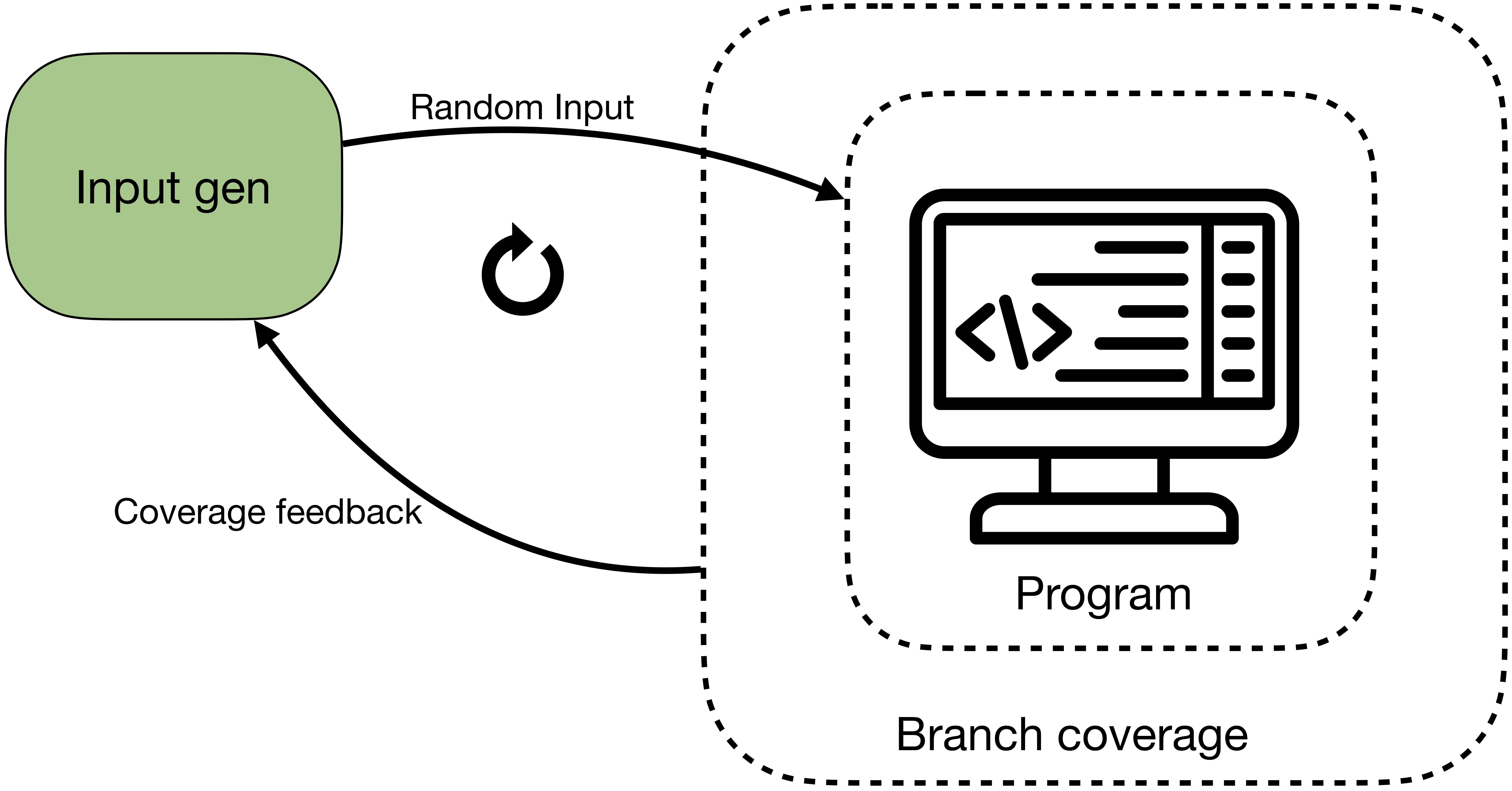
Guided software testing



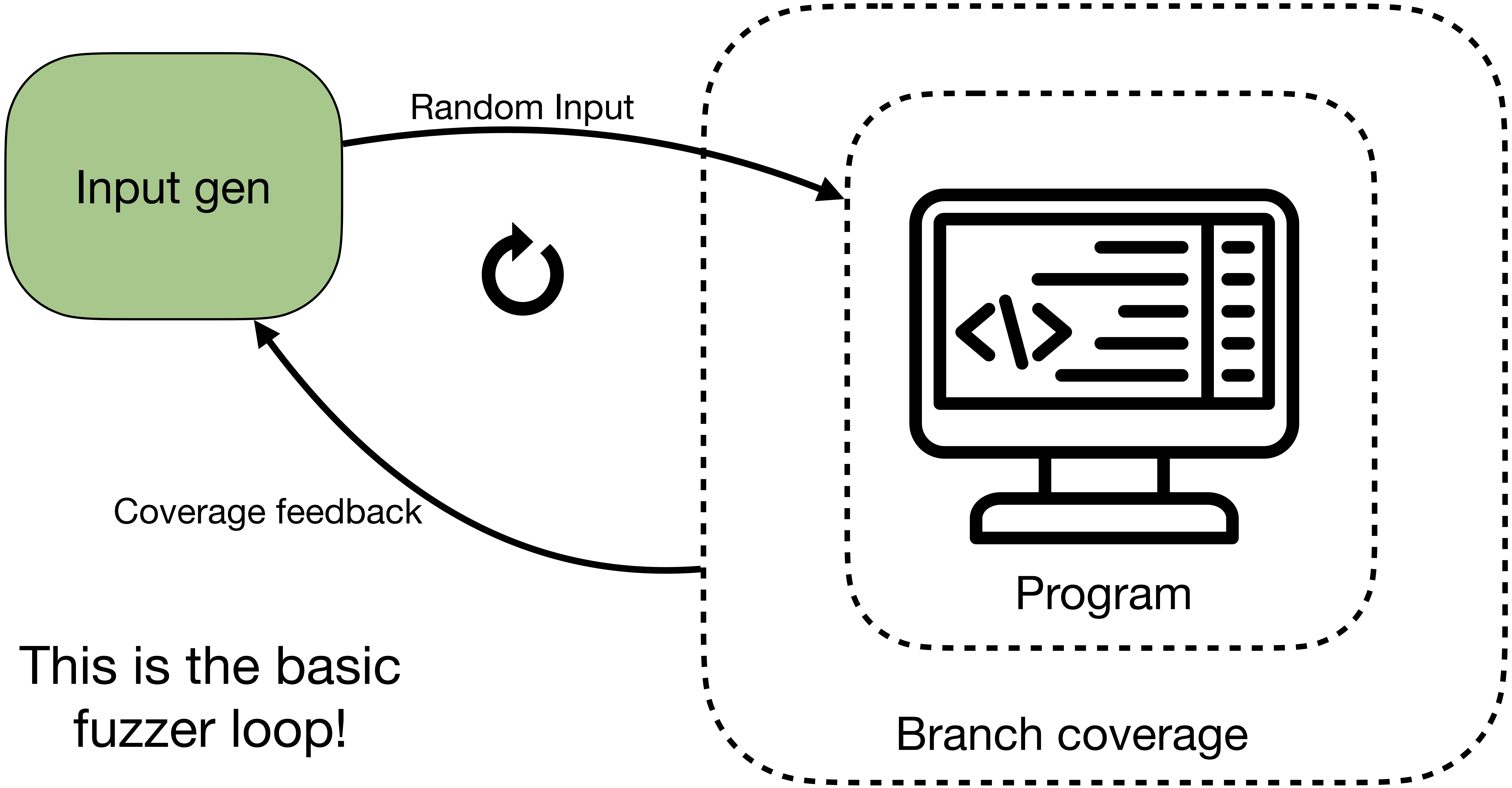
Guided software testing



Guided software testing



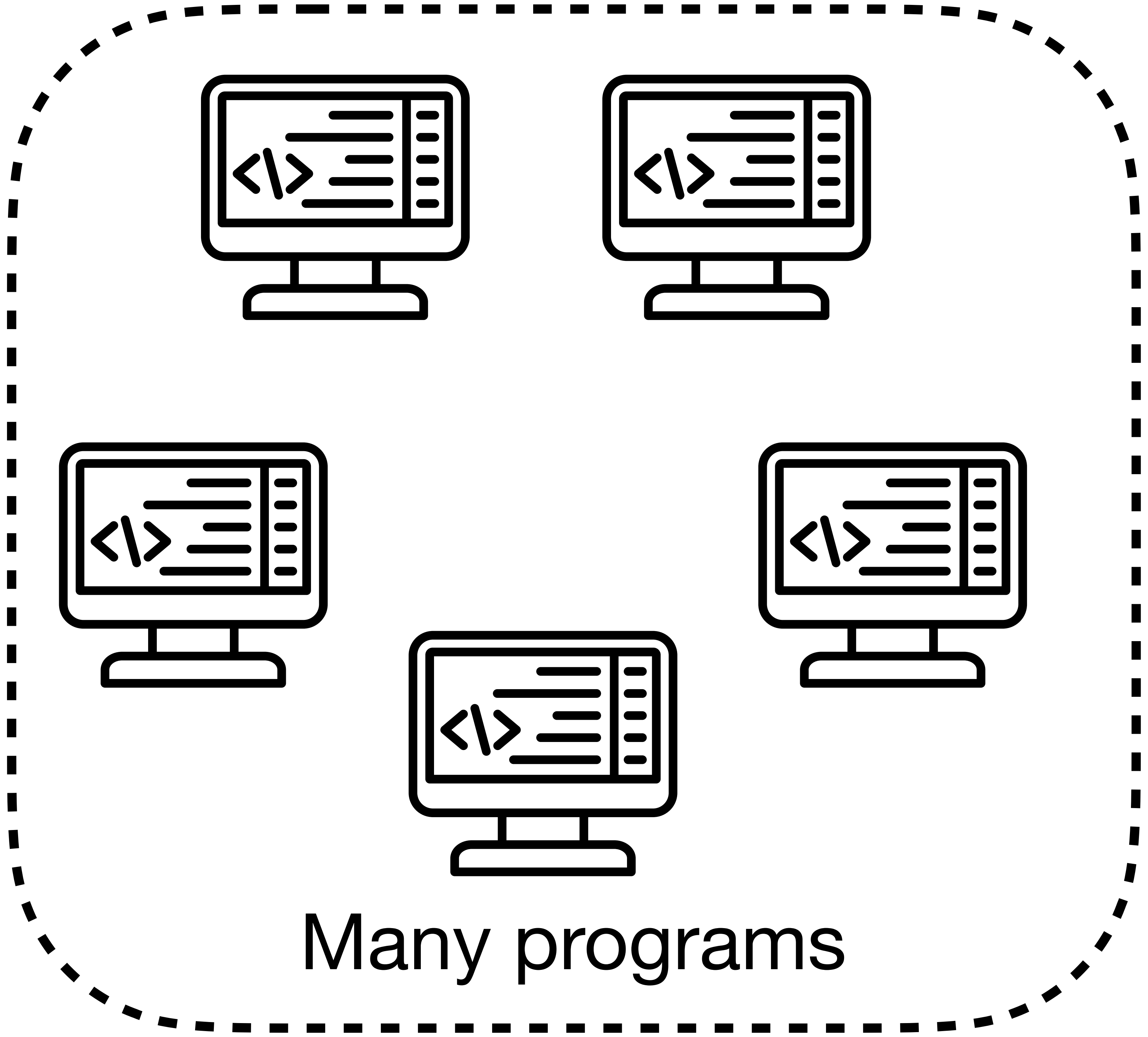
Guided software testing



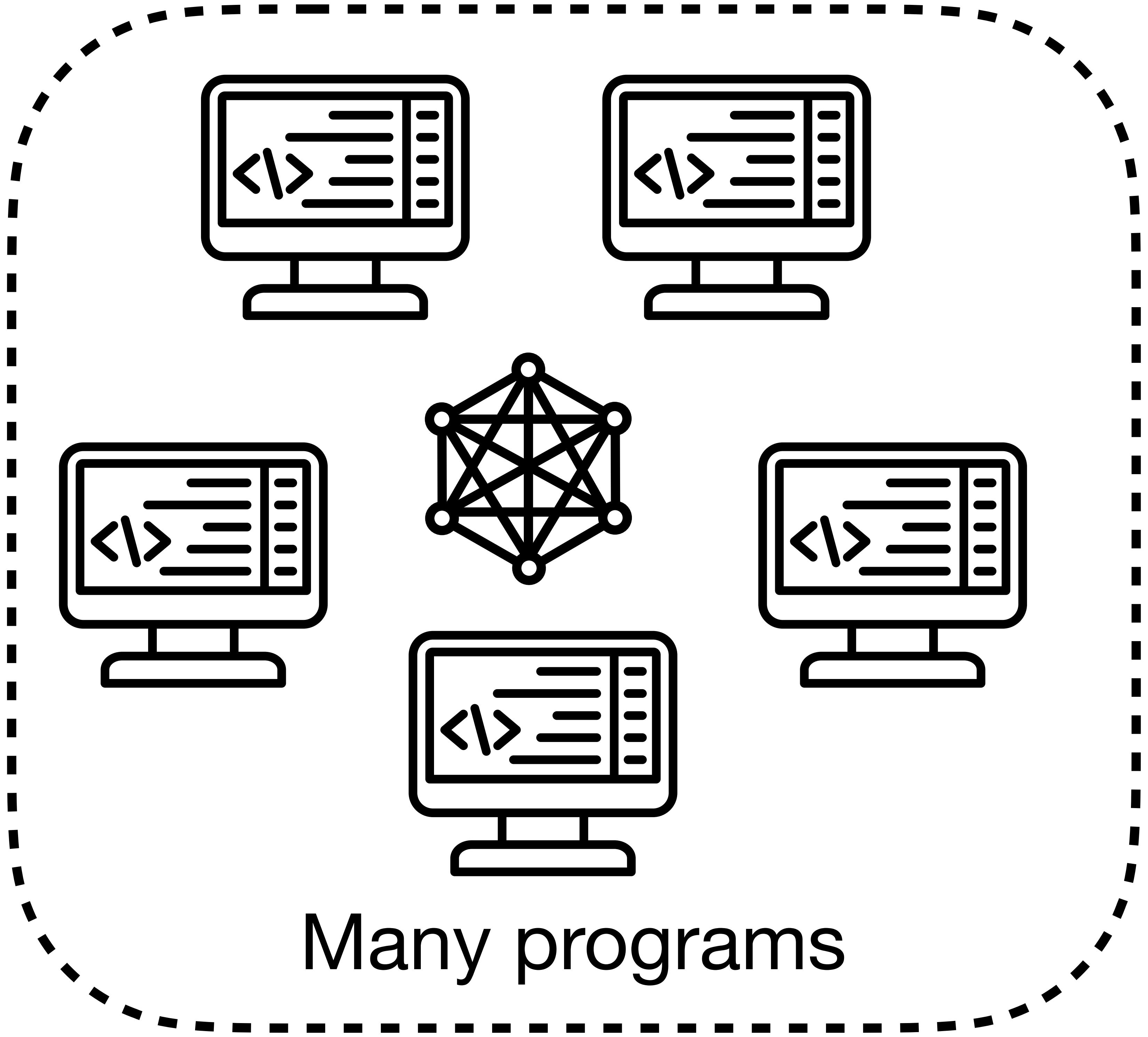
This is the basic fuzzer loop!

Distributed testing

Distributed testing

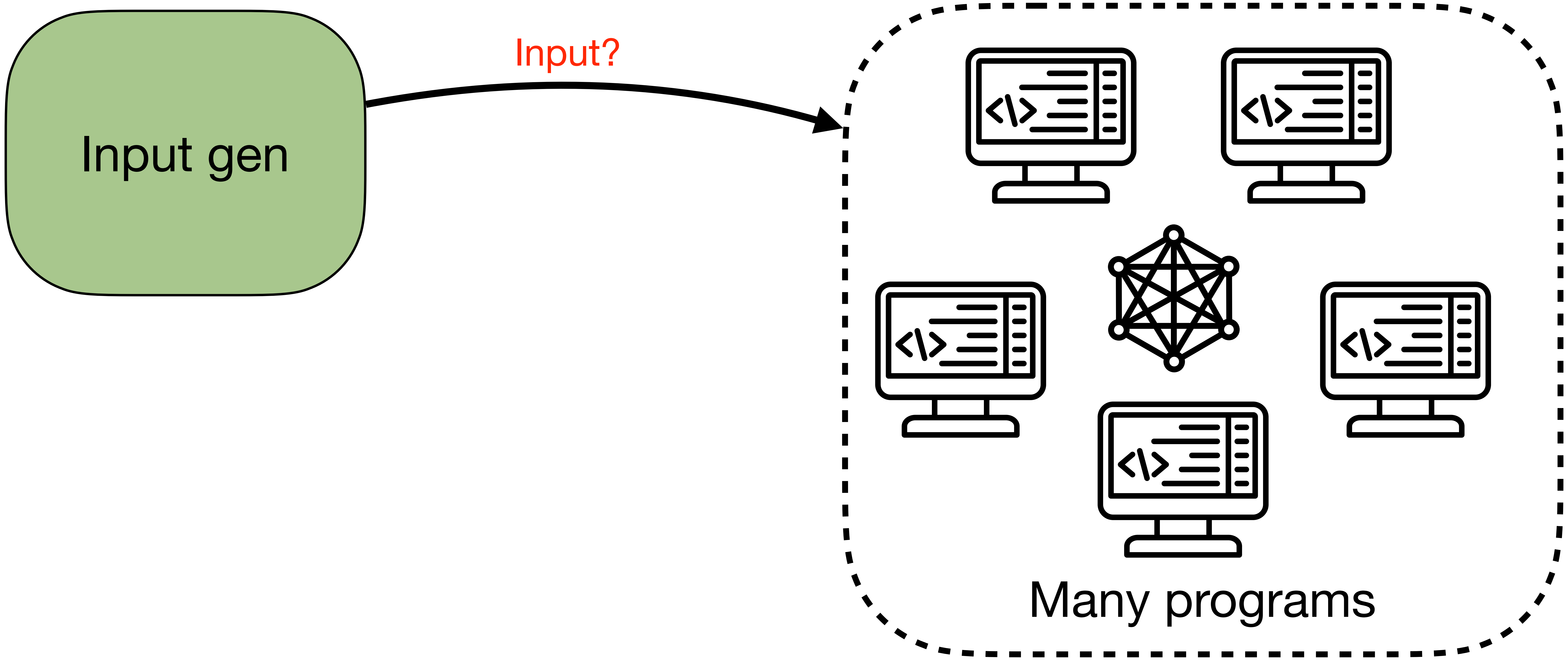


Distributed testing

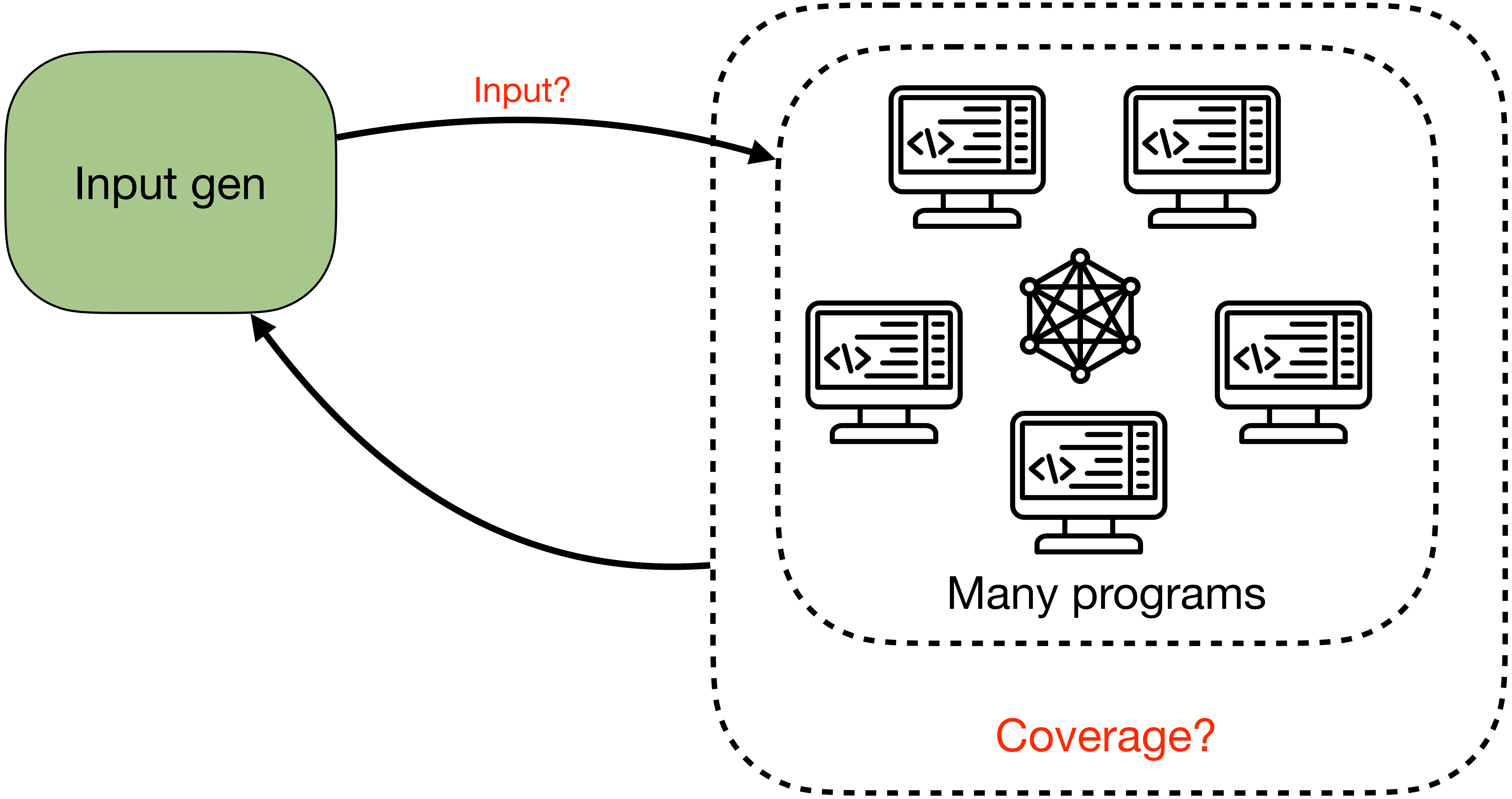


Many programs

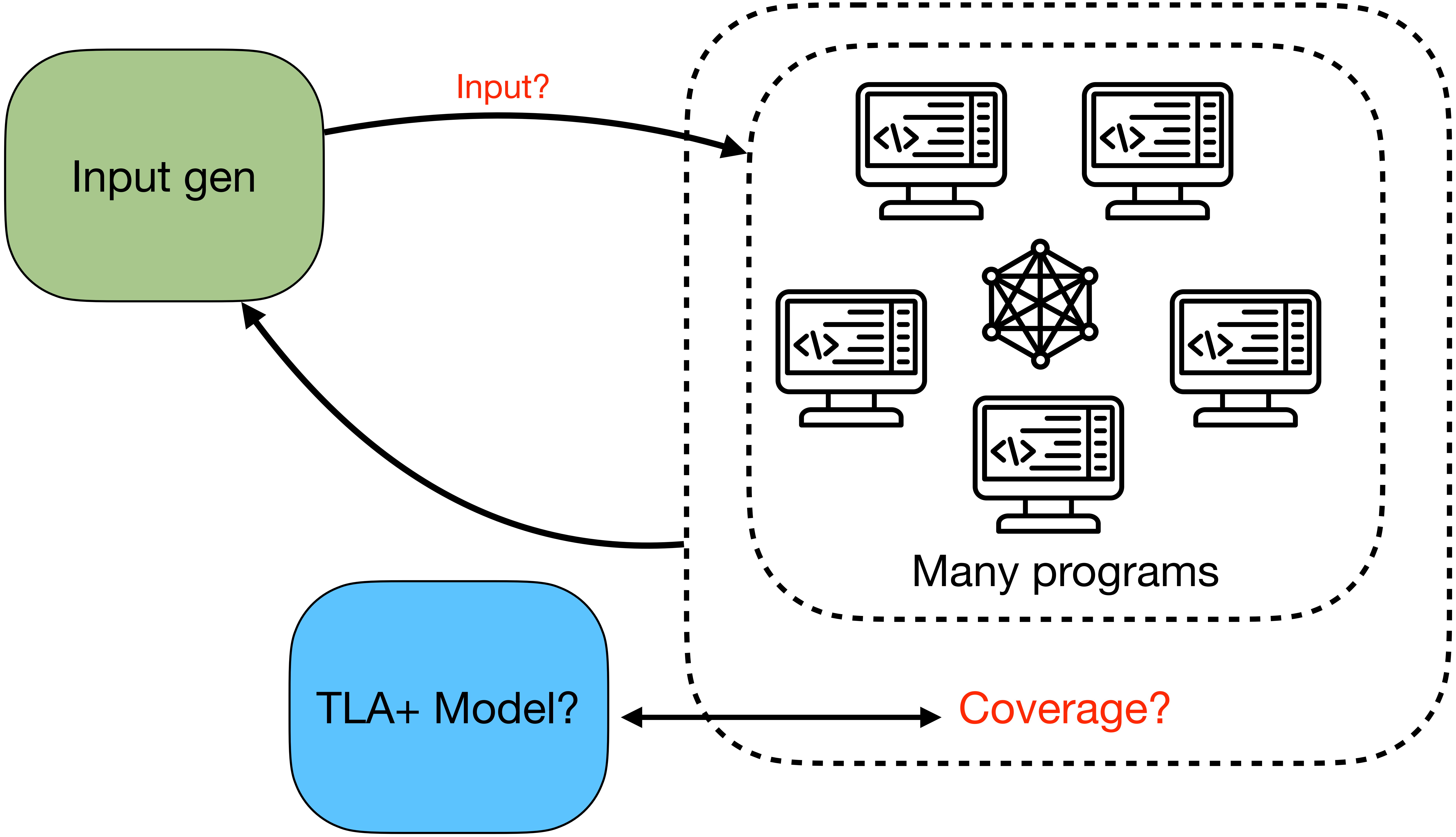
Distributed testing



Distributed testing



Distributed testing



Input gen

Input?

Many programs

TLA+ Model?

Coverage?

Why do we care?

Why do we care?

Why do we care?

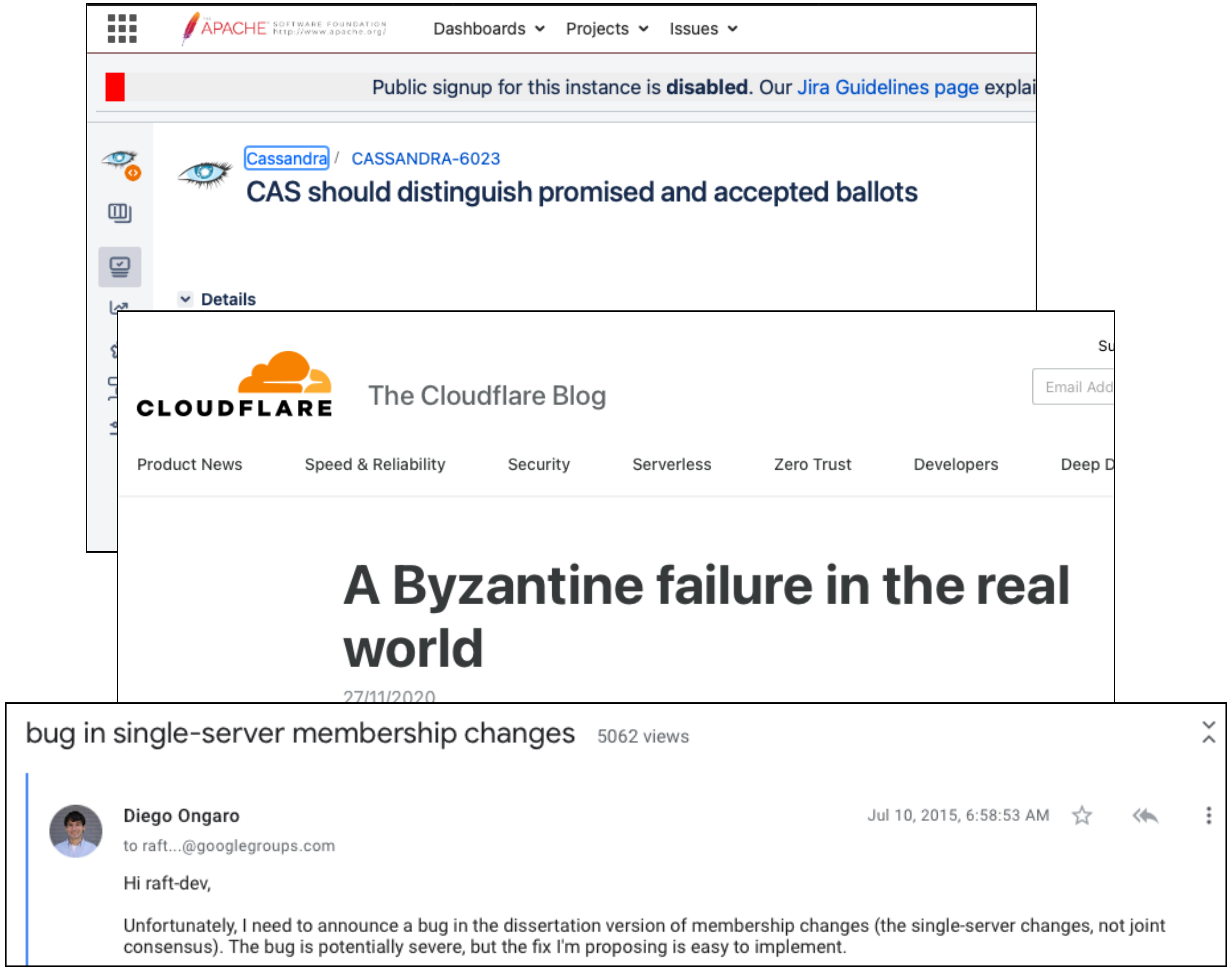
- Complex protocols and Implementations are buggy.

Why do we care?

- Complex protocols and Implementations are buggy.
- Leads to downtimes

Why do we care?

- Complex protocols and Implementations are buggy.
- Leads to downtimes
- E.g. Raft 6 hour outage (liveness), Cassandra inconsistent reorderings (safety)



Public signup for this instance is **disabled**. Our Jira Guidelines page explai

Cassandra / CASSANDRA-6023

CAS should distinguish promised and accepted ballots

Details

The Cloudflare Blog

A Byzantine failure in the real world

27/11/2020

bug in single-server membership changes 5062 views

Diego Ongaro

to raft...@googlegroups.com

Jul 10, 2015, 6:58:53 AM

Hi raft-dev,

Unfortunately, I need to announce a bug in the dissertation version of membership changes (the single-server changes, not joint consensus). The bug is potentially severe, but the fix I'm proposing is easy to implement.

Existing work

Existing work

Implementation testing

Existing work

Implementation testing

JEPSEN

- Jepsen - Randomized testing tool

Existing work

Implementation testing

JEPSEN

- Jepsen - Randomized testing tool

PCT, PCTCP

- Randomised testing with probabilistic guarantees

Existing work

Implementation testing

JEPSEN

- Jepsen - Randomized testing tool

coyote

- Testing framework.
- QL - learning based techniques

PCT, PCTCP

- Randomised testing with probabilistic guarantees

Existing work

Implementation testing

JEPSEN

- Jepsen - Randomized testing tool

coyote

- Testing framework.
- QL - learning based techniques

PCT, PCTCP

- Randomised testing with probabilistic guarantees

Mocket

- model based testing
- Generate tests from TLA+ model

Existing work

Implementation testing

JEPSEN

- Jepsen - Randomized testing tool

coyote

- Testing framework.
- QL - learning based techniques

PCT, PCTCP

- Randomised testing with probabilistic guarantees

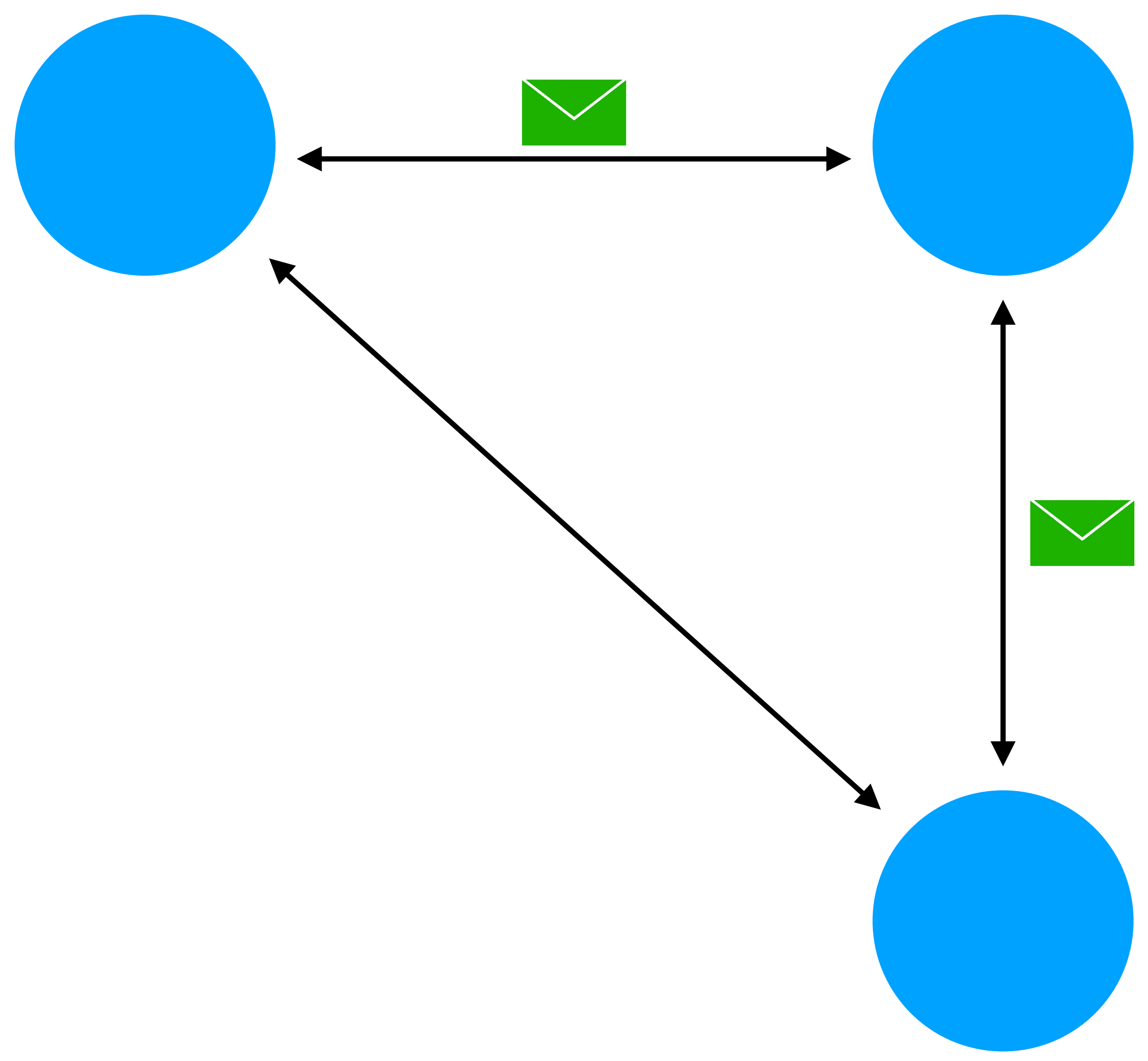
Mocket

- model based testing
- Generate tests from TLA+ model

Example protocol - Raft

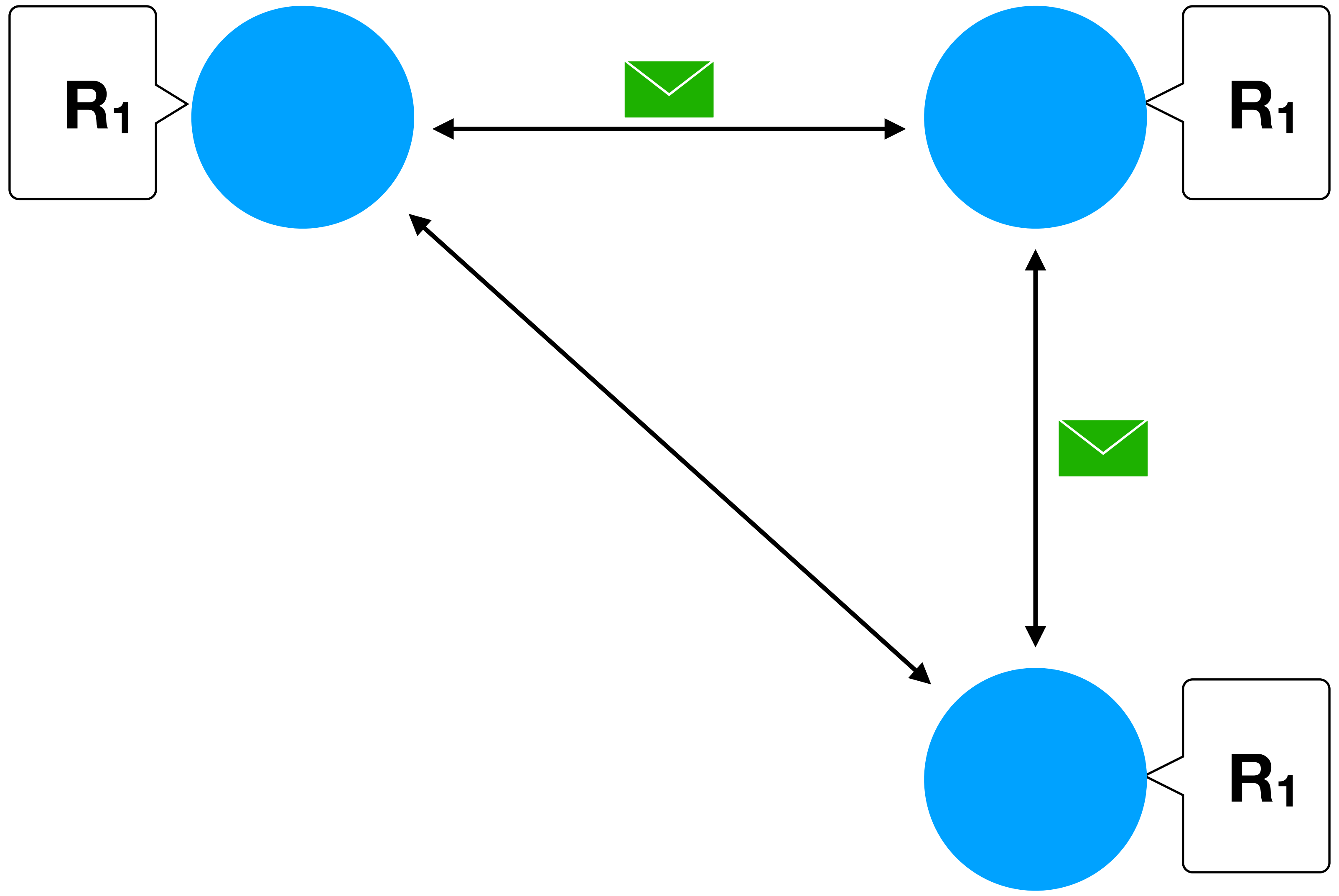
Example protocol - Raft

- Distributed message passing



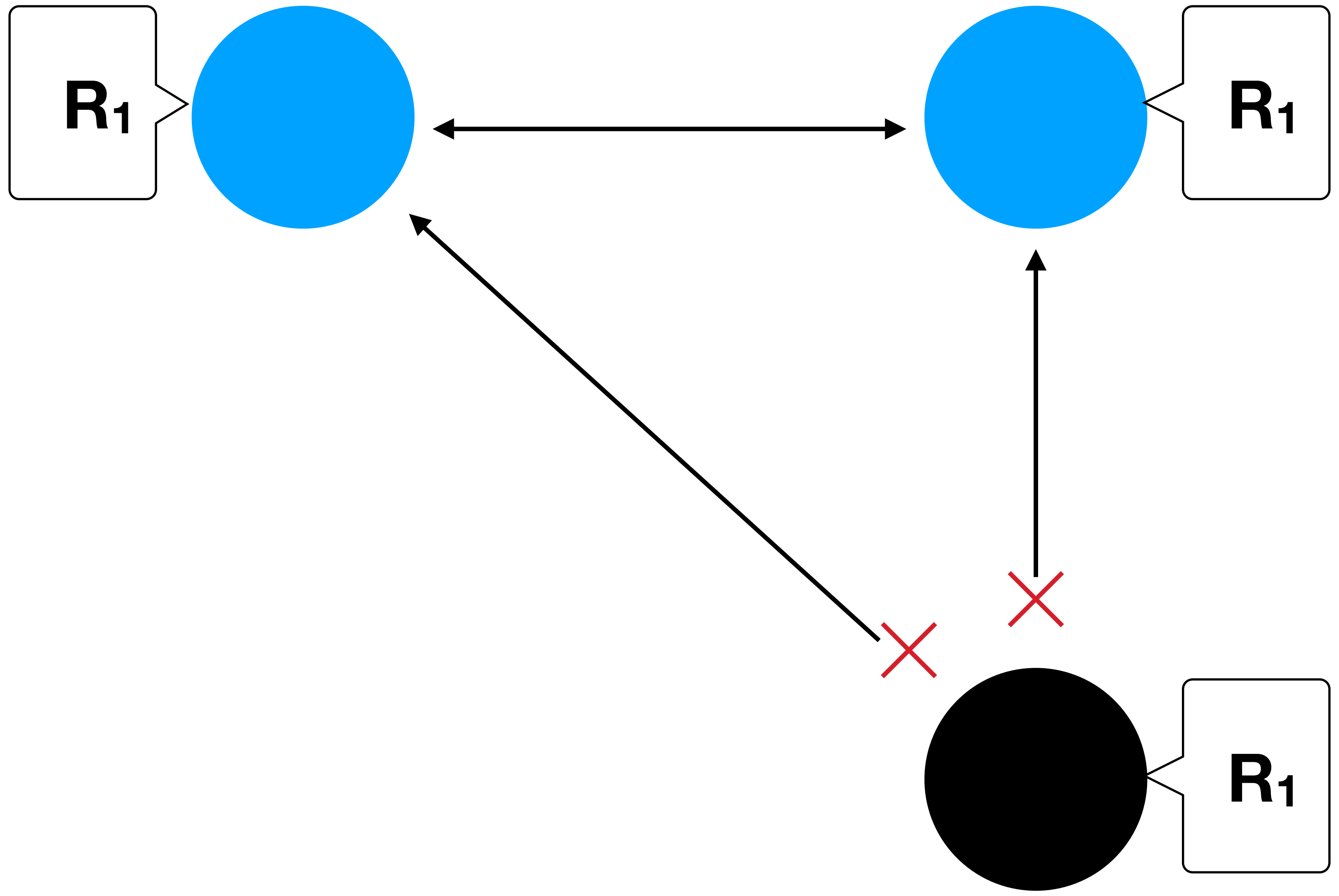
Example protocol - Raft

- Distributed message passing
- Solves consensus



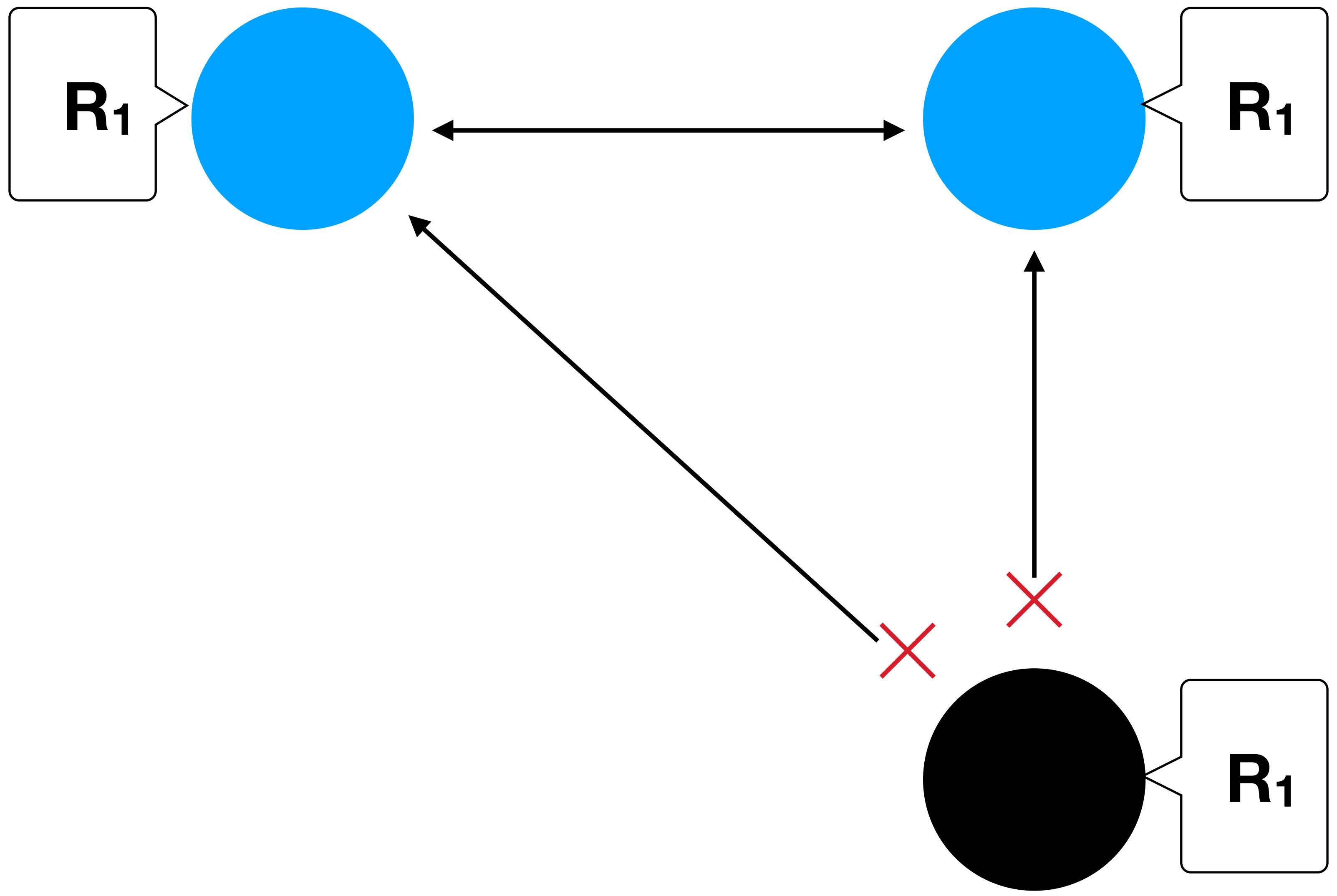
Example protocol - Raft

- Distributed message passing
- Solves consensus
  - With crashes



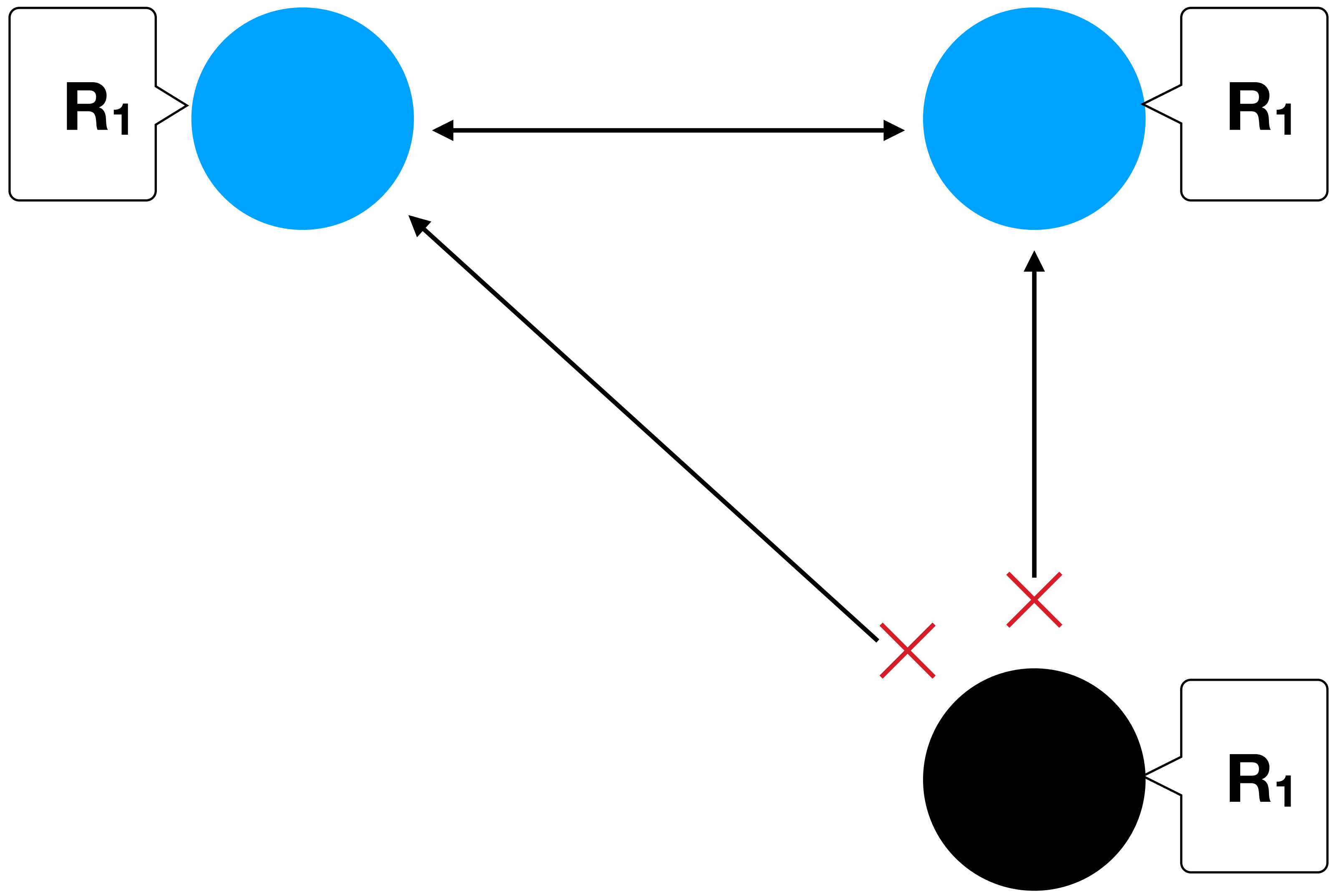
Example protocol - Raft

- Distributed message passing
- Solves consensus
  - With crashes
- Two phases:



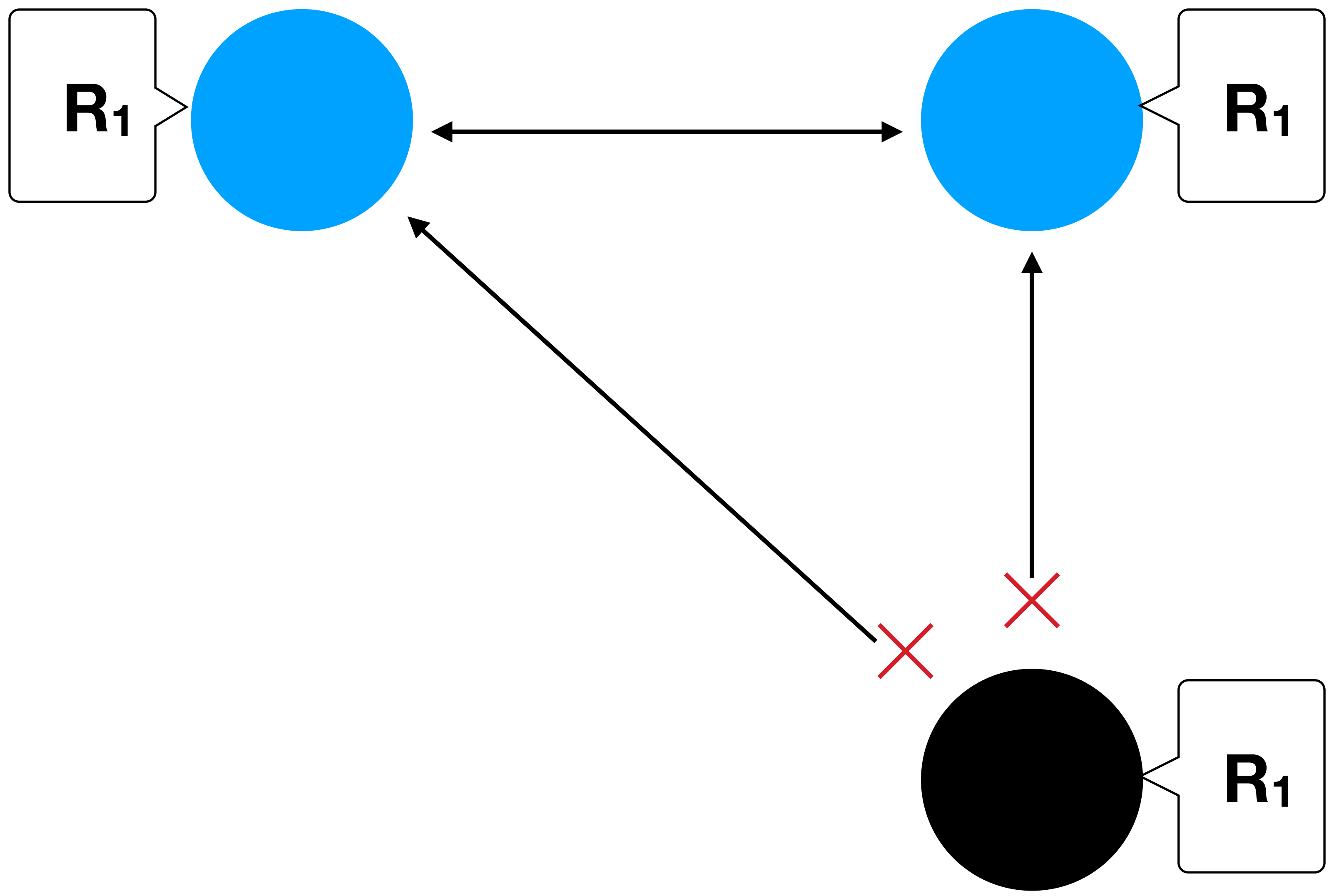
Example protocol - Raft

- Distributed message passing
- Solves consensus
  - With crashes
- Two phases:
  - Leader election phase



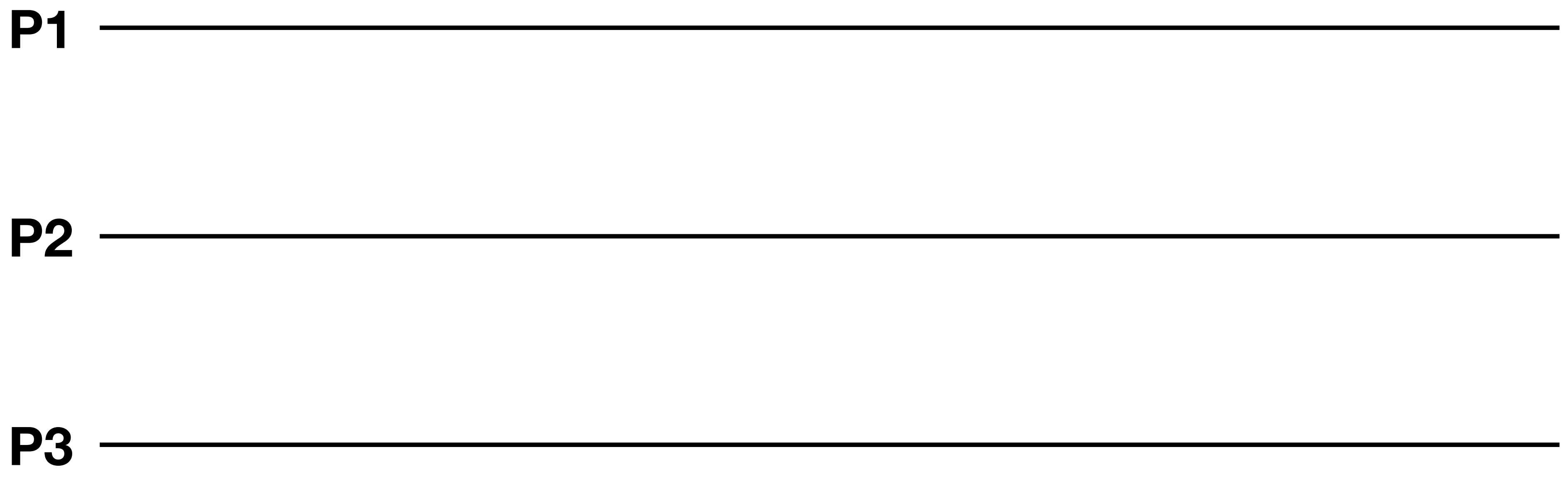
Example protocol - Raft

- Distributed message passing
- Solves consensus
  - With crashes
- Two phases:
  - Leader election phase
  - Leader replication phase

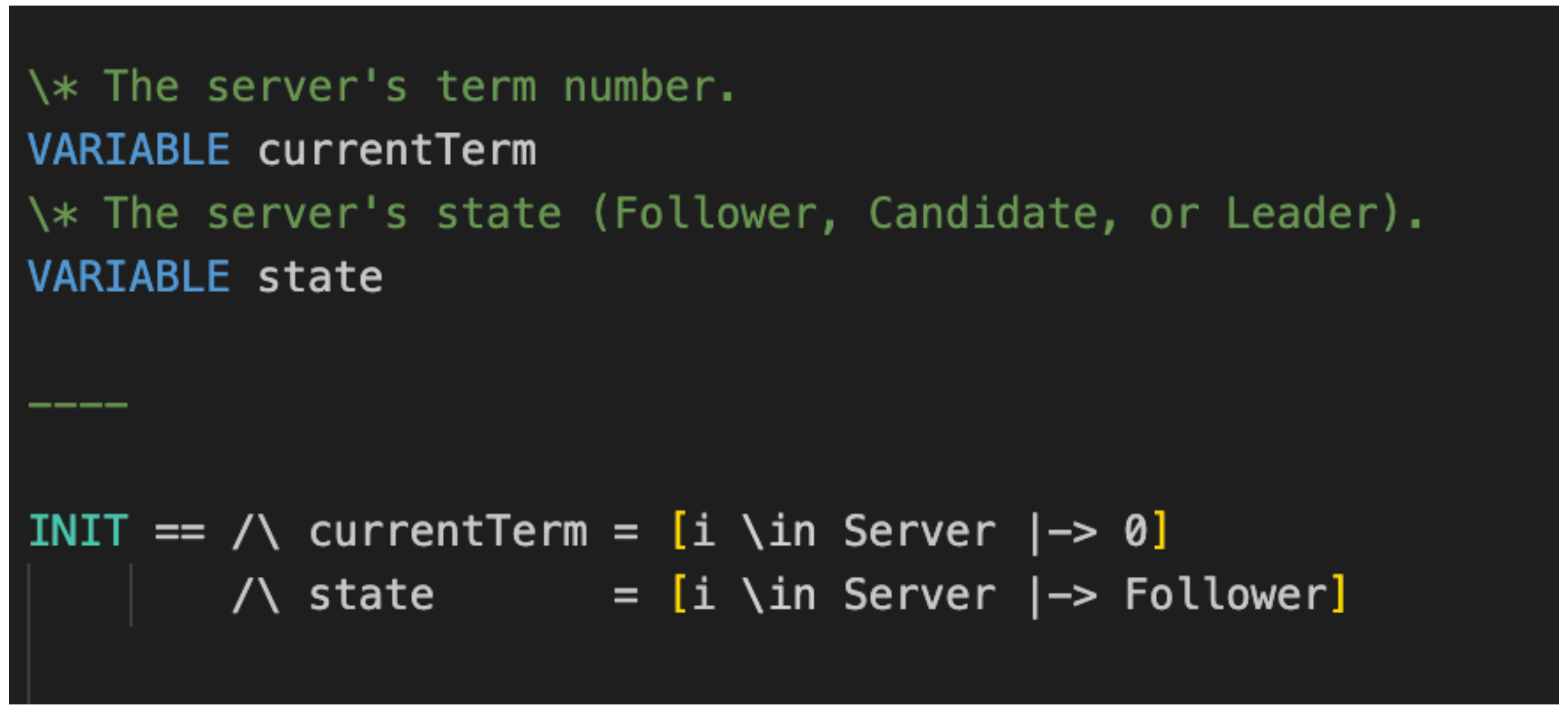


Raft TLA

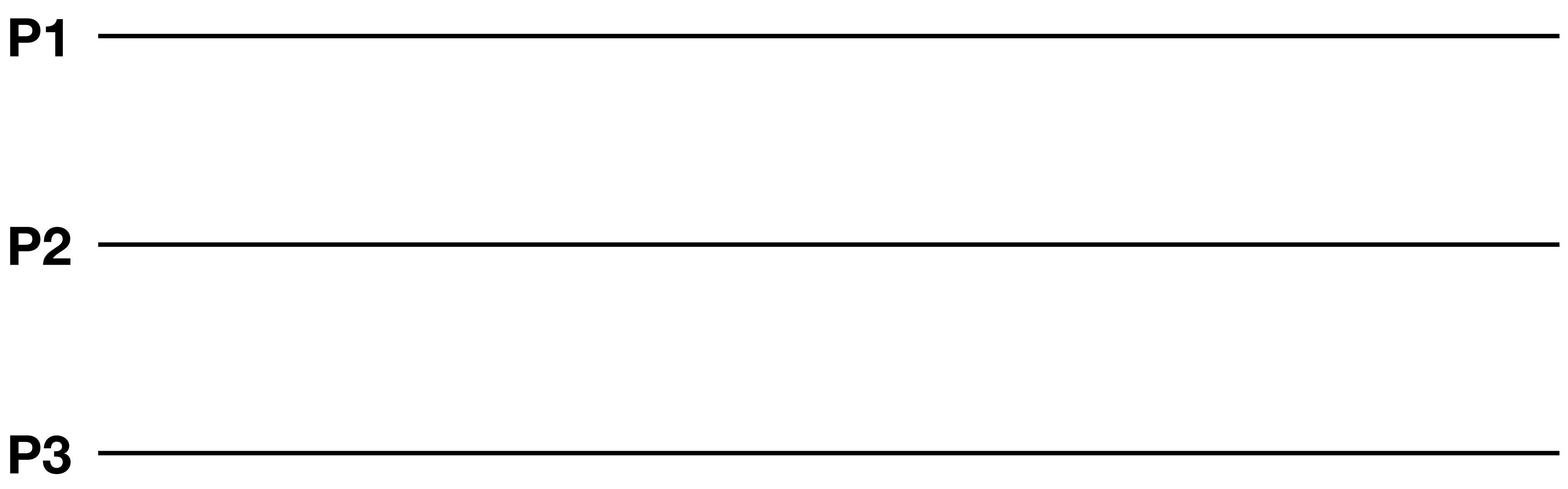
Raft TLA



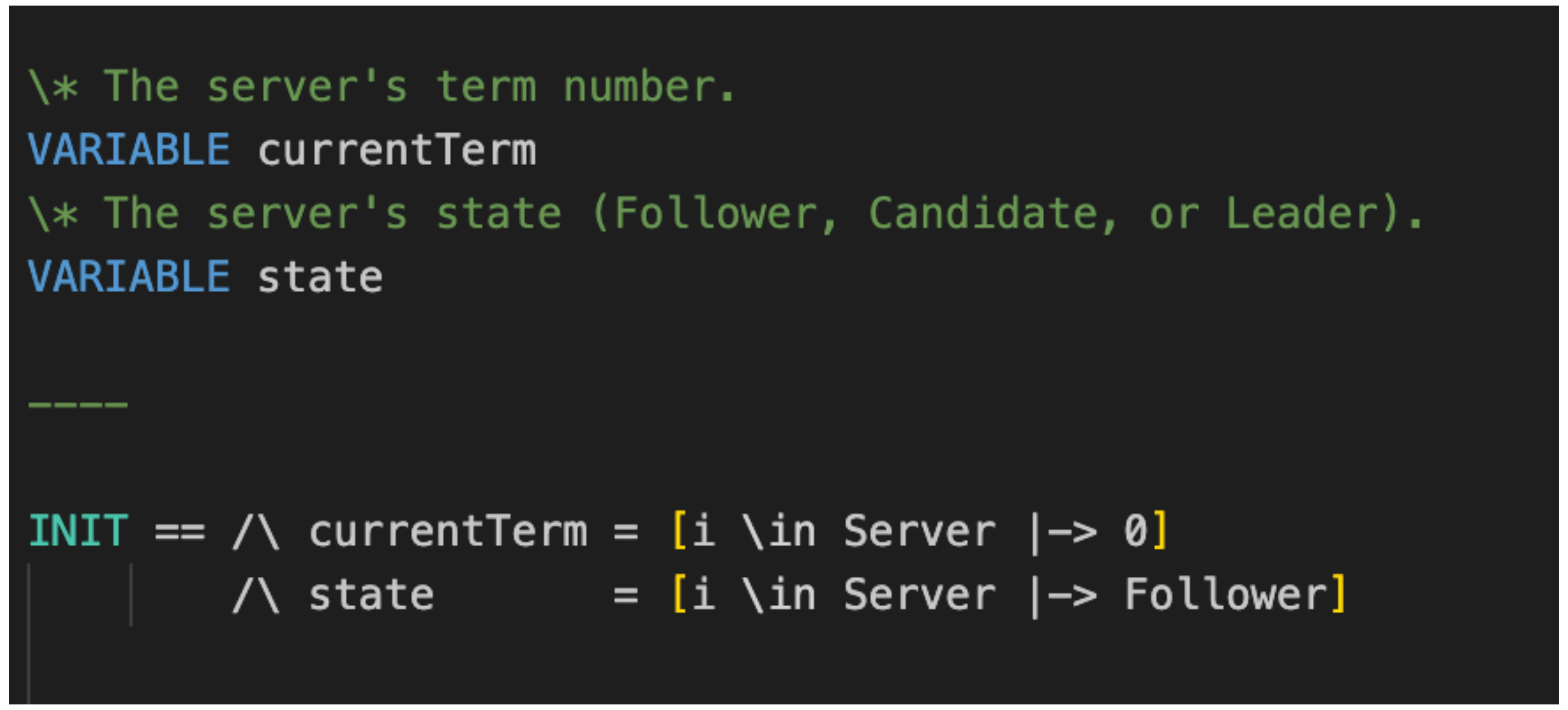
Raft TLA



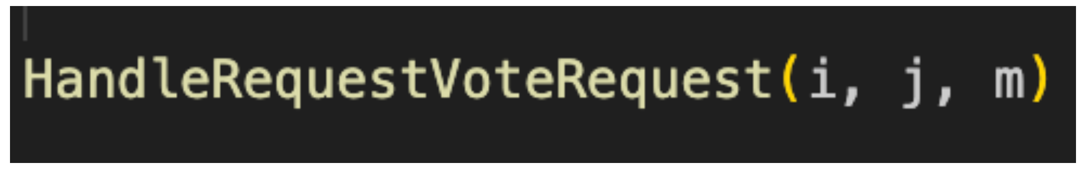
```
\* The server's term number.
VARIABLE currentTerm
\* The server's state (Follower, Candidate, or Leader).
VARIABLE state

----

INIT == /\ currentTerm = [i \in Server |-> 0]
        /\ state       = [i \in Server |-> Follower]
```



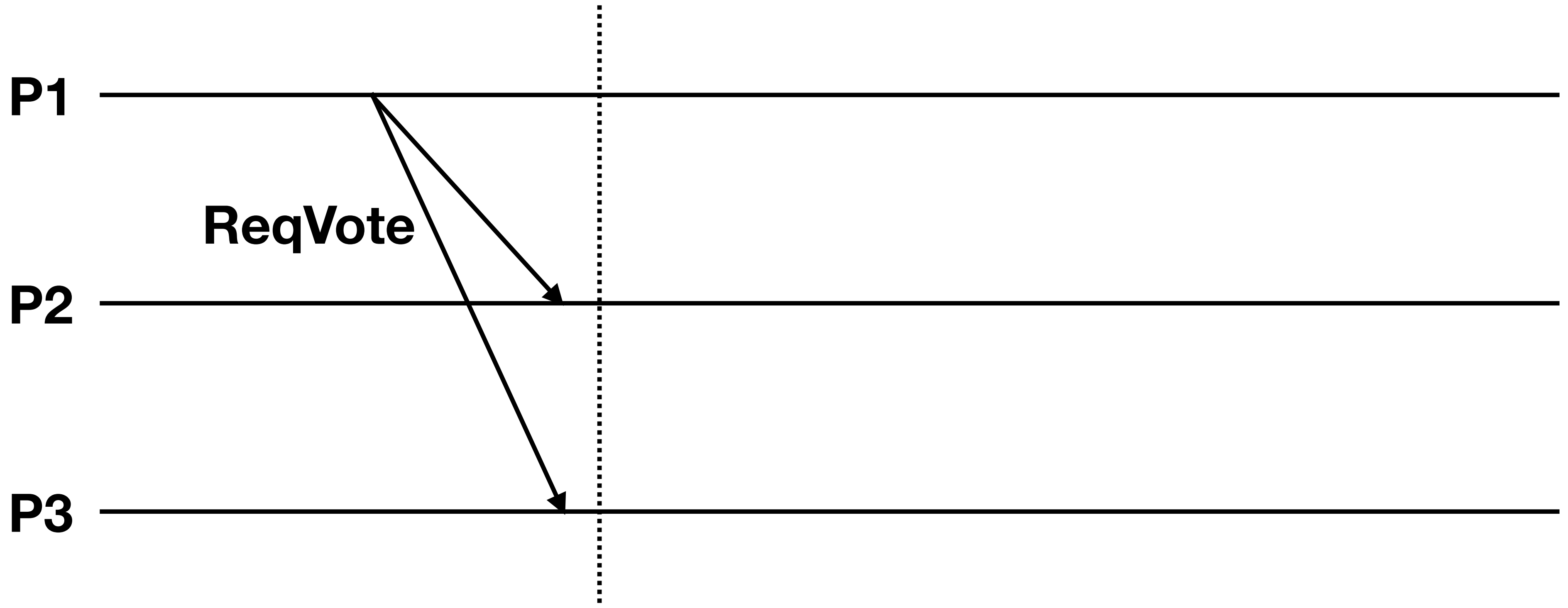
Raft TLA



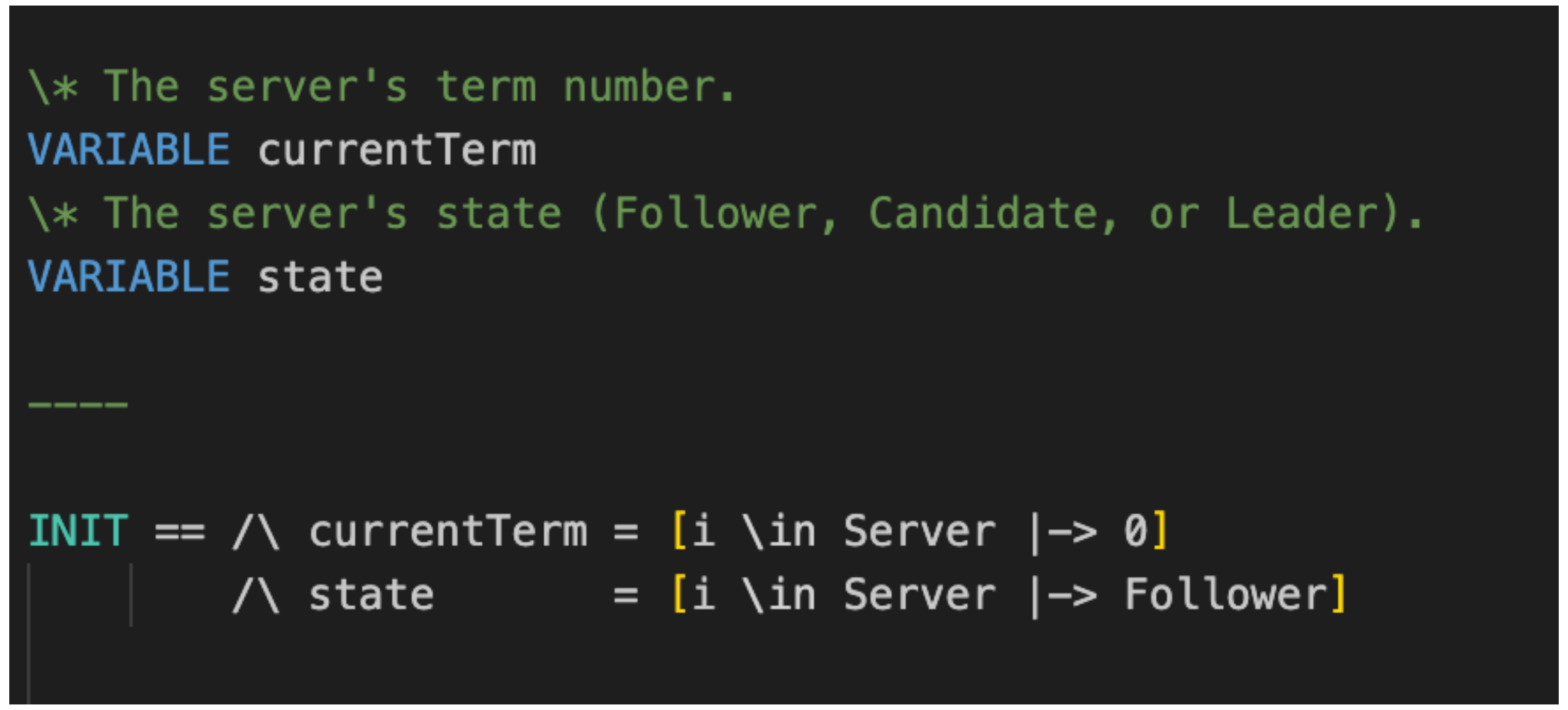
```
\* The server's term number.
VARIABLE currentTerm
\* The server's state (Follower, Candidate, or Leader).
VARIABLE state

----

INIT == /\ currentTerm = [i \in Server |-> 0]
        /\ state       = [i \in Server |-> Follower]
```



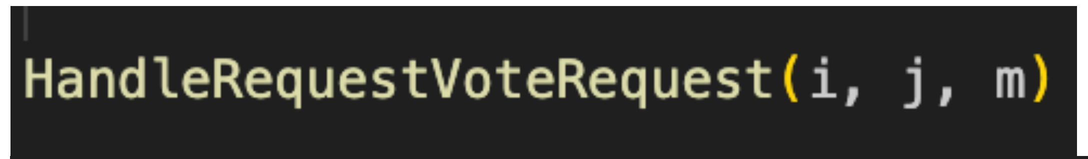
```
HandleRequestVoteRequest(i, j, m)
```



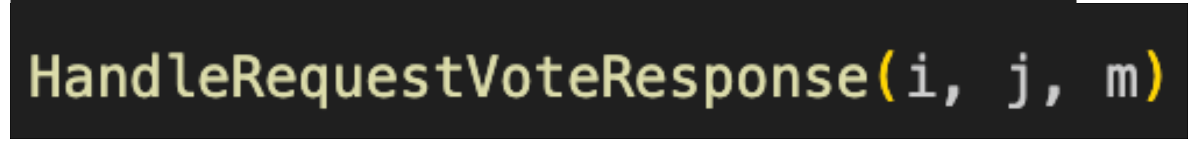
Raft TLA



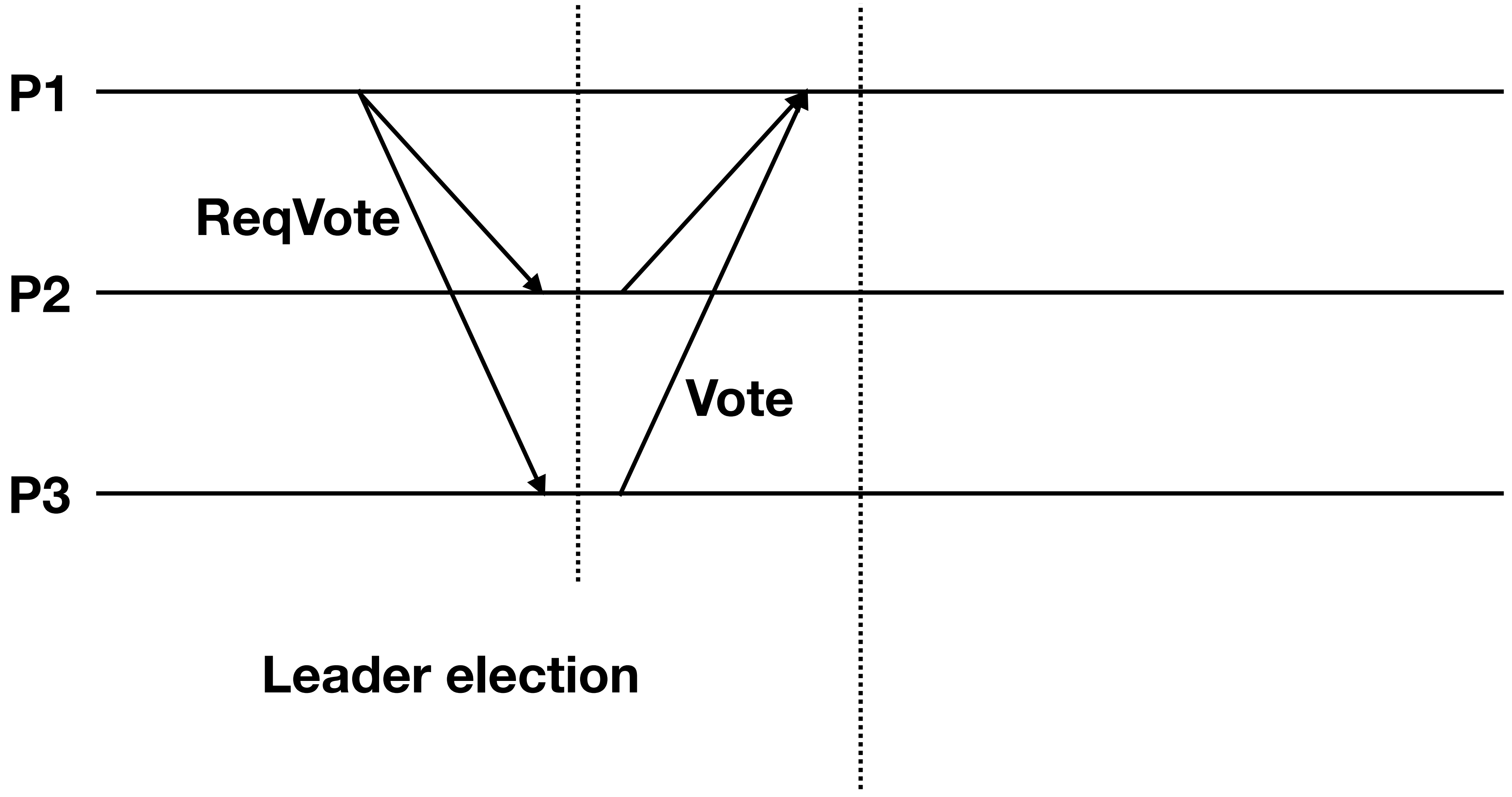
```
\* The server's term number.
VARIABLE currentTerm
\* The server's state (Follower, Candidate, or Leader).
VARIABLE state

----

INIT == /\ currentTerm = [i \in Server |-> 0]
        /\ state       = [i \in Server |-> Follower]
```



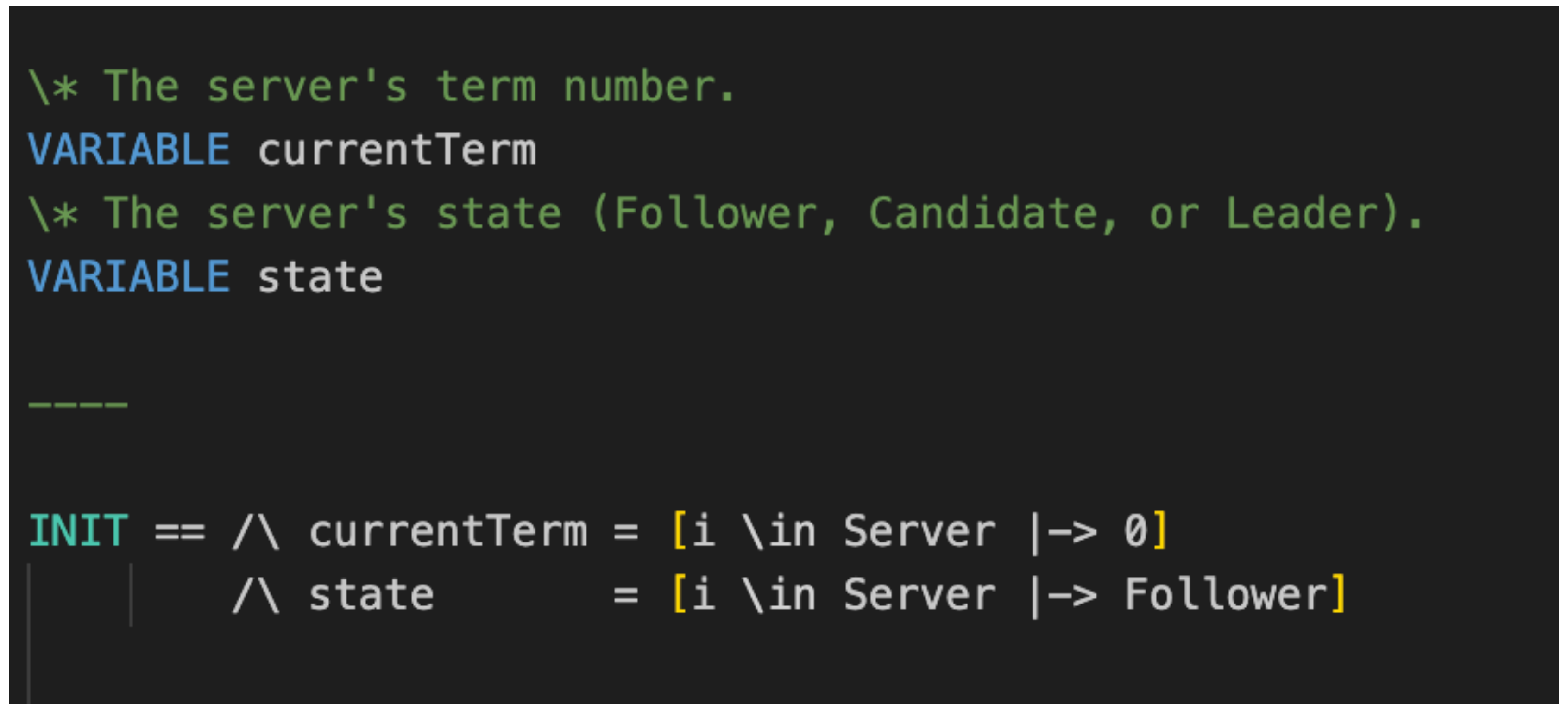
```
HandleRequestVoteRequest(i, j, m)
```



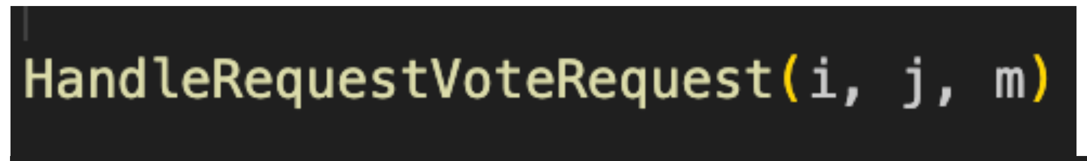
```
HandleRequestVoteResponse(i, j, m)
```



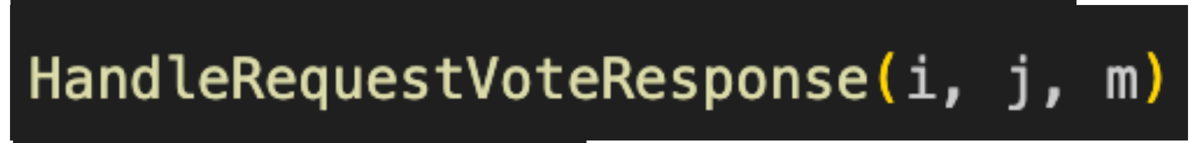
Raft TLA



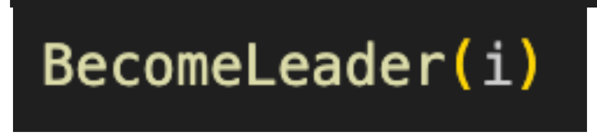
```
\* The server's term number.
VARIABLE currentTerm
\* The server's state (Follower, Candidate, or Leader).
VARIABLE state

----

INIT == /\ currentTerm = [i \in Server |-> 0]
        /\ state       = [i \in Server |-> Follower]
```



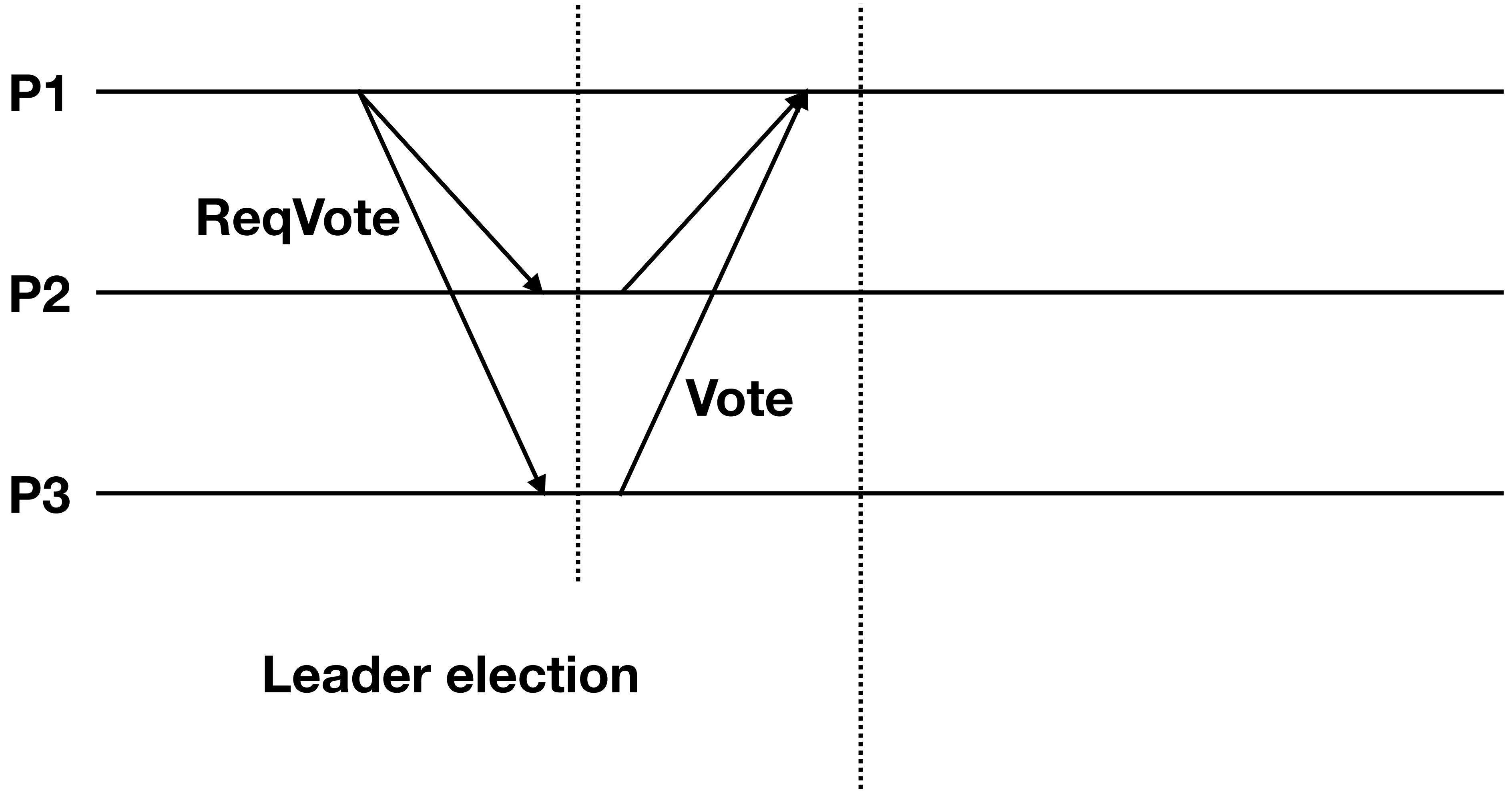
```
HandleRequestVoteRequest(i, j, m)
```



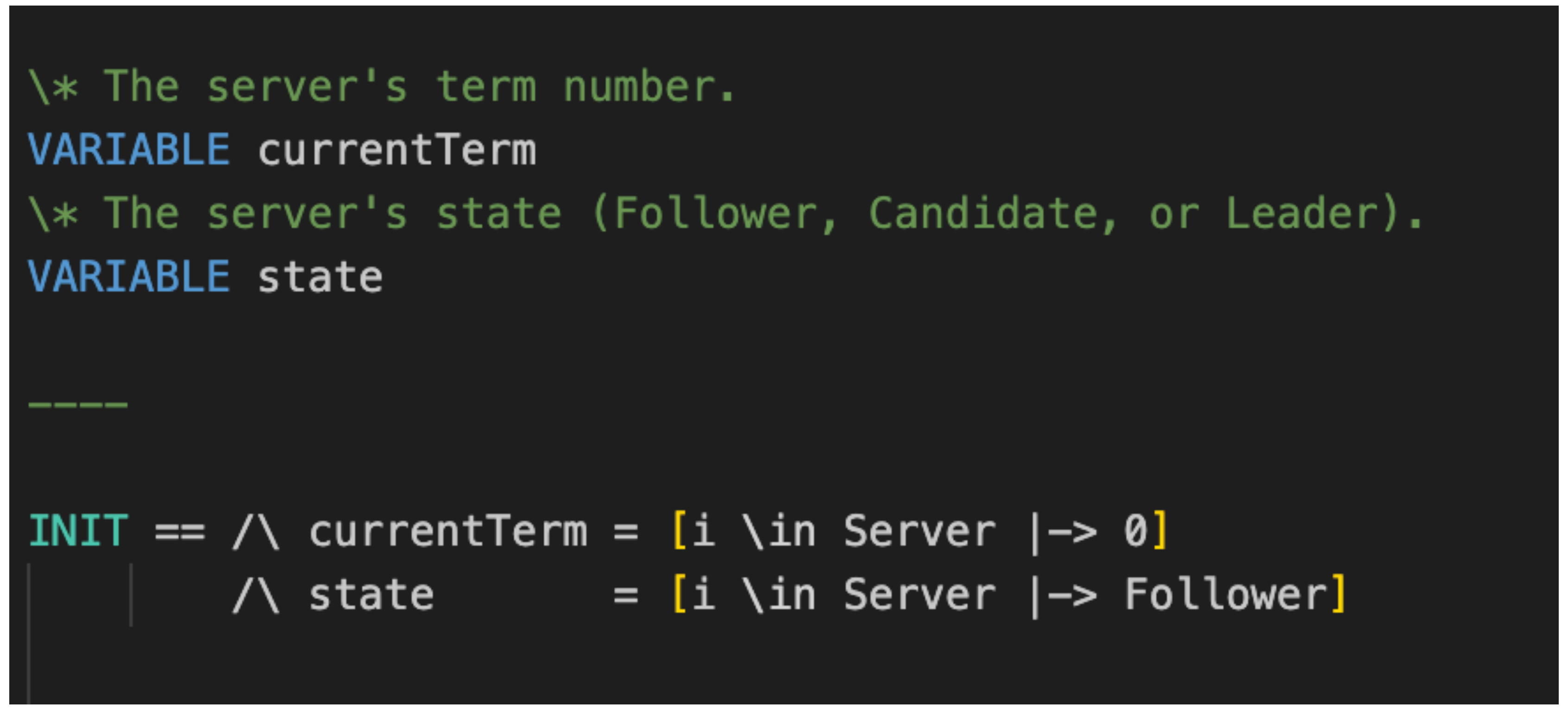
```
HandleRequestVoteResponse(i, j, m)
```



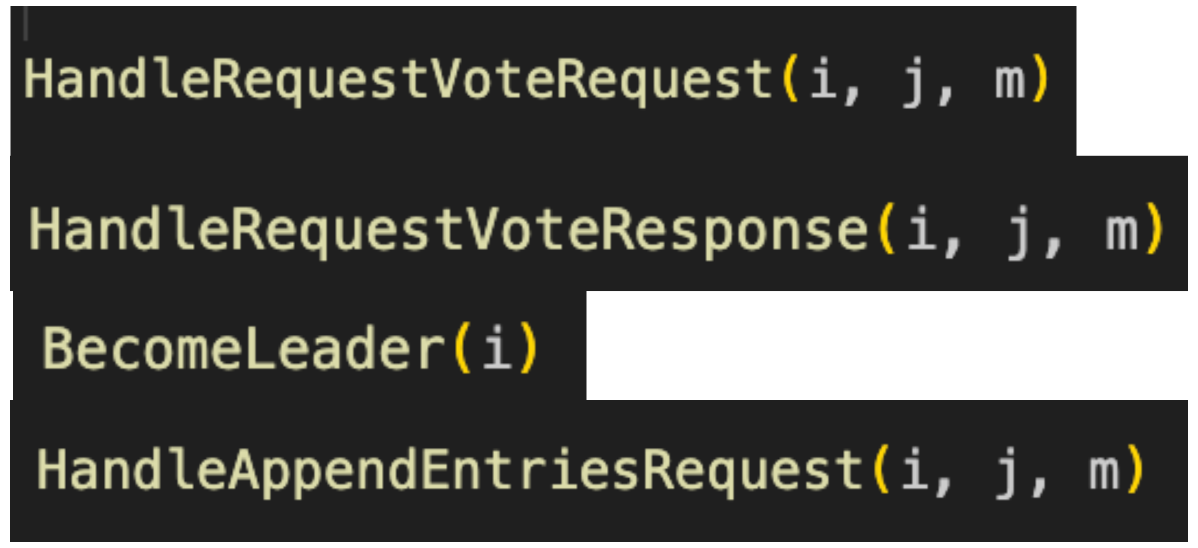
```
BecomeLeader(i)
```



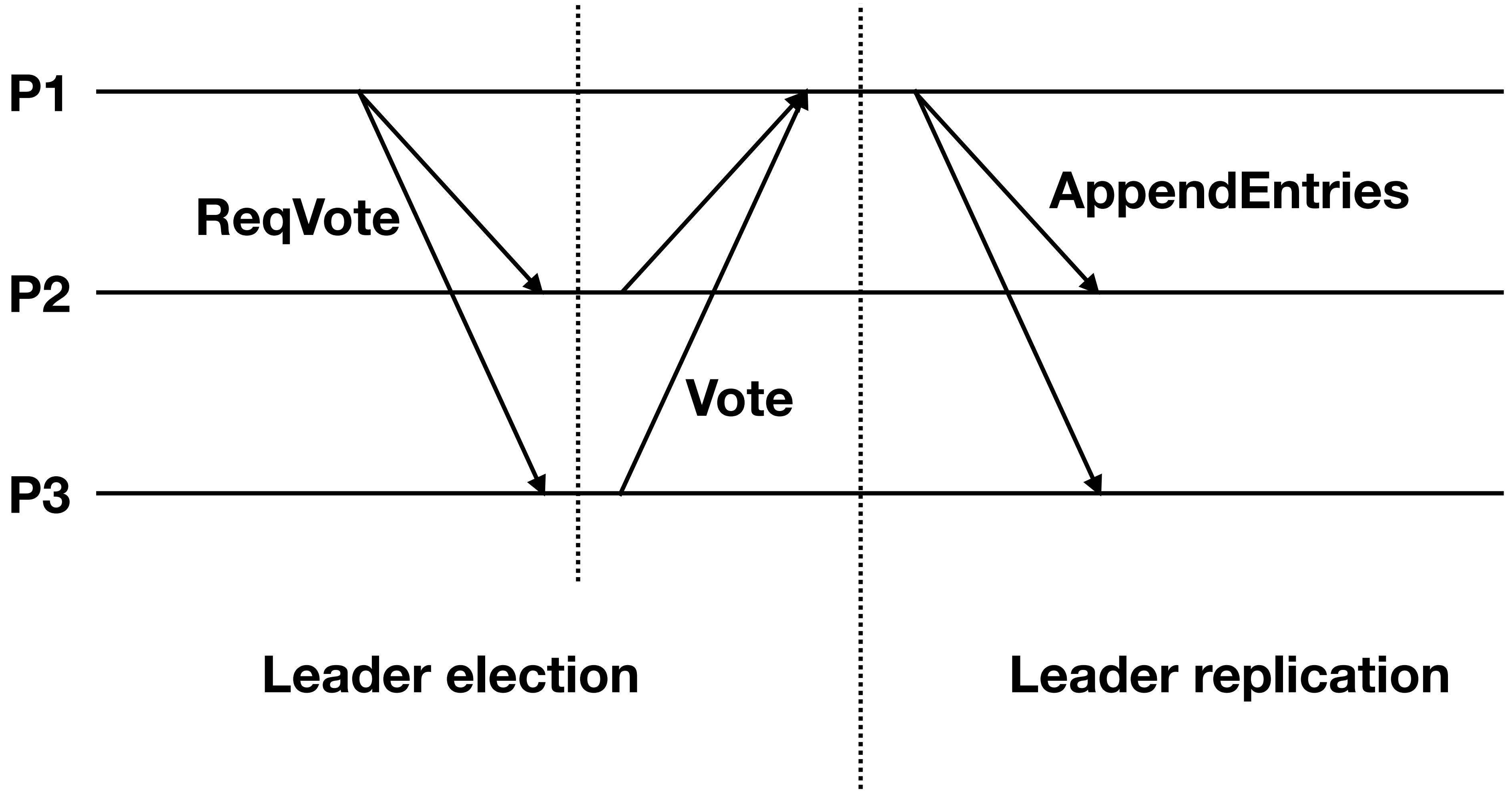
Raft TLA



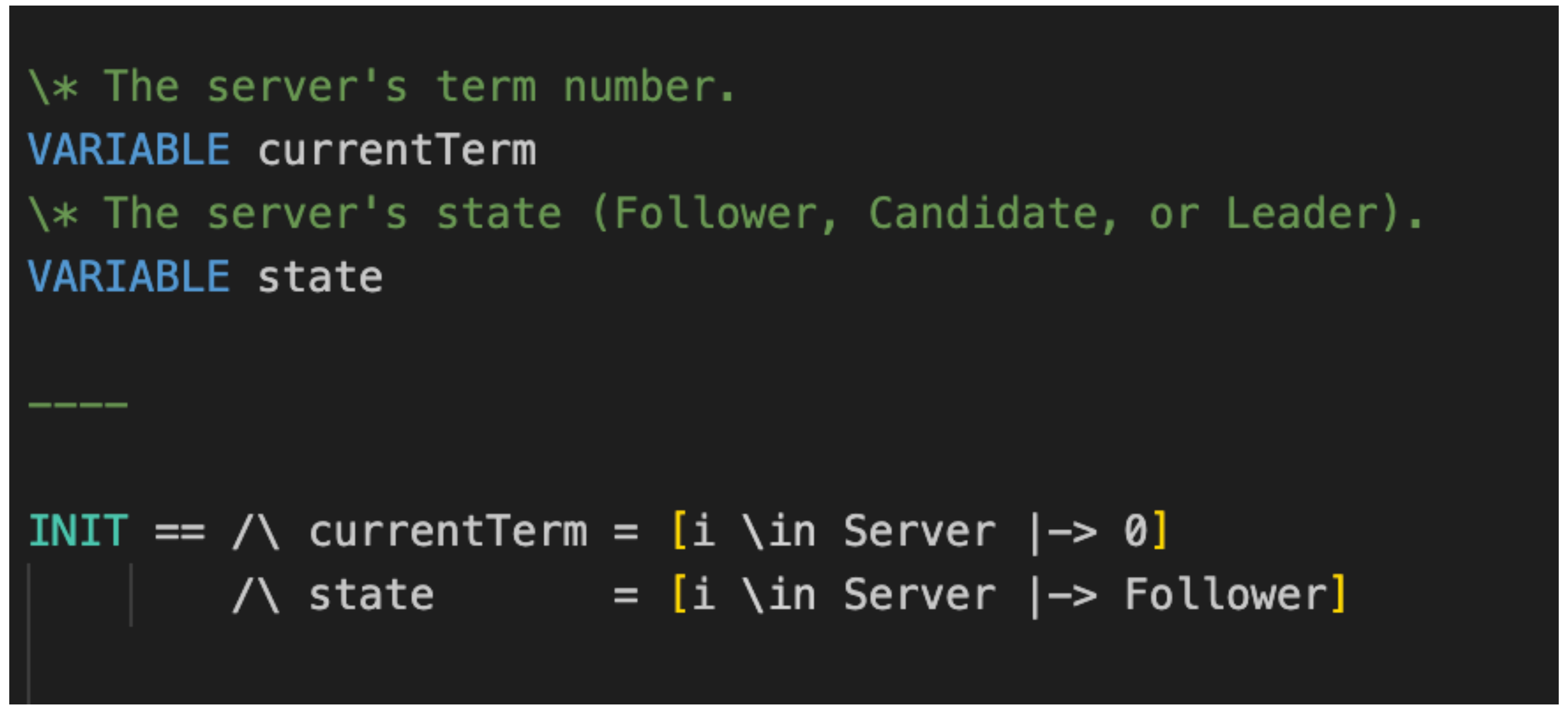
```
\* The server's term number.
VARIABLE currentTerm
\* The server's state (Follower, Candidate, or Leader).
VARIABLE state

----

INIT == /\ currentTerm = [i \in Server |-> 0]
        /\ state       = [i \in Server |-> Follower]
```



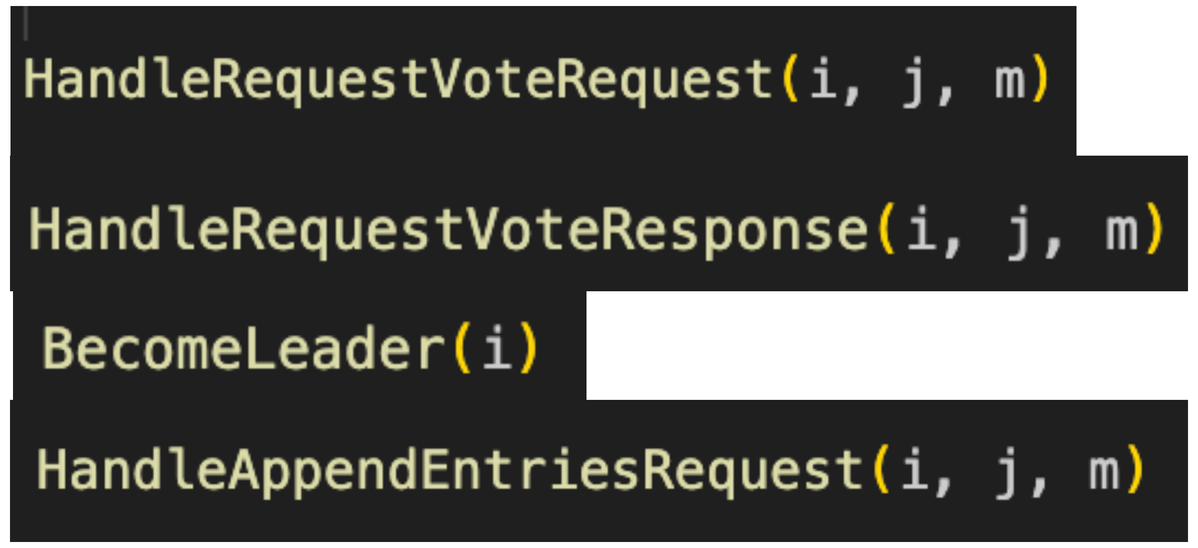
```
HandleRequestVoteRequest(i, j, m)

HandleRequestVoteResponse(i, j, m)

BecomeLeader(i)

HandleAppendEntriesRequest(i, j, m)
```



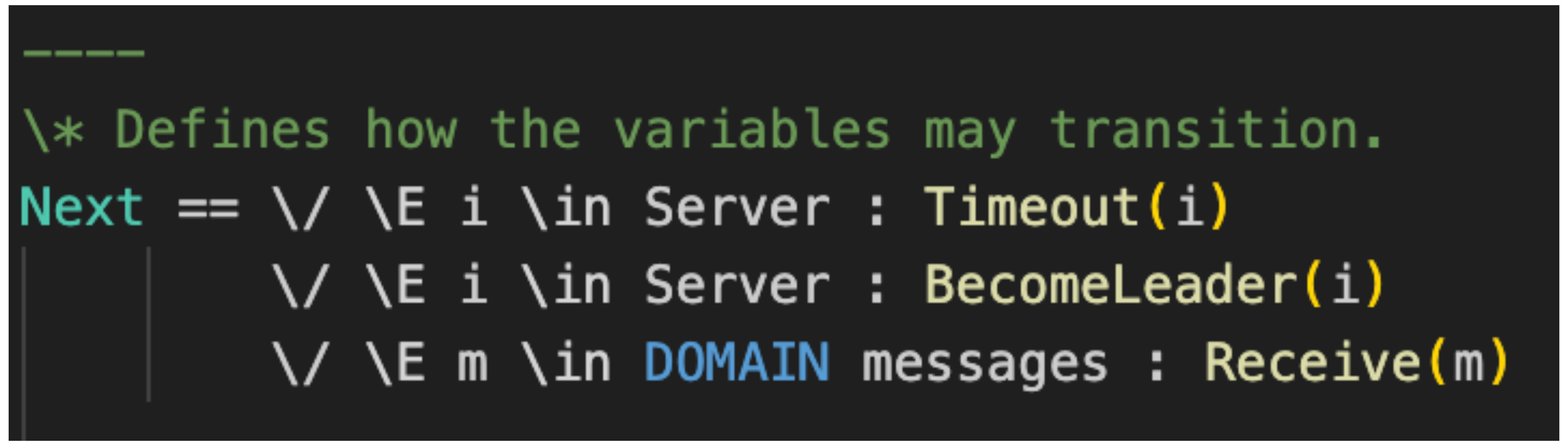
Raft TLA



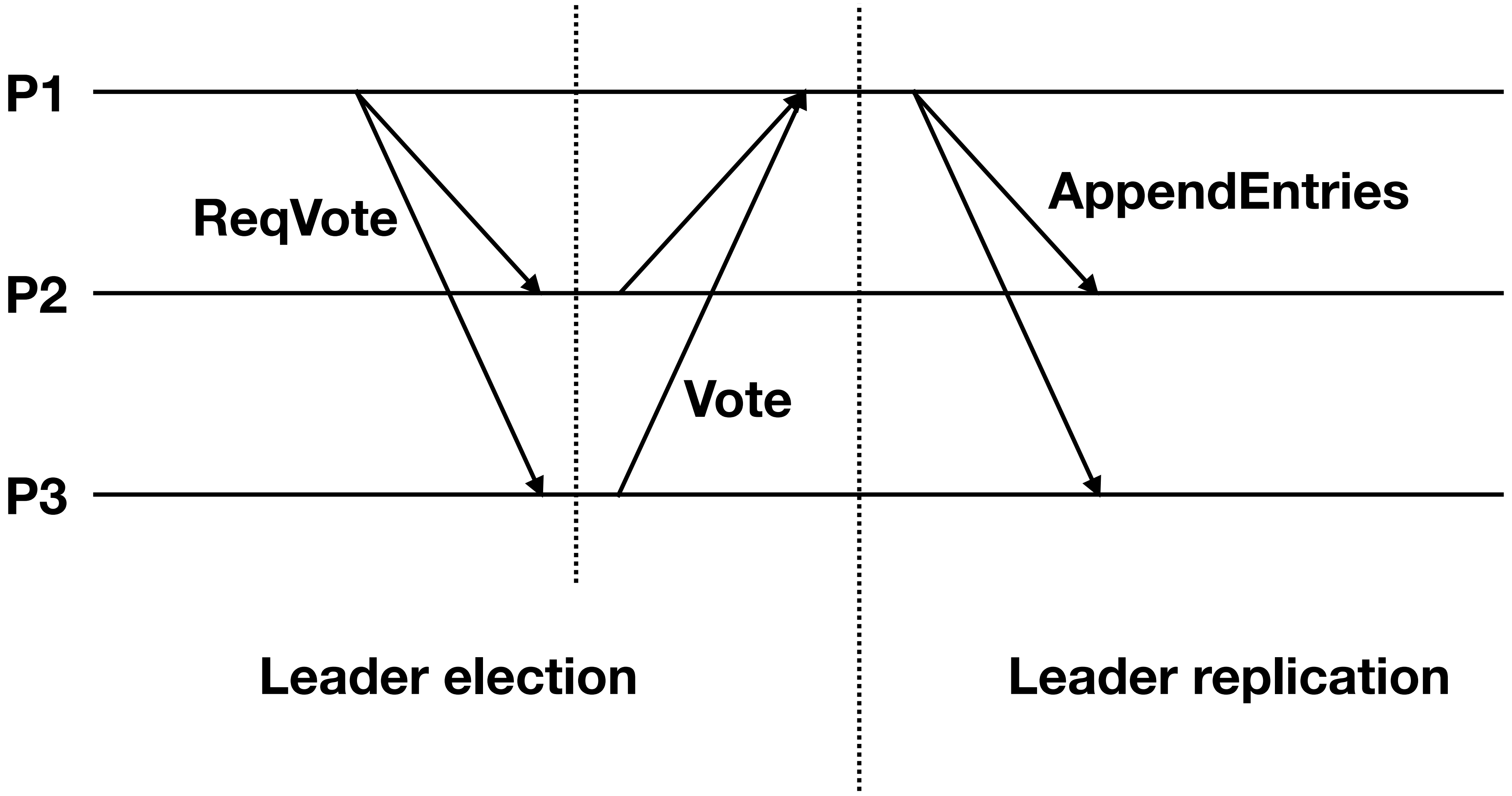
```
\* The server's term number.
VARIABLE currentTerm
\* The server's state (Follower, Candidate, or Leader).
VARIABLE state

----

INIT == /\ currentTerm = [i \in Server |-> 0]
        /\ state       = [i \in Server |-> Follower]
```



```
HandleRequestVoteRequest(i, j, m)

HandleRequestVoteResponse(i, j, m)

BecomeLeader(i)

HandleAppendEntriesRequest(i, j, m)
```

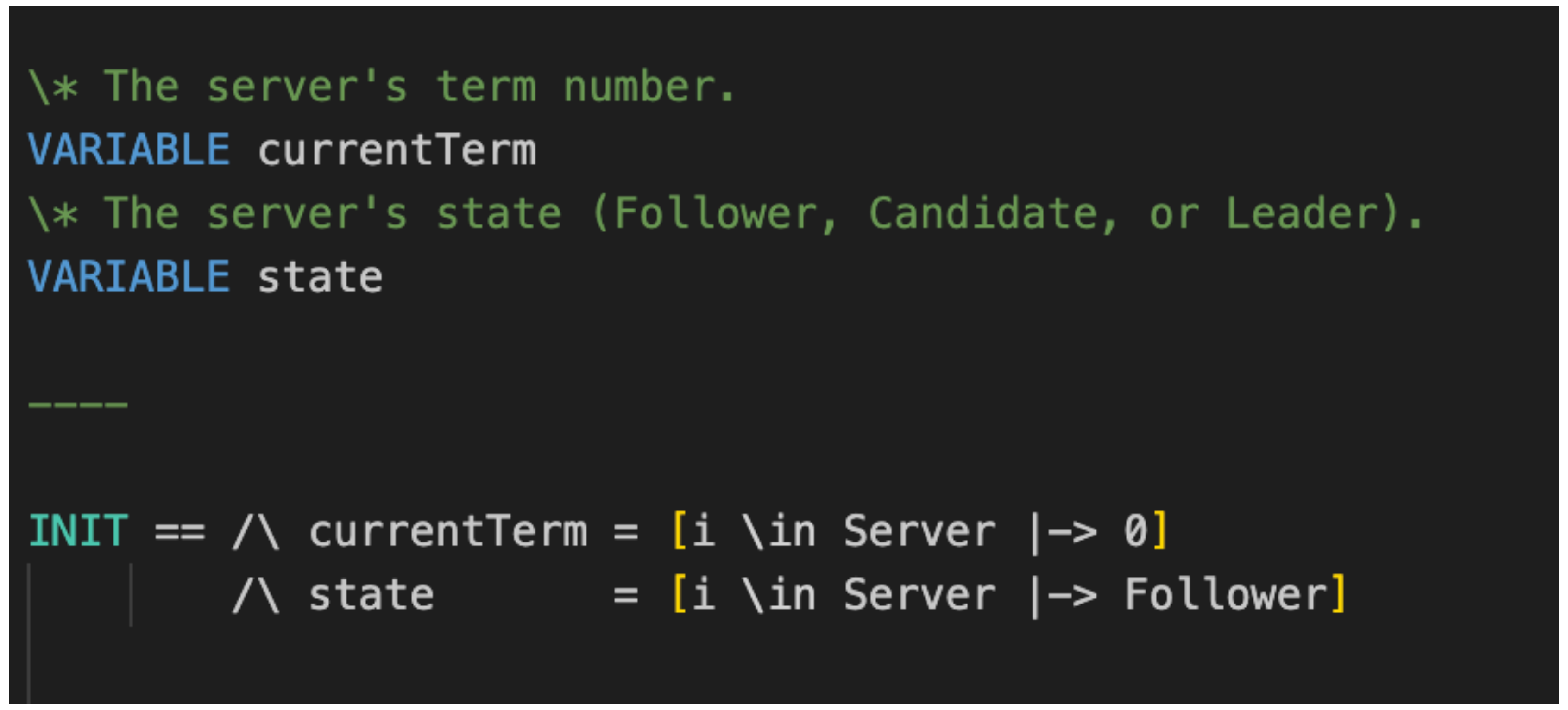


```
----
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E m \in DOMAIN messages : Receive(m)
```

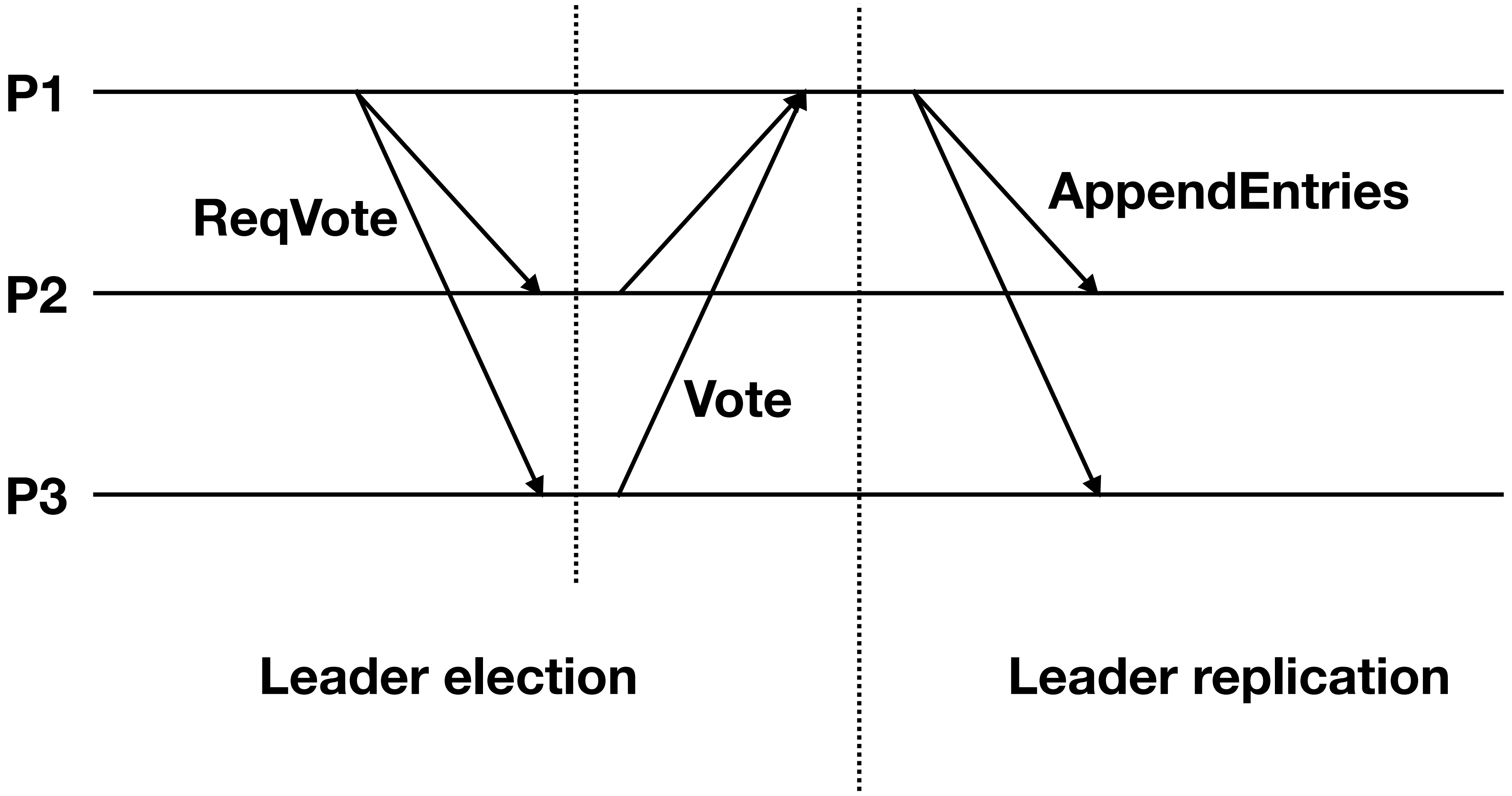


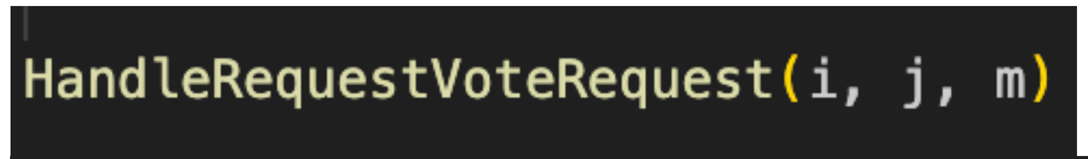
Raft TLA

State

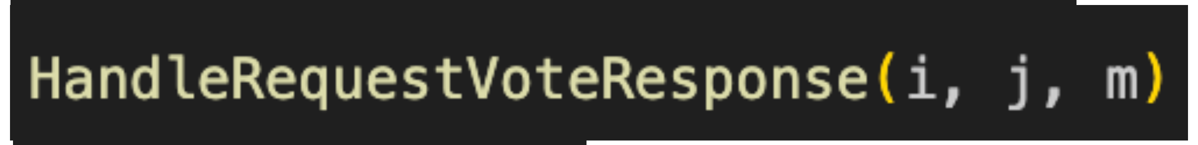


```
\* The server's term number.
VARIABLE currentTerm
\* The server's state (Follower, Candidate, or Leader).
VARIABLE state

----

INIT == /\ currentTerm = [i \in Server |-> 0]
        /\ state       = [i \in Server |-> Follower]
```

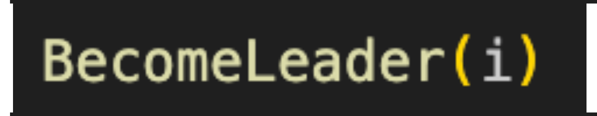




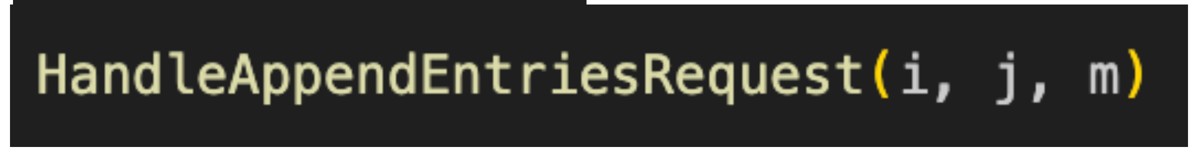
```
HandleRequestVoteRequest(i, j, m)
```



```
HandleRequestVoteResponse(i, j, m)
```

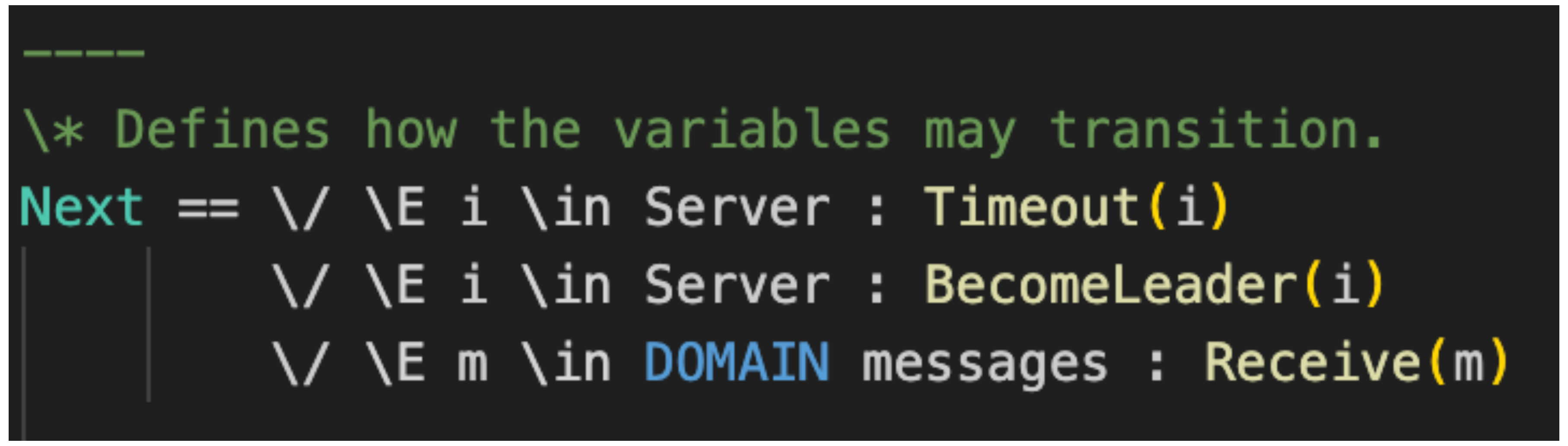


```
BecomeLeader(i)
```



```
HandleAppendEntriesRequest(i, j, m)
```

Actions

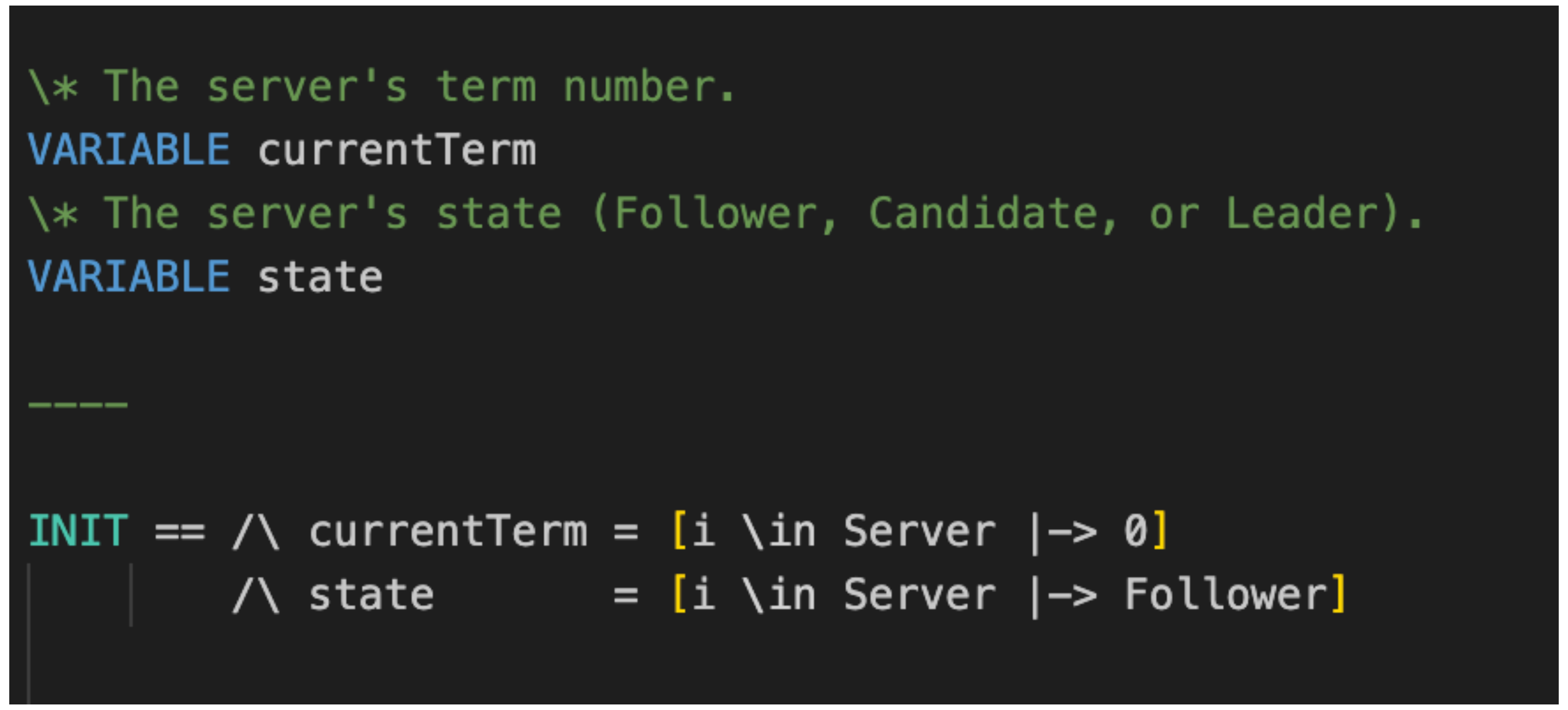


```
-----
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E m \in DOMAIN messages : Receive(m)
```

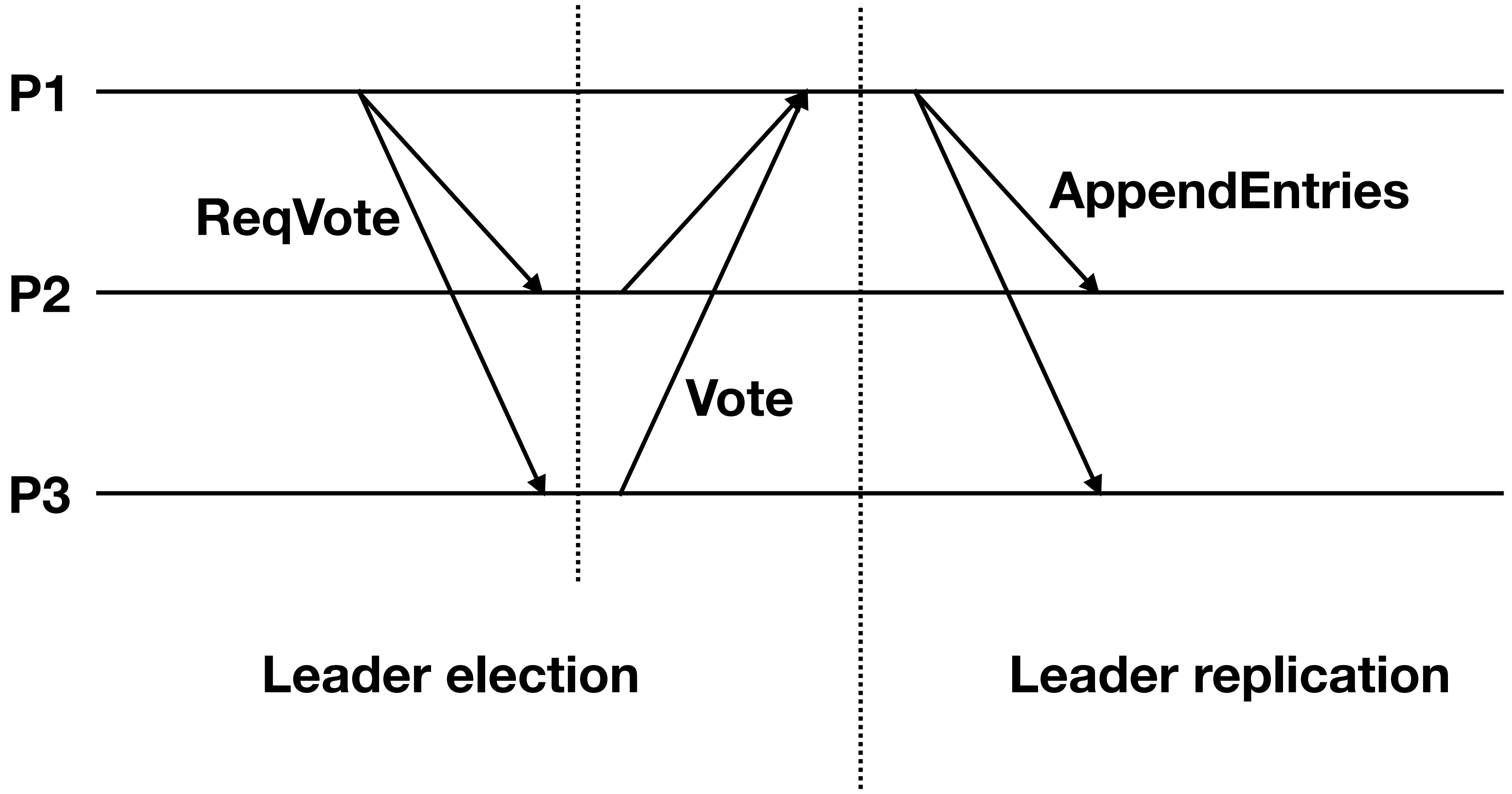
Transition relation

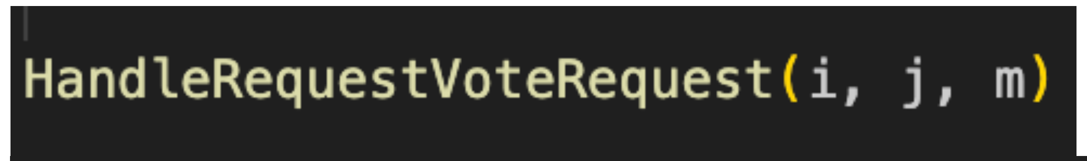
Raft TLA

State

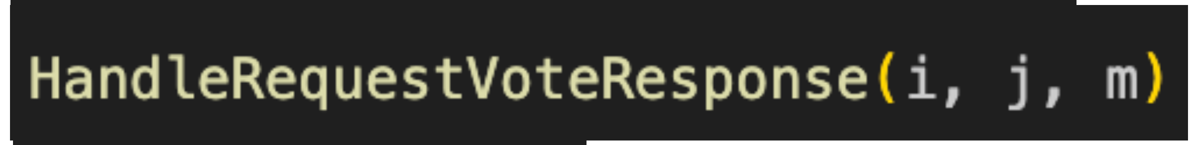


```
\* The server's term number.
VARIABLE currentTerm
\* The server's state (Follower, Candidate, or Leader).
VARIABLE state

----

INIT == /\ currentTerm = [i \in Server |-> 0]
        /\ state       = [i \in Server |-> Follower]
```

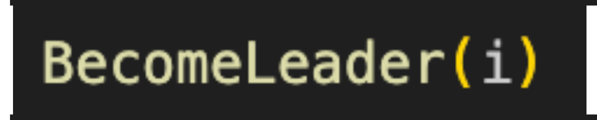




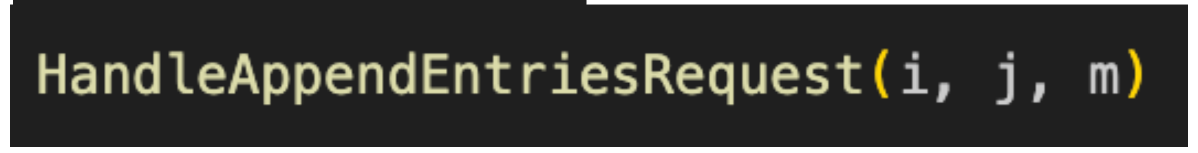
```
HandleRequestVoteRequest(i, j, m)
```



```
HandleRequestVoteResponse(i, j, m)
```

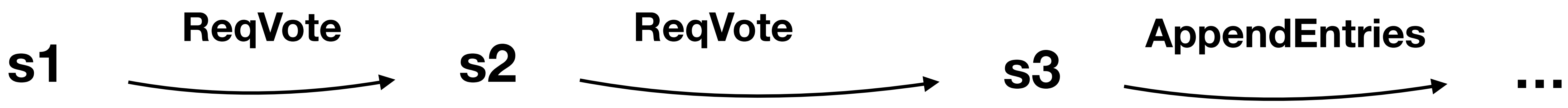


```
BecomeLeader(i)
```

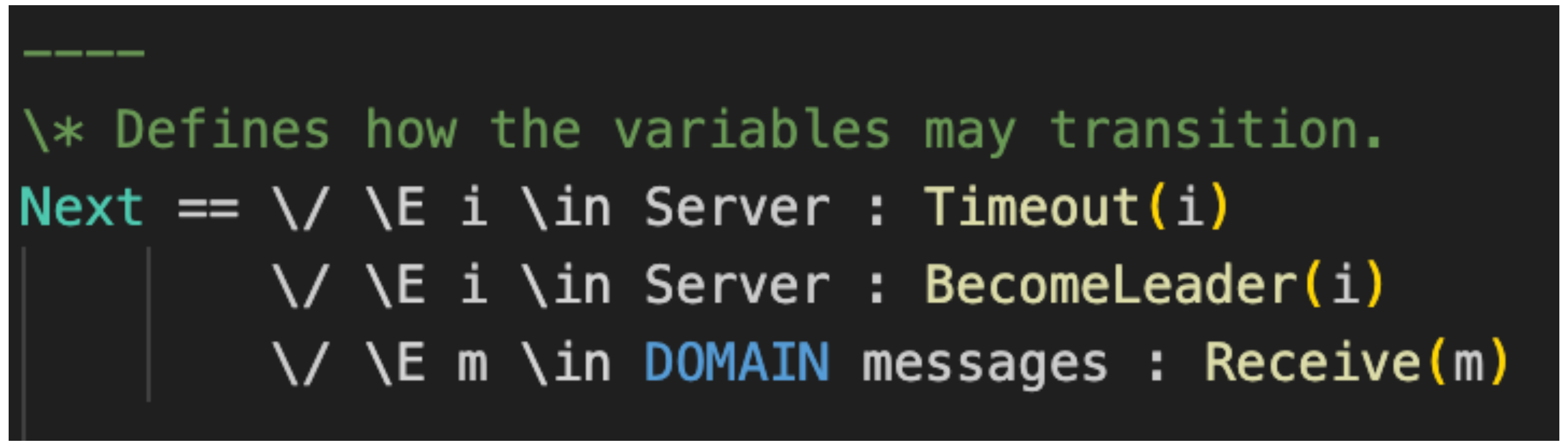


```
HandleAppendEntriesRequest(i, j, m)
```

Actions



s1 ReqVote s2 ReqVote s3 AppendEntries ...



```
----
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E m \in DOMAIN messages : Receive(m)
```

Transition relation

Model based testing

Model based testing

Why not just enumerate all executions from the model?

Model based testing

Why not just enumerate all executions from the model?

1. Too many executions - state explosion

Model based testing

Why not just enumerate all executions from the model?

1. Too many executions - state explosion

2. Too much instrumentation effort - per message annotations in the code

Model based testing

Why not just enumerate all executions from the model?

1. Too many executions - state explosion

2. Too much instrumentation effort - per message annotations in the code

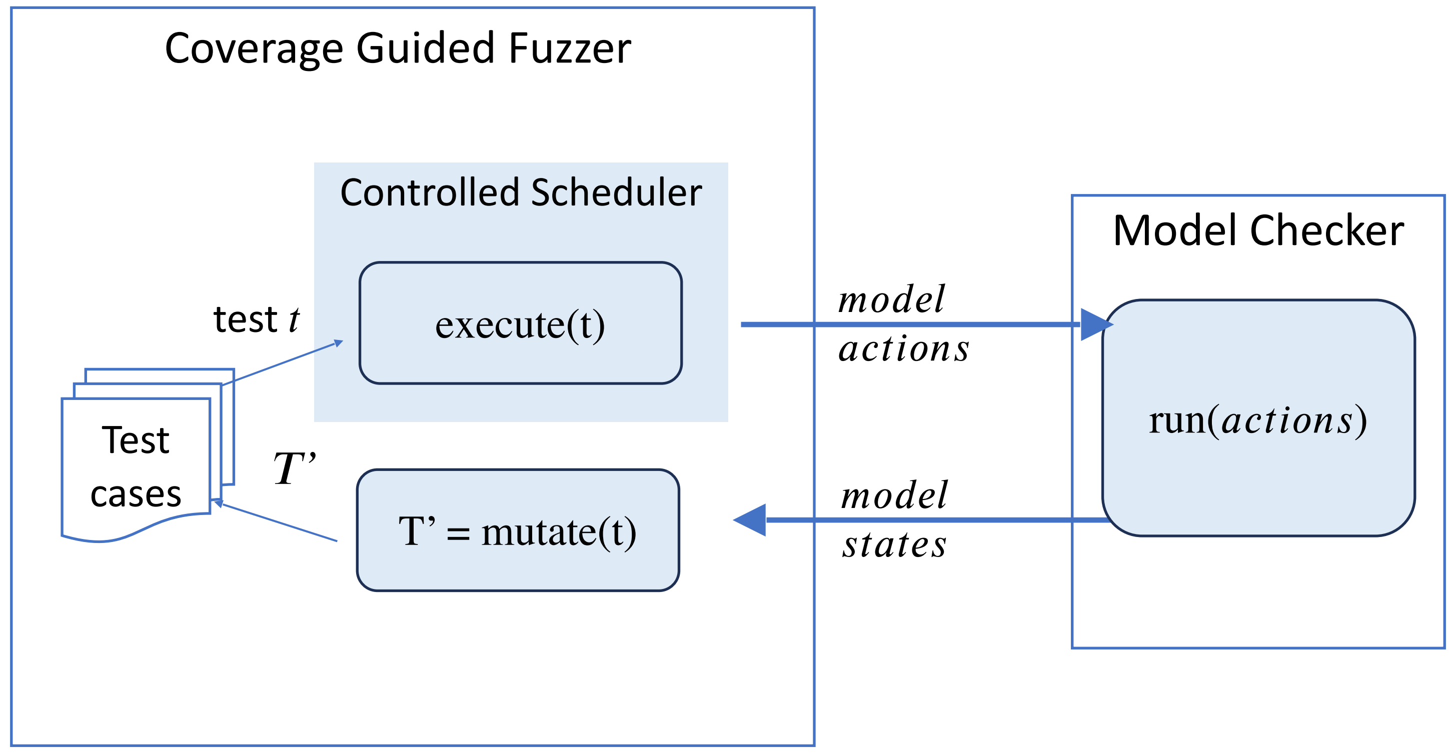
3. Model ignores implementation optimisations. E.g. Snapshots

Our approach - ModelFuzz

ModelFuzz

ModelFuzz

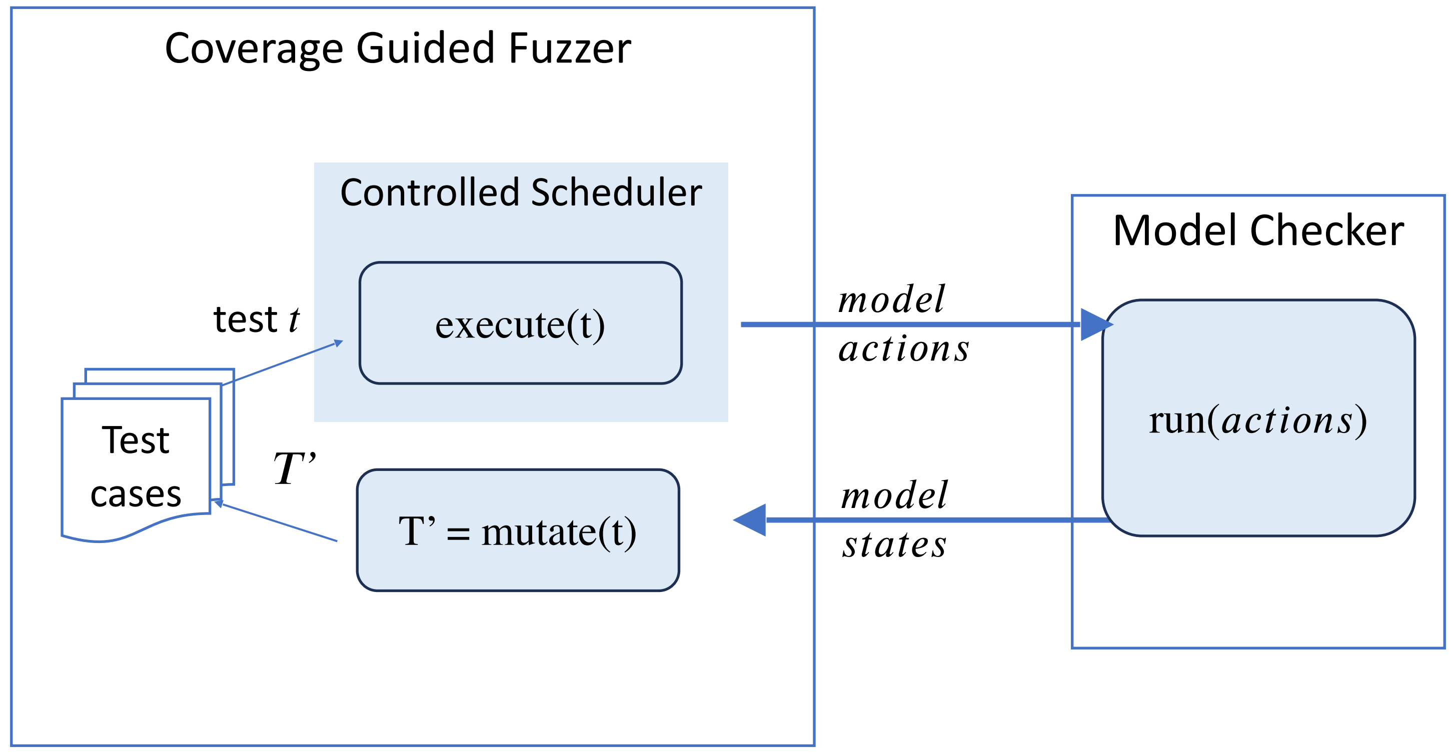
- Randomly sample implementation *test cases*



ModelFuzz

- Randomly sample implementation *test cases*

- Simulate them on the model

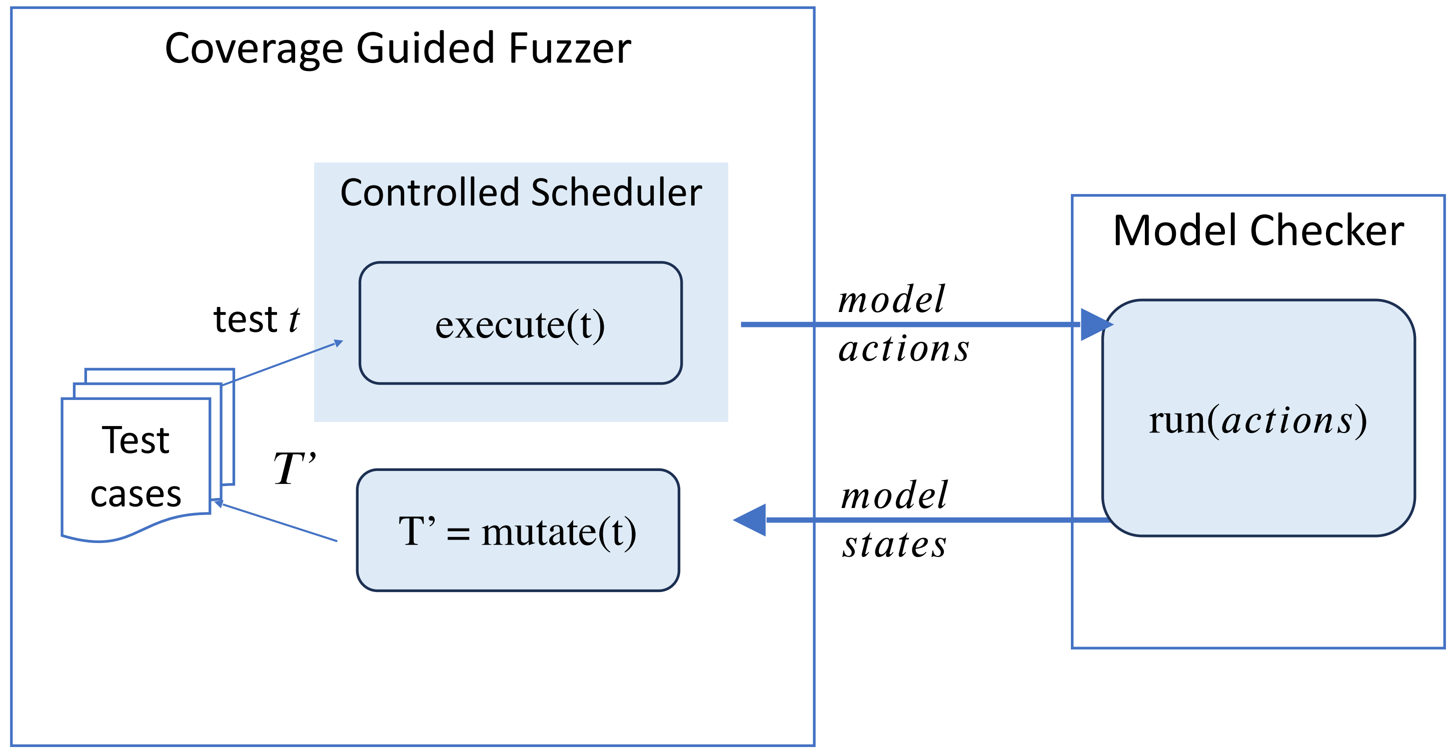


ModelFuzz

- Randomly sample implementation *test cases*

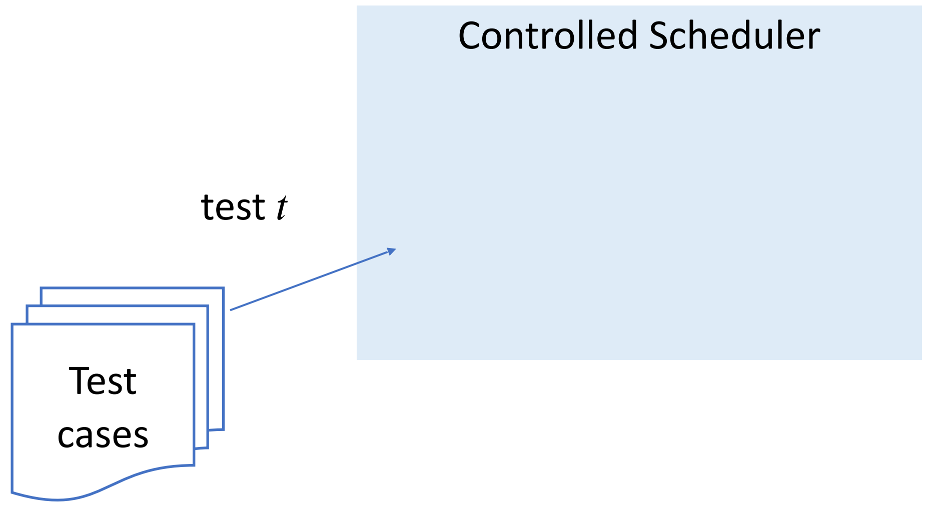
- Simulate them on the model

- Use the coverage information to mutate "interesting" *test cases*



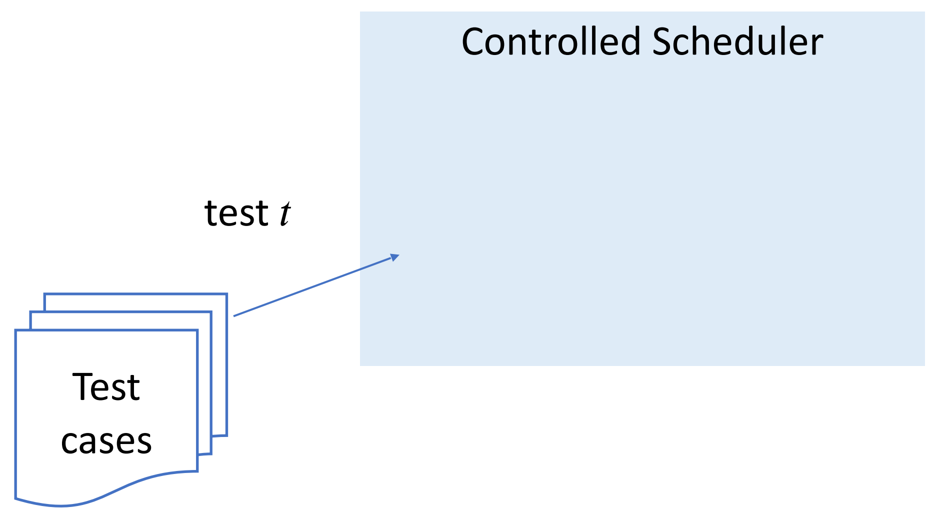
Fuzzer test cases

Fuzzer test cases



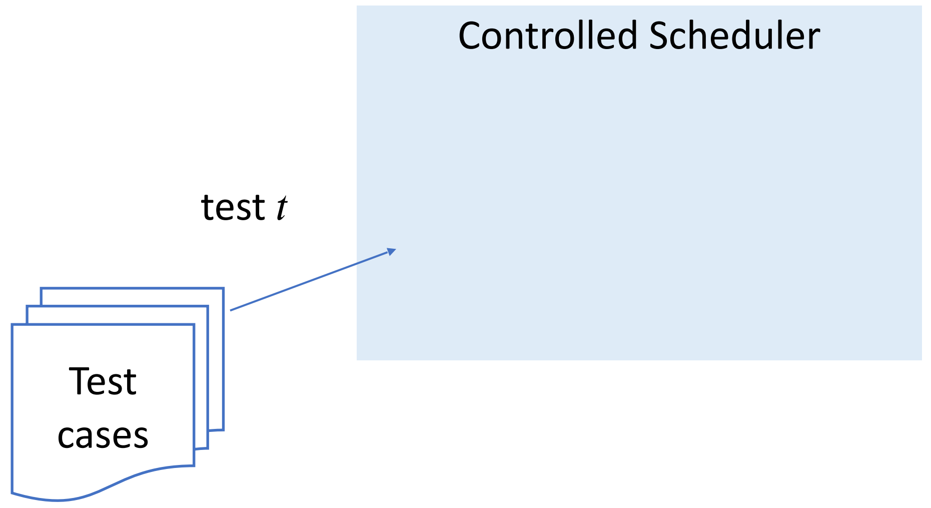
Fuzzer test cases

- Sequence of scheduling choices
  - interleaved with failures



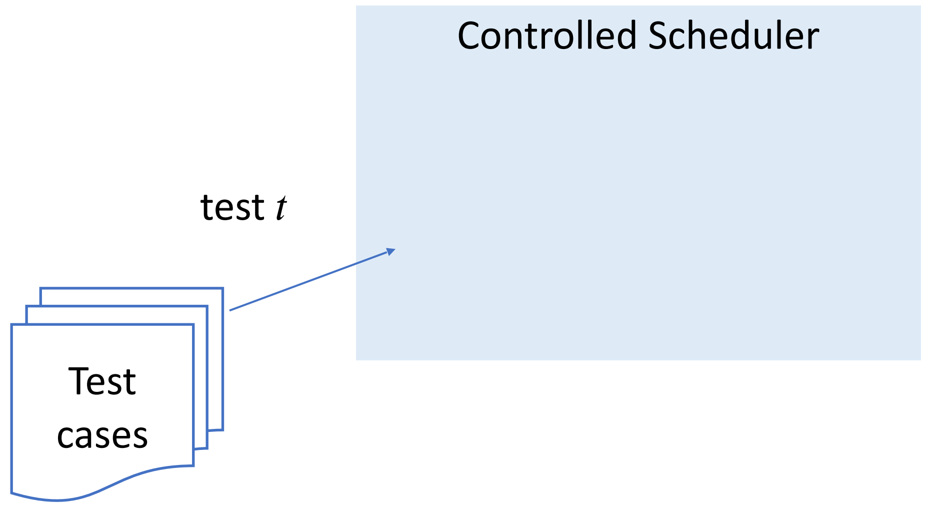
Fuzzer test cases

- Sequence of scheduling choices
    - interleaved with failures
- Deliver(p1,5) . Deliver(p2, 3) . Crash(p1) . Start(p1) . …

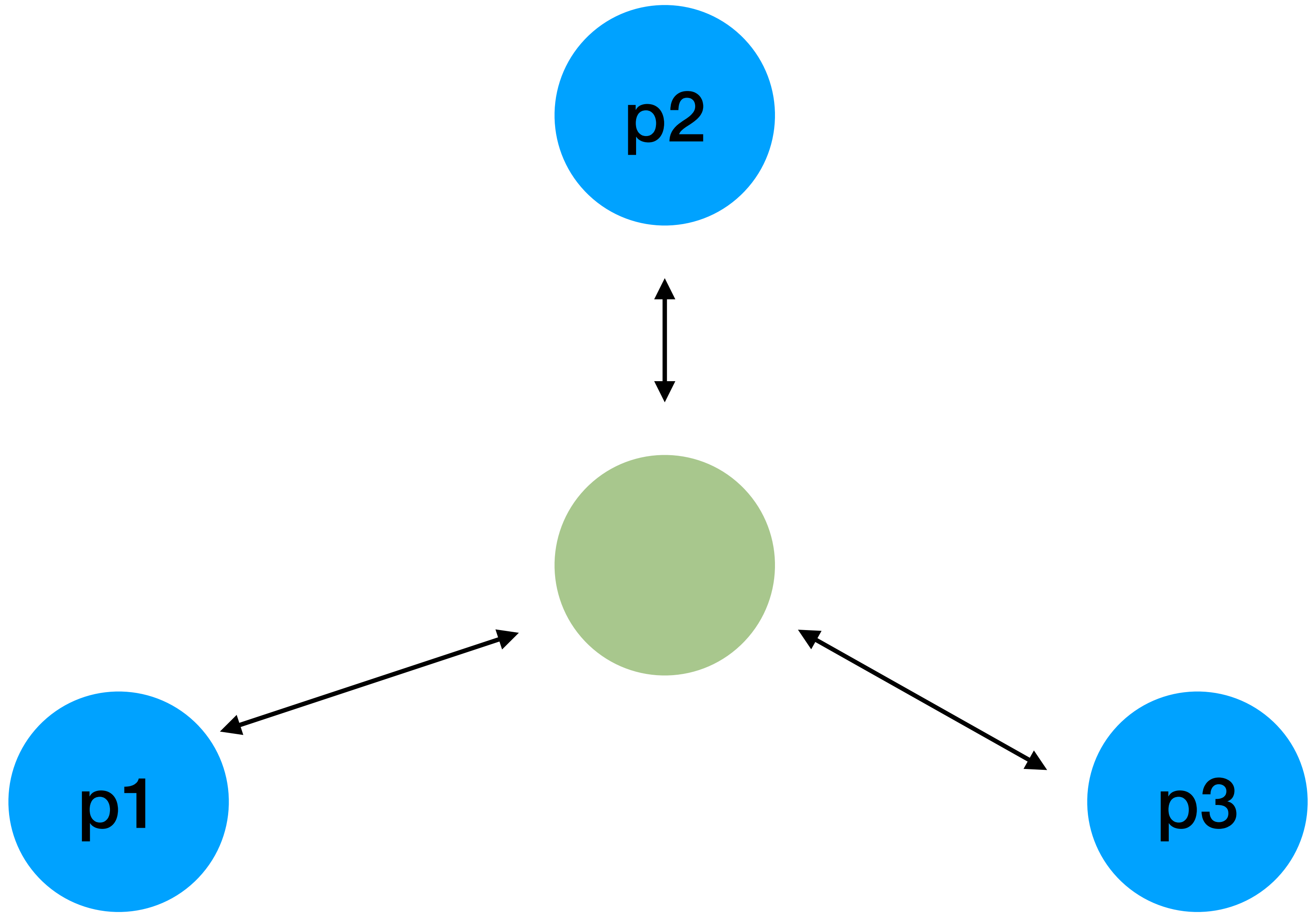


Fuzzer test cases

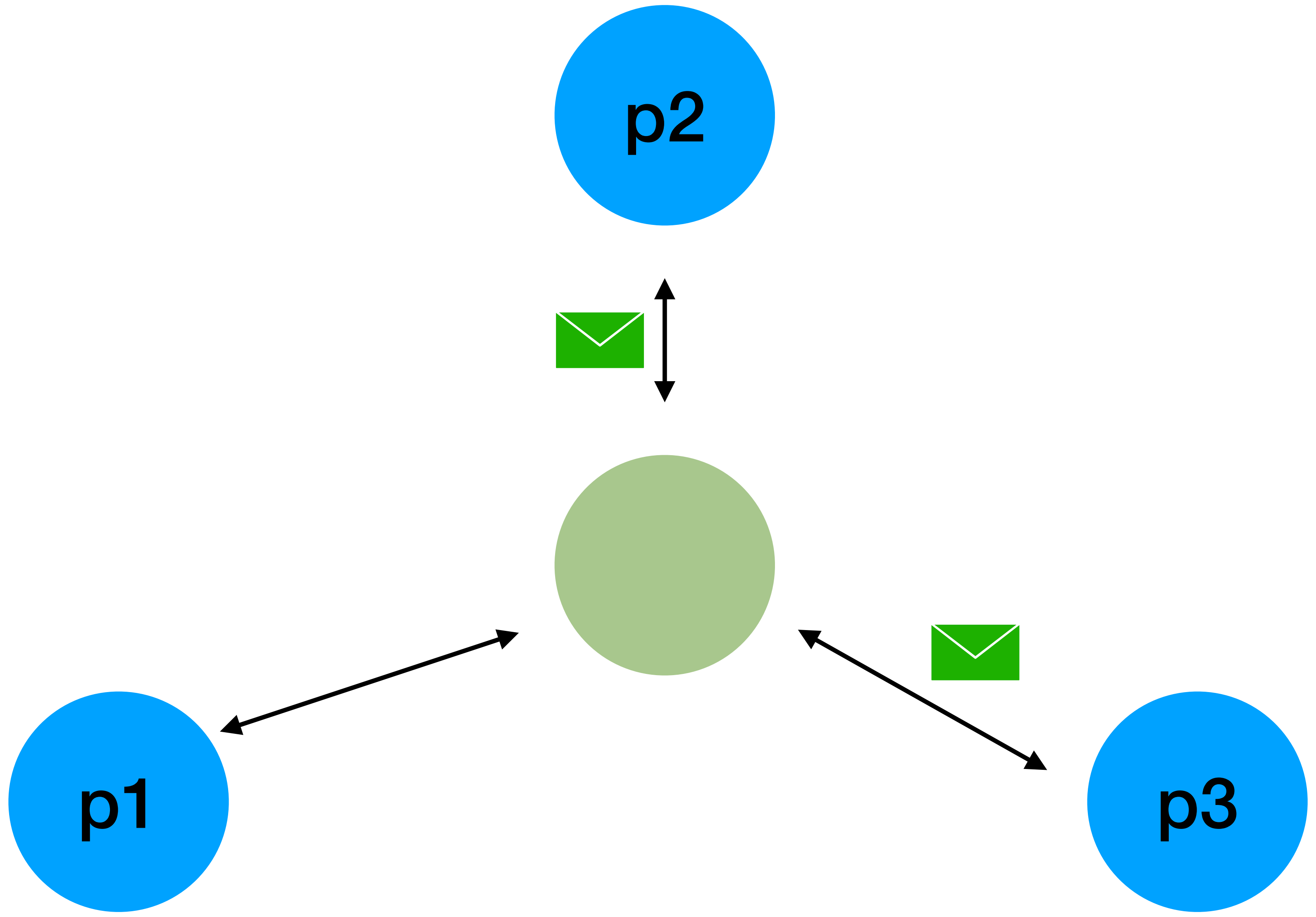
- Sequence of scheduling choices
  - interleaved with failures
- Deliver(p1,5) . Deliver(p2, 3) . Crash(p1) . Start(p1) . …
- Why not messages? Not all inputs are *valid*
  - Non leader cannot send AppendEntries



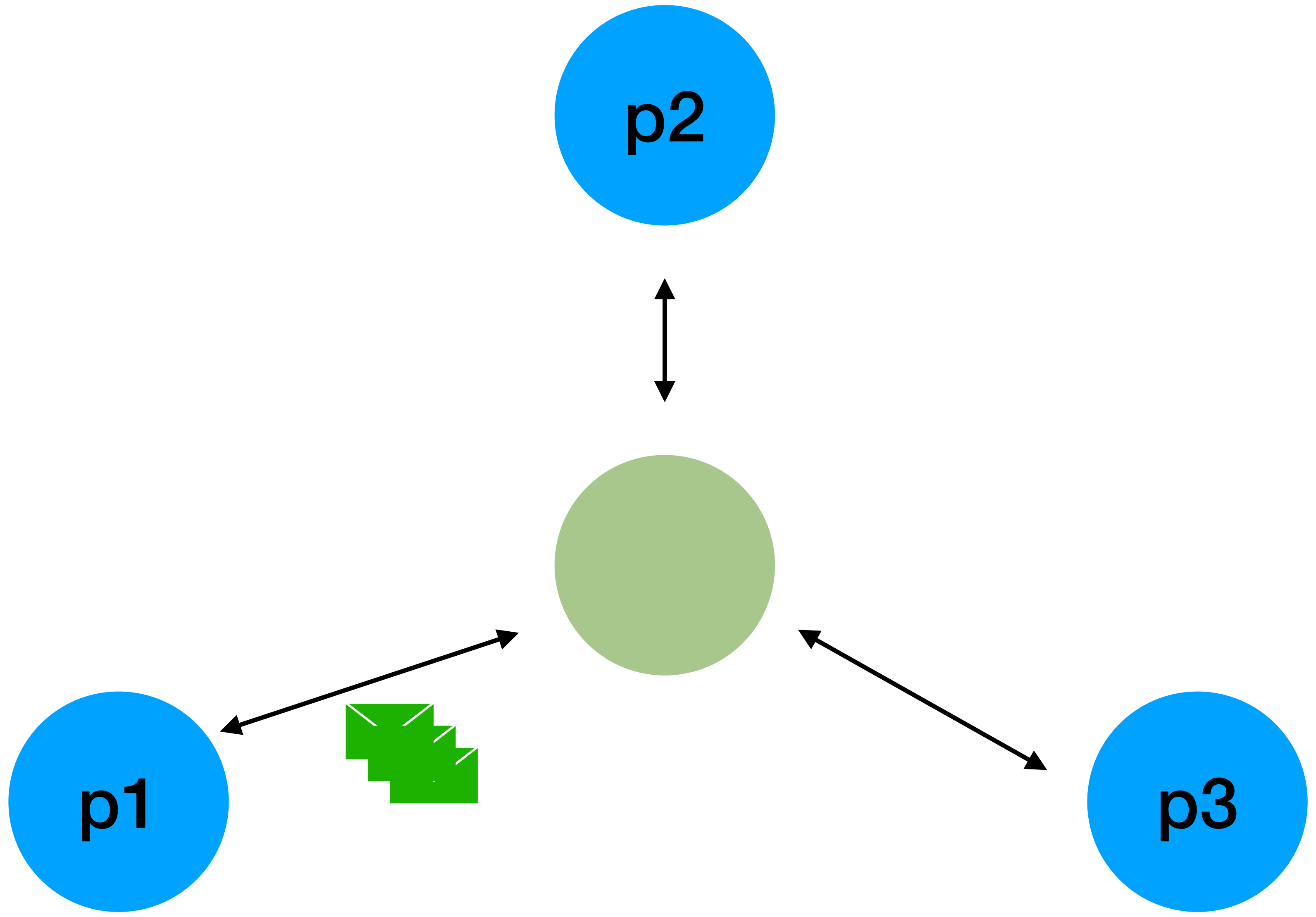
Semantics



Semantics

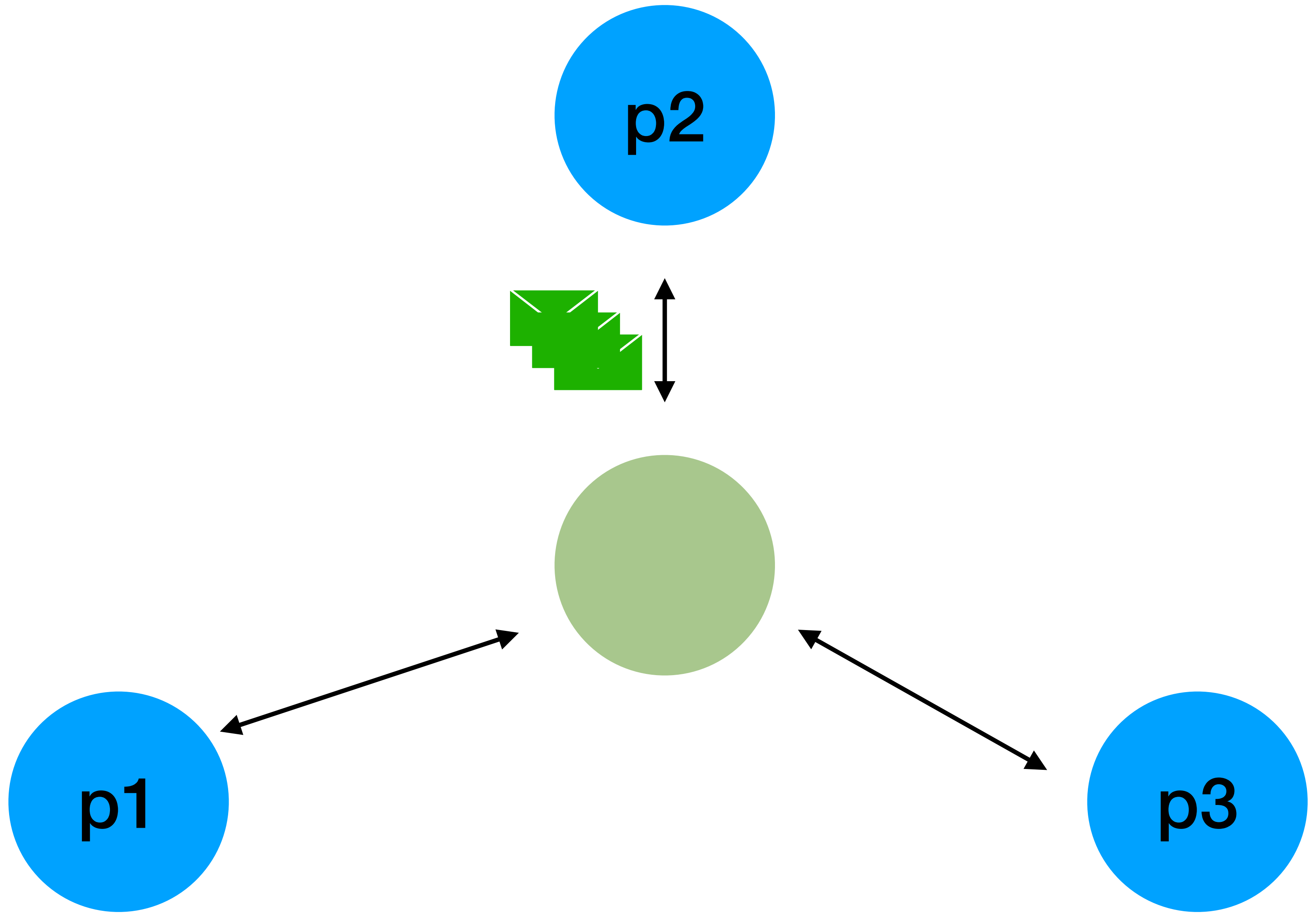


Semantics



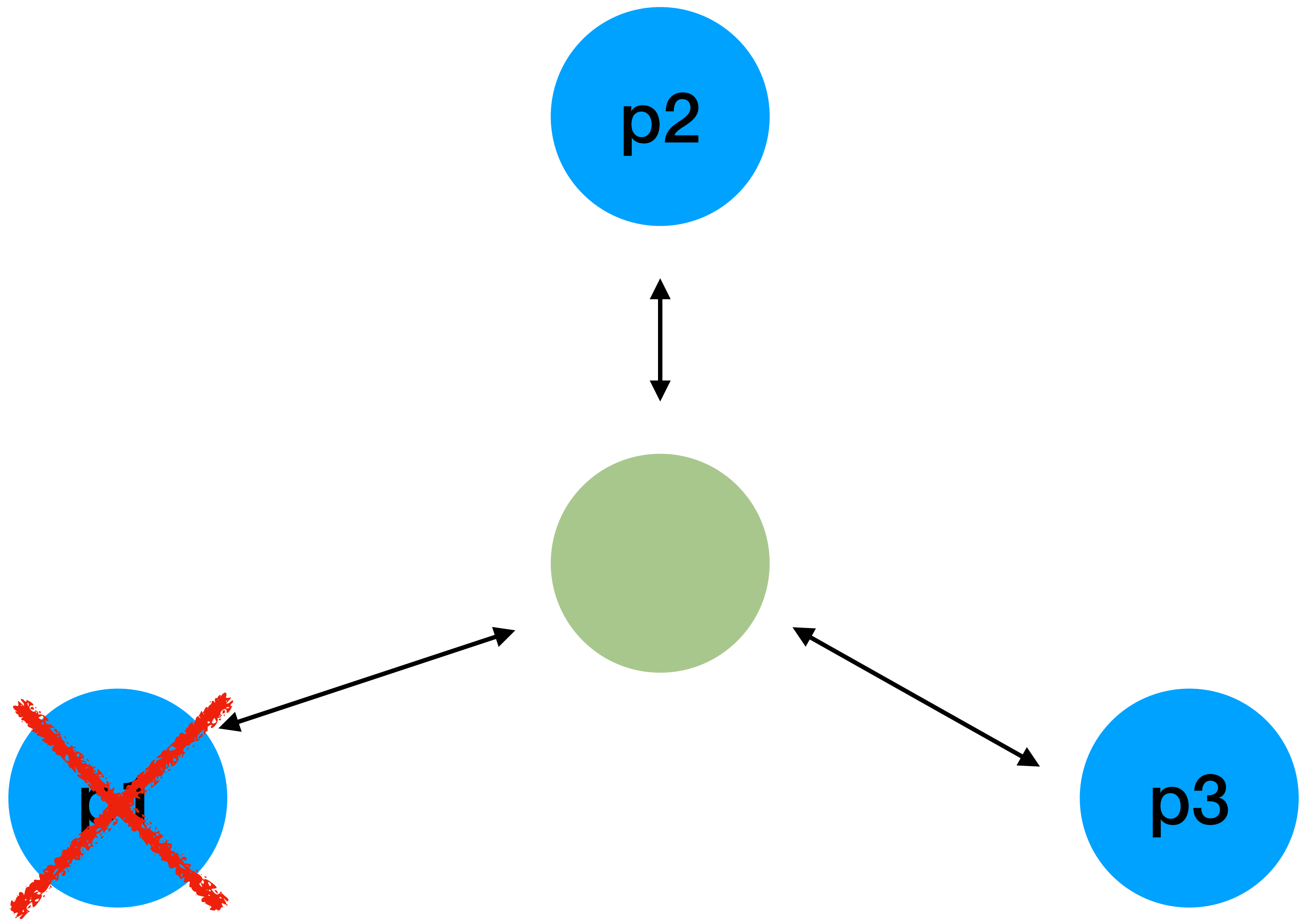
Deliver(p1,5) .

Semantics



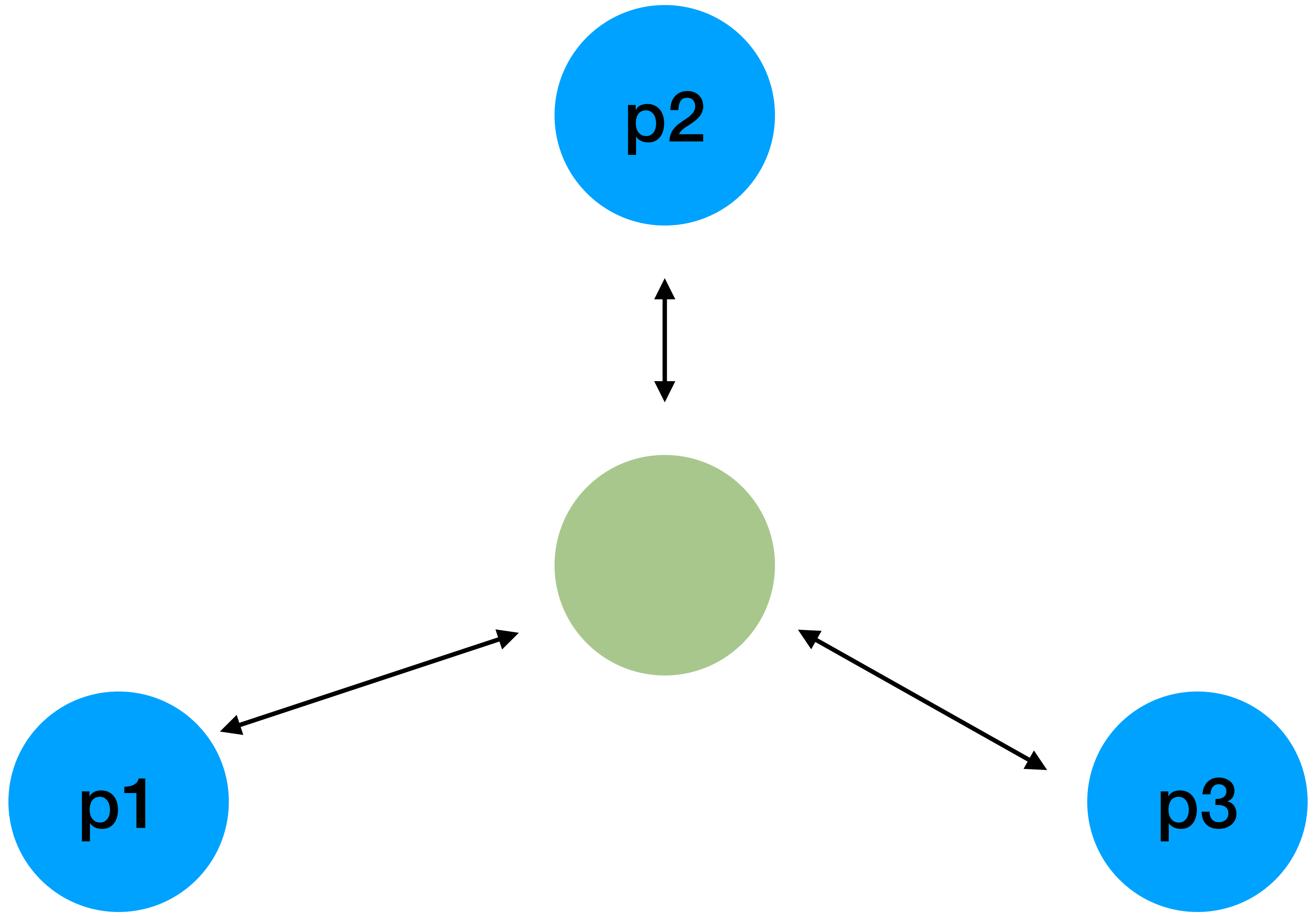
Deliver(p1,5) . Deliver(p2, 3) .

Semantics



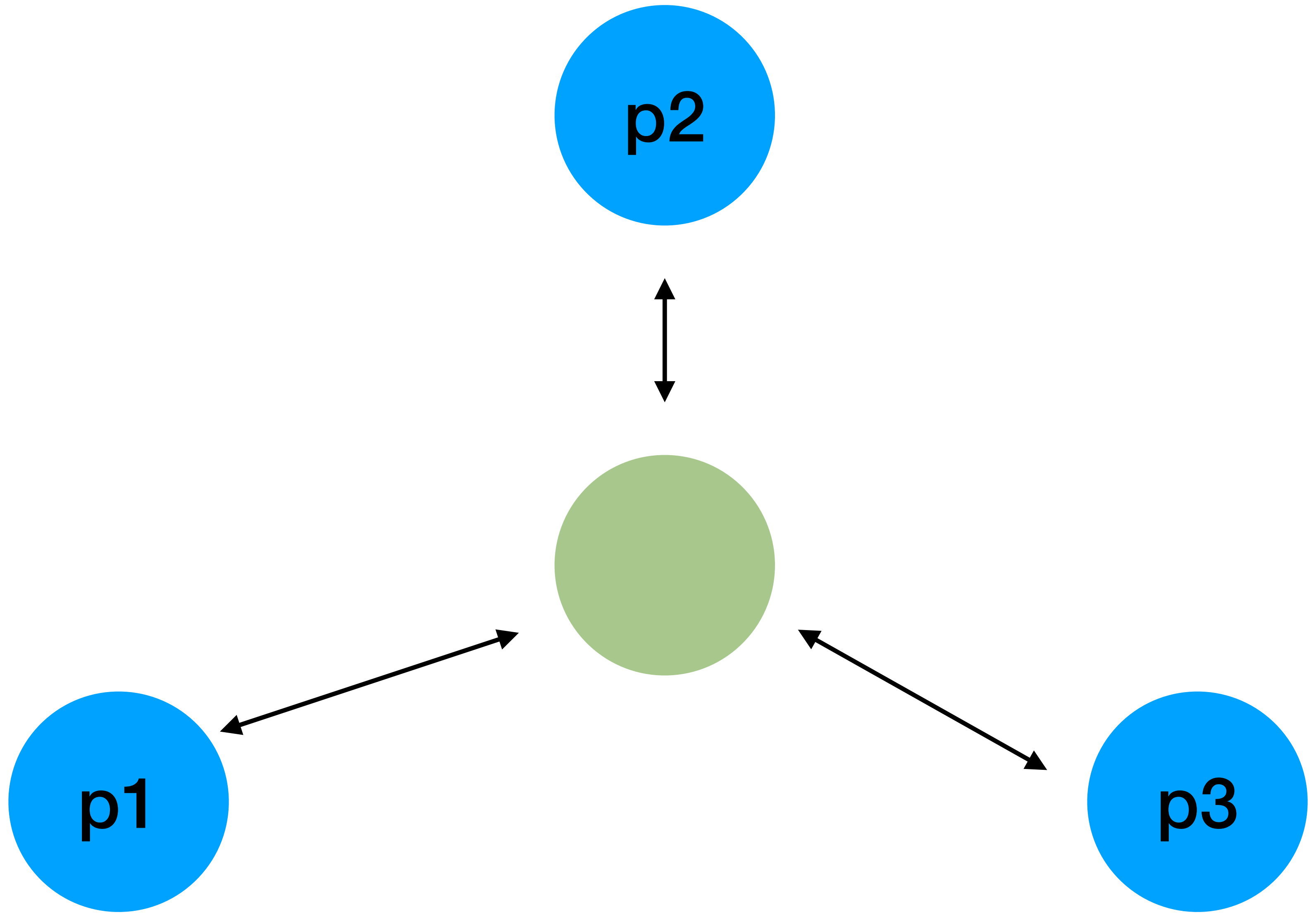
Deliver(p1,5) . Deliver(p2, 3) . Crash(p1) .

Semantics



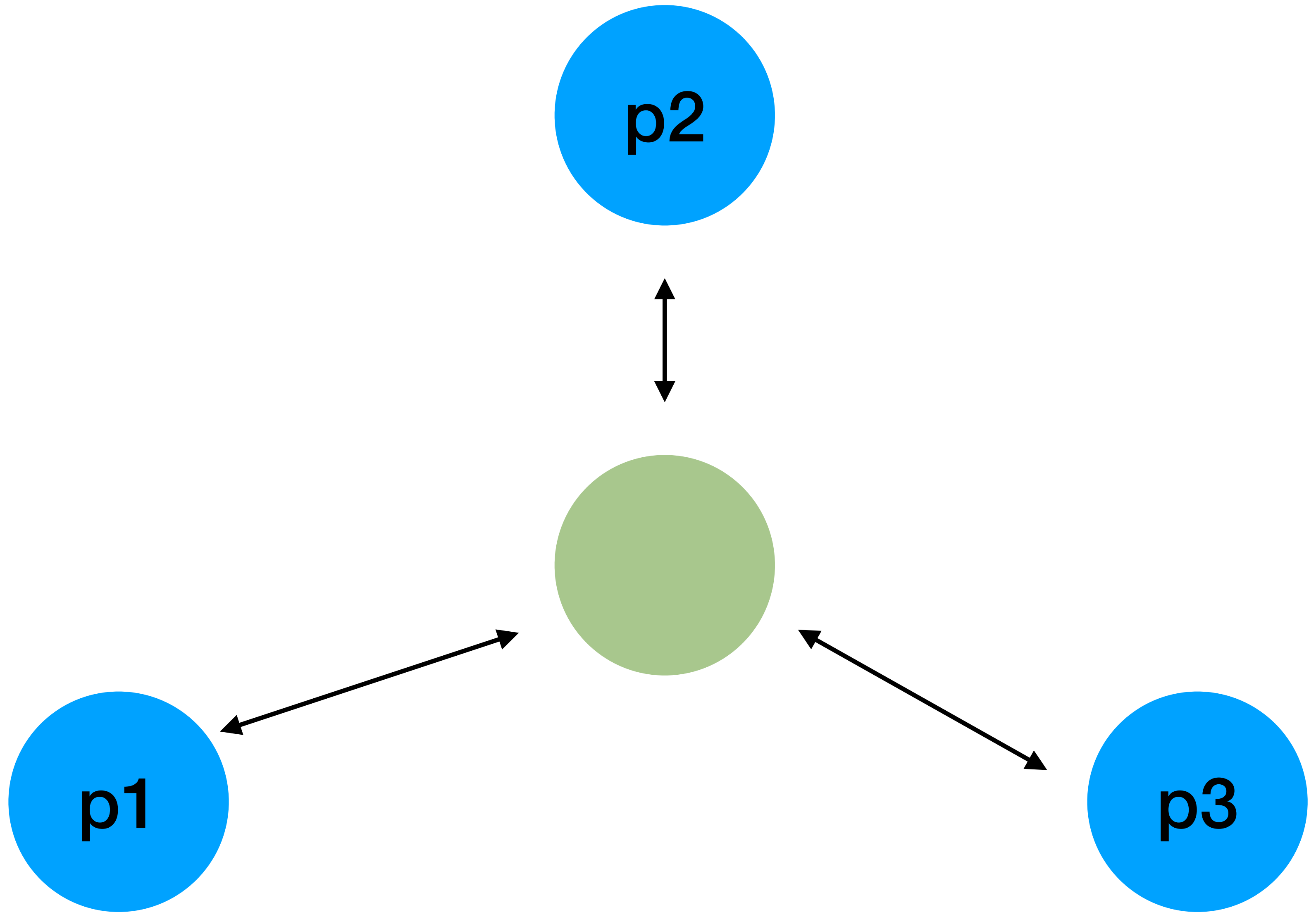
Deliver(p1,5) . Deliver(p2, 3) . Crash(p1) . Start(p1) .

Semantics



Deliver(p1,5) . Deliver(p2, 3) . Crash(p1) . Start(p1) . …

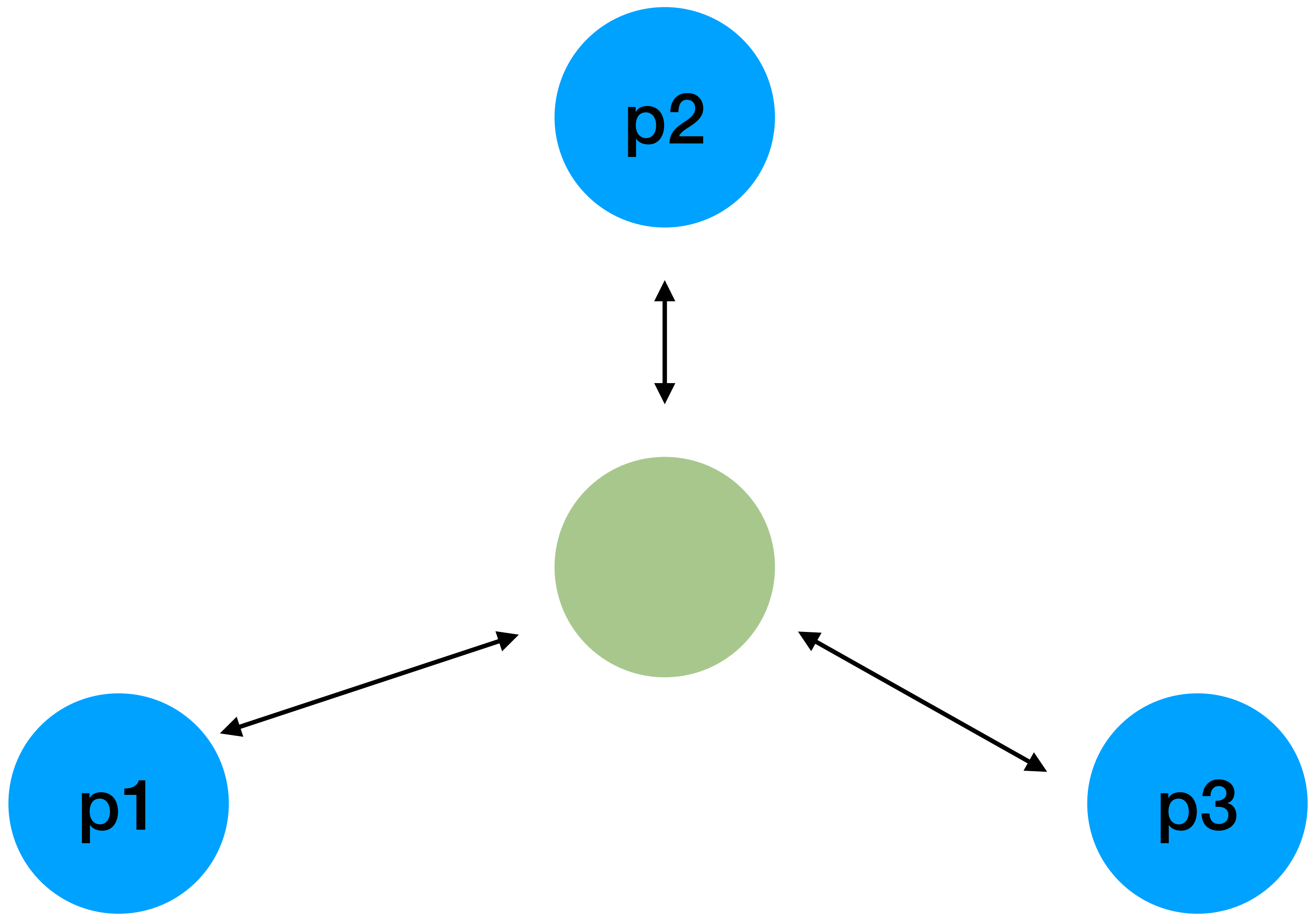
Semantics



- Randomly generate these inputs

Deliver(p1,5) . Deliver(p2, 3) . Crash(p1) . Start(p1) . …

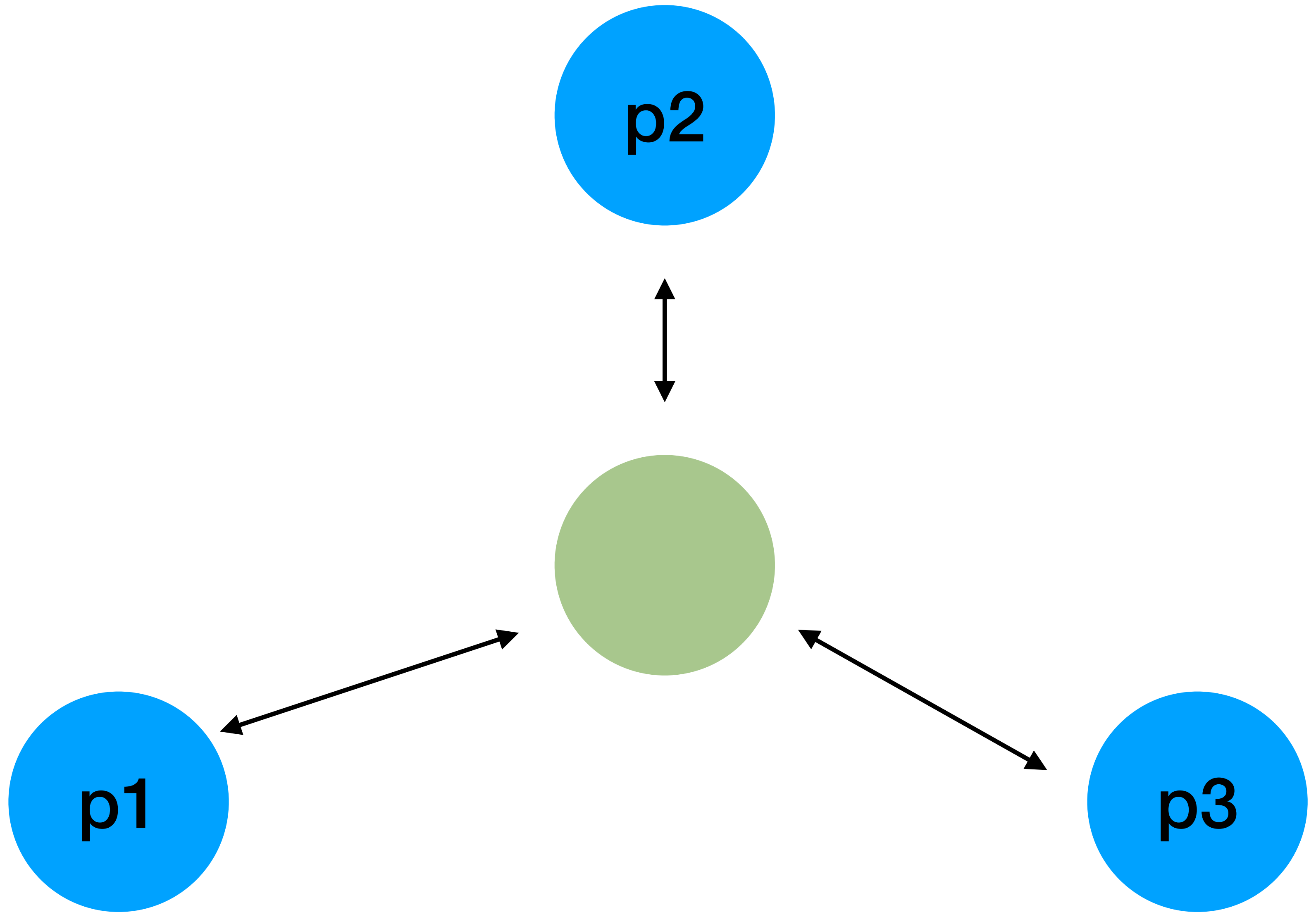
Semantics



- Randomly generate these inputs
- Light instrumentation
  - Messages
  - Process start/stop

Deliver(p1,5) . Deliver(p2, 3) . Crash(p1) . Start(p1) . …

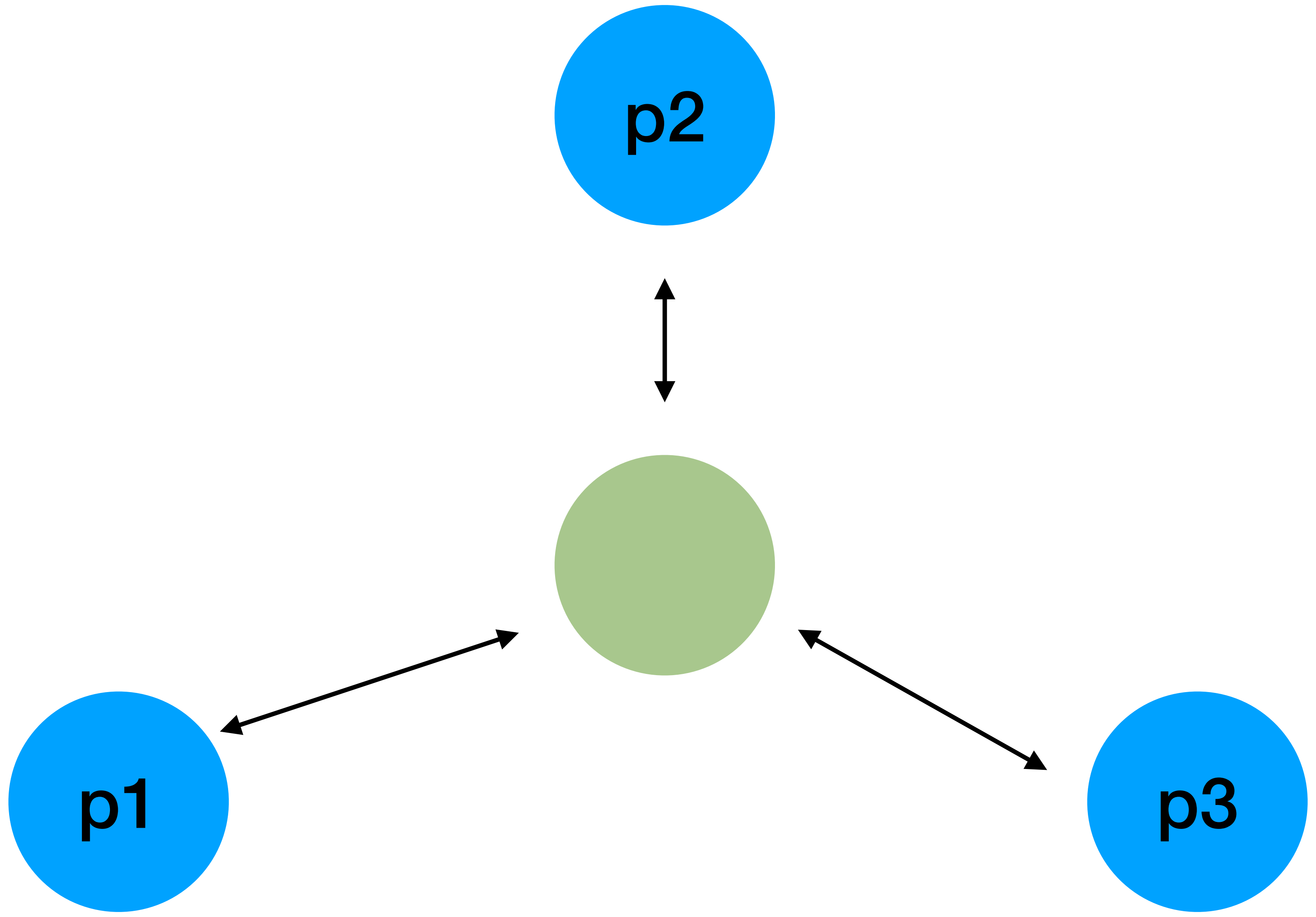
Semantics



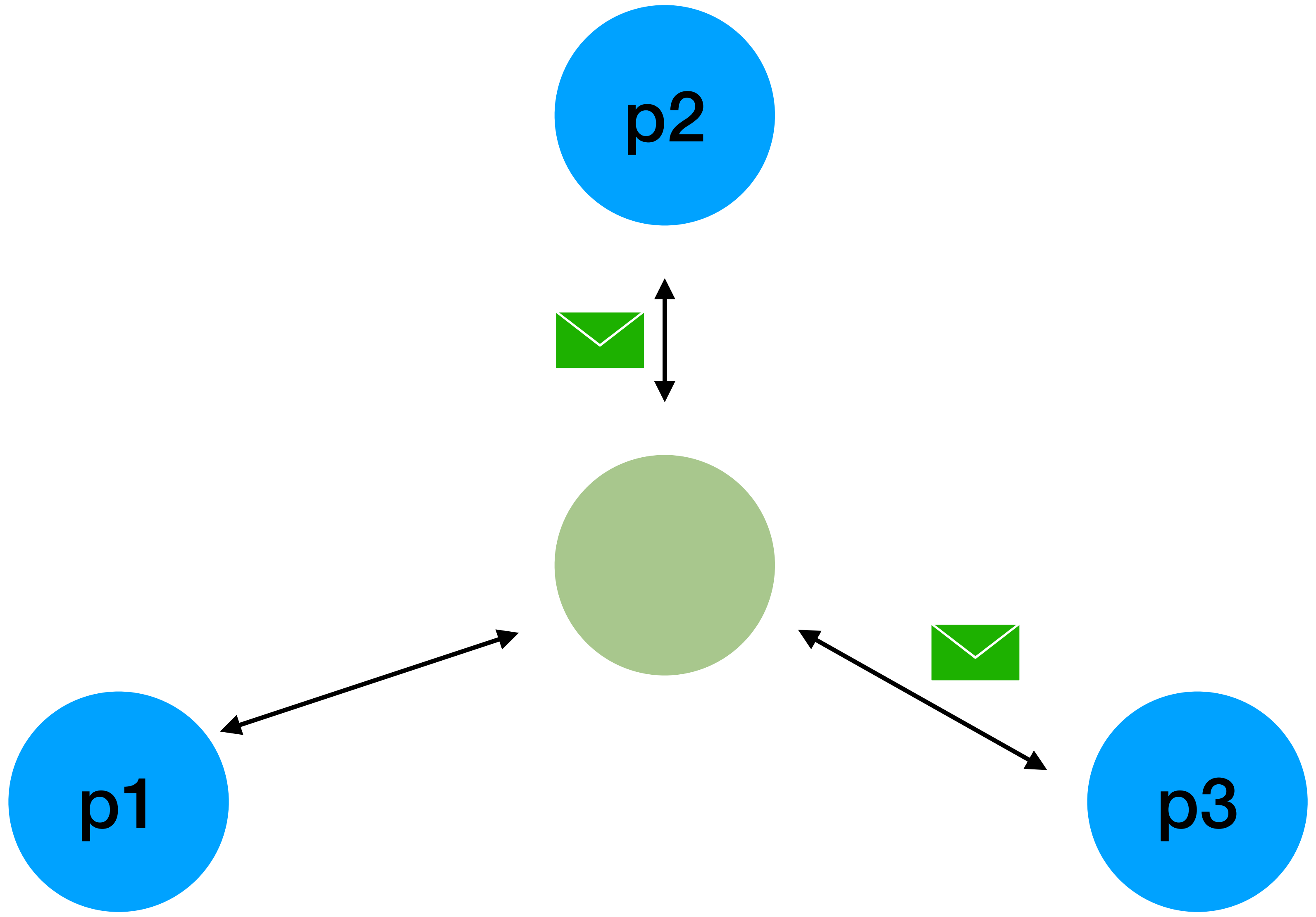
Deliver(p1,5) . Deliver(p2, 3) . Crash(p1) . Start(p1) . …

- Randomly generate these inputs
- Light instrumentation
    - Messages
    - Process start/stop
- Easy to define mutations

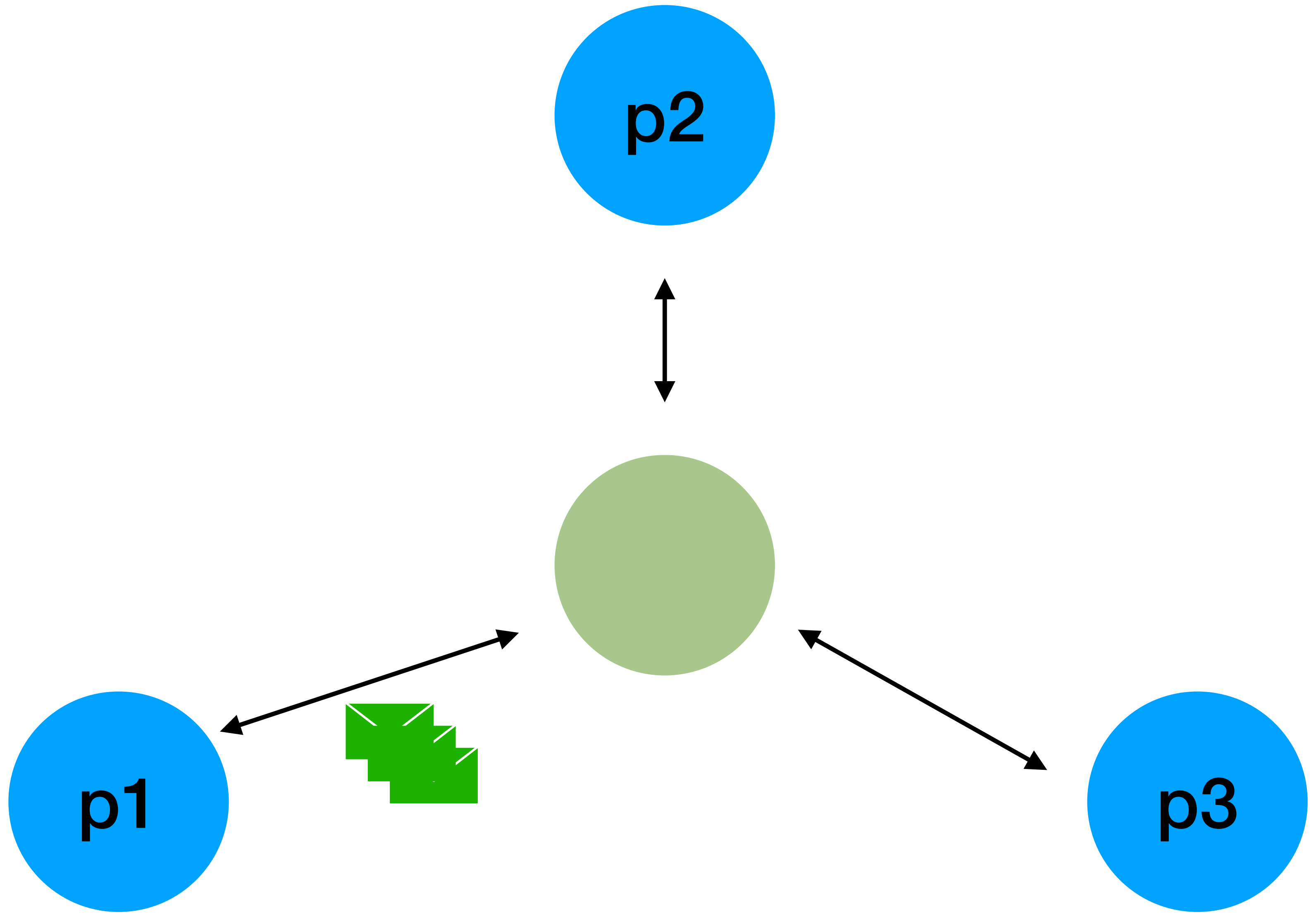
Semantics



Semantics



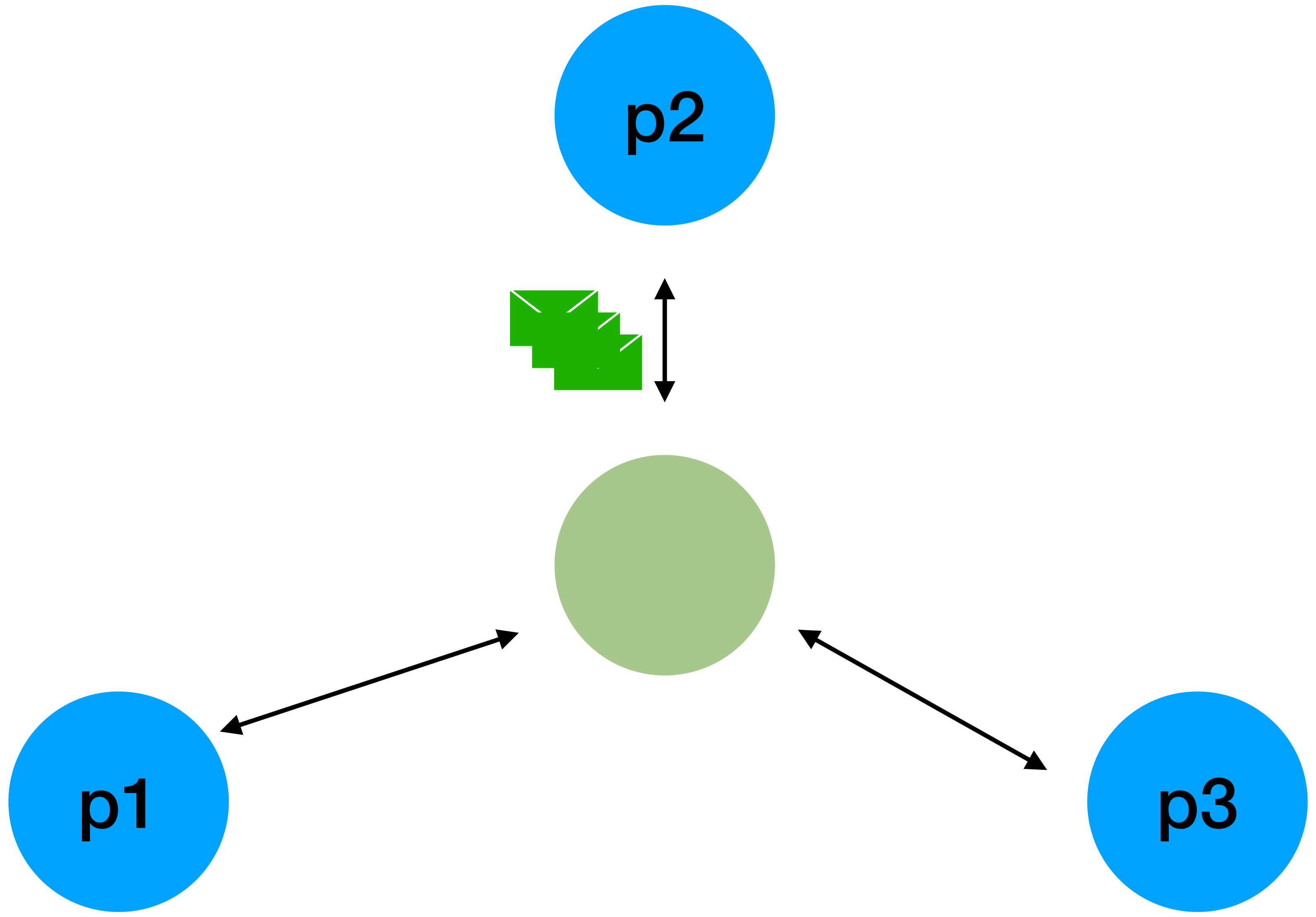
Semantics



Deliver(p1,5) .

SendRV(p1,p2) . SendRV(p1,p3).

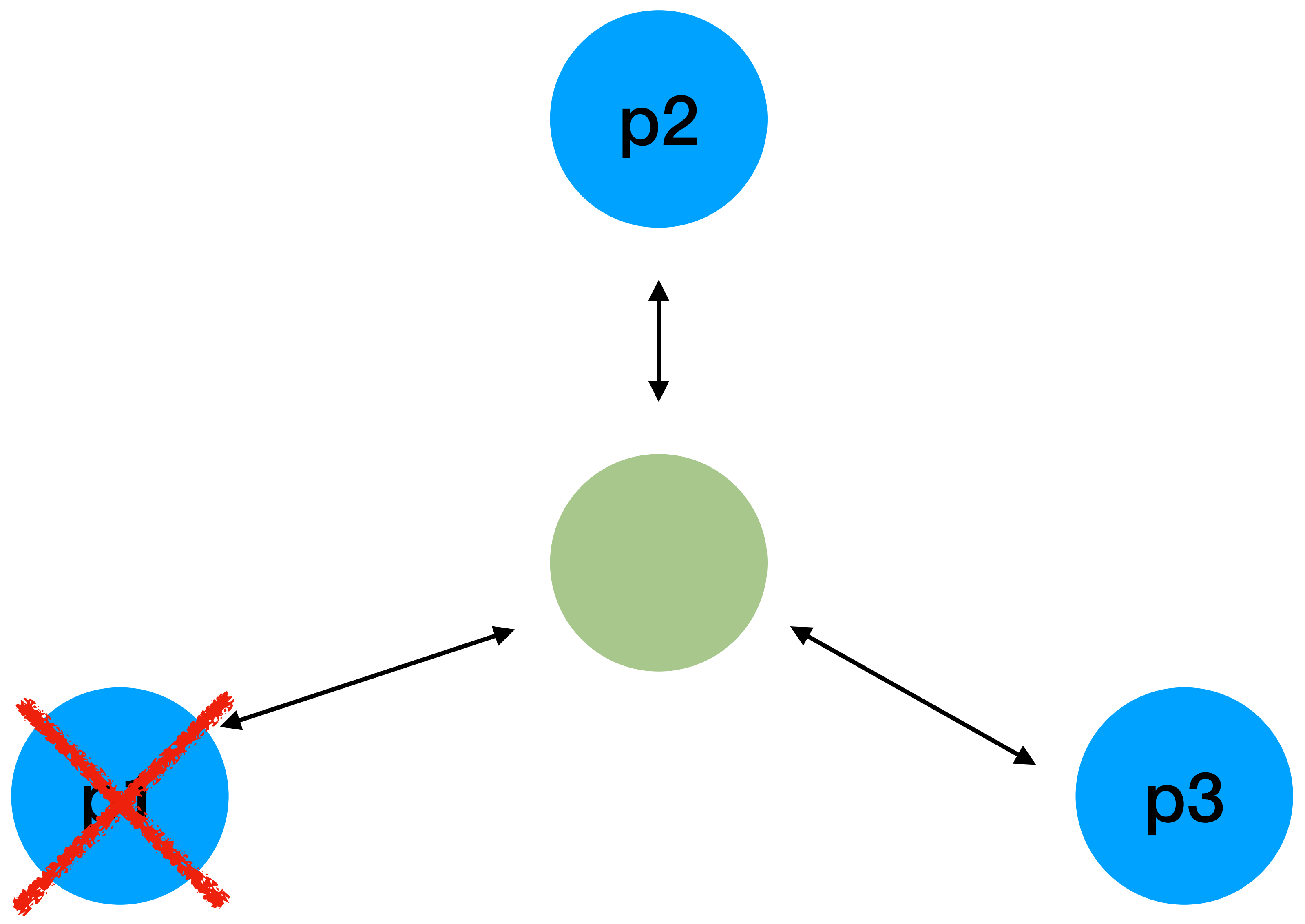
Semantics



Deliver(p1,5) . Deliver(p2, 3) .

SendRV(p1,p2) . SendRV(p1,p3). ReceiveRV(p2,p1) . SendRVResp(p2,p1) .

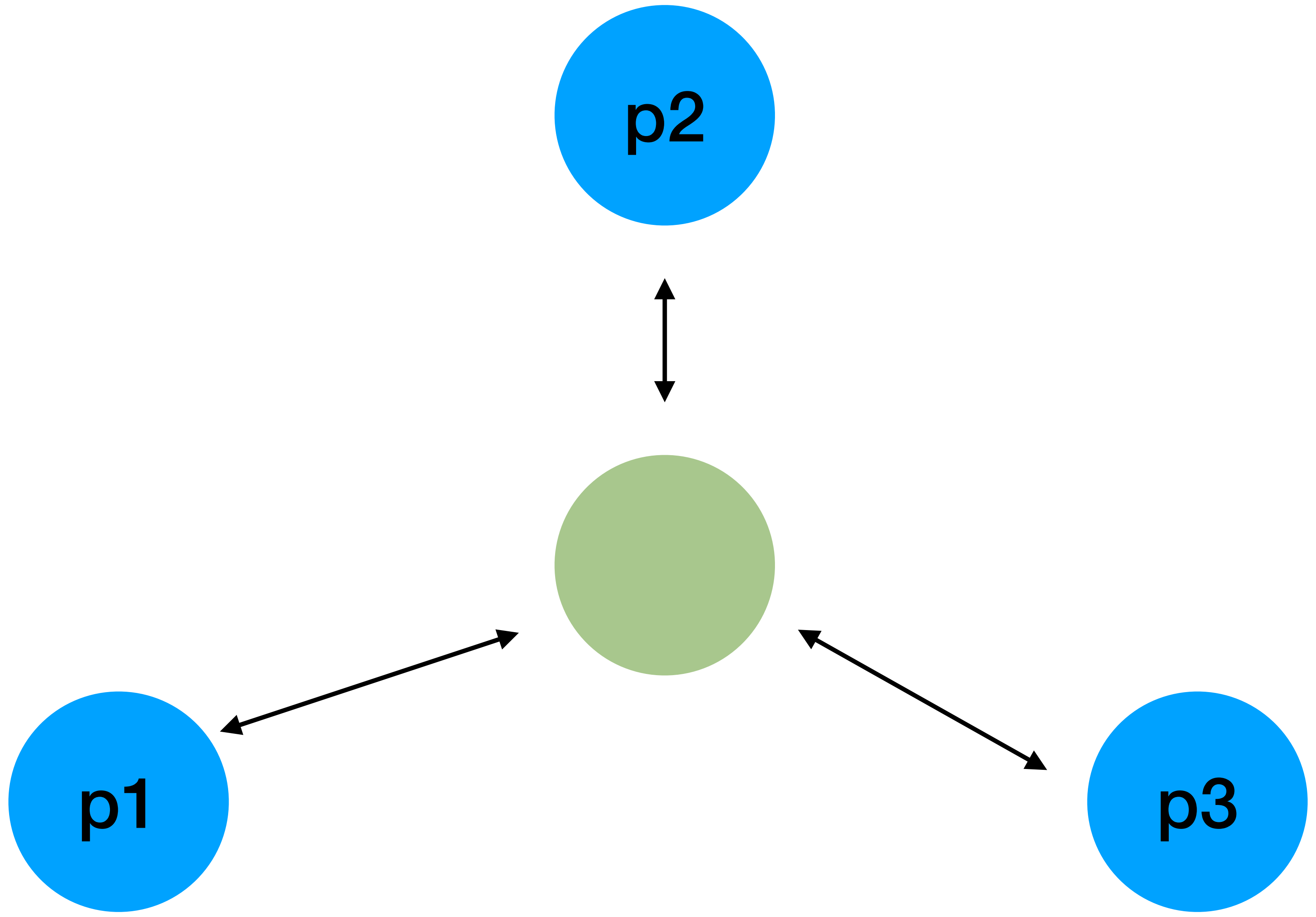
Semantics



Deliver(p1,5) . Deliver(p2, 3) . Crash(p1) .

SendRV(p1,p2) . SendRV(p1,p3). ReceiveRV(p2,p1) . SendRVResp(p2,p1) . StopProcess(p1) .

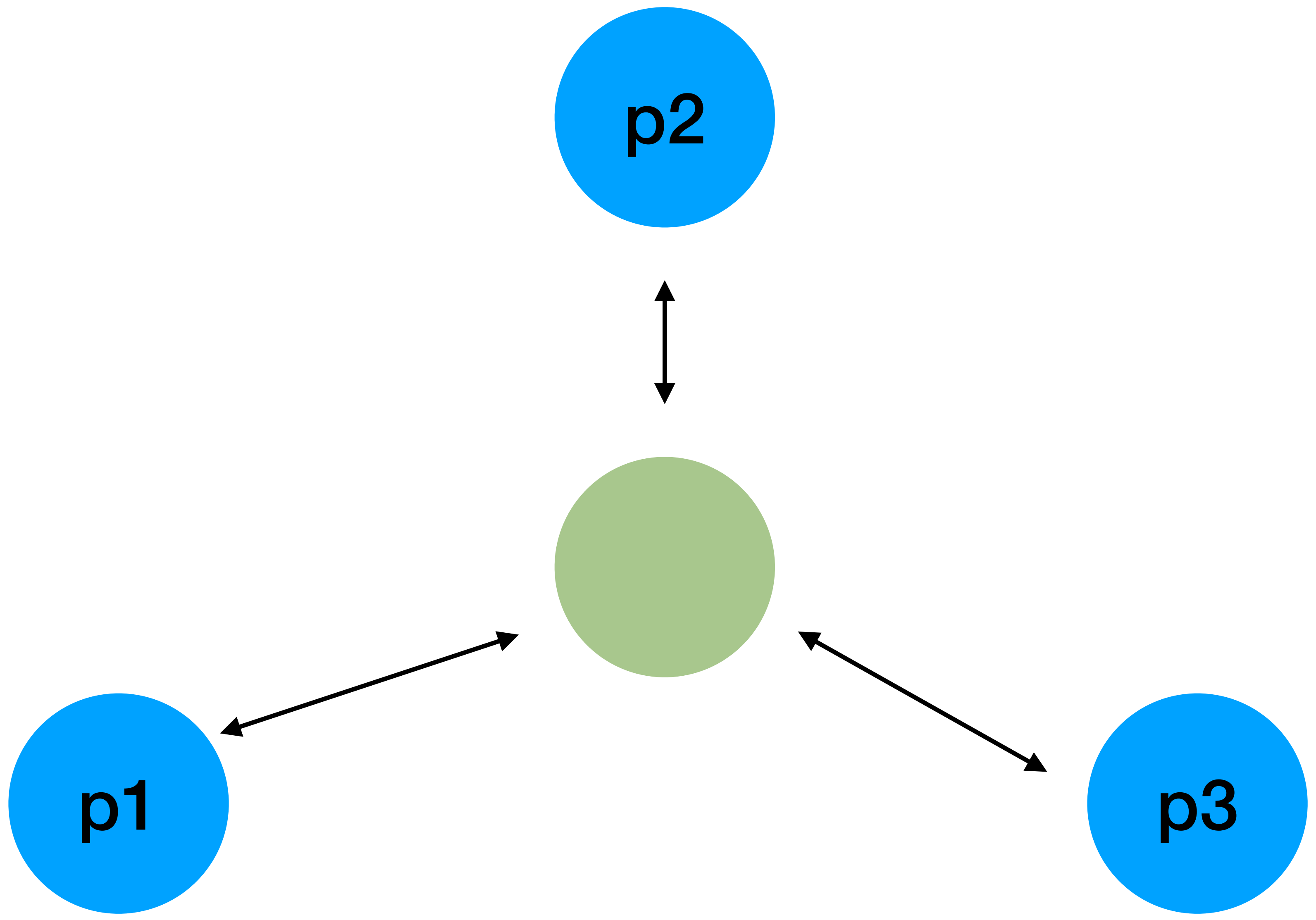
Semantics



Deliver(p1,5) . Deliver(p2, 3) . Crash(p1) . Start(p1) .

SendRV(p1,p2) . SendRV(p1,p3). ReceiveRV(p2,p1) . SendRVResp(p2,p1) . StopProcess(p1) . StartProcess(p1) .

Semantics

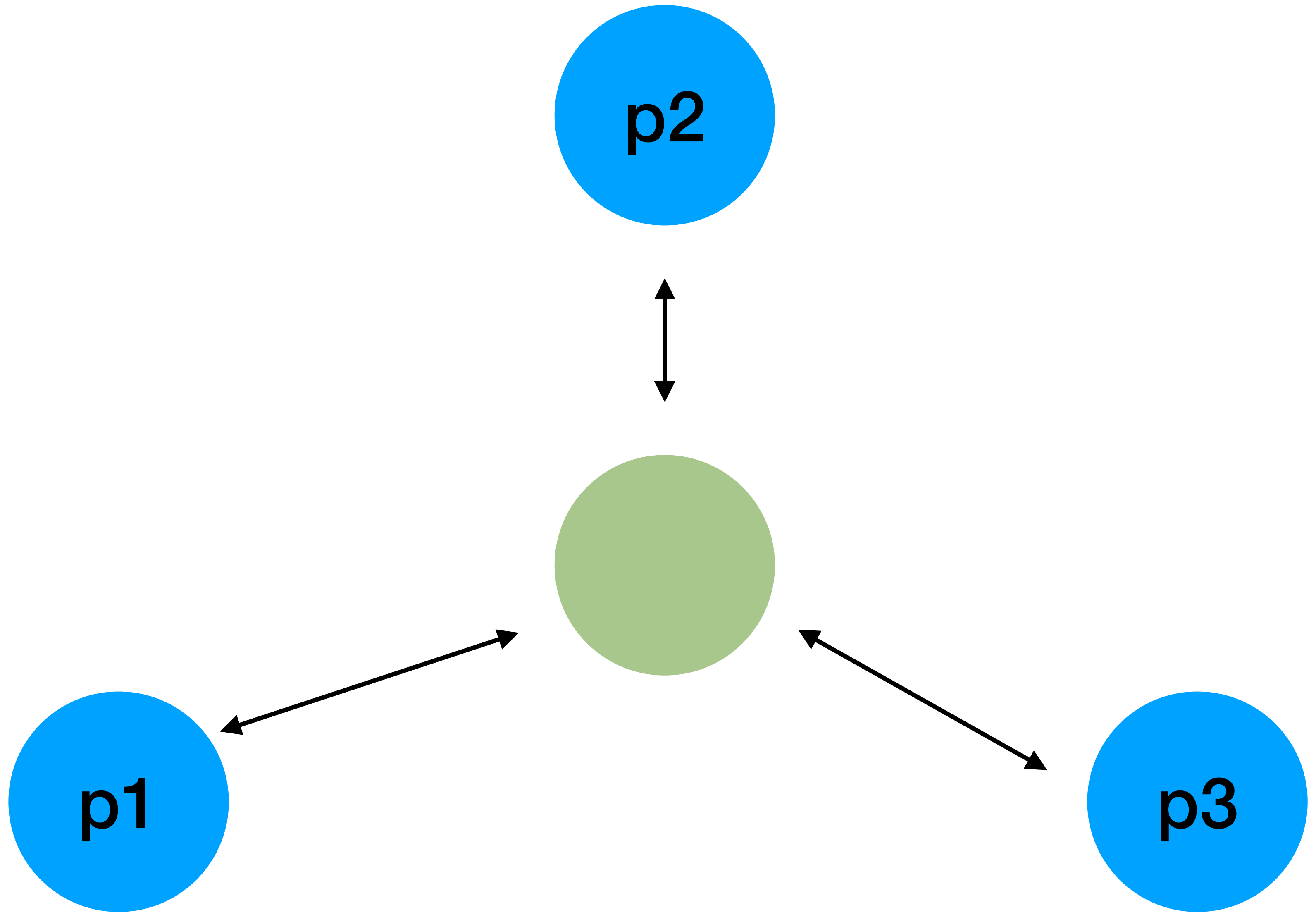


Deliver(p1,5) . Deliver(p2, 3) . Crash(p1) . Start(p1) . …

SendRV(p1,p2) . SendRV(p1,p3). ReceiveRV(p2,p1) . SendRVResp(p2,p1) . StopProcess(p1) . StartProcess(p1) .

… . BecomeLeader(p1,1) …

Semantics



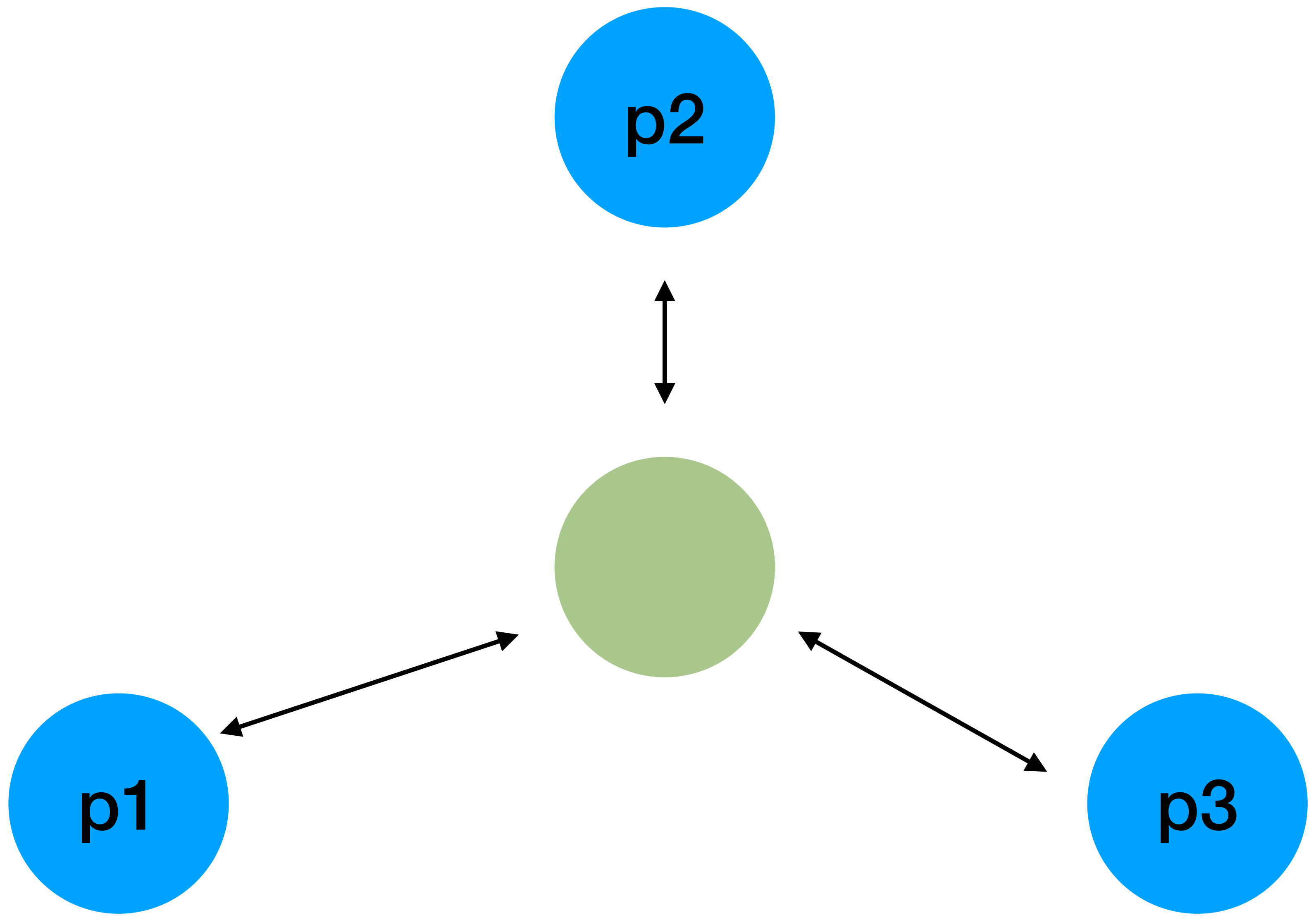
Execution Events

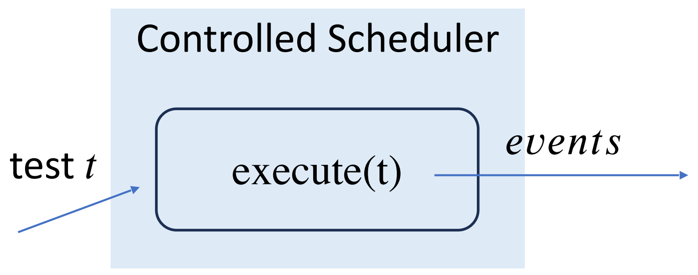
Deliver(p1,5) . Deliver(p2, 3) . Crash(p1) . Start(p1) . …

SendRV(p1,p2) . SendRV(p1,p3). ReceiveRV(p2,p1) . SendRVResp(p2,p1) . StopProcess(p1) . StartProcess(p1) .

… . BecomeLeader(p1,1) …

Semantics





Execution Events

Deliver(p1,5) . Deliver(p2, 3) . Crash(p1) . Start(p1) . …

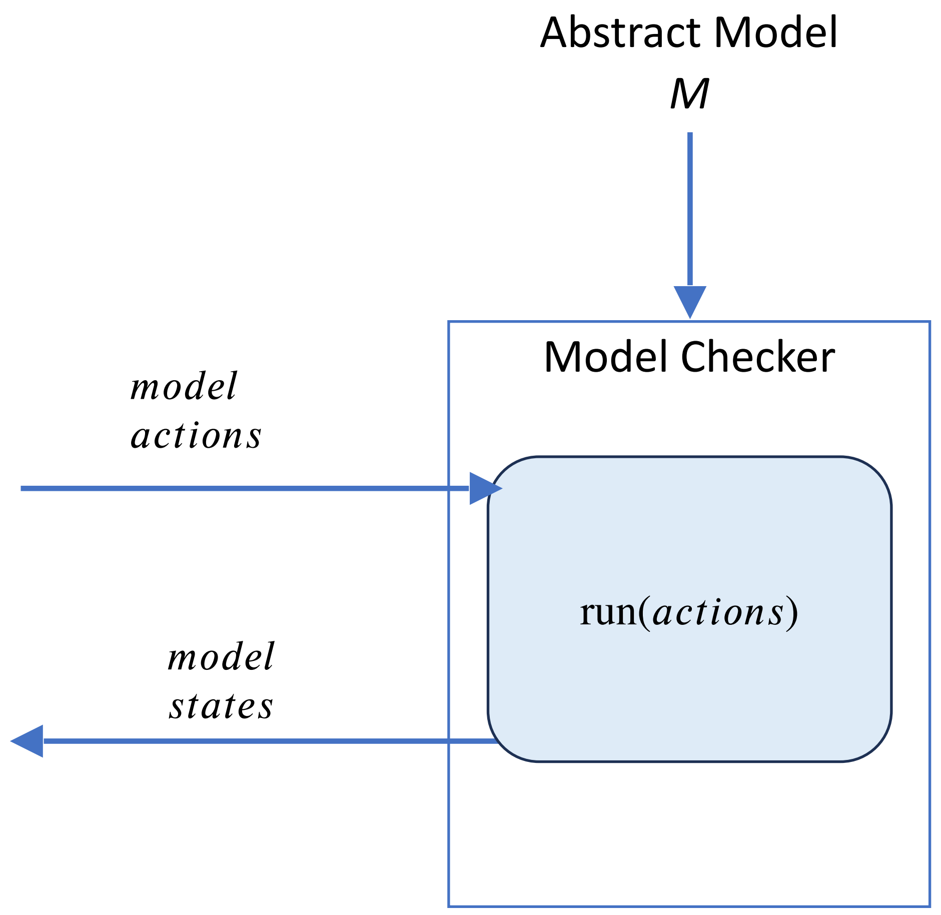
SendRV(p1,p2) . SendRV(p1,p3). ReceiveRV(p2,p1) . SendRVResp(p2,p1) . StopProcess(p1) . StartProcess(p1) .

… . BecomeLeader(p1,1) …

Simulating traces on the Model

Simulating traces on the Model

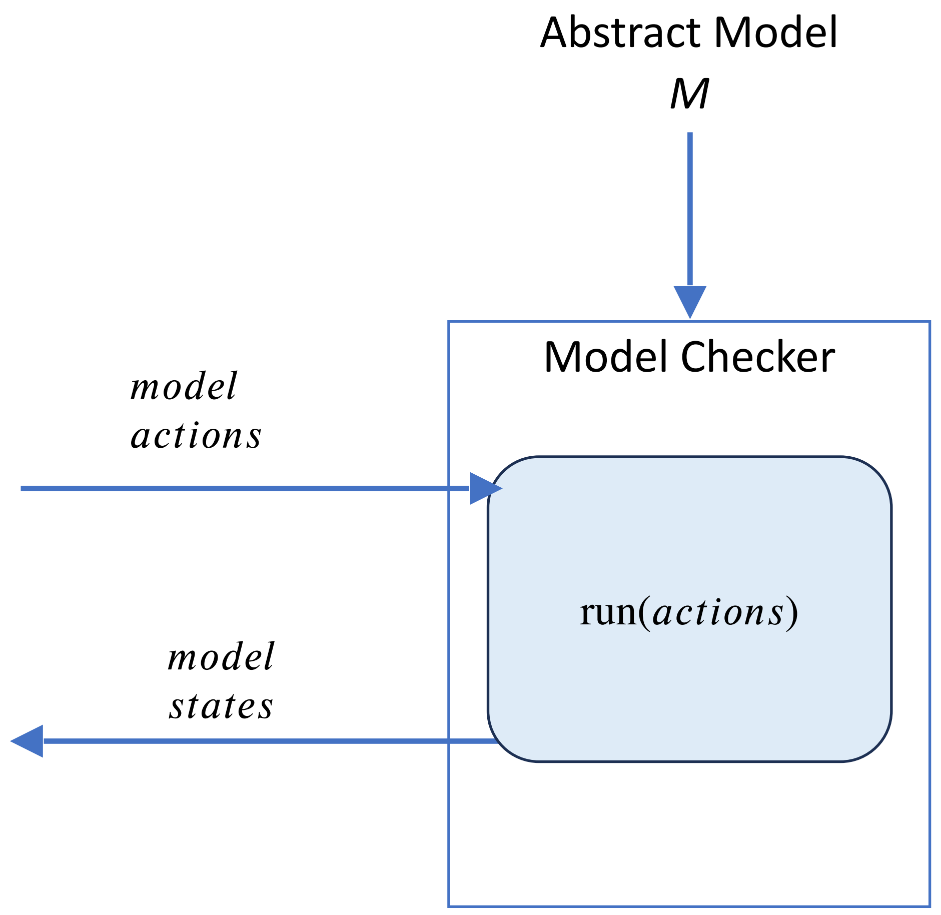
- Goal: To obtain a state sequence trace from the action sequence



Simulating traces on the Model

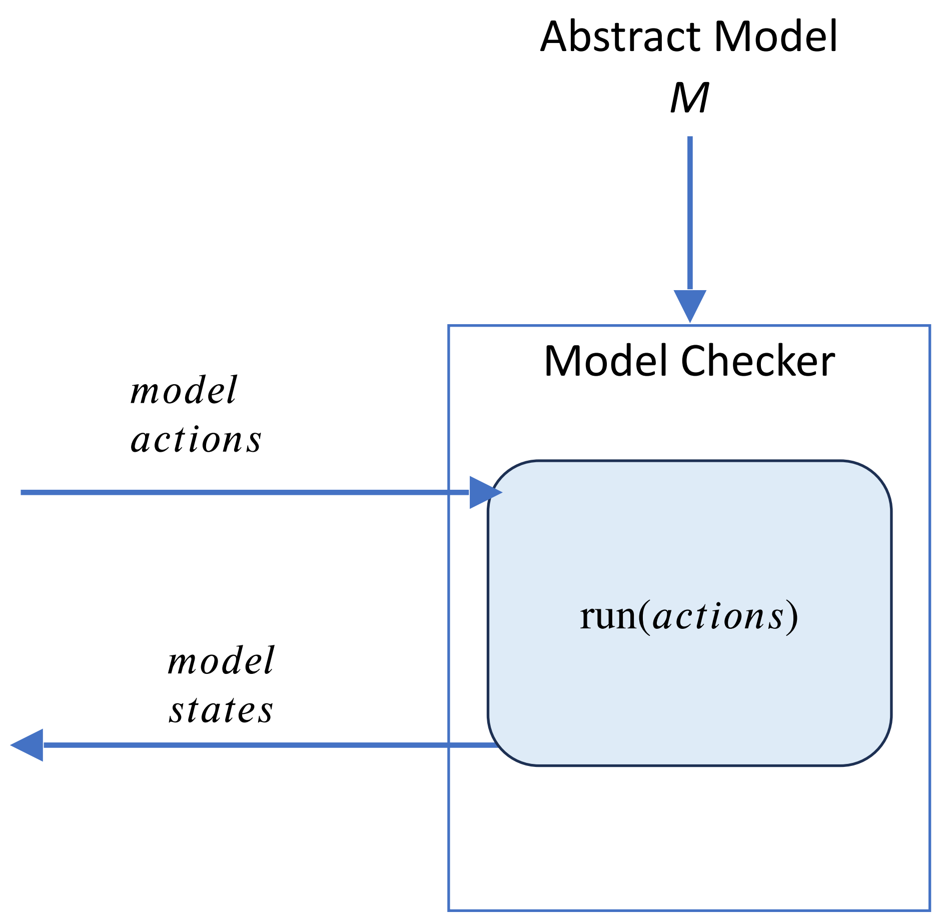
- Goal: To obtain a state sequence trace from the action sequence

- Some challenges



Simulating traces on the Model

- Goal: To obtain a state sequence trace from the action sequence
- Some challenges
  - Should be fast



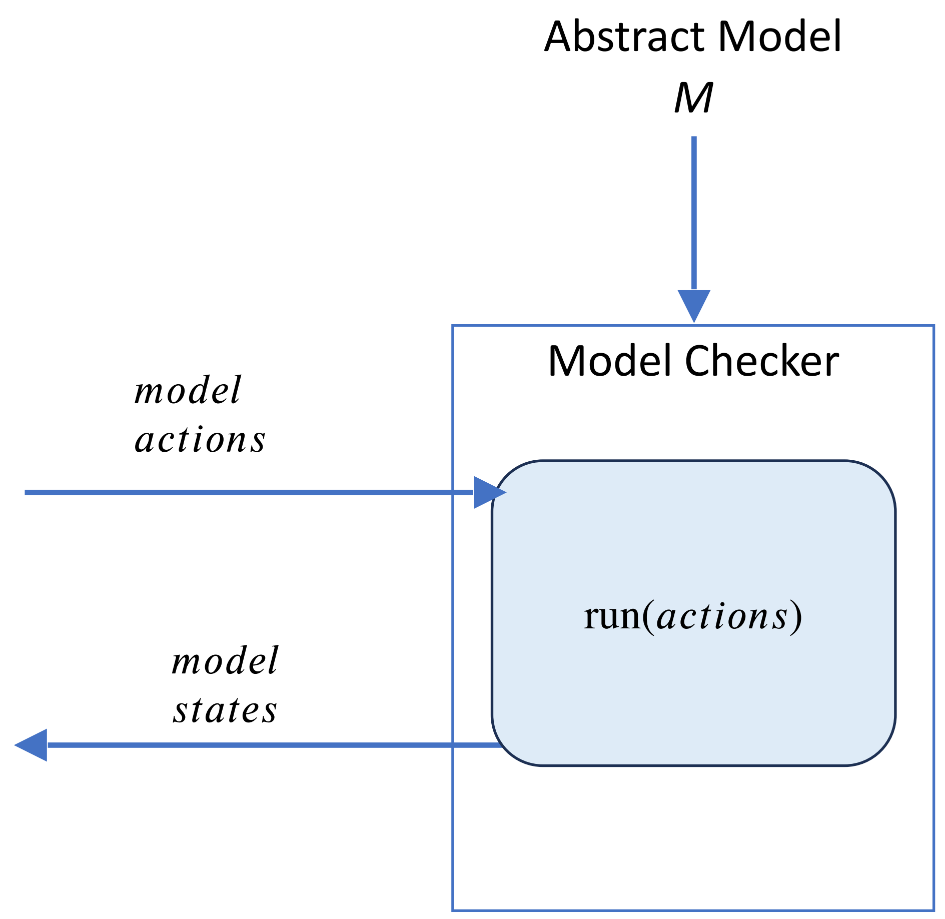
Simulating traces on the Model

- Goal: To obtain a state sequence trace from the action sequence

- Some challenges

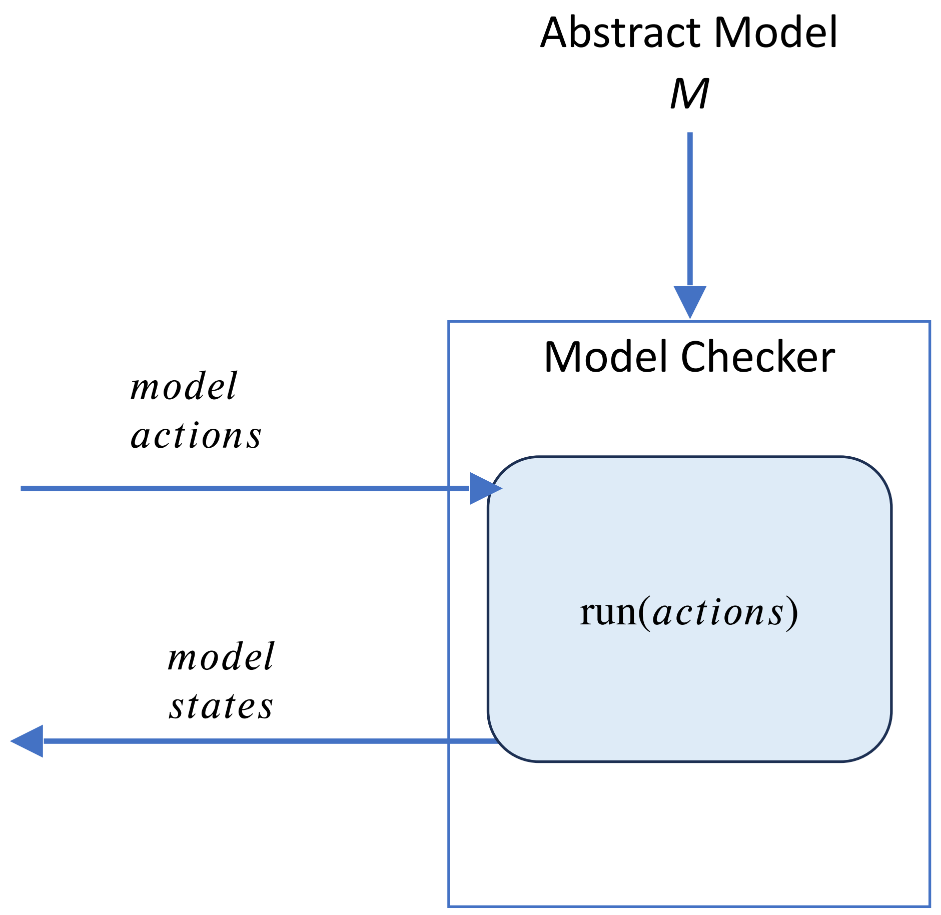
  - Should be fast

  - Model checker should be able to enumerate actions

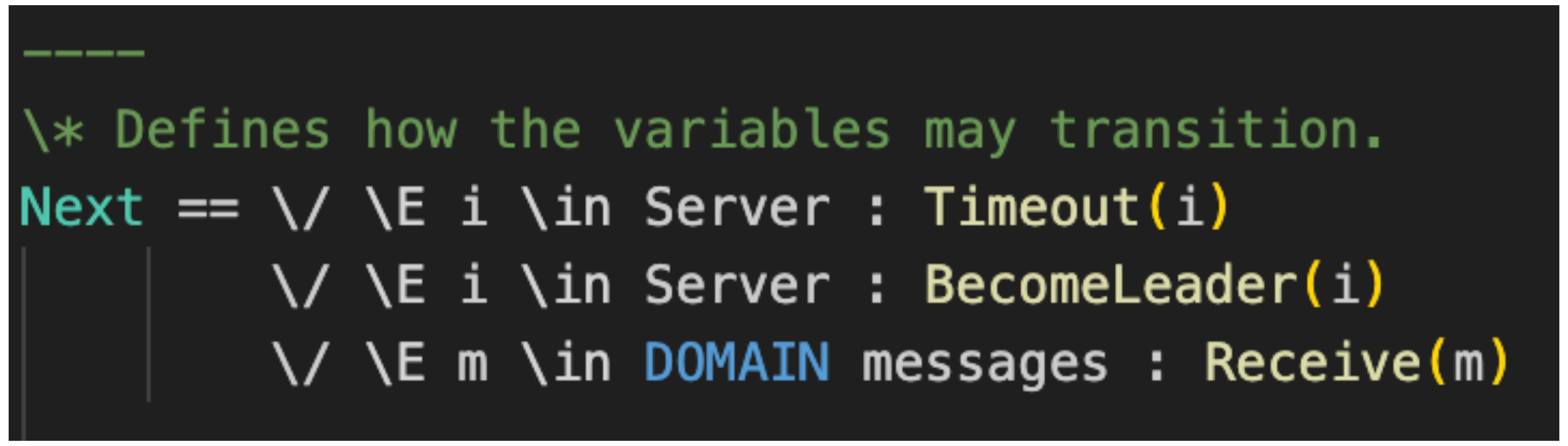


Simulating traces on the Model

- Goal: To obtain a state sequence trace from the action sequence
- Some challenges
  - Should be fast
  - Model checker should be able to enumerate actions
  - Ensure only relevant actions are executed

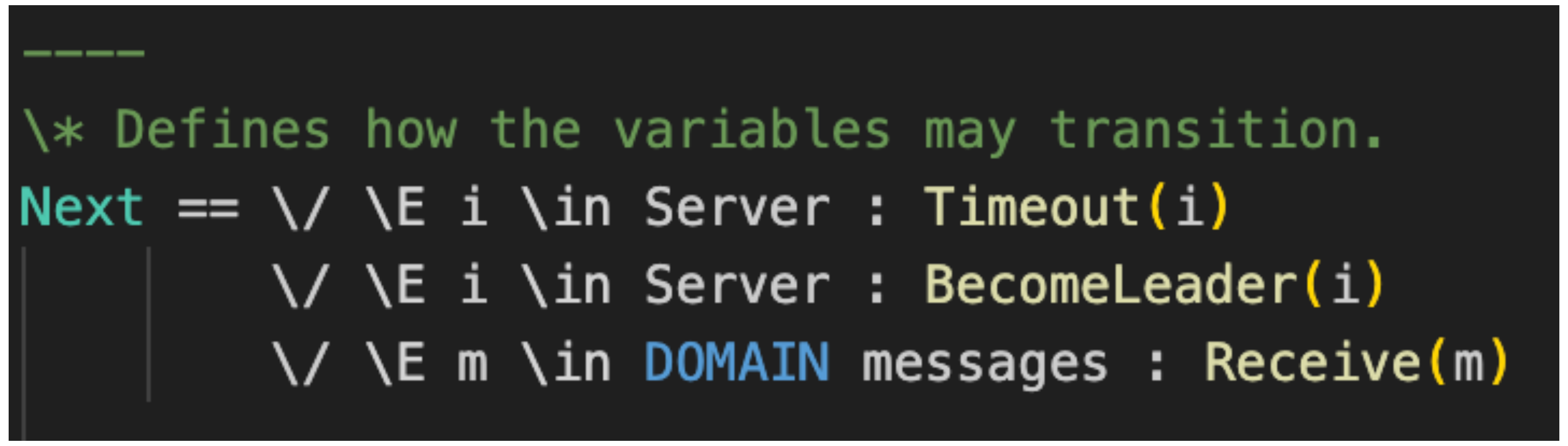


Enumerating actions

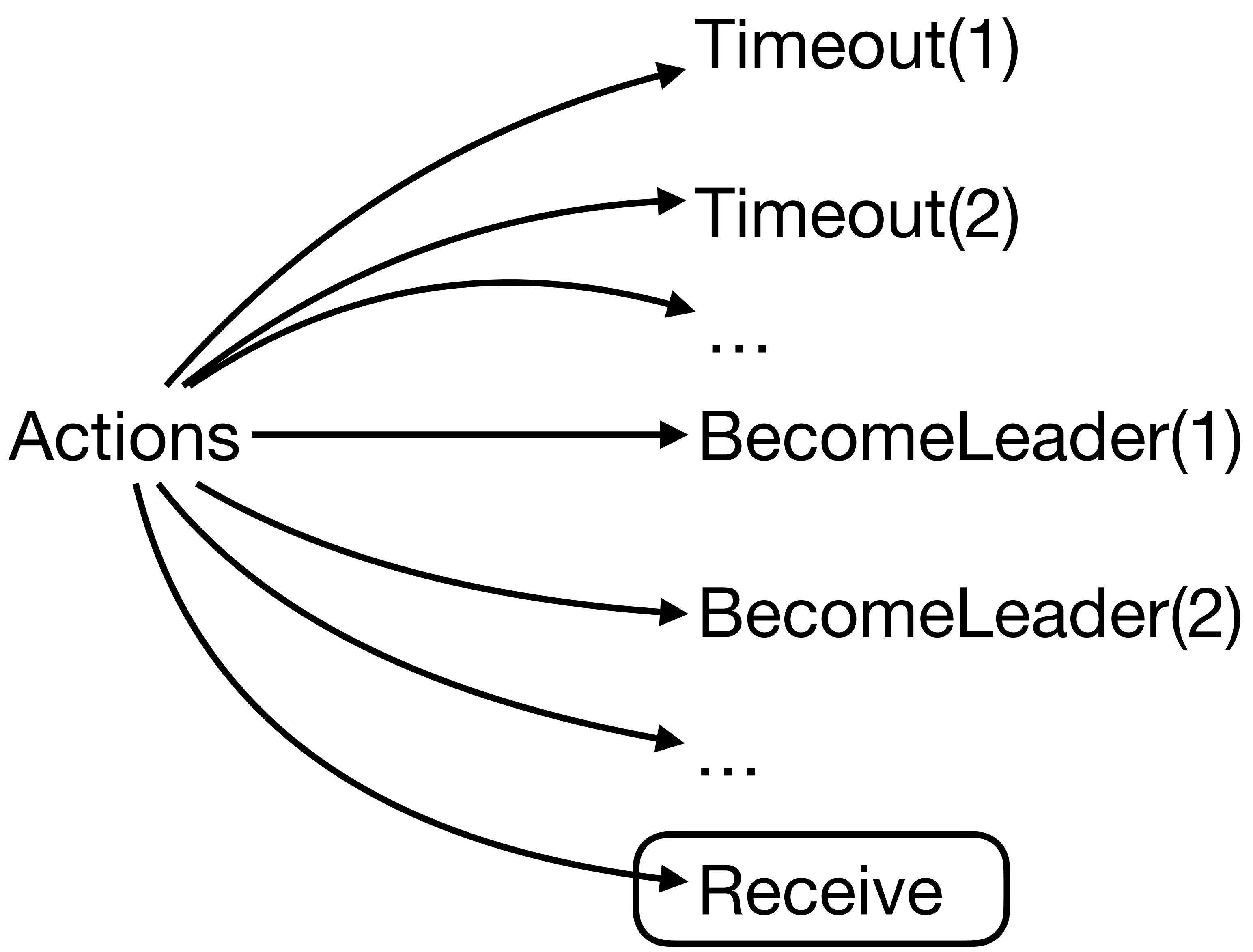


```
----
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E m \in DOMAIN messages : Receive(m)
```

Enumerating actions



```
----
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E m \in DOMAIN messages : Receive(m)
```



Actions

Timeout(1)

Timeout(2)

…

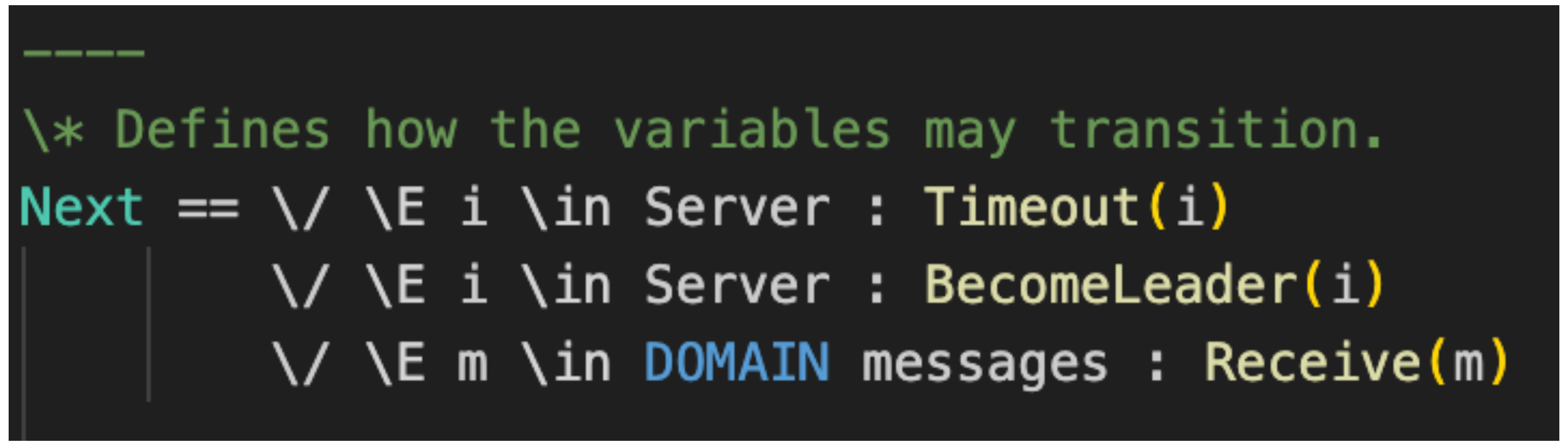
BecomeLeader(1)

BecomeLeader(2)

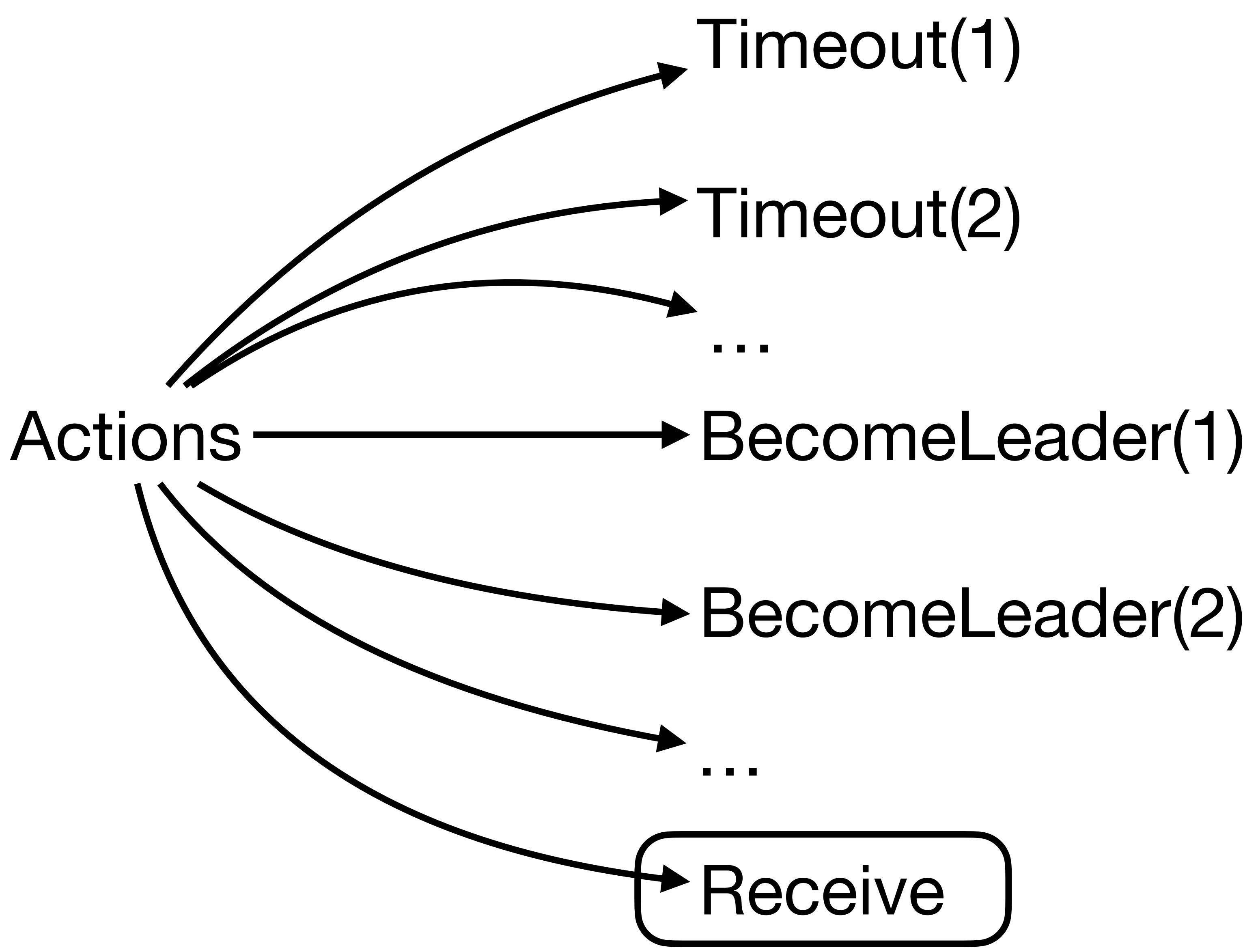
…

Receive

Enumerating actions



```
----
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E m \in DOMAIN messages : Receive(m)
```



Actions

Timeout(1)

Timeout(2)

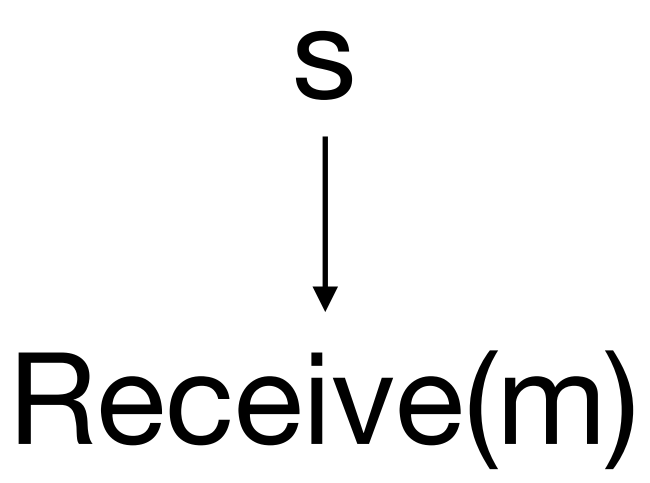
…

BecomeLeader(1)

BecomeLeader(2)

…

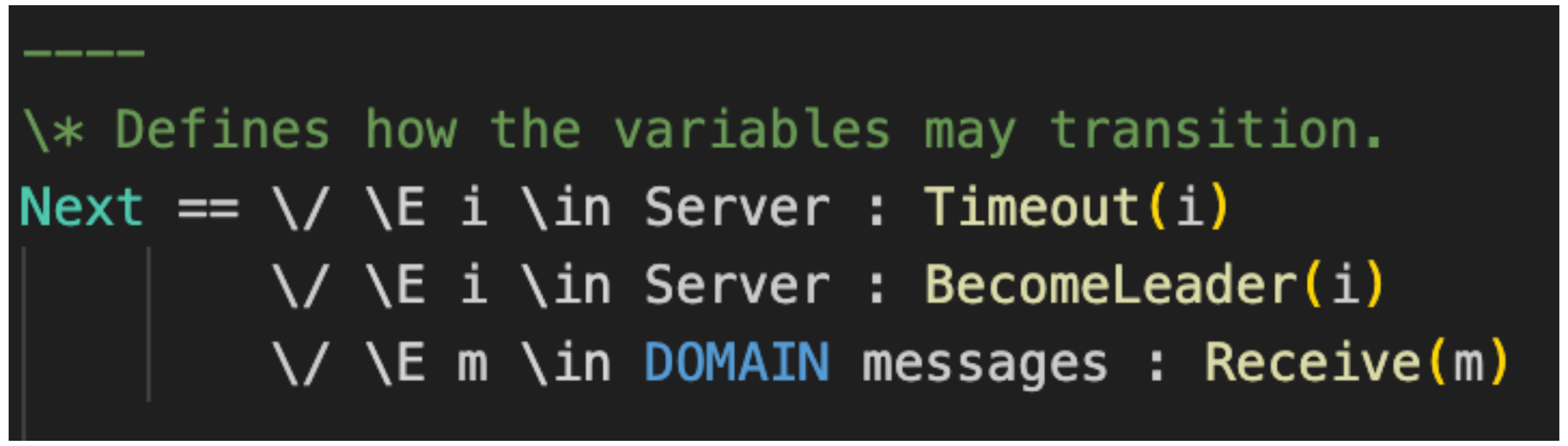
Receive



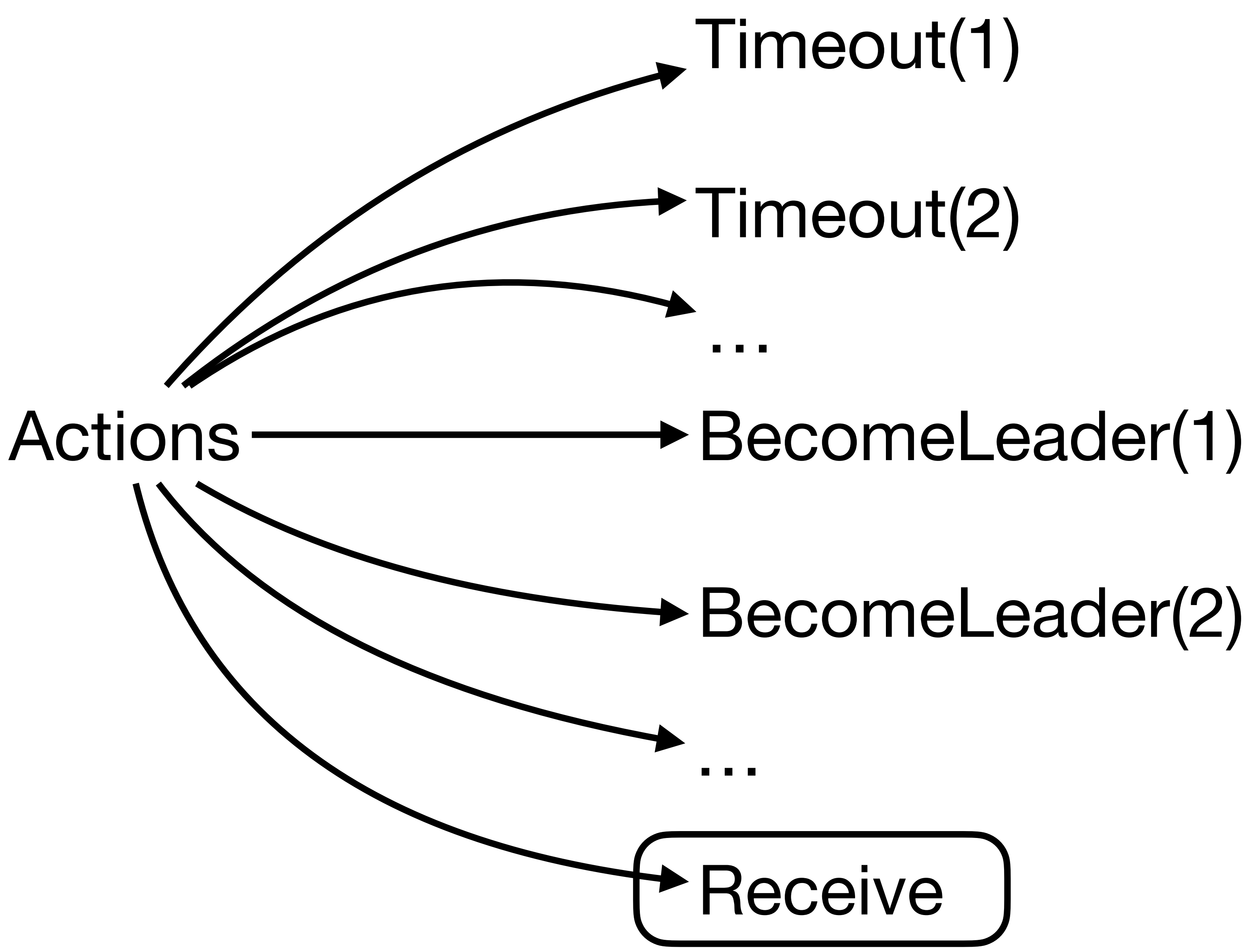
s

Receive(m)

Enumerating actions



```
----
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E m \in DOMAIN messages : Receive(m)
```



Actions

Timeout(1)

Timeout(2)

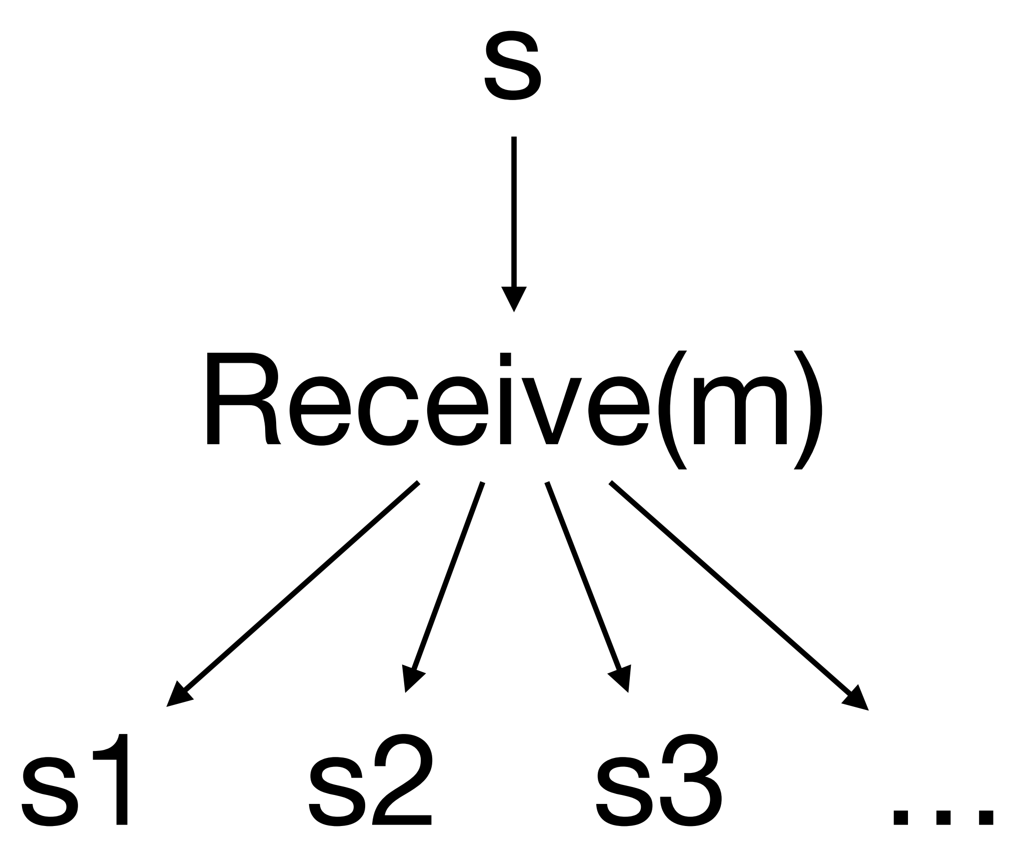
…

BecomeLeader(1)

BecomeLeader(2)

…

Receive



s

Receive(m)

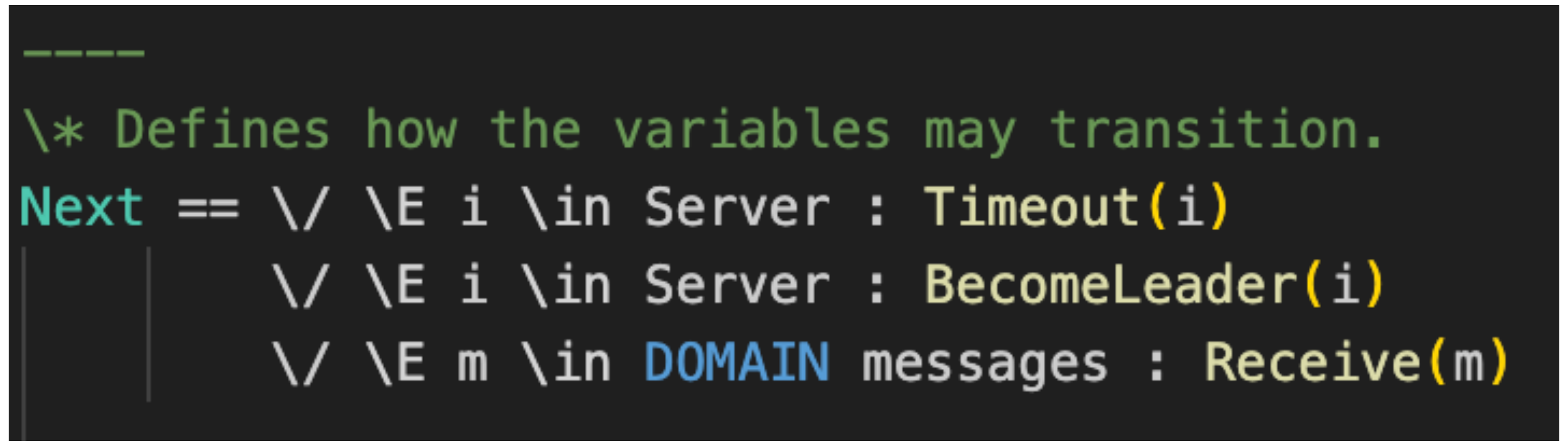
s1

s2

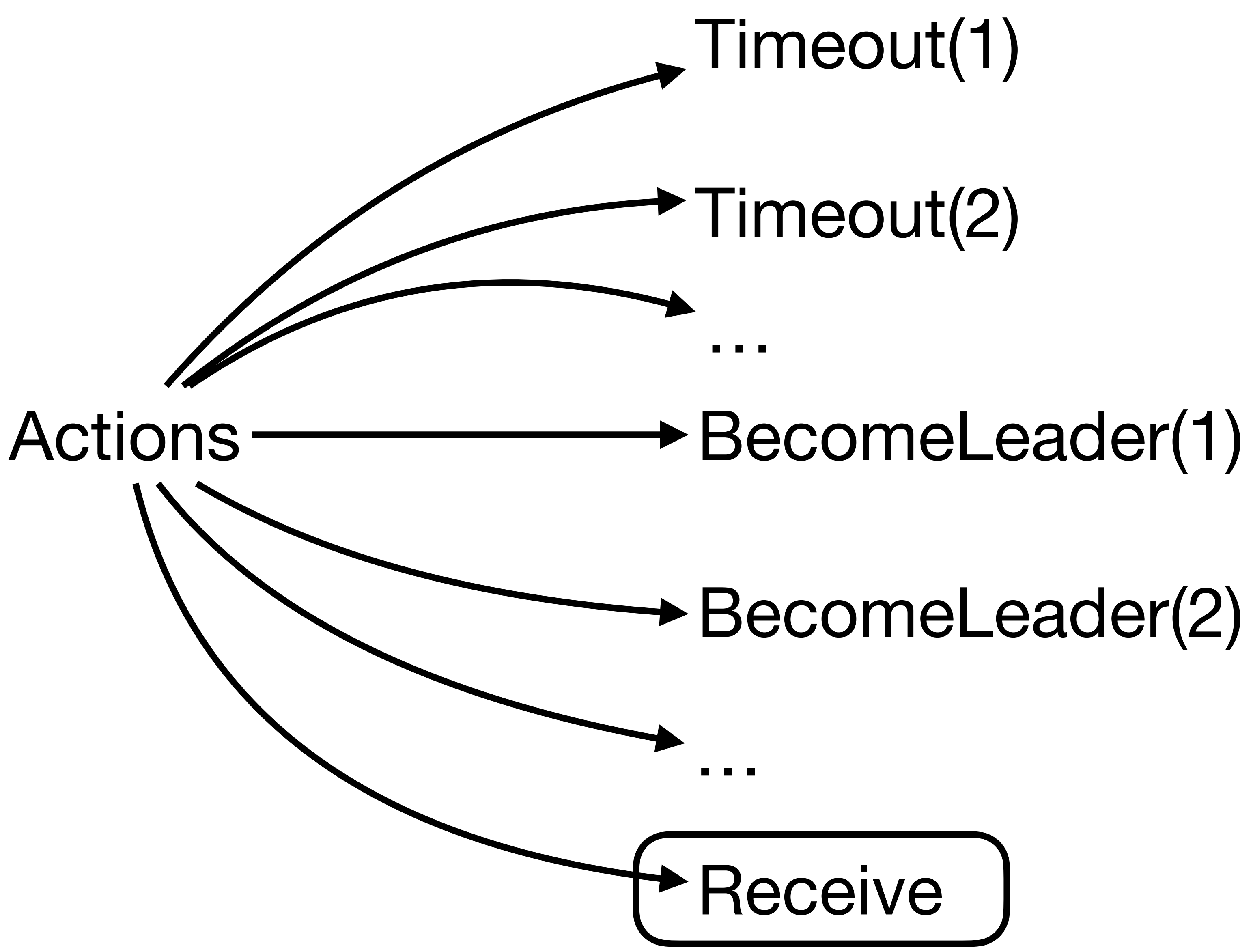
s3

…

Enumerating actions



```
----
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E m \in DOMAIN messages : Receive(m)
```



Actions

Timeout(1)

Timeout(2)

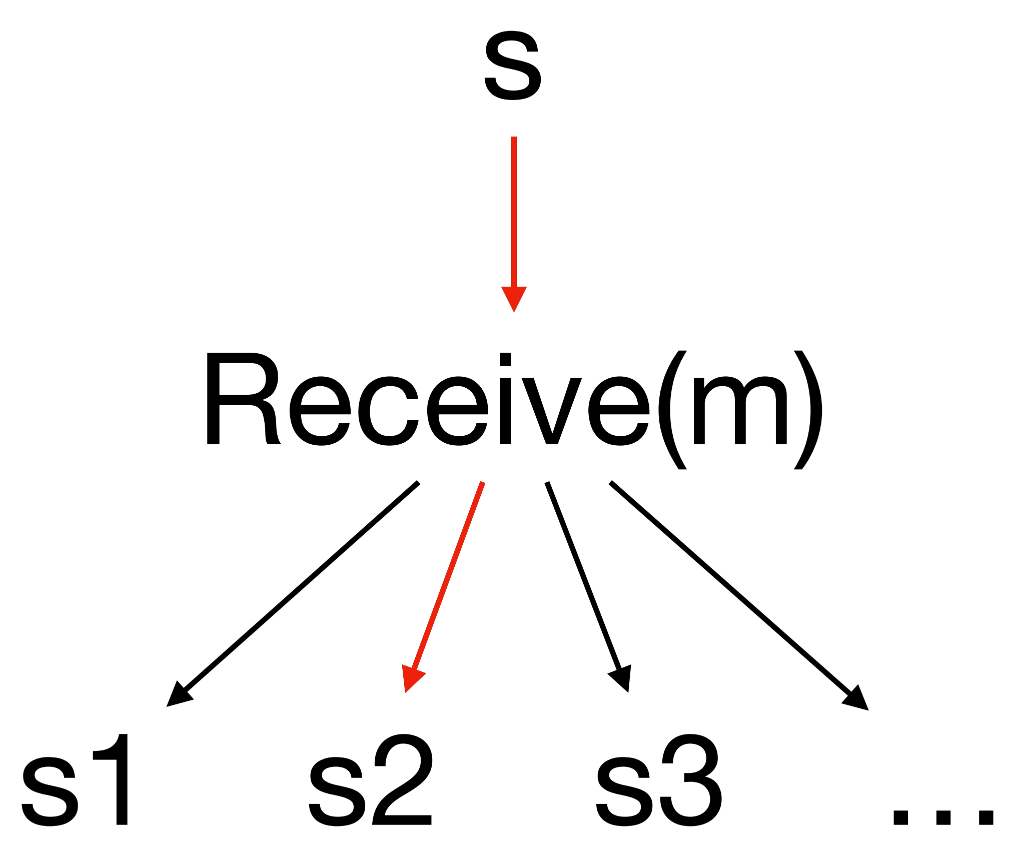
…

BecomeLeader(1)

BecomeLeader(2)

…

Receive



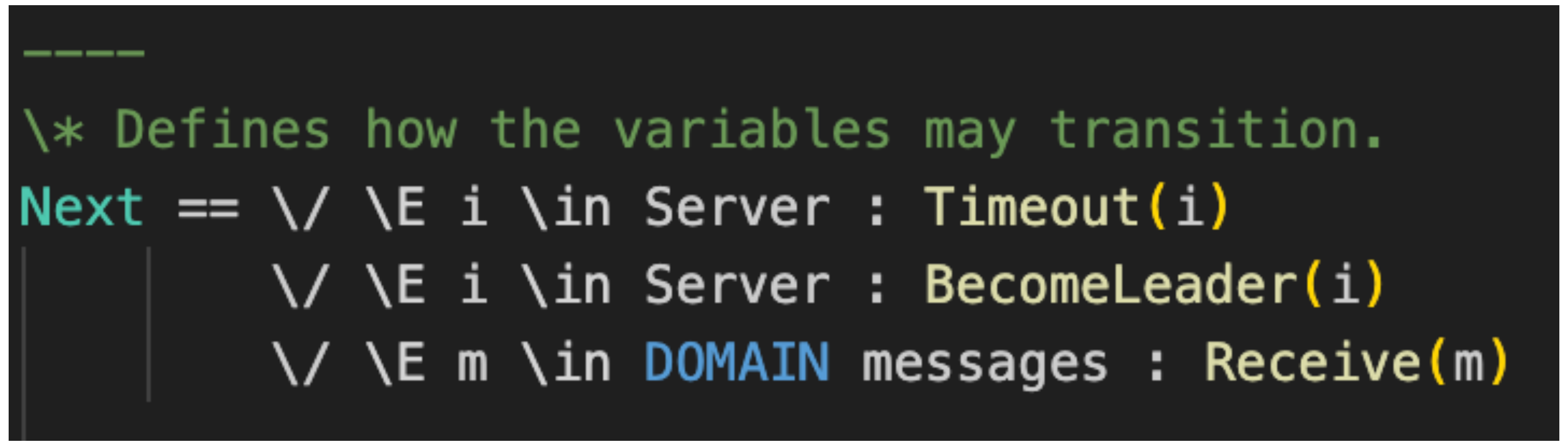
s

Receive(m)

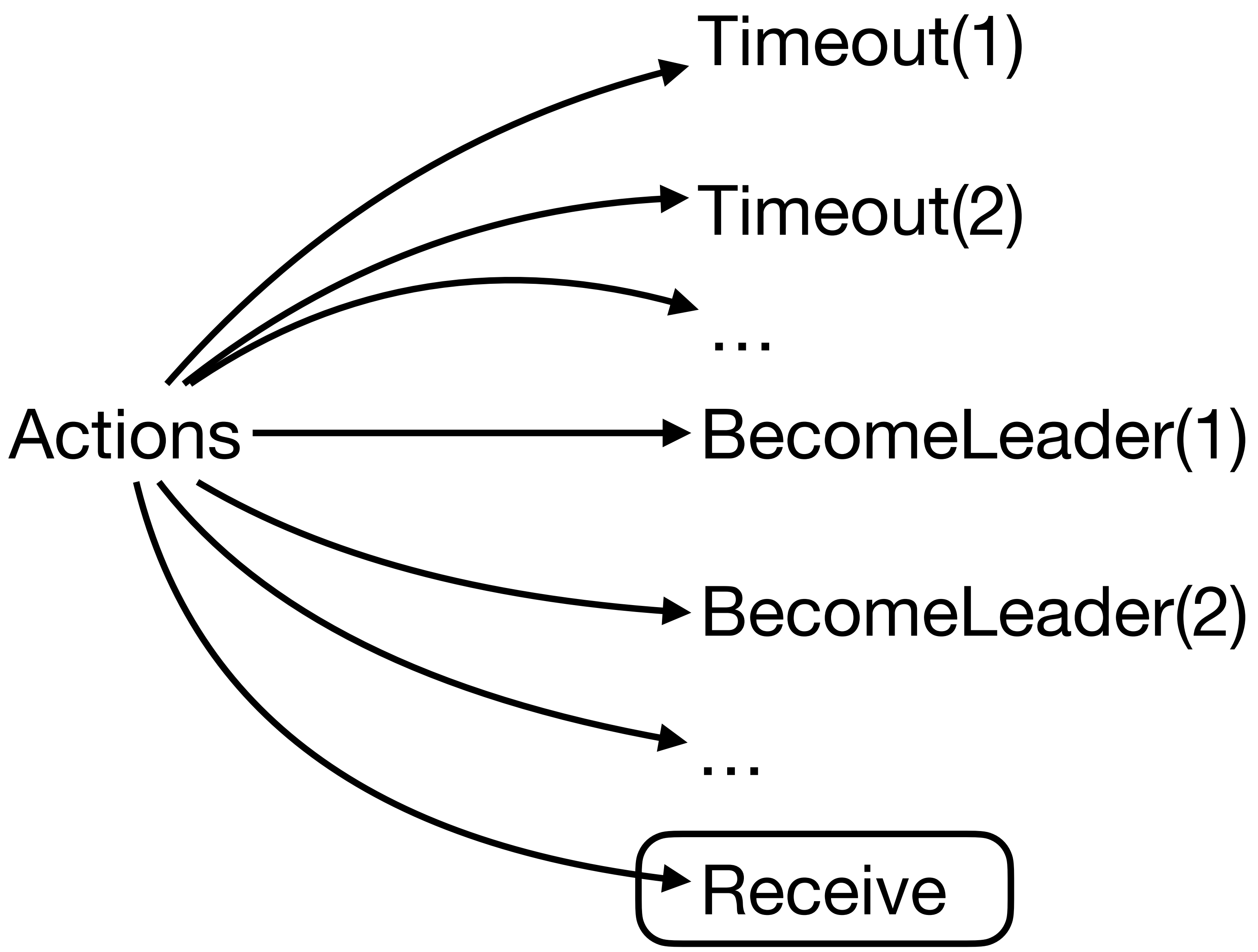
s1 s2 s3 …

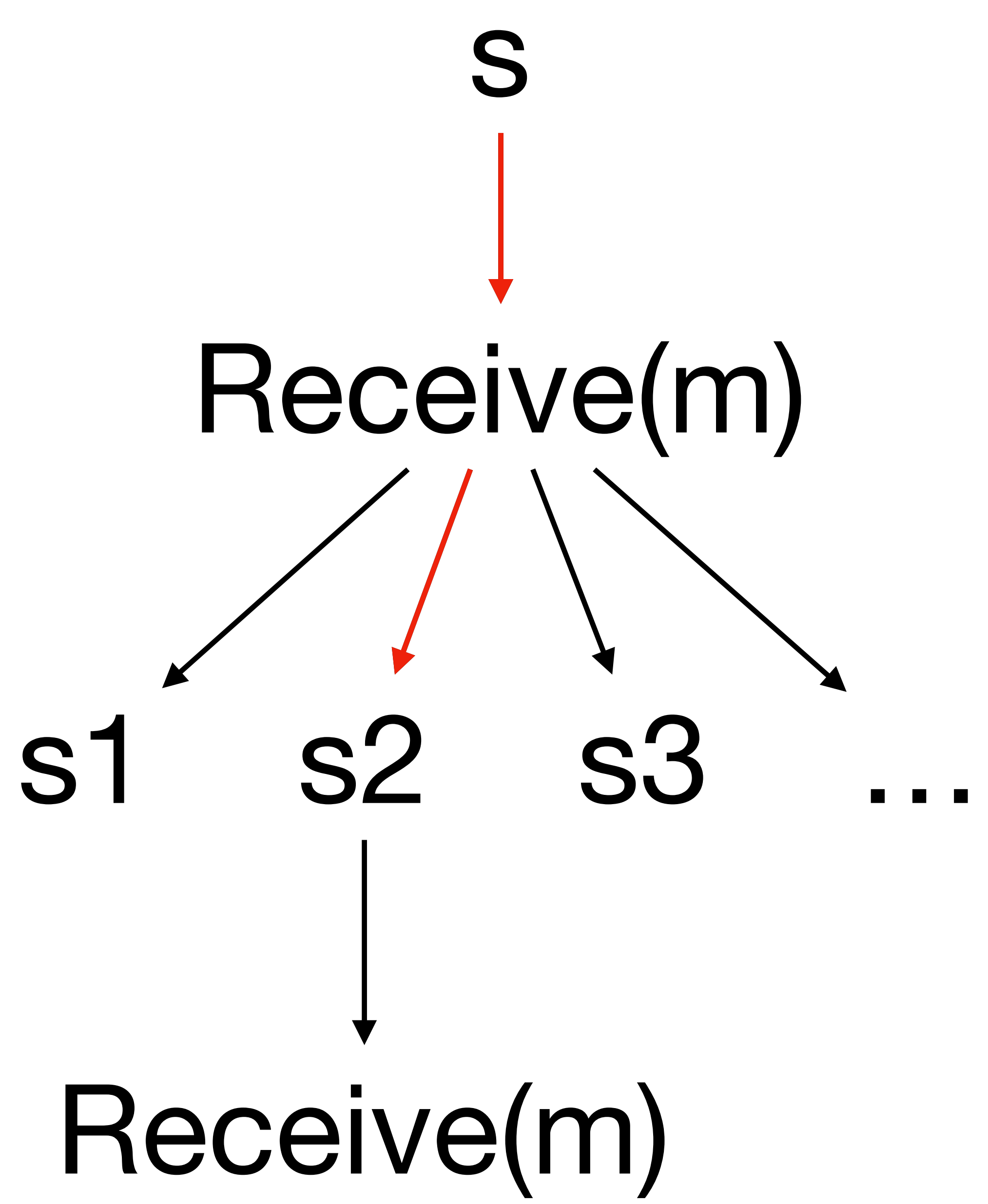
RequestVote(p1,p2)

Enumerating actions



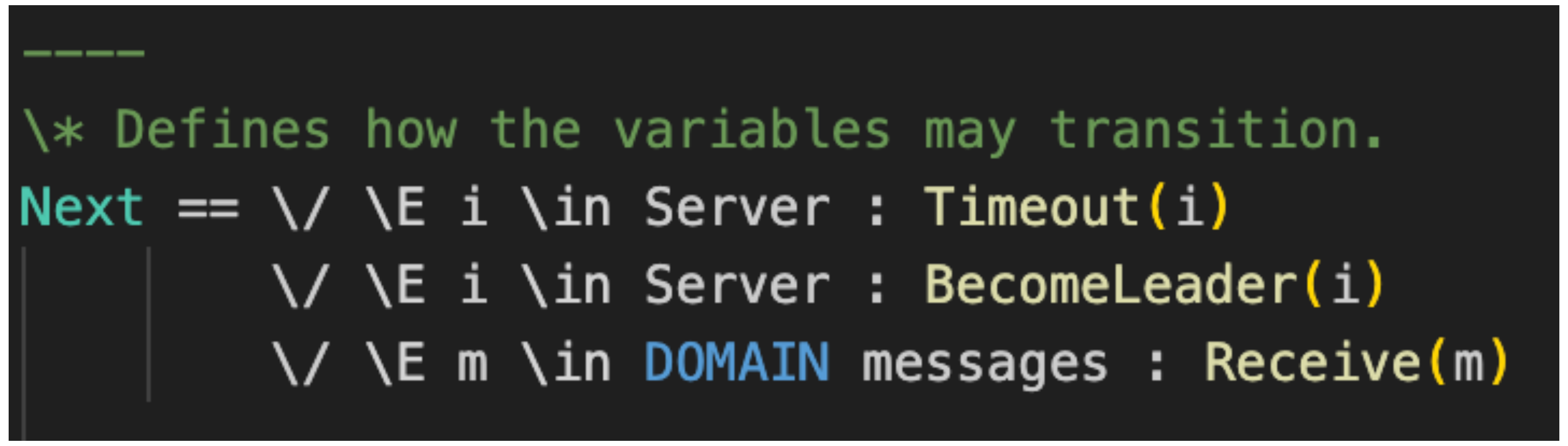
```
----
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E m \in DOMAIN messages : Receive(m)
```



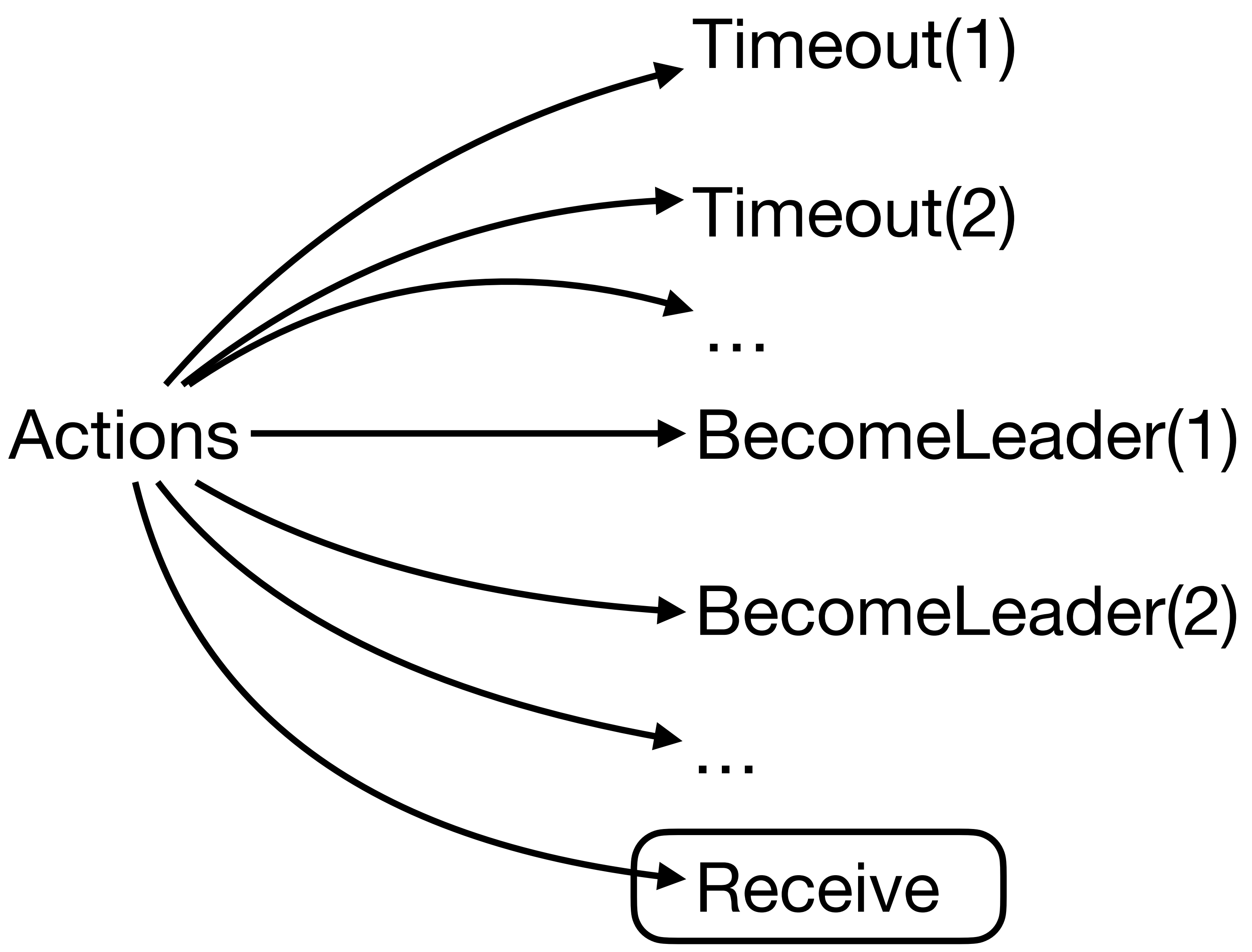


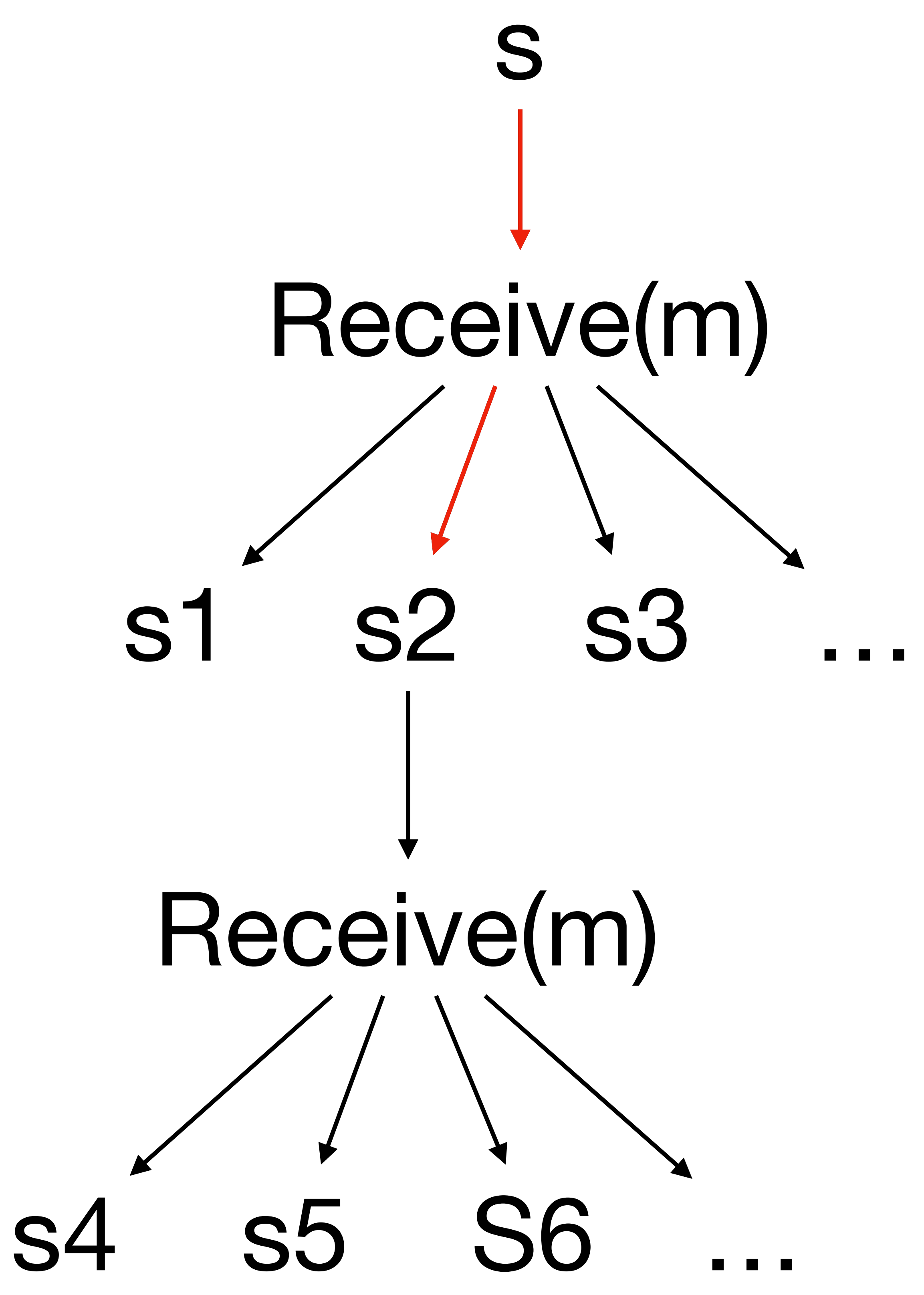
RequestVote(p1,p2)

Enumerating actions



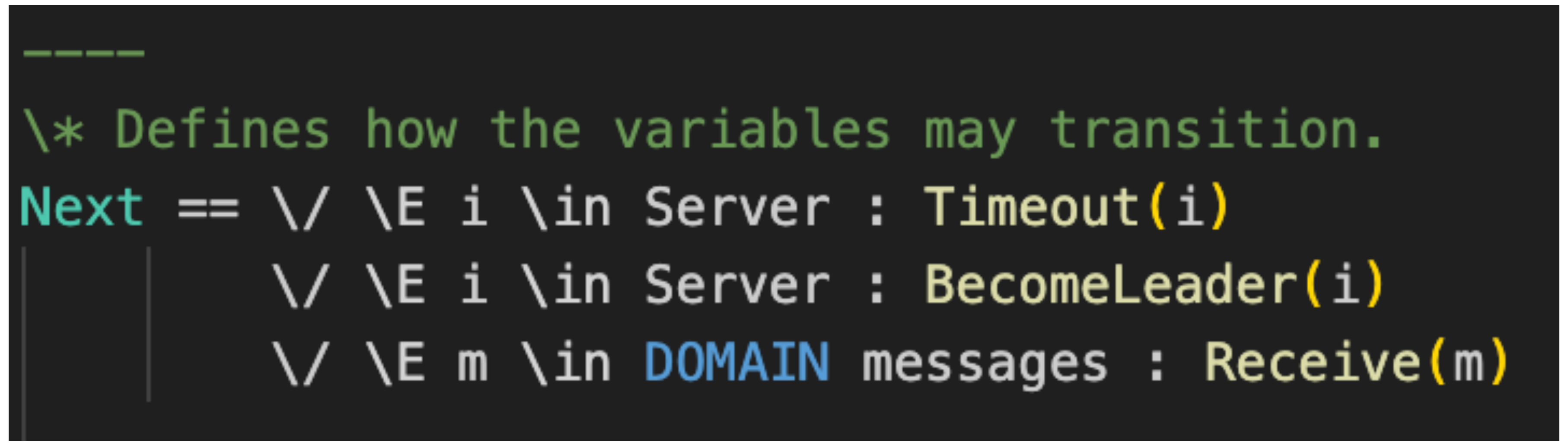
```
----
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E m \in DOMAIN messages : Receive(m)
```



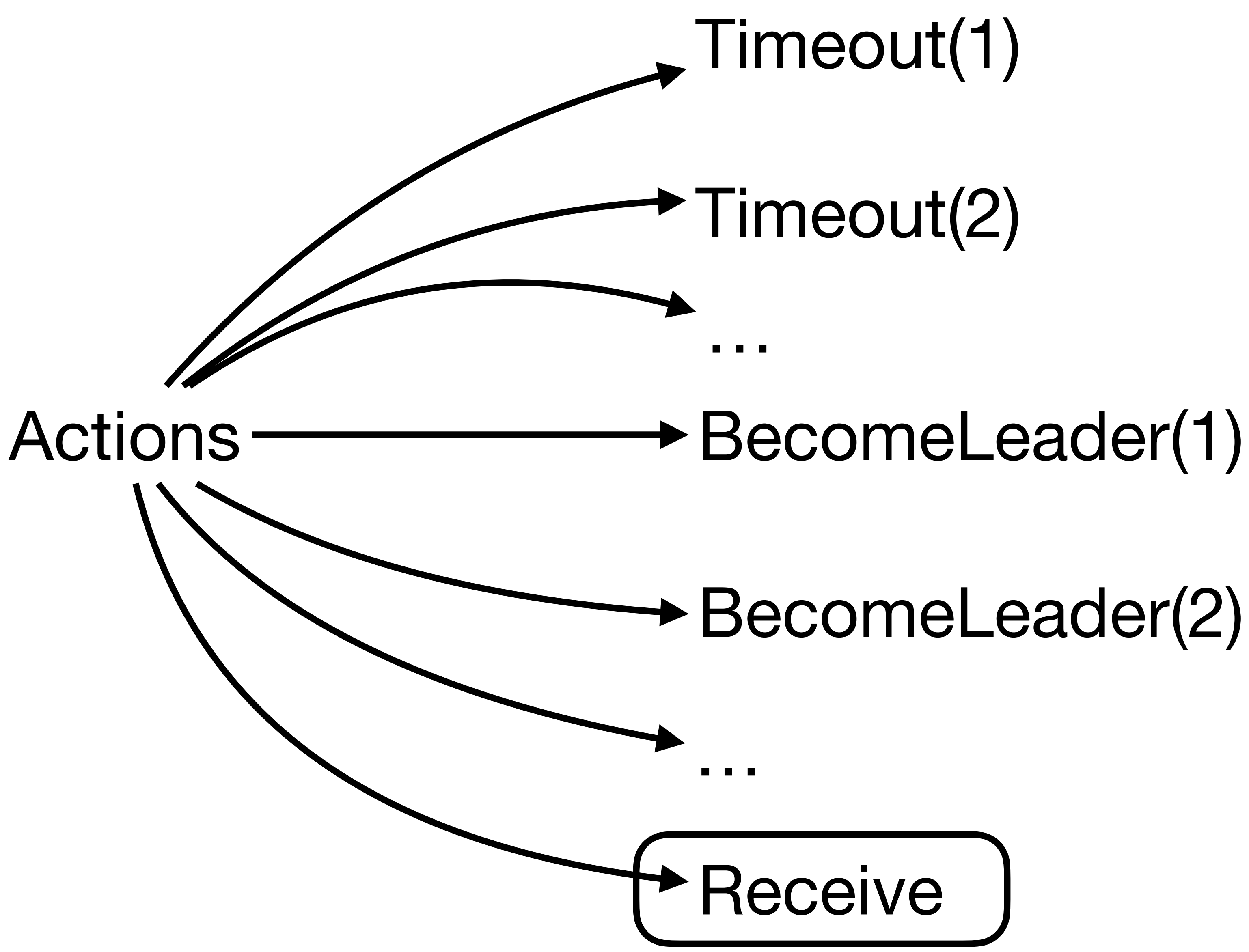


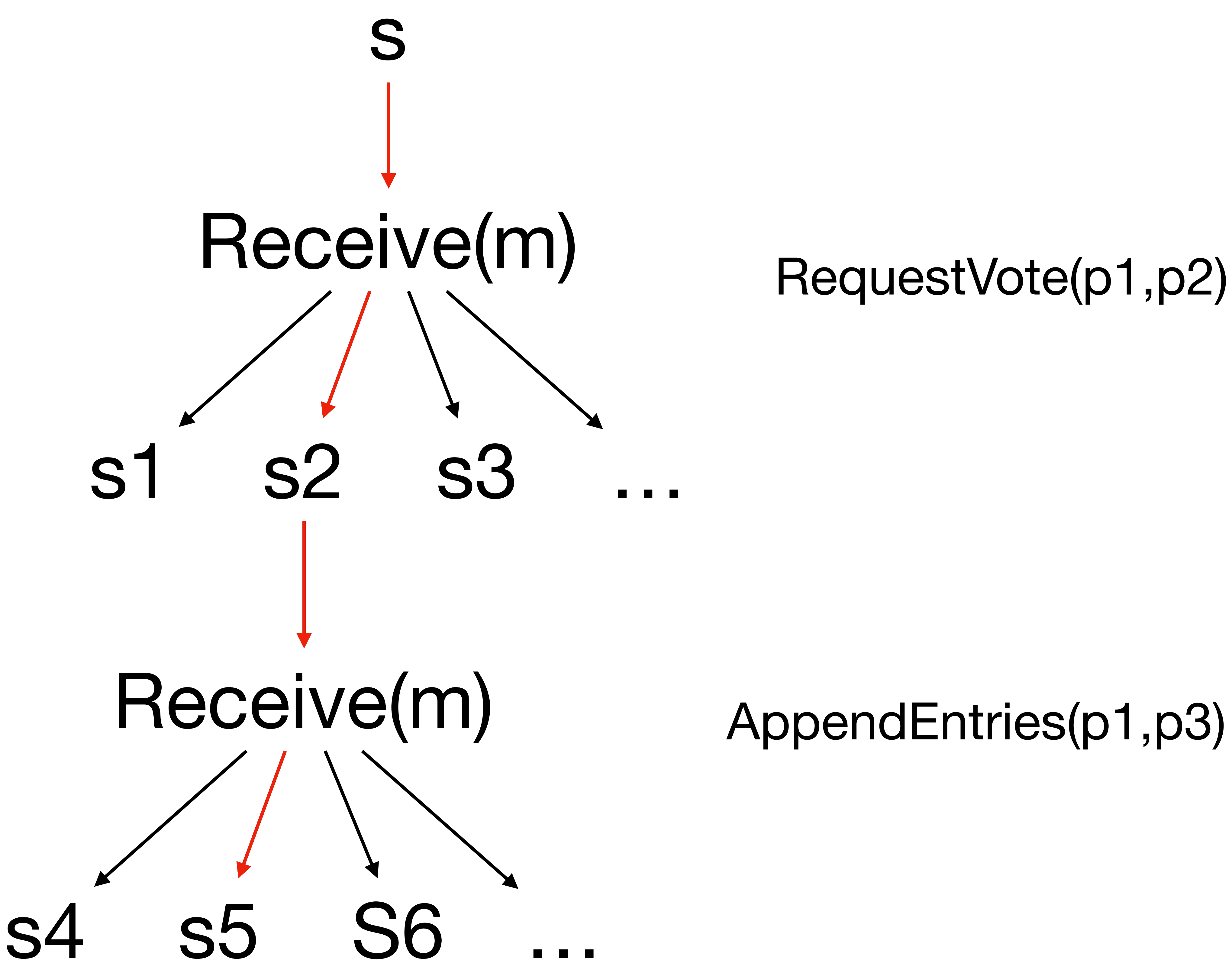
RequestVote(p1,p2)

Enumerating actions

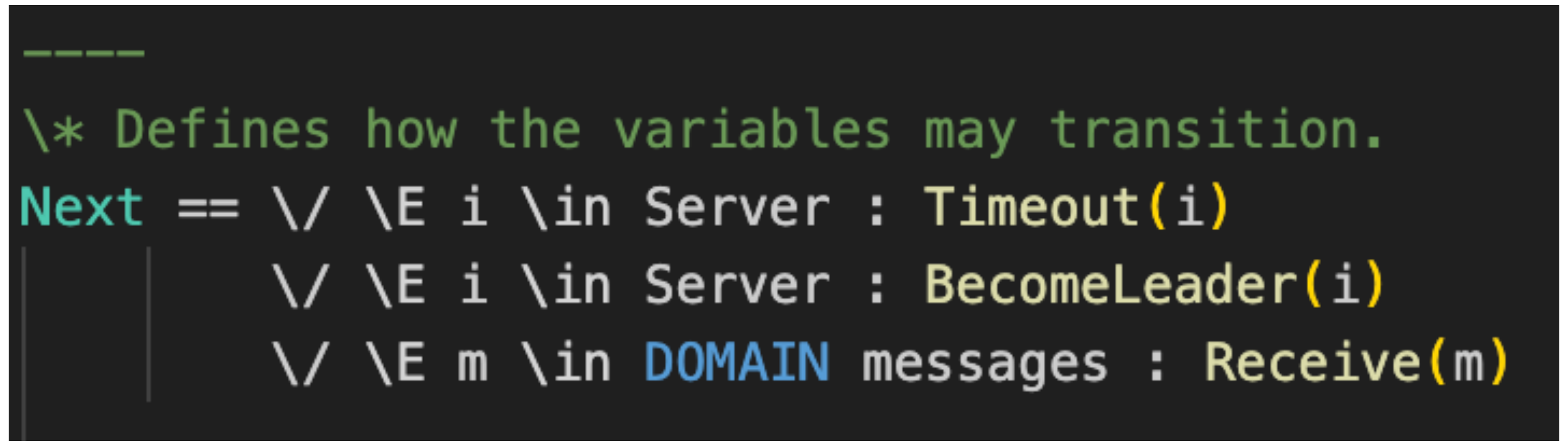


```
----
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E m \in DOMAIN messages : Receive(m)
```

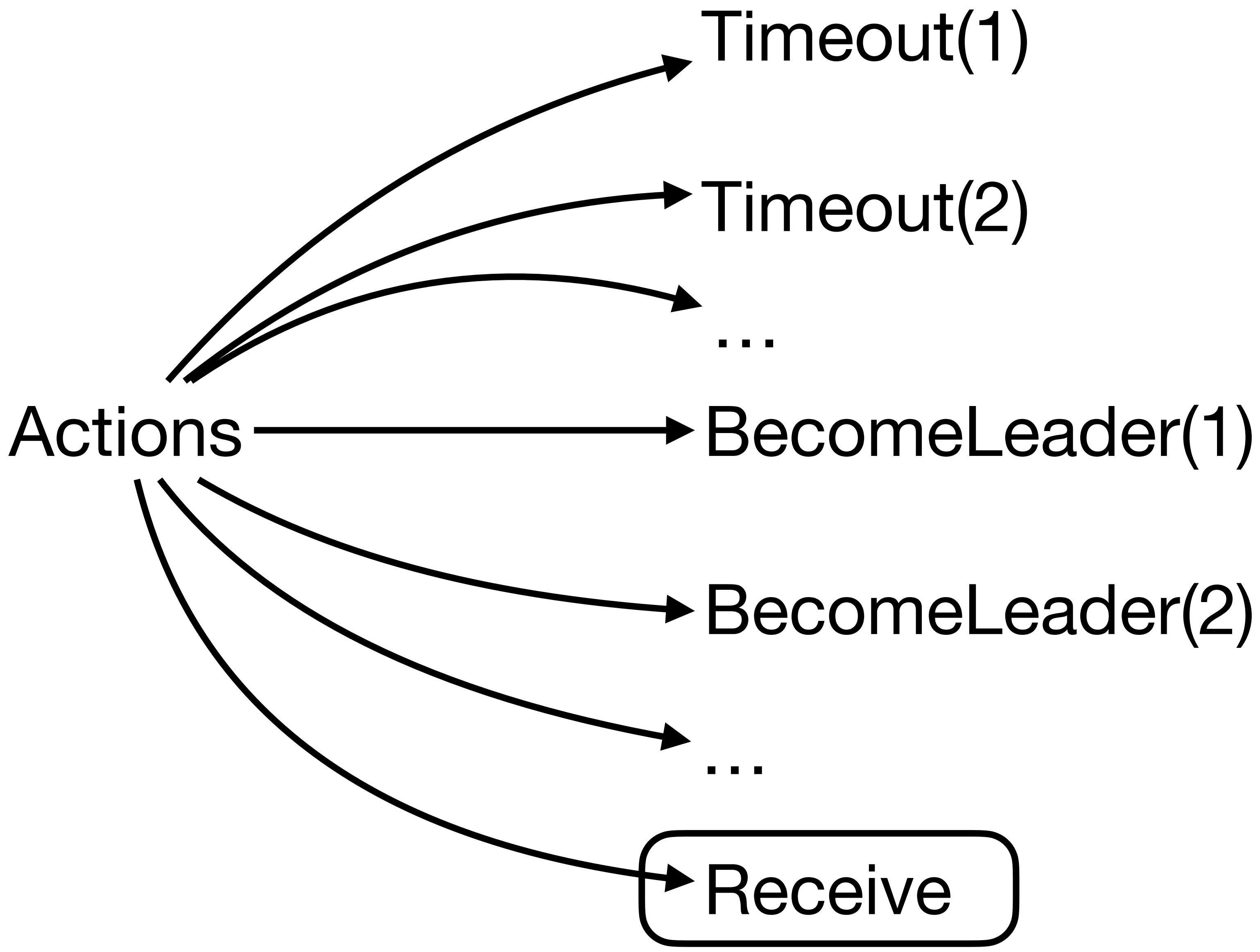


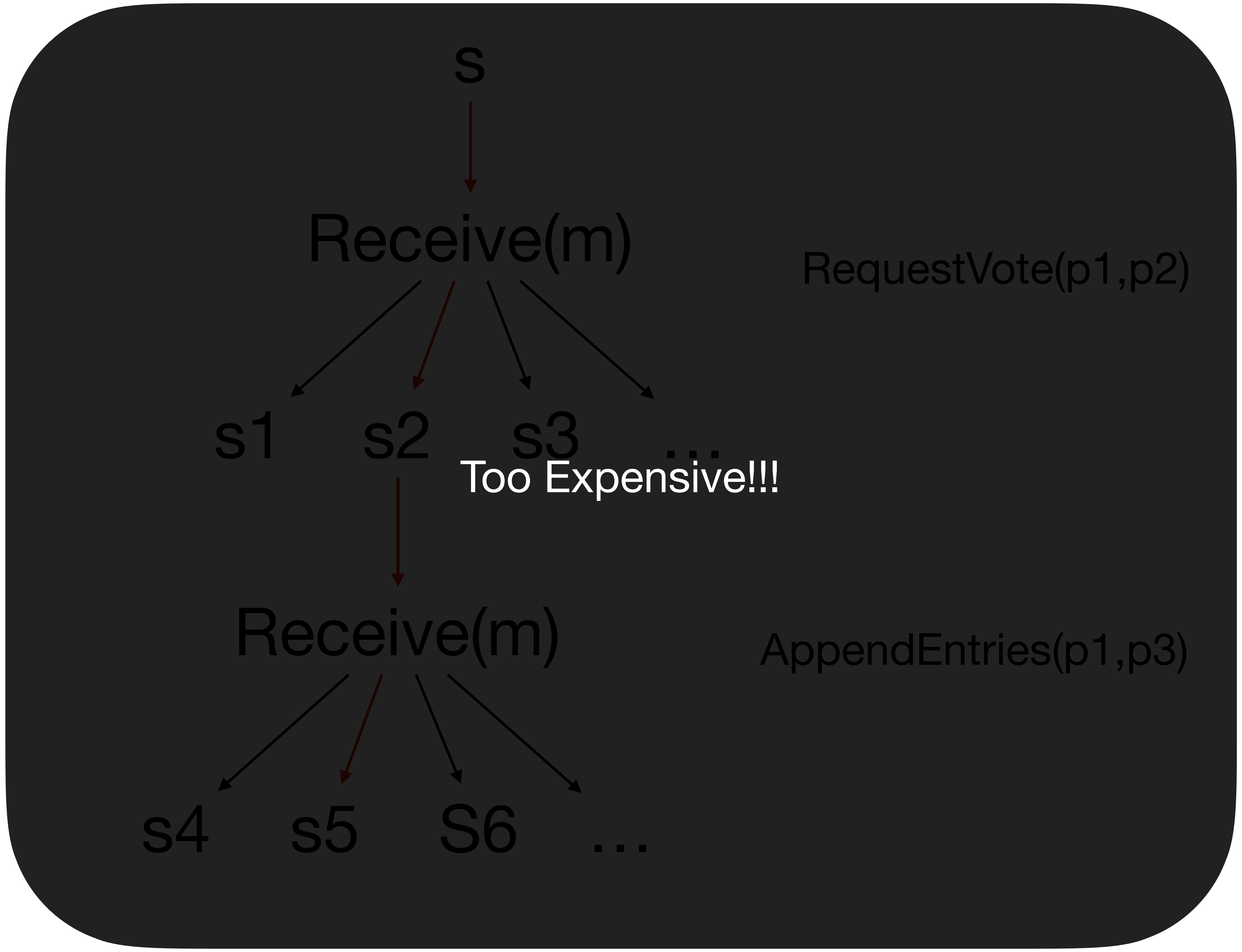


Enumerating actions



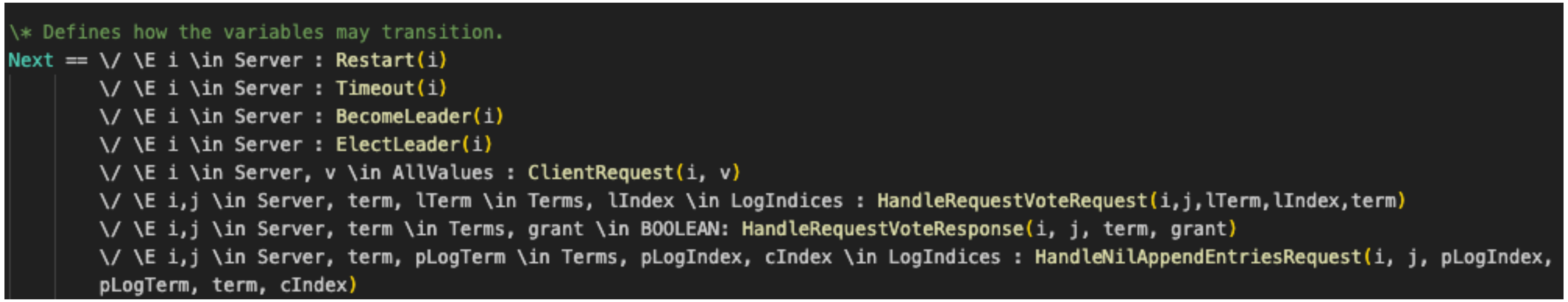
```
----
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E m \in DOMAIN messages : Receive(m)
```





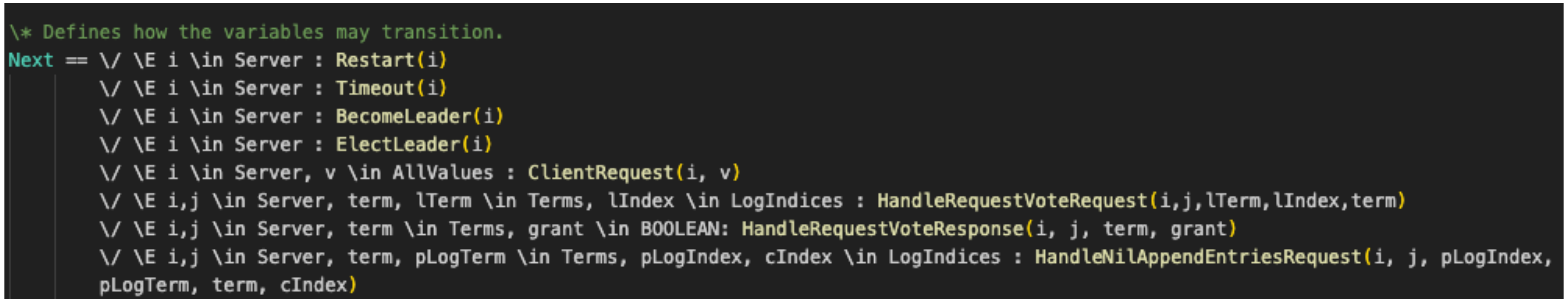
Too Expensive!!!

Enumerating actions

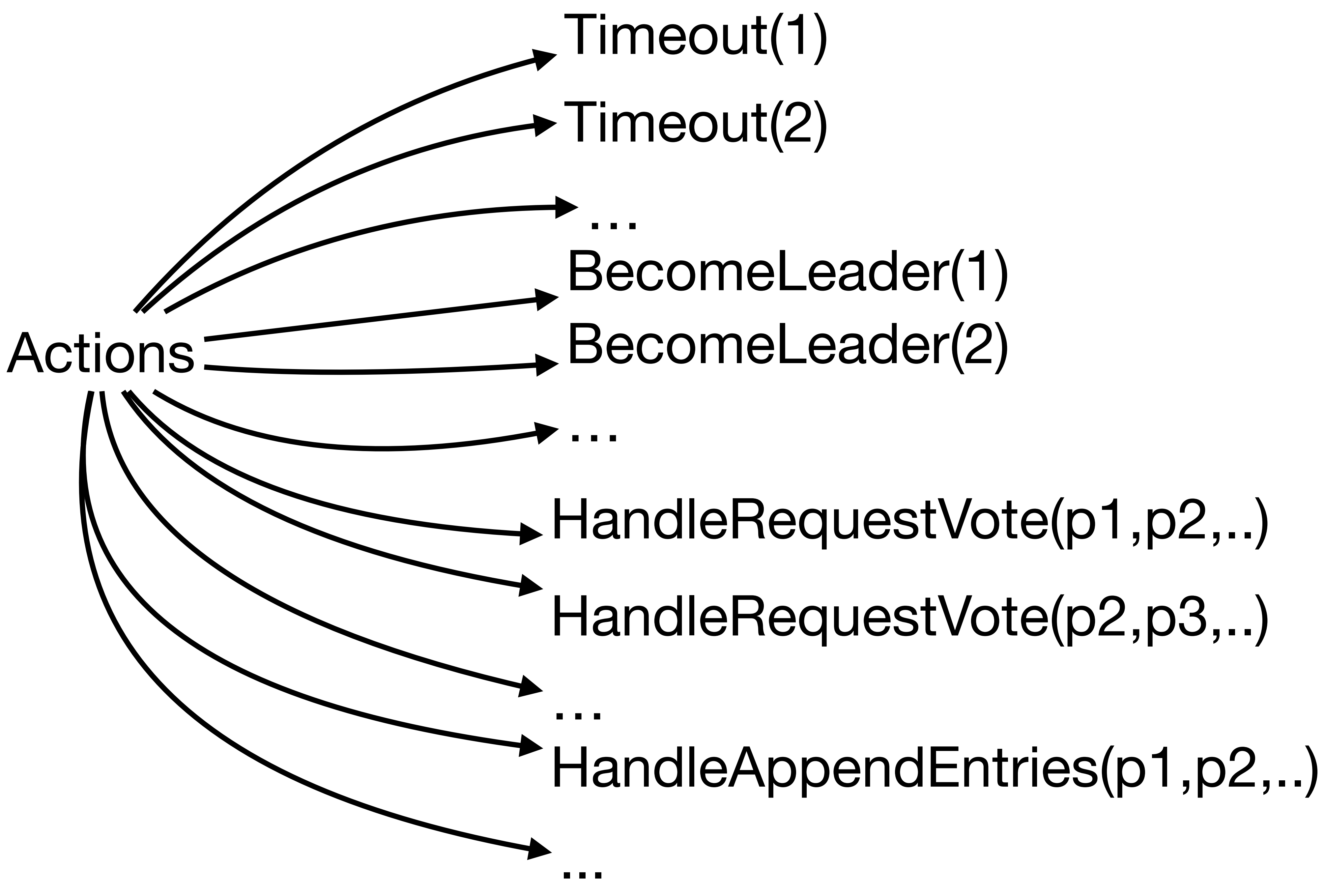


```
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Restart(i)
        \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E i \in Server : ElectLeader(i)
        \/ \E i \in Server, v \in AllValues : ClientRequest(i, v)
        \/ \E i,j \in Server, term, lTerm \in Terms, lIndex \in LogIndices : HandleRequestVoteRequest(i,j,lTerm,lIndex,term)
        \/ \E i,j \in Server, term \in Terms, grant \in BOOLEAN: HandleRequestVoteResponse(i, j, term, grant)
        \/ \E i,j \in Server, term, pLogTerm \in Terms, pLogIndex, cIndex \in LogIndices : HandleNilAppendEntriesRequest(i, j, pLogIndex,
        pLogTerm, term, cIndex)
```

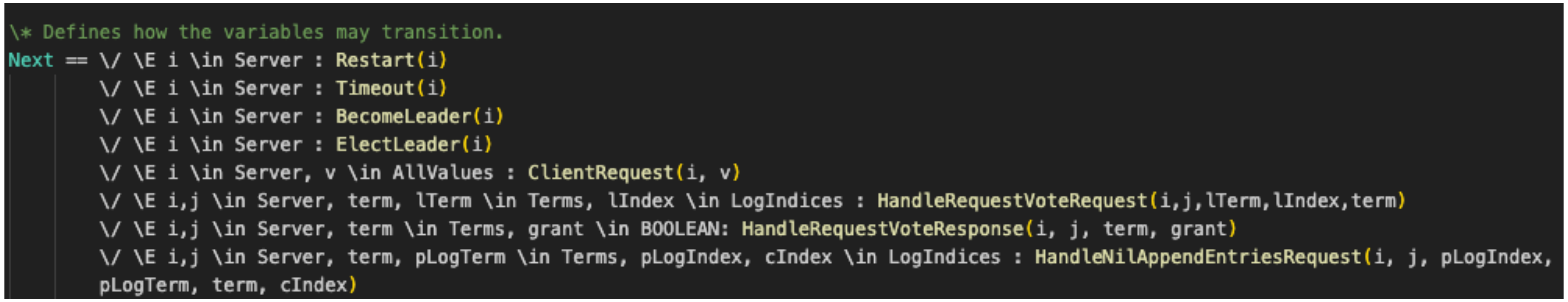
Enumerating actions



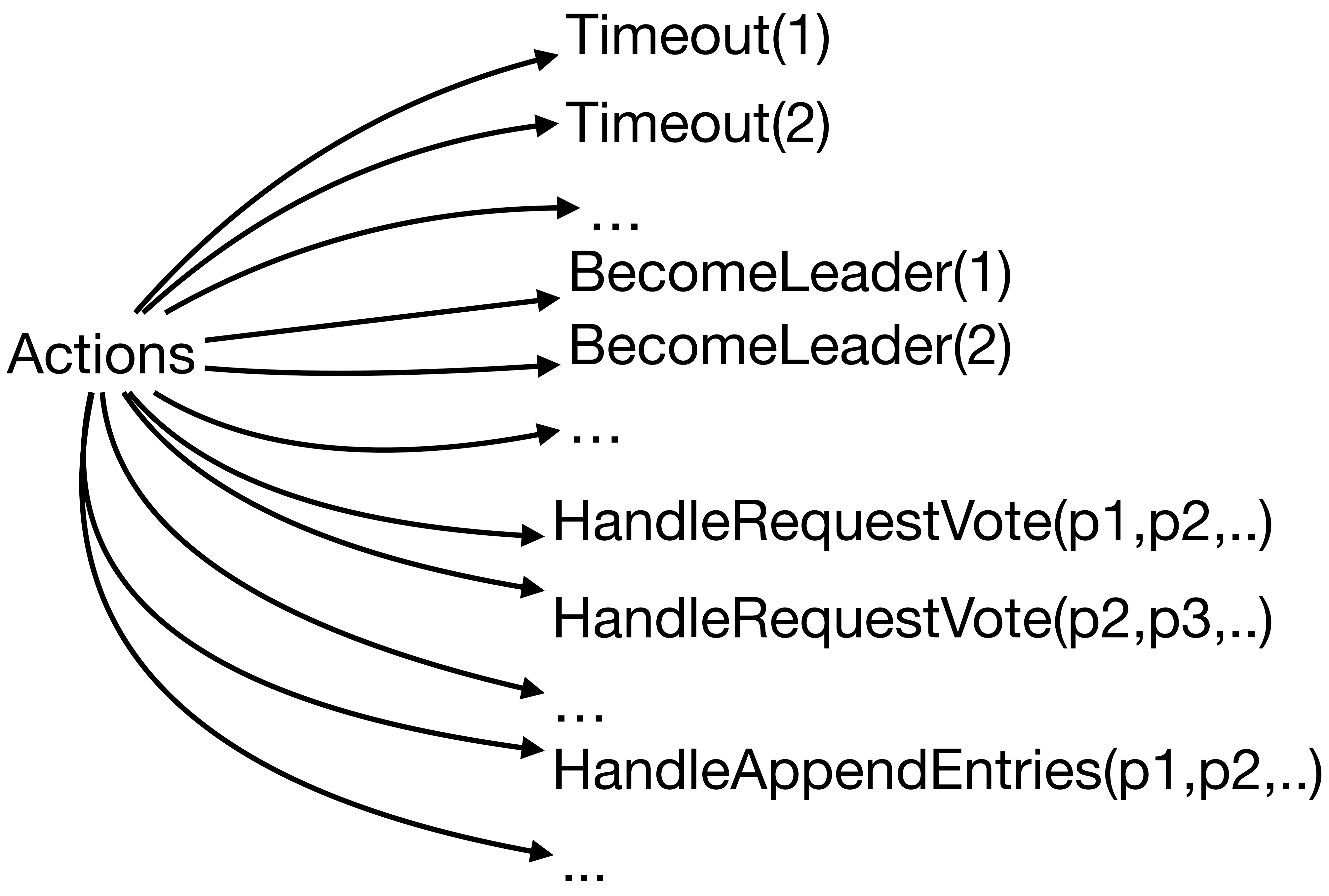
```
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Restart(i)
        \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E i \in Server : ElectLeader(i)
        \/ \E i \in Server, v \in AllValues : ClientRequest(i, v)
        \/ \E i,j \in Server, term, lTerm \in Terms, lIndex \in LogIndices : HandleRequestVoteRequest(i,j,lTerm,lIndex,term)
        \/ \E i,j \in Server, term \in Terms, grant \in BOOLEAN: HandleRequestVoteResponse(i, j, term, grant)
        \/ \E i,j \in Server, term, pLogTerm \in Terms, pLogIndex, cIndex \in LogIndices : HandleNilAppendEntriesRequest(i, j, pLogIndex,
        pLogTerm, term, cIndex)
```


Actions
Timeout(1)
Timeout(2)
…
BecomeLeader(1)
BecomeLeader(2)
…
HandleRequestVote(p1,p2,..)
HandleRequestVote(p2,p3,..)
…
HandleAppendEntries(p1,p2,..)
…

Enumerating actions



```
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Restart(i)
        \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E i \in Server : ElectLeader(i)
        \/ \E i \in Server, v \in AllValues : ClientRequest(i, v)
        \/ \E i,j \in Server, term, lTerm \in Terms, lIndex \in LogIndices : HandleRequestVoteRequest(i,j,lTerm,lIndex,term)
        \/ \E i,j \in Server, term \in Terms, grant \in BOOLEAN: HandleRequestVoteResponse(i, j, term, grant)
        \/ \E i,j \in Server, term, pLogTerm \in Terms, pLogIndex, cIndex \in LogIndices : HandleNilAppendEntriesRequest(i, j, pLogIndex,
        pLogTerm, term, cIndex)
```



Actions

Timeout(1)

Timeout(2)

…

BecomeLeader(1)

BecomeLeader(2)

…

HandleRequestVote(p1,p2,..)

HandleRequestVote(p2,p3,..)

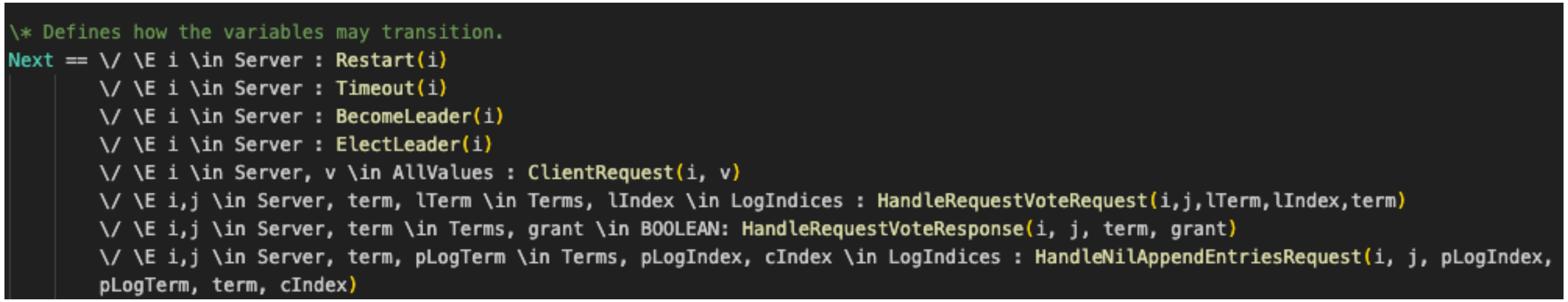
…

HandleAppendEntries(p1,p2,..)

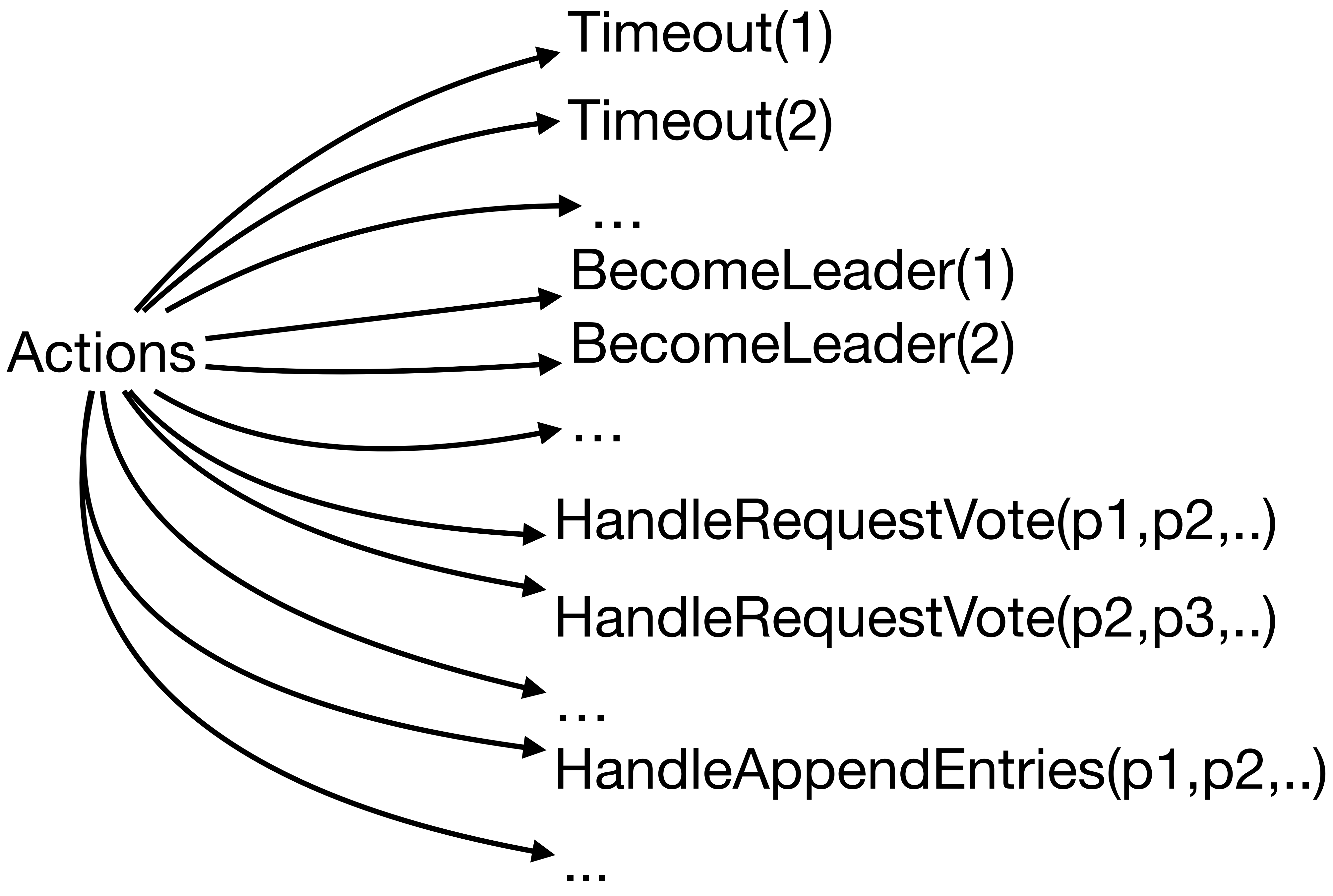
…

- Map and store all actions

Enumerating actions

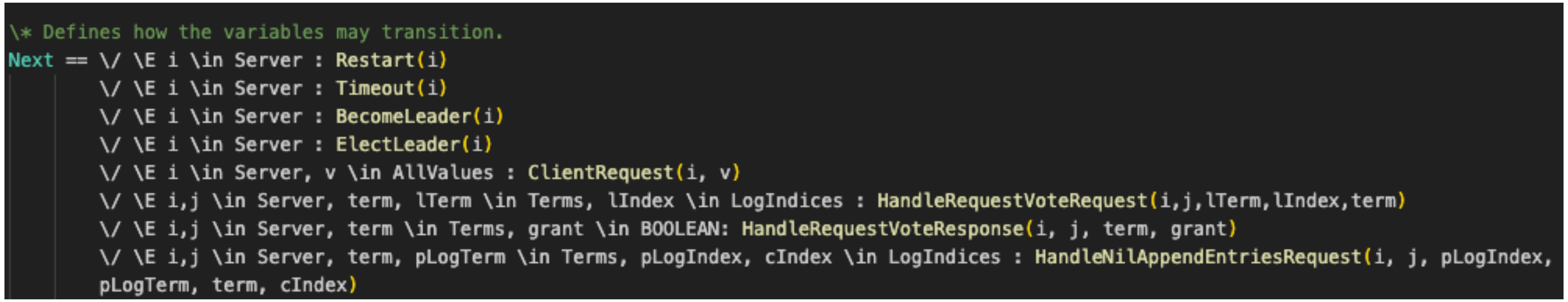


```
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Restart(i)
        \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E i \in Server : ElectLeader(i)
        \/ \E i \in Server, v \in AllValues : ClientRequest(i, v)
        \/ \E i,j \in Server, term, lTerm \in Terms, lIndex \in LogIndices : HandleRequestVoteRequest(i,j,lTerm,lIndex,term)
        \/ \E i,j \in Server, term \in Terms, grant \in BOOLEAN: HandleRequestVoteResponse(i, j, term, grant)
        \/ \E i,j \in Server, term, pLogTerm \in Terms, pLogIndex, cIndex \in LogIndices : HandleNilAppendEntriesRequest(i, j, pLogIndex,
        pLogTerm, term, cIndex)
```

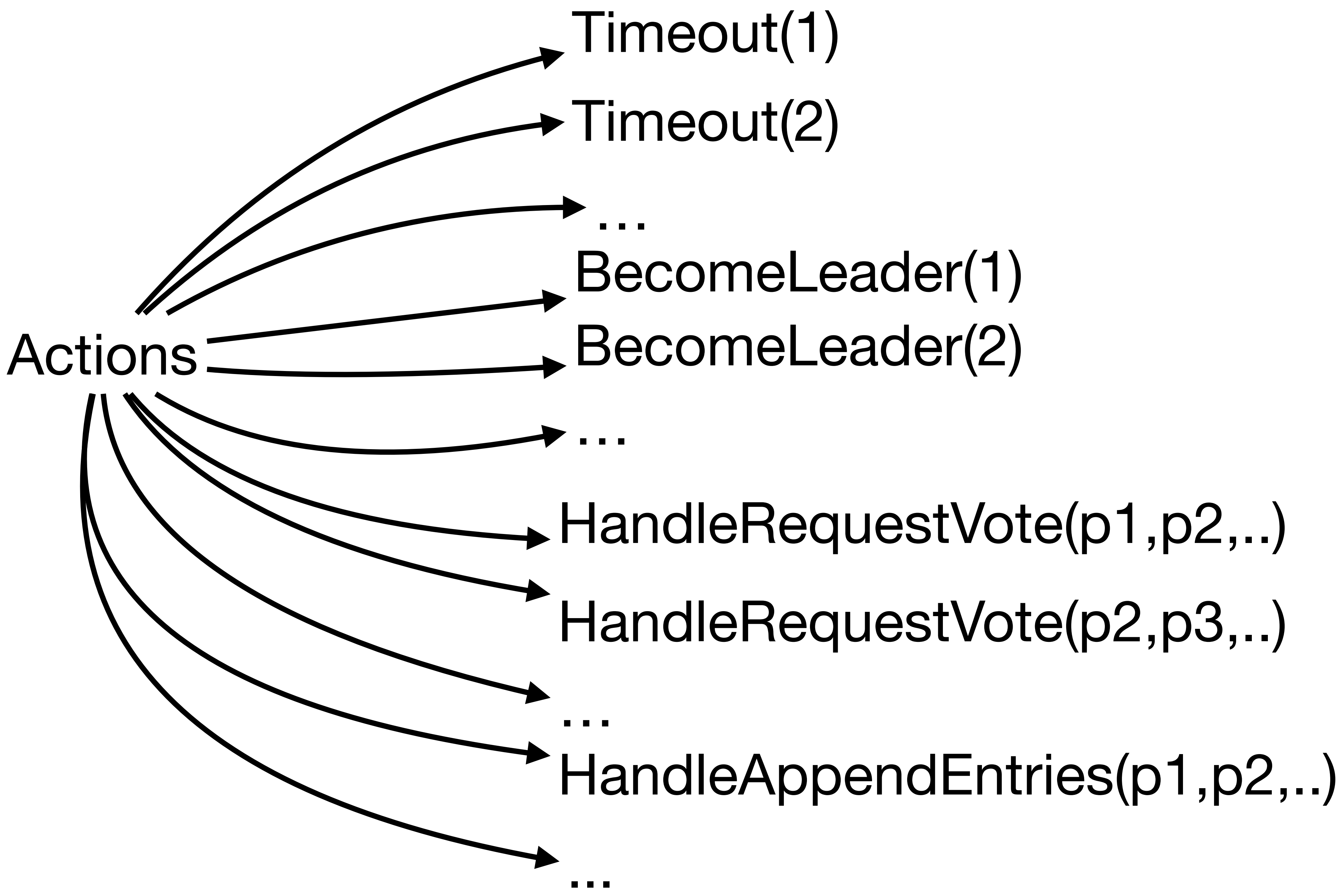

Timeout(1)
Timeout(2)
…
BecomeLeader(1)
BecomeLeader(2)
Actions
…
HandleRequestVote(p1,p2,..)
HandleRequestVote(p2,p3,..)
…
HandleAppendEntries(p1,p2,..)
…

- Map and store all actions
- Simulating is linear in length and fast

Enumerating actions

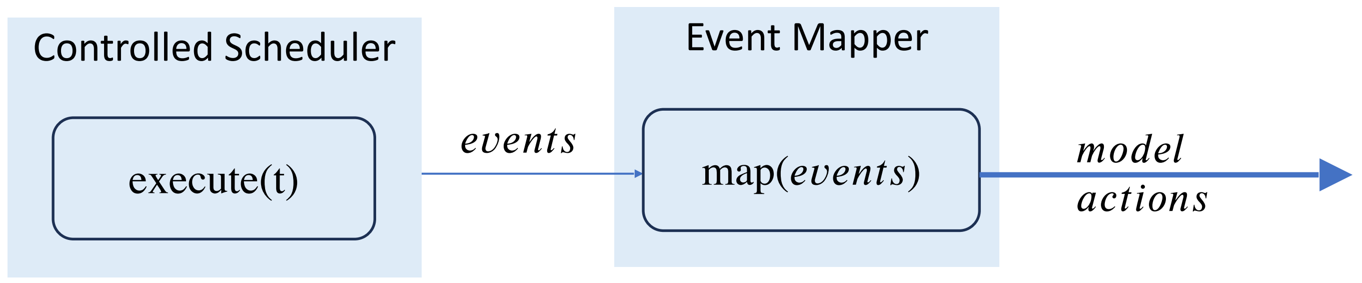


```
\* Defines how the variables may transition.
Next == \/ \E i \in Server : Restart(i)
        \/ \E i \in Server : Timeout(i)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E i \in Server : ElectLeader(i)
        \/ \E i \in Server, v \in AllValues : ClientRequest(i, v)
        \/ \E i,j \in Server, term, lTerm \in Terms, lIndex \in LogIndices : HandleRequestVoteRequest(i,j,lTerm,lIndex,term)
        \/ \E i,j \in Server, term \in Terms, grant \in BOOLEAN: HandleRequestVoteResponse(i, j, term, grant)
        \/ \E i,j \in Server, term, pLogTerm \in Terms, pLogIndex, cIndex \in LogIndices : HandleNilAppendEntriesRequest(i, j, pLogIndex,
        pLogTerm, term, cIndex)
```


Actions
Timeout(1)
Timeout(2)
…
BecomeLeader(1)
BecomeLeader(2)
…
HandleRequestVote(p1,p2,..)
HandleRequestVote(p2,p3,..)
…
HandleAppendEntries(p1,p2,..)
…

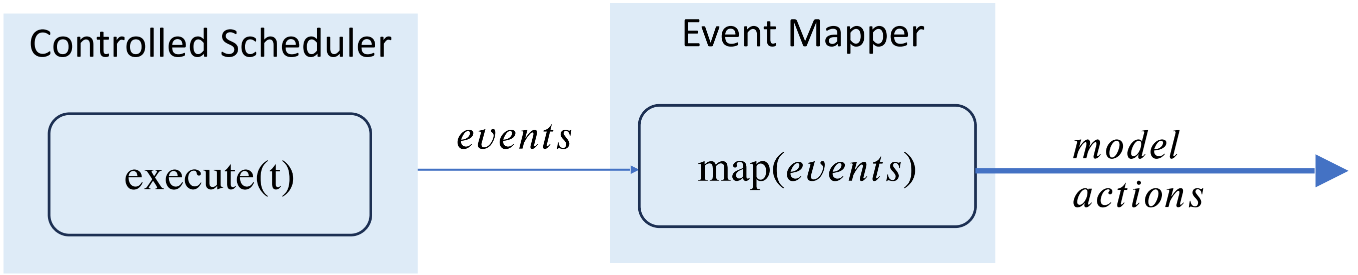
- Map and store all actions
- Simulating is linear in length and fast
- Needs a lot of space

Mapping actions



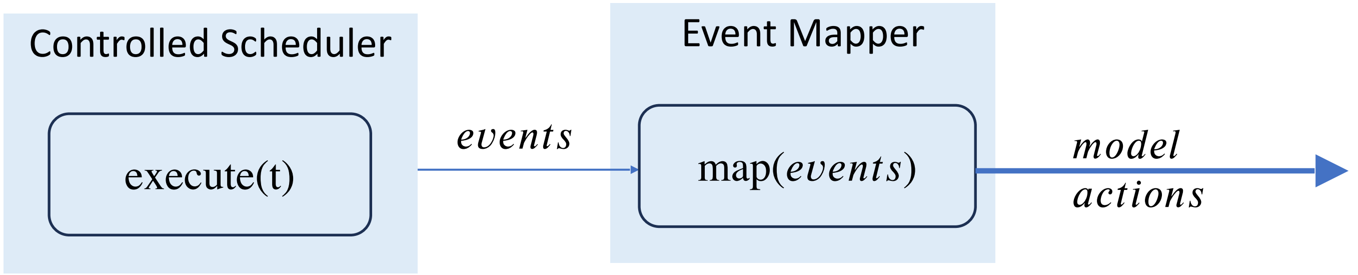
Mapping actions

- The action sequence needs an abstraction



Mapping actions

- The action sequence needs an abstraction
- Only those actions that affect the state represented in the model

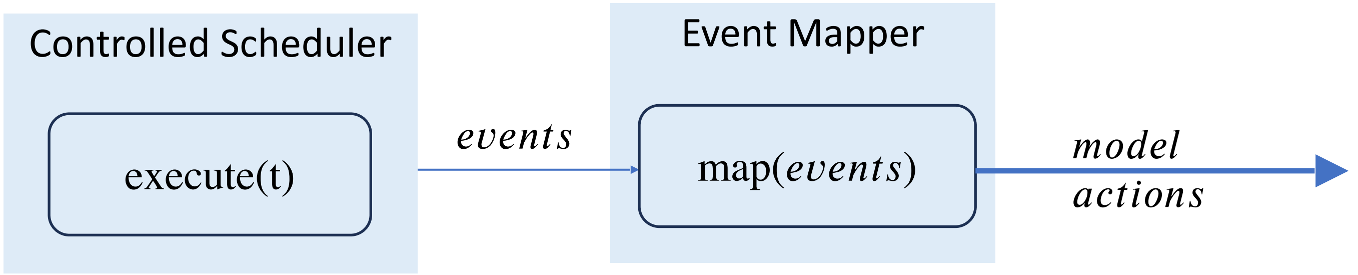


Mapping actions

- The action sequence needs an abstraction

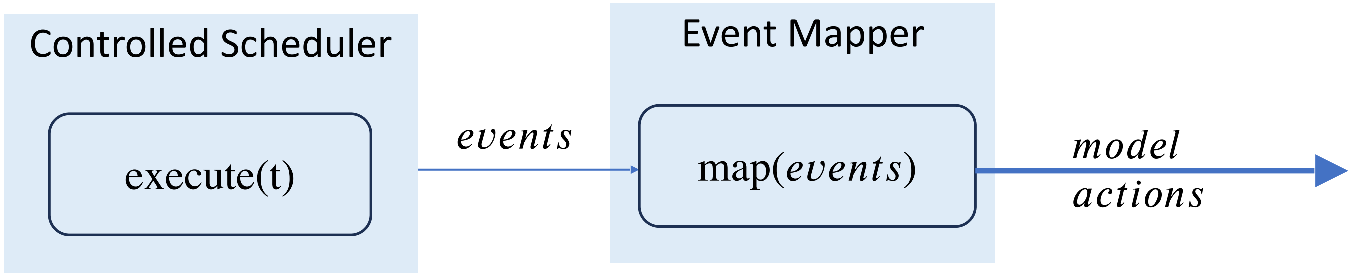
- Only those actions that affect the state represented in the model

- Eg. Heartbeat messages can be ignored



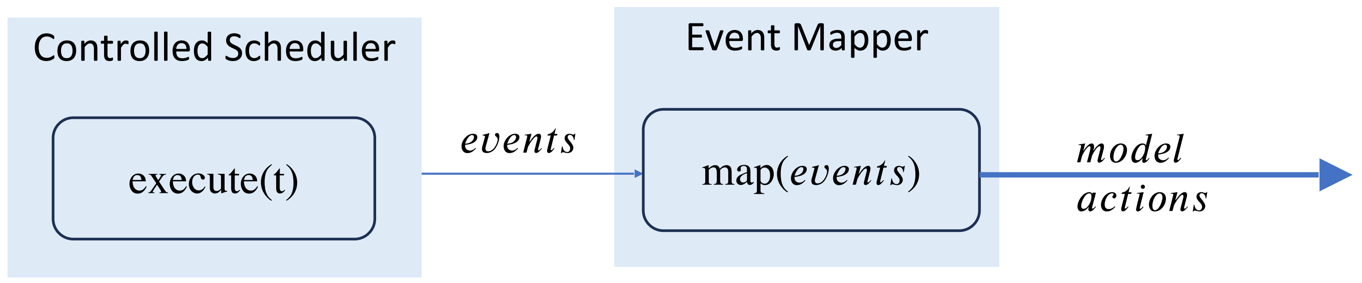
Mapping actions

- The action sequence needs an abstraction
- Only those actions that affect the state represented in the model
- Eg. Heartbeat messages can be ignored
- Specific to each implementation

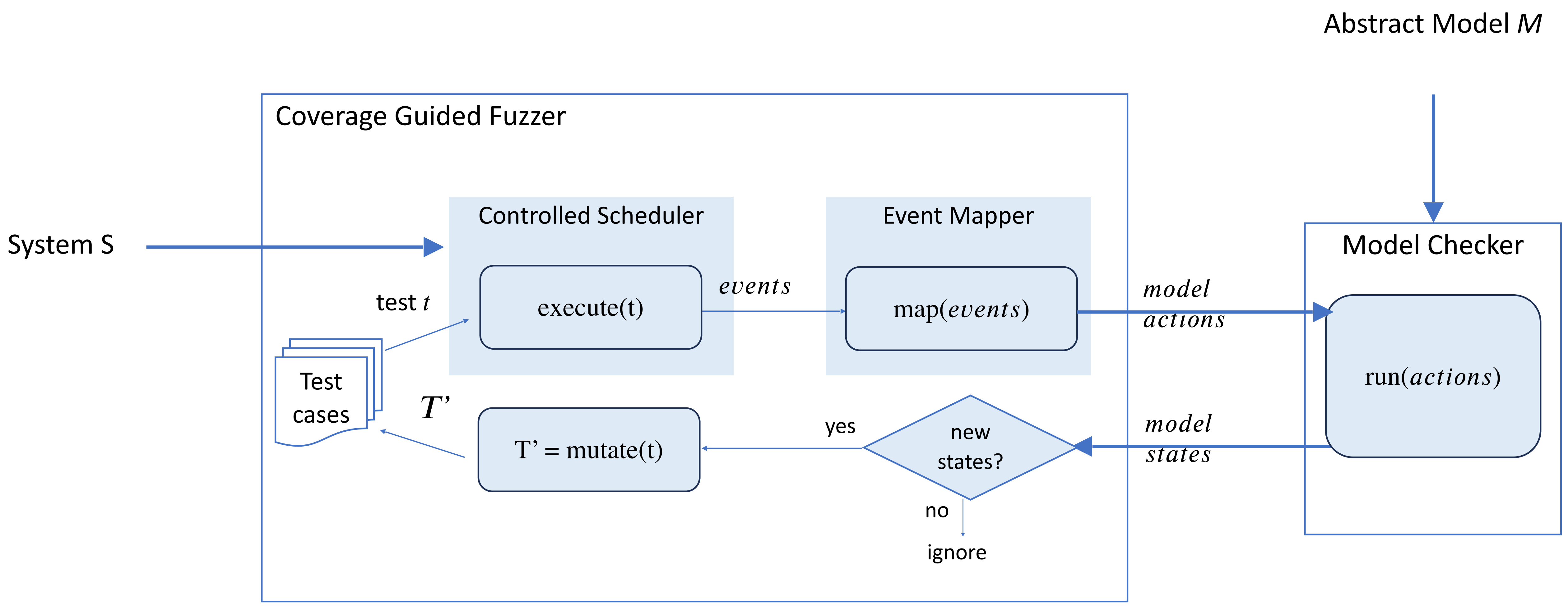


Mapping actions

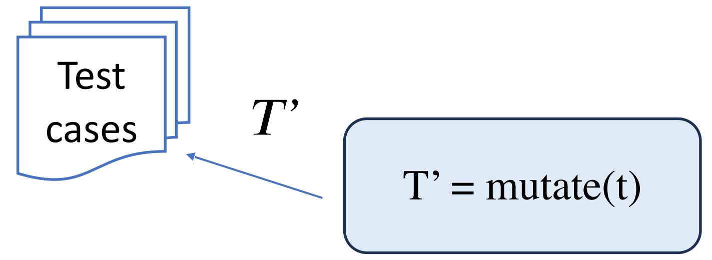
- The action sequence needs an abstraction
- Only those actions that affect the state represented in the model
- Eg. Heartbeat messages can be ignored
- Specific to each implementation
    - Can be generalised to each protocol (modulo different data structures)



Overall picture

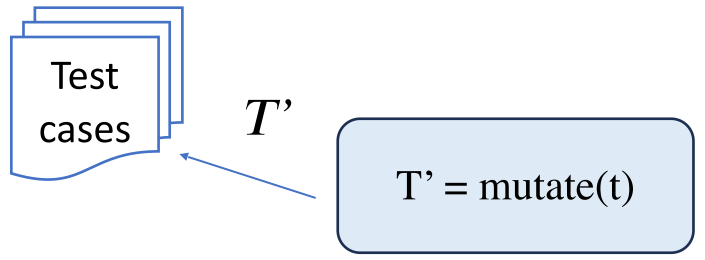


Mutation strategies



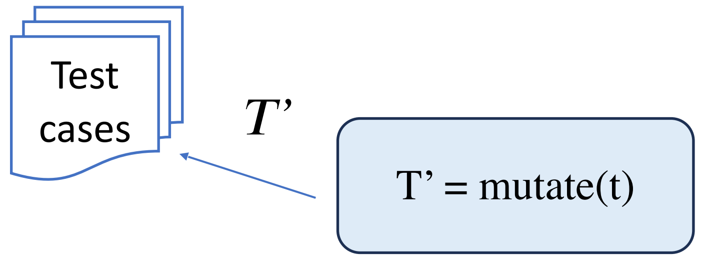
Mutation strategies

- Swaps



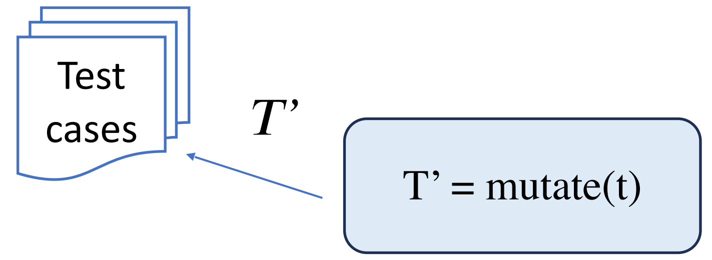
Mutation strategies

- Swaps
  - Swap scheduling choices (A different process becomes leader)



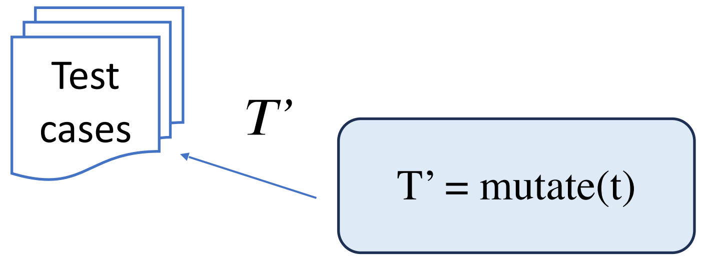
Mutation strategies

- Swaps
  - Swap scheduling choices (A different process becomes leader)
  - Swap crashes (crashing leader instead of a follower)

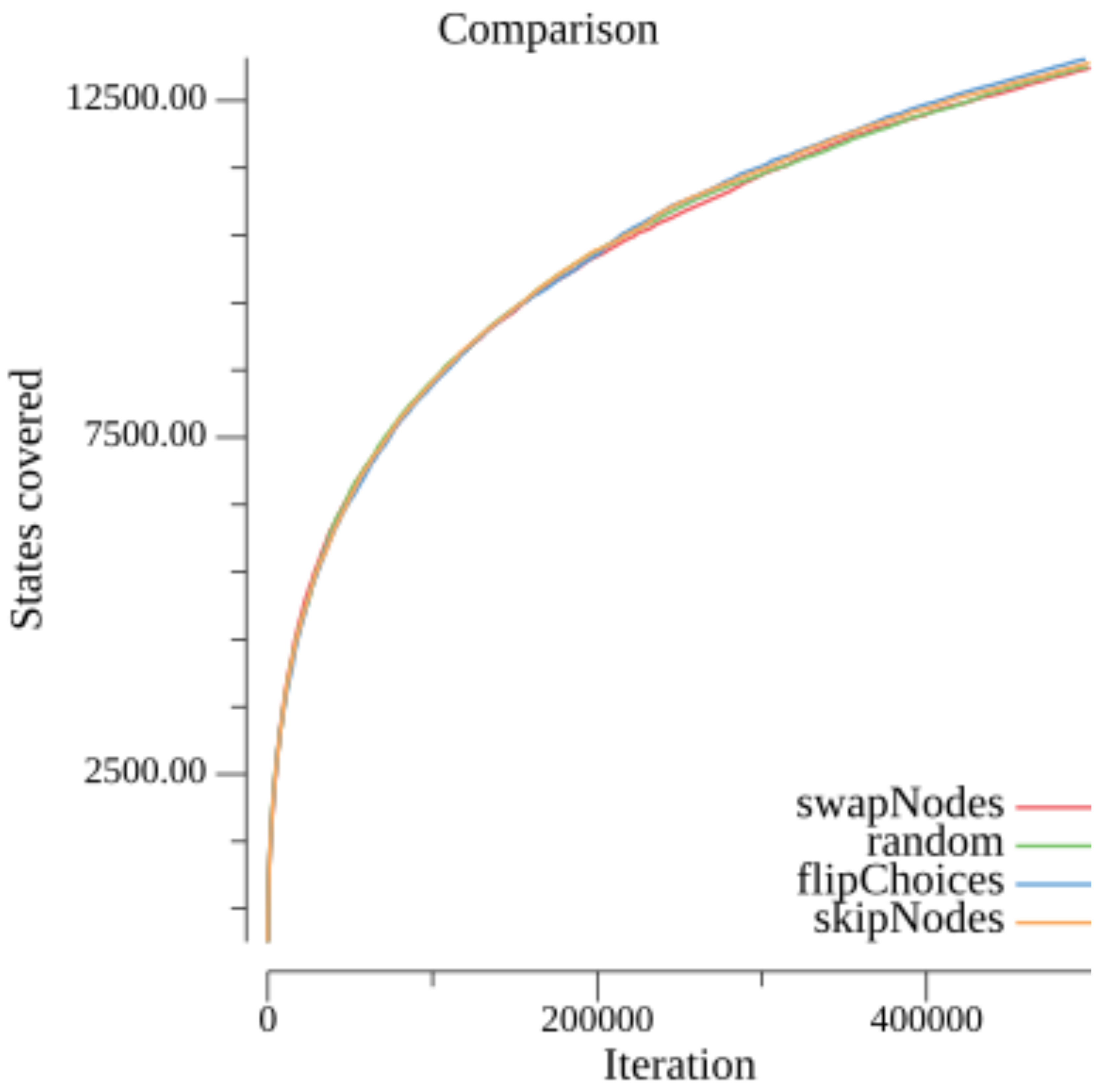


Mutation strategies

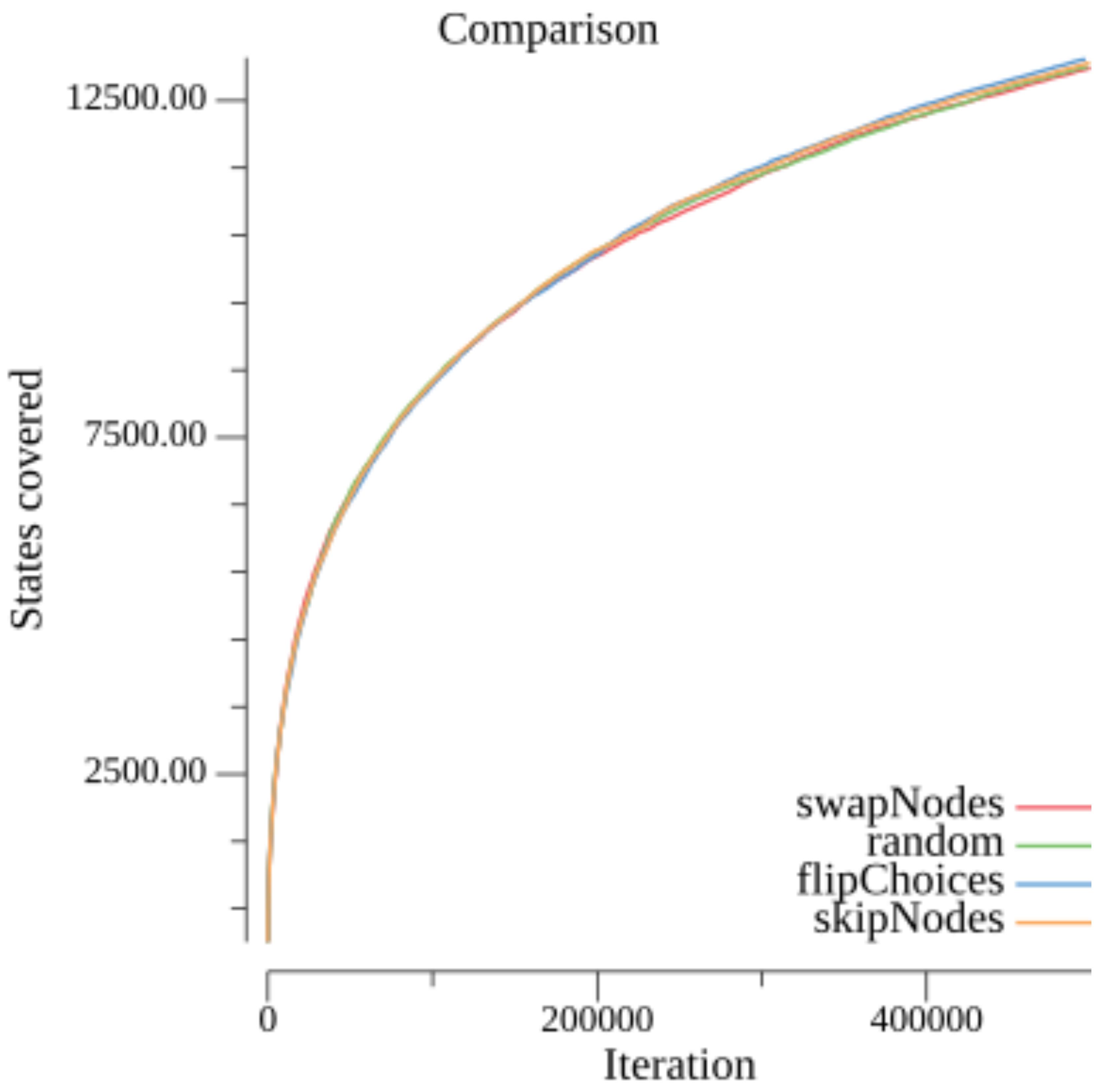
- Swaps
  - Swap scheduling choices (A different process becomes leader)
  - Swap crashes (crashing leader instead of a follower)
  - Swap number of messages delivered



Does it work?

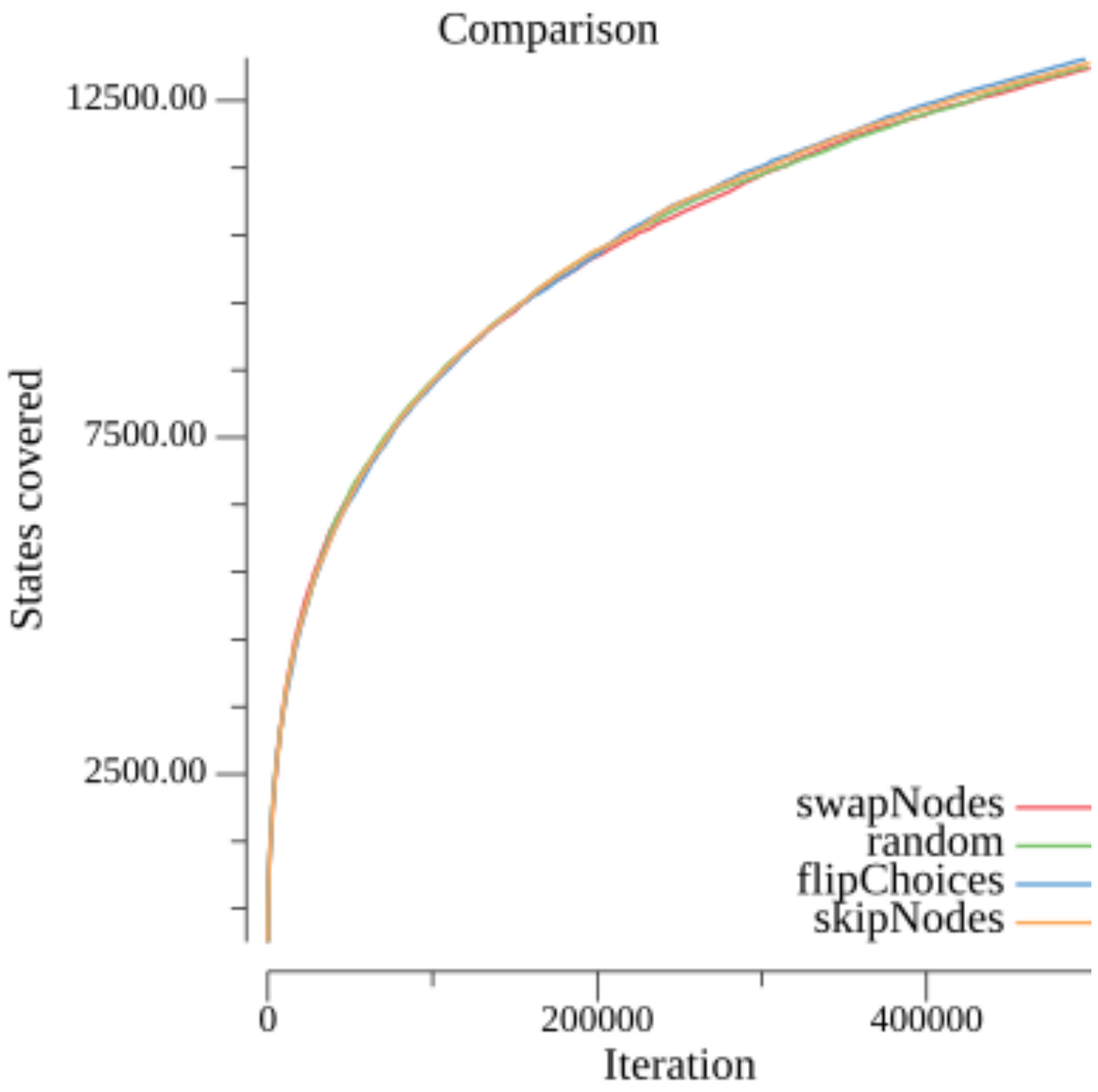

Comparison
States covered
12500.00
7500.00
2500.00
Iteration
0
200000
400000
swapNodes
random
flipChoices
skipNodes

Does it work?



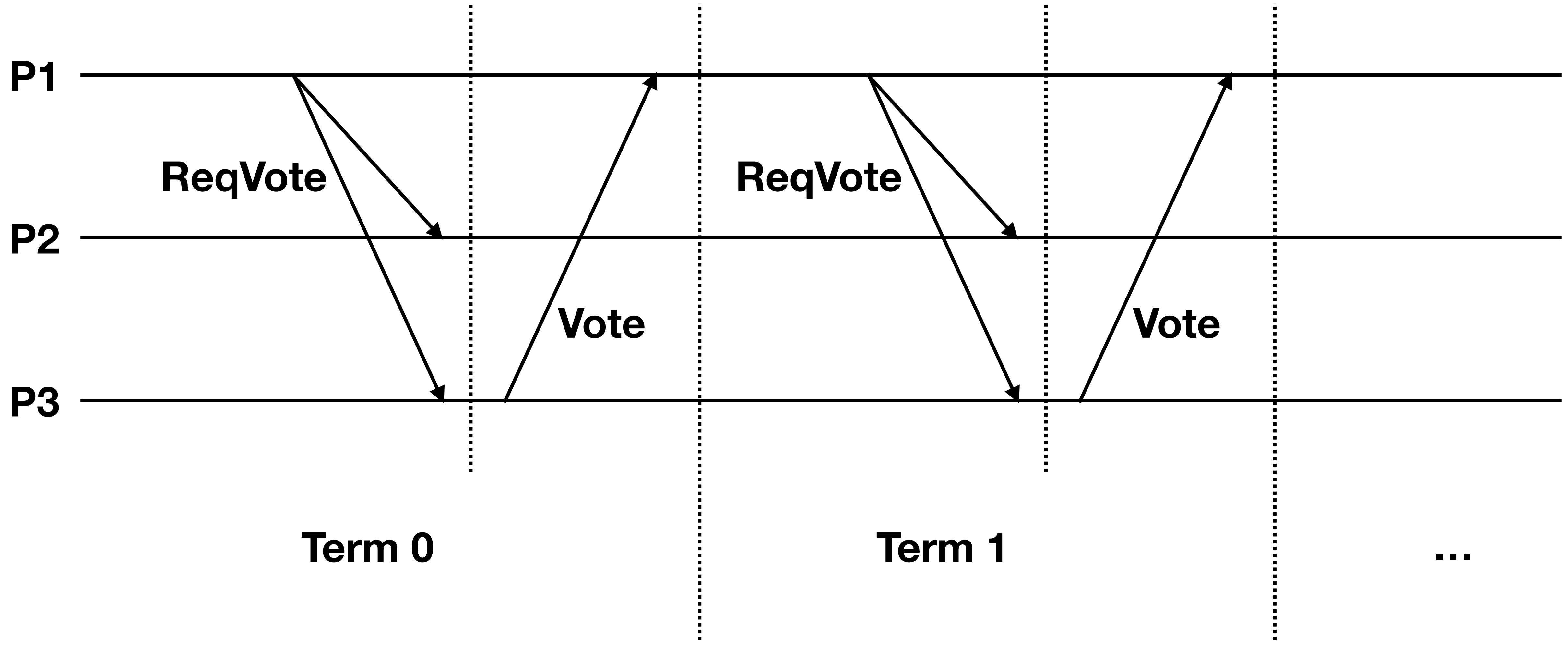
- Still can’t beat random

Does it work?

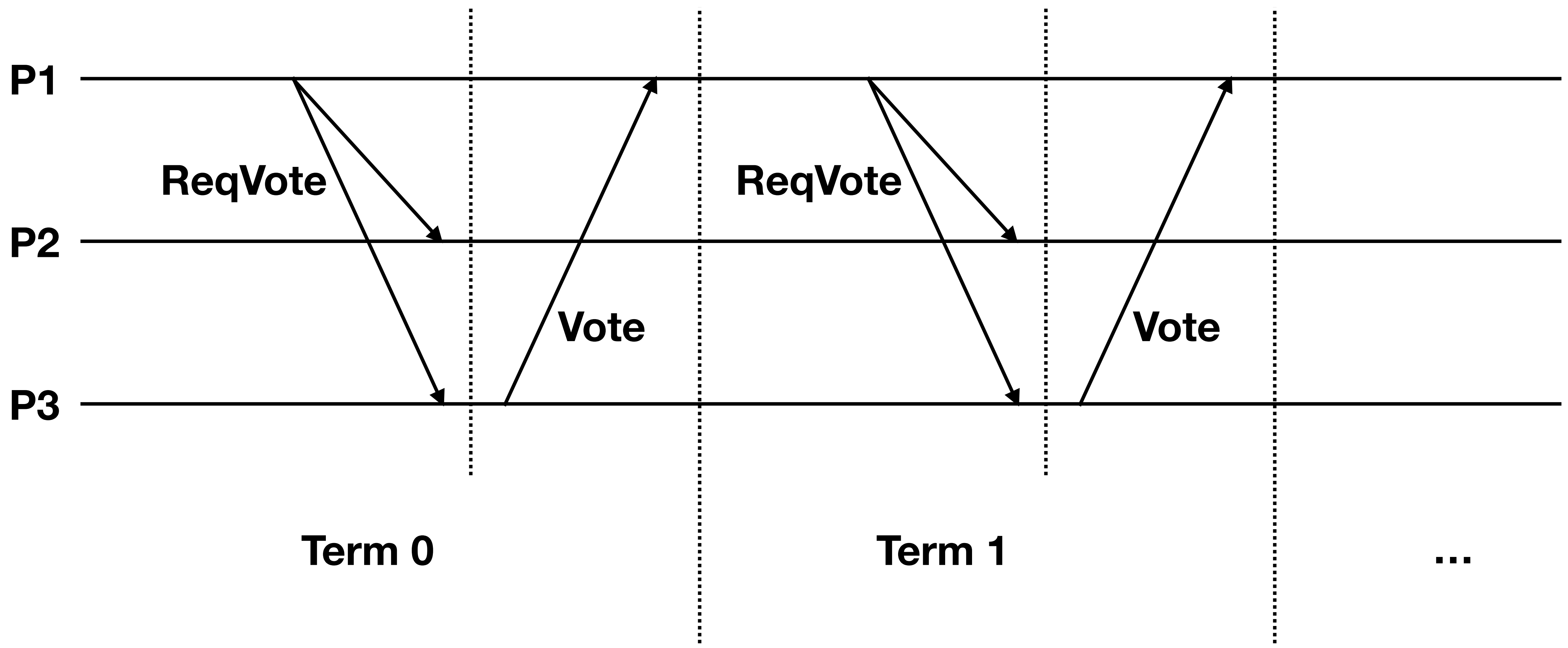


- Still can't beat random
- The problem:
  - Unbounded terms

Unbounded terms

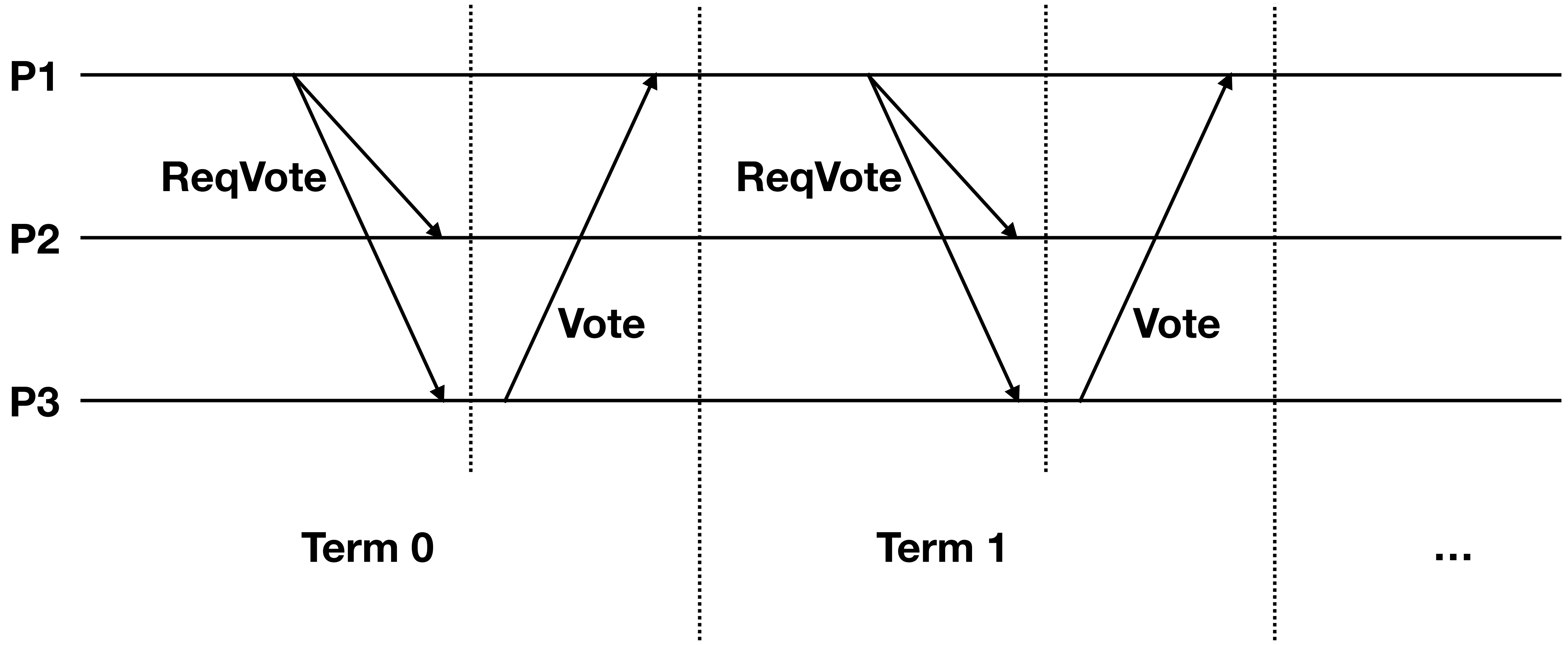


Unbounded terms



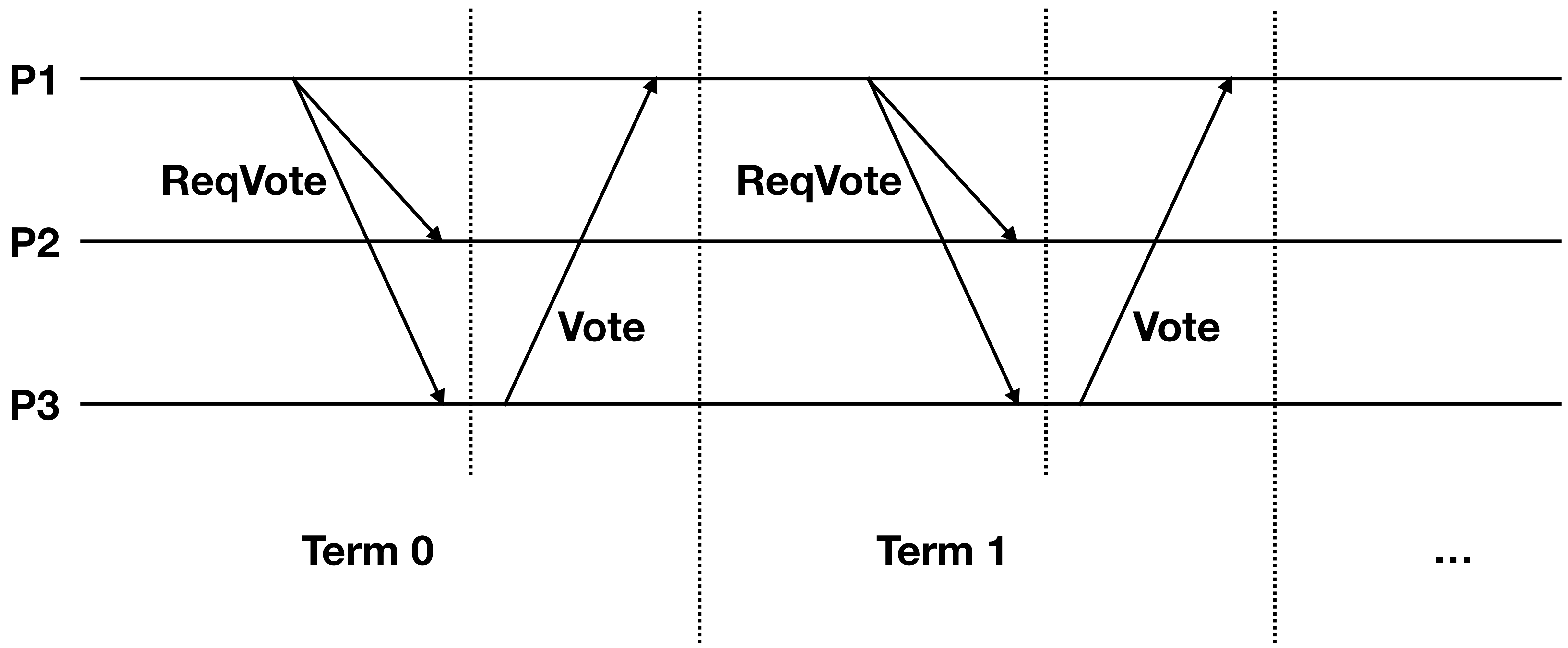
- Need a state abstraction

Unbounded terms



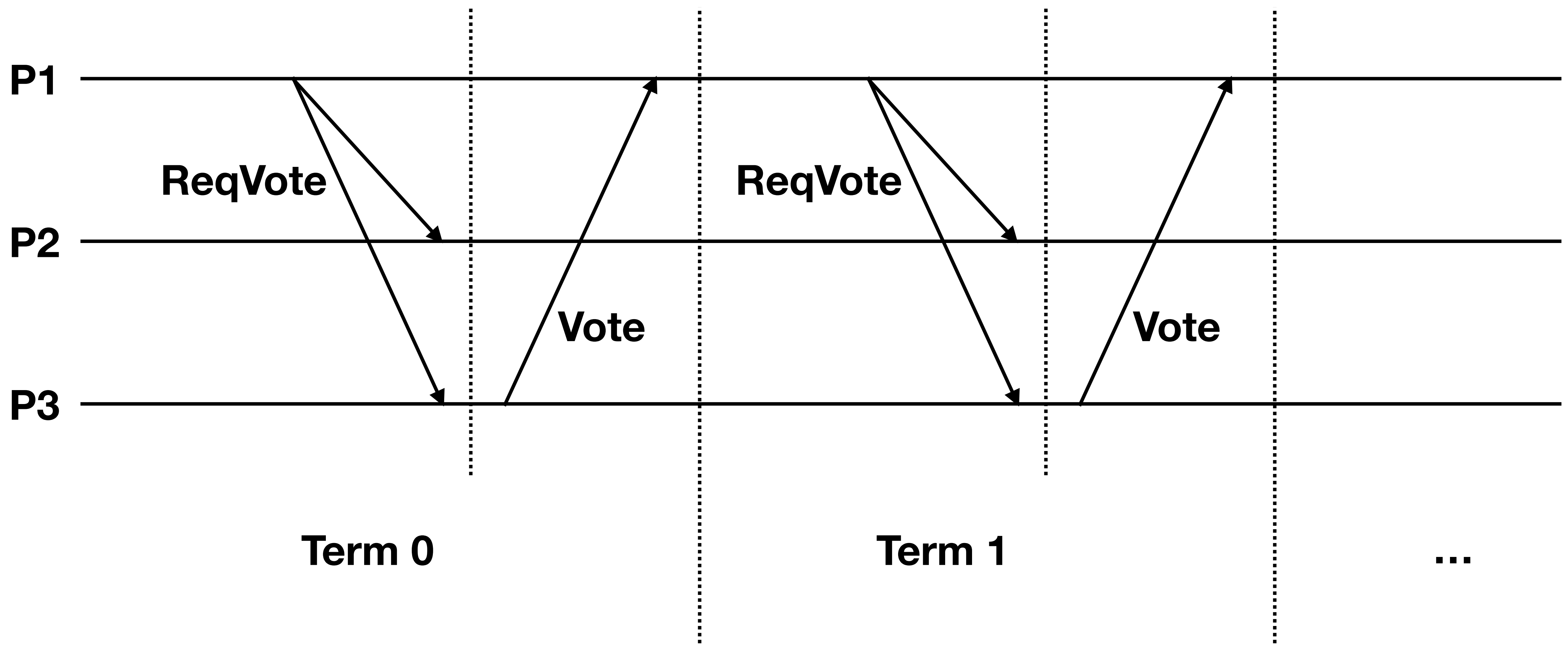
- Need a state abstraction
- Bound maximum term in the model

Unbounded terms



- Need a state abstraction
- Bound maximum term in the model
- Merge states that only differ in term numbers

Unbounded terms



- Need a state abstraction
- Bound maximum term in the model
- Merge states that only differ in term numbers
- Implemented inside TLC

Results

Benchmarks

Benchmarks

- Micro benchmark in Coyote

Benchmarks

- Micro benchmark in Coyote
- Etcd Raft - popular key value store

Benchmarks

- Micro benchmark in Coyote
- Etcd Raft - popular key value store
    - Golang, 1k LOC instrumentation

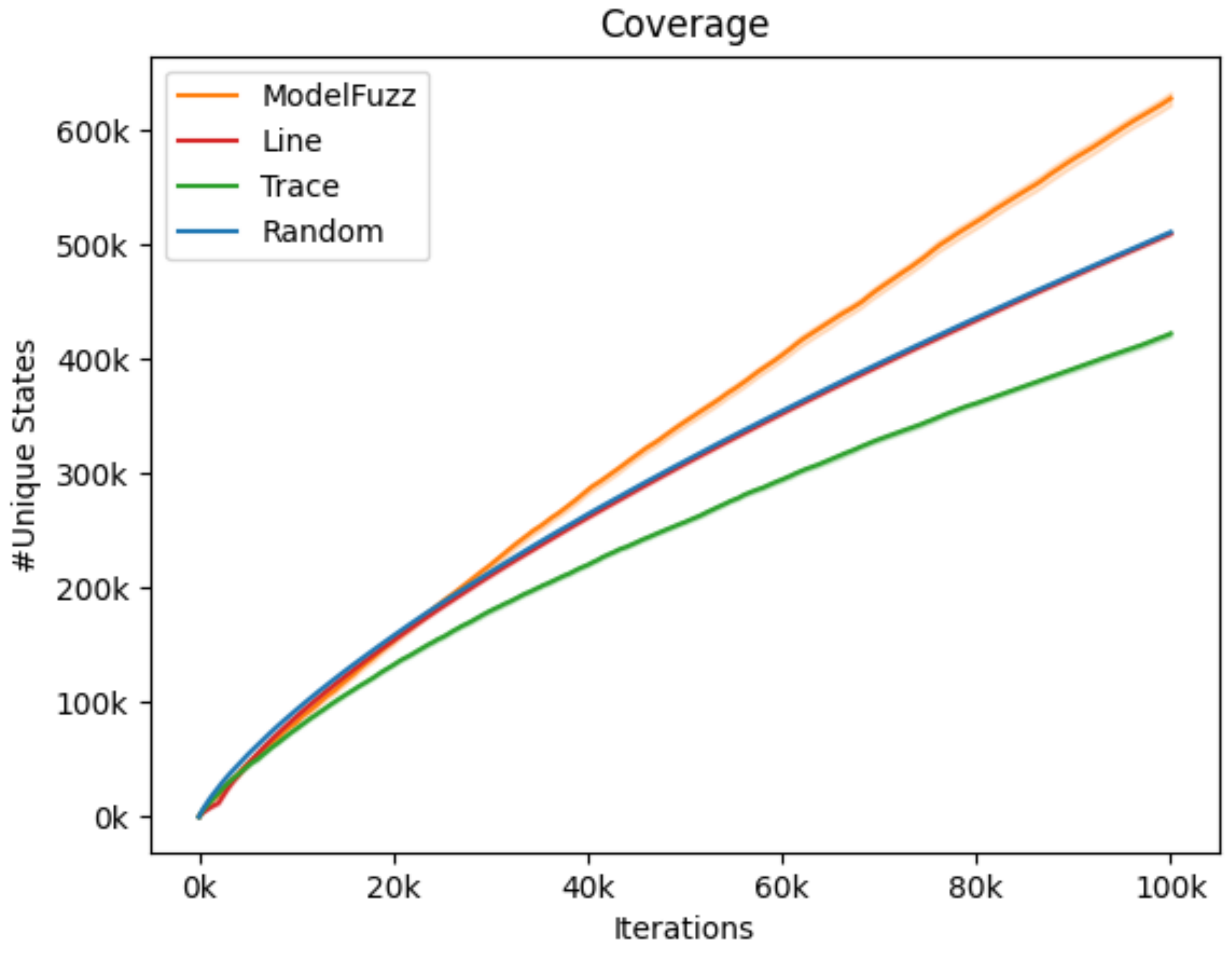
Benchmarks

- Micro benchmark in Coyote
- Etcd Raft - popular key value store
    - Golang, 1k LOC instrumentation
- Redis Raft - distributed in memory key value store

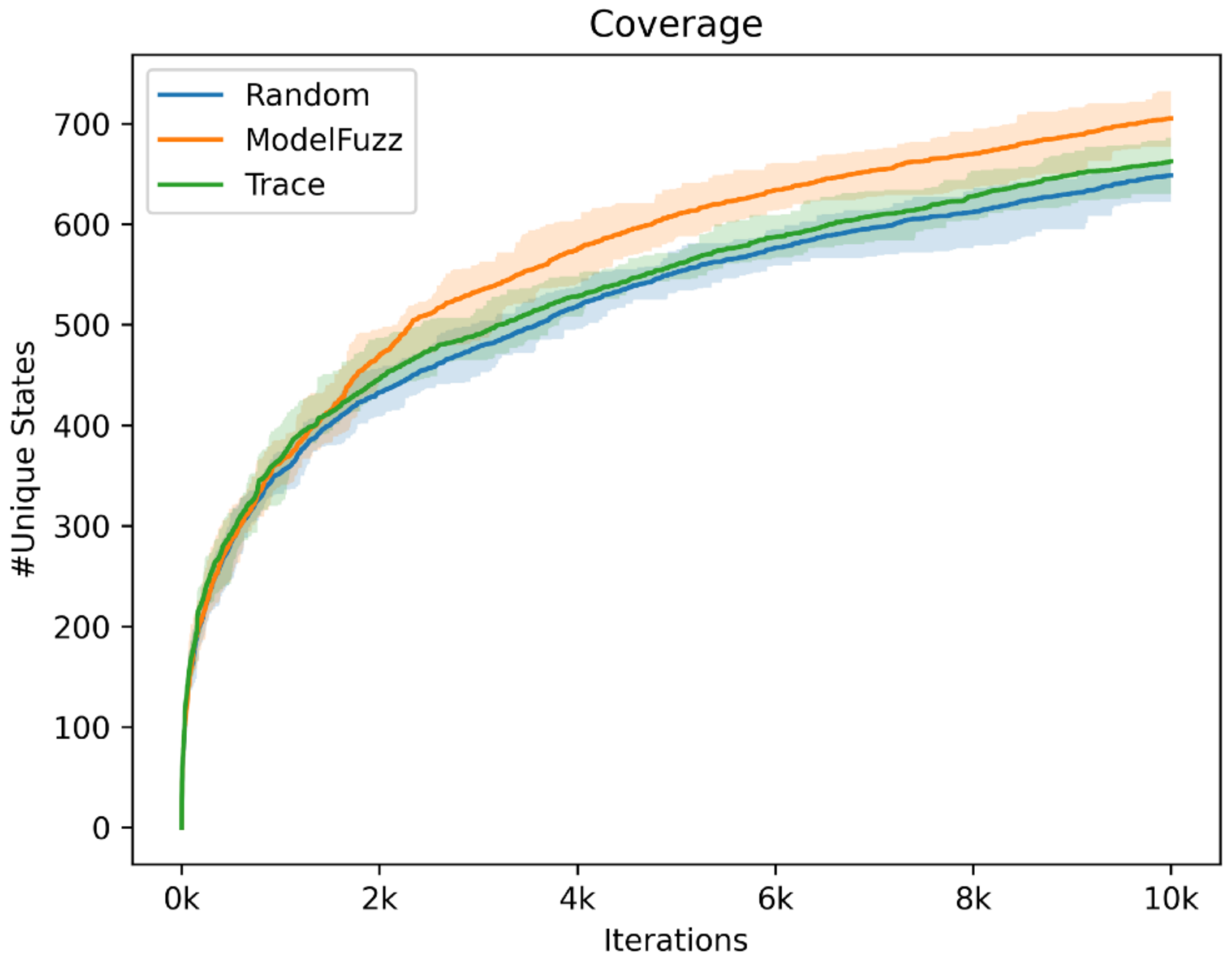
Benchmarks

- Micro benchmark in Coyote
- Etcd Raft - popular key value store
  - Golang, 1k LOC instrumentation
- Redis Raft - distributed in memory key value store
  - C, 1.5k LOC instrumentation

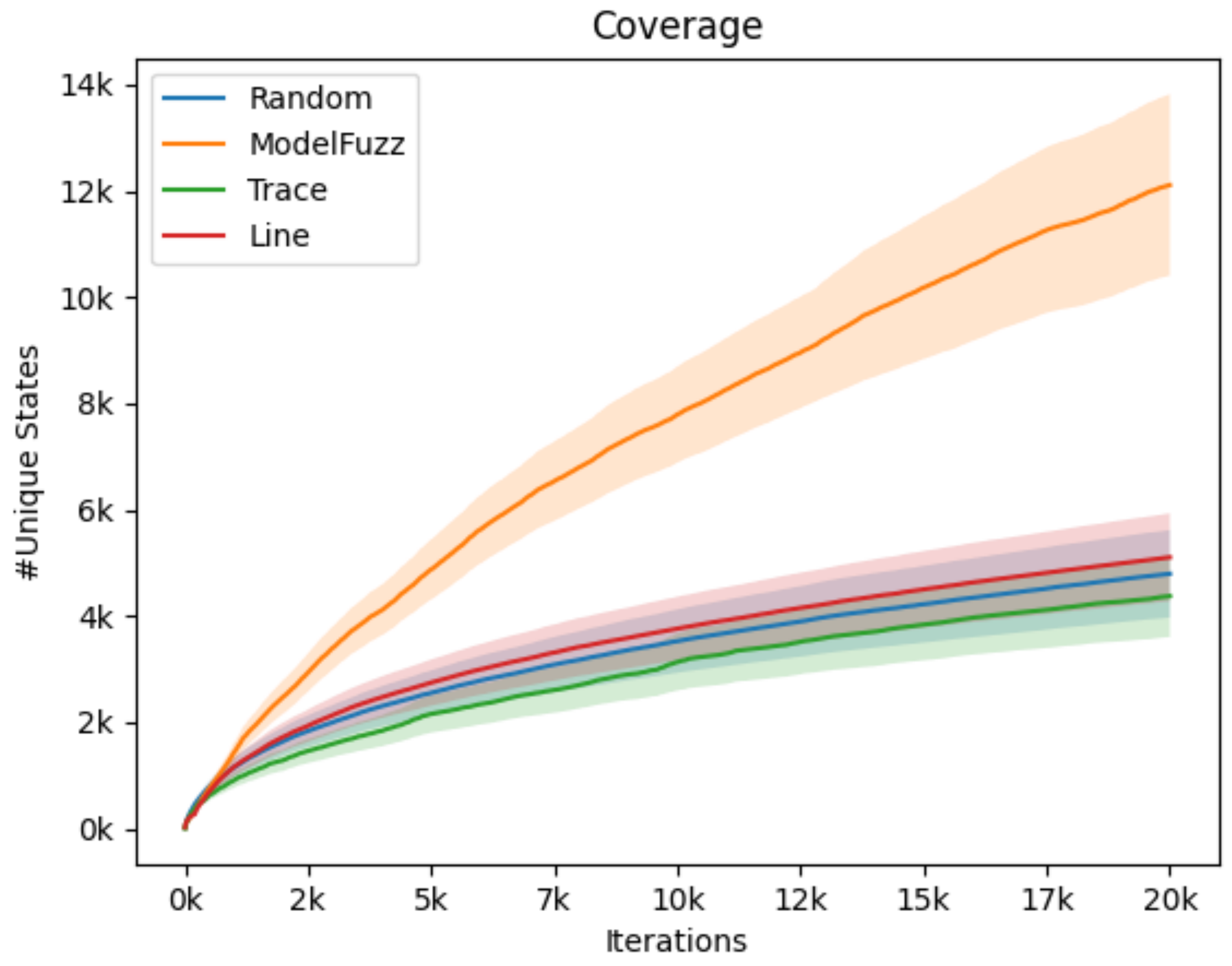
Coverage



Etcd



Micro benchmark



Redis

Comparing guidance

Comparing guidance

- Line coverage - saturates too quickly and hard to use the coverage information to observe new states

Comparing guidance

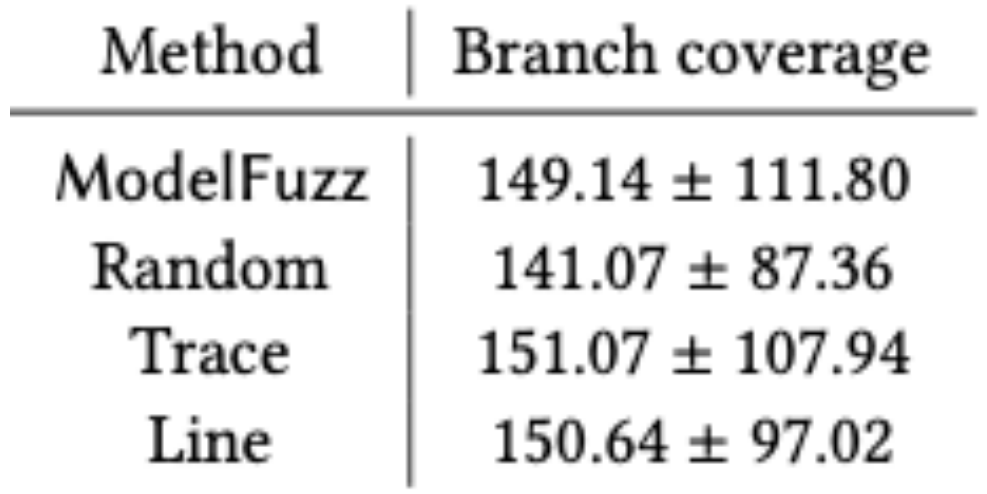
- Line coverage - saturates too quickly and hard to use the coverage information to observe new states
- Trace coverage - too fine grained, per message interleaving does not lead to new states

Comparing guidance

- Line coverage - saturates too quickly and hard to use the coverage information to observe new states

- Trace coverage - too fine grained, per message interleaving does not lead to new states

- Model coverage also provides good line coverage.



Method	Branch coverage
ModelFuzz	149.14 ± 111.80
Random	141.07 ± 87.36
Trace	151.07 ± 107.94
Line	150.64 ± 97.02

Bug finding

Bug finding

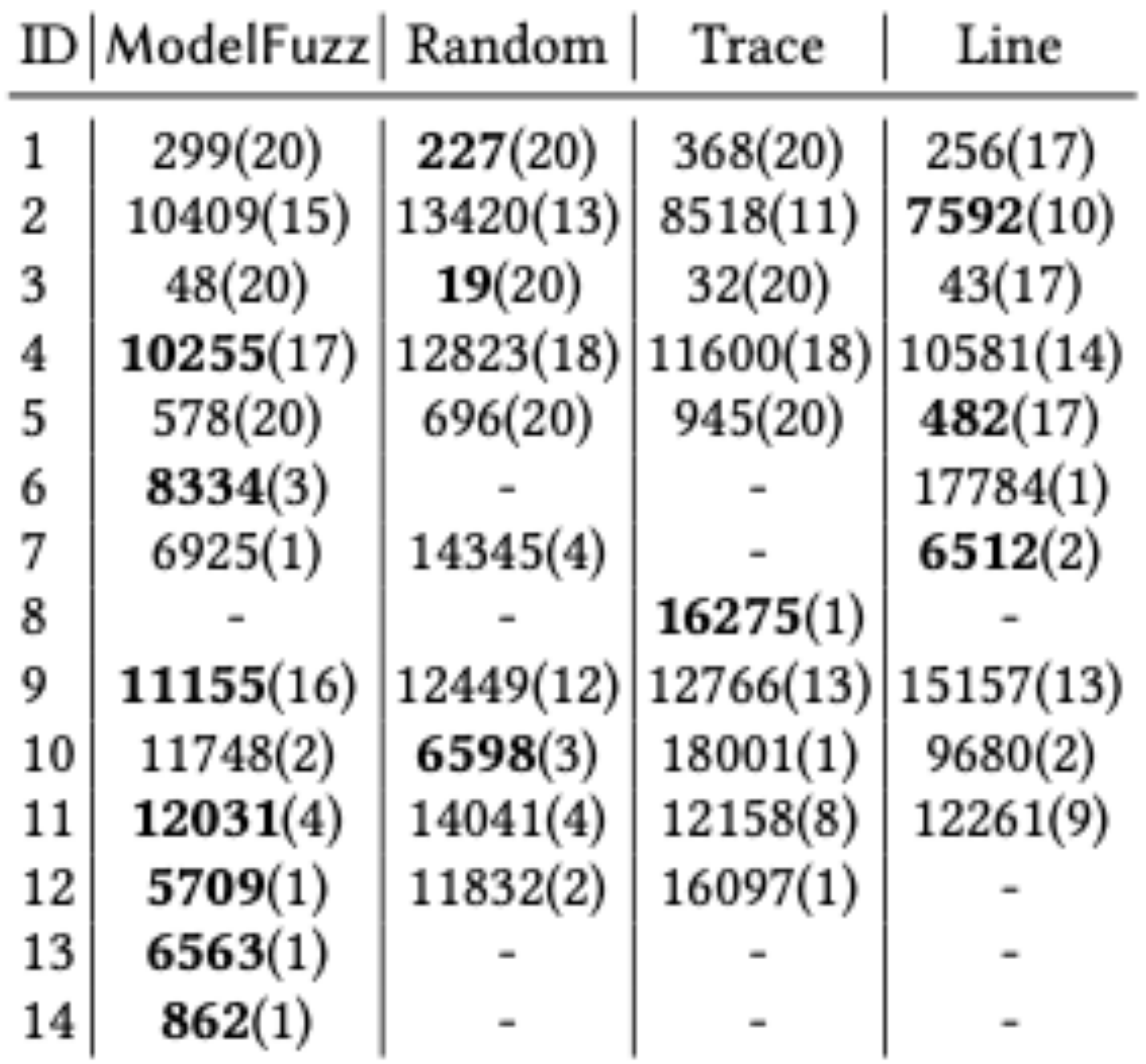
- **1 new bug** in Etcd

Bug finding

- **1 new bug** in Etcd
- 2 known bugs and **12 new bugs** in RedisRaft

Bug finding

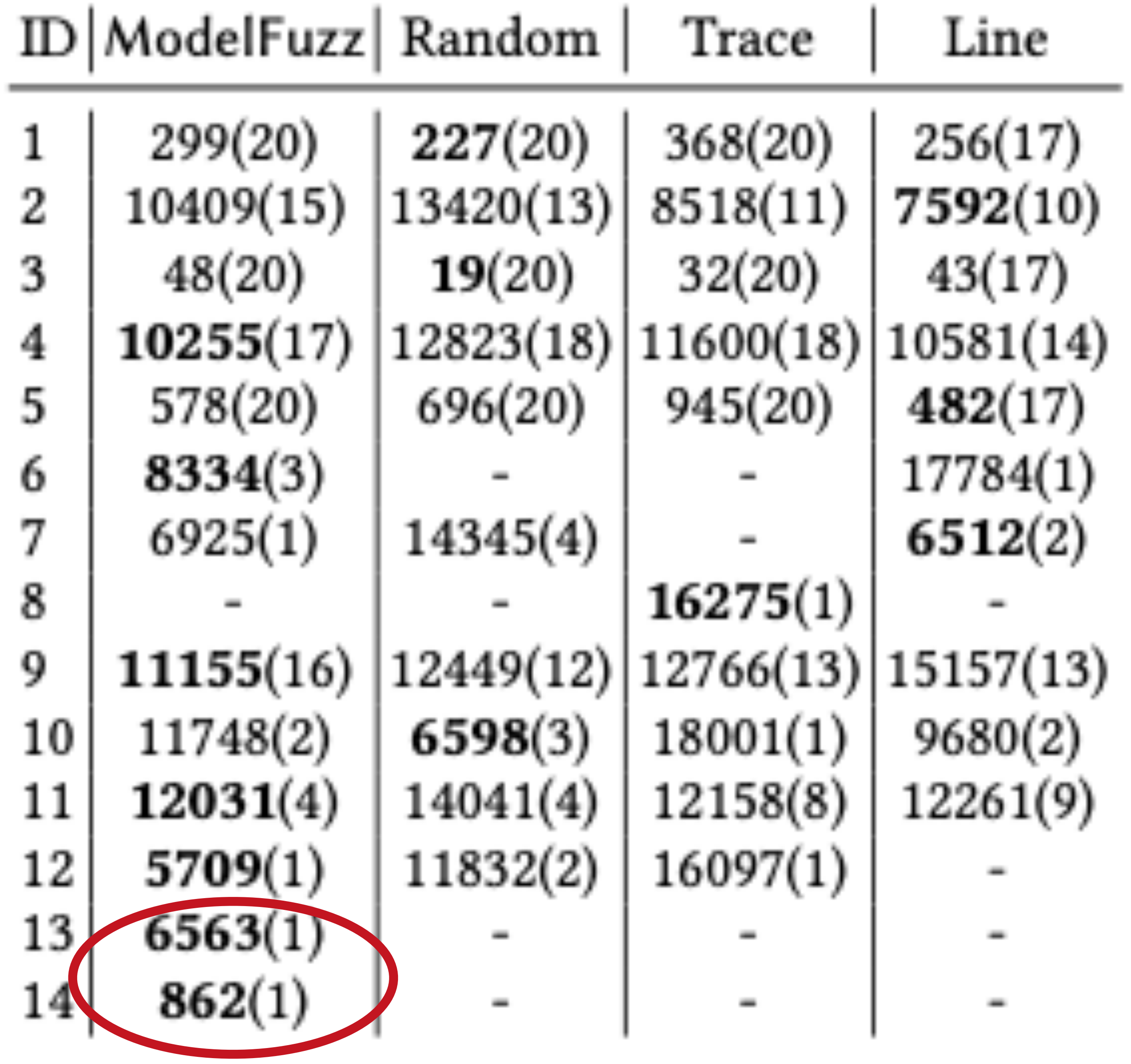
- **1 new bug** in Etcd
- 2 known bugs and **12 new bugs** in RedisRaft
- Bugs are found faster (statistically)



ID	ModelFuzz	Random	Trace	Line
1	299(20)	**227**(20)	368(20)	256(17)
2	10409(15)	13420(13)	8518(11)	**7592**(10)
3	48(20)	**19**(20)	32(20)	43(17)
4	**10255**(17)	12823(18)	11600(18)	10581(14)
5	578(20)	696(20)	945(20)	**482**(17)
6	**8334**(3)	-	-	17784(1)
7	6925(1)	14345(4)	-	**6512**(2)
8	-	-	**16275**(1)	-
9	**11155**(16)	12449(12)	12766(13)	15157(13)
10	11748(2)	**6598**(3)	18001(1)	9680(2)
11	**12031**(4)	14041(4)	12158(8)	12261(9)
12	**5709**(1)	11832(2)	16097(1)	-
13	**6563**(1)	-	-	-
14	**862**(1)	-	-	-

Bug finding

- **1 new bug** in Etcd
- 2 known bugs and **12 new bugs** in RedisRaft
- Bugs are found faster (statistically)
  - Especially when rare



ID	ModelFuzz	Random	Trace	Line
1	299(20)	**227**(20)	368(20)	256(17)
2	10409(15)	13420(13)	8518(11)	**7592**(10)
3	48(20)	**19**(20)	32(20)	43(17)
4	**10255**(17)	12823(18)	11600(18)	10581(14)
5	578(20)	696(20)	945(20)	**482**(17)
6	**8334**(3)	-	-	17784(1)
7	6925(1)	14345(4)	-	**6512**(2)
8	-	-	**16275**(1)	-
9	**11155**(16)	12449(12)	12766(13)	15157(13)
10	11748(2)	**6598**(3)	18001(1)	9680(2)
11	**12031**(4)	14041(4)	12158(8)	12261(9)
12	**5709**(1)	11832(2)	16097(1)	-
13	**6563**(1)	-	-	-
14	**862**(1)	-	-	-

Bird’s eye view

Existing work

Existing work

Model verification

- P, P# - actor runtime with model checking capabilities

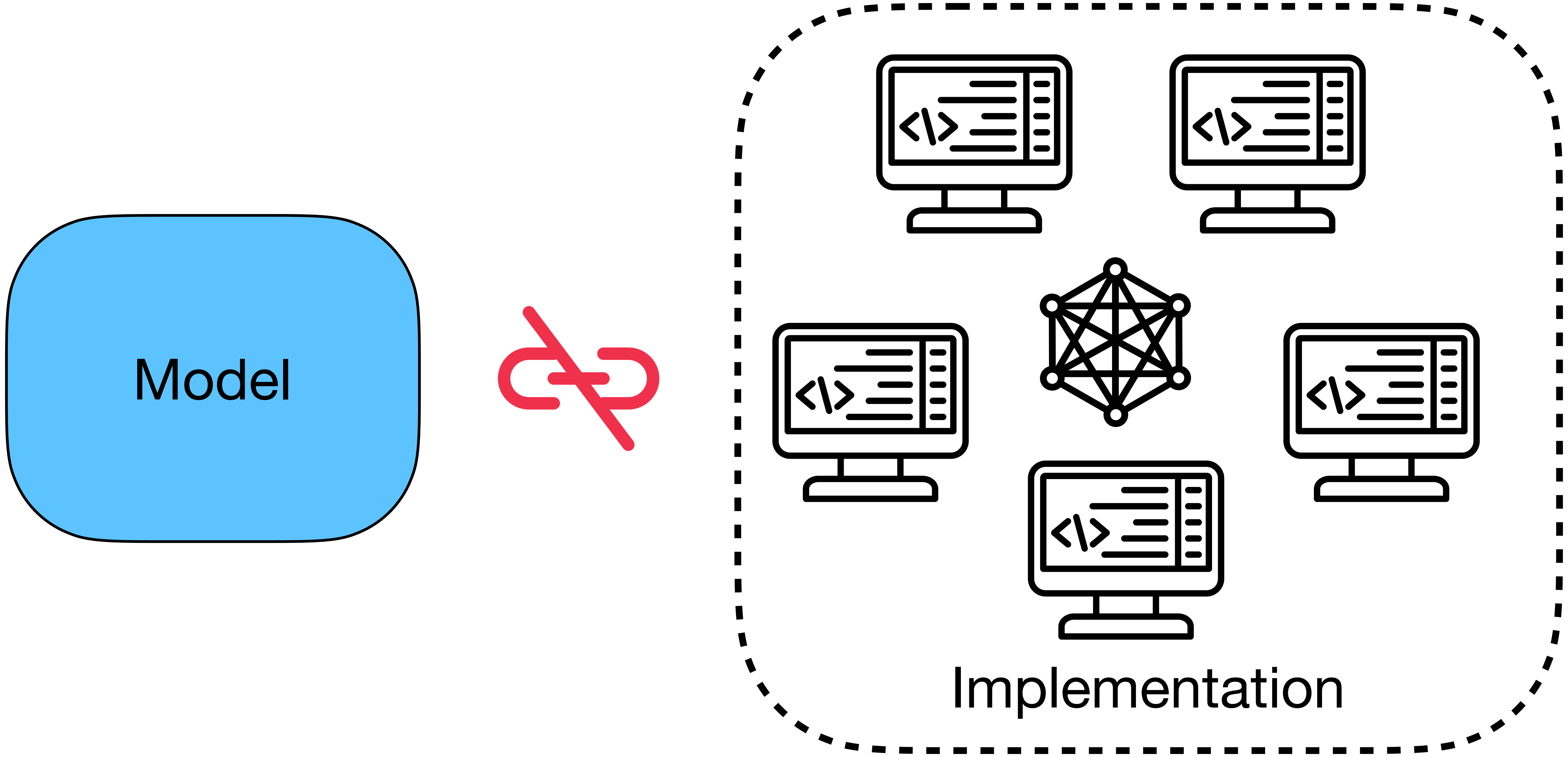
IVy

- Ivy - proof based technique to verify protocols

- Dafny - modelling language with a verification runtime

- TLA - modelling language with a model checker

Main problem



Limitations

Limitations

- Still a lot of effort - mapper, instrumentation, model

Limitations

- Still a lot of effort - mapper, instrumentation, model
- Too sensitive to abstractions

Limitations

- Still a lot of effort - mapper, instrumentation, model
- Too sensitive to abstractions
    - Can’t be too fine grained (too much information to generate tests)

Limitations

- Still a lot of effort - mapper, instrumentation, model
- Too sensitive to abstractions
  - Can’t be too fine grained (too much information to generate tests)
  - Can’t be too coarse grained (no information)

Future work

Future work

- Evaluating different mutation strategies

Future work

- Evaluating different mutation strategies
- Use more intelligent means of sampling

Future work

- Evaluating different mutation strategies
- Use more intelligent means of sampling
  - Machine learning?

Future work

- Evaluating different mutation strategies
- Use more intelligent means of sampling
  - Machine learning?
- Make mapping automatic

Future work

- Evaluating different mutation strategies
- Use more intelligent means of sampling
  - Machine learning?
- Make mapping automatic
- ~~Conformance checking~~ (SEFM ’24 - Cirstea et al)

Questions?