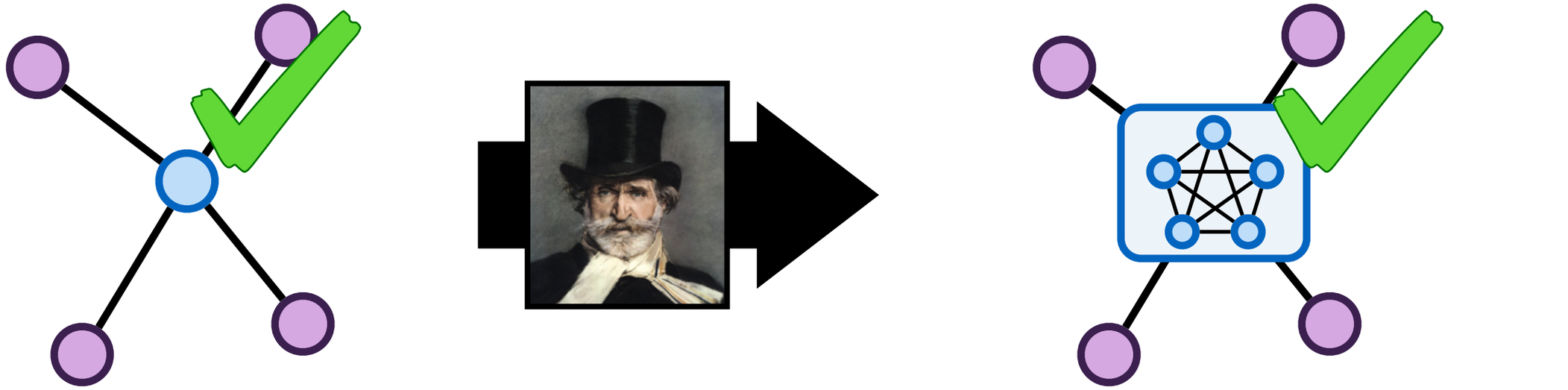
Verifying Distributed Systems



Zachary Tatlock

Verdi Lecture 3 at DeepSpec Summer School 2018

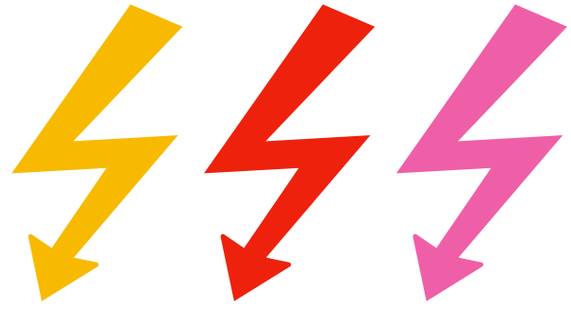
PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

PLSE

Toward verified distributed systems

Formalize *network semantics*

capture how faults can occur



Separate app / fault reasoning

develop and prove in simple fault model

apply generic verified fault handling

Toward verified distributed systems

The Verdi Framework

Verified Raft Consensus

TCB, Tools, Teaching

Enriching Models & Modularity

Toward verified distributed systems

The Verdi Framework

Verified Raft Consensus

TCB, Tools, Teaching

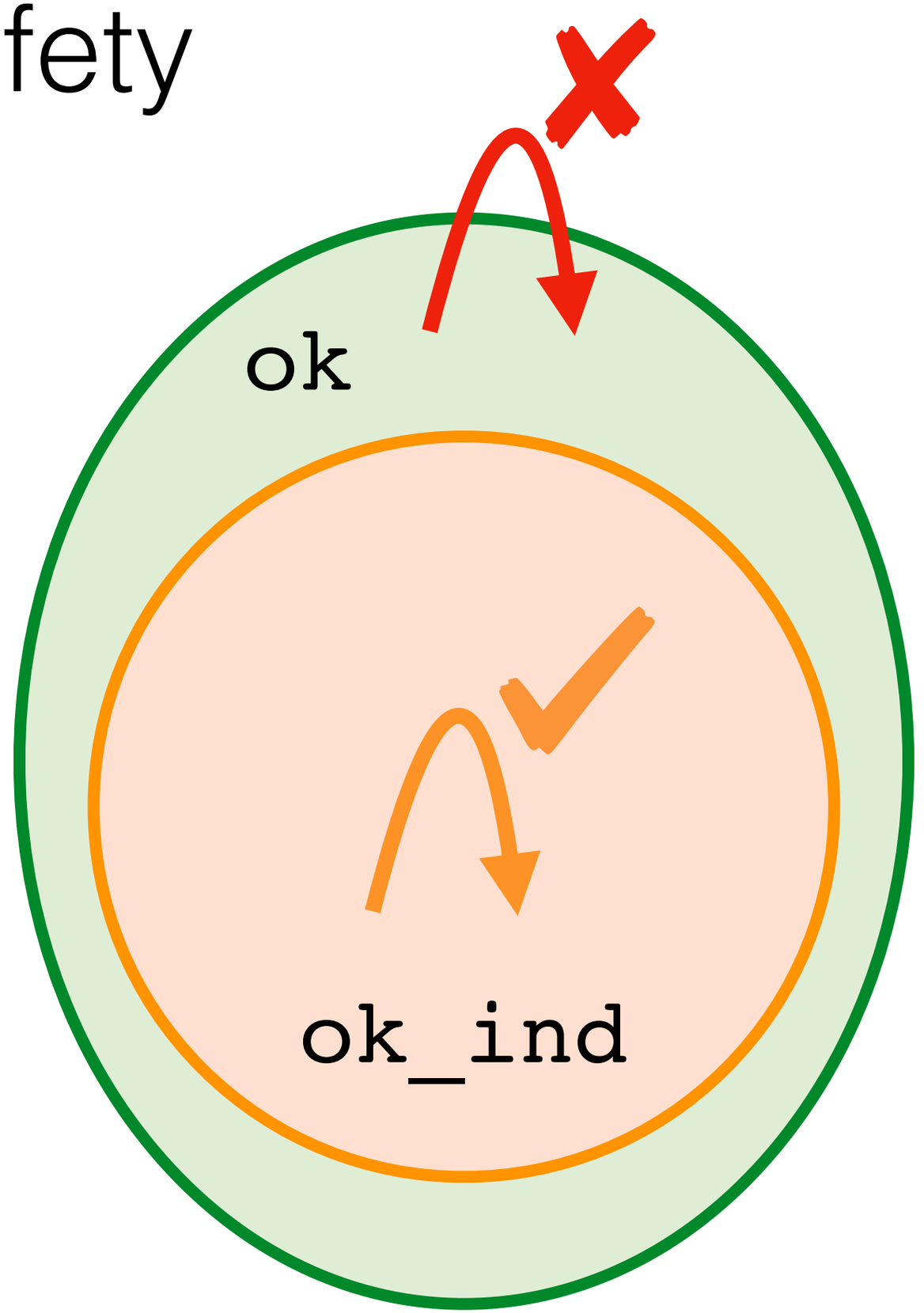
Enriching Models & Modularity

Verifying system safety

```
Def ok : state -> Prop
```

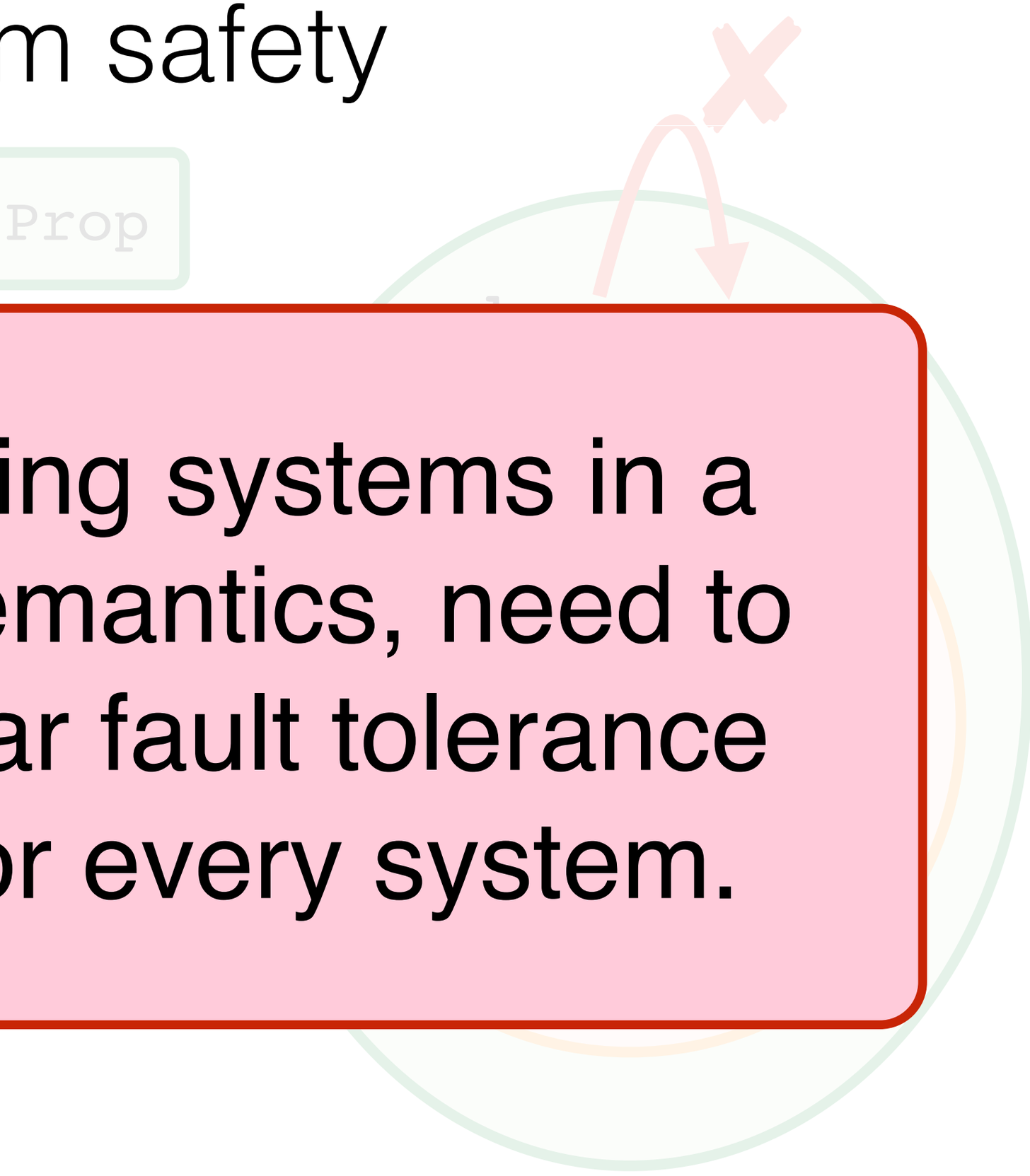
As usual, problem is specs not inductive.

Strengthen “`ok`” to inductive “`ok_ind`”.



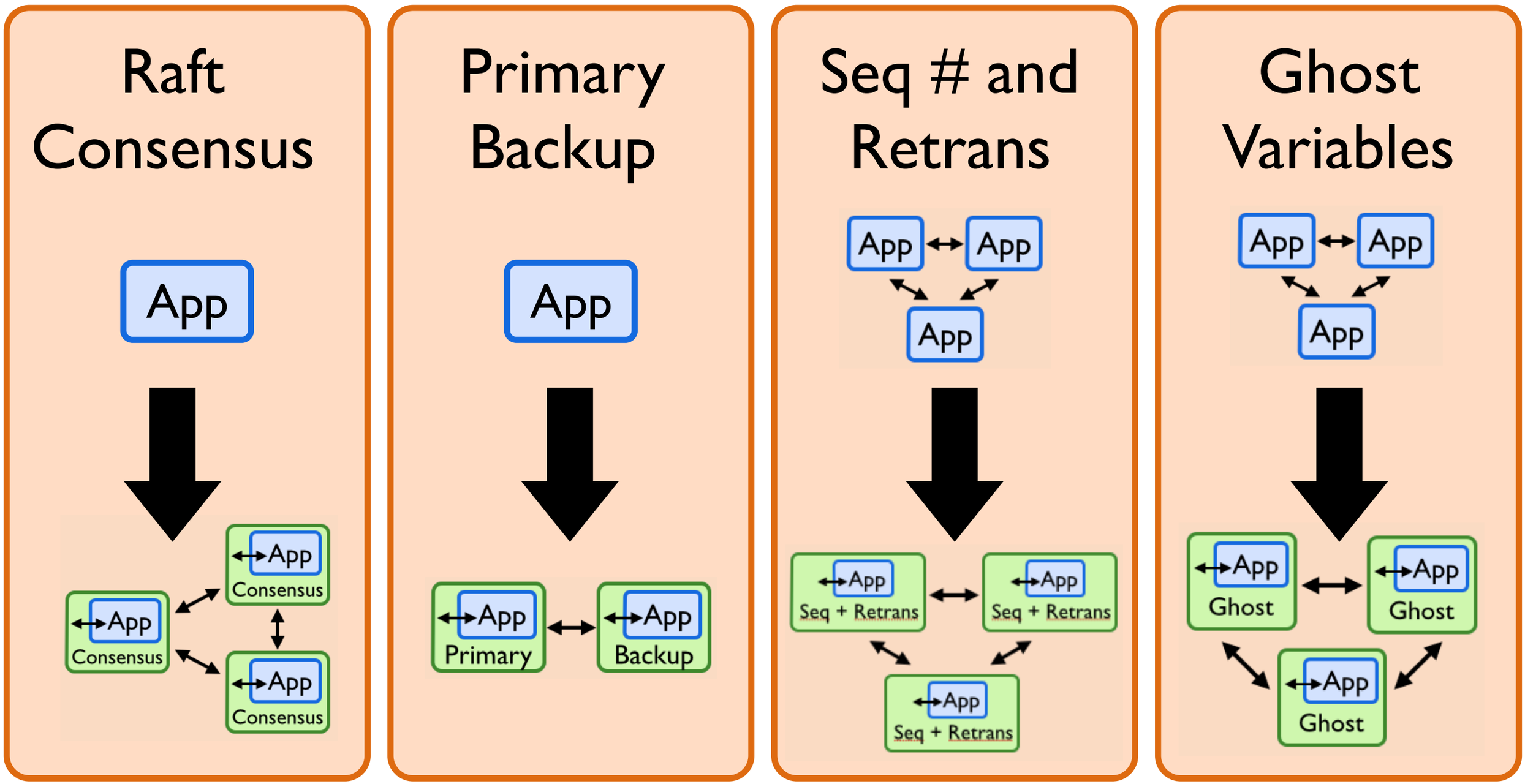
Verifying system safety

```
Def ok : state -> Prop
```



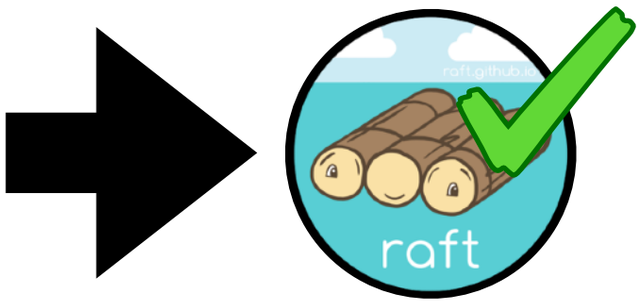
When verifying systems in a particular semantics, need to repeat similar fault tolerance reasoning for every system.

Verifying system *transformers*



Toward verified distributed systems

The Verdi Framework



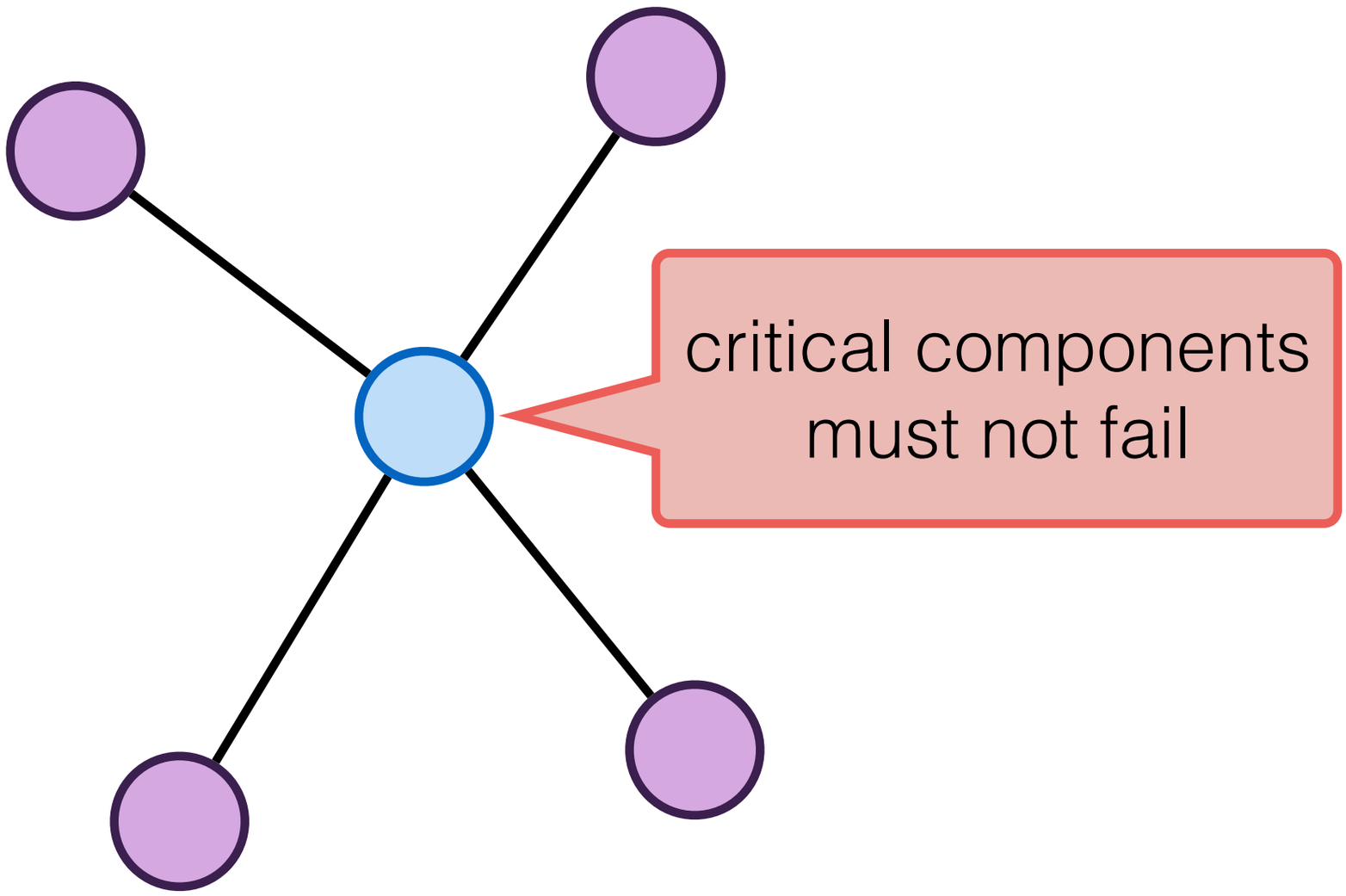
Verified Raft Consensus

TCB, Tools, Teaching

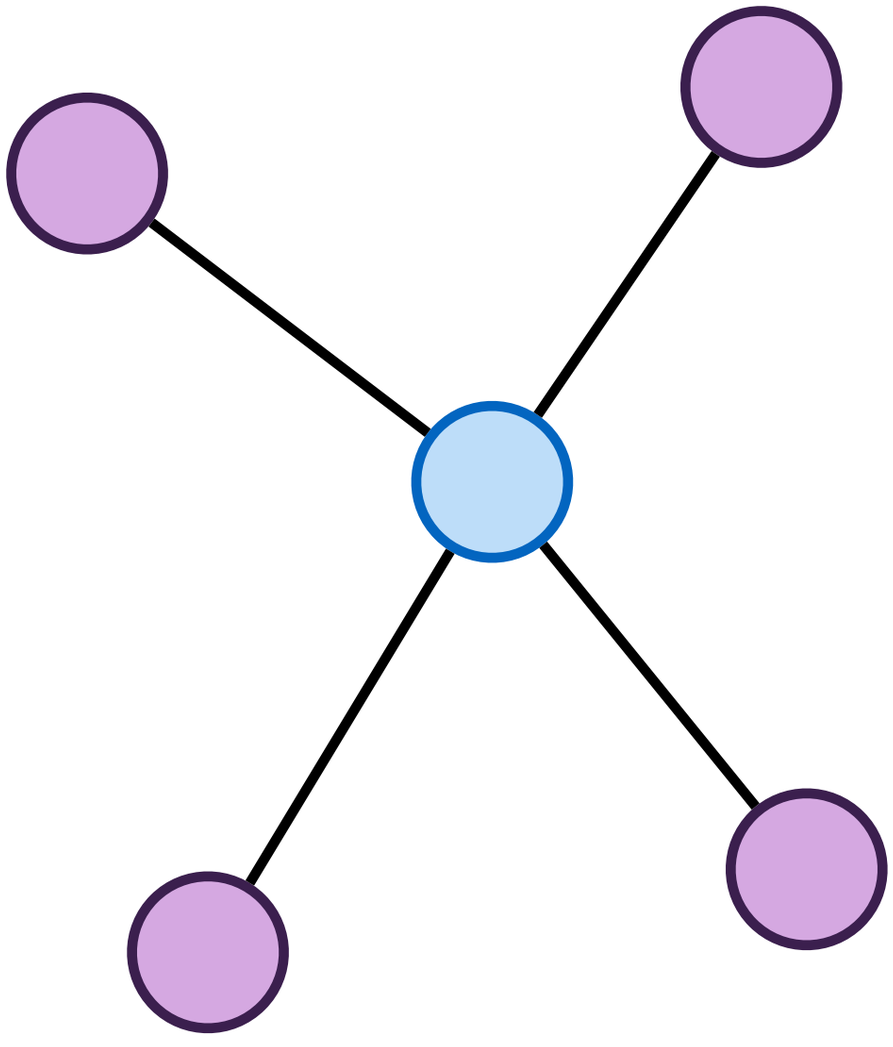
Enriching Models & Modularity

Example: verifying Raft in Verdi

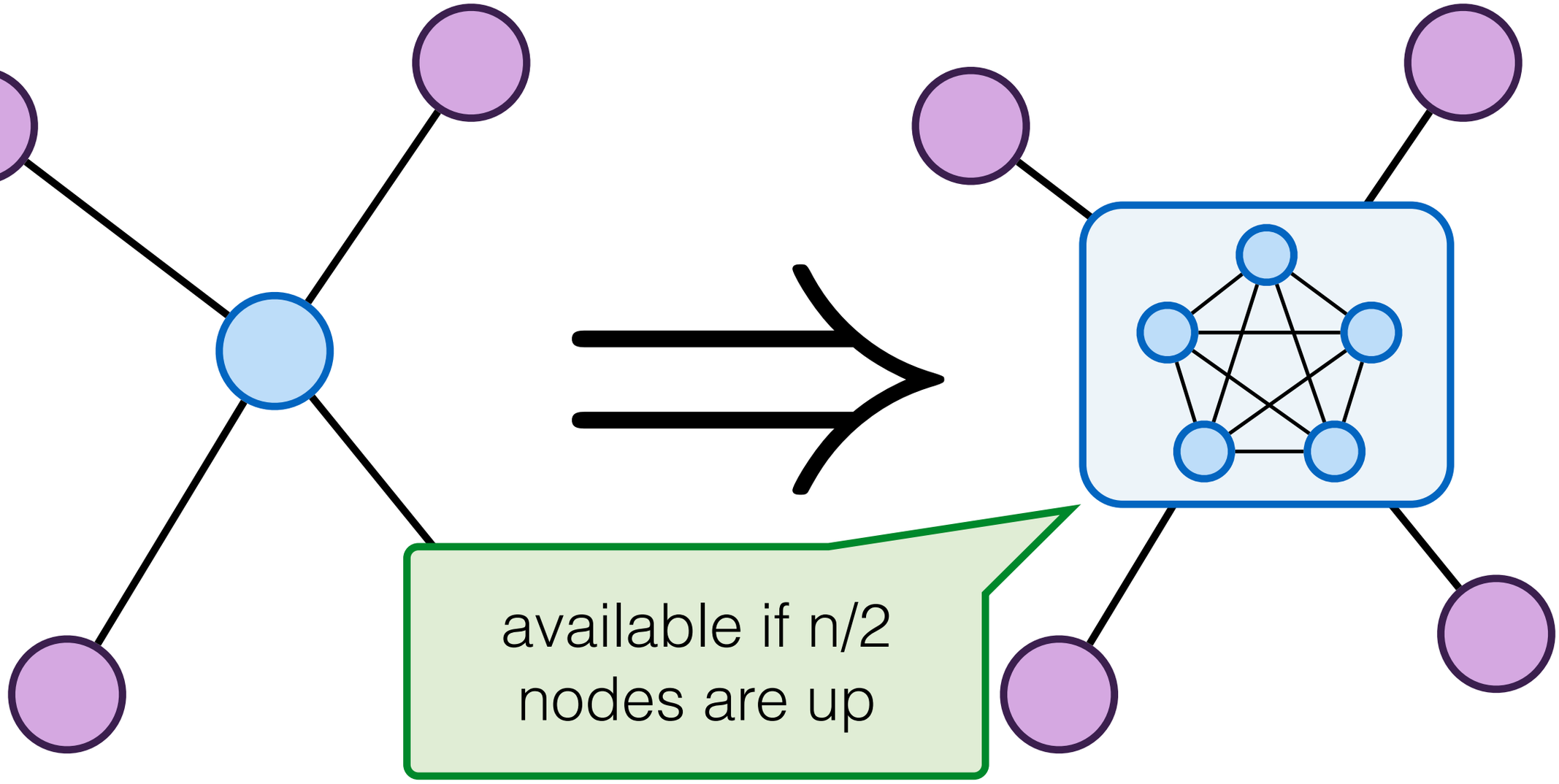
Example: verifying Raft in Verdi



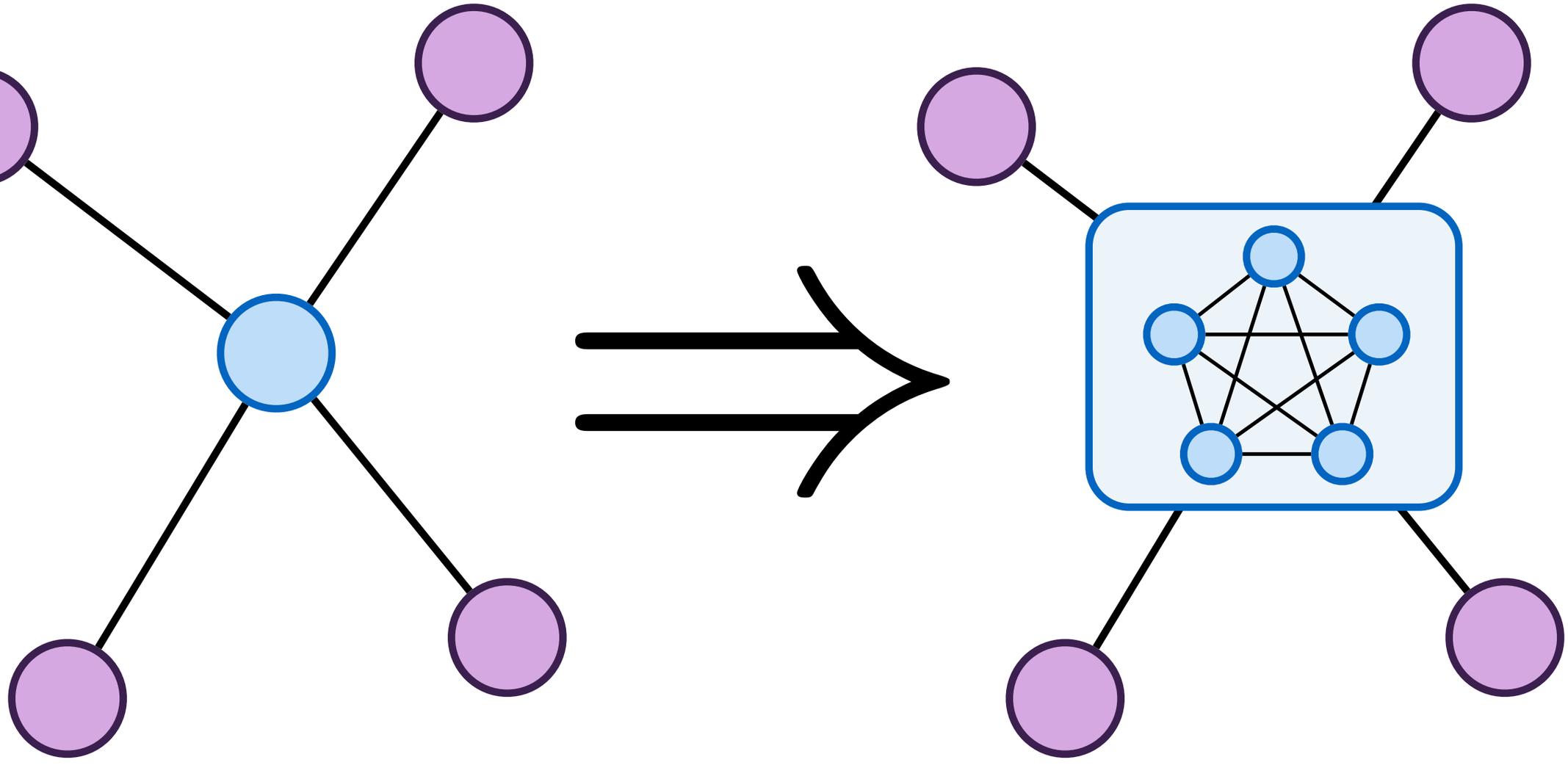
Replication for fault tolerance



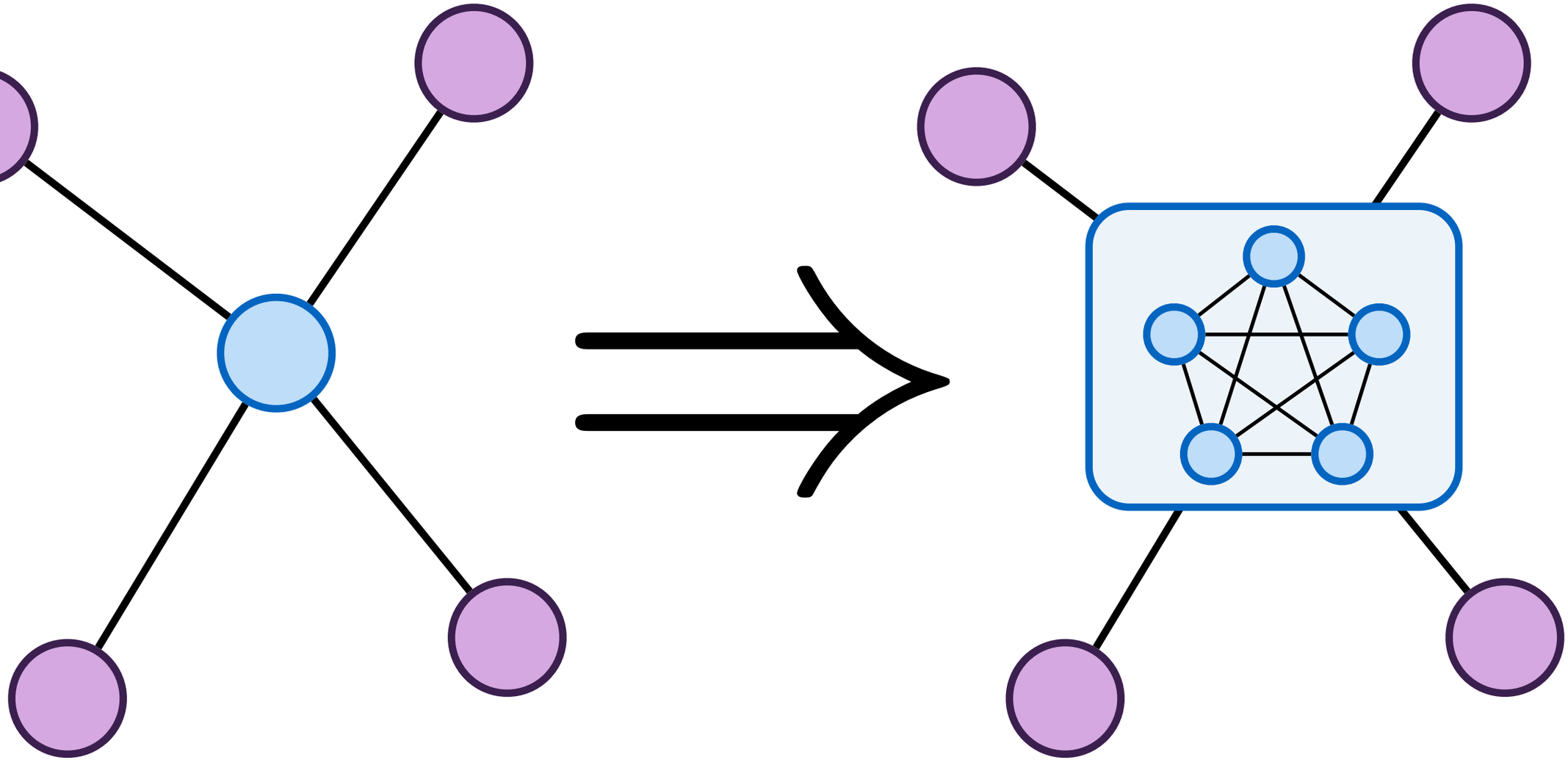
Replication for fault tolerance



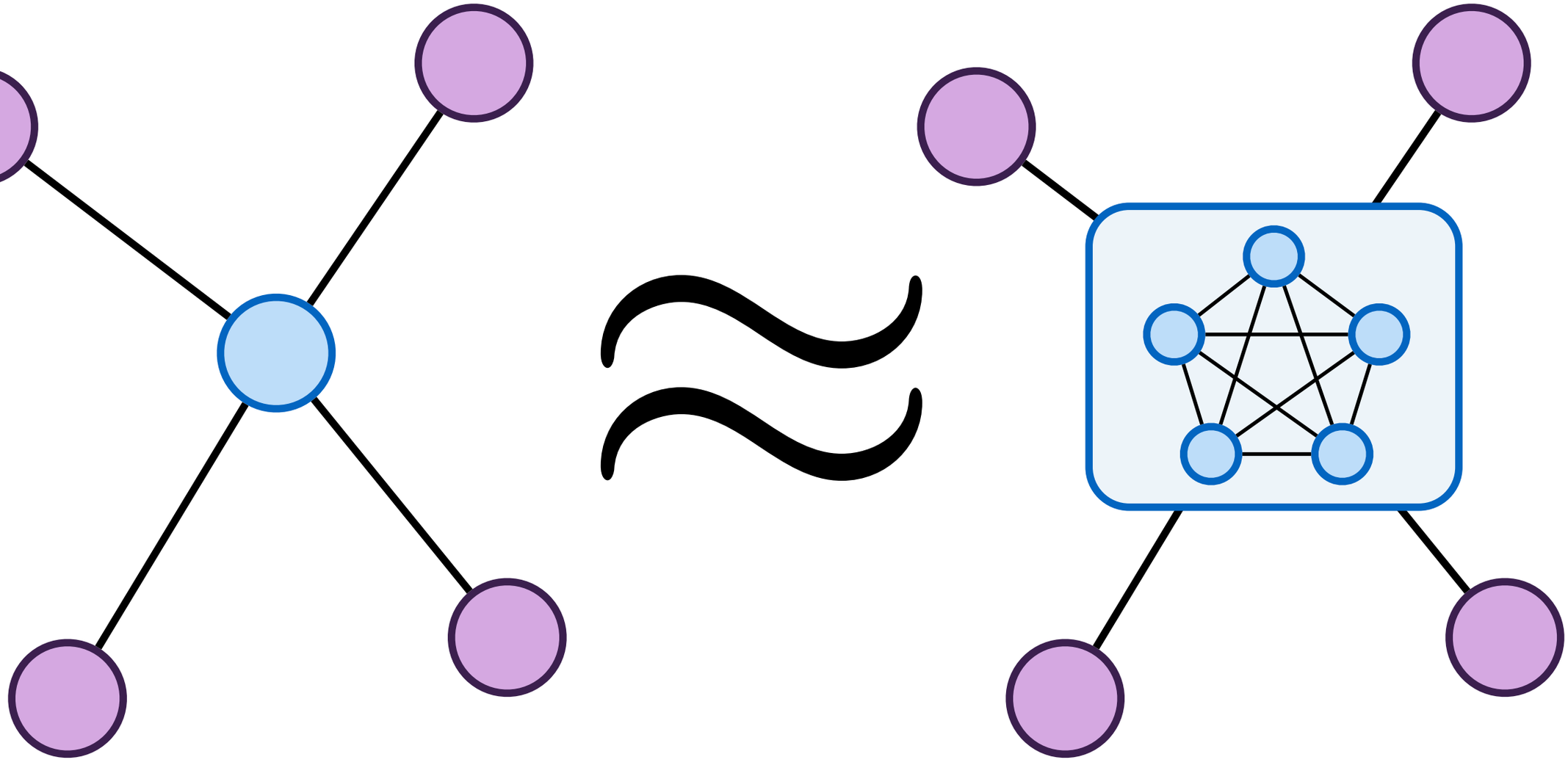
Replication for fault tolerance



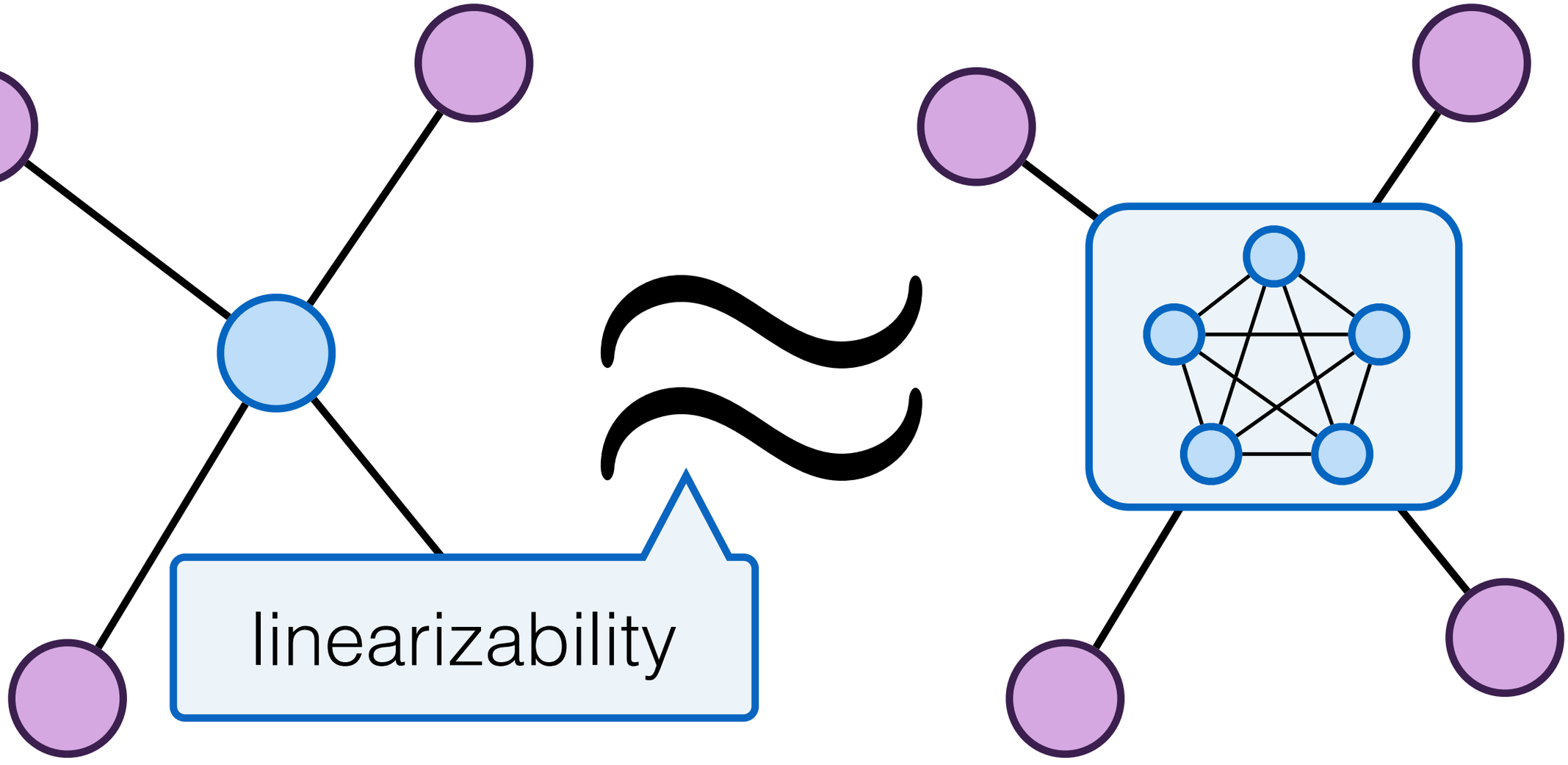
Replication correctness



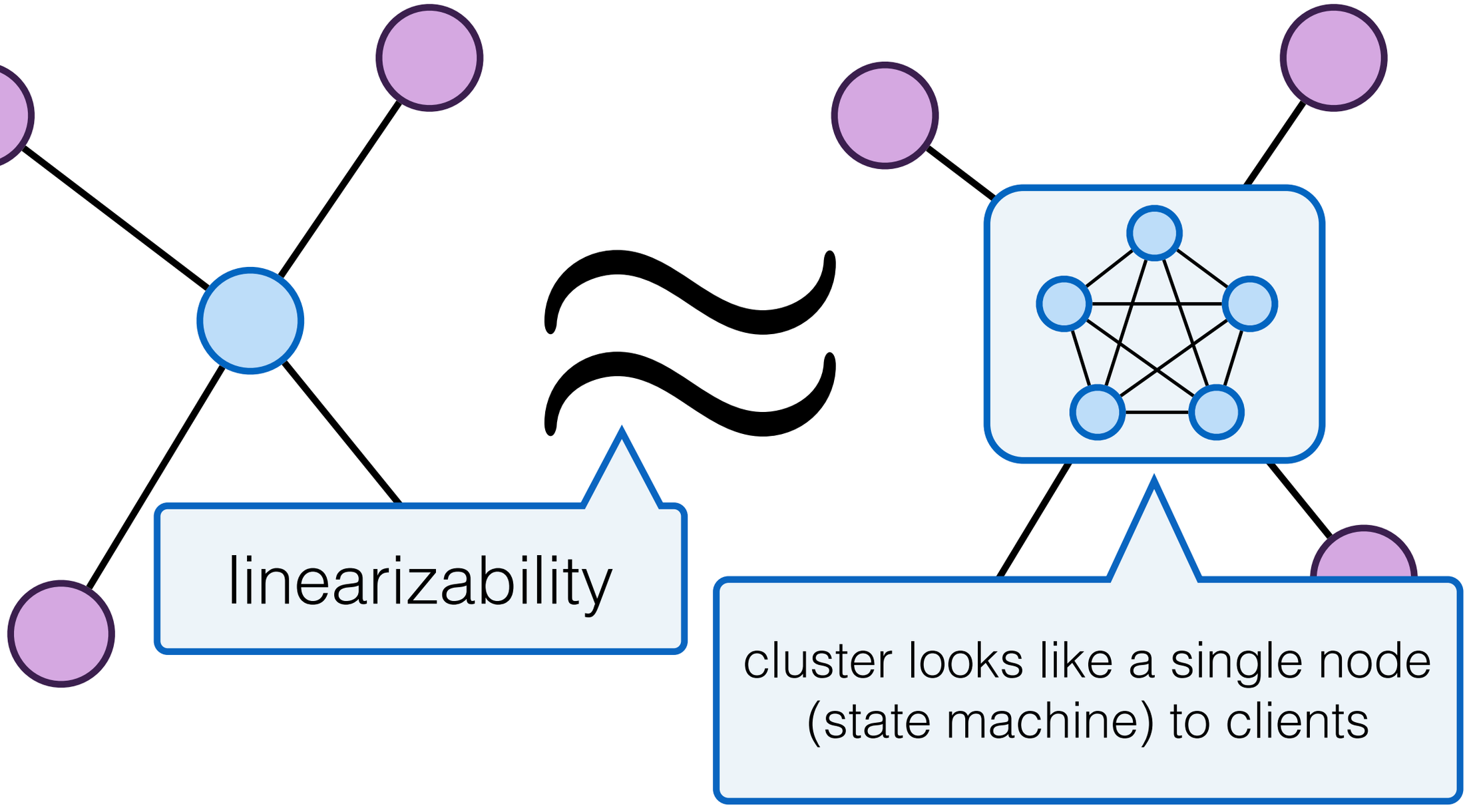
Replication correctness



Replication correctness



Replication correctness



Defining Raft

```
Def raft(sm: state machine, ...) :=

  ...
```

Defining Raft

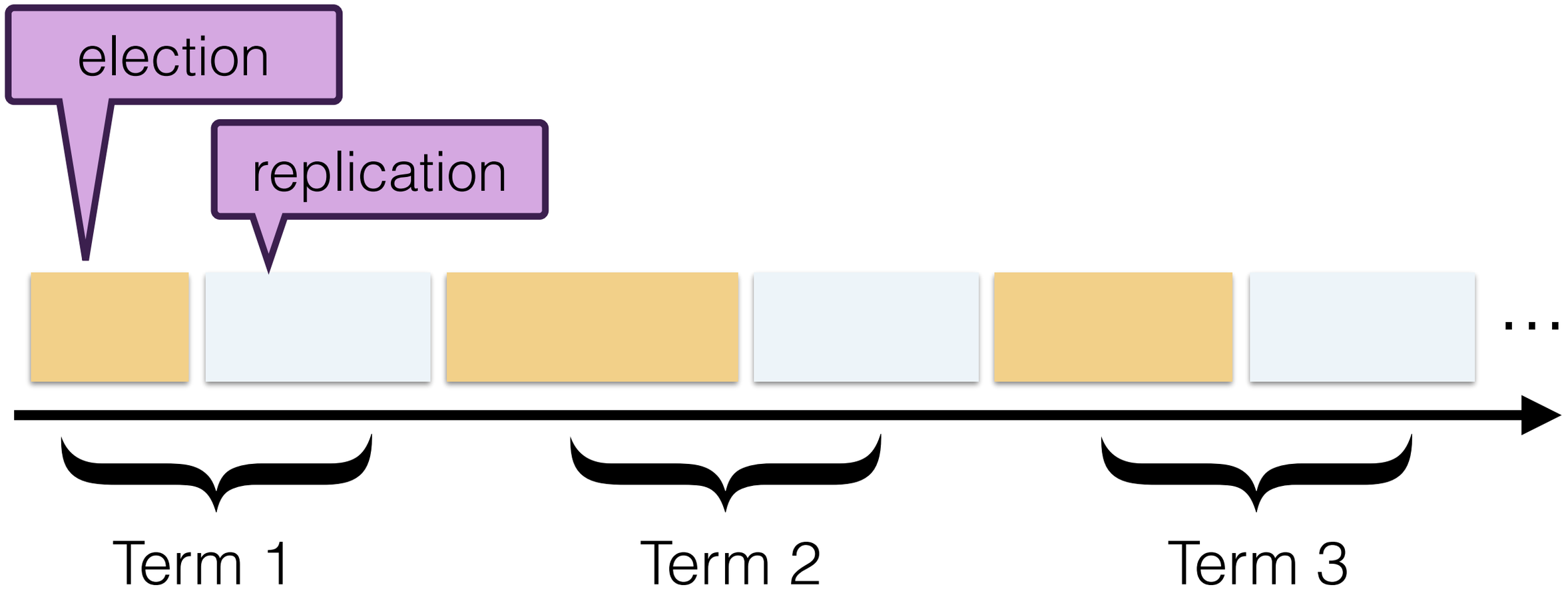
```
Def raft(sm: state machine, ...) :=

  // types for state and I/O
  cmsg := sm.cmsg


  . . .
```

Defining Raft

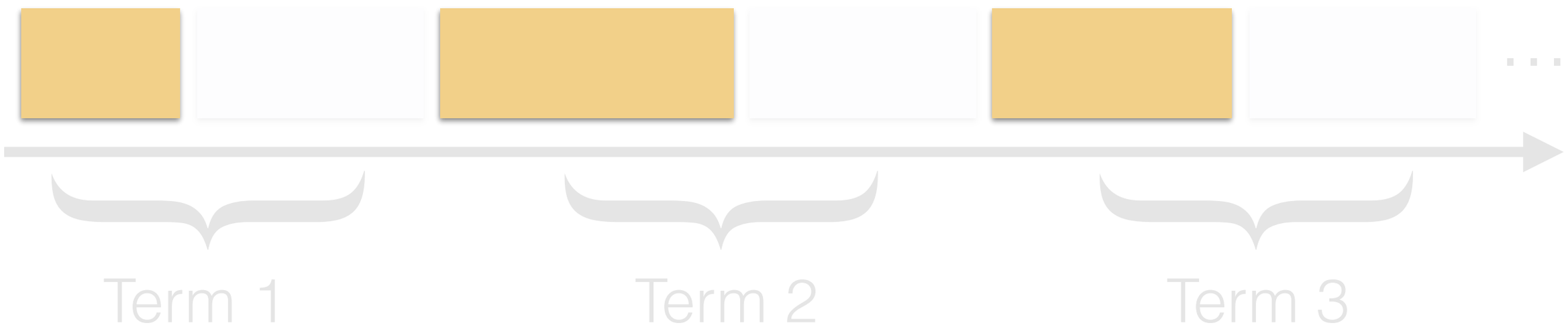
```
Def raft(sm: state machine, ...) :=

  // types for state and I/O
  cmsg := sm.cmsg
  msg  := (* ??? *)
  data := (* ??? *)


  // event handlers
  Def onMsg    := (* ??? *)
  Def onTmOut  := (* ??? *)
  Def onClient := (* ??? *)
```

Raft: election and replication terms

Raft: election and replication terms

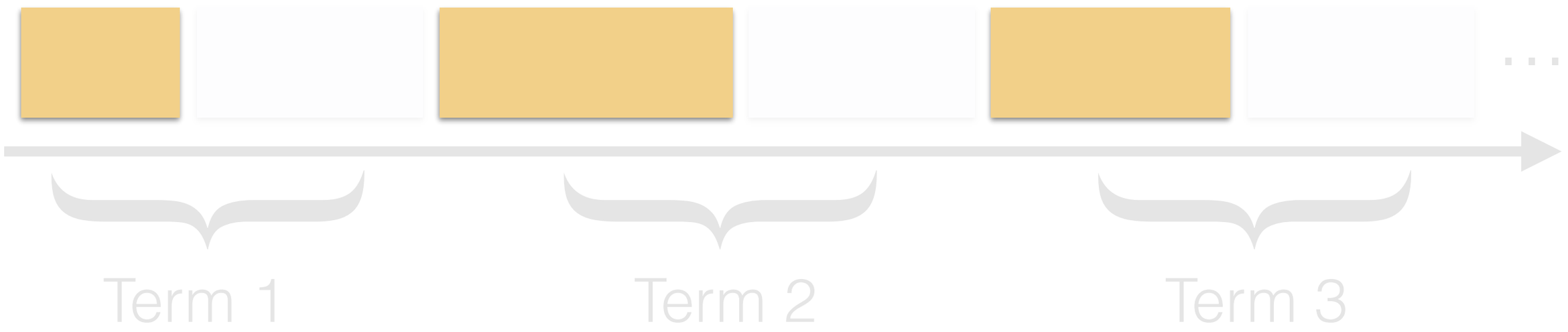


Raft: leader election

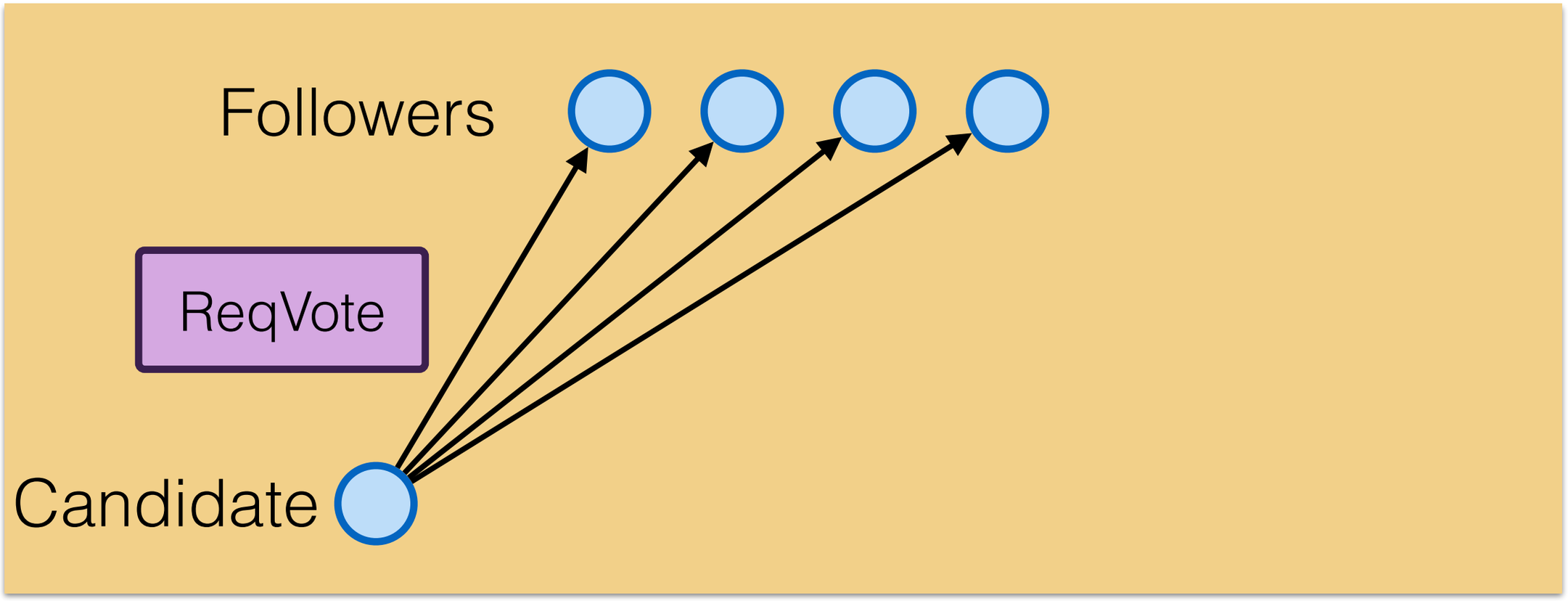


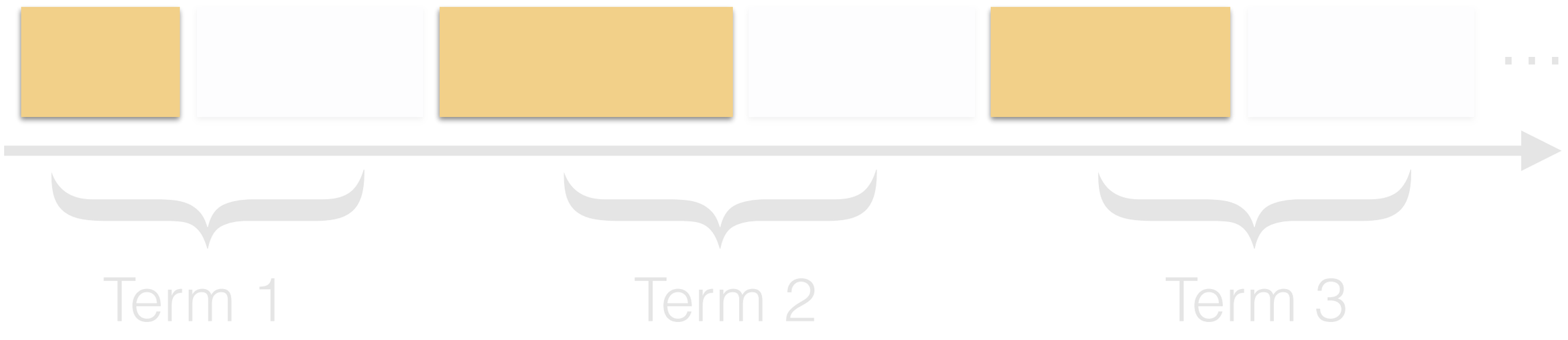
Raft: leader election

Candidate

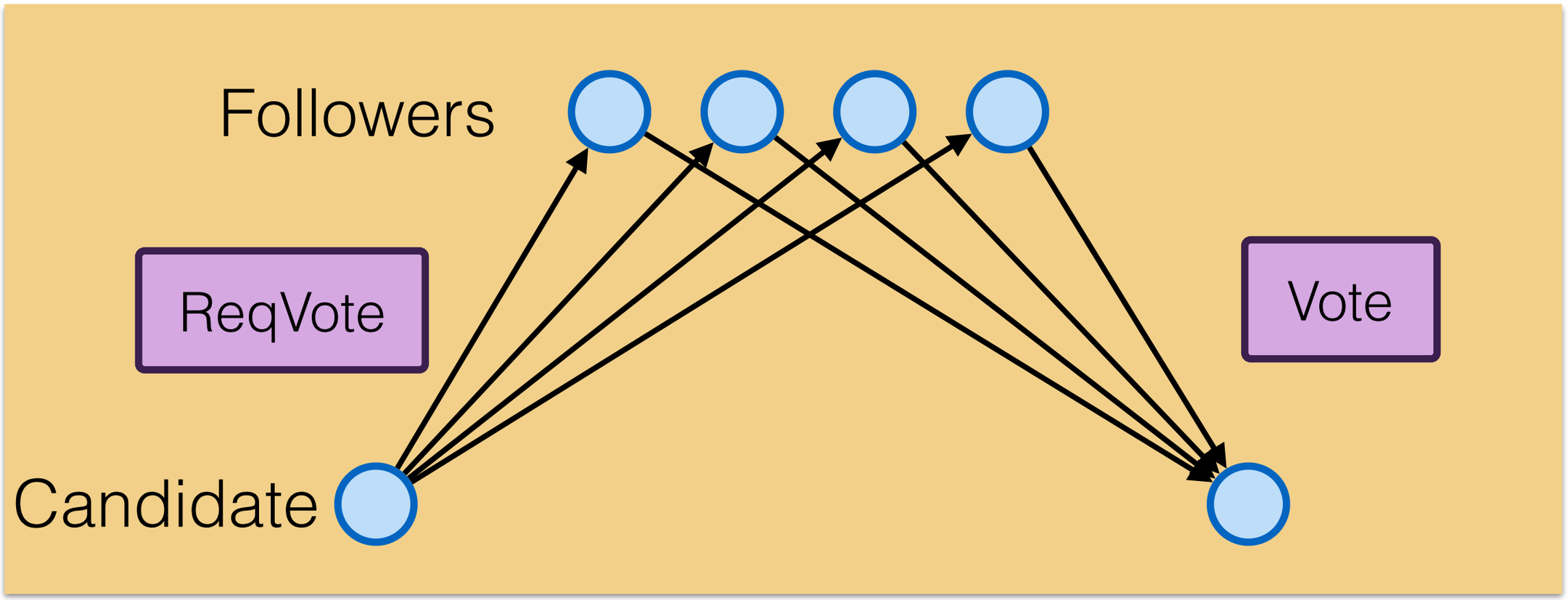


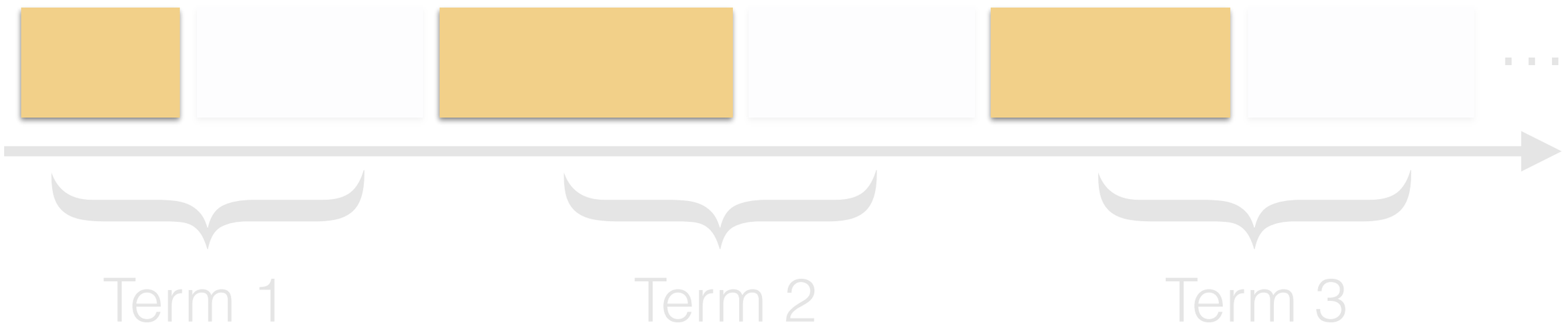
Raft: leader election



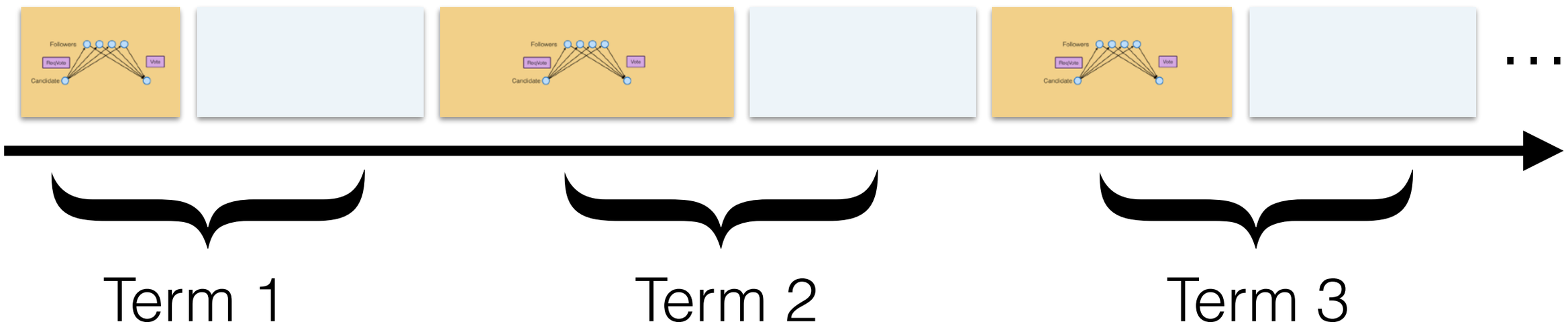


Raft: leader election

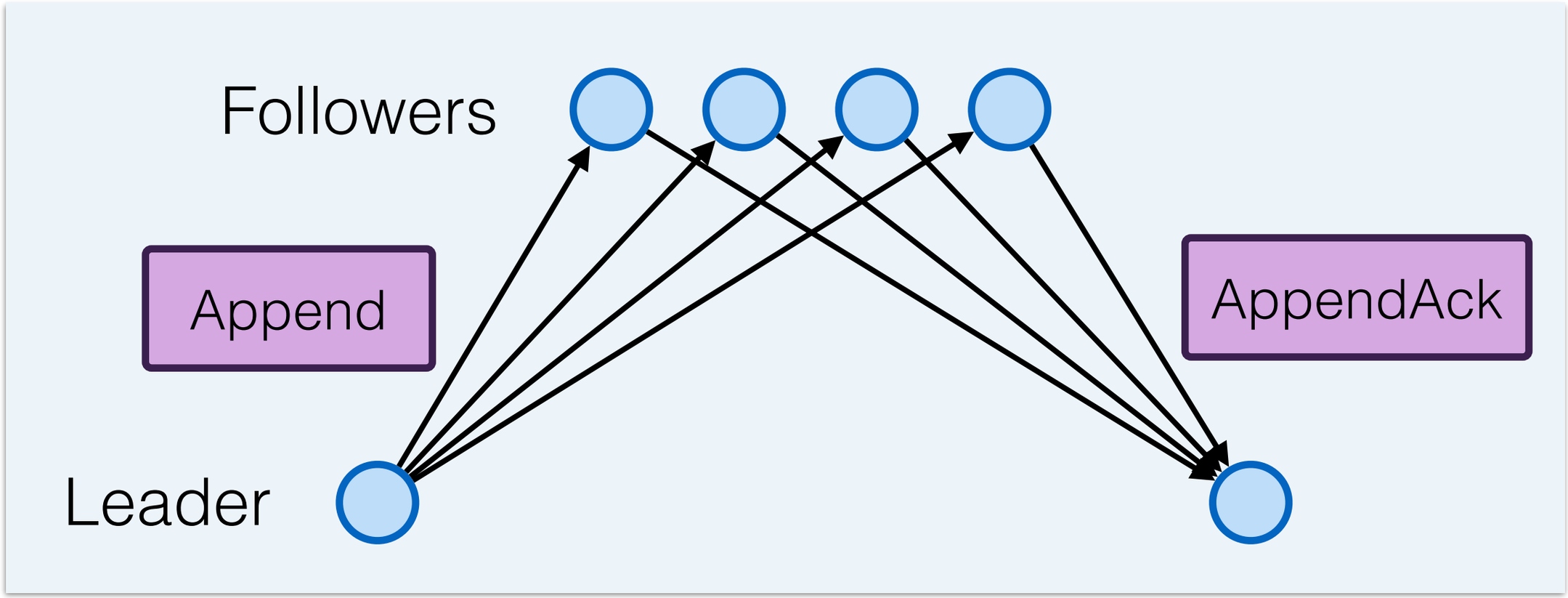




Raft: election and replication terms

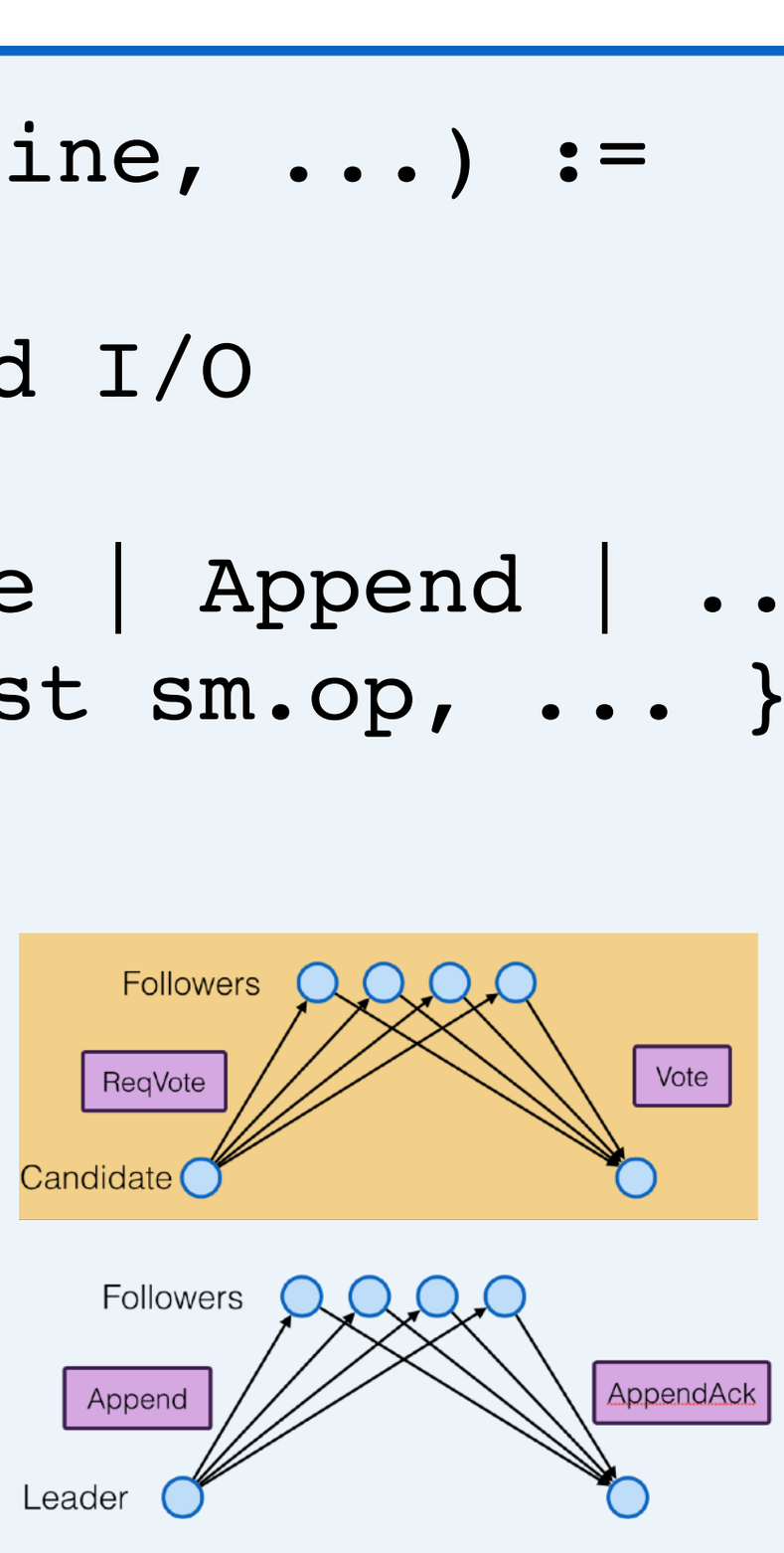


Raft: log replication

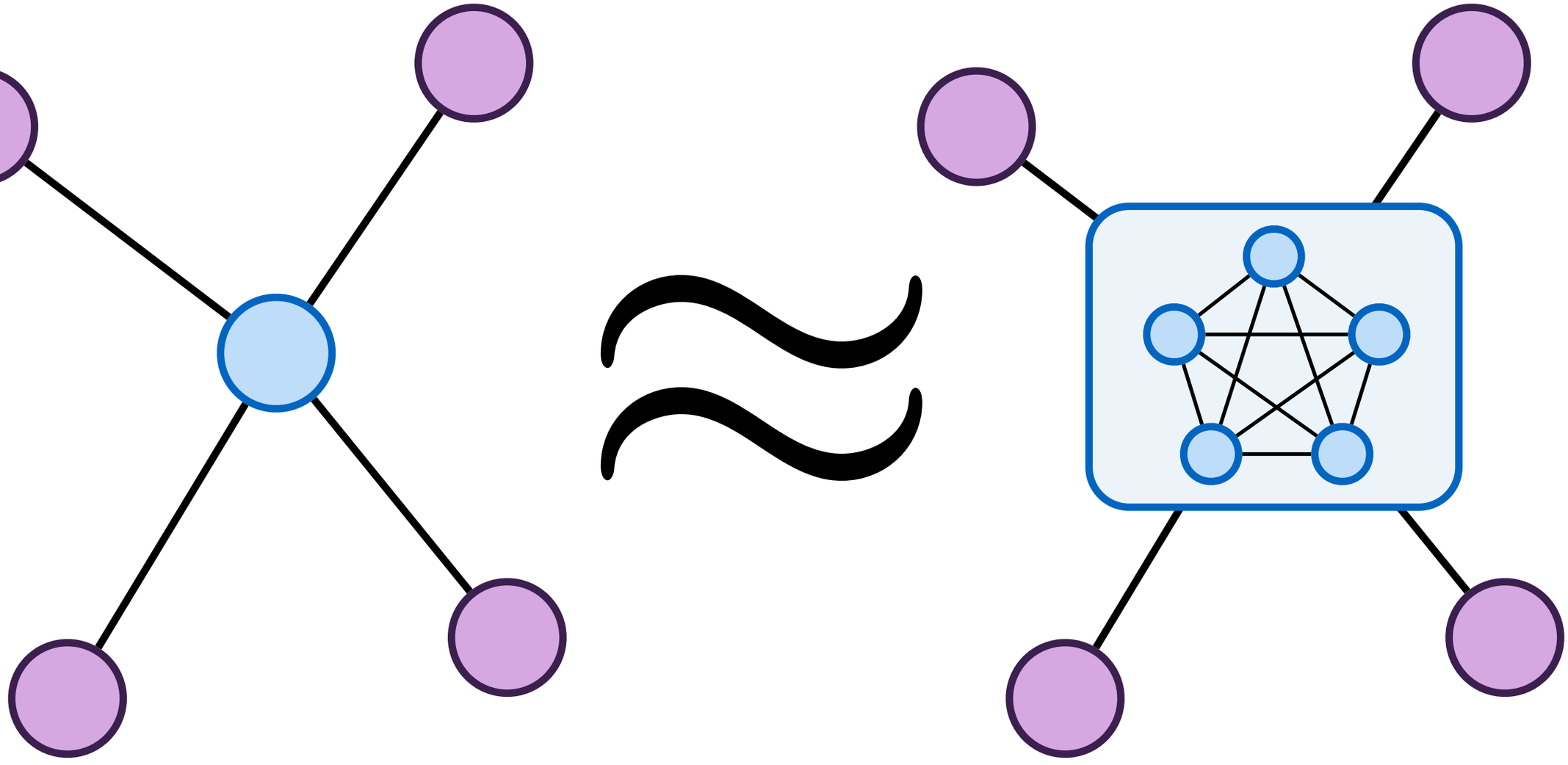


Defining Raft

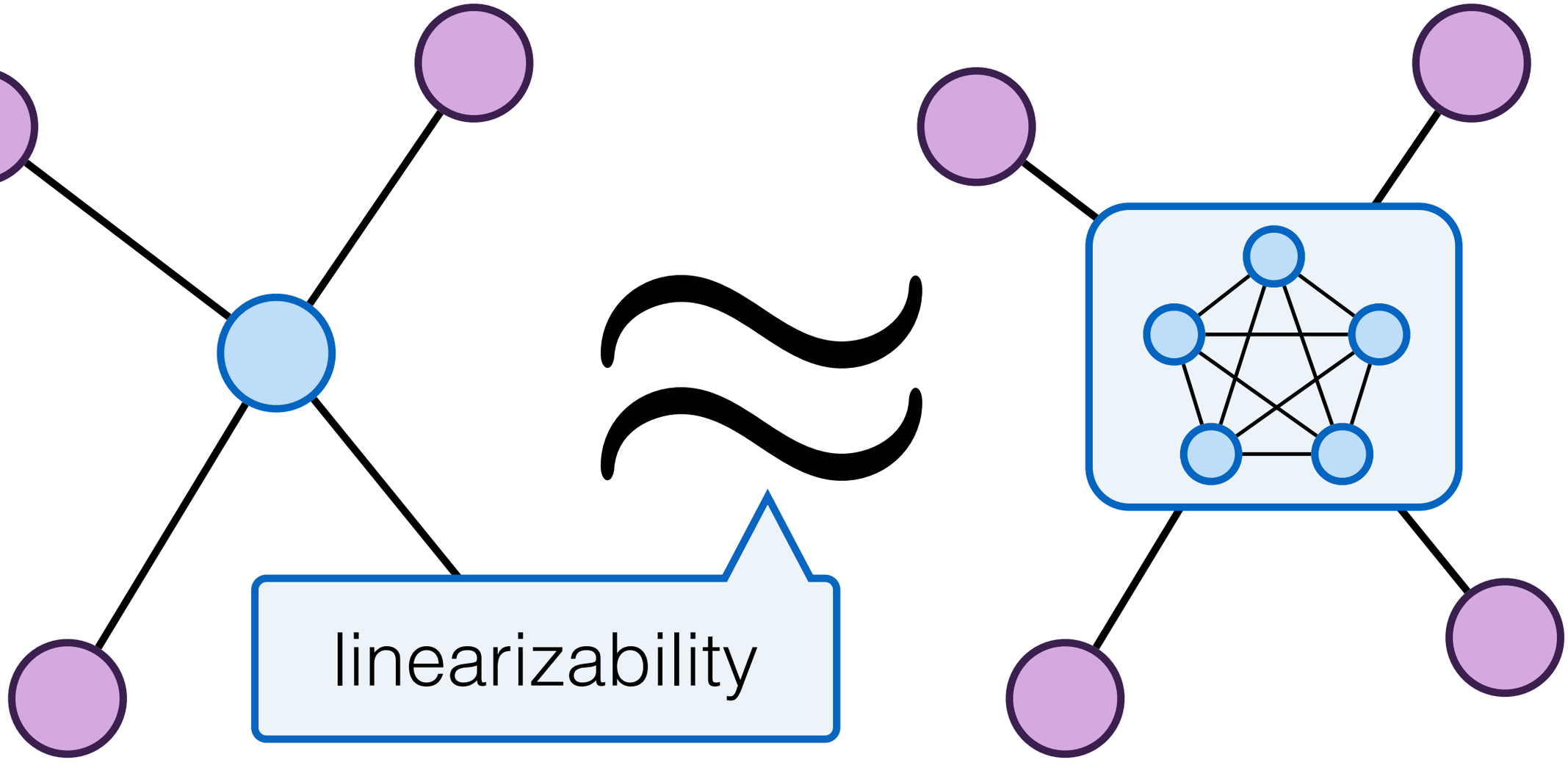
```
Def raft(sm: state machine, ...) :=

  // types for state and I/O
  cmsg := sm.cmsg
  msg  := ReqVote | Vote | Append | ...
  data := { sm.data, list sm.op, ... }


  // event handlers
  Def onMsg     :=
  Def onTmOut   :=
  Def onClient  :=
```



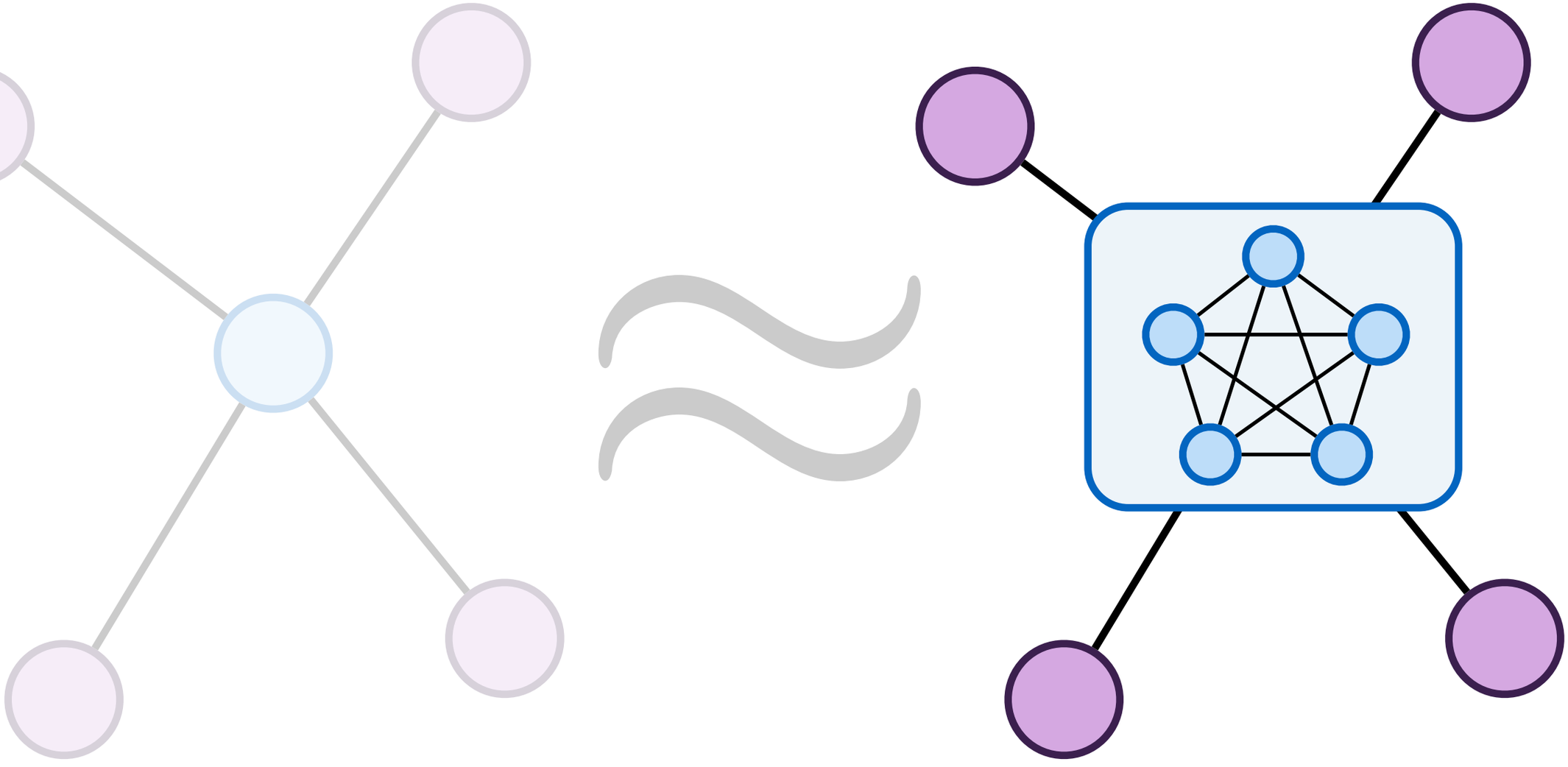
Verifying Raft



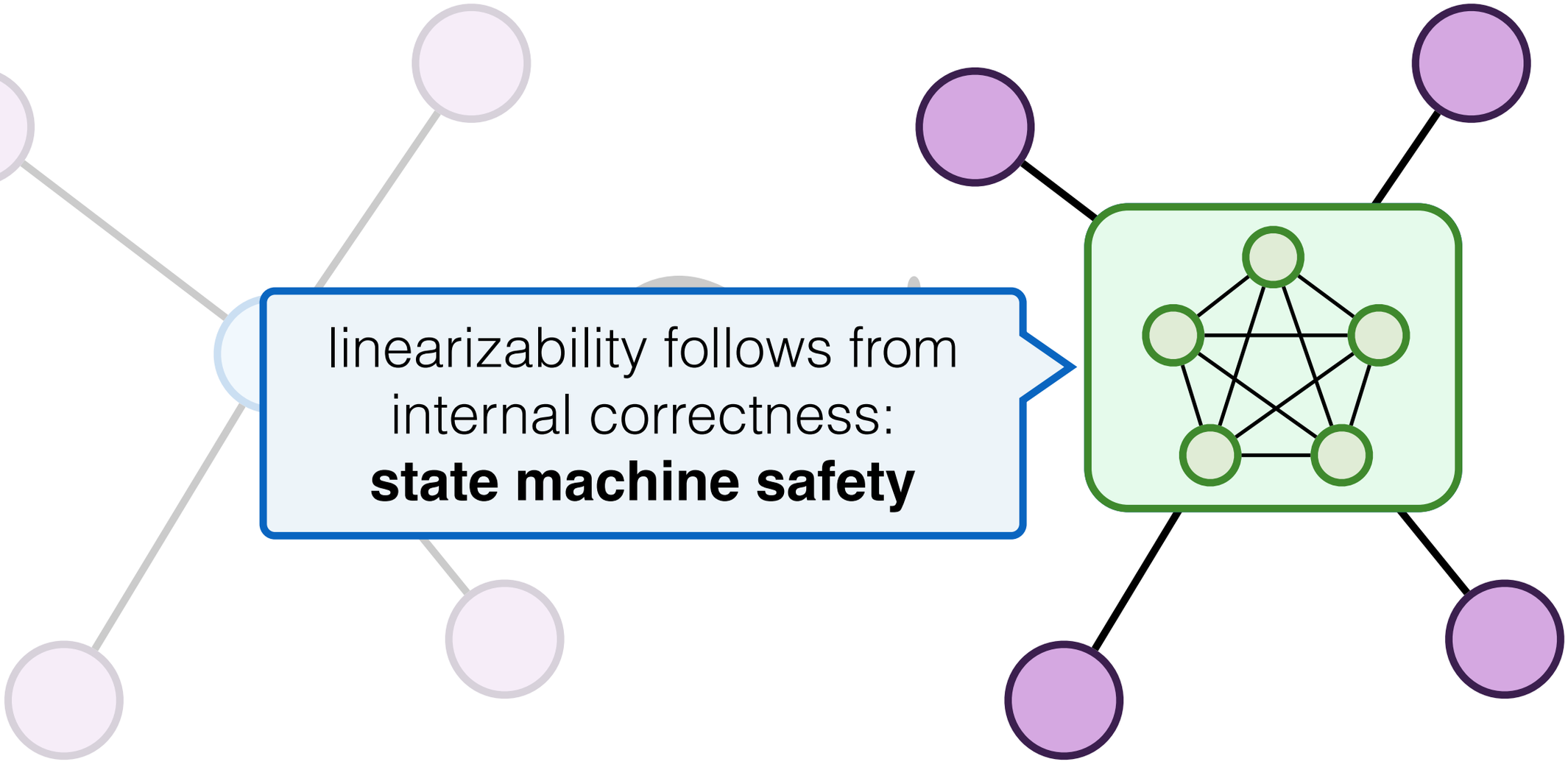
Verifying Raft



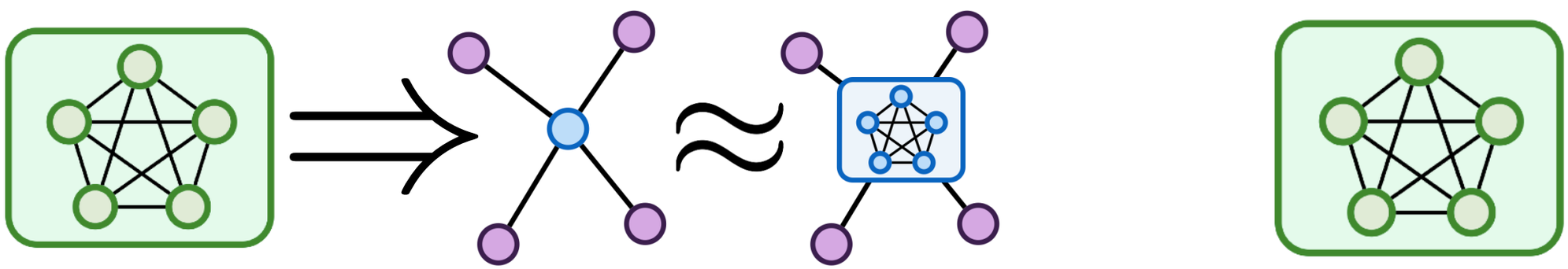
Raft internal correctness

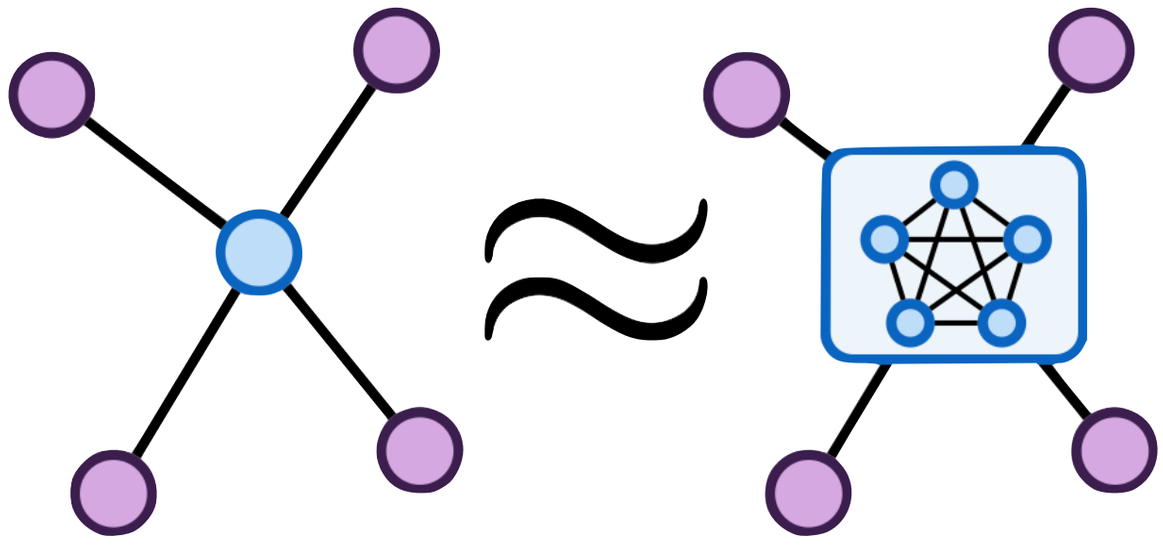


Raft internal correctness

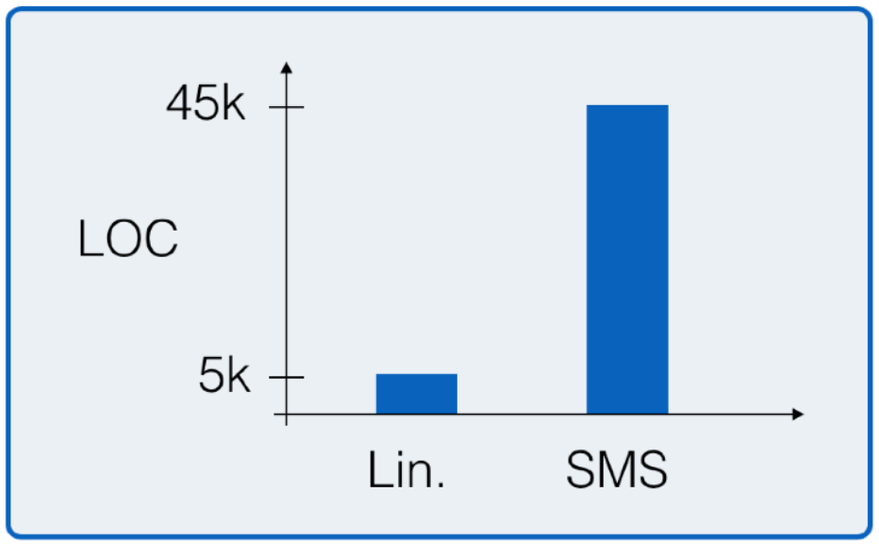


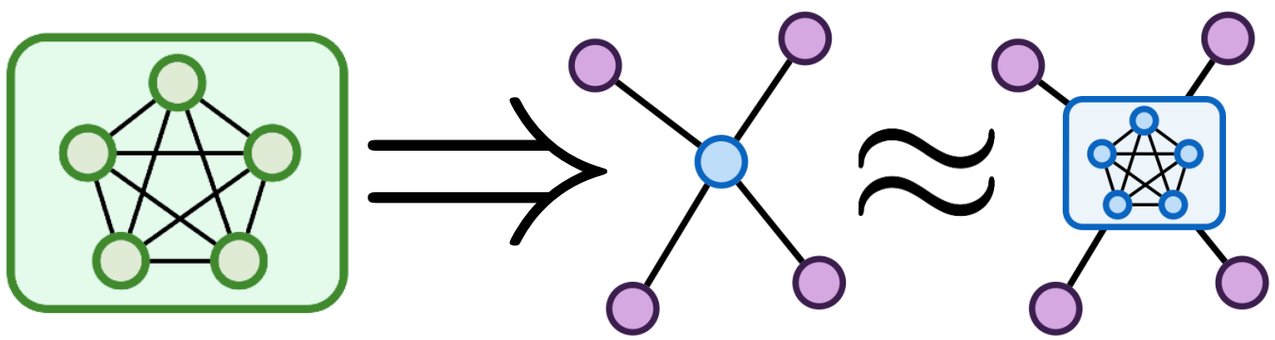
Proving Raft in Verdi

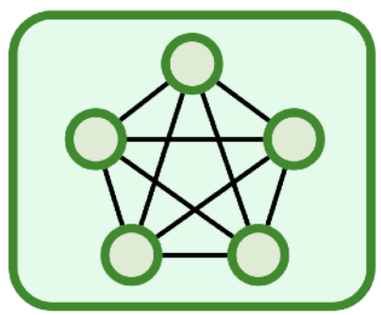


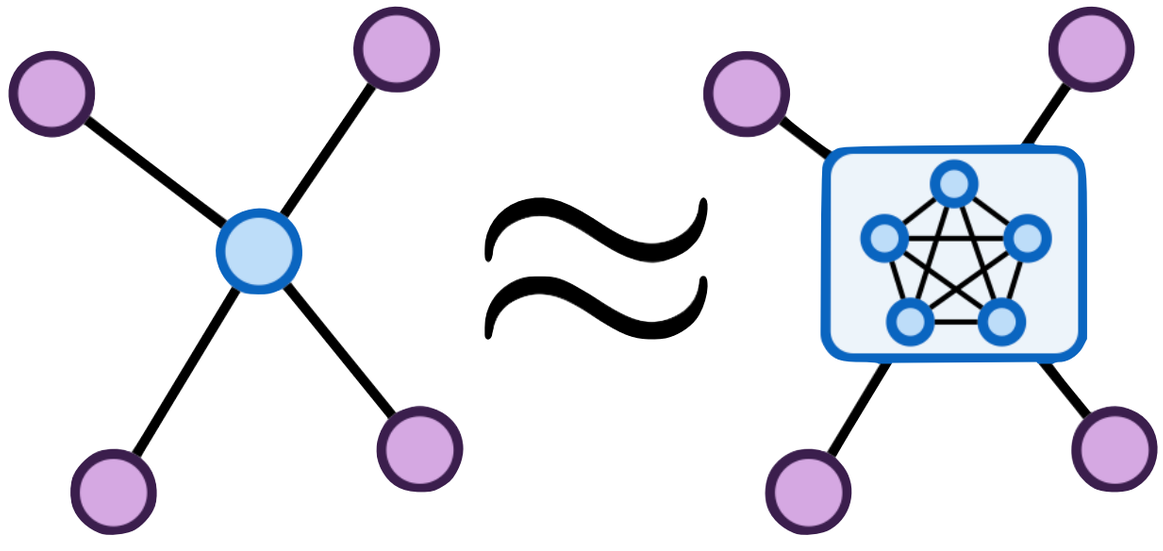


Proving Raft in Verdi



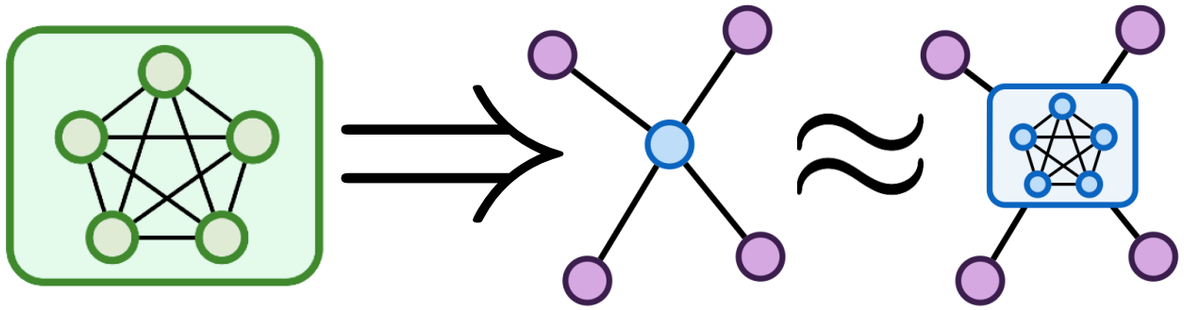




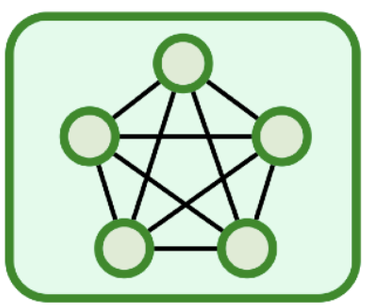


State machine safety

Nodes' logs match on committed entries

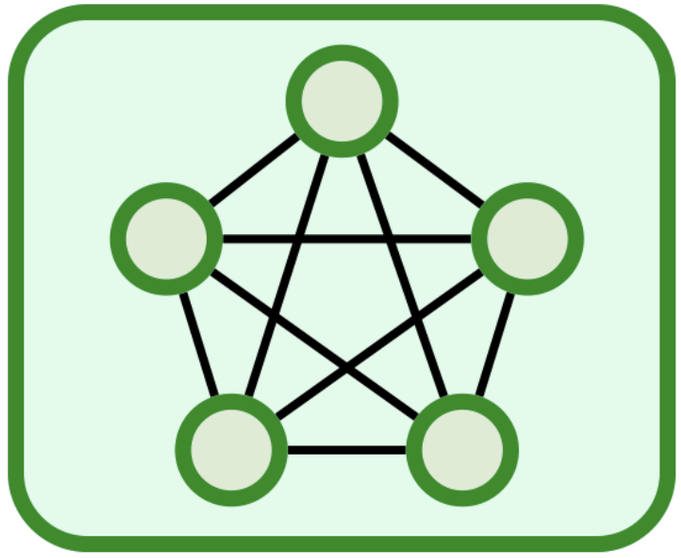


since only committed entries executed



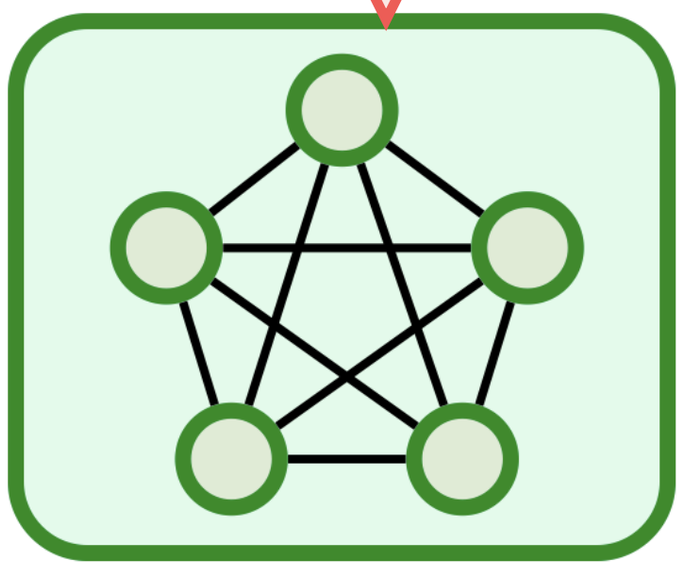
proof by induction over executions

State Machine Safety: Proof

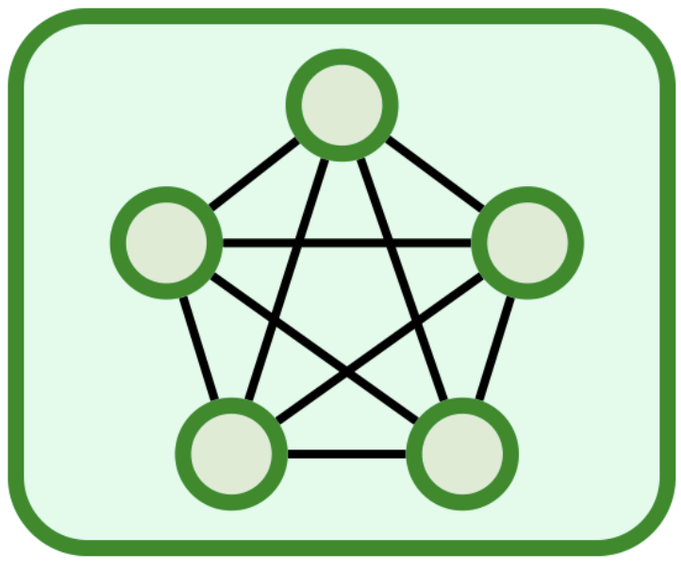


State Machine Safety: Proof

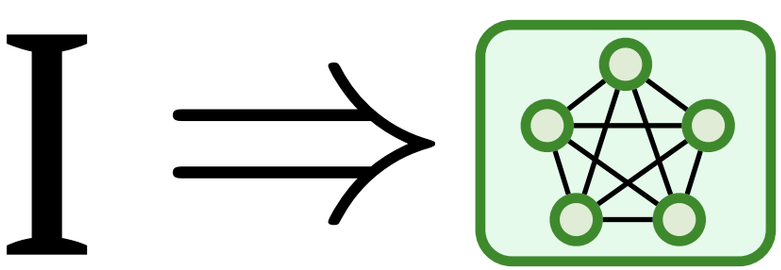
not inductive!

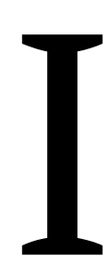


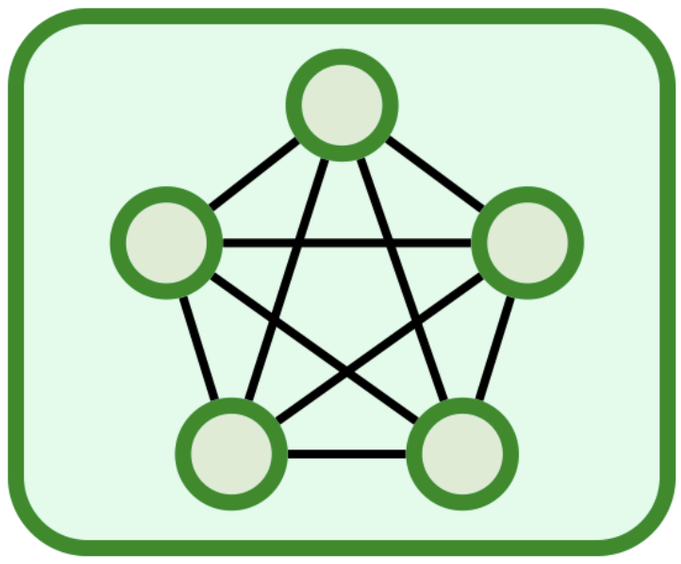
State Machine Safety: Proof



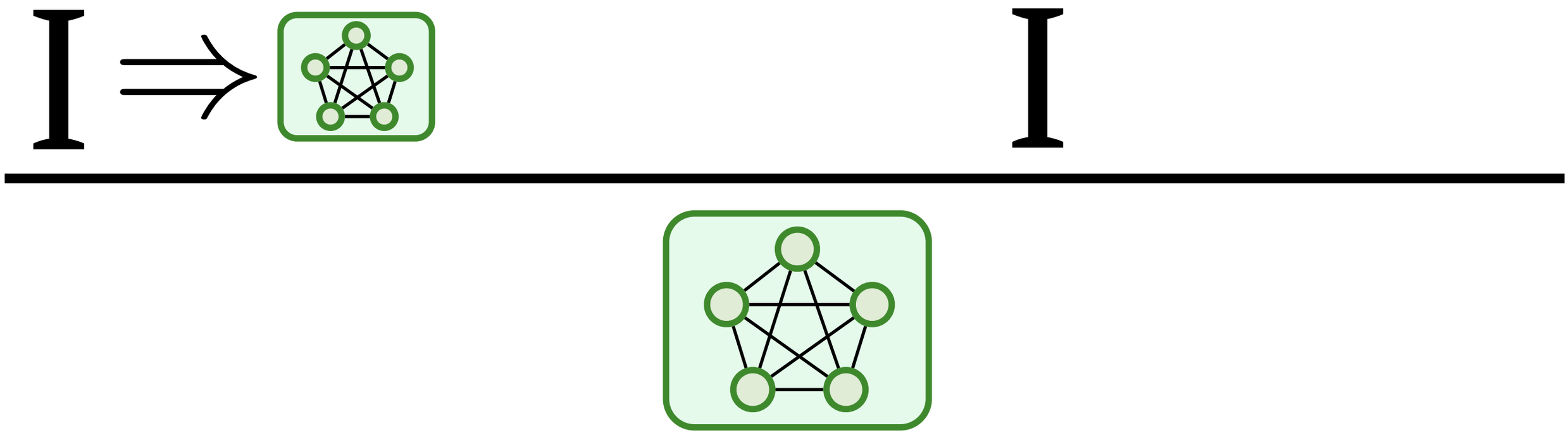
State Machine Safety: Proof





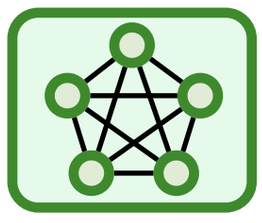


State Machine Safety: Proof

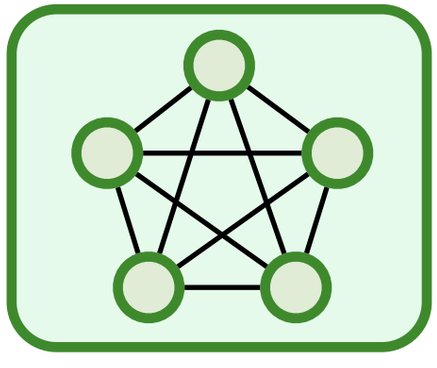


State Machine Safety: Proof

I true initially $\quad$ I preserved

$I \Rightarrow$ 

I



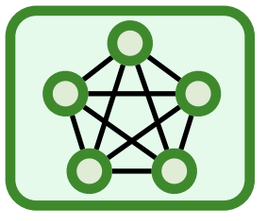
State Machine Safety: Proof

90 invariants in total

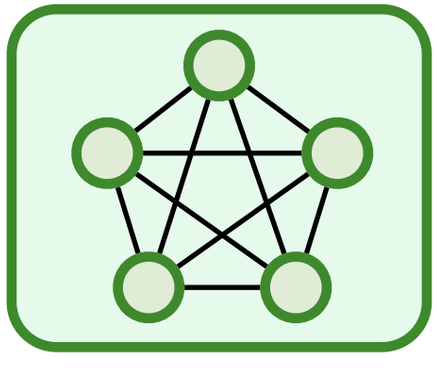
Lemma Lemma Lemma ...

$\mathbf{I}$ true initially

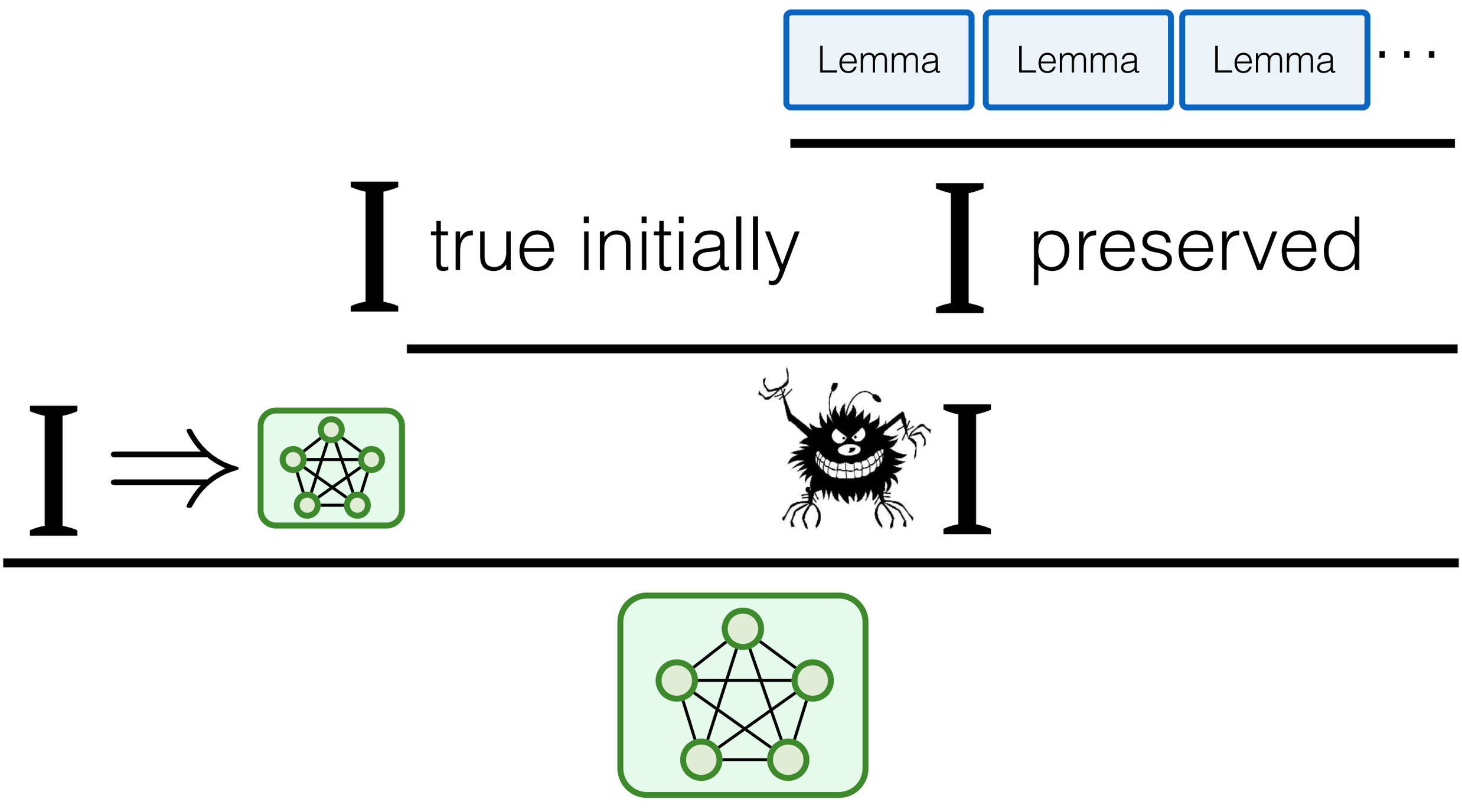
$\mathbf{I}$ preserved

$\mathbf{I} \Rightarrow$ 

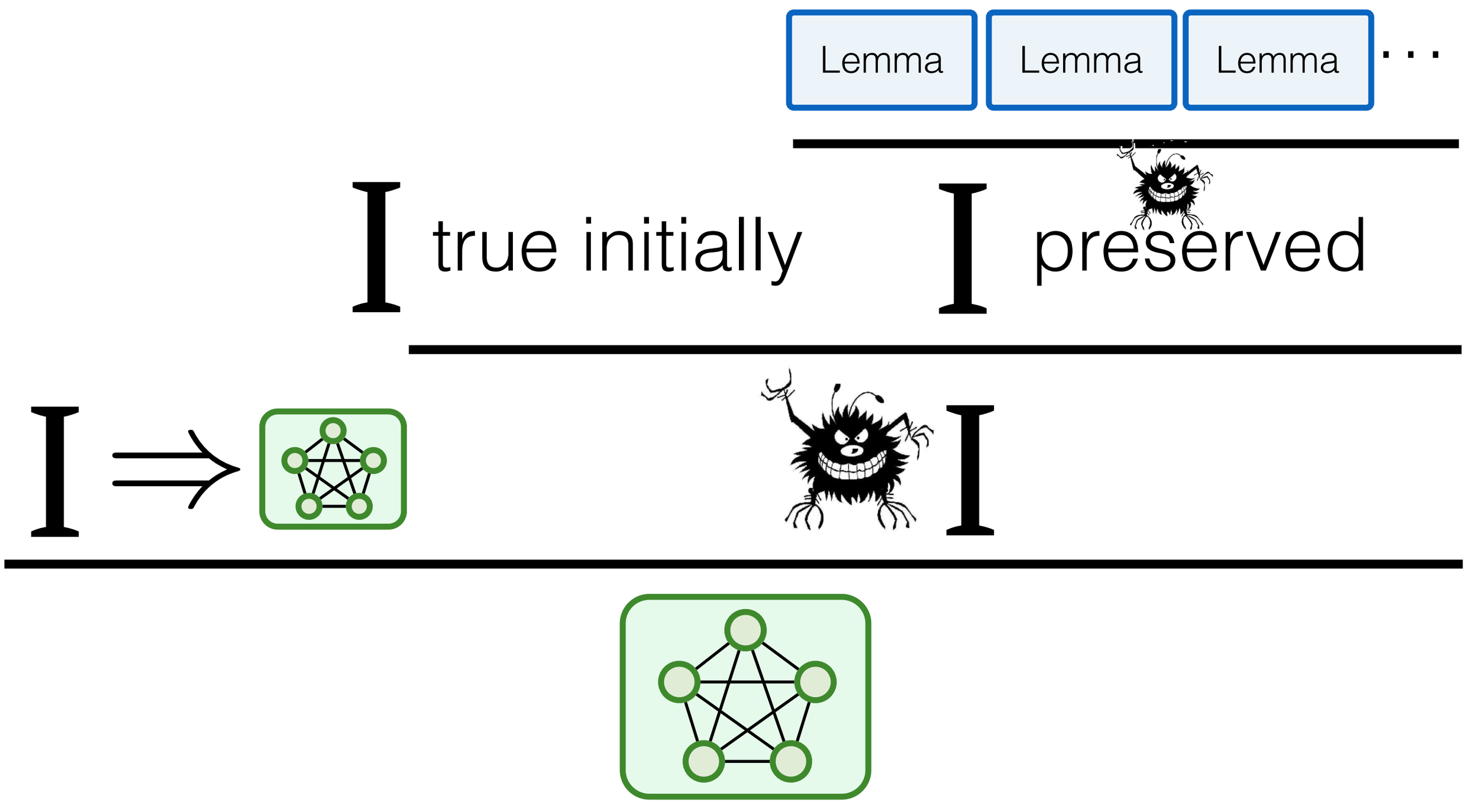
$\mathbf{I}$



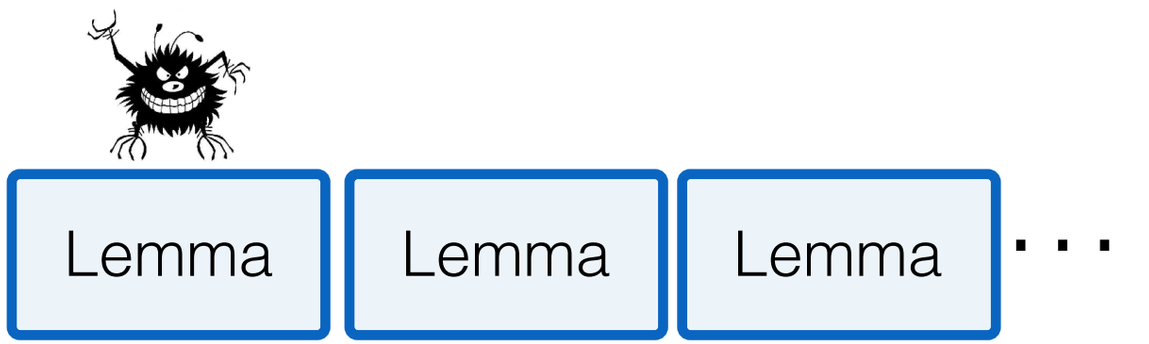
State Machine Safety: Proof



State Machine Safety: Proof

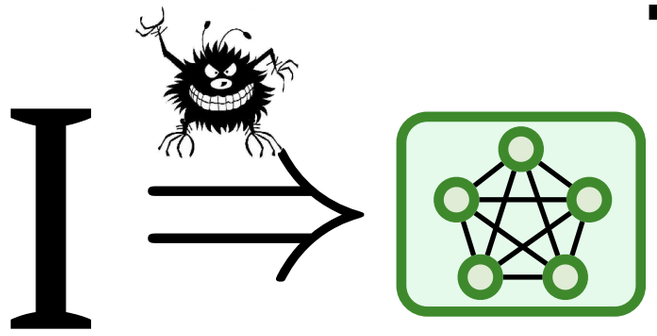


State Machine Safety: Proof

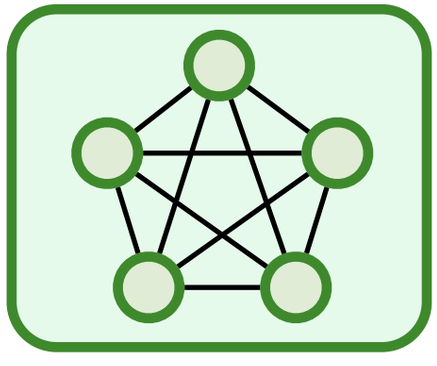


I true initially

I preserved



I



The burden of proof

Re-verification is the primary challenge:

- invariants are not inductive
- not-yet-verified code is wrong
- need additional invariants

The burden of proof

Re-verification is the primary challenge 

P true initially

P preserved

P with ghost state

P

The burden of proof

Re-verification is the primary challenge

Proof engineering techniques help:

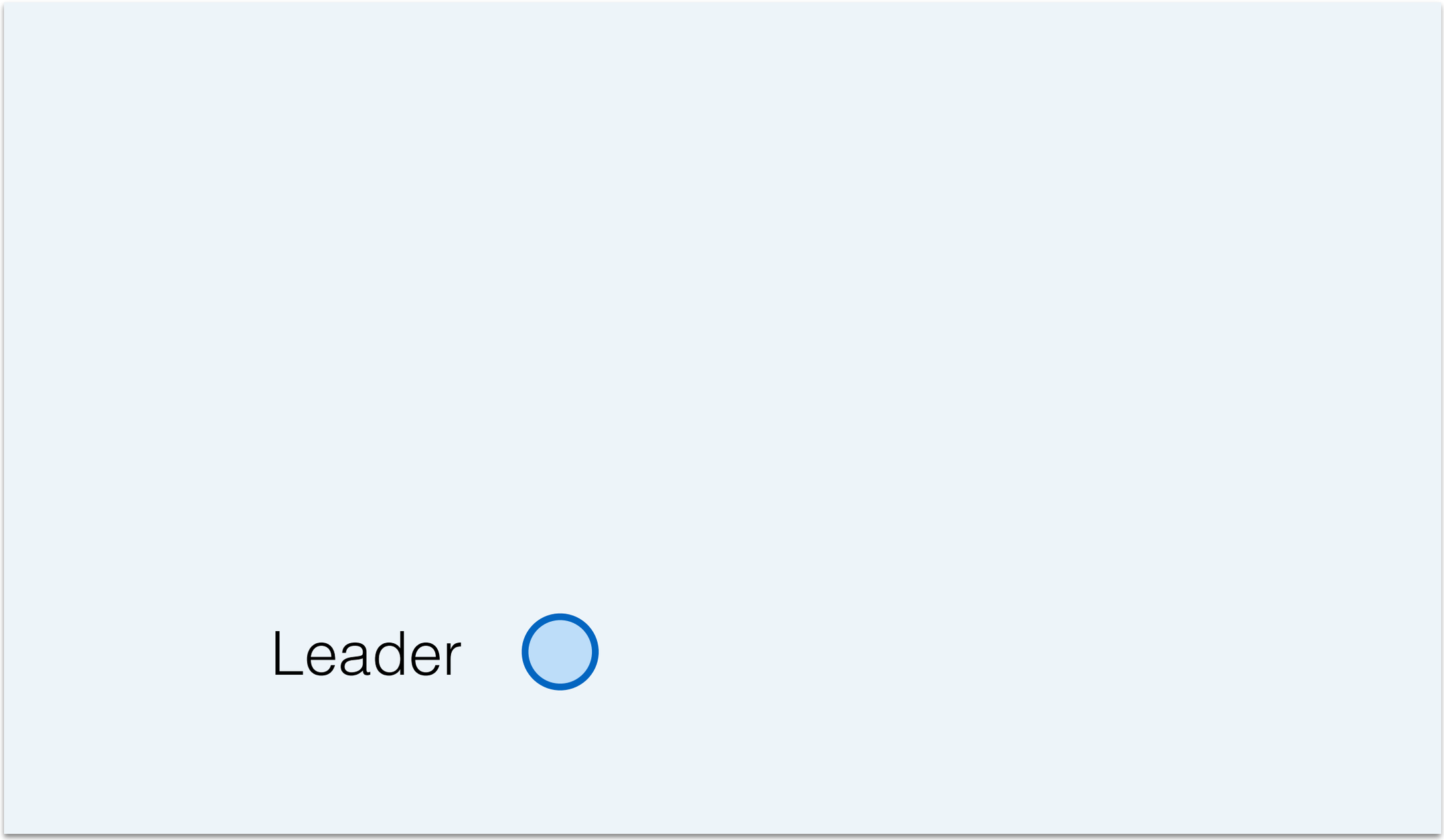
- affinity lemmas
- intermediate reachability
- structural tactics
- information hiding

Ghost state: example

Capture all entries received by a node

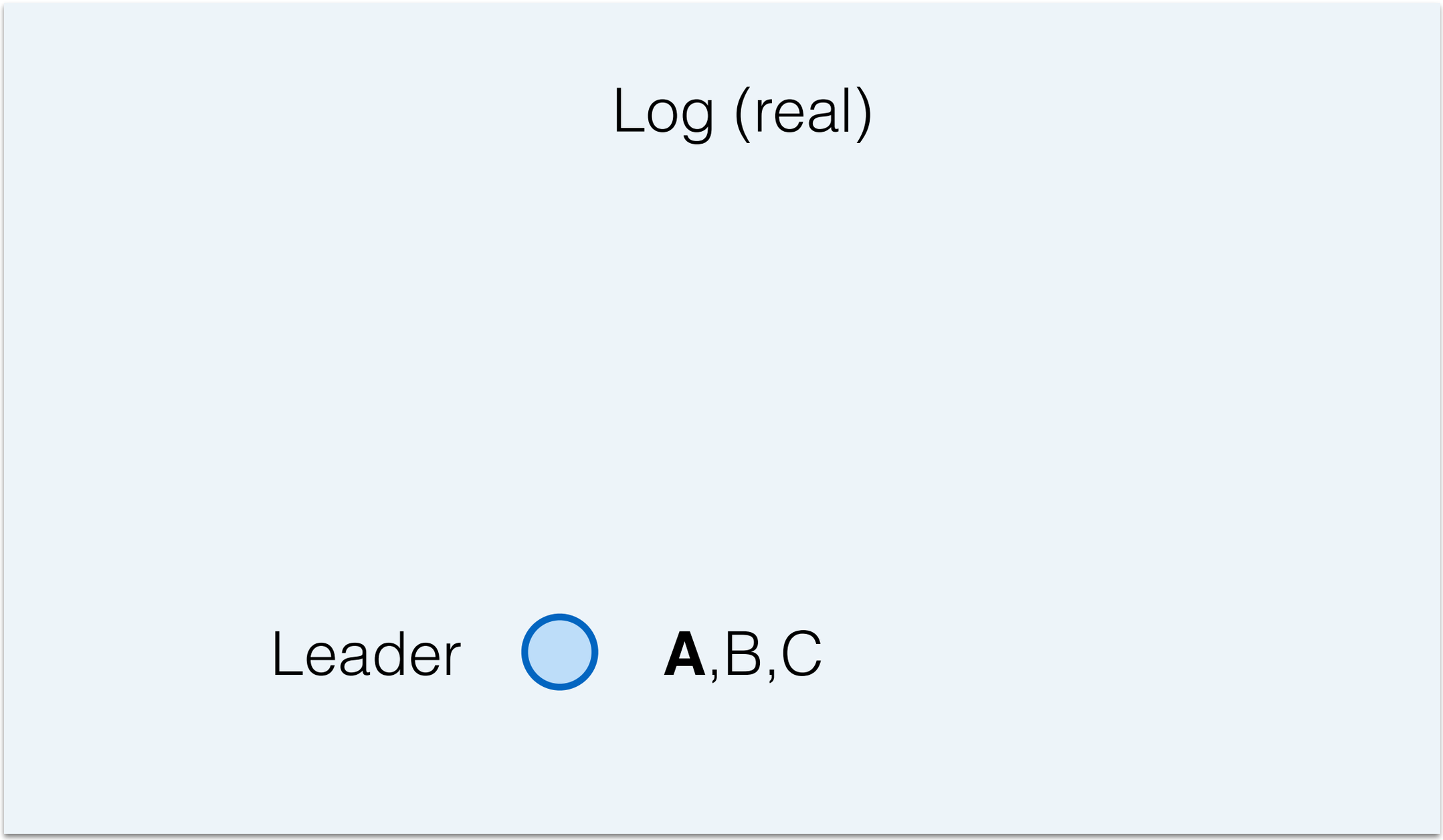
Ghost state: example

Capture all entries received by a node



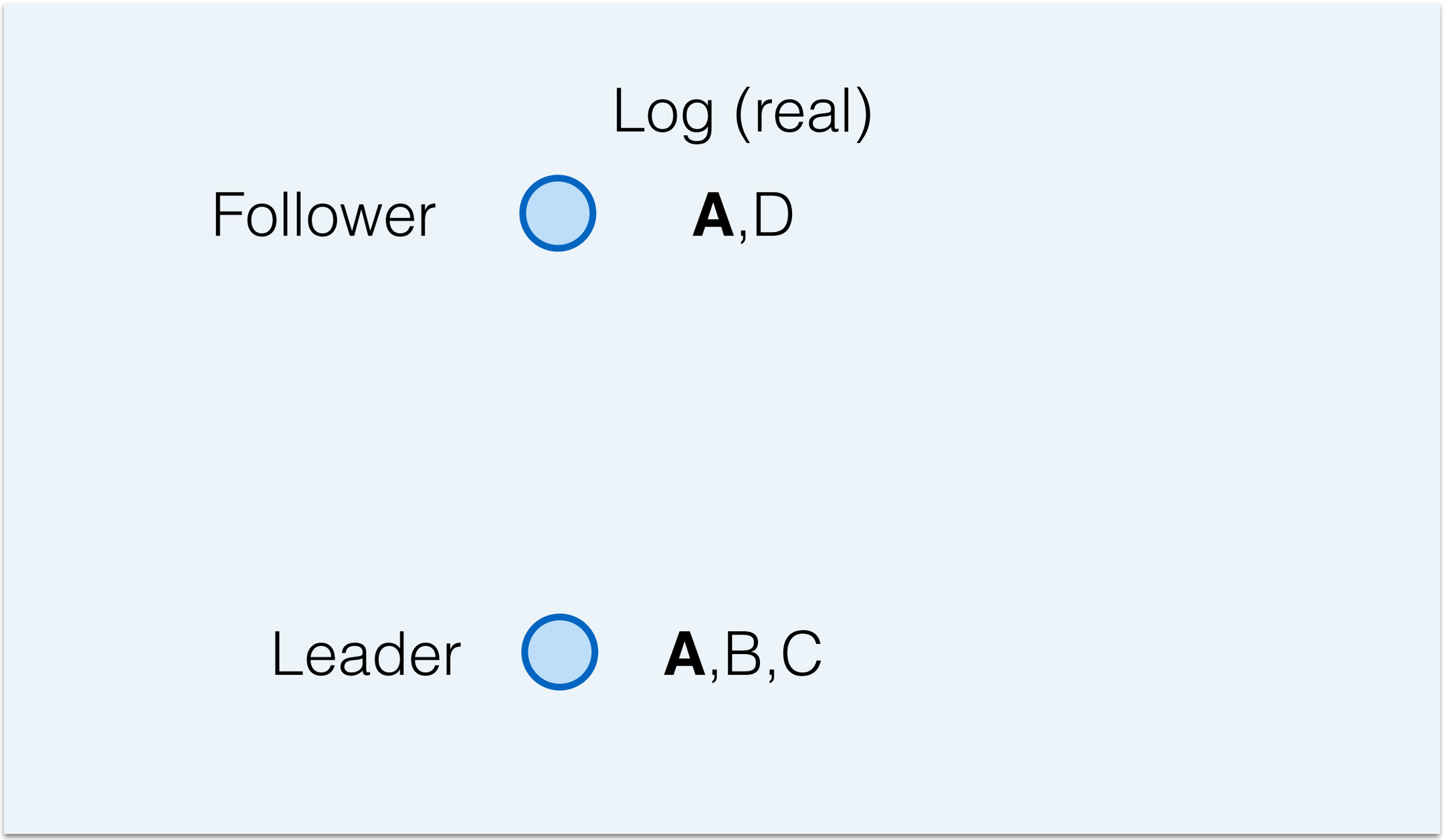
Ghost state: example

Capture all entries received by a node



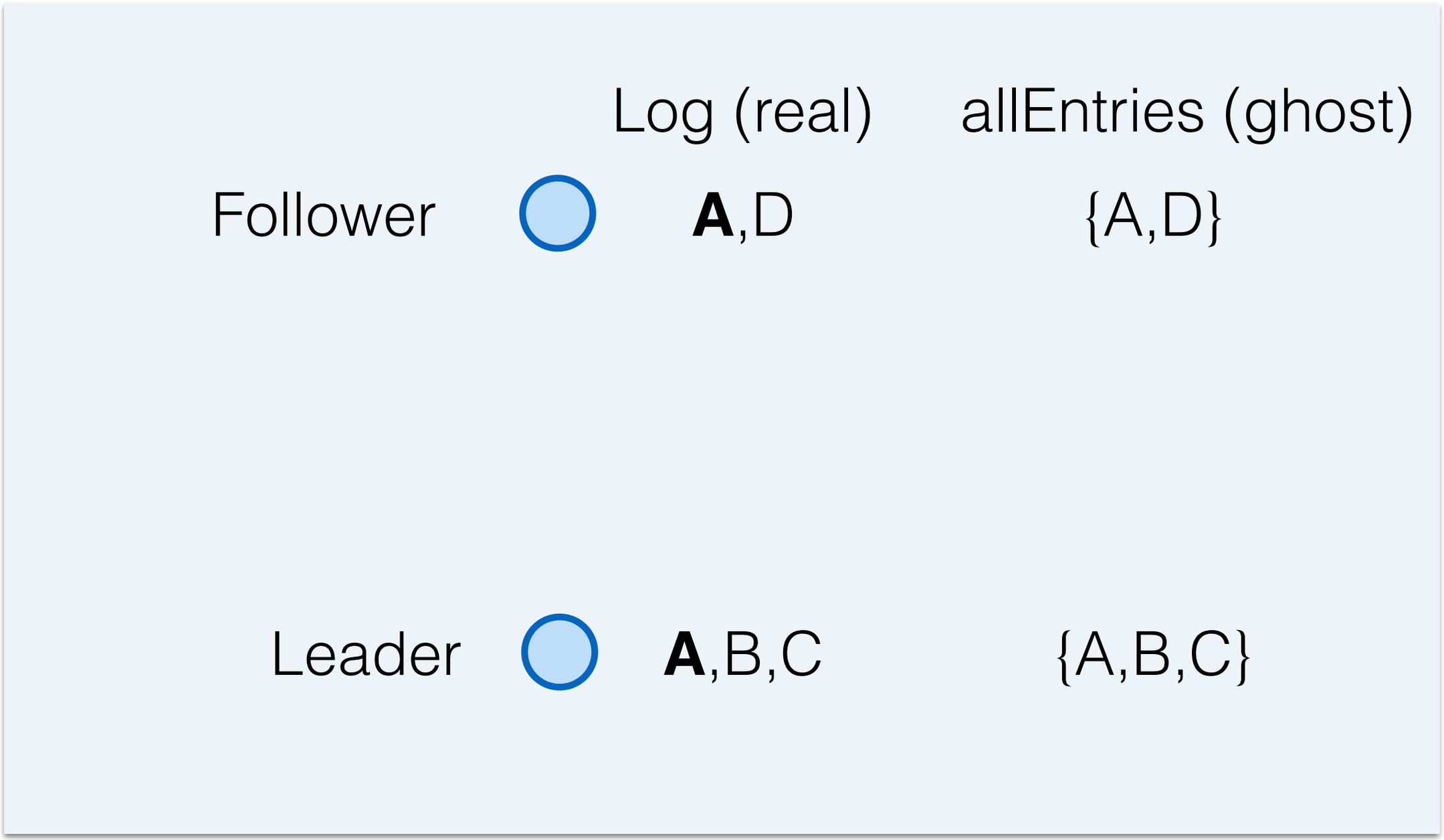
Ghost state: example

Capture all entries received by a node



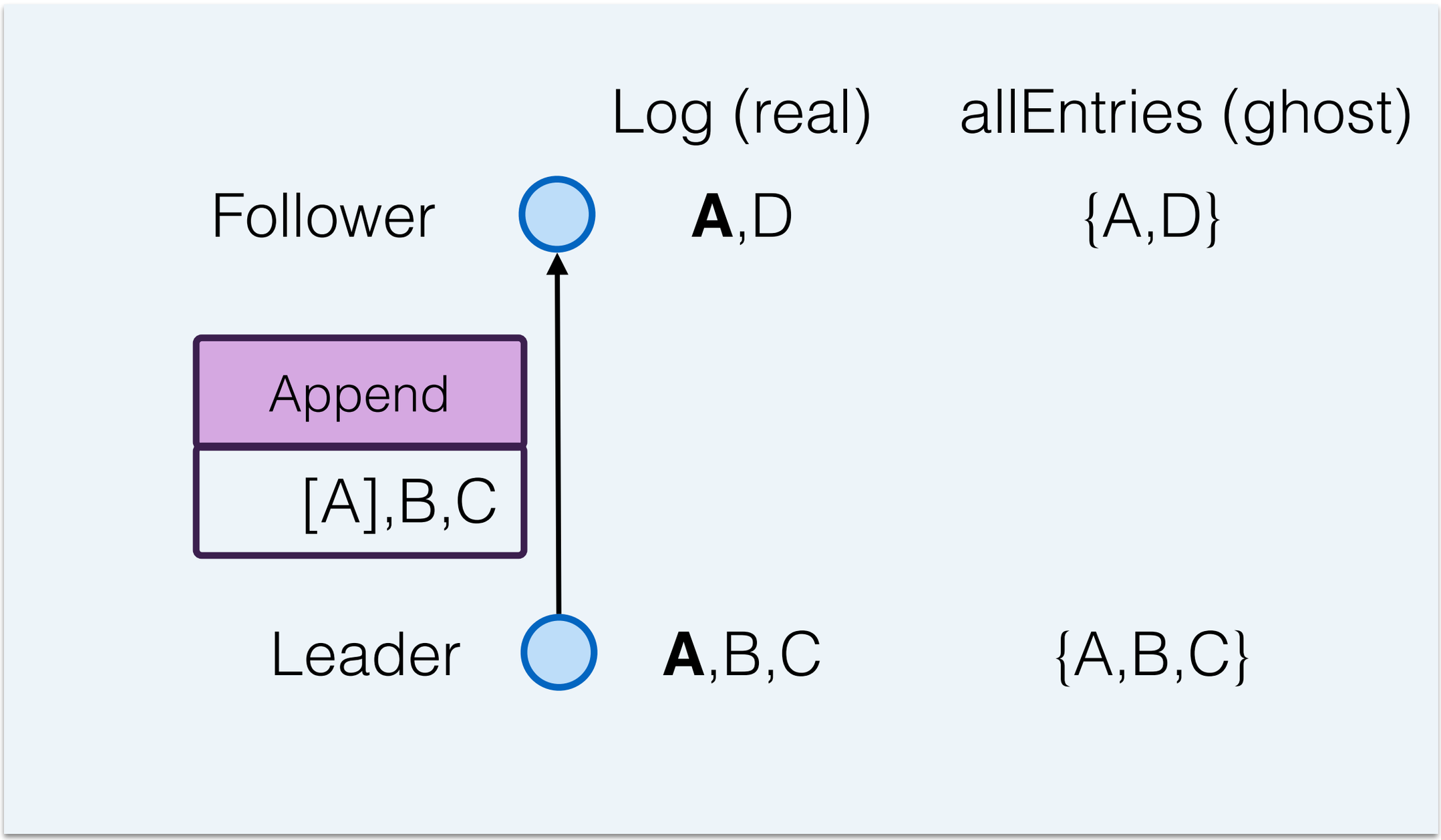
Ghost state: example

Capture all entries received by a node



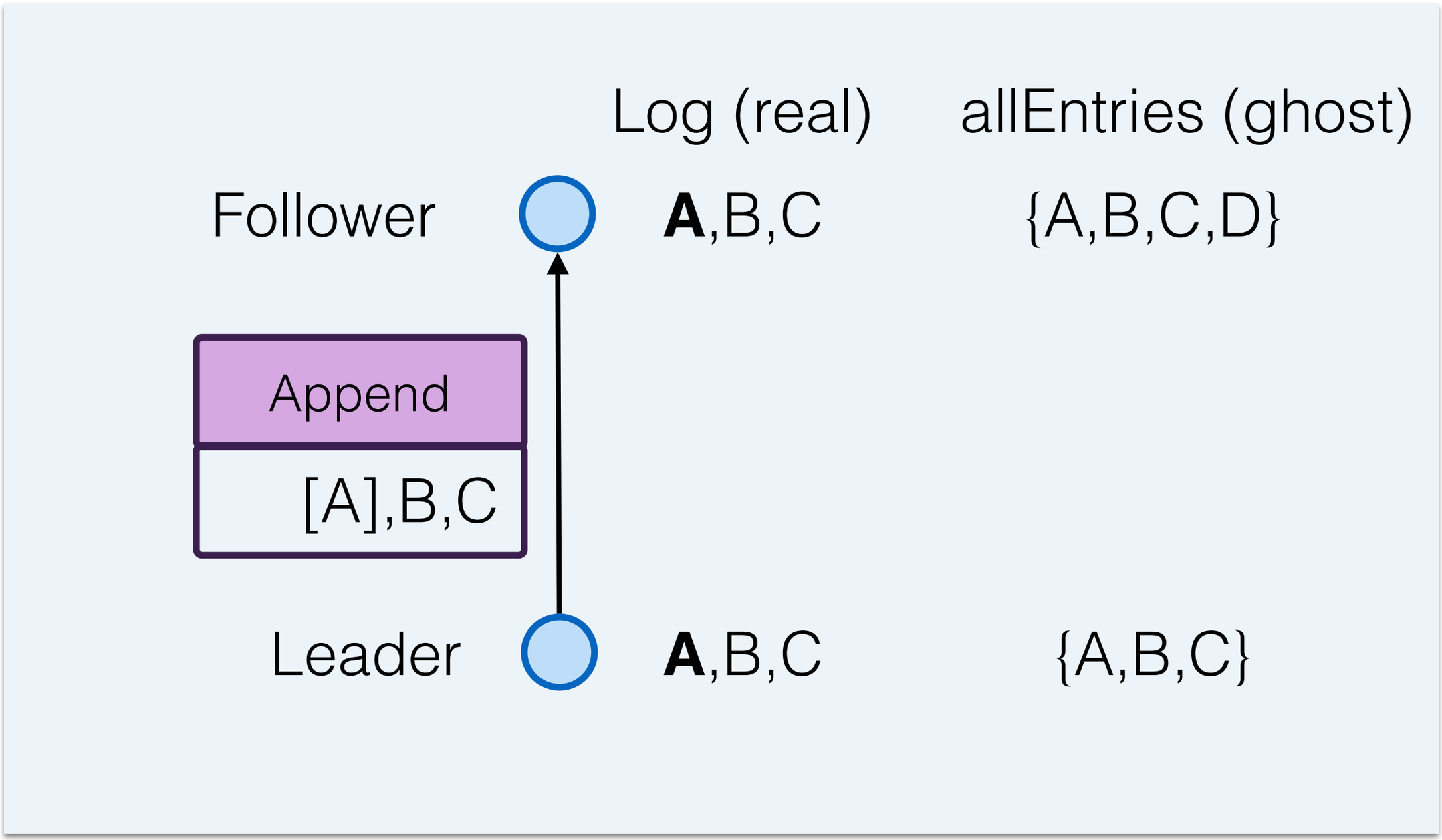
Ghost state: example

Capture all entries received by a node



Ghost state: example

Capture all entries received by a node



Affinity lemmas: example

$$e \in \text{allEntries} \implies e.\text{term} > 0$$

Affinity lemmas: example

$e \in log \Rightarrow$

$e.term > 0$

$e \in allEntries \Rightarrow e.term > 0$

Affinity lemmas: example

$e \in log \Rightarrow$

$e.term > 0$

Affinity Lemma

$e \in allEntries \Rightarrow e.term > 0$

Affinity lemmas: example

$e \in log \Rightarrow$

$e.term > 0$

every invariant of entries in logs is invariant of entries in allEntries

---

$e \in allEntries \Rightarrow e.term > 0$

Affinity lemmas: example

$$\frac{e \in log \Rightarrow P\ e}{e \in allEntries \Rightarrow P\ e}$$

every invariant of entries in logs is invariant of entries in allEntries

More affinity lemmas

Relate ghost state to real state

transfer properties once and for all

Relate current messages to past

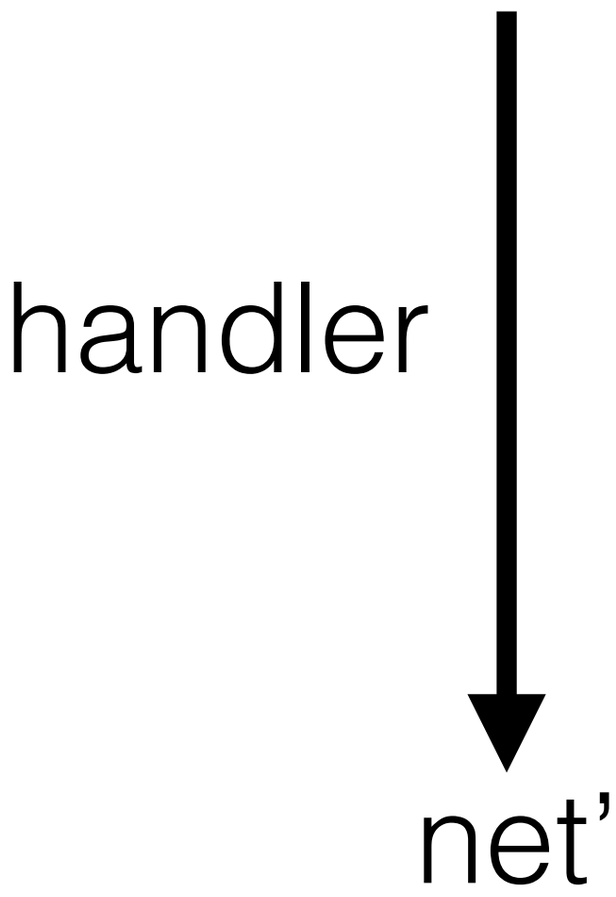
response => past request

Structured handlers

handler = update_state ; respond

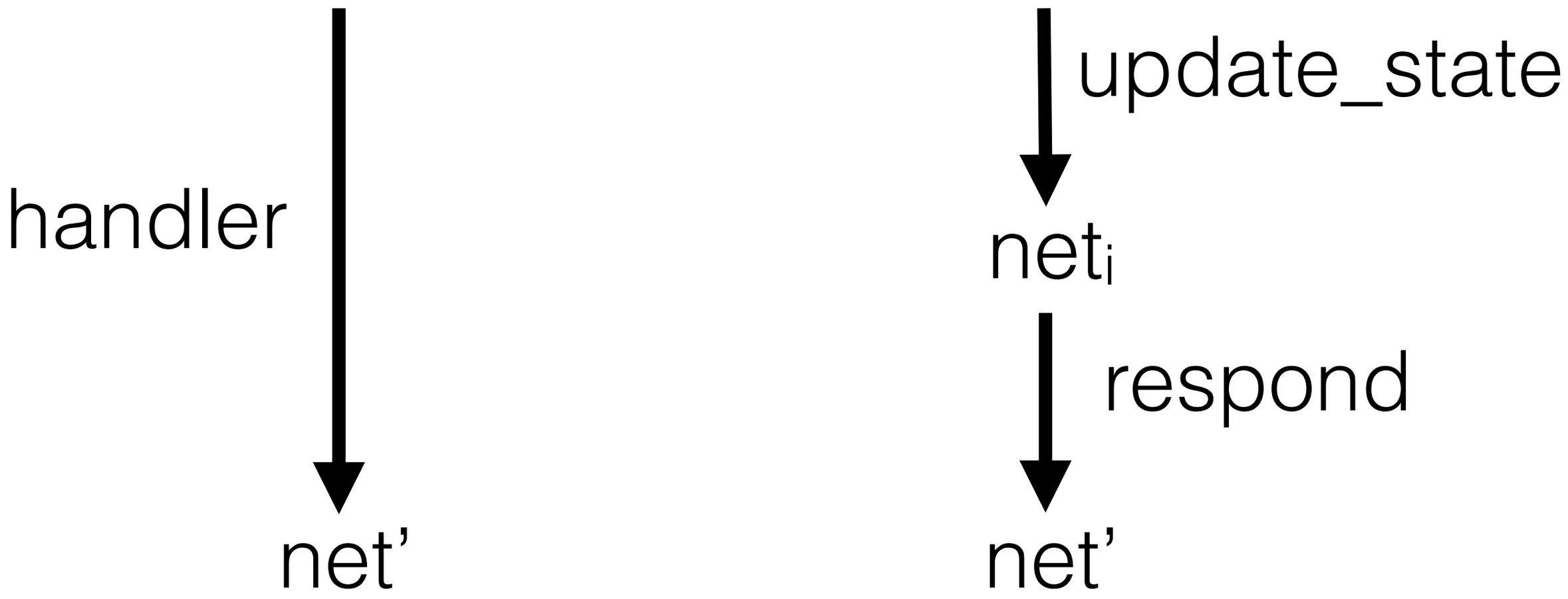
Structured handlers

handler = update_state ; respond



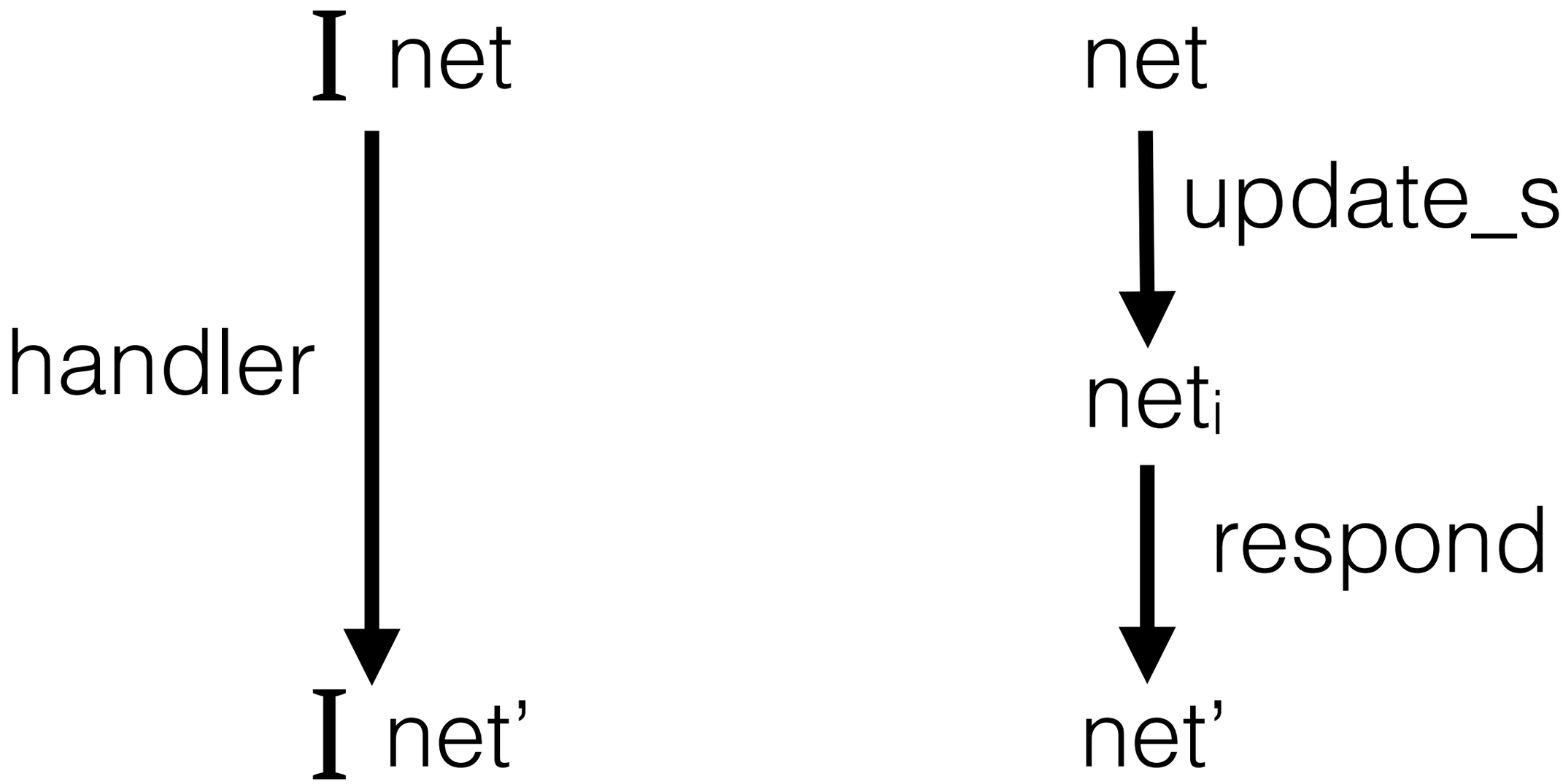
Structured handlers

handler = update_state ; respond



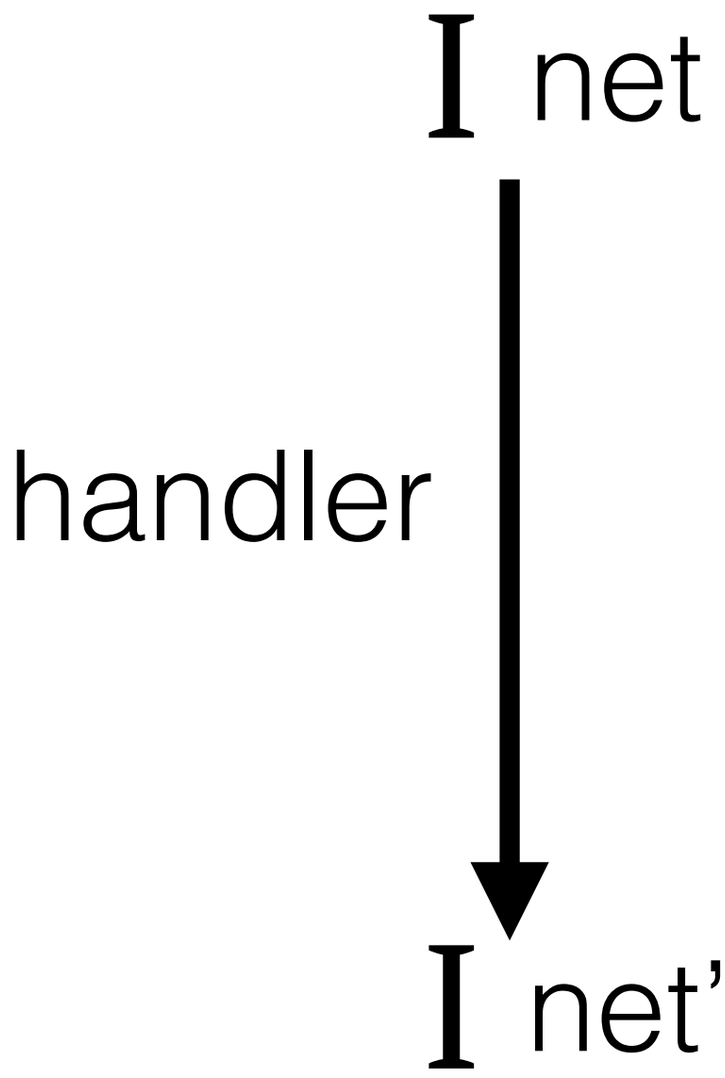
Structured handlers

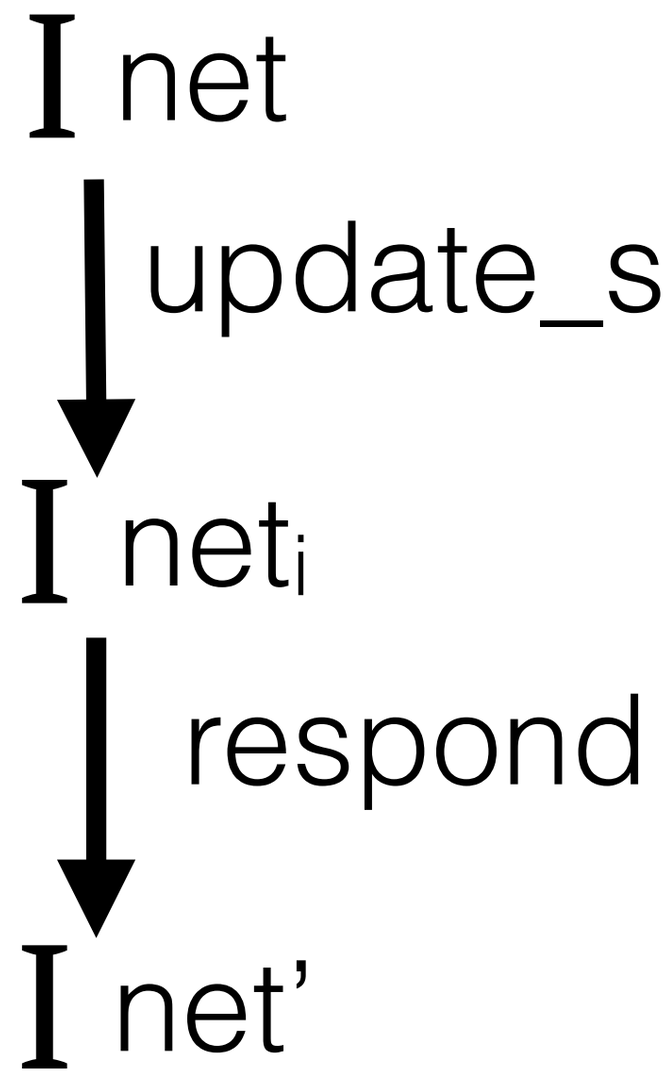
handler = update_state ; respond



Structured handlers

handler = update_state ; respond





First formal verification of Raft

50k lines of Coq

18 person-months

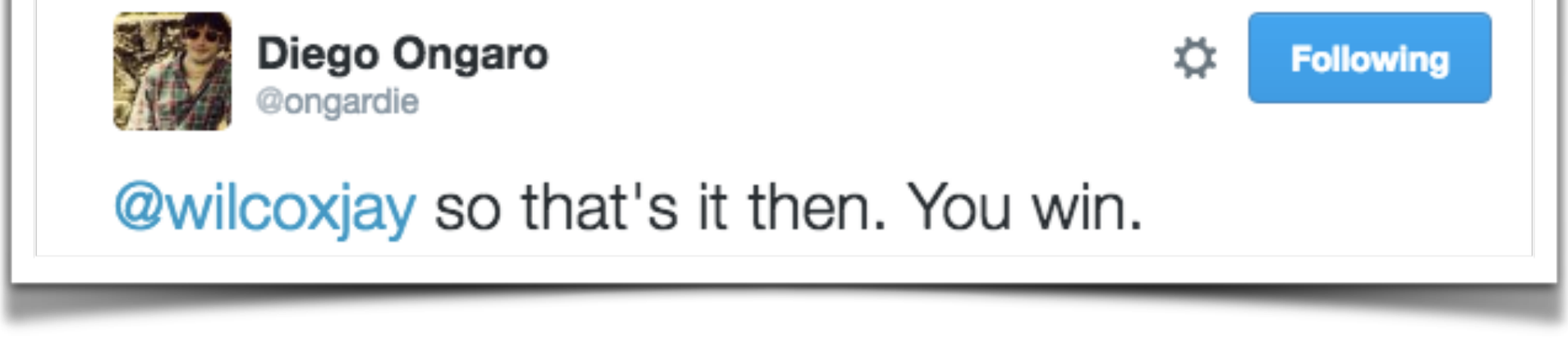
Considerable pizza and beer

First formal verification of Raft

50k lines of Coq

18 person-months

Considerable pizza and beer

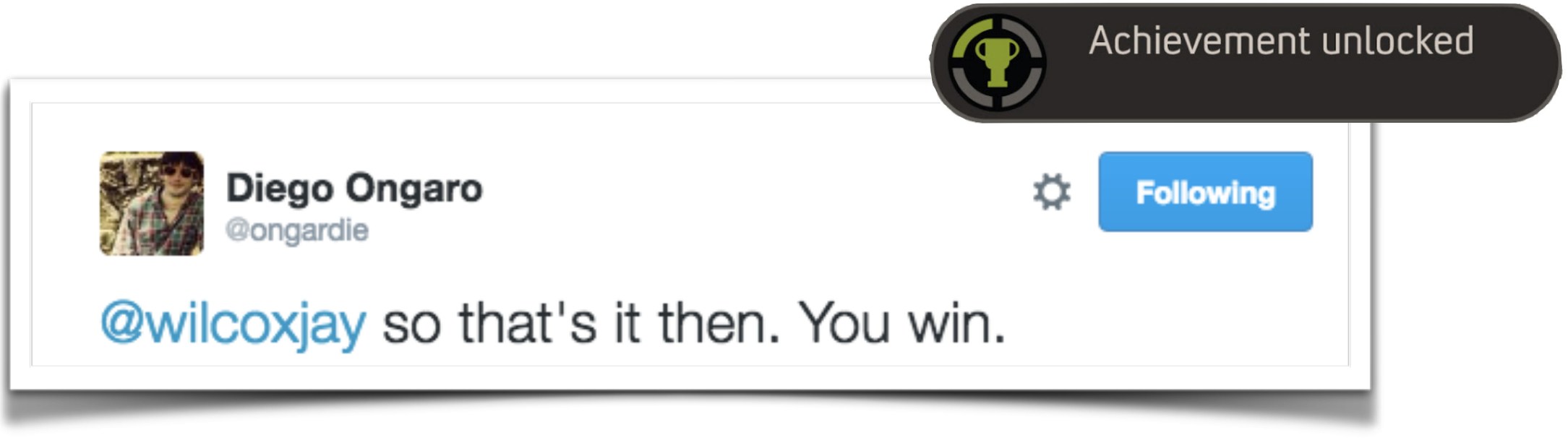


First formal verification of Raft

50k lines of Coq

18 person-months

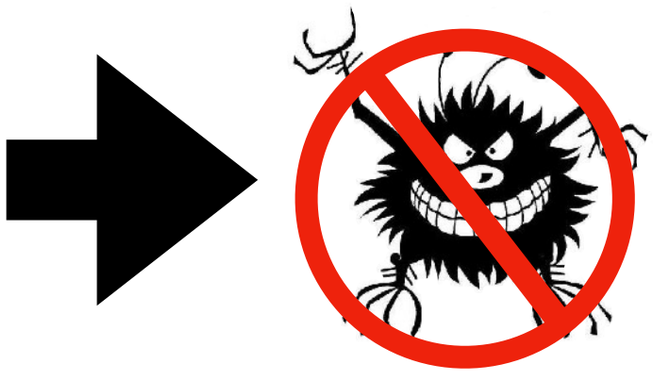
Considerable pizza and beer



Toward verified distributed systems

The Verdi Framework

Verified Raft Consensus



TCB, Tools, Teaching

Enriching Models & Modularity

Verified $\neq$ perfect

Network semantics shim is delicate

atomicity, fairness, serialization,…

Verdi users need Coq + distr sys skills

notorious learning curves hinder impact

Regular development still tricky

maintenance, extension, management

Network semantics shim is delicate

$$\frac{H_{\text{net}}(dst,\ \Sigma[dst],\ src,\ m) = (\sigma',\ o,\ P') \qquad \Sigma' = \Sigma[dst \mapsto \sigma']}{(\{(src,\ dst,\ m)\} \uplus P,\ \Sigma,\ T) \rightsquigarrow (P \uplus P',\ \Sigma',\ T +\!\!+ \langle o \rangle)}\ \textsc{Deliver}$$

$$\frac{p \in P}{(P,\ \Sigma,\ T) \rightsquigarrow (P \uplus \{p\},\ \Sigma,\ T)}\ \textsc{Duplicate}$$

$$\frac{}{(\{p\} \uplus P,\ \Sigma,\ T) \rightsquigarrow (P,\ \Sigma,\ T)}\ \textsc{Drop}$$

$$\frac{H_{\text{tmt}}(n,\ \Sigma[n]) = (\sigma',\ o,\ P') \qquad \Sigma' = \Sigma[n \mapsto \sigma']}{(P,\ \Sigma,\ T) \rightsquigarrow (P \uplus P',\ \Sigma',\ T +\!\!+ \langle \text{tmt},\ o \rangle)}\ \textsc{Timeout}$$

Network semantics shim is delicate

$$\frac{H_{\text{net}}(dst,\ \Sigma[dst],\ src,\ m) = (\sigma',\ o,\ P') \qquad \Sigma' = \Sigma[dst \mapsto \sigma']}{(\{(src,\ dst,\ m)\} \uplus P,\ \Sigma,\ T) \rightsquigarrow (P \uplus P',\ \Sigma',\ T +\!\!+ \langle o \rangle)}\ \textsc{Deliver}$$

$$\frac{p \in P}{(P,\ \Sigma,\ T) \rightsquigarrow (P \uplus \{p\},\ \Sigma,\ T)}\ \textsc{Duplicate}$$

$$\frac{}{(\{p\} \uplus P,\ \Sigma,\ T) \rightsquigarrow (P,\ \Sigma,\ T)}\ \textsc{Drop}$$

$$\frac{H_{\text{tmt}}(n,\ \Sigma[n]) = (\sigma',\ o,\ P') \qquad \Sigma' = \Sigma[n \mapsto \sigma']}{(P,\ \Sigma,\ T) \rightsquigarrow (P \uplus P',\ \Sigma',\ T +\!\!+ \langle \text{tmt},\ o \rangle)}\ \textsc{Timeout}$$

Note that all steps are atomic in semantics!

Shim must carefully persist to ensure fidelity.

Network semantics shim is delicate

Node in singleton cluster never becomes leader #40

Open **tschottdorf** opened this issue 24 days ago · 5 comments

tschottdorf commented 24 days ago

I'm trying to run the benchmarks against a single-node system:

```
$ ./vard.native -dbpath /tmp/vard-8000 -port 8000 -me 0 -node 0,127.0.0.1:9000 -debug
unordered shim running setup for VarD
unordered shim ready for action
client 115512982 connected on 127.0.0.1:49446
client 115512982 disconnected: client closed socket
```

The client logged above is the following invocation:

```
python2 bench/setup.py --service vard --keys 50 --cluster 127.0.0.1:8000
Traceback (most recent call last):
  File "bench/setup.py", line 34, in <module>
    main()
  File "bench/setup.py", line 27, in main
    host, port = Client.find_leader(args.cluster)
  File "/Users/tschottdorf/tla/verdi-raft/extraction/vard
    raise cls.NoLeader
vard.NoLeader
```

I haven't dug deeper but I did verify that I can run the benchmarks a
running on the same machine). So, perhaps I'm silly or there is a p
system.

palmskog commented 23 days ago — Member

I'm pretty sure this is a liveness bug (and thus an issue outside the scope of election safety, which is guaranteed). What happens is that the singleton node never manages to elect itself leader - it waits forever for a `RequestVoteReply` message.

The `tryToBecomeLeader` function in `raft/Raft.v` is called when a timeout occurs. However, `tryToBecomeLeader` does not immediately check whether the candidate wins the vote. This is only done once a `RequestVoteReply` message is received, using a call to `wonElection`.

The original Go implementation of Raft uses a general loop for the Candidate state that first sends all necessary `RequestVote` messages and then immediately checks whether it has enough votes (and becomes leader if possible). The bug could be fixed by adding a similar check to `tryToBecomeLeader`, but I'm not sure how much that would mess with the proofs. Arguably, there is no point in running Raft in a singleton node cluster anyway - it's enough to run a system that directly uses the underlying state machine (`VarD`).

Network semantics shim is delicate

Node in singleton cluster never becomes leader #40

Open **tschottdorf** opened this issue 24 days ago · 5 comments

tschottdorf commented 24 days ago

I'm trying to run the benchmarks against a single-node system:

```
$ ./vard.native -dbpath /tmp/vard-8000 -port 8000 -me 0 -node 0,127.0.0.1:90
unordered shim running setup for VarD
unordered shim ready for action
client 115512982 connected on 127.0.0.1:49446
client 115512982 disconnected: client closed socket
```

The client logged above is the following invocation:

```
python2 bench/setup.py --service vard --keys 50 --cluster 127.0.0.1:8000
Traceback (most recent call last):
  File "bench/setup.py", line 34, in <module>
    main()
  File "bench/setup.py", line 27, in main
    host, port = Client.find_leader(args.cluster)
  File "/Users/tschottdorf/tla/verdi-raft/extraction/vard
    raise cls.NoLeader
vard.NoLeader
```

I haven't dug deeper but I did verify that I can run the benchmarks a
running on the same machine). So, perhaps I'm silly or there is a p
system.

User stumbled across liveness bug for single node cluster.

palmskog commented 23 days ago — Member

I'm pretty sure this is a liveness bug (and thus an issue outside the scope of election safety, which is guaranteed). What happens is that the singleton node never manages to elect itself leader - it waits forever for a `RequestVoteReply` message.

The `tryToBecomeLeader` function in `raft/Raft.v` is called when a timeout occurs. However, `tryToBecomeLeader` does not immediately check whether the candidate wins the vote. This is only done once a `RequestVoteReply` message is received, using a call to `wonElection`.

The original Go implementation of Raft uses a general loop for the Candidate state that first sends all necessary `RequestVote` messages and then immediately checks whether it has enough votes (and becomes leader if possible). The bug could be fixed by adding a similar check to `tryToBecomeLeader`, but I'm not sure how much that would mess with the proofs. Arguably, there is no point in running Raft in a singleton node cluster anyway - it's enough to run a system that directly uses the underlying state machine (`VarD`).

Network semantics shim is delicate

An Empirical Study on the Correctness of Formally Verified Distributed Systems

Pedro Fonseca Kaiyuan Zhang Xi Wang Arvind Krishnamurthy
University of Washington
{pfonseca, kaiyuanz, xi, arvind}@cs.washington.edu

Abstract

Recent advances in formal verification techniques enabled the implementation of distributed systems with machine-checked proofs. While results are encouraging, the importance of distributed systems warrants a large scale evaluation of the results and verification practices.

This paper thoroughly analyzes three state-of-the-art, formally verified implementations of distributed systems: IronFleet, Verdi, and Chapar. Through code review and testing, we found a total of 16 bugs, many of which produce serious consequences, including crashing servers, returning incorrect results to clients, and invalidating verification guarantees. These bugs were caused by violations of a wide-range of assumptions on which the verified components relied. Our results revealed that these assumptions referred to a small fraction of the trusted computing base, mostly at the inter-

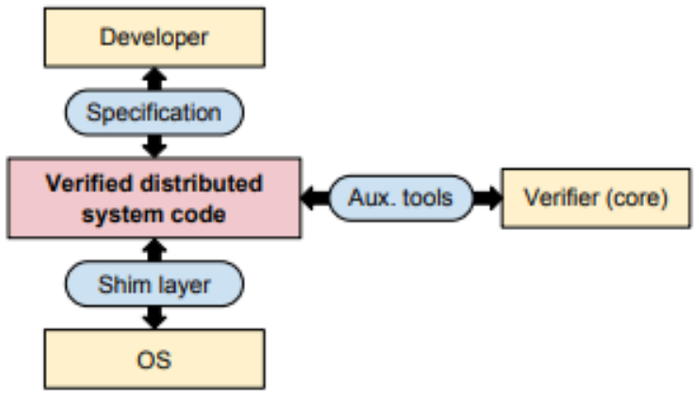


Figure 1: An overview of the workflow to verify a distributed system implementation.

finding tools [26, 37, 53, 54], and formal verification techniques [22, 29, 34, 52].

Network semantics shim is delicate

An Empirical Study on the Correctness of Formally Verified Distributed Systems

Pedro Fonseca Kaiyuan Zhang Xi Wang Arvind Krishnamurthy
University of Washington
{pfonseca, kaiyuanz, xi, arvind}@cs.washington.edu

Abstract

Recent advances in formal verification techniques enabled the implementation of distributed systems with machine-checked proofs. While results are encouraging, the importance of distributed systems warrants a large scale evaluation of the results and verification practices.

This paper thoroughly analyzes three state-of-the-art, formally verified implementations of distributed systems: IronFleet, Verdi, and Chapar. Through code review and testing, we found a total of 16 bugs, many of which produce serious consequences, including crashing servers, returning incorrect results to clients, and invalidating verification guarantees. These bugs were caused by violations of a wide-range of assumptions on which the verified components relied. Our results revealed that these assumptions referred to a small fraction of the trusted computing base, mostly at the inter-

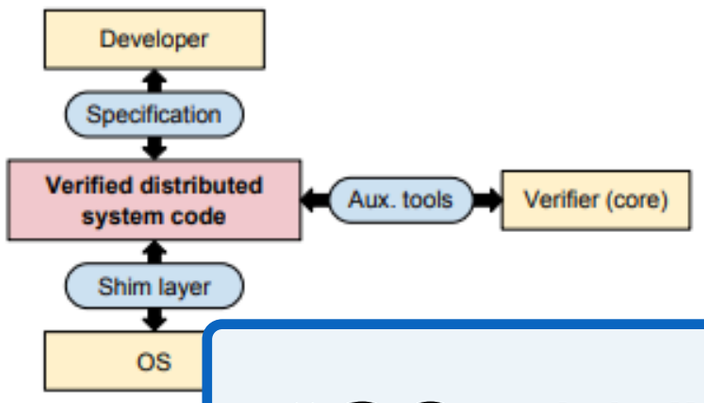
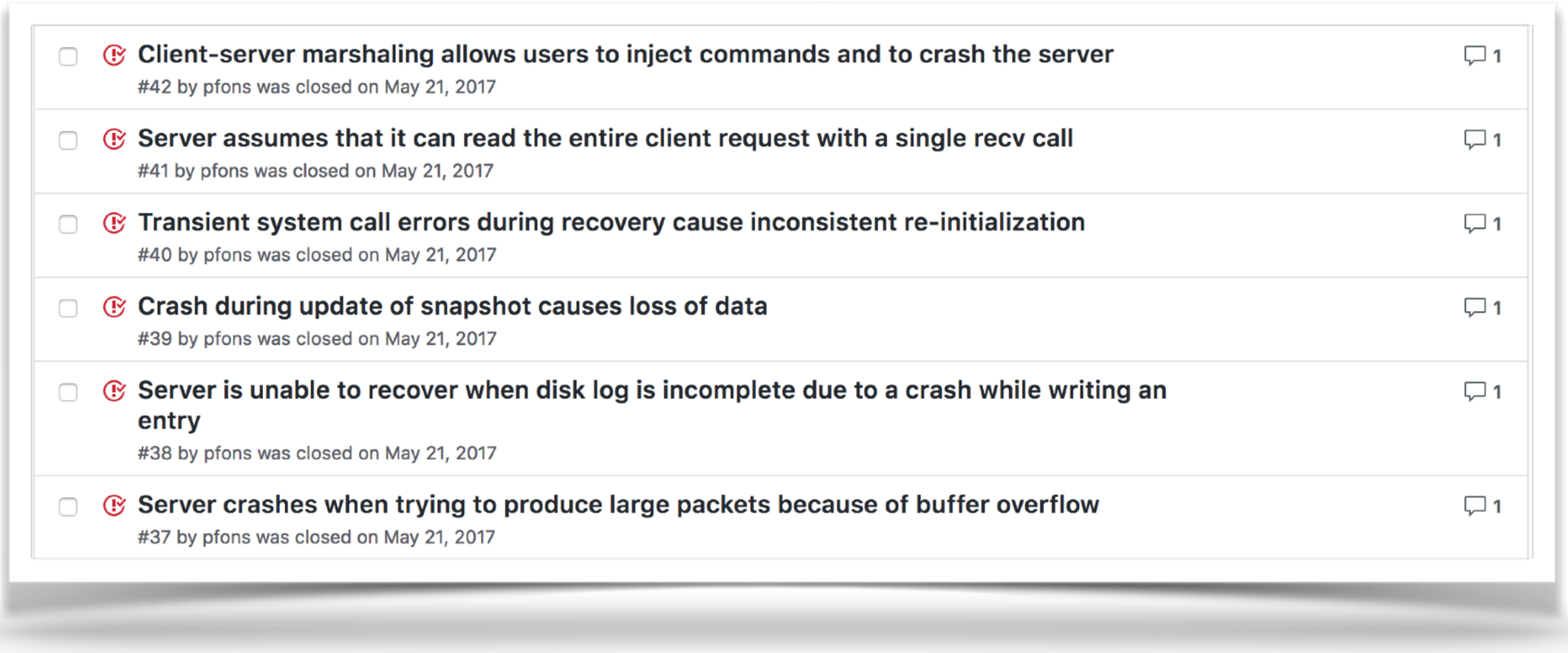


Figure 1: An overv
system implementat

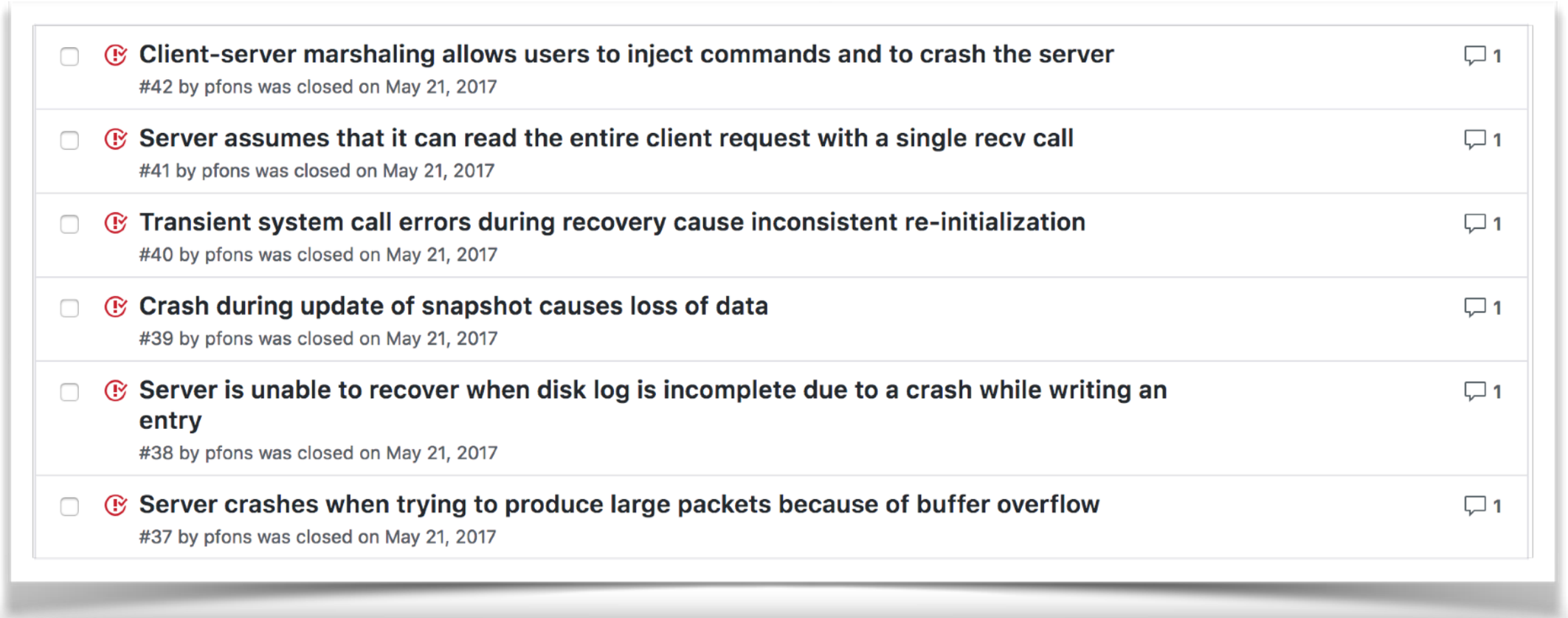
finding tools [26,
niques [22, 29, 34

"CSmith" paper for verified distr sys

Network semantics shim is delicate



Network semantics shim is delicate



Pedro et al. found several bugs, *BUT*

none in any verified components.

Network semantics shim is delicate

Client-server marshaling allows users to inject commands and to crash the server
#42 by pfons was closed on May 21, 2017 — 1

Server assumes that it can read the entire client request with a single recv call
#41 by pfons was closed on May 21, 2017 — 1

Transient system call errors during recovery cause inconsistent re-initialization
#40 by pfons was closed on May 21, 2017 — 1

Crash during update of snapshot causes loss of data — 1

Justin Adsuara

Cheerios:

New system transformer with correct serialization implemented and verified.

Pedro et al. found several bugs, *BUT*

none in any verified components.

Training the next generation

There will always be a TCB

we'll always need informed judgement

Engineers unlikely to pick this up at work

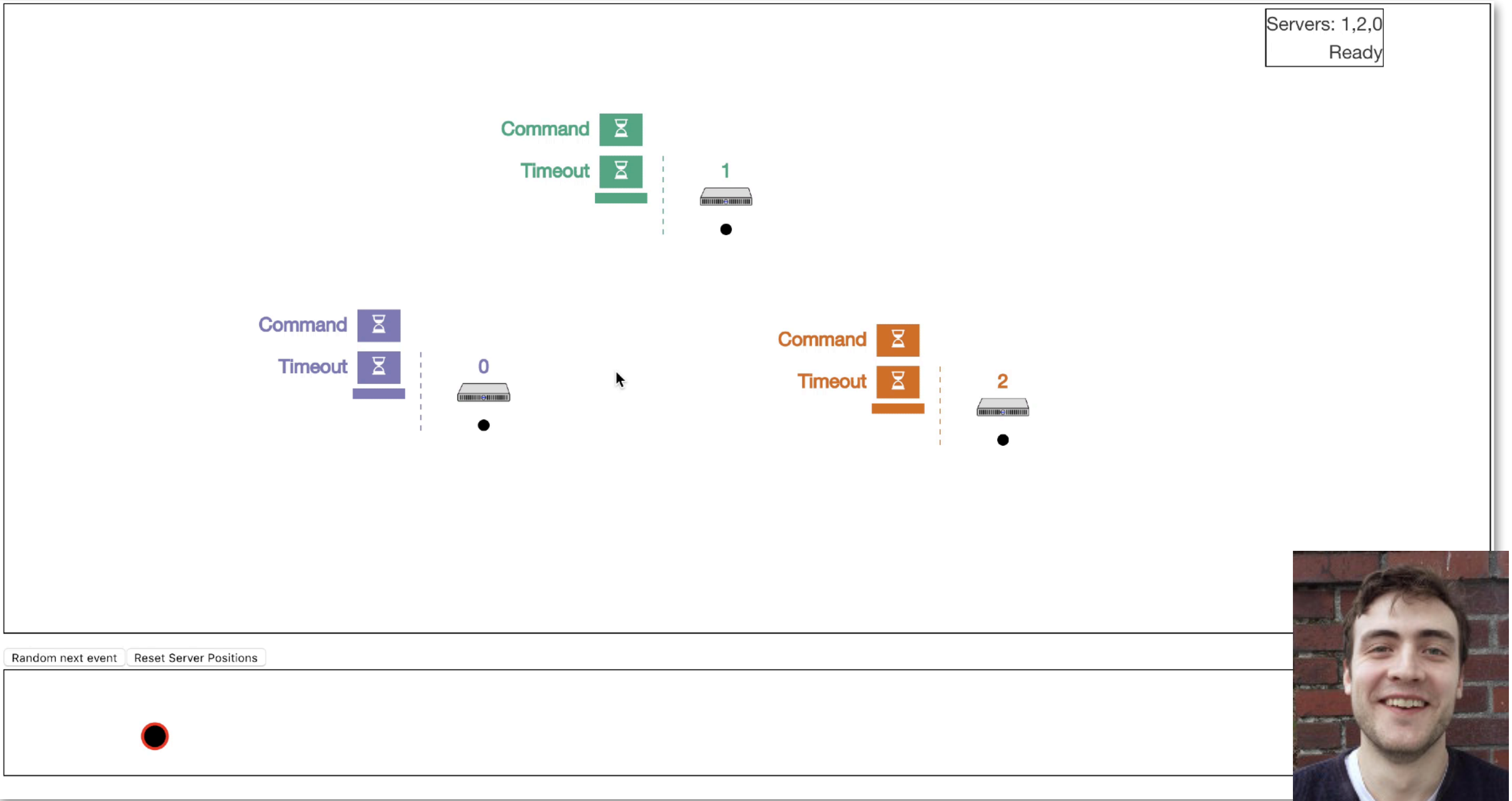
but courses great evangelism opportunity

How to get this into ugrad canon?

need reusable labs and tools

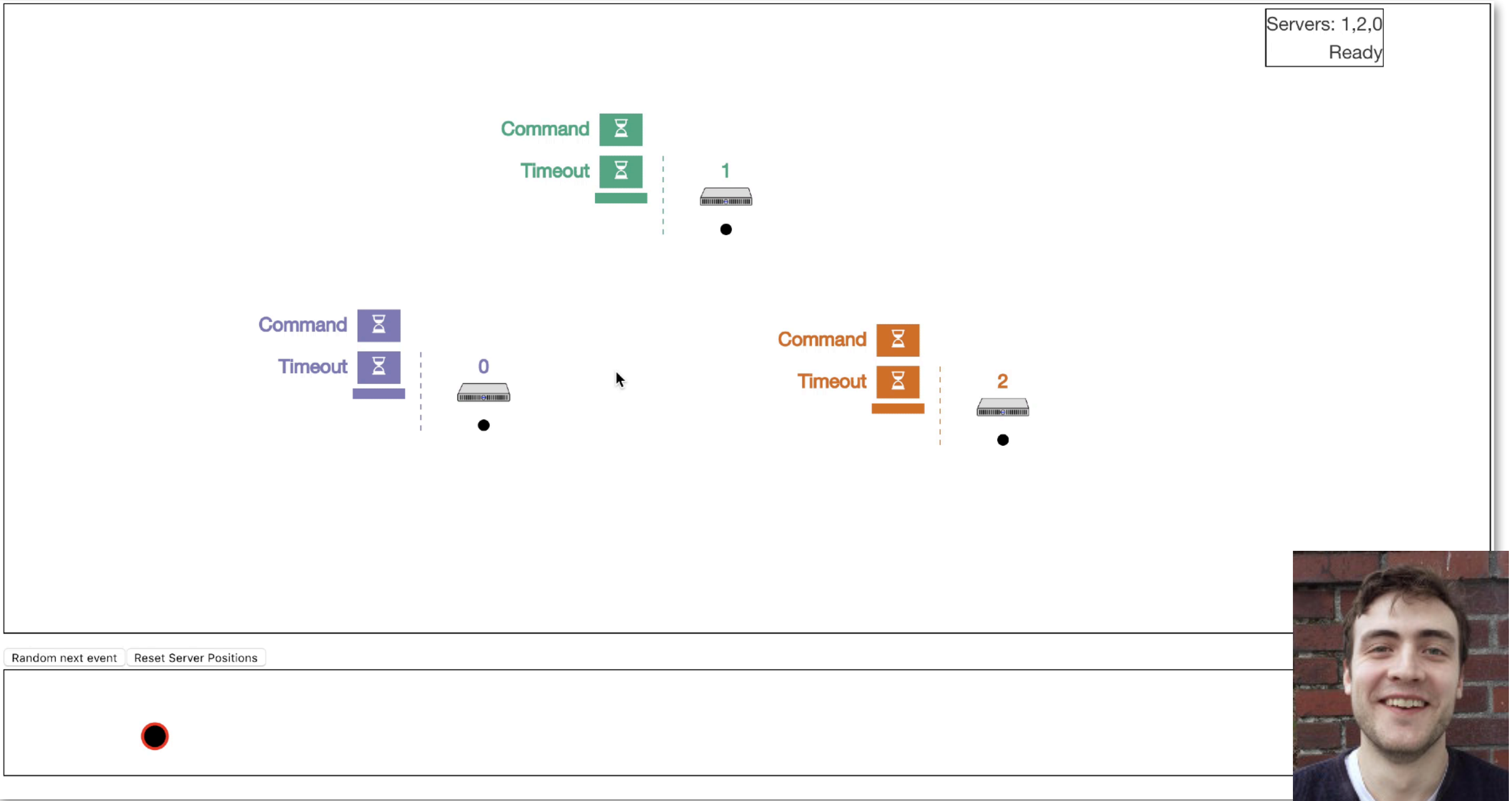
Doug Woos

Training the next generation



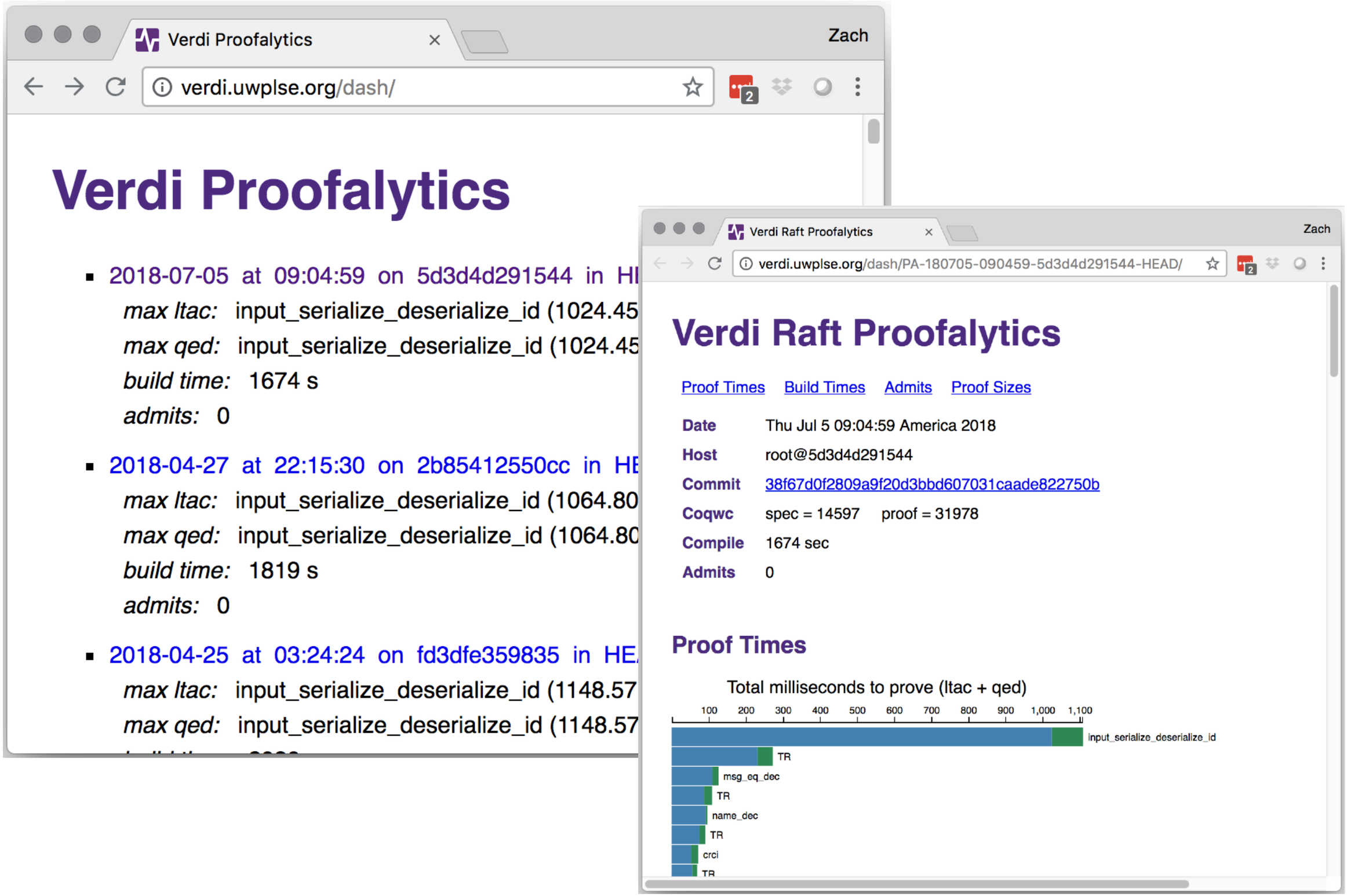
Doug Woos

Training the next generation



Doug Woos

Proof engineering



Proof engineering

PICOQ: Parallel Regression Proving for Large-Scale Verification Projects

Karl Palmskog, University of Texas at Austin, USA, palmskog@acm.org
Ahmet Celik, University of Texas at Austin, USA, ahmetcelik@utexas.edu
Milos Gligoric, University of Texas at Austin, USA, gligoric@utexas.edu

A Regression Proof Selection Tool For Coq

Ahmet Celik, Karl Palmskog, and Milos Gligoric
The University of Texas at Austin, University of Illinois at Urbana-Champaign, The University of Texas at Austin
ahmetcelik@utexas.edu,palmskog@illinois.edu,gligoric@utexas.edu

ICOQ: Regression Proof Selection for Large-Scale Verification Projects

Ahmet Celik, University of Texas at Austin, Austin, TX-78712, USA, ahmetcelik@utexas.edu
Karl Palmskog, University of Illinois at Urbana-Champaign, Urbana, IL-61801, USA, palmskog@illinois.edu
Milos Gligoric, University of Texas at Austin, Austin, TX-78712, USA, gligoric@utexas.edu

Finally catching the interest of the SE community: ASE ’17, ICSE ’18, ISSTA ‘18

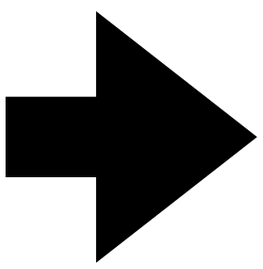
Karl Palmskog

Toward verified distributed systems

The Verdi Framework

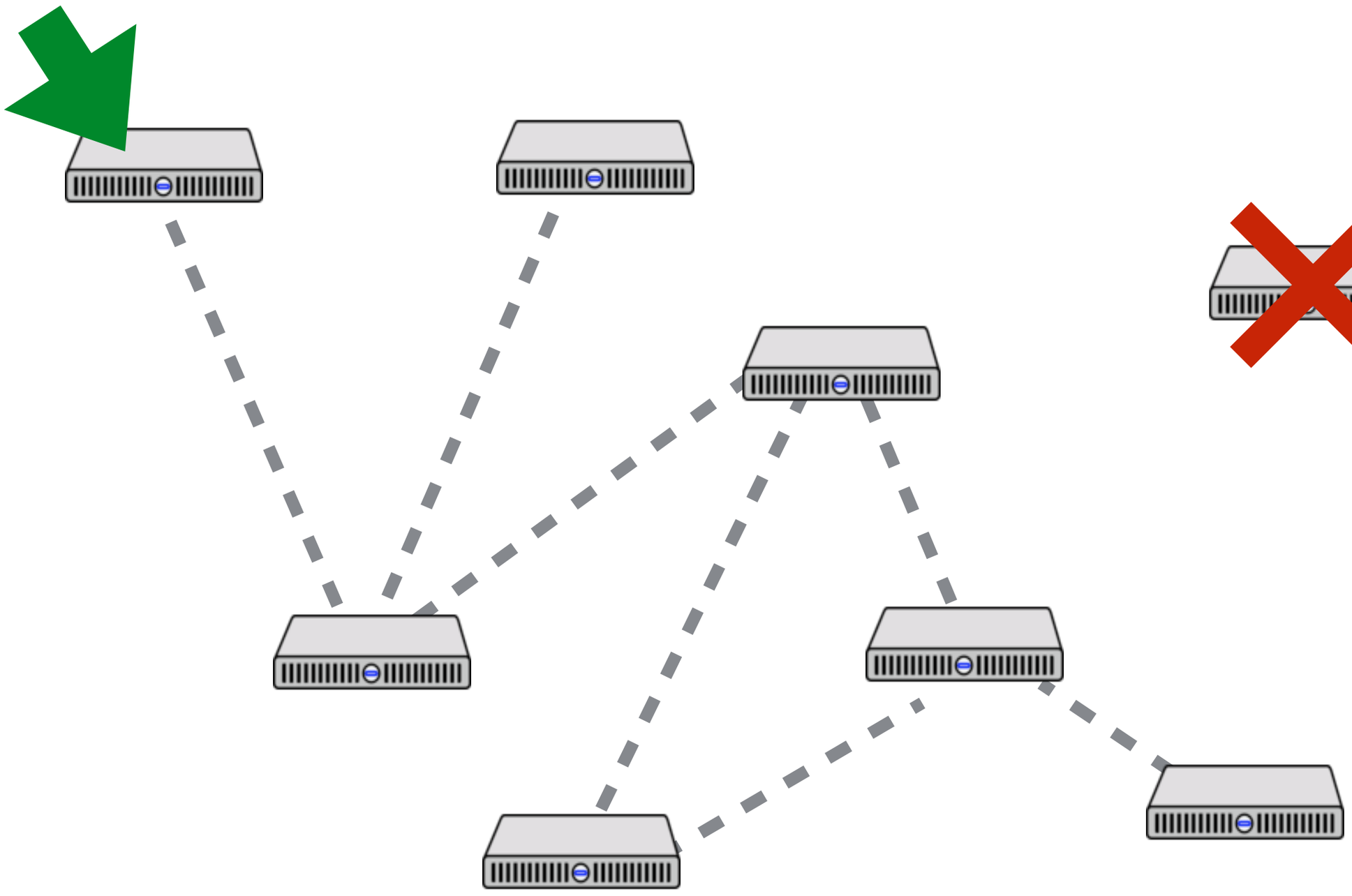
Verified Raft Consensus

TCB, Tools, Teaching

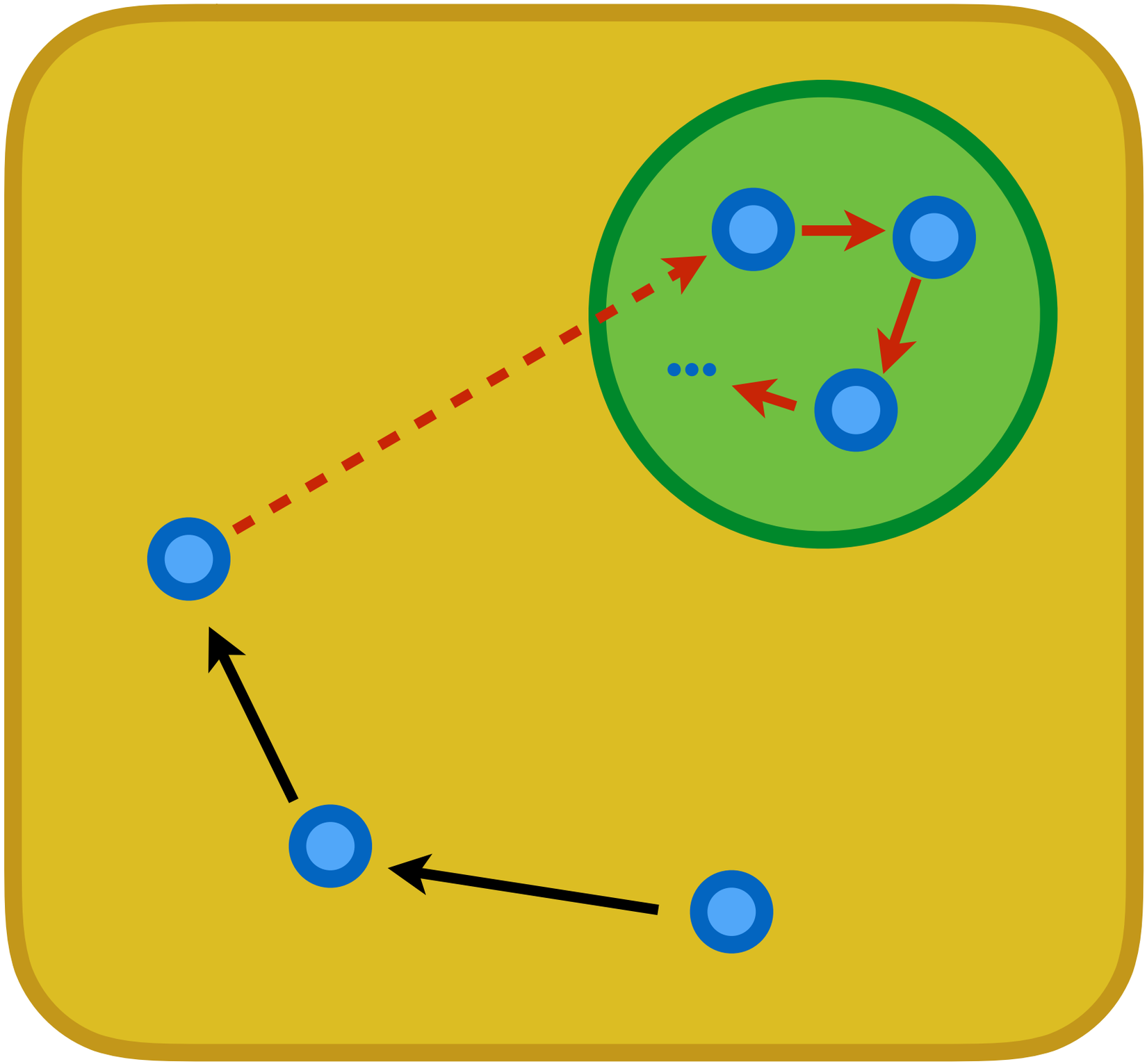


Enriching Models & Modularity

Churn = nodes joining & leaving a system at run time



Punctuated safety properties



Reachable

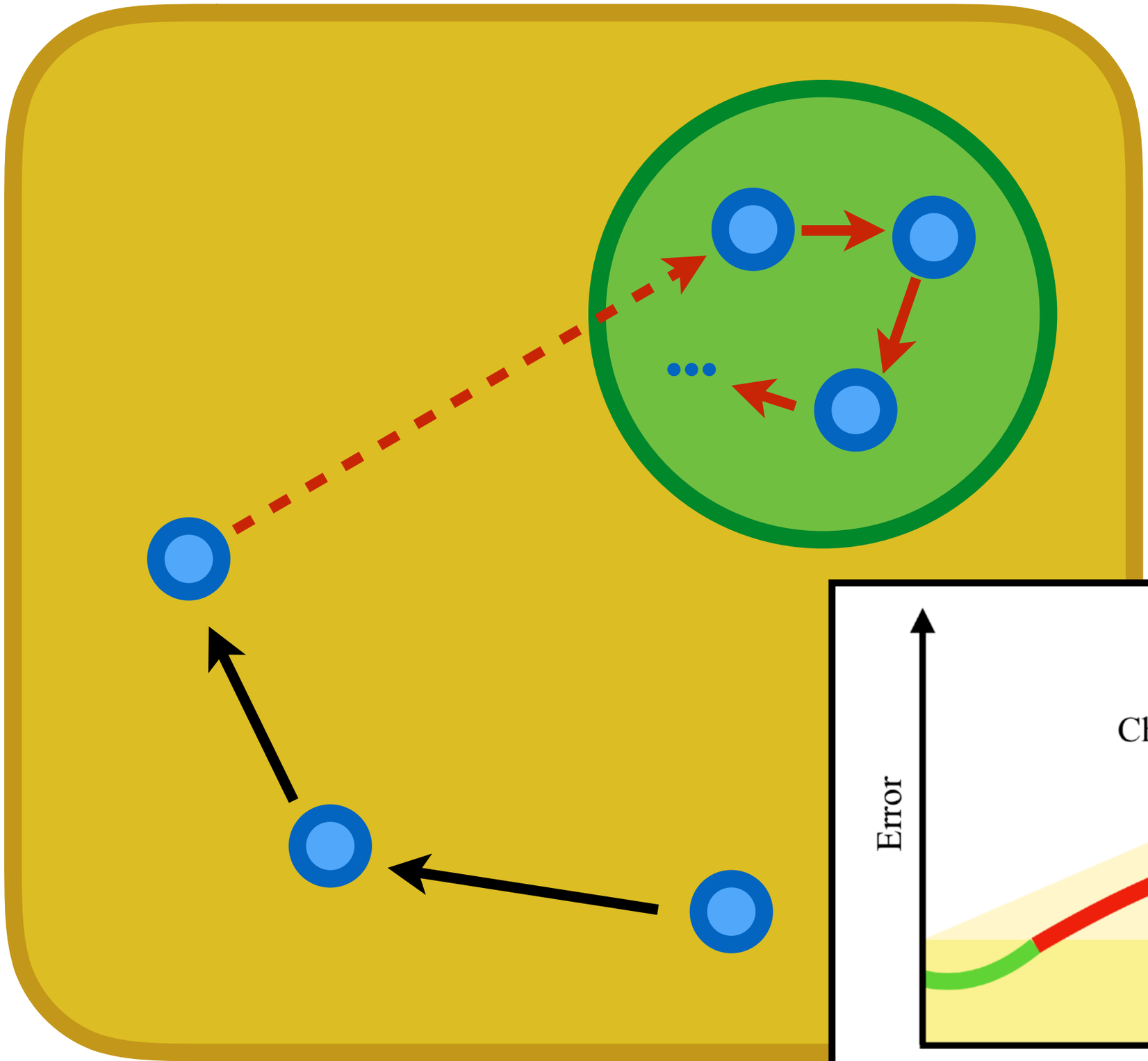
under churn (→)

Safety

after churn stops(→)

Ryan Doenges

Punctuated safety properties

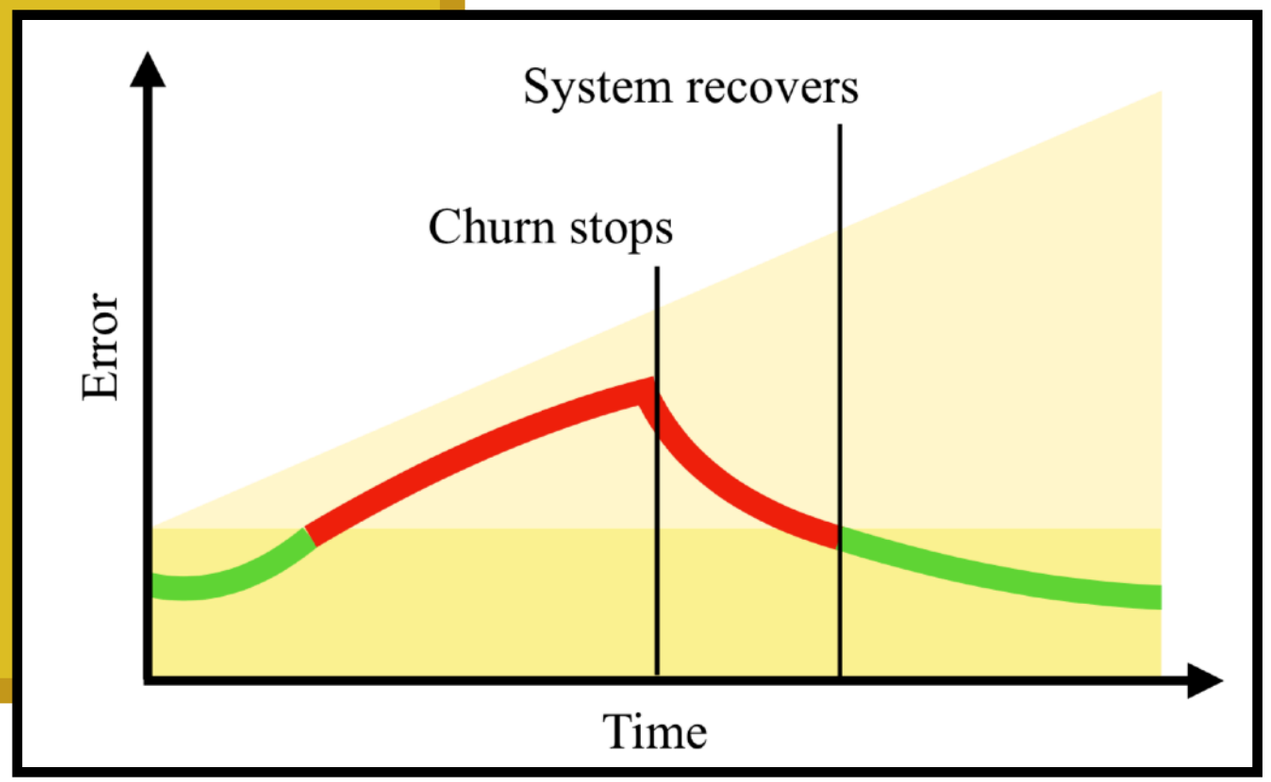


Reachable

under churn (→)

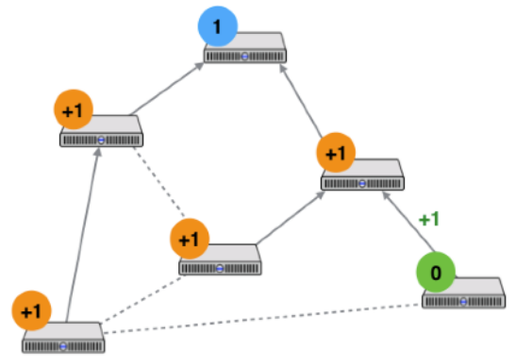
Safety

after churn stops(→)



Toward verifying churn tolerance

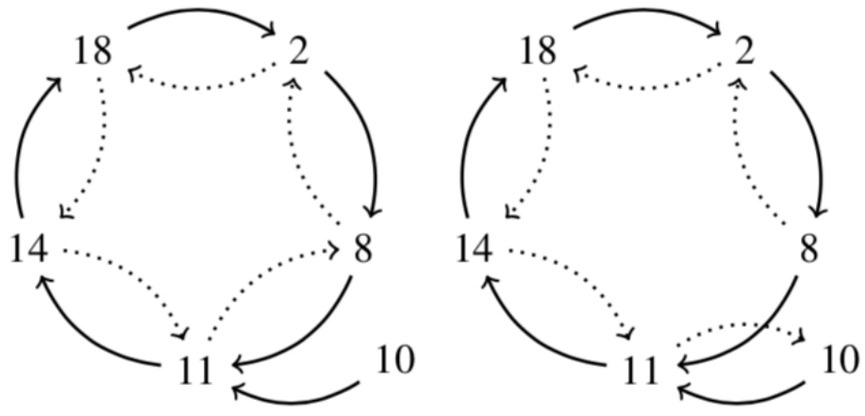
Tree aggregation



aggregate data in sensor networks

designated root node eventually correct

Chord

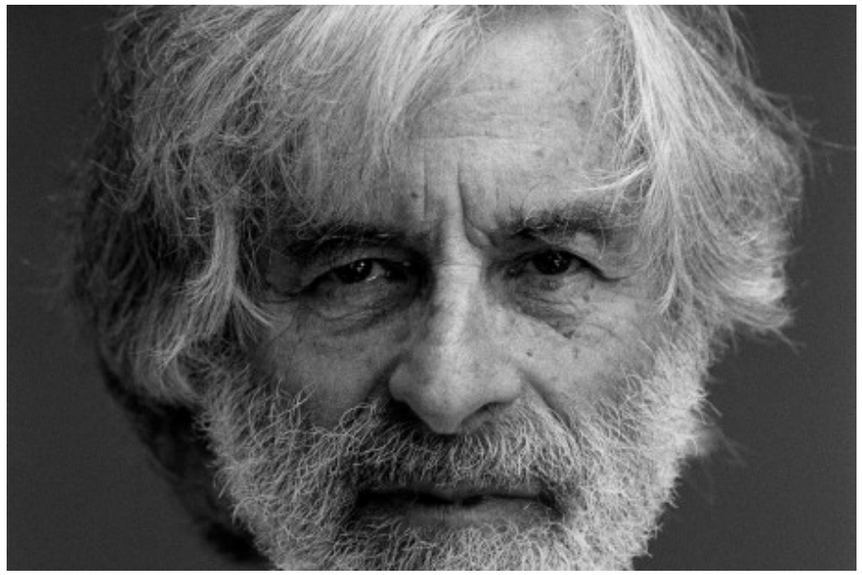


distributed hash table

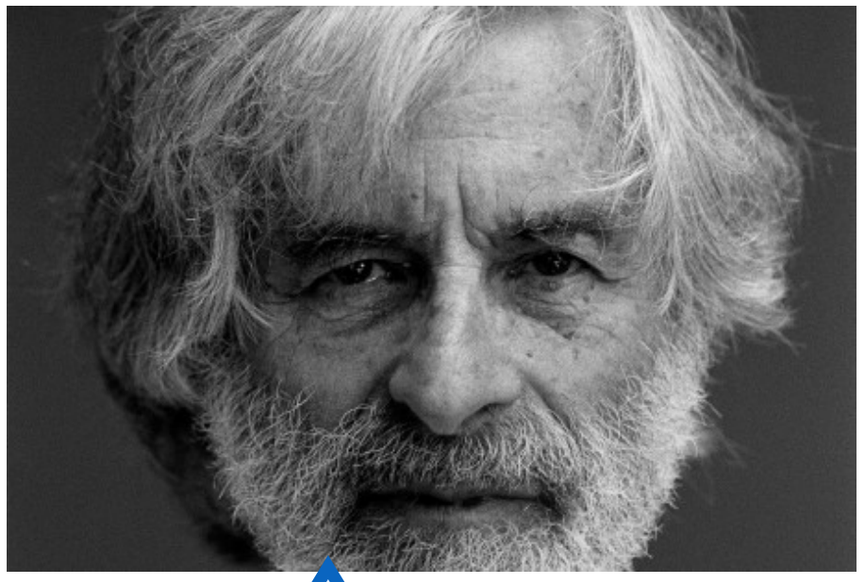
protocol bugs found [Zave 2015]

ring should eventually stabilize

Composition: A way to make proofs harder

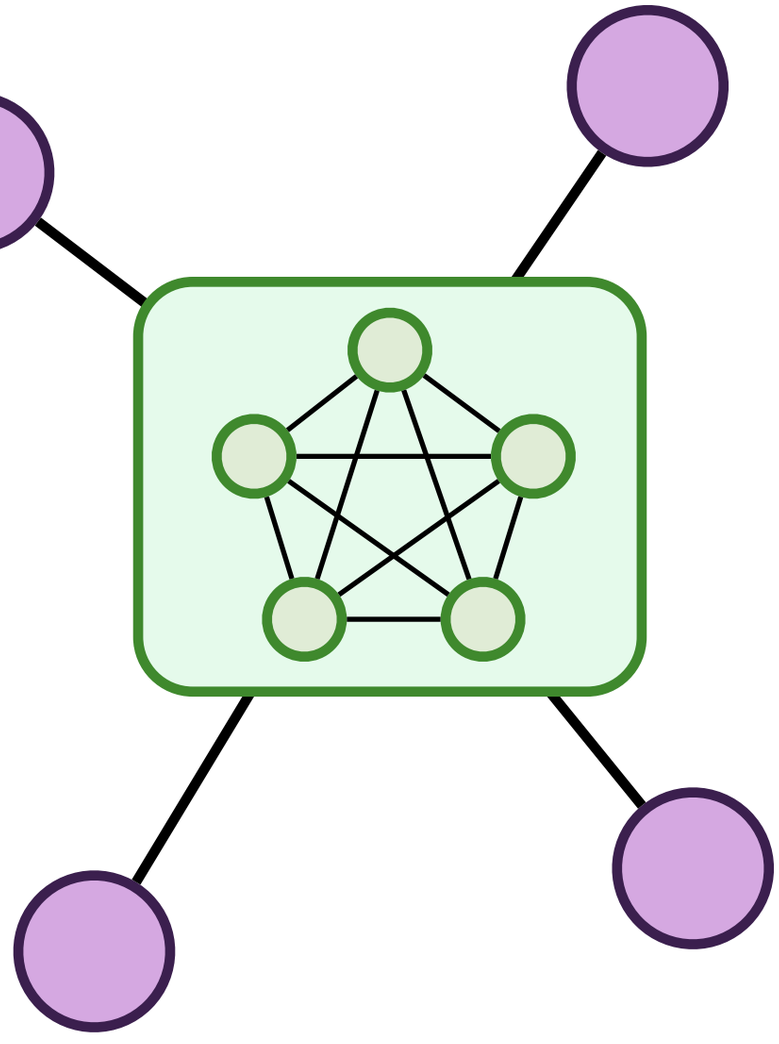


Composition: A way to make proofs harder

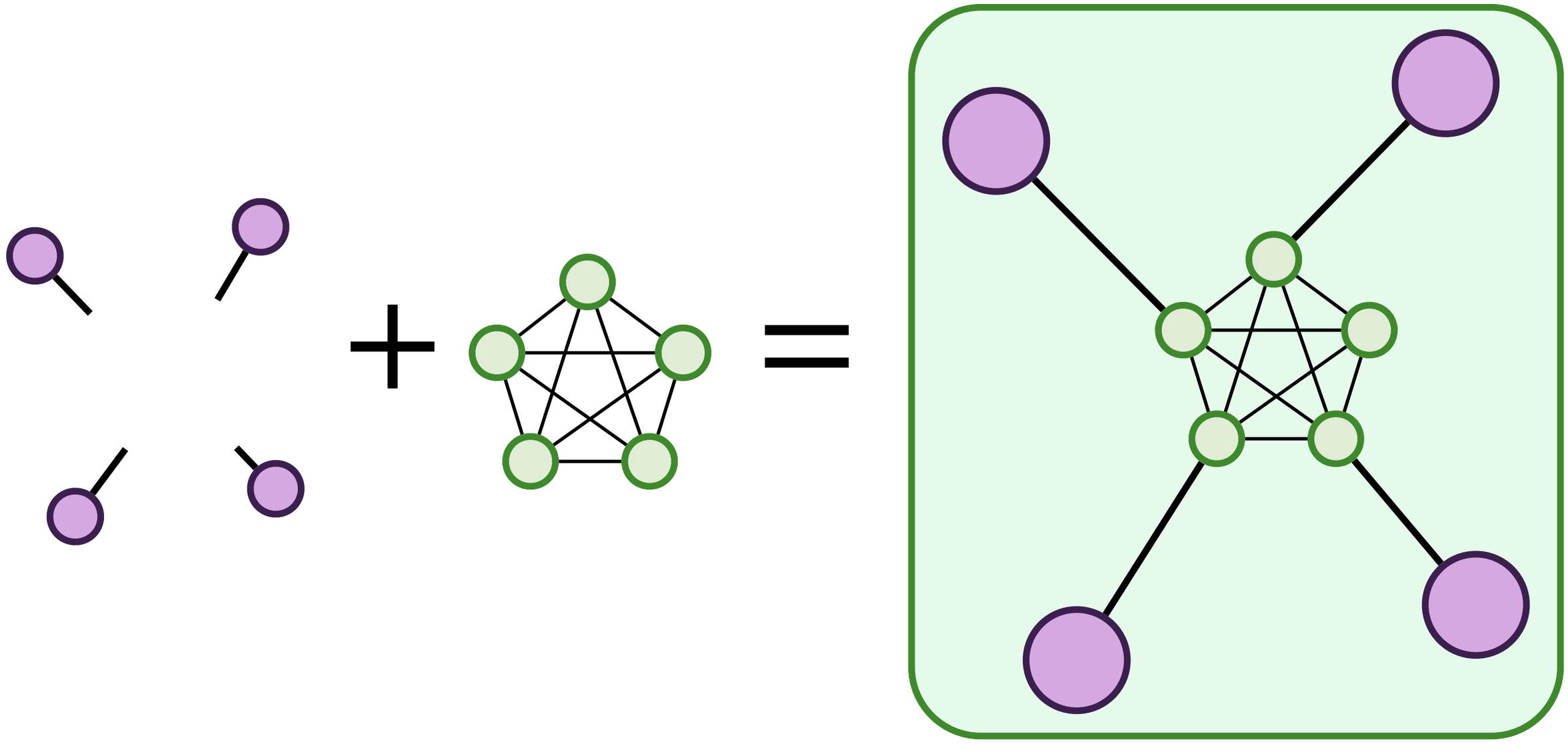


"In 1997, the unfortunate reality is that engineers rarely specify and reason formally about the systems they build. It seems unlikely that reasoning about the composition of open-system specifications will be a practical concern within the next 15 years."

“Horizontal composition”: eliminate closed world hypothesis



“Horizontal composition”: eliminate closed world hypothesis



Compositional Verif of Distr Sys

Challenges

Client reasoning

Invariants

Separation

Solutions

Protocols

WithInv rule

Frame rule/Hooks

James Wilcox

Disel:  $\vdash \{P\}\ c\ \{Q\}$

[POPL 18]

Ilya Sergey

Toward verified distributed systems

The Verdi Framework

Verified Raft Consensus

TCB, Tools, Teaching

Enriching Models & Modularity

Reflections on Verdi experience

Distributed sys good fit for verification

critical, expert-written, I/O bound cases

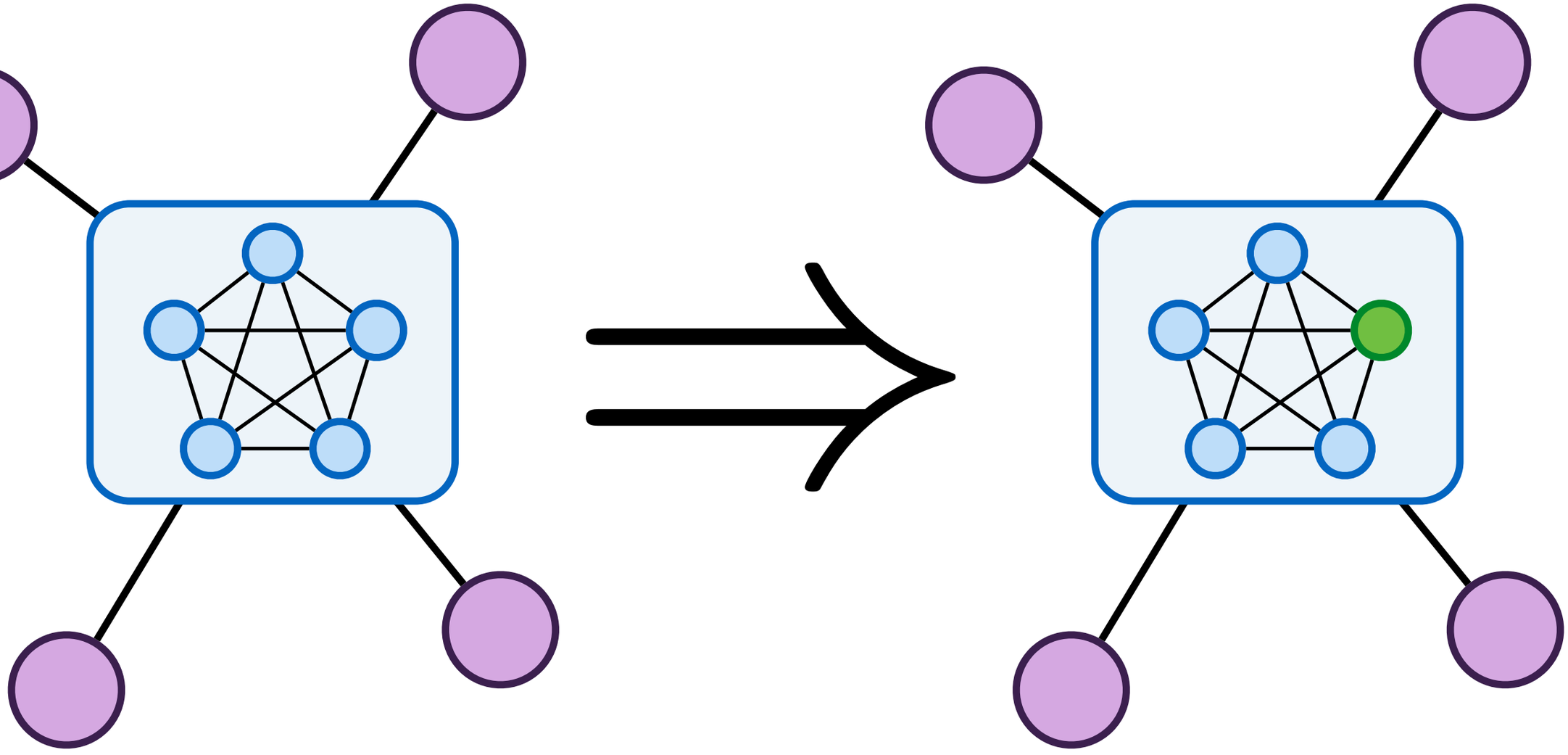
Biggest challenge is proof engineering

reproving and managing scale daunting

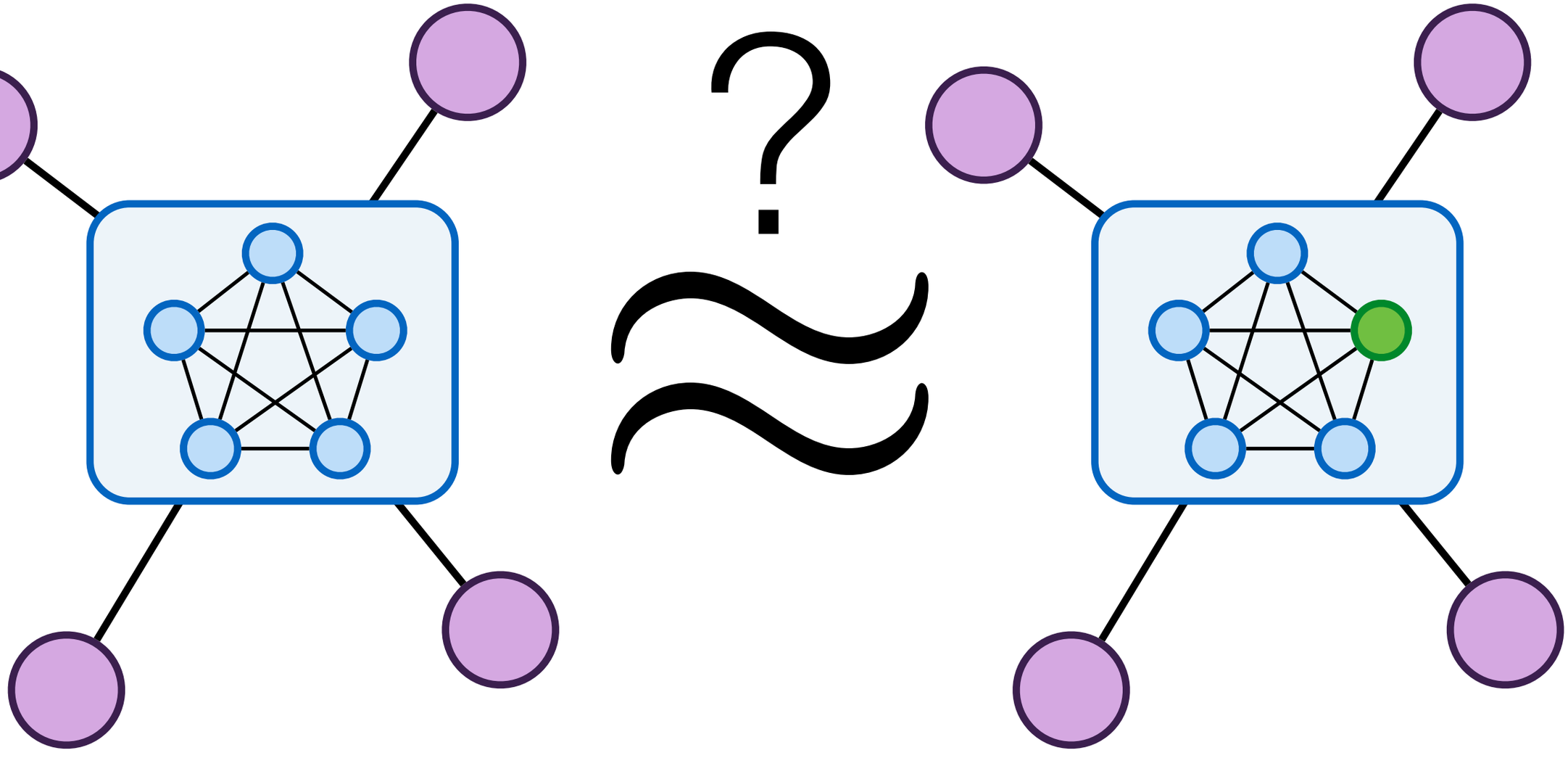
Lots of low-hanging fruit left

dynamic update, concurrency, optimization

Challenge: updating handlers

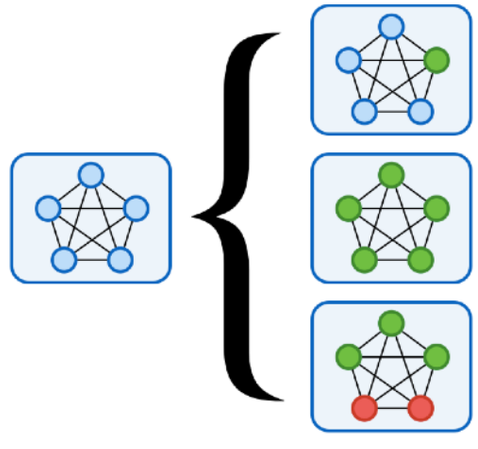


Challenge: updating handlers



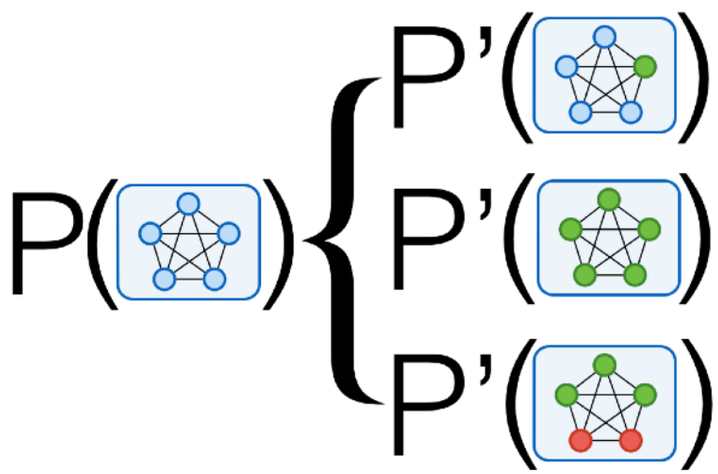
Challenge: updating handlers

1. Specify update correctness
   need a taxonomy of updates



2. Implement update mechanisms (VSTs?)
   stop-the-world, reboot node, dyn handler swap

3. Prove implementation satisfies spec
   preserve old spec, support new invariants, ...



Verdi Team

James Wilcox

Doug Woos

Pavel Panchekha

Ryan Doenges

Justin Adsuara

Keith Simmons

Steve Anton

Miranda Edwards

Karl Palmskog

Ilya Sergey

Xi Wang

Mike Ernst

Tom Anderson

Thank You!

The Verdi Framework

Verified Raft Consensus

TCB, Tools, Teaching

Enriching Models & Modularity

http://distributedcomponents.net