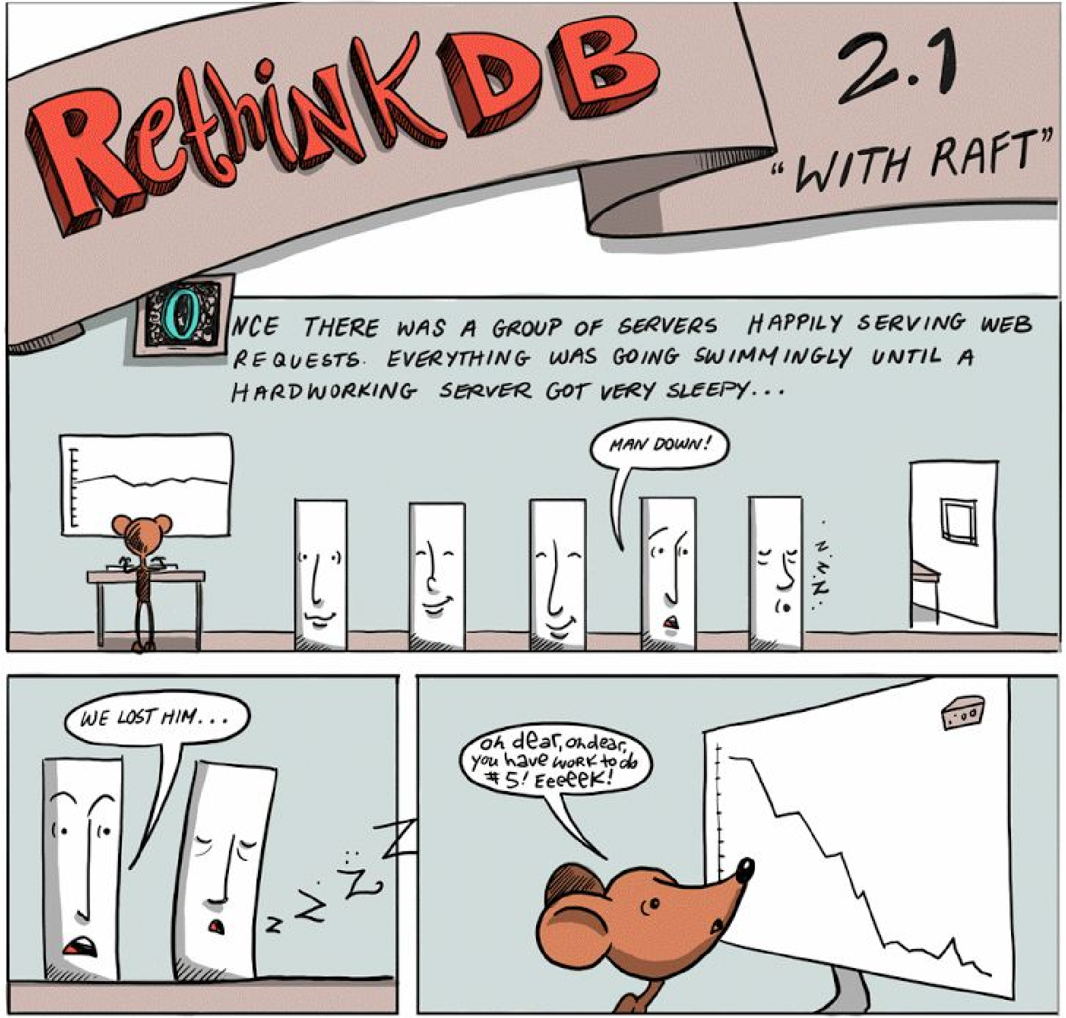
FLP Impossibility & Weakest Failure Detector

Consensus Protocols in Theory

Philip Daian - 10/25

slides influenced by Birman FA12 slides

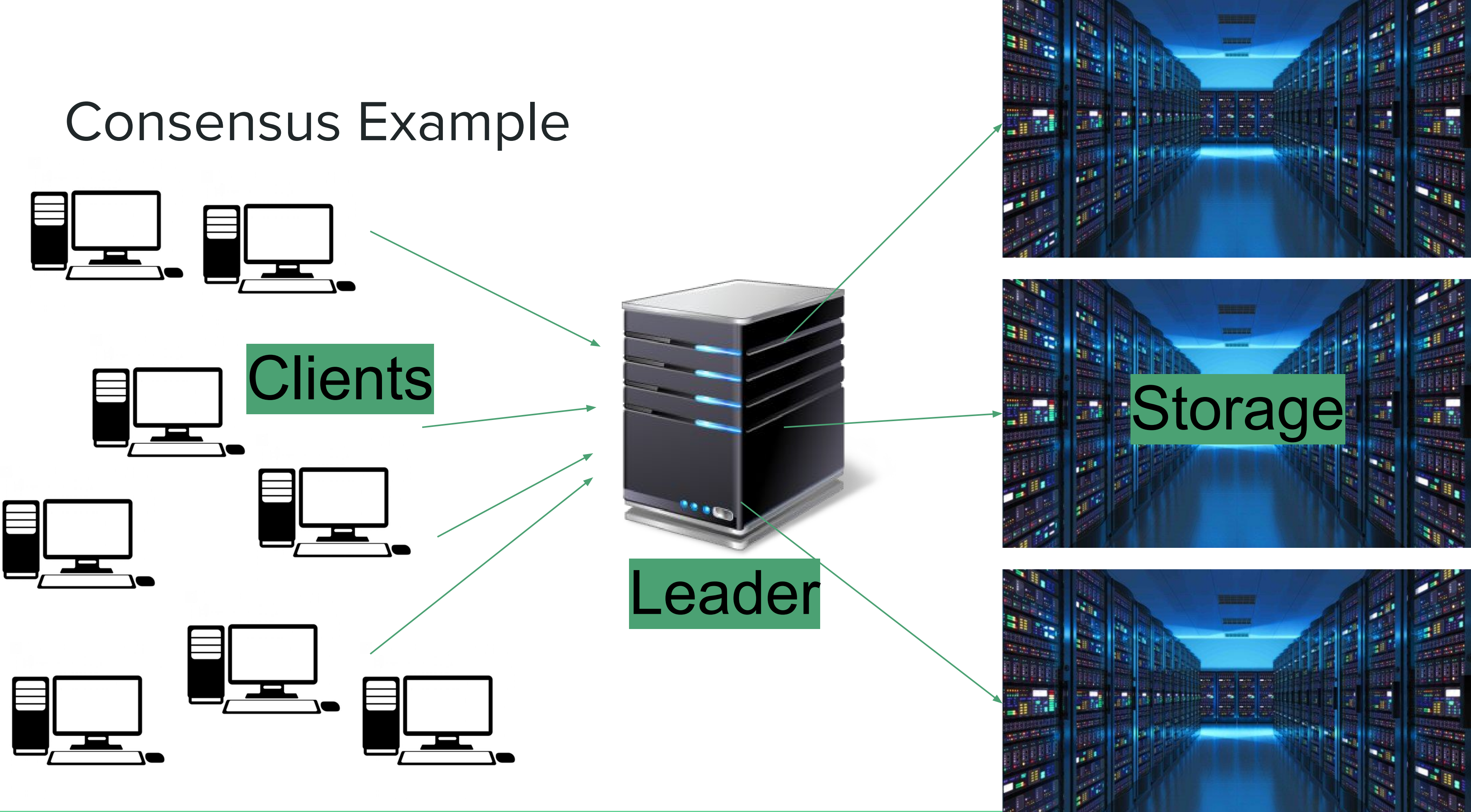
Consensus!



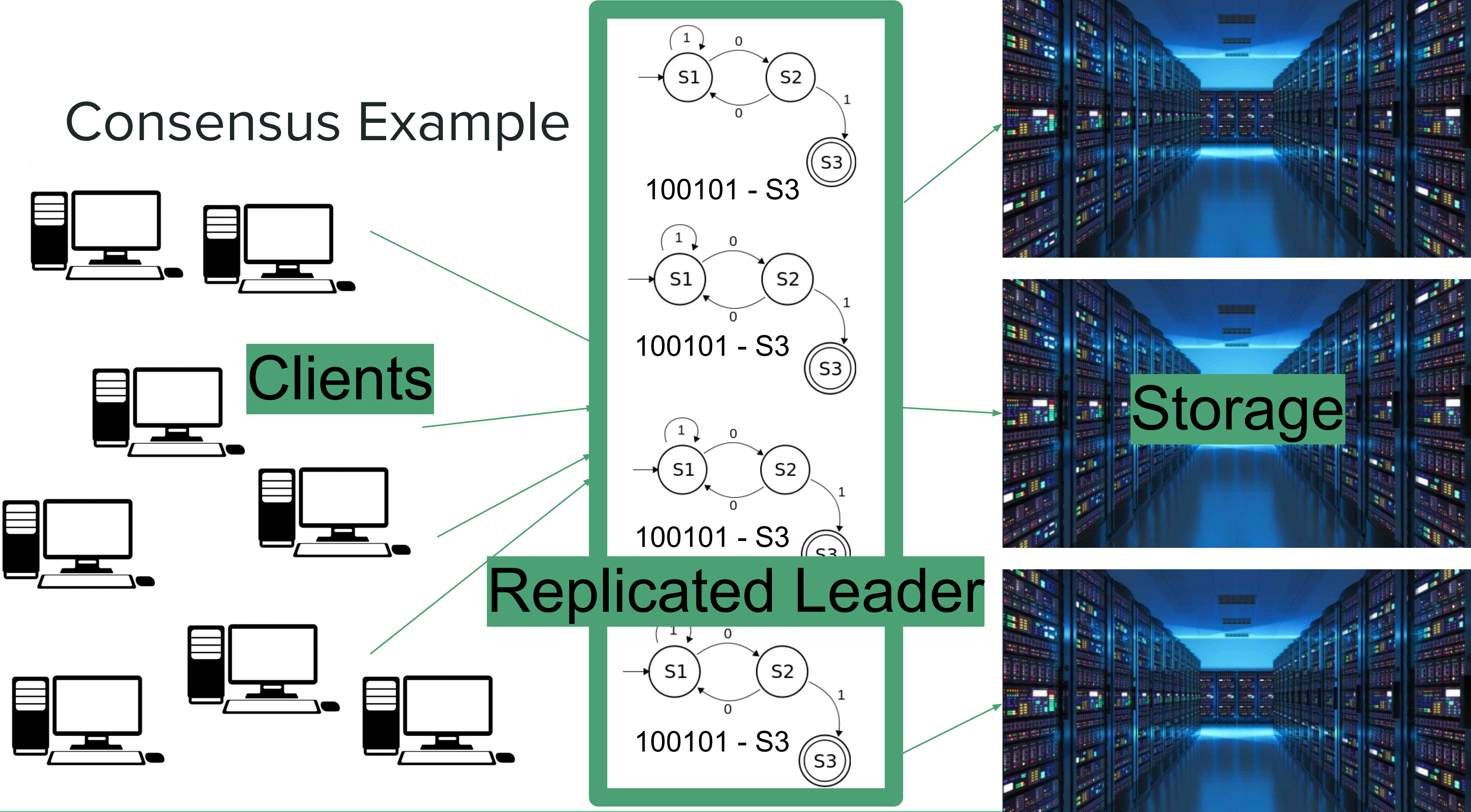
Courtesy of

https://rethinkdb.com

Consensus Example



Consensus Example



Consensus Summary

- Important problem! We've already talked quite a bit about forms of consensus
  - State machine replication -> consensus on state of machine
  - Leader election in leadered protocols -> consensus on leader
  - Paxos, CORFU -> essentially consensus protocols
  - Byzantine Generals -> consensus in malicious actor setting
- Applications: "clock synchronization, PageRank, opinion formation, power smart grids, state estimation, control of UAVs, load balancing and so on" (Wiki)
- Conditions: **Termination**, **Validity, Integrity, Agreement**
  - Conditions vary depending on problem model / definitions
- Focus on consensus on a simple bit for simplicity; such protocols can extend

Impossibility of Distributed Consensus with One Faulty Process 1985

- 2001 Dijkstra prize; best paper in distributed systems

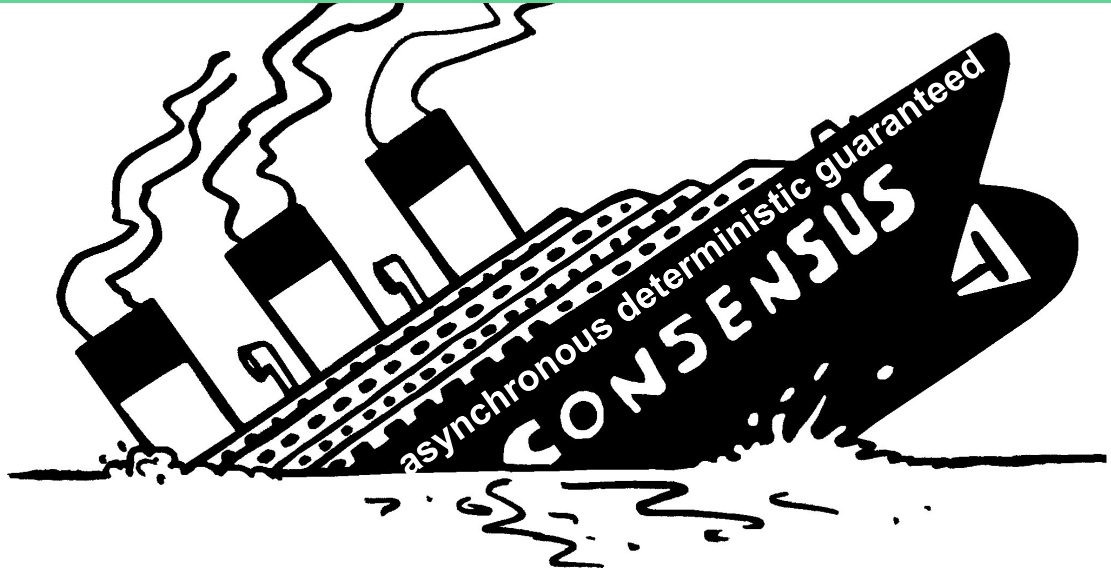
distributed systems,
e-voting,
oblivious transfer

distributed algorithms
and impossibility results,
formal modeling

algorithms, complexity,
theory

FLP : Primary Result

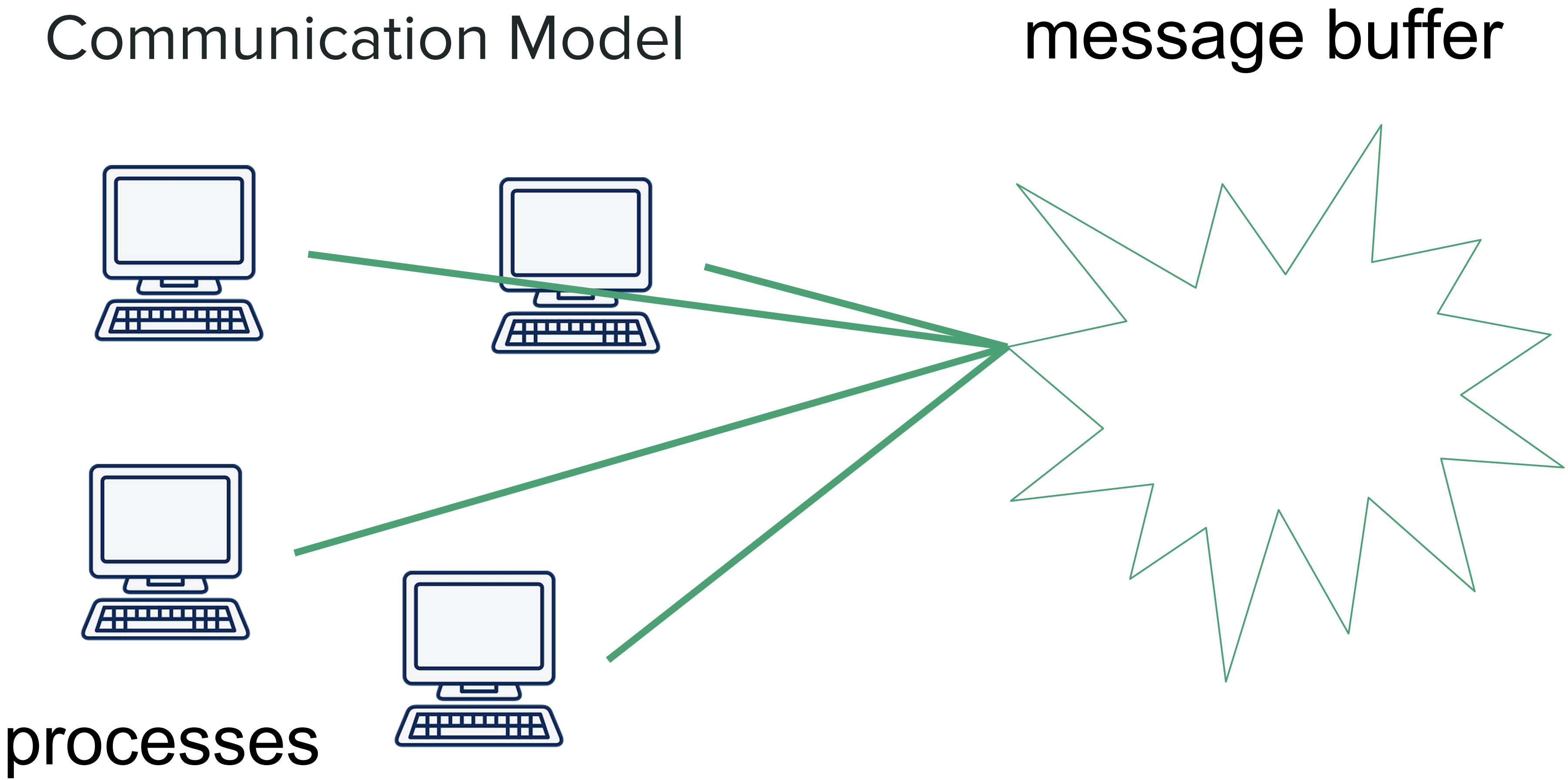
asynchronous deterministic distributed consensus impossible with even 1 crash failure

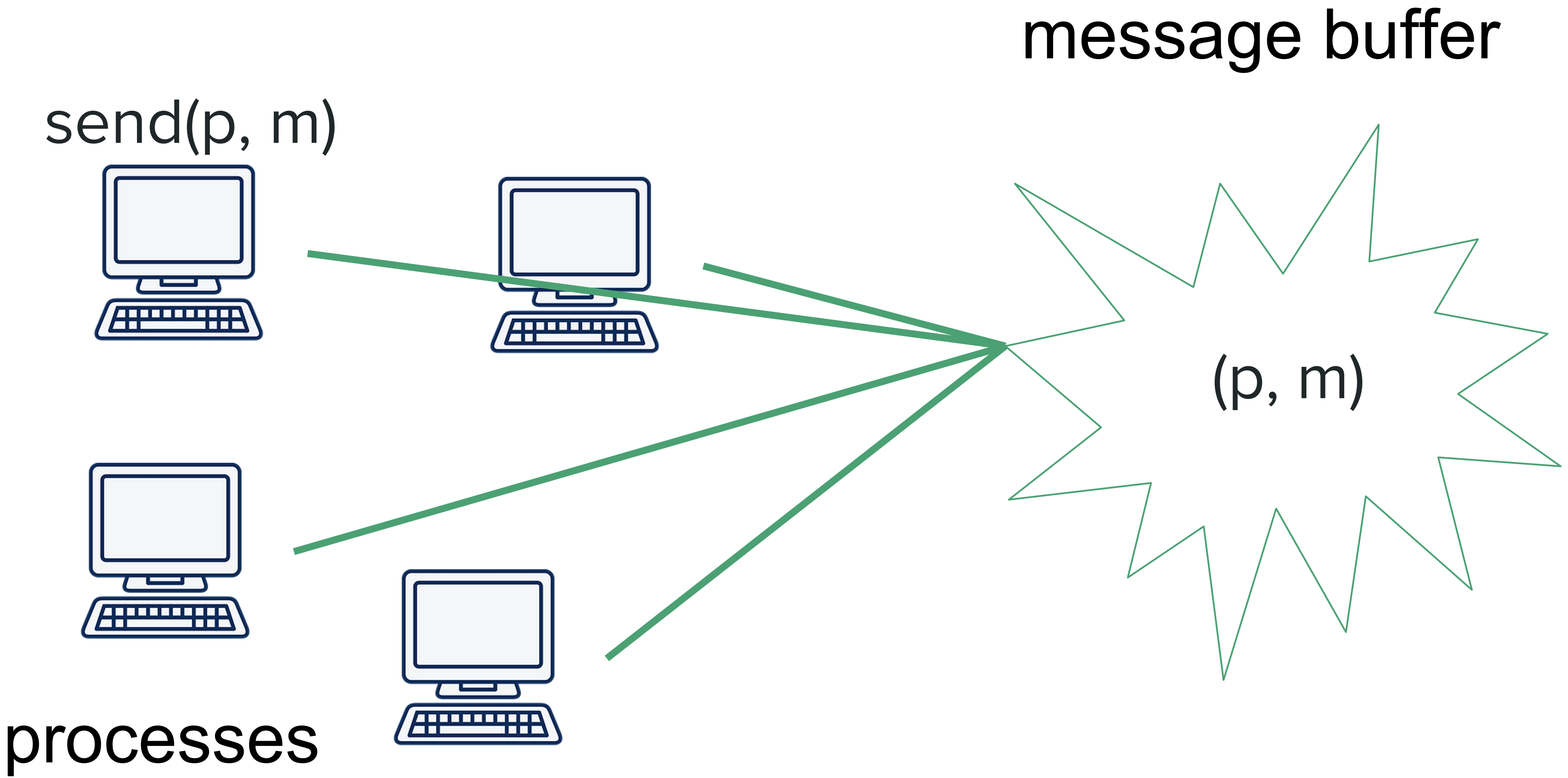


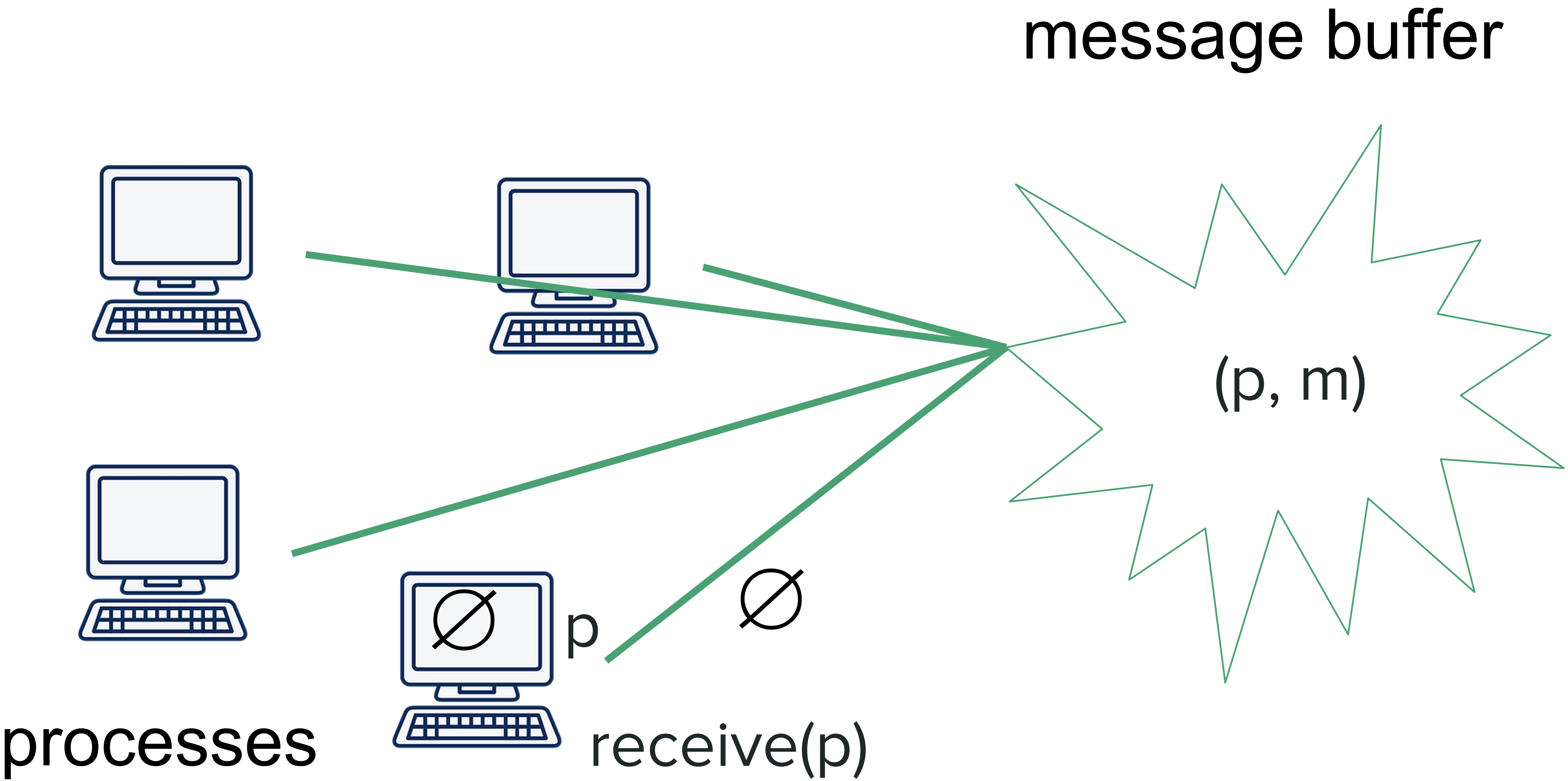
Follow along!

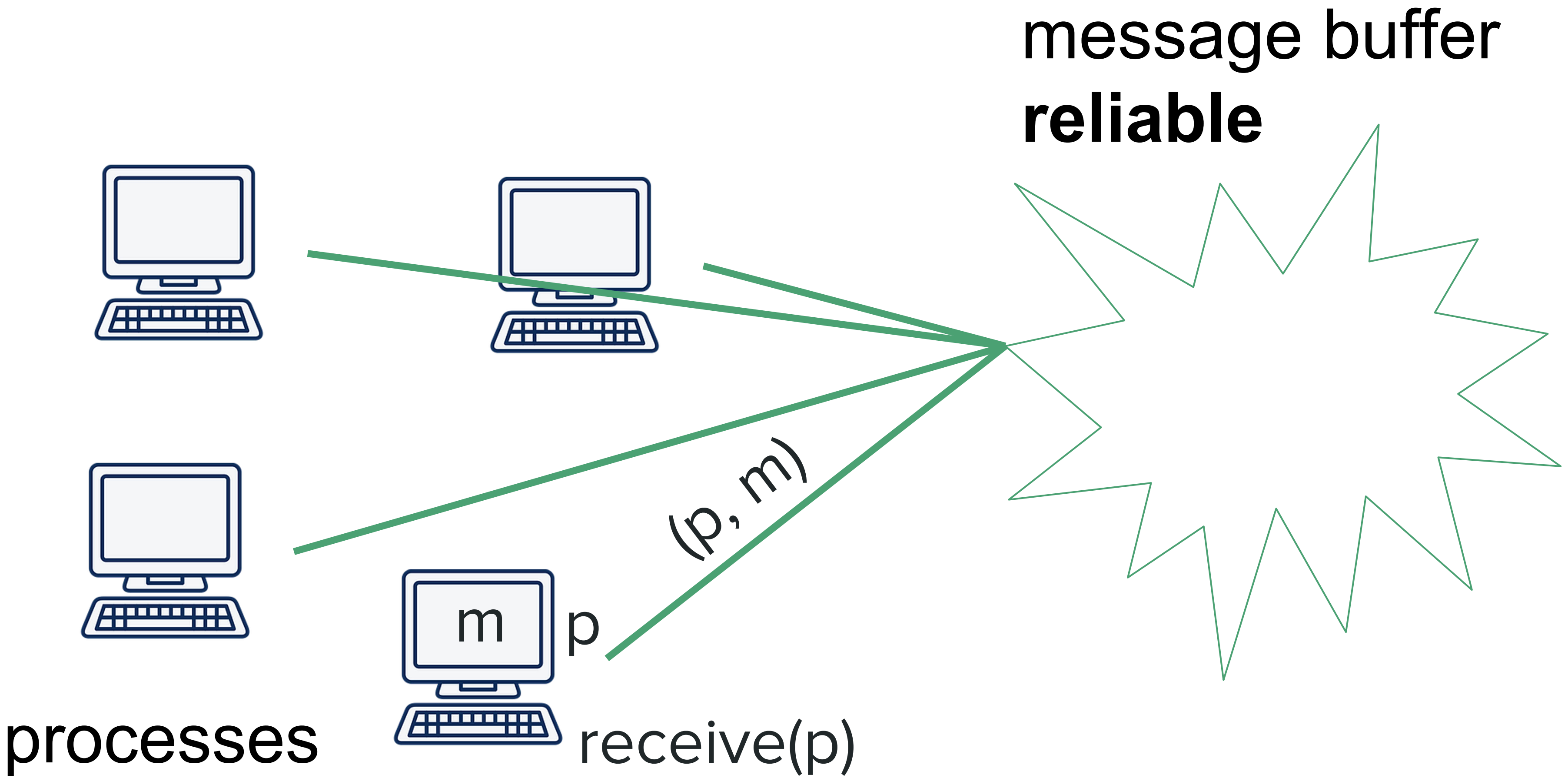
http://the-paper-trail.org/blog/a-brief-tour-of-flp-impossibility/

Communication Model

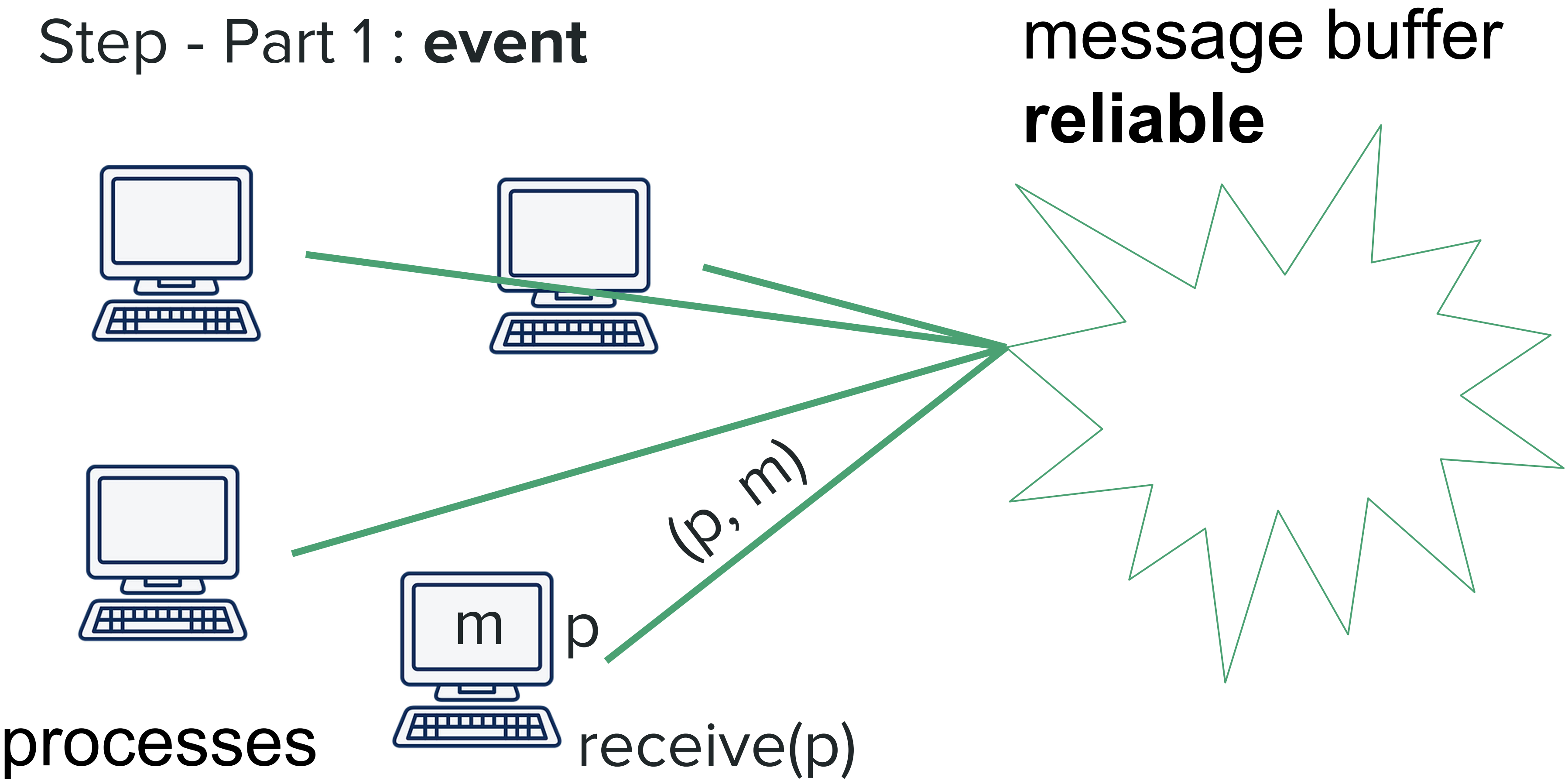



message buffer
send(p, m)
(p, m)
processes

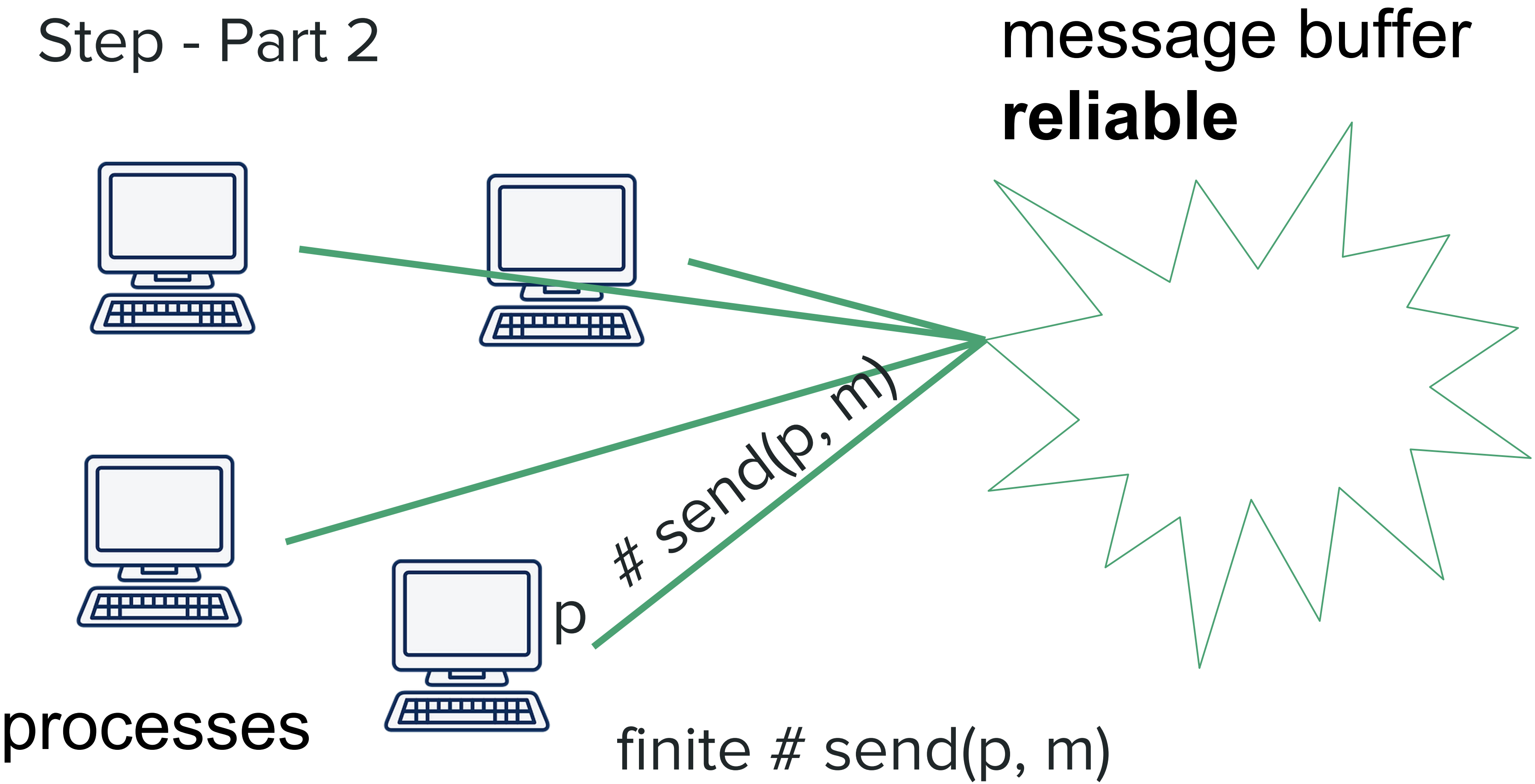

message buffer
(p, m)
∅
p
∅
receive(p)
processes


message buffer
reliable
(p, m)
m
p
processes
receive(p)

Step - Part 1 : **event**

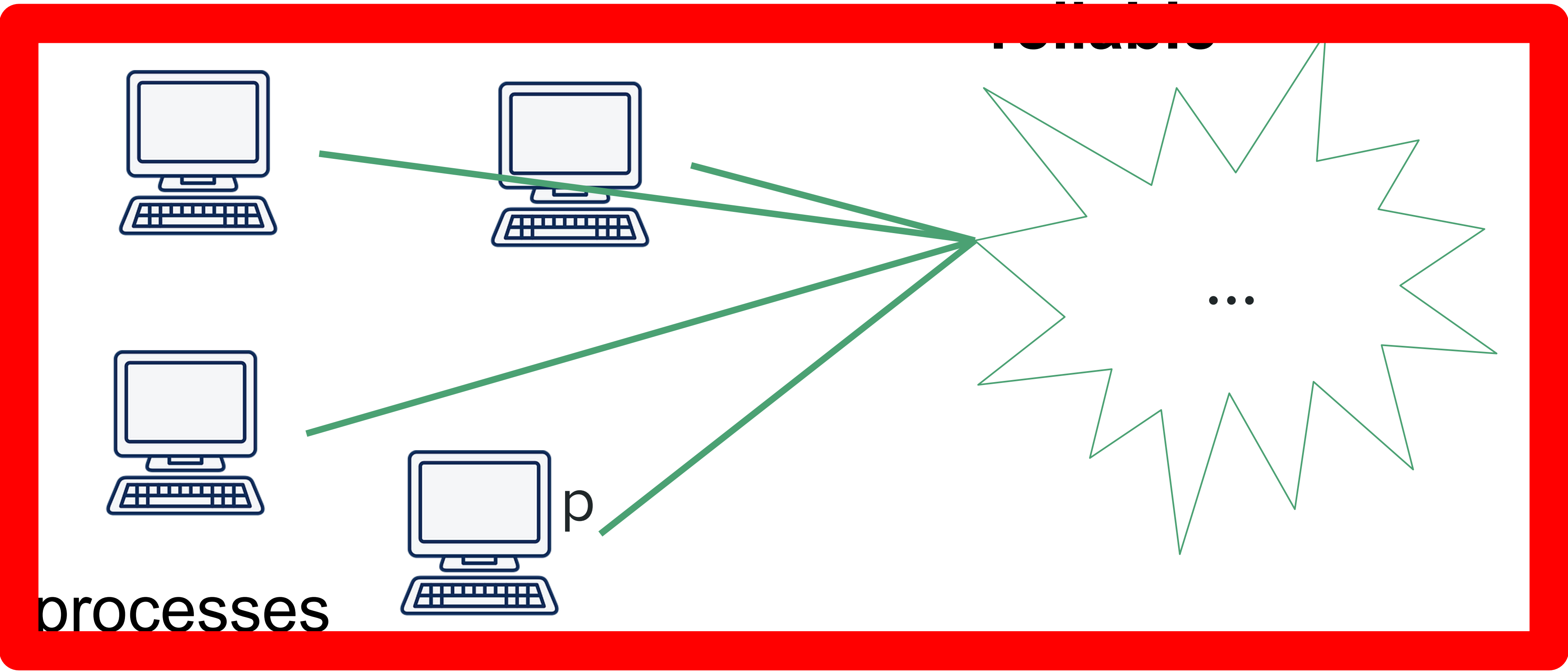


Step - Part 2

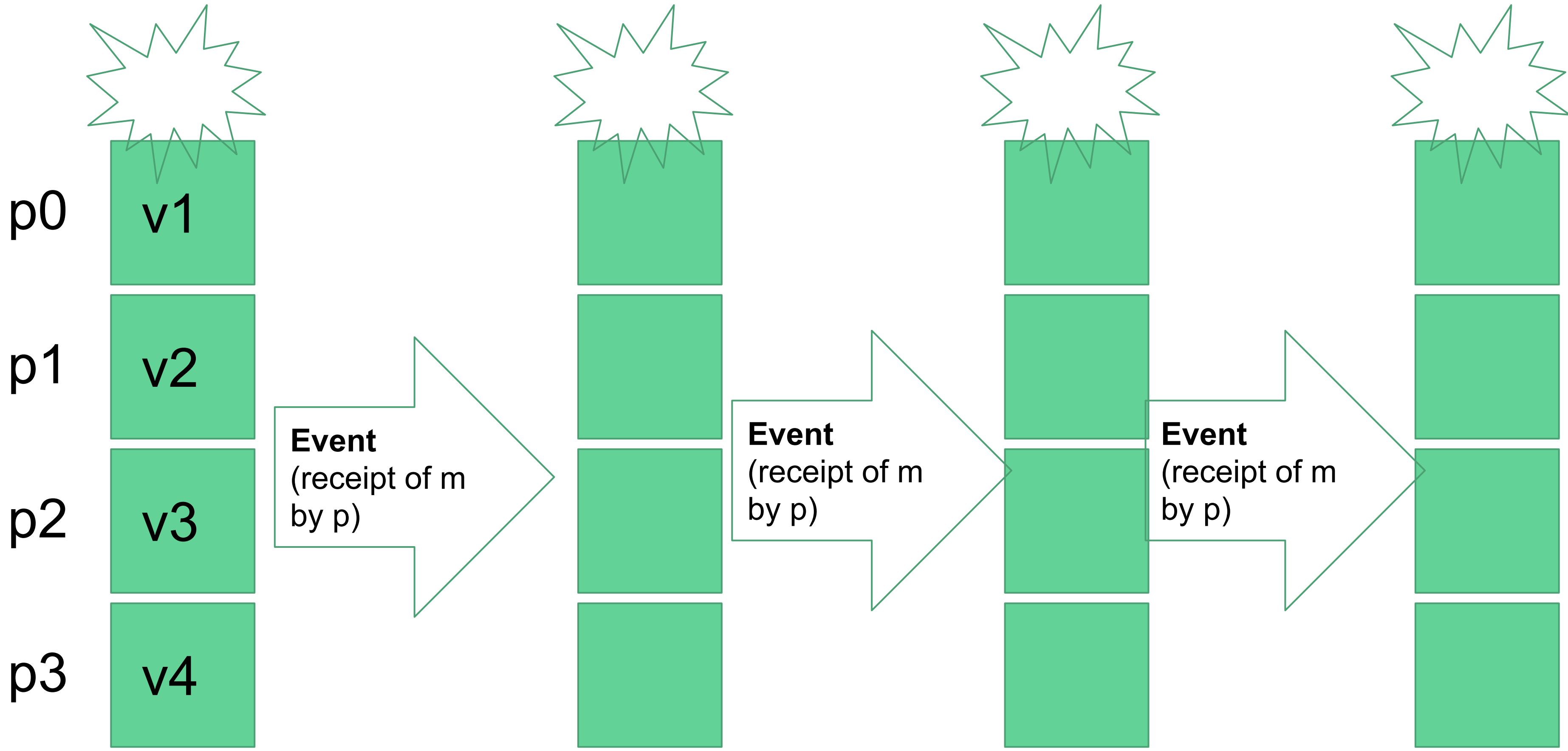


Configuration

message buffer

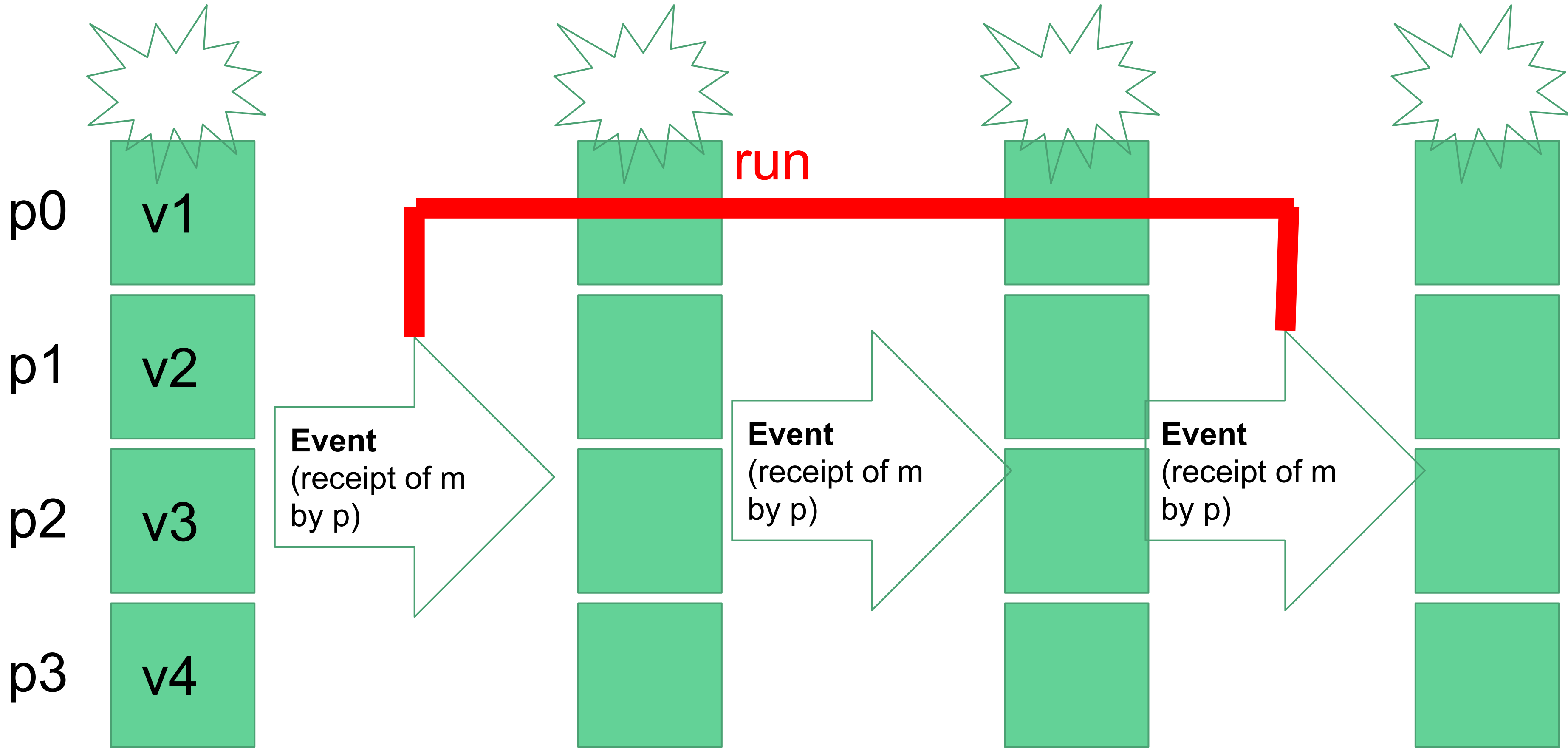


Schedule - σ



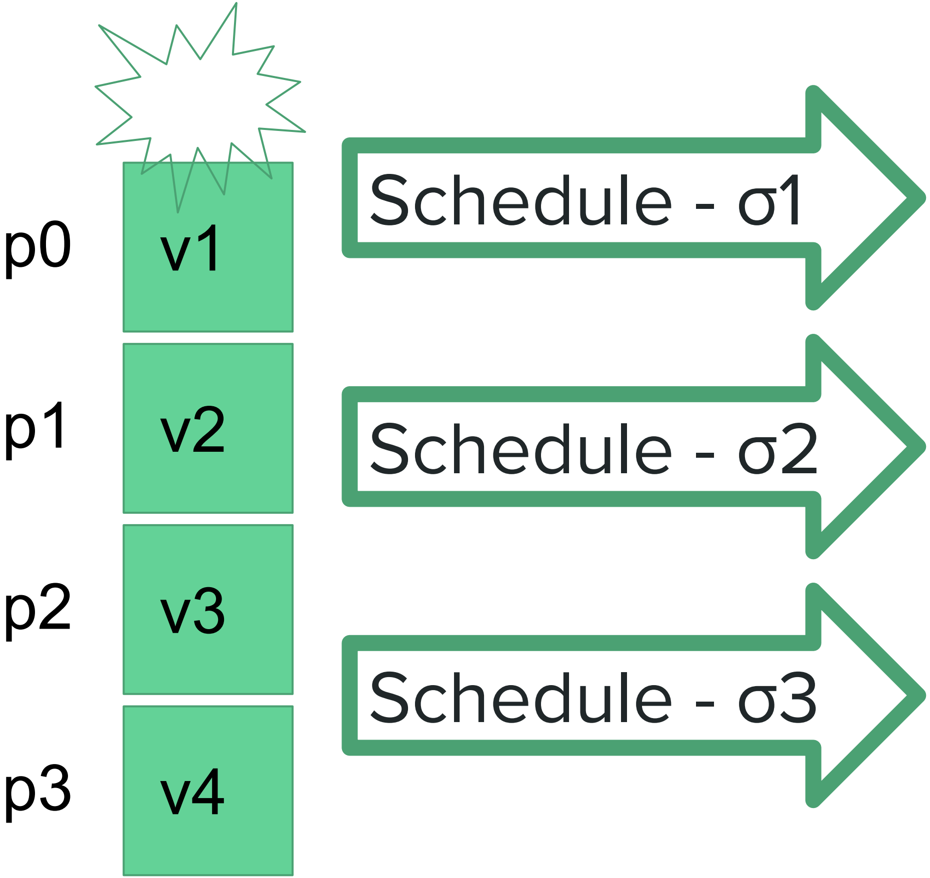
p0
p1
p2
p3
v1
v2
v3
v4
Event
(receipt of m by p)
Event
(receipt of m by p)
Event
(receipt of m by p)

Run



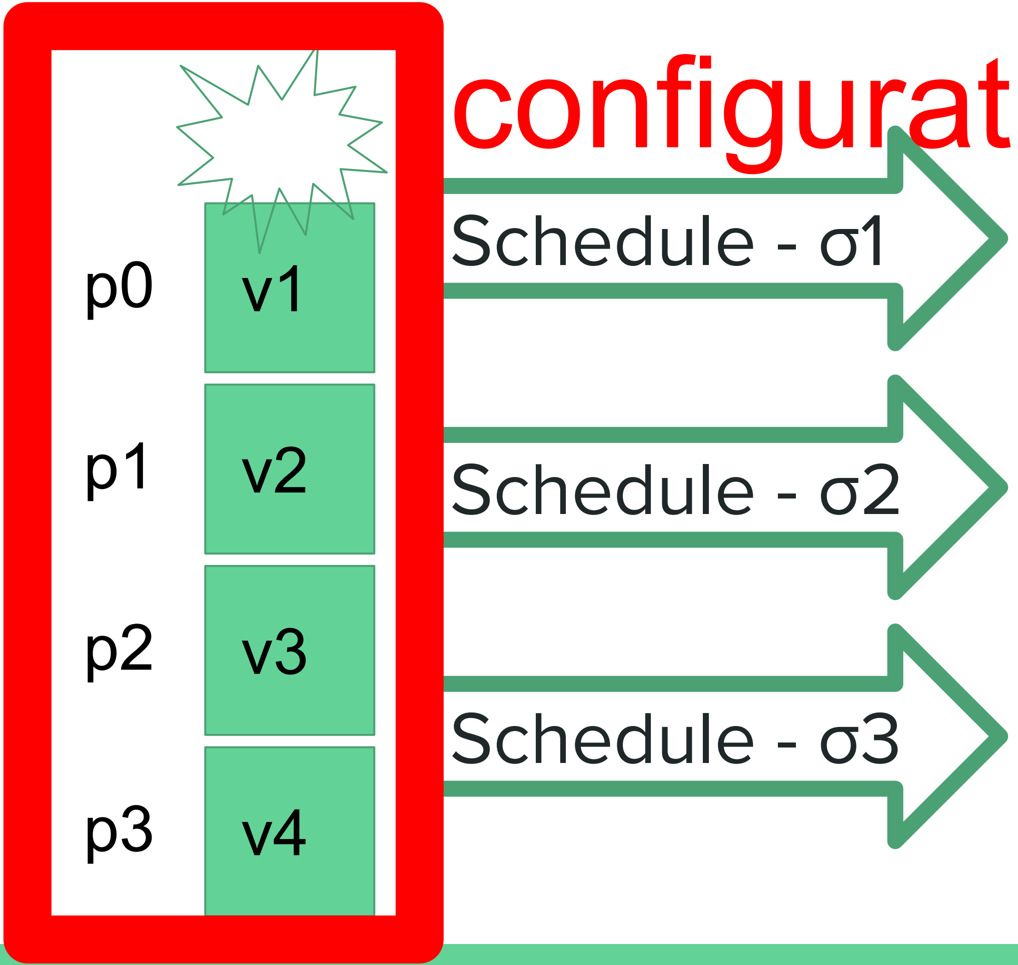
run
p0
p1
p2
p3
v1
v2
v3
v4
Event
(receipt of m
by p)
Event
(receipt of m
by p)
Event
(receipt of m
by p)

0-Valent Configuration



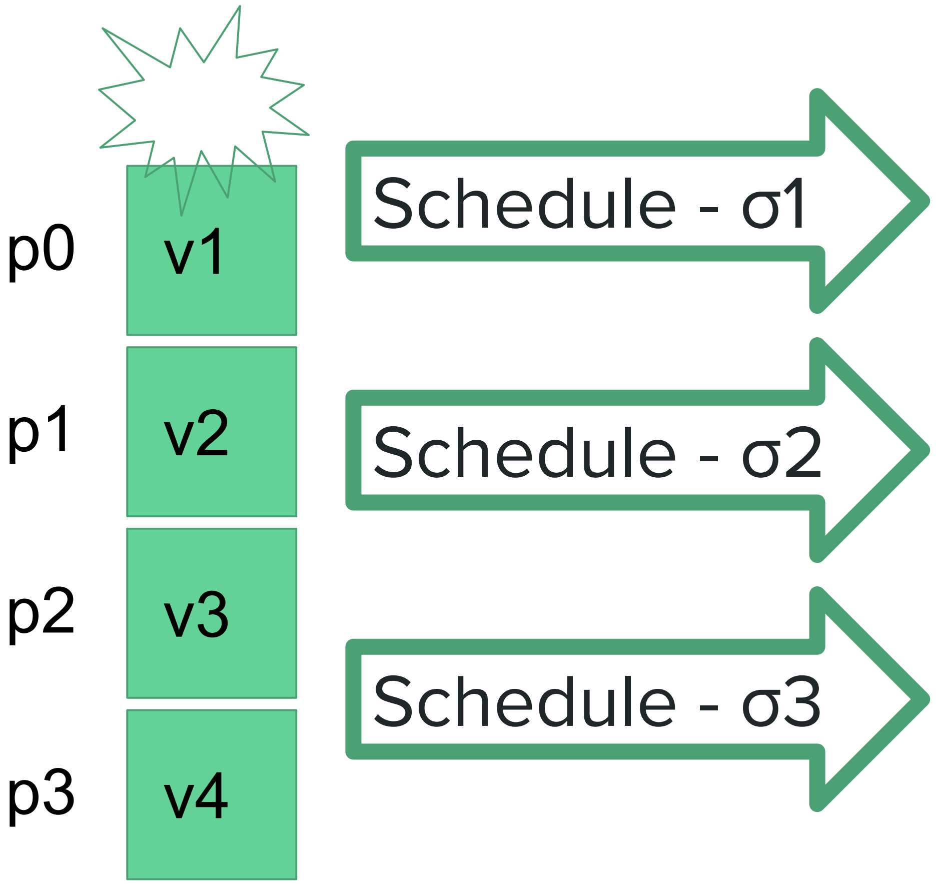
All
Processes
Decide
0

Initial configuration



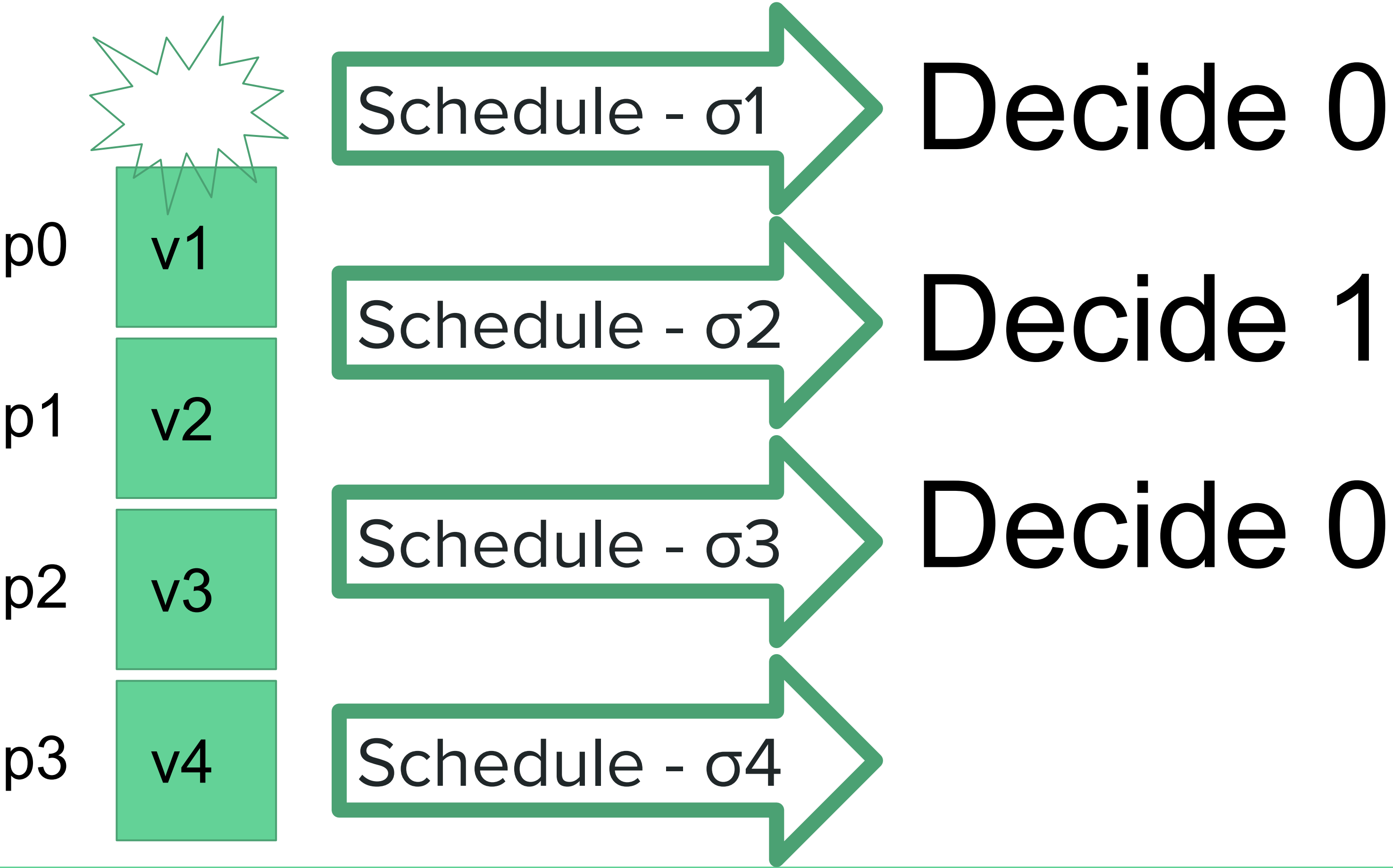
All Processes Decide 0

1-Valent Configuration



All Processes Decide 1

Bivalent Configuration (Read: Undecided)

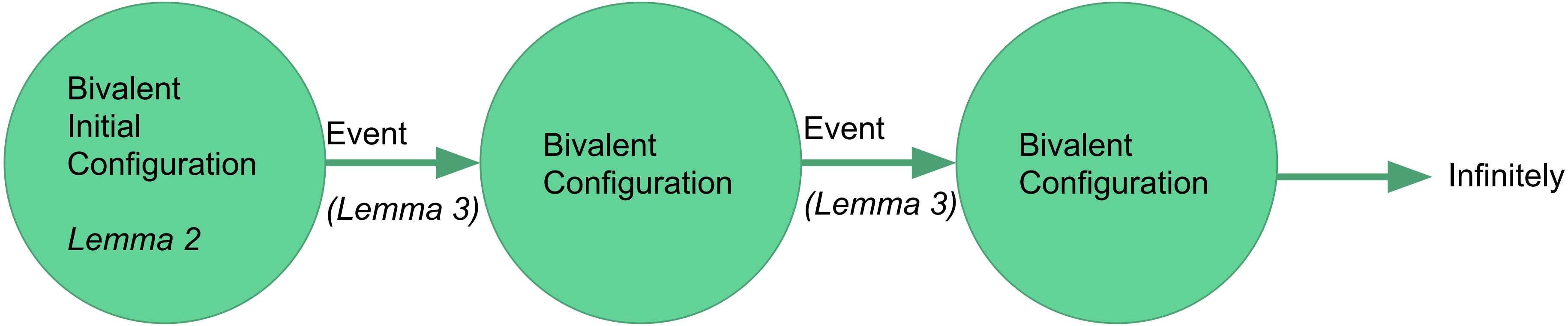


Now, we prove:

Any protocol in our model must have an infinitely long run (that never terminates)

Proof Outline

- Start from the initial guaranteed bivalent configuration (Lemma 2)
- Since the configuration is bivalent, there must be another bivalent configuration reachable from the configuration by applying e last (Lemma 3)
- Since the configuration is bivalent... (Lemma 3)



Lemma 1; Housekeeping

Schedules are commutative

Proof! (Lemma 1) [from the paper]

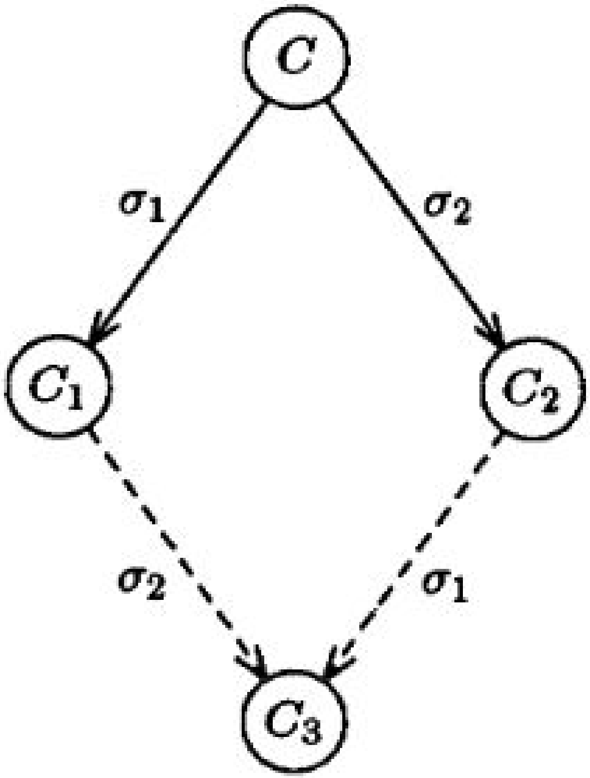


FIGURE 1

Lemma 2

There is an initial **bivalent** configuration

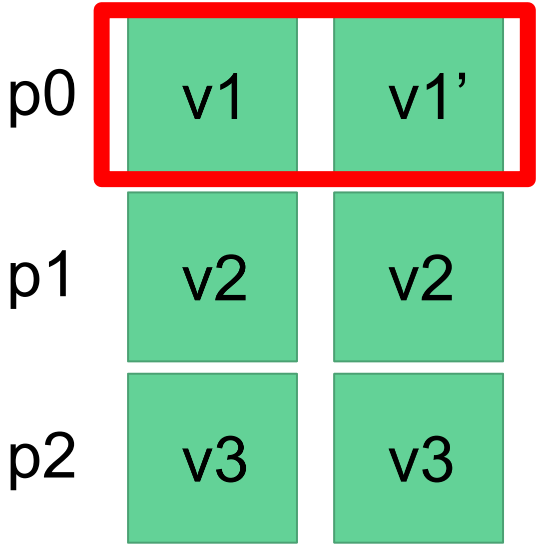
(see: bivalent; read: undetermined / undecided)

Initial Configurations - **neighbors**

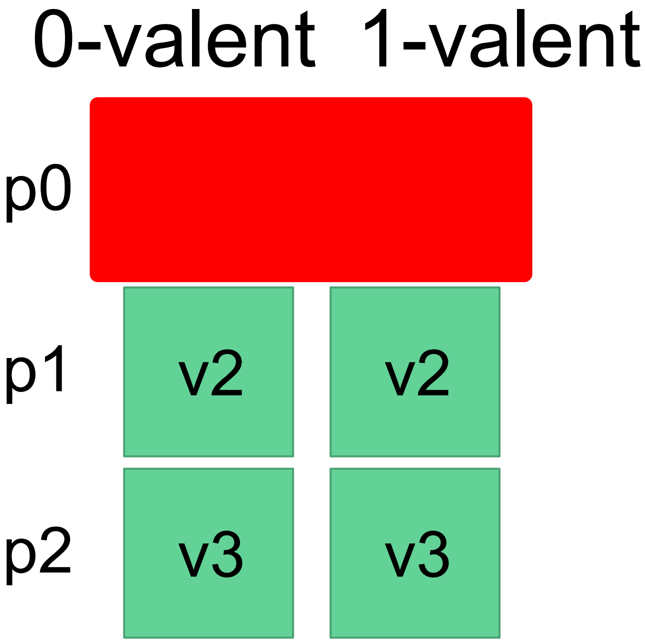
	0-valent	1-valent
p0	v1	v1’
p1	v2	v2
p2	v3	v3

Initial Configurations

0-valent 1-valent

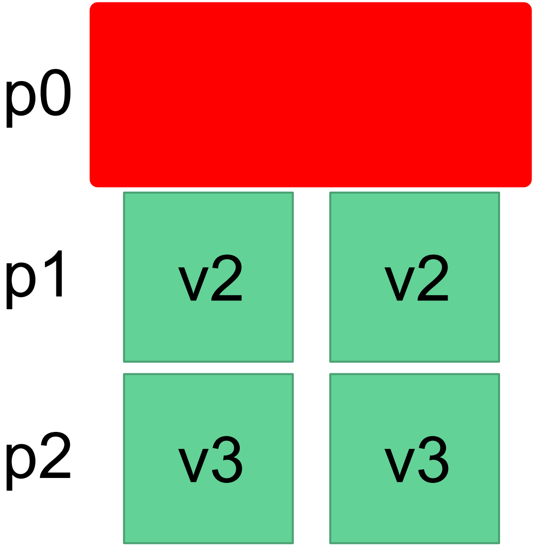


Initial Configurations



Initial Configurations

~~0 valent 1 valent~~



bivalent OR both 0 OR both 1

3 Processes - All Possible Inputs

p0	0	1	1	0	0	1	1	0
p1	0	0	1	1	1	1	0	0
p2	0	0	0	0	1	1	1	1

3 Processes - Neighbors differ by 1 Process Input

p0	0	1	1	0	0	1	1	0
p1	0	0	1	1	1	1	0	0
p2	0	0	0	0	1	1	1	1

We want to prove

There is an initial **bivalent** configuration

assume the opposite -

All initial configurations **univalent**

(see: bivalent; read: undetermined / undecided)

3 Processes - A Univalent-Only Scheme

	1	0	1	0	1	0	1	0
p0	0	1	1	0	0	1	1	0
p1	0	0	1	1	1	1	0	0
p2	0	0	0	0	1	1	1	1

3 Processes - Another Univalent-Only Scheme

	0	0	0	0	1	1	1	1
p0	0	1	1	0	0	1	1	0
p1	0	0	1	1	1	1	0	0
p2	0	0	0	0	1	1	1	1

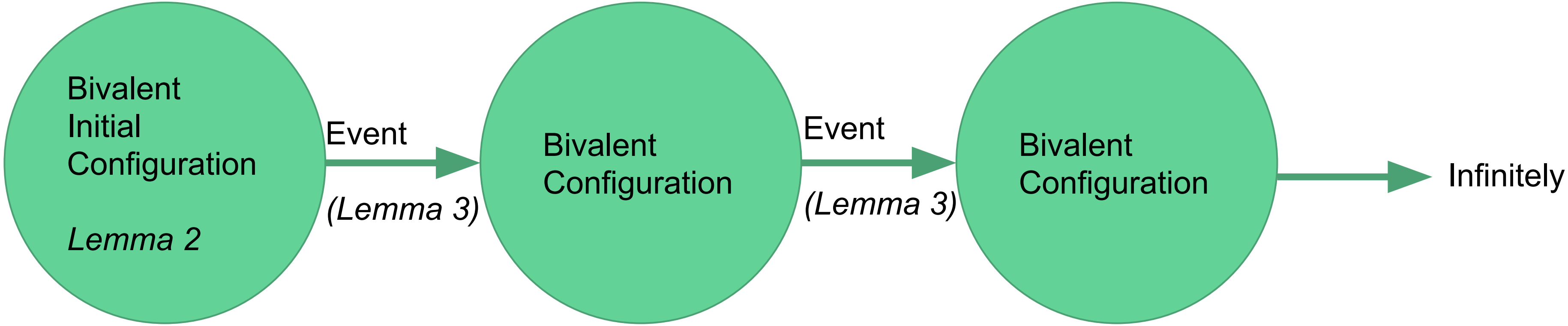
So

Univalent only schemes **don’t work**

Must have initial bivalent configuration!

Reminder

- Start from the initial guaranteed bivalent configuration (Lemma 2)
- Since the configuration is bivalent, there must be another bivalent configuration reachable from the configuration by applying e last (Lemma 3)
- Since the configuration is bivalent… (Lemma 3)



Lemma 3

If C is a bivalent configuration, and e is an event applicable to C, there is a bivalent configuration reachable by applying e last

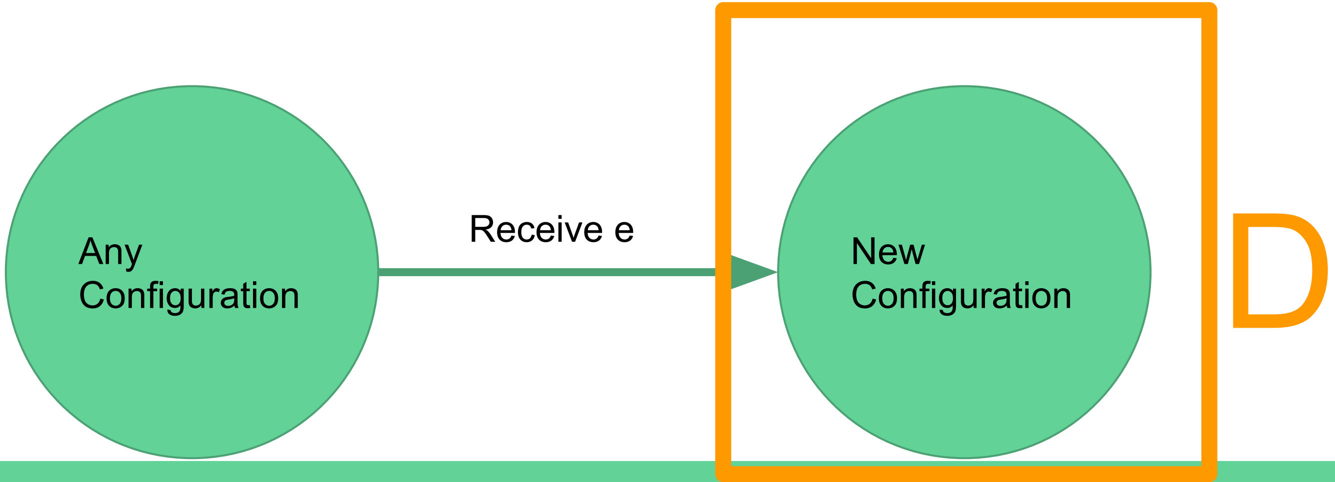
(this is the big one)

Lemma 3

2 Ingredients:

An event, e (fix any event)

D - all configurations right after e



Lemma 3

We will show:

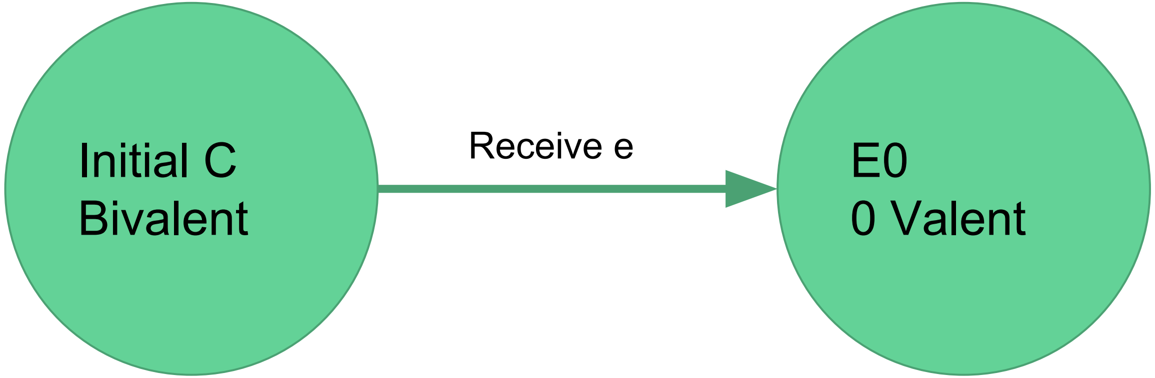
D has a bivalent configuration

(through **series of contradictions**)

Lemma 3 - Contradiction 1

D has only 1-valent configurations

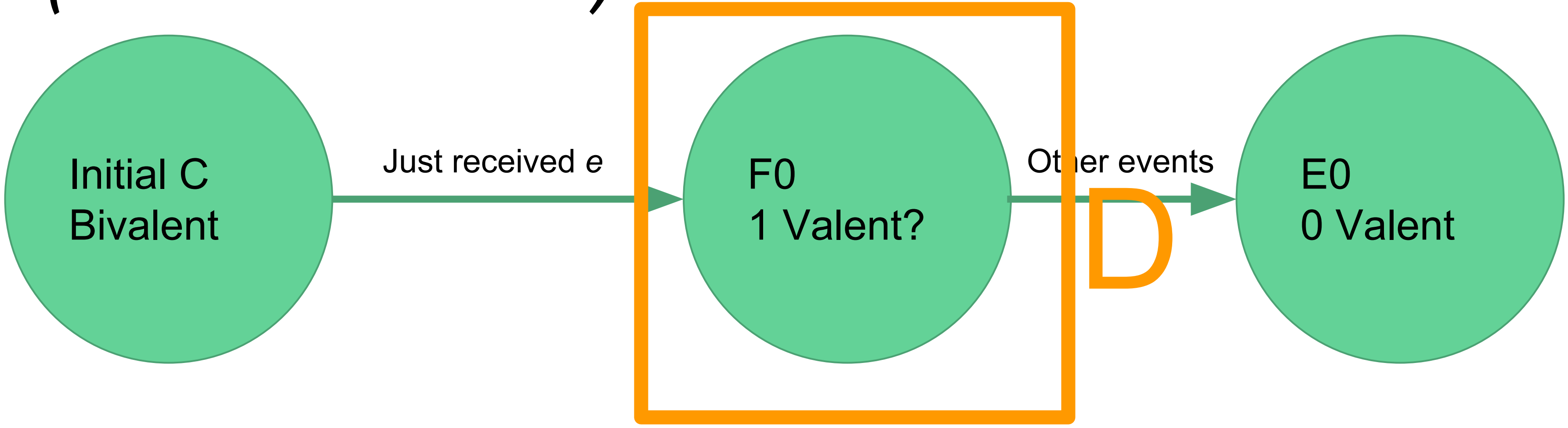
(E0 has seen e)



Lemma 3 - Contradiction 1

D has only 1-valent configurations

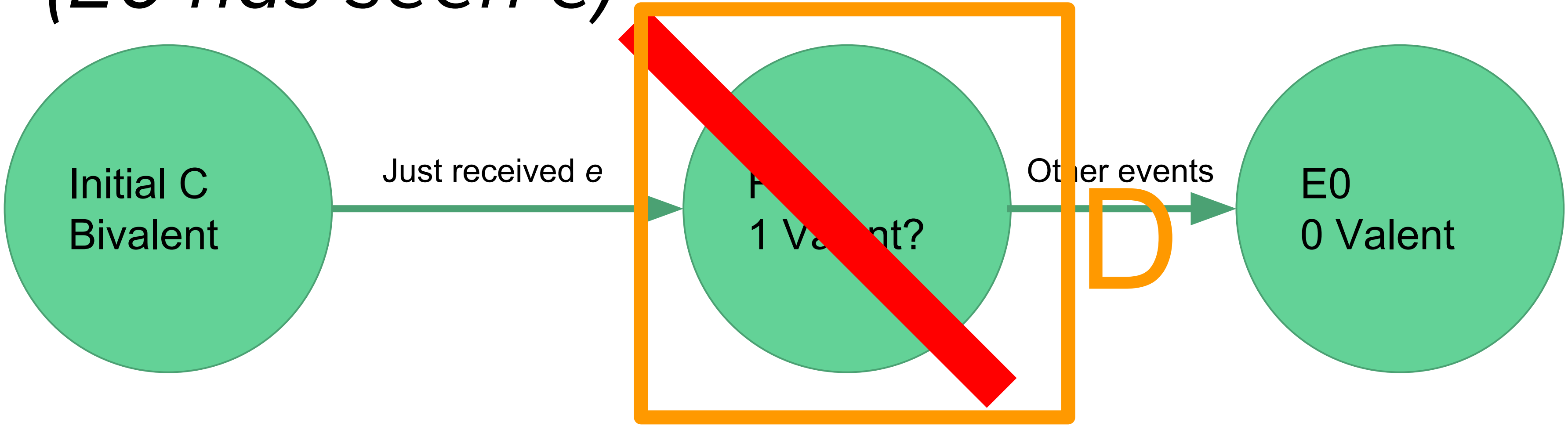
(E0 has seen e)



Lemma 3 - Contradiction 1

~~D has only 1 valent configurations~~

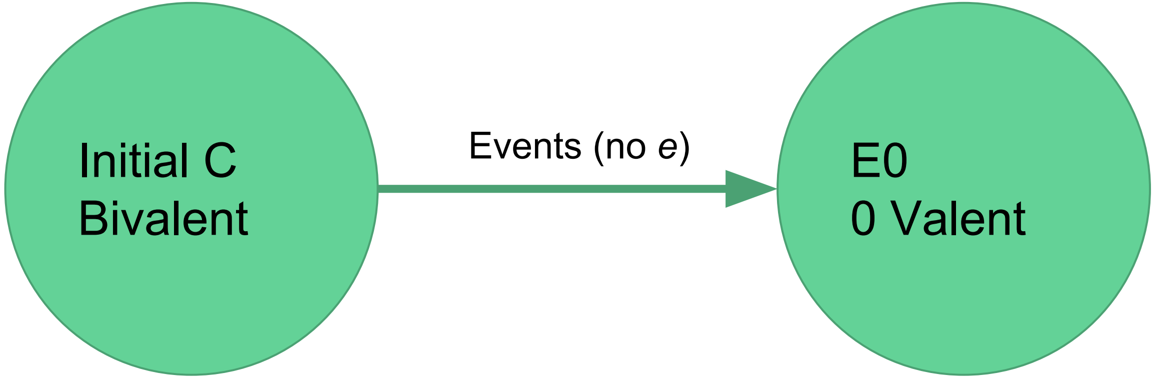
(E0 has seen e)



Lemma 3 - Contradiction 1

D has only 1-valent configurations

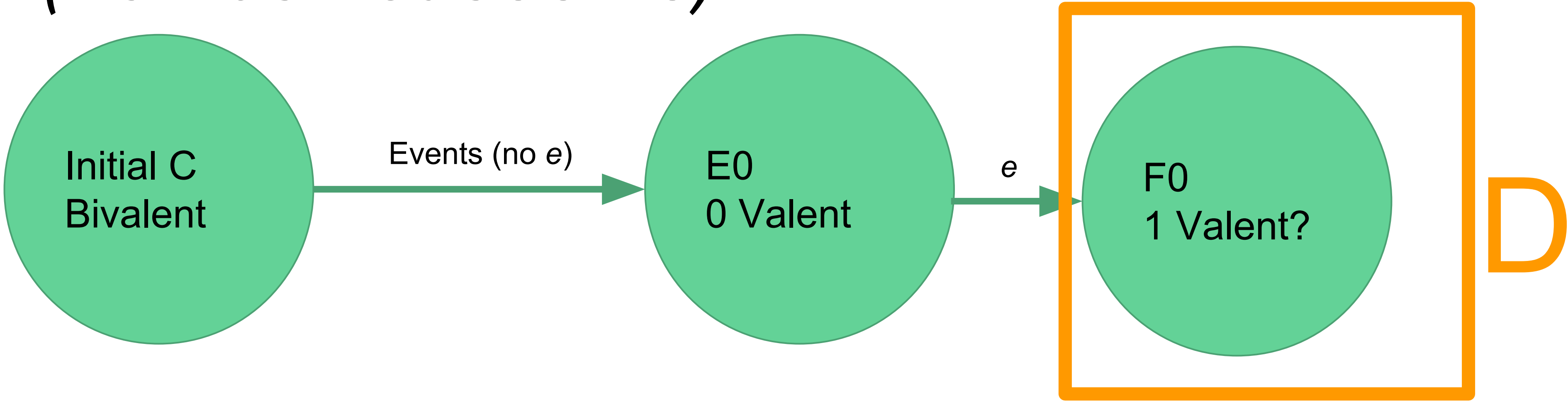
(E0 has not seen e)



Lemma 3 - Contradiction 1

D has only 1-valent configurations

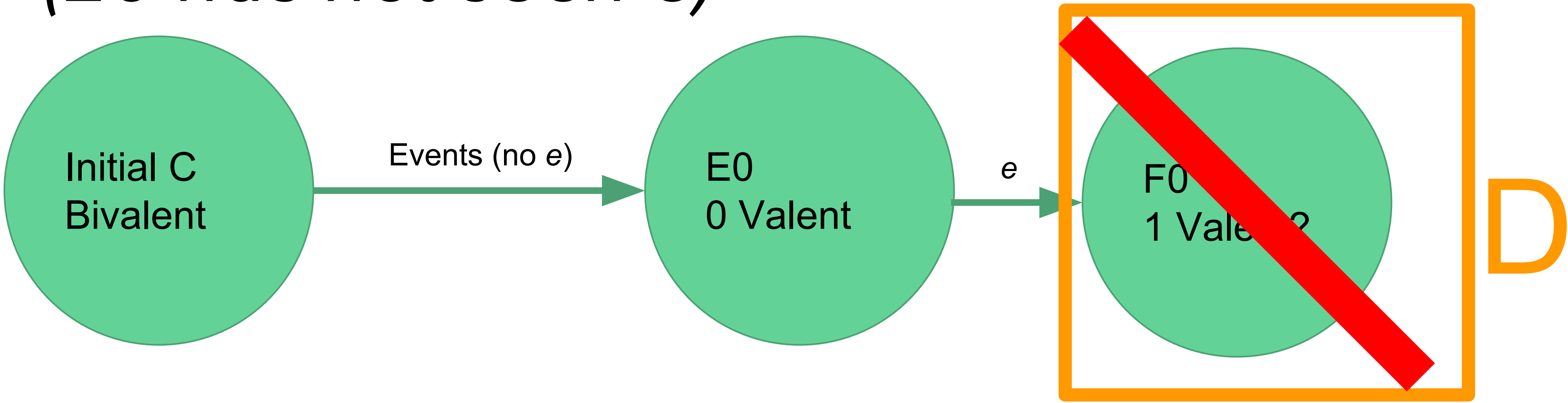
(E0 has not seen e)



Lemma 3 - Contradiction 1

~~**D** has only 1-valent configurations~~

(E0 has not seen e)



Summary

Disproven:

D has only 1-valent configurations

D has only 0-valent configurations (same)

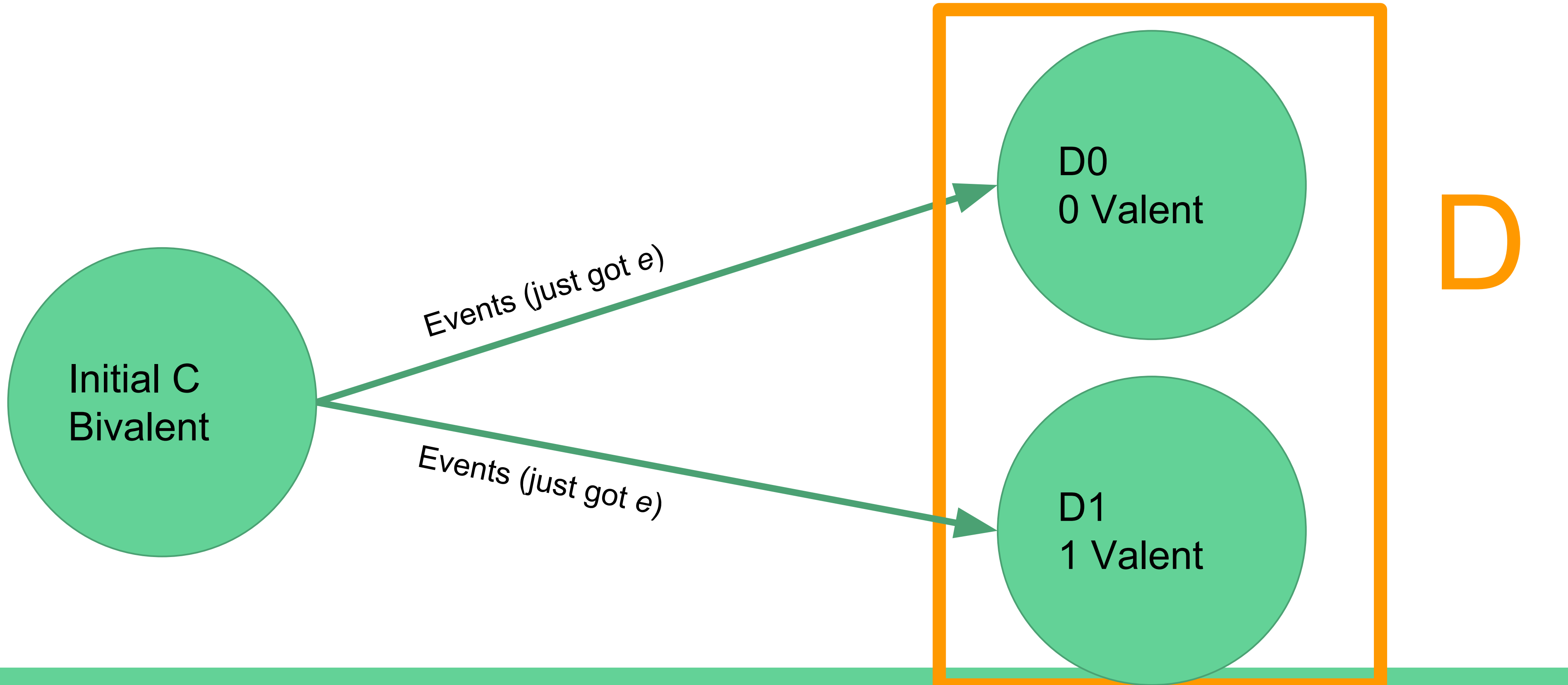
2 Possibilities:

D has only 1, 0 valent configurations (no bivalent) [next]

D has bivalent configurations

Lemma 3 - Contradiction 1

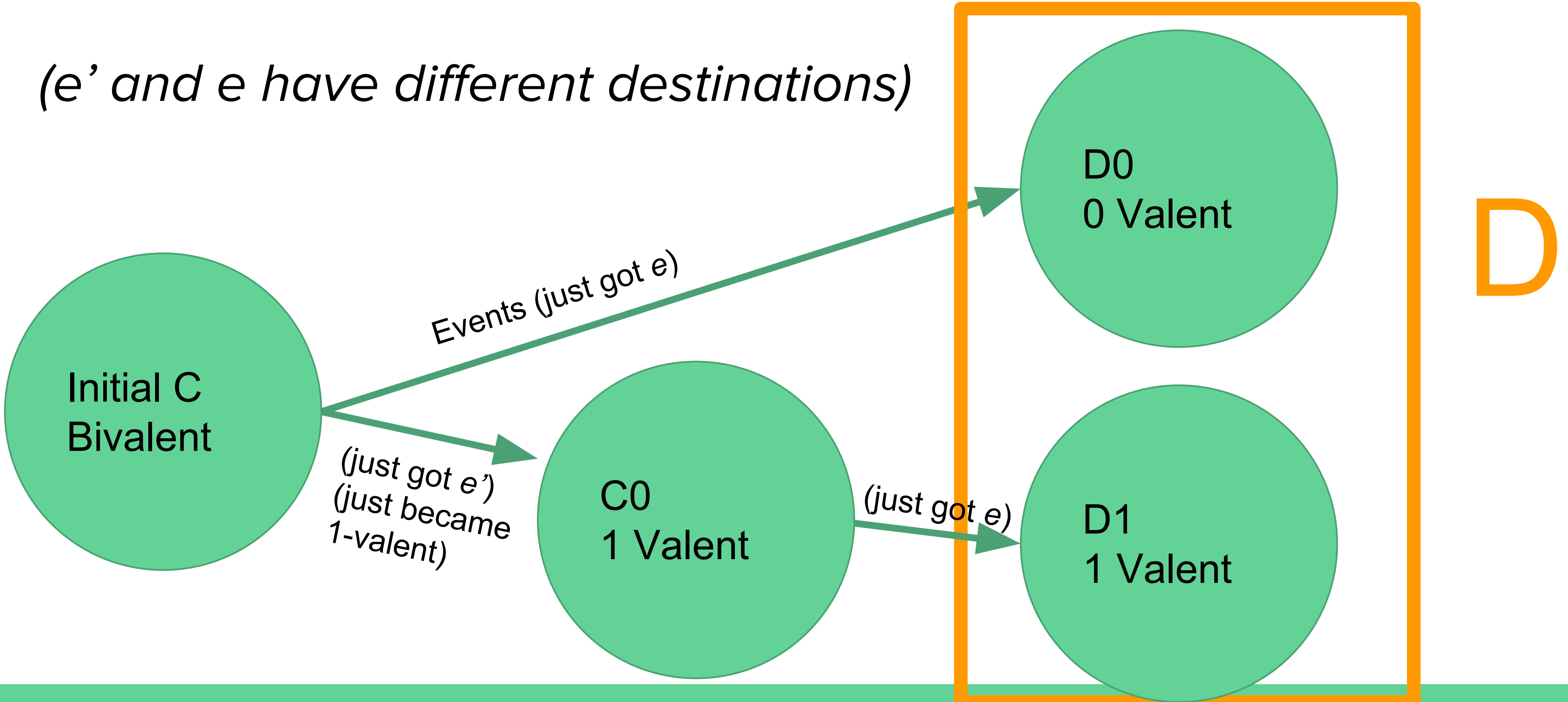
D has only 1, 0-valent configurations



Lemma 3 - Contradiction 1

D has only 1, 0-valent configurations

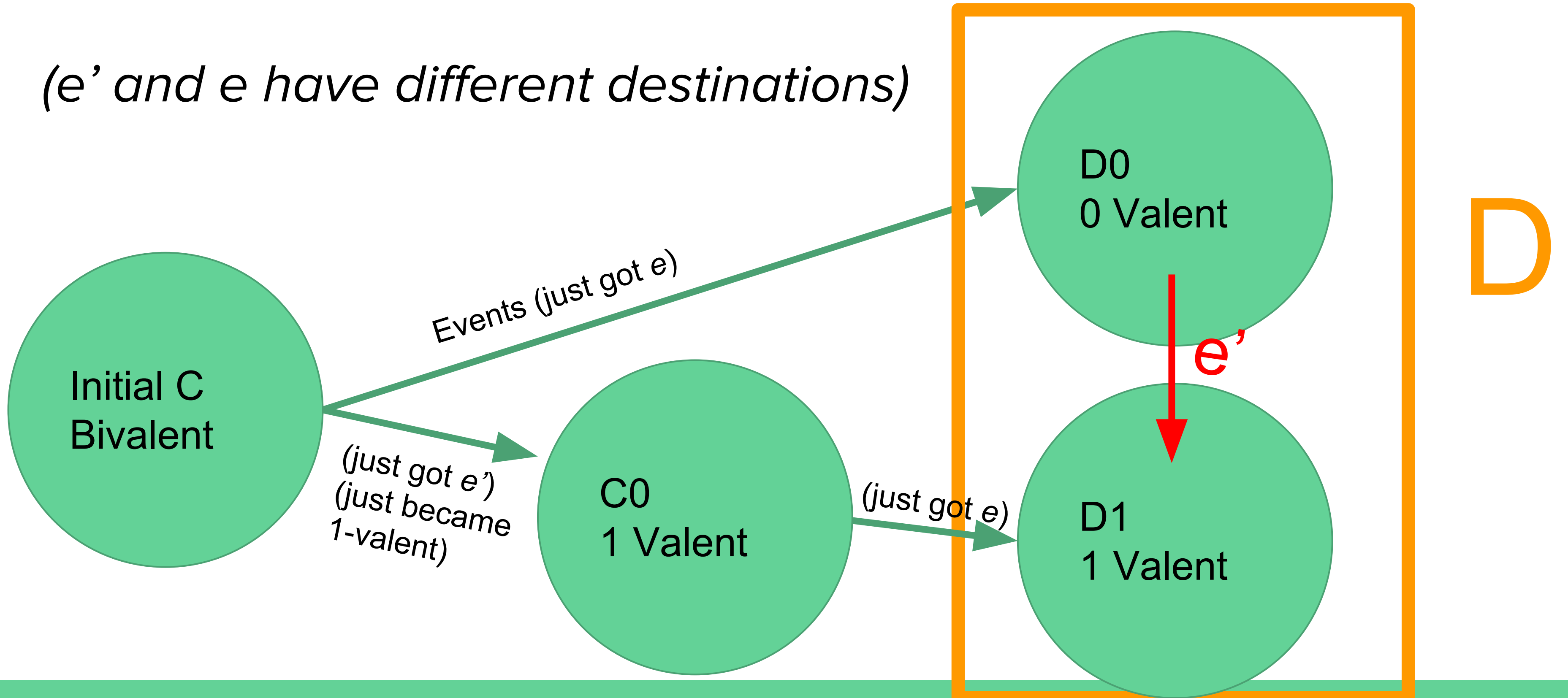
(e' and e have different destinations)



Lemma 3 - Contradiction 1

D has only 1, 0-valent configurations

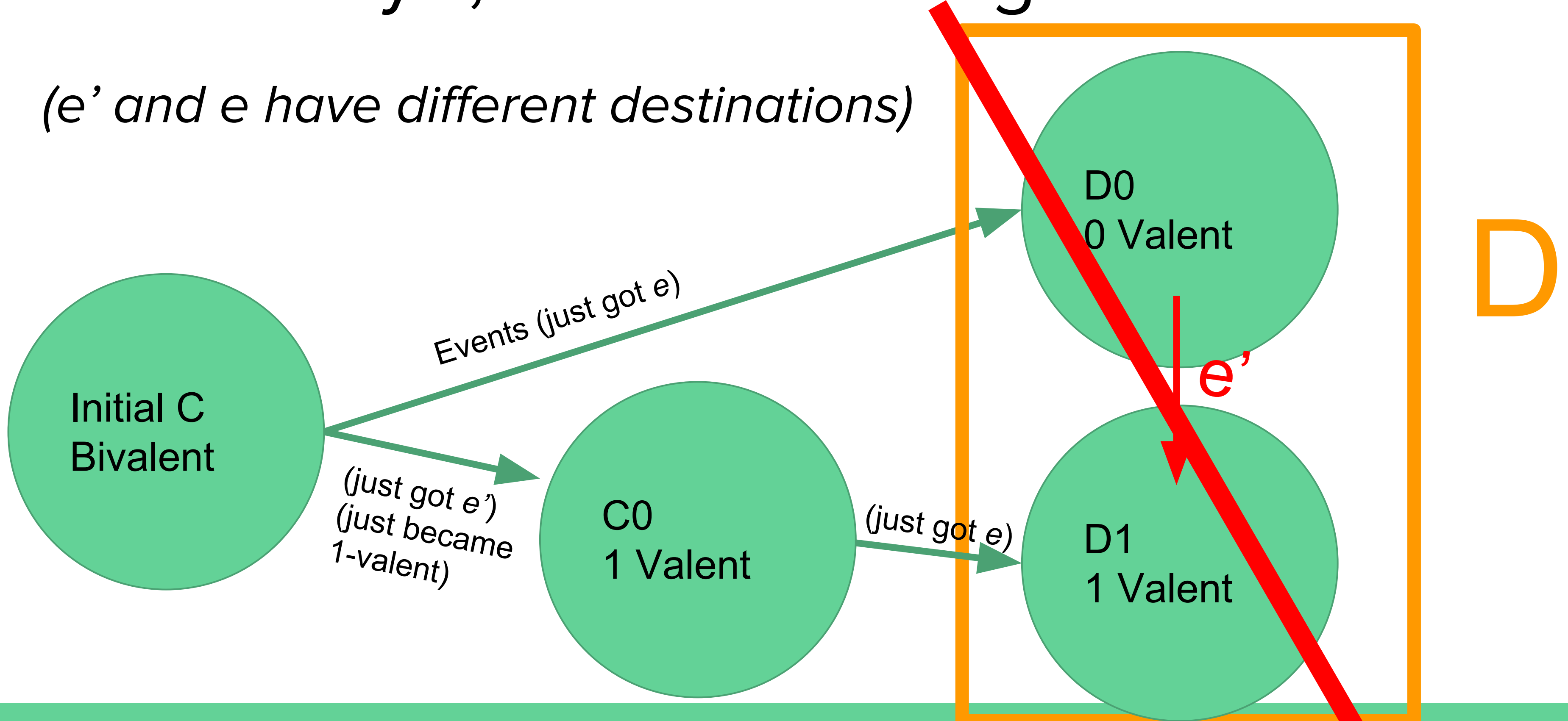
(e' and e have different destinations)



Lemma 3 - Contradiction 1

~~D has only 1, 0 valent configurations~~

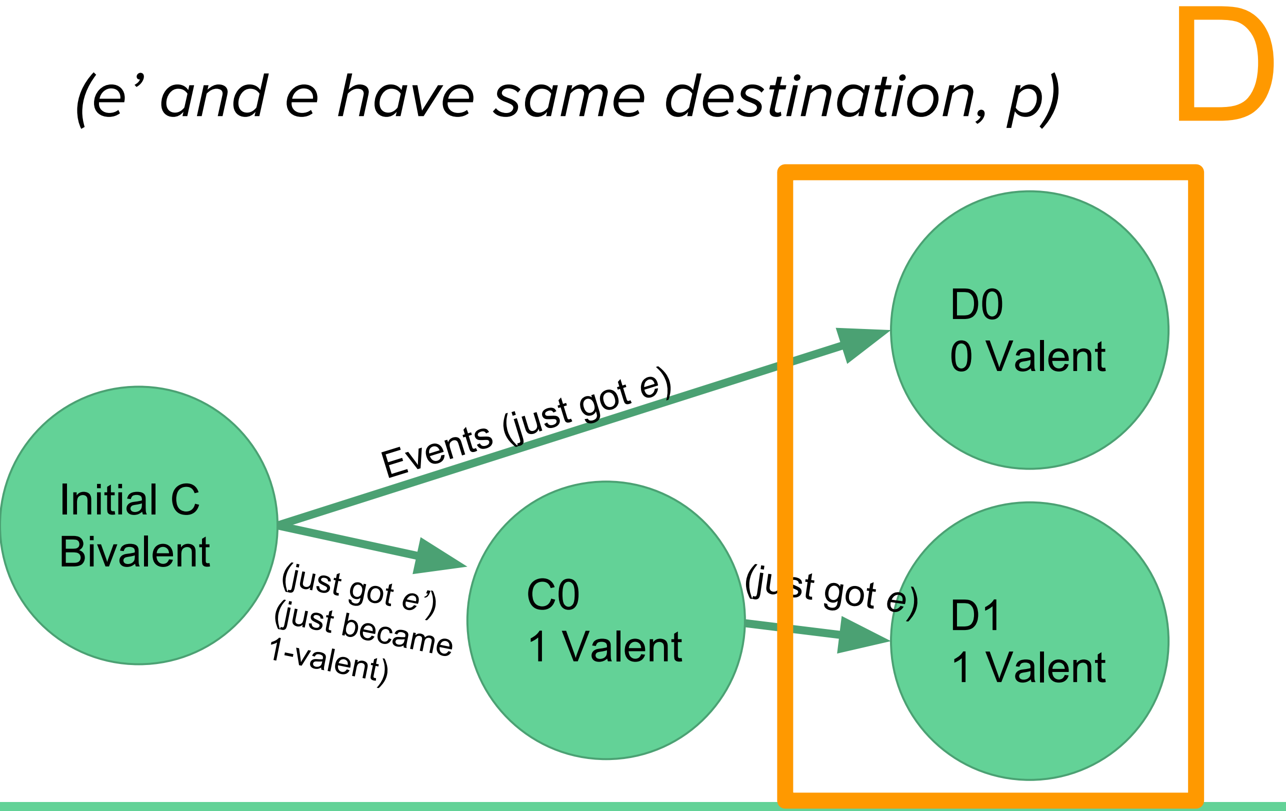
(e' and e have different destinations)



Lemma 3 - Contradiction 1

D has only 1, 0-valent configurations

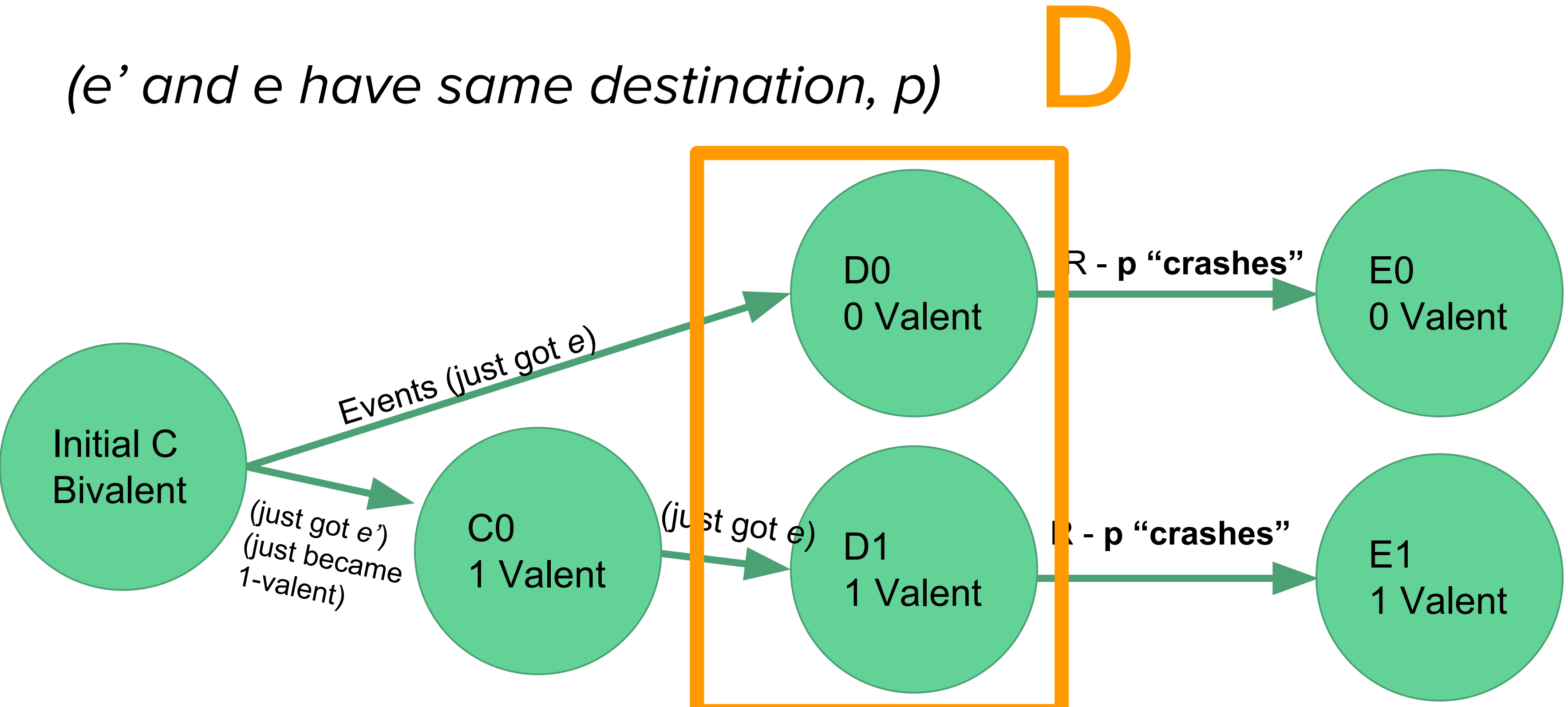
(e' and e have same destination, p)



Lemma 3 - Contradiction 1

D has only 1, 0-valent configurations

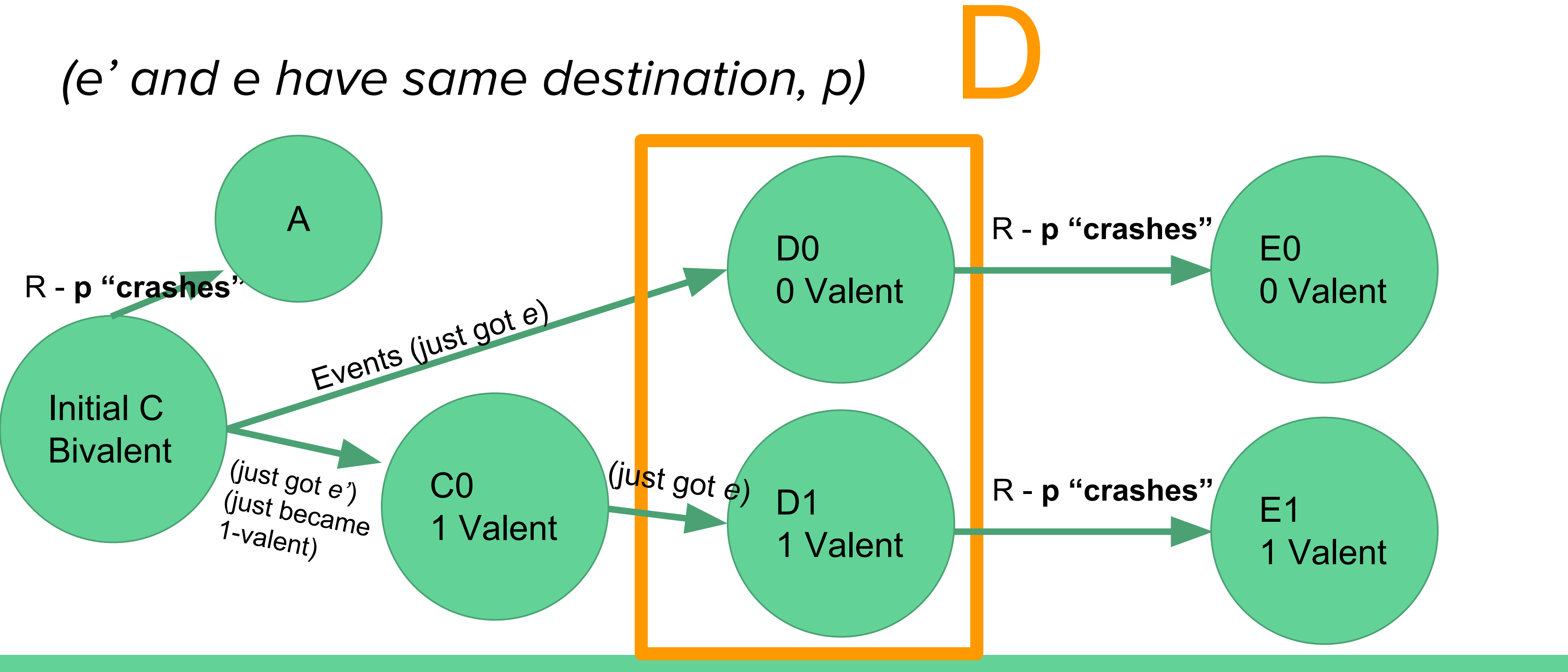
(e' and e have same destination, p)



Lemma 3 - Contradiction 1

D has only 1, 0-valent configurations

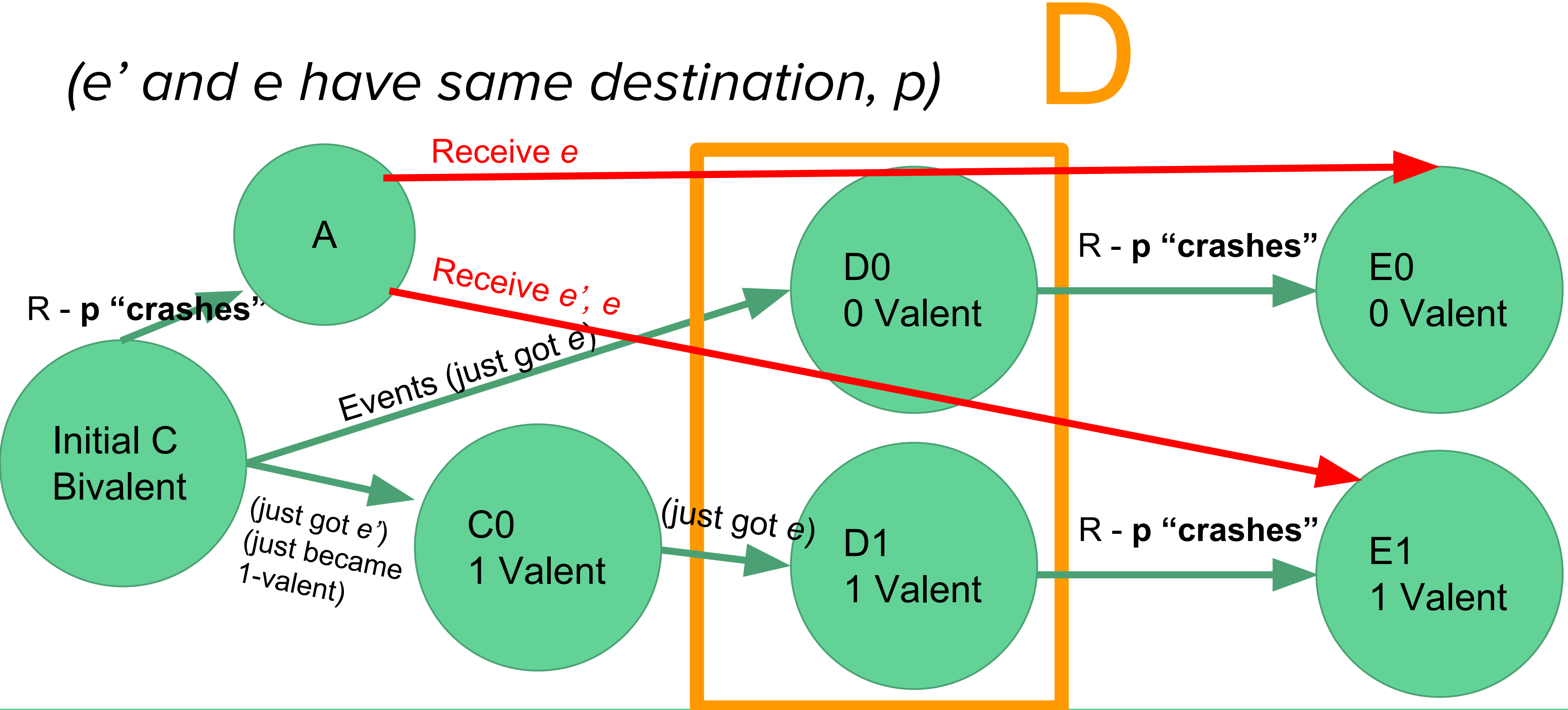
(e' and e have same destination, p)



Lemma 3 - Contradiction 1

D has only 1, 0-valent configurations

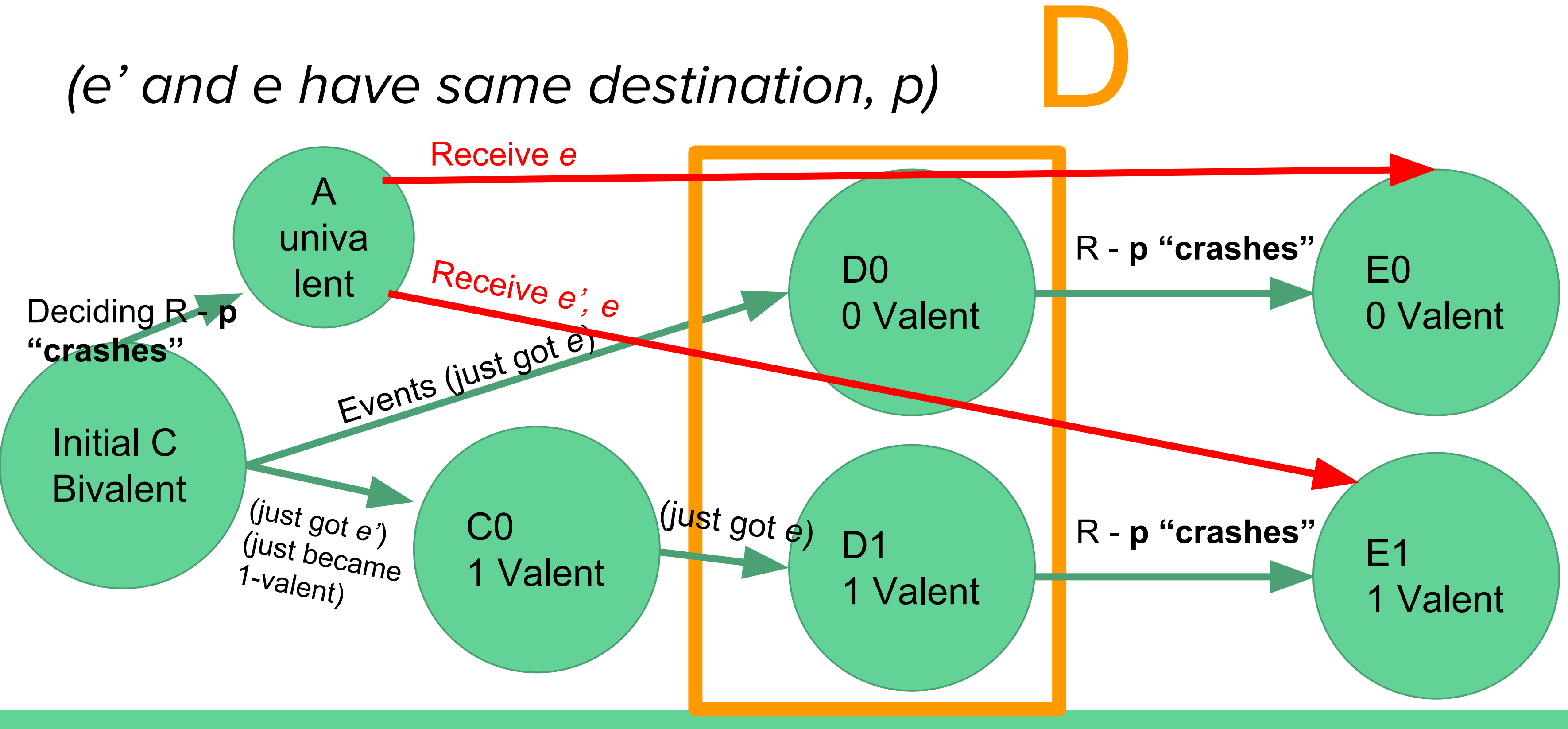
(e' and e have same destination, p)



Lemma 3 - Contradiction 1

~~D has only 1, 0 valent configurations~~

(e' and e have same destination, p)



Summary

Disproven:

D has only 1-valent configurations

D has only 0-valent configurations (same)

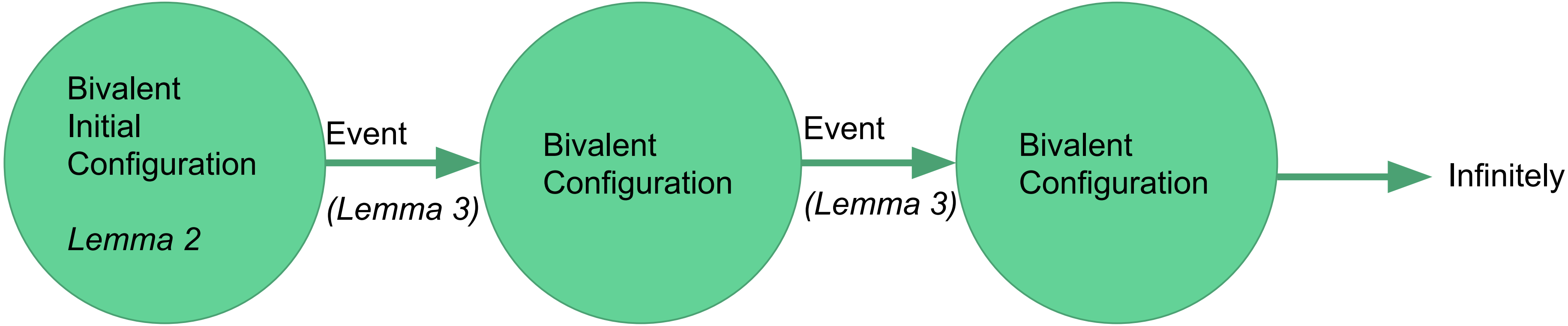
D has only 1, 0 valent configurations (no bivalent)

1 Possibility:

D has bivalent configurations

The whole proof!

- Start from the initial guaranteed bivalent configuration (Lemma 2)
- Since the configuration is bivalent, there must be a bivalent configuration (in D) reachable from the configuration by applying e last (Lemma 3)
- Since the configuration is bivalent... (Lemma 3)

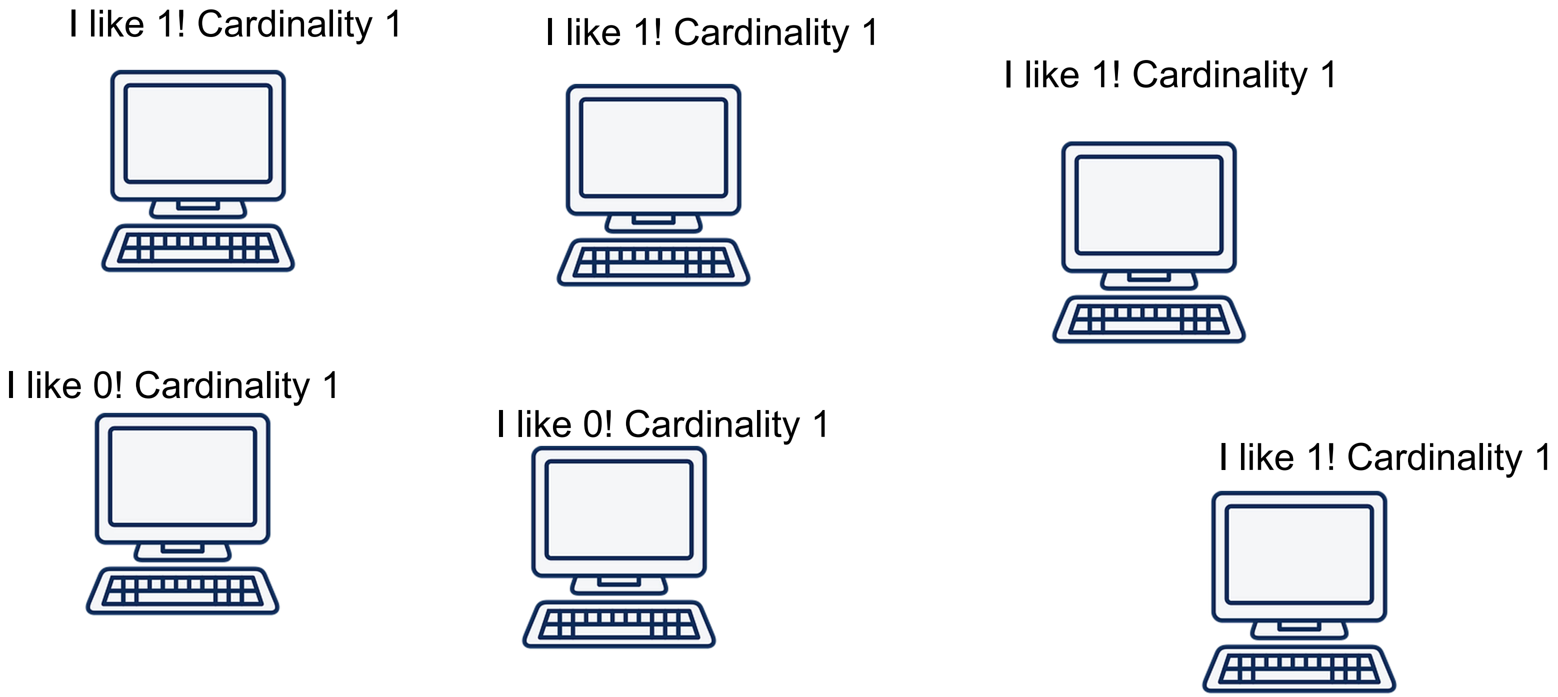


Beyond FLP

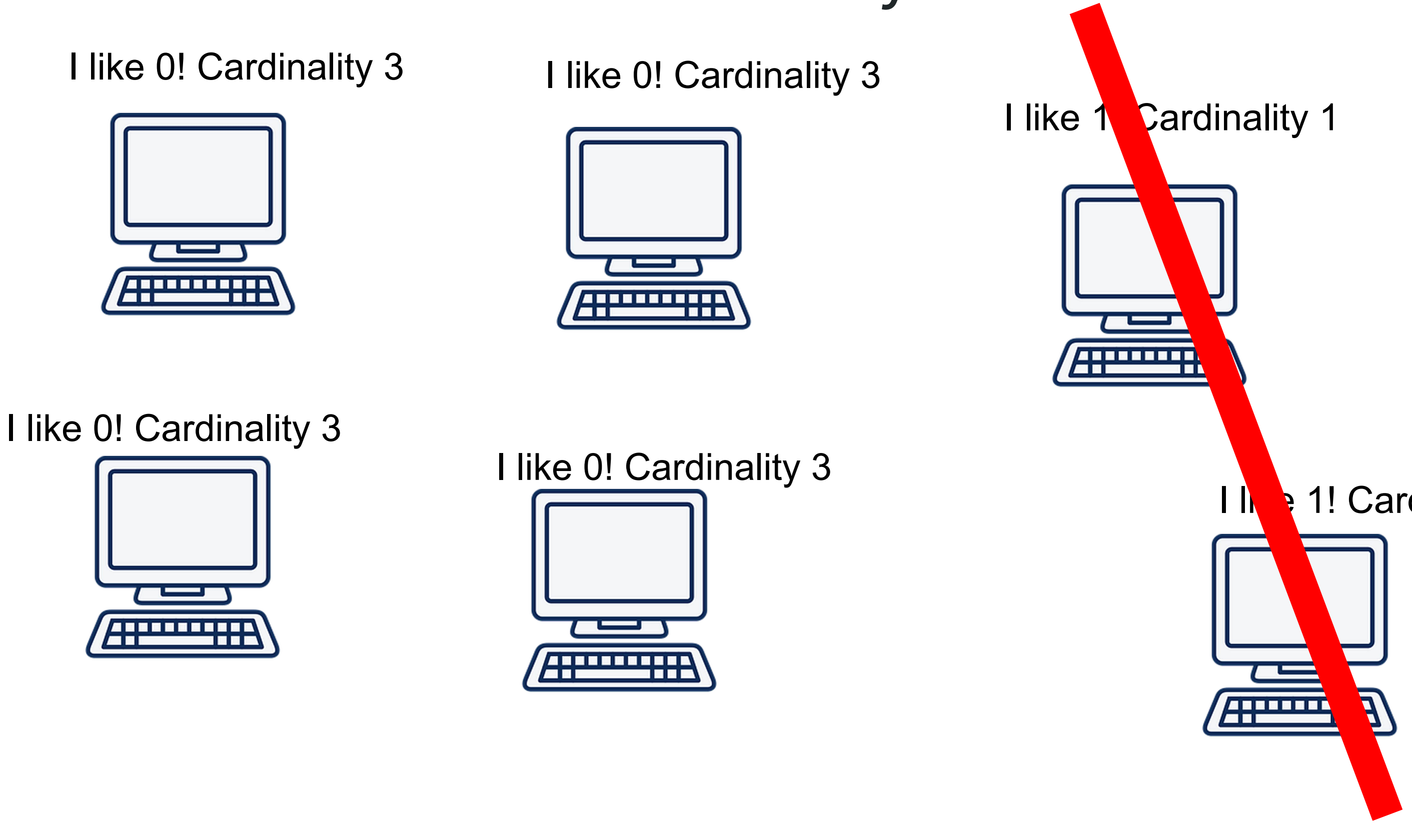
Work has continued far beyond the FLP result:

- Relaxing async model ; failure detectors
  - New models ; partially synchronous, sleepy, etc
  - Coming up next!
- Reducing other problems to consensus
  - SMR, leader election, atomic broadcast, shared log, ...
- New forms of consensus in permissionless models!
  - Bitcoin, blockchains, ByzCoin, etc.

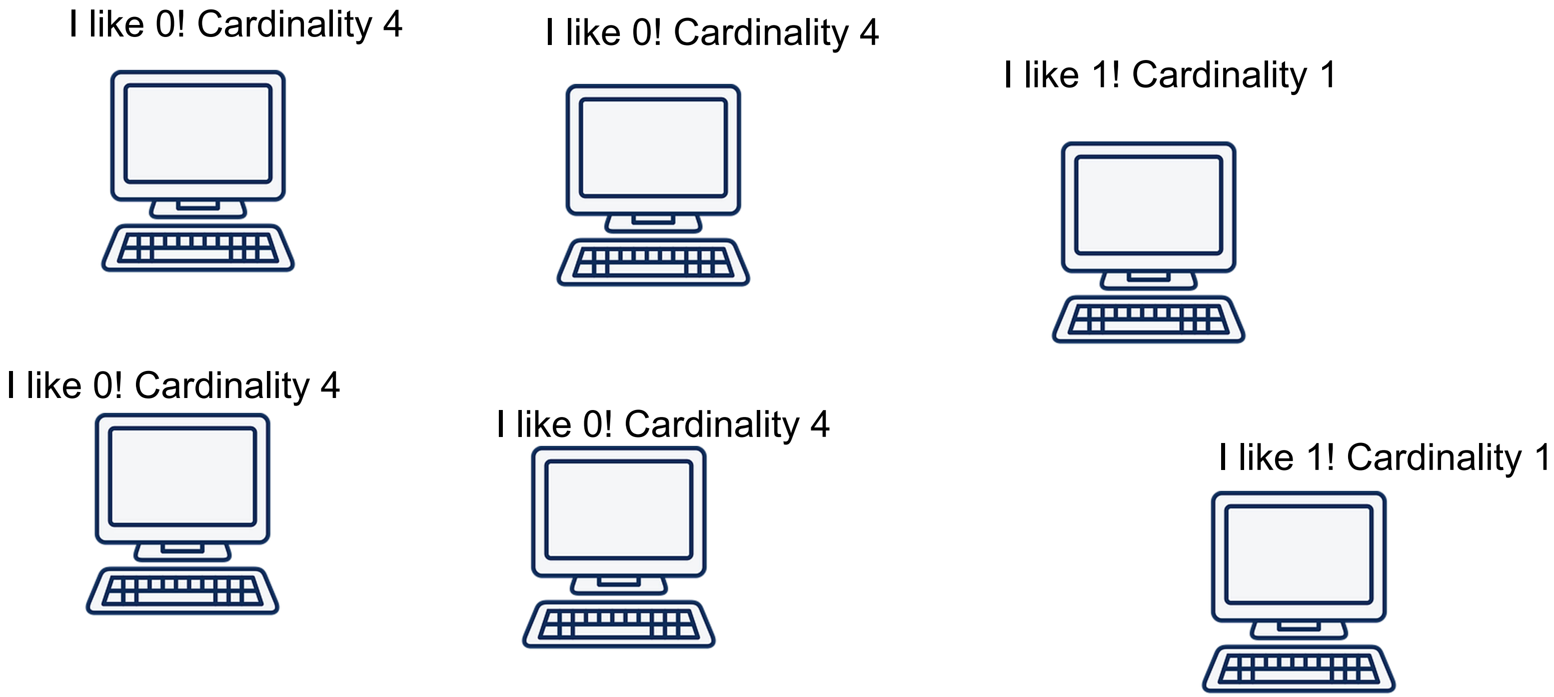
Consensus with Probability 1



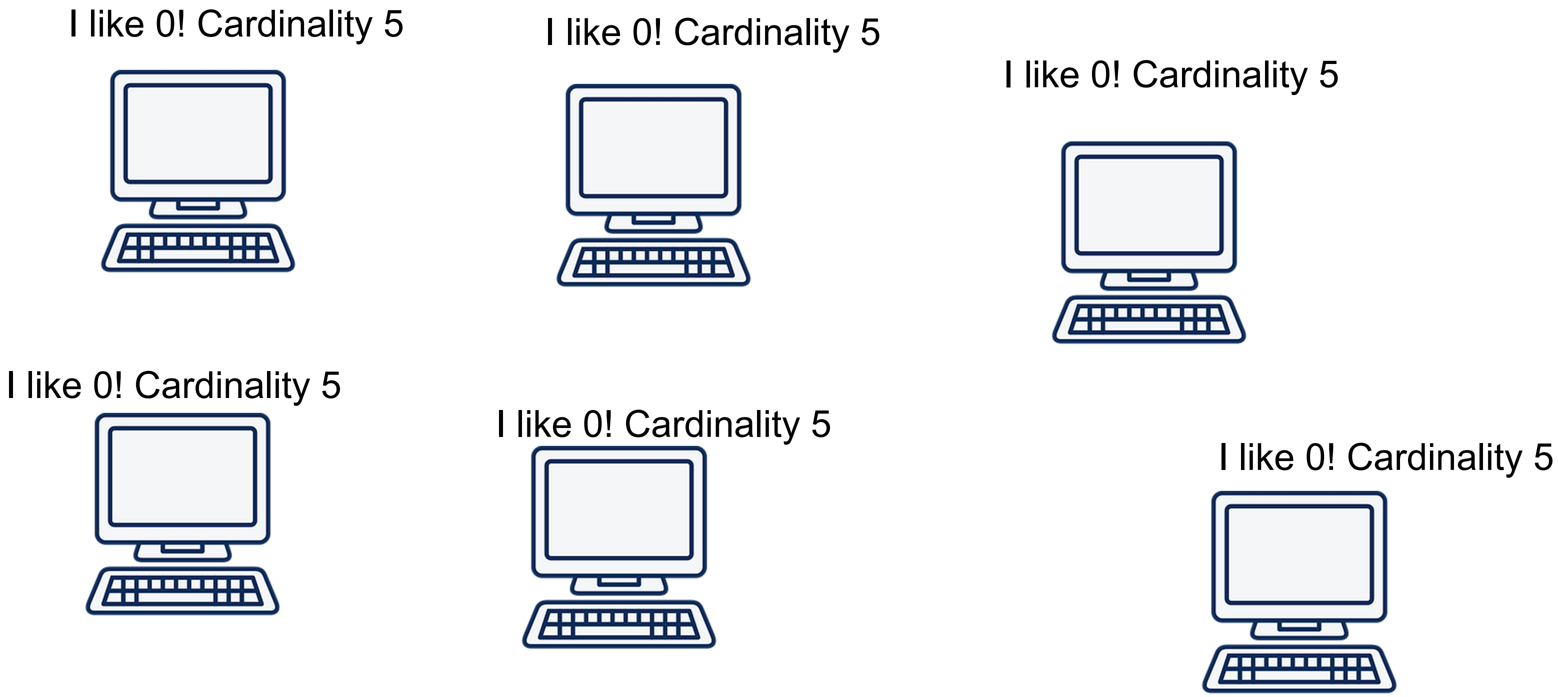
Consensus with Probability 1



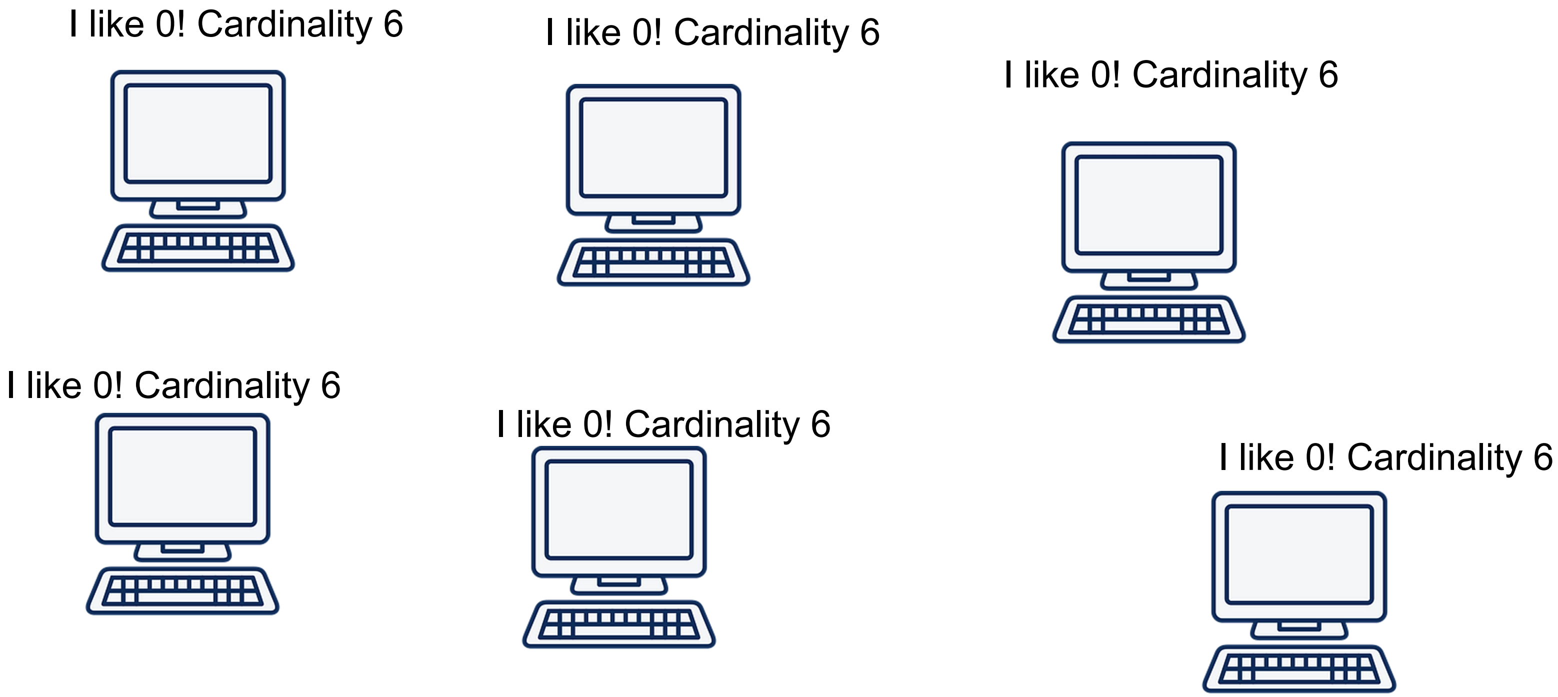
Consensus with Probability 1



Consensus with Probability 1



Consensus with Probability 1



Wrap Up Discussion; FLP

- Test your understanding: what is the difference between a univalent and bivalent state?
- But what does “impossibility” mean in FLP?
- How can we make sure our models are accurate for the desired setting?
- What are the implications for protocols handling Byzantine faults?
- Which one of these assumptions is easiest to relax in a datacenter?

Asynchronous model	Real world
Reliable message passing, unbounded delays	Just resend until acknowledged; often have a delay model
No partitioning faults (“wait until over”)	May have to operate “during” partitioning
No clocks of any kinds	Clocks but limited sync
Crash failures, can’t detect reliably	Usually detect failures with timeout

Failure Detectors!

Motivation: OK, we know FLP impossibility asynchronously.

Can we create **minimal weakening** of model,

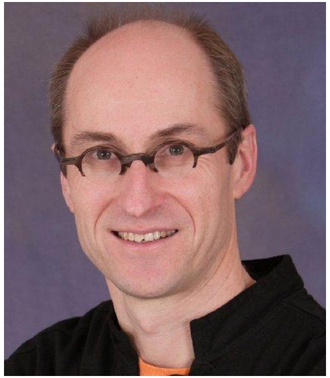
Achieve (*asynchronous** deterministic) consensus?

YES: Failure Detectors

The Weakest Failure Detector for Solving Consensus 1996

- Formalizes “failure detection service”; used by consensus as black box
- Explores types, guarantees, constructions, proofs of failure detectors

Distributed and parallel computing, machine intelligence



Fault tolerance, synchronization, databases, theory

Reliable distributed computing, security, theory applications

Motivation

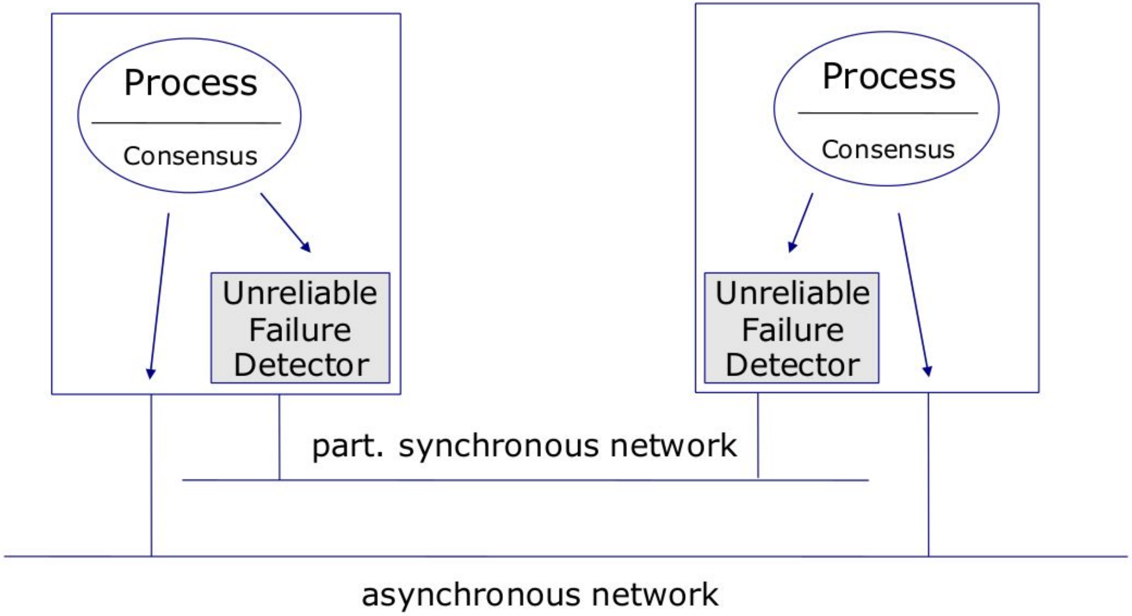


Diagram from Ken Birman's slides, '12FA

Background, Model, Assumptions

Same as last time!

Failure Detection Guarantees

- Want to achieve two properties:

- **Completeness:** failed processes eventually suspected by correct processes
- **Accuracy**: correct processes are never suspected by other correct processes
  - Can you think of a failure detector that is complete but not accurate and vice versa?

- **Incomplete or unreliable failure detectors** provide some but not perfect satisfaction of above
- **Self test**: How to implement perfect FD in synchronous model? Asynchronous model? What about the weakest imaginable FD?

Failure Detection Guarantee Variations

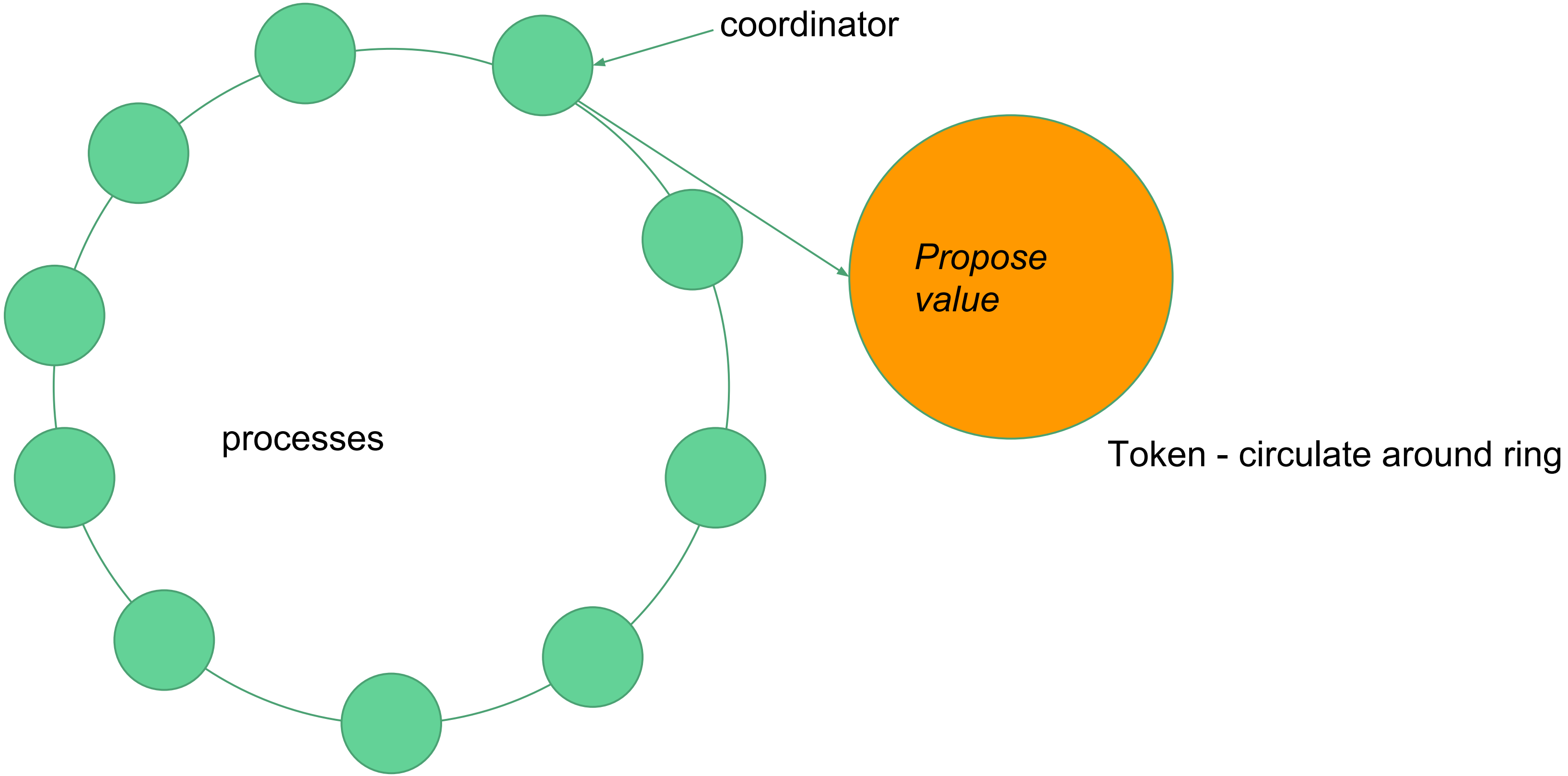
```
Strong completeness:       Eventually every process that crashes permanently suspected by every
                           correct process

Weak completeness:         Eventually every process that crashes permanently suspected by some
                           correct process

Strong accuracy:           Correct processes never suspected
Weak accuracy:             Some correct process never suspected
Eventual strong accuracy:  There is a time after which strong accuracy holds
Eventual weak accuracy:    There is a time after which weak accuracy holds
```

Accuracy

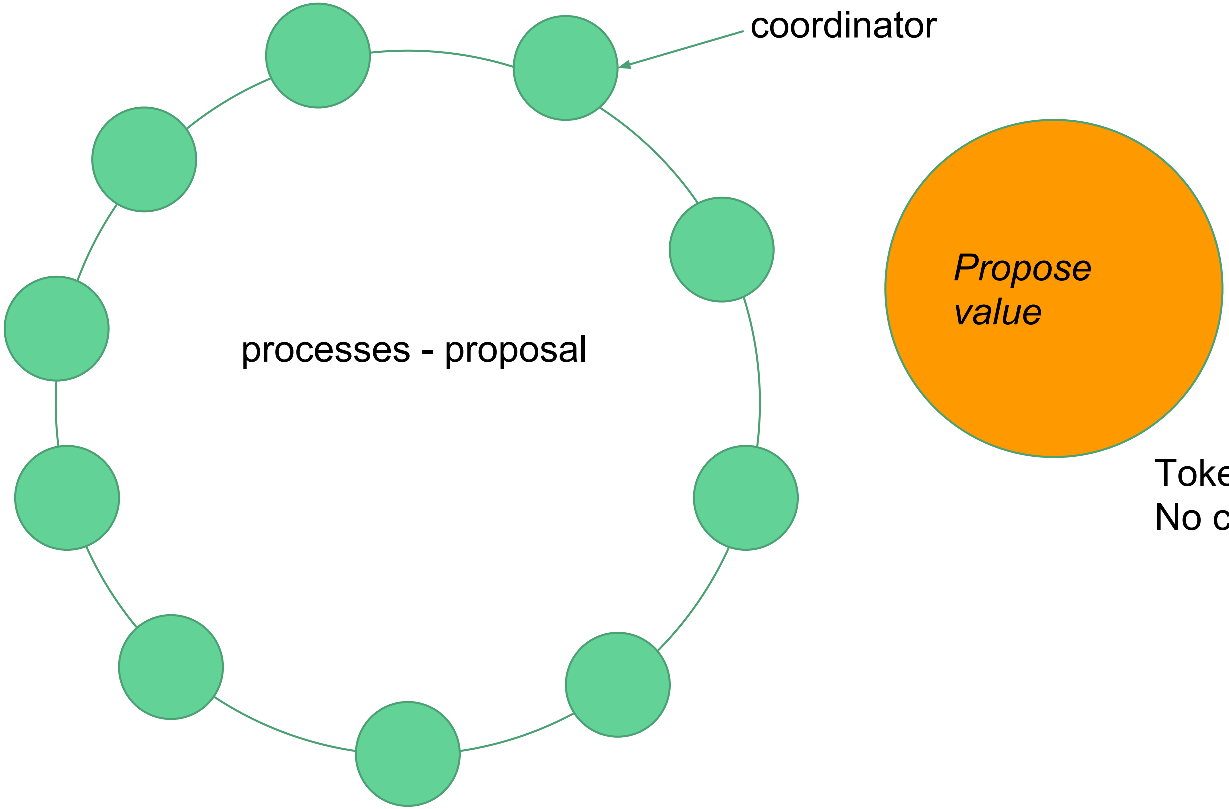
Completeness	**Strong**	**Weak**	**Eventual Strong**	**Eventual Weak**
Strong	*Perfect (P)*	*Strong (S)*	*Eventually Perfect (◊P)*	*Eventually Strong (◊S)*
Weak	*Quasi-Perfect (Q)*	*Weak (W)*	*Eventually Quasi-Perfect (◊Q)*	*Eventually Weak (◊W)*

Diagram adapted from Ken Birman's slides, '12FA

◊W for consensus!

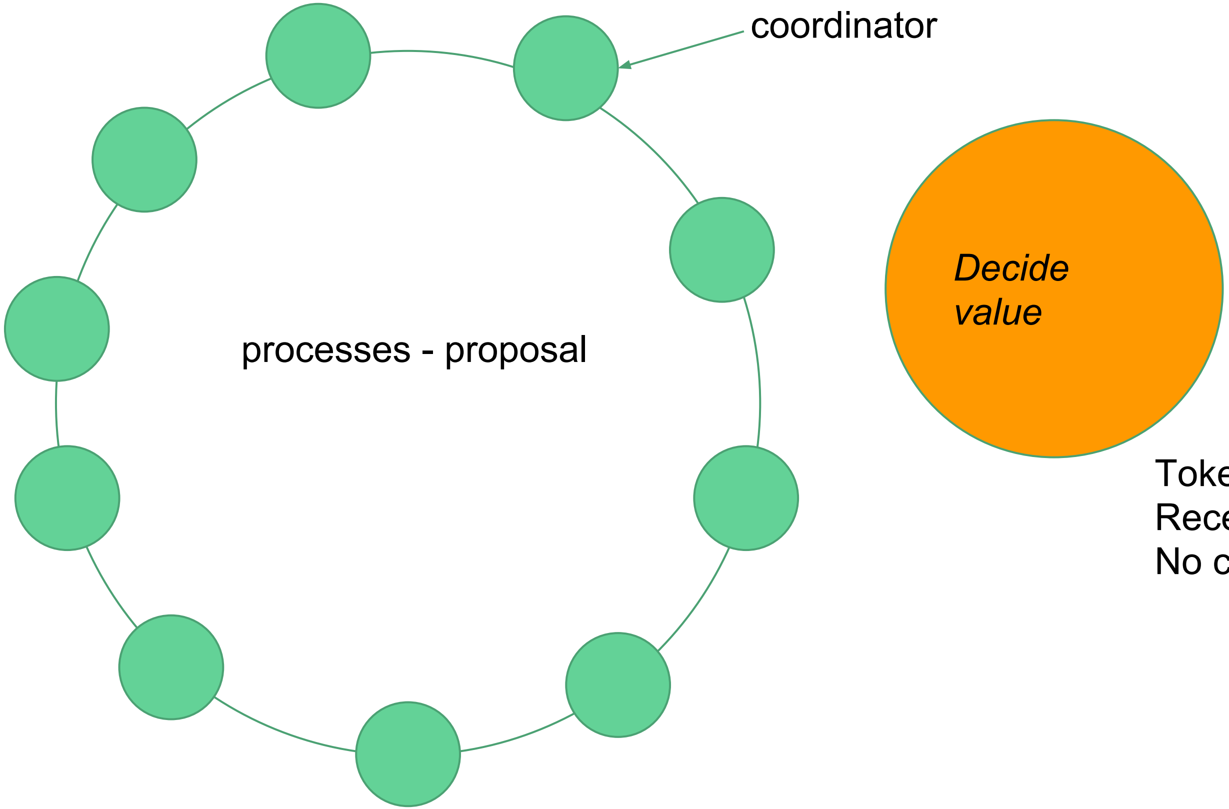


◊W for consensus!



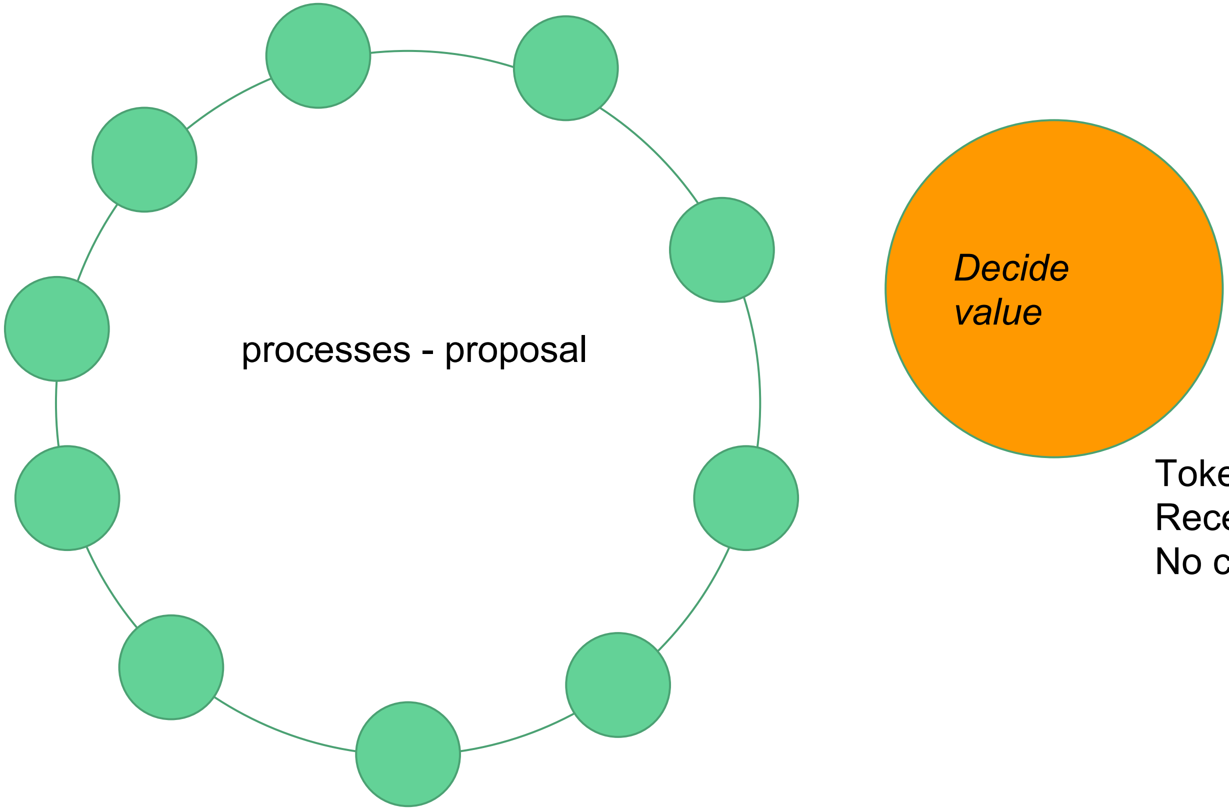
Token - circulated around ring
No change in failure detector

◊W for consensus!



Token - circulated around ring
Received by coordinator
No change in failure detector

◊W for consensus!



Token - circulated around ring
Received by all processes
No change in failure detector

Real systems and failure detectors

- Most common form of failure detection in real-systems timeout based; can you name a few?
- Usually stronger than weakest failure detector
- **No violation of FLP**; FLP applies to full system, incl. detectors

- Real systems achieve consensus with high probability
  - FLP “doesn’t matter” (can be worked around) in practice

Wrap Up Discussion; Open Problems in Consensus

- For crash fault tolerance, “trusted” setting
  - We have Paxos! We have RAFT! Can we get a simpler protocol?
- Fail-stop model; we can achieve failure detection
  - In practice, is this a workable solution?
- Byzantine faults!
  - PBFT protocol; so much easier than Paxos (look it up!)
- “trustless” setting; lots of work to be done
  - What are the unique challenges?
  - Are there model relaxations possible other than computational bounding?
  - How to identify nodes?
  - Canetti et. al 2005 impossibility result

Takeaway!

- Consensus is hard!
- Subtleties in model of consensus strongly influence results
- Make sure to choose model accurately matching reality
- Minor differences in model -> major differences in results
- The consensus ship is not yet sunk, much work to be done
- Consensus is **everywhere** in distributed systems