FLP Impossibility of Consensus

Yan Ji Oct 26, 2017

Slides inspired by Lorenzo Alvisi (CS5414 FA16) slides and Philip Daian (CS6410 FA16) slides

I think you ought to know I'm feeling very depressed.

I think you ought to know I'm feeling very depressed.

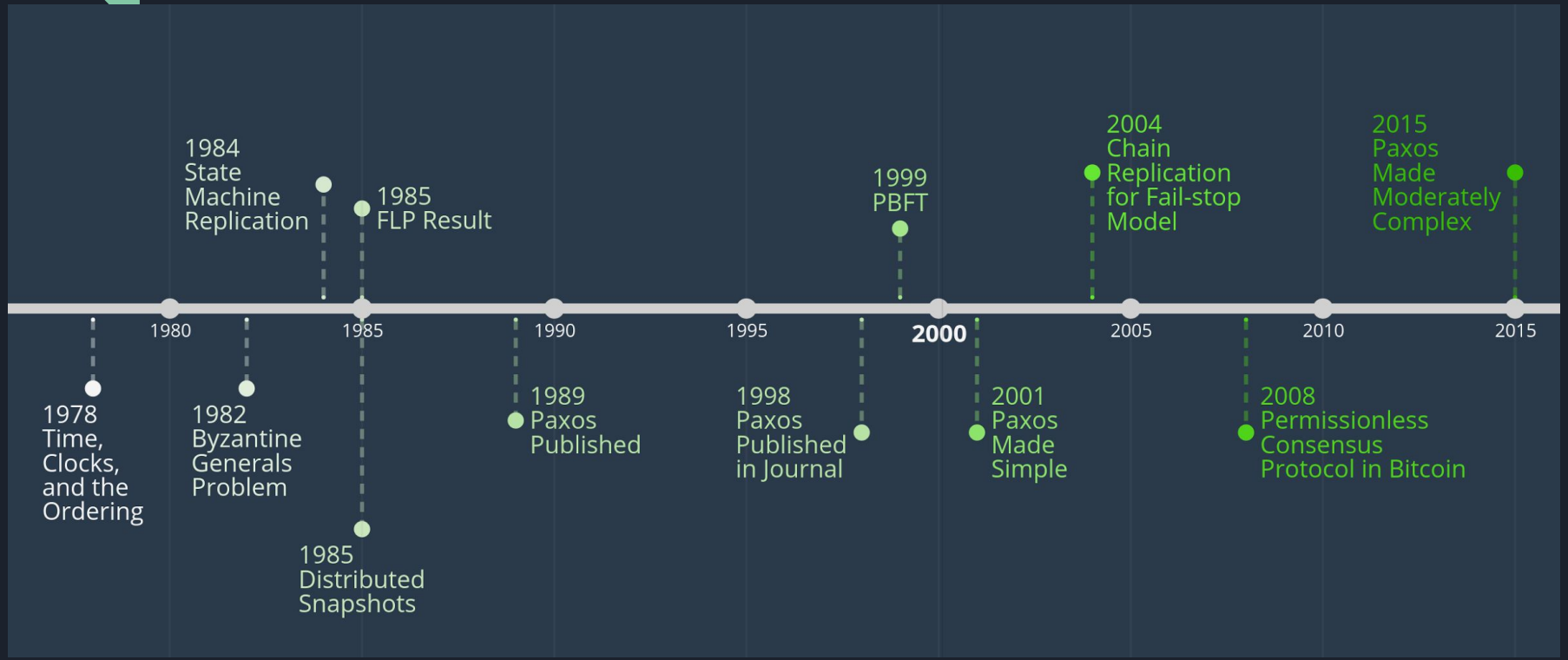
I have a million ideas, but, they all point to certain death…

FLP Impossibility of Consensus

Yan Ji Oct 26, 2017

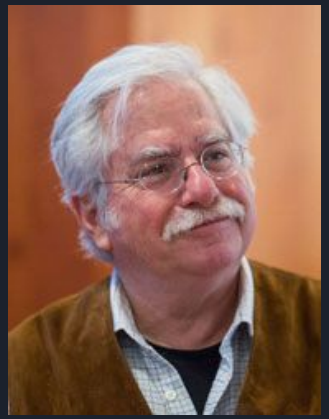
Slides inspired by Lorenzo Alvisi (CS5414 FA16) slides and Philip Daian (CS6410 FA16) slides

Timeline


1978
Time, Clocks, and the Ordering
1982
Byzantine Generals Problem
1984
State Machine Replication
1985
Distributed Snapshots
1985
FLP Result
1989
Paxos Published
1998
Paxos Published in Journal
1999
PBFT
2001
Paxos Made Simple
2004
Chain Replication for Fail-stop Model
2008
Permissionless Consensus Protocol in Bitcoin
2015
Paxos Made Moderately Complex
1980
1985
1990
1995
2000
2005
2010
2015

Impossibility of distributed consensus with one faulty process (1985)

2001 Dijkstra prize for the most influential paper in distributed computing



Michael Fischer, Yale University

Distributed computing, Cryptography

Nancy Lynch, MIT

Distributed computing theory: Algorithms and lower bounds, Modeling and verification, Wireless networks, Biological algorithms

Mike Paterson, University of Warwick

Algorithms, Complexity

FLP Result

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

The Intuition

Considering how Paxos works, what is the most difficult part for reaching consensus in an asynchronous distributed system?

When does liveness fail in Paxos

1. P1 receives promises for n1

When does liveness fail in Paxos

1. P1 receives promises for n1
2. P2 receives promises for n2 > n1
3. P1 sends proposal numbered n1, rejected

When does liveness fail in Paxos

1. P1 receives promises for n1
2. P2 receives promises for n2 > n1
3. P1 sends proposal numbered n1, rejected
4. P1 receives promises for n1′ > n2
5. P2 sends proposal numbered n2, rejected

When does liveness fail in Paxos

1. P1 receives promises for n1
2. P2 receives promises for n2 > n1
3. P1 sends proposal numbered n1, rejected
4. P1 receives promises for n1′ > n2
5. P2 sends proposal numbered n2, rejected
6. P1 receives promises for n2′ > n1’
7. P1 sends proposal numbered n1′ , rejected

When does liveness fail in Paxos

1. P1 receives promises for n1
2. P2 receives promises for n2 > n1
3. P1 sends proposal numbered n1, rejected
4. P1 receives promises for n1′ > n2
5. P2 sends proposal numbered n2, rejected
6. P1 receives promises for n2′ > n1’
7. P1 sends proposal numbered n1′ , rejected
8. . . .

The Intuition

Considering how Paxos works, what is the most difficult part for reaching consensus in an asynchronous distributed system?

The Intuition

Considering how Paxos works, what is the most difficult part for reaching consensus in an asynchronous distributed system?

CANNOT DISTINGUISH between processes:

- Crash failure
- Slow (e.g. in processing or network message delivering)

Assumption

It is impossible to have a deterministic protocol that solves consensus in a message-passing **asynchronous** system in which at most one process may fail by crashing.

Assumption

It is impossible to have a deterministic protocol that solves consensus in a message-passing **asynchronous** system in which at most one process may fail by crashing.

- No assumptions about the relative speeds of processes or about the delay time in delivering a message
- Processes don’t have access to synchronized clocks
- Unable to detect the death of a process

Assumption

It is impossible to have a deterministic protocol that solves **consensus** in a message-passing asynchronous system in which at most one process may fail by crashing.

Assumption

It is impossible to have a deterministic protocol that solves **consensus** in a message-passing asynchronous system in which at most one process may fail by crashing.

- Termination: all non-faulty processes eventually decide on a value
- Agreement: all processes that decide do so on the same value
- Validity: the value that has been decided must have been proposed by some process

Assumption

It is impossible to have a deterministic protocol that solves **consensus** in a message-passing asynchronous system in which at most one process may fail by crashing.

For simplicity,

- Termination: all non-faulty processes eventually decide on a value in {0, 1}
- Agreement: all processes that decide do so on the same value
- Validity: both 0 and 1 are possible decision values

Assumption

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by **crashing**.

Assumption

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by **crashing**.

No Byzantine, no fail-stop, just CRASH.

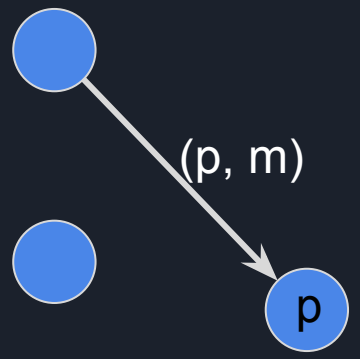
All processes follow the protocol except for that at most one might crash.

Model

It is impossible to have a deterministic protocol that solves consensus in a **message-passing** asynchronous system in which at most one process may fail by crashing.

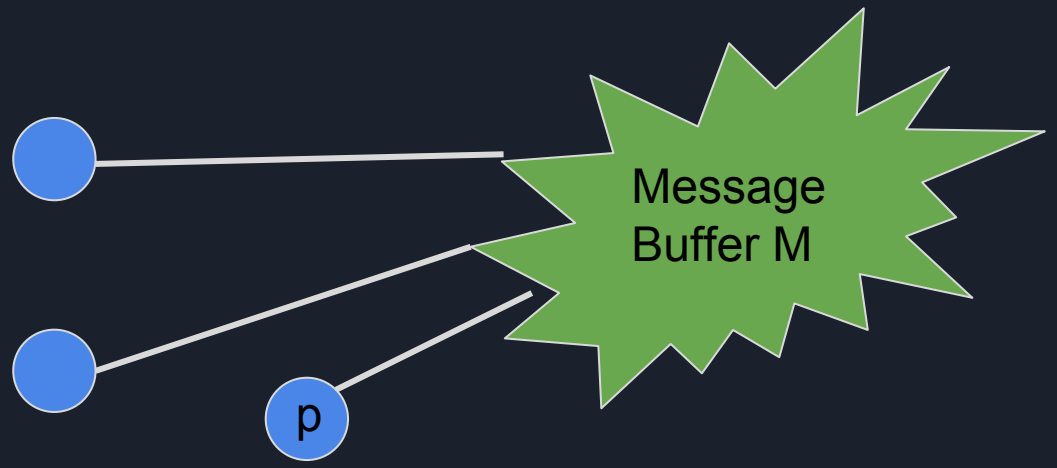
Model

It is impossible to have a deterministic protocol that solves consensus in a **message-passing** asynchronous system in which at most one process may fail by crashing.



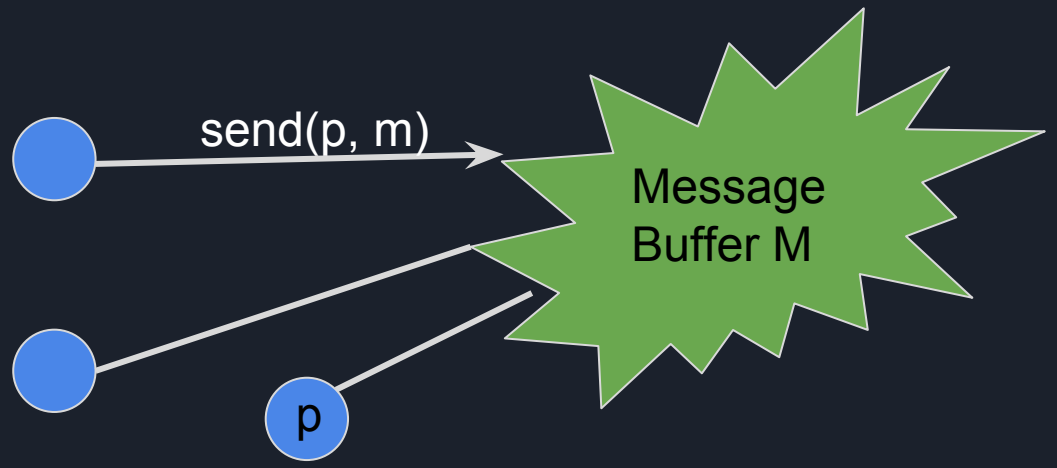
Model

It is impossible to have a deterministic protocol that solves consensus in a **message-passing** asynchronous system in which at most one process may fail by crashing.



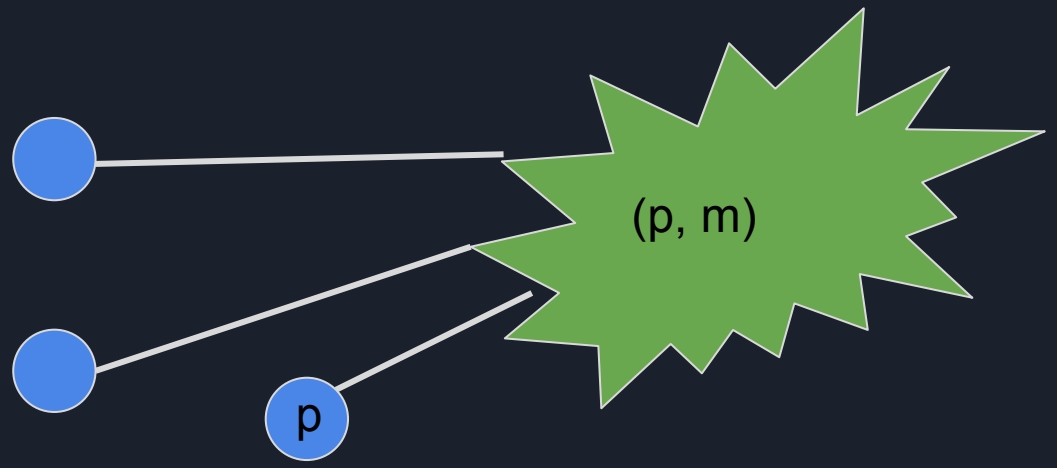
Model

It is impossible to have a deterministic protocol that solves consensus in a **message-passing** asynchronous system in which at most one process may fail by crashing.



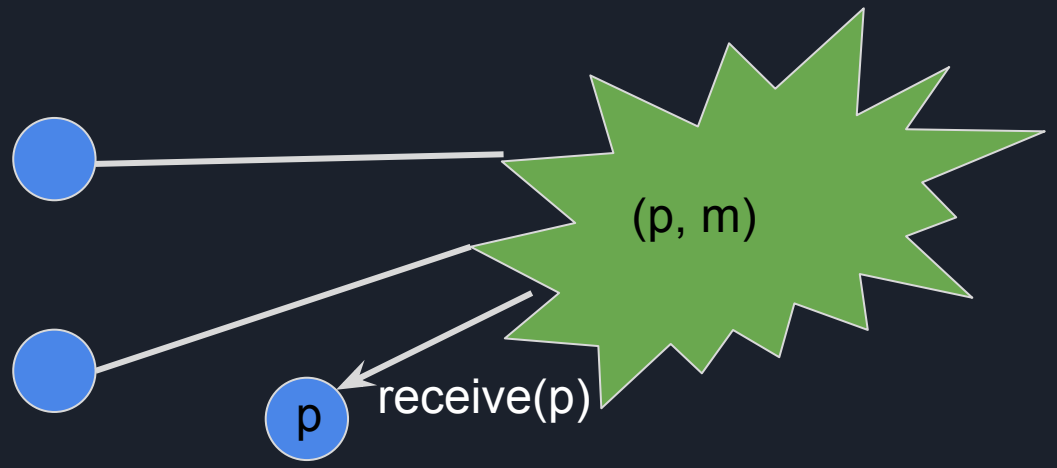
Model

It is impossible to have a deterministic protocol that solves consensus in a **message-passing** asynchronous system in which at most one process may fail by crashing.



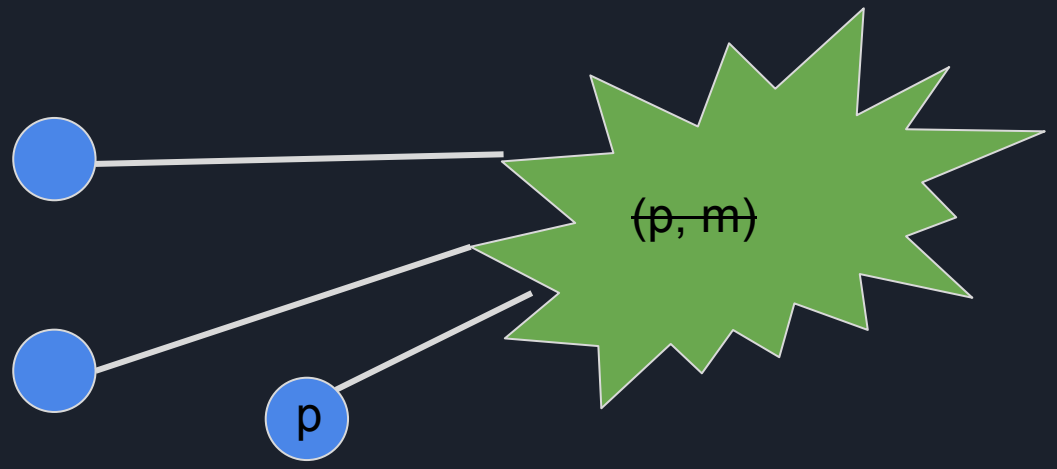
Model

It is impossible to have a deterministic protocol that solves consensus in a **message-passing** asynchronous system in which at most one process may fail by crashing.



Model

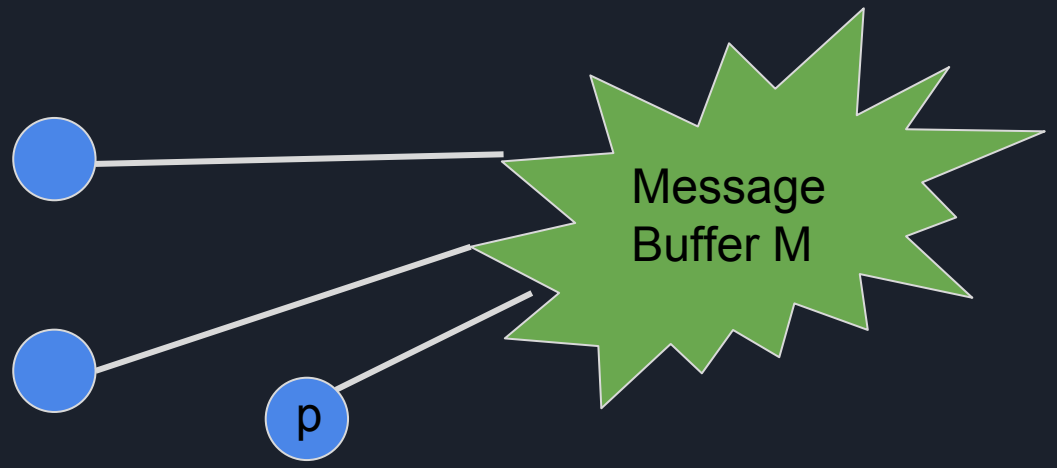
It is impossible to have a deterministic protocol that solves consensus in a **message-passing** asynchronous system in which at most one process may fail by crashing.



Deal with m

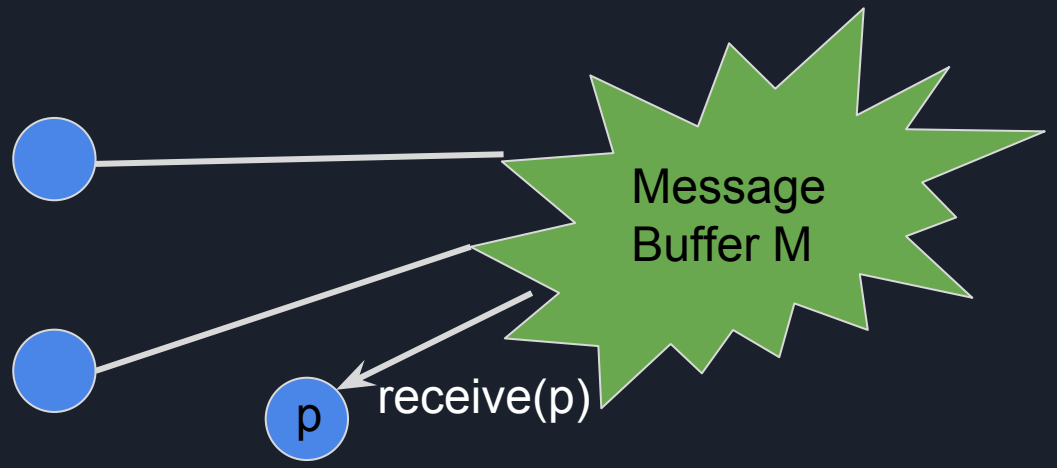
Model

It is impossible to have a deterministic protocol that solves consensus in a **message-passing** asynchronous system in which at most one process may fail by crashing.



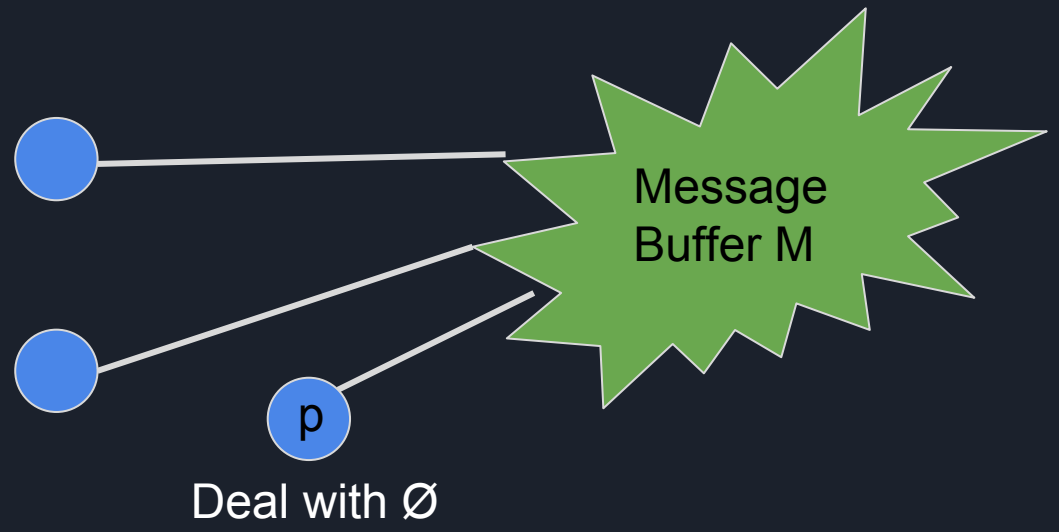
Model

It is impossible to have a deterministic protocol that solves consensus in a **message-passing** asynchronous system in which at most one process may fail by crashing.



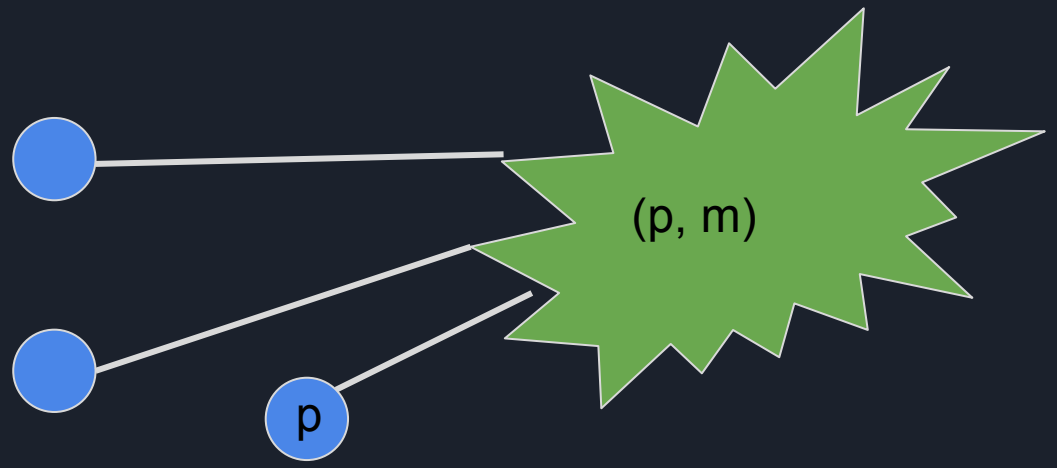
Model

It is impossible to have a deterministic protocol that solves consensus in a **message-passing** asynchronous system in which at most one process may fail by crashing.



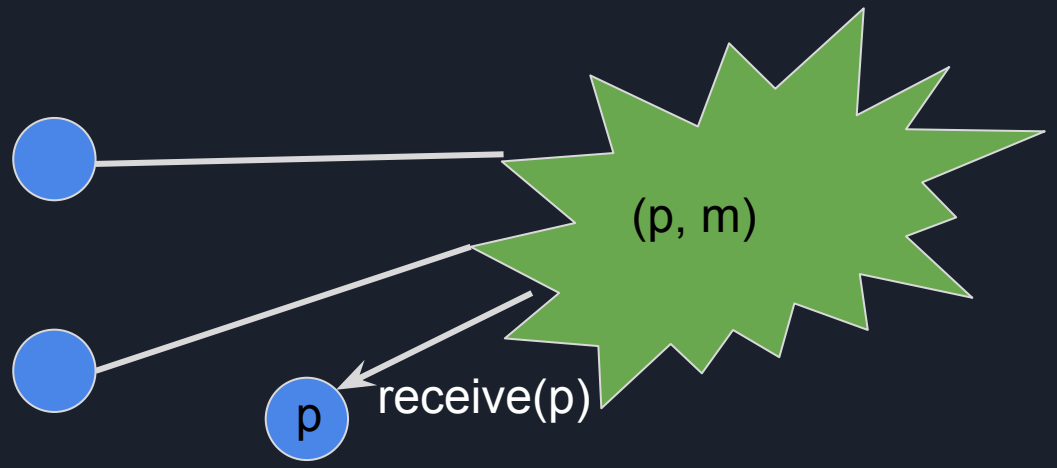
Model

It is impossible to have a deterministic protocol that solves consensus in a **message-passing** asynchronous system in which at most one process may fail by crashing.



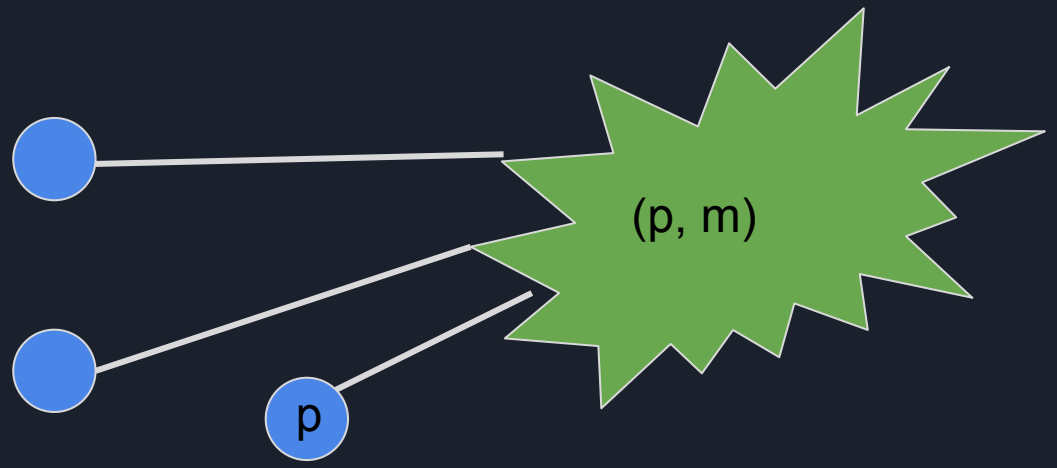
Model

It is impossible to have a deterministic protocol that solves consensus in a **message-passing** asynchronous system in which at most one process may fail by crashing.



Model

It is impossible to have a deterministic protocol that solves consensus in a **message-passing** asynchronous system in which at most one process may fail by crashing.

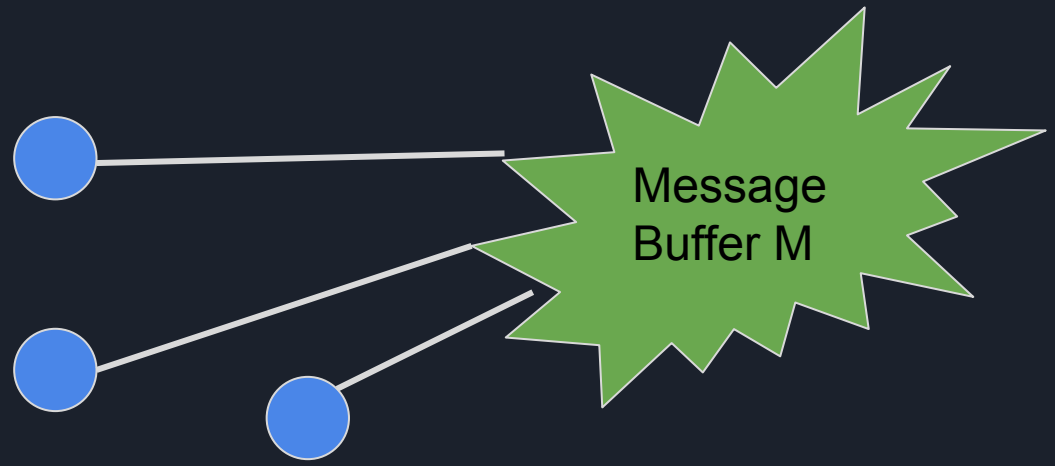


Deal with Ø

Model

It is impossible to have a deterministic protocol that solves consensus in a **message-passing** asynchronous system in which at most one process may fail by crashing.

- All messages are delivered correctly and exactly once
- The message buffer acts nondeterministically
- Inserted with arbitrary number of empty messages



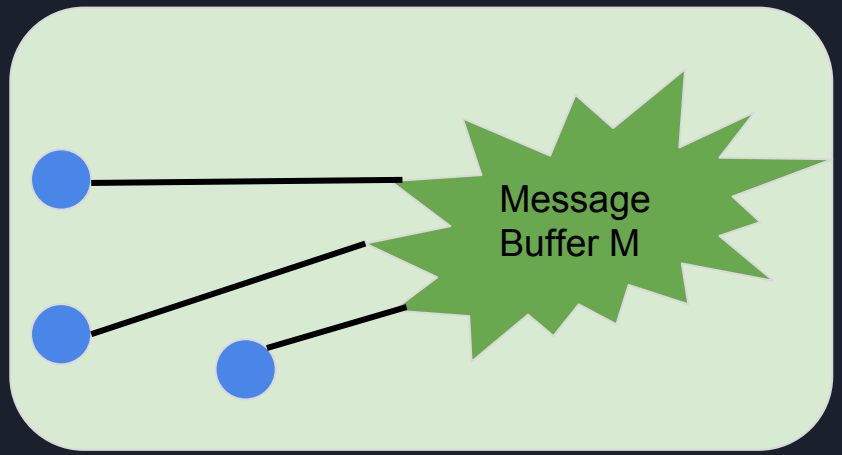
Model

It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

Model

It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

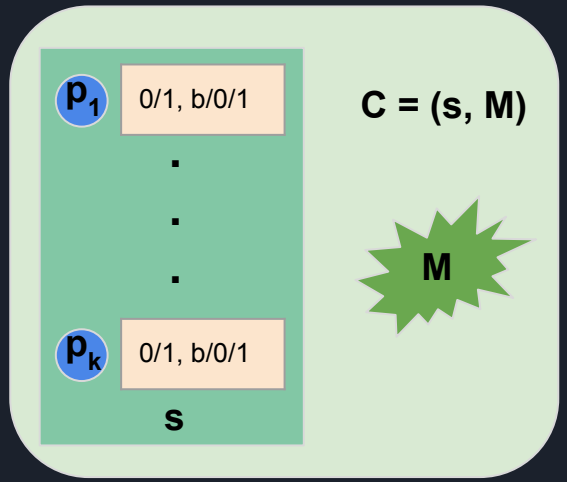
- Configuration: C = (s, M)
  - M: message buffer
  - s: internal states of processes
    - Input register
    - Output register
    - Internal storage
    - Program counter



Model

It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

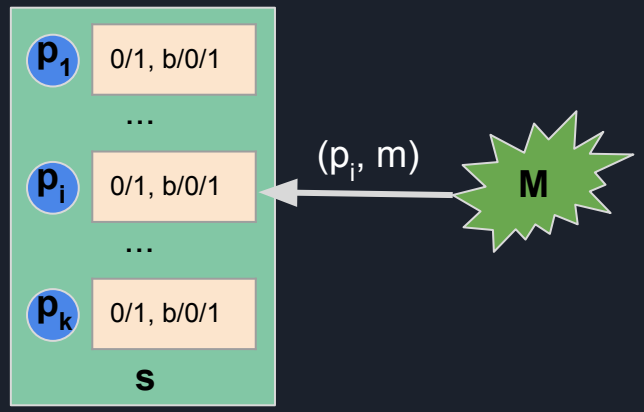
- Configuration: C = (s, M)
  - M: message buffer
  - s: internal states of processes
    - Input register, value in {0, 1}
    - Output register, value in {b, 0, 1}
    - Internal storage
    - Program counter



Model

It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

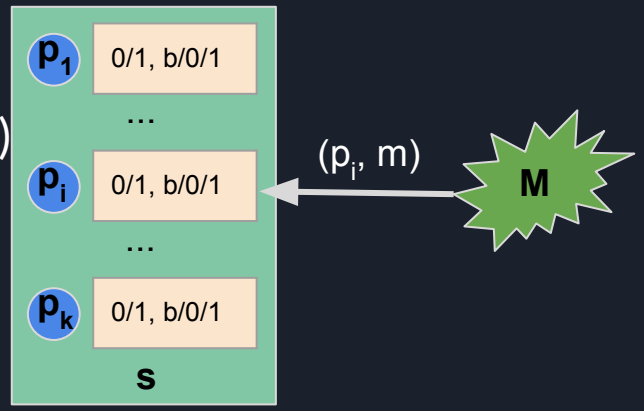
- Event: e = (p, m)

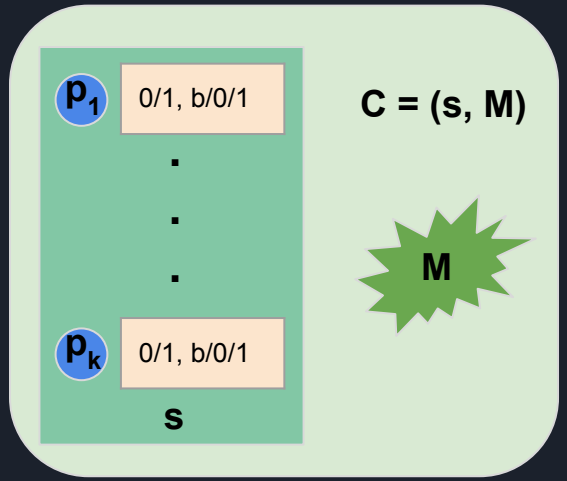


Model

It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- Step: C'= e(C) = (s', M')

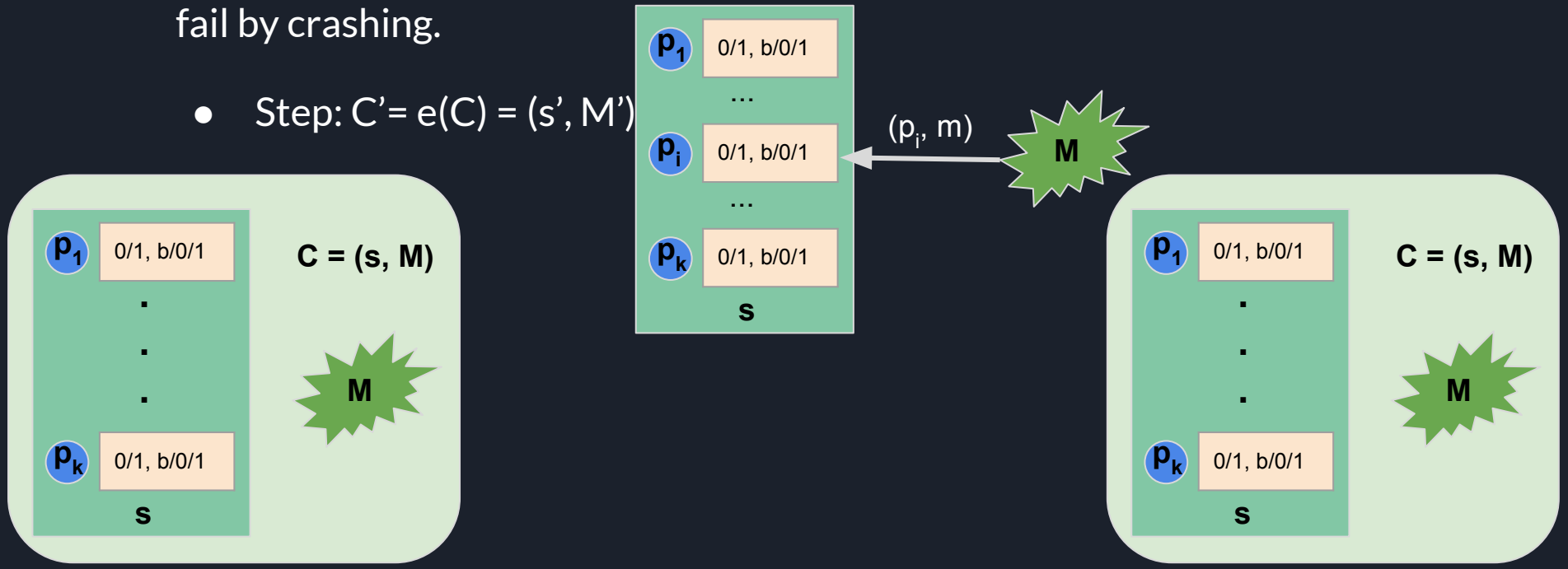




Model

It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

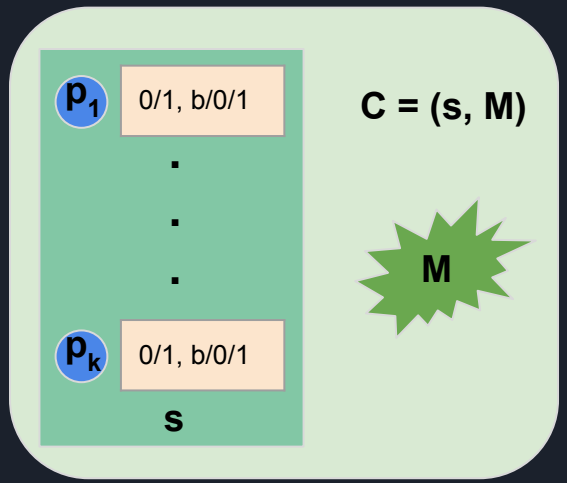
- Step: C'= e(C) = (s', M')

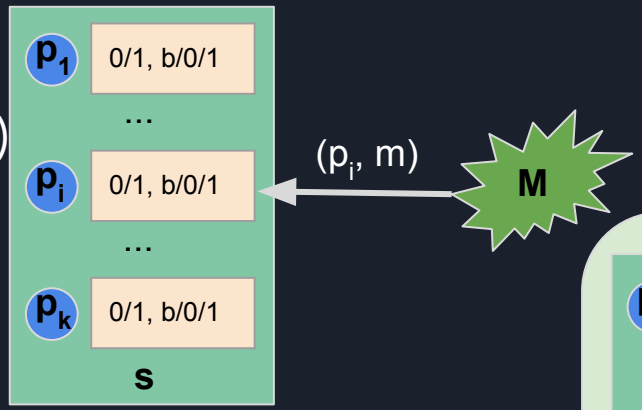


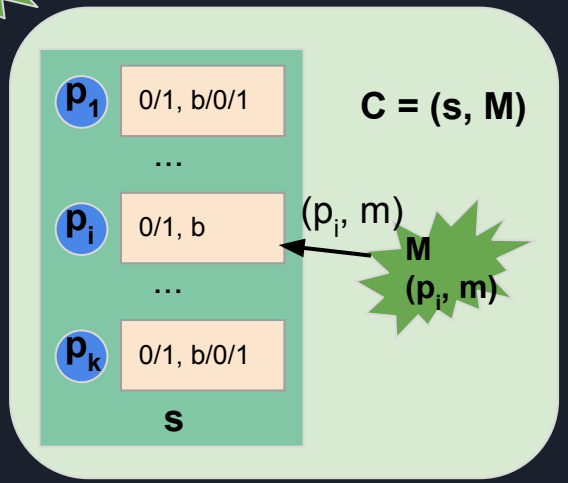
Model

It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- Step: C'= e(C) = (s', M')



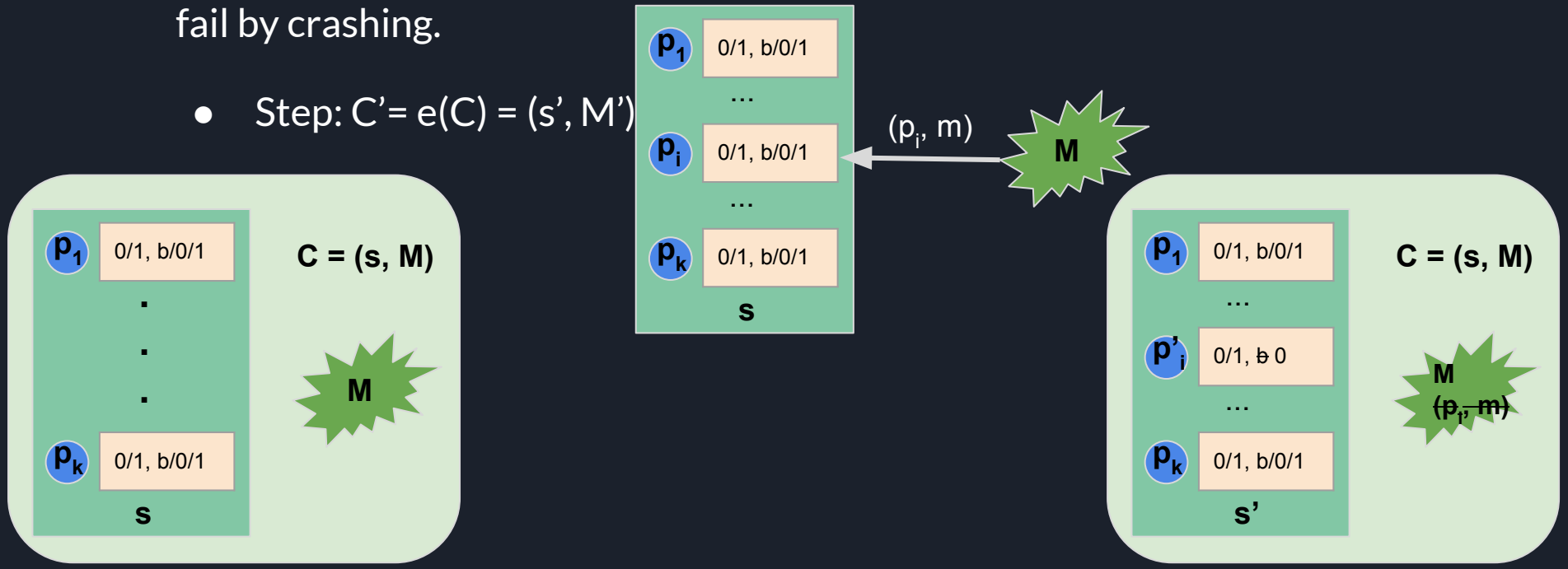




Model

It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

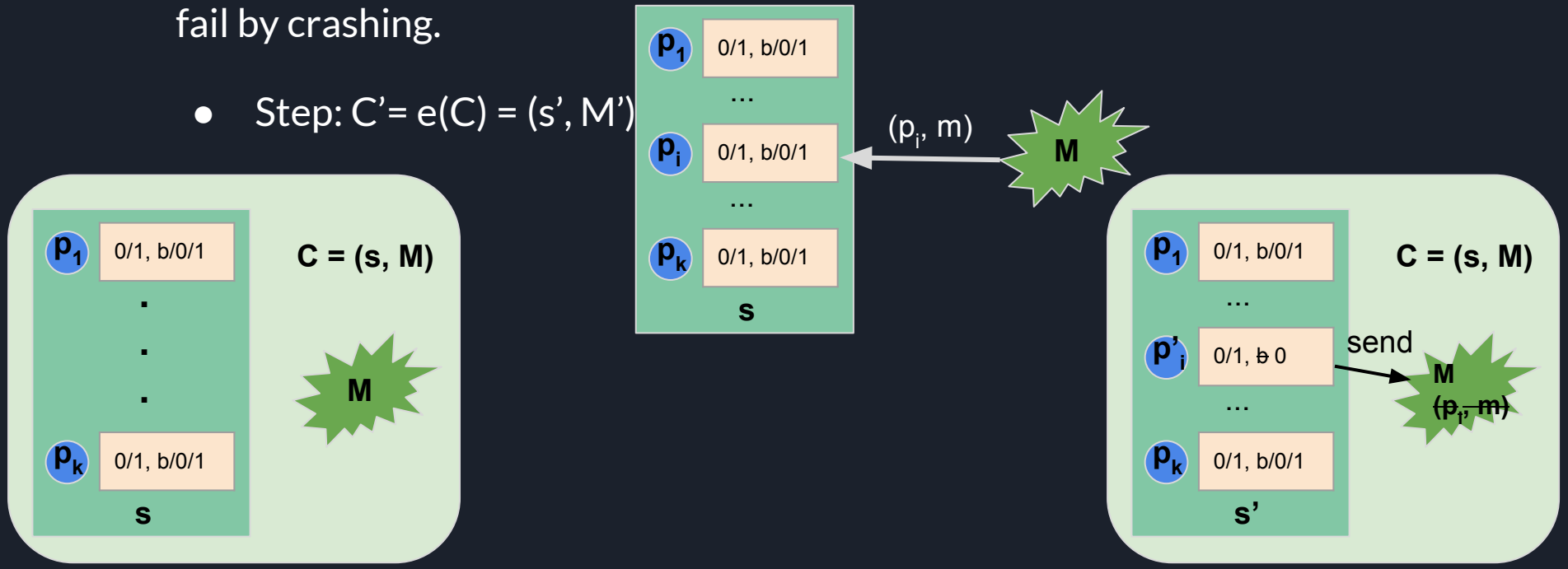
- Step: C'= e(C) = (s', M')



Model

It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

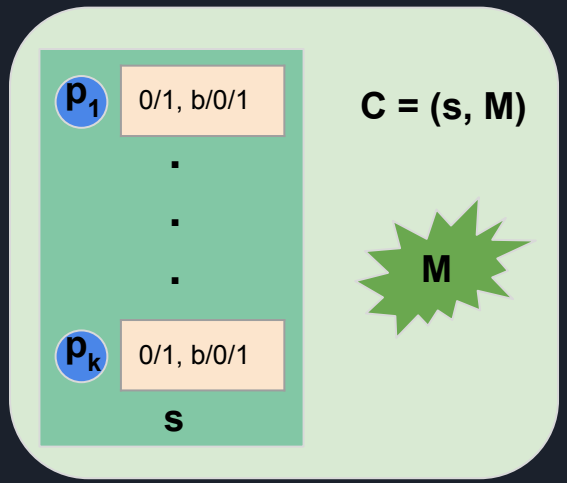
- Step: C'= e(C) = (s', M')

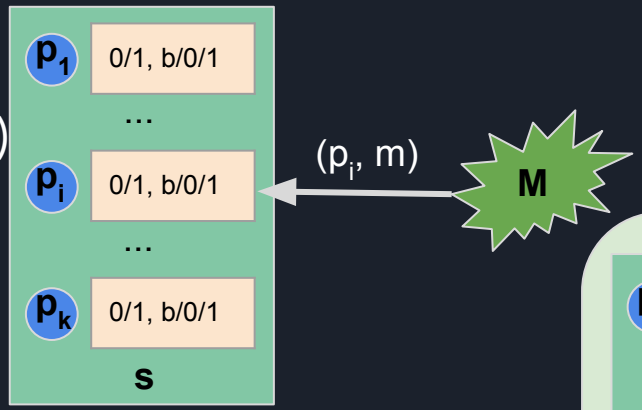


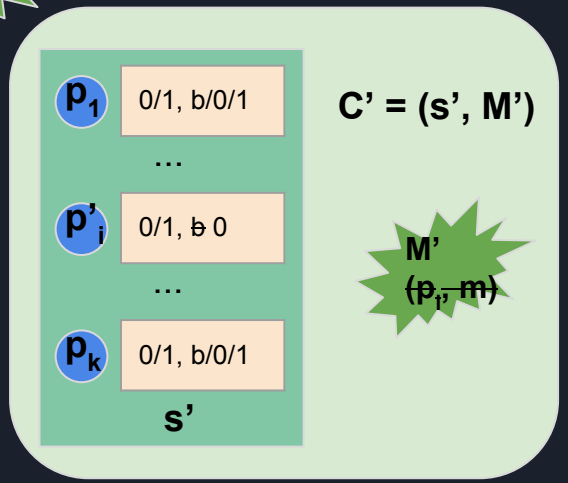
Model

It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- Step: C'= e(C) = (s', M')



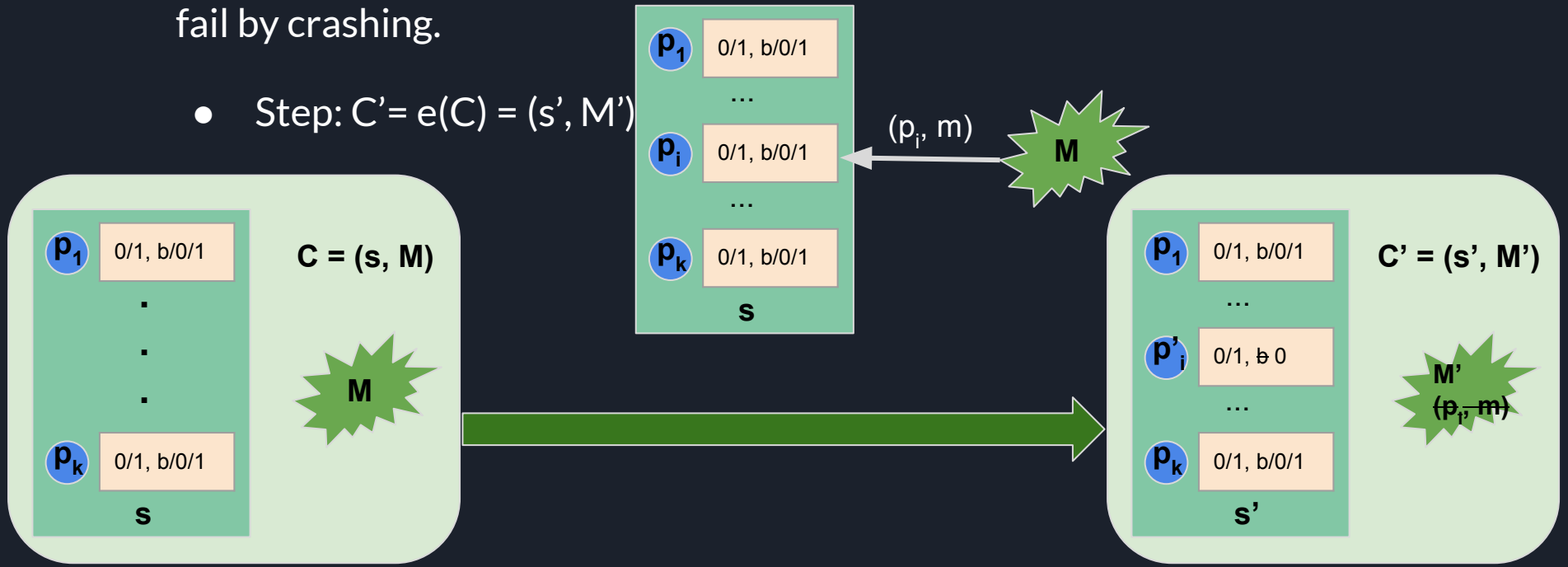




Model

It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- Step: C'= e(C) = (s', M')



Proof

It is **impossible** to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

How to prove impossibility?

Proof

It is **impossible** to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

How to prove impossibility?

Assume to the contrary that there exists a consensus protocol P such that…

How to define P?

More terms

- A schedule S of P is a finite or infinite sequence of events $(e_1, e_2, \ldots, e_k)$of P, $S(C) = e_k(\ldots(e_2(e_1(C)))\ldots)$

More terms

- A schedule S of P is a finite or infinite sequence of events $(e_1, e_2, \ldots, e_k)$of P, $S(C) = e_k(\ldots(e_2(e_1(C)))\ldots)$
- A run of P is a sequence of steps associating a schedule S, in other words, a run is a pair of a configuration C and a schedule S, written as (C, S)

More terms

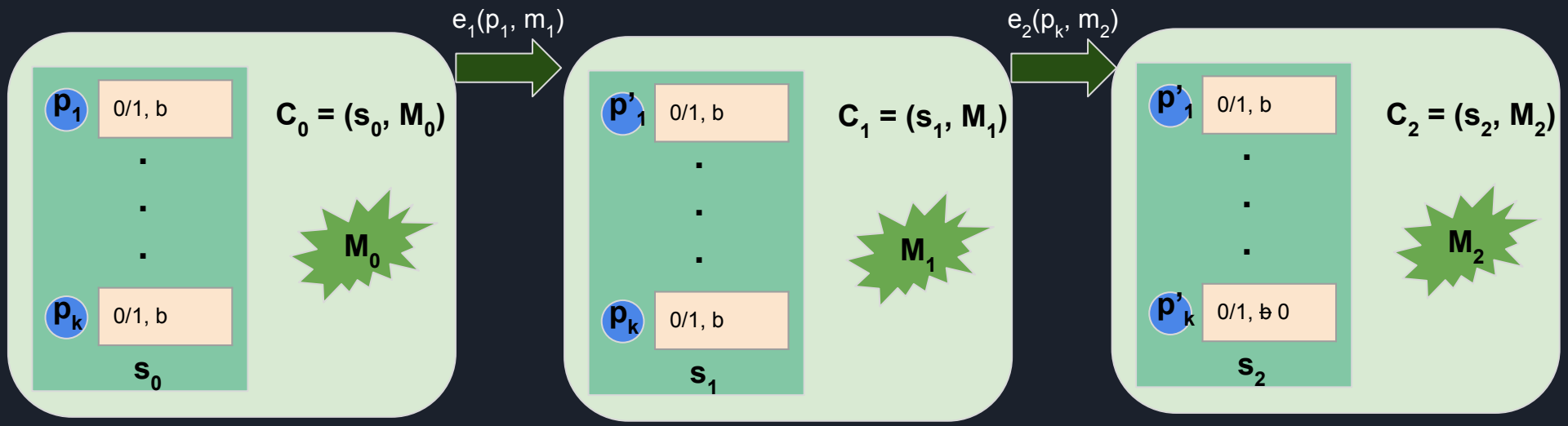
- A configuration C’ is reachable from a configuration C if there exist a schedule S such that C’ = S(C)

More terms

- A configuration C’ is reachable from a configuration C if there exist a schedule S such that C’ = S(C)
- A configuration C′ is accessible from an initial configuration C_0 if C’ is reachable from C_0

More terms

- A configuration C' is reachable from a configuration C if there exist a schedule S such that C' = S(C)
- A configuration C′ is accessible from an initial configuration C_0 if C' is reachable from C_0



More terms

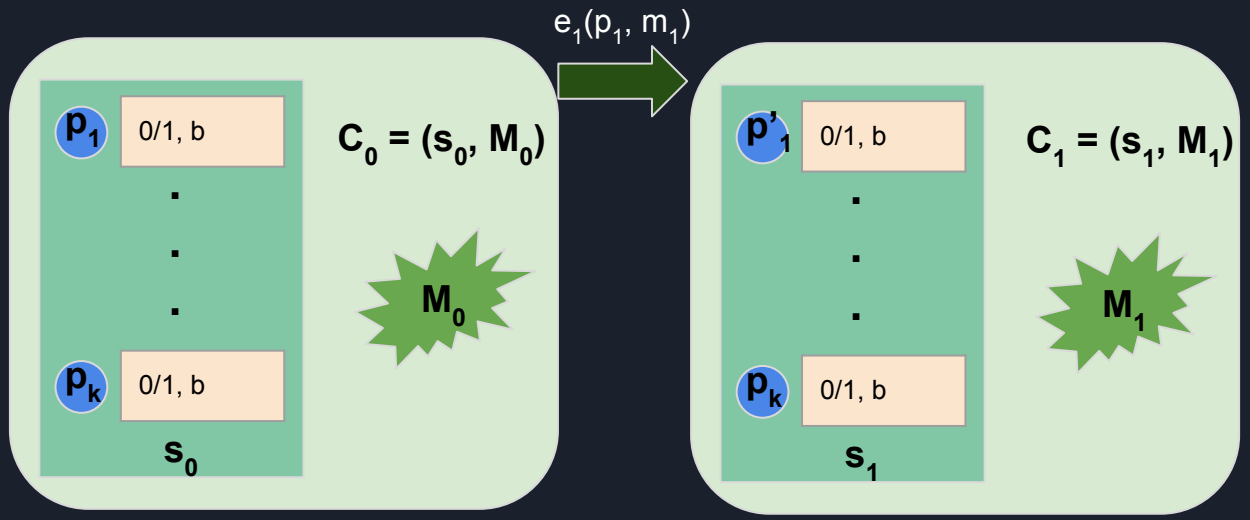
- A configuration C has decision value v if some process p is in a decision state with output=v, which is “write-once”/irreversible

More terms

- A configuration C has decision value v if some process p is in a decision state with output=v, which is “write-once”/irreversible
- A run is a deciding run if some process reaches a decision state.

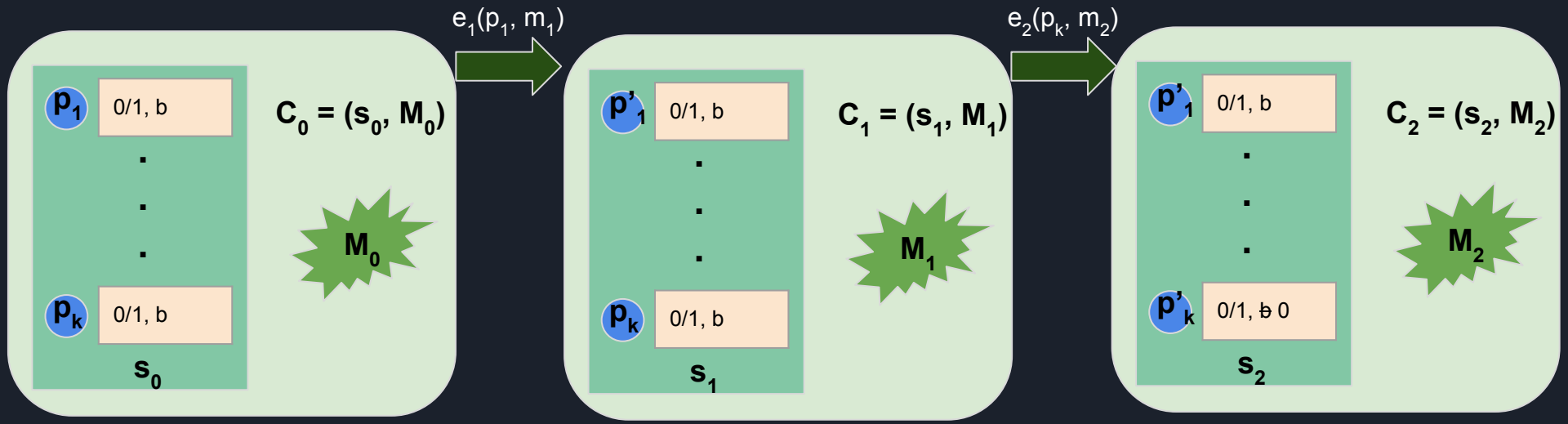
More terms

- A configuration C has decision value v if some process p is in a decision state with output=v, which is “write-once”/irreversible
- A run is a deciding run if some process reaches a decision state.



More terms

- A configuration C has decision value v if some process p is in a decision state with output=v, which is “write-once”/irreversible
- A run is a deciding run if some process reaches a decision state.



More terms

- A consensus protocol P is partially correct if:
  - No accessible configuration has more than one decision value (agreement)
  - For each v in {0, 1}, some accessible configuration has decision value v (validity)
- A run is admissible if every process, except possibly one (faulty process), takes infinitely many steps in S

Assume to the contrary that there exists P such that

- P is partially correct
  - Agreement + Validity

Assume to the contrary that there exists P such that

- P is partially correct
  - Agreement + Validity

- Every admissible run of P is a deciding run
  - Termination

Assume to the contrary that there exists P such that

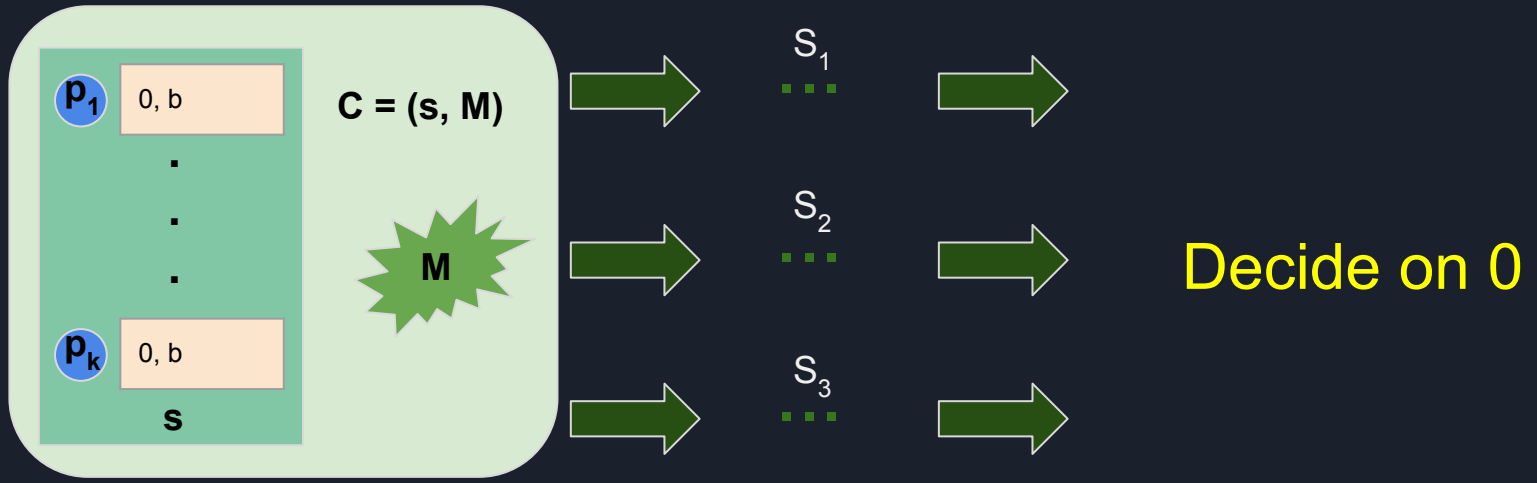
- P is partially correct
  - Agreement + Validity
- Every admissible run of P is a deciding run
  - Termination
- What kind of contradiction should possibly be like?

Categories of configurations

- Univalent, or i-valent (i in {0, 1})
  - A configuration C is univalent or i-valent if some process has decided i in C, or if all configurations accessible from C are i-valent

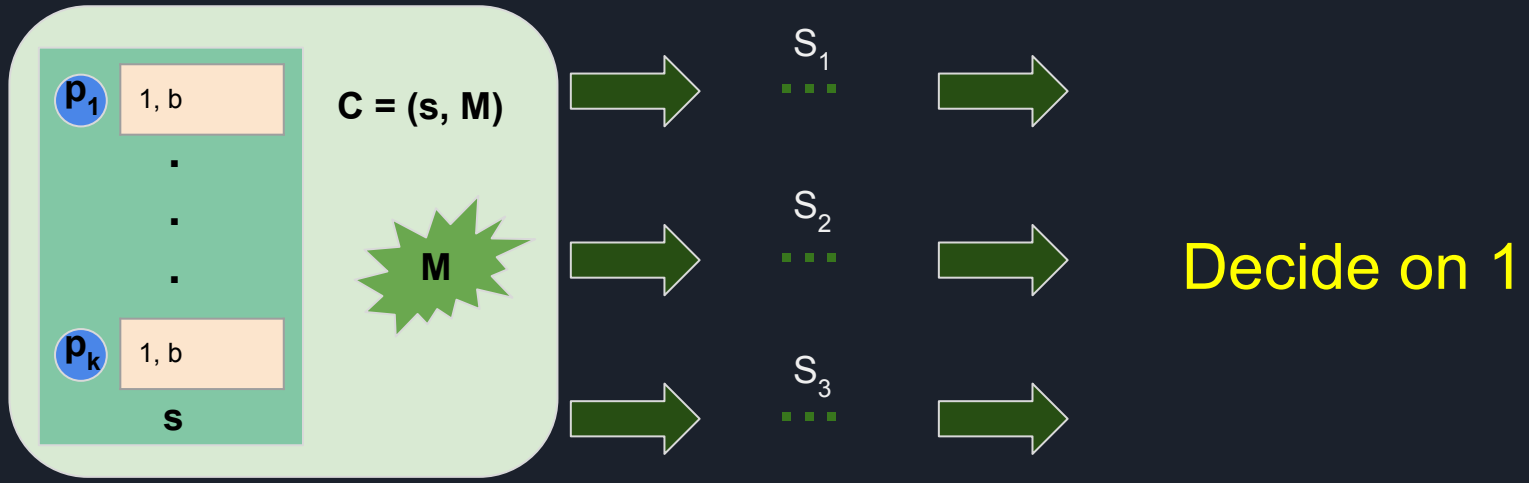
Categories of configurations

- Univalent, or i-valent (i in {0, 1})
  - A configuration C is univalent or i-valent if some process has decided i in C, or if all configurations accessible from C are i-valent



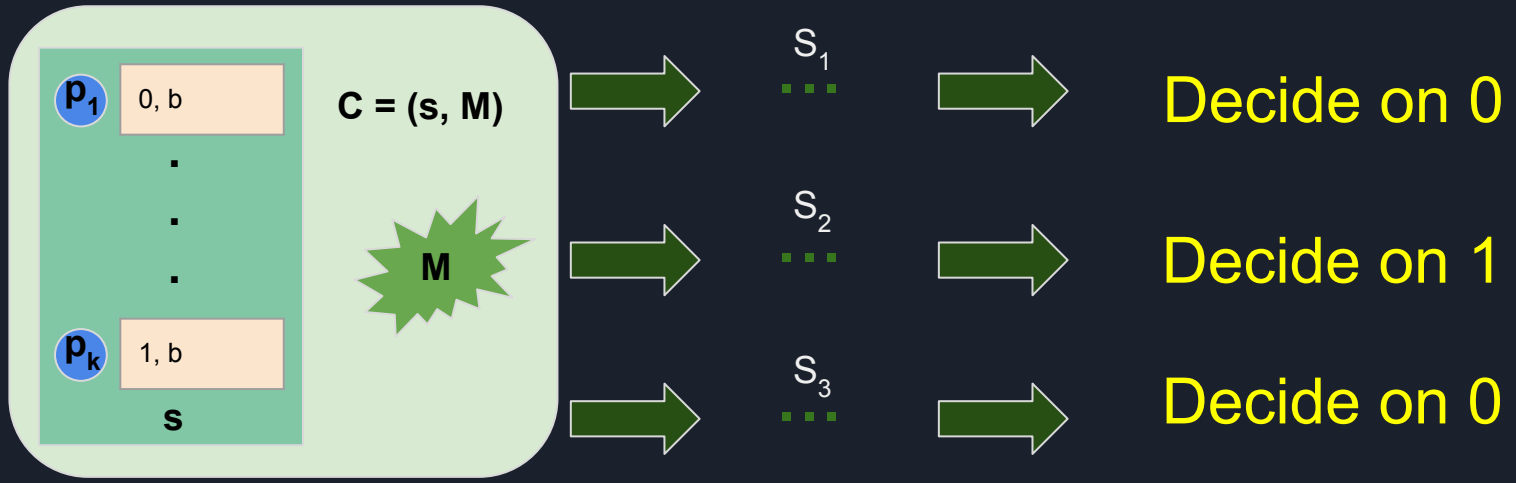
Categories of configurations

- Univalent, or i-valent (i in {0, 1})
  - A configuration C is univalent or i-valent if some process has decided i in C, or if all configurations accessible from C are i-valent



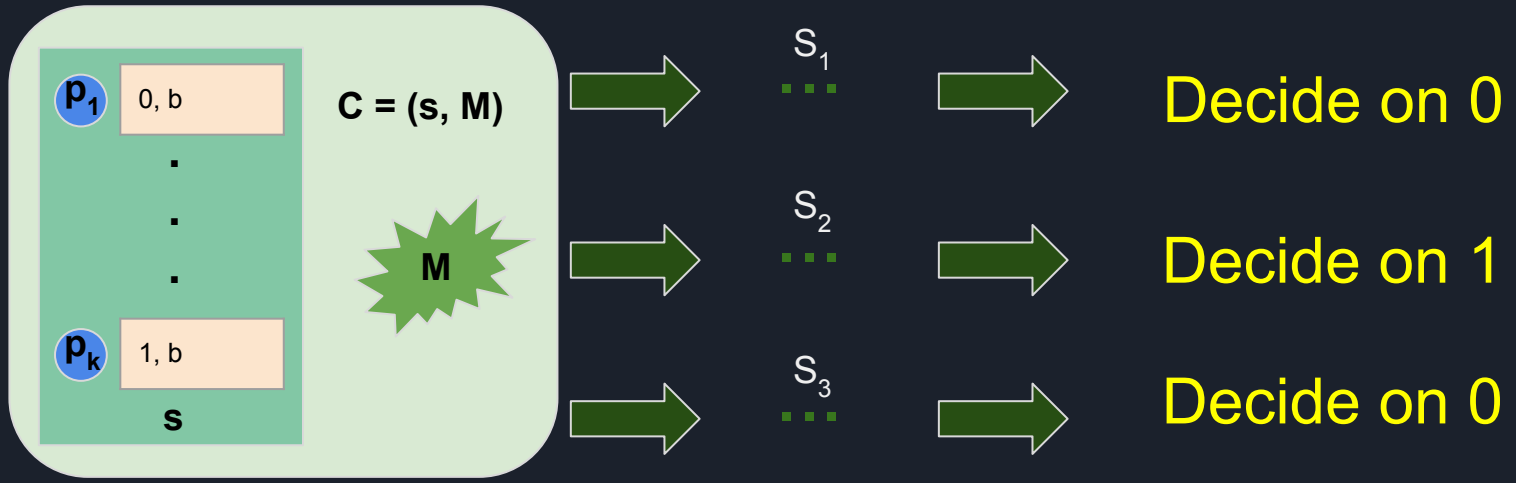
Categories of configurations

- Bivalent
  - A configuration C is bivalent if some of the configurations accessible from it are 0-valent while others are 1-valent



Categories of configurations

- Bivalent (see Bivalent, read Undeciding)
  - A configuration C is bivalent if some of the configurations accessible from it are 0-valent while others are 1-valent



What kind of contradiction should possibly be like?

INDISTINGUISHABILITY between processes:

- Crashed
- Simply slow in processing or having a terrible network condition

What kind of contradiction should possibly be like?

INDISTINGUISHABILITY between processes:

- Crashed
- Simply slow in processing or having a terrible network condition

For any protocol, there exists a configuration that is always bivalent.

What kind of contradiction should possibly be like?

INDISTINGUISHABILITY between processes:

- Crashed
- Simply slow in processing or having a terrible network condition

For any protocol, there exists a configuration that is always bivalent.

Remaining UNDECIDED in the value

What kind of contradiction should possibly be like?

INDISTINGUISHABILITY between processes:

- Crashed
- Simply slow in processing or having a terrible network condition

For any protocol, there exists a configuration that is always bivalent.

Remaining UNDECIDED in the value

Proof Outline

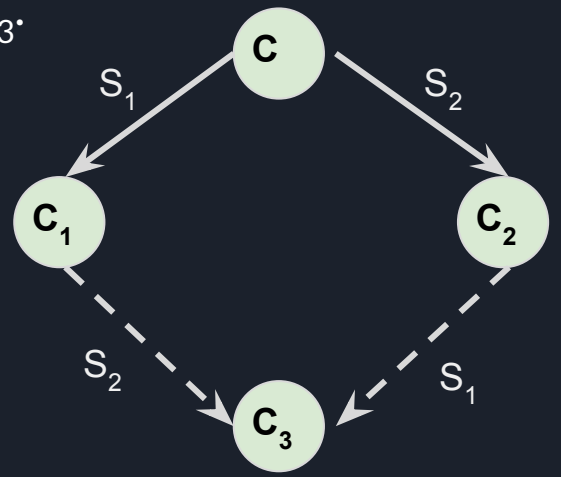
- For any protocol, there is an initial configuration that is bivalent
- Then there is another bivalent configuration reachable from it after applying some event
- And another reachable bivalent configuration
- ...
- An infinite undeciding run

Most exciting part!!!

- Lemma 1 (commutativity of schedules)
  - Suppose that from some C, the schedules S_1, S_2 lead to C_1, C_2 respectively. If the steps in S_1 and in S_2 are disjoint, then S_2 can be applied to C_1 and S_1 can be applied to C_2 and both lead to the same C_3.

Most exciting part!!!

- Lemma 1 (commutativity of schedules)
  - Suppose that from some C, the schedules S_1, S_2 lead to C_1, C_2 respectively. If the steps in S_1 and in S_2 are disjoint, then S_2 can be applied to C_1 and S_1 can be applied to C_2 and both lead to the same C_3.



Proof Outline

- For any protocol, there is an initial configuration that is bivalent
- Then there is another bivalent configuration reachable from it after applying some event
- And another reachable bivalent configuration
- …
- An infinite undeciding run

Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.

p_1	0, b	0, b	0, b	0, b	1, b	1, b	1, b	1, b
p_2	0, b	0, b	1, b	1, b	1, b	1, b	0, b	0, b
p_3	1, b	0, b	0, b	1, b	1, b	0, b	0, b	1, b

Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.

p_1	0, b	0, b	0, b	0, b	1, b	1, b	1, b	1, b
p_2	0, b	0, b	1, b	1, b	1, b	1, b	0, b	0, b
p_3	1, b	0, b	0, b	1, b	1, b	0, b	0, b	1, b
	0	0	0	0	1	1	1	1

Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.

p_1	0, b	0, b	0, b	0, b	1, b	1, b	1, b	1, b
p_2	0, b	0, b	1, b	1, b	1, b	1, b	0, b	0, b
p_3	1, b	0, b	0, b	1, b	1, b	0, b	0, b	1, b
	0	0	0	0	1	1	1	1

Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

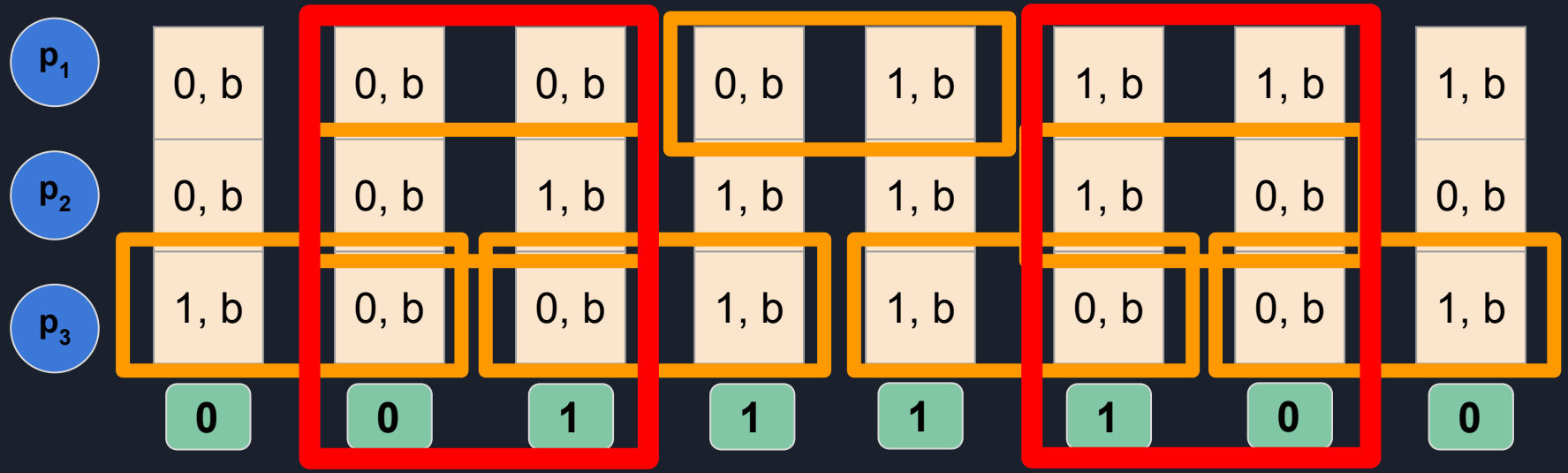
Assume all initial configurations are either 0-valent or 1-valent.

p_1	0, b	0, b	0, b	0, b	1, b	1, b	1, b	1, b
p_2	0, b	0, b	1, b	1, b	1, b	1, b	0, b	0, b
p_3	1, b	0, b	0, b	1, b	1, b	0, b	0, b	1, b
	0	0	1	1	1	1	0	0

Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

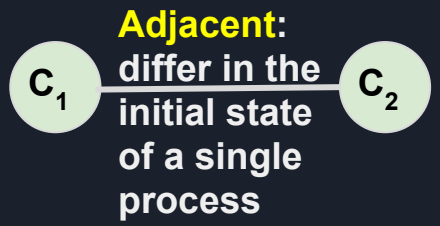
Assume all initial configurations are either 0-valent or 1-valent.



Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

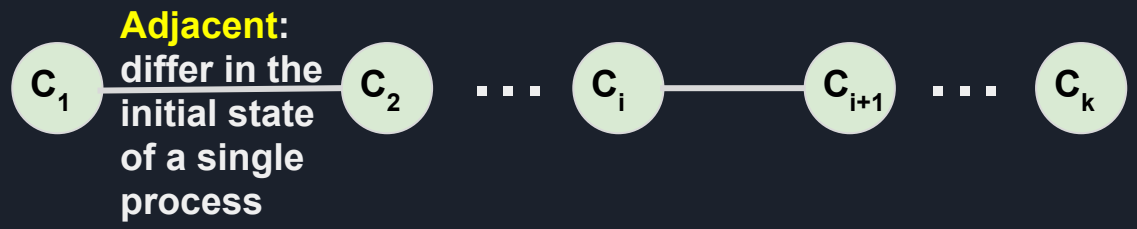
Assume all initial configurations are either 0-valent or 1-valent.



Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

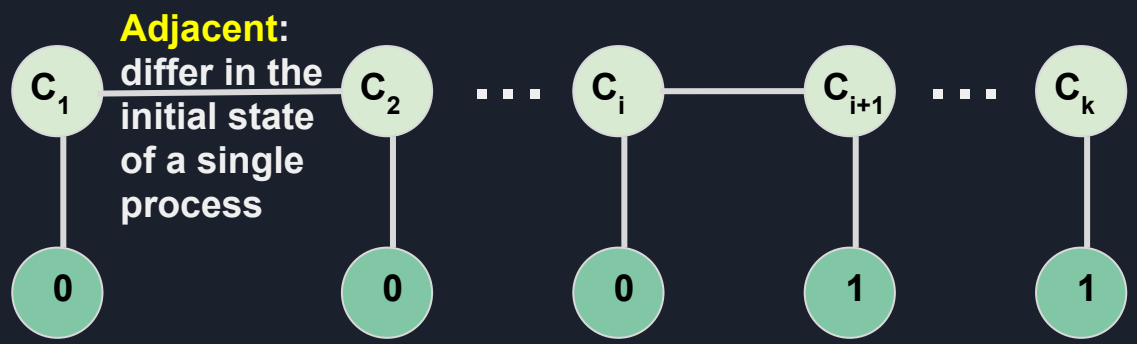
Assume all initial configurations are either 0-valent or 1-valent.



Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

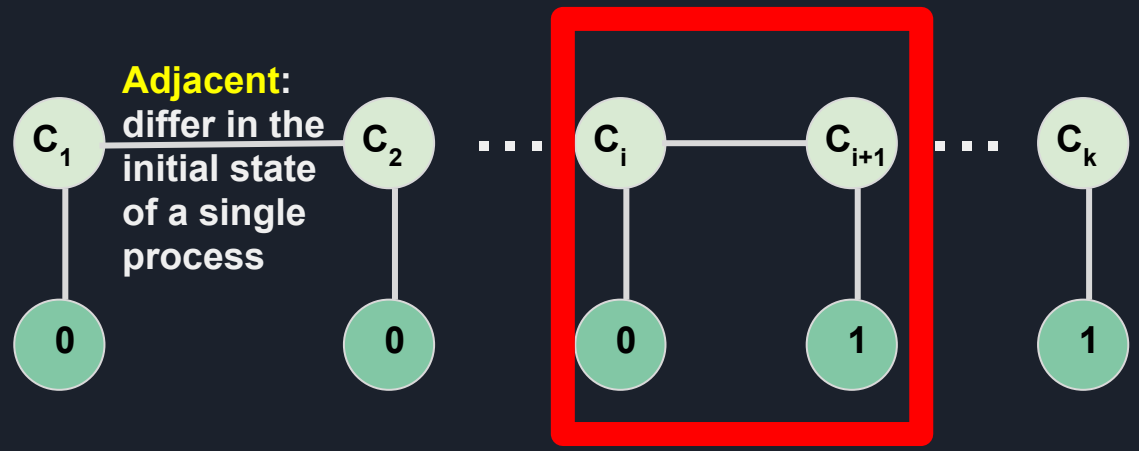
Assume all initial configurations are either 0-valent or 1-valent.



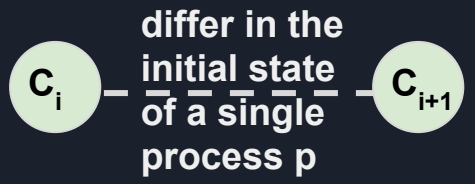
Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.

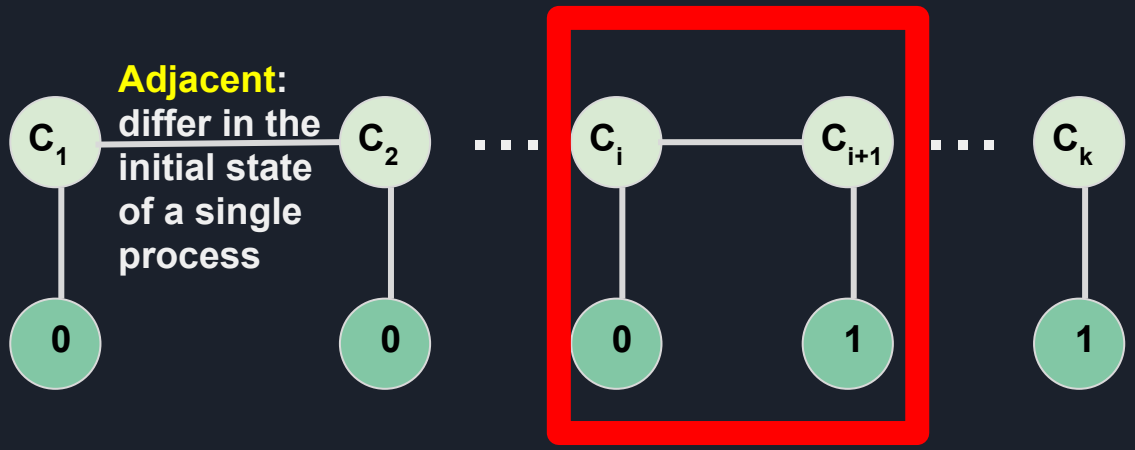


Most exciting part!!!

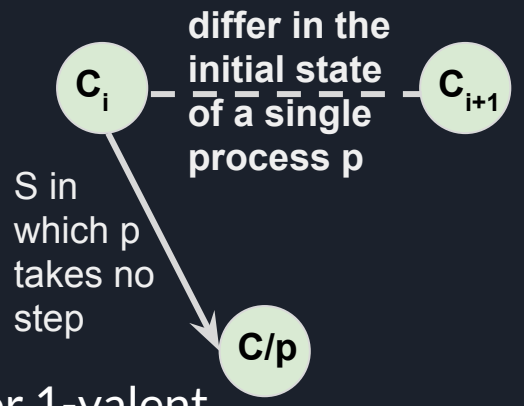


- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.

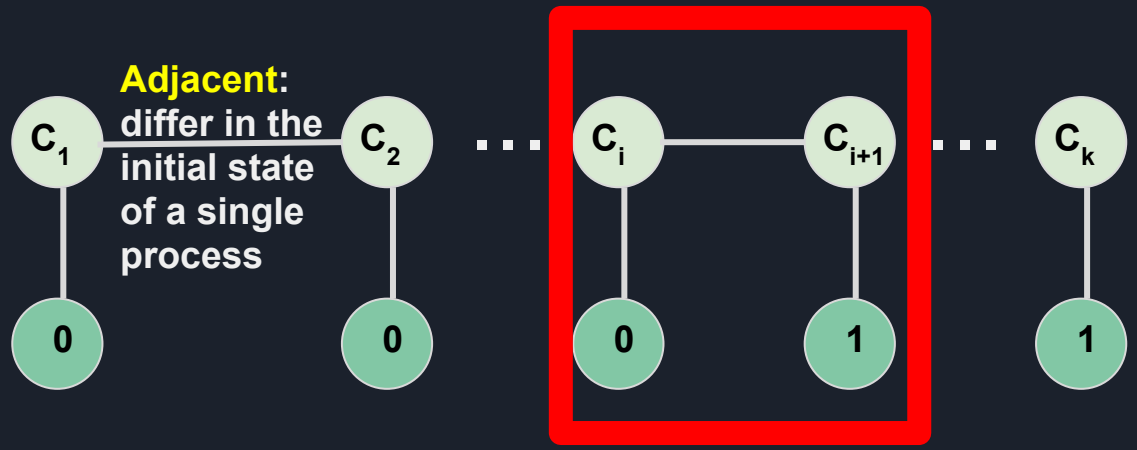


Most exciting part!!!

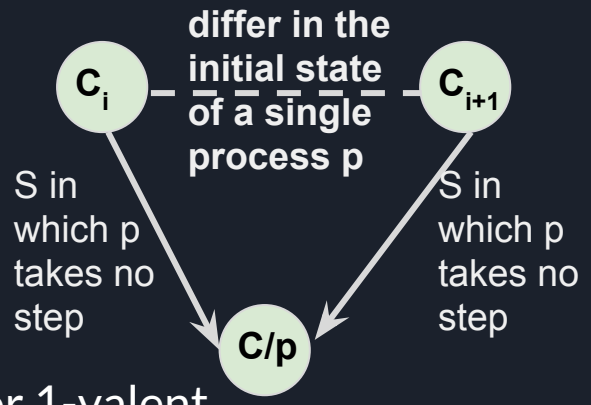


- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.

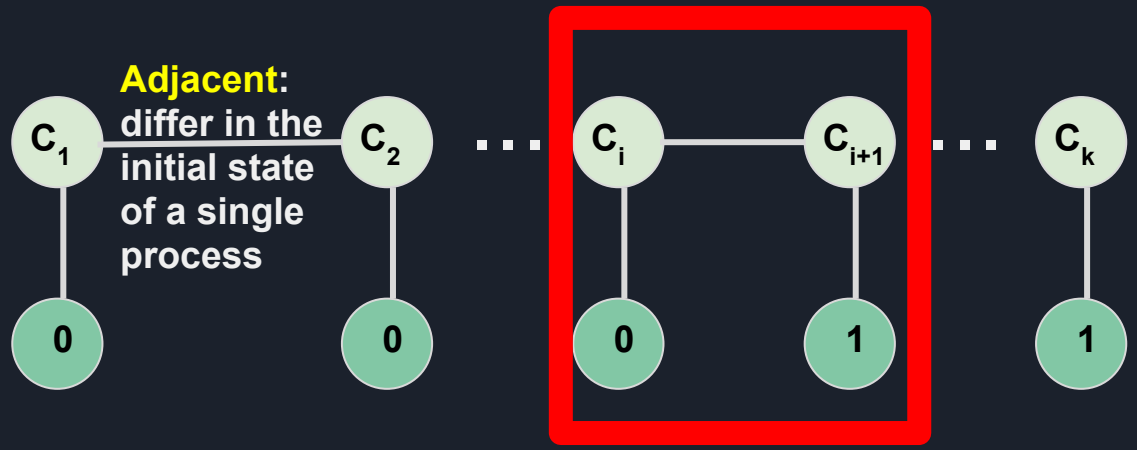


Most exciting part!!!



- Lemma 2
  - P has a bivalent initial configuration.

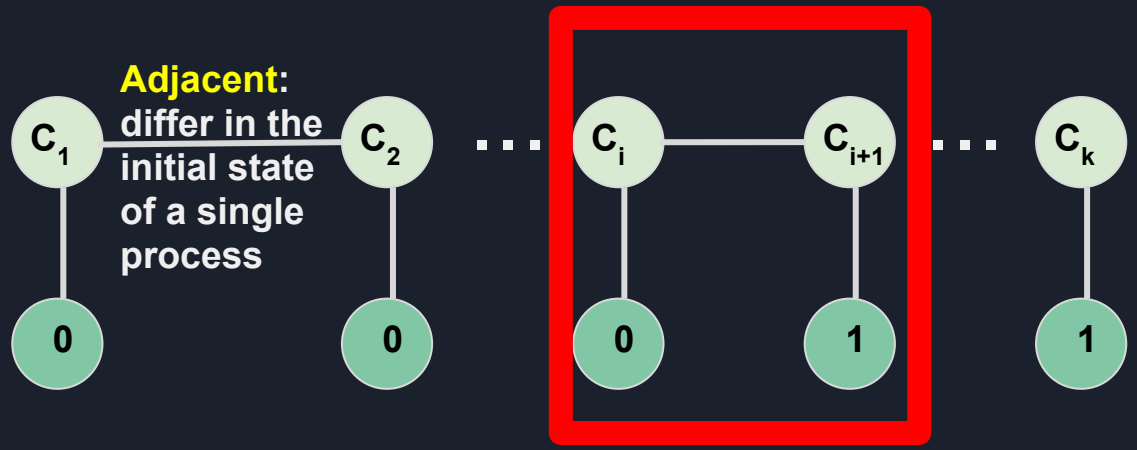
Assume all initial configurations are either 0-valent or 1-valent.

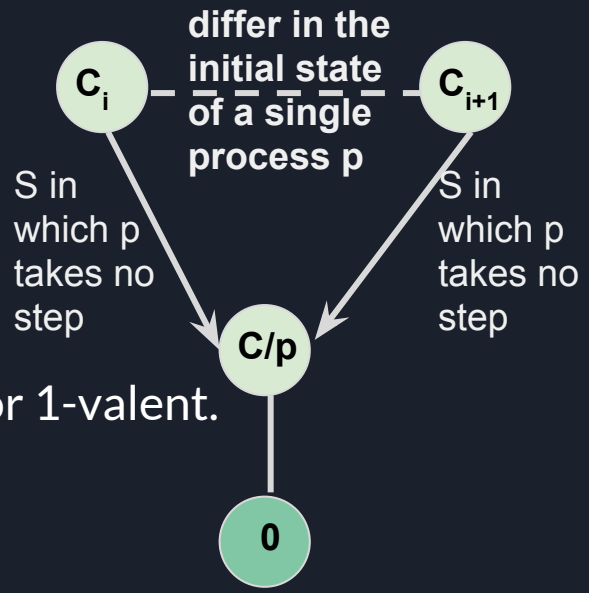


Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.

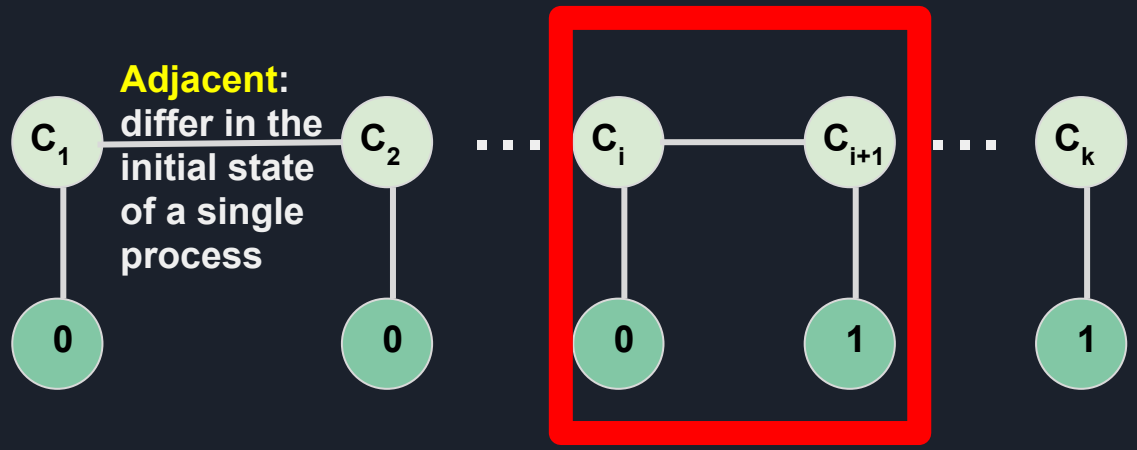


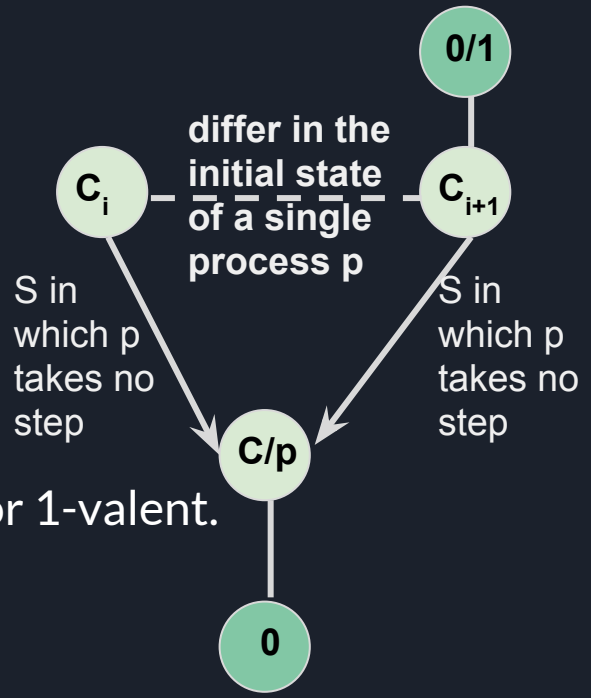


Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.

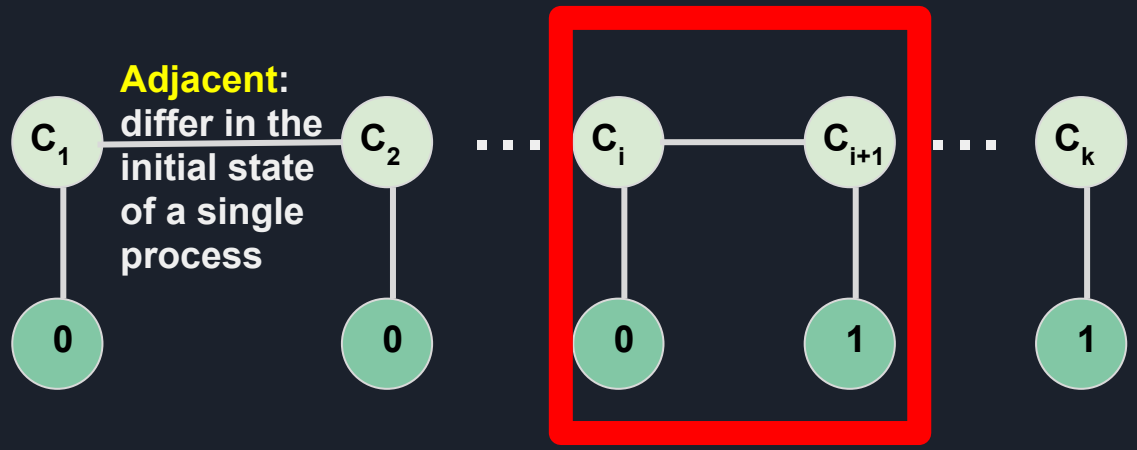


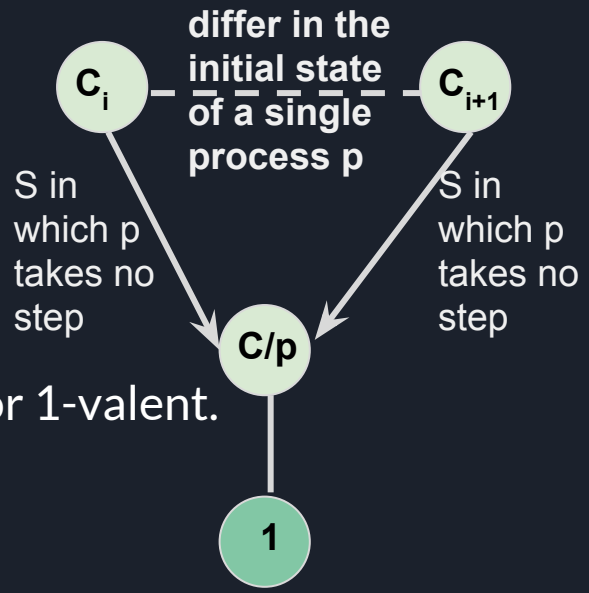


Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.

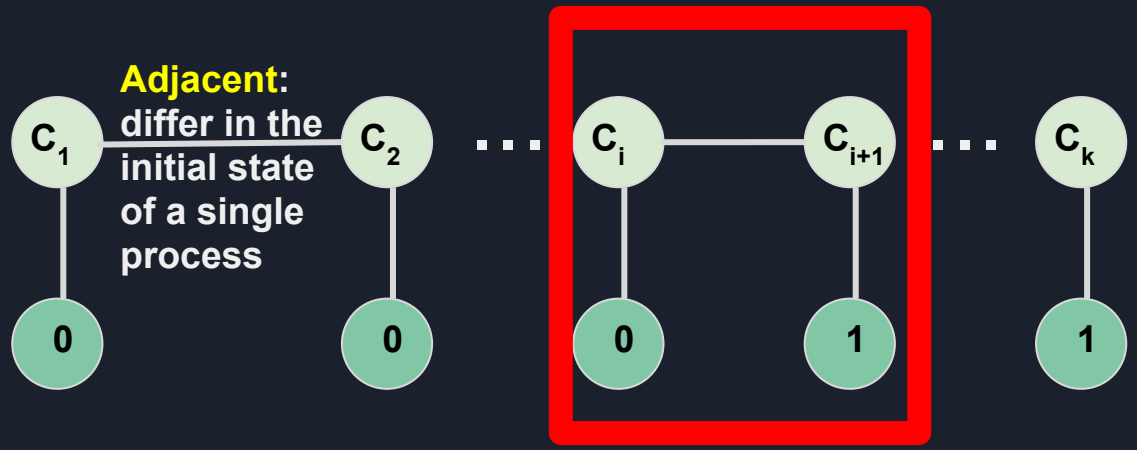


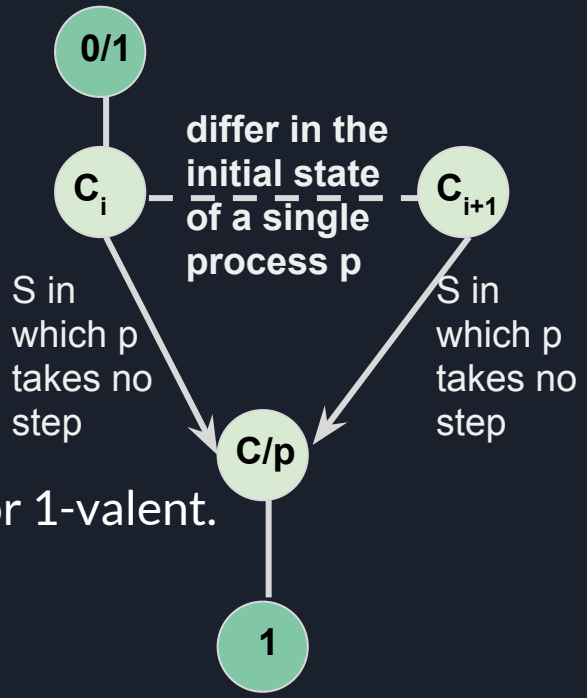


Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.





Proof Outline

- For any protocol, there is an initial configuration that is bivalent
- Then there is another bivalent configuration reachable from it after applying some event
- And another reachable bivalent configuration
- …
- An infinite undeciding run

Most exciting part!!!

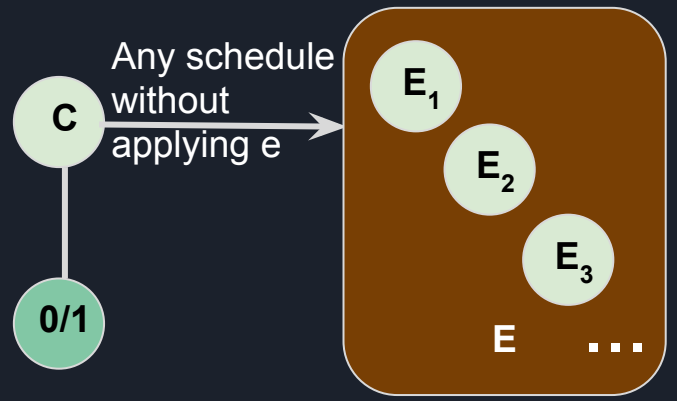
- Lemma 3
  - Let C be a bivalent configuration of P, and e=(p, m) be an event that is applicable to C. Let E be the set of configurations reachable from C without applying e, and let D=e(E), the set of configurations after applying e to all those in E. Then, D contains a bivalent configuration.

Most exciting part!!!

- Lemma 3
  - Let C be a bivalent configuration of P, and e=(p, m) be an event that is applicable to C. Let E be the set of configurations reachable from C without applying e, and let D=e(E), the set of configurations after applying e to all those in E. Then, D contains a bivalent configuration.

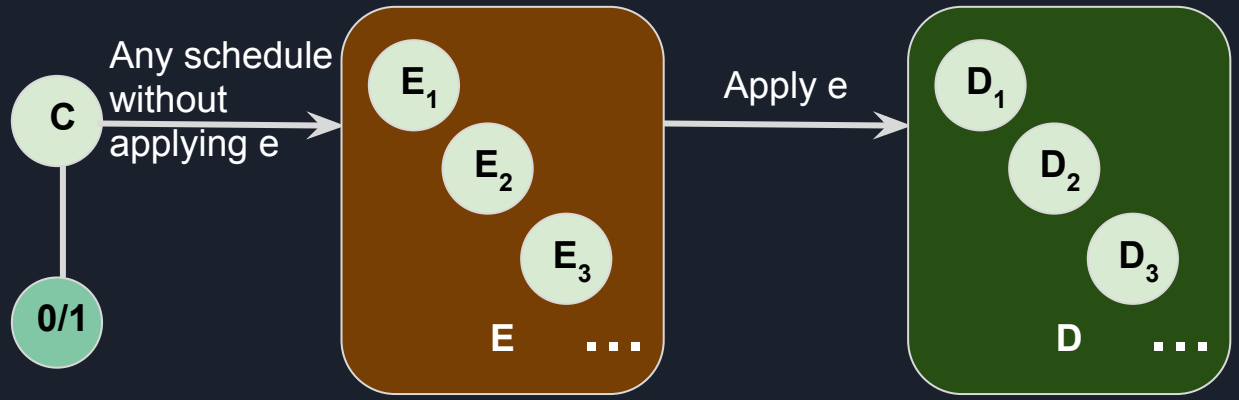
Most exciting part!!!

- Lemma 3
  - Let C be a bivalent configuration of P, and e=(p, m) be an event that is applicable to C. Let E be the set of configurations reachable from C without applying e, and let D=e(E), the set of configurations after applying e to all those in E. Then, D contains a bivalent configuration.



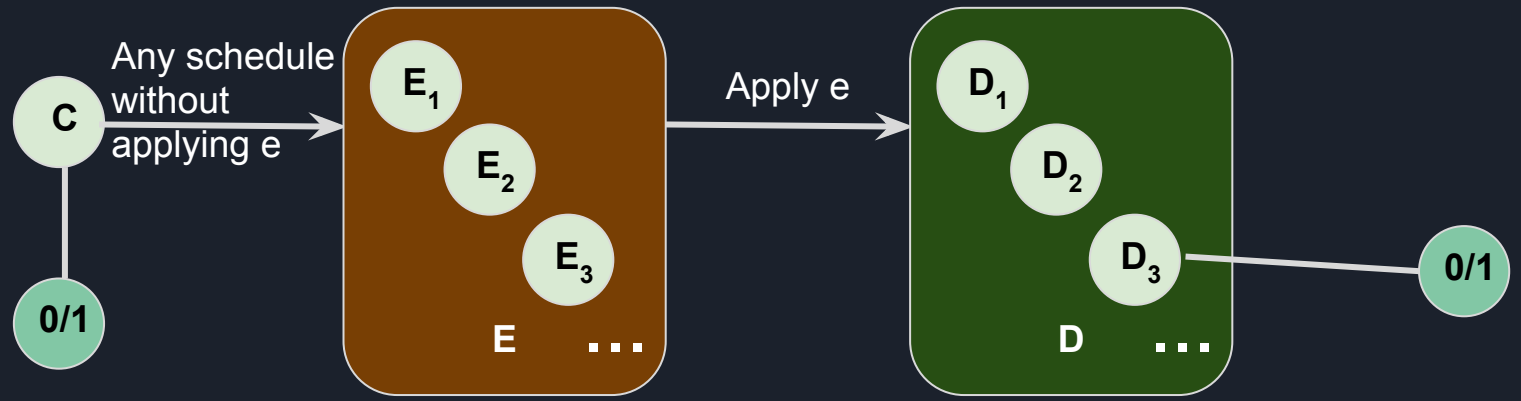
Most exciting part!!!

- Lemma 3
  - Let C be a bivalent configuration of P, and e=(p, m) be an event that is applicable to C. Let E be the set of configurations reachable from C without applying e, and let D=e(E), the set of configurations after applying e to all those in E. Then, D contains a bivalent configuration.



Most exciting part!!!

- Lemma 3
  - Let C be a bivalent configuration of P, and e=(p, m) be an event that is applicable to C. Let E be the set of configurations reachable from C without applying e, and let D=e(E), the set of configurations after applying e to all those in E. Then, D contains a bivalent configuration.



Most exciting part!!!

- Assume all configurations in D are univalent.

Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.

Most exciting part!!!

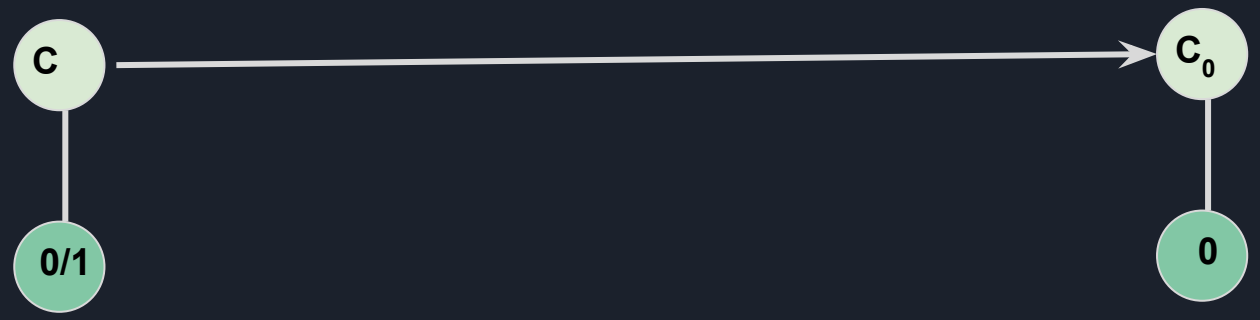
- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in {0, 1} there exists a C_i reachable from C that is i-valent.

Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in {0, 1} there exists a C_i reachable from C that is i-valent.
  - Consider C_0 without loss of generality,

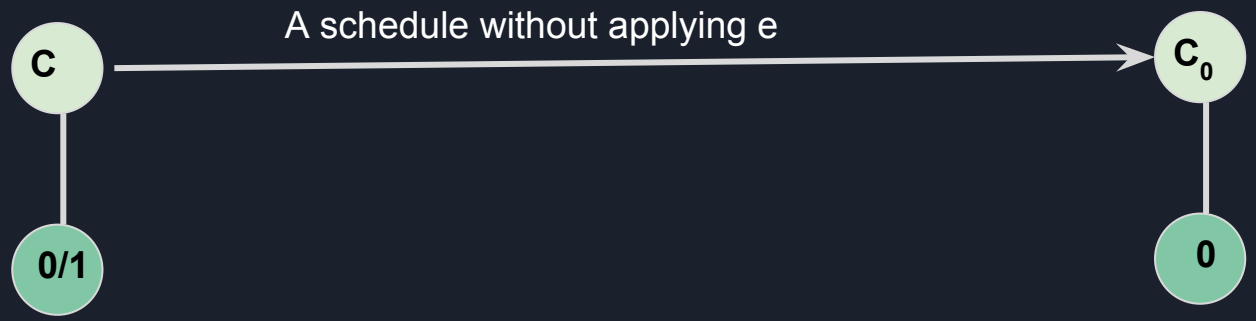
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in {0, 1} there exists a C_i reachable from C that is i-valent.
  - Consider C_0 without loss of generality,



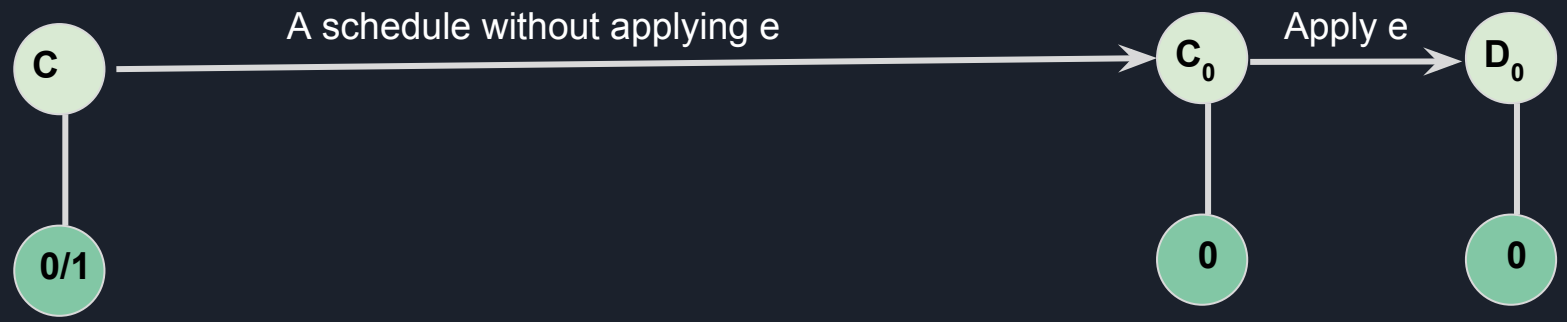
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in {0, 1} there exists a C_i reachable from C that is i-valent.
  - Consider C_0 without loss of generality,



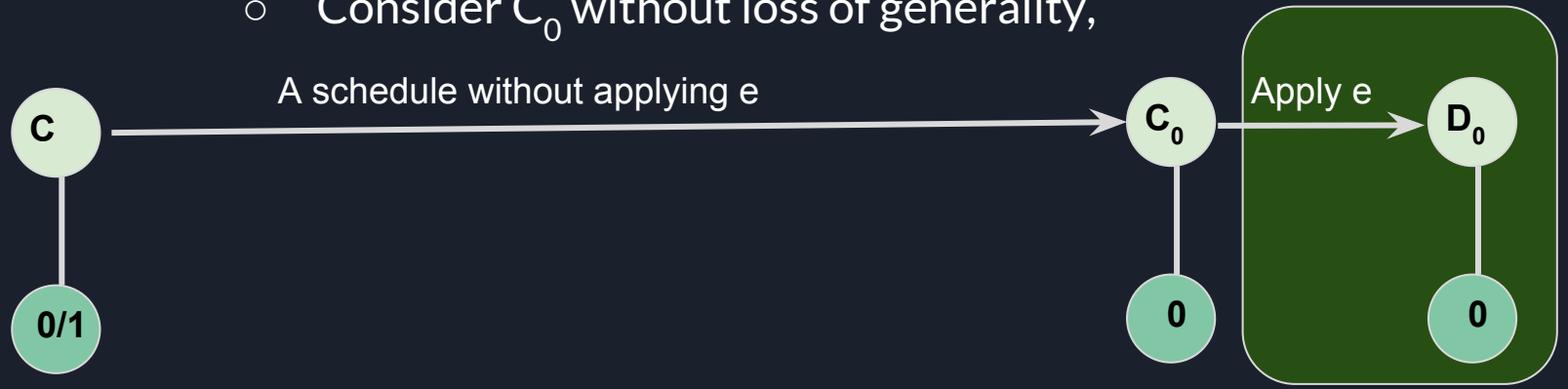
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in {0, 1} there exists a C_i reachable from C that is i-valent.
  - Consider C_0 without loss of generality,



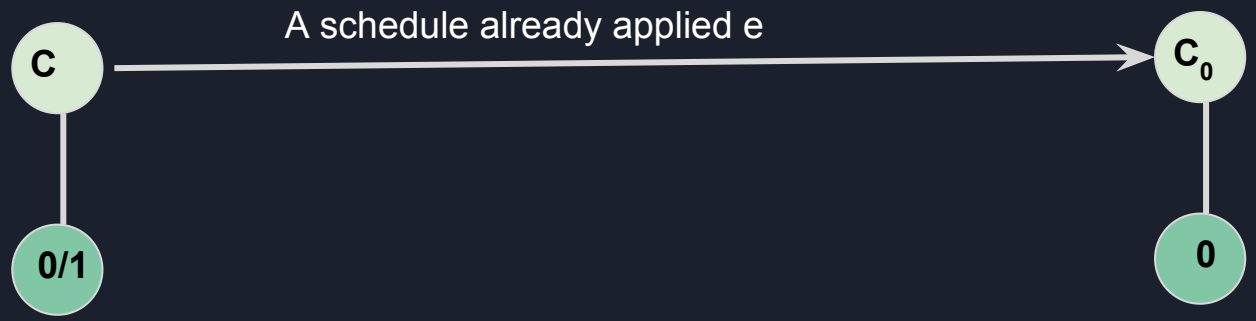
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in {0, 1} there exists a C_i reachable from C that is i-valent.
  - Consider C_0 without loss of generality,



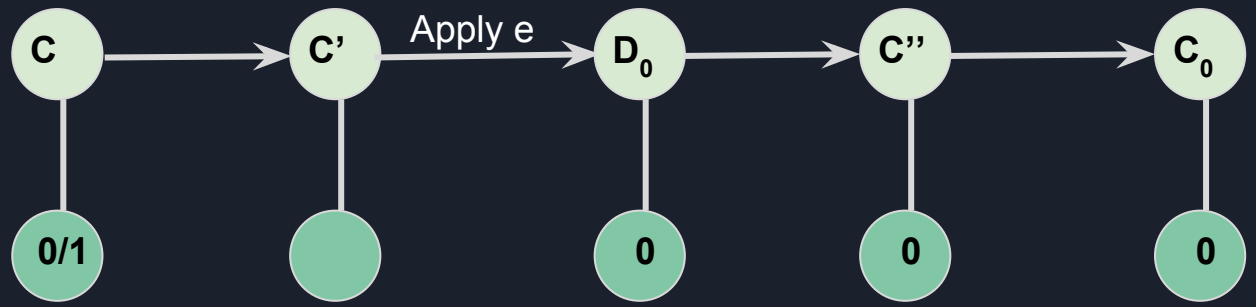
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in {0, 1} there exists a C_i reachable from C that is i-valent.
  - Consider C_0 without loss of generality,



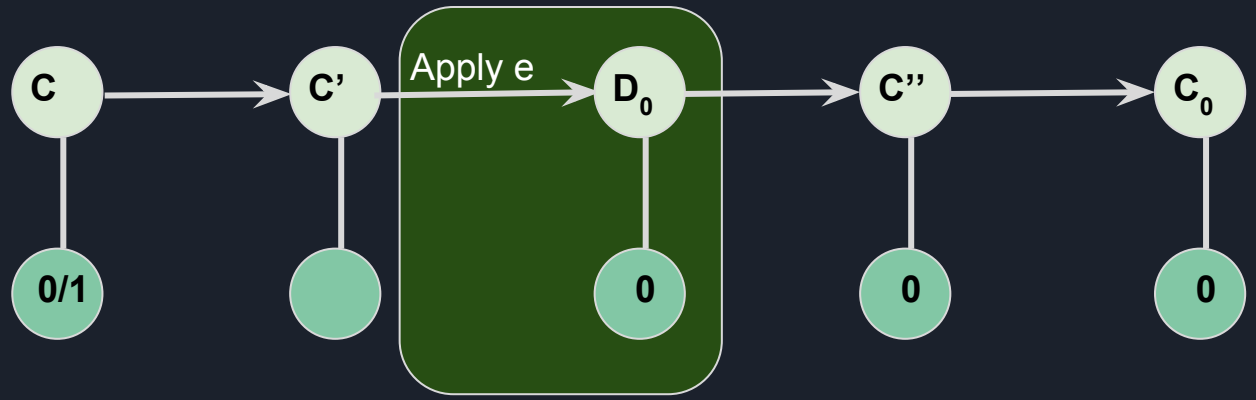
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in {0, 1} there exists a C_i reachable from C that is i-valent.
  - Consider C_0 without loss of generality,



Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in {0, 1} there exists a C_i reachable from C that is i-valent.
  - Consider C_0 without loss of generality,



Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in {0, 1} there exists a C_i reachable from C that is i-valent.
  - Consider C_0 without loss of generality,
  - There exists D_0 that is 0-valent.

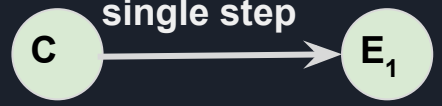
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
- Without loss of generality, assume D_0=e(C) in D is 0-valent

Most exciting part!!!

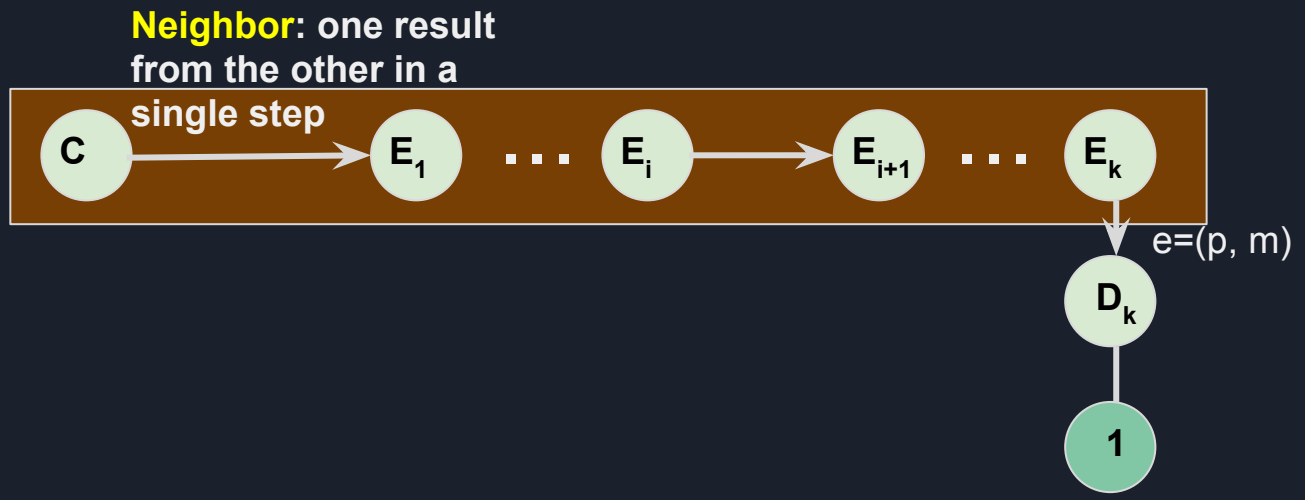
- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
- Without loss of generality, assume D_0=e(C) in D is 0-valent

Neighbor: one result from the other in a single step



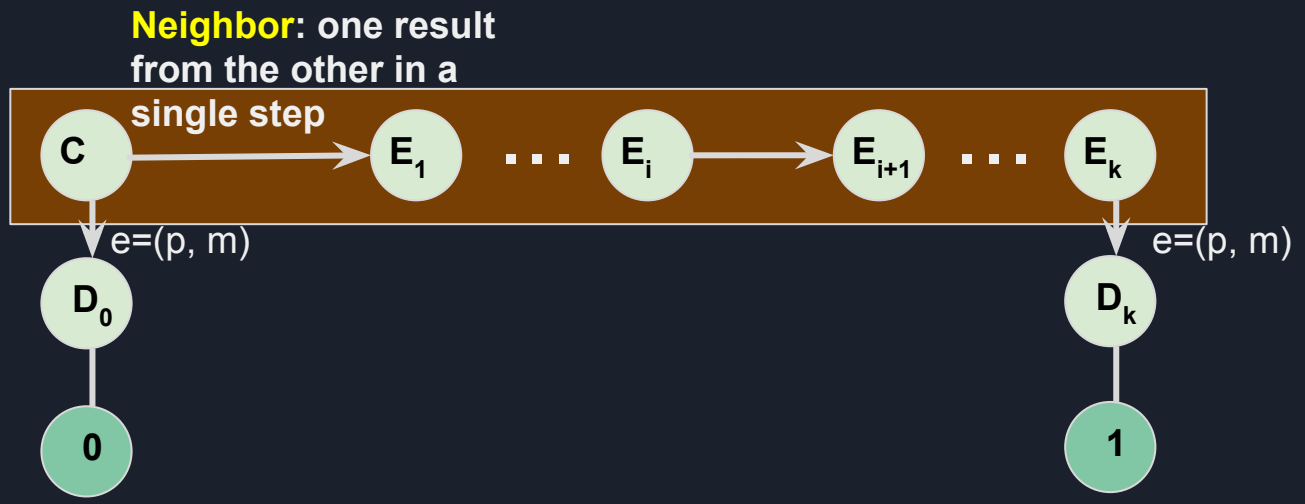
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
- Without loss of generality, assume D_0=e(C) in D is 0-valent



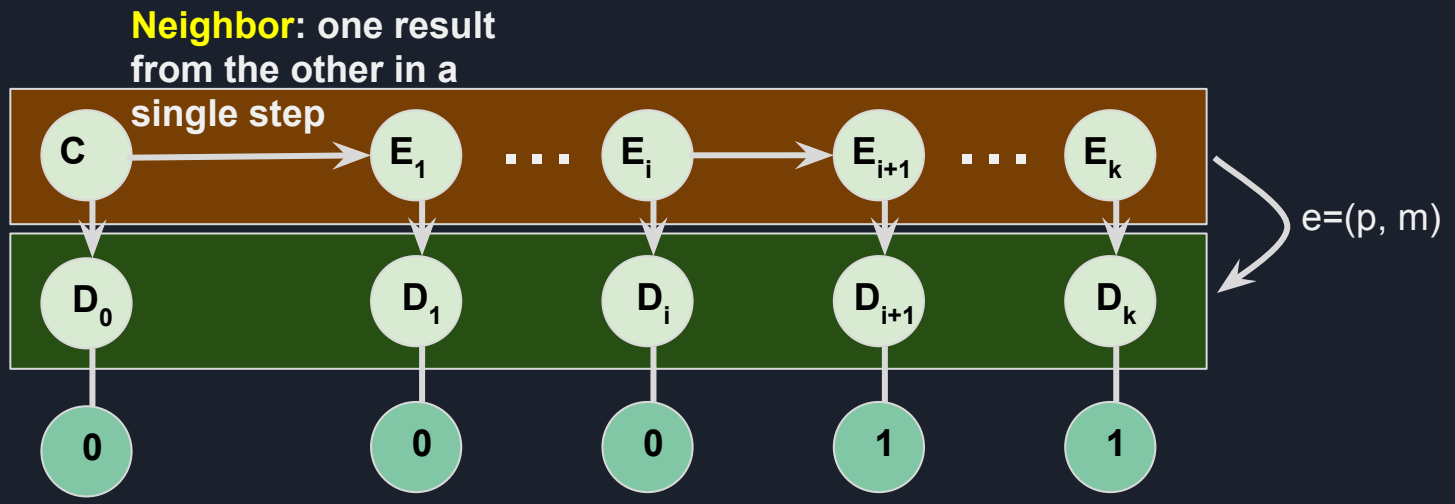
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
- Without loss of generality, assume D_0=e(C) in D is 0-valent



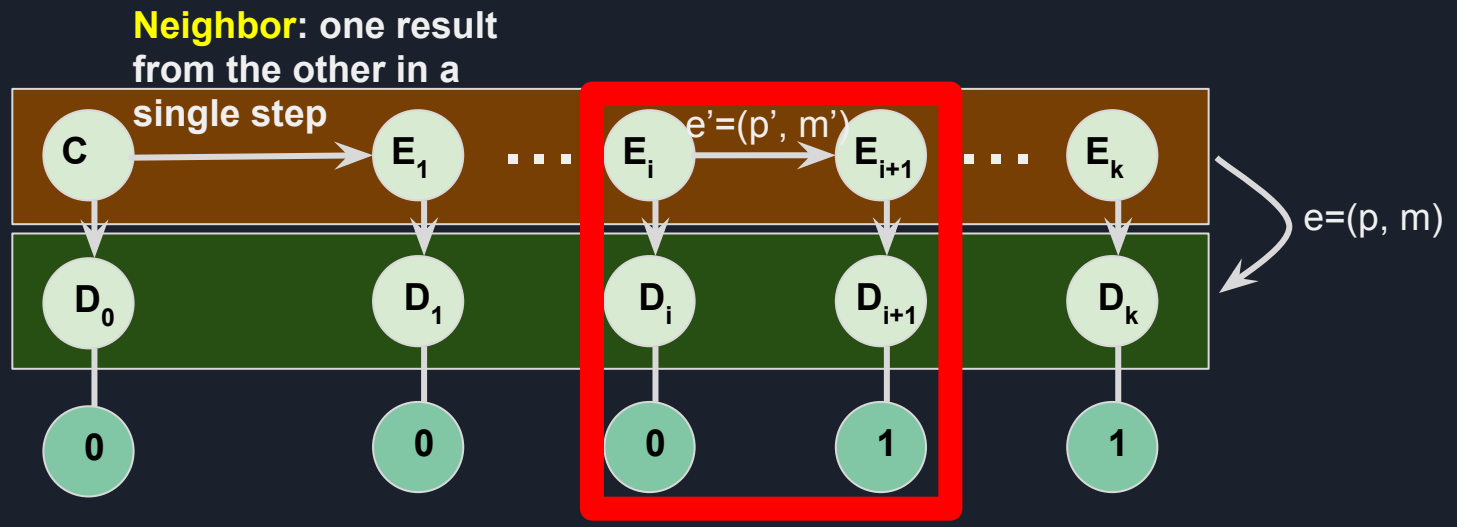
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
- Without loss of generality, assume D_0=e(C) in D is 0-valent



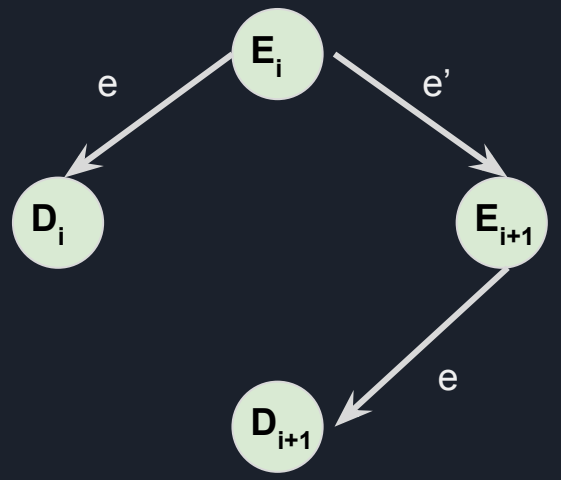
Most exciting part!!!

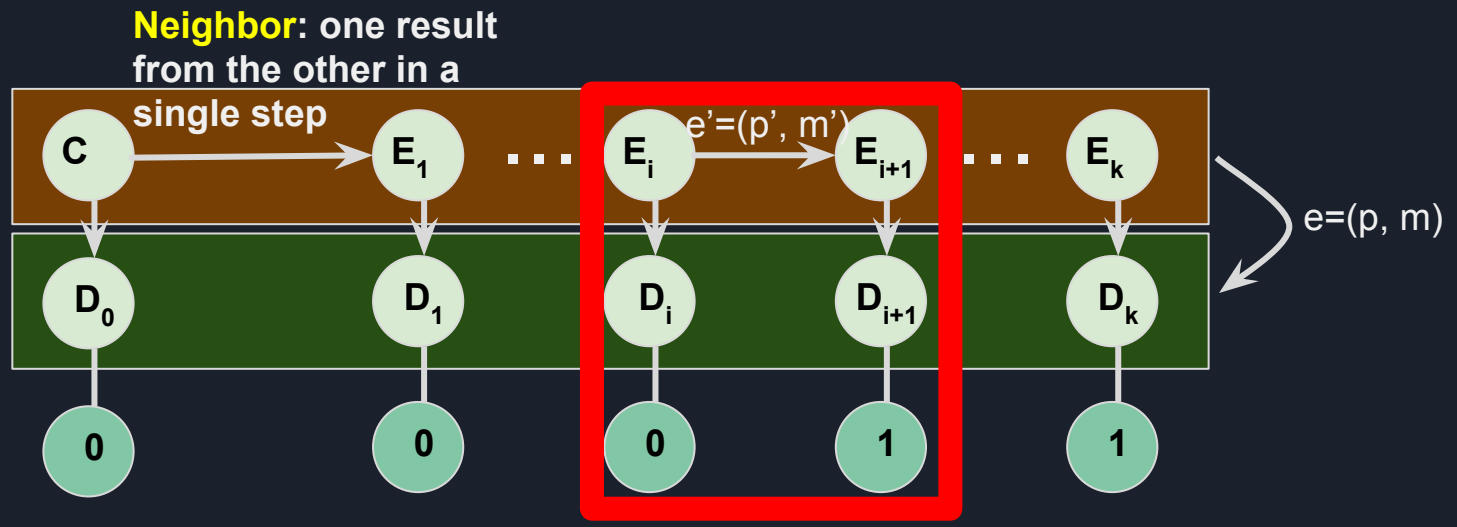
- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
- Without loss of generality, assume D_0=e(C) in D is 0-valent



Most exciting part!!!

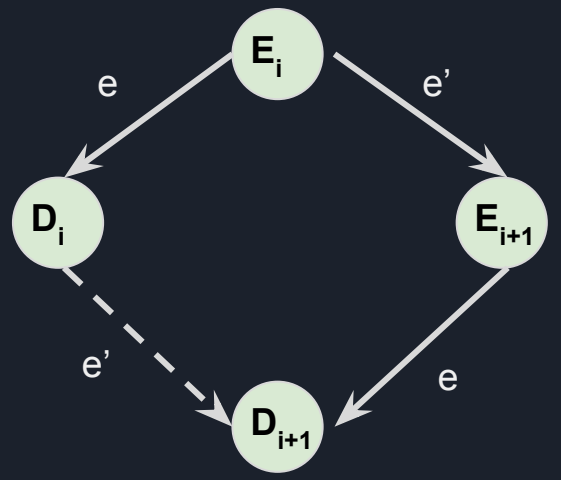
- Case 1: p’ != p

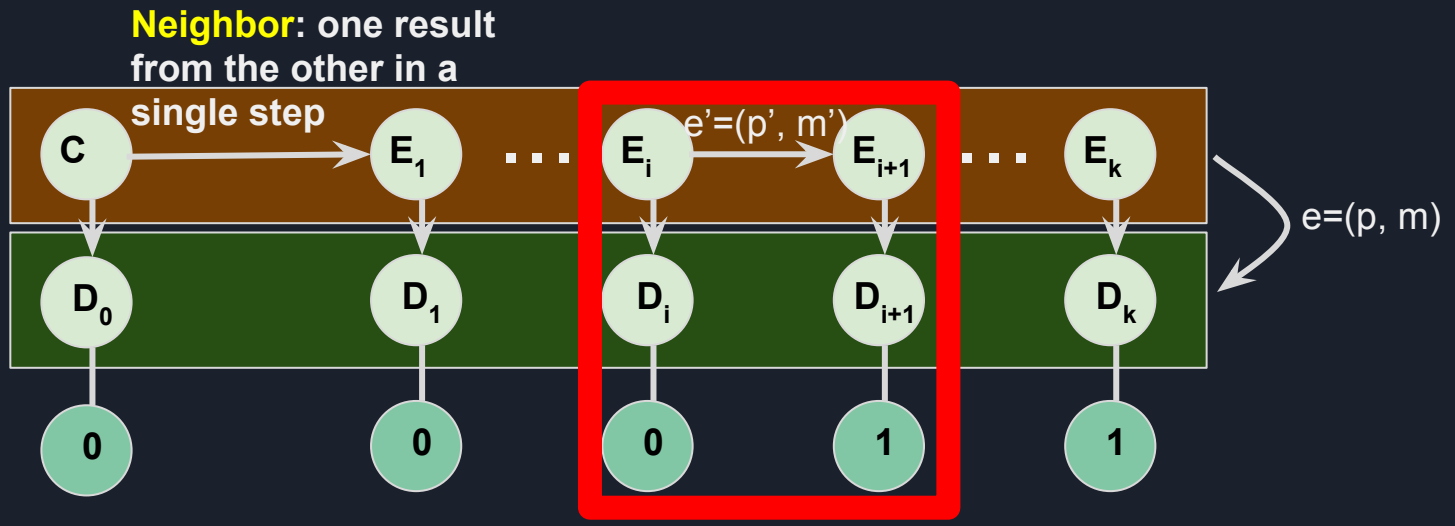




Most exciting part!!!

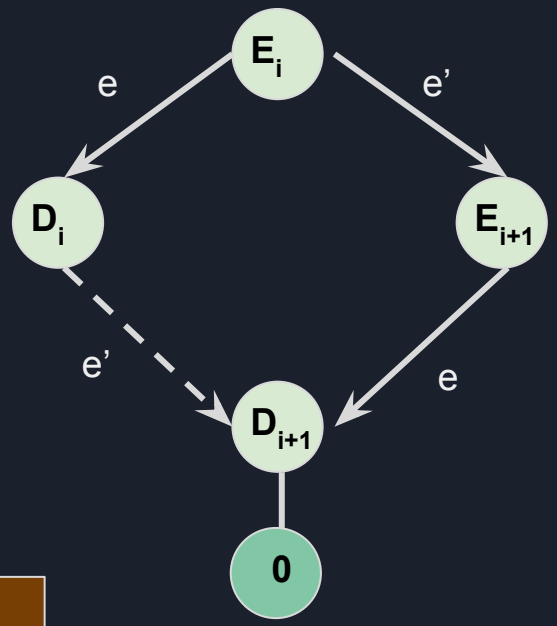
- Case 1: p’ != p
  - Apply Lemma 1

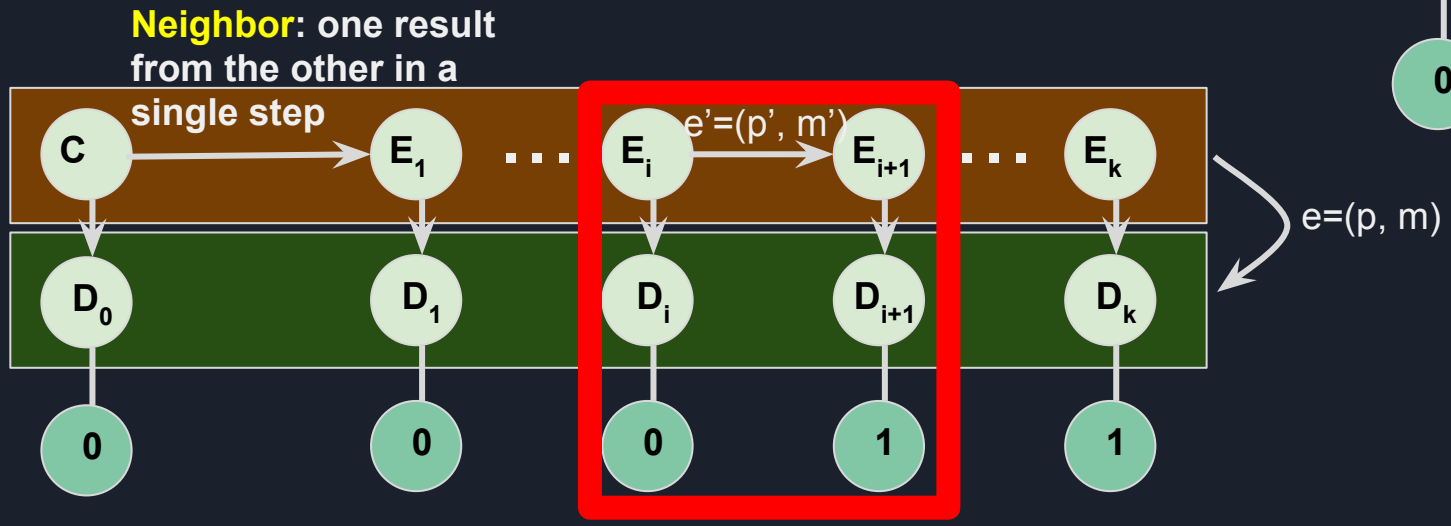




Most exciting part!!!

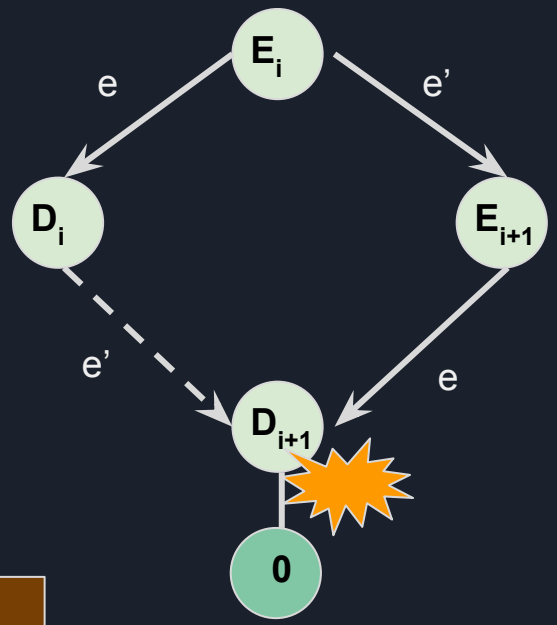
- Case 1: p’ != p
  - Apply Lemma 1

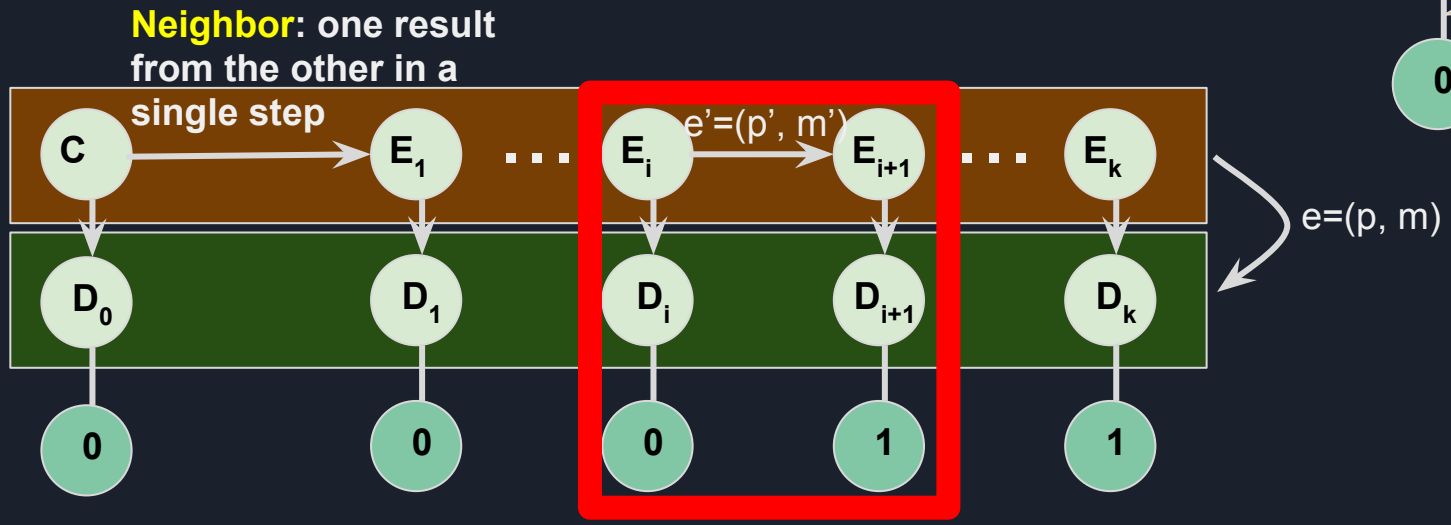




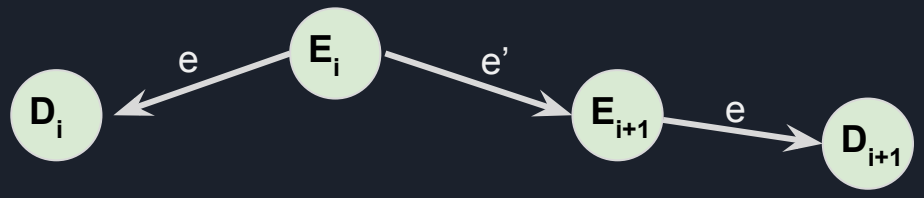
Most exciting part!!!

- Case 1: p’ != p
  - Apply Lemma 1

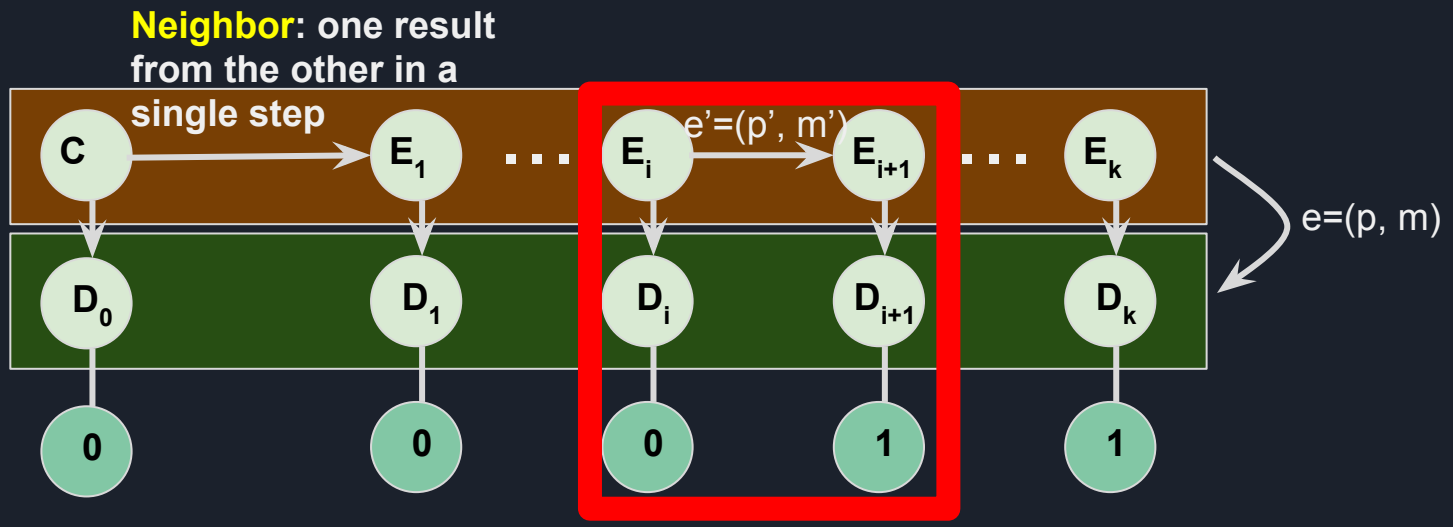




Most exciting part!!!

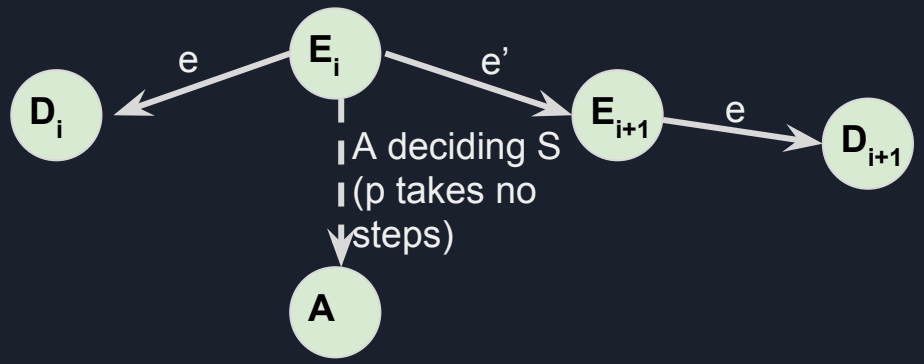


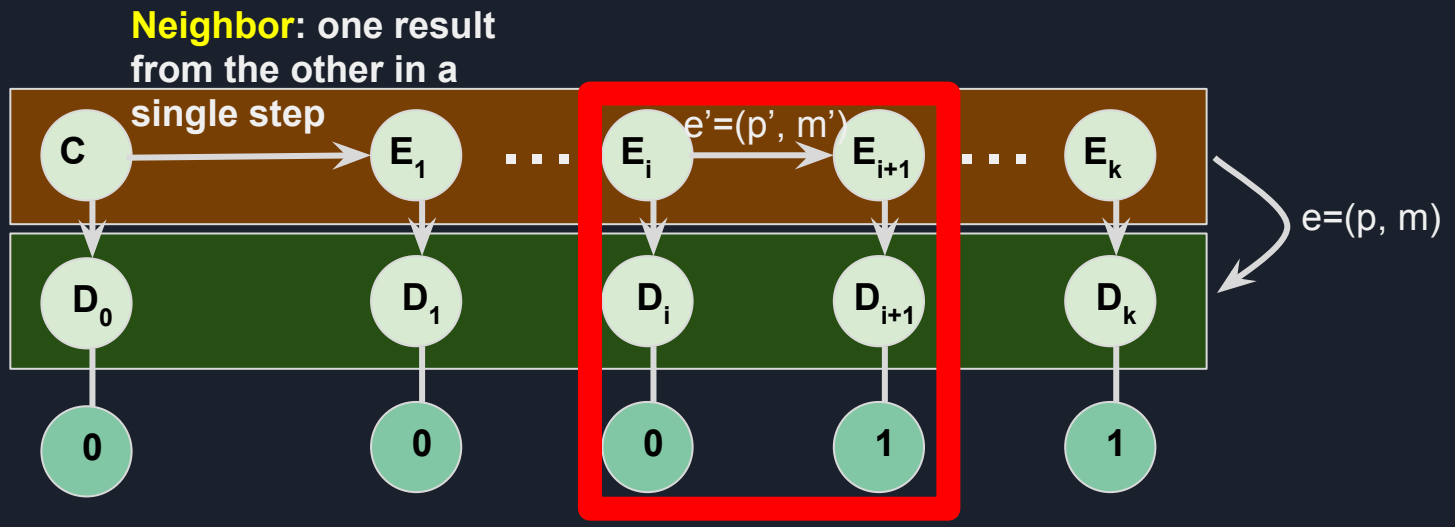
- Case 2: p' = p



Most exciting part!!!

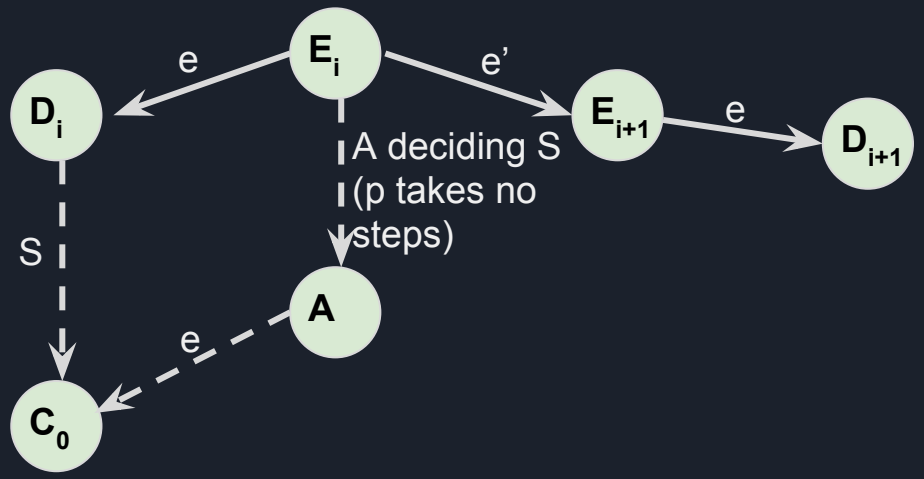
- Case 2: p’ = p

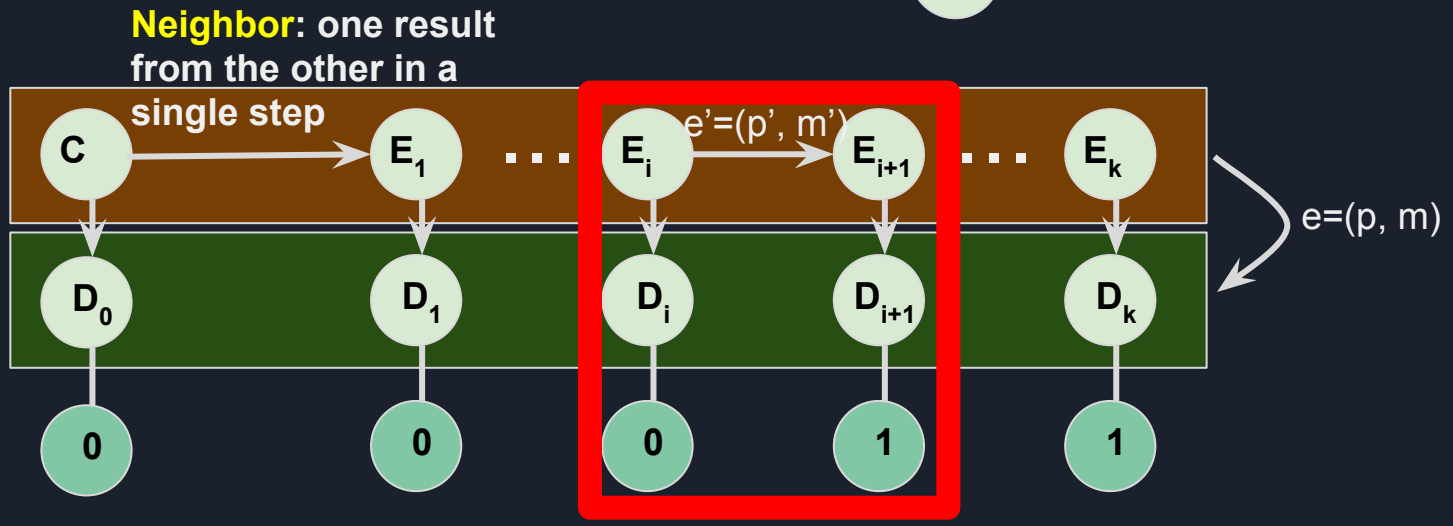




Most exciting part!!!

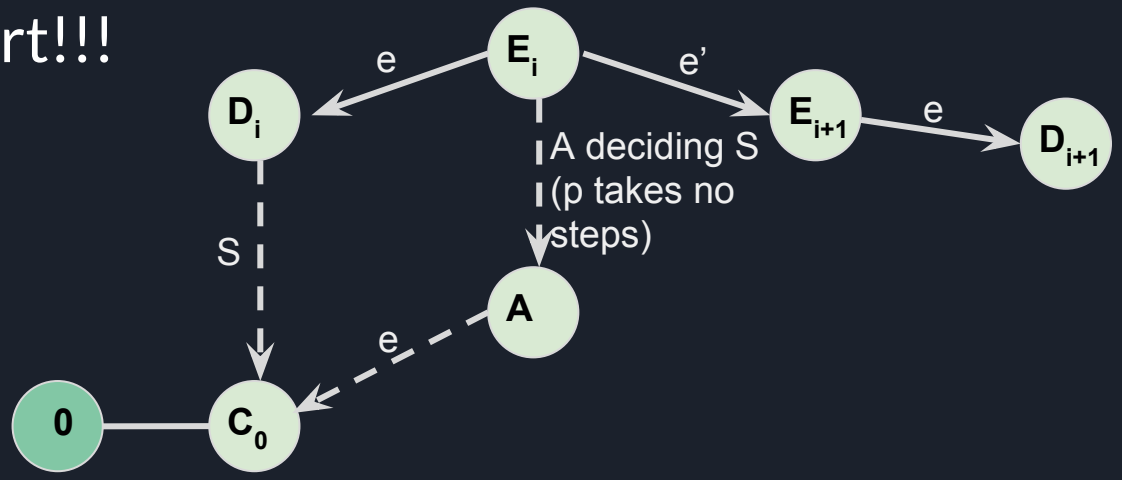
- Case 2: p’ = p
  - Apply Lemma 1

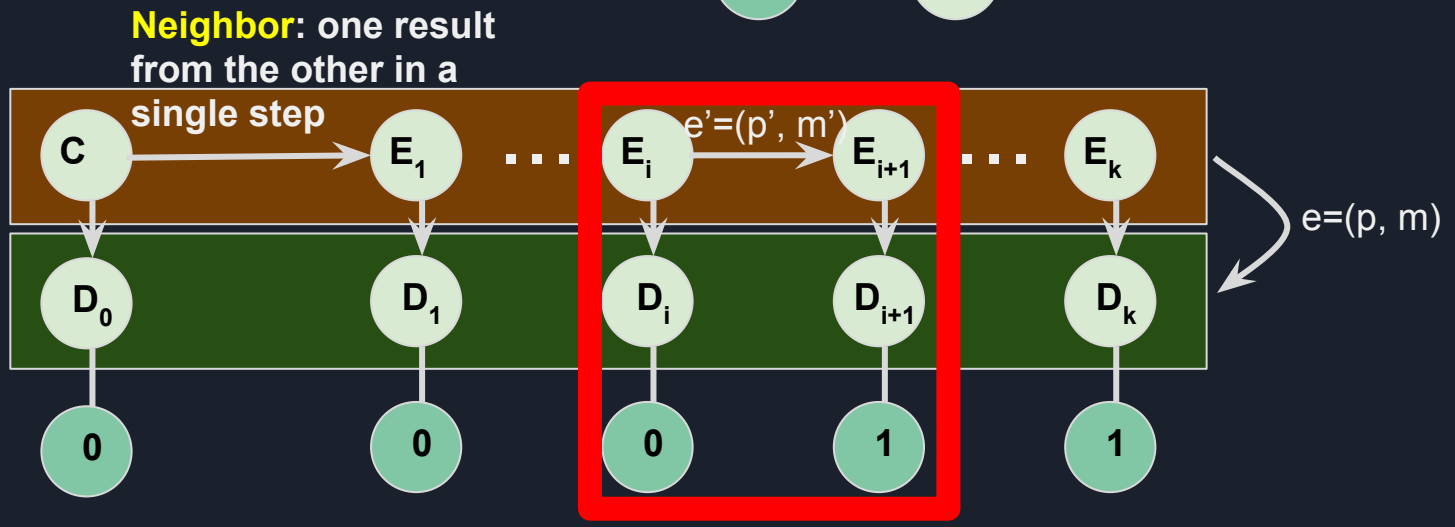




Most exciting part!!!

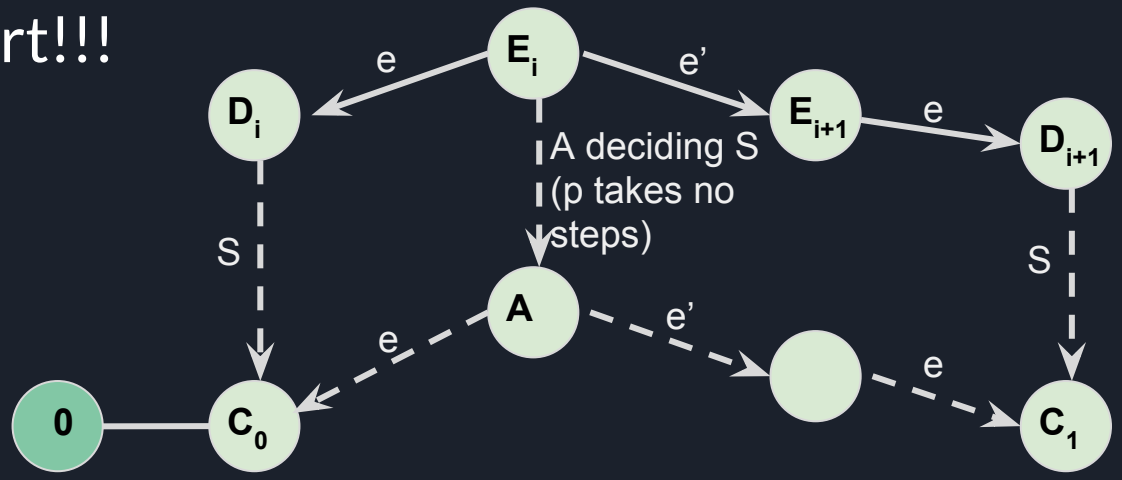
- Case 2: p’ = p
  - Apply Lemma 1

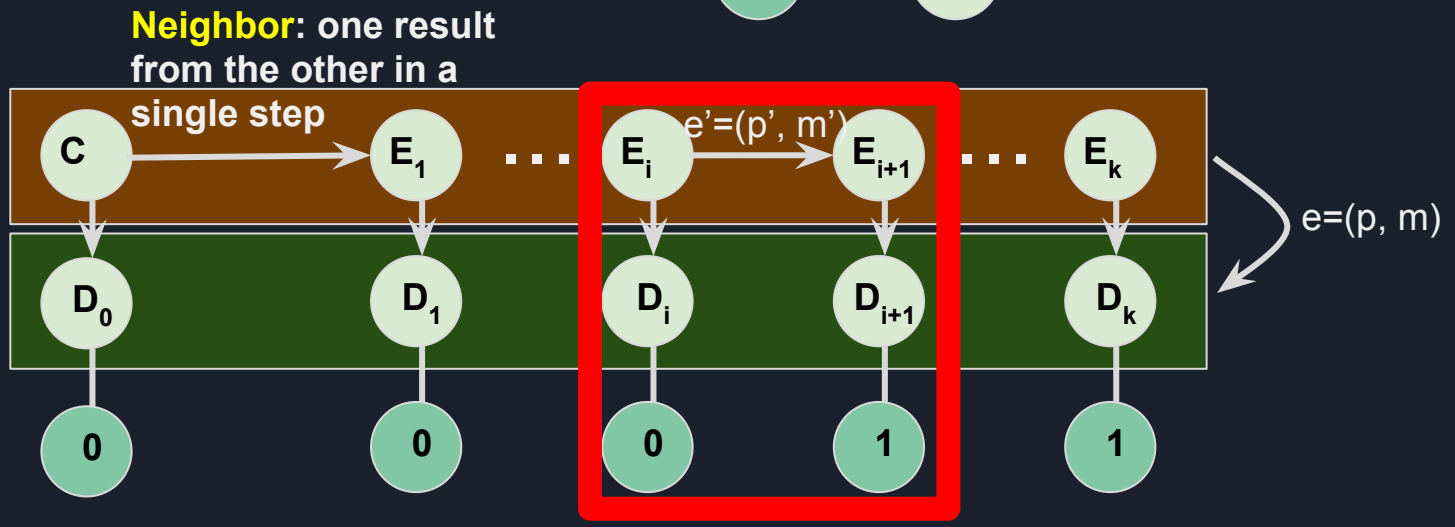




Most exciting part!!!

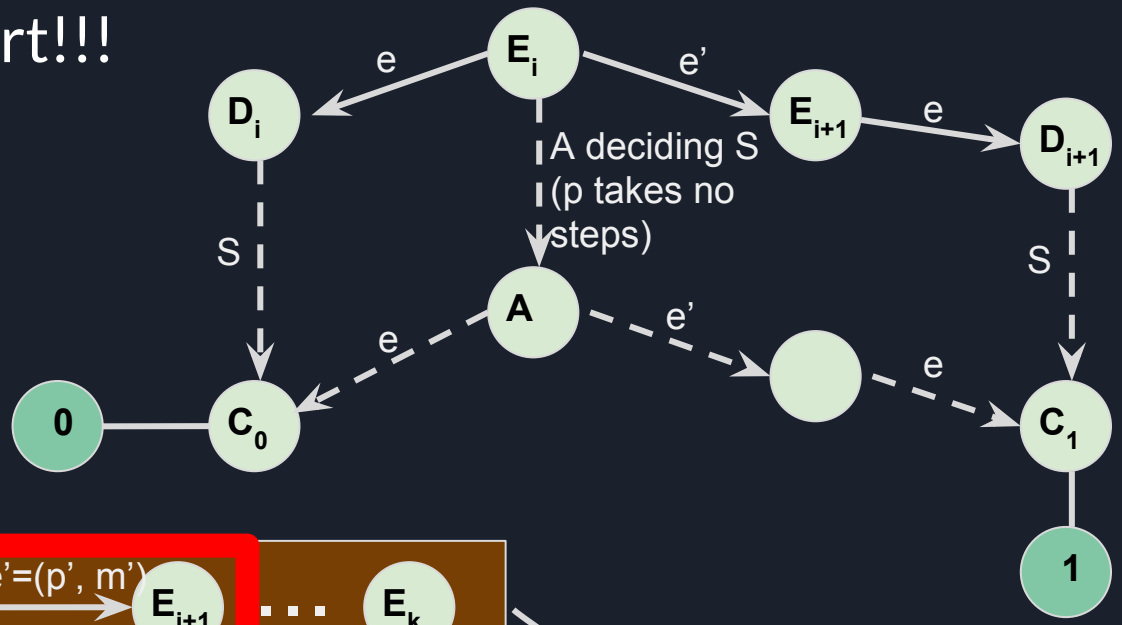
- Case 2: p’ = p
  - Apply Lemma 1

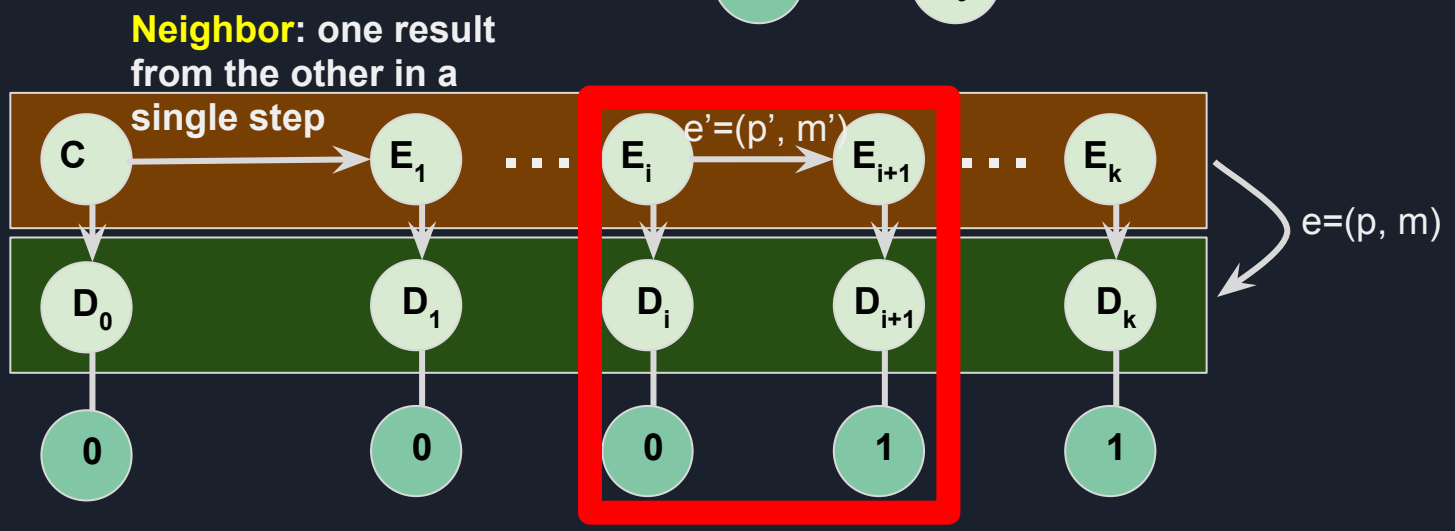




Most exciting part!!!

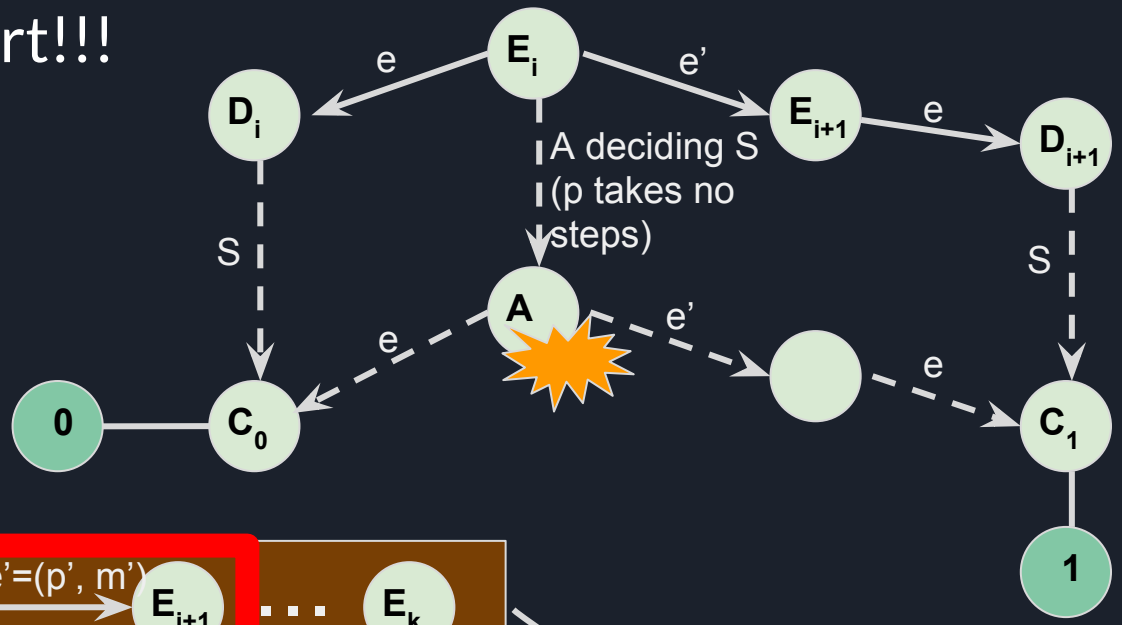
- Case 2: p’ = p
  - Apply Lemma 1



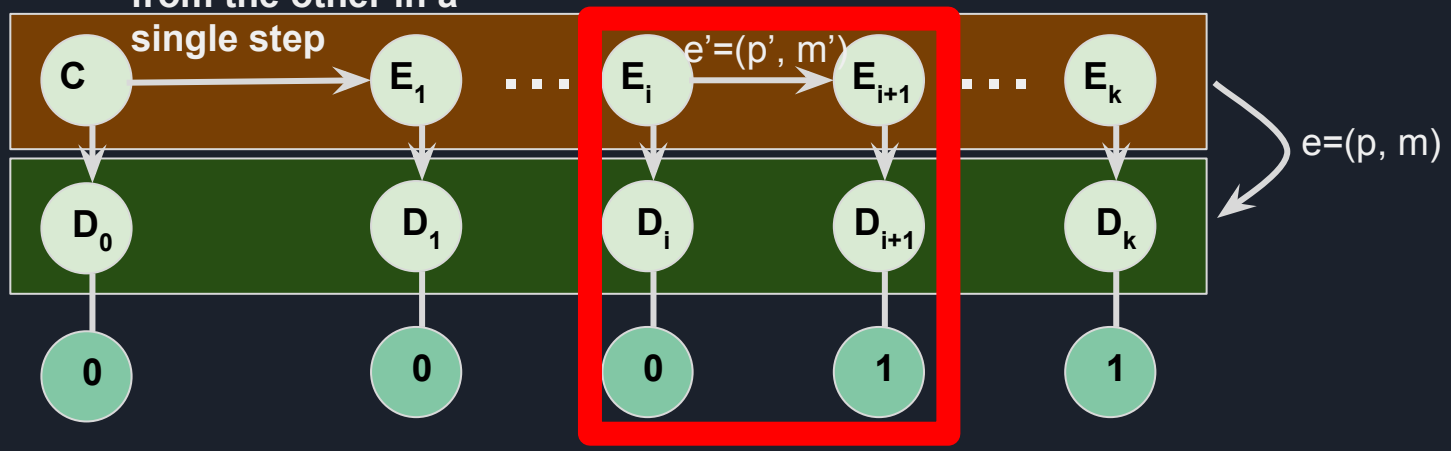


Most exciting part!!!

- Case 2: p’ = p
  - Apply Lemma 1



Neighbor: one result from the other in a single step



Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
- Without loss of generality, assume D_0=e(C) in D is 0-valent
- Contradiction!

Proof Outline

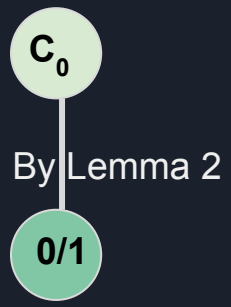
- For any protocol, there is an initial configuration that is bivalent
- Then there is another bivalent configuration reachable from it after applying some event
- And another reachable bivalent configuration
- ...
- An infinite undeciding run

Most exciting part!!!

- Construct an infinite undeciding admissible run

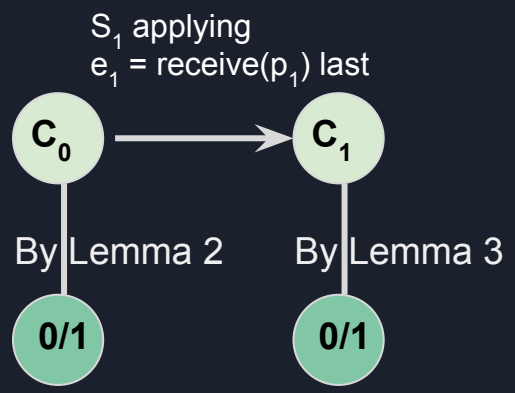
Most exciting part!!!

- Construct an infinite undeciding admissible run



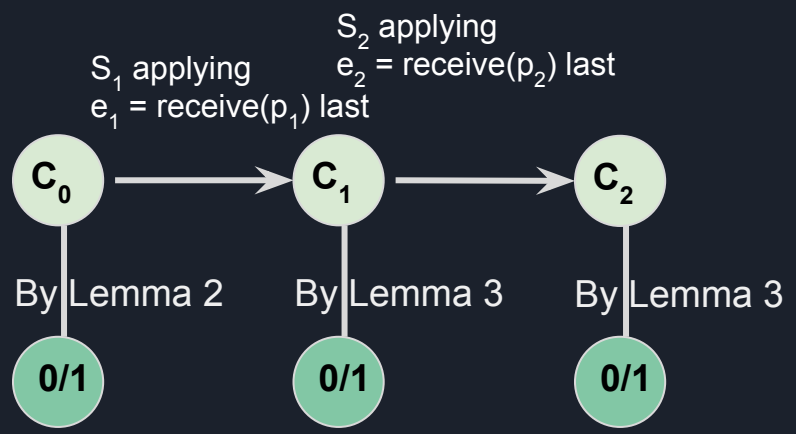
Most exciting part!!!

- Construct an infinite undeciding admissible run



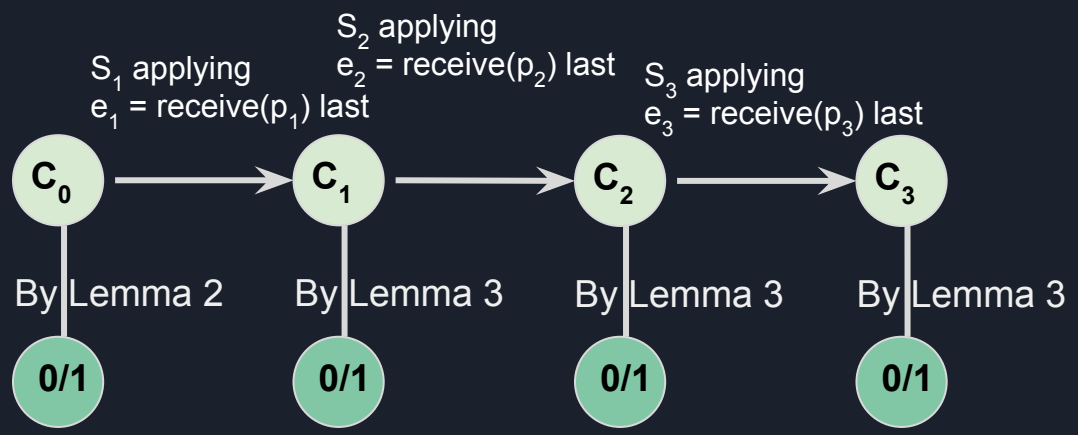
Most exciting part!!!

- Construct an infinite undeciding admissible run



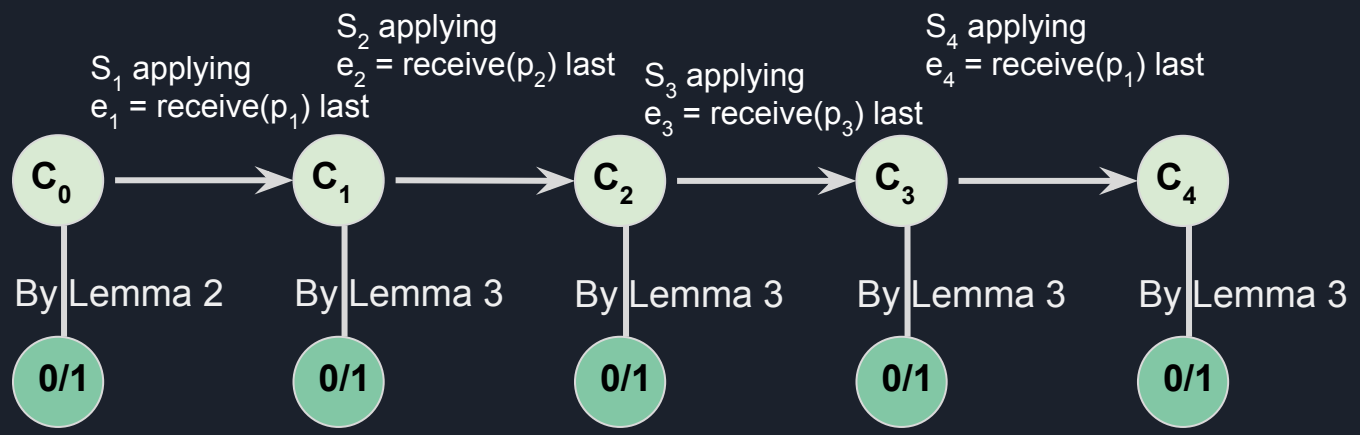
Most exciting part!!!

- Construct an infinite undeciding admissible run



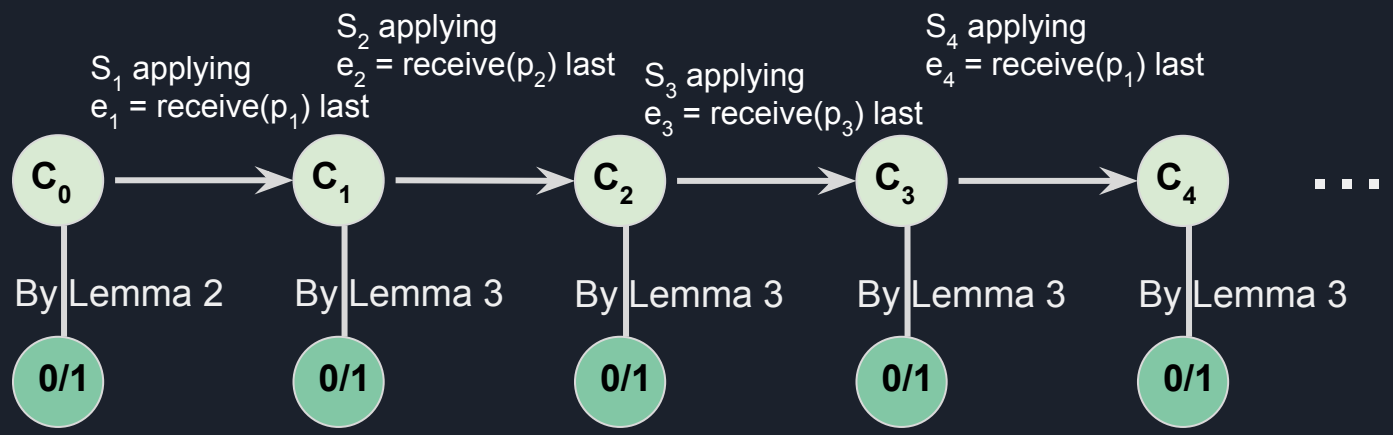
Most exciting part!!!

- Construct an infinite undeciding admissible run



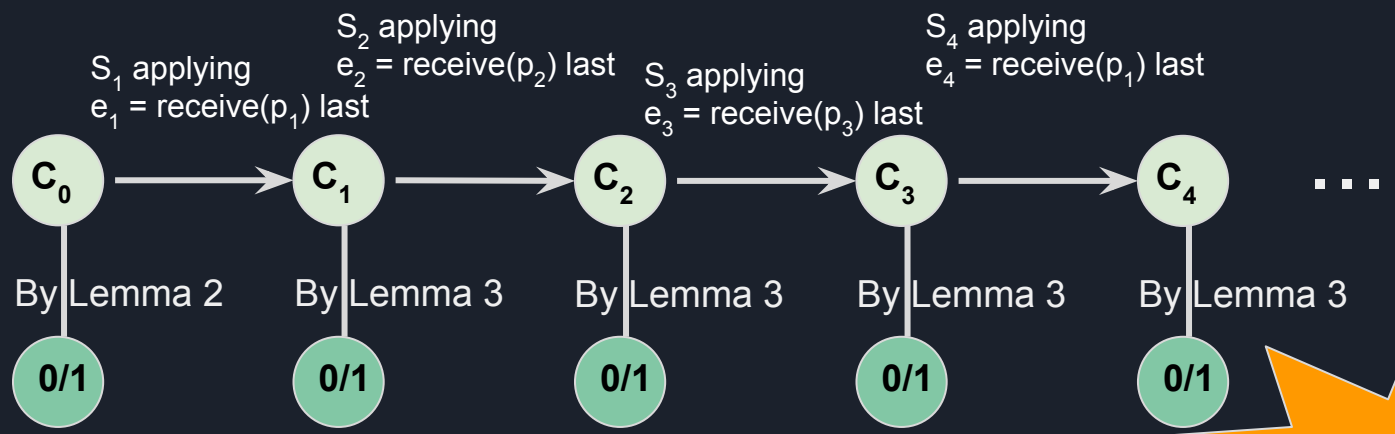
Most exciting part!!!

- Construct an infinite undeciding admissible run



Most exciting part!!!

- Construct an infinite undeciding admissible run



An infinite UNDECIDING run

Assume to the contrary that there exists P such that

- P is partially correct
  - Agreement + Validity

- Every admissible run of P is a deciding run
  - Termination

Assume to the contrary that there exists P such that

- P is partially correct
  - Agreement + Validity

- Every admissible run of P is a deciding run
  - Termination

Wrap-up

- Computation determinism
  - Deterministic
  - Probabilistic
- Timing assumptions
  - Synchronous
  - Asynchronous
- Failure model
  - Fail-stop
  - Crash
  - Byzantine
  - Permissionless Byzantine

Wrap-up

- Computation determinism
  - Deterministic
  - Probabilistic
- Timing assumptions
  - Synchronous
  - Asynchronous
- Failure model
  - Fail-stop
  - Crash
  - Byzantine
  - Permissionless Byzantine

Wrap-up

- Computation determinism
  - Deterministic
  - Probabilistic
- Timing assumptions
  - Synchronous
  - Asynchronous
- Failure model
  - Fail-stop
  - Crash
  - Byzantine
  - Permissionless Byzantine

Takeaway

You CANNOT guarantee safety and liveness at the same time!

Takeaway

You CANNOT guarantee safety and liveness at the same time!

But you CAN get around FLP:

1. Release the failure model

Release the failure model

Model:

1. The majority are non-faulty
2. No process dies during the execution of the protocol

Two-stage protocol:

1. Listens for messages from L-1 other processes, L=N/2+1 (WHY?), and construct the incoming stream graph G
2. Construct G^+ and make decision upon values from the unique initial clique

Takeaway

You CANNOT guarantee safety and liveness at the same time!

But you CAN get around FLP:

1. Release the failure model
2. Terminate with **probability of 1** instead of ALWAYS

Terminate with **probability of 1** instead of ALWAYS

Use randomization to terminate with arbitrarily high probability

M. Ben Or. “Another advantage of free choice: completely asynchronous agreement protocols” (PODC 1983, pp. 27-30)

Takeaway

You CANNOT guarantee safety and liveness at the same time!

But you CAN get around FLP:

1. Release the failure model
2. Terminate with **probability of 1** instead of ALWAYS
3. Use failure detector

Use failure detector

Introduce **failure detectors** to distinguish between crashed processes and very slow processes

Chandra, T.D., Hadzilacos, V. and Toueg, S., 1996. The weakest failure detector for solving consensus. Journal of the ACM (JACM), 43(4), pp.685-722.