Impossibility of Distributed Consensus with One Faulty Process

Michael J. Fischer, Nancy A. Lynch, Michael S. Paterson

Journal of ACM, 1985

Presented by Jialu Bao on CS 6410, Sept 22. 2022

Review from Last Lecture

Review from Last Lecture

- Consensus problem:
  - **Agreement**: if two processes decide, they must decide the same operation.
  - **Validity**: a process can only decide an operation proposed by some replica.

Review from Last Lecture

- Consensus problem:
    - **Agreement**: if two processes decide, they must decide the same operation.
    - **Validity**: a process can only decide an operation proposed by some replica.
- In an asynchronous system:
    - To tolerate f **crash** failures, we need at least $2f + 1$ processes.
        - Paxos meets the $2f + 1$ lower bound.
    - To tolerate f **byzantine** failures, we need at least $3f + 1$ processes.
        - We saw a protocol that works with $5f + 1$ processes.

Review from Last Lecture

- Consensus problem: + Termination?
  - **Agreement**: if two processes decide, they must decide the same operation.
  - **Validity**: a process can only decide an operation proposed by some replica.
- In an asynchronous system:
  - To tolerate f **crash** failures, we need at least ? processes.

This Paper

- Consensus problem: **+ Termination?**
  - **Agreement**: if two processes decide, they must decide the same operation.
  - **Validity**: a process can only decide an operation proposed by some replica.
- In an asynchronous system:
  - To tolerate f **crash** failures, we need at least **?** processes.

Impossible!

Computation Model

Computation Model

- Every process starts with an initial value in (0,1).

Computation Model

- Every process starts with an initial value in (0,1).

Computation Model

- Every process starts with an initial value in (0,1).

	P1	P2	P3	P4
Processes	init 0	init 1	init 0	init 1

- One process may die (stop entirely) at some point.

Computation Model

- Every process starts with an initial value in (0,1).

- One process may die (stop entirely) at some point.
- A non-faulty process may **decide** on a value in (0, 1).

Computation Model

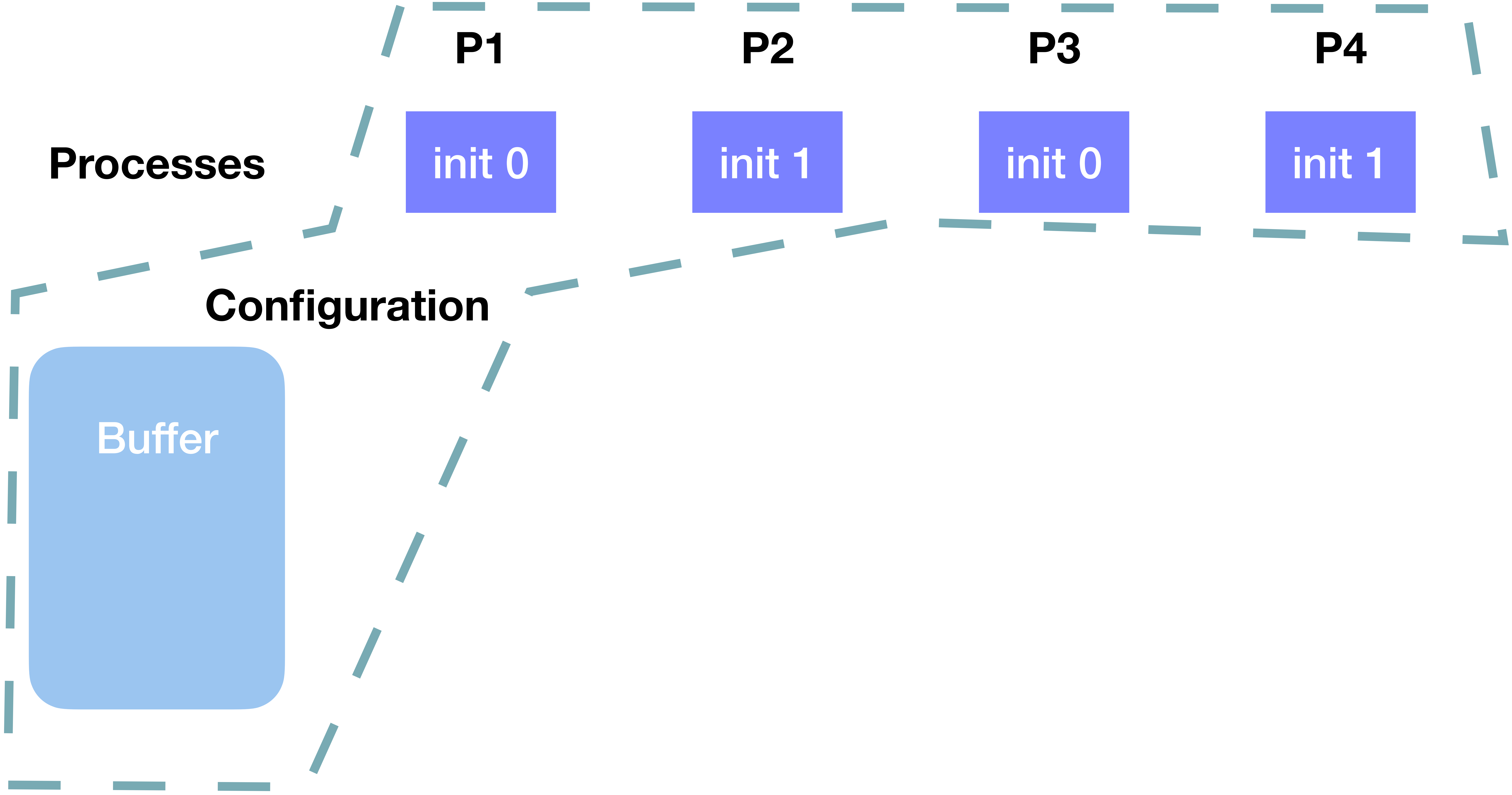
- Every process starts with an initial value in (0,1).

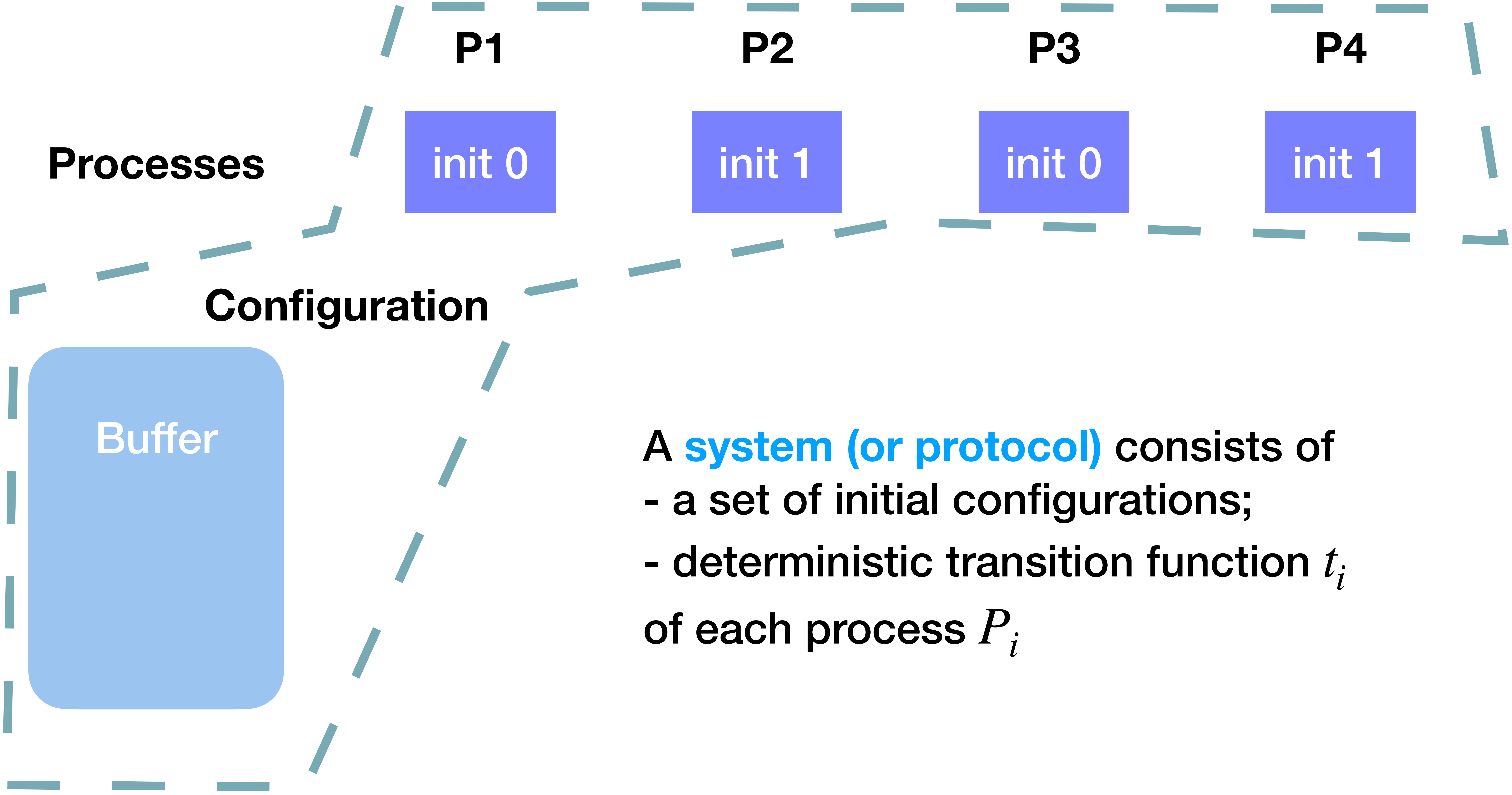
	P1	P2	P3	P4
Processes	init 0	init 1	init 0	init 1

- One process may die (stop entirely) at some point.
- A non-faulty process may **decide** on a value in (0, 1).

Steps after	died	init 1 dec 1	init 0 dec 1	init 1 dec ?

	P1	P2	P3	P4
Processes	init 0	init 1	init 0	init 1


P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
Configuration
Buffer



A **system (or protocol)** consists of
- a set of initial configurations;
- deterministic transition function t_i of each process P_i

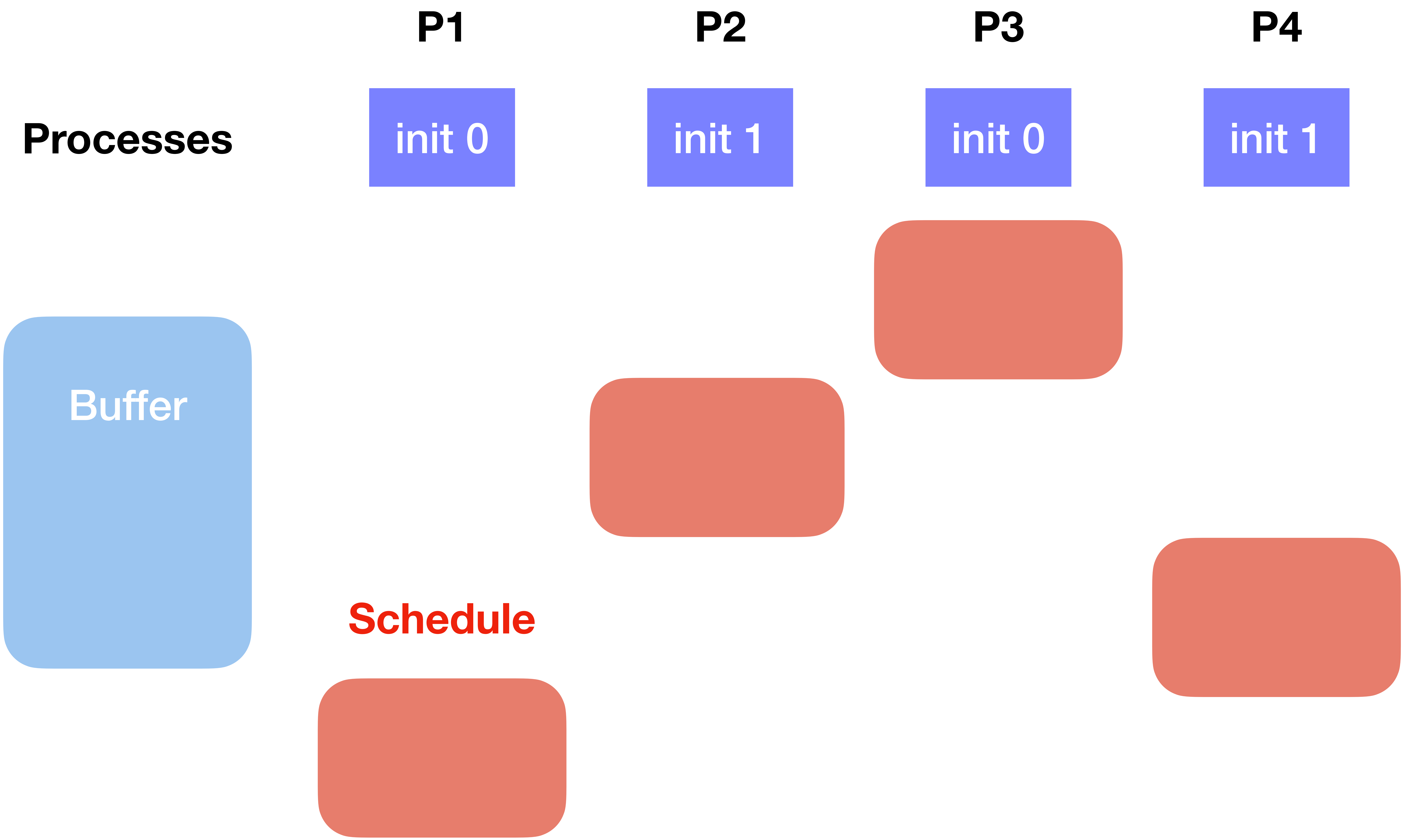
	P1	P2	P3	P4
Processes	init 0	init 1	init 0	init 1

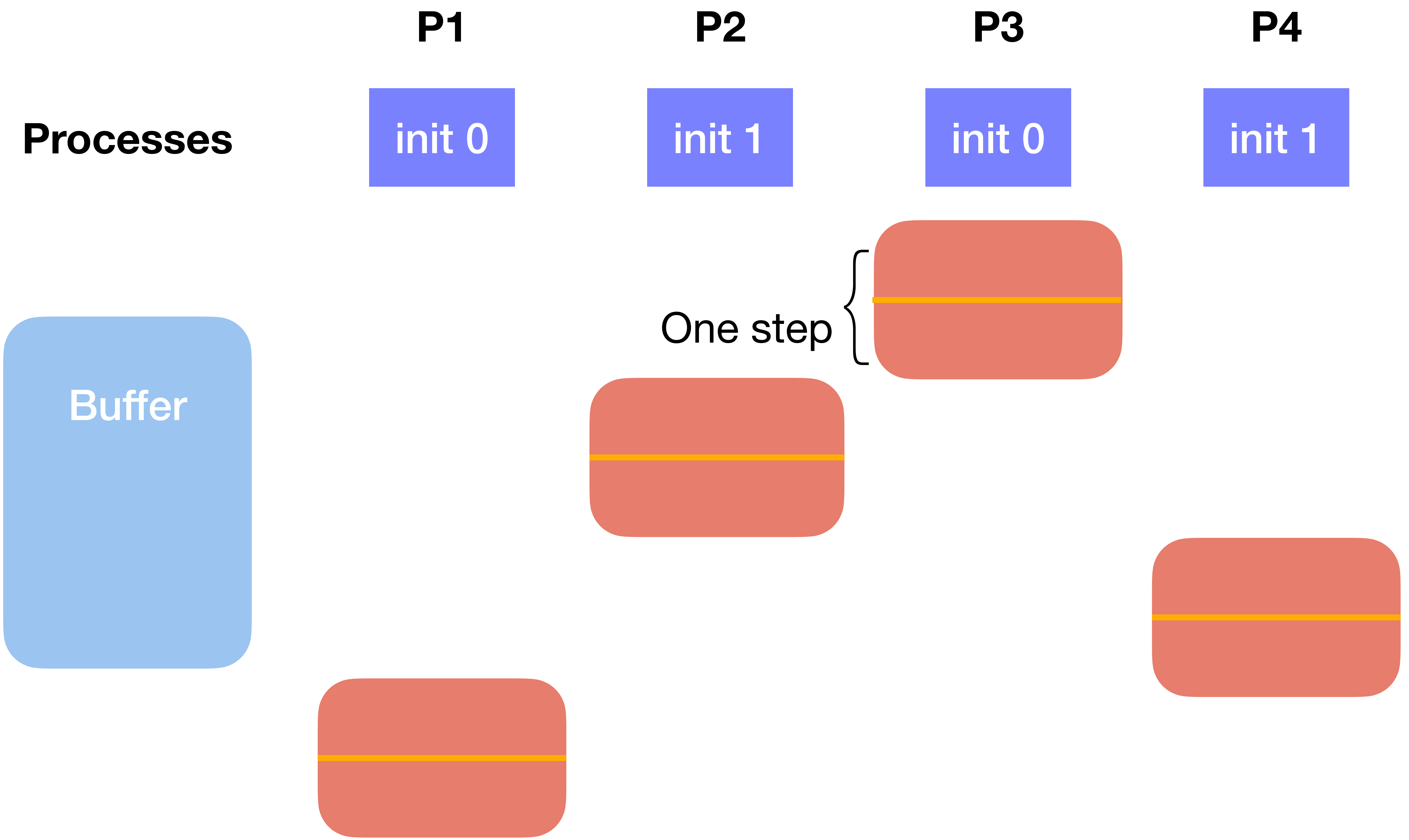
Buffer

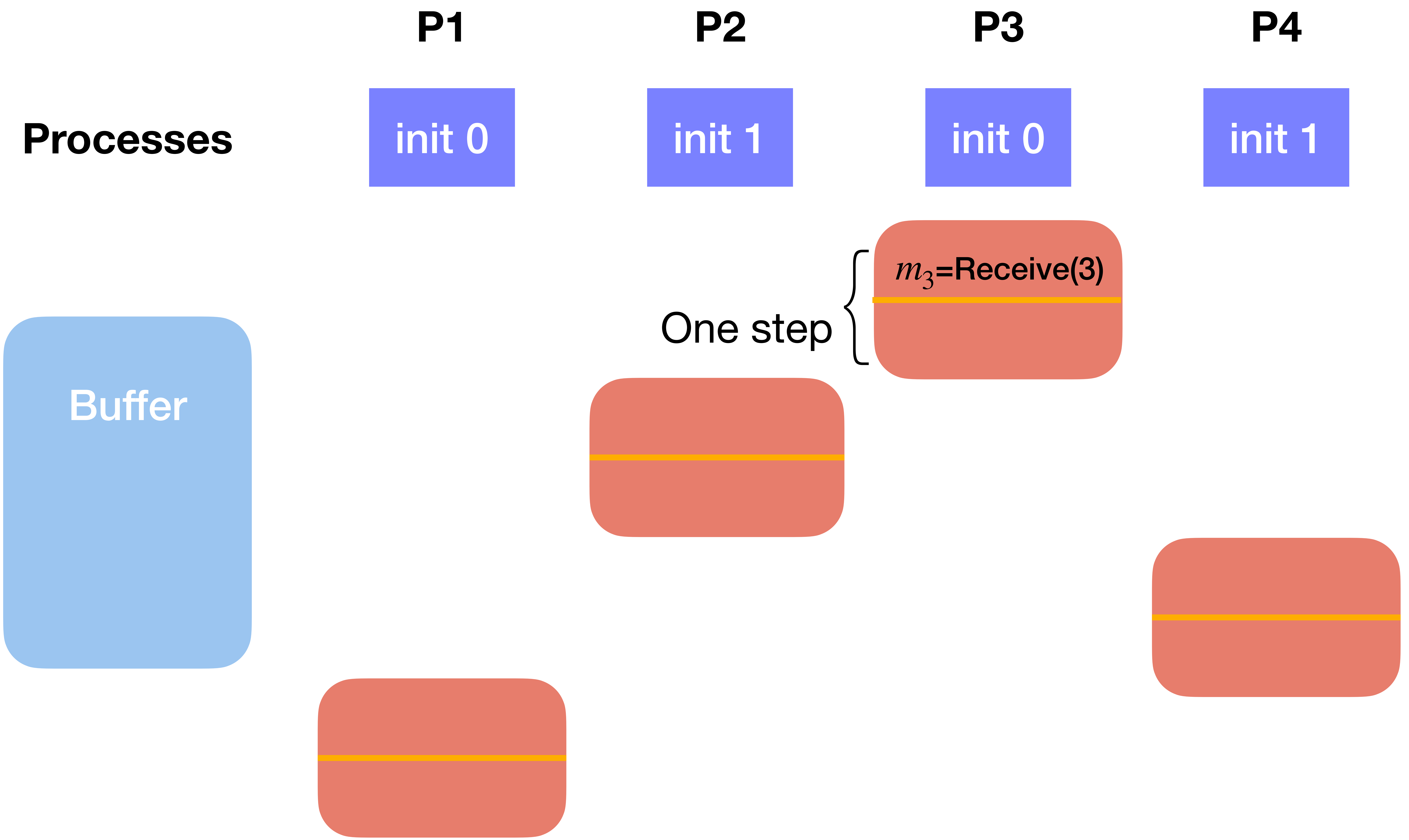
	P1	P2	P3	P4
Processes	init 0	init 1	init 0	init 1

Buffer

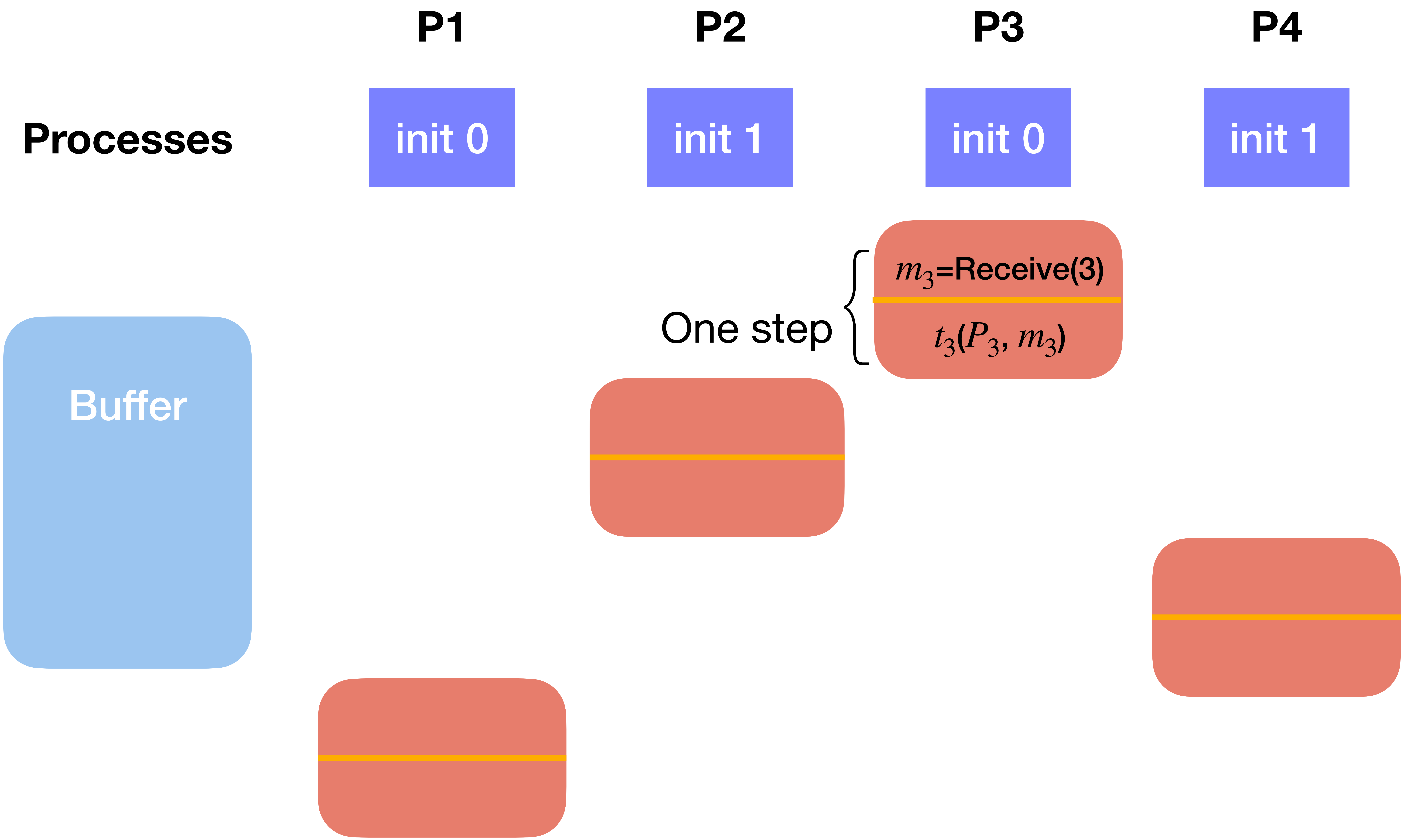
Schedule


P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
Buffer
Schedule

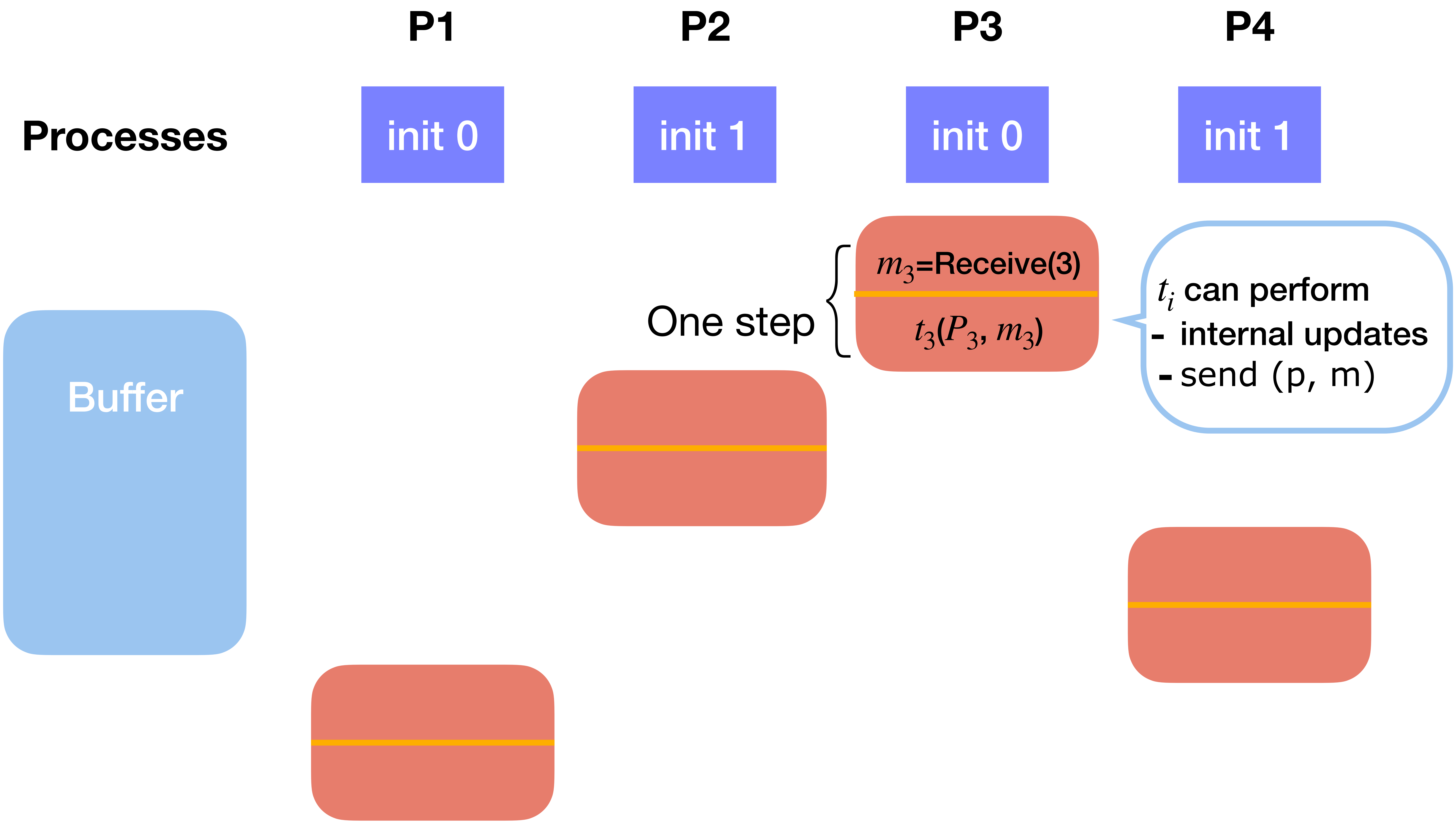

P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
One step
Buffer

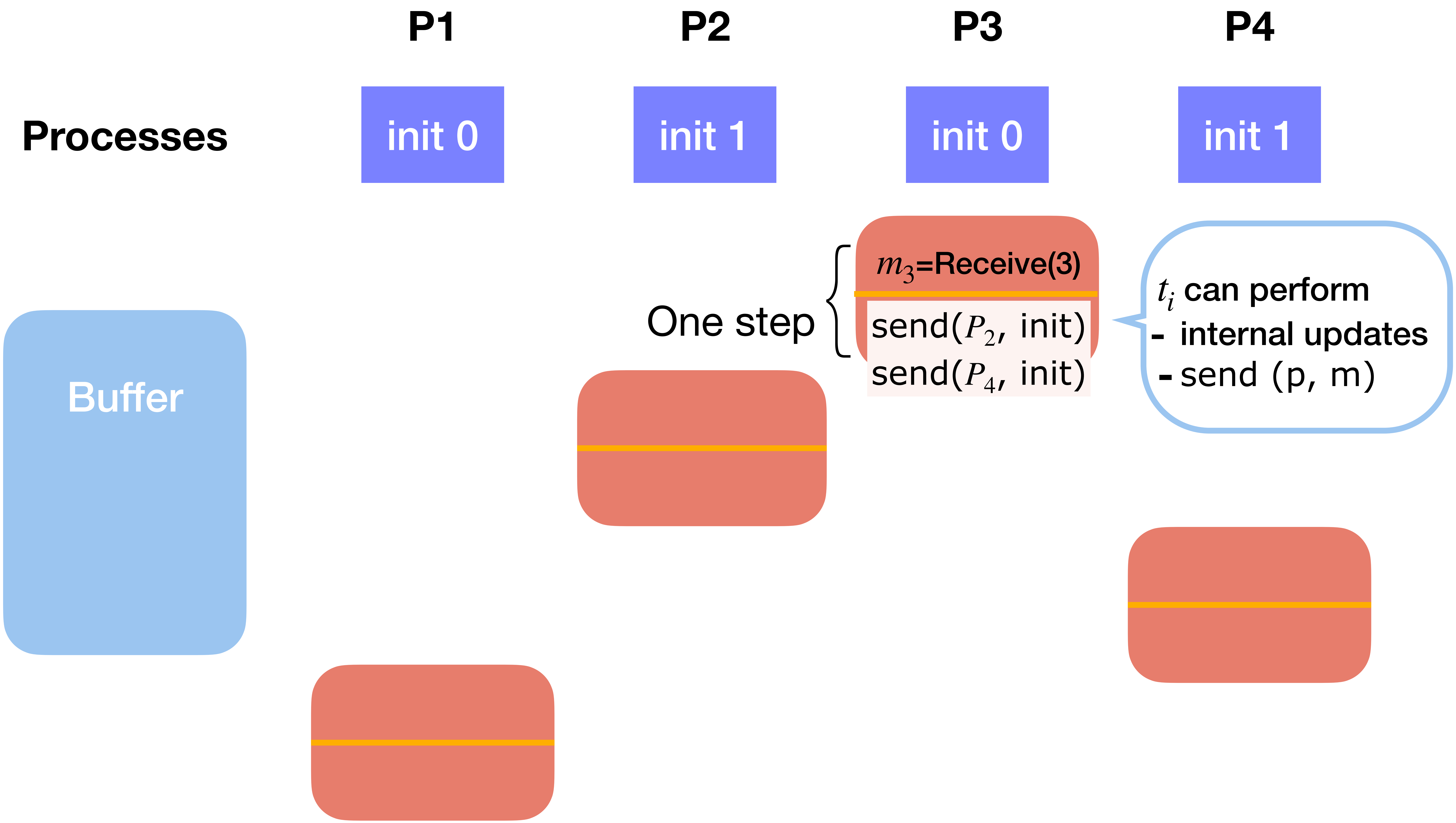


Processes
P1
P2
P3
P4
init 0
init 1
init 0
init 1
m_3=Receive(3)
One step
Buffer

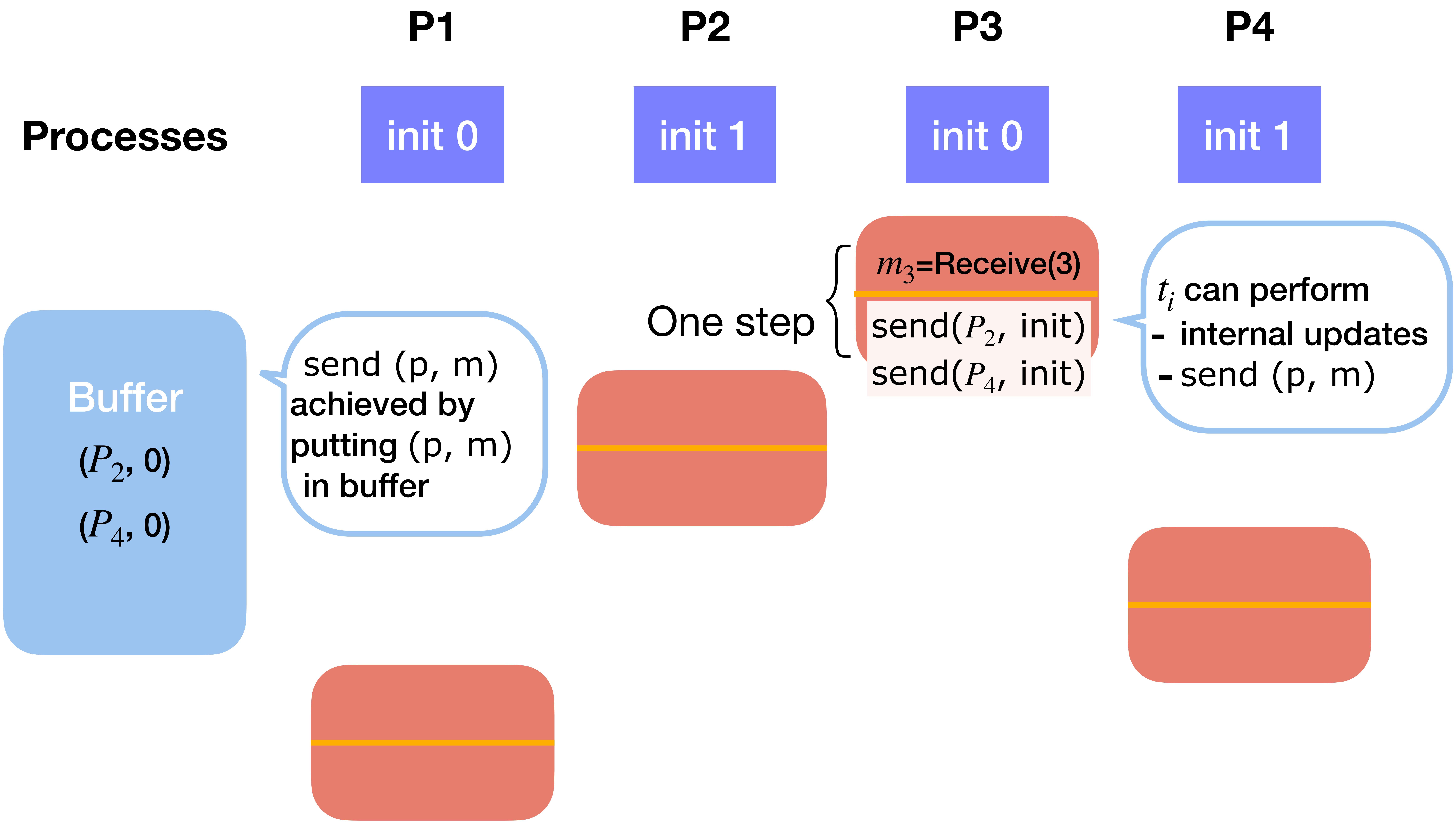


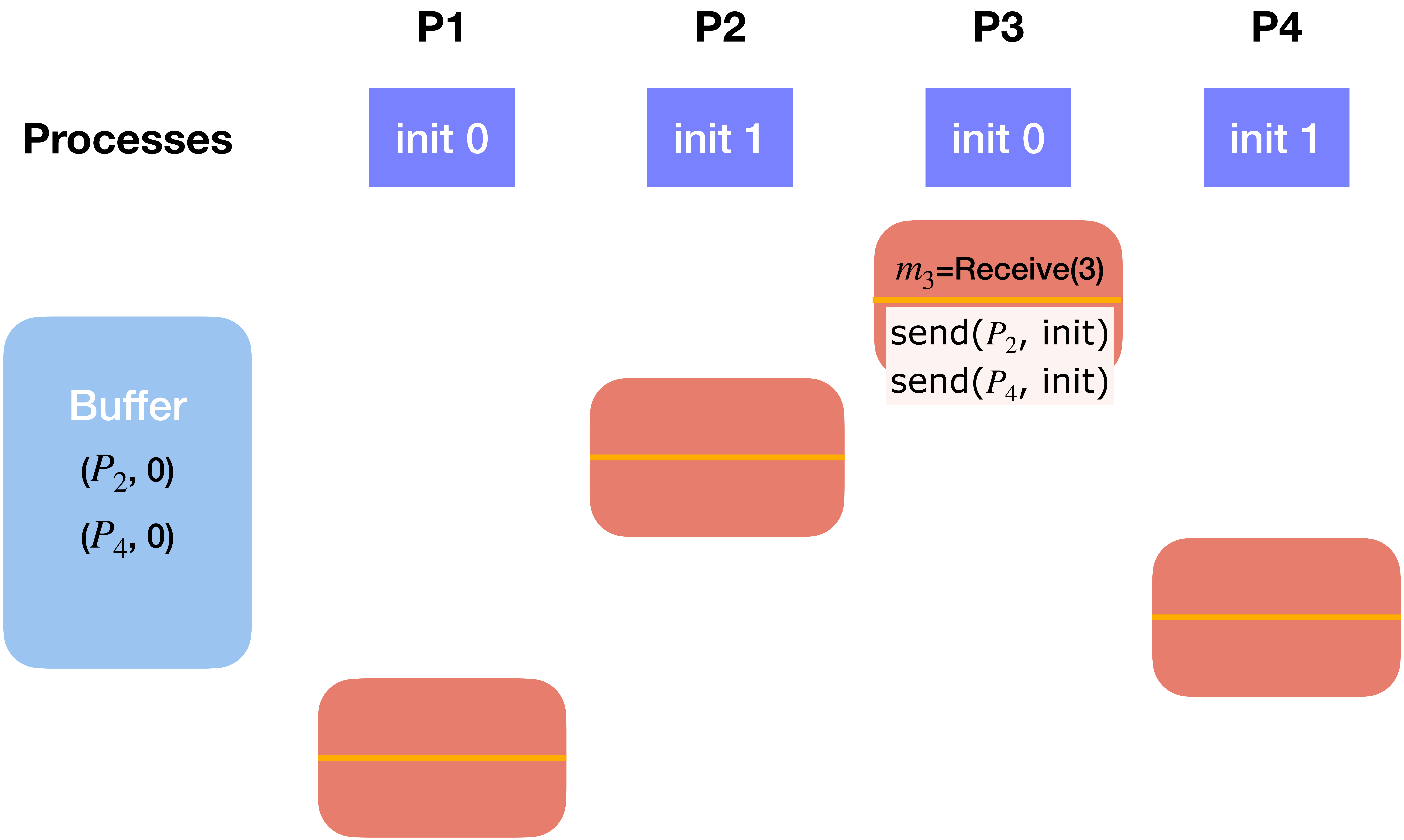
P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
m_3=Receive(3)
One step
$t_3(P_3, m_3)$
Buffer


P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
m_3=Receive(3)
$t_3(P_3, m_3)$
One step
t_i can perform
- internal updates
- send (p, m)
Buffer

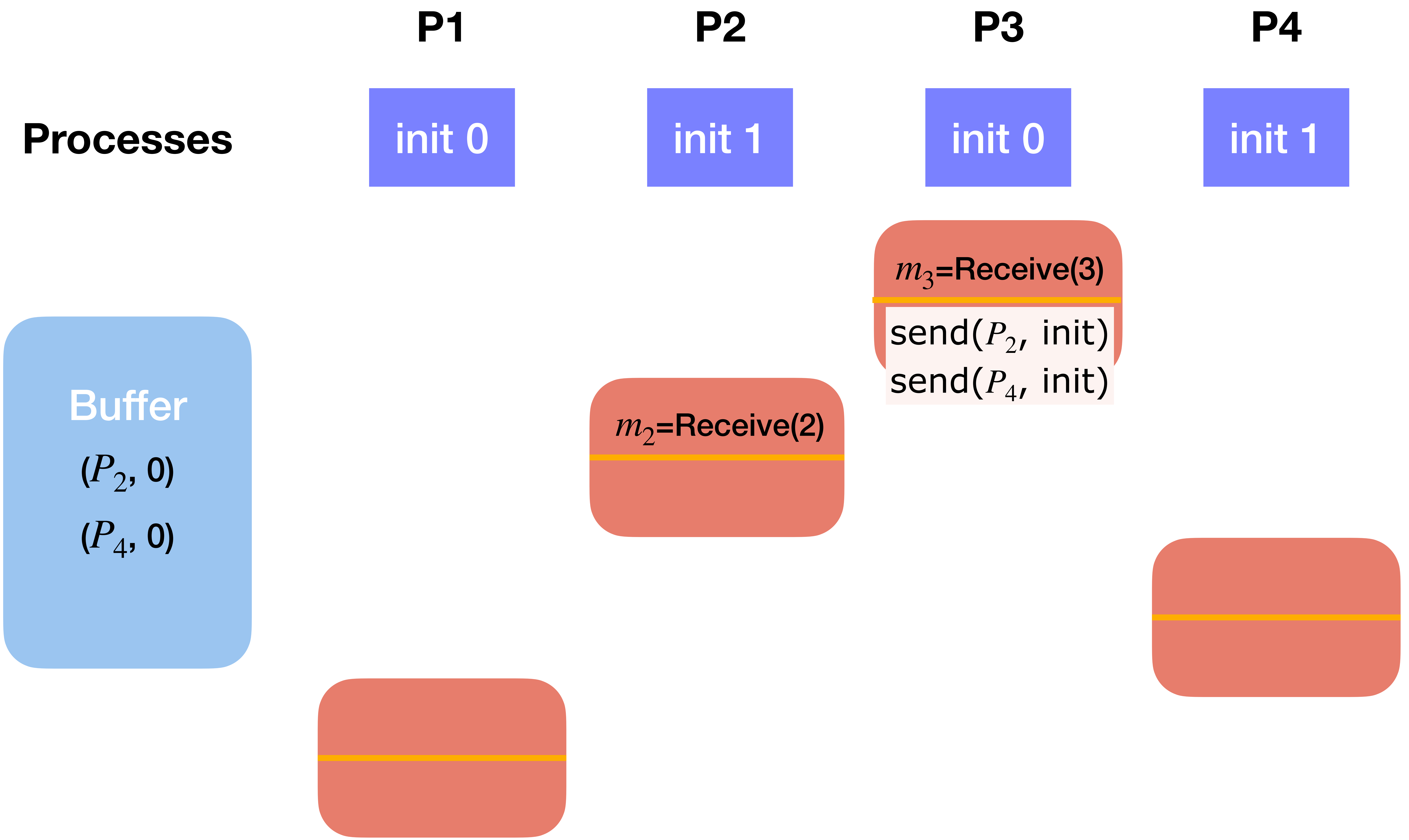


P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
Buffer
m_3=Receive(3)
One step
send(P_2, init)
send(P_4, init)
t_i can perform
- internal updates
- send (p, m)

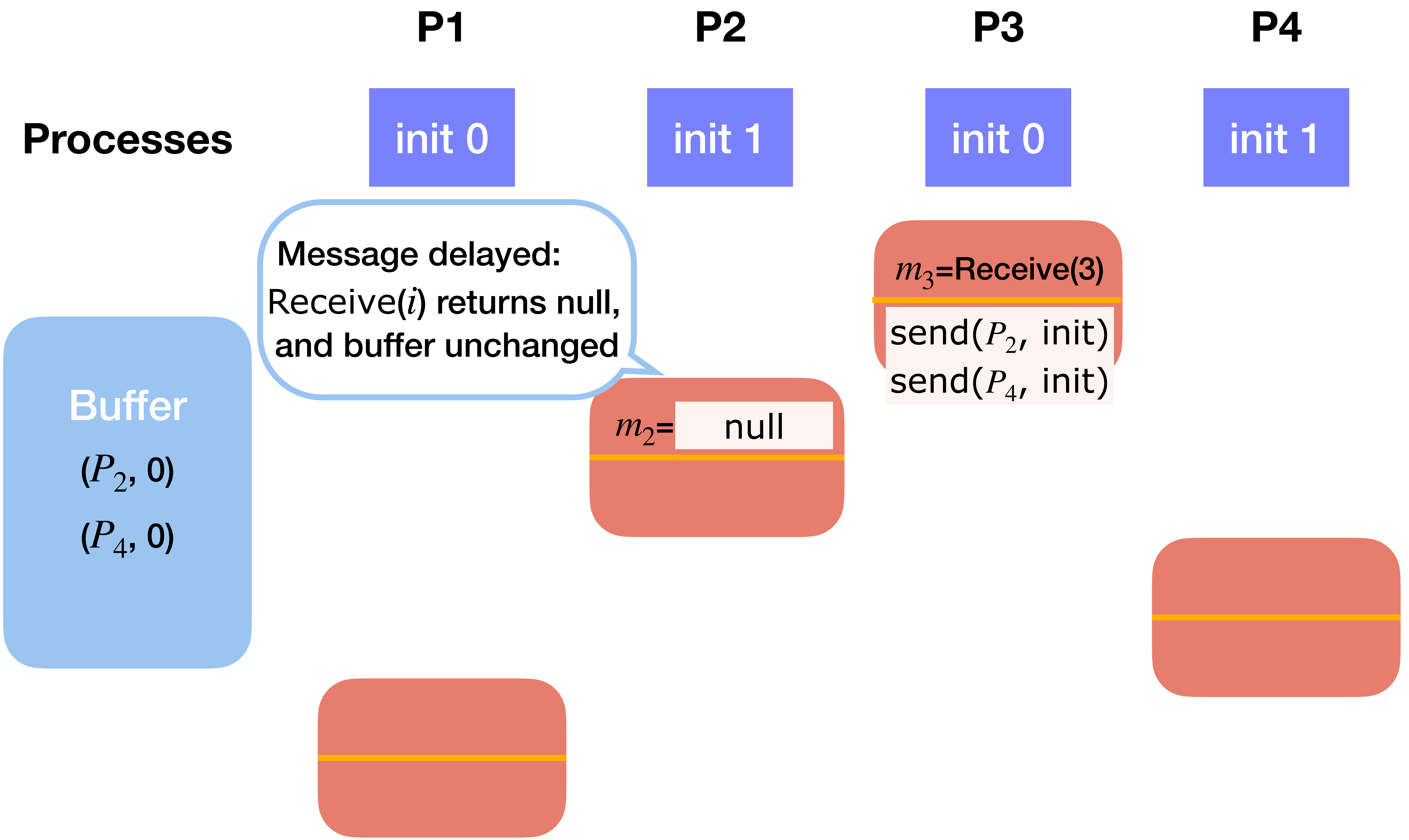

P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
Buffer
$(P_2, 0)$
$(P_4, 0)$
send (p, m) achieved by putting (p, m) in buffer
One step
m_3=Receive(3)
send(P_2, init)
send(P_4, init)
t_i can perform
- internal updates
- send (p, m)

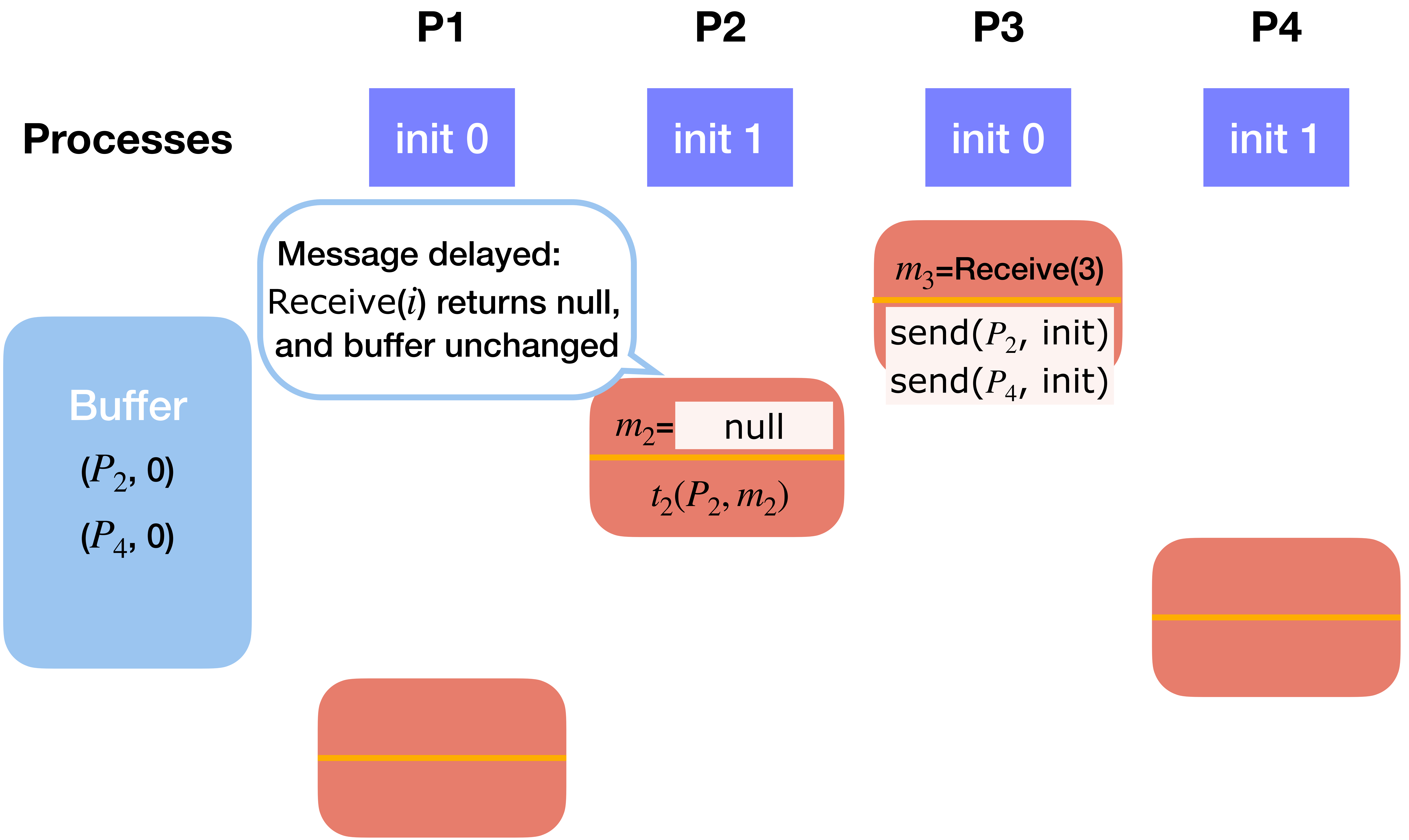


P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
m_3=Receive(3)
send(P_2, init)
send(P_4, init)
Buffer
(P_2, 0)
(P_4, 0)

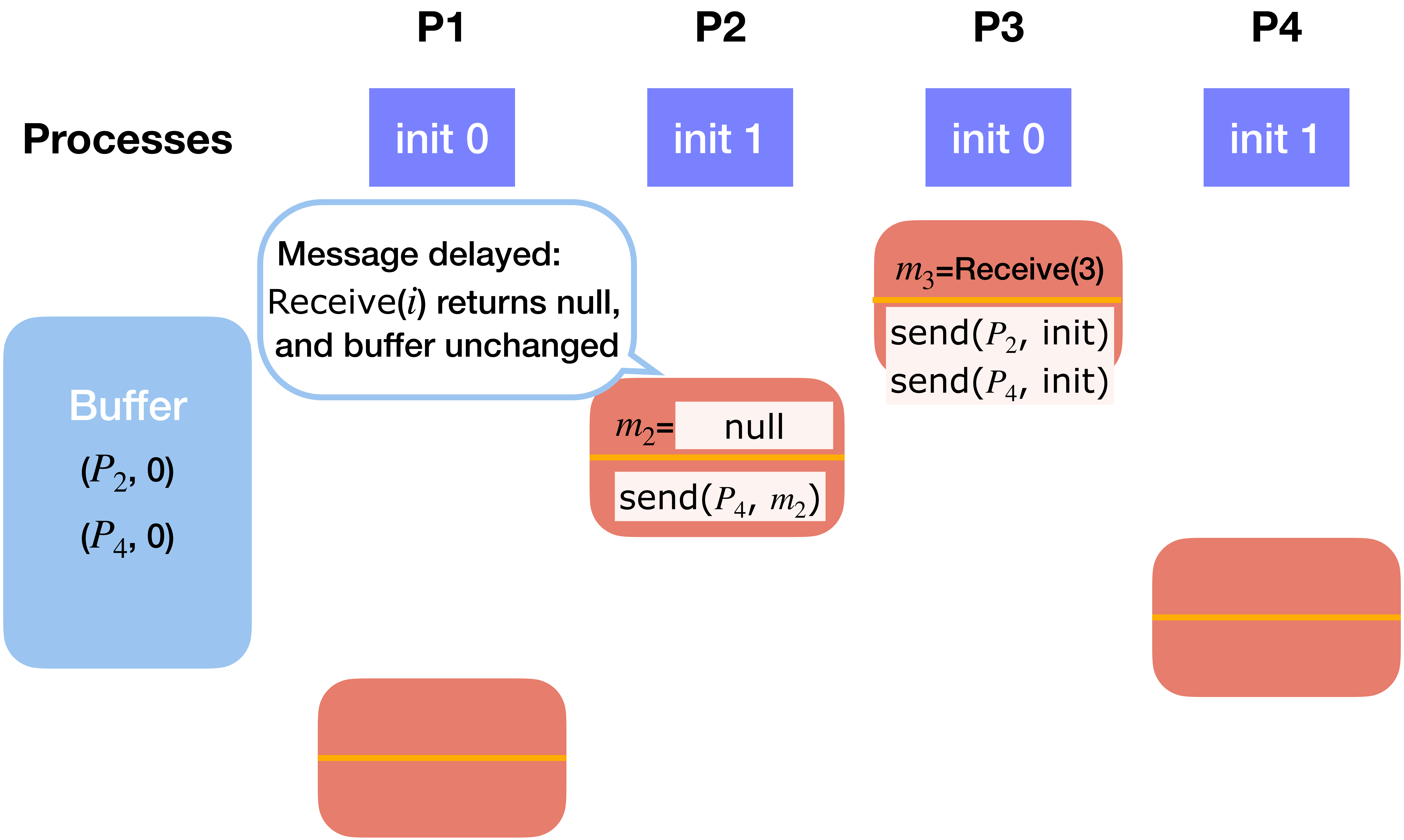


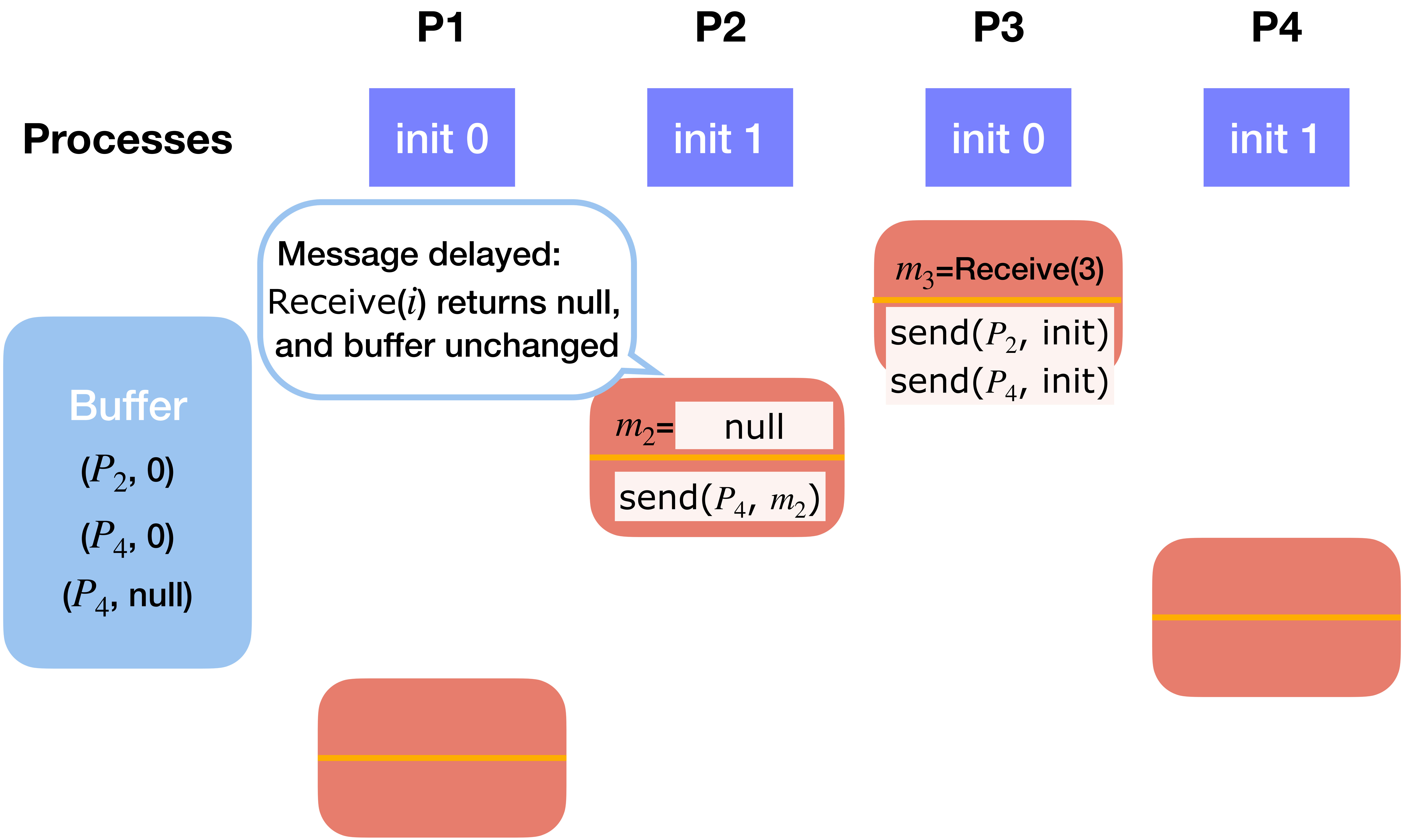
P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
m_3=Receive(3)
send(P_2, init)
send(P_4, init)
Buffer
(P_2, 0)
(P_4, 0)
m_2=Receive(2)

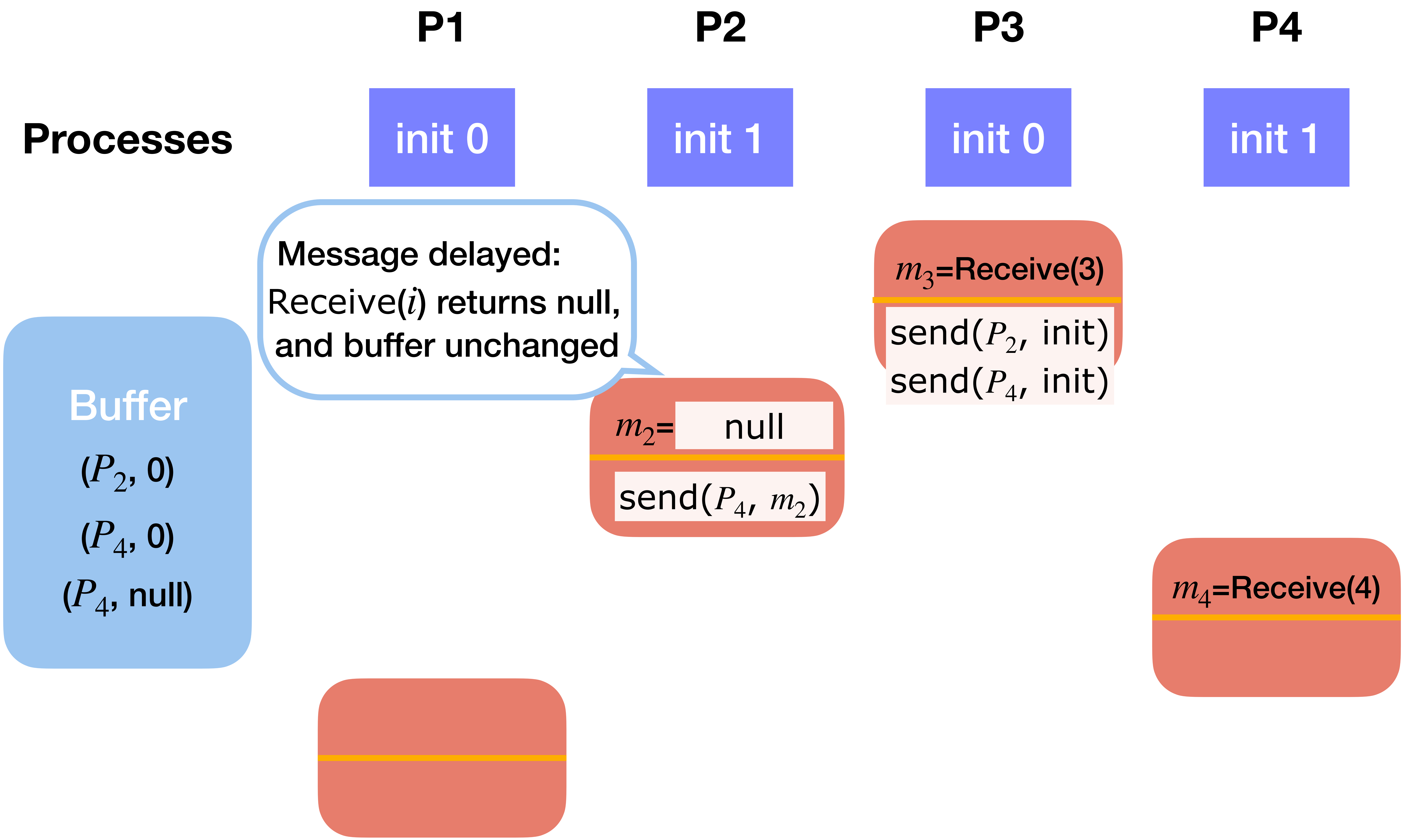

P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
Message delayed:
Receive(i) returns null,
and buffer unchanged
m_3=Receive(3)
send(P_2, init)
send(P_4, init)
m_2= null
Buffer
(P_2, 0)
(P_4, 0)



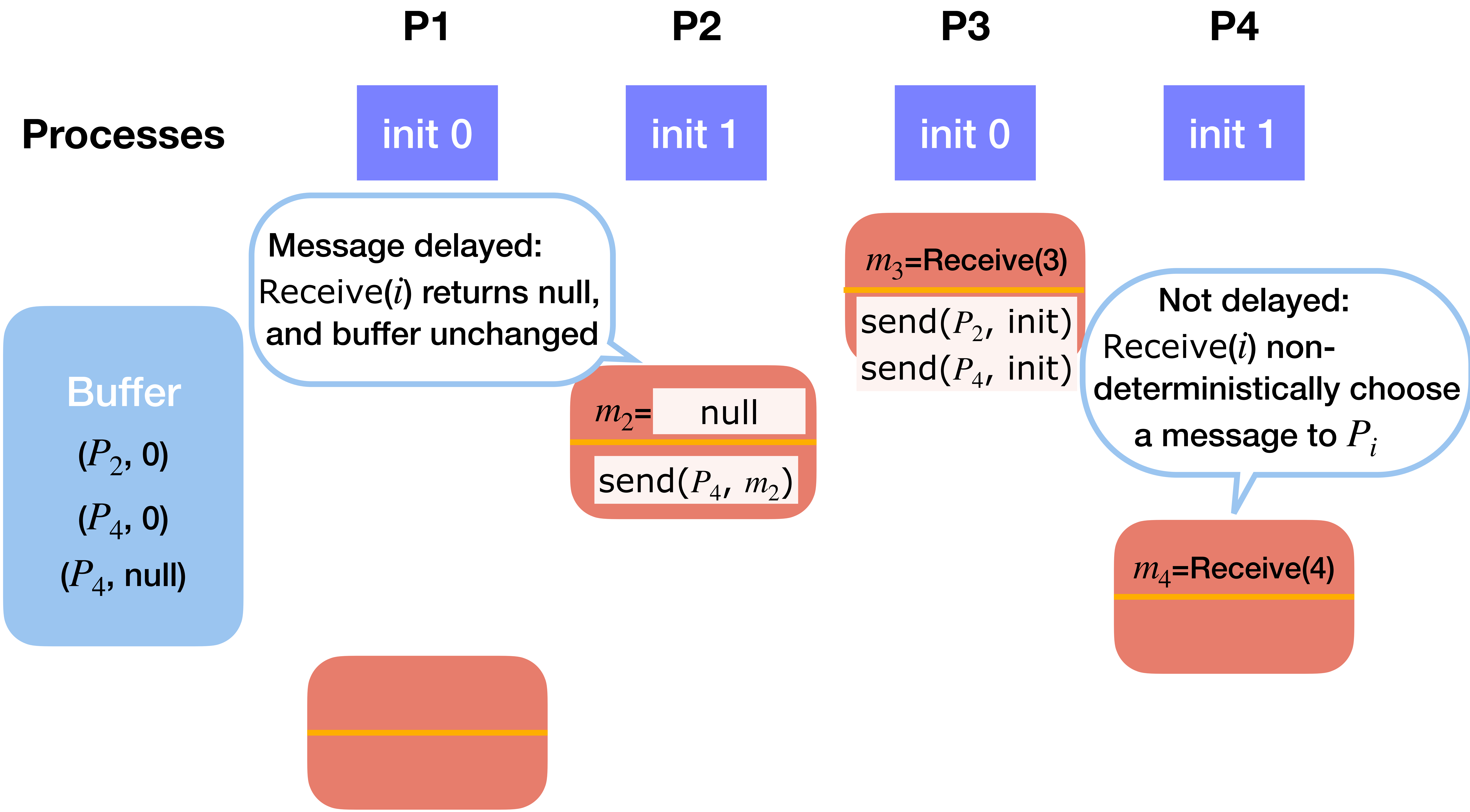
P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
Message delayed:
Receive(i) returns null,
and buffer unchanged
m_3=Receive(3)
send(P_2, init)
send(P_4, init)
Buffer
(P_2, 0)
(P_4, 0)
m_2= null
$t_2(P_2, m_2)$


P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
Message delayed:
Receive(i) returns null,
and buffer unchanged
Buffer
(P_2, 0)
(P_4, 0)
m_2= null
send(P_4, m_2)
m_3=Receive(3)
send(P_2, init)
send(P_4, init)

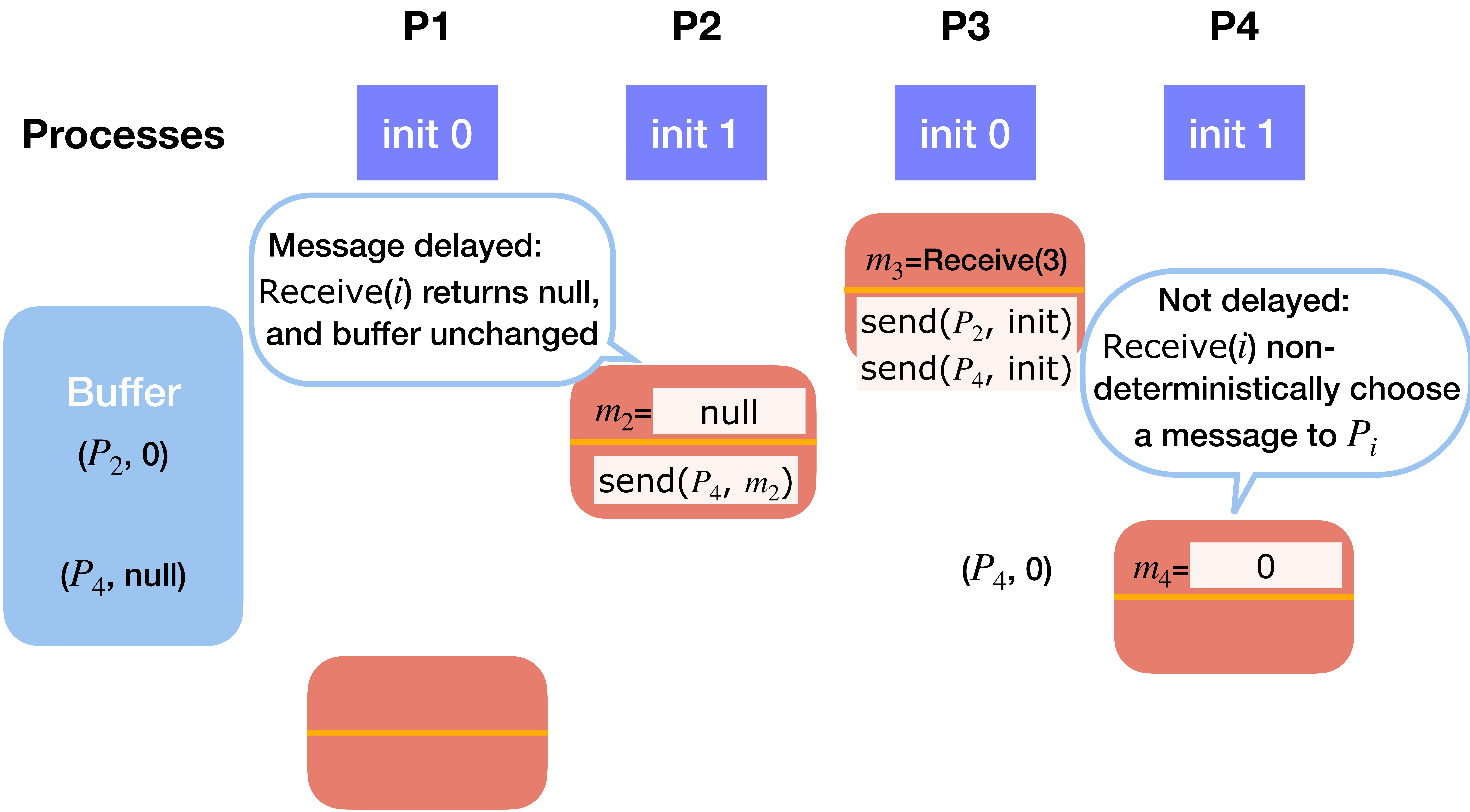

P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
Message delayed:
Receive(i) returns null,
and buffer unchanged
m_3=Receive(3)
send(P_2, init)
send(P_4, init)
m_2= null
send(P_4, m_2)
Buffer
(P_2, 0)
(P_4, 0)
(P_4, null)

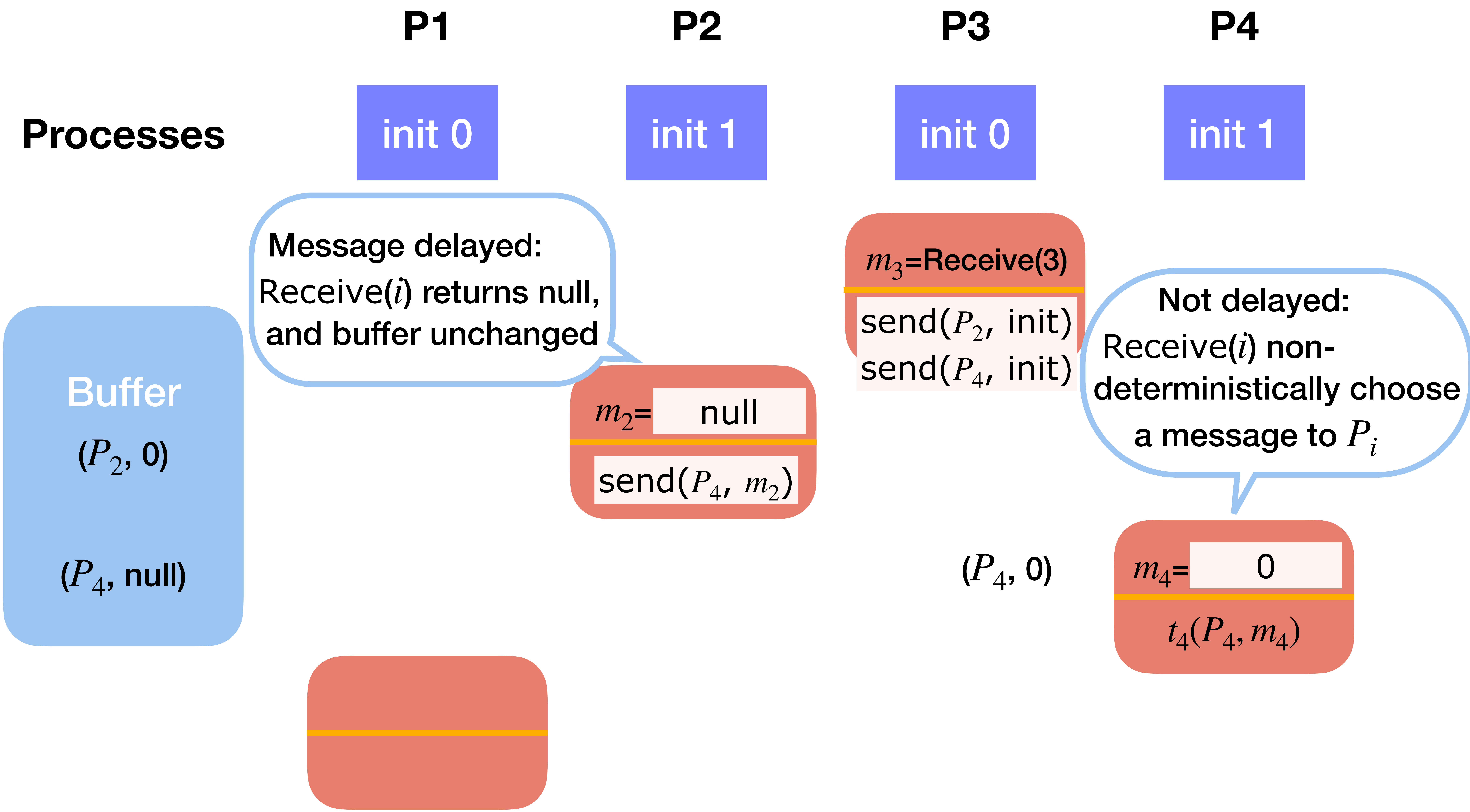


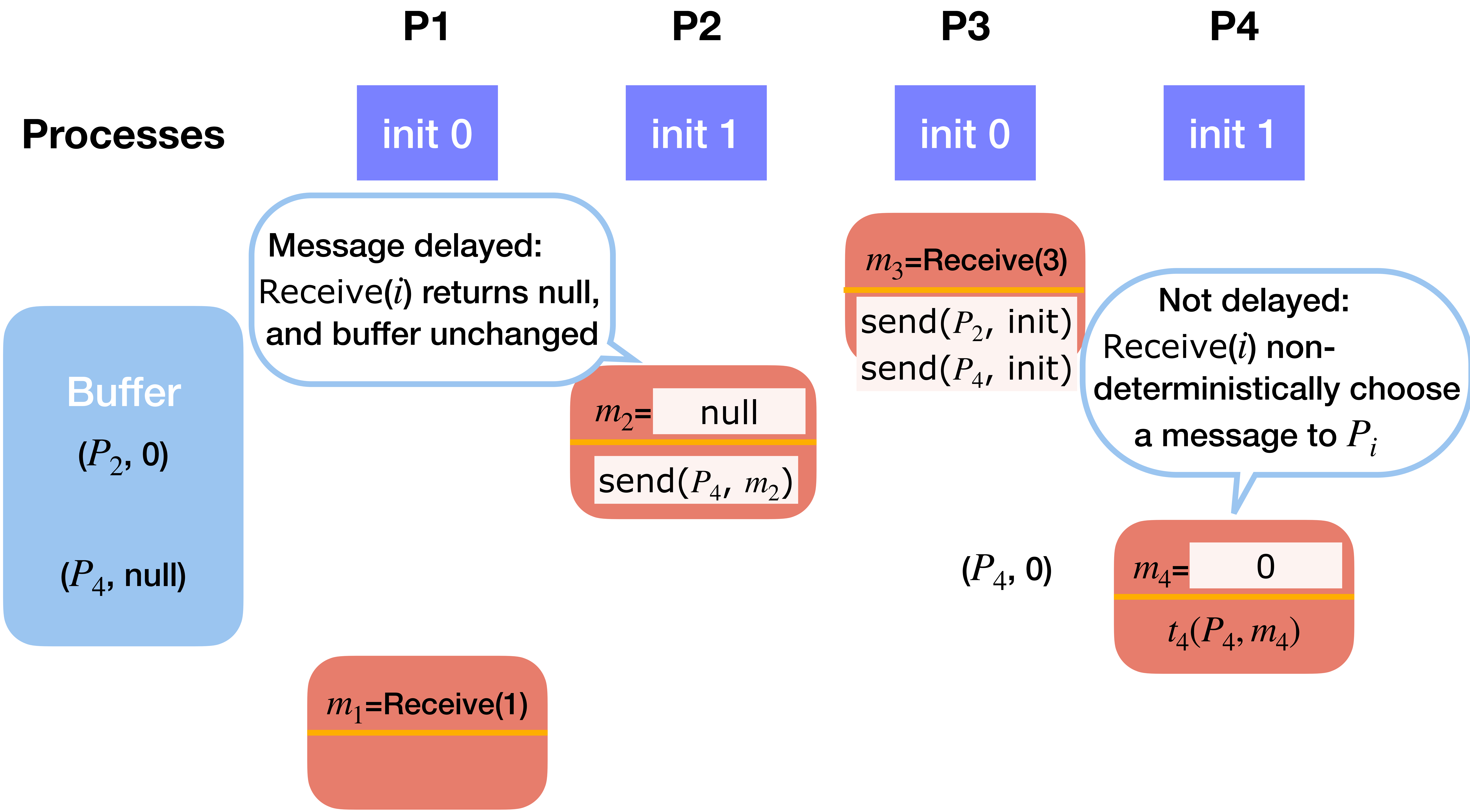
P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
Message delayed:
Receive(i) returns null,
and buffer unchanged
m_3=Receive(3)
send(P_2, init)
send(P_4, init)
Buffer
(P_2, 0)
(P_4, 0)
(P_4, null)
m_2= null
send(P_4, m_2)
m_4=Receive(4)

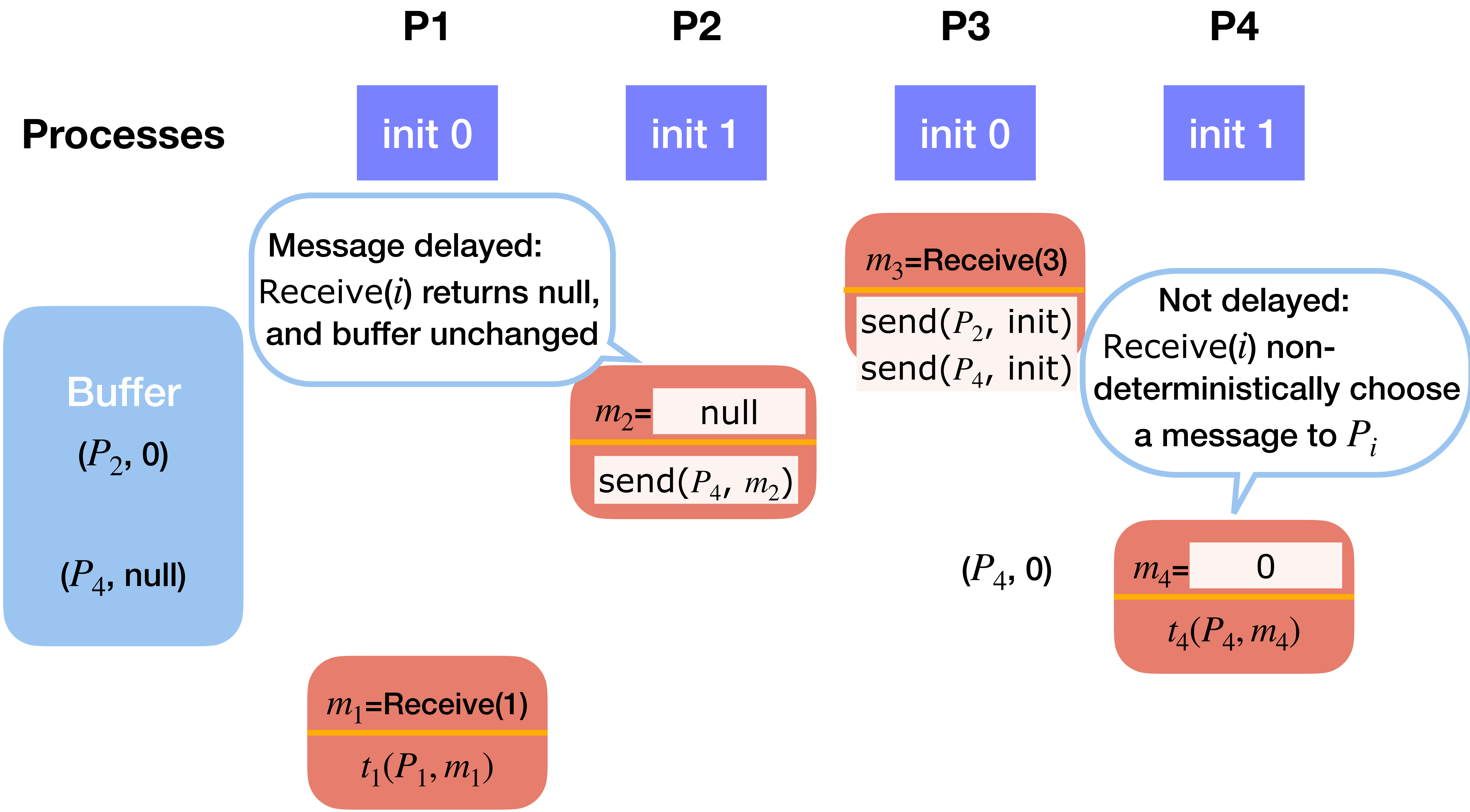


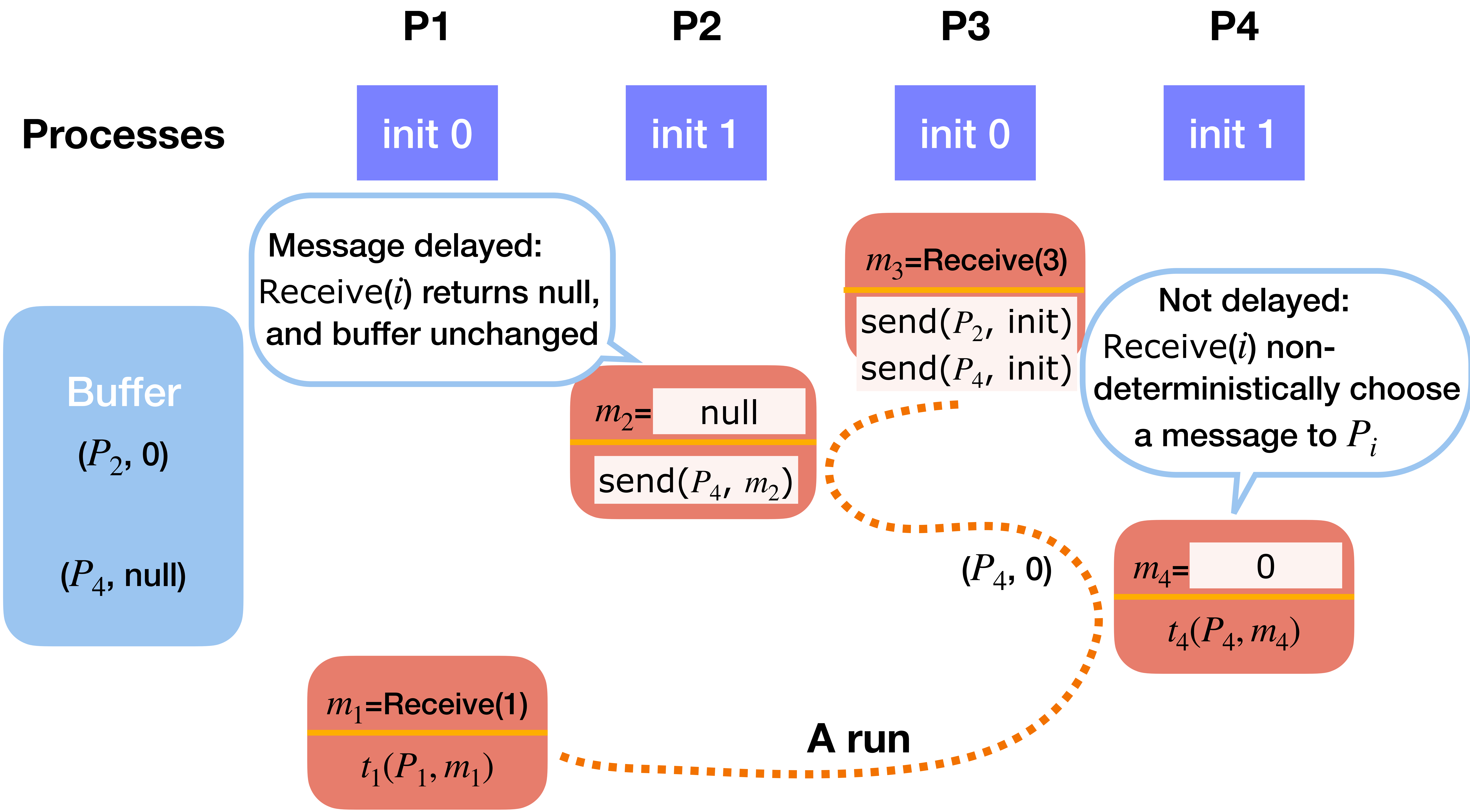
P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
Message delayed:
Receive(i) returns null,
and buffer unchanged
m_3=Receive(3)
send(P_2, init)
send(P_4, init)
Not delayed:
Receive(i) non-deterministically choose a message to P_i
Buffer
(P_2, 0)
(P_4, 0)
(P_4, null)
m_2= null
send(P_4, m_2)
m_4=Receive(4)


P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
Message delayed:
Receive(i) returns null,
and buffer unchanged
m_3=Receive(3)
send(P_2, init)
send(P_4, init)
Not delayed:
Receive(i) non-
deterministically choose
a message to P_i
Buffer
(P_2, 0)
(P_4, null)
m_2= null
send(P_4, m_2)
(P_4, 0)
m_4= 0


P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
Message delayed: Receive(i) returns null, and buffer unchanged
m_3=Receive(3)
send(P_2, init)
send(P_4, init)
Not delayed: Receive(i) non-deterministically choose a message to P_i
Buffer
(P_2, 0)
(P_4, null)
m_2= null
send(P_4, m_2)
(P_4, 0)
m_4= 0
$t_4(P_4, m_4)$


P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
Message delayed:
Receive(i) returns null,
and buffer unchanged
m_3=Receive(3)
send(P_2, init)
send(P_4, init)
Not delayed:
Receive(i) non-deterministically choose a message to P_i
Buffer
(P_2, 0)
(P_4, null)
m_2= null
send(P_4, m_2)
(P_4, 0)
m_4= 0
$t_4(P_4, m_4)$
m_1=Receive(1)


P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
Message delayed:
Receive(i) returns null,
and buffer unchanged
m_3=Receive(3)
send(P_2, init)
send(P_4, init)
Not delayed:
Receive(i) non-deterministically choose a message to P_i
Buffer
(P_2, 0)
(P_4, null)
m_2= null
send(P_4, m_2)
(P_4, 0)
m_4= 0
$t_4(P_4, m_4)$
m_1=Receive(1)
$t_1(P_1, m_1)$


P1
P2
P3
P4
Processes
init 0
init 1
init 0
init 1
Message delayed: Receive(i) returns null, and buffer unchanged
m_3=Receive(3)
send(P_2, init)
send(P_4, init)
Not delayed: Receive(i) non-deterministically choose a message to P_i
Buffer
(P_2, 0)
(P_4, null)
m_2= null
send(P_4, m_2)
(P_4, 0)
m_4= 0
$t_4(P_4, m_4)$
m_1=Receive(1)
$t_1(P_1, m_1)$
A run

Total Correctness

Total Correctness

- C' is **accessible** in a system P if C' is reachable from an initial configuration C in P.

Total Correctness

- C' is **accessible** in a system P if C' is reachable from an initial configuration C in P.
- A run is **admissible** if ≤ 1 process is faulty and all messages sent to non-faulty processes are eventually delivered.

Total Correctness

- C' is **accessible** in a system P if C' is reachable from an initial configuration C in P.
- A run is **admissible** if ≤ 1 process is faulty and all messages sent to non-faulty processes are eventually delivered.
- A system P is **total correct in spite of one fault** if

Total Correctness

- C' is **accessible** in a system P if C' is reachable from an initial configuration C in P.
- A run is **admissible** if ≤ 1 process is faulty and all messages sent to non-faulty processes are eventually delivered.
- A system P is **total correct in spite of one fault** if

Termination: in any admissible run, some processes eventually make decisions.

Total Correctness

- C' is **accessible** in a system P if C' is reachable from an initial configuration C in P.
- A run is **admissible** if ≤ 1 process is faulty and all messages sent to non-faulty processes are eventually delivered.
- A system P is **total correct in spite of one fault** if

Termination: in any admissible run, some processes eventually make decisions.

Agreement: in any accessible configuration, all decided processes agree.

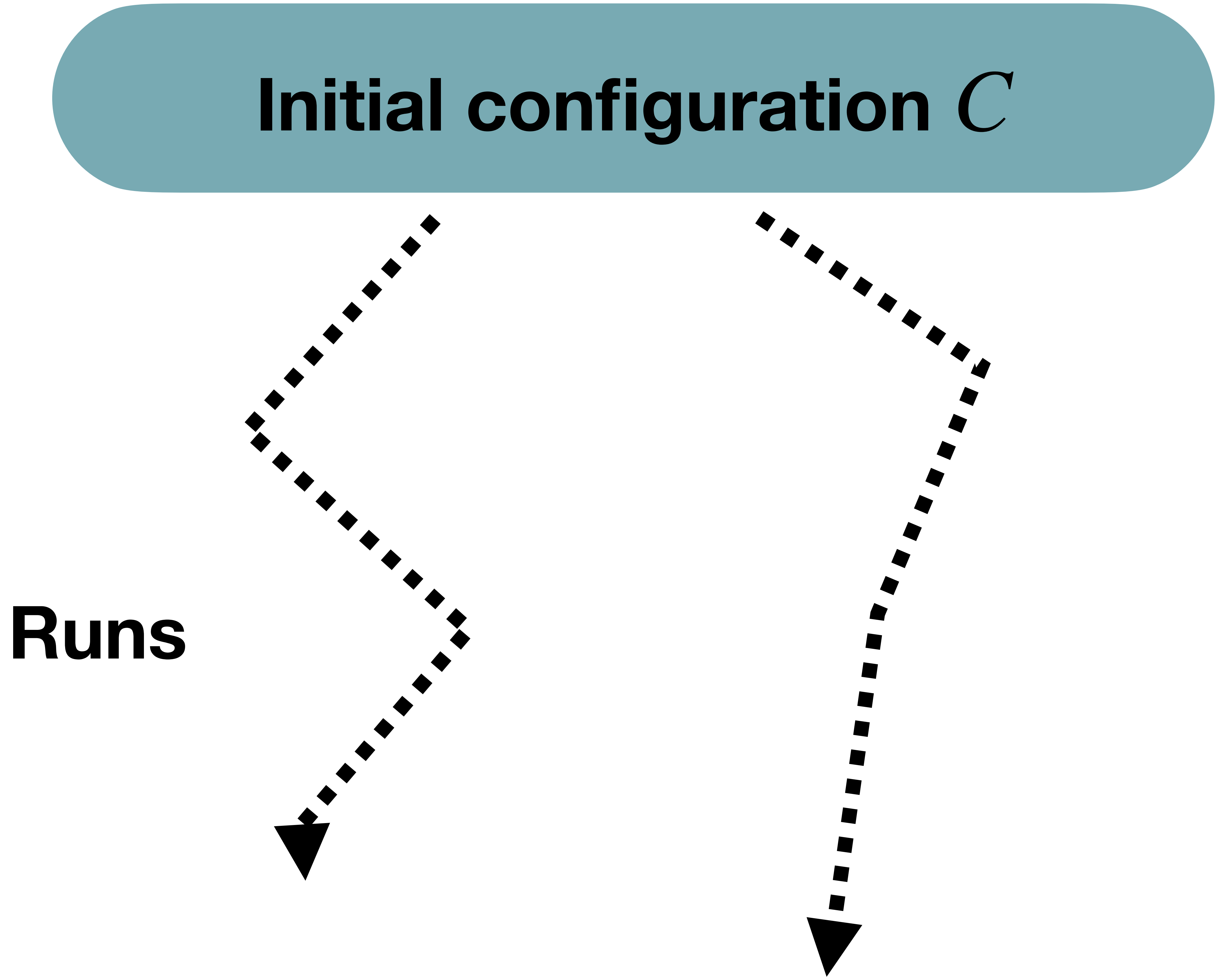
Total Correctness

- C' is **accessible** in a system P if C' is reachable from an initial configuration C in P.
- A run is **admissible** if ≤ 1 process is faulty and all messages sent to non-faulty processes are eventually delivered.
- A system P is **total correct in spite of one fault** if

Termination: in any admissible run, some processes eventually make decisions.

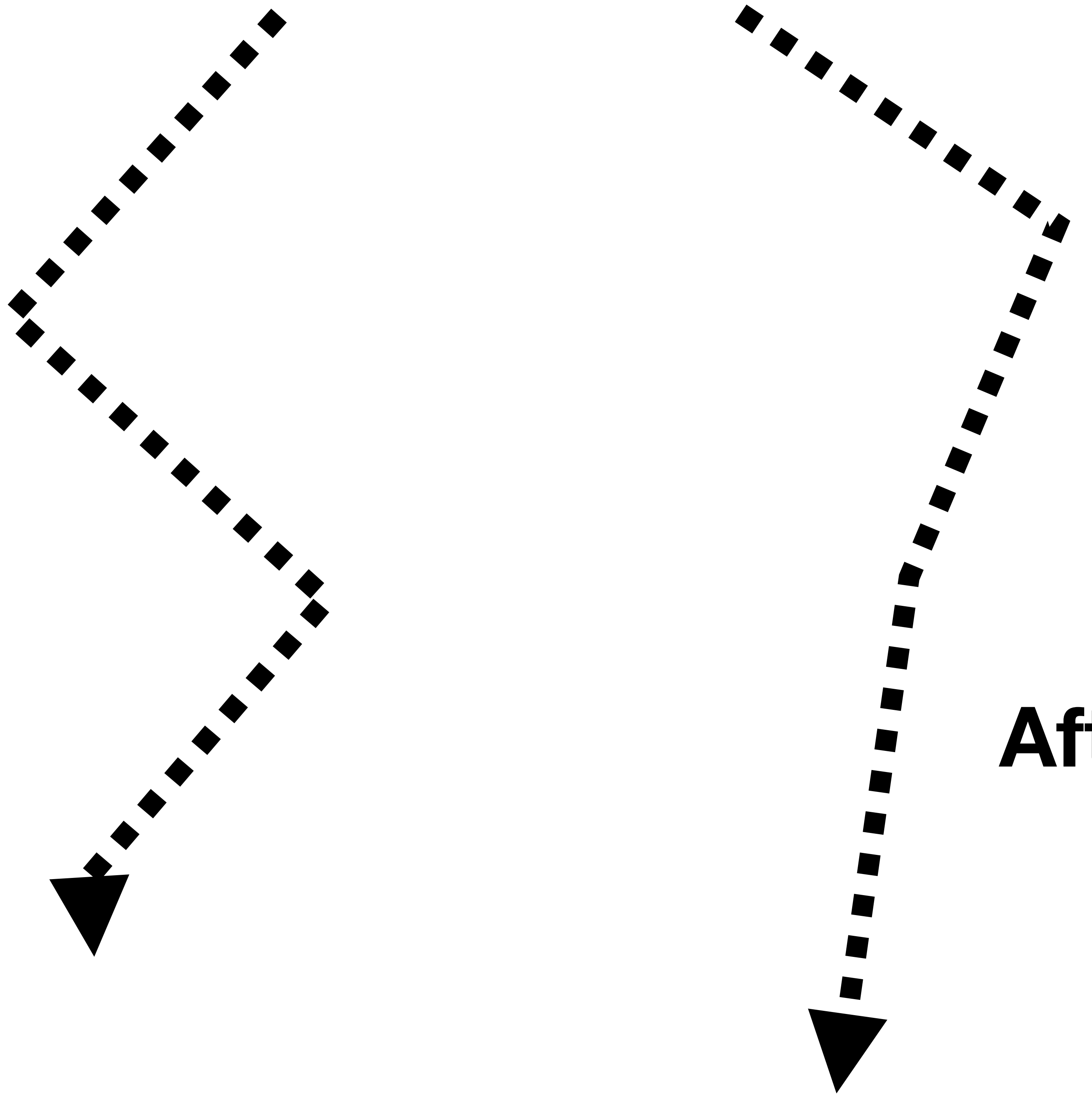
Agreement: in any accessible configuration, all decided processes agree.

Non-trivial: For $i \in \{0,1\}$, exists an accessible configuration in P that agrees on i.

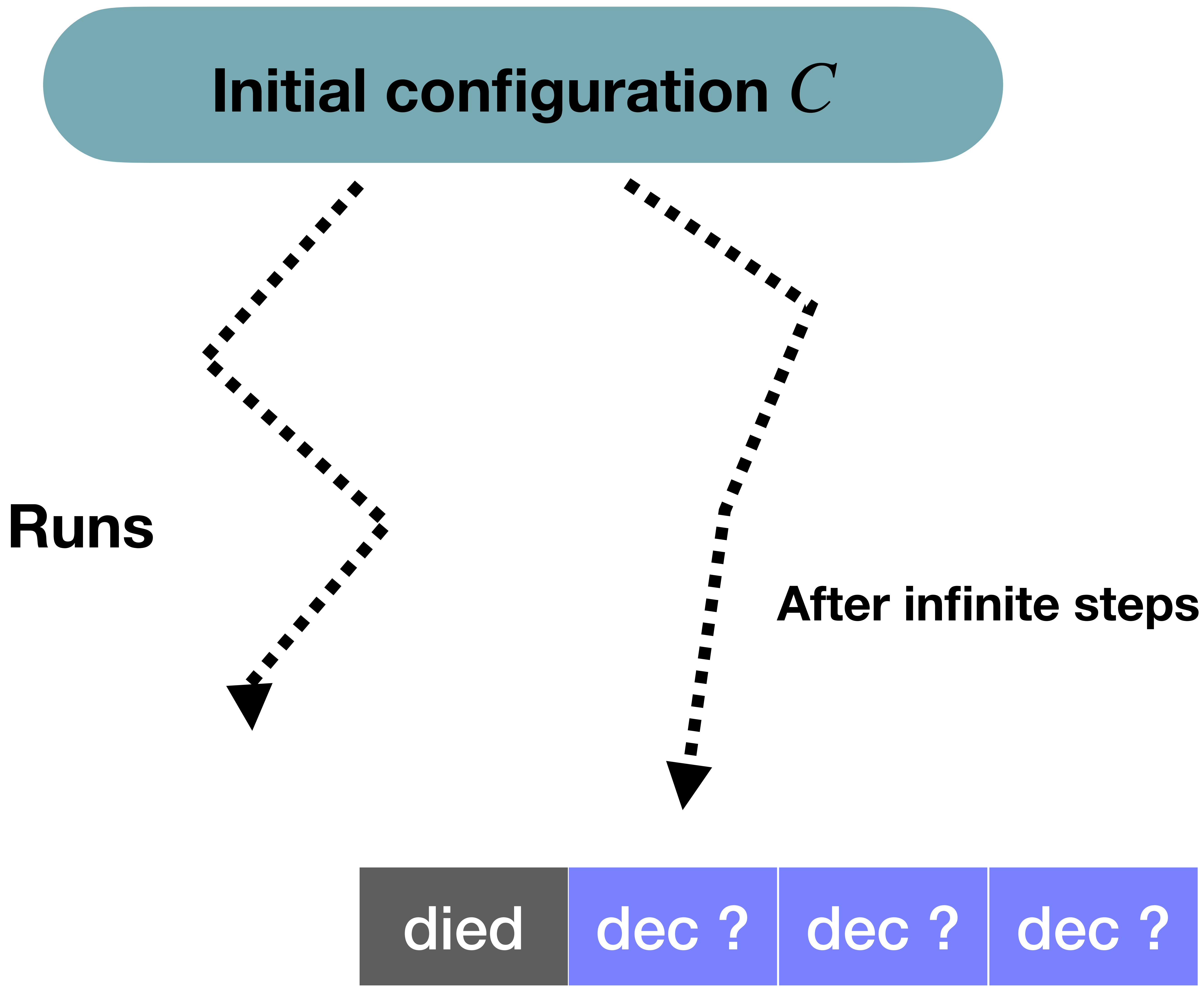

Initial configuration C
Runs

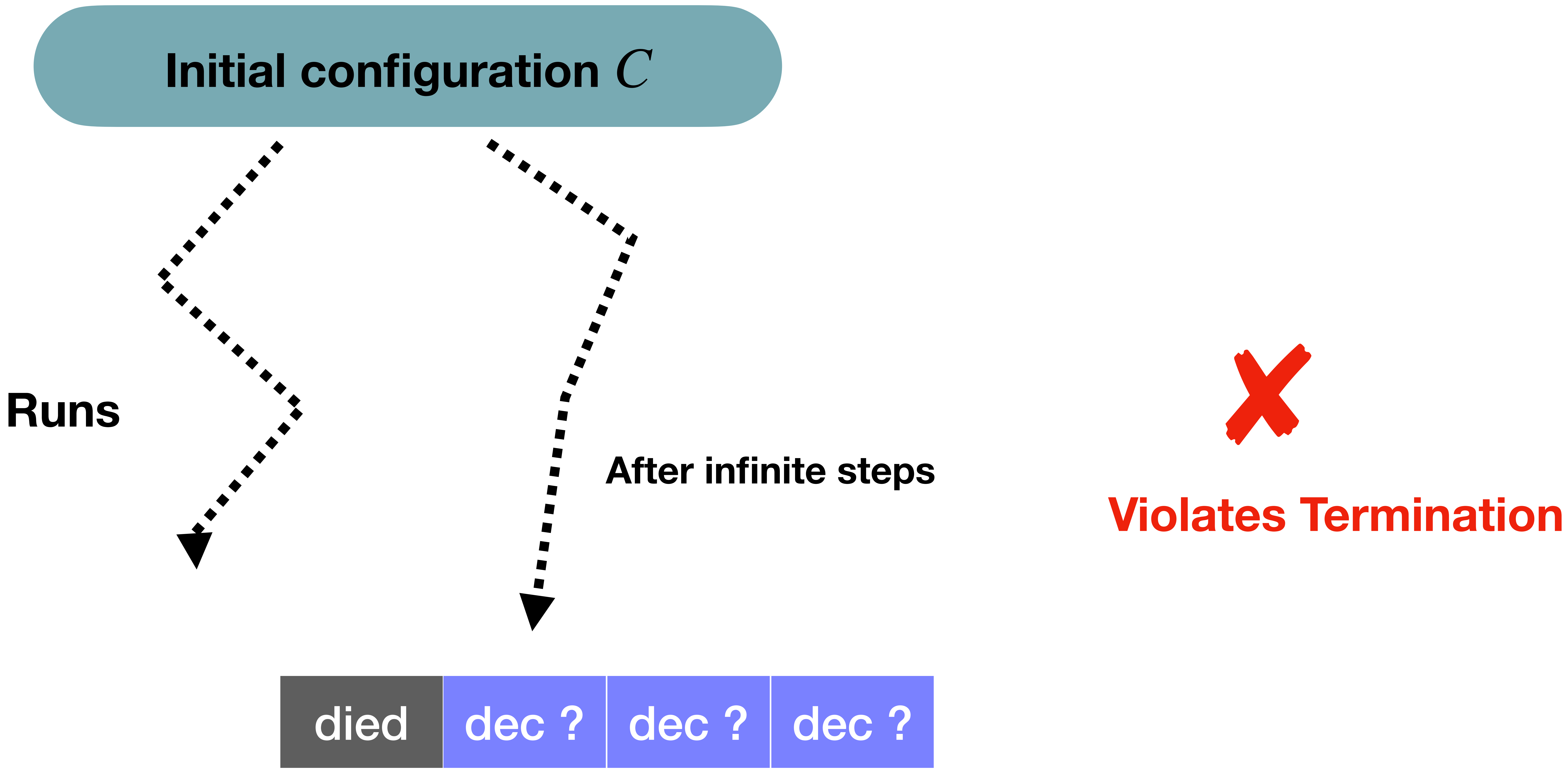
Initial configuration C

Runs

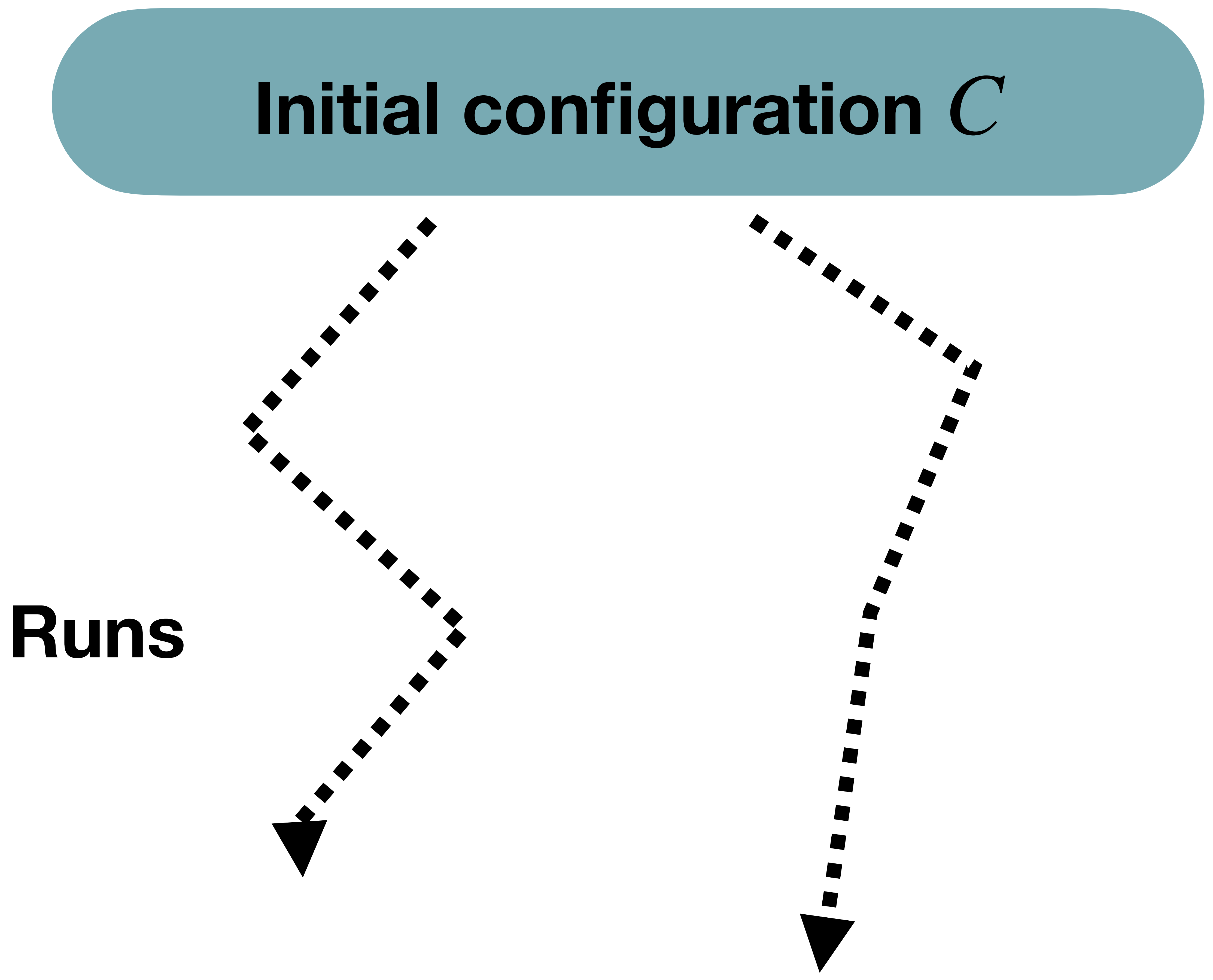


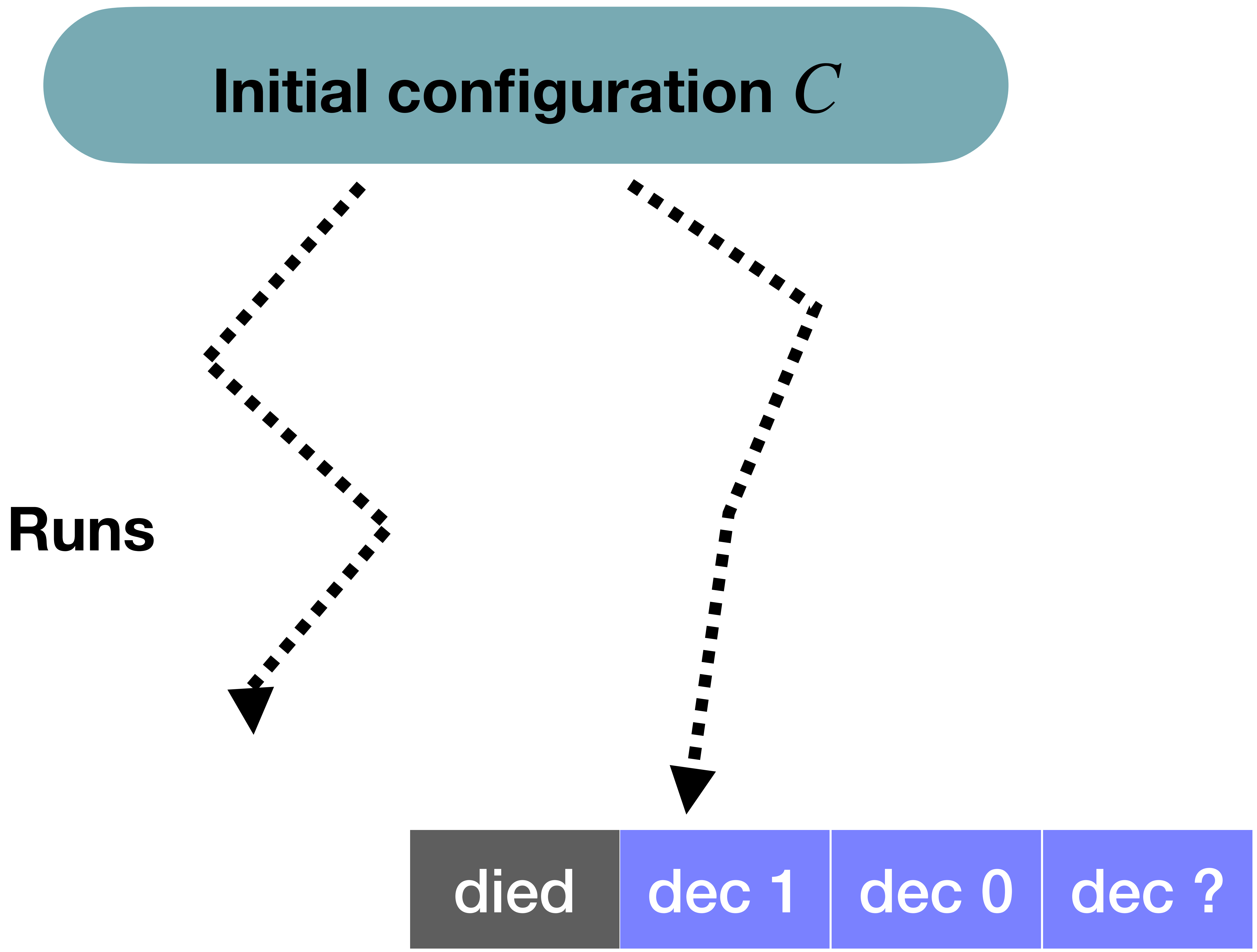
After infinite steps

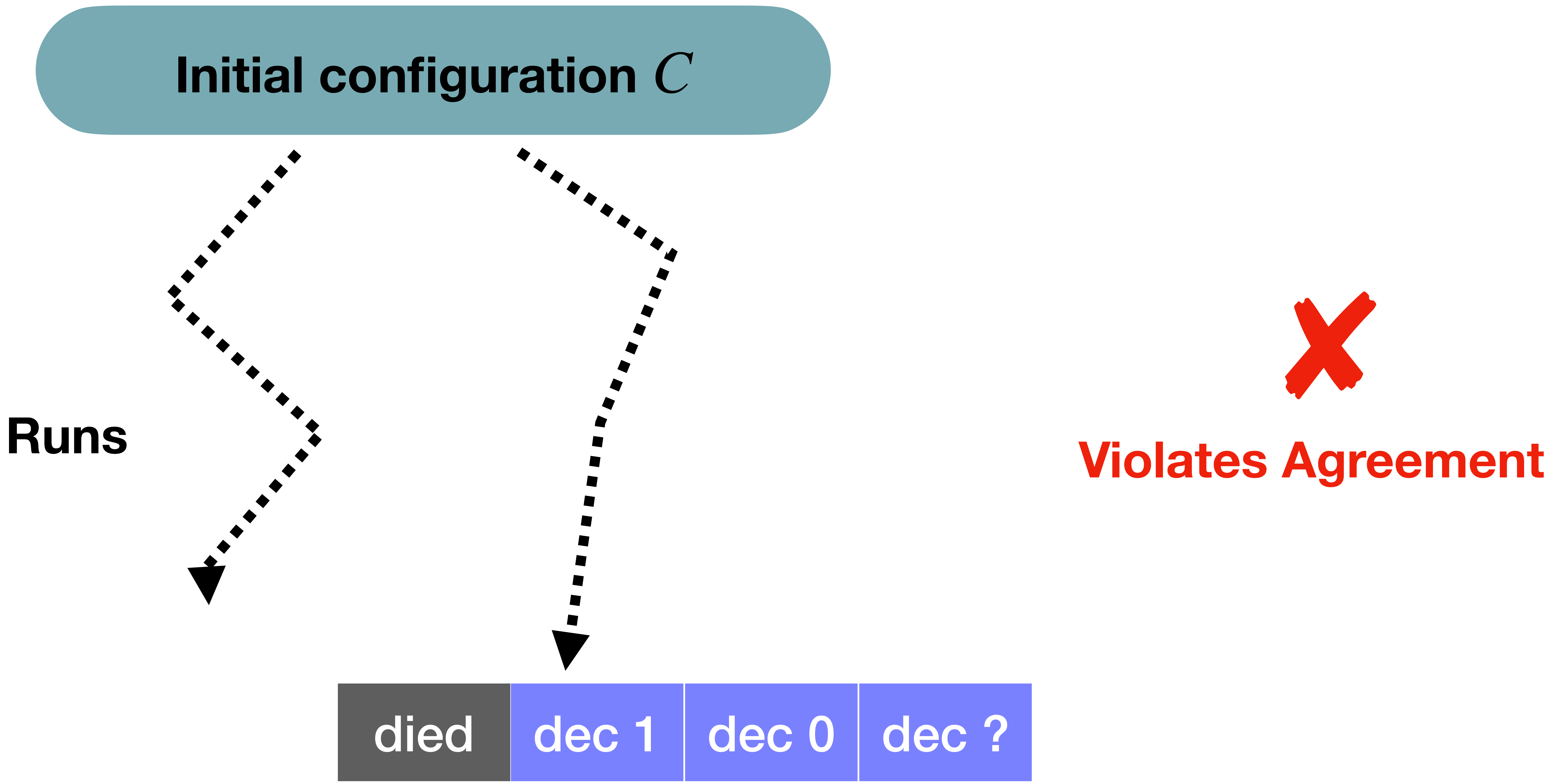

Initial configuration C
Runs
After infinite steps
died
dec ?
dec ?
dec ?

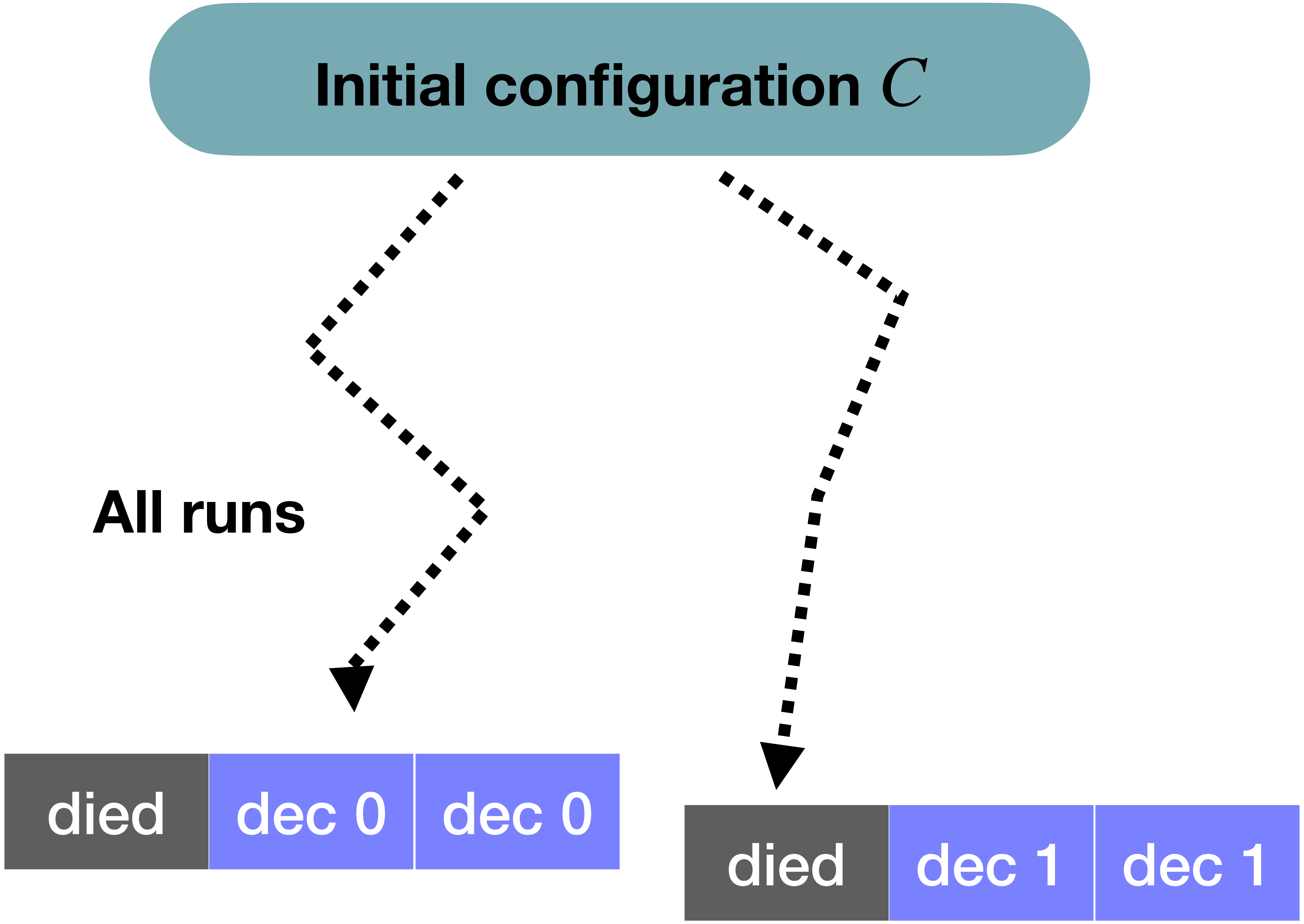


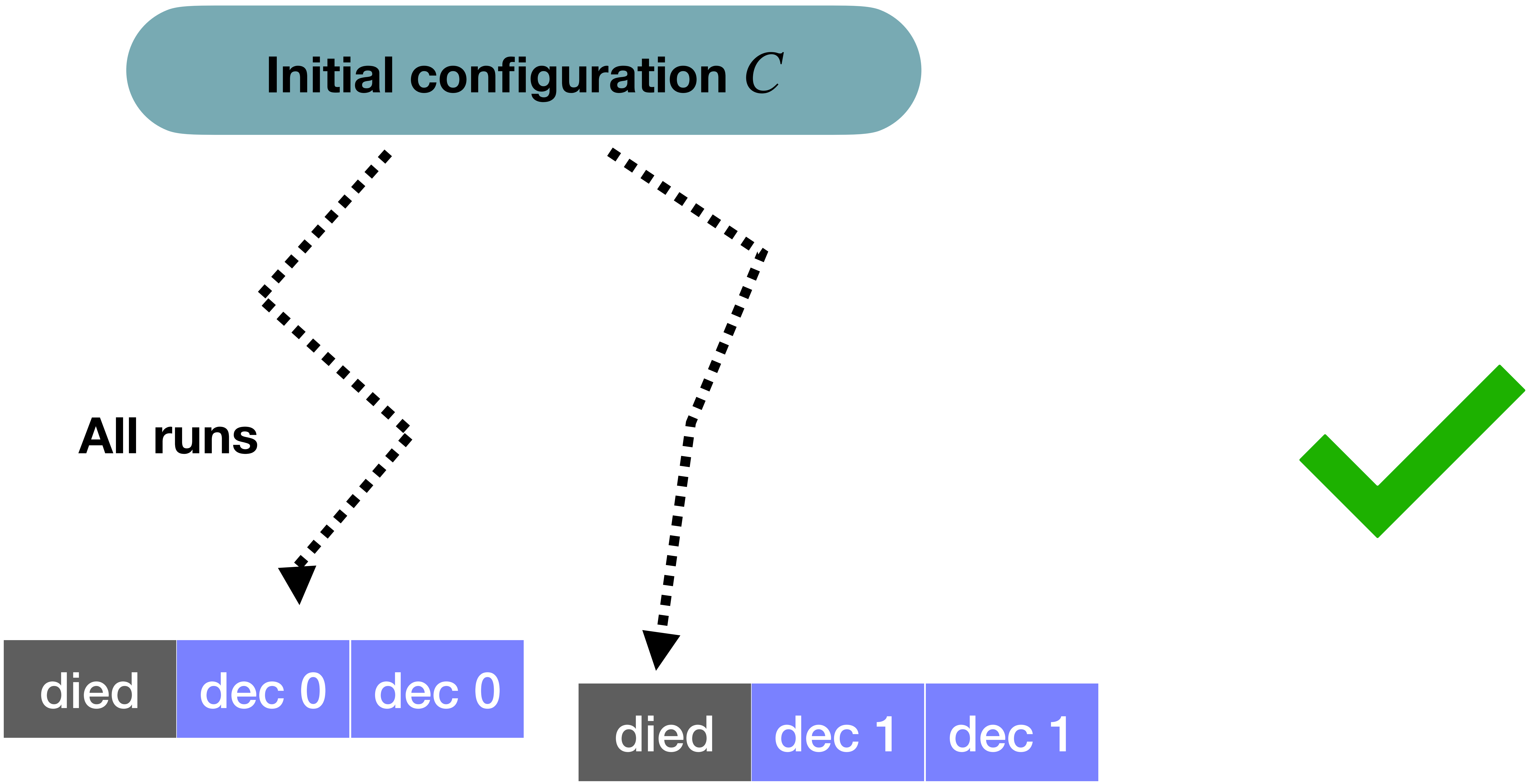
Initial configuration C
Runs
After infinite steps
died
dec ?
dec ?
dec ?
Violates Termination

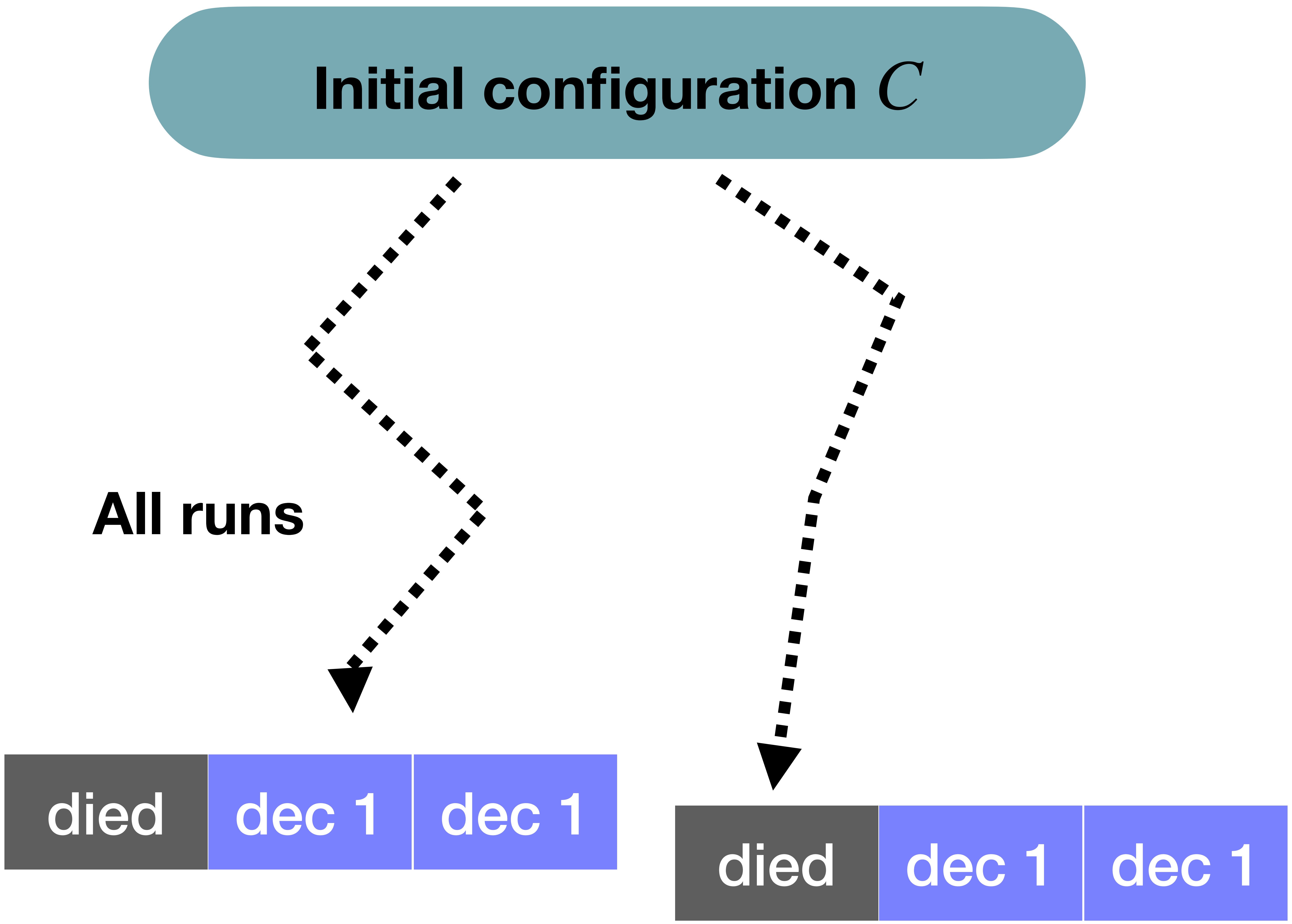

Initial configuration C
Runs

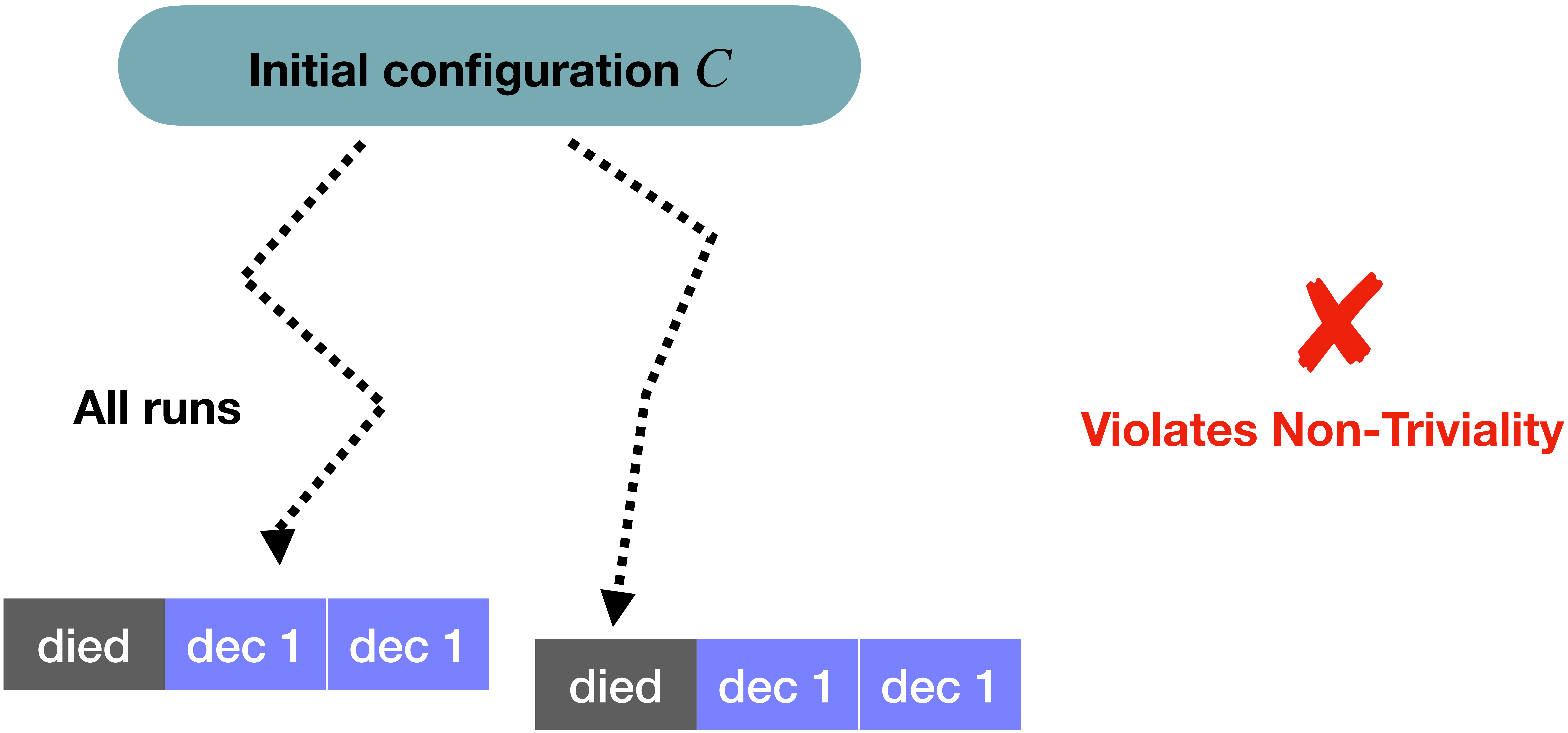

Initial configuration C
Runs
died
dec 1
dec 0
dec ?


Initial configuration C
Runs
died
dec 1
dec 0
dec ?
Violates Agreement


Initial configuration C
All runs
died
dec 0
dec 0
died
dec 1
dec 1


Initial configuration C
All runs
died
dec 0
dec 0
died
dec 1
dec 1


Initial configuration C
All runs
died
dec 1
dec 1
died
dec 1
dec 1


Initial configuration C
All runs
died
dec 1
dec 1
died
dec 1
dec 1
Violates Non-Triviality

The Impossibility Result

The Impossibility Result

- **Theorem.** No consensus system is **totally correct in spite of one fault** in asynchronous system:
  - Messages maybe delayed arbitrarily and delivered out of order.
  - Processes do not have access to synchronized clocks.
  - Processes cannot detect the death of others.

Terminology

Terminology

- Let V_C be the set of decision values of configurations reachable from C.

Terminology

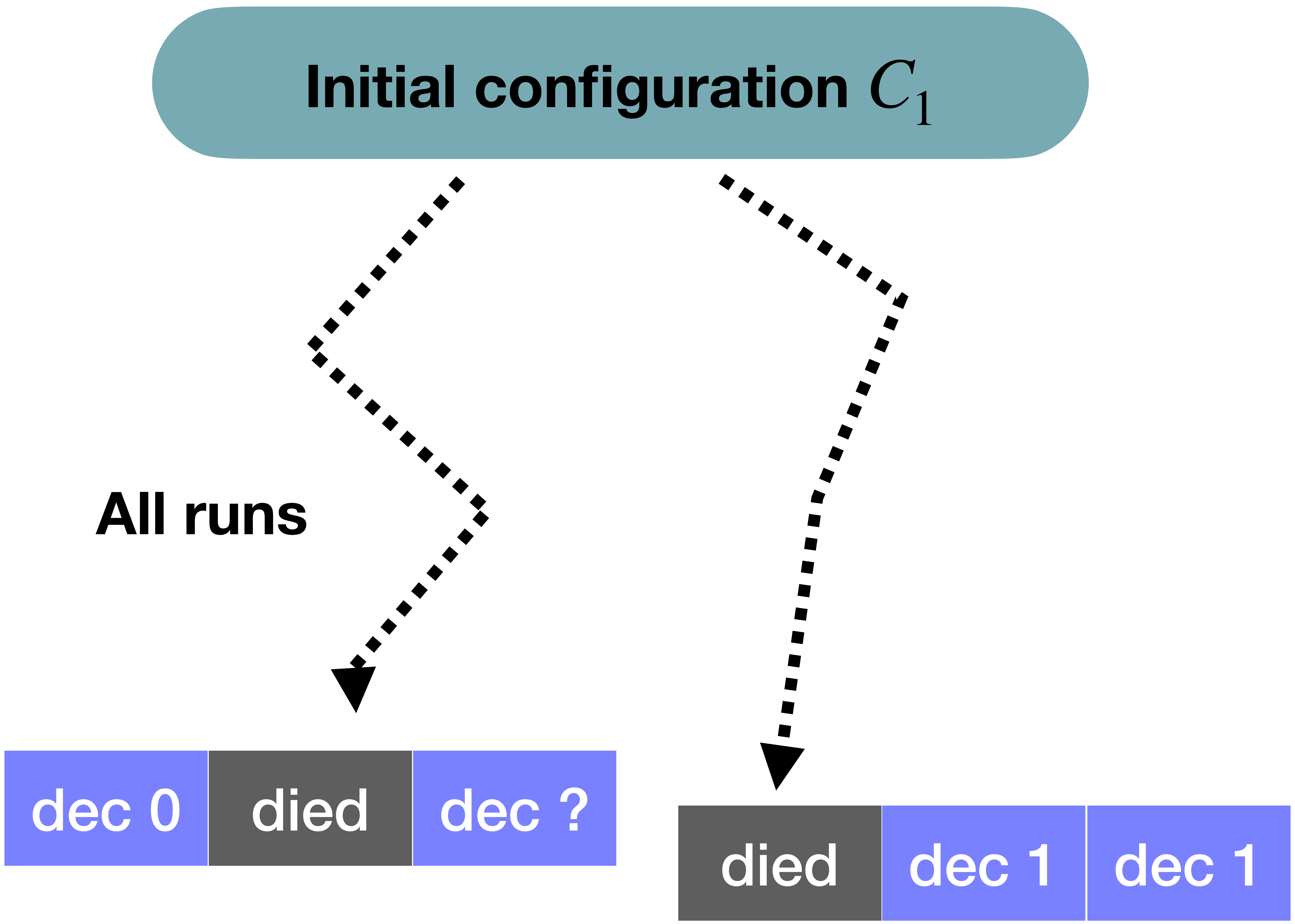
- Let V_C be the set of decision values of configurations reachable from C.
  - Say that C is **bivalent** if $|V_C| = 2$.

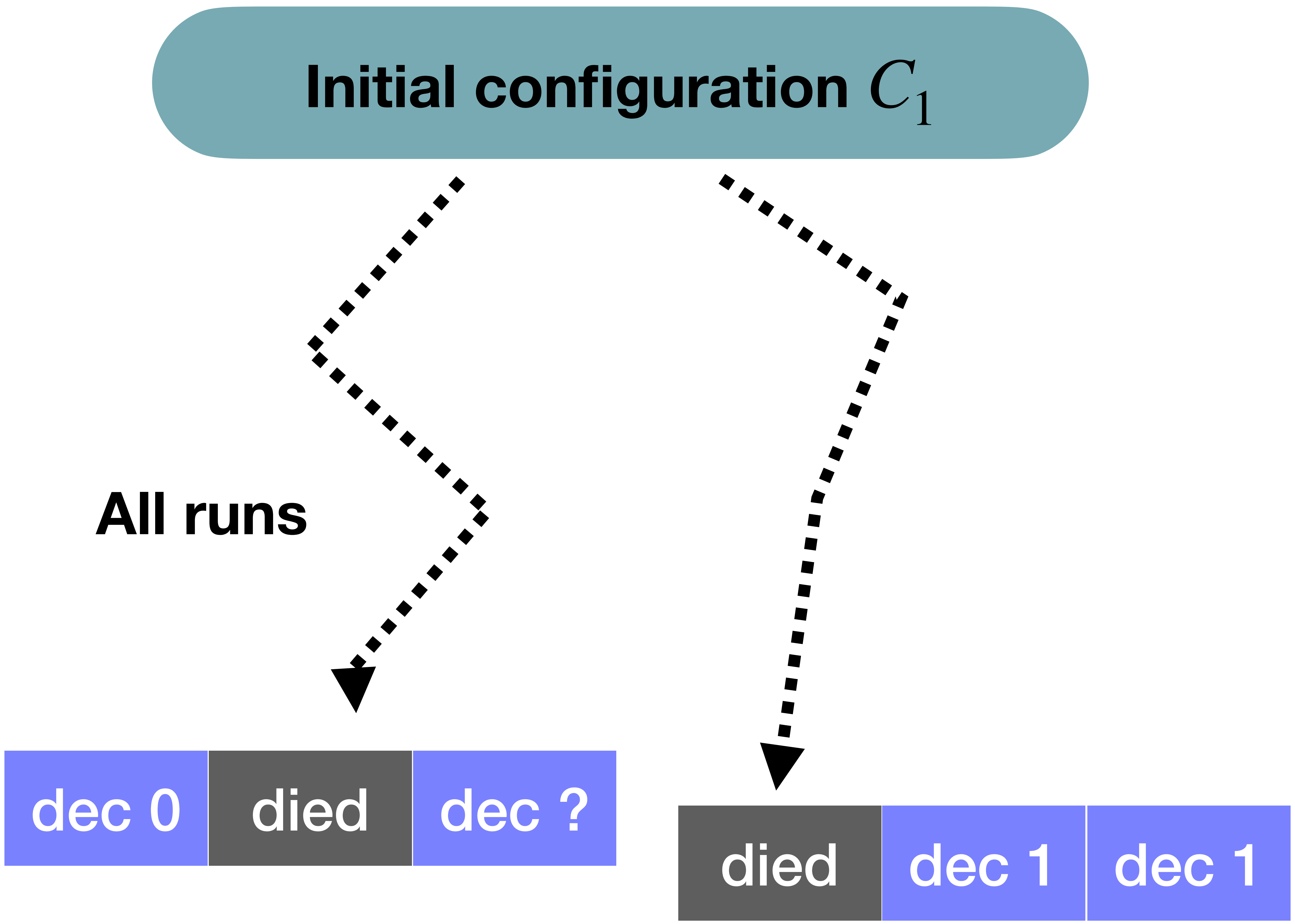
Terminology

- Let V_C be the set of decision values of configurations reachable from C.
  - Say that C is **bivalent** if $|V_C| = 2$.
  - C is **univalent** if $|V_C| = 1$.

Terminology

- Let V_C be the set of decision values of configurations reachable from C.
  - Say that C is **bivalent** if $|V_C| = 2$.
  - C is **univalent** if $|V_C| = 1$.
  - In particular, C is i**-valent** if $V_C = \{i\}$.


Initial configuration C_1
All runs
dec 0
died
dec ?
died
dec 1
dec 1



C_1 **is bivalent assume Agreement.**

C_1 is bivalent assume Agreement.

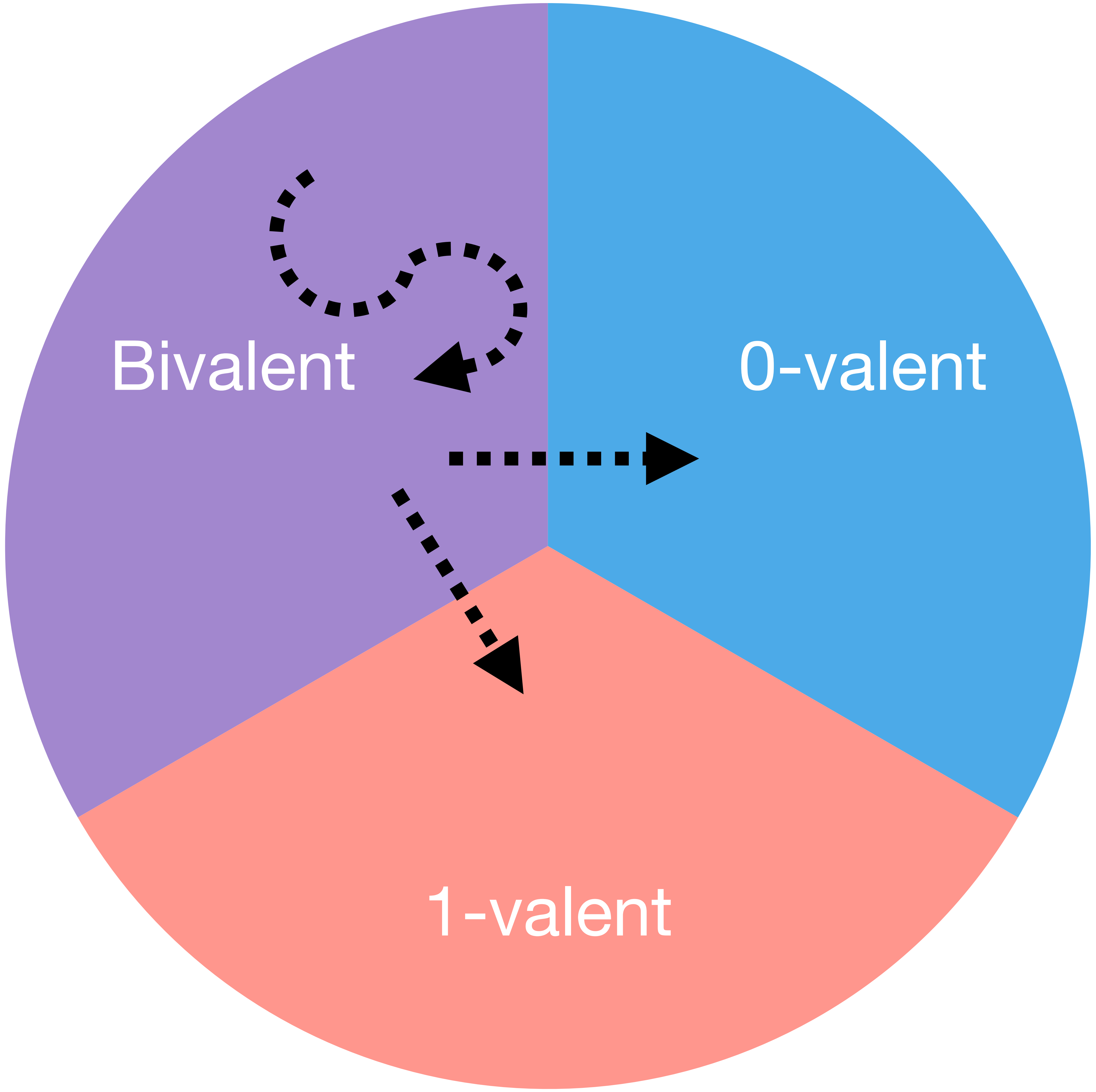
C_1 is bivalent assume Agreement.

C_2 is not 0-valent.

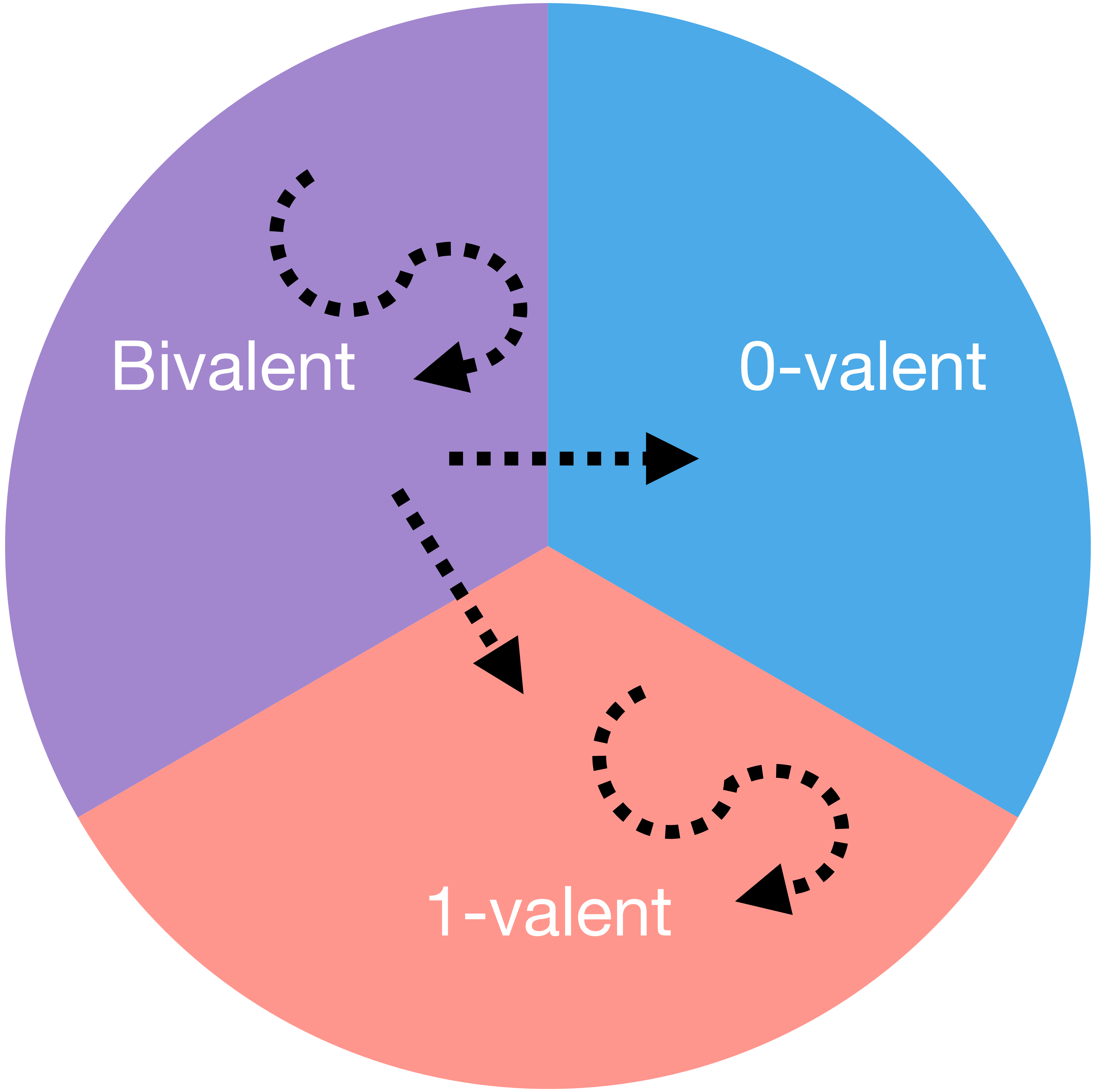
Terminology

Terminology

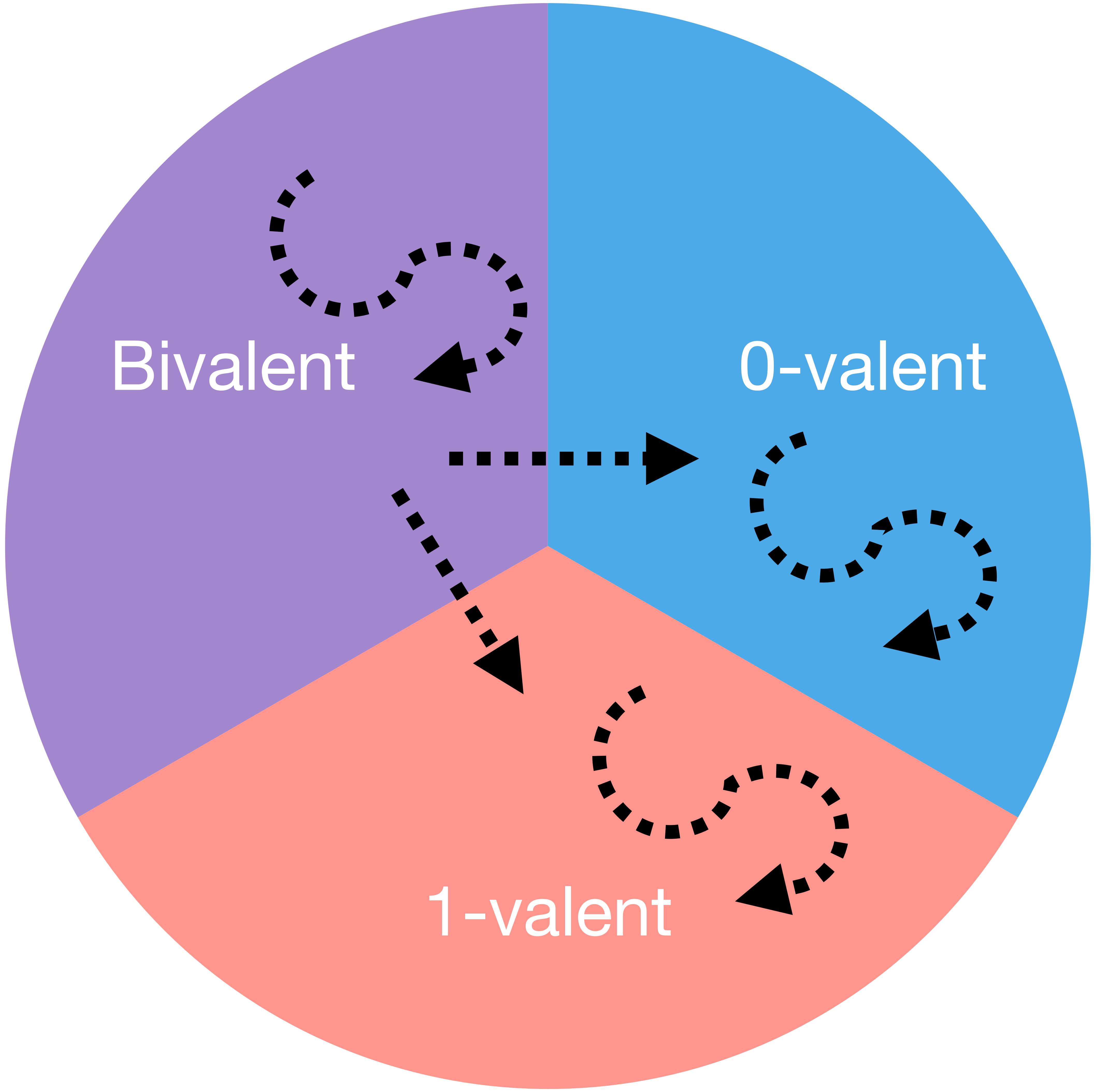
Terminology



Terminology



Terminology



Proof Sketch

Proof Sketch

- Proof by contradiction:

Proof Sketch

- Proof by contradiction:
  - Assume P is a totally correct in spite of one fault. Then we can prove:

Proof Sketch

- Proof by contradiction:
  - Assume P is a totally correct in spite of one fault. Then we can prove:
    - **Claim 1**. There exists a bivalent initial configuration C in P.

Proof Sketch

- Proof by contradiction:
  - Assume P is a totally correct in spite of one fault. Then we can prove:
    - **Claim 1**. There exists a bivalent initial configuration C in P.
    - **Claim 2**. Given a bivalent configuration C and a step e that is applicable to C, there is a schedule σ that applies e in the last step and keeps the configuration $\sigma(C)$ bivalent.

Proof Sketch

- Proof by contradiction:
  - Assume P is a totally correct in spite of one fault. Then we can prove:
    - **Claim 1**. There exists a bivalent initial configuration C in P.
    - **Claim 2**. Given a bivalent configuration C and a step e that is applicable to C, there is a schedule σ that applies e in the last step and keeps the configuration $\sigma(C)$ bivalent.
  - Claim 1 and Claim 2 implies there is an admissible run in P that stays in bivalent configuration, which contradicts with the total correctness.

Claim 1

There exists a bivalent initial configuration in P.

Claim 1

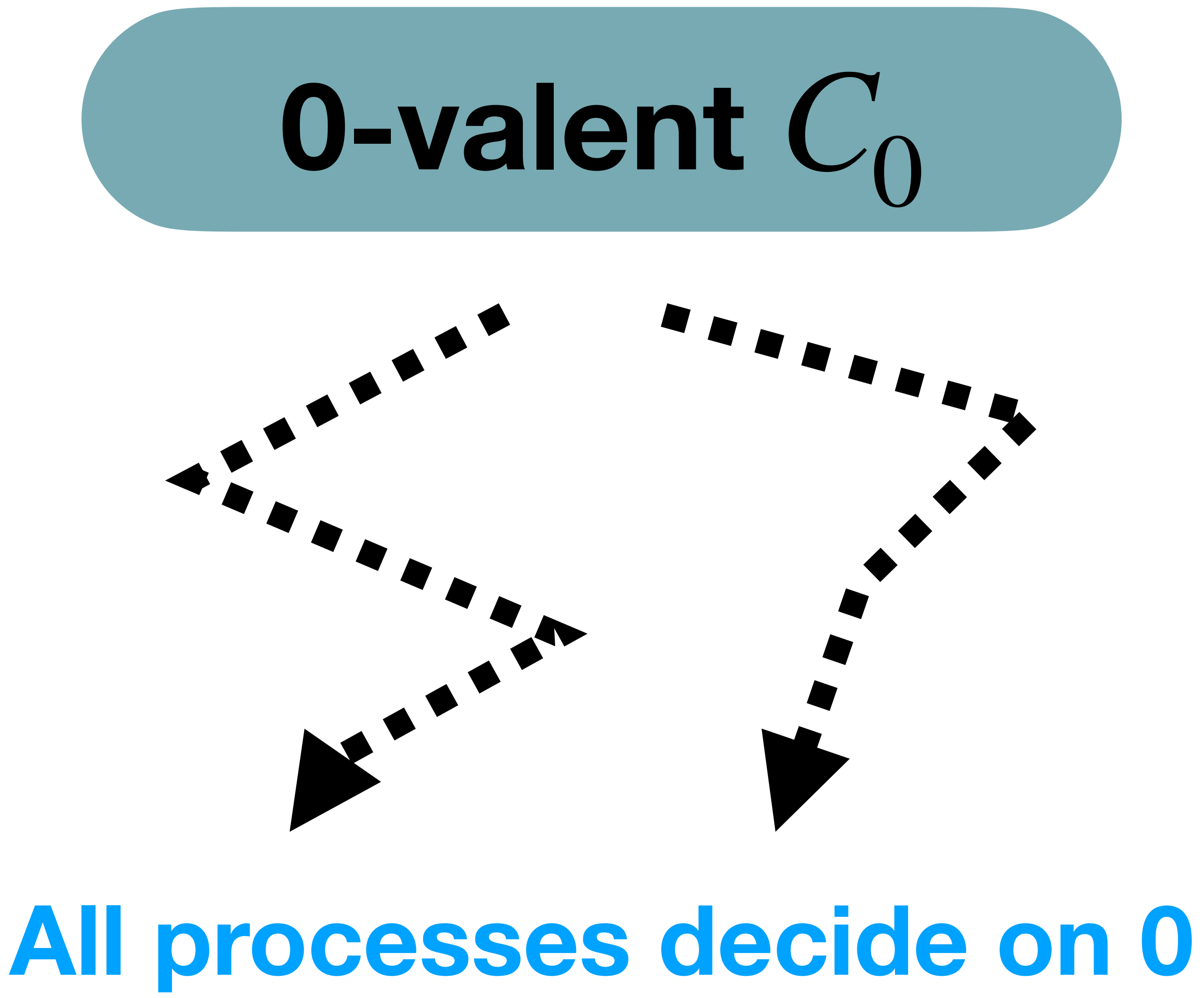
There exists a bivalent initial configuration in P.

- Assume not. Then by **Non-triviality**, the set of initial configurations in P contains:

Claim 1

There exists a bivalent initial configuration in P.

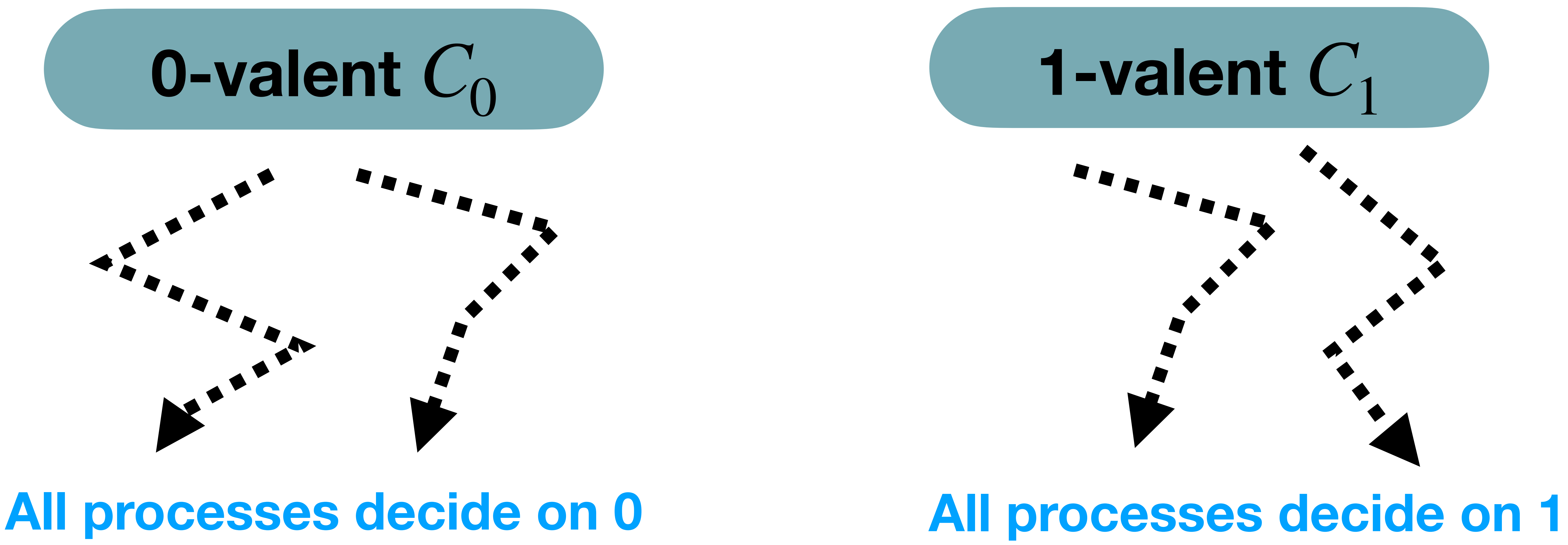
- Assume not. Then by **Non-triviality**, the set of initial configurations in P contains:



Claim 1

There exists a bivalent initial configuration in P.

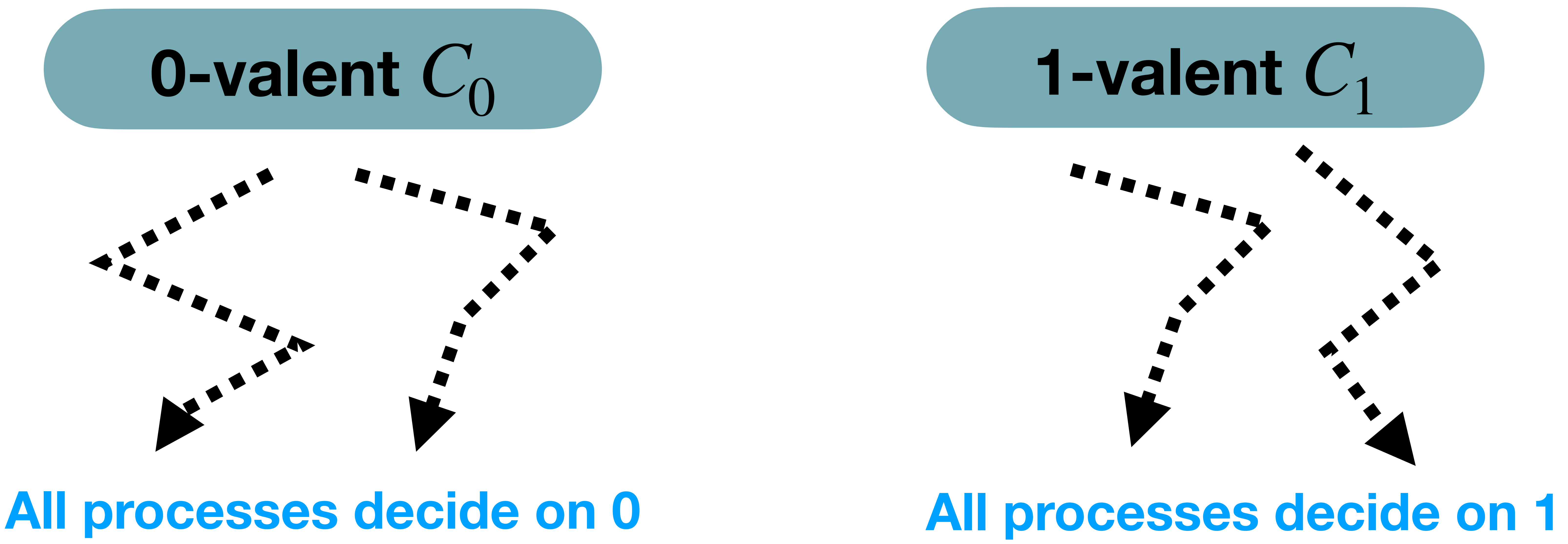
- Assume not. Then by **Non-triviality**, the set of initial configurations in P contains:



Claim 1

There exists a bivalent initial configuration in P.

- Assume not. Then by **Non-triviality**, the set of initial configurations in P contains:



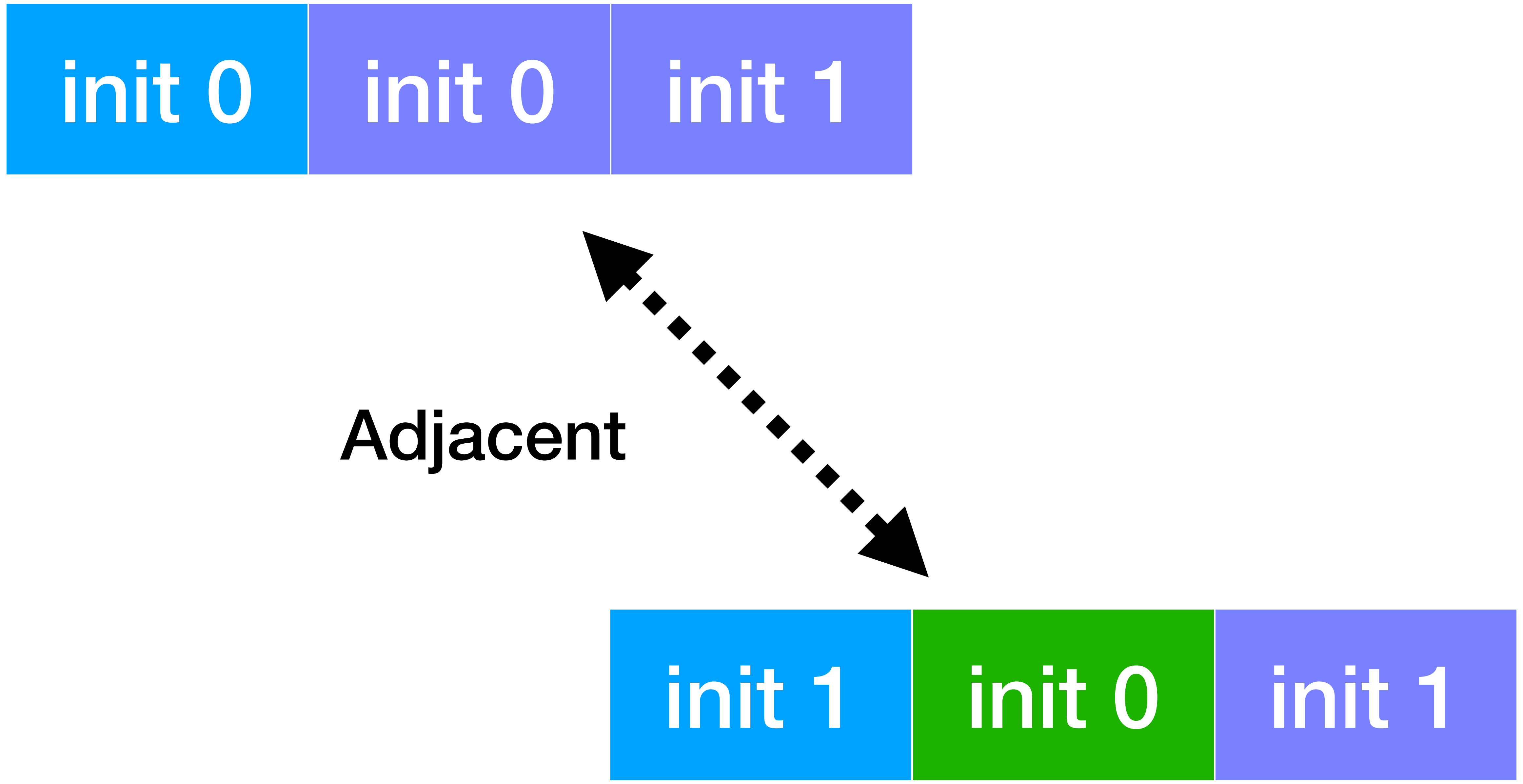
- **Definition**: Two initial configurations are **adjacent** if they only differ in one process.

Claim 1

There exists a bivalent initial configuration in P.

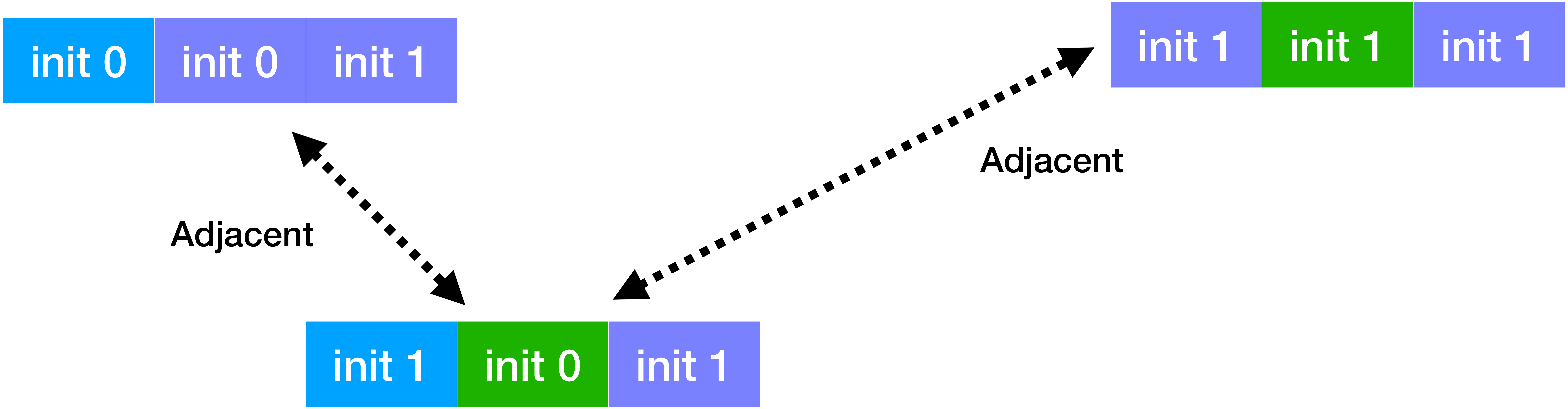
Claim 1

There exists a bivalent initial configuration in P.



Claim 1

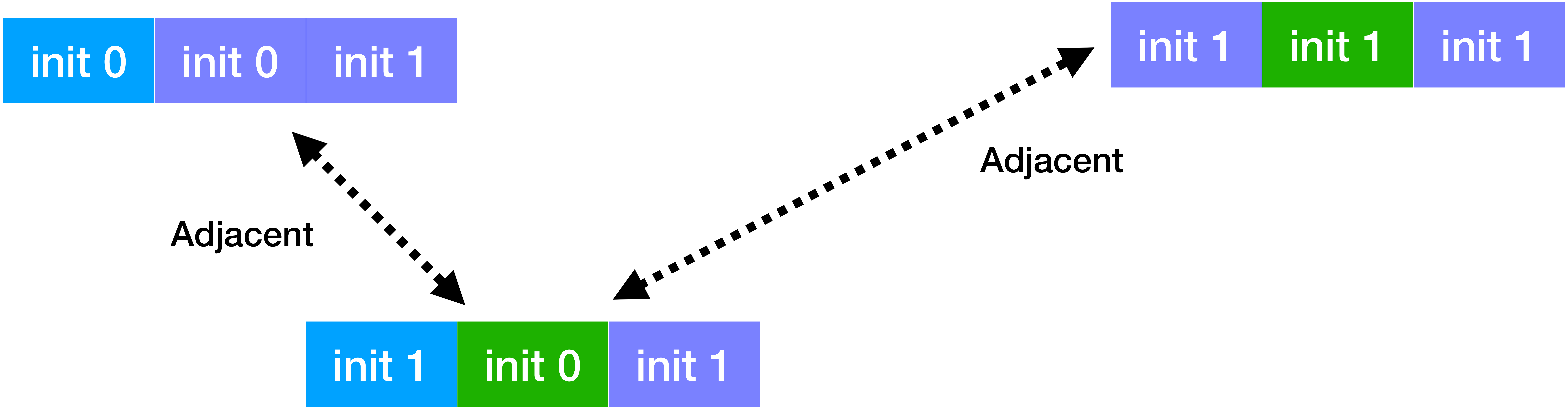
There exists a bivalent initial configuration in P.



Claim 1

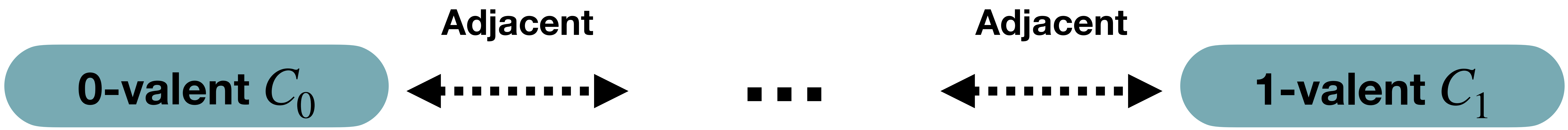
There exists a bivalent initial configuration in P.

Any two configurations can be connected by a chain of adjacent configurations.



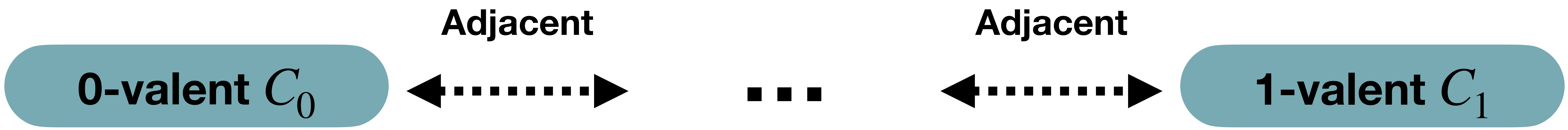
Claim 1

There exists a bivalent initial configuration in P.



Claim 1

There exists a bivalent initial configuration in P.



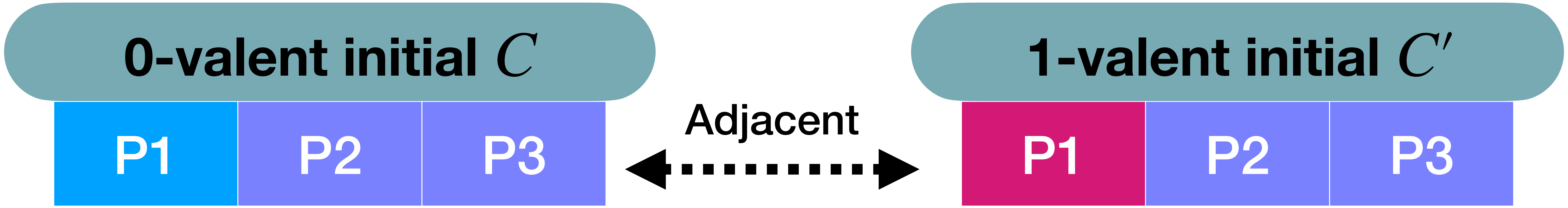
There exists adjacent C, C' in the chain connecting C_0, C_1 such that C is 0-valent, C' is 1-valent.

Claim 1

There exists a bivalent initial configuration in P.

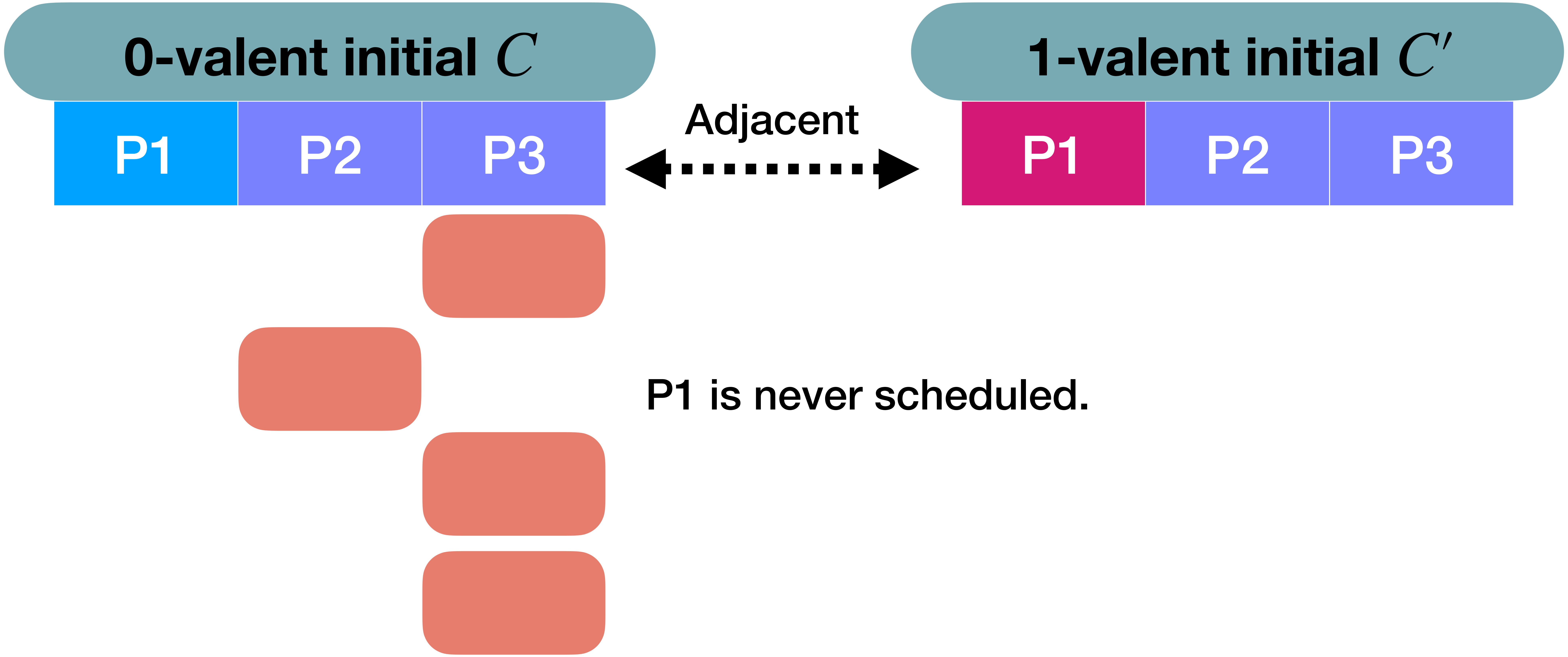
Claim 1

There exists a bivalent initial configuration in P.



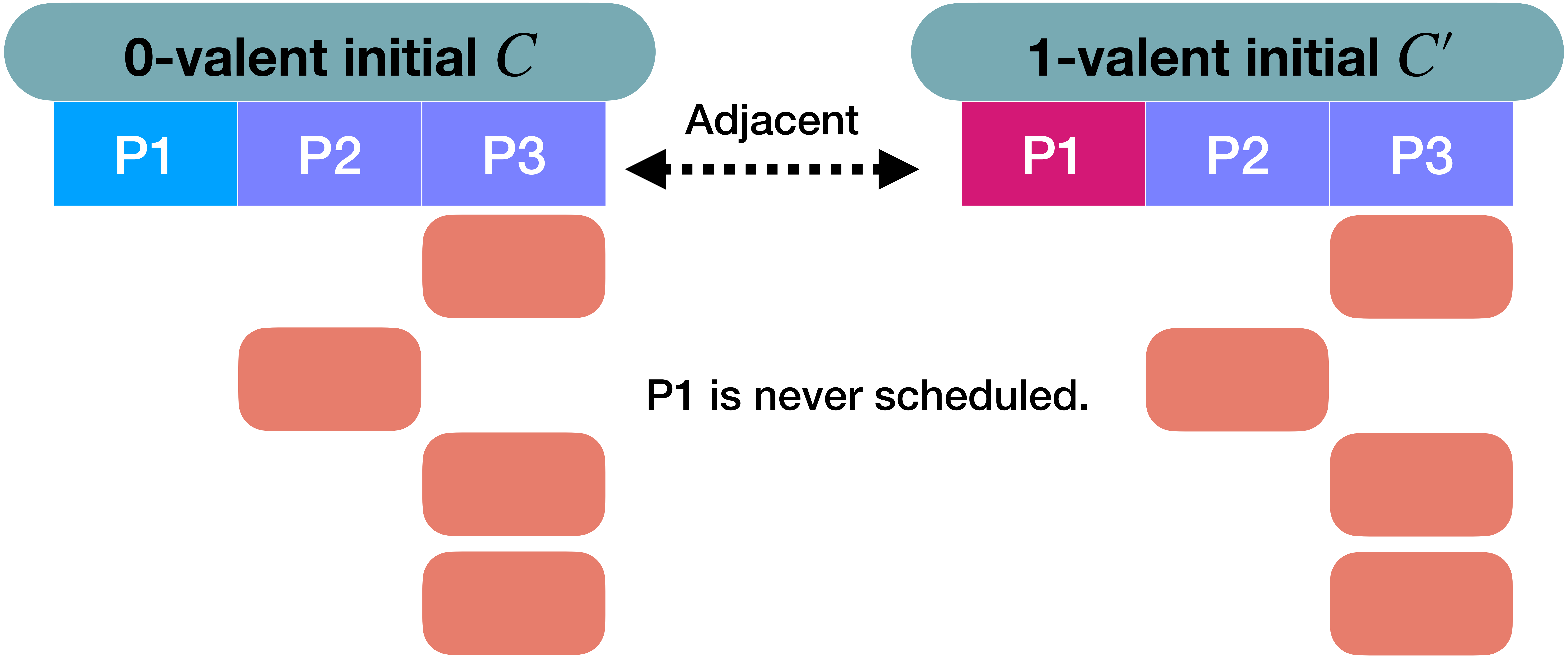
Claim 1

There exists a bivalent initial configuration in P.



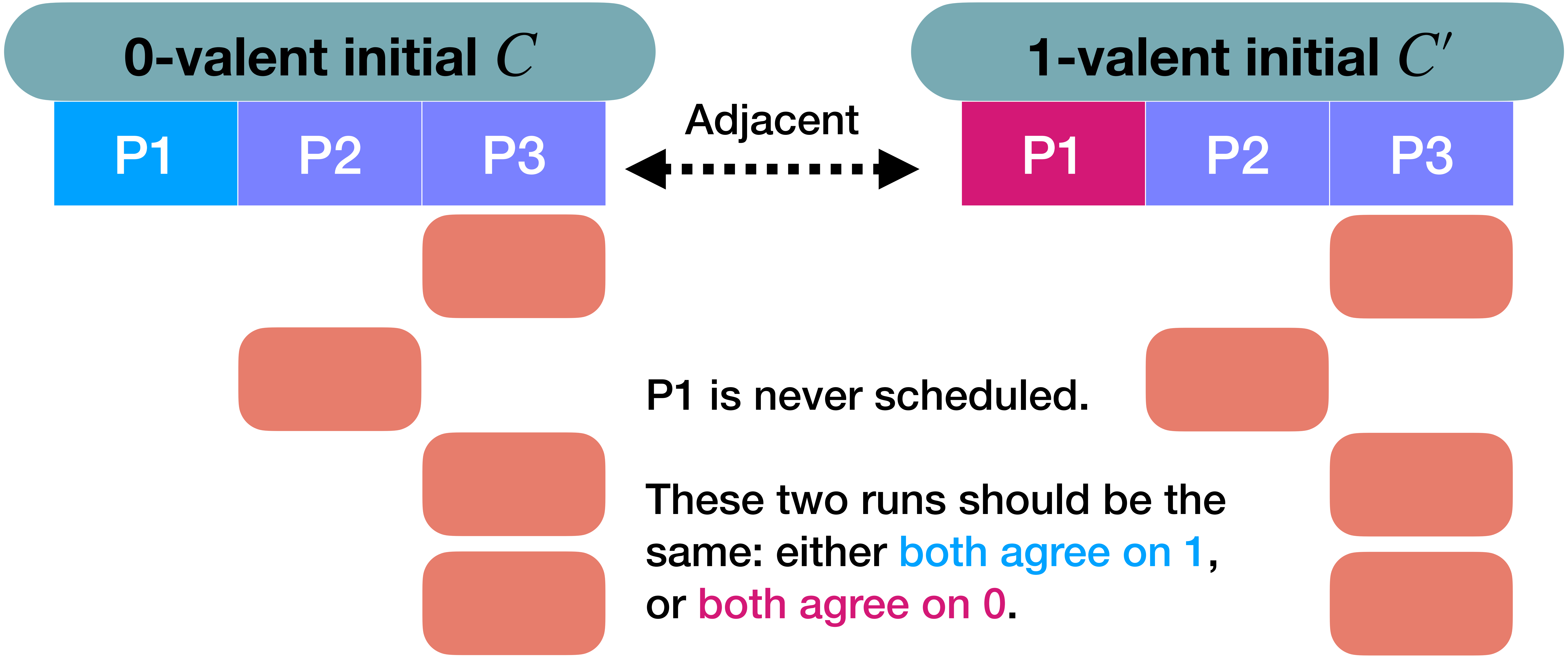
Claim 1

There exists a bivalent initial configuration in P.



Claim 1

There exists a bivalent initial configuration in P.



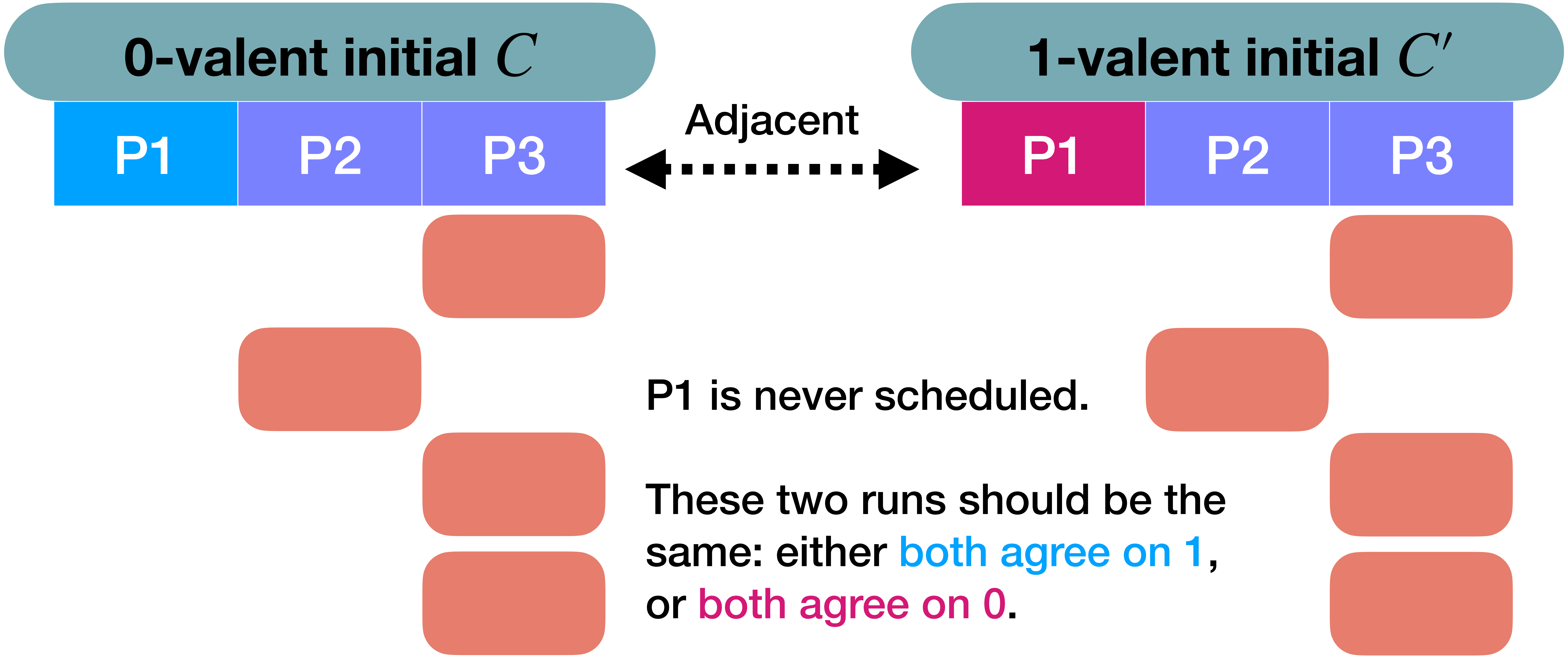
P1 is never scheduled.

These two runs should be the same: either both agree on 1, or both agree on 0.

Claim 1

There exists a bivalent initial configuration in P.

Both agree on 1 contradiction with



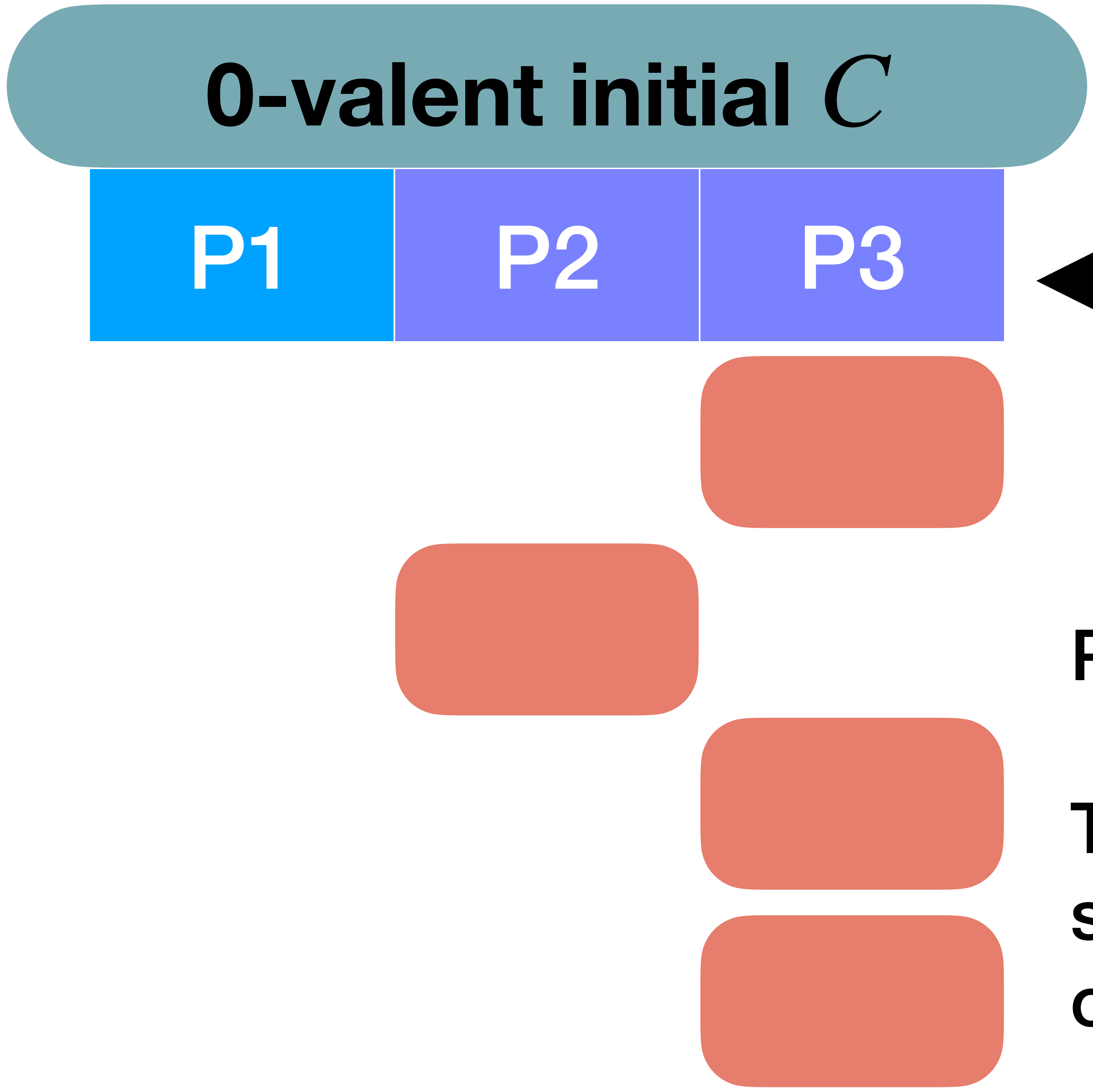
P1 is never scheduled.

These two runs should be the same: either both agree on 1, or both agree on 0.

Claim 1

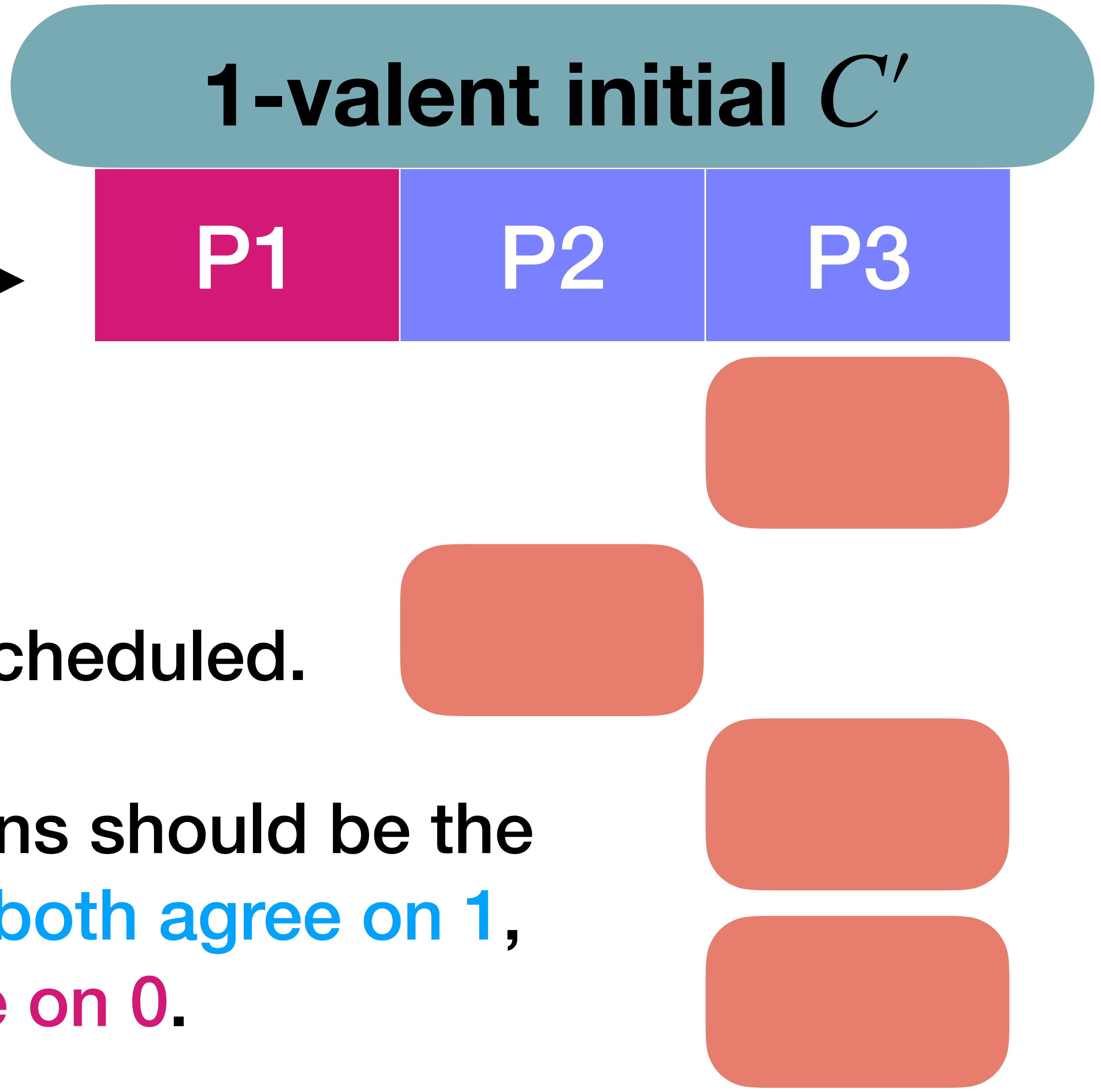
There exists a bivalent initial configuration in P.

Both agree on 1 contradiction with



Adjacent

Both agree on 0 contradiction with



P1 is never scheduled.

These two runs should be the same: either both agree on 1, or both agree on 0.

Claim 2

There exists a schedule that preserves bivalence.

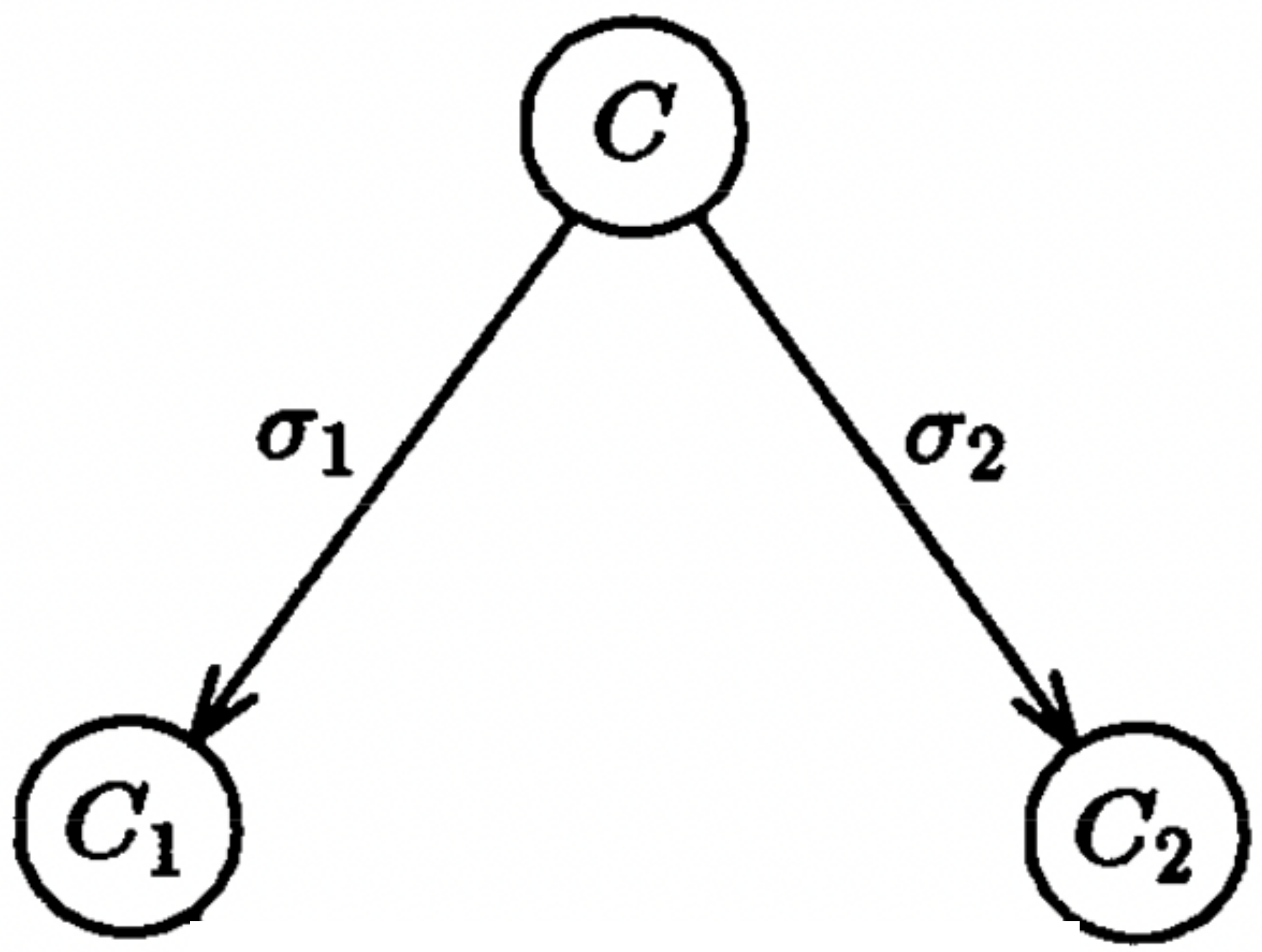
Claim 2

There exists a schedule that preserves bivalence.

Lemma 1.

Claim 2

There exists a schedule that preserves bivalence.

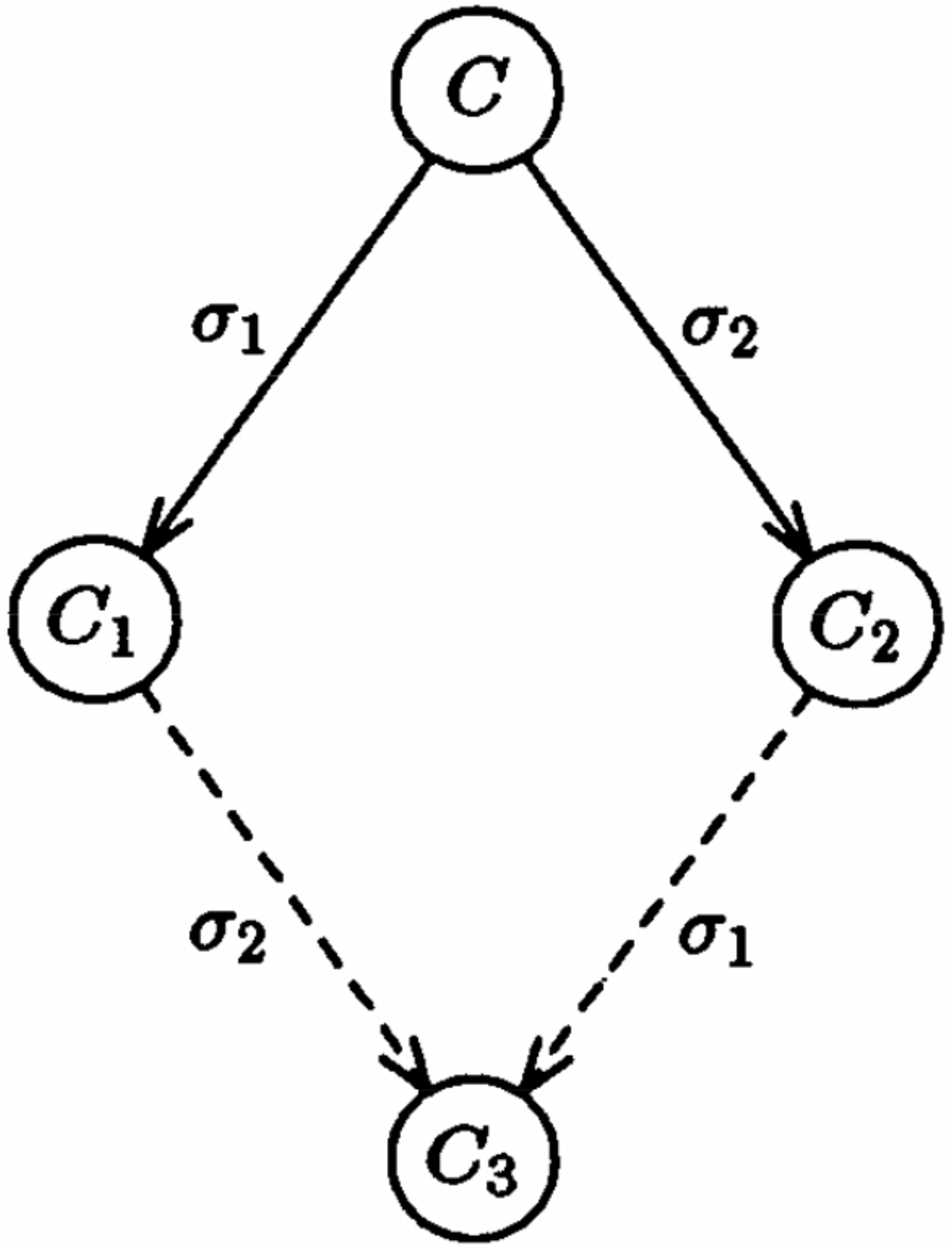


Lemma 1.

If schedule σ_1 and σ_2 are both applicable to the configuration C and the set of processes stepped in σ_1 and σ_2 are disjoint,

Claim 2

There exists a schedule that preserves bivalence.



Lemma 1.

If schedule σ_1 and σ_2 are both applicable to the configuration C and the set of processes stepped in σ_1 and σ_2 are disjoint,

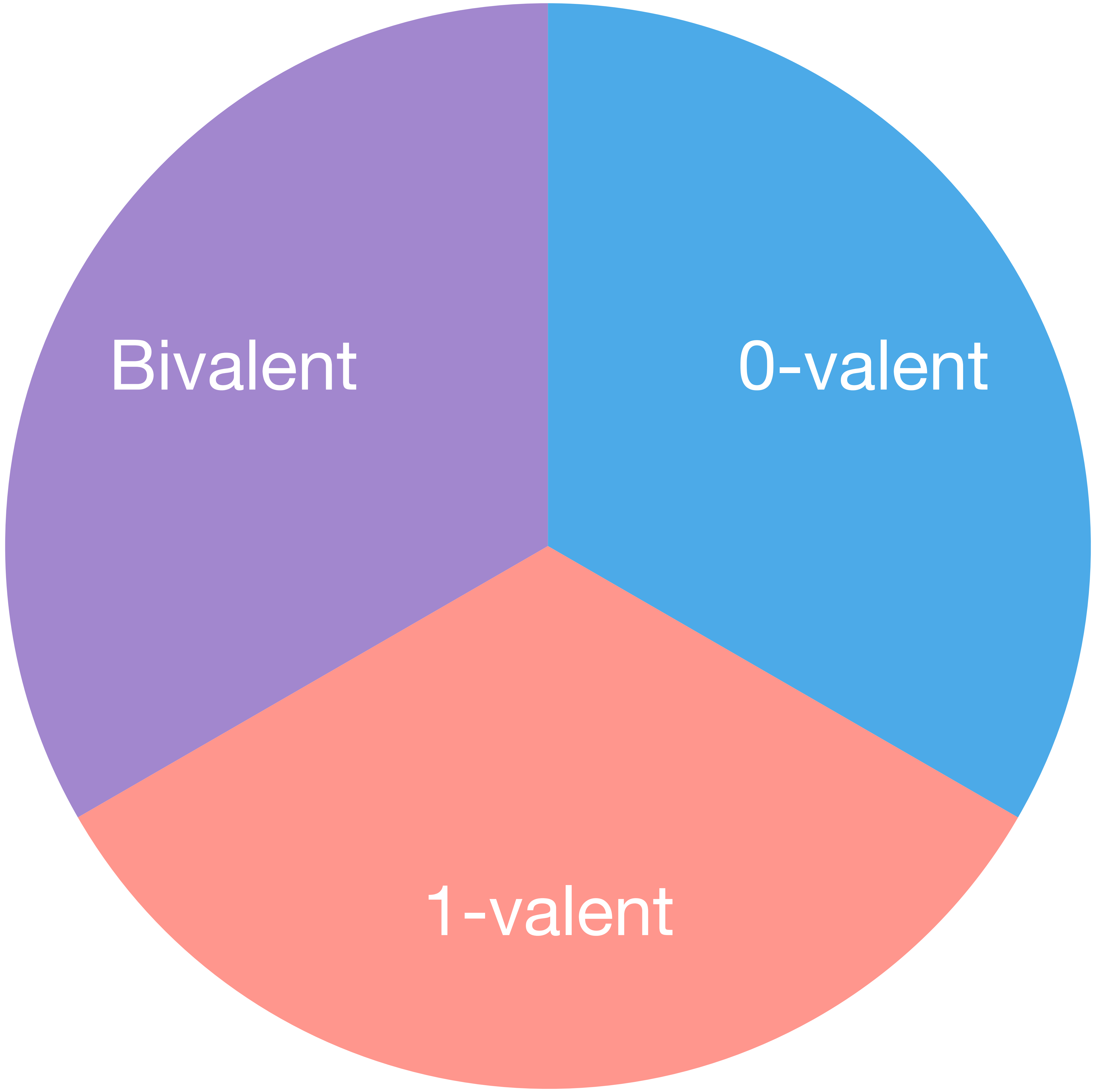
then $\sigma_1; \sigma_2$ and $\sigma_2; \sigma_1$ are also applicable to C and they are equivalent.

Claim 2

There exists a schedule that preserves bivalence.

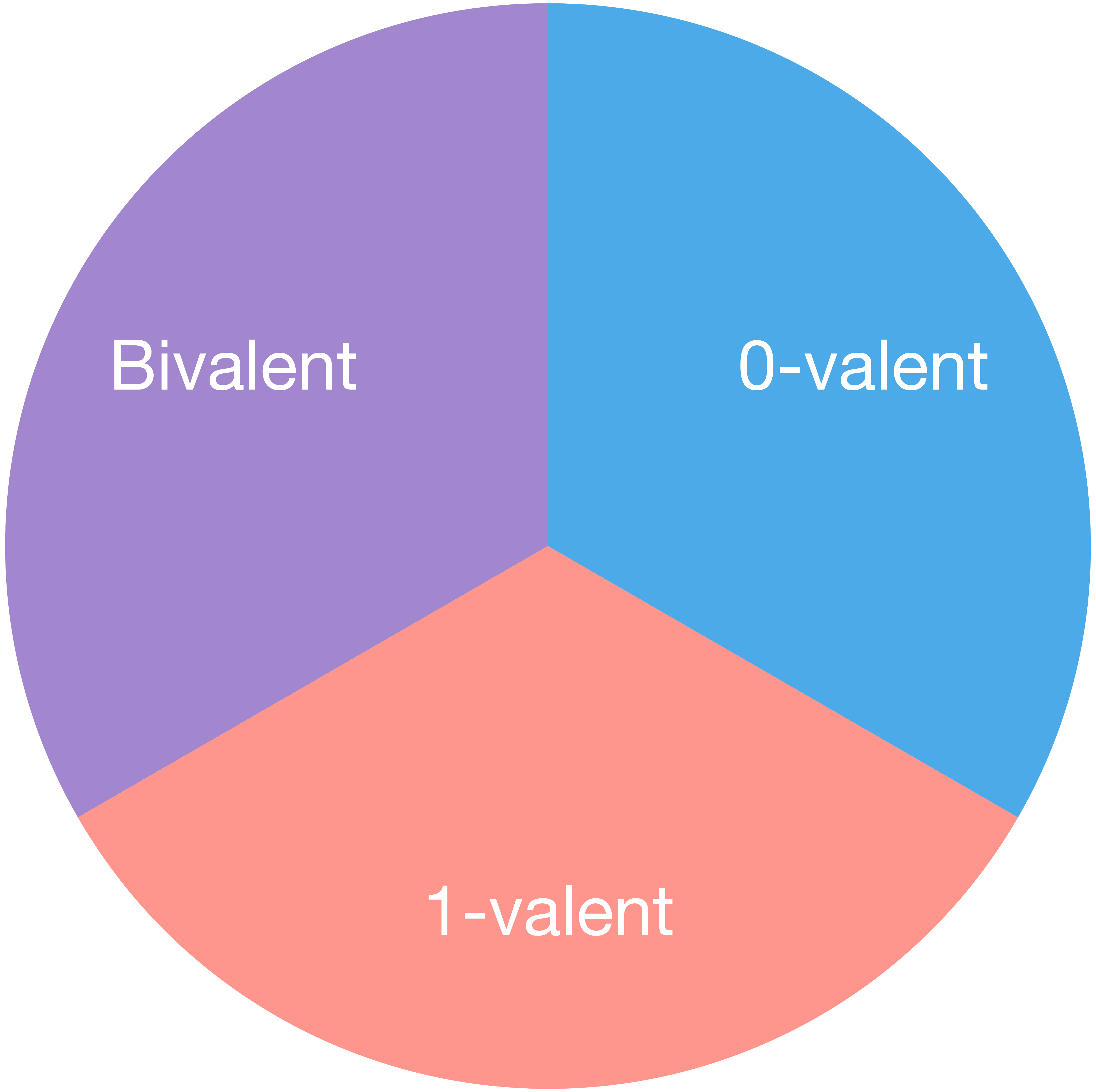
Claim 2

There exists a schedule that preserves bivalence.



Claim 2

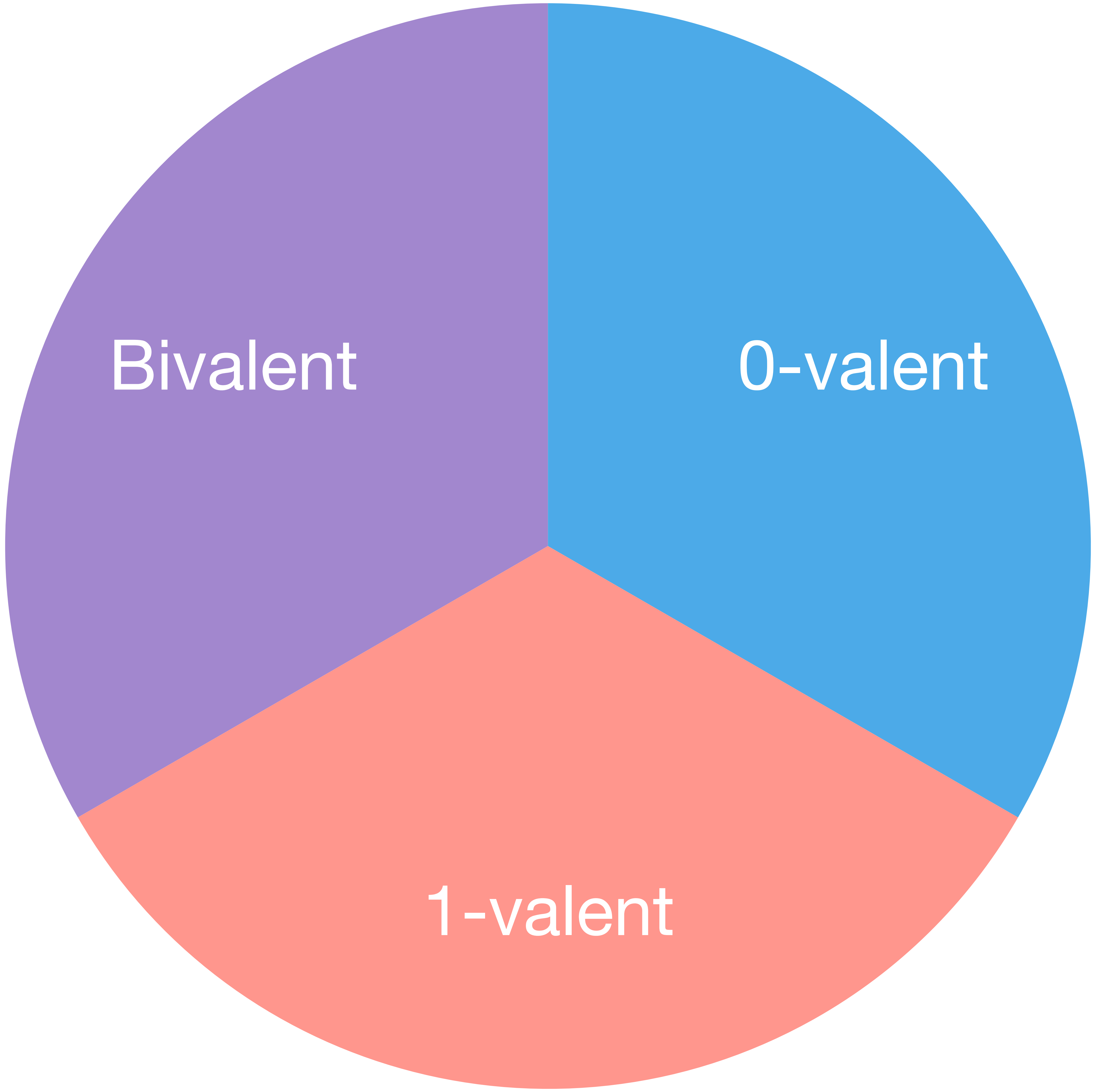
There exists a schedule that preserves bivalence.



Lemma 2.

Claim 2

There exists a schedule that preserves bivalence.



Lemma 2.

If Claim 2 does not hold,

Claim 2

There exists a schedule that preserves bivalence.

Lemma 2.

If Claim 2 does not hold,

then there exists a bivalent C and two steps e, e' operating on the same process p such that

Claim 2

There exists a schedule that preserves bivalence.

Lemma 2.

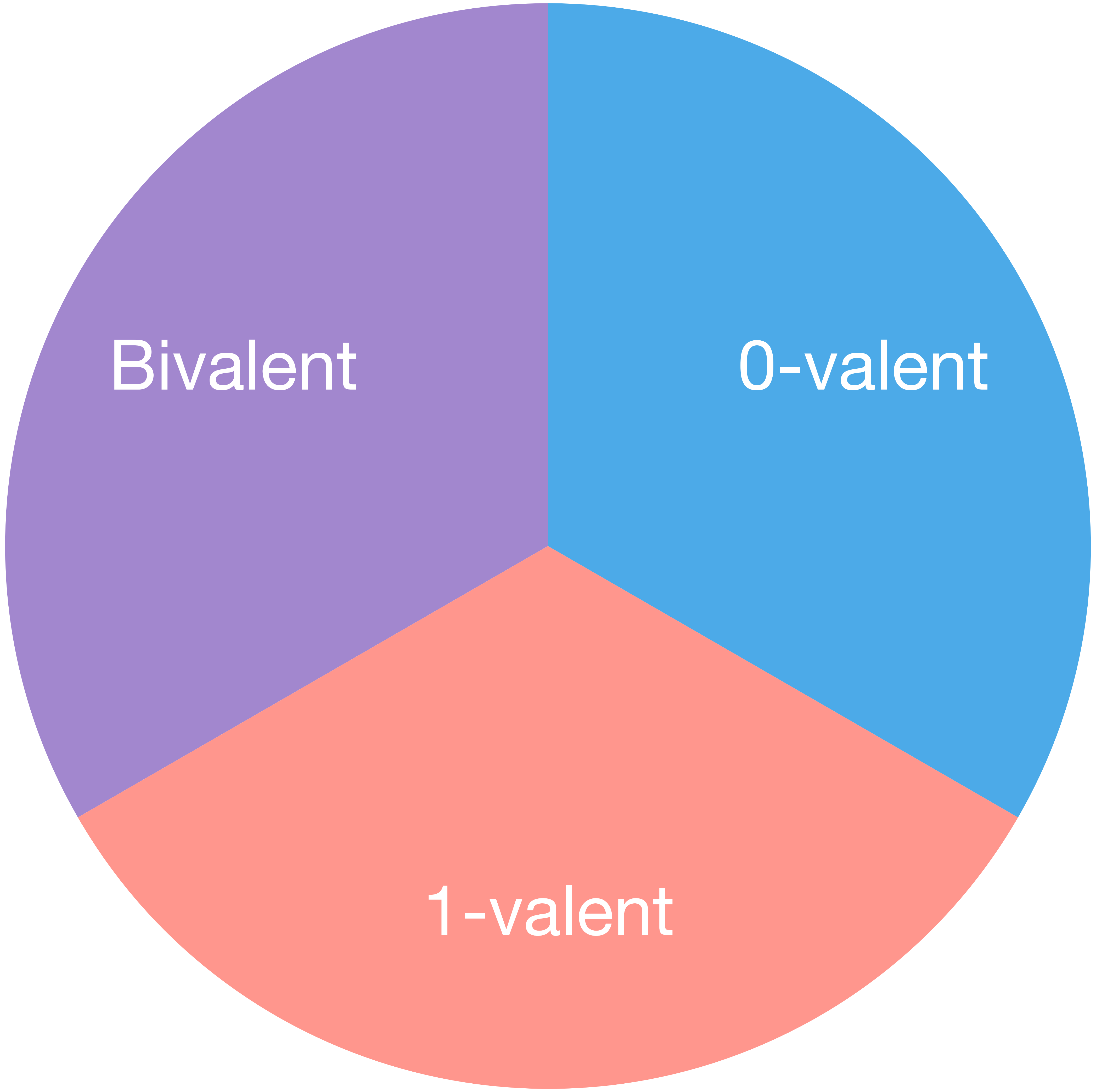
If Claim 2 does not hold,

then there exists a bivalent C and two steps e, e' operating on the same process p such that

- $e(C)$ is a i-valent configuration.

Claim 2

There exists a schedule that preserves bivalence.



Lemma 2.

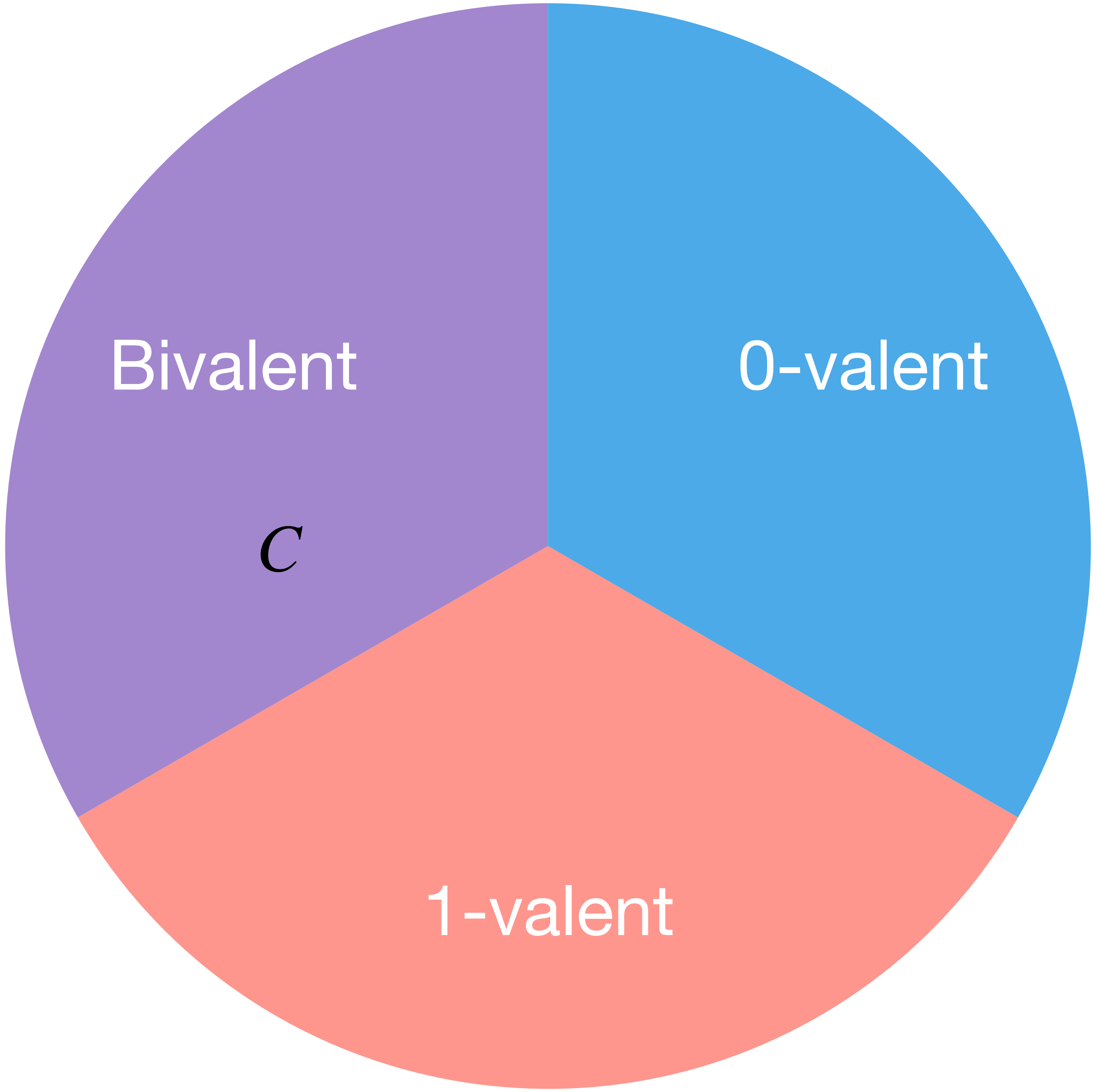
If Claim 2 does not hold,

then there exists a bivalent C and two steps e, e' operating on the same process p such that

- $e(C)$ is a i-valent configuration.
- $e(e'(C))$ is an $(1-i)$-valent configuration.

Claim 2

There exists a schedule that preserves bivalence.



Lemma 2.

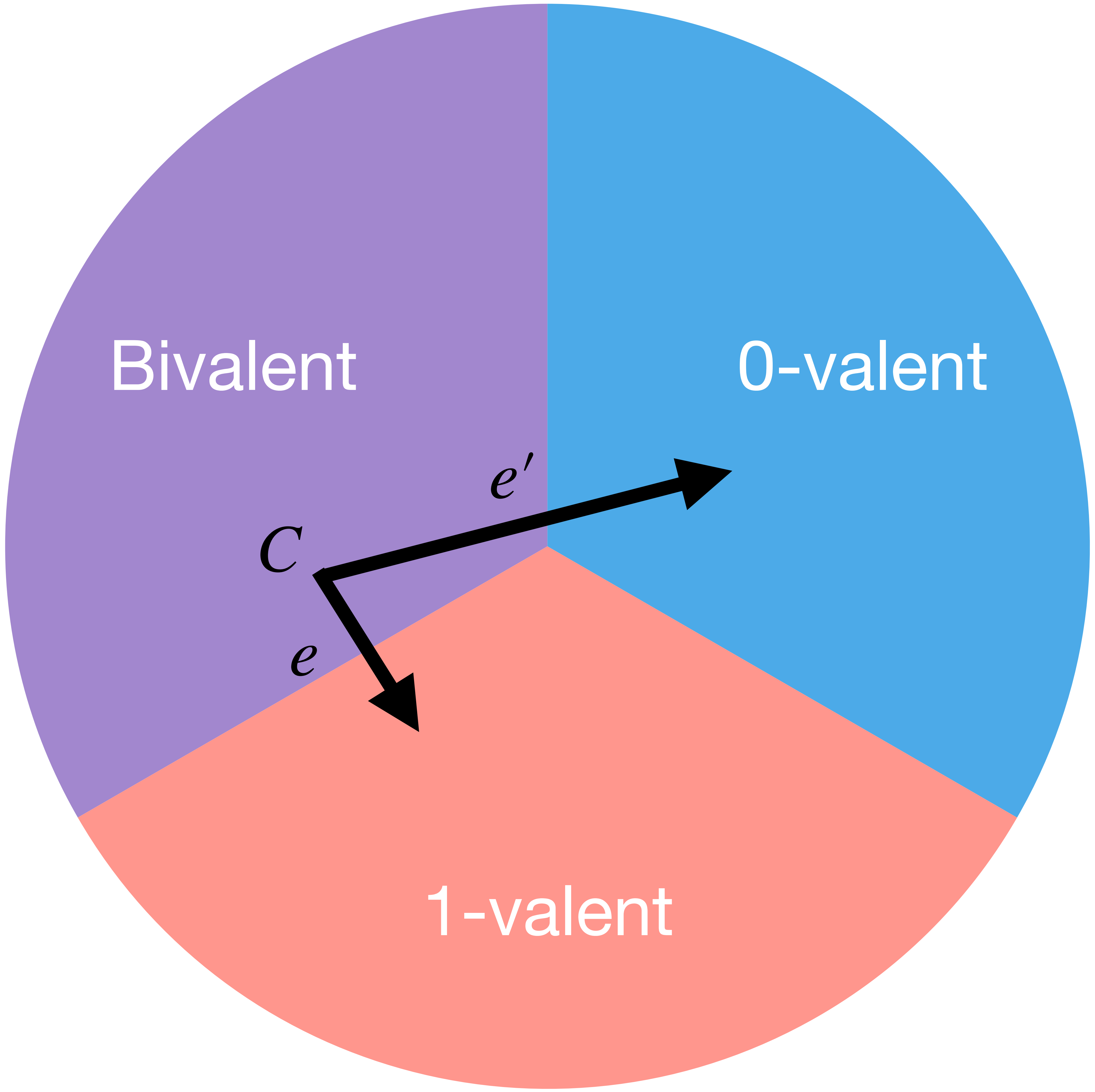
If Claim 2 does not hold,

then there exists a bivalent C and two steps e, e' operating on the same process p such that

- $e(C)$ is a i-valent configuration.
- $e(e'(C))$ is an $(1-i)$-valent configuration.

Claim 2

There exists a schedule that preserves bivalence.



Lemma 2.

If Claim 2 does not hold,

then there exists a bivalent C and two steps e, e' operating on the same process p such that

- $e(C)$ is a i-valent configuration.
- $e(e'(C))$ is an $(1-i)$-valent configuration.

Claim 2 (Proof for Lemma 2)

There exists a schedule that preserves bivalence.

Claim 2 (Proof for Lemma 2)

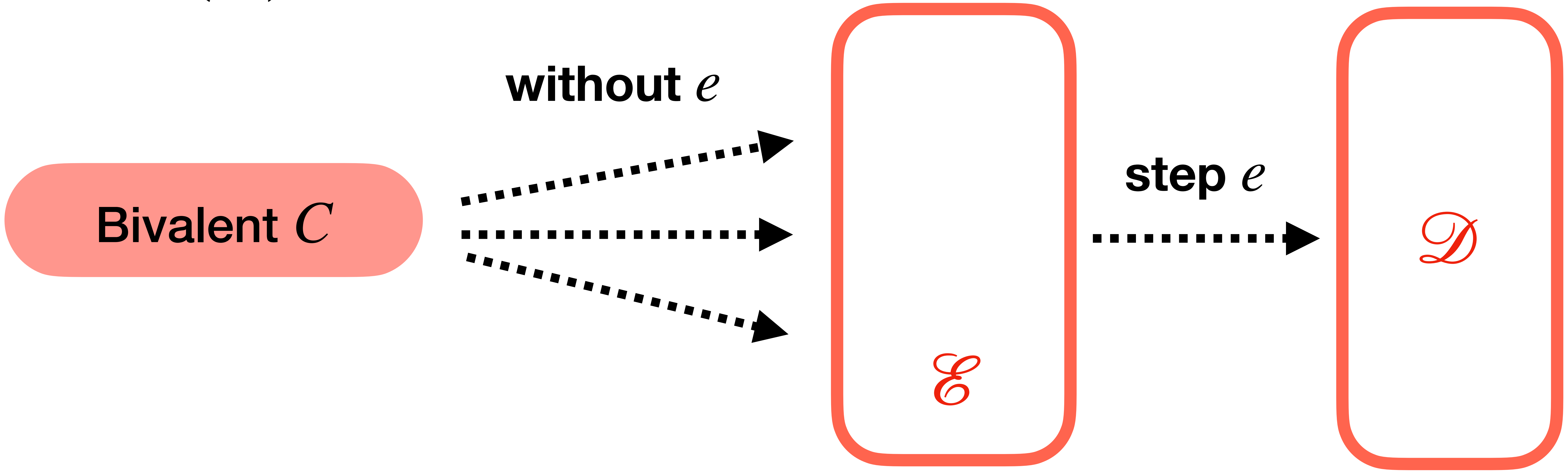
There exists a schedule that preserves bivalence.

- Let $\mathscr{E}$ be the set of configurations reachable from C without applying e. Let $\mathscr{D} = e(\mathscr{E})$.

Claim 2 (Proof for Lemma 2)

There exists a schedule that preserves bivalence.

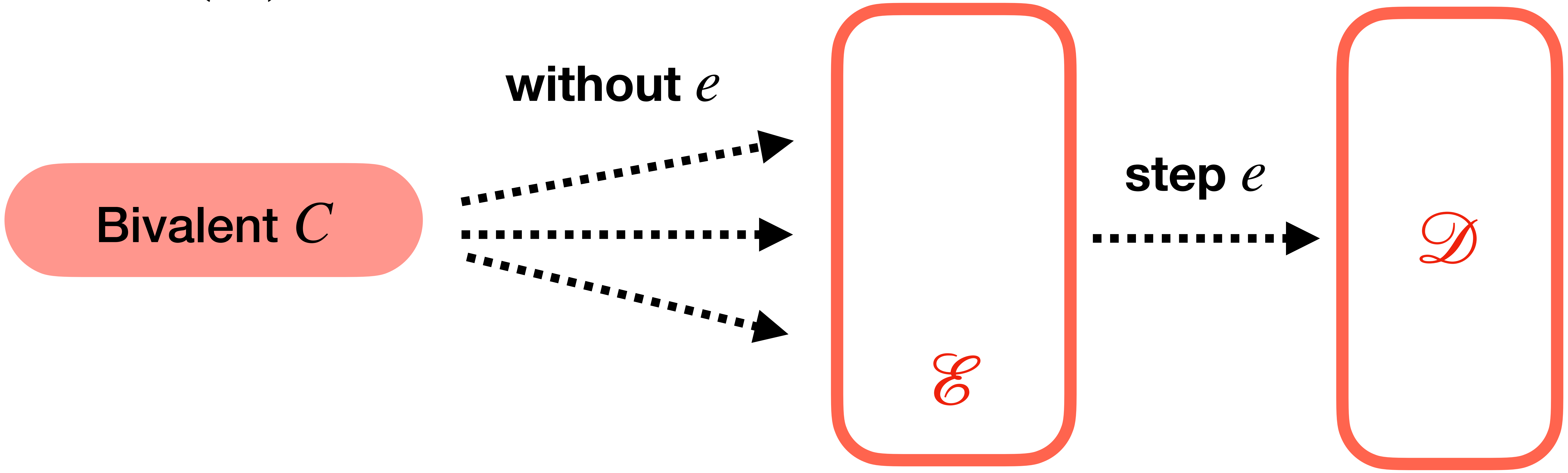
- Let $\mathscr{E}$ be the set of configurations reachable from C without applying e. Let $\mathscr{D} = e(\mathscr{E})$.



Claim 2 (Proof for Lemma 2)

There exists a schedule that preserves bivalence.

- Let $\mathscr{E}$ be the set of configurations reachable from C without applying e. Let $\mathscr{D} = e(\mathscr{E})$.

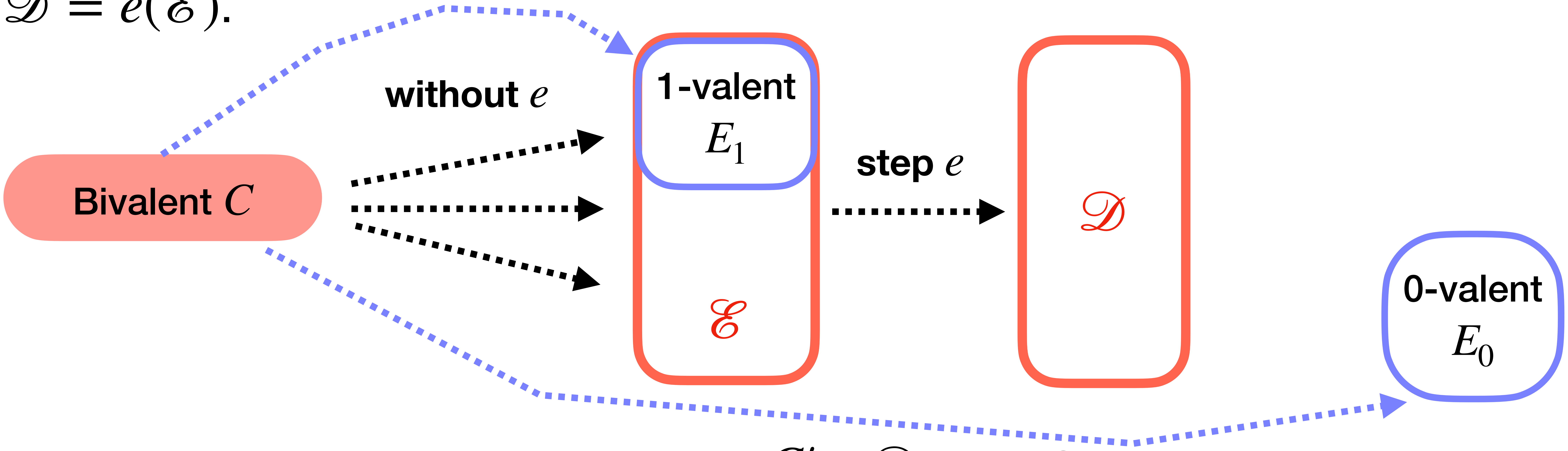


- If there exists a bivalent configuration C' in $\mathscr{D}$, then Claim 2 holds. **Contradiction to the assumption!** So there exists i-valent E_i.

Claim 2 (Proof for Lemma 2)

There exists a schedule that preserves bivalence.

- Let $\mathscr{E}$ be the set of configurations reachable from C without applying e. Let $\mathscr{D} = e(\mathscr{E})$.

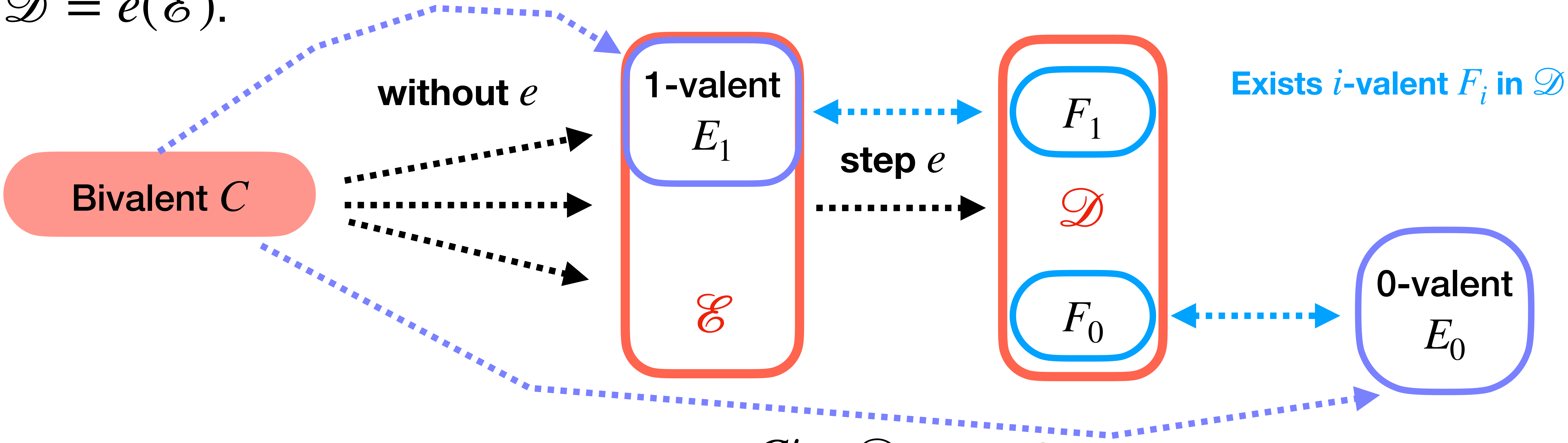


- If there exists a bivalent configuration C' in $\mathscr{D}$, then Claim 2 holds. **Contradiction to the assumption!** So there exists i-valent E_i.

Claim 2 (Proof for Lemma 2)

There exists a schedule that preserves bivalence.

- Let $\mathscr{E}$ be the set of configurations reachable from C without applying e. Let $\mathscr{D} = e(\mathscr{E})$.



- If there exists a bivalent configuration C' in $\mathscr{D}$, then Claim 2 holds. **Contradiction to the assumption!** So there exists i-valent E_i.

Claim 2 (Proof for Lemma 2)

There exists a schedule that preserves bivalence.

Claim 2 (Proof for Lemma 2)

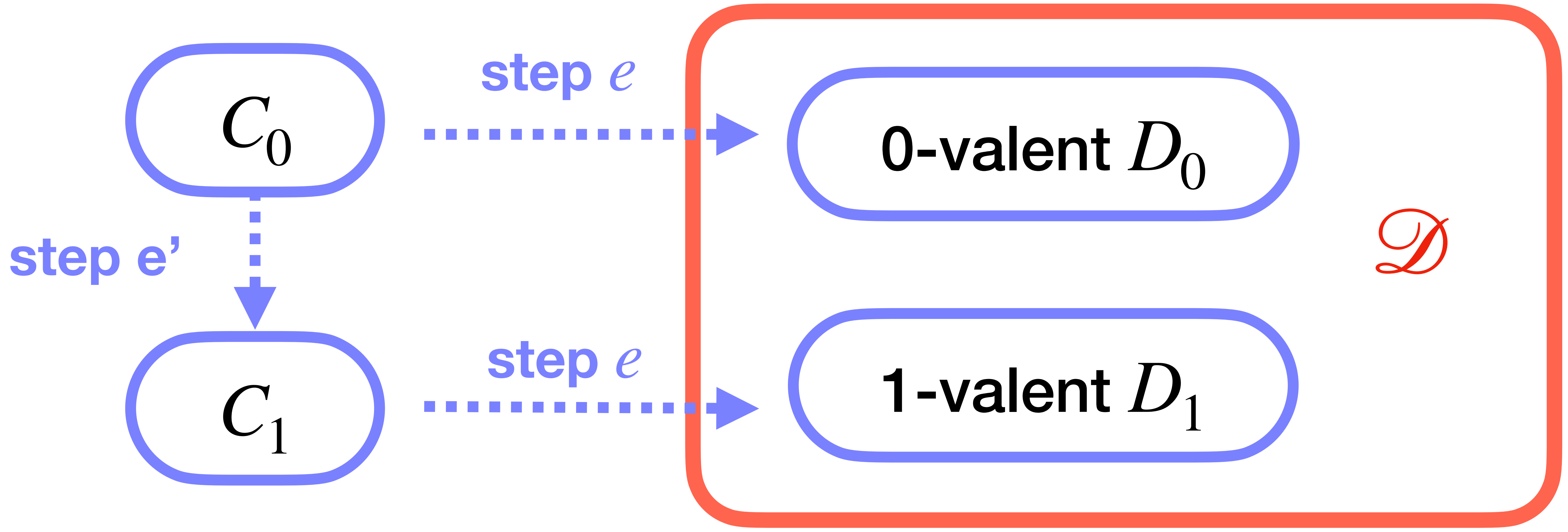
There exists a schedule that preserves bivalence.

- Skip some steps … we can prove that there exists

Claim 2 (Proof for Lemma 2)

There exists a schedule that preserves bivalence.

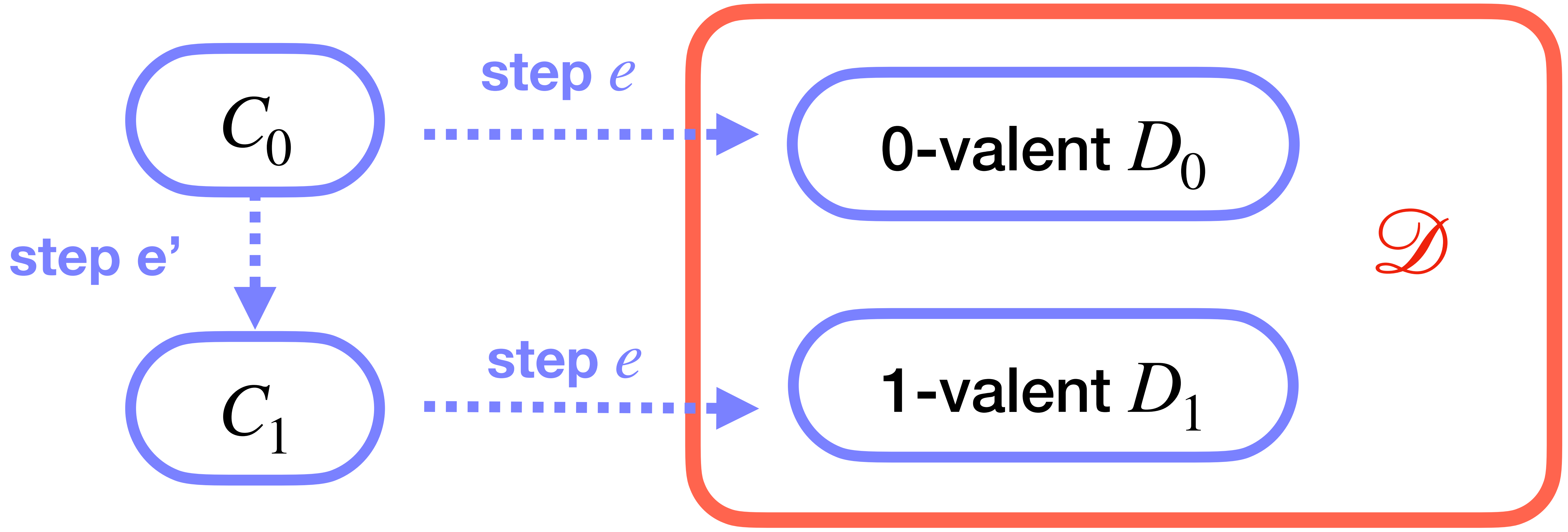
- Skip some steps … we can prove that there exists



Claim 2 (Proof for Lemma 2)

There exists a schedule that preserves bivalence.

- Skip some steps … we can prove that there exists

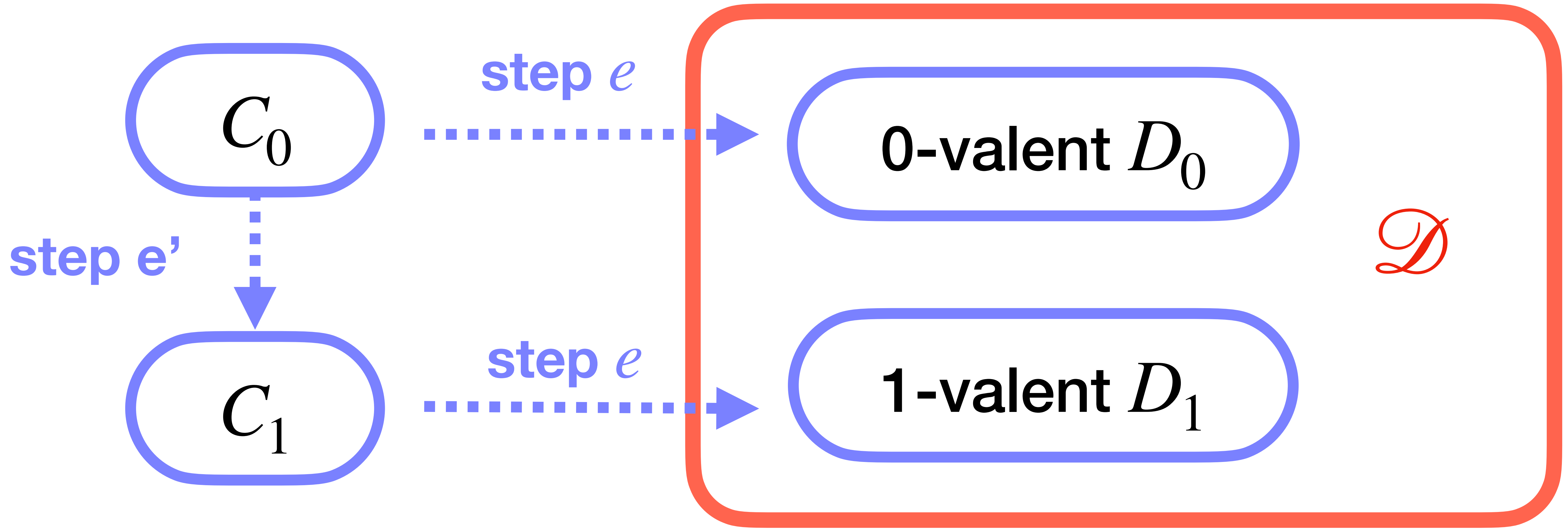


- If e and e' operate on different processors, then we can prove $(e; e')(C_0) = (e'; e)(C_0)$, which implies $D_0 = D_1$. **Impossible!**

Claim 2 (Proof for Lemma 2)

There exists a schedule that preserves bivalence.

- Skip some steps … we can prove that there exists

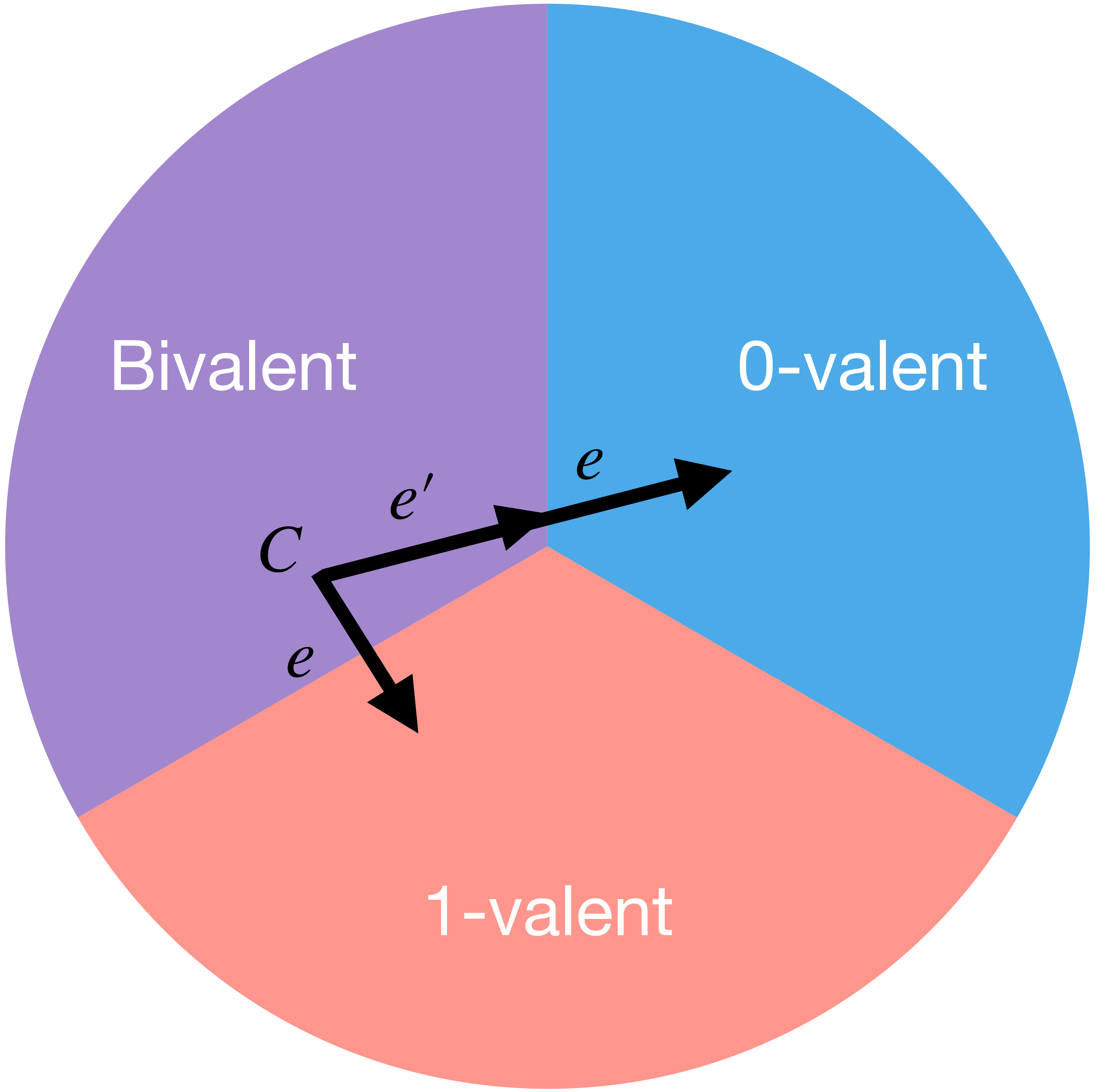


- If e and e' operate on different processors, then we can prove $(e; e')(C_0) = (e'; e)(C_0)$, which implies $D_0 = D_1$. **Impossible!**

- **Lemma 2 proved!**

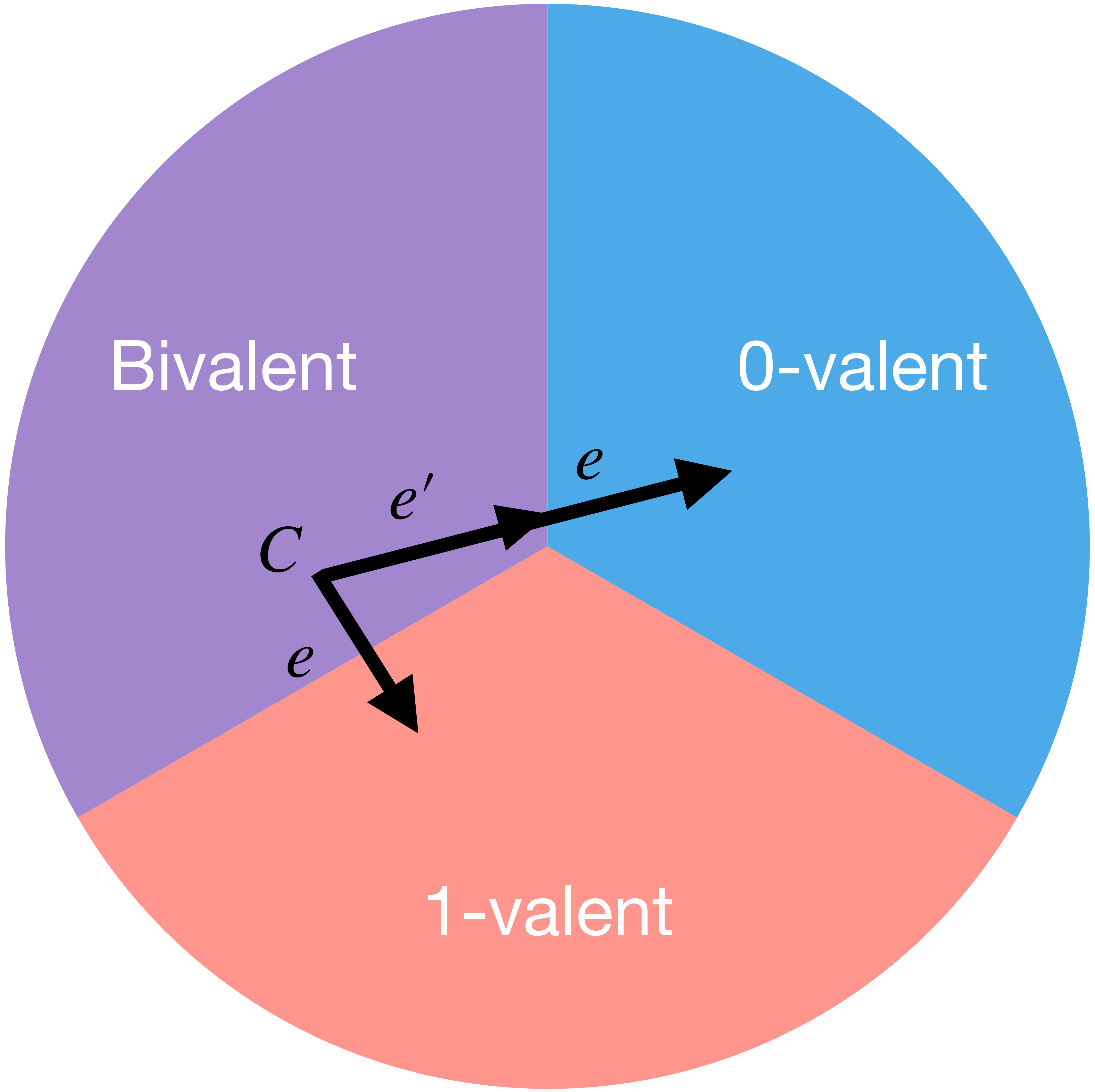
Claim 2

There exists a schedule that preserves bivalence.



Claim 2

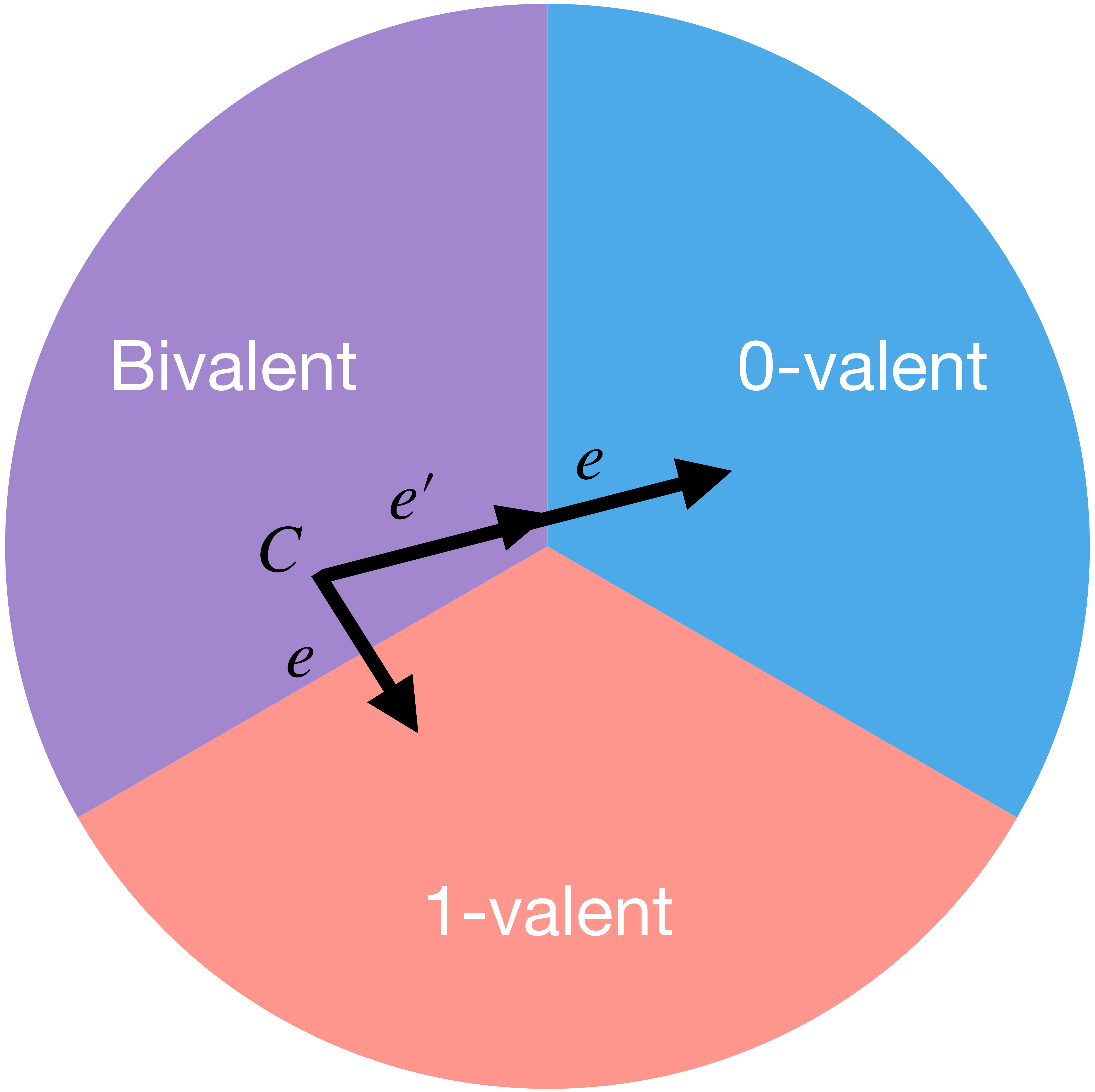
There exists a schedule that preserves bivalence.



Proof by Contradiction (again):

Claim 2

There exists a schedule that preserves bivalence.

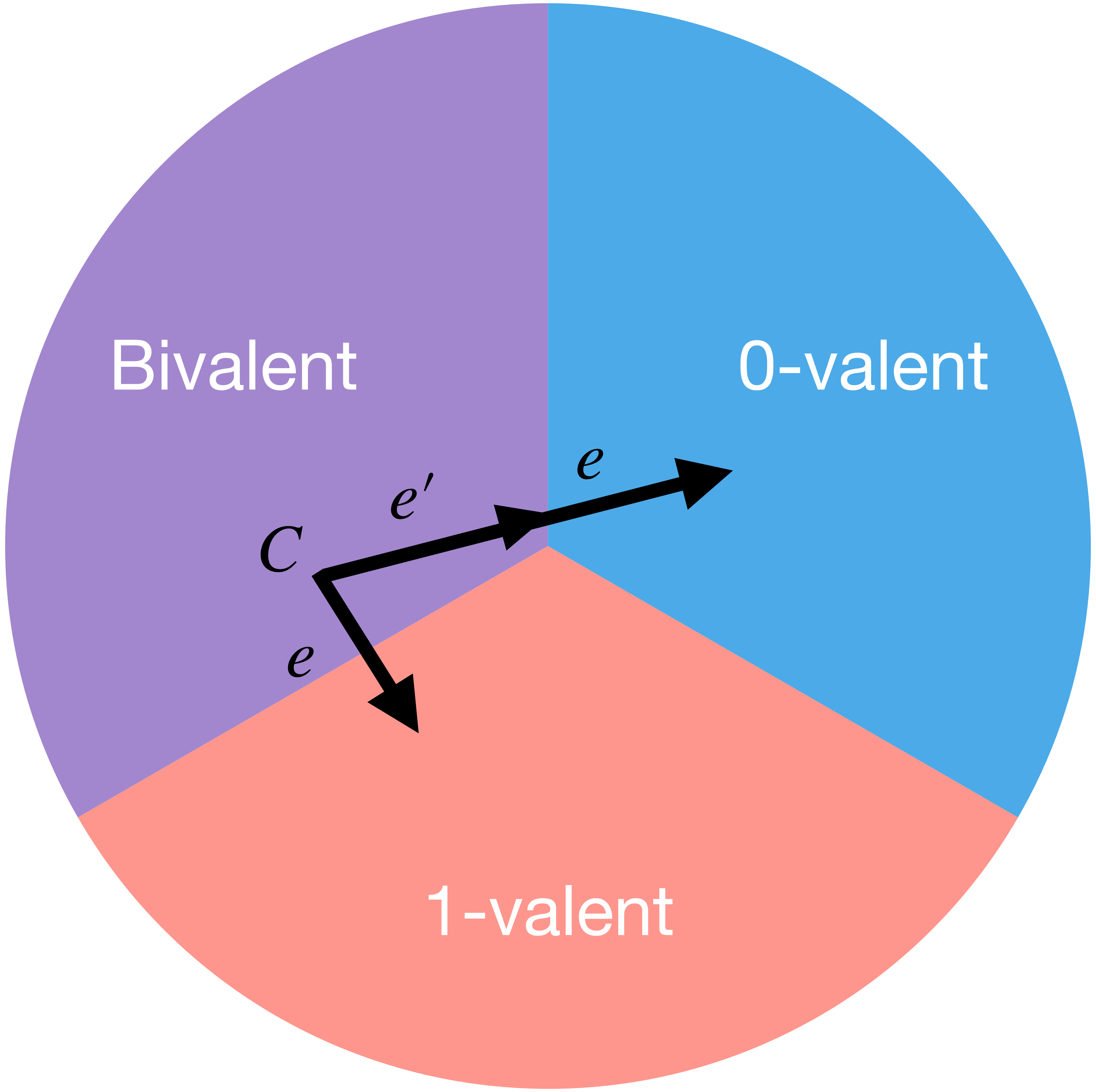


Proof by Contradiction (again):

Assume Claim 2 is not true, then by Lemma 2, there exists a bivalent C and two steps e, e' as depicted in the diagram and e and e' both operate on a process p.

Claim 2

There exists a schedule that preserves bivalence.

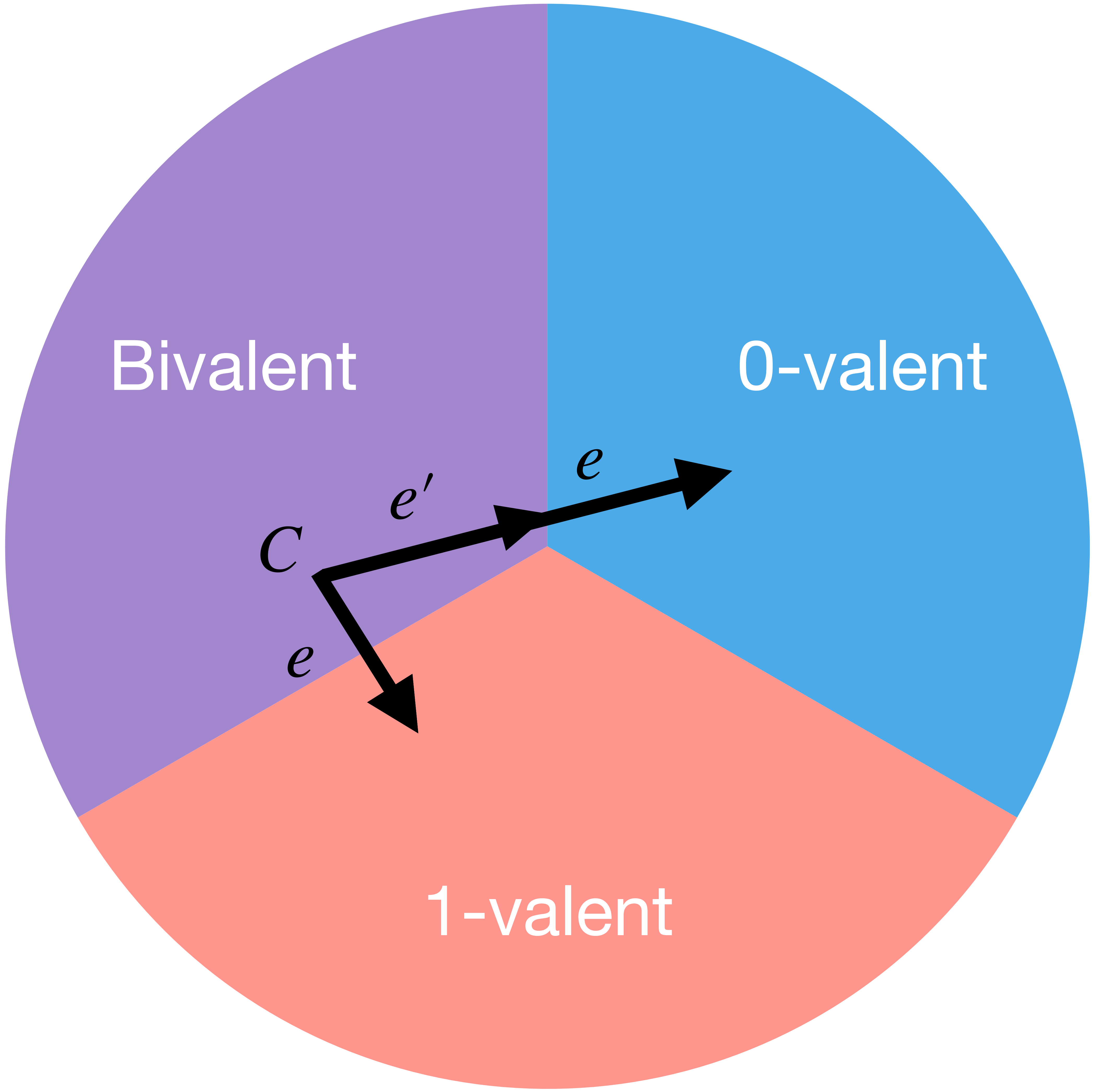


Proof by Contradiction (again):

Assume Claim 2 is not true, then by Lemma 2, there exists a bivalent C and two steps e, e' as depicted in the diagram and e and e' both operate on a process p.

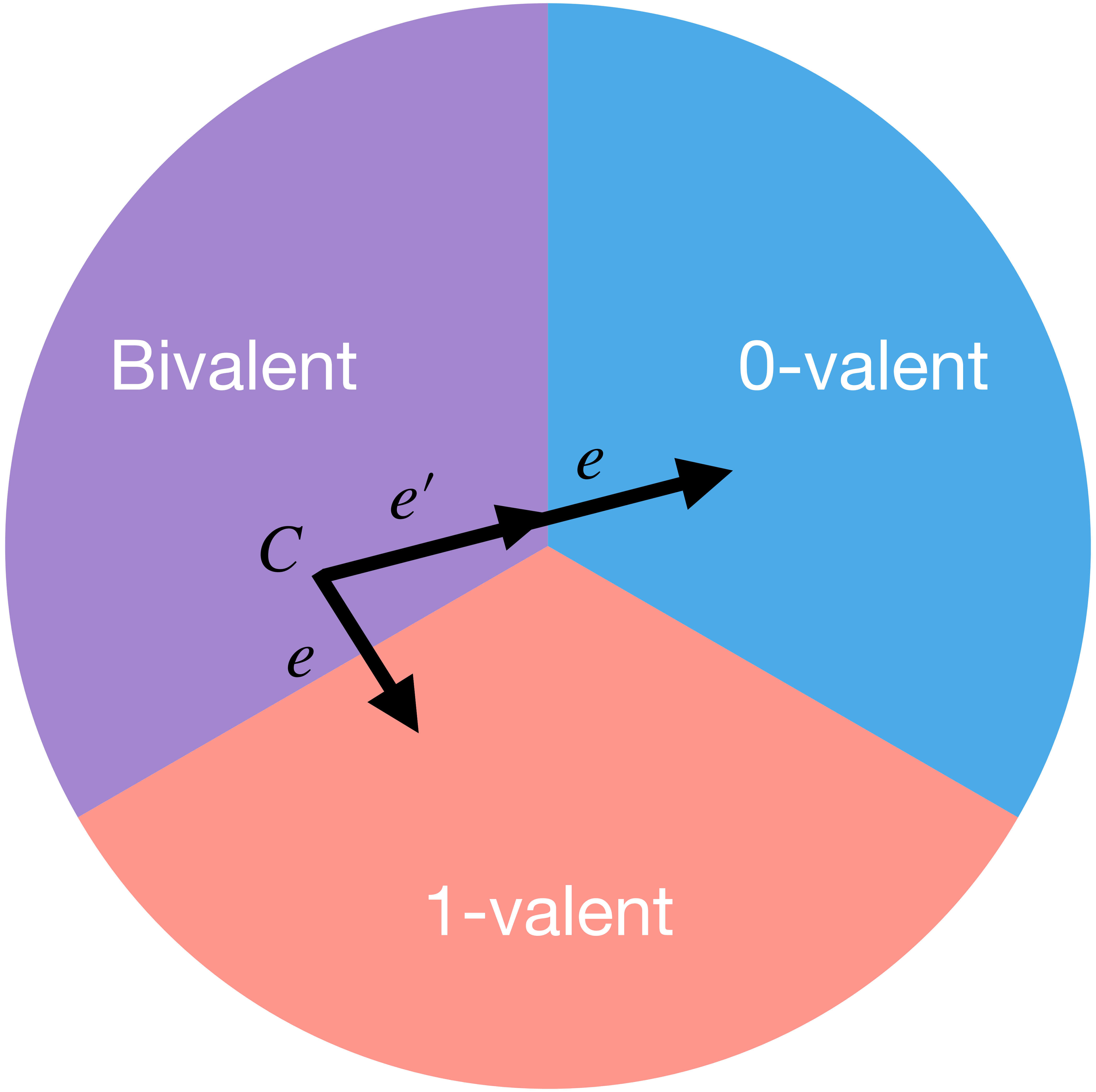
Claim 2

There exists a schedule that preserves bivalence.



Claim 2

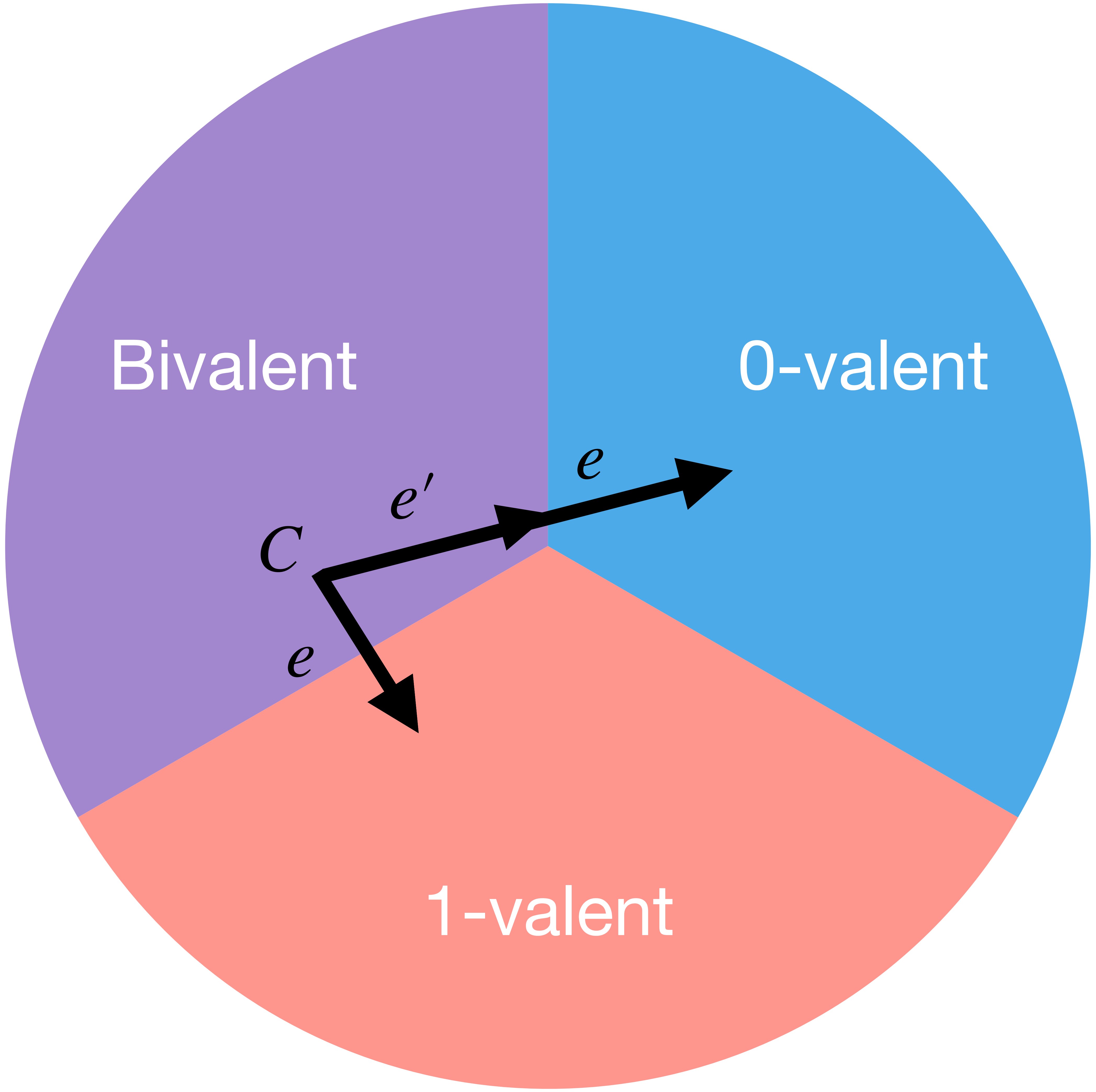
There exists a schedule that preserves bivalence.



There exists schedule σ that leads C to a consensus A without stepping p.

Claim 2

There exists a schedule that preserves bivalence.

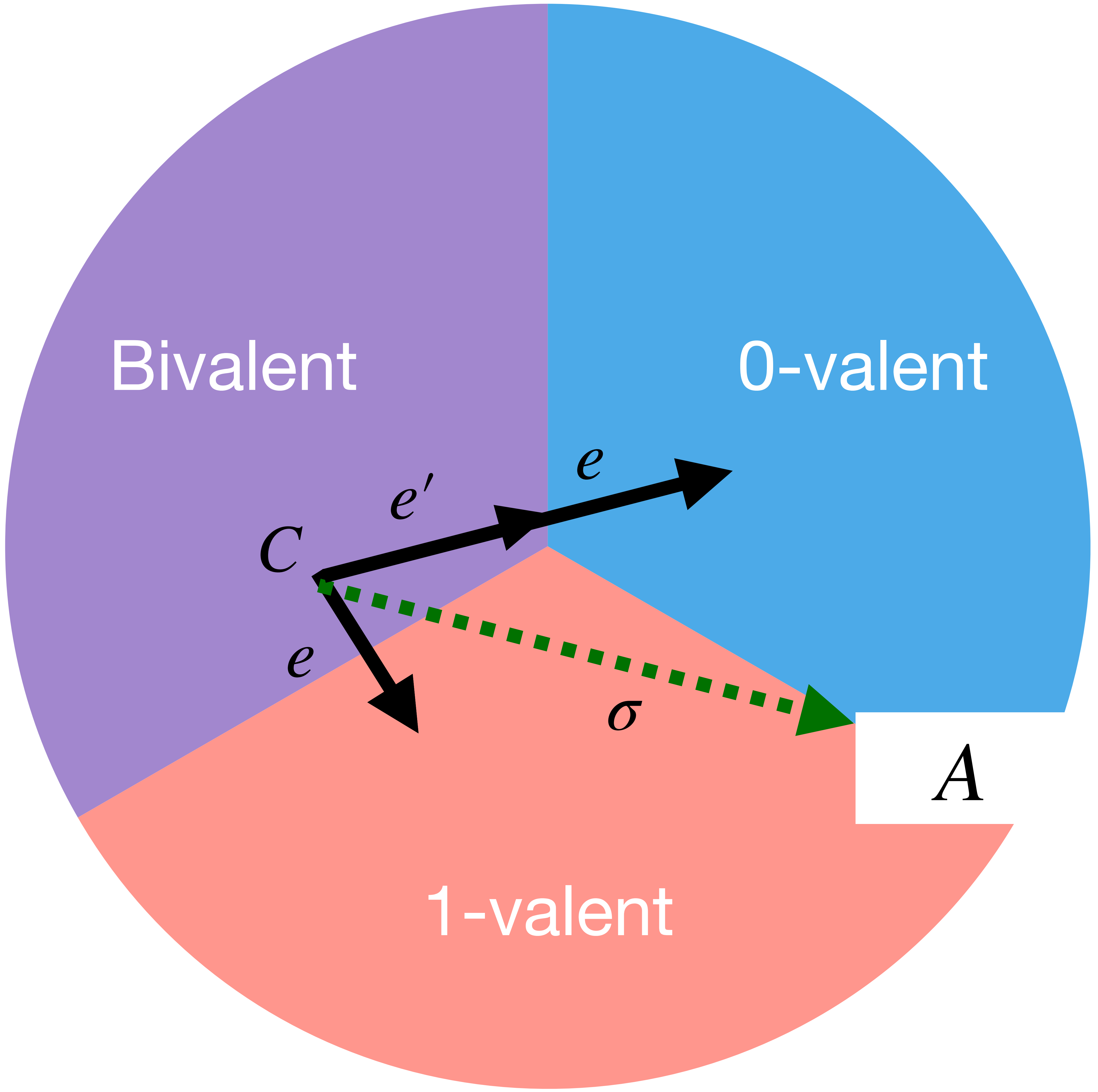


There exists schedule σ that leads C to a consensus A without stepping p.

By lemma 1, $\sigma(e(C)) = e(\sigma(C))$, so $e(\sigma(C))$ has to be 1-valent.

Claim 2

There exists a schedule that preserves bivalence.

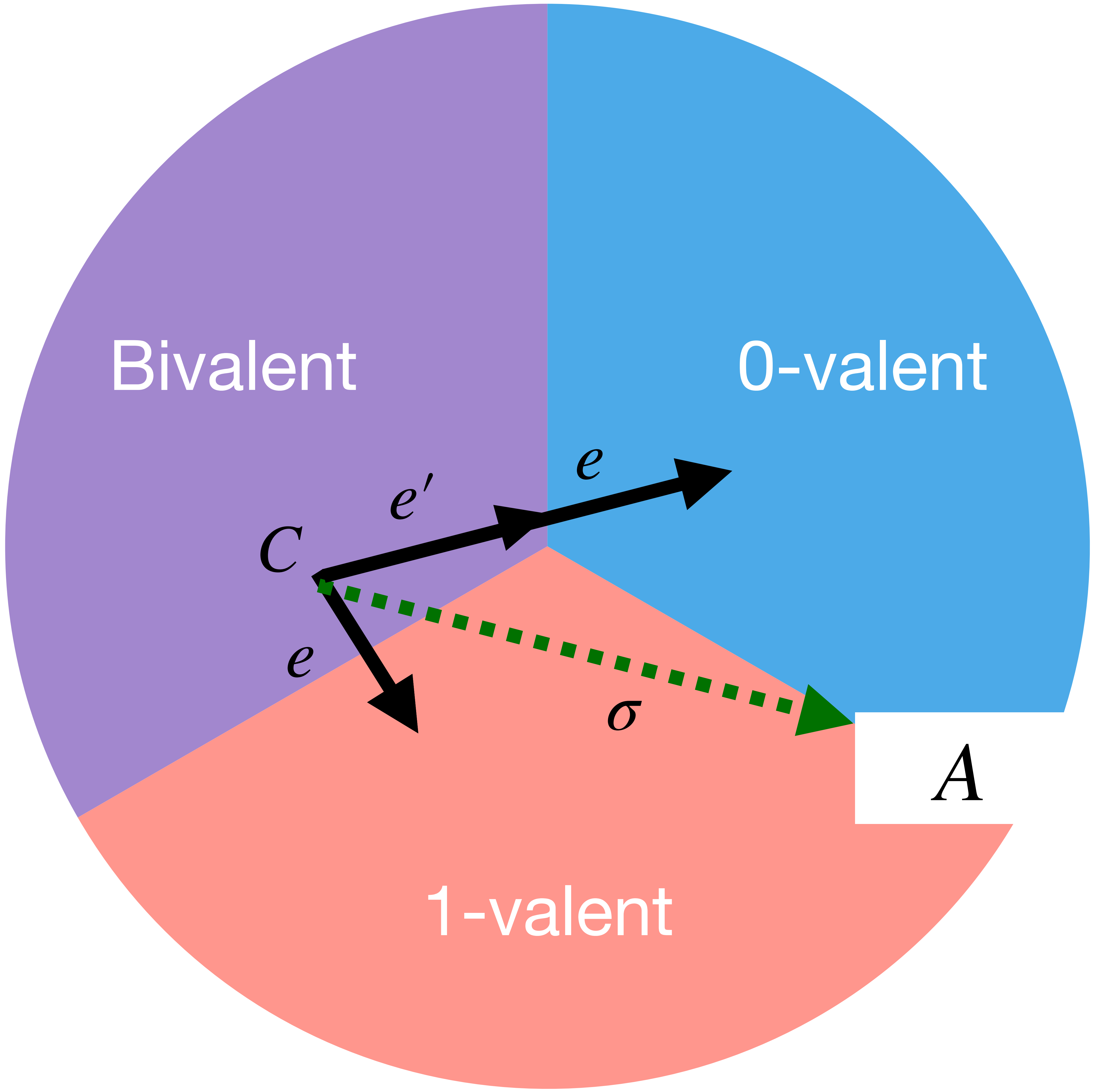


There exists schedule σ that leads C to a consensus A without stepping p.

By lemma 1, $\sigma(e(C)) = e(\sigma(C))$, so $e(\sigma(C))$ has to be 1-valent.

Claim 2

There exists a schedule that preserves bivalence.



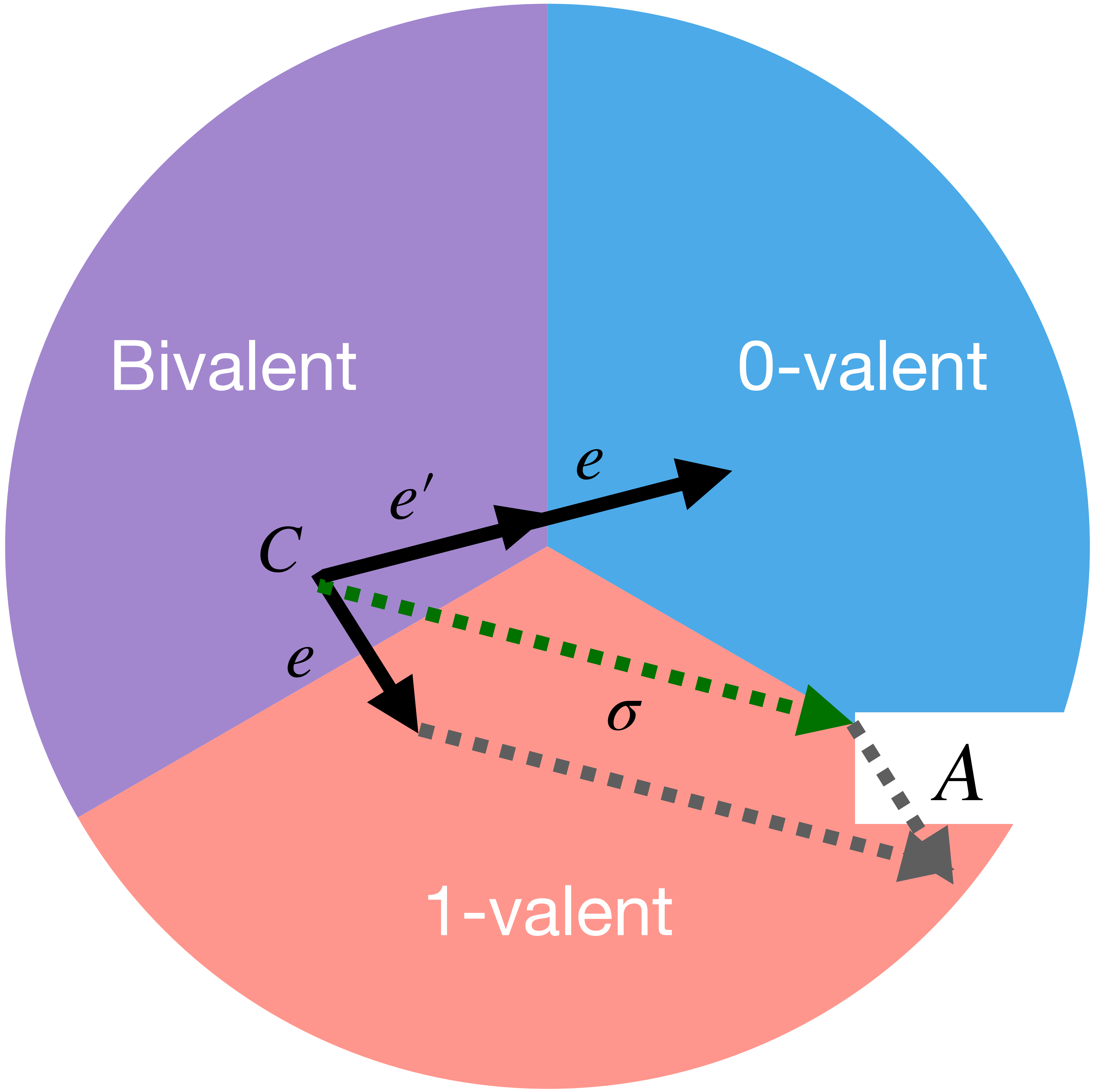
There exists schedule σ that leads C to a consensus A without stepping p.

By lemma 1, $\sigma(e(C)) = e(\sigma(C))$, so $e(\sigma(C))$ has to be 1-valent.

Similarly, $\sigma(e(e'(C))) = e(e'(\sigma(C)))$, so $e(e'(\sigma(C)))$ has to be 0-valent.

Claim 2

There exists a schedule that preserves bivalence.



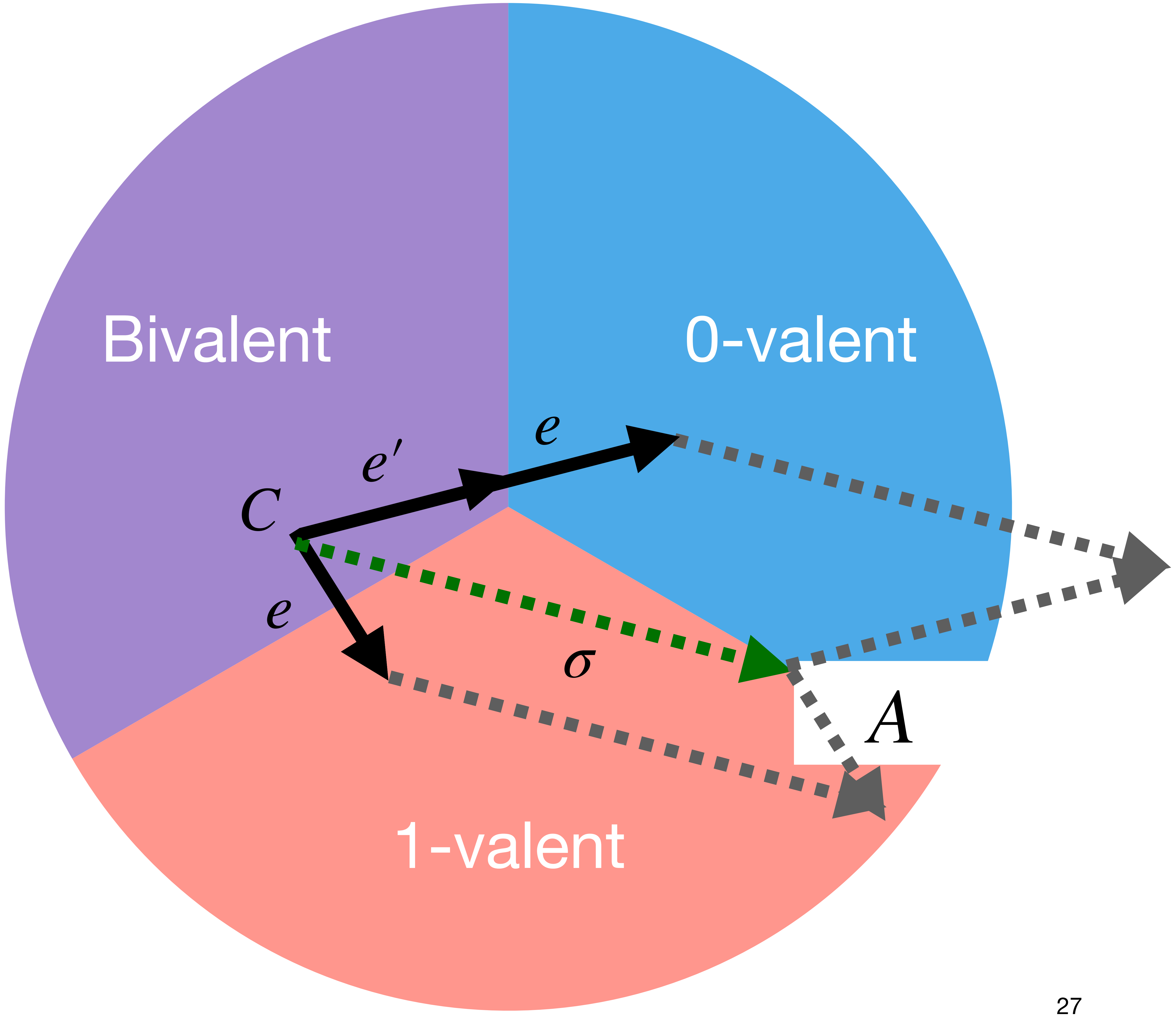
There exists schedule σ that leads C to a consensus A without stepping p.

By lemma 1, $\sigma(e(C)) = e(\sigma(C))$, so $e(\sigma(C))$ has to be 1-valent.

Similarly, $\sigma(e(e'(C))) = e(e'(\sigma(C)))$, so $e(e'(\sigma(C)))$ has to be 0-valent.

Claim 2

There exists a schedule that preserves bivalence.



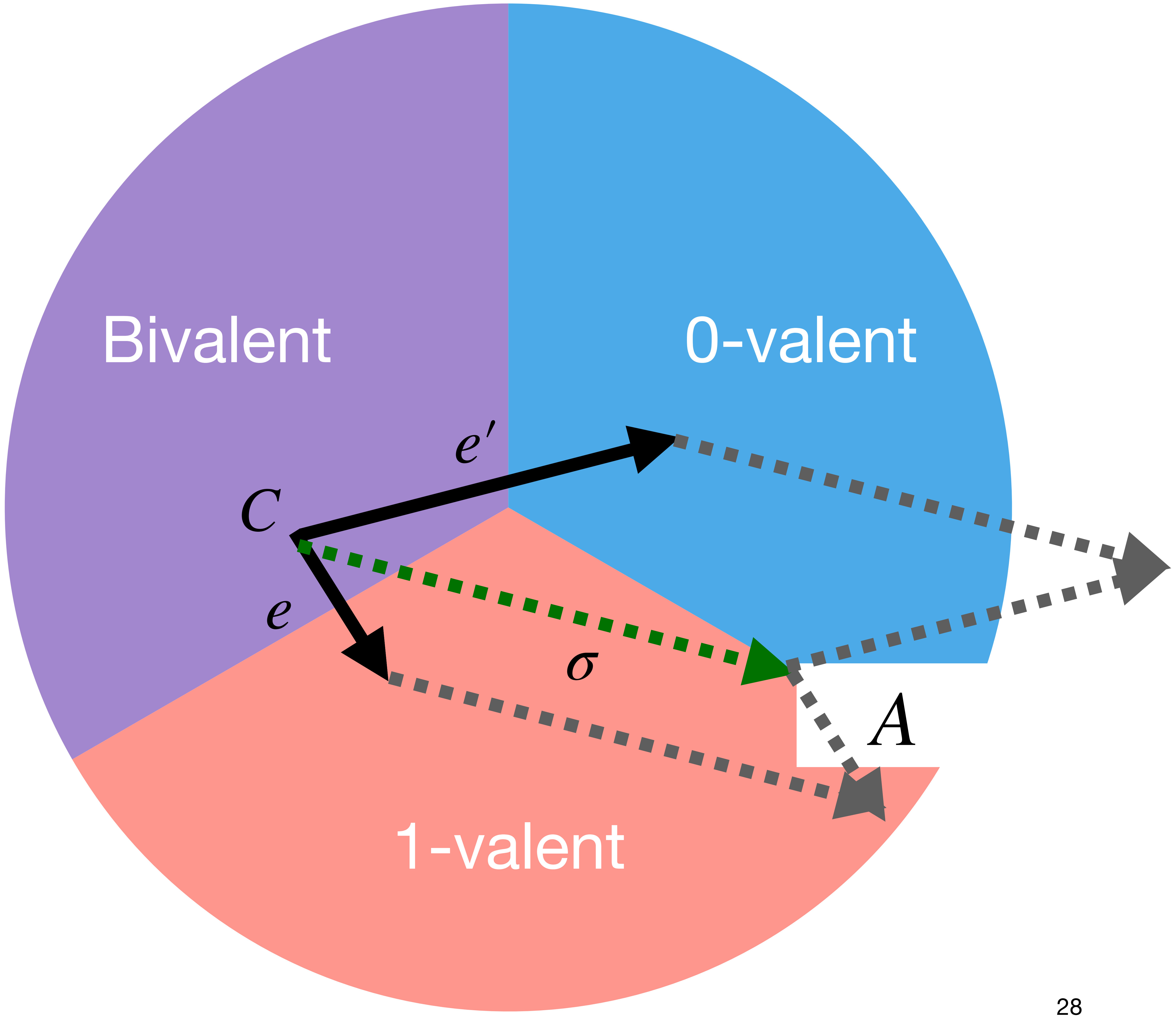
There exists schedule σ that leads C to a consensus A without stepping p.

By lemma 1, $\sigma(e(C)) = e(\sigma(C))$, so $e(\sigma(C))$ has to be 1-valent.

Similarly, $\sigma(e(e'(C))) = e(e'(\sigma(C)))$, so $e(e'(\sigma(C)))$ has to be 0-valent.

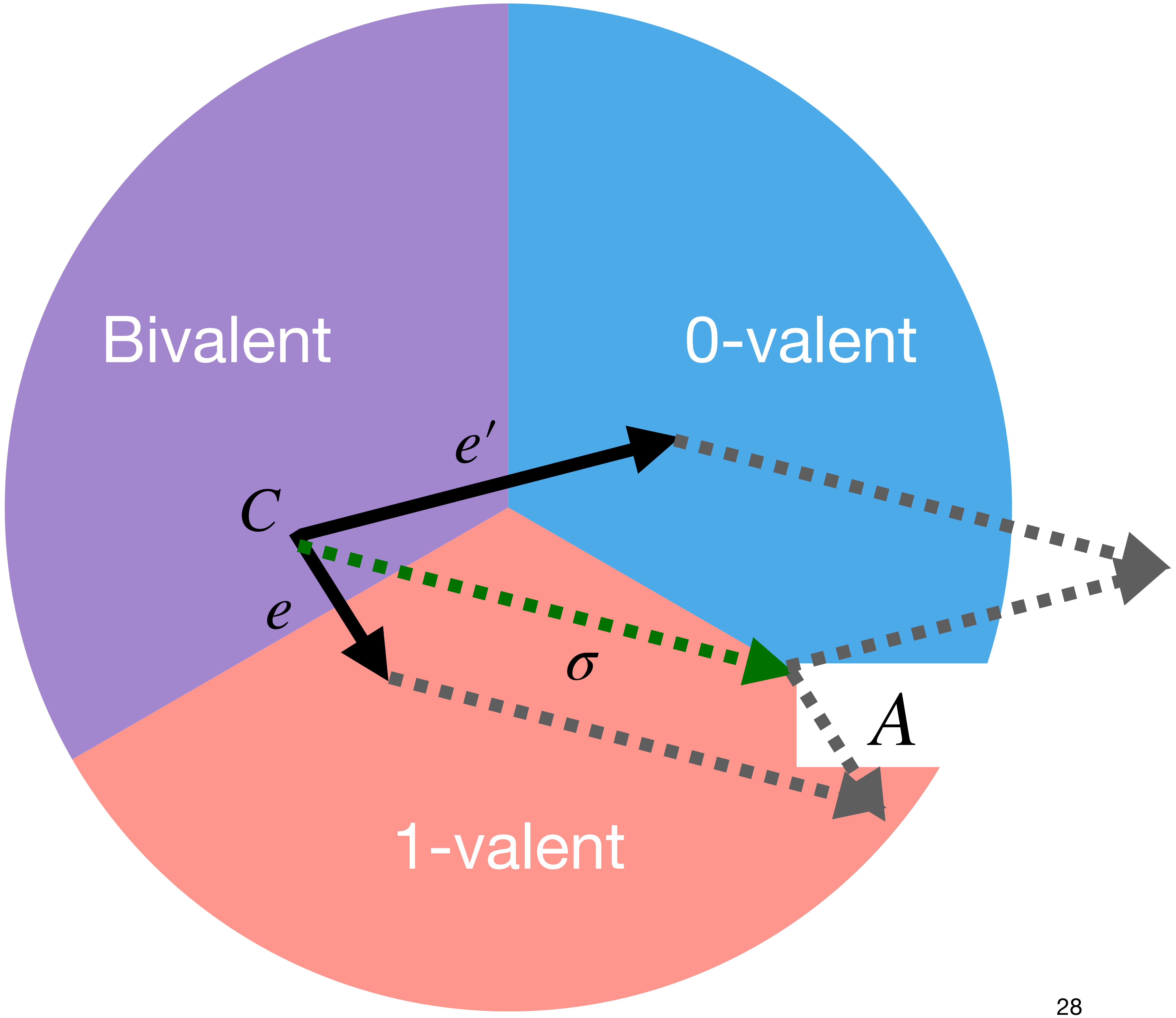
Claim 2

There exists a schedule that preserves bivalence.



Claim 2

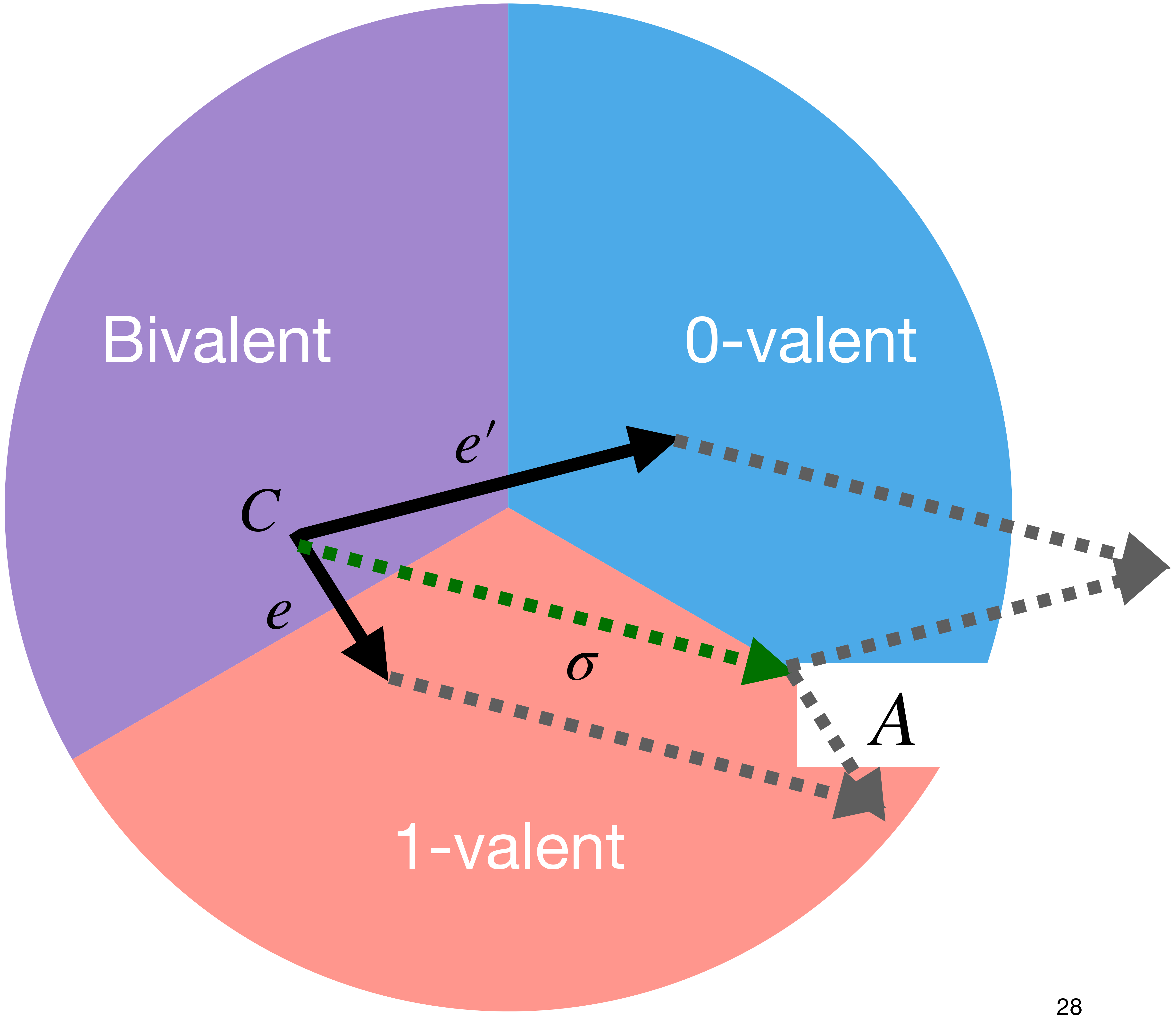
There exists a schedule that preserves bivalence.



$e(\sigma(C))$ has to be 1-valent implies $A = \sigma(C)$ cannot be 0-valent.

Claim 2

There exists a schedule that preserves bivalence.

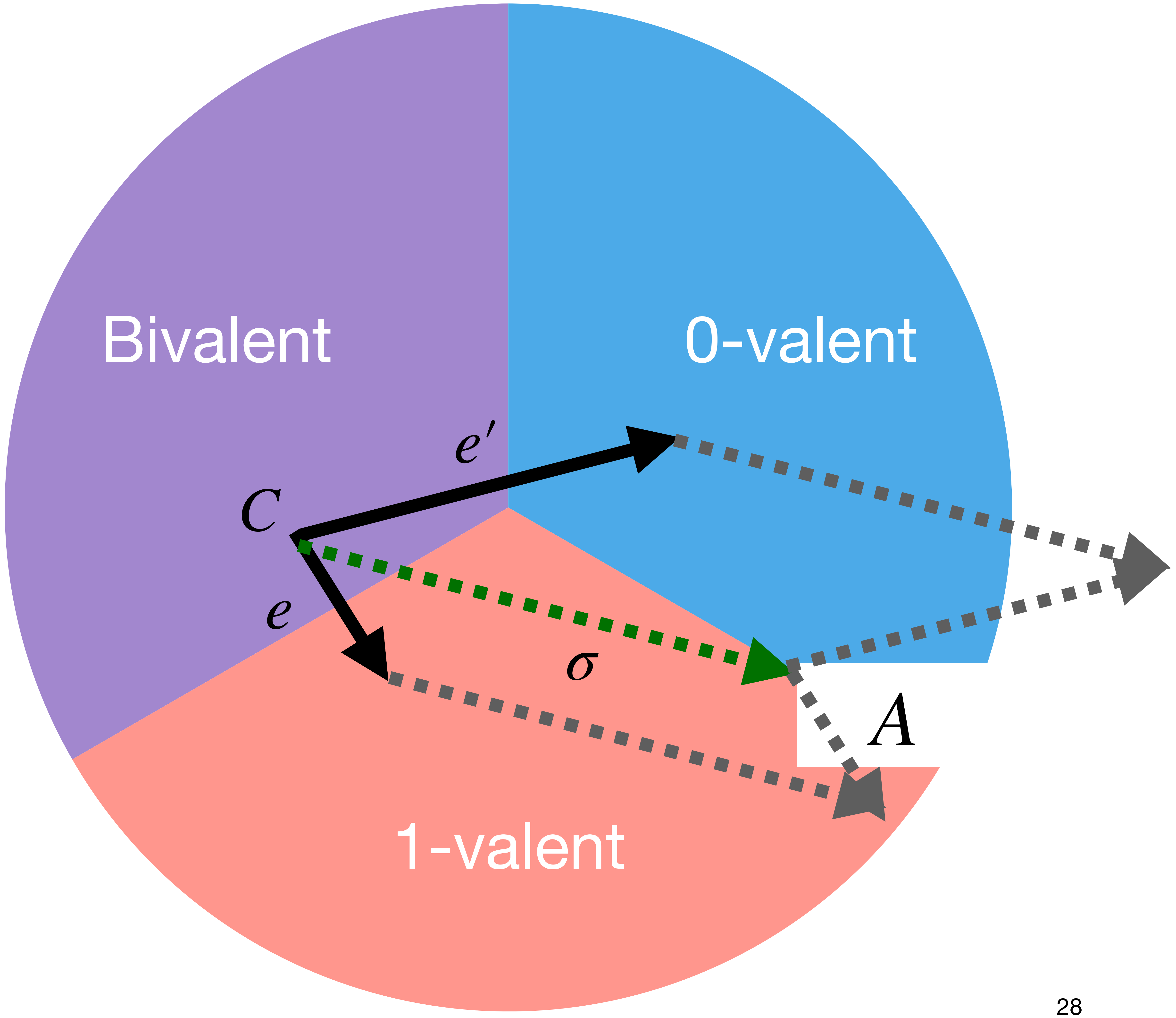


$e(\sigma(C))$ has to be 1-valent implies $A = \sigma(C)$ cannot be 0-valent.

$e(e'(\sigma(C)))$ has to be 0-valent implies $A = \sigma(C)$ cannot be 1-valent.

Claim 2

There exists a schedule that preserves bivalence.



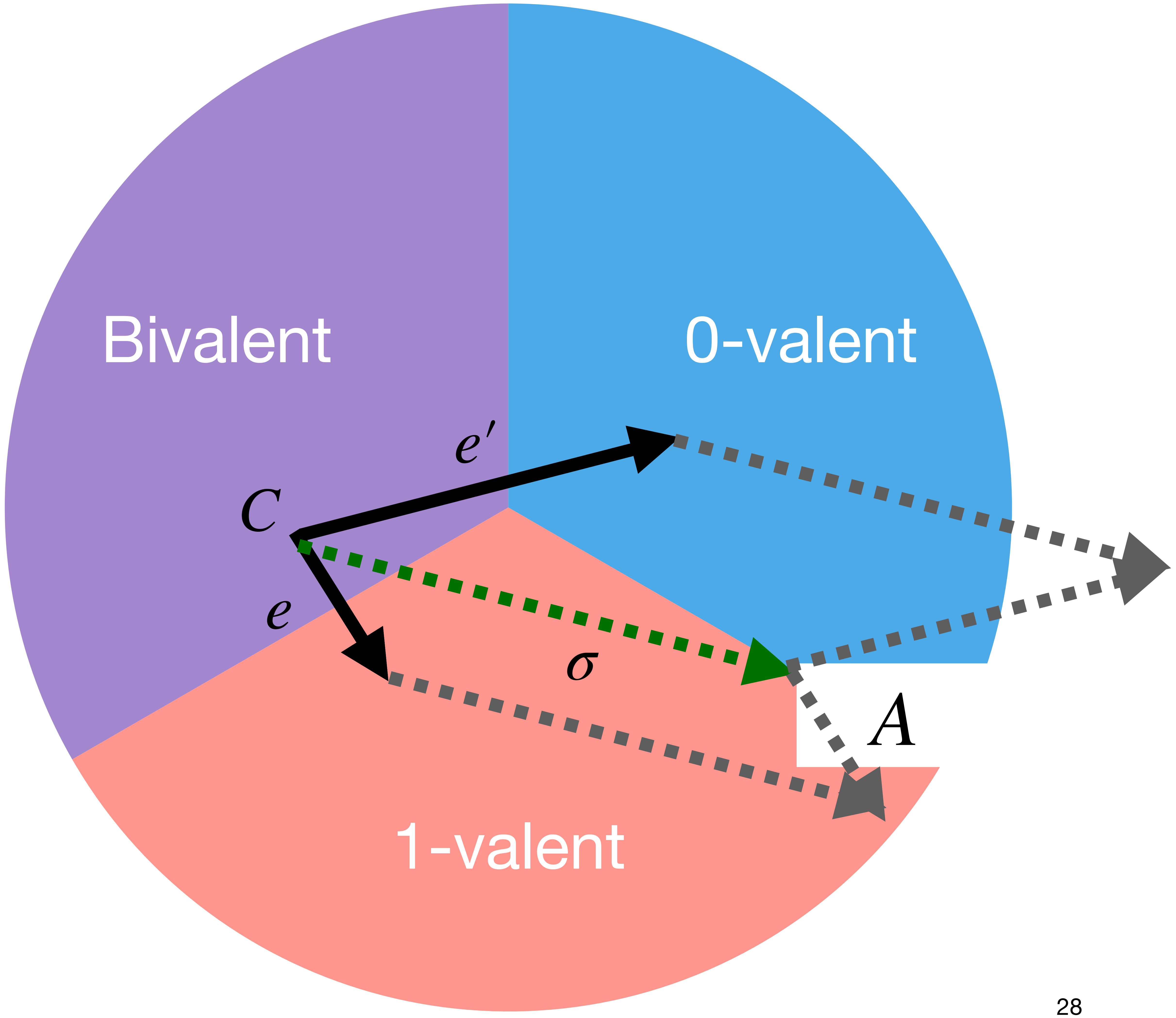
$e(\sigma(C))$ has to be 1-valent implies
$A = \sigma(C)$ cannot be 0-valent.

$e(e'(\sigma(C)))$ has to be 0-valent implies
$A = \sigma(C)$ cannot be 1-valent.

$A = \sigma(C)$ has to be bivalent.

Claim 2

There exists a schedule that preserves bivalence.



$e(\sigma(C))$ has to be 1-valent implies
$A = \sigma(C)$ cannot be 0-valent.

$e(e'(\sigma(C)))$ has to be 0-valent implies
$A = \sigma(C)$ cannot be 1-valent.

$A = \sigma(C)$ has to be bivalent.

Claim 2 proved (with details omitted)!

Discussion

Review the Proof

Discussion

Review the Proof

- Where did the proof use the condition that one process might be faulty?

Discussion

Review the Proof

- Where did the proof use the condition that one process might be faulty?
- Are there other implicit assumptions of the set of initial configurations in P?

Discussion

What are remedies for this impossibility results?

Discussion

What are remedies for this impossibility results?

- The authors considered a case where faulty processes are all dead from the beginning and prove that there exists a system that satisfy **partial correctness** (Agreement and Non-triviality).

Discussion

What are remedies for this impossibility results?

- The authors considered a case where faulty processes are all dead from the beginning and prove that there exists a system that satisfy **partial correctness** (Agreement and Non-triviality).

- What relaxations of the adversarial environment are effective?

    - Is a totally correct protocol possible if the message is delivered in order?

Discussion

What are remedies for this impossibility results?

- The authors considered a case where faulty processes are all dead from the beginning and prove that there exists a system that satisfy **partial correctness** (Agreement and Non-triviality).
- What relaxations of the adversarial environment are effective?
    - Is a totally correct protocol possible if the message is delivered in order?
- What enhancements of the processes would be effective?
    - Is a totally correct protocol possible if the processes can detect the faulty process?