Impossibility of Consensus in Asynchronous Environments

Ellis Michael

Consensus

n processes, all of which have an input value from some domain.

Processes output a value by calling *decide*(v).

Non-faulty processes continue correctly executing protocol steps forever.

We denote the number of faulty processes f.

- **Agreement:** No two correct processes decide different values.
- **Integrity:** Every correct process decides at most one value, and if a correct process decides a value v, some process had v as its input.
- **Termination:** Every correct process eventually decides a value.

Binary Consensus

n processes, all of which have an input value from {0, 1}. Processes output a value by calling *decide*(v).

Non-faulty processes continue correctly executing protocol steps forever. We denote the number of faulty processes f. Here, we only consider **crash failures**.

- **Agreement:** No two processes decide different values.
- **Integrity:** Every process decides at most one value, and if a process decides a value v, some process had v as its input.
- **Termination:** Every correct process eventually decides a value.

BINARY CONSENSUS

n processes, all of which have an input value from {0, 1}. Processes output a value by calling *decide*(v).

Non-faulty processes continue correctly executing protocol steps forever. We denote the number of fa … consider **crash failures**.

If you can solve consensus, you can solve binary consensus.

- **Agreement:** No tw … ues.
- **Integrity:** Every process decides at most one value, and if a process decides a value v, some process had v as its input.
- **Termination:** Every correct process eventually decides a value.

Aside: Both safety and liveness properties are necessary to create a meaningful specification!

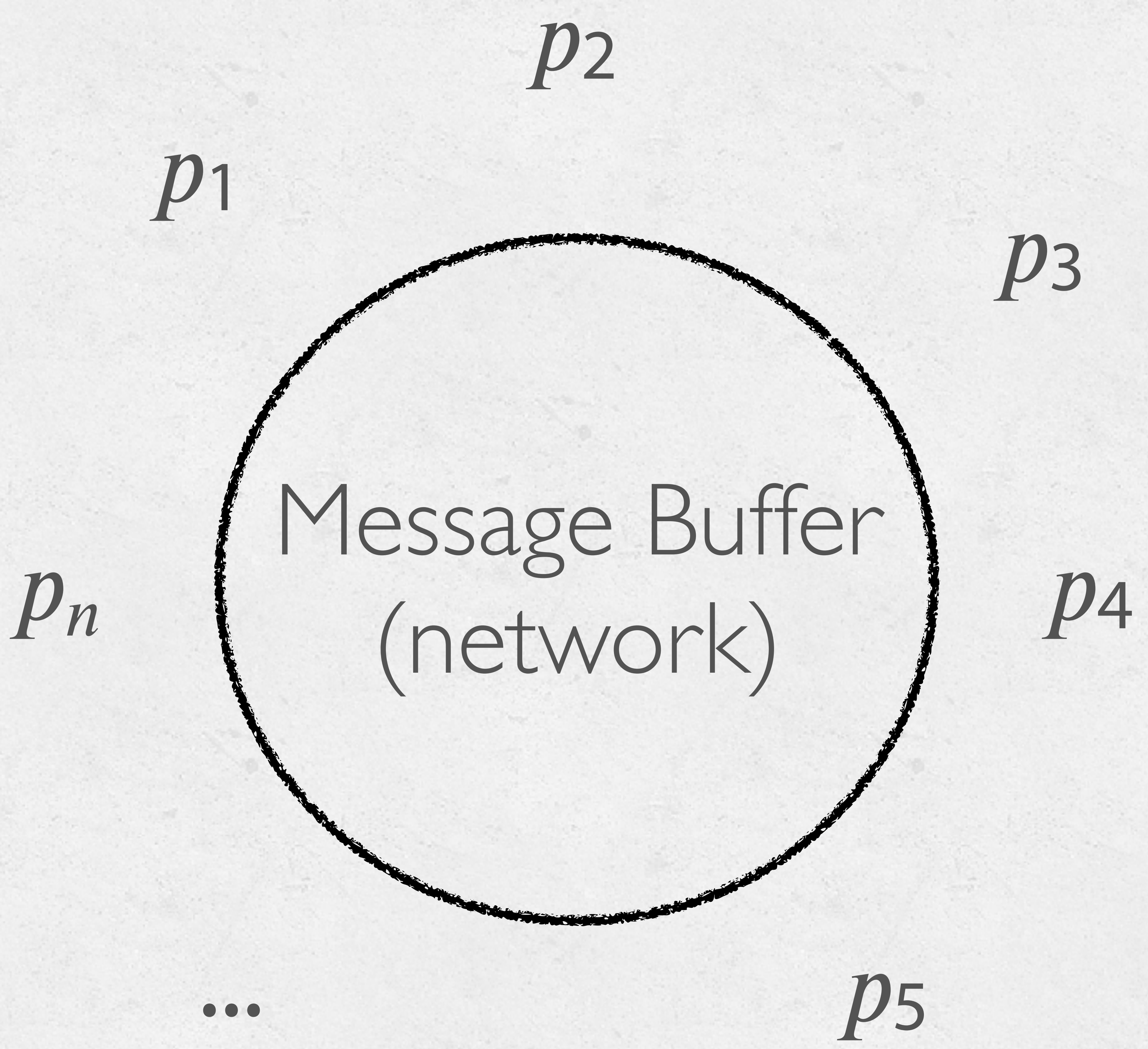
Theorem (FLP Impossibility Result): *In an asynchronous environment in which a single process can fail by crashing, there does not exist a protocol which solves binary consensus.*

Intuition

- In an asynchronous setting, failed processes are indistinguishable from slow processes.
- Waiting for failed processes will take forever.
- Not waiting for slow processes could violate safety.

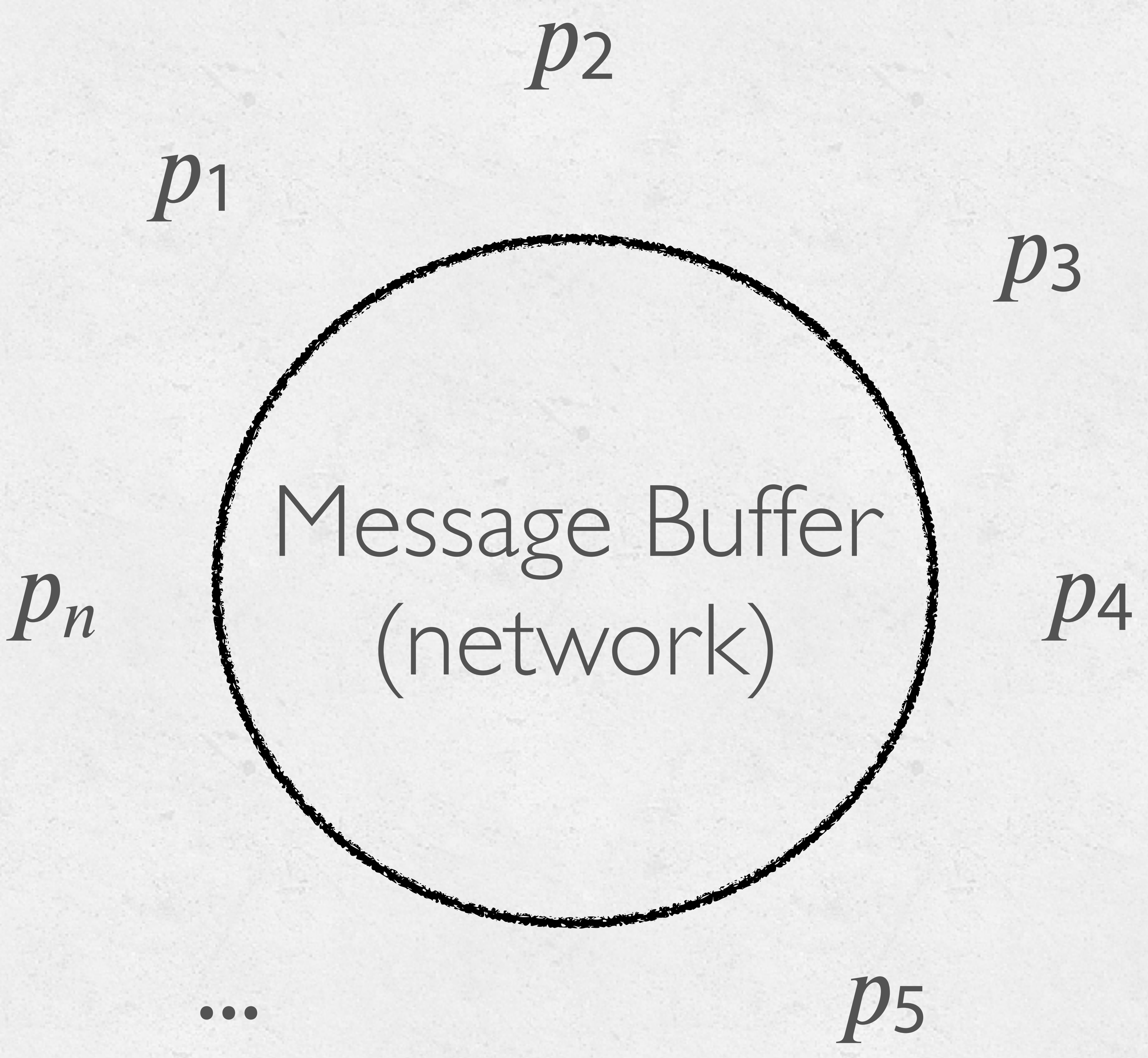
Computation Model

- Processes are deterministic I/O automata (just like in your labs; timers are just messages sent from process to itself).



Computation Model

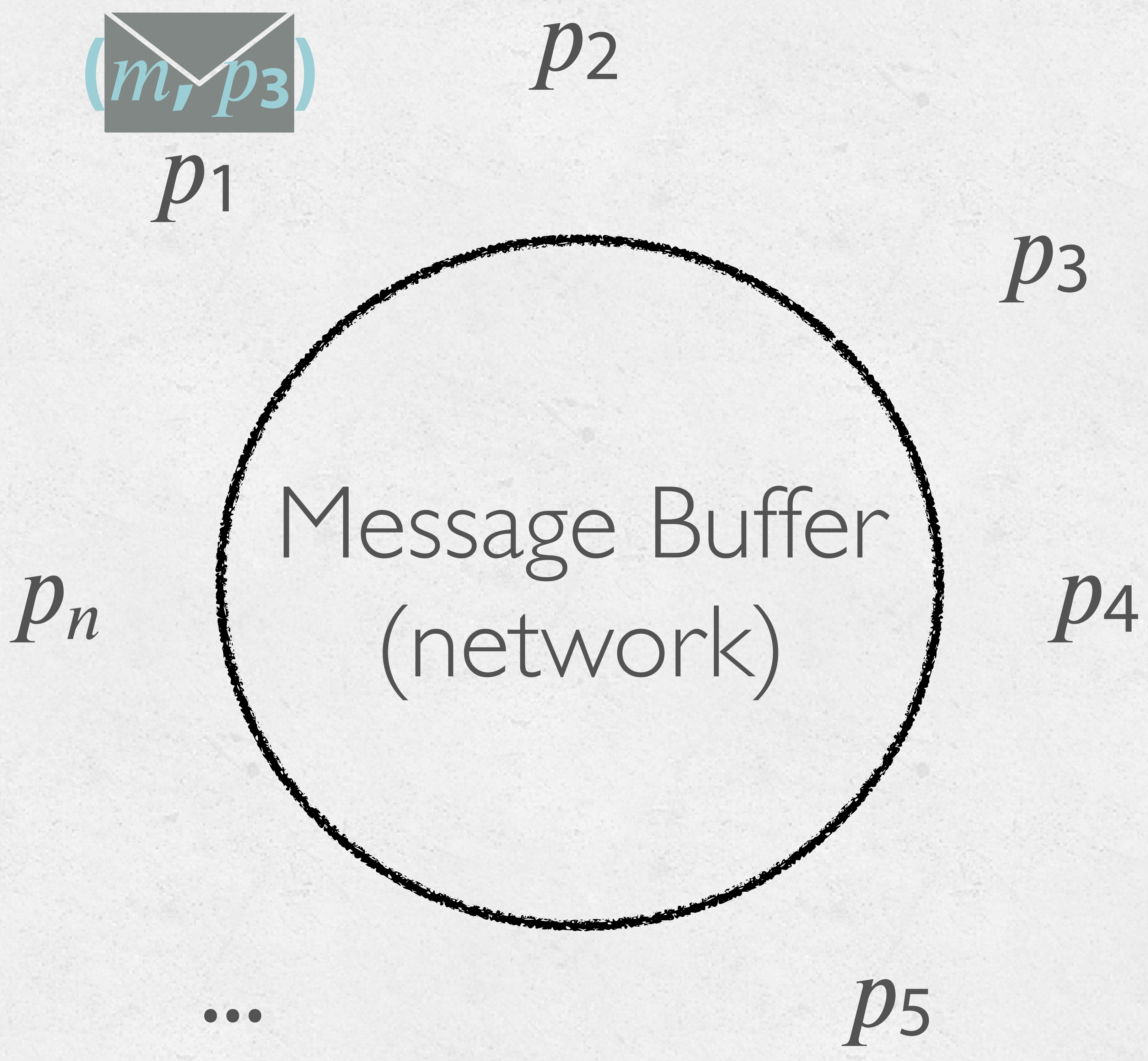
- Processes are deterministic I/O automata (just like in your labs; timers are just messages sent from process to itself).
- They send messages by adding to message buffer, a multi-set (i.e., messages aren't duplicated by network). Processes only send finitely-many messages in a single step.



Computation Model

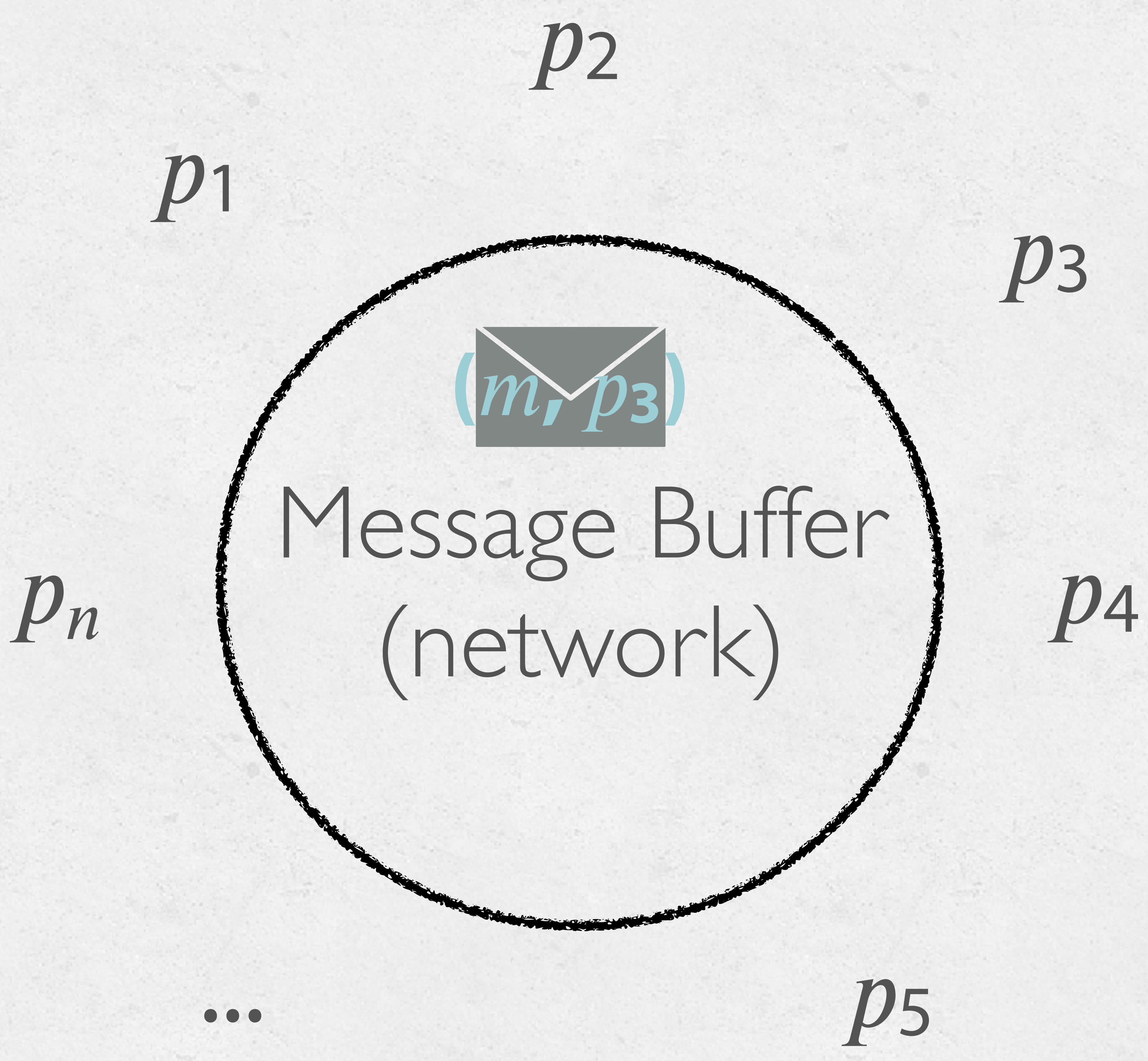
- Processes are deterministic I/O automata (just like in your labs; timers are just messages sent from process to itself).

- They send messages by adding to message buffer, a multi-set (i.e., messages aren't duplicated by network). Processes only send finitely-many messages in a single step.



COMPUTATION MODEL

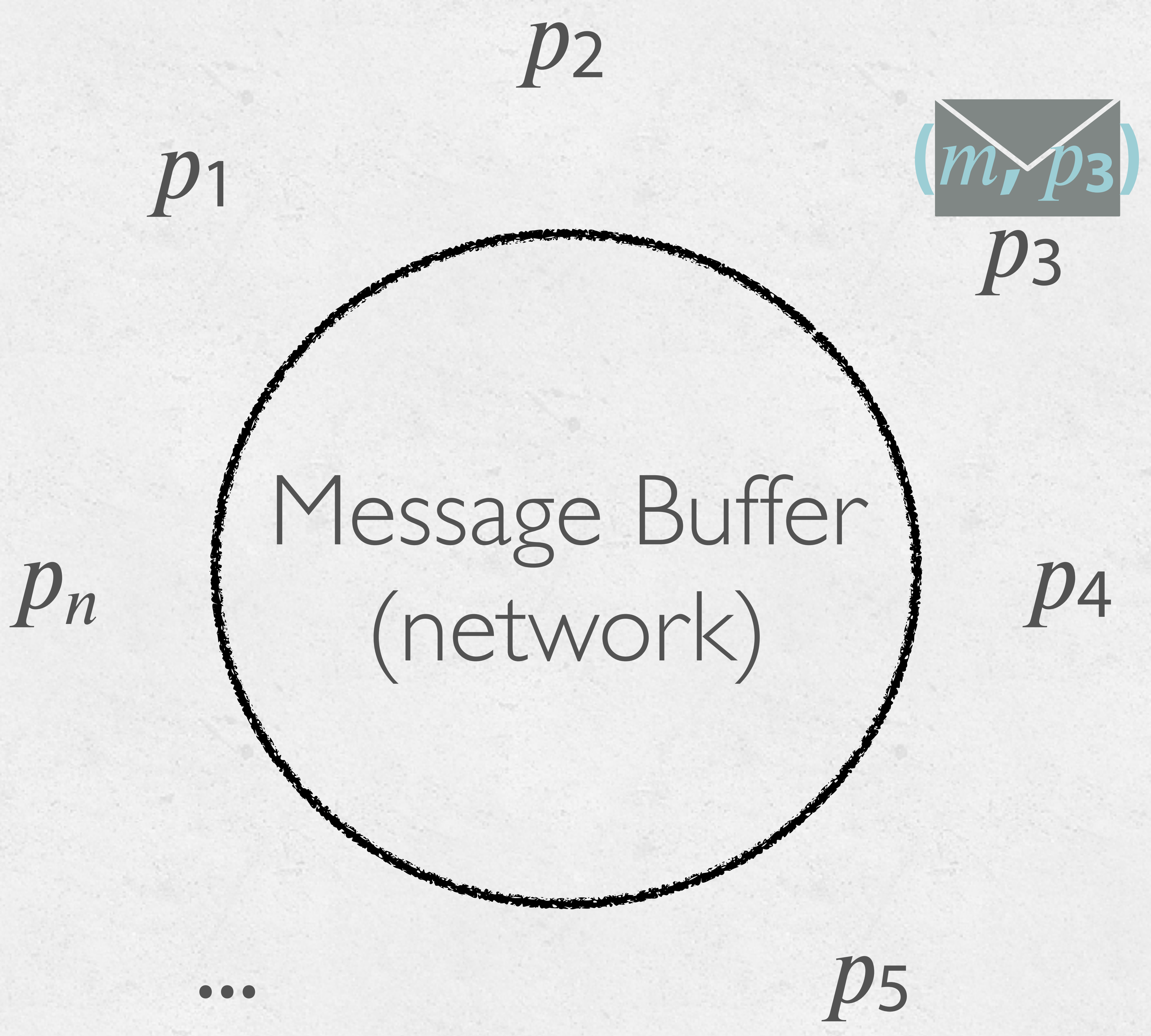
- Processes are deterministic I/O automata (just like in your labs; timers are just messages sent from process to itself).
- They send messages by adding to message buffer, a multi-set (i.e., messages aren't duplicated by network). Processes only send finitely-many messages in a single step.



Computation Model

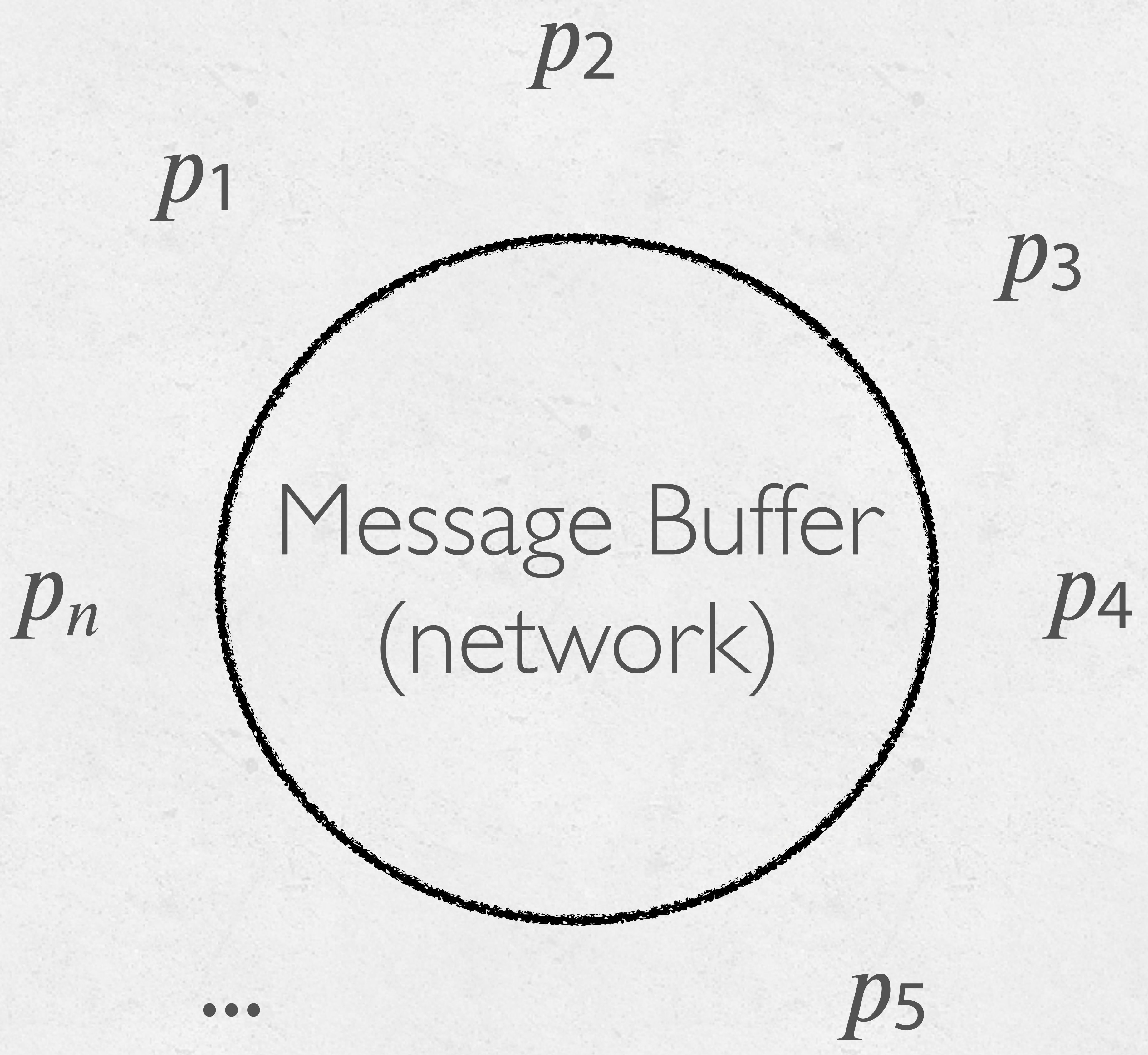
- Processes are deterministic I/O automata (just like in your labs; timers are just messages sent from process to itself).

- They send messages by adding to message buffer, a multi-set (i.e., messages aren't duplicated by network). Processes only send finitely-many messages in a single step.



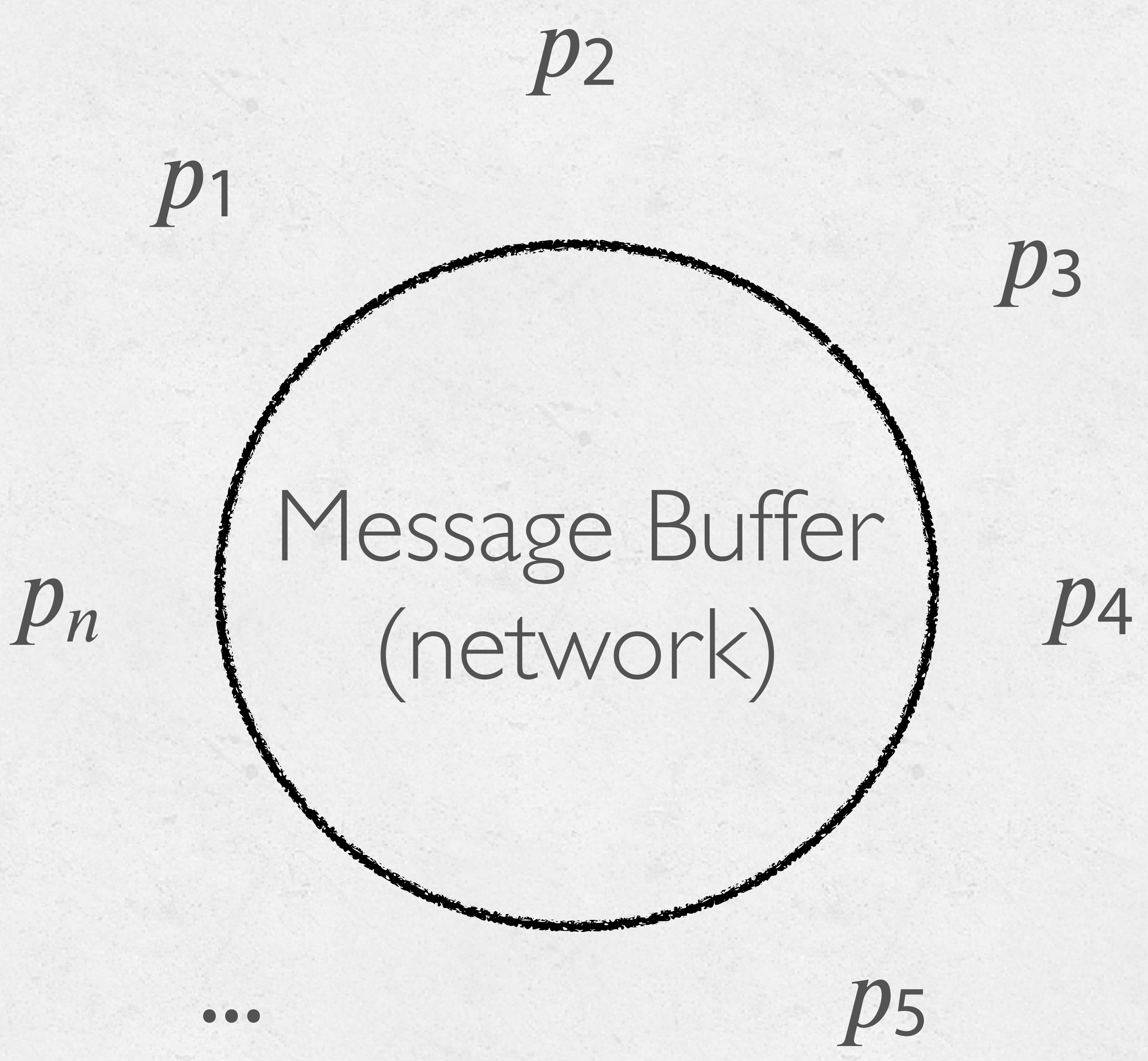
Computation Model

- Processes are deterministic I/O automata (just like in your labs; timers are just messages sent from process to itself).
- They send messages by adding to message buffer, a multi-set (i.e., messages aren't duplicated by network). Processes only send finitely-many messages in a single step.



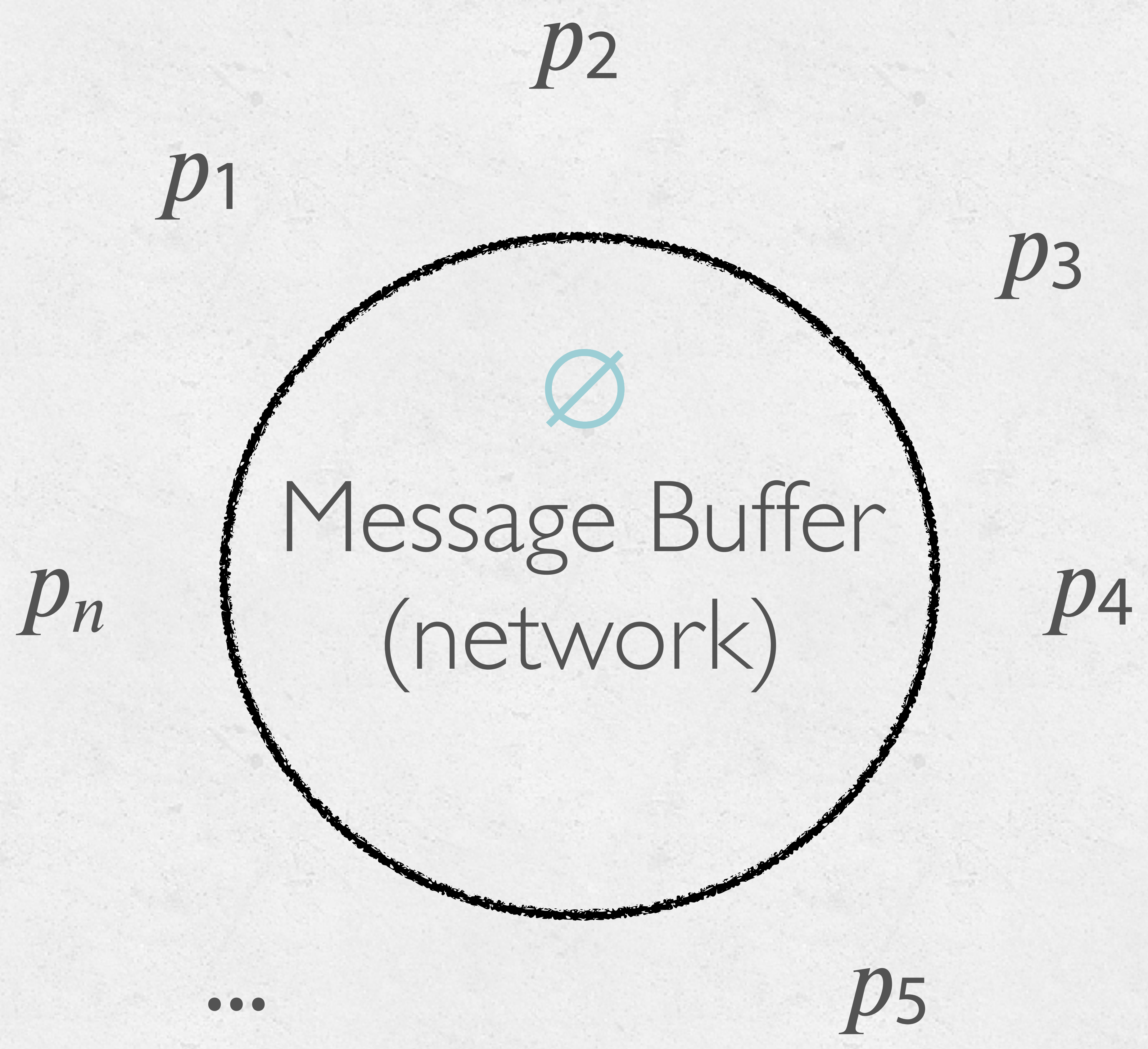
COMPUTATION MODEL

- Processes are deterministic I/O automata (just like in your labs; timers are just messages sent from process to itself).
- They send messages by adding to message buffer, a multi-set (i.e., messages aren't duplicated by network). Processes only send finitely-many messages in a single step.
- Special empty message, always deliverable to any process (even if there are messages for it in the network).



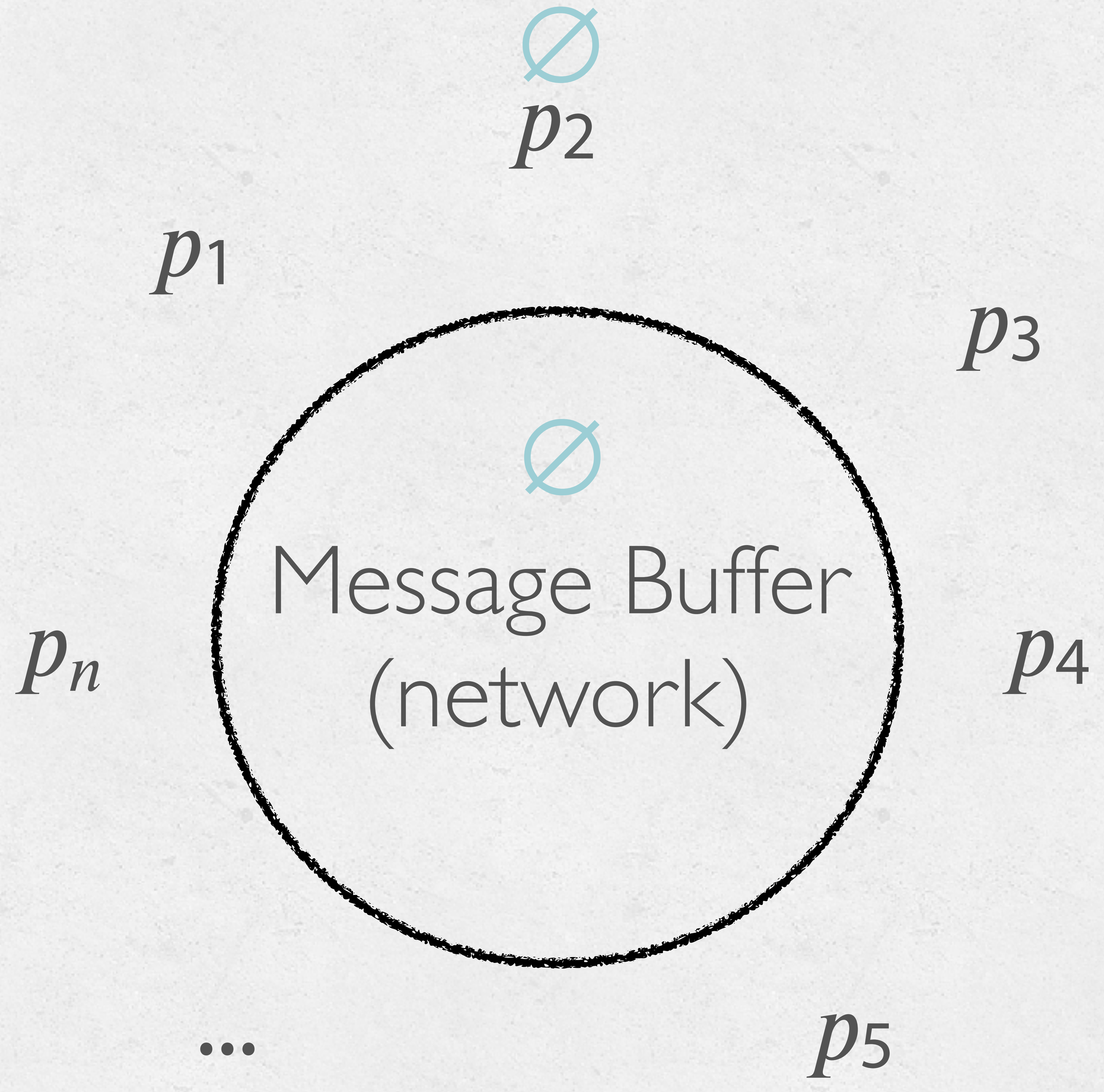
Computation Model

- Processes are deterministic I/O automata (just like in your labs; timers are just messages sent from process to itself).
- They send messages by adding to message buffer, a multi-set (i.e., messages aren't duplicated by network). Processes only send finitely-many messages in a single step.
- Special empty message, always deliverable to any process (even if there are messages for it in the network).



COMPUTATION MODEL

- Processes are deterministic I/O automata (just like in your labs; timers are just messages sent from process to itself).
- They send messages by adding to message buffer, a multi-set (i.e., messages aren't duplicated by network). Processes only send finitely-many messages in a single step.
- Special empty message, always deliverable to any process (even if there are messages for it in the network).

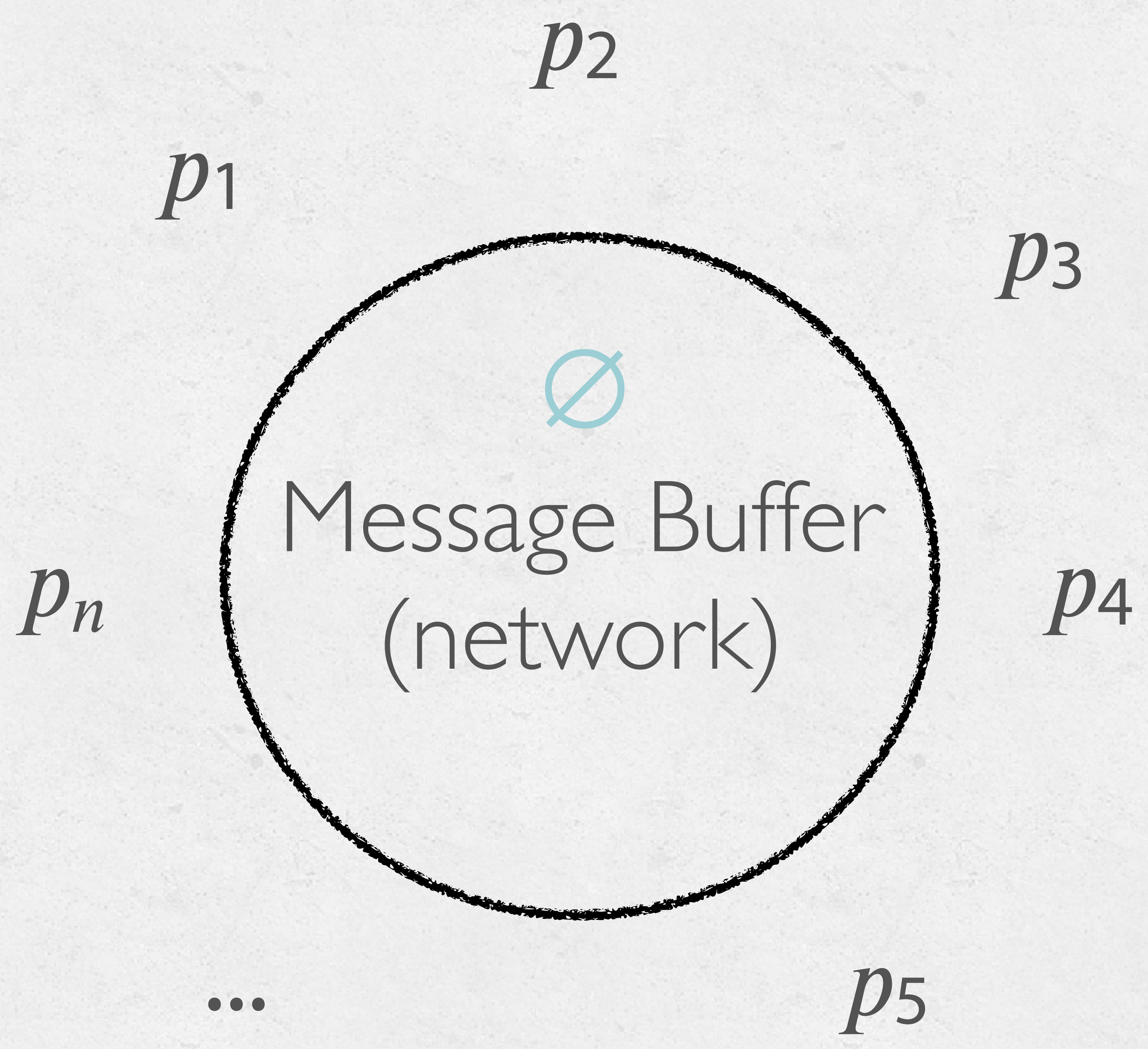


Computation Model

- Processes are deterministic I/O automata (just like in your labs; timers are just messages sent from process to itself).

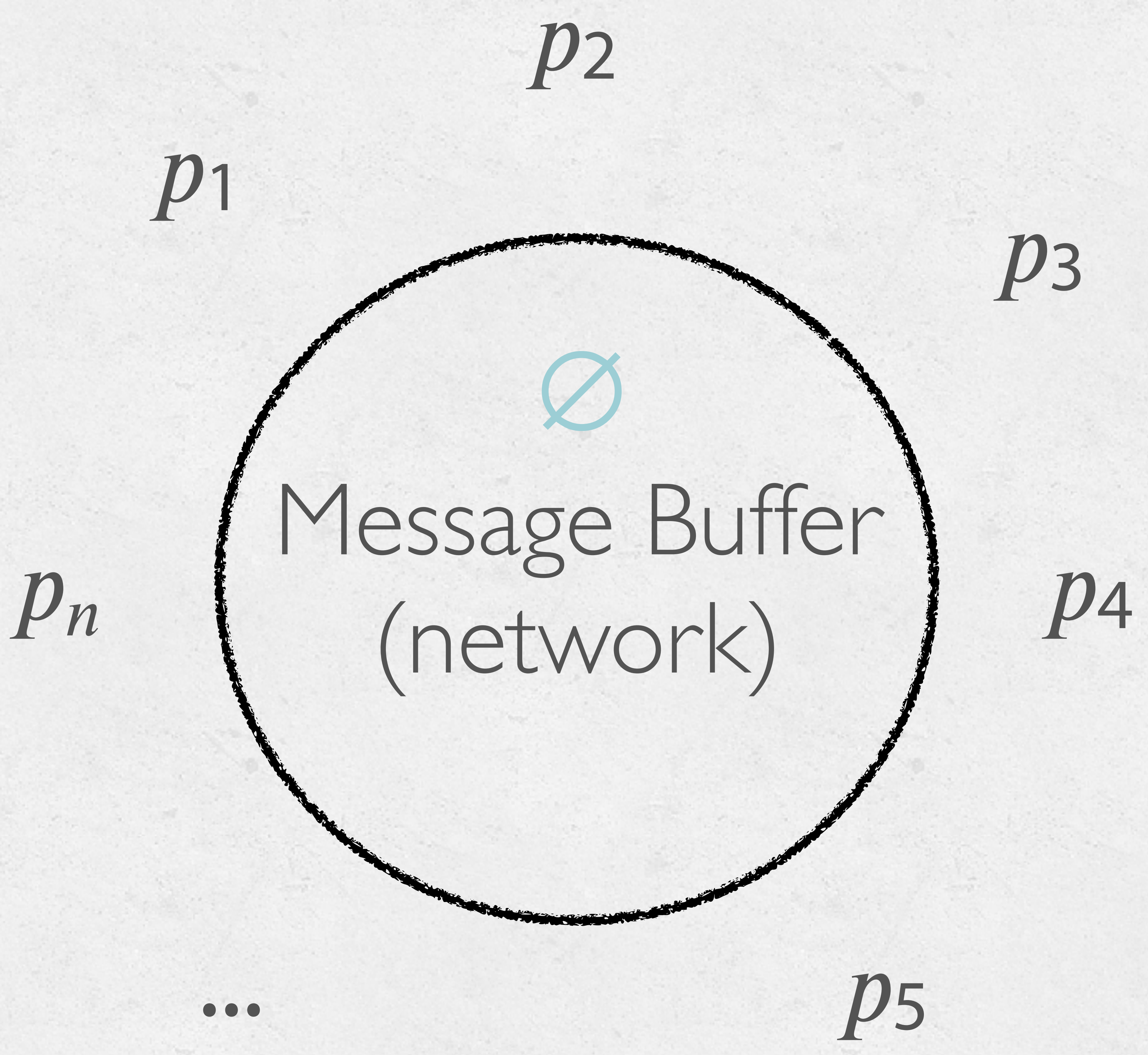
- They send messages by adding to message buffer, a multi-set (i.e., messages aren't duplicated by network). Processes only send finitely-many messages in a single step.

- Special empty message, always deliverable to any process (even if there are messages for it in the network).



Computation Model

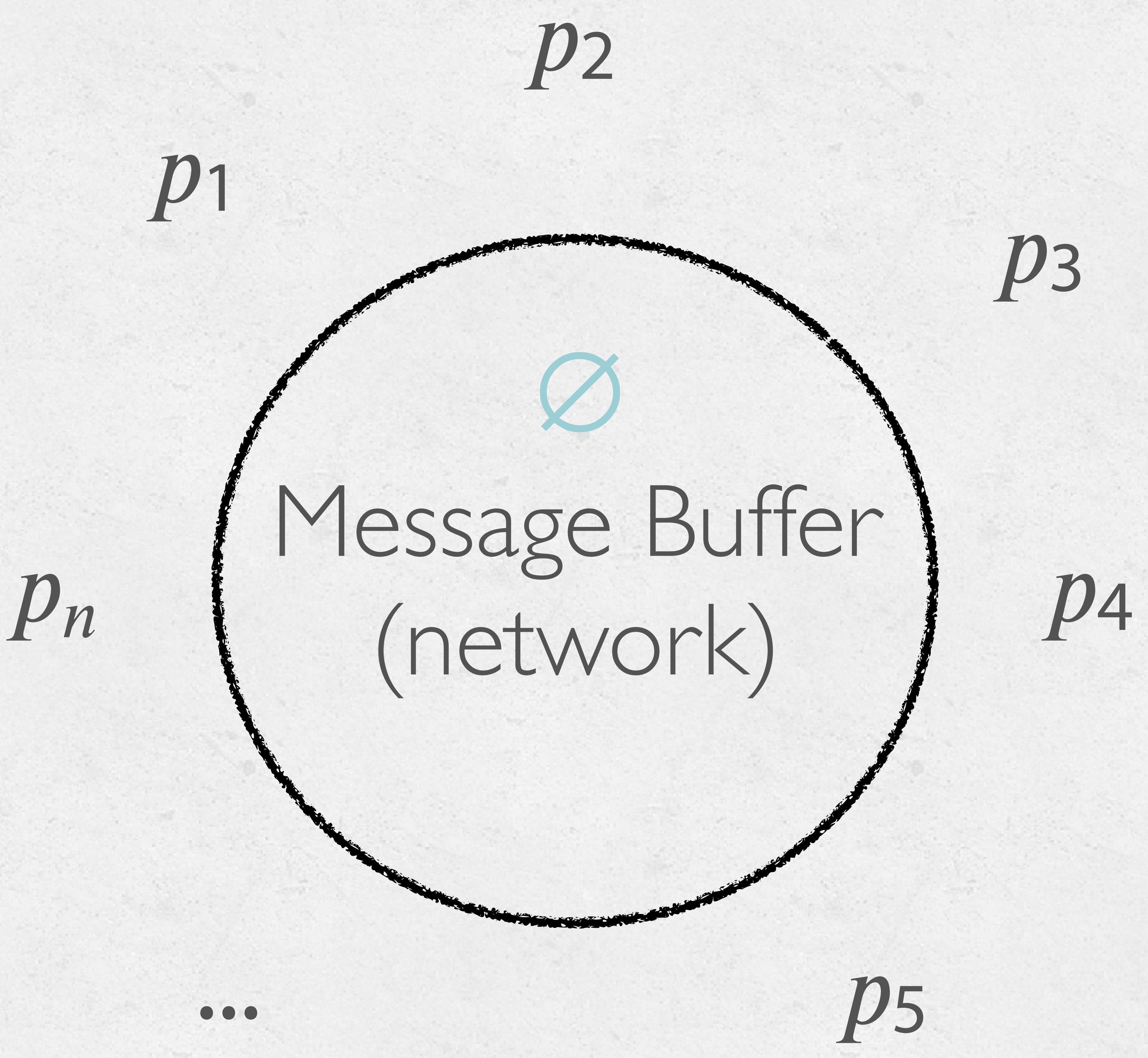
- Processes are deterministic I/O automata (just like in your labs; timers are just messages sent from process to itself).
- They send messages by adding to message buffer, a multi-set (i.e., messages aren't duplicated by network). Processes only send finitely-many messages in a single step.
- Special empty message, always deliverable to any process (even if there are messages for it in the network).
- Any message sent to a non-faulty processes is eventually received. (Stronger assumption than usual!)



Computation Model

- Processes are deterministic I/O automata (just like in your labs; timers are just messages sent from process to itself).
- They send messages by adding to message buffer, a multi-set (i.e., messages aren't duplicated by network). Processes only send finitely-many messages
- Special e able to any process (e it in the network).
- Any message sent to a non-faulty processes is eventually received. (Stronger assumption than usual!)

Makes the impossibility result is stronger!



Configurations

A **configuration** (usually denoted C) consists of the states of all processes and the state of the message buffer.

An **event** is the delivery of a single message (or $\varnothing$) to a process. An event is **applicable** to C if it is a $\varnothing$ or a message in C's message buffer.

A configuration C' is **reachable** from C if there is a (possibly empty) sequence of applicable events starting from C that results in C'.

Configuration C is **decided** if at least one process has decided in C.

Runs

A **run** is an infinite sequence of events starting from an initial configuration.

A process is **non-faulty** in a run if it takes infinitely many steps. It is faulty otherwise.

A run is **admissible** if at most one process is faulty and every message sent to a non-faulty process is eventually delivered.

In other words, the FLP theorem states that **any protocol** for binary consensus either doesn't satisfy safety or allows for an admissible run in which no value is ever decided (i.e., that it doesn't satisfy termination, the liveness property).

From now on, we'll consider a **safe** and **live** binary consensus protocol and show a contradiction.

Valency

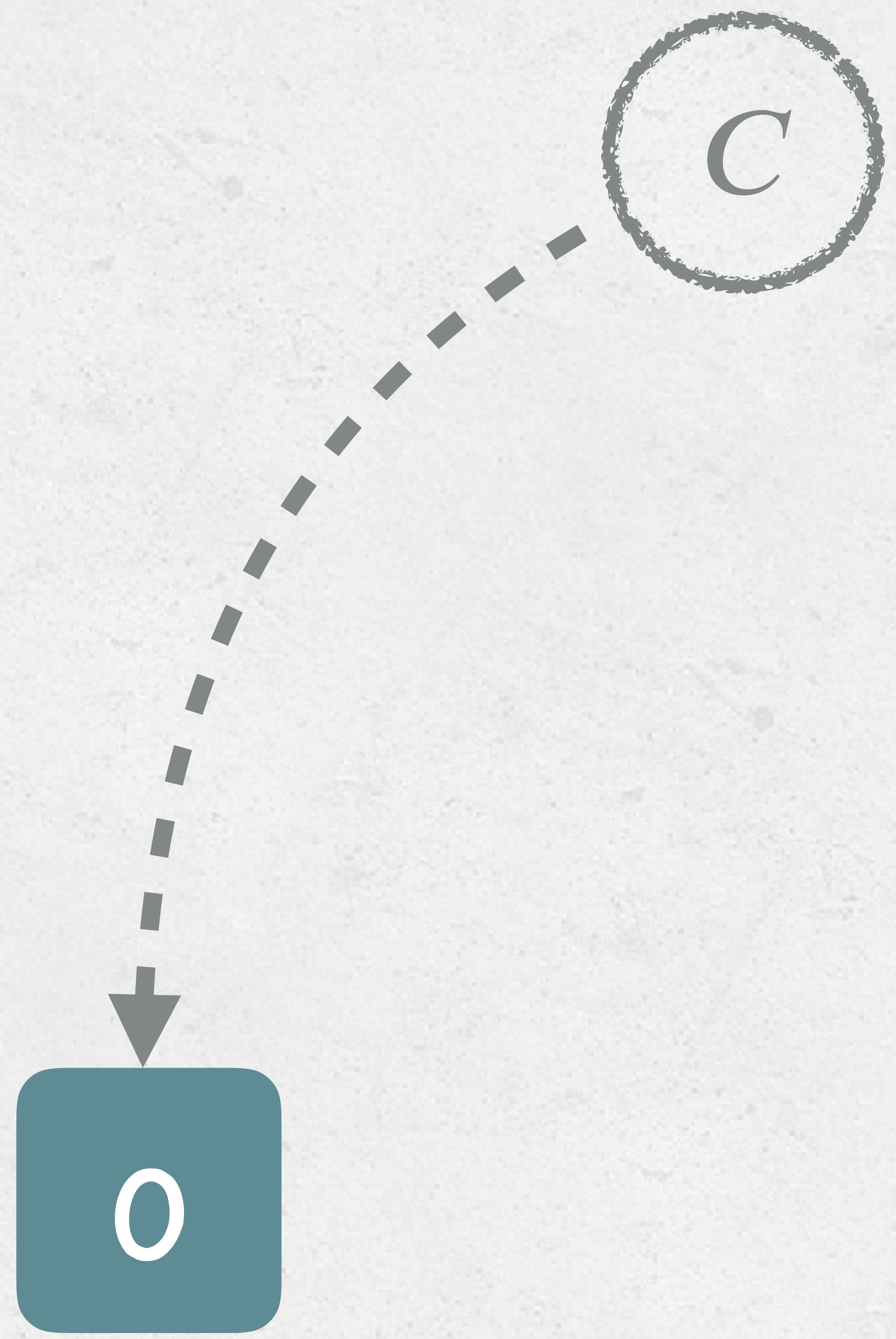
By assumption of safety, no configuration has processes deciding different values.

C is **0-valent** if there are decided configurations reachable from C that decide 0, but none that decide 1.

1-valency is defined in the analogous way.

C is **univalent** if it is 0-valent or 1-valent.

C is **bivalent** if both 0-deciding and 1-deciding are reachable from C.



Valency

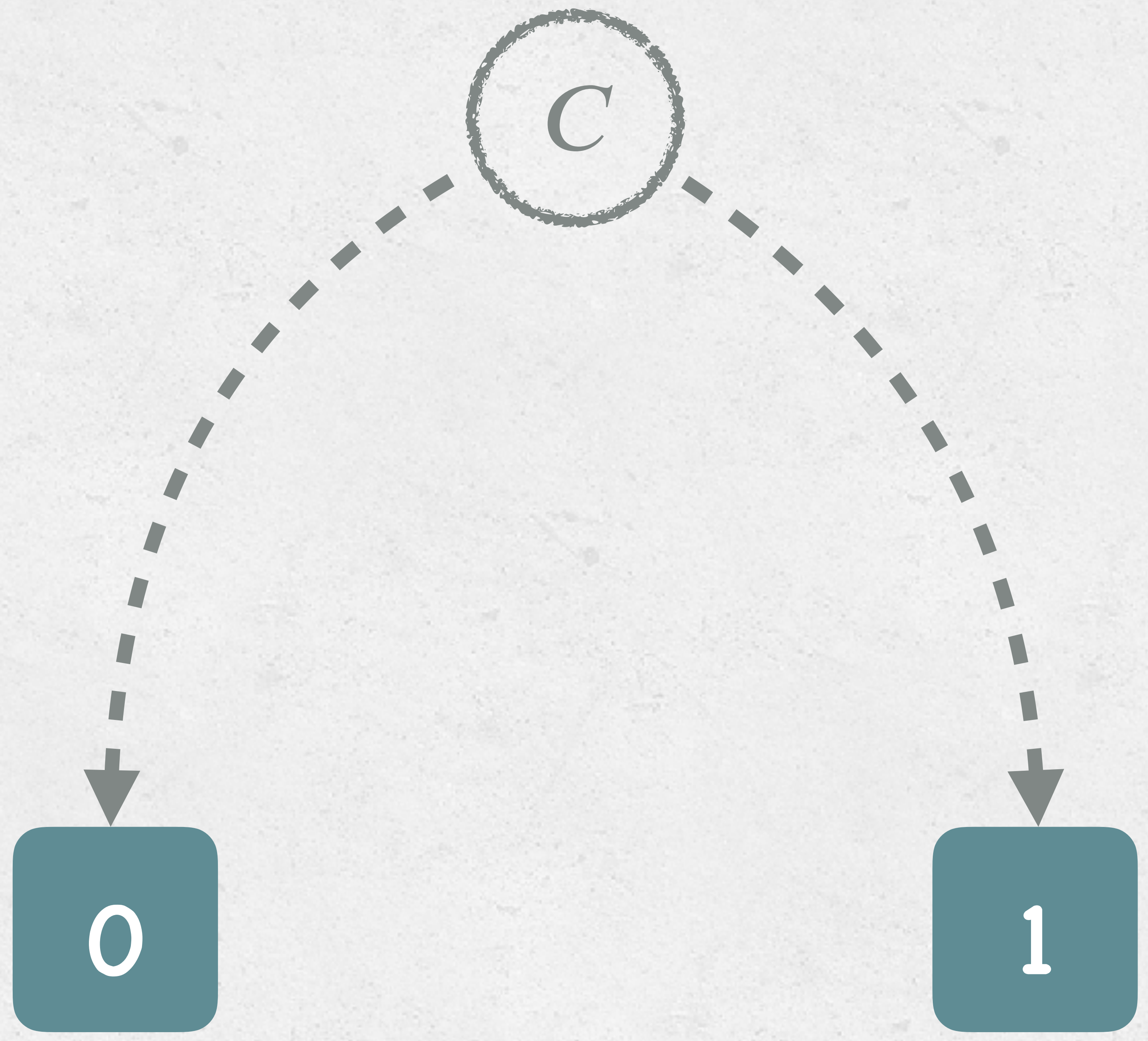
By assumption of safety, no configuration has processes deciding different values.

C is **0-valent** if there are decided configurations reachable from C that decide 0, but none that decide 1.

1-valency is defined in the analogous way.

C is **univalent** if it is 0-valent or 1-valent.

C is **bivalent** if both 0-deciding and 1-deciding are reachable from C.



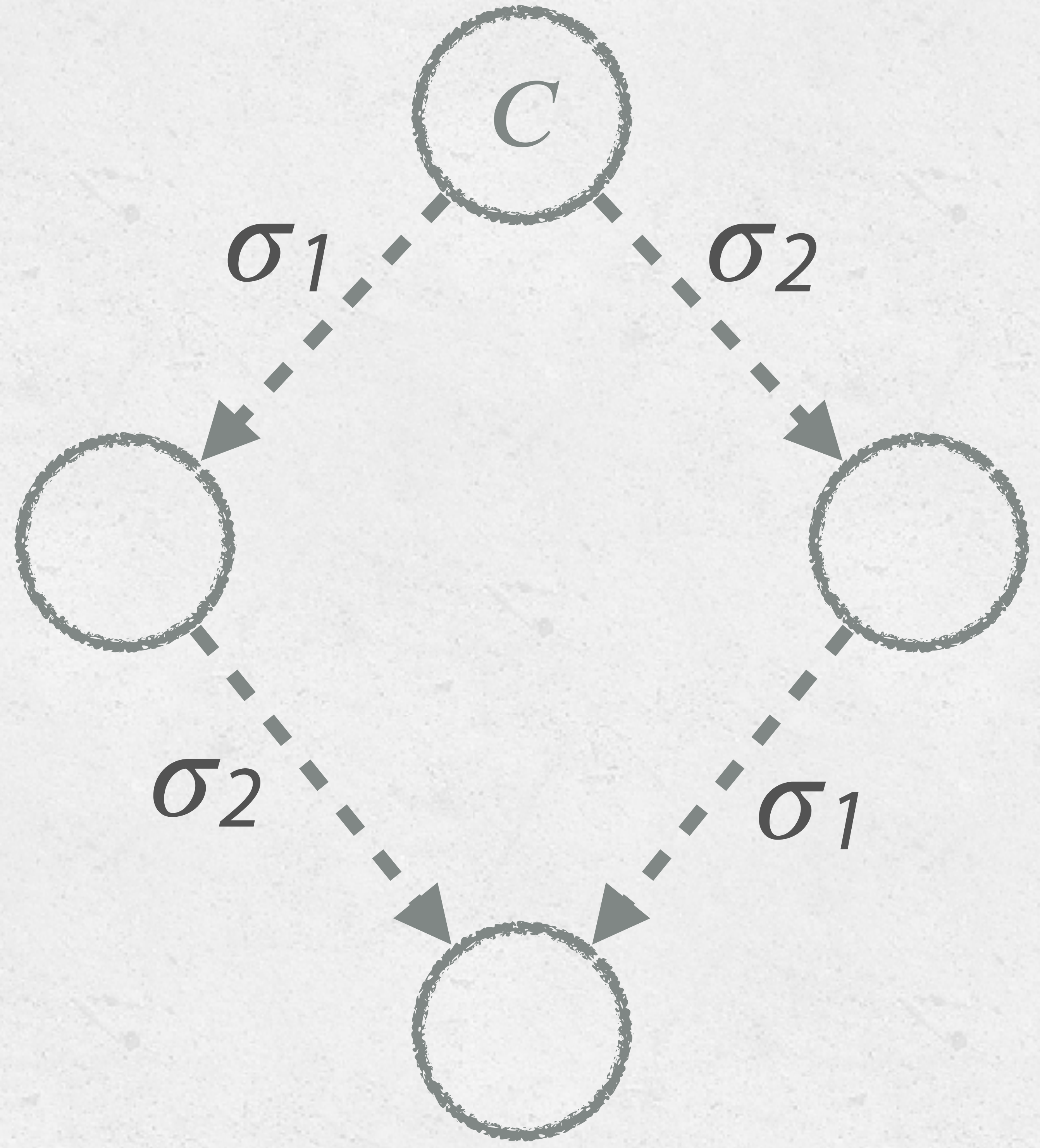
Observation: bivalent configurations are not themselves decided.

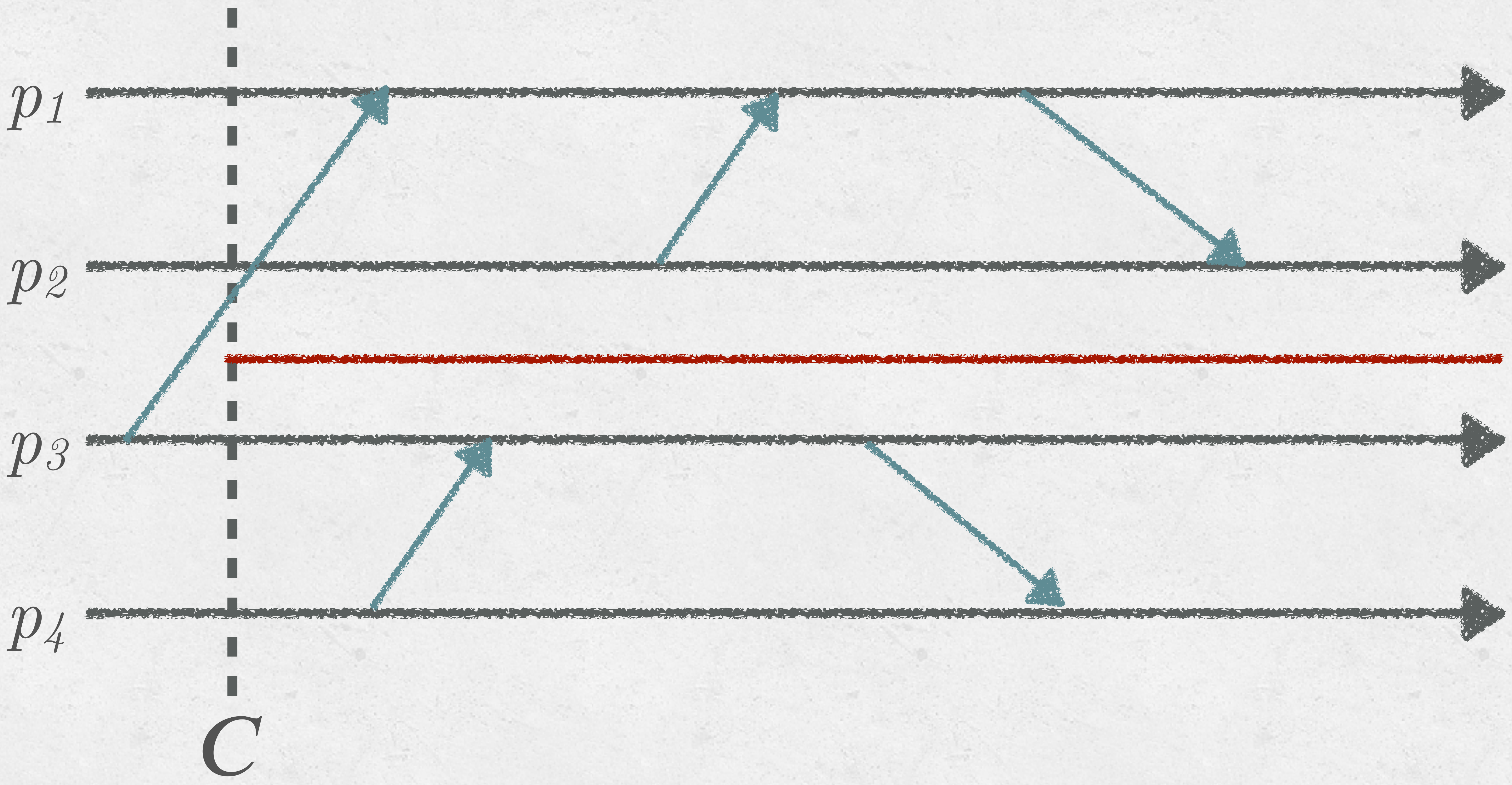
Observation: 1-valent and bivalent configurations are not reachable from 0-valent configurations.

0-valent and bivalent configurations are not reachable from 1-valent configurations.

Commutative Events

Lemma 1: *If two sequences of events, σ_1 and σ_2, are taken by* ***disjoint*** *sets of processes from configuration C, then $\sigma_1(\sigma_2(C)) = \sigma_2(\sigma_1(C))$.*





Bivalent Initial Configurations

Lemma 2: *There exists a bivalent initial configuration.*

Bivalent Initial Configurations

Lemma 2: *There exists a bivalent initial configuration.*

$$\boxed{\begin{array}{c} 0 \rightarrow p_1 \\ 0 \rightarrow p_2 \\ 0 \rightarrow p_3 \\ \dots \\ 0 \rightarrow p_n \end{array}}$$

Bivalent Initial Configurations

Lemma 2: *There exists a bivalent initial configuration.*

$0 \rightarrow p_1$

$0 \rightarrow p_2$

$0 \rightarrow p_3$

$\ldots$

$0 \rightarrow p_n$

0-valent!

Bivalent Initial Configurations

Lemma 2: *There exists a bivalent initial configuration.*

$0 \rightarrow p_1$	$1 \rightarrow p_1$
$0 \rightarrow p_2$	$1 \rightarrow p_2$
$0 \rightarrow p_3$	$1 \rightarrow p_3$
…	…
$0 \rightarrow p_n$	$1 \rightarrow p_n$

0-valent!

Bivalent Initial Configurations

Lemma 2: *There exists a bivalent initial configuration.*

$0 \rightarrow p_1$

$0 \rightarrow p_2$

$0 \rightarrow p_3$

…

$0 \rightarrow p_n$

0-valent!

$1 \rightarrow p_1$

$1 \rightarrow p_2$

$1 \rightarrow p_3$

…

$1 \rightarrow p_n$

1-valent!

Bivalent Initial Configurations

Lemma 2: *There exists a bivalent initial configuration.*

$0 \rightarrow p_1$	$1 \rightarrow p_1$	$1 \rightarrow p_1$
$0 \rightarrow p_2$	$0 \rightarrow p_2$	$1 \rightarrow p_2$
$0 \rightarrow p_3$	$0 \rightarrow p_3$	$1 \rightarrow p_3$
…	…	…
$0 \rightarrow p_n$	$0 \rightarrow p_n$	$1 \rightarrow p_n$

0-valent!

1-valent!

BIVALENT INITIAL CONFIGURATIONS

Lemma 2: *There exists a bivalent initial configuration.*

$0 \rightarrow p_1$	$1 \rightarrow p_1$	$1 \rightarrow p_1$	$1 \rightarrow p_1$
$0 \rightarrow p_2$	$0 \rightarrow p_2$	$1 \rightarrow p_2$	$1 \rightarrow p_2$
$0 \rightarrow p_3$	$0 \rightarrow p_3$	$0 \rightarrow p_3$	$1 \rightarrow p_3$
…	…	…	…
$0 \rightarrow p_n$	$0 \rightarrow p_n$	$0 \rightarrow p_n$	$1 \rightarrow p_n$

0-valent!

1-valent!

Bivalent Initial Configurations

Lemma 2: *There exists a bivalent initial configuration.*

$0 \rightarrow p_1$	$1 \rightarrow p_1$	$1 \rightarrow p_1$		$1 \rightarrow p_1$
$0 \rightarrow p_2$	$0 \rightarrow p_2$	$1 \rightarrow p_2$		$1 \rightarrow p_2$
$0 \rightarrow p_3$	$0 \rightarrow p_3$	$0 \rightarrow p_3$	…	$1 \rightarrow p_3$
…	…	…		…
$0 \rightarrow p_n$	$0 \rightarrow p_n$	$0 \rightarrow p_n$		$1 \rightarrow p_n$
0-valent!				1-valent!

Bivalent Initial Configurations

Lemma 2: *There exists a bivalent initial configuration.*

There must be 0-valent C_0 and 1-valent C_1 that differ only in the input value of a single process, p.

$1 \longrightarrow p \Rightarrow$ 1 is decided

$0 \longrightarrow p \Rightarrow$ 0 is decided

Bivalent Initial Configurations

Lemma 2: *There exists a bivalent initial configuration.*

There must be 0-valent C_0 and 1-valent C_1 that differ only in the input value of a single process, p.

What if p crashes at the beginning?

$1 \longrightarrow p \Rightarrow 1$ is decided

$0 \longrightarrow p \Rightarrow 0$ is decided

Bivalent Initial Configurations

Lemma 2: *There exists a bivalent initial configuration.*

There must be 0-valent C_0 and 1-valent C_1 that differ only in the input value of a single process, p.

What if p crashes at the beginning?

These two configurations are indistinguishable to the rest of the processes.

$1 \longrightarrow p \Rightarrow 1$ is decided

$0 \longrightarrow p \Rightarrow 0$ is decided

Delaying Events

Lemma 3 (The Delay Lemma): *For every bivalent configuration, C, and every event applicable to C, e, there exists a sequence of applicable events σ such that $C' = e(\sigma(C))$ is bivalent.*

Proving the Main Theorem

Proving the Main Theorem

Constructing the non-terminating execution:

Proving the Main Theorem

Constructing the non-terminating execution:

1: Let C be a bivalent initial configuration (Lemma 2).

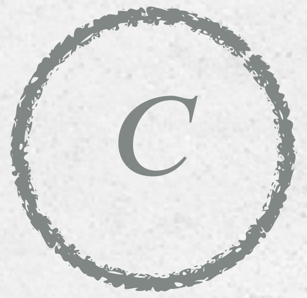
PROVING THE MAIN THEOREM

C

Constructing the non-terminating execution:

1: Let C be a bivalent initial configuration (Lemma 2).

Proving the Main Theorem



Constructing the non-terminating execution:

1: Let C be a bivalent initial configuration (Lemma 2).

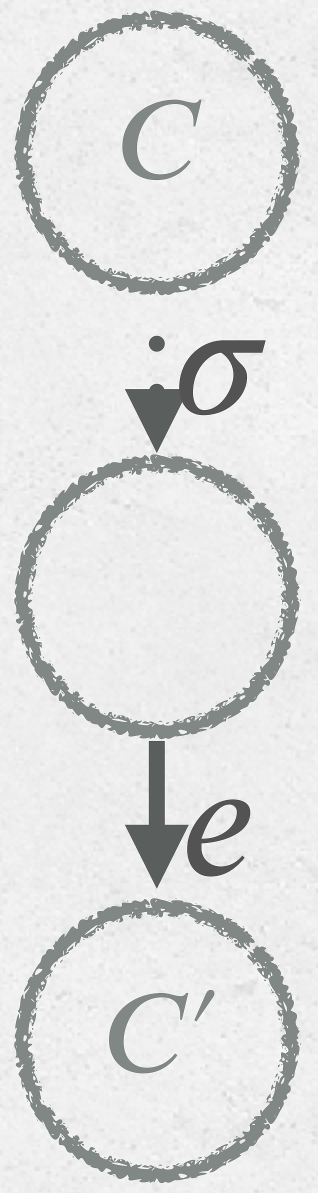
2: For the process which least recently took a step, take the oldest message left in the network for it ($\varnothing$ if none exists), e. By Lemma 3, we first take a sequence of steps σ and *then* deliver e and remain in a bivalent configuration.

PROVING THE MAIN THEOREM

Constructing the non-terminating execution:

1: Let C be a bivalent initial configuration (Lemma 2).

2: For the process which least recently took a step, take the oldest message left in the network for it ($\varnothing$ if none exists), e. By Lemma 3, we first take a sequence of steps σ and *then* deliver e and remain in a bivalent configuration.



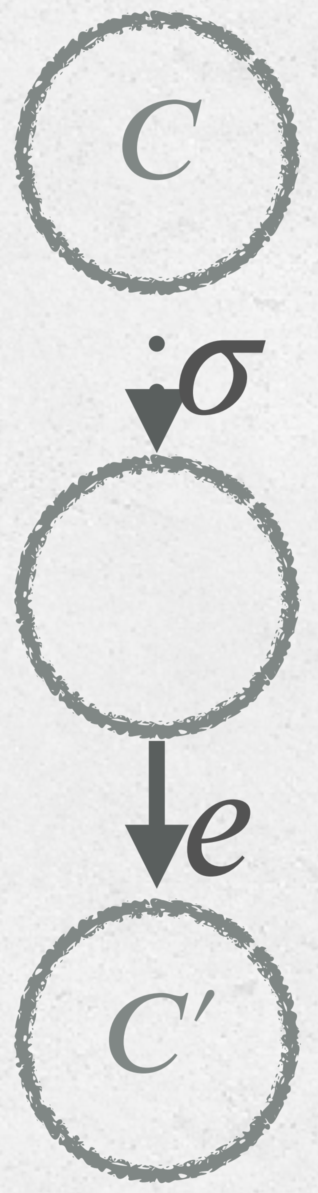
Proving the Main Theorem

Constructing the non-terminating execution:

1: Let C be a bivalent initial configuration (Lemma 2).

2: For the process which least recently took a step, take the oldest message left in the network for it ($\varnothing$ if none exists), e. By Lemma 3, we first take a sequence of steps σ and *then* deliver e and remain in a bivalent configuration.

3: Go to 2.



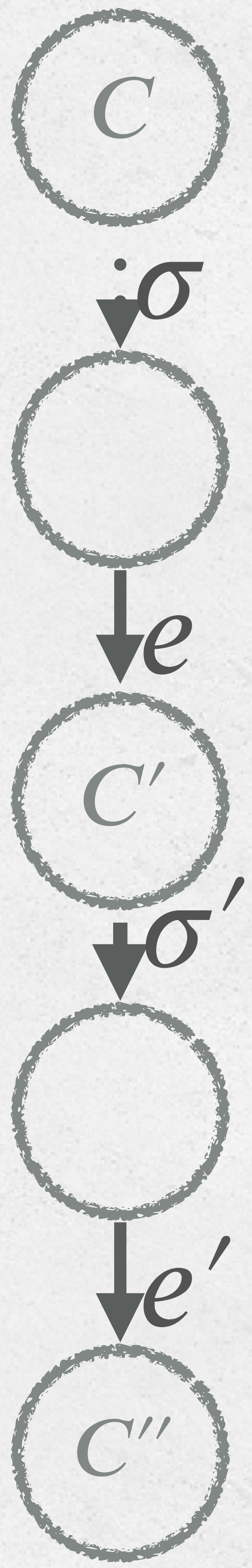
PROVING THE MAIN THEOREM

Constructing the non-terminating execution:

1: Let C be a bivalent initial configuration (Lemma 2).

2: For the process which least recently took a step, take the oldest message left in the network for it ($\varnothing$ if none exists), e. By Lemma 3, we first take a sequence of steps σ and *then* deliver e and remain in a bivalent configuration.

3: Go to 2.



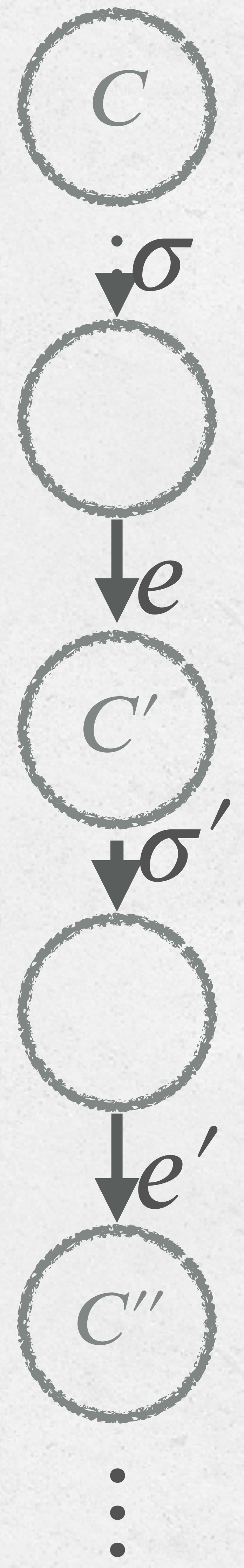
Proving the Main Theorem

Constructing the non-terminating execution:

1: Let C be a bivalent initial configuration (Lemma 2).

2: For the process which least recently took a step, take the oldest message left in the network for it ($\varnothing$ if none exists), e. By Lemma 3, we first take a sequence of steps σ and *then* deliver e and remain in a bivalent configuration.

3: Go to 2.



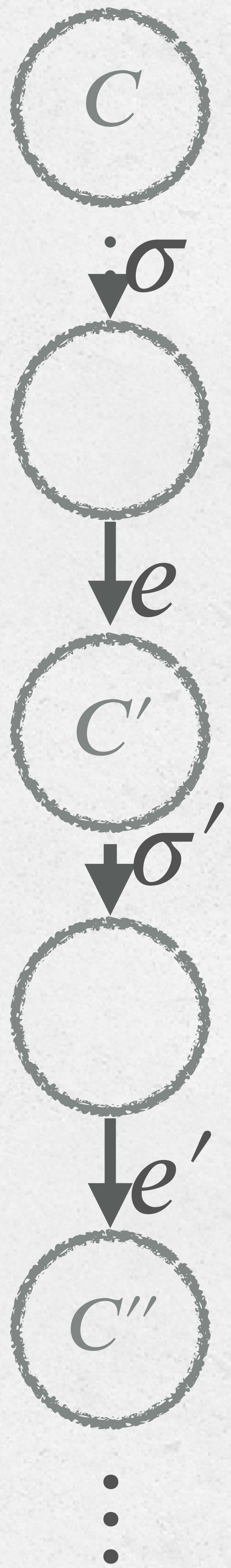
Proving the Main Theorem

Constructing the non-terminating execution:

1: Let C be a bivalent initial configuration (Lemma 2).

2: For the process which least recently took a step, take the oldest message left in the network for it ($\varnothing$ if none exists), e. By Lemma 3, we first take a sequence of steps σ and *then* deliver e and remain in a bivalent configuration.

3: Go to 2.



Every process takes infinitely many steps (i.e., no process is faulty). Every message sent is eventually delivered. This is an admissible execution.

We take infinitely many steps, and no process decides! The protocol fails to meet the termination property of the spec.

Proving the Delay Lemma

Consider a bivalent configuration, C, and

an applicable event, e.

Proving the Delay Lemma

Consider a bivalent configuration, C, and
an applicable event, e.

If $e(C)$ is bivalent, then we're done.

Proving the Delay Lemma

Consider a bivalent configuration, C, and an applicable event, e.

If $e(C)$ is bivalent, then we're done.

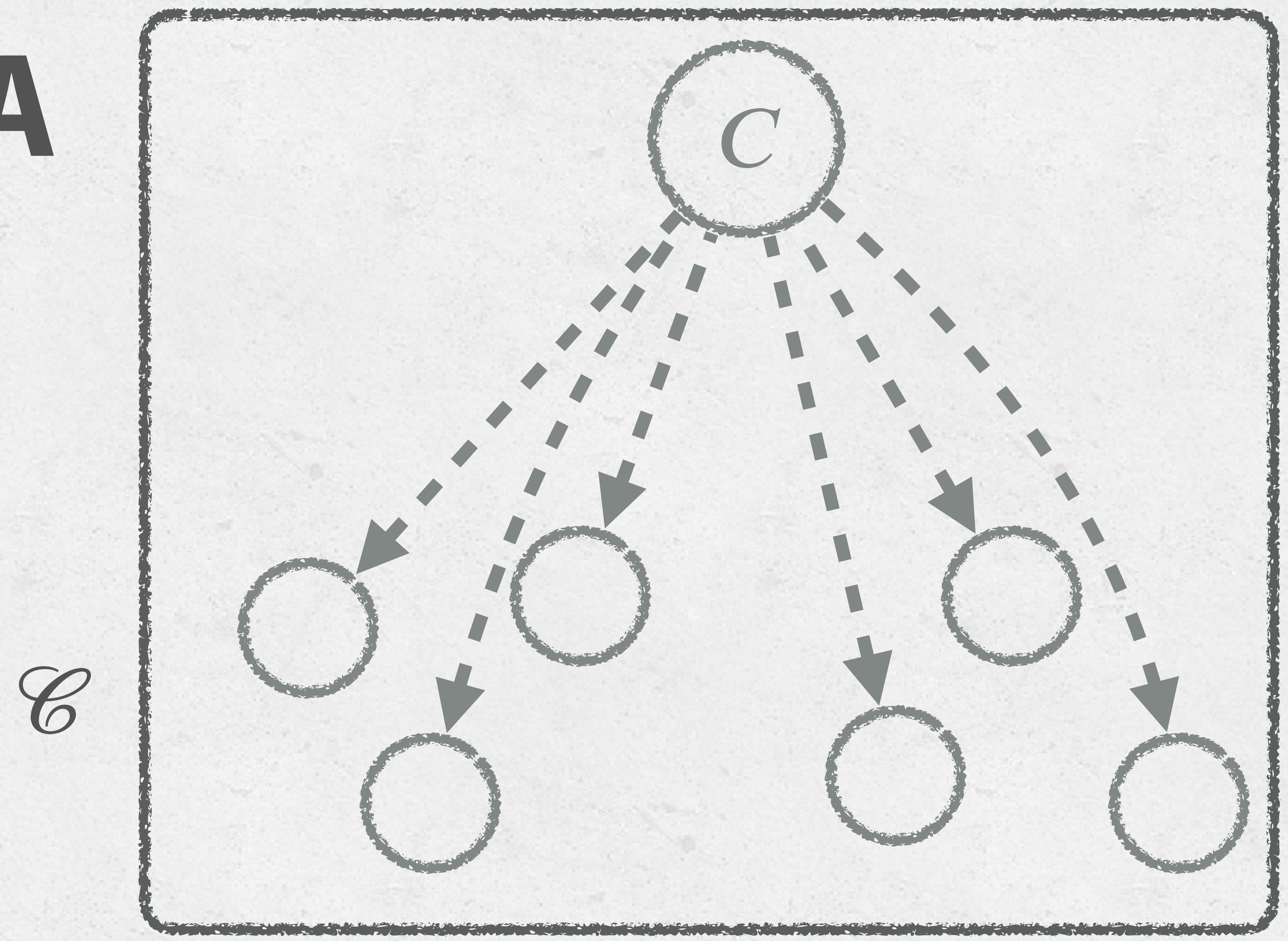
Otherwise, let $\mathscr{C}$ be the set of events reachable from C *without* applying e and $\mathscr{D}$ be $e(\mathscr{C}) = \{\, e(C) : C \in \mathscr{C} \,\}$ (i.e., the set of all configurations reachable from C where e was the last event taken).

Proving the Delay Lemma

Consider a bivalent configuration, C, and an applicable event, e.

If $e(C)$ is bivalent, then we're done.

Otherwise, let $\mathscr{C}$ be the set of events reachable from C *without* applying e and $\mathscr{D}$ be $e(\mathscr{C}) = \{\, e(C) : C \in \mathscr{C} \,\}$ (i.e., the set of all configurations reachable from C where e was the last event taken).

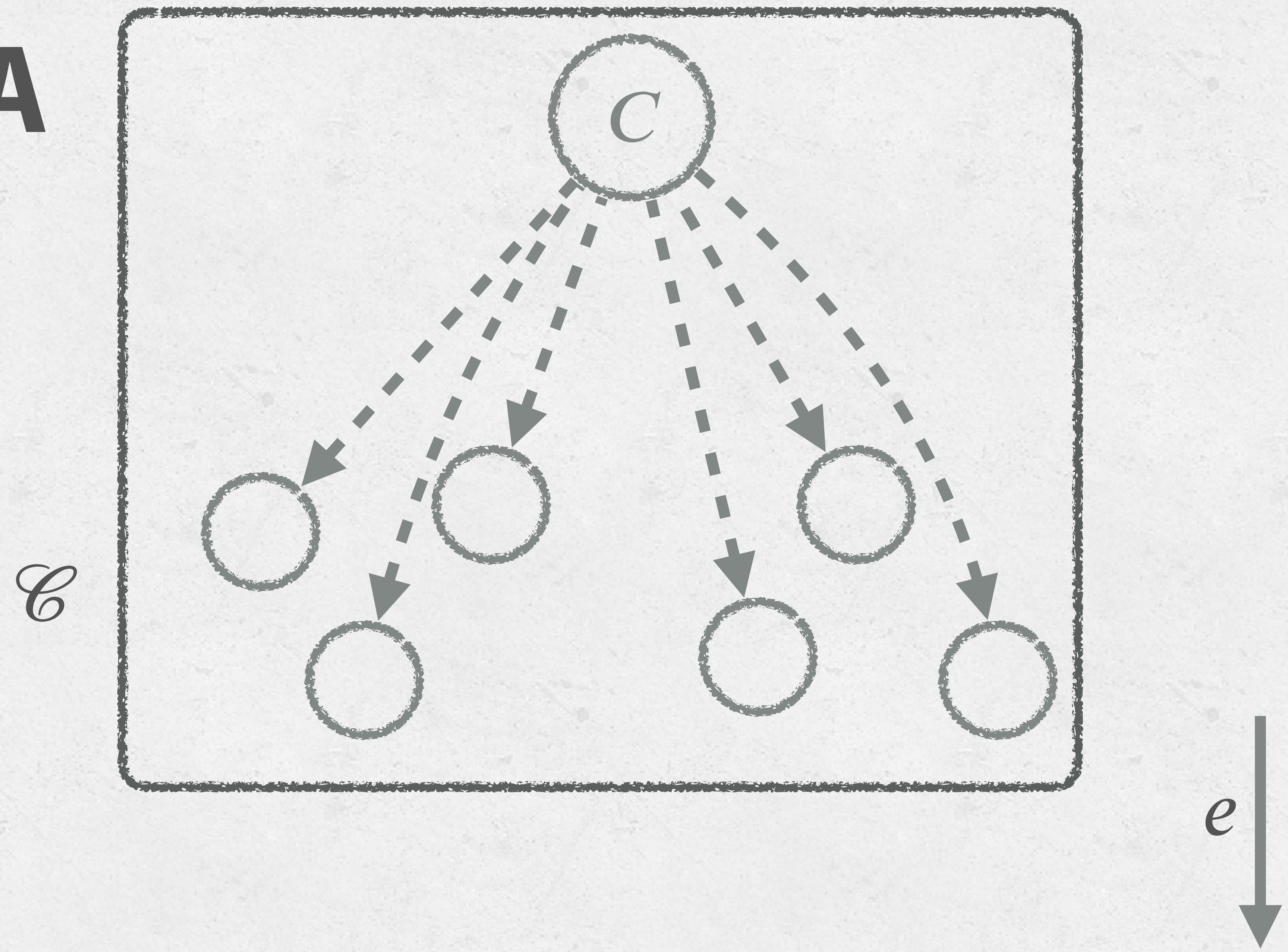


PROVING THE DELAY LEMMA

Consider a bivalent configuration, C, and an applicable event, e.

If $e(C)$ is bivalent, then we're done.

Otherwise, let $\mathscr{C}$ be the set of events reachable from C *without* applying e and $\mathscr{D}$ be $e(\mathscr{C}) = \{\, e(C) : C \in \mathscr{C} \,\}$ (i.e., the set of all configurations reachable from C where e was the last event taken).

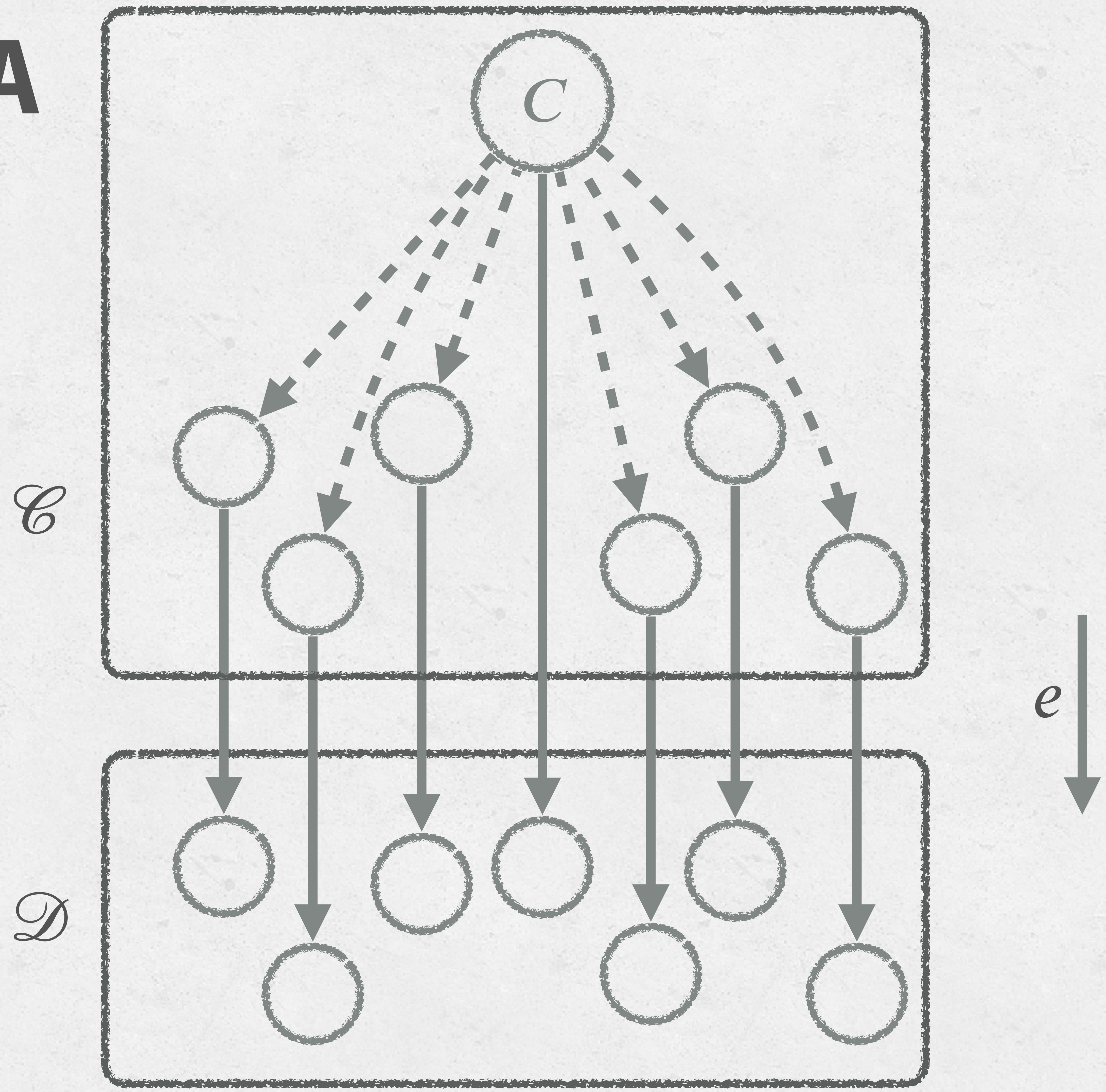


Proving the Delay Lemma

Consider a bivalent configuration, C, and an applicable event, e.

If $e(C)$ is bivalent, then we're done.

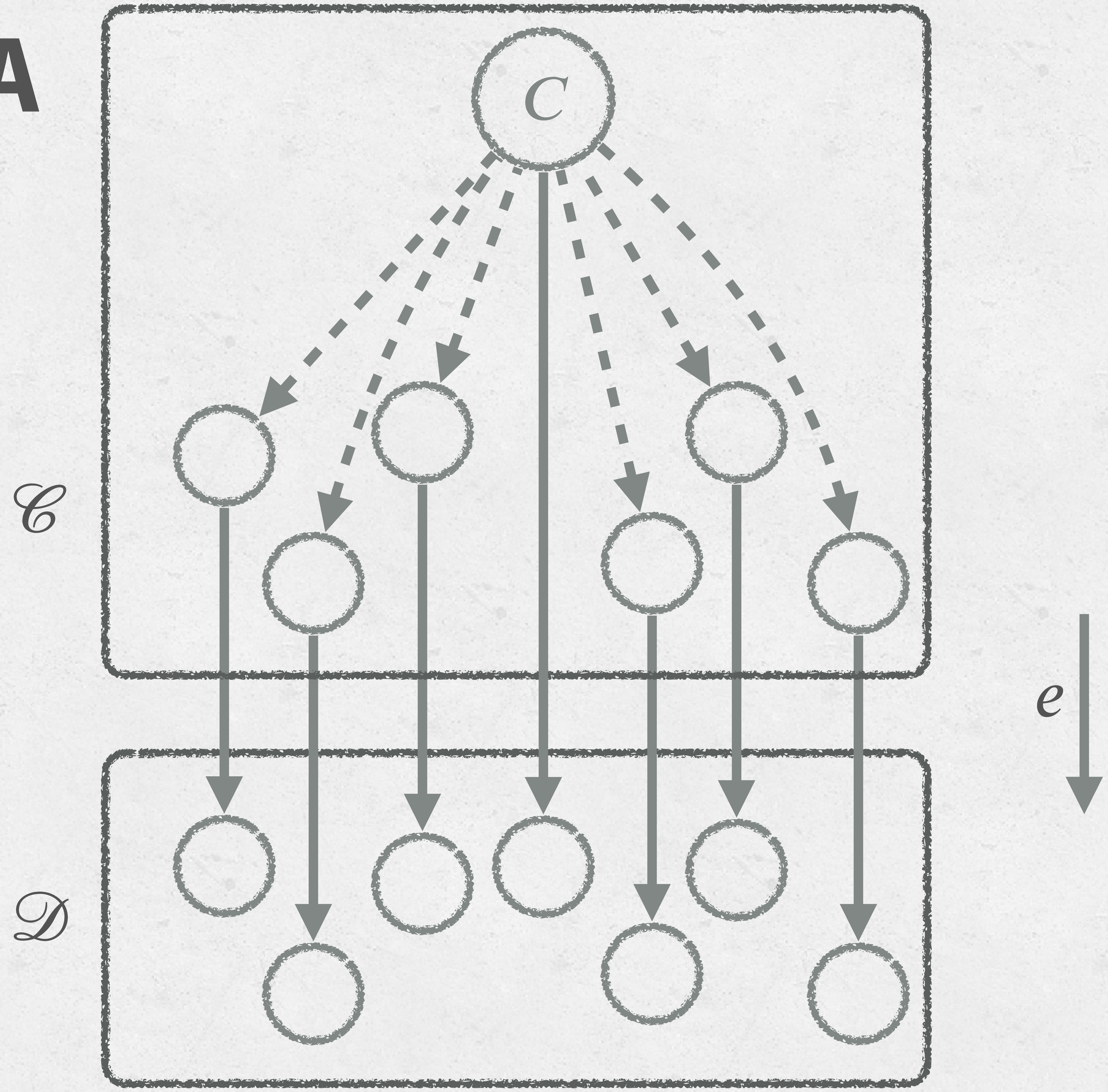
Otherwise, let $\mathscr{C}$ be the set of events reachable from C *without* applying e and $\mathscr{D}$ be $e(\mathscr{C}) = \{\, e(C) : C \in \mathscr{C} \,\}$ (i.e., the set of all configurations reachable from C where e was the last event taken).



Proving the Delay Lemma

We want to show $\mathscr{D}$ contains a bivalent configuration.

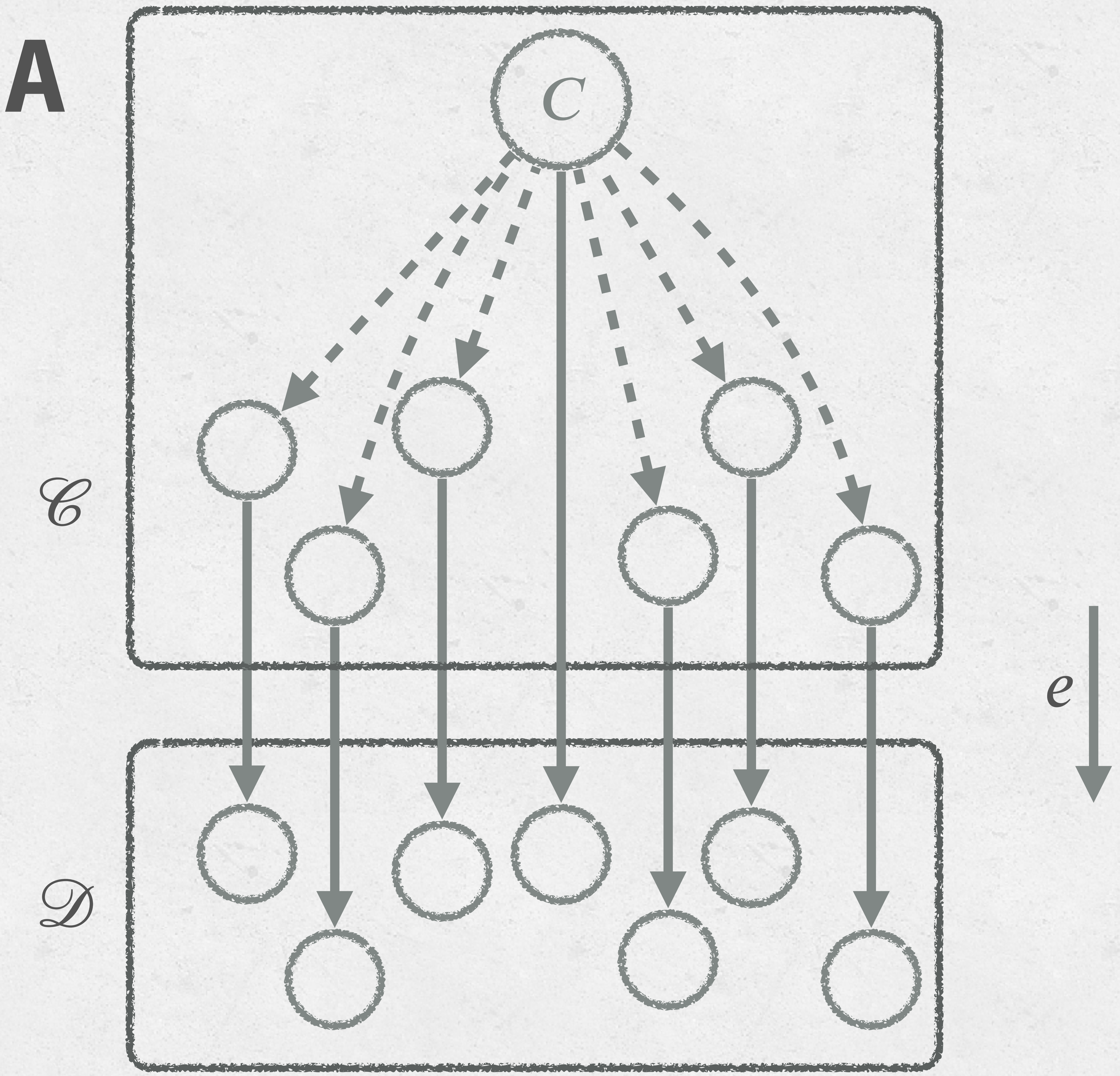
Suppose, for the sake of contradiction, that it doesn't.



Proving the Delay Lemma

We want to show $\mathscr{D}$ contains a bivalent configuration.
Suppose, for the sake of contradiction, that it doesn't.

Then, we first show there must exist **both** 0-valent and 1-valent configurations in $\mathscr{D}$.

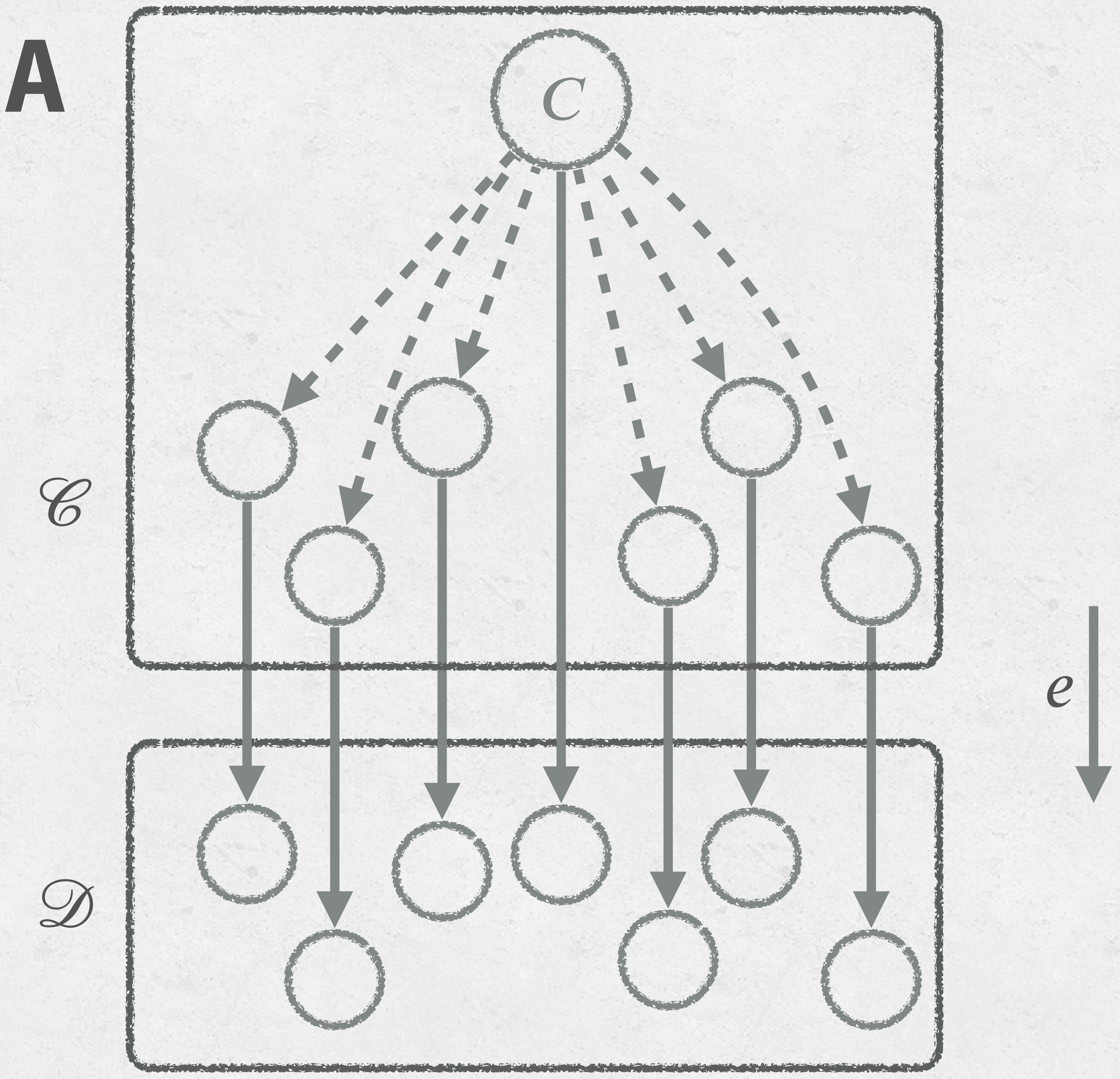


Proving the Delay Lemma

We want to show $\mathscr{D}$ contains a bivalent configuration.
Suppose, for the sake of contradiction, that it doesn't.

Then, we first show there must exist **both** 0-valent and 1-valent configurations in $\mathscr{D}$.

Because C is bivalent, there exist reachable 0-valent and 1-valent configurations. For each, this configuration is either:



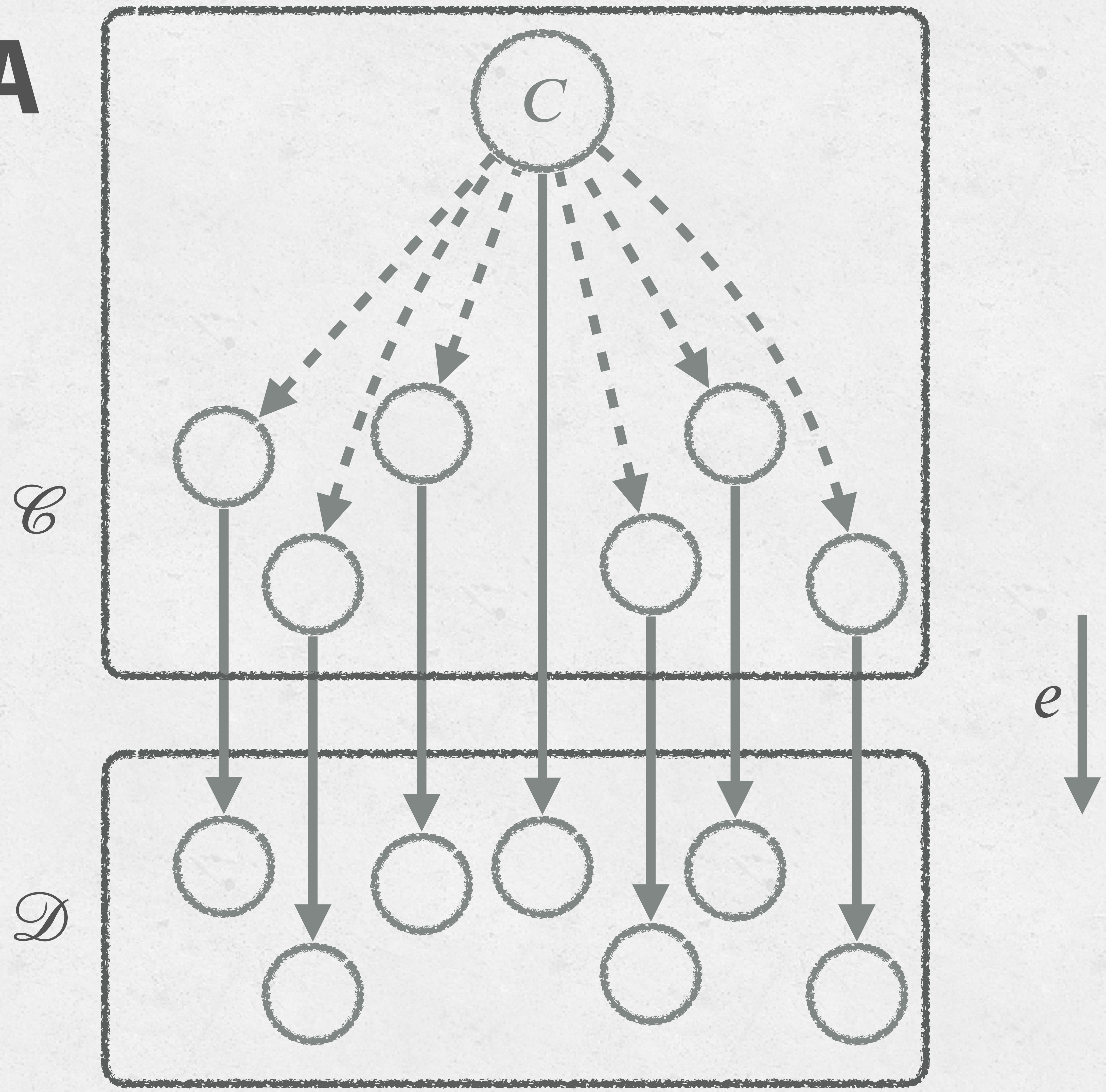
Proving the Delay Lemma

We want to show $\mathscr{D}$ contains a bivalent configuration.
Suppose, for the sake of contradiction, that it doesn't.

Then, we first show there must exist **both** 0-valent and 1-valent configurations in $\mathscr{D}$.

Because C is bivalent, there exist reachable 0-valent and 1-valent configurations. For each, this configuration is either:

1. In $\mathscr{D}$,



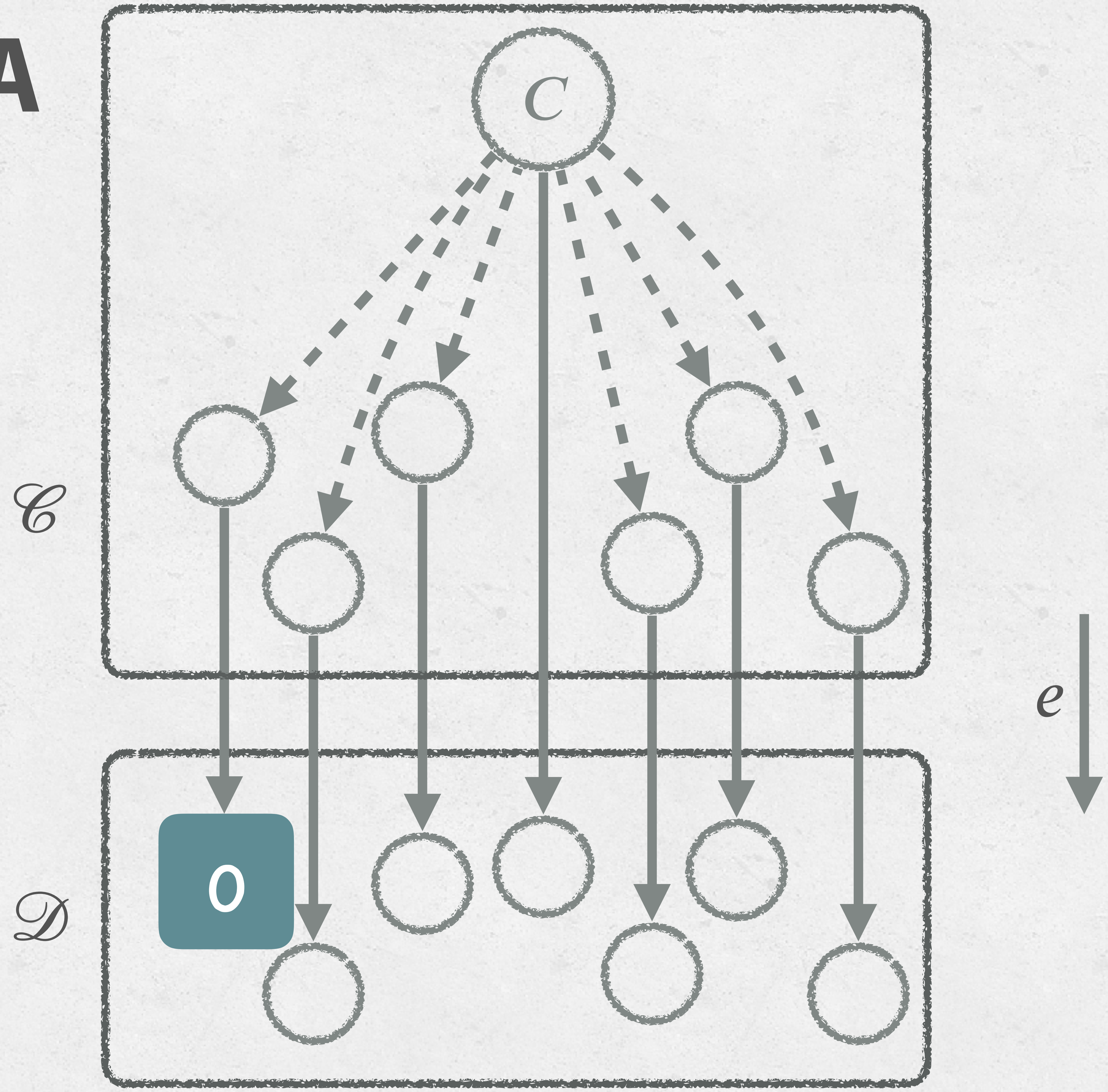
Proving the Delay Lemma

We want to show $\mathscr{D}$ contains a bivalent configuration.
Suppose, for the sake of contradiction, that it doesn't.

Then, we first show there must exist **both** 0-valent and 1-valent configurations in $\mathscr{D}$.

Because C is bivalent, there exist reachable 0-valent and 1-valent configurations. For each, this configuration is either:

1. In $\mathscr{D}$,



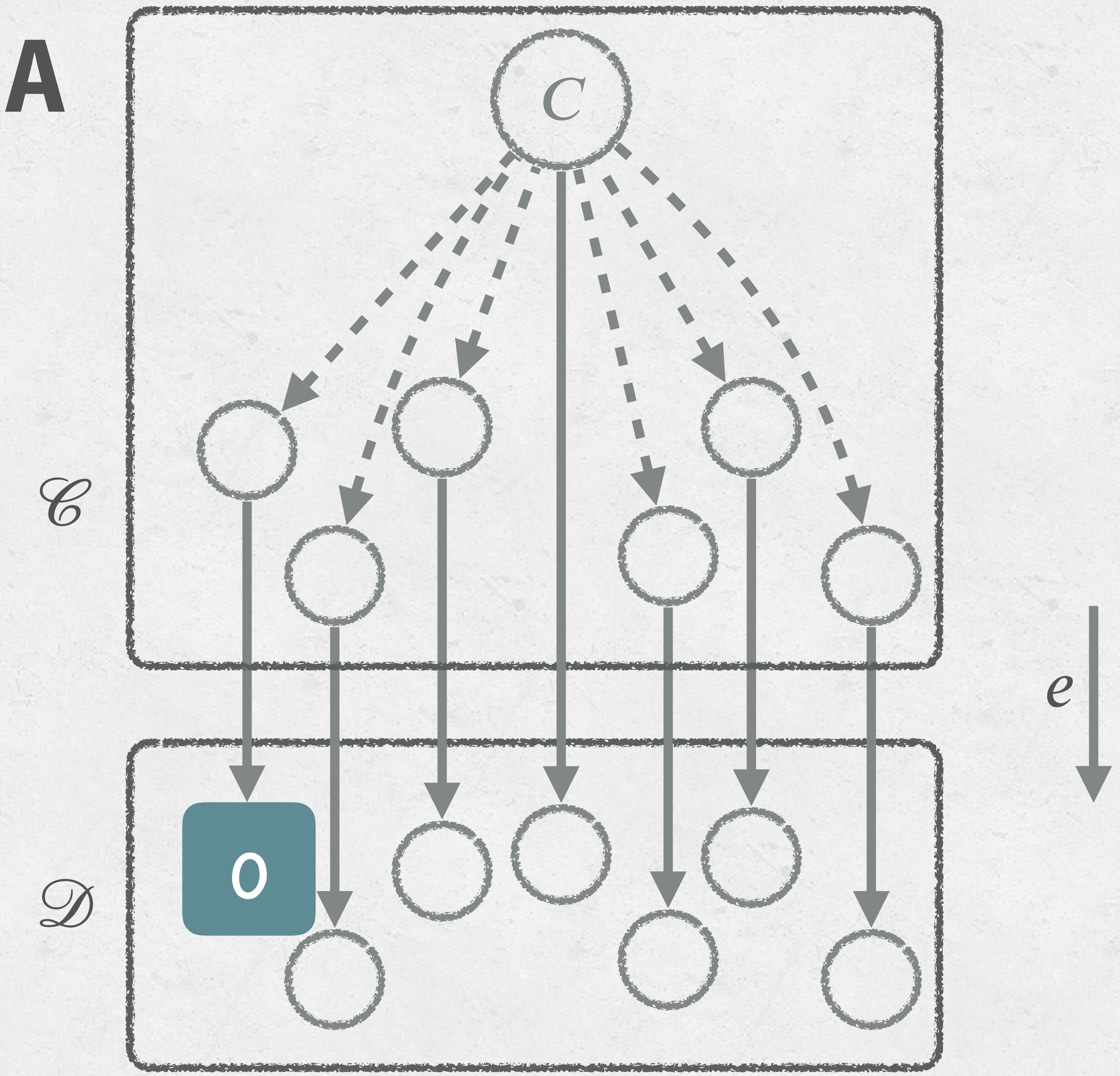
Proving the Delay Lemma

We want to show $\mathscr{D}$ contains a bivalent configuration.
Suppose, for the sake of contradiction, that it doesn't.

Then, we first show there must exist **both** 0-valent and 1-valent configurations in $\mathscr{D}$.

Because C is bivalent, there exist reachable 0-valent and 1-valent configurations. For each, this configuration is either:

1. In $\mathscr{D}$,
2. In $\mathscr{C}$ (just apply e),



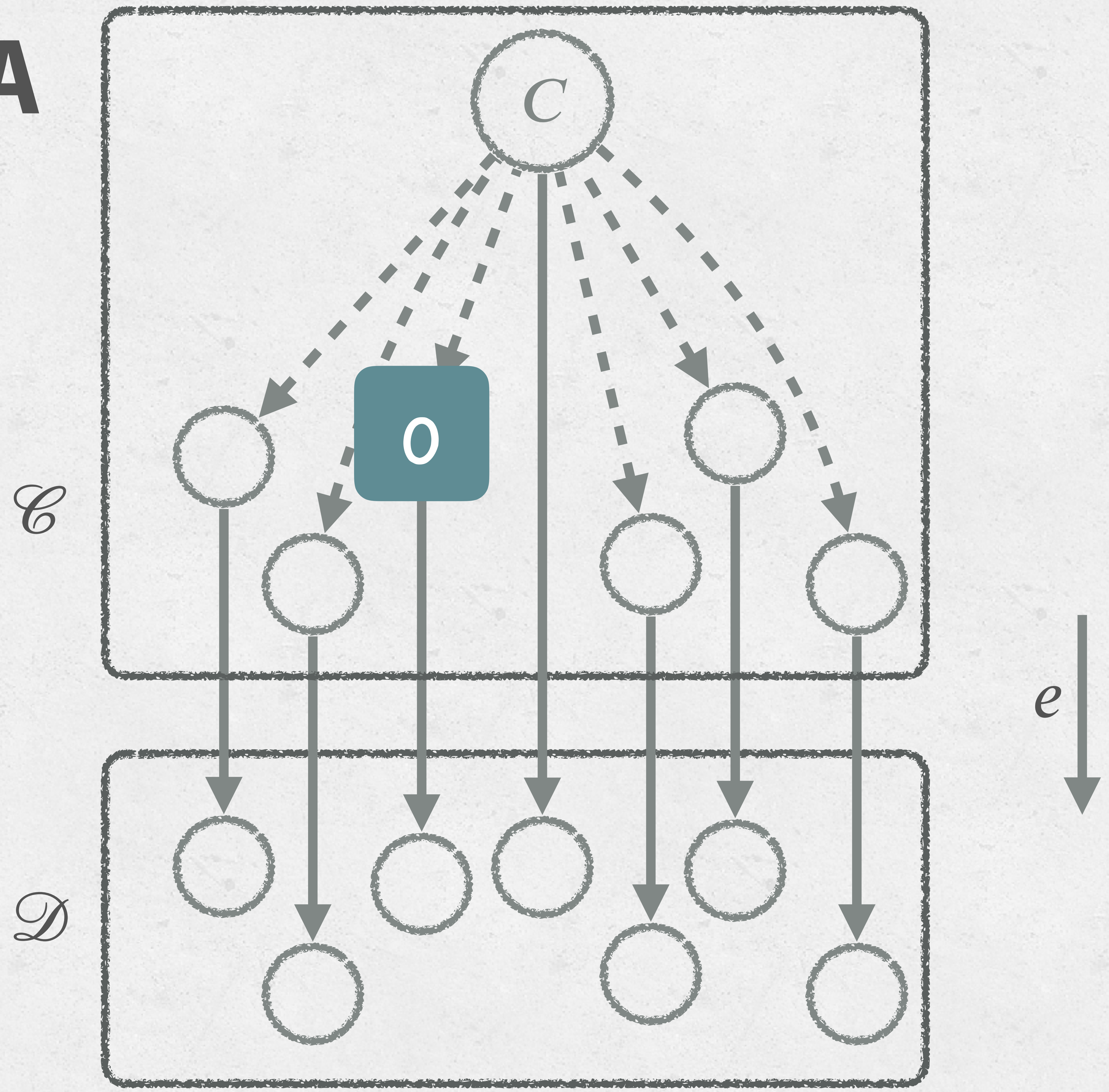
Proving the Delay Lemma

We want to show $\mathscr{D}$ contains a bivalent configuration.
Suppose, for the sake of contradiction, that it doesn't.

Then, we first show there must exist **both** 0-valent and 1-valent configurations in $\mathscr{D}$.

Because C is bivalent, there exist reachable 0-valent and 1-valent configurations. For each, this configuration is either:

1. In $\mathscr{D}$,
2. In $\mathscr{C}$ (just apply e),



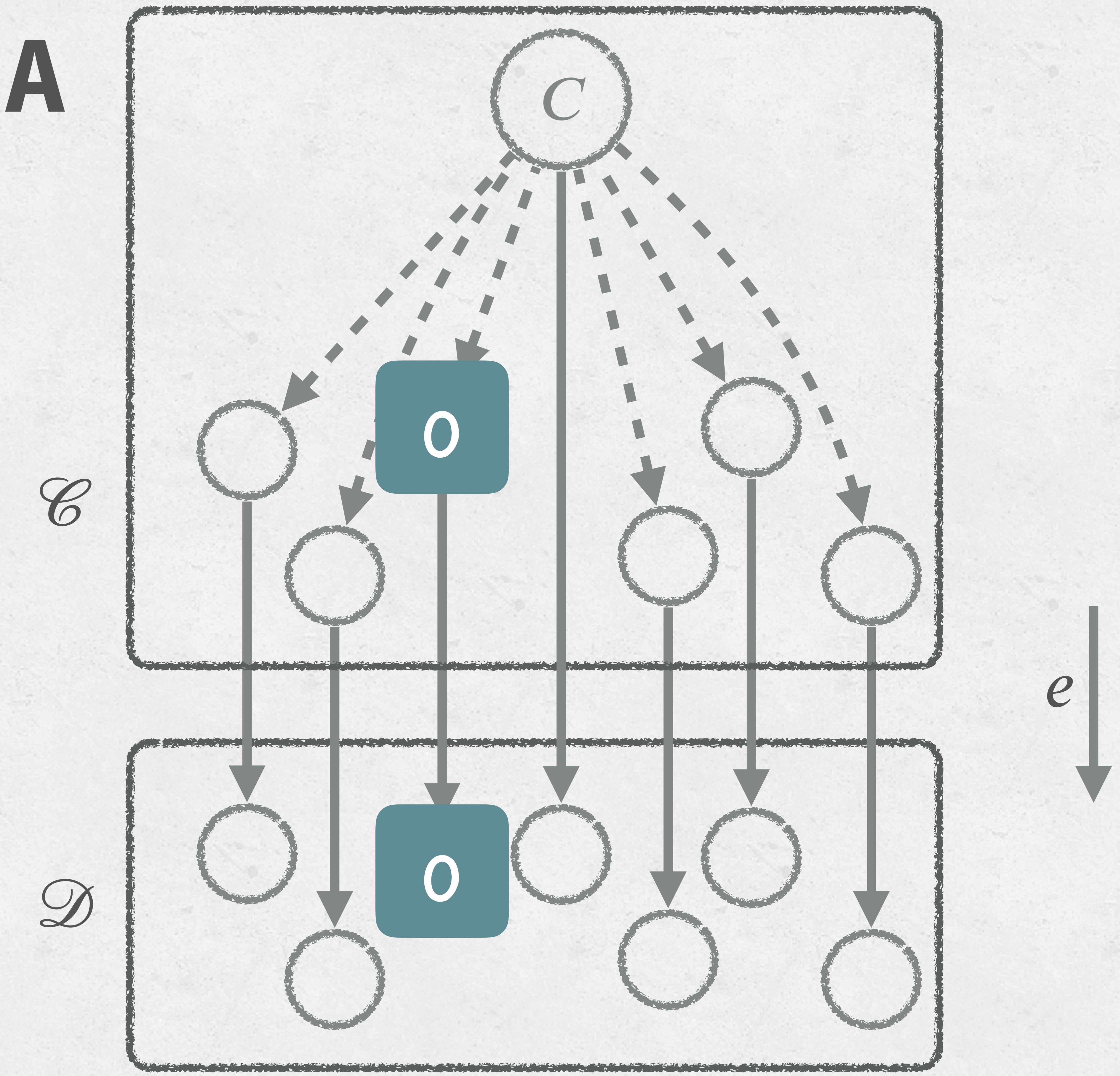
Proving the Delay Lemma

We want to show $\mathscr{D}$ contains a bivalent configuration.
Suppose, for the sake of contradiction, that it doesn't.

Then, we first show there must exist **both** 0-valent and 1-valent configurations in $\mathscr{D}$.

Because C is bivalent, there exist reachable 0-valent and 1-valent configurations. For each, this configuration is either:

1. In $\mathscr{D}$,
2. In $\mathscr{C}$ (just apply e),



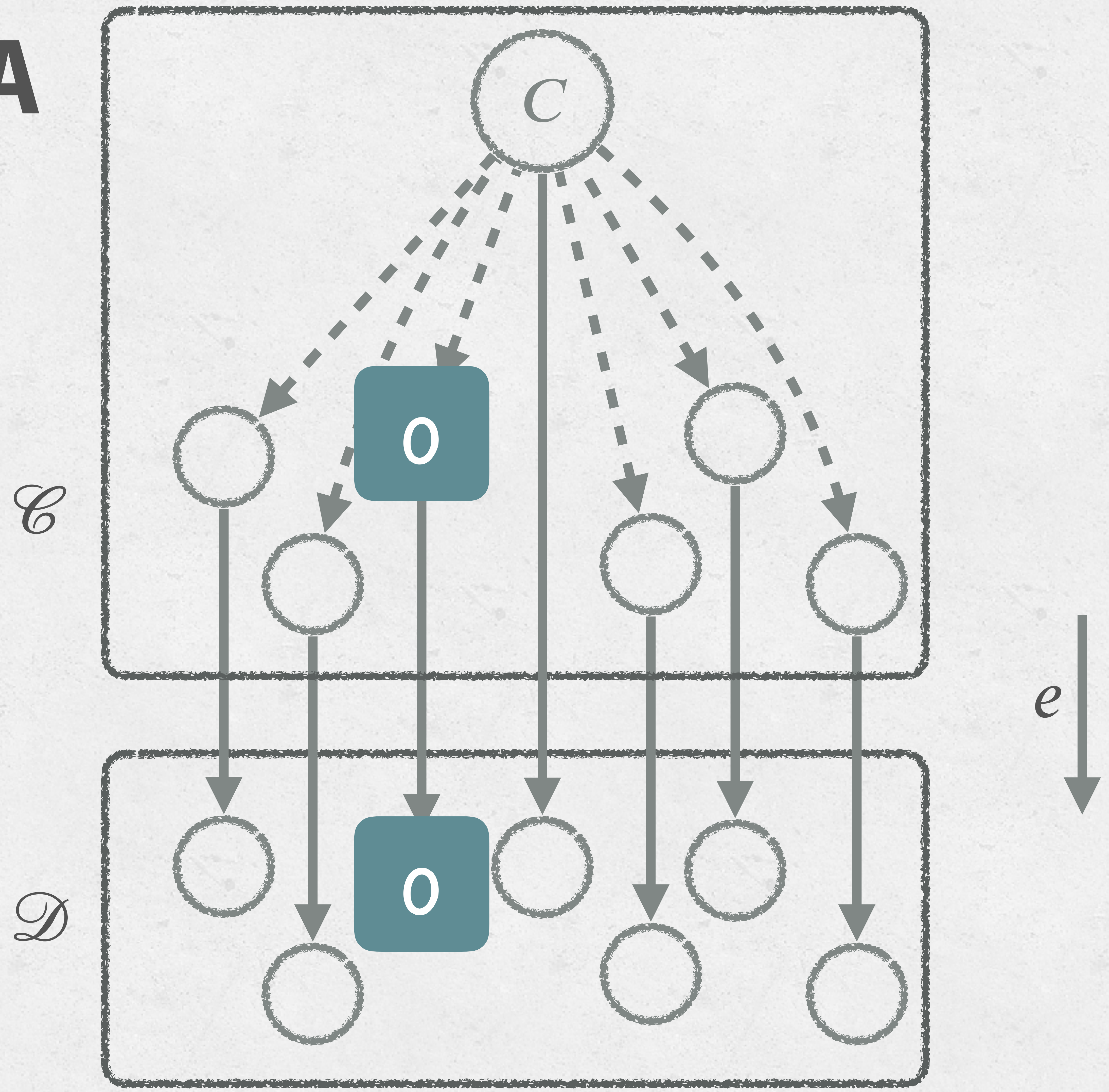
Proving the Delay Lemma

We want to show $\mathscr{D}$ contains a bivalent configuration.
Suppose, for the sake of contradiction, that it doesn't.

Then, we first show there must exist **both** 0-valent and 1-valent configurations in $\mathscr{D}$.

Because C is bivalent, there exist reachable 0-valent and 1-valent configurations. For each, this configuration is either:

1. In $\mathscr{D}$,
2. In $\mathscr{C}$ (just apply e),
3. Or past $\mathscr{D}$ (the ancestor in $\mathscr{D}$ must also be of the same valency since it's not bivalent by assumption).



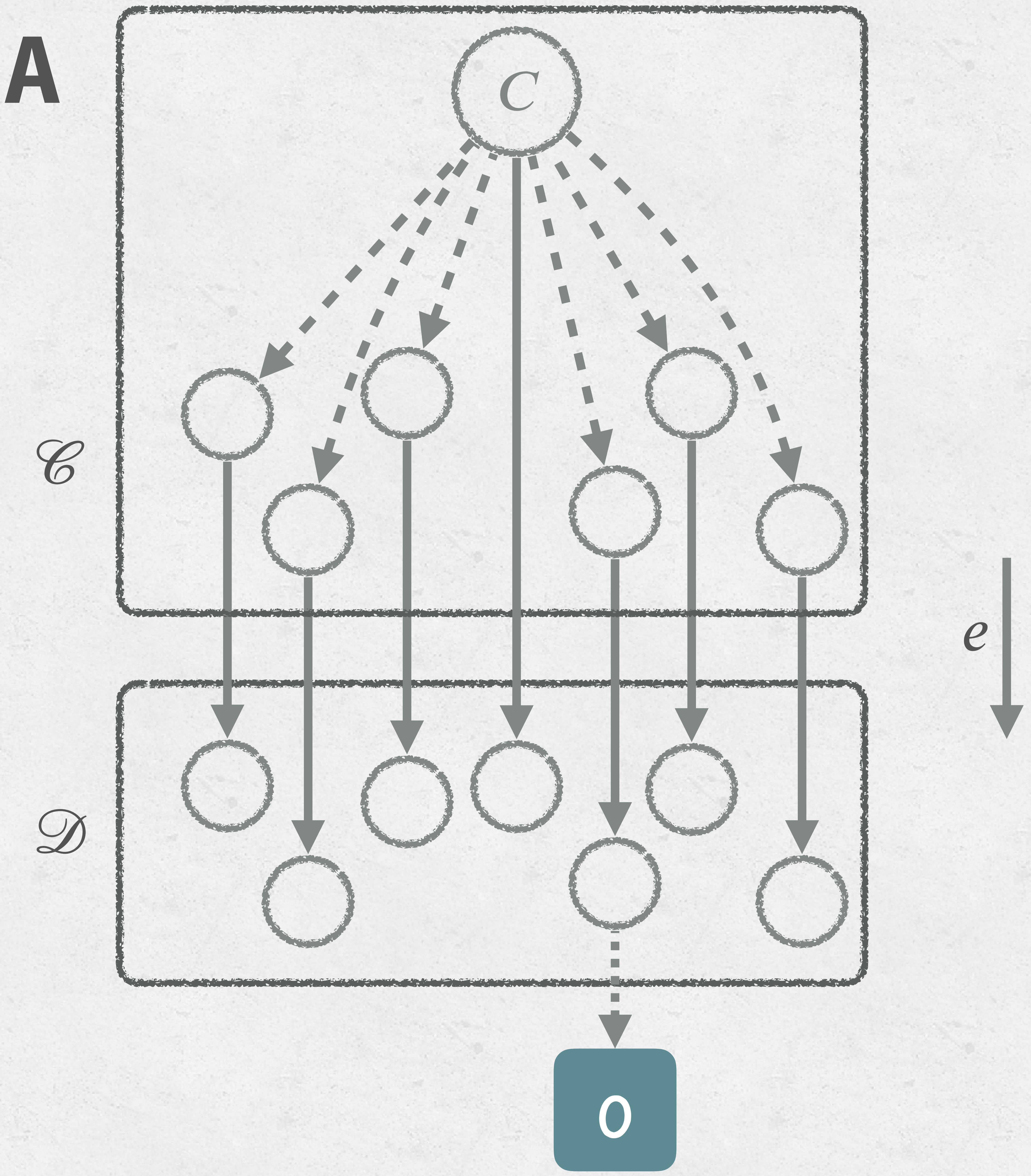
PROVING THE DELAY LEMMA

We want to show $\mathscr{D}$ contains a bivalent configuration.
Suppose, for the sake of contradiction, that it doesn't.

Then, we first show there must exist **both** 0-valent and 1-valent configurations in $\mathscr{D}$.

Because C is bivalent, there exist reachable 0-valent and 1-valent configurations. For each, this configuration is either:

1. In $\mathscr{D}$,
2. In $\mathscr{C}$ (just apply e),
3. Or past $\mathscr{D}$ (the ancestor in $\mathscr{D}$ must also be of the same valency since it's not bivalent by assumption).



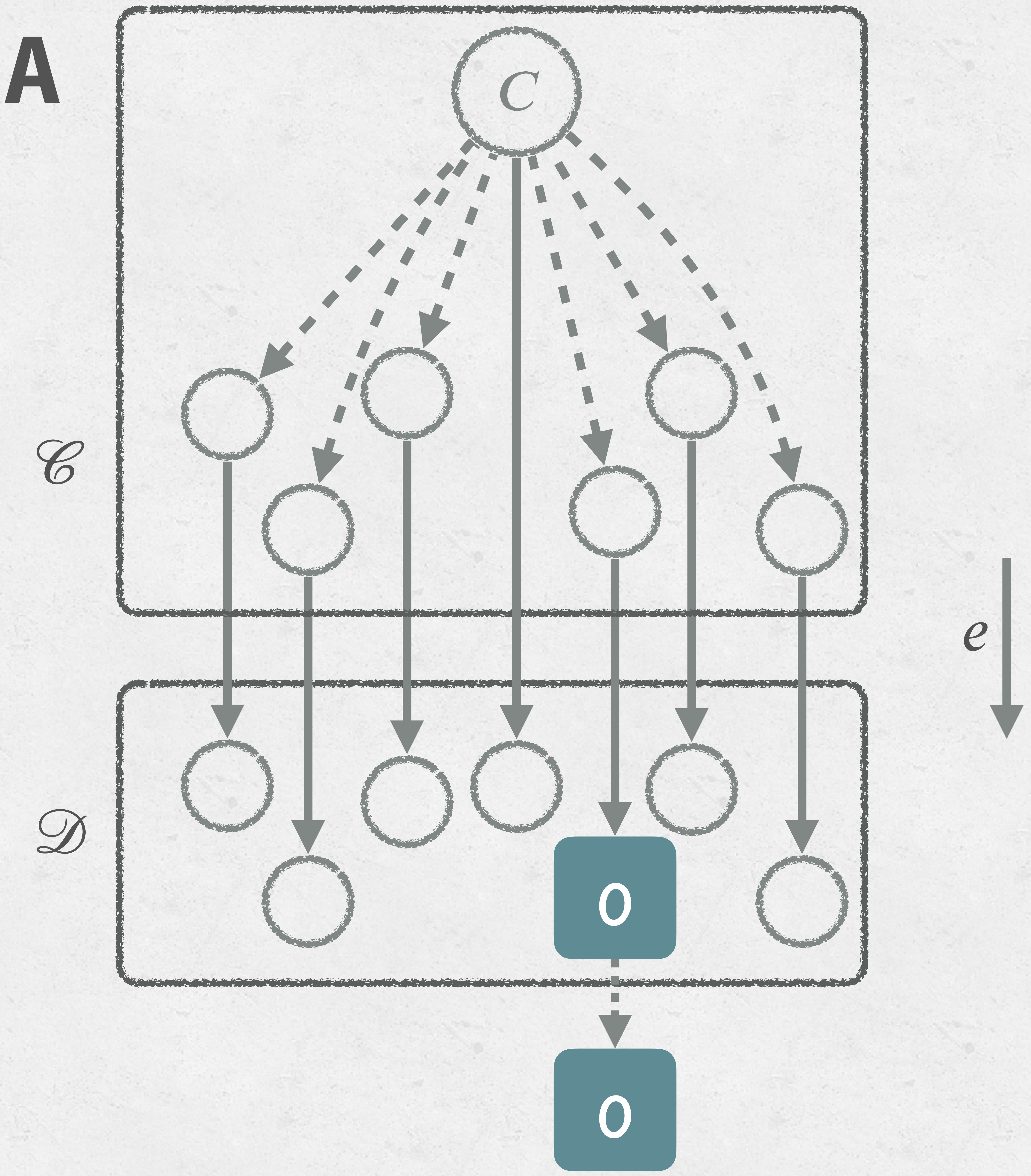
Proving the Delay Lemma

We want to show $\mathscr{D}$ contains a bivalent configuration.
Suppose, for the sake of contradiction, that it doesn't.

Then, we first show there must exist **both** 0-valent and 1-valent configurations in $\mathscr{D}$.

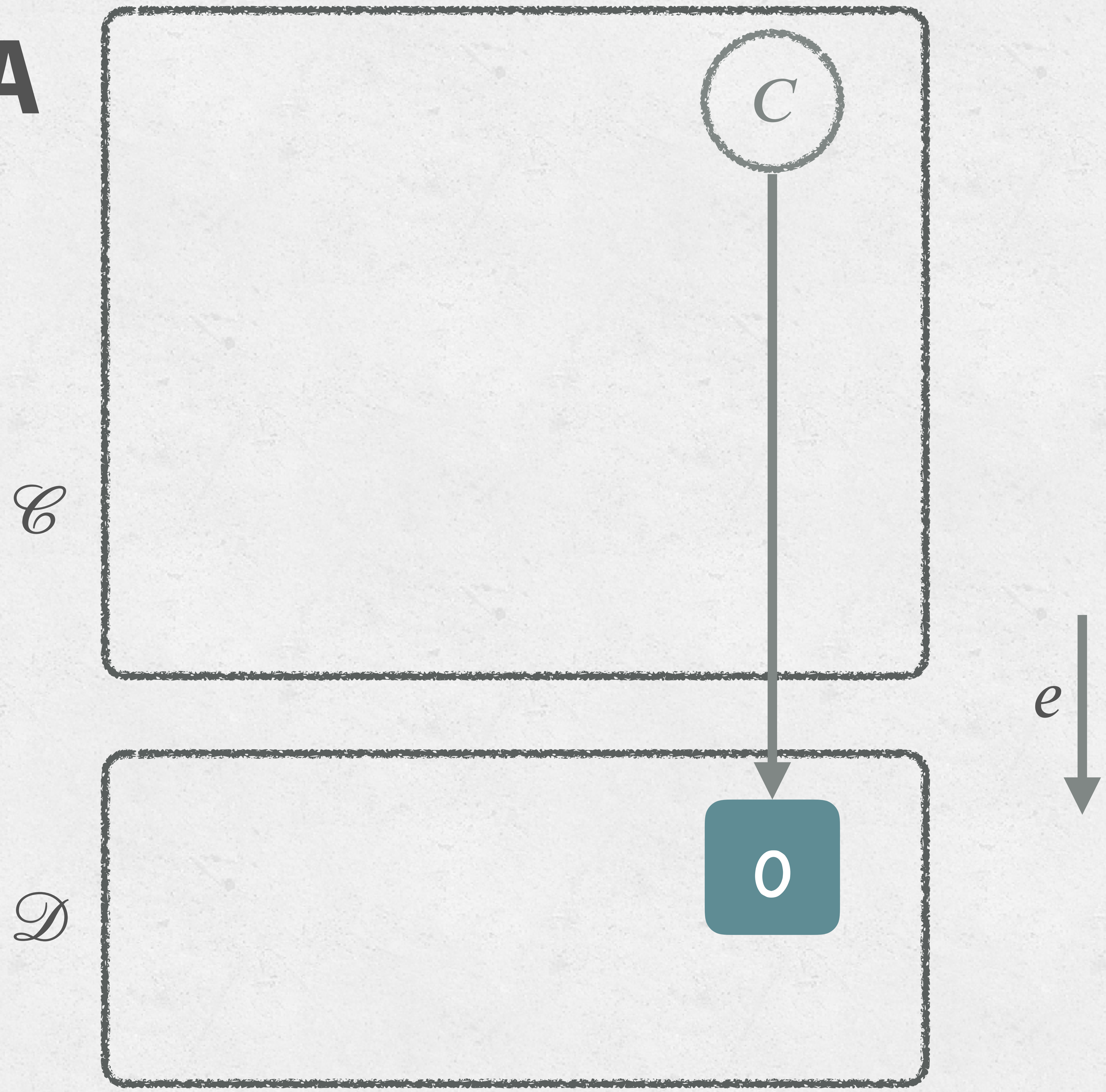
Because C is bivalent, there exist reachable 0-valent and 1-valent configurations. For each, this configuration is either:

1. In $\mathscr{D}$,
2. In $\mathscr{C}$ (just apply e),
3. Or past $\mathscr{D}$ (the ancestor in $\mathscr{D}$ must also be of the same valency since it's not bivalent by assumption).



Proving the Delay Lemma

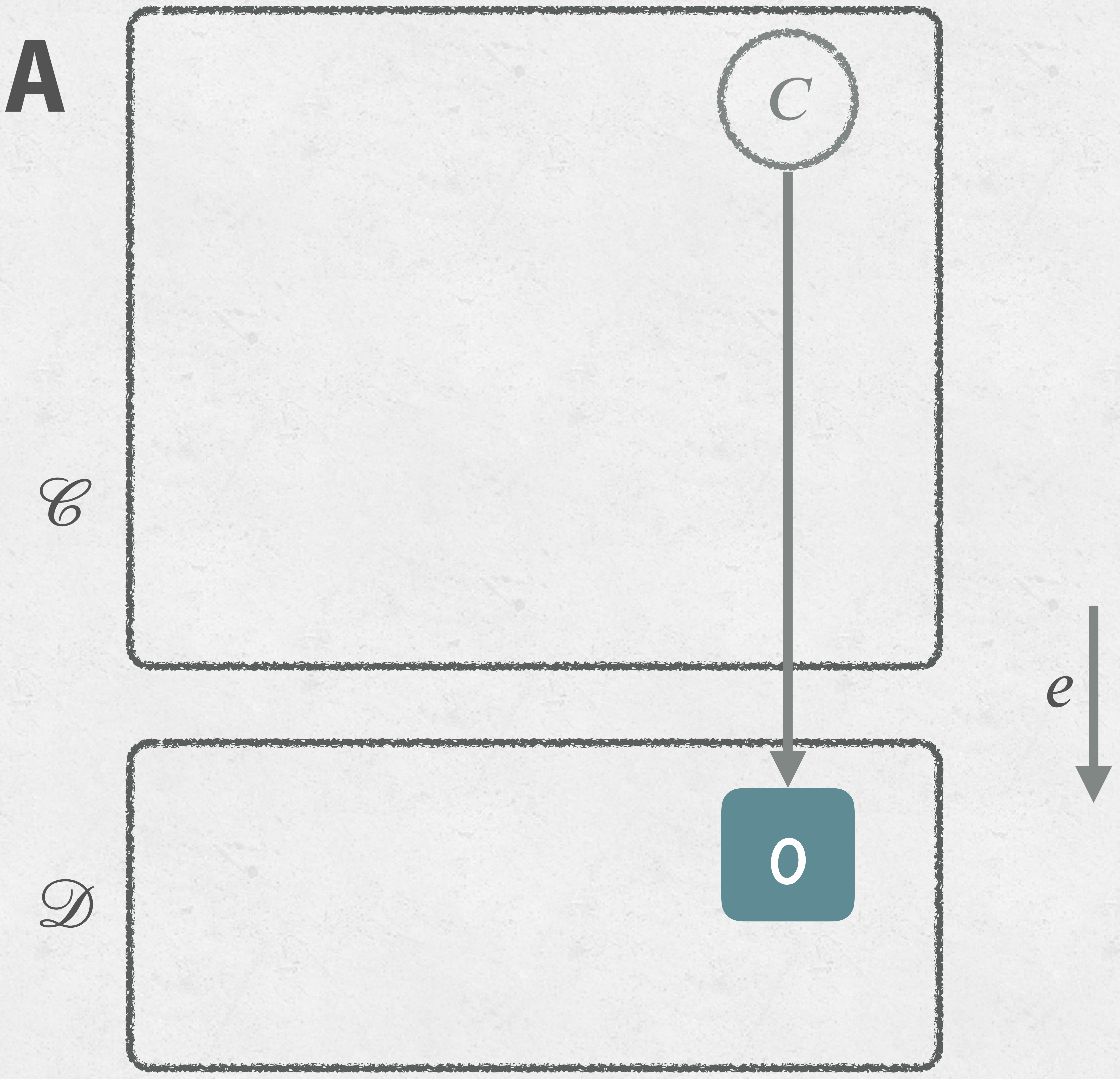
Now, consider the valency of $e(C)$. Without loss of generality, let's say it's 0.



Proving the Delay Lemma

Now, consider the valency of $e(C)$. Without loss of generality, let's say it's 0.

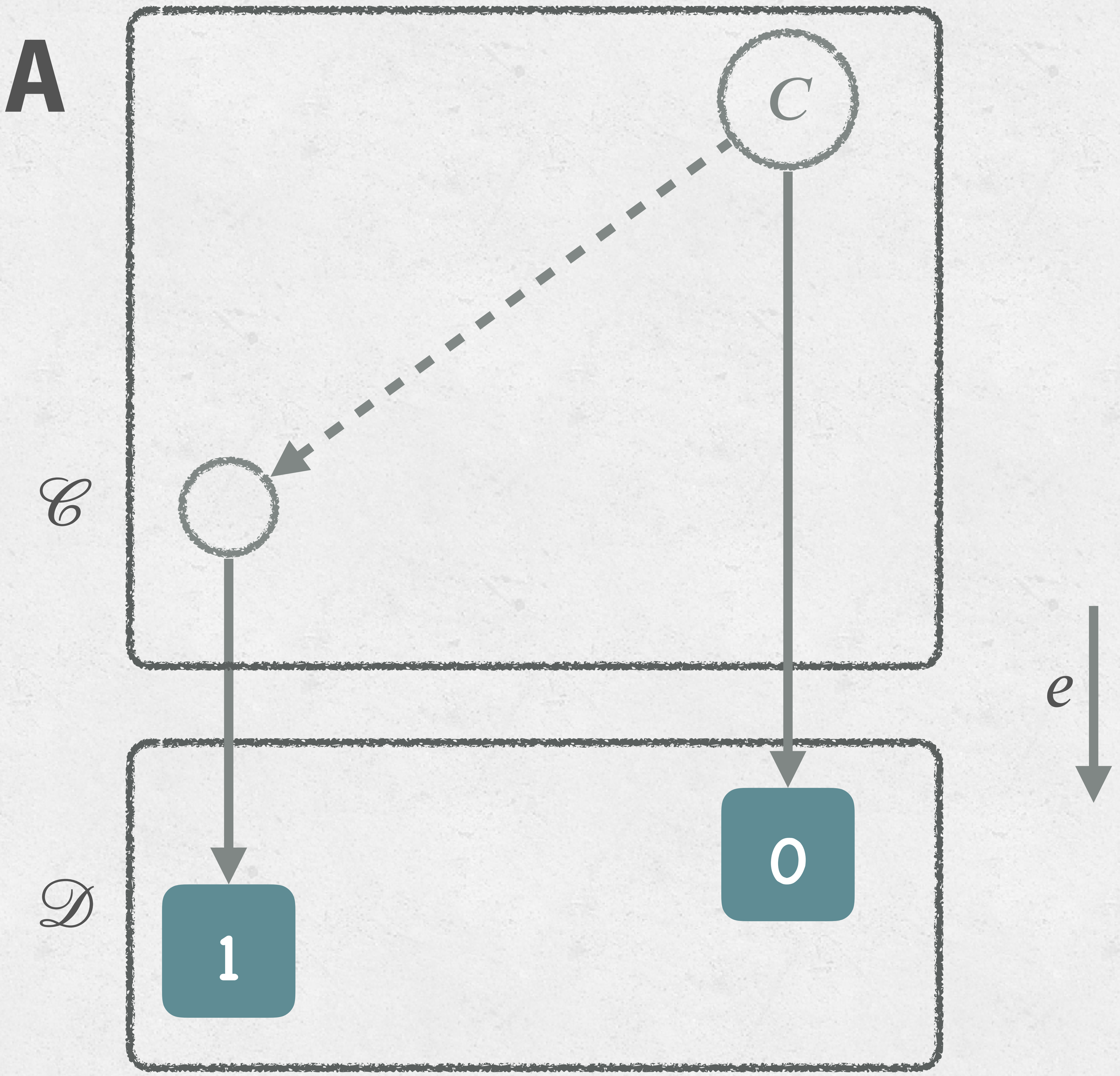
Because there are 1-valent configurations in $\mathscr{D}$, there must be a path from C to one of these.



Proving the Delay Lemma

Now, consider the valency of $e(C)$. Without loss of generality, let's say it's 0.

Because there are 1-valent configurations in $\mathscr{D}$, there must be a path from C to one of these.

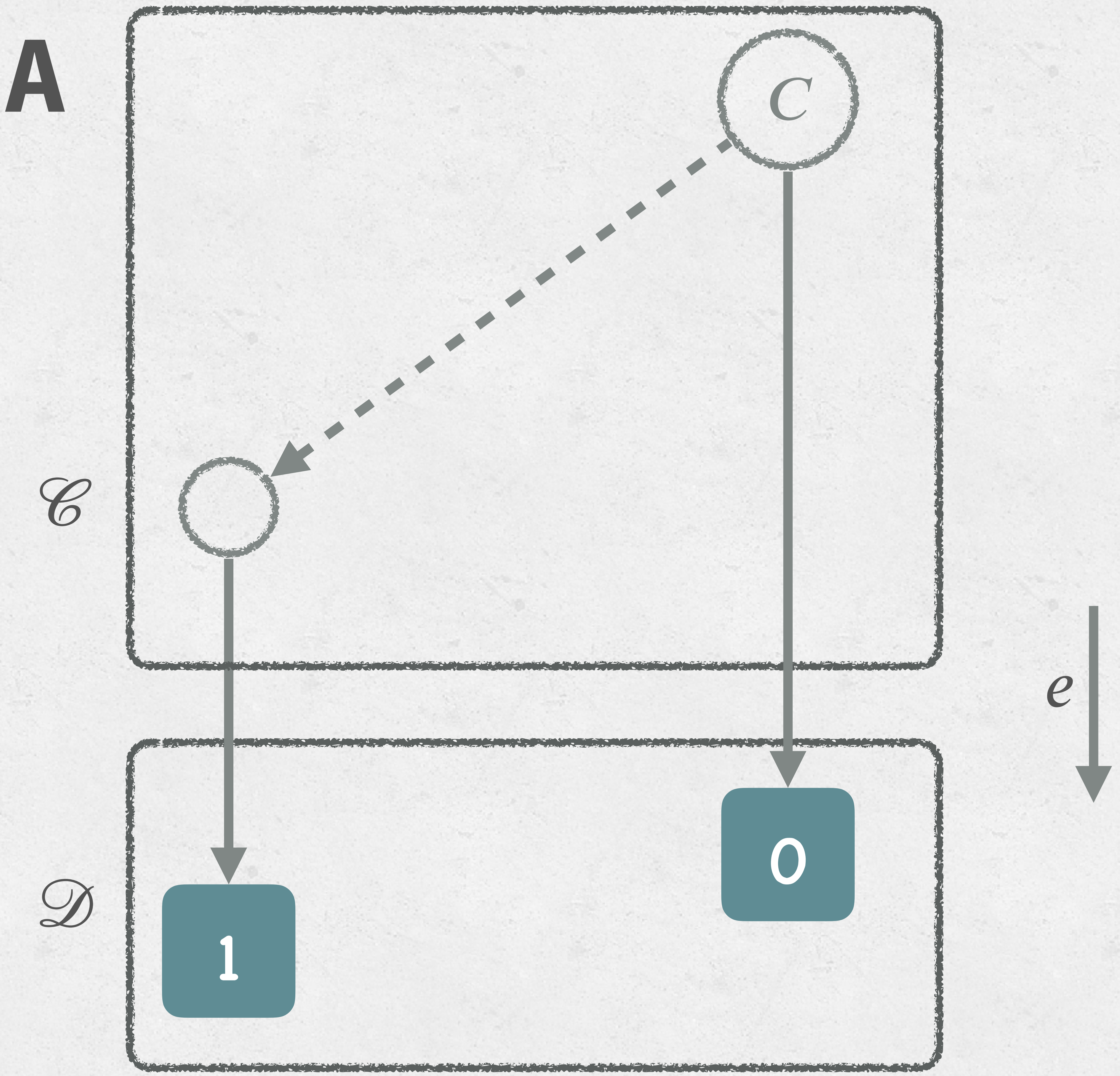


Proving the Delay Lemma

Now, consider the valency of $e(C)$. Without loss of generality, let's say it's 0.

Because there are 1-valent configurations in $\mathscr{D}$, there must be a path from C to one of these.

Then, there must exist adjacent configurations, C_0 and C_1, where $e(C_0)$ is 0-valent and $e(C_1)$ is 1-valent.

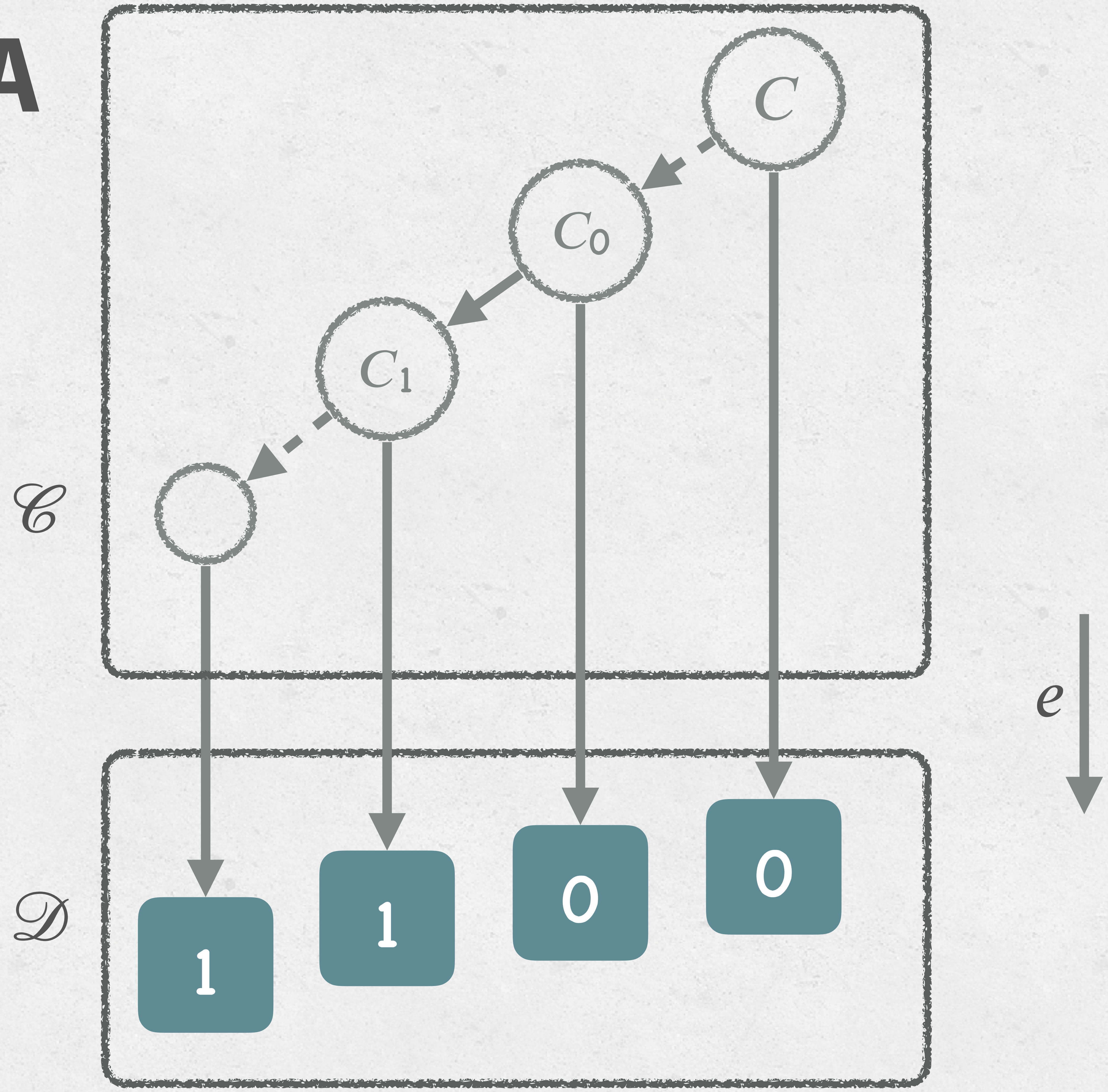


Proving the Delay Lemma

Now, consider the valency of $e(C)$. Without loss of generality, let's say it's 0.

Because there are 1-valent configurations in $\mathscr{D}$, there must be a path from C to one of these.

Then, there must exist adjacent configurations, C_0 and C_1, where $e(C_0)$ is 0-valent and $e(C_1)$ is 1-valent.



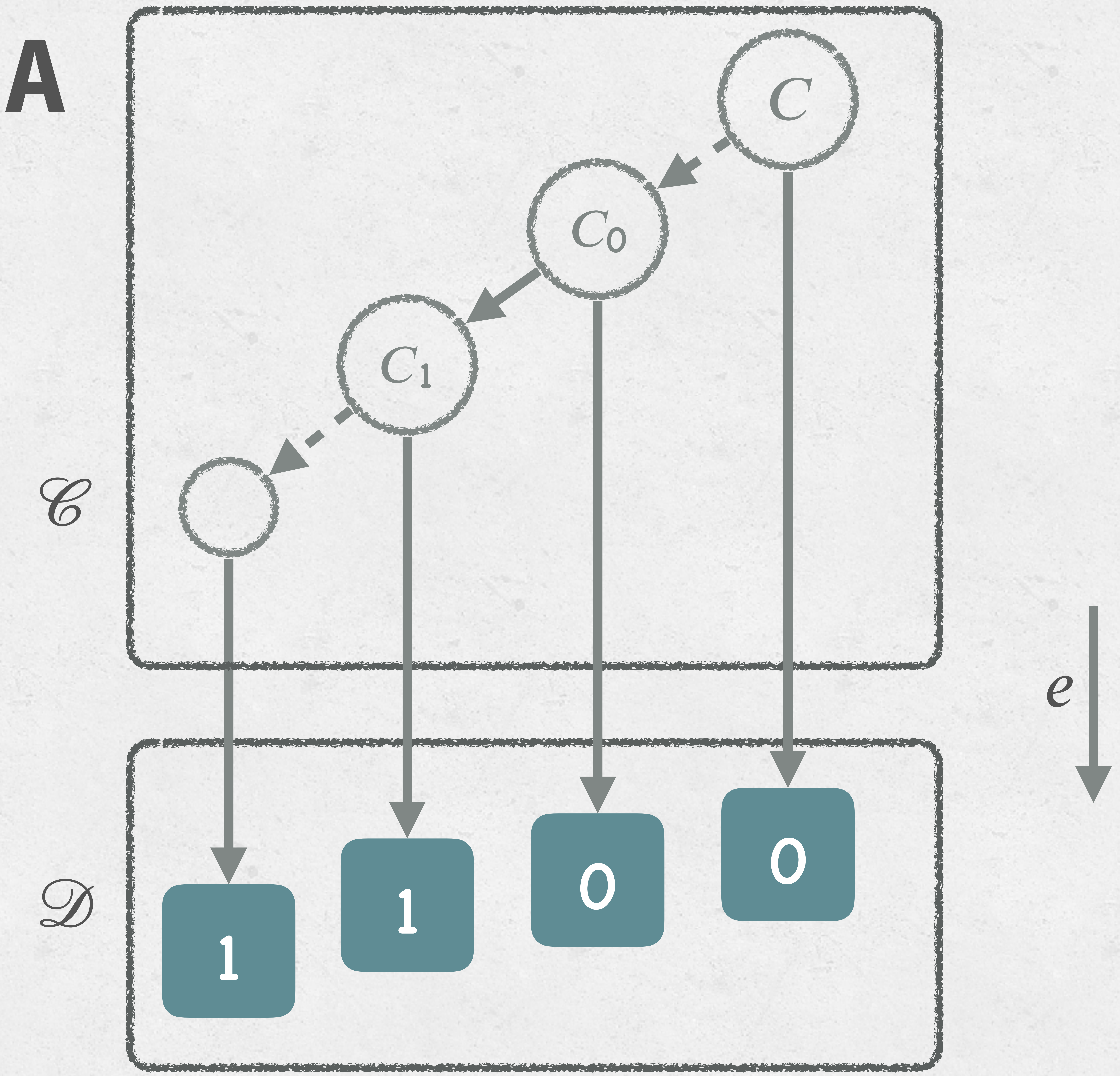
Proving the Delay Lemma

Now, consider the valency of $e(C)$. Without loss of generality, let's say it's 0.

Because there are 1-valent configurations in $\mathscr{D}$, there must be a path from C to one of these.

Then, there must exist adjacent configurations, C_0 and C_1, where $e(C_0)$ is 0-valent and $e(C_1)$ is 1-valent.

Let's call the event that takes C_0 to C_1 g.



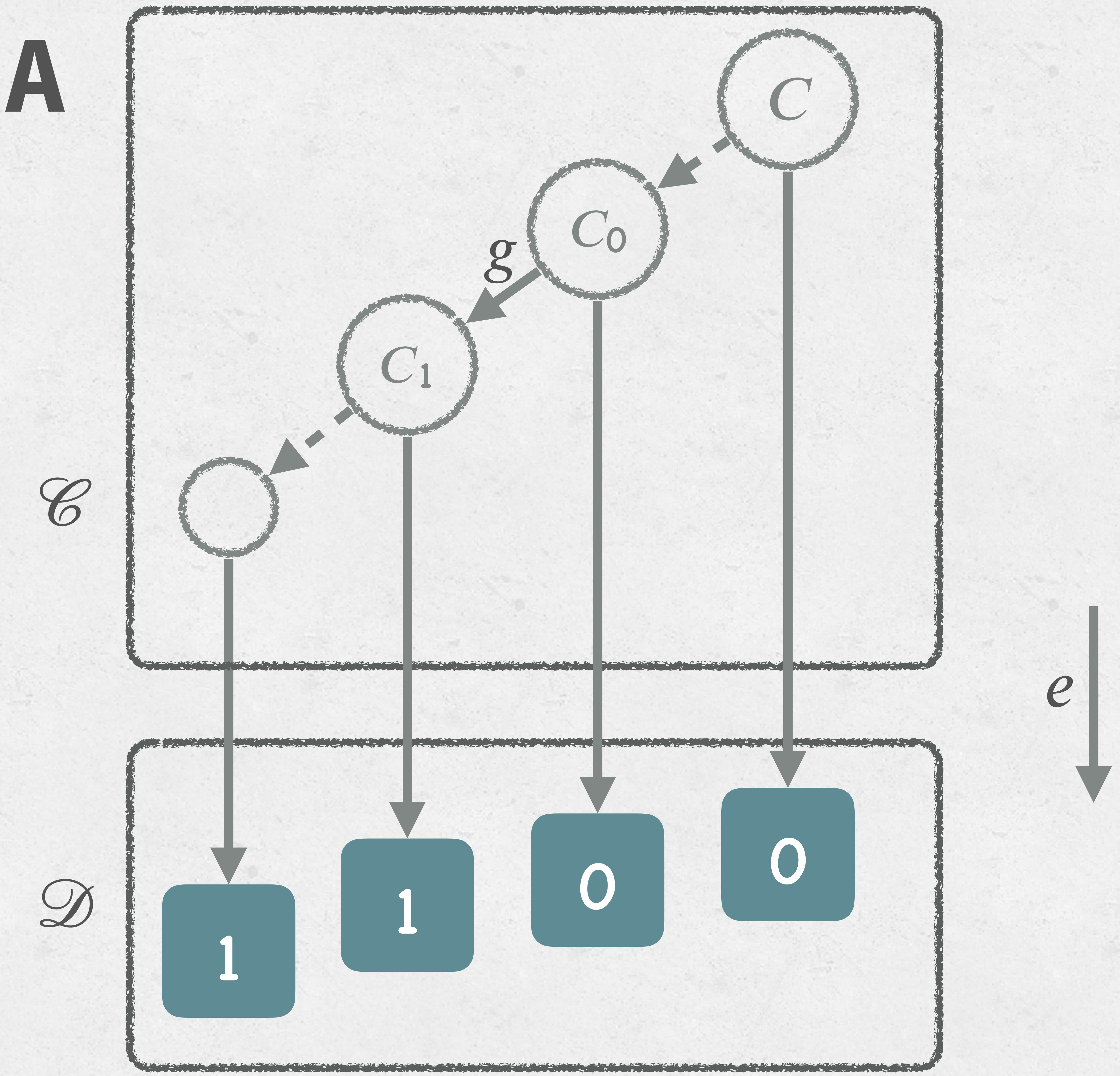
PROVING THE DELAY LEMMA

Now, consider the valency of $e(C)$. Without loss of generality, let's say it's 0.

Because there are 1-valent configurations in $\mathscr{D}$, there must be a path from C to one of these.

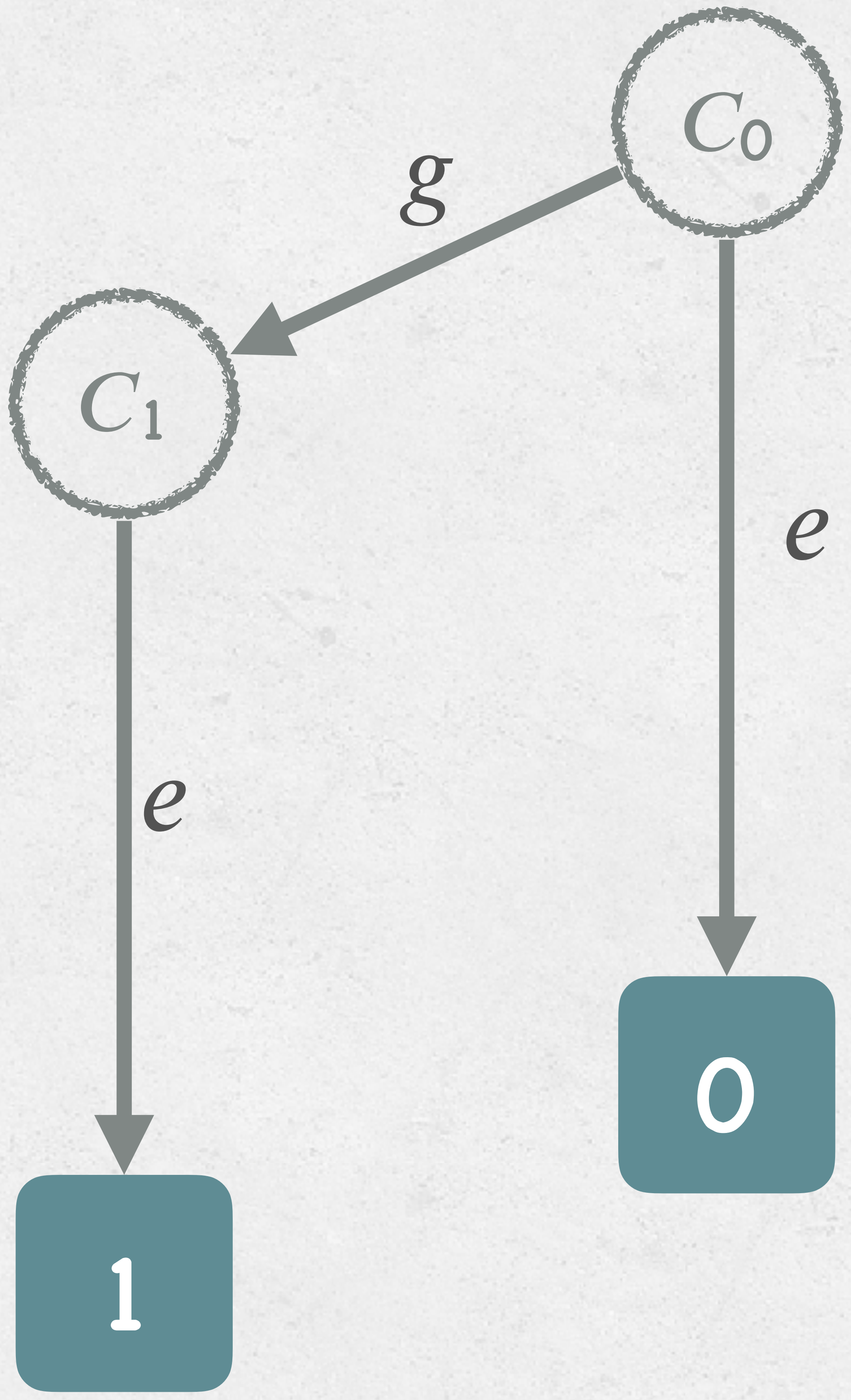
Then, there must exist adjacent configurations, C_0 and C_1, where $e(C_0)$ is 0-valent and $e(C_1)$ is 1-valent.

Let's call the event that takes C_0 to C_1 g.



Proving the Delay Lemma

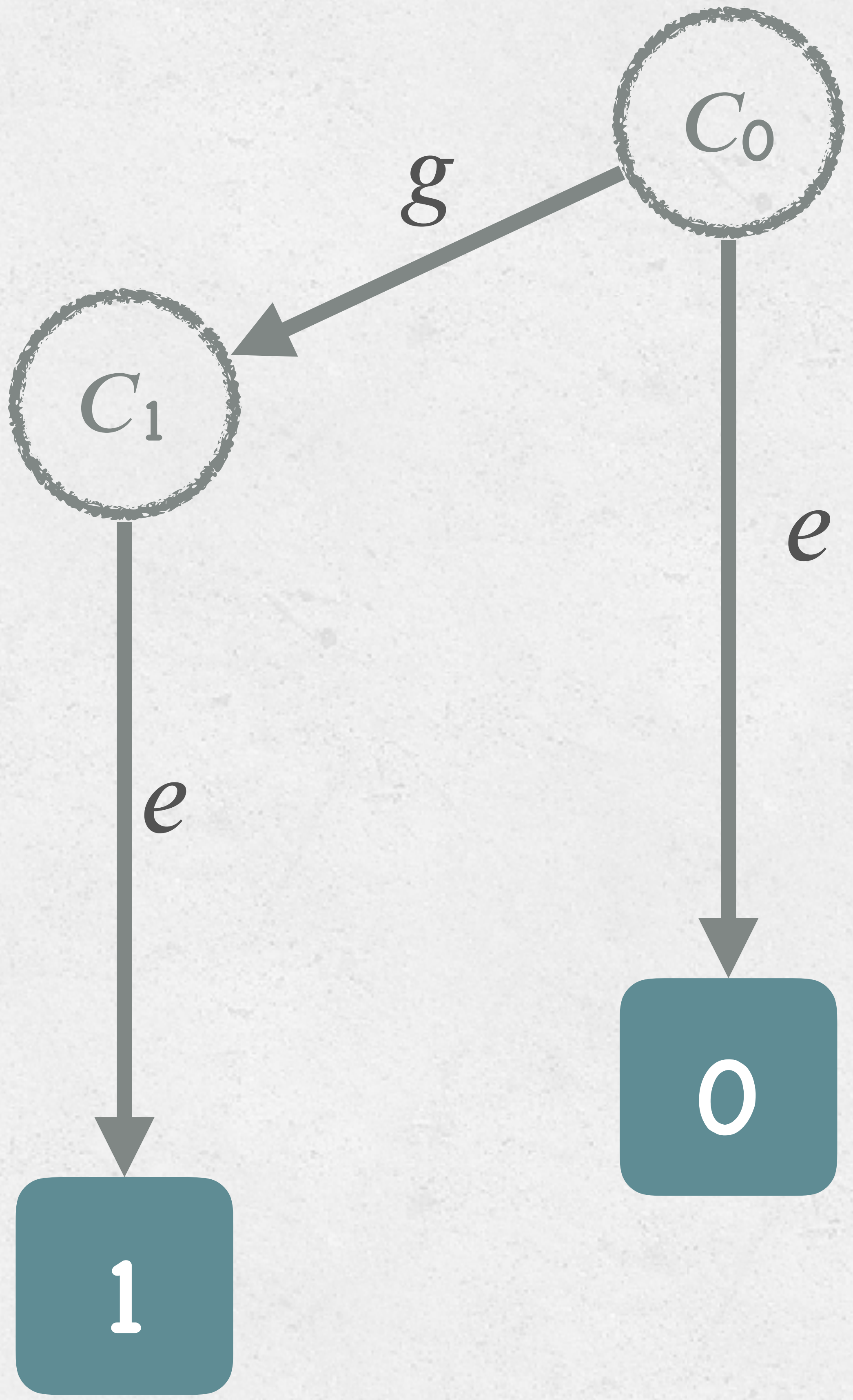
Almost done! First, we will show that the processes taking steps e and g must be the same process.



Proving the Delay Lemma

Almost done! First, we will show that the processes taking steps e and g must be the same process.

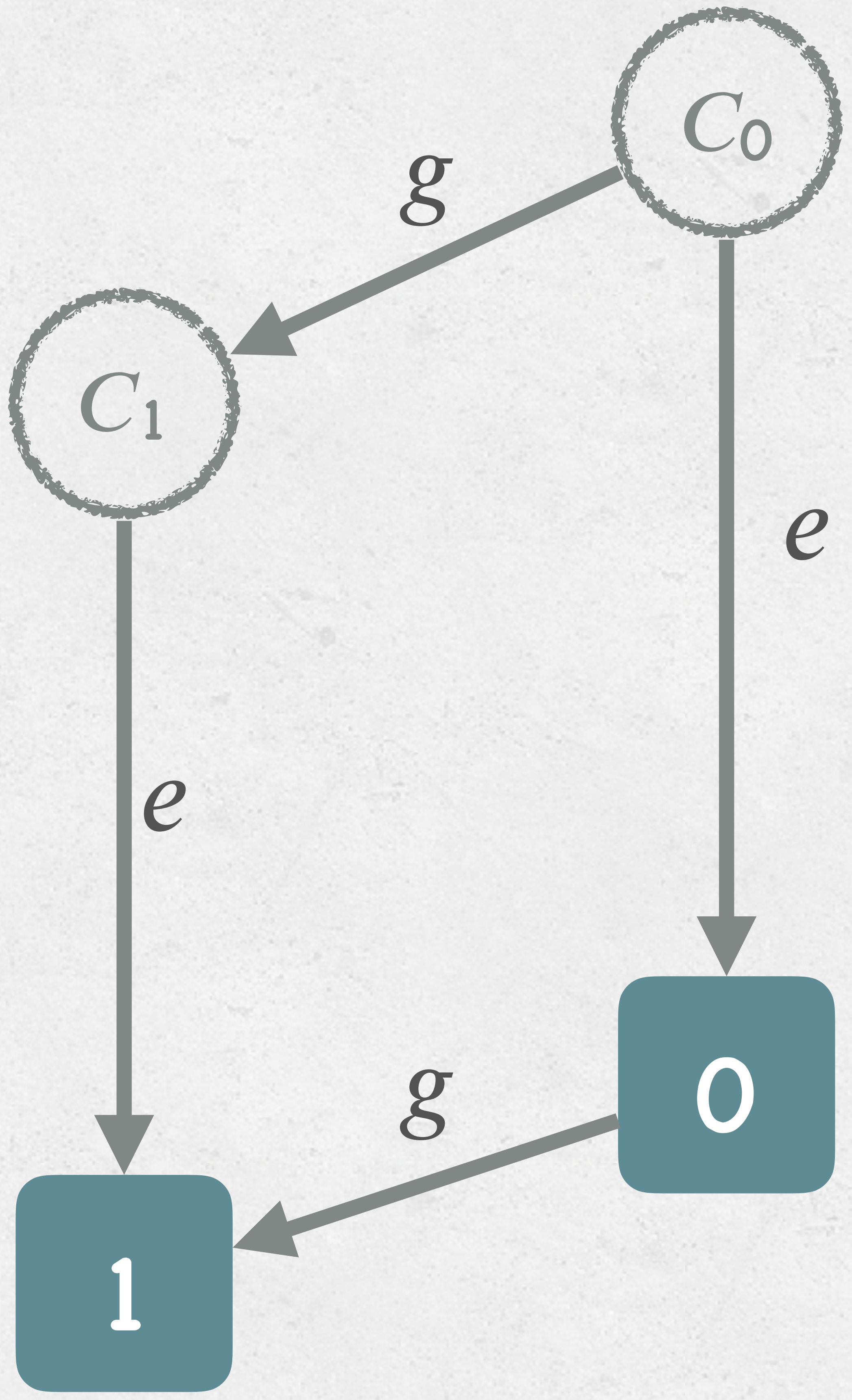
If not, g is applicable to $e(C_0)$ and results in a 1-valent configuration (Lemma 1).



Proving the Delay Lemma

Almost done! First, we will show that the processes taking steps e and g must be the same process.

If not, g is applicable to $e(C_0)$ and results in a 1-valent configuration (Lemma 1).

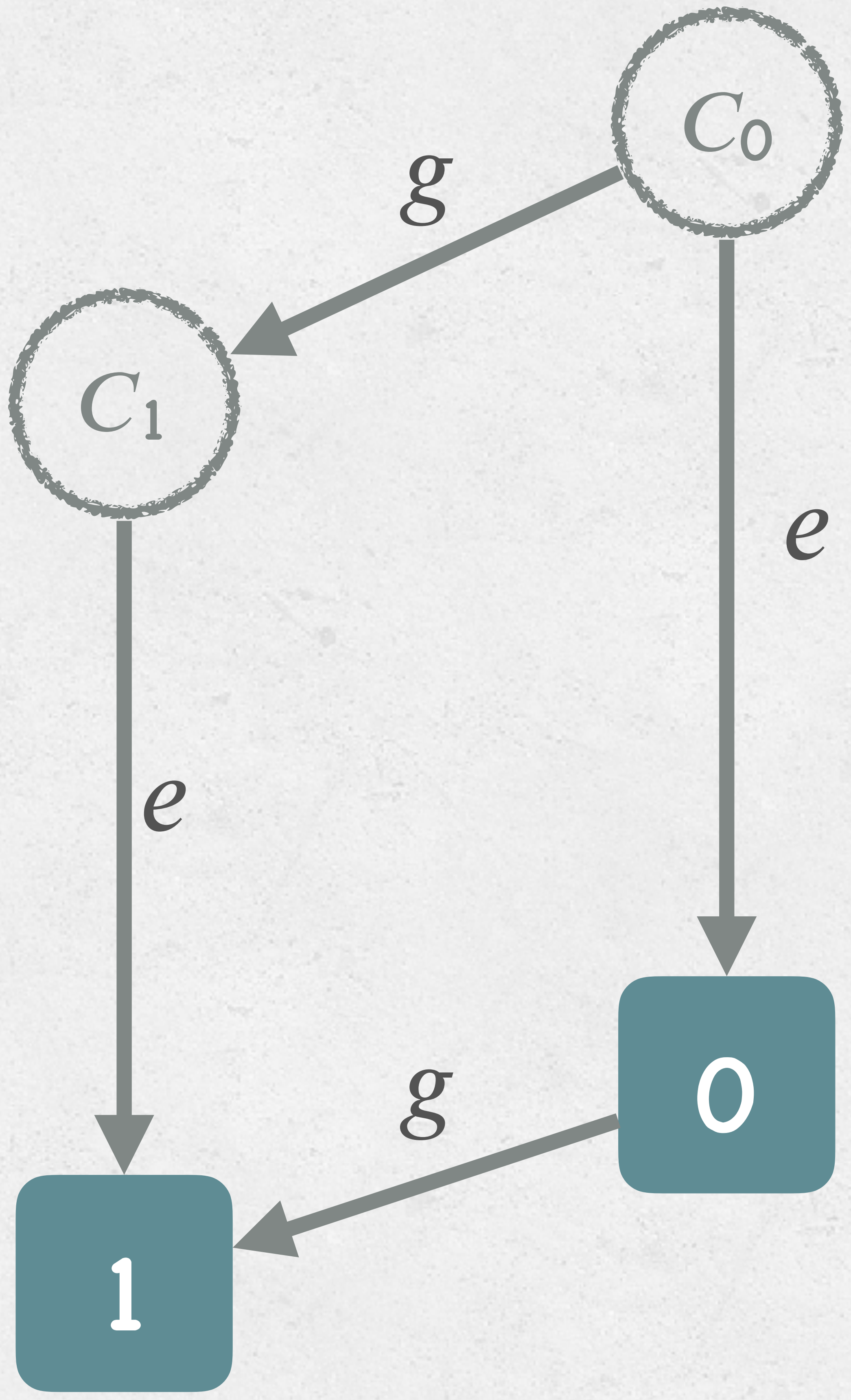


Proving the Delay Lemma

Almost done! First, we will show that the processes taking steps e and g must be the same process.

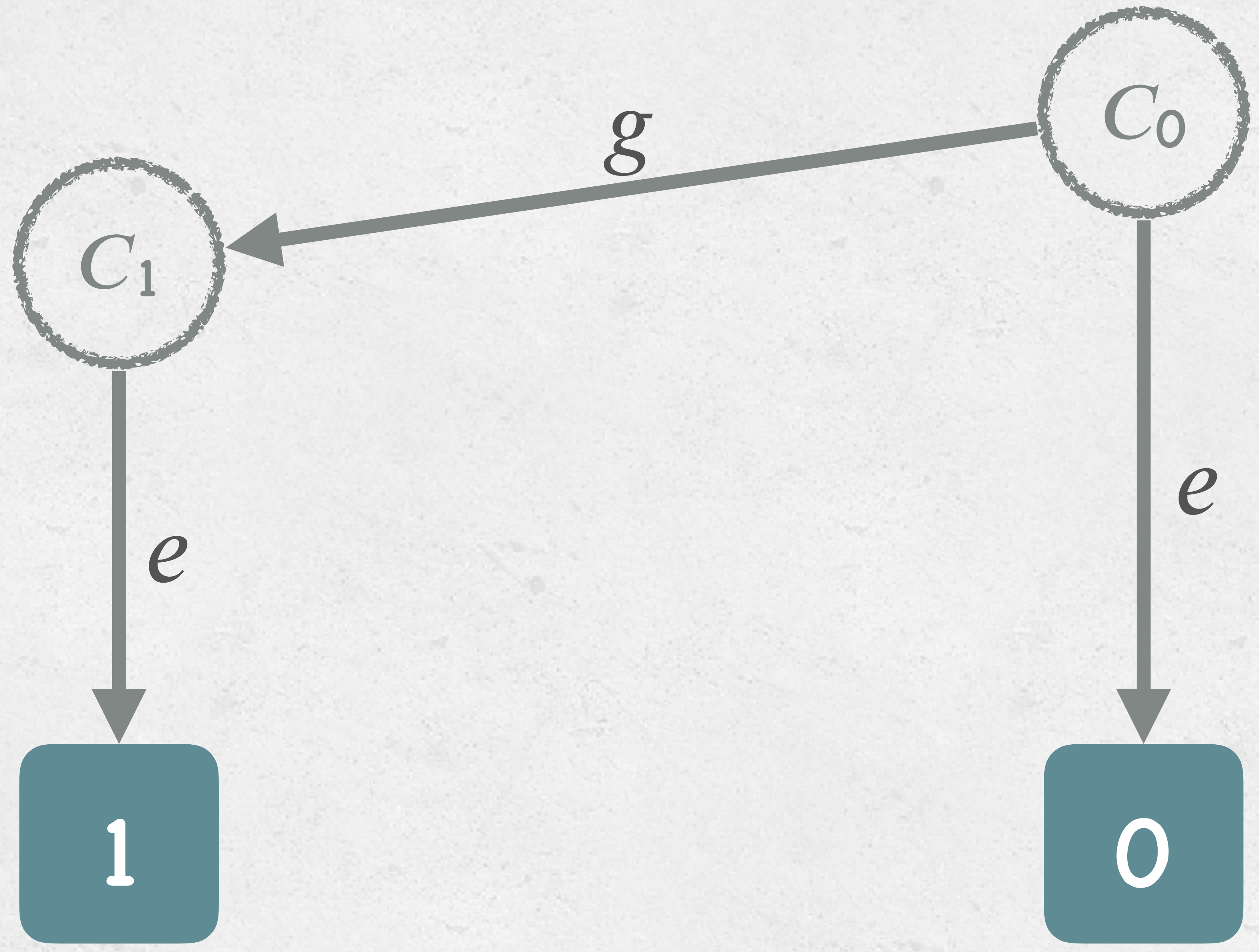
If not, g is applicable to $e(C_0)$ and results in a 1-valent configuration (Lemma 1).

Let's call the process taking these steps p.



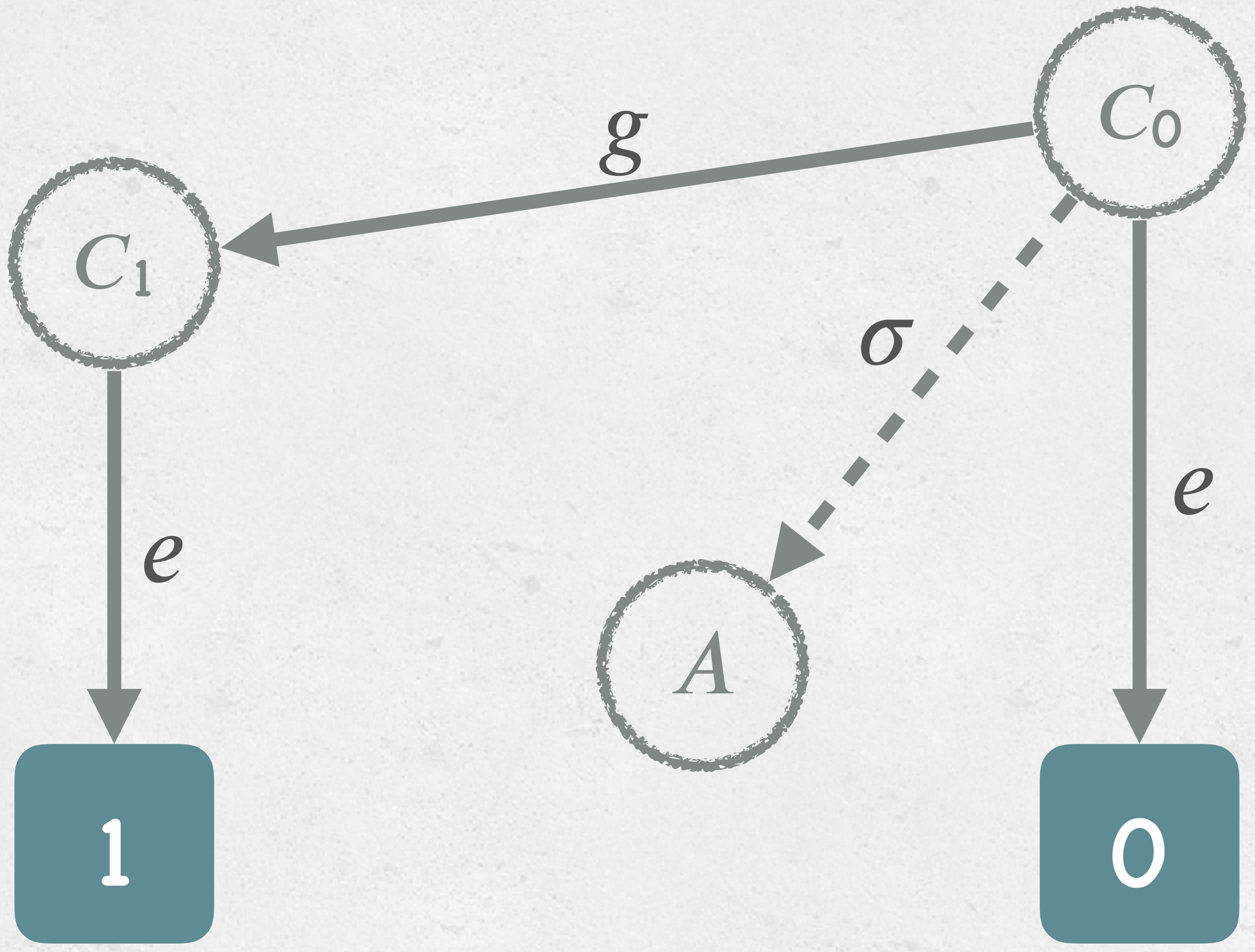
Proving the Delay Lemma

Since the protocol is correct and tolerates one failure, it must be able to reach a decided configuration, A, without p taking steps.



Proving the Delay Lemma

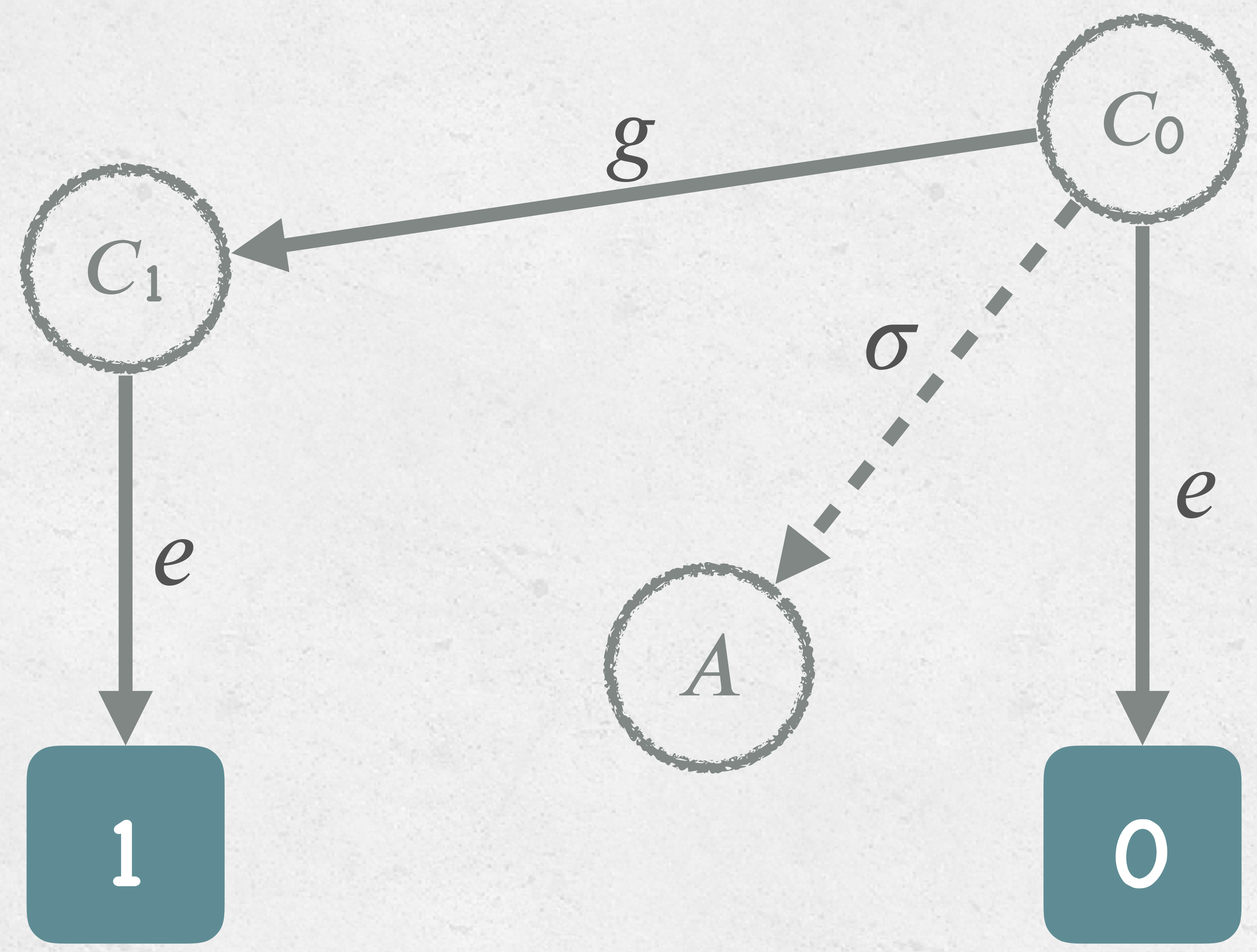
Since the protocol is correct and tolerates one failure, it must be able to reach a decided configuration, A, without p taking steps.



Proving the Delay Lemma

Since the protocol is correct and tolerates one failure, it must be able to reach a decided configuration, A, without p taking steps.

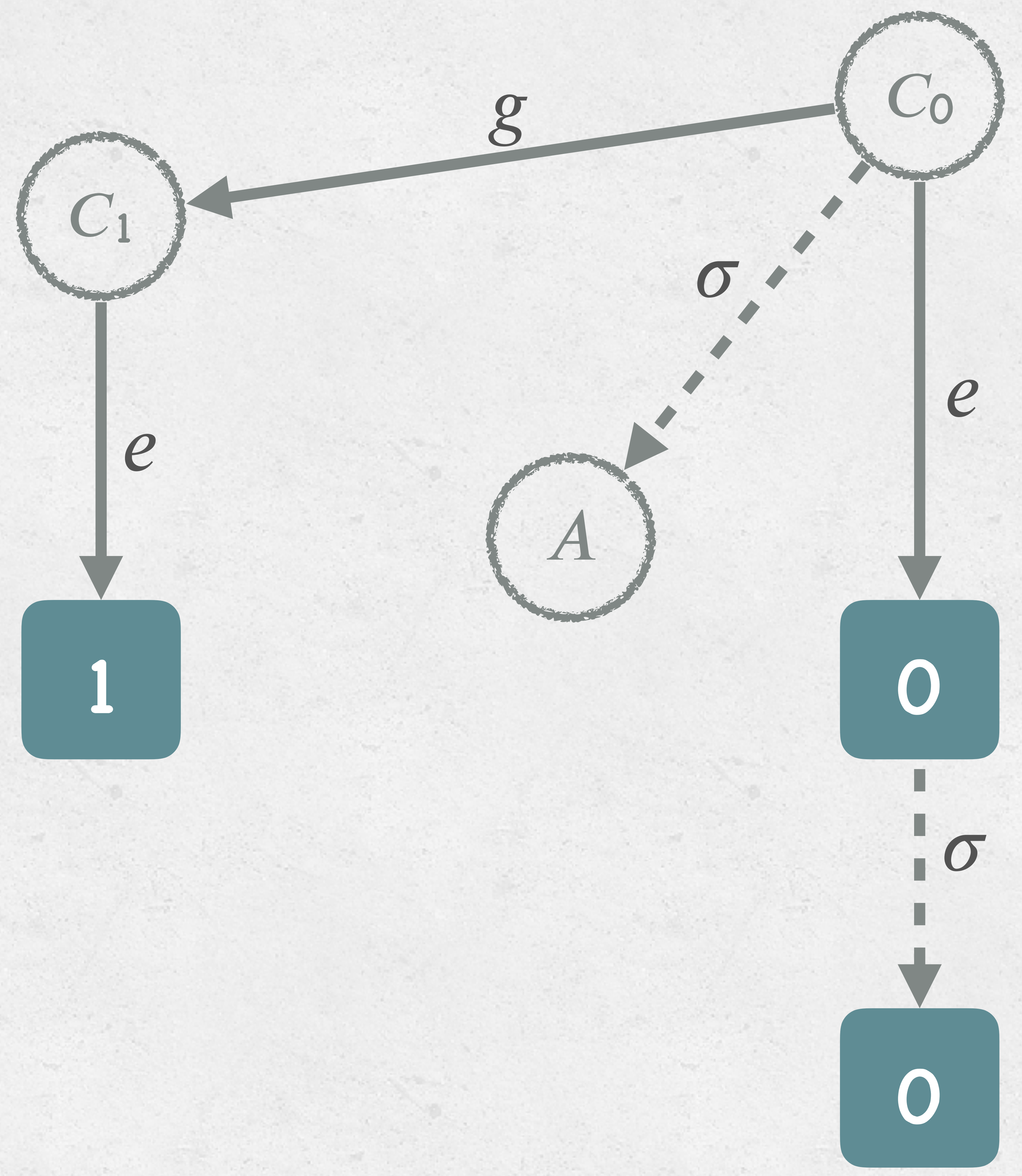
By Lemma 1, we get the commutative diagram on the right. A decided configuration, A, can reach both 1-valent and 0-valent configurations.



Proving the Delay Lemma

Since the protocol is correct and tolerates one failure, it must be able to reach a decided configuration, A, without p taking steps.

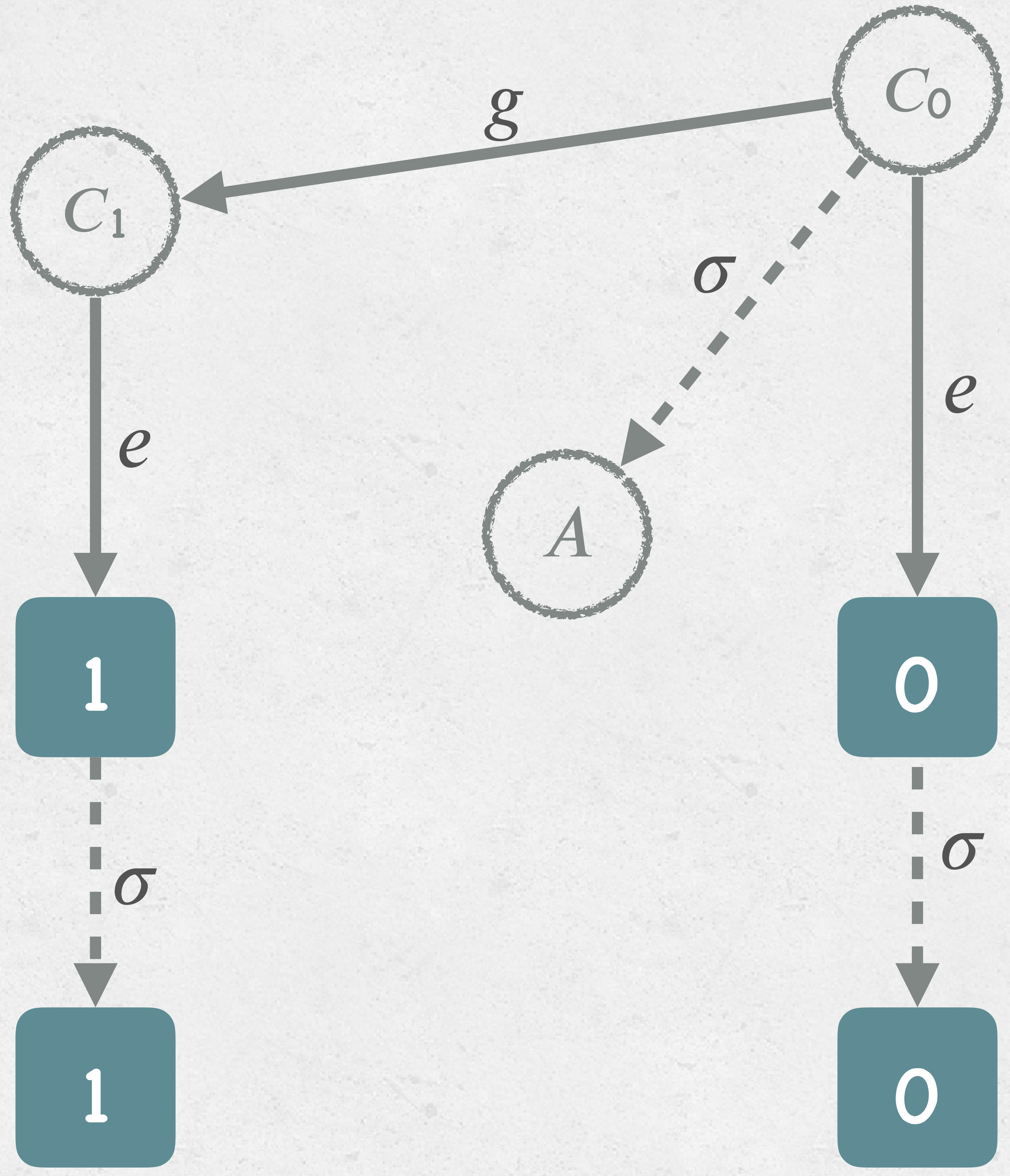
By Lemma 1, we get the commutative diagram on the right. A decided configuration, A, can reach both 1-valent and 0-valent configurations.



Proving the Delay Lemma

Since the protocol is correct and tolerates one failure, it must be able to reach a decided configuration, A, without p taking steps.

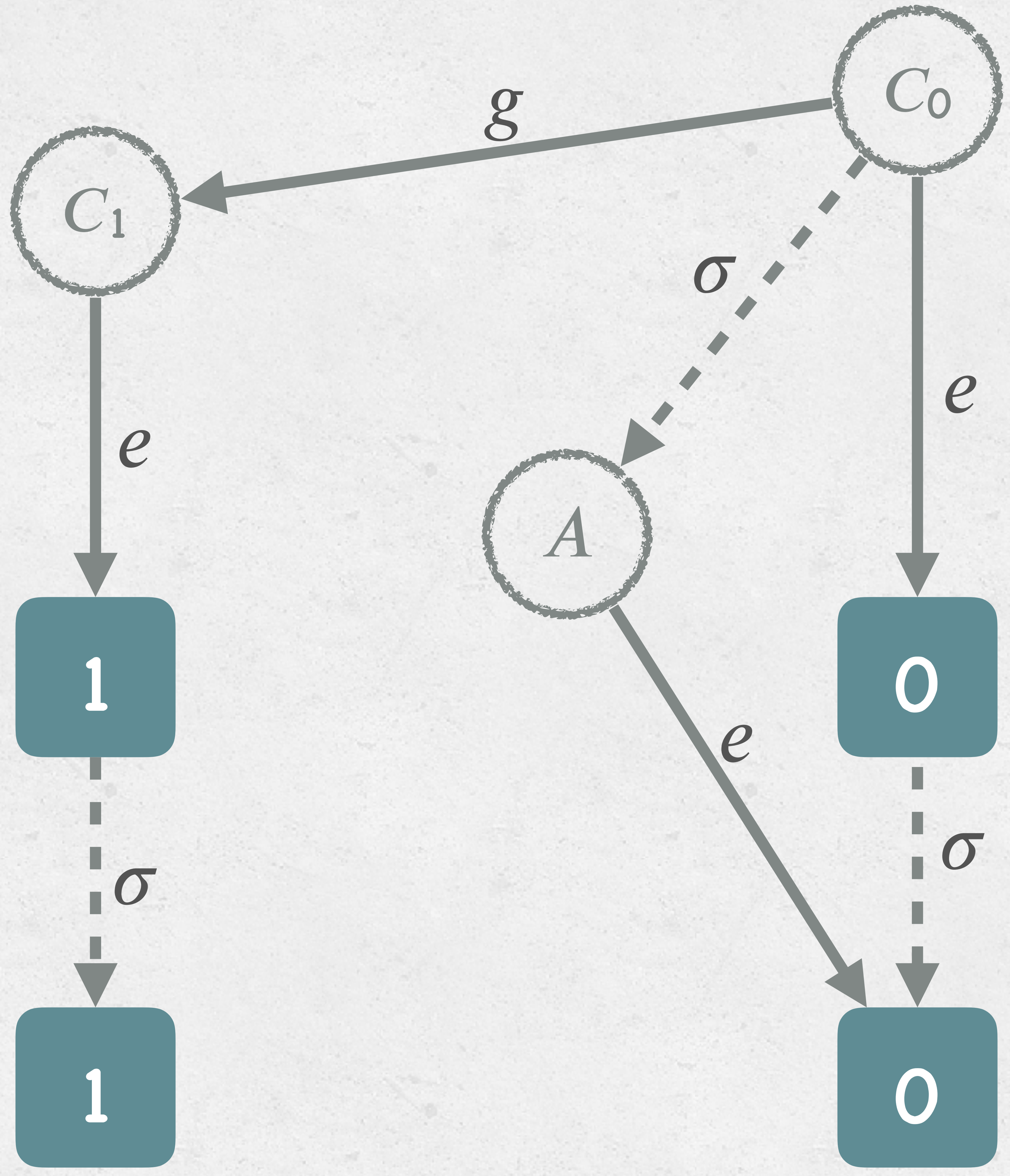
By Lemma 1, we get the commutative diagram on the right. A decided configuration, A, can reach both 1-valent and 0-valent configurations.



Proving the Delay Lemma

Since the protocol is correct and tolerates one failure, it must be able to reach a decided configuration, A, without p taking steps.

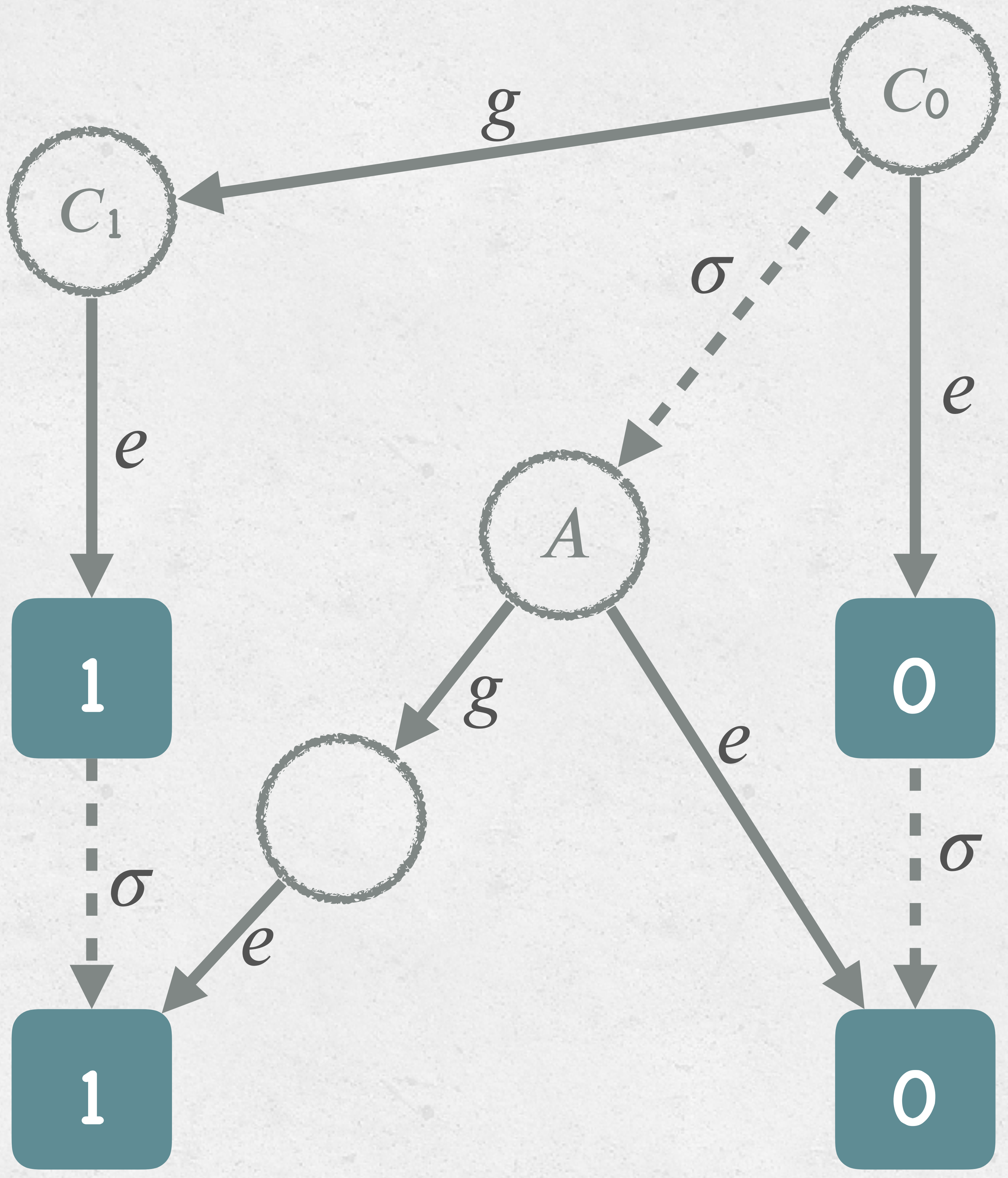
By Lemma 1, we get the commutative diagram on the right. A decided configuration, A, can reach both 1-valent and 0-valent configurations.



Proving the Delay Lemma

Since the protocol is correct and tolerates one failure, it must be able to reach a decided configuration, A, without p taking steps.

By Lemma 1, we get the commutative diagram on the right. A decided configuration, A, can reach both 1-valent and 0-valent configurations.

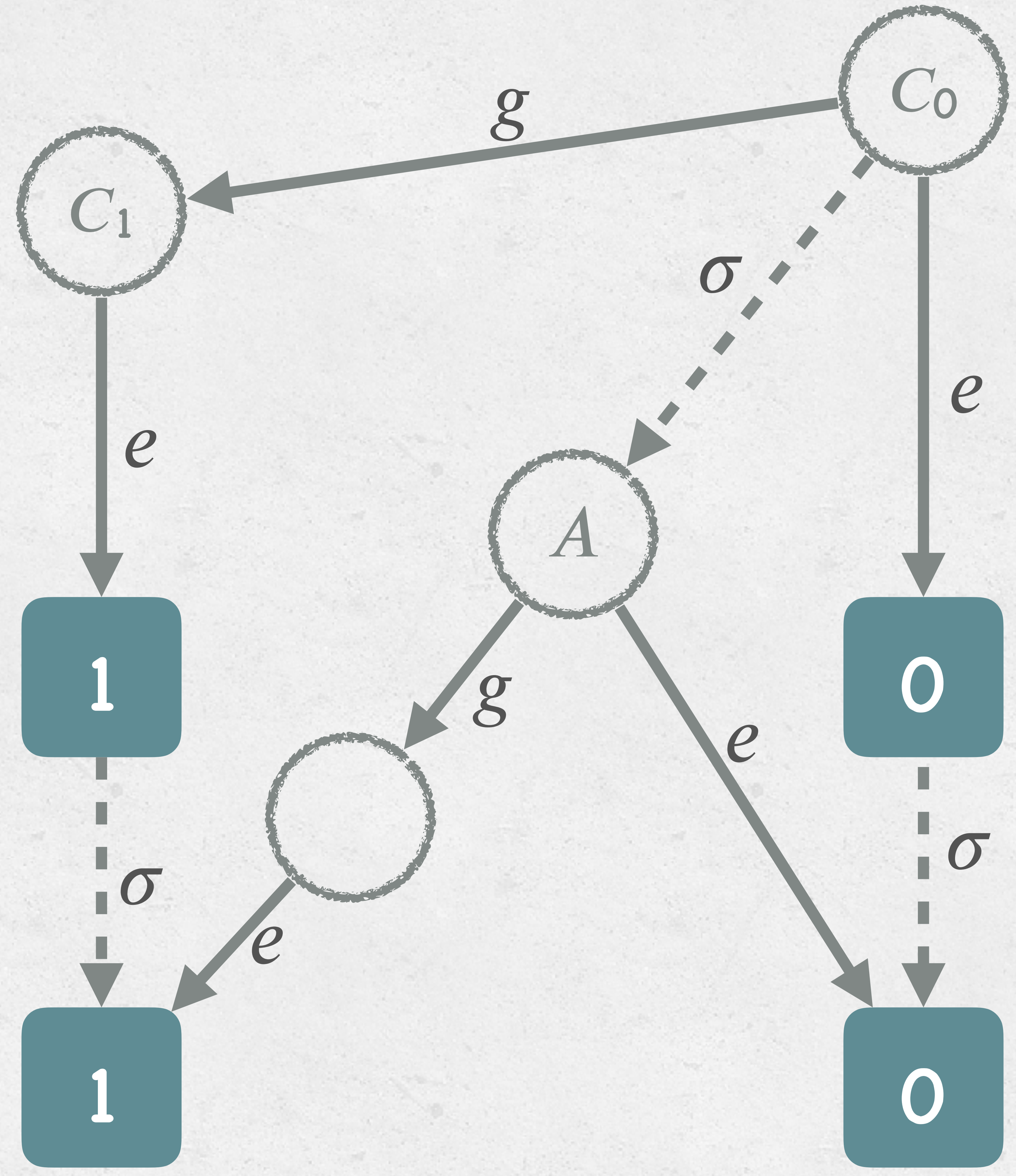


PROVING THE DELAY LEMMA

Since the protocol is correct and tolerates one failure, it must be able to reach a decided configuration, A, without p taking steps.

By Lemma 1, we get the commutative diagram on the right. A decided configuration, A, can reach both 1-valent and 0-valent configurations.

As desired, contradiction!



Proving the Delay Lemma

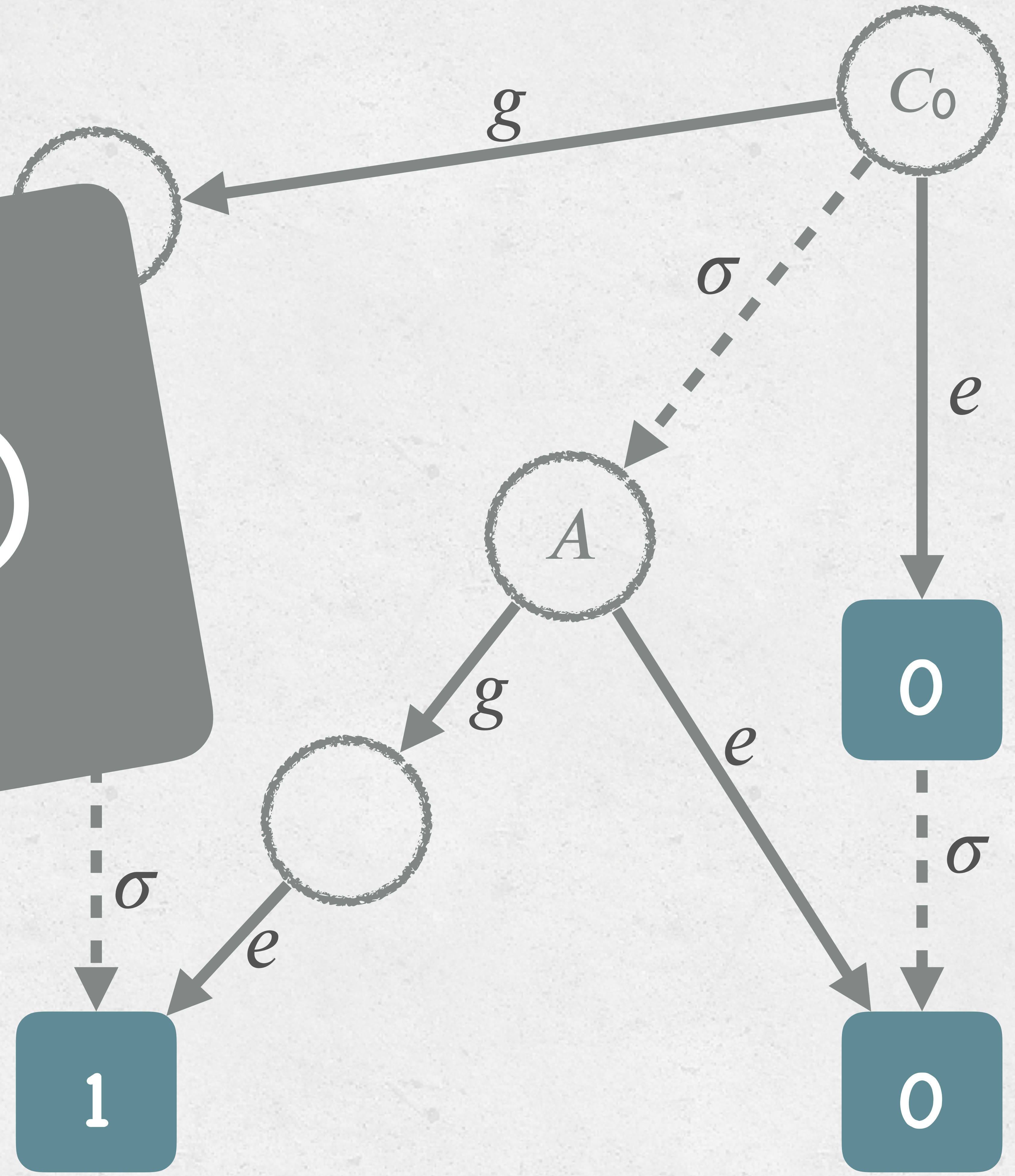
Since the protocol is correct and tolerates one failure, it must be able to reach a decided configuration, A, with

taking steps.

By Lemma 1, we get the commu
diagram on the right. A decided
configuration, A, can reach both
and 0-valent configurations.

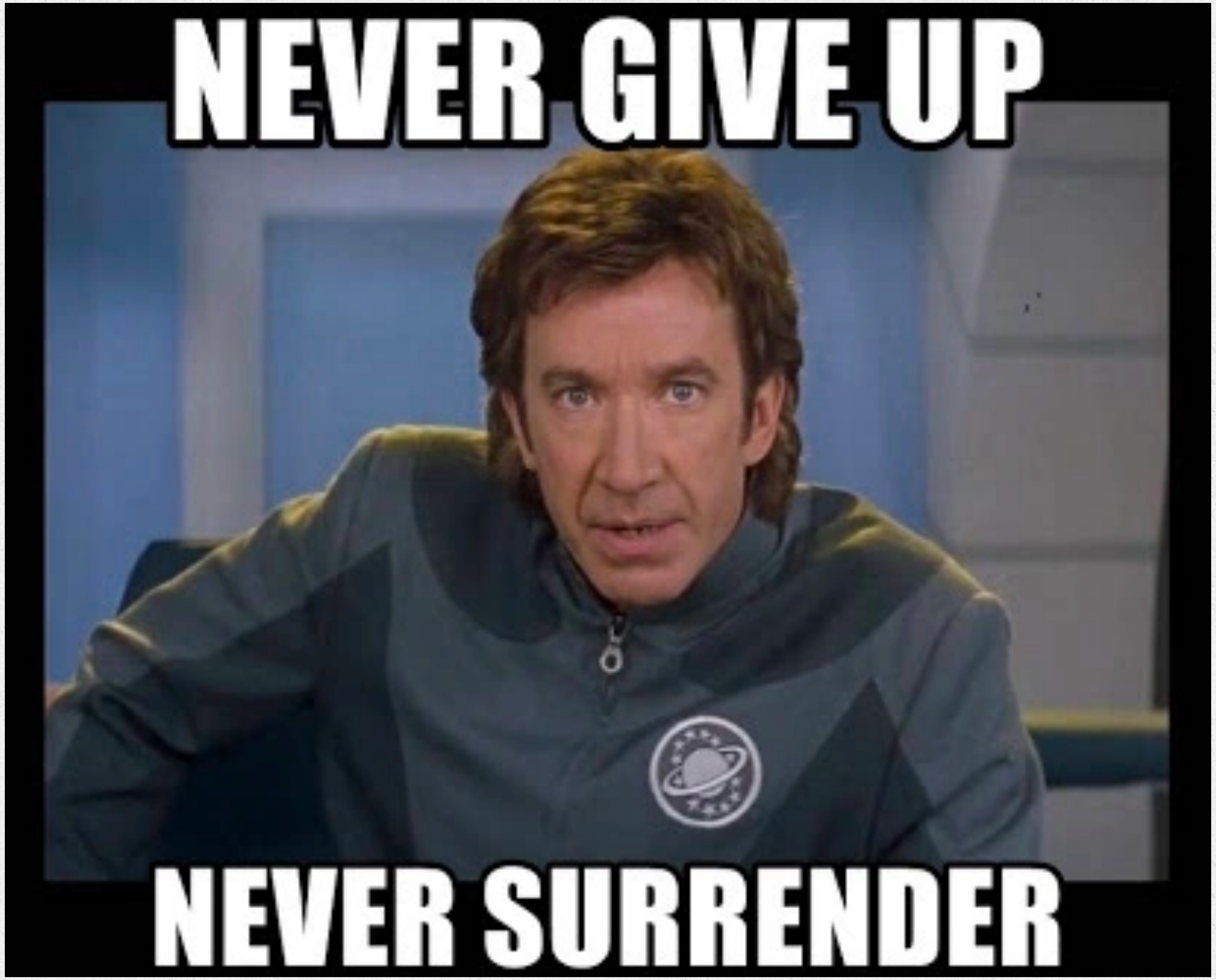
As desired, contradiction!

QED



Is It Over? Do We Give Up Now?

Is It Over? Do We Give Up Now?



Is It Over? Do We Give Up Now?

Options:

- Only guarantee termination during periods of synchrony (**Paxos**); implies that no configuration is ever dead
- Use randomization to guarantee termination with probability 1 (**Ben-Or**)
- Strengthen the assumptions (consensus is solvable in a synchronous system)
- Constrain/weaken the problem

Some Related Problems

- ***k*-set Agreement:** allows up to k different decision values
- **Generalized Lattice Agreement:** processes decide on sets of values, all decision sets are comparable by $\subseteq$
- **Shared read/write register:** processes can read and write to a register

Some Related Problems

Still can't guarantee liveness when $f \geq k$

- **k-set Agreement:** allows up to k different decision values

- **Generalized Lattice Agreement:** processes decide on sets of values, all decision sets are comparable by $\subseteq$

- **Shared read/write register:** processes can read and write to a register

Some Related Problems

Still can't guarantee liveness when $f \geq k$

Solvable, can guarantee both safety and liveness! Of questionable utility.

- **...t:** allows up to k different

- **Generalized Lattice Agreement:** processes decide on sets of values, all decision sets are comparable by $\subseteq$

- **Shared read/write register:** processes can read and write to a register

Some Related Problems

Still can't guarantee liveness when $f \geq k$

Solvable, can guarantee both safety and liveness! Of questionable utility.

- **…t:** allows up to k different

- **Generalized Lattice Agreement:** processes decide on sets of values, all decision sets are comparable by $\subseteq$

Also solvable! And useful!

- **Shared read/write register:** processes can read and write to a register