Byzantine Generals Problem II & FLP Impossibility

August 28, 2019

Recap

- Conditions to define correct behavior
  1. Any two loyal generals use the same value of $v(i)$. (Regardless of i loyal or traitor)
  2. If the ith general is loyal, then the value that he sends must be used by every loyal general as the value of $v(i)$.

- No solution with fewer than $3m+1$ nodes can cope with m malicious nodes if simple messages are transmitted
- If messages can be signed, a solution for $m+2$ generals exist with m traitors
  - This requires knowledge of public keys and timeouts

Byzantine Generals Problem with Signatures

- Solution for m traitors and any number of generals
  - nonsensical/trivial for <m+2 generals
    - only one loyal node, every other node is a traitor

Byzantine Generals Problem with Signatures

- notation
  - $m{:}i$ message m signed by general i
  - $m{:}i{:}j{:}k$
    - message m signed by general i
      - statement "$m{:}i$" signed by j
        - statement "$m{:}i{:}j$" signed by k

- requires function $choice()$
- selects an order (attack, retreat) from a set of orders V
  - if |V|=1, $choice(V) =$ element in V
  - if |V|=0, $choice(V) = RETREAT$

Algorithm *SM(m)* (>m+2 generals)

Initially $V_i = \emptyset$.

(1) The commander signs and sends his value to every lieutenant.
(2) For each i:

(A) If Lieutenant i receives a message of the form $v:0$ from the commander and he has not yet received any order, then

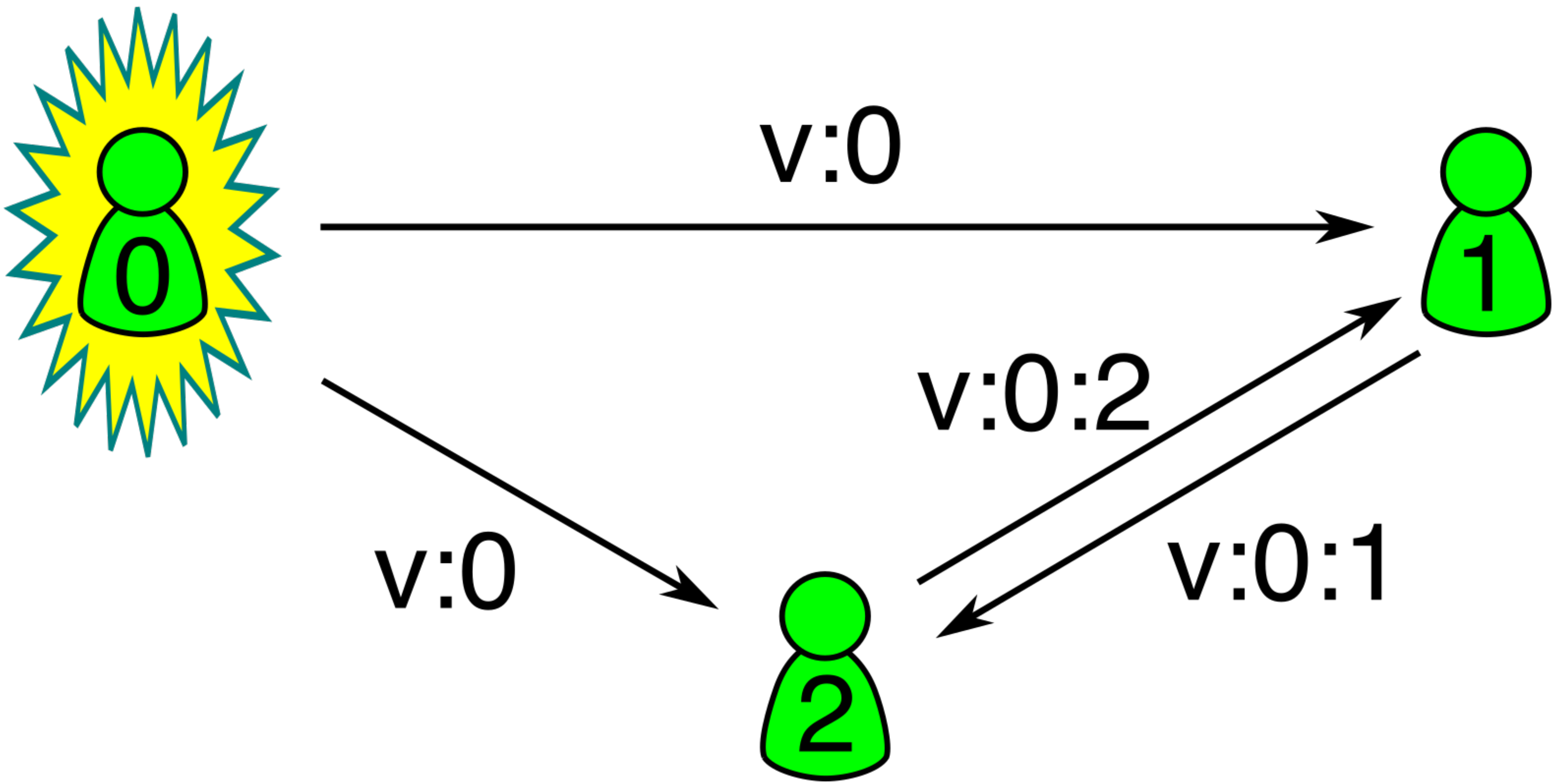
(i) he lets V_i equal $\{v\}$;
(ii) he sends the message $v:0:i$ to every other lieutenant.

(B) If Lieutenant i receives a message of the form $v:0:j_1:\cdots:j_k$ and v is not in the set V_i, then

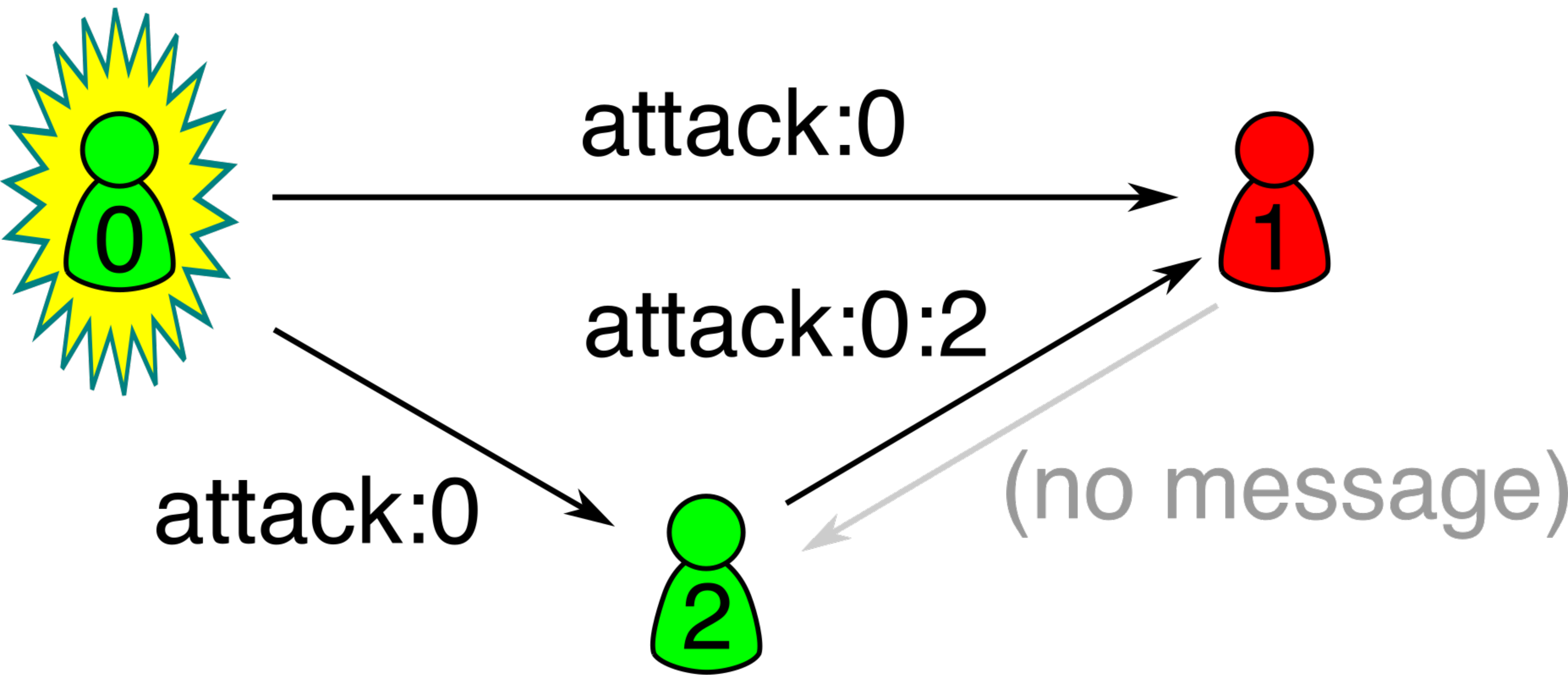
(i) he adds v to V_i;
(ii) if $k < m$, then he sends the message $v:0:j_1:\cdots:j_k:i$ to every lieutenant other than $j_1, \ldots, j_k$.

(3) For each i: When Lieutenant i will receive no more messages, he obeys the order $choice(V_i)$.

Algorithm *SM(m)* (3 generals)

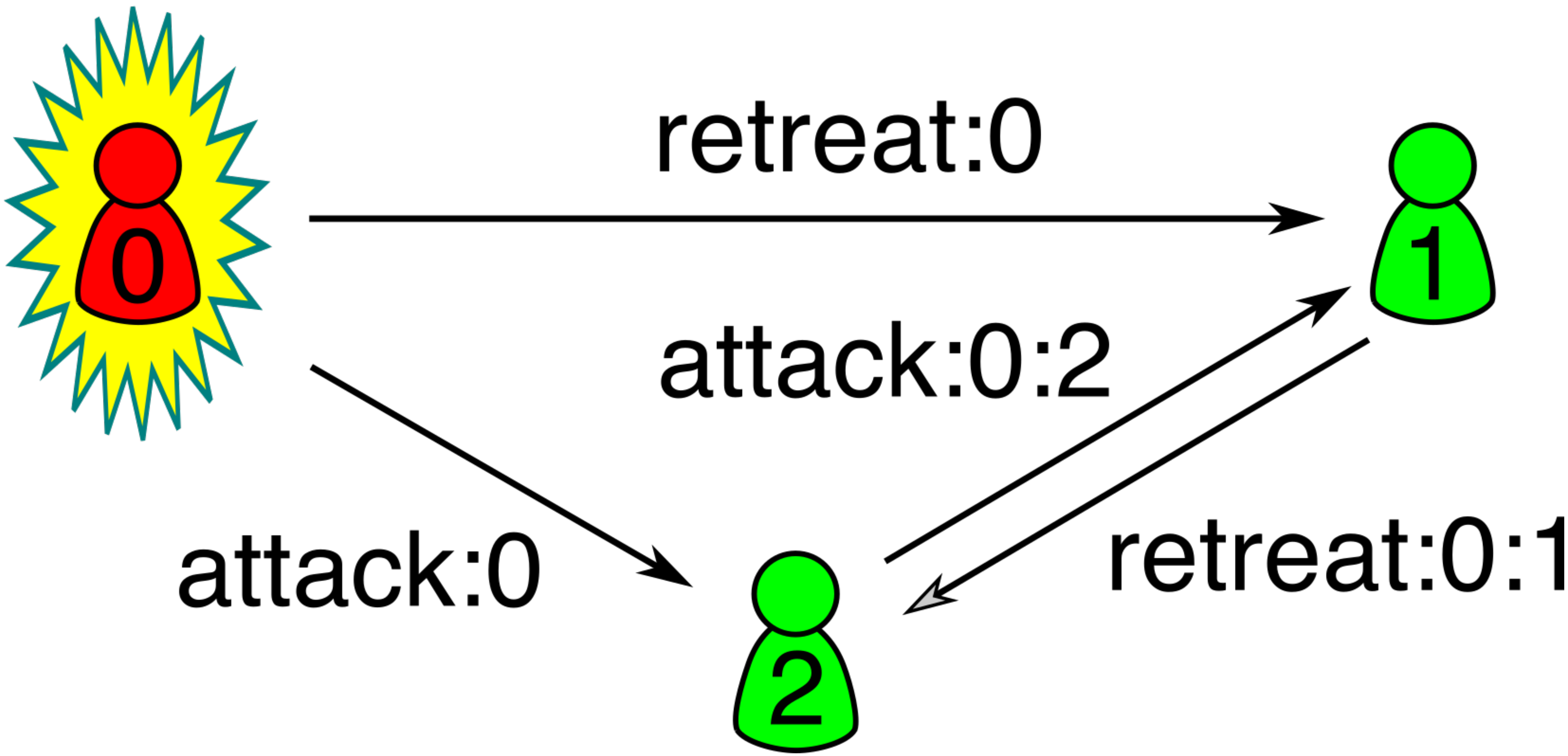


Algorithm *SM(m)* (3 generals, 1 traitor)



Loyal Lieutenant 2 always follows the order

Algorithm *SM(m)* (3 generals, 1 traitor)



Both loyal lieutenants follows the order *choice({attack, retreat})*

Algorithm *SM(m)* (3 generals, 1 traitor)

General:
"attack":0 to L1
"retreat":0 to L2

	order set V
L1	{"attack"}
L2	{"retreat"}

Algorithm *SM(m)* (3 generals, 1 traitor)

L1
“attack”:0:1 to L2

	order set V
L1	{“attack”}
L2	{“retreat”,”attack”}

Algorithm *SM(m)* (3 generals, 1 traitor)

L2
“retreat”:0:2 to L1

	order set V
L1	{“attack”,”retreat”}
L2	{“retreat”,”attack”}

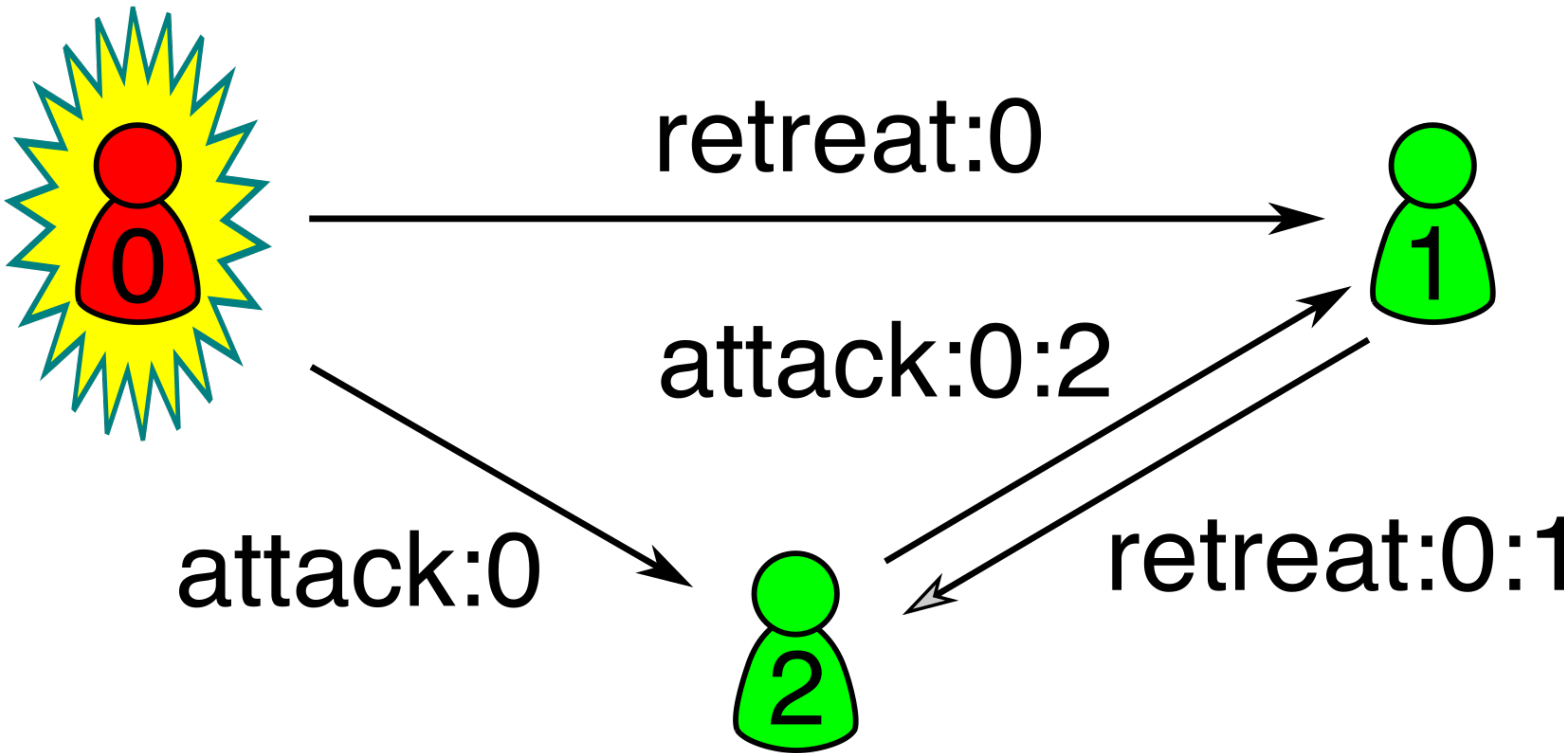
Algorithm *SM(m)* (3 generals, 1 traitor)

(3) **For each i: When Lieutenant i will receive no more messages, he obeys the order $choice(V_i)$.**

	order set V
L1	{“attack”,”retreat”}
L2	{“retreat”,”attack”}

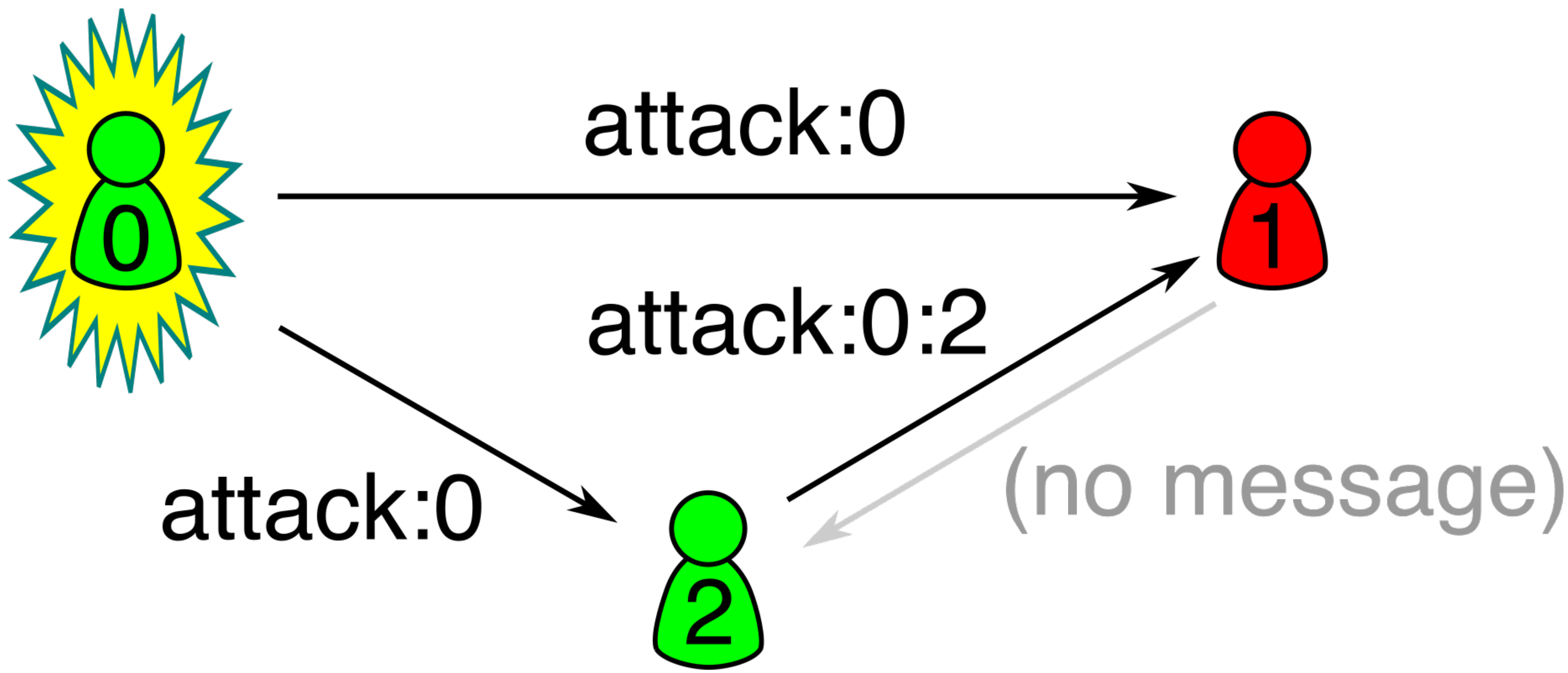
Both loyal lieutenants follows the order *choice({attack, retreat})*

Algorithm *SM(m)* (3 generals, 1 traitor)



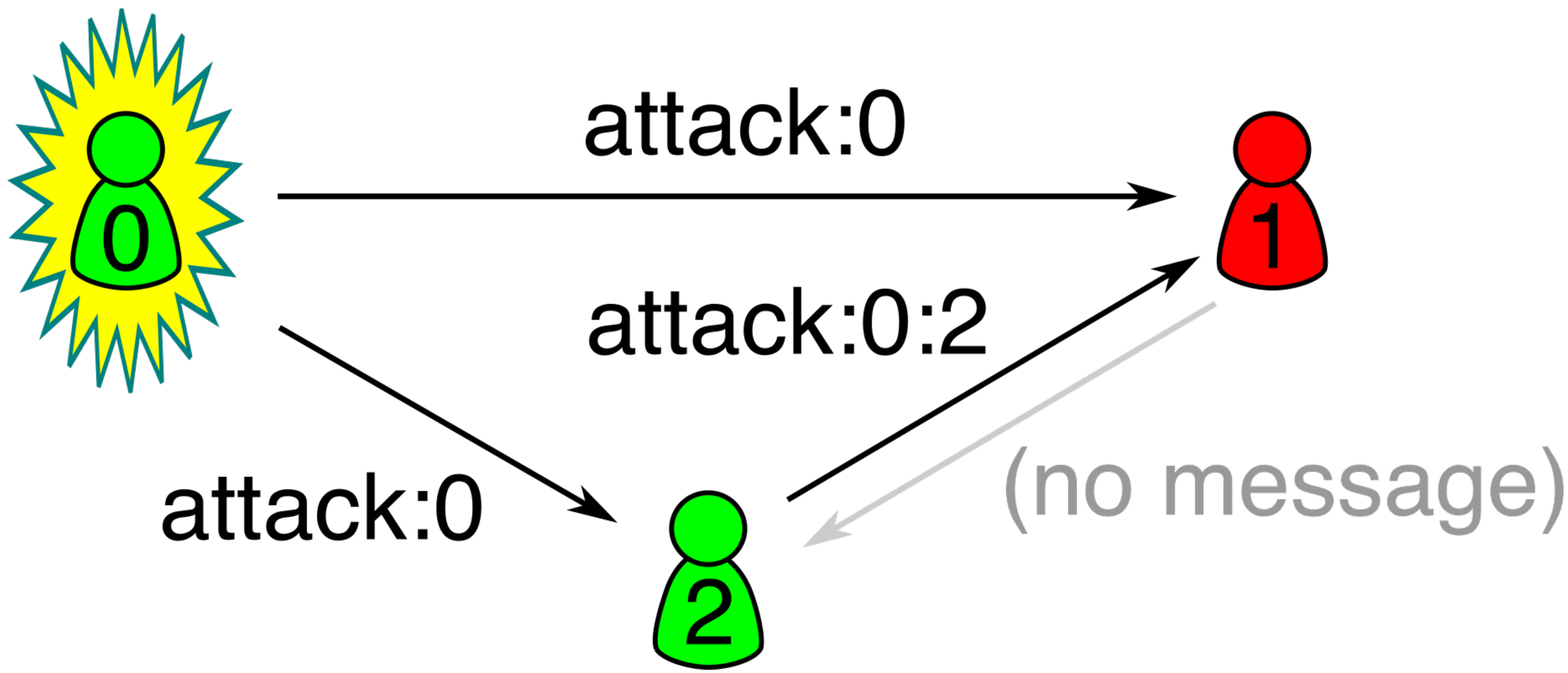
Both loyal lieutenants follows the order *choice({attack, retreat})*

When to execute order



- How does Lieutenant 2 know that 1 does not send a message (as opposed to delayed message)

When to execute order



- How does Lieutenant 2 know that 1 does not send a message (as opposed to delayed message)
  - Maybe timeout … ???

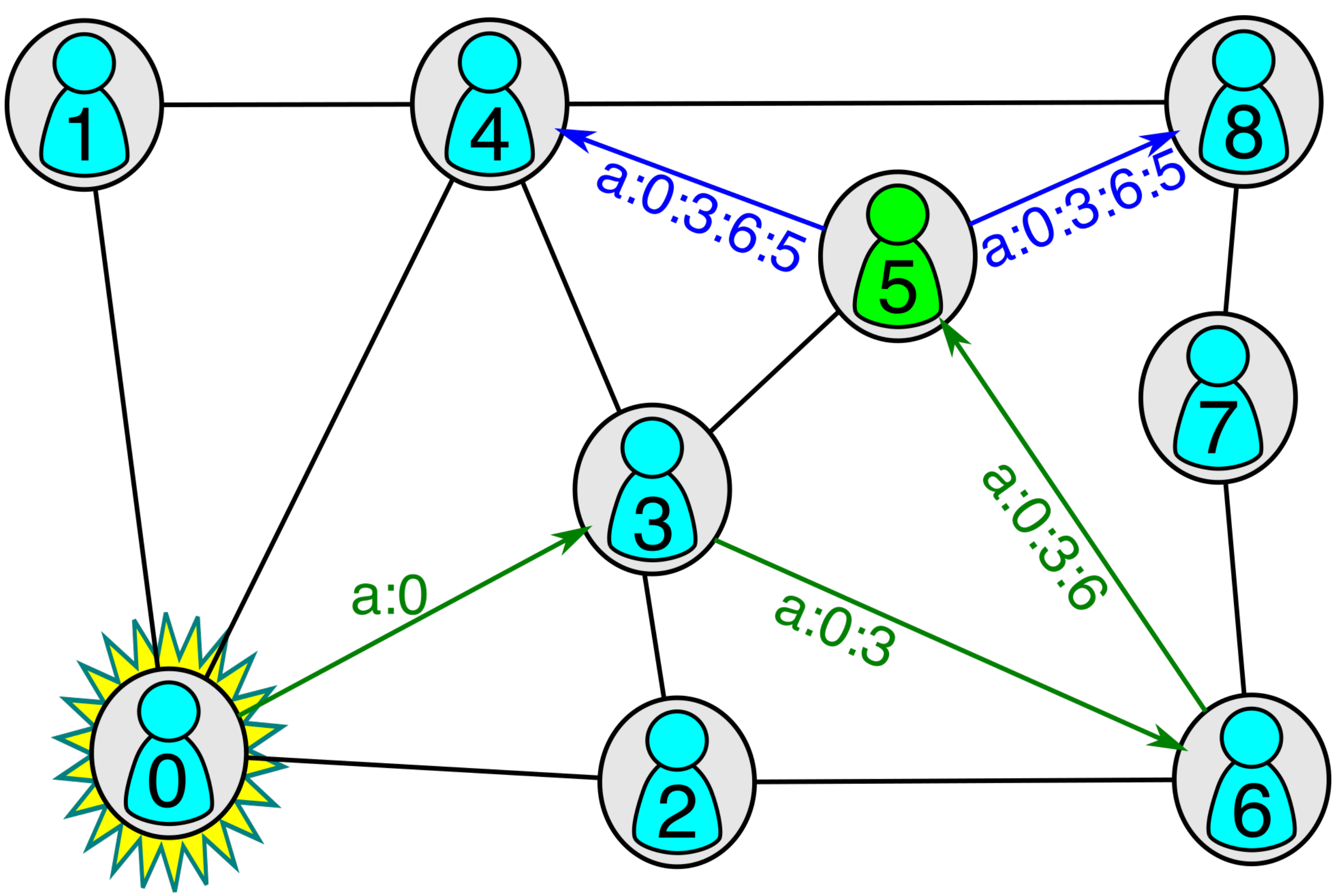
Missing communication paths

- So far, we considered fully connected graphs only
  - What happens, if each node only has some neighbors?

Missing communication paths

- Similar algorithm: Relay message to all neighbors that are not in the signature chain

- $SM(n\text{-}2)$ is a solution for n generals, regardless of the number of traitors
  - Max. signature chain $v{:}0{:}j_1{:}\ldots j_k$ has length $n\text{-}2$



if j_5 received

"$a{:}0{:}3{:}6$",

send

"$a{:}0{:}3{:}6{:}5$"

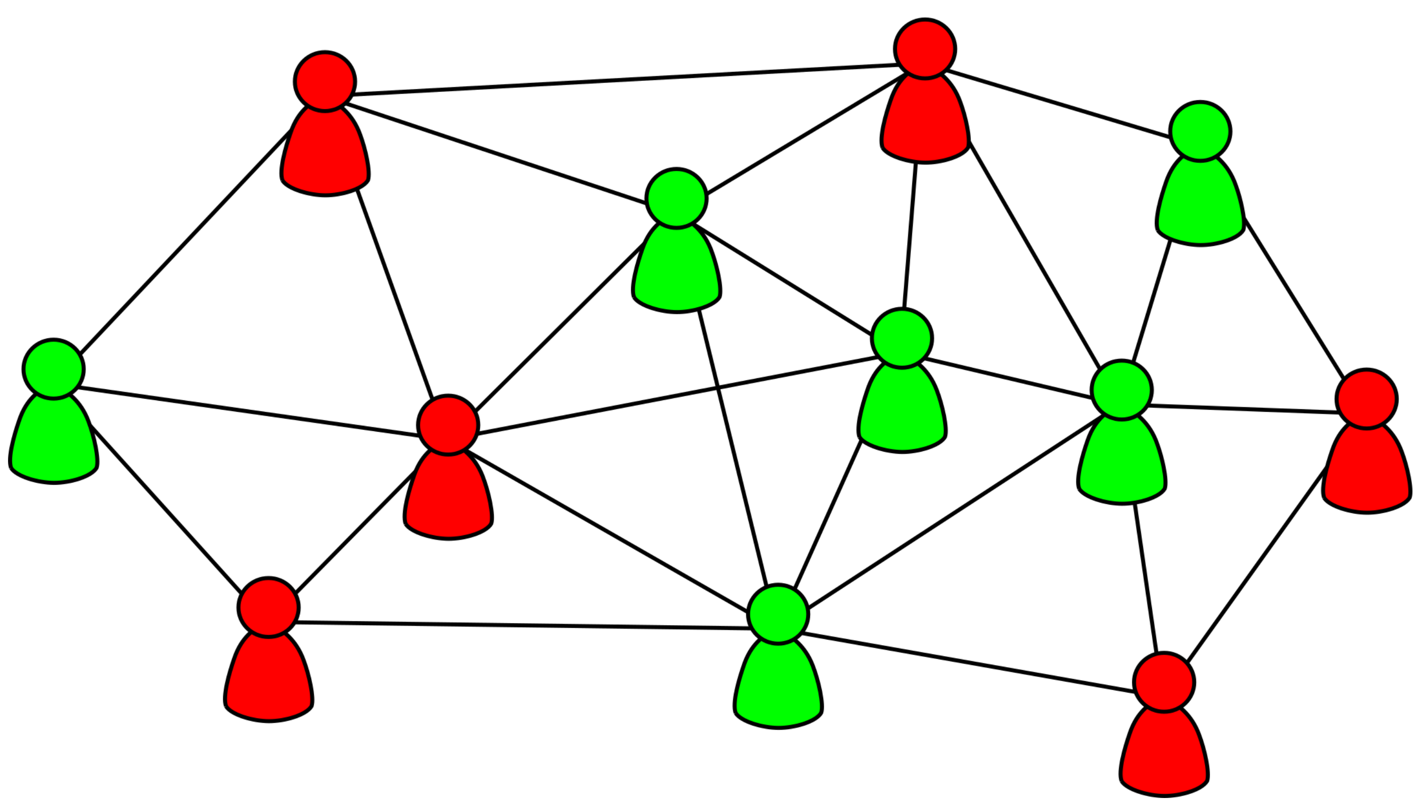
to LT 4 and 8

Missing communication paths

- Assume all loyal generals form a connected subgraph
  - Otherwise only the largest connected subgraph of loyal generals is relevant

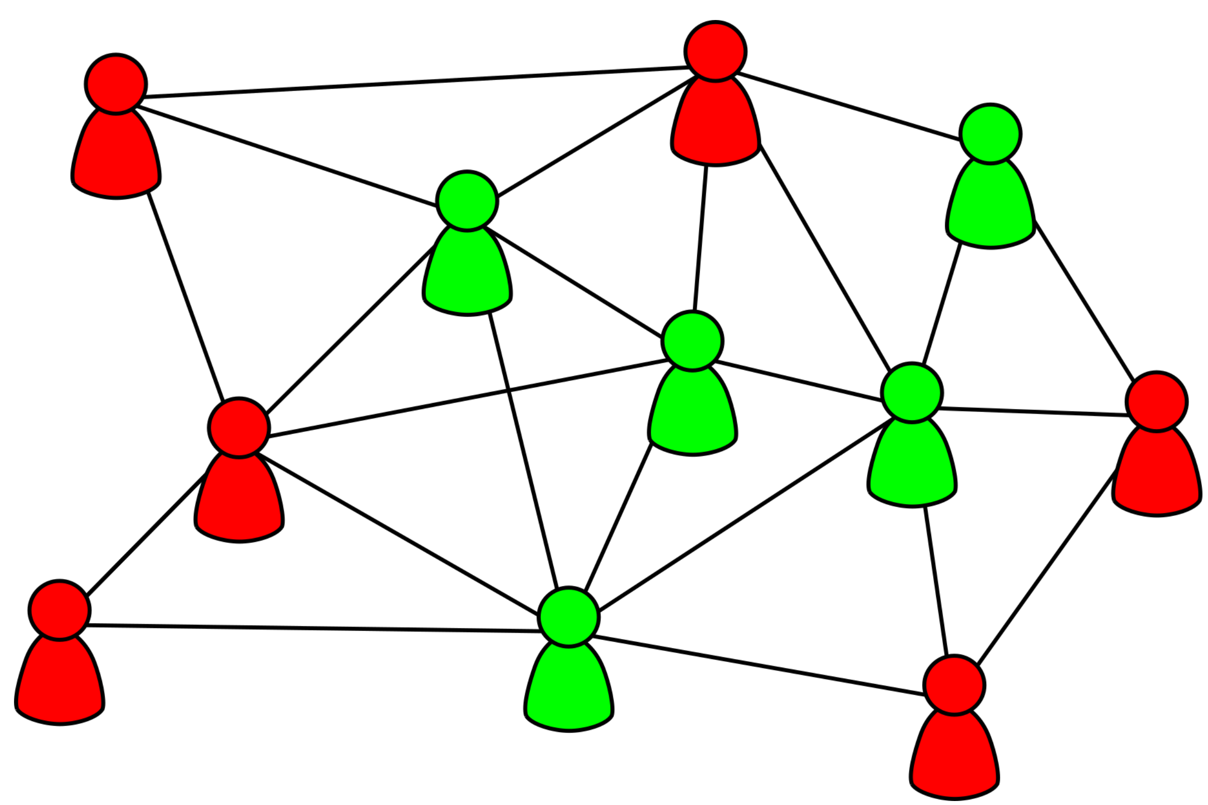
Missing communication paths

- Assume all loyal generals form a connected subgraph
  - Otherwise only the largest connected subgraph of loyal generals is relevant



Missing communication paths

- Assume all loyal generals form a connected subgraph
  - Otherwise only the largest connected subgraph of loyal generals is relevant



Missing communication paths

- C2: If the ith general is loyal, then the value that he sends must be used by every loyal general as the value of $v(i)$.
    - There is a path from the loyal commander to a lieutenant going through d-1 or fewer loyal lieutenants. Those relay the message faithfully. => all loyal lieutenants receive the same value for $v(i)$.

Missing communication paths

- C1: *Any two loyal generals use the same value of $v(i)$. (Regardless of i loyal or traitor)*
    - If general is loyal, C1 is full-filled by same argument
        - There is a path from the loyal commander to a lieutenant going through d-1 or fewer loyal lieutenants. Those relay the message faithfully. => all loyal lieutenants receive the same value for $v(i)$.

Missing communication paths

- C1: *Any two loyal generals use the same value of $v(i)$. (Regardless of i loyal or traitor)*
  - If general is traitor: we show that any order received by lieutenant i is also received by lieutenant j.
    - Assume diameter of loyal subgraph is d,
      - Every loyal general is reached within d steps of reaching the first loyal general
      - $m \leq n\text{-}d$ traitors.
      - Algorithm proceeds in $n\text{-}2 \geq m+d\text{-}2$ rounds.
    - suppose received message is $v{:}0{:}j_1{:}\ldots{:}j_k$ but not signed by j_j
    - We can show that j_j is reached within $n\text{-}2$ total steps
      - if $k>m$: $k<m \leq n\text{-}d \Rightarrow k+(d\text{-}1) \leq n\text{-}1$
      - if $k \geq m$: at least one loyal general was in the signature chain already.

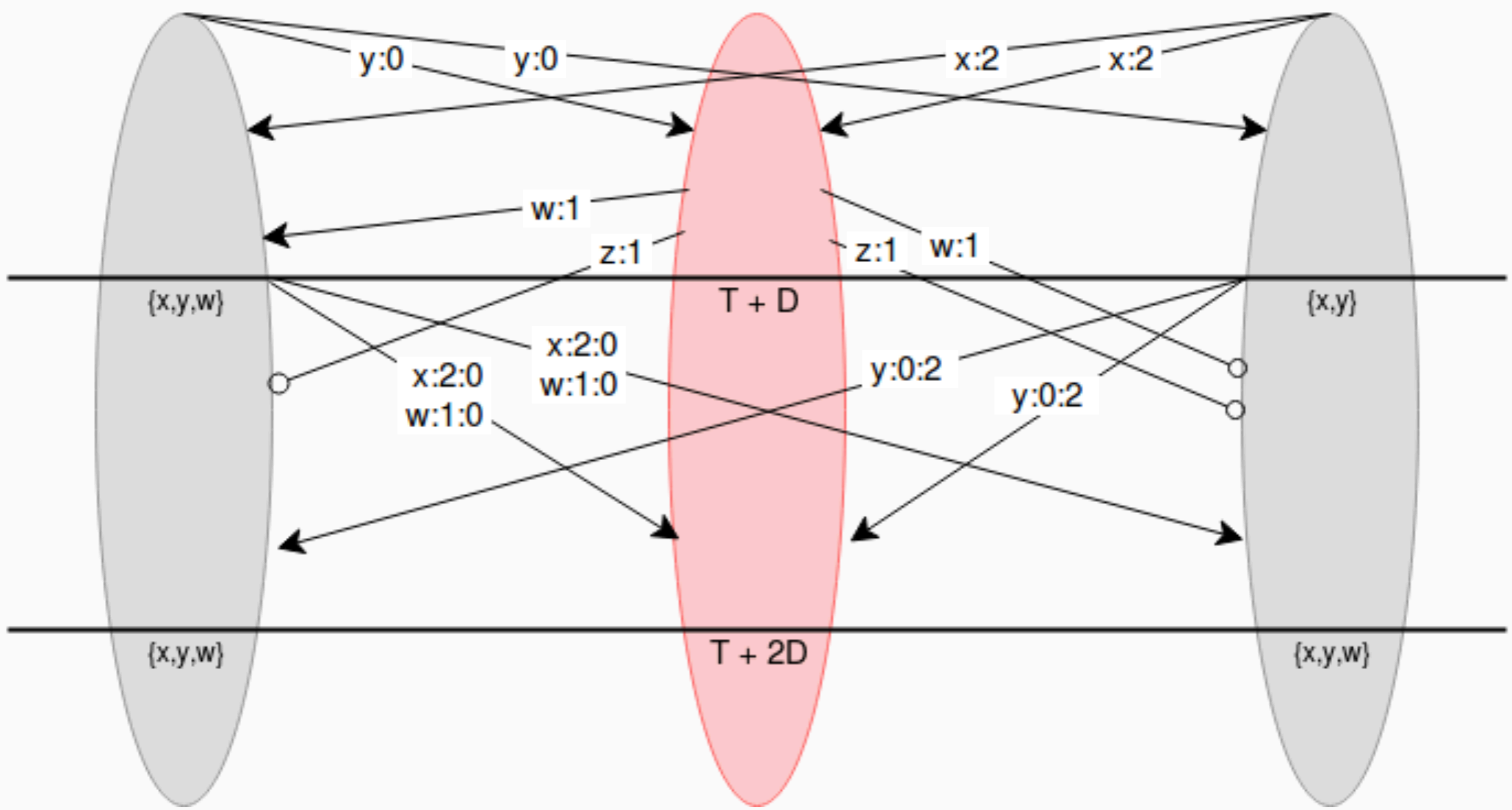
Missing communication paths

- C1: *Any two loyal generals use the same value of $v(i)$. (Regardless of i loyal or traitor)*
  - If general is traitor: we show that any order received by lieutenant i is also received by lieutenant j. Assume diameter of loyal subgraph is d, thus $m \leq n\text{-}d$ traitors.
    - suppose received message is $v{:}0{:}j_1{:}\ldots{:}j_k$ but not signed by j_j
    - $k<m$: j_i will send message to every neighbors and it will reach j_j within $d\text{-}1$ more steps. $k<m \leq n\text{-}d \Rightarrow k+(d\text{-}1) \leq n\text{-}1$
    - $k \geq m$: At least one of the signers must have been loyal, thus forwarding the message to all its neighbors, whereupon it will be relayed by loyal generals and will reach j_j within $d\text{-}1$ steps

Missing communication paths

- *SM(n-2)* is a solution for *n* generals, regardless of the number of traitors
  - (Algorithm *SM* for *n-2* rounds)
  - We can show
    - IC2: There is a path from the loyal commander to a lieutenant going through d-1 or fewer loyal lieutenants. Those relay the message faithfully
    - IC1: Any order received by lieutenant i is also received by lieutenant j, since the subgraph of loyal generals is smaller than *n-2*

Blockchain example

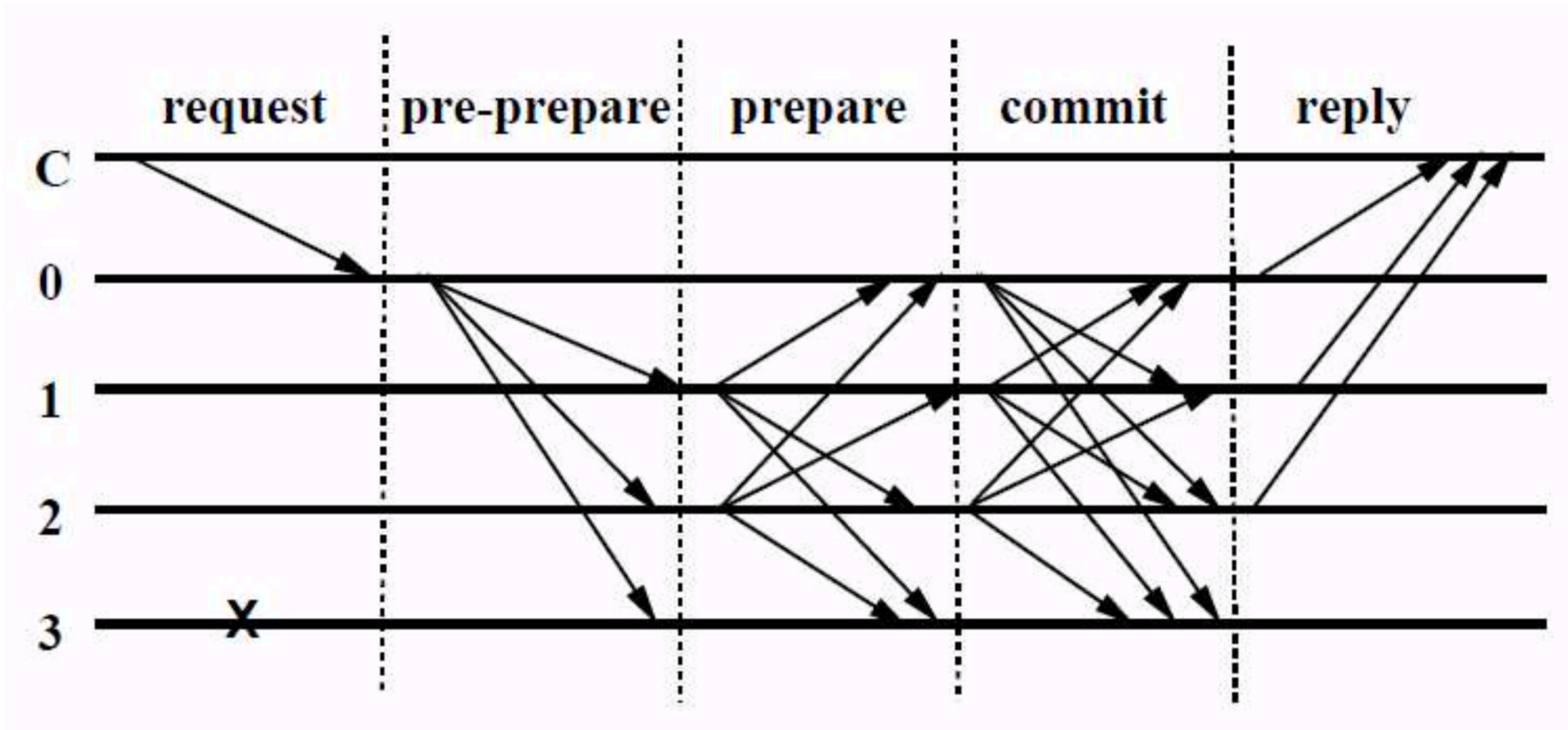


Node 1 (red) is malicious, and nodes 0 and 2 (grey) are honest. At the start, the two honest nodes make their proposals `y` *and* `x`*, and the attacker proposes both* `w` *and* `z` *late.* `w` *reaches node 0 on time but not node 2, and* `z` *reaches neither node on time. At time* `T + D`*, nodes 0 and 2 rebroadcast all values they've seen that they have not yet broadcasted, but add their signatures on (*`x` *and* `w` *for node 0,* `y` *for node 2). Both honest nodes saw* `{x, y, w}`*.*

Vitalik Buterin, https://vitalik.ca/general/2018/08/07/99_fault_tolerant.html

Byzantine Fault Tolerance in Databases

- An example



- Client C:
  - send request to primary (node 0)
  - Wait for (same) answer from m+1 machines
- If primary is faulty, select new primary

Distributed Consensus with Faulty Processes

FLP Statement

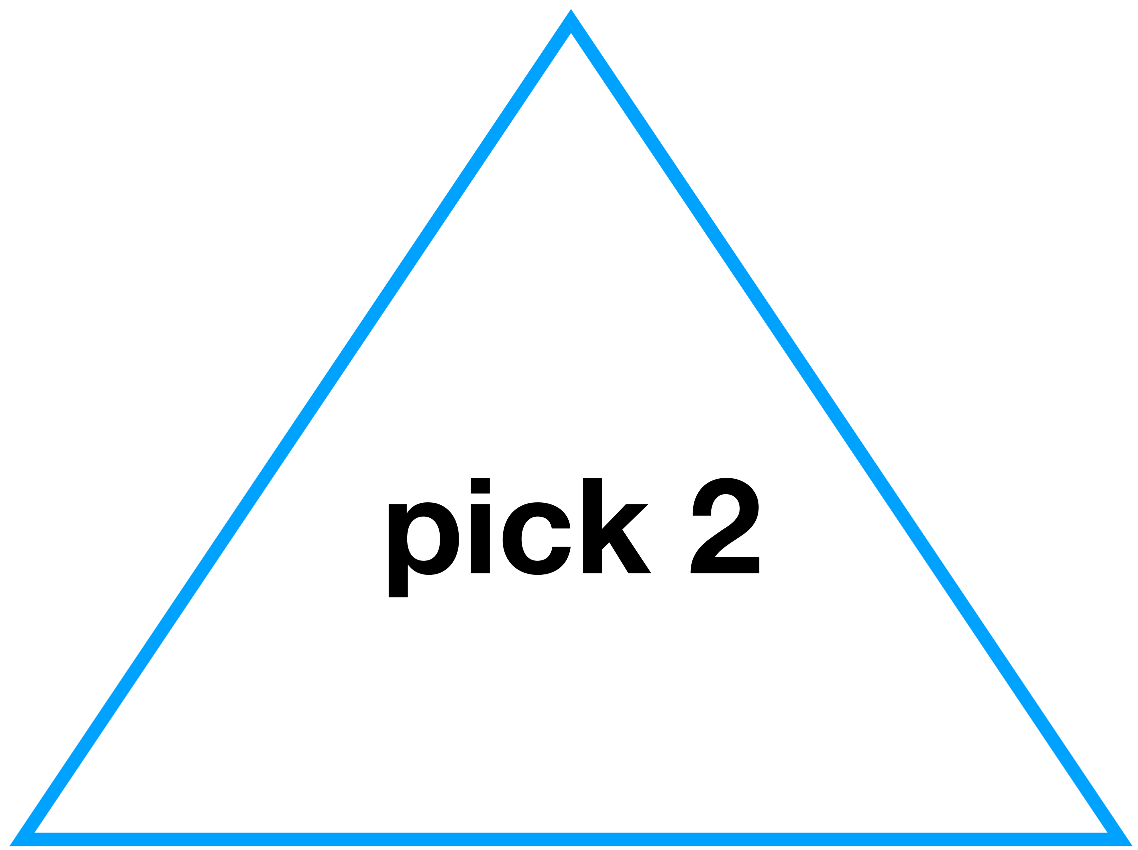
after Michael J. **F**ischer, Nancy **L**ynch, and Mike **P**aterson

- ”we show the surprising result that **no completely asynchronous consensus protocol can tolerate even a single unannounced process death**. We do not consider Byzantine failures, and we assume that the message system is reliable — it delivers all messages correctly and exactly once. Nevertheless, even with these assumptions, the stopping of a single process at an inopportune time can cause any distributed commit protocol to fail to reach agreement.“

FLP Impossibility

- A deterministic consensus protocol that can handle the sudden death of one process does not exist
  - Assumptions
    - Messages may arrive in any order with any delay
    - All messages are eventually received (no lost message)

FLP Result


Fault tolerance
pick 2
termination
(also called liveness,
aka “we make progress”)
Consensus
(also called “safety”, or “agreement”,
aka. “we all do the same”)

FLP Impossibility Proof

- Definitions
  - Consensus Protocol
    - N different processes
    - Write only output register y_p with one value in $\{b,0,1\}$
      - i.e. undecided (bivalent), or a final state
    - Processes act deterministically (no randomness)
    - Processes send messages by adding (p,m) into a single global message queue Q. p=recipient, m=message
    - The global state can be described as $C=(P_1,P_2,P_3,\ldots,Q)$, where P_i is the state of process i and Q the message queueThe protocol proceeds in rounds
      - Take a pair $e=(p,m)$ from the buffer (or ∅, i.e. no message)
      - Depending on p's internal state and m, advance the state of the system

FLP Impossibility Proof

- Faulty: A process that does not react to messages
- Non-Faulty: A process that is not faulty
- Bivalent: A state without a decision, yet. Both outcomes, 0 and 1 are still possible

- Goal:
  - **Termination**: A non-faulty process decides on a value in {0, 1} by entering an appropriate decision state
  - **Weak Agreement**: All non-faulty processes that make a decision are required to choose the same value (only some process need to make a decision)
  - **Validity**: Exclude trivial solutions (constant 0/1), i.e. the final value has to be proposed by some process at some point

- Proof will be done by contradiction
  - Since the trivial solutions are excluded, the initial state must be bivalent
  - We assume that there is a sequence of state transitions from a bivalent state to a deciding state, even if any single process may be unresponsive
  - We prove that there is always a message that keeps the system in a bivalent state

FLP Impossibility Proof

- For the proof, we need 3 ingredients
  1. Messages for different recipients are commutative
     - If two messages are intended for p_1 and p_2, then it does not matter who received the message first
  2. At least one bivalent configuration exists
  3. Given a bivalent configuration and a message, then at least one bivalent following configuration exist
- Any execution of the protocol allow might receive message in such an order that the system will always be bivalent, i.e. never reaches a decision

Commutativity of independent messages

- Suppose we are in state $C=(P_1,P_2,P_3,...,Q)$, and two messages $e_i=(p_i,m_i)$ and $e_j=(p_j,m_j)$ exist.

- Then we can
  - first apply e_i to process p_i and then e_j to process p_j,
  - first apply e_j to process p_j and then p_i to process p_i.

Commutativity of independent messages

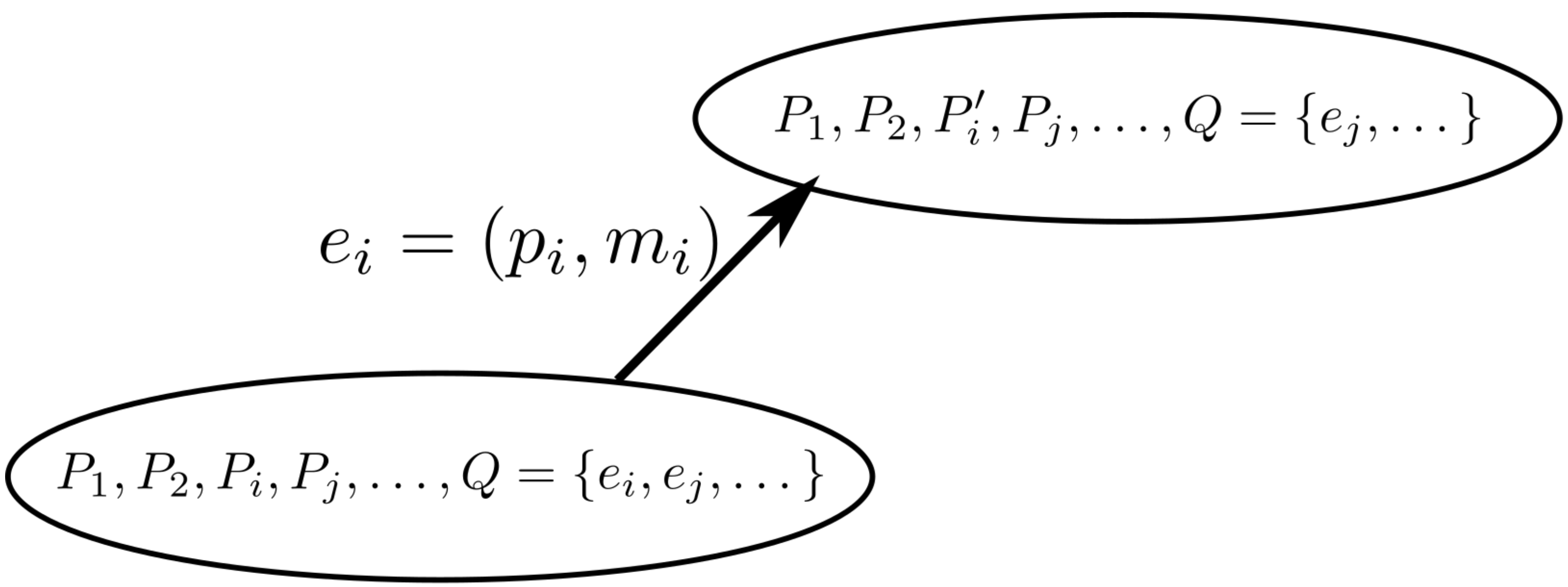
- Suppose we are in state $C=(P_1,P_2,P_3,\ldots,Q)$, and two messages $e_i=(p_i,m_i)$ and $e=(p_j,m_j)$ exist.

- Then we can
  - first apply e_i to process p_i and then e_j to process p_j,
  - first apply e_j to process p_j and then p_i to process p_i.

$$P_1, P_2, P_i, P_j, \ldots, Q = \{e_i, e_j, \ldots\}$$

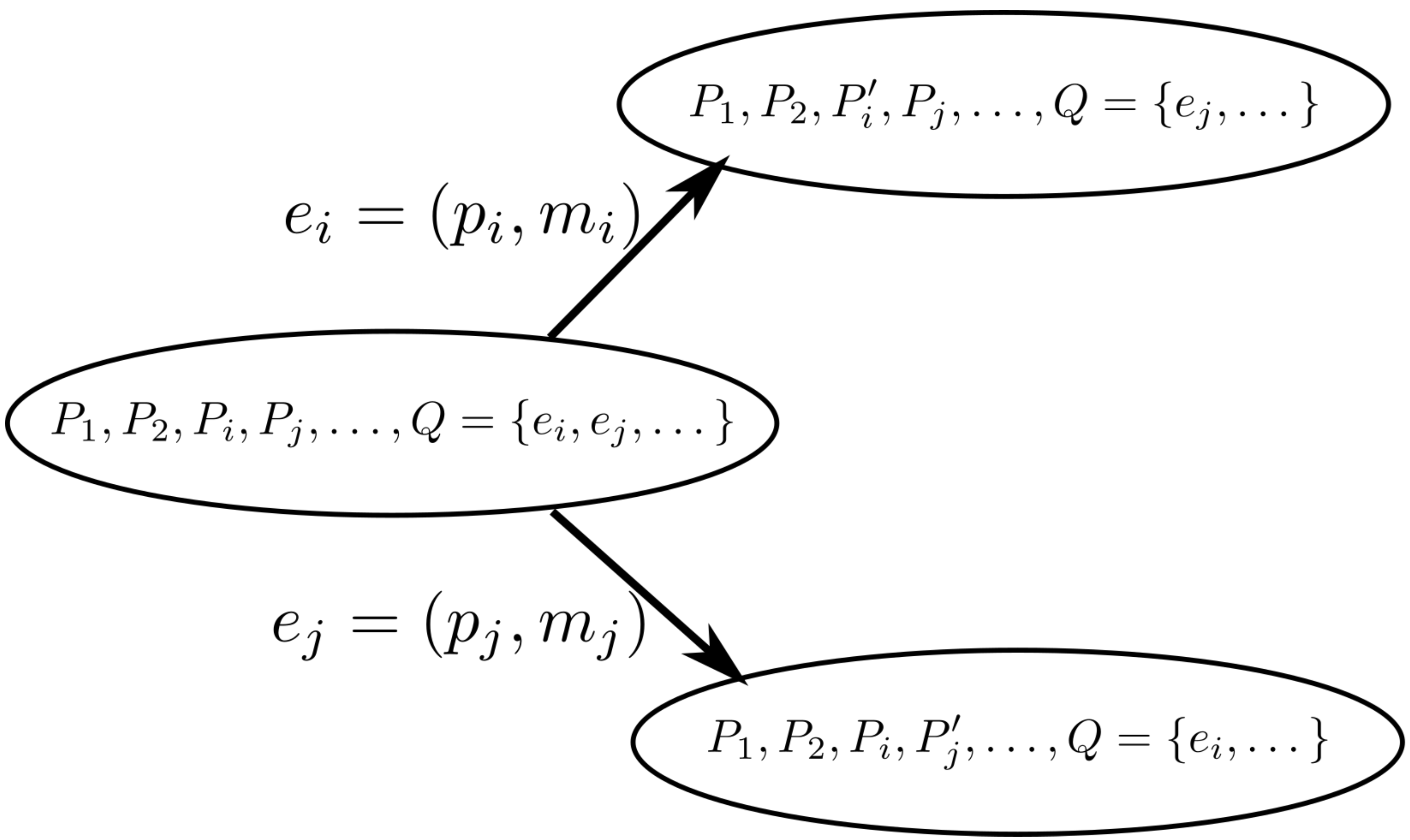
Commutativity of independent messages

- Suppose we are in state $C=(P_1,P_2,P_3,\ldots,Q)$, and two messages $e_i=(p_i,m_i)$ and $e=(p_j,m_j)$ exist.
- Then we can
  - first apply e_i to process p_i and then e_j to process p_j,
  - first apply e_j to process p_j and then p_i to process p_i.



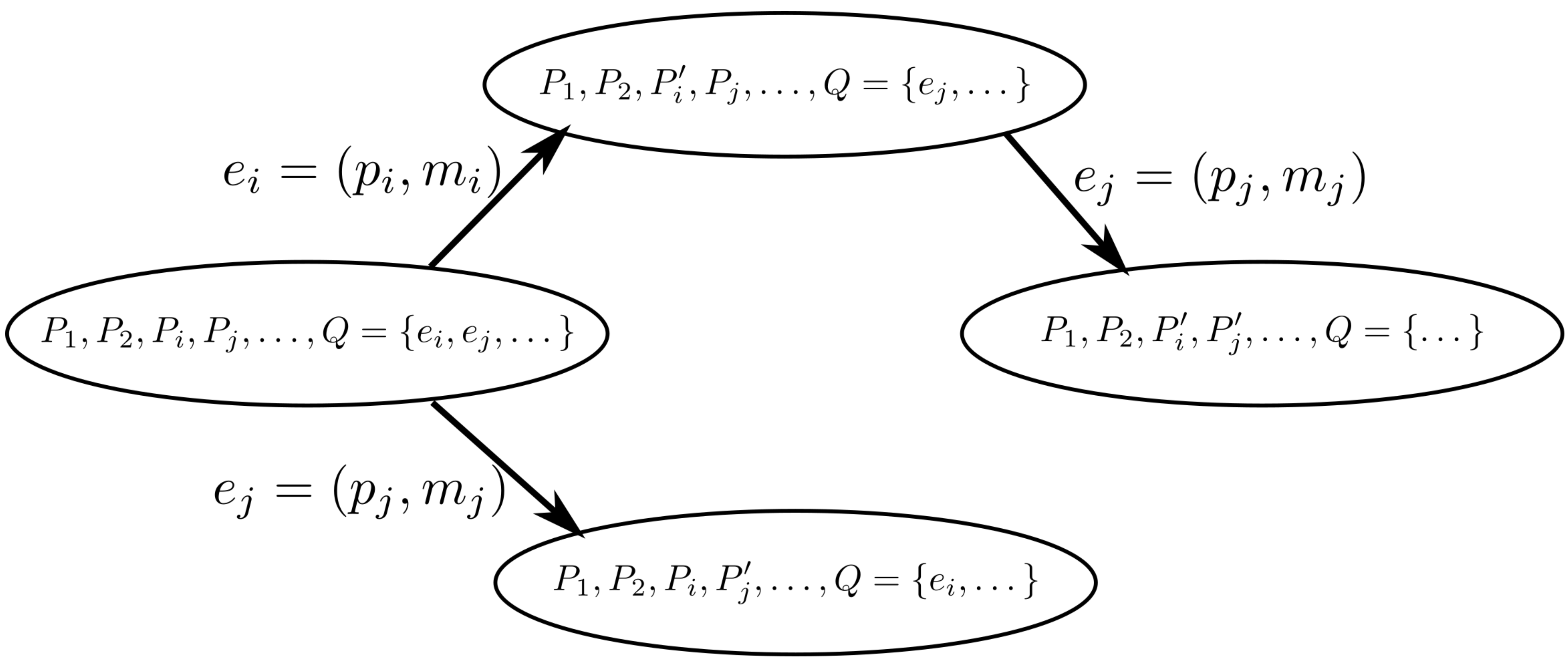
Commutativity of independent messages

- Suppose we are in state $C=(P_1,P_2,P_3,\ldots,Q)$, and two messages $e_i=(p_i,m_i)$ and $e=(p_j,m_j)$ exist.
- Then we can
  - first apply e_i to process p_i and then e_j to process p_j,
  - first apply e_j to process p_j and then p_i to process p_i.



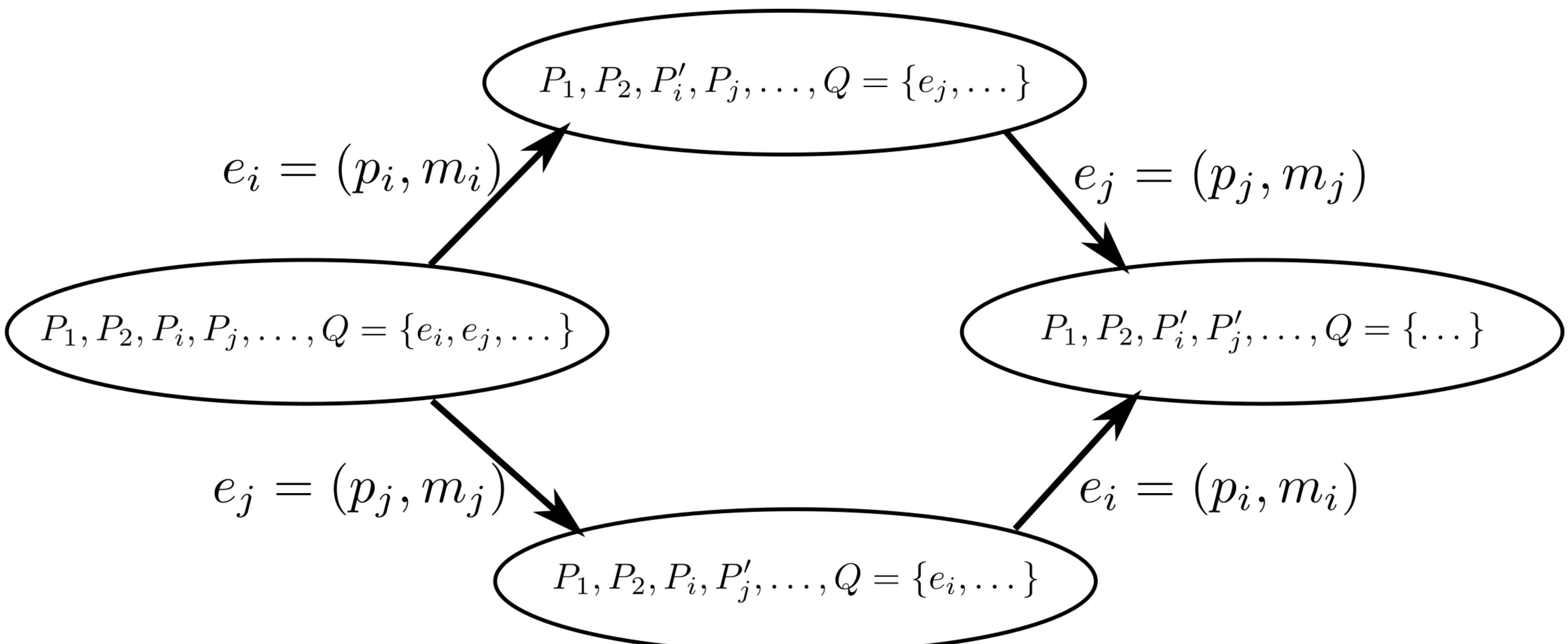
Commutativity of independent messages

- Suppose we are in state $C=(P_1,P_2,P_3,\ldots,Q)$, and two messages $e_i=(p_i,m_i)$ and $e=(p_j,m_j)$ exist.
- Then we can
  - first apply e_i to process p_i and then e_j to process p_j,
  - first apply e_j to process p_j and then p_i to process p_i.



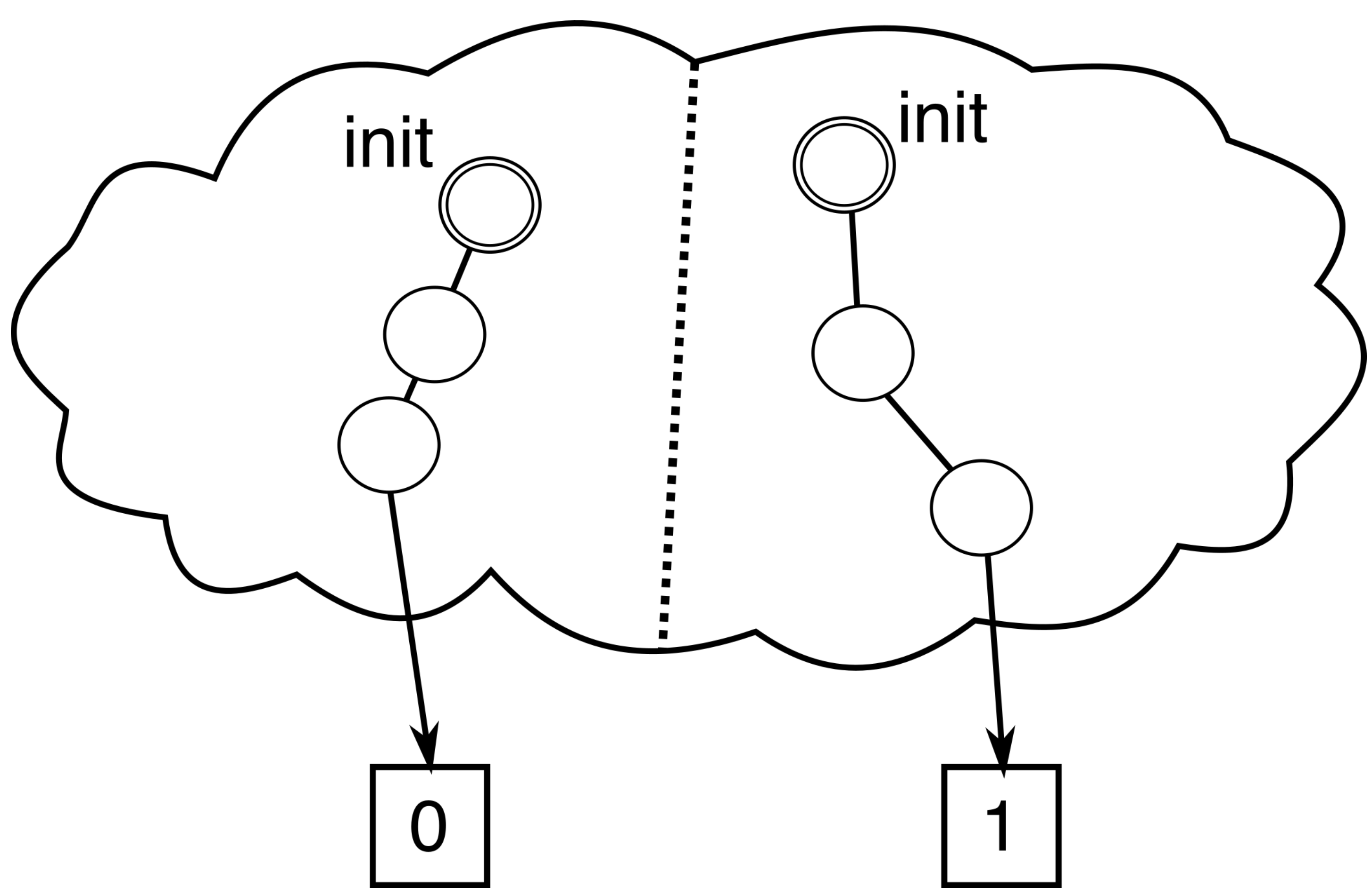
Commutativity of independent messages

- Suppose we are in state $C=(P_1,P_2,P_3,\ldots,Q)$, and two messages $e_i=(p_i,m_i)$ and $e=(p_j,m_j)$ exist.
- Then we can
  - first apply e_i to process p_i and then e_j to process p_j,
  - first apply e_j to process p_j and then p_i to process p_i.



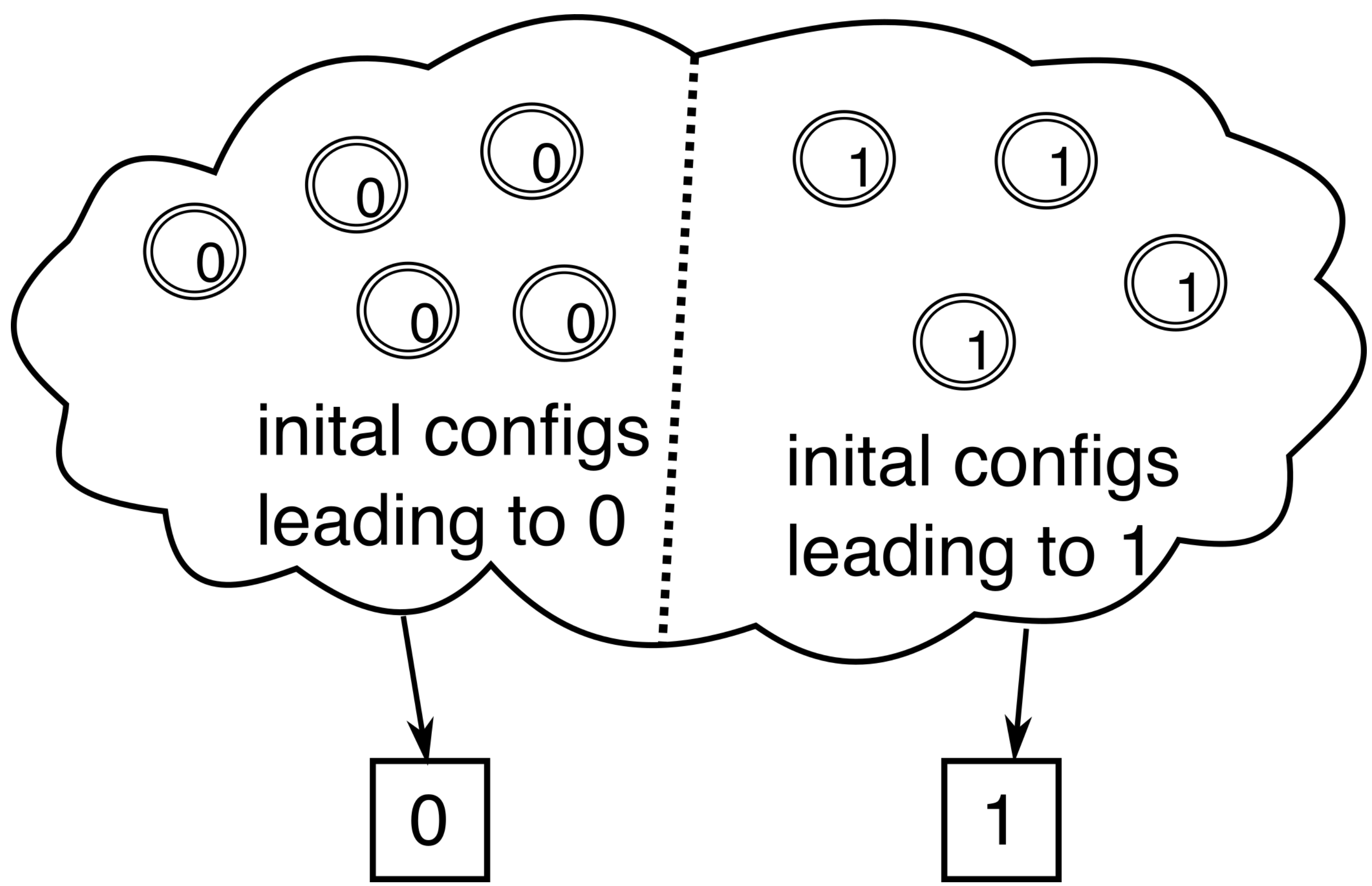
At least one bivalent configuration exists

- Build a contradiction:
  - Assume each initial configuration has only one output value
  - Since we exclude trivial solution, there must be some configurations leading to 0 and some leading to 1



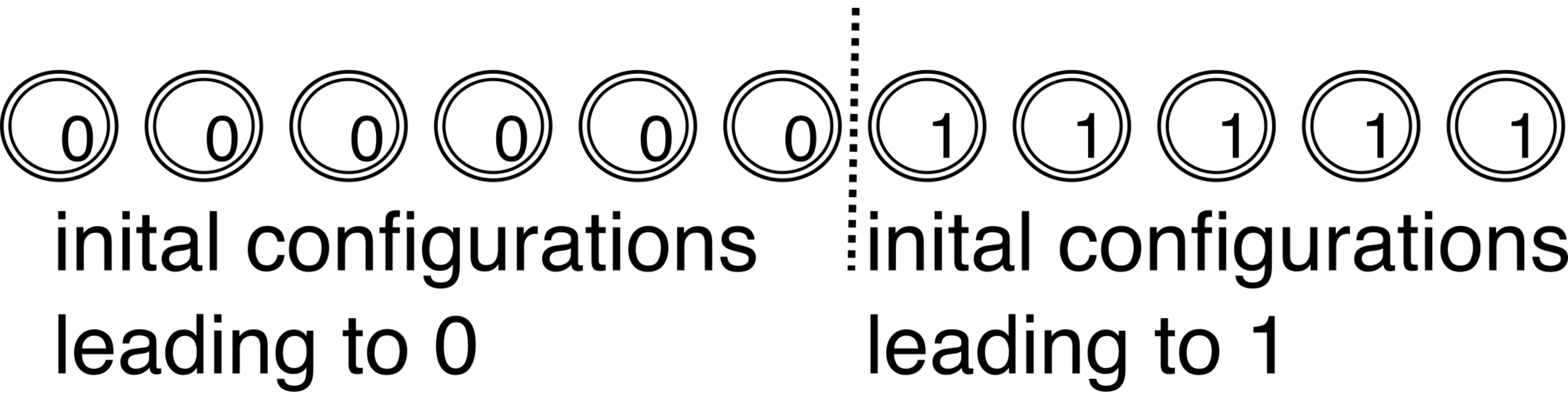
At least one bivalent configuration exists

- Consider all initial configurations and split them into the ones leading to 0 and the ones leading to 1



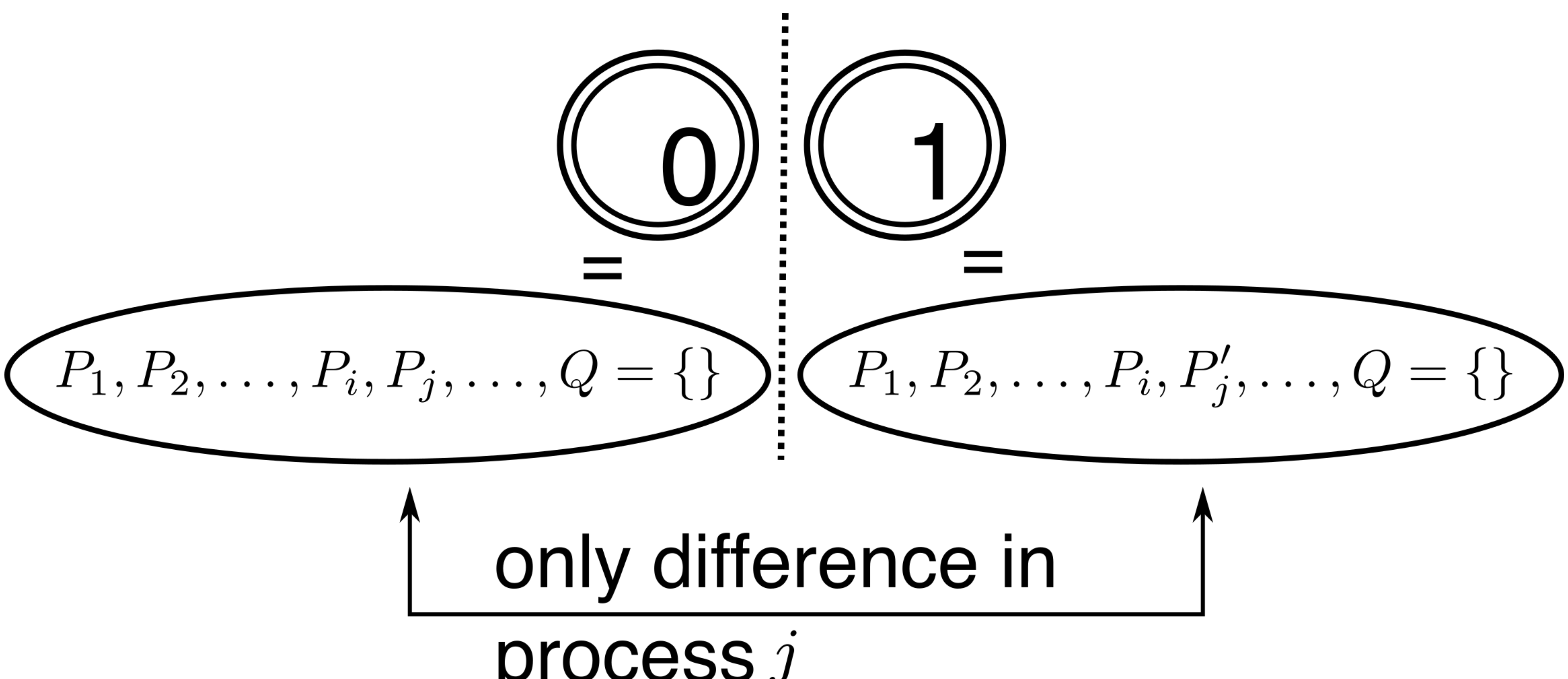
At least one bivalent configuration exists

- Order all initial states
  - difference between neighboring configurations shall be minimal



At least one bivalent configuration exists

- There must be one pair of initial configuration
  - one leads to 0 -> C_0
  - one leads to 1 -> C_1
  - differ in only one process j, all others processes are identical



At least one bivalent configuration exists

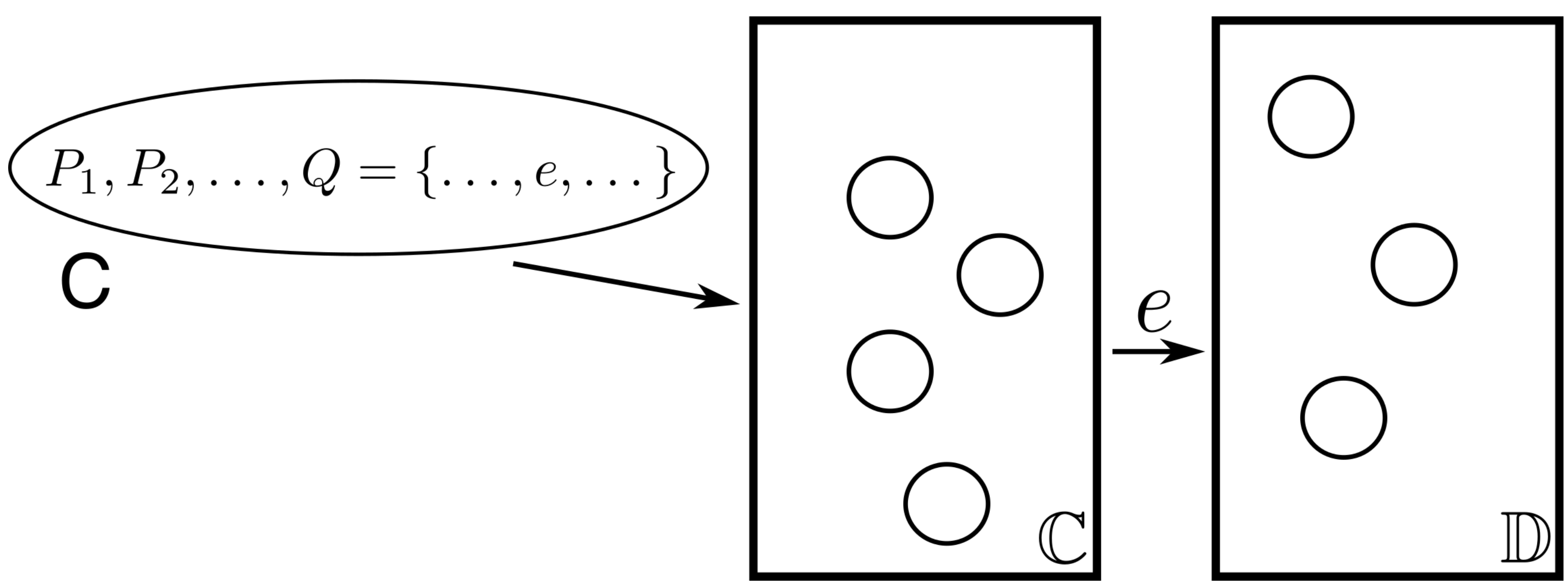
- There must be one pair of initial states
  - one leads to 0 -> C_0
  - one leads to 1 -> C_1
  - differ in only one process j, all others processes are identical

- Our protocol is error tolerant (i.e. it does not matter whether one process is dead)
- Assume process j is dead
- Execution of our protocol must be independent of j
- C_0 and C_1 are indistinguishable, yet lead to 0 resp. 1

Contradiction

Given a bivalent configuration and a message, then at least one bivalent following configuration exist

- Formal:
  - Let C be a bivalent configuration
  - *e=(p,m)* a message of the buffer
  - Let $\mathbb{C}$ be the set of all reachable configurations from C without applying message *e*
  - Let $\mathbb{D}$ be the set of configurations of applying e to the configurations in $\mathbb{C}$
    - There is at least one bivalent configuration in $\mathbb{D}$



Given a bivalent configuration and a message, then at least one bivalent following configuration exist

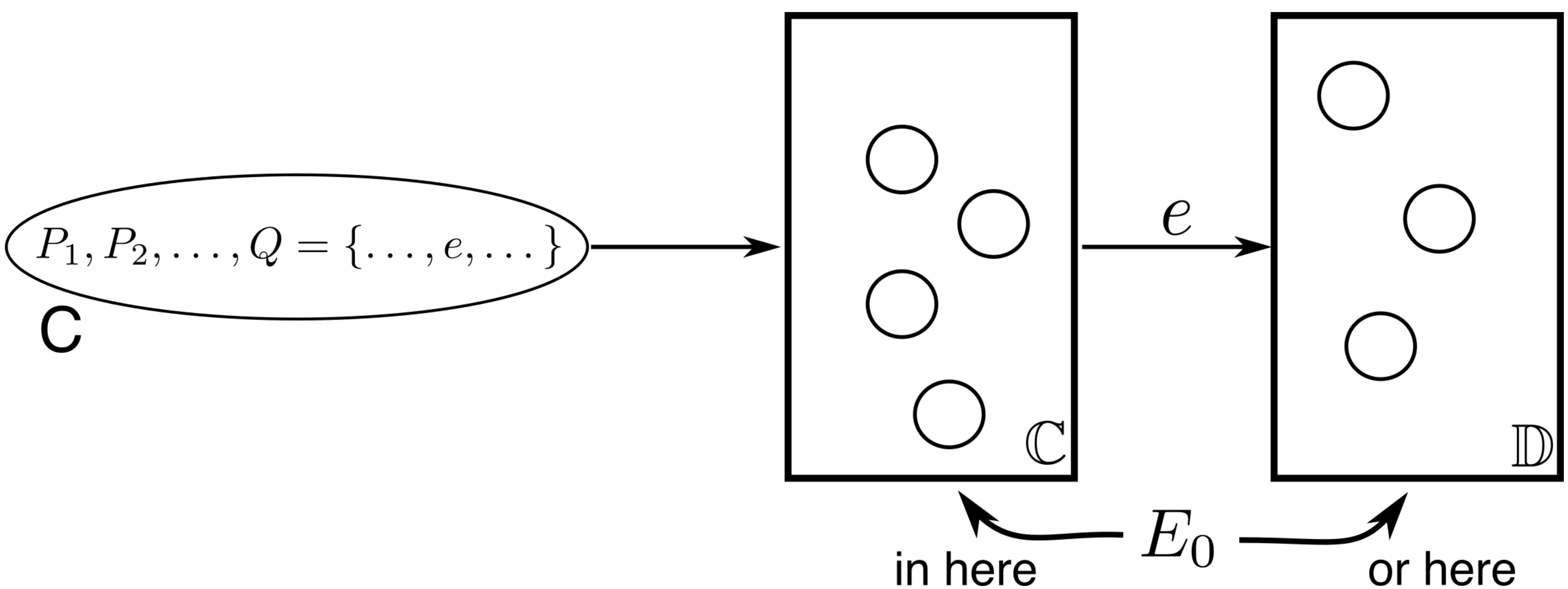
- Formal:
  - Let C be a bivalent configuration
  - $e=(p,m)$ a message of the buffer
  - Let $\mathbb{C}$ be the set of all reachable configurations from C without applying message e
  - Let $\mathbb{D}$ be the set of configurations of applying e to the configurations in $\mathbb{C}$
    - There is at least one bivalent configuration in $\mathbb{D}$

- Proof by contradiction. We show:
  - If no bivalent configurations, then D must have configuration leading to 1 and configurations leading to 0
  - Similar to before, we show that there are configurations that lead to different values, but differ only in one process.
  - If that process is dead, yet our protocol can tolerate dead processes, 0 and 1 must be reachable. Contradiction

Given a bivalent configuration and a message, then at least one bivalent following configuration exist

- Formal:
  - Let C be a bivalent configuration
  - $e=(p,m)$ a message of the buffer
  - Let $\mathbb{C}$ be the set of all reachable configurations from C without applying message e
  - Let $\mathbb{D}$ be the set of configurations of applying e to the configurations in $\mathbb{C}$
    - There is at least one bivalent configuration in $\mathbb{D}$

- Since C is bivalent, there must be a configuration E_0 leading to 0

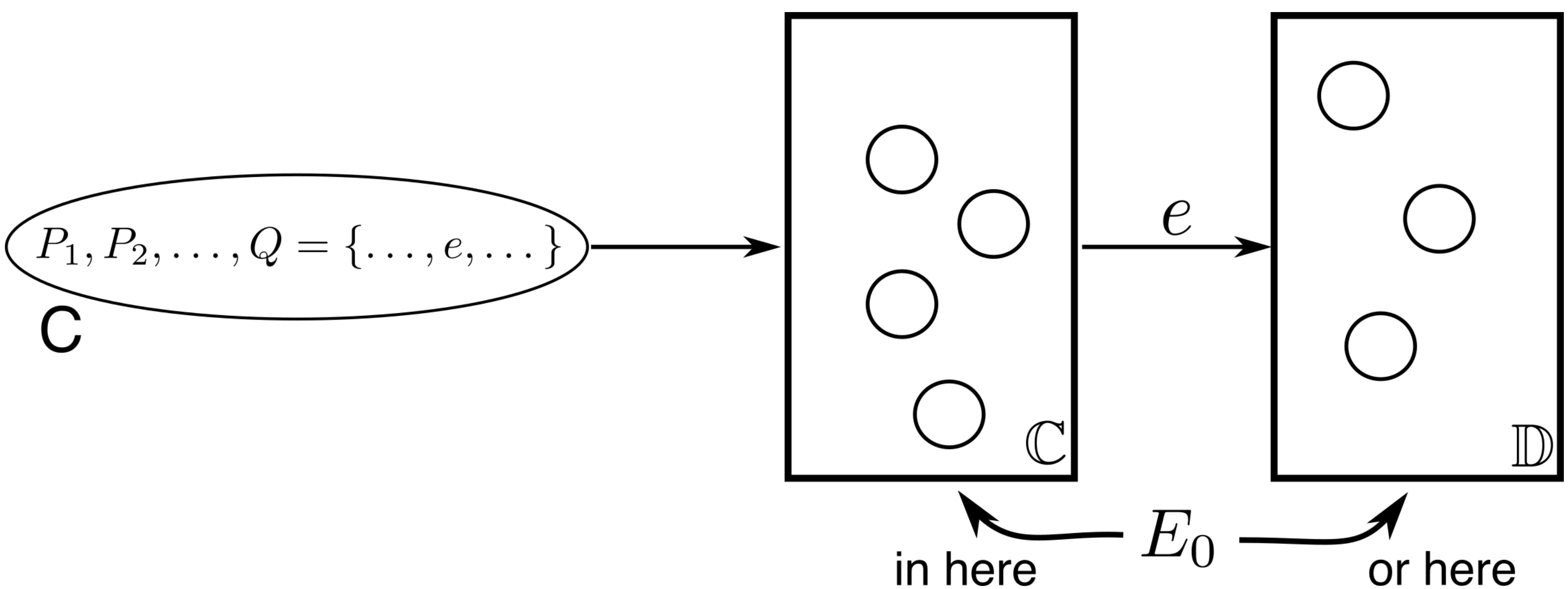


and the same for E_1 leading to 1

Given a bivalent configuration and a message, then at least one bivalent following configuration exist

- Formal:
  - Let C be a bivalent configuration
  - $e=(p,m)$ a message of the buffer
  - Let $\mathbb{C}$ be the set of all reachable configurations from C without applying message e
  - Let $\mathbb{D}$ be the set of configurations of applying e to the configurations in $\mathbb{C}$
    - There is at least one bivalent configuration in $\mathbb{D}$

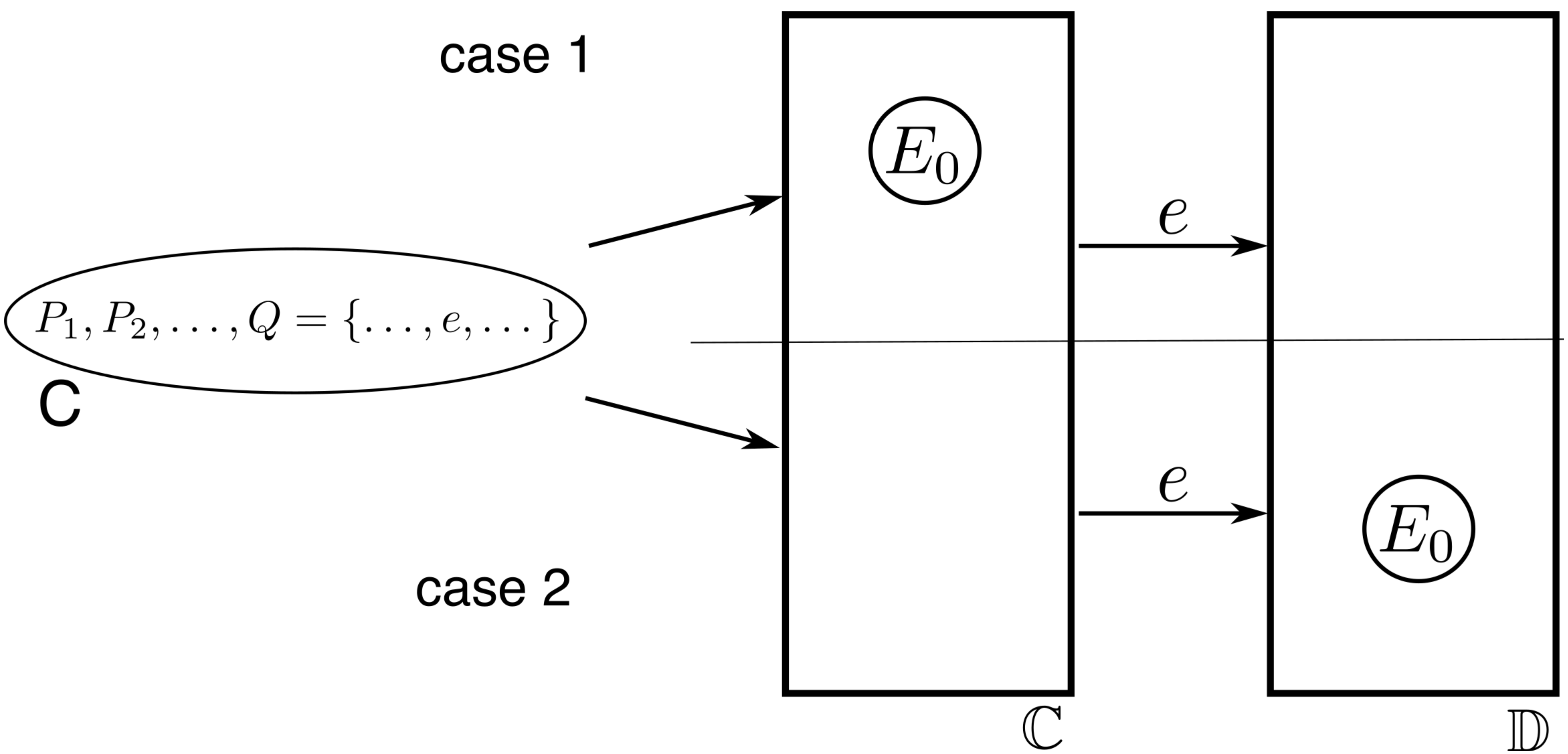
- Since C is bivalent, there must be a configuration E_0 leading to 0



and the same for E_1 leading to 1

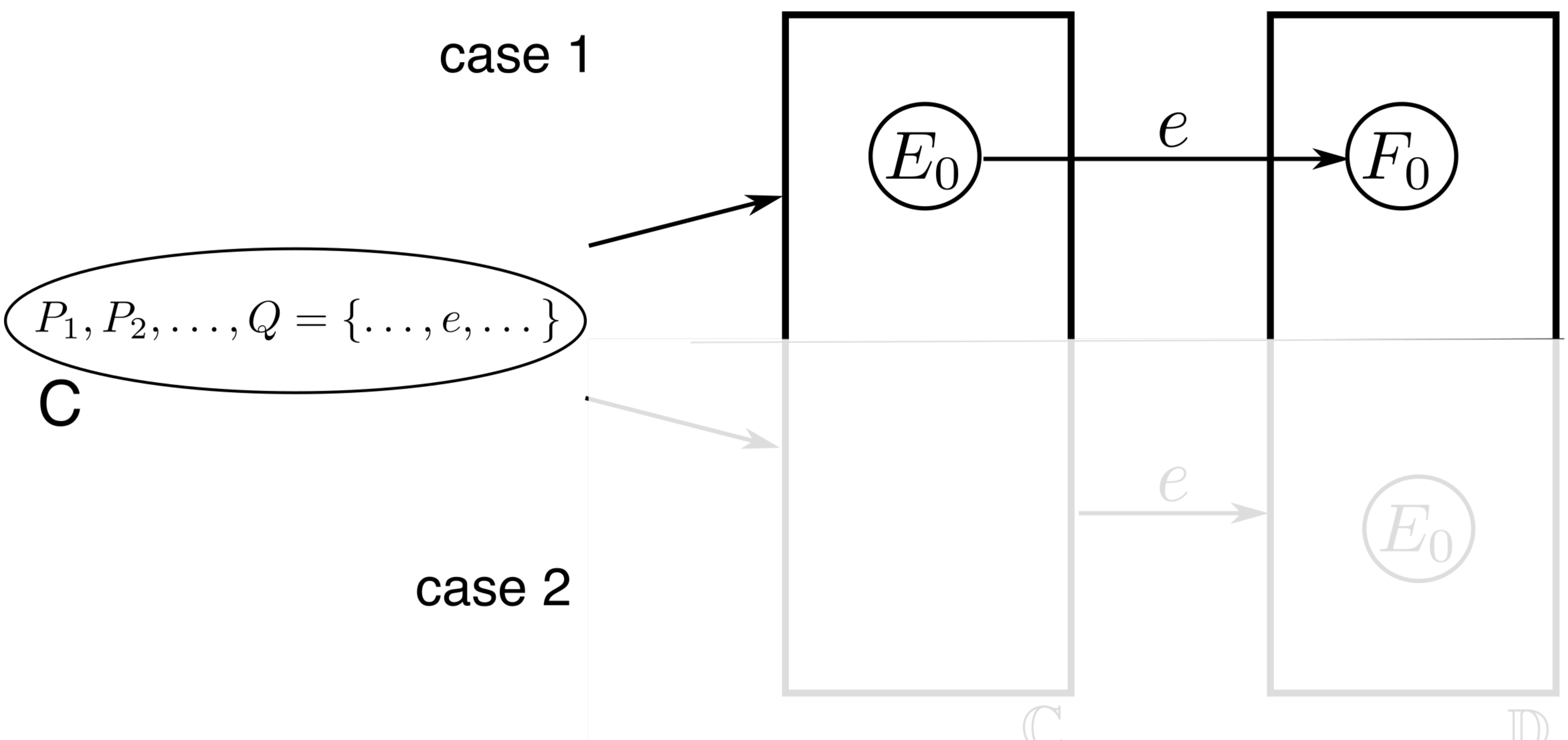
Given a bivalent configuration and a message, then at least one bivalent following configuration exist

- C is bivalent, there must be a configuration E_0 leading to 0
- Let's focus on E_0. E_0 must be
  - case 1: in $\mathbb{C}$
  - case 2: not in $\mathbb{C}$, then it must be in $\mathbb{D}$



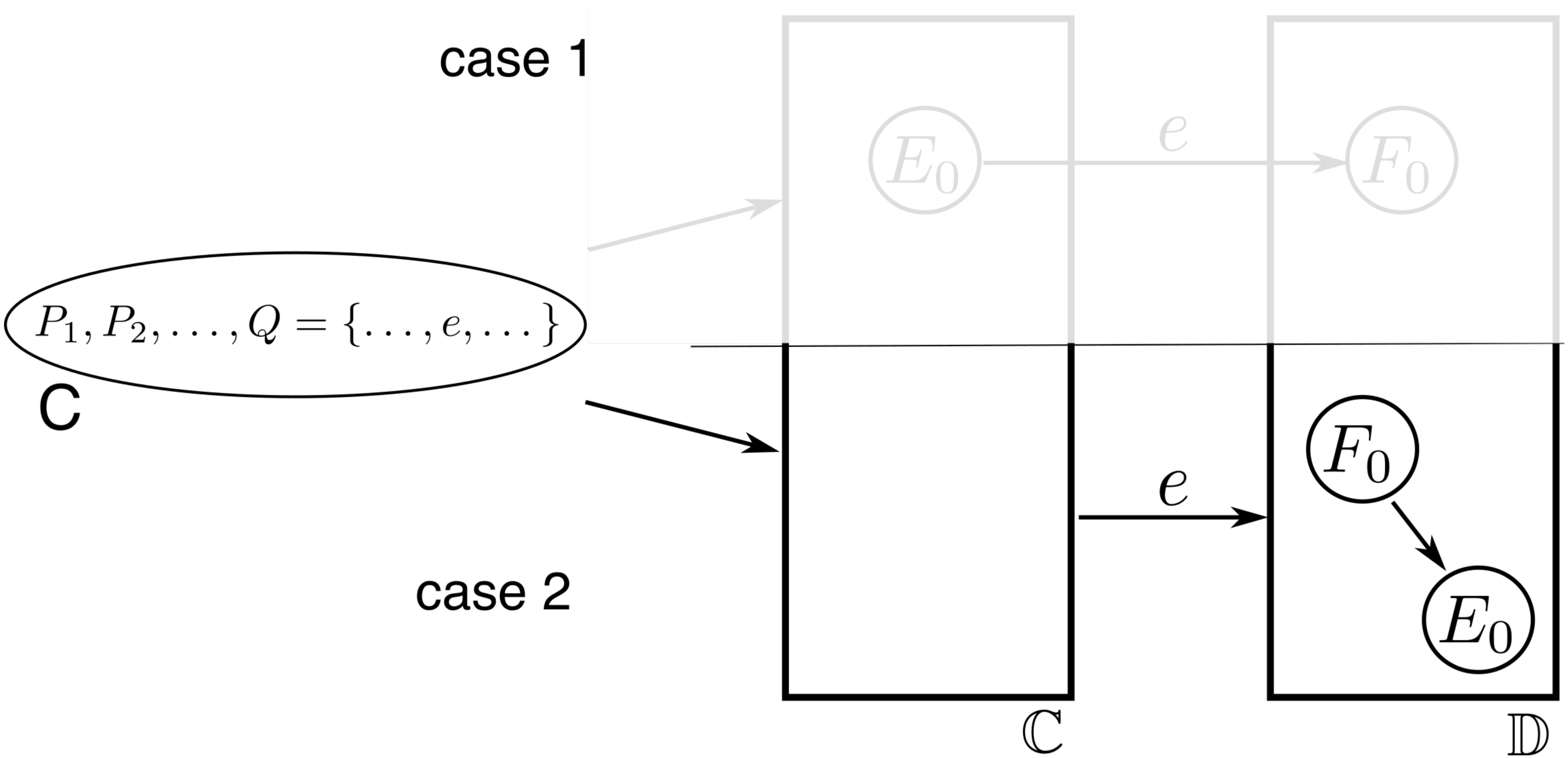
Given a bivalent configuration and a message, then at least one bivalent following configuration exist

- C is bivalent, there must be a configuration E_0 leading to 0
- Let's focus on E_0, case 1
- Let F_0 be the state after applying message e



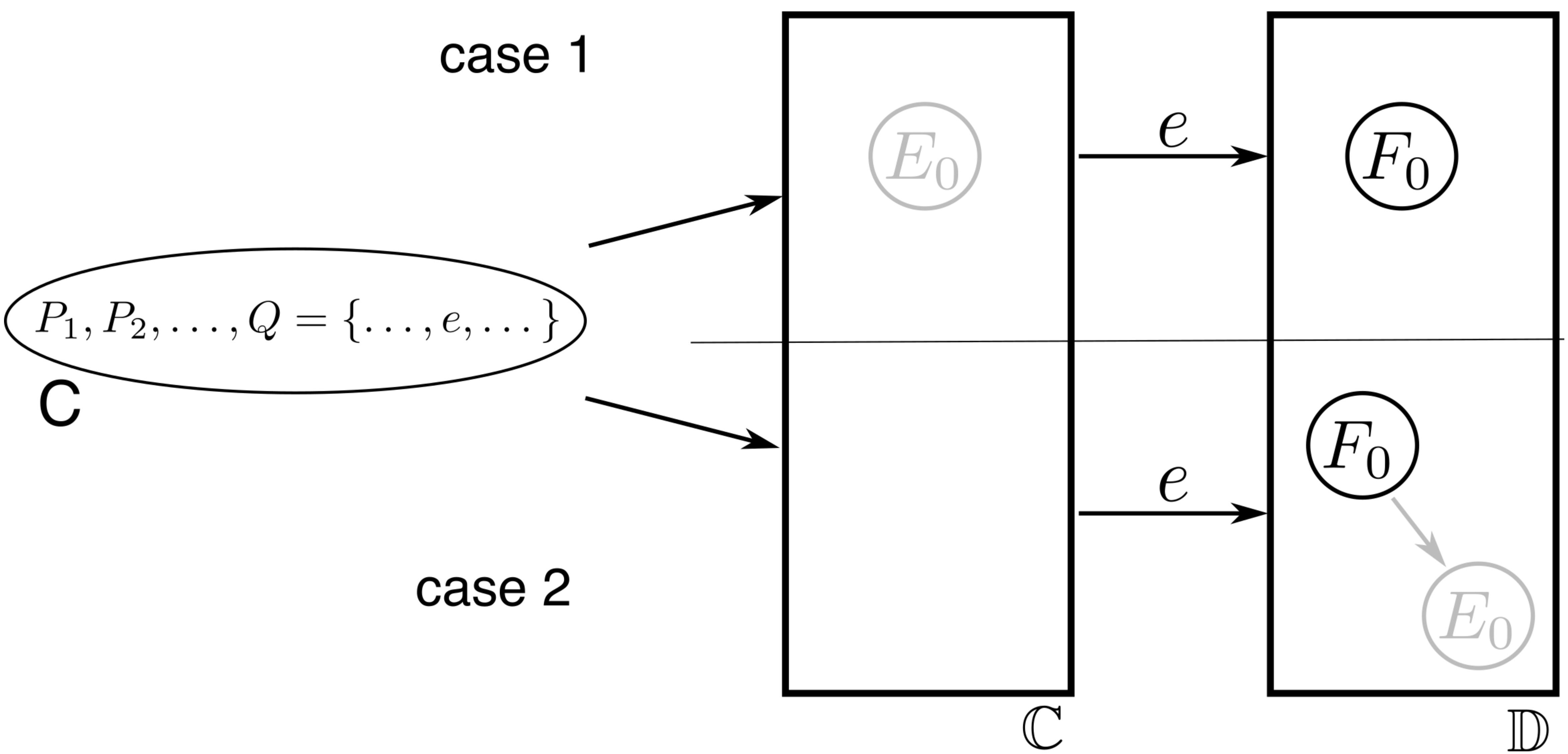
Given a bivalent configuration and a message, then at least one bivalent following configuration exist

- C is bivalent, there must be a configuration E_0 leading to 0
- Let's focus on E_0, case 2
- Let F_0 be the a state in $\mathbb{D}$
  - it must exist, otherwise would the application of e either
    - fix a bivalent configuration (but we assume we do not have bivalent states)
    - change a configuration from 1 to 0 (yet all non-bivalent configs are final)



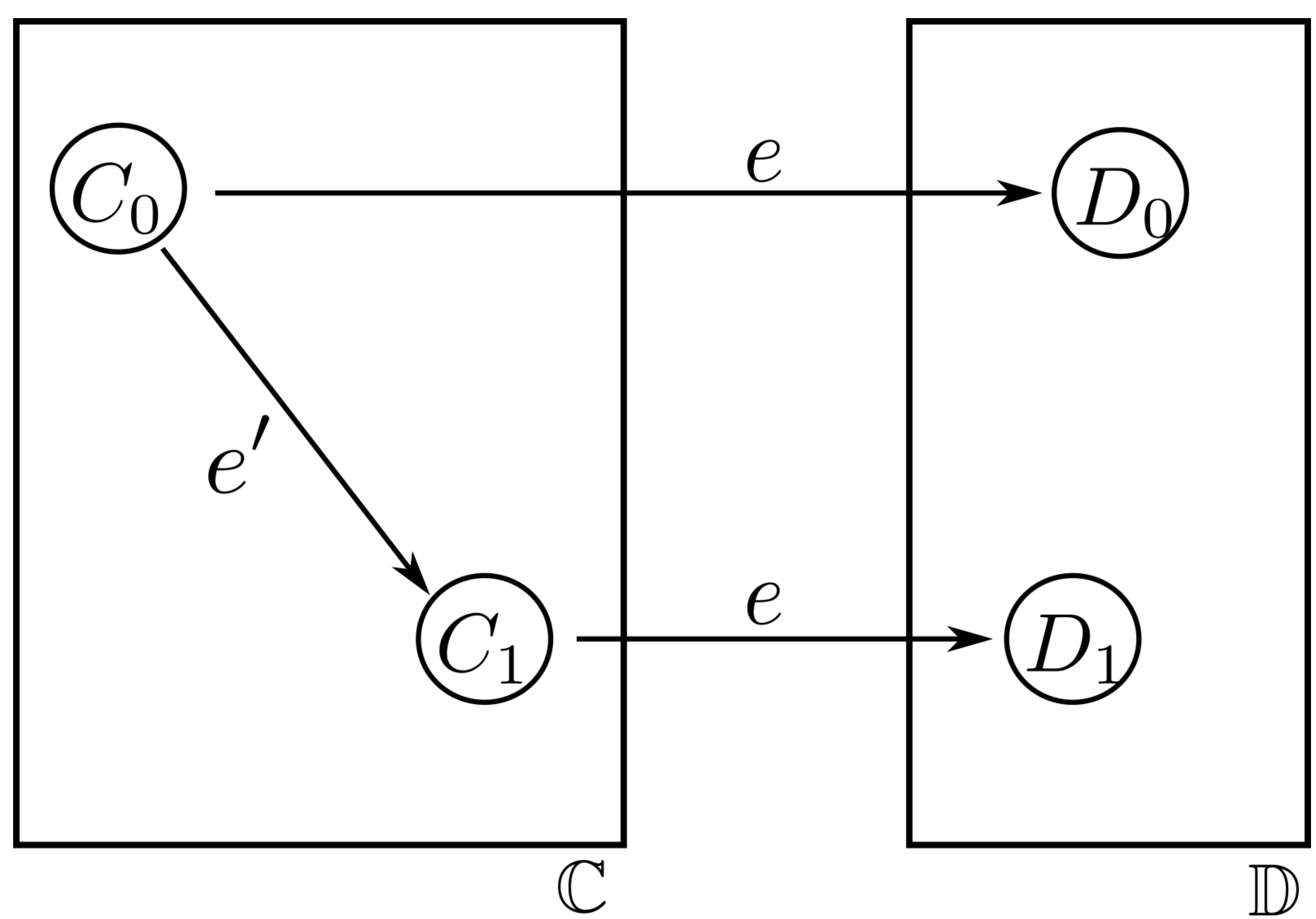
Given a bivalent configuration and a message, then at least one bivalent following configuration exist

- C is bivalent, there must be a configuration E_0 leading to 0
- Le's focus on F_0
  - in both cases, F_0 must exist in $\mathbb{D}$
  - F_0 is a configuration leading to 0
- Similarly, a configuration F_1 leading to 1 must exist in $\mathbb{D}$



Given a bivalent configuration and a message, then at least one bivalent following configuration exist

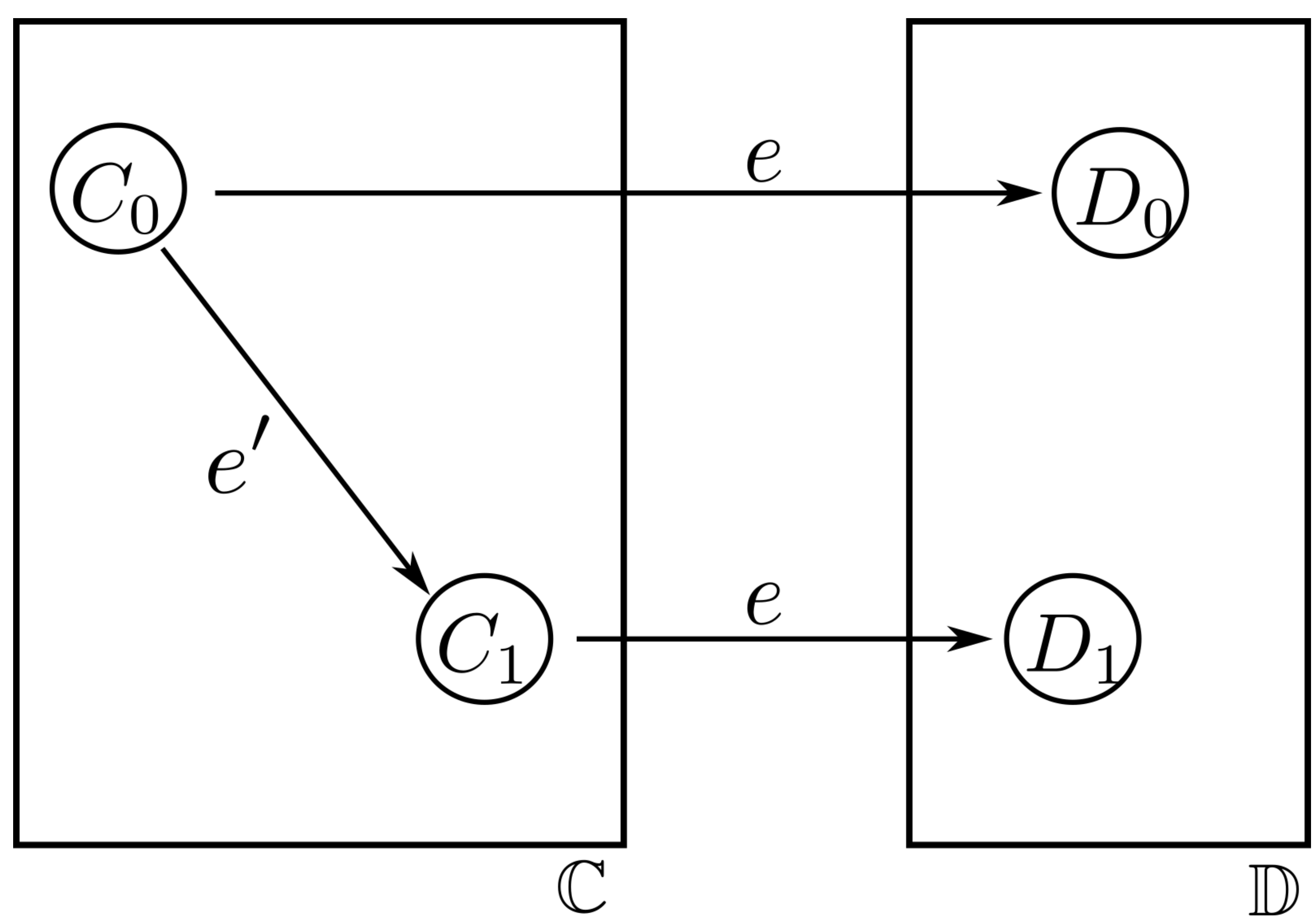
- Set $\mathbb{D}$ must contain
  - D_0 leading to 0
  - D_1 leading to 1
- so that
  - they can be reached from C_0 and C_1 by applying message $e=(p,m)$
  - configurations C_0 and C_1 differ by only one message $e'=(p',m')$
    - configurations C_0 and C_1 are otherwise identical



Given a bivalent configuration and a message, then at least one bivalent following configuration exist

- Configurations C_0 and C_1 lead to D_0 resp. D_1 using $e=(p,m)$
- configurations C_0 and C_1 differ by only one message $e'=(p',m')$
  - configurations C_0 and C_1 are otherwise identical

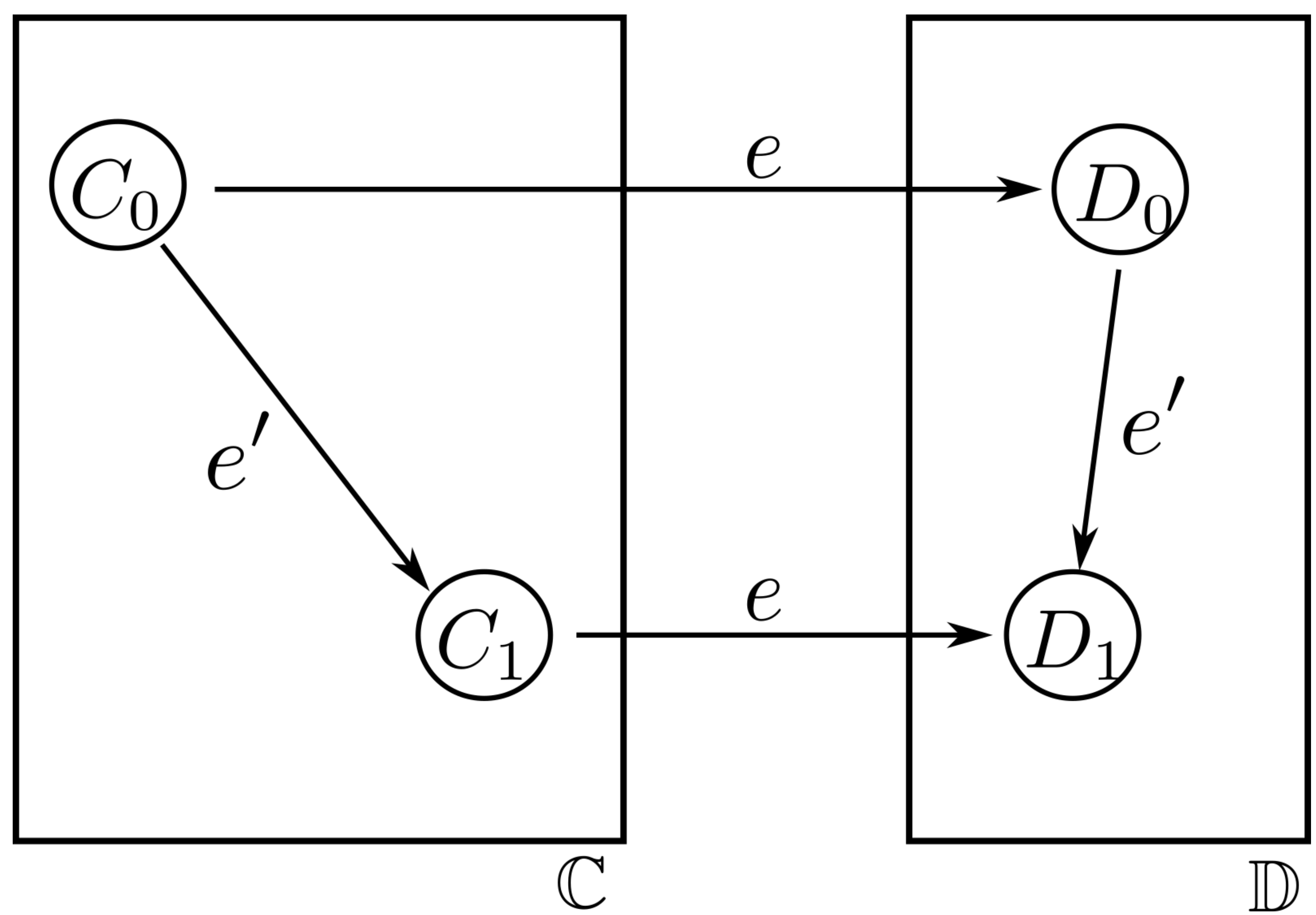
- We distinguish 2 cases, $p=p'$ and $p \neq p'$



Given a bivalent configuration and a message, then at least one bivalent following configuration exist

- Configurations C_0 and C_1 lead to D_0 resp. D_1 using $e=(p,m)$
- configurations C_0 and C_1 differ by only one message $e'=(p',m')$
  - configurations C_0 and C_1 are otherwise identical

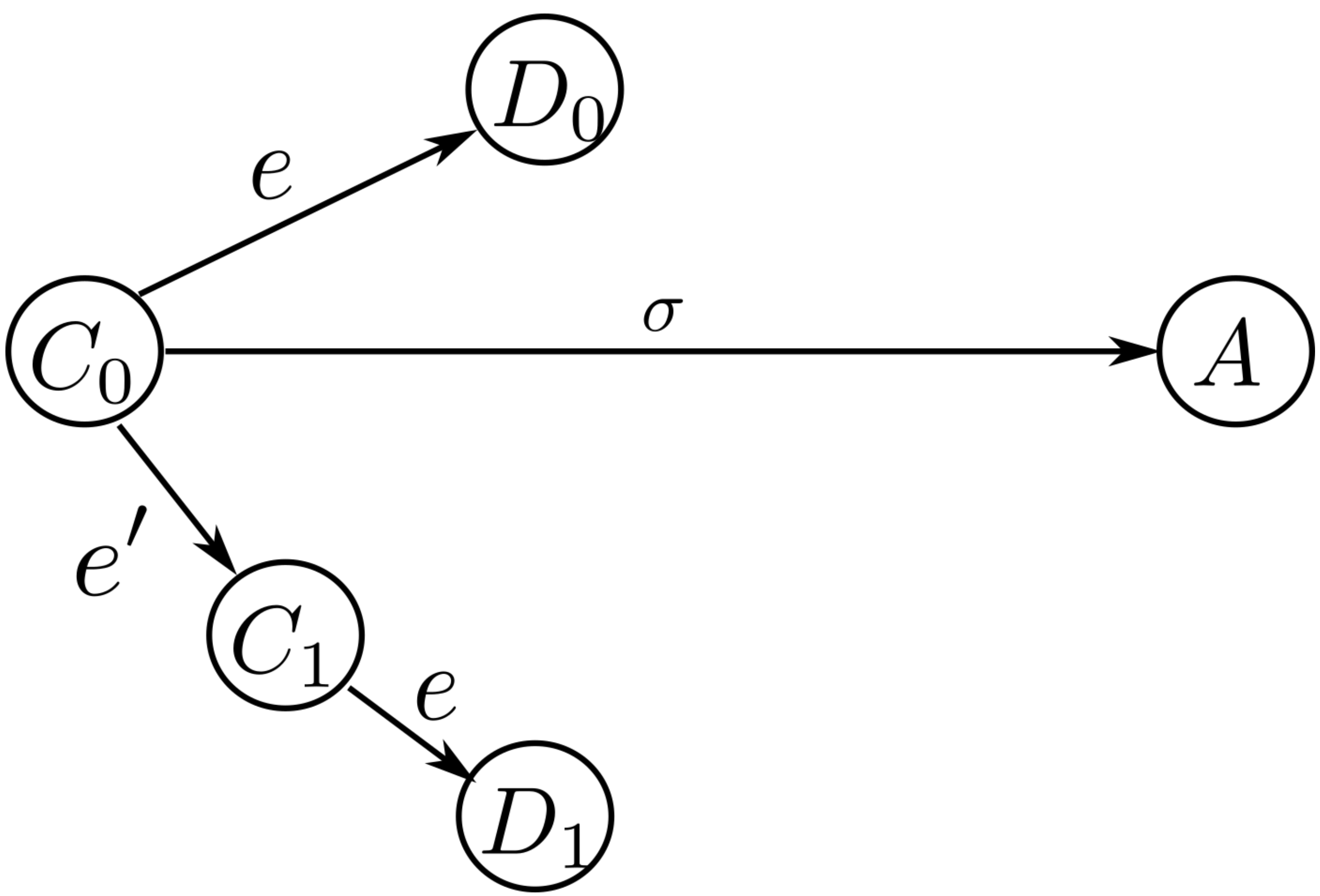
- Case 1, $p \neq p'$:
  - Messages are for two different processes
  - Order in which they are received is irrelevant
  - We can go from D_0 to D_1. Contradiction



Given a bivalent configuration and a message, then at least one bivalent following configuration exist

- Configurations C_0 and C_1 lead to D_0 resp. D_1 using $e=(p,m)$
- configurations C_0 and C_1 differ by only one message $e'=(p',m')$
  - configurations C_0 and C_1 are otherwise identical

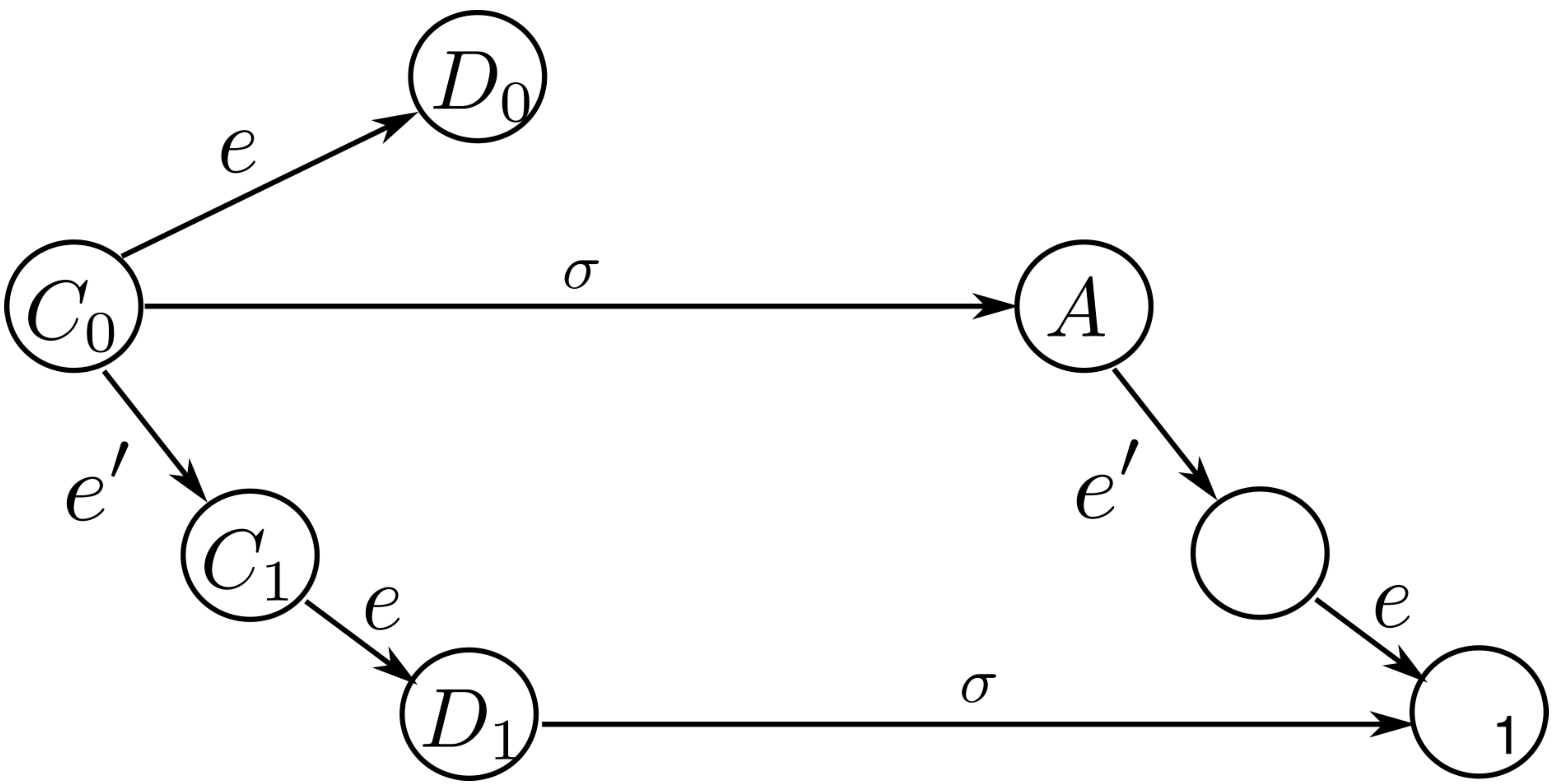
- Case 2, $p=p'$: both messages are for the same processes
  - Our protocol can tolerate one dead process
  - There is an execution path σ that does not need process p
  - execution path σ leads from C_0 to a non-bivalent configuration A



Given a bivalent configuration and a message, then at least one bivalent following configuration exist

- Configurations C_0 and C_1 lead to D_0 resp. D_1 using $e=(p,m)$
- configurations C_0 and C_1 differ by only one message $e'=(p',m')$
  - configurations C_0 and C_1 are otherwise identical

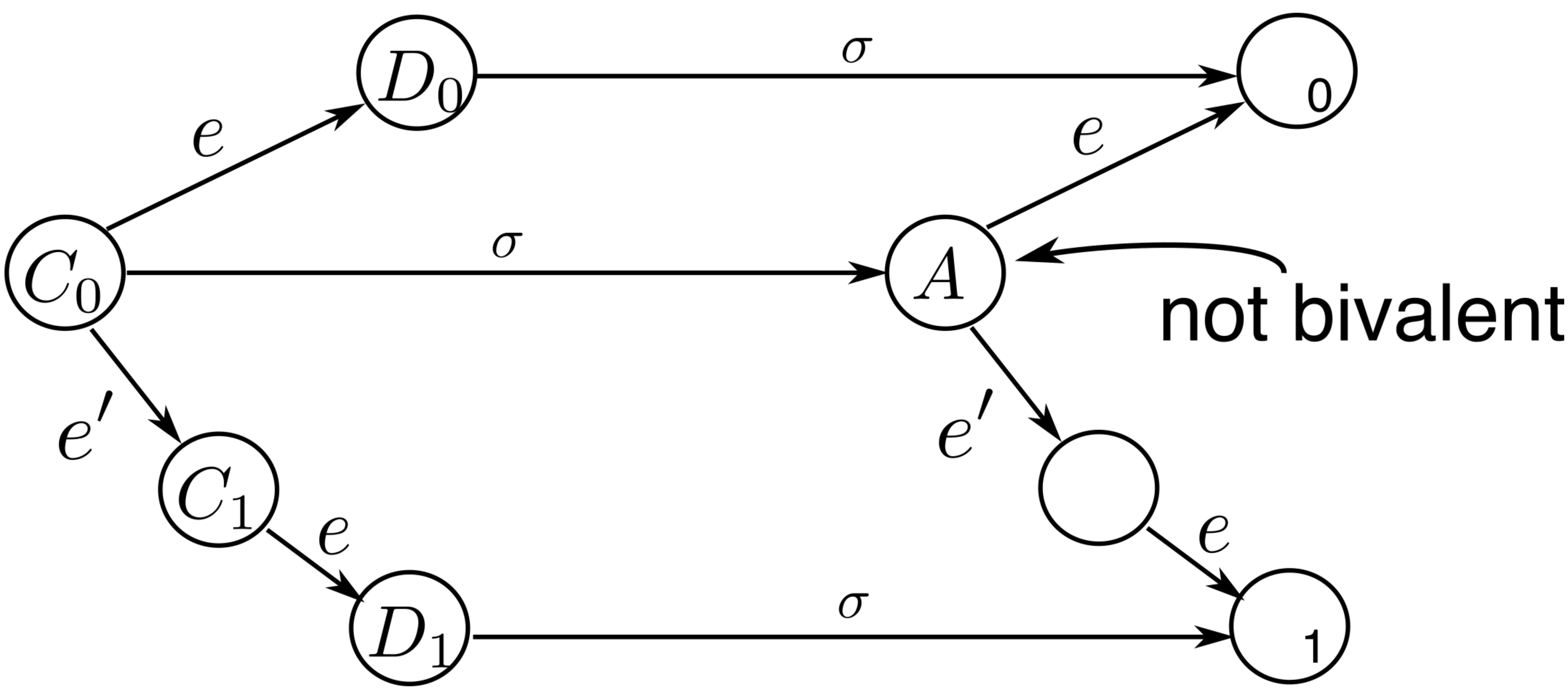
- Case 2, $p=p'$:
  - execution path σ and (e,e') are commutative, since they do not involve the same processes
  - Applying (e,e') to A leads to 1, since $D1$ is a configuration leading to 1



Given a bivalent configuration and a message, then at least one bivalent following configuration exist

- Configurations C_0 and C_1 lead to D_0 resp. D_1 using $e=(p,m)$
- configurations C_0 and C_1 differ by only one message $e'=(p',m')$
    - configurations C_0 and C_1 are otherwise identical

- Case 2, $p=p'$:
    - But we can also apply message e to σ, since they are commutative
    - Thus, from A can lead to 1 and 0
    - Contradiction, A is not bivalent



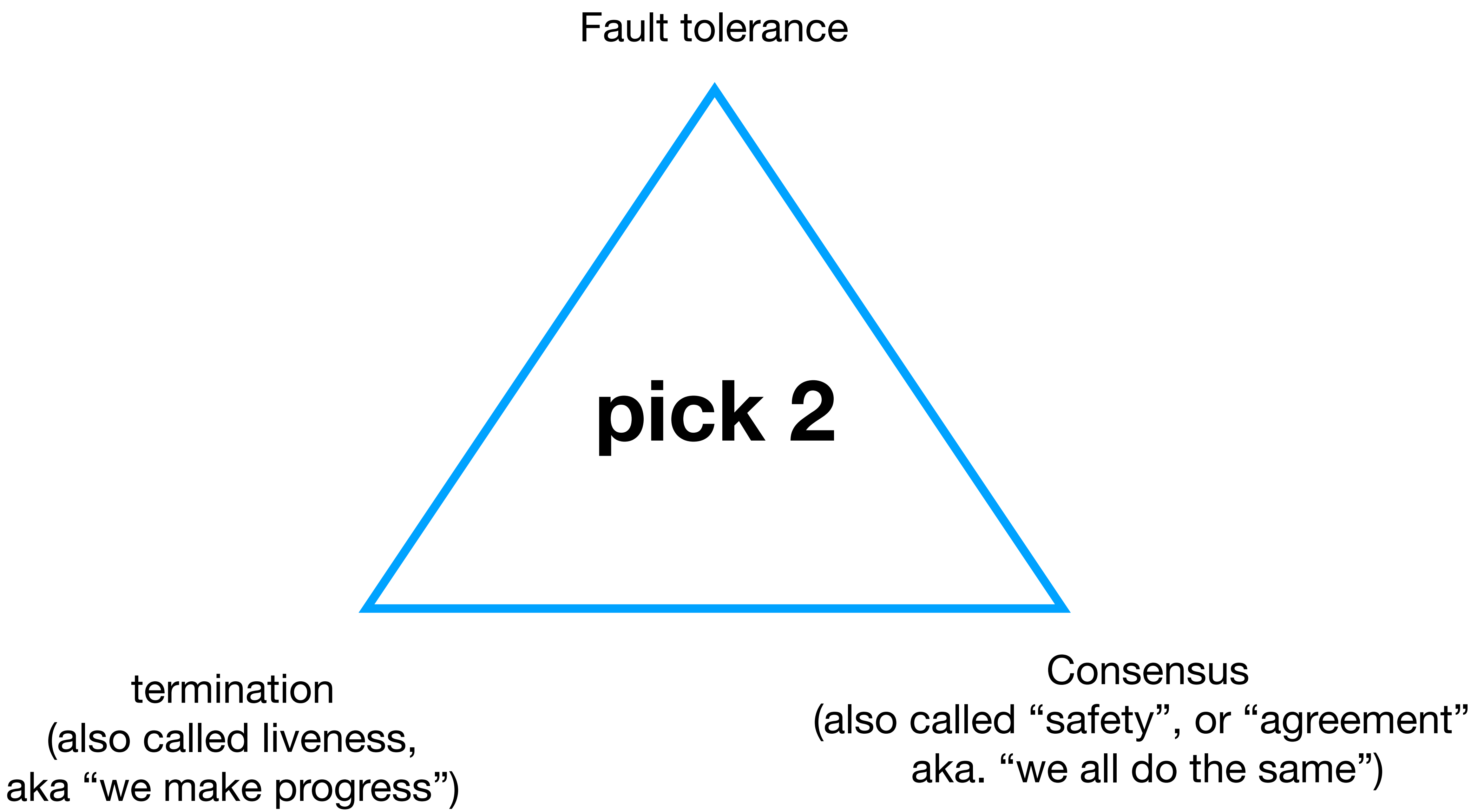
Wrapping up

- If we have a deterministic, fault-tolerant protocol and the system is in a bivalent configuration (output not yet decided), we can always find a processing step that leads to another bivalent configuration

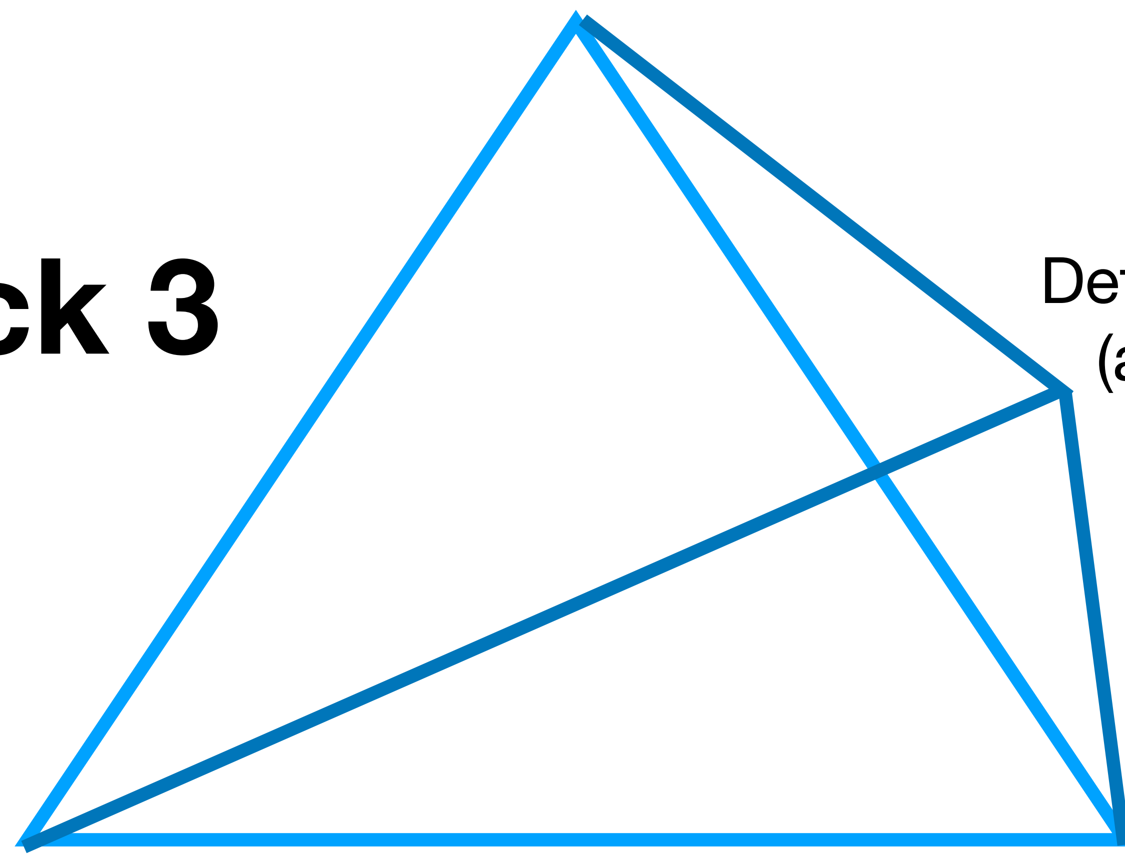
  - Bivalent configurations exist
    (if we ignore trivial solutions that always return 0 or 1)

- No deterministic fault-tolerant protocol can guarantee consensus

Take away
“FLP Result”



Take away
“FLP Result”



Take away

- The exact proofs themselves are not as important as the insight they provide
- Different definitions of a consensus protocols are possible
  - Byzantine Fault Tolerance deals with input into the decision process
    A. Any two non-faulty nodes use the same value $v(i)$.
    B. If the ith node is non-faulty, then it's value must be used by every other non-faulty node as $v(i)$.
  - FLP deals with eventually reaching a decision
    - **Termination**: All non-faulty processes eventually decide on a value
    - **Agreement**: All processes decide on the same value
      - FLP uses **Weak Agreement**: Only the processes that terminate must decide on the same value.
    - **Validity**: The value that has been decided must have proposed by some process

Take away
"Byzantine Fault Tolerance"

- Assuming all messages arrive on time

  - No consensus protocol can tolerate $\geq \frac{1}{3}^{\text{rd}}$ traitors (without signatures and known identities)

  - With signatures and a mechanism when to stop listening to messages, arbitrarily many traitors can be tolerated

Consequences

- These 2 lectures have been rather theoretical
- The results have a HUGE impact on the design of blockchain applications, i.e.
  - Fault tolerance
    - resistance against hostile takeover
  - Problems with determinism
    - how/when to use randomness

Student Presentations

- Starting Sep. 9th, classes will start with student presentations
- Each student has to present twice during the semester
  - One paper (from a list of pre-selected papers)
  - One interesting thing about blockchains
    - Quality/Reputability of source is important
    - Nothing illegal
  - 7-10 min presentation
  - The lecture before, we need to see the presentation