Impossibility of consensus

The FLP theorem and its proof

Hernán Vanzetto
hernan.vanzetto@gmail.com

April 2019

Overview

- Definition of consensus and its properties
- Network models
- Resiliency: maximum number of faults supported by each model
- FLP impossibility theorem and its proof
- Three consensus protocols and how they overcome FLP

Consensus problem

- A collection of **participants** which communicate by sending **messages**
- There may be **faults** in the participants and/or the communication network
- One or more participants propose **values**
- All correct participants must **reach agreement** on one single value

Consensus properties

- An algorithm solves consensus if it satisfies these properties:
  - **Agreement**: All participants that decide do so on the same value
  - **Validity**: The decided value must have been proposed by some participant
  - **Termination**: All non-faulty participants eventually decide on a value

Network models

- Δ = time required for a message to be sent from one participant to another
- ϕ = relative speed of different participants (processors)
- **Synchronous system**: there are known fixed upper bounds Δ and ϕ
- **Asynchronous system**: there are no bounds for Δ and ϕ (delays are arbitrary)
- **Partially synchronous systems**
    - definition 1: Δ and ϕ exist but they are unknown
    - definition 2: Δ and ϕ are known and have to hold starting from some unknown time T

A classification of failures

- **Crash failure**: the process/participant halts; it is irreversible
  - fail-stop (undetectable)
  - fail-safe (detectable)
- **Omission failure**: message lost in transit
- **Byzantine failure**: anything is possible; weakest type of failure
- **Authenticated Byzantine**: Byzantine using digital signatures for messages

Minimum number of participants required to support *f* faults

Failure type	Synchronous	Partially sync. communication, sync. processors	Partially sync. communication & processors	Sync. comm., partially sync. processors	Asynchronous [2]
Fail-stop	f + 1	2f + 1	2f + 1	f + 1	∞
Omission	f + 1	2f + 1	2f + 1	[2f, 2f + 1]	∞
Authenticated Byzantine	f + 1	3f + 1	3f + 1	2f + 1	∞
Byzantine [5]	3f + 1	3f + 1	3f + 1	3f + 1	∞

[4]

Resiliency: maximum number of faulty participants supported by each model

Impossibility of consensus (FLP)

Impossibility of Distributed Consensus with One Faulty Process

MICHAEL J. FISCHER

Yale University, New Haven, Connecticut

NANCY A. LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

MICHAEL S. PATERSON

University of Warwick, Coventry, England

Abstract. The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that every protocol for this problem has the possibility of nontermination, even with only one faulty process. By way of contrast, solutions are known for the synchronous case, the "Byzantine Generals" problem.

FLP: weak assumptions

- For just one failure
- For “weak” consensus (only one process needs to decide)
- For reliable communication: messages delivered correctly and exactly once
- For only two values: 0 and 1
- For crash failures (fail-stop model)
- FLP applies also for: many failures, quorum consensus, unreliable communication, Byzantine failures

FLP: strong assumptions

- Participants take **deterministic** actions
- **Asynchronous** network communication
- All “runs” must eventually achieve consensus

Deterministic processes

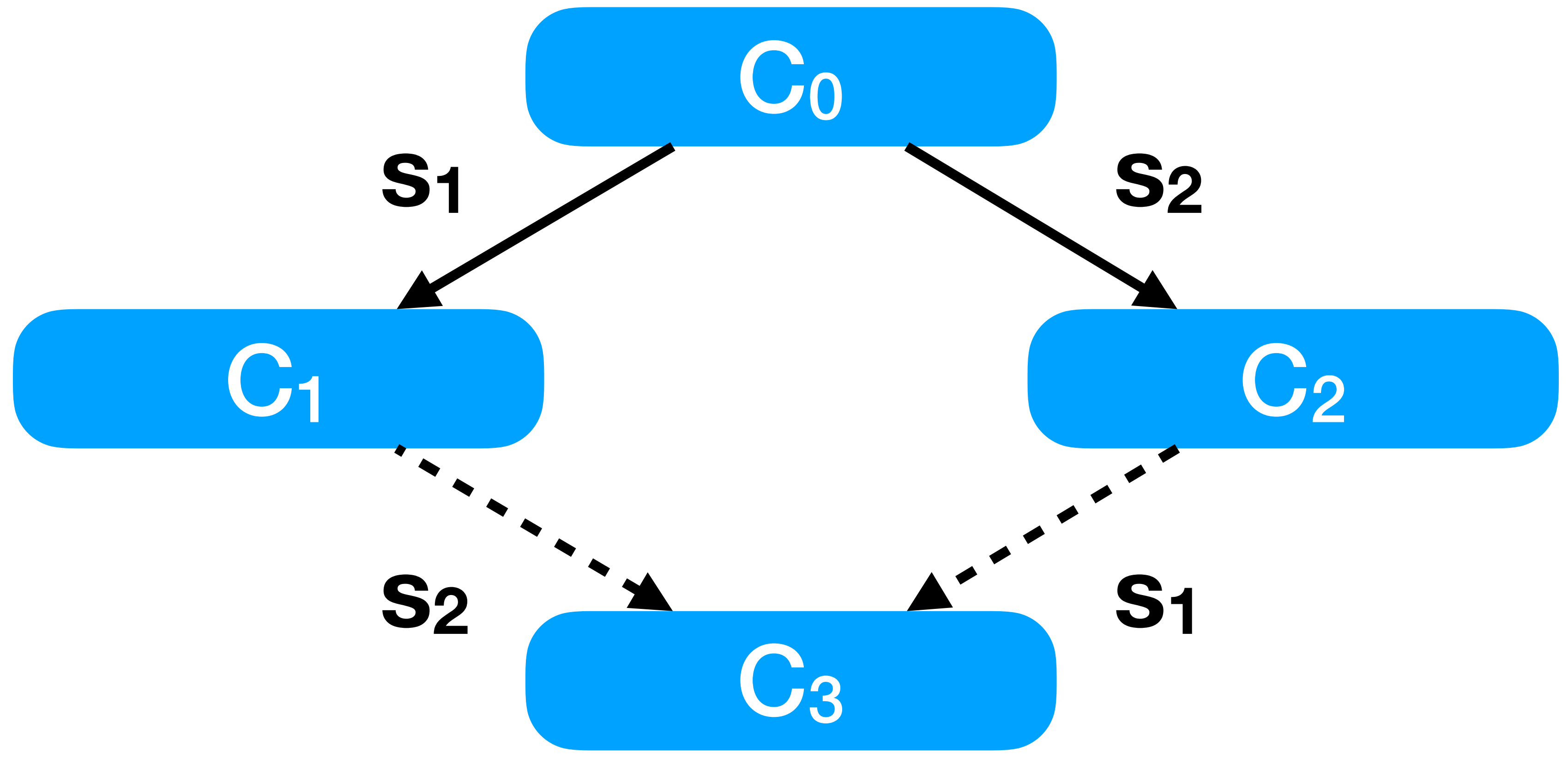
- A set of $N \geq 2$ *processes*
- **Internal state** of a process p:
  - one-bit input register $x_p \in \{0,1\}$
  - output register $y_p \in \{?, 0, 1\}$, initially '?'
- **Deterministic transition** function
  - receive a message, modify its internal state, and send messages
- Write-once: when y_p is in a **decision state** (0 or 1), its value cannot change

Configurations

- **Configuration**: the global state of the system at some point in time
  - internal state of each process + global message buffer
- A transition **step** or **event** takes one configuration to another: $e(C_0) = C_1$
  - e = ⟨p,m⟩: “process p received message m”
- A configuration is **reachable** if it can be accessed from some initial configuration
- A **schedule** is a sequence of events s = ⟨e1, e2, …, en, …⟩
  - the associated sequence of steps is a **run**

Lemma: commutativity of disjoint schedules

- Given:
  - a starting configuration C_0
  - schedules s_1 and s_2 that lead to C_1 and C_2: $s_1(C_0) = C_1 \land s_2(C_0) = C_2$
- Lemma: If the set of processors in s_1 and s_2 are disjoint, then $s_1(C_2) = s_2(C_1) = C_3$



Correct protocols

- A consensus protocol is **partially correct** if satisfies Agreement + Validity
  - Agreement: no reachable configuration has more than one decision value
  - Validity: some reachable configuration has decision value v ∈ {0,1}
- A run is **valid** when
  - at most one process is faulty, and
  - all messages (to non-faulty processes) are eventually received
- A consensus protocol is **totally correct in spite of one fault** if
  - it is partially correct, and
  - Termination: every valid run is a deciding run (some process reaches a decision state)

Decision values of reachable configurations

- A configuration is
  - **bivalent** if it is possible to reach a configuration with decision value 0 or 1
  - **0-valent** if the only reachable configurations have decision value 0
  - **1-valent** if the only reachable configurations have decision value 1
  - **univalent** if it is 0-valent of 1-valent
- **Bivalent means the decision outcome is unpredictable**

FLP main theorem

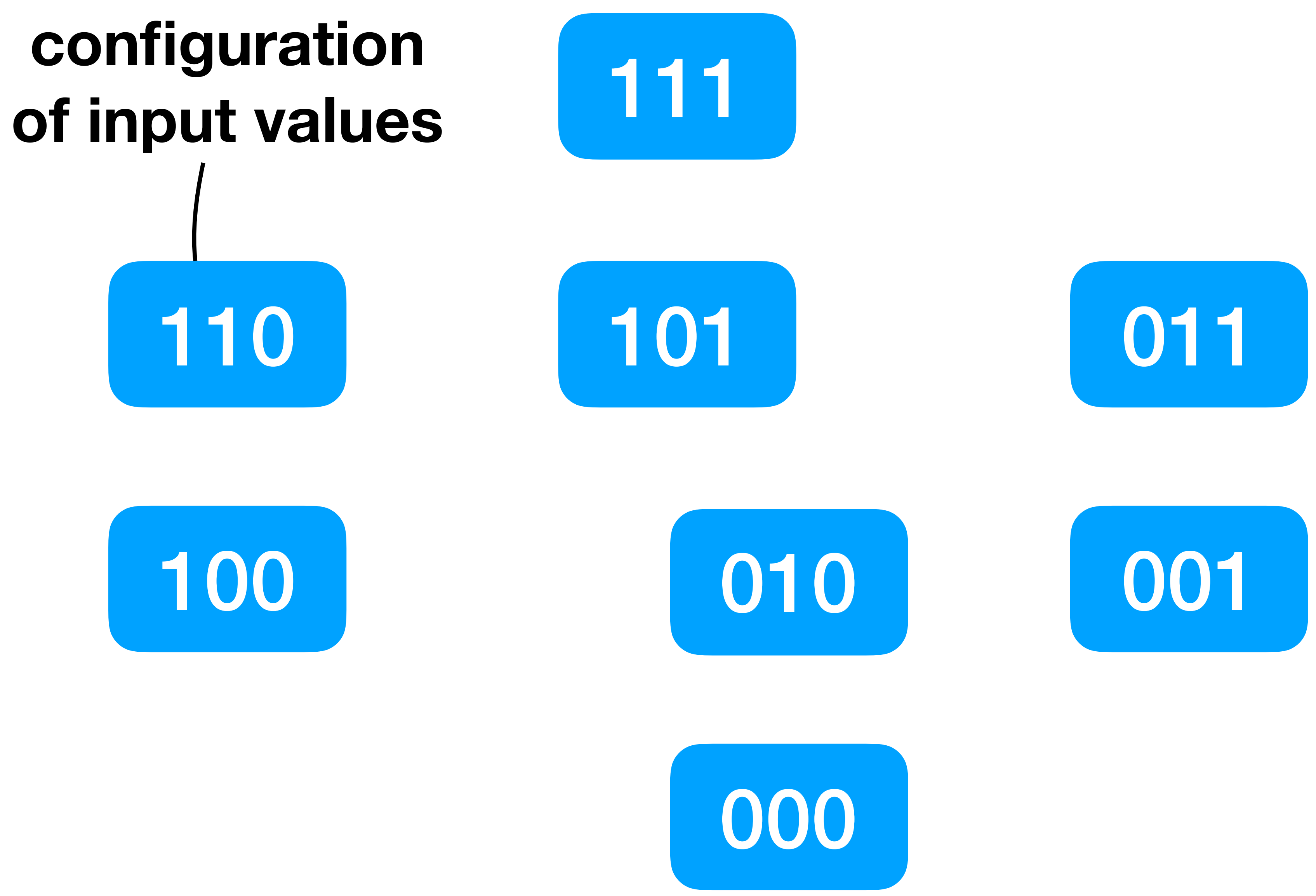
- Theorem: **No consensus protocol P is totally correct in spite of one fault**
- Proof idea: **it is always possible to remain in a bivalent (undecided) state (livelock)**

FLP main theorem

- Theorem: **No consensus protocol P is totally correct in spite of one fault**
- Proof idea: **it is always possible to remain in a bivalent (undecided) state (livelock)**
- Proof outline: By contradiction, assume that P is totally-correct in spite of one fault
- Suffices to prove, by induction, that
    1. Base case: **exists a bivalent initial configuration**
    2. Inductive step: **exists an admissible run that avoids ever taking a decision step**
        - decision step = a step that would commit the system to a particular decision

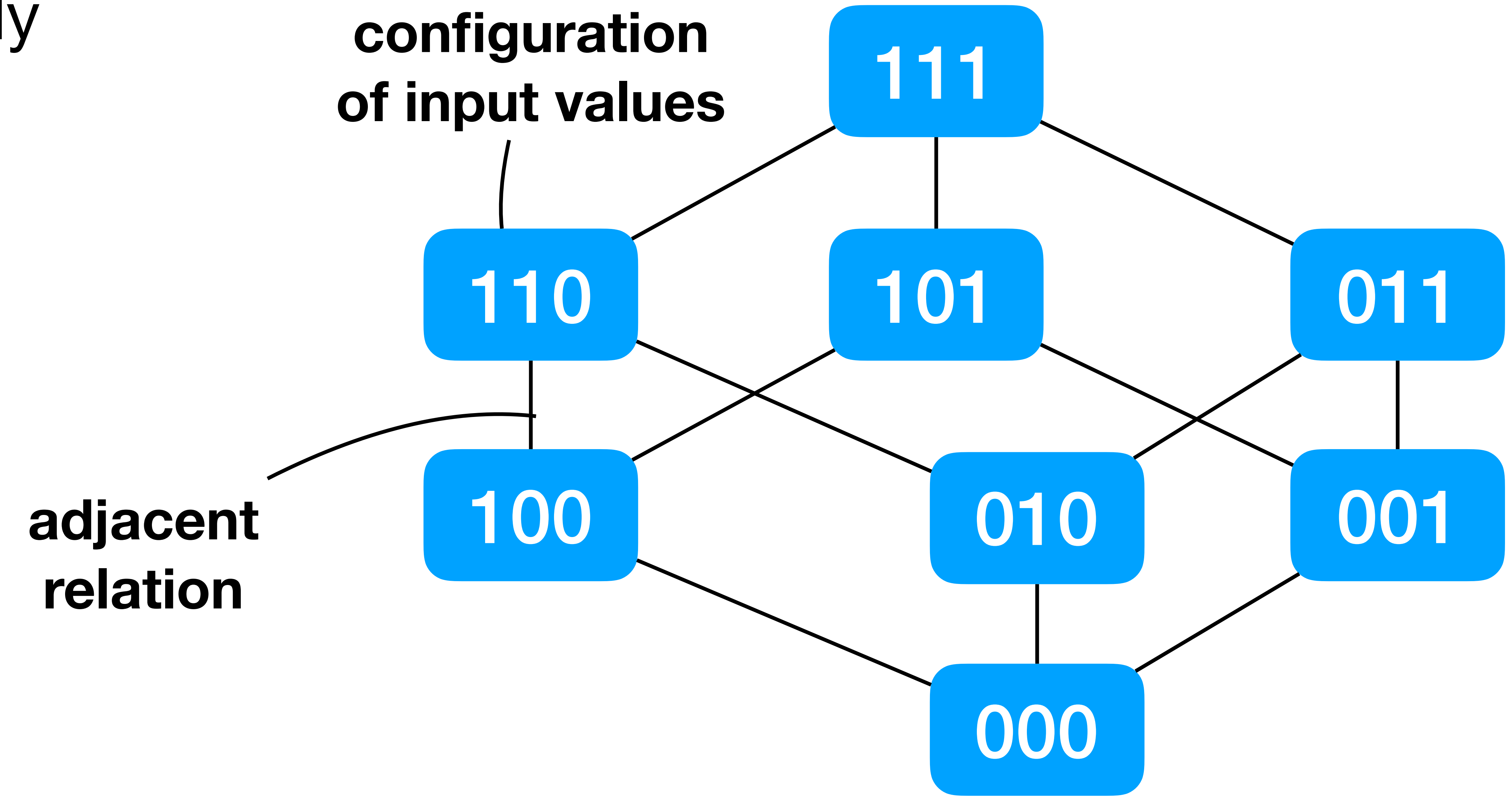
∃ bivalent initial configuration [1/2]

- **With N processes, there are 2^N possible initial configurations**
  - For each process p, input value $x_p \in \{0,1\}$ and output value $y_p = ?$



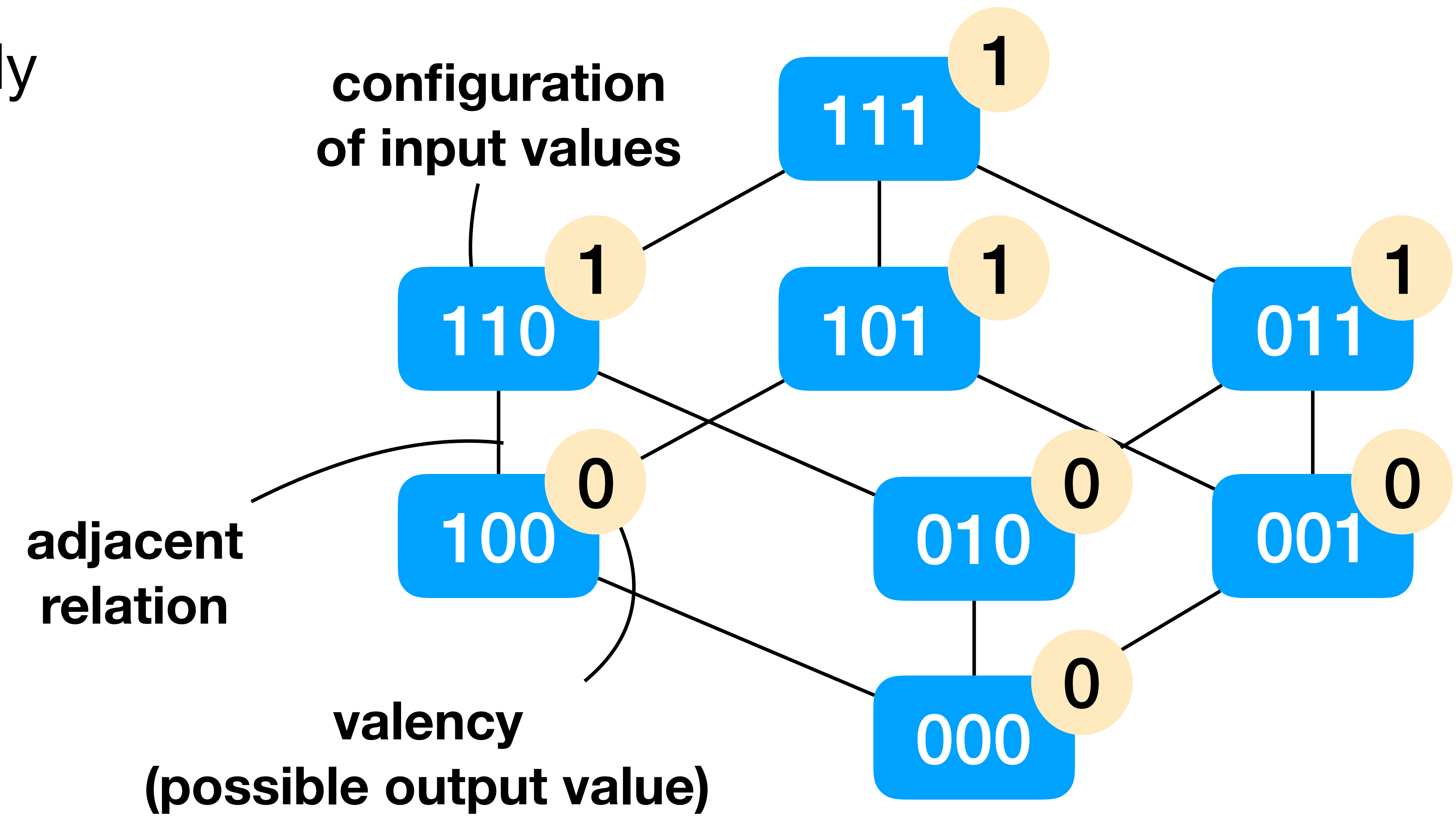
∃ bivalent initial configuration [1/2]

- **With N processes, there are 2^N possible initial configurations**
    - For each process p, input value $x_p \in \{0,1\}$ and output value $y_p = ?$
- **We can build a lattice of adjacent configurations**
    - Two configurations are *adjacent* if they differ only in the input value of one process



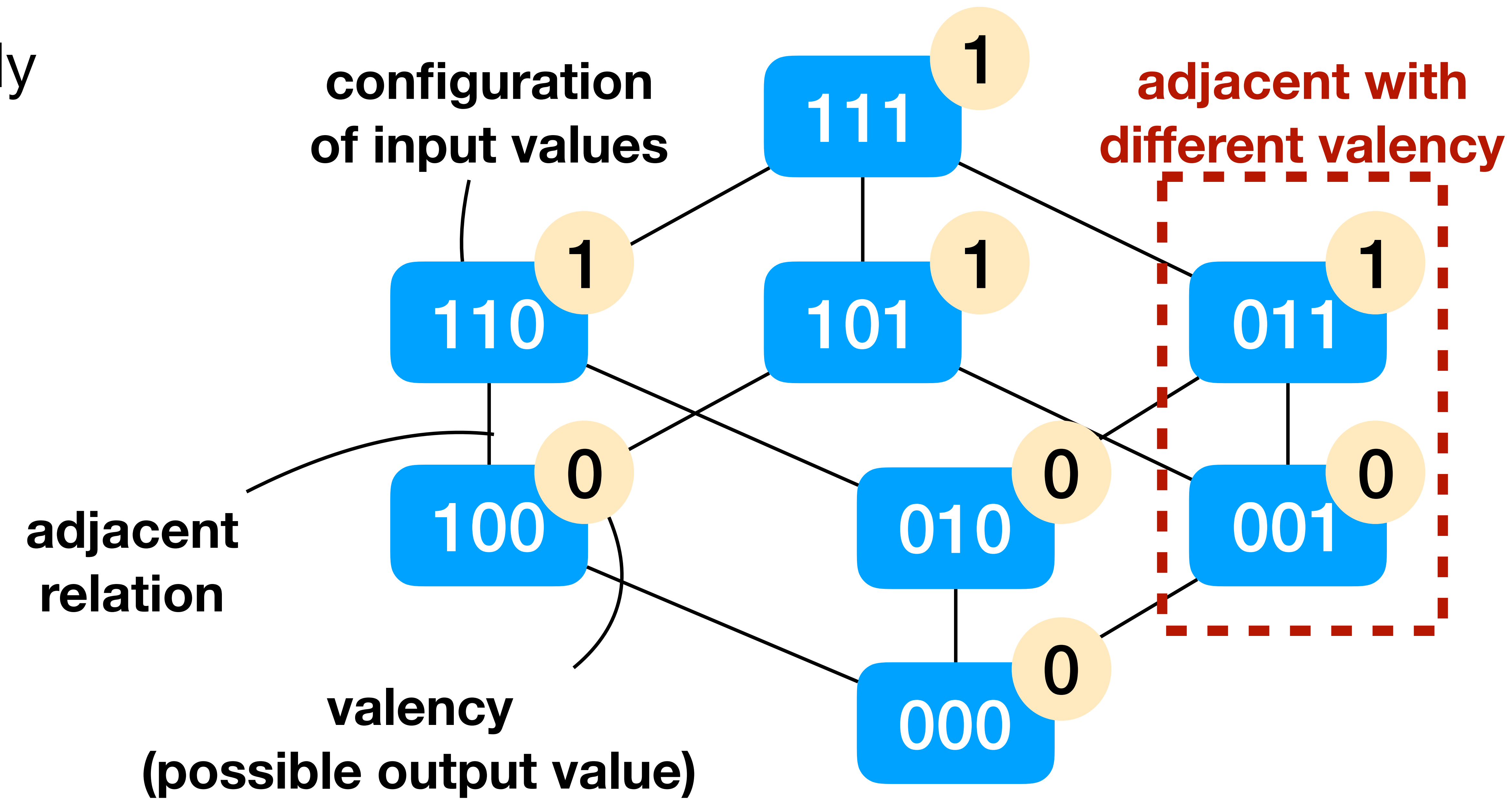
∃ bivalent initial configuration [1/2]

- **With N processes, there are 2^N possible initial configurations**
  - For each process p, input value $x_p \in \{0,1\}$ and output value y_p = ?
- **We can build a lattice of adjacent configurations**
  - Two configurations are *adjacent* if they differ only in the input value of one process
- Proof by contradiction
  - Assume each configuration is univalent, by Validity property
  - Valencies are assigned arbitrarily; we don't know how the protocol is defined

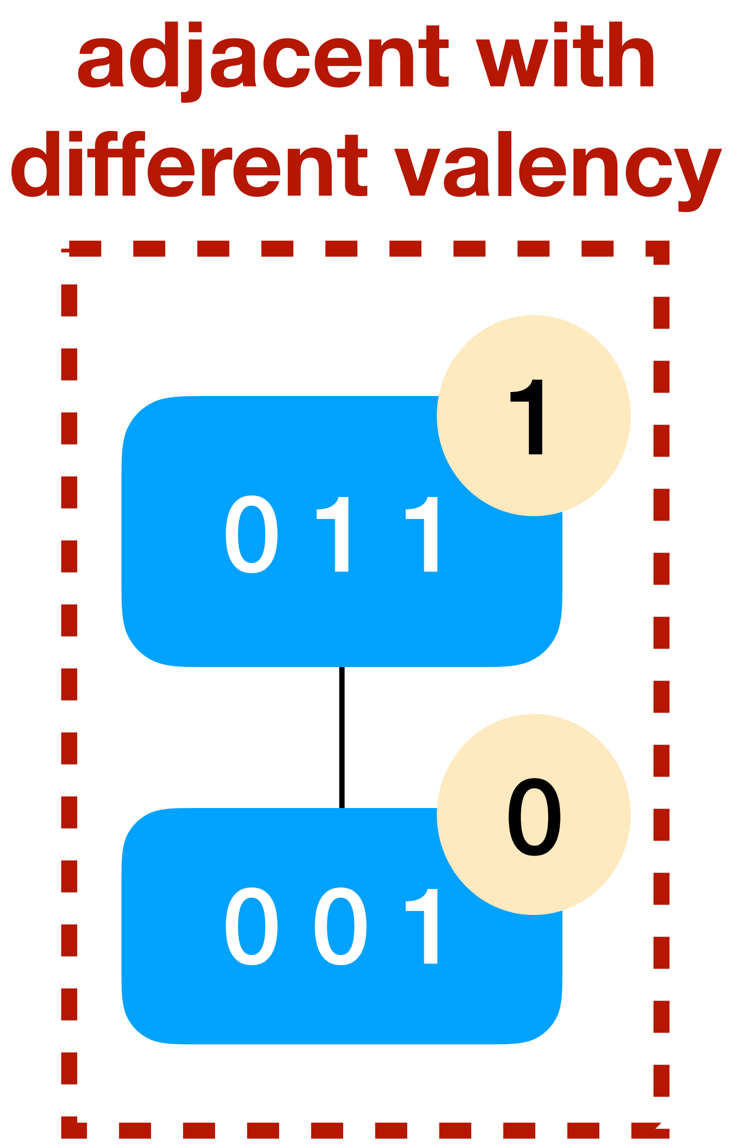


∃ bivalent initial configuration [1/2]

- **With N processes, there are 2^N possible initial configurations**
  - For each process p, input value $x_p \in \{0,1\}$ and output value y_p = ?
- **We can build a lattice of adjacent configurations**
  - Two configurations are *adjacent* if they differ only in the input value of one process
- <u>Proof by contradiction</u>
  - Assume each configuration is univalent, by Validity property
  - Valencies are assigned arbitrarily; we don't know how the protocol is defined
  - We know there will be at least one pair of adjacent configurations with different valency

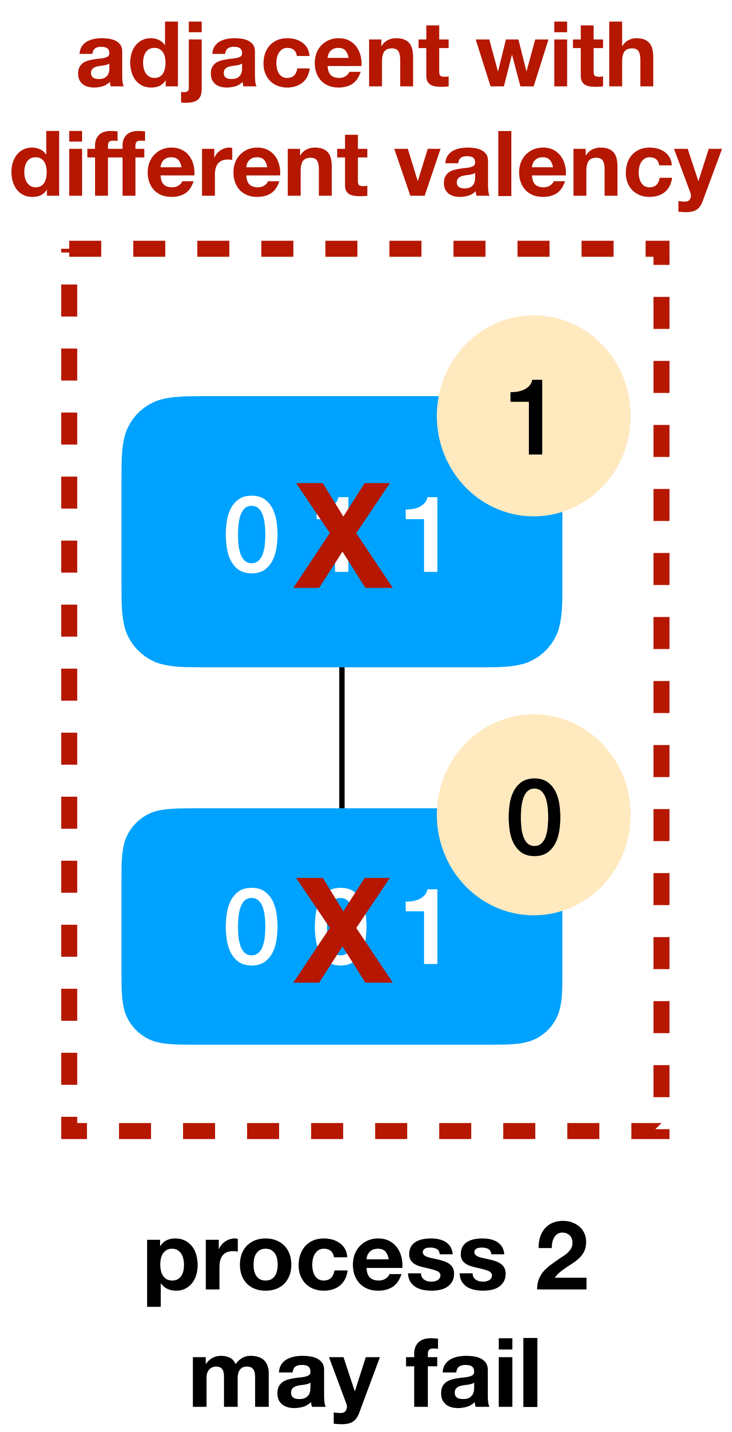


∃ bivalent initial configuration [2/2]



- The process that makes the configurations adjacent may fail

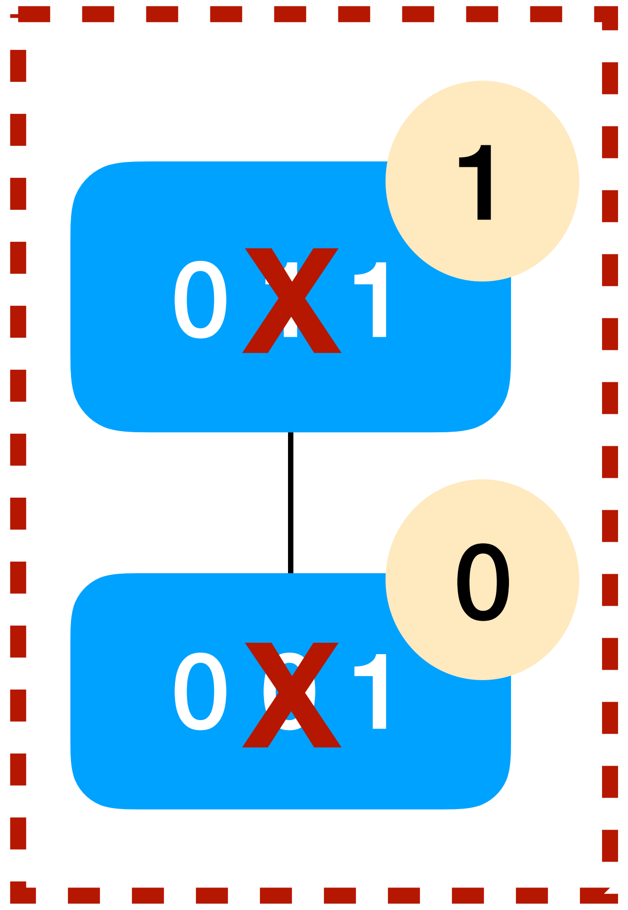
∃ bivalent initial configuration [2/2]



- The process that makes the configurations adjacent may fail
- We obtain two identical configurations with different outcomes

∃ bivalent initial configuration [2/2]

adjacent with different valency



process 2 may fail

- The process that makes the configurations adjacent may fail
- We obtain two identical configurations with different outcomes
- The protocol is deterministic: both initial configurations must lead to the same decision value v
- In case v = 0 or v = 1, one of the configurations is bivalent: Contradiction!
- Then, there must exists a bivalent initial configuration
  QED

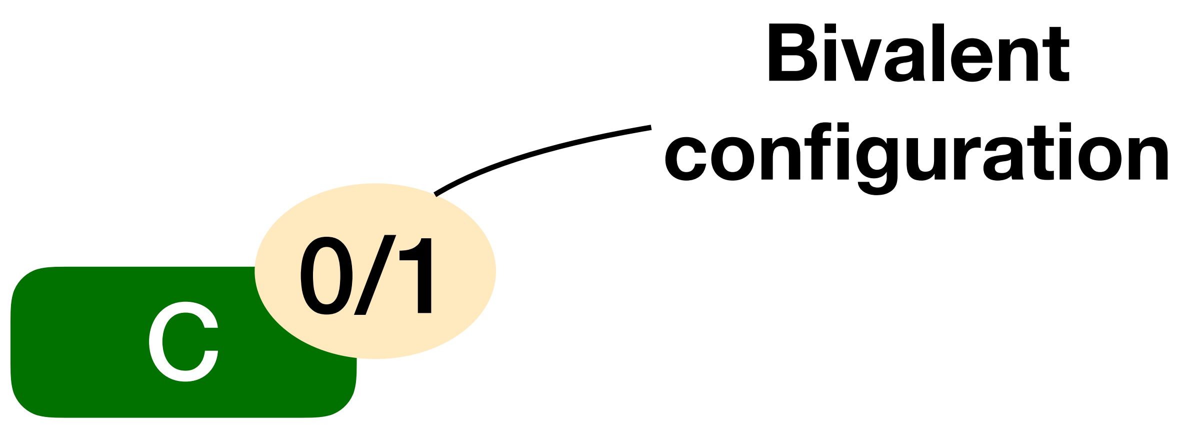
∃ reachable bivalent configuration [1/6]

- Informally:
  - **Starting from a bivalent (undecided) configuration C, there is always a reachable bivalent configuration**
- There is always a way to avoid reaching consensus

∃ reachable bivalent configuration [2/6]

Theorem:

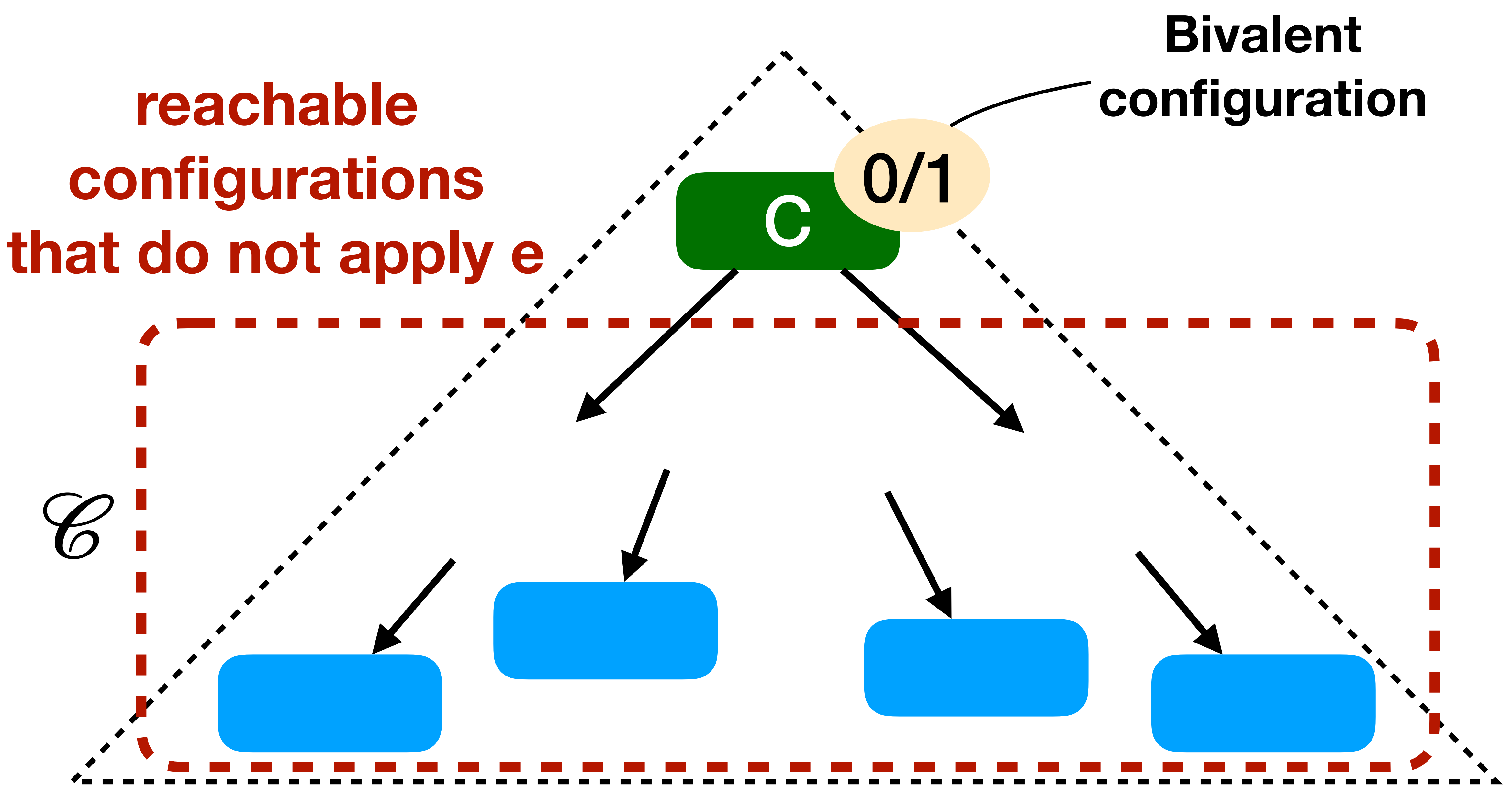
- Assume we have:
  - a bivalent configuration C



∃ reachable bivalent configuration [2/6]

<u>Theorem:</u>

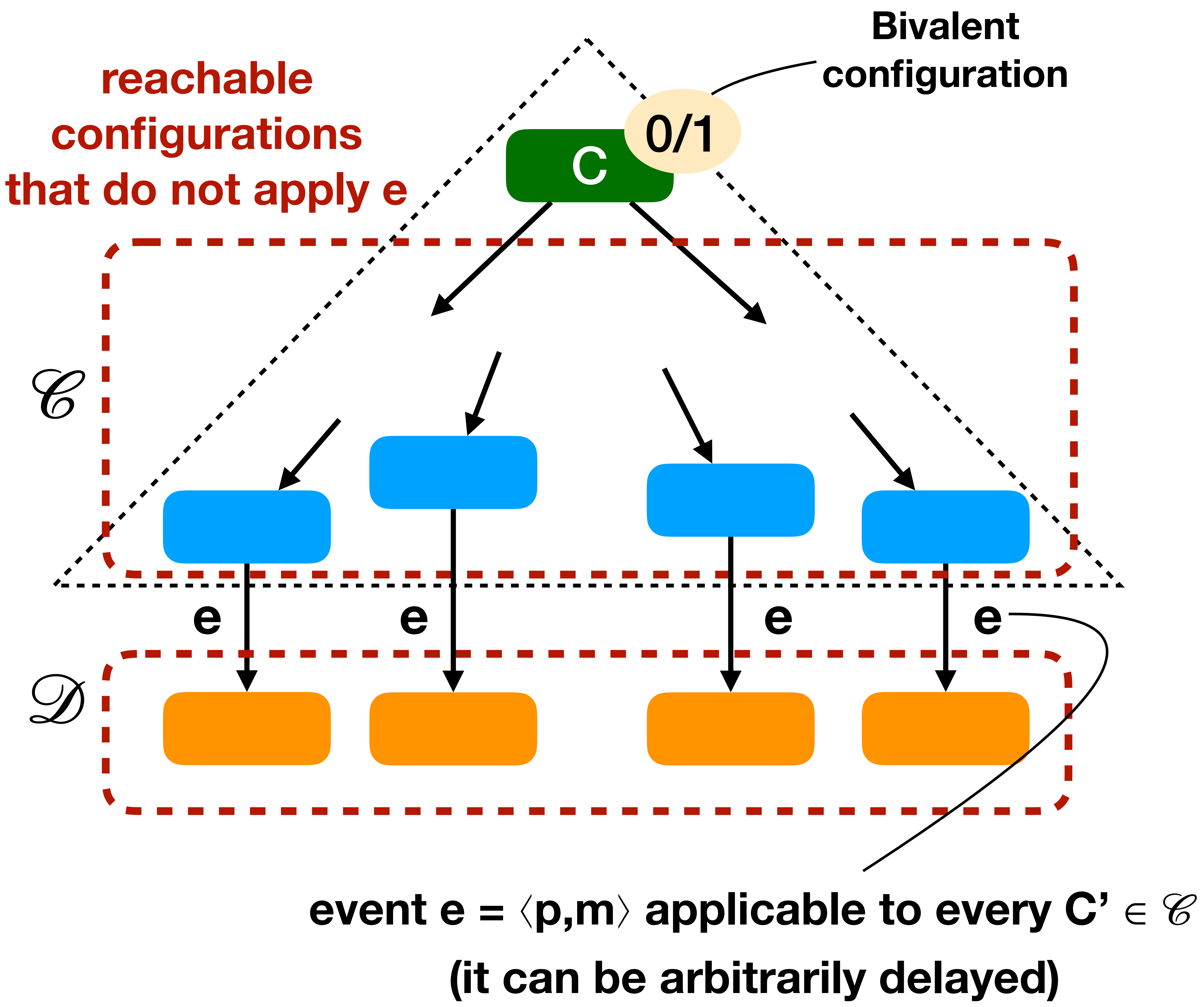
- Assume we have:
  - a bivalent configuration C
  - an event e = ⟨p,m⟩ that is applicable to C
  - $\mathscr{C}$ = the configurations reachable from C *without* applying e



∃ reachable bivalent configuration [2/6]

Theorem:

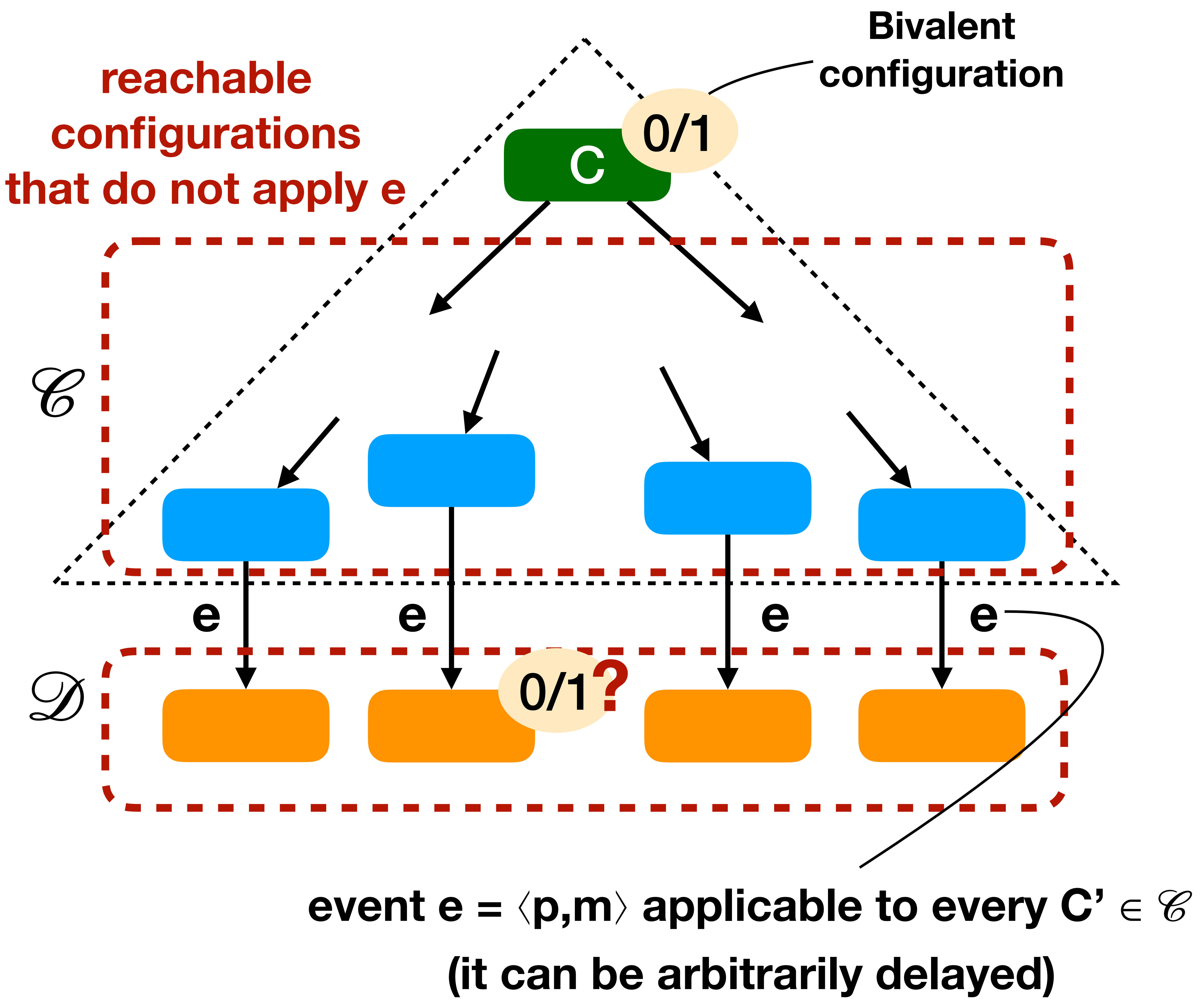
- Assume we have:
  - a bivalent configuration C
  - an event e = ⟨p,m⟩ that is applicable to C
  - $\mathscr{C}$ = the configurations reachable from C *without* applying e
  - $\mathscr{D}$ = the configurations { e(E) : E ∈ $\mathscr{C}$}



∃ reachable bivalent configuration [2/6]

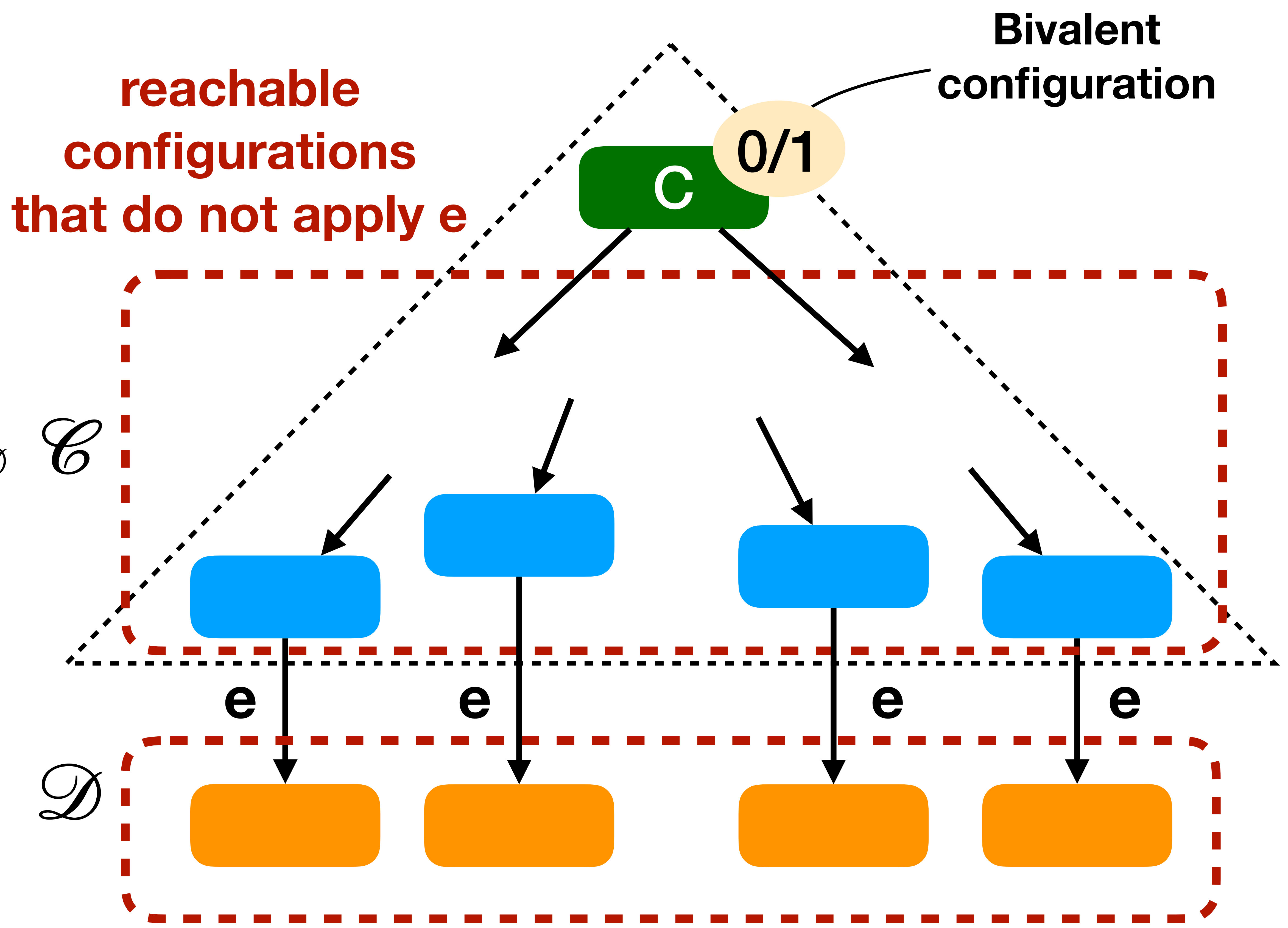
Theorem:

- Assume we have:
  - a bivalent configuration C
  - an event e = ⟨p,m⟩ that is applicable to C
  - $\mathscr{C}$ = the configurations reachable from C *without* applying e
  - $\mathscr{D}$ = the configurations { e(E) : E ∈ $\mathscr{C}$}
- Prove that $\mathscr{D}$ contains a bivalent configuration



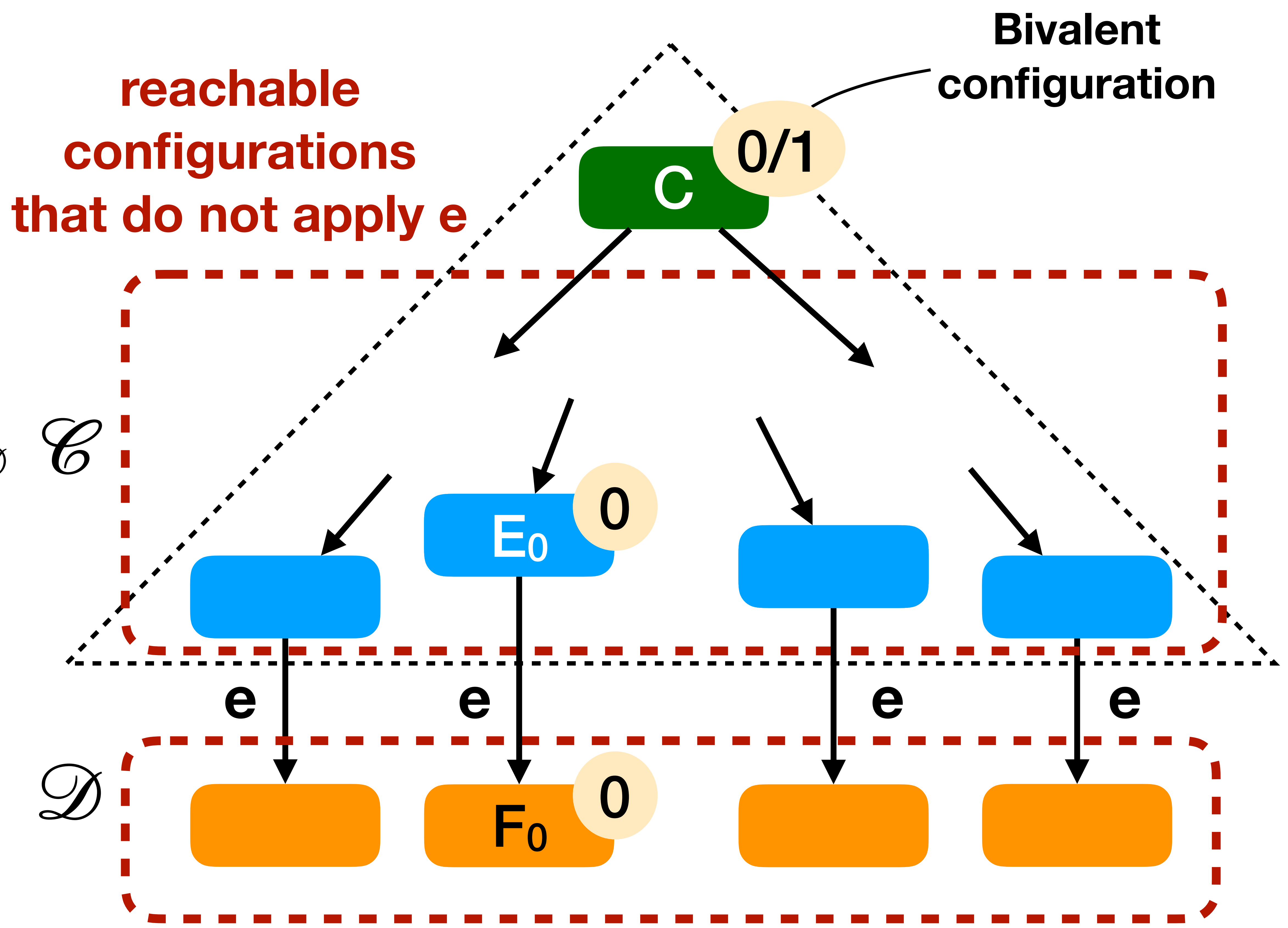
∃ reachable bivalent configuration [3/6]

- Claim: $\mathscr{D}$ contains a bivalent configuration
- Proof: by contradiction
- Assume $\mathscr{D}$ has no bivalent configuration
- There are univalent reachable configurations from C in $\mathscr{D}$



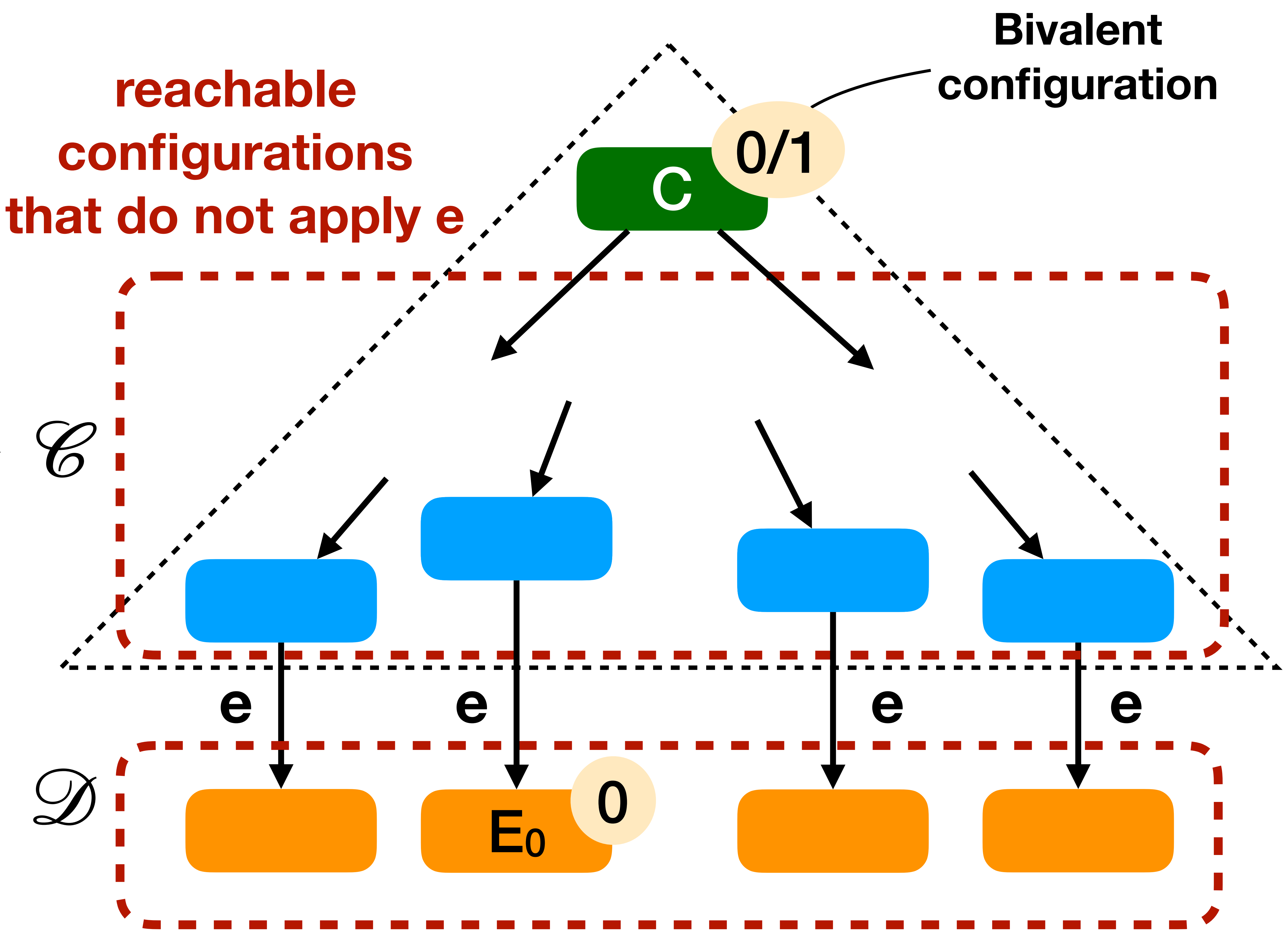
∃ reachable bivalent configuration [3/6]

- Claim: $\mathscr{D}$ contains a bivalent configuration
- Proof: by contradiction
- Assume $\mathscr{D}$ has no bivalent configuration
- There are univalent reachable configurations from C in $\mathscr{D}$
  - Pick E_0, 0-valent and reachable from C
  - If $E_0 \in \mathscr{C}$, ∃ $F_0 \in \mathscr{D}$, and F_0 is also 0-valent



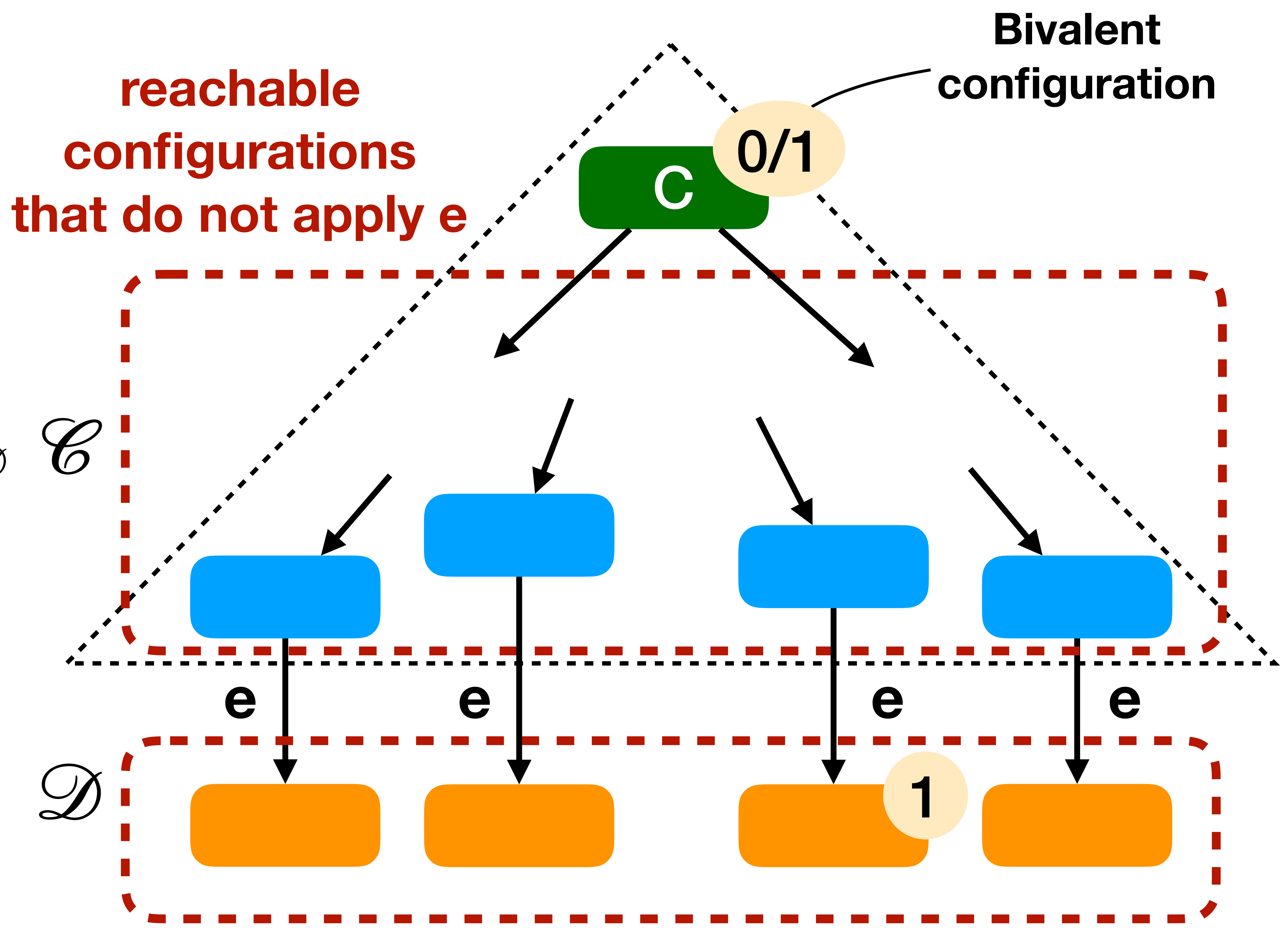
∃ reachable bivalent configuration [3/6]

- Claim: $\mathscr{D}$ contains a bivalent configuration
- Proof: by contradiction
- Assume $\mathscr{D}$ has no bivalent configuration
- There are univalent reachable configurations from C in $\mathscr{D}$
    - Pick E_0, 0-valent and reachable from C
    - If $E_0 \in \mathscr{C}$, $\exists\, F_0 \in \mathscr{D}$, and F_0 is also 0-valent
    - If $E_0 \notin \mathscr{C}$, then e was applied, and $E_0 \in \mathscr{D}$



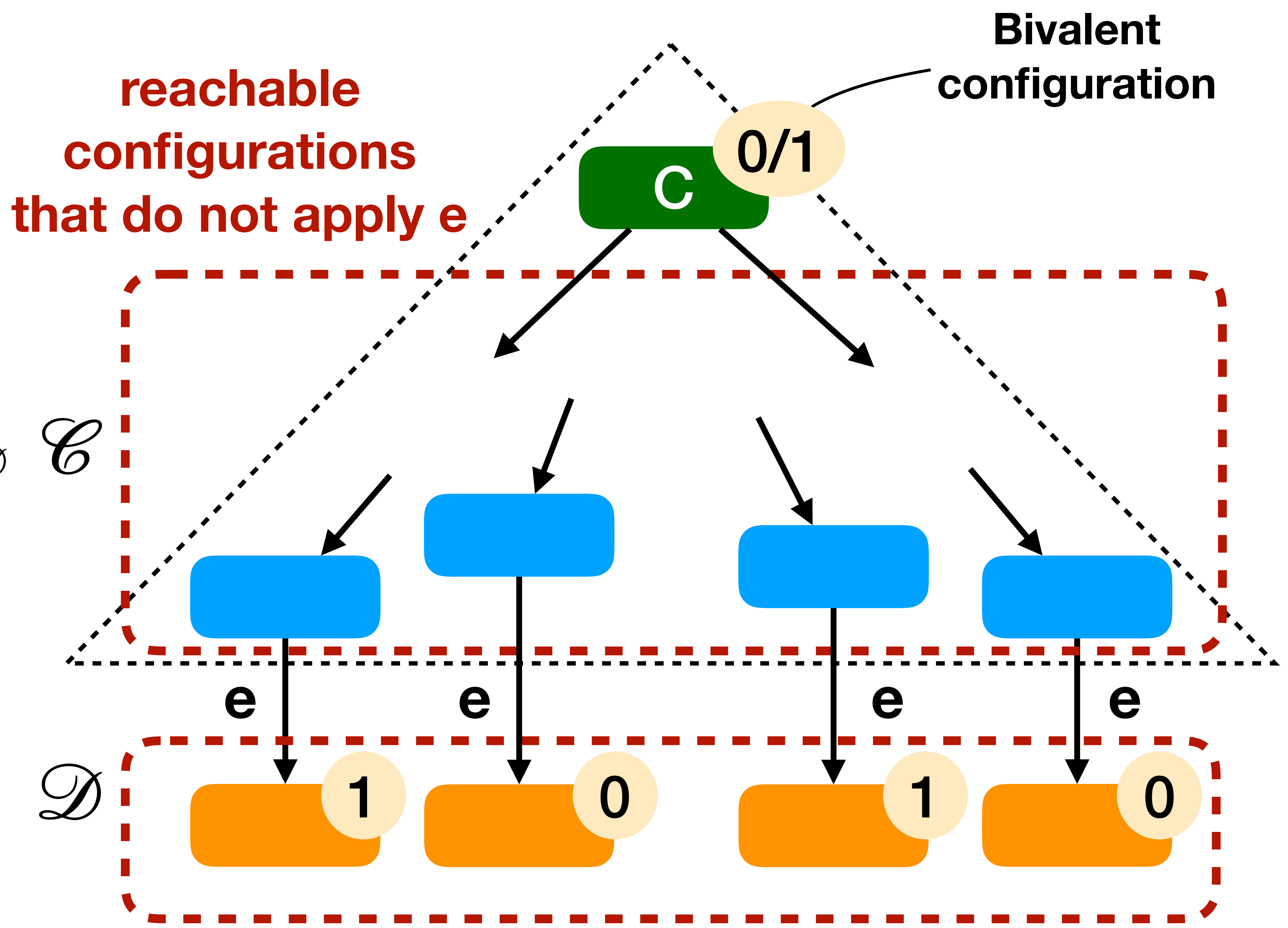
∃ reachable bivalent configuration [3/6]

- Claim: $\mathscr{D}$ contains a bivalent configuration
- Proof: by contradiction
- Assume $\mathscr{D}$ has no bivalent configuration
- There are univalent reachable configurations from C in $\mathscr{D}$
  - Pick E_0, 0-valent and reachable from C
  - If $E_0 \in \mathscr{C}$, $\exists\ F_0 \in \mathscr{D}$, and F_0 is also 0-valent
  - If $E_0 \notin \mathscr{C}$, then e was applied, and $E_0 \in \mathscr{D}$
  - Similar for 1-valent configurations



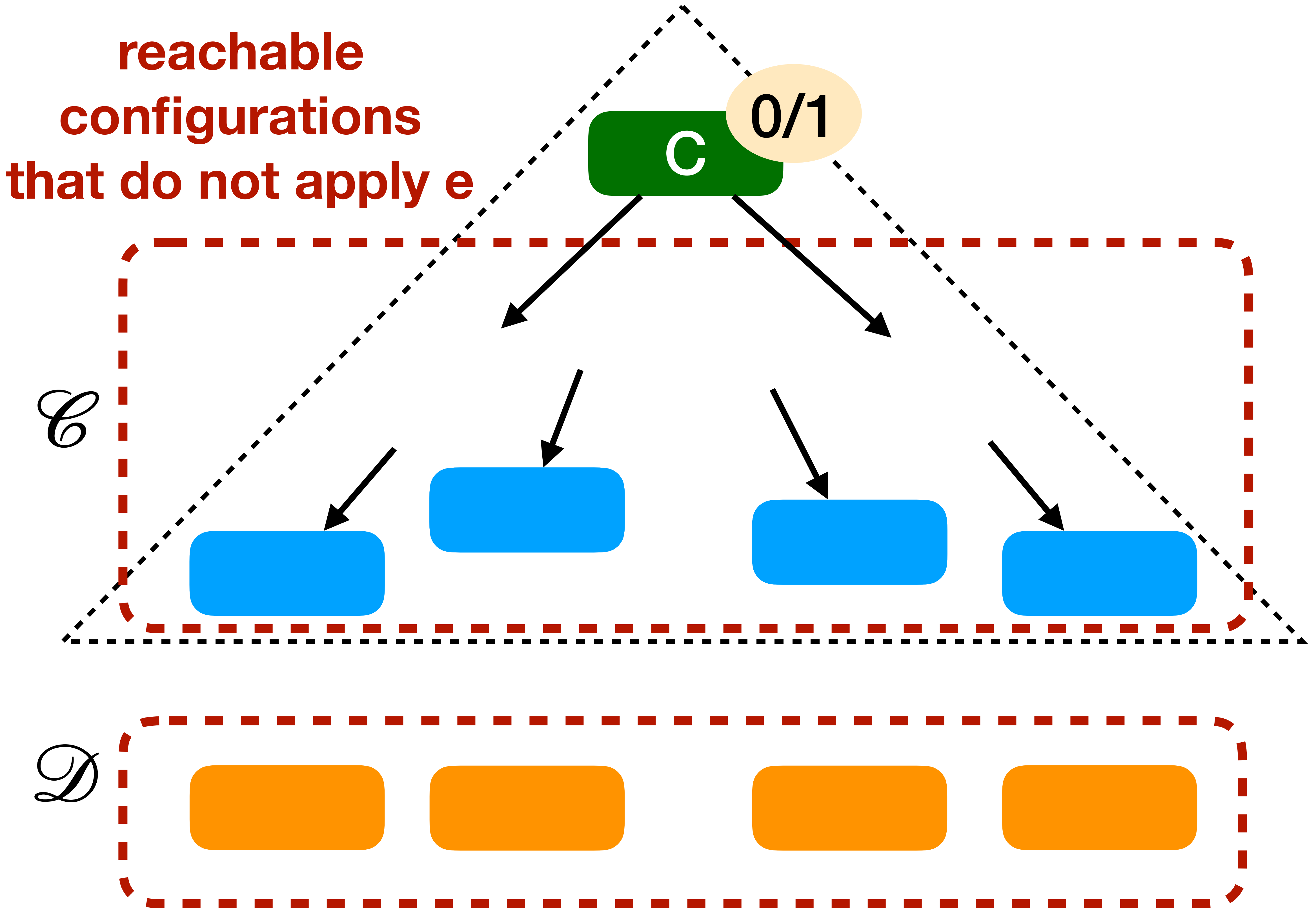
∃ reachable bivalent configuration [3/6]

- <u>Claim:</u> $\mathscr{D}$ contains a bivalent configuration
- <u>Proof:</u> by contradiction
- Assume $\mathscr{D}$ has no bivalent configuration
- There are univalent reachable configurations from C in $\mathscr{D}$
    - Pick E_0, 0-valent and reachable from C
    - If $E_0 \in \mathscr{C}$, $\exists$ $F_0 \in \mathscr{D}$, and F_0 is also 0-valent
    - If $E_0 \notin \mathscr{C}$, then e was applied, and $E_0 \in \mathscr{D}$
    - Similar for 1-valent configurations
- **$\mathscr{D}$ contains both 0-valent and 1-valent configurations**



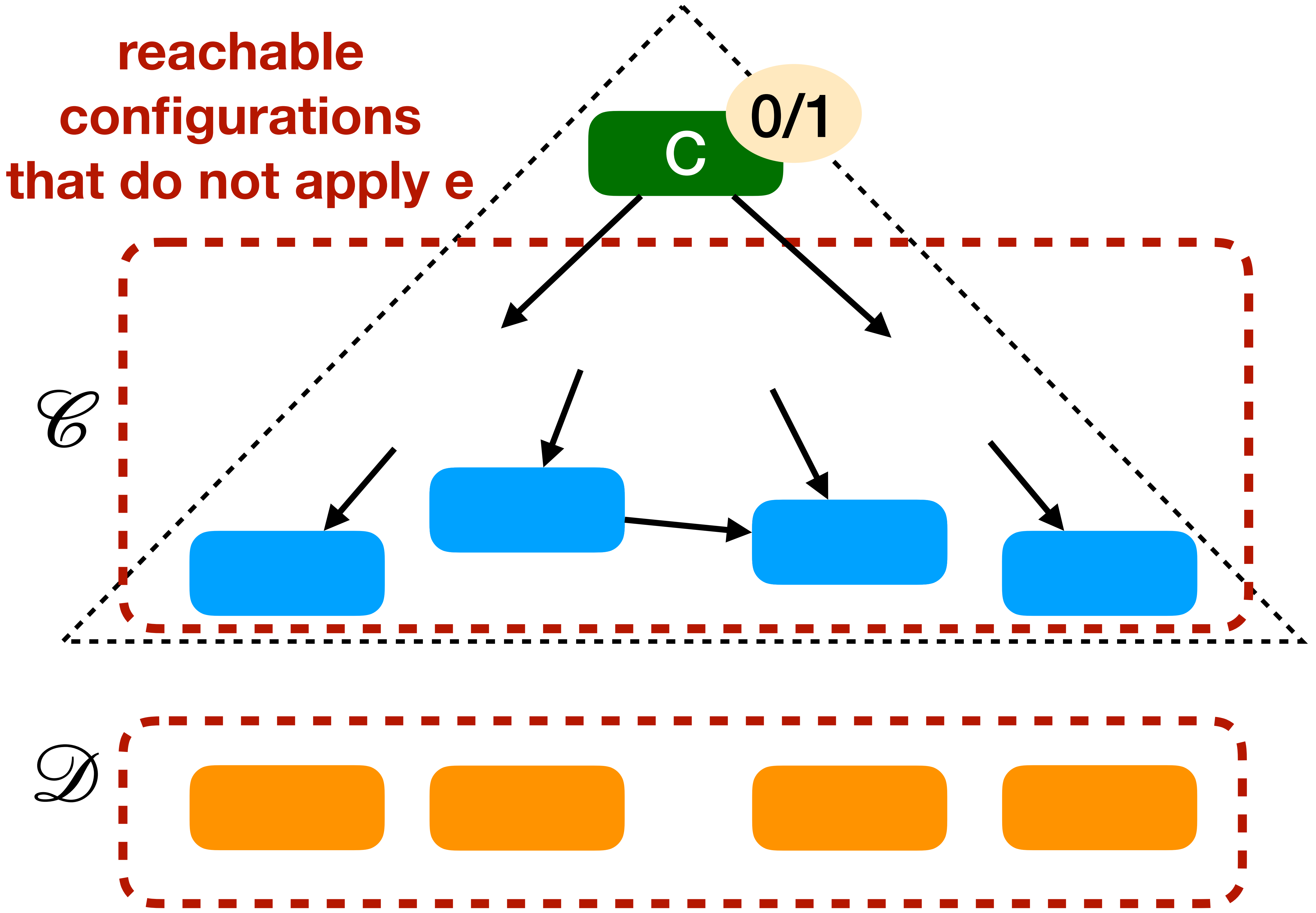
∃ reachable bivalent configuration [4/6]

- All configurations in $\mathscr{C}$ are linked by events other than e



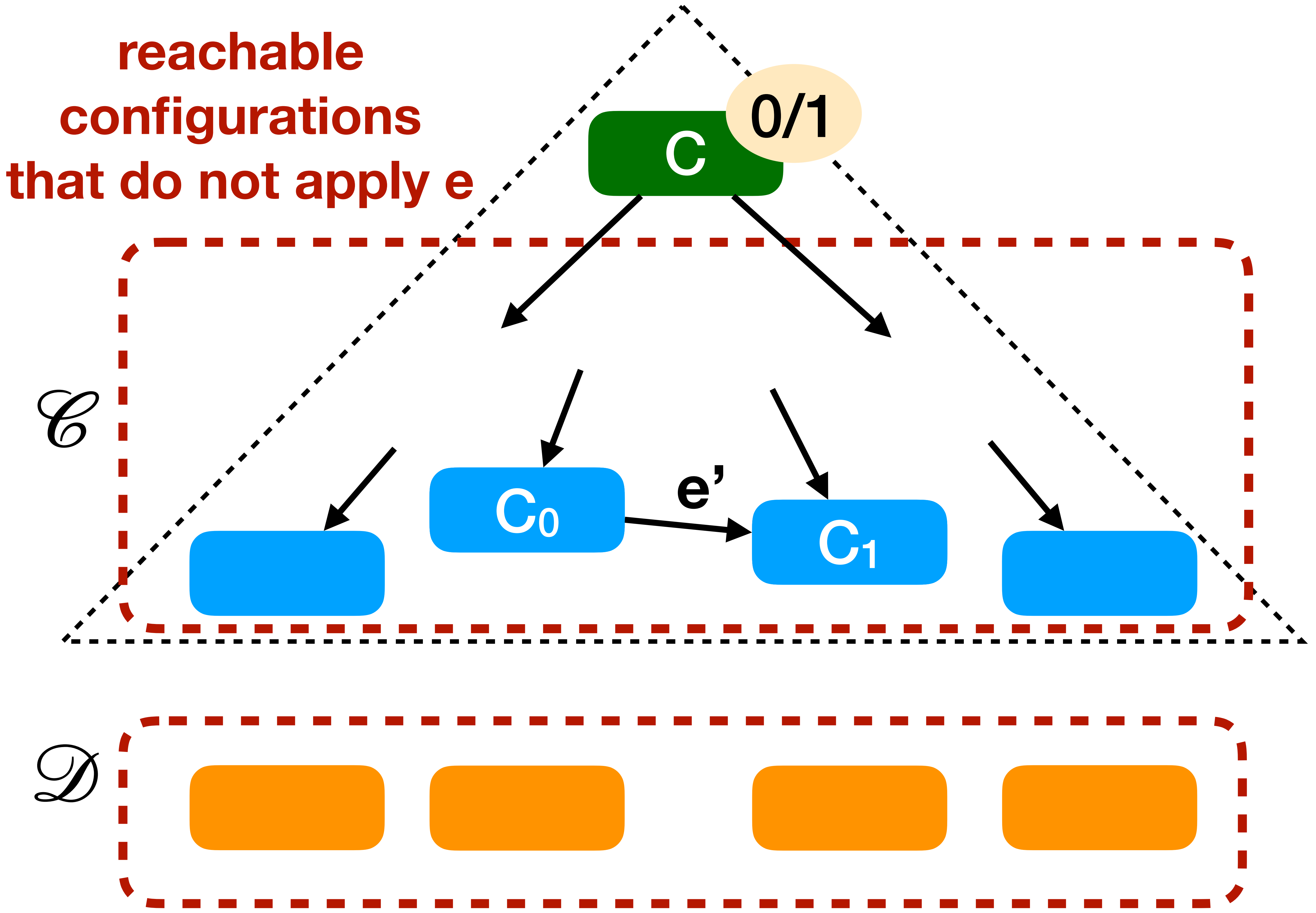
∃ reachable bivalent configuration [4/6]

- All configurations in $\mathscr{C}$ are linked by events other than e
- ∃ neighbor configurations in $\mathscr{C}$, by induction
  - Two configs are *neighbors* if one is followed by the other in one step



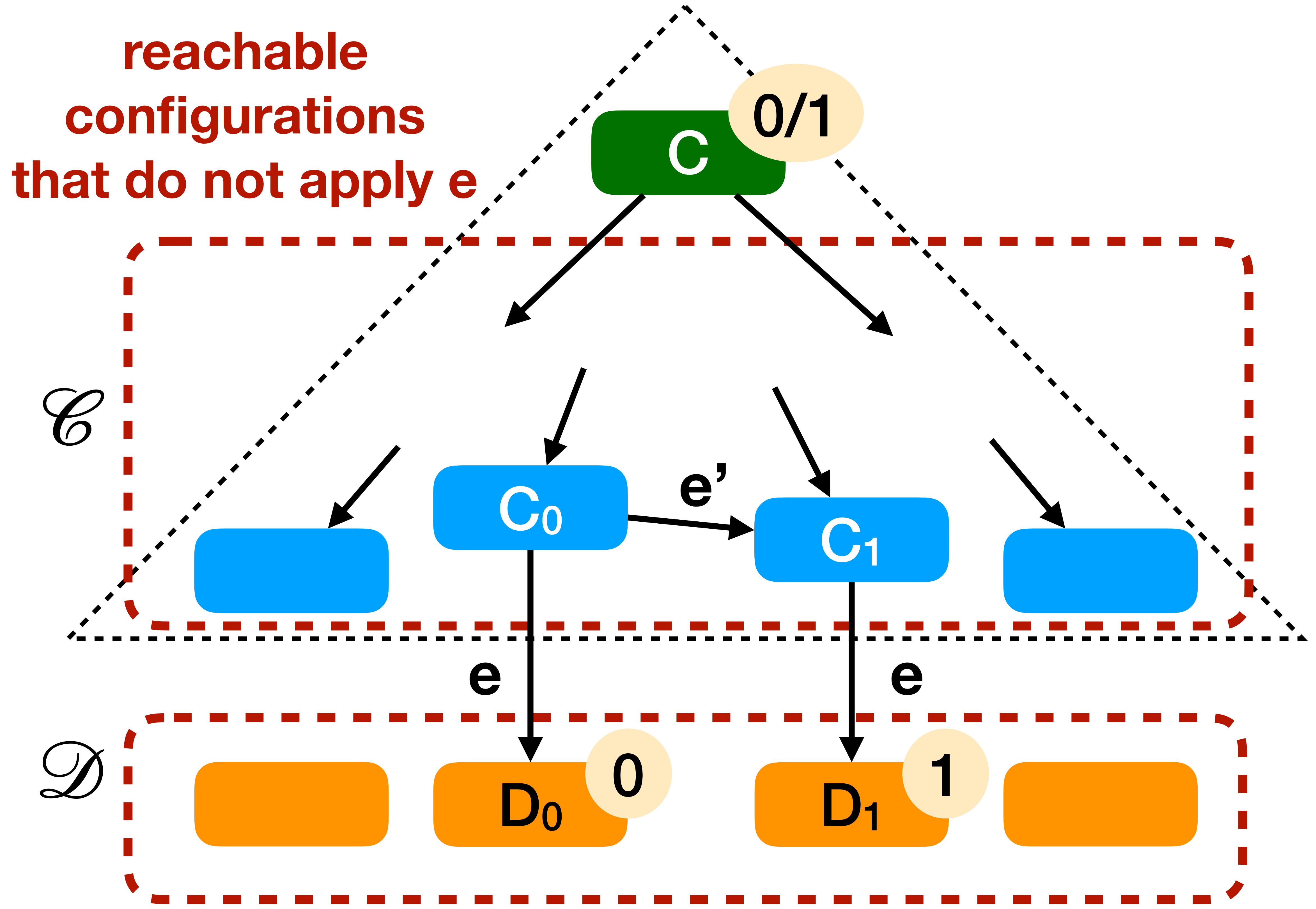
∃ reachable bivalent configuration [4/6]

- All configurations in $\mathscr{C}$ are linked by events other than e
- ∃ neighbor configurations in $\mathscr{C}$, by induction
  - Two configs are *neighbors* if one is followed by the other in one step
- For instance, C_0 and C_1, related by event e': $C_1 = e'(C_0)$



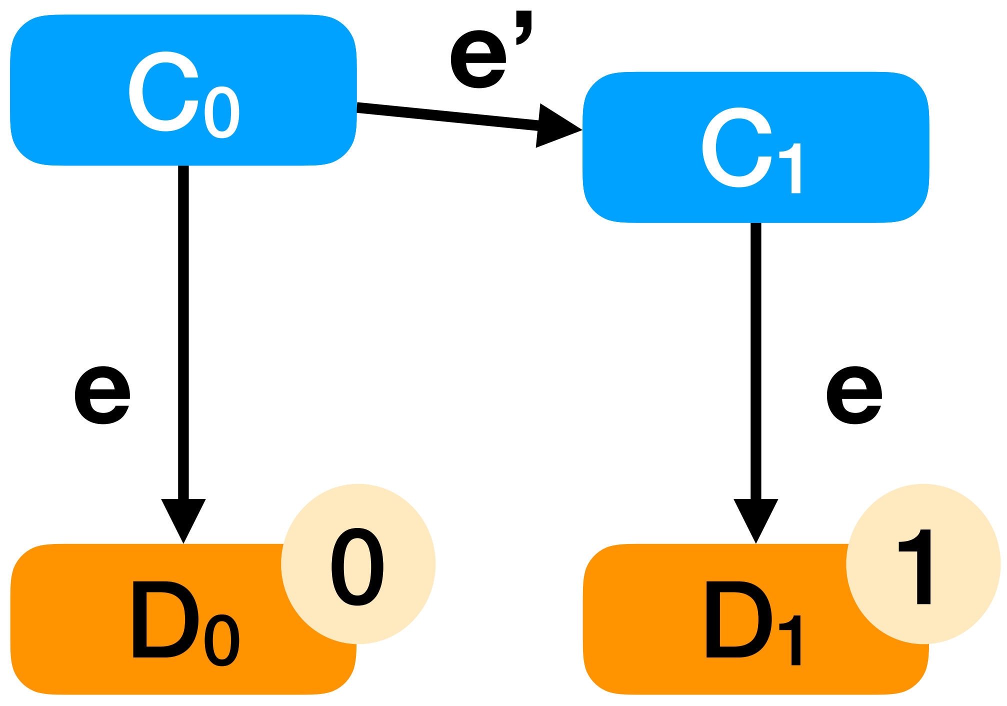
∃ reachable bivalent configuration [4/6]

- All configurations in $\mathscr{C}$ are linked by events other than e
- ∃ neighbor configurations in $\mathscr{C}$, by induction
  - Two configs are *neighbors* if one is followed by the other in one step
- For instance, C_0 and C_1, related by event e': $C_1 = e'(C_0)$
- $\exists\, D_i \in \mathscr{D}$, $D_i = e(D_i)$, $i \in \{0,1\}$

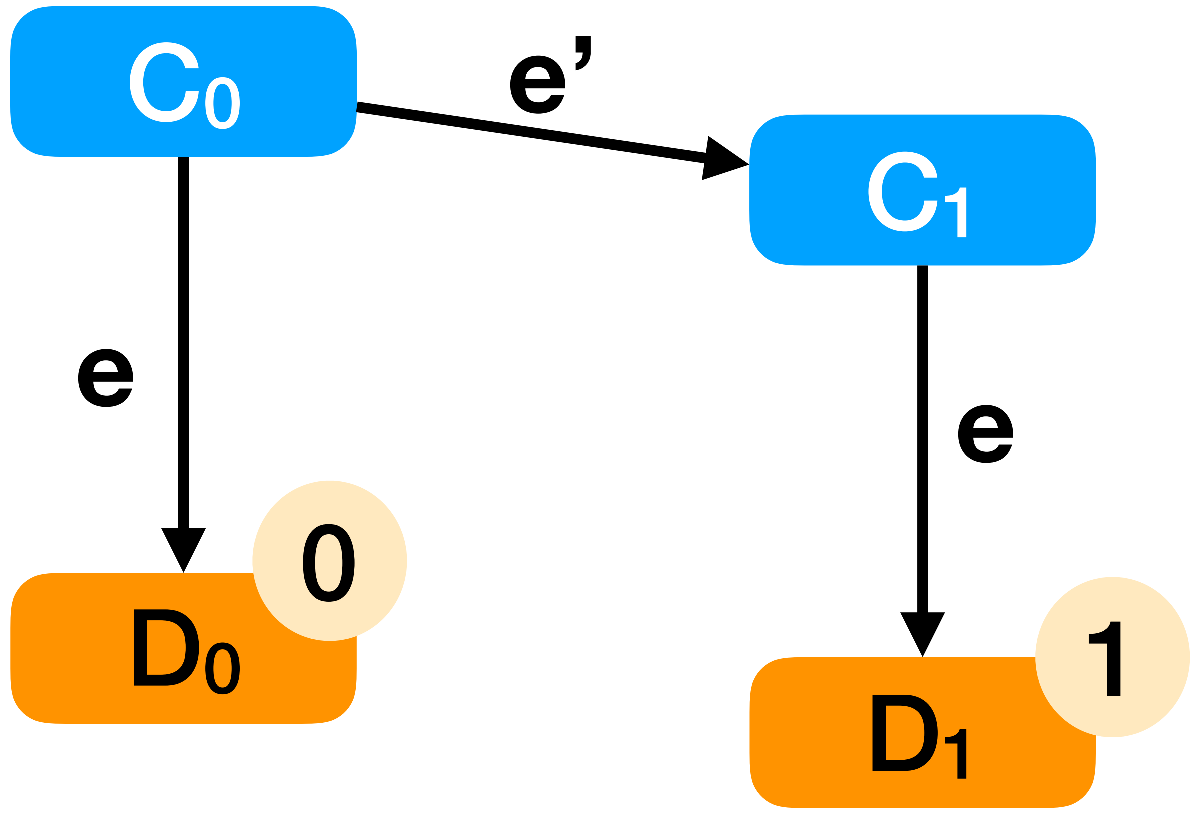


∃ reachable bivalent configuration [4/6]

- All configurations in $\mathscr{C}$ are linked by events other than e
- ∃ neighbor configurations in $\mathscr{C}$, by induction
  - Two configs are *neighbors* if one is followed by the other in one step
- For instance, C_0 and C_1, related by event e’: $C_1 = e'(C_0)$
- ∃ $D_i \in \mathscr{D}$, $D_i = e(D_i)$, $i \in \{0,1\}$

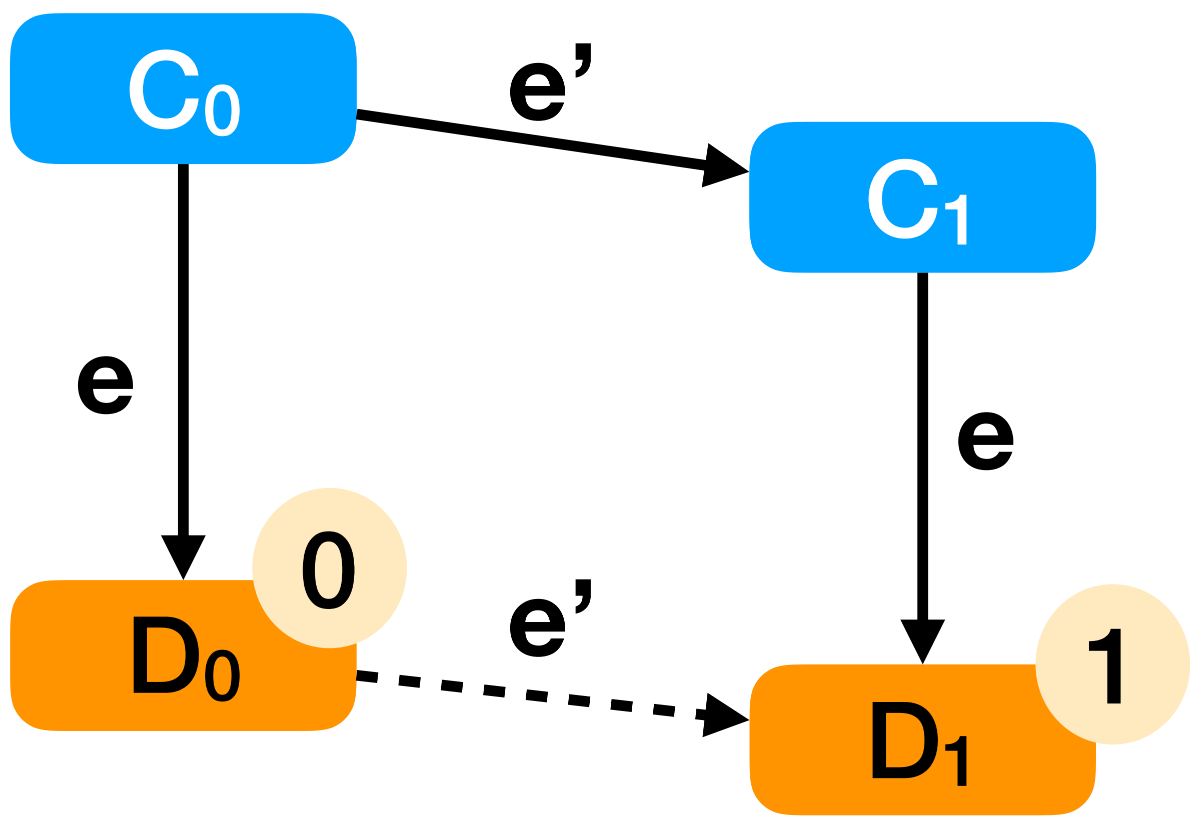


∃ reachable bivalent configuration [5/6]



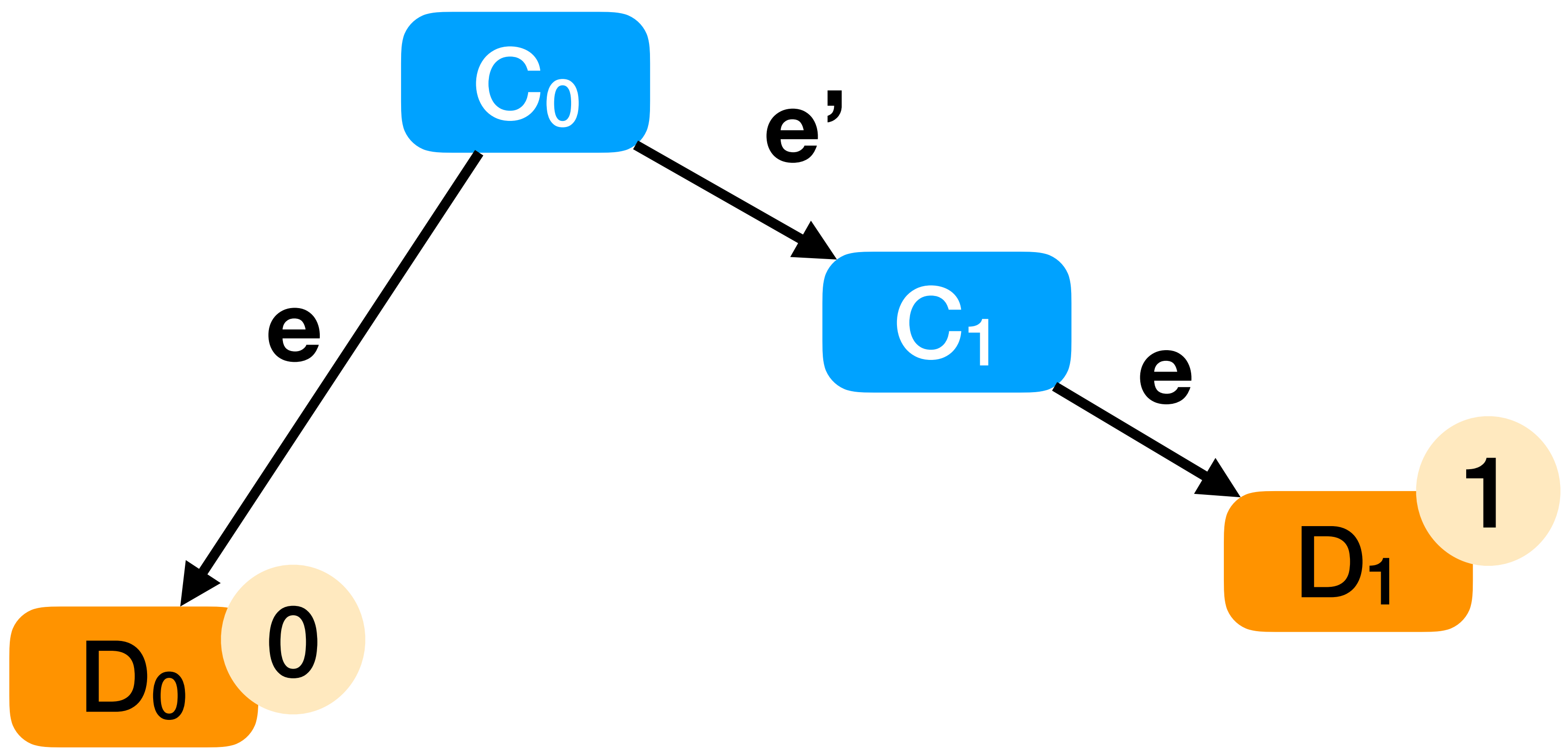
- Two events: $e = \langle p,m \rangle$ and $e' = \langle p',m' \rangle$
- Two cases: either process $p \neq p'$ or $p = p'$

∃ reachable bivalent configuration [5/6]



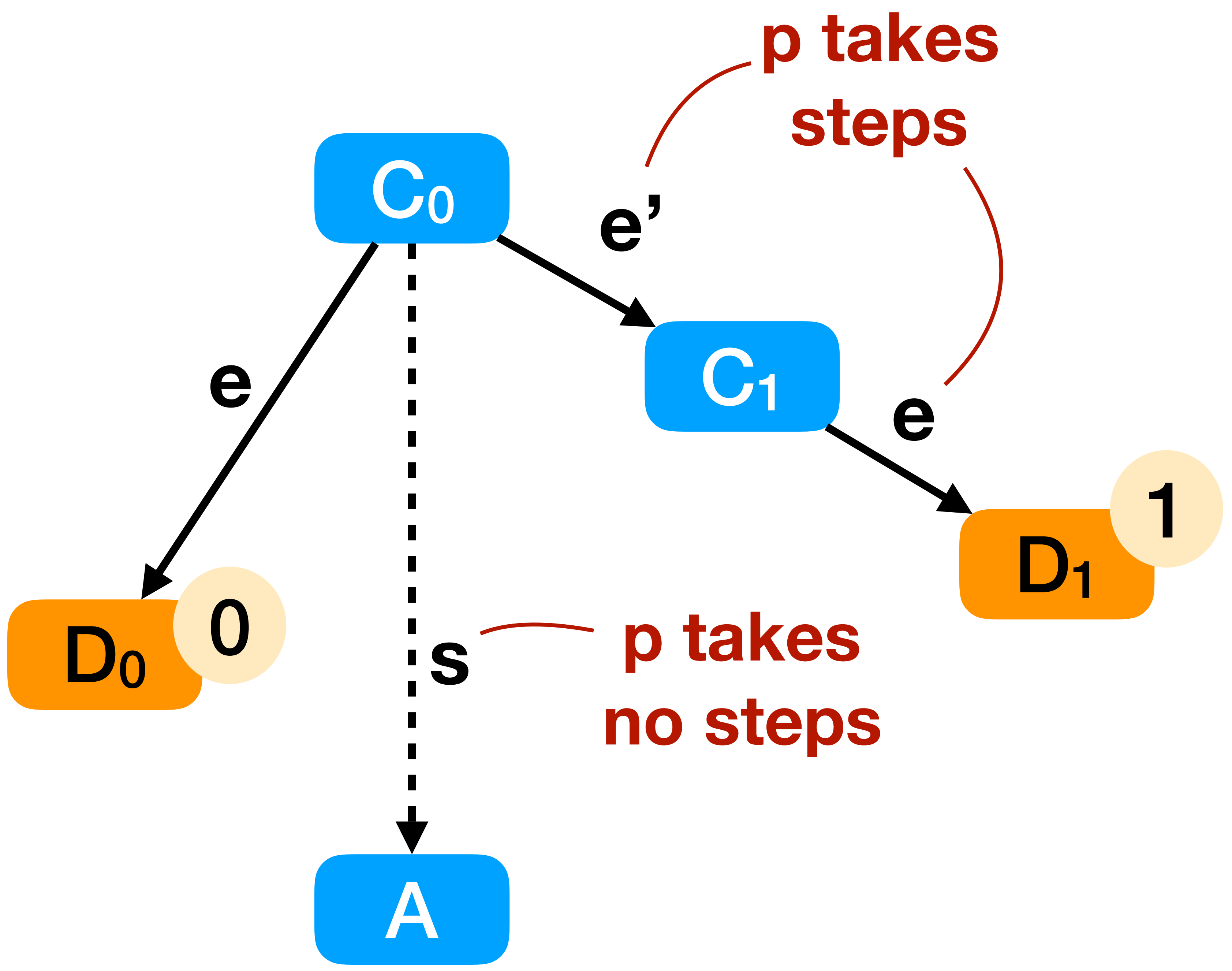
- Two events: $e = \langle p,m \rangle$ and $e' = \langle p',m' \rangle$
- Two cases: either process $p \neq p'$ or $p = p'$
- <u>Case $p \neq p'$</u>
  - $D_1 = e'(D_0)$, by commutativity of disjoint schedules
  - Impossible: a successor of 0-valent should be 0-valent

∃ reachable bivalent configuration [6/6]



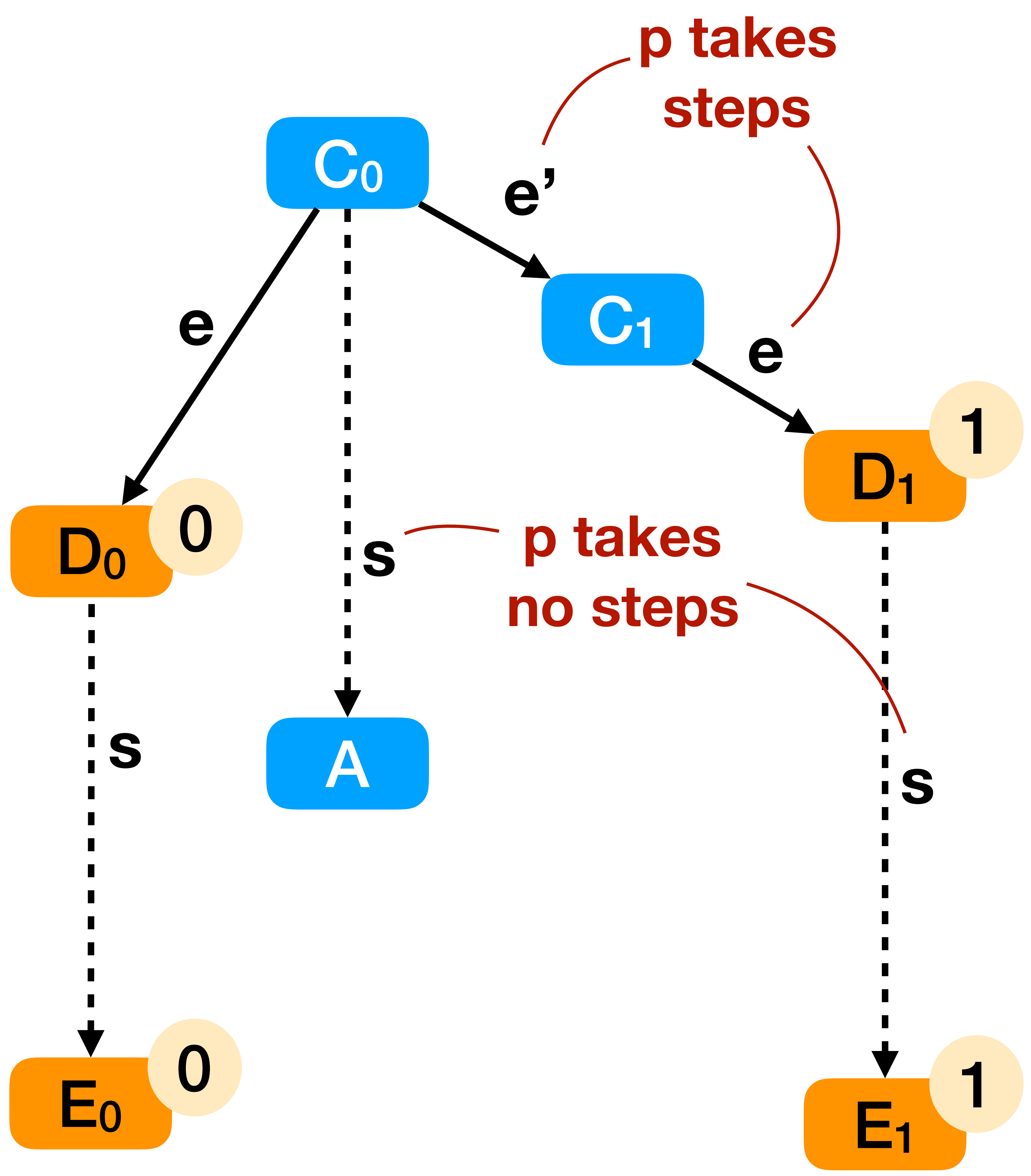
- Case p = p'

∃ reachable bivalent configuration [6/6]



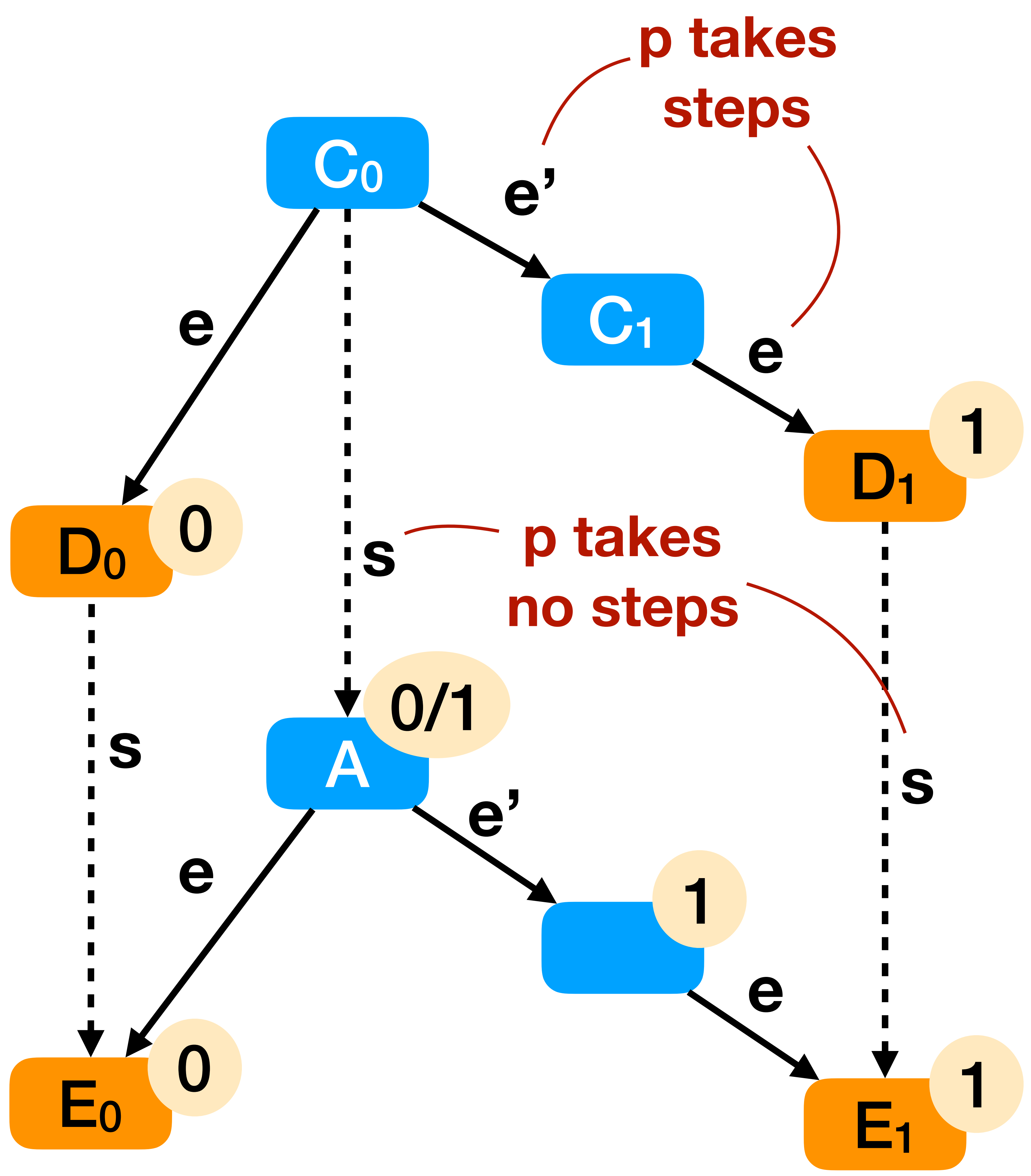
- <u>Case p = p'</u>
- Let *s* = *deciding* schedule from C_0 in which p takes no steps, and a configuration A = s(C_0)

∃ reachable bivalent configuration [6/6]



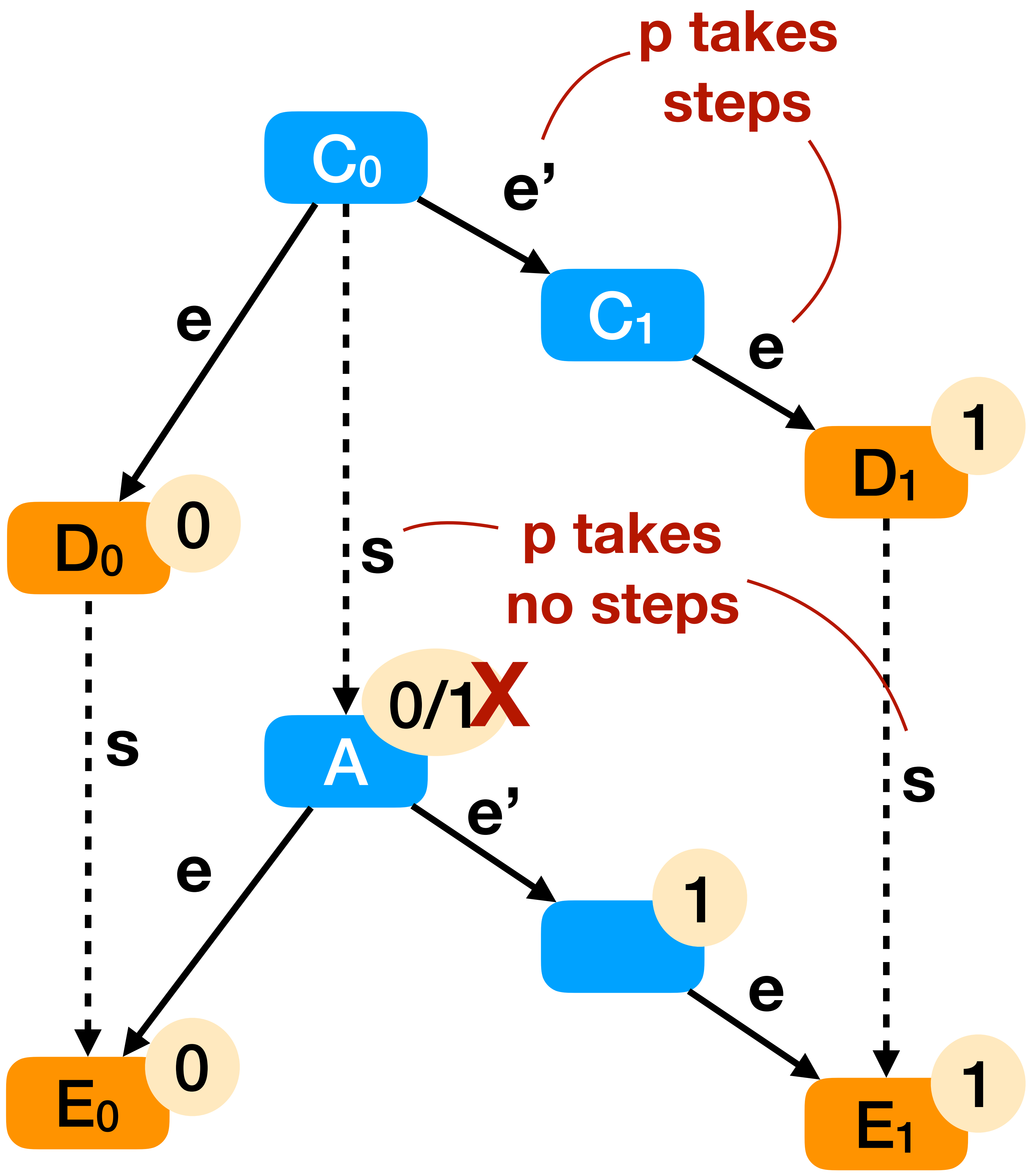
- <u>Case p = p'</u>
- Let *s* = *deciding* schedule from C_0 in which p takes no steps, and a configuration A = s(C_0)
- *s* is applicable to D_0 and D_1

∃ reachable bivalent configuration [6/6]



- <u>Case p = p'</u>
- Let *s* = *deciding* schedule from C_0 in which p takes no steps, and a configuration A = s(C_0)
- *s* is applicable to D_0 and D_1
- On left side, A is 0-valent, on the right is 1-valent, by commutativity of disjoint schedules

∃ reachable bivalent configuration [6/6]



- <u>Case p = p'</u>
- Let s = *deciding* schedule from C_0 in which p takes no steps, and a configuration A = s(C_0)
- s is applicable to D_0 and D_1
- On left side, A is 0-valent, on the right is 1-valent, by commutativity of disjoint schedules
- Then A is bivalent: Impossible!
  The run to A is deciding, by assumption
- Contradiction! Then $\mathscr{D}$ contains a bivalent configuration
  QED

Three consensus protocols and how they overcome FLP

1. Bracha and Toueg's probabilistic algorithm with a fair scheduler [1]

- **Convergence**: for any initial config, $\lim_{t \to \infty}$ Pr[a correct process has not decided within t steps] = 0
- A **fair scheduler** agent determines the next step of the system execution
  - Protocols can be viewed as consisting of rounds
  - R(q, p, t) = event that p receives a message from q in round t
- A scheduler is **fair** if:
  1. $\forall$ processes p, q, round t, $\exists\ \varepsilon > 0$, Pr[R(q, p, t)] $> \varepsilon$, and
  2. $\forall$ distinct processes p, q, r, round t, the events R(q,r,t) and R(q,p,t) are independent
  - Thus, there is a constant probability that all processes receive (n - k) messages from the same set of correct processes, $\forall$ round k
- **Using a fair scheduler, the algorithm terminates with probability 1**

2. Practical Byzantine Fault Tolerance (PBFT) [3]

- Efficient and practical BFT protocol, defined in an asynchronous setting
- **Always safe; liveness guaranteed only during periods of synchrony**
  - liveness: clients eventually receive replies to their requests
- *delay(t)* = time it takes a message to be received, after sent for the first time (time t)
- **Weak synchrony: assumes that *delay(t)* has an asymptotic upper bound**
  - *delay(t)* does not grow faster than *t* indefinitely
  - message delays are eventually bounded

3. Ethereum (proof-of-work)

- Participants (miners) compete to find the decision value (the next block in the chain)
- **Safety (Agreement) is not immediately guaranteed**
    - If two miners produce valid blocks at (almost) the same time, other miners will join first block they see
    - Forks: detached or orphaned blocks are valid but not part of the main chain
    - Agreement comes with a delay (after “enough” block confirmations)
- **Progress: economic reward incentivizes miners to produce blocks and to quickly join the longest chain**
    - The probability of a (slow) miner catching up decreases exponentially as blocks are added
    - Hypothesis: the fewer the miners, the higher the probability of forking forever (as in FLP)
    - Two miners may produce valid blocks at exactly the same time, other miners may split 50/50 to join them

Take away

- **FLP: not always possible to reach consensus in the asynchronous model**
  - In theory, there is always a path the system can take to avoid reaching consensus
  - In practice, it rarely happens, real-world systems have some degree of randomness that don’t match the asynchronous model
  - More realistic models avoid impossibility using failure detectors, fairness conditions, partial/weak synchrony, …

References

- [1] Gabriel Bracha and Sam Toueg. *Asynchronous Consensus and Broadcast Protocols.* In: J. ACM 32 (1985), pp. 824–840

- [2] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. *Impossibility of Distributed Consensus with One Faulty Process.* In: J. ACM 32.2 (Apr. 1985), pp. 374–382.

- [3] Miguel Castro and Barbara Liskov. *Practical Byzantine Fault Tolerance.* In: Proceedings of the Third Symposium on Operating Systems Design and Implementation. OSDI '99. New Orleans, Louisiana, USA: USENIX Association, 1999, pp. 173–186.

- [4] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. *Consensus in the Presence of Partial Synchrony.* In: J. ACM 35.2 (Apr. 1988), pp. 288–323.

- [5] M. Pease, R. Shostak, and L. Lamport. 1980. *Reaching Agreement in the Presence of Faults.* J. ACM 27, 2 (April 1980), 228–234.