Consensus

Fischer, Lynch and Paterson '85
Lamport '01

Presented by K. Vikram
Cornell University

To me, consensus seems to be the process of abandoning all beliefs, principles, values and policies. So it is something in which no one believes and to which no one objects.

Thatcher, Margaret

1925 British Stateswoman Prime Minister (1979-90)

Qui tacet consentire videtur

(He that is silent is thought to consent)

Let's Agree

- Core of many distributed algorithms
  - transaction commit, group membership
- Unreliability poses challenge
  - process crash, network partitioning, garbled messages, Byzantine
- Asynchrony
  - unpredictable delays

More Motivation

- With decentralization and replication, we need to coordinate nodes
  - data consistency
  - update propagation
  - mutual exclusion
  - consistent global states
  - group membership, communication
  - event ordering

Non-blocking Commit

- To install or not to install
- Each process votes a YES or a NO
- COMMIT if all votes are YES
- ABORT otherwise

Non-blocking Commit

- Properties:
- Termination
  - Every good process eventually decides
- Agreement
  - No two processes decide differently
- Obligation
  - decision value is either COMMIT or ABORT
  - COMMIT $\Rightarrow$ all processes voted YES
  - all processes vote YES and none are bad $\Rightarrow$ COMMIT

Surprisingly

A completely ***asynchronous*** protocol for consensus

cannot tolerate even a single

unannounced ***process death***

- forget Byzantine failures
- reliable messages

The Consensus Problem

- Each process has Initial Value in {0,1}
- Each non-faulty process decides on a value
- All non-faulty processes choose the same value
- The decision is eventually made
- Non-triviality

System Model

- Processes modeled as automata*
- Communicate through messages
- Receive, process and send
- Atomic Broadcast available

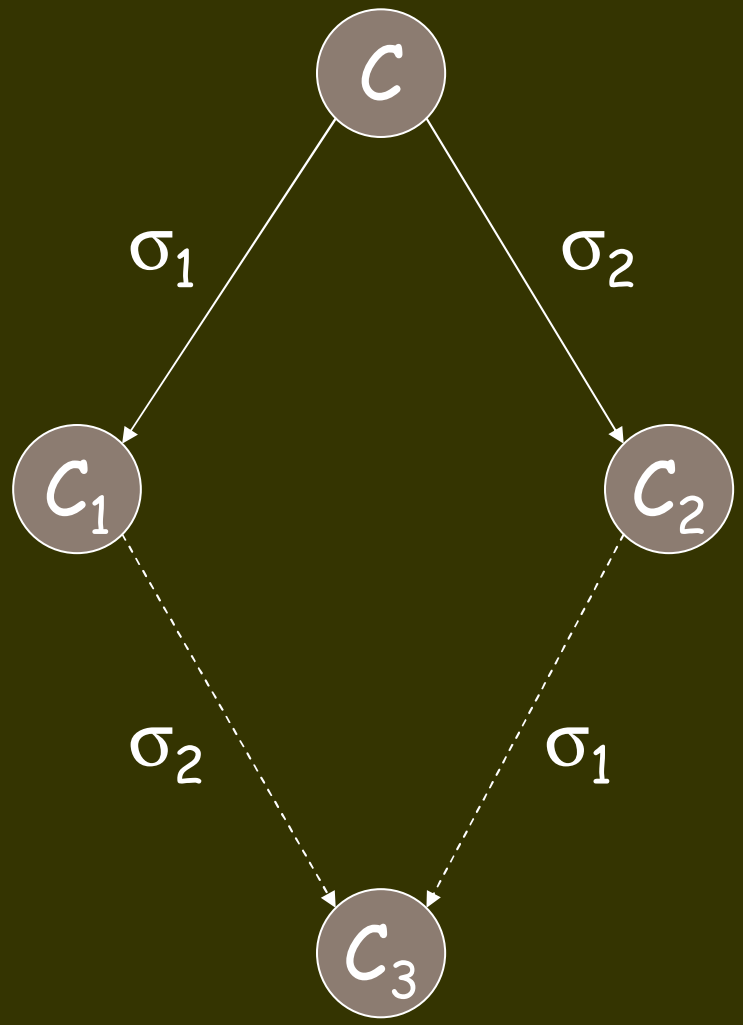
Formalize Consensus

- Asynchronous system of N processes
- Process p has x_p and $y_p \in \{b,0,1\}$
- Internal State
  - x_p, y_p, internal storage, pc
- *Initial State*
- p has a *transition function*
- Message: (p,m)
- Message System

The Message System

- send(p,m)
- receive(p)
- Non-deterministic receive
- *All messages are eventually received*
- Configuration, Initial Configuration
- Step, Event, Schedule, Run
- Commutativity

A Quick Lemma



If σ_1 and σ_2 are disjoint

Partially Correct Protocol

- *Decision Value*
- $\Rightarrow$ Accessible configurations have one decision value
- $\Rightarrow \forall\, v \in \{0,1\}\ \exists\, C$: C has decision value v
- *Nonfaulty Process*
- *Admissible Run*
- *Deciding Run*

Totally Correct Protocol

- In spite of one fault
- Partially Correct
- Every admissible run is a deciding run
- Some admissible run is not deciding

The Intuition

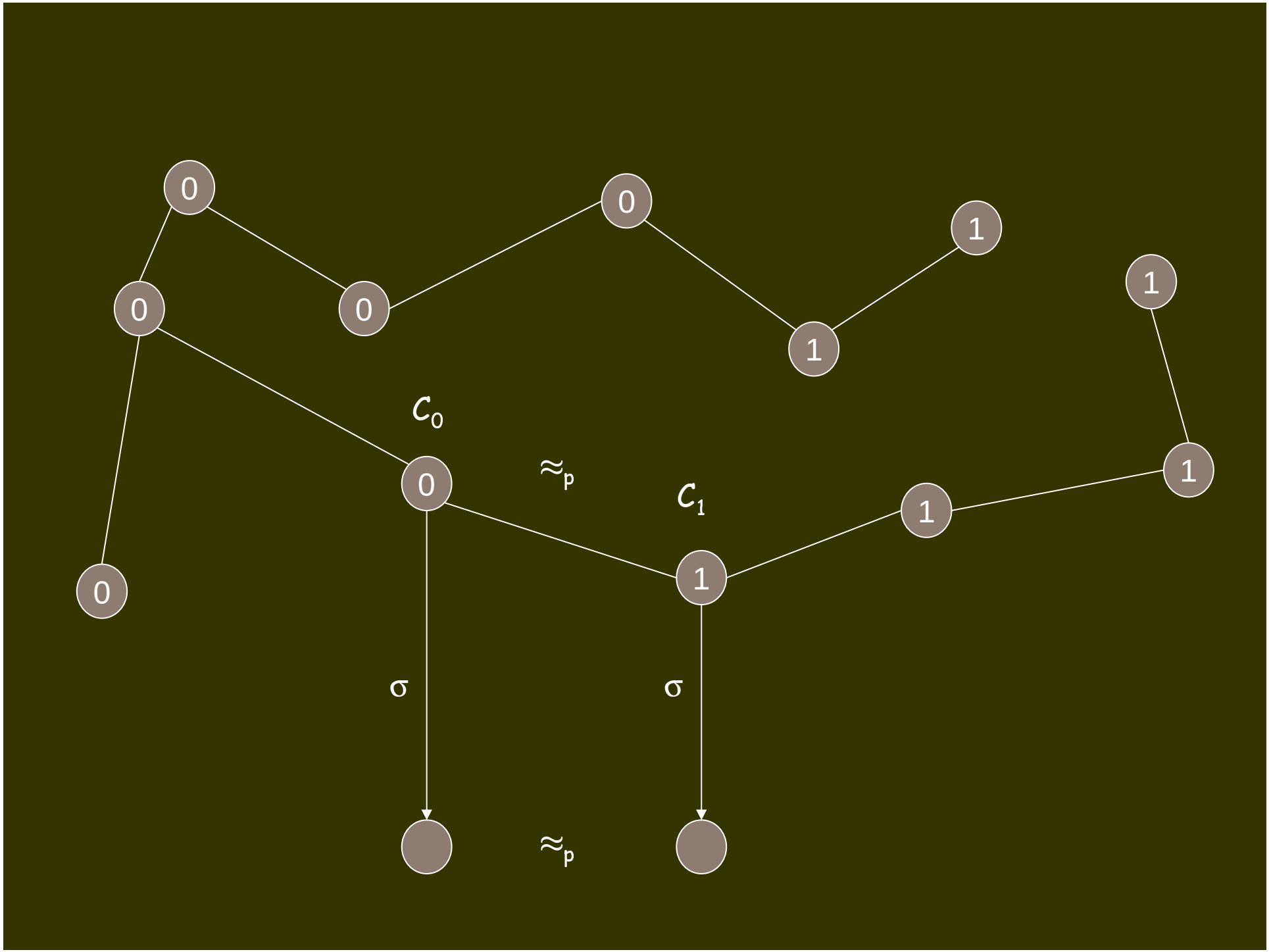
- Show possibility of protocol being indecisive forever
- Show
  - P could be indecisive initially
  - P can always avoid entering a decision state

Some more definitions

- V is the set of decision values of reachable configurations
- Valency of a configuration
  - Univalent if $|V| = 1$
  - Bivalent if $|V| = 2$
  - $V \neq \phi$

Initial Bivalent Configuration

- Lemma: P has a bivalent initial configuration
- P has 0-valent + 1-valent initial configurations
- Consider *adjacent* initial configs
- Consider an admissible deciding run that doesn't involve the differing process p
- A bivalent initial config is inevitable

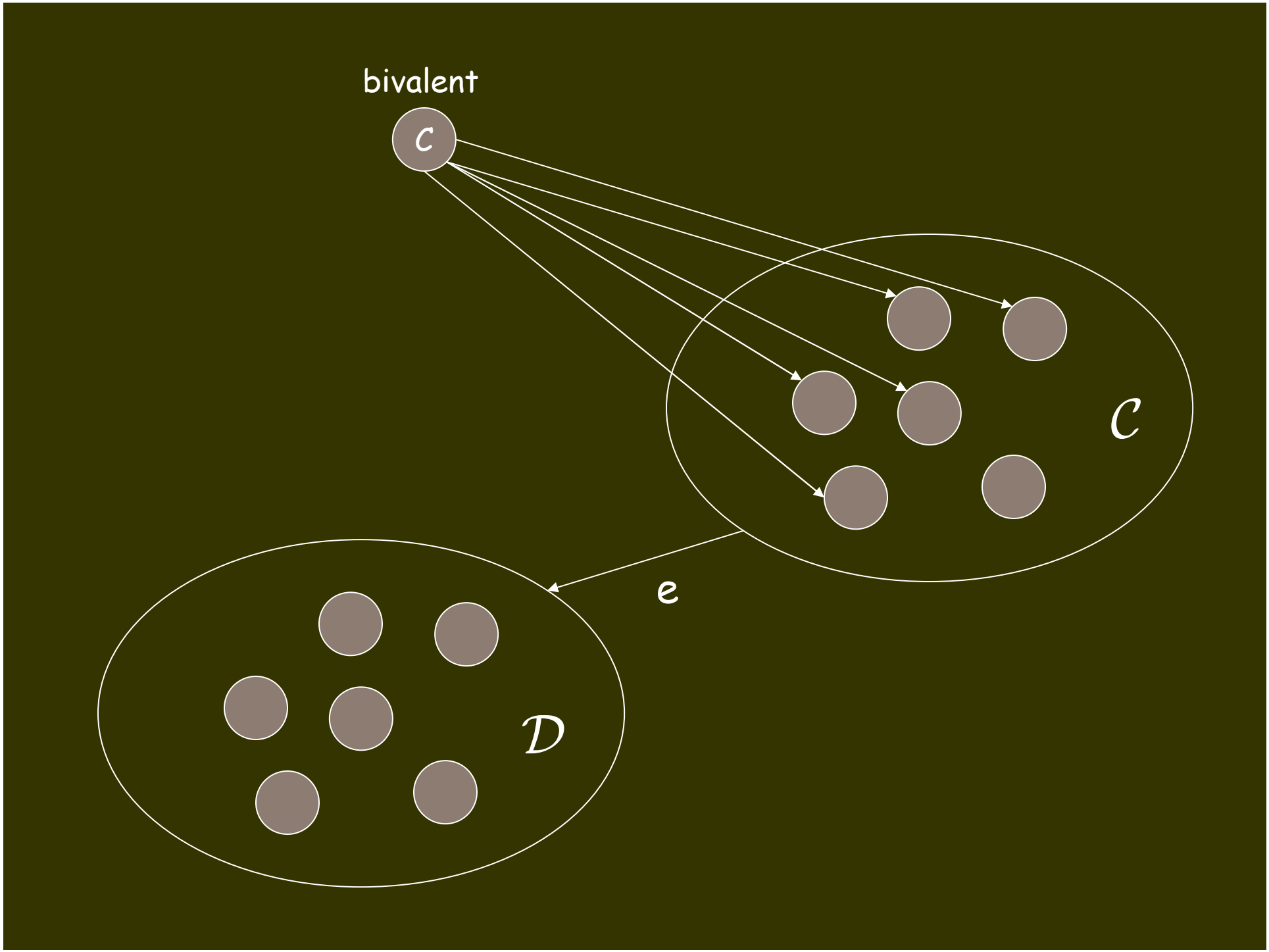

0
0
0
0
1
1
1
0
c_0
$\approx_p$
c_1
0
1
1
1
σ
σ
$\approx_p$

Stay Bivalent

- C is a bivalent configuration of P
- Let e = (p,m) be applicable to C
- $\mathcal{C}$: set of configs reachable from C without applying e
- $\mathcal{D} = e(\mathcal{C}) =$
  $\{e(E) \mid E \in \mathcal{C}$ and e is applicable to $E\}$
- $\mathcal{D}$ contains a bivalent configuration

Proof (reductio ad absurdum)

- e is applicable to every $E \in \mathcal{C}$
- Assume for contradiction
  - $\forall D \in \mathcal{D}$, D is univalent
- Consider E_0, E_1 reachable from C (existence?)
- If $E_i \in \mathcal{C}$, let $F_i = e(E_i) \in \mathcal{D}$
- else E_i is reachable from some $F_i \in \mathcal{D}$


bivalent
c
C
e
D

Proof (continued)

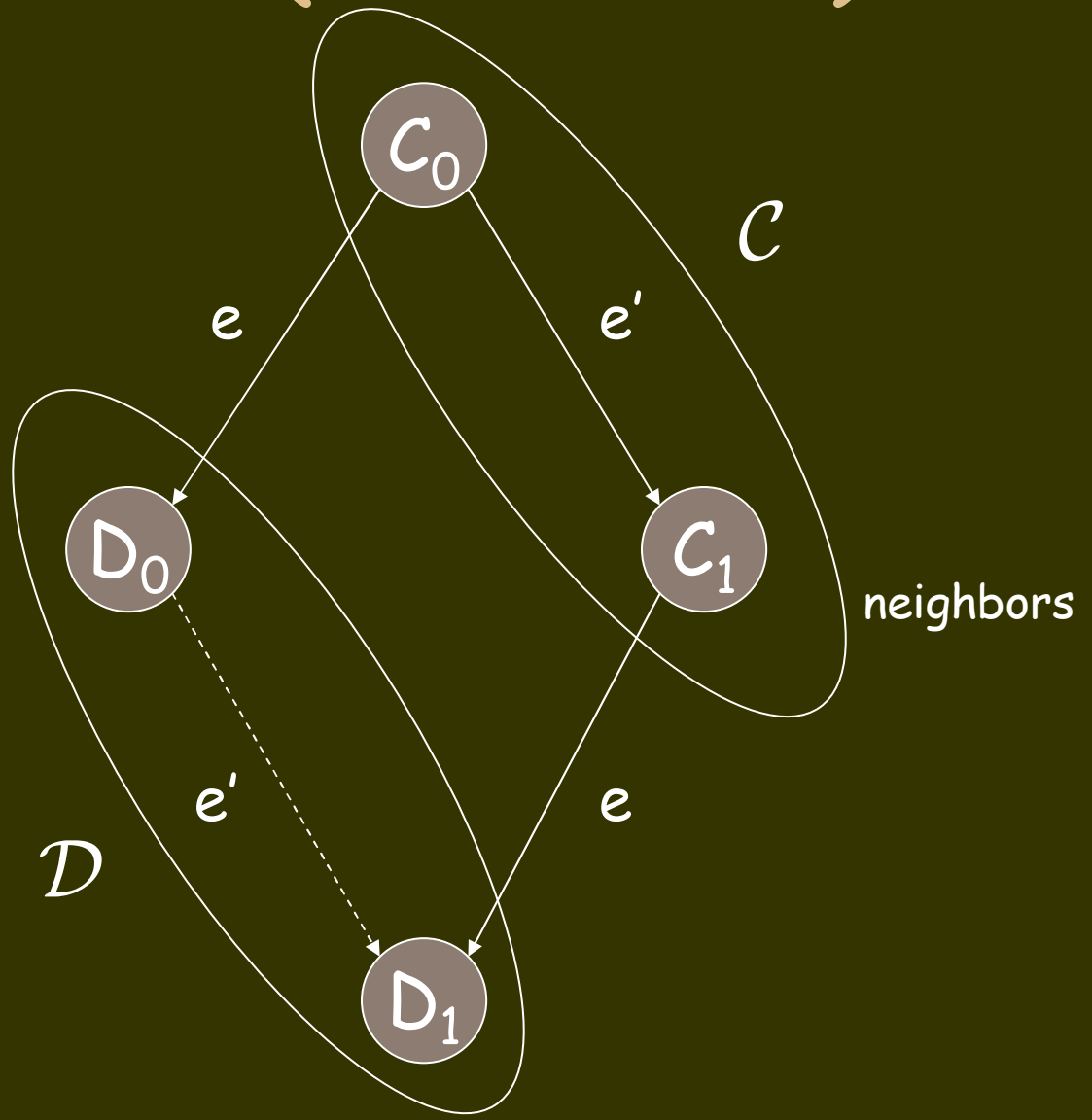
- Either $E_i \to F_i$ or $F_i \to E_i$
- F_i is i-valent
- $\mathcal{D}$ contains both 0-valent and 1-valent configurations

Proof (continued)

- Neighbor configurations
  - reachable in a single step
- C_0 and C_1 are neighbors

Proof (continued)

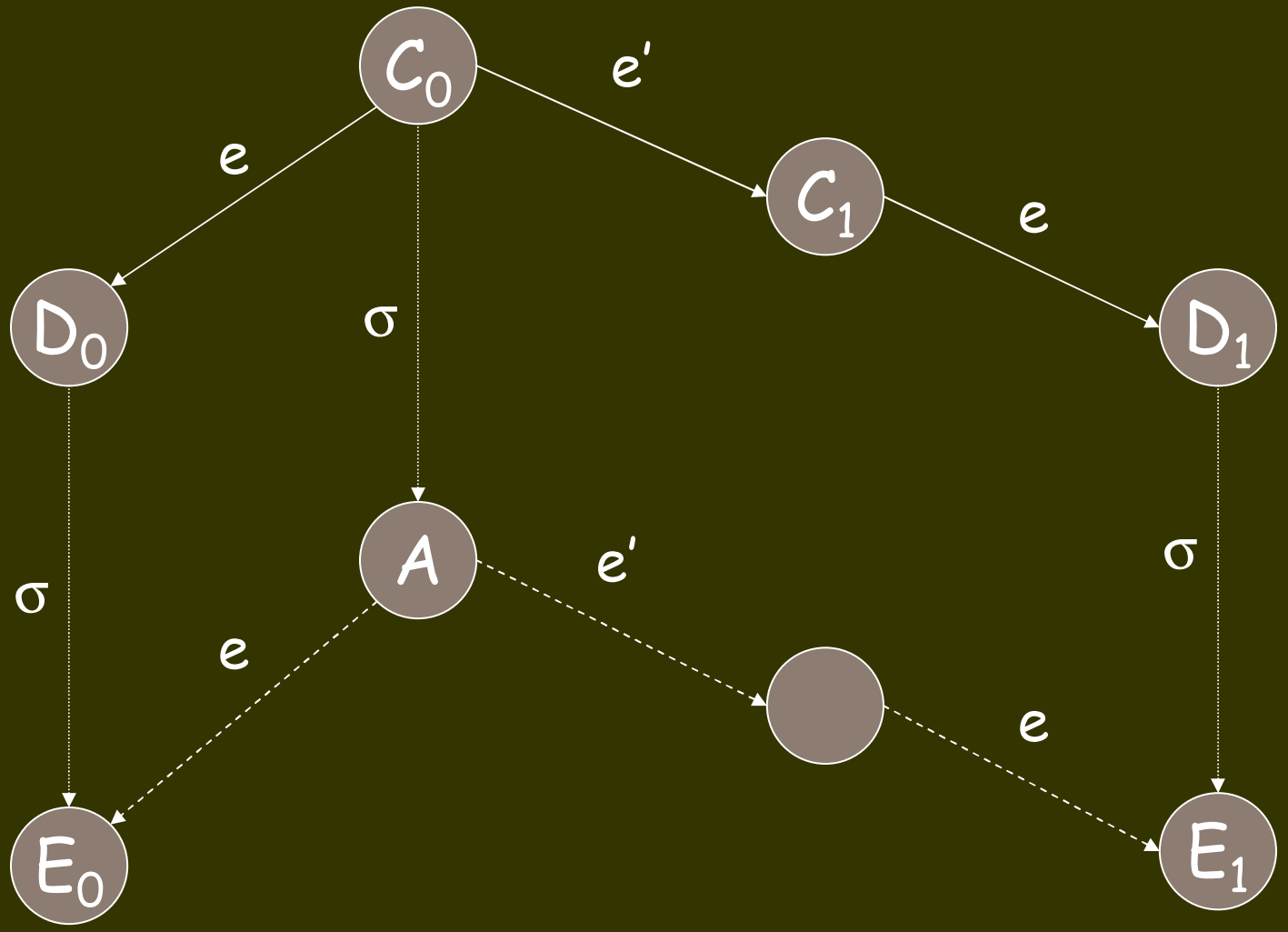
e = (p,m)
e′ = (p′,m′)



Proof (continued)

- If $p' = p$
- Consider any finite **deciding** run from C_0 where p takes no steps
- Suppose σ is the schedule, $A = \sigma(C_0)$
- σ applicable to D_i (Quick Lemma)
- $A \rightarrow E_0$, $A \rightarrow E_1$
- $\Rightarrow$ A is bivalent $\Rightarrow\Leftarrow$

Proof (continued)



How to avoid deciding…

- Given these lemmas, we show how to construct an ***admissible nondeciding run***
- Ensure *admissibility* via
  - a queue of processes
  - messages in buffer ordered by sent time
- The first process receives its earliest message

How to avoid deciding…

- Ensure nondecision as follows:
- Begin execution at C_0, a bivalent configuration
- At any bivalent configuration C
  - if p heads the process queue
  - m is p's earliest message in the buffer
  - e = (p,m)
- $\exists$ bivalent C' reachable from C, where e is the last event applied

All hope is not lost…

- Consensus possible
  - if no process dies during protocol execution
  - majority are nonfaulty
- Each process broadcasts its number
- Listens for messages from $\lceil (N+1)/2 \rceil - 1$
- Create G
- Create G^+ (transitive closure of G)
- Compute initial clique of G^+

Conclusion

- Cannot distinguish among
  - Crashed Process
  - Very Slow Process
  - Slow Communication
- Problem of modeling
- Relax restrictions
  - Asynchrony
  - Probabilistic guarantees

Paxos – a case study

- Safety requirements for consensus
  - Only a proposed value can be chosen
  - Only a single value is chosen
  - A process learns that the value is chosen only after it has been chosen
- Eventually (Liveness)
  - A proposed value is chosen
  - If chosen, a process learns its value

The Setup

- Three Roles
  - Proposer
  - Acceptor
  - Learner
- Asynchronous, non-Byzantine model
  - Agents may delay, fail or restart
  - Messages can be delayed, duplicated or lost but not corrupted

Choosing a Value

- An acceptor must choose the first proposal it receives
- A value is chosen when majority acceptors accept it

Choosing a Value

- Acceptors accept multiple proposals, distinguishing them by a proposal number
- A value is chosen when a single proposal with that value is accepted by majority

Choosing a Value

- Allow multiple proposals to be chosen, if they have the same value
- If a proposal with value v is chosen, every higher-numbered proposal chosen has value v
- … accepted by any acceptor …
- … issued by any proposer …

Choosing a Value

- Proposers maintain the invariant:
- For any v and n, if a proposal with v and n is issued, ∃ set S of majority acceptors such that either
  - no acceptor in S accepted any proposal < n
  - v is the value of the highest-numbered proposal (< n) accepted by S

Proposer's Job

- Choose a proposal number n and send a *prepare* request to a set of majority acceptors asking for:
  - accepted proposal with highest number < n
  - promise not to accept proposals < n
- Depending on the response, it sends an accept request with a self-chosen value, or the value of the highest proposal
- Can abort anytime, without reusing n

Acceptor's Job

- Receive prepare or accept requests
- A response is not required for safety
- A response is allowed
  - always for a prepare request
  - for an accept request, without violating an earlier promise

Progress in Paxos

- Two proposers can keep making prepare requests and getting each other's accept requests ignored
- Solution: use a distinguished proposer
- Liveness still not guaranteed

Conclusions

- Simple mechanism
- Processes allowed to restart
- No liveness guarantee

Finally

- Comments/Criticisms
- Questions?
- Thank You!