CLOUD COMPUTING CONCEPTS

with Indranil Gupta (Indy)

PAXOS

Lecture D

THE FLP PROOF

Consensus in an Asynchronous System

- Impossible to achieve!

- Proved in a now-famous result by Fischer, Lynch, and Patterson, 1983 (FLP)
  - Stopped many distributed system designers dead in their tracks
  - A lot of claims of "reliability" vanished overnight

Recall

Asynchronous system: All message delays and processing delays can be arbitrarily long or short.

Consensus:

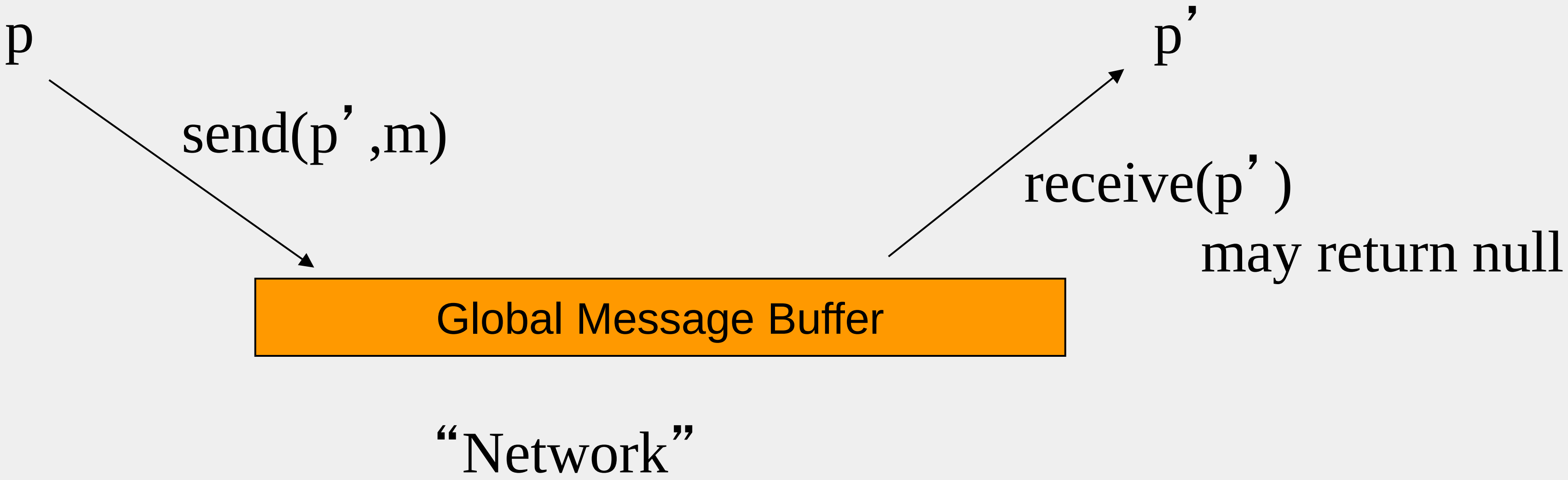
- Each process p has a state
  - Program counter, registers, stack, local variables
  - Input register xp : initially either 0 or 1
  - Output register yp : initially b (undecided)
- Consensus Problem: design a protocol so that either
  - All processes set their output variables to 0 (all-0's)
  - Or all processes set their output variables to 1 (all-1's)
  - Non-triviality: at least one initial system state leads to each of the above two outcomes

Proof Setup

- For impossibility proof, OK to consider

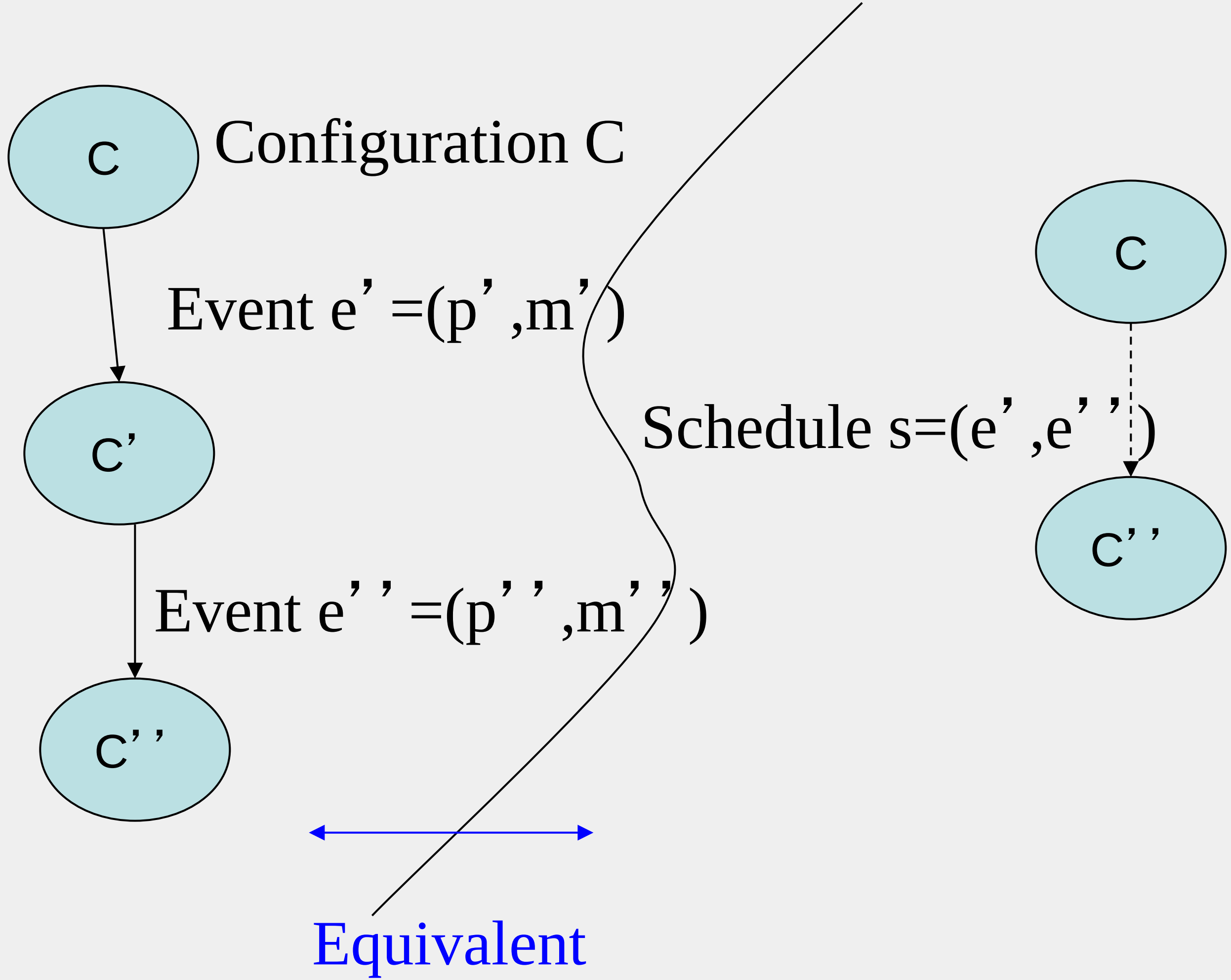
1. More restrictive system model, and
2. Easier problem
   - Why is this ok?

Network

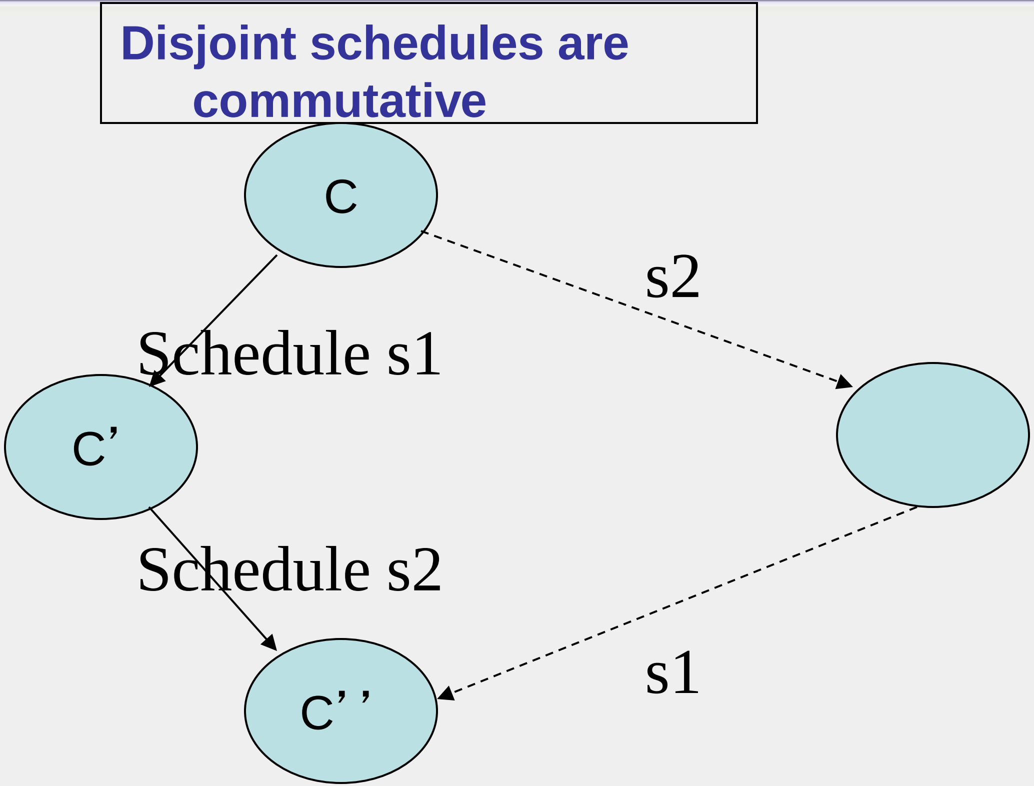

p
p’
send(p’,m)
receive(p’)
may return null
Global Message Buffer
“Network”

States

- State of a process
- **Configuration**=global state. Collection of states, one for each process; alongside state of the global buffer.
- Each event (different from Lamport events)
  - Receipt of a message by a process (say p)
  - Processing of message (may change recipient's state)
  - Sending out of all necessary messages by p
- Schedule: sequence of events


C
Configuration C
Event e’ =(p’ ,m’)
C’
Event e’ ’ =(p’ ’ ,m’ ’)
C’ ’
Equivalent
C
Schedule s=(e’ ,e’ ’)
C’ ’

Lemma 1



s1 and s2 involve <u>disjoint</u> sets of <u>receiving</u> processes, and are <u>each</u> applicable on C

Easier Consensus Problem

Easier Consensus Problem:

some process eventually sets yp to be 0 or 1

Only one process crashes – we're free to choose which one

Easier Consensus Problem

- Let config. C have a set of decision values V reachable from it
  - If |V| = 2, config. C is bivalent
  - If |V| = 1, config. C is 0-valent or 1-valent, as is the case

- Bivalent means outcome is unpredictable

What the FLP proof shows

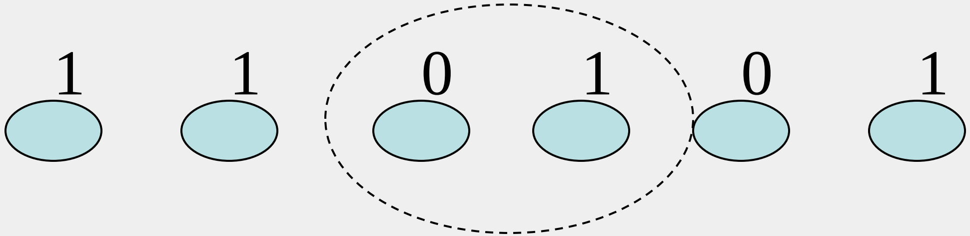
1. There exists an initial configuration that is bivalent

2. Starting from a bivalent config., there is always another bivalent config. that is reachable

LEMMA 2

Some initial configuration is bivalent

- Suppose all initial configurations were either 0-valent or 1-valent.
- If there are N processes, there are 2^N possible initial configurations
- Place all configurations side-by-side (in a lattice), where adjacent configurations differ in initial xp value for <u>exactly one</u> process.

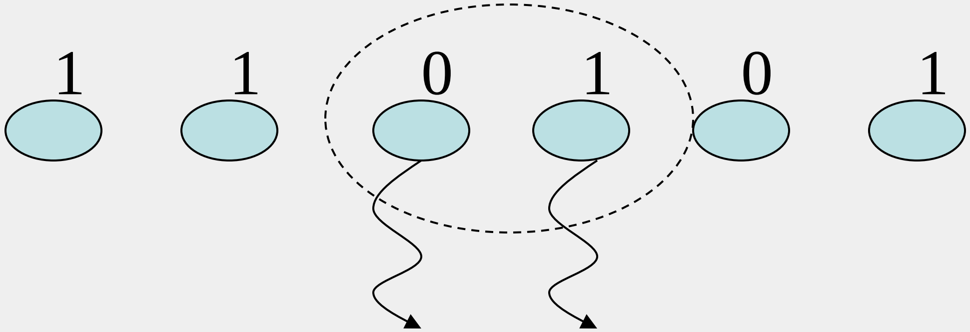


- There has to be some adjacent pair of 1-valent and 0-valent configs.

LEMMA 2

Some initial configuration is bivalent

- There has to be some adjacent pair of 1-valent and 0-valent configs.
- Let the process p, that has a different state across these two configs., be the process that has crashed (i.e., is silent throughout)



Both initial configs. will lead to the same config. for the same sequence of events

Therefore, both these initial configs. are <u>bivalent</u> when there is such a failure

What we'll show

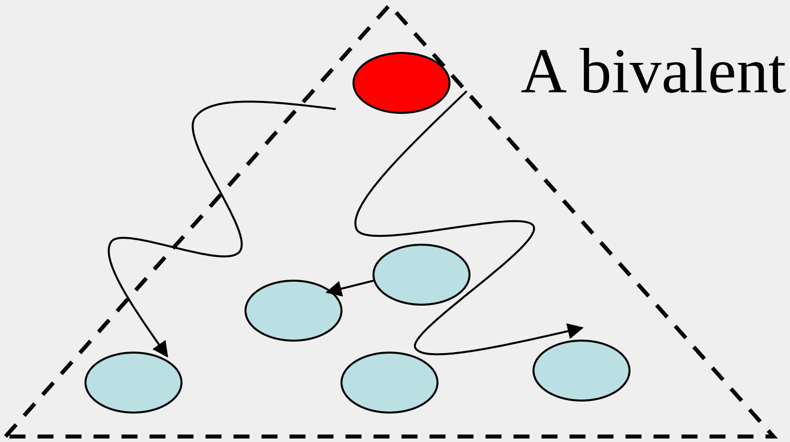
1. There exists an initial configuration that is bivalent

2. Starting from a bivalent config., there is always another bivalent config. that is reachable

Lemma 3

Starting from a bivalent config., there is always another bivalent config. that is reachable

Lemma 3

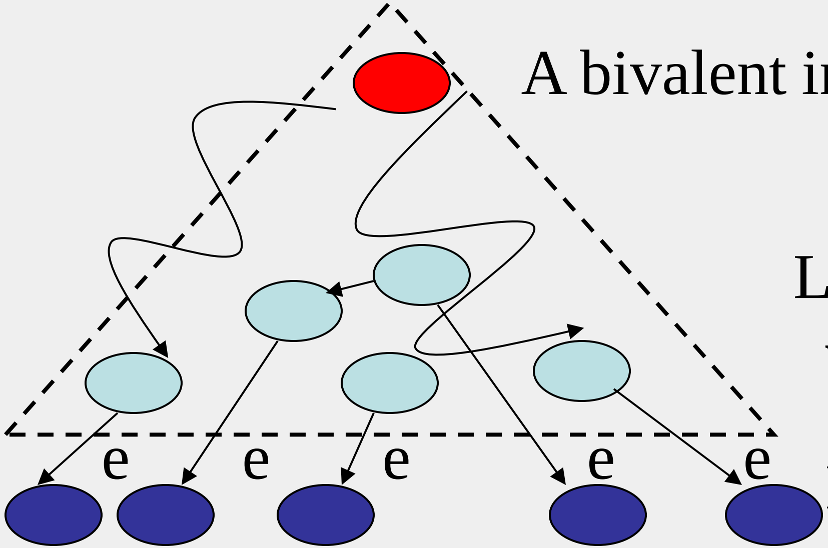


A bivalent initial config.

let e=(p,m) be some event applicable to the initial config.

Let *C* be the set of configs. reachable **without** applying e

Lemma 3



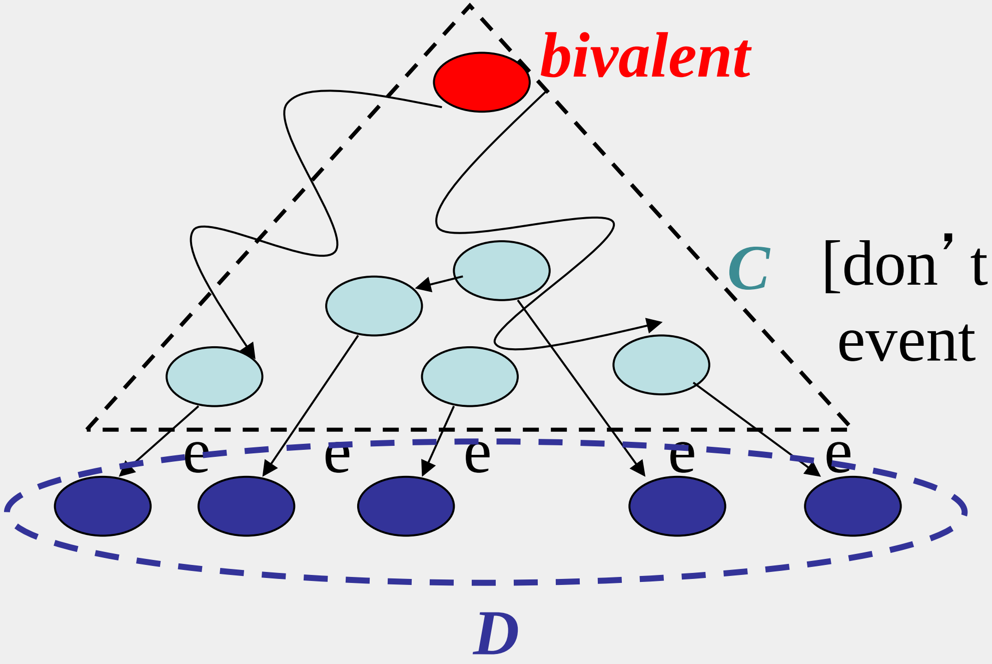
A bivalent initial config.

let e=(p,m) be some event applicable to the initial config.

Let ***C*** be the set of configs. reachable **without** applying e

Let ***D*** be the set of configs. obtained by **applying e** to some config. in ***C***

Lemma 3



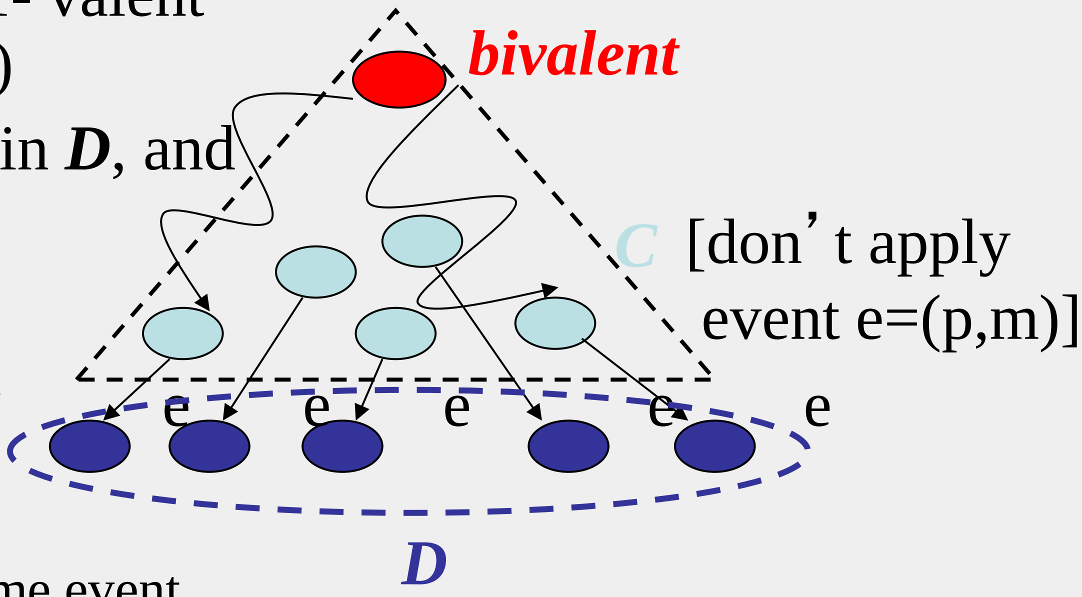
[don't apply
event e=(p,m)]

Claim. Set D contains a bivalent config.

Proof. By contradiction. That is,
suppose ***D*** has only 0- and 1- valent states (and no bivalent ones)

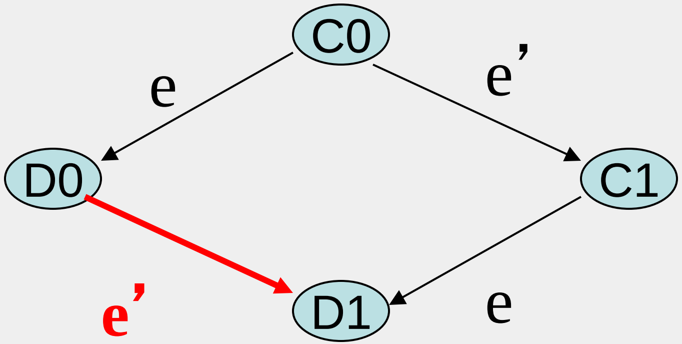
- There are states D0 and D1 in ***D***, and C0 and C1 in ***C*** such that
  - D0 is 0-valent, D1 is 1-valent
  - D0=C0 foll. by e=(p,m)
  - D1=C1 foll. by e=(p,m)
  - And C1 = C0 followed by some event e’=(p’,m’)

(why?)

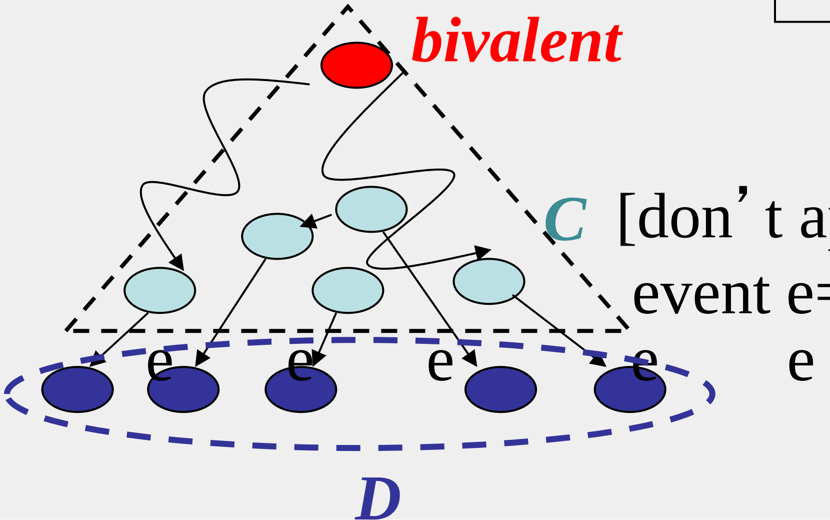


Proof. (contd.)

- Case I: p' is not p
- Case II: p' same as p

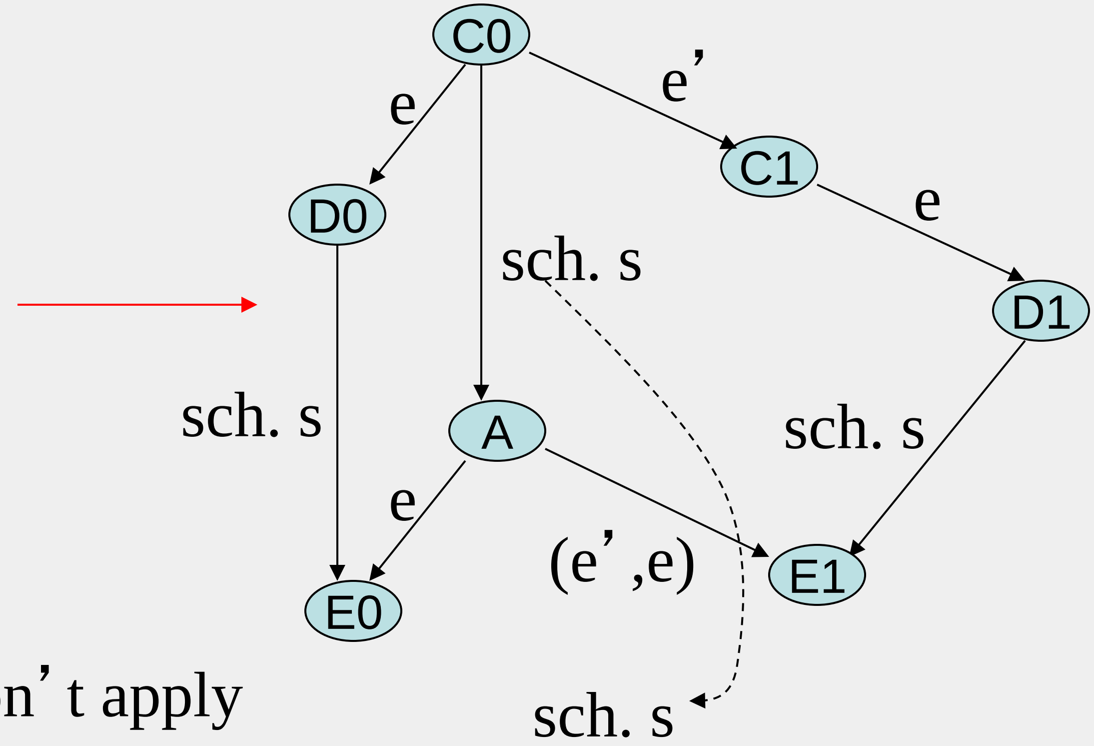


Why? (Lemma 1)
But D0 is then bivalent!

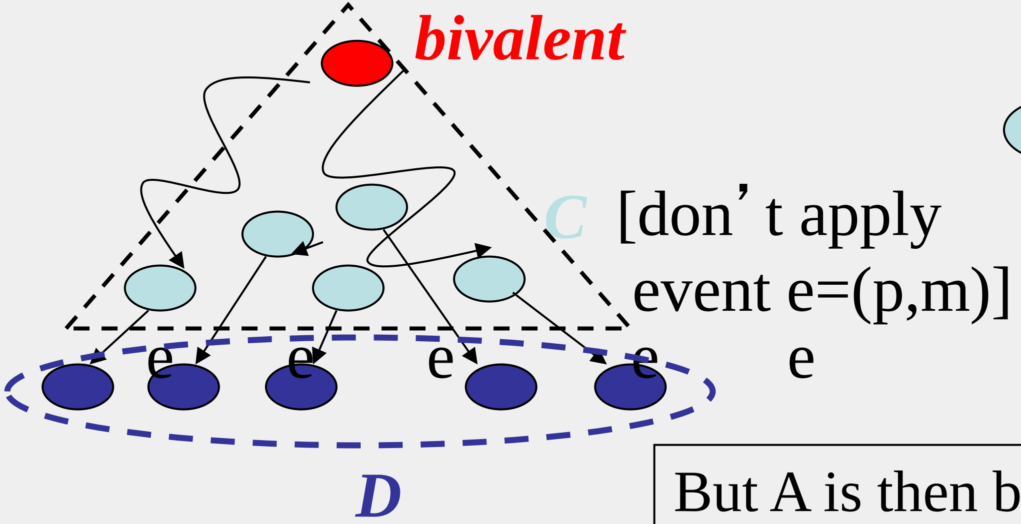


Proof. (contd.)

- Case I: p' is not p
- Case II: p' same as p



- finite
- **deciding run** from C0
- *p takes no steps*



But A is then bivalent!

Lemma 3

Starting from a bivalent config., there is always another bivalent config. that is reachable

Putting it all Together

- Lemma 2: There exists an initial configuration that is bivalent
- Lemma 3: Starting from a bivalent config., there is always another bivalent config. that is reachable

- Theorem (Impossibility of Consensus): There is always a run of events in an asynchronous distributed system such that the group of processes never reaches consensus (i.e., stays bivalent all the time)

Summary

- Consensus problem
  - Agreement in distributed systems
  - Solution exists in synchronous system model (e.g., supercomputer)
  - Impossible to solve in an asynchronous system (e.g., Internet, Web)
    - Key idea: with even one (adversarial) crash-stop process failure, there are always sequences of events for the system to decide any which way
    - Holds true regardless of whatever algorithm you choose!
  - FLP impossibility proof
- One of the most fundamental results in distributed systems