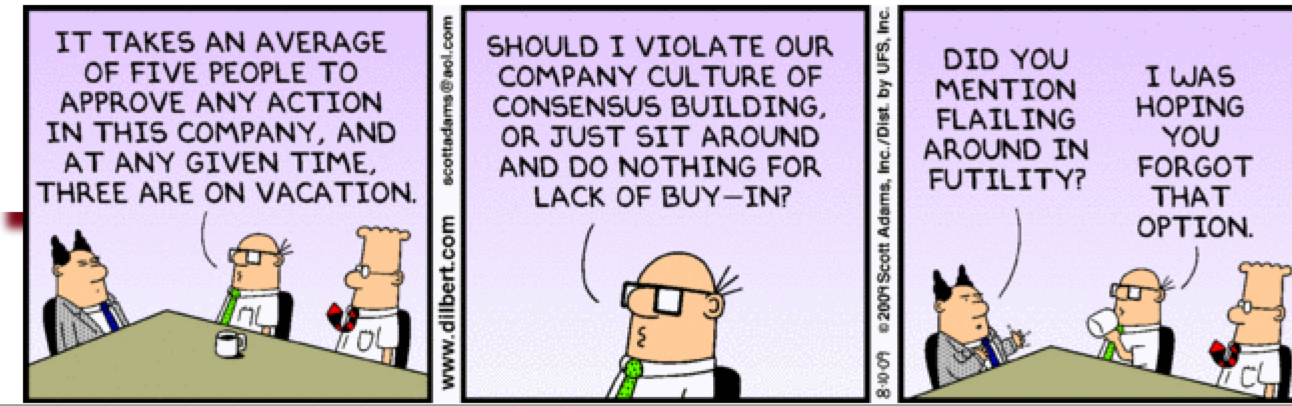
Computer Science 425
Distributed Systems

CS 425 / CSE 424 / ECE 428



Indranil Gupta (Indy)

October 9, 2012

Lecture 13

(Impossibility of) Consensus

Reading: Paper on Website (Sections 1-3)

Give it a thought

Have you ever wondered why vendors of (distributed) software solutions always only offer solutions that promise five-9′s reliability, seven-9′s reliability, but never 100% reliability?

Give it a thought

Have you ever wondered why software vendors always only offer solutions that promise five-9′s reliability, seven-9′s reliability, but never 100% reliability?

The fault does not lie with Microsoft or Amazon or Google

The fault lies in the *impossibility of consensus*

What is Consensus?

- **N processes**
- **Each process p has**
  - **input variable xp : initially either 0 or 1**
  - **output variable yp : initially b (b=undecided) – can be changed only once**
- **Consensus problem: design a protocol so that either**
  1. **all non-faulty processes set their output variables to 0**
  2. **Or non-faulty all processes set their output variables to 1**
  3. **There is at least one initial state that leads to each outcomes 1 and 2 above**
  4. **(There might be other conditions too, but we'll consider the above weaker version of the problem).**

Let's Solve Consensus!

- **Uh, what′s the** model**? (assumptions!)**
- **Processes fail only by** *crash-stop***ping**
- **Synchronous system: bounds on**
  - **Message delays**
  - **Max time for each process step**

  e.g., multiprocessor (common clock across processors)
- **Asynchronous system: no such bounds!**

  e.g., The Internet! The Web!

Consensus in Synchronous Systems

For a system with at most *f* processes crashing, the algorithm proceeds in *f+1* **rounds** (with timeout), using basic multicast (B-multicast).

- A round is a numbered period of time where processes know its start and end (kinda like an hour, only smaller)

- $Values^r_i$: the set of proposed values known to process P_i at the beginning of round *r*.
- Initially $Values^0_i = \{\}$; $Values^1_i = \{v_i = \text{xp}\}$

for round r = 1 to f+1 do

multicast ($Values^r_i$)

$Values^{r+1}_i \leftarrow Values^r_i$

for each V_j received

$Values^{r+1}_i = Values^{r+1}_i \cup \mathrm{V}_j$

end

end

$\text{yp} = d_i = \text{minimum}(Values^{f+1}_i)$

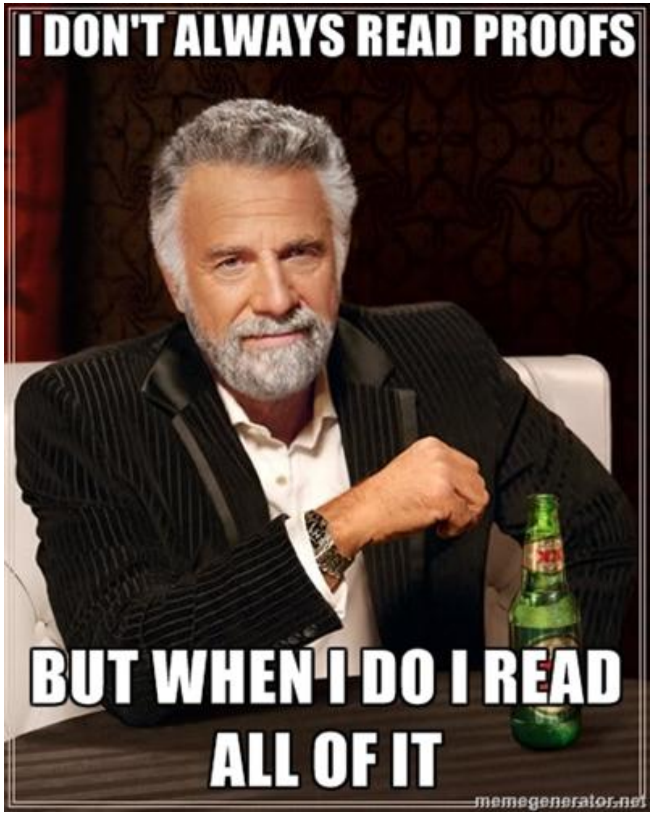
Why does the Algorithm Work?

- **Proof by contradiction.**
- **Assume that two non-faulty processes differ in their final set of values.**
- **Suppose p_i and p_j are these processes.**
- **Assume that p_i possesses a value v that p_j does not possess.**
  - → **In the last (f+1) round, some third process, p_k, sent v to p_i, but crashed before sending v to p_j.**
  - → **Any process sending v in the penultimate (f) round must have crashed; otherwise, both p_k and p_j should have received v.**
  - → **Proceeding in this way, we infer at least one crash in each of the preceding rounds.**
  - → **But we have assumed at most f crashes can occur; yet there are $f+1$ rounds ==> contradiction.**

Consensus in an Asynchronous System

- **Messages have arbitrary delay, processes arbitrarily slow**
- **Impossible to achieve!**
  - **even a <u>single failed</u> process is enough to avoid the system from reaching agreement!**
  - **Key observation: a slow process indistinguishable from a crashed process**
- **Impossibility Applies to *any* protocol that claims to solve consensus!**

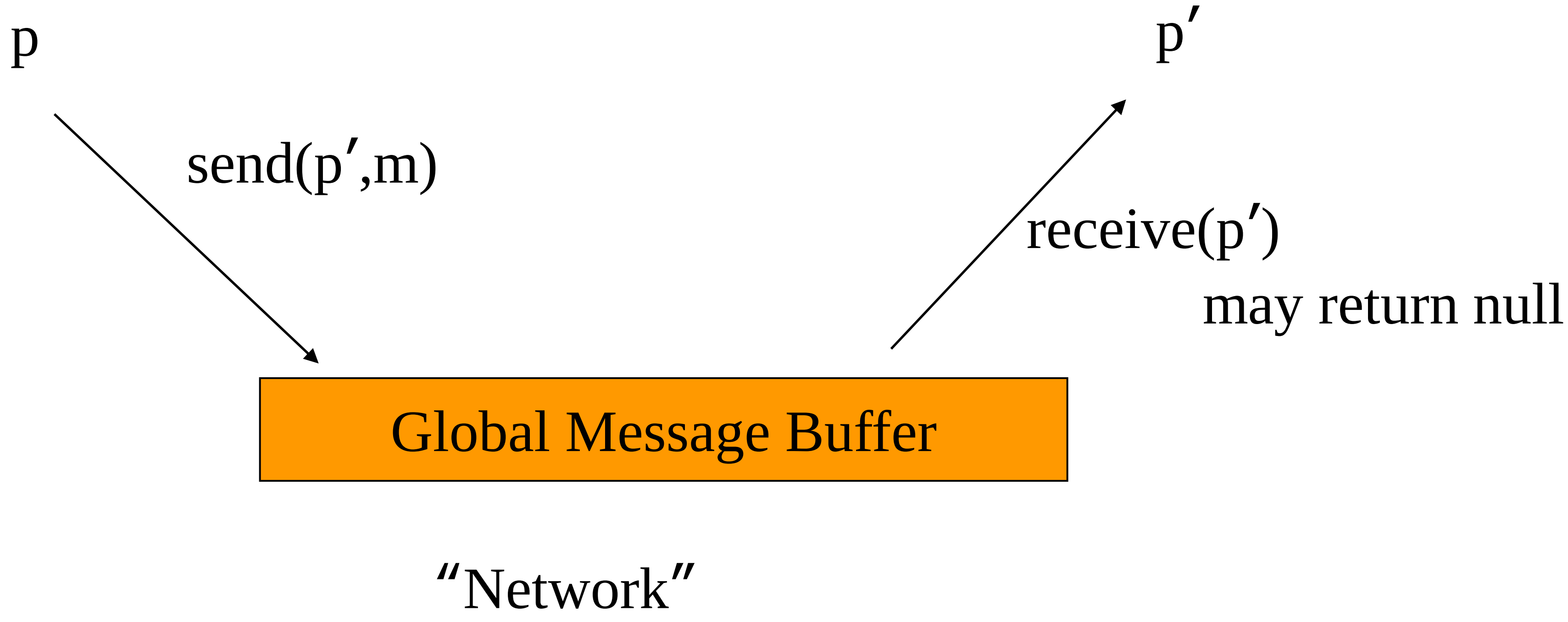
- **Proved in a now-famous result by Fischer, Lynch and Patterson, 1983 (FLP)**
  - **Stopped many distributed system designers dead in their tracks**
  - **A lot of claims of "perfect reliability" vanished overnight**



Let’s look at the proof of the impossibility!

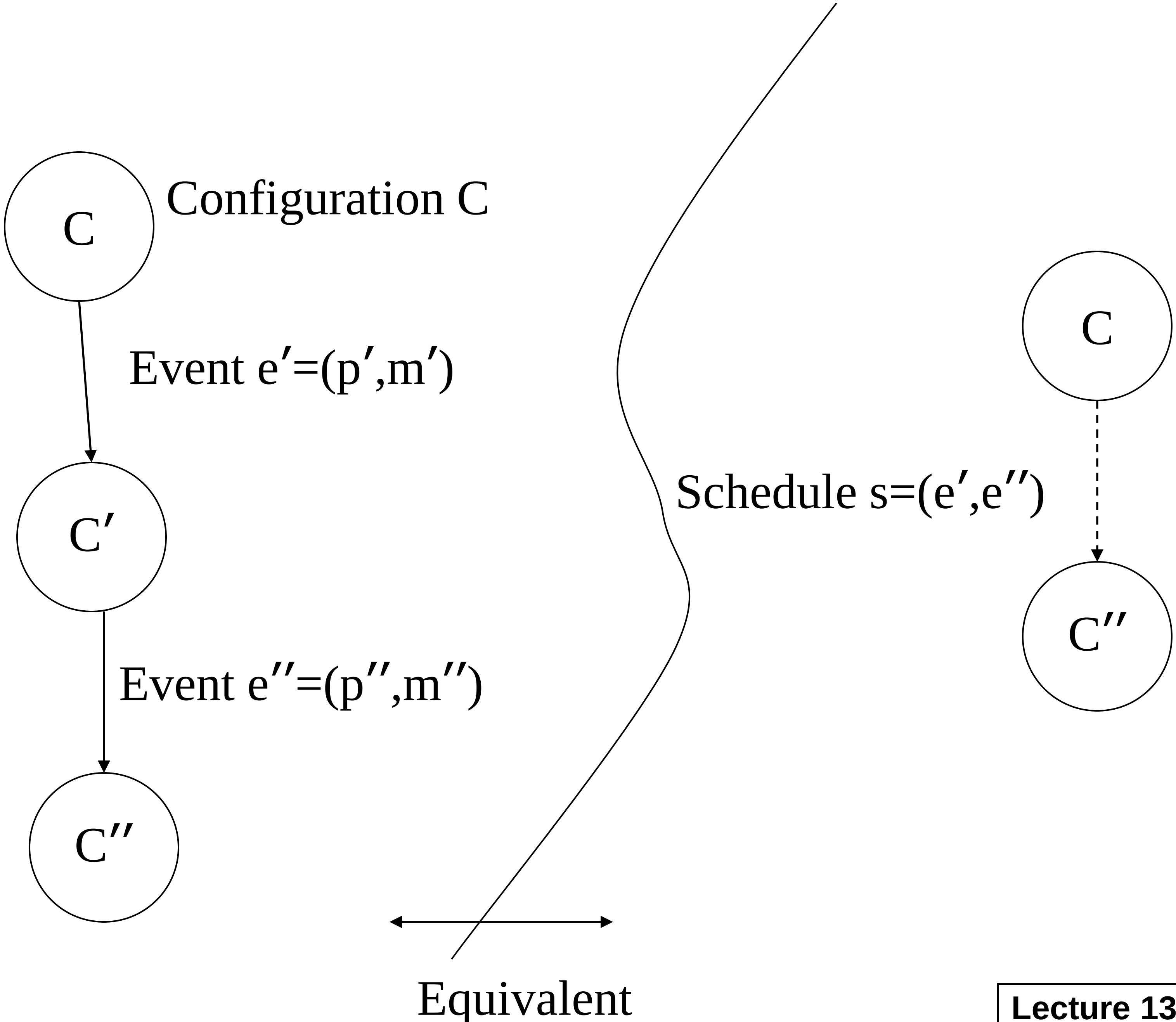
Recall

- 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 (b=undecided)
- Consensus Problem: design a protocol so that either
  1. all non-faulty processes set their output variables to 0
  2. Or non-faulty all processes set their output variables to 1
  3. (No trivial solutions allowed)


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

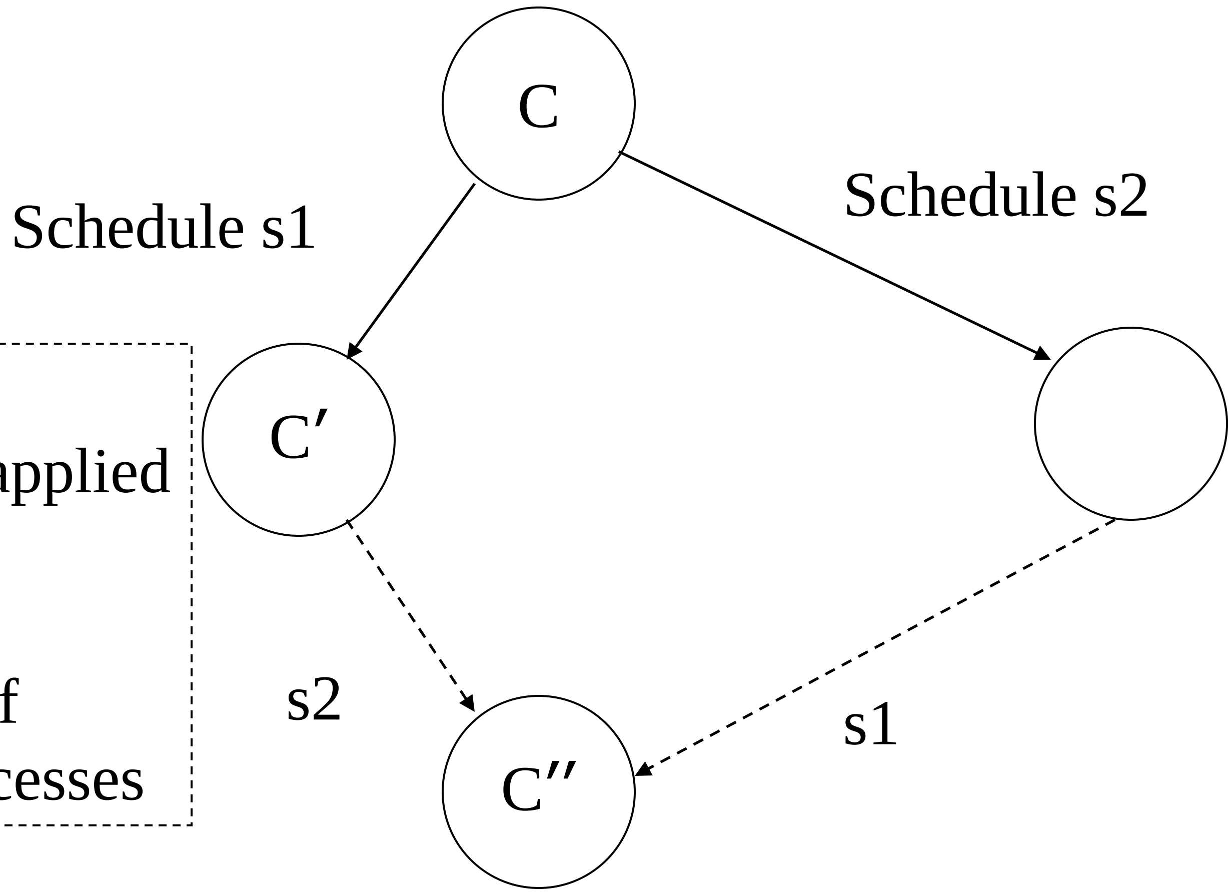
Different Definition of "State"

- State of a process
- Configuration: = Global state. Collection of states, one per process; and state of the global buffer
- Each Event consists atomically of three sub-steps done together:
  - receipt of a message by a process (say p), and
  - processing of message, and
  - sending out of all necessary messages by p (into the global message buffer)
- Note: this event is different from the Lamport events
- Schedule: sequence of events


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

Lemma 1

Schedules are commutative



s1 and s2
- can each be applied to C
- involve <u>disjoint</u> sets of receiving processes

State Valencies

- Let config. C have a set of decision values V reachable from it
  - If |V| = 2, config. C is <u>bivalent</u> (i.e., system could lead to either 0-consensus or 1-consensus)
  - If |V| = 1, config. C is said univalent. If it leads to 0-consensus, then we call it 0-valent (similarly 1-valent)
- Bivalent means outcome is unpredictable

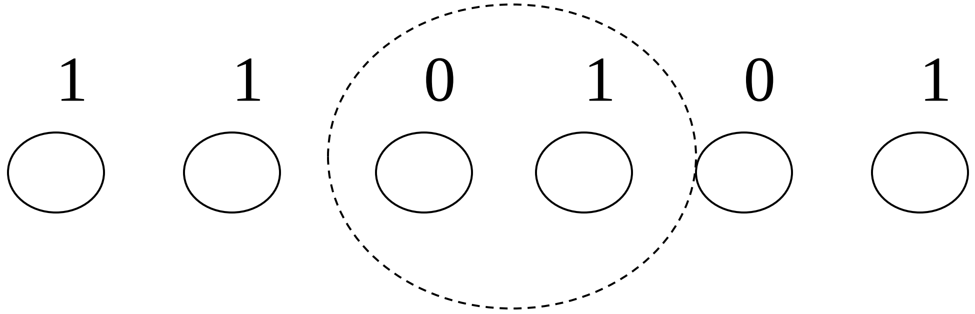
What we'll Show

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 2

Some initial configuration is bivalent

- Suppose all initial configurations were either 0-valent or 1-valent (but none bivalent)
- Place all configurations side-by-side, where adjacent configurations differ in initial xp value for *exactly one* process.
- Creates a lattice of states

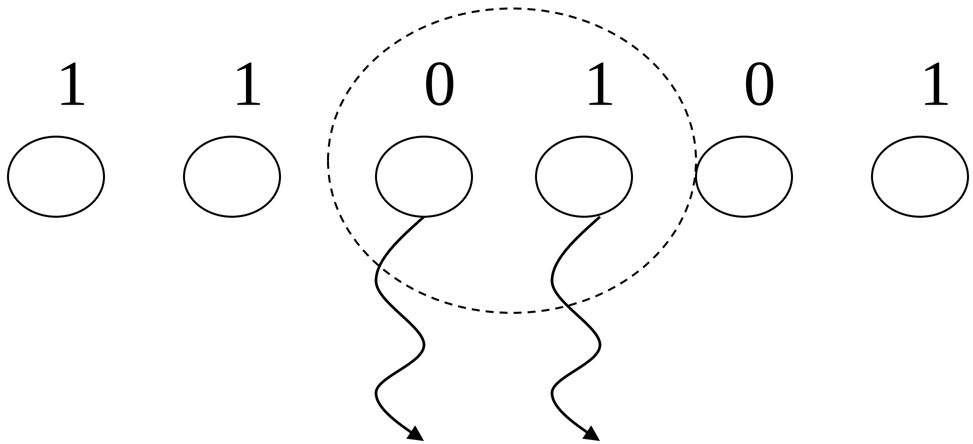


- There *has* to be some adjacent pair of 1-valent and 0-valent configs.

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 be the one with a different state across these two configs.
•Now consider the world where process p has crashed



Both these initial configs. are *indistinguishable*. But one gives a 0 decision value. The other gives a 1 decision value.

So, both these initial configs. are bivalent when there is a failure

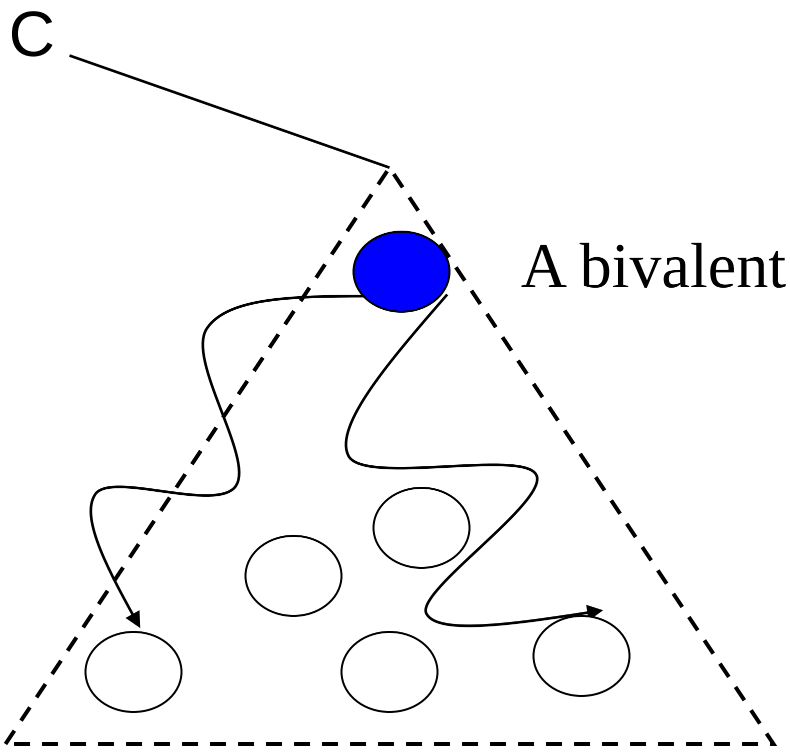
What we'll Show

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

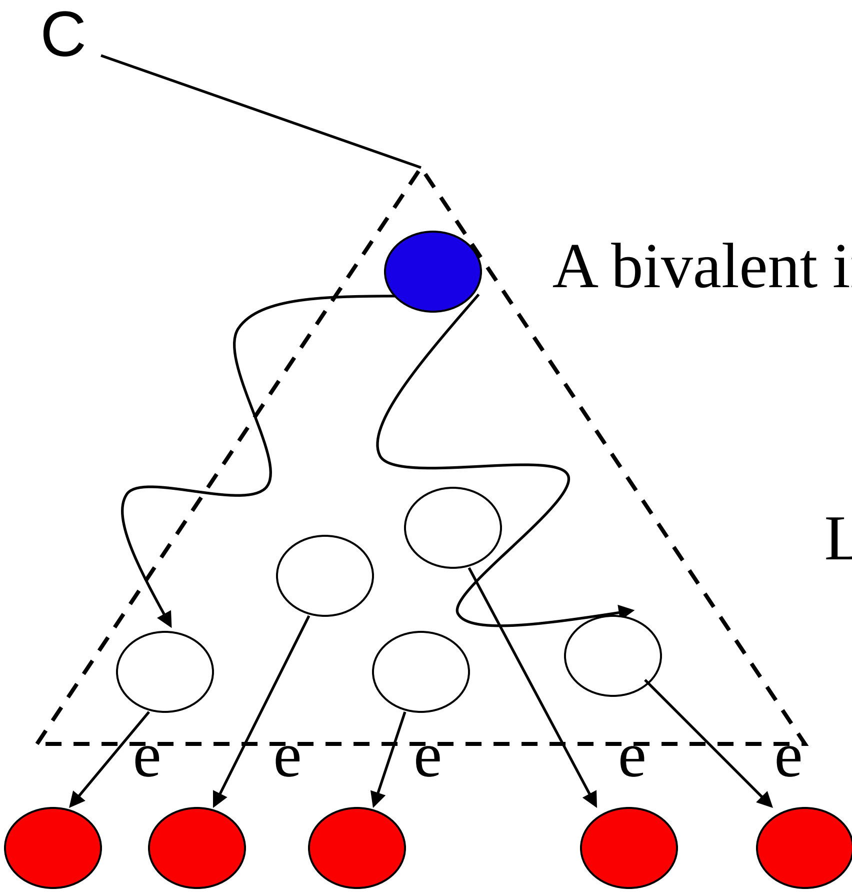


A bivalent initial config.

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

Let C be the set of configs. reachable <u>without</u> applying e

Lemma 3



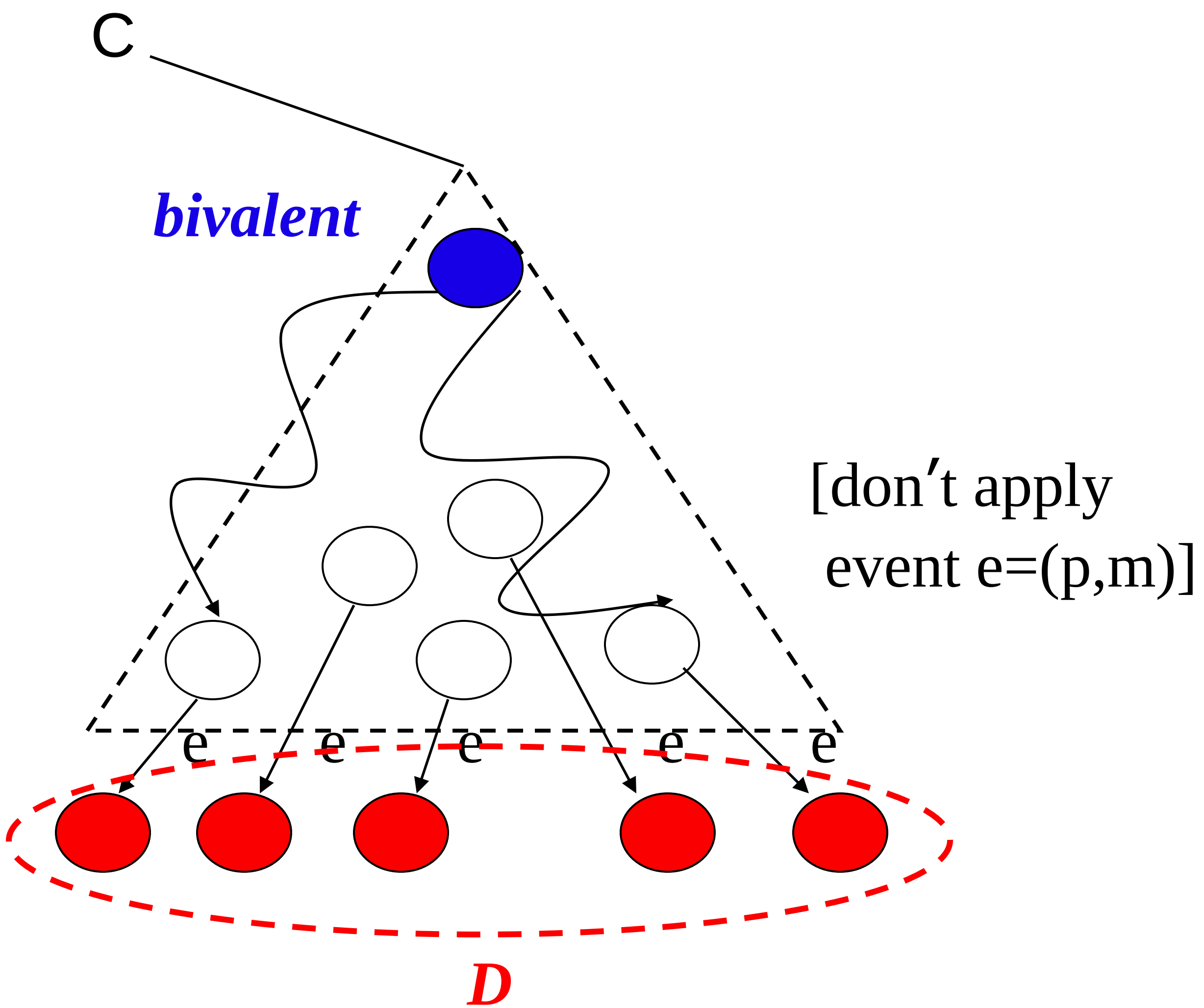
A bivalent initial config.

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

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

Let ***D*** be the set of configs. obtained <ins>by applying single event e</ins> to any config. in ***C***

Lemma 3

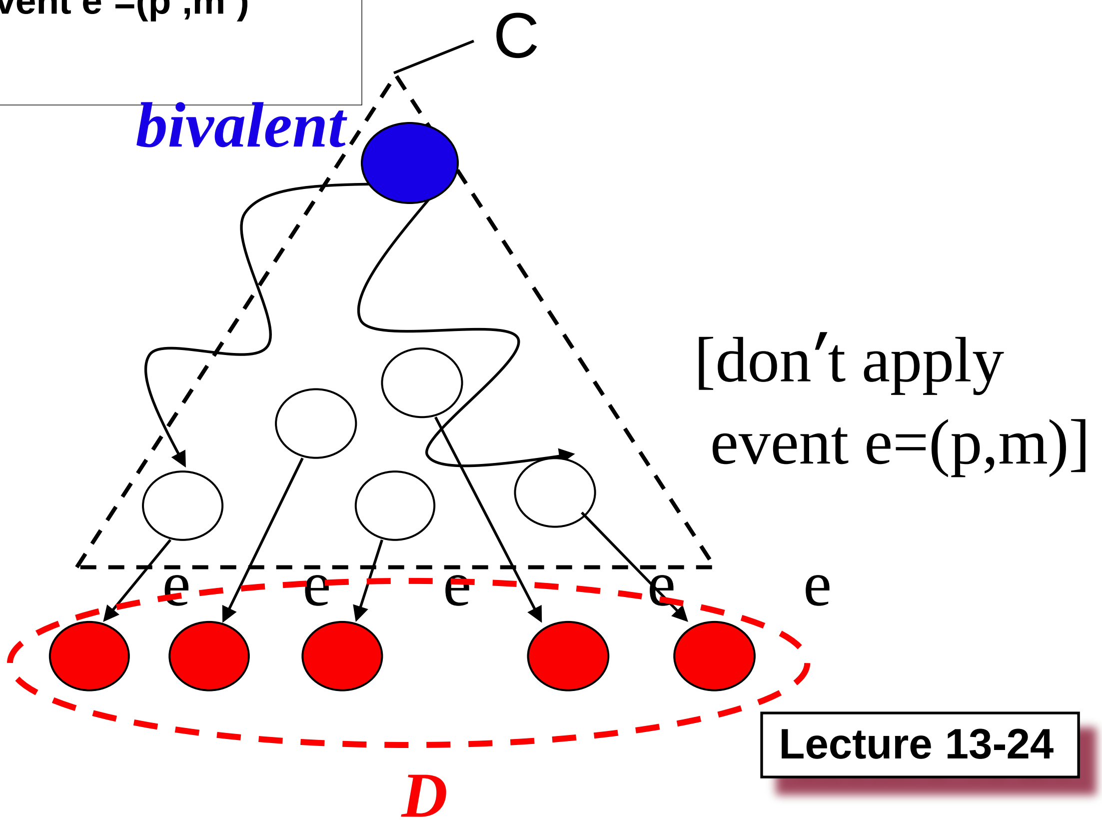


Claim. **Set D contains a bivalent config.**

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

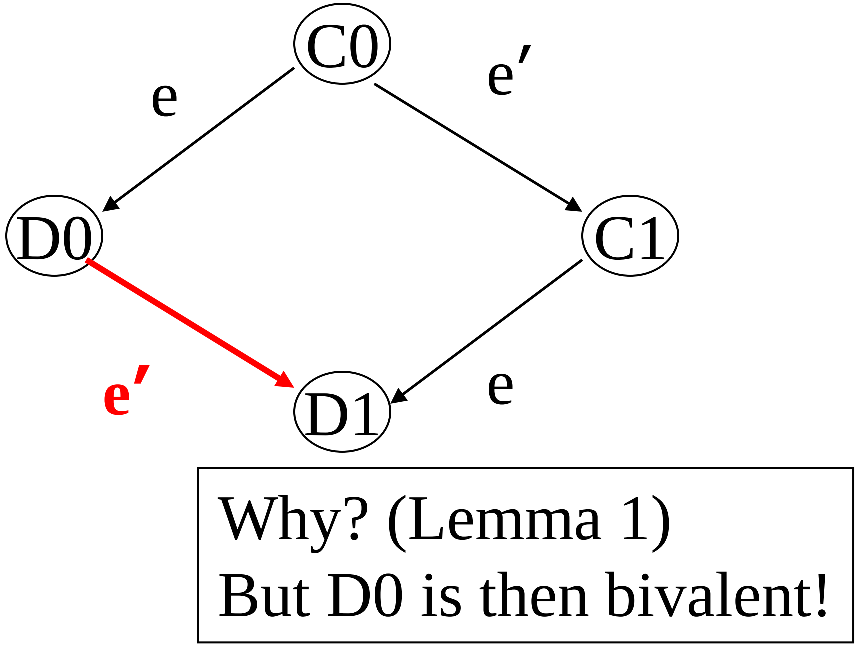
- **There are states D0 and D1 in *D*, and C0 and C1 in *C* such that**
  - **D0 is 0-valent, D1 is 1-valent**
  - **D0=C0 foll. by e=(p,m)**
  - **D1=C1 foll. by e=(p,m)**
  - **And C1 = C0 followed by some event e′=(p′,m′)**

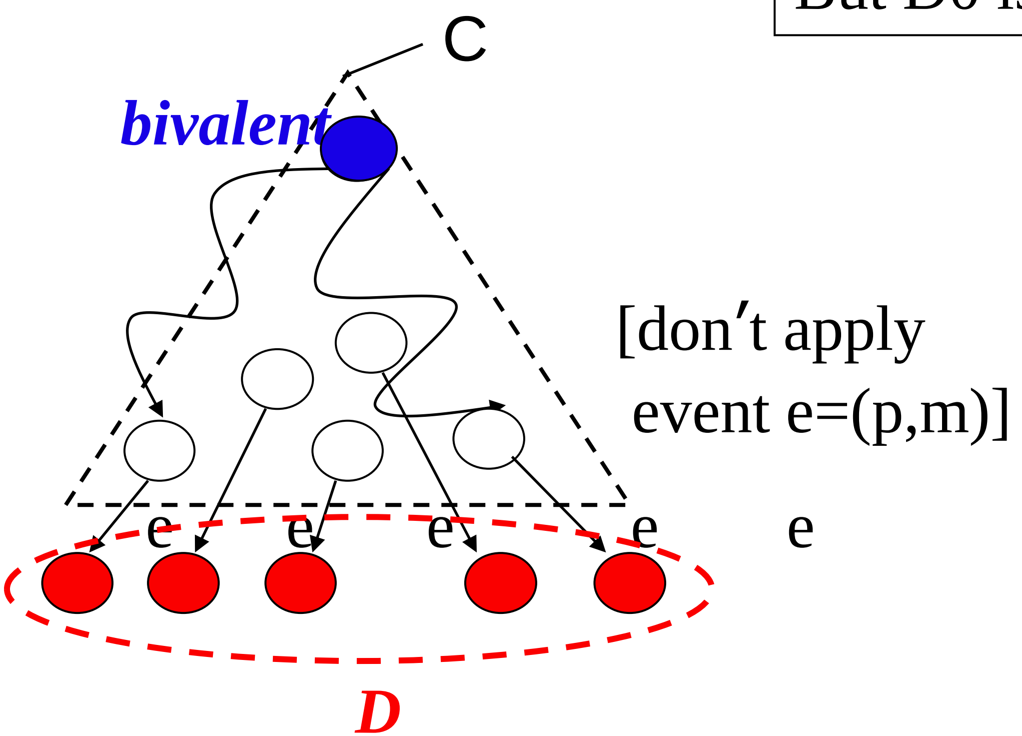
  (why?)



Proof. (contd.)

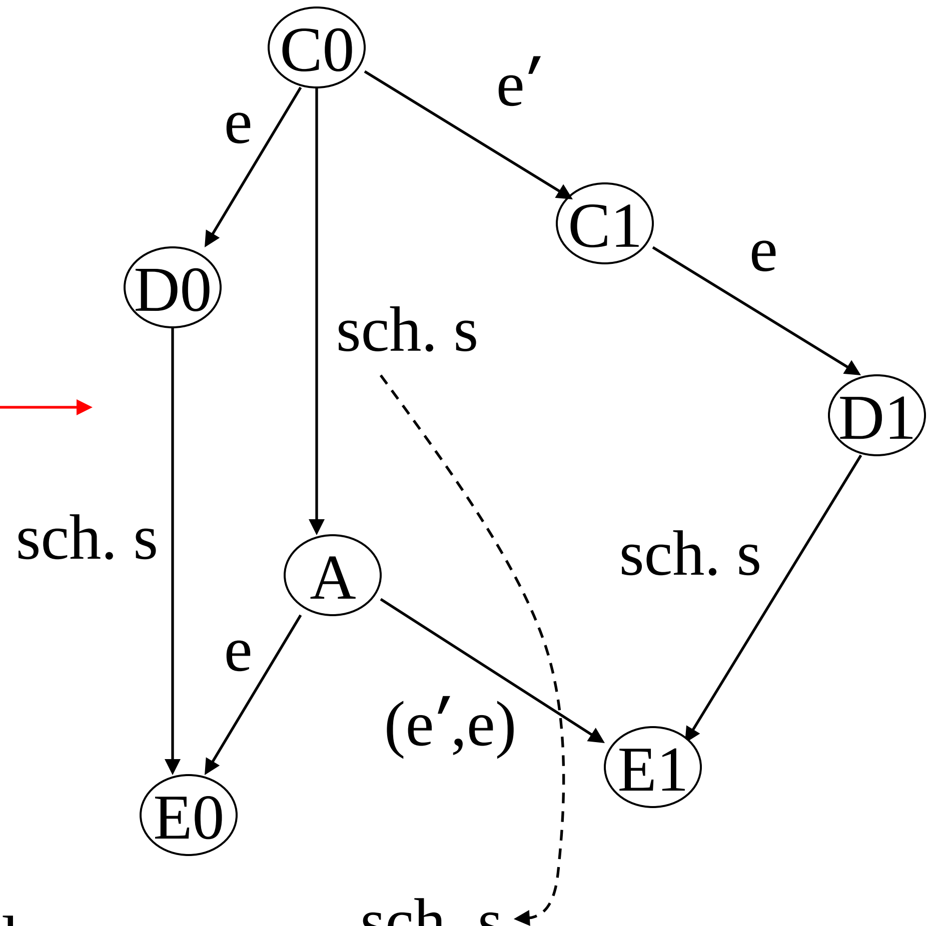
- Case I: p′ is not p
- Case II: p′ same as p



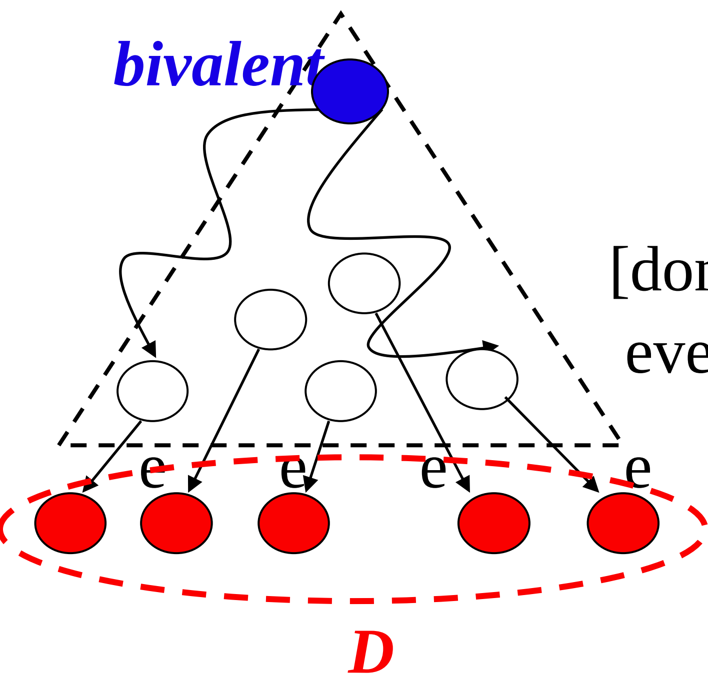


Proof. (contd.)

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



sch. s
- finite
- <u>deciding run</u> from C0
  (i.e., A decides 0)
- <u>p takes no steps</u>



[don′t apply
event e=(p,m)]

e

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

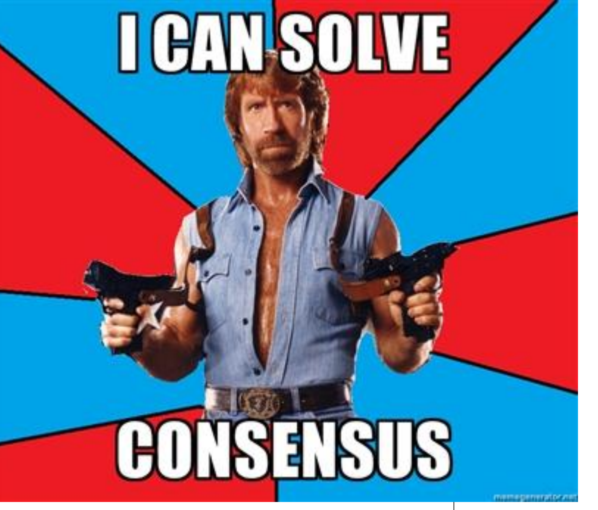
- Theorem (Impossibility of Consensus): There is always a run of events in an asynchronous distributed system (given any algorithm) such that the group of processes never reaches consensus (i.e., always stays bivalent)
  - "The devil's advocate always has a way out"

Why is Consensus Important?

Many problems in distributed systems are equivalent to *(or harder than)* consensus!

- Agreement, e.g., on an integer (harder than consensus, since it can be used to solve consensus) is impossible!
- Leader election is impossible!
  - » **A leader election algorithm can be designed using a given consensus algorithm as a black box**
  - » **A consensus protocol can be designed using a given leader election algorithm as a black box**
- Accurate Failure Detection is impossible!
  - » **Should I mark a process that has not responded for the last 60 seconds as failed? (It might just be very, very, slow)**
  - » **Completeness + Accuracy impossible to guarantee**

What can we do about it?



- **One way is to design** *Probabilistic Algorithms*
  - *E.g., probabilistic accuracy in failure detector algorithms*
- **Another way is to design <u>safe</u> algorithms that have some chance (when network is good) of making a decision, *e.g., Paxos***
  - **(We'll discuss this later in the course)**
  - **A lot of companies/datacenters use Paxos or its variants (e.g., Google's Chubby system)**

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 only one process failure and arbitrarily slow processes, there are always sequences of events for the system to decide any which way. Regardless of which consensus algorithm is running underneath.
  - FLP impossibility proof

Next Week

- Thursday – HW2 due

- Midterm next Tuesday October 16th
  - Location: Here! (1310 DCL)
  - Syllabus: Lectures 1-12, HWs1-2, MPs1-2.
  - Closed book, closed notes. NO cheatsheets or calculators.
  1. Multiple choice questions
  2. Big problems: like HW problems, either design or application
- Practice midterm posted on Website (Assignments page) – no solutions will be posted
  - Please use our office hours!

Optional Slides (Not Covered)

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

Consensus Protocol correct if

1. No accessible config. (config. reachable from an initial config.) has > 1 decision value
2. For each v in {0,1}, there is an accessible config. (reachable from some initial state) that has value v
   - avoids trivial solution to the consensus problem