

# CS 525 Advanced Topics in Distributed Systems Spring 08

Indranil Gupta (Indy)  
Lecture 5  
Distributed Systems Fundamentals  
January 31, 2008

1

## Agenda

- I. Synchronous versus Asynchronous systems
- II. Lamport Timestamps
- III. Global Snapshots
- IV. Impossibility of Consensus proof

2

## I. Two Different System Models

### Synchronous Distributed System

- Each message is received within bounded time
- Drift of each process' local clock has a known bound
- Each step in a process takes  $lb < \text{time} < ub$

Ex: A collection of processors connected by a communication bus, e.g., a Cray supercomputer or a multicore machine

### Asynchronous Distributed System

- No bounds on process execution
- The drift rate of a clock is arbitrary
- No bounds on message transmission delays

Ex: The Internet is an asynchronous distributed system, so are ad-hoc and sensor networks

- ❑ This is a more general (and thus challenging) model than the synchronous system model. A protocol for an asynchronous system will also work for a synchronous system (though not vice-versa)

- ❑ It would be **impossible** to accurately synchronize the clocks of two communicating processes in an asynchronous system

3

## II. Logical Clocks

- ❖ But is accurate (or approximate) clock sync. even required?
- ❖ Wouldn't a **logical ordering** among **events at processes** suffice?

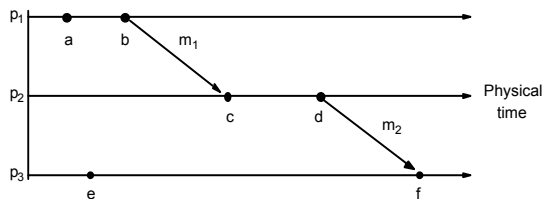
### Lamport's happens-before ( $\rightarrow$ ) among **events**:

- ❑ On the same process:  $a \rightarrow b$ , if  $\text{time}(a) < \text{time}(b)$
- ❑ If p1 sends  $m$  to p2:  $\text{send}(m) \rightarrow \text{receive}(m)$
- ❑ If  $a \rightarrow b$  and  $b \rightarrow c$  then  $a \rightarrow c$

### Lamport's **logical timestamps** preserve causality:

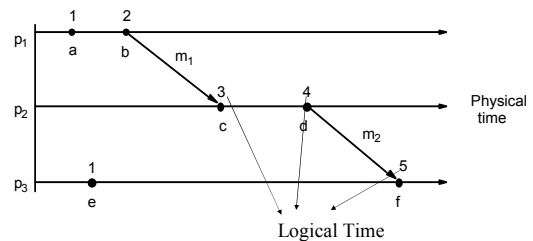
- ❑ All processes use a **local counter** (logical clock) with initial value of zero
- ❑ Just before each **event**, the local counter is incremented by 1 and assigned to the event as its timestamp
- ❑ A **send (message)** event carries its timestamp
- ❑ For a **receive (message)** event, the counter is updated by  $\max(\text{receiver's-local-counter}, \text{message-timestamp}) + 1$

## Example



5

## Lamport Timestamps



- Logical timestamps preserve **causality of events**, i.e.,  $a \rightarrow b \implies TS(a) < TS(b)$
- Can be used instead of physical timestamps

6



## IV. Give it a thought

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

The fault does not lie with Microsoft Corp. or Apple Inc. or Cisco

The fault lies in the [impossibility of consensus](#)

13

## What is Consensus?

- N processes
- Each process p has
  - input variable  $x_p$  : initially either 0 or 1
  - output variable  $y_p$  : initially b
- **Consensus problem**: design a protocol so that at the end, either:
  1. all processes set their output variables to 0
  2. Or all processes set their output variables to 1
  - Also, there is at least one initial state that leads to each outcome above

14

## Why is Consensus Important

- Many problems in distributed systems are **equivalent to (or harder than)** consensus!
  - Agreement (harder than consensus, since it can be used to solve consensus)
  - Leader election (select exactly one leader, and every alive process knows about it)
  - Failure Detection
- **Consensus using leader election**  
Choose 0 or 1 based on the last bit of the identity of the elected leader.

15

## Let's Try to Solve Consensus!

- Uh, what's the **model**? (assumptions!)
- **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!
- **Processes can fail by stopping (crash-stop or crash failures)**

16

## Consensus in a Synchronous System

- For a system with at most  $f$  processes crashing Possible to achieve!
  - All processes are synchronized and operate in "rounds" of time
  - the algorithm proceeds in  $f+1$  rounds (with timeout), using reliable communication to all members -  $Values^r_i$ : the set of proposed values known to  $P_i$  at the beginning of round  $r$ .
- Initially  $Values^0_i = \{x_i\}$ ;  $Values^1_i = \{v_i\}$   
for round = 1 to  $f+1$  do  
    multicast ( $Values^r_i$  -  $Values^{r-1}_i$ )  
     $Values^{r+1}_i \leftarrow Values^r_i$   
    for each  $V_j$  received  
         $Values^{r+1}_i = Values^{r+1}_i \cup V_j$   
    end  
end  
 $d_i = \text{minimum}(Values^{f+1}_i)$

17

## Why does the Algorithm Work?

- Proof by contradiction.
- Assume that two non-faulty processes, say  $p_i$  and  $p_j$ , differ in their final set of values (i.e., after  $f+1$  rounds)
- Assume that  $p_i$  possesses a value  $v$  that  $p_j$  does not possess.
  - $p_i$  must have received  $v$  in the **very last** round (why?)
  - A third process,  $p_k$ , sent  $v$  to  $p_i$ , and crashed before sending  $v$  to  $p_j$ .
  - Similarly, a fourth process sending  $v$  in the **last-but-one round** must have crashed; otherwise, both  $p_k$  and  $p_j$  should have received  $v$ .
  - Proceeding in this way, we infer at least one (unique) crash in each of the preceding rounds.
  - This means a total of  $f+1$  crashes, while we have assumed at most  $f$  crashes can occur → contradiction.

18

## Consensus in an Asynchronous System

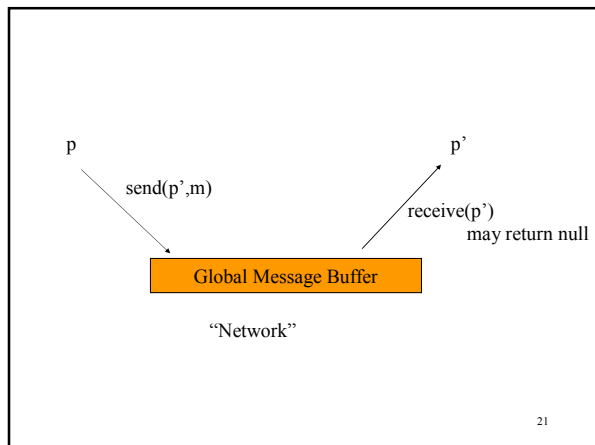
- **Impossible to achieve!**
  - even a single failed process is enough to avoid the system from reaching agreement
- Proved in a now-famous result by Fischer, Lynch and Patterson, 1983 (FLP)
  - Stopped many distributed system designers dead in their tracks
  - A lot of claims of “reliability” vanished overnight

19

## Recall

- Each process  $p$  has a **state**
  - program counter, registers, stack, local variables
  - input register  $x_p$  : initially either 0 or 1
  - output register  $y_p$  : initially  $b$  (undecided)
- Consensus Problem: design a protocol so that either
  - all processes set their output variables to 0
  - Or all processes set their output variables to 1
- For impossibility proof, OK to consider (i) more restrictive system model, and (ii) easier problem

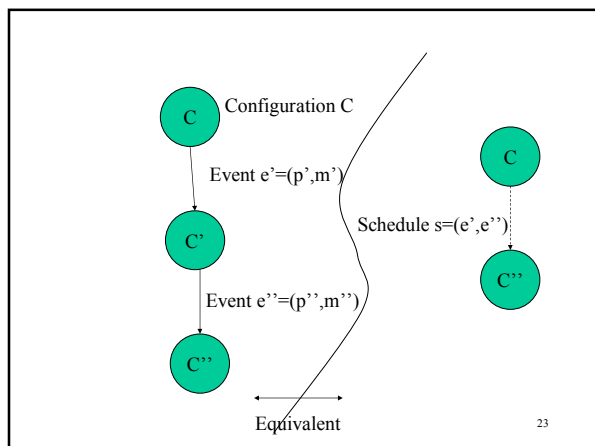
20



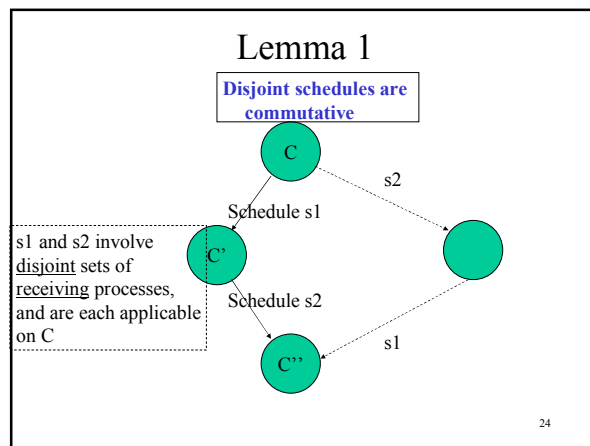
21

- State of a process
- **Configuration=global state.** Collection of states, one for each process; and 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

22



23



24

## Easier Consensus Problem

Easier Consensus Problem: **some** process eventually sets  $x_p$  to be 0 or 1  
**Only one process crashes** – we're free to choose which one

25

- 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**

26

## What the FLP Proof Shows

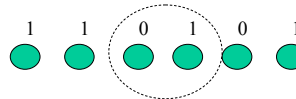
- There exists an initial configuration that is bivalent
- Starting from a bivalent config., there is always another bivalent config. that is reachable

27

## 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  $x_p$  value for exactly one process.



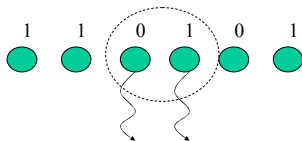
- There has to be **some** adjacent pair of 1-valent and 0-valent configs.

28

## 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 **bivalent** when there is such a failure

29

## What we'll Show

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

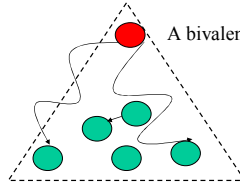
30

### Lemma 3

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

31

### Lemma 3



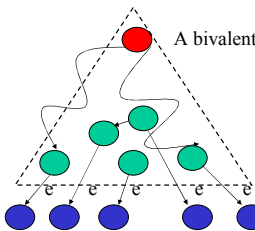
A bivalent initial config.

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

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

32

### Lemma 3



A bivalent initial config.

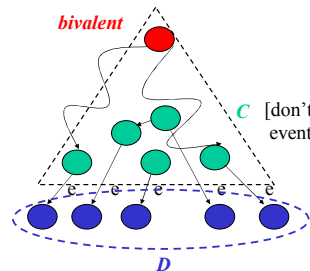
let  $e=(p,m)$  be some event  
applicable to the initial config

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  $C$

33

### Lemma 3



*bivalent*

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

$D$

34

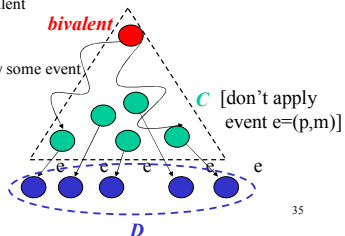
**Claim.** Set  $D$  contains a bivalent config.

**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  $e'=(p',m')$

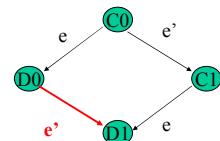
(why?)



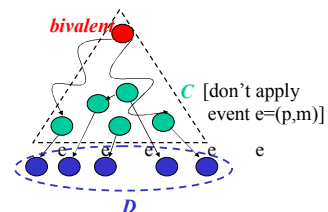
35

**Proof.** (contd.)

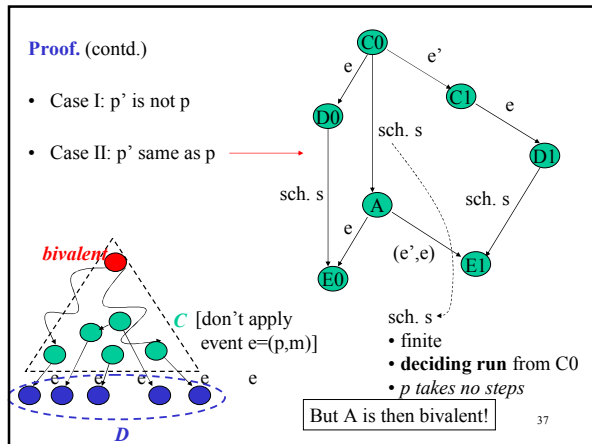
- Case I:  $p'$  is not  $p$
- Case II:  $p'$  same as  $p$



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



36



### Lemma 3

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

38

### 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 reach consensus (i.e., stays bivalent all the time)**

39

### 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

40

### Announcements

- No office hours today

41

### 2 Weeks from now

- Student led presentations start
  - Organization of presentation is up to you
  - Suggested: describe background and motivation for the session topic, present an example or two, then get into the paper topics
    - Make sure you read relevant background papers in addition to the Main Papers! Look at the reference list in the Main Papers...
- Reviews: You have to submit both an **email copy** (which will appear on the course website) **and a hardcopy** (on which I will give you feedback). See website for detailed instructions.

42

## Before Next Lecture

- Sign up for a presentation slot if you have not already!
- Read the two papers for the topic “The Grid” for next lecture
- Read the 2 optional papers for today’s session (first the one on CSP, and then the one on the State Machine approach)
  - From now on, I will assume that you have read these papers (these are classics and form the basics of a lot of what we will discuss in the future sessions in this course!)

43