CS 525
Advanced Distributed Systems
Spring 2018

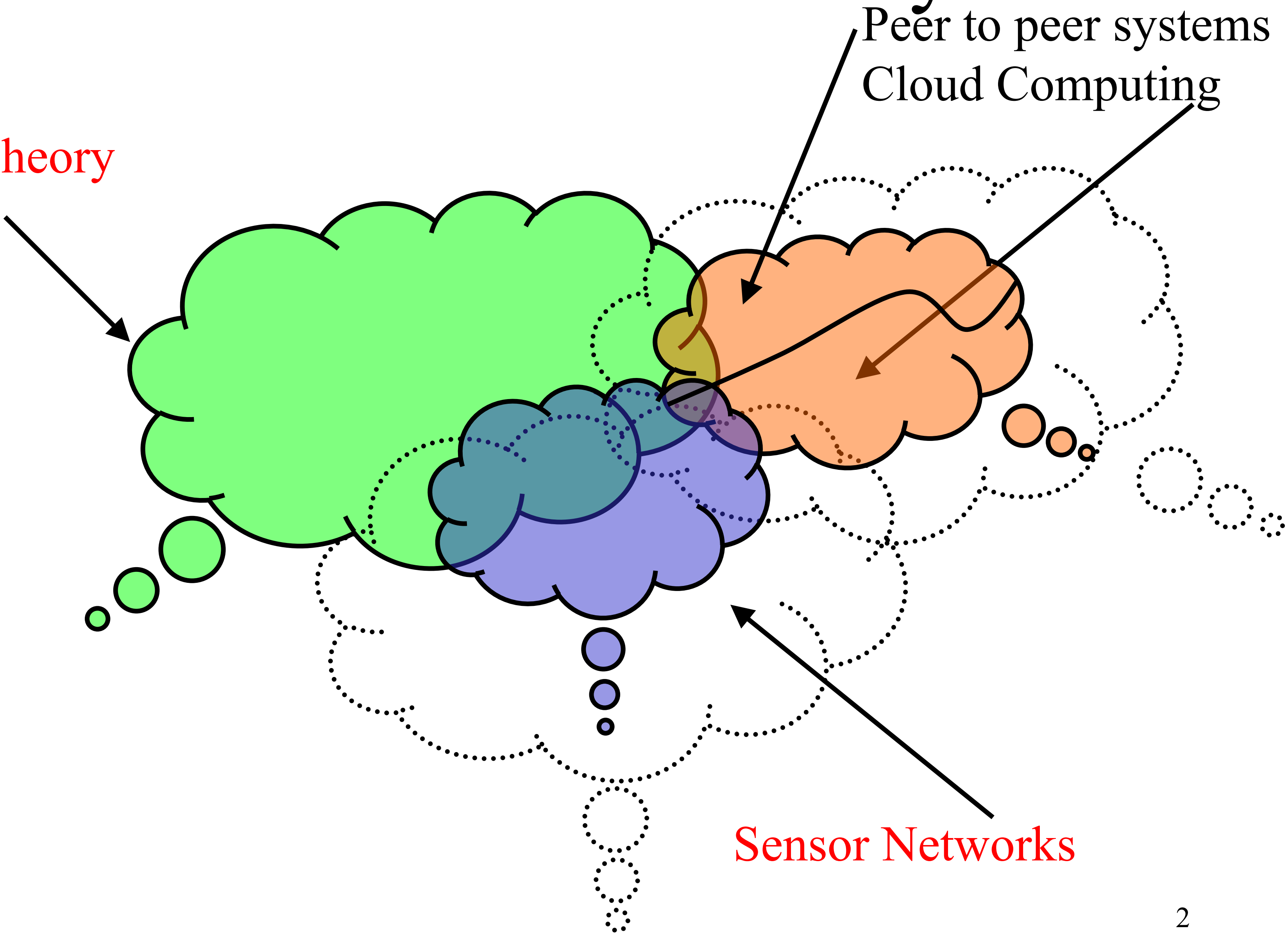
Indranil Gupta (Indy)

Lecture 7

Distributed Algorithms Fundamentals + Introduction to Sensor Networks

February 7, 2018

CS 525 and Distributed Systems



D.S. Theory

Peer to peer systems

Cloud Computing

Sensor Networks

Distributed Algorithms Fundamentals – Outline

I. Synchronous versus Asynchronous systems

II. Lamport Timestamps

III. Global Snapshots

IV. Impossibility of Consensus proof

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 < 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

  Ex: 13 us of GPS satellite error caused 12 hours of problems: http://www.bbc.com/news/technology-35491962

❑ *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)*

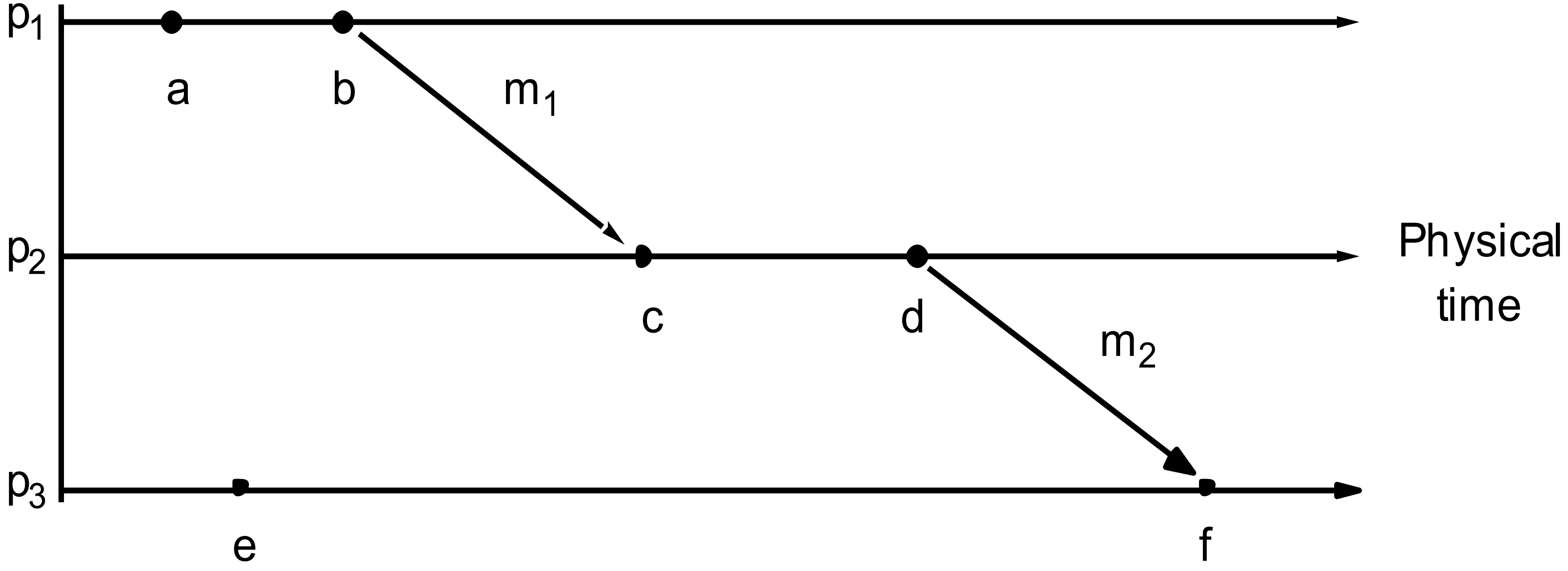
❑ It would be ***im***possible to accurately synchronize the clocks of two communicating processes in an asynchronous system

II. Logical Clocks

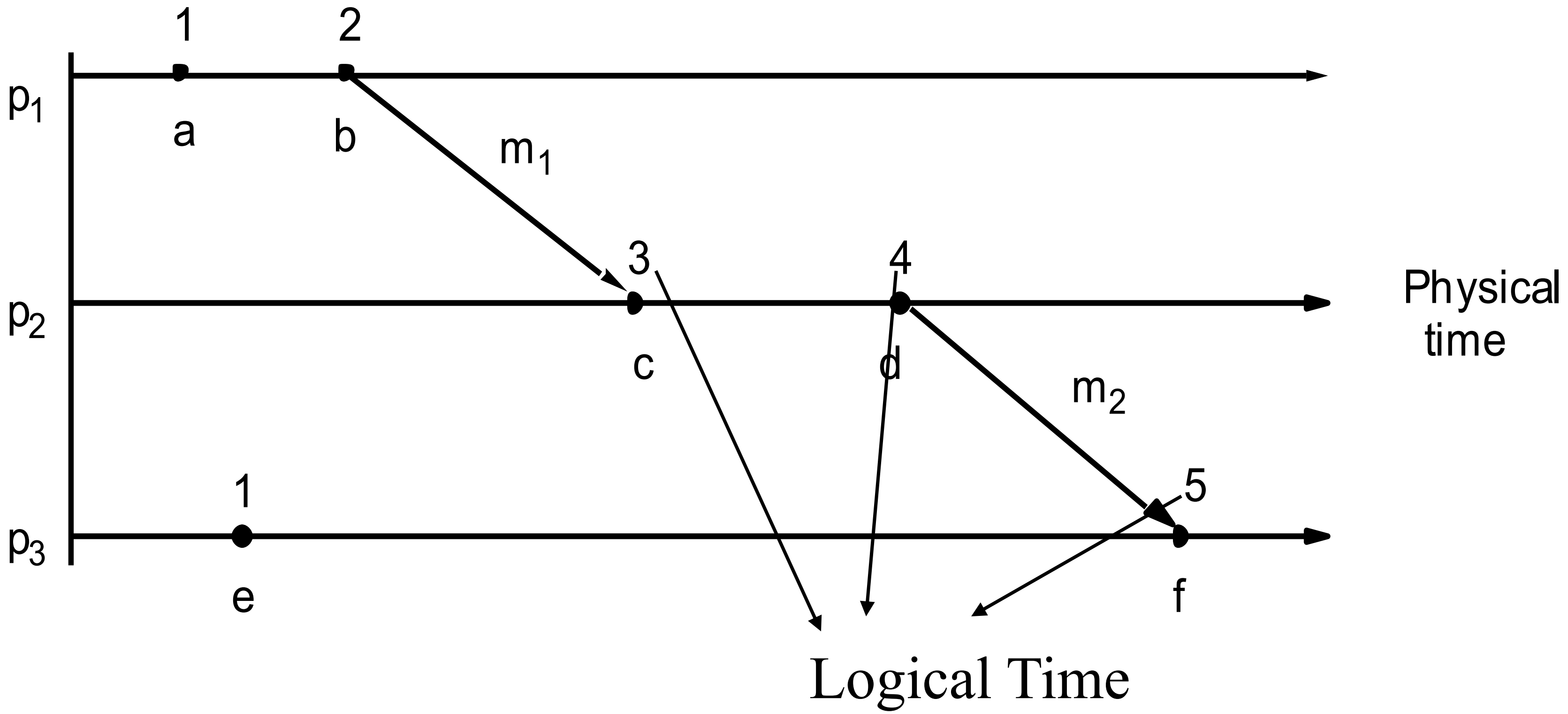
- But is accurate (or approximate) clock sync. even required?
- Wouldn't a **logical ordering** among events at processes suffice?
- Lamport's **happens-before** *(→)* among **events**:
  - On the same process: *a → b*, if *time(a) < time(b)*
  - If p1 sends *m* to p2: *send(m) → receive(m)*
  - If *a → b and b → c* then *a → 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(receiver's-local-counter, message-timestamp) + 1

Example



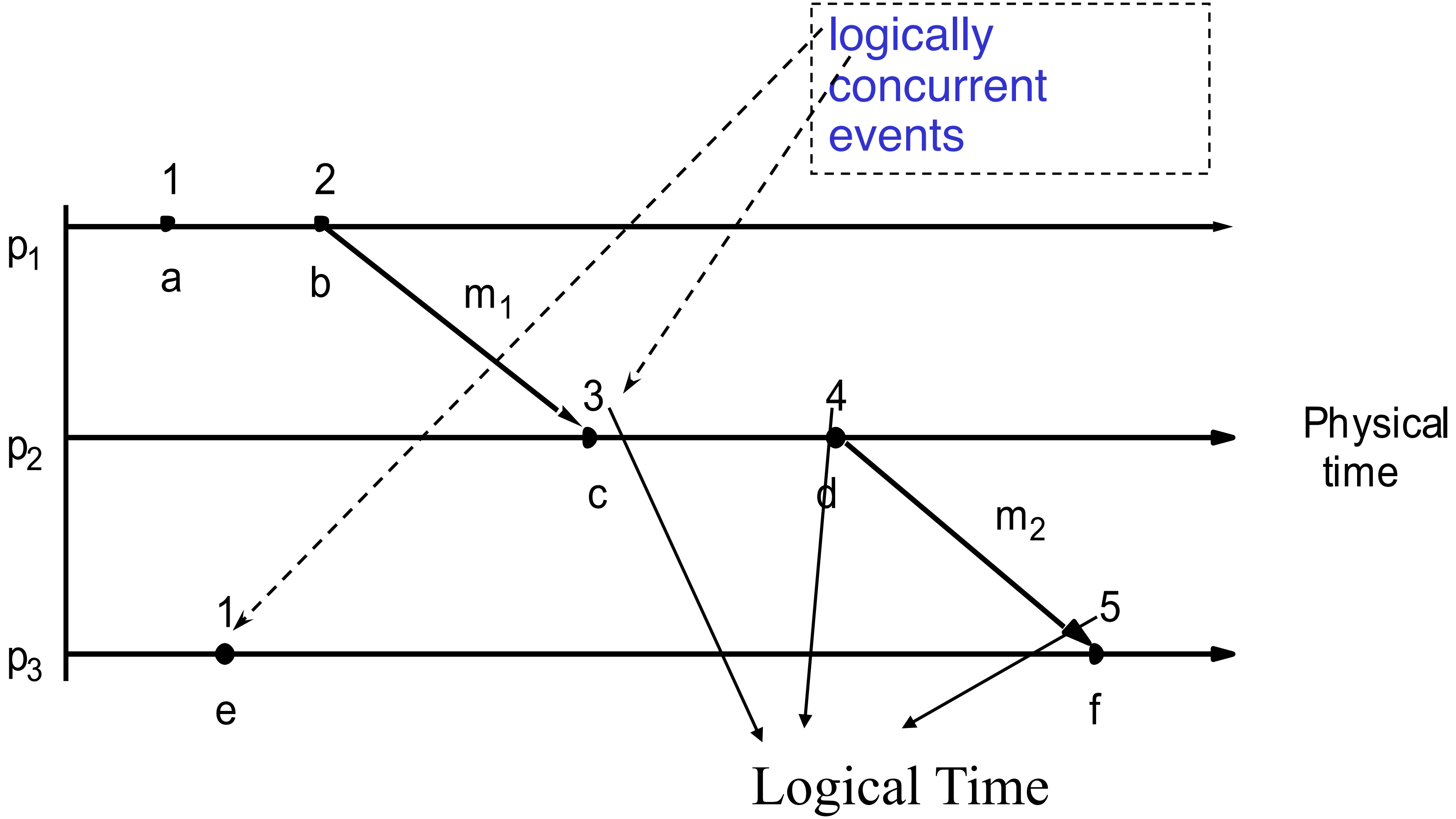
Lamport Timestamps



- *Logical timestamps preserve causality of events,*

  i.e., a → b ==> TS(a) < TS(b)
- *Can be used instead of physical timestamps*

Lamport Timestamps



- *Logical timestamps preserve causality of events,*

  i.e., $a \rightarrow b \Rightarrow TS(a) < TS(b)$

•*Other way implication may not be true! (may be concurrent)*

III. Global Snapshot Algorithm

- Can you capture (record) the states of all processes and communication channels at exactly 10:04:50 am?
- Is it even necessary to take such an exact snapshot?
- Chandy and Lamport snapshot algorithm: records a *logical (or causal)* snapshot of the system.
- *System Model:*
  - No failures, all messages arrive intact, exactly once, eventually
  - There is a communication path between every process pair
  - Communication channels are unidirectional and FIFO-ordered

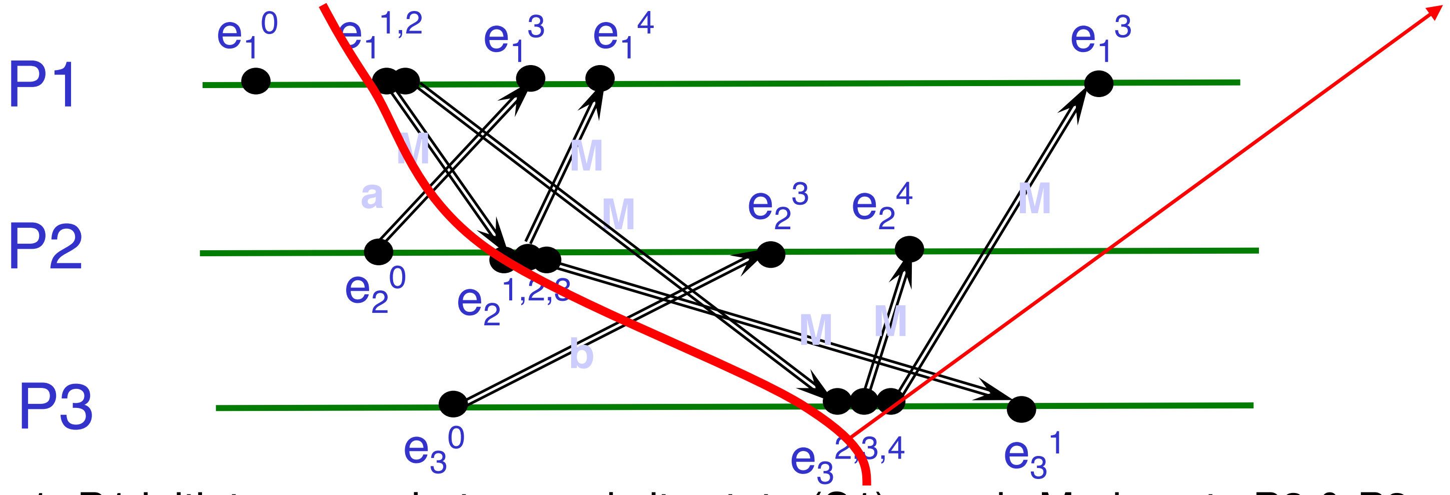
Chandy and Lamport Snapshot Algorithm

1. *Marker (token message) sending rule for initiator process P_0*
   - After P_0 has recorded its state
     - for each outgoing channel C, send a marker on C
2. *Marker receiving rule for a process P_k :*

   On receipt of a marker over channel C
   - *if this is first marker being received at P_k*
     - record P_k' s state
     - record the state of C as "empty"
     - turn on recording of messages over all other incoming channels
     - for each outgoing channel C, send a marker on C
   - *else // messages were already being recorded on channel C*
     - turn off recording messages only on channel C, and mark state of C as = all the messages recorded over C (since recording was turned on, until now)

- ❑ Protocol terminates when every process has received a marker from every other process

Snapshot Example



Consistent Cut

1- P1 initiates snapshot: records its state (S1); sends Markers to P2 & P3; turns on recording for channels C21 and C31

2- P2 receives Marker over C12, records its state (S2), sets state(C12) = {} sends Marker to P1 & P3; turns on recording for channel C32

3- P1 receives Marker over C21, sets state(C21) = {a}

4- P3 receives Marker over C13, records its state (S3), sets state(C13) = {} sends Marker to P1 & P2; turns on recording for channel C23

5- P2 receives Marker over C32, sets state(C32) = {b}

6- P3 receives Marker over C23, sets state(C23) = {}

7- P1 receives Marker over C31, sets state(C31) = {}

Consistent Cut =time-cut across processors and channels so no event to the right of the cut “happens-before” an event that is left of the cut

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

What is Consensus?

- N processes
- Each process p has
  - input variable xp : initially either 0 or 1
  - output variable yp : initially b (can be changed only once)
- 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 (non-triviality)
  - There might be other constraints (Validity=if everyone proposes same value that's what's decided. Integrity = decided value must have been proposed by some process)

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)
  - Perfect Failure Detection
- Consensus using leader election

Choose 0 or 1 based on the last bit of the identity of the elected leader.

- So consensus is a very important problem, and solving it would be really useful!

Possible or not

- In the synchronous system model
  - Consensus is solvable
  - Use a multicast protocol in each round to disseminate all known values, for (N+1) rounds. At the end, everyone has the same value set. Pick from this set using a consistent function.
- In the asynchronous system model
  - Consensus is impossible to solve
  - This means that no matter what protocol/algorithm you suggest, there is always a worst-case possible (with failures and message delays) such that the system is prevented from reaching consensus
  - Powerful result (see the FLP proof in Backup slides of this slide set)
  - Subsequently, safe or probabilistic solutions have become quite popular to consensus or related problems.
  - Paxos is a safe solution (next lecture/session).
  - FLP proof in appendix of slides (peruse in your own time)

Intro to Sensor Networks

A Gram of Gold=How Many Processors?

- Smallest state-of-the-art transistor today is made of a single Gold atom
  - Still in research, not yet in industry.
- Pentium P4 contains 42 M transistors
- Gold atomic weight is 196 ~ 200.
- 1 g of Au contains 3 X 10^21 atoms => 7.5 X 10^18 P4 processors from a gram of Au => 1 billion P4's per person
- CPU speedup ~ √(# transistors on die)

Sensors Today

- IoT = Internet of Things
  - Smart devices in home, office, road, and in our hands and bodies
  - All networked
- Raspberry Pis
- Arduinos
- At core: these are sensors, and sensor networks

Sensor Networks Hype, But do we really need this technology?

- Coal mines have always had CO/CO2 sensors
- Industry has used sensors for a long time

Today…

- Excessive Information
  - Environmentalists collecting data on an island
  - Army needs to know about enemy troop deployments
  - Humans in society face information overload
- Sensor Networking technology can help filter and process this information (And then perhaps respond automatically?)

Growth of a technology requires

I. Hardware

II. Operating Systems and Protocols

III. Killer applications

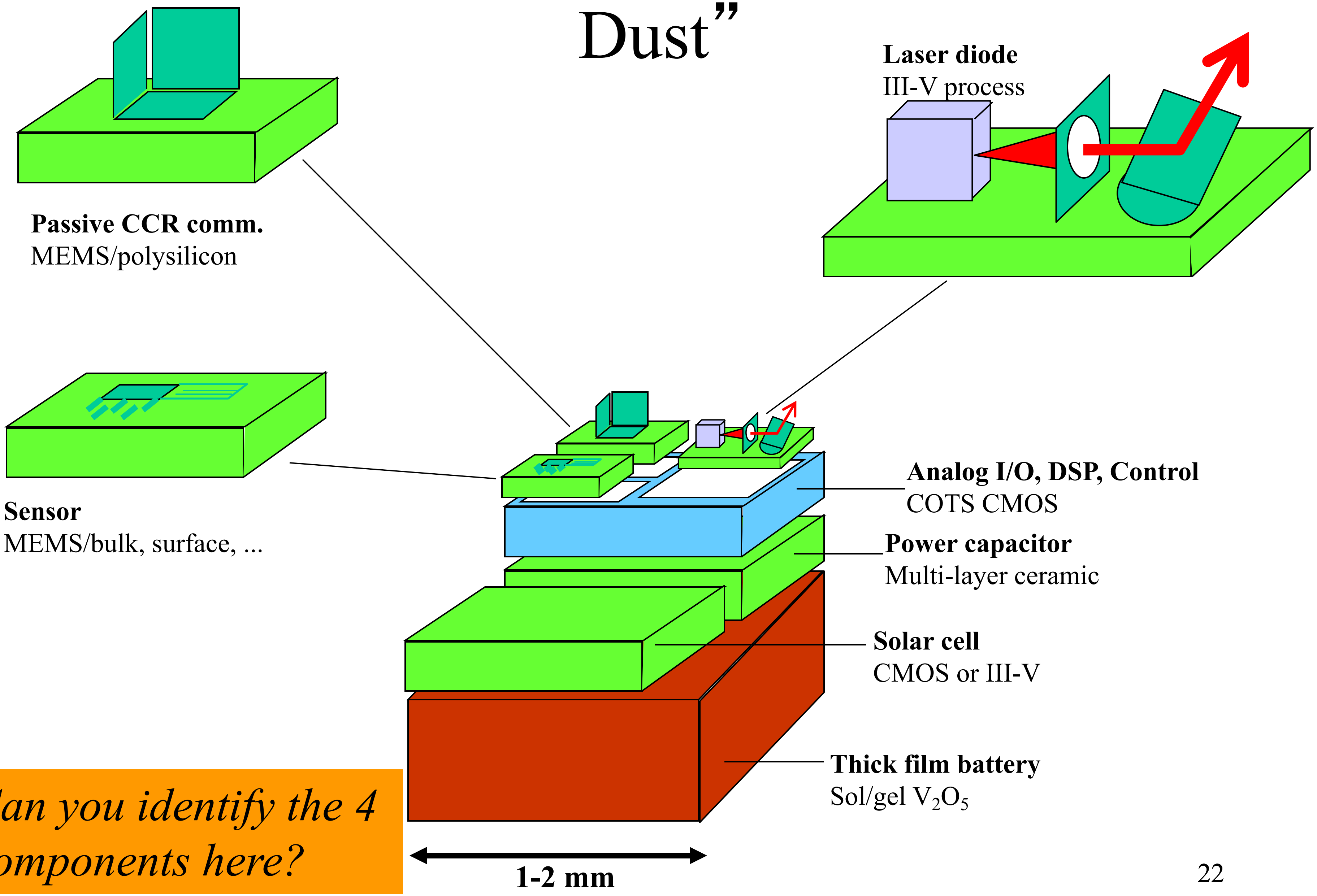
- Military and Civilian

Sensor Nodes

- Motivating factors for emergence: applications, Moore's Law (or variants), wireless comm., MEMS (micro electro mechanical sensors)
- Canonical *Sensor Node* contains

1. Sensor(s) to convert a different energy form to an electrical impulse e.g., to measure temperature
2. Microprocessor
3. Communications link e.g., wireless
4. Power source e.g., battery

Example: Berkeley "Motes" or "Smart Dust"



Passive CCR comm.
MEMS/polysilicon

Laser diode
III-V process

Sensor
MEMS/bulk, surface, ...

Analog I/O, DSP, Control
COTS CMOS

Power capacitor
Multi-layer ceramic

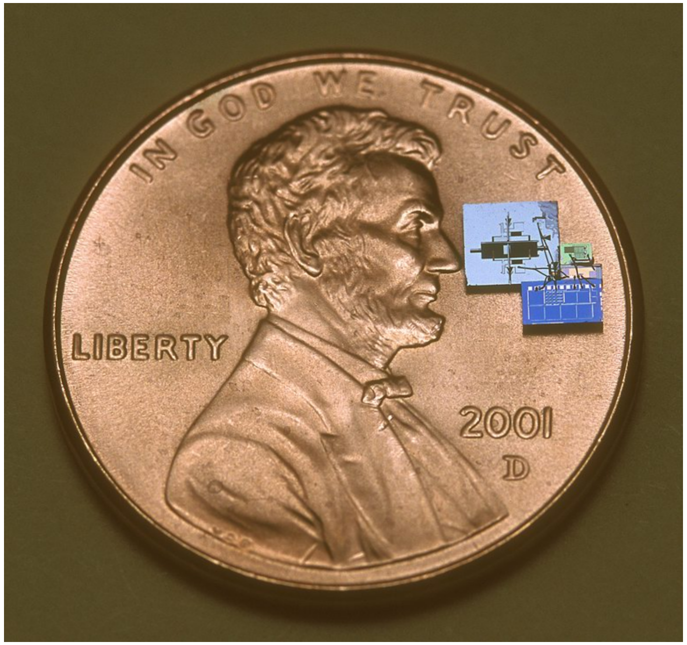
Solar cell
CMOS or III-V

Thick film battery
Sol/gel V_2O_5

1-2 mm

Can you identify the 4 components here?

Example Hardware

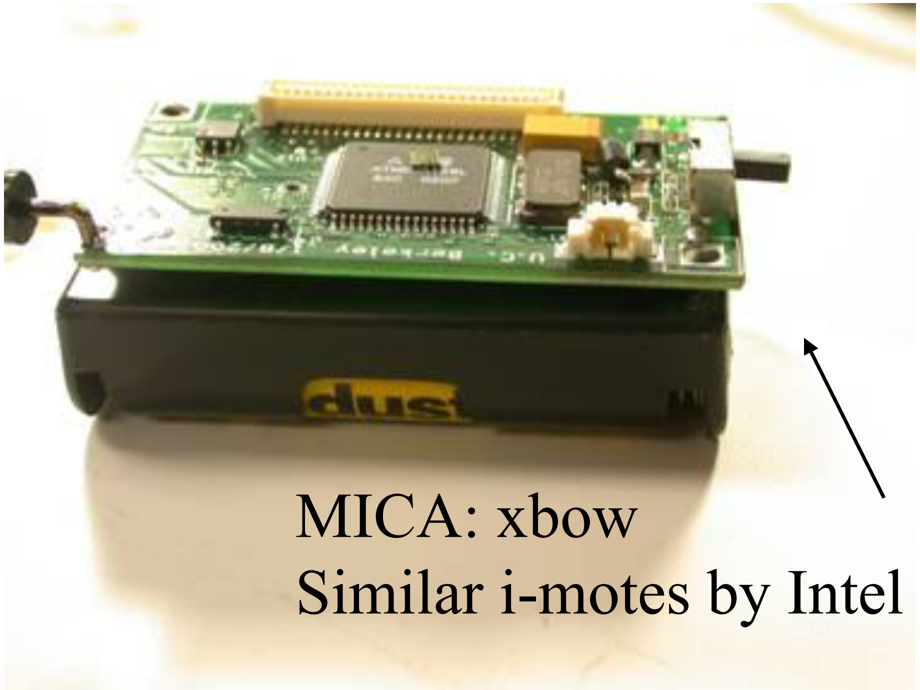


- Size
  - Golem Dust: 11.7 cu. mm
  - MICA motes: Few inches
- Everything on one chip: micro-everything
  - processor, transceiver, battery, sensors, memory, bus
  - MICA: 4 MHz, 40 Kbps, 4 KB SRAM / 512 KB Serial Flash, lasts 7 days at full blast on 2 x AA batteries

Examples

Spec, 3/03

- 4 KB RAM
- 4 MHz clock
- 19.2 Kbps, 40 feet
- Supposedly $0.30

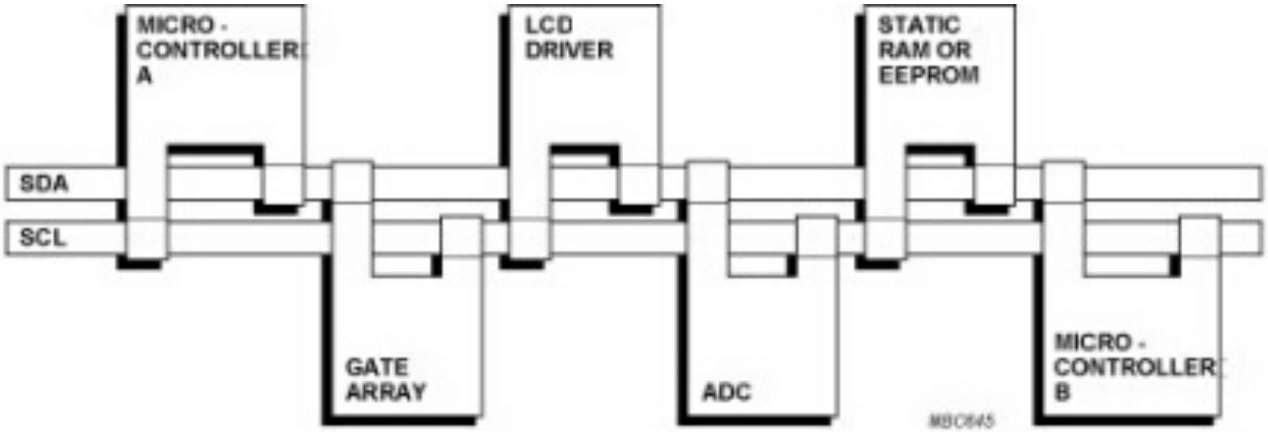


MICA: xbow
Similar i-motes by Intel

Types of Sensors

- Micro-sensors (MEMS, Materials, Circuits)
  - acceleration, vibration, gyroscope, tilt, magnetic, heat, motion, pressure, temp, light, moisture, humidity, barometric, sound
- Chemical
  - CO, CO2, radon
- Biological
  - pathogen detectors
- [Actuators too (mirrors, motors, smart surfaces, micro-robots)]

I2C bus – simple technology



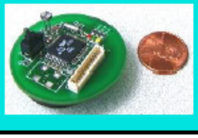
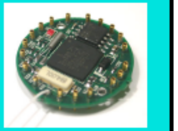
- Inter-IC connect
  - e.g., connect sensor to microprocessor
- Simple features
  - Has only 2 wires
  - Bi-directional
  - serial data (SDA) and serial clock (SCL) bus
- Up to 3.4 Mbps
- Developed By Philips

Transmission Medium

- Spec, MICA: Radio Frequency (RF)
  - Broadcast medium, routing is “store and forward”, links are bidirectional
- Smart Dust : smaller size but RF needs high frequency => higher power consumption

  Optical transmission: simpler hardware, lower power
  - Directional antennas only, broadcast costly
  - Line of sight required
  - Switching links costly : mechanical antenna movements
  - Passive transmission (reflectors) => wormhole routing
  - Unidirectional links

Berkeley Family of Motes

Mote Type	*WeC* 	*René* 	*René 2*	*Dot* 	*Mica* 	*MicaDot* 
Microcontroller						
Type	AT90LS8535		ATmega163		ATmega128	
Program memory (KB)	8		16		128	
RAM (KB)	0.5		1		4	
Nonvolatile storage						
Chip	24LC256				AT45DB041B	
Connection type	I^2C				SPI	
Size (KB)	32				512	
Default power source						
Type	Lithium	Alkaline	Alkaline	Lithium	Alkaline	Lithium
Size	CR2450	2 x AA	2 x AA	CR2032	2 x AA	3B45
Capacity (mAh)	575	2850	2850	225	2850	1000
Communication						
Radio	TR1000					CC1000
Radio speed (kbps)	10	10	10	10	40	38.4
Modulation type	OOK				ASK	FSK

Summary: Sensor Node

- Small Size : few mm to a few inches
- Limited processing and communication
  - MhZ clock, MB flash, KB RAM, 100's Kbps (wireless) bandwidth
- Limited power (MICA: 7-10 days at full blast)
- Failure prone nodes and links (due to deployment, fab, wireless medium, etc.)

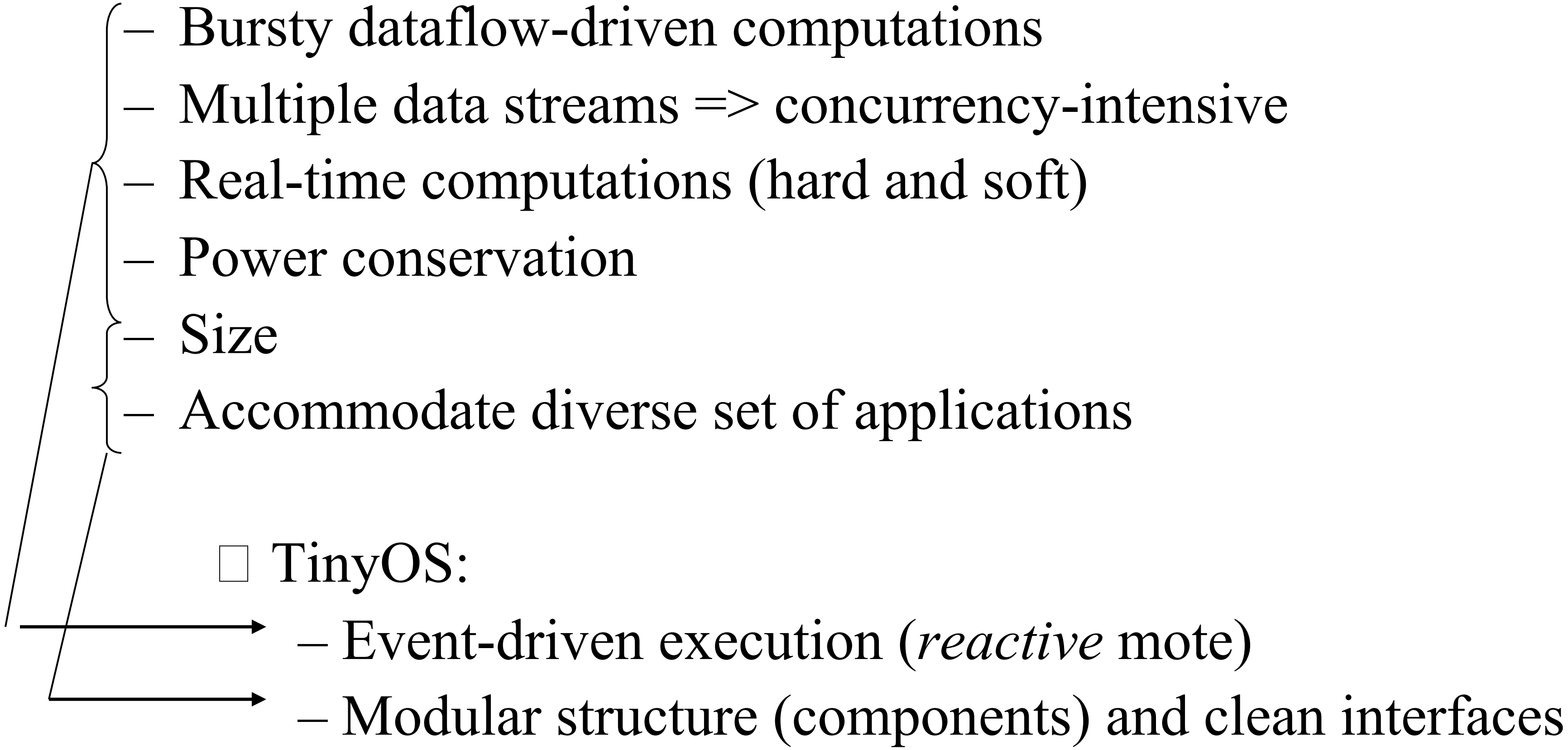
- But easy to manufacture and deploy in large numbers
- *Need to offset this with scalable and fault-tolerant OS's and protocols*

Sensor-node Operating System

Issues

- Size of code and run-time memory footprint
  - Embedded System OS’s inapplicable: need hundreds of KB ROM
- Workload characteristics
  - Continuous ? Bursty ?
- Application diversity
  - Want to reuse sensor nodes
- Tasks and processes
  - Scheduling
  - Hard and soft real-time
- Power consumption
- Communication

TinyOS design point



- Bursty dataflow-driven computations
- Multiple data streams => concurrency-intensive
- Real-time computations (hard and soft)
- Power conservation
- Size
- Accommodate diverse set of applications

□ TinyOS:

- Event-driven execution (*reactive* mote)
- Modular structure (components) and clean interfaces

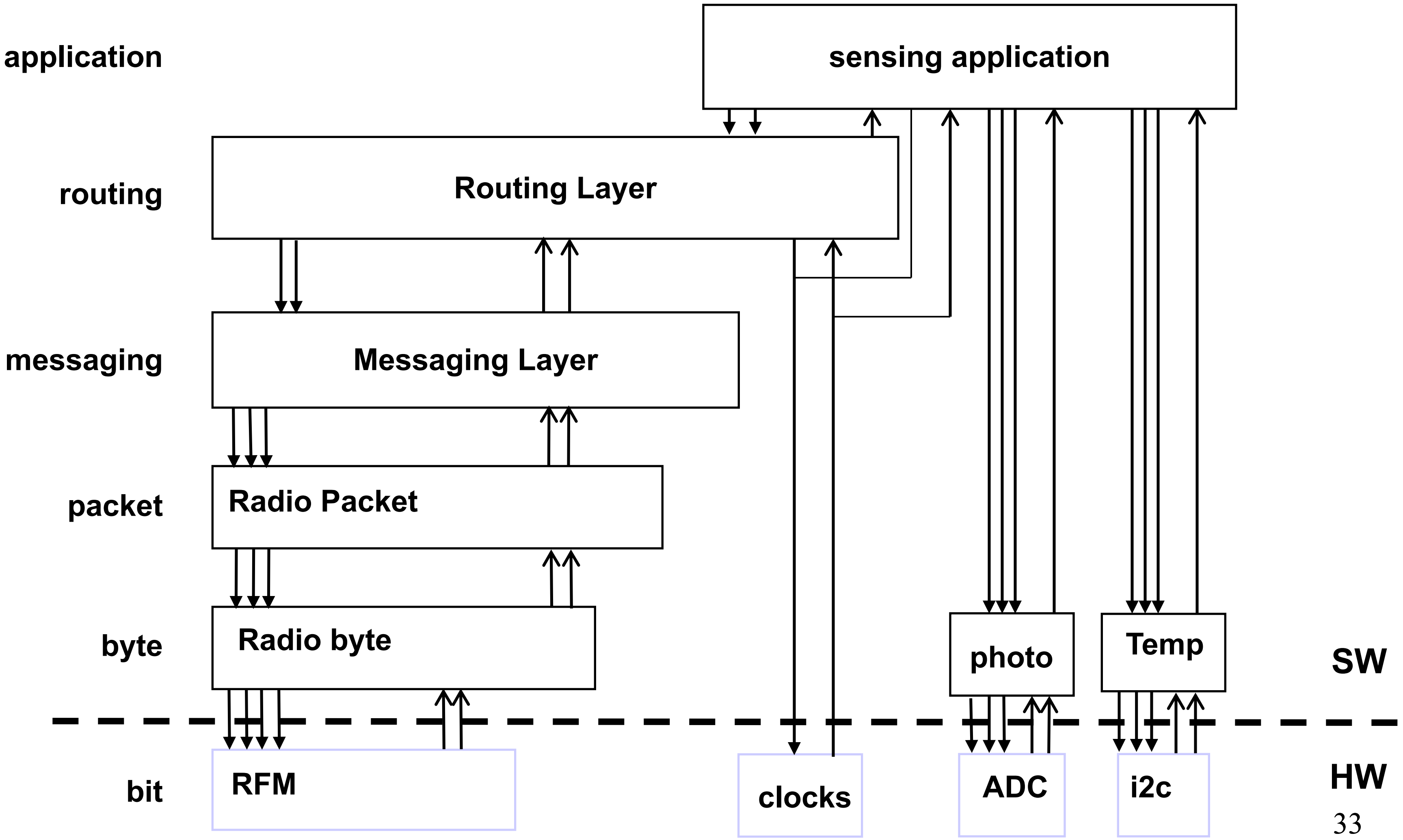
Programming TinyOS

- Use a variant of C called NesC
- NesC defines *components*
- A component is either
  - A *module* specifying a set of methods and internal storage (~like a Java static class)

    A module corresponds to either a hardware element on the chip (e.g., the clock or the LED), or to a user-defined software module

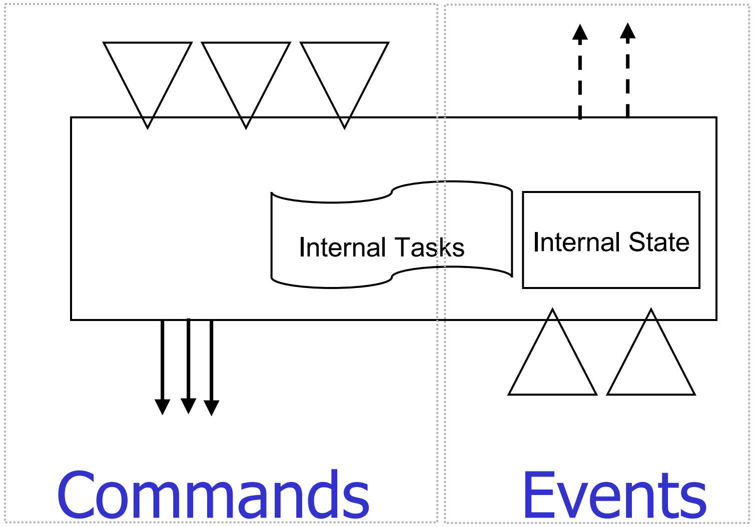
    Modules implement and use *interfaces*
  - Or a *configuration*, a set of other components *wired* together by specifying the unimplemented methods
- A complete NesC application then consists of one top level configuration

A Complete TinyOS Application



TinyOS component model

- Component specifies:



- Component invocation is event driven, arising from hardware events
- Static allocation only avoids run-time overhead
- Scheduling: dynamic, hard (or soft) real-time
- Explicit interfaces accommodate different applications

Steps in writing and installing your NesC app

(applies to MICA Mote)

- On your PC
  - Write NesC program
  - Compile to an executable for the mote
  - Plug the mote into the parallel port through a connector board
  - Install the program
- On the mote
  - Turn the mote on, and it's already running your application

TinyOS Facts

- Software Footprint 3.4 KB
- Power Consumption on Rene Platform
  Transmission Cost: 1 µJ/bit
  Inactive State: 5 µA
  Peak Load: 20 mA
- Concurrency support: at peak load CPU is asleep 50% of time
- Events propagate through stack <40 µS

Energy – a critical resource

- Power saving modes:
  - MICA: active, idle, sleep
- Tremendous variance in energy supply and demand
  - Sources: batteries, solar, vibration, AC
  - Requirements: long term deployment v. short term deployment, bandwidth intensiveness
  - 1 year on 2xAA batteries => 200 uA average current

Energy – a critical resource

Component	*Rate*	*Startup time*	*Current consumption*
CPU Active	4 MHz	N/A	4.6 mA
CPU Idle	4 MHz	1 us	2.4 mA
CPU Suspend	32 kHz	4 ms	10 uA
Radio Transmit	40 kHz	30 ms	12 mA
Radio Receive	40 kHz	30 ms	3.6 mA
Photo	2000 Hz	10 ms	1.235 mA
I2C Temp	2 Hz	500 ms	0.150 mA
Pressure	10 Hz	500 ms	0.010 mA
Press Temp	10 Hz	500 ms	0.010 mA
Humidity	500 Hz	500 ms	0.775 mA
Thermopile	2000 Hz	200 ms	0.170 mA
Thermistor	2000 Hz	10 ms	0.126 mA

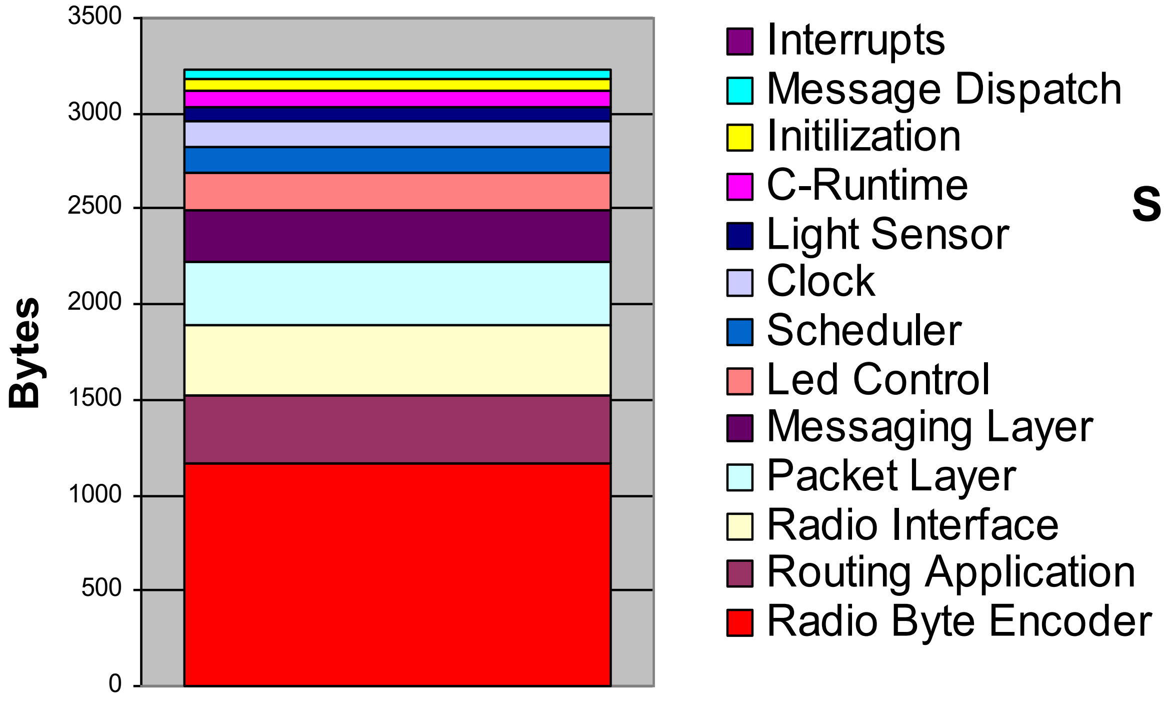
Which consumes the most power?

TinyOS: More Performance Numbers

- Byte copy – 8 cycles, 2 microsecond
- Post Event – 10 cycles
- Context Switch – 51 cycles
- Interrupt – h/w: 9 cycles, s/w: 71 cycles

TinyOS: Size

Code size for ad hoc networking application



Scheduler:	**144 Bytes code**
Totals:	**3430 Bytes code**
	226 Bytes data

TinyOS: Summary

Matches both

- Hardware requirements
  - power conservation, size
- Application requirements
  - diversity (through modularity), event-driven, real time

Discussion

System Robustness

- @ Individual sensor-node OS level:
  - Small, therefore fewer bugs in code
  - TinyOS: efficient network interfaces and power conservation
  - Importance? Failure of a few sensor nodes can be made up by the distributed protocol
- @ Application-level ?
  - Need: Designer to know that sensor-node system is flaky
- @ Level of Protocols?
  - Need for fault-tolerant protocols
    - Nodes can fail due to deployment/fab; communication medium lossy

    e.g., ad-hoc routing to base station:

    - TinyOS' s *Spanning Tree* Routing: simple but will partition on failures
    - DAG approach - more robust, but more expensive maintenance
  - Application-specific, or generic but tailorable to application ?

Scalability

@ OS level ?

TinyOS:

- Modularized and generic interfaces admit a variety of applications
- Correct direction for future technology
  - Growth rates: data > storage > CPU > communication > batteries
- Move functionality from base station into sensor nodes (*In-network processing*)
- In sensor nodes, move functionality from s/w to h/w

@ Application-level ?

- Need: Applications written with scalability in mind
- Need: Application-generic scalability strategies/paradigms

@ Level of protocols?

- Need: protocols that scale well with thousands of nodes
- In-network processing

Etcetera

- Option: ASICs versus generic-sensors
  - Performance vs. applicability vs money
  - Systems for sets of applications with common characteristics
- Event-driven model to the extreme:Asynchronous VLSI
- Need: Self-sufficient sensor networks
  - In-network processing, management, monitoring, and healing
- Need: Scheduling
  - Across networked nodes
  - Mix of real-time tasks and normal tasks
- Need: Security, and Privacy
- Need: Protocols for anonymous sensor nodes
  - E.g., Directed Diffusion protocol

Summary: Distributed Protocols for Sensor Systems…

…should match with both

- Hardware (e.g., energy use, small memory footprint, fault-tolerance, scalability)
- Application requirements (e.g., generic, scalability, fault-tolerance)

Other Projects

- Berkeley
  - TOSSIM (+TinyViz)
    - TinyOS simulator (+ visualization GUI)
  - TinyDB
    - Querying a sensor net like a database
  - Maté, Trickle
    - Virtual machine for TinyOS motes, code propagation in sensor networks for automatic reprogramming, like an active network.
  - CITRIS
- Several projects in other universities too
  - UI, UCLA: networked vehicle testbed

Looking Forward

- February 12 onwards: 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...
- Scribes: Split work, look at instructions on webpage
  - If no scribe in your session, presenters are responsible for scribe tasks
- Presentation/scribe is worth 25% of your grade!
- Check website for instructions.
- Reviews: You have to submit an **online copy** (Piazza) **by noon (on day of class)**.
- Project Discussion meetings (Mandatory) – soon
  - Signup sheet on Piazza. Form groups + sign up **by next Monday**.

Backup Slides (FLP Impossibility of Consensus Proof)

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)

Consensus in a Synchronous System

Possible to achieve!

- For a system with at most *f* processes crashing
  - 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 $Values^0_i = \{\}$ ; $Values^1_i = \{v_i\}$
    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
    $d_i$ = minimum($Values^{f+1}_i$)
```

Why does the Algorithm Work?

- After $f+1$ rounds, all non-faulty processes have received the same set of Values. Why?
- 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

  → Else, p_i would have sent v to p_j in the last round

  → So, in the last round: a third process, p_k, must have sent v to p_i, but then 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.

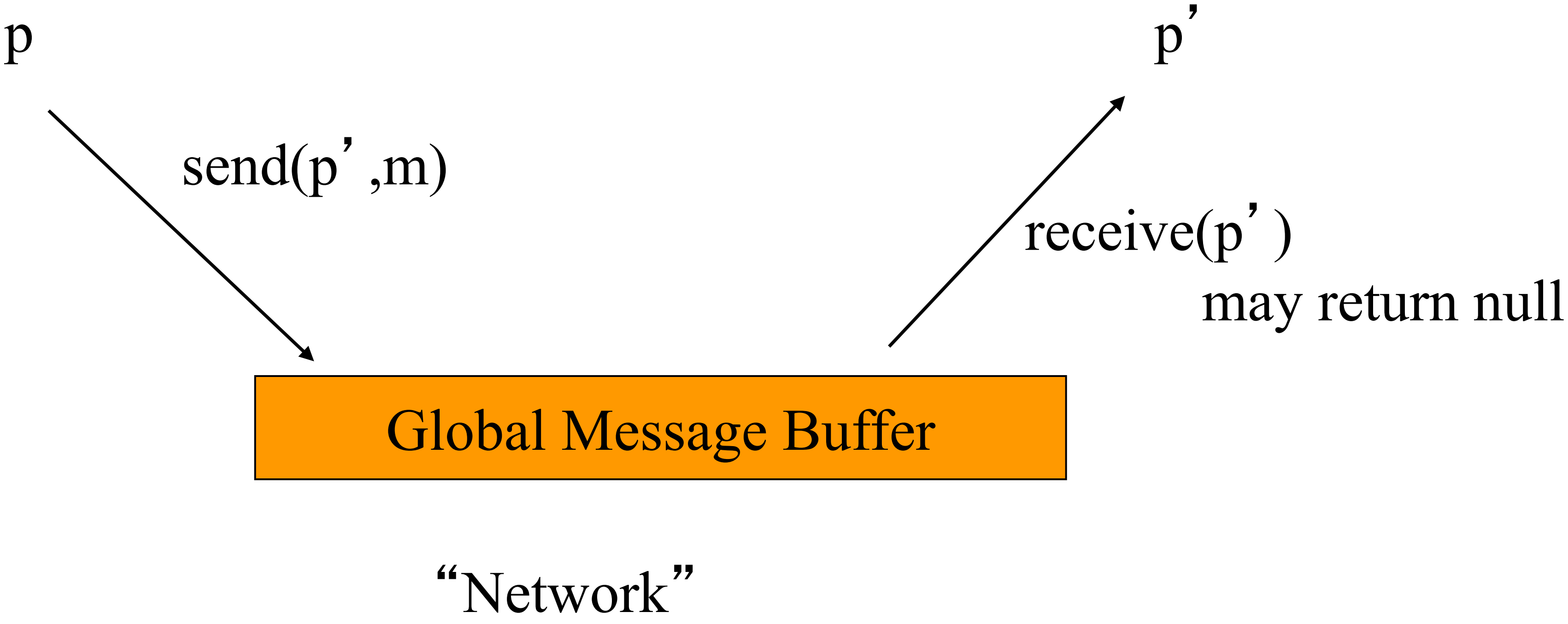
  → This means a total of $f+1$ crashes, while we have assumed at most f crashes can occur → contradiction.

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

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 (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
  - Why is this is ok?



p

p'

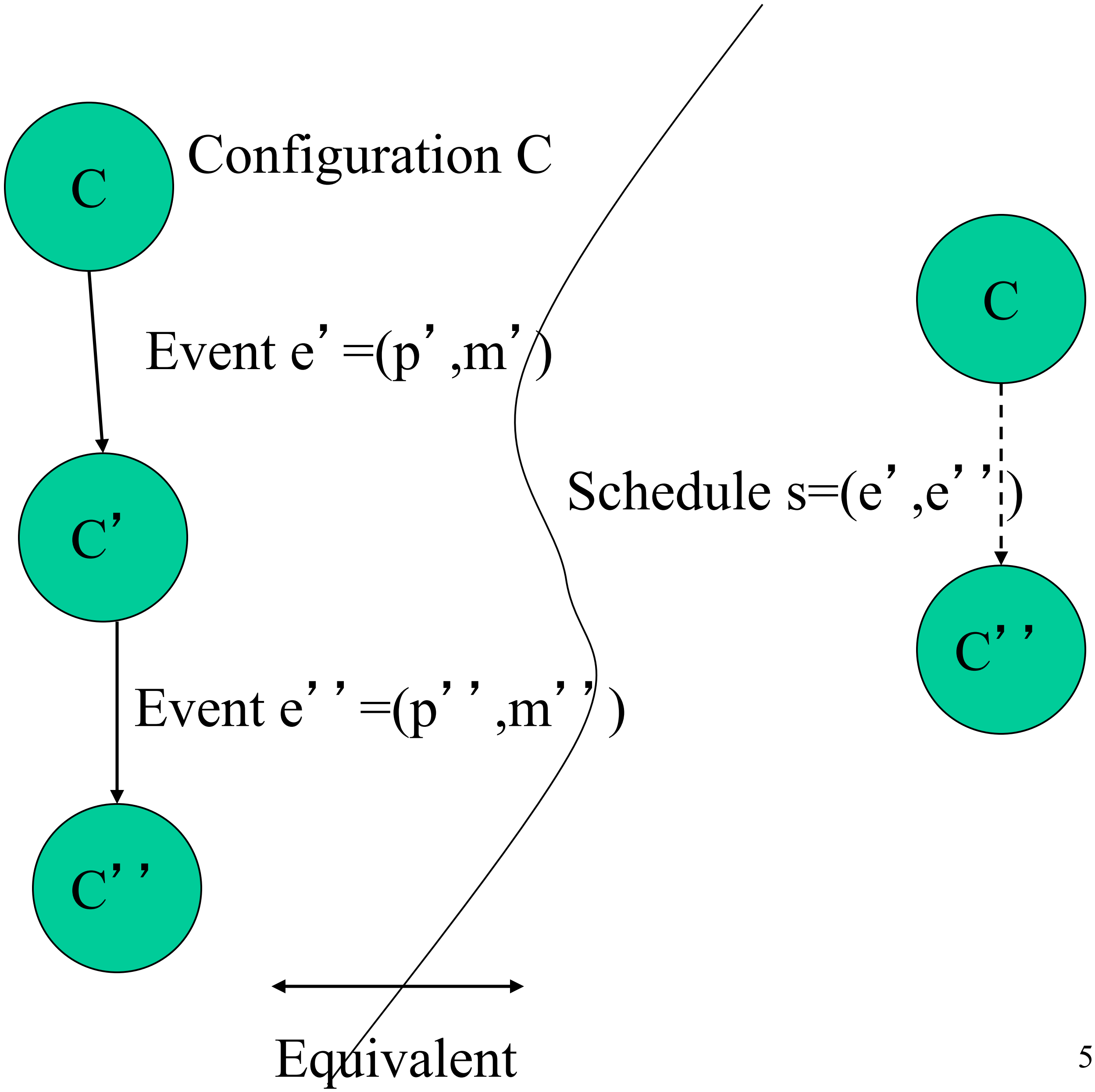
send(p' ,m)

receive(p')
may return null

Global Message Buffer

"Network"

- State of a process
- **Configuration**=global state. Collection of states, one for each process; alongside state of the global buffer.
- Each Event (<u>different</u> 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’ ’

C

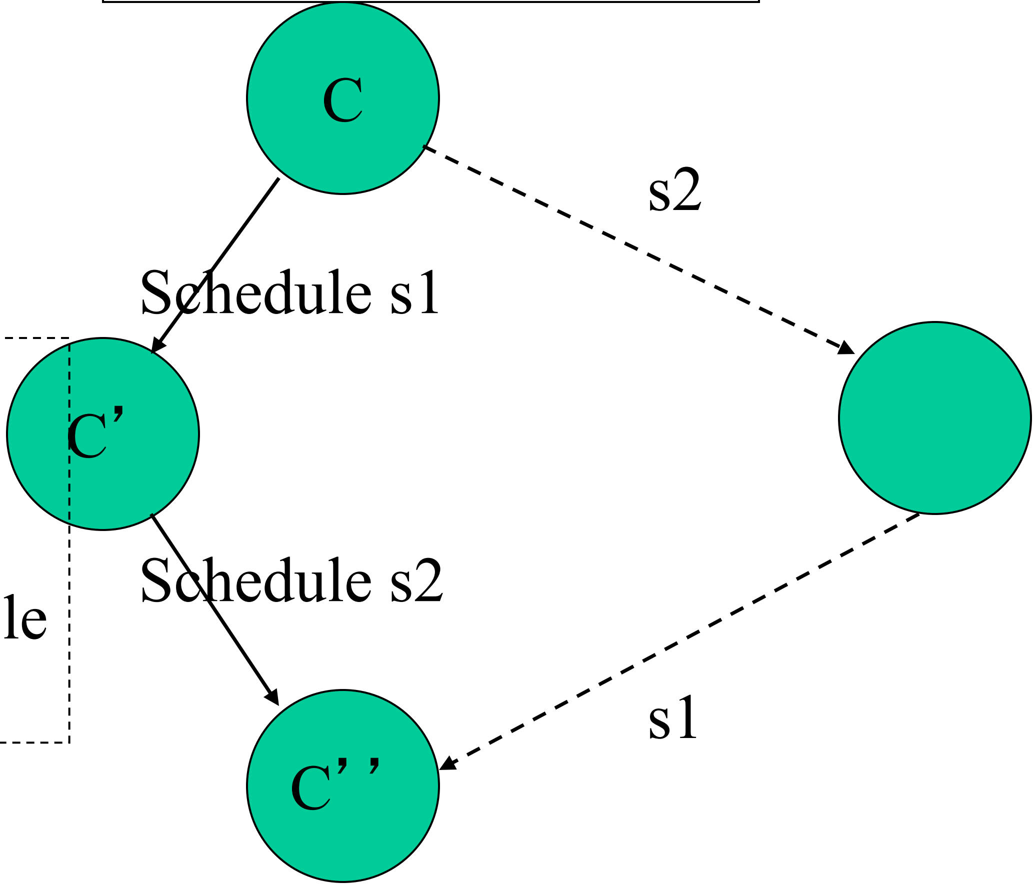
Schedule s=(e’ ,e’ ’)

C’ ’

Equivalent

Lemma 1

Disjoint schedules are commutative



s1 and s2 involve disjoint sets of receiving processes, and are each 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

- 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

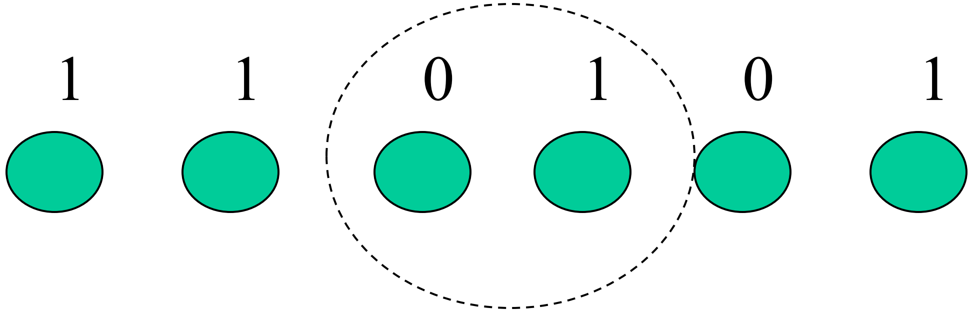
What the FLP Proof Shows

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 exactly one process.

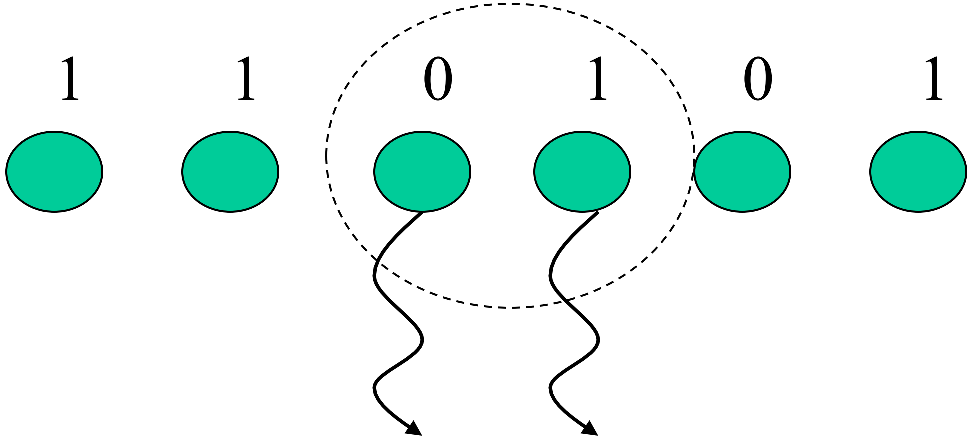


- There has to be some adjacent pair of 1-valent and 0-valent configs.

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 bivalent when there is such a failure

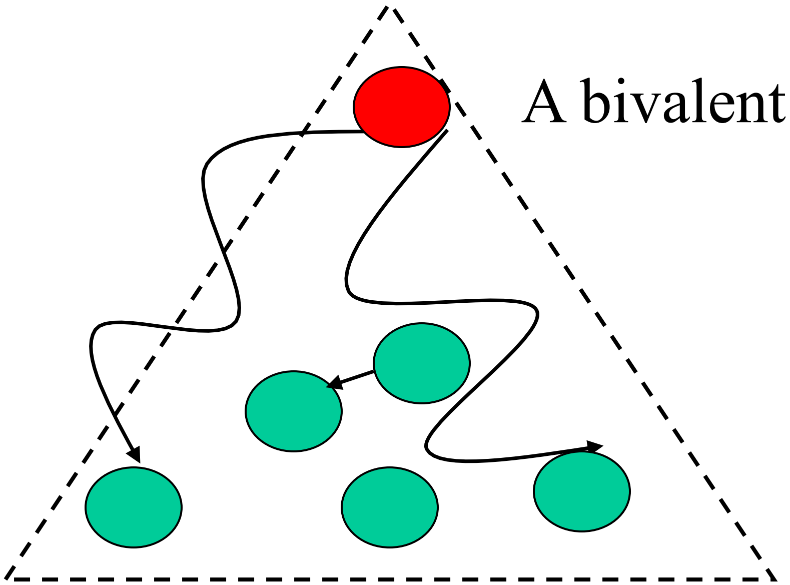
What we’ll Show

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

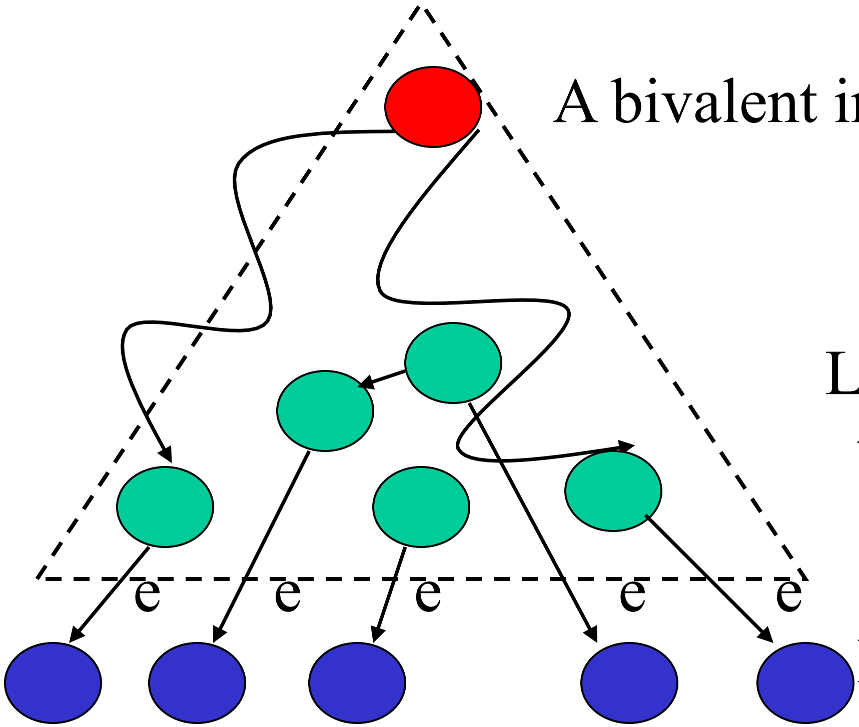


A bivalent initial config.

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



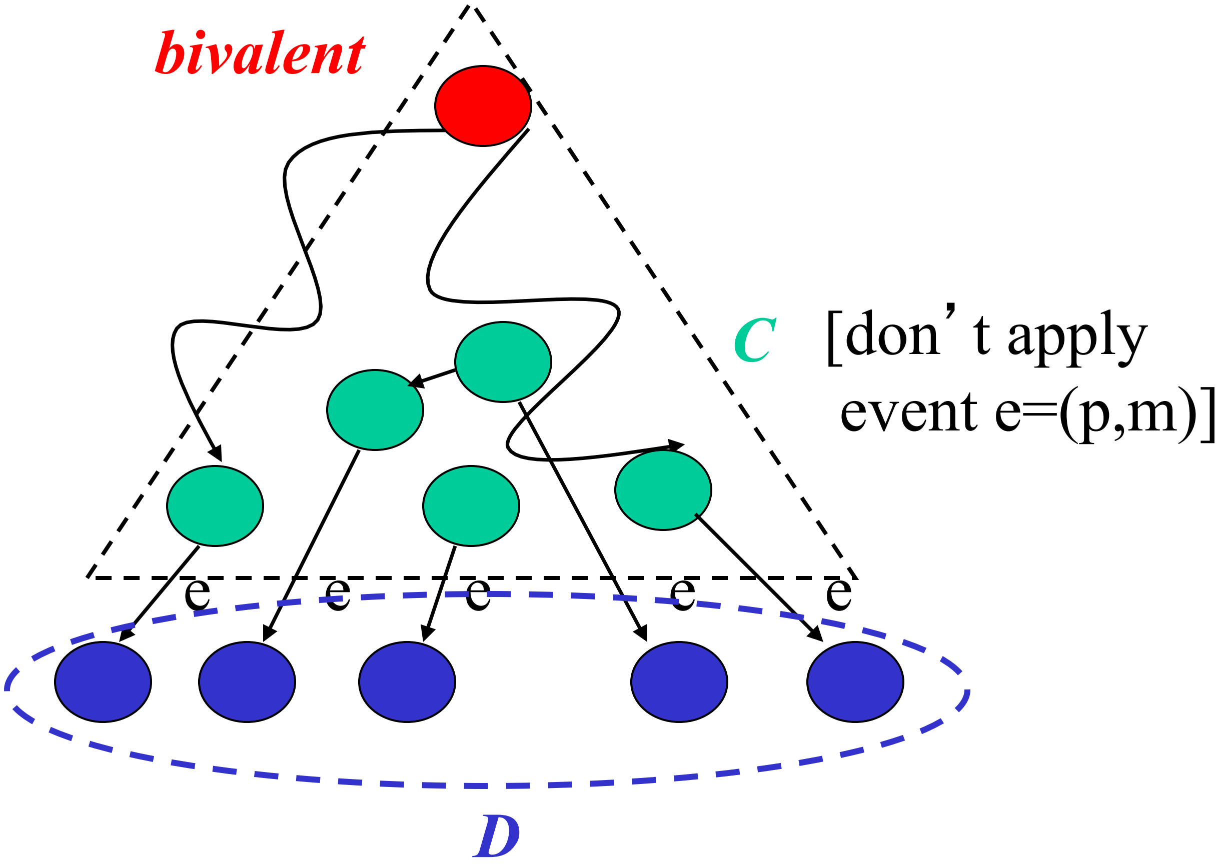
A bivalent initial config.

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



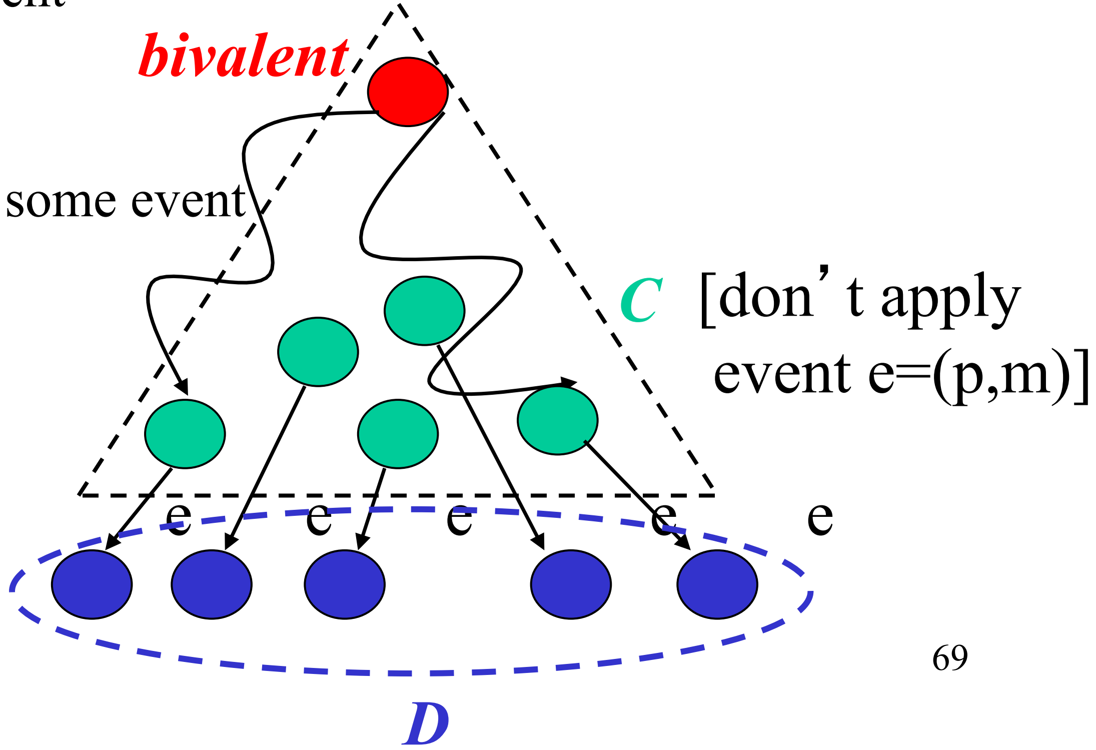
[don’t apply event e=(p,m)]

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)

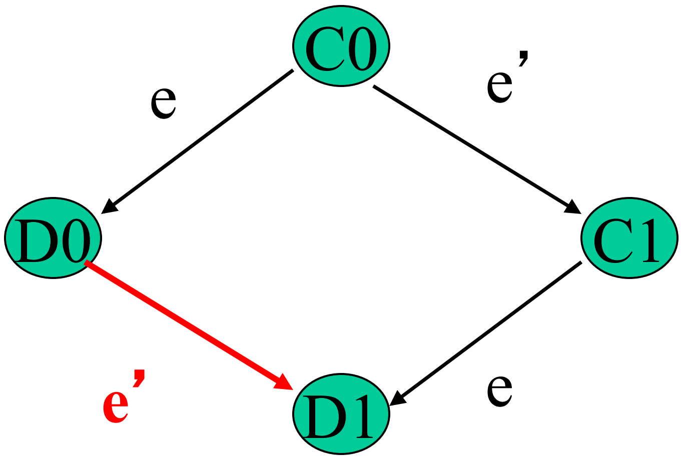
- There are states D0 and D1 in ***D***, and C0 and C1 in ***C*** such that
  - D0 is 0-valent, D1 is 1-valent
  - D0=C0 foll. by e=(p,m)
  - D1=C1 foll. by e=(p,m)
  - And C1 = C0 followed by some event e'=(p',m')

(why?)

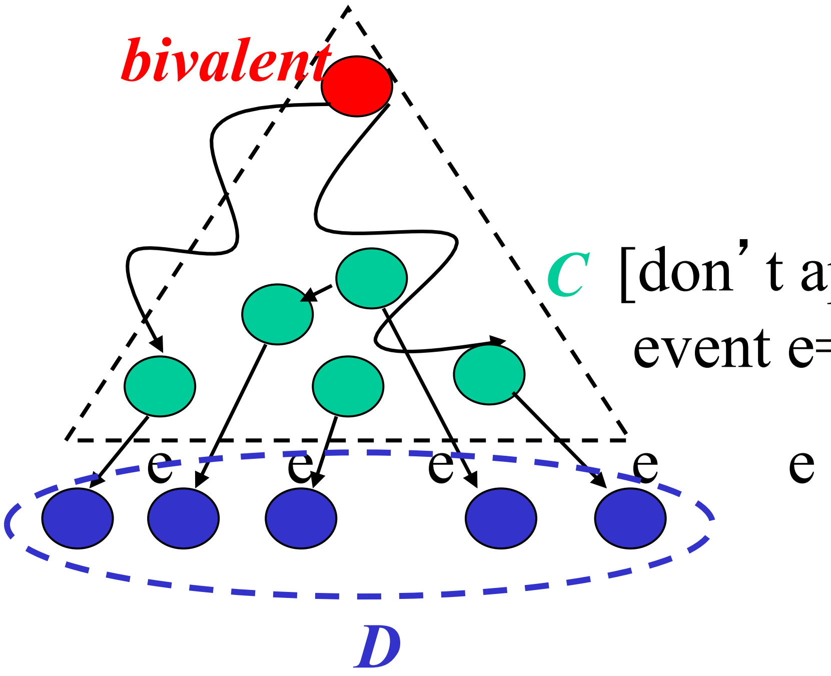


Proof. (contd.)

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



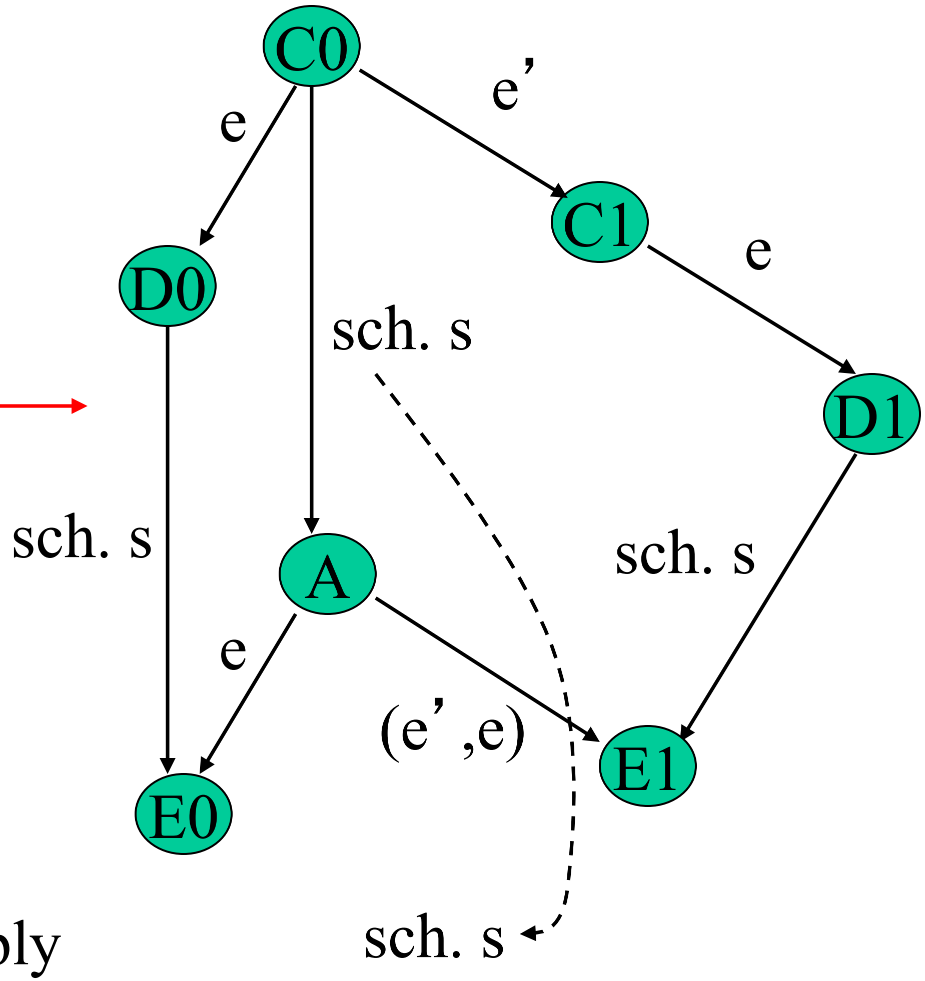
Why? (Lemma 1)
But D0 is then bivalent!



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

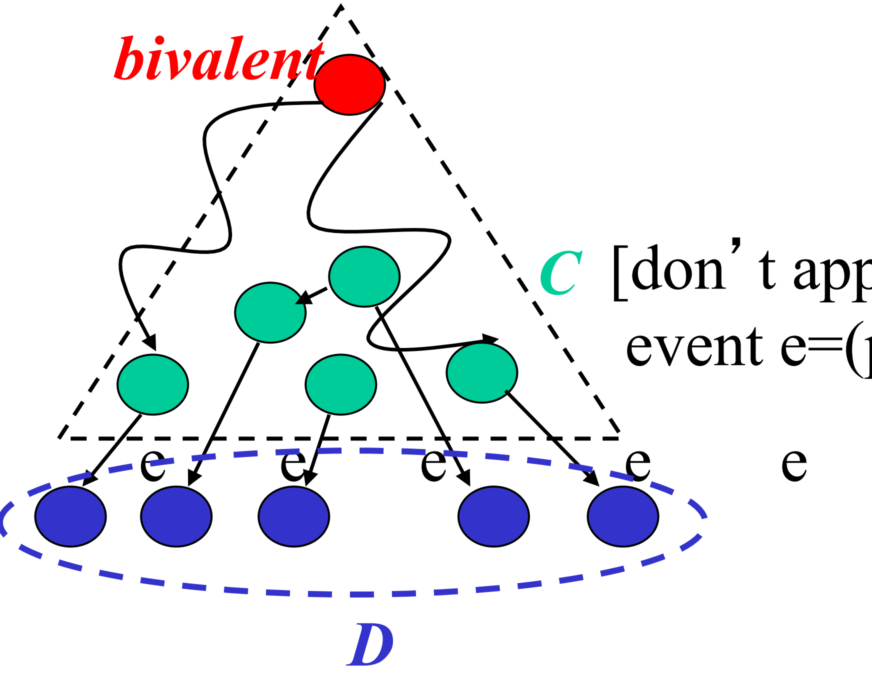
Proof. (contd.)

- 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 reach 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

Entr. Tidbits: Business Plan

- No one will give your company a thought without looking at your business plan.
- But that doesn’t mean the business plan has to be comprehensive (or fortune-telling).
- Hotmail kept a public business plan (Javasoft), and their hidden business plan was web-based email (which they revealed to only to super-interested VCs).
- TiVo’s original business plan was for network servers, and their hidden business plan was DVR.
- You’ve got to continuously adapt and change your business plan
  - Geschke (Adobe) received the following advice when several folks asked them for their Postscript product rather than their main product (which was printers): “You guys are nuts. Throw out your business plan. Your customers-or potential customers-are telling you what your business should be. The business plan was only used to get you the money. Why don't you rewrite a business plan that is focused just on providing what your customers want?”
  - Geschke also says why his competitors disappeared: “When we got our money for that original business plan, there were about half a dozen companies who had raised money to do something similar. Not the same, but similar. Fortunately, the other five all executed that business plan, and we didn't. And they all disappeared.”

Business Plan (contd.)

- (Bhatia, Hotmail) “A business plan is nothing more than your own communication to a person not sitting in front of you-an imaginary person who will read it. Try to answer every possible question that that person could raise. That's the description of a business plan, really. I didn't take any formal lessons. I just sat down and I wrote about the problem we were trying to solve, and in two paragraphs I described the World Wide Web and how it had grown and what its future potential could be. I said, this is the problem today that we are trying to address, this is how we hope to address it, with this idea. This is how we hope to monetize it and this is what page impressions are able to fetch you in the print world. If you translate it into the online world, this is how it will happen. And that's it, that was the core of our business plan. I wrote it in one night, and the next day I went to work looking really sleepy and tired. My boss said, "Another one of those days of late-night partying?" I'm like, "Yeah, something like that." He said, "Alright, you'll be productive only in the afternoon. Take the morning off." Little did he know that I was actually up all night writing a business plan, not partying.”

Business Plan (contd.)

- (Levchin, PayPal) “I think the hallmark of a really good entrepreneur is that you're not really going to build one specific company. The goal-at least the way I think about entrepreneurship-is you realize one day that you can't really work for anyone else. You have to start your own thing. It almost doesn't matter what that thing is. We had six different business plan changes, and then the last one was PayPal.”
- (Geschke, Adobe) “It didn't matter whether or not some guy at IBM thought it looked good. What mattered was someone at Random House or Time-Life or Ogilvy & Mather or someone like that appreciated it.”