Time and Global States

- Introduction
- Clocks, events and process states
- Synchronizing physical clocks
- Logical time and logical clocks
- Global states
- Distributed debugging

Time is an important issue in DS

- At what time particular event occurred.
  - Necessary to synchronize with an authoritative source of time
  - Need to measure accurately
    - E.g. auditing in e-commerce
- Maintaining the consistency of distributed data
- Timestamp the events at different nodes

Time is an important issue in DS

- Temporal ordering of events produced by concurrent processes
- Synchronization between senders and receivers of messages
- Coordination of joint activity
- Serialization of concurrent access for shared objects

Model of a distributed system

- **P**
  - A collection of N processes $\boldsymbol{p_i}$, i = 1,2, .. N
- $\boldsymbol{s_i}$
  - The state of p_i
  - E.g. variables
- **Actions of $\boldsymbol{p_i}$**
  - Operations that transform p_i's state
  - Send or receive message between p_j

Model of a distributed system

- ***e***
  - Event: occurrence of a single action
- $\rightarrow_i$ **relationship between events**
  - occur before in p_i , e.g. $e \rightarrow_i e`$
  - Total order of events in p_i
- ***history***$(p_i) = h_i$
  - $h_i = \langle e_i^0, e_i^1, e_i^2, \ldots \rangle$

Logical vs. physical clocks

Logical clock keeps track of event ordering

- **among related (causal) events**

Physical clocks keep time of day

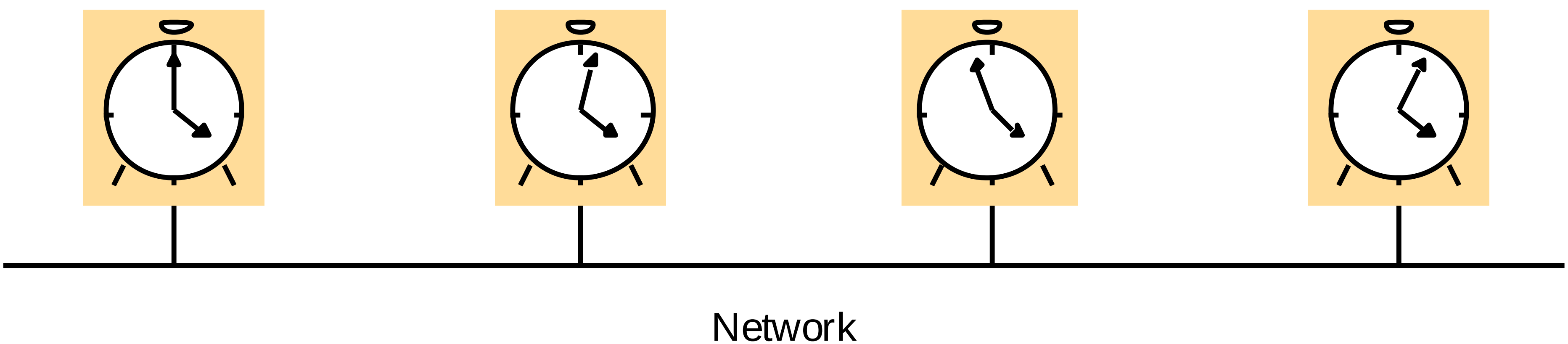
- **Consistent across systems**

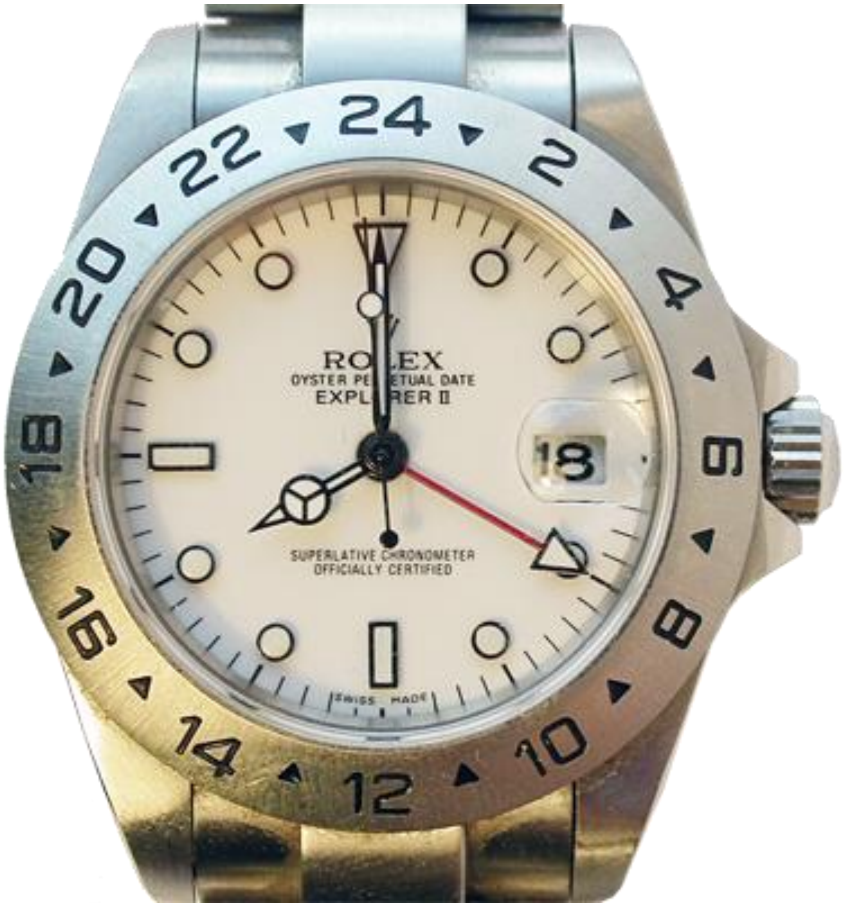
Clock in computer

- Timestamp: clock is used to assign date and time as a timestamp
- Physical Clock: A device that count oscillations occurring in a crystal at a definite frequency
- Hardware time: $H_i(t)$
  - The counts of oscillation since an original point
- Software time: $C_i(t) = \alpha H_i(t)+\beta$
  - *Timestamp* of an event

Clock skew and clock drift

- Clock drift
  - Crystal oscillate at different rate
  - Clock drift can not be avoided
  - Ordinary quartz clocks drift by about 1 sec in 11-12 days. (10^{-6} secs/sec). High precision quartz clocks drift rate is about 10^{-7} or 10^{-8} secs/sec
- Clock skew
  - The instantaneous difference between the readings of any two clocks





8:00:00

8:00:00

Sept 18, 2006

8:00:00

8:01:24

Skew = +84 seconds
+84 seconds/35 days
Drift = +2.4 sec/day

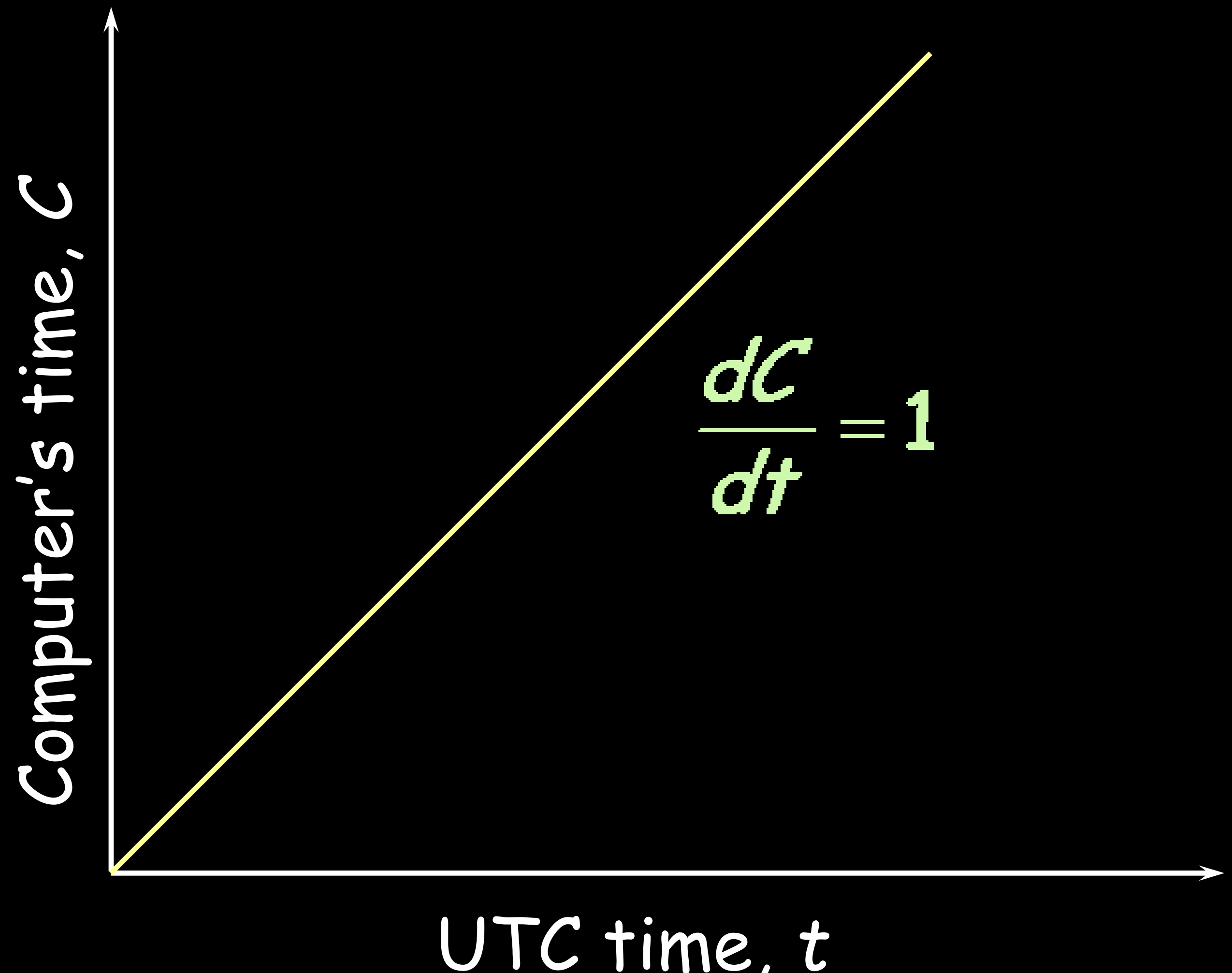
Oct 23, 2006

8:00:00

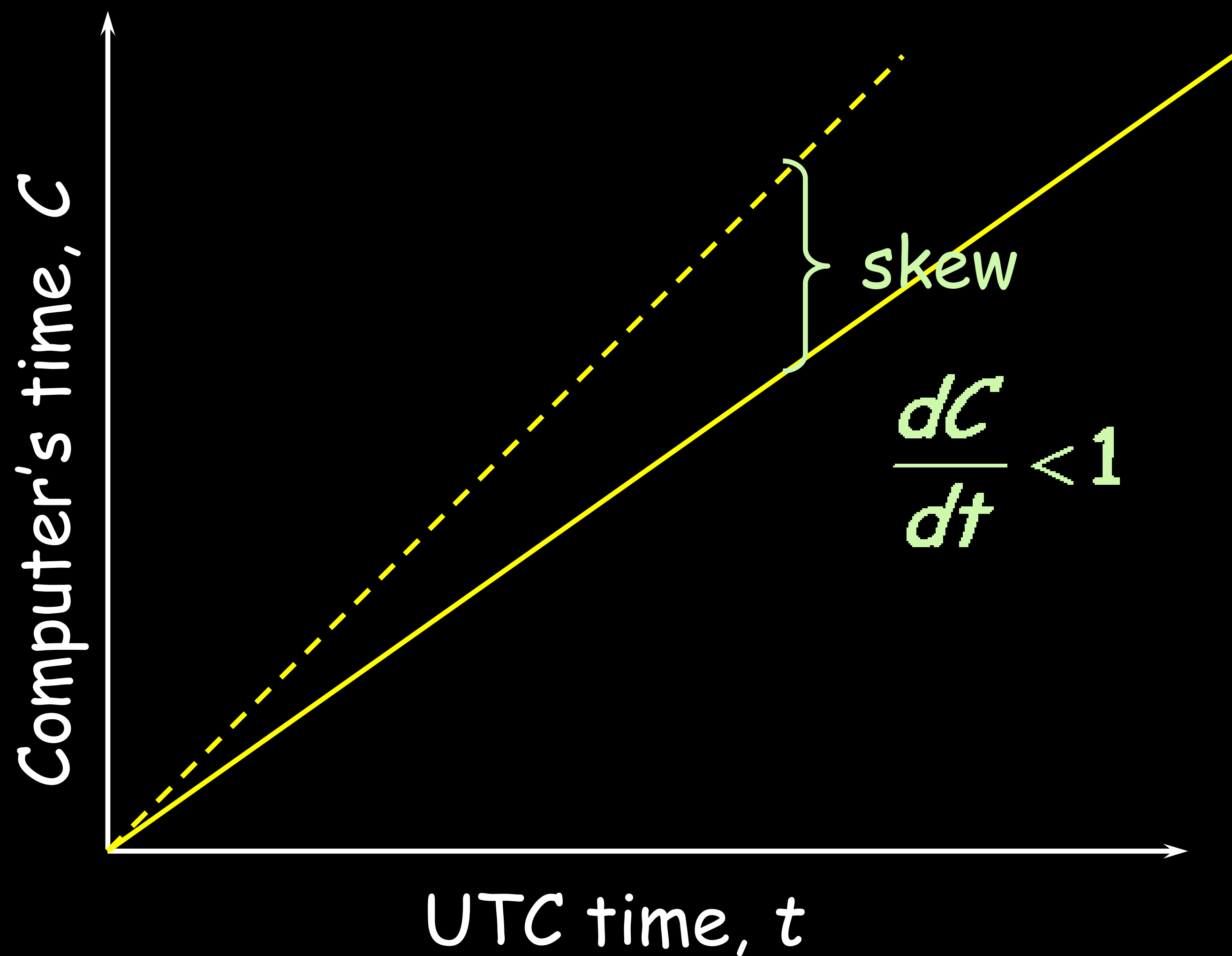
8:01:48

Skew = +108 seconds
+108 seconds/35 days
Drift = +3.1 sec/day

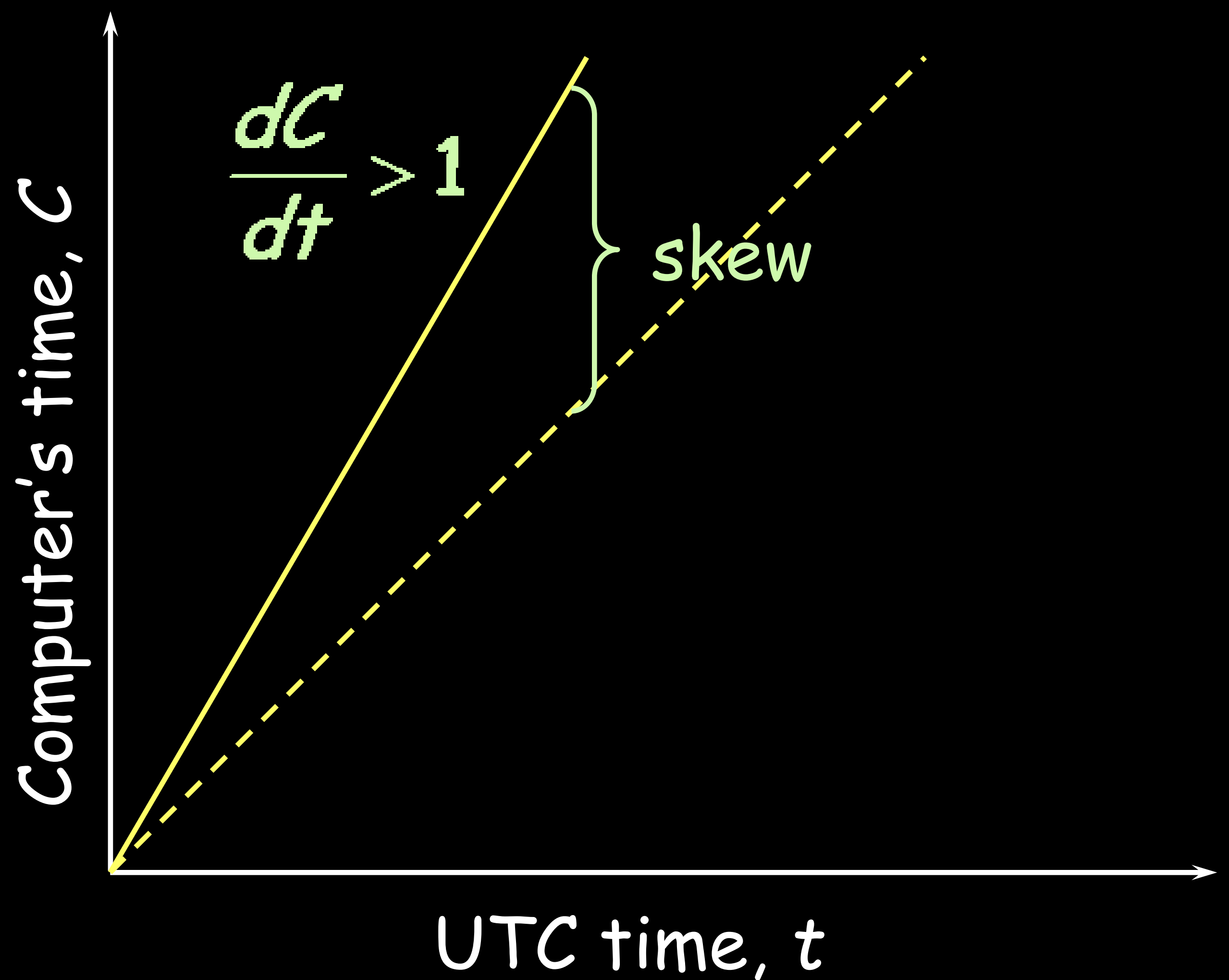
Perfect clock



Drift with slow clock



Drift with fast clock



Coordinated Universal Time (UTC)

- **International Atomic Time** is based on very accurate physical clocks (drift rate 10^{-13} secs/sec)

- **UTC** is an international standard for time keeping
  - It is based on atomic time, but occasionally adjusted to astronomical time
  - It is broadcast from radio stations on land and satellite (e.g. GPS)

- Computers with receivers can synchronize their clocks with these timing signals
  - Signals from land-based stations are accurate to about 0.1-10 millisecond
  - Signals from GPS are accurate to about 1 microsecond

Time and Global States

- Introduction
- Clocks, events and process states
- Synchronizing physical clocks
- Logical time and logical clocks
- Global states
- Distributed debugging
- Summary

External & Internal synchronization

- **External synchronization**
  - Clocks C_i of a set of N computers are synchronized with an external authoritative time source S if:
  - $\boldsymbol{|S(t) - C_i(t)| < D}$ for $i = 1, 2, \ldots N$ for all $\boldsymbol{t}$ in an interval of real time
  - The clocks C_i are **accurate** to within the bound D.
- **Internal synchronization**
  - The clocks C_i of a set of N computers are synchronized with one another if:
  - $\boldsymbol{|C_i(t) - C_j(t)| < D}$ for $i,j = 1, 2, \ldots N$ for all $\boldsymbol{t}$ in an interval of real time
  - The clocks C_i **agree** within the bound D.

- Internally synchronized clocks are not necessarily externally synchronized, as they may **drift collectively**
  - if the set of processes P is synchronized externally within bound D, it is also internally synchronized within bound $2D$

General synchronization issues

- Correctness of a hardware clock *H*
  - A bounded drift rate ρ, e.g. 10^{-6} seconds/second, t and t' are real time
  - $(1 - \rho)(t' - t) <= H(t') - H(t) <= (1 + \rho)(t' - t)$

$$(1 - \rho) <= \frac{H(t') - H(t)}{(t' - t)} <= (1 + \rho)$$

- Correctness of a software clock
  - *Monotonicity*: $t' > t \Rightarrow C(t') > C(t)$
  - Set clock back
    - Errors in the *make* process
  - Change the clock rate

General synchronization issues (2)

- Clock failures

a faulty clock is one that does not obey its correctness condition

  - ***crash failure*** - a clock stops ticking
  - ***arbitrary* failure** - any other failure e.g. jumps in time, Y2K

Synchronization Methods

- Synchronous Systems
  - Simpler, relies on known time bounds on system actions
- Asynchronous Systems:
  - Intranets:
    - Cristian's Algorithm
    - Berkeley Algorithm
  - Internet:
    - The Network Time Protocol

Synchronization in a synchronous system

- Condition in DS for Synchronous communication:
  - Time execution of each process has upper and lower bounds
  - Each message transmitted over a channel is received in a known bounded time
  - Each process has a local clock whose drift rate from real time has known bound

Synchronization in a synchronous system

- Protocol: p1 process sends its local time t to another process p2
  - Sender: send $M(t)$
  - Receiver: set time to $t + T_{trans}$
- Bounds are known in synchronous system
  - ❖ min < T_{trans} < max
- So, set T_{trans} = (min+max) / 2
  - Receiver's clock = t + (min+max) / 2

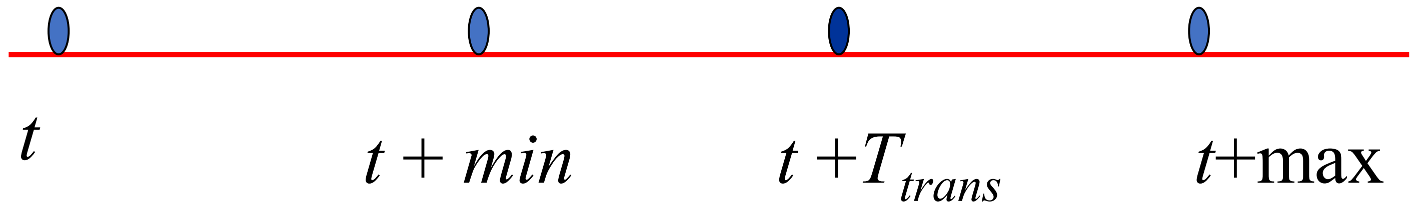
Synchronization in a synchronous system(2)

- Clock skew between sender and receiver

  uncertainty **u = max-min**.

  Set clock to **t + (max - min)/2** then

  skew ≤ u/2

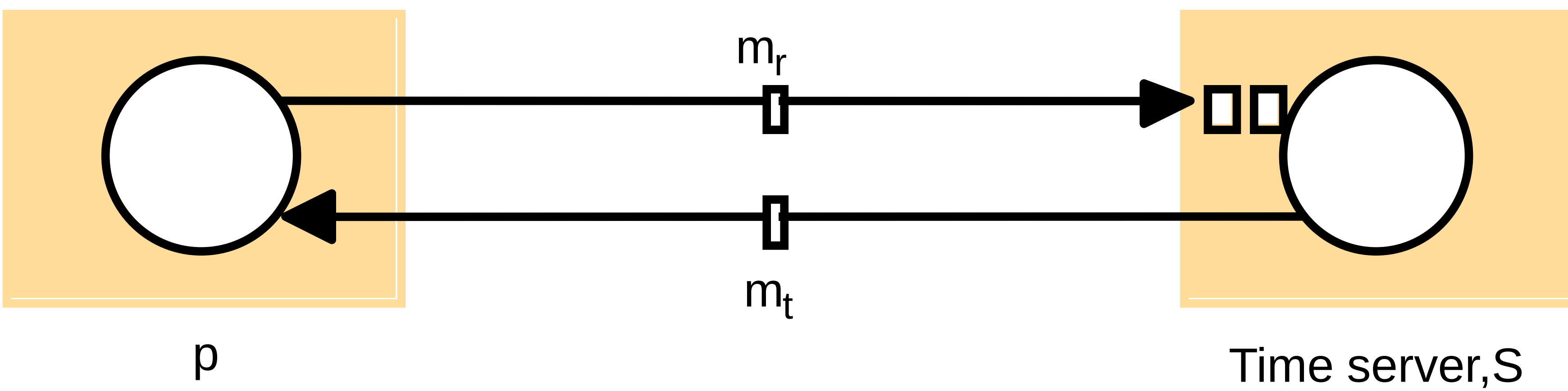


Synchronization Methods

- Synchronous Systems
  - Simpler, relies on known time bounds on system actions
- Asynchronous Systems:
  - Intranets:
    - Cristian’s Algorithm
    - Berkeley Algorithm
  - Internet:
    - The Network Time Protocol

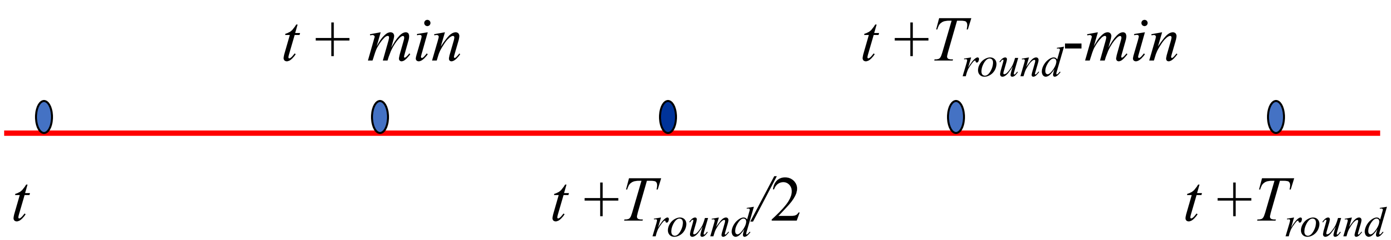
Cristian's method of synchronizing clocks

- Use of Time Server.
- Applying circumstance
  - C/S Round-trip time is short compared with the required accuracy
- Protocol
  - m_r, $m_t(t)$, T_{round}
  - Estimated time: t in $m_t + T_{round}/2$



Cristian's method of synchronizing clocks (2)

- Accuracy analysis
  - If the minimum delay of a message transmission is *min*, then accuracy: $\pm(T_{round}/2 - min)$



The Berkeley algorithms

- Internal synchronization

1. The *master* polls the *slaves*' clocks
2. The *master* estimates the *slaves*' clocks by round-trip time
   - Similar to Christian's algorithm
3. The *master* averages the *slaves*' clock values
   - Cancel out the individual clock's tendencies to run fast or slow

The Berkeley algorithms (2)

4. The master sends back to the slaves the amount that the slaves' clocks should adjust by
   - Positive or negative value
   - Avoid further uncertainty due to the message transmission time

- Slave adjust its clock
- Fault tolerant average to check faulty clocks: difference of two clocks not more than specified amount.

Design aims of Network Time Protocol

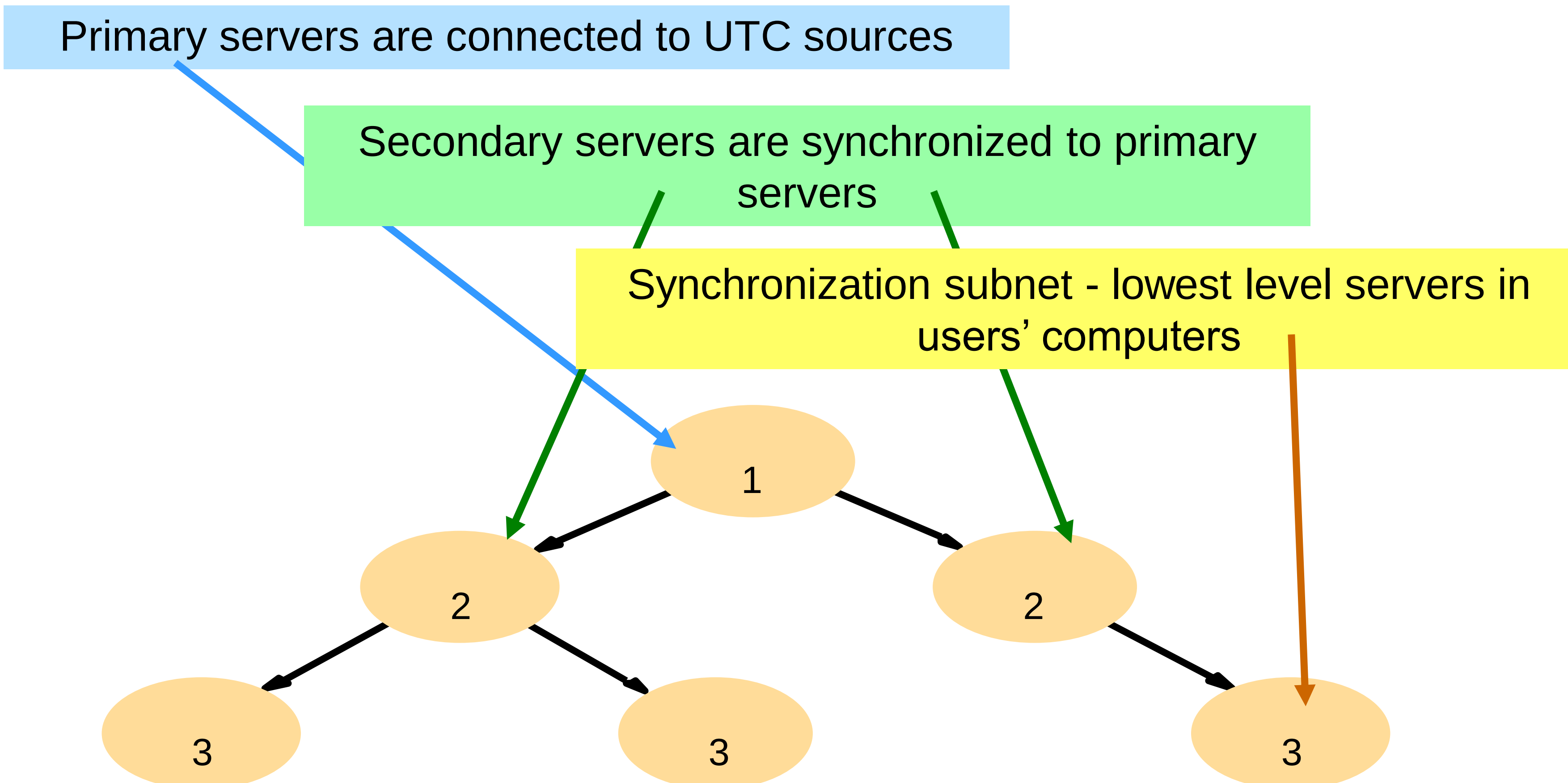
- External synchronization
  - Enable clients across the Internet to be synchronized accurately to UTC
- Reliability
  - Can survive lengthy losses of connectivity
    - Redundant server & redundant path between servers

Design aims of Network Time Protocol (2)

- Scalability
  - Enable clients to resynchronize sufficiently frequently to offset the rates of drift found in most computers
- Security
  - Protect against interference with the time service

Network Time Protocol Architecture

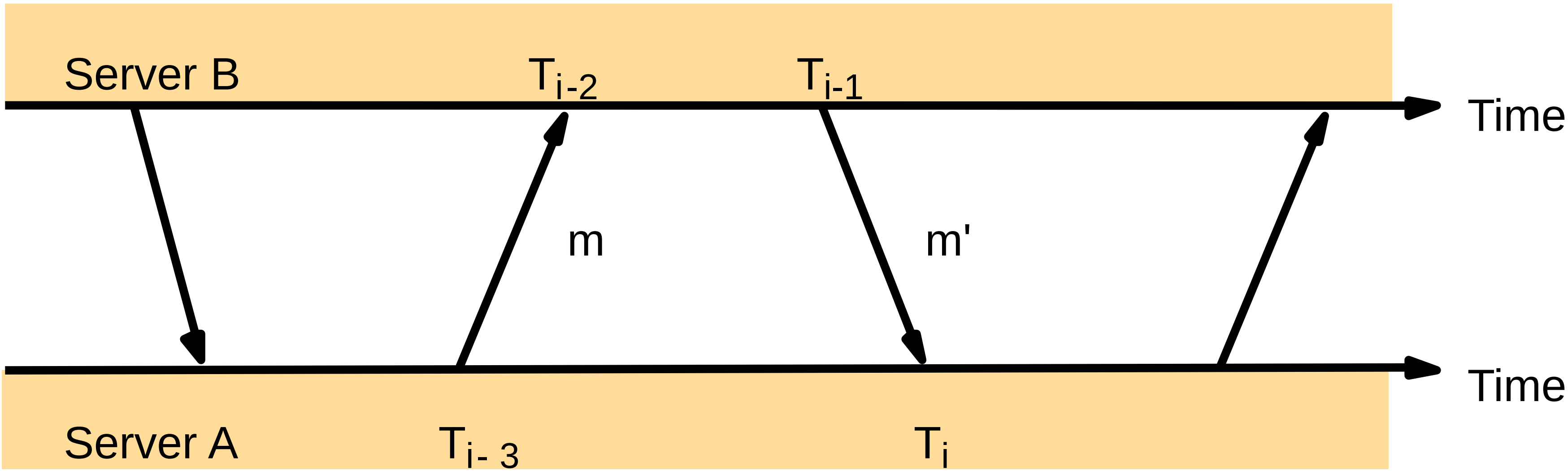
- A time service for the Internet - synchronizes clients to UTC



Synchronization measures

- Multicast mode
  - Intend for use on a high-speed LAN
  - Assuming a small delay
  - Low accuracy but efficient
- Procedure-call mode
  - Similar to Christian’s
  - Higher accuracy than multicast
- Symmetric mode
  - The highest accuracy

Symmetric mode synchronization



Assuming

t, t': actual transmission time of m, m'; *o*: actual B's clock skew relative to A

We have

$$T_{i-2} = T_{i-3} + t + o \,,\; T_i = T_{i-1} + t' - o$$

Symmetric mode synchronization (2)

$$T_{i-2} = T_{i-3} + t + o \ , \ T_i = T_{i-1} + t' - o$$

Then

addition :

$$d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}$$

subtraction :

$$o = (T_{i-2} - T_{i-3} + T_{i-1} - T_i + t' - t)/2$$

where $o_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2$

we have $o = o_i + (t' - t)/2$

Symmetric mode synchronization (2)

we have $\boldsymbol{o = o_i + (t' - t)/2}$

Accuracy analysis

Due t, $t' >= 0$, then

$$o_i - (t' + t)/2 <= o <= o_i + (t' + t)/2$$

Then

$$o_i - d_i/2 <= o <= o_i + d_i/2$$

o_i is the estimated time

d_i is the measure of the accuracy

Symmetric mode sync. implementation

- NTP servers retain 8 most recent pairs $<o_i, d_i>$
- The value o_i of that corresponds to the minimum value d_i is chosen to estimate o
- A NTP server exchanges with several peers in addition to with parent
  - Peers with lower stratum numbers are favored
  - Peers with the lowest synchronization dispersion are favored

Time and Global States

- Introduction
- Clocks,events and process states
- Synchronizing physical clocks
- Logical time and logical clocks
- Global states
- Distributed debugging
- Summary

Happen-before relation

- $\rightarrow$
  - HB1: process p_i: $e \rightarrow_i e`$, then $e \rightarrow e`$
  - HB2: For any message m,

    send(m) $\rightarrow$receive(m)
  - HB3: IF e, e`and e`` are events such that e $\rightarrow$e` and e` $\rightarrow$e``, then e $\rightarrow$e``
  - *Causal ordering*

Happen-before relation

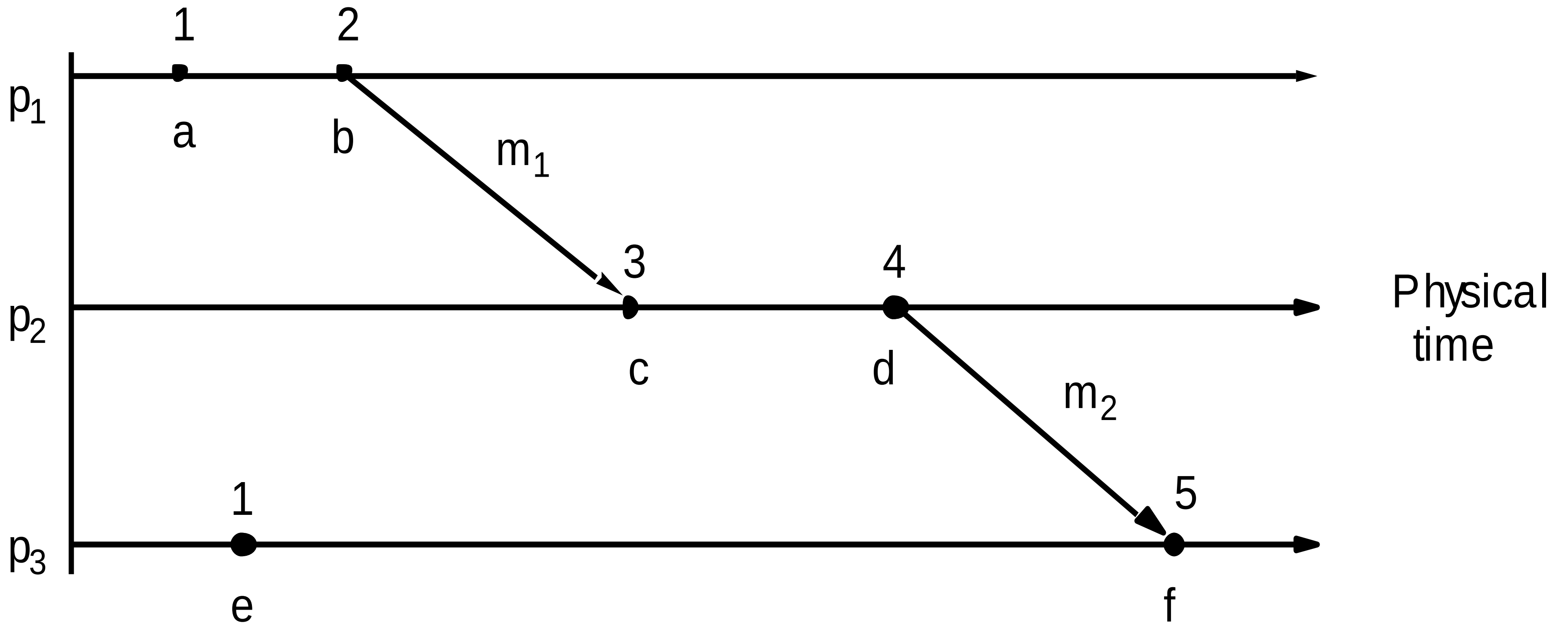
- Example
  - a || e
- Shortcomings
  - Not suitable to processes collaboration that does not involve messages transmission
  - Capture *potential causal ordering*

Lamport timestamps algorithm

- LC1
  - L_i is incremented before each event is issued at process p_i : $L_i := L_i + 1$
- LC2:
  - (a) When a process p_i sends a message m, it piggybacks on m the value $t = L_i$
  - (b) On receiving (m,t), a process P_j computes $L_j := max(L_j, t)$ and then applies LC1 before timestamping the event receive(m)

Lamport timestamps algorithm (2)

- $e \rightarrow e' \Rightarrow L(e) < L(e')$
- $L(e) < L(e') \Rightarrow e \rightarrow e'$ or $e || e'$



Totally ordered logical clocks

Assumption

T_i : local timestamp of e that is an event occurring at p_i

T_j : local timestamp of $e`$ that is an event occurring at p_j

Define the timestamps of e, $e`$ are (T_i, i), (T_j, j)

Define <

$(T_i, i) < (T_j, j)$ if $T_i < T_j$, or $T_i = T_j$ *and* $i < j$

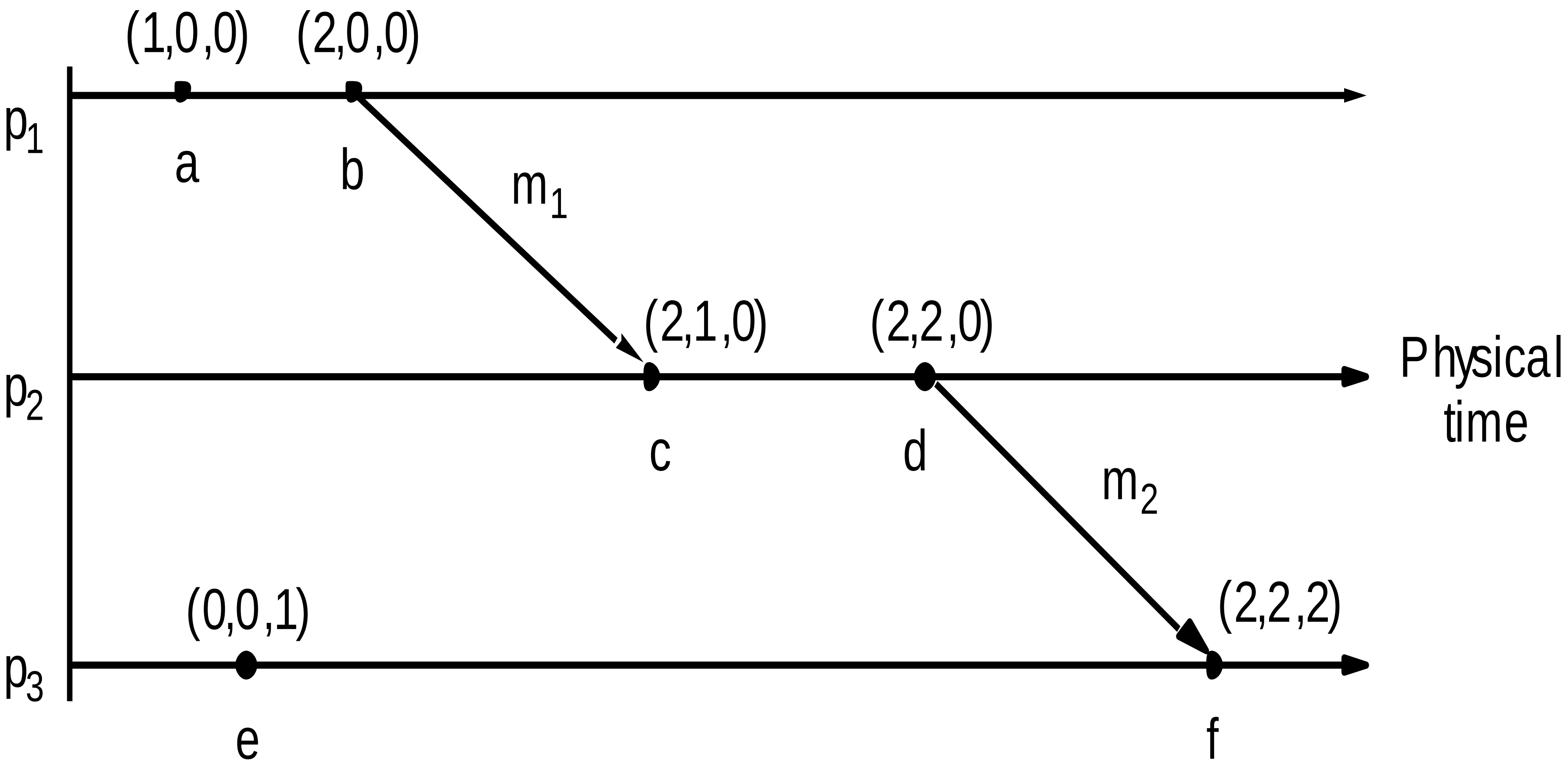
- Useful in some applications

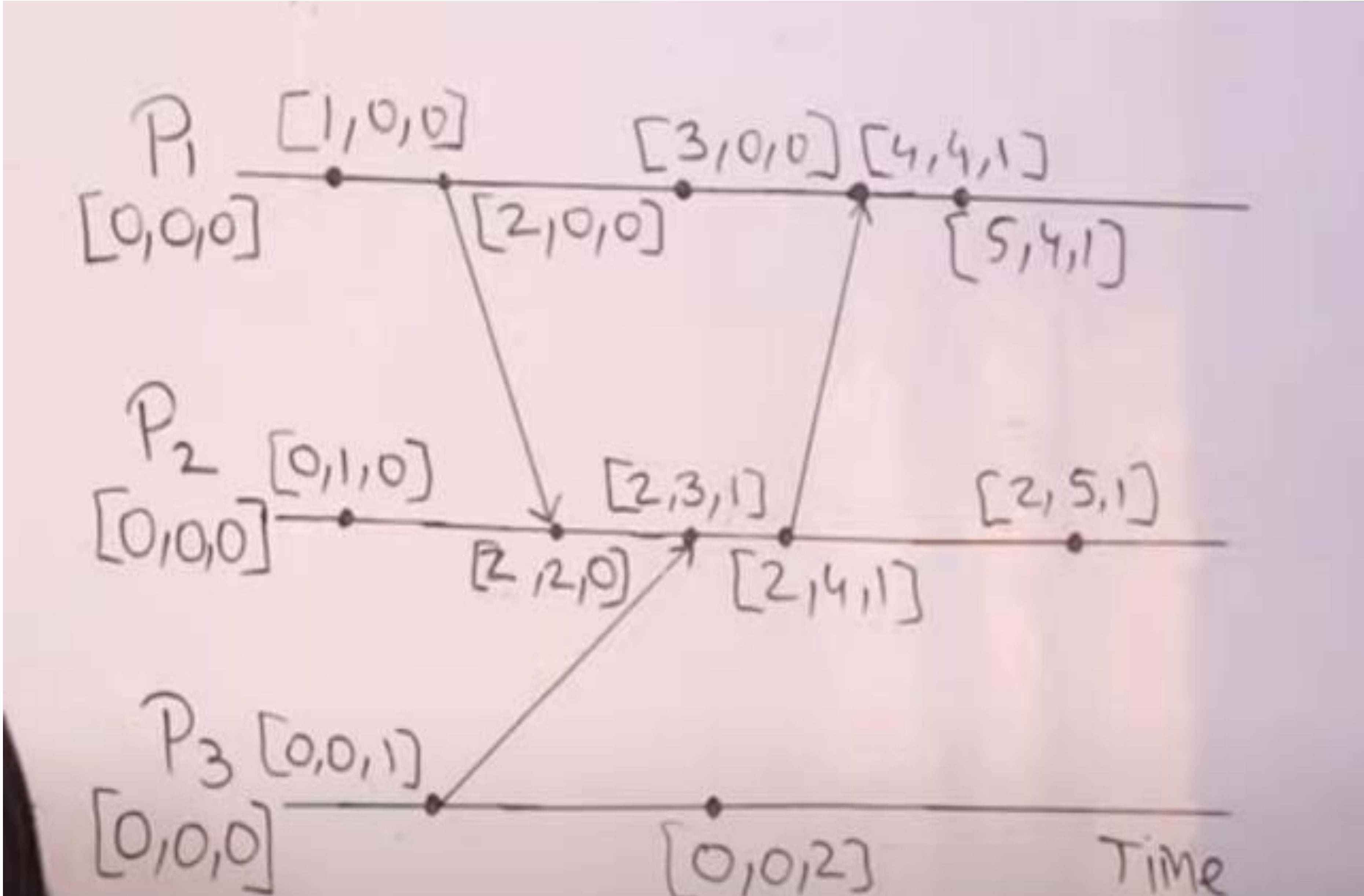
Vector Clocks - algorithm

Each process p_i keeps a vector clock V_i

- VC1: Initially, $V_i[j]=0$, for $i, j = 1,2\ldots, N$
- VC2: Just before p_i timestamps an event, it sets $V_i[i] := V_i[i] +1$
- VC3: p_i includes the value $t= V_i$ in every message it sends
- VC4: When p_i receives a timestamp t in a message, it sets $V_i[j] := max(V_i[j], t[j])$, for $j=1,2\ldots,N$

Vector Clocks - example





P1
[0,0,0]
[1,0,0]
[2,0,0]
[3,0,0]
[4,4,1]
[5,4,1]
P2
[0,0,0]
[0,1,0]
[2,2,0]
[2,3,1]
[2,4,1]
[2,5,1]
P3
[0,0,0]
[0,0,1]
[0,0,2]
Time

Vector Clocks - significance

- Compare vector timestamps
  - $V = V`$ iff $V[j] = V`[j]$ for $j = 1,2\ldots, N$
  - $V <= V`$ iff $V`[j] <= V`[j]$ for $j = 1,2\ldots, N$
  - $V < V`$ iff $V`[j] < V`[j]$ for $j = 1,2\ldots, N$

Time and Global States

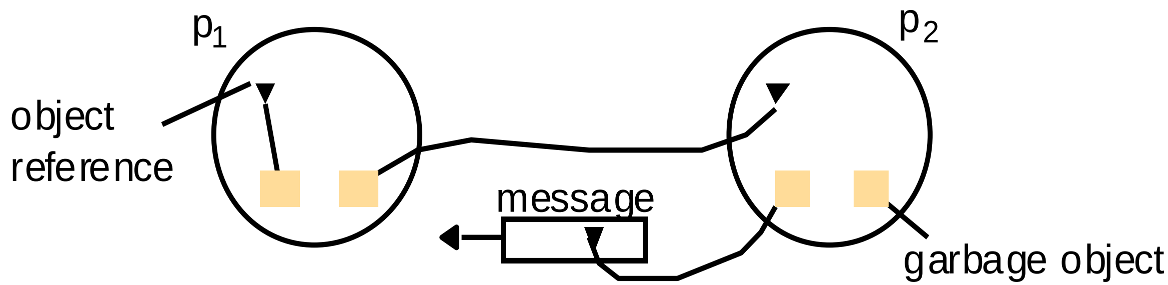
- Introduction
- Clocks,events and process states
- Synchronizing physical clocks
- Logical time and logical clocks
- Global states
- Distributed debugging
- Summary

Requirements of global states

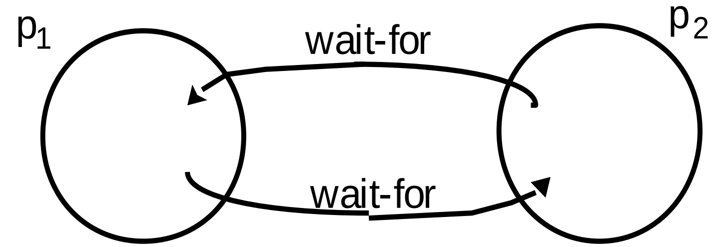
- Distributed garbage collection
  - Based on reference counting
  - Should include the state of communication channels
- Distributed deadlock detection
  - Look for “waits-for” relationship
- Distributed termination detection
  - Look for state in which all processes are passive
- Distributed debugging
  - Need collect values of distributed variables at the same time

Detecting global properties

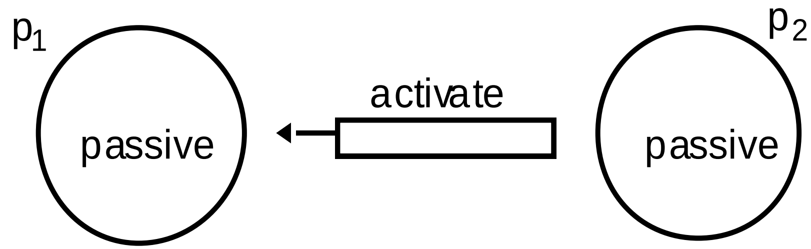
a. Garbage collection



b. Deadlock



c. Termination



Global states and consistent cuts

- The essential problem of Global states
  - Absence of global time

History of process p_i

$$h_i = <e_i^0, e_i^1, e_i^2 \dots>$$

Prefix of a process's history

$$h_i^k = <e_i^0, e_i^1 \dots e_i^k>$$

Global history of processes set £

$$H = h_1 \cup h_2 \cup \dots \cup h_N$$

The state of process p_i

s_i^k : the state before the kth event occurs

The state of channel

between the p_i and p_j when transmit the massage

A *cut* of a system execution

A subset of of its global history that is a union of prefixes of process histories.

$C = <h_1^{c1}, h_2^{c2} \ldots h_3^{c3} >$

A global state

$S = (s_1, s_2, \ldots s_N)$

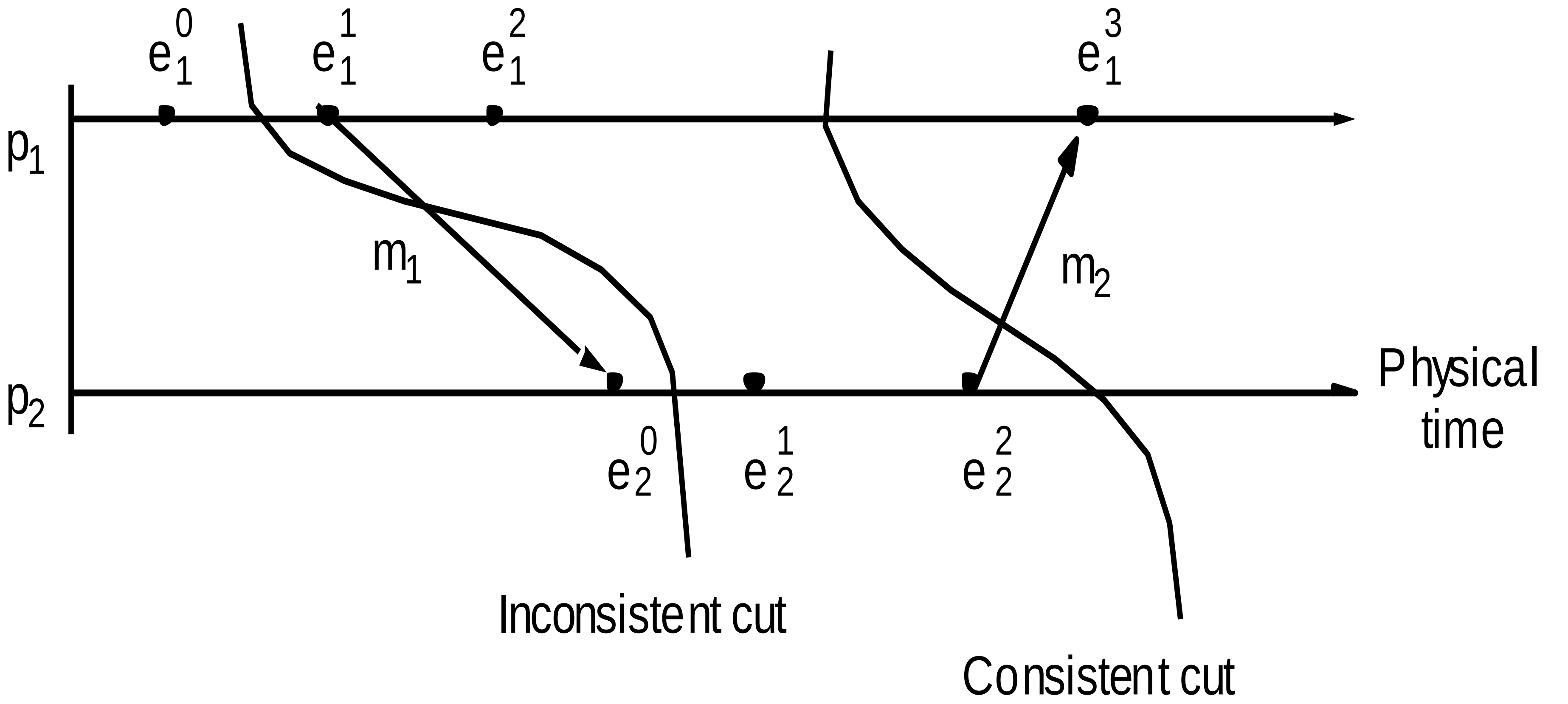
s_i corresponding to the cut of C is that of p_i immediately after the last event processed by p_i *in the* cut- e_i^{ci}

Global states and consistent cuts

A cut C is consistent:

For all events $e \in C, f \rightarrow e \Rightarrow f \in C$

$< e_1^0, e_2^0 >, < e_1^2, e_2^2 >$



Global states and consistent cuts

A consistent global state:

correspond to a consistent cut

The s_i corresponding to the cut C is that of p_i immediately after the last event processed by p_i in C

Execution of a distributed system

$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \ldots$

Global states and consistent cuts

A *run*

a total ordering of all the events in a global history that is consistent with each local history's ordering, $\rightarrow_i$

Not all runs pass through consistent global state

A *linearization* (*consistent*) *run*

an ordering of the events in a global history H that is consistent with this happened-before relation $\rightarrow$ on H.

Pass only consistent global state

Global states and consistent cuts

S'* is reachable from a state *S

there is a linearization that pass through *S* and then *S'*

The “snapshot” algorithm of Chandy and Lamport

- Aim
  - Capture *consistent global state* of a distributed system

The “snapshot” algorithm - assumptions

- Neither channels nor processes fail
- Unidirectional channels, FIFO message delivery
- Complete connection among all processes
- Any process may initiate a global snapshot at any time
- Process may continue execution and send and receive normal message while snapshot takes place

The "snapshot" algorithm - idea

- When one process record a state S_i, make all other processes record states that have been caused by S_i

The “snapshot” algorithm - method

- *Incoming channels, outgoing channels*
- Process state + channel state
- *Marker* message
  - *Marker sending rule:* a process sends a marker after it has recorded its state, but before it send any other messages
  - *Marker receiving rule*: a process records its state if the state has changed since last recording, or record the states of the incoming channel

Marker receiving rule for process p_i

On p_i's receipt of a *marker* message over channel c:

if (p_i has not yet recorded its state) it

records its process state now;

records the state of c as the empty set;

turns on recording of messages arriving over other incoming channels;

else

p_i records the state of c as the set of messages it has received over c

since it saved its state.

end if

Marker sending rule for process p_i

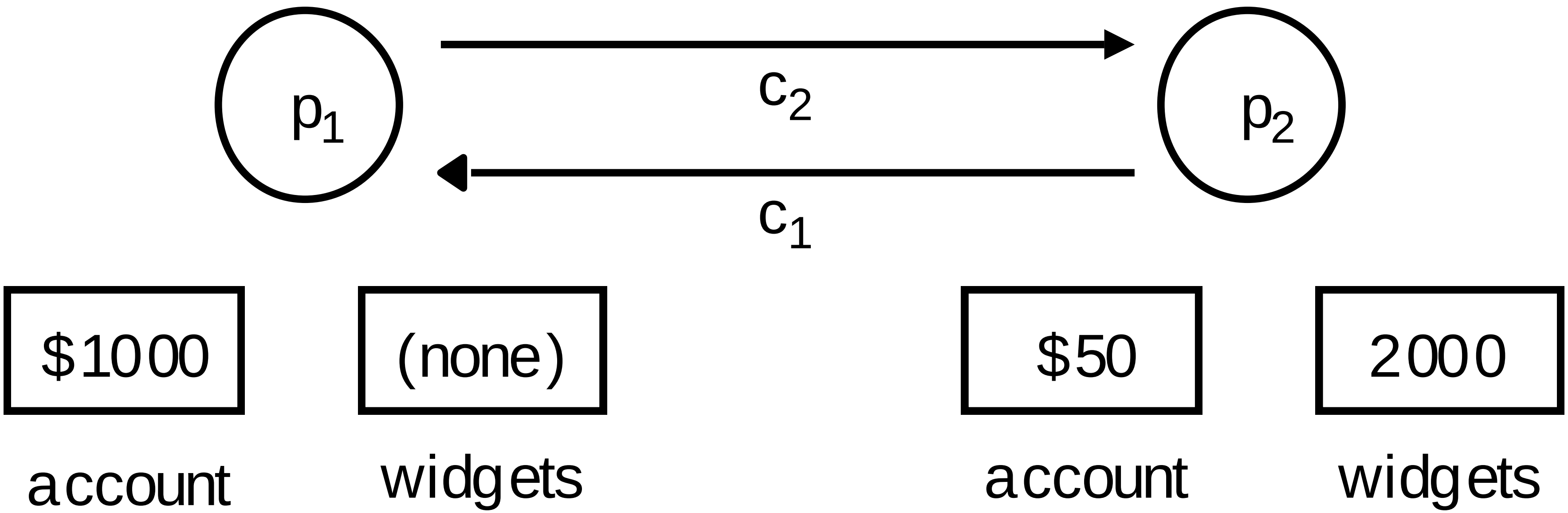
After p_i has recorded its state, for each outgoing channel c:

p_i sends one marker message over c

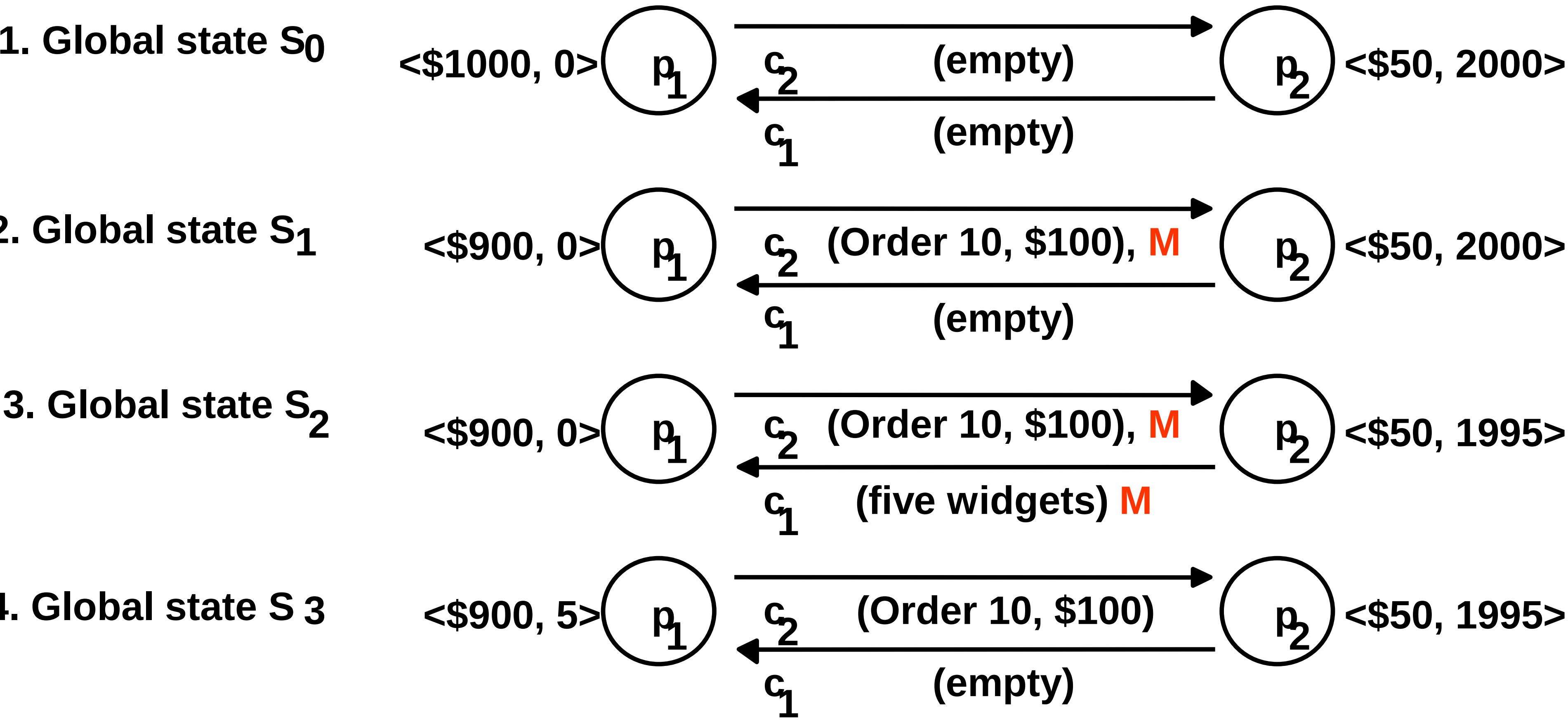
(before it sends any other message over c).

The "snapshot" algorithm - example

- p_1 trade p_2 in widget which is *10$* per item
- Initial state
  - p_1 has sent *50$* to p_2 to buy 5 widget, and p_2 has received the order



Execution of the processes in the example



(M = marker message)

- **The final recorded state**
  - P_1:<$1000, 0>; p_2:<$50,1995>;c_1:<(five widgets)>;c_2:<>

Characterising the observed state - proof

- **The caught states are consistent**
  - *Examine two events e_i, e_j between p_i and p_j, such that $e_i \rightarrow e_j$*

Proof:

We want to prove:

if e_j occurred before p_j recorded its state, then e_i must have occurred before p_i recorded its state

The opposite of what we want to prove:

p_i recorded its state before e_i occurred

Characterising the observed state - proof

Proving:

Because $e_i \rightarrow e_j$, then there are messages m1, m2… at p_j.

Before these messages, there must be a marker saying p_i has recorded its state

These marker message let p_j record state before e_j

So:

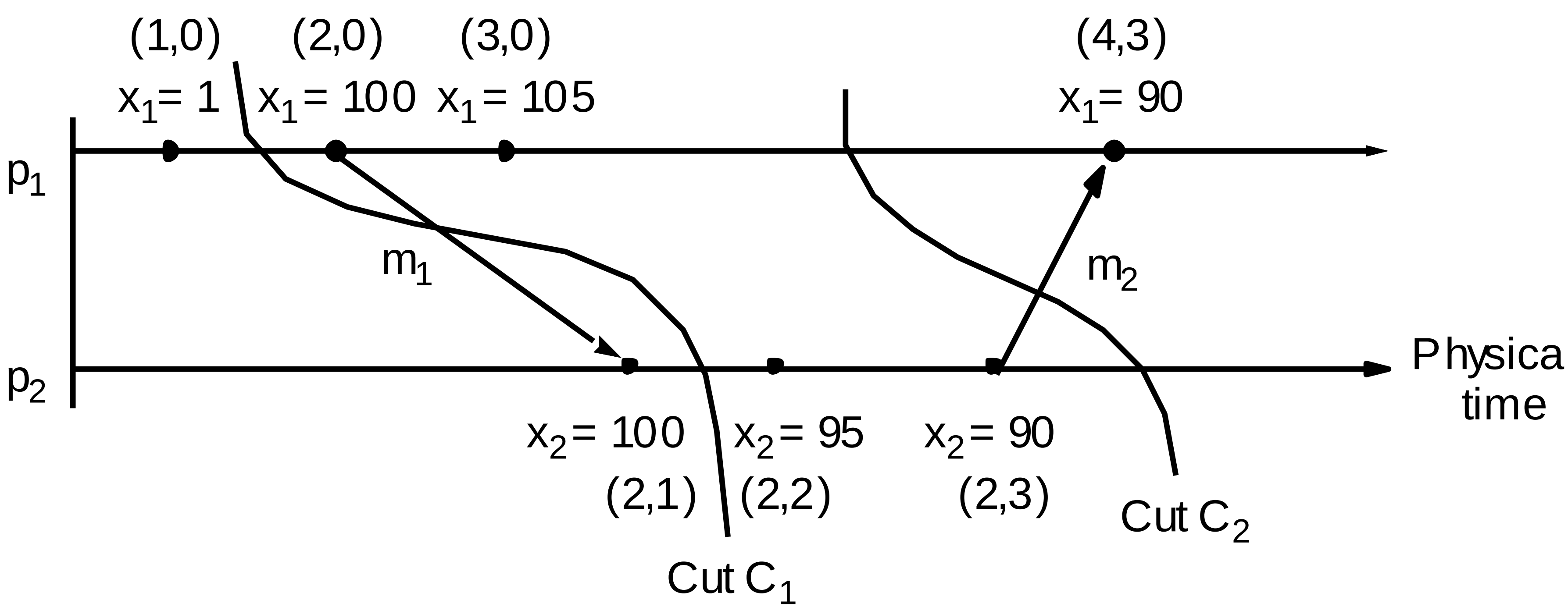
the caught state is consistent

Chapter 10: Time and Global States

- Introduction
- Clocks,events and process states
- Synchronizing physical clocks
- Logical time and logical clocks
- Global states
- Distributed debugging
- Summary

Distributed debug introduction

- **example**
  - Safety condition of a distributed system: $|x_i - x_j| <= \delta$
- **approach**
  - A monitor
    - Collect states of other distributed processes



Observing consistent global states

- **Vector clock at each process**
  - Timestamp each event occurring at each process
  - Each process send the timestamped event to the *monitor*
- **Find consistent global states by the monitor**
  - Let $S = (s_1, s_2, \ldots, s_N)$
    - S is a global state drawn from the state messages that the monitor has received
  - S is a consistent global state if and only if $V(s_i)[i] >= V(s_j)[i]$ for $i,j = 1,2,\ldots, N$
  - If one process's state depends upon another, the global state also encompasses the state upon which it depends

Observing consistent global states … continued

- The lattice of collected global states
  - Monitor construct the reachability lattice by the consistent global state identification algorithm
    - Find consistent global states
    - Establish the reachability relation between states
      - S_{ij} is in level $(i+j)$
  - Show all the linearization corresponding to a history

Observing consistent global states

- **Apply a given global state predicate ϕ on the states**
  - Possibly ϕ: there is a consistent global state s through which a linearization of H passes such that $\phi(s)$ is true
  - Definitely ϕ: for all linearizations L of H, there is a consistent global state set S through which L passes such that $\phi(S)$ is true

Evaluating possibly ϕ and definitely ϕ

- Evaluating possibly ϕ
  - There is a downwards way in which there is a state evaluated to *True* by ϕ
- Evaluating definitely ϕ
  - There is no downwards way in which there is not a state evaluated to *True* by ϕ
- Example
  - If ϕ evaluates to True in the state at level 5, then definitely ϕ
  - If ϕ evaluates to false in the state at level 5, then possibly ϕ

Evaluating possibly ϕ and definitely ϕ in synchronous systems

- **Asynchronous systems**
  - High time cost
    - To find consistent global state $S = (s_1, s_2, \ldots, s_n)$, the monitor Should examine any two local states s_i and s_j
- **Synchronous systems**
  - $|C_i(t)\text{-}C_j(t)| < D$ for $i,j = 0, 1,\ldots, N$
- **Algorithm modification**
  - The observed process sends vector time and physical time with the event to the monitor
  - Monitor find consistency state
    - $V(s_i)[i] >= V(s_j)[i]$
    - s_i and s_j should occurred at the same real time

Chapter 10: Time and Global States

- Introduction
- Clocks,events and process states
- Synchronizing physical clocks
- Logical time and logical clocks
- Global states
- Distributed debugging
- Summary

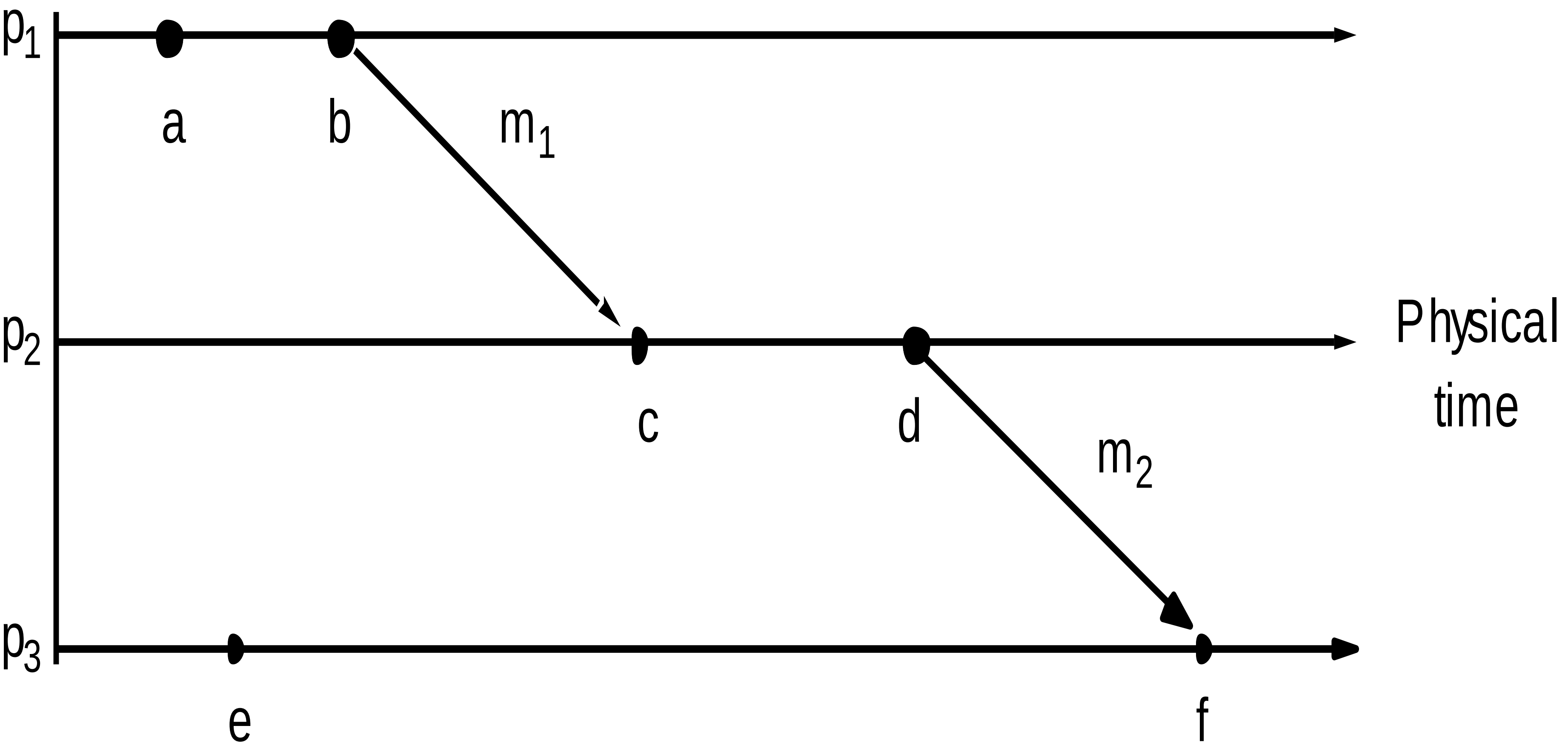
Summary

- Clock skew, clock drift
- Synchronize physical clocks
    - Christian’s algorithm
    - Berkeley algorithm
    - Network Time Protocol
- Logical time
    - Happen-before relation
    - Lamport timestamp algorithm
    - Vector clock

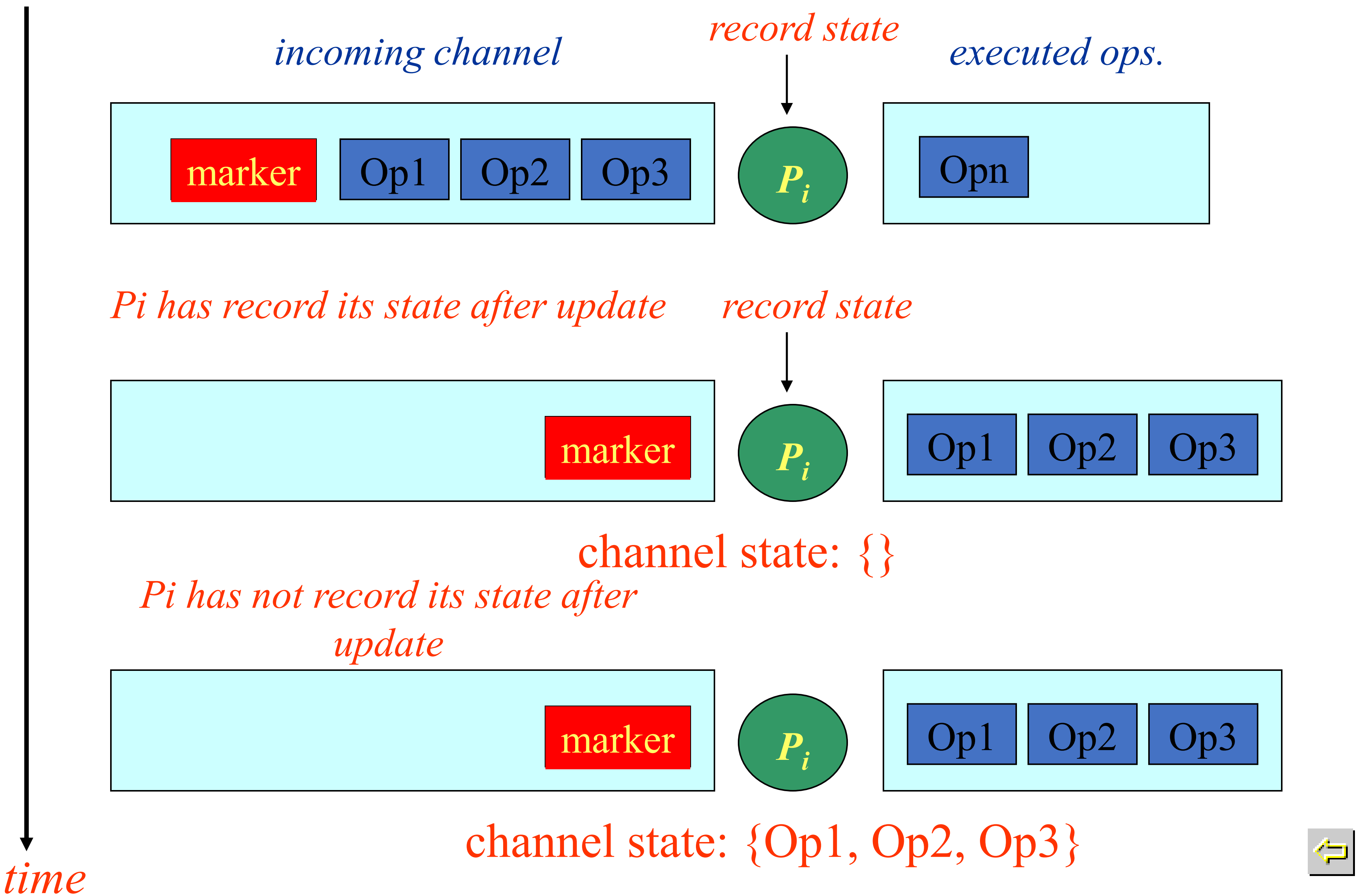
Summary …*continued*

- Global states
  - Consistent cut, consistent state
  - Snapshot algorithm
  - Construct reachability relationship by snapshot
- Global debugging
  - The monitor collects distributed events with vector timestamp
  - Construct reachability relationship
  - Examine possibly ϕ and definitely ϕ

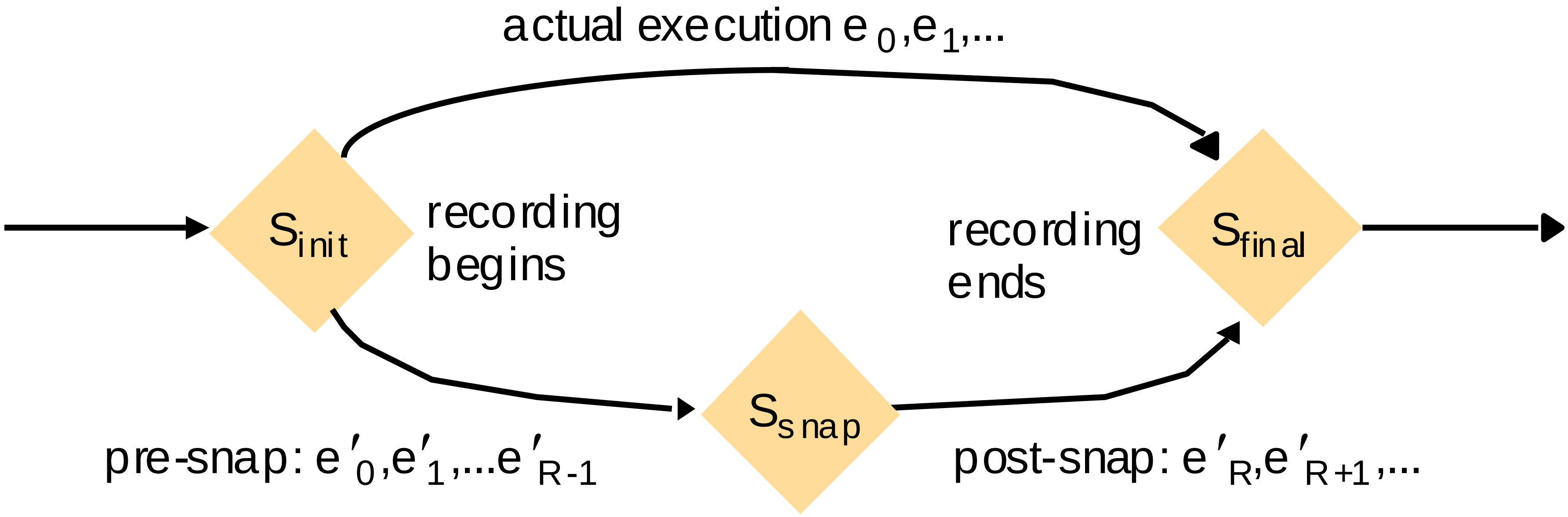
Events occurring at three processes



P_i has record its state?



Reachability between states in the snapshot algorithm



Find pre-snap events and post-snap events

1. The snapshot is consistent global states that record a set of events that occurred on some processes

2. Approach:

   Swap e_j that should belong to *post-snap events* and e_{j+1} that should belong to *pre-snap events* according to the *snap*

3. Analysis

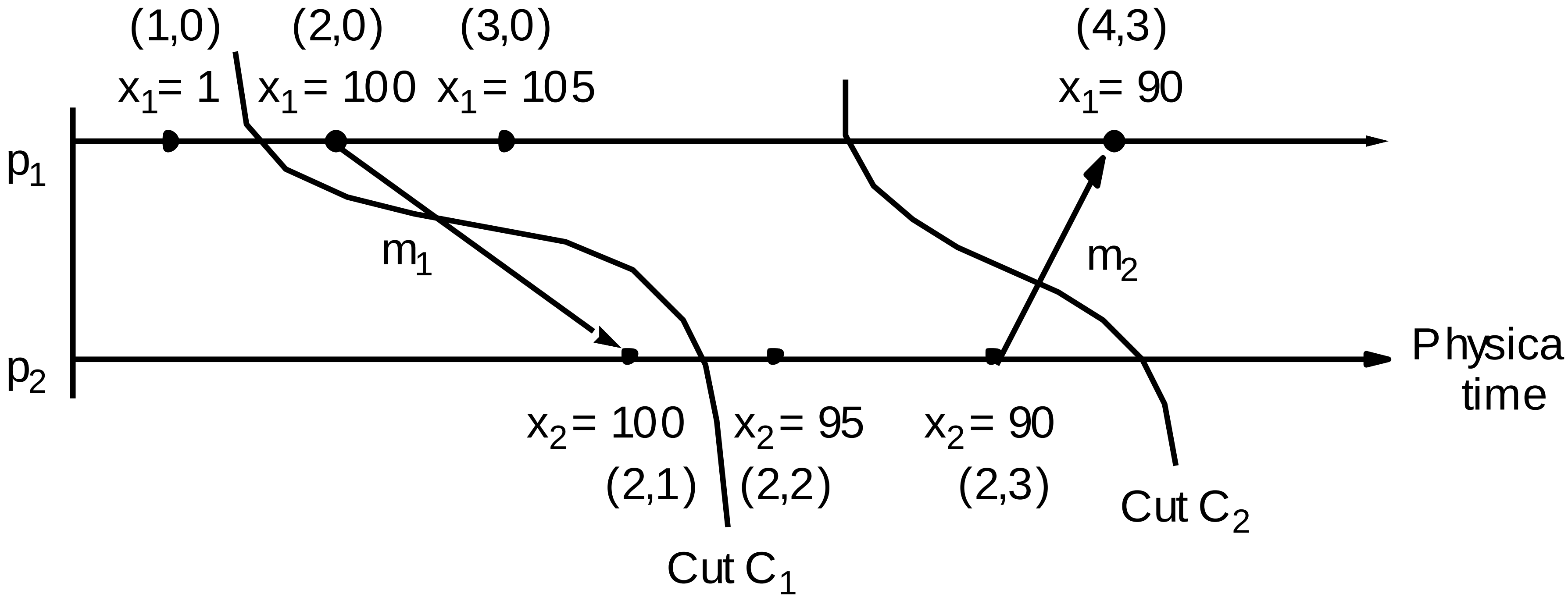
   (1) This situation could not happen if $e_j \rightarrow e_{j+1}$

   Since if e_{j+1} belongs to the pre-snap events, because the *snapshot* is consistent global states, so e_j must belongs to the pre-snap events

   (2) This situation could happen if and only $e_j \parallel e_{j+1}$

   Then swap e_j and e_{j+1} will not change the happen-before relationship, so the linearization condition isn't broken

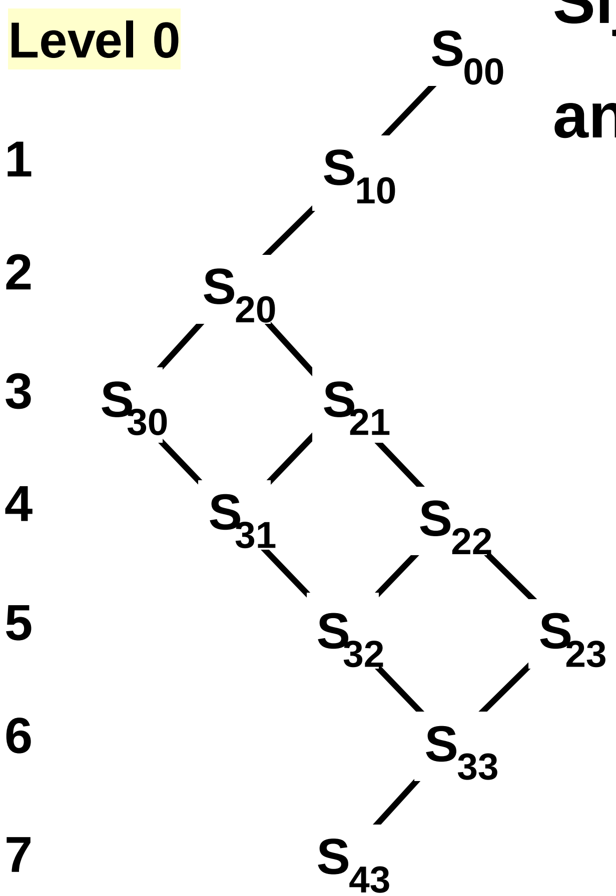
Vector timestamps and variable values for the execution of Figure 10.9

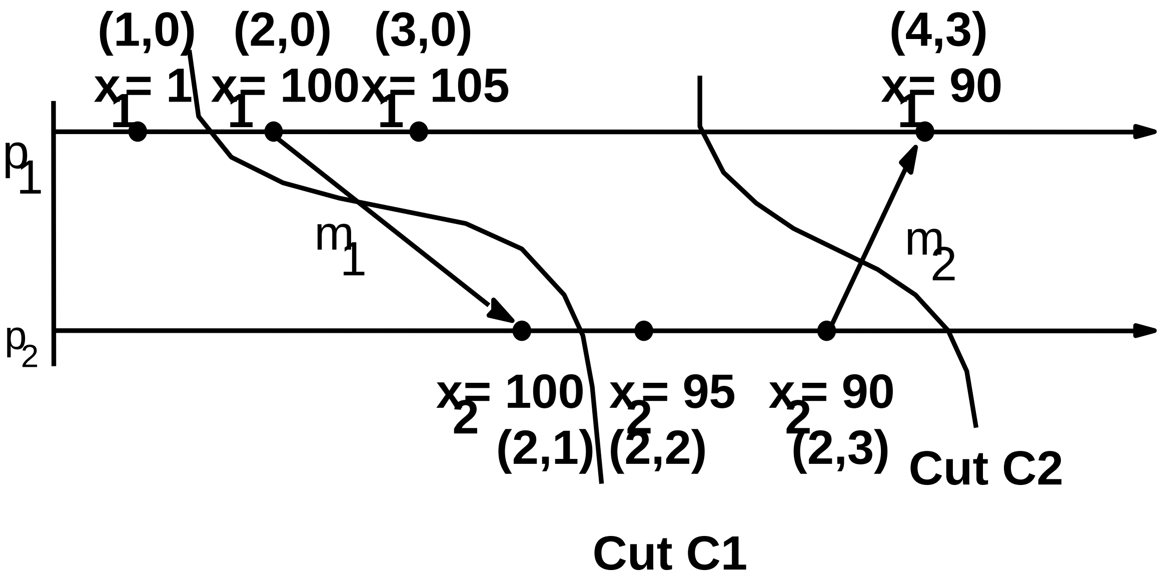


The lattice of global states for the execution of

Sij= global state after i events at process 1

and j events at process 2





Algorithms to evaluate *possibly* ϕ and *definitely* ϕ

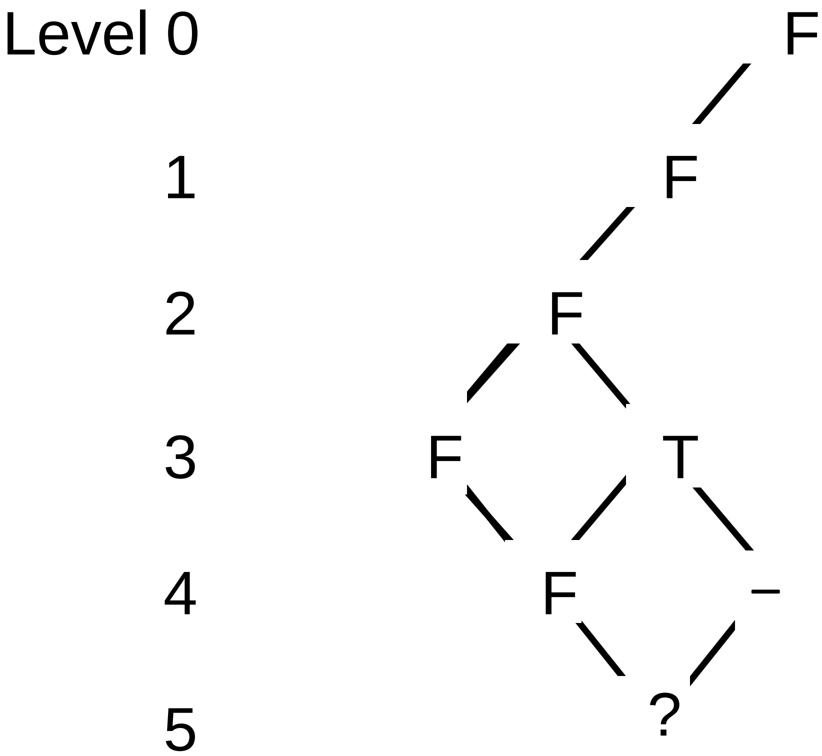
1. *Evaluating possibly ϕ for global history H of N processes*

$L := 0;$
$States := \{ (s_1^0, s_2^0, \ldots, s_N^0) \};$
while $(\phi(S) = False$ for all $S \in$ States$)$
$\quad L := L + 1;$
$\quad Reachable := \{ S' : S'$ reachable in H from some $S \in$ States $\wedge$ level$(S') = L \};$
$\quad States := Reachable$
end while
output "*possibly* ϕ";

2. *Evaluating definitely ϕ for global history H of N processes*

$L := 0;$
if $(\phi(s_1^0, s_2^0, \ldots, s_N^0))$ *then* $States := \{\}$ *else* $States := \{ (s_1^0, s_2^0, \ldots, s_N^0) \};$
while $(States \neq \{\})$
$\quad L := L + 1;$
$\quad Reachable := \{ S' : S'$ reachable in H from some $S \in$ States $\wedge$ level$(S') = L \};$
$\quad States := \{ S \in Reachable : \phi(S) = False \}$
end while
output "*definitely* ϕ";

Evaluating *definitely* ϕ



$F = (\phi(S) = \text{False}); T = (\phi(S) = \text{True})$

Global state predicates, stability, safety and liveness

- **Global state predicates**
  - A function that maps from the set of global states of processes in the system **£** to {True, False}
- **Characteristics of global state predicates**
  - ***Stability***: once the system enters a state in which the predicate is *True*, it remains True in all future states reachable from that state
    - Useful in deadlock detecting, or termination detecting
  - ***Safety* with respect to predicate α**: α evaluates to *False* for all states *S* reachable from S_0
    - E.g., α is a property of being deadlocked
  - ***Liveness* with respect to predicate β**: for any linearization *L* starting in the state S_0, Evaluates to *True* for some state S_L reachable from S_0
    - *E.g.*, β is a property of reaching termination

Characterising the observed state

- **Construct reachability relationship**
  - Reachability between the observed global state and the initial and final global states
  - $Sys = e_0, e_1, \ldots$: linearization of the system as it executed
  - Find a permutation of Sys, $Sys` = e_0`, e_1`, \ldots$ such that all three states $S_{init}$, $S_{snap}$ and $S_{final}$ occur in $Sys`$
    - Sys` is also a linearization
  - Approach
    - Find pre-snap events / post-snap events according to a snap
  - figure