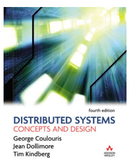
Slides for Chapter 11: Time and Global State



From **Coulouris, Dollimore and Kindberg**
Distributed Systems:
Concepts and Design
Edition 4, © Pearson Education 2005

Logical time and clocks (11.4)

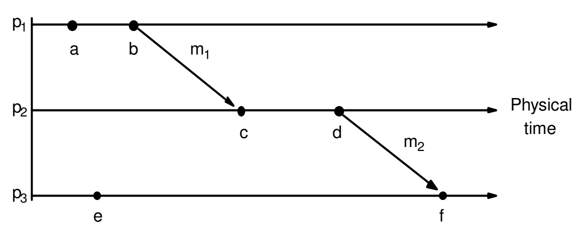
- can't synchronize physical clocks perfectly
- absolute time might not be necessary, just need ordering of events
- logical clocks: Lamport, 1978
- happened-before relationship among events
- potential causal ordering

Happen-before relation

- *e -> e'*
  - two events occur in the same process
  - a message is sent between two processes
- Happened-before relation:
  - HB1: if *e -> e'* in process *i*, then *e -> e'*
  - HB2: for any message *m*, send(*m*) -> receive(*m*)
  - HB3: if *e -> e'* and *e'* -> *e*", then *e -> e"*

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

Events occurring at three processes



Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

Happen-before vs. causality

- *e* || *e'* if the two events aren't in a particular order (concurrent)
- potential causality: *e -> e'* doesn't mean that e causes *e'*
- naturally, if e causes *e', e -> e'*

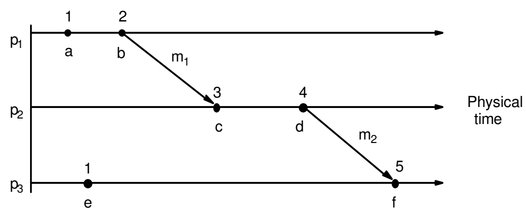
Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

Logical clocks (Lamport)

- each process *i* keeps a local clock L_i
- updating logical clocks:
  - LC1: $L_i := L_i + 1$ for each event in process *i*
  - LC2:
    - When a process *i* sends a message *m*
      - it piggybacks on *m* the value of $t = L_i$
    - Upon receiving *(m,t)*, process *j*
      - computes $L_j := \max(L_j, t)$ and then
      - applies LC1 before timestamping the event receive(*m*)
- if *e -> e'*, then $L(e) < L(e')$
- but $L(e) < L(e')$ doesn't imply *e -> e'*. Why? (Fig. 11.6)

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

Lamport timestamps (Fig 11.6)



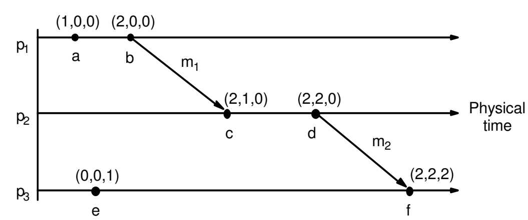
Totally ordered logical clocks

- different processes, same Lamport time
- Time: (T_i, i)
- $(T_i, i) < (T_j, j)$ iff
  - $T_i < T_j$ or
  - $T_i = T_j$ and $i < j$

Vector clocks

- Lamport clocks: *L(e) < L(e')* doesn't imply *e -> e'*
- each process keeps its own vector clock V_i
- piggyback timestamps on messages
- updating vector clocks:
  - VC1: Initially, $V_i[j] := 0$ for p_i, j=1.. N (N processes)
  - VC2: before p_i timestamps an event, $V_i[i] := V_i[i]+1$
  - VC3: p_i piggybacks $t = V_i$ on every message it sends
  - VC4: when p_j receives a timestamp t, it sets $V_j[k] := \max(V_j[k], t[k])$ for k=1..N (merge operation)

Vector timestamps (Fig 11.7)



Vector clocks

- At p_i
  - $V_i[i]$ is the number of events p_i timestamped locally
  - $V_i[j]$ is the number of events that have occurred at p_j (that has potentially affected p_i)
  - Could more events than $V_i[j]$ have occurred at p_j?

Comparing vector timestamps

- $V = V'$ iff
  - $V[j] = V'[j]$, $j = 1 .. N$
- $V <= V'$ iff
  - $V[j] <= V'[j]$, $j = 1 .. N$
- $V < V'$ iff
  - $V <= V'$ and $V != V'$
    - Different from < in all elements

Vector timestamps

- if $e \rightarrow e'$, then $V(e) < V(e')$
- if $V(e) < V(e')$, then $e \rightarrow e'$. (Exercise 11.13)
  - Figure 11.7
    - neither $V(c) <= V(e)$ nor $V(c) >= V(e)$
    - $c \parallel e$
- Disadvantage compared to Lamport timestamps?