Lecture 10: Time & Clocks

CDK4: Sections 11.1 – 11.4
CDK5: Sections 14.1 – 14.4
TVS: Sections 6.1 – 6.2

Topics:

Synchronization

Logical time (Lamport)

Vector clocks

We assume there are benefits from having different systems in a network able to agree the answers to time-related questions ….

Synchronization

- Two parts:
  - Difficulty of setting the same time
  - Clock drift – i.e. difficulty of maintaining synchronization once achieved

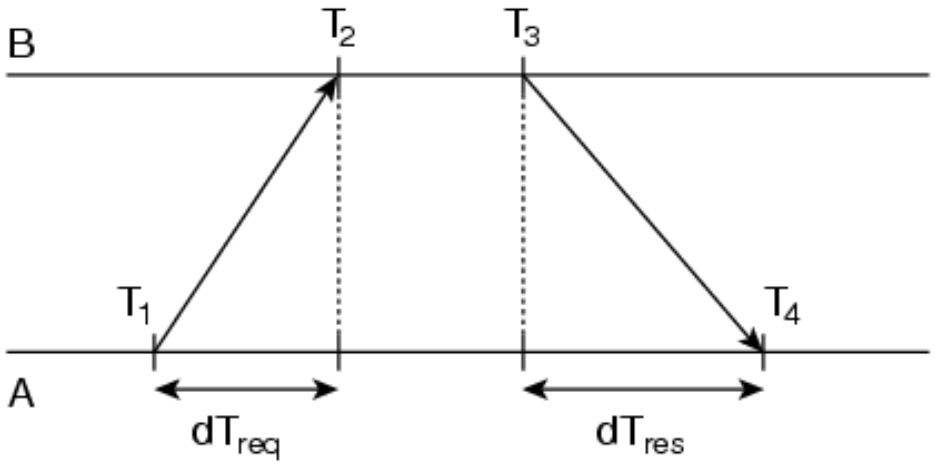
UTC (Coordinated Universal Time)

- International Atomic Time – derived from clocks with "atomic oscillators", drift rate about 1 in 10^13
- Astronomical time derived from stars, sun, etc.
- Slowing of earth's rotation leads to divergence
- UTC based on atomic time, but with occasional insertion of leap seconds to keep it in step with astronomical time
- UTC broadcast by terrestrial radio and satellite (GPS)

Computer Time

- GPS receivers accurate to about 1 microsec.
- Receivers from terrestrial stations, or over dedicated telephone line – to a few millisec.
- In reality few computers in a network have either of these ways of setting the time
- And then there is drift (typically 1 in 10^6 for inexpensive crystal clocks)

TVS figure 6.6 – synchronizing clocks



A's offset from B:

((T2 – T1) + (T4 – T3)) /2

Cristian's Clock Synchonization

- With a "time server", clients set their own clocks by measuring the round-trip time to process their request, rtt, and adding half that to the time in the reply
- Assumes time-out = time-back, more likely to be true for short rtt
- If good estimate of min transmission time available can estimate accuracy

The Berkeley Algorithm

- 1 processor, the *master*, polls others (*slaves*)
- Slaves reply with their times
- Master estimates their local times using round-trip times (as above)
- Master averages all these (and own time) – eliminating any times with excessive rtt
- Also eliminates any clocks wrong wrt others

The Berkeley Algorithm (cont.)

- Rather than send back correct time, master sends back to each slave its own delta (+/-)
- If the master fails, a *distributed election algorithm* exists to elect one of the slaves as replacement
- Cristian's algorithm & the Berkeley algorithm designed (primarily) for intranets

Network Time Protocol (NTP)

- Designed for larger scale internet
- Network of servers:
  - Primary (stratum 1)– with UTC clock
  - Secondary (stratum 2), synchronized with primary
- Can reconfigure – e.g. if UTC source fails primary can become secondary, etc.

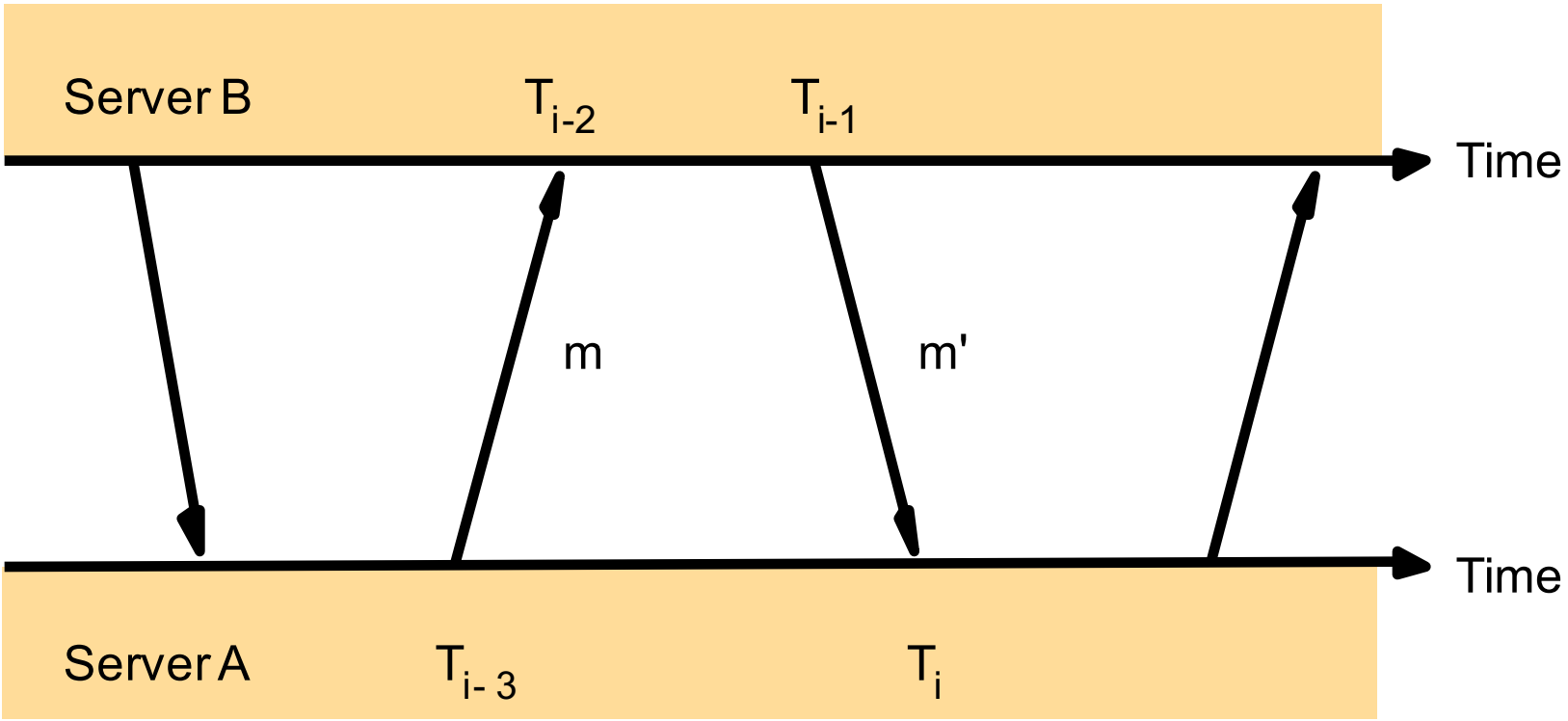
NTP Synchronization

- Three methods of synchronization
  - Multicast mode
  - Procedure call mode
  - Symmetric mode
- Multicast mode used on high-speed LANs
  - Server sends time to all servers on LAN at once
  - Each reset clocks (assuming a small delay)
  - Not highly accurate

Procedure-call mode

- Procedure-call mode:
  - Effectively Cristian's algorithm
  - Server accepts requests and replies with the time
  - Used when multicast not supported or higher accuracy required
- Symmetric mode
  - Used where highest accuracy is required
  - Messages exchanged, and data built up to improve accuracy of synchronization over time.
  - Each message sent contains timing info about the previous message received (time sent, time received) and time it is sent

CDK Figure 11.4 Messages exchanged between a pair of NTP peers



Using the information

- Use this information to estimate the offset between the two clocks, o, from the equations (where t, t' are transmission times for m, m' resp.), and a d, delay, total transmission time of the two messages.

$$T_{i-2} = T_{i-3} + t + o$$

$$T_i = T_{i-1} + t' - o$$

Hence:

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

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

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

Using the fact that t and t' are both >= 0, leads to

$$o_i - d_i/2 \leq o \leq o_i + d_i/2$$

Data filtering

- NTP servers filters successive (o,d) values to identify best (lowest d value), and measure the reliability of the other server
- Each server will interact with several peers identifying most reliable ones
- Achieves accuracies of 10s of millisec over internet paths

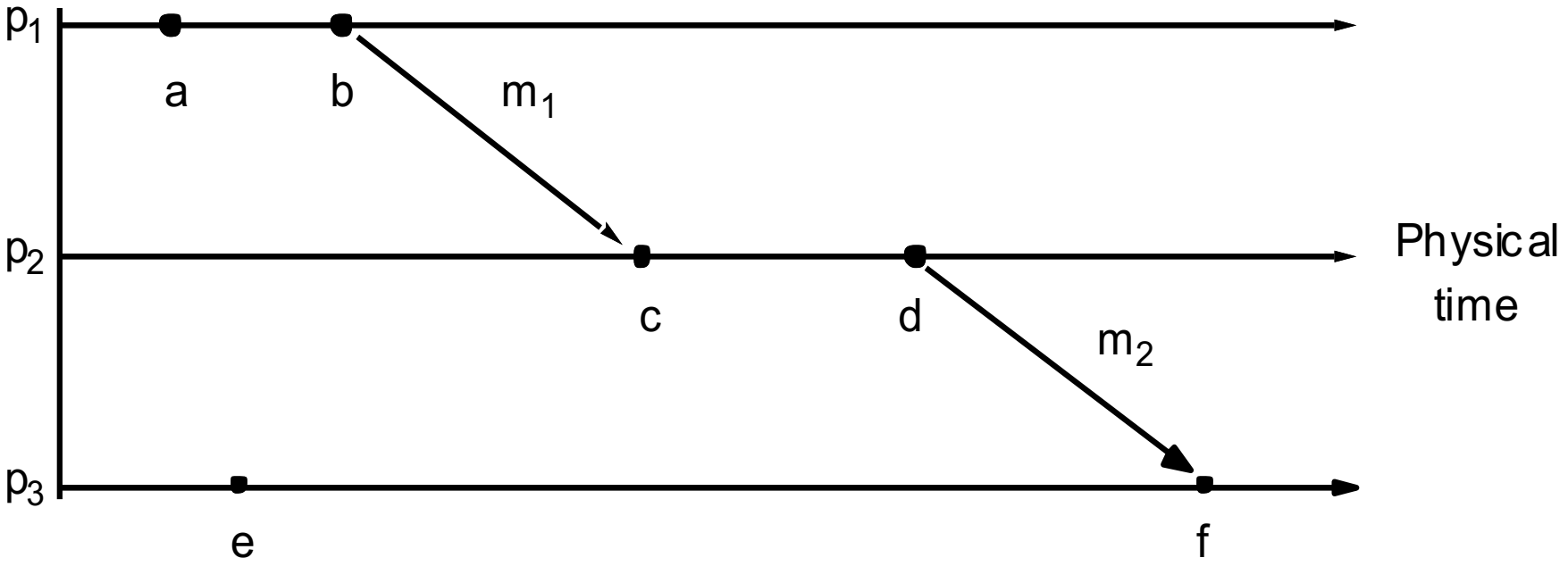
Logical Time (Lamport)

- In single processor, every event can be uniquely ordered in time using the local clock
- What we want is to be able to do this in a distributed system, where synchronization between clocks is not sufficiently good to use physical time

Simple principles

- If two events happen in the same process, they occur in the order given by that process
- If a message is sent from 1 process to another, the event of sending happens before the event of receiving
- These define a partial ordering of events, given by the *happens-before* relationship

CDK Figure 11.5
Events occurring at three processes



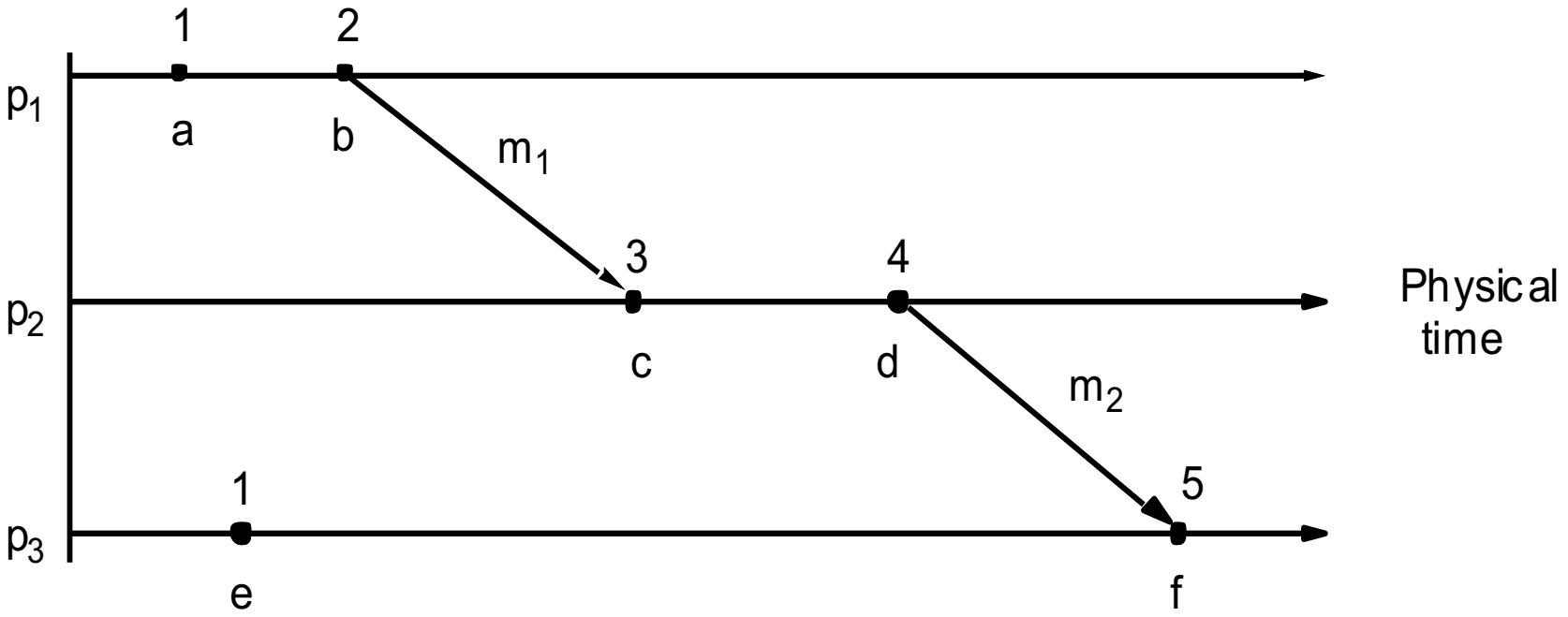
Logical clocks

- A logical clock is a monotonically increasing software counter
- Each process keeps its own, L, and uses it to timestamp events
  - L++ before each event
  - Each message sent contains current L (as t)
  - Each message received sets L = max(L,t)+1

Logical clocks (cont.)

- Now if event e1 *happens-before* e2,
  $L(e1) < L(e2)$
- Note that the converse is not true, i.e. we cannot deduce ordering from the timestamps

CDK Figure 11.6
Lamport timestamps for the events shown in CDK Figure 11.5



Totally ordered logical clocks

- Can make the ordering of events above total, so that there is an order between every pair of events, by using an ordering of process identifiers (using local timestamps) to resolve cases where logical clocks are the same in different processes
- This has no physical reality, but may be used to control entry to critical sections, etc.

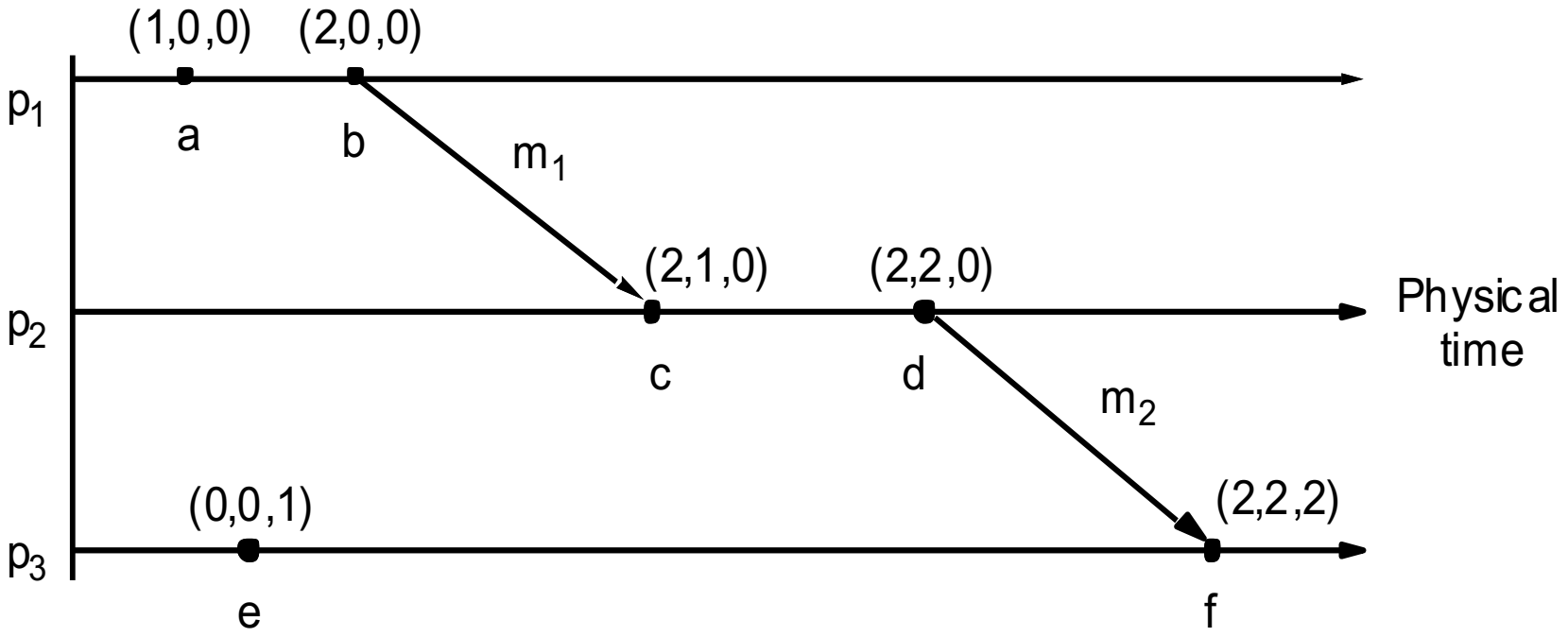
Vector Clocks

- A vector clock in a system with n processes is an array of n integers
- Each process keeps its own
- Messages between processes contain the vector clock of the sender as a timestamp
- Each clock starts with all integers 0

Vector clocks (cont.)

- Events in process i increment the i'th element in its vector clock
- When process i receives a timestamp, t, in a message it resets each element in its clock V[j] = max(V[j], t[j]) for j = 1 …n
- This operation is referred to as a *merge*

CDK Figure 11.7
Vector timestamps for the events shown in CDK Figure 11.5



Comparing Vector clocks

- V1 = V2 iff V1[j] = V2[j] for all j
- V1 <= V2 iff V1[j] <= V2[j] for all j
- V1 < V2 iff V1 <= V2 & V1 != V2
- Now if event e1 *happened-before* event e2, V(e1) < V(e2)
- if V(e1) < V(e2), e1 *happened-before* e2

Advantages and Disadvantages

- We don't end up with an arbitrary order when none is needed (e.g. between c and e in the figures: neither $V(c) < V(e)$ nor $V(e) < V(c)$)
- Cost is the extra amount of data in a timestamp.
- Lamport's clocks do not capture causality, which can be captured by means of vector clocks.