

Overview of Ordering and Logical Time

Prof. Dave Bakken

Cpt. S 464/564 Lecture

November 16, 2011

Context

- This material is NOT in CDKB5 textbook
- Rather, from second text by Verissimo and Rodrigues, chapters 1.4 & 2.7
- Do read the pertinent sections in CKDB5 Chapter 14, however!

Outline

- Logical Time
- Global States
- DS Properties

Logical Time

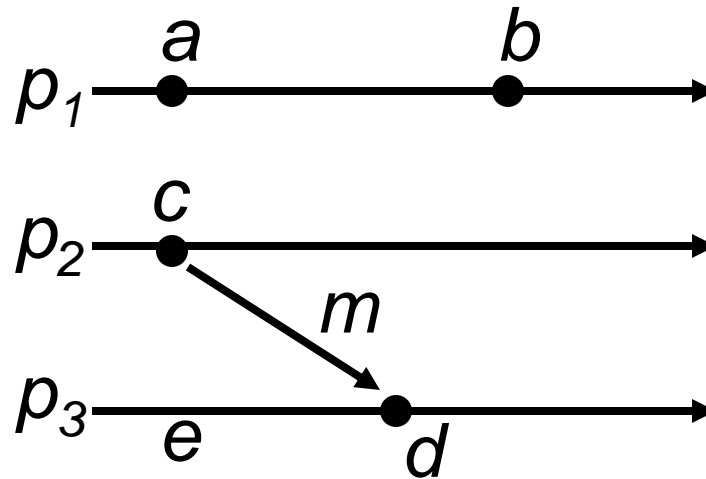
- Time in Distributed Systems
 - Computers can only be synchronized by network messages, but the latency can vary
 - We can not synchronize enough to be able to, in general, tell the ordering of two arbitrary events at different computers.
 - We can, however, establish an ordering on some of the events, and this can be used in many situations.
- Logical Time
 - Builds up a notion of what we can reason about w.r.t. the order of events
 - Defines the “Happened-before” relation
 - Source: Lamport, Leslie. “Time, Clocks and the Ordering of Event in a Distributed System”, *Communications of the ACM*, Vol. 21, July 1978, pp. 558-565.
 - One of the seminal works in distributed systems...
 - **Assigned for 564 students to read (see web page)**

Happened-Before Relation

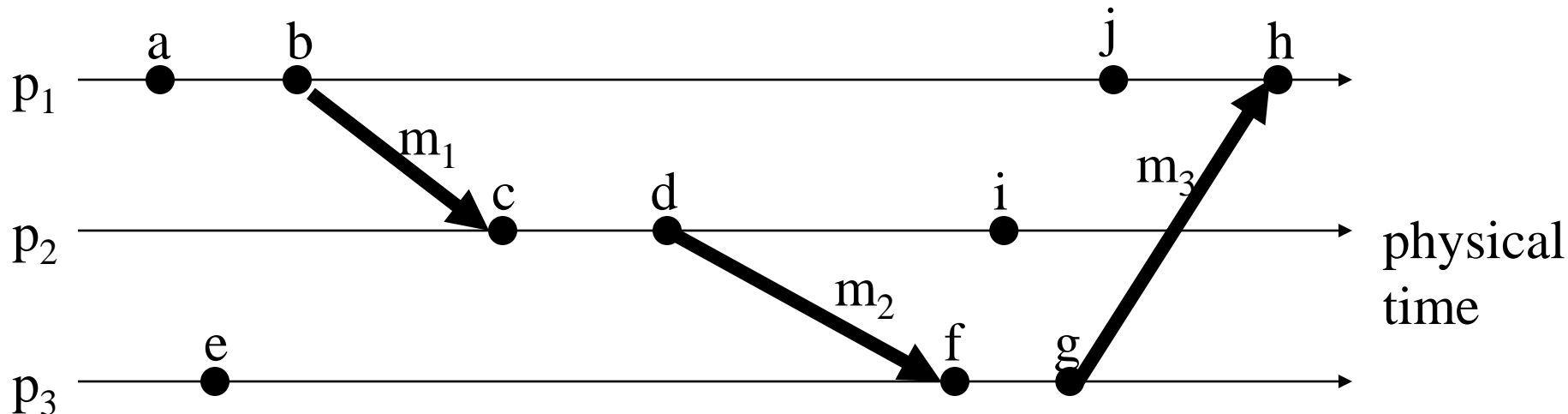
- Happened-Before relation, \rightarrow , based on observations:
 1. If two events occur in the same process, then they occurred in the order in which that process observes them.
 2. The receipt of a message happens after its being sent.
 3. “Happened-before” is transitive
- Corresponding Rules for events x , y , z , process p , and message m
 - HB1: $x -_p -> y$, then $x \rightarrow y$
 - HB2: $\text{send}(m) \rightarrow \text{recv}(m)$
 - Transitivity: $x \rightarrow y$ and $y \rightarrow z$, then $x \rightarrow z$
- Concurrency: If $a \sim \rightarrow b$ and $b \sim \rightarrow a$, then $a \parallel b$ (“ a is concurrent with b ”)
- Note: if $x \rightarrow y$ (“ x happened before y ”) then $y \leftarrow x$ (“ y happened after x ”), notationally

Representing Distributed Computations

- Events at a process can be
 - execution events: internal computations
 - send events: sending a message to another process
 - receive events: receiving a message from another process
- Message exchanges depicted with timelines: e.g.



Happened-Before Example



- Example table of \rightarrow , \leftarrow , \parallel
- Limitations of Happened-Before
 - Covert channels
 - Too pessimistic: some things $a \rightarrow b$ did not have a causing b !
- Happened-before also called
 - Causal ordering
 - Potential causality
 - Lamport ordering
 - (irreflexive) partial ordering

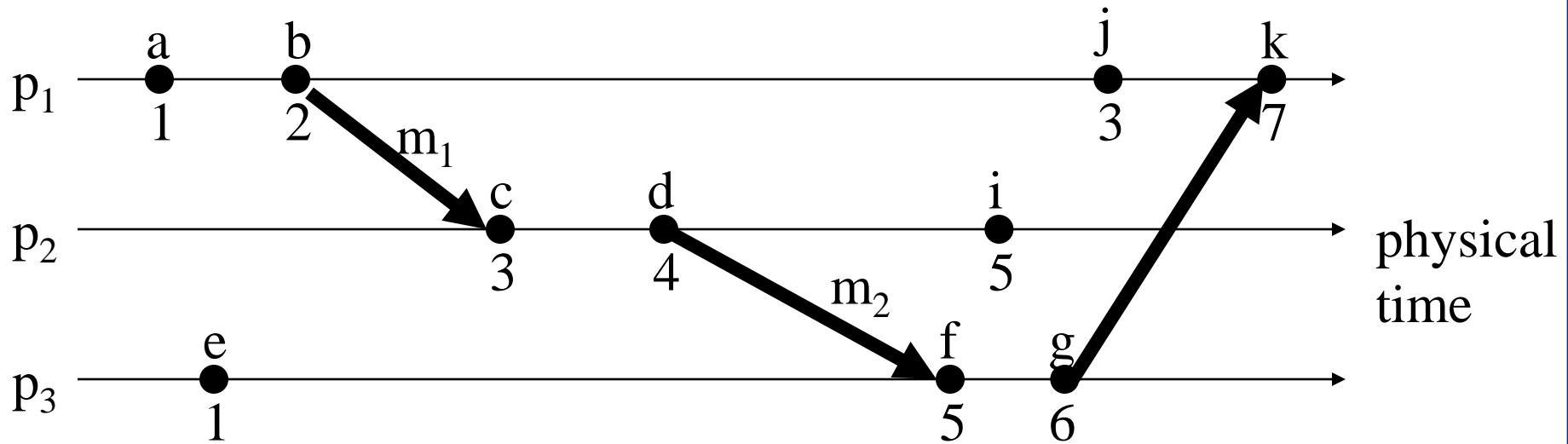
Logical Clocks

- How to implement “Happened Before”??
- Logical Clock, a monotonically increasing counter.
- Let
 - Each process p keeps its own logical clock, C_p , which it uses to timestamp events
 - $C_p(a)$ is the logical time at process p at which event a occurred
 - $C(a)$ is the logical time at which event a occurred at the process it occurred at
- Processes keep their own logical clocks, initialized to 0.

Updated by rules:

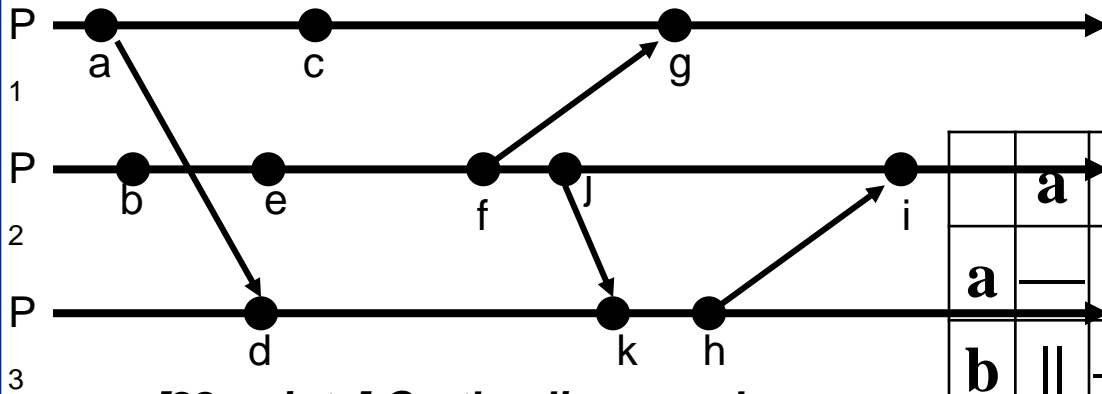
- LC1: Before each event occurs, increment C_p
- LC2:
 - When a process p sends a message m , it piggybacks on m value $t = C_p$
 - When process q receives $\langle m, t \rangle$, q computes $C_q = \max(C_q, t) + 1$ then timestamps m

Logical Clock Example



- Note if $a \rightarrow b$ then $LC(a) < LC(b)$
- However, $LC(a) < LC(b)$ does not imply $a \rightarrow b$
 - Above, $C(e) < C(b)$ yet $b \parallel e$
 - Also note that concurrency is not transitive: $a \parallel e$ and $e \parallel b$ yet $a \rightarrow b$

Logical Time & Clocks from 2003 Midterm



[22 points] On the diagram above, write the Logical Clock (LC) time at its processor for each event to the left of the dot for that event.

[45 points] Fill out the empty cells in the table to give the relations between each event: “ \rightarrow ” denotes “happened before”, “ \leftarrow ” denotes “happened after”, and “ \parallel ” denotes “concurrent”. (For examples of this notation, because there is a message from ‘a’ to ‘d’, it is filled in “ \rightarrow ” in the [a,d] cell and “ \leftarrow ” in the [d,a] cell. Also, ‘b’ and ‘a’ are concurrent, and are so marked.)

	a	b	c	d	e	f	g	h	i	j	k
a		\parallel		\rightarrow							
b	\parallel										
c											
d	\leftarrow										
e											
f											
g											
h											
i											
j											
k											

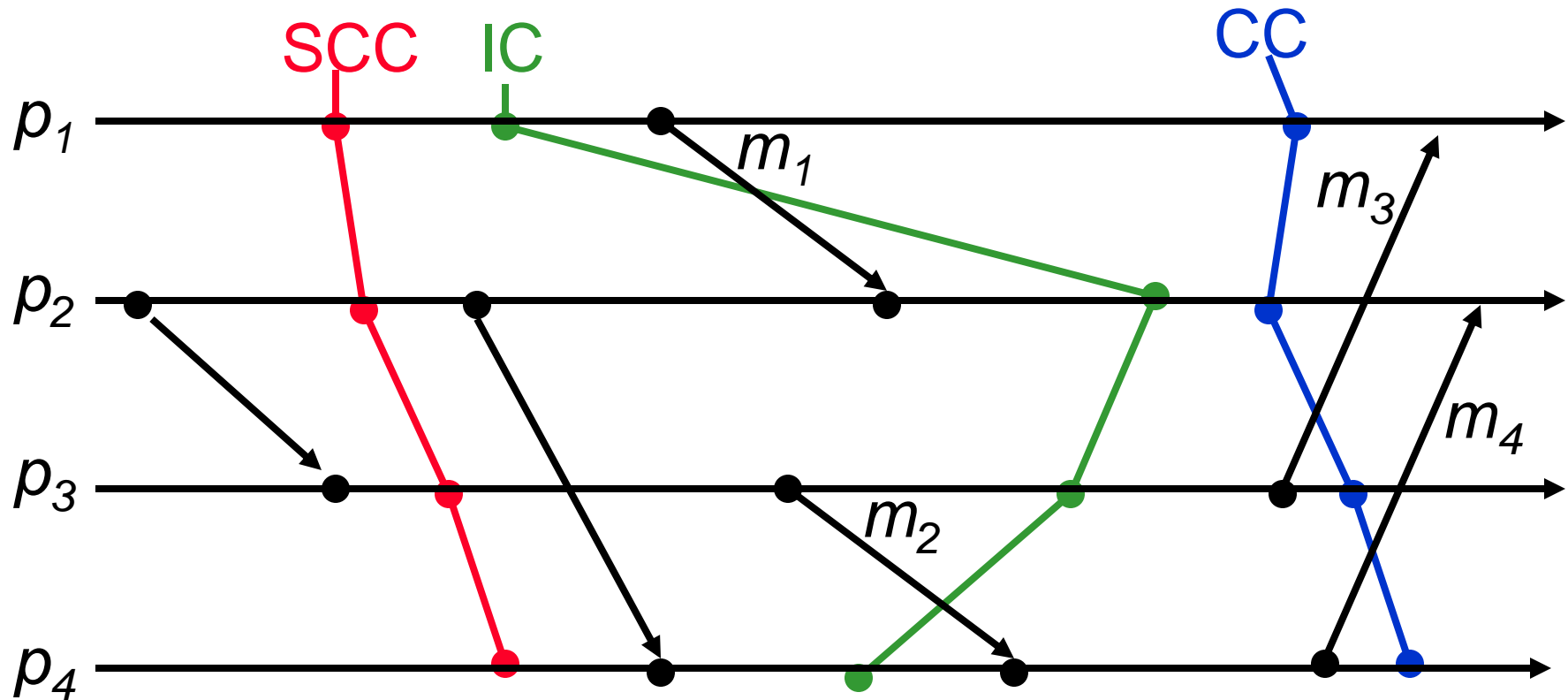
Global States

- Sometimes very useful to get a global “picture” of a distributed system
- Global state (GS) of a DS at any point is a vector of its individual process states: $S = \{S_1, S_2, \dots, S_M\}$
- Two viewpoints of how a system evolves:
 - Interleaving view: system goes through a succession of states (like above)
 - Space-time view: system goes through a partially ordered set of events occurring in several processes in the system
- A cut (in space-time view) is a segment intersecting the timelines of all processes.
 - A cut involves coordination with computers across a DS
 - Many different ways to implement a cut in a DS that provide a range of properties and costs

Global States (cont.)

- Inconsistent cut (IC)
 - Snapshot gives invalid picture of the DS (a state that could never happen)
 - Example: message received but not sent in the snapshot
- Consistent cut (CC)
 - Snapshot gives correct (state that could have happened) but possibly incomplete picture of the DS
 - Example: messages in transit not accounted for in a snapshot
- Strongly consistent cut (SCC)
 - Snapshot is a faithful representation of an actual global state of the DS
 - No messages in transit when state read at each node, atomic checkpoints taken in that interval across nodes, ...
- Note: TvS Chap6 has lots on consistency; we may cover some later in this class...

Example Cuts for Global States



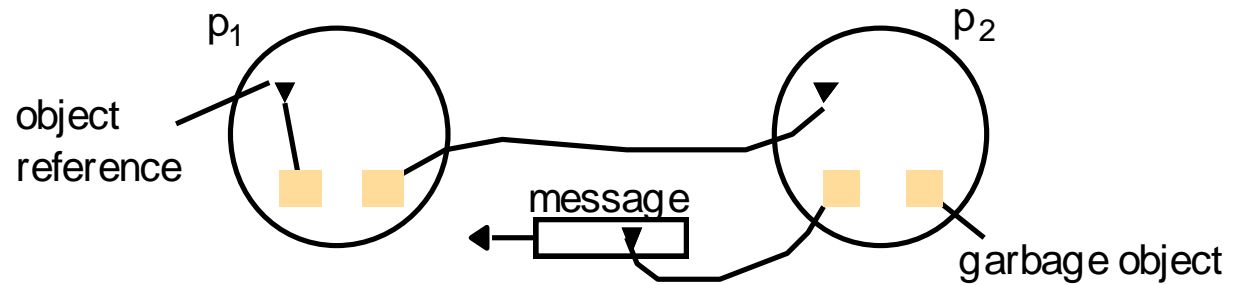
- Strongly consistent cut (SCC): faithfully represents GS of the system
- Inconsistent cut (IC): gives invalid picture of any GS
- Consistent cut (CC): gives valid but possibly incomplete picture of the GS of the system

DS Properties

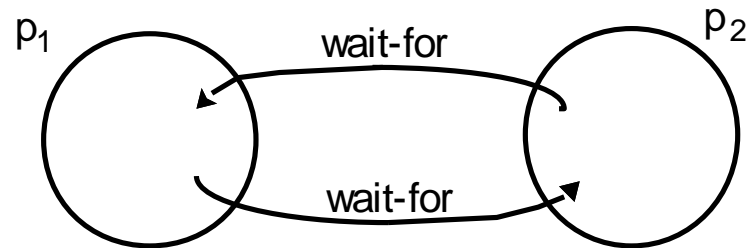
- Goal: specify a system with high-level properties
- Safety properties: something bad (wrong events) never take place
 - Specification: predicate P will never be true in the DS
- Liveness properties: something good (positive event) eventually takes place
 - Specification: predicate P will eventually be true in the DS
- “any delivered message is delivered to all correct participants”: safety property (atomicity)
- “any message sent is delivered to at least one participant”: liveness property
- Timeliness properties specify a time that a predicate will be true in the DS at a given instant in time

Examples of DS Properties (aux)

a. Garbage collection



b. Deadlock



c. Termination

