Hwajung Lee

ITEC452
Distributed Computing

Lecture 10
Time in a Distributed System

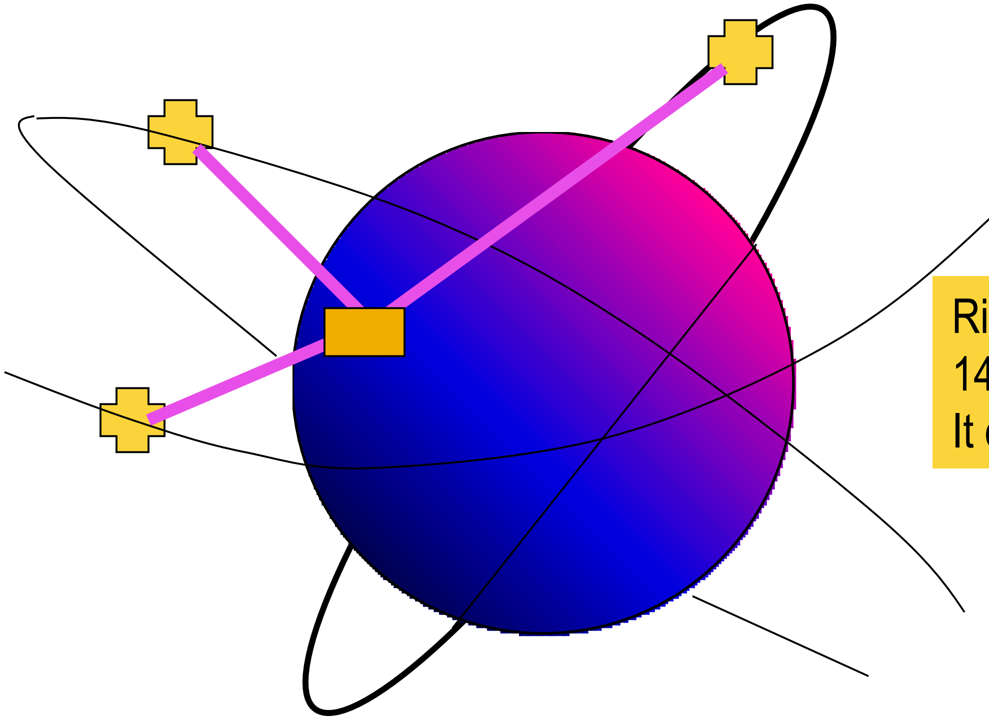
Time and Clock

Primary standard = **rotation of earth**
De facto primary standard = **atomic clock**

(1 atomic second = **9,192,631,770** orbital transitions of **Cesium 133** atom.
86400 atomic sec = 1 solar day – 3 ms

Coordinated Universal Time **(UTC)** = GMT ± number of hours in your time zone

Global positioning system: GPS



Location and precise time computed by triangulation

Right now GPS time is nearly
14 seconds ahead of UTC, since
It does not use leap sec. correction

Per the theory of relativity, an additional correction is needed. Locally compensate by the Receivers.

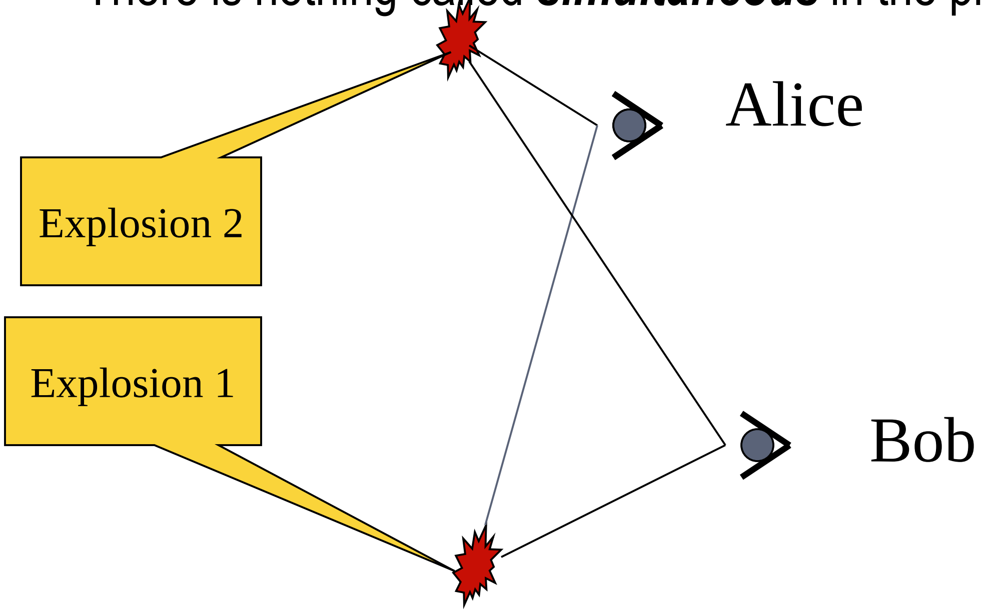
A system of 32 satellites broadcast accurate spatial coordinates and time maintained by atomic clocks

What does “concurrent” mean?

Simultaneous? Happening at the same time?

NO.

There is nothing called ***simultaneous*** in the physical world.



Sequential and Concurrent events

Sequential = Totally ordered in time.

Total ordering is feasible in a single process that has only one clock. This is not true in a distributed system.

Two issues are important here:

- How to synchronize physical clocks?
- Can we define sequential and concurrent events without using physical clocks?

Causality

Causality helps identify ***sequential*** and ***concurrent*** events without using physical clocks.

Joke $\prec$ Re: joke ($\prec$ implies causally ordered before or happened before)

Message sent $\prec$ message received

Local ordering: a $\prec$ b $\prec$ c (based on the local clock)

Defining causal relationship

Rule 1. If **a**, **b** are two events in a single process **P**, and the time of **a** is less than the time of b then $\mathbf{a} \prec \mathbf{b}$.

Rule 2. If **a** = sending a message, and **b** = receipt of that message, then $\mathbf{a} \prec \mathbf{b}$.

Rule 3. $\mathbf{a} \prec \mathbf{b} \wedge \mathbf{b} \prec \mathbf{c} \Rightarrow \mathbf{a} \prec \mathbf{c}$

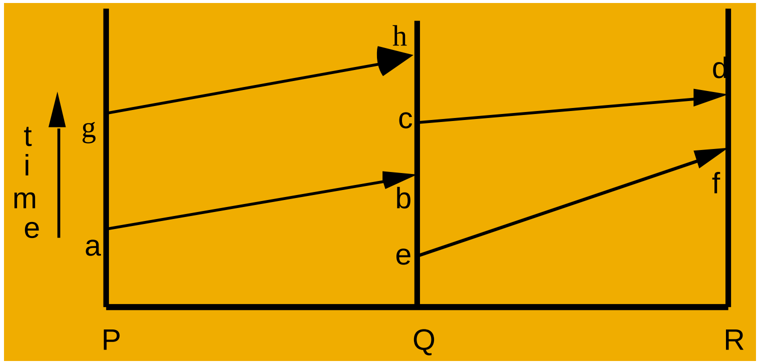
Example of causality

e $\prec$ d?
Yes since (**e** $\prec$ **f** $\wedge$ **f** $\prec$ **d**)

a $\prec$ d ?
Yes since (**a** $\prec$ **b** $\wedge$ **b** $\prec$ **c** $\wedge$ **c** $\prec$ **d**)

(Note that $\prec$ defines a **PARTIAL** order).

Is **g$\prec$ f or f$\prec$ g?**
NO. They are **concurrent.**



Note: a distributed system cannot always be totally ordered.

Concurrency = absence of causal order

Logical clocks

LC is a counter. Its value respects causal ordering as follows

$$a \prec b \Rightarrow LC(a) < LC(b)$$

Each process maintains its logical clock as follows:

LC1. Each time a local event takes place, increment **LC**.

LC2. Append the value of **LC** to outgoing messages.

LC3. When receiving a message, set **LC** to **1 + max (local LC**, **message LC**)

Total order in a distributed system

Total order is important for some applications like scheduling (first-come first served). But total order does not exist! What can we do?

Strengthen the causal order $\prec$ to define a *total order (<<)* among events. Use LC to define total order (in case two LC's are equal, process id's will be used to break the tie).

Let **a, b** be events in processes **i** and **j** respectively. Then

a << b iff
-- LC(a) < LC(b) **OR**
-- **LC(a) = LC(b)** and **i < j**

a $\prec$ **b** $\Rightarrow$ **a << b,** but the converse is not true.

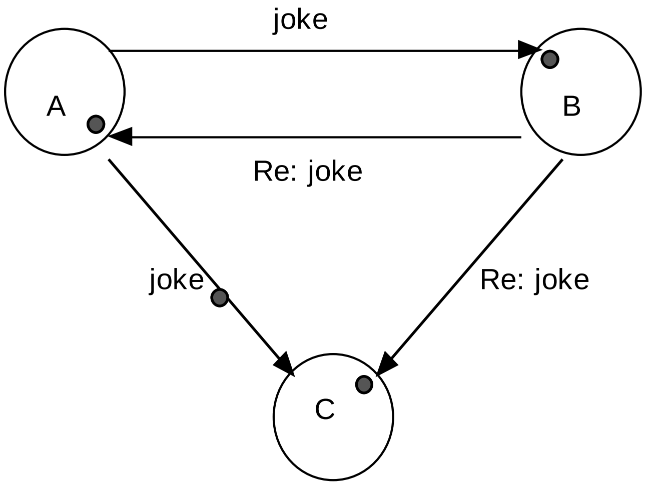
The value of LC of an event is called its *timestamp*.

Vector clock

Causality detection can be an important issue in applications like **group communication**.

Logical clocks do **not** detect causal ordering. **Vector clocks** do. Mapping VC from events to integer arrays, and an order < such that for any pair of a, b:

$$\mathbf{a} \prec \mathbf{b} \Leftrightarrow \mathbf{VC(a)} < \mathbf{VC(b)}$$



C may receive Re:joke before joke, which is bad!

Implementing VC

{Actions of process j}

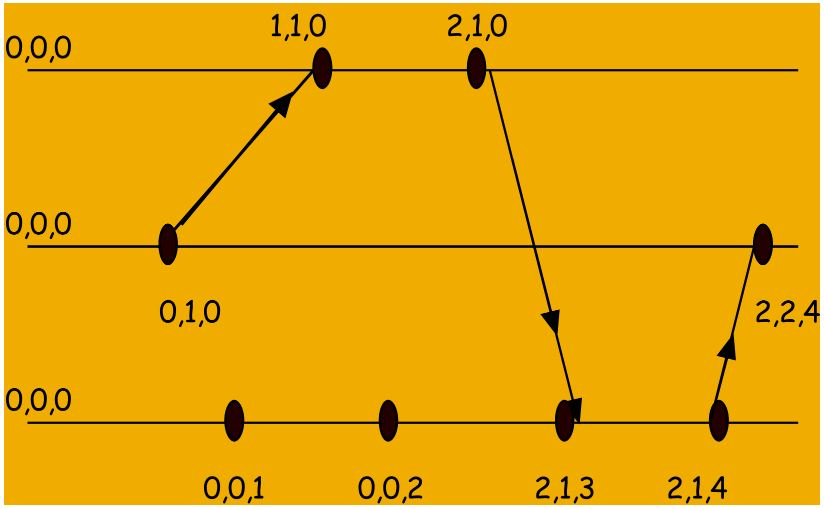
jth component of VC

1. Increment **VC[j]** for each local event.

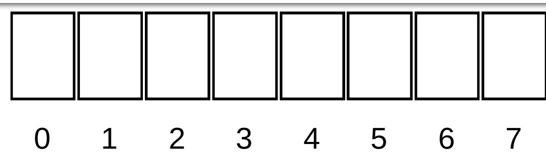
2. Append the local **VC** to every outgoing message.

3. When a process j receives a message with a vector timestamp **T** from another process, first increment the j^{th} component **VC[j]** of its own vector clock, and then update it as follows:

   $\forall$**k: 0 ≤ k ≤N-1:: VC[k] := max (T[k], VC[k]).**



Vector clocks



Vector Clock of an event in a system of 8 processes

Let a, b be two events.

Define. VC(a) **<** VC(b) iff

$\forall i : 0 \leq i \leq N-1 : VC(a)[i] \leq VC(b)[i]$, and

$\exists j : 0 \leq j \leq N-1 : VC(a)[j] < VC(b)[j]$,

VC(a) < VC(b) $\Rightarrow$ **a** $\prec$ **b**

Causality detection

Example

[3, 3, 4, 5, 3, 2, 1, 4] <
[3, 3, 4, 5, 3, 2, 2, 5]

But,

[3, 3, 4, 5, 3, 2, 1, 4] and
[3, 3, 4, 5, 3, 2, 2, 3]
are not comparable