Logical Clocks

dDistSys

11/05/2015

Johan K. Jensen

Happened-before relationship

Logical clocks

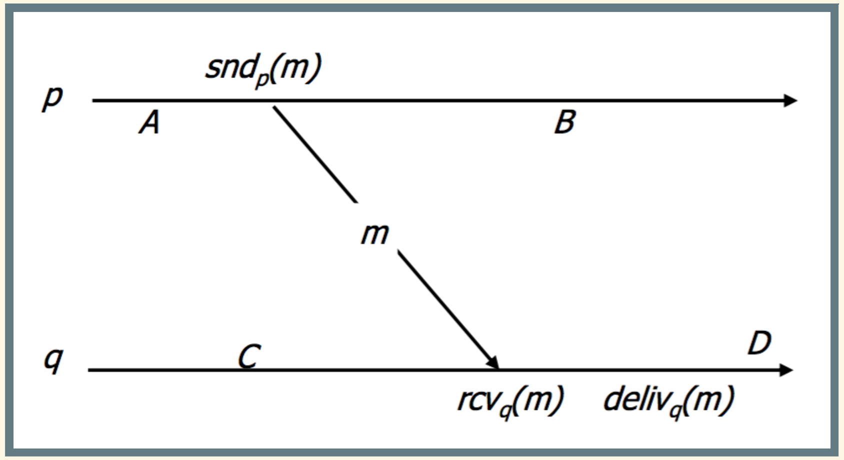
Totally-ordered multicast

Vector clocks

Happened-before relationship

Happened-before relationship

- If a and b are two events in the same process, and a comes before b, then $a \rightarrow b$.
- If a is the sending of a message, and b is the receipt of that message, then $a \rightarrow b$.
- If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$.



$$A \rightarrow^{p} B$$

$$C \rightarrow^{q} D$$

$$\mathrm{snd}_p(m) \rightarrow^{M} \mathrm{rcv}_q(m)$$

$$A \rightarrow^{p} \mathrm{snd}_p(m) \rightarrow^{M} \mathrm{rcv}_q(m) \rightarrow^{q} D$$

$$A \rightarrow D$$

Lamport clocks

Lamport clocks

Attach a timestamp $C(e)$ to each event e

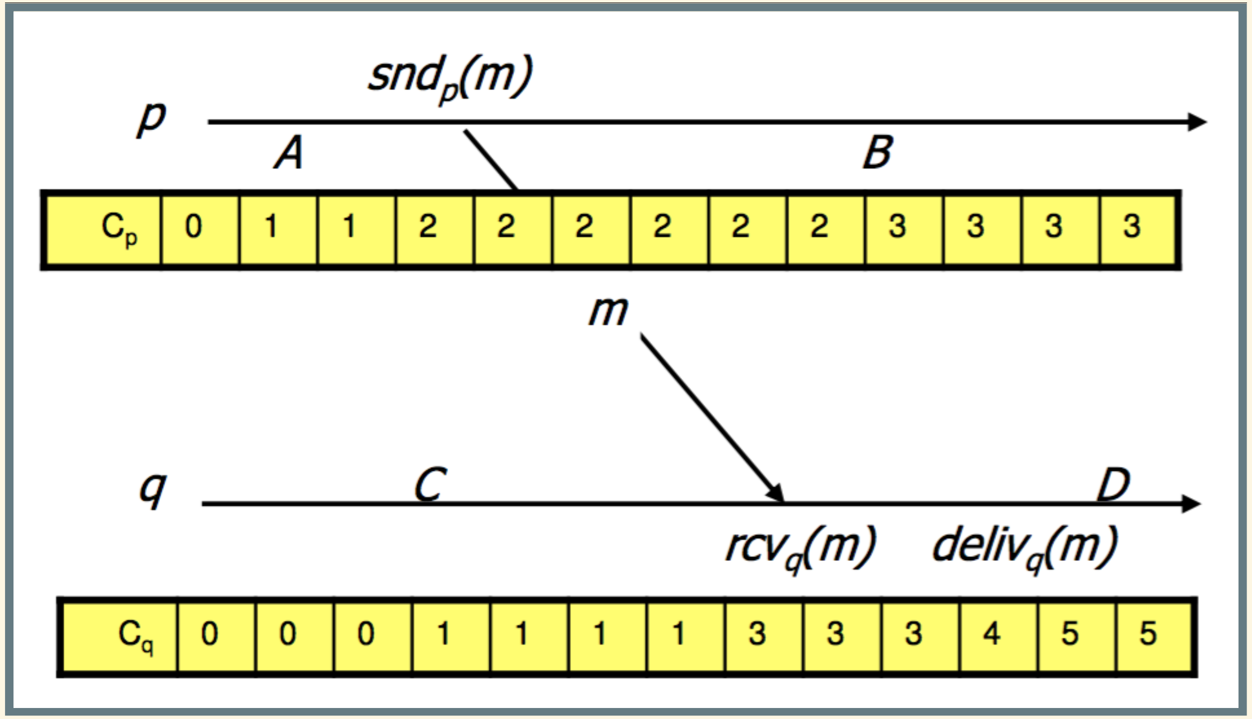
- If $A \rightarrow^p B$ then $C(A) < C(B)$
- If A corresponds to sending a message m, and B to the receipt of that message, then also $C(A) < C(B)$.

Lamport clocks

Process P_i maintains a counter C_i and adjusts this counter:

- C_i is incremented for any two successive events within P_i
- Each $snd(m)$ by process P_i, receives a timestamp $ts(m) = C_i$
- When $recv(m)$ in process P_j, then $C_j = max(C_j, ts(m))$

Lamport clocks



$snd_p(m)$
p
A
B
C_p | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3
m
q
C
D
$rcv_q(m)$
$deliv_q(m)$
C_q | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 3 | 3 | 3 | 4 | 5 | 5

So if $A \rightarrow B$ then $C(A) < C(B)$
but **not**: if $C(A) < C(B)$ then $A \rightarrow B$.

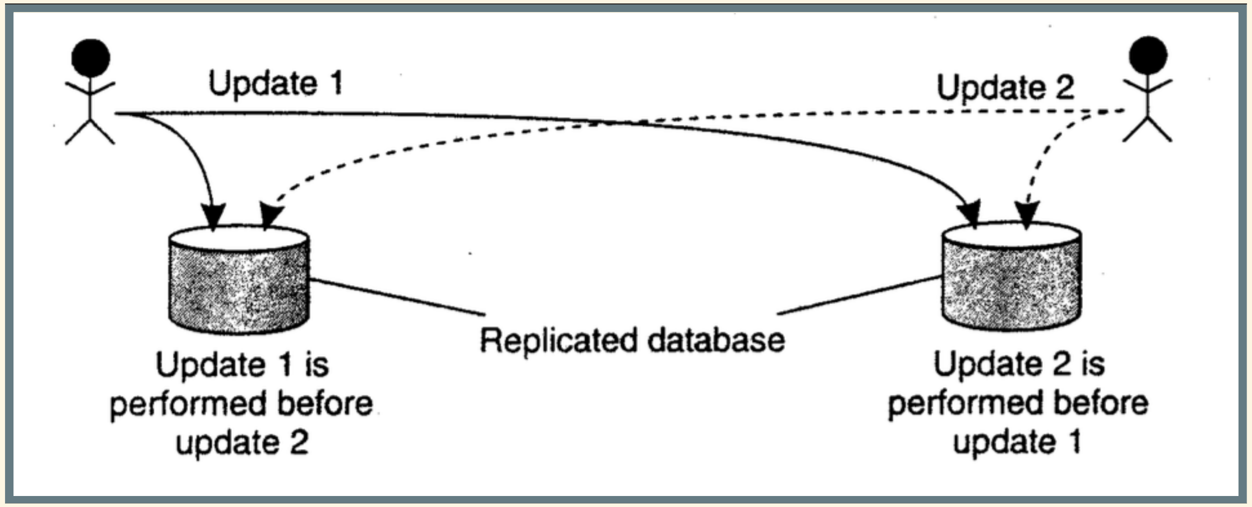
Totally-ordered multicast

Totally-ordered multicast

A guarantee that concurrent updates on a replicated database are seen in the same order everywhere:

- P_1 adds 100 to an account with 1000
- P_2 increments balance with 1%

In absence of proper synchronization:
replica #1 ← 1111, while replica #2 ← 1110



Update 1
Update 2
Replicated database
Update 1 is performed before update 2
Update 2 is performed before update 1

Totally-ordered multicast

- A multicast operation
- All messages are delivered in the same order to each receiver

Lamport's logical clocks can be used to implement totally-ordered multicasts in a completely distributed fashion.

Totally-ordered multicast

- A message has a Lamport clock value
- When receiving a message, send an acknowledge to everybody
- Order all messages according to clock value
  - If clashes, order by e.g. process ID
- Deliver message which is acknowledged by all
- Acknowledges makes the queues become identical

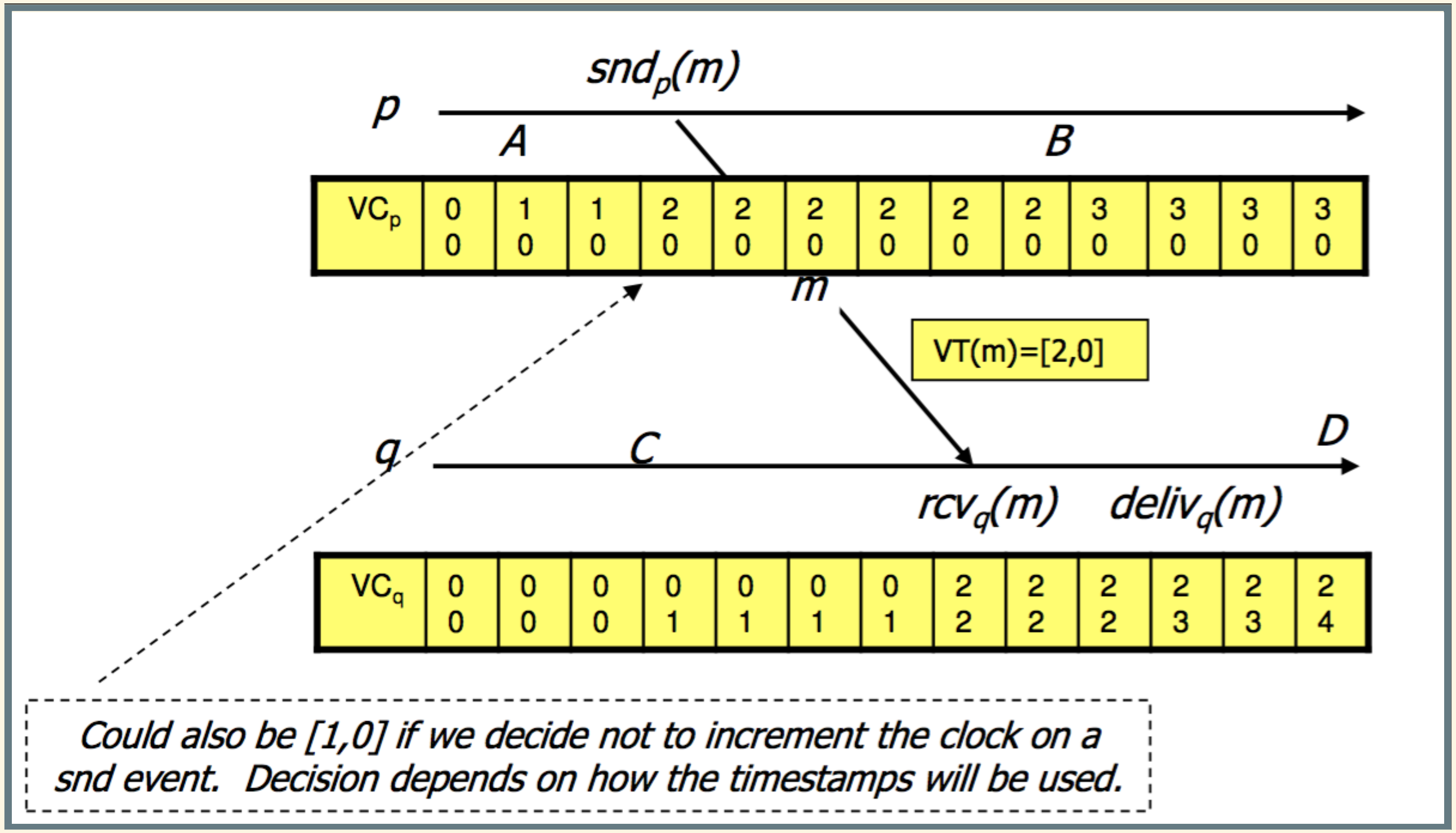
Vector clocks

Vector clocks

Extension of Logical clocks where each process has a unique vector of times for each process:
$VC(A) = [q, p]$

Sending message: $VC(m) = VC_p$
Receive message: $VC_q = max(VC_q, VC(m))$

Vector clocks



VC_p	0 0	1 0	1 0	2 0	2 0	2 0	2 0	2 0	2 0	3 0	3 0	3 0	3 0

VT(m)=[2,0]

VC_q	0 0	0 0	0 0	0 1	0 1	0 1	0 1	2 2	2 2	2 2	2 3	2 3	2 4

Could also be [1,0] if we decide not to increment the clock on a snd event. Decision depends on how the timestamps will be used.

Comparing Vector clocks

- $VC(A) \leq VC(B)$ if
  - $\forall i, VC_A[i] \leq VC_B[i]$
- $VC(A) < VC(B)$ if
  - $VC(A) \leq VC(B)$ and $VC(A) \neq VC(B)$
  - $\exists i, VC_A[i] < VC_B[i]$

Properties of Vector clocks

If $A \rightarrow B$, then $VC(A) < VC(B)$ (as Lamport clocks).

But also: if $VC(A) < VC(B)$ then $A \rightarrow B$.

This is shown by:
If $\neg(A \rightarrow B)$ then $\neg(VC(A) < VC(B))$

Vector clocks

If $\neg(A \rightarrow B)$ then $\neg(VC(A) < VC(B))$

- A occurs at p and B at q
- $VC(A)[p] = k$
- There cannot be a sequence of events $A = E_0 \rightarrow E_1 \rightarrow E_2 \rightarrow \ldots \rightarrow E_n = B$ (otherwise $A \rightarrow B$)
- Thus we must have $VC(B)[p] < k$
- Consequently $\neg(VC(A) < VC(B))$