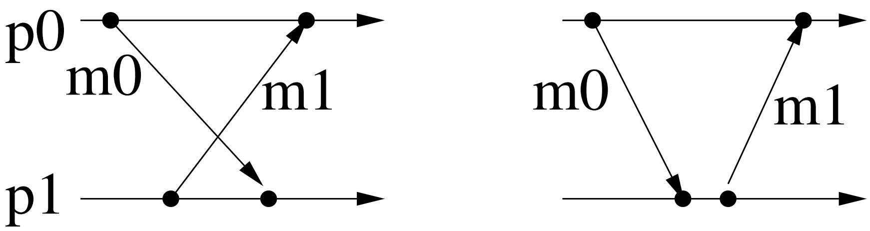
Logical Clocks Motivation

In an *asynchronous* system, often cannot tell which of two events occurred before the other.

Example A: p_0 sends message m_0 to p_1. Before m_0 arrives, p_1 sends message m_1 to p_0.
The processors cannot deduce which message was sent first, and in fact *it makes no difference.*

Example B: p_0 sends message m_0 to p_1. After m_0 arrives, p_1 sends message m_1 to p_0.
In this case it *is* possible, and perhaps important, for the processors to know that m_0 was sent, and received, before m_1 was sent.



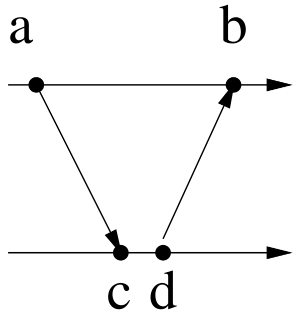
Thus instead of ordering all system events, a more natural way to order events is with a *partial order*.

Happens Before Partial Order

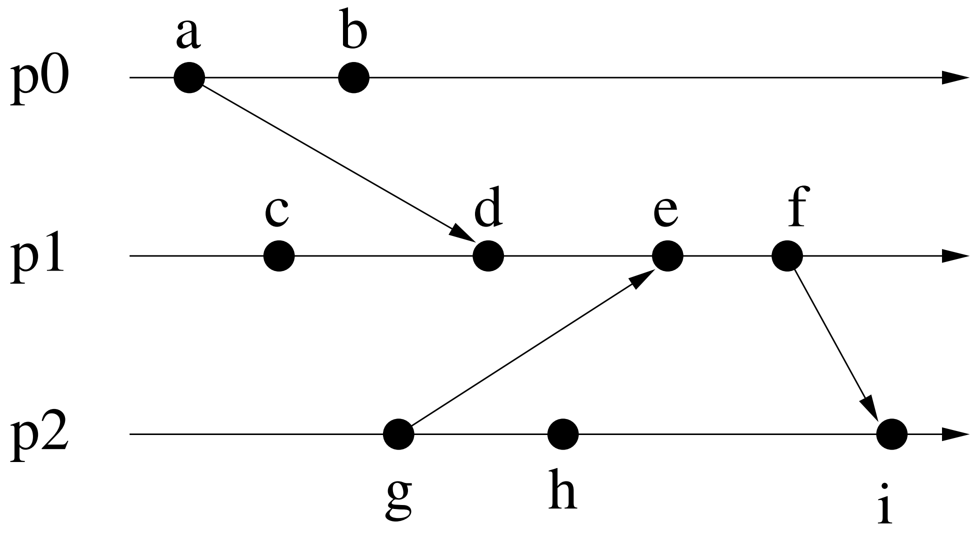
Given an execution, computation event a **happens before** computation event b, denoted $a \to b$, if

1. a and b occur at the same processor and a precedes b in the execution, or
2. a results in the sending of message m and b includes the receipt of m, or
3. there exists computation event c such that $a \to c$ and $c \to b$ (transitive closure of first two).

Happens before means that information can flow from a to affect b, i.e., that a might cause b.



Happens Before Example



Rule 1: $a \to b$, $c \to d \to e \to f$, $g \to h \to i$
Rule 2: $a \to d$, $g \to e$, $f \to i$
Rule 3: $a \to e$, $c \to i$, ...

If neither of two events happens before the other, then they are **concurrent**, denoted $||$.

$h||e$
$e||h$
etc.

Logical Clocks

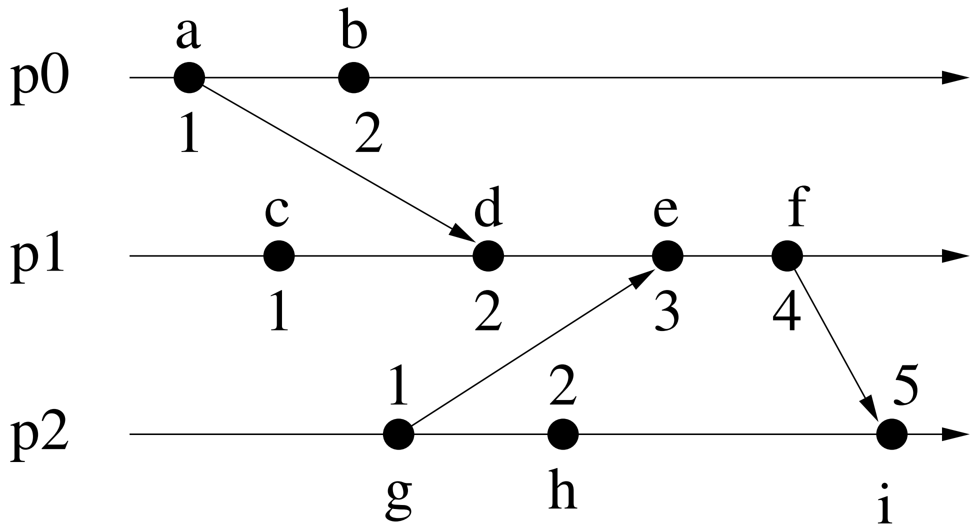
Logical clocks are values assigned to events to provide some information about the order in which events happen.

Assign an integer $L(e)$ to each computation event e in an execution such that if $a \to b$, then $L(a) < L(b)$.

Logical Timestamps Algorithm:

- Each p_i keeps a counter (logical timestamp) L_i, initialized to 0.
- Every message p_i sends is timestamped with current value of L_i.
- L_i is incremented at each step to be greater than
  - its previous value and
  - the timestamps on any message received at this step
- If a is an event at p_i, then assign $L(a)$ to be value of L_i at end of a, denoted $L_i(a)$.

Logical Timestamps Example



$a \to b$ and $L(a) = 1 < 2 = L(b)$
$f \to i$ and $L(f) = 4 < 5 = L(i)$
$a \to e$ and $L(a) = 1 < 3 = L(e)$
etc.

To get a total order, break ties with processor ids.

For example, $L(a) = (1, 0)$, $L(c) = (1, 1)$, etc.

Timestamps are ordered lexicographically.
Thus $L(a) < L(c)$.

Vector Clocks Motivation

Potential problem with logical clocks:

$a \to b$ implies $L(a) < L(b)$ but
$L(a) < L(b)$ does not necessarily imply $a \to b$.

In previous example, $L(g) = 1$ and $L(b) = 2$ but g does not happen before b.

Reason is that "happens before" is a partial order, but logical clock values are integers, which are totally ordered.

Vector clocks are a generalization of logical clocks and are implemented with values drawn from a partially ordered set instead of a totally ordered set.

Vector Clocks

Vector clocks are values that are assigned to events to provide non-causality information as well as causality information about events.

Assign a value $V(e)$, drawn from a partially ordered set, to each computation event e in an execution such that $a \to b$ iff $V(a) < V(b)$.

Vector Timestamps Algorithm:

- Each p_i keeps an n-vector V_i, initialized to $\langle 0, \ldots, 0 \rangle$.
- Every message p_i sends is timestamped with current value of V_i.
- At every step, set $V_i[i] := V_i[i] + 1$ (number of steps p_i has taken)
- When receiving a message with vector timestamp T, update V_i's components $j \neq i$:
  $V_i[j] := \max(T[j], V_i[j])$
  (p_i's estimate of how many steps p_j has taken)
- If a is an event at p_i, then assign $V(a)$ to be value of V_i at end of a, denoted $V_i(a)$.

Manipulating Vector Timestamps

Let V_1 and V_2 be two n-vectors of integers.

Equality: Define $V_1 = V_2$ iff $V_1[i] = V_2[i]$ for all i.
Example: $\langle 3, 2, 4 \rangle = \langle 3, 2, 4 \rangle$

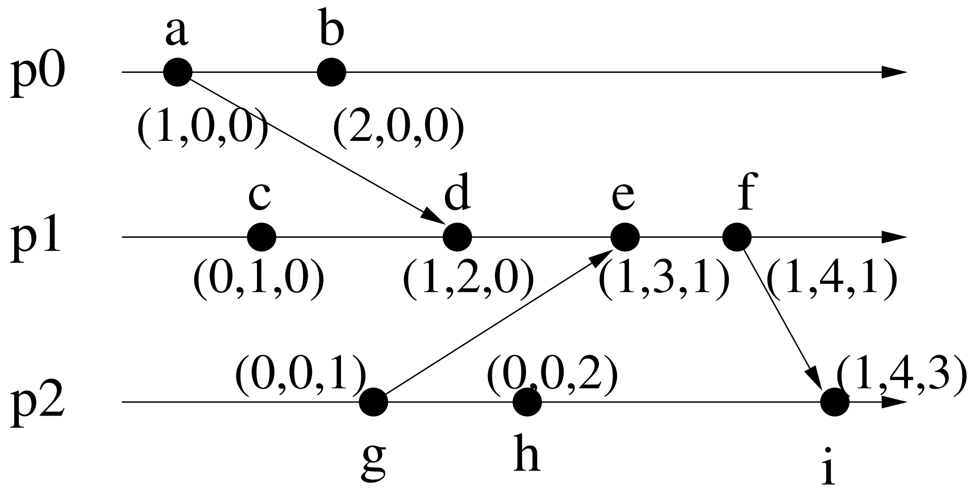
Less than or equal: Define $V_1 \leq V_2$ iff $V_1[i] \leq V_2[i]$ for all i.
Example: $\langle 2, 2, 3 \rangle \leq \langle 3, 2, 4 \rangle$ and $\langle 3, 2, 4 \rangle \leq \langle 3, 2, 4 \rangle$

Less than: Define $V_1 < V_2$ iff $V_1 \leq V_2$ but $V_1 \neq V_2$.
Example: $\langle 2, 2, 3 \rangle < \langle 3, 2, 4 \rangle$

Incomparable: $V_1 || V_2$ iff $V_1 \not\leq V_2$ and $V_2 \not\leq V_1$.
Example: $\langle 3, 2, 4 \rangle || \langle 4, 1, 4 \rangle$

Note that this is *not* the same as lexicographic ordering!

Vector Timestamps Example



$V(g) = \langle 0, 0, 1 \rangle$ and $V(b) = \langle 2, 0, 0 \rangle$, which are incomparable.

Recall that $L(g) = 1$ and $L(b) = 2$.

Correctness of Vector Timestamps

Theorem: Vector timestamps implement vector clocks.

Proof: Show $a \to b$ implies $V(a) < V(b)$.

Case 1: a and b both occur at p_i, say a before b. Since V_i increases at every step of p_i, $V(a) < V(b)$.

Case 2: a occurs at p_i and causes the sending of m, while b occurs at p_j and includes the receipt of m.

Recipient p_j updates its vector timestamp in such a way that $V(a) \leq V(b)$.

Now show $V[j](a) < V[j](b)$: p_i's estimate of the number of steps taken by p_j is never an over-estimate. Since m is not received before it is sent, p_i's estimate of the number of p_j's steps when a occurs is less than the number of steps taken by p_j when b occurs.

Thus $V(a) < V(b)$.

Correctness of Vector Timestamps (cont'd)

Case 3: There exists c such that $a \to c$ and $c \to b$. By induction (from Cases 1 and 2) and transitivity of $<$, $V(a) < V(b)$.

Conversely, show $V(a) < V(b)$ implies $a \to b$.

Equivalent to showing $a \not\to b$ implies $V(a) \not< V(b)$.

Suppose a occurs at p_i and b occurs at p_j.

Let $V[i](a) = \ell$.

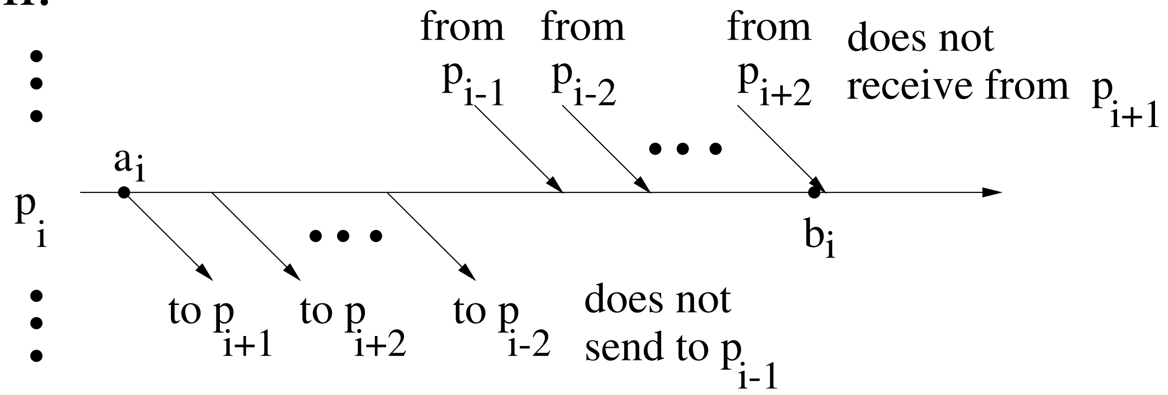
$V[i](b) < \ell$ since there is no chain of messages from p_i to p_j, originating at p_i's ℓ-th step or later.

Thus $V(a) \not< V(b)$.
$\square$

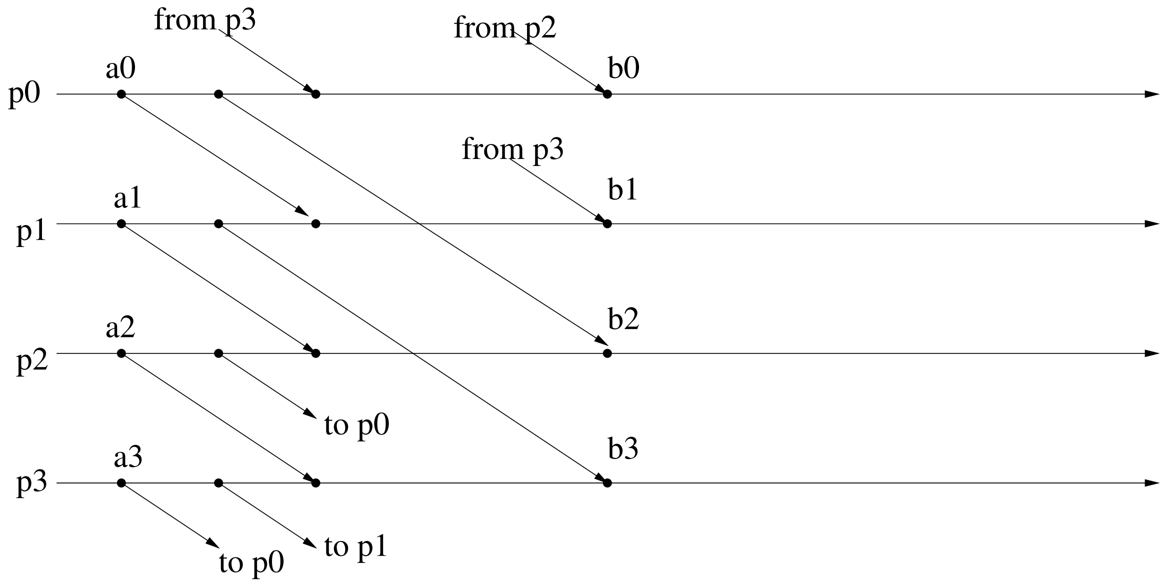
Lower Bound on Size of Vector Clocks

Vector timestamps are big — each contains n pieces of information. Can we do better? No.

Show that any implementation of vector clocks using vectors of reals requires n-vectors. Consider this execution:



For $n = 4$:



Vector Clock Size Lower Bound (cont'd)

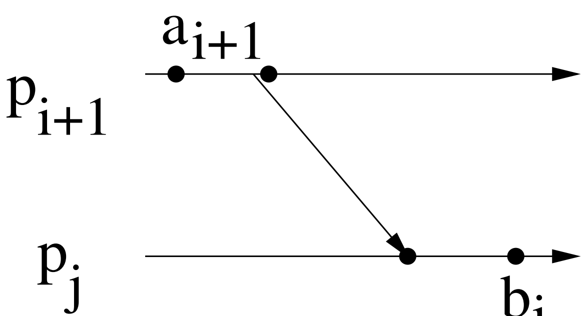
Claim 1: $a_{i+1} || b_i$, for all i (with wraparound).

Proof: Since each processor does all sends before any receives, there is no transitivity. Also, p_{i+1} does not send to p_i.

□

Claim 2: $a_{i+1} \to b_j$, for all $j \neq i$.

Proof: If $j = i + 1$, then obvious.
If $j \neq i + 1$, then p_{i+1} sends to p_j, since $j \neq i$:



□

Vector Clock Size Lower Bound (cont'd)

Theorem: If V implements vector clocks for a system of n processors using vectors in R^k, then $k \geq n$.

Proof: By Claim 1, $a_{i+1} || b_i$
$\Rightarrow V(a_{i+1})$ and $V(b_i)$ are incomparable
$\Rightarrow V(a_{i+1})$ is larger than $V(b_i)$ in some coordinate $\ell(i)$
$\Rightarrow \ell : \{0, \ldots, n-1\} \rightarrow \{0, \ldots, k\}$

To show $k \geq n$, show ℓ is 1-1. Suppose in contradiction there exist i and j such that $\ell(i) = \ell(j) = r$:
$V(a_{i+1})[r] > V(b_i)[r]$ and $V(a_{j+1})[r] > V(b_j)[r]$.

By Claim 2, $a_{i+1} \rightarrow b_j$.
$V(a_{j+1})[r] > V(b_j)[r]$ by def of r
$\geq V(a_{i+1})[r]$ by Claim 2
$> V(b_i)[r]$ by def of r.

Thus $V(a_{j+1}) \not< V(b_i)$, contradicting Claim 2 ($a_{j+1} \rightarrow b_i$) and correctness of V.
$\square$