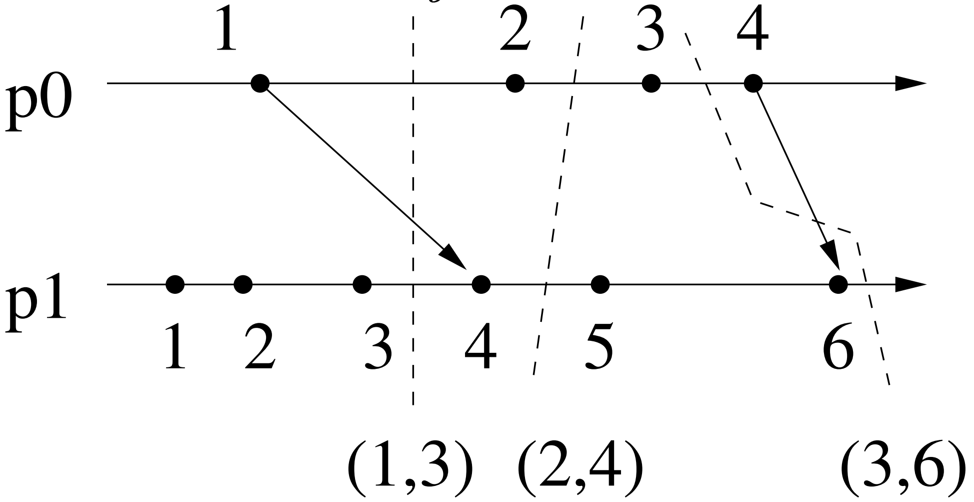
Application of Causality: Consistent Cuts

Assume FIFO message delivery and at most one message is received per computation step.

Number computation steps of each processor $1, 2, 3, \ldots$

A **cut** of the execution is $K = \langle k_0, \ldots, k_{n-1} \rangle$, where k_i indicates number of computation steps taken by p_i.

In a **consistent cut**, $(k_i + 1)$-st computation step of p_i does not happen before k_j-th computation step of p_j.



$\langle 1, 3 \rangle$ and $\langle 2, 4 \rangle$ are consistent.
$\langle 3, 6 \rangle$ is inconsistent: includes receipt of a message but not the sending. I.e., p_0's 4th step happens before p_1's 6th step.

Finding "Recent" Consistent Cut

Problem Version 1: The processors are all given a cut K and must find a maximal consistent cut that is $\leq K$.

Application: Logging-based crash recovery.

Solution 1: Append vector timestamps to application messages.
Store the vts of each computation step in an array `store`.
When p_i is given K:

```
m := K[i]
for index := m downto 1 do
   if store[index] <= K then
      return index
endfor
```

This returns the index of the most recent step whose vts is $\leq K$.

Global answer is $\langle \ldots, \mathtt{index}_i, \ldots \rangle$.

Finding "Recent" Consistent State (cont'd)

Problem Version 2: A subset of processors are told, at arbitrary starting times, to find a consistent cut that includes the state of at least one processor at its starting time. Called a *distributed snapshot*.

Solution 2: Instead of altering the application messages, add *marker* messages. Code for p_i:

```
initially answer = -1 and num = 0
when application message is received:
   num := num + 1
   perform application action
when marker is received or start:
   if answer = -1 then
      answer := num
      send marker to all neighbors
   endif
```

Global answer is $\langle \ldots, \mathtt{answer}_i, \ldots \rangle$.
Cost is $O(E)$ messages, one marker per link.

What About Channel States?

To capture the entire system state, processor states are not sufficient. The messages in transit must also be known.

Approach 1:

- Include in each entry of the `store` array the number of messages received so far from each neighbor.
- Each p_i scans `store` from earliest to latest, "replaying" the execution.
- Keep track of messages to be sent during replay.
- Stop replay when find latest m' s.t. $\texttt{store}[m']$ has vts $\leq K$.
- p_i sends to each neighbor p_j the number of messages it has received from p_j according to $\texttt{store}[m']$.
- p_j then computes messages in transit to p_i in the consistent cut as those it was supposed to send that have not yet been received.

Channel States (cont'd)

Approach 2:

- p_i records sequence of messages received from p_j between time p_i records its answer and time p_i gets marker from p_j.
- These are the messages in transit from p_j to p_i in the consistent cut.

Session Problem — Limitations on Happens Before

Happens before only captures causal dependencies that occur *inside* the system. It cannot capture relationships observed on the outside.

Example: In the **session problem**, each processor periodically performs a "special action" (e.g., increments some local variable). A **session** occurs when each processor has executed at least one special action. The goal is to indicate when at least s sessions have occurred.

1 1 0 1 0 2   2 2 0 2 0 1   0 1 2

session 1   session 2   session 3

Synchronous: Time required is s rounds, no communication is needed.

Asynchronous: Time required is greater than $(s-1)\cdot$ *diameter*. The term *diameter* is required to accomplish communication after each session.

Hardware Clocks

Suppose real time, or some approximation of it, is available to the processors through **hardware clocks**.

Processor p_i has hardware clock H_i, modeled as a function from real times to clock times.

During p_i's computation event at real time t, the value $H_i(t)$ can be used as input to its transition function.

What properties might H_i satisfy?

- increasing
- weakest: $H_i(t) =$ number of steps taken by p_i through real time t
- strongest: $H_i(t) = t$
- intermediate: $H_i(t) = t + c_i$, where c_i is a constant offset. Hardware clock runs at same rate as real time. (Consider drift later.)

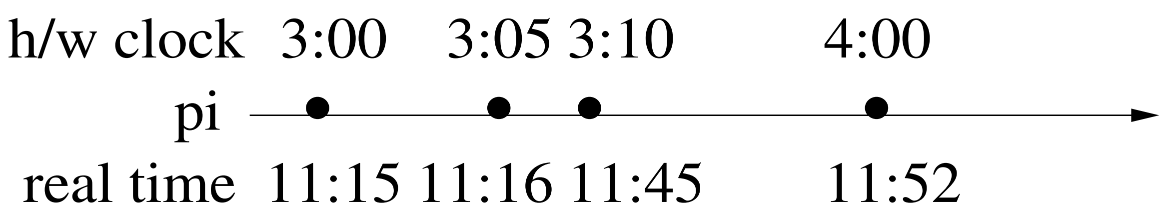
Modeling System with Hardware Clocks

Alternative representation of an execution (sequence of events) is a set of n sequences, one per processor.

A **view** of p_i is:

- initial state of p_i
- sequence of events (computation and deliver) at p_i
- hardware clock value for each event

A **timed view** of p_i is a view with a *real time* associated with each event (increasing).

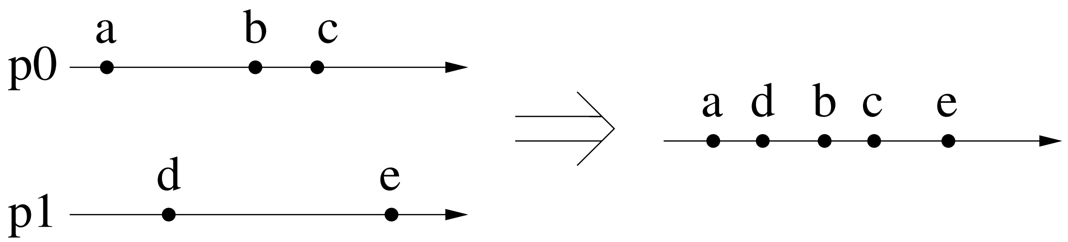


Given a timed execution α, let $\alpha|i$ be p_i's timed view extracted from α.

Modeling System with Hardware Clocks (cont'd)

How to merge n timed views into a timed execution:

- initial configuration is combination of initial states
- obtain sequence of events by interleaving events from views in real-time order (break ties with ids)
- apply events in order to initial configuration



Is the result an execution? Potential problem is delivering a message that is not in transit.

These ideas are used to prove lower bounds:

1. start with a timed execution,
2. modify processors' views, and
3. recombine the results.