Same problem, different approach

- Monitor process does not query explicitly
- Instead, it passively collects information and uses it to build an observation.
  (reactive architectures, Harel and Pnueli [1985])

An observation is an ordering of event of the distributed computation based on the order in which the receiver is notified of the events.
Observations:

a few observations

An observation puts no constraint on the order in which the monitor receives notifications

\[ p_0 \rightarrow e_1 \rightarrow p_1 \]
Observations:
a few observations

An observation puts no constraint on the order in which the monitor receives notifications

\[ p_0 \rightarrow e_1 \rightarrow p_1 \]

\[ e_1^1 \rightarrow e_1^2 \]
Observations: a few observations

An observation puts no constraint on the order in which the monitor receives notifications.
Observations: a few observations

An observation puts no constraint on the order in which the monitor receives notifications.

To obtain a run, messages must be delivered to the monitor in FIFO order.
Observations: a few observations

An observation puts no constraint on the order in which the monitor receives notifications.

To obtain a run, messages must be delivered to the monitor in FIFO order.

What about consistent runs?
Causal delivery

FIFO delivery guarantees:

\[ \text{send}_i(m) \rightarrow \text{send}_i(m') \Rightarrow \text{deliver}_j(m) \rightarrow \text{deliver}_j(m') \]
Causal delivery

FIFO delivery guarantees:

\[ \text{send}_i(m) \rightarrow \text{send}_i(m') \Rightarrow \text{deliver}_j(m) \rightarrow \text{deliver}_j(m') \]

Causal delivery generalizes FIFO:

\[ \text{send}_i(m) \rightarrow \text{send}_k(m') \Rightarrow \text{deliver}_j(m) \rightarrow \text{deliver}_j(m') \]
Causal delivery

FIFO delivery guarantees:

\[ \text{send}_i(m) \rightarrow \text{send}_i(m') \Rightarrow \text{deliver}_j(m) \rightarrow \text{deliver}_j(m') \]

Causal delivery generalizes FIFO:

\[ \text{send}_i(m) \rightarrow \text{send}_k(m') \Rightarrow \text{deliver}_j(m) \rightarrow \text{deliver}_j(m') \]
**Causal delivery**

FIFO delivery guarantees:

\[
sendi(m) \rightarrow sendi(m') \Rightarrow deliver_j(m) \rightarrow deliver_j(m')
\]

Causal delivery generalizes FIFO:

\[
sendi(m) \rightarrow send_k(m') \Rightarrow deliver_j(m) \rightarrow deliver_j(m')
\]
Causal delivery

FIFO delivery guarantees:

\[ \text{send}_i(m) \to \text{send}_i(m') \Rightarrow \text{deliver}_j(m) \to \text{deliver}_j(m') \]

Causal delivery generalizes FIFO:

\[ \text{send}_i(m) \to \text{send}_k(m') \Rightarrow \text{deliver}_j(m) \to \text{deliver}_j(m') \]
Causal delivery

FIFO delivery guarantees:

\[ \text{send}_i(m) \rightarrow \text{send}_i(m') \Rightarrow \text{deliver}_j(m) \rightarrow \text{deliver}_j(m') \]

Causal delivery generalizes FIFO:

\[ \text{send}_i(m) \rightarrow \text{send}_k(m') \Rightarrow \text{deliver}_j(m) \rightarrow \text{deliver}_j(m') \]
Causal delivery

FIFO delivery guarantees:
\[ \text{send}_i(m) \rightarrow \text{send}_i(m') \Rightarrow \text{deliver}_j(m) \rightarrow \text{deliver}_j(m') \]

Causal delivery generalizes FIFO:
\[ \text{send}_i(m) \rightarrow \text{send}_k(m') \Rightarrow \text{deliver}_j(m) \rightarrow \text{deliver}_j(m') \]
Causal Delivery in Synchronous Systems

We use the upper bound $\Delta$ on message delivery time.
Causal Delivery in Synchronous Systems

We use the upper bound $\Delta$ on message delivery time

**DR1:** At time $t$, $p_0$ delivers all messages it received with timestamp up to $t - \Delta$ in increasing timestamp order
Causal Delivery with Lamport Clocks

**DR1.1:** Deliver all received messages in increasing (logical clock) timestamp order.
Causal Delivery with Lamport Clocks

DR1.1: Deliver all received messages in increasing (logical clock) timestamp order.
**Causal Delivery with Lamport Clocks**

**DR1.1:** Deliver all received messages in increasing (logical clock) timestamp order.

Should $p_0$ deliver?
Causal Delivery with Lamport Clocks

**DR1.1:** Deliver all received messages in increasing (logical clock) timestamp order.

![Diagram](image)

**Problem:** Lamport Clocks don’t provide **gap detection**

Given two events $e$ and $e'$ and their clock values $LC(e)$ and $LC(e')$—where $LC(e) < LC(e')$—determine whether some event $e''$ exists s.t.

$$LC(e) < LC(e'') < LC(e')$$
Stability

**DR2**: Deliver all received stable messages in increasing (logical clock) timestamp order.

A message \( m \) received by \( p \) is stable at \( p \) if \( p \) will never receive a future message \( m' \) s.t.

\[
TS(m') < TS(m)
\]
Implementing Stability

- Real-time clocks
- Wait for $\Delta$ time units
Implementing Stability

- Real-time clocks
  - wait for $\Delta$ time units

- Lamport clocks
  - wait on each channel for $m$ s.t. $TS(m) > LC(e)$

- Design better clocks!
Clocks and STRONG Clocks

- Lamport clocks implement the clock condition:
  \[ e \rightarrow e' \Rightarrow LC(e) < LC(e') \]

- We want new clocks that implement the strong clock condition:
  \[ e \rightarrow e' \equiv SC(e) < SC(e') \]
Causal Histories

The causal history of an event $e$ in $(H, \rightarrow)$ is the set

$$\theta(e) = \{ e' \in H \mid e' \rightarrow e \} \cup \{ e \}$$
Causal Histories

The causal history of an event \( e \) in \((H, \rightarrow)\) is the set

\[
\theta(e) = \{ e' \in H \mid e' \rightarrow e \} \cup \{ e \}
\]
The causal history of an event $e$ in $(H, \rightarrow)$ is the set

$\theta(e) = \{ e' \in H \mid e' \rightarrow e \} \cup \{ e \}$

$e \rightarrow e' \equiv \theta(e) \subset \theta(e')$
How to build $\theta(e)$

Each process $p_i$:

- initializes $\theta : \theta := \emptyset$

- if $e^k_i$ is an internal or send event, then
  $$\theta(e^k_i) := \{e^k_i\} \cup \theta(e^{k-1}_i)$$

- if $e^k_i$ is a receive event for message $m$, then
  $$\theta(e^k_i) := \{e^k_i\} \cup \theta(e^{k-1}_i) \cup \theta(\text{send}(m))$$
Pruning causal histories

- Prune segments of history that are known to all processes (Peterson, Bucholz and Schlichting)
- Use a more clever way to encode $\theta(e)$
Vector Clocks

Consider $\theta_i(e)$, the projection of $\theta(e)$ on $p_i$.

$\theta_i(e)$ is a prefix of $h^i$: $\theta_i(e) = h^i_{k_i}$ – it can be encoded using $k_i$.

$\theta(e) = \theta_1(e) \cup \theta_2(e) \cup \ldots \cup \theta_n(e)$ can be encoded using $k_1, k_2, \ldots, k_n$.

Represent $\theta$ using an $n$-vector $VC$ such that $VC(e)[i] = k \iff \theta_i(e) = h^i_{k_i}$.
Message $m$ is timestamped with $TS(m) = VC(send(m))$

\[ VC(e_i) := max(VC, TS(m)) \]

\[ VC(e_i)[i] := VC[i] + 1 \]
Example

$p_1$ [1,0,0] [2,1,0] [3,1,2] [4,1,2] [5,1,2]

$p_2$ [0,1,0] [1,2,3] [4,3,3]

$p_3$ [1,0,1] [1,0,2] [1,0,3] [5,1,4]
Operational interpretation

\[ VC(e_i)[i] = \]
\[ VC(e_i)[j] = \]
Operational interpretation

$VC(e_i)[i] = \text{no. of events executed by } p_i \text{ up to and including } e_i$

$VC(e_i)[j] = $
Operational interpretation

$\text{Operational interpretation}$

$VC(e_i)[i] = \text{no. of events executed by } p_i \text{ up to and including } e_i$

$VC(e_i)[j] = \text{no. of events executed by } p_j \text{ that happen before } e_i \text{ of } p_i$
**VC properties: event ordering**

Given two vectors $V$ and $V'$, less than is defined as:

$$V < V' \equiv (V \neq V') \land (\forall k : 1 \leq k \leq n : V[k] \leq V'[k])$$

- **Strong Clock Condition:** $e \rightarrow e' \equiv VC(e) \leq VC(e')$

- **Simple Strong Clock Condition:**
  
  Given $e_i$ of $p_i$ and $e_j$ of $p_j$, where $i \neq j$
  
  $$e_i \rightarrow e_j \equiv VC(e_i)[i] \leq VC(e_j)[i]$$

- **Concurrency**
  
  Given $e_i$ of $p_i$ and $e_j$ of $p_j$, where $i \neq j$
  
  $$e_i \parallel e_j \equiv (VC(e_i)[i] > VC(e_j)[i]) \land (VC(e_j)[j] > VC(e_i)[j])$$
VC properties: consistency

Pairwise inconsistency
Events $e_i$ of $p_i$ and $e_j$ of $p_j$ ($i \neq j$) are pairwise inconsistent (i.e., can’t be on the frontier of the same consistent cut) if and only if

$$\forall i, j : 1 \leq i, j \leq n : (VC(e_i^c)[i] \geq VC(e_j^c)[i])$$

Consistent Cut
A cut defined by $(c_1, \ldots, c_n)$ is consistent if and only if

$$\forall i, j : 1 \leq i, j \leq n : (VC(e_i^{c_i})[i] \geq VC(e_j^{c_j})[i])$$
VC properties: weak gap detection

Weak gap detection
Given $e_i$ of $p_i$ and $e_j$ of $p_j$, if $VC(e_i)[k] < VC(e_j)[k]$ for some $k \neq j$, then there exists $e_k$ s.t

$$- (e_k \rightarrow e_i) \land (e_k \rightarrow e_j)$$

- $p_i$ [2,0,1]
- $p_j$ [2,2,2]
- $p_k$ [0,0,2]
VC properties: weak gap detection

Weak gap detection
Given $e_i$ of $p_i$ and $e_j$ of $p_j$, if $VC(e_i)[k] < VC(e_j)[k]$ for some $k \neq j$, then there exists $e_k$ s.t

$$\neg(e_k \rightarrow e_i) \land (e_k \rightarrow e_j)$$
VC properties: strong gap detection

Weak gap detection
Given $e_i$ of $p_i$ and $e_j$ of $p_j$, if $VC(e_i)[k] < VC(e_j)[k]$ for some $k \neq j$, then there exists $e_k$ s.t.

$$-(e_k \rightarrow e_i) \land (e_k \rightarrow e_j)$$

Strong gap detection
Given $e_i$ of $p_i$ and $e_j$ of $p_j$, if $VC(e_i)[i] < VC(e_j)[i]$ then there exists $e'_i$ s.t.

$$(e_i \rightarrow e'_i) \land (e'_i \rightarrow e_j)$$
VCs for Causal Delivery

- Each process increments the local component of its $VC$ only for events that are notified to the monitor.
- Each message notifying event $e$ is timestamped with $VC(e)$.
- The monitor keeps all notification messages in a set $M$. 