Many distributed systems problems require evaluating a global systems property. Evaluating the truth of a global property may require construction of a CONSISTENT global system state.

Objective of this lecture:
- Formally define basic concepts:
  - Distributed computations
  - (Consistent) global system state, cut, run
- Examine different approaches for constructing a global system state:
  - Passive monitoring of a computation
  - Active construction of a global snapshot
- Study Chandy & Lamport’s distributed snapshot algorithm
- Reading list: Mullender Chapter 4

Global Detection of Properties

- Many distributed systems problems can be formulated as the problem of checking whether a global system state satisfies a property $\theta$:
  - deadlock detection
  - token loss
  - termination detection
  - checkpointing and restart
  - debugging and monitoring

- To detect a deadlock, evaluate the predicate:
  $\theta = (a$ cycle in the $WFG^+)$

- How to construct a global system state without freezing the computation?
  
  distributed computation $\Rightarrow$ global state $\Rightarrow$ consistent gs $\Rightarrow$ snapshot algorithms
What is a distributed computation?

System Model:
- A collection of sequential processes $p_1, p_2, \ldots, p_n$ and a network of unidirectional channels between pairs of processes.
- Each channel is reliable and not necessarily FIFO.
- Strongly connect network.
- Asynchronous system, neither synchronized clocks, nor a bound on message delay.

Definitions: (Figure 1)
- Execution of a sequential process is modeled by a sequence of events.
- An event is either a local event, a send event or a receive event.
- Local History of a process $p_i$ is a sequence of events corresponding to the total order imposed by the execution: $h_i = e_i^1, e_i^2, \cdots$
- Initial Prefix of a history containing the first $k$ events is denoted by $h_i^k = e_i^1, e_i^2, \cdots, e_i^k$
Global History of a computation is defined to be the set

\[ H = h_1 \cup h_2 \cup \cdots \cup h_n \]

Asynchronous system implies that no global time frame exists and the notion of 'cause and effect' imposes an order on pairs of events.

A pair of local events are ordered; a pair of send(m) and receive(m) are also ordered.

The happens before relation, denoted by \( \rightarrow \), is defined over events such that:

1. If \( e^k_i, e^l_j \in h_k \) and \( k < l \), then \( e^k_i \rightarrow e^l_j \).
2. If \( e_i = \text{send}(m) \) and \( e_j = \text{receive}(m) \), then \( e_i \rightarrow e_j \).
3. If \( e \rightarrow e' \) and \( e' \rightarrow e'' \), then \( e \rightarrow e'' \).

A pair of events are concurrent if \( e \not\rightarrow e' \) and \( e' \not\rightarrow e \), denoted by \( e || e' \).

Distributed Computation: is a partially ordered set defined by the pair \((H, \rightarrow)\).

Initial State: of process \( p_i \) denoted by \( \sigma^0_i \).

Local State: of process \( p_i \) after the \( k \)th event is denoted by \( \sigma^k_i \).
• **Global State:** of a distributed computation is a n-tuple of local states, \( \Sigma = (\sigma_1, \ldots, \sigma_n) \).

• **Cut:** of a distributed computation is a subset \( C \) of its global history \( H \) and contains an initial prefix of each of the local histories: (Figure 2)
  \[
  C = H_i^1, \ldots, H_i^n
  \]

• **Frontier of Cut:** is the last set of events for each process.

• **Run:** of a distributed computation is a total ordering \( R \) that includes all of the events in the global history and that is consistent with the local history.

  A run is a sequence of global system states.

  A distributed computation may have many runs.

---

**Solution 1 - Active Monitor**

Monitor process \( p_6 \) is an active monitor:

- Send 'state request' message to all processes
- Each process replies with its local state \( \sigma_i \)
- Construct global state \( (\sigma_1, \sigma_2, \ldots, \sigma_n) \)
Will this algorithm work? (figure 2)

The deadlock detected by the cut labeled $C'$ is...
The local states sent to the monitor: $\sigma_1^3, \sigma_2^2,$ and $\sigma_3^3$.

All cuts are not consistent!

Consistent ...

- Cut is consistent if for all events $e$ and $e'$

  $$ (e \in C \text{ and } e' \rightarrow e) \Rightarrow e' \in C $$

- Consistent global state corresponds to a consistent cut.

- Consistent run is one such that for all events $e \rightarrow e'$ implies that $e$ appears before $e'$

- A consistent run is a sequence of consistent global states.
Process $p_\parallel$ is a passive monitor:

Each process sends its events to the monitor. $p_\parallel$ constructs a consistent global state as events arrive.
Global Timestamps

Solution 2:
- FIFO delivery
- Synchronous model with global clock and a bound $\delta$ on delivery
- Each event $e$ is timestamped by the global clock $RC(e)$.

**Delivery Rule:** at time $t$, deliver all messages with timestamp up to $t - \delta$ in the increasing order.

Why this works?
- At time $t$, all messages sent before time $t - \delta$ have arrived to the monitor.
- The following property is true because of the global clock:
  \[
  e \rightarrow e' \Rightarrow RC(e) < RC(e')
  \]

Logical Timestamps

Solution 3:
- Relax global clock requirement
- Timestamp message $e$ with a logical clock $LC(e)$
- When a new event $e$ occurs,

\[
LC(e) = \begin{cases} 
  LC + 1 & \text{if internal or send event} \\
  \max \{LC, TS(m)\} + 1 & \text{if } e = \text{receive}(m)
\end{cases}
\]

such that $TS(m)$ is the timestamp of $m$ from the sender.
Why it works?
- Clock values are increasing with respect to causal precedence.
  \[ e \rightarrow e' \Rightarrow LC(e) < LC(e') \]

What about liveness property?
A message to the monitor may not be delivered since assumptions on bound on delay and clock to measure it are removed!

**Delivery Rule:** Condition for message delivery at \( p_0 \)
1. Deliver messages in FIFO, and
2. Deliver messages in increasing timestamp and
3. Deliver a message \( m \) after at least one message from each other process with a timestamp greater than or equal to \( TS(m) \) arrives to \( p_0 \).

---

**Solution 4**

- Definition: (Causal Delivery)
  \[ send_i(m) \rightarrow send_j(m') \Rightarrow deliver_k(m) \rightarrow deliver_k(m') \]

- Example:

---

- **Delivery Rule:** Deliver messages to \( p_0 \) in causal order.
- How to construct causal delivery?
  Read section 4.10 and 4.11;
  Read paper by Birman, Shiper and Stephenson 1991.
Distributed Snapshot

Solution 5:

- Goal: construct a consistent global state in a distributed fashion.
- Algorithm first proposed by Chandy & Lamport 1985.
- Assume FIFO delivery in a strongly connected network.
- Protocol on p. 80 Mullender.

$\sigma_i$: local state of process $p_i$.
$X_{i,j}$: channel state between process $p_i$ and $p_j$.
$IN_i$: set of all processes with output channels to $p_i$.
$OUT_i$: set of all processes with input channels to $p_i$. 