Distributed Systems

CS425/ECE428

Instructor: Radhika Mittal
Logistics Related

• VM clusters have been assigned!

• Newly registered students:
  • Please make sure you have access to Campuswire and Gradescope
  • If you are in 4 credits, make sure you have been allocated a VM cluster for the MPs.
  • Email Sarthak (netid: sm106) to get the required access.

• Please say your name before speaking up in class 😊
Recap: Logical timestamps

- How to reason about ordering of events across processes without synchronized clocks?

- Happened-before Relationship

- Lamport Logical Clock

- Vector Clock
Today’s agenda

• Global State
  • Chapter 14.5
  • Goal: reason about how to capture the state across all processes of a distributed system without requiring time synchronization.
Process, state, events

• Consider a system with $n$ processes: $<p_1, p_2, p_3, \ldots, p_n>$.

• Each process $p_i$ is associated with state $s_i$.
  • State includes values of all local variables, affected files, etc.

• Each channel can also be associated with a state.
  • Which messages are currently pending on the channel.
  • Can be computed from process’ state:
    • Record when a process sends and receives messages.
    • if $p_i$ sends a message that $p_j$ has not yet received, it is pending on the channel.

• State of a process (or a channel) gets transformed when an event occurs. 3 types of events:
  • local computation, sending a message, receiving a message.
Capturing a global snapshot

• Useful to capture a global snapshot of the system:
  • Checkpointing the system state.
  • Reasoning about unreferenced objects (for garbage collection).
  • Deadlock detection.
  • Distributed debugging.
Capturing a global snapshot

• Global state or global snapshot is state of each process (and each channel) in the system at a given *instant of time*.

• Difficult to capture a global snapshot of the system.

• Strawman:
  • Each process records its state at 2:05pm.
  • We get the global state of the system at 2:05pm.
    • *But precise clock synchronization is difficult to achieve.*

• How do we capture global snapshots without precise time synchronization across processes?
Some more notations and definitions

• For a process $p_i$, where events $e_i^0, e_i^1, \ldots$ occur:

  $\text{history}(p_i) = h_i = <e_i^0, e_i^1, \ldots>$

  $\text{prefix history}(p_i^k) = h_i^k = <e_i^0, e_i^1, \ldots, e_i^k>$

  $s_i^k : p_i$’s state immediately after $k^{th}$ event.

• For a set of processes $<p_1, p_2, p_3, \ldots, p_n>$:

  $\text{global history}: H = \bigcup_i (h_i)$

  a cut $C \subseteq H = h_1^c_1 \cup h_2^c_2 \cup \ldots \cup h_n^c_n$

  the frontier of $C = \{e_i^c_i, i = 1,2, \ldots n\}$

  $\text{global state} S$ that corresponds to cut $C = \bigcup_i (s_i^c_i)$
Consistent cuts and snapshots

• A cut $C$ is consistent if and only if
  \[ \forall e \in C \text{ (if } f \rightarrow e \text{ then } f \in C) \]

• A global state $S$ is consistent if and only if it corresponds to a consistent cut.
How to capture global state?

• Ideally: state of each process (and each channel) in the system at a given instant of time.
  • Difficult to capture -- requires precisely synchronized time.

• Relax the problem: find a consistent global state.
  • For a system with n processes $<p_1, p_2, p_3, \ldots, p_n>$, capture the state of the system after the $c_i^{th}$ event at process $p_i$.
    • State corresponding to the cut defined by frontier events $\{e_i^{c_i}, \text{for } i = 1,2, \ldots n\}$.
  • We want the state to be consistent.
    • Must correspond to a consistent cut.

How to find a consistent global state that corresponds to a consistent cut?
Chandy-Lamport Algorithm

• Goal:
  • Record a global snapshot
    • Process state (and channel state) for a set of processes.
    • The recorded global state is consistent.
  • Identifies a consistent cut.
  • Records corresponding state locally at each process.
Chandy-Lamport Algorithm

• **System model and assumptions:**
  - System of \( n \) processes: \(<p_1, p_2, p_3, \ldots, p_n>\).
  - There are two uni-directional communication channels between each ordered process pair: \( p_j \) to \( p_i \) and \( p_i \) to \( p_j \).
  - Communication channels are FIFO-ordered (first in first out).
    - if \( p_i \) sends \( m \) before \( m' \) to \( p_j \), then \( p_j \) receives \( m \) before \( m' \).
  - All messages arrive intact, and are not duplicated.
  - No failures: neither channel nor processes fail.
Chandy-Lamport Algorithm

• Requirements:
  • Snapshot should not interfere with normal application actions, and it should not require application to stop sending messages.
  • Any process may initiate algorithm.
Chandy-Lamport Algorithm Intuition

• First, initiator $p_i$:
  • records its own state.
  • creates a special marker message.
  • sends the marker to all other processes.

• When a process receives a marker:
  • records its own state.
Chandy-Lamport Algorithm Intuition

• First, initiator $p_i$:
  • records its own state.
  • creates a special marker message.
  • sends the marker to all other process.

• When a process receives a marker.
  • records its own state.
Chandy-Lamport Algorithm Intuition

Cut frontier: \{e_1^2, e_2^2\}
Chandy-Lamport Algorithm Intuition

• First, initiator $p_i$:
  • records its own state.
  • creates a special marker message.
  • sends the marker to all other process.

• When a process receives a marker.
  • records its own state.

This captures the local state at each process.
How do we ensure the state is consistent?
What about the channel state?
Chandy-Lamport Algorithm Intuition

• First, initiator $p_i$:
  • records its own state.
  • creates a special marker message.
  • sends the marker to all other process.

• When a process receives a marker:
  • If marker is received for the first time.
    • records its own state.
    • sends marker on all other channels.

  Leads to a consistent cut (we’ll get back to it)
  What about the channel state?
Chandy-Lamport Algorithm Intuition

Cut frontier: \{e_1^2, e_2^2\}
Chandy-Lamport Algorithm Intuition

Physical time

Cut frontier: \{e_1^2, e_2^2\}
Chandy-Lamport Algorithm Intuition

• First, initiator $p_i$:
  • records its own state.
  • creates a special marker message.
  • sends the marker to all other process.
  • start recording messages received on other channels.
    • until a marker is received on a channel.

• When a process receives a marker.
  • If marker is received for the first time.
    • records its own state.
    • sends marker on all other channels.
    • start recording messages received on other channels.
      • until a marker is received on a channel.
Chandy-Lamport Algorithm

• First, initiator $p_i$:
  • records its own state.
  • creates a special **marker** message.
  • for $j = 1 \text{ to } n$ except $i$
    • $p_i$ sends a **marker** message on outgoing channel $c_{ij}$
    • starts recording the incoming messages on each of the incoming channels at $p_i : c_{ji}$ (for $j = 1 \text{ to } n$ except $i$).
Chandy-Lamport Algorithm

Whenever a process \( p_i \) receives a marker message on an incoming channel \( c_{ki} \)

- if (this is the first marker \( p_i \) is seeing)
  - \( p_i \) records its own state first
  - marks the state of channel \( c_{ki} \) as “empty”
  - for \( j=1 \) to \( n \) except \( i \)
    - \( p_i \) sends out a marker message on outgoing channel \( c_{ij} \)
    - starts recording the incoming messages on each of the incoming channels at \( p_i : c_{ji} \) (for \( j=1 \) to \( n \) except \( i \) and \( k \)).
- else // already seen a marker message
  - mark the state of channel \( c_{ki} \) as all the messages that have arrived on it since recording was turned on for \( c_{ki} \).
Chandy-Lamport Algorithm

The algorithm terminates when

- All processes have received a marker
  - To record their own state
- All processes have received a marker on all the \((n-1)\) incoming channels
  - To record the state of all channels
Example

A      B                                  C                   D        E

E             F                          G

H                                I                                          J

Message

Instruction or Step

Message
Example

p₁ is initiator:
- Record local state s₁,
- Send out markers
- Start recording on channels c₂₁, c₃₁

Diagram:

P1

A   B   C   D   E

P2

E   F   G

P3

H   I   J

Time
First marker!
- Record own state as $s_3$
- Mark $c_{13}$ state as empty
- Start recording on other incoming $c_{23}$
- Send out markers
Example

$P1$  
A  B  C  D  E

$P2$  
E  F  G

$P3$  
H  I  J

$s_1$, Record $c_{21}, c_{31}$

$s_3$  
$c_{13} = < >$

Record $c_{23}$
Example

Duplicate marker!
State of channel $c_{31} = \langle \rangle$

$s_1$, Record $c_{21}, c_{31}$

$s_3$
$c_{13} = \langle \rangle$
Record $c_{23}$
Example

Record $c_{21}$, $c_{31}$

$\mathbf{s}_1$, Record $c_{21}$, $c_{31}$

$\mathbf{c}_{31} = \langle \rangle$

First marker

Record own state as $\mathbf{s}_2$

Mark $\mathbf{c}_{32}$ state as empty

Turn on recording on $\mathbf{c}_{12}$

Send out markers

$\mathbf{s}_3$

$\mathbf{c}_{13} = \langle \rangle$

Record $\mathbf{c}_{23}$
Example

- $s_1$, Record $c_{21}, c_{31}$
- $c_{31} = < >$
- $s_2$
- $c_{32} = < >$
- Record $c_{12}$

- $s_3$
- $c_{13} = < >$
- Record $c_{23}$
Example

\[ s_1, \text{Record } c_{21}, c_{34} \]
\[ c_{31} = < > \]

\[ s_2, c_{32} = < >, \text{Record } c_{42} \]

\[ s_3, c_{13} = < >, \text{Record } c_{23} \]

Duplicate!
Example

- Duplicate!
- $c_{21} = \langle \text{message G to D} \rangle$
- $c_{31} = \langle \rangle$
- $s_3 = \langle \rangle$
- $c_{13} = \langle \rangle$
- Record $c_{23}$
- $s_2 = \langle \rangle$
- $c_{32} = \langle \rangle$
- $c_{12} = \langle \rangle$
Example

- $s_1$: Record $e_{24}, e_{34}$
- $c_{31} = < >$
- $c_{21} = \langle \text{message G to D} \rangle$
- $c_{12} = < >$
- $c_{23} = < >$
- Duplicate!
Example

Algorithm has terminated!

• Duplicate!

• $c_{23} = <>$

$s_1, \text{Record } e_{21}, e_{31}$

$c_{21} = \langle \text{message G to D} \rangle$

$s_3$

$c_{13} = <>$

$e_{23}$

$e_{12}$

$e_{32}$

$(<>)$
Example

Frontier for the resulting cut: \{B, G, H\}

Channel state for the cut: Only $c_{21}$ has a pending message.
Example

Global snapshots pieces can be collected at a central location.
Chandy-Lamport Algorithm: Properties

- Any run of the Chandy-Lamport Global Snapshot algorithm creates a consistent cut.
- Let $e_i$ and $e_j$ be events occurring at $p_i$ and $p_j$, respectively such that
  
  - $e_i \rightarrow e_j$ (\(e_i\) happens before \(e_j\))
  
  - The snapshot algorithm ensures that if $e_j$ is in the cut then $e_i$ is also in the cut.

  That is: if $e_j \rightarrow <p_j\text{ records its state}>$, then it must be true that $e_i \rightarrow <p_i\text{ records its state}>.$
Chandy-Lamport Algorithm: Properties

• If $e_j \rightarrow <p_j$ records its state>, then it must be true that $e_i \rightarrow <p_i$ records its state>.

• By contradiction, suppose $e_j \rightarrow <p_j$ records its state>, and $<p_i$ records its state> $\rightarrow e_i$. 

\[ p_i \]
\[ e_i \]
\[ p_k \]
\[ p_j \]
\[ Time \]
\[ e_j \]
Chandy-Lamport Algorithm: Properties

• If $e_j \rightarrow <p_j$ records its state>, then it must be true that $e_i \rightarrow <p_i$ records its state$>.$

• By contradiction, suppose $e_j \rightarrow <p_j$ records its state$>$, and $<p_i$ records its state$> \Rightarrow e_i.$
Chandy-Lamport Algorithm: Properties

- If \( e_j \rightarrow <p_j \text{ records its state}> \), then it must be true that \( e_i \rightarrow <p_i \text{ records its state}> \).
- By contradiction, suppose \( e_j \rightarrow <p_j \text{ records its state}> \), and \( <p_i \text{ records its state}> \rightarrow e_i \).

\[
\begin{align*}
\text{Time} & \\
p_i & \quad e_i \\
p_k & \quad \text{must reach } p_k \text{ before } m \quad \text{due to FIFO order:} \\
p_j & \quad e_j
\end{align*}
\]
Chandy-Lamport Algorithm: Properties

• If $e_j \rightarrow <p_j$ records its state>, then it must be true that $e_i \rightarrow <p_i$ records its state$>$.  

• By contradiction, suppose $e_j \rightarrow <p_j$ records its state$>$, and $<p_i$ records its state$> \rightarrow e_i$. 

\[ \begin{align*} 
\text{Time} \\
\text{p}_i \\
\text{p}_k \\
\text{p}_j \\
e_j \\
e_i \\
\text{m'} \\
m \\
\text{must reach } \text{p}_j \text{ before } \text{m'} \\
de \text{ due to FIFO order.} 
\end{align*} \]
Chandy-Lamport Algorithm: Properties

• If $e_j \rightarrow <p_j \text{ records its state}>$, then it must be true that $e_i \rightarrow <p_i \text{ records its state}>$.

• By contradiction, suppose $e_j \rightarrow <p_j \text{ records its state}>$, and $<p_i \text{ records its state}> \rightarrow e_i$.

• Consider the path of app messages (through other processes) that go from $e_i$ to $e_j$.

• Due to FIFO ordering, markers on each link in above path will precede regular app messages.

• Thus, since $<p_i \text{ records its state}> \rightarrow e_i$, it must be true that $p_j$ received a marker before $e_j$.

• Thus $e_j$ is not in the cut => contradiction.
Global Snapshot Summary

• The ability to calculate global snapshots in a distributed system is very important.
• But don’t want to interrupt running distributed application.
• Chandy-Lamport algorithm calculates global snapshot.
• Obey causality (creates a consistent cut).
• Can be used to detect global properties.
  • Safety vs. Liveness.
**Revisions: notations and definitions**

- For a process $p_i$, where events $e_{i,0}, e_{i,1}, \ldots$ occur:
  
  \[
  \text{history}(p_i) = h_i = <e_{i,0}, e_{i,1}, \ldots>
  \]

  \[
  \text{prefix history}(p_i^k) = h_{i,k} = <e_{i,0}, e_{i,1}, \ldots,e_{i,k}>
  \]

  $s_{i,k} : p_i$'s state immediately after $k^{th}$ event.

- For a set of processes $\langle p_1, p_2, p_3, \ldots, p_n \rangle$:

  \[
  \text{global history: } H = \bigcup_i (h_i)
  \]

  a cut $C \subseteq H = h_{1,c_1} \cup h_{2,c_2} \cup \ldots \cup h_{n,c_3}$

  the frontier of $C = \{e_{i,c_i}, i = 1,2, \ldots n\}$

  global state $S$ that corresponds to cut $C = \bigcup_i (s_{i,c_i})$
More notations and definitions

• A **run** is a total ordering of events in H that is consistent with each $h_i$’s ordering.

• A **linearization** is a run consistent with happens-before ($\rightarrow$) relation in H.
Example

Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$  
Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$

Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

Run: $< e_1^0, e_1^1, e_1^2, e_1^3, e_2^0, e_2^1, e_2^2 >$
Linearization: $< e_1^0, e_1^1, e_1^2, e_2^0, e_2^1, e_2^2, e_1^3 >$
Example

Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$  Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$

Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

Run: $< e_1^0, e_1^1, e_1^2, e_1^3, e_2^0, e_2^1, e_2^2 >$

Linearization: $< e_1^0, e_1^1, e_1^2, e_2^0, e_2^1, e_2^2, e_1^3 >$
Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$  Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$

Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

$< e_1^0, e_1^1, e_2^0, e_2^1, e_1^2, e_2^2, e_1^3 >$:
Example

Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$  
Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$

Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

$< e_1^0, e_1^1, e_2^0, e_2^1, e_1^2, e_2^2, e_1^3 >$: Linearization

$< e_1^0, e_2^1, e_2^0, e_1^1, e_1^2, e_2^2, e_1^3 >$: 
Example

Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$  
Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$

Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

$< e_1^0, e_1^1, e_2^0, e_2^1, e_1^2, e_2^2, e_1^3 >$: Linearization

$< e_1^0, e_2^1, e_2^0, e_1^1, e_1^2, e_2^2, e_1^3 >$: Not even a run
More notations and definitions

• A **run** is a total ordering of events in H that is consistent with each $h_i$'s ordering.

• A **linearization** is a run consistent with happens-before ($\rightarrow$) relation in H.

• Linearizations pass through consistent global states.
Example

Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$    Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$
Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

Linearization: $< e_1^0, e_1^1, e_1^2, e_2^0, e_2^1, e_2^2, e_1^3 >$
Example

Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$  
Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$

Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

Linearization: $< e_1^0, e_1^1, e_1^2, e_2^0, e_2^1, e_2^2, e_1^3 >$
Example

Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$  
Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$

Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

Linearization: $< e_1^0, e_1^1, e_1^2, e_2^0, e_2^1, e_2^2, e_1^3 >$
Order at $p_1$: $\langle e_1^0, e_1^1, e_1^2, e_1^3 \rangle$  
Order at $p_2$: $\langle e_2^0, e_2^1, e_2^2 \rangle$

Causal order across $p_1$ and $p_2$: $\langle e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 \rangle$

Linearization: $\langle e_1^0, e_1^1, e_1^2, e_2^0, e_2^1 | e_2^2, e_1^3 \rangle$
Example

Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$  
Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$

Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

Linearization: $< e_1^0, e_1^1, e_1^2, e_2^0, e_2^1, e_2^2, e_1^3 >$
Example

Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$  
Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$

Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

Linearization: $< e_1^0, e_1^1, e_1^2, e_2^0, e_2^1, e_2^2, e_1^3 >$
Linearization $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$
More notations and definitions

- A run is a total ordering of events in H that is consistent with each \( h_i \)'s ordering.

- A linearization is a run consistent with happens-before (\( \rightarrow \)) relation in H.

- Linearizations pass through consistent global states.

- A global state \( S_k \) is reachable from global state \( S_i \), if there is a linearization that passes through \( S_i \) and then through \( S_k \).

- The distributed system evolves as a series of transitions between global states \( S_0, S_1, \ldots \).
State Transitions: Example

Many linearizations:
- \( <p_0, p_1, p_2, q_0, q_1, q_2> \)
- \( <p_0, q_0, p_1, q_1, p_2, q_2> \)
- \( <q_0, p_0, p_1, q_1, p_2, q_2> \)
- \( <q_0, p_0, p_1, p_2, q_1, q_2> \)
- \( \ldots \)

Causal order:
- \( p_0 \rightarrow p_1 \rightarrow p_2 \)
- \( q_0 \rightarrow q_1 \rightarrow q_2 \)
- \( p_0 \rightarrow p_1 \rightarrow q_1 \rightarrow q_2 \)

Concurrent:
- \( p_0 \parallel q_0 \)
- \( p_1 \parallel q_0 \)
- \( p_2 \parallel q_0, p_2 \parallel q_1, p_2 \parallel q_2 \)
State Transitions: Example

Execution Lattice. Each path represents a linearization.
State Transitions: Example

Execution Lattice. Each path represents a linearization.
State Transitions: Example

**Execution Lattice.** Each path represents a linearization.
State Transitions: Example

Execution Lattice. Each path represents a linearization.
State Transitions: Example

**Execution Lattice.** Each path represents a linearization.

Not valid! Why?
State Transitions: Example
State Transitions: Example

\[ s_{\{0,0\}} \xrightarrow{p_0} s_{\{0,1\}} \xrightarrow{q_0} s_{\{1,0\}} \xrightarrow{p_1} s_{\{1,1\}} \xrightarrow{q_0} s_{\{2,0\}} \xrightarrow{p_2} s_{\{2,1\}} \xrightarrow{q_0} s_{\{3,0\}} \xrightarrow{q_0} s_{\{3,1\}} \xrightarrow{q_1} s_{\{3,2\}} \xrightarrow{q_2} s_{\{3,3\}} \]
State Transitions: Example
State Transitions: Example
More notations and definitions

- A **run** is a total ordering of events in H that is consistent with each $h_i$’s ordering.

- A **linearization** is a run consistent with happens-before ($\rightarrow$) relation in H.

- Linearizations pass through consistent global states.

- A global state $S_k$ is reachable from global state $S_i$, if there is a linearization that passes through $S_i$ and then through $S_k$.

- The distributed system evolves as a series of transitions between global states $S_0, S_1, \ldots$. 
Global State Predicates

• A global-state-predicate is a property that is true or false for a global state.
  • Is there a deadlock?
  • Has the distributed algorithm terminated?

• Two ways of reasoning about predicates (or system properties) as global state gets transformed by events.
  • Liveness
  • Safety
Liveness

• **Liveness** = guarantee that something **good** will happen, eventually

• Examples:
  • Guarantee that a distributed computation will terminate.
  • “Completeness” in failure detectors.
  • All processes eventually decide on a value.

• A global state $S_0$ satisfies a **liveness** property $P$ iff:
  • $\text{liveness}(P(S_0)) \equiv \forall L \in \text{linearizations from } S_0, \ L \text{ passes through a } S_L \land P(S_L) = \text{true}$
  • For any linearization starting from $S_0$, $P$ is true for **some** state $S_L$ reachable from $S_0$. 
Liveness Example

If predicate is true only in the marked states, does it satisfy liveness? No
If predicate is true only in the marked states, does it satisfy liveness?

No
Liveness Example

If predicate is true only in the marked states, does it satisfy liveness? Yes
Liveness

• **Liveness** = guarantee that something **good** will happen, eventually

• **Examples:**
  • Guarantee that a distributed computation will terminate.
  • “Completeness” in failure detectors.
  • All processes eventually decide on a value.

• A global state $S_0$ satisfies a **liveness** property $P$ iff:
  • $\text{liveness}(P(S_0)) \equiv \forall L \in \text{linearizations from } S_0, \ L \text{ passes through a } S_L \& P(S_L) = \text{true}$
  • For any linearization starting from $S_0$, $P$ is true for some state $S_L$ reachable from $S_0$. 
Safety

- Safety = guarantee that something **bad** will **never** happen.

- Examples:
  - There is no deadlock in a distributed transaction system.
  - “Accuracy” in failure detectors.
  - No two processes decide on different values.

- A global state \( S_0 \) satisfies a **safety** property \( P \) iff:
  - \( \text{safety}(P(S_0)) \equiv \forall S \text{ reachable from } S_0, P(S) = \text{true} \).
  - For all states \( S \) reachable from \( S_0 \), \( P(S) \) is true.
Safety Example

If predicate is true only in the marked states, does it satisfy safety? **No**
Safety Example

If predicate is true only in the unmarked states, does it satisfy safety?

Yes
Safety

• **Safety** = guarantee that something **bad** will **never** happen.

• Examples:
  • There is no deadlock in a distributed transaction system.
  • “Accuracy” in failure detectors.
  • No two processes decide on different values.

• A global state $S_0$ satisfies a **safety** property $P$ iff:
  • $\text{safety}(P(S_0)) \equiv \forall S \text{ reachable from } S_0, P(S) = \text{true}.$
  • For all states $S$ reachable from $S_0$, $P(S)$ is true.
Global Snapshot Summary

• The ability to calculate global snapshots in a distributed system is very important.
• But don’t want to interrupt running distributed application.
• Chandy-Lamport algorithm calculates global snapshot.
• Obeys causality (creates a consistent cut).
• Can be used to detect global properties.
• Safety vs. Liveness.