Global State

Today
- Global snapshot
Global state – what for?

- Garbage collection – an object is garbage if there are no longer any references anywhere in the system
  - Looking at the example, we clearly need both the state of the processes and the communication channels

- Deadlock detection

- Debugging
  - Check the state of a related set of variables held by different processes
Global state

- How can we determine a given property holds in a distributed system in execution?
- With shared memory, a single clock or perfectly synchronized clocks, easy

- Take snapshot at 6:25:00PM CST
- But you know there’s nothing like it
Global state

- Each process can record the states that take place there
  - \( s_i^k \) – state of \( p_i \) right before \( kth \) event
  - \( s_i^0 \) is the initial event

- To capture the state of the channel
  - Process record the sending and received of all messages as part of their state
  - if sent but not yet received, then it’s in the channel

- Global state – can be made of any set of state of all processes \( S = (s_1, s_2, \ldots s_N) \)
  - But which global states are meaningful?
Global states, cuts and frontiers

- Cut of a system’s execution – a subset of its global history (which is a union of prefixes of processes histories) \( C = h_{1}^{c_{1}} \cup h_{2}^{c_{2}} \cup ... \cup h_{N}^{c_{N}} \)

Prefix of process \( p_1 \) history

Cut

Prefix of process \( p_2 \) history
The set of events \( \{e_i^C: i = 1, 2, \ldots N\} \) is called the \textit{frontier} of the cut.
Consistent cuts and global states

- A cut $C$ is consistent if, for each event, it contains all the events that happened-before that event.
- A consistent global state is one that corresponds to a consistent cut.
Runs and linearization

- A run – a total ordering of all events in a global history that is consistent with each local history’s ordering
- A linearization or consistent run – an ordering of the events in a global history that is consistent with the happened-before relation on \( H \)
- All linearization (but not all runs) pass only through consistent global states
- A state \( S' \) is reachable from a state \( S \) if there is a linearization that passes through \( S \) and then \( S' \)
- Detecting a condition (i.e., deadlock) = evaluating a global state predicate
  - A function of set of global states of a system \( \rightarrow \) \{true, false\}
  - Some predicates are stable (deadlock), others are not
Chandy & Lamport’s snapshot algorithm

- **Goal** – to record a consistent global state
- **Algorithm** records state locally at processes, it says nothing about collecting it

**Assumptions**
- Neither channel nor processes fail; reliable, exactly-once communication
- Channels are unidirectional and provide FIFO ordering
- Graph of processes and channels is strongly connected
- Any process may initiate the global snapshot at any time (just send a marker on a non-existing channel)
- Processes may continue execution and communication while snapshot is taking place
Algorithm idea

- Each process records
  - Its state and, for each incoming channel, the set of messages sent to it
  - For each channel, any message that arrived after it records its state but before the sender records its own state

- Algorithm relies on special message, *markers*
  - A prompt for the receiver to save its own state, if it has not yet done it
  - A mean of determining which messages to include in the channel state
  - Since channels are FIFO, markers separate the messages in the channel to be included in the snapshot (process or channel state) from those not to be included
Algorithm

Algorithm is defined by two rules

**Marker receiving rule for process** \( p_i \)

On receipt of a marker message at \( p_i \) over channel \( c \)

If \( (p_i \) has not yet recorded its state) it
- records its process state
- records the state of \( c \) as the empty set
- turns on recording of messages arriving over other incoming channels

Else
\( p_i \) records the state of \( c \) as the set of messages it has received over \( c \) since it saved its state

endif
Algorithm

Algorithm is defined by two rules …

Marker sending rule for process $p_i$
After $p_i$ has recorded its state, for each outgoing channel $c$, before it sends any other message over $c$
- $p_i$ sends one marker message over $c$
Example execution

- Two processes trading widgets; \( p_1 \) sends order over \( c_2 \) for widgets at $10 per widget; later \( p_2 \) send widgets over \( c_1 \)

Initial state; \( p_2 \) has already received an order for 5 widgets that’s about the send

\[
\begin{array}{c|c}
\text{Account} & \text{Widgets} \\
\hline
$1000 & 2000 \\
\end{array}
\]
Example execution

$p_1$ saves state in global state $S_0$ and sends a **marker** before sending a new order.
Example execution

$p_2$ emits order for 5 widgets from previous request; system enters $S_2$
Example execution

$p_1$ receives order and $p_2$ receives marker; $p_2$ saves its state <$\$50, 1995$> and that of channel $c_2$ as empty; sends marker over $c_1$

$p_1$ receives marker over $c_1$ it records the state of that channel as the single message (five widgets) that has received after first recorded its state
Example execution

- Final recorded state:
  \[ p_1: <$1000,0> \]
  \[ p_2: <$50,1995> \]
  \[ c_1: <\text{(five widgets)}> \]
  \[ c_2: <> \]

- Note that this state differ from all the global states through which the system actually passed!
  - But could have passed through it in an equivalent execution
  - And if a stable property holds before, it will hold in the recorded global snapshot

- The snapshot algorithm selects a cut, and therefore a state, that is consistent; that is, for all \( e_i \rightarrow e_j \), if \( e_j \) is in the cut, then \( e_i \) is too
Many other approaches since

- Optimizing concurrent initiation and distribution of the recorded snapshot – Spezialletti-Kearns ‘89
- Optimizing incremental snapshots – Venkatesan ‘93
- Handling non-FIFO channels – Helary ‘89, Lai-Yang ‘87
- …
Summary

- Recording global state is an important paradigm in the design of distributed systems
- So it’s designing efficient methods to do it
- The challenge comes from the lack of shared memory or a global clock
- There are many alternative algorithms out there
  - Look at your application requirements to choose among them
  - E.g. checkpointing, termination detection, global state monitoring, …