Today

- Global snapshot and applications
Global state – what for?

- Garbage collection – an object is garbage if there are no longer any references anywhere in the system
  - If garbage, you can reclaim the memory
  - To decide, check there are no references
  - From the example, we clearly need both the state of the processes and the communication channels
    - Process $p_1$ has 2 objects, one with a local, one with a remote ref
    - Process $p_2$ has one garbage object (gray) and one with a reference in a message in transit
Global state – what for?

- Deadlock detection
  - Checking each process, but a message in the channel can get them running again

- Distributed termination detection
  - Check the state of a related set of variables held by different processes

Diagram:

```
  p1  Wait for  p2
```

```
  Wait for
```

Wait for
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

- A distributed system – processes that communicate by passing messages over channels

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

- To capture the state of the channel
  - Process record sending/received of msgs as part of their state
  - if sent but not yet received, then it must be 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?
An (inconsistent) global state

- A system that maintains bank accounts A and B

<table>
<thead>
<tr>
<th>Time (t)</th>
<th>Account A</th>
<th>Channel C1</th>
<th>Channel C2</th>
<th>Account B</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>$500</td>
<td>$0</td>
<td>$0</td>
<td>$200</td>
</tr>
<tr>
<td>t2</td>
<td>$400</td>
<td>$0</td>
<td>$100</td>
<td>$200</td>
</tr>
<tr>
<td>t3</td>
<td>$400</td>
<td>$50</td>
<td>$100</td>
<td>$150</td>
</tr>
<tr>
<td>t4</td>
<td>$450</td>
<td>$0</td>
<td>$100</td>
<td>$150</td>
</tr>
<tr>
<td>t5</td>
<td>$450</td>
<td>$0</td>
<td>$0</td>
<td>$250</td>
</tr>
</tbody>
</table>

Record state of A at t1: A: $500

Record state of B and channels C1 and C2 at t3:
- B: $150
- C1: $50
- C2: $100

Global state shows $800 in the system!

State of A was recorded before sending $100 but state of C2 after $100 transfer was initiated
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

\[ e_1^0 \quad e_1^1 \quad e_1^2 \]

\[ m_1 \]

Cut

\[ e_1^3 \]

Physical time

\[ m_2 \]

Prefix of process \( p_2 \) history

\[ e_2^0 \quad e_2^1 \quad e_2^2 \]
The set of events $\{e_{i}^{C_i} : i = 1, 2, ..., N\}$ is called the *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.

![Diagram showing consistent and inconsistent cuts](image-url)

- **Inconsistent cut**: Frontier: $\langle e_1^0, e_2^0 \rangle$
- **Consistent cut**: Frontier: $\langle e_1^2, e_2^2 \rangle$

Physical time
The execution of a system can be characterized as a series of transitions between global states:

\[ S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \ldots \]

- 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 linearizations (but not all runs) pass only through consistent global states
Runs and linearization

- 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
  - BTW, it says nothing about collecting it
- Algorithm records state locally at processes
- 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 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

Defined by two rules – marker sending and receiving

<table>
<thead>
<tr>
<th>Marker receiving rule for process $p_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>On receipt of a marker message at $p_i$ over channel $c$</td>
</tr>
<tr>
<td>If ($p_i$ has not yet recorded its state) it</td>
</tr>
<tr>
<td>- records its process state</td>
</tr>
<tr>
<td>- records the state of $c$ as the empty set</td>
</tr>
<tr>
<td>- turns on recording of messages arriving over other incoming channels</td>
</tr>
<tr>
<td>Else</td>
</tr>
<tr>
<td>$p_i$ records the state of $c$ as the set of messages it has received over $c$ since it saved its state</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram of a process $p_i$ receiving marker messages over channels $c_1$ and $c_2$." /></td>
</tr>
<tr>
<td>$p_i$ records the state of channel $c_2$ as the part of its state.</td>
</tr>
</tbody>
</table>
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
Example execution

\[ S_0 \]

\[ p_1 \rightarrow (\text{empty}) \leftarrow p_2 \]

\$1000,0 \quad \$50,2000

\[ c_2 \quad c_1 \]

\[ (\text{empty}) \quad (\text{empty}) \]

\( p_1 \) saves state in global state \( S_0 \)

\[ S_1 \]

\[ p_1 \rightarrow (10, \$100), M \leftarrow p_2 \]

\$900,0 \quad \$50,2000

and sends a **marker** on its outgoing channel before sending a new order
Example execution

$p_2$ receives the marker 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!

\[
S_0: \{<$1000,0>, <$50,2000>,<> ,<>\}
S_1: \{<$900,0>, <$50,2000>, <>,<(Order 10, $100)M>\}
S_2: \{<$900,0>, <$50,1995>,<5 widgets>,<(Order 10, $100)M>\}
S_3: \{<$900,5>, <$50,1995>,<> ,<(Order 10, $100)M>\}\]
Example execution

- 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
Several variants 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, …