Global State

- Global state of a distributed system consists of
  - Local state of each process: messages sent and messages received
  - State of each channel: messages sent but not received
- Problem: how to record the global state of a distributed system?
  - Cannot require that all local observations must happen at the same time due to lack of global time
  - Using the state of the individual processes at arbitrary points in time may not result in a consistent overall picture
- Problem: record a consistent global state, also called a **distributed snapshot**
  - A global state is consistent if for any received message in the state the corresponding send is also in the state

A global state is consistent if it corresponds to a consistent cut.

(a) A consistent cut. b) An inconsistent cut
Distributed Snapshot Algorithm

- Assumptions: communication channels are reliable, message delivery is ordered
- Any process can initiate the algorithm by
  - Recording its local state, and
  - Sending a marker over each outgoing channel
- On receiving a marker over incoming channel C
  - If local state is not saved, save the local state and send a marker over each outgoing channel
  - Otherwise, save all messages received via C since local state is saved and until the marker arrived
- A process finishes when it receives a marker on each incoming channel and processes them all. At this time, it has accumulated
  - A local state snapshot
  - For each incoming channel, a set of messages received after performing the local snapshot and before the marker came down that channel

---

An example of the distributed snapshot algorithm

<table>
<thead>
<tr>
<th>Entity</th>
<th>Recoded State</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>No messages have been sent or received</td>
</tr>
<tr>
<td>P2</td>
<td>m1 and m2 have been sent. No messages have been received</td>
</tr>
<tr>
<td>P3</td>
<td>m2 has been received. No messages have been sent</td>
</tr>
<tr>
<td>CH12</td>
<td>Empty</td>
</tr>
<tr>
<td>CH21</td>
<td>Contains m1</td>
</tr>
<tr>
<td>CH13</td>
<td>Empty</td>
</tr>
<tr>
<td>CH31</td>
<td>Empty</td>
</tr>
<tr>
<td>CH23</td>
<td>Empty</td>
</tr>
<tr>
<td>CH32</td>
<td>Empty</td>
</tr>
</tbody>
</table>
Recorded State versus Global State

![Diagram showing recorded state versus global state]

**Figure 17.10** State recording of the system of Figures 17.6 and 17.7.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description of recorded state</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>No messages have been sent. Message $m_{11}$ has been received.</td>
</tr>
<tr>
<td>$P_2$</td>
<td>Message $m_{11}$ has been sent. No messages have been received.</td>
</tr>
<tr>
<td>$P_3$</td>
<td>No messages have been sent or received.</td>
</tr>
<tr>
<td>$P_4$</td>
<td>No messages have been sent or received.</td>
</tr>
</tbody>
</table>

*States of all channels are recorded as empty.

A state recorded by the distributed snapshot algorithm may not match any global state of the system!

Distributed Termination Detection

- **When a process terminates, OS frees its resources**
  - This approach is not adequate for distributed systems
- **Processes of a distributed computation should be terminated when all of them have completed their tasks**
  - A process is *active* when it is performing work, and *passive* when it has no work
  - Work is assigned to a process through a message
    - A passive process becomes active on receiving a message
- **Distributed termination condition (DTC):**
  - All processes of a distributed computation are passive
  - No messages are in transit
Distributed Termination Detection (continued)

- Credit distribution based termination detection
  - A distributed computation is initiated with credit \( C \), which is distributed among its processes
    - A process sends some of its credit in each message
    - A process receiving a message adds the credit from the message to its own credit
  - When a process becomes passive, it sends its credit to collector process
  - The distributed computation is known to have terminated when credit accumulated by collector is \( C \)
- Diffusion computation based termination detection
  - Each process that becomes passive initiates a diffusion computation to determine if DTC holds
  - Assumption: sender of a message becomes blocked until it receives an ACK for the message

Diffusion Computation Based Termination Detection

**Algorithm 18.6 Distributed Termination Detection**

1. When a process becomes passive: The process initiates a diffusion computation through the following actions:
   a. Send “Shall I declare distributed termination?” queries along all edges connected to it.
   b. Remember the number of queries sent out, and await replies.
   c. After replies are received for all of its queries, declare distributed termination if all replies are yes.
2. When a process receives an engaging query: If the process is in the active state, it sends a no reply; otherwise, it performs the following actions:
   a. Send queries along all edges connected to it excepting the edge on which it received the engaging query.
   b. Remember the number of queries sent out, and await replies.
   c. After replies are received for all of its queries: If all replies are yes, send a yes reply to the process from which it received the engaging query; otherwise, send a no reply.
3. When a process receives a nonengaging query: The process immediately sends a yes reply to the process from which it received the query.

Definition: The first query received by a node is called an engaging query.