Global States and Snapshots
Key Properties

- Multiple computers
  - Concurrent execution
  - Independent failures
  - Autonomous administrators
  - Heterogeneous capacities, properties
  - Large numbers (scalability)
- Networked communication
  - Asynchronous execution
  - Unreliable delivery
  - Insecure medium
- Common goal
  - Consistency – can discuss whole-system properties
  - Transparency – can use the system without knowing details
Outline

- Global States
  - Motivation
  - Definition
- Snapshots
  - Chandy-Lamport algorithm
Global State

- Want to find out a *global* property of the system
  - but can only make local observations at any time
Global State: Count

- Count the number of people in all EWS labs
  - Option 1:
    - Go to each lab and count up people
    - Problems?
  - Option 2:
    - Send one person to each lab to count
    - Problems?
Distributed Garbage Collection

- Newspaper on kitchen counter
  - Should you recycle it?
- Go to each roommate’s room, ask if roommate done
  - When can you throw out the paper?
- Stable property
  - If someone says “not done”, may not be safe to recycle
  - If everyone says “done”, must be safe to recycle
Other Applications

- Detect other stable properties
  - Termination
  - Deadlock
- Save a checkpoint
  - Recover to a “known good” state
For a process \( P_i \), where events \( e_i^0, e_i^1, \ldots \) occur:

- history (\( P_i \)) = \( h_i = \langle e_i^0, e_i^1, \ldots \rangle \)
- prefix history (\( P_i^k \)) = \( h_i^k = \langle e_i^0, e_i^1, \ldots, e_i^k \rangle \)
- \( S_i^k \): \( P_i \)'s state immediately after \( k \)th event

For a set of processes \( P_1, \ldots, P_i, \ldots \):

- global history: \( H = \bigcup_i \langle h_i \rangle \)
- global state: \( S = \bigcup_i \langle S_i^k \rangle \)
- a cut \( C \subseteq H = h_1^{c_1} \cup h_2^{c_2} \cup \ldots \cup h_n^{c_n} \)
- the frontier of \( C = \{e_i^{c_i}, i = 1, 2, \ldots, n\} \)
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
- A consistent cut $\equiv$ a global snapshot
Consistent Snapshots

**Snapshot**: state $S_i^{T_i}$ of each process $P_i$ at (local) time $T_i$

**Consistent snapshot**: any message $m$ from $P_i$ to $P_j$ received by $P_j$ at time $t \leq T_j$ must have been sent at time $t' \leq T_i$

If $e_i^k \rightarrow e_j^l$ then either $l > T_j$ or $k \leq T_i$
Chandy-Lamport Algorithm

- Goal: Record global snapshot
  - Process state
  - Channel state
  - Consistent cut
- System model
  - FIFO, reliable channels between any two processes
  - Each process records own state (no central collection)
  - Any process may initiate algorithm
Two colors for processes / events / channels
- Blue: before snapshot
- Red: after snapshot

To initiate:
- Color self red
- Send red marker to each other process

When red marker received in blue state
- Color self red
- Send red marker to each other process
Example

Diagram showing connections between P1, P2, and P3 with points a, b, c, d, e, f, g, h, i, and j. Arrows labeled X and Y.
Example
Example

P1: a b c d

P2: e f g Y X

P3: h i j

initiate snapshot
Example

P1

P2

P3

a b c d

e f g

initiate
snapshot

X Y

h i
Example
Marker receiving rule for process $p_i$

On $p_i$’s receipt of a marker message over channel $c$:

- If ($p_i$ has not yet recorded its state) it
  - records its process state now;
  - 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.

End if

Marker sending rule for process $p_i$

After $p_i$ has recorded its state, for each outgoing channel $c$:

- $p_i$ sends one marker message over $c$
  - before it sends any other message over $c$.

Note: initiating the snapshot is like receiving a marker from yourself
1- P1 initiates snapshot: records its state (S1); sends Markers to P2 & P3; turns on recording for channels C21 and C31

2- P2 receives Marker over C12, records its state (S2), sets state(C12) = {} sends Marker to P1 & P3; turns on recording for channel C32

3- P1 receives Marker over C21, sets state(C21) = {a}

4- P3 receives Marker over C13, records its state (S3), sets state(C13) = {} sends Marker to P1 & P2; turns on recording for channel C23

5- P2 receives Marker over C32, sets state(C32) = {b}

6- P3 receives Marker over C23, sets state(C23) = {}

7- P1 receives Marker over C31, sets state(C31) = {}}
Habitrail
Habitrail

Pod 1

Pod 2

Pod 3

N. Borisov, U. Illinois
Pod 1 has 3 hamsters

Pod 1

Pod 2

Pod 3

N. Borisov, U. Illinois
Channel State

Pod 1

Channel 2 -> 1 has 1 hamster

Pod 2

Channel 3 -> 1 is empty

Pod 3

N. Borisov, U. Illinois
Pod 1

Hamster dies in Pod 1

Hamster born in Pod 2

Pod 2

Hamster leaves on channel 1->3

Pod 3

Hamster arrives on channel 3->2

N. Borisov, U. Illinois

Fall'21
Pod 1 state:
3 hamsters

Pod 2 state:
1 hamster

Pod 3 state:
1 hamster

Channel 1->3
1 hamster

Channel 2->3
empty

Channel 3->1
empty

Channel 1->2
empty

Channel 2->1
empty

Channel 3->2
1 hamster
Let $e_i$ and $e_j$ be events occurring at $p_i$ and $p_j$, respectively such that $e_i \rightarrow e_j$.

The snapshot algorithm ensures that if $e_j$ is in the cut then $e_i$ is also in the cut.

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 $<p_i$ records its state$>$ $\rightarrow e_i$
- Consider the path of app messages (through other processes) that go from $e_i \rightarrow e_j$
- Due to FIFO ordering, markers on above path precede regular app messages
- Thus, since $<p_i$ 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 $\Rightarrow$ contradiction