03_04_Global State

1. Preliminary

1.1. Definition

A (global) snapshot of an execution of a distributed algorithm is a **configuration of this execution**, consisting of

1. the local states of the processes and
2. the messages in transit

1.2. Challenge

1. Snapshot cannot be defined based on physical time (e.g., the composition of all local state at the same time instant)
2. How can we determine the (a) global state of an asynchronous distributed system without lack of a common clock and with arbitrary delays of messages

1.3. Terminology

1.3.1. Basic Messages & Control Messages

Suppose we design an **algorithm** that takes a snapshot of **another distributed algorithm**. We call the messages of the underlying algorithm **basic messages** and messages of the snapshot algorithm **control messages**

1.3.2. Cut

- A cut is a **set** consisting of one internal event for every process:

$$\{c_1, c_2, ..., c_n\}$$

with c_i an internal event in P_i

- A cut C = $\{c_1, c_2, \cdots, c_n\}$ is **consistent iff** for every $i, j = 1, 2, ..., n, i \neq j$,there do not exist events $e_i \in E_i$ and $e_j \in E_j$ with$(e_i \rightarrow e_j) \wedge (e_j \rightarrow c_j) \wedge (e_i \nrightarrow c_i)$.
- **Theorem 1**:A cut C = $\{c_1, c_2, \cdots, c_n\}$ is consistent iff all i, j $c_i || c_j$
- The **vector time** of cut C:

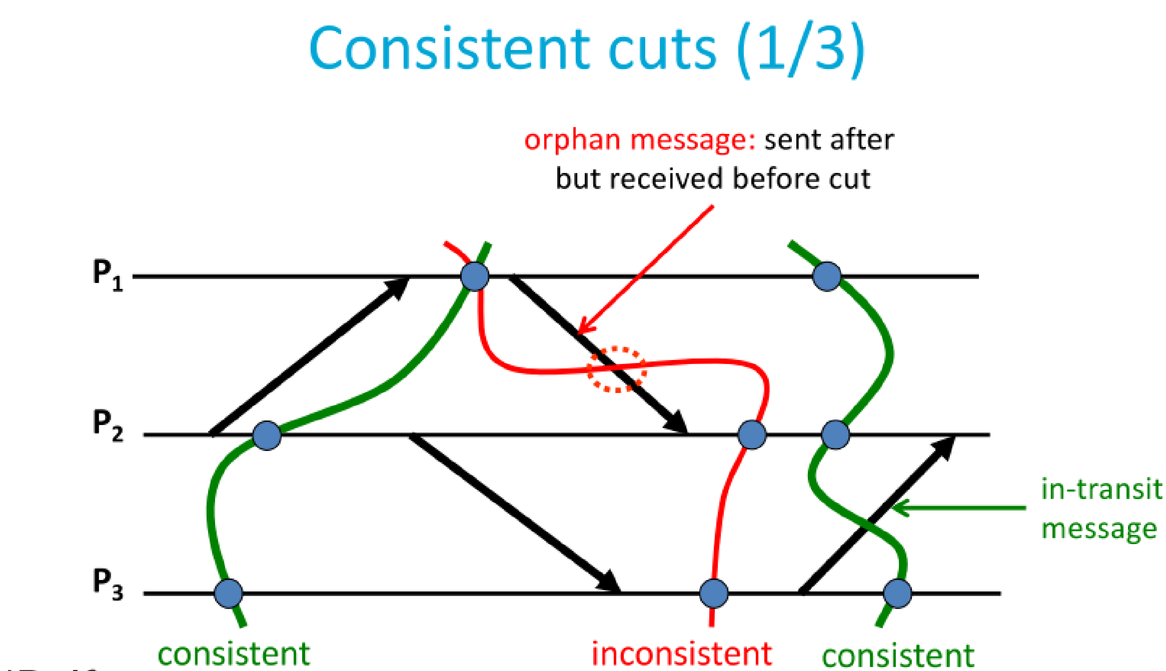
$$\mathbf{V}(\mathbf{C})[\mathbf{i}] = \max\left(\mathbf{V}\left(\mathbf{c}_1\right)[i], \mathbf{V}\left(\mathbf{c}_2\right)[\mathbf{i}], \ldots, \mathbf{V}\left(\mathbf{c_n}\right)[\mathbf{i}]\right) \text{ component-wise maximum}$$

- **Theorem 2**: C is consistent iff

$$V(C) = \left(V\left(c_1\right)[1], V\left(c_2\right)[2], \ldots, V\left(c_n\right)[n]\right)$$
$$\text{local component is maximum}$$

Understanding:

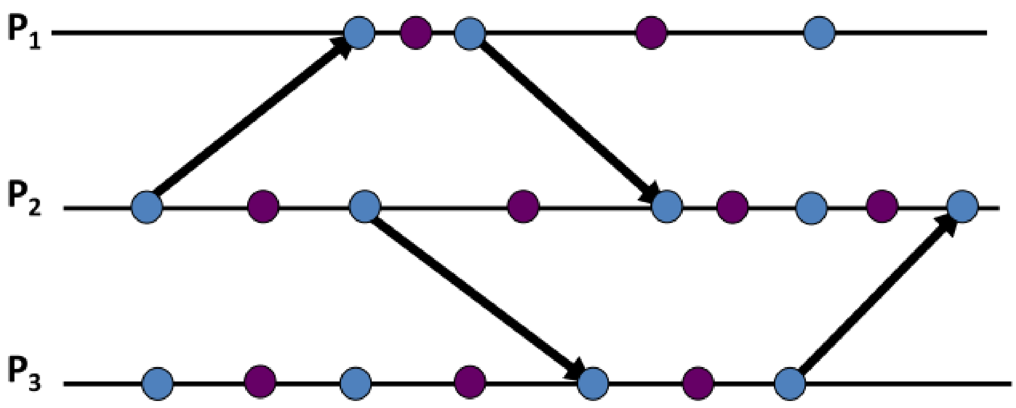
1. A cut divide the timeline of processes into two parts
2. A consistent cut guarantee an event happens in the later part should have no effect on the previous part.
3. So there should be a sent after but received before in a consistent cut



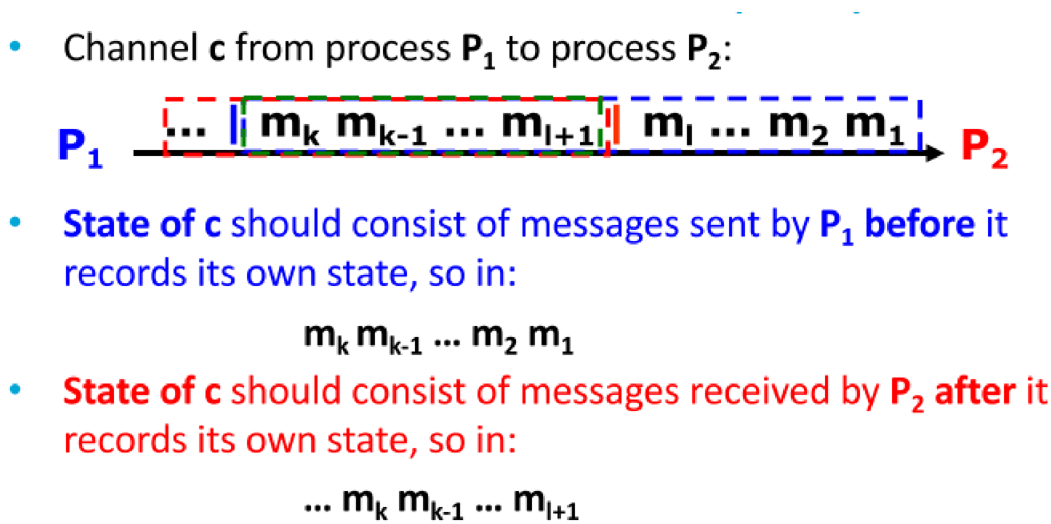
4. The Theorem 2 means the local component is maximum, means when the cut happens, a process should not have more knowledge than another process of the process itself.

Problem Transformation

So now, the global state detection problem becomes a problem to find a set of concurrent state-recording events



2. State of a Channel



State of a channel (supposed to be FIFO) is a **sequence of messages**

3. Global State Detection Method

3.1. Chandy's and Lamport's Algorithm

3.1.1. Assumption

1. Channels are fault-free and **FIFO**
2. State of a channel is a **sequence of consecutive messages**
3. Every process may **spontaneously start** the algorithm for recording the global state
4. The **directed graph** of processors and unidirectional channels must be **strongly connected** (path from every processor to every other processor

3.1.2. Preparation

1. Message Types: Marker
2. Process: **Buffer** to record the message from incoming link

3.1.3. Procedure

1. Any processor wish to record the global state of a system:
    a. first **records its own** local state
    b. sends a special message, **a marker,**along every outgoing channel.
2. First Receipt of a marker:
    a. records the state of that **channel** as the **empty** state
    b. records its own local state
    c. sends a **marker** along every outgoing channel
    d. creates an **initially empty FIFO message buffer** for each of its incoming channels
3. Later Receipt of a marker:
    a. the state of **that channel** is recorded as the sequence of messages in the corresponding buffer

3.1.4. Implementation

I. Spontaneously recording a processor's state

```
record_and_send_markers:
record local state
loc_state_recorded ← true
for every outgoing channel c do
    send marker along c
for every incoming channel c do
    create message buffer B_c
```

II. Receiving a marker along a channel

```
upon receipt of marker along channel c do
    if (¬ loc_state_recorded) then
        record state of c as empty
        record_and_send_markers
    else
        record state of c as contents of B_c
```

III. Receiving a message along a channel

```
upon receipt of m along channel c do
    append m to B_c
```

3.1.5. Understanding

1. The recorded state may not has occured, It is a state that might have occurred if the sequence of events would have been different
2. By using the marker, it ensures the exact snapshots of each channel

3.1.6. Correctness

- the global state recorded **may not actually have occurred** between any two successive events in S.

- We will now show that there is a sequence S′ of events **equivalent** to S such that the recorded state does occur in S, or to say It is a state that **might have occurred** if the sequence of events would have been different

An event in a process is:

- **pre-recording** (**post-recording**) if it occurs in the process **before** (**after**) the process has recorded its own state

Of course, **all events of a single process p** appear in **S** in the order **first** all pre-recording, **then** all post-recording events:

$$S = e\ e\ e\ e_p\ e_p\ e\ e\ e\ e_p\ e\ e\ e_p \mid e\ e\ e\ e_p\ e_p\ e\ e\ e_p\ e\ e\ e_p$$

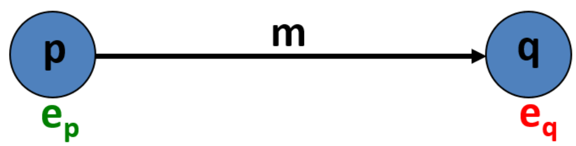
Events of **different processes** may **occur out of order**

- But we can **reorder** two neighboring events (in different processes p and q) in S that are out of order

We can reorder e_p and e_q, **unless** e_p is sending a message **m** and e_q is receiving that message

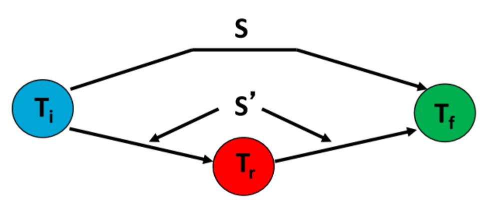
But that is not possible:

- e_p is post-recording
- so **p** sent a marker before sending **m**
- so **q** received the marker and recorded its state before receiving **m**
- but then e_q is post-recording!! **Contradiction**



Let S’ be the reordered sequence of events

- With S’, the recorded states of the processes are the same or we say **equivalent** (same events before/after recording)



- With S’, the recorded states of the channels are also the same, or we say **equivalent**