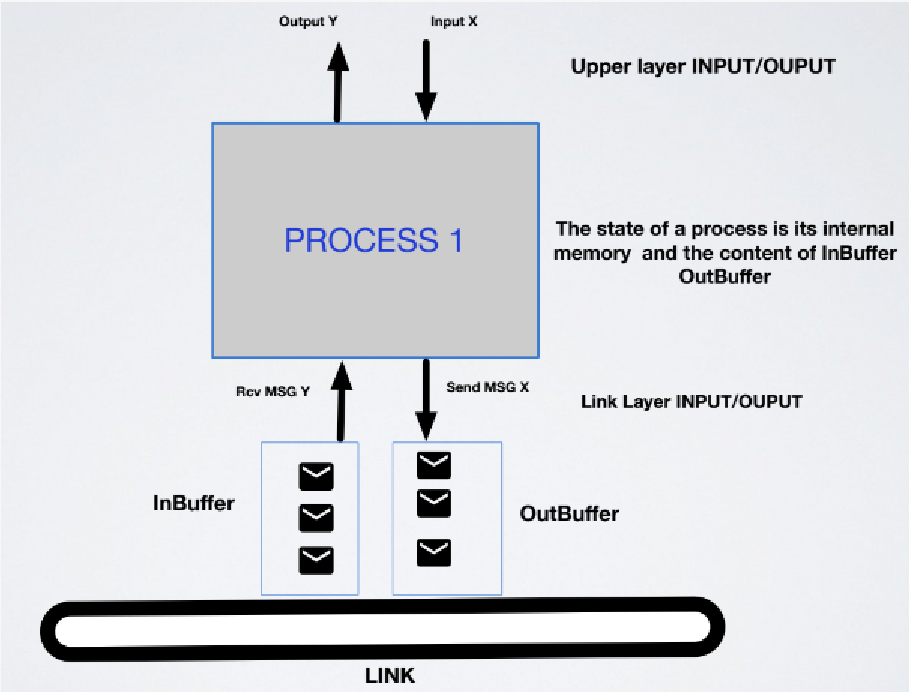
1. Models, Abstractions and Basic Concepts

> **DEFINITION** (System)
> We have n processes in $\Pi : \{p_0, \ldots, p_{n-1}\}$ with distinct identities.
> Processes communicate with a **communication graph**: $G : (\Pi, E)$ (usually G is complete).
> The communication happens by exchanging messages on communication link.

> **DEFINITION** (Process)
> A process is a (possibly infinite) State Machine (I/O Automaton).



Processes Communication is based on a link defined by an Input Buffer and an Output Buffer.

- internal states: set Q
- initial states: set $Q_i \subset Q$
- Messages: set all possible messages M in the form $(< sender, reciver, payload >)$
- $InBuff_j$: multiset of delivered messages
- $OutBuff_j$: multiset of inflight messages (messages sent but not delivered)

$$P_j\,(q \in Q \cup Q_{in}\,, InBuff_j\,) = (q' \in Q, Send_{msg} \subset M)$$

where:

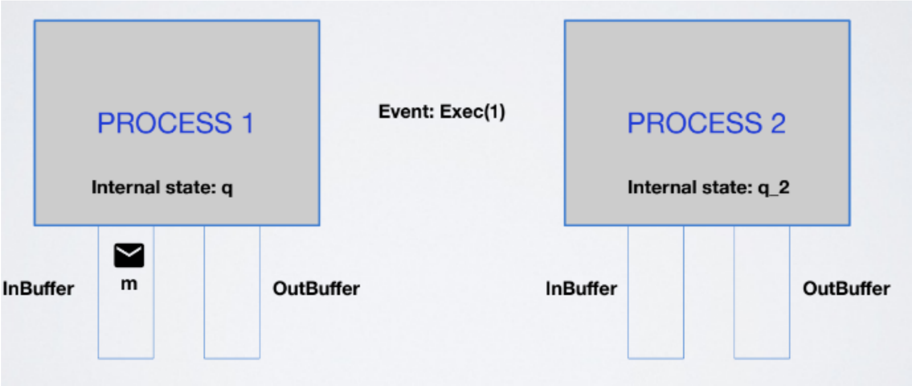
- $OutBuff_j = OutBuff_j \cup Send_{msg}$
- $InBuf = \emptyset$

MODEL: Asynchronous Executions

> **DEFINITION** (Execution)
> Scheduling of a set of events (scheduler):

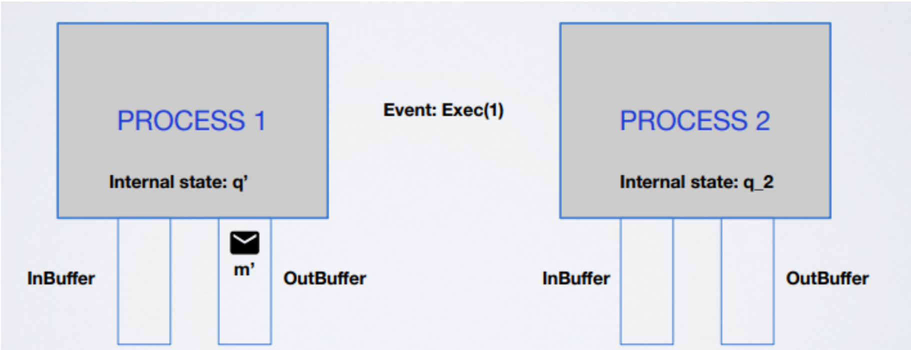
- Delivery of a message $Del(m, i, j)$: move message m from $OutBuff_i$ to $InBuff_j$
- Execution of a local step $Exec(i)$: process i executes one step of its state machine

Message is on InBuffer of the first process



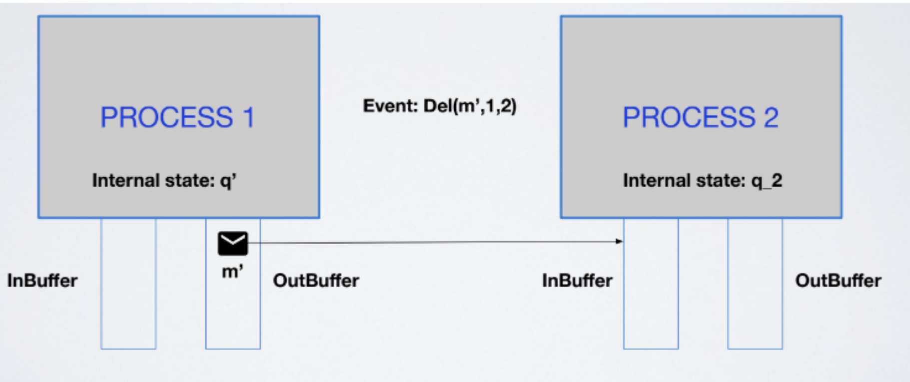
Message passes at the OutBuffer of P_1 doing:

- $P_1(q, \{m\}) = (q', \{< 1, 2, m'\})$
- $OutBuff_1 = OutBuf_1 \cup \{< 1, 2, m' >\}$

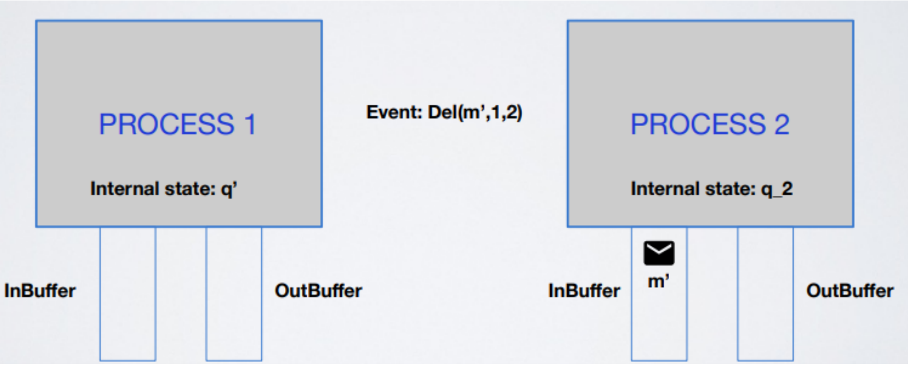


OutBuffer of P_1 send the message m' to InBuffer of P_2 doing:

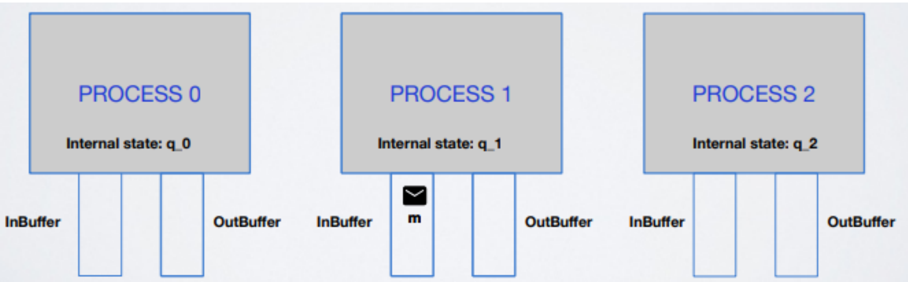
- $InBuf_2 = InBuf_2 \cup \{< 1, 2, m' >\}$



result:



Configuration C_t : is a vector of n components.
Component j is the state of process j: $C_t[j](q_i, InBuff_j, outBuff_j)$



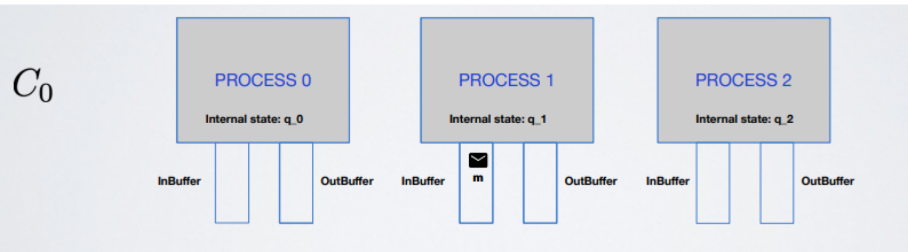
$$C_0 =< (q_o, \{\}, \{\}), (q_1, \{m\}, \{\}), (q_2. \{\}, \{\}) >$$

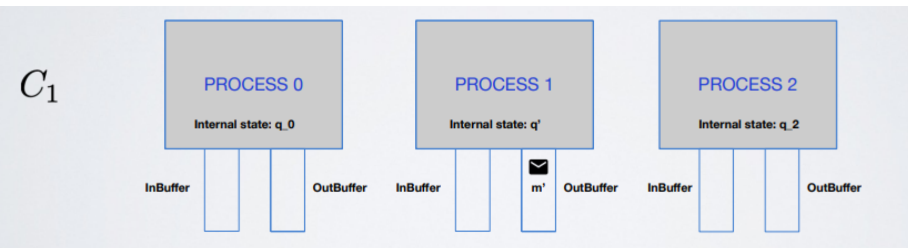
An event e is enabled in a configuration C if:

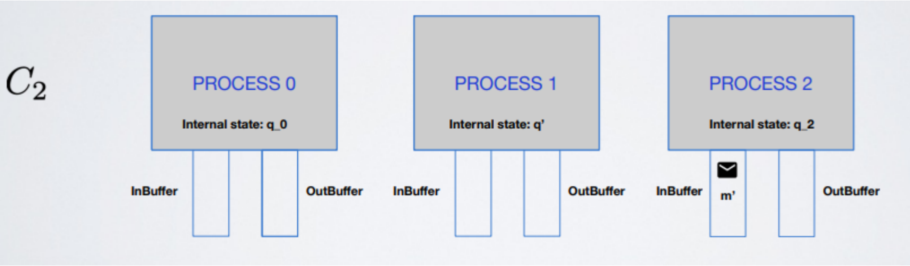
- $Del(m', 0, 2)$ is not enabled in C_0 because $OutBuff_0$ does not contain a message m'.
- $Exec(1)$ is enabled in $C_0 as\$Exec(0)$ and $Exec(2)$.

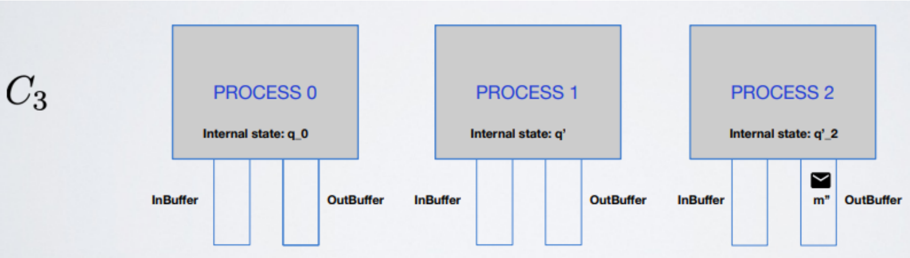
Execution: infinite sequence that alternate configurations and events
example: $(C_0, e_0, C_1, e_1, \dots)$ such that each event e_t is enabled in configuration C_t that is obtained by applying $e_{t-1} \to C_{t-1}$

$$\varepsilon : (C_0, e_0 = Exec(1), C_1, e_1 = Del(1, 2, m'), C_2, e_2 = Exec(2), C_3, e_3 = Del(2, 3, m'), \dots)$$

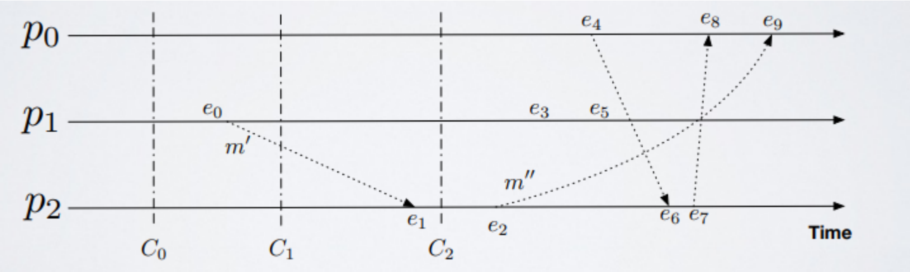








Plotting this in a graph $(time \times processes)$ we obtain something like:



> **DEFINITION** (Fair)
> E is fair if each process p_i executes an infinite number of local computation ($Exec(i)$ events are not finite) and each message m is eventually delivered (not possible to stall forever a message: there must exists a $Del(m, x, y)$).

Unfair executions brake any possible non-trivial algorithm so we will always consider fair executions.

When you create a process, the only thing that you know is the id of the other processes.

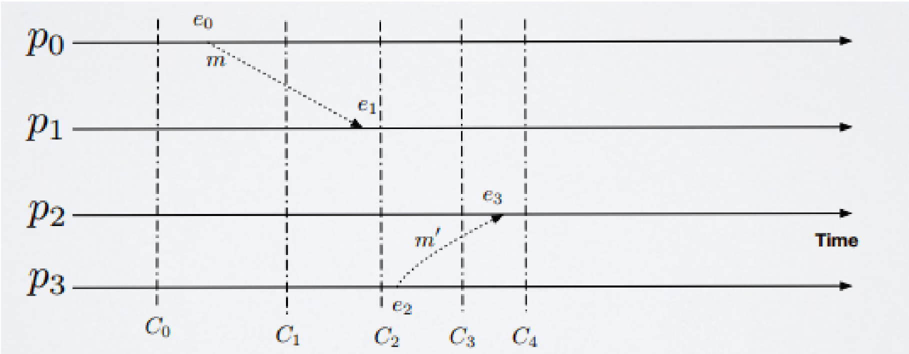
> **DEFINITION** (Local Execution (local view)
> Given an execution E and a process p_j , we define the local execution ($E|p_j$) of p_j the subset as the subset of events in E that impact p_j.

Considered:

$$\varepsilon : (C_0, e_0 = Exec(1), C_1, e_1 = Del(1, 2, m'), C_2, e_2 = Exec(2), C_3, e_3 = Del(2, 3, m'), \dots)$$

we have:

- $\varepsilon|p_1 = (Del(0, 1, m), \dots)$
- $\varepsilon|p_2 = (Del(3, 2, m'), \dots)$



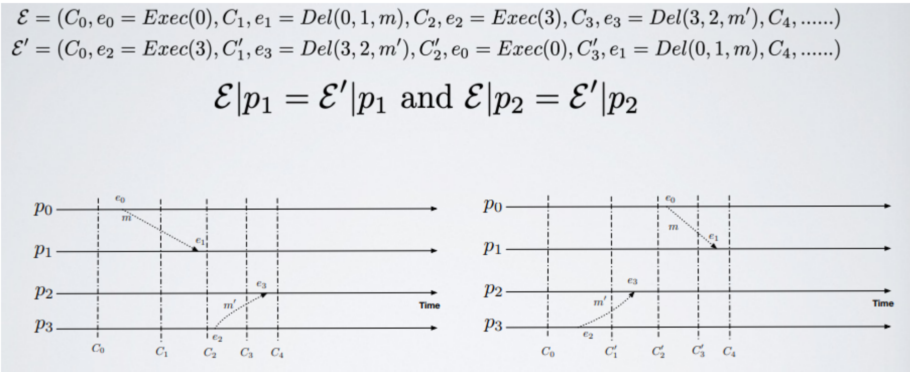
The only way to reach a status where all processes know informations about the others is communicate them with a row from p_2 to p_1 for example but in that case the event must end before the next one.

Different execution could give the same local execution:

$\varepsilon = (C_0, e_0 = Exec(0), C_1, e_1 = Del(0,1,m), C_2, e_2 = Exec(3), C_3, e_3 = Del(3,2,m'), C_4\,, \dots)$

$\varepsilon = (C_0, e_2 = Exec(3), C_1', e_3 = Del(3,2,m'), C_2', e_0 = Exec(0), C_3', e_1 = Del(0,1,m), C_4\,, \dots)$

$$\varepsilon|p_1 = \varepsilon'|p_1 \qquad\qquad \varepsilon|p_2 = \varepsilon'|p_2$$



To identify an execution all the process have to communicate them information to each others. In this case we say that p_1 and p_2 cannot distinguish E to E'.

> **DEFINITION** (Indistinguishability)
> $\varepsilon = (C_0, e_0 = Exec(0), C_1, e_1 = Del(0,1,m), C_2, e_2 = Exec(3), C_3, e_3 = Del(3,2,m'), C_4\,, \dots)$
> $\varepsilon = (C_0, e_2 = Exec(3), C_1', e_3 = Del(3,2,m'), C_2', e_0 = Exec(0), C_3', e_1 = Del(0,1,m), C_4\,, \dots)$
> if we have:
> $\forall p_j \in \Pi, \varepsilon|p_j = \varepsilon'|p_J$
> then we can say that E and E' are indistinguishable.

> **THEOREM**:
> In the asynch. model there is no distributed algorithm capable of reconstructing the system execution.