Distributed Systems

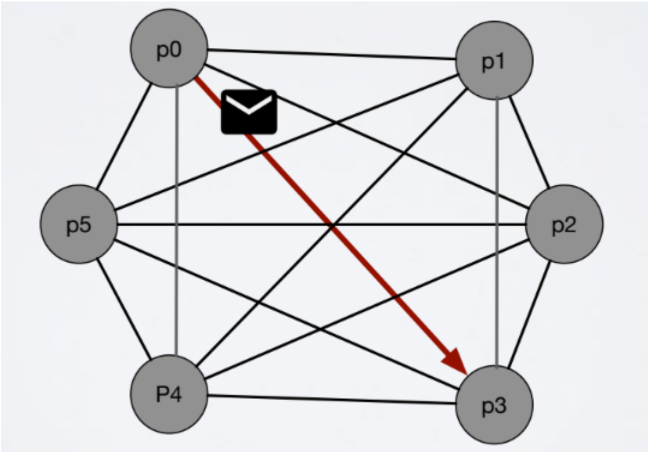
Raffaele Castagna

Academic Year 2025-2026

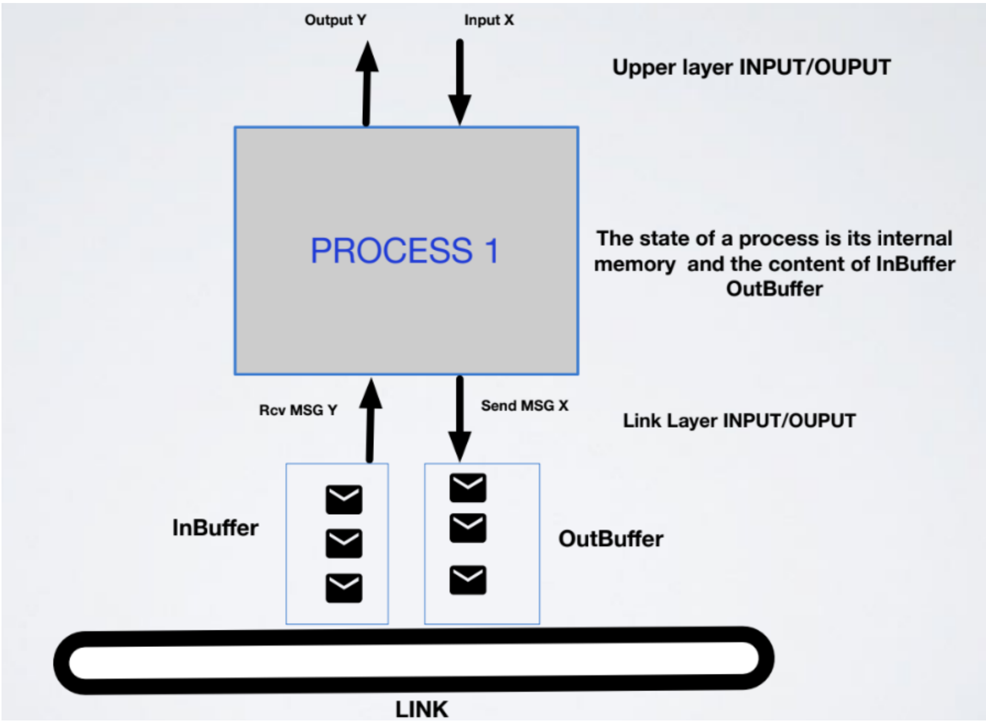
Indice

1 Introduction

Definition 1. *In a system we have n processes in* $\prod : p_0 \ldots, p_{n-1}$ *each with a distinct identity they communicate by utilizing a communication graph* $\boldsymbol{G}$ *:* $(\prod, E)$*, the communication is done by exchanging messages.*



Definition 2. *A process is a (possibily infinite) State Machine (I/O Automaton).*



Each process has multiple qualities:

- A set of internal states Q
- A set of initial states $Q_i \subset Q$
- A set of all possible messages M in the form ¡sender,receiver,payload¿
- Multiset of delivered messages $InBuf_j$
- Multiset of inflight messages $OutBuf_j$

We can formally describe this as follows: (this isnt part of the exam btw)

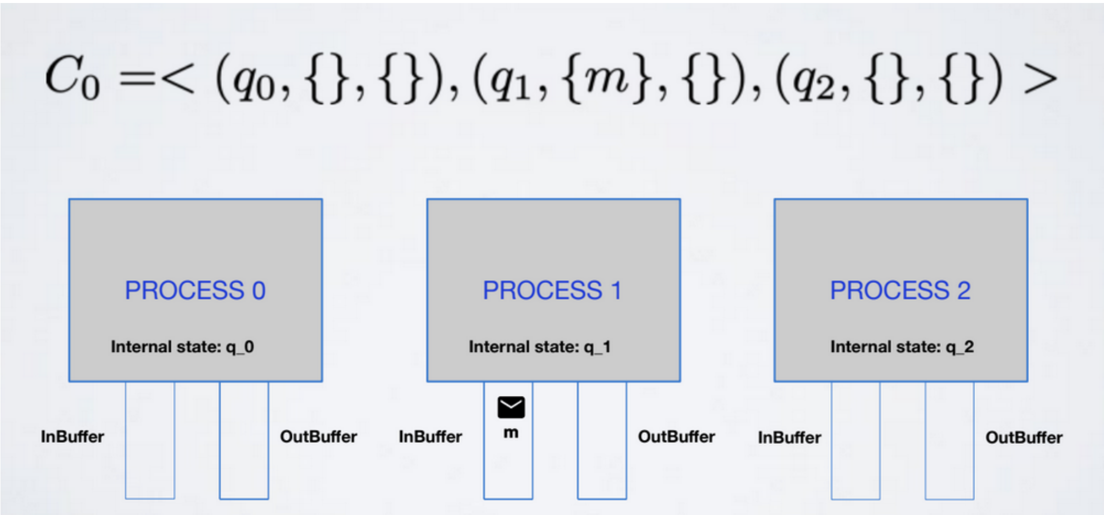
$$P_j(q \in Q \cup Q_{in}, InBuf_j) = (q' \in Q, SendMsg \subset M)$$

$$OutBuf_j = OutBuf_j \cup SendMsg$$

$$InBuf_j = \emptyset$$

To execute a process we have an adversary that schedules a set of events (scheduler), these events may be for example a delivery (e.g. Del(m,i,j)) or it can be one step of the step machine of process i (Exec(i))

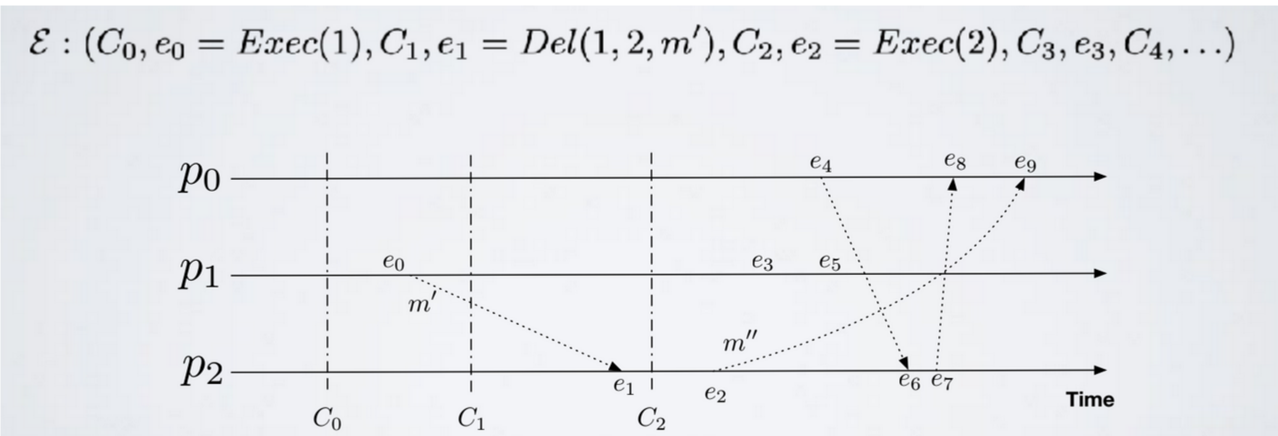
Definition 3. *A configuration C_t is a vector of n components, component j indicates the state of process j.*



An event is **enabled** in configuration c if it can happen.

Definition 4. *An execution is an infinite sequence that alternates configurations and events:* $(C_0, e_0, C_1, e_1, C_2, e_2, .$ *such that each event e_t is enabled in configuration C_t and C_t is obtained by applying e_{t-1} to C_{t-1}*

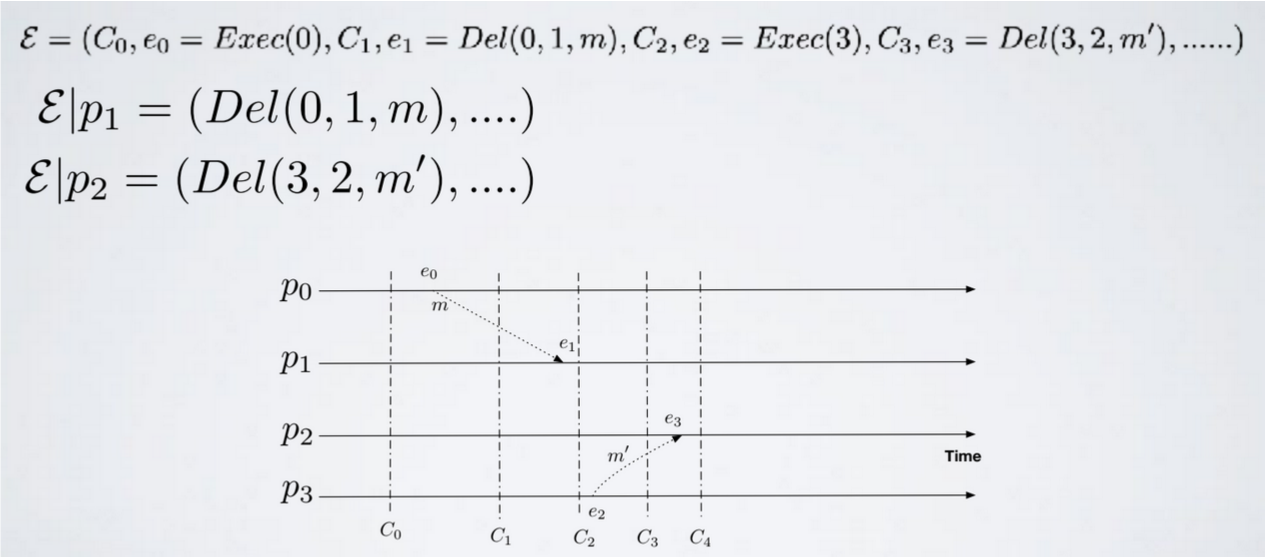
It may be useful to visualize how an execution involving multiple processes works, here we have an example:



Definition 5. *A **fair execution** is an execution E where each process p_i executes an infinite number of local computations (Exec(i) events are not finite) and each message m is eventually delivered (we can't stall messages)*

We will always use fair executions unless stated otherwise.

Definition 6. *Given an execution E and a procces p_j, we define the local view/ local execution of $E|p_j$ the subset of events in E that impact p_j*



But these executions do not account for time, so we may have executions that are the same even though the events happened at different times, in case this does happen, we say that two executions are ***indistinguishable***.

Theorem 1. *In the asynch. model there is no distributed algorithm capable of reconstructing the system execution.*

1.1 Synchronous Vs Asynchronous

We have 3 main types of synchrony:

- Asynchronous Systems
- Eventually Synchronous Systems
- Synchronous systems

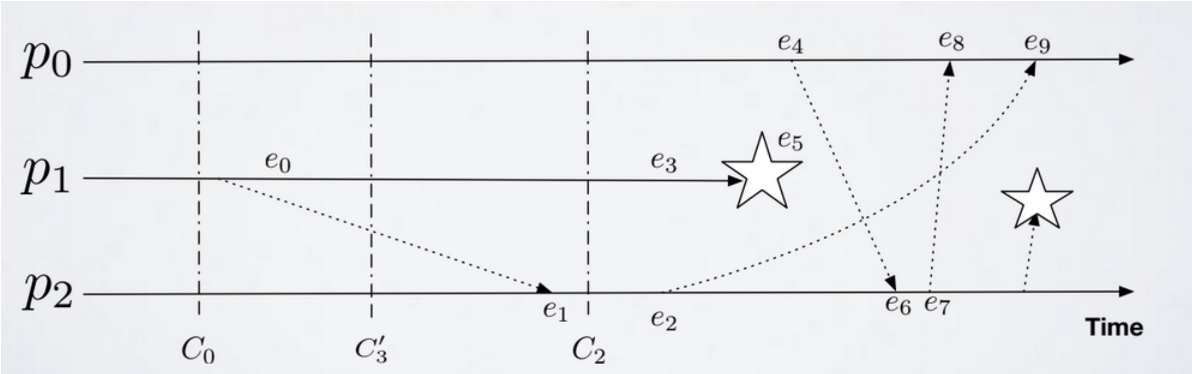
We can say that is a system is synchronous if it has a fixed bound on the daly of messages, on the time of actions executed by processes, and a fixed bound between execution of actions.

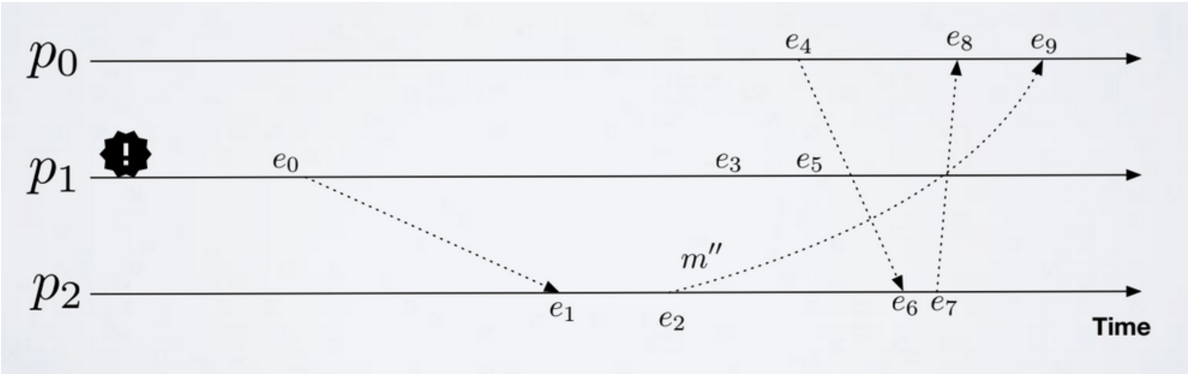
1.2 Failures

We have 2 main models for failures:

- Crash-stop Failures (The program crashes, and doesnt respond)
- Byzantine Failures (The behaviour of the program is random)

We signal crash failures with a star sign and byzantile failures with a !





Byzantine failures are a superset of Crashstop failures, so algorithms that will work on byzantine failures will always work on crash stop failures, but not the contrary.
A process is **correct** if it does not experience a failure, every algorithm has a maximum number f of failures that i can experience.