Modern Distributed Computing

Theory and Applications

Ioannis Chatzigiannakis

Sapienza University of Rome

Lecture 5
Tuesday, April 9, 2013

Part 2: Failures

1. Link failures, Node failures, Impossibility Results
2. Agreement
3. Byzantine Failures
4. Failures in Asynchronous Systems

Byzantine Failures

- The network includes faulty processes that do not terminate but continue to participate in the execution of the algorithm.
- The behavior of the processes may be completely unpredictable.
- The internal state of a faulty process may change during the execution of a round arbitrarily, without receiving any message.
- A faulty process may send a message with any content (i.e., fake messages), independently of the instructions of the algorithm.
- We call such kind of failures as Byzantine failures.
- We use byzantine failures to model malicious behavior (e.g. cyber-security attacks).

Why study Byzantine Fault Tolerance?

- Does this happen in the real world?
  The "one in a million" case.
  - Malfunctioning hardware,
  - Buggy software,
  - Compromised system due to hackers.
- Assumptions are vulnerabilities.
- Is the cost worth it?
  - Hardware is always getting cheaper,
  - Protocols are getting more and more efficient.

Coordinated Attack of 4 Byzantine Generals

Four generals wish to coordinate the attack of their armies in an enemy city. Among the generals there exits a traitor. All loyal generals must agree to the same attack (or retreat) plan regardless of the actions of the traitor. Communication among generals is carried out by messengers. The traitor is free to do as he chooses.

- Consensus problem in a system with $n = 4$ processes under the presence of byzantine failures.
- Possible input/output values are "yes" or "no" – that is $S = \{"\text{yes}", "\text{no}"\}$

Problem Statement

- On general achieves the role of Chief of Staff.
- The Chief of Staff has to send an order to each of the $n - 1$ generals such that:
  1. All faithful generals follow the same order
     (all non faulty processes receive the same message)
  2. If the Chief of Staff is faithful, then all faithful generals follow his orders
     (if all processes are non-faulty then the messages received are the same with the transmitting process)
- The above conditions are known as the conditions for "consistent broadcast".
- Note: If the Chief of Staff is faithful, then the 1st condition derives from the 2nd. But he may be the traitor.

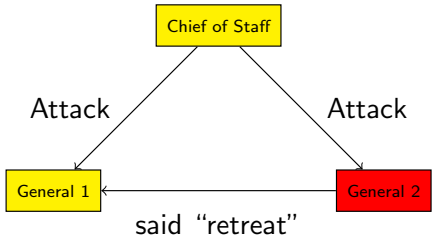
Discussion

- A solution for the Byzantine Generals problems allows:
  1. Reliable communication in the presence of tampered messages
  2. Reliable communication in the presence of message omissions
- Dealing with message omissions (link/stopping failures) is the most common approach.
- We name faults Byzantine all faults that fall under these two categories.
- All solutions to the problem require a network size at least three times the number of failures – that is $n > 3\beta$.
  - Different situation from stopping failures where n and σ did not follow any relationship.
  - May sound surprising high, due to the *triple-modular redundancy* – that states that $n > 2\beta + 1$.

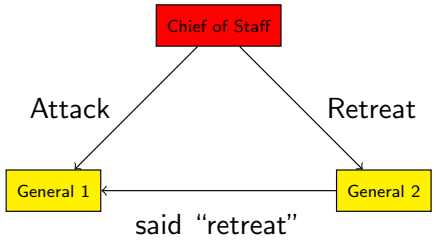
Impossibility result

Let's examine the following cases involving 3 generals:

Case #1



Case #2



- In case #1, General 1 in order to meet the 2nd condition, he has to attack.

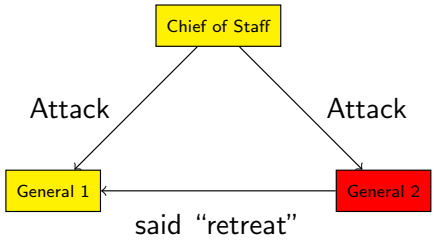
2nd Condition

If the Chief of Staff is faithful, then all faithful generals follow his orders.

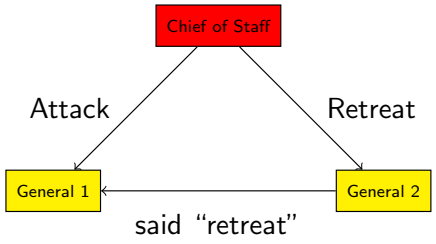
Impossibility result

Let's examine the following cases involving 3 generals:

Case #1



Case #2



- In case #2, if General 1 attacks then he violates the 1st condition.

1st Condition

All faithful generals follow the same order

Impossibility result

- Given the messages received by General 1, each case looks symmetric.
- General 1 cannot break the symmetry.
- No solution exists for the Byzantine Generals in case of 3 generals and 1 traitor.
- Generalization of the impossibility result:
  No solution exists for less then $3\beta + 1$ generals if it has to deal with β traitors.

Lamport, Shostak and Pease Algorithm

L. Lamport, R. Shostak, M. Pease: "The Byzantine Generals Problem", ACM Transactions on Programming Languages and Systems, 4(3): pp 382-401, 1982.

- The algorithm makes three assumptions regarding communication:
  1. All message transmissions are delivered correctly.
  2. The receivers knows the identity of the sender.
  3. The absence of a message can be detected.
- The 1st and 2nd assumptions limit the traitor from interfering with the transmissions of the other generals.
- The 3rd assumptions prevents the traitor to delay the attack by not sending any message.
- In computer networks conditions 1 and 2 assume that the processors are directly connected and communication failures are counted as part of the β failures.

Lamport, Shostak and Pease Algorithm

- Let n processes and β failures.
- Processes have a predefined decision o_{def} that is used when the Chief of Staff is a traitor (e.g., retreat).
- We define function majority$(o_1, \ldots, o_{n-1}) = o$ that computes the majority of decisions $o_u = o$

Algorithm UM(n,0) (for 0 traitors)

1. The Chief of Staff transmits decision o to all generals.
2. All generals decide o or if they do not receive a message, they decide o_{def}.

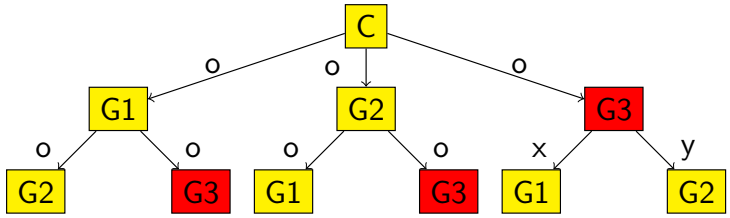
Lamport, Shostak and Pease Algorithm

Algorithm UM(n, m) (for m traitors)

1. The Chief of Staff transmits decision o to all generals.
2. For each general u
   - Set o_u to the value received, or if no message received, set to o_{def}.
   - Send the value o_u to the $n-2$ generals by invoking UM($n-1, m-1$).
3. For each general u and each $v \neq u$
   - Set o_v to the value received from u at step 2, or if no message received set to o_{def}.
   - Decide on value majority($o_1, \ldots, o_{n-1}$).

Example of Execution

$n = 4, \beta = 1$ – G3 is the traitor



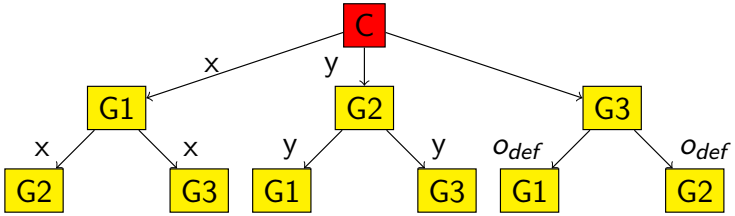
- At the end of 1st phase: G1 ($o_1 = o$), G2 ($o_2 = o$), G3 ($o_3 = o$)
- At the end of 2nd phase:

  G1 – $o_1 = o, o_2 = o, o_3 = x$
  G2 – $o_1 = o, o_2 = o, o_3 = y$
  G3 – $o_1 = o, o_2 = o, o_3 = o$

- At the end of 2nd phase, each general has the same number of values and reaches the same decision due to condition 1.
- The decision of the Chief coincides with the majority (2nd condition)

Example of Execution

$n = 4, \beta = 1$ – the Chief of Staff is the traitor



- At the end of 1st phase: G1 ($o_1 = x$), G2 ($o_2 = y$), G3 ($o_3 = o_{def}$)
- At the end of 2nd phase:

  G1 – $o_1 = x, o_2 = y, o_3 = o_{def}$
  G2 – $o_1 = x, o_2 = y, o_3 = o_{def}$
  G3 – $o_1 = x, o_2 = y, o_3 = o_{def}$

- The three loyal generals decide majority(x, y, o_{def}) thus both 1st and 2nd conditions are met.

Lemma

For any m and k, UM(m) adheres the 2nd condition given $2k + m$ generals and at most k traitors.

Proof: (By induction on m)
In the 1st step, UM(0) works if the Chief of Staff is loyal, i.e. UM(0) meets the 2nd condition.
Let's assume that UM($m-1$) meets the 2nd condition for $m > 0$.
We can show that it holds for m:

- In the 1st step, the loyal general sends the value o to $n-1$ generals.
- In the 2nd step all loyal general execute UM($m-1$).
- From the original assumption it holds that $n > 2k + m$ or $n - 1 > 2k + (m-1)$.

- From the induction step that we defined, each loyal general u receives $o_u = o_v$ from each loyal general v.
- Since there are at most k traitors and $n-1 > 2k + (m-1) \geq 2k$, i.e., $k < \frac{n-1}{2}$ then the majority is reached from the $n-1$ loyal generals.
- Thus each loyal general has $o_u = o$ for majority of $n-1$ values – thus in the 3rd step, by invoking majority$(o_1, \ldots, o_{n-1})$ it outputs o that meets the 2nd condition.

■

Theorem

For any m, UM(m) adheres the 1st and 2nd condition given 3m generals and at most m traitors.

Proof: (By induction on m)
If no traitors exists it is easy to show that with the user of the algorithm, in the 1st step, conditions 1 and 2 hold.
If we assume that UM$(m-1)$ meets conditions 1 and 2 for $m > 0$.
We can show for m:
Case 1:

- Assume the Chief of Staff is loyal.
- For $k = m$ due to the Lemma, UM(m) meets the 2nd condition.
- Since the 1st condition derives from the 2nd condition when the Chief of Staff is loyal, it is enough to show the second case:

Case 2:

- The Chief of Staff is a traitor.
- There exist at most m traitors and the Chief of Staff is among them.
- Thus, at most $m-1$ generals are traitors.
- Since we have $3m$ generals, the loyal generals must be $3m - 1 > 3(m-1)$
- Therefore we can apply the inductive step and conclude that UM$(m-1)$ meets the 1st and 2nd condition.
- Thus for each v, each pair of loyal generals receives the same value o_v in the 3rd step.
- Thus, each pair of loyal generals receives the same number of values and thus majority$(o_1, \ldots, o_{n-1})$ returns the same value – which meets the 1st condition.

■

Properties of Algorithm

- By applying UM(n,β) we get $n-1$ messages
- For each message the UM(n,$\beta-1$) is activated that generates $n-2$ messages
- ...
- The total number of messages is $\mathcal{O}(n^{\beta+1})$
- The $\beta+1$ steps during which messages are exchanged between the processes is a mandatory feature of algorithms that need to reach consensus in the presence of β faulty processes.

Study of Synchronous Message Passing Model

- The assumption of synchronous execution does not reflect the real conditions of operation of distributed systems.
- However, it allows us to understand some fundamental aspects.
  - The assumption of coordinated execution of algorithm's steps,
  - The assumption of simultaneous delivery of all messages across all channels.
- We investigated the correctness and performance of protocols
  - in terms of number of rounds – time complexity
  - in terms of number of message exchanges – communication complexity

Study of Synchronous Message Passing Model

- We define the network as a graph $G = (V, E)$:
  - comprised of a finite set V of points – the vertices – representing the processing untis (i.e., processes) – $n = |V|$
  - a collection E of ordered pairs of elements of V ($E \subset [V]^2$) – the edges – representing the communication channels of the network – $m = |E|$
- Each process $u \in V$ is defined by a set of states $states_u$
  - A nonempty set of states $start_u$, known as starting states or initial states.
  - A nonempty set of states $halt_u$, known as halting states or terminating states.
- Each process uses a message-generator function $msgs_u : states_u \times nbrs_u^{out} \rightarrow M \cup \{\texttt{null}\}$
- Uses a state-transition function $trans_u : states_u \times (M \cup \{\texttt{null}\})^{nbrs_u^{in}} \rightarrow states_u$

Study of Synchronous Message Passing Model

- All processes, repeat in a "synchronized" manner the following steps:

  1^{st} Step

  1. Apply the message generator function.
  2. Generate messages for each outgoing neighbor.
  3. Transmit messages over the corresponding channels.

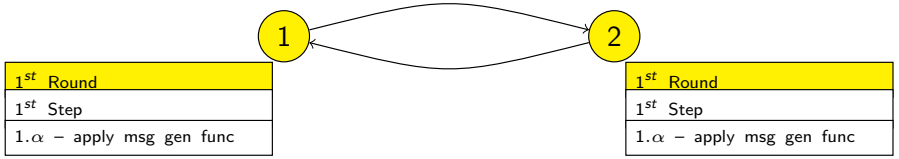
  2^{nd} Step

  1. Apply the state transition function.
  2. Remove all incoming messages from all channels.
- The combination of these two steps is called a round (of execution).

Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

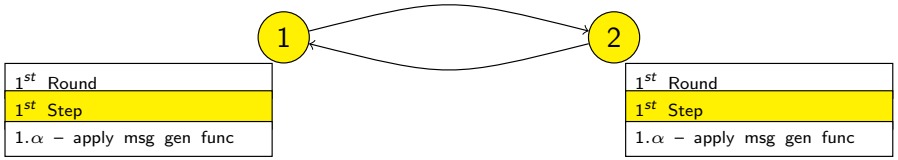
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

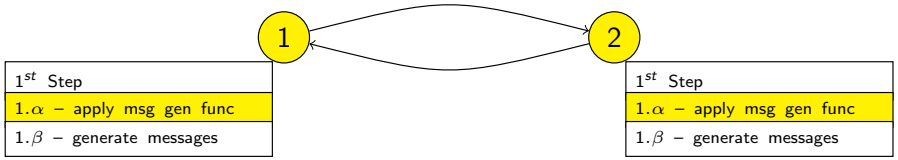
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

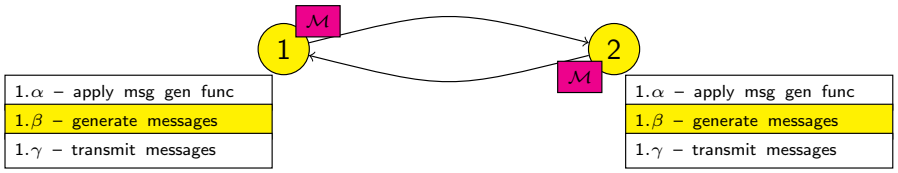
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

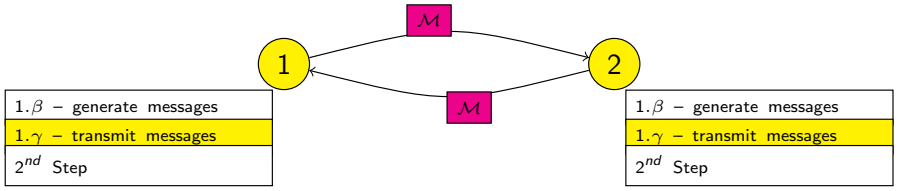
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

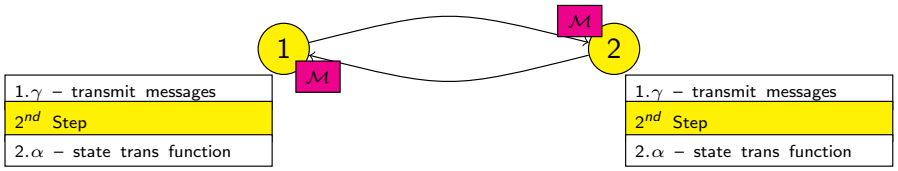
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

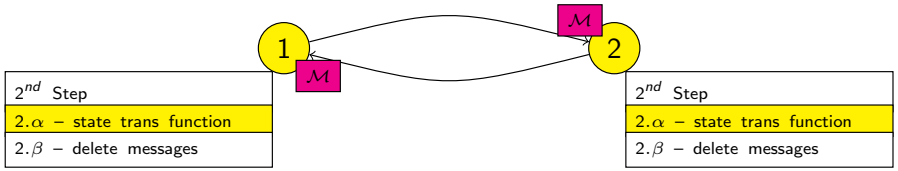
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

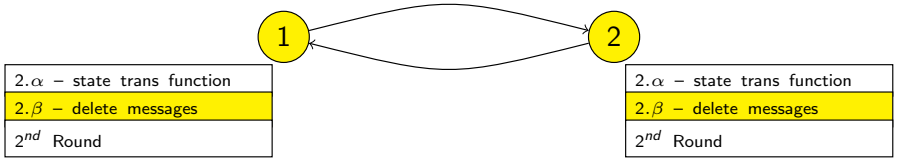
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

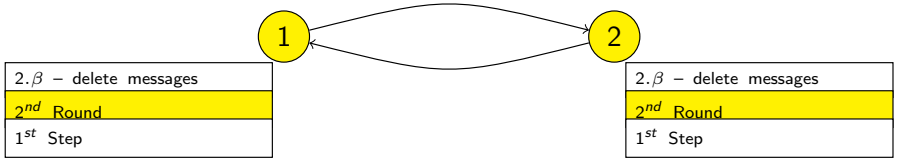
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

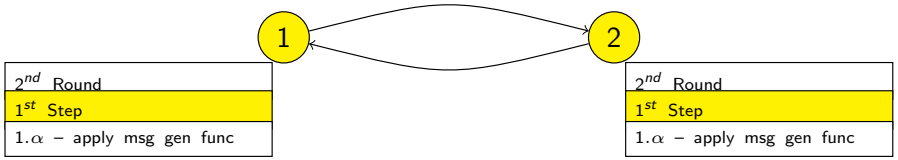
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

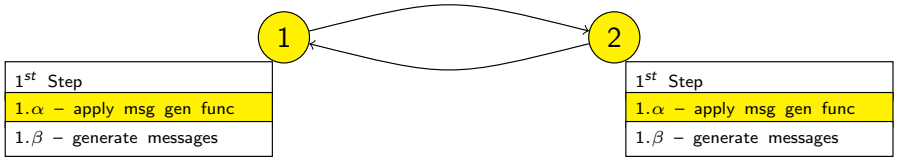
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

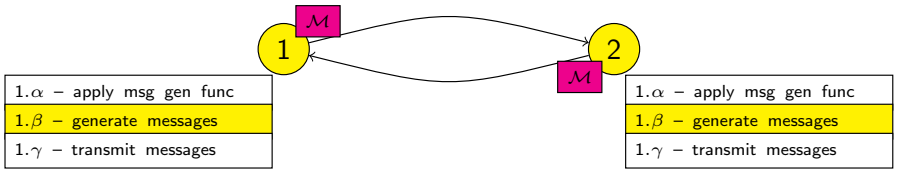
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

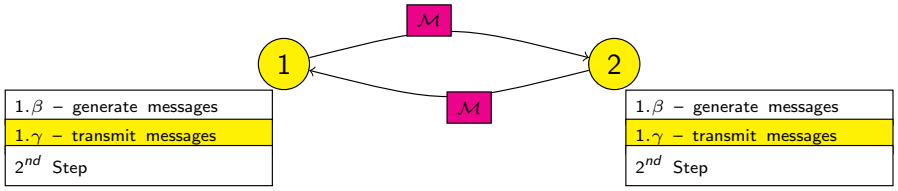
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

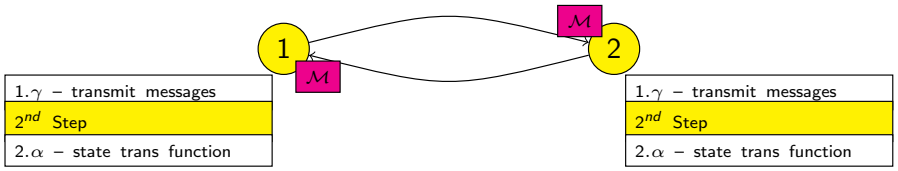
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

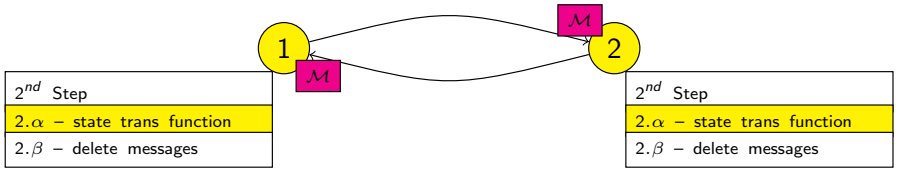
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

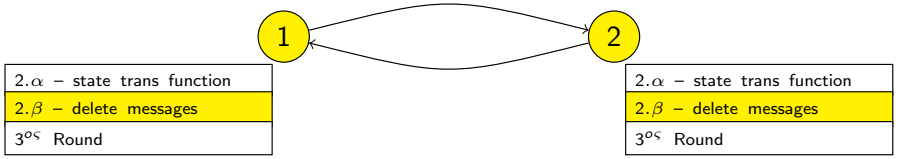
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

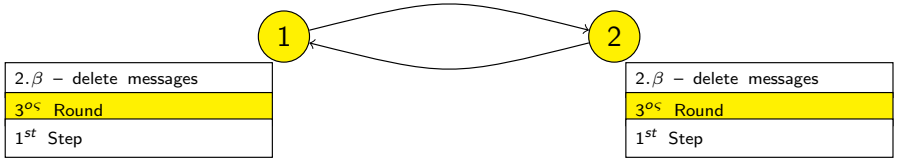
Execution of Synchronous System



Example of execution of a Synchronous System

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- the processes execute in a "synchronized" manner the protocol.

Execution of Synchronous System



Example of execution with processors of variant speed

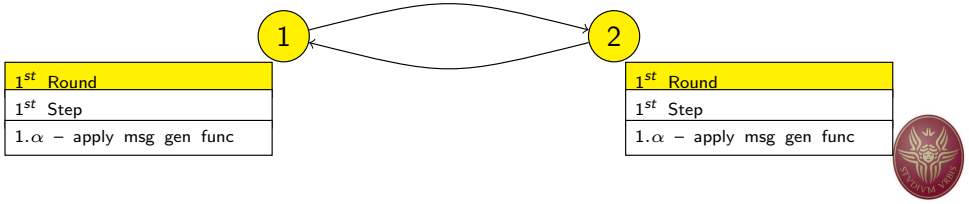
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

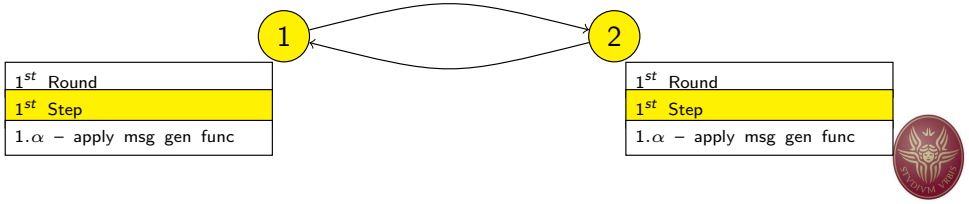
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

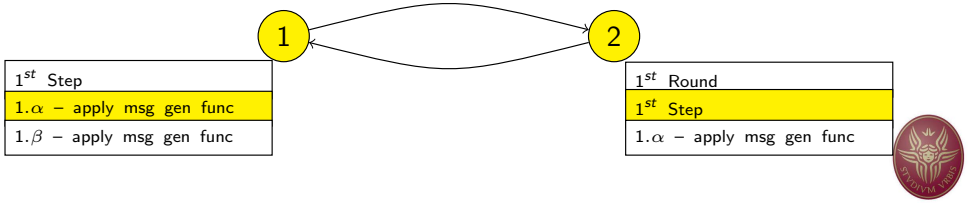
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

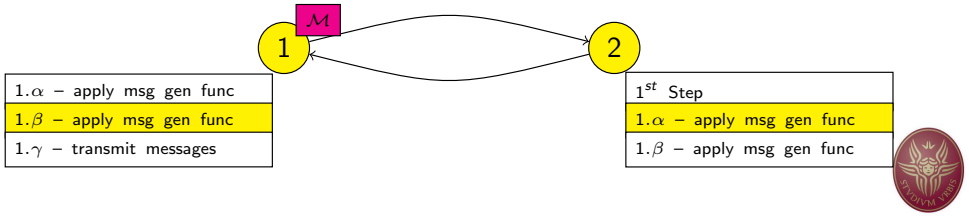
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

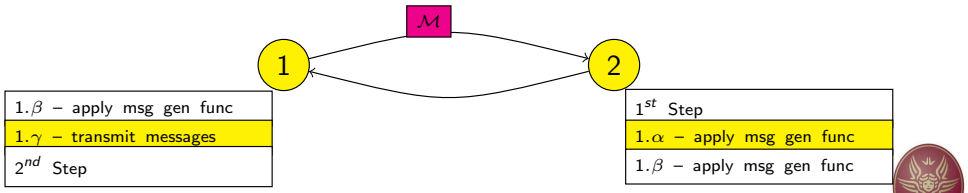
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

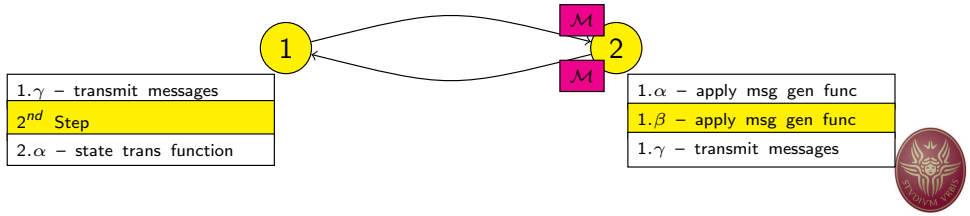
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

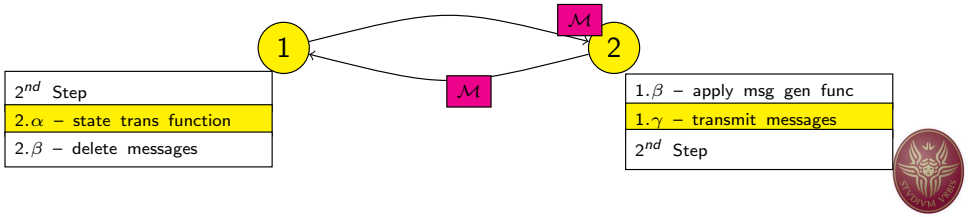
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

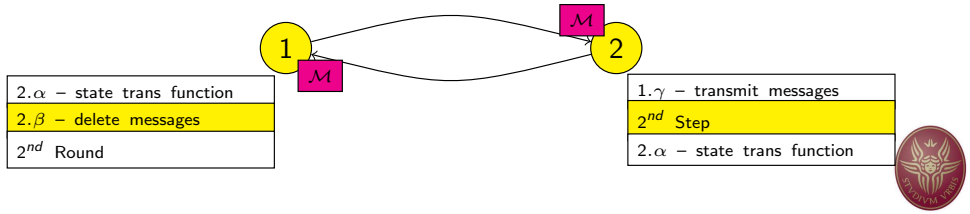
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

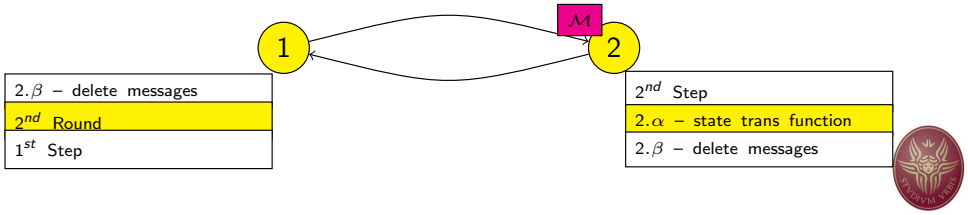
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

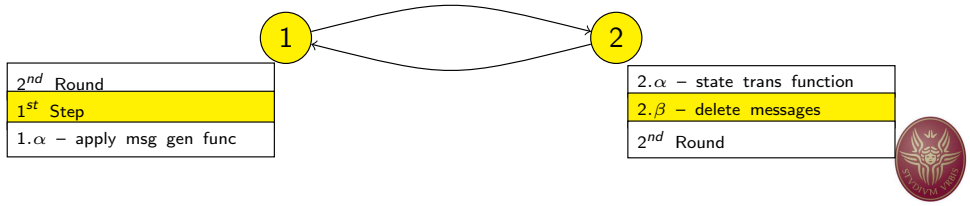
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

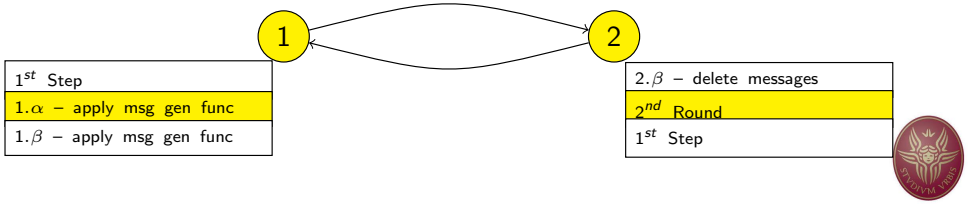
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

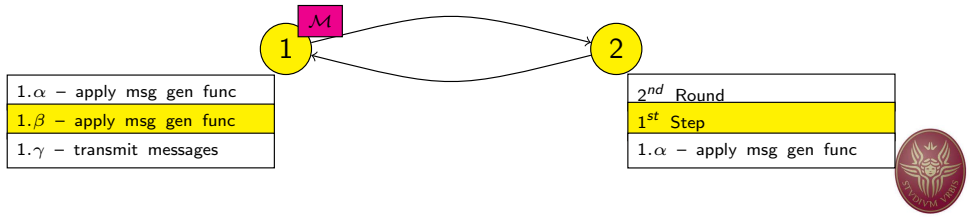
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

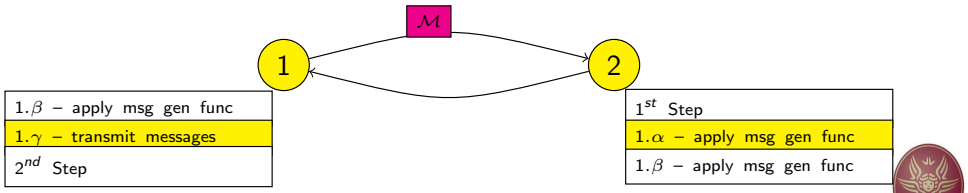
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

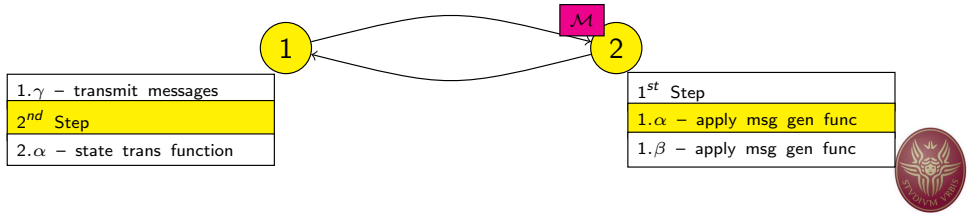
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

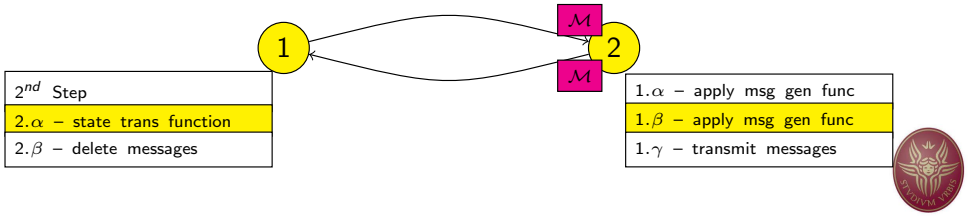
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

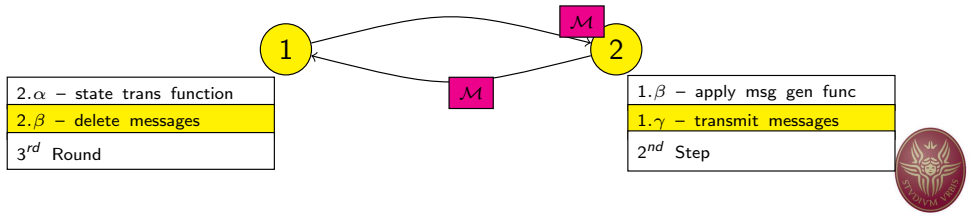
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

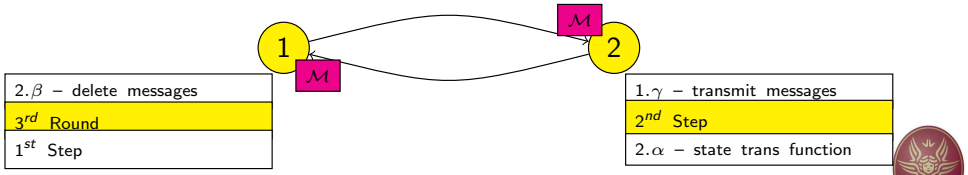
The synchronous message passing model assumes that all processors have identical properties (e.g., speed).

- therefore, processes, execute in a coordinated manner the protocol.

What if the processors do not have common speeds?

- some processes execute the protocol steps slower than the others.

Executions by Processors of different speed



Example of execution with processors of variant speed

- The execution of process 2 is slower.
- How are messages handled?
  - Do we assume a queue where incoming messages are inserted ?
  - How many messages can it store?
  - What if the queue is short, are messages lost ?
- Process 1 may delete a message before it's being processed.
- What other anomalies will occur in such a system ?

Example of execution with channels of variant speed

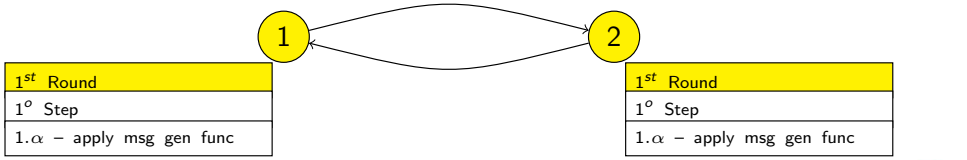
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

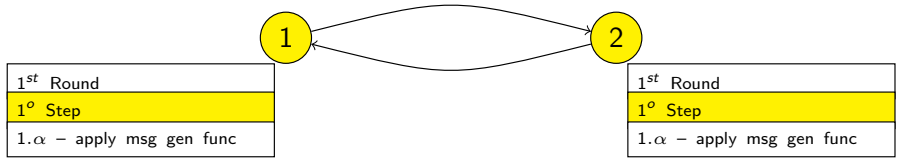
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

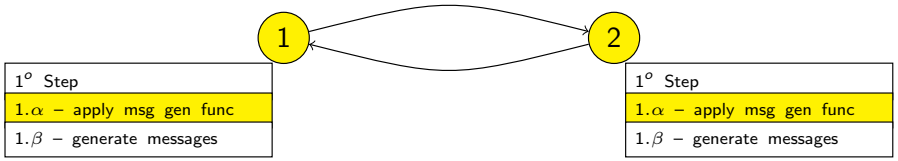
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

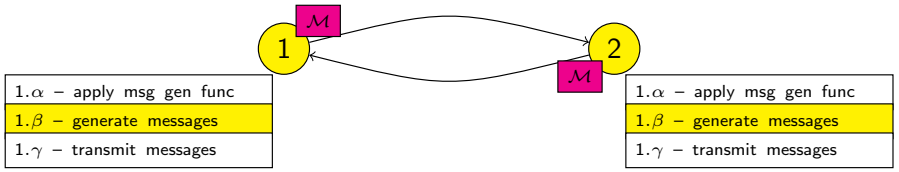
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

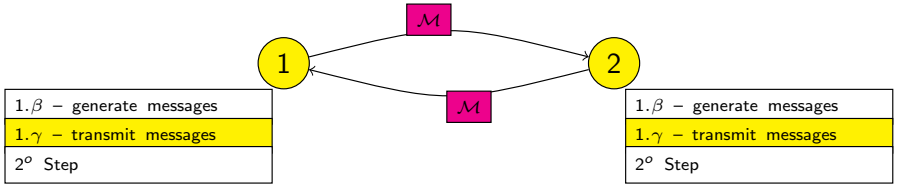
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

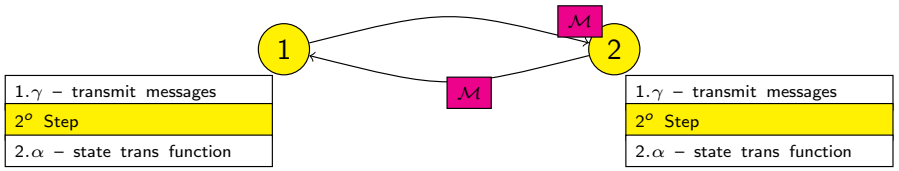
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

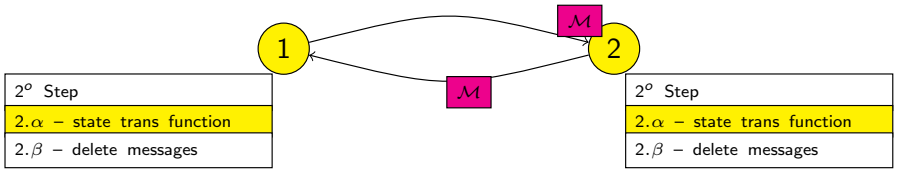
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

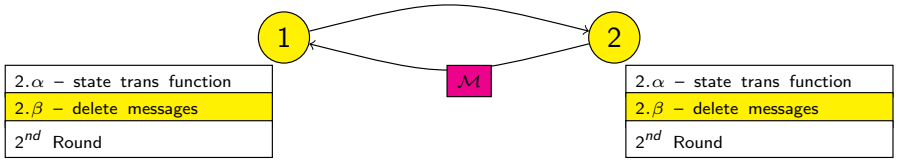
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

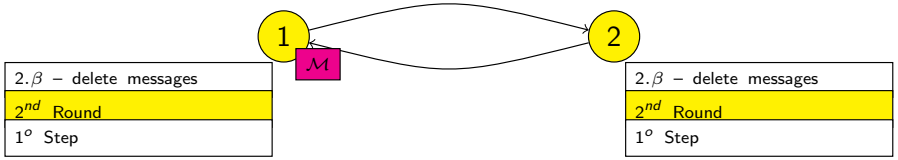
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

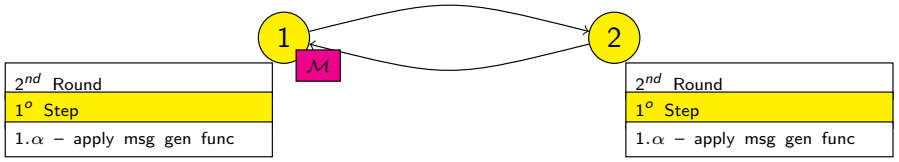
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

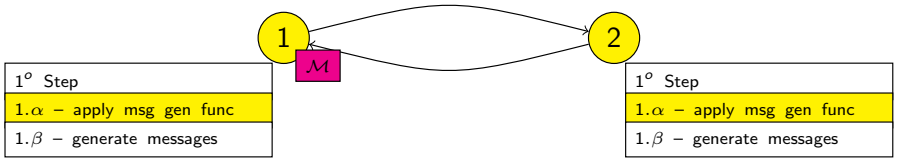
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

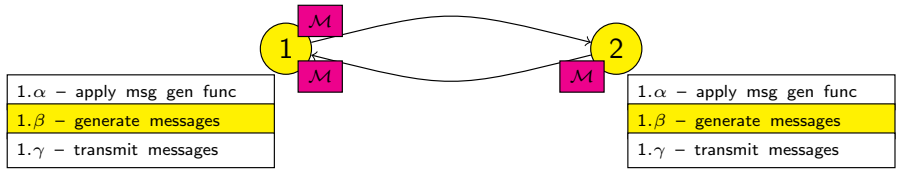
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

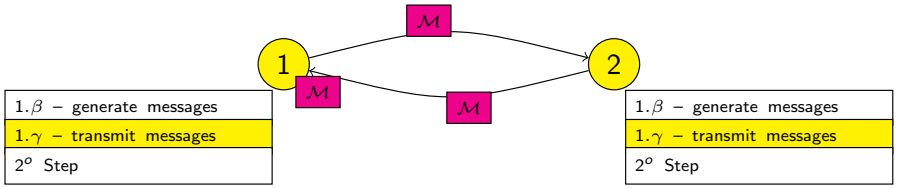
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

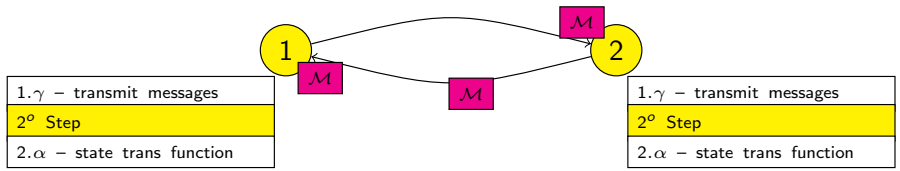
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

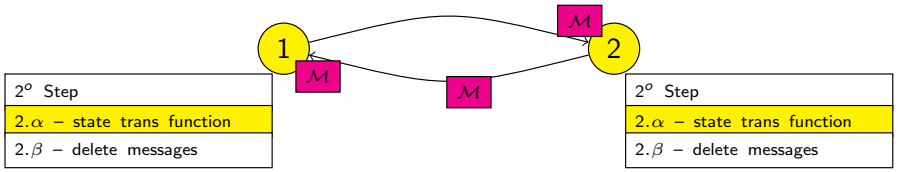
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

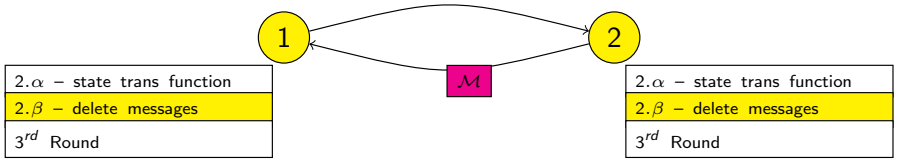
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

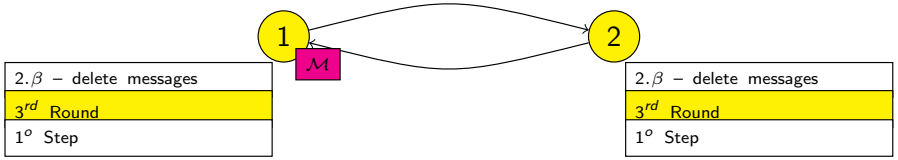
The synchronous message passing model assumes that all communication channels have identical properties (e.g., speed).

- therefore, messages, are delivered simultaneously.

What if the communication channels do not have common speeds?

- some messages are delivered later than the others.

Execution with Communication Channels of different speed



Example of execution with channels of variant speed

- The communication channel connecting process 2 with process 1 is slower than the communication channel connecting process 1 with 2.
- What if a process tries to send a message while the channel is transmitting another message?
  - Do we assume a queue for outgoing messages ?
  - How many messages can be stored in the queue?
  - In case of a short queue, are messages lost ?
- What other anomalies will occur in such a system ?

Asynchronous Message Passing Systems

- Systems of Independent Interconnected Entities
- Interconnection allows coordination
- Coordination allows solving common goals (e.g. Spanning Trees, DB transactions)

Problem

To appropriately describe the way that the entities coordinate for a systemic generation of desired properties.

Distributed Algorithms: Properties

- Dictates how entities coordinate.
- The description of the algorithm defines the actions of each entity
- . . . and how they communicate.
  - It is identical for each entity.
- Is part of the systemic solution of the given problem.
  - properly deals with each instance of the problem (correctness)
- Is evaluated based on the number of actions required by each entity and the number of messages generated (time & communication complexity)

Distributed Algorithms: Weaknesses

- Time Free Systems: Does not allow any a priori assumption on the speeds by which actions are being completed by each entity or the speeds of delivering each message transmitted over each channel.
- Each entity is invoked to execute the appropriate action with partial knowledge on the global state of the system and the progress achieved so far (in the execution)
- Real Life: A large number of undetermined factors affecting the system:
  - messages lost, entities stop/restart, entities act arbitrarily (or maliciously), entities "losing"' their state . . .
- Since the factors are undetermined they impose a non-deterministic execution of the system that needs to be addressed in order to end up with deterministic results.

Asynchronous Message Passing Model

We study distributed systems where

1. the entities of the system execute their actions
   - with **arbitrary** order,
   - and with **arbitrary** speed relative to the other entities,
   - we do not assume any rate of execution actions.
2. the communications channels of the system deliver messages with **arbitrary** speeds relative to the other channels
   - we do not assume any message delivery rate.

Asynchronous Message Passing Model

- We model this undetermined temporal behavior using Input/Output automata
  - Each process is modeled as an I/O automaton,
  - Each communication channel is modeled as an I/O automaton.
- The I/O automata model is generic enough.
  - We can use it to describe almost all types of asynchronous message passing systems.

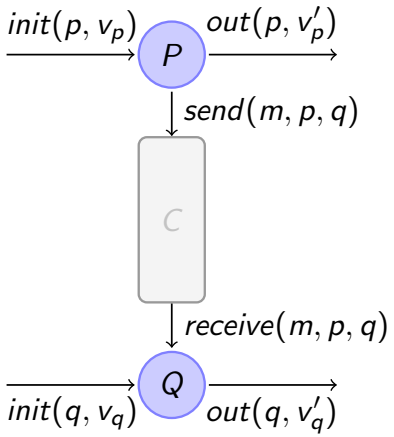
Input/Output Automata

- An I/O automaton models an entity of the distributed system that interacts with other entities of the system.
- It is a state automaton (state machine) where transitions between states are connected by a set of actions.
- The actions are grouped:
  1. Input Actions
  2. Output Actions
  3. Internal Actions

Input/Output Automata

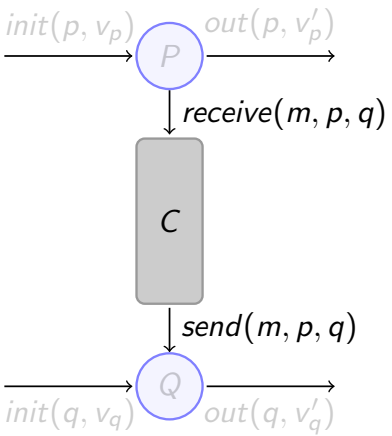
- The input and output actions are used to model the communication of the automata with their environment
  - Example – an input action is the reception of a message by a neighboring process.
  - Example – an output action is the delivery of a message from a communication channel.
- Internal actions are visible only by the automaton performing the action.
- An automaton does not determine when an input action will be invoked – this depends on its neighboring automata.
  - It can only determine when an output action or and internal action will be executed.

Modeling Processes



- **I/O automata**
- Defined by a state
  - Special states: initial, halt states
- and a state transition function: $states \times (message_{in}, process) \rightarrow states \times (message_{out}, process)$
- Some tasks may be executed internally $states \times \{(message_{in}, process) \cup \emptyset\} \rightarrow states \times \{(message_{out}, process) \cup \emptyset\}$
- We need to define the notion of fareness

Modeling Communication Channels



- **I/O automata**
- Defined by a state
  - We assume a queue of messages that need to be delivered
  - e.g., a FIFO priority queue
- Execute actions $receive(m, p, q)$ and $send(m, p, q)$

To stop or not to stop waiting

In a realistic asynchronous system, situations appear with very similar properties which however need to be addressed with different actions

- Messages sent by a process are lost before being properly received
- A process executes an action very slowly
- A process fails and terminates

Common property

The existence of the process is not evident by the rest of the network.

The common property makes it difficult (if not impossible) to differentiated between each different case.

Still the differentiation is necessary to provide correct results.

To stop or not to stop waiting

Since a process does not respond we are faced with the dilemma if the rest of the network should wait or not. The problem is the combination of the time free and fault vulnerable aspects of the system. Are we mistaken by mixing up these two properties ?

- A time free model abstracts the operation conditions of most real systems.
- Failures occur without being able to predict them or avoid them, at different levels (e.g., hardware failures, software bugs. . .)

So how can we differentiated between a slow system (time free) or a faulty system (fault vulnerabilities)?

Consensus: Problem Definition

Consensus

A process v has an initial value $init_v \in \{0, 1\}$. The goal of the system is that all processes agree on a common value.

- e.g., in a DB transactions 1=accept/commit, 0=reject/rollback
- The algorithm can be viewed as a decision system with the initial values as the premises and the common value its goals.
- We wish the results to be characterized by

  Soundness: Common value "accept" only if $\forall v \ init_v = 1$

  Completeness: If $\forall v \ init_v = 1$ then the only common value is "accept"

Consensus: Problem Definition

We relax the assumption in a way such that the decision reaches a common value:

Consensus

Each process v has an initial value $init_v \in \{0, 1\}$. Goal of all processes is to decide a common value matching the following criteria:

- The decision is common for all processes.
- For any decision value $v \in \{0, 1\}$, there is some initial state that leads to this common value v

Consensus with One Faulty Process: Impossibility Result

- We use the concept of a configuration

Proof Idea

1. Let an algorithm P that solves the problem
2. An initial configuration exists that may lead to any decision value (non trivial)
3. Given a configuration C, at some point the delivery of a message m_1 bye process p will determine the decision value
4. Each such message is delayed while another set of messages are delivered which do not fix the decision value
5. Later on m_1 is delivery which fails to determine the decision value. Why?

Consensus with One Faulty Process: Impossibility Result

Proof Idea

- If we do not delivery the message (faulty process) the algorithm will eventually terminate after a sequence of steps S during which p does not participate.
- Since C may lead to any decision value, the termination may determine either `dec_value=0` or 1.
- However if we delay the delivery too long, the algorithm will still terminate. The delivery of m_1 at this point will not finalize the decision value since the algorithm has already terminated to any value.

Consensus with One Faulty Process: Impossibility Result

Proof Idea

- Process p does not participate in S. Therefore if first steps S occur and then the message of p is delivered will lead to the same configuration configuration where m is first delivered and then S occur.
- Therefore in the second scenario, p remains in the execution and the decision value is not finalized.

Consensus with One Faulty Process: Impossibility Result

Definitions

Definition
A configuration of a system is a set of all processes states.

Definition
Two configurations are called neighboring if they differ in the state of a single process.

Definition
The event $e = (p, m)$ is the delivery of message m to process p, and leads to the transition between two configurations.

Consensus with One Faulty Process: Impossibility Result

Definitions

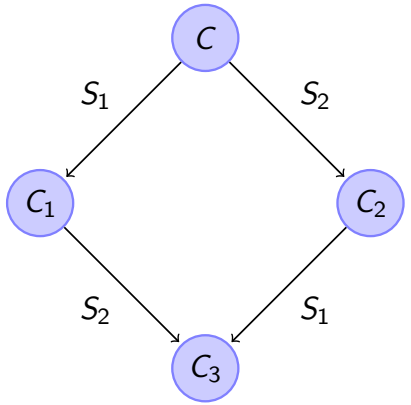
Definition
A configuration C is reachable by another configuration C' if a sequence of events S may lead from C' to C.

Definition
A configuration C is bivalent if the set of all terminating configurations that are reachable from C includes configuration where `dec_value=0` and configurations with `dec_value=1`. Otherwise C is univalent or i-valent depending on `dec_value`.

Consensus with One Faulty Process: Impossibility Result

Lemma
Let configuration C and sequence of events S_1, S_2 that lead to configuration C_1, C_2 respectively. If the set of processes that participate in S_1 and S_2 are disjoint then S_2 may be applied to C_1 and S_1 to C_2, such that the execution will lead to a common configuration C_3.

Consensus with One Faulty Process: Impossibility Result



$$S_2(S_1(C)) = S_1(S_2(C))$$

Consensus with One Faulty Process: Impossibility Result

Lemma

Every algorithm A that solves the problem has at lease on bivalent initial state.

- In contrast to the case where no failures occur.
- In that case each initial state leads to a unique decision value
  - The decision is "known" from the very start. It is up to the algorithm to coordinate the processes towards reaching this common decision.
- The inherent non-determinism is due to the unpredictable failures
  - If the part of the network that will force decision "1" fails, the execution will decide 0
  - However, we do not know if it will fail ...
- Can we prove this?

Consensus with One Faulty Process: Impossibility Result

Proof: Let assume that no such configuration exists. Then:

1. It will include 0-valent and 1-valent initial configuration since both decisions are possible.
2. We order the initial configuration such that every pair of configurations is neighboring
3. Thus a 0-valent C_0 configuration will be neighboring a 1-valent configuration C_1 where the state of process p is different
4. We apply the same sequence of events S to both configurations
   - Where p does not participate in S
5. In both cases we will end up with the same decision value
6. Thus one of C_0, C_1 is bi-valent. Impossible.

■

Consensus with One Faulty Process: Impossibility Result

- This lemma establishes that a bi-valent initial configuration exists.
- The next lemma shows that from each bi-valent configuration we may lead the system to another bi-valent configuration.
  - What will happen if we delay indefinitely the event that will lead the system to a uni-valent configuration ?

Lemma

Let C a bi-valent configuration of the algorithm and $e = (p, m)$ an event applicable to C. Let X the set of configurations that are reachable by C without applying e, and $Y = e(X) = \{e(E)|E \in X\}$. Then Y may include at least one bi-valent configuration.

- The decision is delayed indefinitely.

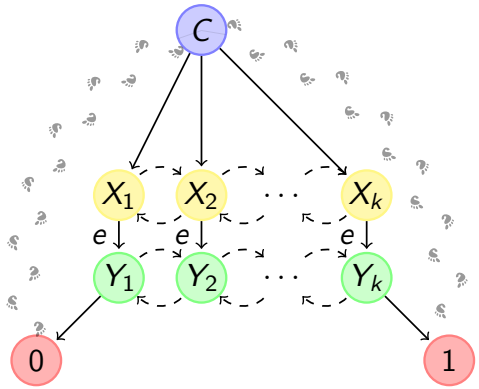
Consensus with One Faulty Process: Impossibility Result

Proof: Let assume that Y does not include bi-valent configurations.
Then it includes both 0-valent and 1-valent configurations

- Since C is bi-valent,
- It leads to both 0-valent and to 1-valent configurations.
- Even after the delivery of e each i-valent remains i-valent.

■

Consensus with One Faulty Process: Impossibility Result



Consensus with One Faulty Process: Impossibility Result

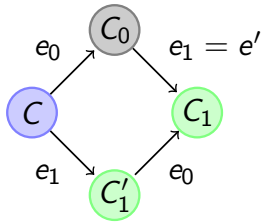
Proof: There exist C_0, C_1 in X such that $D_i = e(C_i)$ is i-valent. If we observe carefully, there exist C_0, C_1 such that $C_i = e'(C_{-i})$ where $e' = (p', m')$ an event

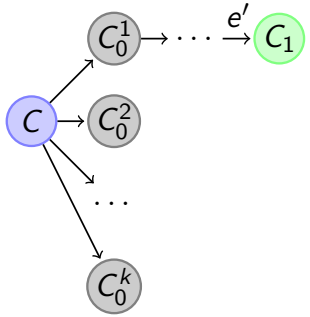
- One results from the other by the delivery of the message
- Without loss of generality $C_1 = e'(C_0)$

■

Let's examine the events that occur in C

Consensus with One Faulty Process: Impossibility Result

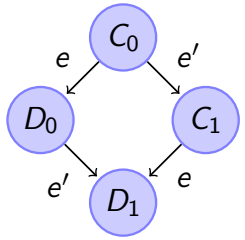




Consensus with One Faulty Process: Impossibility Result I

Proof: Let's examine p'

- $p' \neq p$: Then $D_1 = e'(D_0)$ due to the commutative of independent events. Impossible since from a 0-valent configuration we cannot reach a 1-valent configuration.

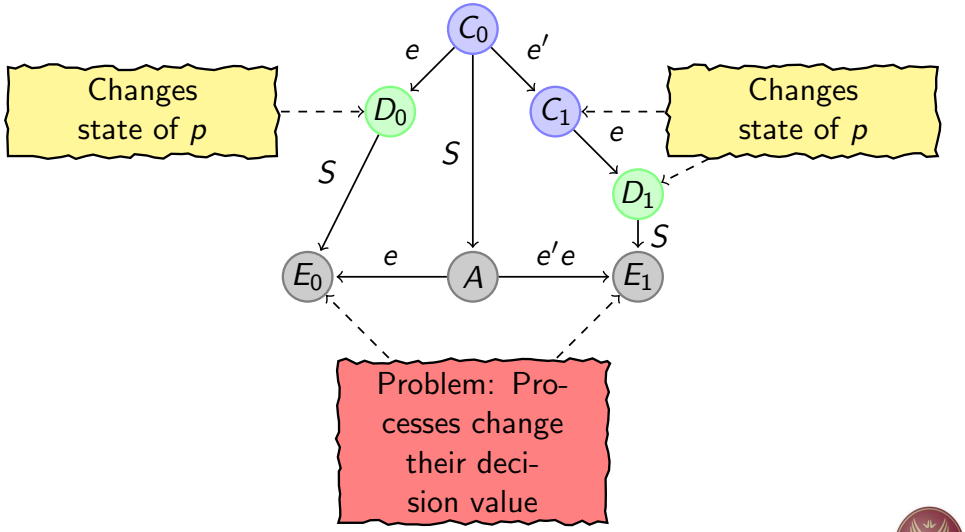


Consensus with One Faulty Process: Impossibility Result

Proof: (continued)

- $p' = p$: If the sequence of events S during which p does not participate leads to a decision A, S may be applied to D_0 and to D_1
- S exists since in the case when p fails the rest of the network will decide
- When S is applied to D_i it leads to a i-valent configuration E_i (since D_i is i-valent)
- However due to the commutative law:
  - $e(A) = E_0$
  - $e'(e(A)) = E_1$
- Thus the resulting configuration is bi-valent. Impossible since the algorithm has decided!

Consensus with One Faulty Process: Impossibility Result



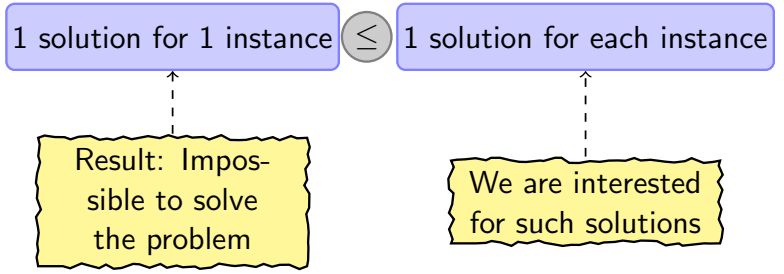
Consensus with One Faulty Process: Impossibility Result

Based on the previous observations we may construct an execution that does not terminate

- There is a bi-valent initial configuration
- Each event that my lead to a uni-valent configuration is delayed long enough to lose this property
- Since we interchange two bi-valent configuration, the decision is never reached
- The algorithm does not terminate

Importance of the impossibility result

- Why bother?
  - We prove the non-existence of an applicable solution in any network
  - We just proved impossibility in a network of 2 processes
- The above result is very strong!
- It shows that no solution exists event if we look into a specific network
  - Even if just one process fails
  - It is enough not to know a-priori which process fails



Open Problem # 9

Assume a synchronous distributed system consisting of a set of n processes that are connected by a directed ring network, where each process has a unique identity but is not aware of the total number of processes neither of the network's topology. Design a distributed leader election algorithm that can tolerate 1 byzantine failure. Define algorithm's properties and verify it's correctness, as well as the time and message complexity. You should prove your claims.