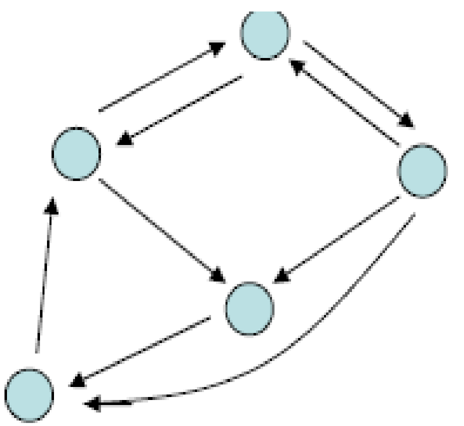
Synchronous network model

- A Synchronous Network System consists of a collection of computing elements (“processors” of “Processes”) located at the nodes of a directed network graph.
- The computing elements are connected via a connected network (i.e., there exists one channel between any pair of processes).

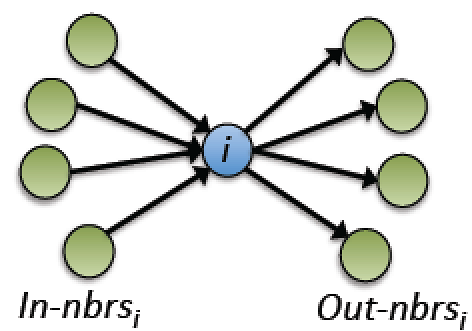


Synchronous network model

- Formally,

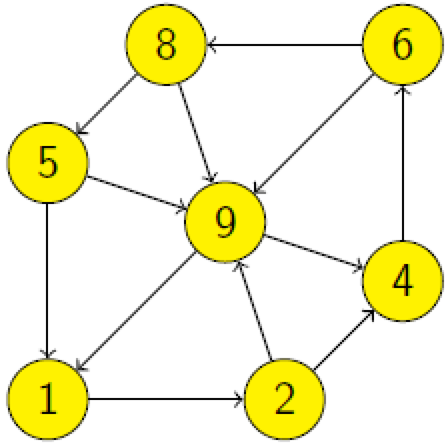
  Directed graph $G=(V,E)$

  - $n = |V|$, number of nodes in the graph
  - For $i \in V$
    - *Out-nbrs*$_i$ set of nodes j s.t. $(i,j) \in E$
    - *In-nbrs*$_i$ set of nodes j s.t. $(j,i) \in E$



Neighboring Processes

- We say vertex v is outgoing neighbor of vertex u if
  - the edge uv is included in G.
- We say vertex u is incoming neighbor of vertex v if
  - the edge uv is included in G.
- We define $nbrs_u^{out} = \{v|(u,v) \in E\}$ all the vertices that are *outgoing neighbors* of vertex u.
- We define $nbrs_u^{in} = \{v|(v,u) \in E\}$ all the vertices that are *incoming neighbors* of vertex u.



5 is outgoing neighbor of 8

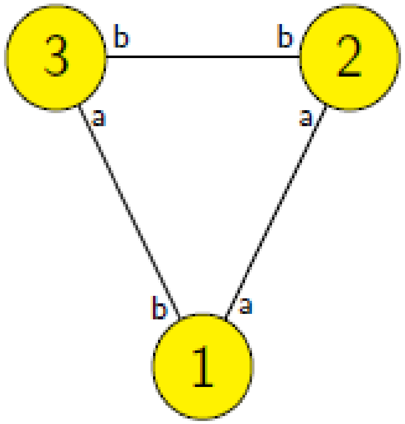
8 is incoming neighbor of 5

$nbrs_9^{out} = \{1, 4\}$

$nbrs_9^{in} = \{2, 5, 6, 8\}$

Modeling Communication Channels

- Channels are the edges of the graph.
  - The edges may be directed – to represent unidirectional communication.
  - or undirected – to represent bidirectional communication.
- Processes can distinguish each communication channel and select a specific one to use.



Modeling Messages

- Data exchange over communication channels is done via message exchanges.
- We assume that each communication channel may transmit only one message at any time instance.
- We assume that there exists a fixed message alphabet M
  - remains fixed throughout the execution of the system.
  - contains the symbol `null` a placeholder indicating the absence a message.

Network Properties

`distance`(u,v)

Let `distance`(u,v) denote the length of the shortest directed path from u to j in G, if any exists; otherwise `distance`$(u,v)=\infty$.

`diam`(G)

Let `diam`(G) denote the diameter of the graph G, the maximum distance `distance`(u,v), taken over all paths (u,v).

Network Topology & Initial Knowledge

- Distributed algorithms may be designed for a specific network topology
  - ring, tree, fully connected graph ...
- Distributed algorithm may be designed for networks with specific properties
  - we say that the algorithm has "initial knowledge"
- An algorithm assuming a large number of specific properties is called "weak" algorithm.
  - An algorithm that does not assume any specific property is called "strong" algorithm – since it can be executed in a broader range of possible networks.

Process States

- 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 $msgs_u : states_u \times nbrs_u^{out} \to M \cup \{\texttt{null}\}$
  - given a current state,
  - generates messages for each neighboring process.
- Uses a state-transition function $trans_u : states_u \times (M \cup \{\texttt{null}\})^{nbrs_u^{in}} \to states_u$
  - given a current state,
  - and messages received,
  - computes the next state of the process.

System Initialization

- Initially
  - all processes are set to an initial state,
  - all channels are empty.
- Algorithms groups processes in two sets
  1. Initiators – a process is initiator if it activates the execution of the algorithm in the local neighborhood.
  2. Non-initiators – a non-initiating process is activated when a message is received from a neighboring process.

Centralized vs Decentralized

An algorithm is classified as centralized if the exists one and only one initiator in each execution and decentralized if the algorithm may be initialized with an arbitrary subset of processes.

- Usually centralized algorithms achieve low message complexity.
- Usually decentralized algorithms achieve improved performance in the presence of failures.

Uniformity

An algorithm is uniform if its description is independent of the network size n.

- A property that holds for a small network size, also holds for large network sizes.
- We only have to examine the behavior of a protocol (for a given property) in small network sizes.

Algorithm execution: Steps and Rounds

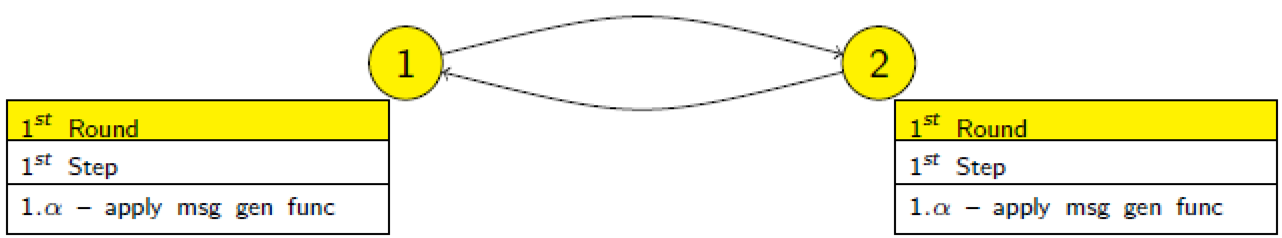
- All processes, repeat in a "synchronized" manner the following steps:
  1^{st} Step
  1. Apply the message generator function.
  2. Generate messages for each outgoing neighbor.
  3. Transmit messages over the corresponding channels.

  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.

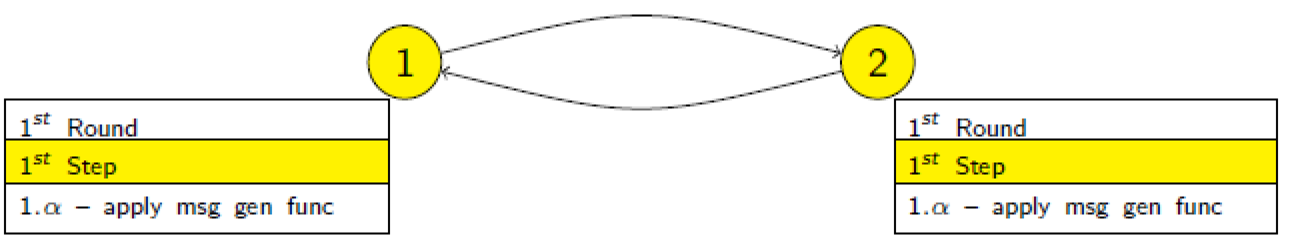
Execution of Synchronous System



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.

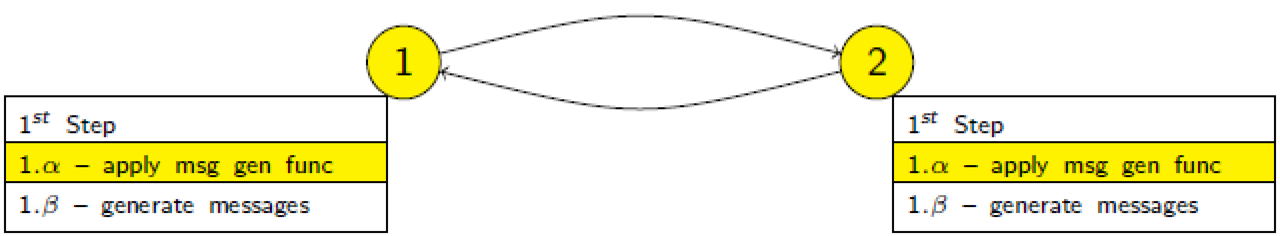
Execution of Synchronous System



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.

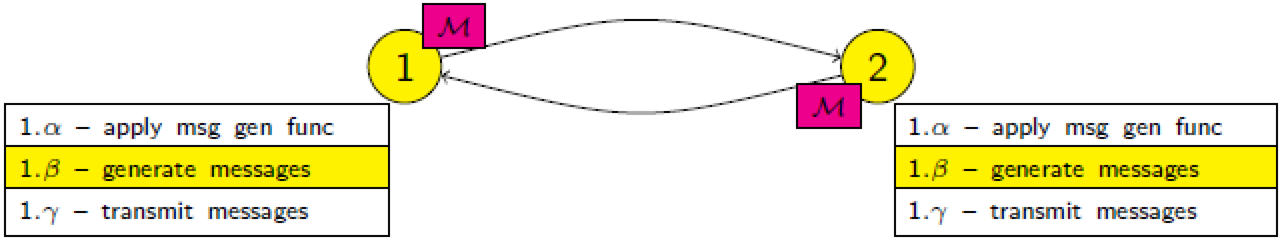
Execution of Synchronous System



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.

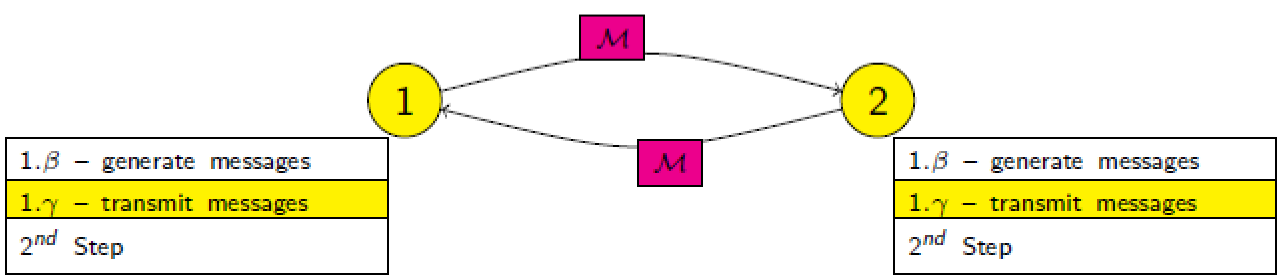
Execution of Synchronous System



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.

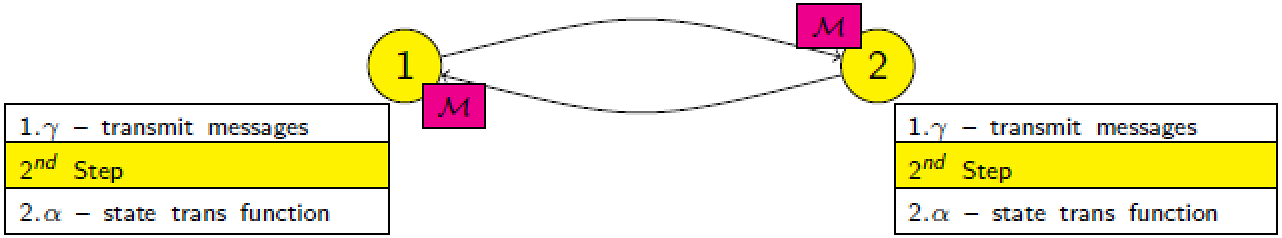
Execution of Synchronous System



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.

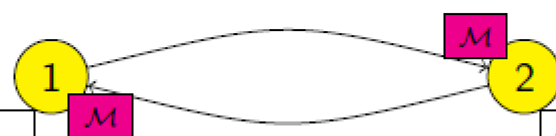
Execution of Synchronous System



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



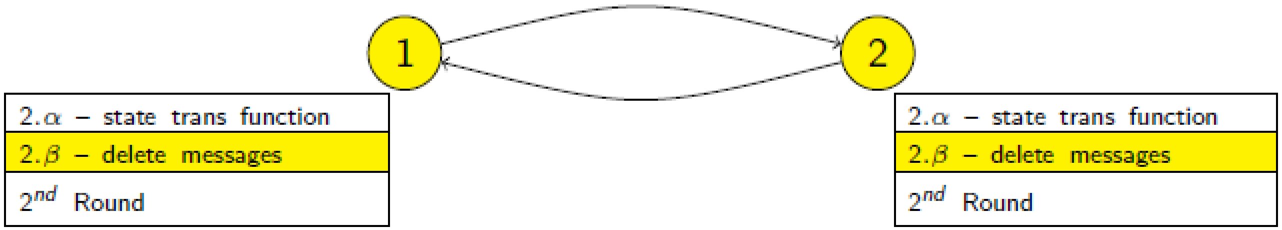
2^{nd} Step
$2.\alpha$ – state trans function
$2.\beta$ – delete messages

2^{nd} Step
$2.\alpha$ – state trans function
$2.\beta$ – delete messages

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.

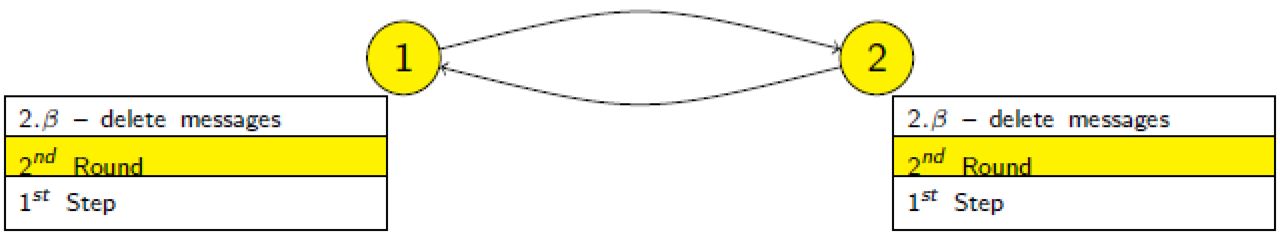
Execution of Synchronous System



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.

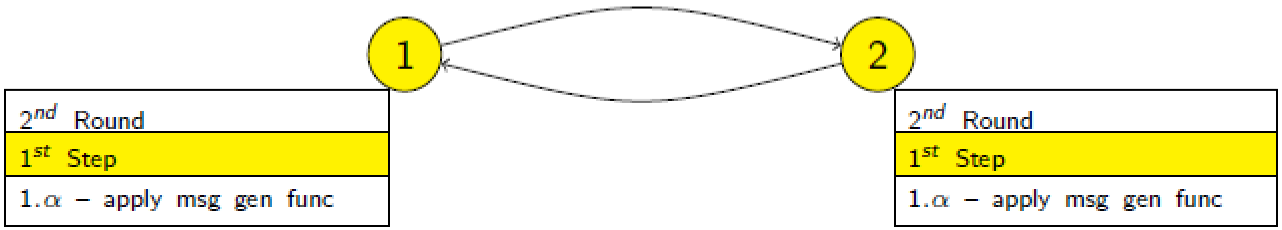
Execution of Synchronous System



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.

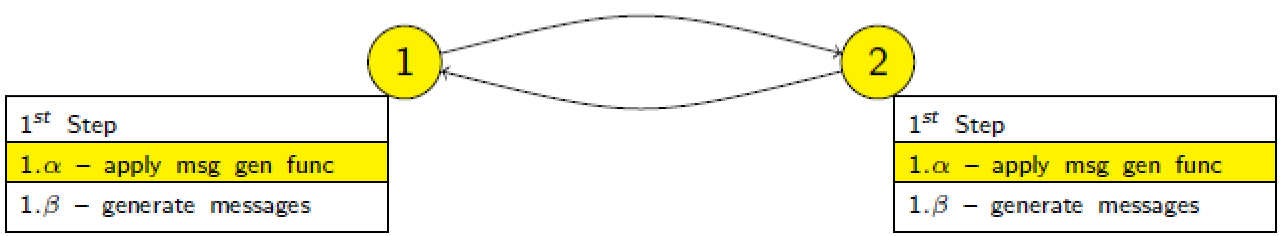
Execution of Synchronous System



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.

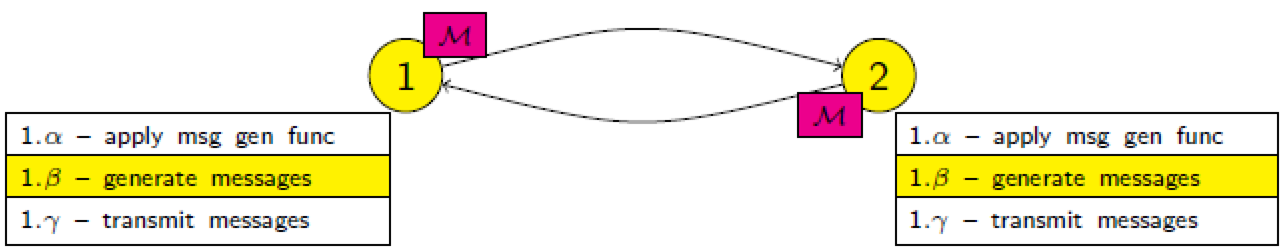
Execution of Synchronous System



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.

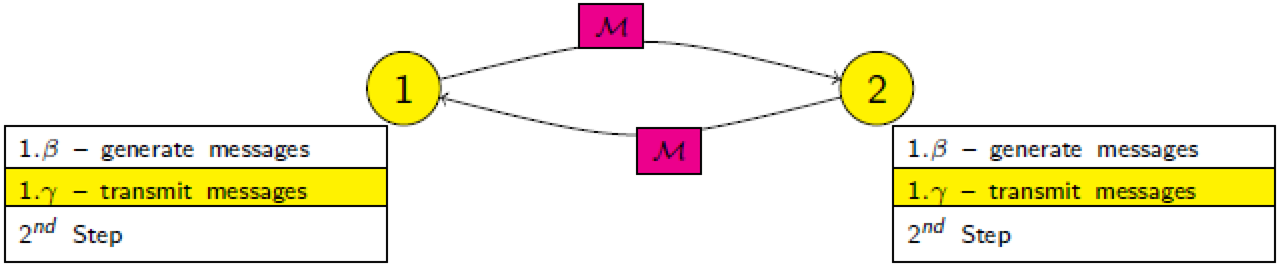
Execution of Synchronous System



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.

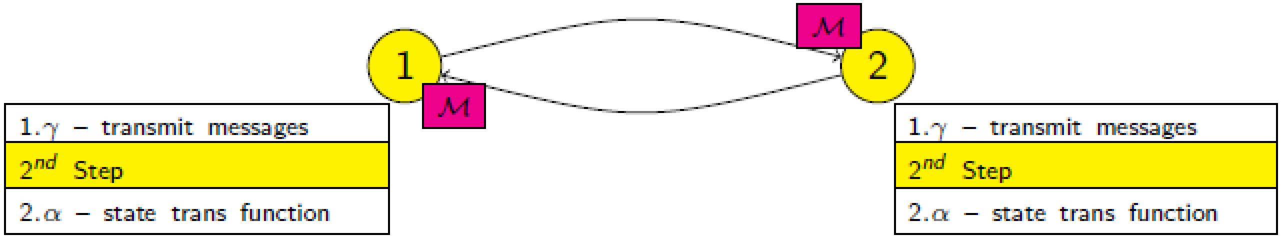
Execution of Synchronous System



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.

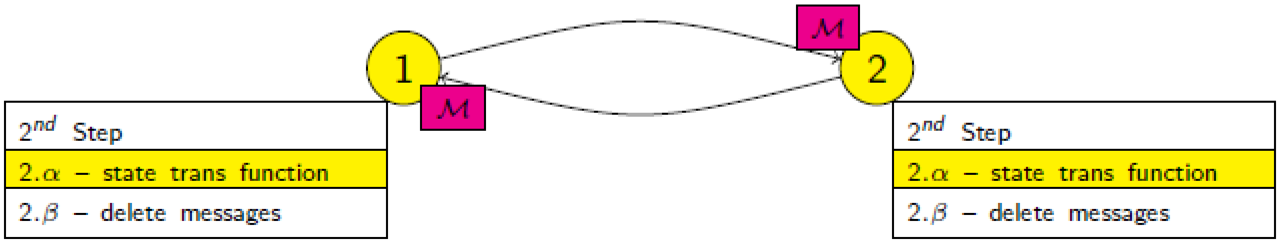
Execution of Synchronous System



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.

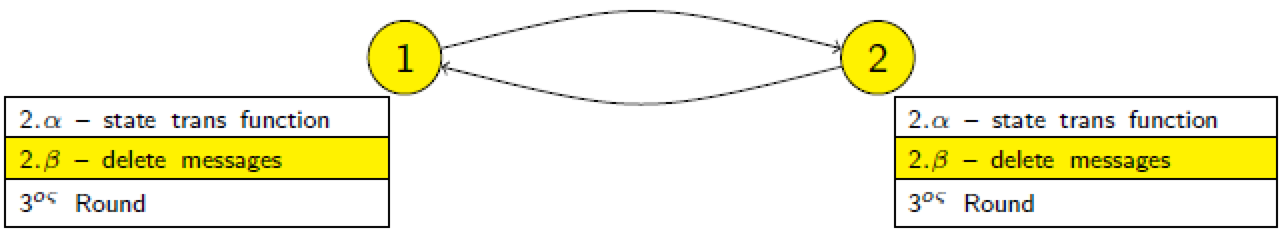
Execution of Synchronous System



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.

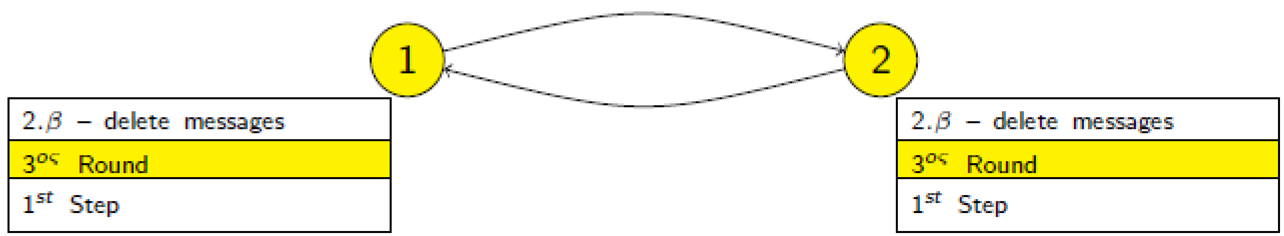
Execution of Synchronous System



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



System Configuration

We wish to describe the execution of a distributed algorithm.

- We assume a sequence of state transitions of the processes of the system
  - produced as result of transmissions and receptions of messages, or
  - internal (to each process) reasons.
- Lets assume a given time instance i
  - each process u is in state $states_u$.
  - the characterization of the state of all processes defines a configuration of the system C_i.

Execution of a distributed algorithm

- Initially, processes execute a single round of the algorithm
  - a given set of message transmissions M_i take place,
  - a given set of message N_i are received.
- The next round $i+1$, we say that the system is in configuration C_{i+1}
- The execution of the distributed algorithm can be defined as an infinite sequence
  $C_0, M_1, N_1, C_1, M_2, N_2, C_2, \ldots$