



Module-3

Time and Global States

Department of
Computer Science & Engineering

notesforvtu.in

www.cambridge.edu.in



Clocks

- The instantaneous difference between the readings of any two clocks is called their **skew**.
- **Clock drift**, which means that they count time at different rates, and so diverge.

Coordinated Universal Time

- Computer clocks can be synchronized to external sources of highly accurate time.
- Coordinated Universal Time – abbreviated as UTC (from the French equivalent) – is an international standard for timekeeping.
- UTC signals are synchronized and broadcast regularly from land based radio stations and satellites covering many parts of the world.

Synchronizing Physical clocks

In order to know at what time of day events occur at the processes in our distributed system – for example, for accountancy purposes – it is necessary to synchronize the processes' clocks.

External synchronization: means that all computers in the system are synchronized with an external source of time (e.g., a UTC signal). Clocks C_i are synchronized with one another to a known degree of accuracy.

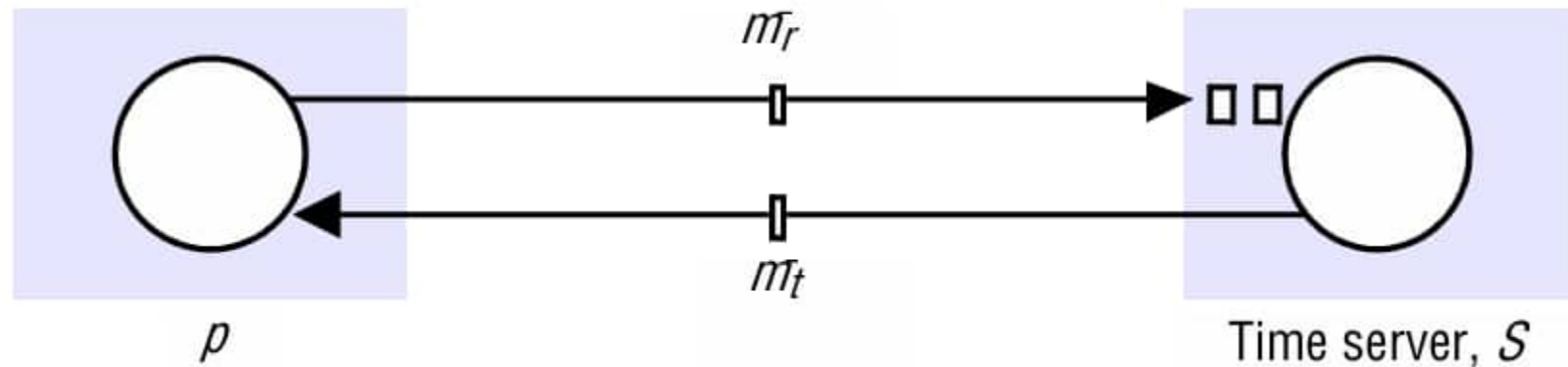
Internal synchronization: means that all computers in the system are synchronized with one another, but the time is not necessarily accurate with respect to UTC.

Synchronization in a synchronous system

- Internal synchronization between two processes in a synchronous distributed system.
- One process sends the time t on its local clock to the other in a message m .
- In principle, the receiving process could set its clock to the time $t + T_{\text{trans}}$, where T_{trans} is the time taken to transmit m between them. The two clocks would then agree (since the aim is internal synchronization, it does not matter whether the sending process's clock is accurate).

Cristian's method for synchronizing clocks

Clock synchronization using a time server



Cristian's method for synchronizing clocks

Algorithm:

- 1) The process on the client machine sends the request for fetching clock time(time at the server) to the Clock Server at time T_0 .
- 2) The Clock Server listens to the request made by the client process and returns the response in form of clock server time.
- 3) The client process fetches the response from the Clock Server at time T_1 and calculates the synchronized client clock time using the formula given below.

$$T_{\text{client}} = T_{\text{server}} + (T_1 - T_0) / 2$$

Cristian's method for synchronizing clocks

- Cristian's method suffers from the problem associated with all services implemented by a single server:
that the single time server might fail and thus render synchronization temporarily impossible.
- Cristian suggested, for this reason, that time should be provided by a group of synchronized time servers, each with a receiver for UTC time signals.
- For example, a client could multicast its request to all servers and use only the first reply obtained.
- Note that a faulty time server that replied with spurious time values, or an imposter time server that replied with deliberately incorrect times, could wreak havoc in a computer system.

The Berkeley algorithm

- Describe an algorithm for internal synchronization that they developed for collections of computers running Berkeley UNIX.
- In it, a coordinator computer is chosen to act as the **master**. Unlike in Cristian's protocol, this computer periodically polls the other computers whose clocks are to be synchronized, called **slaves**.
- The slaves send back their clock values to it. The master estimates their local clock times by observing the round-trip times (similarly to Cristian's technique), and it averages the values obtained.

The Network Time Protocol

- Cristian's method and the Berkeley algorithm are intended primarily for use within intranets.
- The Network Time Protocol (NTP) [Mills 1995] defines an architecture for a time service and a protocol to distribute time information over the Internet.

The Network Time Protocol

NTP's chief design aims and features are as follows:

- To provide a service enabling clients across the Internet to be synchronized accurately to UTC.
- To provide a reliable service that can survive lengthy losses of connectivity.
- To enable clients to resynchronize sufficiently frequently to offset the rates of drift found in most computers.
- To provide protection against interference with the time service, whether malicious or accidental.

The Network Time Protocol

The NTP service is provided by a network of servers located across the Internet.

Primary servers are connected directly to a time source such as a radio clock receiving UTC.

Secondary servers are synchronized with primary servers.

The servers are connected in a logical hierarchy called a **synchronization subnet** whose levels are called **strata**.

Logical Time and Logical clocks

We can use a scheme that is similar to physical causality but that applies in distributed systems to order some of the events that occur at different processes.

This ordering is based on two simple and intuitively obvious points:

- If two events occurred at the same process p_i ($i = 1, 2, \dots, N$), then they occurred in the order in which p_i observes them.
- Whenever a message is sent between processes, the event of sending the message occurred before the event of receiving the message.

Logical Time and Logical clocks

Lamport called the partial ordering obtained by generalizing these two relationships the **happened-before relation**. It is also sometimes known as the relation of causal ordering or potential causal ordering.

Logical Time and Logical clocks

The **happened-before** relation is a partial ordering of events in distributed systems such that: if events a and b are two events happening at the same node, the relation $a \rightarrow b$ holds if the occurrence of event a preceded the occurrence of event b . This is easy to determine for a single node.

Logical Time and Logical clocks

We can define the happened-before relation, denoted by $\rightarrow$, as follows:

HB1: If every process $P_i : e \rightarrow_i e'$, then $e \rightarrow e'$.

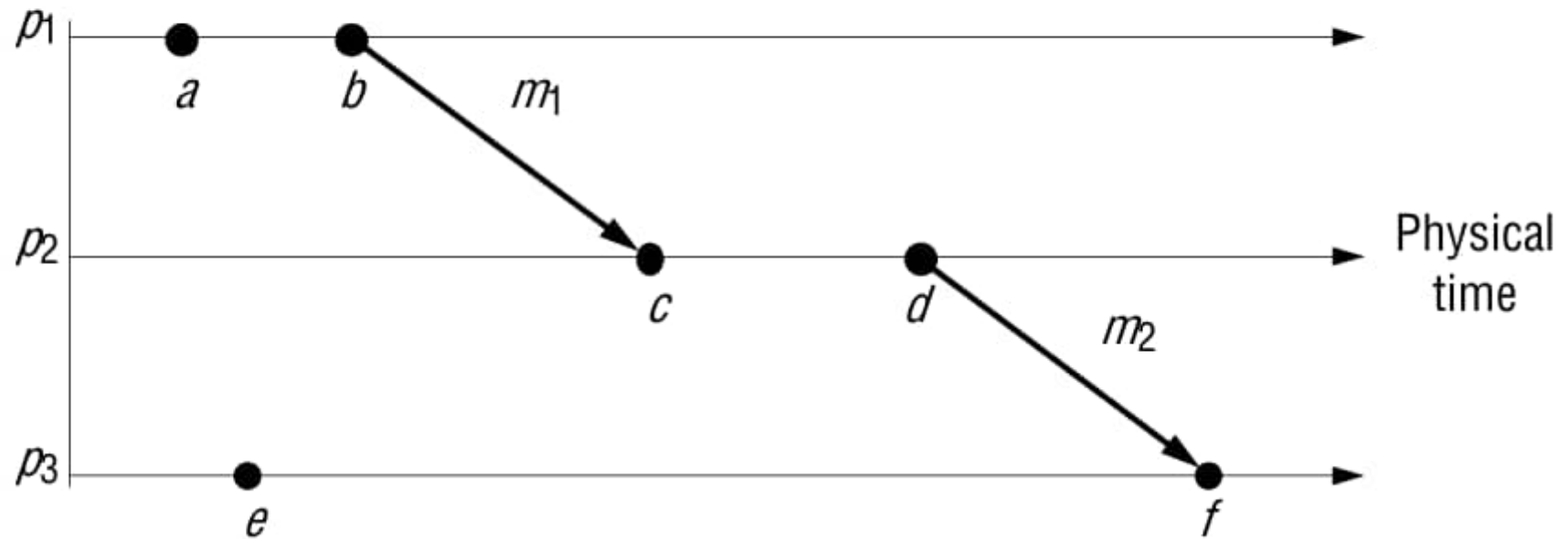
HB2: For any message m , $\text{send}(m) \rightarrow \text{receive}(m)$

– where $\text{send}(m)$ is the event of sending the message, and
 $\text{receive}(m)$ is the event of receiving it.

HB3: If e , e' and e'' are events such that $e \rightarrow e'$ and $e' \rightarrow e''$, then $e \rightarrow e''$.

Logical Time and Logical clocks

Events occurring at three processes



Logical clocks

- Lamport [1978] invented a simple mechanism by which the happened before ordering can be captured numerically, called a logical clock.
- A Lamport logical clock is a monotonically increasing software counter, whose value need bear no particular relationship to any physical clock.
- Each process p_i keeps its own logical clock, L_i , which it uses to apply so-called Lamport timestamps to events. We denote the timestamp of event e at p_i by $L_i(e)$, and by $L(e)$ at whatever process it occurred at.

Logical clocks

- To capture the happened-before relation $\rightarrow$ processes update their logical clocks and transmit the values of their logical clocks in messages as follows:
 - LC1: L_i is incremented before each event is issued at process p_i :
$$L_i := L_i + 1.$$
 - LC2: (a) When a process p_i sends a message m , it piggybacks on m the value $t = L_i$.
(b) On receiving (m, t) , a process p_j ,computes $L_j := \max(L_j ,t)$ and then applies LC1 before timestamping the event $\text{receive}(m)$.

Totally ordered logical clocks

- If e is an event occurring at p_i with local timestamp T_i , and e is an event occurring at p_j with local timestamp T_j , we define the global logical timestamps for these events to be (T_i, i) and (T_j, j) , respectively.
- And we define $(T_i, i) < (T_j, j)$ if and only if either $T_i < T_j$, or $T_i = T_j$ and $i < j$. This ordering has no general physical significance (because process identifiers are arbitrary), but it is sometimes useful. Lamport used it, for example, to order the entry of processes to a critical section.

Vector clocks

- Mattern [1989] and Fidge [1991] developed vector clocks to overcome the shortcoming of Lamport's clocks: the fact that from $L(e) < L(e')$ we cannot conclude that $e \rightarrow e'$.
- A vector clock for a system of N processes is an array of N integers.
- Each process keeps its own vector clock, V_i , which it uses to timestamp local events.

Vector clocks

- Like Lamport timestamps, processes piggyback vector timestamps on the messages they send to one another, and there are simple rules for updating the clocks:

VC1: Initially, $V_i[j] = 0$, for $i, j = 1, 2, \dots, N$.

VC2: Just before p_i timestamps an event, it sets $V_i[i] := V_i[i] + 1$.

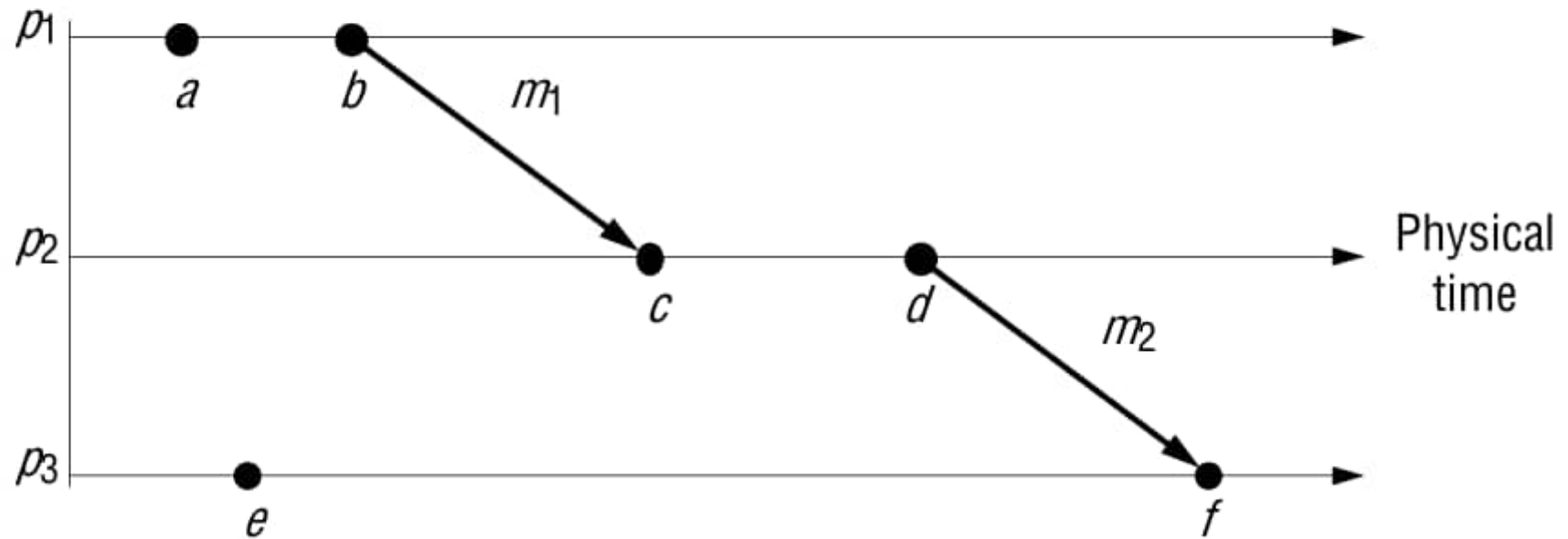
VC3: p_i includes the value $t = V_i$ in every message it sends.

VC4: When p_i receives a timestamp t in a message, it sets

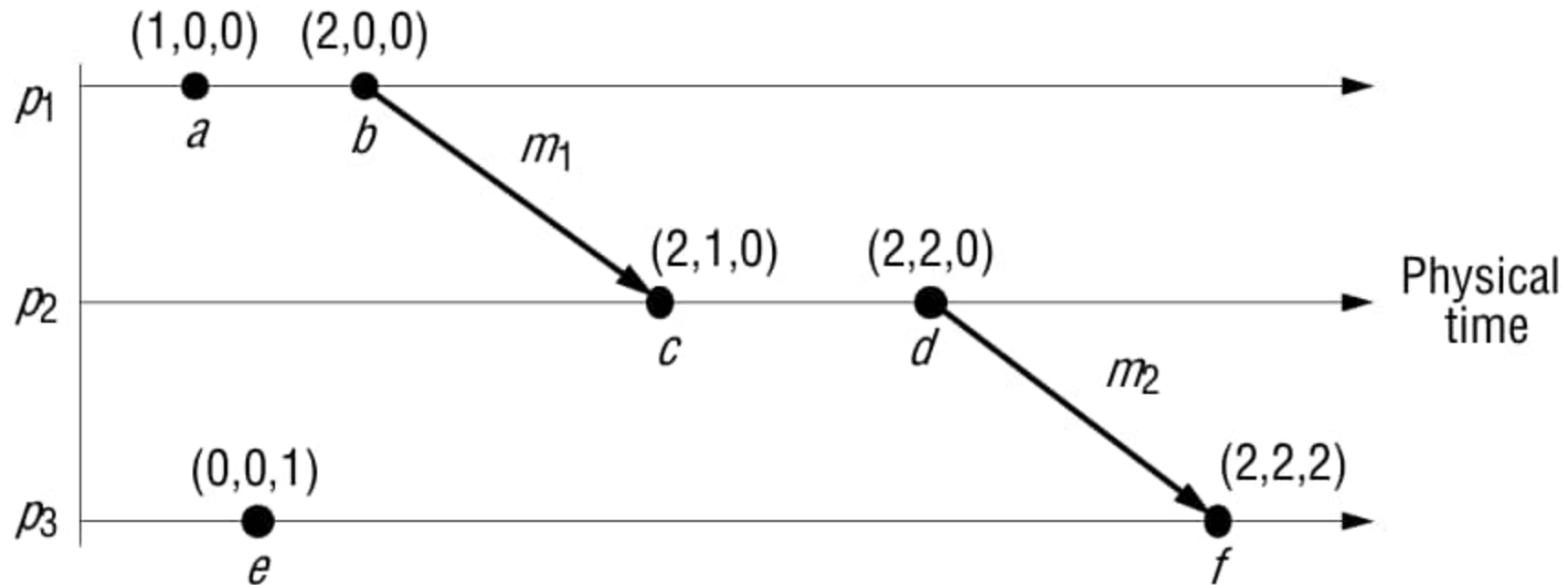
$V_i[j] := \max(V_i[j], t[j])$, for $j = 1, 2, \dots, N$. Taking the component wise maximum of two vector timestamps in this way is known as a merge operation.

Logical Time and Logical clocks

Events occurring at three processes



Vector clocks

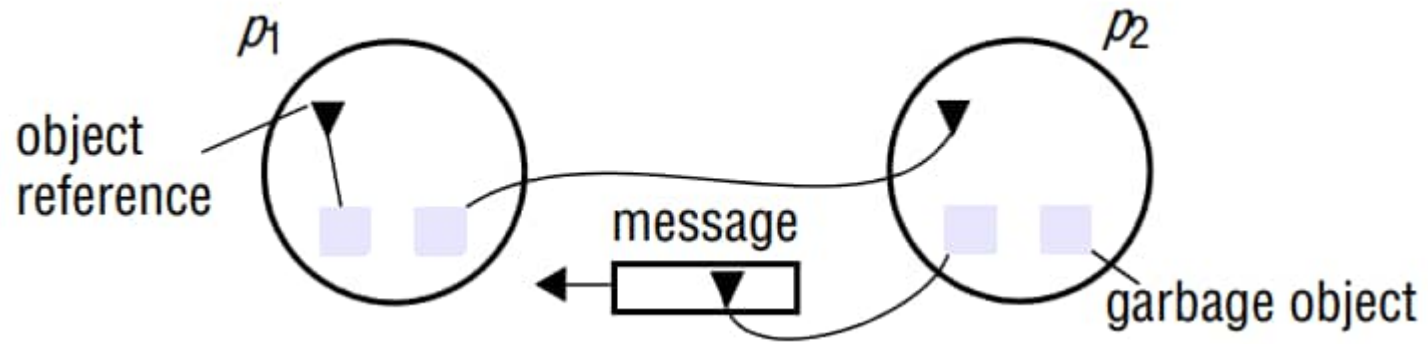


Global States

Some of the global properties are:

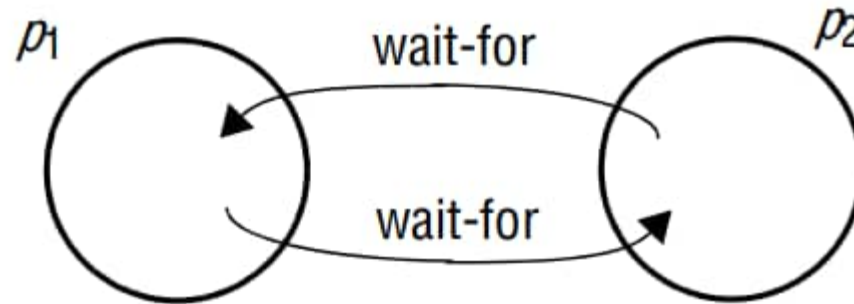
- distributed garbage collection
- deadlock detection
- termination detection and
- debugging

Distributed garbage collection



- An object is considered to be garbage if there are no longer any references to it anywhere in the distributed system

Distributed deadlock detection



- A distributed deadlock occurs when each of a collection of processes waits for another process to send it a message, and where there is a cycle in the graph of this 'waits-for' relationship.

The 'snapshot' algorithm of Chandy and Lamport

The algorithm assumes that:

- Neither channels nor processes fail – communication is reliable so that every message sent is eventually received intact, exactly once.
- Channels are unidirectional and provide FIFO-ordered message delivery.
- The graph of processes and channels is strongly connected (there is a path between any two processes).
- Any process may initiate a global snapshot at any time.
- The processes may continue their execution and send and receive normal messages while the snapshot takes place.

Chandy and Lamport's 'snapshot' algorithm

Marker receiving rule for process p_i

On receipt of a *marker* message at p_i over channel c :

if (p_i has not yet recorded its state) it

records its process state now;

records the state of c as the empty set;

turns on recording of messages arriving over other incoming channels;

else

p_i records the state of c as the set of messages it has received over c
since it saved its state.

end if

Marker sending rule for process p_i

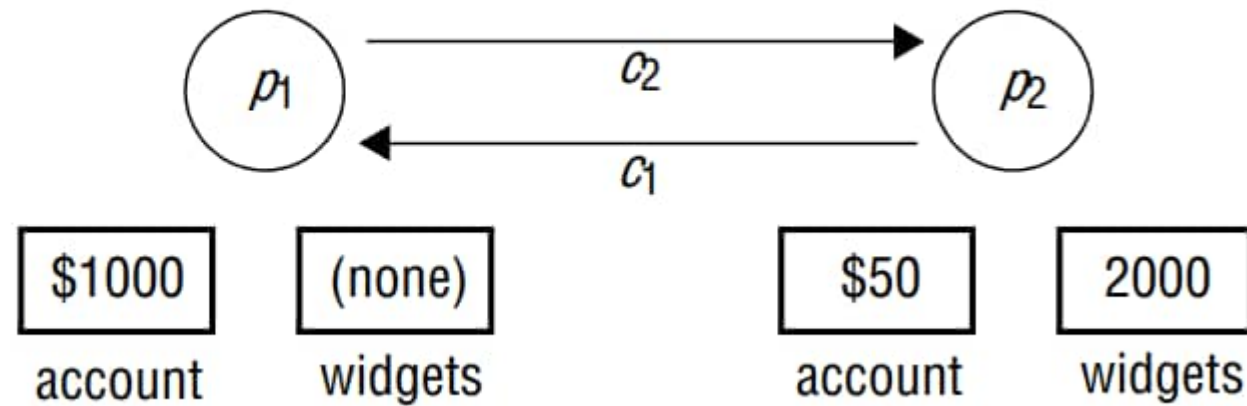
After p_i has recorded its state, for each outgoing channel c :

p_i sends one marker message over c

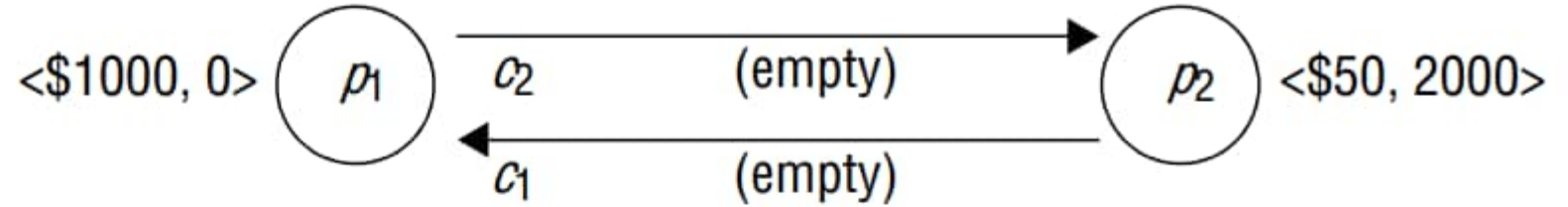
(before it sends any other message over c).

Example

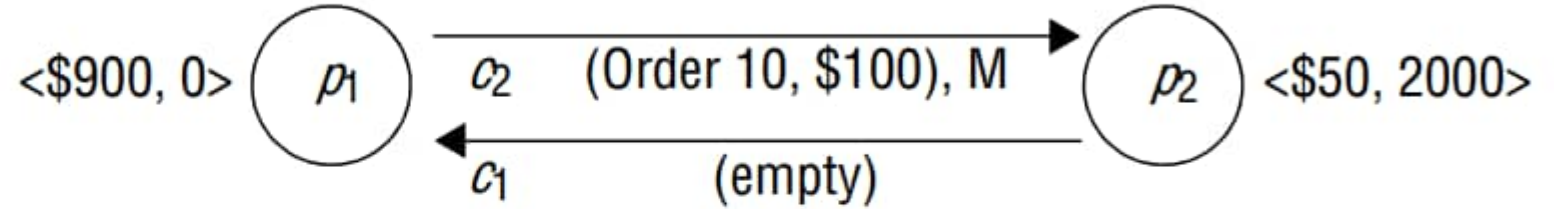
Two processes and their initial states



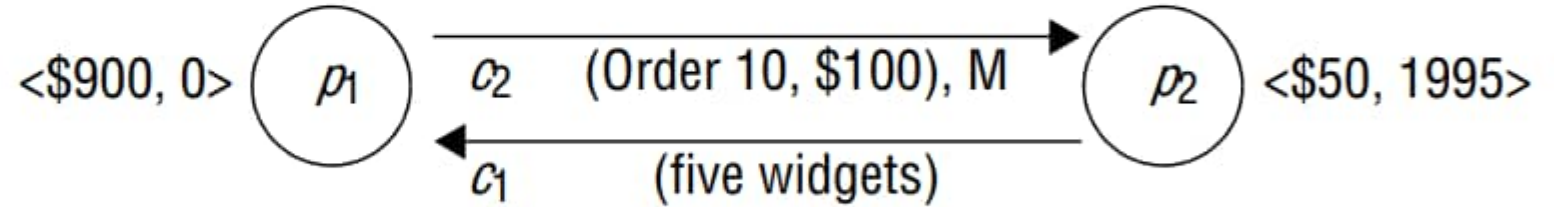
1. Global state S_0



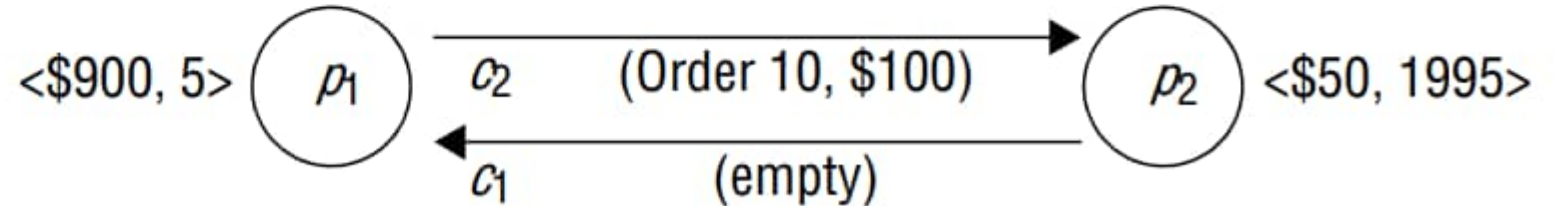
2. Global state S_1



3. Global state S_2



4. Global state S_3



(M = marker message)