Execution of a Sequential Process

1. A process which has a local clock executes by processing events. The order in which the events are processed (occur) is determined by the execution of the program. The sequential order is usually represented as the values read from the local clock but any monotone sequence of numbers could be used as well.

2. While we can say that an event occurred at a specific time, the correctness of the execution does not depend upon the time of occurrence of the event, only that each event occurs in a specified order in the sequence of events.

Execution of Parallel/Distributed Processes

1. Execution of a program is partitioned among a set of logical processes each of which has its own local clock and executes a stream of events. Execution of local events assigns a sequential order to the occurrence of the local events. 2. The set of processes must collaborate to attain some goal.

3. Therefore the local sequences of events generated by the execution of each process must be interleaved to create a global sequence of events.

4. The processes interact only by sending messages.

5. Each message has is time-stamped with the local time of the sending process.

6. A Lamport clock can be used to generate a global sequence of event occurrence which is consistent with some sequence which would have been generated by a sequential execution of the same program.

Single Clock Execution of Discrete Event Simulations

1. Events are generated and <u>assigned</u> a time to be executed.

2. The events must be executed in the order determined by their assigned time.

3. Events are placed on a queue or list ordered by their time for execution.

4. The simulation engine fetches the events from the queue and invokes the evaluator for each event.

5. Evaluation of an event typically generates other events each of which has a time to be executed.

6. This process requires that all of the events be assigned times from a single clock or a single monotone sequence.

Parallel Execution of Discrete Event Simulation

1. **Generation and evaluation of events is partitioned among several logical processes which are implemented on some number of physical processors.**

2. **A means of generating consistent times and evaluating the events in a sequence which will conform to some sequential execution must be developed.**

3. **We will now take a look at the several methods which have been proposed for parallel/distributed execution of a single system simulation.**

4. **There are many approaches for parallel execution of simulations. We will sketch a few of these before we examine parallel/distributed simulation.**

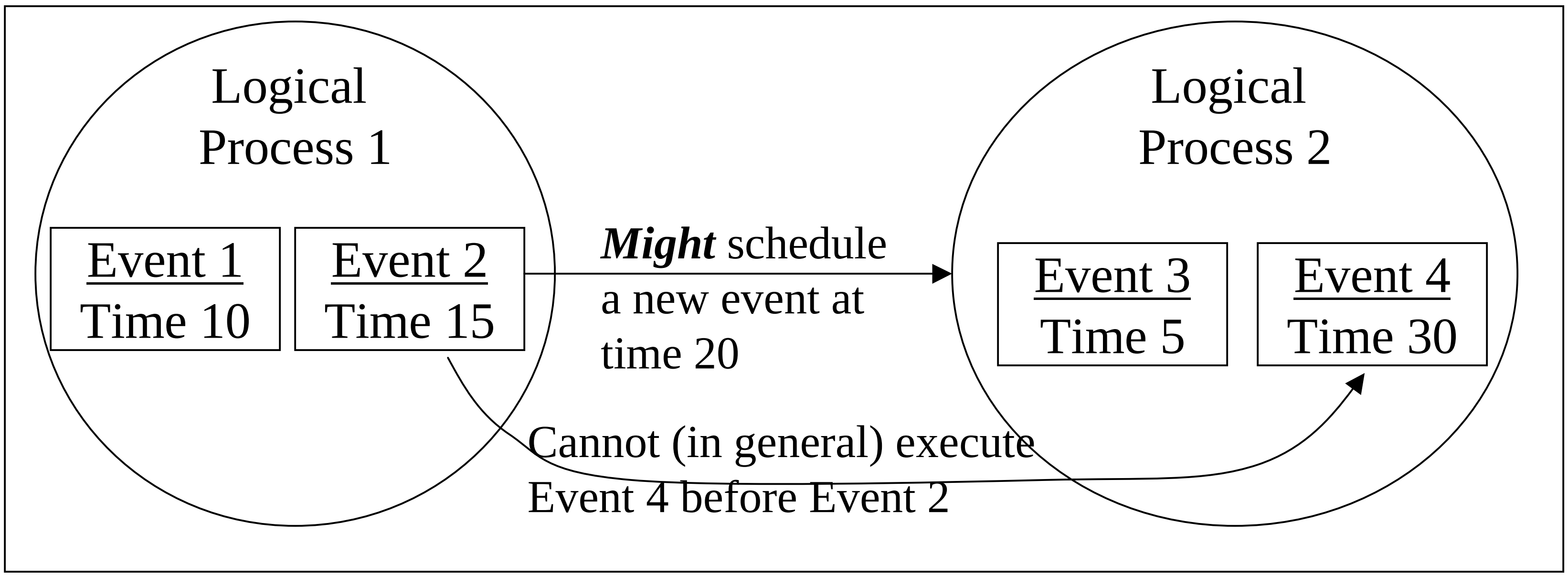
Why is Parallel Simulation Hard?

- **Causality: Sequencing constraints must be maintained in order for the computation to be correct (free of causality errors).**
- **For example, if events E1 and E2 occur at times T1 and T2, where T1 < T2, and E1 writes into a state variable that is read by E2, then E1 must be executed before E2.**

- **Unpredictability: Because simulations involve random events, we cannot predict the precise sequence or pattern of events that will occur in either a sequential or parallel simulation.**
  - **In general, each process in a parallel simulation cannot advance beyond the earliest time that it MIGHT receive a message from another process.**

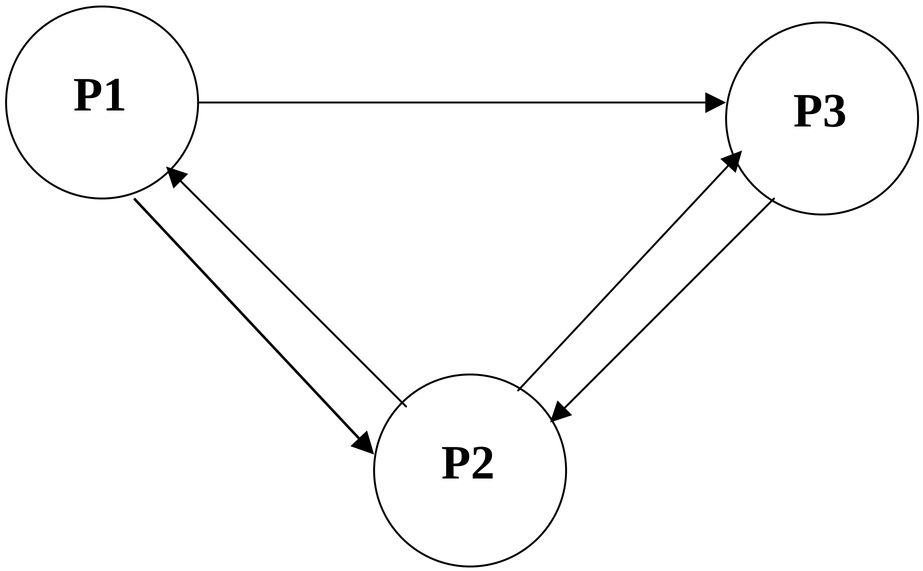
Parallel/Distributed Discrete Event Simulation

How we we sequence across multiple sites?



Conservative (Chandy-Misra-Bryant)Parallel/Distributed Simulation

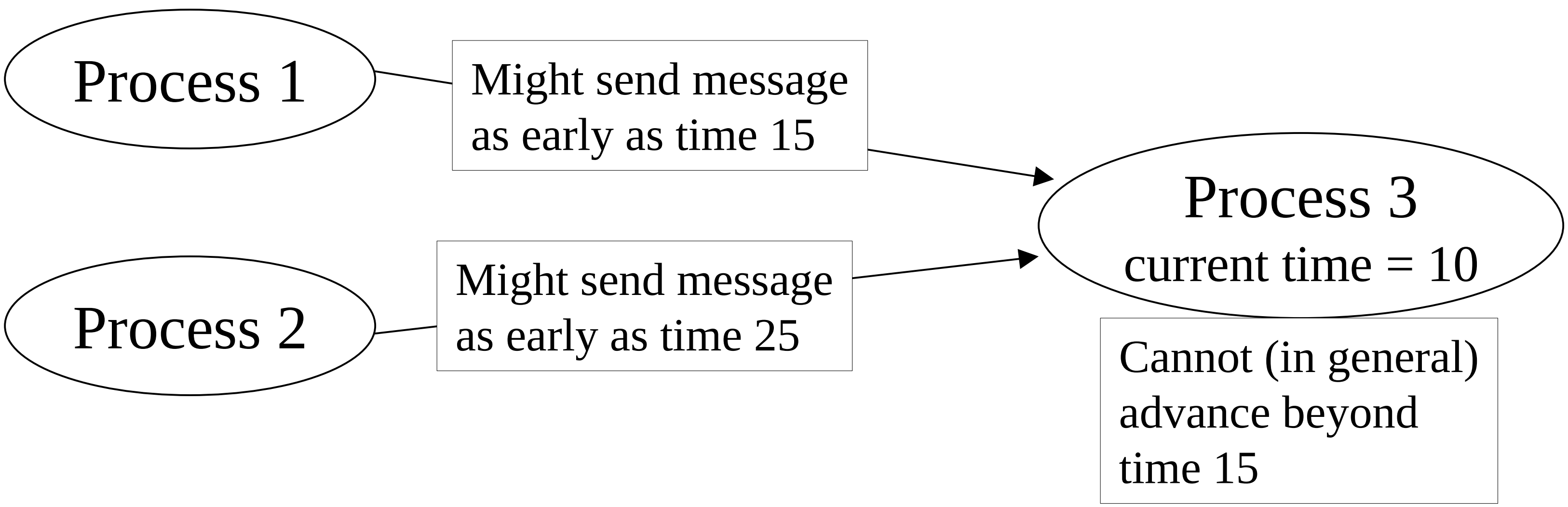
1. Let us partition the system into subsystems and agree to process the events in each subsystem by a separate process. The processes are connected by a set of one way channels. The processes interact by sending messages on the channels.



2. Each process may generate events for local processing, send events to other processes for evaluation and receive events from other processes.

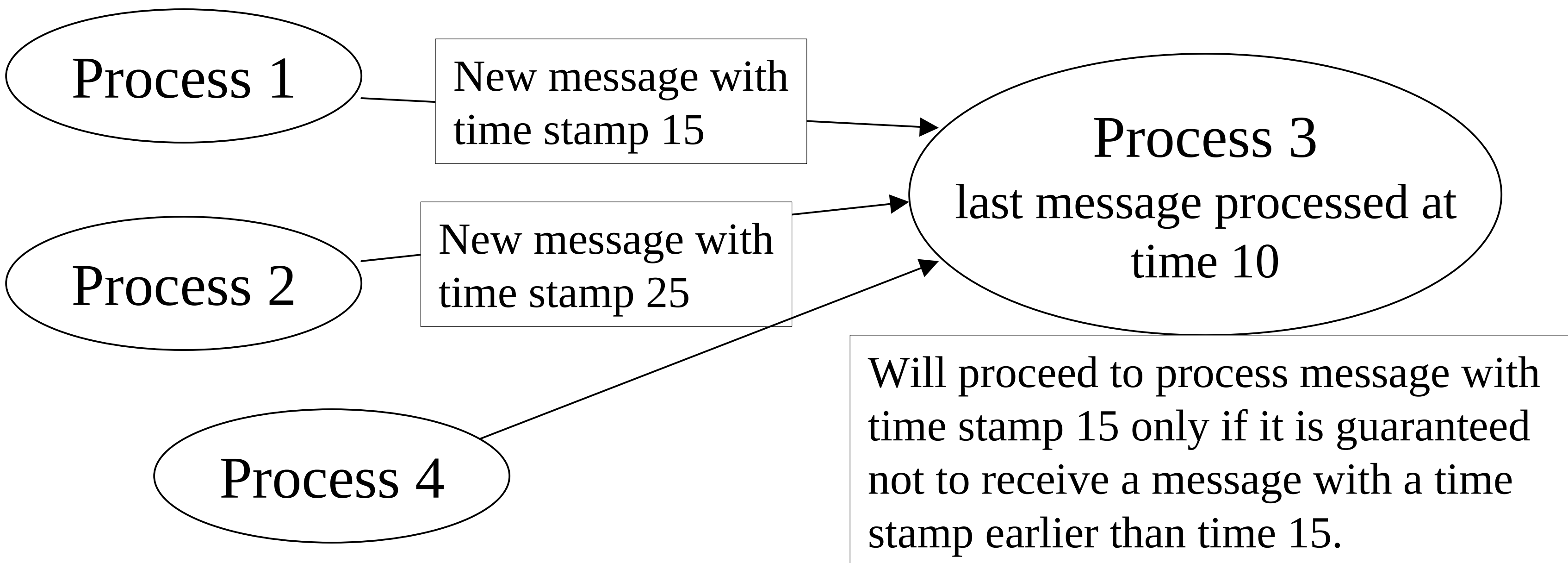
3. Each process has a local clock which is updated by the evaluation of events. The current time on the local clock is called the local virtual time or LVT. LVT is monotone non-decreasing and is updated when an event is evaluated.

4. Each event that is sent to another process for evaluation is given the LVT as a timestamp.



Parallel/Distributed Discrete Event Simulation

5. Each channel has a FIFO queue at the sink end of the channel. The events in each queue come from a single source. Therefore no event can be received from the source process with a timestamp smaller than the event at the tail of the channel queue.



6. A process can evaluate any locally generated events up to the minimum timestamp among the events at the tails of the channel queues. (Call it LVTH) But it cannot process any local events with time stamps larger than than LVTH since it might receive an external event with an earlier timestamp.

7. Suppose process P1 completes (or will not send any more messages) but P2 and P3 have additional work to do. Given this execution model P2 and P3 cannot progress unless they know that P1 has completed or will not further communicate.

8. One solution is for P1 to send a special event which notifies its neighbors of its termination if it terminates. Or if is still active it could send messages with null events and updated timestamps periodically. The receiving processes could close the incoming channel.

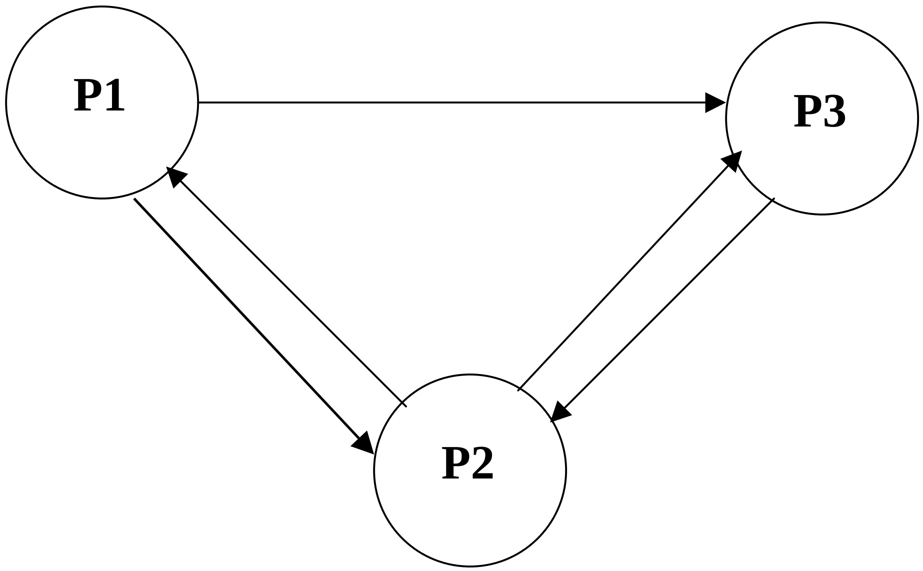
9. Or when P2 or P3 want to execute past a LVTH grounded on the channel from P1 they ask permission. (Note this may require additional channels and that P1 remains alive to respond.)

10 Lookahead - A process can often look at its internal event list and predict when it will send out the next event. It can then send out a null message with a timestamp of the time the next actual event will be generated. This will enable the other processes to proceed up to the time of the next event.

11. There are certain simulation models which cannot be executed using the CMB algorithm.

Optimistic (Jefferson or Time Warp) Parallel/Distributed Simulation

1. Let us partition the system into subsystems and agree to process the events in each subsystem by a separate process. The processes are connected by a set of one way channels. The processes interact by sending messages on the channels.



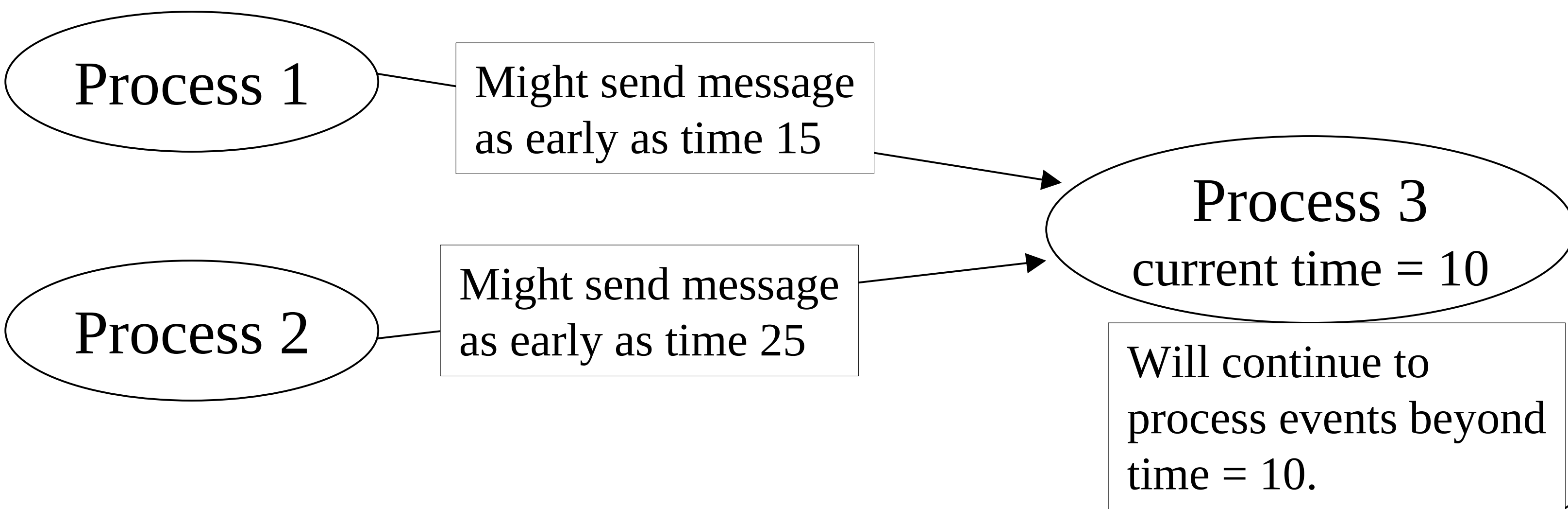
Each message carries an event, a timestamp and a directional marker. The directional marker may be either forward (positive) or backward (negative). More on the directional marker later.

Parallel/Distributed Discrete Event Simulation

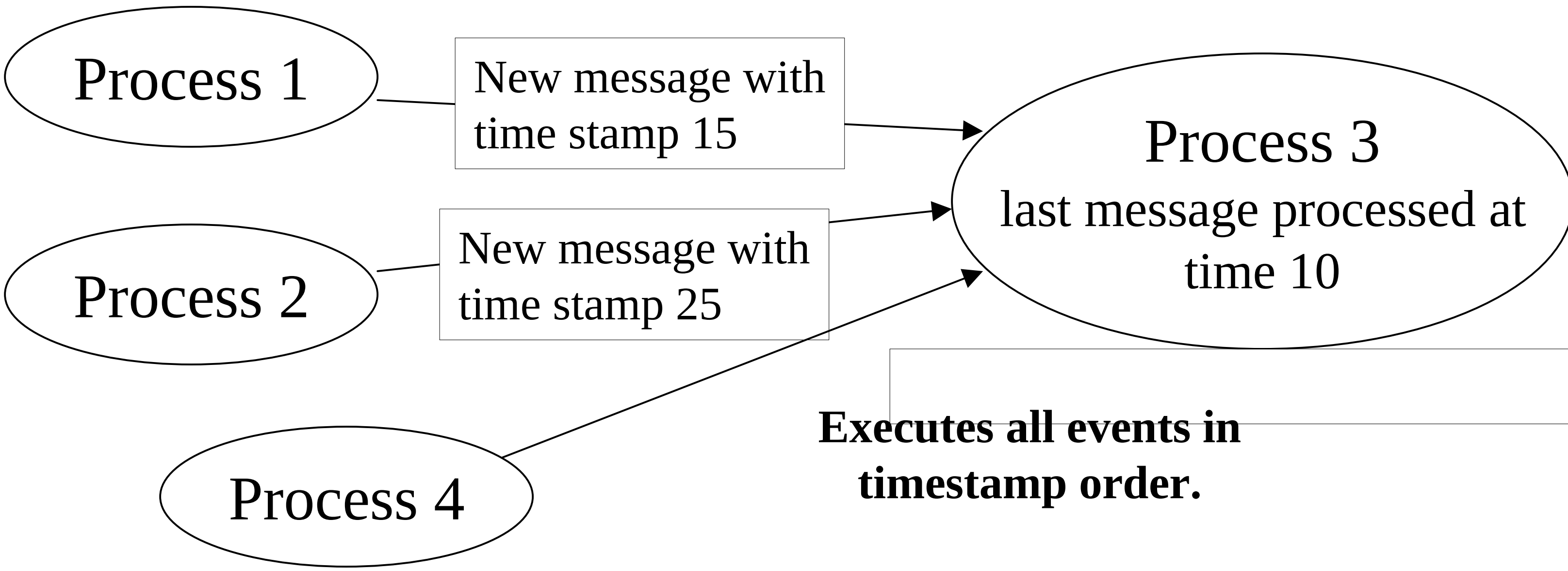
2. Each process may generate events for local processing, send events to other processes for evaluation and receive events from other processes.

3. Each process has a local clock which is updated by the evaluation of events. The current time on the local clock is called the local virtual time or LVT. LVT is monotone non-decreasing and is updated when an event is evaluated.

4. Each event that is sent to another process for evaluation is given the LVT as a timestamp.



5. Each Process has an single input queue for messages which are received from other processes and a single output queue for messages which are sent to other processes.



Executes all events in timestamp order.

Parallel/Distributed Discrete Event Simulation

6. A process executes all events in timestamp order unless it encounters a message with a timestamp earlier than some event it has already processed or a message with a backward directional marker.

7. Each process maintains a checkpoint and a log which enables it to rollback to a correct state. It also maintains a log of messages sent back to the checkpoint.

8. If a process does receive a message with a timestamp earlier than some messages it has already processed this is a violation of causality and the processing after this point must be rolled back.

9. But it can be even worse. Suppose that the process has, as a part of the processing events which must now be rolled back, sent messages to other processes.

10. Then each process while has received a message since the violation of causality occurred must be notified that it must rollback any processing it has done since receiving the first out-of-sequence message.

11. This is where the directional marker on the messages comes into play. The messages (with timestamps later than the timestamp of the out of order message) which have been sent by the process which received the out of order message are resent with a negative directional marker.

12. When a process receives a message with a negative directional marker one of two things can happen. The original message with positive direction is removed from the queue if it has not yet been processed. If there is any negative message which is not cancelled then the process knows it must rollback. Therefore it rolls back.

If it has also sent messages with timestamps later than the timestamp with the negative directional marker it will resend those messages with negative directional markers.

This is the cascading rollback phenomena. Clearly the depth of the rollback must be bounded.

Parallel/Distributed Discrete Event Simulation

14. Define a Global Virtual Time which is a lower bound to the minimum LVT of any process in the set of processes. There are many algorithms for computing a GVT.

15. Make the checkpoints for each process at the GVT and keep all of the messages received back to the GVT. The rollbacks will then go back no further than the GVT.

16. GVT is updated periodically at which times new checkpoints are taken.

17. Note that the optimistic algorithm can function with a dynamic set of processes.

> **A) Create a process with an LVT equal to the current GVT.**
>
> **B) Initiate a protocol to introduce the new process into the namespace of the processes.**
>
> **C) Initiate processing by the new process.**

Data Flow Graph Model of Parallel/Distributed Simulation

1. The execution model for the simulation is traversal of a dependence graph where the nodes are simulation engines which evaluate some set of events and the arcs carry the events which specify the dependences (causal relationships) among the events processed by the source and sink nodes of the arc.

2. There is a designated start node where execution begins. There is a designated stop node where execution terminates.

3. Each node is a seven tuple

- **a) a set of input ports - ports are typed FIFO queues**
- **b) a set of output ports**
- **c) a firing rule - a predicate over port states and local state**
- **d) a simulation engine**
- **e) a local clock - the LVT is updated as events are evaluated**
- **f) a routing rule - a predicate over port and local state which selects which output ports receive events.**
- **g) local state including a termination state which is set to true if it can be guaranteed that the node will not be enabled for execution again.**

4. An arc binds an output port of one node to an input port of another node.

5. A port is a FIFO queue of messages (events). A node may have many ports. Each port receives only one type of event.

6. A message is a three tuple (timestamp, event/data, termination state of the node.) A null message has a null event.

7. A firing rule is a predicate over the states of the input ports and the local state of the node. When it evaluates to true, events are taken from some set of input ports and bound to local state of the node and the node is enabled for execution.

8. A routing rule is a predicate over the local state of the node which selects one or more output ports to receive non-null messages. It may choose to place null messages on other ports.

Data Flow Execution Model

1. Execution begins at the start node and progresses by control flowing from node to node as the nodes complete processing of their internal events and send a message to their successor nodes.
2. The execution cycle of a node proceeds as follows:

   a) The firing rule becomes true, the node is enabled and the Lamport clock algorithm is invoked to establish an LVT.

   b) The simulation engine of the node begins execution with the initial state generated by satisfaction of the firing rule and executes until it completes the local processing associated with this firing.

   c) The routing rule is evaluated to select which output ports are to receive non-null messages and the messages a bound to the output ports.

   d) If this node is not a part of a backward loop then the termination state is set.

3. Execution completes when the stop node is executed.

How is this model of execution different from the interacting set of processes model of execution?

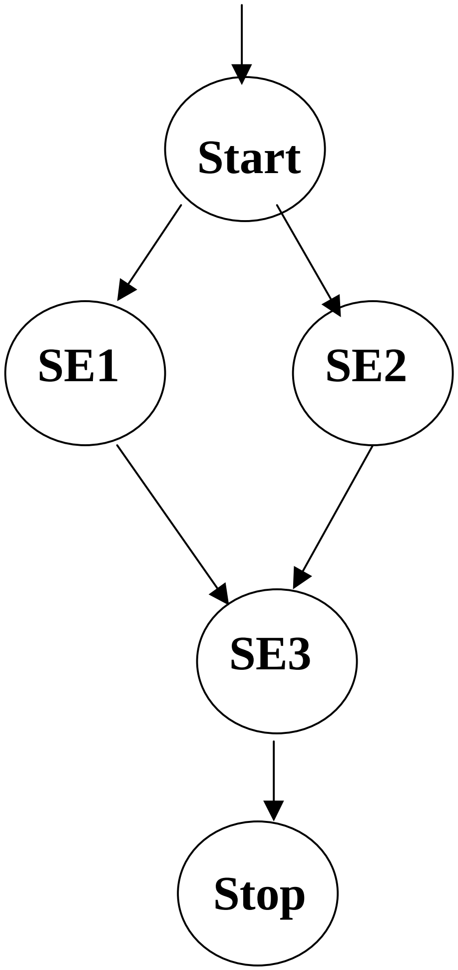
a) There is a concept of progress induced by traversal of the graph. Progress is captured by Lamport clocks.

b) Relationships among externally events are explicitly modeled.

c) The execution of each node is atomic. All events generated for local processing in a given cycle are evaluated in that cycle. A node, once, initiated will always complete execution of all locally generated events before receiving another externally generated event.

d) Because execution order is determined by data flow graph structure a node cannot receive an event with a timestamp earlier than the time stamp which initiated its current execution cycle.

Parallel/Distributed Discrete Event Simulation



Let us look at some of the possible cases for control flow.

1. The firing rule for the SE3 node is an "and" over the two inputs from SE1 and SE2. The Lamport clock algorithm (LCA) will initiate execution of SE3 with a LVT which is the largest of the LVTs of SE1, SE2 and SE3.

2. The firing rule for the SE3 node is an "or" over the two inputs from SE1 and SE2. SE3 will be enabled for execution whenever one of SE1 or SE2 places an message in a port of SE3. The LCA will initiate execution of SE3 with an LVT which is the largest of the local LVT and the LVT of the enabling event.

3. A possible sequence of execution is that an event arrives at SE3 from SE1 with a timestamp of 5. SE3 has an initialized timestamp of 0. Execution of SE3 therefore begins with an LVT of 5.

4. The next event may be the arrival at SE3 of a message from SE2 with a timestamp of 4. The LCA sets the LVT for this initiation of SE3 at the LVT at the completion of the last execution plus 1.

5. Does this violate causality?

> **A) Yes - because the event from SE2 arrived at SE3 with a timestamp less than the event from SE1.**
>
> **B) No - Because the execution of SE3 is enabled by the arrival of an event from either of SE1 or SE2.**

6. Is the result of the simulation reproducible?

> **Yes - so long as the same random number seeds are used in each simulation.**

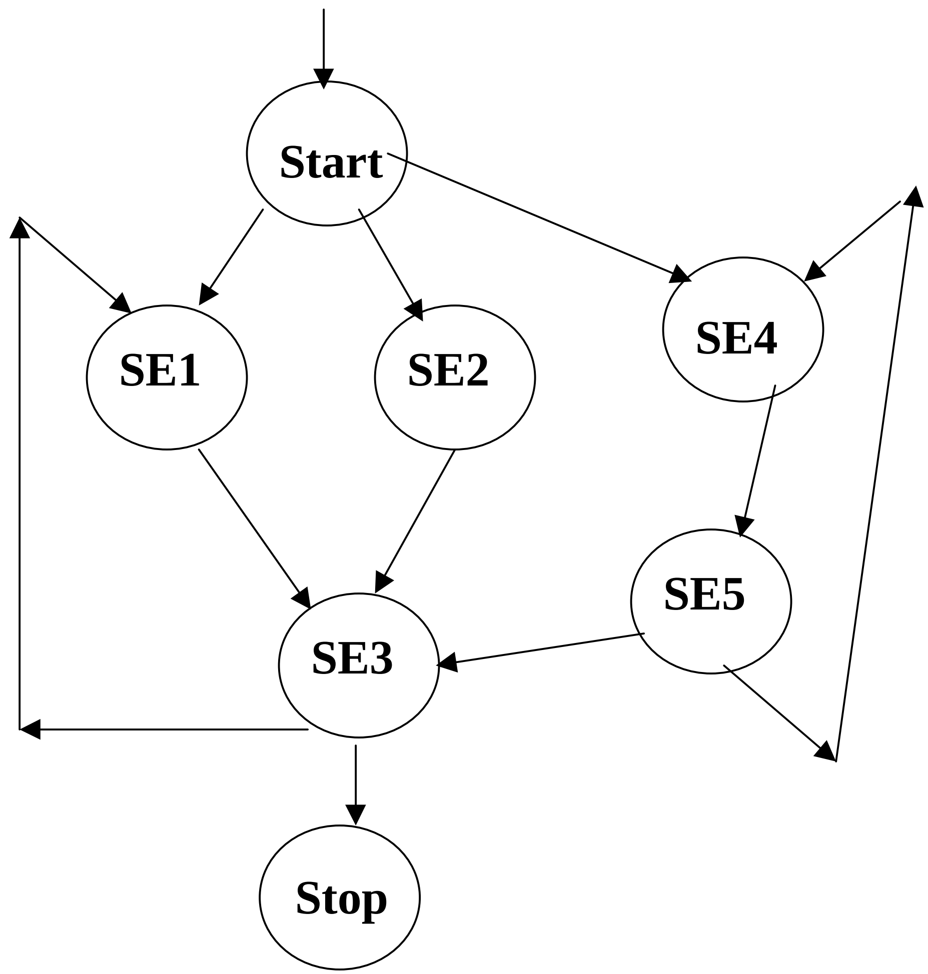
7. Will this execution model produce a sequence of events consistent with some correct serial execution of the simulation?

> **Yes- because the times taken to follow each path may be chosen by random selection of execution times.**

8. If the execution times are deterministic will this execution model produce a sequence of events consistent with some correct serial execution of the simulation?

> **Yes - It will produce the same sequence as would be generated by the CMB simulation.**

Parallel/Distributed Discrete Event Simulation



The logic of the program is made explicit by the firing rules and the dependence relations. The routing rule of SE5 must be an "or" if the simulation is to terminate.

Each node can chose to send a null message to those successors which do not receive an event.

The order in which SE3 should receive events from SE1, SE2 and SE5 is not determined. The simulation must be correct for any order of receipt of events.