Dicii

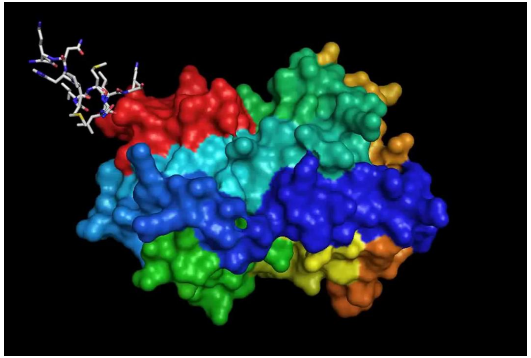
Università di Roma Tor Vergata

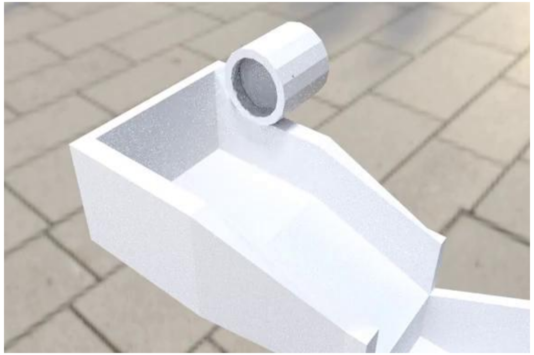
Parallel Discrete Event Simulation

Alessandro Pellegrini
a.pellegrini@ing.uniroma2.it

Simulation: what’s that?

- It’s an *umbrella term*
- From latin *simulare* (to mimic or to fake)
- It is the imitation of a real-world process' or system's operation over time
- It allows to collect results long before a system is actually built (*what-if analysis*)
- It can be used to drive physical systems (*symbiotic simulation*)
- Widely used: medicine, biology, physics, economics, sociology, ...





Simulation vs Models

- The terms *simulation* and *model* are often used synonymously
- However, they are fundamentally distinct
- **Model**
  - a representation of a portion of the world/of a system of interest
  - typically simpler than the system it represents
    - it approximates most of the salient features of the real system as close as possible.
  - typically a judicious tradeoff between realism and simplicity.
- **Simulation**
  - the *process* of using a model to study the behaviour and performance of the system of interest
  - the purpose is to study the behavior of the system manipulating variables that couldn’t be controlled in the real system
  - a simulation can use a model to explore states that would not be possible in the original system.

Efficient Simulation

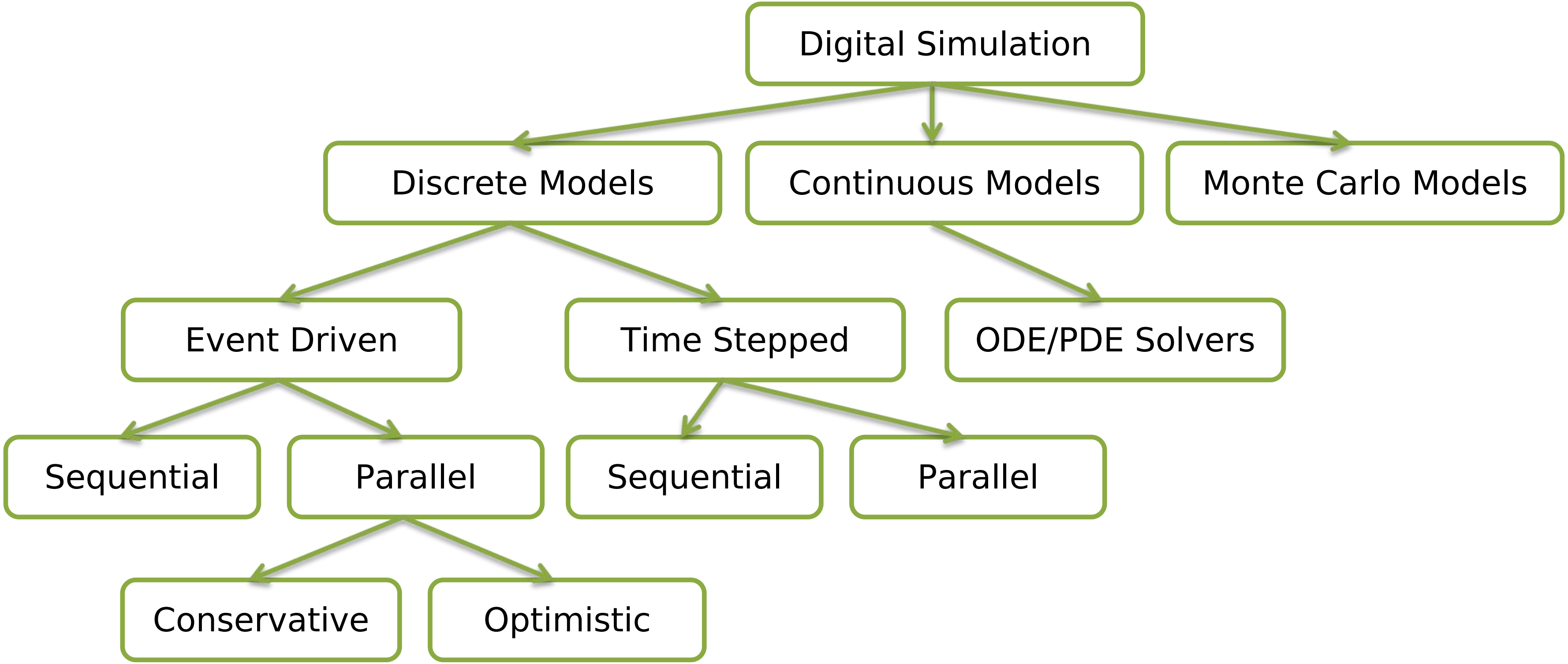
- The ultimate goal of models is to allow conducting simulation studies
- Models can be large and complex
- The time required to complete the execution of a model can be large
- To perform a simulation study, a large number of execution of the same model may be required
  - calibration
  - validation
  - exploration
- Reducing the time required to run a model can dramatically reduce the time to perform a simulation study

Different Notions of Time

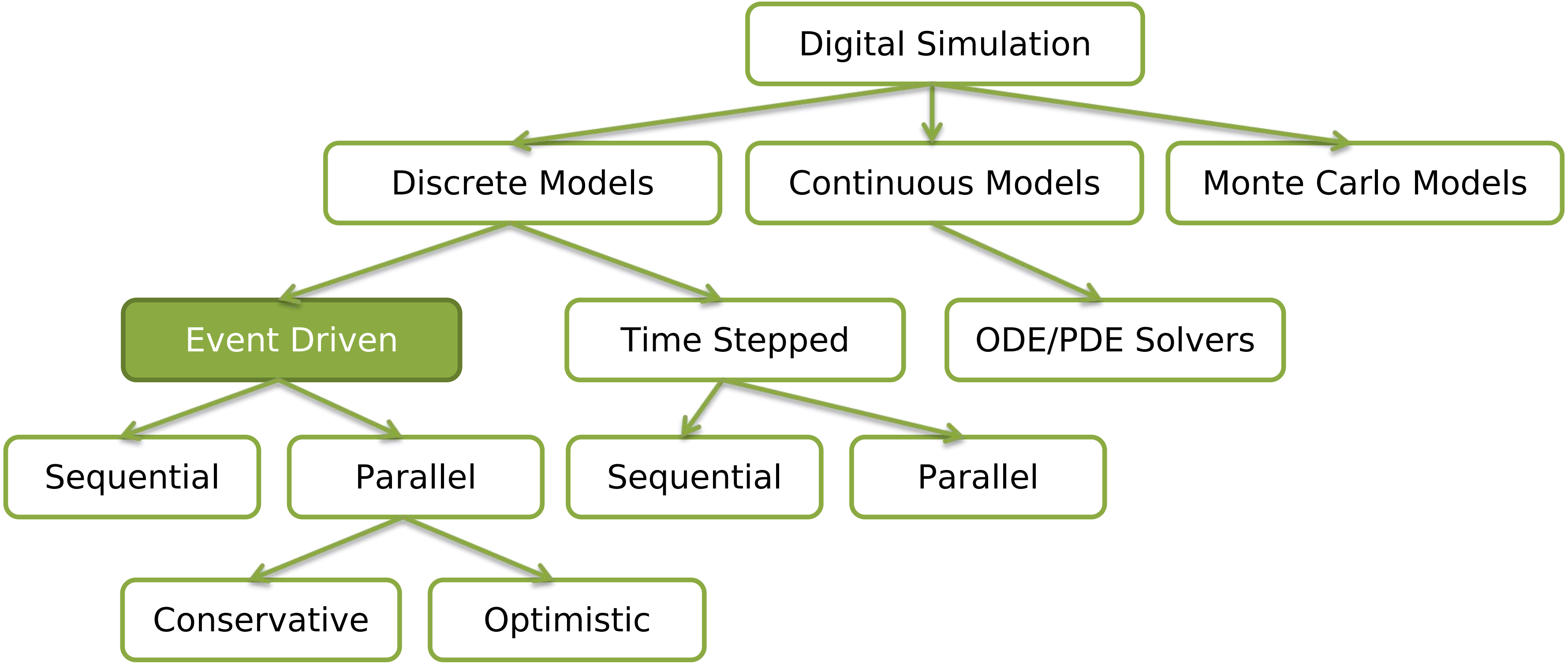
- Three different notions of time are present in a simulation

- **Wall-Clock Time**: the *elapsed time* required to carry on a digital simulation (the shorter, the higher is the efficiency)
- **Logical Time**: the actual *simulated time*
  - Also referred to as *simulation time* or *virtual time*
- **Physical Time**: the notion of time observed in the (modelled) physical system

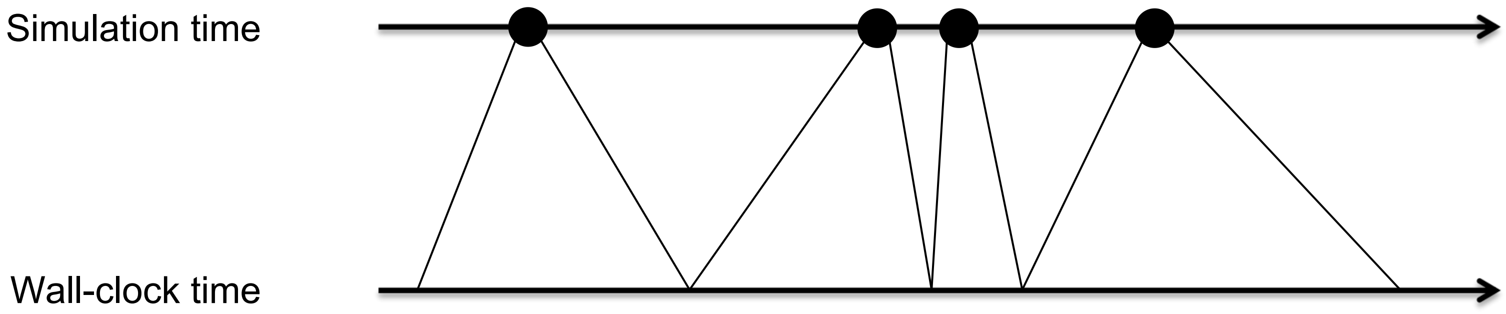
Simulation Taxonomy



Simulation Taxonomy



Wall-Clock Time vs Logical Time



Event-Driven Programming

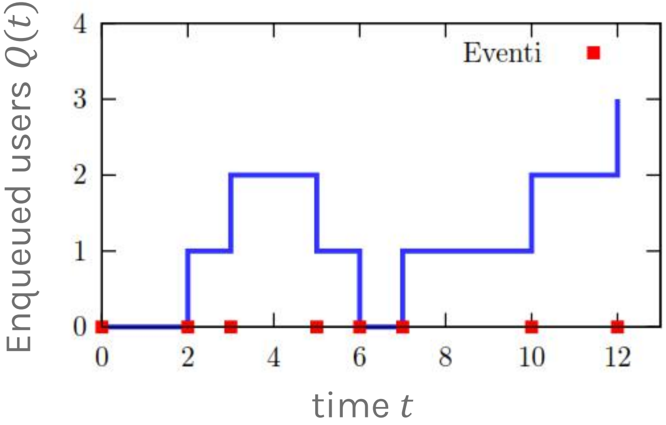
- Event-Driven Programming is a programming paradigm in which the flow of the program is determined by *events*
    - Sensors outputs
    - User actions
    - Messages from other programs or threads
- Based on a *main loop* divided into two phases:
    - Event selection/detection
    - Event handling
- Events resemble what *interrupts* do in hardware systems

Event Handlers

- An event handler is an *asynchronous callback*
- Each event represents a piece of application-level information, delivered from the underlying framework:
  - In a GUI, events can be mouse movements, key pression, action selection, . . .
- Events are processed by an *event dispatcher*, which manages associations between events and event handlers and notifies the correct handler
- Events can be *queued* for later processing if the involved handler is busy at the moment

The Queueing Server Example

t	Event	$Q(t)$
0	INIT	0
2	Request 1	1
3	Request 2	2
5	Complete 1	1
6	Complete 2	0
7	Request 3	1
10	Request 4	2
12	Request 5	3



This may involve data structures setup

Running a DES Model: Building Blocks

- **Clock**
  - Independently of the measuring unit, the simulation must keep track of the current simulation time
  - Being discrete, time *hops* to the next event's time
- **Event List**
  - At least the *pending event set* must be maintained by the simulation architecture
  - Events can arrive at a higher rate than they can be processed
- **Objects**
  - Represent discrete objects in the system being simulated!
  - Also called LPs (Logical Processes)
- **Simulation State**
  - Represents the physical state of the object being modeled!
- **Random Number Generators**
- **Statistics**
- **Ending Condition**

Sequential DES Core Skeleton

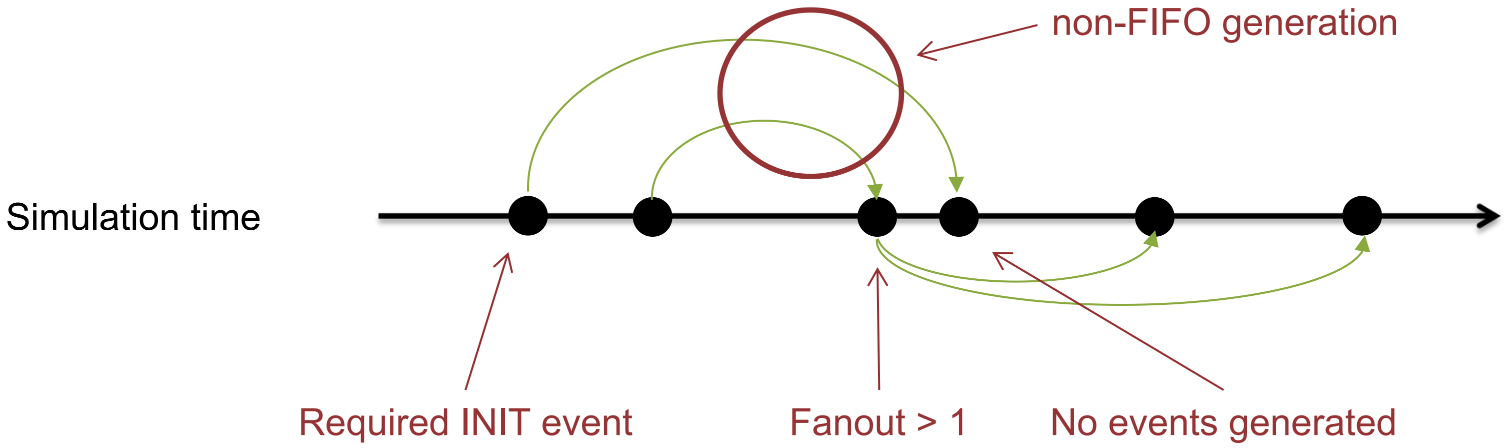
```
procedure INIT
    End ← false
    initialize State, Clock
    schedule INIT
end procedure

procedure SIMULATION-LOOP
    while End == false do
        Clock ← next event's time
        process next event
        Update Statistics
    end while
end procedure
```

Efficient, scalable, easy-to-use parallelization of this one simple algorithm is what the rest of this lecture is about

Events Generation Relationships

- The execution of an event can generate new events
- Events can be scheduled only *in the present or in the future*



Management of Events

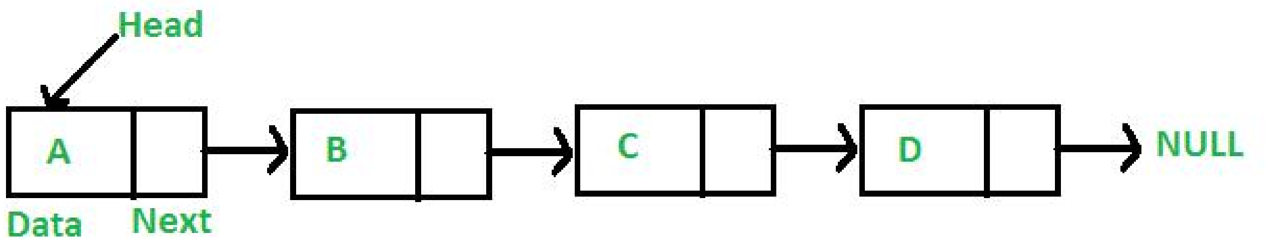
- When executing an event, other events can be injected
- A single event can generate more than one event in the future
  - The *fanout* of an event
- Yet, we can process one event at a time
- There is the need to maintain future events
  - *Future Event Set* or *Pending Event Set*
- The *scheduler* of the simulation kernel must then decide what is the next event to execute
  - All events *must* be executed in strict timestamp order

Data Structures for Simulation: Priority Queue

- Is an abstract data type similar to a regular queue
- Elements have a priority associated with each of them
- An element with a high priority is served before
- Operations:
    - insert with priority: add an element to the queue with associated priority
    - pull highest priority element: remove the element from the queue that has the highest priority, and return it
- Highest priority can be either minimum or maximum value
- It can be used to implement the FEL
    - What about the ordering of simultaneous events?

Data Structures for Simulation: Linked List

- The most classical implementation of an *ordered set*
- A node keeps an event, and nodes are ordered by ascending timestamp
- Extraction is easy: you can always peek the head node
- Insertion can be costly: in the worst-case scenario, you have to scan the whole list: $O(n)$ complexity



Data Structures for Simulation: Calendar Queue

- It's based on the concept of a *desk calendar*
  - In each day, you can set appointments
  - They are ordered by time
- It's a *cheap* desk calendar
  - We use just one sheet for all months
  - A day can keep appointments belonging to different months

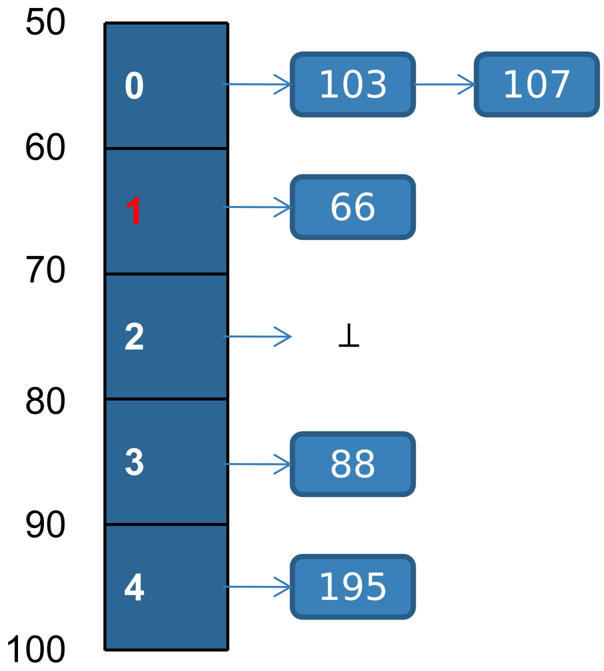
Calendar Queue (Brown 1988)

- The time axis is divided into *buckets*, each of which has a certain *width* (or *time coverage*) w.
  - Only n buckets are physically allocated
- It maintains the notion of “last extracted priority” (or *current time*)
- Upon inserting an element with priority $p > current\ time$, it will be inserted in the bucket:

$$\left\lfloor \frac{p}{w} \right\rfloor \bmod n$$

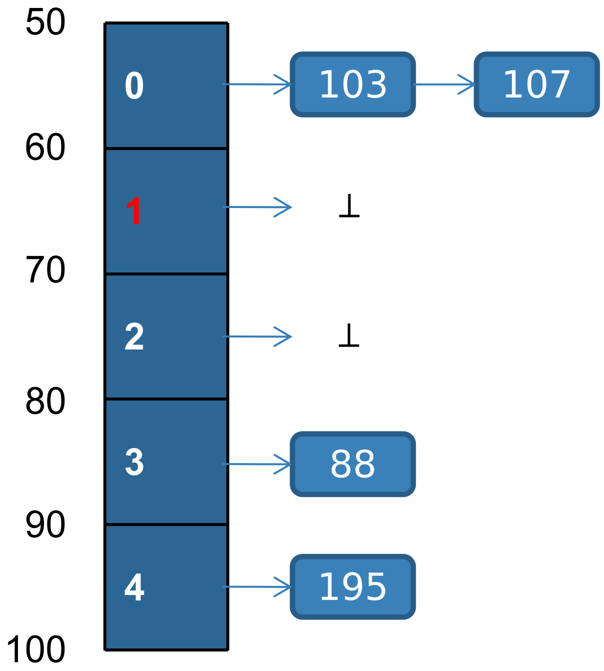
- n and w should be chosen so as to minimize the number of elements in each bucket
  - *Resize* operation: double/halve n if the number of elements per bucket grows/shrinks too much

Calendar Queue (Brown 1988)



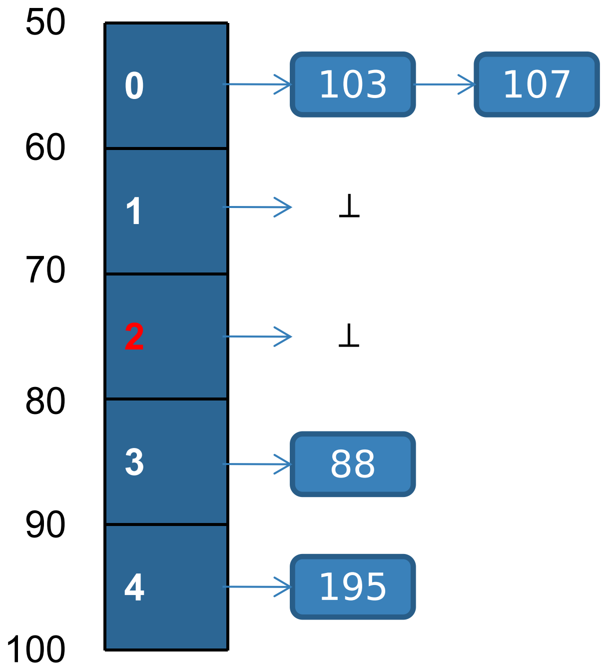
- 5 buckets
- width = 10
- current time = 63

Calendar Queue (Brown 1988)



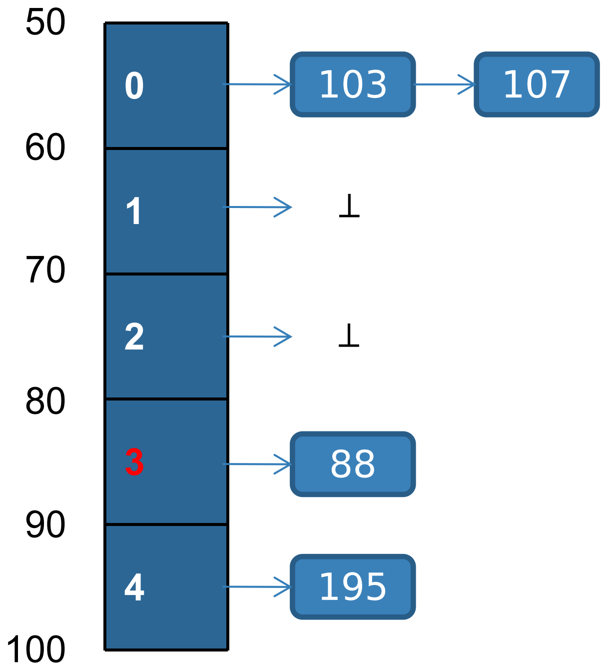
- 5 buckets
- width = 10
- current time = 66

Calendar Queue (Brown 1988)



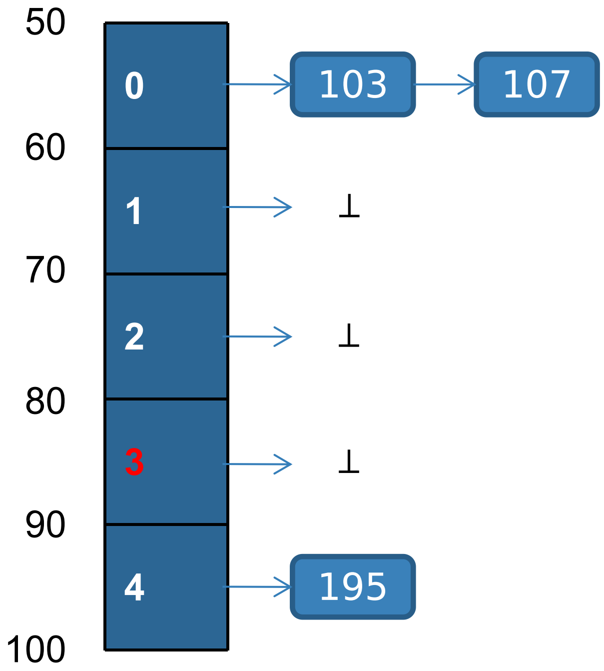
- 5 buckets
- width = 10
- current time = 66

Calendar Queue (Brown 1988)



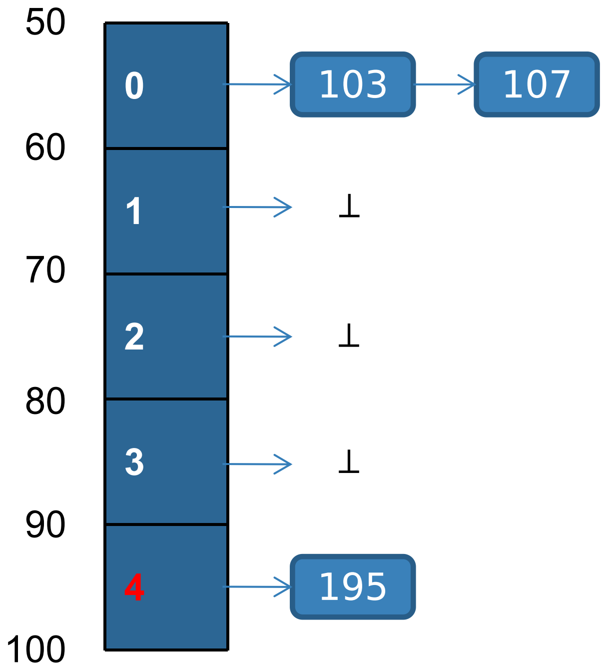
- 5 buckets
- width = 10
- current time = 66

Calendar Queue (Brown 1988)



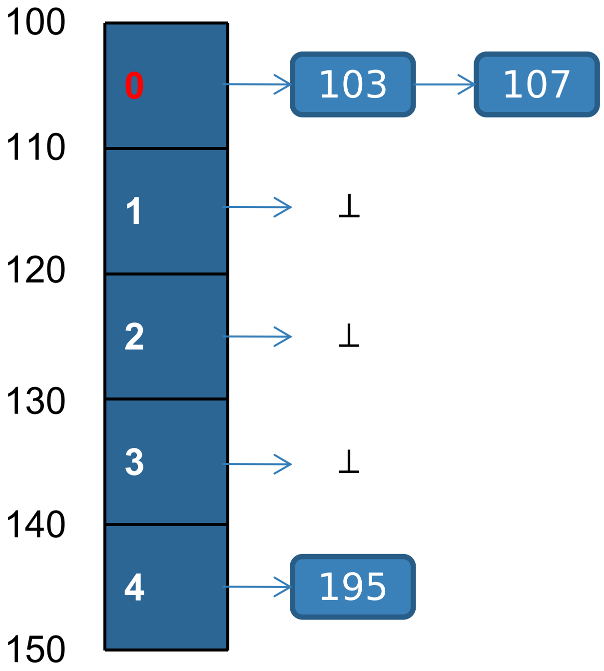
- 5 buckets
- width = 10
- current time = 88

Calendar Queue (Brown 1988)



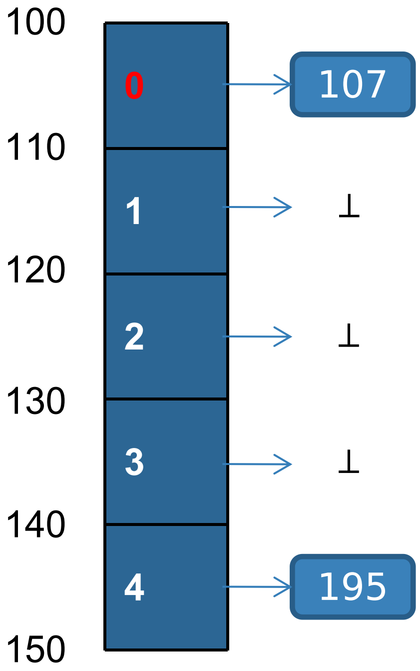
- 5 buckets
- width = 10
- current time = 88

Calendar Queue (Brown 1988)



- 5 buckets
- width = 10
- current time = 88

Calendar Queue (Brown 1988)



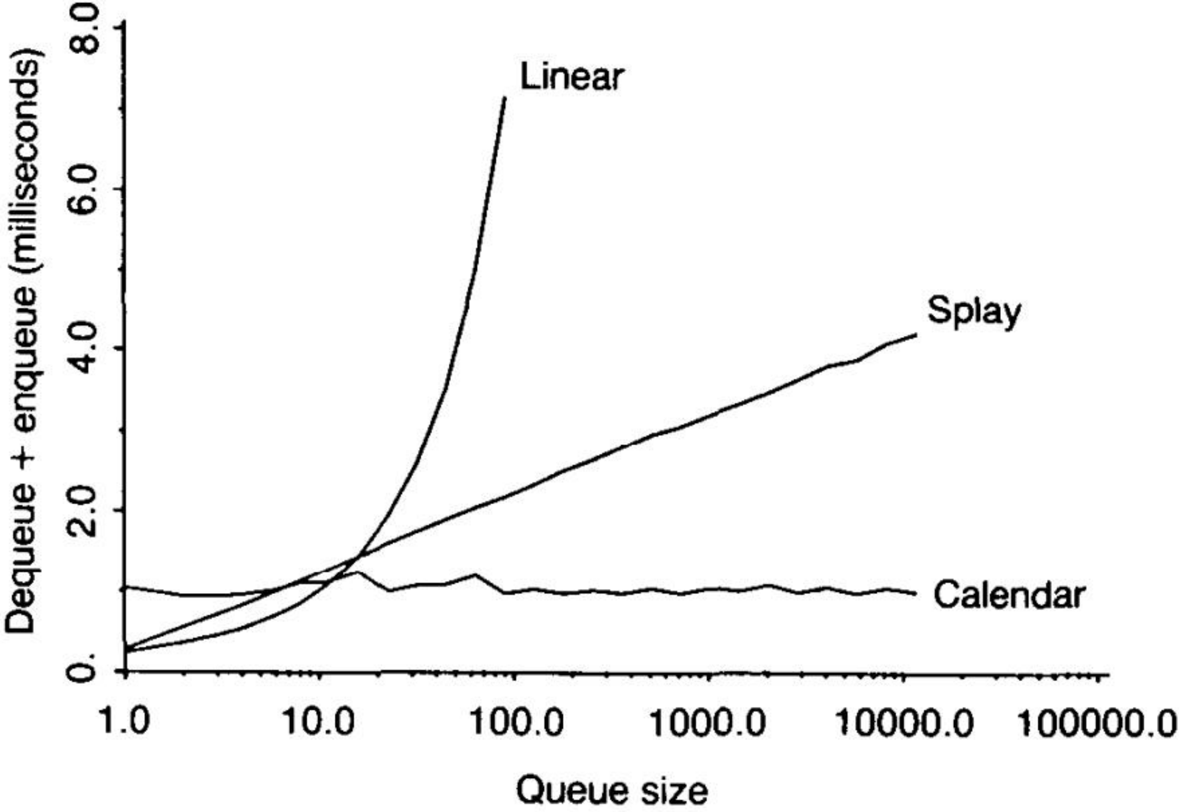
- 5 buckets
- width = 10
- current time = 103

Calendar resize

- Based on a statistic approach
  - Recompute w considering *average event separation*
  - This approach works well if, in the upcoming future, event timestamps already have a uniform distribution
  - To reduce problems: exclude from computation events with a too-large separation

- The new time coverage is computed as $3 \cdot \overline{separation}$

Empirical cost: $O(1)$



Example Session

Performance of Future Event Sets

Space Partitioning: Hexagonal Worlds

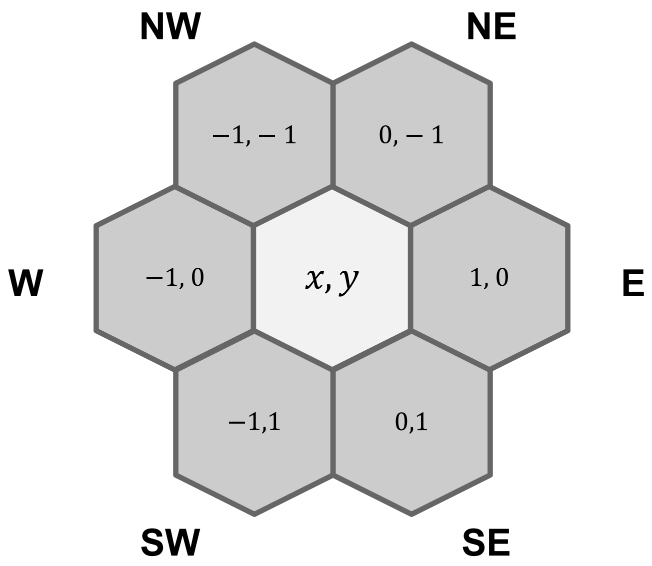
- The namespace of LPs which we use is the interval $[0, \infty)$
- A hexagonal world follows a different namespace
- To handle the topology, we must perform a *linear to hex* translation

$$edge = \sqrt{num\ LPs}$$

$$x = LPid \bmod edge$$

$$y = \left\lfloor \frac{LPid}{edge} \right\rfloor$$

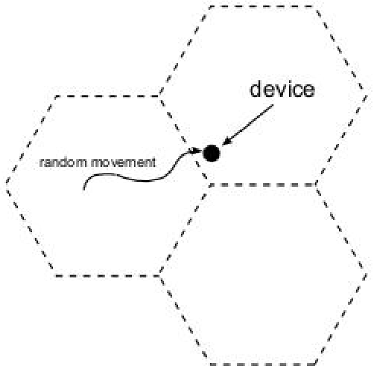
Hexagonal World: Neighbors




```
case NW:
        nx = (y % 2 == 0 ? x - 1 : x);
        ny = y - 1;
        break;
case NE:
        nx = (y % 2 == 0 ? x : x + 1);
        ny = y - 1;
        break;
case SW:
        nx = (y % 2 == 0 ? x - 1 : x);
        ny = y + 1;
        break;
case SE:
        nx = (y % 2 == 0 ? x : x + 1);
        ny = y + 1;
        break;
case E:
        nx = x + 1;
        ny = y;
        break;
case W:
        nx = x - 1;
        ny = y;
        break;
```

Personal Communication Service (PCS)

- Mobile network adhering to GSM technology
- Each LP is a hexagonal cell
- All cells offer network coverage to a squared region
- Upon the start of a call, a call-setup record is created
  - Keep track of metadata related to a single channel
  - Power, Fading, SIR, and Path Gain are accurately computed
  - Records organized in a linked list
- If no channel is available, the call is dropped
- Devices are subject to random movement
  - A device might move to some adjacent cell
  - The call is then transferred to the new cell
  - The channel is freed in the current cell
  - A new channel (if available) is setup in the destination cell



Personal Communication Service (PCS)

- Model parameters:
  - Number of channels in the cells
  - τ_A: intera-arrival time of subsequent call (varied based on time)
  - τ_D: average expected call duration
  - τ_C: average expected residual time of a device into the current cell

- Overall utilization factor is:

$$\eta = \frac{\tau_D}{N \cdot \tau_A}$$

- η affects the granularity of events:
  - the higher the number of used calls, the higher the computational cost to recompute path gain and fading

Example Session

Personal Communication Service

Parallel/Distributed Simulation: System Aspects

- Multiple concurrent process/threads must *cooperate* while executing the model
- Processes live in different address spaces (especially critical in distributed setups)
- A *message transfer layer* is needed to provide primitives for process coordination
  - E.g., data dependency is supported via message-exchange
  - Reference messaging layer: Message Passing Interface (MPI)

Goals for PDES Simulation

- Speedup from running events in parallel is primary
  - Other kinds of parallelism can be combined, but are orthogonal
  - Occasional secondary goals:
    - access more RAM by splitting large simulations over many nodes
    - isolate different components of federated model onto different nodes
- Efficiency is *not* the primary goal
  - Efficiency is only useful insofar as it contributes to speed
  - If we can run faster by using more resources, or using them less efficiently, we will do so!
  - Efficiency can be addressed at a later stage

Entity partitioning

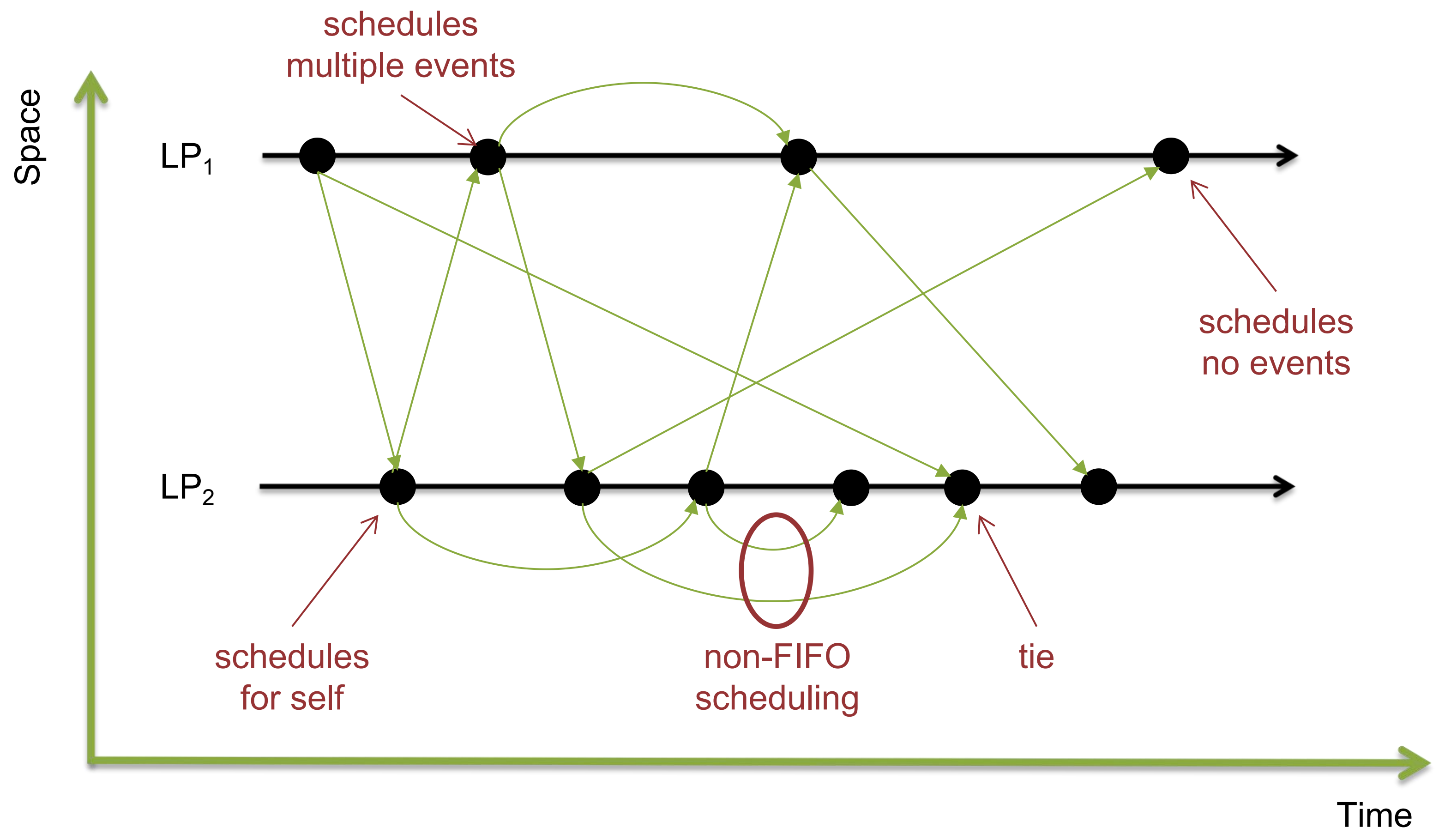
- The DES model is partitioned into *N* different entities, each one representing a portion of the whole simulated system
- The different entities have been historically named Logical Processes (LPs)
- The evolution of the state of each individual LP mimics the evolution of the corresponding sub-portion of the simulation model
- The states of the LPs are *disjoint*, and the state of the simulation model is represented by the *union* of individual LP states:

$$S = \bigcup_{i=0}^{N-1} S_i \quad \wedge \quad S_i \cap S_j = \emptyset, \forall i \neq j$$

Entity partitioning

- LPs can process simulation events *concurrently*
- Each LP has its own view of the current simulation time (Local Virtual Time – LVT)
    - At a given Wall-Clock Time instant, two different LPs can be at a different Simulation Time
    - This is only possible thanks to state disjointness

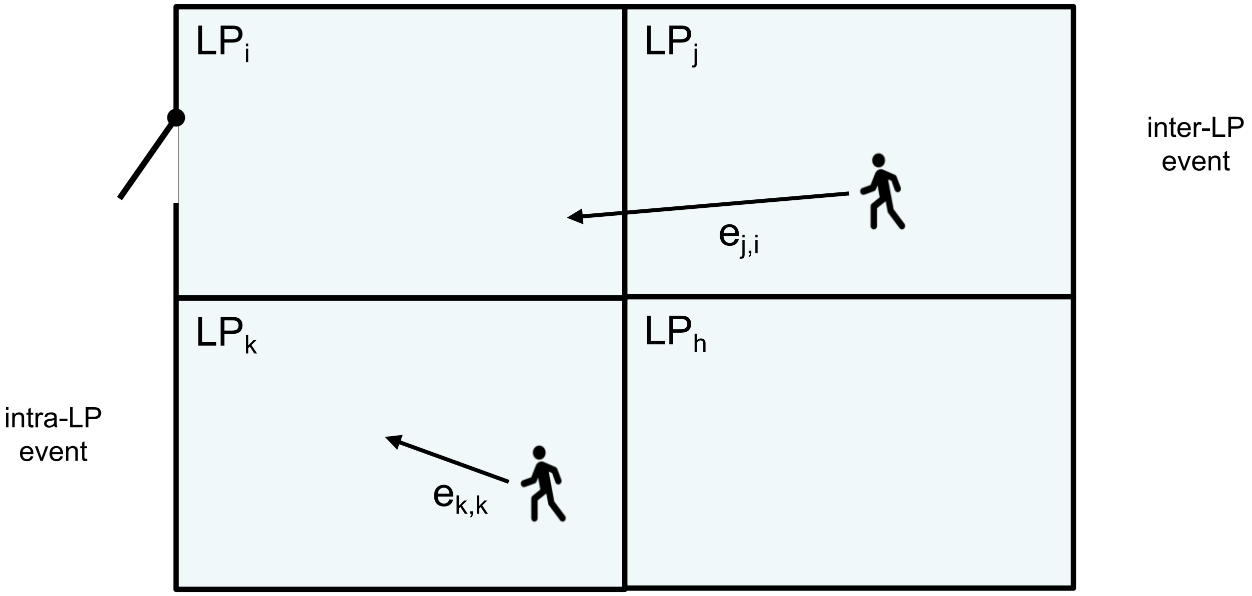
Events Generation Relationships: Space-Time



The Synchronization Problem

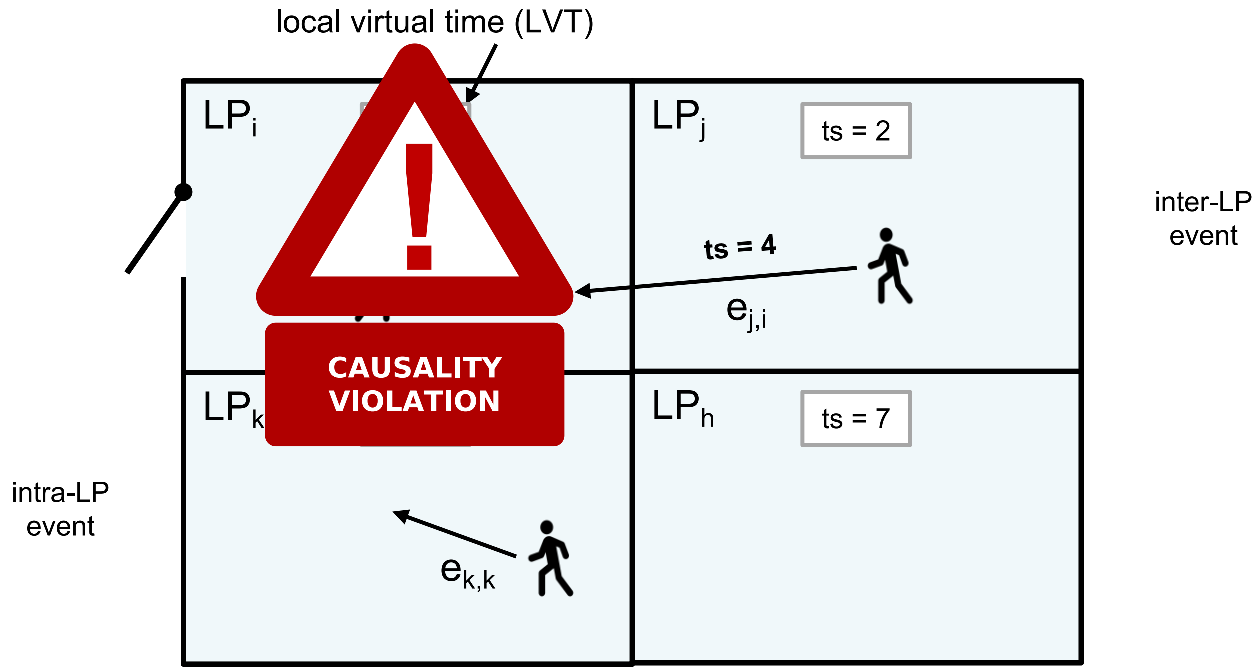
- Consider a simulation program composed of several *logical processes* exchanging timestamped messages
- Consider the *sequential execution*: this ensures that events are processed in timestamp order
- Consider the *parallel execution*: the greatest opportunity arises from processing events from different LPs concurrently
- Is *correctness* always ensured?

The Synchronization Problem



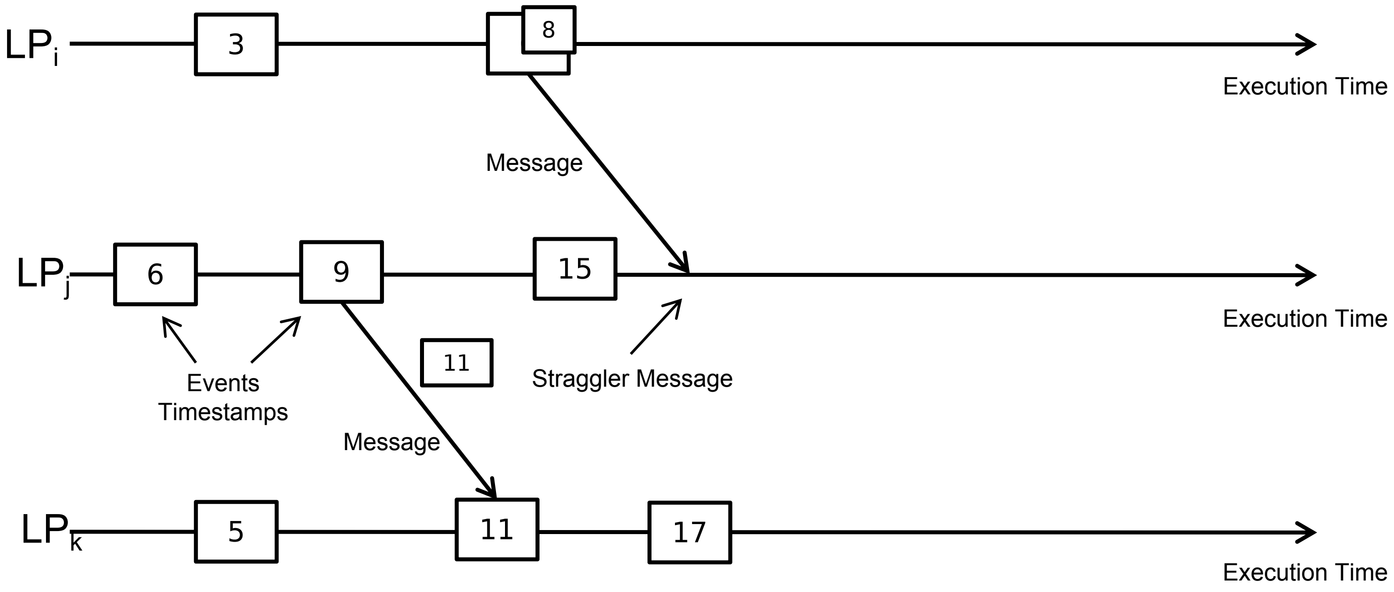
Simulated Surface

The Synchronization Problem

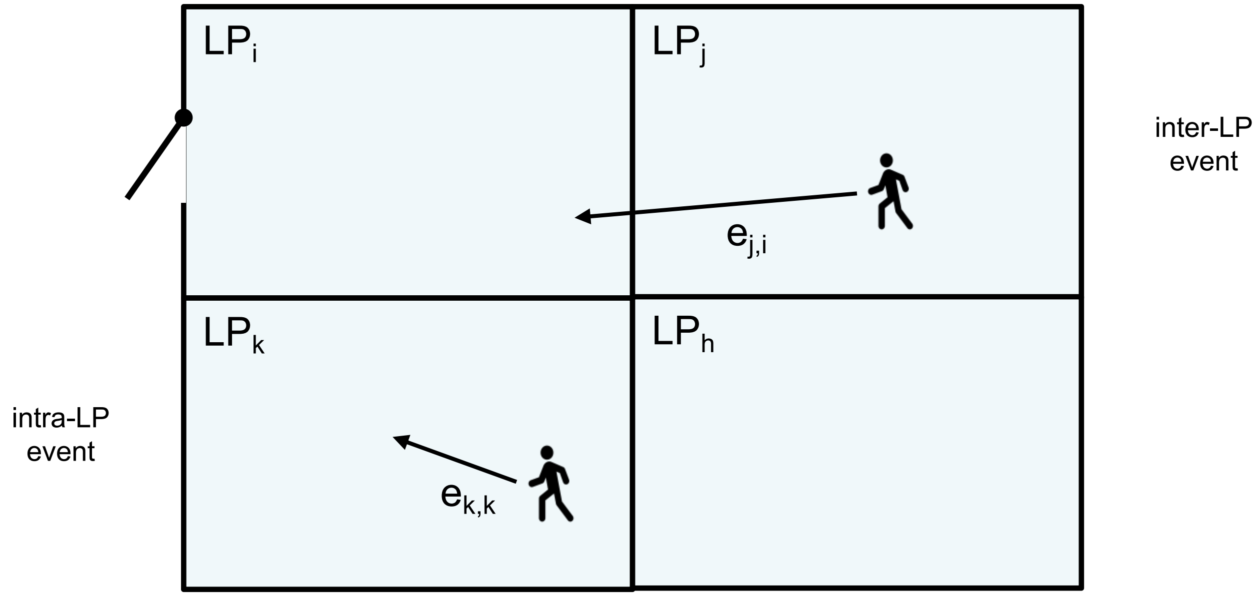


Simulated Surface

The Synchronization Problem

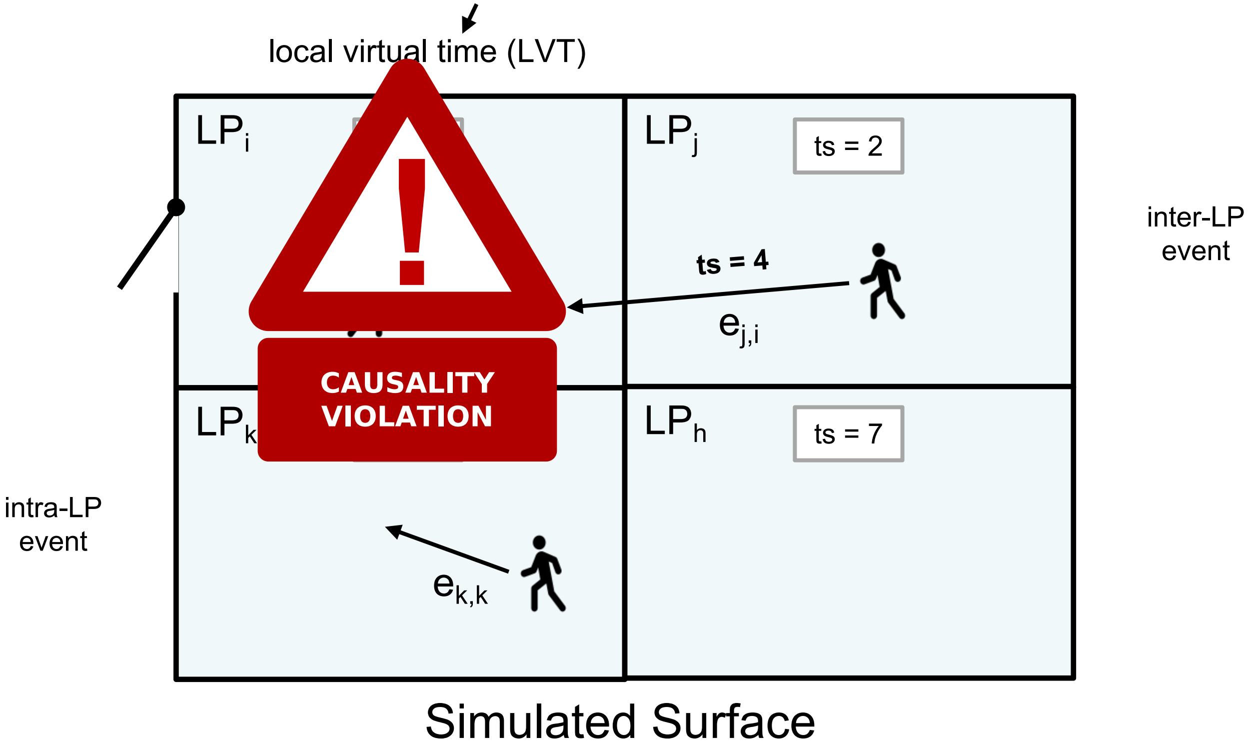


The Synchronization Problem

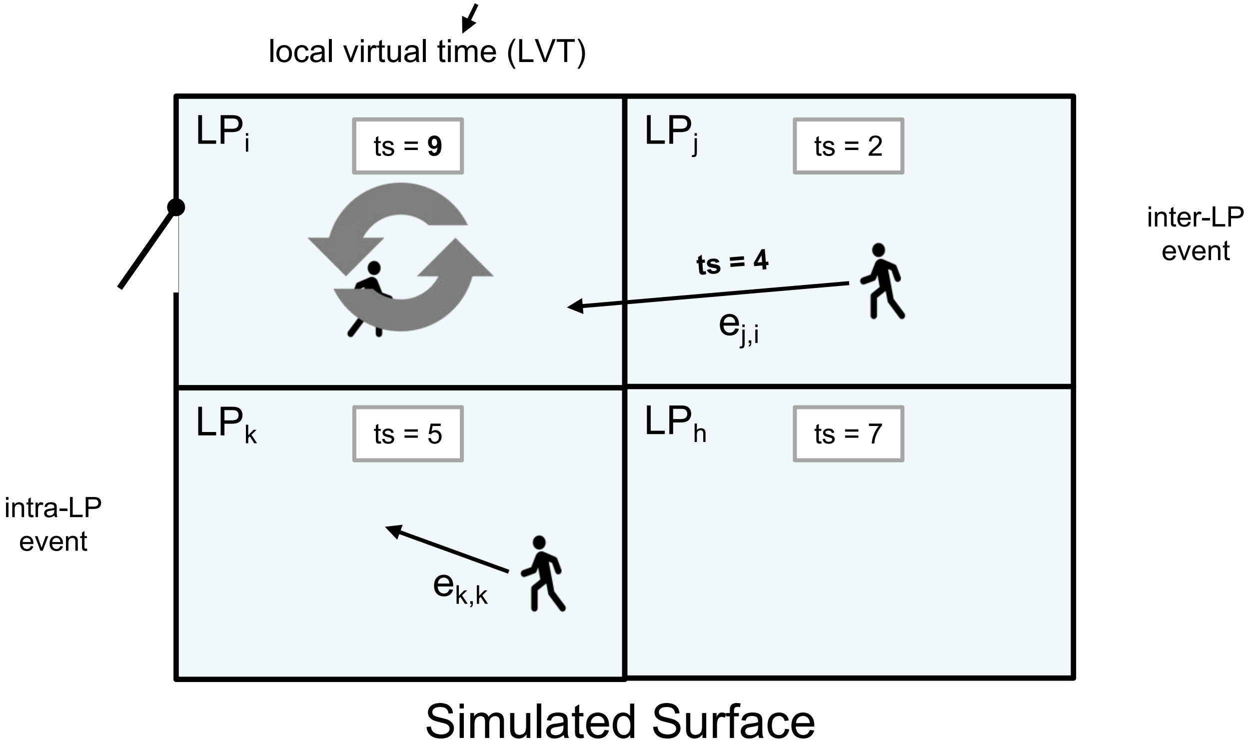


Simulated Surface

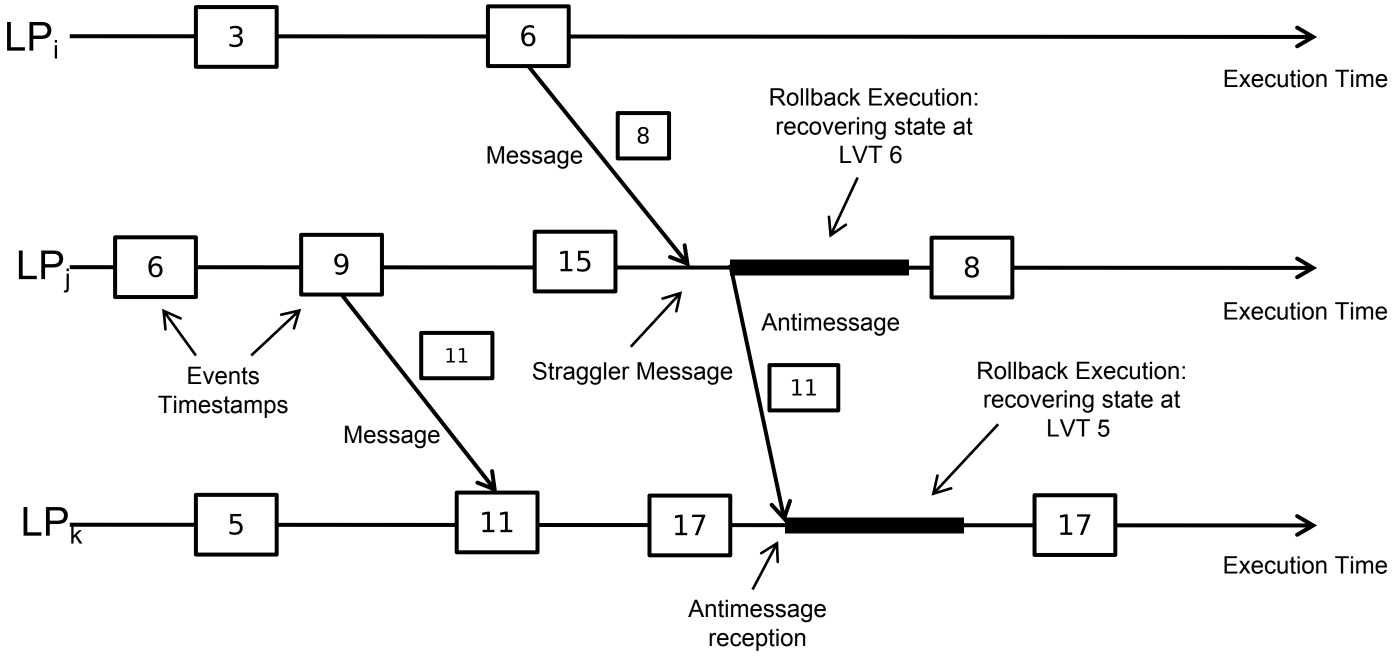
The Synchronization Problem



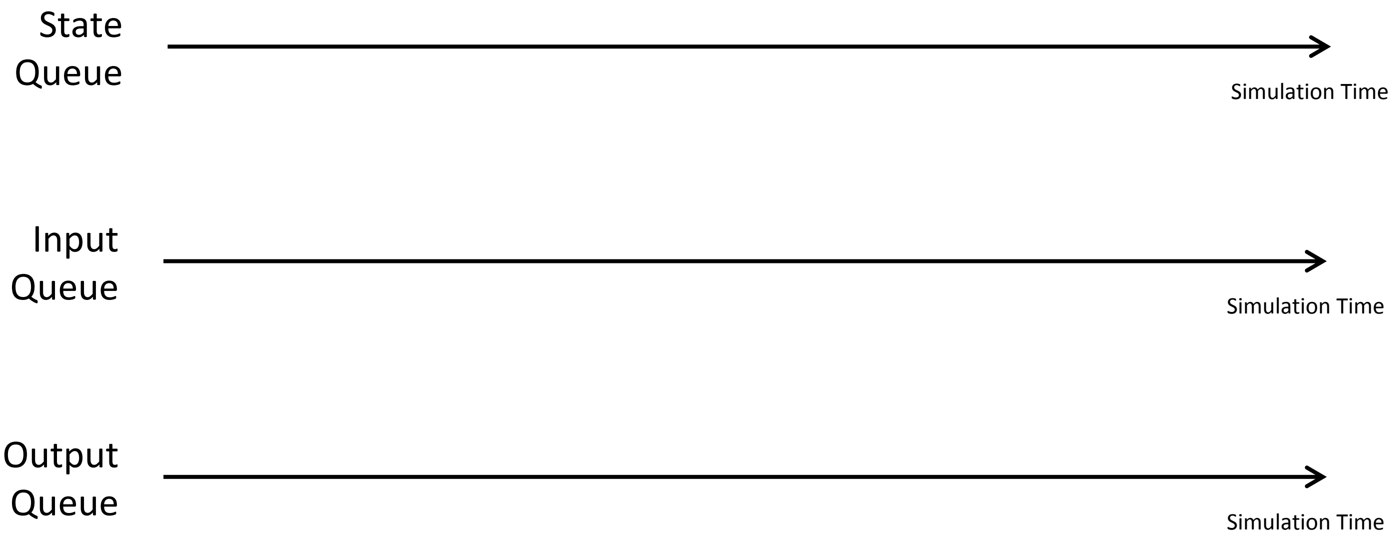
The Synchronization Problem



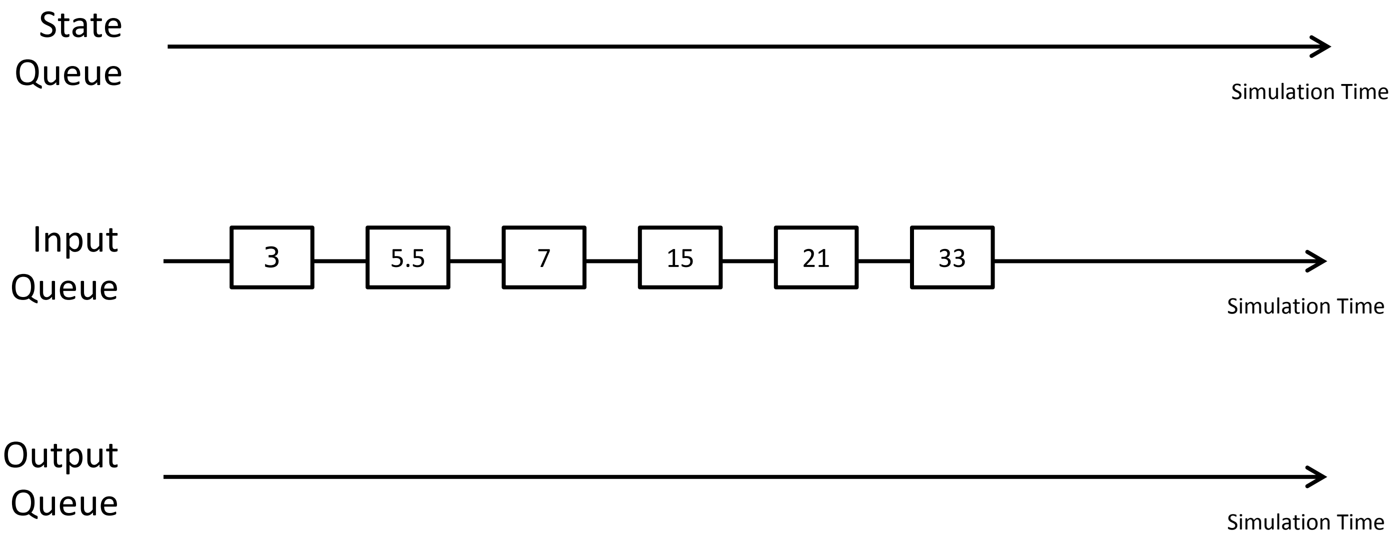
Time Warp: State Recoverability



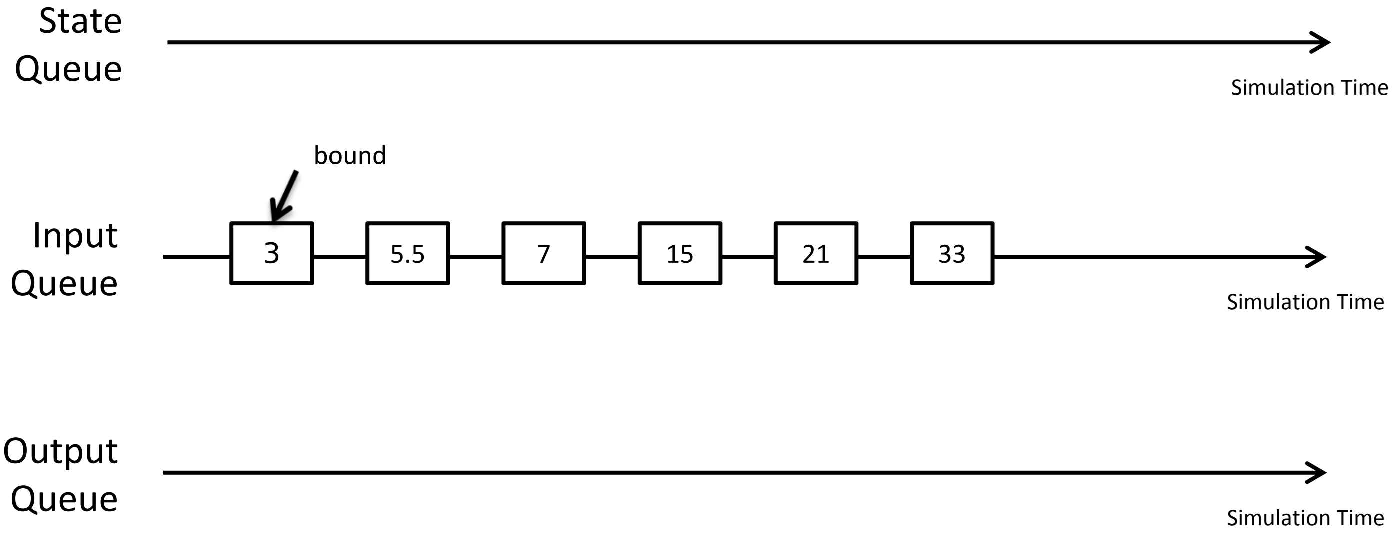
State Saving and Restore



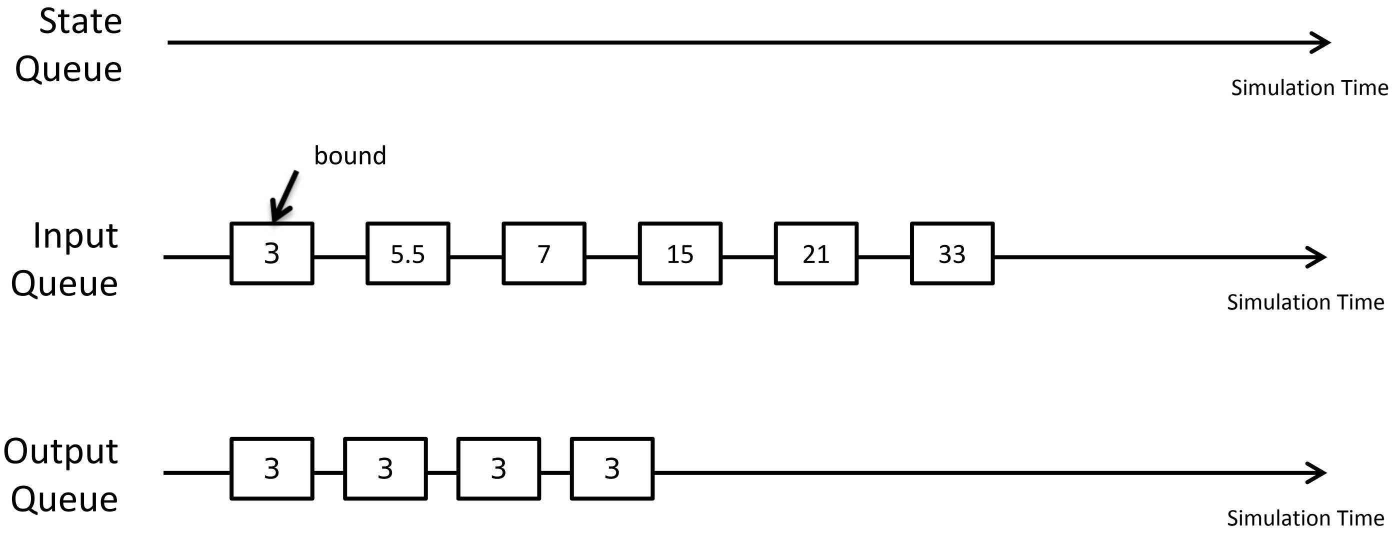
State Saving and Restore



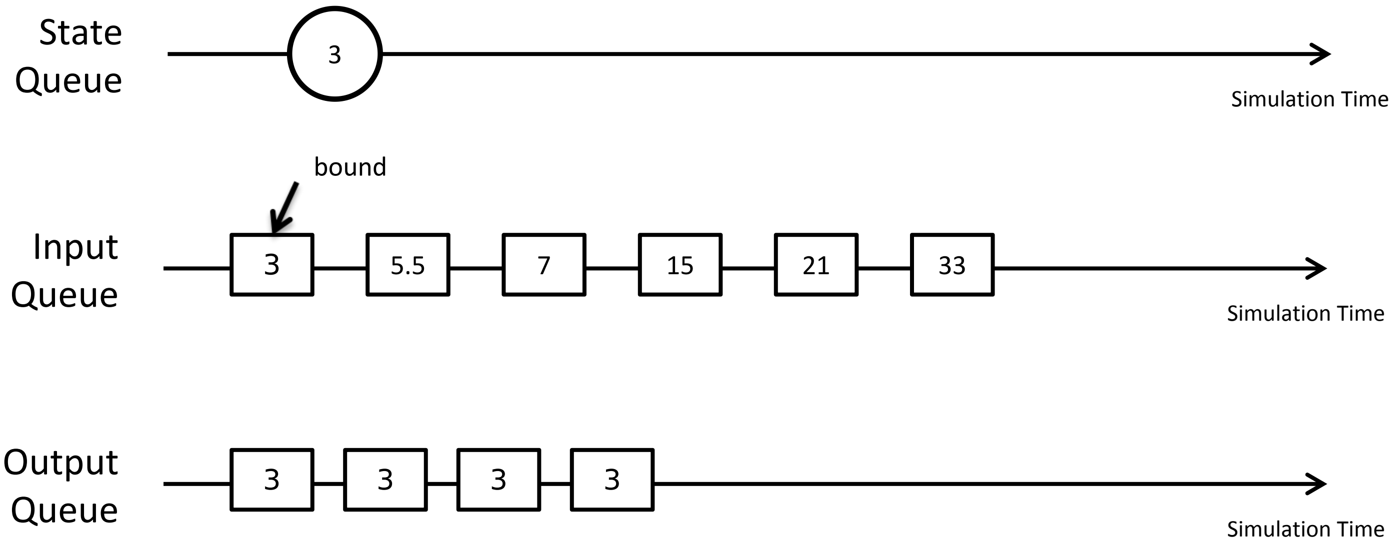
State Saving and Restore



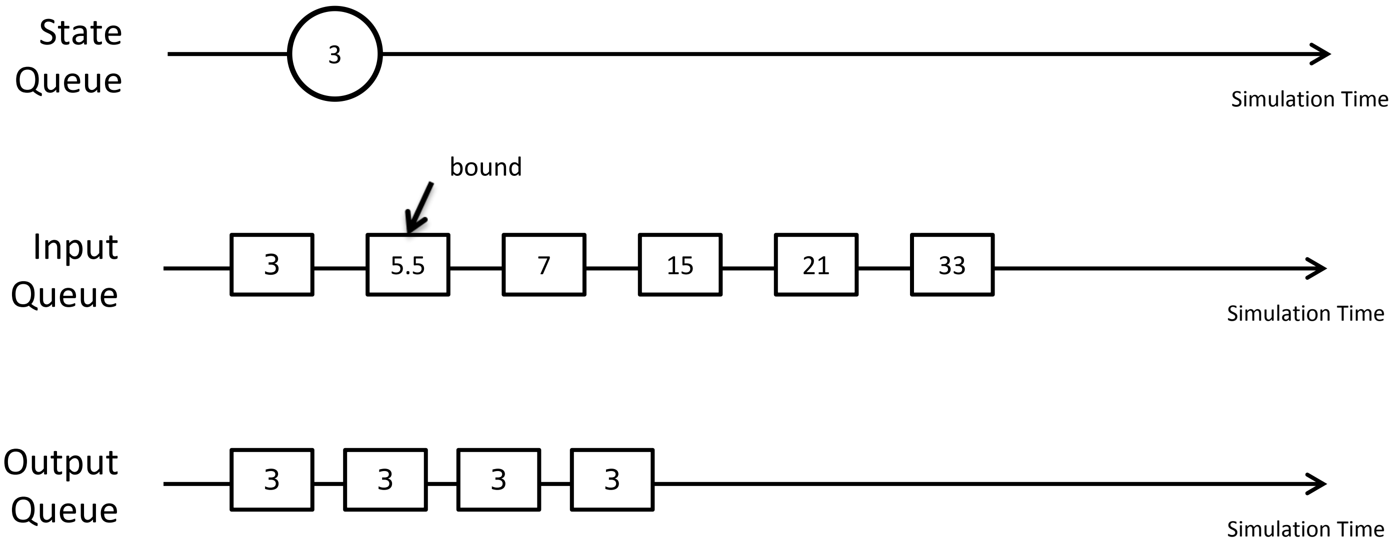
State Saving and Restore



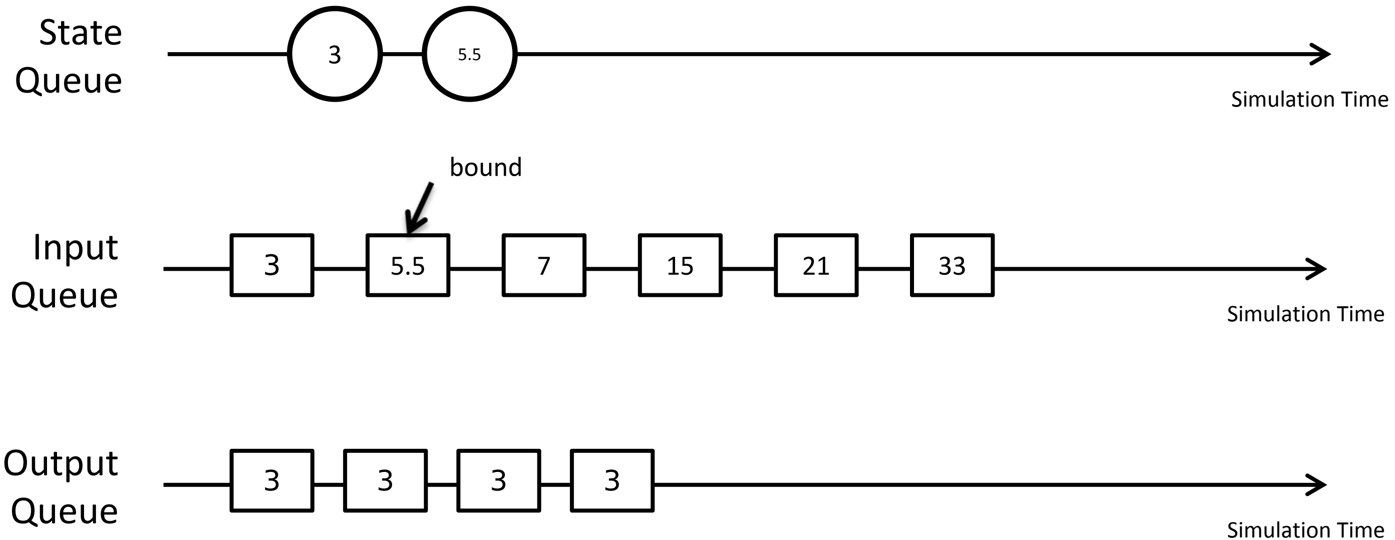
State Saving and Restore



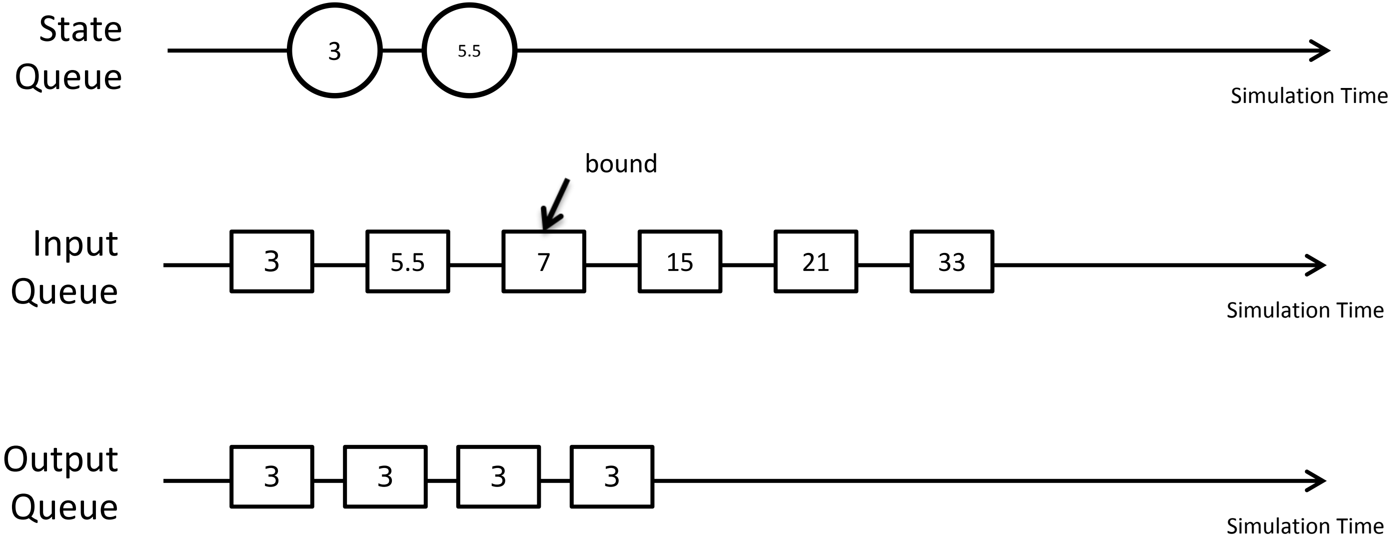
State Saving and Restore



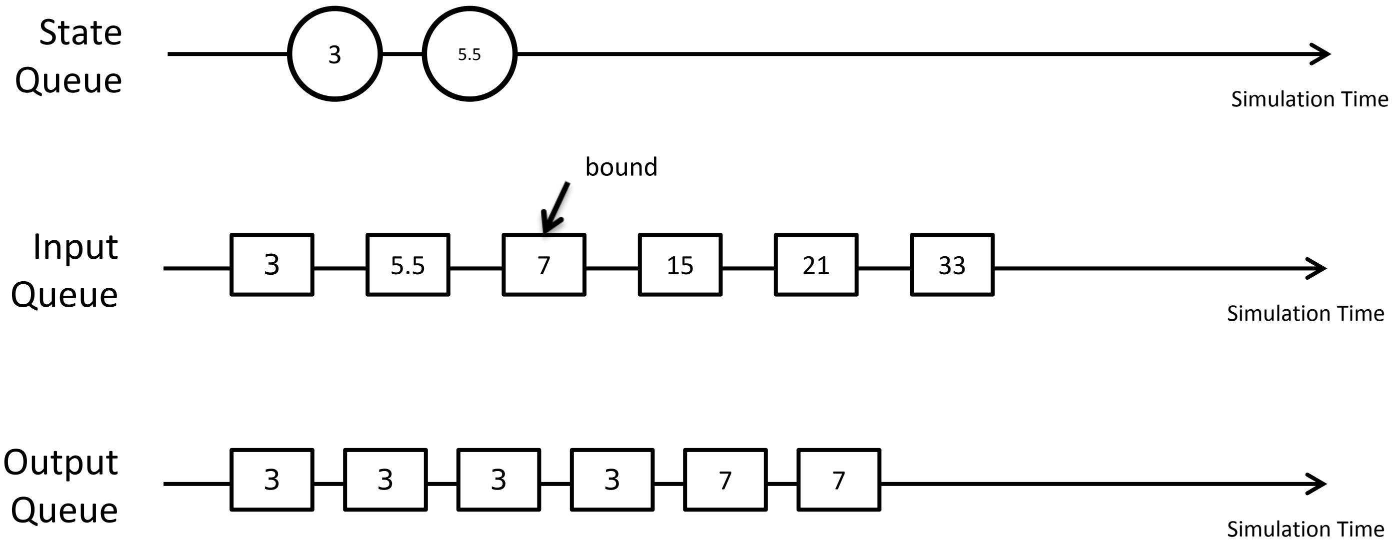
State Saving and Restore



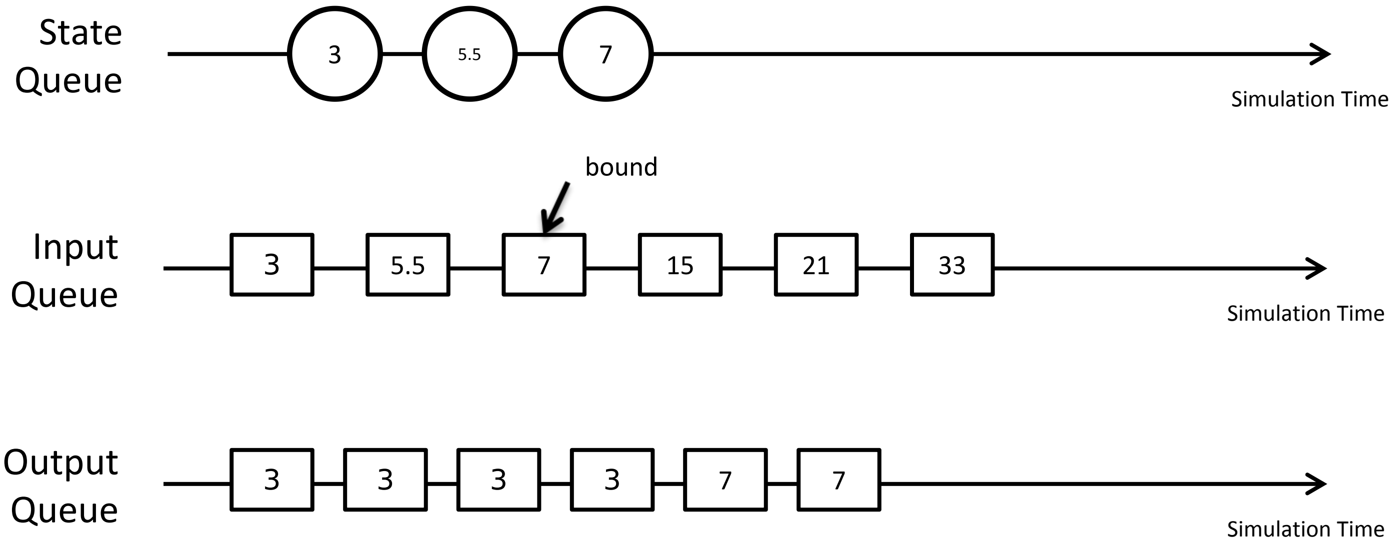
State Saving and Restore



State Saving and Restore

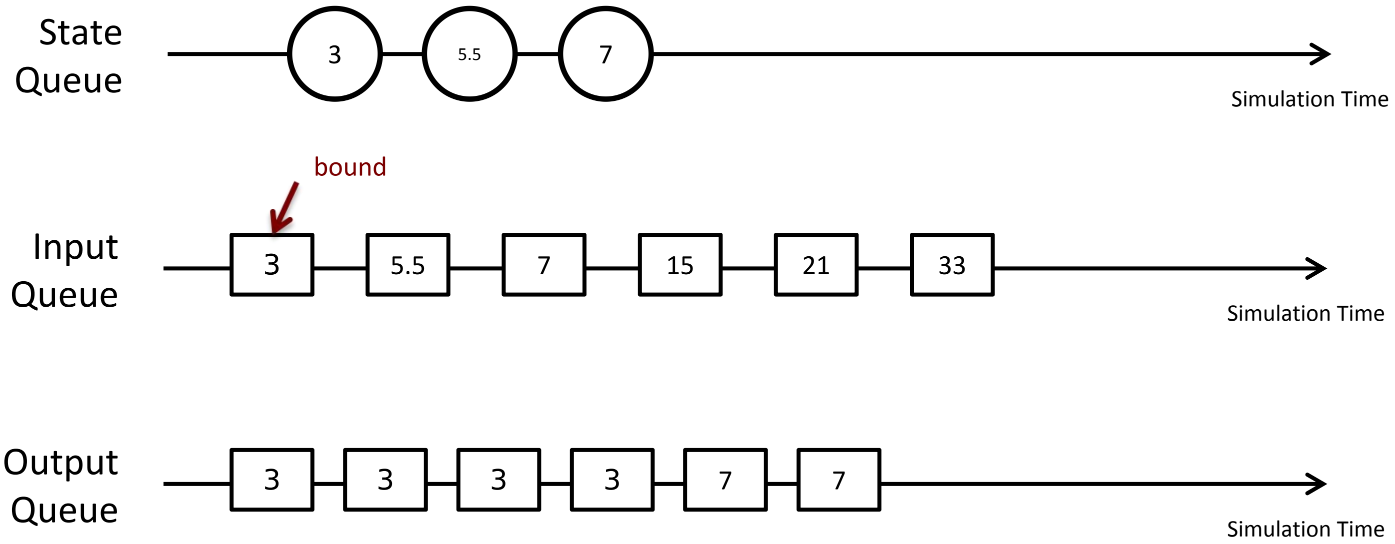


State Saving and Restore

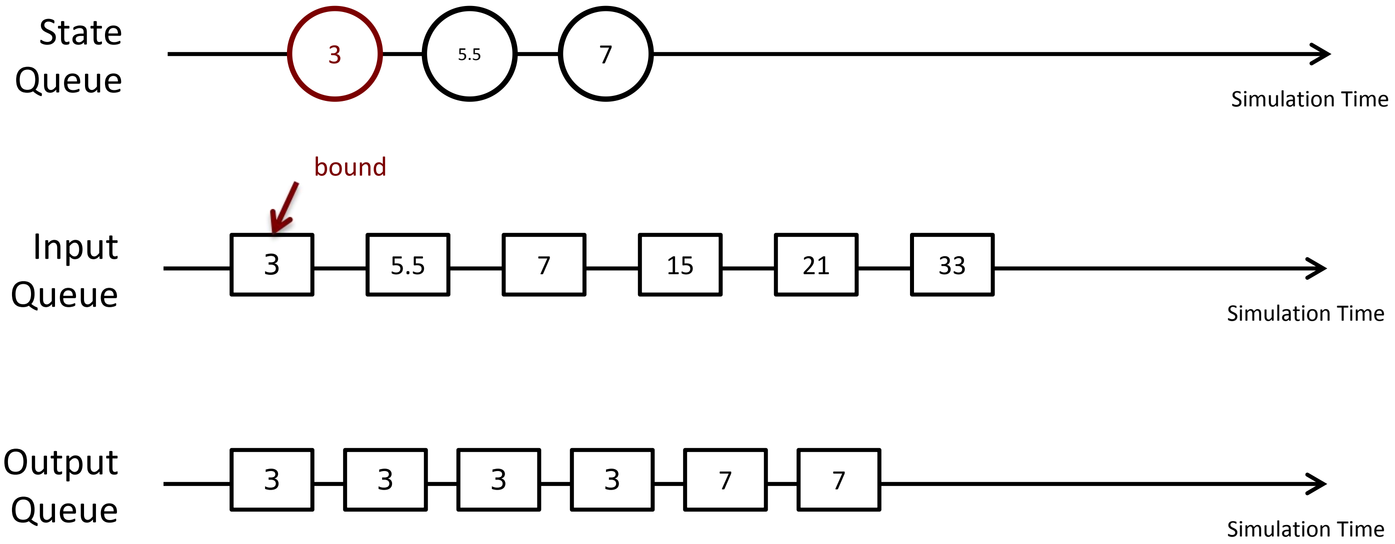


State Saving and Restore

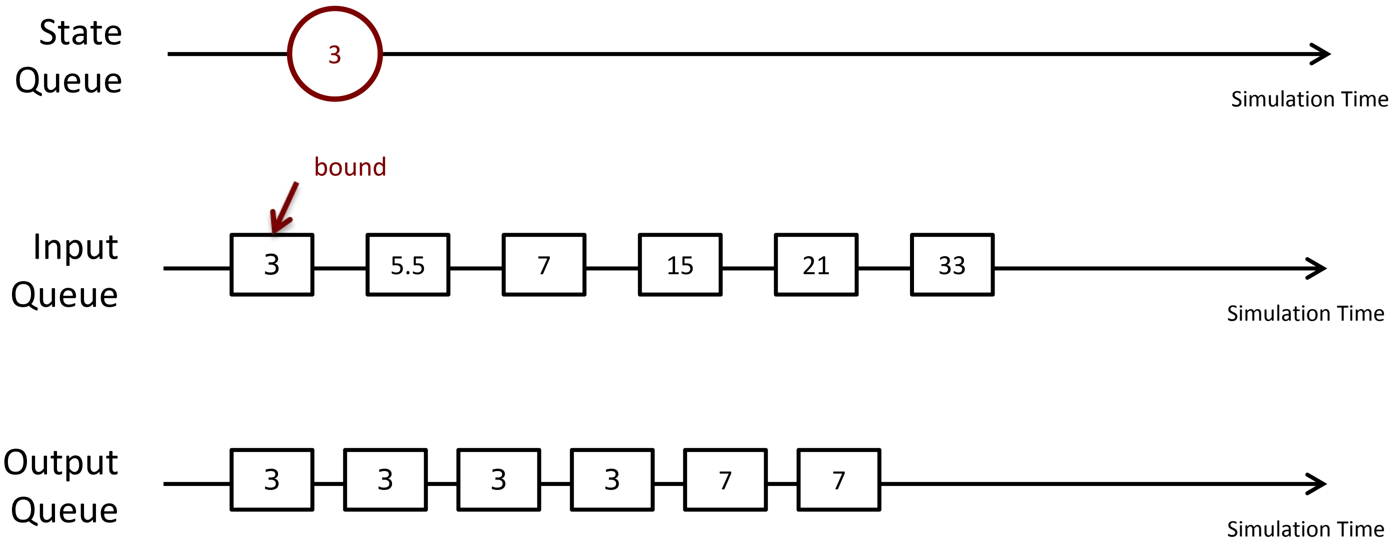
State Saving and Restore



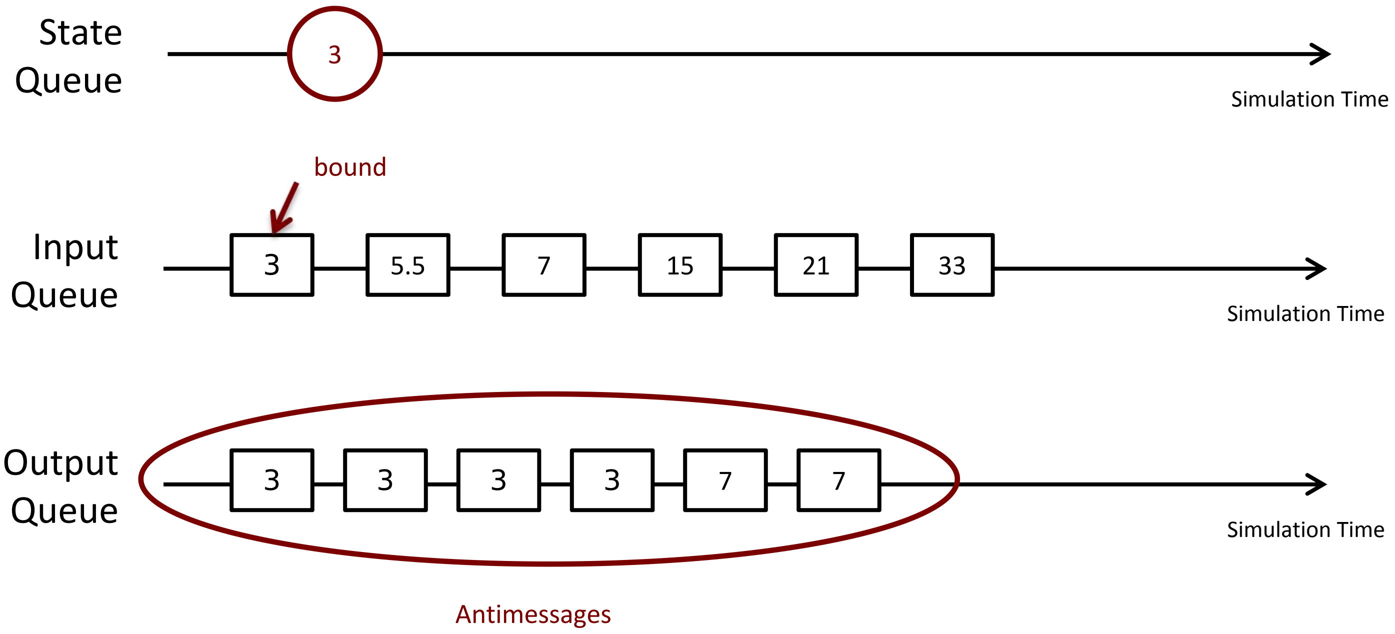
State Saving and Restore



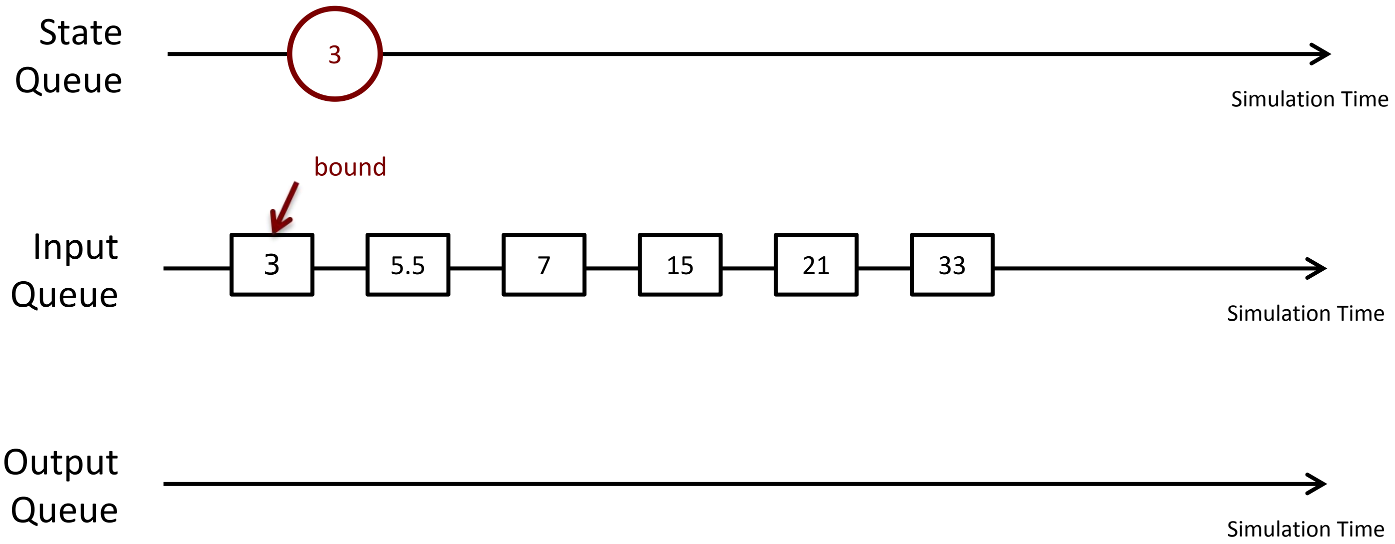
State Saving and Restore



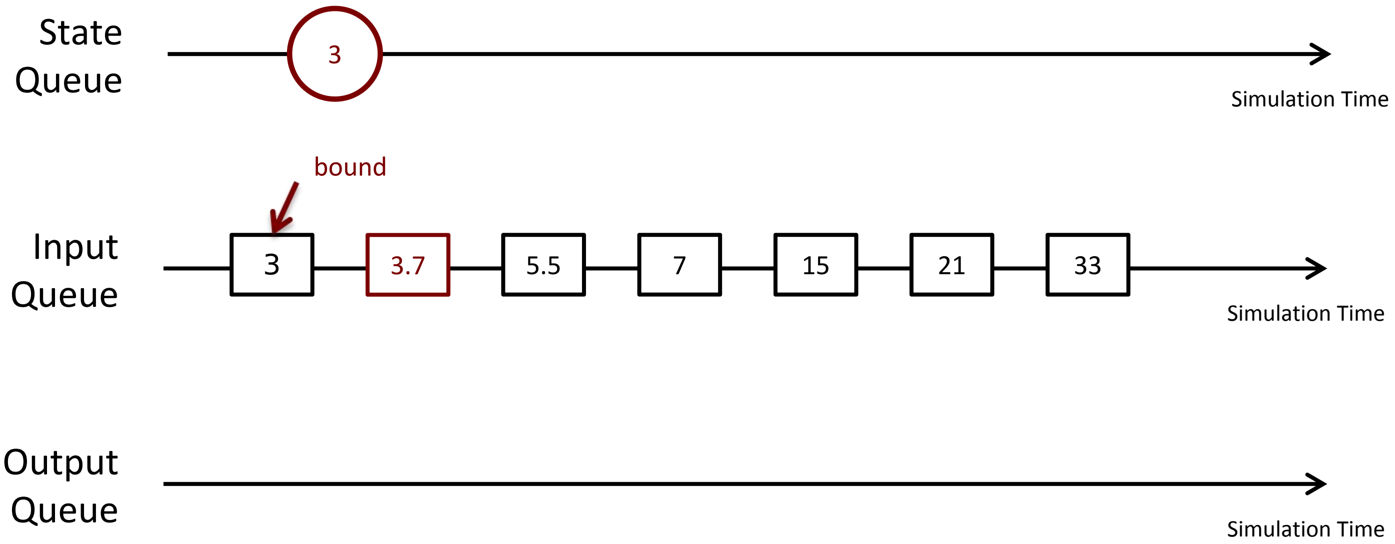
State Saving and Restore



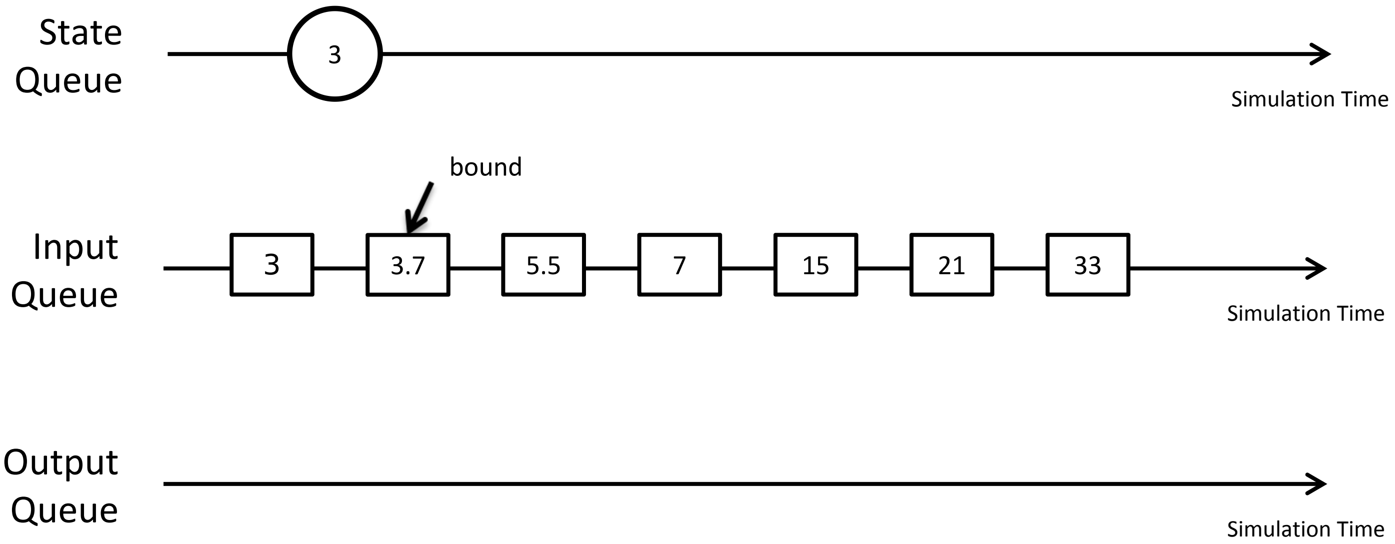
State Saving and Restore



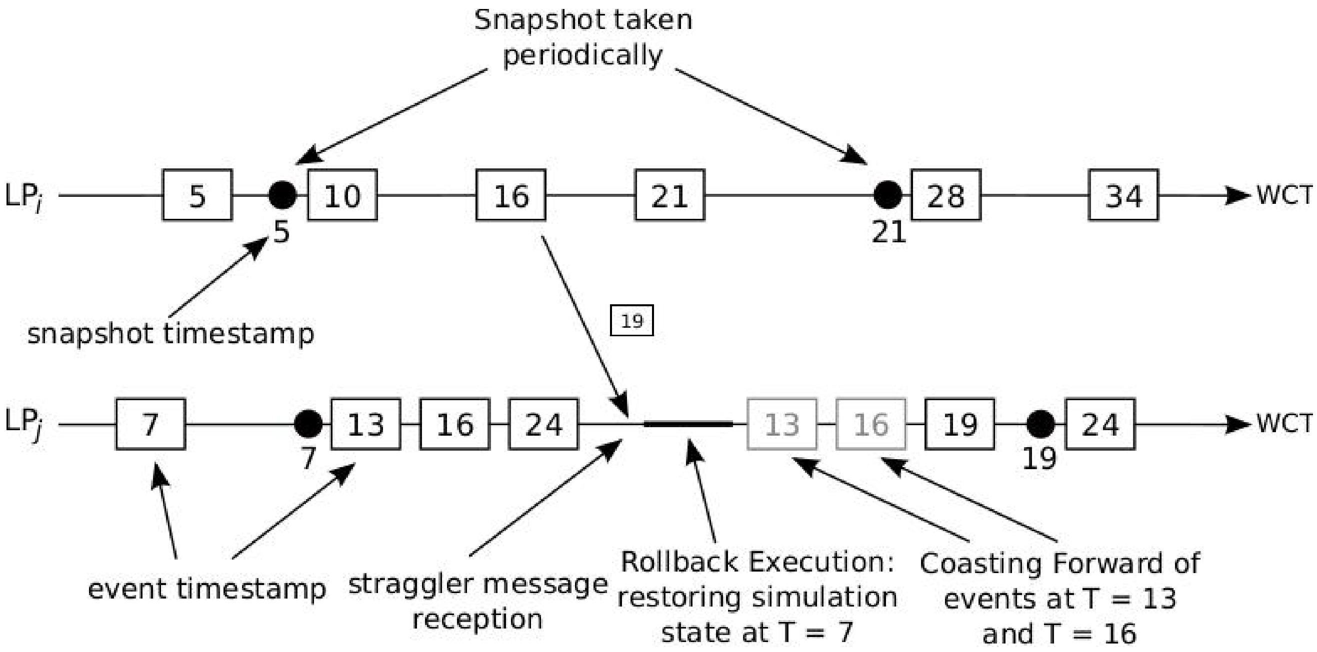
State Saving and Restore



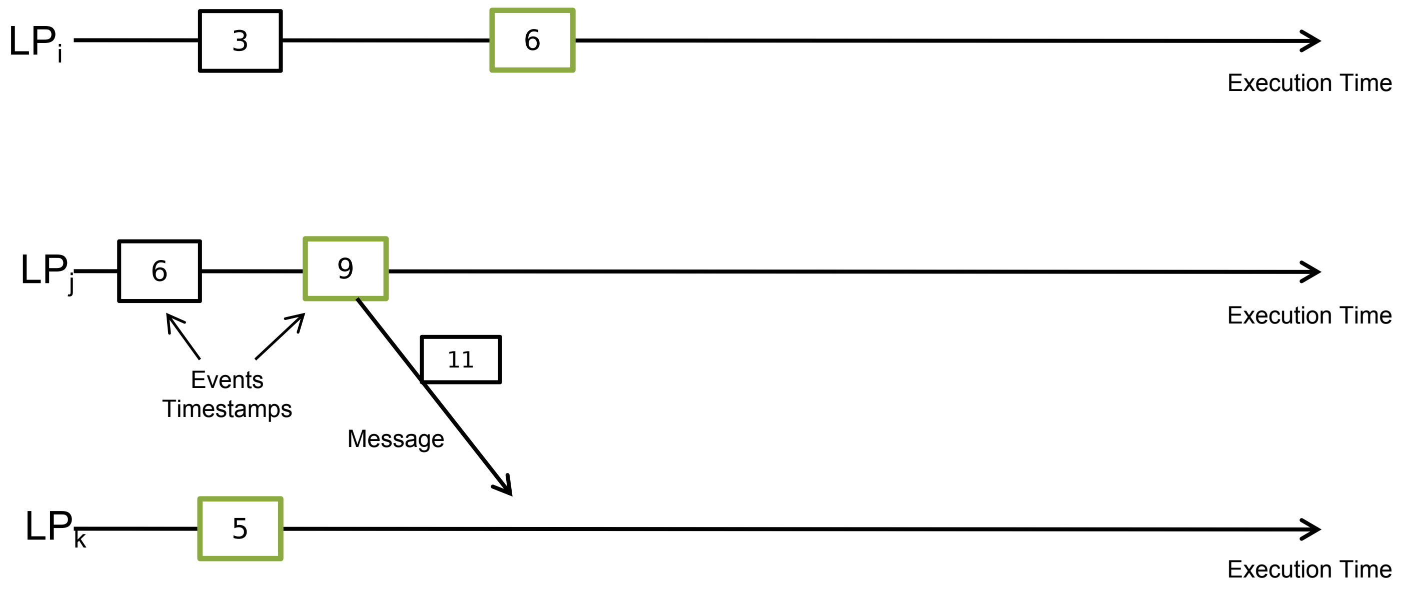
State Saving and Restore



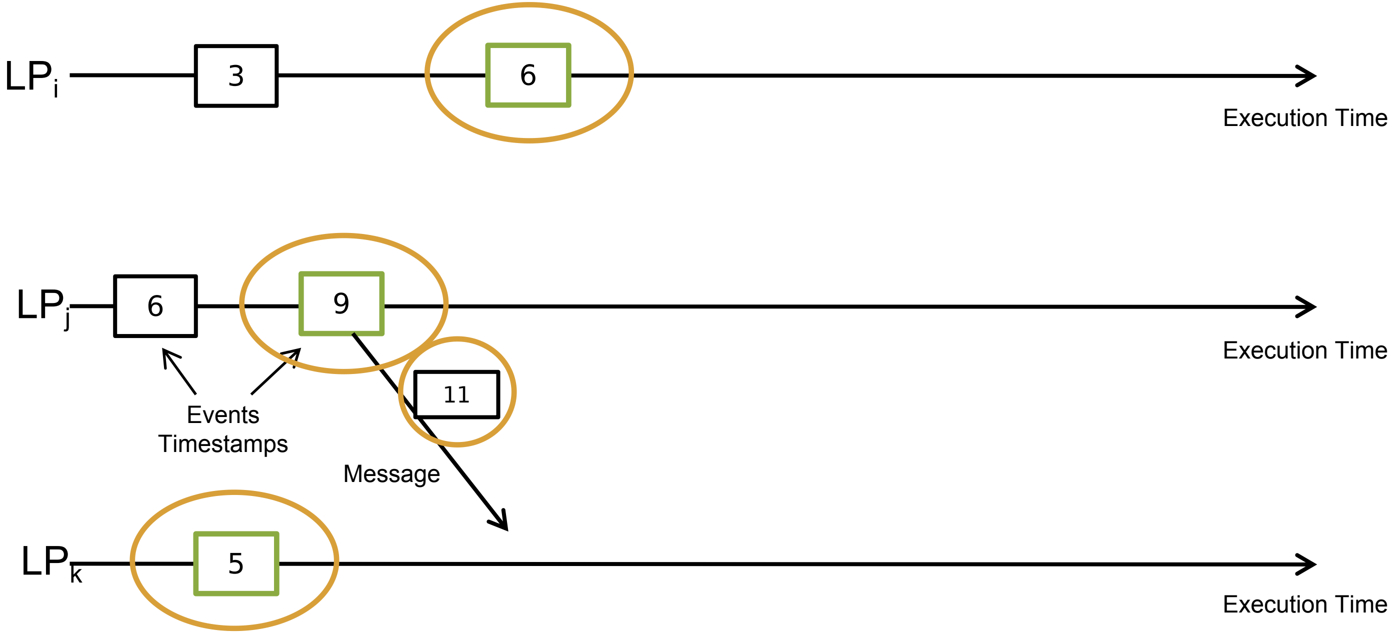
Sparse State Saving (SSS)



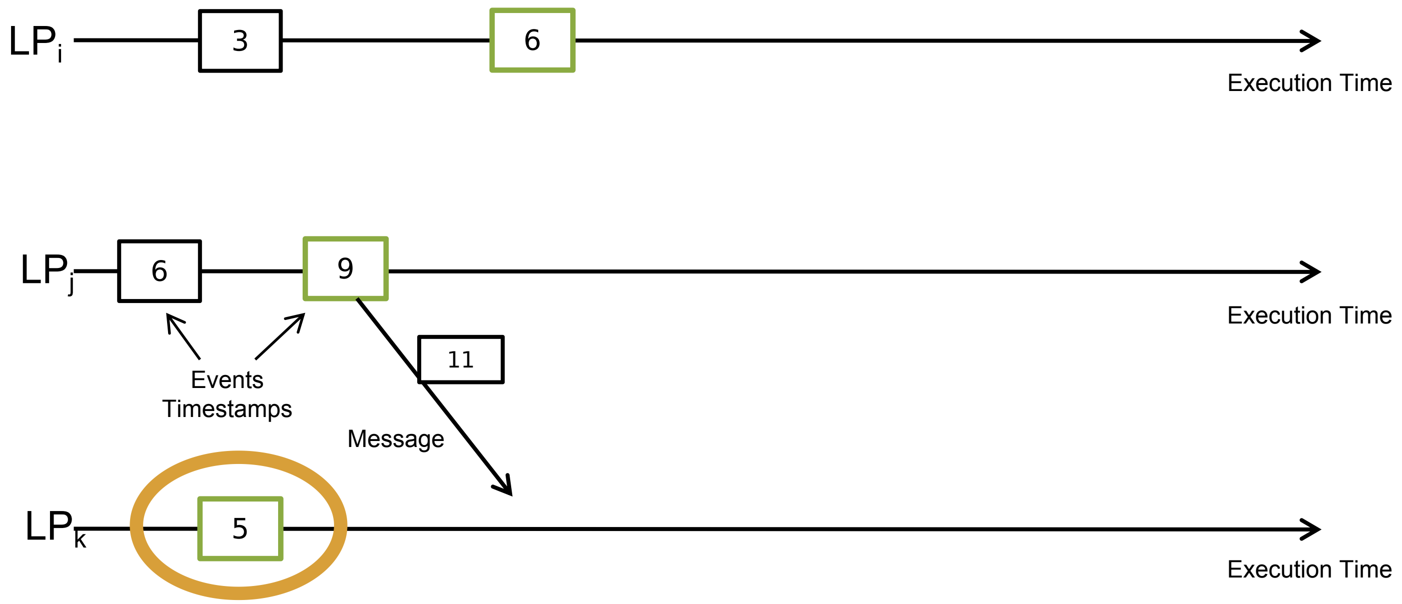
Global Virtual Time



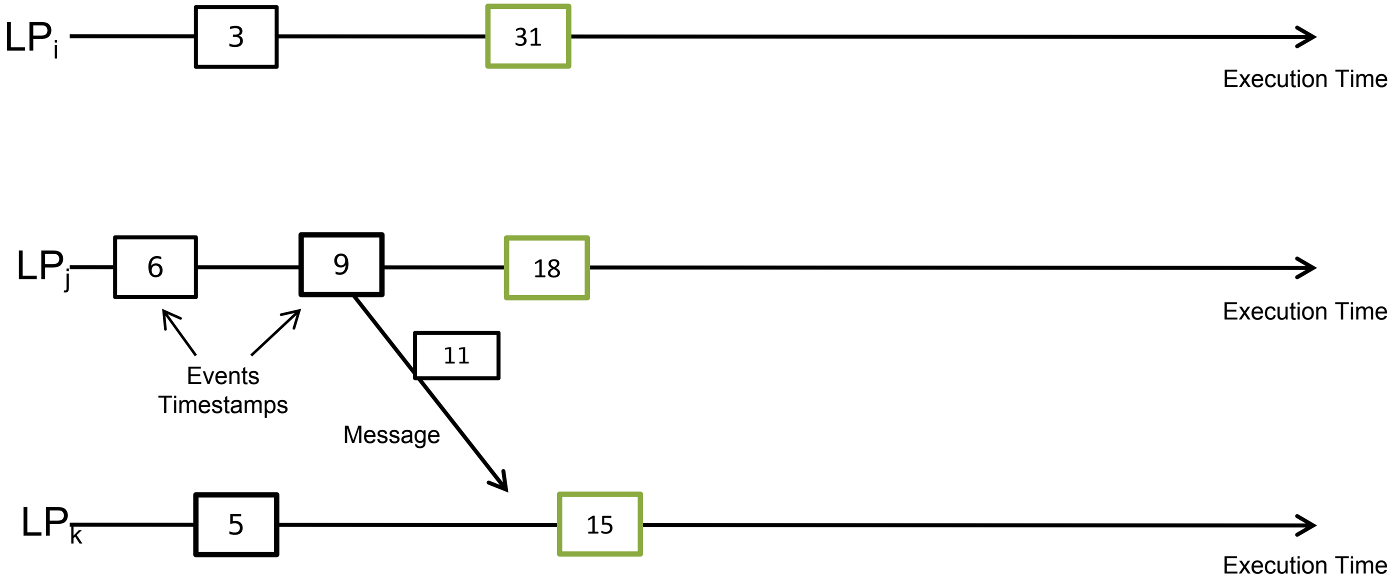
Global Virtual Time



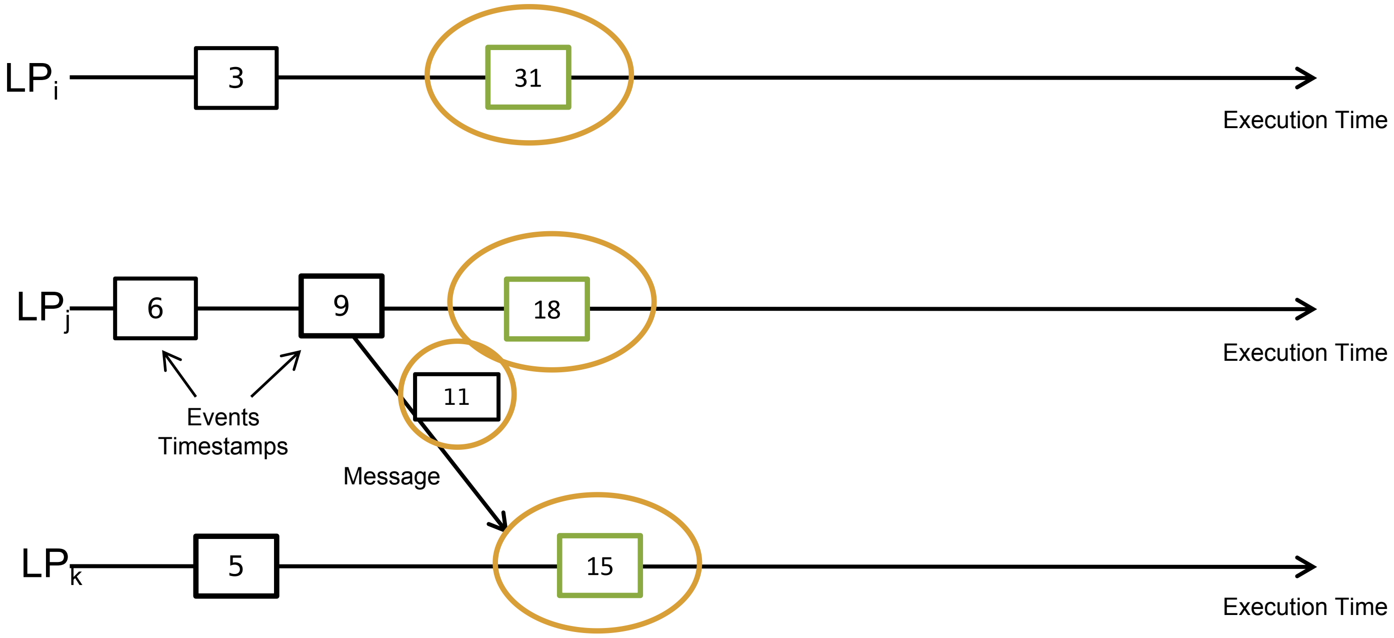
Global Virtual Time



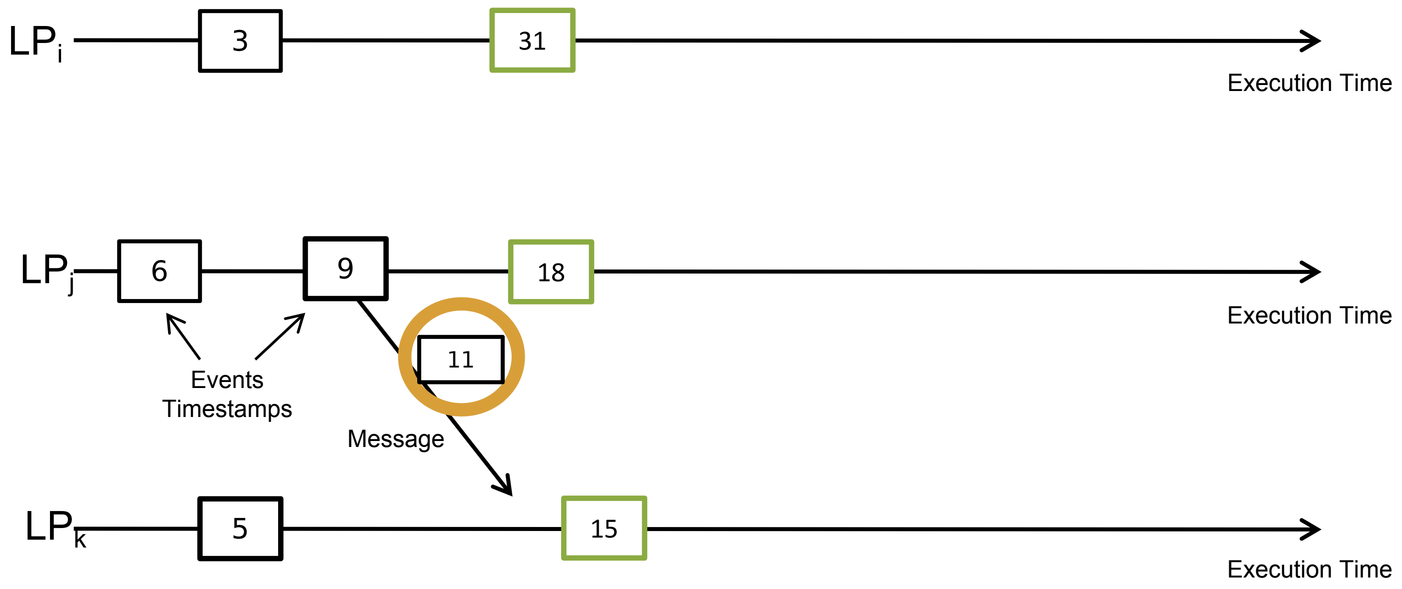
Global Virtual Time



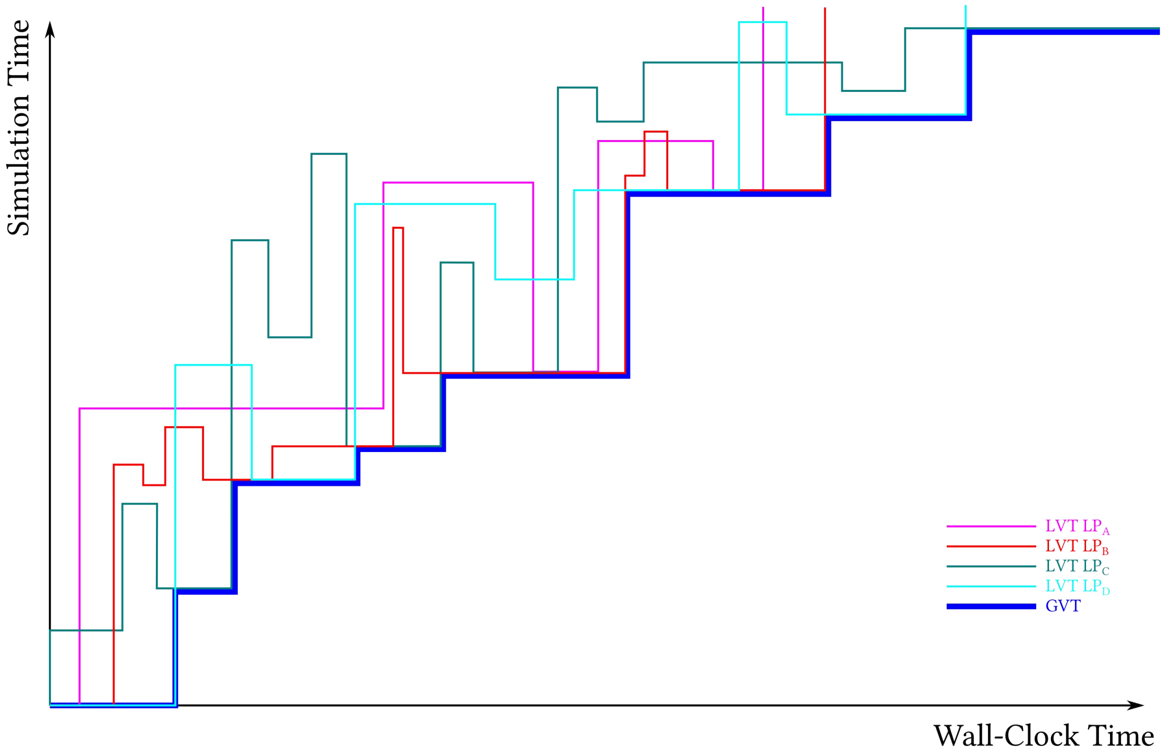
Global Virtual Time



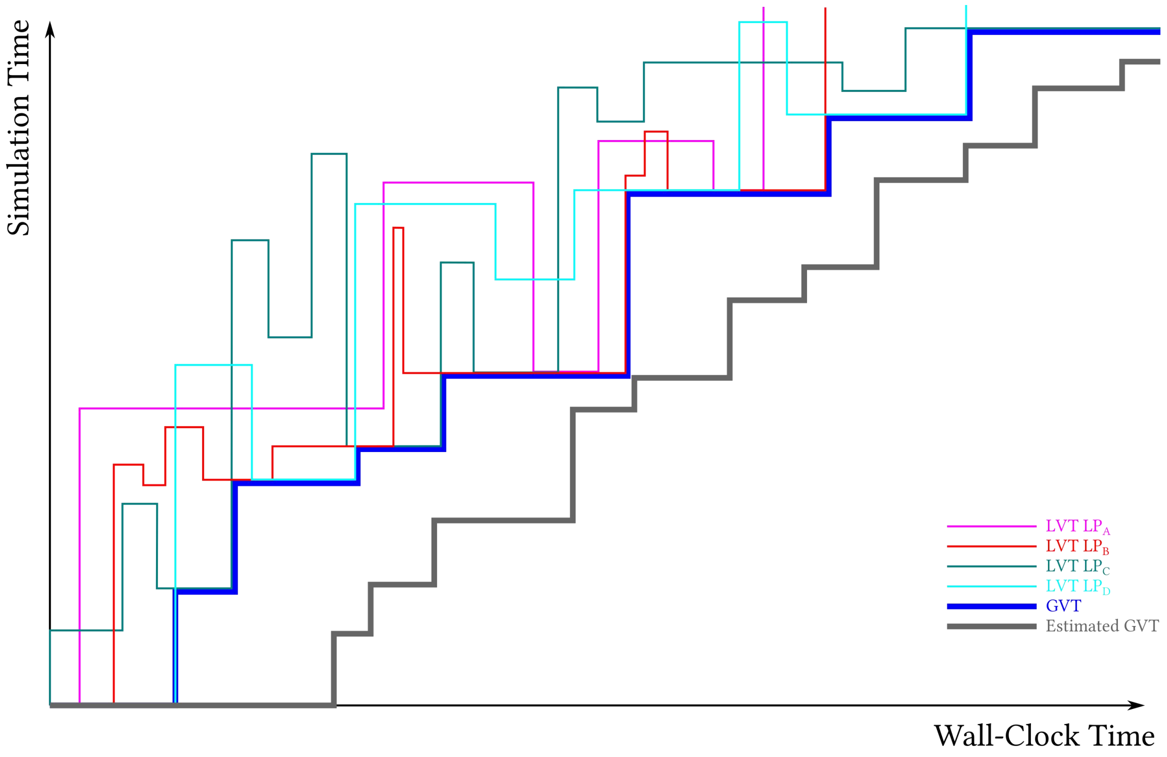
Global Virtual Time



Relations among GVT and LVT



Relations among GVT and LVT

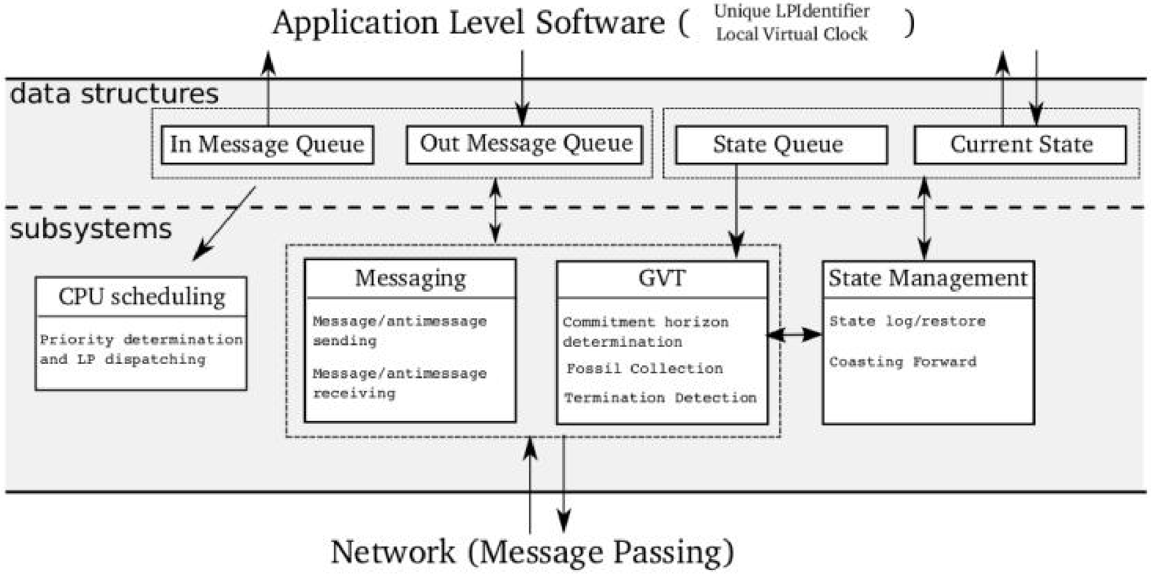


GVT Operations

- Once a correct GVT value is determined we can perform:
  - **Fossil Collection**: the actual garbage collection of old memory buffers
  - **Termination Detection**: check whether GVT $= \infty$ or check a predicate on the simulation state
  - **I/O Commitment**:
    - Irreversible output operations that were postponed (*delay until commit*) can now be executed
    - Input operations before the GVT that have not been “un-put” can be preserved
  - **Runtime error handling**: errors should be trapped in the speculative portion, and the simulation should fail if the corresponding state is committed

- GVT identifies the *commitment horizon* of the speculative execution

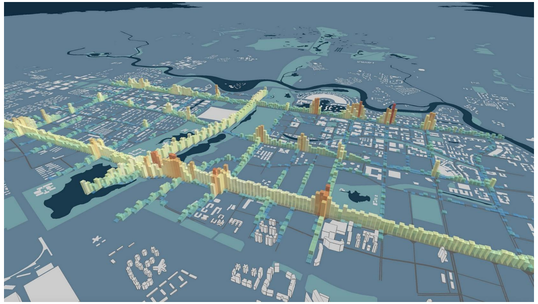
Recap: Time Warp Fundamentals



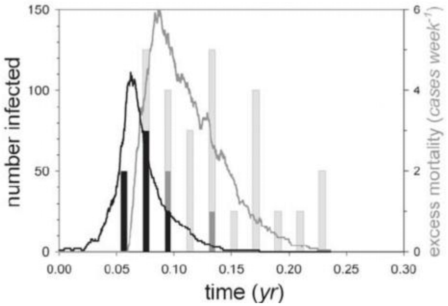
ROOT-Sim

- The ROme OpTimistic Simulator
  https://github.com/ROOT-Sim

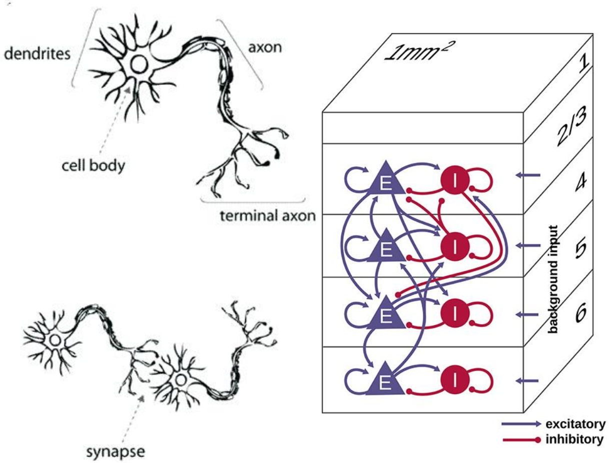
A general-purpose speculative simulation core based on state saving



Traffic Simulation



Epidemics (tuberculosis)



Spiking Neural Networks

Example Session

PCS on ROOT-Sim