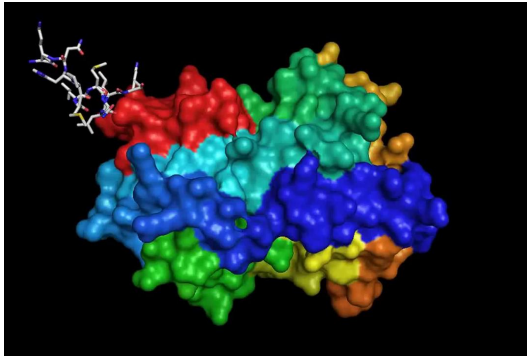
Parallel Discrete Event Simulation

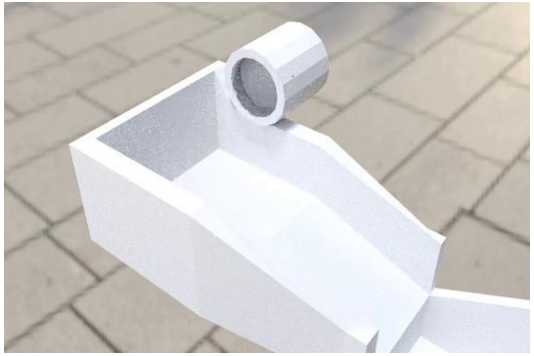
Alessandro Pellegrini
a.pellegrini@ing.uniroma2.it

Simulation: what's that?

"simulare" = imitare qualcosa di finto, ma che nella realtà esiste. E' un termine vago. IL tempo è fondamentale. Inoltre studio come funzionerà qualcosa prima di realizzarlo, come i centri commerciali, simulando ad esempio casi di emergenza. Parliamo di "what if analysis". Le scelte possono essere guidate da tali simulazioni.

- 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

Definiamo "modello" = simulazione di una parte del mondo che ci interessa, piu semplice del mondo stesso, perchè voglio approssimarlo. Anche i modelli possono essere complessi. Il loro obiettivo è farci fare degli studi, richiedono tempo per darci risultati, ci sono tanti fasi, e devo eseguirlo tante volte.

- 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.

"simulazione" = uso il modello creato. Posso gestire variabili, che nel modello reale sarebbero piu complesse.

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

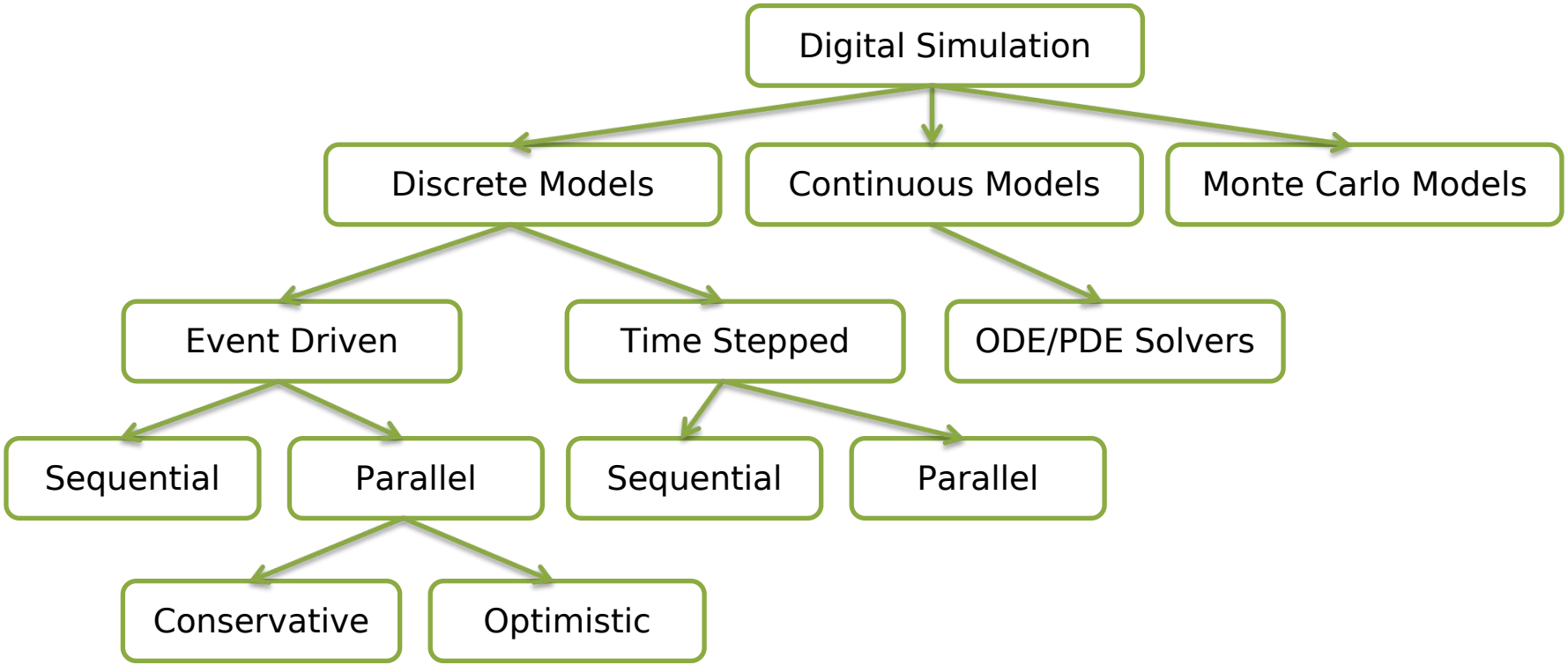
Different Notions of Time

- 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

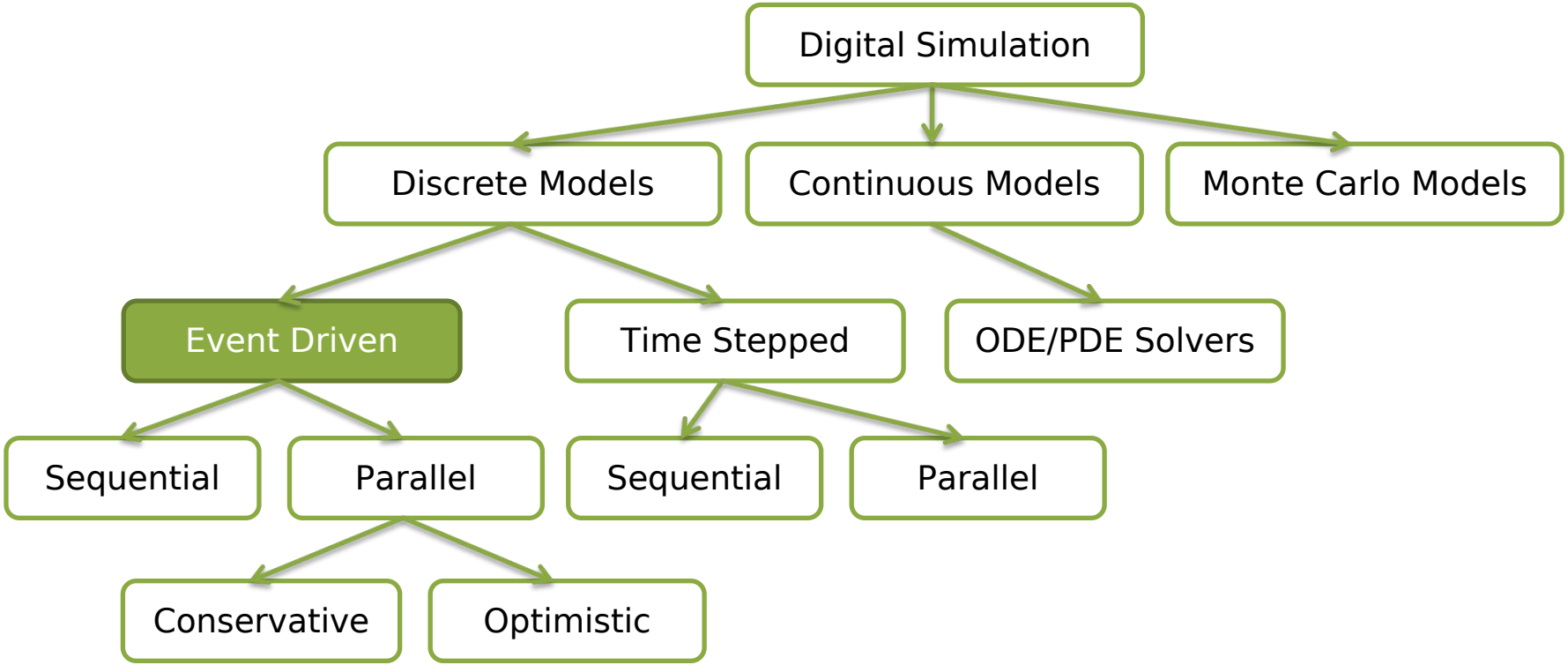
Un obiettivo è ridurre il tempo necessario per ridurre i tempi di simulazione. Il tempo è fondamentale, e nella simulazione vuol dire maneggiare 3 tempi:
- wall clock time: tempo per eseguire modello (da start ai risultati).
- tempo logico: tempo del modello, simulato nel modello. e' un tempo simulato, si muove all'interno del modello, ed è discrepante dal wall clock time (magari 1 sec qui richiede 2 ore del wall clock).
- tempo fisico: il tempo che sistema reale osserva, diverso dal wall clock.

Simulation Taxonomy



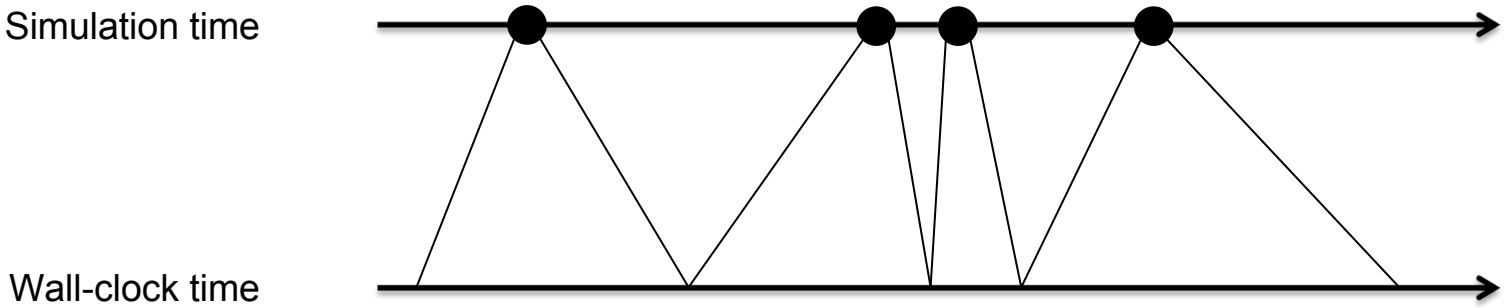
Simulation Taxonomy

event driven: eventi avvengono possibilmente in qualsiasi istante, il modello "salta avanti" di evento in evento. Se sequenziale, estratto un evento genero il successivo, parallela riprende il "sequenziale" ma lo parallelizza con thread o core.



Wall-Clock Time vs Logical Time

confronto logic time vs wall clock
su simulation ogni pallino è un evento rilevante, aperiodici. quando eseguo evento eseguo codice piu o meno complesso, perchè rappresentano cose diverse.
eventi complessi richiedono piu tempo di "wall clock time" rispetto a eventi piu leggeri. noi vogliamo ridurre wall clock time, posto che il modello è quello.



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

"simulazione guidata dagli eventi" appartiene a "programmazione guidata dagli eventi", ovvero il flow di programmazione è guidato da eventi (sensori, messaggi, azioni).
c'è "Main loop = ciclo" in cui sistema si accorge di evento, lo estrae e lo consegna a gestore di eventi (handler), che si attiva. E' come le interrupt nei sistemi hw.

Event Handlers

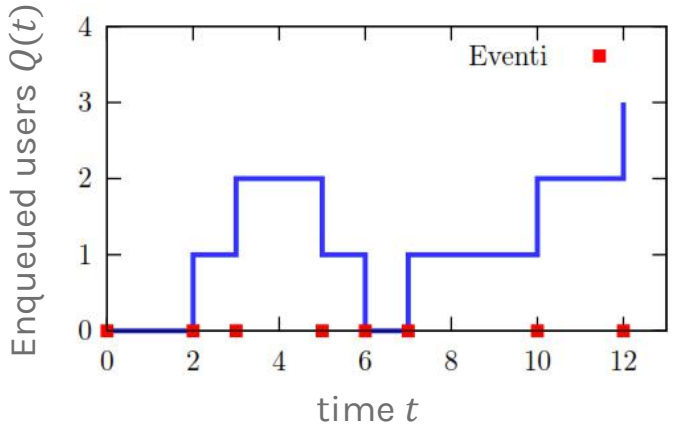
- An event handler is an *asynchronous callback*
- Each event represents a piece of application-level information, delivered from the underlying framework:
  - In a GUI, events can be mouse movements, key pression, action selection, . . .
- Events are processed by an *event dispatcher,* which manages 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

GESTORE DELL'EVENTO
è un call back asincrono, chiamabile a run time quando c'è evento di interesse per quel gestore. I gestori ricevono eventi ed eseguono logica che modifica lo stato.
E' disaccoppiabile rispetto a quello c'è sotto. C'è un dispatcher che vede il tipo dell'evento e lo consegna a chi deve gestirlo, sto disaccoppiando dal main loop. Il numero di eventi è arbitrario. Se gestore occupato? devo accodarlo. La coda è necessaria.

The Queueing Server Example

se mi baso su eventi discreti, come modello qualcosa che ha una durata di tempo logico? (cioè mi arriva un evento, ma come faccio a dire quando finisce, se è un punto discreto?). Serve "inizio" e "fine", ovvero " ricezione" e "completamento", aventi tempi diversi. Ho quindi due classi di eventi.
Q(t) è quante richieste vengono accodate. Graficamente ho dei gradini. Mi dice come si evolve il sistema nel tempo.

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** orologio che rappresenta evoluzione evento simulato. Non continuo, salta da tempo corrente a evento successivo.
  - 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** bufferizzo eventi che non posso processare ora, ma processerò dopo. (pending event set).
  - 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** sottoporzione del mio modello, che interagisce con altri oggetti mediante scambio di eventi.
  - Represent discrete objects in the system being simulated!
  - Also called LPs (Logical Processes)
- **Simulation State** insieme di variabili che descrivono a che punto della mia simulazione mi trovo
  - Represents the physical state of the object being modeled!
- **Random Number Generators** perchè il modello è approssimato, servono per studiare evoluzione modello, che cambio sviluppo in base ai numeri generati (what if analysis).
- **Statistics** sistema raccolta statistiche: devo capire cosa mi dice il sistema.
- **Ending Condition** istante temporale o condizione. mi dice quando fermare simulazione per studiarla.
  Come li metto insieme?

Sequential DES Core Skeleton

```
1:  procedure INIT
2:      End ← false
3:      initialize State, Clock
4:      schedule INIT
5:  end procedure
6:
7:  procedure SIMULATION-LOOP
8:      while End == false do
9:          Clock ← next event's time
10:         process next event
11:         Update Statistics
12:     end while
13: end procedure
```

- inizializzo lo stato, il clock, end<- false (simulazione non terminata), schedulo il primo evento init.
- loop simulazione: finchè non termino, schedulo tempo prossimo evento, "process next event" (la logica è tutta qui dentro), aggiorno statistiche.

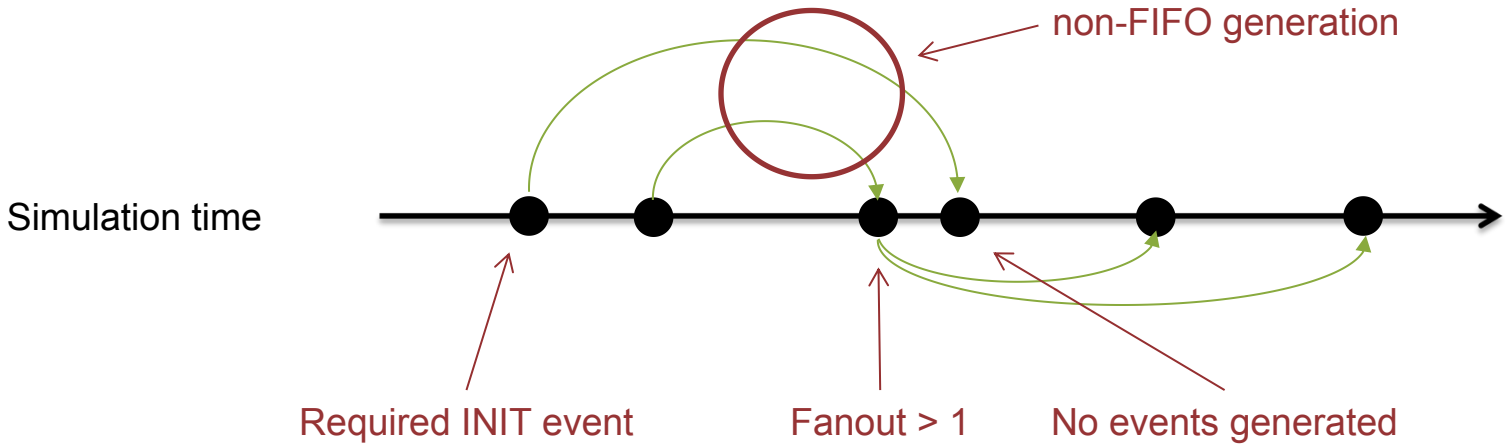
Efficient, scalable, easy-to-use parallelization of this one simple algorithm is what the rest of this lecture is about

Questo è thread singolo, ma se volessi farle andare più veloce?

Events Generation Relationships

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

eventi relazionati, un evento può generare altri eventi, presenti o futuri (non passati). la generazione può essere non fifo (se x1 genera x3, e x2 genera x4, non è detto che x3<x4), non generare nulla, fanout>1 (genera >1 eventi). tutto ciò avviene quando processo un certo evento. con fanout>1 serve event list.



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

gestione eventi fondamentale:
il pending event set è collo bottiglia, ci vanno tutti eventi generati, da cui estraggo evento successivo da eseguire. deve garantire che il timestamp sia ordinato.
voglio anche considerare una coda di priorità.

Data Structures for Simulation: Priority Queue

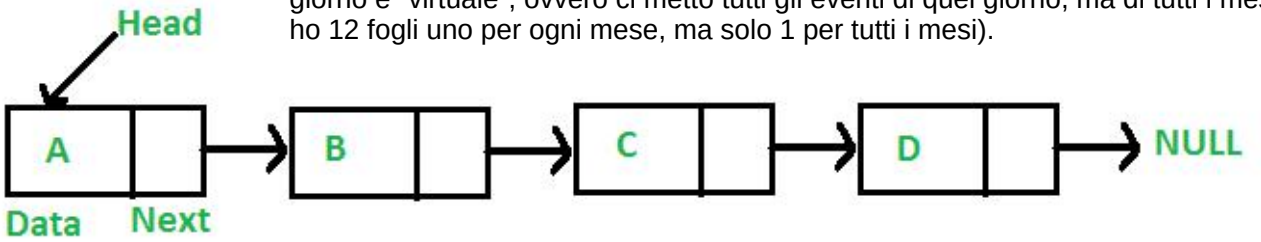
- Is an abstract data type similar to a regular queue
- Elements have a priority associated with each of them
- An element with a high priority is served before
- Operations:
  - insert with priority: add an element to the queue with associated priority
  - pull highest priority element: remove the element 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?

la lista concatenate (linked list) è la struttura più semplice, dalla testa estraggo evento successivo. estrazione costante, inserimento ha costo o(n), non ho garanzie su come vado a generare gli eventi, quindi devo vedere tutta la coda per inserirla.

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

Miglioro con calendar queue
nel calendario ad ogni giorno inserisco eventi ordinati. (concetto di giorno ed evento ordinato). il giorno è "virtuale", ovvero ci metto tutti gli eventi di quel giorno, ma di tutti i mesi dell'anno (non ho 12 fogli uno per ogni mese, ma solo 1 per tutti i mesi).



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

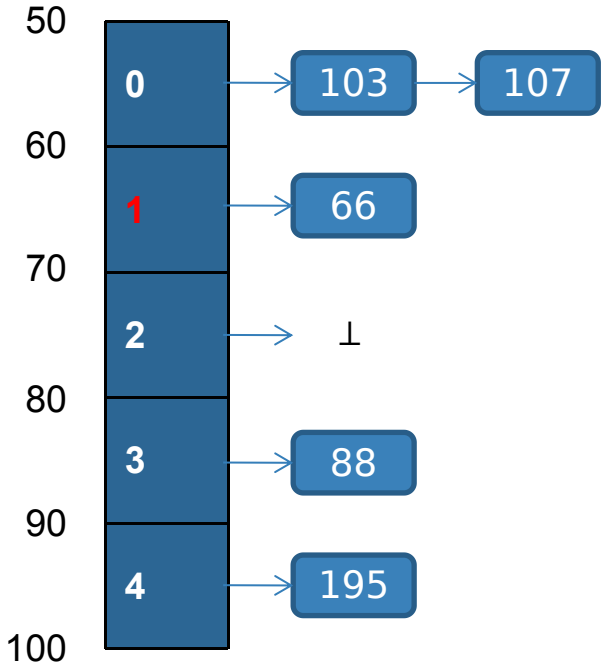
Calendar Queue (Brown 1988)

- 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)



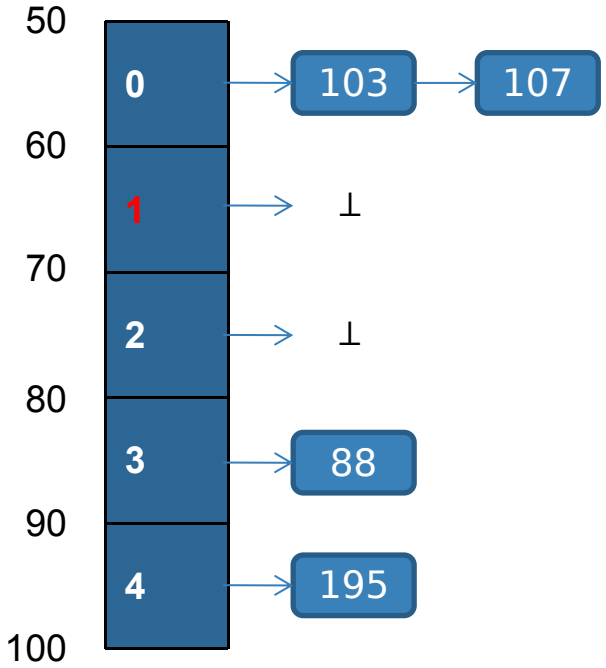
- 5 buckets
- width = 10
- current time = 63

il giorno è detto "bucket", ci metto tutti gli eventi di quel giorno, ogni bucket copre un certo intervallo di tempo, e un evento lo metto nel bucket vedendo il timestamp.
se elemento ha priorità p, lo metto nel bucket |p/copertura temporale di tutti i bucket| mod n

ogni bucket ha >1 evento, e può essere lista ordinata. ho vettore di liste, e ogni bucket contiene sottoinsieme eventi del mio sistema.
per inserimento, capisco subito la lista per contenere evento. come minimizzo? devo tenere liste corte, modifico "w" tale che tutte le liste siano circa lunghe uguali.

quindi w deve essere tale che gli eventi siano equamente distribuite.

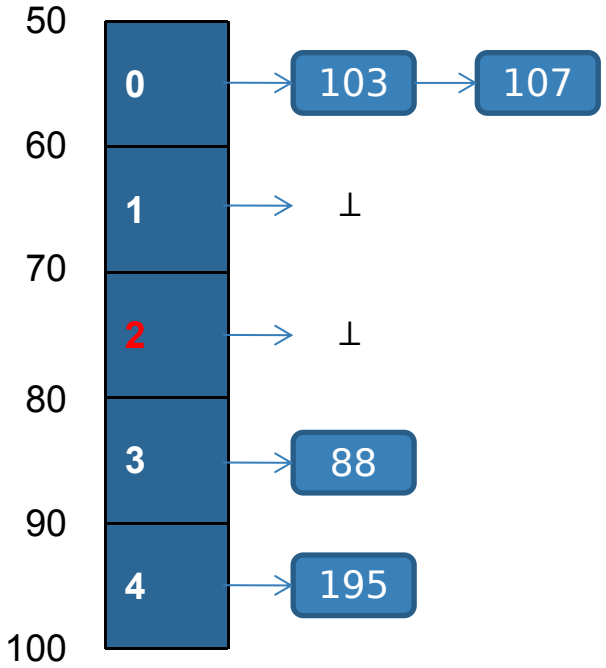
Calendar Queue (Brown 1988)



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

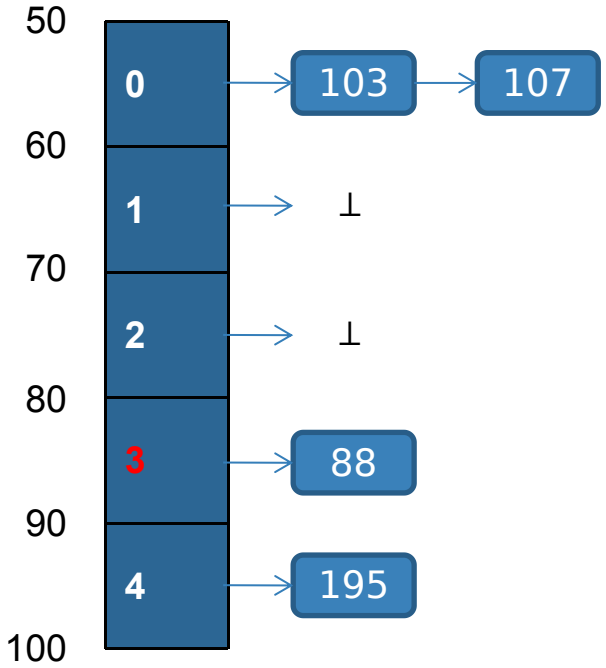
ho 5 bucket, da 0 a 4, tempo corrente è 63. w = 10, ogni bucket copre intervallo di 10 unità di tempo. (bucket 0 da 50 a 60, bucket 1 da 60 a 70).
notiamo che in "0" c'è 103, e non andrebbe bene. nel bucket 1 ho 66 e va bene, posso estrarlo e processarlo.
clock avanza, time = 66, tra 60 e 70 è vuoto, vado a 70-80, vuota, vado a 80-90, c'è evento e va bene, lo processo.
salto a tempo 88, 195 ricade tra 90 e 100? no, dovrei eseguire prima 103 e 107. NON POSSO ESTRARLO.
ritorno all'inizio, AGGIORNO LA COPERTURA (quindi bucket 0 va 100-110), estraggo 103, estraggo 107, e li prendo.
faccio estrazioni in test

Calendar Queue (Brown 1988)



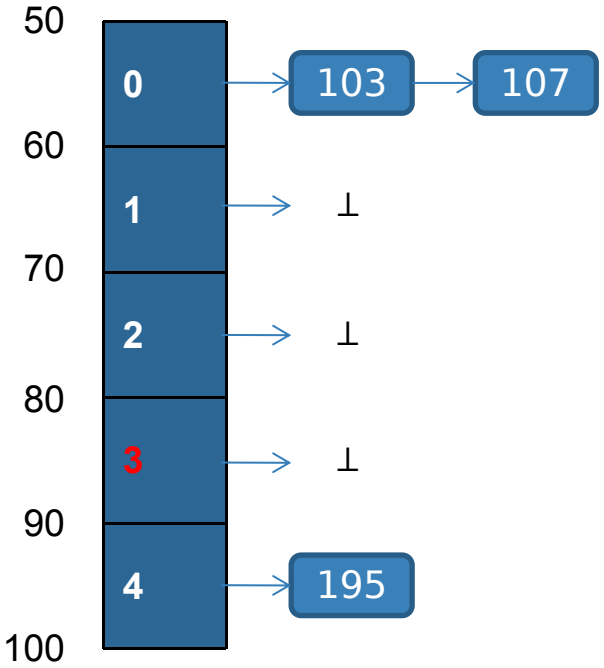
- 5 buckets
- width = 10
- current time = 66

Calendar Queue (Brown 1988)



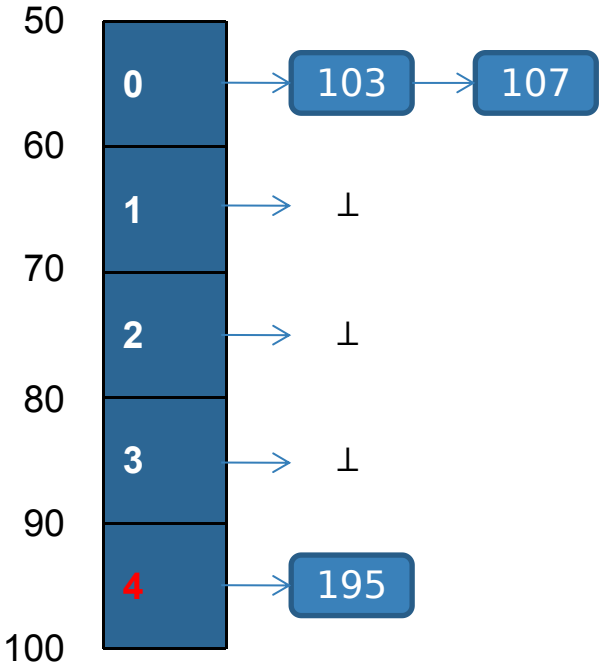
- 5 buckets
- width = 10
- current time = 66

Calendar Queue (Brown 1988)



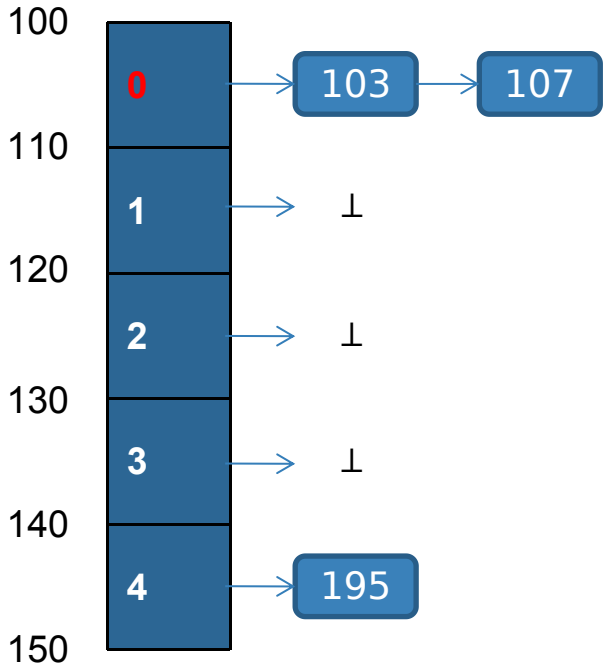
- 5 buckets
- width = 10
- current time = 88

Calendar Queue (Brown 1988)



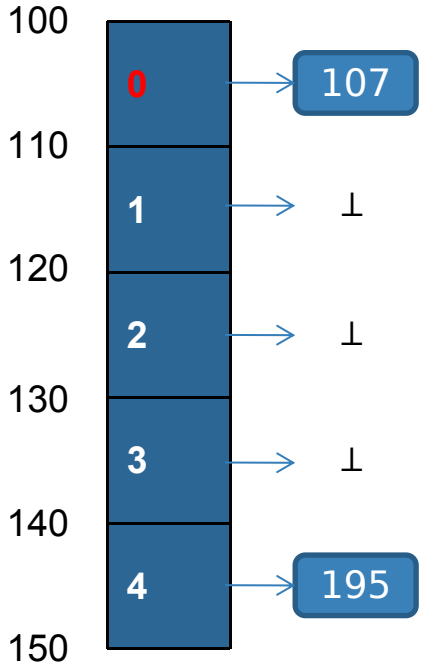
- 5 buckets
- width = 10
- current time = 88

Calendar Queue (Brown 1988)



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

Calendar Queue (Brown 1988)



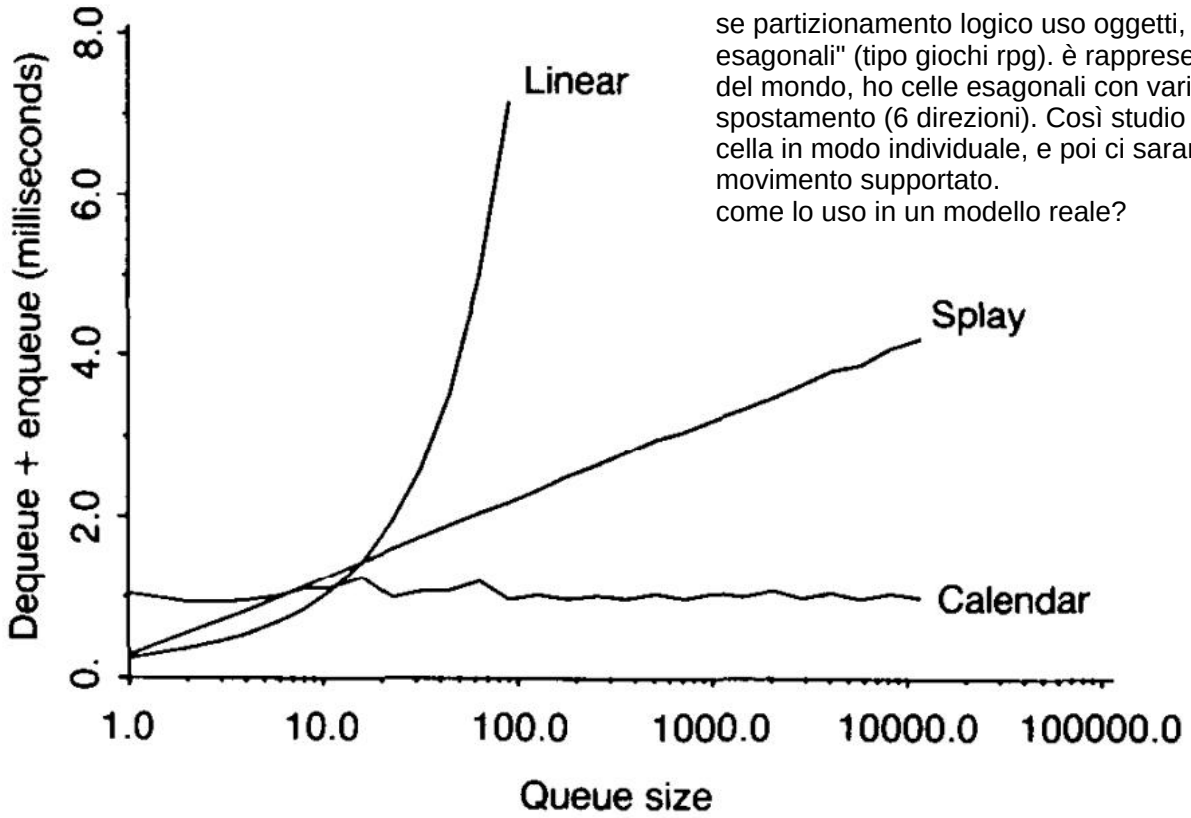
- 5 buckets
- width = 10
- current time = 103

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: $O(1)$



confronto prestazionale, miglioramento netto, ma ancora non sufficiente.
considero ASPETTI SPAZIALI

se partizionamento logico uso oggetti, se fisico uso "mondi esagonali" (tipo giochi rpg). è rappresentazione approssimata del mondo, ho celle esagonali con varie direzioni di spostamento (6 direzioni). Così studio cosa accade in ogni cella in modo individuale, e poi ci saranno interazioni. Riduco movimento supportato.
come lo uso in un modello reale?

in una coda, inserisco un certo numero di nodi, 30k, con priorità casuale, faccio inserimenti.
//retrieve elemetns from calendar queue
li estraggo, studio costo inserimento ed estrazione.
con coda lineare: 3.5 secondi per inserimento + estrazione in ordine
con calendar queue: mezzo secondo. (65k bucket, w si ridimensiona dinamicamente, è 3 volte la distanza media tra eventi).

Example Session

Performance of Future Event Sets

Space Partitioning: Hexagonal Worlds

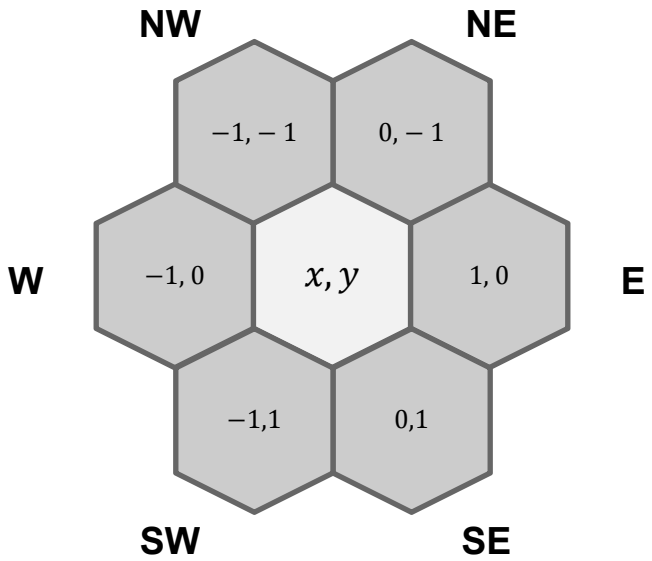
- The namespace of LPs which we use is the interval $[0, \infty)$
- A hexagonal world follows a different namespace
- To handle the topology, we must perform a *linear to hex* translation

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

$$x = LPid \bmod edge$$

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

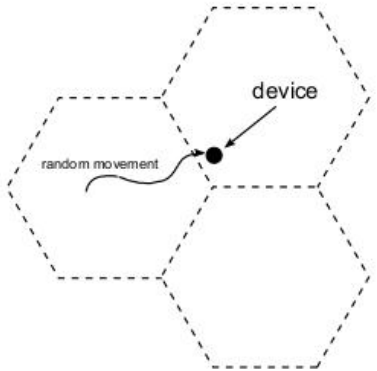
Hexagonal World: Neighbors




```
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

Rete PCS usata per studiare RETI 3G, per vedere QoS. la cella esagonale garantisce copertura esagonale (come la cella vera e propria).
persone si spostano casualmente nella mappa, e telefonano. Se passo da cella ad altra c'è handoff (stazioni si mettono d'accordo per trasferire chiamata). Se canale successivo occupato? casca la chiamata.
Per ciascun canale mantengo struttura dati con tutte le info necessarie.
Tante chiamate -> tanti record. Tali record li organizzo in lista.

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)

posso fare di meglio? parallelizzo
abbiamo visto componente temporale e spaziale.
voglio massimizzare speed-up, non massimizzando l'efficienza delle risorse.
voglio essere il più veloce possibile.

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

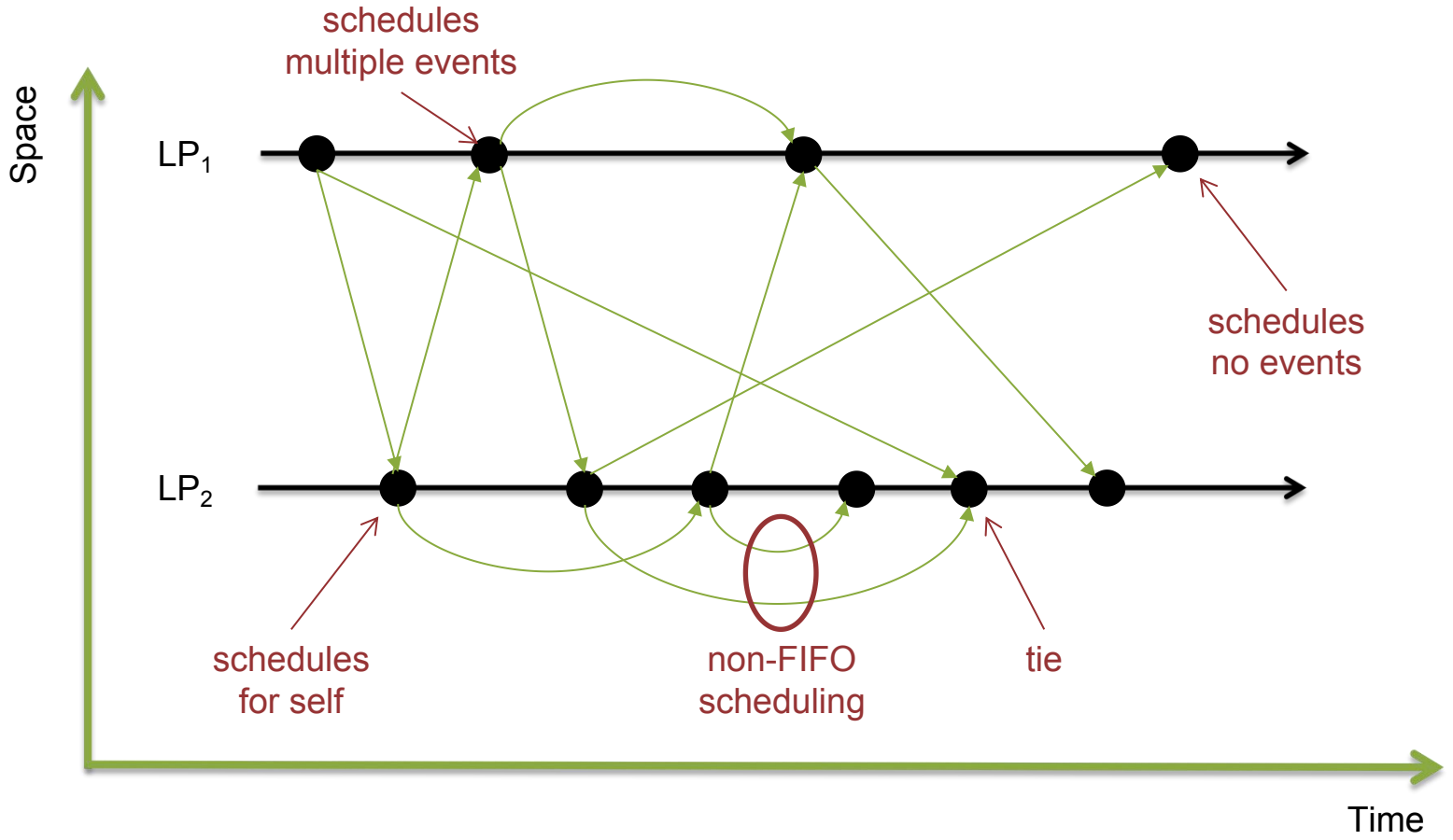
- The DES model is partitioned into N different entities, each one representing a portion of the whole simulated system
- The different entities have been historically named Logical Processes (LPs)
- The evolution of the state of each individual LP mimics the evolution of the corresponding sub-portion 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



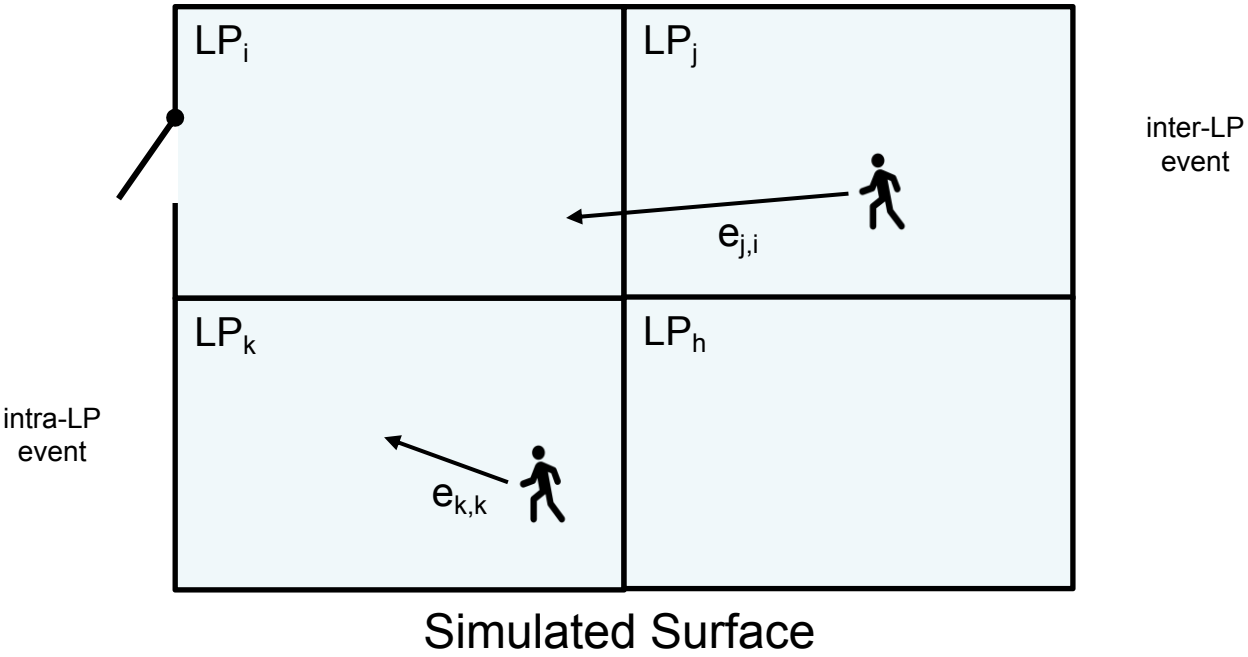
The Synchronization Problem

- 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?

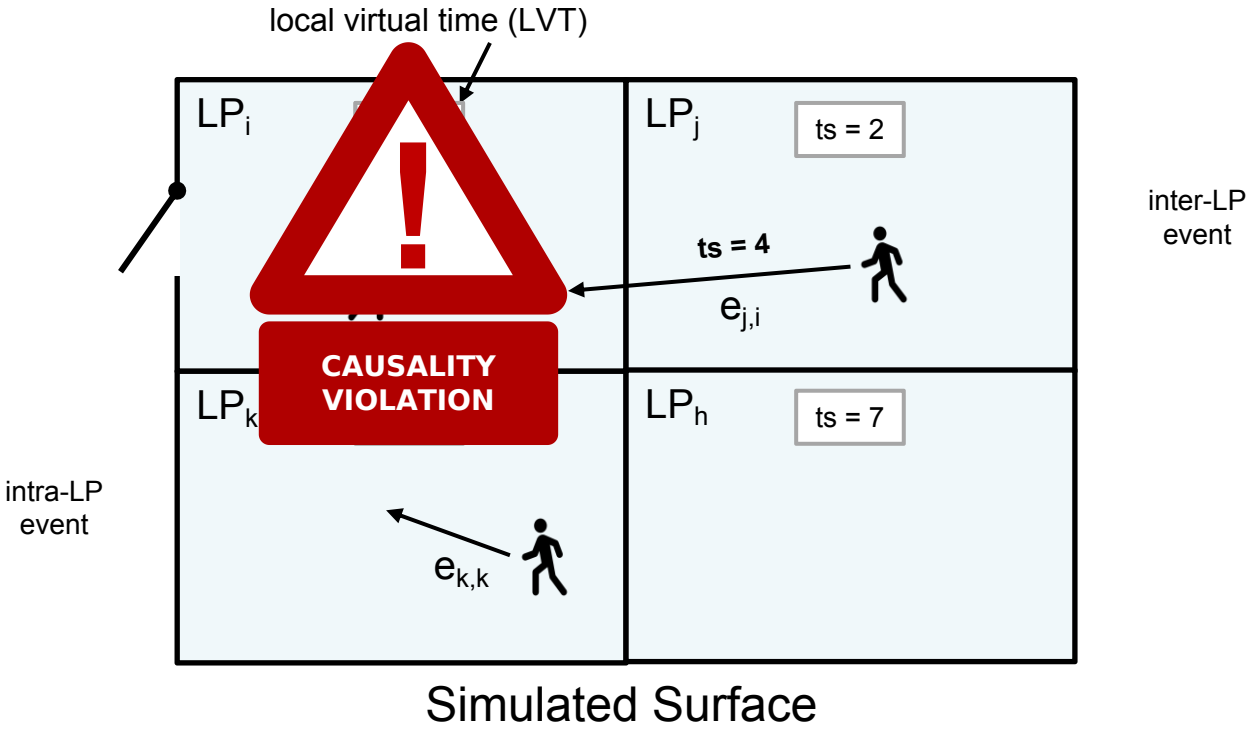
sfrutto divisione spaziale per partizionare stato simulazione, non ho un unico stato, ogni oggetto di simulazione ha stato privato disgiunto. unendoli tutti ho unico stato. Mi muovo verso verso rappresentazione spazio-temporale, ogni oggetto ha timeline indipendente. coda degli eventi è stata suddivisa. Oggetto1 estrae ciò che vede nella sua coda, e può mandare eventi verso altri. Con più code, come capisco l'evento successivo? sincronizzarli è complesso computazionalmente.
se li simulo ognuno per conto proprio? perdo la correttezza, potrebbe essere tutto a caso. (ogni oggetto ha propria visione del tempo). Ho violato la causalità.
Se un evento arriva dopo, ma era fissato prima, la soluzione è fare rollback per aggiustare la causalità. Ciò avviene in caso di messaggi STRAGGLER (un msg ha timestamp = 8, io sto a ts = 15, devo tornare indietro). Però se io ho fatto cose nel mentre? mando antimessaggio, ovvero informo altre code che devono fare pure loro rollback (usando i log). Tale approccio è detto OTTIMISTICO. finchè non c'è inversione di priorità posso estrarre localmente miei eventi ed andare avanti, se va male rimetto tutto a posto. come? con due code azionali: coda stati e coda msg mandati in output.

The Synchronization Problem

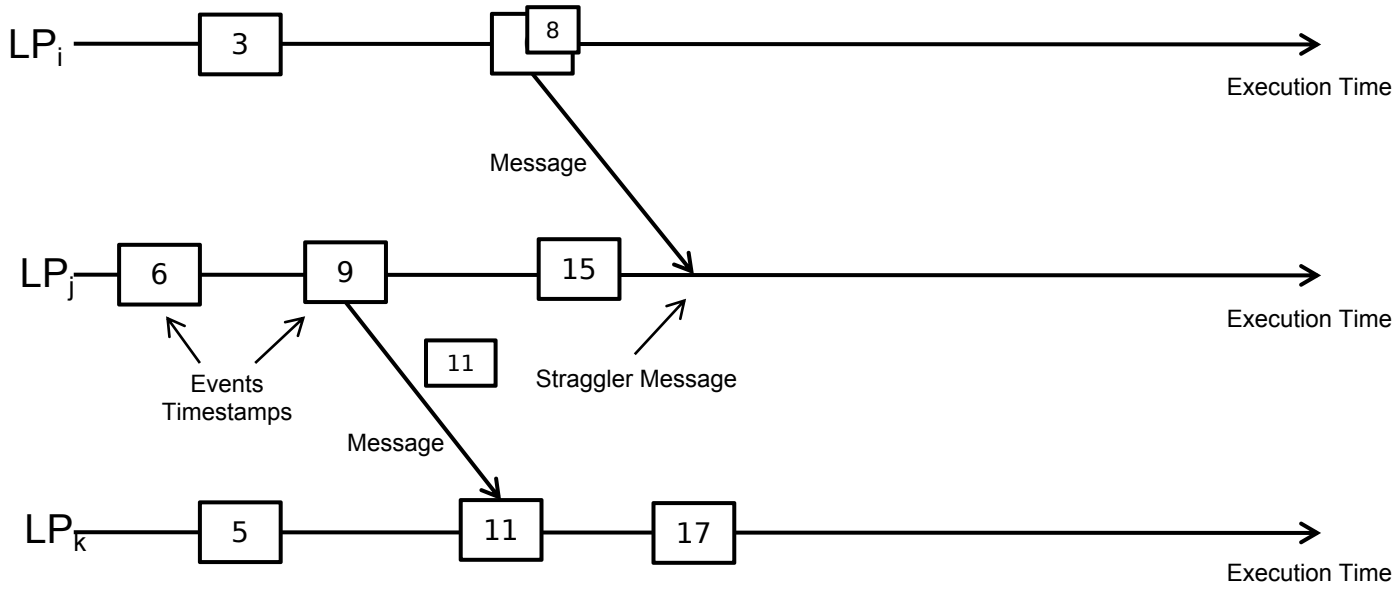
se da input queue estraggo msg, mantengo "output queue" con antimessaggio, faccio snapshot del msg a quel tempo. quando arriva al successivo, non butto via il precedente (non so se ci saranno errori). mantengo stato dei processi. se arriva straggler? torno dietro in input queue, all'ultimo evento corretto (il più grande prima dello straggler). butto via in "state queue" gli eventi errati, e gli "output queue" diventano antimessaggi, li consegno e svuoto output queue. Input queue ora può andare avanti.



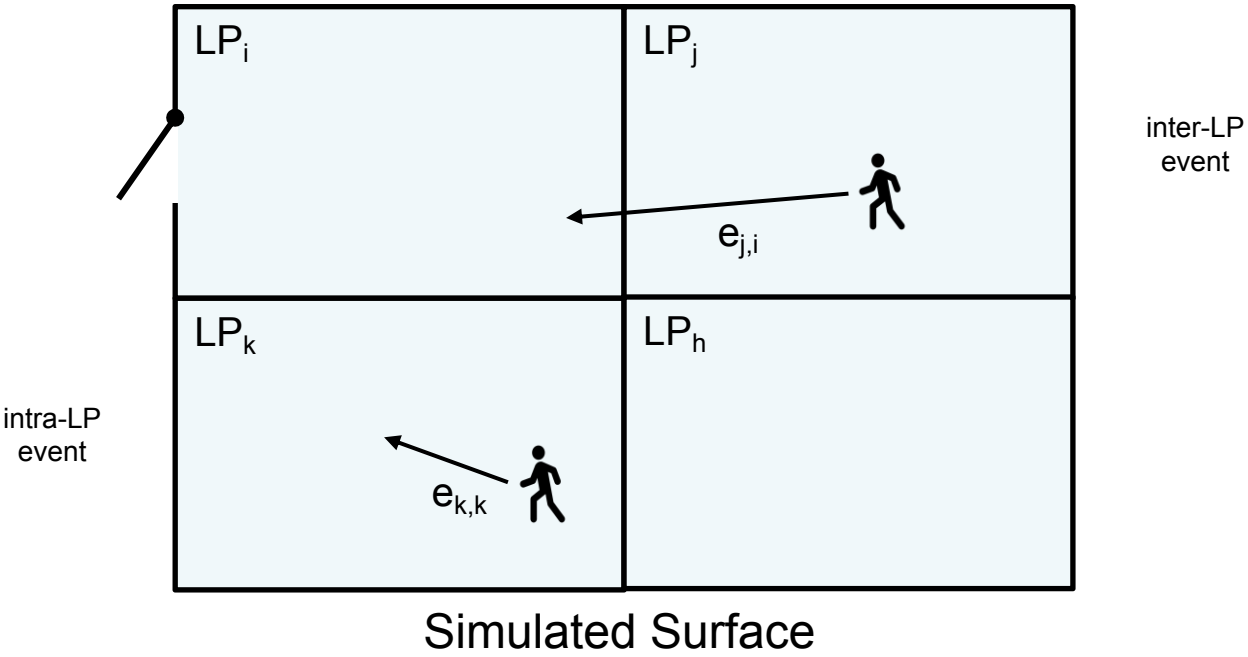
The Synchronization Problem



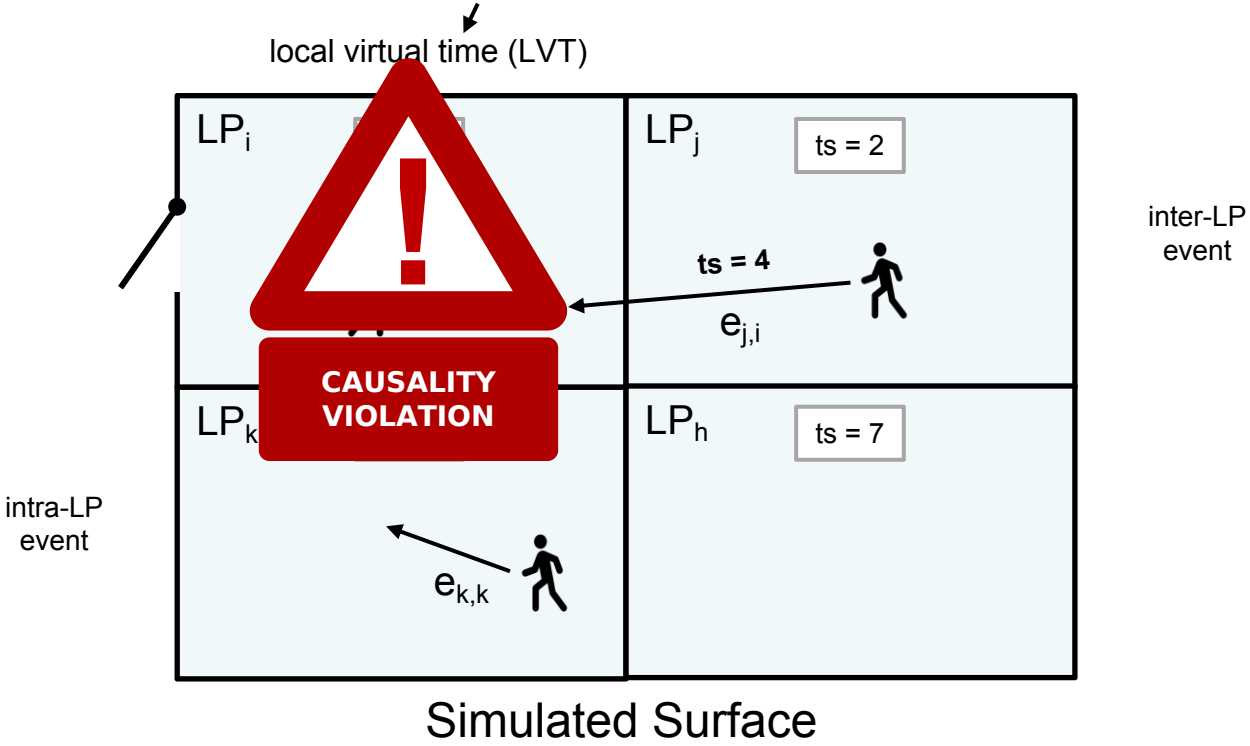
The Synchronization Problem



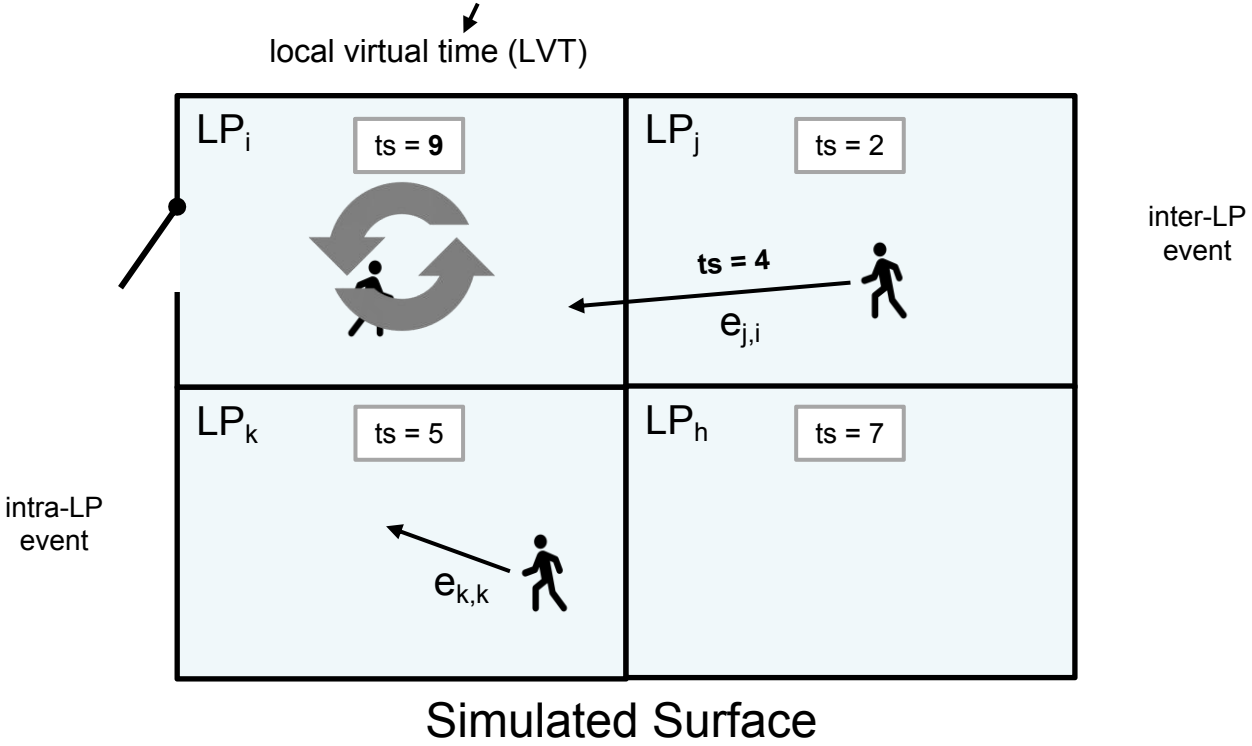
The Synchronization Problem



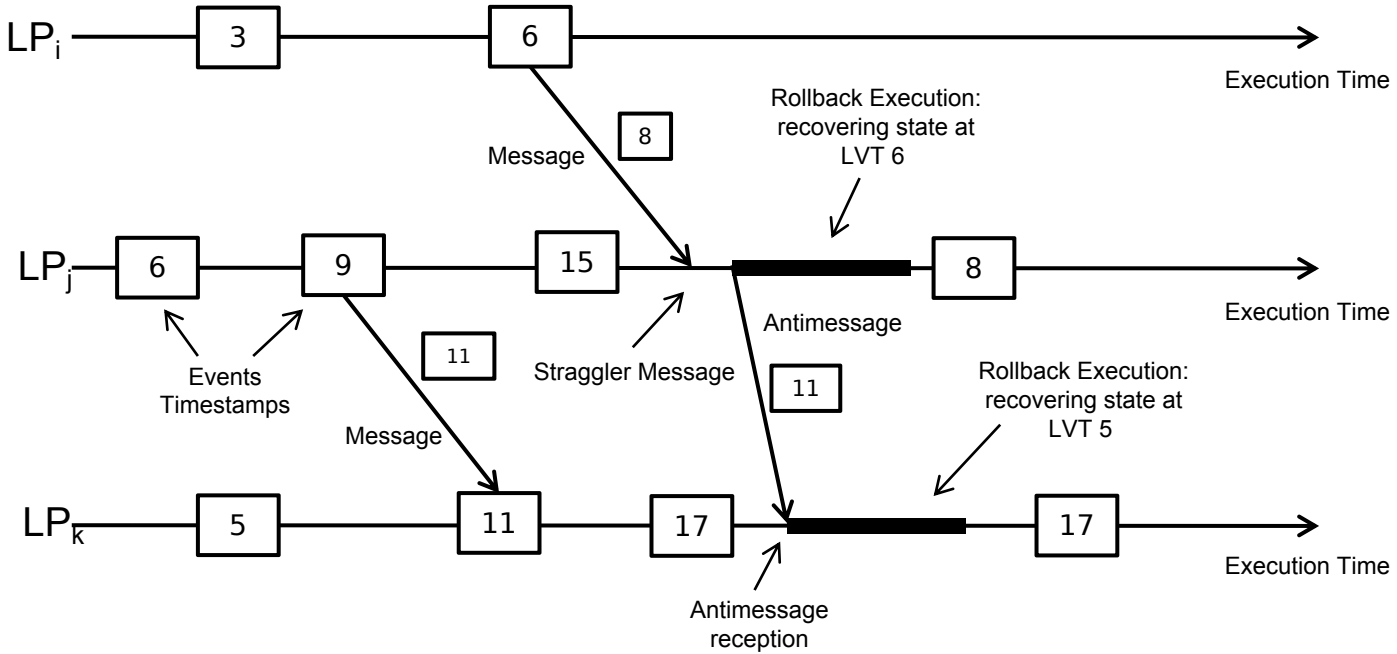
The Synchronization Problem



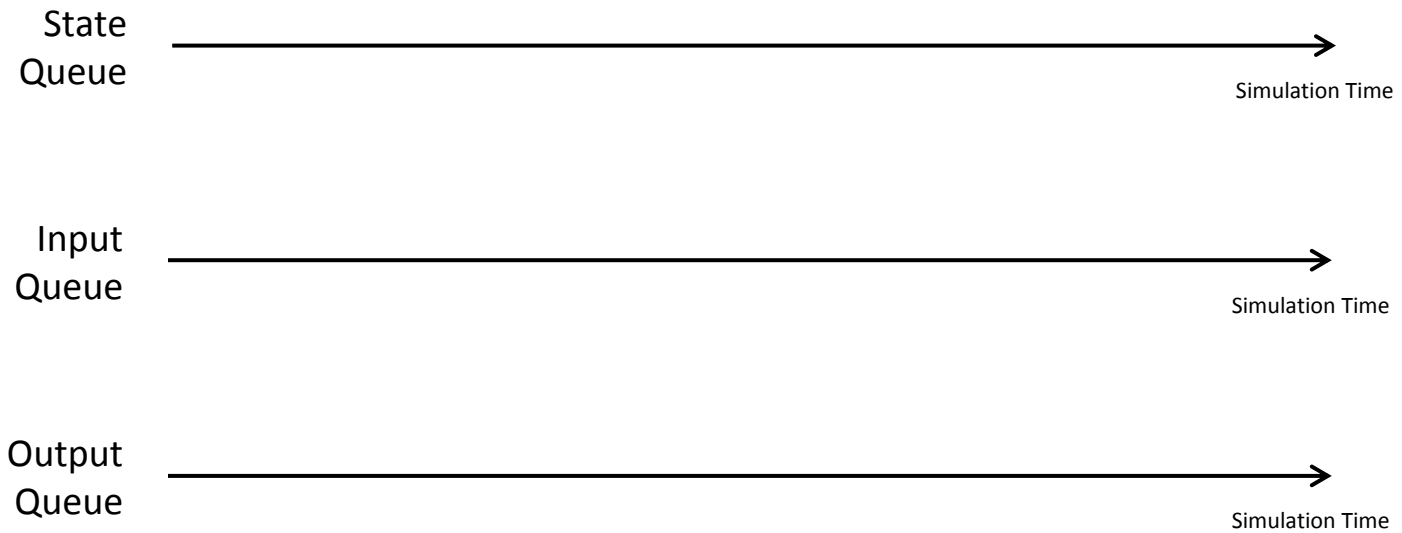
The Synchronization Problem



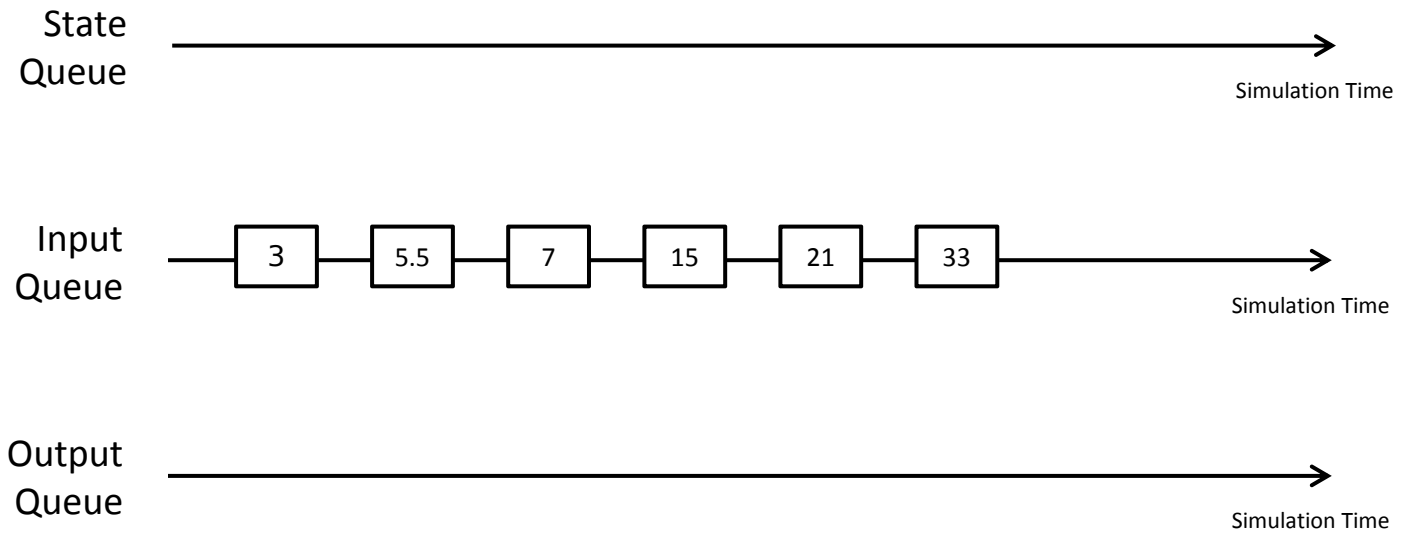
Time Warp: State Recoverability



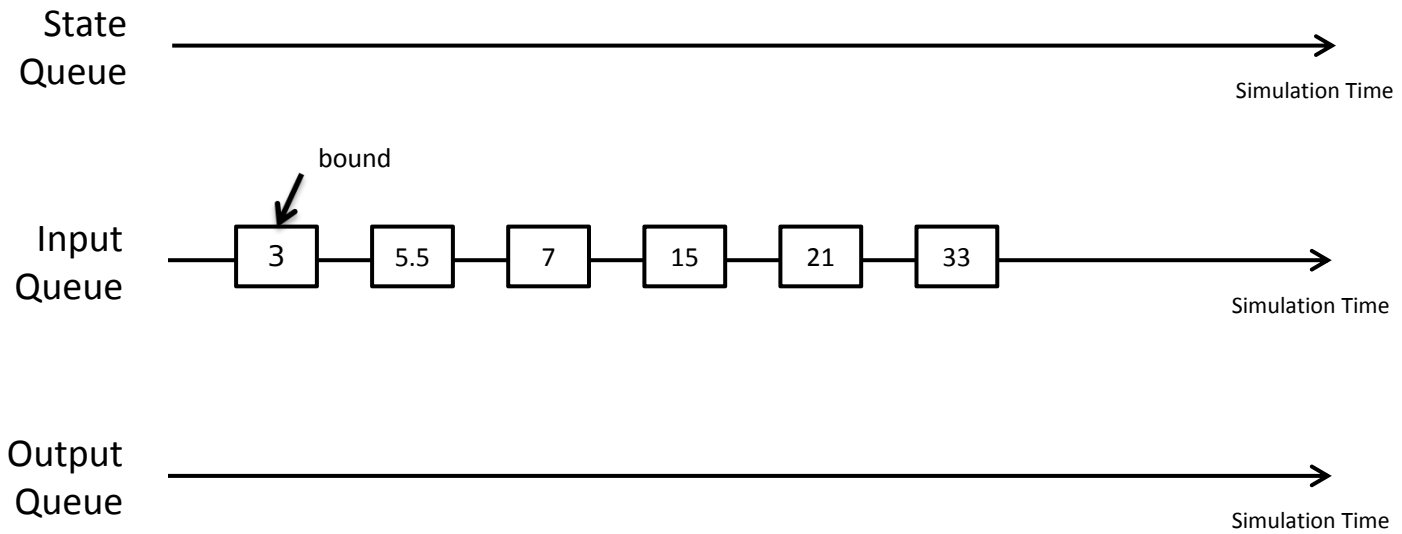
State Saving and Restore



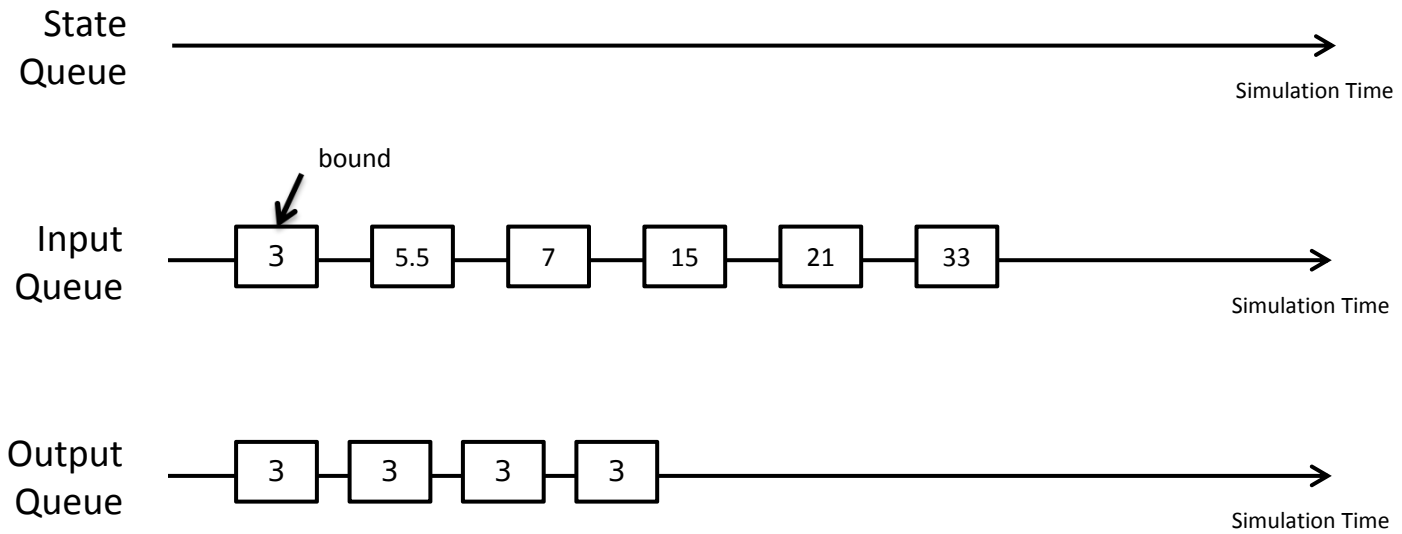
State Saving and Restore



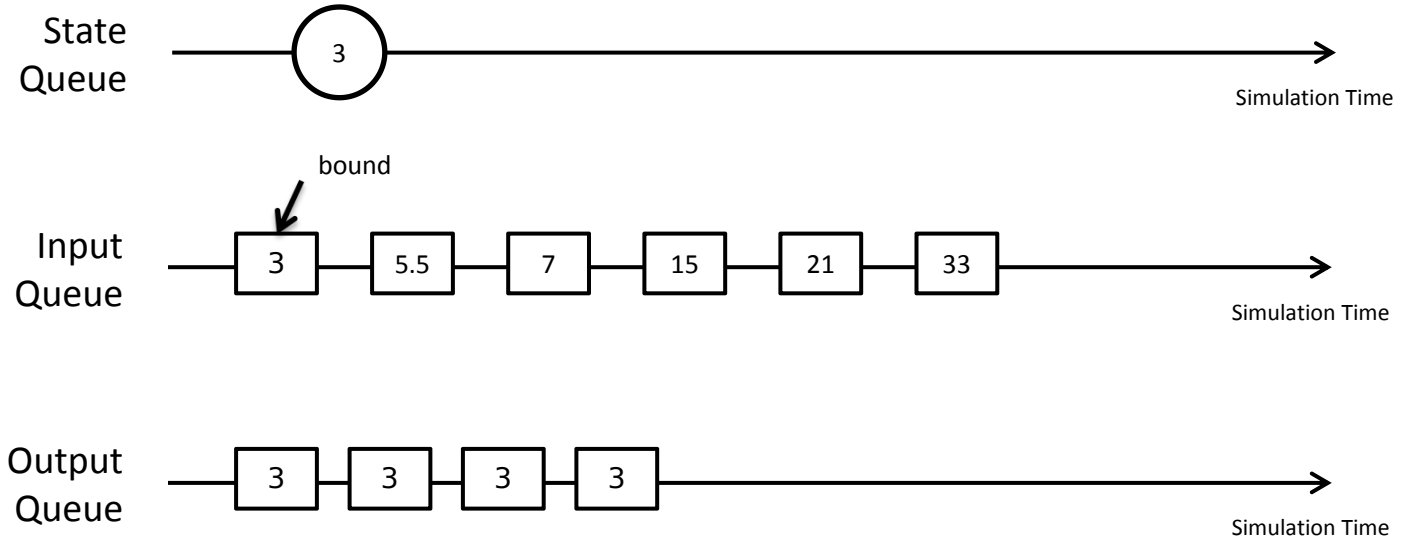
State Saving and Restore



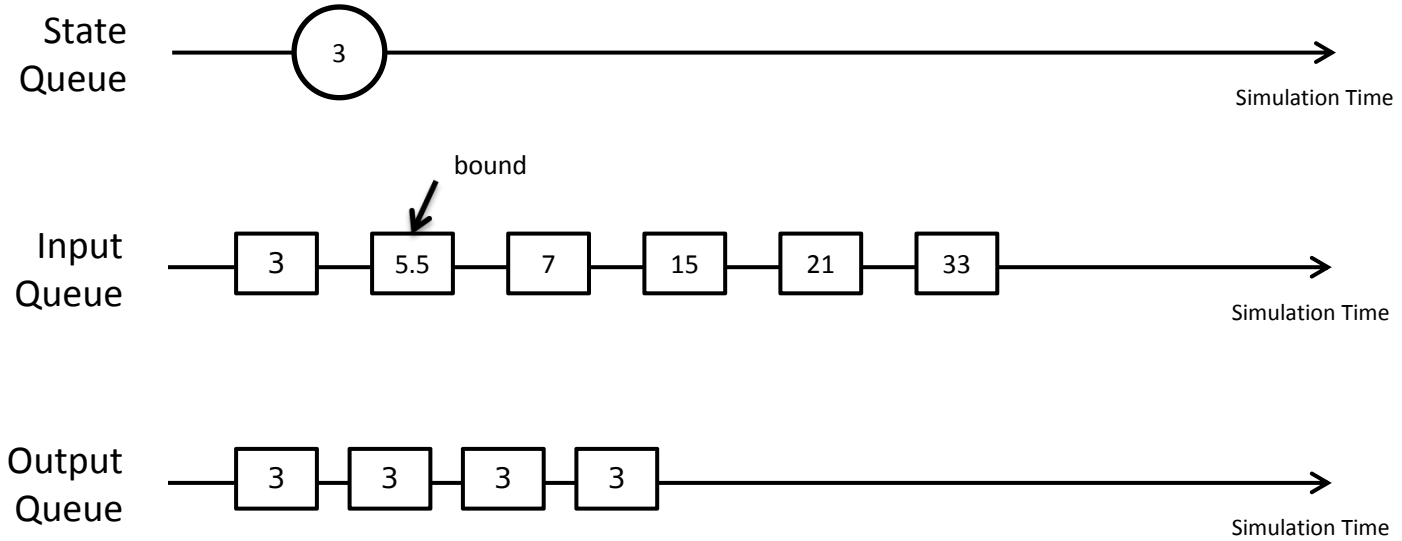
State Saving and Restore



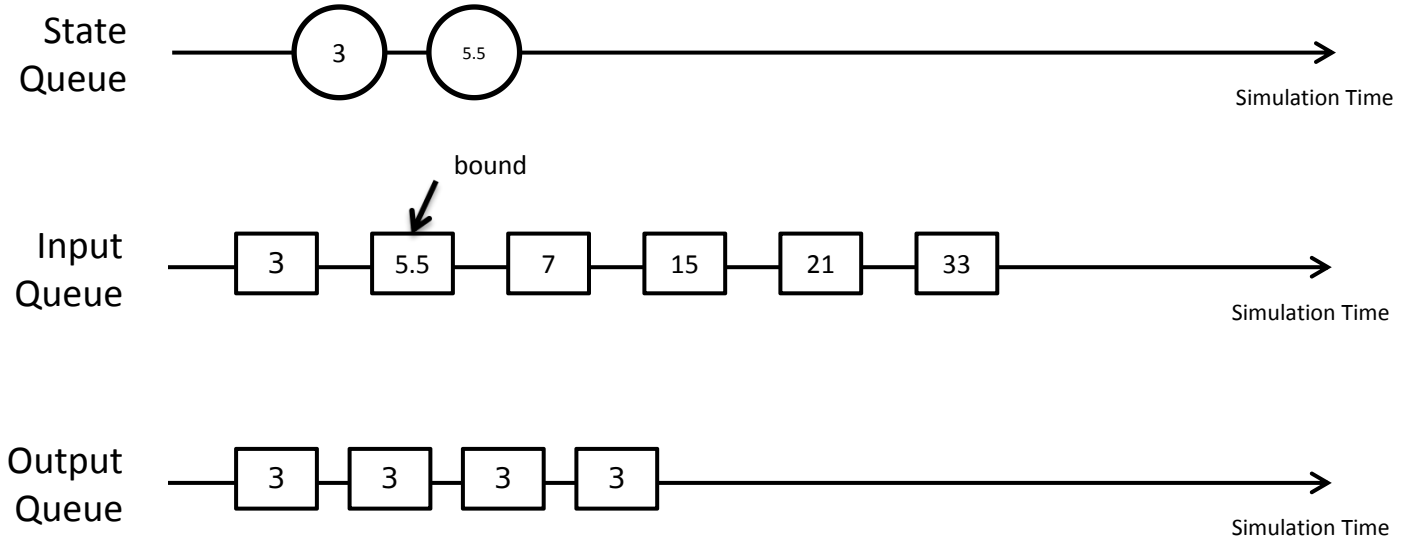
State Saving and Restore



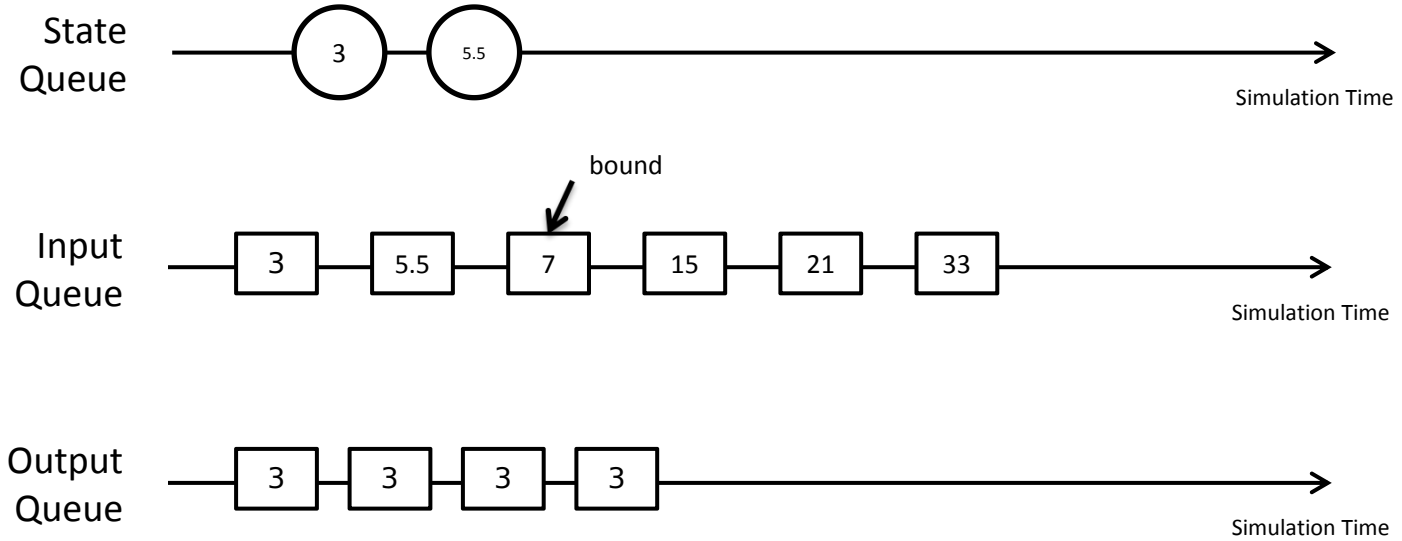
State Saving and Restore



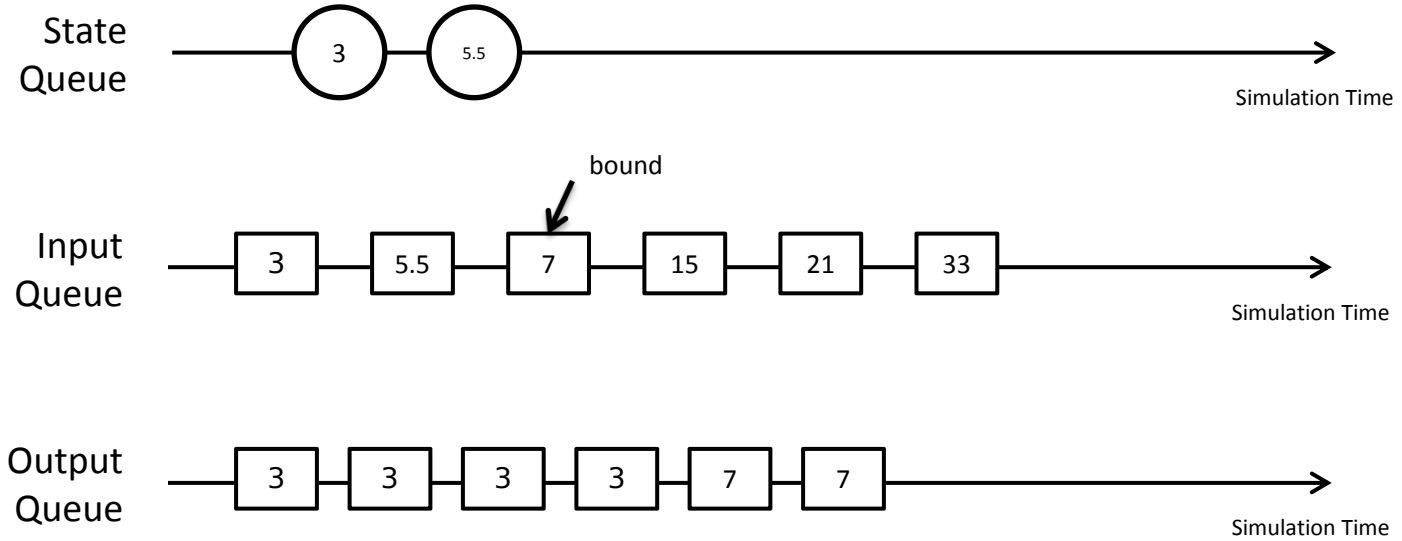
State Saving and Restore



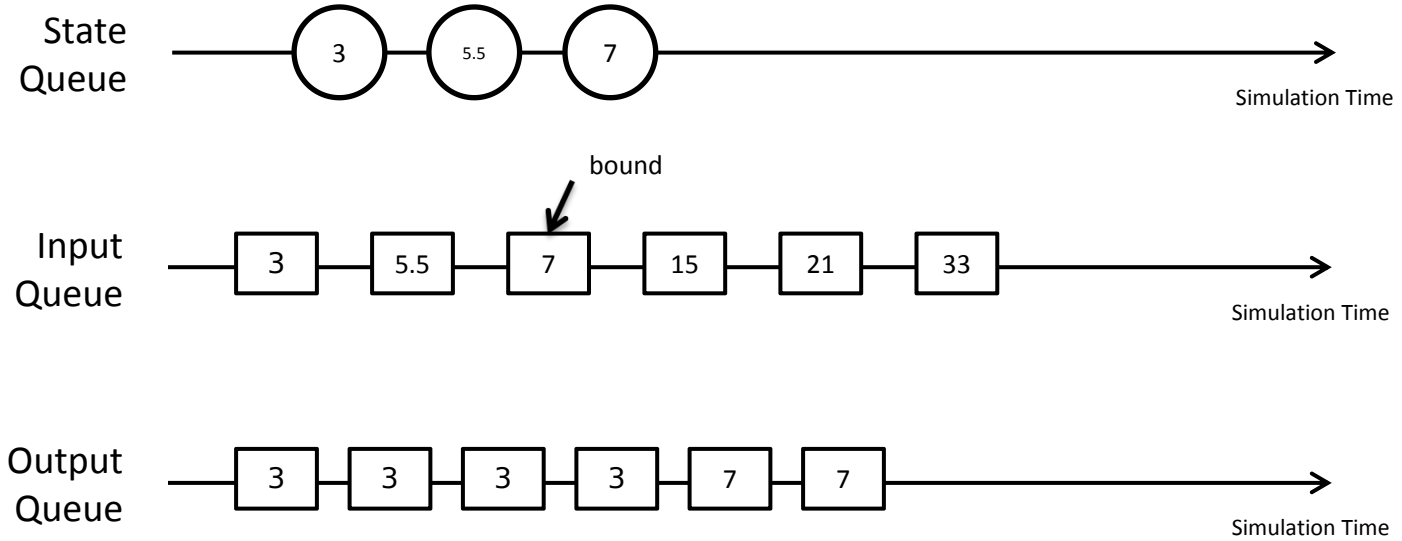
State Saving and Restore



State Saving and Restore

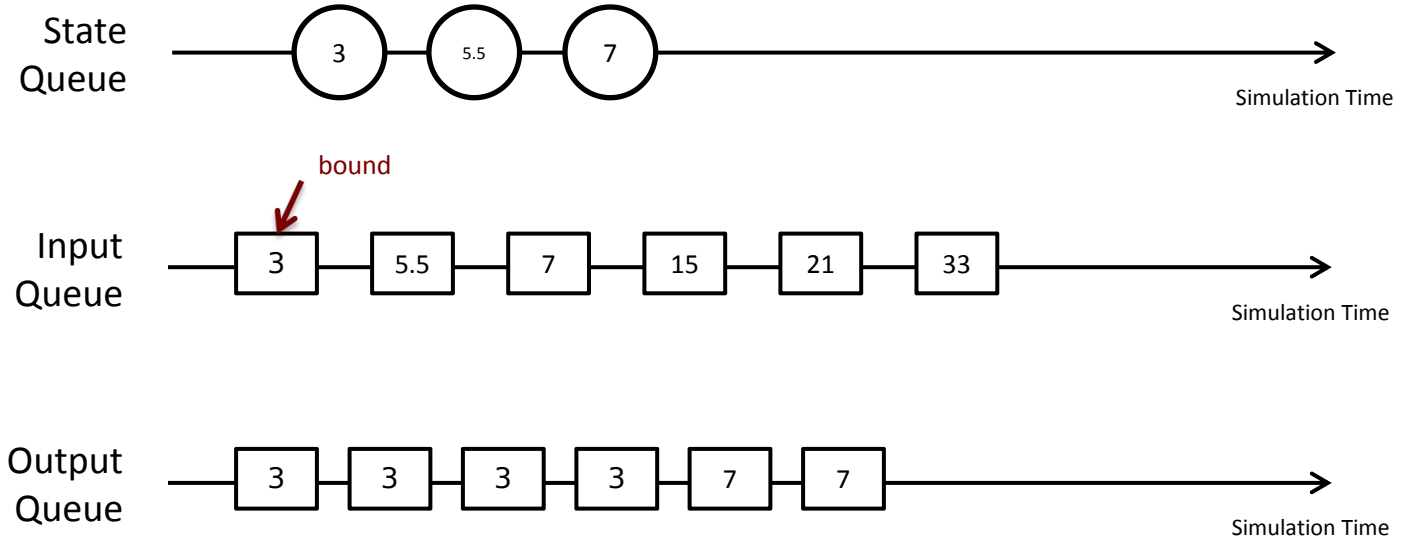


State Saving and Restore

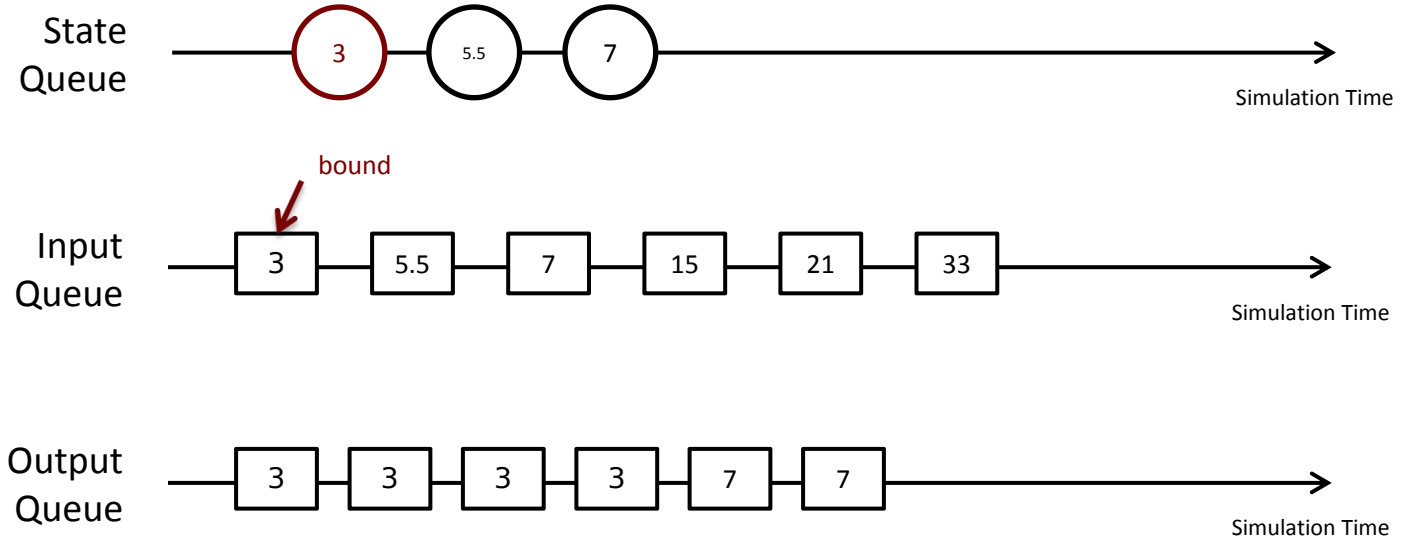


State Saving and Restore

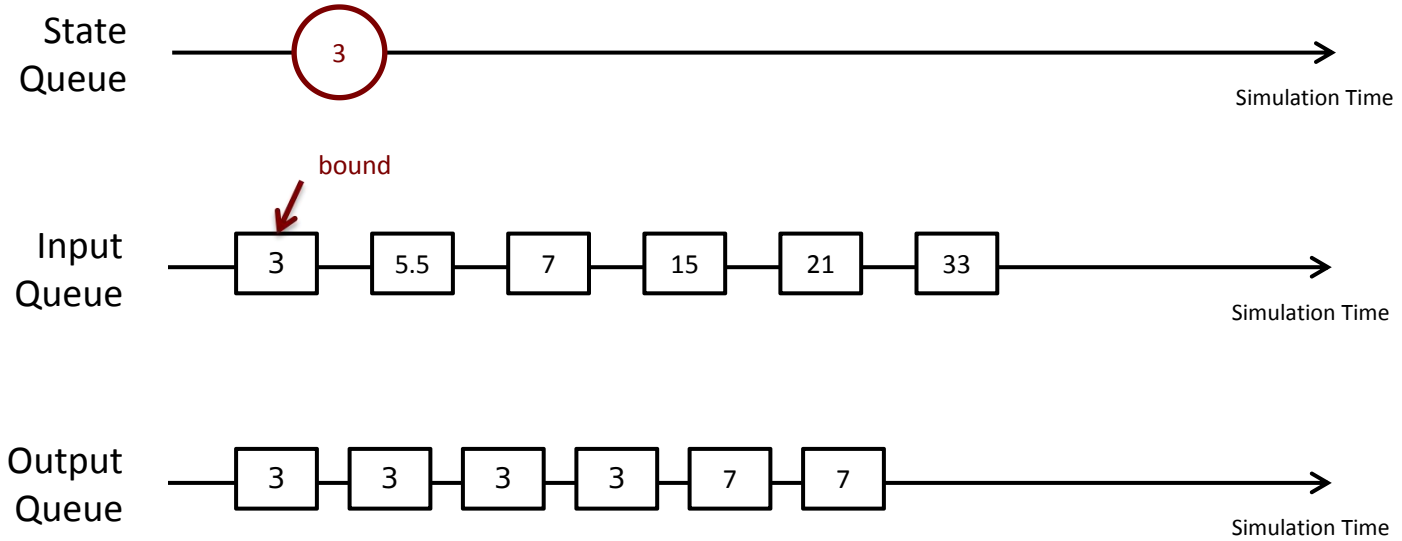
State Saving and Restore



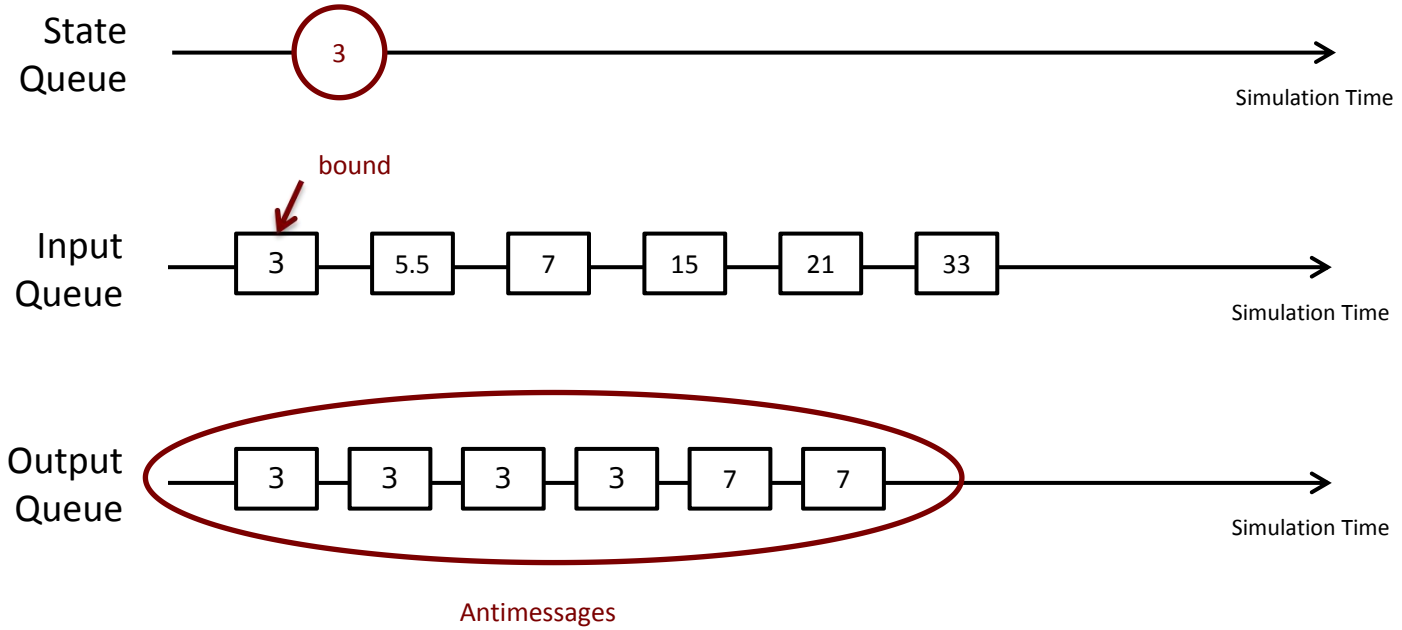
State Saving and Restore



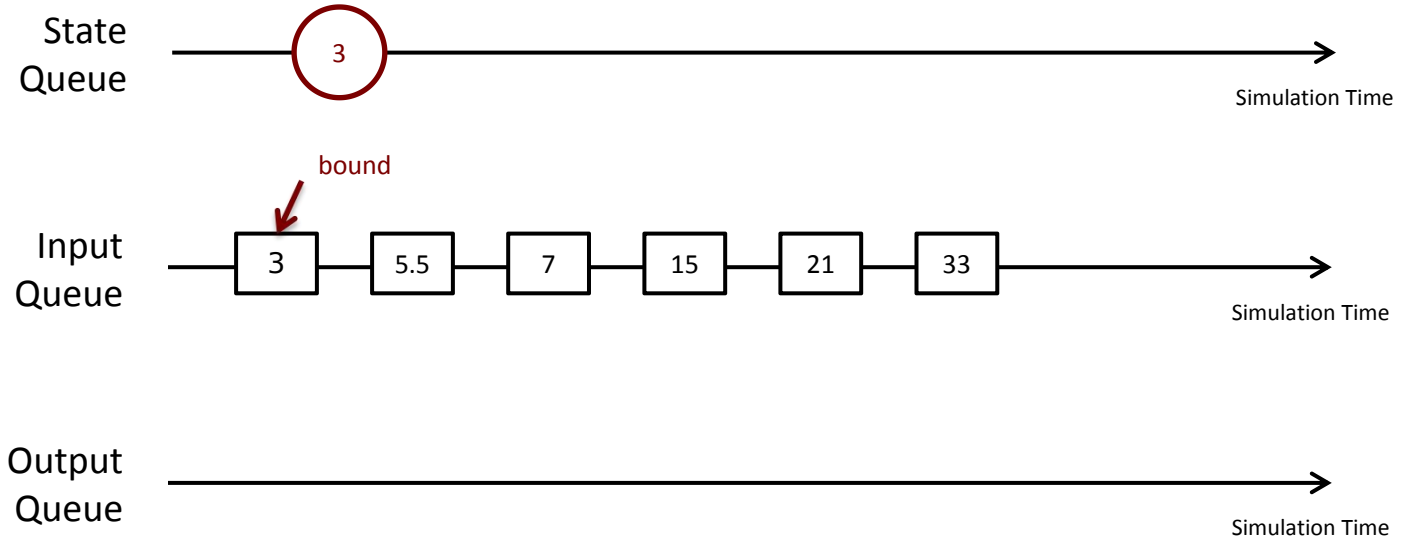
State Saving and Restore



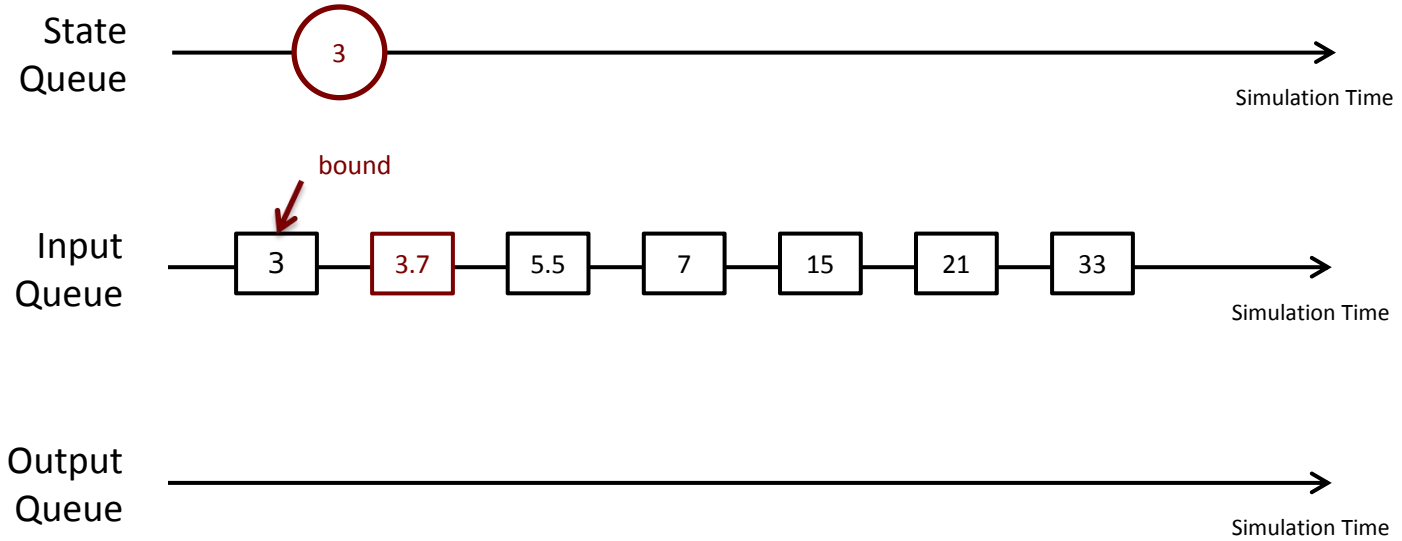
State Saving and Restore



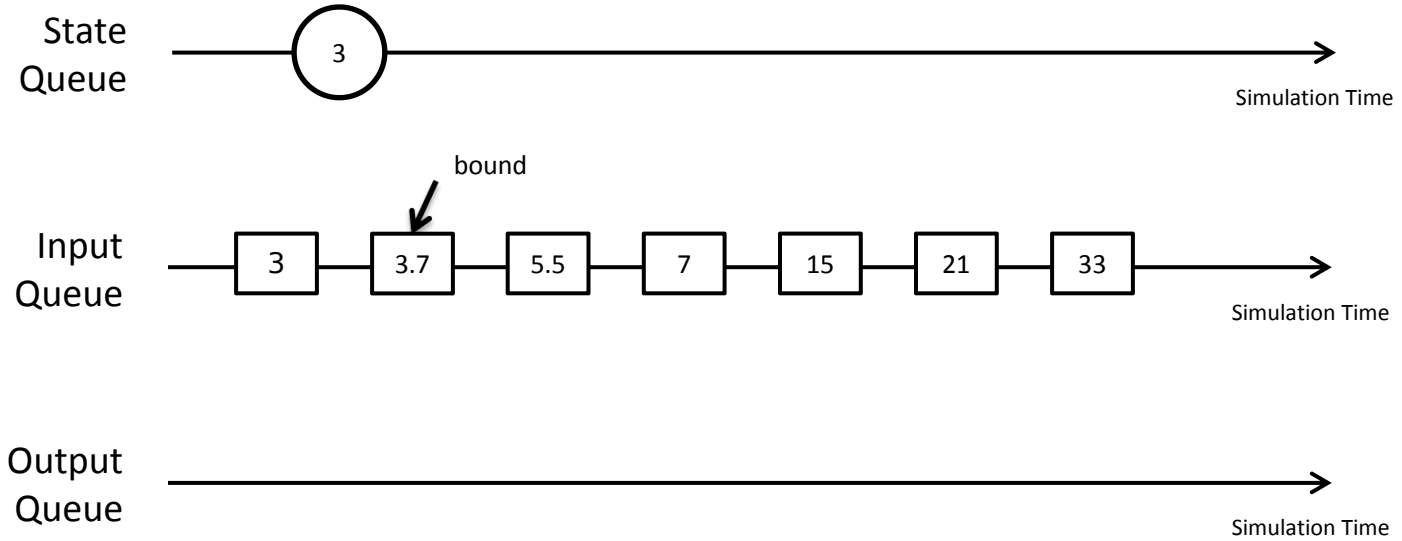
State Saving and Restore



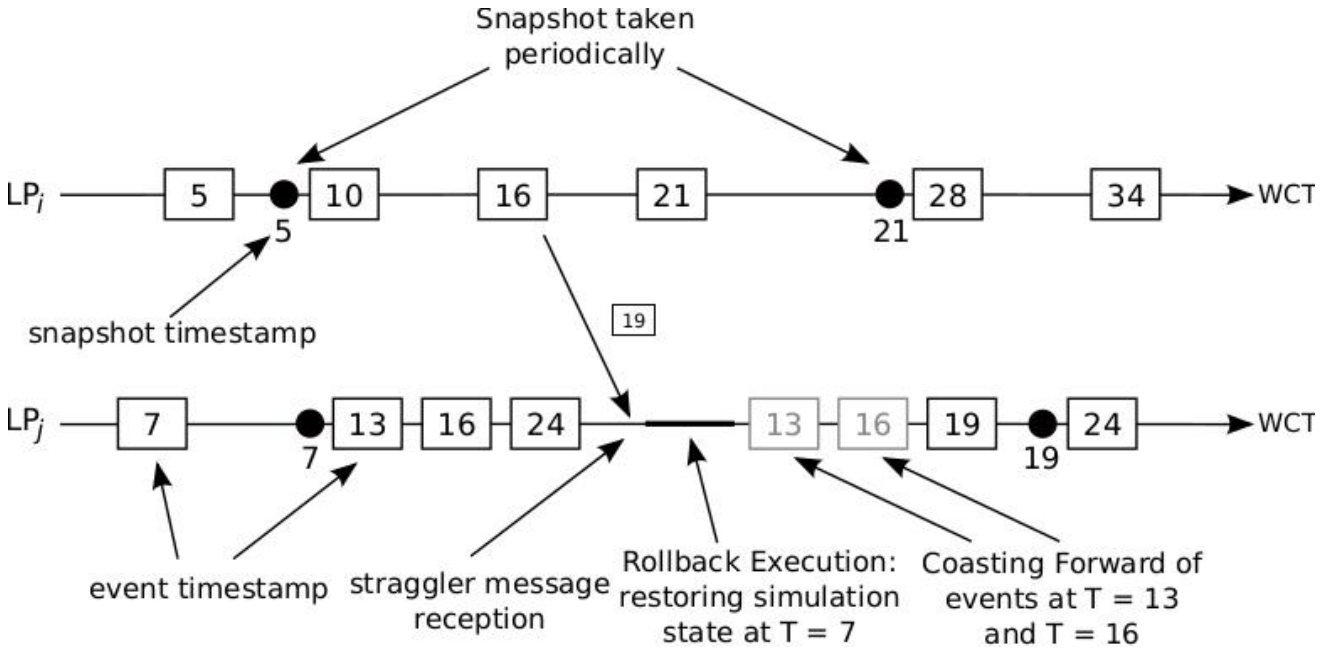
State Saving and Restore



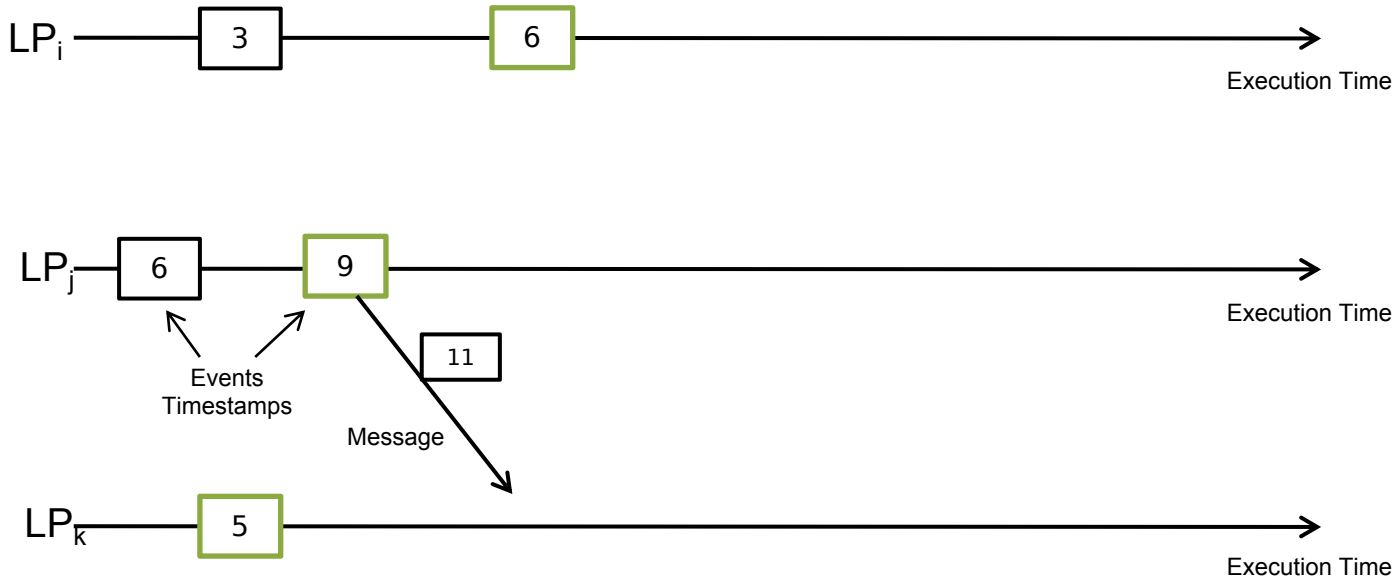
State Saving and Restore



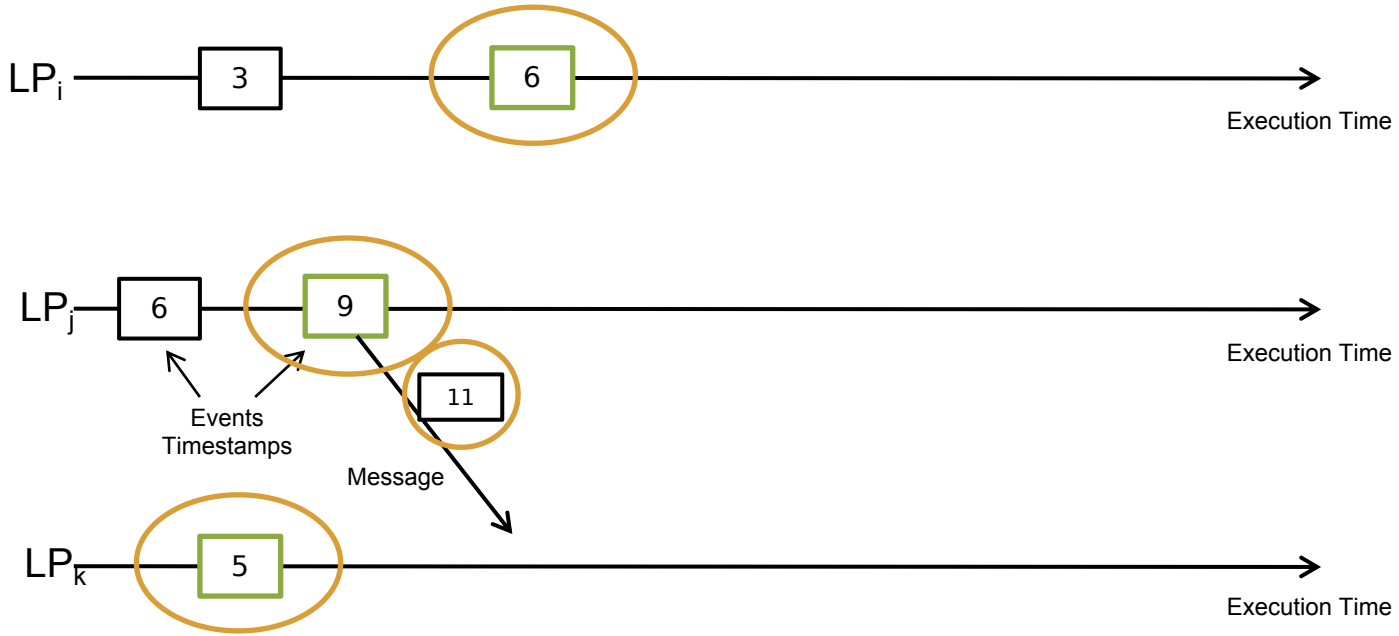
Sparse State Saving (SSS)



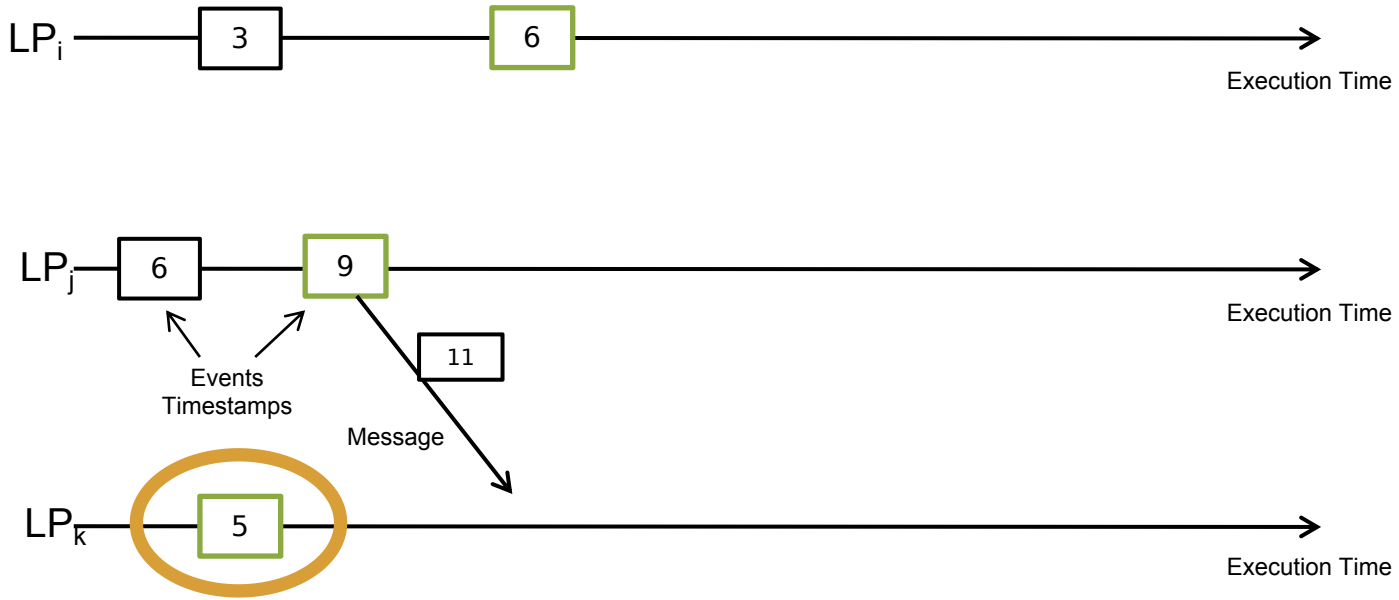
Global Virtual Time



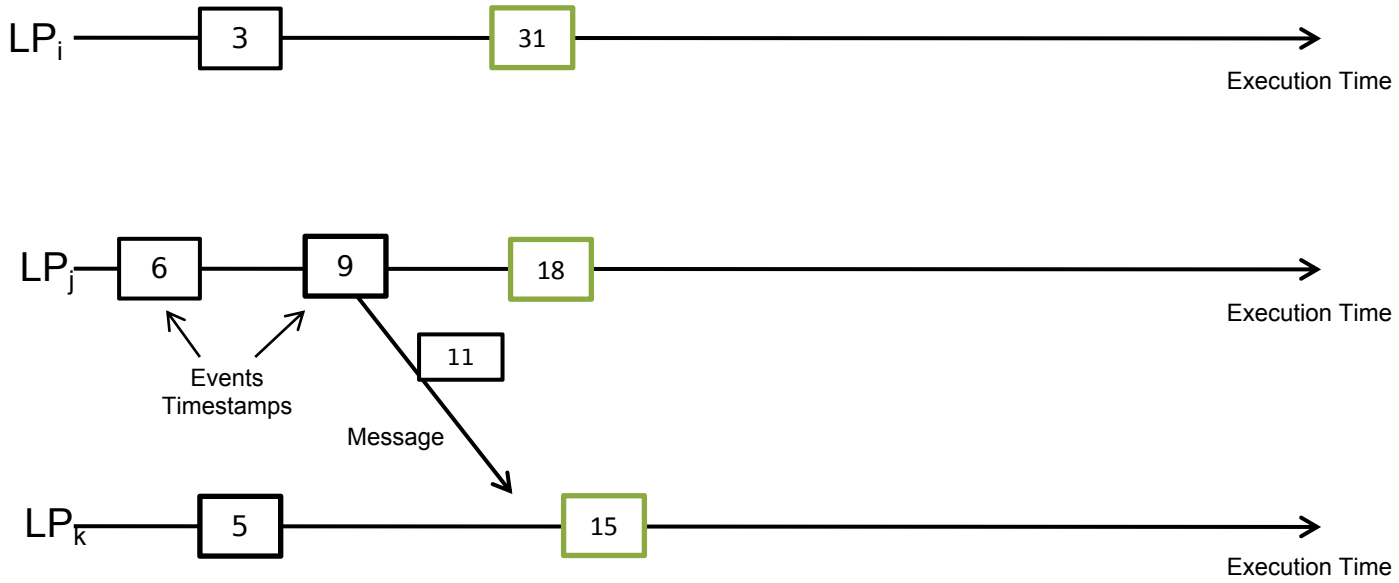
Global Virtual Time



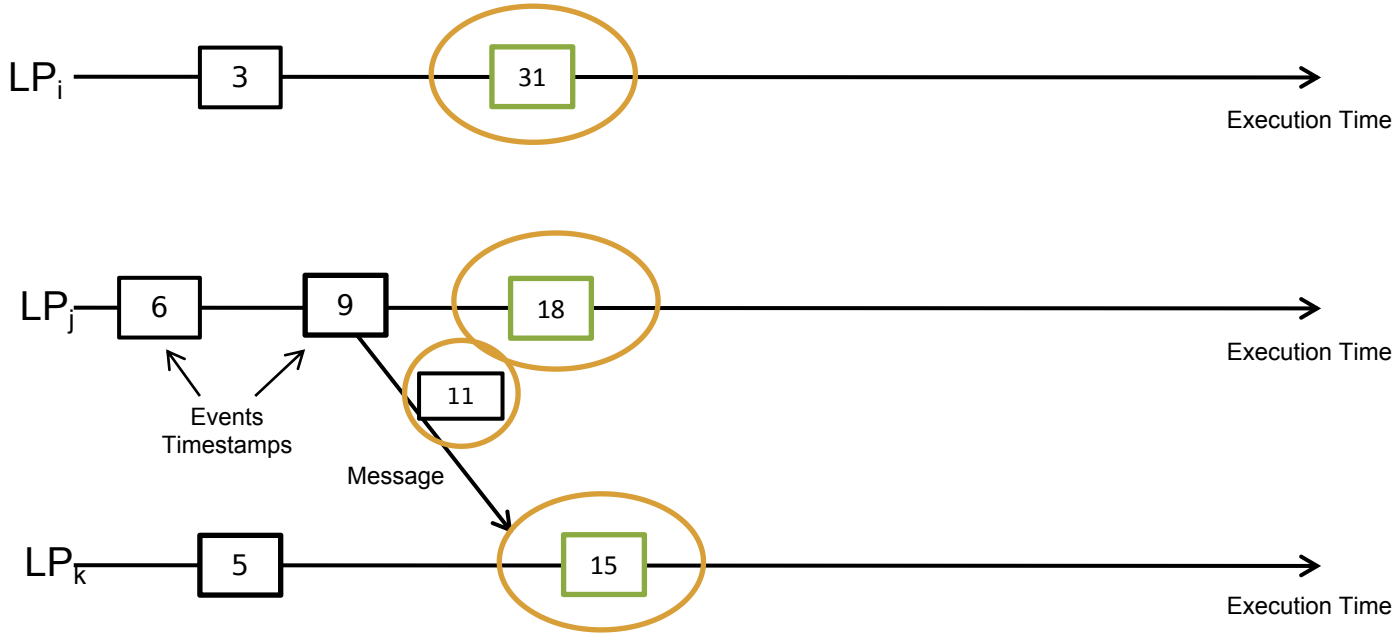
Global Virtual Time



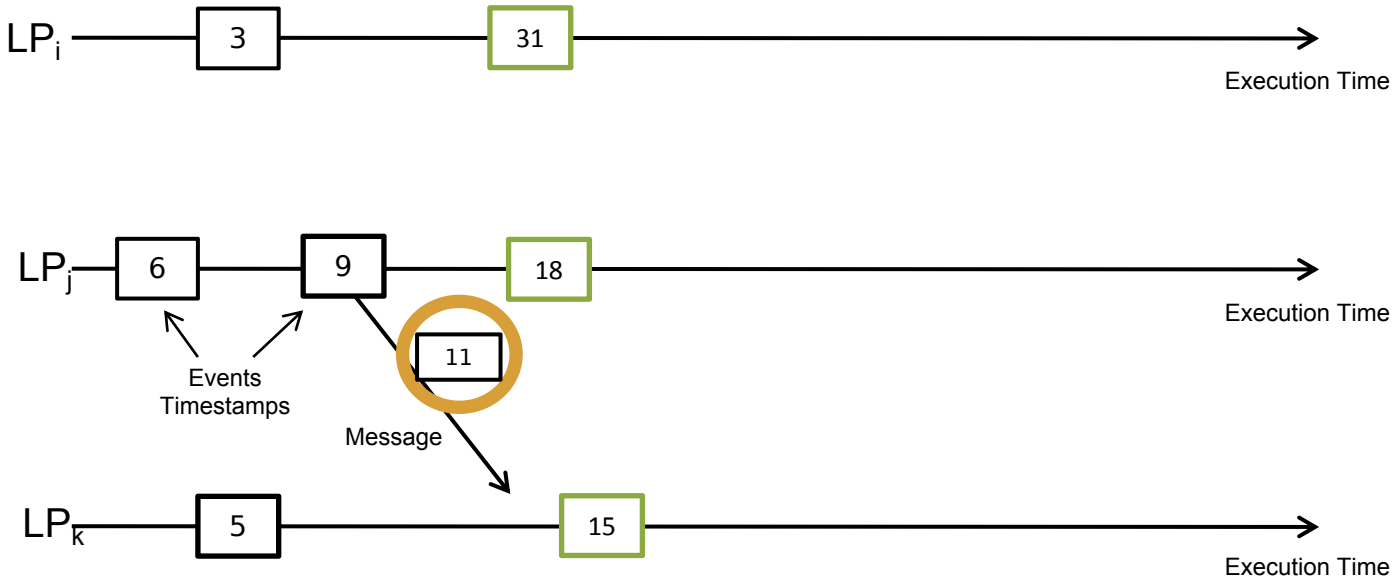
Global Virtual Time



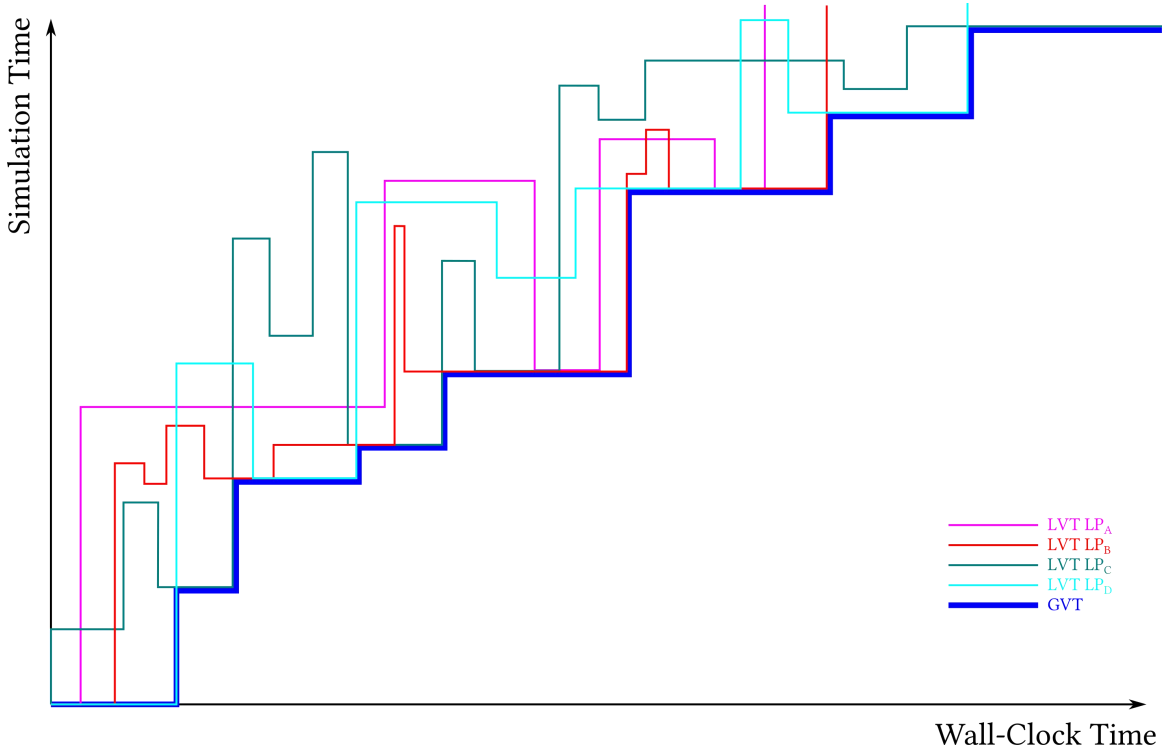
Global Virtual Time



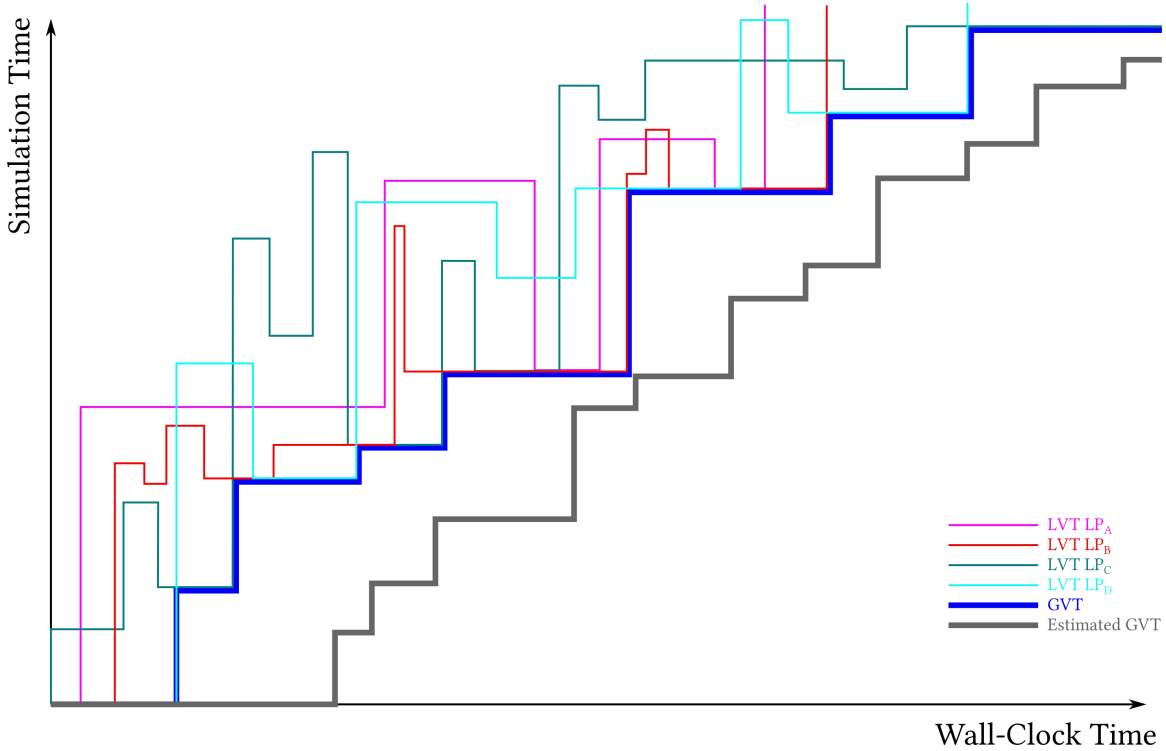
Global Virtual Time



Relations among GVT and LVT



Relations among GVT and LVT

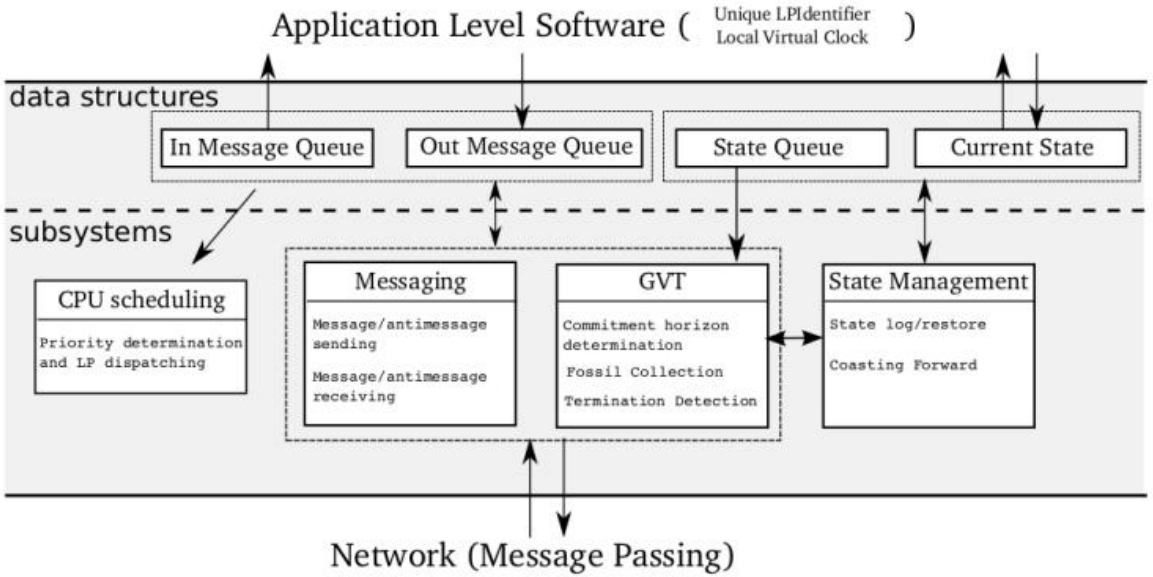


GVT Operations

- 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 = ∞ 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

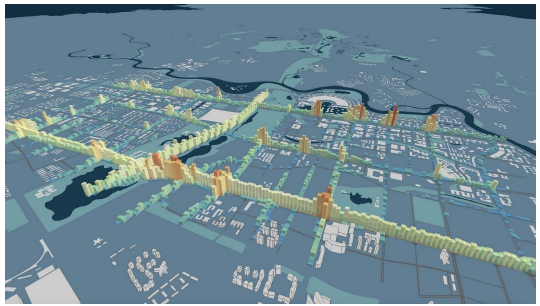
Recap: Time Warp Fundamentals



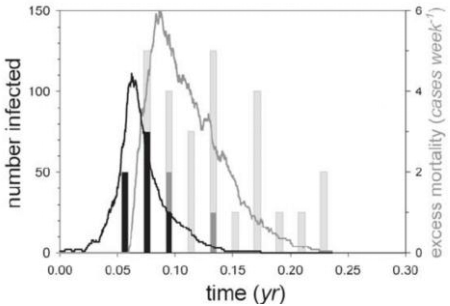
ROOT-Sim

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

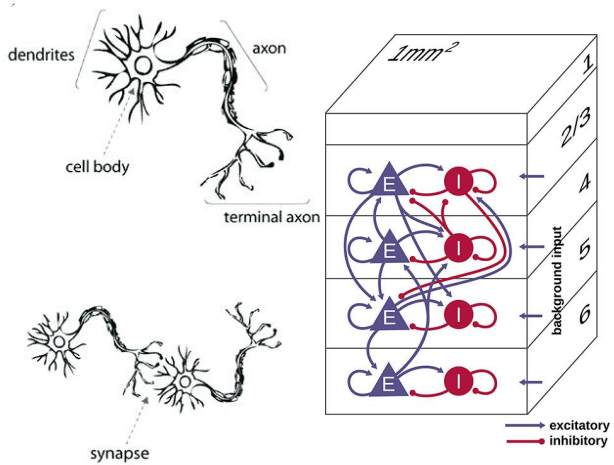
A general-purpose speculative simulation core based on state saving



Traffic Simulation



Epidemics (tuberculosis)



Spiking Neural Networks

Quanto funziona bene 'sta cosa? ROOT-SIM si basa su tale paradigma, compatibile con eventi discreti. lo applico (macchina lollopelle 4 core, 4 thread, 256 code) : 2.6 secondi. Ed ho solo 4 thread.

statisticamente funziona perchè l'inizializzazione dello stato di simulazione, il seed usato fa parte dello stato. con rollback annullo anche parte di generazione di numeri pseudorandom, come se non li avessi mai estratti.

Example Session

PCS on ROOT-Sim