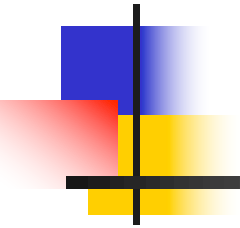
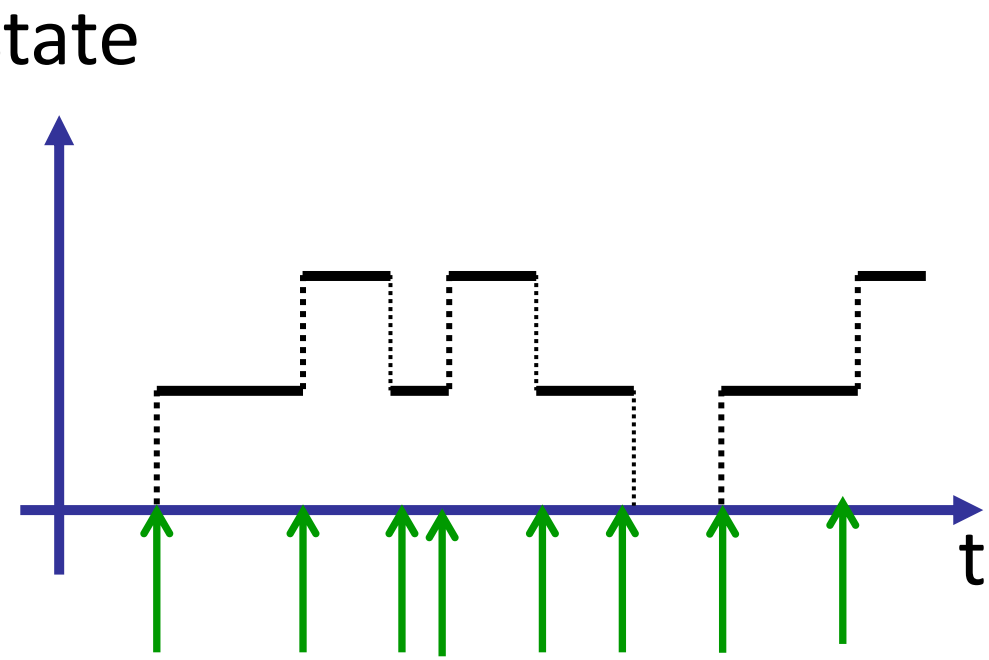


Discrete-Event Modeling

Discrete-event simulation

- We want to model a system using a discrete-event simulator
  - Events are instantaneous time occurrences at which the system *state* changes



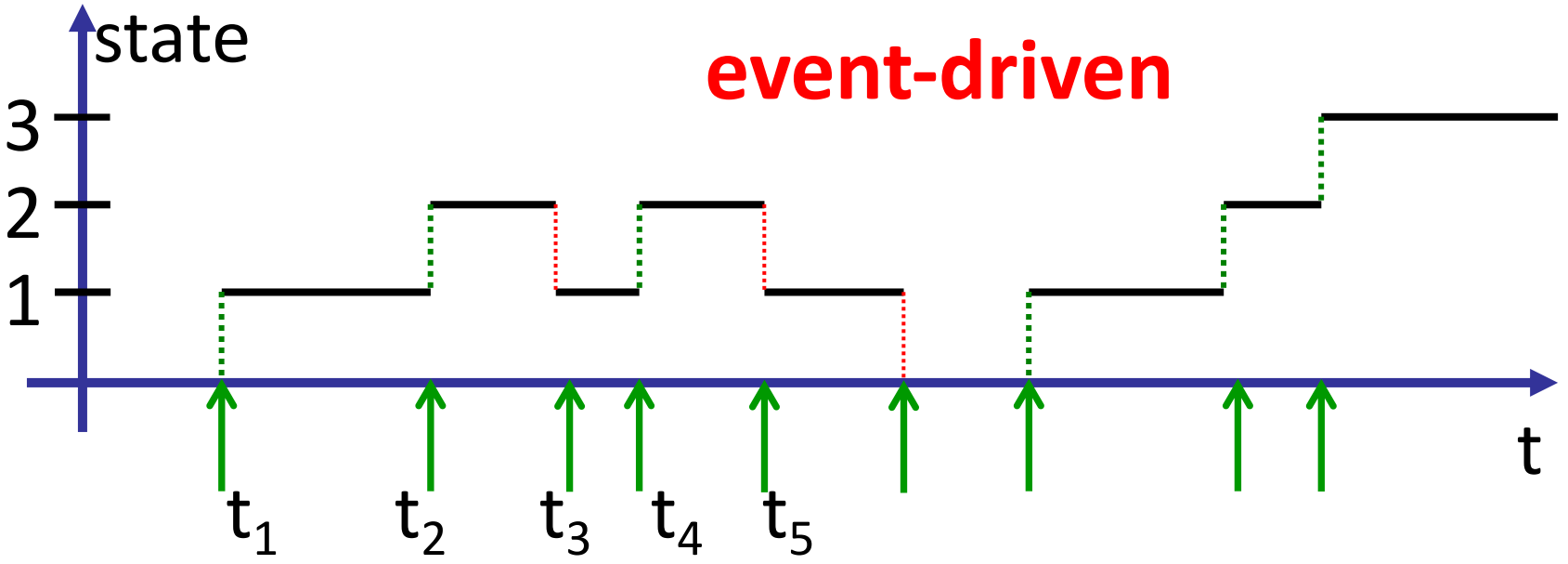
Discrete-event simulation

- We want to model a system using a discrete-event simulator
  - Events are instantaneous time occurrences at which the system *state* changes
  - We are interested in representing the system in these instants of time
  - Time advances through discrete steps from one event to the next
  - The way the system evolves depends on the nature of the intervals between events, whose duration can be either deterministic or stochastic

Discrete-event simulation

- Since we are interested in the system just in the moments at which its state changes, there is no need to model the system between state changes
- We use a next event approach: the system time jumps forward with discrete steps to the time instant at which the next state change (event) happens

Example



- The events occur in time instants t_1, t_2, …
- Time advances corresponding to these instants

Discrete-event simulation

- Definitions (reminder)
  - *Entities:* objects of interest in the system
  - *Event:* an instantaneous occurrence that changes the system state
  - *Activity:* a time period dedicated to a specific operation
  - *Process:* a sequence of events, ordered in time, involving a single entity

Discrete-event simulation

- We have some alternative solutions for discrete-event modeling
  - we can focus on the actions to be performed when each event happens -> *event scheduling*
  - we consider all the actions of each entity while it is inside the system, focusing on its relations with other entities -> *process interaction*

Event scheduling

- *To schedule*
  - To plan the future execution of an operation
- The *event scheduling* approach requires that we decide when in the future an event must happen and we schedule its execution
- It is the most common approach
  - Supported by almost any simulation language
  - Easy coding

Event scheduling

- We need to identify or define
  - the entities in the system
  - the results we want to obtain from the simulation
  - the state variables
  - the events, i.e., what makes the state variables change

Event scheduling

- For each event, we must define the sequence of operations to be executed when the event happens
- These operations depend on the state of
  - the entity that triggered the event
  - other entities in the system

Event scheduling

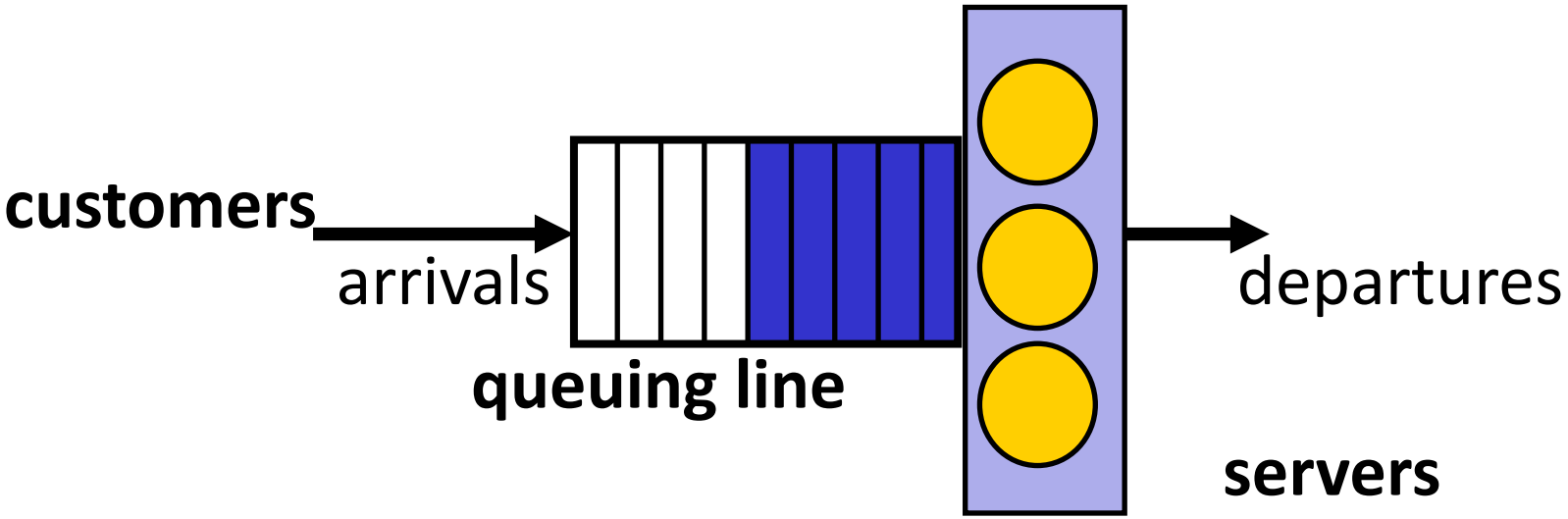
- A special and very important operation executed when an event is triggered is *the scheduling of new events*
- In this way, starting from few events scheduled at the beginning of the simulation, the system will naturally evolve through the cause/effect relations among events

Event scheduling

- Let us use a concrete model: the queue model
- The best way to gain a deep understanding of the next-event scheduling approach is to build step-by-step the simulation model for a key element in telecommunication networks: the queue

Queuing systems

- Systems in which
  - **Customers** arrive to receive **service**
  - One or more **servers** provide service



Model of a queuing system

- We consider a queuing system
- In such a system users (customers or clients) receive in an ordered way a service, provided by one or more agents (servers) according to a specific service discipline
- It is the most common (sub)system we can find studying a telecommunication network

Model of a queuing system

- Which are the elements/processes?
  - the clients requiring service
  - the servers providing it
- What are we interested in?
  - average time to be served (waiting time)
  - average number of clients waiting for service
  - when there is limited space for the waiting clients, the probability that a client might not be served

Model of a queuing system

- Which are the state variables?
  - the number of clients in the system
- Which are the events?
  - the arrival of a client
  - the end of the service of a client

Model of a queuing system

- Let us start with a very simple version of our model, improving it later with incremental steps after identifying any detail needed and missing
- Version 1.0

- Arrival
  - If the server is idle, occupy it
  - Otherwise, enter the waiting line

- End of Service
  - If there is someone in the waiting line, serve it
  - otherwise, become idle

Model of a queuing system

- Version 1.0 is very simplistic
- The relations between events are missing
  - End of Service events are caused by the presence of clients in the system and by their services
- We need to explicit these relations

Model of a queuing system

- Version 2.0

- Arrival
  - If the server is idle
    - Make the server busy
    - Schedule the end of service
  - else
    - Wait in line

Model of a queuing system

- Version 2.0
- End of Service
  - If someone is in the waiting line
    - Select next in line
    - Schedule the end of service
  - else
    - Make the server idle

Model of a queuing system

- We have pointed out how the end of each service is planned at its start
- We are still missing a few key details in our simulation model
  - Each service has its own duration that must be chosen according to the requirements of the considered system
  - We may need to store information about the waiting clients in suitable data structures

Model of a queuing system

- Version 3.0

- Arrival
  - If the server is idle
    - Make the server busy
    - Determine the service time T_s
    - Schedule the end of service at time $T_{curr}+T_s$
  - else
    - Create a record for the client
    - Insert the record in the queue

Model of a queuing system

- Version 3.0
- End of Service
  - If someone is in the waiting line
    - Select a client to be served
    - Extract its record from the queue
    - Determine service time T_s
    - Schedule the end of service at time $T_{curr}+T_s$
  - else
    - Make the server idle

Model of a queuing system

- We are still missing a key detail

  Who is controlling the arrival of clients?

- Clients arrival is usually characterized as a random process, e.g. through the distribution of the inter-arrival time
- The most common solution to manage the arrival of clients in the system is by self-sustaining

  At each arrival, the next one is scheduled

- We also need to develop a correct and coherent management of the records for the clients

Model of a queuing system v4.0

- Arrival
  - Compute the inter-arrival time T_{ia} for next client
  - Schedule an arrival at time $T_{curr}+T_{ia}$
  - Create a record for the client
  - Insert the record in the queue
  - If the server is idle
    - Make the server busy
    - Determine the service time T_s
    - Schedule the end of service at time $T_{curr}+T_s$

Model of a queuing system v4.0

- End of Service
  - Extract from the queue the record for the client who has just been served
  - Destroy the record
  - If someone is in the waiting line
    - Select a client to be served
    - Determine its service time T_s
    - Schedule the end of service at time $T_{curr}+T_s$
  - else
    - Make the server idle

Model of a queuing system

- We can further increase the details of the model, considering
  - the queuing policy
  - detailed management of the record for the clients
  - the operations needed to measure the desired performance indexes
- Measurements are quite a critical issue
  - Where and how do we carry them out?

Process interaction

- This modeling approach is focused on the actions performed by each entity during its lifetime in the system
- A *process* is a sequence of events and activities involving a single entity
- The behavior of each entity depends on the actions and the state of the other entities in the system
- In the model we need to highlight the interaction between entities and, as a consequence, between the processes they are represented from

Process interaction

- The process represents the actions of the entity during its lifetime
- The events are now representing suspensions and reactivations in the execution flow of the entity
- Two classes of events:
  - *unconditional* -> the time at which the event happens is known in advance
  - *conditional* -> the time at which the event actually happens depends on external conditions (e.g. the state of other entities)

Process interaction

- The server model is a non-terminating loop: we need just to schedule its execution once, when the simulation starts
- Several client processes may be active at the same time, one for each waiting client, but there is always a single server process
- Suspension/reactivation points are indicated by the blue text: they are the moments at which the control flow moves from one process to the other

Process interaction

- Process interaction is a very powerful modeling approach
  - The behavioral description of each entity is concentrated in a single block, instead of being distributed among several events
- The compact representation of the model hides the implementation complexity due to
  - conditional/unconditional events
  - reactivation points of the execution flow

Wrap-up

- In discrete-event simulation, events are instants in which the system state changes
- Next event approach: times evolves according to the occurrence of events
- Two main approaches:
  - Event scheduling:
    - model actions performed at each event
  - Process interaction
    - model actions of each entity