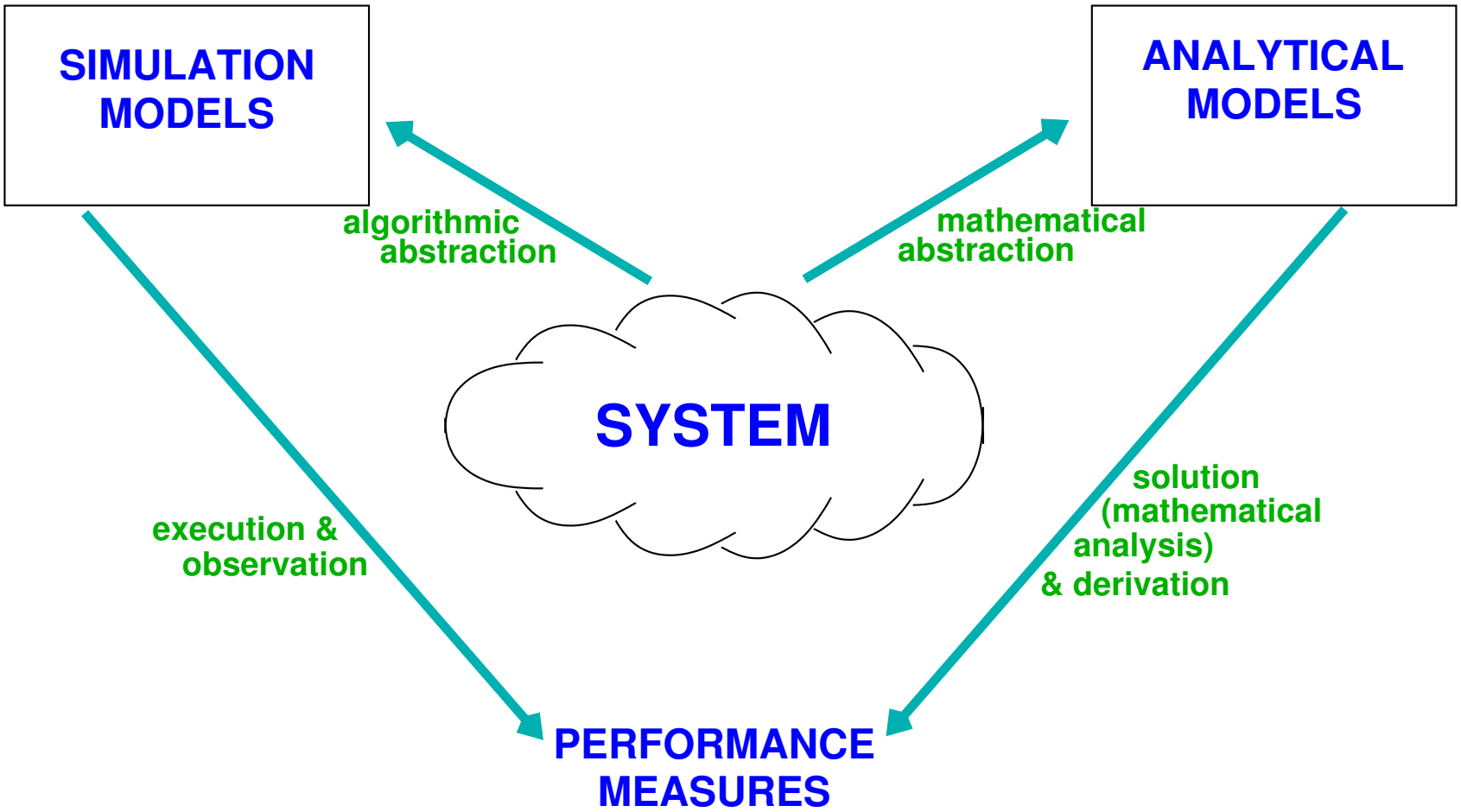
Performance Modelling — Lecture 13: Simulation Models — Introduction and Motivation

Jane Hillston
School of Informatics
The University of Edinburgh
Scotland

6th March 2017

Introduction



Assumptions

- We still assume that the system is characterised by a family of random variables $\{X(t), t \in T\}$.

Assumptions

- We still assume that the system is characterised by a family of random variables $\{X(t), t \in T\}$.
- As the value of time increases, and in response to the "environment" (represented by random variables within the model) the stochastic process progresses from state to state.

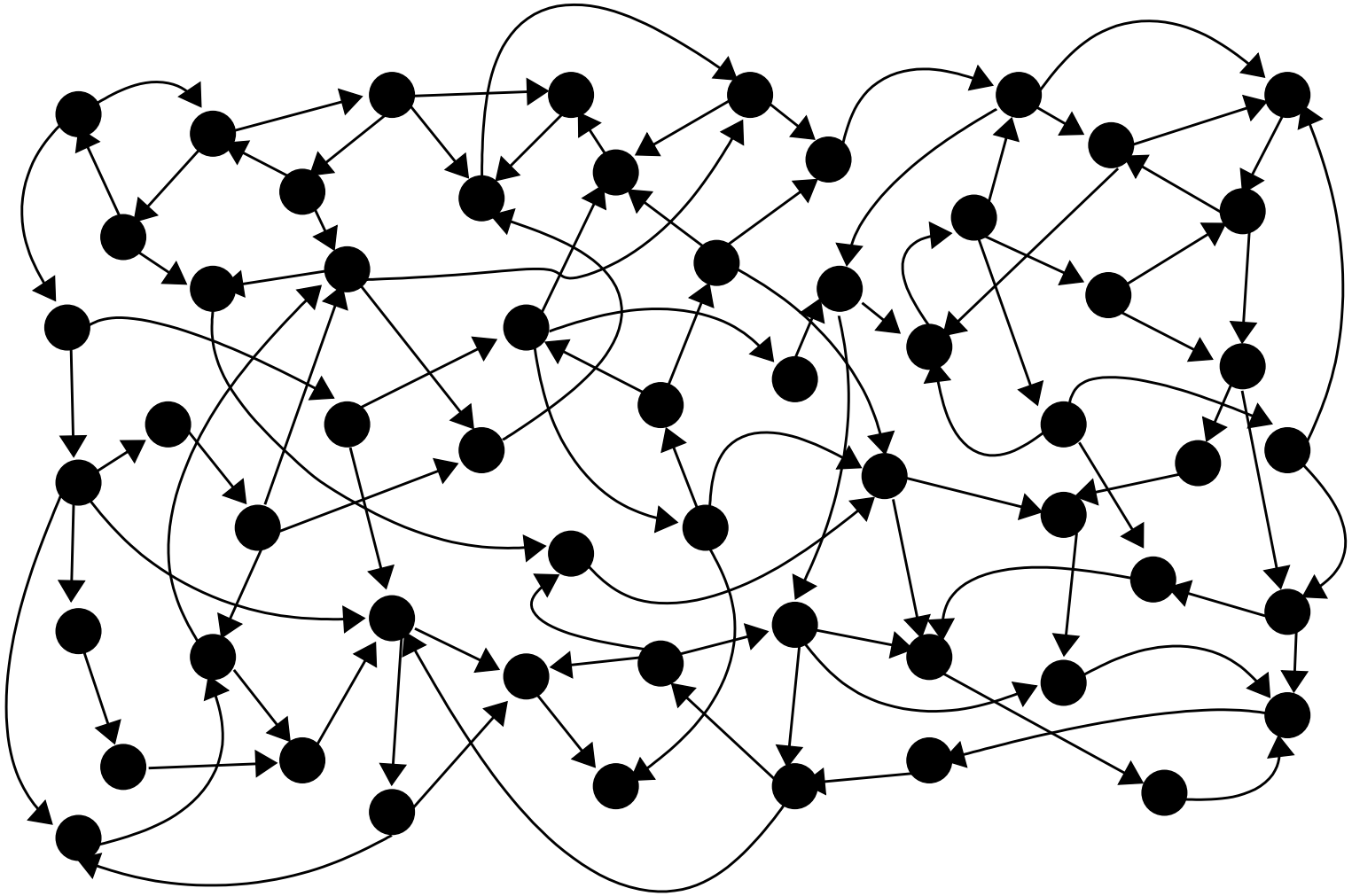
Assumptions

- We still assume that the system is characterised by a family of random variables $\{X(t), t \in T\}$.
- As the value of time increases, and in response to the "environment" (represented by random variables within the model) the stochastic process progresses from state to state.
- Any set of instances of $\{X(t), t \in T\}$ can be regarded as a path of a particle moving randomly in a state space, S, its position at time t being $X(t)$.

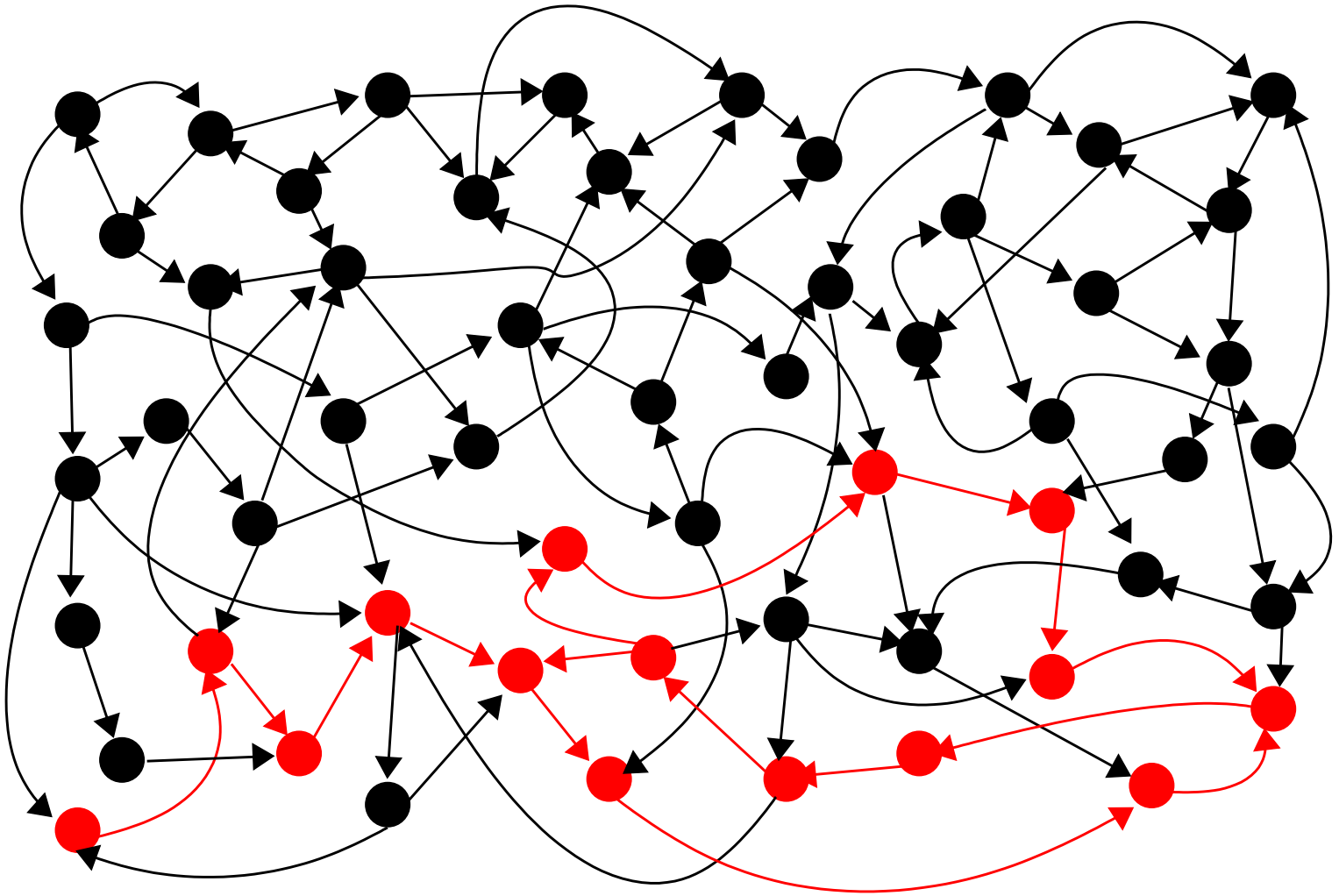
Assumptions

- We still assume that the system is characterised by a family of random variables $\{X(t), t \in T\}$.
- As the value of time increases, and in response to the "environment" (represented by random variables within the model) the stochastic process progresses from state to state.
- Any set of instances of $\{X(t), t \in T\}$ can be regarded as a path of a particle moving randomly in a state space, S, its position at time t being $X(t)$.
- These paths are called sample paths.

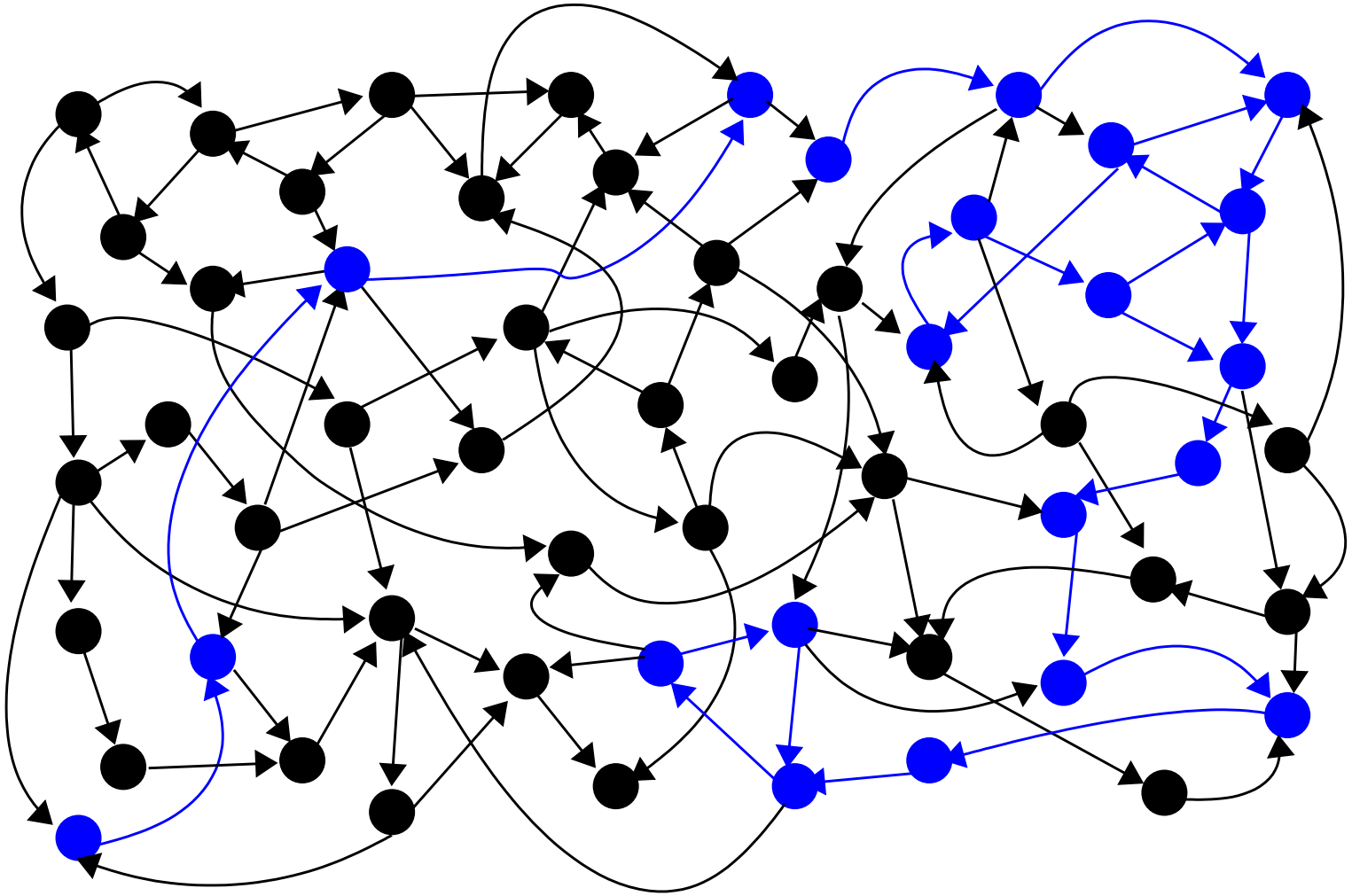
State space and sample paths



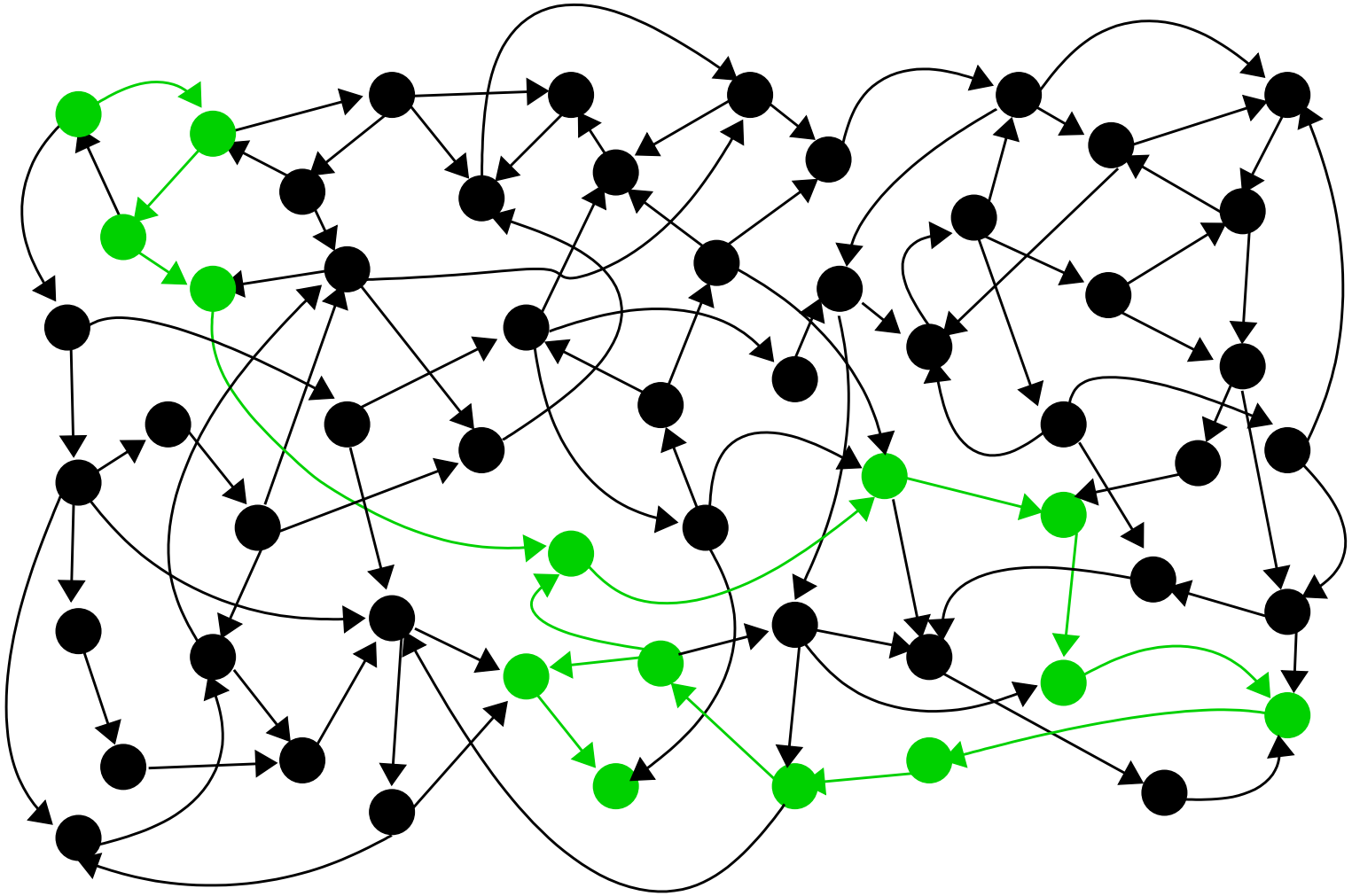
State space and sample paths



State space and sample paths



State space and sample paths



Sample paths and runs

- Using the analytic approach of Markov processes we characterised all possible sample paths by the global balance equations.

Sample paths and runs

- Using the analytic approach of Markov processes we characterised all possible sample paths by the global balance equations.
- Using simulation we investigate the sample paths directly.

Sample paths and runs

- Using the analytic approach of Markov processes we characterised all possible sample paths by the global balance equations.
- Using simulation we investigate the sample paths directly.
- We allow the model to trace out a sample path over the state space.

Sample paths and runs

- Using the analytic approach of Markov processes we characterised all possible sample paths by the global balance equations.
- Using simulation we investigate the sample paths directly.
- We allow the model to trace out a sample path over the state space.
- Each run of the simulation model will generate another, usually distinct, sample path.

Benefits of simulation

There are a variety of reasons why simulation may be preferable to analytical modelling:

Level of Abstraction

- Markovian modelling relies on many assumptions and abstractions which may not be appropriate for the system being studied.
- It may be unrealistic to assume that only one event can happen at any time, or that the inter-event times are all exponentially distributed.
- Simulation models allow us to represent a system at arbitrary levels of detail. This can also be a disadvantage since elaborate models take a long time to run and produce statistically significant output.

Benefits of simulation

There are a variety of reasons why simulation may be preferable to analytical modelling:

Transient Analysis

- In some cases we are not interested in the steady state behaviour of a system, but in its transient behaviour.
- Some systems never reach a steady state. Those that do usually have a "warm-up" period while the behaviour settles into the regular pattern which characterises steady state.
- The analytic solutions ignore this period since the global balance equations only capture the behaviour after steady state has been reached.
- A sample path derived from a simulation model will clearly represent transient behaviour in addition to steady state.

Benefits of simulation

There are a variety of reasons why simulation may be preferable to analytical modelling:

Size of State Space

- Generally solving a model analytically involves constructing and storing the complete state space of the model.
- For a Markov process with N states solving the global balance equations involves (at least) an $N \times N$ matrix ($\mathbf{Q}$) and a vector with N elements (π).
- When N becomes very large this becomes infeasible.
- In contrast, in a simulation model the state space is generated "on-the-fly" by the model itself during execution so it does not need to be all stored at once.

Constructing simulation models

- Simulation models are complex computer programs. They can be programmed directly in any programming language but there are distinct advantages to using a package specifically designed for simulation.

Constructing simulation models

- Simulation models are complex computer programs. They can be programmed directly in any programming language but there are distinct advantages to using a package specifically designed for simulation.
- Simulation packages such as Stochastic Simulation in Java (SSJ) provide facilities for many of the routine features of a simulation model. These features are common to all models, regardless of the system being represented.

Constructing simulation models

- Simulation models are complex computer programs. They can be programmed directly in any programming language but there are distinct advantages to using a package specifically designed for simulation.
- Simulation packages such as Stochastic Simulation in Java (SSJ) provide facilities for many of the routine features of a simulation model. These features are common to all models, regardless of the system being represented.
- This allows the performance analyst to concentrate on the issues specific to the system being modelled and to not worry about issues which are general to all simulations.

Simulation management

Some of the common features of simulation management are listed below.

- Event scheduler
- Simulation clock and time management
- System state variables
- Event routines
- Random number/random variate generator
- Report generator
- Trace routines
- Dynamic memory management

Event scheduler

An event scheduler keeps track of the events which are waiting to happen, usually as a linked list, and allows them to be manipulated in various ways. For example,

- schedule event E at time T;
- hold event E for a time interval ∂t;
- cancel a previously scheduled event E;
- hold event E indefinitely (until it is scheduled by another event);
- schedule an indefinitely held event.

Event scheduler

An event scheduler keeps track of the events which are waiting to happen, usually as a linked list, and allows them to be manipulated in various ways. For example,

- schedule event E at time T;
- hold event E for a time interval ∂t;
- cancel a previously scheduled event E;
- hold event E indefinitely (until it is scheduled by another event);
- schedule an indefinitely held event.

Event scheduler must be efficient

The event scheduler is called before every event, and it may be called several times during one event to schedule other new events.

Simulation clock and time management

- Every simulation model must have a global variable representing the simulated time.
- The event scheduler is usually responsible for advancing this time, either one unit at a time or, more commonly, directly to the time of the next scheduled event.
- This latter approach is called event-driven time management.

System state variables

- Since a simulation model generates a random walk over the state space of the system it is essential that the model has variables to capture the state of the system at each step.
- If a simulation run is stopped in the middle, it can be restarted later if, and only if, the values of all state variables are known.

Event routines

- Each event in the system brings about a state change.
- In the simulation model the effect of each event must be represented in a way which updates the system state variables, and in some cases, schedules other events.
- How the event routines are generated will depend on the simulation modelling paradigm used to construct the model.

Random number/random variate generator

- Random numbers play a crucial role in most discrete event simulations.
- A random number generator is used to generate a sequence of random values between 0 and 1.
- These values are then transformed to produce a sequence of random values which satisfy the desired distribution. This second step is sometimes called random variate generation.

Random number/random variate generator

- Random numbers play a crucial role in most discrete event simulations.
- A random number generator is used to generate a sequence of random values between 0 and 1.
- These values are then transformed to produce a sequence of random values which satisfy the desired distribution. This second step is sometimes called random variate generation.

Example

The impact of the environment on the system, e.g. inter-arrival times, is usually represented by random variables of some specified distribution.

Report generator

- Performance measures are derived from a simulation run by observing the values of parameters of interest during the execution.
- Most simulation modelling languages and packages contain built-in routines to calculate statistics from these observations and generate a report when the run is completed.

Trace routines

- A trace of the system can be a useful tool for debugging (sometimes called verifying) and validating a model.
- It is a time-ordered list of events, state variable values or output parameter values.
- Most simulation languages provide routines to generate traces which can be switched on or off in a particular run of the model.

Trace routines

- A trace of the system can be a useful tool for debugging (sometimes called verifying) and validating a model.
- It is a time-ordered list of events, state variable values or output parameter values.
- Most simulation languages provide routines to generate traces which can be switched on or off in a particular run of the model.

Note

Since trace generation is usually very inefficient it is generally only used during model development.

Dynamic memory management

- The number of active entities during the execution of a simulation model will vary continuously as new entities are created and old ones become obsolete.
- Most simulation languages provide automatic garbage collection to remove obsolete entities.

Approaches to simulation

- There are a number of approaches to discrete event simulation; the two most commonly used are event based modelling and process based modelling.

Approaches to simulation

- There are a number of approaches to discrete event simulation; the two most commonly used are event based modelling and process based modelling.
- Note that the high level modelling paradigms which we have already considered in the course—SPN, GSPN, PEPA and queueing networks—can be used to generate simulation models as well as Markov processes.

Event-based Simulation

- Event-based simulation focusses the modeller's attention on the individual events which can occur within the system.

Event-based Simulation

- Event-based simulation focusses the modeller's attention on the individual events which can occur within the system.
- An event within the system may generate several actions in the model—these are grouped together in an event routine.

Event-based Simulation

- Event-based simulation focusses the modeller's attention on the individual events which can occur within the system.
- An event within the system may generate several actions in the model—these are grouped together in an event routine.
- The event scheduler maintains a pointer to the appropriate event routine, and this is executed when the event reaches the head of the event list.

Example: A simple queue

- A good example of a system suited to event-based simulation would be a simple queue with deterministic arrival times, deterministic service times and customer defections at fixed times.

Example: A simple queue

- A good example of a system suited to event-based simulation would be a simple queue with deterministic arrival times, deterministic service times and customer defections at fixed times.
- The events are a customer arrives, A; a customer defects, D; a customer begins service, B; and a customer ends service, E.

Example: A simple queue

- A good example of a system suited to event-based simulation would be a simple queue with deterministic arrival times, deterministic service times and customer defections at fixed times.
- The events are a customer arrives, A; a customer defects, D; a customer begins service, B; and a customer ends service, E.
- At each event time as well as the processing of the event to represent the behaviour of the system, some processing internal to the model might be done.

Example: A simple queue

- A good example of a system suited to event-based simulation would be a simple queue with deterministic arrival times, deterministic service times and customer defections at fixed times.
- The events are a customer arrives, A; a customer defects, D; a customer begins service, B; and a customer ends service, E.
- At each event time as well as the processing of the event to represent the behaviour of the system, some processing internal to the model might be done.
- This can be event tracing or statistical collection. For example, calculating the average queue length and throughput.

Events: a customer arrives, A

An event A at time t

- add one to the state variable representing queue length, and record the time at which the change occurred,
- schedule an event D at the time $t + d$, where d is the length of time a customer will wait without defecting,
- schedule an event B to occur as soon as possible, depending on the availability of the server,
- schedule another event A at time $t + a$ where a is the inter-arrival time.

Events: a customer defects, D

An event D at time $t + d$

- decrease the queue length by one and the record the time at which the change occurred,
- de-schedule event B on hold since time t.

Events: a customer begins service, B

An event B at time $t + w$ $(w < d)$

- decrease the queue length by one and record the time at which the change occurred,
- de-schedule the event D at time $t + d$,
- schedule an event E at time $t + w + s$, where s is the service time.

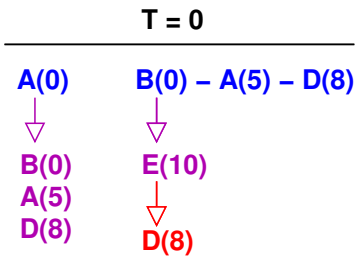
Events: a customer ends service, E

An event E at time $t + w + s$

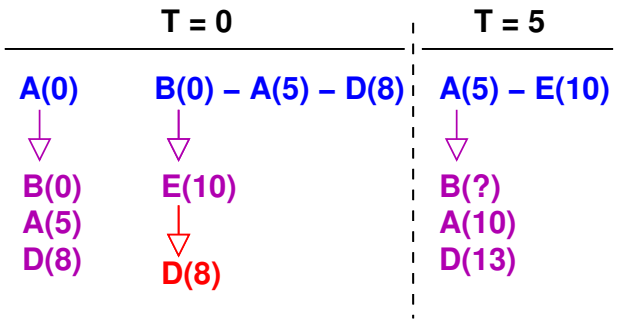
- increment the busy time of the service centre by s,
- add one to the total number of customers served,
- activate the first event B waiting in the event list.

Illustration

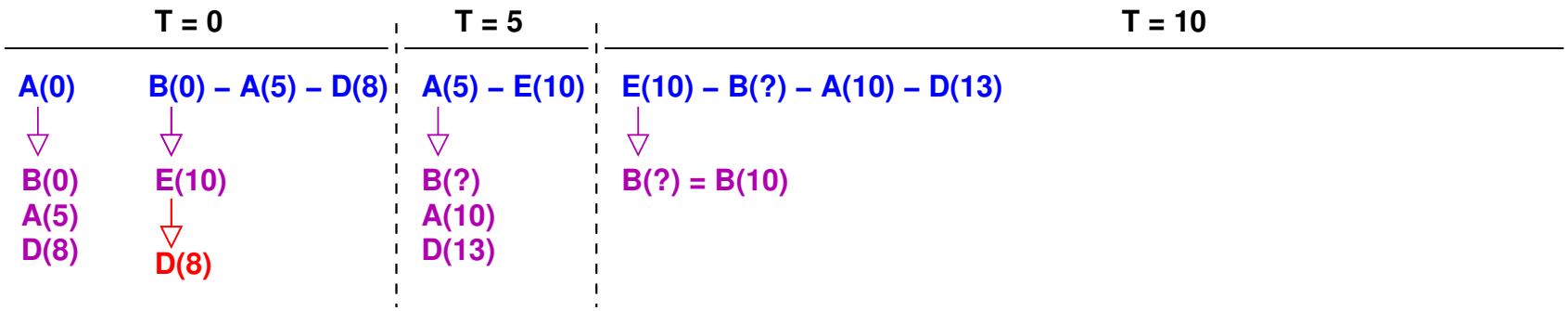
Illustration



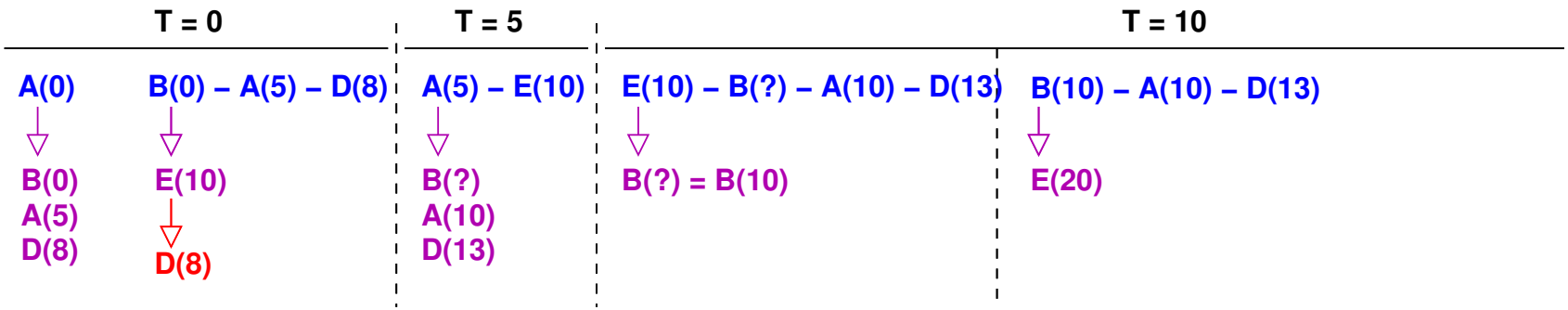
Illustration



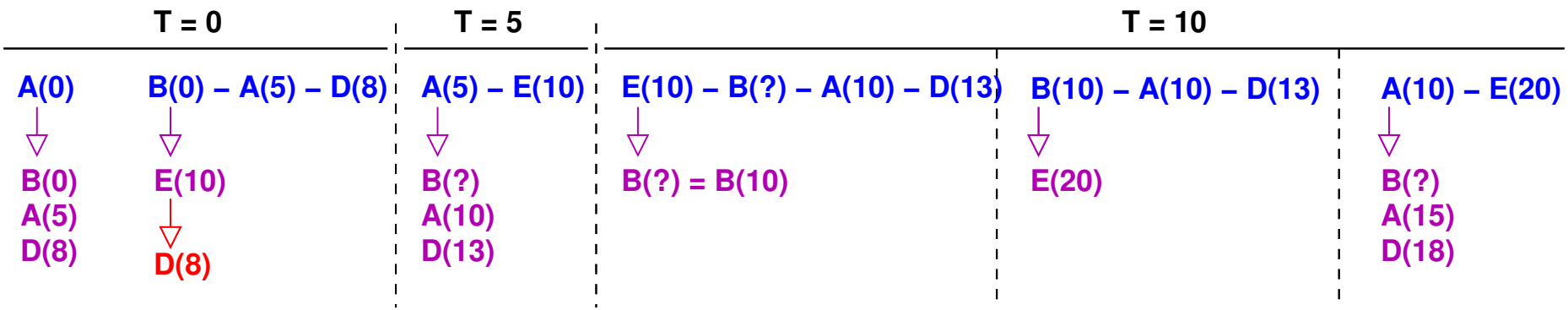
Illustration



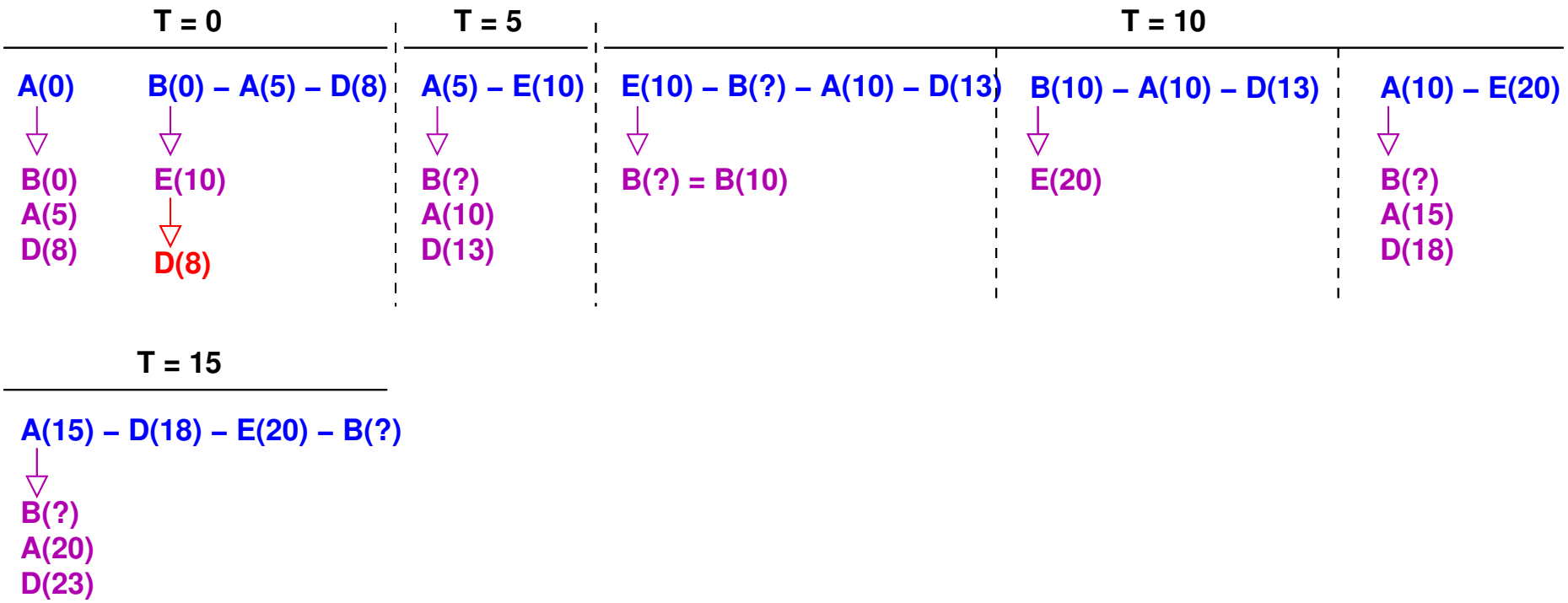
Illustration



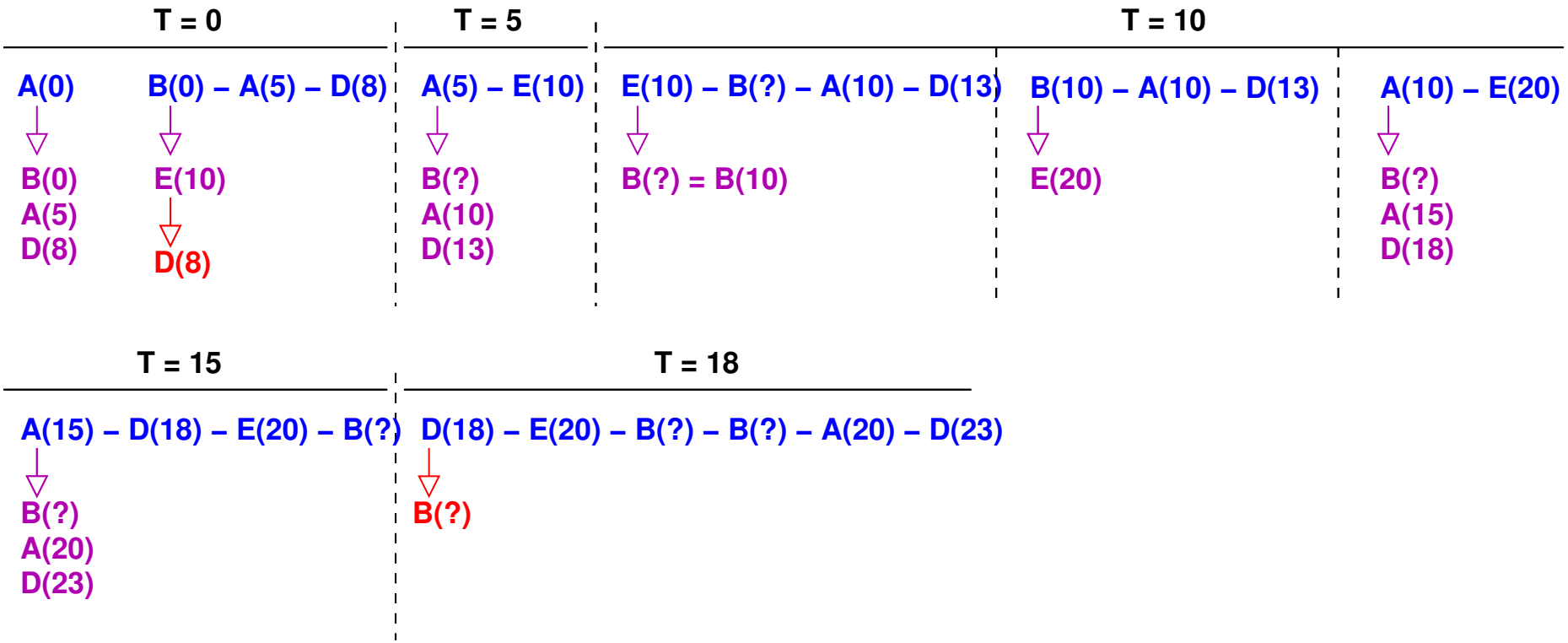
Illustration



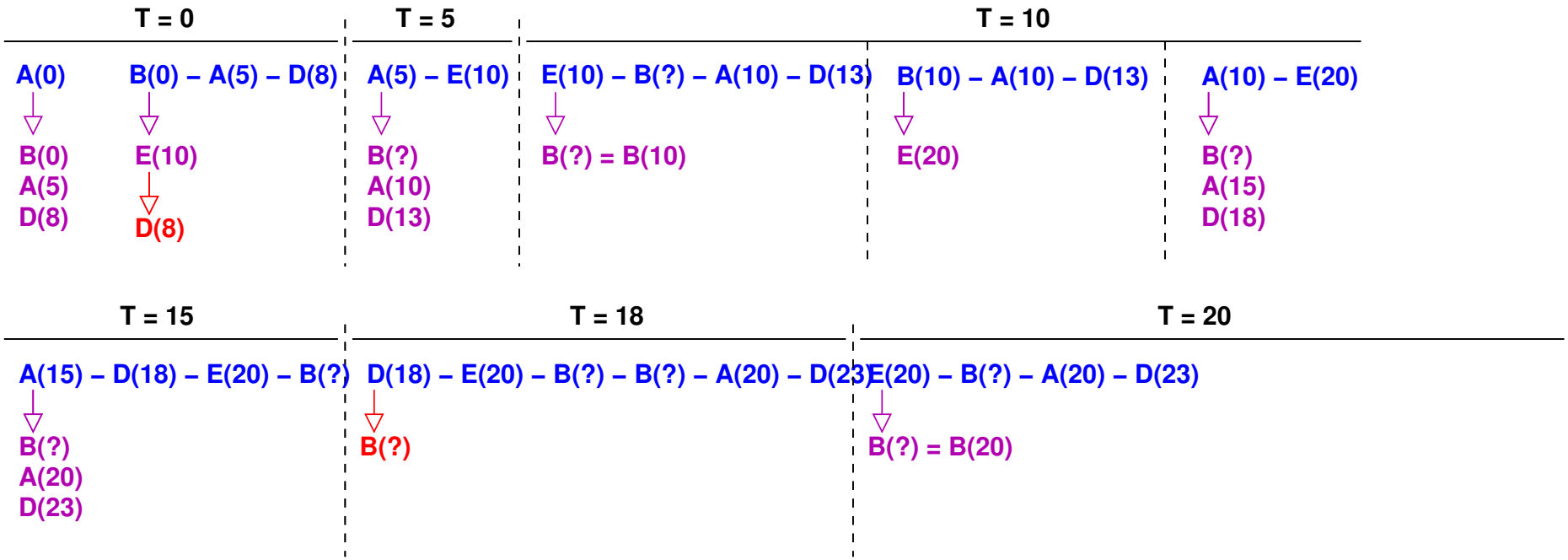
Illustration



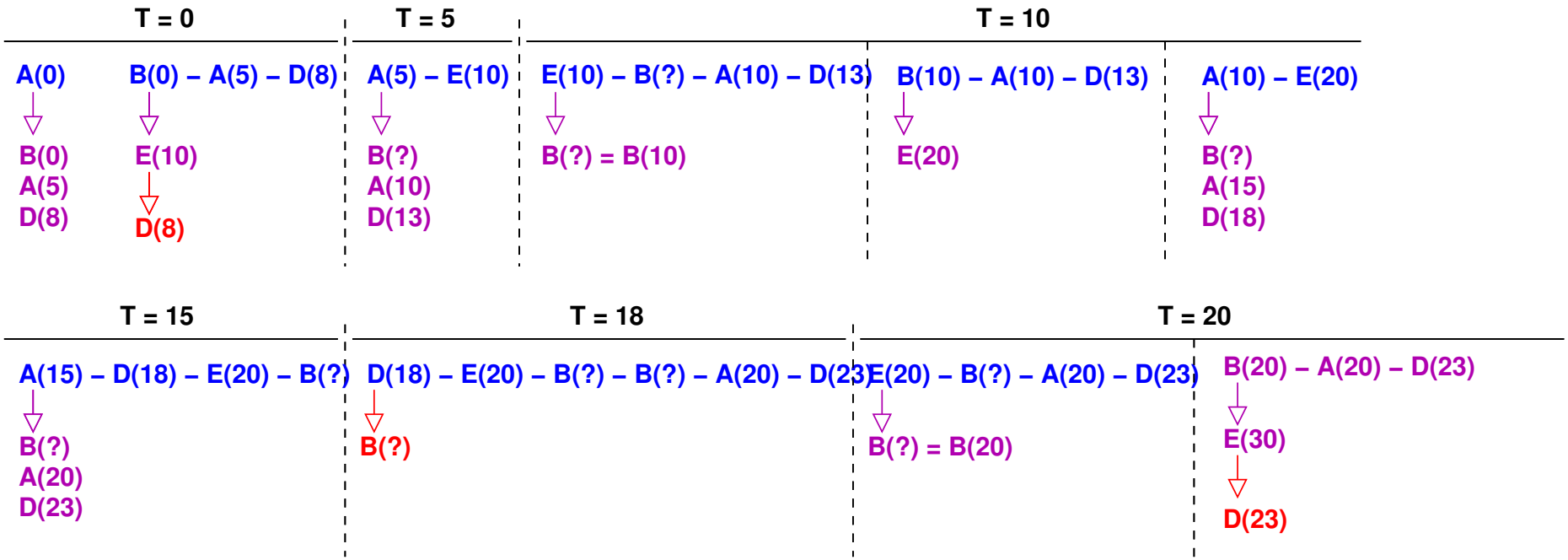
Illustration



Illustration



Illustration



Process-based Simulation

- The process-based approach to simulation modelling collects events together into related sequences which are ordered by time.

Process-based Simulation

- The process-based approach to simulation modelling collects events together into related sequences which are ordered by time.
- These sequences are related in the sense that they all involve the same entity within the system; they are termed processes.

Process-based Simulation

- The process-based approach to simulation modelling collects events together into related sequences which are ordered by time.
- These sequences are related in the sense that they all involve the same entity within the system; they are termed processes.
- For example, above we could consider each customer to be a process within the system, since it generates a sequence of related events, and track its progress through the queue.

Processes and life-cycles

- The process-based approach views a system as a web of concurrent, interacting entities—the processes.

Processes and life-cycles

- The process-based approach views a system as a web of concurrent, interacting entities—the processes.
- All the actions associated with an entity's behaviour are grouped together to form a life-cycle for entities of that type.

Processes and life-cycles

- The process-based approach views a system as a web of concurrent, interacting entities—the processes.
- All the actions associated with an entity's behaviour are grouped together to form a life-cycle for entities of that type.
- The event scheduler still maintains a list of scheduled events centrally but this will now generally be in the form of a pointer to a process/object.

Processes and life-cycles

- The process-based approach views a system as a web of concurrent, interacting entities—the processes.
- All the actions associated with an entity's behaviour are grouped together to form a life-cycle for entities of that type.
- The event scheduler still maintains a list of scheduled events centrally but this will now generally be in the form of a pointer to a process/object.
- The process will maintain a record of its current state and which action it should perform when next scheduled.

Processes and life-cycles

- The process-based approach views a system as a web of concurrent, interacting entities—the processes.
- All the actions associated with an entity's behaviour are grouped together to form a life-cycle for entities of that type.
- The event scheduler still maintains a list of scheduled events centrally but this will now generally be in the form of a pointer to a process/object.
- The process will maintain a record of its current state and which action it should perform when next scheduled.
- This style of modelling maps well to object-oriented programming: a class is associated with each type of entity; objects then represent instances of the entity.

Example: A simple queue

- For the example of a queue with deterministic inter-arrival and service times (but no defections), we could define a class to represent the arrival process.

Example: A simple queue

- For the example of a queue with deterministic inter-arrival and service times (but no defections), we could define a class to represent the arrival process.
- This class (`Source`) would generate the event representing a customer entering the queue and then delay until the arrival time of the next customer.

Example: A simple queue

- For the example of a queue with deterministic inter-arrival and service times (but no defections), we could define a class to represent the arrival process.
- This class (`Source`) would generate the event representing a customer entering the queue and then delay until the arrival time of the next customer.
- A second class would represent the server (`Server`).

Example: A simple queue

- For the example of a queue with deterministic inter-arrival and service times (but no defections), we could define a class to represent the arrival process.
- This class (`Source`) would generate the event representing a customer entering the queue and then delay until the arrival time of the next customer.
- A second class would represent the server (`Server`).
- This is passive in the sense that it first waits to be notified of an event (the arrival of a customer) and then represents the service of the customer as a delay.

Common mistakes in simulation studies

- **Inappropriate level of detail**
  Because arbitrary levels of detail are possible it is sometimes tempting to represent too much in the model. This will have a cost in terms of execution time.

Common mistakes in simulation studies

- **Inappropriate level of detail**
  Because arbitrary levels of detail are possible it is sometimes tempting to represent too much in the model. This will have a cost in terms of execution time.
- **Unverified models**
  Simulation models are complex programs and as such are prone to bugs in the same way that any complex program is. Verification is intended to make sure that the model behaves as it was intended to.

Common mistakes in simulation studies

- **Inappropriate level of detail**
  Because arbitrary levels of detail are possible it is sometimes tempting to represent too much in the model. This will have a cost in terms of execution time.
- **Unverified models**
  Simulation models are complex programs and as such are prone to bugs in the same way that any complex program is. Verification is intended to make sure that the model behaves as it was intended to.
- **Invalid models**
  Validation is needed ensure that the model is a good representation of the system. A model may be bug-free but still be incorrect in the sense that it is based on invalid assumptions.

Common mistakes in simulation studies

- Too short simulation runs

  A large model must be executed for a long (simulated) time to ensure that the sample path which is generated is statistically valid.

Common mistakes in simulation studies

- Too short simulation runs
  A large model must be executed for a long (simulated) time to ensure that the sample path which is generated is statistically valid.
- Single simulation runs
  Each run of the simulation represents only one sample path based on a particular sequence of random numbers. In order for results to be statistically valid they should be based on several sample paths obtained using different sequences.

Common mistakes in simulation studies

- **Too short simulation runs**
  A large model must be executed for a long (simulated) time to ensure that the sample path which is generated is statistically valid.
- **Single simulation runs**
  Each run of the simulation represents only one sample path based on a particular sequence of random numbers. In order for results to be statistically valid they should be based on several sample paths obtained using different sequences.
- **Poor random-number generators**
  Random number generators are used extensively in simulation models. A poor random-number generator may introduce correlation and/or bias into the value of those random variables.