Stochastic Simulation

Jan-Pieter Dorsman[1] & Michel Mandjes[1,2,3]

[1]Korteweg-de Vries Institute for Mathematics, University of Amsterdam

[2]CWI, Amsterdam

[3]Eurandom, Eindhoven

University of Amsterdam,
Fall, 2017

Intermezzo
DISCRETE-EVENT SIMULATION

This course so far

By now, you have (hopefully!) seen

- why simulation encompasses a necessary set of techniques in applied probability.
- how your computer draws uniform random numbers... and why you should not try to come up with your own random number generator.
- several cool techniques to transform these to obtain random numbers for other distributions.

So, we are pretty much done right now?

So, we are pretty much done right now?

Negative, we just got started! Recall all of the issues Michel pointed out in the first lecture.

Today we mainly focus on the issue what programming techniques to use to create simulation programs.

Let's see what happens when, for example, we try to simulate a check-out queue at a small super market (G/G/1 queue).

Simulation of a G/G/1 queue

The G/G/1 super market queue:

- G: Customers arrive at the check-out counter according to an arbitrary renewal process.

Simulation of a G/G/1 queue

The G/G/1 super market queue:

- G: Customers arrive at the check-out counter according to an arbitrary renewal process.
- G: The service times all customer require are independent and identically generally distributed.

Simulation of a G/G/1 queue

The G/G/1 super market queue:

- G: Customers arrive at the check-out counter according to an arbitrary renewal process.
- G: The service times all customer require are independent and identically generally distributed.
- 1: Just one check-out counter available.

To date, hardly anything is known about performance measures of this model.

Simulation of a G/G/1 queue

The G/G/1 super market queue:

- G: Customers arrive at the check-out counter according to an arbitrary renewal process.
- G: The service times all customer require are independent and identically generally distributed.
- 1: Just one check-out counter available.

To date, hardly anything is known about performance measures of this model.

Let's suppose we want to estimate $\mathbb{E}[W]$, the average waiting time of customers in a stationary system.

Lindley's recurrence relation

Some notation:

- A_j: the interarrival time between the j-th and the $j+1$-st customer
- B_j: the service time that the j-th customer requires
- W_j: the waiting time incurred by the j-th customer

Lindley's recurrence relation

Some notation:

- A_j: the interarrival time between the j-th and the $j+1$-st customer
- B_j: the service time that the j-th customer requires
- W_j: the waiting time incurred by the j-th customer

Lindley's recurrence relation (see the board):

$$W_{j+1} = (W_j + B_j - A_j)^+$$

with $W_1 = 0$.

Simulation method for a G/G/1 queue

Lindley's recurrence relation:

$$W_{j+1} = (W_j + B_j - A_j)^+$$

with $W_1 = 0$.

So, apparently, using the methods we learned, we can

1. sample a whole bunch of A_n's and B_n's
2. compute the W_n's using the recurrence relation,
3. compute $\frac{1}{n}\sum_{j=1}^{n} W_j$ for large n, which is an estimate of $\mathbb{E}[W]$.

Simulation of a G/G/1 queue

Suppose

1. interarrival-times are i.i.d. standard uniformly distributed, and
2. service times are i.i.d. uniformly distributed on [0,0.5].

Then the Mathematica code for $n = 1.000.000$ would look something like this:

```
SeedRandom[123]; n = 1000000;

A = Table[RandomReal[], {n}];
B = Table[RandomReal[]/2, {n}];

W[1] = 0;
W[j_] := W[j] = Max[W[j - 1] + B[[j - 1]] - A[[j - 1]] , 0];

Sum[W[j], {j, 1, n}]/n
```

Mathematica spits out the estimate of an expected 0.0748002 units of waiting time in no time.

Lindley's recurrence relations are nice, but think of all the follow-up questions you could ask me:

Lindley's recurrence relations are nice, but think of all the follow-up questions you could ask me:

Q: **W**hat if we want to simulate $\mathbb{P}(W < t)$ instead of $\mathbb{E}[W]$?

Lindley's recurrence relations are nice, but think of all the follow-up questions you could ask me:

Q: **W**hat if we want to simulate $\mathbb{P}(W < t)$ instead of $\mathbb{E}[W]$?

A: Compute $\sum_{j=1}^{n} \frac{1\{W_n < t\}}{n}$.

Lindley's recurrence relations are nice, but think of all the follow-up questions you could ask me:

Q: **W**hat if we want to simulate $\mathbb{P}(W < t)$ instead of $\mathbb{E}[W]$?

A: Compute $\sum_{j=1}^{n} \frac{1\{W_n < t\}}{n}$.

Q: **W**hat if we want to know $\mathbb{E}[L]$, the average number of customers in the system?

Lindley's recurrence relations are nice, but think of all the follow-up questions you could ask me:

Q: **W**hat if we want to simulate $\mathbb{P}(W < t)$ instead of $\mathbb{E}[W]$?

A: Compute $\sum_{j=1}^{n} \frac{1\{W_n < t\}}{n}$.

Q: **W**hat if we want to know $\mathbb{E}[L]$, the average number of customers in the system?

A: We can use Little's law: $\mathbb{E}[W] = \mathbb{E}[L]\mathbb{E}[A]$.

Lindley's recurrence relations are nice, but think of all the follow-up questions you could ask me:

Q: **W**hat if we want to simulate $\mathbb{P}(W < t)$ instead of $\mathbb{E}[W]$?

A: Compute $\sum_{j=1}^{n} \frac{1\{W_n < t\}}{n}$.

Q: **W**hat if we want to know $\mathbb{E}[L]$, the average number of customers in the system?

A: We can use Little's law: $\mathbb{E}[W] = \mathbb{E}[L]\mathbb{E}[A]$.

Q: **W**hat if we introduce additional servers?

Lindley's recurrence relations are nice, but think of all the follow-up questions you could ask me:

Q: **W**hat if we want to simulate $\mathbb{P}(W < t)$ instead of $\mathbb{E}[W]$?

A: Compute $\sum_{j=1}^{n} \frac{1\{W_n < t\}}{n}$.

Q: **W**hat if we want to know $\mathbb{E}[L]$, the average number of customers in the system?

A: We can use Little's law: $\mathbb{E}[W] = \mathbb{E}[L]\mathbb{E}[A]$.

Q: **W**hat if we introduce additional servers?

A: Beats me! Well actually, Kiefer & Wolfowitz (1959) extended Lindley's recurrence relation to multiple servers, but it's computationally demanding.

Lindley's recurrence relations are nice, but think of all the follow-up questions you could ask me:

Q: **W**hat if we want to simulate $\mathbb{P}(W < t)$ instead of $\mathbb{E}[W]$?

A: Compute $\sum_{j=1}^{n} \frac{1\{W_n < t\}}{n}$.

Q: **W**hat if we want to know $\mathbb{E}[L]$, the average number of customers in the system?

A: We can use Little's law: $\mathbb{E}[W] = \mathbb{E}[L]\mathbb{E}[A]$.

Q: **W**hat if we introduce additional servers?

A: Beats me! Well actually, Kiefer & Wolfowitz (1959) extended Lindley's recurrence relation to multiple servers, but it's computationally demanding.

Q: **W**hat if for the some reason the customers are served LIFO instead of FIFO?

Lindley's recurrence relations are nice, but think of all the follow-up questions you could ask me:

Q: **W**hat if we want to simulate $\mathbb{P}(W < t)$ instead of $\mathbb{E}[W]$?

A: Compute $\sum_{j=1}^{n} \frac{1\{W_n < t\}}{n}$.

Q: **W**hat if we want to know $\mathbb{E}[L]$, the average number of customers in the system?

A: We can use Little's law: $\mathbb{E}[W] = \mathbb{E}[L]\mathbb{E}[A]$.

Q: **W**hat if we introduce additional servers?

A: Beats me! Well actually, Kiefer & Wolfowitz (1959) extended Lindley's recurrence relation to multiple servers, but it's computationally demanding.

Q: **W**hat if for the some reason the customers are served LIFO instead of FIFO?

A: Okay, you got me.

Lindley's recurrence relations are nice, but think of all the follow-up questions you could ask me:

Q: **W**hat if we want to simulate $\mathbb{P}(W < t)$ instead of $\mathbb{E}[W]$?

A: Compute $\sum_{j=1}^{n} \frac{1\{W_n < t\}}{n}$.

Q: **W**hat if we want to know $\mathbb{E}[L]$, the average number of customers in the system?

A: We can use Little's law: $\mathbb{E}[W] = \mathbb{E}[L]\mathbb{E}[A]$.

Q: **W**hat if we introduce additional servers?

A: Beats me! Well actually, Kiefer & Wolfowitz (1959) extended Lindley's recurrence relation to multiple servers, but it's computationally demanding.

Q: **W**hat if for the some reason the customers are served LIFO instead of FIFO?

A: Okay, you got me.

Q: What if we want to know $\mathrm{Var}[L]$?

Lindley's recurrence relations are nice, but think of all the follow-up questions you could ask me:

Q: **W**hat if we want to simulate $\mathbb{P}(W < t)$ instead of $\mathbb{E}[W]$?

A: Compute $\sum_{j=1}^{n} \frac{1\{W_n < t\}}{n}$.

Q: **W**hat if we want to know $\mathbb{E}[L]$, the average number of customers in the system?

A: We can use Little's law: $\mathbb{E}[W] = \mathbb{E}[L]\mathbb{E}[A]$.

Q: **W**hat if we introduce additional servers?

A: Beats me! Well actually, Kiefer & Wolfowitz (1959) extended Lindley's recurrence relation to multiple servers, but it's computationally demanding.

Q: **W**hat if for the some reason the customers are served LIFO instead of FIFO?

A: Okay, you got me.

Q: What if we want to know $\mathrm{Var}[L]$?

A: Alright, alright, I yield...

Today's topic

Sometimes, we can immediately map our uniform samples directly to performance measures. For the G/G/1 case this was possible using Lindley's recurrence relation.

What if we don't have such a device?

Today's topic

Sometimes, we can immediately map our uniform samples directly to performance measures. For the G/G/1 case this was possible using Lindley's recurrence relation.

What if we don't have such a device?

Answer: we use discrete-event simulation to get from uniform samples to performance measures.

I'll be basing myself on a hand-out stemmed on Chapter 10 of the Dutch book 'Operationale analyse' by Prof. dr. H. C. Tijms.

Key ingredients

Discrete event simulation is a general technique, built around the idea of 'discrete events' that has been developed to help one follow a model over time and determine the relevant quantities of interest.

Key ingredients

Discrete event simulation is a general technique, built around the idea of 'discrete events' that has been developed to help one follow a model over time and determine the relevant quantities of interest.

It generally consists of a number of important components:

1. *A simulation 'clock' t*: this is a variable that keeps track of how much time has elapsed since the start of the system.

Key ingredients

Discrete event simulation is a general technique, built around the idea of 'discrete events' that has been developed to help one follow a model over time and determine the relevant quantities of interest.

It generally consists of a number of important components:

1. *A simulation 'clock' t*: this is a variable that keeps track of how much time has elapsed since the start of the system.
2. *System state variable*: a variable that keeps information on the current state of the system. In G/G/1 example: the number of customers in the system, let's call it I.

Key ingredients

Discrete event simulation is a general technique, built around the idea of 'discrete events' that has been developed to help one follow a model over time and determine the relevant quantities of interest.

It generally consists of a number of important components:

1. *A simulation 'clock' t*: this is a variable that keeps track of how much time has elapsed since the start of the system.
2. *System state variable*: a variable that keeps information on the current state of the system. In G/G/1 example: the number of customers in the system, let's call it l.
3. *Statistical counter variables*: these are variables that are kept to compute quantities of interest later on. In G/G/1 example: e.g. number of served customers n and their total waiting time w.

Key ingredients

Discrete event simulation is a general technique, built around the idea of 'discrete events' that has been developed to help one follow a model over time and determine the relevant quantities of interest.

It generally consists of a number of important components:

1. *A simulation 'clock' t*: this is a variable that keeps track of how much time has elapsed since the start of the system.
2. *System state variable*: a variable that keeps information on the current state of the system. In G/G/1 example: the number of customers in the system, let's call it l.
3. *Statistical counter variables*: these are variables that are kept to compute quantities of interest later on. In G/G/1 example: e.g. number of served customers n and their total waiting time w.
4. *Events*: these are the things that can occur to the system over time. Every time an event occurs, the above three variables are updated. In earlier example: arrivals and departures of customers

General idea for the G/G/1 example

In a nutshell, the idea of a discrete event simulation program for the G/G/1 example would be to discretely cycle through all events as follows:

- Step 1: Plan the first event. Starting with an empty system, the first event will be an arrival of the customer. This will occur at time A_1, where A_1 is the sample of the interarrival-time distribution (use uniform sample). The event time will be A_1.

General idea for the G/G/1 example

In a nutshell, the idea of a discrete event simulation program for the G/G/1 example would be to discretely cycle through all events as follows:

- Step 1: Plan the first event. Starting with an empty system, the first event will be an arrival of the customer. This will occur at time A_1, where A_1 is the sample of the interarrival-time distribution (use uniform sample). The event time will be A_1.
- Step 2: Handle the earliest upcoming planned event:
  1. Set t to be equal to the time of this event.

General idea for the G/G/1 example

In a nutshell, the idea of a discrete event simulation program for the G/G/1 example would be to discretely cycle through all events as follows:

- Step 1: Plan the first event. Starting with an empty system, the first event will be an arrival of the customer. This will occur at time A_1, where A_1 is the sample of the interarrival-time distribution (use uniform sample). The event time will be A_1.
- Step 2: Handle the earliest upcoming planned event:
  1. Set t to be equal to the time of this event.
  2. *In case the event is an arrival:*
     - Plan the next arrival event at time $t + A_i$, with A_i a sample of the interarrival-time distribution.
     - If $L = 0$, plan the next departure event at time $t + B_i$, with B_i a sample of the service time distribution.
     - $I := I + 1$

General idea for the G/G/1 example

In a nutshell, the idea of a discrete event simulation program for the G/G/1 example would be to discretely cycle through all events as follows:

- Step 1: Plan the first event. Starting with an empty system, the first event will be an arrival of the customer. This will occur at time A_1, where A_1 is the sample of the interarrival-time distribution (use uniform sample). The event time will be A_1.
- Step 2: Handle the earliest upcoming planned event:
  1. Set t to be equal to the time of this event.
  2. *In case the event is an arrival:*
     - Plan the next arrival event at time $t + A_i$, with A_i a sample of the interarrival-time distribution.
     - If $L = 0$, plan the next departure event at time $t + B_i$, with B_i a sample of the service time distribution.
     - $l := l + 1$

     Otherwise, if the event is a departure:
     - $l := l - 1$
     - If $l > 0$, then plan the next departure event at time $T + B_i$, with B_i a sample of the service time distribution.
     - $n := n + 1$. We also increase w with the departed customer's waiting time, which is T minus its actual arrival time minus its service time.

General idea for the $G/G/1$ example cntd.

- Step 3: If t has not reached a certain number T, return to step 2. Otherwise go to step 4. (How large should T be chosen? Upcoming lecture!)

General idea for the $G/G/1$ example cntd.

- Step 3: If t has not reached a certain number T, return to step 2. Otherwise go to step 4. (How large should T be chosen? Upcoming lecture!)
- Step 4: Calculate the average of all stored waiting times. This is an estimate for $\mathbb{E}[W]$.

Often, when looking for a stationary waiting time, it is better not to include the first portion of the stored waiting times in the average. Why?

This is all very nice, but we'll need the computer to perform these steps.

How do we write a suitable computer program?

Let's see an example.

Simulation example: the bank

- A bank opens at 10AM. At 5PM it closes, but any remaining customers arriving prior to this time will still be served.
- There are c bank clerks, who work equally fast, and serve queued customers in the order of arrival.
- Customers arrive according to a Poisson arrival process with rate λ customers/minute.
- The service times that each of the customers require is Unif$[a, b]$ distributed.

For the queueing theorists among us: this is an M/G/c queue!

Problem statement:

Determine for given values of c, λ, a and b:

a) the average length of the queue during the day

b) the average waiting time of a customer during the day

c) the fraction of time that the clerk is busy during the day

The bank: system state variables

The system state variables (or simply states) should contain enough information about the history of the system, such that, given the current state, the previous states become irrelevant to predict the future behaviour of the system.

For the bank example, the following should be stored in the system state variables:

The bank: system state variables

The system state variables (or simply states) should contain enough information about the history of the system, such that, given the current state, the previous states become irrelevant to predict the future behaviour of the system.

For the bank example, the following should be stored in the system state variables:

1. The number of customers in the queue (to track average queue length)

The bank: system state variables

The system state variables (or simply states) should contain enough information about the history of the system, such that, given the current state, the previous states become irrelevant to predict the future behaviour of the system.

For the bank example, the following should be stored in the system state variables:

1. The number of customers in the queue (to track average queue length)
2. The arrival times of the customers in the queue (to track waiting times of customers)

The bank: system state variables

The system state variables (or simply states) should contain enough information about the history of the system, such that, given the current state, the previous states become irrelevant to predict the future behaviour of the system.

For the bank example, the following should be stored in the system state variables:

1. The number of customers in the queue (to track average queue length)
2. The arrival times of the customers in the queue (to track waiting times of customers)
3. The status of each of the bank clerks: busy/not busy (to track busy fraction of clerks)

The bank: events

The most important notion in discrete event simulation encompasses *events*, which can potentially the state of the model.

For discrete event simulation, it is essential that events can only occur at discrete points in time!

The bank: events

The most important notion in discrete event simulation encompasses *events*, which can potentially the state of the model.

For discrete event simulation, it is essential that events can only occur at discrete points in time!

The assumption that events can only occur *discretely* in time, enables us to compress the true scale of time into just *changing the simulation clock only at moments at which an event occurs*.

Time intervals in between events where the system state variables do not change are *irrelevant* and can be *skipped*.

For the bank example:

The bank: events

The most important notion in discrete event simulation encompasses *events*, which can potentially the state of the model.

For discrete event simulation, it is essential that events can only occur at discrete points in time!

The assumption that events can only occur *discretely* in time, enables us to compress the true scale of time into just *changing the simulation clock only at moments at which an event occurs*.

Time intervals in between events where the system state variables do not change are *irrelevant* and can be *skipped*.

For the bank example: the events are the arrivals and departures (i.e. end of service) of customers. Only at these time instants, the system state variables actually change!

The bank: statistical counters

The *statistical counter variables* are variables that store statistical information which can be used to compute the requested performance measures at the end of a simulation.

Also the statistical counter variables are updated only at times when an event occurs.

For the bank example, the statistical counter variables are:

The bank: statistical counters

The *statistical counter variables* are variables that store statistical information which can be used to compute the requested performance measures at the end of a simulation.

Also the statistical counter variables are updated only at times when an event occurs.

For the bank example, the statistical counter variables are:

1. The number of customers that arrived in the system so far.
2. The total waiting time of customers that have arrived in the system so far (the waiting time of a customer does not include the actual service time.
3. The total amount of time that a bank clerk has been busy so far.
4. The total surface under the graph of the number of customers in the queue so far (this number does not include the customers being served by a bank clerk).

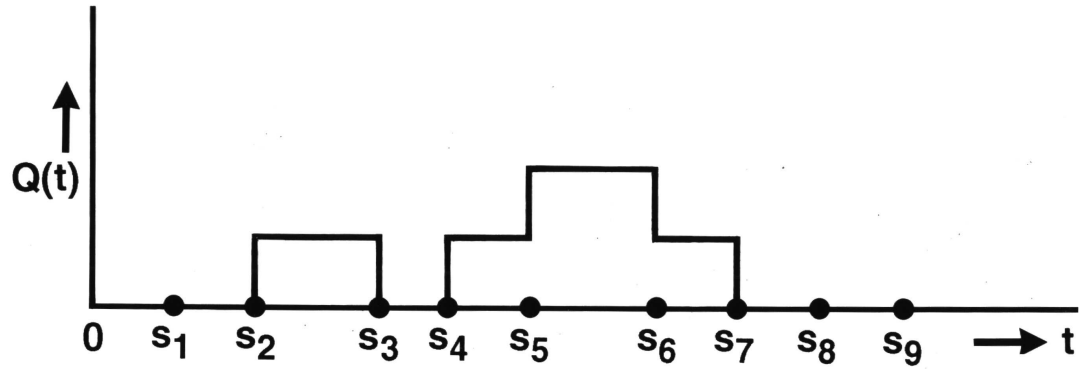
The last one may need some explanation...

The bank: statistical counters cntd.

Let $Q(t)$ be the number of customers in the queue at time t. This is a step function that only changes at event times.

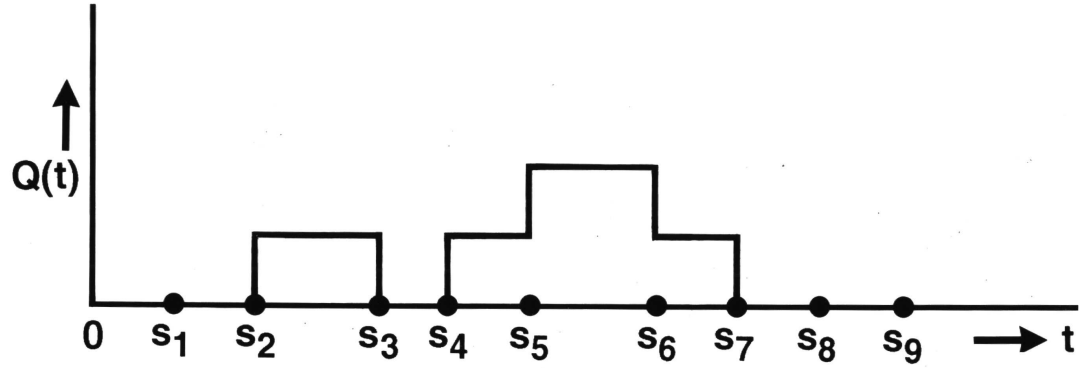
The bank: statistical counters cntd.

Let $Q(t)$ be the number of customers in the queue at time t. This is a step function that only changes at event times. For example:



The bank: statistical counters cntd.

Let $Q(t)$ be the number of customers in the queue at time t. This is a step function that only changes at event times. For example:



After a large number of events, the average number in the queue $\mathbb{E}[Q]$ after an event time t^* can be estimated as:

$$\mathbb{E}[Q] \approx \frac{1}{t^*} \int_0^{t^*} Q(t)dt.$$

To compute last integral: update the sum of the surfaces of rectangles under the graph after every event!

The bank: towards a computer program

Idea of a discrete event simulation program: cycle through all events, at which point system states and statistical counter variables will be adjusted.

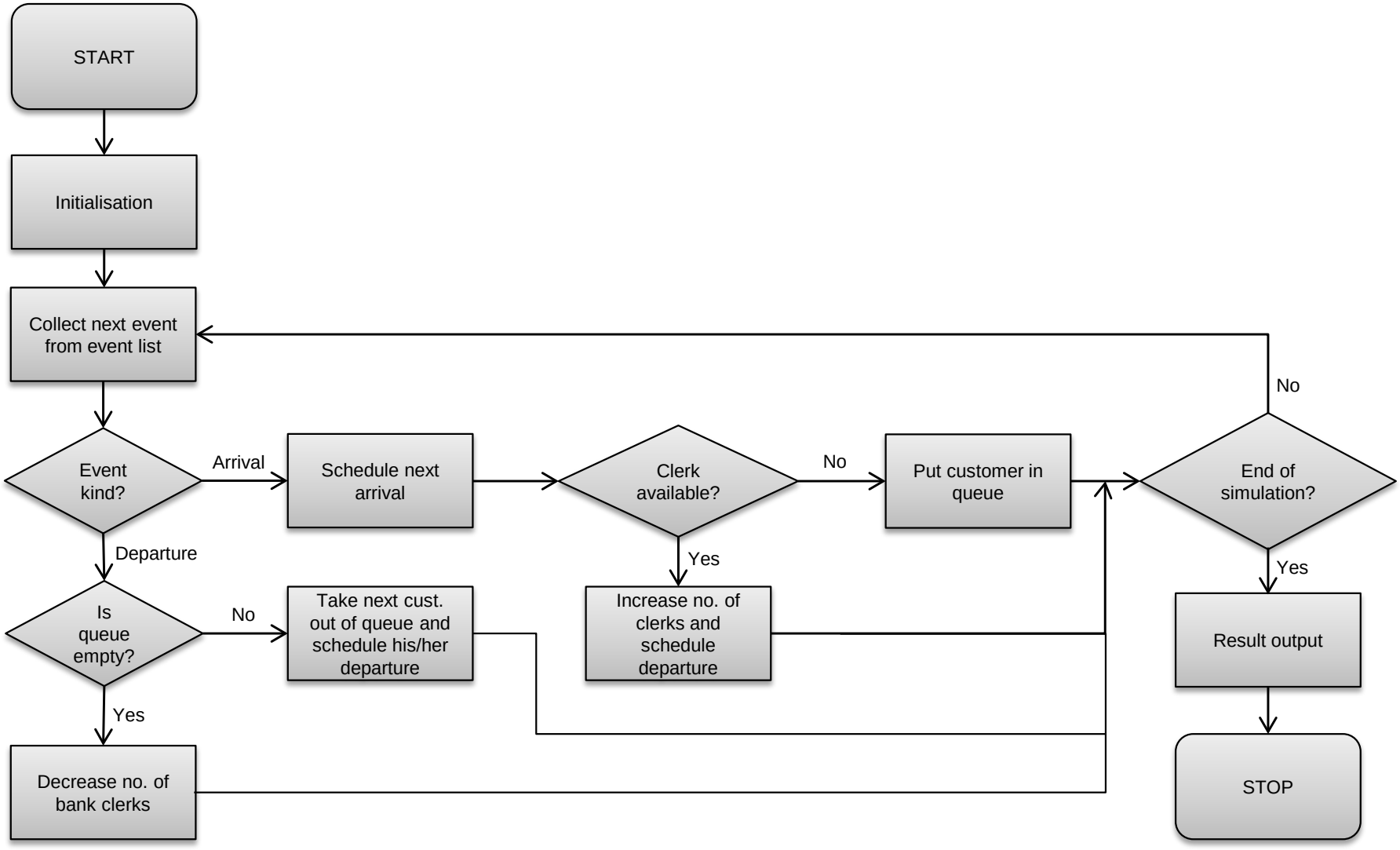
- The system starts with arrival event, namely at time A_1 (obtained from a uniform sample!). The occurence of any event typically triggers new events. For example, first event will incite a new arrival event at time $A_1 + A_2$ (another sample!), and because all clerks are still available, a departure event at time $A_1 + B_1$.

The bank: towards a computer program

Idea of a discrete event simulation program: cycle through all events, at which point system states and statistical counter variables will be adjusted.

- The system starts with arrival event, namely at time A_1 (obtained from a uniform sample!). The occurence of any event typically triggers new events. For example, first event will incite a new arrival event at time $A_1 + A_2$ (another sample!), and because all clerks are still available, a departure event at time $A_1 + B_1$.
- A computer program keeps all events in an event list in increasing order of event times. It constantly deletes the first event of the event list, leading to new events in the event list and updates in all variables.
- At the end of the simulation, distill performance measure estimates from statistical counter variables.

A flow chart of the computer program



A Mathematica implementation

Now: demonstration and treatment of example Mathematica code.

Some things to note

- Effectively modularising is key to writing a good simulation program.
- Object oriented programming languages like Java, C++, Python will yield much faster programs than Mathematica/Matlab, et cetera.

Some things to note

- Effectively modularising is key to writing a good simulation program.
- Object oriented programming languages like Java, C++, Python will yield much faster programs than Mathematica/Matlab, et cetera.
- I opted for a simple array to implement the event list. However, anytime an event is added to the event list, the whole event list needs to be sorted. For large simulation programs, this is *very inefficient*.
- Popular data objects to use for event lists are *binary trees*, *heaps* and *linked lists*, as they sort much more efficiently. This is however beyond the scope of this course.

Some issues for future lectures

- The bank simulation program only simulates one day. To get true expectations of performance measures, we would have to rerun the program many times and take averages. Key issue: how many runs are sufficient for reliable estimates?

Some issues for future lectures

- The bank simulation program only simulates one day. To get true expectations of performance measures, we would have to rerun the program many times and take averages. Key issue: how many runs are sufficient for reliable estimates?
- The bank only runs for seven hours. Some systems run in a stationary setting (i.e., as if they have run for an infinite amount of time already). Key issue: how long does a simulation need to run before it sufficiently reached stationarity?

Some issues for future lectures

- The bank simulation program only simulates one day. To get true expectations of performance measures, we would have to rerun the program many times and take averages. Key issue: how many runs are sufficient for reliable estimates?
- The bank only runs for seven hours. Some systems run in a stationary setting (i.e., as if they have run for an infinite amount of time already). Key issue: how long does a simulation need to run before it sufficiently reached stationarity?

See you next time!