COMPUTER SCIENCE LARGE PRACTICAL

SURVEY RESULTS

- Approx. 1/5 of class responded; statistically significant?
- The majority of you have substantial experience in Java, and all have at least some basic experience in Python.
- Naturally, Java is the preferred choice for CSLP.
  - That's fine; just make sure you can work comfortably in other languages in the long run.
- Nearly half of you never wrote Bash scrips
  - I will explain some Bash scripting concepts today; link to comprehensive tutorial on the course web page.

SURVEY RESULTS (II)

- Almost 1/2 of respondents never used git.
  - Link to quick guide on course web page;
    I will give a summary today.
- To cover code reusability and optimisation concepts.
- Graph theory: 2/3 unfamiliar with shortest path/graph traversal algorithms.
- Statistics: 2/3 can only compute basic statistics (e.g. mean) or have no knowledge.

SURVEY RESULTS (III)

- Results visualisation: major gap!
  - Training somewhat out of scope of these lectures, but will try to cover some basics at the end;
  - Guides to various tools already on course web page.
- Topics you want covered:
  - Version control, Bash, graph theory, code optimisation, statistics → all planned;
  - Results plotting → time permitting;
  - Music (!) → Not the right course, but will try to put out a (highly subjective!) Spotify playlist for coding.

THE SIMULATOR

DEFINITIONS

- In the requirements I have stated that your simulator will be a:
  - stochastic,
  - discrete event,
  - discrete time

  simulator.
- Let's see what each of these terms means.

STOCHASTICITY

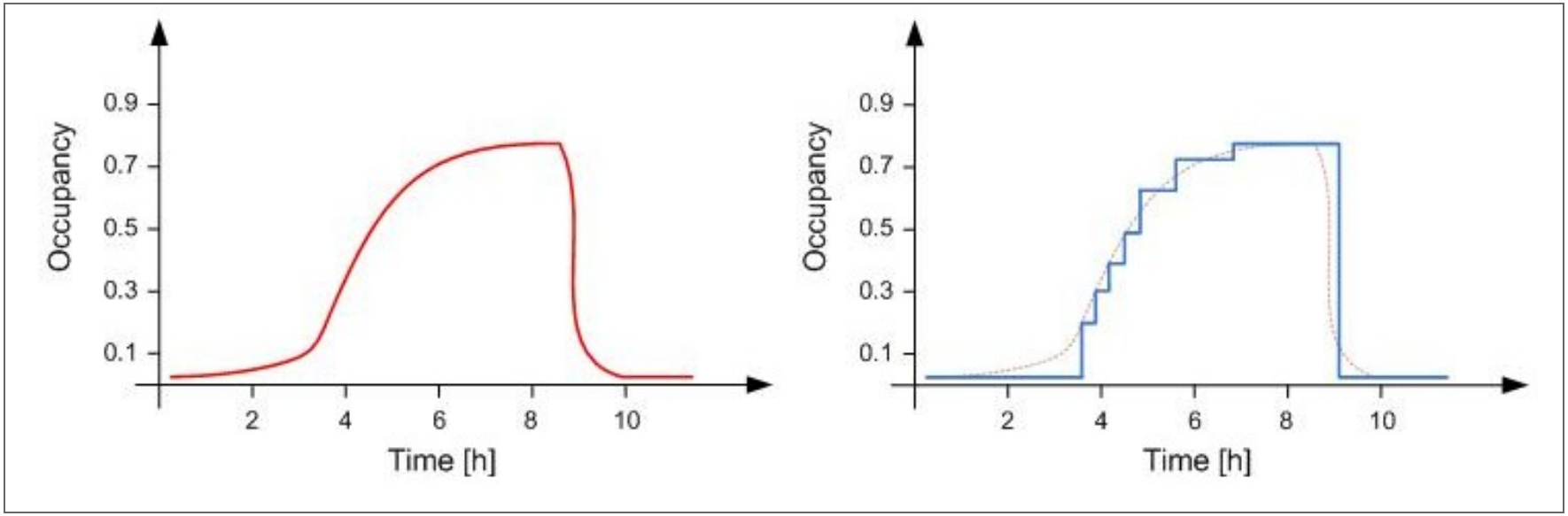
- A stochastic process is one whose state evolves “non-deterministically”, i.e. the next state is determined according to a probability distribution.
- This means a stochastic simulator may produce slightly different results when run repeatedly with the same input.
- Therefore it is appropriate to compute certain statistics to characterise the behaviour of the simulated system.
- Remember, these are statistics about the **model**:
  - You **hope** that the real system exhibits behaviour with similar statistics.

DISCRETE EVENTS

- Discrete events happen at a particular time and mark a change of state in the system.
- This means discrete-event simulators do not track system dynamics continuously, i.e. an event either takes place or it does not.
- There is no fine-grained time slicing of the states, i.e.
- Generally a state **could** be encoded as an integer.
- Usually it is encoded as a set of integers, possibly coded as different data types.
- Discrete-event simulations run faster than continuous ones.

DISCRETE VS CONTINUOUS STATES

- When working with discrete events, it is common to consider that *states are also discrete*.
- Example:



DISCRETE TIME

- Discrete time simulations operate with a discrete number of points:
  - Seconds, Minutes, Hours, Days, etc.
- These can also be *logical* time points:
  - Moves in a board game,
  - Communications in a protocol.
- Your task is to write a *discrete time* simulator.
- Events will occur with *second level granularity*.

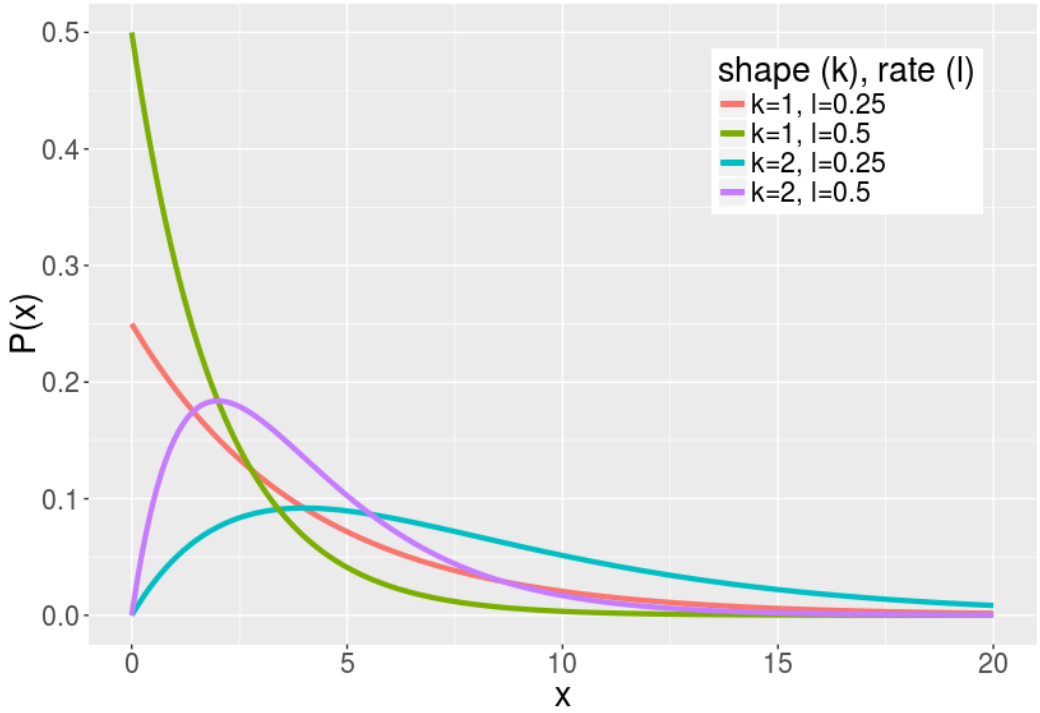
THE ERLANG-K DISTRIBUTION

- The *probability distribution* gives the probability of the different possible values of a random variable.
- The Erlang-k distribution is the distribution of the sum of k independent exponential variables with mean $\mu = 1/\lambda$, where λ is the rate parameter.
- An exponential distribution describes the time between events in a *Poisson process*.
  - The time X between two events follows exponential distribution if the prob. that an event occurs during a certain time interval is proportional to its length.

THE ERLANG DISTRIBUTION

- Special case of Gamma distribution (Gamma allows k real)
- Mean $\mu = k/\lambda$; that is if something occurs at rate λ, then we can expect to wait k/λ time units on average to see each occurrence.
- Applications: telephone traffic modelling, queuing systems, biomathematics, etc.

THE ERLANG DISTRIBUTION

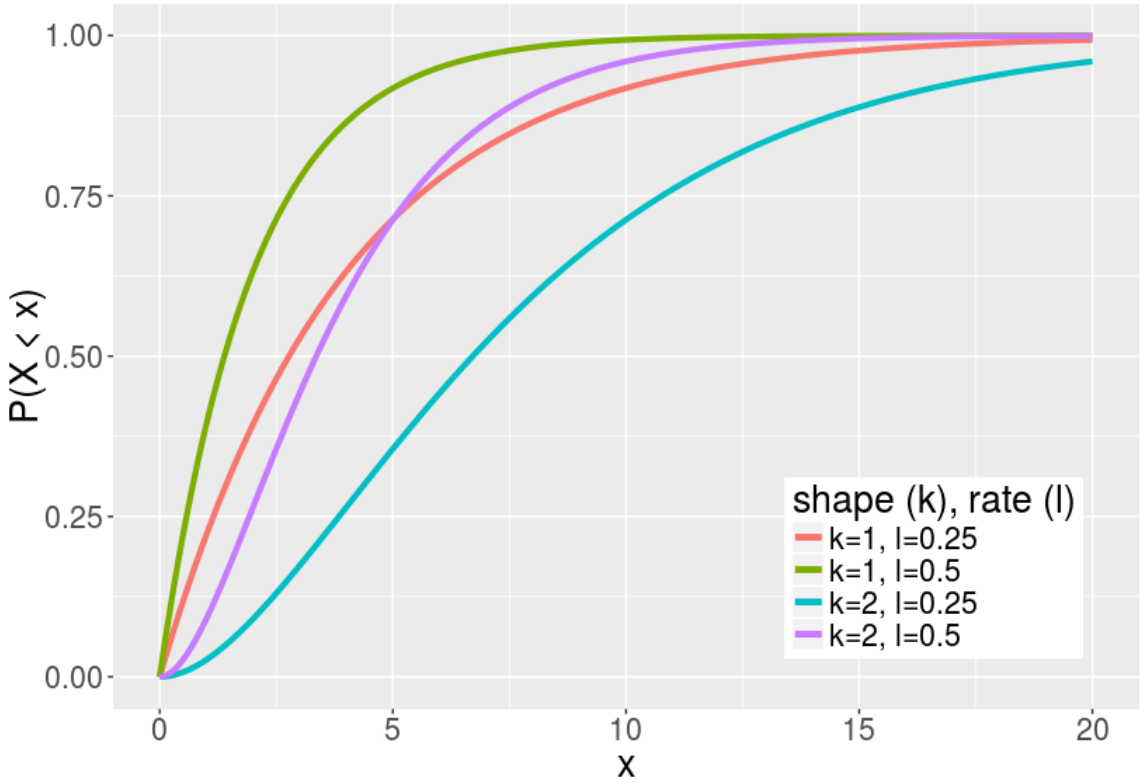


- The *probability density function* (PDF) is given by:

$$f(x,\ k,\ \lambda)\ =\ \frac{\lambda^{k} x^{k\ -\ 1} e^{-\lambda x}}{(k\ -\ 1)\,!},\ \forall x,\ \lambda\ \geq\ 0$$

- Describes the relative likelihood that an event with rate λ occurs at time x.

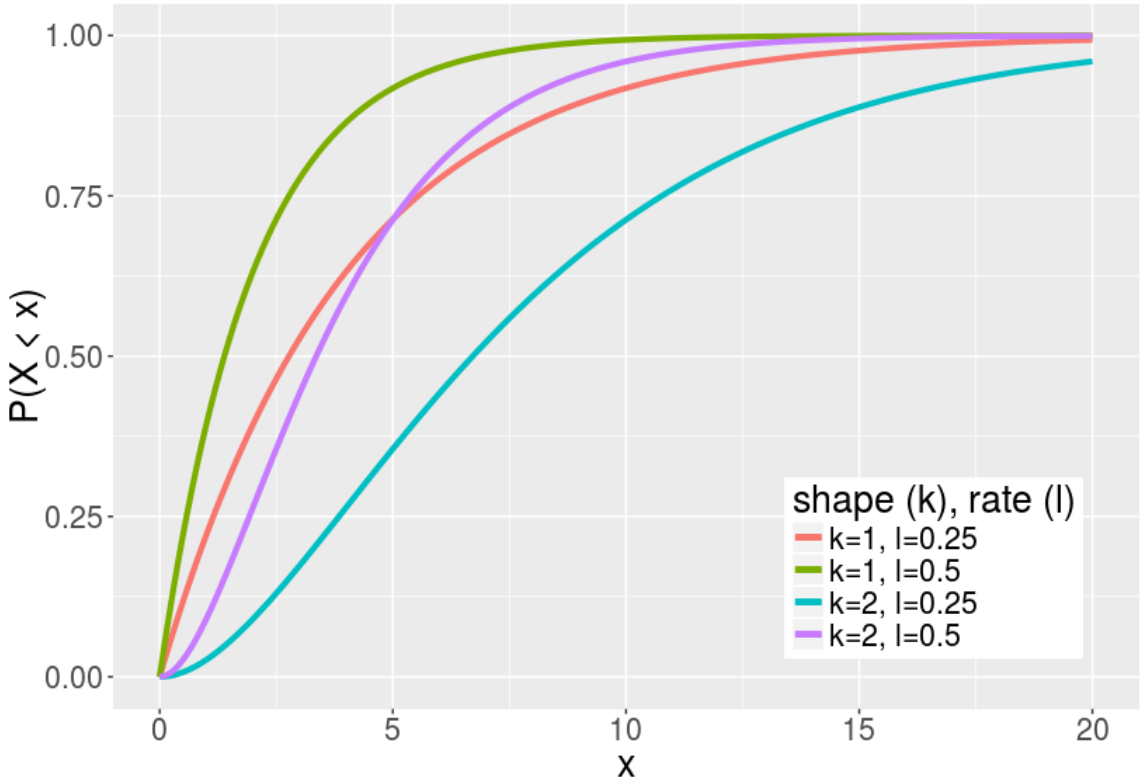
THE ERLANG DISTRIBUTION



- The *cumulative distribution function* (CDF) is given by:

$$F(x,\ k,\ \lambda)\ =\ 1\ -\ \sum_{n\ =\ 0}^{k\ -\ 1}\frac{1}{n\ !}e^{-\lambda x}(\lambda x)^{n}\ .$$

THE ERLANG DISTRIBUTION



- So if something happens at a rate of 0.5 per unit of time, and the shape of the distribution is 2, then the probability that we will observe it occurring within 1 time unit is:

$$F(1,\ 2,\ 0.5)\ =\ 1\ -\ (e^{-0.5\ \times\ 1}\ +\ e^{-0.5\ \times\ 1}\ \times\ 0.5)\ =\ 0.09$$

EXERCISE

What is the probability that a random variable X is less than its expected value, if X has an Erlang distribution with rate λ and shape $k = 1$?

The expected value is:

$$E[X] = \frac{k}{\lambda} = \frac{1}{k}$$

EXERCISE

We need to compute $P(X \leq E[X])$ using the distribution function:

$$P(X \leq E[X]) = P(X \leq 1/\lambda)$$

$$= F(x, k, \lambda)$$

$$= 1 - e^{-\lambda * \frac{1}{\lambda}}$$

$$= 1 - \frac{1}{e}$$

HOW DO WE SAMPLE FROM A DISTRIBUTION?

INVERSE TRANSFORM METHOD

- Let X be a RV with continuous and increasing distribution function F. Denote the inverse by F^{-1}.
- Let U be a random variable uniformly distributed on the unit interval (0, 1).
- Then X can be generated by $X = F^{-1}(U)$.

SAMPLING FROM AN ERLANG DISTRIBUTION

If we use an **Erlang** CDF for F, then we effectively sample from that distribution by

$$X \approx \sum_{i=1}^{k} -\frac{1}{\lambda}\ln(U_i) = -\frac{1}{\lambda}\ln \prod_{i=1}^{k} U_i$$

DRAWING UNFORMLY DISTRIBUTED NUMBERS

You will probably do this in Java

```
import java.util.Random;

...

Random r = new Random();   // Uses time in ms as seed

...

double d = r.nextDouble(); // draws between 0 (inclusive!)
                           // and 1 (exclusive)
```

Remember you will need to feed that into a logarithm.

If using other languages, careful how generator is seeded.

WRITING BASH SCRIPTS

WHAT'S BASH

- Command line interface for working with Unix-type systems (default on Linux, Mac OS).
- You can work interactively, i.e. write commands one at the time to the prompt, press ENTER ...
- or write scripts that execute multiple commands for you → great if you want to schedule jobs, test code, parse files.
- A script is nothing but a text file where you write a sequence of system commands.

BASH SCRIPTING

- Make sure you made the script executable, e.g.

```
$ editor test.sh
$ chmod +x test.sh
```

- Bash scripts typically start with a line

```
#!/bin/bash
```

to let the system know this is a Bash script (you may alternatively work in Perl or others).

BASH SCRIPTING

- Scripts work pretty much like any other program, so can take arguments; you can easily check how many were given, or display them

```
...
echo "Number of arguments $#"

if [[ $# > 0 ]] ; then
 echo "First argument $1"
else
 echo "No arguments given"
fi
```

- Running produces:

```
$ ./test.sh param
Number of arguments: 1
```

BASH SCRIPTING

A couple of things happened in this example:

- `echo` used to print to standard output;
- the `$` character preceded arguments (same for variables);
- `$#` used to retrieve the number of arguments passed;
- Used an if statement to check if number of arguments was no-zero; it wasn't so `$1` was identifying the first;
- **IMPORTANT:** Careful about the spaces; Bash is very picky when it comes to writing conditionals;
- To launch a script put `./` before script name.

SCRIPTING YOUR APPLICATION

- Your program must take as argument an input script (this is THE input containing all the simulation parameters; not to be confused with the Bash script!)
- The Bash script will take the name of the input and pass it down to your executable file.
- This in turn must be able to handle command line arguments.
- Bash script snippet:

```
...
java Simulator $1
```

SCRIPTING YOUR APPLICATION

- Java code:

```
class Simulator{
  public static void main(String args[]){
    System.out.println("Arguments passed:");
    for (String s: args) {
      System.out.println(s);
    }
  }
}
```

- Executing script:

```
$ ./test.sh basic_input
Arguments passed:
basic_input
```

DESIGN CHOICES

- There are a few things you need to decide whether to implement inside the Bash script or inside your Java/C/Python/other code.
- This includes checking if arguments have been passed, displaying usage information, checking if the file exists when a file is expected as argument, etc.
- This is entirely your choice.
- You may later write more sophisticated scripts to run experiments on multiple files through a single command.
- Bash works nicely with AWK for text processing (parsing your output).

SOURCE CODE CONTROL

SOURCE CODE CONTROL

- For this project must `git` for version control and the Bitbucket platform.
  - This is somewhat realistic
  - Any project you join will likely already have some form of source code control set up which you will have to learn to use rather than any system you might already be familiar with
  - See **the git homepage** for detailed documentation.

SOURCE CODE CONTROL

- The practical is not looking for you to become an expert in `git`;
- You will **not need** to be able to perform complicated branches or merges;
- This is, after all, an individual practical
- What is key, is that your commits are appropriate:
  - Small frequent commits of single units of work;
  - Clear, coherent and unique commit messages.

GETTING STARTED

- By now I assume everyone has forked the skeleton CSLP-16-17 repository and granted read permissions to the marker and me.
- You can start with a simple README file to plan and document your work (use any text editor you wish).

```
$ cd simulator
$ editor README.md
$ git add README.md
$ git commit -m "Initial commit including a README"
$ git push origin master
```

THE MAIN POINT

- After each portion of work, commit to the repository what you have done.
- Everything you have done since your last commit, is not recorded.
- You can see what has changed since your last commit, with the *status* and *diff* commands:

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

STAGING AND COMMITTING

- When you commit, you do not have to record **all** of your recent changes. Only changes which have been *staged* will be recorded
- You *stage* those changes with the `git add` command.
- Here a file has been modified but not *staged*

```
$ editor README.md
$ git status
# On branch master
# Changed but not updated:
#   (use "git add ‹file›..." to update what will be committed)
#   (use "git checkout -- ‹file›..." to discard changes in working directory)
#
#       modified:   README.md
#
no changes added to commit (use "git add" and/or "git commit -a")
```

UNRECORDED AND UNSTAGED CHANGES

- A `git diff` at this point will tell you the changes made that have not been committed or staged

```
$ git diff
diff --git a/README.md b/README.md
index 9039fda..eb8a1a2 100644
--- a/README.md
+++ b/README.md
@@ -1 +1,2 @@
 This is a stochastic simulator.
+It is a discrete event/state, discrete time simulator.
```

TO ADD IS TO STAGE

- If you *stage* the modified file and then ask for the status you are told that there are staged changes waiting to be committed.
- To *stage* the changes in a file use `git add`

```
$ git add README.md
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD ‹file›..." to unstage)
#
#       modified:   README.md
#
```

VIEWING STAGED CHANGES

- At this point `git diff` is empty because there are no changes that are not either committed or staged.
- Adding `--staged` will show differences which have been staged but not committed.

```
$ git diff # outputs nothing
$ git diff --staged
diff --git a/README.md b/README.md
index 9039fda..eb8a1a2 100644
--- a/README.md
+++ b/README.md
@@ -1 +1,2 @@
 This is a stochastic simulator.
+It is a discrete event/state, discrete time simulator.
```

NEW FILES

- Creating a new file causes git to notice there is a file which is not yet tracked by the repository.
- At this point it is treated equivalently to an unstaged/ uncommitted change.

```
$ editor mycode.mylang
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD ‹file›..." to unstage)
#
#       modified:   README.md
#
# Untracked files:
#   (use "git add ‹file›..." to include in what will be committed)
#
#       mycode.mylang
```

NEW FILES

- `git add` is also used to tell git to start tracking a new file.
- Once done, the creation is treated exactly as if you were modifying an existing file.
- The addition of the file is now treated as a staged but uncommitted change.

```
$ git add mycode.mylang
# On branch master
# Changes to be committed:
#   (use "git reset HEAD ‹file›..." to unstage)
#
#       modified:   README.md
#       new file:   mycode.mylang
#
```

COMMITTING

- Once you have staged all the changes you wish to record, use `git commit` to record them.
- Give a useful message to the commit.

```
$ git commit -m "Added more to the readme and started the implementation"
[master a3a0ed9] Added more to the readme and started the implementation
 2 files changed, 2 insertions(+), 0 deletions(-)
 create mode 100644 mycode.mylang
```

PUSHING

- Your changes are now committed to your local working copy.
- You must also send those changes to the remote Bitbucket repository, otherwise the marker/I will not be able to see the updates.

```
$ git push origin master
```

A CLEAN REPOSITORY FEELS GOOD

- After a commit, you can take the status, in this case there are no changes
- In general though there might be some if you did not *stage* all of your changes

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

FINALLY GIT LOG

- The `git log` command lists all your commits and their messages

```
$ git log
commit a3a0ed90bc90e601aca8cc9736827fdd05c97f8d
Author: Name ‹author email›
Date:   Wed 28 Sep 09:15:32 BST 2016

    Added more to the readme and started the implementation

commit 22de604267645e0485afa7202dd601d7c64c857c
Author: Name ‹author email›
Date:   Wed 28 Sep 09:15:32 BST 2016

    Initial commit
```

MORE ON THE WEB

- Clearly this was a very short introduction
- More can be found at the git book online at:
  - **http://git-scm.com/documentation**
- And countless other websites