Discrete Event Simulation and SimPy

Discrete Event Simulation for Modelling Pathways and Queuing Problems

Dr Daniel Chalk

What is Discrete Event Simulation?

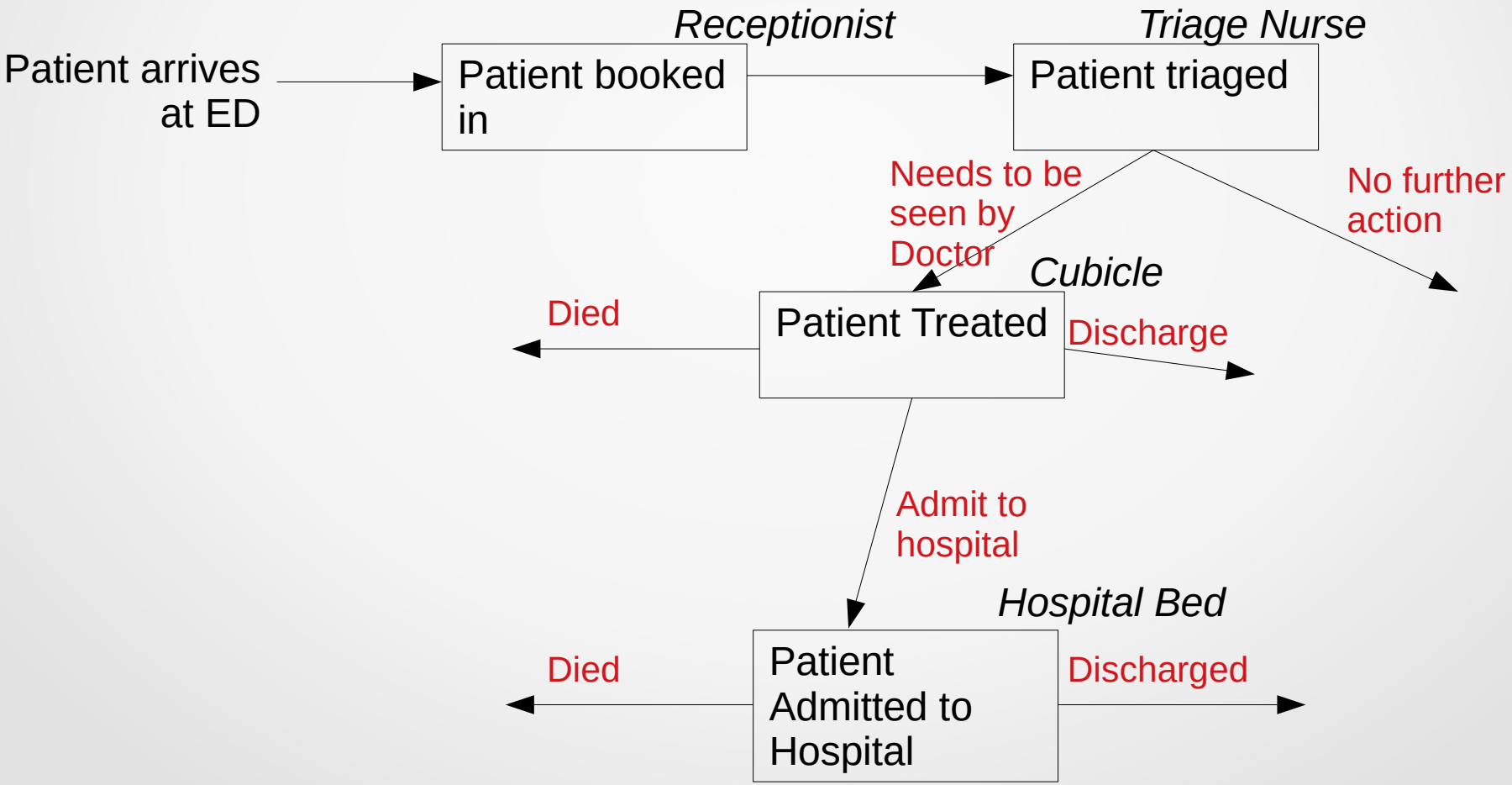
Discrete Event Simulation (DES) is a way of modelling *queuing problems*.

In a DES, *entities* flow through (and queue for) *discrete sequential processes* that use *resources*.

DES is typically used to model processes and pathways. For example, what happens to patients when they arrive at the Emergency Department.

Therefore, DES is useful for asking “what if?” questions about process / pathway changes.

An Example


Patient arrives at ED
Receptionist
Patient booked in
Triage Nurse
Patient triaged
Needs to be seen by Doctor
No further action
Cubicle
Patient Treated
Died
Discharge
Admit to hospital
Hospital Bed
Patient Admitted to Hospital
Died
Discharged

Components of Discrete Event Simulation

Entities are the things flowing through the sequential processes in the model (e.g. patients, telephone calls, blood test results)

Generators are the way in which entities enter the model and come into being (e.g. brought in by paramedics, self-presenting at the ED)

Inter-arrival Times specify the time between entities being generated (arriving in the model)

Components of Discrete Event Simulation

Activities / Servers represent the activities that happen to entities (e.g. triage, treatment, ward admission)

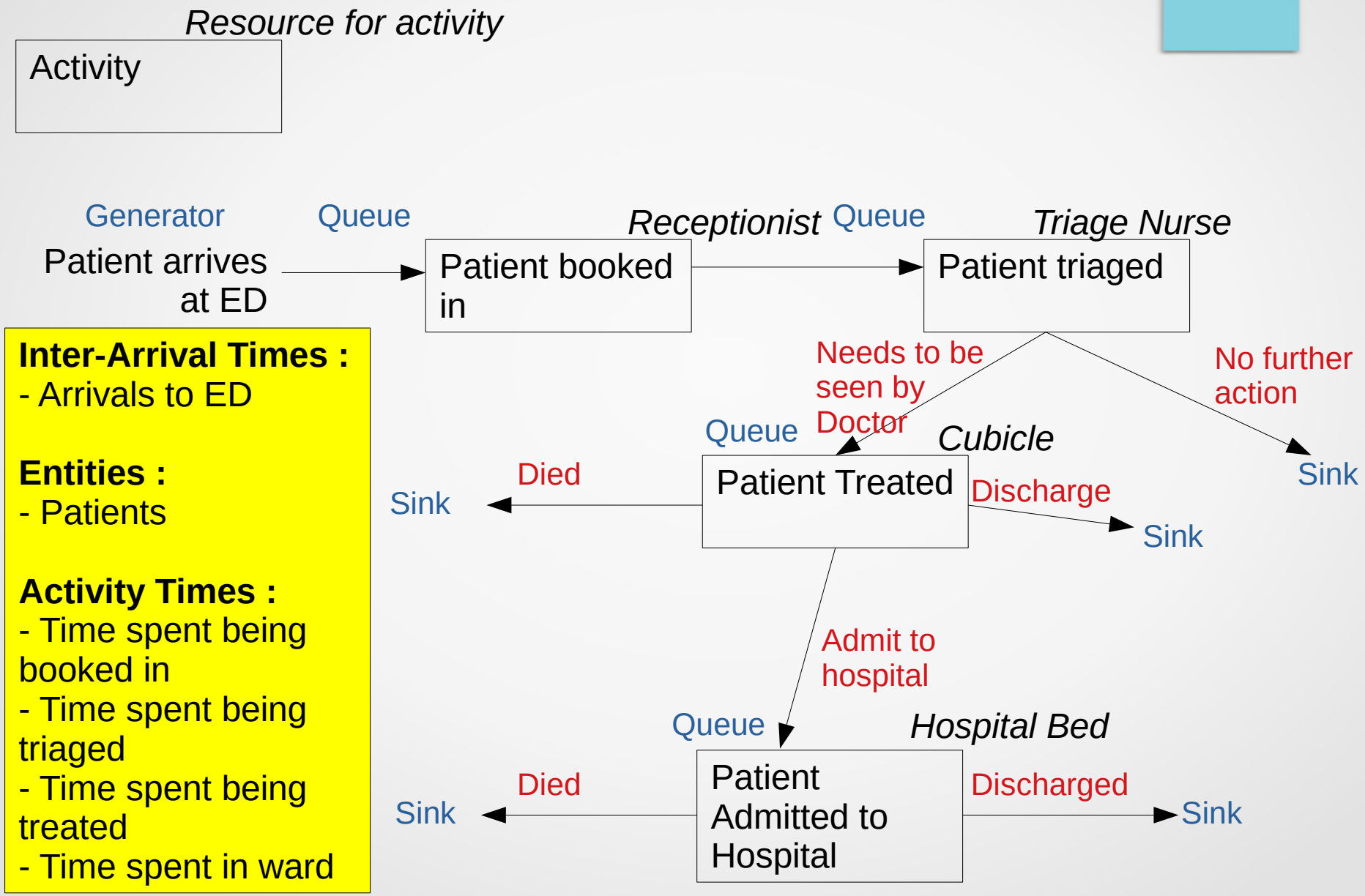
Activity / Server Time represents the amount of time it takes for an activity to happen to an entity.

Resources are required for activities to take place and may be shared between activities (e.g. nurse, doctor, receptionist, bed)

Queues are where entities are held until an activity has capacity and the required resources to begin.

Sinks are how entities leave the model.

An Example



Inter-Arrival Times :
- Arrivals to ED

Entities :
- Patients

Activity Times :
- Time spent being booked in
- Time spent being triaged
- Time spent being treated
- Time spent in ward

Some Examples of Real World DES Models

Let’s look at some example Discrete Event Simulation models built by PenCHORD

Reducing referral to treatment times for bladder cancer in Cornwall using simulation modelling

Dr Daniel Chalk
Senior Research Fellow
PenCLAHRC

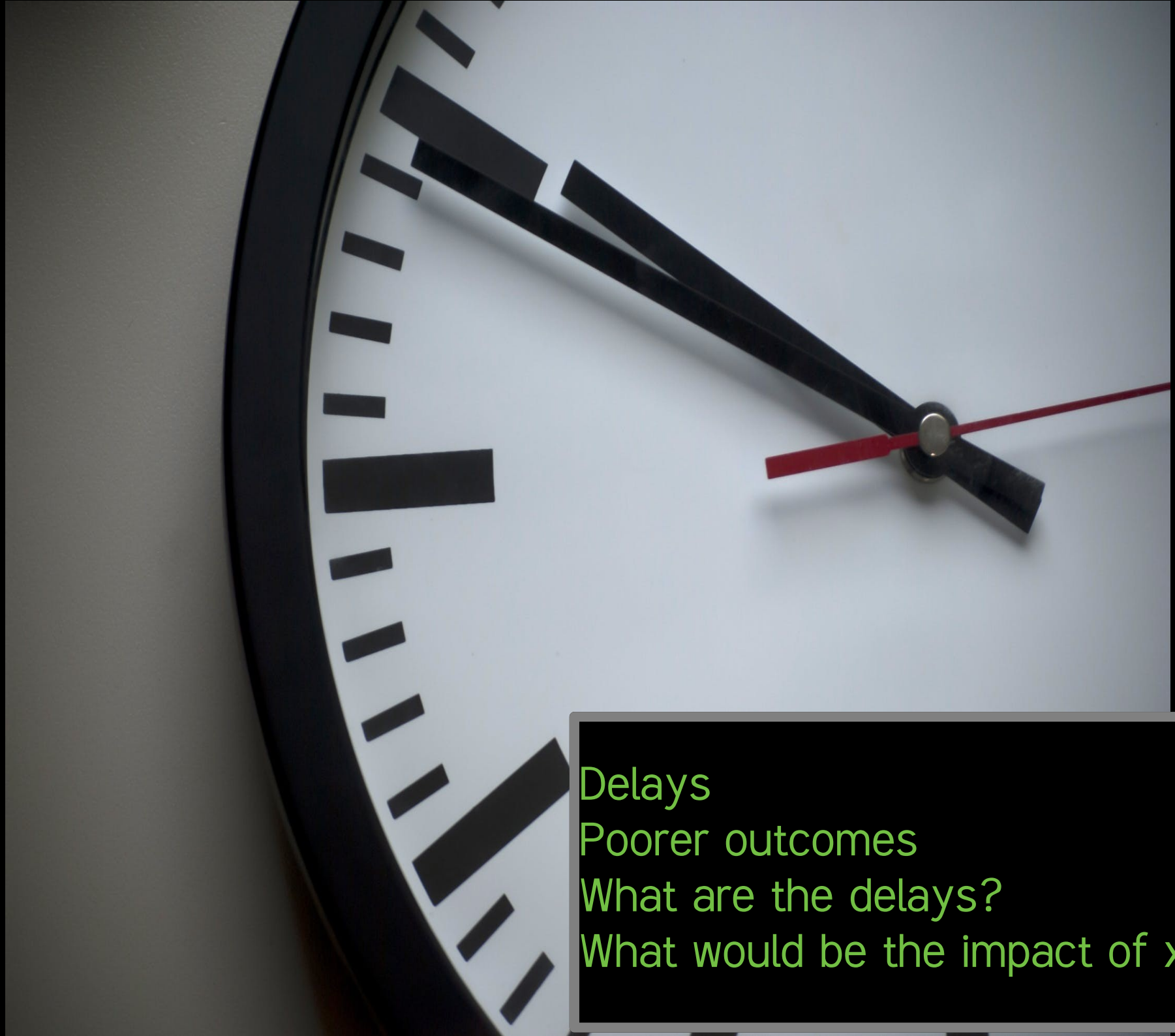
Bladder cancer = 7th most common cancer in UK (~10,000 new cases per annum)

20–25% invade muscle wall of bladder

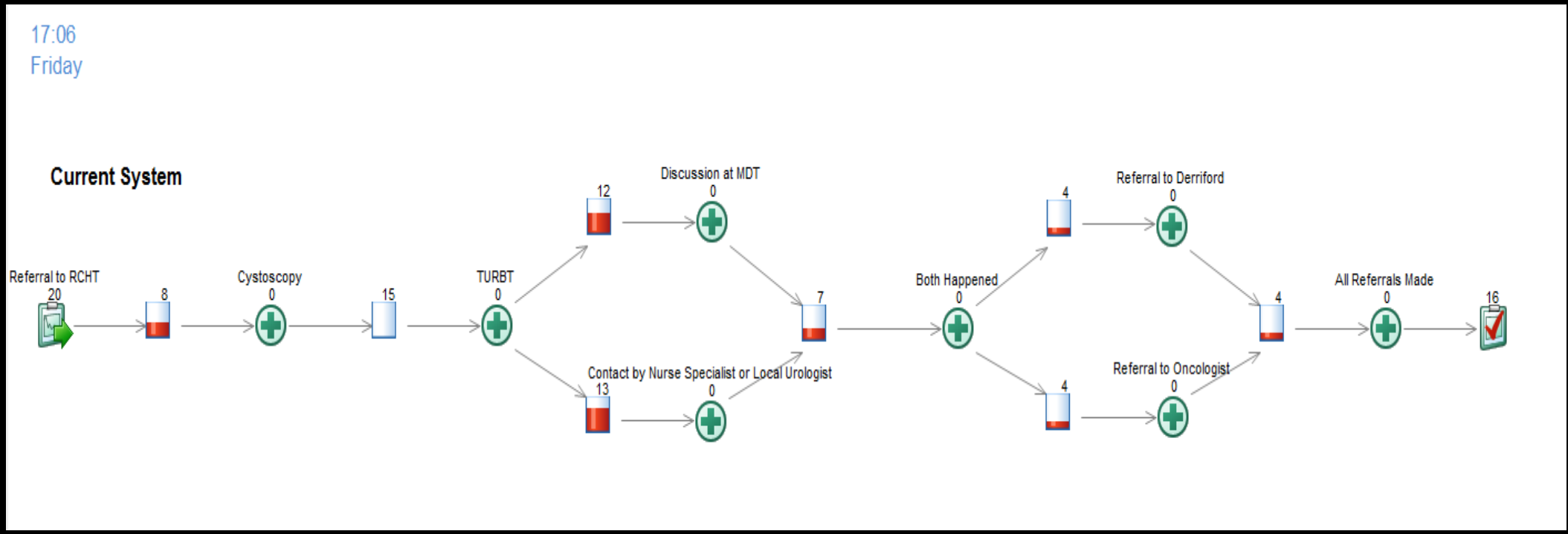
Muscle-invasive bladder cancer (MIBC) five-year survival rate only around 50%

Definitive treatment = cystectomy + chemotherapy

Collaboration with Royal Cornwall Hospitals Trust (RCHT)

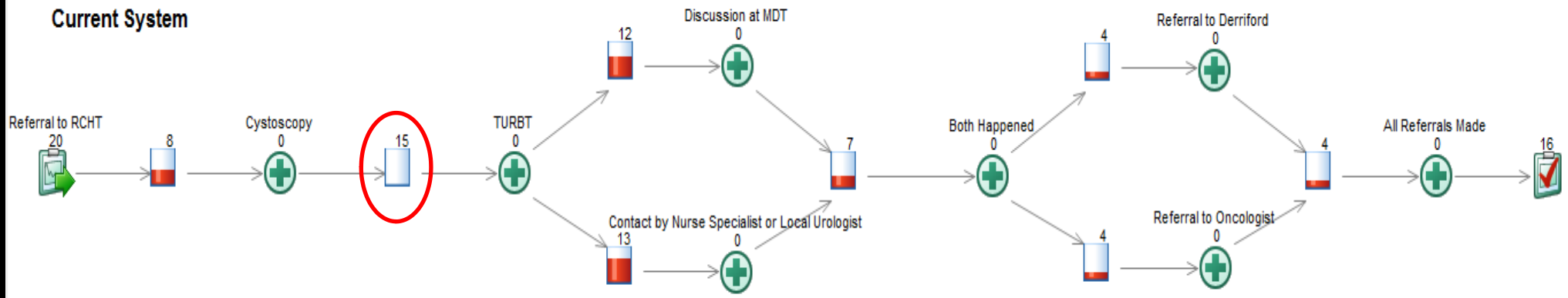

Delays
Poorer outcomes
What are the delays?
What would be the impact of x?


2 years of data
2015–16
Time to referral for definitive treatment ~ 89 days



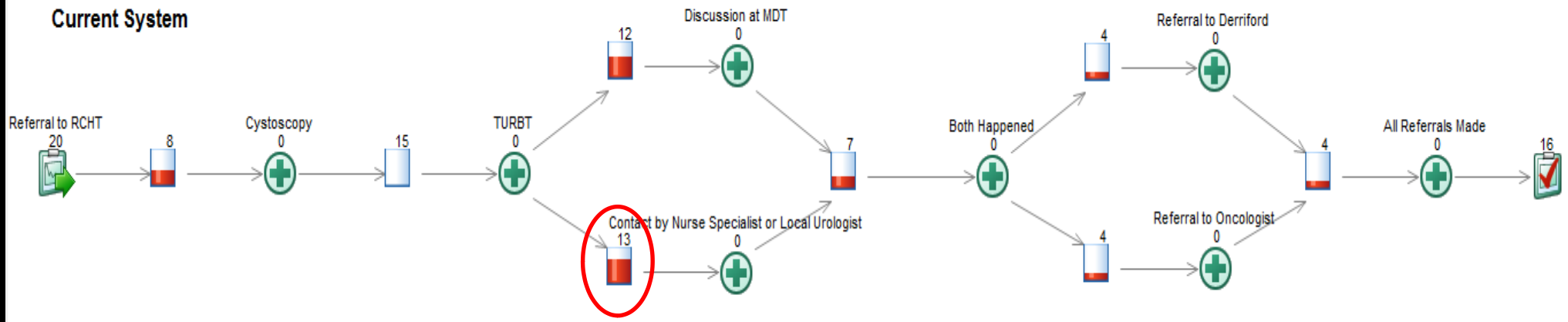
Model of (then) current pathway
Simul8
Identify bottlenecks

17:06
Friday

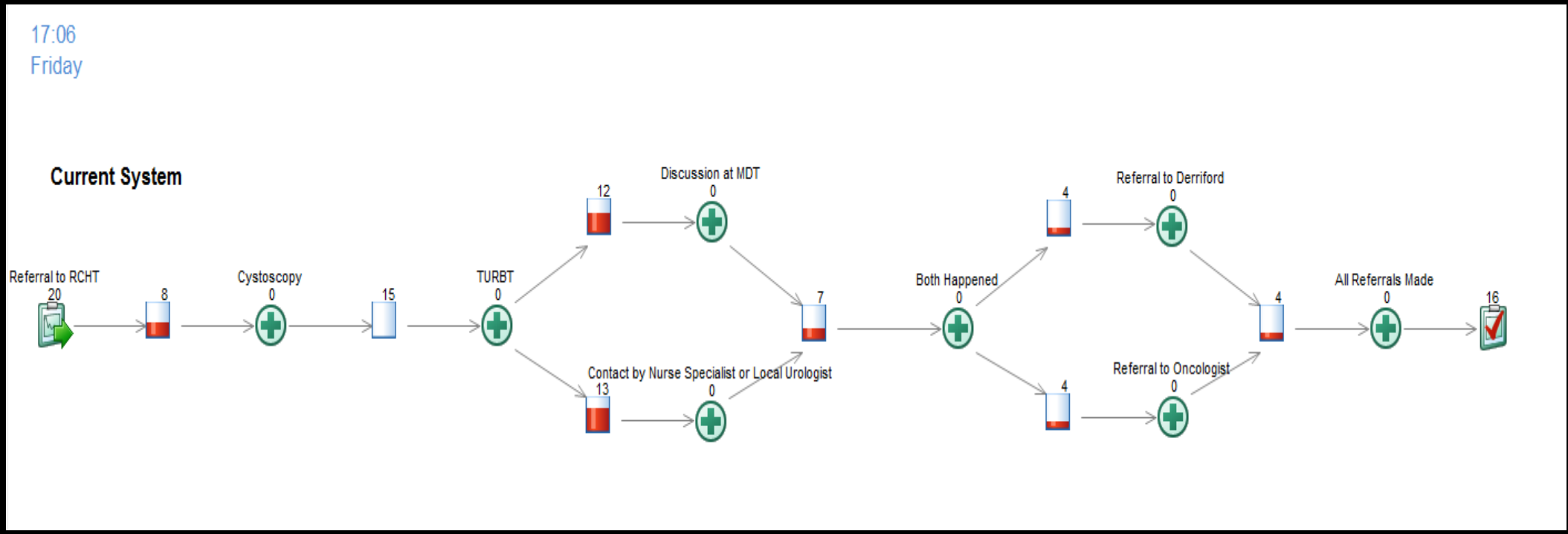


Bottleneck 1:
Wait for TURBT after cystoscopy
(mean 43 days, 32% of time)

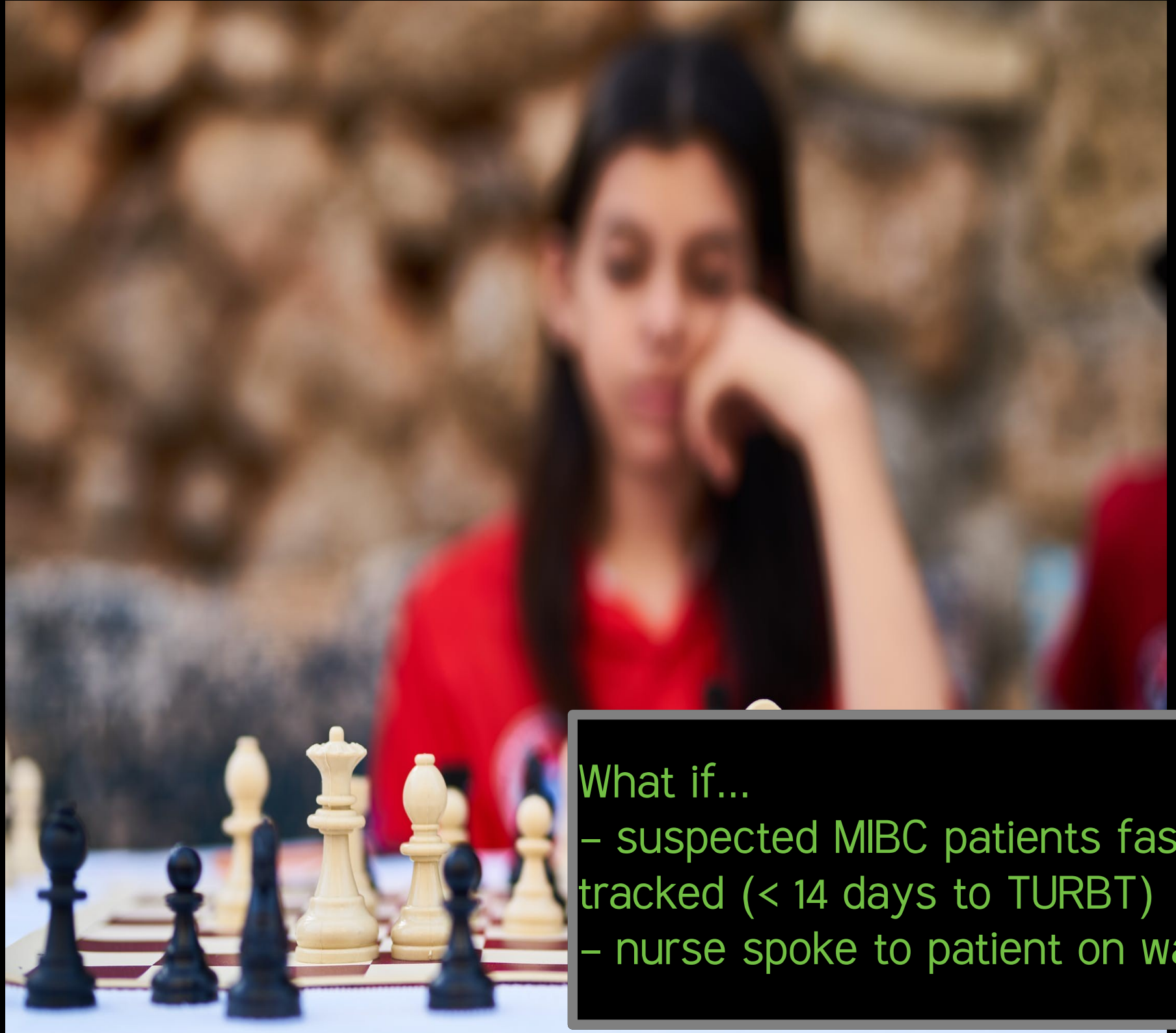
17:06
Friday



Bottleneck 2 : wait for nurse specialist / urologist to discuss diagnosis and treatment options (mean 25 days, 19% of time)



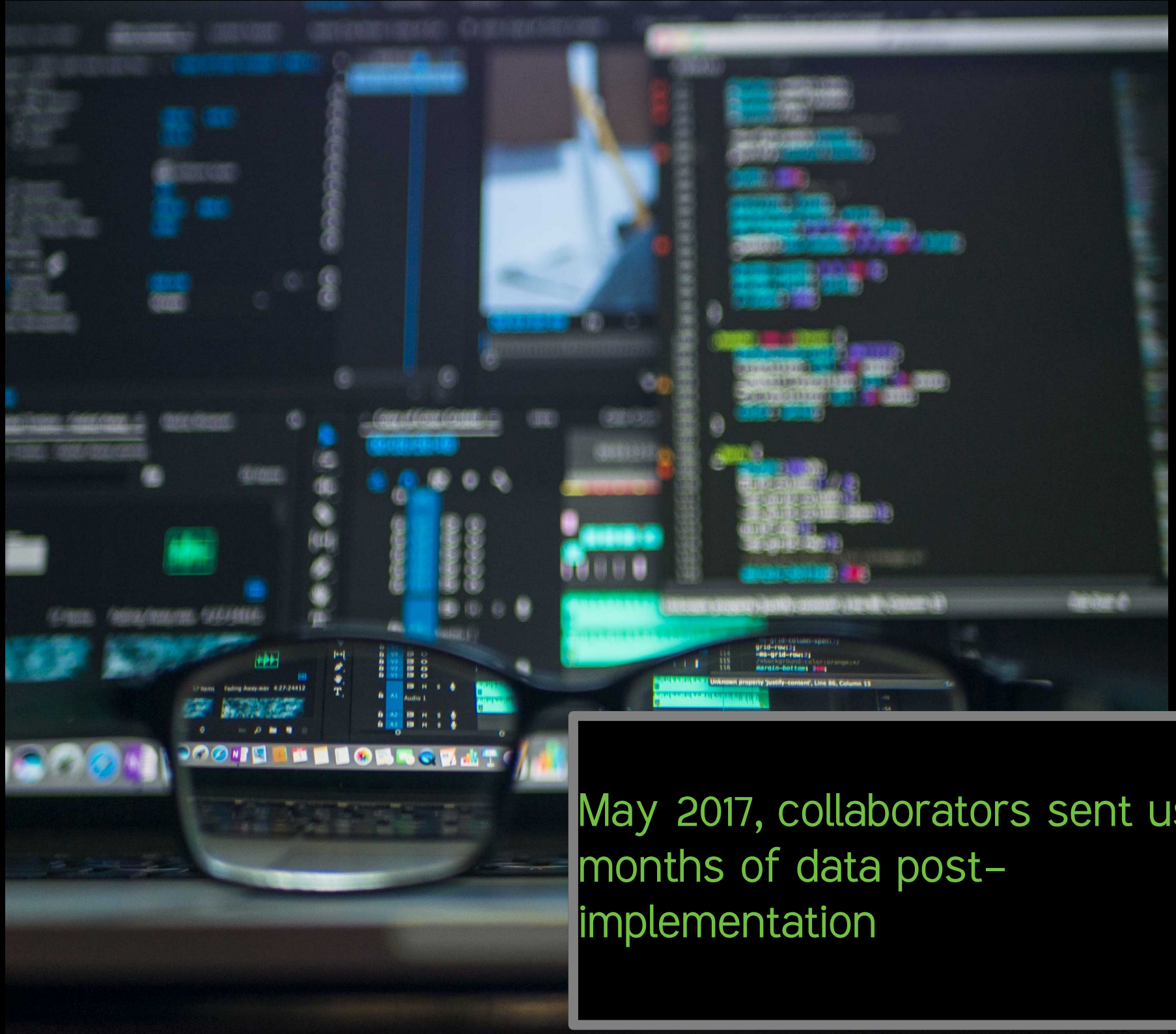
Live demo for consultants, urologists, surgeons
Ideas to reduce delays
Live tested in models



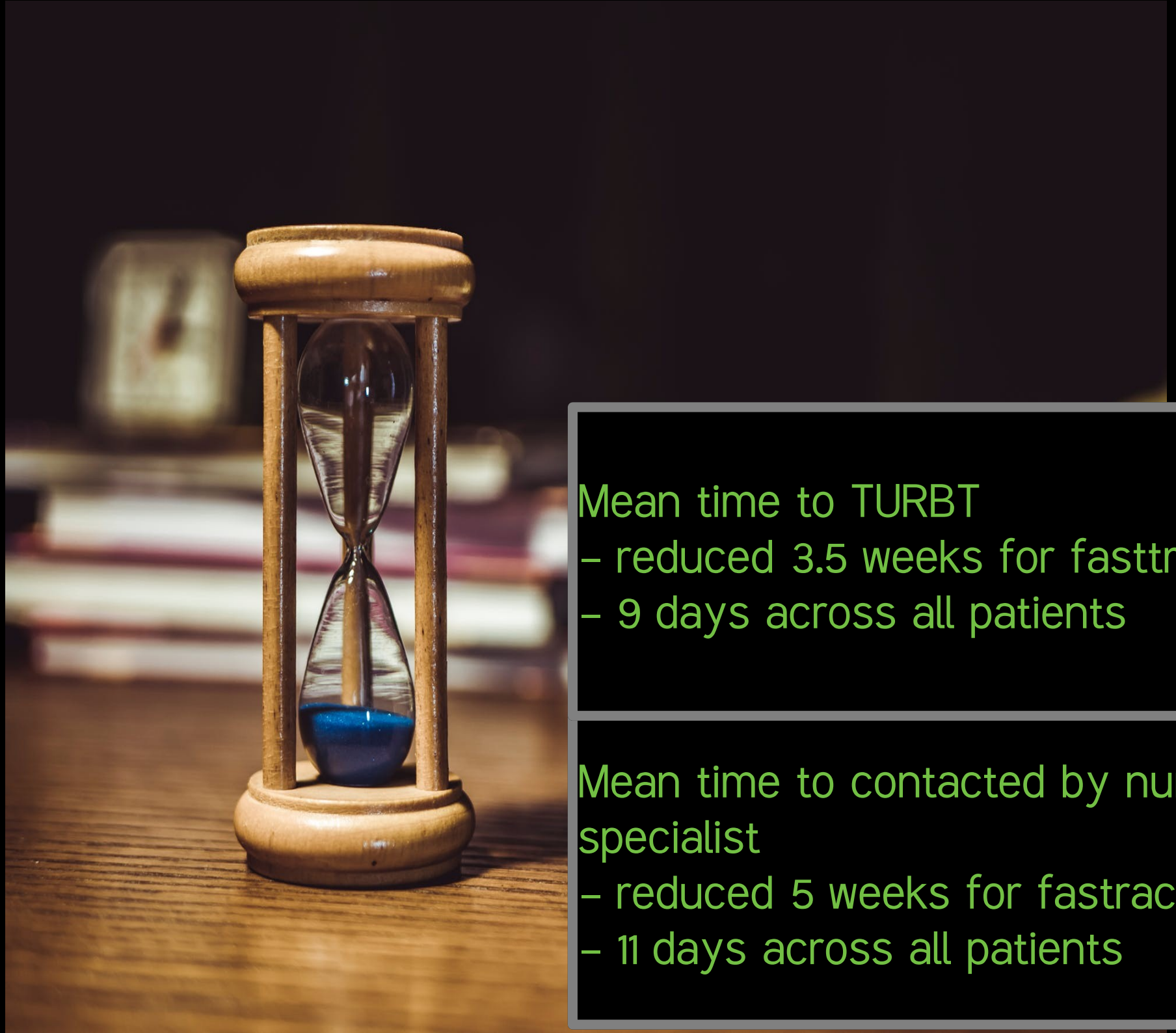
What if...
- suspected MIBC patients fast tracked (< 14 days to TURBT)
- nurse spoke to patient on ward

Model predicted if changes were made, time to referral for definitive treatment would reduce by 5.5 weeks


Immediate impact
Cancer Lead for Urology rewrote protocol within 24 hours
In place 2 days later

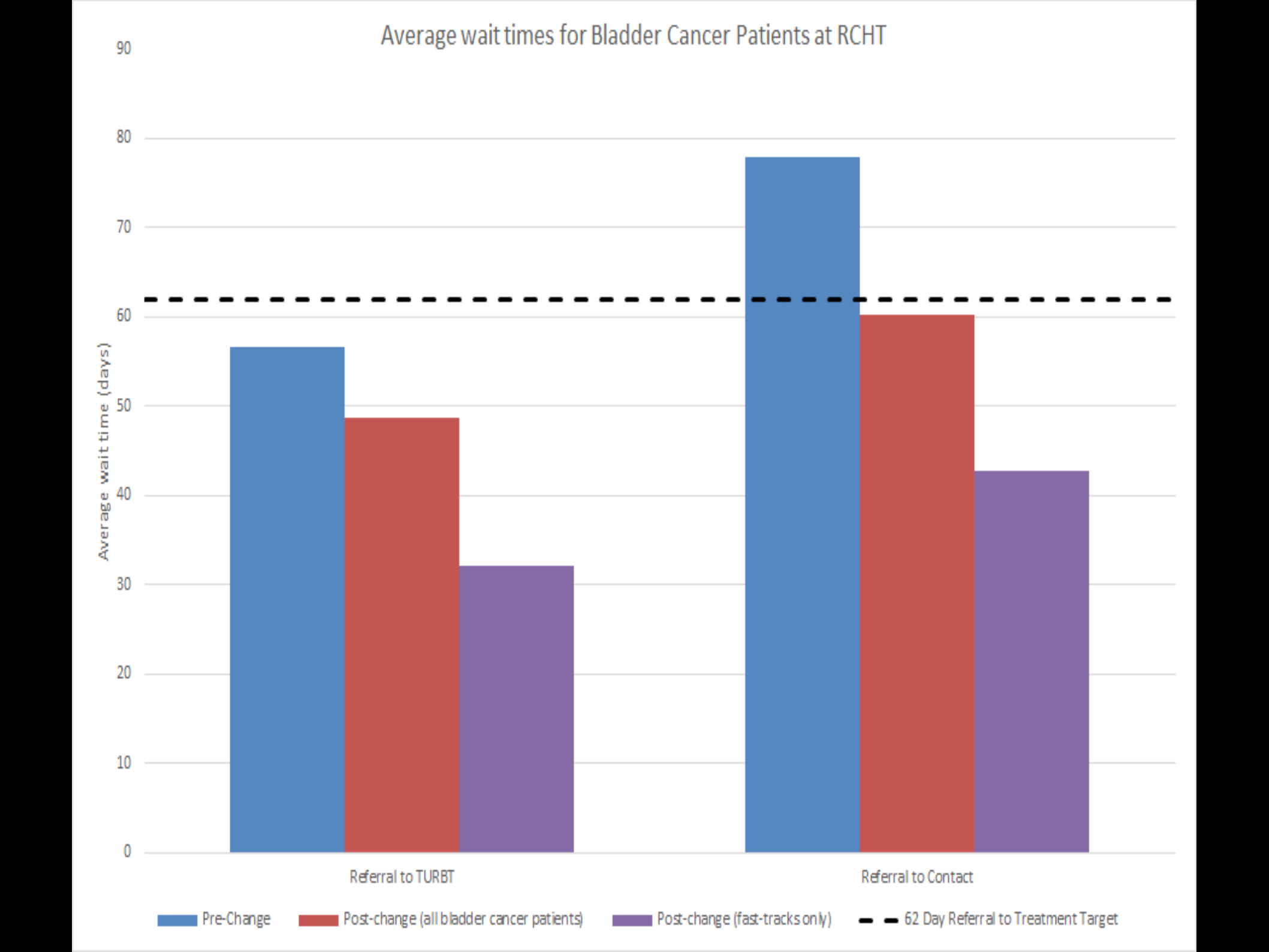


May 2017, collaborators sent us 3 months of data post-implementation



Mean time to TURBT
– reduced 3.5 weeks for fasttrack
– 9 days across all patients

Mean time to contacted by nurse specialist
– reduced 5 weeks for fastrack
– 11 days across all patients


Average wait times for Bladder Cancer Patients at RCHT
Average wait time (days)
90
80
70
60
50
40
30
20
10
0
Referral to TURBT
Referral to Contact
Pre-Change
Post-change (all bladder cancer patients)
Post-change (fast-tracks only)
62 Day Referral to Treatment Target

Alison’s Presentation

Alison’s Presentation

Mike and Tom’s Presentation (Dialysis)

Mike and Tom’s Presentation (Dialysis)

Andy’s Presentation

Andy’s Presentation

Exercise 1

In small groups of 3 – 4, read through the following and :
1. Draw a process map of the system described (note that there's more than one process being described here)
2. Write a "what if?" question(s) that captures what has been asked of you as the modeller
3. Draw up a design for a Discrete Event Simulation model, based on the "what if?" question that has been asked. Indicate the generators, queues, activities, resources and sinks in the model, and list the entities, inter-arrival times and activity times that will be used in the model. Also, remember our discussion of "scope" from the first session – you may not need to include everything from the process map of the system in your DES model design...

A local GP surgery has two receptionists, three GPs, and a nurse. The surgery is open Monday – Friday from 0830 to 1800. Patients who want to see a GP must first call into the surgery, where a receptionist will ask what the problem is, and then pass their details to a GP to call them back. Typically within a few hours a GP will call the patient and triage the patient over the phone initially. The GP may decide no further action is needed, or they may prescribe some medication for the patient, which will be ready to collect from reception within 1 hour. In some cases, the GP may decide that the patient needs to be seen in person so they can perform an examination, and these patients are asked to come into the surgery, and are seen on a first-come, first-served basis by the GP who asked them to come in. The GP may provide the patient with a prescription, which will be handed directly to the patient by the GP at the end of their consultation. If tests are required, the GP will ask the patient to speak to the receptionist on their way out to book in an appointment with the nurse to perform the test.

As well as answering calls from patients looking to make appointments, the receptionists also deal with calls enquiring about prescriptions and test results, provide prescription slips to patients, book appointments for patients with the nurse, and carry out administrative duties. The nurse, in addition to performing tests for patients, also runs a daily 2 hour weight loss clinic, providing advice to anyone who wants to speak to them. People wanting to attend the weight loss clinic simply arrive at the surgery during the time of the clinic and wait to be seen by the nurse.

The GP partners have approached you as they are very pleased with how well the weight loss clinic is going, and would like to set up nurse-led quit smoking consultations within this clinic too. However, they're unsure whether the nurse they have currently has the capacity to lead these consultations, or whether they'd need to bring in a second nurse to run a dedicated separate quit smoking clinic help to avoid lengthy waits to see the nurse for tests and clinics. They have asked you to build a model to help them better understand this.

You have 90 minutes to complete this exercise.

SimPy

SimPy is a Python library that provides a framework for building Discrete Event Simulations.

It has already defined the concepts of entities, activities, inter-arrival times etc so that we don't have to write these things from scratch.

SimPy is built around something in Python called *Generator Functions*. Let's talk about what these are.

Generator Functions

Conventional functions in Python are called, and then run with some (optional) inputs, and then finish (usually by returning some output).

Generator functions remember where they were and what they did when control is passed back (they retain their “local state”), so that they can continue where they left off, and can be used as powerful *iterators* (for and while loops are other examples of *iterators*).

This is *very* useful where we want state to be maintained (e.g. during a simulation run)

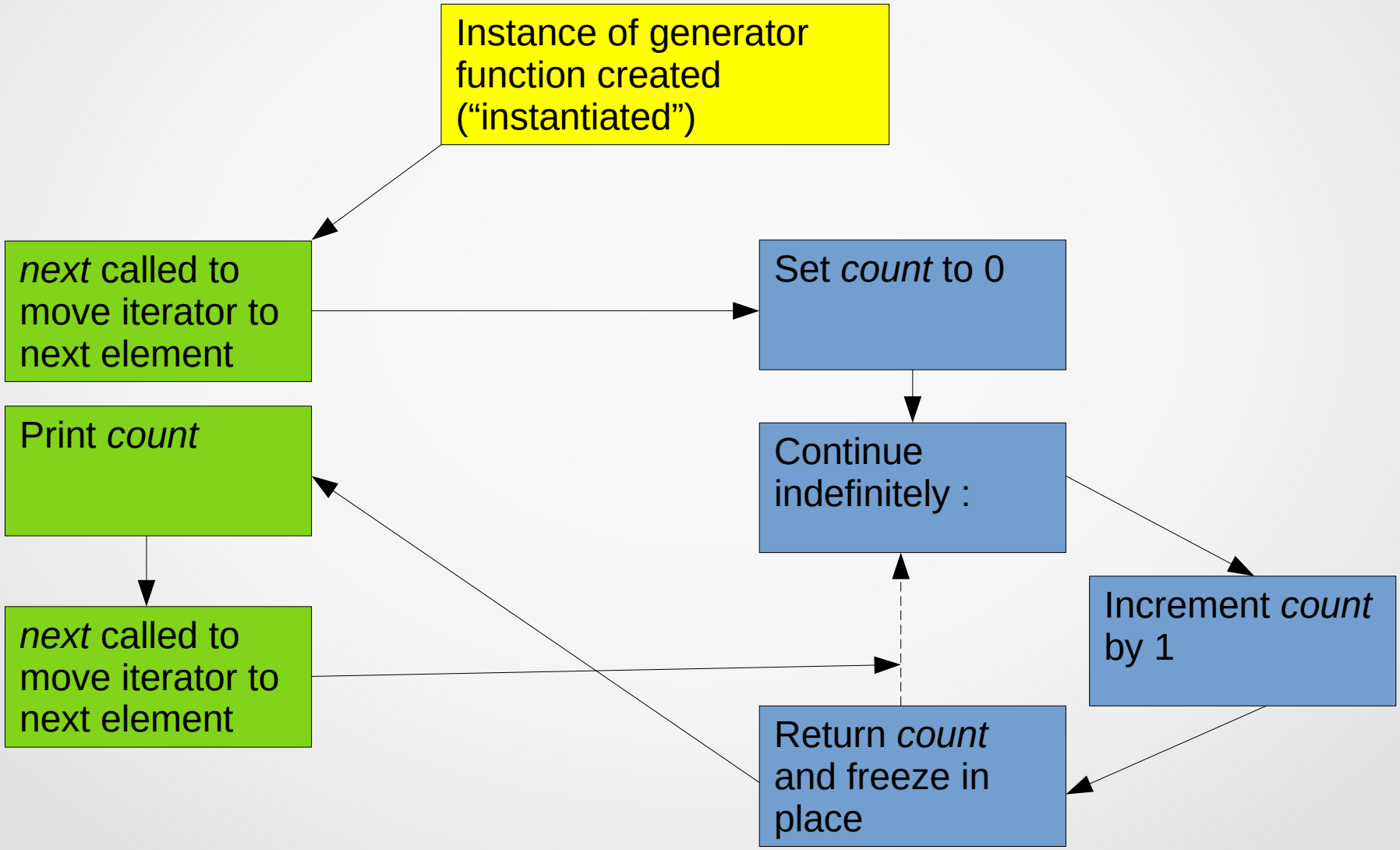
Let’s look at a very simple example of a generator function to see how they work.

```python
# We define a generator function in the same way as a conventional function
def keep_count_generator():
    # We'll set count to 0
    count = 0

    # Keep doing this indefinitely
    while True:
        # Add 1 to the count
        count += 1

        # The 'yield' statement identifies this as a generator function.
        # Yield is like return, but it tells the generator function to freeze
        # in place and remember where it was ready for the next time it's
        # called
        yield count

# We run a generator function a little differently than a conventional
# function.  Here, we create an 'instance' of the generator function, and then
# we can work with that instance.
my_generator = keep_count_generator()

# Let's print the output of the generator function 5 times.  The 'next'
# statement is used to move to the next element of the iterator.  Here, that
# means the generator unfreezes and carries on until it hits another yield
# statement.
# I've not put these print statements in a for loop to make it clear what's
# happening.
print (next(my_generator))
print (next(my_generator))
print (next(my_generator))
print (next(my_generator))
print (next(my_generator))

# The above wouldn't work with a conventional function - every time the
# function is called, it would reset the count to 0, add 1 and then return
# a value of 1
def conventional_keep_count():
    count = 0

    while True:
        count += 1
        return count

a = conventional_keep_count()
print(a)
a = conventional_keep_count()
print(a)
a = conventional_keep_count()
print(a)
a = conventional_keep_count()
print(a)
a = conventional_keep_count()
print(a)
```

```
1
2
3
4
5
```

```
1
1
1
1
1
```

Generator Functions

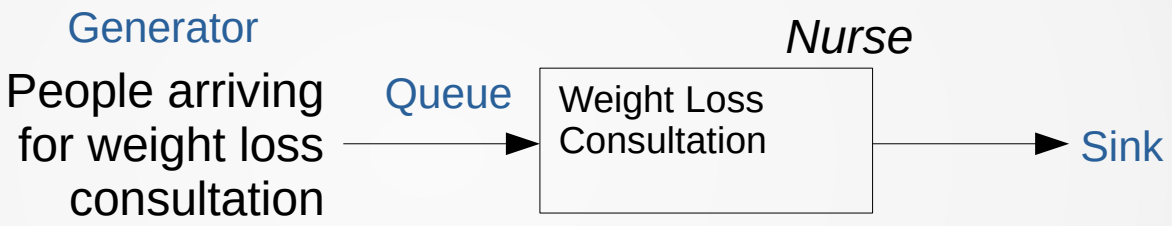


A very simple SimPy example

Inter-Arrival Times :
- Arrivals waiting for weight loss consultation

Entities :
- Patients

Activity Times :
- Time spent in weight loss consultation



Let’s look at the code for this. Let’s open simple_simpy.py in Spyder