THE UNIVERSITY OF MELBOURNE

**SLIDES + CODE**
> GitHub
> koendegeling
> RforHTA2021_simmer

# Discrete Event Simulation Using `simmer` for Models Featuring Resource Constraints

**Koen Degeling, PhD**

Cancer Health Services Research Fellow
Centre for Cancer Research & Centre for Health Policy
University of Melbourne

# Resource Constrained Health Economic Models

Social Science & Medicine 200 (2018) 59–64

Contents lists available at ScienceDirect

Social Science & Medicine

journal homepage: www.elsevier.com/locate/socscimed

## Health care input constraints and cost effectiveness analysis decision rules

Pieter van Baal[a,*], Alec Morton[b], Johan L. Severens[a]

[a] Erasmus University Rotterdam, Erasmus School of Health Policy & Management, Rotterdam, The Netherlands
[b] University of Strathclyde, Department of Management Science, Glasgow, United Kingdom

ARTICLE INFO

*Keywords:*
Cost-effectiveness analysis
Human resource constraints
Decision rules
Health care input constraints
Opportunity costs
Eye-care services
LMIC

ABSTRACT

Results of cost effectiveness analyses (CEA) studies are most useful for decision makers if they face only one constraint: the health care budget. However, in practice, decision makers wishing to use the results of CEA studies may face multiple resource constraints relating to, for instance, constraints in health care inputs such as a shortage of skilled labour. The presence of multiple resource constraints influences the decision rules of CEA and limits the usefulness of traditional CEA studies for decision makers. The goal of this paper is to illustrate how results of CEA can be interpreted and used in case a decision maker faces a health care input constraint.

We set up a theoretical model describing the optimal allocation of the health care budget in the presence of a health care input constraint. Insights derived from that model were used to analyse a stylized example based on a decision about a surgical robot as well as a published cost effectiveness study on eye care services in Zambia.

Our theoretical model shows that applying default decision rules in the presence of a health care input constraint leads to suboptimal decisions but that there are ways of preserving the traditional decision rules of CEA by reweighing different cost categories. The examples illustrate how such adjustments can be made, and makes clear that optimal decisions depend crucially on such adjustments.

We conclude that it is possible to use the results of cost effectiveness studies in the presence of health care input constraints if results are properly adjusted.

PharmacoEconomics (2019) 37:1011–1027
https://doi.org/10.1007/s40273-019-00801-9

SYSTEMATIC REVIEW

## Accounting for Capacity Constraints in Economic Evaluations of Precision Medicine: A Systematic Review

Stuart J. Wright[1] · William G. Newman[2,3] · Katherine Payne[1]

Published online: 13 May 2019


**Abstract**

**Background and Objective** Precision (stratified or personalised) medicine is underpinned by the premise that it is feasible to identify known heterogeneity using a specific test or algorithm in patient populations and to use this information to guide patient care to improve health and well-being. This study aimed to understand if, and how, previous economic evaluations of precision medicine had taken account of the impact of capacity constraints.

**Methods** A meta-review was conducted of published systematic reviews of economic evaluations of precision medicine (test–treat interventions) and individual studies included in these reviews. Due to the volume of studies identified, a sample of papers published from 2007 to 2015 was collated. A narrative analysis identified whether potential capacity constraints were discussed qualitatively in the studies and, if relevant, which quantitative methods were used to account for capacity constraints.

**Results** A total of 45 systematic reviews of economic evaluations of precision medicine were identified, from which 222 studies focusing on test–treat interventions, published between 2007 and 2015, were extracted. Of these studies, 33 (15%) qualitatively discussed the potential impact of capacity constraints, including budget constraints; quality of tests and the testing process; ease of use of tests in clinical practice; and decision uncertainty. Quantitative methods (nine studies) to account for capacity constraints included static methods such as capturing inefficiencies in trials or models and sensitivity analysis around model parameters; and dynamic methods, which allow the impact of capacity constraints on cost effectiveness to change over time.

**Conclusions** Understanding the cost effectiveness of precision medicine is necessary, but not sufficient, evidence for its successful implementation. There are currently few examples of evaluations that have quantified the impact of capacity constraints, which suggests an area of focus for future research.

# The `simmer` Package



- Developed by Iñaki Ucar and Bart Smeets
- First release on CRAN in 2015 with regular updates
- Generic framework like SimPy and SimJulia, with backend in C++
- Process-oriented and trajectory-based models including resources
- Chaining/piping workflow introduced by the `magrittr` package
- Extensive information, tutorials, and extensions (https://r-simmer.org/)
  - `simmer.plot`
  - `simmer.bricks`
  - `simmer.optim`
  - and more…

# Trajectories and Simulations

**Trajectories:** `trajectory()`

- Process through which agents flow
- Define what events can happen
- Define when those events happen
- Define which resources are utilized
- Define when resources are utilized
- Define for how long resources are utilized
- Define when agent attributes are updated

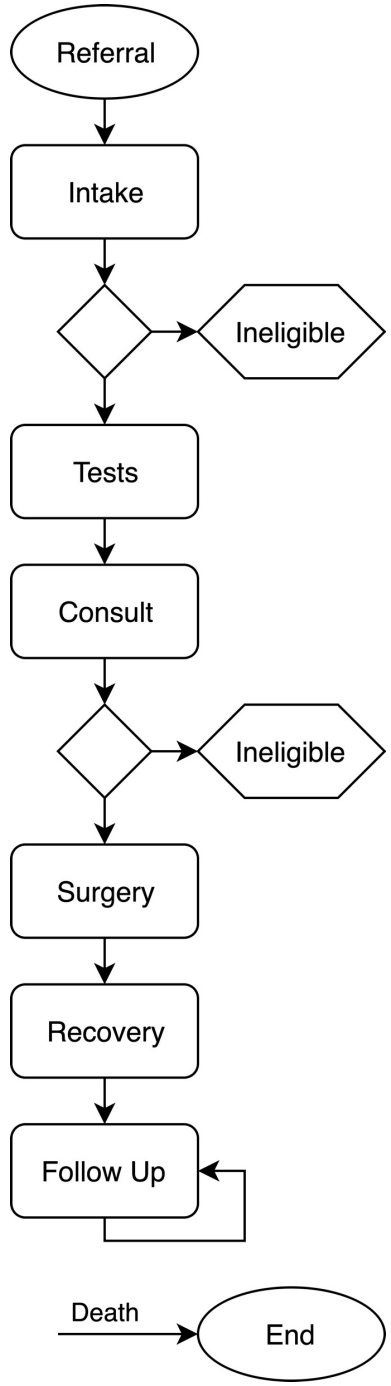
**Simulations:** `simmer()`

- Contain the state of the system
- Define the number of agents
- Define the (inter)arrival times
- Define the trajectory that is used
- Define the amount/schedule of resources
- Define the queue sizes
- Define the level of monitoring

# Hypothetical Case Study

- Cochlear implants can result in substantial better hearing for individuals with hearing loss, resulting in improved quality of life
- **Hypothetical case study:**
  - Maximum number of implants/surgical procedures: *104 per year / 2 per week*
  - Demand for cochlear implants exceeds this capacity: *6 months waiting period*
  - Increased time between referral and surgery results in lower QoL afterwards
- **Simulation objectives:** For 5 years worth of newly referred individuals…
  - Estimate how waiting times and quality-adjusted life years (QALYs) will develop
  - Estimate the impact of two alternative scenarios on waiting times and QALYs:
    > Increase capacity by 50% to 3 surgeries per week
    > Double capacity (4 surgeries) for 2 years and then fix at 3 surgeries per week

*Inspired by a current project of Hugo Nijmeijer, Dr Wendy Huinck, and Dr Emmanuel Mylanus from the Radboud UMC in the Netherlands, but with hypothetical parameters.*



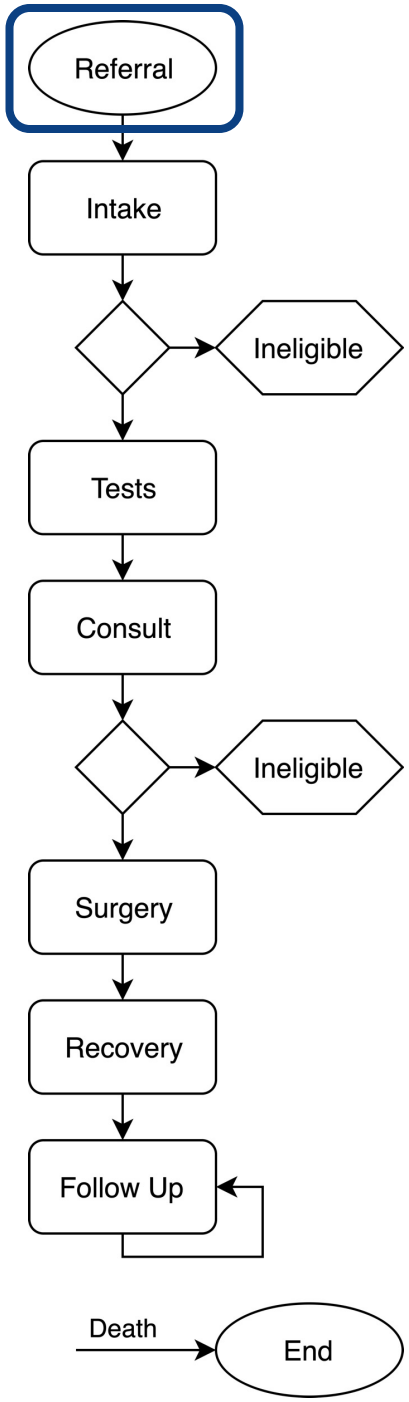
```
trj_main <- trajectory() %>%
  set_attribute(keys = "TimeOfReferral", values = function() now(.env = sim)) %>%
  renege_in(t = function() now(.env = sim) + rgompertz(1, d_death_shape, d_death_rate), out = trj_end) %>%
```

`set_attribute()` Record or update individual-level attributes

- Essential arguments:
  - `keys` character vector of the names of the attributes to be set/update
  - `values` numerical vector of values to/with which attributes are to be set/updated
  - `mod` character defining if it concerns a recording 'NA' or update, e.g. '+' or '-'
- Note that attributes can be numerical only
- The function to set global variables is `set_global()`

`now()` Obtain the current simulation time of the simulation defined by `.env`

`renege_in()` Schedule an event to occur at a certain point in time



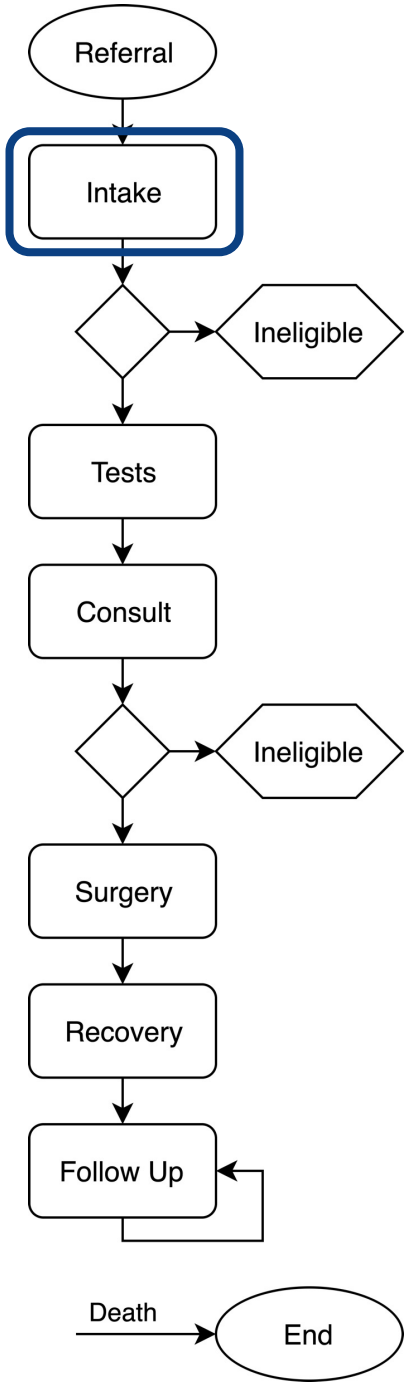
```
seize(resource = "Intake") %>%

timeout(task = t_intake) %>%

release(resource = "Intake") %>%
```

`seize()/release()` Seize or release a resource once it is available

- Essential arguments:
  - `resource` character defining the resource that is to be seized/released
  - `amount` numeric defining the amount of resource to be seized/released
- If there are no resources available, the individual enters the queue, settings for which are specified in the simulation environment

`timeout()` Delay the individual for a certain amount of time

- Essential arguments:
  - `task` numeric defining the duration for which the individual is to be delayed
- Note that the modeler is responsible for ensuring time is defined consistently
- The function `timeout_from_attribute()` takes the time from an attribute

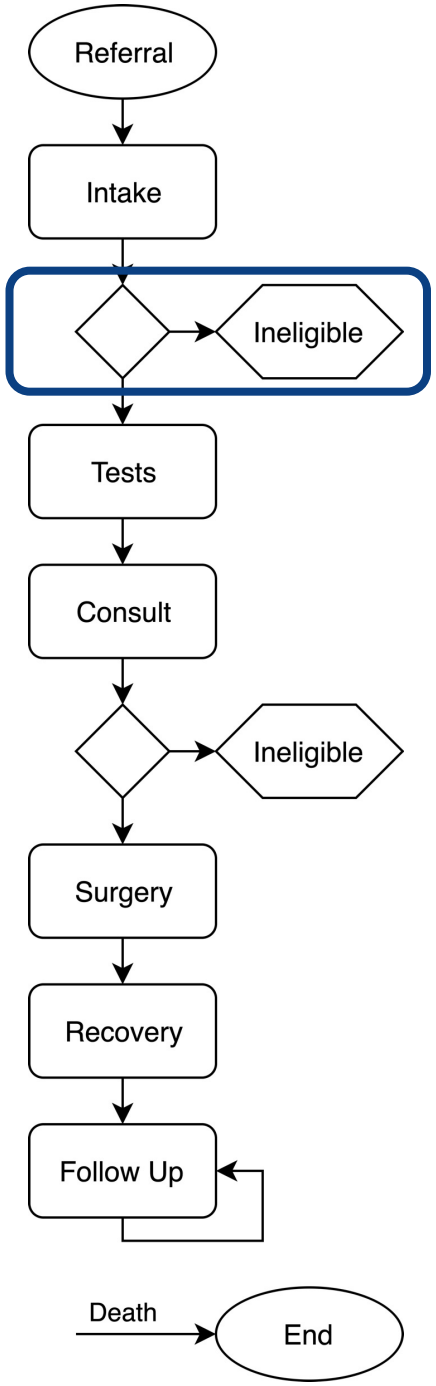


```
#   0) continue to testing (i.e., skip the branch)
#   1) not eligible (i.e., wait until the individual is transferred to trj_end)
branch(option = function() fn_eligible_intake(), continue = c(F),

       # 1) not eligible
       trajectory() %>%
         set_attribute(keys = "Rejected", values = 1) %>%
         wait()

) %>%
```

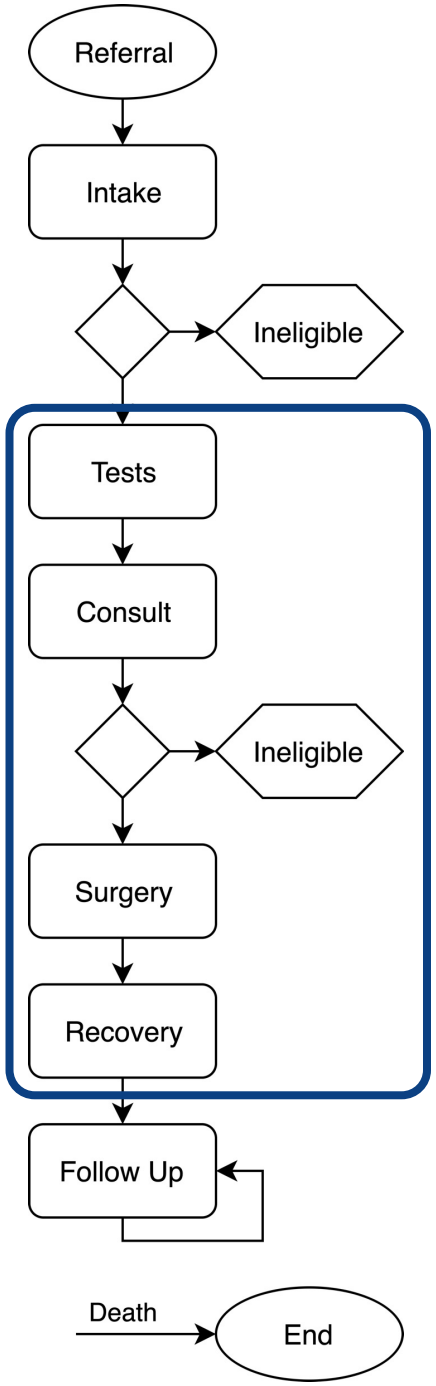
`branch()` Direct the individuals to alternative sub-trajectories

- Essential arguments:
  - `option` numeric defining the sub-trajectory to direct the individual to
  - `continue` logical defining whether to continue beyond the branch
  - … the sub-trajectories separated by commas
- The branch can be skipped by returning `0` (i.e., zero) to the `option` argument

`wait()` Delay until a certain signal is received, e.g. from `renege_in()`



```r
# Testing
seize(resource = "Testing") %>%
timeout(task = t_testing) %>%
release(resource = "Testing") %>%

# Final consult
seize(resource = "Consult") %>%
timeout(task = t_consult) %>%
release(resource = "Consult") %>%

# 0) continue to testing (i.e., skip the branch)
# 1) not eligible (i.e., wait until the individual is transferred to trj_end)
branch(option = function() fn_eligible_consult(), continue = c(F),

       # 1) not eligible
       trajectory() %>%
         set_attribute(keys = "Rejected", values = 1) %>%
         wait()

) %>%

# Surgery
seize(resource = "Surgery") %>%
set_attribute(keys = "TimeOfSurgery", values = function() now(.env = sim)) %>%
timeout(task = t_surgery) %>%
release(resource = "Surgery") %>%

# Recovery
seize(resource = "Recovery") %>%
timeout(task = t_recovery) %>%
release(resource = "Recovery") %>%
```

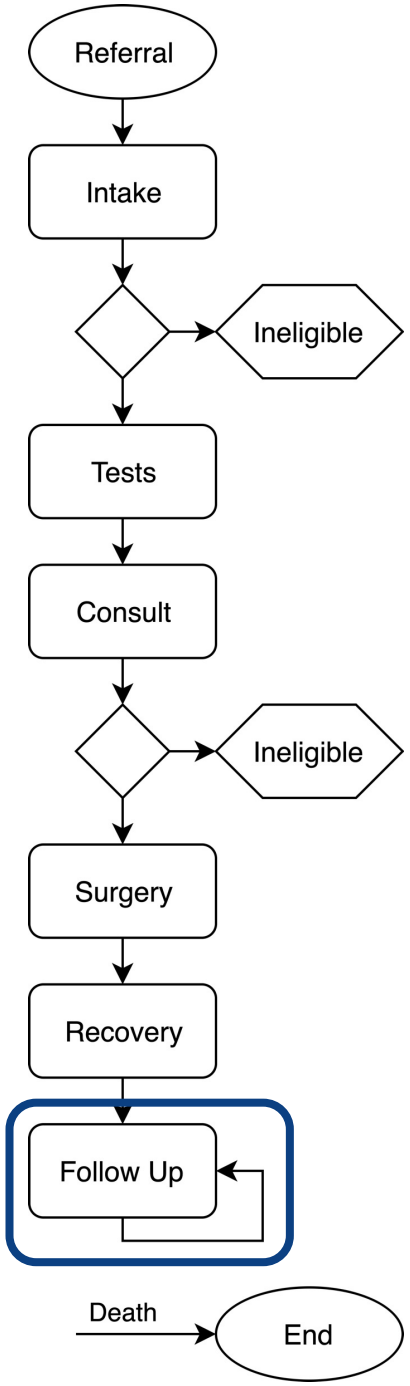


```
seize(resource = "FollowUp") %>%
set_attribute(keys = "FollowUpCount", values = 1, mod = "+") %>%
timeout(task = t_followup) %>%
release(resource = "FollowUp") %>%

timeout(task = t_next_followup) %>%
rollback(amount = 5)
```

`rollback()` Direct the individual a certain amount of steps back in the trajectory

- Essential arguments:
  - `amount` numerical defining the number of steps to go back
  - `times` numerical defining the maximum times the individual can roll back
  - `check` logical-returning function to indicate whether a rollback is allowed
- Ensure to `plot()` the trajectory to visually check the rollback amount is right

Also note the use of the `mod = '+'` argument in the `set_attribute()` function.

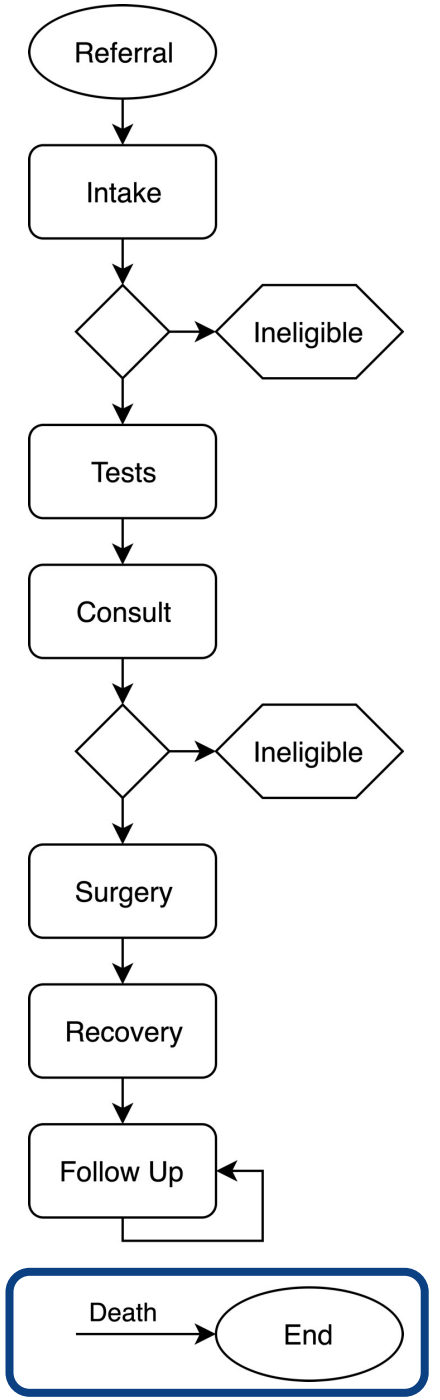


```
trj_end <- trajectory() %>%
  set_attribute(
    keys   = c("TimeOfDeath", "TimeToDeath", "TimeToSurgery", "UtilityAfterSurgery", "QALWs", "dQALWs"),
    values = function() fn_calculate_impact(
      CurrentTime = now(.env = sim),
      Attrs       = get_attribute(.env = sim, keys = c("TimeOfReferral", "TimeOfSurgery"))
    )
  )
```

`get_attribute()` Read the values of individual-level attributes

- Essential arguments:
  - `.env` the `simmer()` simulation environment monitoring the attribute
  - `keys` character vector defining the names of the attributes to be read
- The function to read global variables is `get_global()`

Also note how multiple attributes are set at once.



# More About Trajectories

- Static vs. dynamic function calls/values: importantly, without using the `function()` statement, expressions are only evaluated once when the `trajectory()` object is defined.
  - This apply to all functions used in trajectories
  - For example:

```
# One value will be sampled and used for all individuals
timeout(task = rweibull(1, 1.2, 10))

# A value will be sampled for each individual separately
timeout(task = function() rweibull(1, 1.2, 10))
```

- Other useful functions are:
  - `join()` Join trajectories together
  - `log_()` Print a message to the console (useful for debugging)
  - See the `simmer` documentation/website for other functions

```
sim <- simmer() %>%
  add_resource(name = "Intake", capacity = Inf) %>%
  add_resource(name = "Testing", capacity = Inf) %>%
  add_resource(name = "Consult", capacity = Inf) %>%
  add_resource(name = "Surgery", capacity = schedule(timetable = c(0, 26), values = c(0, 2))) %>%
  add_resource(name = "Recovery", capacity = Inf) %>%
  add_resource(name = "FollowUp", capacity = Inf) %>%
  add_generator(name_prefix = "Ind", trajectory = trj_main, mon = 2,
                distribution = to(stop_time = 5*52, dist = function() rexp(n = 1, rate = r_referral)))
```

`add_resource()` Define a resource to be available in the simulation

- Essential arguments:
  - `name` character defining the name of the resource (should correspond to trajectory)
  - `capacity` integer or `schedule()` defining the amount of resources available
  - `queue_size` integer or `schedule()` defining the maximum size of the resource queue

`schedule()` Function to define changes in resources and queues over time

- Essential arguments:
  - `timetable` numeric vector of time points at which the value is to change
  - `values` integer vector of desired value for each point in time

```
sim <- simmer() %>%
  add_resource(name = "Intake", capacity = Inf) %>%
  add_resource(name = "Testing", capacity = Inf) %>%
  add_resource(name = "Consult", capacity = Inf) %>%
  add_resource(name = "Surgery", capacity = schedule(timetable = c(0, 26), values = c(0, 2))) %>%
  add_resource(name = "Recovery", capacity = Inf) %>%
  add_resource(name = "FollowUp", capacity = Inf) %>%
  add_generator(name_prefix = "Ind", trajectory = trj_main, mon = 2,
                distribution = to(stop_time = 5*52, dist = function() rexp(n = 1, rate = r_referral)))
```

`add_generator()` Specify how individuals are to be simulated through a certain trajectory

- Essential arguments:
  - `name_prefix` character used as a prefix for the individuals' names
  - `trajectory` `trajectory()` object through which the individuals are to be simulated
  - `distribution` function returning a numeric representing an interarrival time
  - `mon` integer defining the level of monitoring (0 = none, 1 = arrival, 2 = attributes)

`to()` Convenience function to generate interarrival times until a certain time point

- Essential arguments:
  - `stop_time` numeric indicating when individuals should no longer be generated
  - `dist` function returning a numeric representing an interarrival time
- Other convenience functions are `at()`, `from()`, and `from_to()`

```r
set.seed(123); sim %>% reset() %>% run();

df_scen1_attributes <- get_mon_attributes(sim)
df_scen1_arrivals   <- get_mon_arrivals(sim)
df_scen1_resources  <- get_mon_resources(sim)

df_scen1_summary <- fn_summarise(df_scen1_attributes)
```

```r
> head(df_scen1_attributes)
       time name            key     value replication
1 0.2008232 Ind0 TimeOfReferral 0.2008232           1
2 0.3381113 Ind1 TimeOfReferral 0.3381113           1
3 0.6545530 Ind2 TimeOfReferral 0.6545530           1
4 0.6620714 Ind3 TimeOfReferral 0.6620714           1
5 0.6754549 Ind4 TimeOfReferral 0.6754549           1
6 0.7508124 Ind5 TimeOfReferral 0.7508124           1
```

| | |
|---|---|
| `get_mon_attributes()` | Extract the values of the individual-level attributes over time |
| `get_mon_arrivals()` | Extract information on individuals' start, end, and activity time |
| `get_mon_resources()` | Extract information on all resource-related events |

Note that the resulting `data.frame` is in long format, tracking each event for each entity

| | |
|---|---|
| `fn_summarise()` | Custom function that summarises the output from a call of `get_mon_attributes()` into wide format using the last recorded value for the attributes of interest. |

# Strategies

- Remember the current waiting period of 6 months (or 26 weeks)

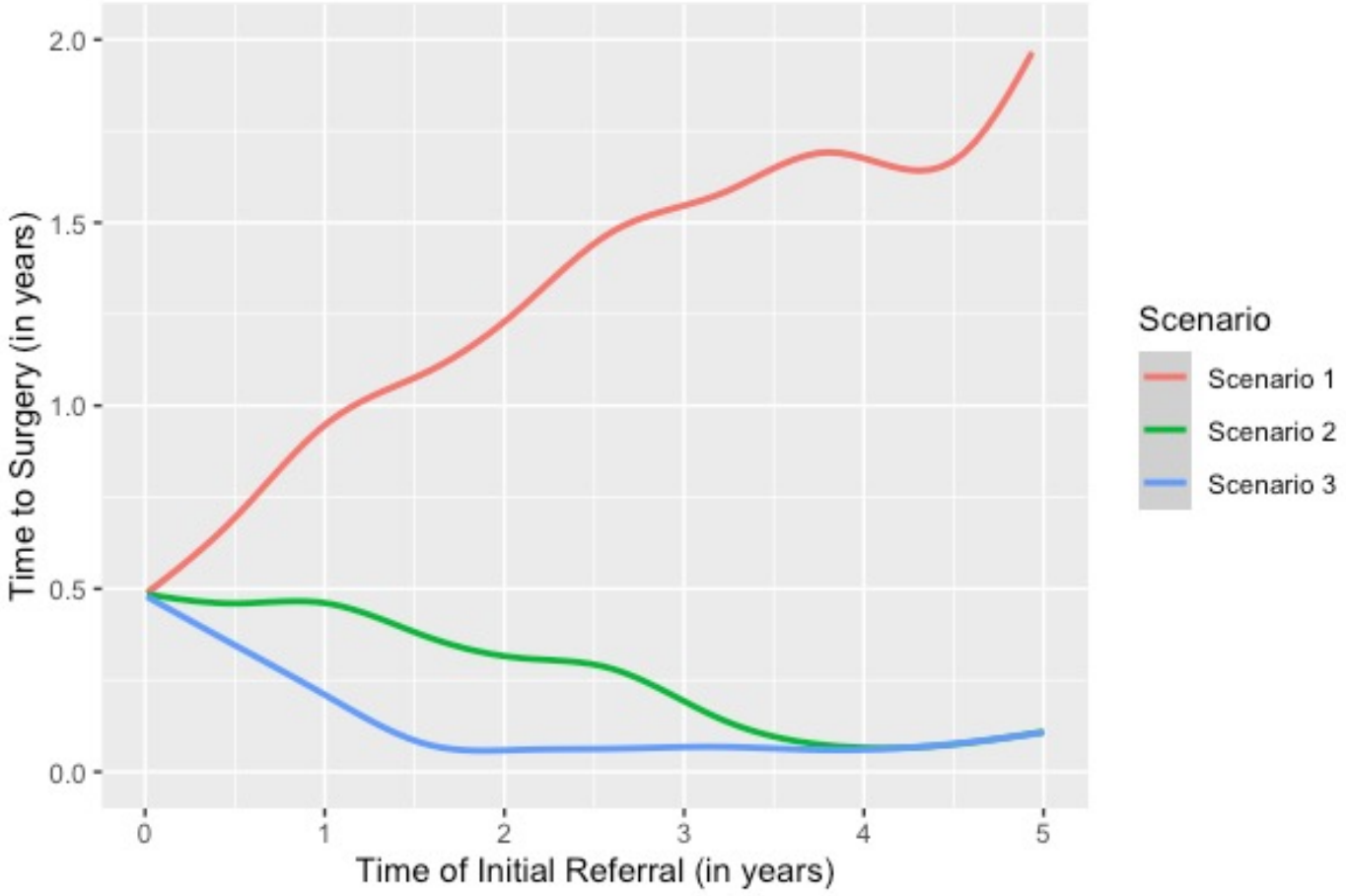
- **Strategy 1:** current scenario in which 104 surgeries (or 2 per week) can be performed per year
  - schedule(timetable = c(0, 26), values = c(0, 2))

- **Strategy 2:** increase the surgery capacity by 50% to 3 surgeries per week
  - schedule(timetable = c(0, 26), values = c(0, 3))

- **Strategy 3:** double the capacity (4 surgeries per week) for 2 years and then fix it at 3 surgeries per week
  - schedule(timetable = c(0, 26, 130), values = c(0, 4, 3))

# Findings

| Scenario | LYs | Years to Surgery | Utility after Surgery | QALYs |
|---|---|---|---|---|
| 1 | 23.5 | 1.29 | 0.60 | 9.9 |
| 2 | 23.5 | 0.27 | 0.73 | 11.3 |
| 3 | 23.5 | 0.14 | 0.81 | 12.1 |



# Discussion

- The `simmer` package provides a relatively simple, though highly flexible framework for implementing resource constrained discrete event simulations in `R`
  - Set of basic functions that directly translate to conceptual model structures
  - This process-oriented/trajectory-based approach makes it great for those new to code-based DES

- `simmer` vs. `base R`
  - Implementing resource-constrained models using `base R` is quite challenging
  - Code for large/complex `simmer` models may become somewhat more challenging to manage
  - Debugging `simmer` models is challenging, incremental approach toward development essential
  - Models without resource constraints may be faster in `base R` compared to `simmer`

THE UNIVERSITY OF MELBOURNE

**SLIDES + CODE**
> GitHub
> koendegeling
> RforHTA2021_simmer

# Thank you!

@k_degeling
koen.degeling@unimelb.edu.au