Discrete Event Simulation in R to Support Healthcare Decision Making

Using `simmer` for Discrete Event Simulation in `R`

H. Koffijberg, PhD, MSc

University of Twente
Enschede, The Netherlands

K. Degeling, PhD, MSc

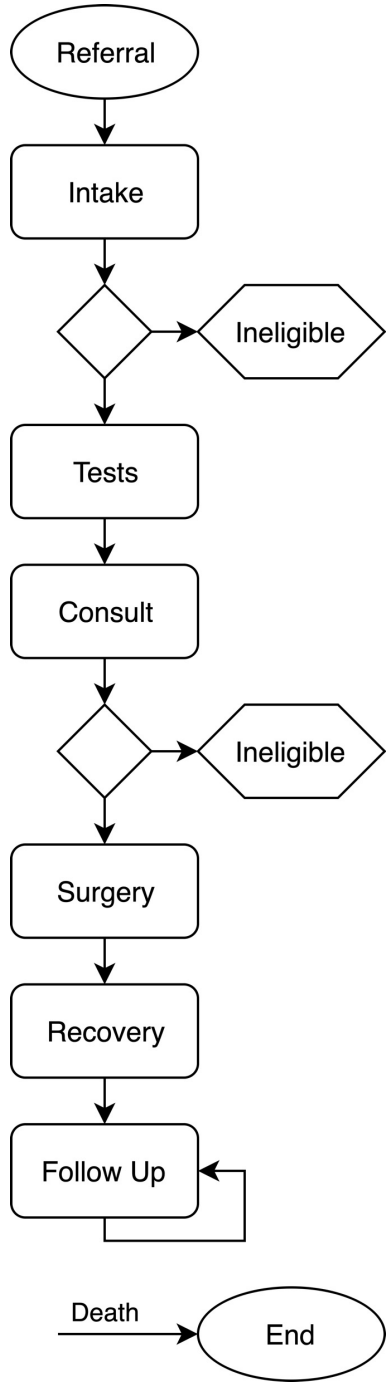
University of Melbourne
Melbourne, Australia

M. van de Ven, MSc

OPEN Health
Rotterdam, The Netherlands

Hypothetical Case Study

- **Hypothetical case study:**
  - Pathway for someone who is referred for a knee replacement
  - Low quality of life before surgery (0.6), higher after (0.9)
  - Current standard of care:
    - 70% eligible at the intake, 90% at the final consult
    - Surgery costs 8,132, recovery costs 4,576
  - Experimental strategy:
    - 77% eligible at the intake, 94.5% at the final consult
    - Surgery costs 12,716, recovery costs 6,329
- **Simulation modelling objectives:**
  - Simulation model of the pathway
  - Health economic impact to compare strategies



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

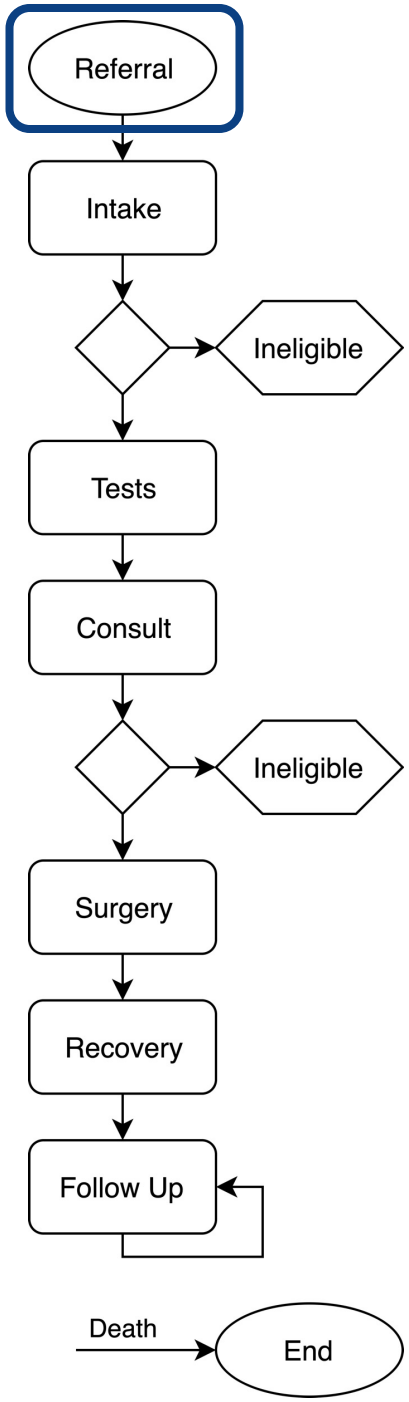
```
trj_main <- trajectory() %>%
  set_attribute(keys = "TimeOfReferral", values = function() now(.env = sim)) %>%
  renege_in(t = function() now(.env = sim) + rgompertz(1, d_death_shape, d_death_rate), out = trj_end) %>%
```

`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`

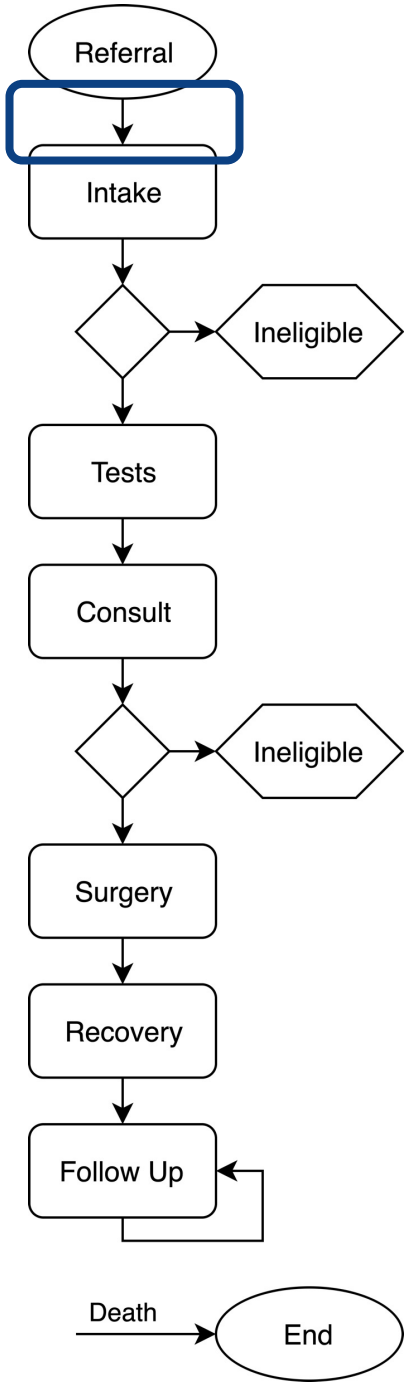
`renege_in()` Schedule an event to occur at a certain point in time



```
timeout(task = function() rweibull(1, d_intake_shape, d_intake_scale)) %>%
```

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

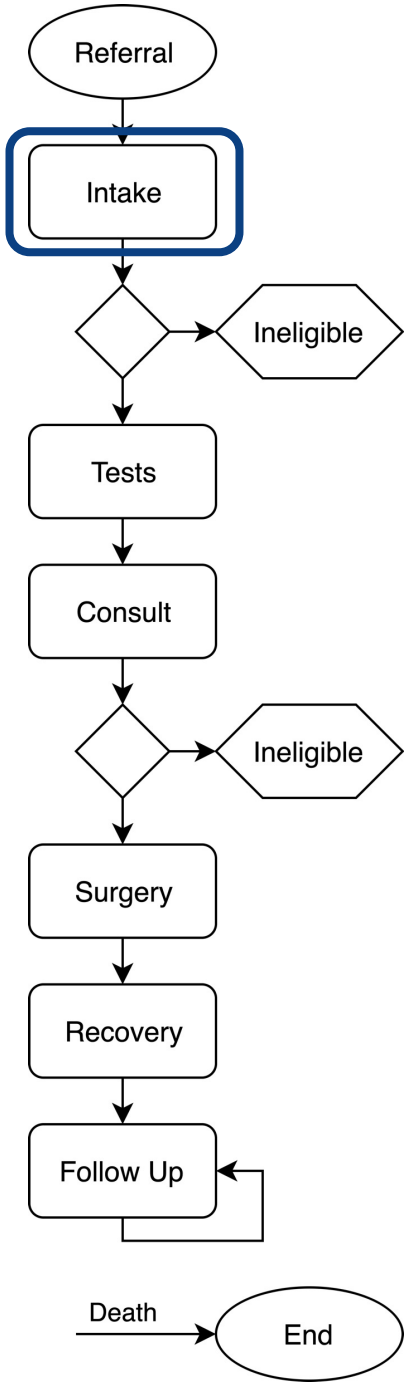
- Essential arguments:
  - `task` numeric defining the duration for which the individual is to be delayed
- Note that the modeler is responsible for ensuring time is defined consistently
- The function `timeout_from_attribute()` takes the time from an attribute



```
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

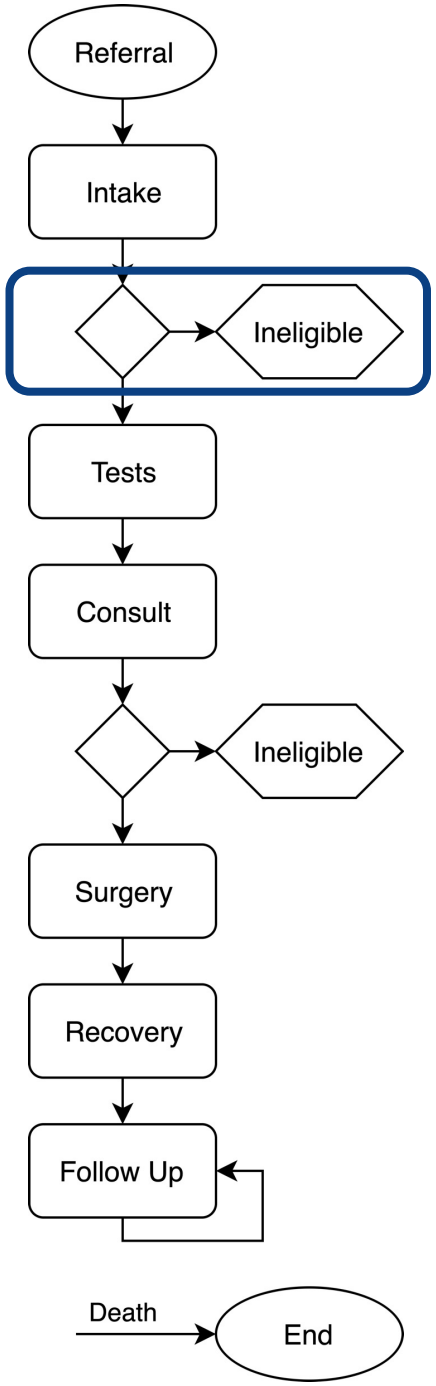


```
#   0) continue to testing (i.e., skip the branch)
#   1) not eligible (i.e., wait until the individual is transferred to trj_end)
branch(option = function() fn_eligible_intake(), continue = c(F),

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

) %>%
```

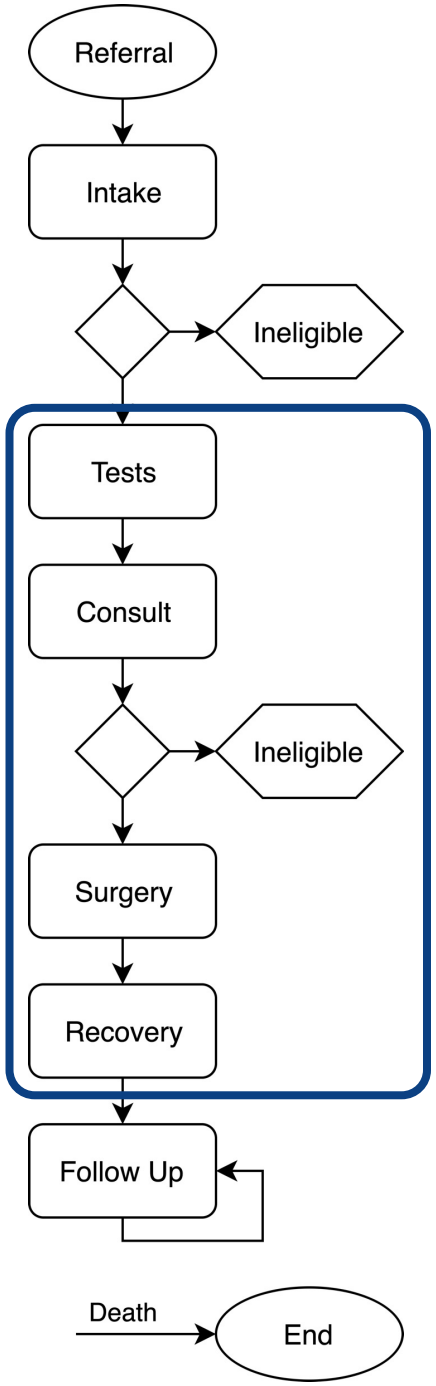
`branch()` Direct the individuals to alternative sub-trajectories

- 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()`



```
# Time to next event: Testing
timeout(task = function() rweibull(1, d_testing_shape, d_testing_scale)) %>%

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

# Time to next event: Consult
timeout(task = function() rweibull(1, d_consult_shape, d_consult_scale)) %>%

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

# 0) continue to testing or 1) not eligible
branch(option = function() fn_eligible_consult(), continue = c(F),

       trajectory() %>%
         set_attribute(keys = "Rejected", values = 1) %>%
         wait()
) %>%

# Time to next event: Surgery
timeout(task = function() rweibull(1, d_surgery_shape, d_surgery_scale)) %>%

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

# Time to next event: Recovery
timeout(task = function() rweibull(1, d_recovery_shape, d_recovery_scale)) %>%

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

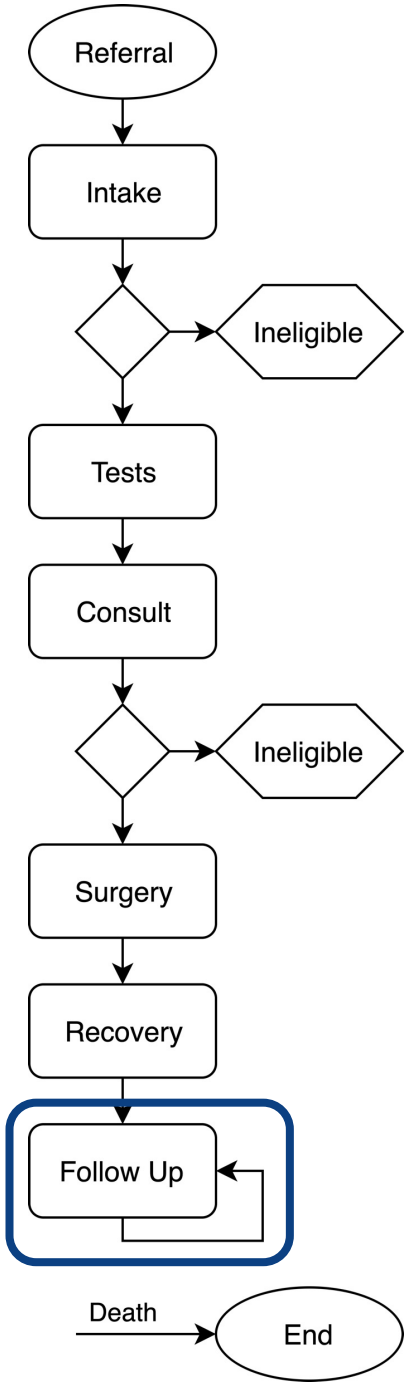


```
# Time to next event: Follow up
timeout(task = function() rweibull(1, d_followup_shape, d_followup_scale)) %>%

# Follow-up visit
seize(resource = "FollowUp") %>%
set_attribute(keys = "FollowUpCount", values = 1, mod = "+") %>%
timeout(task = t_followup) %>%
release(resource = "FollowUp") %>%
rollback(amount = 5, times = n_followup_rounds - 1) %>%
```

`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.

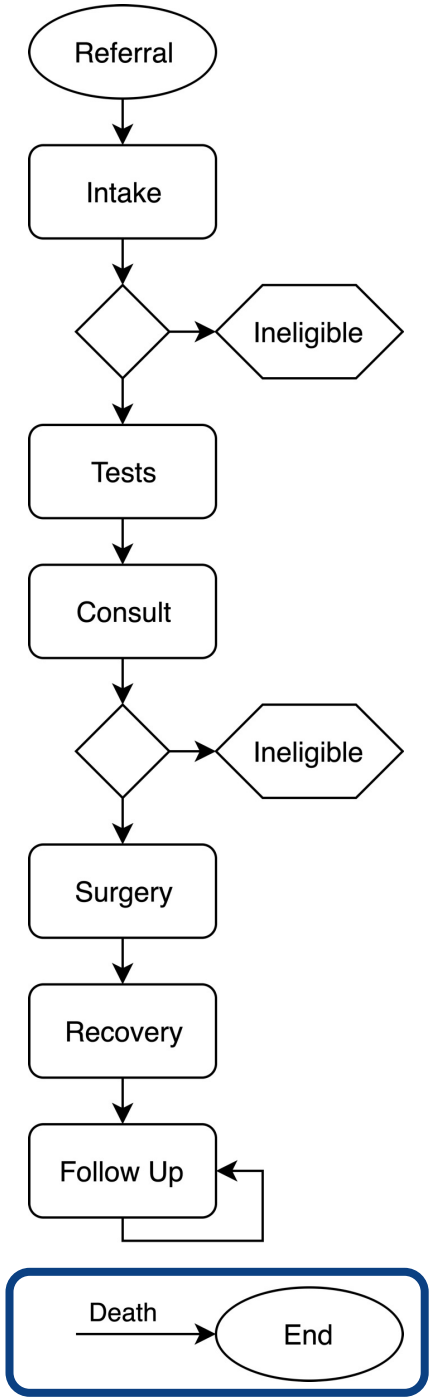


```
trj_end <- trajectory() %>%
  set_attribute(
    keys   = c("TimeOfDeath", "TimeToDeath", "TimeToSurgery"),
    values = function() fn_calculate_impact(
      CurrentTime = now(.env = sim),
      Attrs       = get_attribute(.env = sim, keys = c("TimeOfReferral", "TimeOfSurgery"))
    )
  )
```

`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 = Inf) %>%
  add_resource(name = "Recovery", capacity = Inf) %>%
  add_resource(name = "FollowUp", capacity = Inf) %>%
  add_generator(name_prefix = "Ind", trajectory = trj_main, mon = 2,
                distribution = at(rep(x = 0, times = n_individuals)))
```

`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 = Inf) %>%
  add_resource(name = "Recovery", capacity = Inf) %>%
  add_resource(name = "FollowUp", capacity = Inf) %>%
  add_generator(name_prefix = "Ind", trajectory = trj_main, mon = 2,
                distribution = at(rep(x = 0, times = n_individuals)))
```

`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)

`at()` Convenience function to generate arrival times rather than interarrival times

- Essential arguments:
  - … numeric vector of arrival times
- Other convenience functions are `t0()`, `from()`, and `from_to()`

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

df_attributes <- get_mon_attributes(sim)
df_arrivals   <- get_mon_arrivals(sim)
df_resources  <- get_mon_resources(sim)

df_out <- fn_summarise(df_attributes)
```

```
> head(df_attributes)
  time name            key value replication
1    0 Ind0 TimeOfReferral     0           1
2    0 Ind1 TimeOfReferral     0           1
3    0 Ind2 TimeOfReferral     0           1
4    0 Ind3 TimeOfReferral     0           1
5    0 Ind4 TimeOfReferral     0           1
6    0 Ind5 TimeOfReferral     0           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.

Case Study Files on GitHub

- This pathway, without health economic outcomes, is implemented in the `1_basic_structure.R` script

- The remainder of this demonstration will be about implementing the costs and utility values to obtain quality-adjusted life years, as is demonstrated in the `2_health_economics.R` script

- In the second demonstration, the 3_probabilistic_analysis.R script will be discussed to demonstrate how a probabilistic analysis can be implemented to quantify the impact of parameter uncertainty