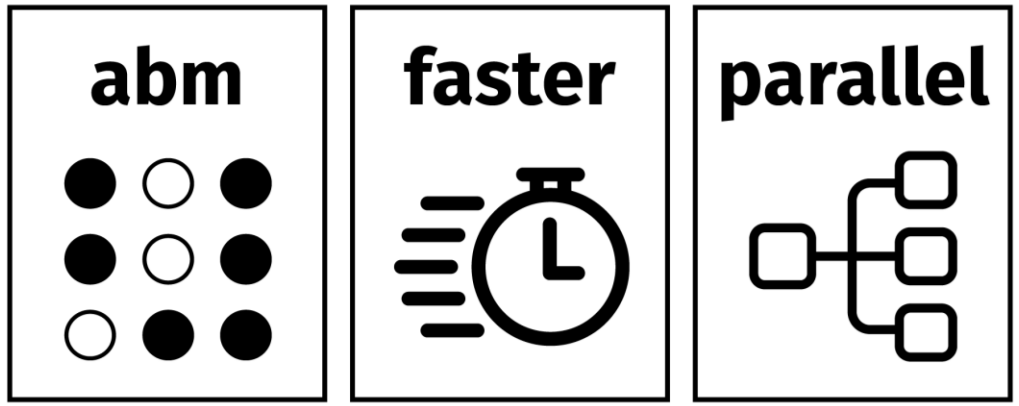


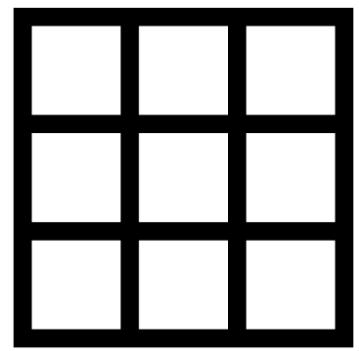
Hands-on 1 discussion

Supercomputing with R

Supercomputing structure

1. Create a **parameter grid** (data frame)
2. Create an **analysis function** that takes in a row of the parameter grid and outputs a result
3. Create a **self-contained job script** to load a chunk of the grid and run the analysis function on several rows in parallel
4. Create a **shell script** to run the R job with SLURM parameters

grid



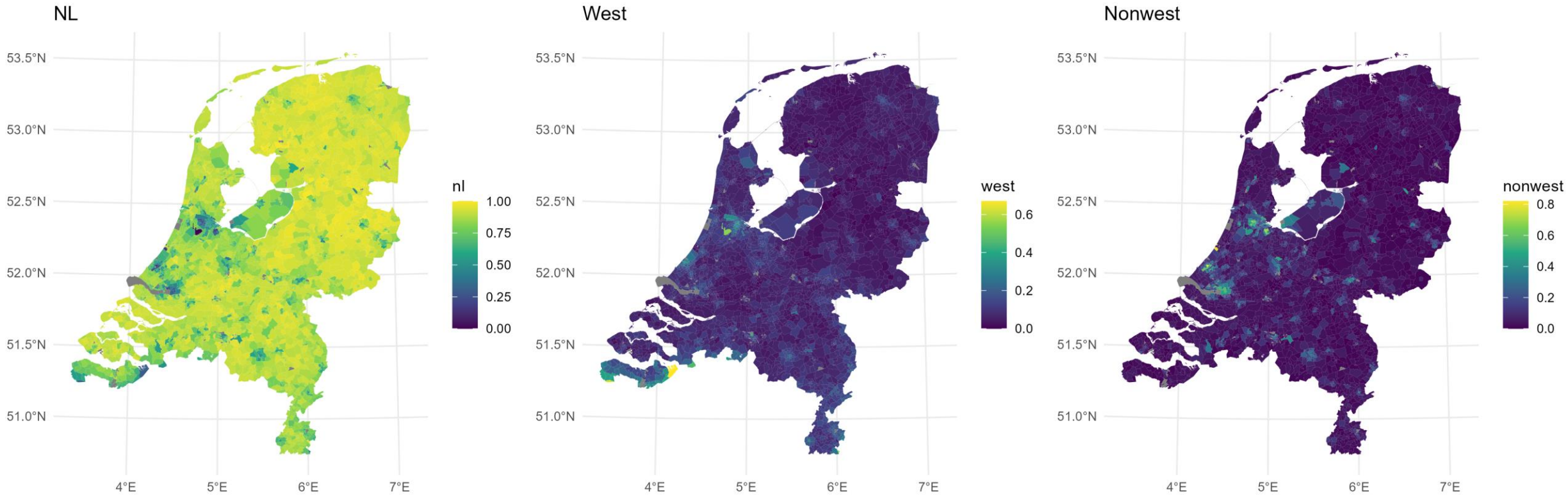
ABM for the Netherlands

- We have a good ABM implementation now.
- Let's connect our ABM to real data
- Base our **proportion parameter** on population data about neighbourhoods in NL

(fake, illustrative) research question:

What proportion of non-western migrants is "happy" with different levels of neighbourhood preference B_a?

ABM for the Netherlands



https://www.pdok.nl/introductie/-/article/cbs-wijken-en-buurten

ABM for the Netherlands

```
Simple feature collection with 3248 features and 5 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 13565.4 ymin: 306846.2 xmax: 278026.1 ymax: 619374.9
Projected CRS: Amersfoort / RD New
First 10 features:
   wijkcode       wijknaam   nl west nonwest                          geom
1  WK001400        Centrum 0.69 0.19    0.12 MULTIPOLYGON (((233335.8 58 ...
2  WK001401       Oud-Zuid 0.75 0.16    0.09 MULTIPOLYGON (((235128 5811 ...
3  WK001402       Oud-West 0.74 0.16    0.10 MULTIPOLYGON (((233335.8 58 ...
4  WK001403      Oud-Noord 0.68 0.14    0.18 MULTIPOLYGON (((234047.5 58 ...
5  WK001404 Oosterparkwijk 0.71 0.13    0.16 MULTIPOLYGON (((234689.3 58 ...
6  WK001405       Zuidoost 0.78 0.14    0.08 MULTIPOLYGON (((239416.9 57 ...
7  WK001406   Helpman e.o. 0.77 0.13    0.10 MULTIPOLYGON (((235885.3 57 ...
8  WK001407       Zuidwest 0.80 0.10    0.10 MULTIPOLYGON (((233581 5794 ...
9  WK001408  Hoogkerk e.o. 0.85 0.09    0.06 MULTIPOLYGON (((231577.2 58 ...
10 WK001409     Nieuw-West 0.68 0.14    0.18 MULTIPOLYGON (((230032.1 58 ...
```

Parameter grid

- There are **3248** neighbourhoods in NL
- We will inspect **91** different levels of Ba parameter
- For stability, we want **50** iterations to average over

3248*91*50 = 14 778 400 ABMs to run!

Tibbles and nested columns

- In the hands-on, you will go through the grid code
- This is just 1 version / implementation
- There are other ways to create the grid (probably faster, too)

- End result: one row per desired result

Tibbles and nested columns

- Useful function: `expand_grid()`

```
expand_grid(
  theta = c(1, 2, 3),
  phi = c("A", "B", "C"),
  rho = c(0.1, 0.15, 0.20)
)

# A tibble: 27 x 3
   theta phi     rho
   <dbl> <chr> <dbl>
 1     1 A      0.1
 2     1 A      0.15
 3     1 A      0.2
 4     1 B      0.1
 5     1 B      0.15
 6     1 B      0.2
 7     1 C      0.1
 8     1 C      0.15
 9     1 C      0.2
10     2 A      0.1
# ... with 17 more rows
```

Tibbles and nested columns

- We will use tibbles with nested columns
- We `unnest_longer()` those nested columns to different rows:

```
# A tibble: 3,248 x 6
     row    nl  west nonwest iter       Ba
   <int> <dbl> <dbl>   <dbl> <list>     <list>
 1     1  0.69  0.19    0.12 <int [50]> <dbl [91]>
 2     2  0.75  0.16    0.09 <int [50]> <dbl [91]>
 3     3  0.74  0.16    0.1  <int [50]> <dbl [91]>
 4     4  0.68  0.14    0.18 <int [50]> <dbl [91]>
 5     5  0.71  0.13    0.16 <int [50]> <dbl [91]>
 6     6  0.78  0.14    0.08 <int [50]> <dbl [91]>
 7     7  0.77  0.13    0.1  <int [50]> <dbl [91]>
 8     8  0.8   0.1     0.1  <int [50]> <dbl [91]>
 9     9  0.85  0.09    0.06 <int [50]> <dbl [91]>
10    10  0.68  0.14    0.18 <int [50]> <dbl [91]>
# ... with 3,238 more rows
```

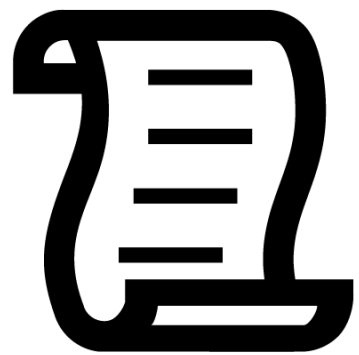
```
# A tibble: 14,778,400 x 6
     row    nl  west nonwest  iter    Ba
   <int> <dbl> <dbl>   <dbl> <int> <dbl>
 1     1  0.69  0.19    0.12     1  0.05
 2     1  0.69  0.19    0.12     2  0.05
 3     1  0.69  0.19    0.12     3  0.05
 4     1  0.69  0.19    0.12     4  0.05
 5     1  0.69  0.19    0.12     5  0.05
 6     1  0.69  0.19    0.12     6  0.05
 7     1  0.69  0.19    0.12     7  0.05
 8     1  0.69  0.19    0.12     8  0.05
 9     1  0.69  0.19    0.12     9  0.05
10     1  0.69  0.19    0.12    10  0.05
# ... with 14,778,390 more rows
```

More of this in the hands-on later

Supercomputing structure

1. Create a **parameter grid** (data frame)
2. Create an **analysis function** that takes in a row of the parameter grid and outputs a result
3. Create a **self-contained job script** to load a chunk of the grid and run the analysis function on several rows in parallel
4. Create a **shell script** to run the R job with SLURM parameters

script



The analysis function

What it does

Input a row from the grid

Output our quantity of interest
(proportion of happy nonwestern migrants)

Should be **robust**, i.e., deal with problematic inputs gracefully

- You should spend time testing this, you will literally run this code a million times

The analysis function

```
analysis_function ← function(row_idx) {
  # Get the parameters belonging to this row
  settings ← as.list(grid_tbl[row_idx,])

  # compute the proportion of happy nonwestern migrants
  # use trycatch to avoid crashing. This is important otherwise
  # you will have a lot of problems with underused compute!
  out ← tryCatch(
    # this is the expression to evaluate
    expr = {
      prop_vec ← c(settings$nl, settings$west, settings$nonwest)
      res ← abm_cpp(prop = prop_vec, Ba = settings$Ba)
      return(res$h_prop[3])
    },
    # if there is an error, return NA as output!
    error = function(e) return(NA)
  )

  return(out)
}
```

The R job script

What it does

Input a job number

Output a file with results from the ABMs and a log file

- Self-contained, runnable from the command-line
- Nice logging capabilities to show where things are going wrong if they do
- Within-node parallellization(!)

Self-contained R scripts

- You can run R in non-interactive mode
  (if it's in your environment variables)

```
# getting the arguments from the commandline
args ← commandArgs(trailingOnly = TRUE)
num  ← as.numeric(args[1])

# return random numbers
rnorm(num)
```

```
> Rscript my_script.R 10

 [1]  1.01272137 -2.10078427  0.58351622 -0.62444158 -1.22377068 -0.07592772
 [7] -0.12156296  2.17437392 -0.18906297  1.78086798
```

Logging

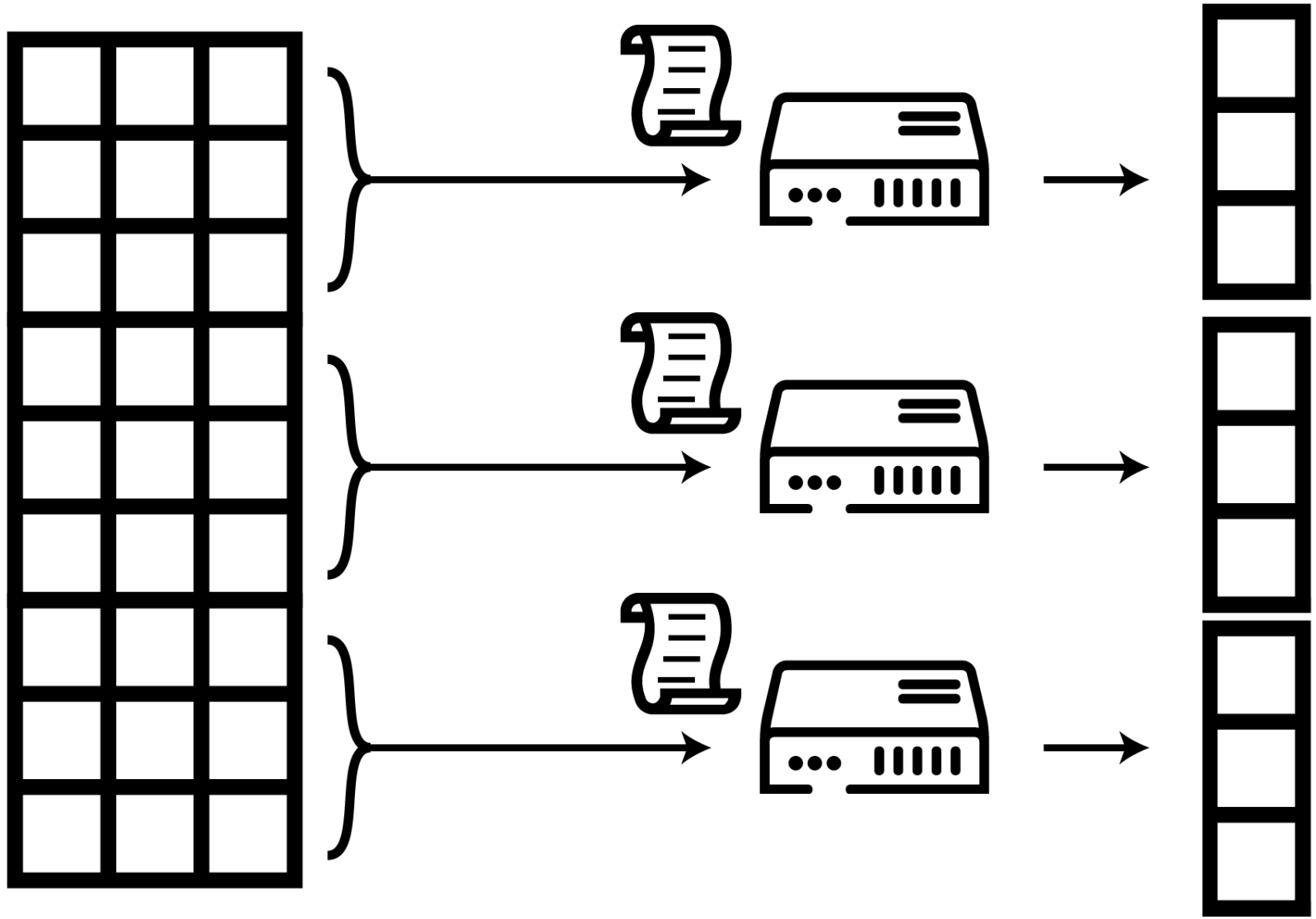
- There are several options available, e.g., the package **`logging`**
- For our case, we only need simple print (`cat()`) statements
- SLURM will store the R console output to a file

- Include statements in the script about which step is running
- Include timestamps / elapsed time

Within-node parallellization

- In SLURM, you get (and pay for!) an entire node at a time
- Therefore, your R scripts need within-node parallellization
- Compute results for multiple grid rows at a time

- "**Chunking**" your grid



Within-node parallellization

- Get chunk
- Start cluster / child processes
- Compute chunk results on cluster
- Save output to file (`results_001.rds`)

Within-node parallellization

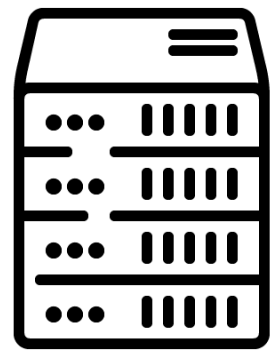
- How big should the chunk be?
- Depends on
  - speed (ABM runs/second/core)
  - number of cores on the node (16)
  - how long you want each job to take
- How long? Make it manageable, e.g., 30 minutes
  - If something goes wrong (and something will go wrong!) you can rerun in reasonable amount of time
  - Balance manageability and overhead: data loading, Rcpp code compiling, results storing

More of this in the hands-on later

Supercomputing structure

1. Create a **parameter grid** (data frame)
2. Create an **analysis function** that takes in a row of the parameter grid and outputs a result
3. Create a **self-contained job script** to load a chunk of the grid and run the analysis function on several rows in parallel
4. Create a **shell script** to run the R job with SLURM parameters

submit



Shell script

- We will use array jobs (as per SLURM terminology)

```
sbatch –a 1-77 my_script.sh
```

- Will queue 77 jobs
- Each job has a different environment variable **SLURM_ARRAY_TASK_ID**
- Pass this environment variable to Rscript

Shell script

```
#!/bin/bash

# usage: sbatch -a 1-100 05_array_job.sh

# Array job shell script which runs R code
# last edited 2022-04-04 by @vankesteren
# ODISSEI Social Data Science team

# Set job requirements
#SBATCH -n 1
#SBATCH -t 00:45:00
#SBATCH -o ./logs/output.%a.out # STDOUT
#SBATCH --mail-type=BEGIN,END
#SBATCH --mail-user=e.vankesteren1@uu.nl


# Loading modules
module load 2021
module load R/4.1.0-foss-2021a

# Run the script
Rscript "04_array_job.R" $SLURM_ARRAY_TASK_ID
```

```
lcur0520@login3:~/ossc_workshop$ sbatch -a 1-77 05_array_job.sh
Submitted batch job 9185382
lcur0520@login3:~/ossc_workshop$ squeue -u lcur0520
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
    9185382_[1-77]    normal 05_array lcur0520 PD       0:00      1 (Resources)
```

```
lcur0520@login3:~/ossc_workshop$ squeue -u lcur0520
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
    9185382_[4-77]    normal 05_array lcur0520 PD       0:00      1 (Resources)
         9185382_1    normal 05_array lcur0520  R       0:19      1 r27n17
         9185382_2    normal 05_array lcur0520  R       0:19      1 r25n17
         9185382_3    normal 05_array lcur0520  R       0:19      1 r25n27
```

```
lcur0520@login3:~/ossc_workshop$ scancel 9185382
lcur0520@login3:~/ossc_workshop$ squeue -u lcur0520
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
         9185382_6    normal 05_array lcur0520 CG       0:07      1 r27n6
         9185382_7    normal 05_array lcur0520 CG       0:07      1 r27n7
         9185382_5    normal 05_array lcur0520 CG       0:23      1 r26n7
         9185382_4    normal 05_array lcur0520 CG       0:26      1 r27n31
         9185382_1    normal 05_array lcur0520 CG       0:58      1 r27n17
         9185382_2    normal 05_array lcur0520 CG       0:58      1 r25n17
         9185382_3    normal 05_array lcur0520 CG       0:58      1 r25n27
```

Supercomputing structure

1. Create a **parameter grid** (data frame)
2. Create an **analysis function** that takes in a row of the parameter grid and outputs a result
3. Create a **self-contained job script** to load a chunk of the grid and run the analysis function on several rows in parallel
4. Create a **shell script** to run the R job with SLURM parameters

Hands-on session 2