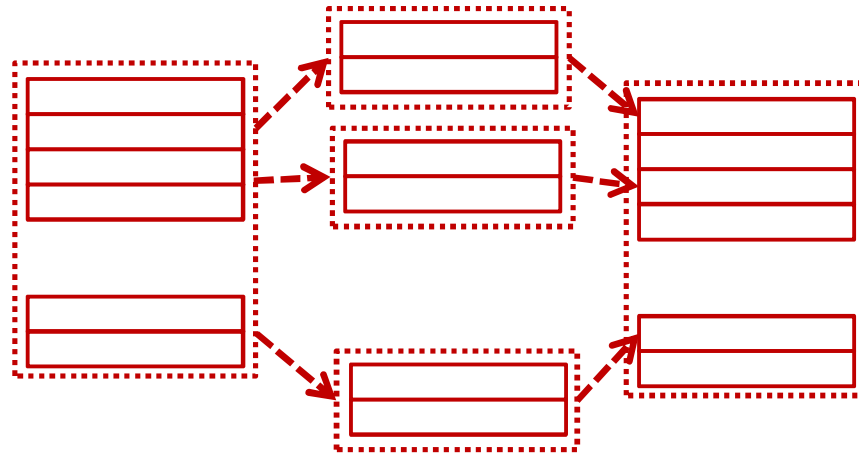


High-performance Clinical Scenario Evaluation and Simulation

“Embarrassingly simple” parallel computing using R

Jouni Kerman, Novartis Statistical Methodology Group
BaselR meeting, 25 January 2012

Agenda



About embarrassingly parallel clinical trial simulations

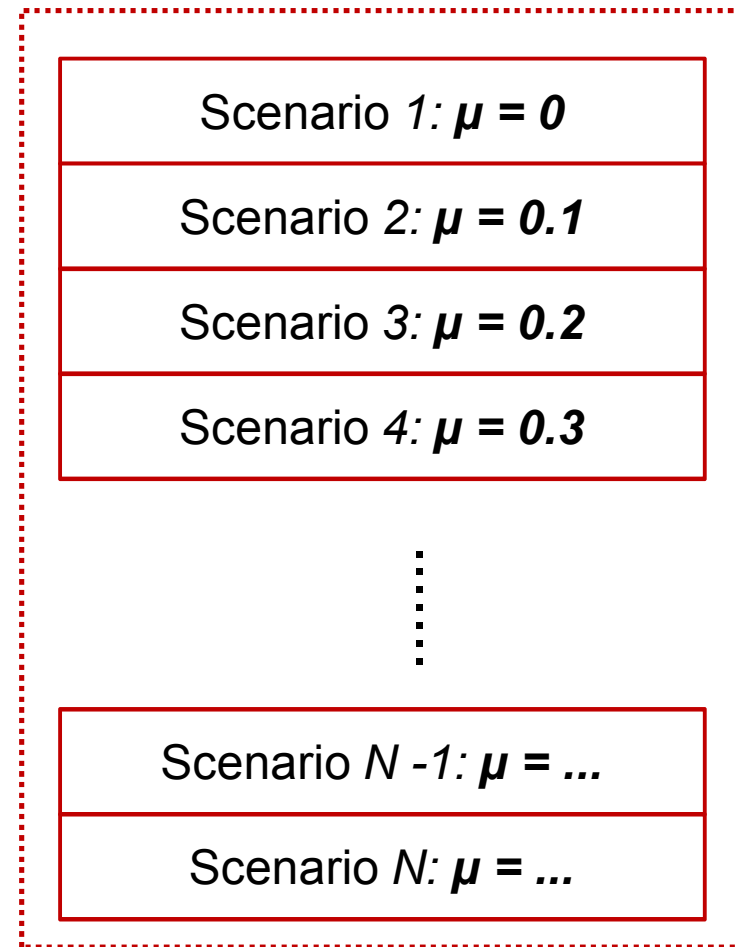
Using Rjob

Example

Summary

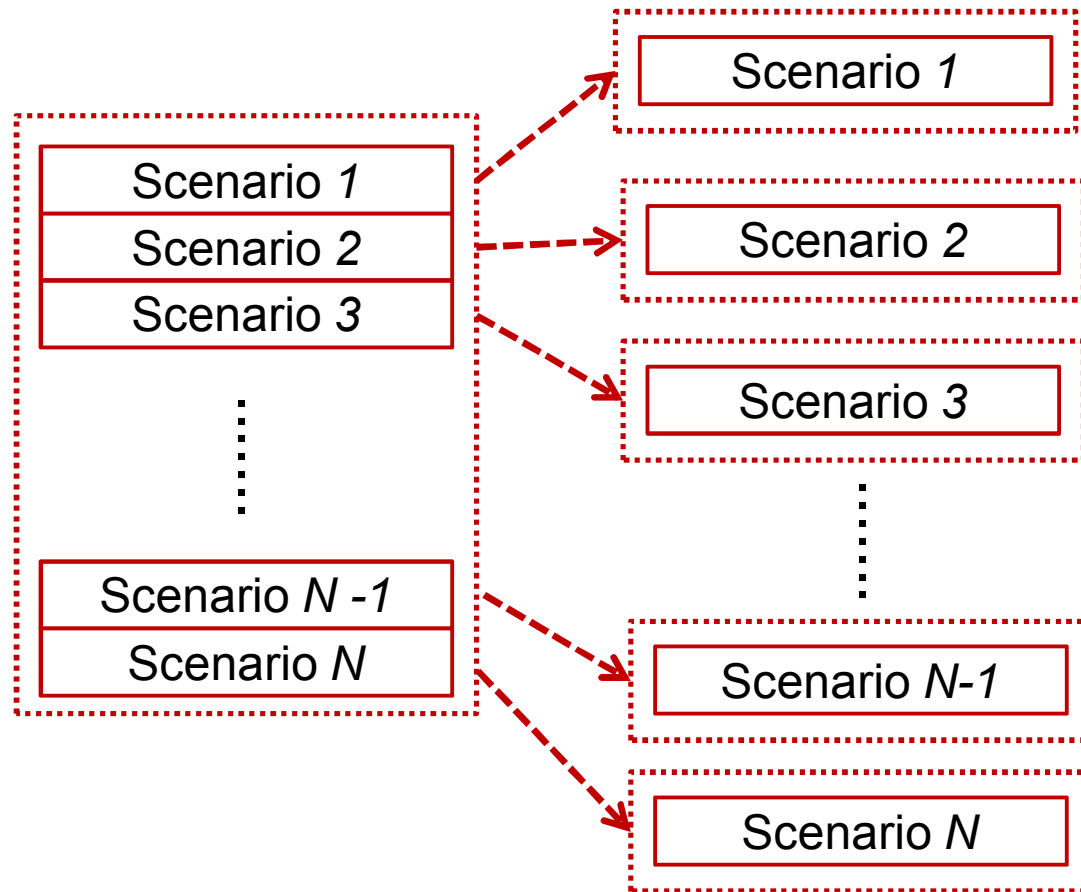
Clinical scenario evaluation and simulation

- Typical case: Computing 'operating characteristics'
 - Each scenario returns an estimate of the statistical **power** given the scenario parameter values

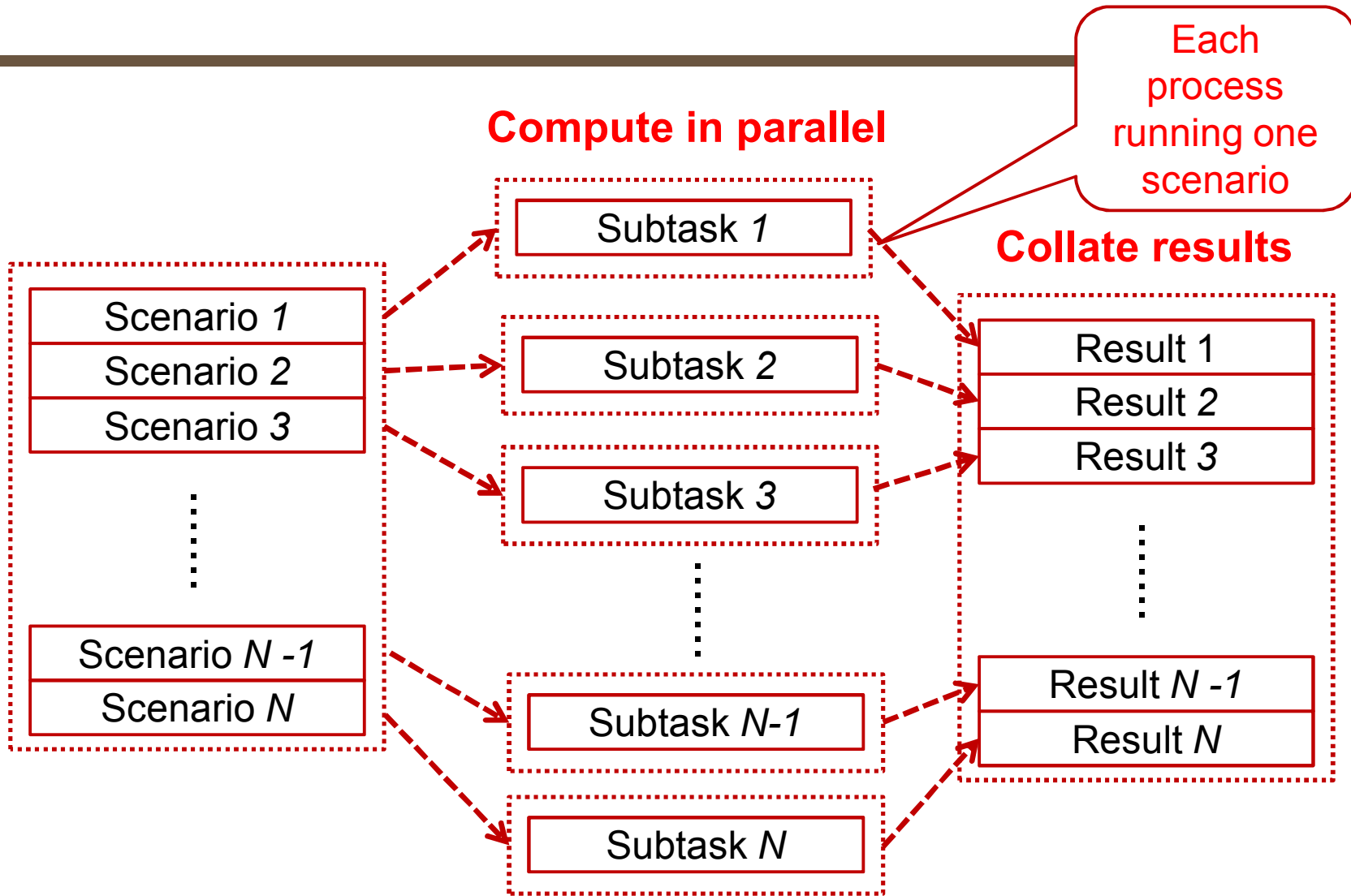


'Embarrassingly parallel' computational problems

- Mutually independent tasks
 - Each task ("trial scenario") can be executed in parallel



Divide into subtasks, compute & collate



Computing ‘embarrassingly parallel’ clinical scenarios

- One function call per subtask
 - Call a function on each parameter combination
 - Collate results
 - How to implement this in parallel?

Subtask	Parameters	Function call
#1	$(n_1, n^{interim}, \mu_1, \sigma_1, \lambda_1, L_1)$	CompScenario(.)
#2	$(n_2, n^{interim}, \mu_1, \sigma_1, \lambda_1, L_1)$	CompScenario(.)
...		...
#3	$(n_a, n_b^{interim}, \mu_c, \sigma_d, \lambda_e, L_f)$	CompScenario(.)
...		...
#N-1	$(n_A, n_B^{interim}, \mu_C, \sigma_D, \lambda_E, L_{F-1})$	CompScenario(.)
#N	$(n_A, n_B^{interim}, \mu_C, \sigma_D, \lambda_E, L_F)$	CompScenario(.)

Parallel execution

Package 'Rjob': Simplicity ...

- Simplifies writing programs that can be parallelized
 - Only **one single function** to execute the scenarios
 - Provide arguments ('scenario parameters'), whether lists, objects or atomic objects, in a **data-frame like** structure
 - **Results**, whether lists, objects or atomic objects, can be collated automatically into a data-frame like structure

Package 'Rjob': ... and transparency

- Write once, run anywhere
 - Run the **same program**
 - on Sun Grid Engine (using distributed computing)
 - on a multicore computer
 - on a single-core computer
- without any modifications to the code

R program structure

```
## R PROGRAM ##
```

```
library(Rjob)
```

```
Scenarios <- ...
```

```
Compute <- function ...
```

```
X <- do.tasks(Compute,  
  ARGS      = Scenarios,  
  JOB.ID    = "ABC123X2201",  
  SEED      = 20120124)
```

```
results <- Process(X)
```

```
write.csv(results,  
  file="ABC123X2201.csv")
```

A data frame providing arguments to function 'Compute'

Function to compute a result

Call do.tasks, receive results

Process results and output

Many ways to run the program

```
qrjob prog.R
```

```
qrjob --cores 5 --batch.size 10 prog.R
```

Batch mode:
Distributed
computing on
the SGE

```
R --vanilla -f prog.R
```

Batch mode:
Execution on a
workstation or
Grid node

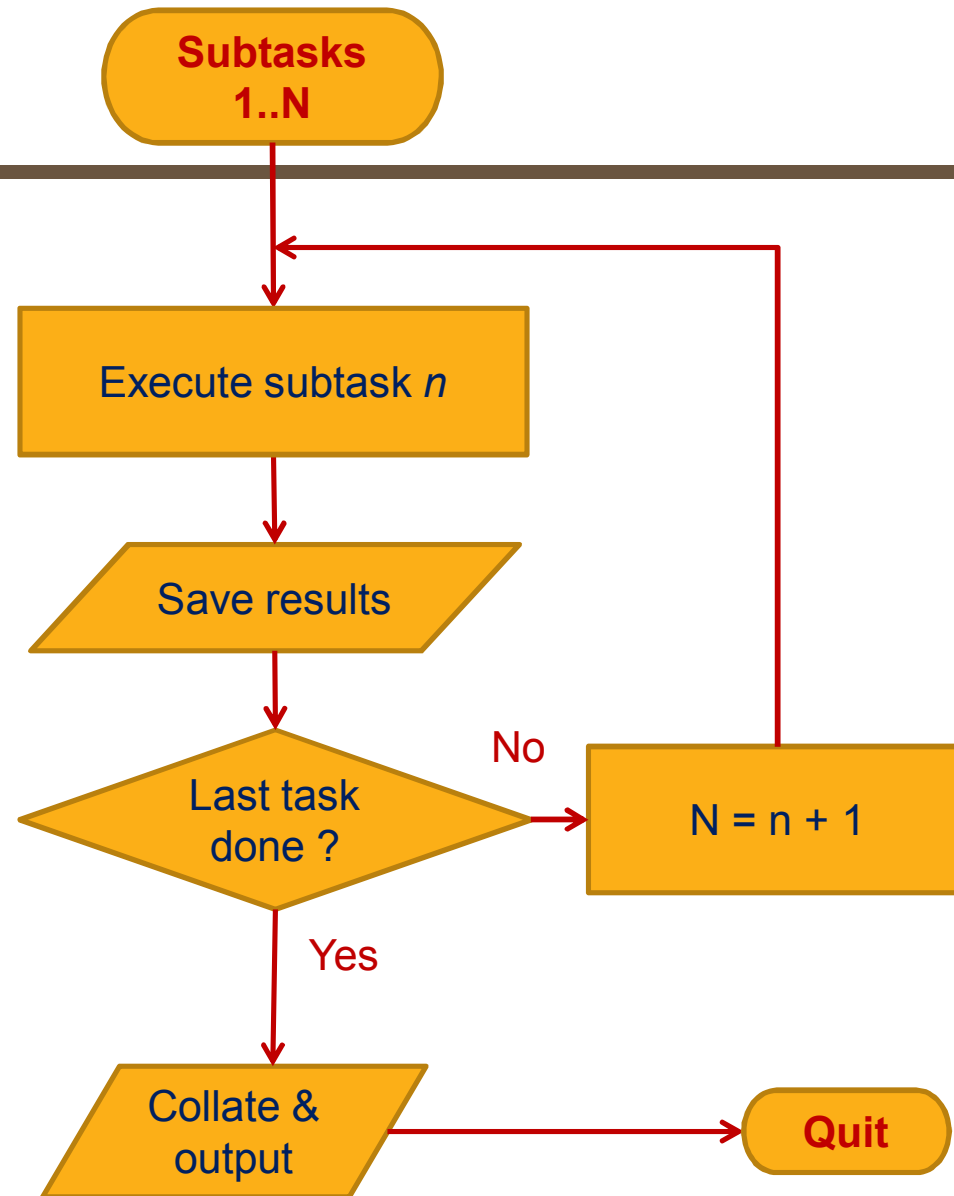
```
> source("prog.R")
```

```
> print(X)
```

*Interactive
mode:*
On the
R console

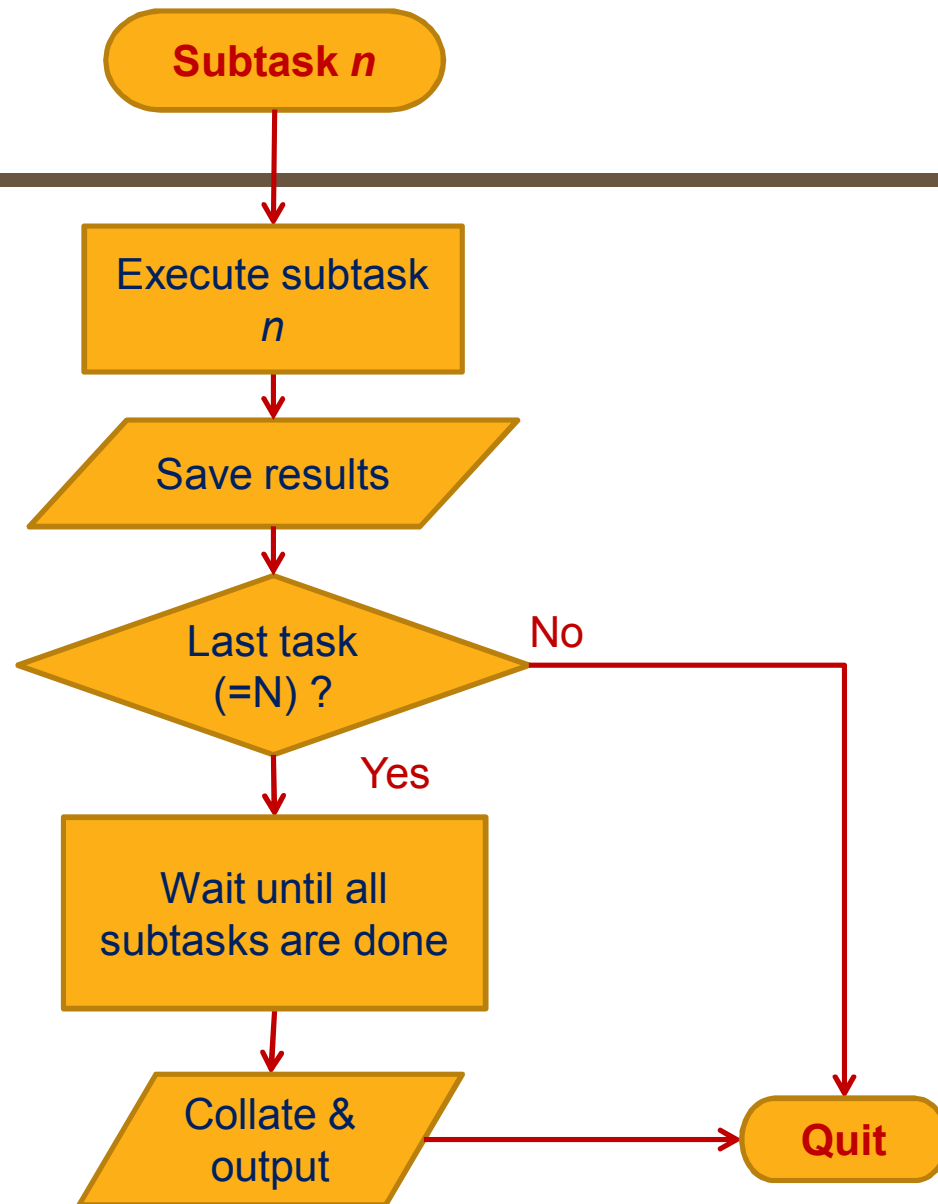
Rjob & non-parallel computing

- Running sequentially
 - Each SGE task is assigned an ID number => our subtask number
 - Each subtask follows the same **process logic**:



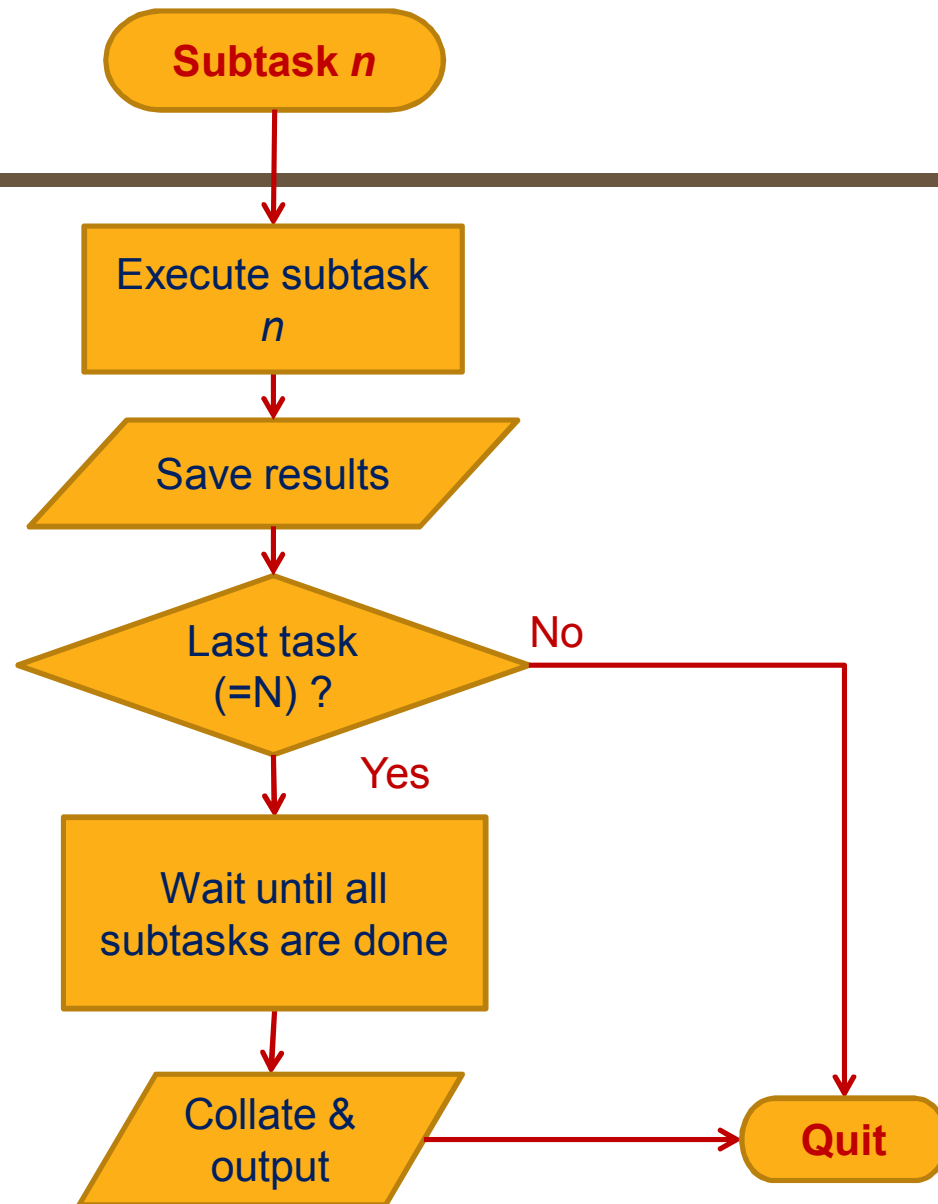
Rjob & non-parallel computing

- Using the package 'multicore'
 - Each subtask can be run in parallel (in any order)
 - Leave everything to `multicore::mclapply` to execute things in parallel



Rjob & distributed computing

- Execute N copies of the same program on the Sun Grid Engine
 - Each SGE task is assigned an ID number => our subtask number
 - Each process “knows” which **one subtask** to do



Example

Examining power of a statistical procedure

Compute Pr('Significance' | scenario):

'Significance' := p -value of 1-sided t-test $< \alpha$

Data = $y \sim$ Normal with given mean δ & sd σ

Design parameters: $(n, \delta, \sigma, \alpha)$

Example

Parameters for 216 scenarios

Scenarios

Scenarios:

<u>scenario</u>	<u>delta</u>	<u>sigma</u>	<u>alpha</u>	<u>n</u>
1	0.0	0.1	0.025	6
2	0.1	0.1	0.025	6
3	0.5	0.1	0.025	6
4	1.0	0.1	0.025	6
5	0.0	0.5	0.025	6
6	0.1	0.5	0.025	6
7	0.5	0.5	0.025	6
8	1.0	0.5	0.025	6
...				
210	0.1	0.5	0.100	1000
211	0.5	0.5	0.100	1000
212	1.0	0.5	0.100	1000
213	0.0	1.0	0.100	1000
214	0.1	1.0	0.100	1000
215	0.5	1.0	0.100	1000
216	1.0	1.0	0.100	1000

(1) Set-up

```
library(Rjob)

Power <- function (n, delta, sigma, alpha, n.trials) {
  .significant.trial <- function (i) {
    y <- rnorm(n=n, mean=delta, sd=sigma)
    p.value <- t.test(x=y, alt="greater")$p.value
    return(p.value < alpha)
  }
  success <- sapply(1:n.trials, .significant.trial)
  power <- mean(success)
  df <- data.frame(power=mean(success))
  return(df)
}

Scenarios <- expand.grid(delta = c(0.0, 0.1, 0.5, 1.0),
                        sigma = c(0.1, 0.5, 1.0),
                        alpha = c(0.025, 0.05, 0.10),
                        n      = c(6, 12, 24, 36, 48, 1000))
```

Function to
evaluate one
scenario

Parameter
combinations

(2) Compute tasks and (3) Output results

```
D <- do.tasks(FUN      = Power,  
              ARGS    = Scenarios,  
              n.trials = 1000,  
              JOB.ID   = "ABC123X2011",  
              FORCE    = ! interactive(),  
              SEED     = 20111000)
```

2. Call
do.tasks

```
write.csv(D, file="oc_ABC123X2201.csv")
```

3. Output

Example: full code

```
library(Rjob)

Power <- function (n, delta, sigma, alpha, n.trials) {
  .significant.trial <- function (i) {
    y <- rnorm(n=n, mean=delta, sd=sigma)
    p.value <- t.test(x=y, alt="greater")$p.value
    return(p.value < alpha)
  }
  success <- sapply(1:n.trials, .significant.trial)
  power <- mean(success)
  df <- data.frame(power=mean(success))
  return(df)
}

Scenarios <- expand.grid(delta = c(0.0, 0.1, 0.5, 1.0),
                        sigma = c(0.1, 0.5, 1.0),
                        alpha = c(0.025, 0.05, 0.10),
                        n      = c(6, 12, 24, 36, 48, 1000))

D <- do.tasks(FUN      = Power,
              ARGS     = Scenarios,
              n.trials = 1000,
              JOB.ID   = "ABC123X2011",
              FORCE     = ! interactive(),
              SEED     = 20111000)

write.csv(D, file="oc_ABC123X2201.csv")
```

Set up

Call do.tasks

Output

Rjob on Sun Grid Engine

```
[@node01] qrjob ABC123X2011.R
```

```
...
```

```
Forcing re-execution of all tasks
```

```
Sending off 216 tasks to the grid ...
```

```
Your job-array 100601.1-216:1 ("Rjob.ABC123X2011") has been submitted
```

```
[@node01]
```

A unix shell script provided with Rjob

Rjob (in batch mode) on a workstation

```
[@node01] R --vanilla --slave -f ABC123X2011.R
```

```
...
```

```
[@node01]
```

Rjob on the R console

“Just re-run the program”

```
[@node01] R
```

```
R version 2.13.2 (2011-09-30)
```

```
Copyright (C) 2011 The R Foundation for Statistical Computing
```

```
...
```

```
R 2.13.2 > source("ABC123X2201.R")
```

```
R 2.13.2 > print(D)
```

```
...
```

If the subtasks have already been done once, running the program will simply collect the results from disk!

Automatic collating of results into a data frame

	delta	sigma	alpha	n	n.trials	<u>power</u>
1	0.0	0.1	0.025	6	1000	0.021
2	0.1	0.1	0.025	6	1000	0.507
3	0.5	0.1	0.025	6	1000	1.000
4	1.0	0.1	0.025	6	1000	1.000
5	0.0	0.5	0.025	6	1000	0.024
6	0.1	0.5	0.025	6	1000	0.062
7	0.5	0.5	0.025	6	1000	0.494
8	1.0	0.5	0.025	6	1000	0.967
9	0.0	1.0	0.025	6	1000	0.030
10	0.1	1.0	0.025	6	1000	0.038
11	0.5	1.0	0.025	6	1000	0.182
12	1.0	1.0	0.025	6	1000	0.512
13	0.0	0.1	0.050	6	1000	0.057
...						
205	0.0	0.1	0.100	1000	1000	0.084
206	0.1	0.1	0.100	1000	1000	1.000
207	0.5	0.1	0.100	1000	1000	1.000
208	1.0	0.1	0.100	1000	1000	1.000
209	0.0	0.5	0.100	1000	1000	0.093
210	0.1	0.5	0.100	1000	1000	1.000
211	0.5	0.5	0.100	1000	1000	1.000
212	1.0	0.5	0.100	1000	1000	1.000
213	0.0	1.0	0.100	1000	1000	0.093
214	0.1	1.0	0.100	1000	1000	0.975
215	0.5	1.0	0.100	1000	1000	1.000
216	1.0	1.0	0.100	1000	1000	1.000

Each call to function **Power()** returns one data frame with one column 'power' with one single row.

Results have been combined with the corresponding scenario parameters

Automatic collating of results into an 'xframe'

	delta	sigma	alpha	n	n.trials	power	<u>obj</u>
1	0.0	0.1	0.025	6	1000	0.021	lm.1*
2	0.1	0.1	0.025	6	1000	0.507	lm.2*
3	0.5	0.1	0.025	6	1000	1.000	lm.3*
4	1.0	0.1	0.025	6	1000	1.000	lm.4*
5	0.0	0.5	0.025	6	1000	0.024	lm.5*
6	0.1	0.5	0.025	6	1000	0.062	lm.6*
7	0.5	0.5	0.025	6	1000	0.494	lm.7*
8	1.0	0.5	0.025	6	1000	0.967	lm.8*
9	0.0	1.0	0.025	6	1000	0.030	lm.9*
10	0.1	1.0	0.025	6	1000	0.038	lm.10*
11	0.5	1.0	0.025	6	1000	0.182	lm.11*
12	1.0	1.0	0.025	6	1000	0.512	lm.12*
13	0.0	0.1	0.050	6	1000	0.057	lm.13*
...							
205	0.0	0.1	0.100	1000	1000	0.084	lm.205*
206	0.1	0.1	0.100	1000	1000	1.000	lm.206*
207	0.5	0.1	0.100	1000	1000	1.000	lm.207*
208	1.0	0.1	0.100	1000	1000	1.000	lm.208*
209	0.0	0.5	0.100	1000	1000	0.093	lm.209*
210	0.1	0.5	0.100	1000	1000	1.000	lm.210*
211	0.5	0.5	0.100	1000	1000	1.000	lm.211*
212	1.0	0.5	0.100	1000	1000	1.000	lm.212*
213	0.0	1.0	0.100	1000	1000	0.093	lm.213*
214	0.1	1.0	0.100	1000	1000	0.975	lm.214*
215	0.5	1.0	0.100	1000	1000	1.000	lm.215*
216	1.0	1.0	0.100	1000	1000	1.000	lm.216*

Each call to function **Power()** can return arbitrary objects in a data-frame like xframe object. Individual xframes are automatically collated into an xframe.

Why use Rjob?

- Simple to use
 - Only one function needed, **do.tasks**
- Transparent
 - Write once, run “anywhere”
- Convenient “user interface”
 - Unified way to define scenarios and deliver outputs, in data-frame like ‘xframe’ objects
- Readable & easy to understand
 - Simple program structure