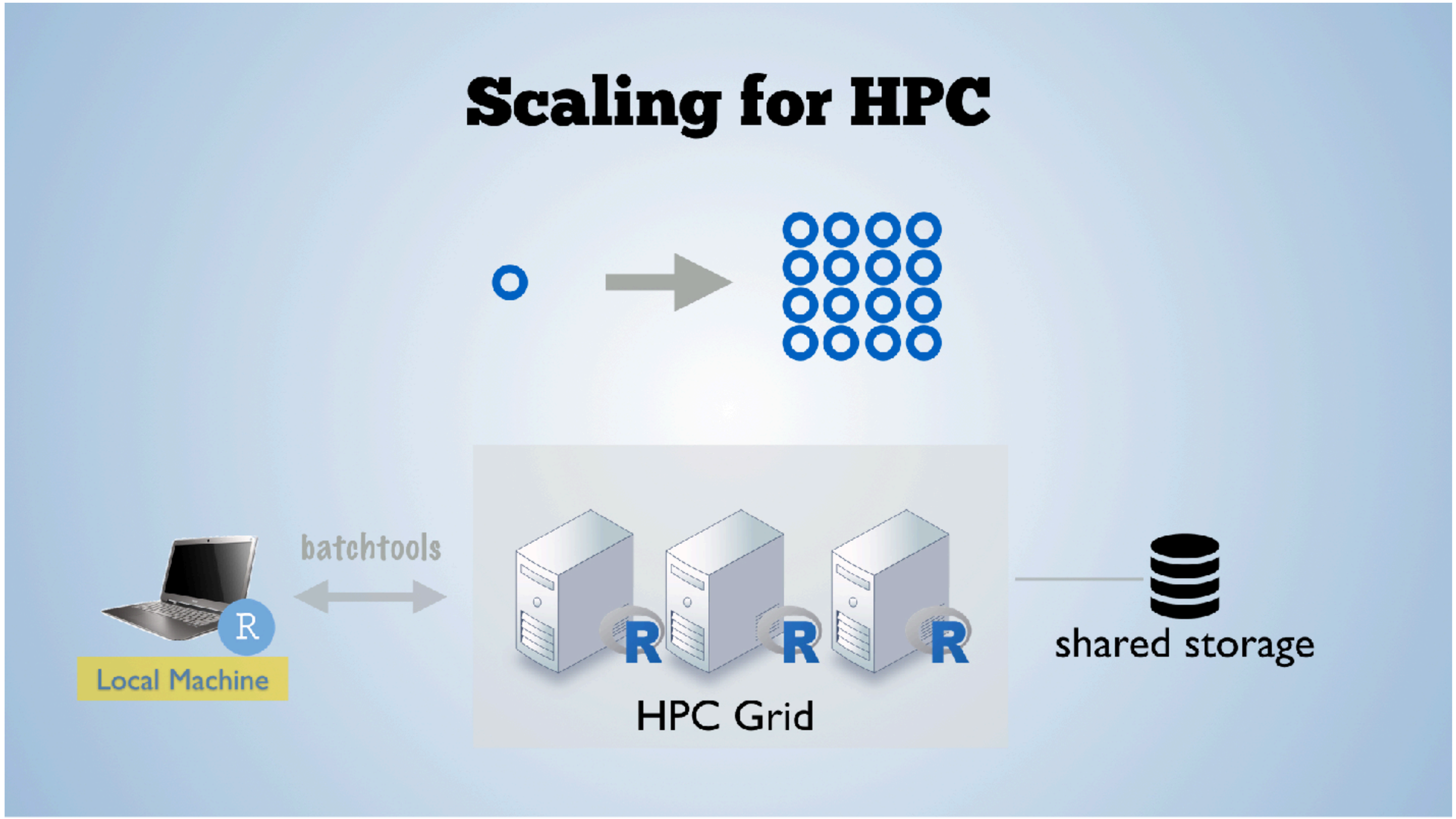
Code Snippets:
Batch Jobs in R

Soo-Heang EO
SIDA
(The Sokcho Institute for Data Analytics)

GOAL



https://rviews.rstudio.com/2016/12/21/r-for-enterprise-how-to-scale-your-analytics-using-r/

GOAL

Computer Experiments to solve large-scale machine learning problems

```
##     job.id problem algorithm ratio kernel epsilon ntree   mce
##      <int>  <char>    <char> <num> <char>   <num> <num> <num>
## 1:       1    iris       svm  0.67 linear    0.01    NA  0.04
## 2:       2    iris       svm  0.67 linear    0.01    NA  0.00
## 3:       3    iris       svm  0.67 linear    0.01    NA  0.06
## 4:       4    iris       svm  0.67 linear    0.01    NA  0.04
## 5:       5    iris       svm  0.67 linear    0.01    NA  0.02
## 6:       6    iris       svm  0.67 linear    0.10    NA  0.04
```

batchtools

Control a batch procedure in **R** + **Experiment** statistical designs with parameters of algorithms and problems

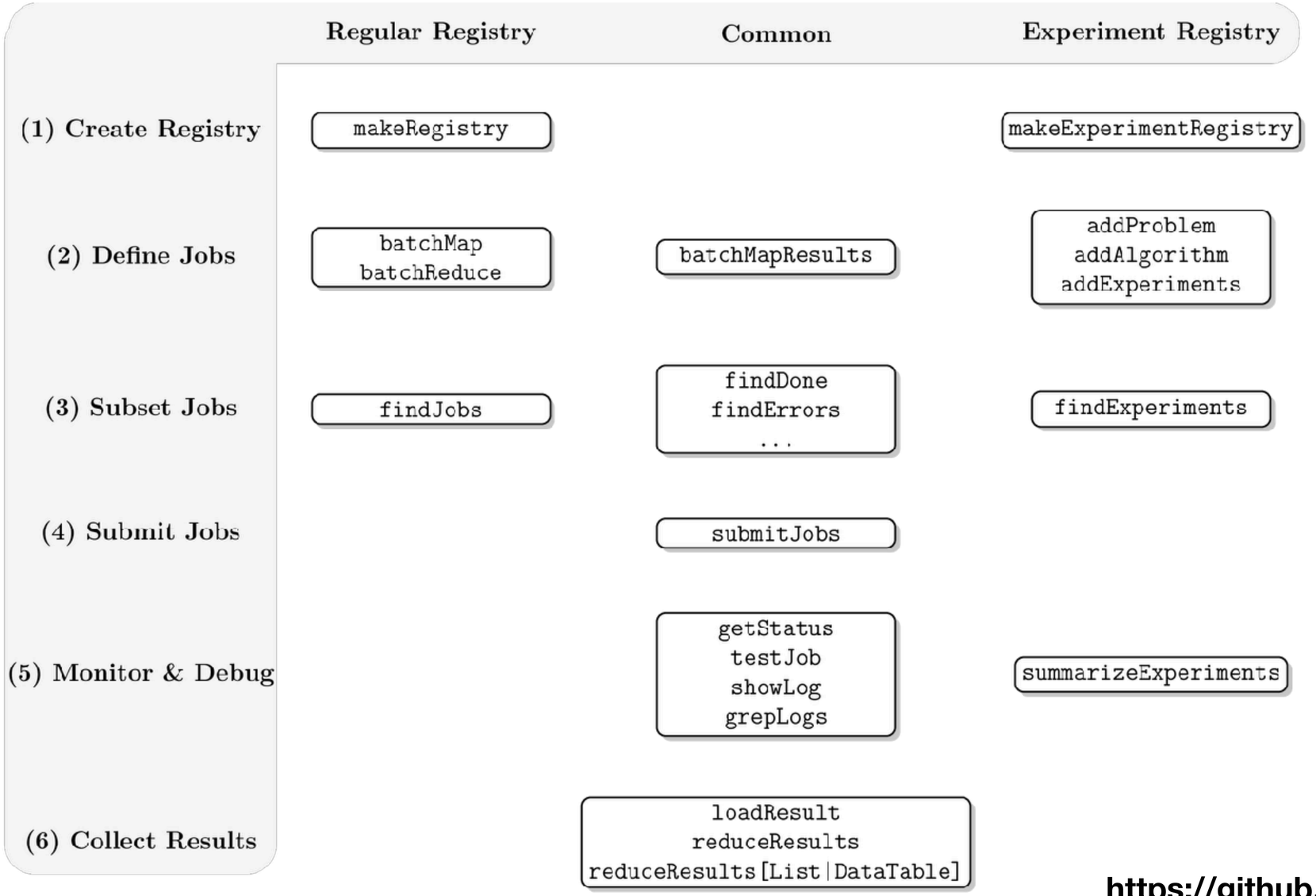
As a successor of the packages BatchJobs and BatchExperiments, batchtools provides a parallel implementation of Map for high performance computing systems managed by schedulers like Slurm, Sun Grid Engine, OpenLava, TORQUE/OpenPBS, Load Sharing Facility (LSF) or Docker Swarm (see the setup section in the vignette).

Main features:

- **Convenience**: All relevant batch system operations (submitting, listing, killing) are either handled internally or abstracted via simple R functions
- **Portability**: With a well-defined interface, the source is independent from the underlying batch system - prototype locally, deploy on any high performance cluster
- **Reproducibility**: Every computational part has an associated seed stored in a data base which ensures reproducibility even when the underlying batch system changes
- **Abstraction**: The code layers for algorithms, experiment definitions and execution are cleanly separated and allow to write readable and maintainable code to manage large scale computer experiments

https://github.com/mllg/batchtools

Workflow of *batchtools*



https://github.com/mllg/batchtools

batchtools: (1) Create Registry

- First, we create a registry, the central meta-data object which records technical details and the setup of the experiments.
- We use an `ExperimentRegistry` where the job definition is split into creating problems and algorithms.
- Again, we use a temporary registry and make it the default registry.

```
library(batchtools)
reg = makeExperimentRegistry(file.dir = NA, seed = 1)
```

```
> str(reg)
Classes 'ExperimentRegistry', 'Registry' <environment: 0xbc23d08>
```

https://mllg.github.io/batchtools/articles/batchtools.html#example-2-machine-learning

batchtools: (2) Define Jobs - Problem

`addProblem()` files the problem to the file system and the problem gets recorded in the registry.

```
subsample = function(data, job, ratio, ...) {
  n = nrow(data)
  train = sample(n, floor(n * ratio))
  test = setdiff(seq_len(n), train)
  list(test = test, train = train)
}
data("iris", package = "datasets")

addProblem(name = "iris", data = iris, fun = subsample, seed = 42)
```

https://mllg.github.io/batchtools/articles/batchtools.html#example-2-machine-learning

batchtools: (2) Define Jobs - Algorithm

The algorithms for the jobs are added to the registry in a similar manner. When using `addAlgorithm()`, an identifier as well as the algorithm to apply to are required arguments.

```
svm.wrapper = function(data, job, instance, ...) {
  mod = e1071::svm(Species ~ ., data = data[instance$train, ], ...)
  pred = predict(mod, newdata = data[instance$test, ], type = "class")
  table(data$Species[instance$test], pred)
}

addAlgorithm(name = "svm", fun = svm.wrapper)
```

https://mllg.github.io/batchtools/articles/batchtools.html#example-2-machine-learning

batchtools: (2) Define Jobs - Experiments

`addExperiments()` is used to parametrize the jobs and thereby define computational jobs.

```
# problem design: try two values for the ratio parameter
pdes = list(iris = data.table(ratio = c(0.67, 0.9)))

# algorithm design: try combinations of kernel and epsilon
exhaustively,
# try different number of trees for the forest
ades = list(
  svm = CJ(kernel = c("linear", "polynomial", "radial"),
           epsilon = c(0.01, 0.1)),
  forest = data.table(ntree = c(100, 500, 1000))
)

addExperiments(pdes, ades, repls = 5)
```

https://mllg.github.io/batchtools/articles/batchtools.html#example-2-machine-learning

batchtools: (3) Submit Jobs

To submit the jobs, we call `submitJobs()` and wait for all jobs to terminate using `waitForJobs()`.

```
submitJobs()
## Submitting 90 jobs in 90 chunks using cluster functions 'Interactive' ...
waitForJobs()
## [1] TRUE
```

https://mllg.github.io/batchtools/articles/batchtools.html#example-2-machine-learning

batchtools: (4) Monitor & Debug

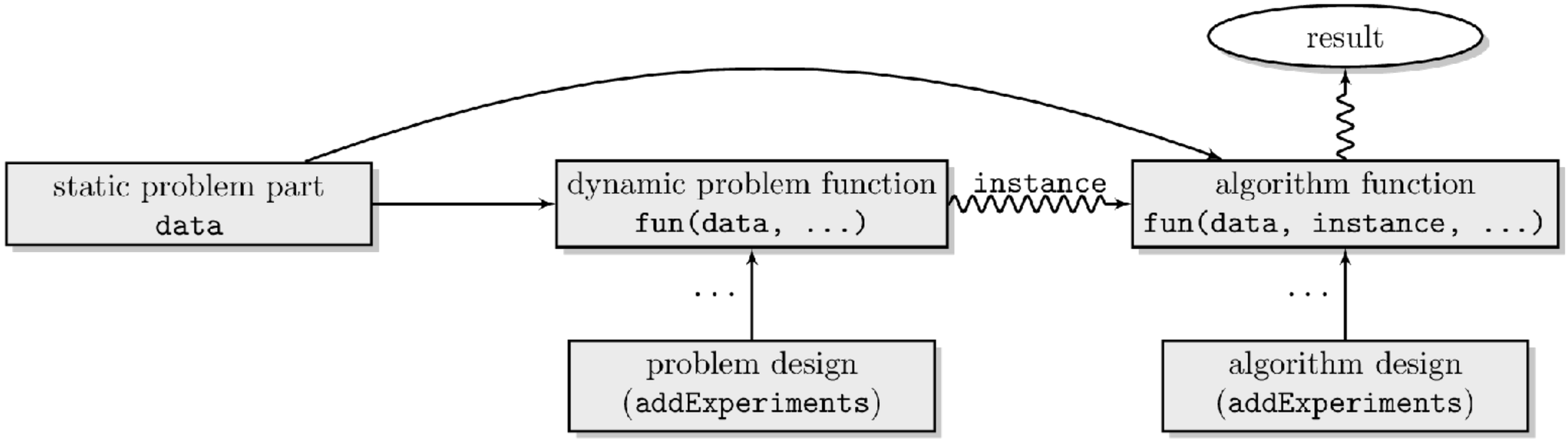
After jobs are finished, the results can be collected with `reduceResultsDataTable()` where we directly extract the mean misclassification error:

```
reduce = function(res) list(mce = (sum(res) - sum(diag(res))) / sum(res))
results = unwrap(reduceResultsDataTable(fun = reduce))
head(results)
```

```
##    job.id   mce
##     <int> <num>
## 1:      1  0.04
## 2:      2  0.00
## 3:      3  0.06
## 4:      4  0.04
## 5:      5  0.02
## 6:      6  0.04
```

https://mllg.github.io/batchtools/articles/batchtools.html#example-2-machine-learning

batchtools: summary



https://mllg.github.io/batchtools/articles/batchtools.html#example-1-approximation-of-pi

Some Materials

- https://github.com/HenrikBengtsson/future
- https://github.com/mschubert/clustermq
- http://www.maths.lancs.ac.uk/~rowlings/HPC/RJobs/
- https://confluence.csiro.au/display/SC/Run+parallel+R+jobs+using+the+package+rslurm
- https://www.r-bloggers.com/batch-processing-vs-interactive-sessions/
- https://github.com/kirillseva/ruigi
- https://github.com/spotify/luigi
- https://cran.r-project.org/web/packages/future.batchtools/vignettes/future.batchtools.html