Laboratório
Nacional de
Computação
Científica

R em ambiente HPC

Kary Ocaña – karyann@lncc.br

Autores: Kary Ocaña, Micaela Coelho, Guilherme Gall, Raquel Costa
Laboratório Nacional de Computação Científica – LNCC
Programa de Verão do LNCC 2021

15 de Janeiro de 2021

Sumário

- ➔ Introdução ao R
  - O que é o R? Por que usar R? E por que utiliza-lo em ambientes HPC?
- ➔ Carregar o módulo R no supercomputador (Santos Dumont, SD)
  - Instalar um pacote R no SD. Submetendo um script para o SD
- ➔ Família apply
- ➔ Pacotes de paralelismo no R
  - Análogos à famíla apply (parallel, snow)
  - foreach e seus backends paralelos (foreach, doParallel, doSnow)
- ➔ *Benchmark*
- ➔ Melhorando o código R, boas práticas
- ➔ Estudo de caso: Paralelismo de tarefas
  - "*Machine Learning*" e Model-R
- ➔ Considerações finais

Avisos

- Horário do curso:
  - Início: 14:30
  - Final: 18:30
- Clonar repositório do curso:
  - git clone https://github.com/Micaella/R_HPC

Supercomputador Santos Dumont

O supercomputador Santos Dumont

Supercomputador Santos Dumont, LNCC, Petrópolis.

- MoBull - solução para *datacenter* baseada em *containers*
- Plug & Boot
- 2 containers com 28 racks 42U
- Arquitetura de cluster de propósito geral
- O Santos Dumont possui 18.144 núcleos de CPU, distribuídos em 756 nós computacionais
    - ✓ Thin nodes:
        - 504 nós totalizando 12.096 núcleos de CPU
        - 64 GB de memória RAM por nó
    - ✓ Hybrid nodes:
        - 252 nós totalizando 6.048 núcleos de CPU
        - 64 GB de memória RAM por nó
        - Aceleradores do tipo Nvidia K40 (GPU) e Intel Xeon Phi 7120
    - ✓ Fat node (MESCA):
        - 1 nó totalizando 240 núcleos de processamento
        - 6 TB de memória RAM

O supercomputador Santos Dumont

Supercomputador Santos Dumont, LNCC, Petrópolis.

- Com o *upgrade* o Santos Dumont ganhou 2 novos módulos:

✓1x Célula Sequana X1000 CPU
✓Total: 282 nodes
✓Cada node possui:
✓2x Intel Xeon Gold 6252 (24c @ 2.1Gz)
✓384Gb ou 768Gb de RAM
✓1x 960GB SSD.
✓1x Infiniband EDR;
✓1x GbE for Management

✓1x Célula Sequana X1000 GPU
✓Total: 94 nodes
✓Cada node possui:
✓2x Intel Xeon Gold 6252 (24c @ 2.1Gz)
✓384Gb de RAM
✓4x NVIDIA Volta V100 GPU
✓1x 960GB SSD.
✓1x Infiniband EDR;
✓1x GbE for Management

Cursos - edição verão e inverno

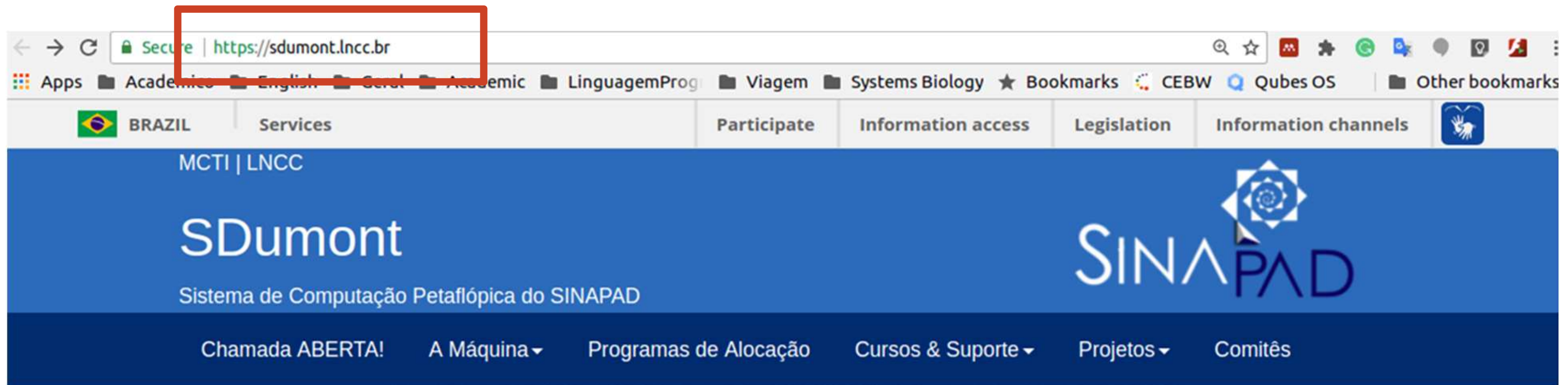
As inscrições nas atividades dos eventos do Programa de Verão 2021 acontecerão no periodo de **01 de dezembro de 2020** a **05 de janeiro de 2021** na aba Incrições.

Calendário dos Eventos do Programa de Verão 2021

Escola do Supercomputador SDumont	11 a 22 de janeiro de 2021
VII Encontro de Modelagem Matemática do Crescimento Tumoral	25 a 29 de janeiro de 2021
Jornada em Ciência de Dados	01 a 05 de fevereiro de 2021
XIV Encontro Acadêmico de Modelagem Computacional	08 a 11 de fevereiro de 2021
Jornada de Iniciação Científica e Tecnológica	11 de fevereiro de 2021
Minicursos Avulsos (não estão associados a nenhum dos eventos acima)	25 de janeiro a 05 de fevereiro de 2021

Laboratório Nacional de Computação Científica

O supercomputador SD - submissão de propostas



O QUE É R?

O QUE É O R?

- Desenvolvido a partir da linguagem S pelos estatísticos **Ross Ihaka** e **Robert Gentleman** da Universidade de Auckland (Nova Zelândia) em 1995
- Objetivo inicial era desenvolver um programa estatístico de domínio público, ou seja, com o código aberto e disponível para toda comunidade (Free Software Foundation’s GNU General Public License - GPL)
- Atualmente, o R é muito mais do que um ambiente estatístico...
  - Linguagem interpretável e de propósito geral
  - Uma das linguagens mais utilizadas em "*Big Data*"
- Fornece uma saída mínima e armazena o resultado em um objeto que pode ser integrado a outras funções
  - Encadeamento de tarefas

Por que usar o R?

Além da vantagem de ***software* livre**, o R também apresenta:

- **Multiplataforma** (Linux, macOS, Windows, …)
- **Manipulação de dados** eficaz e facilidade de armazenamento
- Uma série de **operadores para cálculos** com arranjos, especialmente matrizes
- Extensa, coerente e integrada **coleção de ferramentas** intermediárias para análise de dados
- **Instalações gráficas** para análises de dados e exibição tanto direta no computador quanto para cópia permanente (impressões)
- Linguagem que inclui condições, *loops*, funções recursivas definidas pelo usuário e instalações de entradas e saídas
- Possibilidade de **criar e compartilhar pacotes**

R no Terminal

```
karyocana — R — 80×24
karyocana@Karys-MacBook-Air ~ % R

R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin18.6.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Acessando R via terminal. Janela de comando ou console.

- ➔ Os comandos em R podem ser digitados após o *prompt* de comando **>**
- ➔ Para finalizar uma sessão utiliza-se a função `q()` ‘quit’

Instalação do ambiente integrado do R, o Rstudio

- O RStudio consiste numa ambiente integrado de desenvolvimento
  - Inclui um *console*
  - Editor de *syntax-highlighting*
  - Ferramentas para gráficos
  - *Debugging*
  - Gerenciador de *workspace*
  - ...

O RStudio

Workspace

Editor de código

Gráficos e arquivos

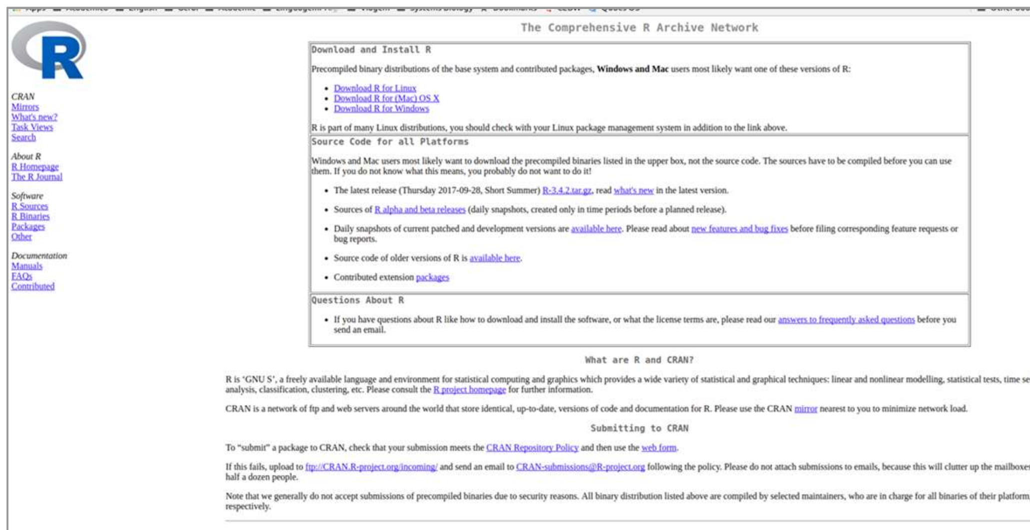
Console

Exemplo do ambiente RStudio.

Instalação de pacotes

- Uma outra vantagem do R é o uso de pacotes que o torna altamente extensível
- Pacotes são **bibliotecas** para funções gerais ou áreas de estudo específicas
- Um conjunto de pacotes é incluído com a instalação do R
- Outros pacotes estão disponíveis em repositórios de pacotes como o CRAN (The Comprehensive R Archive Network, CRAN), Bioconductor ou mesmo no github

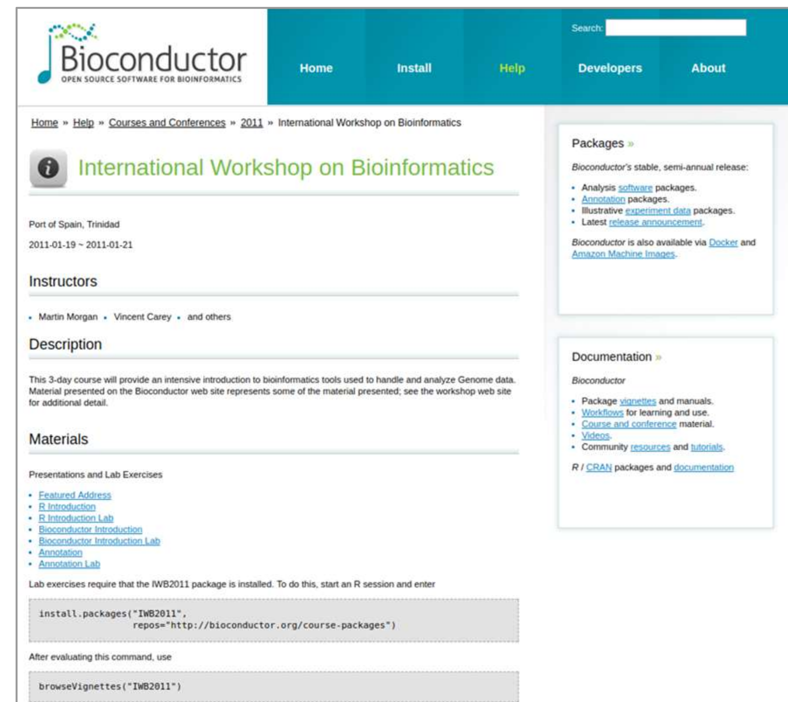
Repositório de pacotes do R



Página inicial do CRAN.

CRAN Task View: High-Performance and Parallel Computing with R

Maintainer: Dirk Eddelbuettel
Contact: Dirk.Eddelbuettel at R-project.org
Version: 2021-01-10
URL: https://CRAN.R-project.org/view=HighPerformanceComputing



Página inicial do Bioconductor.

O Pacote devtools

O pacote **devtools** não apenas facilita o processo para desenvolver pacotes R, mas também fornece outra maneira de distribuir pacotes R.

Mac/Linux:

```
devtools::install_github("hadley/devtools")
```

Windows:

```
library(devtools)
build_github_devtools()
#### Restart R before continuing ####
install.packages("devtools.zip", repos = NULL)

# Remove the package after installation
unlink("devtools.zip")
```

ANTES DE COMEÇARMOS...DICAS PRECIOSAS

- As funções em R são sempre acompanhadas por parênteses `()`
- Verifique seu diretório atual e se necessário altere-o
  - `getwd()`
  - `setwd('/home/aluno/Dados')`
- Em caso de dúvida `help('sqrt')` ou `?sqrt` ou `help.search('sqrt')`

- O buscador http://www.rseek.org/ restringe a busca para os sites que possuem conteúdo relacionado apenas à linguagem R

Por que usar o R em HPC?

Por que usar R?

O conhecimento sobre **MPI, SOCKET, C, C++** e **Fortran** podem ser **barreiras** para grande parte dos estatísticos que lidam com cálculos estatísticos intensivos e estão tentando acelerar os cálculos implementando algoritmos paralelos.

Por que usar R?

- Muitos problemas de análise computacional estatística envolvem avançados algoritmos com conjunto de dados (*datasets*) com **grande número de parâmetros** necessários para ser estimado. Tarefas **embaraçosamente paralelas** (muitos cálculos executáveis separados e independentes entre si).
  - Análise de sequência de DNA em bioinformática
  - Bootstrap
  - Redes neurais
  - Validação cruzada (*cross-validation*)

CRAN: conjunto de pacotes específicos

CRAN Task Views

CRAN task views aim to provide some guidance which packages on CRAN are relevant for tasks related to a certain topic. They give a brief overview of the included packages and can be automat
packages should be included (or excluded) - and they are *not* meant to endorse the "best" packages for a given task.

- To automatically install the views, the ctv package needs to be installed, e.g., via
  `install.packages("ctv")`
  and then the views can be installed via `install.views` or `update.views` (where the latter only installs those packages are not installed and up-to-date), e.g.,
  `ctv::install.views("Econometrics")`
  `ctv::update.views("Econometrics")`
- The task views are maintained by volunteers. You can help them by suggesting packages that should be included in their task views. The contact e-mail addresses are listed on the individual
- For general concerns regarding task views contact the ctv package maintainer.

Topics

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
ExtremeValue	Extreme Value Analysis
Finance	Empirical Finance
FunctionalData	Functional Data Analysis
Genetics	Statistical Genetics
HighPerformanceComputing	**High-Performance and Parallel Computing with R**
MetaAnalysis	Meta-Analysis
MissingData	Missing Data
ModelDeployment	Model Deployment with R
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
NumericalMathematics	Numerical Mathematics
OfficialStatistics	Official Statistics & Survey Methodology
Optimization	Optimization and Mathematical Programming
Pharmacokinetics	Analysis of Pharmacokinetic Data
Phylogenetics	Phylogenetics, Especially Comparative Methods
Psychometrics	Psychometric Models and Methods
ReproducibleResearch	Reproducible Research
Robust	Robust Statistical Methods
SocialSciences	Statistics for the Social Sciences
Spatial	Analysis of Spatial Data
SpatioTemporal	Handling and Analyzing Spatio-Temporal Data
Survival	Survival Analysis
TimeSeries	Time Series Analysis
WebTechnologies	Web Technologies and Services
gR	gRaphical Models in R

CRAN: material sobre R HPC

CRAN Task View: High-Performance and Parallel Computing with R

Maintainer: Dirk Eddelbuettel
Contact: Dirk.Eddelbuettel at R-project.org
Version: 2021-01-10
URL: https://CRAN.R-project.org/view=HighPerformanceComputing

This CRAN task view contains a list of packages, grouped by topic, that are useful for high-performance computing (HPC) with R. In this context, we are defining 'high-performance computing' rather loosely as just about anything related to pushing R a little further: using compiled code, parallel computing (in both explicit and implicit modes), working with large objects as well as profiling.

Unless otherwise mentioned, all packages presented with hyperlinks are available from CRAN, the Comprehensive R Archive Network.

Several of the areas discussed in this Task View are undergoing rapid change. Please send suggestions for additions and extensions for this task view to the task view maintainer .

Suggestions and corrections by Achim Zeileis, Markus Schmidberger, Martin Morgan, Max Kuhn, Tomas Radivoyevitch, Jochen Knaus, Tobias Verbeke, Hao Yu, David Rosenberg, Marco Enea, Ivo Welch, Jay Emerson, Wei-Chen Chen, Bill Cleveland, Ross Boylan, Ramon Diaz-Uriarte, Mark Zeligman, Kevin Ushey, Graham Jeffries, Will Landau, Tim Flutre, Reza Mohammadi, Ralf Stubner, Bob Jansen, Matt Fidler, Brent Brewington and Ben Bolder (as well as others I may have forgotten to add here) are gratefully acknowledged.

Contributions are always welcome, and encouraged. Since the start of this CRAN task view in October 2008, most contributions have arrived as email suggestions. The source file for this particular task view file now also reside in a GitHub repository (see below) so that pull requests are also possible.

The `ctv` package supports these Task Views. Its functions `install.views` and `update.views` allow, respectively, installation or update of packages from a given Task View; the option `coreOnly` can restrict operations to packages labeled as *core* below.

Direct support in R started with release 2.14.0 which includes a new package **parallel** incorporating (slightly revised) copies of packages multicore and snow. Some types of clusters are not handled directly by the base package 'parallel'. However, and as explained in the package vignette, the parts of parallel which provide snow -like functions will accept snow clusters including MPI clusters. Use `vignette("parallel")` to view the package vignette.
The **parallel** package also contains support for multiple RNG streams following L'Ecuyer et al (2002), with support for both mclapply and snow clusters.
The version released for R 2.14.0 contains base functionality: higher-level convenience functions are planned for later R releases.

CRAN: High-Performance and Parallel Computing with R

- **Parallel computing: Explicit parallelism**
- Parallel computing: Implicit parallelism
- Parallel computing: Grid computing
- Parallel computing: Hadoop
- Parallel computing: Random numbers
- Parallel computing: Resource managers and batch schedulers
- Parallel computing: Applications
- Parallel computing: GPUs
- Large memory and out-of-memory data
- Easier interfaces for Compiled code
- Profiling tools

- **<u>Parallel computing: Explicit parallelism</u>**
  rpvm Rmpi pbdMPI nws snow snowfall foreach future Rborist h2o randomForestSRC parSim qsub
- Parallel computing: Implicit parallelism
  Pnmath romp Rdsm RhpcBLASctl Rhp drake flexiblas
- Parallel computing: Grid computing
  multiR biocep-distrib
- Parallel computing: Hadoop
  RHIPE rmr segue RProtoBuf HistogramTools
- Parallel computing: Random numbers
  Rlecuyer rstream sitmo dqrng doRNG
- Parallel computing: Resource managers and batch schedulers
  Rslurm Condor toolkit sfCluster batch BatchJobs flowr clustermq
- Parallel computing: Applications
  caret maanova pvclust tm varSelRF bcp multtest Matching bugsparallel partDSA dclone pbapply keras mvnfast
- Parallel computing: GPUs
  rgpu gcbd OpenCL permGPU tensorflow tfestimators Bdgraph ssgraph
- Large memory and out-of-memory data
  biglm ff bigmemory sqldf data.table HadoopStreaming speedglm MonetDB ffbase LaF bigstatsr disk.frame
- Easier interfaces for Compiled code
  Inline Rcpp RcppParallel rJava reticulate
- Profiling tolos
  profr profvis proftools aprof GUIProfiler

CRAN: High-Performance and Parallel Computing with R

➔ **Parallel computing: Explicit parallelism**

Nome	Descrição	Data de publicação
Rmpi	Interface (Wrapper) to MPI (Message-Passing Interface)	2018-11-13
pbdSLAP	Programming with Big Data – Scalable Linear Algebra Packages	2020-02-28
pbdMPI	Programming with Big Data – Interface to MPI	2020-01-29
future	Unified Parallel and Distributed Processing in R for Everyone	2020-12-10

rpvm Rmpi pbdMPI nws snow snowfall foreach future Rborist h2o randomForestSRC parSim qsub

Utilizando o R no Santos dumont

Carregar o módulo R no Santos Dumont

```
$ module load R/3.5.2_openmpi_2.0_gnu
```

É com esse módulo é que os exemplos estão funcionando, mas existem outros módulos e outras versões.

Use `$ module avail` para ver todos os módulos disponíveis.

O $SCRATCH

IMPORTANTE!!! DIRECIONAR PARA O SCRATCH

CD $SCRATCH

Scratch: Estrutura montada a partir do diretório **/scratch.** Utilizado para armazenar todos os arquivos que serão utilizados durante a execução de um *job* (*scripts* de submissão, executáveis, dados de entrada, dados de saída etc).

Página do supercomputador Santos Dumont: http://sdumont.lncc.br/

Pacotes instalados globalmente

- Vários pacotes já encontram-se instalados, inclusive os comentados no minicurso.
- Alguns deles:
  - bigmemory
  - doMPI
  - doParallel
  - doSNOW
  - foreach
  - microbenchmark
  - parallel
  - rbenchmark
  - snow
  - snowfall
- Para ver a lista completa, use a função `installed.packages()`

Instalação de pacotes no Santos Dumont

Se desejar usar um pacote não instalado, é possível fazer uma instalação local de um pacote da seguinte maneira:

1. Crie um diretório no scratch para manter os pacotes
   - `mkdir /scratch/projeto/usuario/R`
2. Instale os pacotes no diretório criado
   - `R -e "install.packages('raster', repos='http://cran.rstudio.com/', lib='/scratch/projeto/usuario/R')"`
3. Use a variável de ambiente R_LIBS_USER para indicar o path do diretório com os pacotes
   - `export R_LIBS_USER='/scratch/projeto/usuario/R'`

Script de submissão

Do Santos Dumont

```
#!/bin/sh

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=20
#SBATCH -p treinamento_gpu
#SBATCH -J nome_do_job
#SBATCH --exclusive

#source /scratch/app/modulos/intel-psxe-2016.2.062.sh
#module load openmpi/icc/2.0.4.2
module load R/3.5.2_openmpi_2.0_gnu
#module load R/3.3.1_intel

# Suprime aviso sobre uso de fork()
export OMPI_MCA_mpi_warn_on_fork=0
ulimit -s 10240

EXEC=${1}

$EXEC
```

Instalação de pacotes no Santos Dumont

Da documentação oficial:

> The library search path is initialized at startup from the environment variable ‘R_LIBS’ (which should be a colon-separated list of directories at which R library trees are rooted) followed by those in environment variable ‘R_LIBS_USER’. Only directories which exist at the time will be included.

Instalação de pacotes no Santos Dumont

- Também é possível modificar `.libPaths()` manualmente dentro de um *script*:

```
>.libPaths(c('/scratch/projeto/usuario/minha_library', .libPaths()))
```

- Útil para testar outras versões de pacotes já instalados, sem perder a instalação anterior.

Família apply

Função apply

- Usada para aplicar uma função às linhas ou colunas de uma matriz
- O retorno é um vetor ou *array*
- Exemplo

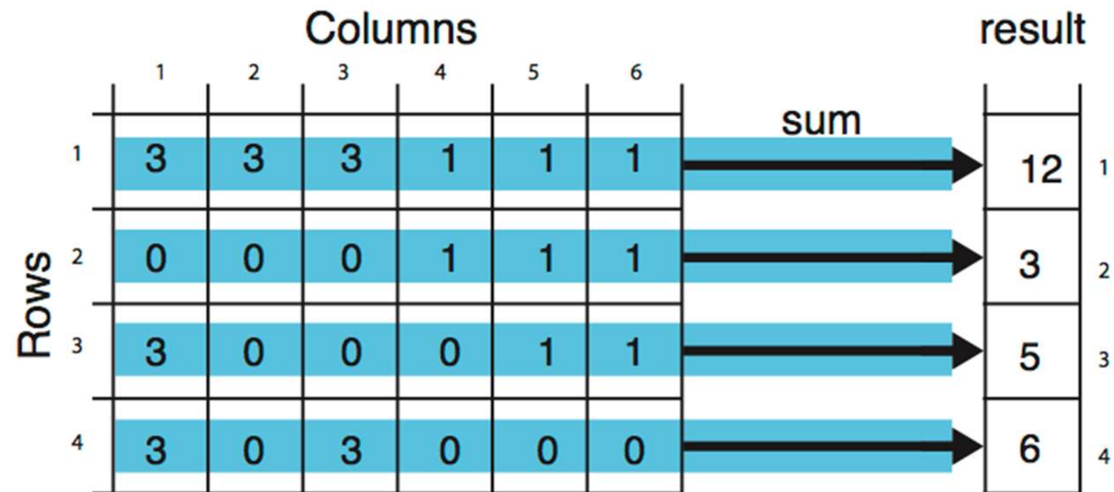
```
> m <- matrix(seq(1,16), 4, 4)
> m
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

Função apply

```
> # Aplica min() às linhas
> apply(m, 1, min)
[1] 1 2 3 4

> # Aplica max() às colunas
> apply(m, 2, max)
[1]  4  8 12 16
```

Função apply

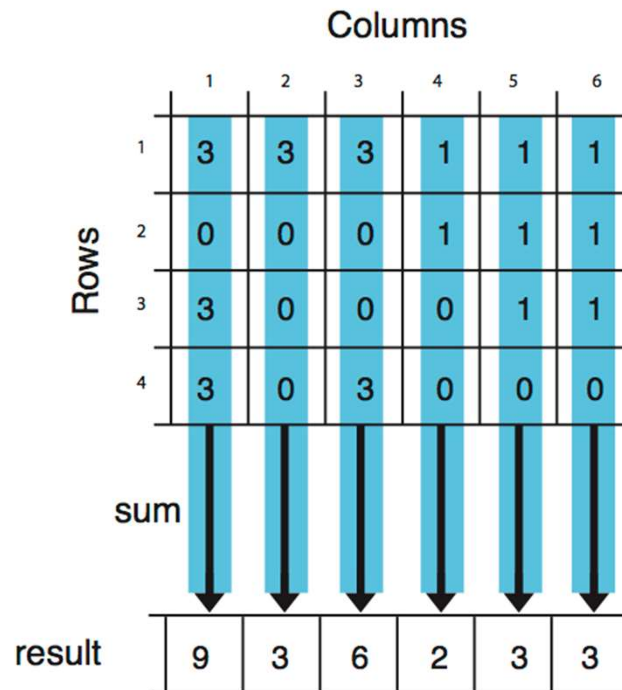
```
result<-apply(mat,1,function(x) sum(x) )
resut<-apply(mat,1,sum)
```



De Notes on Computational Genomics with R de Altuna Akalin

Função apply

```
result<-apply(mat,2,function(x) sum(x))
result<-apply(mat,2,sum)
```



De Notes on Computational Genomics with R de Altuna Akalin

Função apply

- Execute `apply.R` para ver apply() em ação.

- Gere uma matriz 100 por 100 de números aleatórios e encontre o menor valor de cada linha e de cada coluna.
- Dica: funções `runif()` e `min()`.

Função lapply

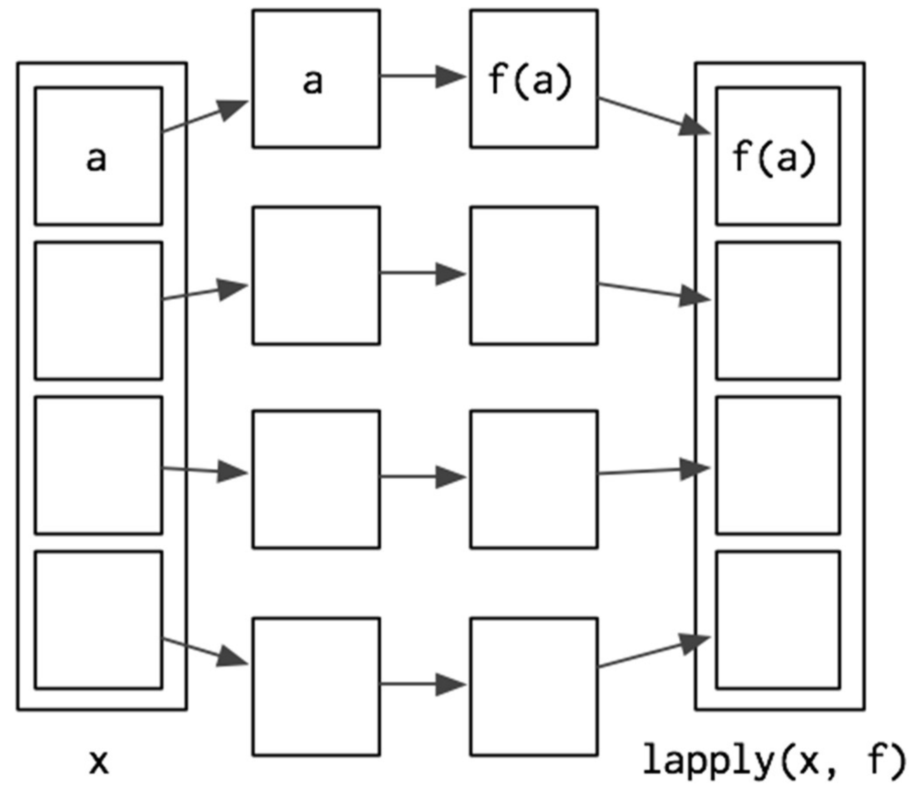
- Aplica uma função à cada elemento de uma lista
- **Retorna sempre uma lista**
- Exemplo

```
> x <- list(a = 1, b = 1:10, c = c("Olá", "Mundo"))
> x
$a
[1] 1

$b
[1]  1  2  3  4  5  6  7  8  9 10

$c
[1] "Olá"   "Mundo"
```

Função lapply

```
> # Aplica a função length() a cada elemento
> lapply(x, FUN = length)
$a
[1] 1

$b
[1] 10

$c
[1] 2
```

Função lapply



Do livro Advanced R de Hadley Wickham

Função lapply

- Veja lapply() em ação em lapply.R

- Gere uma lista com 3 elementos...
  - 10 números aleatórios
  - uma matriz 4 x 4
  - Uma string
- e gere uma lista de 3 elementos em que cada elemento terá as dimensões de cada um dos objetos listados acima

Função sapply

- Aplica uma função à cada elemento de uma lista
- **Retorna um vetor ou matriz ao invés de uma lista**
- Exemplo

```
> x <- list(a = 1, b = 1:10, c = c("Olá", "Mundo"))
> x
$a
[1] 1

$b
[1]  1  2  3  4  5  6  7  8  9 10

$c
[1] "Olá"   "Mundo"
```

Função sapply

```
> # Aplica a função length() a cada elemento
> # retornando um vetor
> sapply(x, FUN = length)
 a  b  c
 1 10  2
```

- Considere o uso de sapply() no lugar de unlist(lapply())

```
> unlist(lapply(x, FUN=length))
 a  b  c
 1 10  2
```

Função sapply

sapply() pode retornar uma matriz se a função aplicada retornar vetores de mesmo tamanho

```
> estados <- function(x) sample(state.name, 3)

> estados(1)
[1] "Hawaii"       "South Dakota" "Montana"

> sapply(1:4, FUN=estados)
     [,1]            [,2]        [,3]        [,4]
[1,] "Kentucky"      "Oregon"    "Nebraska"  "Oregon"
[2,] "West Virginia" "Hawaii"    "Louisiana" "Kansas"
[3,] "Iowa"          "Minnesota" "Oregon"    "Tennessee"
```

Função sapply

- Veja sapply() em ação em sapply.R

- Gere uma lista com 3 elementos...
  - 10 números aleatórios
  - uma matriz 4 x 4
  - Uma string
- e gere **um vetor** de 3 elementos em que cada elemento terá as dimensões de cada um dos objetos listados acima

Função mapply

- `mapply()` é a versão multivariável de `sapply()`.
- Ela aplica a função aos primeiros elementos das estruturas de dados, depois aos segundos, aos terceiros e assim por diante.
- A função deve aceitar múltiplos argumentos.
- **Retorna vetor ou matriz sempre que possível.**
- **Se não, retorna lista.**

Função mapply

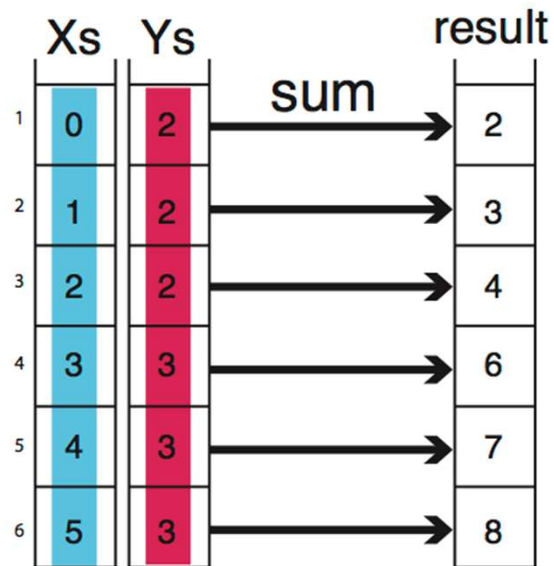
```
> # Retorna um vetor de 5 elementos com
> # sum(1, 5, 1) na 1ª posição,
> # sum(2, 4, 2) na 2ª posição,
> # sum(3, 3, 3) na 3ª posição,
> # sum(4, 2, 4) na 4ª posição e
> # sum(5, 1, 5) na 5ª posição.

> mapply(sum, 1:5, 5:1, 1:5)
[1]  7  8  9 10 11
```

Função mapply

```
> # Faz rep(1, 4), rep(2, 3), rep(3, 2), rep(4, 1)
> # Nesse caso, só é possível retornar uma lista
> mapply(rep, 1:4, 4:1)
[[1]]
[1] 1 1 1 1

[[2]]
[1] 2 2 2

[[3]]
[1] 3 3

[[4]]
[1] 4
```

Função mapply

```
result<-mapply(function(x,y) sum(x,y),Xs,Ys)
result<-mapply(sum,Xs,Ys)
```



De Notes on Computational Genomics with R de Altuna Akalin

Função mapply

- Veja mapply() em ação em mapply.R

- Gere 2 vetores
  - X: um com números de 1 a 10
  - Y: um com números de 10 a 1
- e gere **um vetor** de 10 elementos em que cada elemento será o resultado de X[1]^Y[1], X[2]^Y[2] … X[10]^Y[10]

Pacotes de paralelismo

Pacote parallel

- Disponível na instalação base do R desde a versão 2.14.0.
- Baseado nos pacotes snow e multicore.
  - **multicore** para computação paralela em plataformas de memória compartilhada
  - **snow** para computação paralela em plataformas de memória distribuída
- Fornece substitutos para a maioria das funções desses pacotes.

Pacote parallel

O processamento com o pacote parallel segue o seguinte modelo:

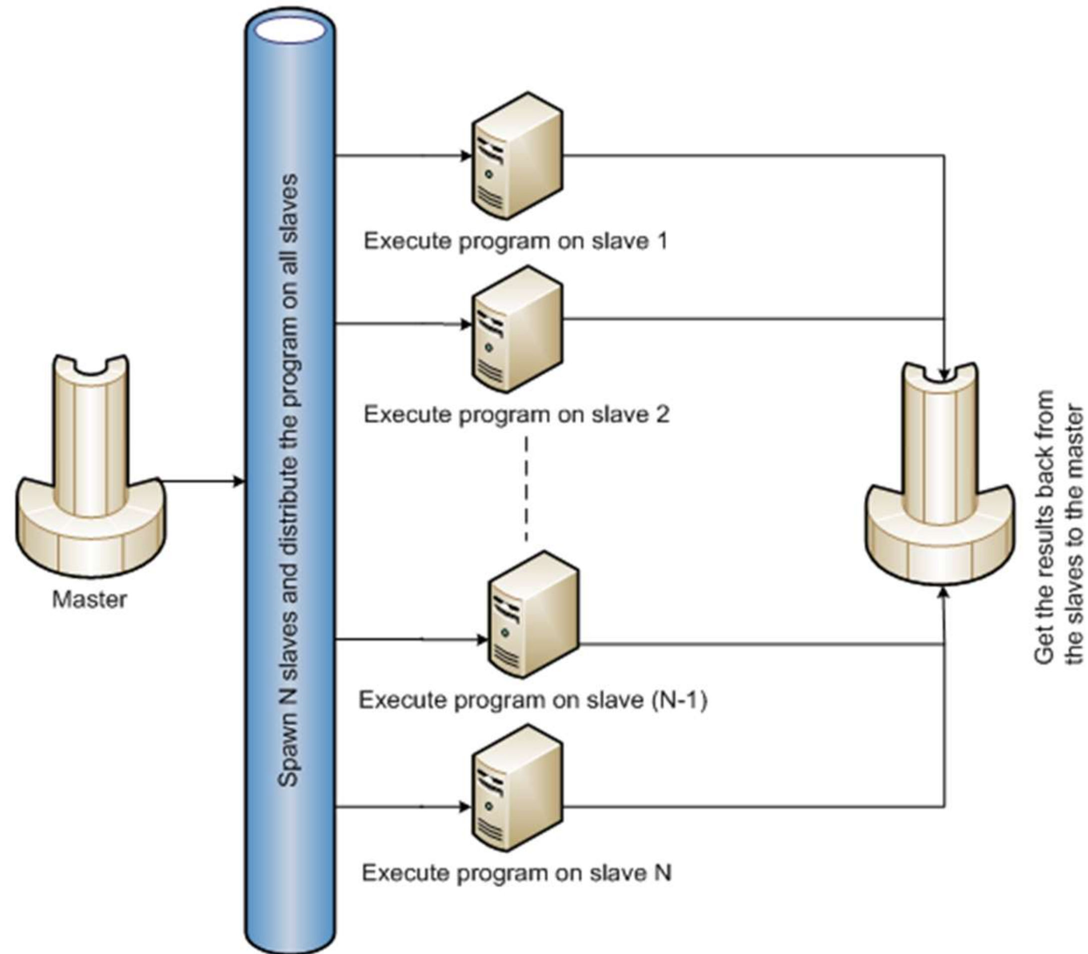
a. M processos *workers* são iniciados
b. Os dados necessários para completar a tarefa são enviados para cada *worker*
c. A tarefa é dividida em M pedaços de tamanho aproximadamente iguais e eles são enviados para os *workers*
d. Espera-se até que todos os *workers* completem suas tarefas
e. Os passos b-d são repetidos para as tarefas seguintes
f. Os processos *workers* são terminados

Pacote parallel

- Esse modelo é implementado em funções que são as contrapartes paralelas para as funções da família apply, possibilitando o processamento de cada elemento do vetor/lista de entrada em um *core* diferente.

- Um modelo ligeiramente diferente é dividir a tarefa em $M_1 > M$ pedaços, enviar os primeiros M pedaços para os *workers* e repetidamente aguardar um *worker* terminar e enviar o pedaço restante.
    - “Balanceamento de carga” (*load balancing*)

Pacote parallel



Funções do Pacote Parallel

➔ Memória Compartilhada

Função	Descrição	Exemplo
detectCores	Detecta o número de núcleos da CPU	ncores <- detectCores()
mclapply	versão paralelizada de lapply	mclapply(1:5, runif, mc.cores = ncores)

➔ Memória Distribuída

Função	Descrição	Exemplo
makeCluster	Inicia o cluster[1]	cl <- makeCluster(10, type="MPI")
clusterSetRNGStream	Define o seed no cluster	clusterSetRNGStream(cl, 321)
clusterExport	Exporta variáveis para os workers	clusterExport(cl, list(a=1:10, x=runif(10)))
clusterEvalQ	Avalia expressões nos workers	clusterEvalQ(cl, {x <- 1:3 myFun <- function(x) runif(x)})
clusterCall	Chama uma função em todos os workers	clusterCall(cl, function(y) 3 + y, 2)
parLapply	Versão paralelizada de lapply	parLapply(cl, 1:100, Sys.sleep)
parLapplyLB	parLapply com balanceamento de carga	parLapplyLB(cl, 1:100, Sys.sleep)
stopCluster	Para o cluster	stopCluster(cl)

([1]os tipos permitidos são PSOCK, FORK, SOCK, MPI e NWS)

Pacote parallel

- Para iniciar os processos *workers*, use a função makeCluster()
  - Ela recebe por parâmetro o número de *workers* e o tipo de comunicação entre eles
- Para encerrar os workers, use a função stopCluster()
  - Ela recebe por parâmetro o objeto criado via makeCluster()

Pacote parallel

```
# Cria um cluster de 4 workers com comunicação MPI
cl <- makeCluster(4, type = "MPI")

# Faz trabalho em paralelo


# Para os workers
stopCluster(cl)
```

Função parApply

- parApply() é análoga à apply()
- A diferença no uso é o 1° argumento: ele deve ser o objeto cluster criado via makeCluster()
- parRapply() é oferecida como conveniência: ela opera automaticamente nas **linhas**
- parCapply() é oferecida como conveniência: ela opera automaticamente nas **colunas**
- A documentação cita que parRapply() e parCapply() podem ser mais eficientes que parApply() em algumas situações

Função parApply

- Execute parApply.R para ver parApply() em ação.

sbatch exemplo.sbatch ./parApply.R

- Gere uma matriz 100 por 100 de números aleatórios e encontre o menor valor de cada linha e de cada coluna.
- Dica: funções runif() e min().
- Faça isso num cluster de 10 *workers*.

Função parLapply

- parLapply() é análoga à lapply()
- Também deve receber o objeto cluster criado via makeCluster() via 1º parâmetro
- **Retorna sempre uma lista**
- Possui uma versão com balanceamento de carga: parLapplyLB()
  - Use-a quando aplicar a função a diferentes elementos da lista de entrada levar tempos muito diferentes
  - Adiciona *overhead* de comunicação

Função parLapply

- Veja parLapply() em ação em parLapply.R

- Gere uma lista com 3 elementos...
    - 10 números aleatórios
    - uma matriz 4 x 4
    - Uma string
- e gere uma lista de 3 elementos em que cada elemento terá as dimensões de cada um dos objetos listados acima.
- Faça isso num cluster de 3 *workers*.

Função parSapply

- parSapply() é análoga à sapply()
- Também deve receber o objeto cluster criado via makeCluster() via 1º parâmetro
- **Retorna um vetor ou matriz**
- Possui uma versão com balanceamento de carga: parSapplyLB()
  - Use-a quando aplicar a função a diferentes elementos da lista de entrada levar tempos muito diferentes
  - Adiciona *overhead* de comunicação

Função parSapply

- Veja parSapply() em ação em parSapply.R
- Gere uma lista com 3 elementos...
  - 10 números aleatórios
  - Uma matriz 4 x 4
  - Uma string
- Gere **um vetor** de 3 elementos em que cada elemento terá as dimensões de cada um dos objetos listados acima
- Faça isso num cluster de 3 *workers*.

Função clusterMap

- clusterMap() é análoga à mapply()
- Também deve receber o objeto cluster criado via makeCluster() via 1º parâmetro.
- A função chamada deve receber múltiplos argumentos.
- **Retorna sempre uma lista, ao contrário de mapply()**

Função clusterMap

- Veja clusterMap() em ação em clusterMap.R

- Gere 2 vetores
  - X: um com números de 1 a 10
  - Y: um com números de 10 a 1
- Gere **uma lista** de 10 elementos em que cada elemento será o resultado de X[1]^Y[1], X[2]^Y[2] … X[10]^Y[10]
- Faça isso num cluster de 10 *workers*.

Função clusterExport

- clusterExport() faz a(s) variável(is) do mestre passadas por parâmetros disponíveis aos processos *workers*.
- Um vetor de *strings* com os nomes das variáveis a serem disponibilizadas devem ser passadas por parâmetro, não as variáveis em si.
- Exemplo

```
> expoente <- 3

> clusterExport(cl, "expoente")
```

Função clusterExport

- Veja os efeitos de clusterExport() em clusterExport.R
- Comente a linha...

```
clusterExport(cl, "expoente")
```

e teste novamente. O que aconteceu? Por quê?

Função clusterCall

- clusterCall() chama uma função em cada *worker* do cluster, com argumentos idênticos.
- Os argumentos **são avaliados no mestre** e seus valores são transmitidos aos *workers* que executam a função.

Função clusterEvalQ

- clusterEvalQ() avalia uma expressão literal em **cada *worker* do cluster**.
- É uma versão paralela de evalq()
- Disponibilizada como conveniência: chama clusterCall(cl, evalq, expr) para cada *worker*.
- Muito usada para carregar um pacote nos workers: clusterEvalQ(cl, library(PACOTE))

Funções clusterEvalQ e clusterCall

- Execute clusterEvalQ.R para ver a diferença entre clusterEvalQ() e clusterCall().

Função clusterApply

- clusterApply() recebe uma sequência de argumentos (vetor ou lista) e uma função.
- Chama a função com o 1º elemento da sequência como parâmetro da função no 1º *worker* do cluster, o 2º elemento da sequência como argumento da função no 2º *worker* do cluster e assim por diante.

Função clusterApply

- O tamanho da lista de argumentos deve ser menor ou igual à quantidade de *workers* no cluster.
  - Existe uma versão com balanceamento de carga, clusterApplyLB(), que não possui essa limitação.
- **Retorna sempre uma lista.**
- Muito parecida com parLapply(), mas não retorna uma lista nomeada.

- Exercício: substitua parLapply() por clusterApply() em parLapply.R

Função clusterSplit

- clusterSplit() divide uma sequência em pedaços.
- Retorna um pedaço para cada *worker* em um cluster.
- Cada pedaço traz elementos consecutivos.
- **Retorna sempre uma lista.**
- Toda a computação é feita no mestre.

Função clusterSplit

- Exemplo

```
> # cluster com 4 workers
clusterSplit(cl, 1:8)
[[1]]
[1] 1 2

[[2]]
[1] 3 4

[[3]]
[1] 5 6

[[4]]
[1] 7 8
```

Função clusterSplit

- Usada para fatiar uma entrada em partes iguais a serem processadas pelos *workers*.
- É importante dividir seu trabalho de forma a reduzir o *overhead* de comunicação.
- Veja isso na prática em `clusterSplit.R`

Função detectCores

- detectCores() tenta retornar a quantidade de cores da máquina.
- Pode ser útil para descobrir que quantidade de *workers* criar.
- Não muito útil em clusters.

Função clusterSetRNGStream

- clusterSetRNGStream() configura a seed do gerador de números aleatórios.
- Use para garantir reprodutibilidade em execuções em paralelo.
- Execute clusterSetRNGStream.R múltiplas vezes para testar.

Correspondência apply - parallel

apply()	parApply()
apply(,1,)	parRapply()
apply(,2,)	parCapply()
lapply()	parLapply() / parLapplyLB()
sapply()	parSapply() / parSapplyLB()
mapply()	clusterMap()
evalq()	clusterEvalQ()
set.seed()	clusterSetRNGStream()

FOREACH

Pacote foreach

- Oferece suporte a uma construção de loop que permite iterar sobre elementos de uma coleção, sem o uso de uma variável contadora explícita.
- Usado pelo seu **valor de retorno**, não pelos seus efeitos colaterais.
- Permite execuções em paralelo após o registro de um *backend* paralelo.

Pacote foreach

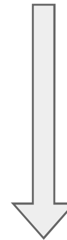
```
> x <- foreach(i=1:3) %do% sqrt(i)
> x

[[1]]
[1] 1

[[2]]
[1] 1.414214

[[3]]
[1] 1.732051
```

Retorna um valor, ao contrário do for tradicional.

Por padrão, retorna uma lista

Pacote foreach

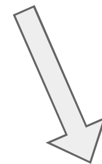
É possível especificar uma
função para combinar a saída



```
> x <- foreach(i=1:3, .combine='c') %do% exp(i)
> x

[1]  2.718282  7.389056 20.085537
```

Pacote foreach

Se a expressão retorna um vetor, é possível combiná-los numa matriz.



```
> x <- foreach(i=1:4, .combine='cbind') %do% rnorm(4)
> x

      result.1   result.2   result.3   result.4
[1,]  1.351555  0.6232773  1.3019642  0.4170242
[2,] -1.032726  1.5233215 -1.6029630 -0.8573203
[3,]  1.399842 -0.1439332  0.1347221  0.2907462
[4,]  1.343903  1.7250600  1.4653007  0.2795696
```

Pacote foreach

- Para fazer execuções em paralelo é necessário registrar um *backend* paralelo.
- Alguns disponíveis:
  - doMC
  - doSNOW
  - **doParallel**

Pacote foreach

```
library(foreach)
library(doParallel)

cl<-makeCluster(...)
registerDoParallel(cl)

foreach(...) %dopar% {
        # instruções
}

stopCluster(cl)
```

A instrução muda para **%dopar%**

Pacote foreach

- Execute doParallel.R para ver foreach() em ação.
- Meça o tempo de execução de...

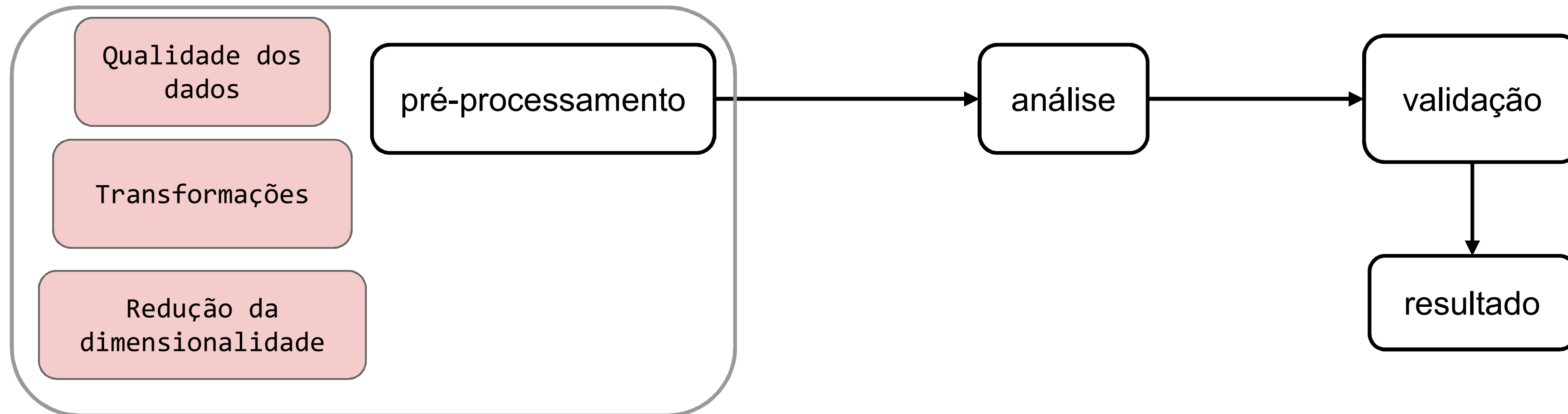
```
out <- foreach(x=a, y=b, z=c) %dopar% {
        (x + y)**z
}
```

Com a técnica empregada em clusterSplit.R

- Troque o operador **%dopar%** por **%do%** e meça novamente.

Benchmark no R

Fluxo de tarefas em ciência de dados



- ★ Etapas comuns em várias áreas das ciências experimentais
- ★ Muitos scripts em R fazem uso desse encadeamento de atividades/tarefas
- ★ Gargalos computacionais: do maior para o menor

Exemplo 1: função sys.time (R-base)

```
do_lm = function(value){
   X <- matrix(rnorm(value), 100, 10)
   y <- X %*% sample(1:10, 10) + rnorm(100)
   b <- lm(y ~ X + 0)$coef
}

start_time <- Sys.time()
do_lm(1000)
end_time <- Sys.time()

print(end_time - start_time)
```

```
## Time difference of 0.00741148 secs
```

Execute o script t1_bench_sys-time.R

Exemplo 2: Função system.time (R-base)

```r
do_lm = function(value){
   X <- matrix(rnorm(value), 100, 10)
   y <- X %*% sample(1:10, 10) + rnorm(100)
   b <- lm(y ~ X + 0)$coef
}

system.time({do_lm(1000)})
```

```
##    user  system elapsed
##   0.002   0.000   0.002
```

wall clock time

Execute o script t2_bench_systemTime.R

Exemplo 3: Pacote rbenchmark

- Wrapper do systems.time()
- Mais funcionalidades
- Retorno como data.frame

Execute o script
t3_bench_rbenchmark.R

```r
library(rbenchmark)

benchmark("lm" = {
            X <- matrix(rnorm(1000), 100, 10)
            y <- X %*% sample(1:10, 10) + rnorm(100)
            b <- lm(y ~ X + 0)$coef
          },
          "pseudoinverse" = {
            X <- matrix(rnorm(1000), 100, 10)
            y <- X %*% sample(1:10, 10) + rnorm(100)
            b <- solve(t(X) %*% X) %*% t(X) %*% y
          },
          "linear system" = {
            X <- matrix(rnorm(1000), 100, 10)
            y <- X %*% sample(1:10, 10) + rnorm(100)
            b <- solve(t(X) %*% X, t(X) %*% y)
          },
          replications = 1000,
          columns = c("test", "replications", "elapsed","relative",
                      "user.self", "sys.self")
          )
```

```
##            test replications elapsed relative user.self sys.self
## 3 linear system         1000   0.272    1.000     0.272    0.000
## 1            lm         1000   1.386    5.096     1.373    0.012
## 2 pseudoinverse         1000   0.285    1.048     0.284    0.000
```

Exemplo 4: Pacote microbenchmark

- Mais funcionalidades
  - ggplot2
- Permite inserir funções criadas pelo usuário
- Resultado das funções devem retornar valores iguais

Execute o script

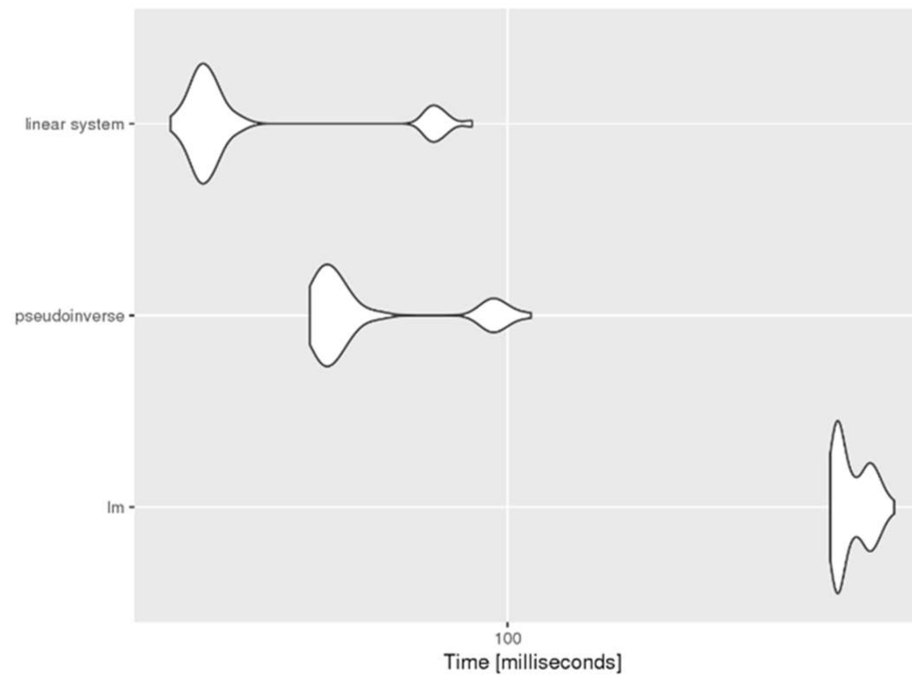
t4_bench_microbenchmark.R

```
## Unit: milliseconds
##           expr       min        lq      mean    median        uq       max
##             lm 335.73504 344.59359 365.34562 353.33344 387.12642 426.62201
##  pseudoinverse  47.61561  50.06029  61.59383  52.35680  59.71544 109.18576
##  linear system  28.24556  31.32272  40.82109  32.58404  35.16226  87.50643
##  neval cld
##    100   c
##    100  b
##    100 a
```

```r
library(microbenchmark)

set.seed(2017)
n <- 10000
p <- 100
X <- matrix(rnorm(n*p), n, p)
y <- X %*% rnorm(p) + rnorm(100)

check_for_equal_coefs <- function(values){
  tol <- 1e-12
  max_error <- max(c(abs(values[[1]] - values[[2]]),
                     abs(values[[2]] - values[[3]]),
                     abs(values[[1]] - values[[3]])))
  max_error < tol
}

mbm <- microbenchmark("lm" = { b <- lm(y ~ X + 0)$coef },
               "pseudoinverse" = {
                 b <- solve(t(X) %*% X) %*% t(X) %*% y
               },
               "linear system" = {
                 b <- solve(t(X) %*% X, t(X) %*% y)
               },
               check = check_for_equal_coefs)

mbm
```

Exemplo 4: Pacote microbenchmark

```r
library(ggplot2)
autoplot(mbm)
```



Boas Práticas na escrita do código

- Pequenas alterações no código R podem ter grandes efeitos no tempo de execução das funções

Exemplo 1: um loop ineficiente

```r
system.time({
  for (i in 1:nrow(df)) {
    if ((df[i, 'col1'] + df[i, 'col2'] + df[i, 'col3'] + df[i, 'col4']) > 4) { # verifica se o num e > 4
      df[i, 5] <- "greater_than_4"
    } else {
      df[i, 5] <- "lesser_than_4"
    }
  }
})
```

```
##    user  system elapsed
## 603.174  18.983 623.079
```

Execute o script
speedupWithoutPar.R

Exemplo 2: utilizando uma estrutura pré alocada

```
output <- character (nrow(df)) # inicializa um vetor de saida
system.time({
  for (i in 1:nrow(df)) {
    if ((df[i, 'col1'] + df[i, 'col2'] + df[i, 'col3'] + df[i, 'col4']) > 4) {
      output[i] <- "greater_than_4"
    } else {
      output[i] <- "lesser_than_4"
    }
  }
  df$output})
```

```
##    user  system elapsed
##  23.062   0.002  23.073
```

Exemplo 3: Verificando determinada condição fora do loop

```r
output <- character (nrow(df))
condition <- (df$col1 + df$col2 + df$col3 + df$col4) > 4  # verifica a condicao fora do loop
system.time({
  for (i in 1:nrow(df)) {
    if (condition[i]) {
      output[i] <- "greater_than_4"
    } else {
      output[i] <- "lesser_than_4"
    }
  }
  df$output <- output
})
```

```
##    user  system elapsed
##   0.091   0.000   0.091
```

Exemplo 4: Utilizando o Which

```
system.time({
  want = which(rowSums(df) > 4)
  output = rep("less than 4", times = nrow(df))
  output[want] = "greater than 4"
})
```

Exemplo 5: Usando o ifelse

```
system.time({
  output <- ifelse ((df$col1 + df$col2 + df$col3 + df$col4) > 4, "greater_than_4", "lesser_than_4")
  df$output <- output
})
```

```
##    user  system elapsed
##   0.092   0.010   0.102
```

Melhorando o código R, boas praticas

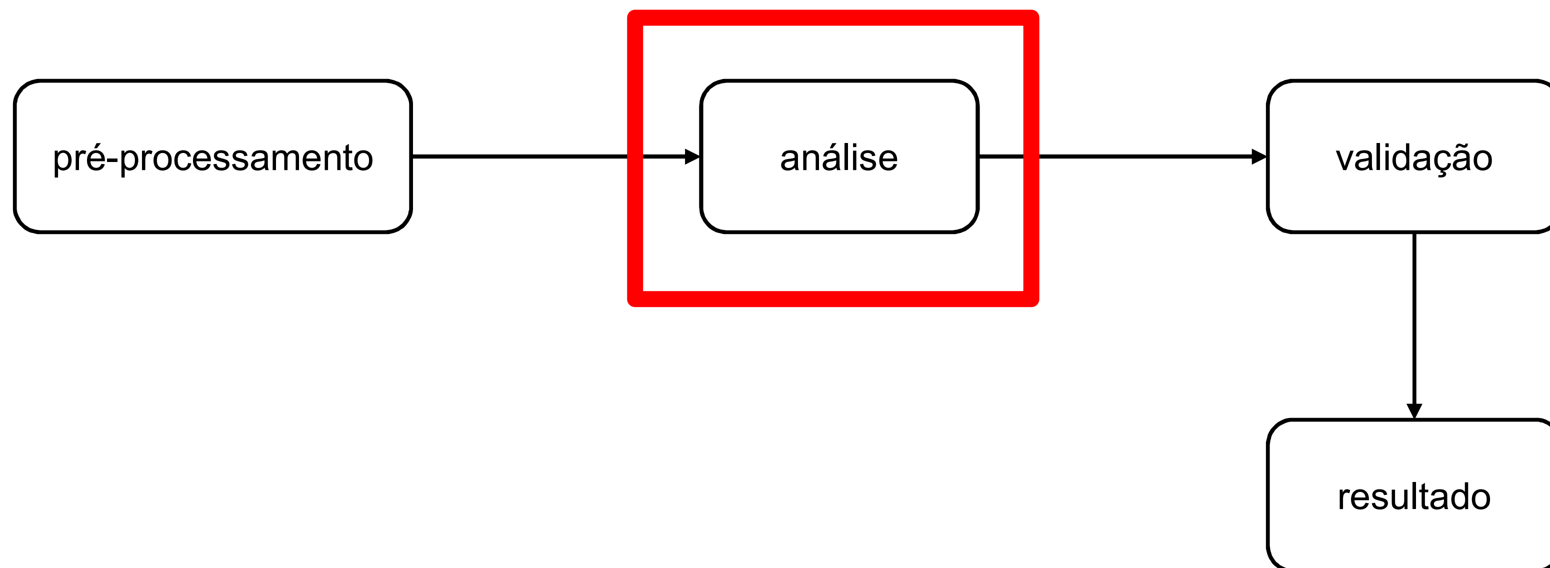
Boas Práticas na escrita do código

- Comente o código para que esse torne legível aos demais colaboradores e até mesmo para você.
- Seja claro, conciso e significativo ao criar o nome das variáveis
  - Geralmente nome de variáveis: substantivos e nomes de funções: verbos
- Seja explícito sobre os requisitos e dependências do seu código
- Identifique e separe componentes distintos no seu código -> modularize!!!
- Use um estilo consistente no seu código.
  - Exemplo: nomeie todas as matrizes com algo terminando em `_mat`

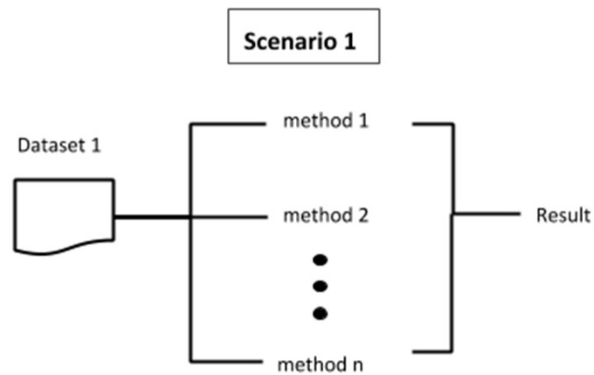
Boas Práticas - outras dicas

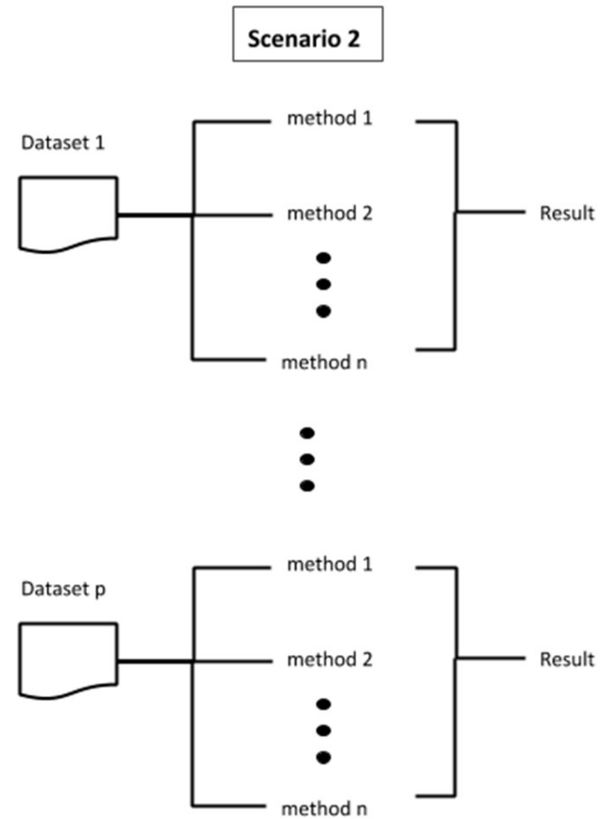
- Remova objetos não utilizados e que carregam a memória.
  - use rm()
  - Não salve o workspace
- Sempre que possível, utilize estruturas que consumam menos memória. Por exemplo, data.table
- Utilize controle de versão quando você compartilha o código
  - Ex: Github, Gitlab, ...

Estudo de caso: paralelismo de tarefas



Paralelismo de tarefas - Possíveis cenários



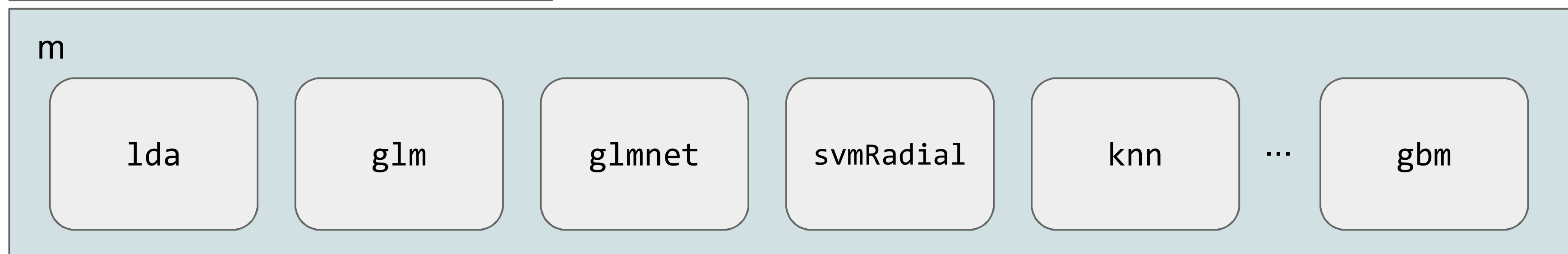


Exemplo de métodos em Machine learning

- Classificação binária
- Conjunto de dados extraído do UCI ML repository
    - Pima Indians dataset
        - Dados clínicos sobre o surgimento de diabetes após um período de 5 anos
    - Câncer de mama
- Utiliza o pacote caret
    - Fornece uma interface amigável para vários algoritmos diferentes em ML e ferramentas úteis para avaliar e comparar modelos
- Exemplo extraído e adaptado a partir de: **How to Evaluate Machine Learning Algorithms with R**
    - https://machinelearningmastery.com/evaluate-machine-learning-algorithms-with-r/

Cenário 2

```
d <- list(dataset1, dataset2)
m <- c('lda', 'glm', 'glmnet', 'svmRadial', 'knn', 'nb',
'rpart', 'C5.0', 'treebag', 'rf', 'gbm')
```



Cenário 2

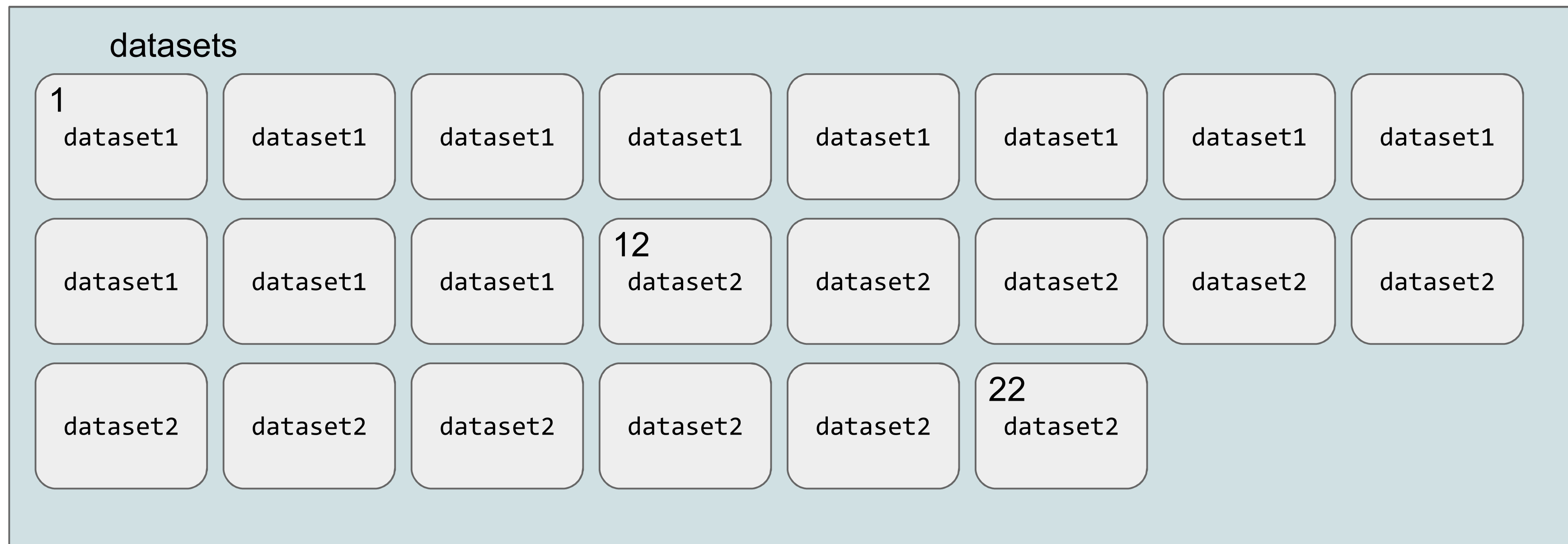
```
methods <- rep(m, length(d))
```

methods

1 lda	glm	glmnet	svmRadial	knn	nb	rpart	C5.0
treebag	rf	gbm	12 lda	glm	glmnet	svmRadial	knn
nb	rpart	C5.0	treebag	rf	22 gbm		

Cenário 2

```
datasets <- rep(d, each=length(m))
```



Cenário 2

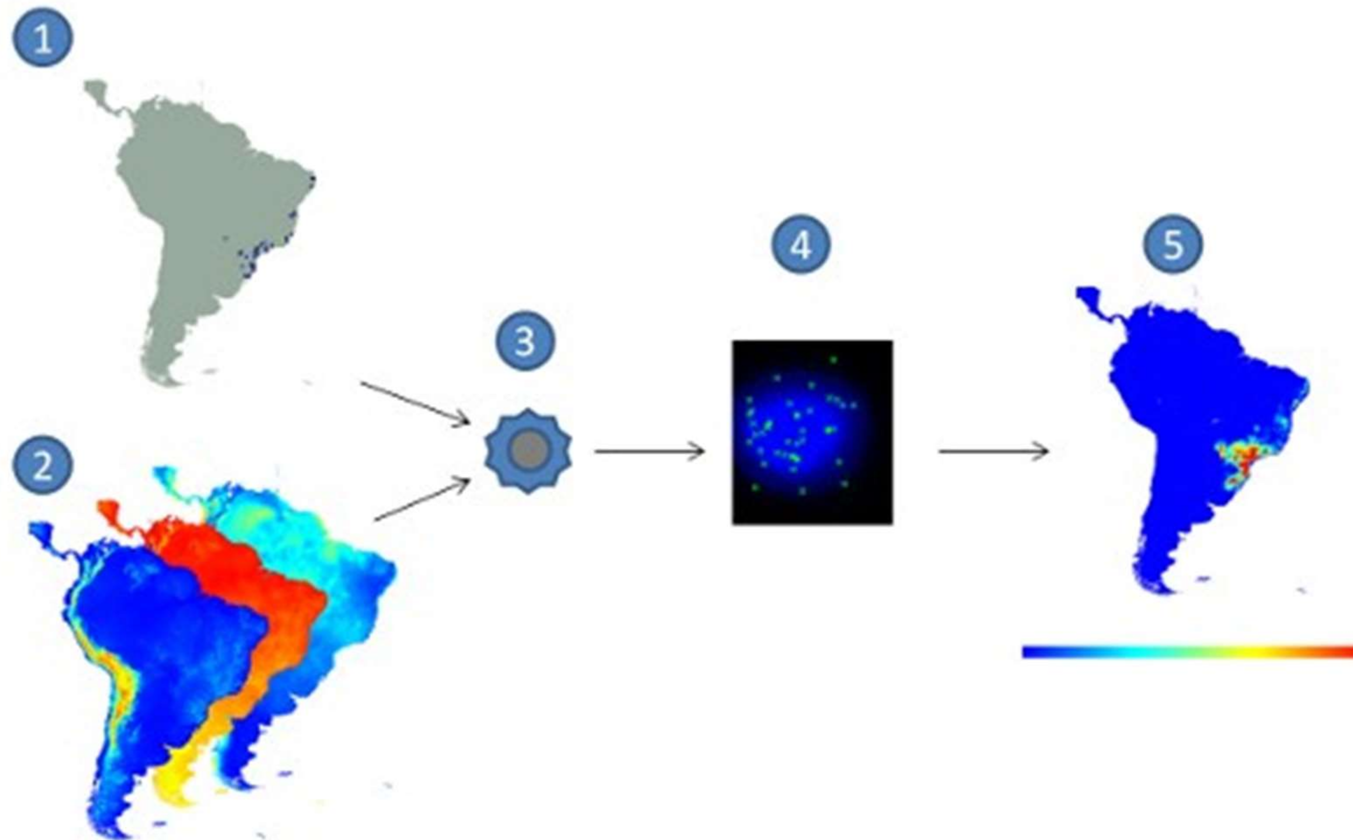
Worker	Modelo	Dataset
Worker 1	lda	dataset1
Worker 2	glm	dataset1
Worker 3	glmnet	dataset1
Worker 4	svmRadial	dataset1

...

Worker	Modelo	Dataset
Worker 19	C5.0	dataset2
Worker 20	treebag	dataset2
Worker 21	rf	dataset2
Worker 22	gbm	dataset2

Estudo de caso: Model R

Model-R: Modelagem de nichos ecológicos



Outros frameworks (Não cobertos neste módulo)

- Big data
  - **SparkR** (R + Apache Spark), **Rhipe**
  - HadoopR
  - **bigmemory, ff**
  - Data management
    - DBMS: MySQL, PostgreSQL, RNeo4j
- GPU
  - **gpuR, gputools**
- Computação em grid: **multiR**
- Bibliotecas científicas como **RcpEigen, RcppArmadillo**

Referências, apostilas, cursos, ...

Apóstilas

- CRAN Task View: High-Performance and Parallel Computing with R: https://CRAN.R-project.org/view=HighPerformanceComputing
- Curso introdutório de R: http://statmath.wu.ac.at/~schwendinger/HPC/
- R em HPC: http://glennklockwood.com/data-intensive/r/on-hpc.html
- Benchmarking em R: https://www.alexejgossmann.com/benchmarking_r/
- Tutorial (Luxembourg University): https://ulhpc-tutorials.readthedocs.io/en/latest/maths/R/#hpc_pkg

Códigos

- https://github.com/eddelbuettel/ctv-hpc
- https://github.com/glennklockwood/paraR

Cursos online

Via coursera:

- R Programming (Johns Hopkins University): https://www.coursera.org/learn/r-programming
- Statistics with R (Duke University): https://www.coursera.org/specializations/statistics

Paneis de discussão

- https://www.r-bloggers.com/
- http://whyr.pl//foundation/2020/Bio_Panel/