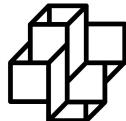


Escola Supercomputador



# O R EM AMBIENTE HPC



National Laboratory for Scientific Computing

L|N|C|C

Laboratório Nacional de Computação Científica - LNCC  
Professores: Raquel L. Costa ([quelopes@lncc.br](mailto:quelopes@lncc.br))  
Guilherme Gall ([gmgall@lncc.br](mailto:gmgall@lncc.br))

Dezembro de 2018 (edição extra UnB)

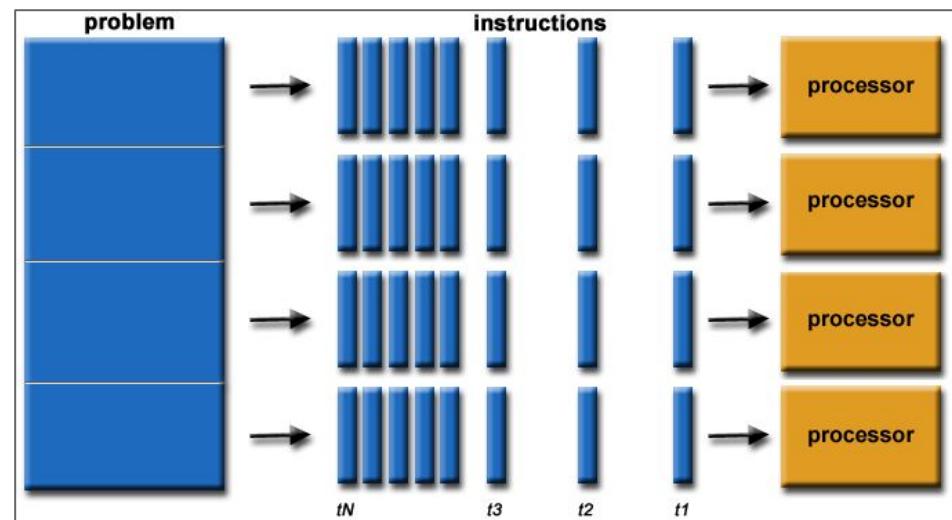
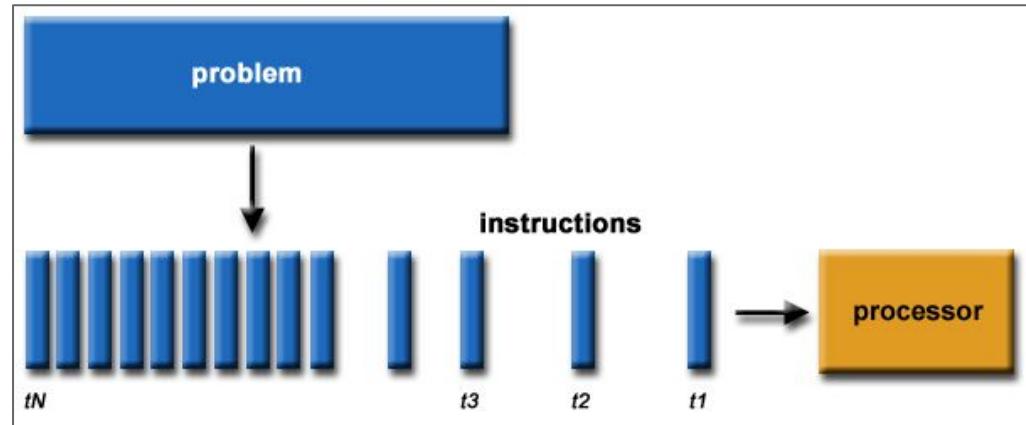
# SUMÁRIO

- Arquitetura, HPC, Supercomputadores e o Santos Dumont
- 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 família 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

- Clonar repositório do curso:
  - `git clone https://github.com/quelopes/R\_for\_HPC`
  - Quem gostou dá uma estrela 
- Instalar pacotes
  - Ver em:
- Horário do curso:
  - Início: 14:00
  - Final: 18:00 (no máximo 18:20)

# COMPUTAÇÃO PARALELA



Fonte: [https://computing.llnl.gov/tutorials/parallel\\_comp/](https://computing.llnl.gov/tutorials/parallel_comp/)

# INTRODUÇÃO À HPC

- "High Performance Computing (HPC) refere-se geralmente à prática de agregar poder computacional numa forma que entregue muito mais desempenho do que o obtível de um desktop ou workstation a fim de resolver problemas complexos em ciência, engenharia e negócios."
- Na maioria dos casos, um sistema de computação de alto desempenho é um cluster de nós de computação
- Problemas complexos são divididos em subunidades independentes executadas em paralelo

# INTRODUÇÃO À HPC

- Meramente agrupar várias máquinas numa rede não constitui um cluster
- É necessário um software de gerência de recursos para
  - Alocar acesso à recursos (nós de computação)
  - Consultar status de recursos
  - Prover um framework para iniciar, executar e monitorar tarefas no conjunto de recursos alocados
  - Arbitrar requisições conflitantes por recursos através da gerência de uma fila de tarefas pendentes

# SUPERCOMPUTADOR

# O QUE É UM SUPERCOMPUTADOR?

- Envolve um sistema que trabalha com o máximo desempenho potencial de qualquer computador, atualmente chegando a *Petaflops* ( $10^{15}$  operações aritméticas de ponto flutuante por segundo)
- Recursos de computação concentrados de múltiplos sistemas de computador trabalhando em paralelo
- Utilizado para processamento de problemas maciçamente complexos ou carregados de dados
- Exemplos de casos de uso incluem genômica, proteômica, meteorologia, astronomia e assim por diante.

# SUPERCOMPUTADORES

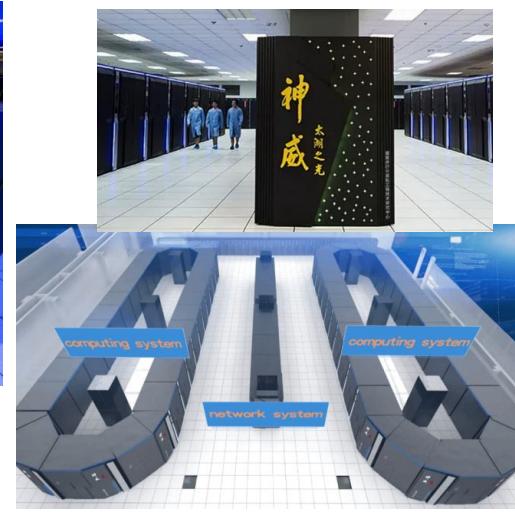
→ <https://www.top500.org/>



1º Summit, Oak Ridge National Laboratory, EUA.



2º Sierra, Lawrence Livermore National Laboratory (LLNL), EUA.



3º Sunway TaihuLight, Wuxi, China.



Supercomputador Santos Dumont, Laboratório Nacional de Computação Científica (LNCC), Petrópolis.

Computador	Número de CPU cores	Rmax (PFlop/s)
Santos Dumont	18.144	1
Summit	2.397.824	144
Sierra	1.572.480	95
Sunway Taihulight	10.649.600	93

# UM SUPERCOMPUTADOR É SEMPRE UM SUPERCOMPUTADOR?



Cray-1, considerado um supercomputador em 1975.



iPhone 6.

COMPUTADOR DESEMPENHO	
Nome	flops
megaflop	$10^6$
gigaflop	$10^9$
teraflop	$10^{12}$
petaflop	$10^{15}$
exaflop	$10^{18}$
zettaflop	$10^{21}$
yottaflop	$10^{24}$

- Cray-1: 160 Megaflops/s
- iPhone6: 7 Gigaflops/s

iPhone 6 ~20x mais  
rápido do que o  
Cray-1!!!

# O SUPERCOMPUTADOR SANTOS DUMONT

- 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 um total de 18.144 núcleos de CPU, distribuídos em 756 nós computacionais
- Três tipos de nós:
  - Thin nodes
  - Hybrid nodes
  - Fat-node



Supercomputador Santos Dumont, LNCC, Petrópolis.



Laboratório Nacional de Computação Científica (LNCC), Petrópolis.

# O SUPERCOMPUTADOR SANTOS DUMONT

- Thin nodes:
  - 504 nós totalizando 12.096 núcleos de processamento
  - 64 GB de memória RAM por nó
- Hybrid nodes:
  - 252 nós totalizando 6.048 núcleos de processamento
  - 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

# CURSOS - EDIÇÃO VERÃO E INVERNO

Laboratório Nacional de Computação Científica



MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES

## Destaques

Projeto Rede Avançada em Biologia Computacional

Programa de Verão 2019



Laboratório  
Nacional de  
Computação  
Científica

O LNCC

COORDENAÇÕES

PESQUISA E  
DESENVOLVIMENTO

SERVIÇOS  
COMPUTACIONAIS DE  
ALTO DESEMPENHO

## EVENTOS - PROGRAMAÇÃO

### Programa de Verão 2019

#### Jornadas / Encontros / Workshops

Jornada em e-Biodiversidade - 25/2/2019 a 1/3/2019

V Encontro em Modelagem Matemática do Crescimento Tumoral - 20/2/2019 a 22/2/2019

Jornada em Ciência de Dados - 18/2/2019 a 22/2/2019

XII Encontro Acadêmico de Modelagem Computacional do LNCC - 4/2/2019 a 7/2/2019

Jornada de Iniciação Científica e Tecnológica - 8/2/2019

Workshop on Computational Modeling and Numerical Analysis - WCCMNA 2019 - 25/2/2019 a 28/2/2019

Jornada de Verão de Bioinformática do LNCC - 25/2/2019 a 1/3/2019

Escola Supercomputador SDumont - 18/2/2019 a 22/2/2019

## Escola Supercomputador



## Escola de Inverno Supercomputador



Principal

Programação

Minicursos

Horário Mini-Cursos

Eventos

Organização

Impressão Boleto Bancário

Hotéis/Pousadas

# O SUPERCOMPUTADOR SD - SUBMISSÃO DE PROPOSTAS

The screenshot shows the homepage of the SDumont website. At the top, there's a navigation bar with links like 'Secure | https://sdumont.lncc.br', 'Apps', 'Academics', 'English', 'Services', 'Participate', 'Information access', 'Legislation', 'Information channels', and 'Other bookmarks'. Below this is a header with the text 'BRAZIL' and 'MCTI | LNCC'. The main title 'SDumont' is prominently displayed, followed by the subtitle 'Sistema de Computação Petaflópica do SINAPAD'. To the right is the SINAPAD logo, which features a stylized blue hexagon with a central gear-like pattern. Below the title, there are several menu items: 'Chamada ABERTA!', 'A Máquina', 'Programas de Alocação', 'Cursos & Suporte', 'Projetos', and 'Comitês'. A large banner below the menu reads 'Projetos STANDARD e EDUCACIONAL agora em FLUXO' and 'CONTÍNUO!'. The banner also includes a quote from mobull: 'Não se expande com a altura da vila. Quando mais alto, mais longe de projeto. Quando mais baixo, mais tempo para desenvolver uma peça. É quando se tem pressa de sólido que se deve desenhar.' On the left side of the banner, there's a photograph of a server room with green gas cylinders and white server racks. On the right side, there's a photograph of a Bull supercomputer system.

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

```
quelopes@quelopes-Inspiron-5458: ~/Downloads/JabotMSTR
main.tex:193 Encoding '/tmp/Consulta_10_-_Bioma_e_vegetacao_por_agregacao.png'

quelopes@quelopes-Inspiron-5458:~/Downloads/JabotMSTR$ R
  
R version 3.4.2 (2017-09-28) -- "Short Summer"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (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

Editor de código

The screenshot shows the RStudio interface with several panes:

- Editor de código (Code Editor):** Contains an R script named `Parameters.R` with code related to gene expression analysis.
- Console:** Displays the output of the R script, including package imports and the definition of variables like `nameExp`, `orgAnnot`, and `GPL`.
- Workspace:** Shows the global environment with objects like `dirR_raw`, `dirRk`, `GPL`, `infoExp`, `method`, `nameExp`, `orgAnnot`, `overallDesign`, `platAccess`, and `platname`.
- Files:** Shows the project structure with files like `Annotation`, `Dockerfile`, `GennetShiny`, `GSE62232`, `import.cql`, `Module-A`, `Parameters.R`, `README.md`, and `shiny-server.sh`.

Console

Gráficos e arquivos

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

The screenshot shows the CRAN homepage with a sidebar on the left containing links for GRAN, Mirrors, What's new?, Task Views, Search, About R, R Homepage, The R Journal, Software, R Sources, R Binaries, Packages, Other, Documentation, Manuals, FAQs, and Contributed. The main content area is titled "The Comprehensive R Archive Network" and includes sections for "Download and Install R", "Source Code for all Platforms", "Questions About R", "What are R and CRAN?", "Submitting to CRAN", and "Note".

Página inicial do CRAN.

The screenshot shows the Bioconductor homepage with a navigation bar at the top for Home, Install, Help, Developers, and About. The main content area features a section for the "International Workshop on Bioinformatics" with details about the location (Port of Spain, Trinidad), dates (2011-01-19 ~ 2011-01-21), instructors (Martin Morgan, Vincent Carey, others), a description of the 3-day course, materials (presentations and lab exercises), and a code block for installing the IW2011 package. A sidebar on the right lists packages, documentation, and other resources.

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:

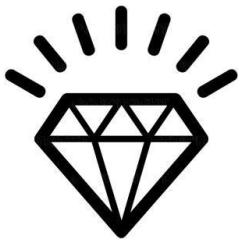
```
devtools::install_github("hadley/devtools")
```

## 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



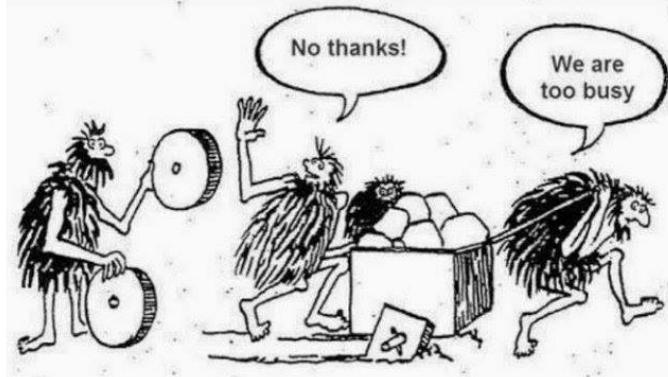
- As funções em R são sempre acompanhadas por parênteses ()
- Verifique seu diretório atual e se necessário altere-o
  - ◆ `getwd()`
  - ◆ `setwd('/home/aluno/Dados')`
- Em caso de dúvida `help('sqrt')` ou `?sqrt` ou `help.search('sqrt')`
- O buscador <http://www.rseek.org/> restringe a busca para os 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 **embarrasosamente 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 automatically installed (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 task view pages.
- For general concerns regarding task views contact the [ctv](#) package maintainer.

**Topics**

<a href="#">Bayesian</a>	Bayesian Inference
<a href="#">ChemPhys</a>	Chemometrics and Computational Physics
<a href="#">ClinicalTrials</a>	Clinical Trial Design, Monitoring, and Analysis
<a href="#">Cluster</a>	Cluster Analysis & Finite Mixture Models
<a href="#">DifferentialEquations</a>	Differential Equations
<a href="#">Distributions</a>	Probability Distributions
<a href="#">Econometrics</a>	Econometrics
<a href="#">Environmetrics</a>	Analysis of Ecological and Environmental Data
<a href="#">ExperimentalDesign</a>	Design of Experiments (DoE) & Analysis of Experimental Data
<a href="#">ExtremeValue</a>	Extreme Value Analysis
<a href="#">Finance</a>	Empirical Finance
<a href="#">FunctionalData</a>	Functional Data Analysis
<a href="#">Genetics</a>	Statistical Genetics
<a href="#">Graphics</a>	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
<a href="#">HighPerformanceComputing</a>	High-Performance and Parallel Computing with R
<a href="#">MachineLearning</a>	Machine Learning & Statistical Learning
<a href="#">MedicalImaging</a>	Medical Image Analysis
<a href="#">MetaAnalysis</a>	Meta-Analysis
<a href="#">MissingData</a>	Missing Data
<a href="#">ModelDeployment</a>	Model Deployment with R
<a href="#">Multivariate</a>	Multivariate Statistics
<a href="#">NaturalLanguageProcessing</a>	Natural Language Processing
<a href="#">NumericalMathematics</a>	Numerical Mathematics
<a href="#">OfficialStatistics</a>	Official Statistics & Survey Methodology
<a href="#">Optimization</a>	Optimization and Mathematical Programming
<a href="#">Pharmacokinetics</a>	Analysis of Pharmacokinetic Data
<a href="#">Phylogenetics</a>	Phylogenetics, Especially Comparative Methods
<a href="#">Psychometrics</a>	Psychometric Models and Methods
<a href="#">ReproducibleResearch</a>	Reproducible Research
<a href="#">Robust</a>	Robust Statistical Methods
<a href="#">SocialSciences</a>	Statistics for the Social Sciences
<a href="#">Spatial</a>	Analysis of Spatial Data
<a href="#">SpatioTemporal</a>	Handling and Analyzing Spatio-Temporal Data
<a href="#">Survival</a>	Survival Analysis
<a href="#">TimeSeries</a>	Time Series Analysis
<a href="#">WebTechnologies</a>	Web Technologies and Services
<a href="#">gR</a>	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](mailto:Dirk.Eddelbuettel@R-project.org)

**Version:** 2018-02-07

**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, and Tim Flutre (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 `cvt` 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

**Escola Supercomputador**



UTILIZANDO O R NO SANTOS DUMONT

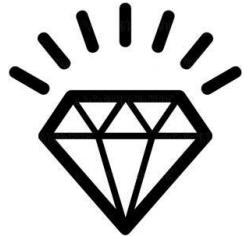
# CARREGAR O MÓDULO R NO SANTOS DUMONT

```
$ source /scratch/app/modulos/intel-psxe-2016.sh (PSXE 2016) ou  
$ source /scratch/app/modulos/intel-psxe-2017.sh (PSXE 2017)  
$ module load openmpi/1.10_intel  
$ module load R/3.3.1_intel  
$ module load R/3.4.3_gnu
```

É com esses módulos 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).

# 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'`

# 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)
```

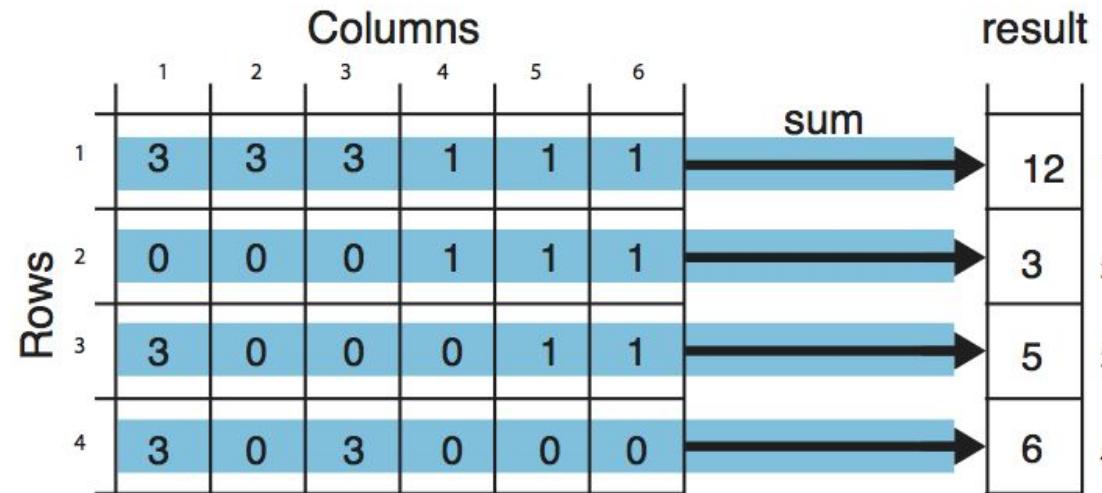
```
> m  
      [,1] [,2] [,3] [,4]  
[1,]    1    5    9   13  
[2,]    2    6   10   14  
[3,]    3    7   11   15  
[4,]    4    8   12   16
```

# 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
```

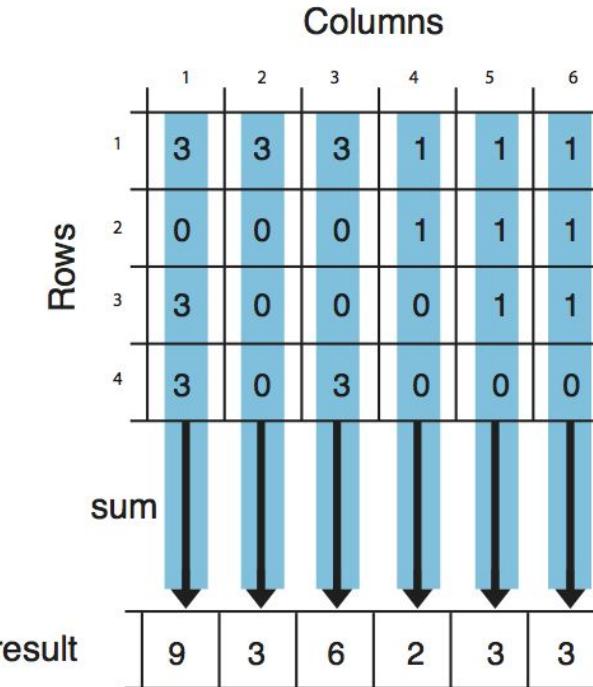
# FUNÇÃO APPLY

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



# FUNÇÃO APPLY

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



# 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
```

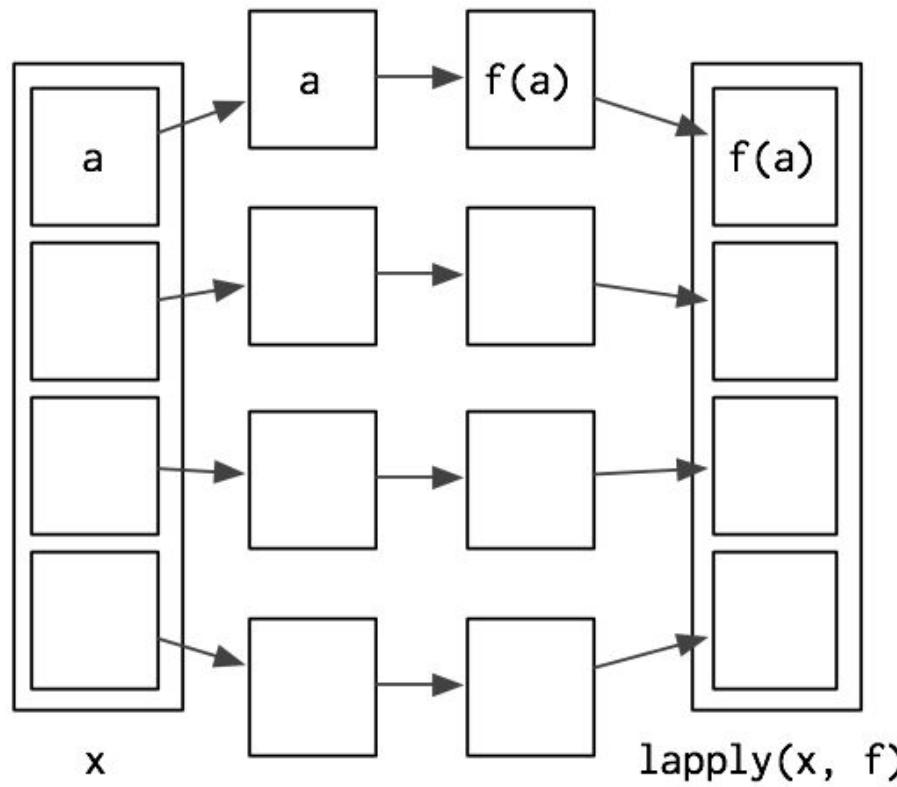
```
$b
```

```
[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 SPLY

- 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 S APPLY

```
> # 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 SAPPY

**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 SAPPY

- 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

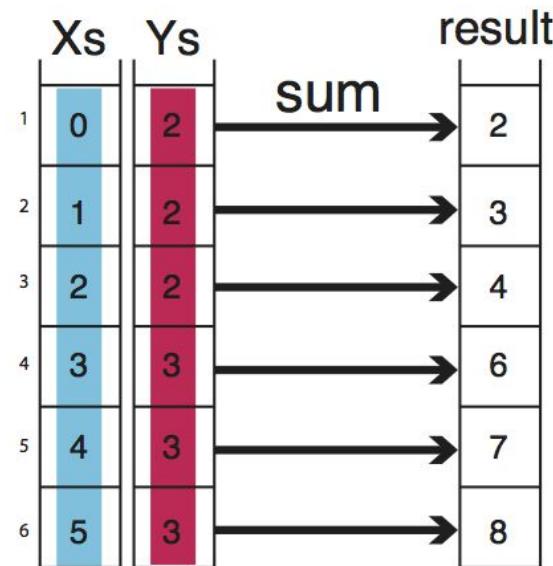
[[2]]
[1] 2 2 2

[[3]]
[1] 3 3

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

# FUNÇÃO MAPPLY

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



# 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**.
- Fornece substitutos para a maioria das funções desses pacotes.

# PACOTE PARALLEL

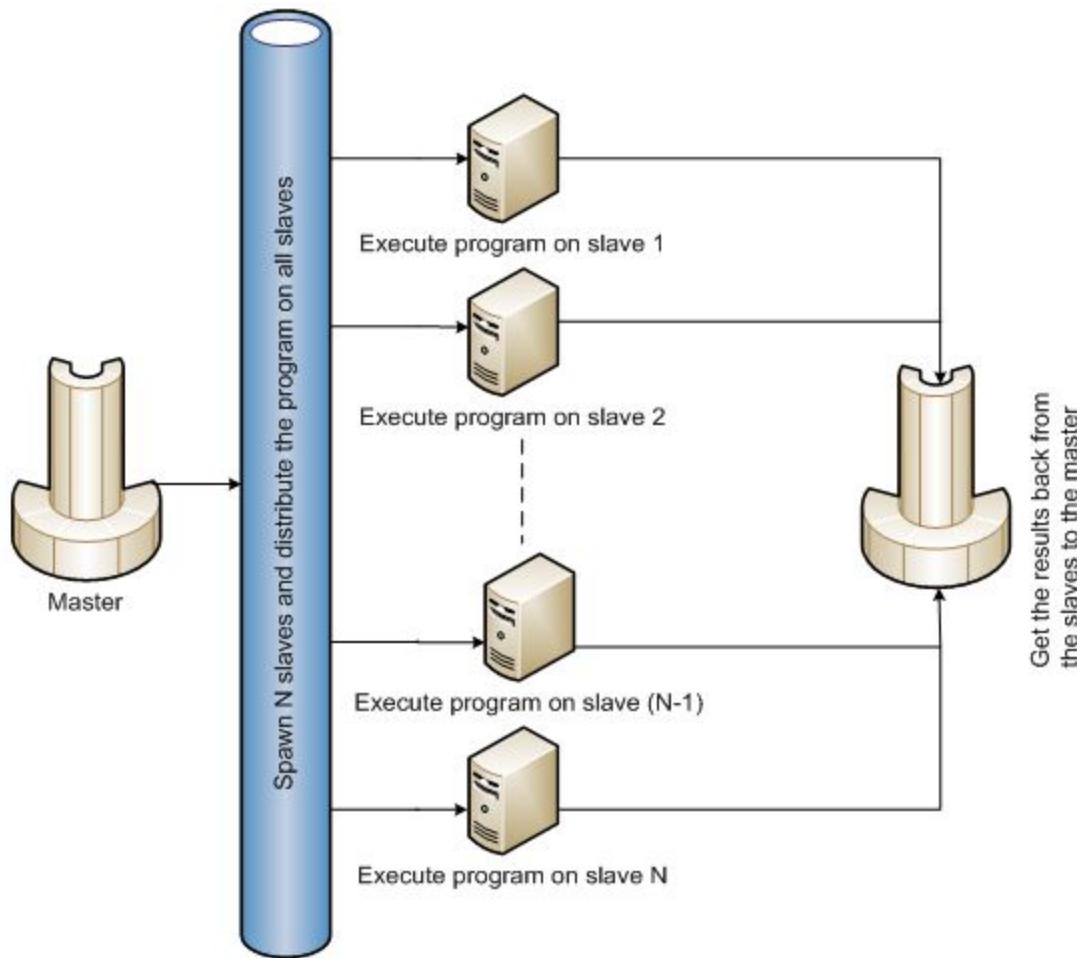
O processamento com o pacote parallel segue o seguinte modelo:

- 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



# 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 1º 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.
- 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 PAR\\$APPLY

- 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 PAR\\$APPLY

- 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 parâmetro 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 CLUSTER EVALQ

- `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 CLUSTER`EVALQ` E CLUSTER`CALL`

- 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

Retorna um valor,  
ao contrário do  
for tradicional.

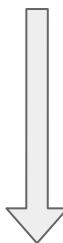
Por padrão,  
retorna uma  
lista

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

```
[[1]]  
[1] 1  
  
[[2]]  
[1] 1.414214  
  
[[3]]  
[1] 1.732051
```

# 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...

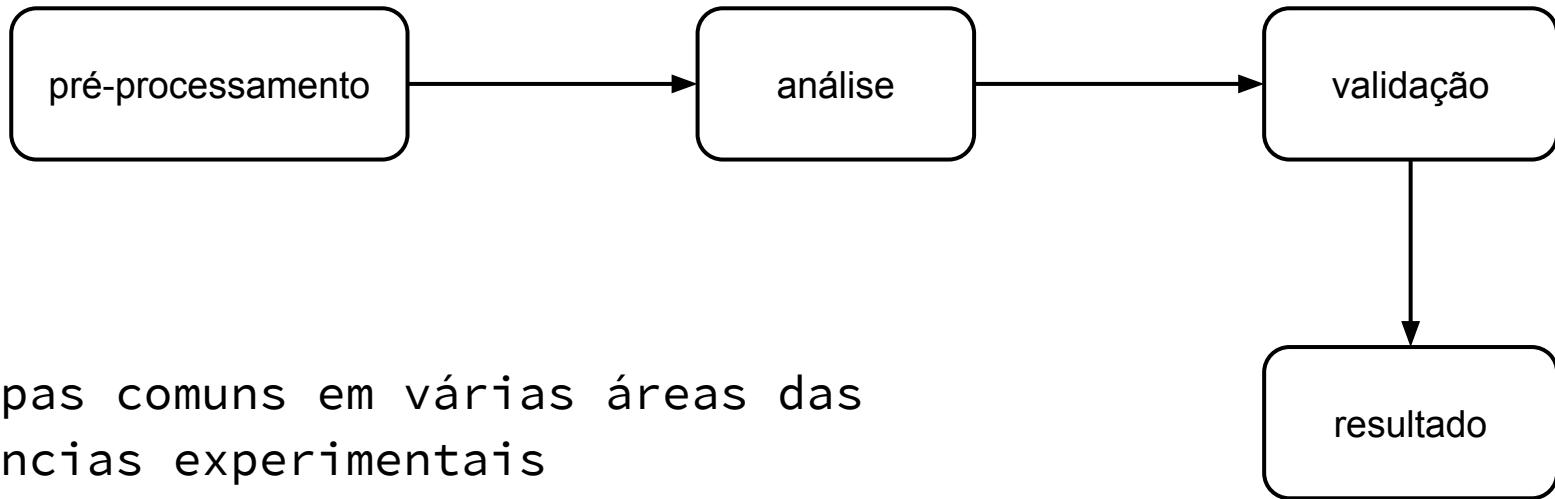
```
out <- foreach(x=a, y=b, z=c) %dopar% {  
  (x + y)**z  
}
```

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 utilizam esse 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)

```
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
```

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**

```
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
##  n eval cld
##    100  c
##    100  b
##    100  a
```

```
library(microbenchmark)

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

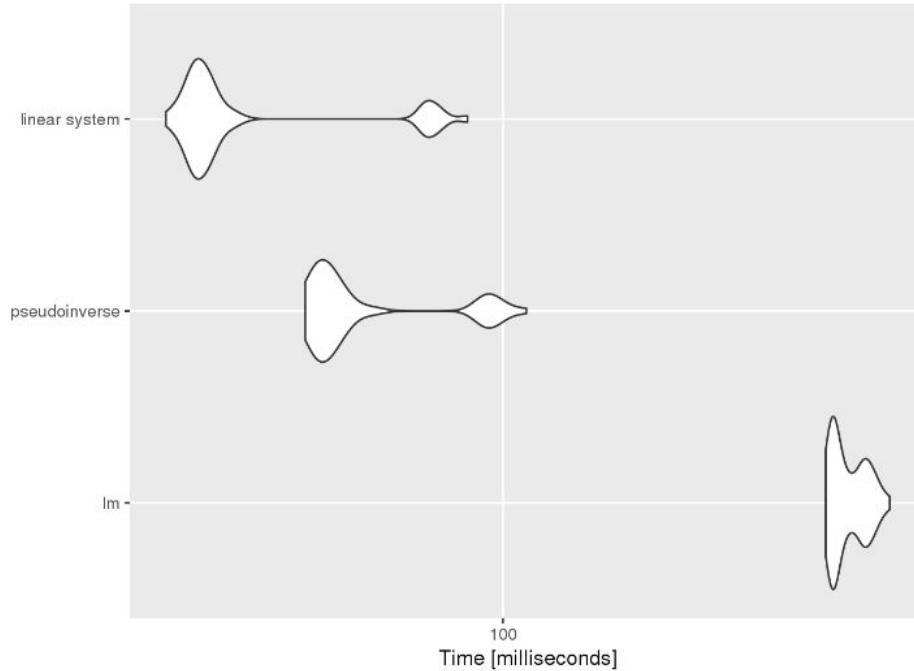
check_for_equal_coefs <- function(values){
  tol <- 1e-12
  max_error <- max(c(abs(values[[1]] - values[[2]]),
                      abs(values[[2]] - values[[3]]),
                      abs(values[[1]] - values[[3]])))
  max_error < tol
}

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

```
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

```
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

```
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

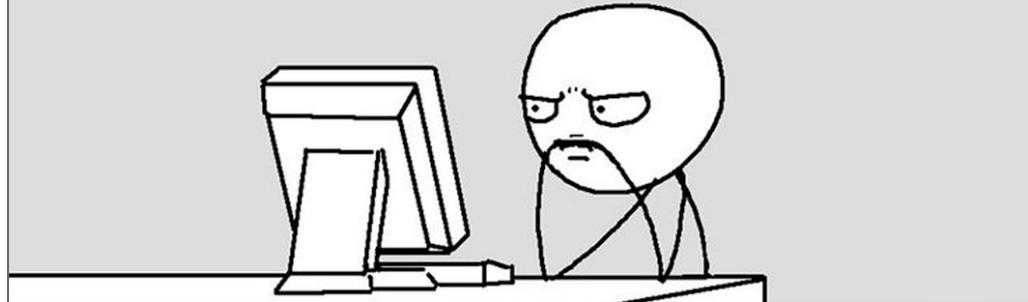
```
system.time({
  output <- ifelse ((df$col1 + df$col2 + df$col3 + df$col4) > 4, "greater_than_4", "lesser_than_4")
  df$output <- output
})

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

MELHORANDO O CÓDIGO R,  
BOAS PRATICAS

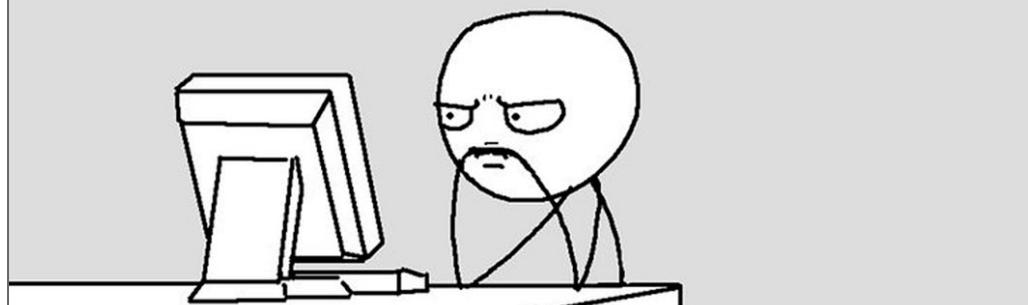
**EU CODIFICANDO...**

estou cansado,  
vou continuar amanhã...



**NO DIA SEGUINTE**

Que merda é essa que eu fiz?



**DEVMEDIA**

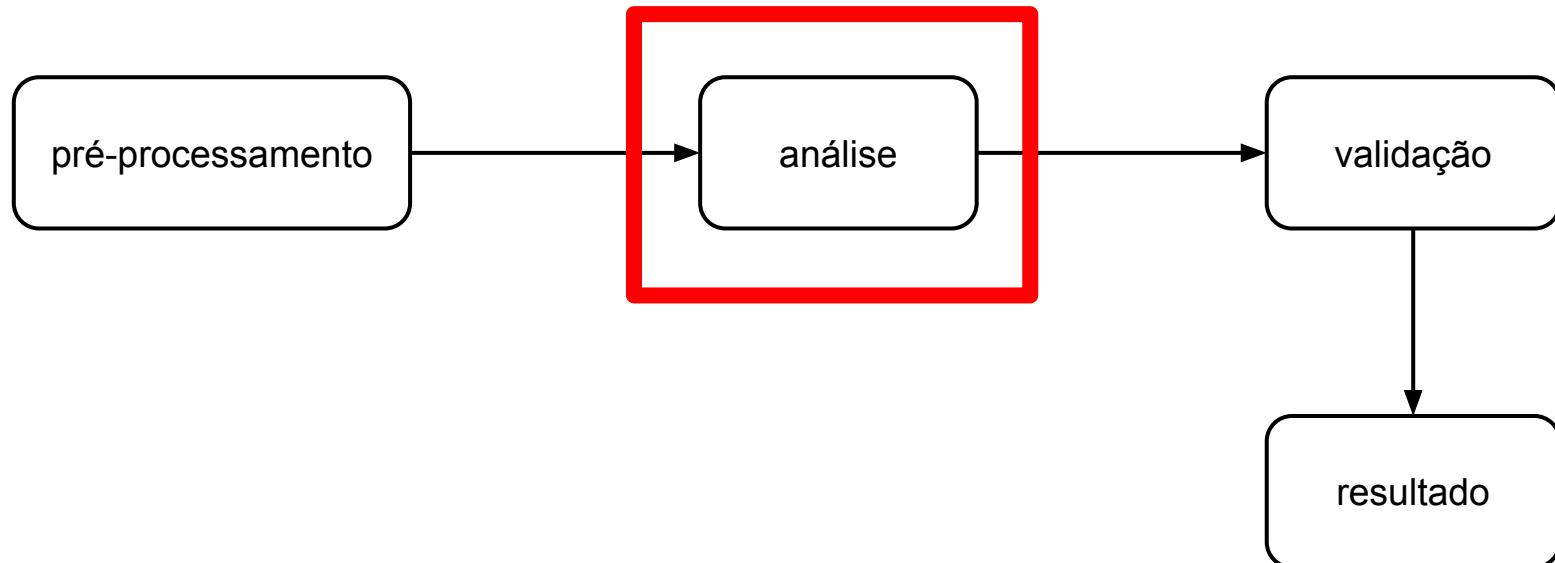
# BOAS PRÁTICAS NA ESCRITA DO CÓDIGO

- 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`

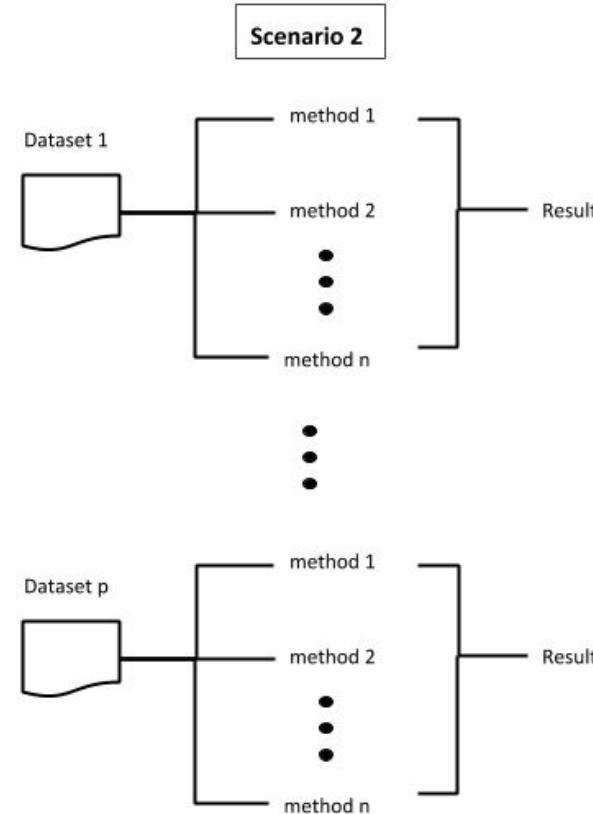
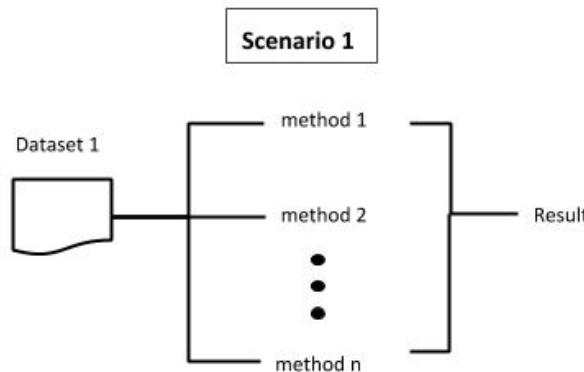
# BOAS PRÁTICAS - OUTRAS DICAS

- 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

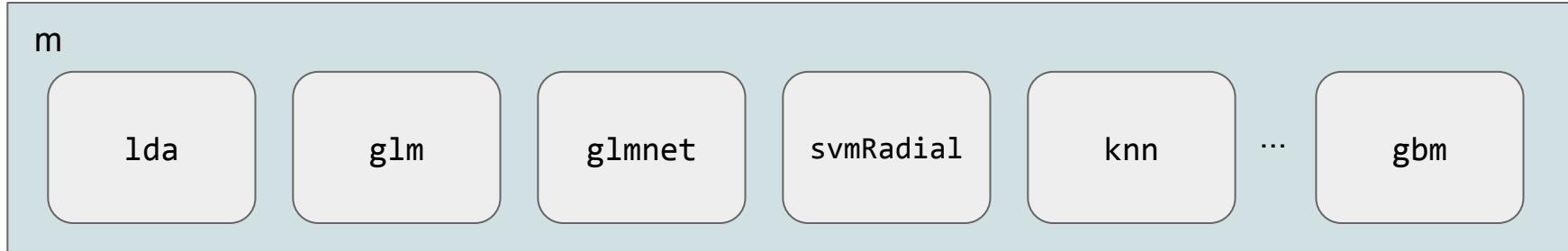


# EXEMPLO DE MÉTODOS EM MACHINE LEARNING

- 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')
```



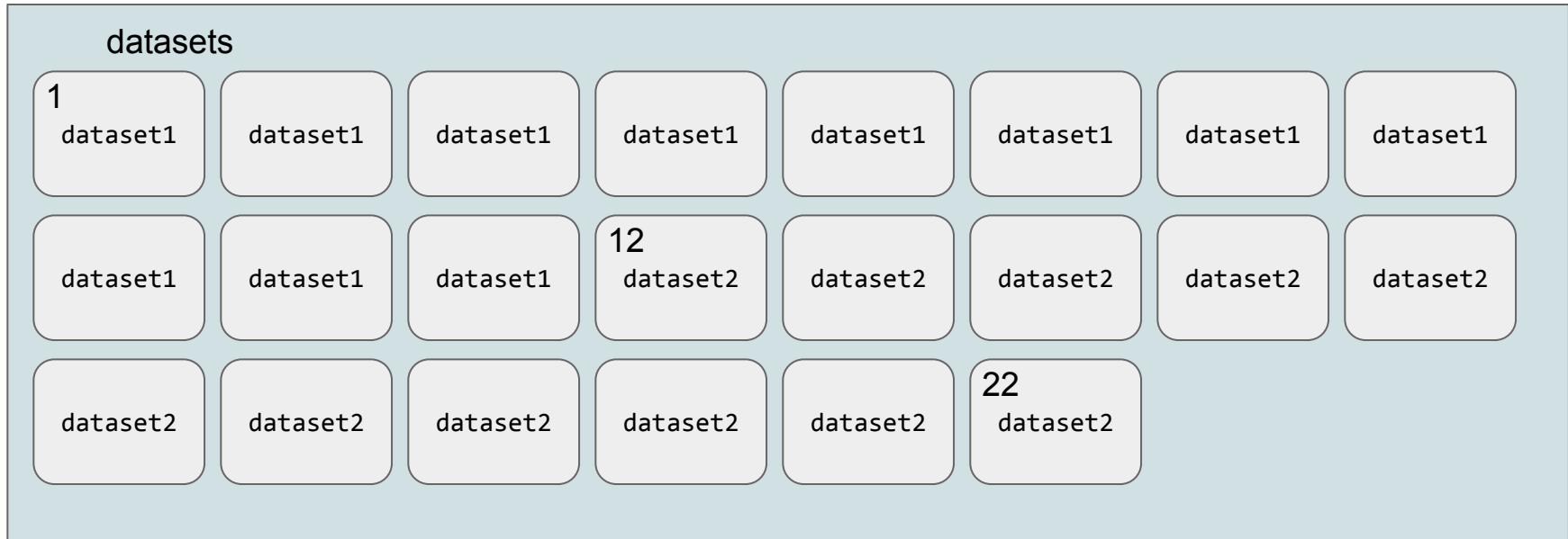
# CENÁRIO 2

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

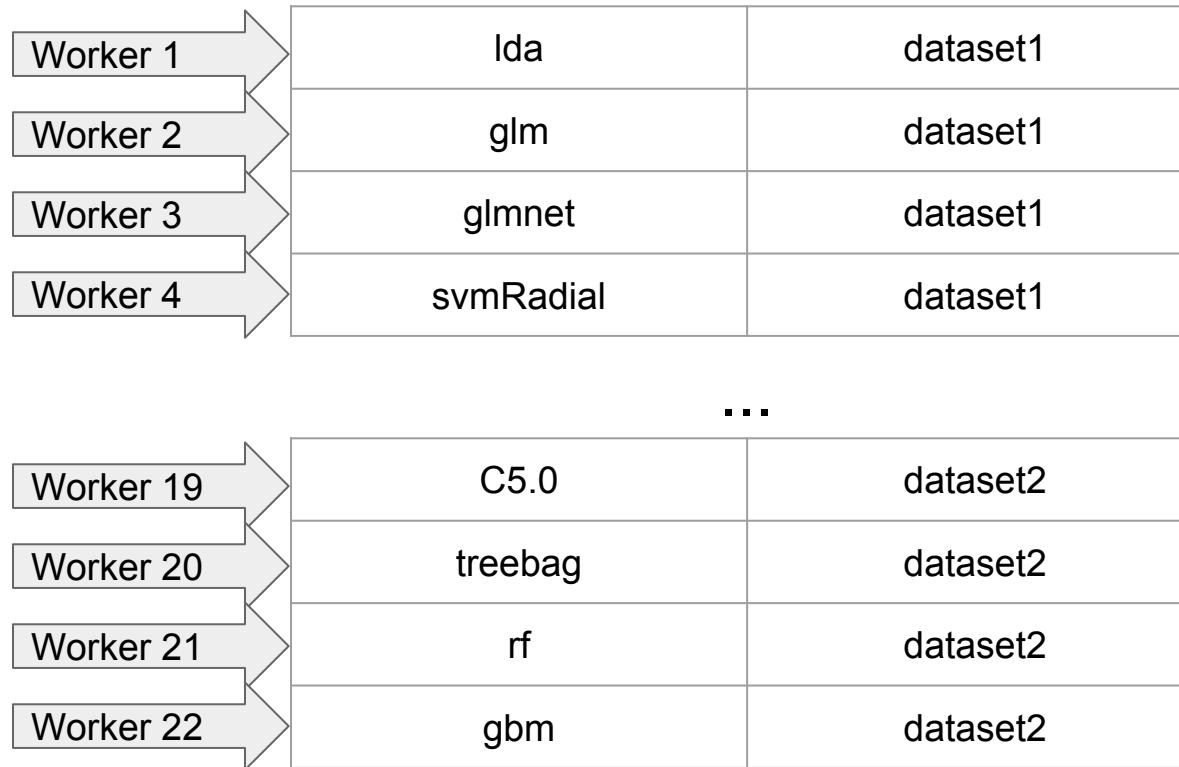
methods	lda	glm	glmnet	svmRadial	knn	nb	rpart	C5.0
1	lda	glm	glmnet	svmRadial	knn	nb	rpart	C5.0
12	treebag	rf	gbm	lda	glm	glmnet	svmRadial	knn
22	nb	rpart	C5.0	treebag	rf	gbm		

# CENÁRIO 2

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

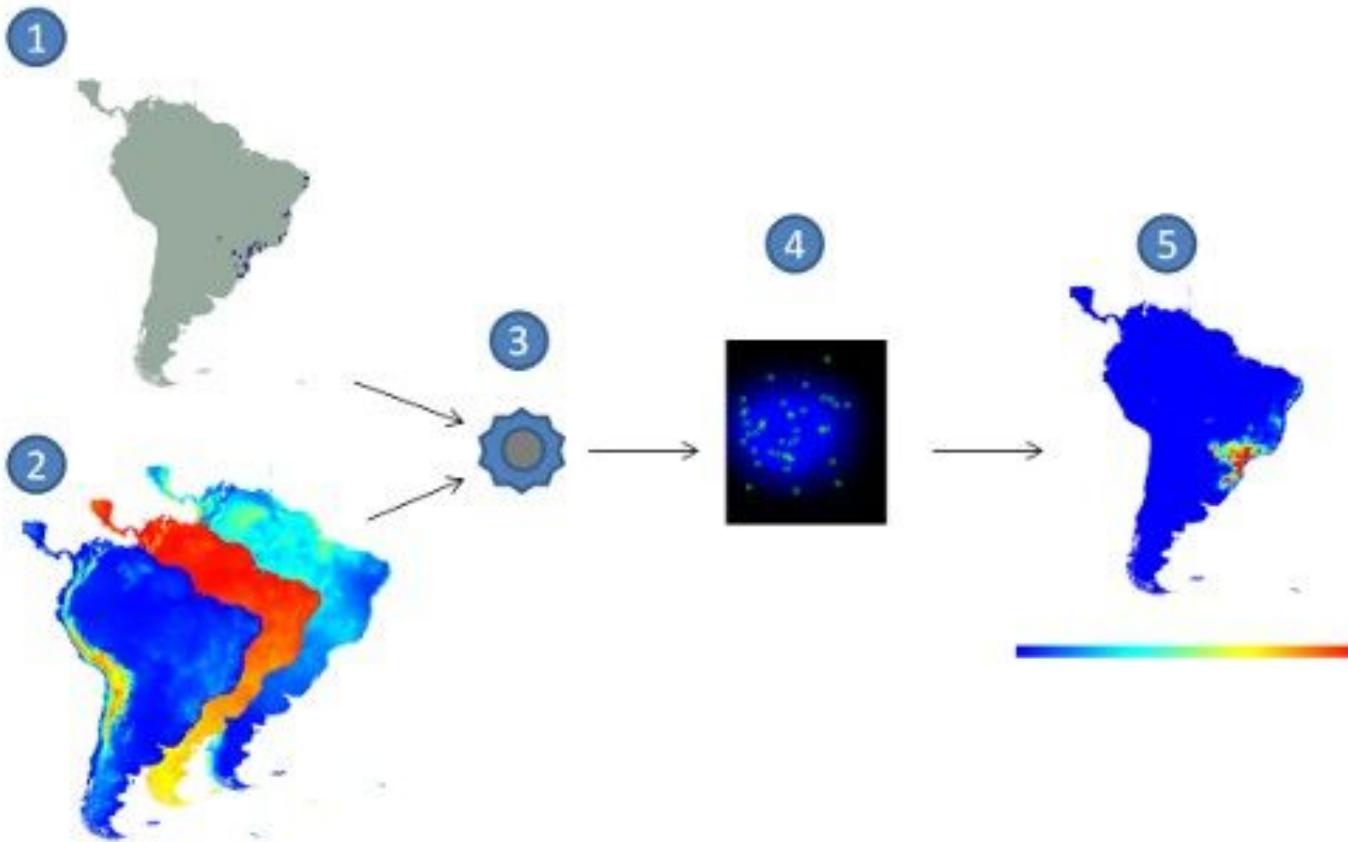


# CENÁRIO 2



# 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 **RcppEigen**, **RcppArmadillo**

# REFERÊNCIAS, APOSTILAS, CURSOS, ...

## Apostilas

- **Introdução ao R nas Ciências da Vida:** <https://quelopes.github.io>
- **CRAN Task View: High-Performance and Parallel Computing with R:** <https://cran.r-project.org/web/views/HighPerformanceComputing.html>
- **Curso introdutório de R:** <http://statmath.wu.ac.at/~schwendinger/HPC/>
- **R em HPC:** <http://www.glennclockwood.com/data-intensive/r/on-hpc.html>
- **Benchmarking in R:** [http://www.alexejgossmann.com/benchmarking\\_r/](http://www.alexejgossmann.com/benchmarking_r/)

## Códigos

- <https://github.com/eddelbuettel/ctv-hpc>
- <https://github.com/glennclockwood/paraR>

## Cursos online

Via coursera:

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

## Referências bibliográficas

- State of the Art in Parallel Computing with R. *Journal of Statistical Software*. August 2009, Vol. 31(1).
- A Survey of R Software for Parallel Computing. *American Journal of Applied Mathematics and Statistics*. 2014, Vol. 2(4), pg. 224-230.
- Parallel R. (Book). Q. Ethan McCallum e Stephen Weston. 2012. O'reilly.

# FIM! OBRIGADO

Código: [https://github.com/quelopes/R\\_for\\_HPC](https://github.com/quelopes/R_for_HPC)

Apresentação: [https://quelopes.github.io/files/courses/RforHPC\\_Lncc2018/](https://quelopes.github.io/files/courses/RforHPC_Lncc2018/)

Guilherme Gall ([gmgall@lncc.br](mailto:gmgall@lncc.br))  
Raquel L. Costa ([quelopes@gmail.com](mailto:quelopes@gmail.com))