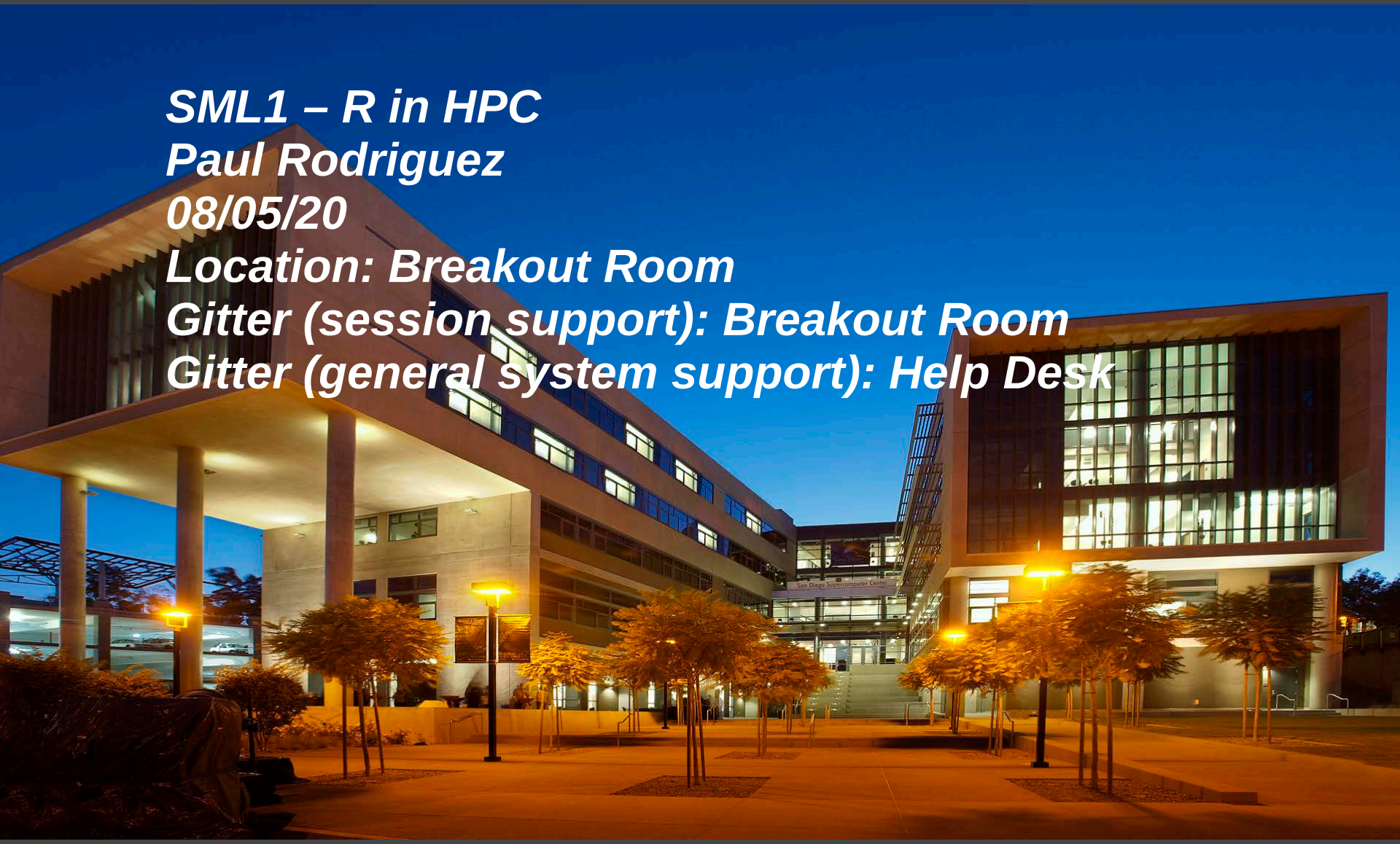
Scalable Machine Learning Agenda

8:00 - 9:15: R in HPC
9:15 - 9:30: Break
9:30 - 10:00: ML with Spark
10:00 - 10:20: PySpark Hands-on
10:20 - 10:40: SparkR Hands-on
10:40 - 10:45: Wrap-up

SDSC Summer Institute 2020



SML1 – R in HPC
Paul Rodriguez
08/05/20
Location: Breakout Room
Gitter (session support): Breakout Room
Gitter (general system support): Help Desk

R, Scaling R, Parallel R

- A Glimpse of R (recap)
- R and Scaling
- Parallel options for R
- R on Comet exercise

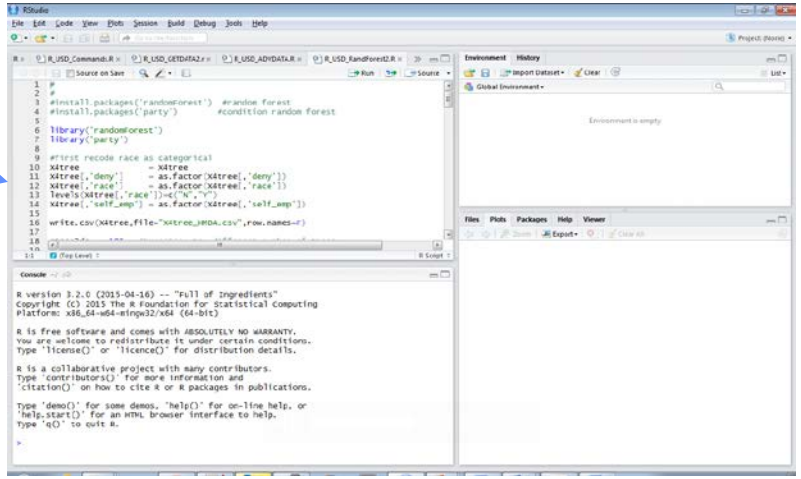
A typical R development workflow

- **R studio: An Integrated development environment for R on your local machine – good for development**

Menu tab

Edit window to Build scripts

R console



Environment Information on variables and command history

Plots, help docs, package lists

R commands in brief

- **A typical R code workflow:**

```
#READ DATA (housing mortage cases)
X                    =read.csv('hmda_aer.csv',header=T,stringsAsFactors=T)
#SUBSET DATA
indices_2keep =which(X[,'s13'] %in% c(3,4,5)))
X                    =X[unique(indices_2keep),]
#CREATE/TRANSFORM VARIABLES
pi_rat               = as.numeric(X[,'s46']/100)          #debt2income ratio
race                 = as.numeric(X[,'s13'] %in% c(3,4)) #make race values 1-4 into values 0 or 1
deny                 = as.numeric(X[,'s7']==3)            #make deny values into 0 or 1,
                                                                1 only for deny='3'
#RUN MODEL and SHOW RESULTS
lm_result            =lm(deny~race+pi_rat)          #lm is 'linearmodel'
summary(lm_result)
```

R strengths for HPC

- **Data Wrangling**

R strengths for HPC

- **Data Wrangling**
- **Sampling/bootstrap methods**

R strengths for HPC

- **Data Wrangling**
- **Sampling/bootstrap methods**
- **Particular Statistical procedures that you won't find implemented anywhere else, e.g.**

  Multiple Imputation methods,

  Instrument Variable (2 stage) Regression

  Matching subjects for pairwise analysis

  MCMC routines

Scaling, practically

- **Scaling (with or without more data):**
  - more complex analysis (ie optimizations)
  - more sampling (ie more trees in Random Forest)

Scaling, practically

- **Scaling (with or without more data):**
  - more complex analysis (ie optimizations)
  - more sampling (ie more trees in Random Forest)

- **Sometimes easy to parallelize (like with sampling)**

Scaling, practically

- **Scaling (with or without more data):**
  - more complex analysis (ie optimizations)
  - more sampling (ie more trees in Random Forest)

- **Sometimes easy to parallelize (like with sampling)**

- **Sometimes too much communication between parts (matrix inversion)**

R Scaling In a nutshell

- R takes advantage of math libraries for vector operations

R Scaling In a nutshell

- R takes advantage of math libraries for vector operations
- R packages provide multicore, multimode, or distributed data (SparkR) options

R Scaling In a nutshell

- **R takes advantage of math libraries for vector operations**
- **R packages provide multicore, multimode, or distributed data (SparkR) options**
- **However, model implementations not necessarily built to use parallel backends**
  - Some models more amenable to parallel versions

Consider Regression Computations

- **Linear Model:** $Y = X * B$

  where Y=outcomes , X=data matrix

Consider Regression Computations

- **Linear Model:** $Y = X * B$

  where Y=outcomes , X=data matrix

- **Algebraically, we could**:
  - take "inverse" $of\ X * Y = B$ (time consuming)
  - use derivatives to search for solutions (very general)

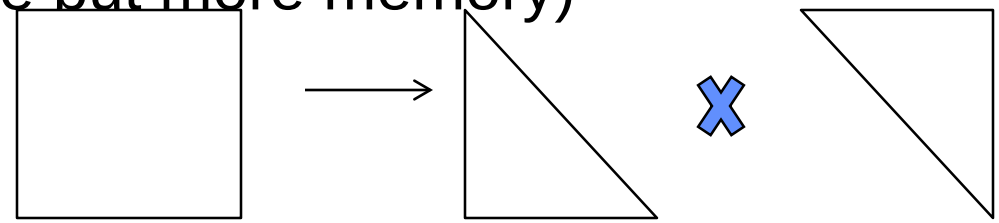
Consider Regression Computations

- **Linear Model:** $Y = X * B$

  where Y=outcomes , X=data matrix

- **Algebraically, we could**:
  - take "inverse" $of\ X * Y = B$ (time consuming)
  - use derivatives to search for solutions (very general)

- **Or, better:**
  - QR decomposition of *X* into triangular matrices (easier to solve but more memory)



Consider Regression models in R

- **Related Models and Functions :**

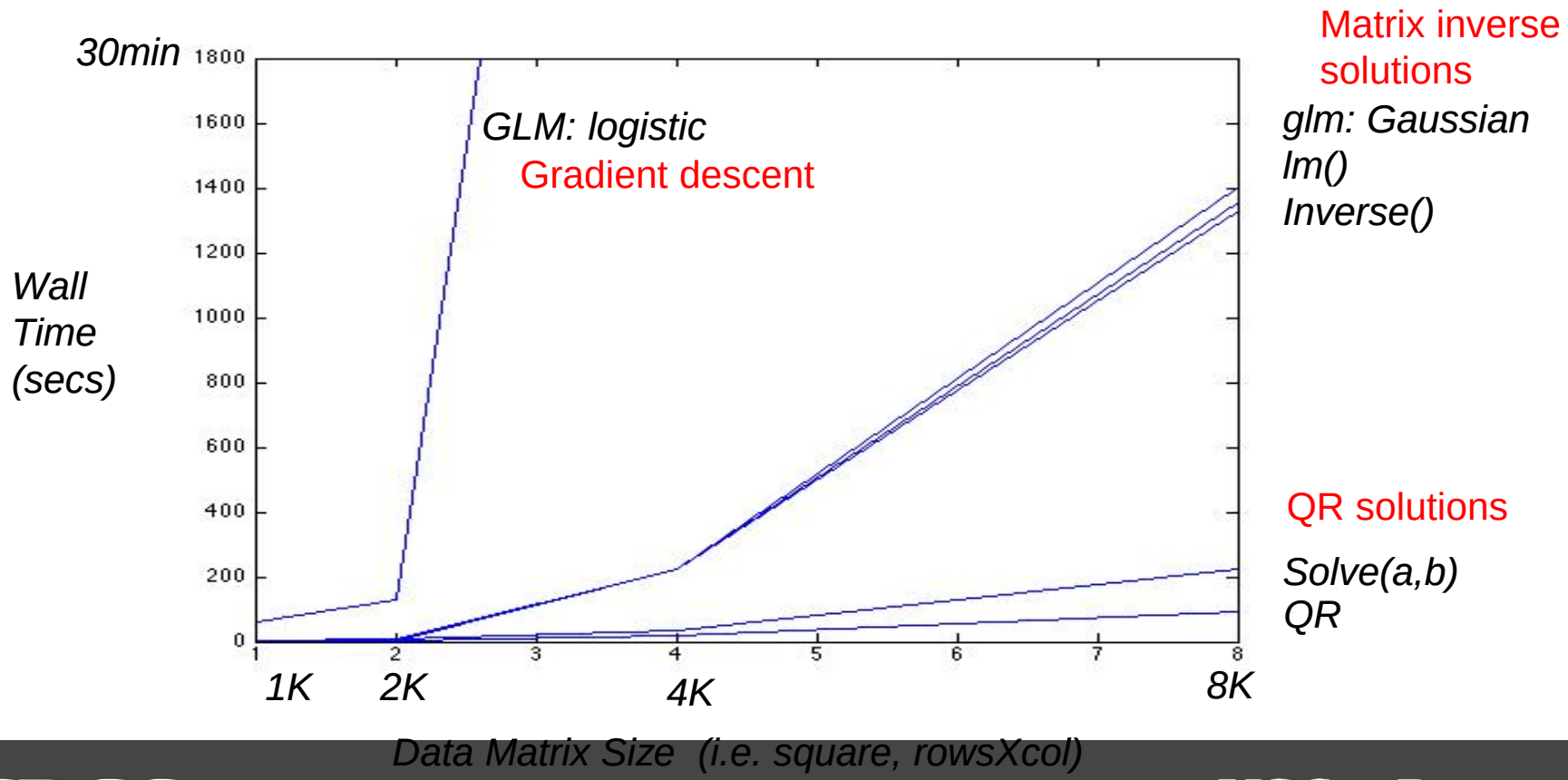
  lm() #Linear Model

  glm() #Generalized Linear Model

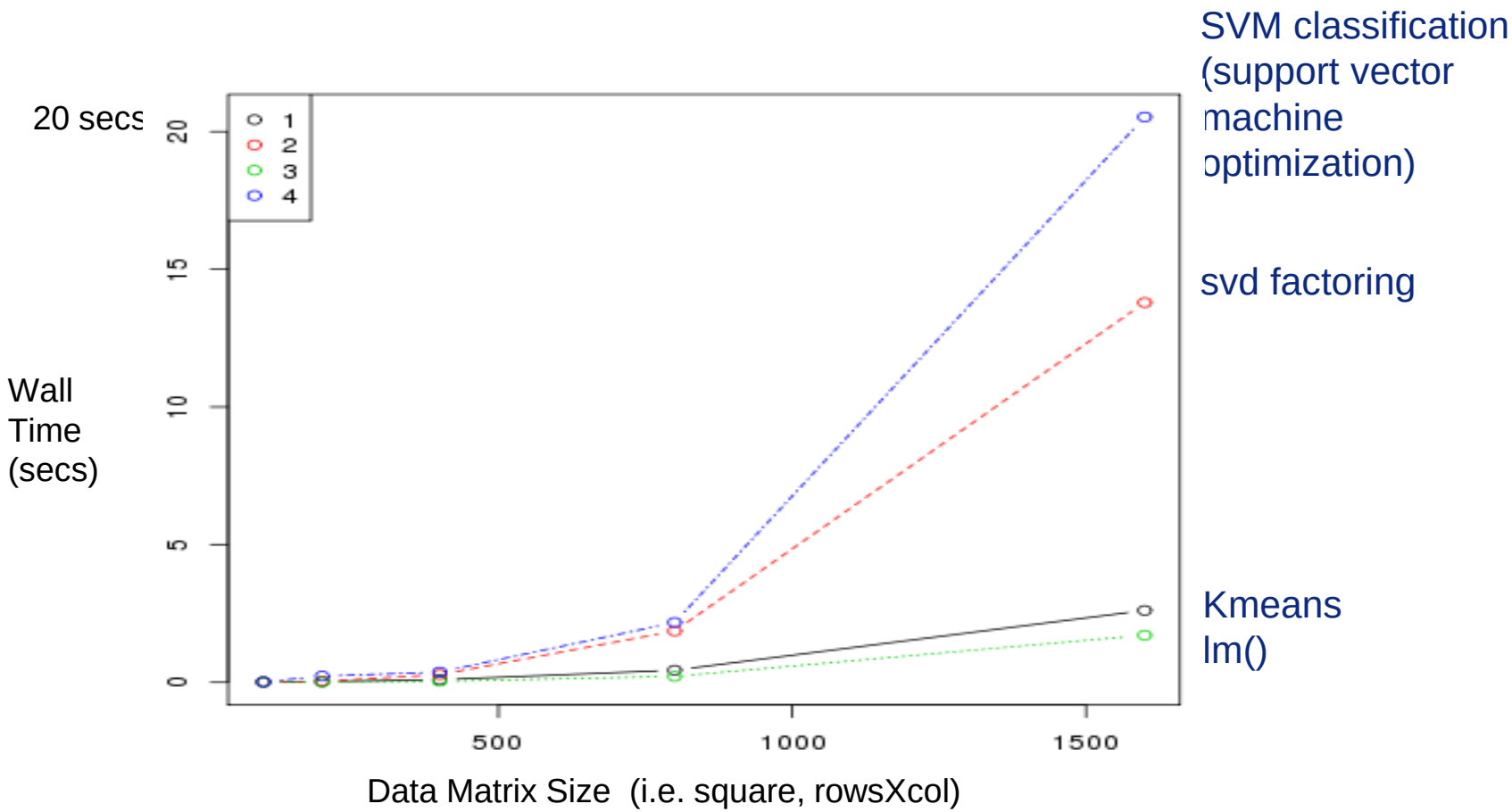
  (logistic regression, etc)

Solving Linear Systems Performance with R, 1 compute node

R:
glm(Y~X,family=gaussian) #gaussn regrssn (like lm)
glm(Y~X,family=binomial) # logistic regrssn (Y=0 or 1)



Machine learning models: Performance on 1 compute node



R multicore processing

- ‘doParallel’ package – provides the back end to the ‘for each’ parallel processing command

R multicore processing

- ‘doParallel’ package – provides the back end to the ‘for each’ parallel processing command
- uses threads across cpu cores to pass data & commands

R multicore processing

- ‘doParallel’ package – provides the back end to the ‘for each’ parallel processing command
- uses threads across cpu cores to pass data & commands
- Updates and combines the previous ‘snow’ and ‘multicore’ packages, so that is also works for multinode (and it’s similar to doMPI, both run on top of RMPI) .

See https://cran.r-project.org/web/packages/doParallel/vignettes/gettingstartedParallel.pdf

R multicore coding

```
install.packages(doParallel)
library(doParallel)
registerDoParallel(cores=24)
```

1. allocate workers

R multicore coding

```
install.packages(doParallel)
library(doParallel)
registerDoParallel(cores=24)

    my_data_frame = …..

    my_results = foreach(
```

1. allocate workers

2. Make ‘foreach’ loop

R multicore coding

```
install.packages(doParallel)
library(doParallel)
registerDoParallel(cores=24)

    my_data_frame = …..

    my_results = foreach(i=1:24,.combine=rbind)
```

1. allocate workers

2. Make ‘foreach’ loop

3. specify how to combine results

R multicore coding

```
install.packages(doParallel)
library(doParallel)
registerDoParallel(cores=24)

    my_data_frame = …..

    my_results = foreach(i=1:24,.combine=rbind) %dopar%
      {   …
```

1. allocate workers

2. Make ‘foreach’ loop

3. specify how to combine results

4. %dopar% runs it across cores, (%do% runs it serially)

R multicore coding

```
install.packages(doParallel)
library(doParallel)
registerDoParallel(cores=24)

    my_data_frame = …..

    my_results = foreach(i=1:24,.combine=rbind) %dopar%
      {   …
            your code here

          return(  a variable or object )
    })
```

1. allocate workers

2. Make ‘foreach’ loop

3. specify how to combine results

4. %dopar% runs it across cores, (%do% runs it serially)

R multicore coding

```
install.packages(doParallel)
library(doParallel)
registerDoParallel(cores=24)

    my_data_frame = …..

    my_results = foreach(i=1:24,.combine=rbind) %dopar%
      {   …
            your code here

          return(  a variable or object )
    })
```

1. allocate workers

2. Make 'foreach' loop

3. specify how to combine results

4. %dopar% runs it across cores, (%do% runs it serially)

BEWARE: foreach will copy data it thinks is need to every core

R multinode: parallel backend

```
library(doParallel)

cl <- makeCluster(48)
registerDoParallel(cl)
```

1. allocate cluster as parallel backend

R multinode: parallel backend

```
library(doParallel)

cl <- makeCluster(48)
registerDoParallel(cl)

my_data_frame = …..

results = foreach(i=1:48,.combine=rbind) %dopar%
  {   … your code here


        return(  a variable or object )
})
stopCluster(cl)
```

1. allocate cluster as parallel backend

2. %dopar% puts loops across cores and nodes

R multinode: parallel backend

```
library(doParallel)

cl <- makeCluster(48)
registerDoParallel(cl)

my_data_frame = …..

results = foreach(i=1:48,.combine=rbind) %dopar%
  {   … your code here


        return(  a variable or object )
})
stopCluster(cl)
```

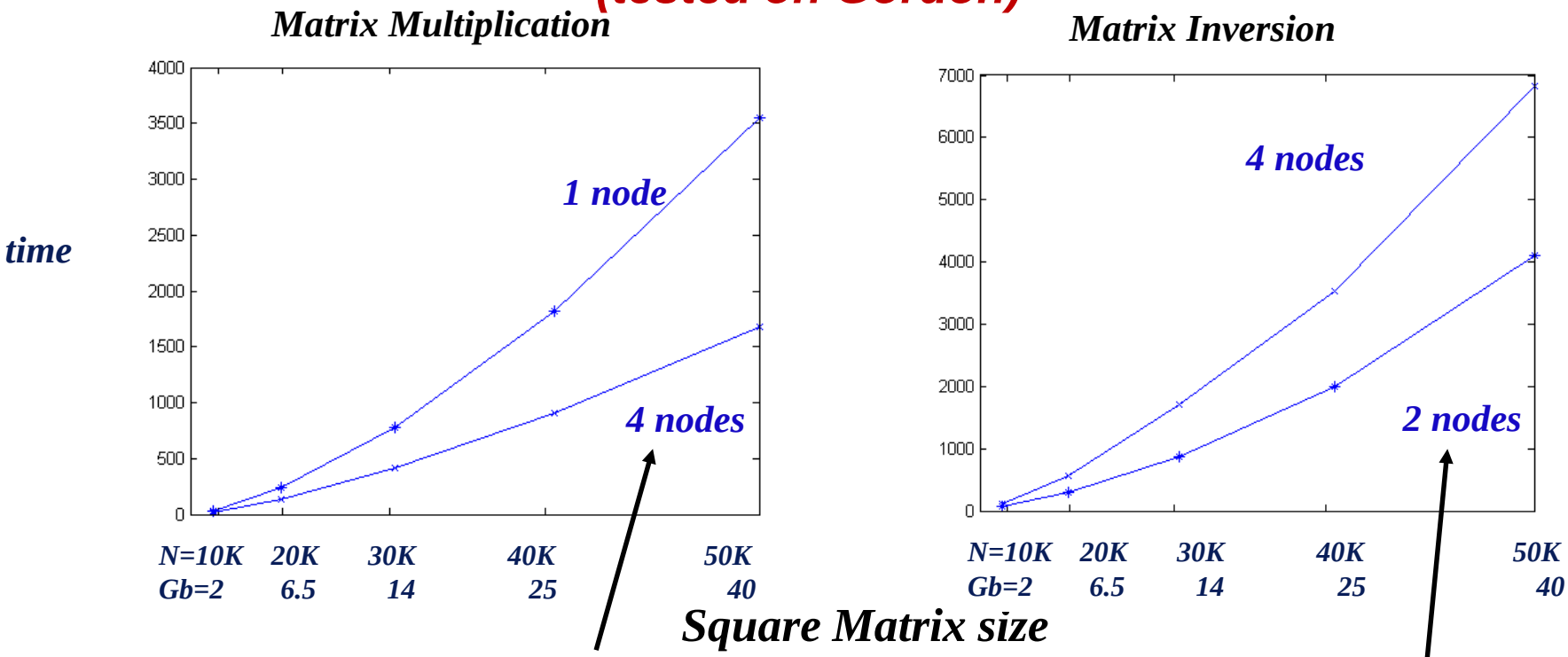
1. allocate cluster as parallel backend

2. %dopar% puts loops across cores and nodes

BEWARE: foreach will copy data it thinks is need to every core and node

Multiple Compute Nodes not always help

(tested on Gordon)

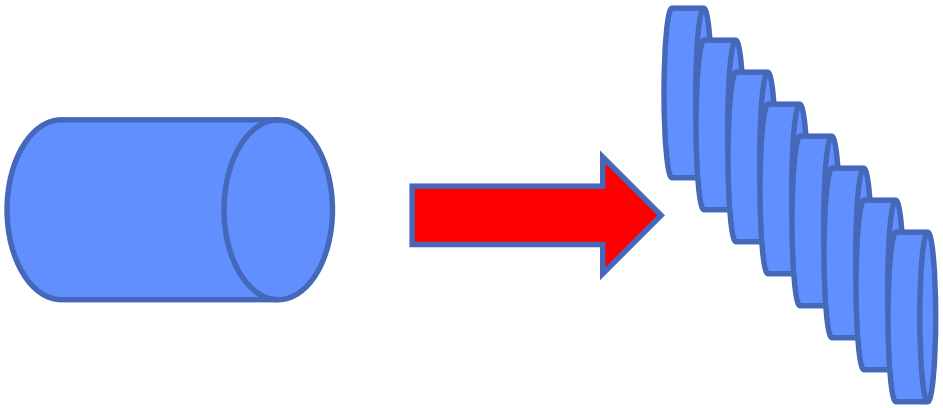


multinodes: more nodes is less time for multiplication,

less nodes is better for inversion

Another option for (embarrassingly) Parallel R

1. Split up data into N parts



Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

```
ibrun -np processors  My-perl-script
```

Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

```
ibrun -np processors  My-perl-script
```

My-perl-script:

get cpu-id &

pass it to R

Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

ibrun -np processors My-perl-script

Use MPI here

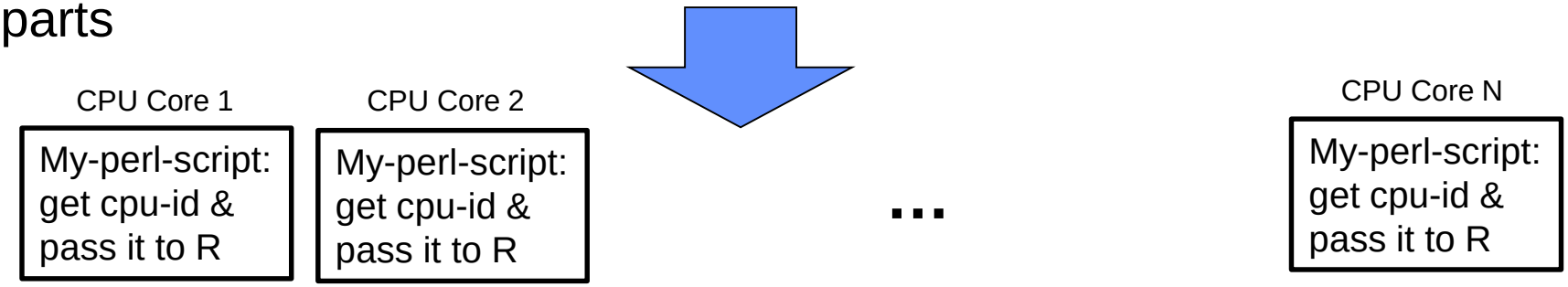
My-perl-script:
get cpu-id &
pass it to R

Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

ibrun -np processors My-perl-script

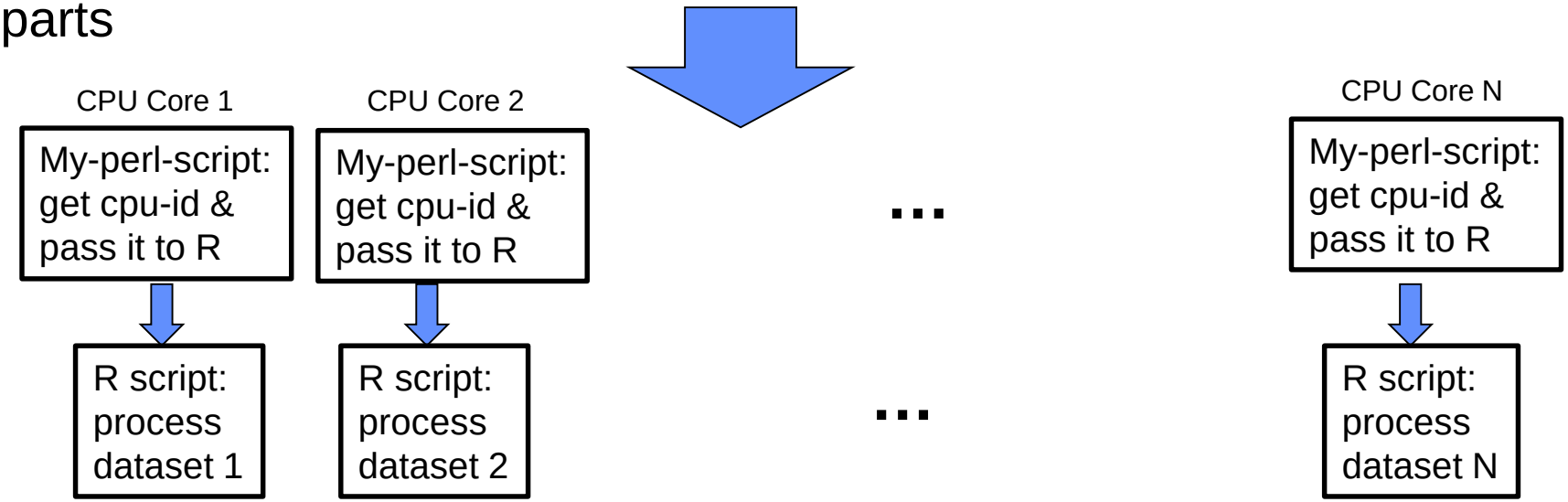


Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

ibrun -np processors My-perl-script

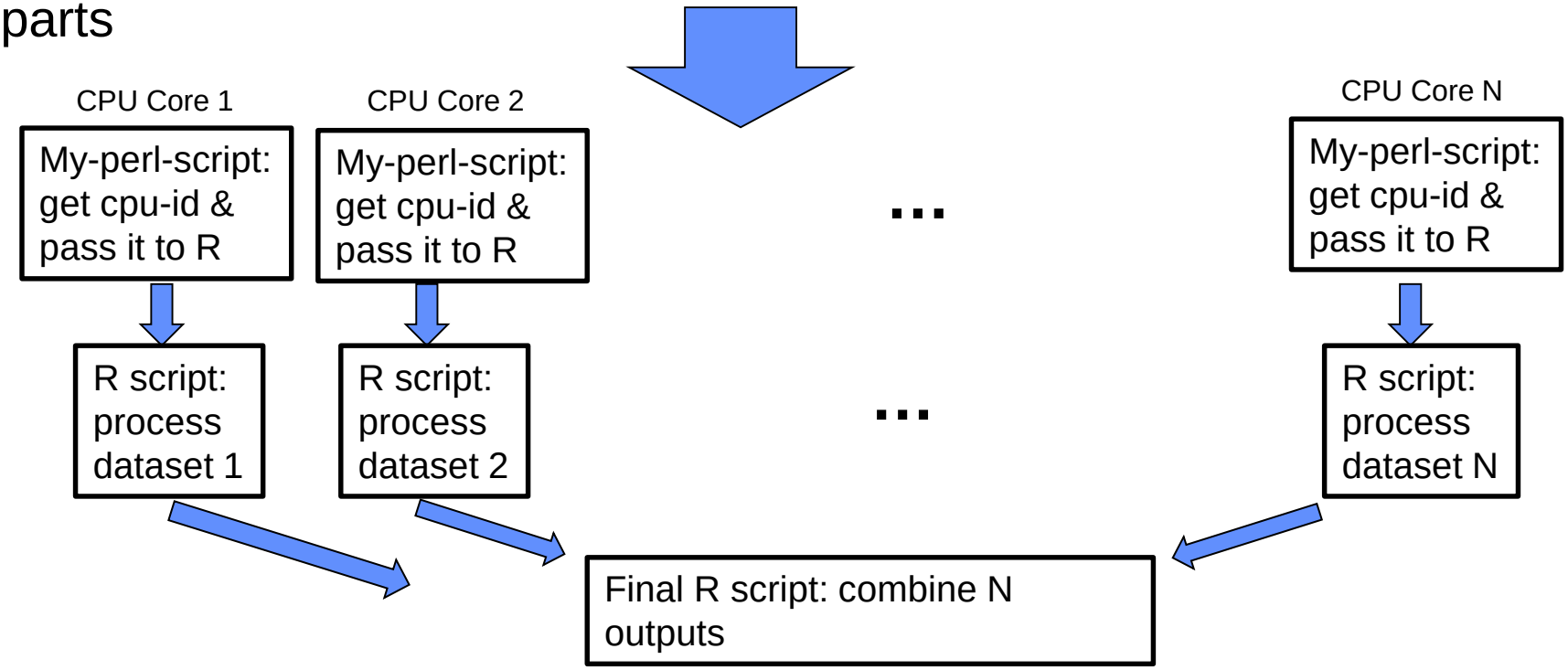


Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

ibrun -np processors My-perl-script

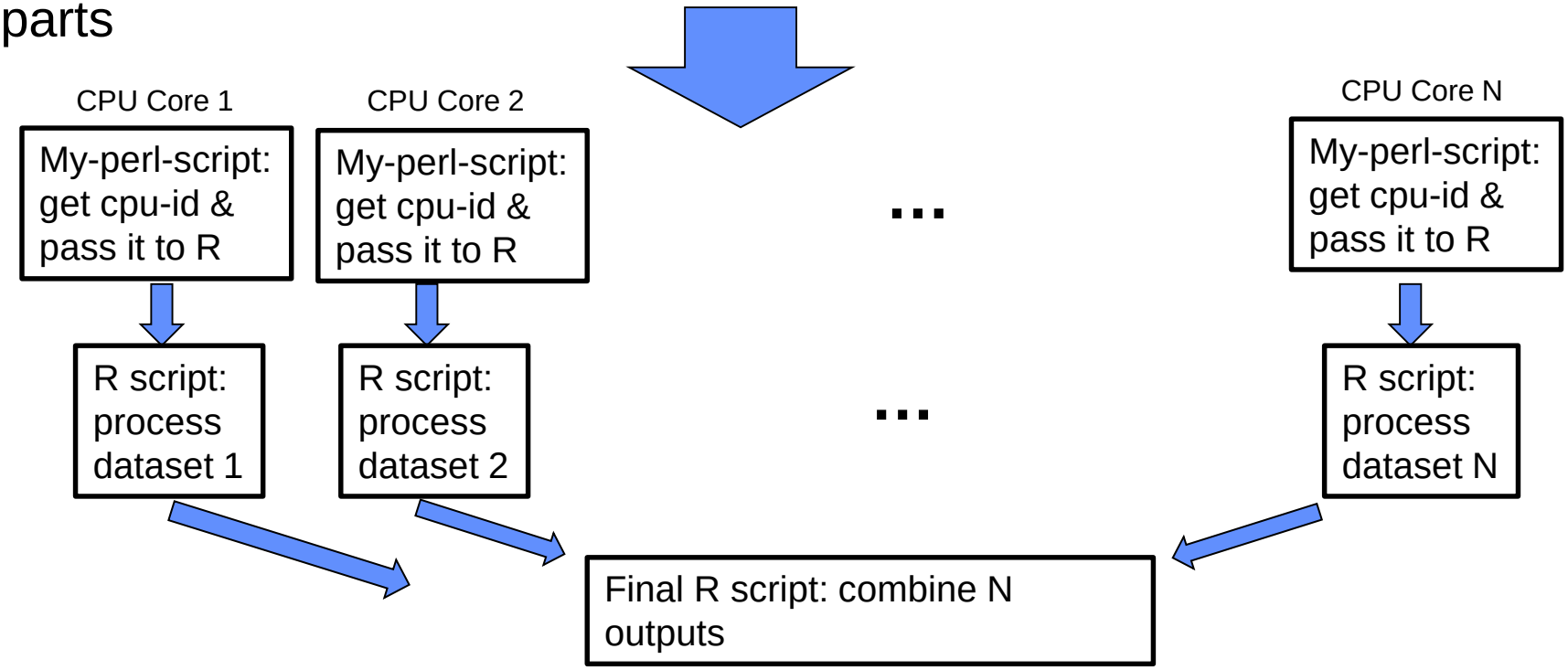


Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

ibrun -np processors My-perl-script



More programming but more flexible

Batch Script for embarrassingly Parallel R

Normal batch job info

```
#!/bin/bash
…
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
…
```

Batch Script for embarrassingly Parallel R

Normal batch job info

```
#!/bin/bash
…
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
…

module load R

ibrun --npernode 24 --tpp 1 perl my_perl_script.pl
```

One perl script on each core (ie processor) on each node

Batch Script for embarrassingly Parallel R

Normal batch job info

```
#!/bin/bash
…
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
…

module load R

ibrun --npernode 24 --tpp 1 perl my_perl_script.pl

# ---  OR  ---------

ibrun --npernode 1 --tpp 24 perl my_perl_script.pl
```

One perl script on each core (ie processor) on each node

Hybrid:
One perl script on each node, R can then use 24 cores with ‘foreach’

Example: scaling MCMC

Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models, Frederico Bumbaca, UCIrvine, et al in print

- *Probabilities of user web activity interdependent through a hierarchical model*
- *MCMC search for probabilities made independent through a phased approach.*
- *Ran on SDSC Comet with embarrasing parallelization*

(Using rhierMnlRwMixturefunction in the R package, bayesm)

# Individuals	Cores	Individ per Core	Total Minutes (I/O time)
100 million	**1,7282 (max)**	**~ 58K**	**206 (38)**

Example 2: scaling likelihood estimation

Social network evolution

- *A large model of users' connections with interdependent variance terms for different actions*

- *Optimization, with ~70M observations (5-8Gb), takes > 48 hours on 1 compute node.*

- *R doParallel copies too much data across nodes or cores*
- *R-mpi not flexible enough with nodes and cores*

- *Ran with embarrasing parallelization on parts of data across nodes, with R parallel across cores (but not all cores),*

(using Optim, doParallel, and send results back to main node through files)

# Connections	Nodes (Cores)	Approx Hours
~70M	12 (180 of 288)	2-3

R parallel exercise

- **Open & run Exercise**
  - remember that foreach assumes independence between loops
  - Start with smallish NxP data matrix
- **Look at memory usage in top command**
- **R will crash if too many cores use too much memory (128Gb is max on a node)**

R doParallel exercise

1 start a notebook,
2 before running the cells, in terminal window enter:
$ squeue -u $USER
$ ssh comet-##-##
$ top -u $USER

3 run the SML1 notebook, then view top listing

```
p4rodrig@comet-19-14:~
(base) [p4rodrig@comet-ln2 reverse-proxy]$
(base) [p4rodrig@comet-ln2 reverse-proxy]$
(base) [p4rodrig@comet-ln2 reverse-proxy]$
(base) [p4rodrig@comet-ln2 reverse-proxy]$
(base) [p4rodrig@comet-ln2 reverse-proxy]$
(base) [p4rodrig@comet-ln2 reverse-proxy]$ squeue -u $USER
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REA
SON)
          35036476   compute batch_r. p4rodrig  R    1:34:05      1 comet-19-14
(base) [p4rodrig@comet-ln2 reverse-proxy]$ ssh comet-19-14
Rocks Compute Node
Rocks 7.0 (Manzanita)
Profile built 15:20 03-Dec-2019

Kickstarted 16:09 03-Dec-2019

                    WELCOME TO

[p4rodrig@comet-19-14 ~]$ top -u $USER
```

```
p4rodrig@comet-19-14:~
top - 16:04:19 up 144 days,  2:05,  1 user,  load average: 0.00, 0.01, 0.05
Tasks: 575 total,   1 running, 574 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 13165400+total, 12268892+free,  3747980 used,  5217100 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 12264201+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
28264 p4rodrig  20   0  165368   2672   1552 R   0.3  0.0   0:01.05 top
 1402 p4rodrig  20   0  113324   1732   1340 S   0.0  0.0   0:00.03 slurm_scri+
 2184 p4rodrig  20   0  330120  58316   9464 S   0.0  0.0   0:02.11 ZMQbg/1
 2442 p4rodrig  20   0 1259244 111404  10780 S   0.0  0.1   0:08.83 R
28170 p4rodrig  20   0  178644   3032   1176 S   0.0  0.0   0:00.00 sshd
28171 p4rodrig  20   0  118524   2192   1664 S   0.0  0.0   0:00.01 bash
28910 p4rodrig  20   0 1268160 111392   1728 S   0.0  0.1   0:00.02 R
28911 p4rodrig  20   0 1268160 111392   1728 S   0.0  0.1   0:00.02 R
28912 p4rodrig  20   0 1268160 111392   1728 S   0.0  0.1   0:00.02 R
28913 p4rodrig  20   0 1268160 111392   1728 S   0.0  0.1   0:00.02 R
28914 p4rodrig  20   0 1268160 111392   1728 S   0.0  0.1   0:00.02 R
28915 p4rodrig  20   0 1268160 111388   1724 S   0.0  0.1   0:00.02 R
28916 p4rodrig  20   0 1268160 111388   1724 S   0.0  0.1   0:00.02 R
28917 p4rodrig  20   0 1268160 111388   1724 S   0.0  0.1   0:00.02 R
28918 p4rodrig  20   0 1268160 111388   1724 S   0.0  0.1   0:00.02 R
```

- **pause**

Other R packages:

- "Stan" - MCMC tool with an R or Python interface

  (an evolution from BUGS)

- Rspark - R interface to Spark
- pdbR - higher level over R-MPI, distributed matrix support and other (better for dense matrices vs Spark)
- R openMP - (e.g. if you want to program your own foreach)
- Ff, bigmemory – map data to files (can help with foreach)

- HiPLAR - GPU and multicore for linear algebra
- Rgputools – GPU support

  (GPUs have data transfer overhead costs)

pbdR package

See https://pbdr.org/packages.html

- **API on top of MPI and Scalapack Lin. Algebra library**
- **Sets up virtual grid to handle large matrix multiplication**

pbdR sample code

Set it up like MPI

```
library(pbdDMAT)

init.grid()                    #you can select grid sizes to map matrices onto cores

myr   =comm.rank()   #it works over MPI
mys   =comm.size()


dx <- ddmatrix(…      # ddmatrix is a new object with associated methods
```

A new object type for matrices

R bigmemory library

“bigmemory” package for large matrices

maps data to files

might help foreach memory management

```
library('bigmemory')    #load libraries
library('bigalgebra')

 X = as.big.matrix(X)
          #convert regular matrices to “big” matrices and use them as normal
```

A new object type for matrices

How to use R directly on Comet

1. Get a compute node:

[Unix]$ *: srun --partition=computed --pty --nodes=1 --ntasks-per-node=24 -t 00:30:00 --wait=0 --export=ALL -A your-account /bin/bash*

2. *Start R*

[Unix]$ *module load R*
[Unix]$ *R* *(this gets an interactive R session)*

>quit() *(to exit R)*

[Unix]$ exit *(to exit the compute node)*

Installing your own R Packages

- **In R:**

  install.packages('package-name')

  (see https://cran.r-project.org/ for package lists and reviews)

- **Sometimes on Comet, you might need to be explicit:**

  install.packages('ggmap',
  repos='http://cran.us.r- project.org',dependencies=TRUE)

  If compiling is required and you get an error, call support

THE END