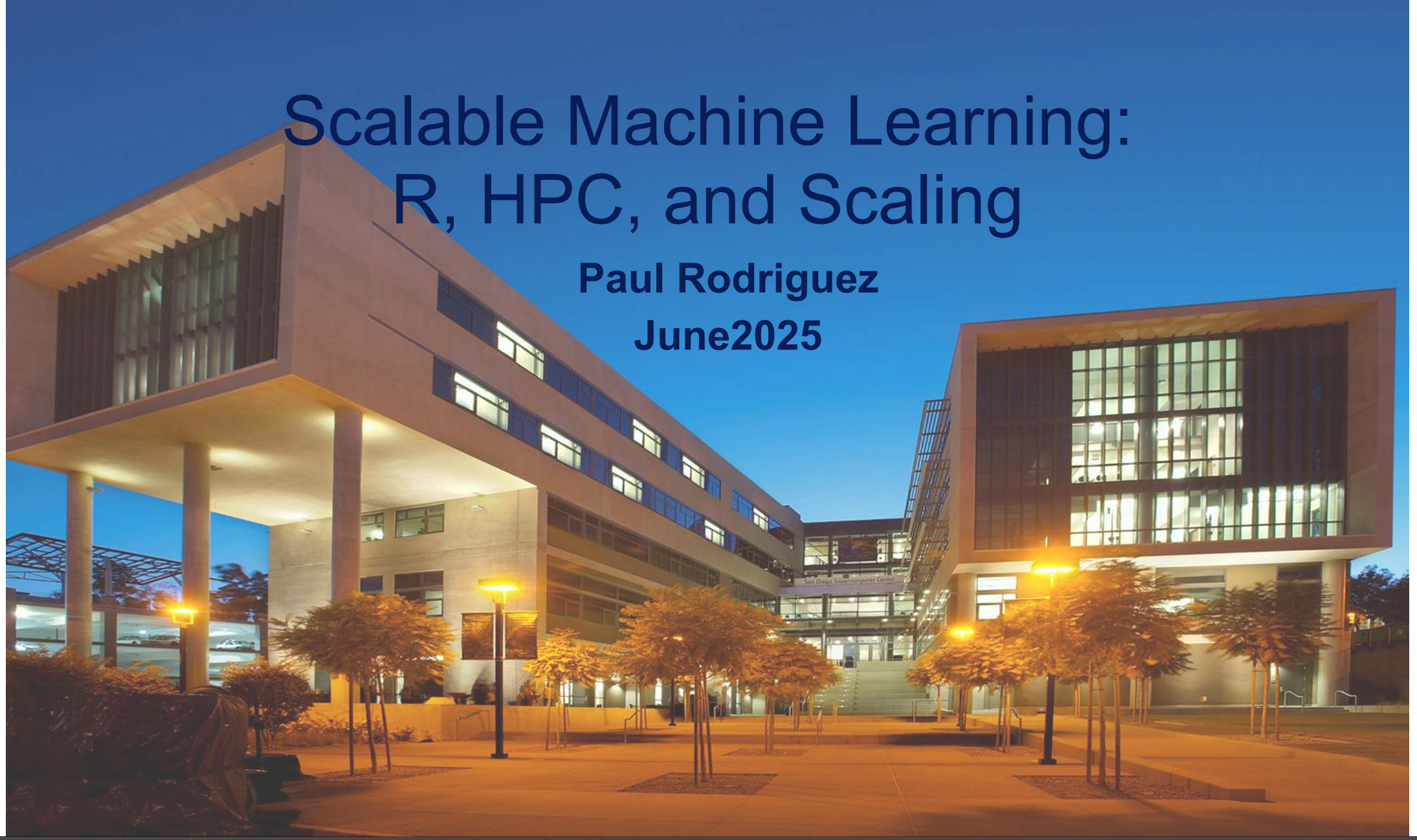
Scalable Machine Learning: R, HPC, and Scaling

Paul Rodriguez

June2025



Outline

- R on Expanse
- R and Scaling
- Parallel R
- Embarrassingly Parallel R

3 ways to run R on Expanse

1. **Run a SLURM batch script of R commands**

2. **Acquire a node with the “srun” command and use terminal window**

3. **Use Expanse portal to start R-studio**

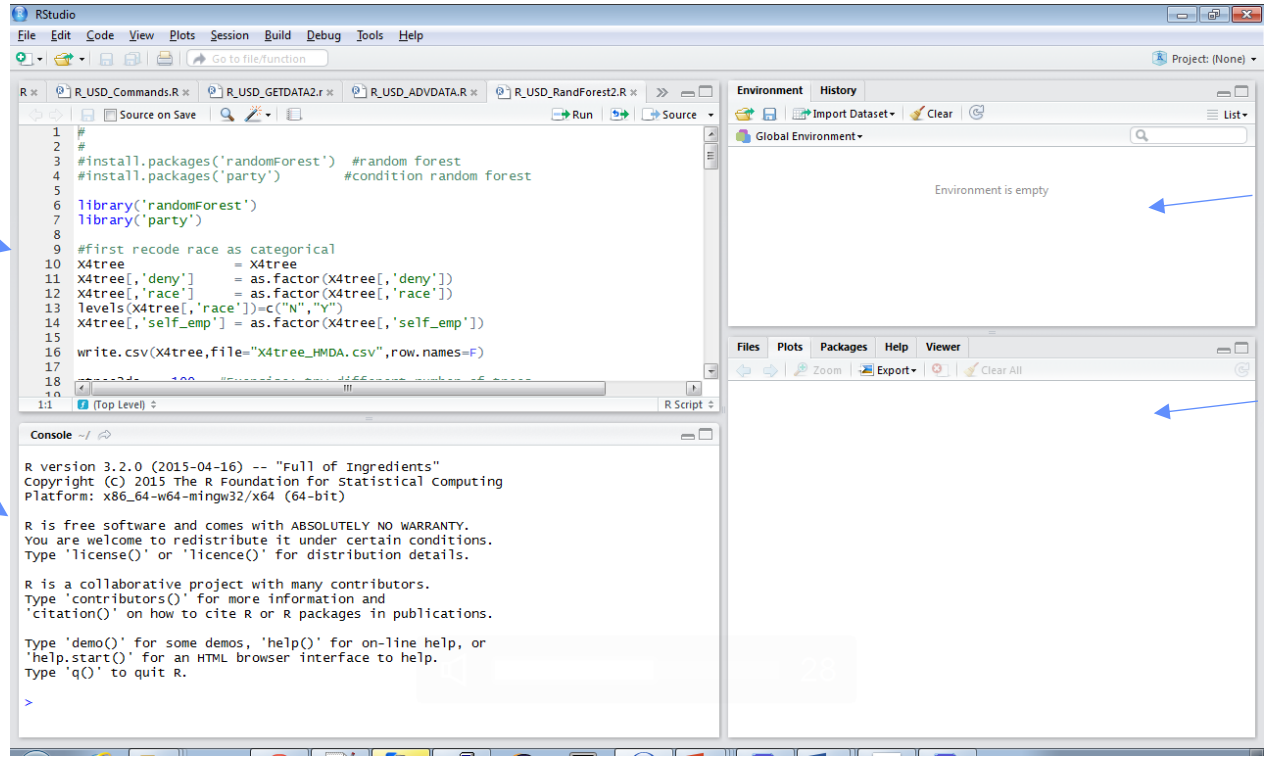
   (see user guide for details)

(4. Also you can run R in a Jupyter notebook)

Rstudio: an integrated development environment

Edit window to Build scripts

R console



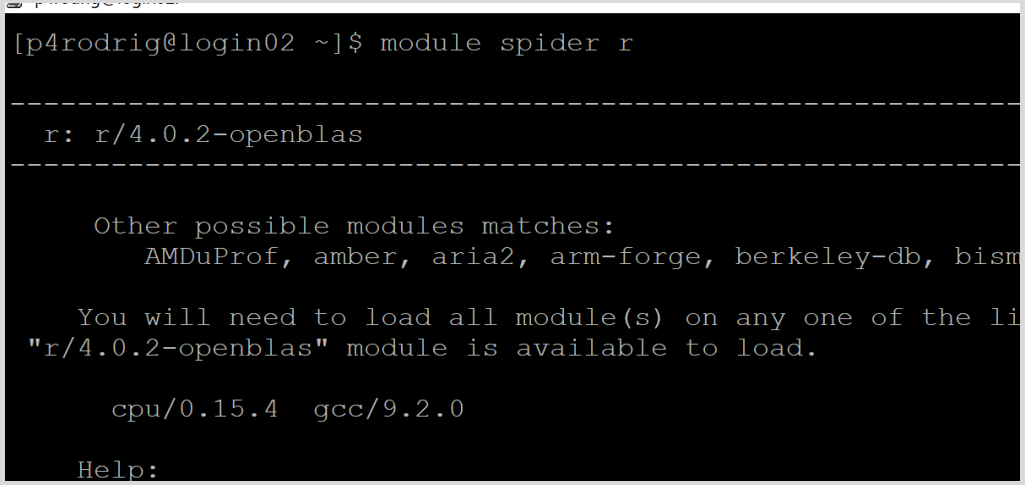
Environment Information on variables and command history

Plots, help docs, package lists

R interactively on Expanse command line

1. Get an interactive compute node (see demo slides):

2. Try
$ module spider r *(this tells you what modules you need)*



3. Enter
$ module load cpu/0.15.4
$ module load gcc/9.2.0
$ module load r/4.0.2-openblas

$ R

R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
…..
Type 'q()' to quit R.

>

R strengths for HPC (IMHO)

- **Data Wrangling -**

- **Particular statistical procedure implementations -**
  Imputation methods (for missing data)
  Sampling methods
  Instrument Variable (2 stage) Regression
  Matching subjects for pairwise analysis
  Generalized Linear Model (e.g. logistic regression)
  MCMC routines (but Stan is likely better package)
  Some ML models (e.g. randomForest, LASSO)

R strengths for HPC (IMHO)

- **Data Wrangling –**

- **Particular statistical procedure implementations -**
  - Imputation methods (for missing data)
  - Sampling methods
  - Instrument Variable (2 stage) Regression
  - Matching subjects for pairwise analysis
  - Generalized Linear Model (e.g. logistic regression)
  - MCMC routines (but Stan is likely better package)
  - Some ML models (e.g. randomForest, LASSO)

Inferential Stats, Experimental Studies

Biostats/BioInformatics Data Science, Machine Learning

R Scaling In a nutshell

- R uses BLAS/LAPACK math libraries for operations on vectors

  [Same for Matlab and Python]

- R packages provide multicore, out-of-core, multinode, or distributed data (SparkR) options

  [Same for Matlab and Python]

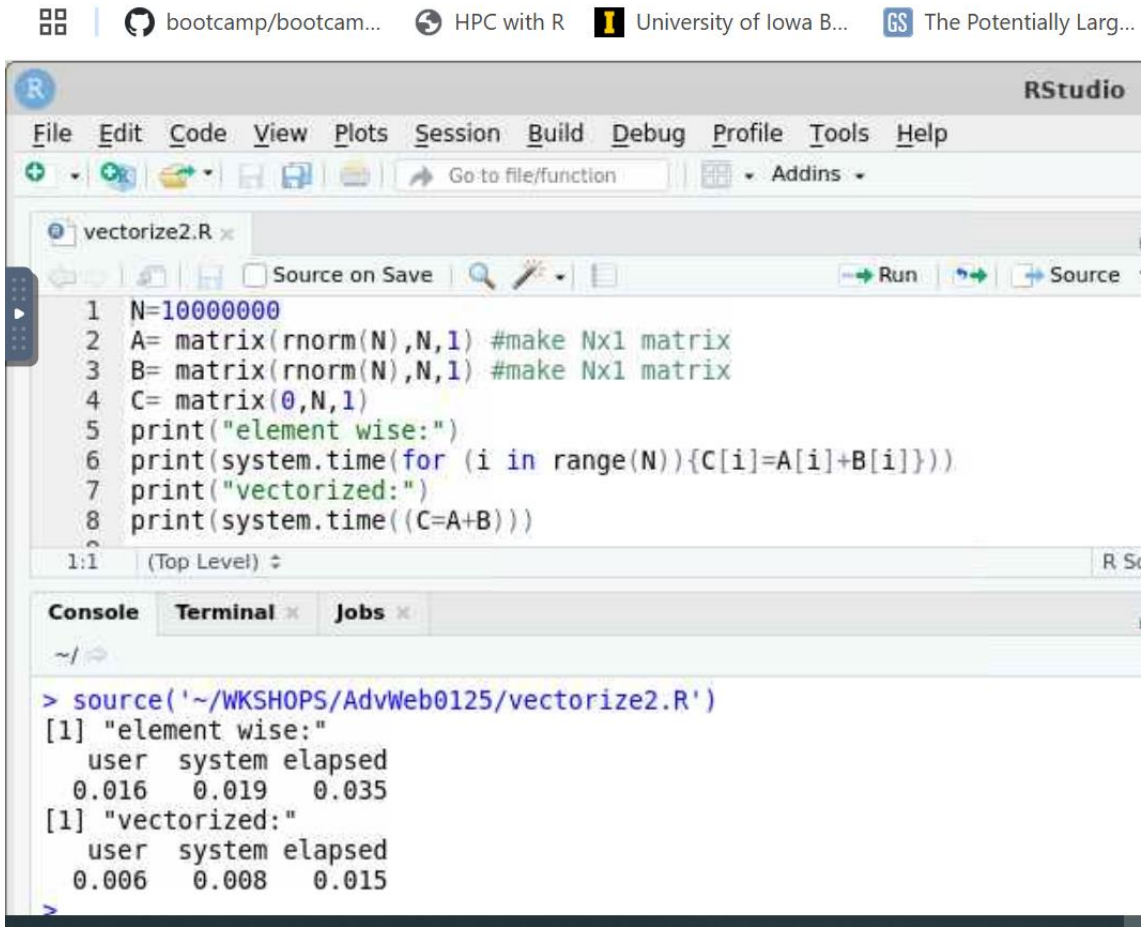
- Some ML model implementations may be built to use parallel backends (review the available options)

Optimizing R code

- **In general:**

uses vector operations instead of for loops on individual vector elements

(eg add 2 vectors directly, instead of each element 1…N)



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
- It also works for multinode (runs 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

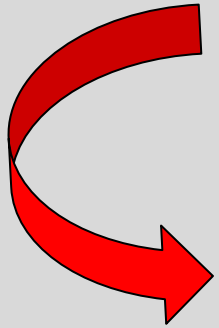
```
install.packages(doParallel)
library(doParallel)
registerDoParallel(cores=24)

    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

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

3. specify how to combine results



BEWARE: foreach will copy data to every core if it seems necessary

R multinode uses same steps, but:

BEWARE: copying data across nodes is *MUCH* higher communication costs

```
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

2. Use foreach and %dopar%

Exercise for home: Testing R parallel, command line

Step:

Log into expanse terminal

$ srun --account=__
--partition=shared
--nodes=1 --ntasks-per-node=1
--cpus-per-task=8 --mem=16G
--time=04:00:00 --pty --wait=0
/bin/bash

$ cd github repo: _r_on_HPC folder

$ 'module spider r' to see what to load

$ R (run one time)

> install.packages("doParallel") (say 'yes' for local install)

7. $ Rscript –vanilla TestDoParallel_v1.R

```
4rodrig@login01 RHPC]$ module load cpu/0.15.4  gcc/9.2.0

e following have been reloaded with a version change:
1) cpu/0.17.3b => cpu/0.15.4

4rodrig@login01 RHPC]$ module load r/4.0.2-openblas
4rodrig@login01 RHPC]$
```

```
train113@exp-1-24 4.2.RandHPC]$ Rscript --vanilla ./TestDoParal
oading required package: doParallel
oading required package: foreach
oading required package: iterators
oading required package: parallel
1] "starting dopar test"
1] "Using N rows= 10000  P cols= 200"
1] "X size is: 15.3 Mb"
```

Exercise: Testing R parallel, 'top' command

Also try this:

Login to Expanse with a second terminal window

$ squeue –u $USER

$ ssh exp-#-## (ssh into that compute node)

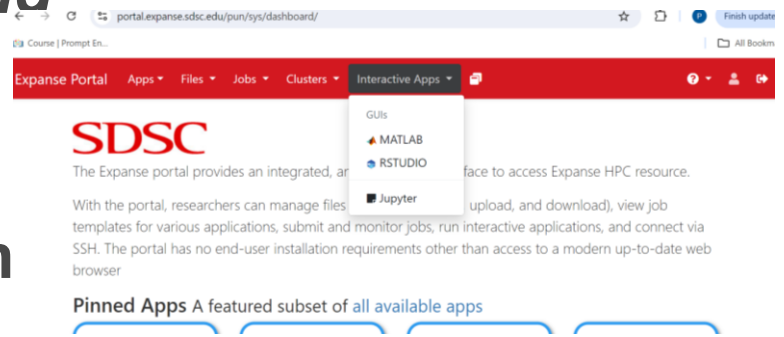
$ top –u $USER

- how's memory usage?

enter H to see threads,
enter f -> down arrow -> space -> esc to toggle cpuid

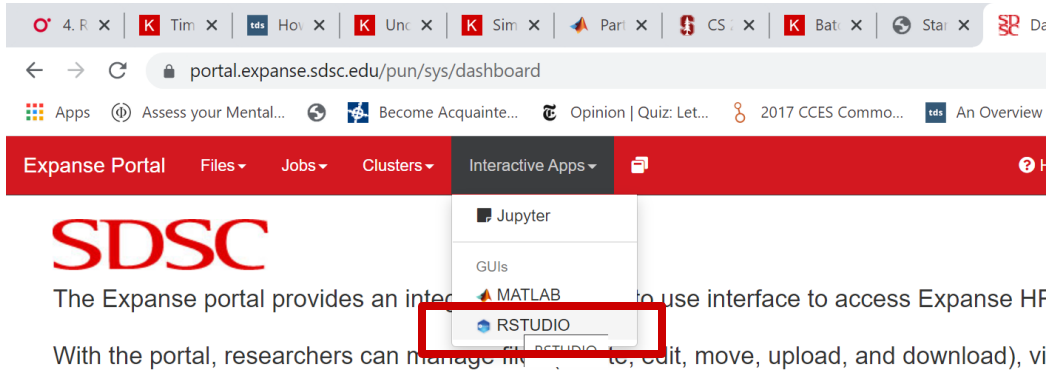
Exercise: Testing R parallel in portal

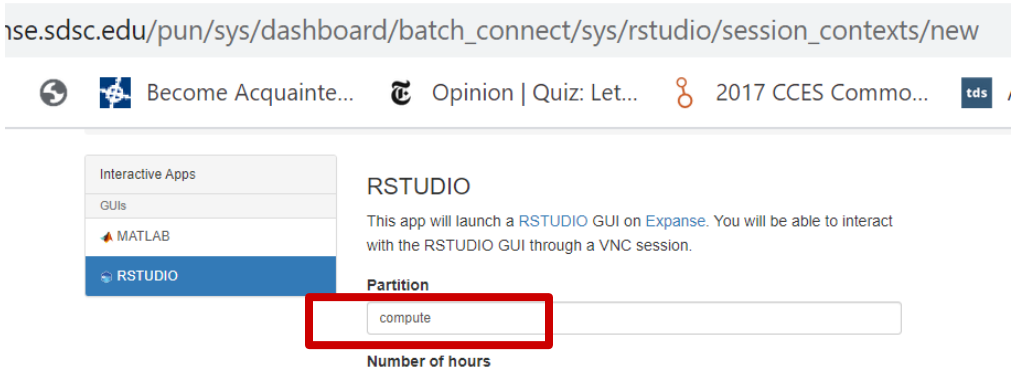
1. **Log into expanse portal and start R studio (need Access id)**
   goto URL: ***https://portal.expanse.sdsc.edu/training***

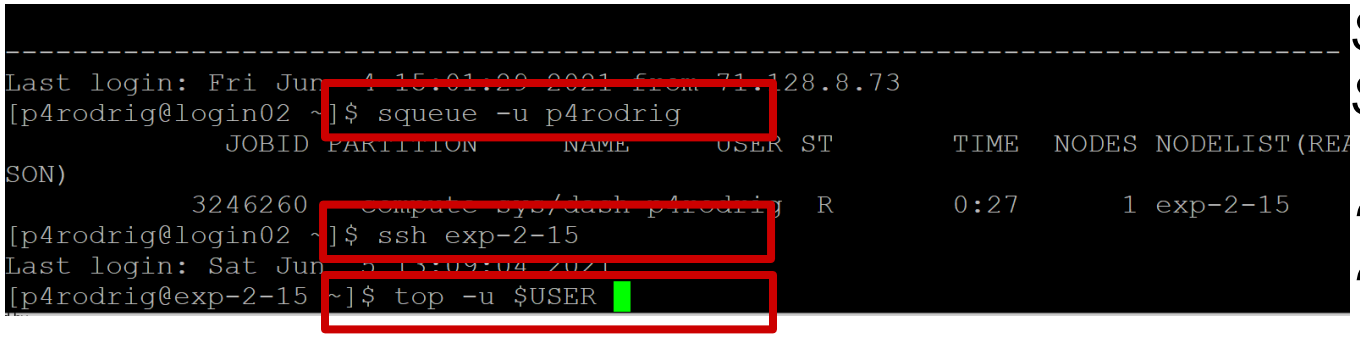


2. **Also log into expanse command line and ssh to com node:**

3. **run ‘top –u username’ to see performance**
   - look for tradeoffs in memory vs execution as matrix size varies (see next slides)







1 Open portal ->
Interactive Apps ->
Rstudio

Enter
Node: "compute"
Cores: "64"
Memory: 124 Gb
Res: ciml25cpu
(other fields defaults ok)

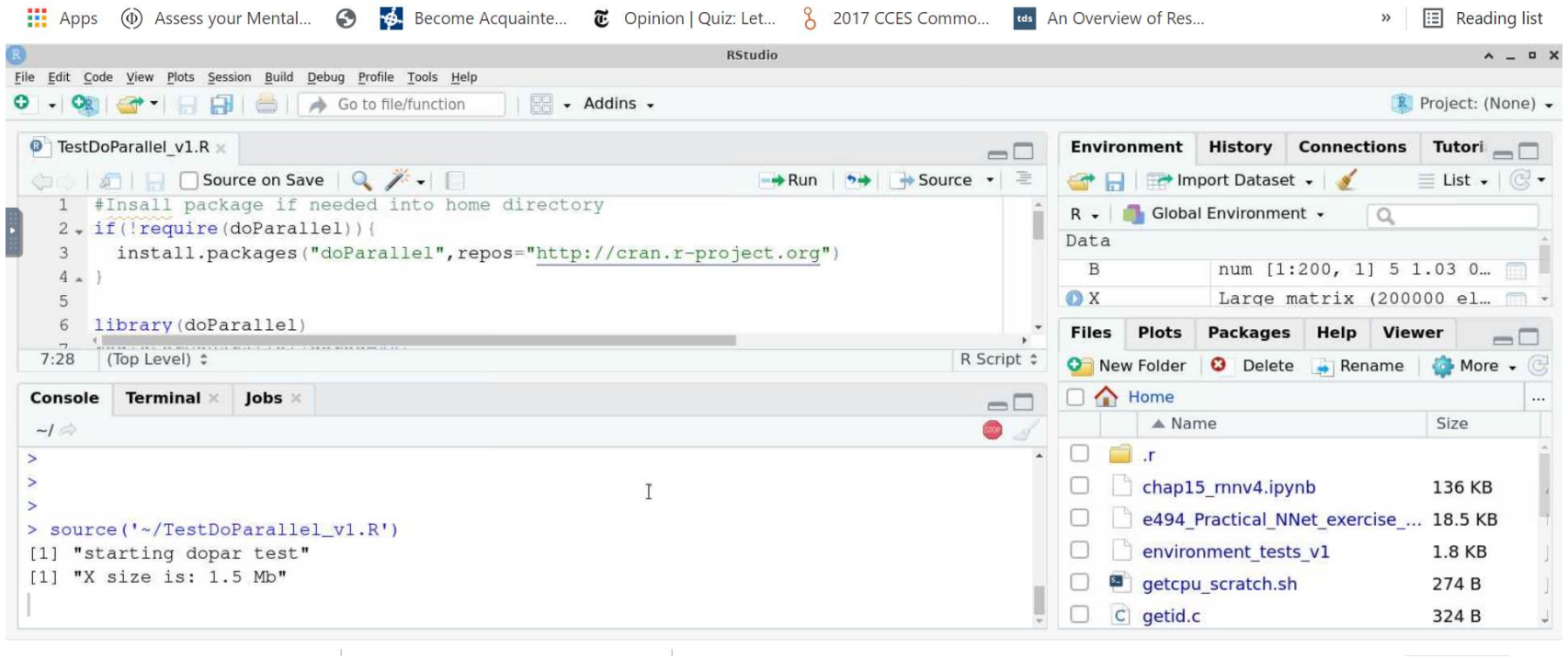
2 Also login to Expanse terminal window
$ squeue –u $USER
$ ssh exp-##-##
$ top –u $USER

'H' will toggle threads
'f' , downarrow to P, space, esc.

3 Open the 'Test_doParallel ' Rscript

Select 'source' to run the whole script, it will install 'doParallel' package (if the R installation doesn't have it already)

look for # <<< ----- comments to change data parameters

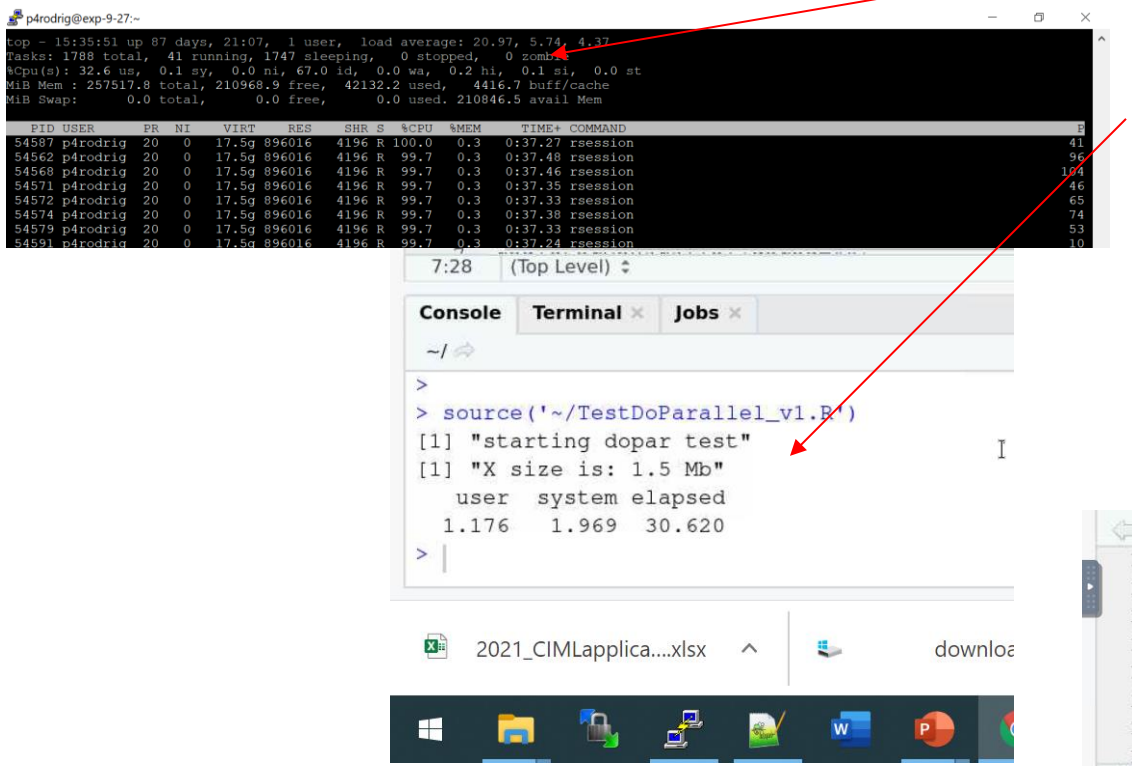


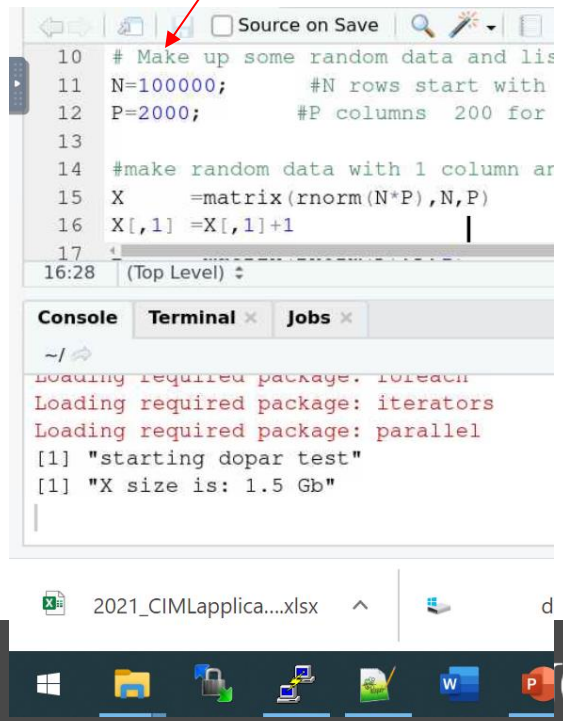
Review the top output

Notice the elapsed time and memory size

Change the NxP matrix size and rerun

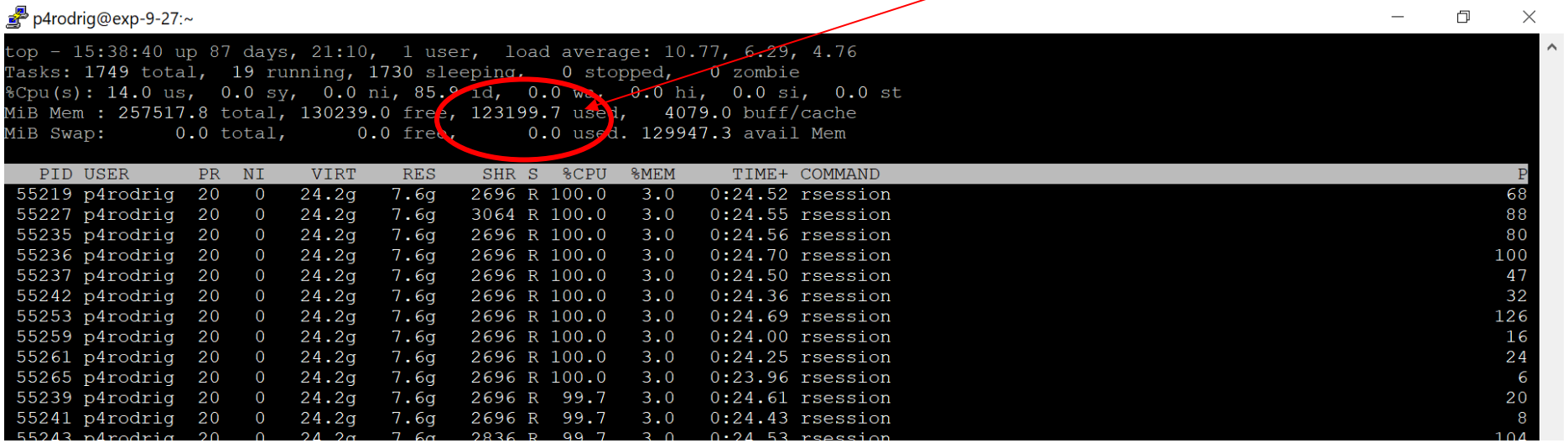
(start with N=10K, P=2K)





Try this at home:

Let N=100K , P=2000
Notice the memory used is close to 124Gb we asked for



If you ask for 248Gb will it run?
What if you use only 24 cores?

Parallezing for loops
(pseudo code)

R with
doParallel

registerDoParallel()

foreach with dopar,

combine results

Parallezing for loops

(pseudo code)

R with doParallel	Matlab with parallel toolbox
registerDoParallel()	*parpool()*
foreach with dopar,	*parfor* or *'spmd' with distributed arrays*
combine results	gather array

Parallezing for loops

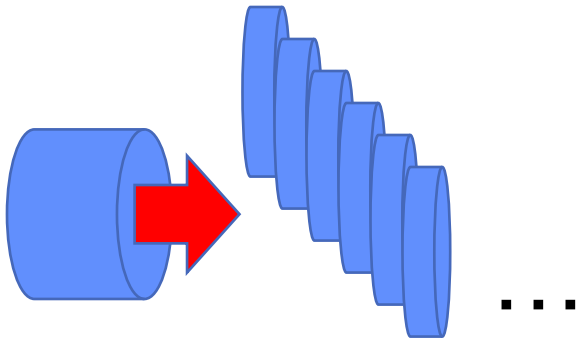
(pseudo code)

R with doParallel	Matlab with parallel toolbox	Python with dask.distributed
registerDoParallel()	*parpool()*	Client(numwkrs)
foreach with dopar,	*parfor* or *'spmd' with distributed arrays*	for i in range(numwkrs): A=delayed(my_func)(i) Acombine.append(A)
combine results	gather array	Acombined.compute()

An option for (embarrassingly) Parallel R

If you can process data parts independently then you can parallelize in an embarrassingly simple way

1. First, s: Split up data into N parts



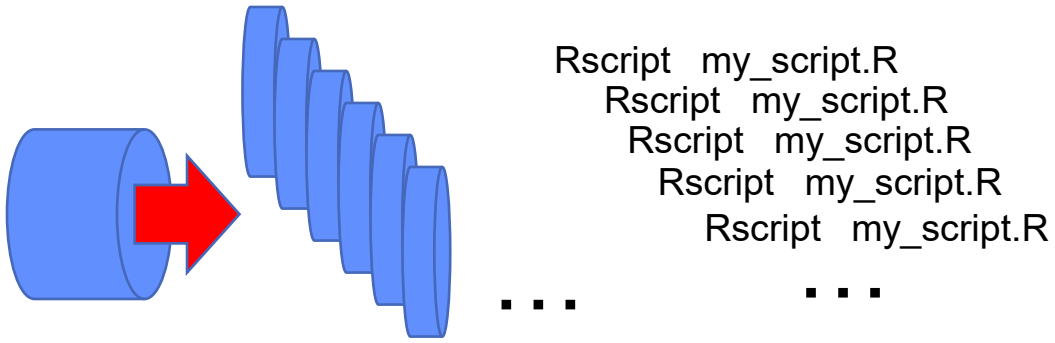
…

An option for (embarrassingly) Parallel R

If you can process data parts independently then you can parallelize in an embarrassingly simple way

1. First, s: Split up data into N parts

2. Launch N instances of your R script, one per CPU core. Each instance processes one (or more) datasets

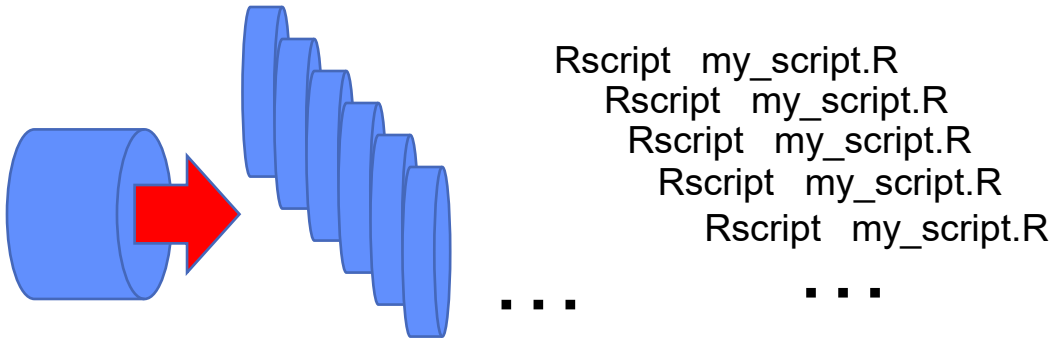


An option for (embarrassingly) Parallel R

If you can process data parts independently then you can parallelize in an embarrassingly simple way

1. First, s: Split up data into N parts

2. Launch N instances of your R script, one per CPU core. Each instance processes one (or more) datasets



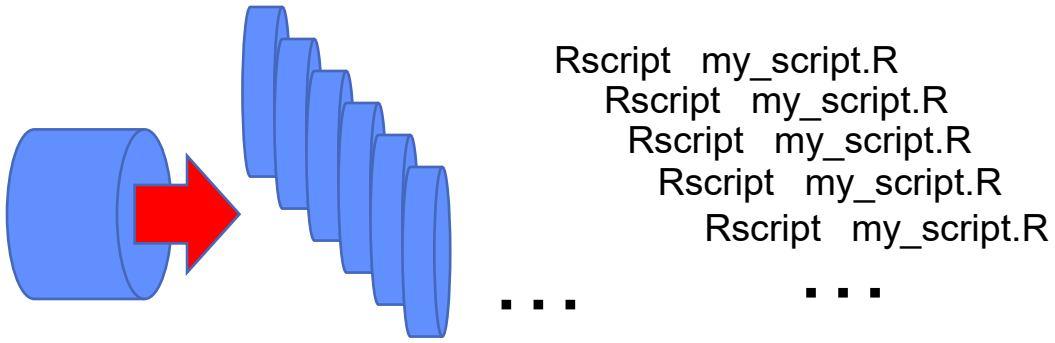
ISSUE: how to launch R scripts and pass arguments about which dataset to process?

An option for (embarrassingly) Parallel R

If you can process data parts independently then you can parallelize in an embarrassingly simple way

1. First, s: Split up data into N parts

2. Launch N instances of your R script, one per CPU core. Each instance processes one (or more) datasets



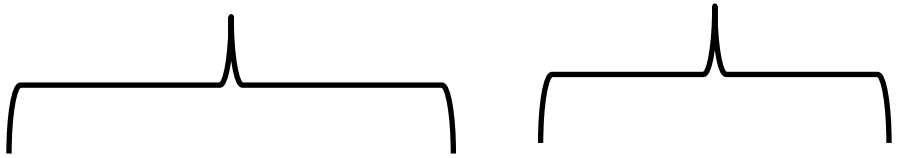
ISSUE: how to launch R scripts and pass arguments about which dataset to process?

2 ways:
mpirun command (more flexible)
parallel command (Unix tool, maybe simpler)

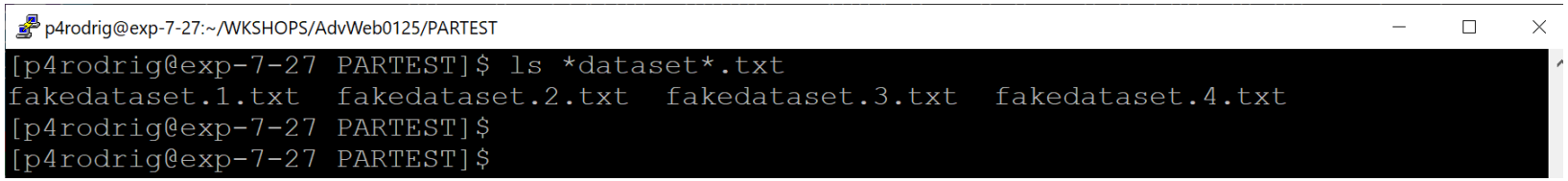
Example Using GNU tool “parallel”

Get list of files

‘|’ pipe list to parallel command



```
$ ls *dataset*.txt   |   parallel   …
```



Example Using GNU tool “parallel”

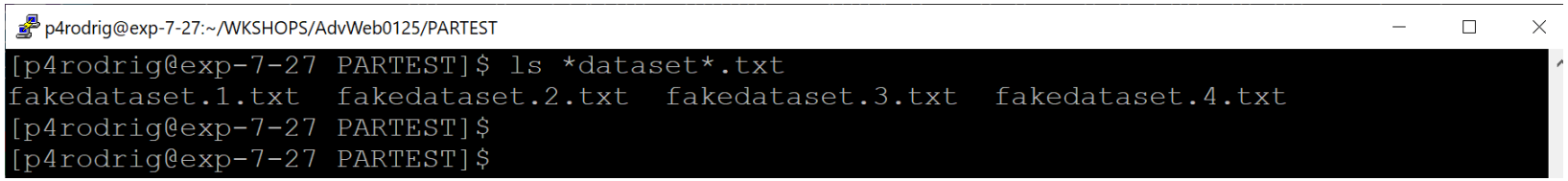
Get list of files

‘|’ pipe list to parallel command

‘-j’ 4 means run 4 jobs (1 job on each CPU core)

```
$ ls *dataset*.txt   |   parallel   -j  4  Rscript  your-r-script.R
```

The parallel command passes file names as arguments to Rscript




```
#!/usr/bin/env Rscript
args = commandArgs(trailingOnly=TRUE)

print(paste('starting script'))
for (i in seq(length(args))) {
    print(paste('arg: ',i,' value:', args[i]))
    Sys.sleep(5)
}
```

Example Using GNU tool “parallel”

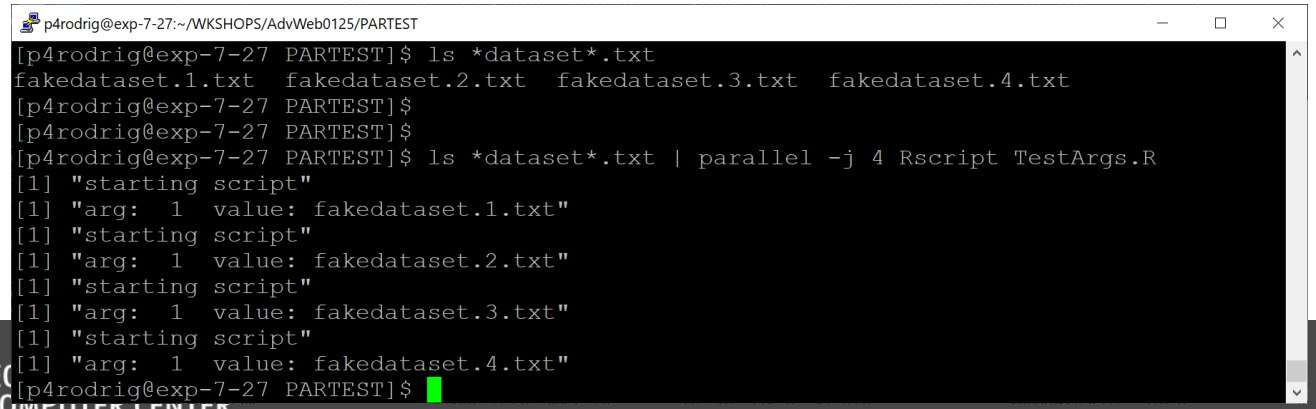
Get list of files

‘|’ pipe list to parallel command

‘-j’ 4 means run 4 jobs (1 job on each CPU core)

```
$ ls *dataset*.txt   |   parallel   -j  4  Rscript  your-r-script.R
```

The parallel command passes file names as arguments to Rscript

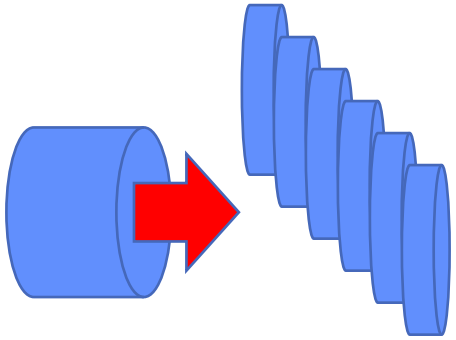


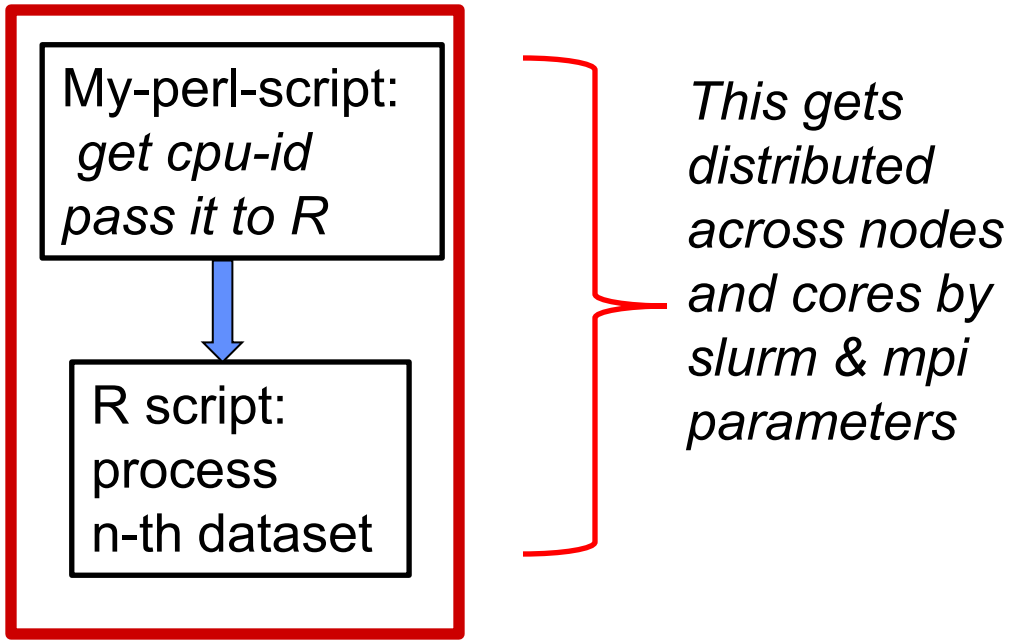

```
#!/usr/bin/env Rscript
args = commandArgs(trailingOnly=TRUE)

print(paste('starting script'))
for (i in seq(length(args))) {
    print(paste('arg: ',i,' value:', args[i]))
    Sys.sleep(5)
}
```

mpirun option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script or command line:
   mpirun … my-perl-script





Slurm parameters: one R instance per core across all nodes

Normal batch job info

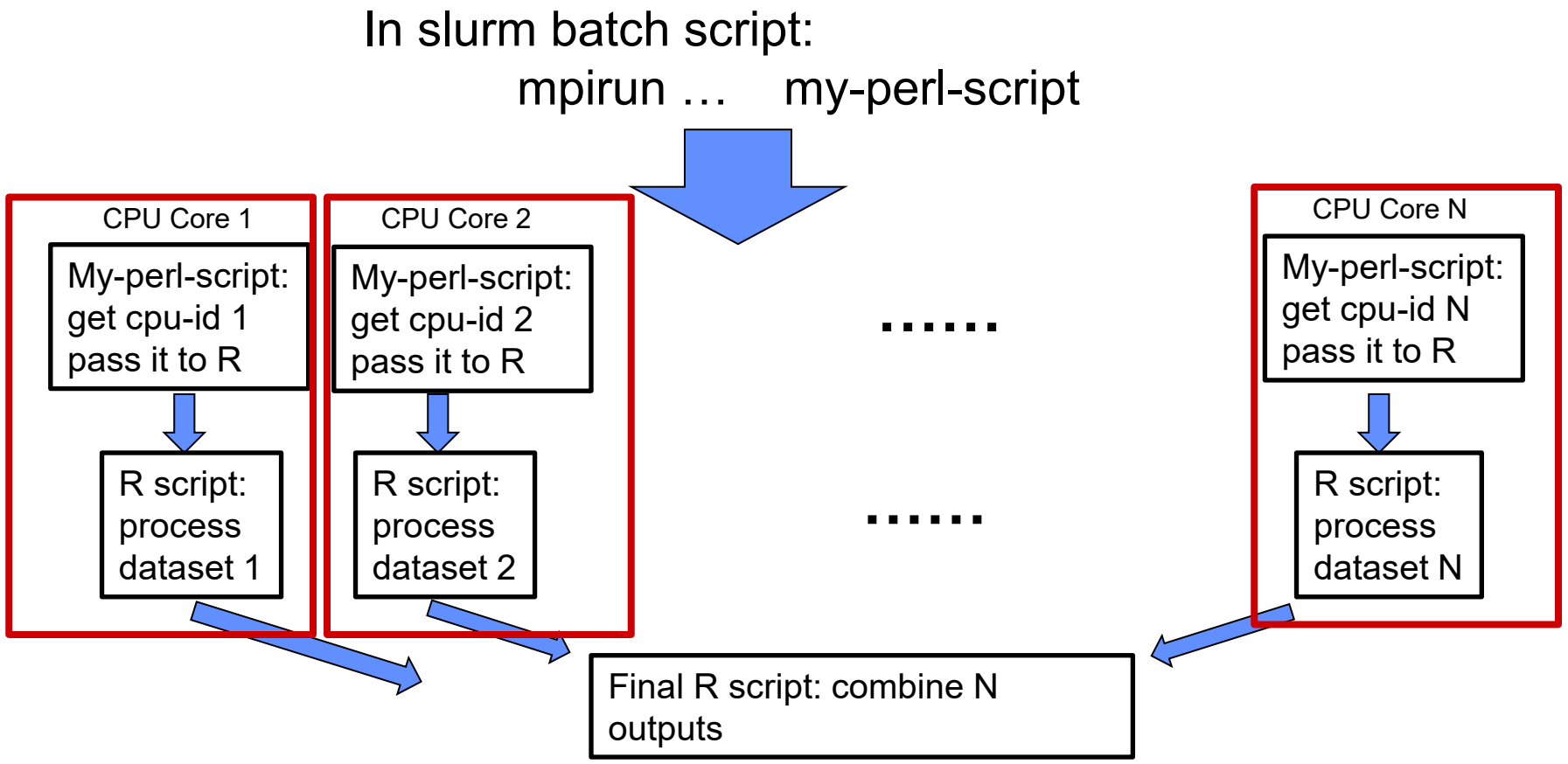
```
…
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=128
#SBATCH --cpus-per-task=1

module load slurm
module load cpu
module load gcc
module load intel-mpi

mpirun -genv I_MPI_PIN_DOMAIN=omp:compact ./get_mpirank_runcmd.pl
```

2 x128 = 256 mpi ranks

256 perl script/R instances
1 core each

(based on /cm/shared/examples/sdsc/mpi-openmp-hybrid/hybrid-slurm.sb)

one R instance per core across all nodes



Slurm parameters: one R instance per node with 128 cores per R instance

Normal batch job info

```
…
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=128

module load slurm
module load cpu
module load gcc
module load intel-mpi

module load r

mpirun -genv I_MPI_PIN_DOMAIN=omp:compact ./get_mpirank_runcmd.pl
```

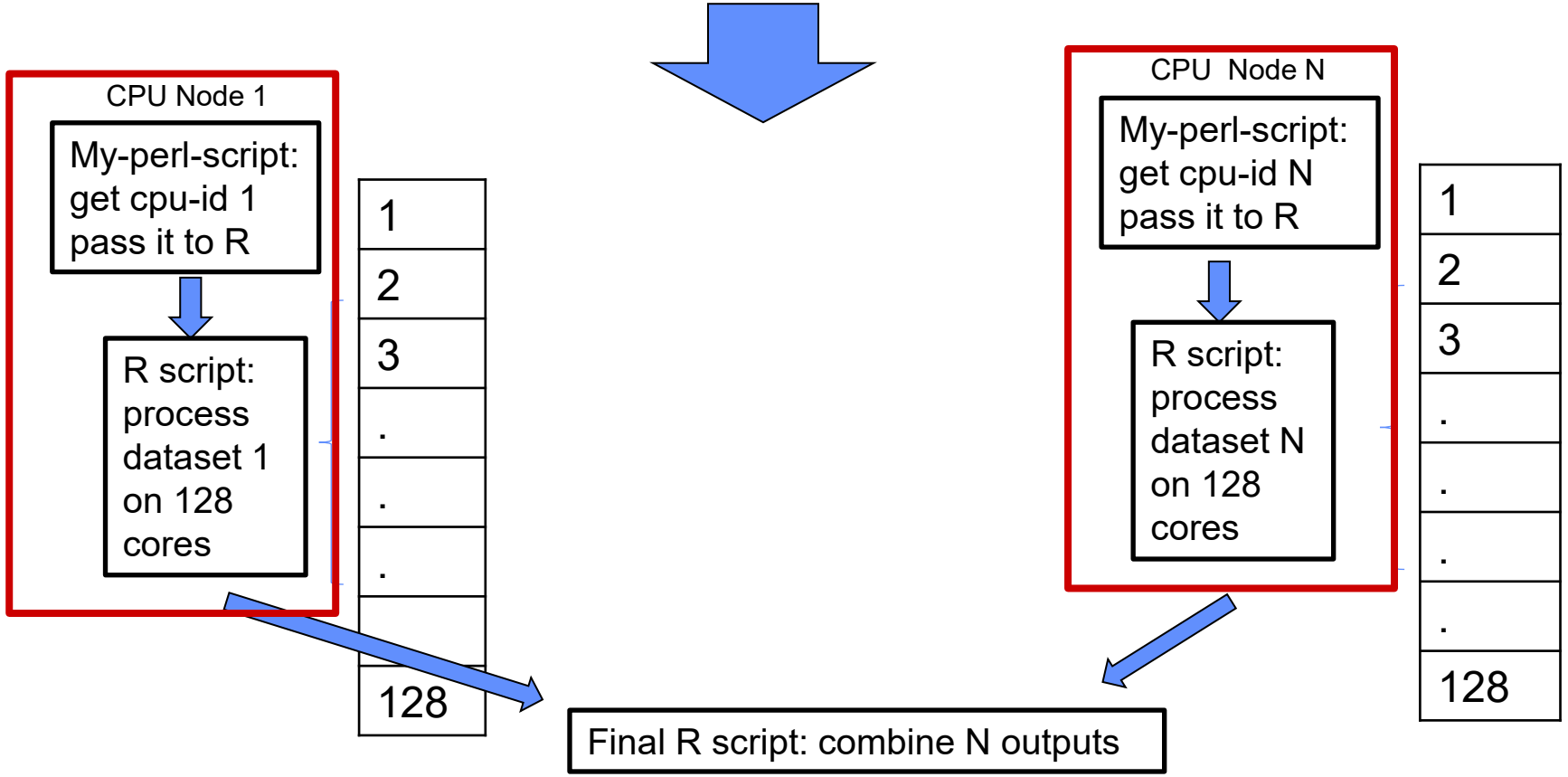
2 x1 = 2 mpi ranks

2 perl script/R instances
128 cores each
(doParallel can use them)

Example: One R instance per node, doParallel across all cores in each node

In slurm batch script:

mpirun … my-perl-script



- **Some example parallelizations that launch R instances onto CPU cores**

Example 1: scaling MCMC

Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models,
F. Bumbaca, UCIrvine

- Probabilities of individual's web activity and clicks
- Used *mpirun* launch parallelization for MCMC model on each individual
- Then combined results via a hierarchical model
  (*rhierMnlRwMixturefunction* in the R package *bayesm*)

Example 2: gene expression prediction models

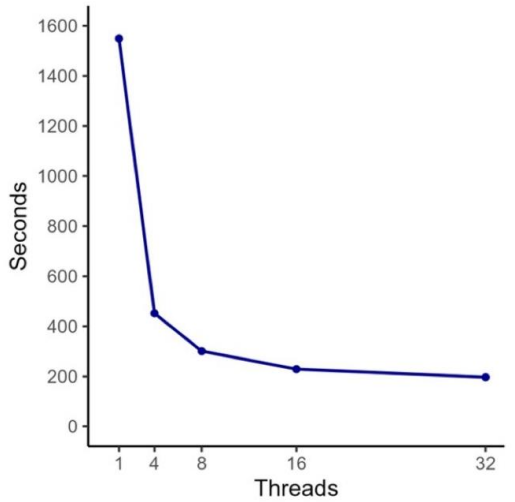
MAGEPRO ***https://github.com/kaiakamatsu/MAGEPRO***

Kai Akamatsu, *Amariuta Lab, UCSD Medicine*

https://www.medrxiv.org/content/10.1101/2024.09.25.24314410v1.full.pdf

- Goal: Enable the identification of novel gene-trait associations through transcriptome-wide association studies.

- Created a pipeline for each gene
  - *a) eQTL statistics (plink)*
  - *b) gene expression heritability*
  - *c) Lasso based prediction model*
- Ran 22 batch jobs, 1 per chromosome
- Within each batch job, used GNU tool ‘parallel’ to run steps with 1 core per gene



Launching Independent R sessions

- **Placing *independent* R sessions onto cores is more flexible for:**
  - data management
  - large number of separate models
  - large variation in time per model
  - hybrid multimode/multicore scripts

But perhaps requires more programming or pre/postprocessing then ‘doParallel’

A note on installing R Packages (into your own directories)

- **In R (might help to be on interactive node):**

  install.packages('package-name')

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

- **Sometimes you have 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
  Packages are put into your /home/user/R directory

Installing R Packages: Conda example

- **Conda is a general package manager**
- **If install.packages not working in R, Conda and a new base R might work better.**
- **Example: from unix prompt on Expanse node:**

```
module load anaconda3
conda  create -n myrenv          #create your conda environment
conda activate myrenv
conda update -n base -c defaults conda
conda install r-base=4.3.1       #install R and your packages in your environment
conda install r-biomod2
```

End