

# CIML

*R and Scaling on HPC*

*Paul Rodriguez*

# R, Scaling R, Parallel R

- A Glimpse of R (recap)
- R and Scaling
- Parallel options for R
- doParallel demo on Expanse portal

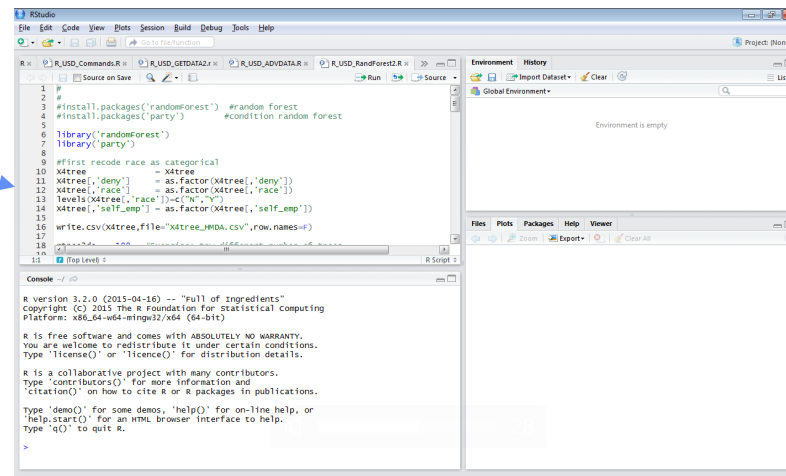
# A typical R development workflow

- R studio: An Integrated development environment for R

*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 mortgage 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, multinode, 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

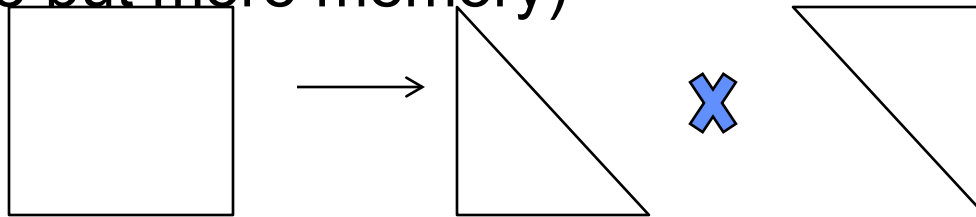
- **Linear Model:**  $Y = X * B$   
where  $Y$ =outcomes ,  $X$ =data matrix

# 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,  
LASSO version from Hastie et al.,etc)

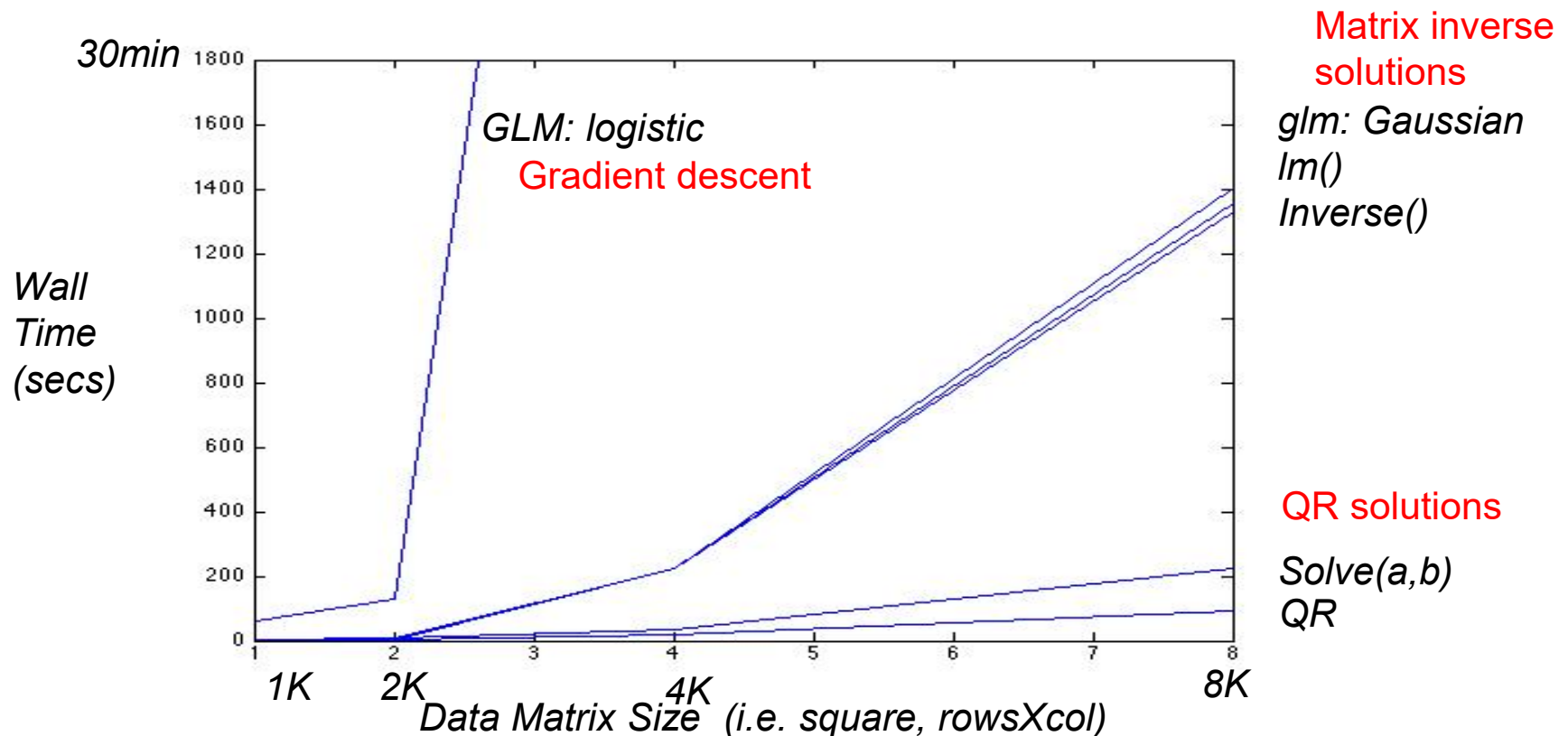
# Solving Linear Systems

## Performance with R, 1 compute node

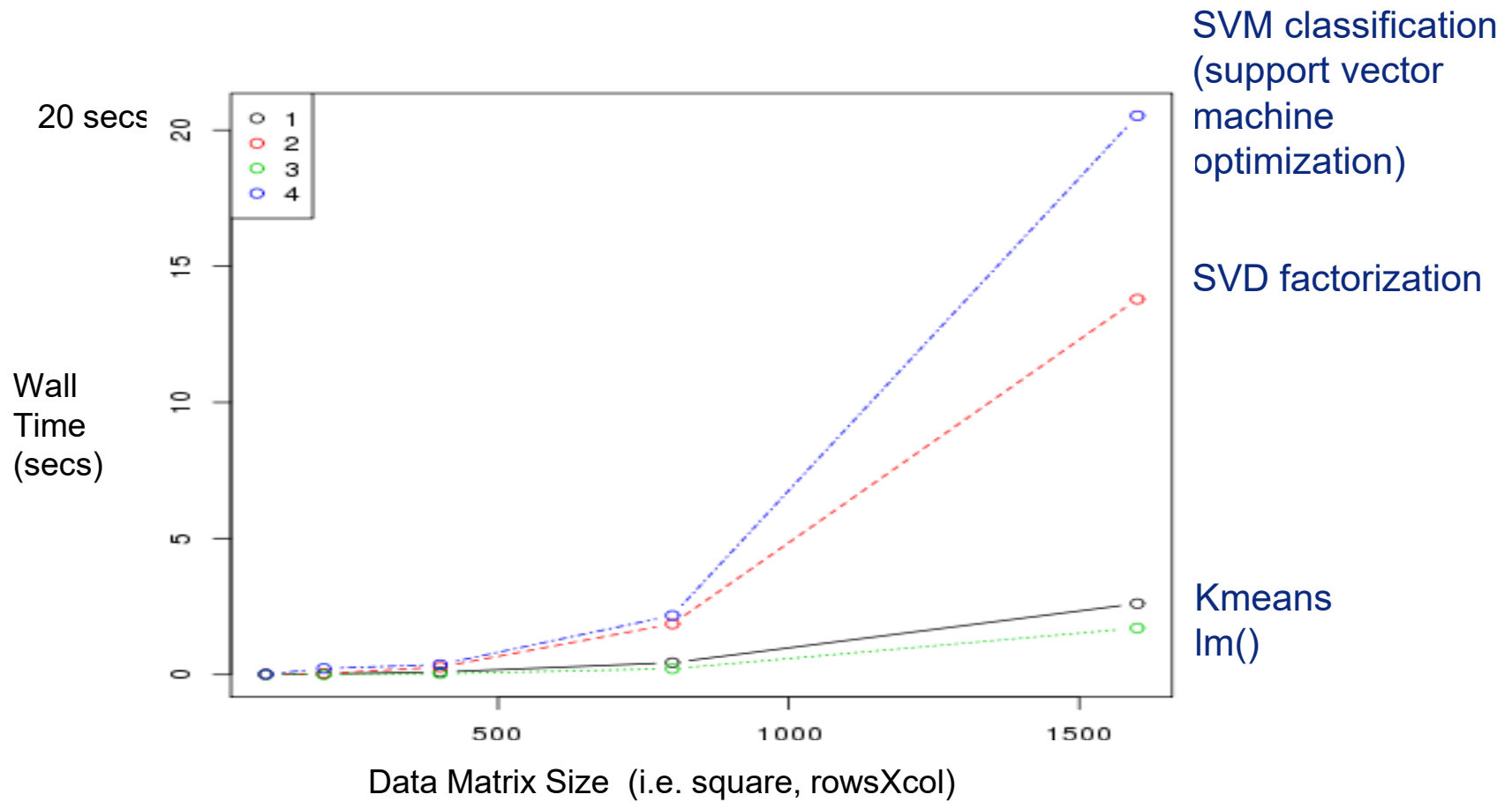
R:

`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 it 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)
```

**1. allocate workers**



```
my_data_frame = .....  
my_results = foreach(
```

**2. Make 'foreach' loop**



# R multicore coding

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


**1. allocate workers**



```
my_data_frame = ..... 2. Make 'foreach' loop
```

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

**3. specify how to  
combine results**



# R multicore coding

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


**1. allocate workers**




```
my_data_frame = ..... 2. Make 'foreach' loop
```

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

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



**3. specify how to  
combine results**



# R multicore coding


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

**1. allocate workers**




```
my_data_frame = ..... 2. Make 'foreach' loop
```

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



```
my_results = foreach(i=1:24,.combine=rbind) %dopar%  
{ ...  
  your code here  
  return( a variable or object )  
})
```

**3. specify how to combine results**



# R multicore coding

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

**1. allocate workers**

```
my_data_frame = .....
```

**2. Make 'foreach' loop**

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

```
{ ...
```

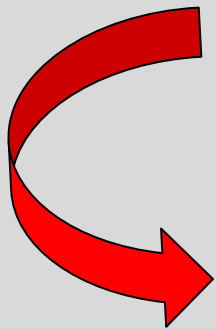
```
  your code here
```

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

```
}
```

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

**3. specify how to combine results**



**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)
```


**1. allocate cluster as  
parallel backend**



```
my_data_frame = .....
```

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

**2.  
%dopar% puts  
loops across  
cores and  
nodes**



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

# R multinode: parallel backend

```
library(doParallel)
```

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


**1. allocate cluster as  
parallel backend**



```
my_data_frame = .....
```


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

**2.  
%dopar% puts  
loops across  
cores and  
nodes**



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

```
})  
stopCluster(cl)
```

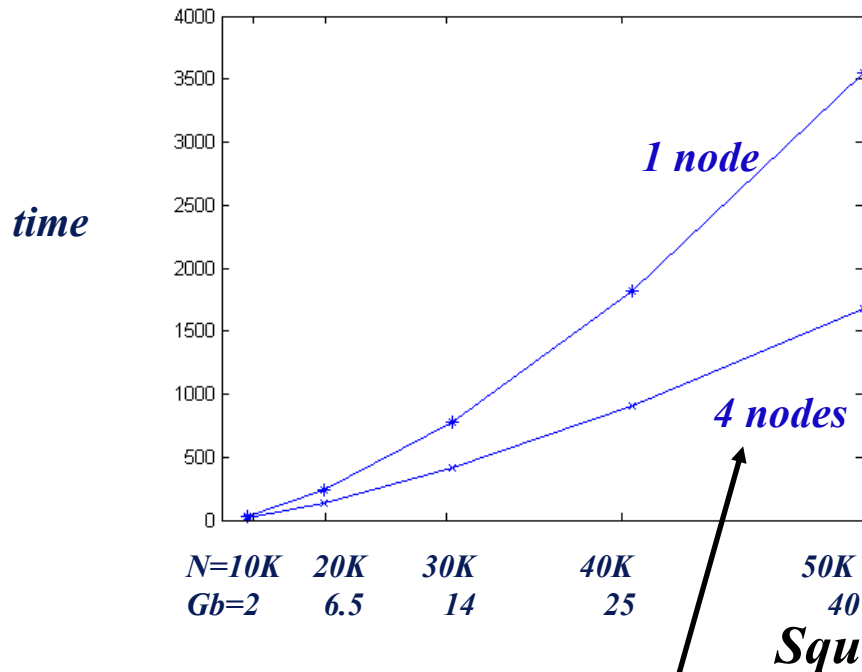


**BEWARE: foreach will copy data it thinks is need to every core and node**

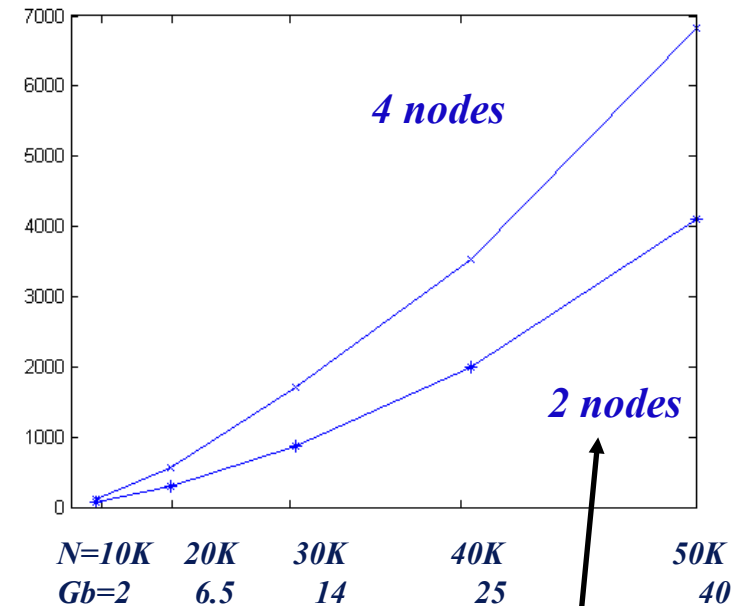
# Multiple Compute Nodes not always help

(tested on Gordon)

*Matrix Multiplication*



*Matrix Inversion*



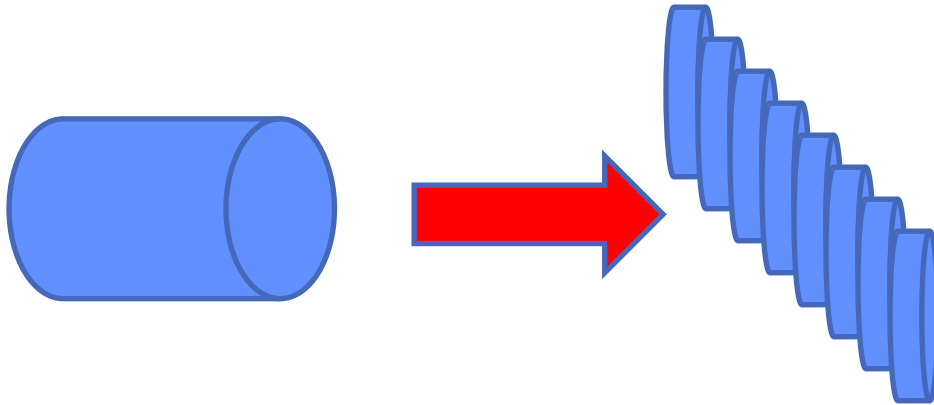
*multinodes: more nodes is less time for multiplication,*

*less nodes is better for inversion*



# An option for (embarrassingly) Parallel R

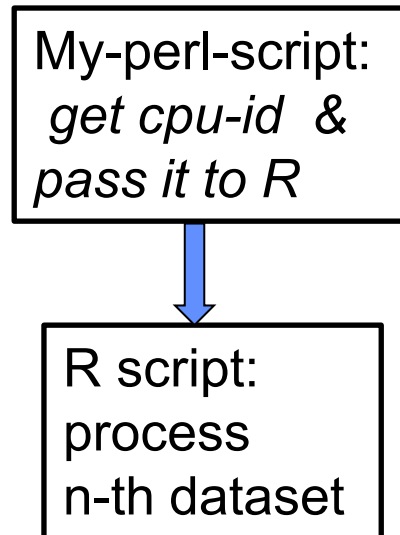
1. Split up  
data into N  
parts



# An option for (embarrassingly) Parallel R

1. Split up  
data into N  
parts

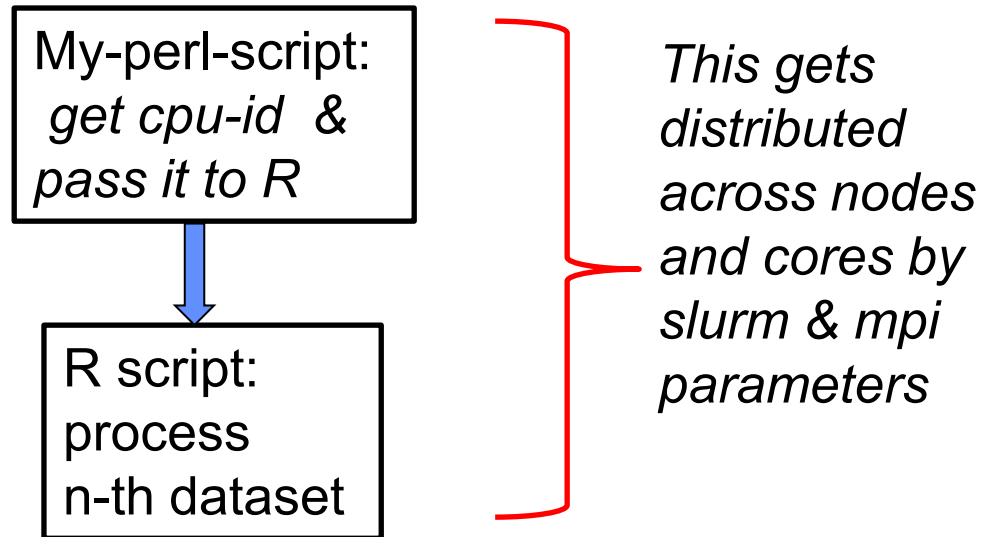
2. In slurm batch script:  
`mpirun ... my-perl-script`



# An option for (embarrassingly) Parallel R

1. Split up  
data into N  
parts

2. In slurm batch script:  
`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
```

2 x 128 = 256 mpi ranks

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

256 perl script/R instances  
1 core each

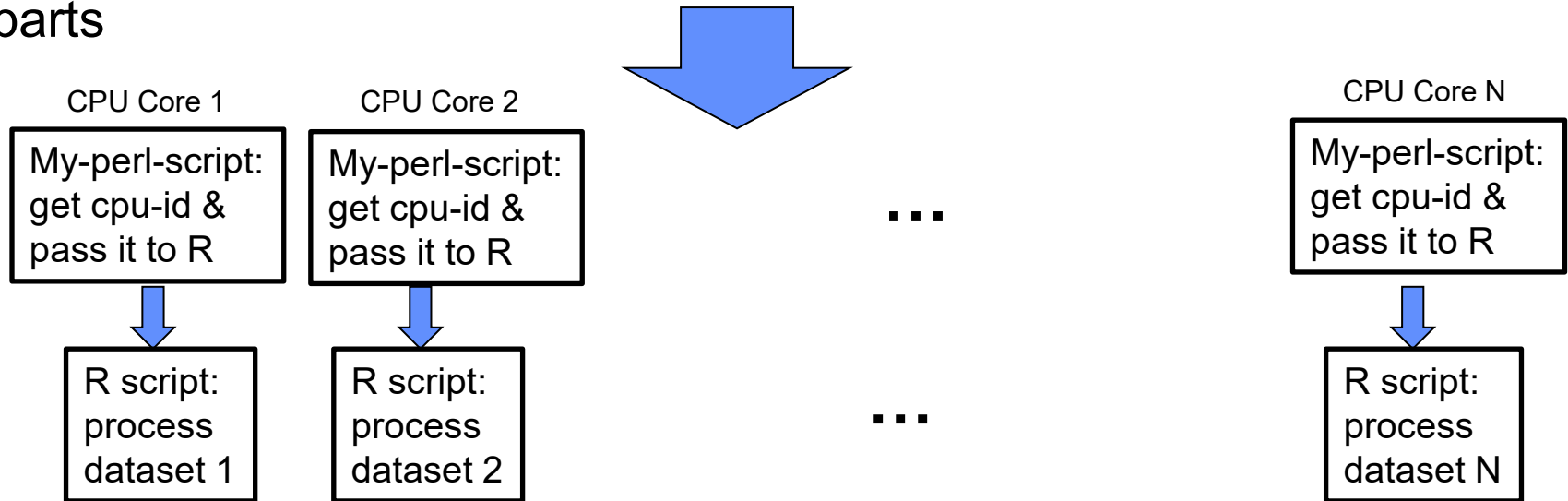
```
mpirun -genv I_MPI_PIN_DOMAIN=omp:compact ./get_mpirank_runcmd.pl
```

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

# one R instance per core across all nodes

1. Split up  
data into N  
parts

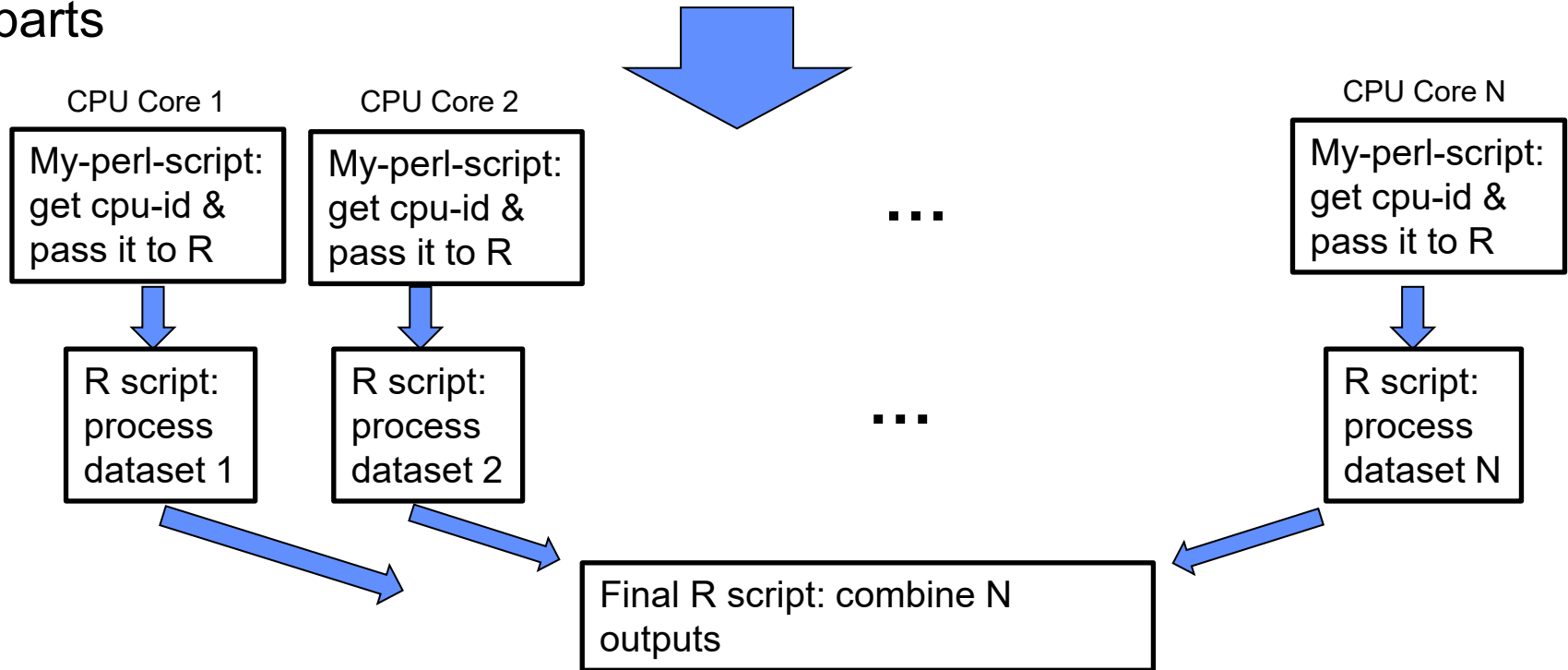
2. In slurm batch script:  
mpirun ... my-perl-script



# one R instance per core across all nodes

1. Split up data into N parts

2. In slurm batch script:  
mpirun ... my-perl-script



*More programming but perhaps more useful*

# Slurm parameters: one R instance per nodes 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
```

2 x 1 = 2 mpi ranks

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

```
module load r
```

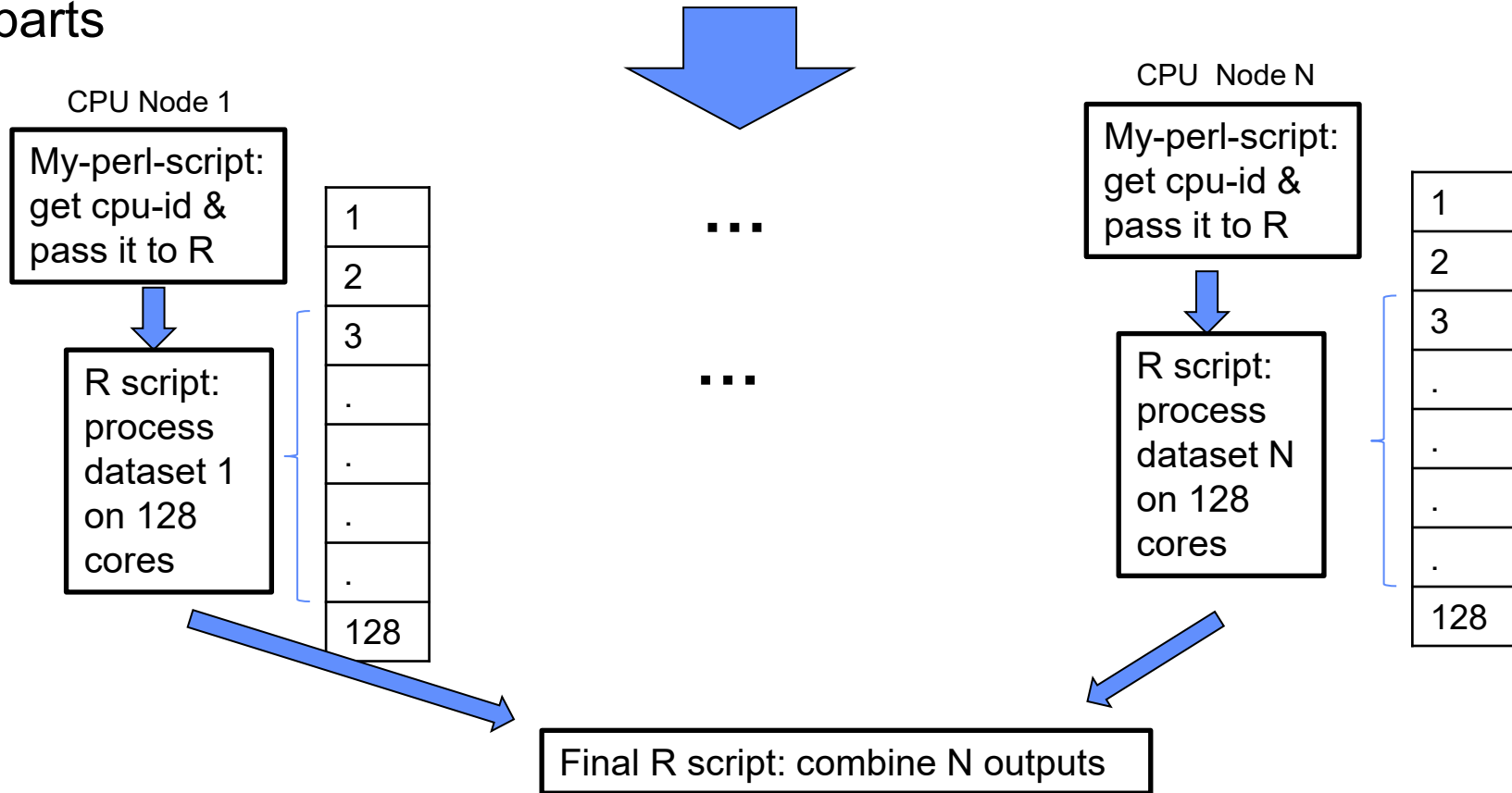
2 perl script/R instances  
128 cores each  
(doParallel can use them)

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

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

1. Split up data into N parts

2. In slurm batch script:  
mpirun ... my-perl-script





# Example: scaling MCMC

*Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models,  
Frederico Bumbaca, UC Irvine,*

- *MCMC searches parameter space through long sequences of updates to get complete probability distribution*
- *Individual sequences are not parallelizable, but sometimes parameters can be ‘partitioned’*
- *Used the R package, “bayesm”*
- *Ran on SDSC Comet with embarrassing parallelization*

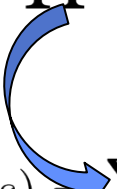
# Individuals	Cores	Individ per Core	Total Minutes (I/O time)
100 million	1,7282 (max)	~ 58K	206 (38)

# Another MCMC option: Stan program

R or Python interface, with many options

Stan script → translated into C++ code and compiled

If you take log likelihood:

$$P(data, parameters) = \prod P(data_i, parameters)$$

$$\log P(data, parameters) = \sum_i^N \log P(data_i, parameters)$$

then Stan will partition the data across cores.  
(  $P()$  is calculated 1000's of times in a sequential chain)

# Other R packages:

**Also, for big data or big matrix**

- **Rspark** - R interface to Spark (upcoming session)
- **pdbR** - distributed matrix support (better for dense matrices vs Spark)

**Also:**

- **R openMP, Rmpi** –
- **Ff, bigmemory** – map data to files
- **Rgputools** – GPU support

# How to use R directly on Expanse

1. Get an interactive compute node:

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.

>

```
[p4rodrig@login02 ~]$ module spider r
-----
r: r/4.0.2-openblas
-----

Other possible modules matches:
  AMDuProf, amber, aria2, arm-forge, berkeley-db, bism

You will need to load all module(s) on any one of the li
"r/4.0.2-openblas" module is available to load.

  cpu/0.15.4  gcc/9.2.0

Help:
```

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

- In R:

*install.packages('package-name')*

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

- Sometimes on Comet, you had 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

- **pause**

# TestdoParallel R script

1 start Rstudio from portal and run script  
(it repeatedly does a regression)

2 review execution using 'top' utility

3 vary the NxP matrix size or number of cores

Goal: look for tradeoffs in memory vs execution  
e.g. If N gets too large then use less cores

SDSC

The Expance portal provides an interface to access Expance HF

With the portal, researchers can manage files, edit, move, upload, and download, vi

se.sdsc.edu/pun/sys/dashboard/batch\_connect/sys/rstudio/session\_contexts/new

Interactive Apps

GUIs

MATLAB

RSTUDIO

RSTUDIO

This app will launch a RSTUDIO GUI on Expance. You will be able to interact with the RSTUDIO GUI through a VNC session.

Partition

compute

Number of hours

Open portal ->  
Interactive Apps ->  
Rstudio

Enter  
Node: "compute"  
Cores: "64"  
(other fields defaults ok)

Also login:  
login.expance

\$ queue -u \$USER  
\$ ssh exp-##-##  
\$ top -u \$USER

'H' will toggle threads

'fP' will toggle last cpuid

```
-----
Last login: Fri Jun 4 15:01:29 2021 from 71.128.8.73
[p4rodrig@login02 ~]$ queue -u p4rodrig
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
3246260 compute Sys/dash p4rodrig R 0:27 1 exp-2-15
[p4rodrig@login02 ~]$ ssh exp-2-15
Last login: Sat Jun 5 13:09:04 2021
[p4rodrig@exp-2-15 ~]$ top -u $USER
```



Open the 'Test\_doParallel' Rscript

Select 'source' to run the whole script

Apps Assess your Mental... Become Acquaint... Opinion | Quiz: Let... 2017 CCES Commo... tds An Overview of Res... » Reading list

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function Addins Project: (None)

TestDoParallel\_v1.R x

Source on Save Run Source

```
1 #Install package if needed into home directory
2 if(!require(doParallel)){
3   install.packages("doParallel", repos="http://cran.r-project.org")
4 }
5
6 library(doParallel)
7
```

7:28 (Top Level) R Script

Environment History Connections Tutori

R Global Environment

Data

B	num [1:200, 1] 5 1.03 0...
X	Large matrix (200000 el...

Files Plots Packages Help Viewer

New Folder Delete Rename More

Home

Name	Size
.r	
chap15_rnnv4.ipynb	136 KB
e494_Practical_NNet_exercise...	18.5 KB
environment_tests_v1	1.8 KB
getcpu_scratch.sh	274 B
getid.c	324 B

Console Terminal x Jobs x

~/

```
>
>
>
> source('~/.TestDoParallel_v1.R')
[1] "starting dopar test"
[1] "X size is: 1.5 Mb"
```

```
, load average: 0.00, 0.03, 0.26
eeping, 0 stopped, 0 zombie
id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
15384.5 used, 2560.7 buff/cache
0.0 used, 239374.2 avail Mem
```

SHR	S	%CPU	%MEM	TIME+	COMMAND
3592	S	1.0	0.1	0:16.39	rstudio
3760	R	1.0	0.0	0:01.34	top
0676	S	0.7	0.0	0:21.31	Xvnc
9828	S	0.7	0.1	0:06.58	Compositor
9828	S	0.3	0.1	0:19.86	QtWebEngineProc
9828	S	0.3	0.1	0:01.71	Chrome_ChildIOT
3492	S	0.3	0.1	0:00.04	rsession
3492	S	0.3	0.1	0:00.03	rsession
3492	S	0.3	0.1	0:00.04	rsession
3492	S	0.3	0.1	0:00.03	rsession
3492	S	0.3	0.1	0:00.04	rsession
3484	S	0.3	0.1	0:00.03	rsession
8180	S	0.0	0.0	0:00.03	systemd
0	S	0.0	0.0	0:00.00	(sd-pam
2852	S	0.0	0.0	0:00.00	slurm s
0676	S	0.0	0.0	0:00.37	Xvnc
0676	S	0.0	0.0	0:00.19	Xvnc
0676	S	0.0	0.0	0:00.11	Xvnc

*Review the top output*

*Notice the elapsed time and memory size*

*Change the NxP matrix size and rerun*

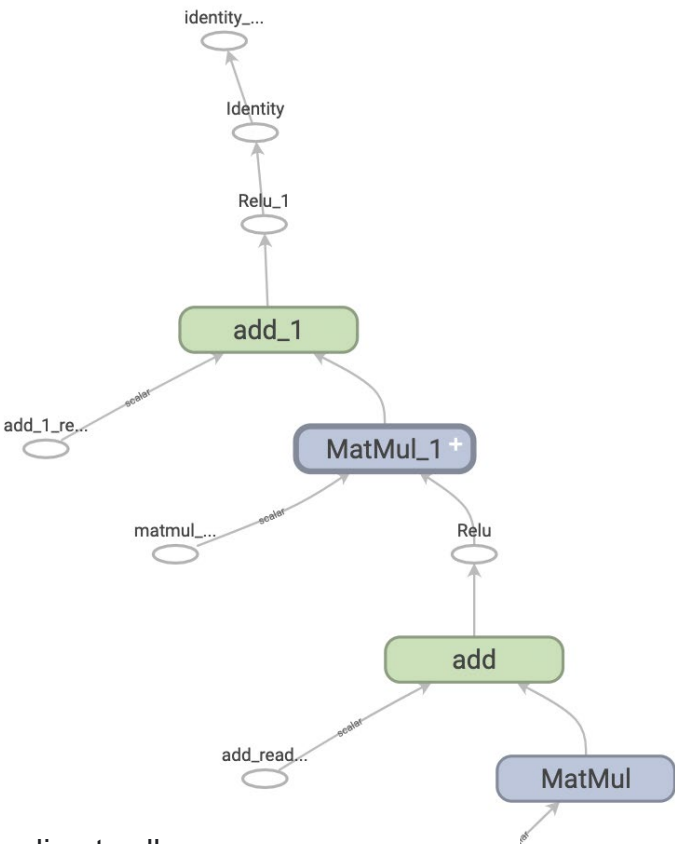
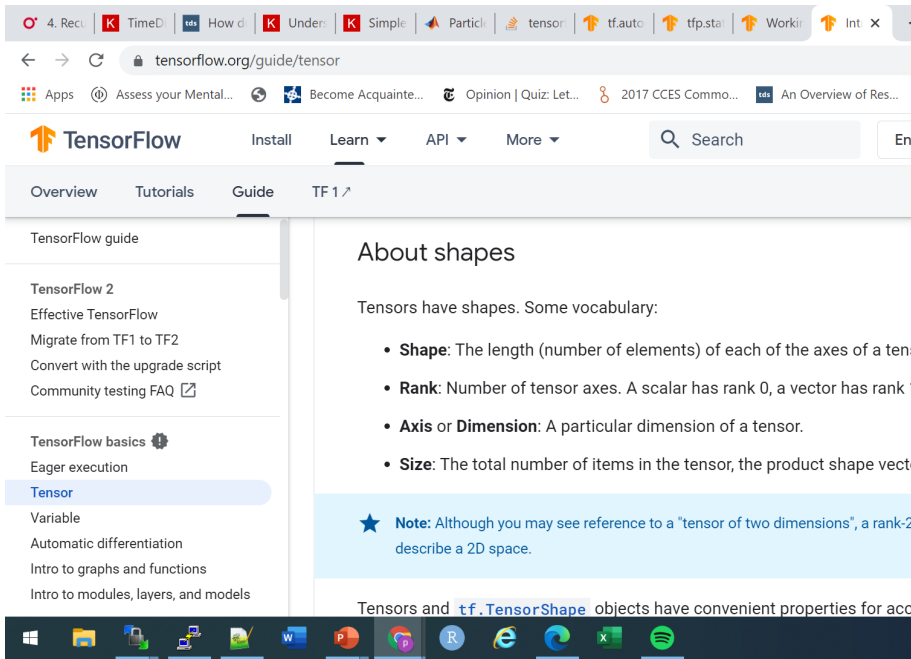
```
6 library(doParallel)
7 (Top Level)
8
9 Console Terminal x Jobs x
10 ~/
11 >
12 > source('~/.TestDoParallel_v1.R')
13 [1] "starting dopar test"
14 [1] "X size is: 1.5 Mb"
15      user system elapsed
16 1.176   1.969   30.620
17 >
```

```
10 # Make up some random data and lis
11 N=100000;      #N rows start with
12 P=2000;        #P columns 200 for
13
14 #make random data with 1 column ar
15 X =matrix(rnorm(N*P),N,P)
16 X[,1] =X[,1]+1
17
18 16:28 (Top Level)
19
20 Console Terminal x Jobs x
21 ~/
22 Loading required package: foreach
23 Loading required package: iterators
24 Loading required package: parallel
25 [1] "starting dopar test"
26 [1] "X size is: 1.5 Gb"
```

*THE END*

Graphs are data structures that contain a set of [tf.Operation](#) objects, which represent units of computation; and [tf.Tensor](#) objects, which represent the units of data that flow between operations. They are defined in a [tf.Graph](#) context. Since these graphs are data structures, they can be saved, run, and restored all without the original Python code.

This is what a TensorFlow graph representing a two-layer neural network looks like when visualized in TensorBoard.



You create and run a graph in TensorFlow by using [tf.function](#), either as a direct call or as a decorator. [tf.function](#) takes a regular function as input and returns a Function. **A Function is a Python callable that builds TensorFlow graphs from the Python function. You use a Function in the same way as its Python equivalent.**

