Scalable Machine Learning Agenda

1:30 – 3:00 – R in HPC

3:15 - 3:30 – Break

3:15 - 3:40 – Intro to Spark

3:45 - 4:15 – ML with pySpark

4:15 - 4:45 – Spark R

4:45 - 5:00 – Wrap-up

Scaling R in HPC

SDSC SAN DIEGO SUPERCOMPUTER CENTER

UC San Diego

R, Scaling R, Parallel R

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

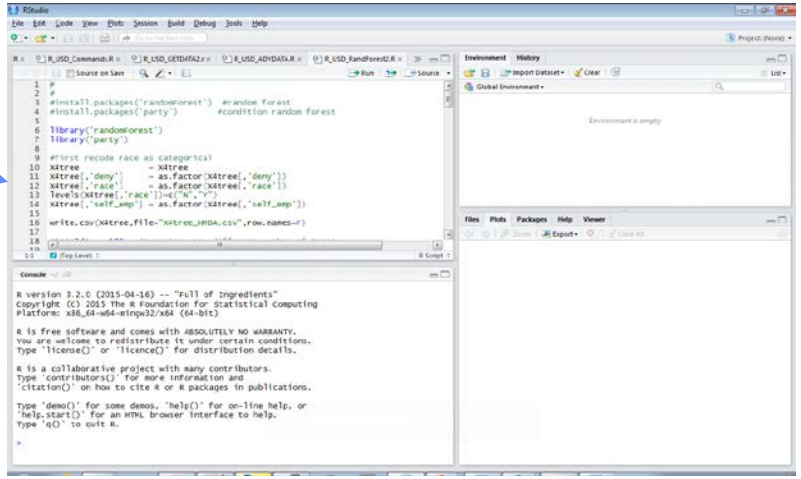
A typical R development workflow

- **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

- **Sampling/bootstrap methods**
- **Data Wrangling**
- **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)
- **Sometimes easy to parallelize (like with sampling)**
- **Sometimes too much communication between parts (matrix inversion)**

R Scaling In a nutshell

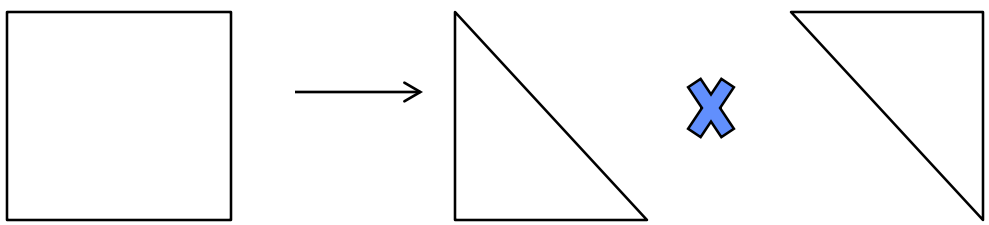
- **R takes advantage of math libraries for vector operations**
- **R packages provide multicore, multimode, or distributed data (SparkR) options**
- **However, model implementations not necessarily built to use parallel backends**
  - Some models more amenable to parallel versions

Consider Regression Computations

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

  where Y=outcomes , X=data matrix

- **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

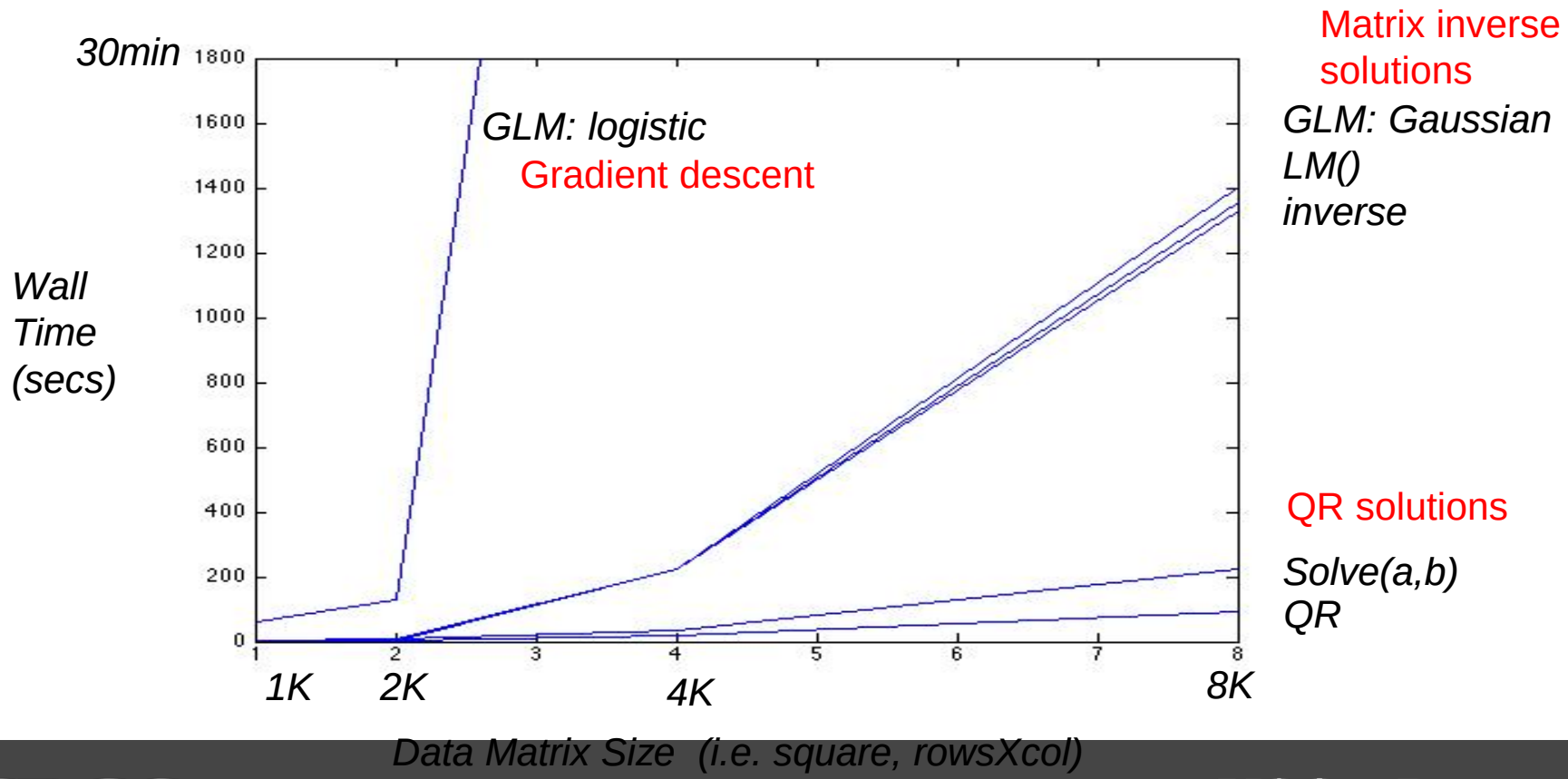
  glm() #Generalized Linear Model
  (logistic regression, etc)

  aov() #Analysis of Variance
  (returns ANOVA table of F-scores)

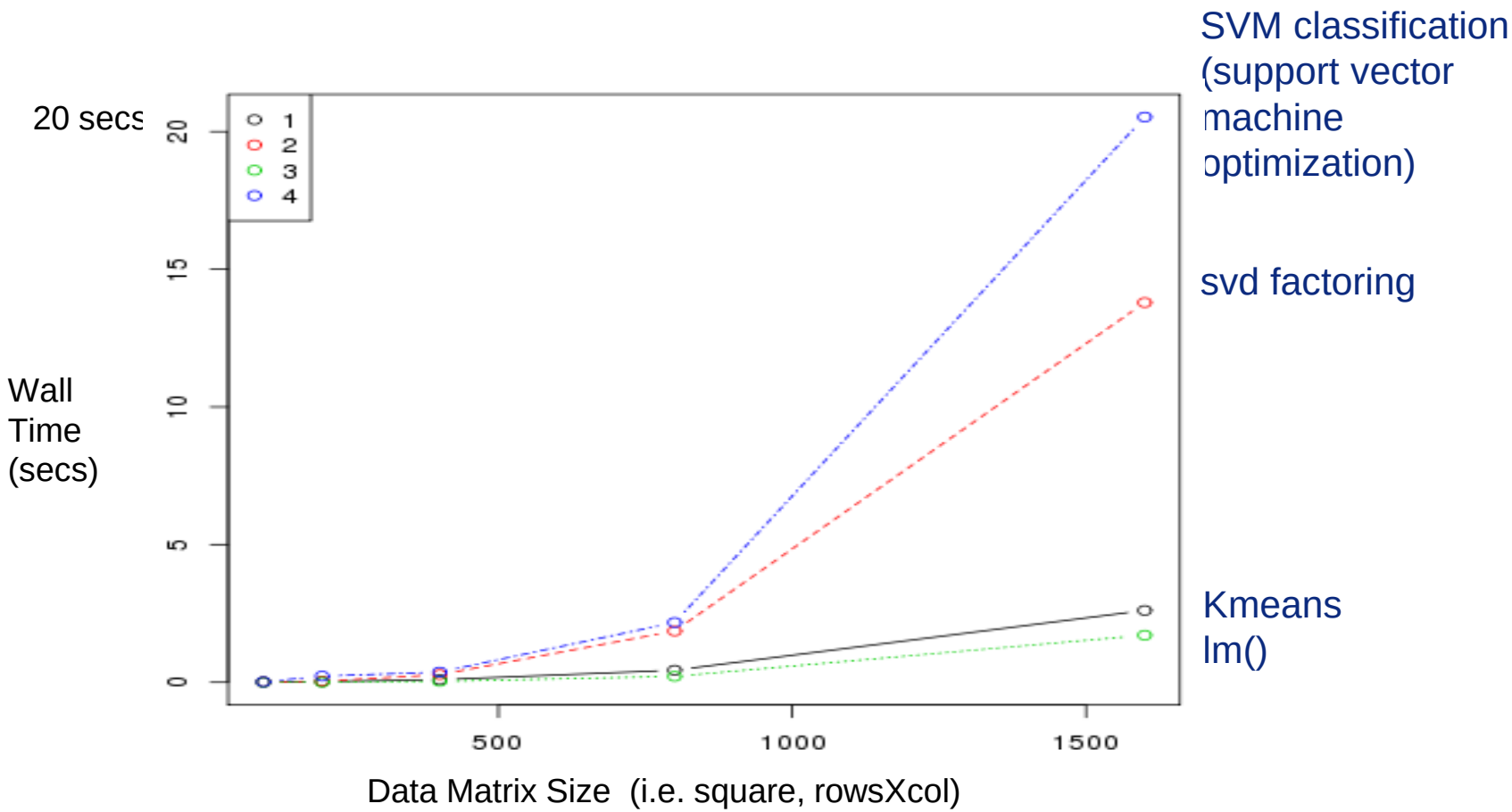
All these work on system of equations

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

- ‘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 multimode.

R multicore

- **Run loop iterations on separate cores**

```
install.packages(doMC)
library(doMC)
registerDoMC(cores=24)
getDoParWorkers()

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

allocate workers

%dopar% puts loops across cores, (loops are independent) %do% runs it serially

returned items 'combined' into list by default

specify to combine results into array with row bind

R multinode: parallel backend

- **Run loop iterations on separate nodes**

```
install.packages('doSNOW')
library('doSNOW')
…
cl <- makeCluster( mpi.universe.size()-1, type='MPI' )
clusterExport(cl,c('data'))
registerDoSNOW(cl)

results = foreach(i=1:47,.combine=rbind) %dopar%
  {   … your code here


      return(  a variable or object )
})
stopCluster(cl)
mpi.exit()
```

allocate cluster as parallel backend

%dopar% puts loops across cores and nodes

R multicore

- **Run loop iterations on separate cores**

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

allocate workers

R multicore

- **Run loop iterations on separate cores**

```
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 )
      })
```

allocate workers

%dopar% puts loops across cores,
(loops are independent)
%do% runs it serially

R multicore

- **Run loop iterations on separate cores**

```
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 )
      })
```

allocate workers

%dopar% puts loops across cores,
(loops are independent)
%do% runs it serially

returned items 'combined' into list by default

specify to combine results into array with row bind

R multicore

- **Run loop iterations on separate cores**

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

```
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 )
})
```

allocate workers

%dopar% puts loops across cores, (loops are independent) %do% runs it serially

returned items 'combined' [illegible] by default

specify to combine results into array with row bind

R multinode: parallel backend

- **Run loop iterations on separate nodes**

Using R mpi functions

allocate cluster as parallel backend

%dopar% puts loops across cores and nodes

```
library(Rmpi)

…
cl <- makeCluster( mpi.universe.size()-1, type='MPI' )
clusterExport(cl,c('data'))
registerDoSNOW(cl)

results = foreach(i=1:47,.combine=rbind) %dopar%
  {   … your code here


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

R multinode: parallel backend

- **Run loop iterations on separate nodes**

BEWARE: foreach will copy data it thinks is need to every node – that can take a long time!

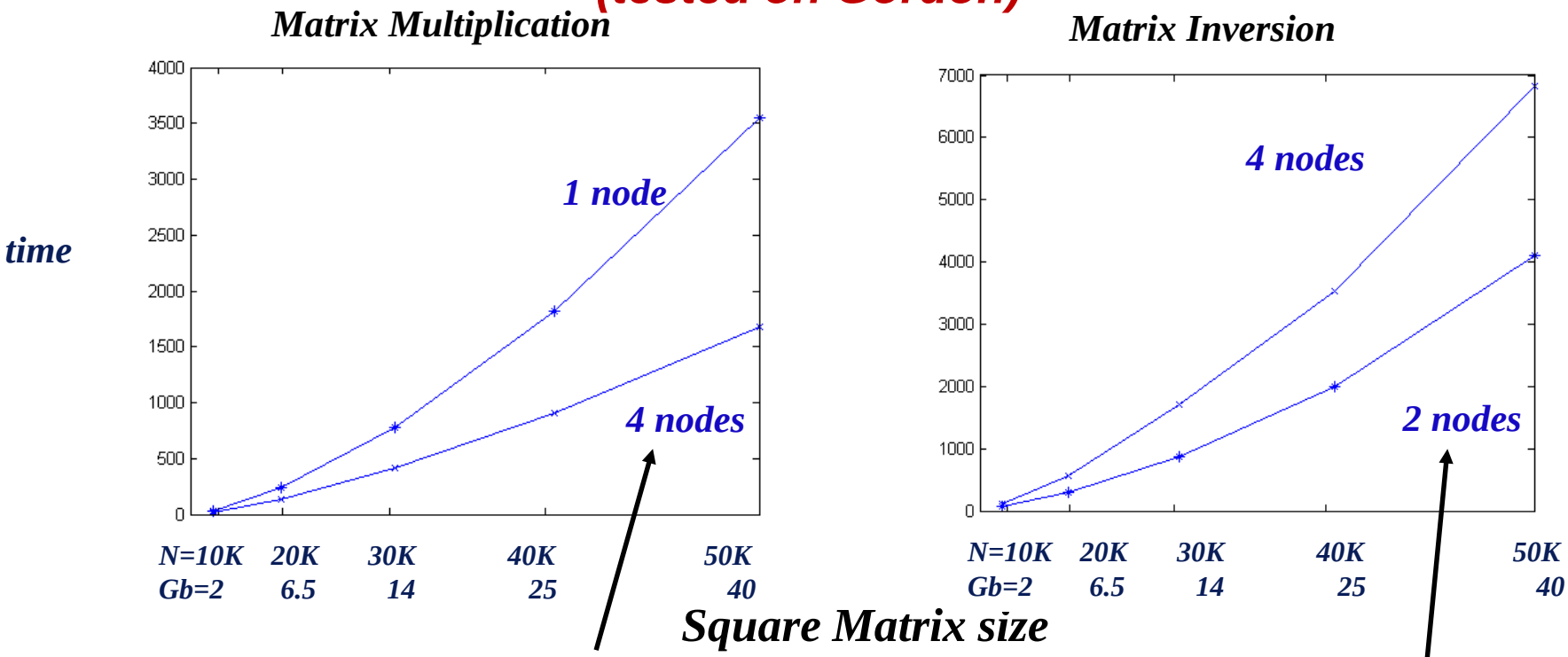
allocate cluster as parallel backend

%dopar% puts loops across cores and nodes

```
library(Rmpi)

…
cl <- makeCluster( mpi.universe.size()-1, type='MPI' )
clusterExport(cl,c('data'))
registerDoSNOW(cl)

results = foreach(i=1:47,.combine=rbind) %dopar%
  {   … your code here


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

Multiple Compute Nodes not always help

(tested on Gordon)

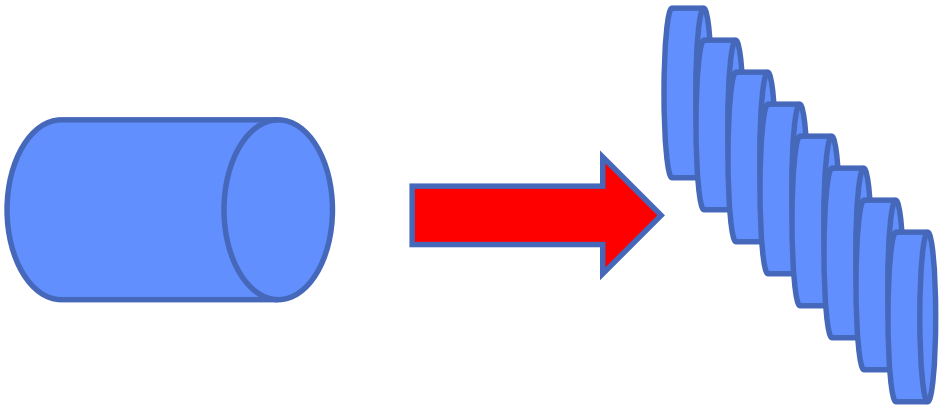


multinodes: more nodes is less time for multiplication,

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
```

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

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

Init MPI and get MPI rank

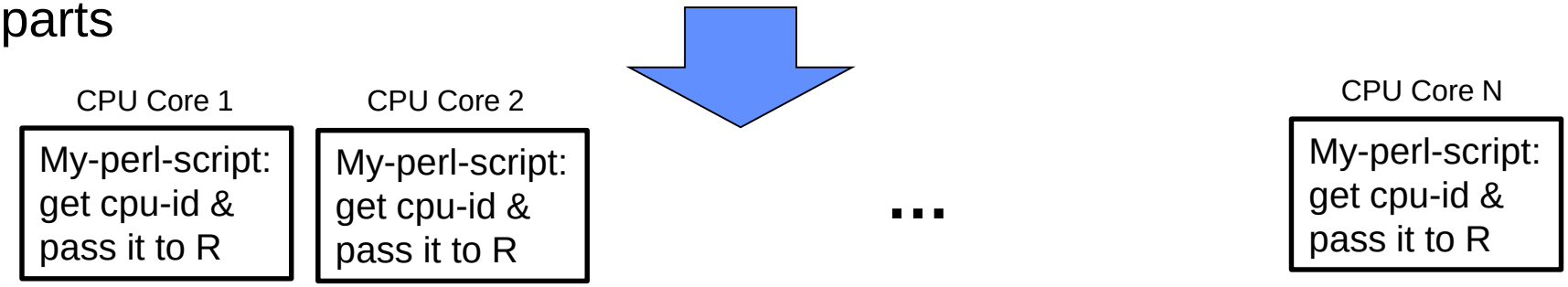
No other MPI calls made

Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

ibrun -np processors My-perl-script

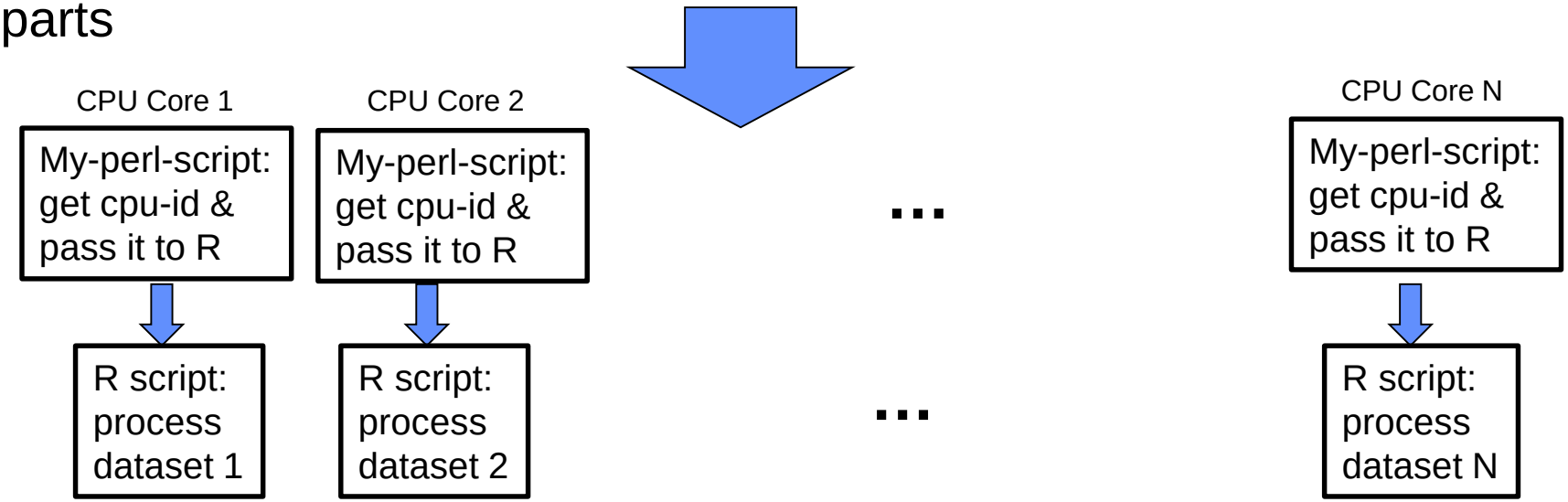


Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

ibrun -np processors My-perl-script

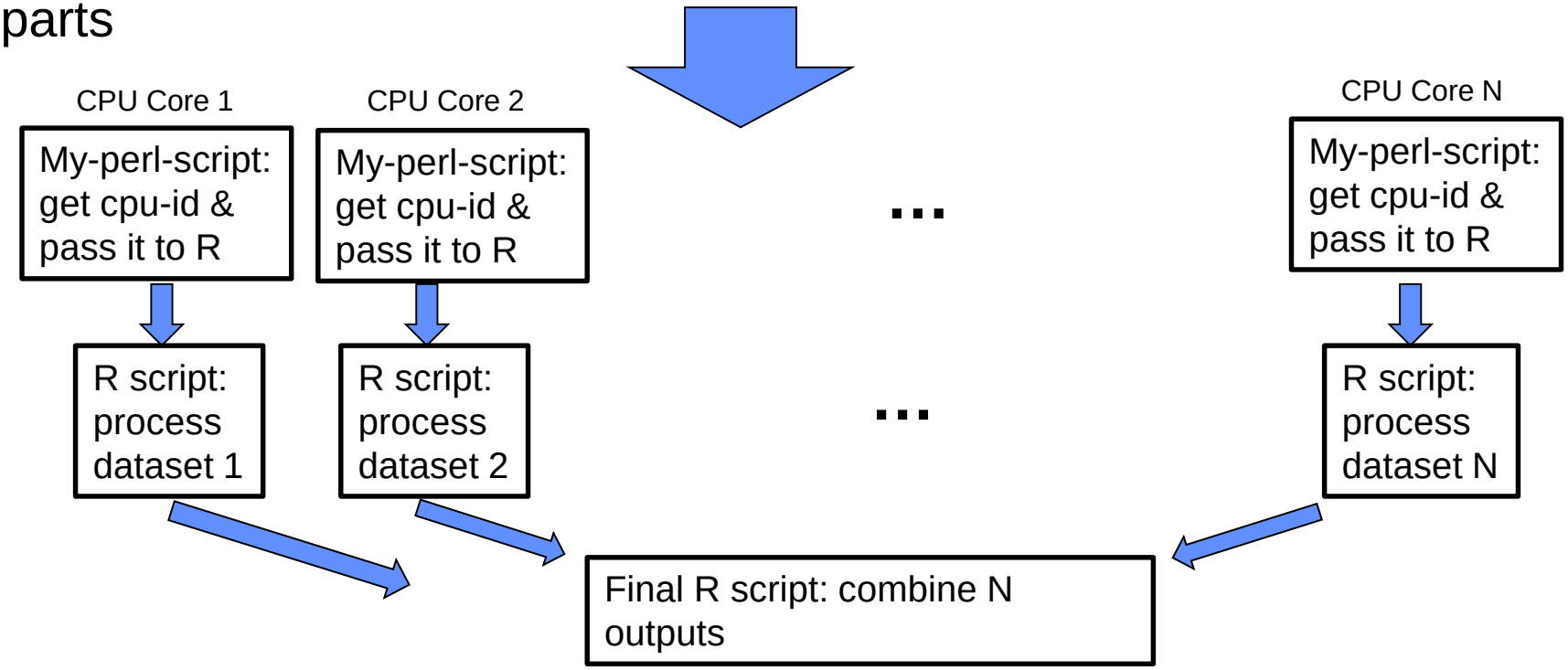


Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

ibrun -np processors My-perl-script

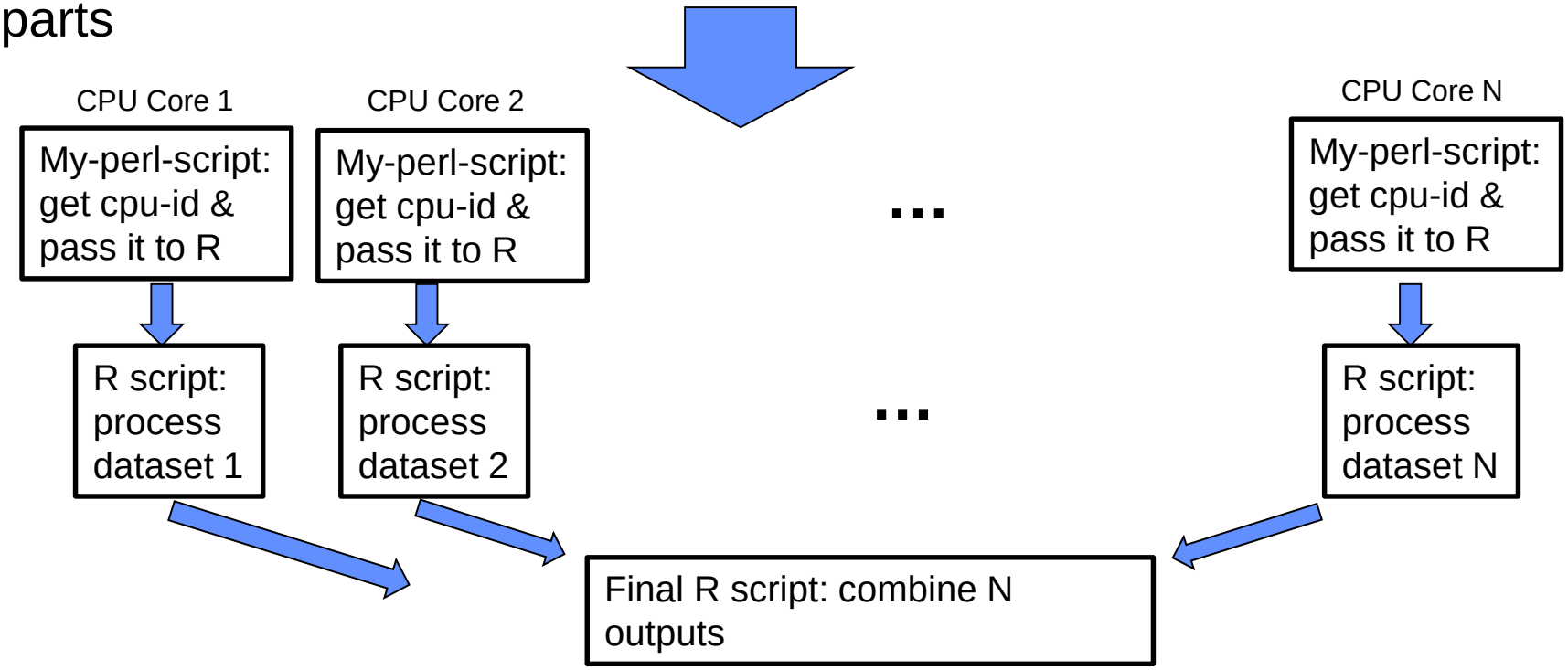


Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

ibrun -np processors My-perl-script



More programming but more flexible

Normal batch job info

```
#!/bin/bash
# ----------------------------
# slurm script for a batch job on comet
# to run a task on individual cores
# ----------------------------
#SBATCH --job-name="packR"
#SBATCH --output="serial-pack.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 1:00:00
#SBATCH -A sds164

bash

#Generate a hostfile from the slurm node list
export SLURM_NODEFILE=`generate_pbs_nodefile`
module load R

#launch 24x2=48 tasks on 48 cores,
# and start this perl script on each task
ibrun --npernode 24 --tpp 1 perl ./bundlerxP.pl

#One can also run hybrid:
#  launch 1 process per node, with 24 threads, and
#  use doParallel
ibrun --npernode 1 --tpp 24 perl ./bundlerxP.pl
```

ibrun the 'bundler' perl script on 24 cores per nodes, and 1 thread each

the 'bundler' Perl script

```
#!/usr/bin/perl
use strict;
use warnings;

my ($myid, $numprocs) = split(/\s+/, `./getid`);

# ------------------------------------
# launch an R session for this task
# ------------------------------------
my $task_index = $myid+1;
`module load R;/opt/R/bin/Rscript Test_PackingR.R $task_index >
                    Rstd_out.$task_index.txt`;
```

the backtick executes system command

Get current cpu id and number of processes

execute R and pass the rank id as an argument

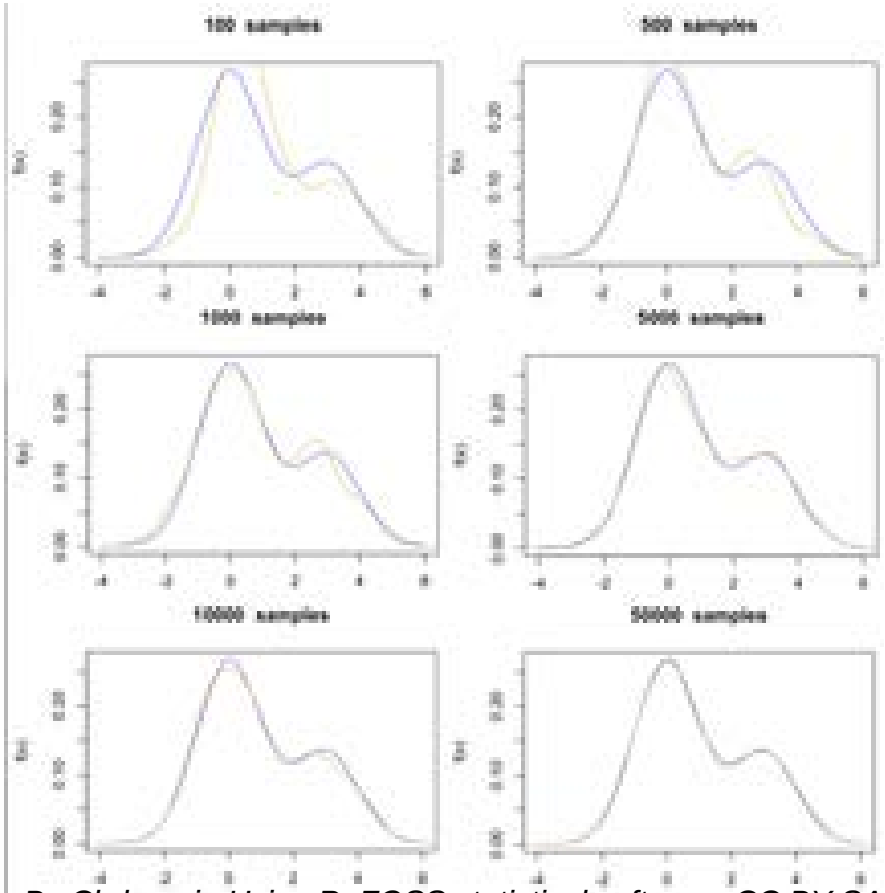
Scaling doParallel vs ‘Packing’ R sessions

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

But requires more programming or preprocessing

Re: Markov Chain Monte Carlo (MCMC)

- complicated probability distributions
- hard to analyze, easier to sample
- ‘search’ out peaks
- eventually, samples converge to the distribution
- not generally parallelizable b/c of interdependencies



By Chdrappi - Using R; FOSS statistical software, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=25674906

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 **'serial packing'** 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 MCMC

Localizing social media hot spots (work in progress with UCIrvine)

- *Individual spatial mixture models for users' geocoded social media use*
- *MCMC search for location probabilities are independent across users, but convergence time varies depending on user variations*
- *Ran on SDSC Comet with **'serial packing'** parallelization, with many cores for short runs, then few cores for longer runs*

(using Rgeoprofile package with MCMC)

# Individuals	Cores	Approx Hours
~3000	192-288	2-3
~2000	48-96	4-8
~100	24	12-24

Example 3: scaling likelihood estimation

Social network evolution (work in progress with UTDallas)

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

- *Ran with **'serial packing'** 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

Installing your own R Packages

- **In R:**

  install.packages('package-name')

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

- **on Comet:**

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

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

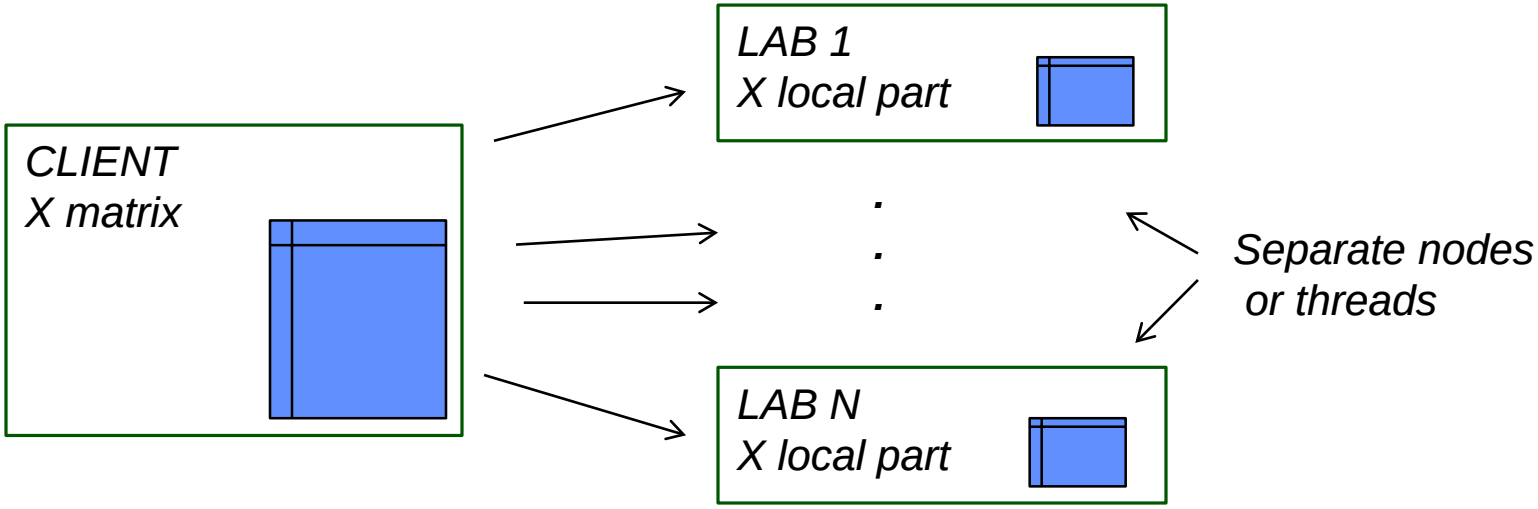
Other R packages:

- **Rspark - R interface to Spark**
- **pdbR - higher level over R-MPI, distributed matrix support and other**

  (better for dense matrices vs Spark)
- **HiPLAR - GPU and multicore for linear algebra**
- **Rgputools – GPU support**
- **R openMP , better data mgt than doParallel**
- **Ff, bigmemory; Revolution Scale R (commercial) – map data to files**

Matlab quickview

- **Distributed Toolbox:**
  - allocate distributed matrices using ‘spmd’ code
  - MPI or threads under the hood
  - You decide data/task set up



pause

R multicore exercise

- **Login to comet**
- **cd SI2018**
- **Get an interactive compute node session**
  - getcpu
- **Start up singularity image:**
  - module load singularity
  - singularity shell /home/train129/keras-tensorflow-cpu.img
  - jupyter notebook --no-browser --ip="*" &

Starting a singularity image and jupyter notebook

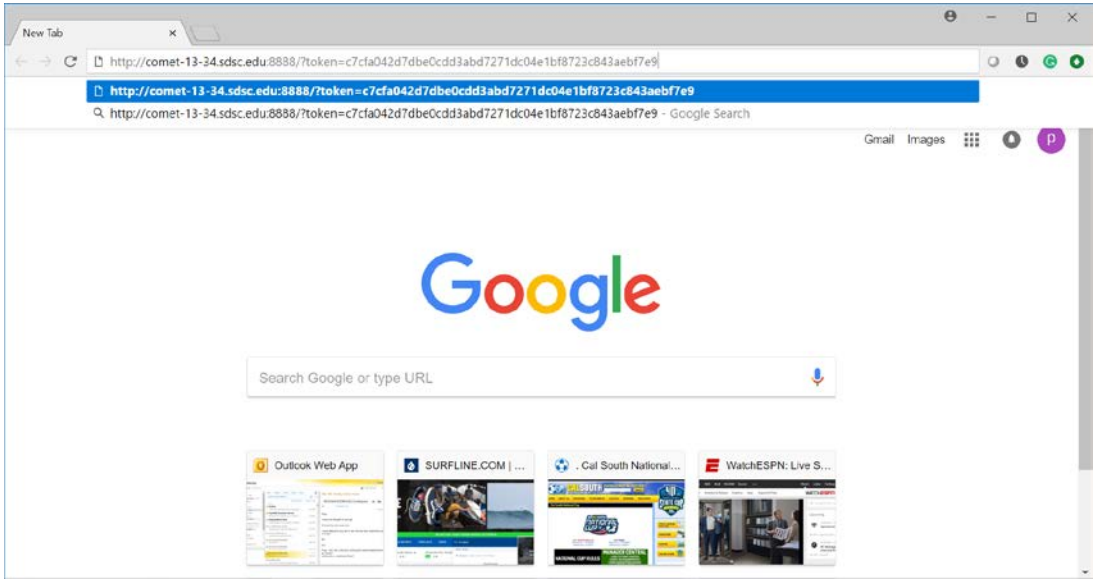
etrain108@comet-ln2:~/SI2018

```
[etrain108@comet-13-34 SI2018]$ module load singularity
[etrain108@comet-13-34 SI2018]$ singularity shell /home/train129/keras-tensorflo
w-cpu.img
Singularity: Invoking an interactive shell within container...

Singularity keras-tensorflow-cpu.img:~/SI2018> jupyter notebook --no-browser --ip="*" &
[1] 3291
Singularity keras-tensorflow-cpu.img:~/SI2018>
```

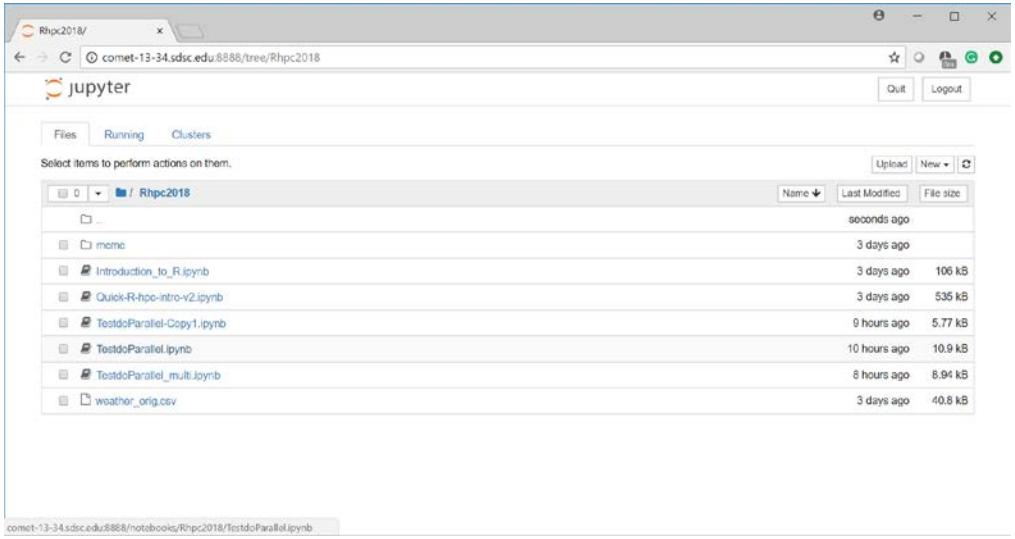
etrain108@comet-ln2:~/SI2018

```
w-cpu.img
Singularity: Invoking an interactive shell within container...

Singularity keras-tensorflow-cpu.img:~/SI2018> jupyter notebook --no-browser --ip="*" &
[1] 3291
Singularity keras-tensorflow-cpu.img:~/SI2018> [W 03:56:34.025 NotebookApp] WARNING: The noteboo
k server is listening on all IP addresses and not using encryption. This is not recommended.
[I 03:56:34.035 NotebookApp] Serving notebooks from local directory: /home/etrain108/SI2018
[I 03:56:34.035 NotebookApp] The Jupyter Notebook is running at:
[I 03:56:34.035 NotebookApp] http://(comet-13-34.sdsc.edu or 127.0.0.1):8888/?token=c7cfa042d7db
e0cdd3abd7271dc04e1bf8723c843aebf7e9
[I 03:56:34.035 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice
to skip confirmation).
[C 03:56:34.038 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://(comet-13-34.sdsc.edu or 127.0.0.1):8888/?token=c7cfa042d7dbe0cdd3abd7271dc04e1bf
8723c843aebf7e9
```

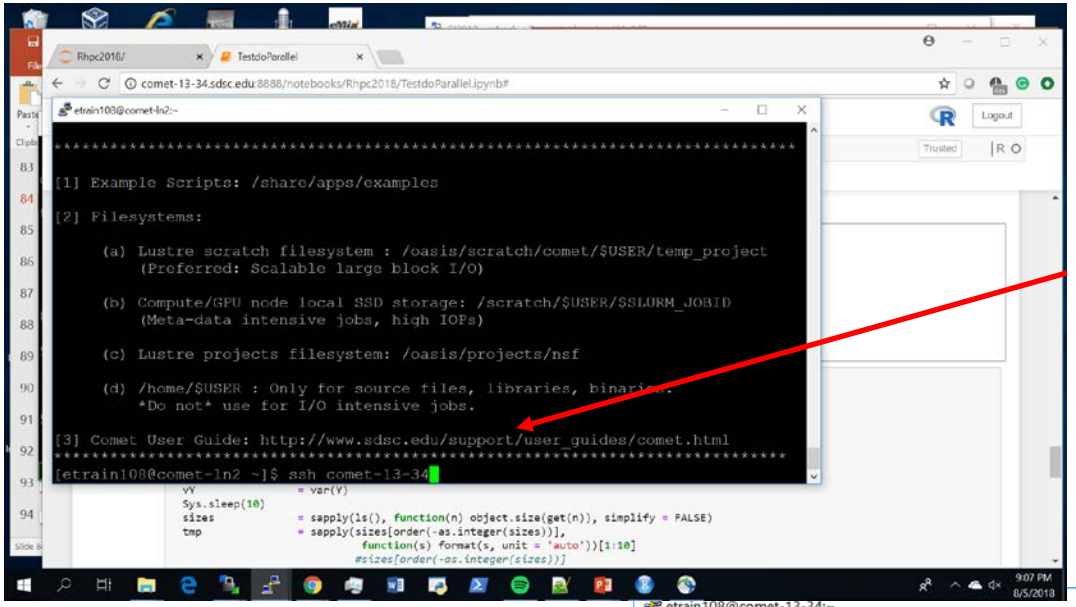
Starting a singularity image and jupyter notebook



Open TestdoParallel.ipynb



Starting an extra terminal window to see execution on the compute node

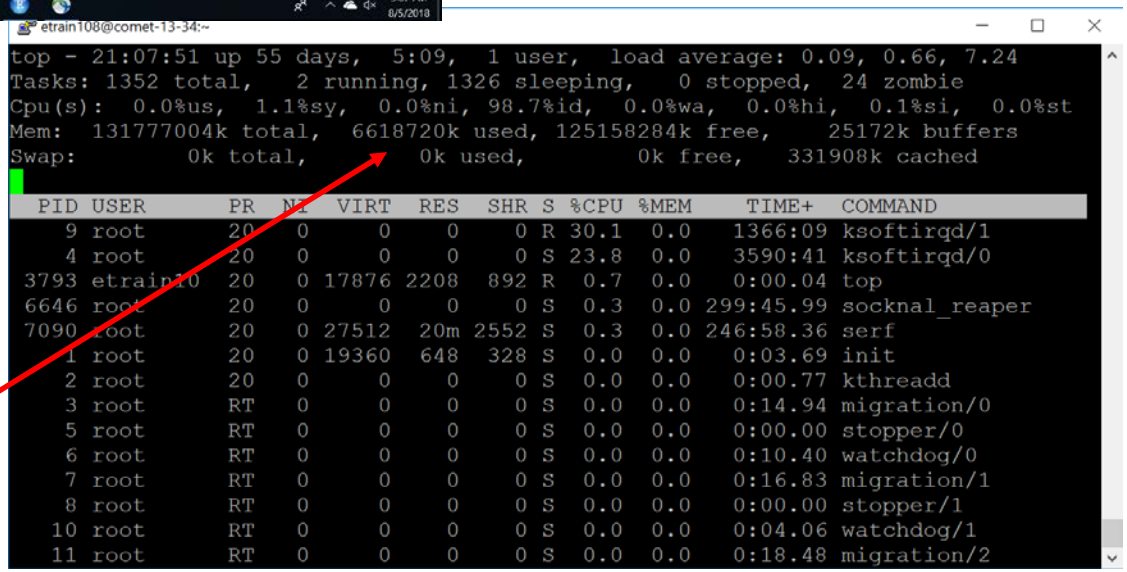


ssh to compute node running the singularity image

Enter

>Top

See memory usage



- **See html version of exercise**

THE END