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

(or subtract 5 hrs for 8:30-12)

Scaling R in HPC

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

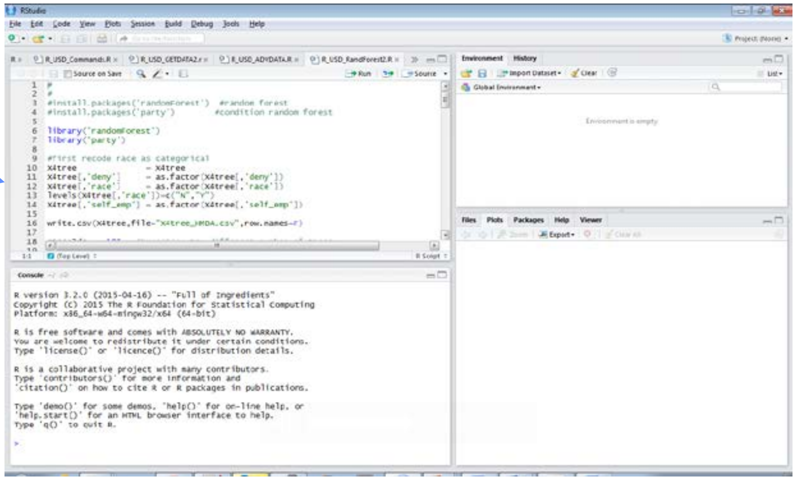
Menu tab

Edit window to Build scripts

R console

Environment Information on variables and command history

Plots, help docs, package lists



```
#
#
#install.packages('randomForest')  #random forest
#install.packages('party')         #condition random forest

library('randomForest')
library('party')

#first recode race as categorical
X4tree              = X4tree
X4tree[,'deny']     = as.factor(X4tree[,'deny'])
X4tree[,'race']     = as.factor(X4tree[,'race'])
levels(X4tree[,'race'])=c("N","Y")
X4tree[,'self_emp'] = as.factor(X4tree[,'self_emp'])

write.csv(X4tree,file="X4tree_HMDA.csv",row.names=F)
```

```
R version 3.2.0 (2015-04-16) -- "Full of Ingredients"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (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.

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.

>
```

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

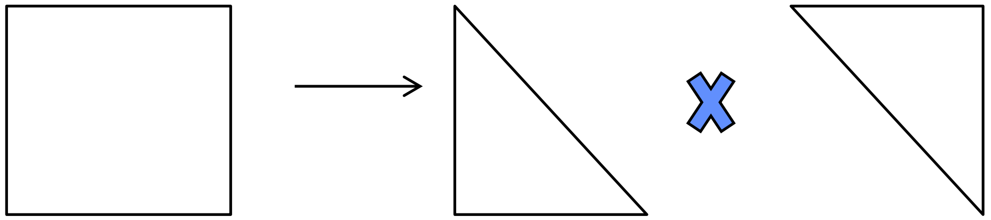
Consider Regression Computations

- **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)

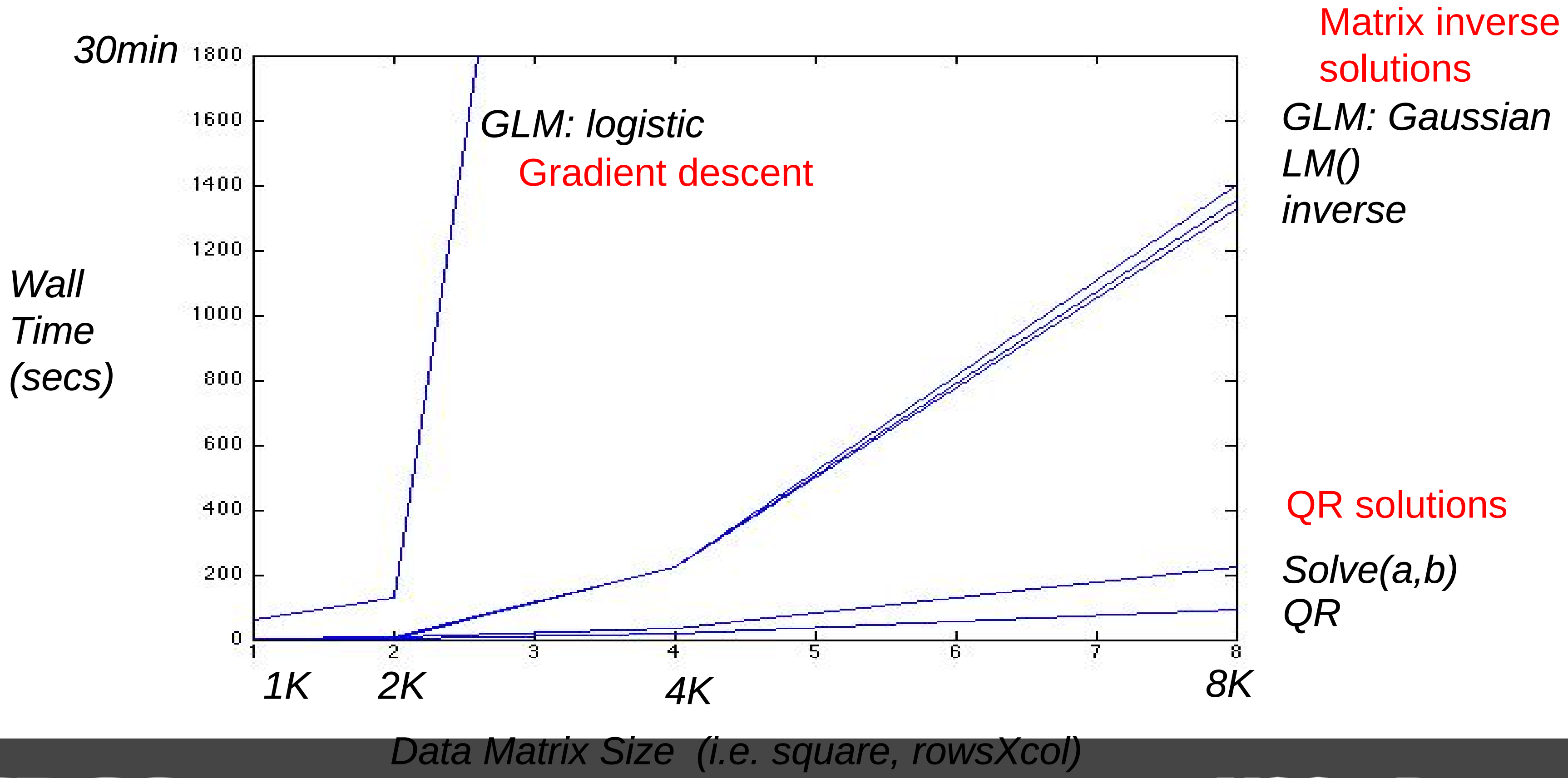
  aov()   #Analysis of Variance

  (returns ANOVA table of F-scores)

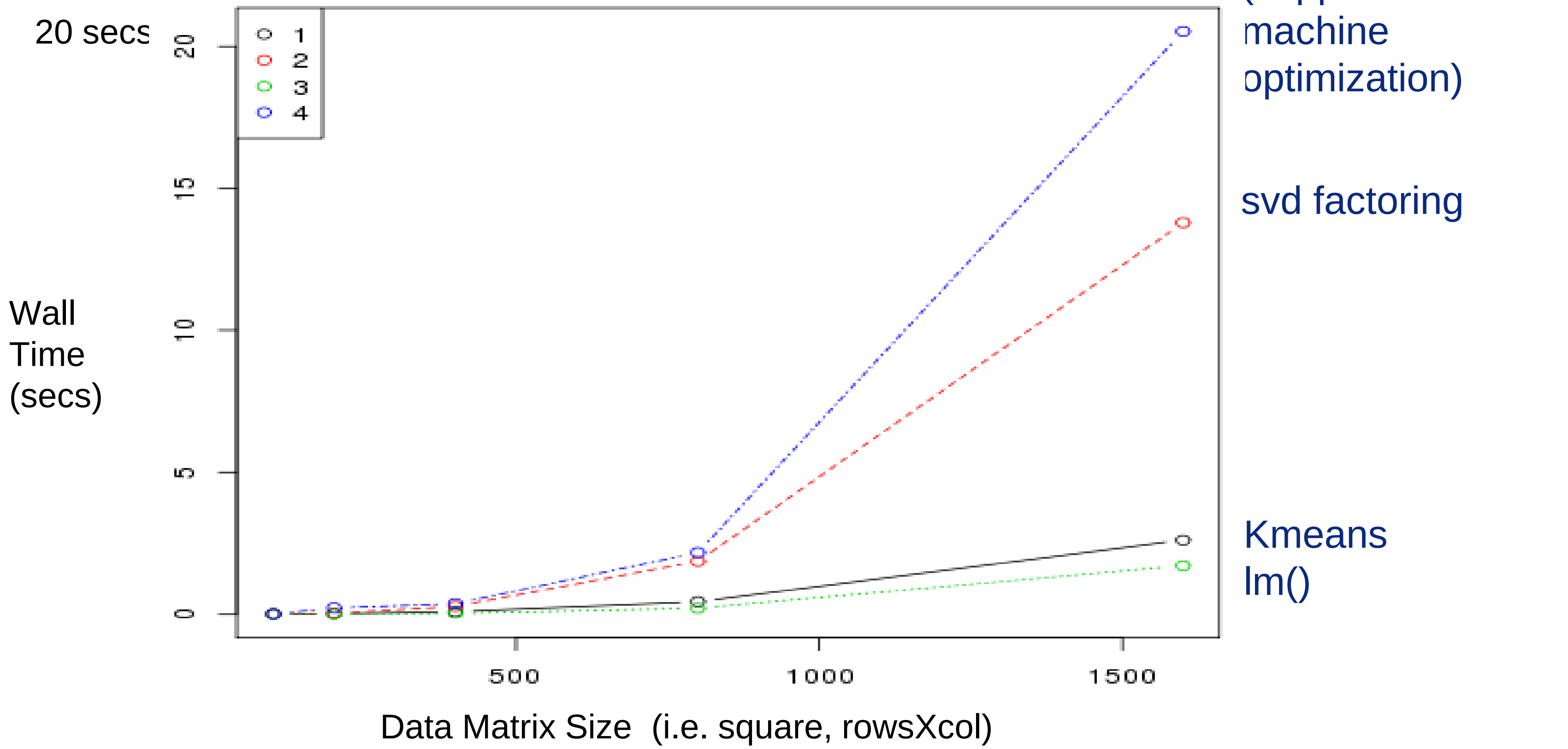
All these work on system of equations

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 multinode.

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

R multicore

- **Run loop iterations on separate cores**

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

allocate workers

$\swarrow$

R multicore

- **Run loop iterations on separate cores**

allocate workers

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

```
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 )
        })
```

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

allocate workers

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

```
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 )
  })
```

returned items
'combined' [illegible]t
by default

specify to combine results into
array with row bind

R multinode: parallel backend

- **Run loop iterations on separate nodes**

```
library(doParallel)

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

allocate cluster as
parallel backend

R multinode: parallel backend

- Run loop iterations on separate nodes

```
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)
```

allocate cluster as parallel backend

%dopar% puts loops across cores and nodes

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!

```
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)
```

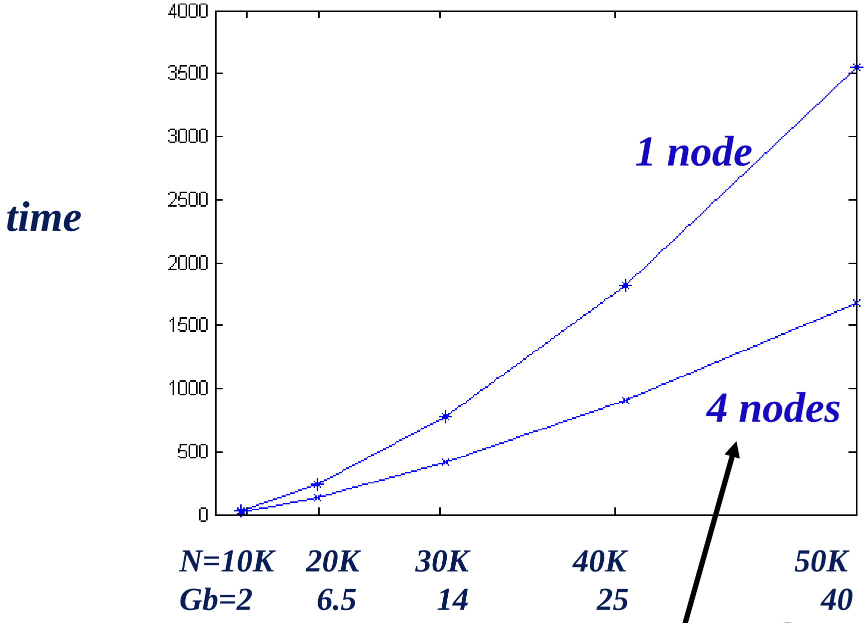
allocate cluster as parallel backend

%dopar% puts loops across cores and nodes

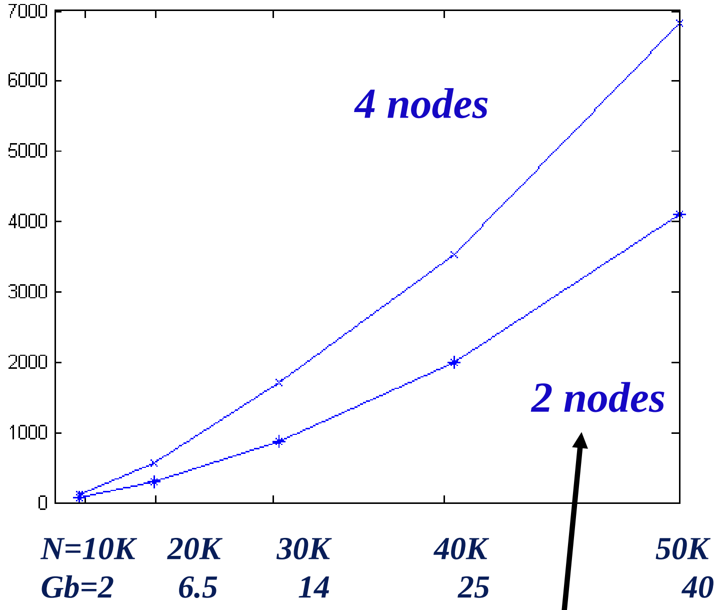
Multiple Compute Nodes not always help

(tested on Gordon)

Matrix Multiplication



Matrix Inversion



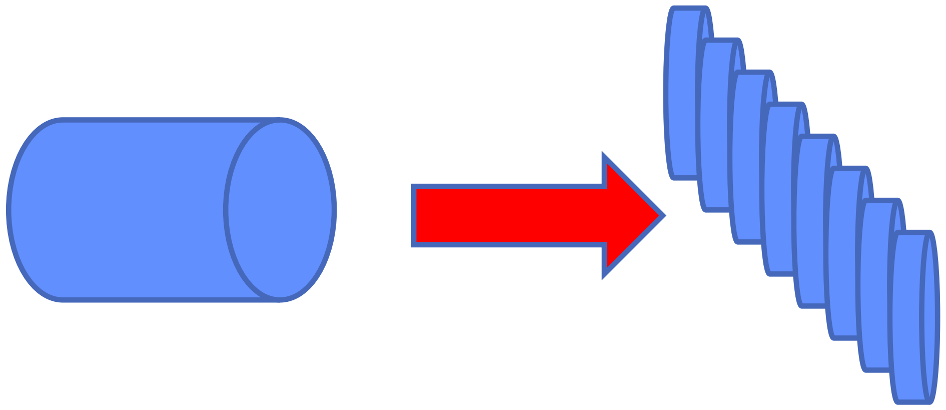
Square Matrix size

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
```

Init MPI and get MPI rank

No other MPI calls made

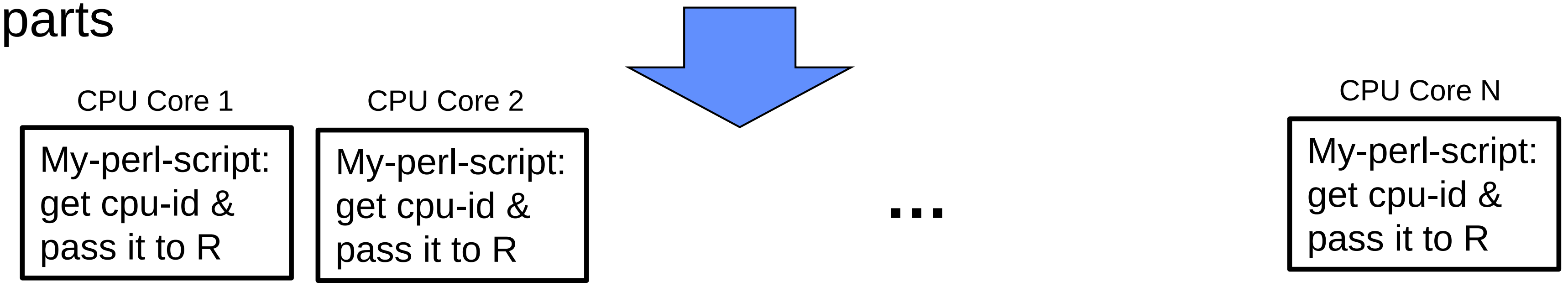
My-perl-script:
get cpu-id &
pass it to R

Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

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

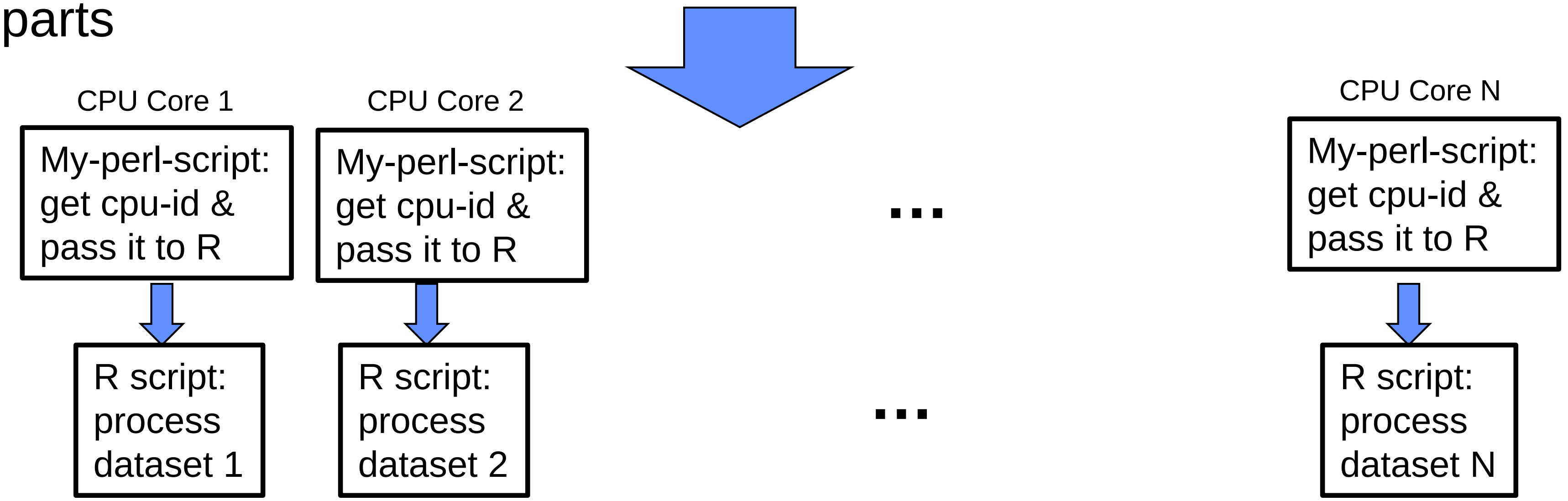

CPU Core 1
My-perl-script: get cpu-id & pass it to R
CPU Core 2
My-perl-script: get cpu-id & pass it to R
…
CPU Core N
My-perl-script: get cpu-id & pass it to R

Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

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

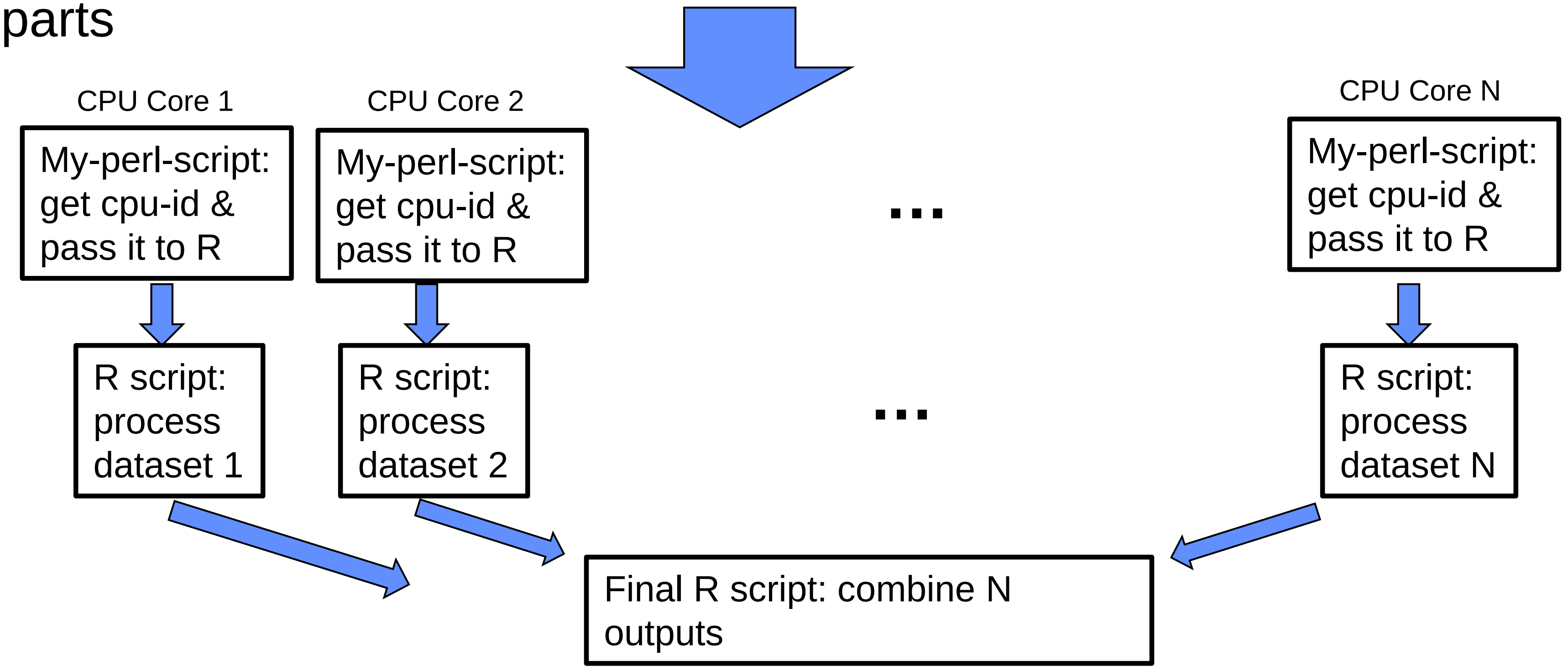

CPU Core 1
My-perl-script: get cpu-id & pass it to R
R script: process dataset 1
CPU Core 2
My-perl-script: get cpu-id & pass it to R
R script: process dataset 2
…
…
CPU Core N
My-perl-script: get cpu-id & pass it to R
R script: process dataset N

Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

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

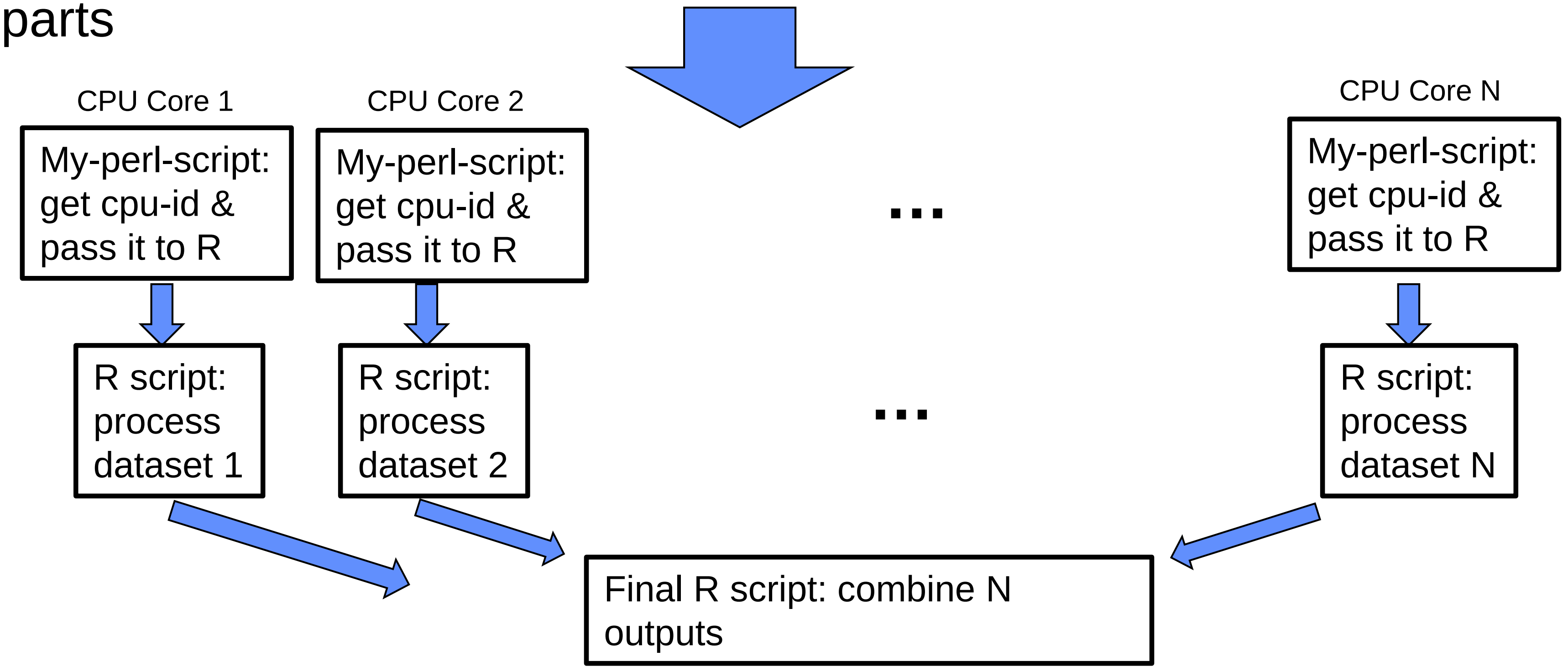

CPU Core 1
My-perl-script: get cpu-id & pass it to R
R script: process dataset 1
CPU Core 2
My-perl-script: get cpu-id & pass it to R
R script: process dataset 2
...
...
CPU Core N
My-perl-script: get cpu-id & pass it to R
R script: process dataset N
Final R script: combine N outputs

Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

ibrun -np processors My-perl-script


CPU Core 1
My-perl-script: get cpu-id & pass it to R
R script: process dataset 1
CPU Core 2
My-perl-script: get cpu-id & pass it to R
R script: process dataset 2
...
...
CPU Core N
My-perl-script: get cpu-id & pass it to R
R script: process dataset N
Final R script: combine N outputs

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

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)

- **R openMP**

  (e.g. if you want to program your own foreach)

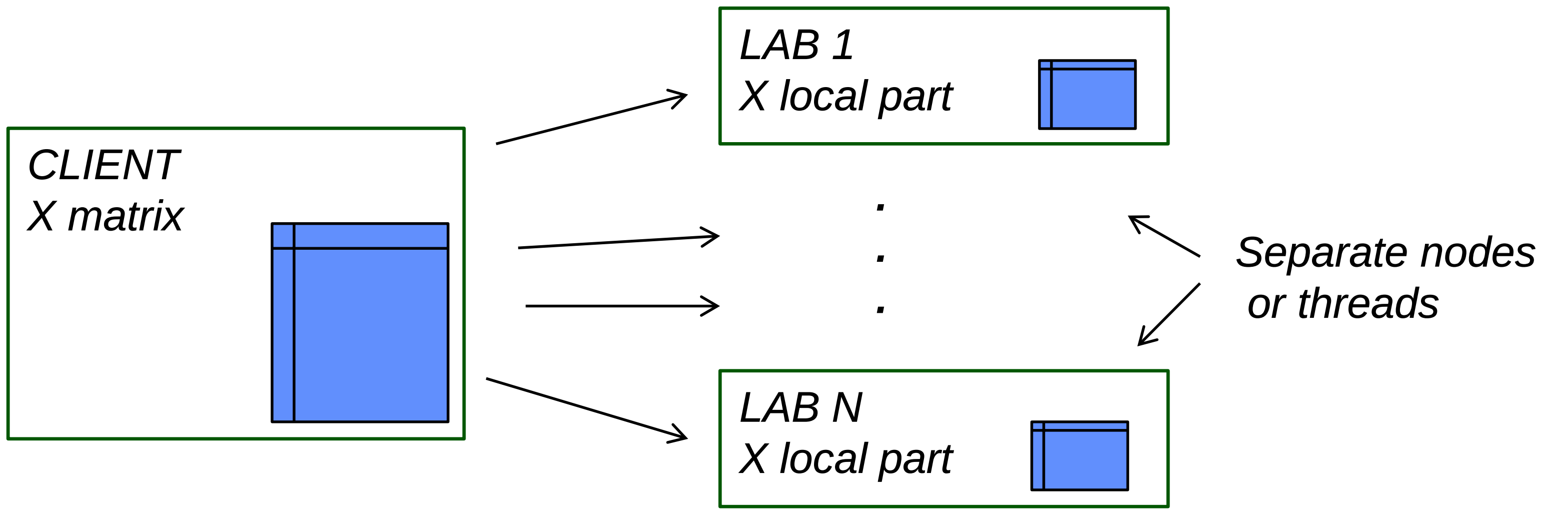
- **Ff, bigmemory – map data to files**

  (can help with foreach)

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

Matlab quickview

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



pause

R on Comet terminal window

1. Get a compute node:

```
[Unix]$ :   srun --partition=debug --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)
```

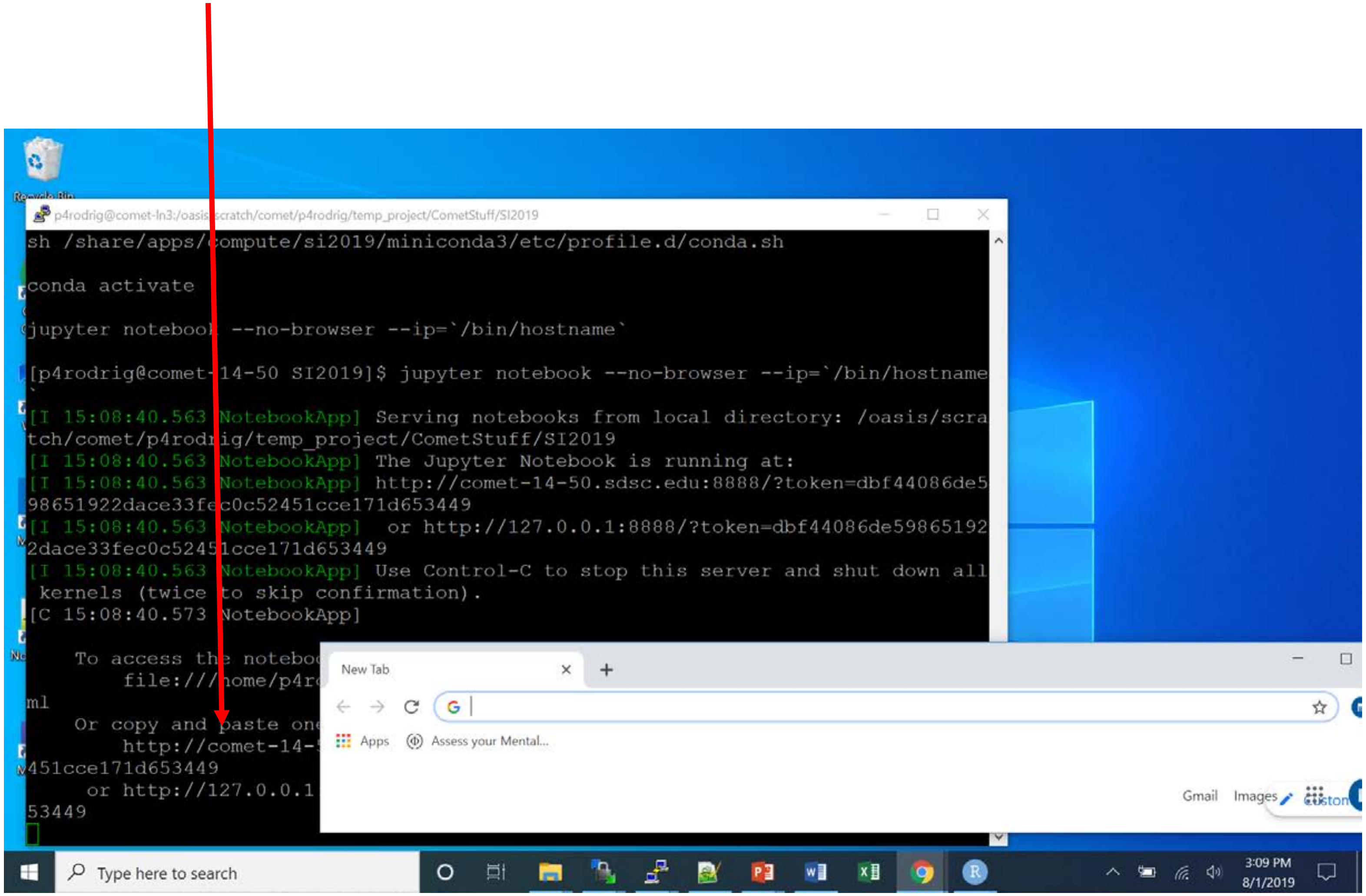
R multicore exercise

- **Login to comet**
  - cd to this lecture folder
- **Get an interactive compute node session**
- **Start notebook**
  - jupyter notebook --no-browser --ip="*" &

R parallel exercises

- **Open & run TestdoParallel Exercise 1,2,3**
  - remember that foreach assumes independence between loops
  - Start with smallish N,P
- **Look at memory usage in top command**
- **R does not well manage large data frames across cores**
  - N=800000 P=2000, makes ~12Gb data frames, R fails
- **Ex 3 will split up data for large data frames and have each core read a separate data**

Starting jupyter notebook and copy paste URL into browser



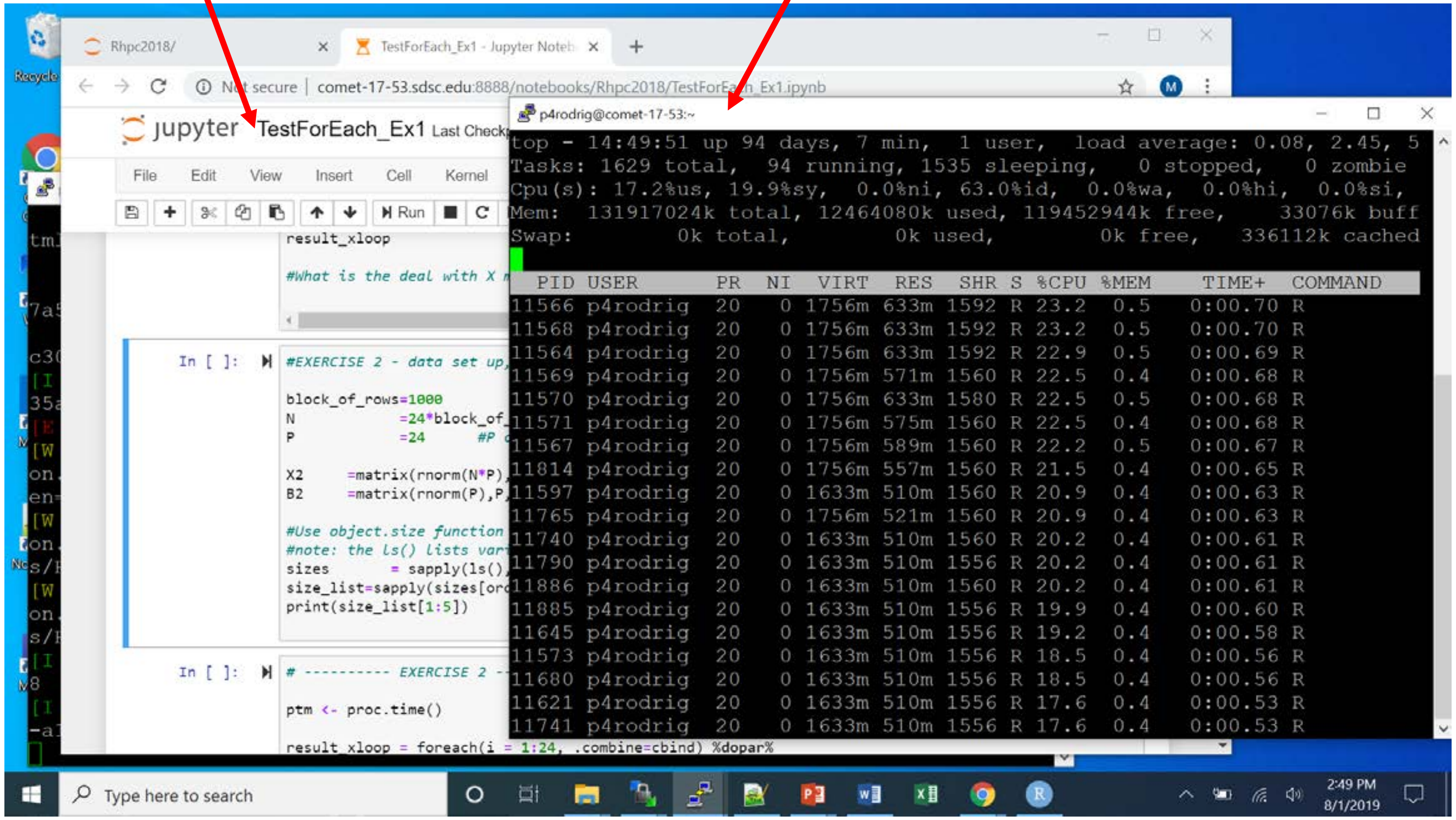
p4rodrig@comet-ln3:/oasis/scratch/comet/p4rodrig/temp_project/CometStuff/SI2019

```
sh /share/apps/compute/si2019/miniconda3/etc/profile.d/conda.sh

conda activate

jupyter notebook --no-browser --ip=`/bin/hostname`

[p4rodrig@comet-14-50 SI2019]$ jupyter notebook --no-browser --ip=`/bin/hostname
`
[I 15:08:40.563 NotebookApp] Serving notebooks from local directory: /oasis/scra
tch/comet/p4rodrig/temp_project/CometStuff/SI2019
[I 15:08:40.563 NotebookApp] The Jupyter Notebook is running at:
[I 15:08:40.563 NotebookApp] http://comet-14-50.sdsc.edu:8888/?token=dbf44086de5
98651922dace33fec0c52451cce171d653449
[I 15:08:40.563 NotebookApp]  or http://127.0.0.1:8888/?token=dbf44086de59865192
2dace33fec0c52451cce171d653449
[I 15:08:40.563 NotebookApp] Use Control-C to stop this server and shut down all
 kernels (twice to skip confirmation).
[C 15:08:40.573 NotebookApp]

    To access the notebo
        file:///home/p4r
ml
    Or copy and paste one
        http://comet-14-
451cce171d653449
     or http://127.0.0.1
53449
```

Select Rhpc2019 folder and select TestdoParallel exercises

Open 2nd terminal window directly in to comet-XX-XX.sdsc.edu comput node

Run top –u $user (then enter H) to see usage



Rhpc2018/
TestForEach_Ex1 - Jupyter Noteb
Not secure | comet-17-53.sdsc.edu:8888/notebooks/Rhpc2018/TestForEach_Ex1.ipynb
jupyter TestForEach_Ex1 Last Checkp
File Edit View Insert Cell Kernel
Run
p4rodrig@comet-17-53:~
Type here to search
2:49 PM
8/1/2019

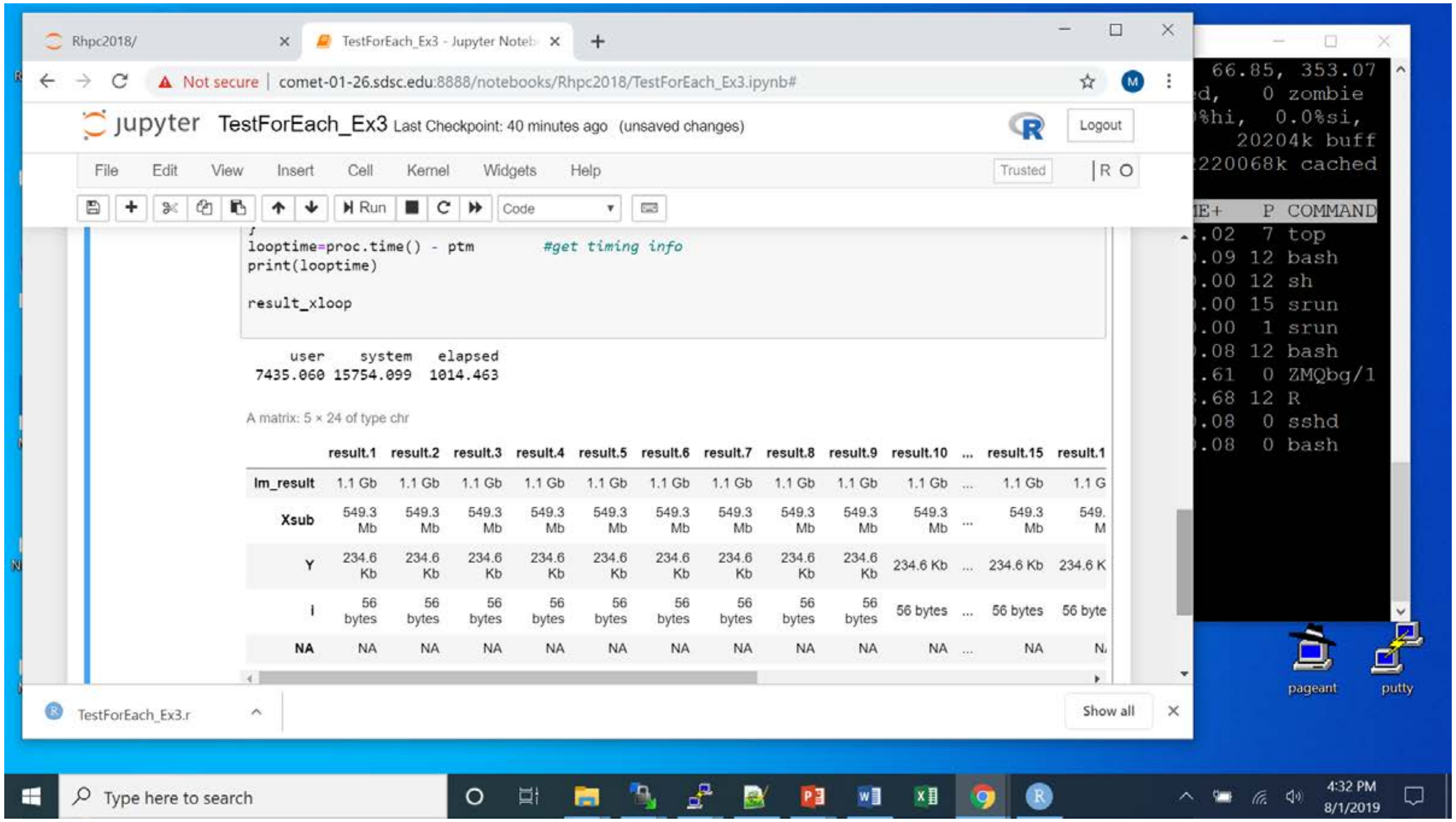
```
result_xloop

#What is the deal with X r
```

```
In [ ]:  #EXERCISE 2 - data set up,

         block_of_rows=1000
         N             =24*block_of_
         P             =24       #P c

         X2      =matrix(rnorm(N*P),
         B2      =matrix(rnorm(P),P,

         #Use object.size function
         #note: the ls() lists vari
         sizes         = sapply(ls(),
         size_list=sapply(sizes[ord
         print(size_list[1:5])
```

```
In [ ]:  # ---------- EXERCISE 2 --

         ptm <- proc.time()

         result_xloop = foreach(i = 1:24, .combine=cbind) %dopar%
```

```
top - 14:49:51 up 94 days, 7 min,  1 user,  load average: 0.08, 2.45, 5
Tasks: 1629 total,  94 running, 1535 sleeping,   0 stopped,   0 zombie
Cpu(s): 17.2%us, 19.9%sy,  0.0%ni, 63.0%id,  0.0%wa,  0.0%hi,  0.0%si,
Mem:  131917024k total, 12464080k used, 119452944k free,    33076k buff
Swap:         0k total,         0k used,         0k free,   336112k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
11566 p4rodrig  20   0 1756m 633m 1592 R 23.2  0.5   0:00.70 R
11568 p4rodrig  20   0 1756m 633m 1592 R 23.2  0.5   0:00.70 R
11564 p4rodrig  20   0 1756m 633m 1592 R 22.9  0.5   0:00.69 R
11569 p4rodrig  20   0 1756m 571m 1560 R 22.5  0.4   0:00.68 R
11570 p4rodrig  20   0 1756m 633m 1580 R 22.5  0.5   0:00.68 R
11571 p4rodrig  20   0 1756m 575m 1560 R 22.5  0.4   0:00.68 R
11567 p4rodrig  20   0 1756m 589m 1560 R 22.2  0.5   0:00.67 R
11814 p4rodrig  20   0 1756m 557m 1560 R 21.5  0.4   0:00.65 R
11597 p4rodrig  20   0 1633m 510m 1560 R 20.9  0.4   0:00.63 R
11765 p4rodrig  20   0 1756m 521m 1560 R 20.9  0.4   0:00.63 R
11740 p4rodrig  20   0 1633m 510m 1560 R 20.2  0.4   0:00.61 R
11790 p4rodrig  20   0 1633m 510m 1556 R 20.2  0.4   0:00.61 R
11886 p4rodrig  20   0 1633m 510m 1560 R 20.2  0.4   0:00.61 R
11885 p4rodrig  20   0 1633m 510m 1556 R 19.9  0.4   0:00.60 R
11645 p4rodrig  20   0 1633m 510m 1556 R 19.2  0.4   0:00.58 R
11573 p4rodrig  20   0 1633m 510m 1556 R 18.5  0.4   0:00.56 R
11680 p4rodrig  20   0 1633m 510m 1556 R 18.5  0.4   0:00.56 R
11621 p4rodrig  20   0 1633m 510m 1556 R 17.6  0.4   0:00.53 R
11741 p4rodrig  20   0 1633m 510m 1556 R 17.6  0.4   0:00.53 R
```

Sample output



TestForEach_Ex3 Last Checkpoint: 40 minutes ago (unsaved changes)
comet-01-26.sdsc.edu:8888/notebooks/Rhpc2018/TestForEach_Ex3.ipynb#

```
looptime=proc.time() - ptm        #get timing info
print(looptime)

result_xloop
```

```
    user    system   elapsed
7435.060 15754.099  1014.463
```

A matrix: 5 × 24 of type chr

	result.1	result.2	result.3	result.4	result.5	result.6	result.7	result.8	result.9	result.10	...	result.15	result.1
lm_result	1.1 Gb	1.1 Gb	1.1 Gb	1.1 Gb	1.1 Gb	1.1 Gb	1.1 Gb	1.1 Gb	1.1 Gb	1.1 Gb	...	1.1 Gb	1.1 G
Xsub	549.3 Mb	549.3 Mb	549.3 Mb	549.3 Mb	549.3 Mb	549.3 Mb	549.3 Mb	549.3 Mb	549.3 Mb	549.3 Mb	...	549.3 Mb	549. M
Y	234.6 Kb	234.6 Kb	234.6 Kb	234.6 Kb	234.6 Kb	234.6 Kb	234.6 Kb	234.6 Kb	234.6 Kb	234.6 Kb	...	234.6 Kb	234.6 K
i	56 bytes	56 bytes	56 bytes	56 bytes	56 bytes	56 bytes	56 bytes	56 bytes	56 bytes	56 bytes	...	56 bytes	56 byte
NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	...	NA	N

```
 66.85, 353.07
d,    0 zombie
%hi,  0.0%si,
   20204k buff
220068k cached

E+    P COMMAND
.02   7 top
.09  12 bash
.00  12 sh
.00  15 srun
.00   1 srun
.08  12 bash
.61   0 ZMQbg/1
.68  12 R
.08   0 sshd
.08   0 bash
```

SDSC SAN DIEGO SUPERCOMPUTER CENTER

UC San Diego

- **Pause**

pbdR package

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

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

pbdR sample code

```
library(pbdDMAT)

init.grid()                # <<< ----  pbdR will select grid sizes for you by default

myr   =comm.rank()
mys   =comm.size()

#Simple ways to print information
comm.print(paste("comm print myrank:",myr, " size:",mys),all=FALSE)

p=10000
dx <- ddmatrix(rnorm(p*p*10),p*10,p)    # <<< --- you and indicate how to block data onto grid
comm.print(dx,all=F)

….
```

To run: edit Runpbd script and enter: sbatch Runpbd

Test 1

For 1 node 24 cores:

```
Using 6x4 for the default grid size

[1] "comm print myrank: 0  size: 24"
[1] " matrix width: 10000"
--------------------------------------------------------------------------
orterun noticed that process rank 0 with PID 26491 on
node comet-18-56 exited on signal 9 (Killed).
--------------------------------------------------------------------------
```

But runs out of memory
(2 nodes 24 cores also runs out of memory)

Test 2

For 1 node 12 cores:

```
Using 4x3 for the default grid size

[1] "comm print myrank: 0  size: 12"
[1] " matrix width: 10000"
COMM.RANK = 0

DENSE DISTRIBUTED MATRIX
---------------------------
Process grid:                    4x3
Global dimension:    100000x10000
(max) Local dimension:           25008x3344
Blocking:            16x16
BLACS ICTXT:                     0
```

data split up among cores

Runs in about 950 secs

Test 3

For 2 node 12 cores:

```
Using 6x4 for the default grid size

[1] "comm print myrank: 0  size: 24"
[1] " matrix width: 10000"
COMM.RANK = 0

DENSE DISTRIBUTED MATRIX
---------------------------
Process grid:                    6x4
Global dimension:    100000x10000
(max) Local dimension:           16672x2512
Blocking:              16x16
BLACS ICTXT:                     0
```

Runs in about 320 secs

THE END