High Performance Computing and Big Data Analytics: An Introduction

Matthew J. Denny
University of Massachusetts Amherst

matthewjdenny@gmail.com
www.mjdenny.com

May 28, 2015

INSTITUTE FOR SOCIAL SCIENCE RESEARCH
UNIVERSITY OF MASSACHUSETTS AMHERST

UMassAmherst

Overview

1. What is High Performance Computing (**HPC**)/Big Data Analytics?
2. Strategy - work smart, not hard.
3. Software and programming choices.
4. Hardware.
5. Resources.

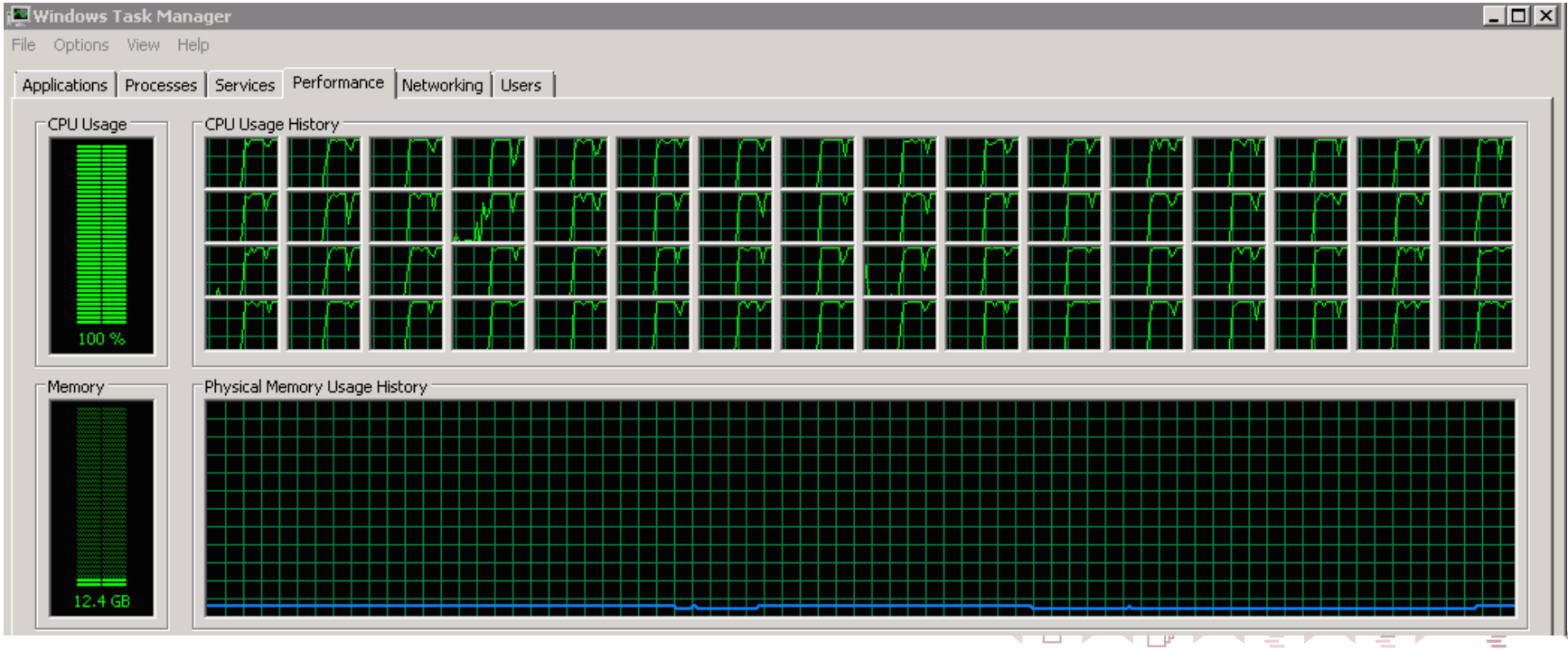
1. What is it?

What it is

- An **approach** more than a specific set of tools
- Effort to scale up analysis.
- Determining the most **efficient** way to complete your analysis task.
  - Time
  - Resources

High Performance Computing

- Run analysis faster.
- Run analysis at larger scale.
- Work on more complex problems.

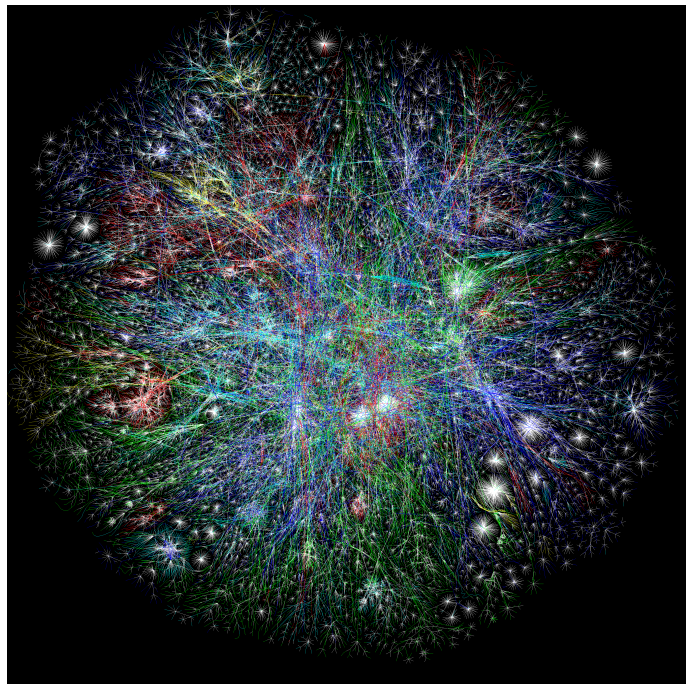


How?

- Make use of low overhead, high speed programming languages (`C, C++, Fortran, Java, etc.`)
- Parallelization.
- Efficient implementation.
- **Good scheduling**.

Big Data Analytics

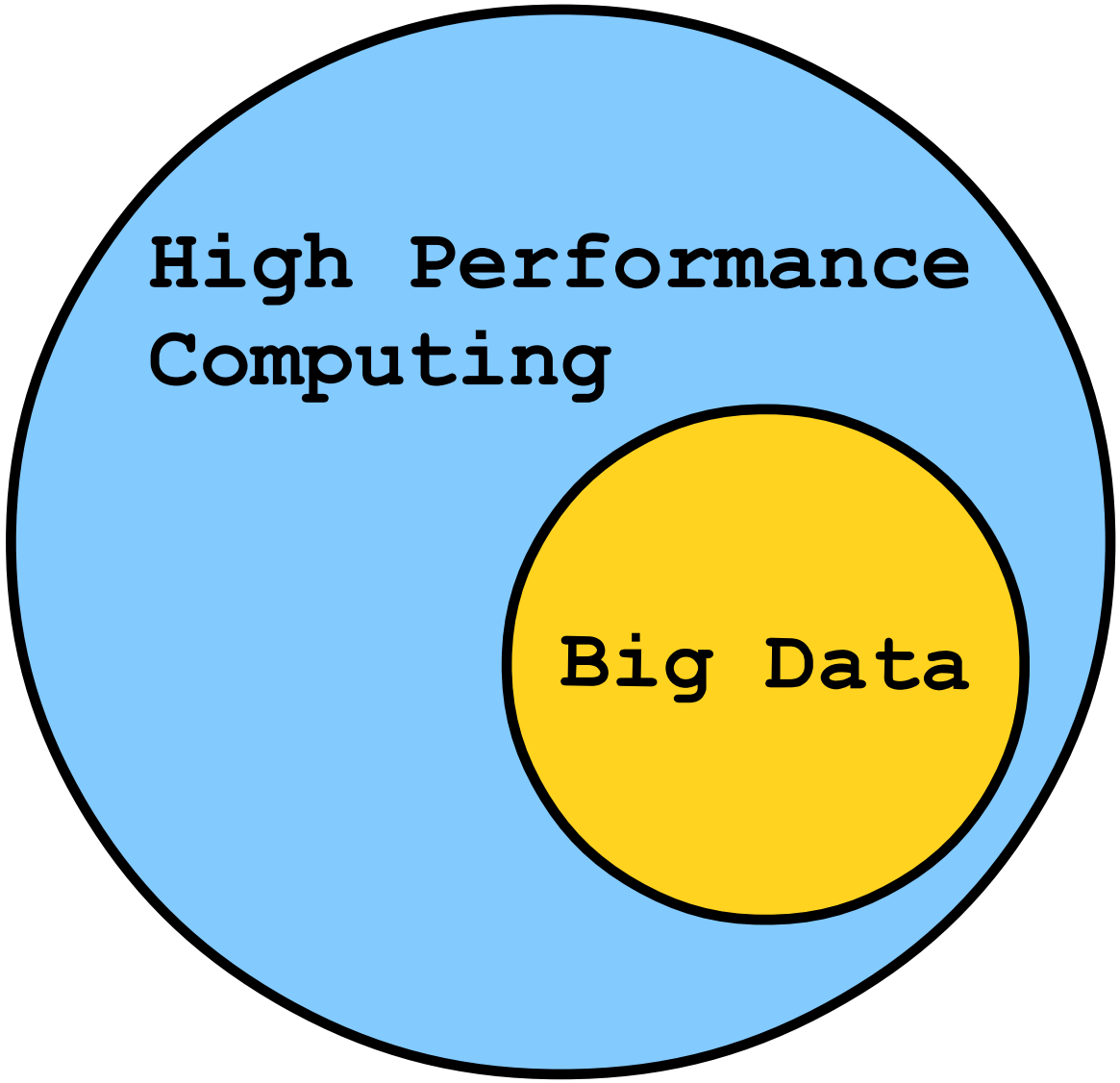
- Work with larger datasets.
- Efficiently analyze large datasets.
- Leverage large amounts of data.



How?

- Use memory efficient data structures and programming languages.
- More RAM.
- Databases.
- Efficient inference procedures.
- **Good scheduling**.

How they fit together



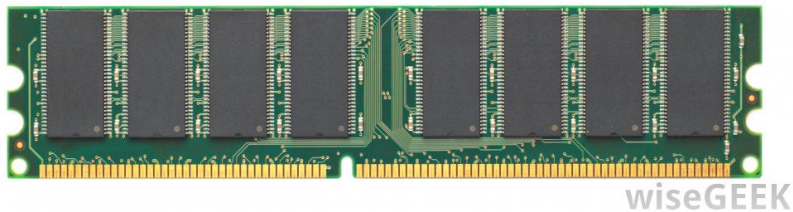
2. Strategy

Checklist

1. **Understand** your challenge.
2. Determine **which approach** is appropriate.
3. **Exercise Constraint**.
4. Work **smarter**, not harder.

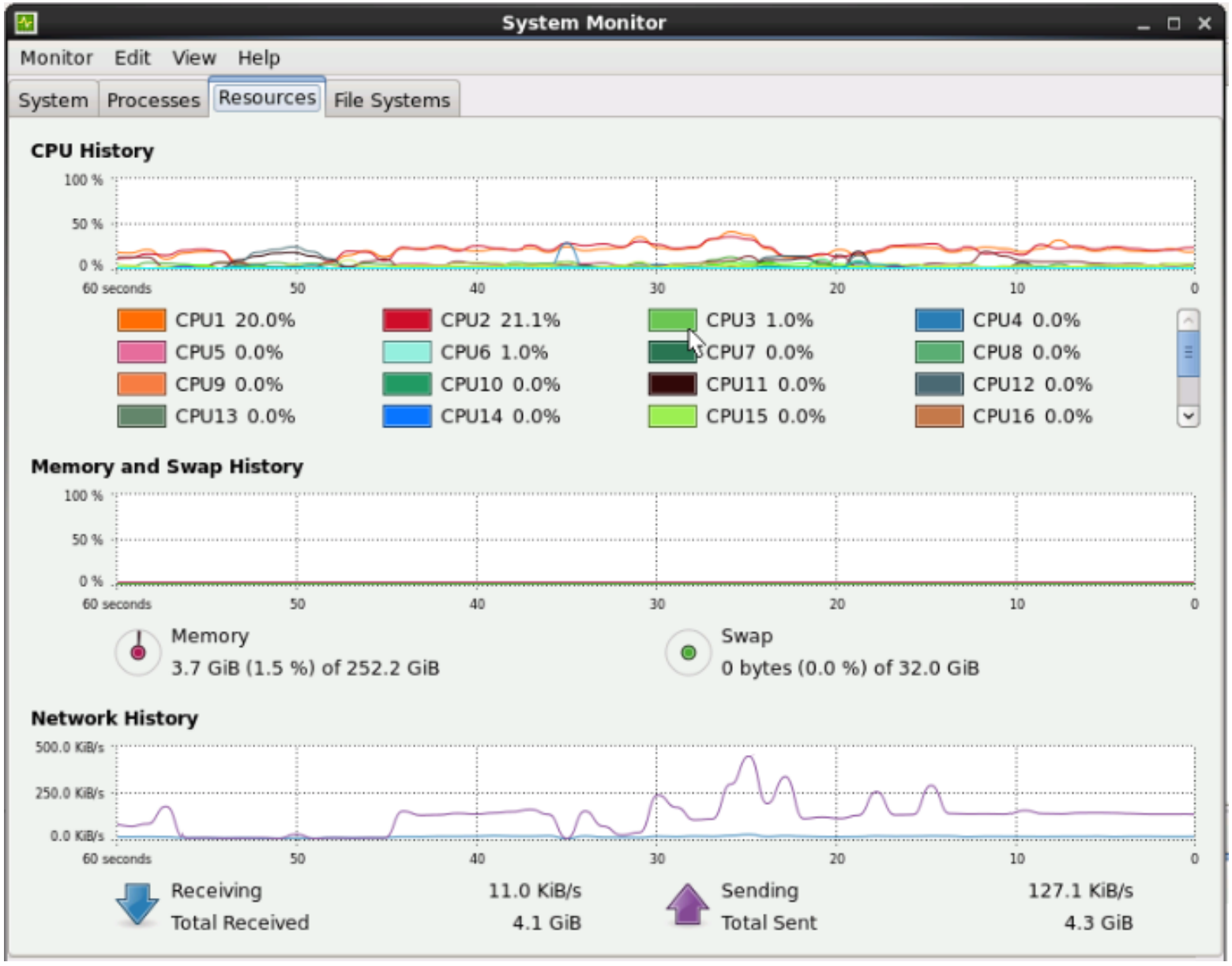
Hardware constraints

- **RAM** = computer working memory – determines size of datasets you can work on.



- **CPU** = processor, determines speed of analysis and degree of parallelization.

Look at your activity monitor!



Other factors to consider

- Different analysis packages are designed for different scales.
- **Know your data.**
- When does the project have to be complete?

Understand your challenge

- Large dataset.
- Analysis takes a long time to run.
- Analysis requires many replications.

Large dataset – panel data, event history data

- Determine memory requirements.
- Use memory efficient software.
- Find a high RAM computer.
- Break problem up.

Long run time – MCMC, optimization

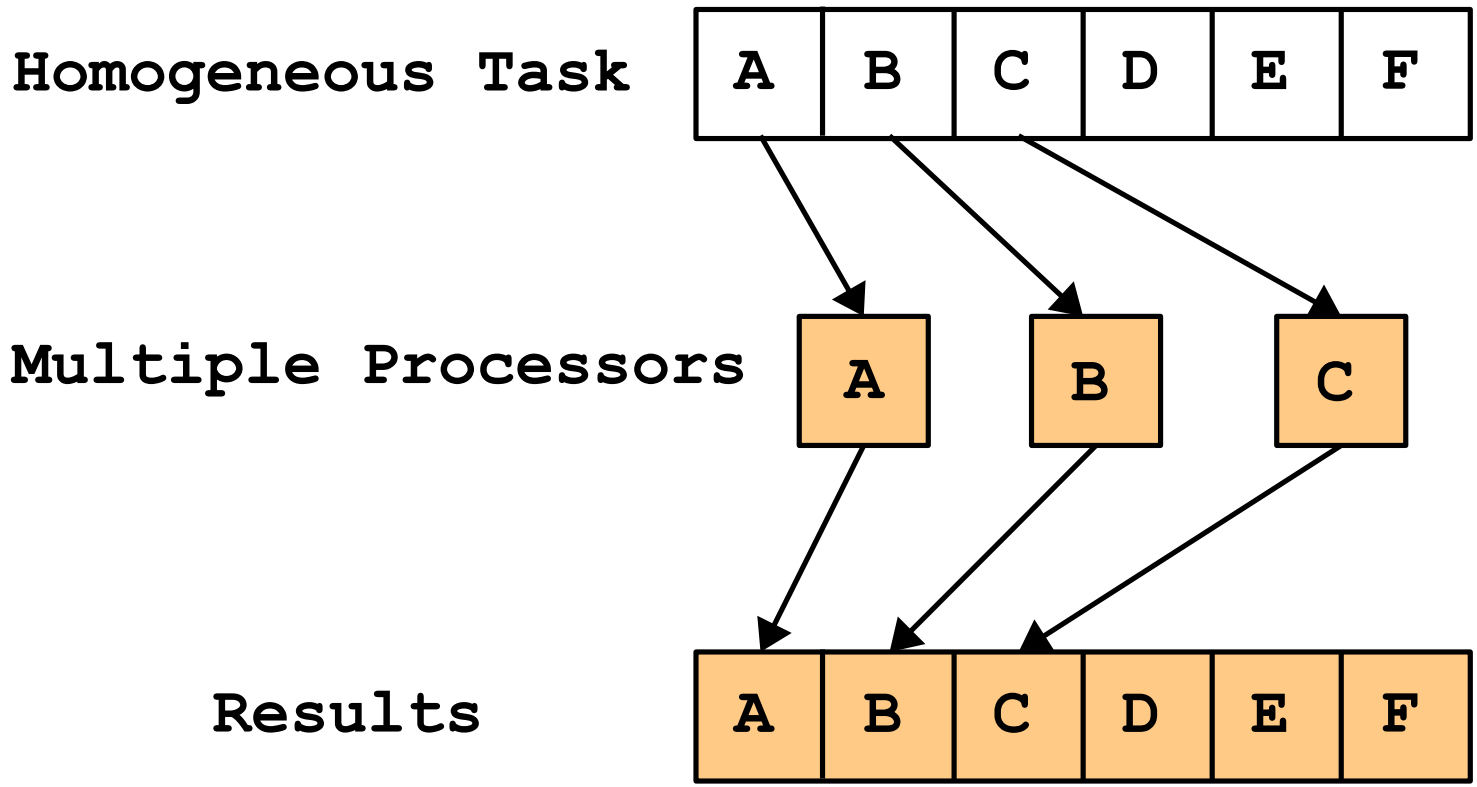
- Determine approximate run time.
  - Less than a month? – Just let it run.
- Reliable power, turn off automatic updates, save periodically if possible.
- Implement in a faster language (C++ gives 1,000+ times speedup over R)

Many replications – cross validation, bootstrapping

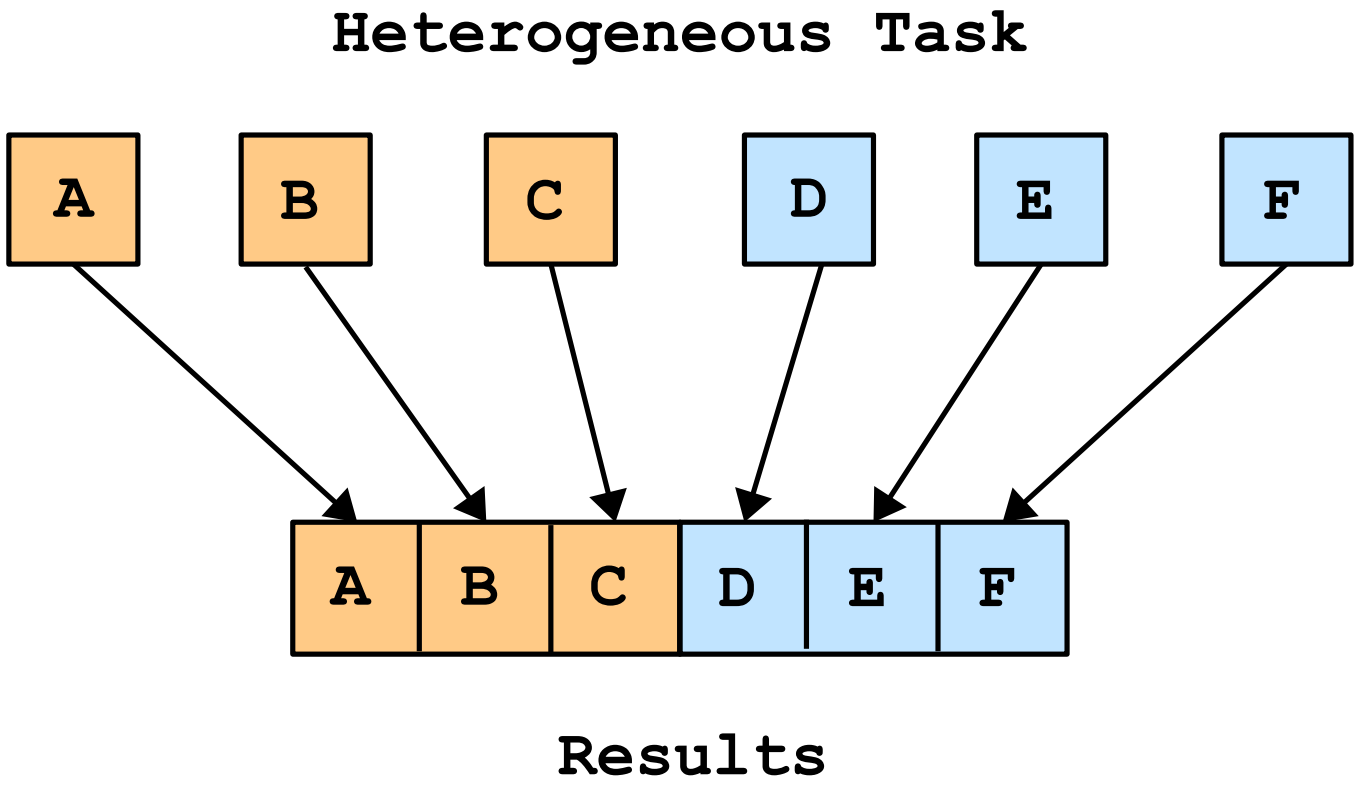
- Write code to run one instance.
- Use looping, try subset first.
- **Parallelize.**
- Use several computers/cluster.

2.1 Parallelization

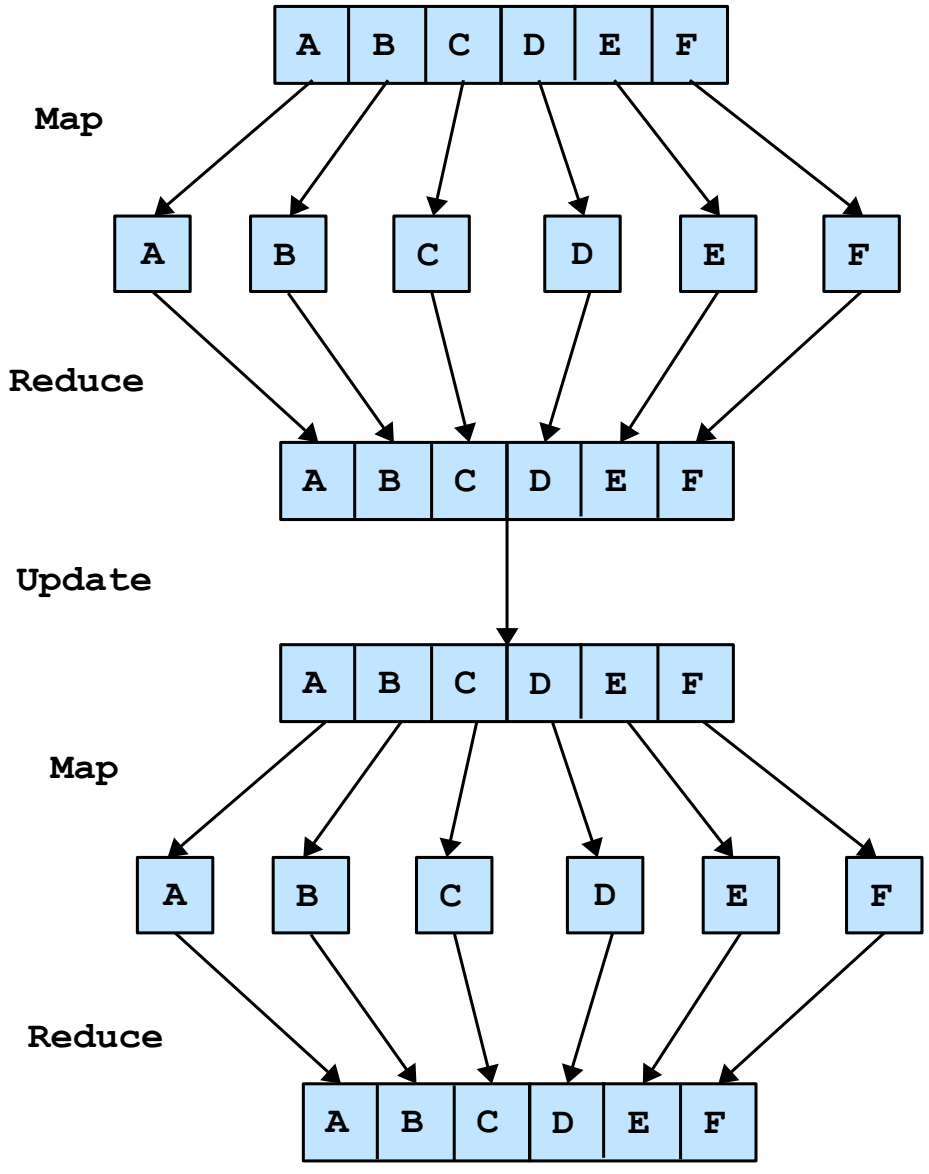
Homogeneous parallelization



Heterogeneous parallelization



Map-Reduce


A B C D E F
Map
A B C D E F
Reduce
A B C D E F
Update
A B C D E F
Map
A B C D E F
Reduce
A B C D E F

2.2 General Advice

It takes time...

- **Implementing HPC can take more time than you save.**
- Reuse somebody else's code where possible – GOOGLE IT!
- **You will not be professionally rewarded for HPC skills.**
- Get help, find a collaborator.

Exercise restraint

- Weight the costs of an HPC project before pursuing it – get advice.
- HPC resources are expensive, be careful with your money.
- Invest in resources/languages that will be transferable to other projects.

Know the benefits

- Developing HPC skillset can make you a **valuable collaborator**.
- HPC skills most valuable when they let you **work on a problem you could not otherwise**.
- Industry loves HPC, can get internships in data science.

3. Software and Programming Choices

Important considerations

1. Software platform can make a big difference in speed/efficiency.
2. Programming speed and readability vs. run time.
3. Repetitive tasks can be automated.
4. Remote access is valuable.

Software choices

- **Stata, SAS, SPSS, Matlab**
  - Readable syntax, fast programming, less control.
- **R, Python**
  - Flexible, more control, harder to code.
- **C++, C, Fortran, Java, CUDA**
  - Most control, fastest, hardest to code.

Stata

- Memory efficient
- Reasonably fast.
- Not flexible.
- Have to pay for multithreaded version.

Python

- Great for file I/O.
- Easy to read
- Incredibly flexible.
- Can interface with other languages.

R

- Access to many statistical packages.
- Existing code base for HPC.
- Not memory efficient, or fast.

C++

- **Very fast.**
- Interface with R.
- Difficult to program.

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

3.1 Software Packages

R packages for HPC

- snowfall – cluster paralellization
  - `sfClusterApplyLB( )`
- parallel – included in base R
  - `mclapply( )` or `foreach`
- biglm – high memory regression
  - `bigglm( )`

C++ in R

- `Rcpp` – allows for the integration of C++ code in R.
- `RcppArmadillo` – Gives access to Armadillo libraries.
- `RStudio` – IDE of R with built in C++ debugging

3.2 Programming Choices

Efficient R programming

- Loops are slow in R, but faster than doing something by hand
- Built-in functions are mostly written in C – much faster!
- Subset data before processing when possible.
- Test with `system.time({ code })`

Loops are "slow" in R

```
system.time({
    vect <- c(1:10000000)
    total <- 0
    #check using a loop
    for(i in 1:length(as.numeric(vect))){
        total <- total + vect[i]
    }
    print(total)
})
```

```
[1] 5e+13
   user  system elapsed
  7.641   0.062   7.701
```

And fast in C

```
system.time({
    vect <- c(1:10000000)
    #use the builtin R function
    total <- sum(as.numeric(vect))
    print(total)
})

[1] 5e+13
   user  system elapsed
  0.108   0.028   0.136
```

Summing over a sparse dataset

```
#number of observations
numobs <- 100000000

#observations we want to check
vec <- rep(0,numobs)

#only select 100 to check
vec[sample(1:numobs,100)] <- 1

#combine data
data <- cbind(c(1:numobs),vec)
```

Conditional checking

```
system.time({
    total <- 0
    for(i in 1:numobs){
        if(data[i,2] == 1)
        total <- total + data[i,1]
    }
    print(total)
})

[1] 5385484508
   user  system elapsed
199.917   0.289 200.350
```

Subsetting

```
system.time({
    dat <- subset(data, data[,2] ==1)
    total <- sum(dat[,1])
    print(total)
})
```

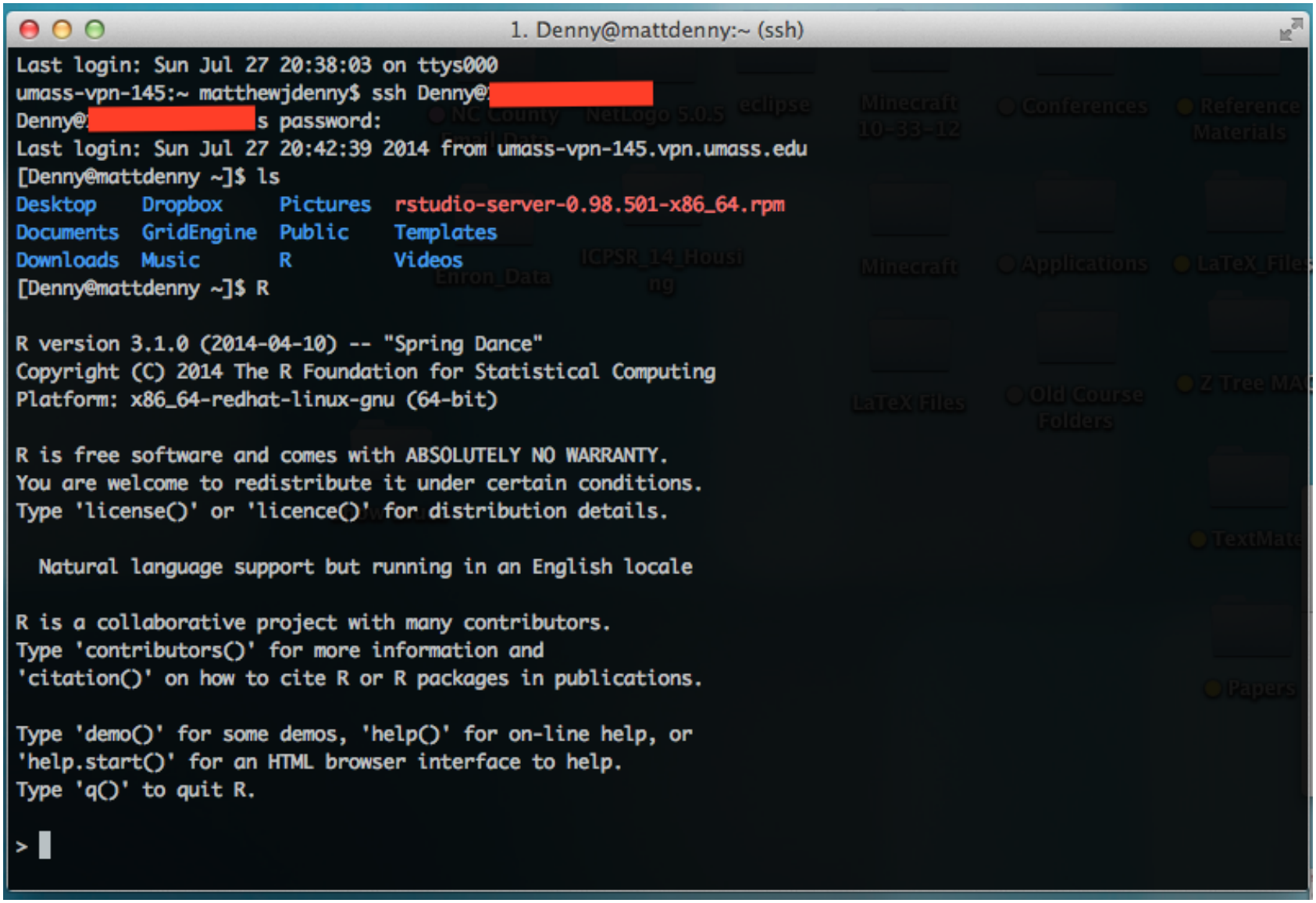
```
[1] 5385484508
   user  system elapsed
  5.474   1.497   8.245
```

3.3 Remote Access

Overview

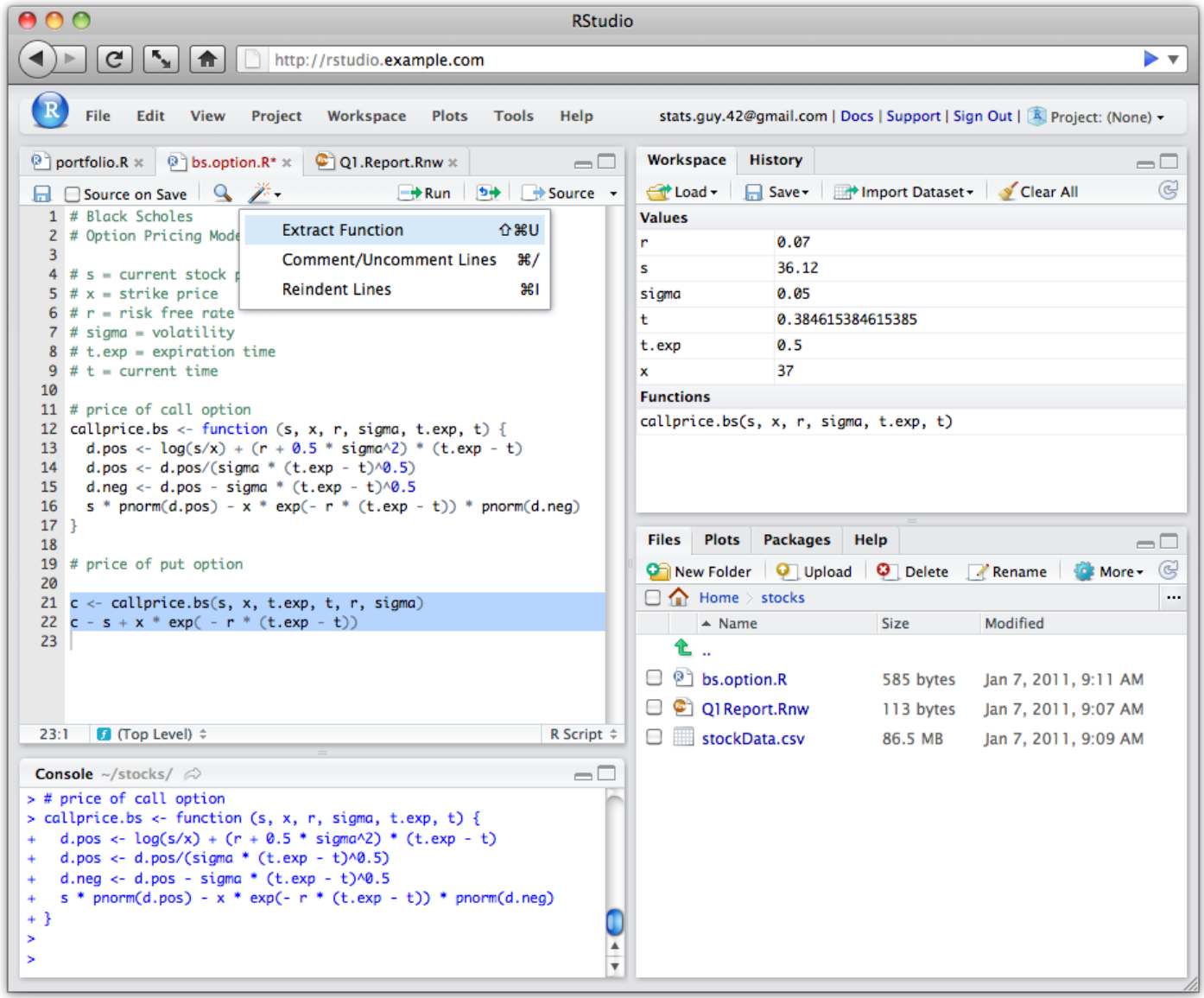
- Connect from your laptop to an HPC resource.
- Secure shell (SSH), graphical interface (RDP/VNC) or web interface (RStudio Server)
- Cluster job scheduling.

SSH

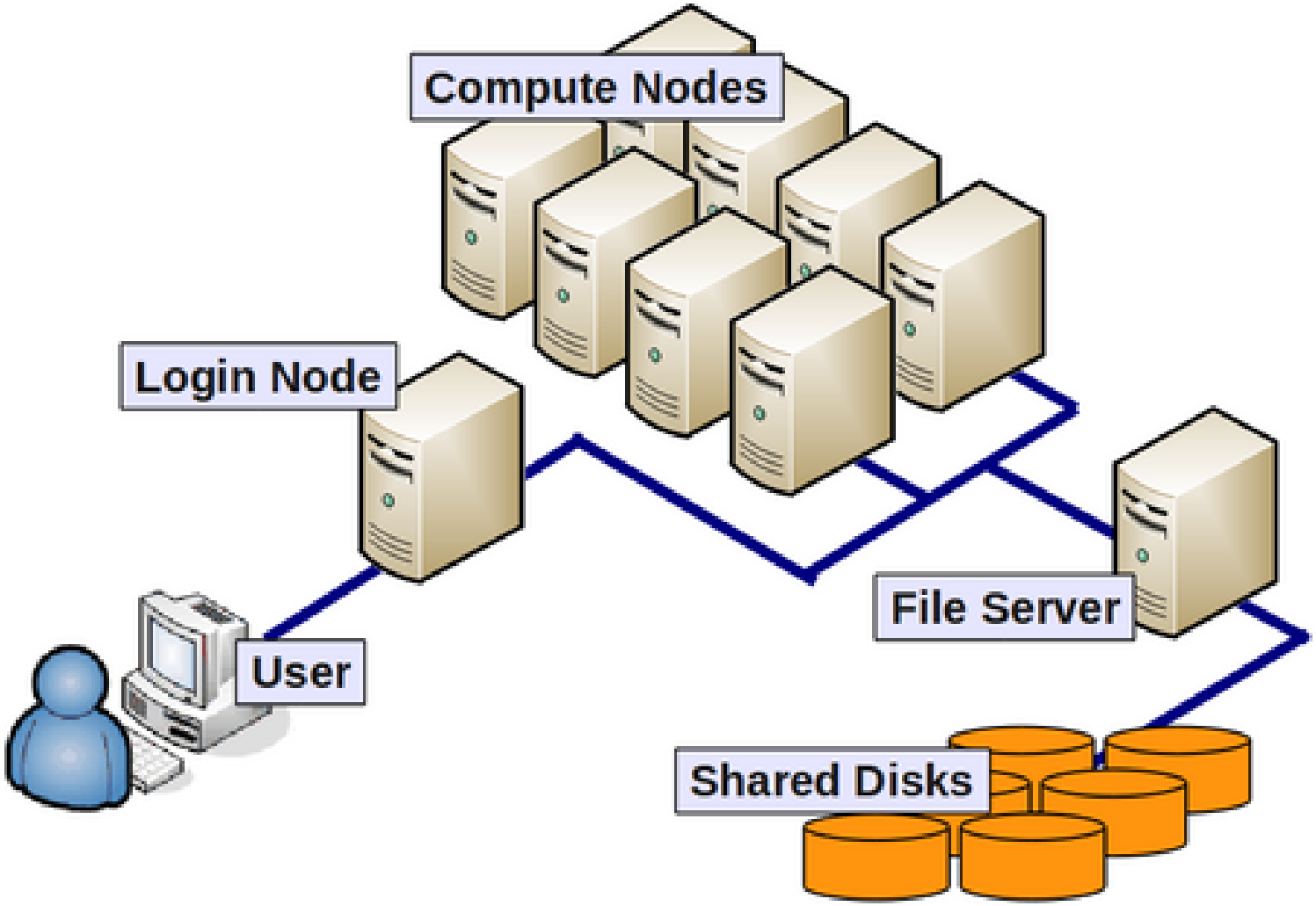

1. Denny@mattdenny:~ (ssh)
Last login: Sun Jul 27 20:38:03 on ttys000
umass-vpn-145:~ matthewjdenny$ ssh Denny@[redacted]
Denny@[redacted]s password:
Last login: Sun Jul 27 20:42:39 2014 from umass-vpn-145.vpn.umass.edu
[Denny@mattdenny ~]$ ls
Desktop Dropbox Pictures rstudio-server-0.98.501-x86_64.rpm
Documents GridEngine Public Templates
Downloads Music R Videos
[Denny@mattdenny ~]$ R
R version 3.1.0 (2014-04-10) -- "Spring Dance"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-redhat-linux-gnu (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.
Natural language support but running in an English locale
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.
>

Connecting to a remote desktop/Workstation

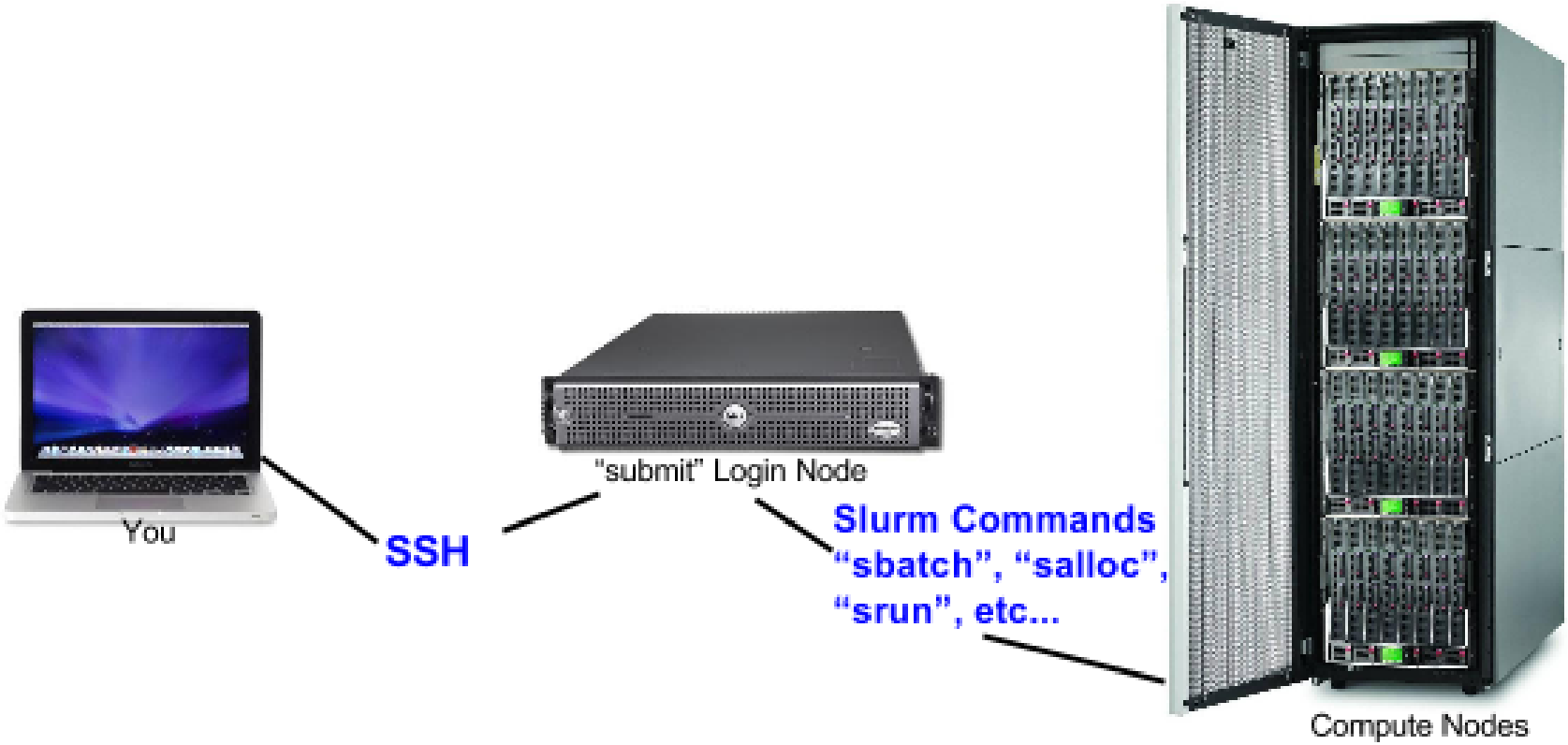
RStudio Server over Internet (Linux only)



How a cluster works



Connecting to a cluster

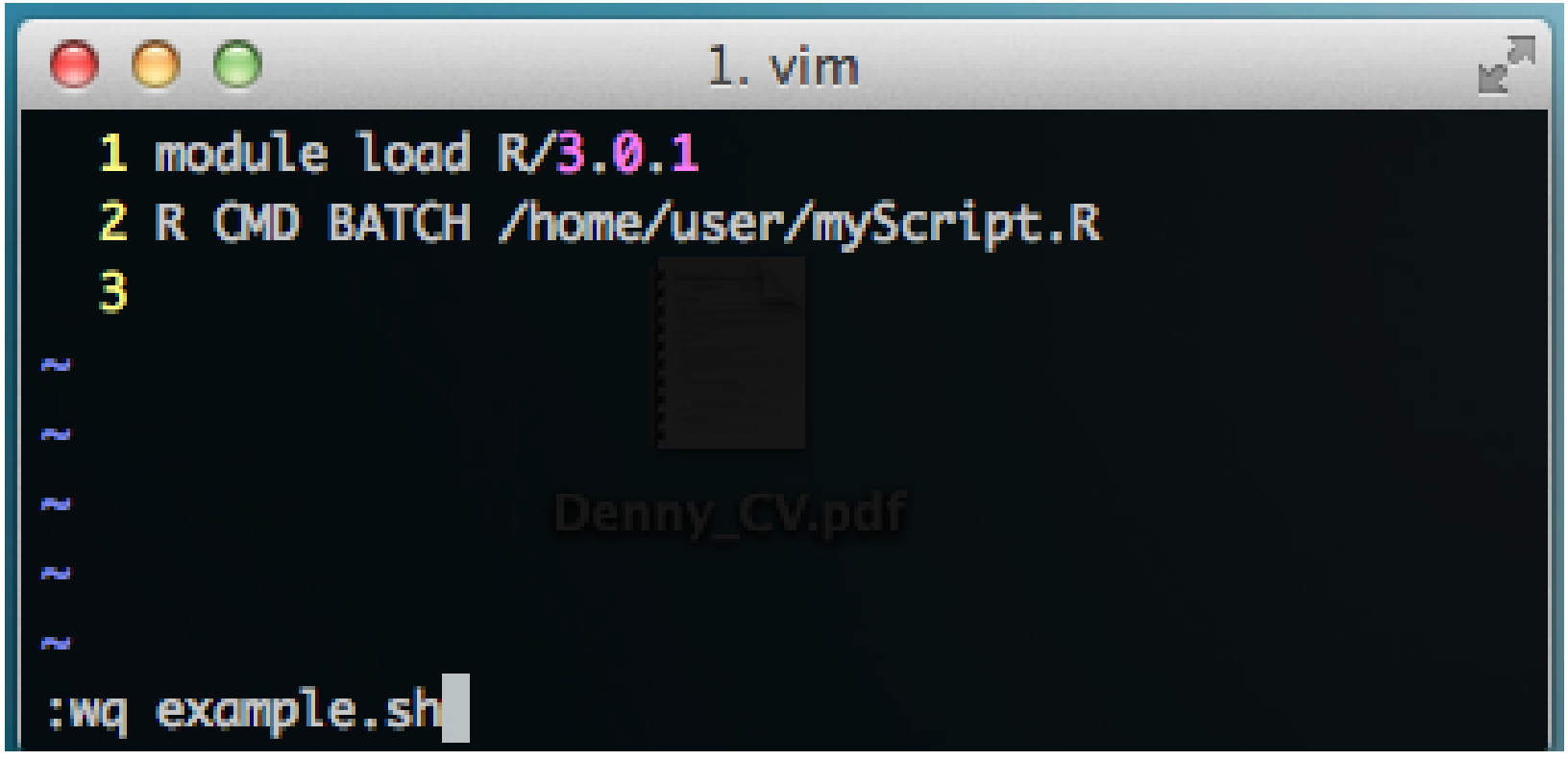


Job scheduling on a cluster

- System to share resources.
- Job Schedulers (Moab ,Grid Engine, LoadLeveler, SLURM, LSF).
- SSH $\longrightarrow$ FTP data to local directory $\longrightarrow$ submit "job"

```
bsub -n 4 -R "rusage[mem=2048]"
-W 0:10 -q long example.sh
```

example.sh



The file has now been created and saved:

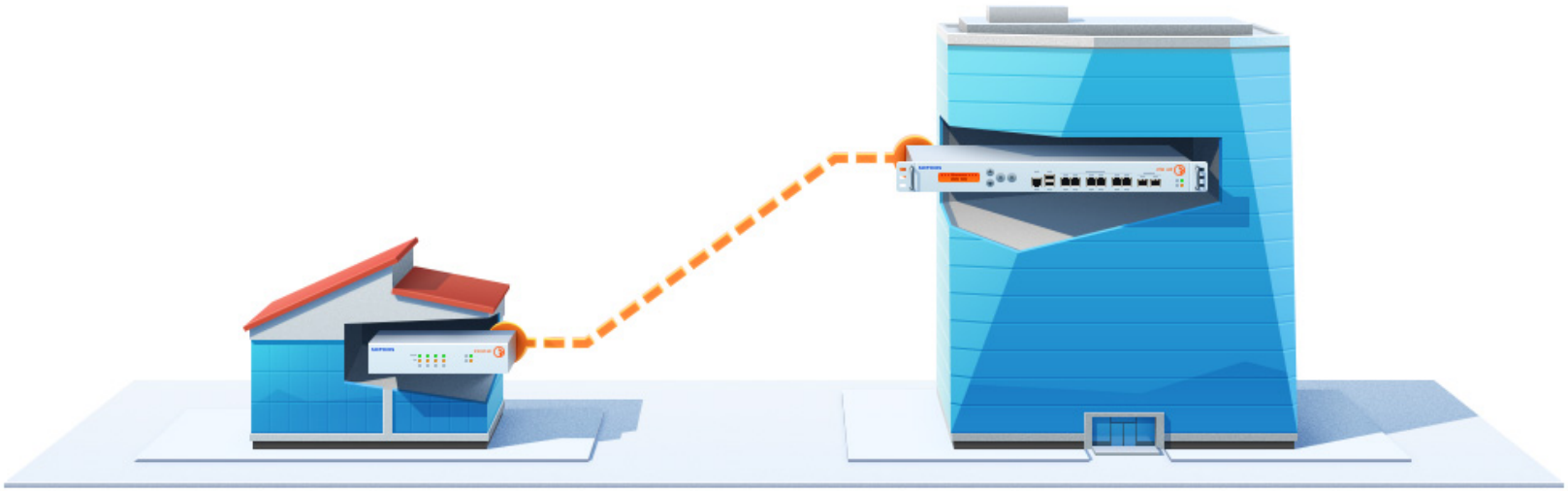
```
0587377979:ICPSR_Summer_14 matthewjdenny$ cd HPC_Workshop_Materials/
0587377979:HPC_Workshop_Materials matthewjdenny$ ls
example.sh
0587377979:HPC_Workshop_Materials matthewjdenny$
```

Connecting to your own desktop/Workstation

- You will need an IP address:
  Example – `35.2.23.132`
- Static vs. Dynamic
- Your university should be able to provide a static IP for free.
- Dyn DNS – $25 per year
  http://dyn.com/remote-access/

Security concerns

- If you get a static IP, people will try to hack into your computer.
- Set firewall (see course handout)
- Use a Virtual Private Network (VPN).



4. Hardware

Classes of hardware

1. Supercomputers
2. Mainframes
3. **Cluster Computing Resources**
4. Servers
5. **HPC Workstations**
6. **Consumer Desktops**
7. GPGPU

Supercomputers

- Used when all computing resources are needed to solve one problem.
- Physics, engineering, materials science

Mainframes

- Used for large database applications.
- Business analytics, healthcare.

Cluster Computing Resources

- Flexible, used for parallel and high memory tasks.
- General purpose academic computing infrastructure.

Servers

- Most often used for hosting websites.
- Can be useful for long jobs, high memory.

HPC Workstations

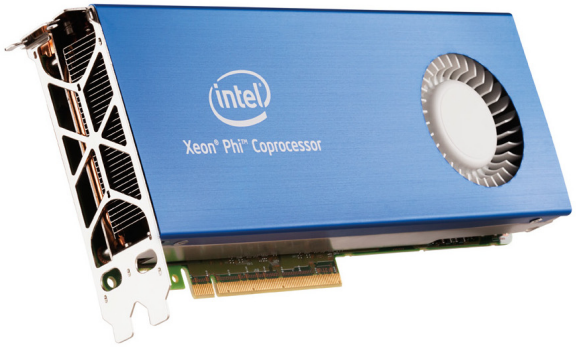
- Personal mid-size high memory and parallel computing.
- For people who moderate resources **constantly**.

Desktop

- Everything, depending on how long you are willing to wait.
- Will run 95% of what you want to do.

General Purpose GPU Computing

- Problems that break down to small, interdependent parts.
- Bootstrapping, complex looping, optimization.



Pricing tiers

- **Cluster Access :** Usually free through your institution but often requires application/faculty sponsorship.
- **HPC Workstation:** \$8,000-\$15,000 – Not a good investment for most.
- **Desktop:** \$700-\$2000 – Often a very good investment.

My suggestion

- Ask a faculty member for access. **TRY THIS FIRST.**
- Investing in a desktop with 4C/8T (Intel i7) and 16GB of ram is often a smart idea if it will **not get in the way of conference attendance**.
- Access a university cluster only once you are confident a desktop can no longer meet your needs.

Upgrades

- Old computers are good for HPC tasks that simply take a while to run.
- Locate computer in academic office for free electricity/internet/easier remote access.
- Relatively cheap upgrades can dramatically improve performance.
- **BENCHMARK**

Know your motherboard

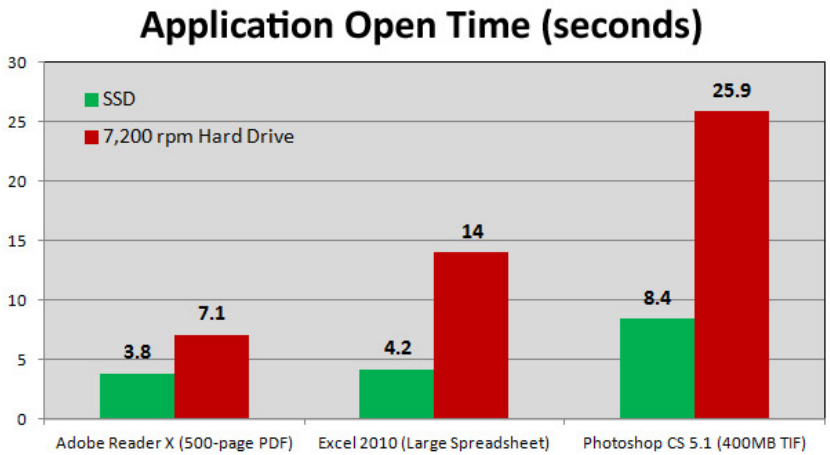
- How many RAM Slots?
- Peripherals, CPU, GPU slots.

RAM

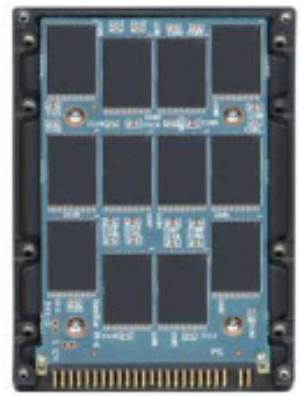
- Work with larger datasets.
- 16GB Kits – ($100-150)
- 32GB Kits – ($200-300)

Solid state hard drive

- General system performance, data I/O.
- \$0.40-\$0.80 per GB
- Leave 15-20% free.



Traditional hard disk drive



Solid state hard drive

Check review sites

Things to remember

- Get an Uninterrupted Power Supply (UPS) for stable power.

- Put a sign on your computer that says don't touch.

Summary

- Don't buy it unless you **absolutely need it**.
- Most resources can be borrowed/had for free.
- More powerful resources require **more time** to learn.

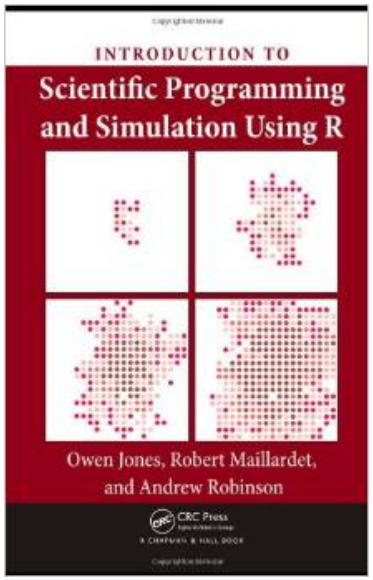
5. Resources

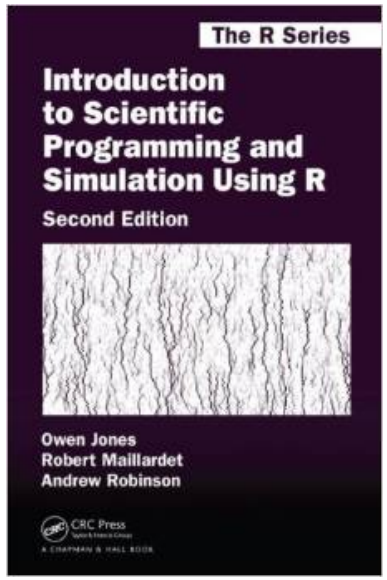
R HPC resources

- Tim Churches – parallelization tutorial

  Link

- *Introduction to Scientific Programming and Simulation Using R*





Rcpp/C++ resources

- Dirk Eddelbuettel – Rcpp
  http://www.rcpp.org/
- Hadley Wickham – Advanced R
  http://adv-r.had.co.nz/
- Armadillo library API documentation
  http://arma.sourceforge.net/docs.html