The BU Shared Computing Cluster for Economists

Johannes Schmieder

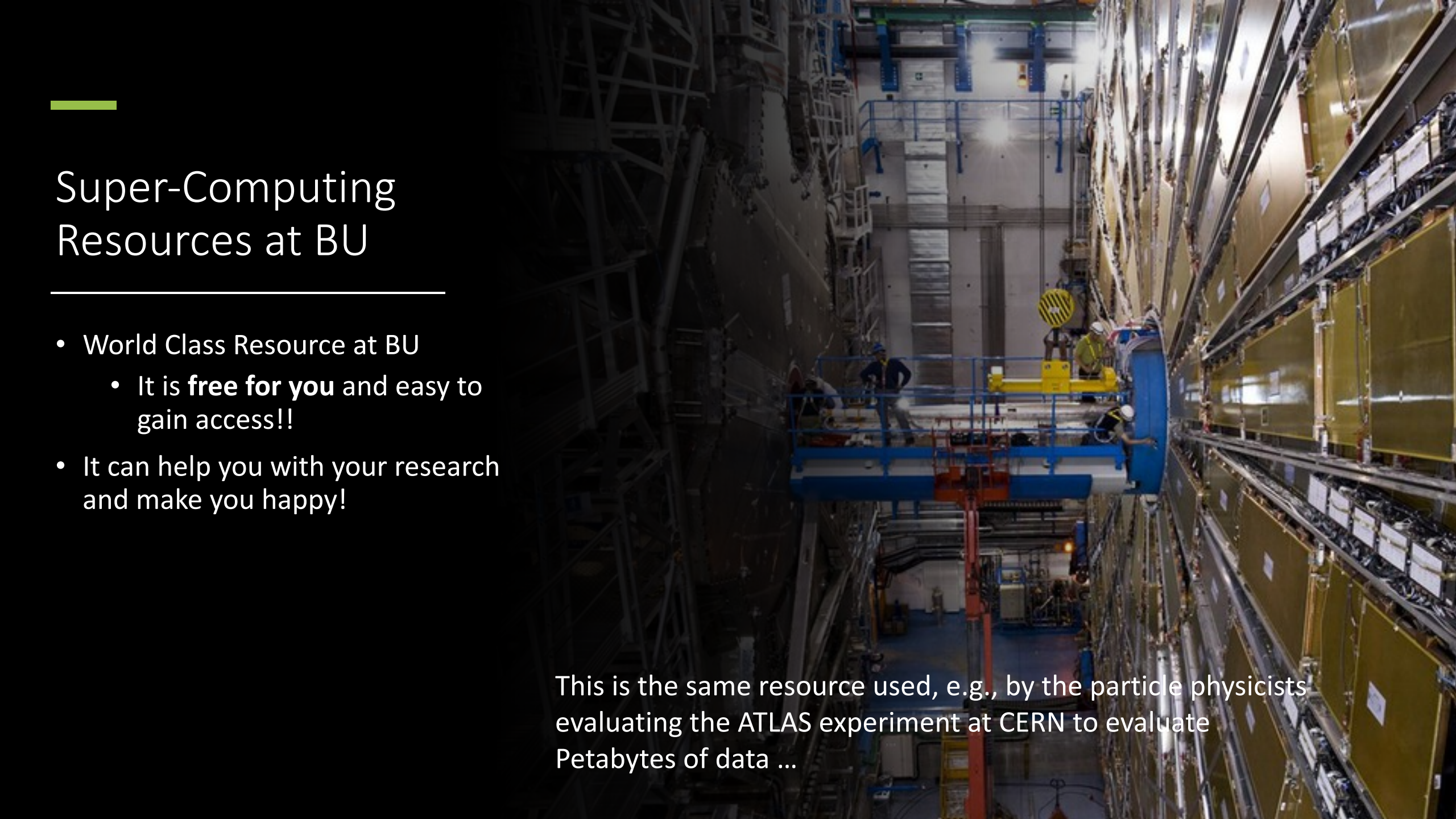
Empirical Micro Workshop – Fall 2020

Massachusetts Green High Performance Computing Center

- BU's research computing resources are housed at the MGHPCC.
- Collaboration with Harvard, MIT, Northeastern, UMass and BU.
- World-class computational infrastructure
- Powered by Green energy!

Super-Computing Resources at BU

- World Class Resource at BU
  - It is **free for you** and easy to gain access!!
- It can help you with your research and make you happy!

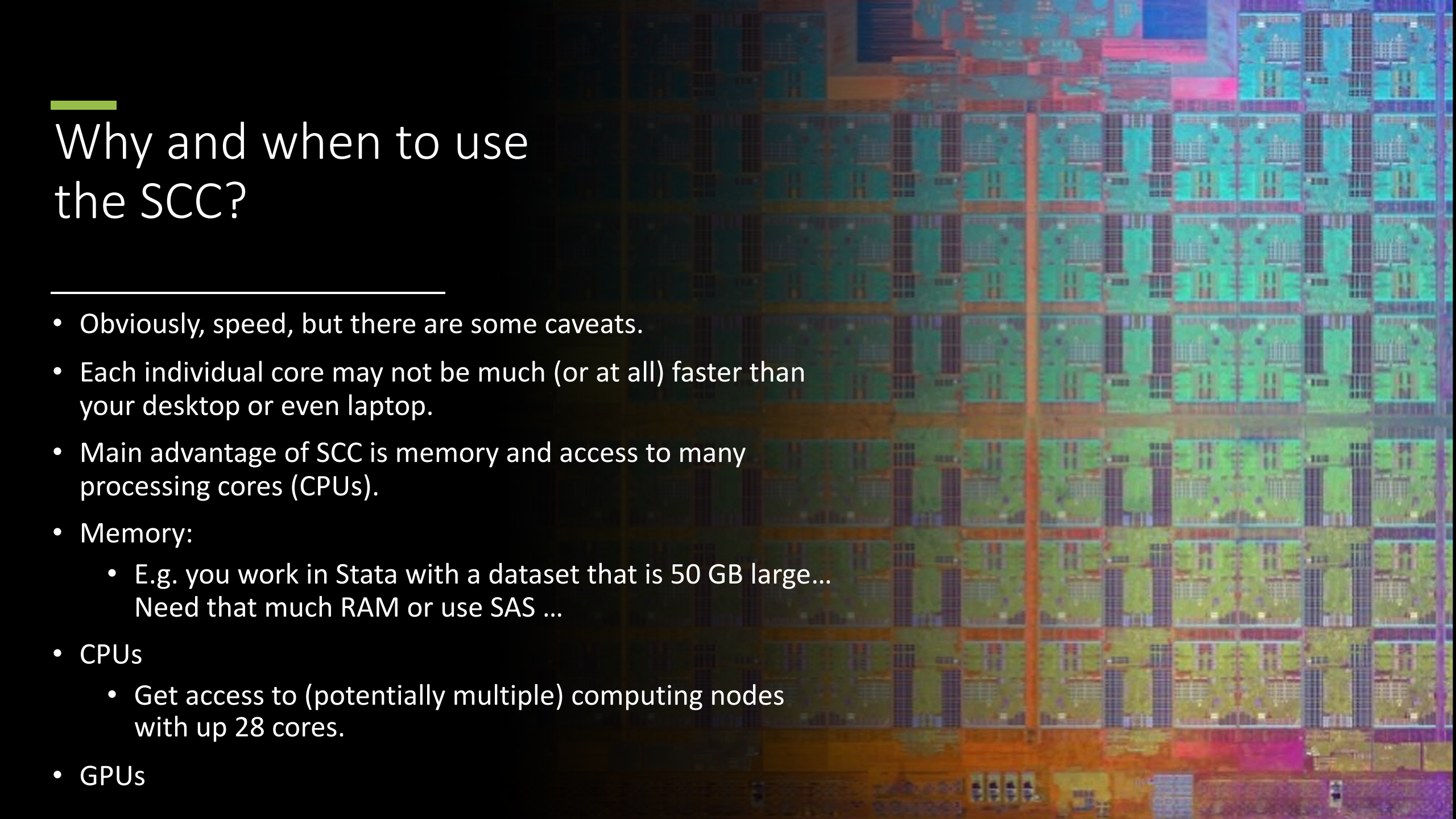


This is the same resource used, e.g., by the particle physicists evaluating the ATLAS experiment at CERN to evaluate Petabytes of data …

Advantages of using the BU Shared Computing Cluster (SCC)

- Access many processors simultaneously:
  - You can complete your computer jobs faster.
  - You can complete many jobs at once.
- Access from any computer (or phone?)
  - Check on your jobs.
  - Start them with new parameters.
- Super-computers don’t get restarted.
- Look cool – impress friends (and potential employers)!

Why and when to use the SCC?



- Obviously, speed, but there are some caveats.
- Each individual core may not be much (or at all) faster than your desktop or even laptop.
- Main advantage of SCC is memory and access to many processing cores (CPUs).
- Memory:
  - E.g. you work in Stata with a dataset that is 50 GB large… Need that much RAM or use SAS …
- CPUs
  - Get access to (potentially multiple) computing nodes with up 28 cores.
- GPUs

When do multiple CPUs help?

- Ideally: 2 x N(CPU) = 2 x Speed
- But most software is written to run on a single core (thread).
- To use multiple cores in **parallel** requires some thoughtful programming.
- For our purposes there are essentially 2 ways:
  - Implicit parallelization
    → your software environment does the work for you.
  - Explicit parallelization
    → you tell your software environment exactly what and how to parallelize.

Example 1: Stata

- Stata MP provides implicit (automatic) parallelization of your code.
  - E.g. matrix multiplication.
- Decent speed-up out of the box, no work needed.
- Stata MP licenses restrict number of cores.
  - On SCC MP 8 core available.
- Can run multiple jobs in parallel, …

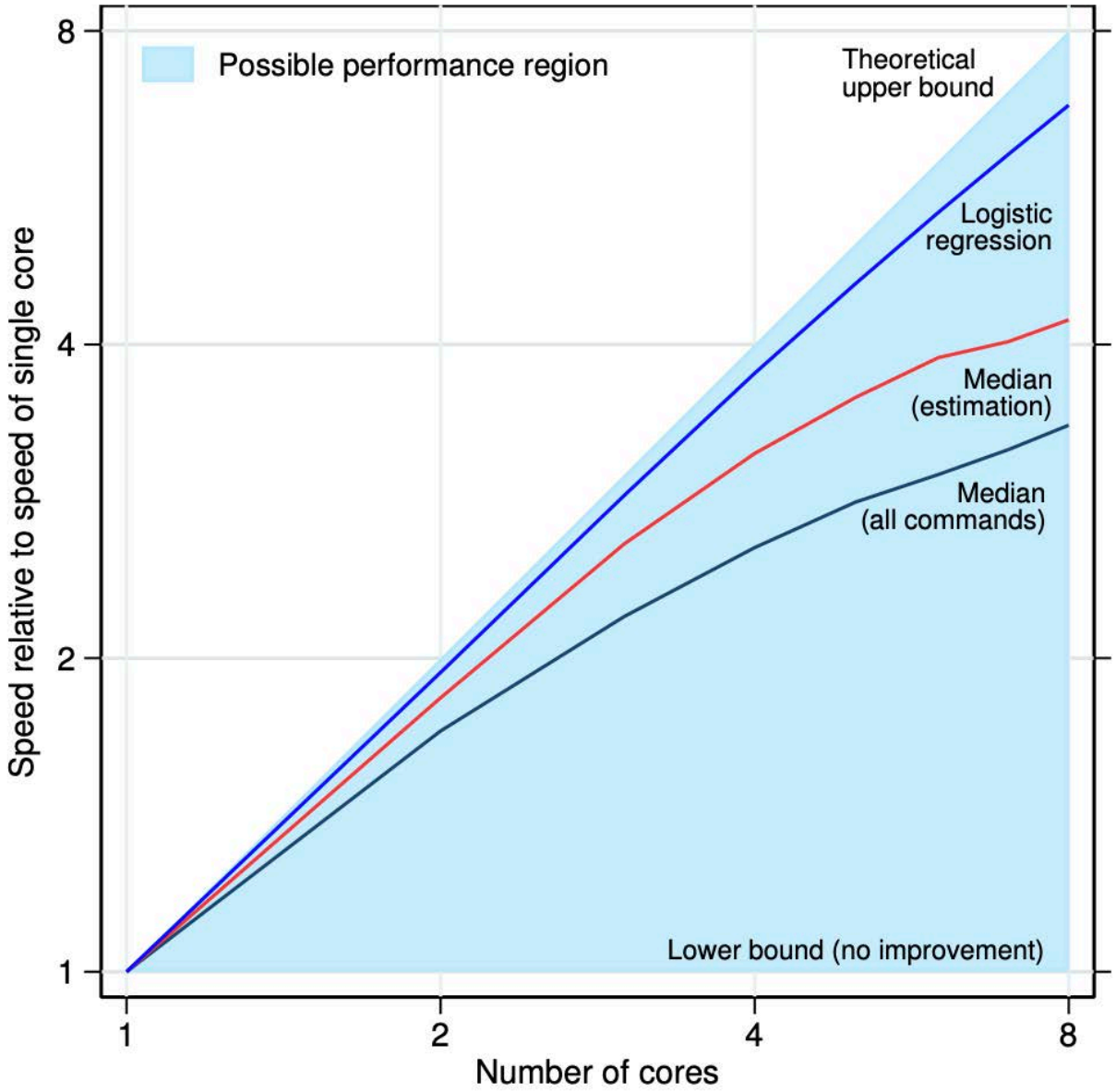


Figure 1. **Performance of Stata/MP.** Speed on multiple cores relative to speed on a single core.

Example 1: Stata

- Maybe particularly useful for working with large datasets.
  - Up to 1024 GB possible.
- Let jobs run for long time, …
- E.g. estimating model with many many fixed effects on hundreds of millions of observations … (AKM).

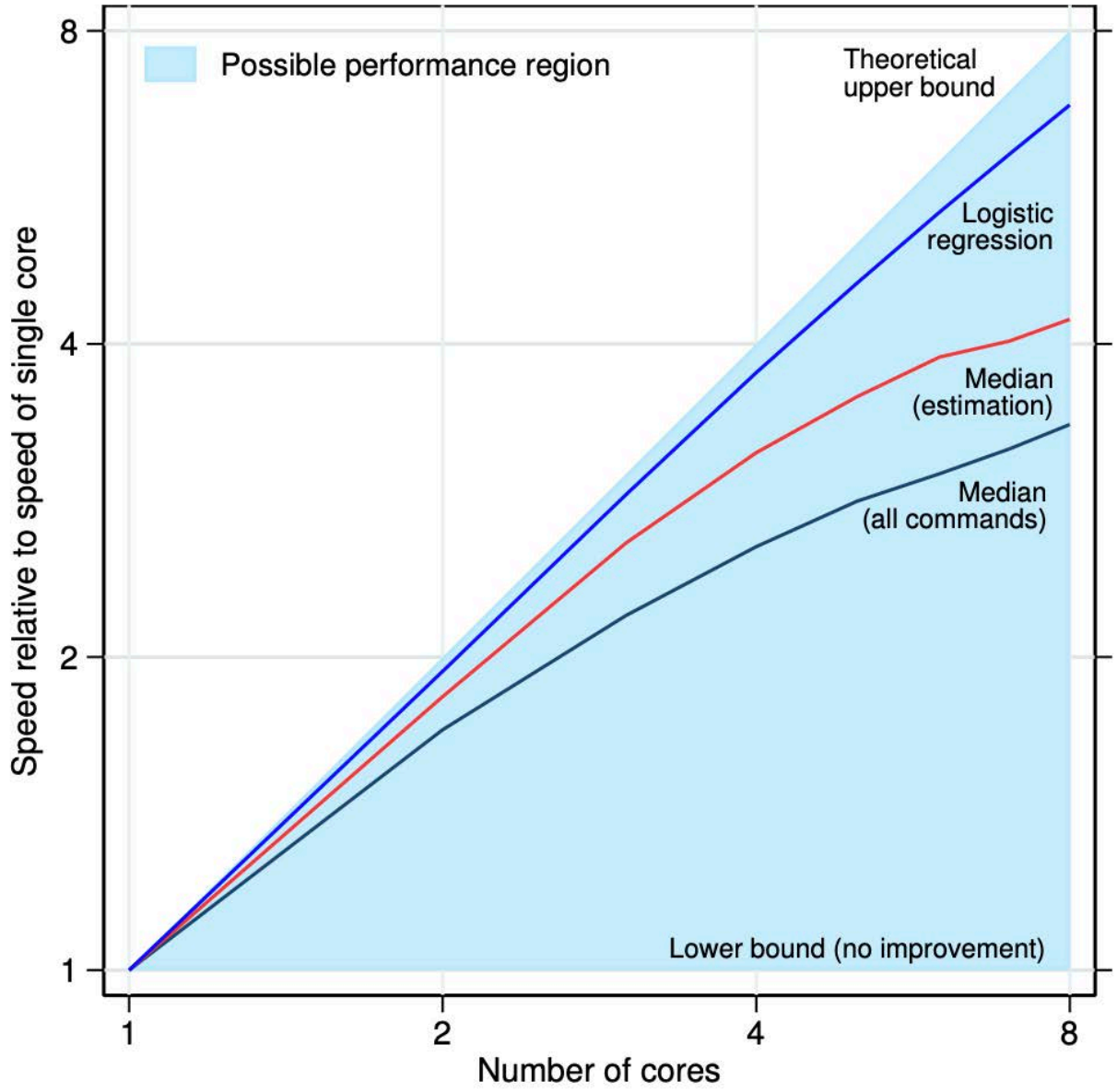


Figure 1. **Performance of Stata/MP.** Speed on multiple cores relative to speed on a single core.

Example 1: Stata – Explicit Parallelization

- User created tool: parallel
  - https://github.com/gvegayon/parallel
- Allows you to break out work into individual pieces and run in parallel.
- This works by spawning stata child processes that run their own version of Stata. The parallel package automates this process.
- Suppose you have a large dataset to clean.
- Parallel package makes it easy to run cleaning code on separate slices of data in parallel.
- Also works well for bootstrapping.

Stata: parallel package -- Bootstrapping

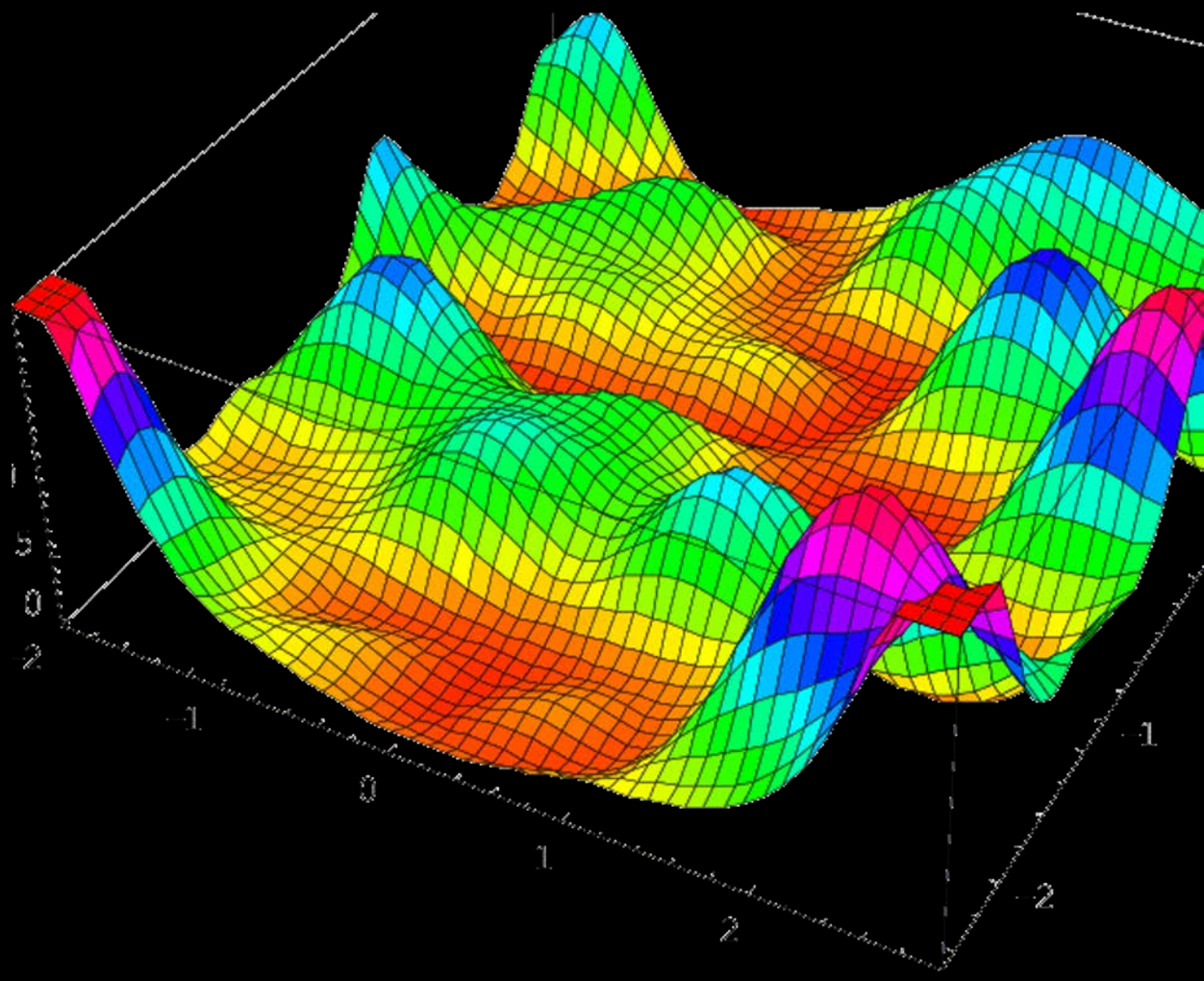
```
clear
sysuse auto

timer clear

set processors 1
timer on 1
bs, reps(5000) nodots: reg price c.weig##c.weigh foreign rep
timer off 1

set processors 4
timer on 2
bs, reps(5000) nodots: reg price c.weig##c.weigh foreign rep
timer off 2

parallel initialize 4, f
timer on 3
parallel bs, reps(5000): reg price c.weig##c.weigh foreign rep
timer off 3

timer list
```

```
. timer list
   1:     12.50 /        1 =      12.5040
   2:     13.03 /        1 =      13.0340
   3:      5.29 /        1 =       5.2950
```

Speedup around by around x2.4.
No benefit from MP alone

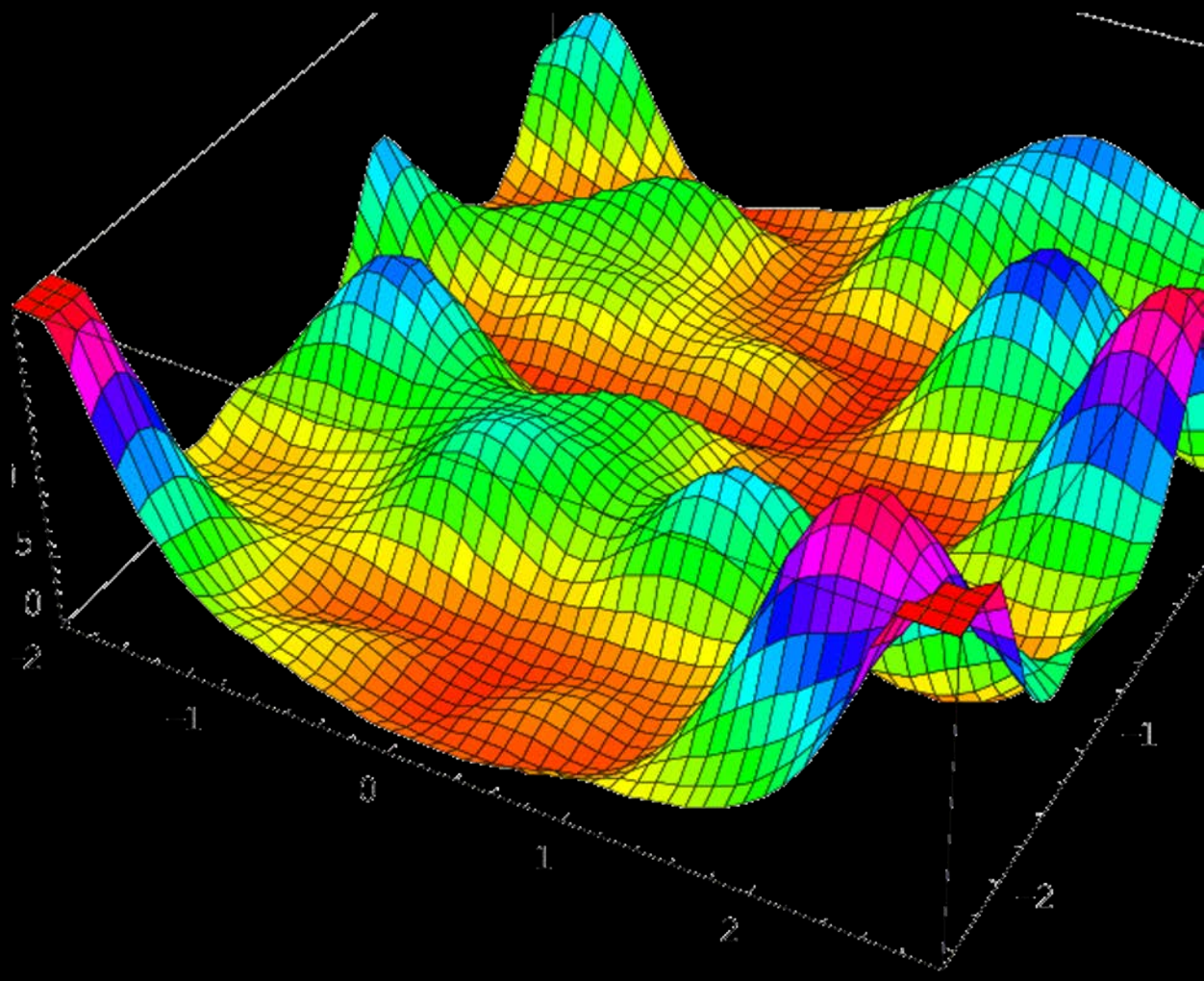
Example 2: Matlab – Global Optimization

- Suppose you want to estimate a complicated dynamic model via GMM.
- Optimization problem:
  $\min_{\xi} \ (m(\xi) - \hat{m})' W (m(\xi) - \hat{m})$
- If objective function is convex this is easy.
  - Just use minimizer (e.g. fmincon) and you should be good.
- But what if this function is non-convex?
  - With large parameter vector ξ this becomes a very complicated problem (curse of dimensionality).



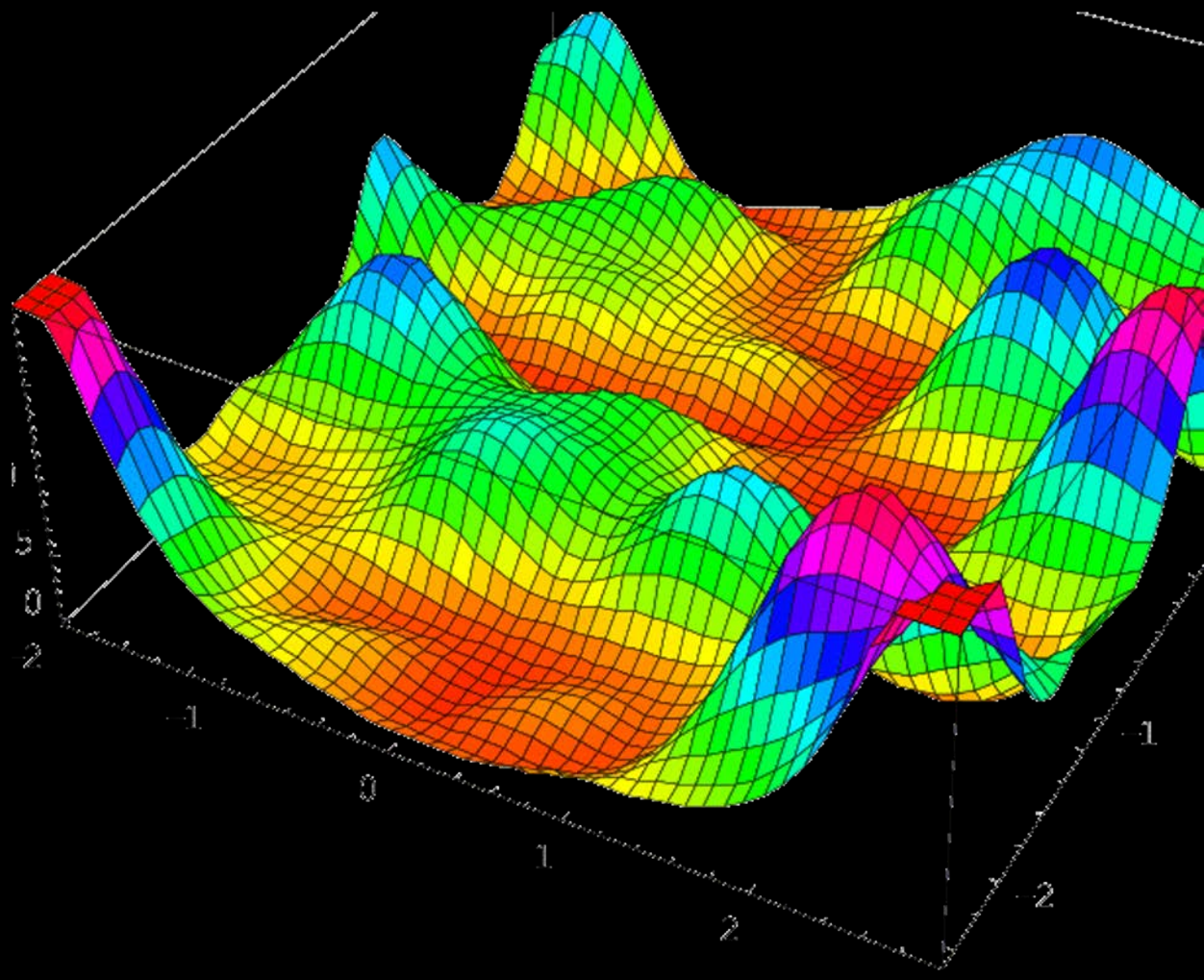
Example 2: Matlab – Global Optimization

- Optimization problem:
  $\min_{\xi} \ (m(\xi) - \hat{m})' W (m(\xi) - \hat{m})$
- If non-convex need to search through parameter space in some way.
  - E.g. pick 1000 random starting vectors and run local minimizer on them.
  - Best overall point might be close to global optimum.
- This lends itself well to parallelization.



Example 2: Matlab – Global Optimization

```
parfor j = 1:noSearchInits
    x0 = searchInits(:,j);
    options = optimset('Display', 'iter','Algorithm','interior-point');
    try
     [paramHats(:,j), sse(j), exitFlag(j),outputf(j)] = ...
          fmincon(@objFun,x0,[],[],[],[],lb_res,ub_res,[],options)
    catch
     warning('Problem in fmincon');
     disp('Initial Values: ');
     disp(x0);
     exitFlag(j) = -1;
    end
end
```



Example 3: Python

- Similarly to Stata and Matlab, there are many ways to parallelize code.
  - Implicit: Underlying numerical libraries parallelize some operations. E.g. MKL, OpenBlas, …
  - Unless you work with very large matrices this is neat, but also probably not dramatically helpful.
- More promising is explicit parallelization as in Matlab.
- E.g. Global Maximization with PyGMO
  - PyGMO is a neat global optimization library.
  - Developed by the **European Space Agency** to find interplanetary spacecraft trajectories.

Example 3: Python - PyGMO

- Global Maximization with PyGMO
  - Easy to set up maximization problem that is distributed over many cores.
  - Idea that each core is an island that runs an optimization algorithm over potentially many initial starting values.
  - Each round the candidates from each island are moved between islands.
  - This way different algorithms can work together to find the best solution:
    - Gradient based algorithms (Newtonian,…),
    - Genetic algorithms,
    - Particle swarms,
    - Simulated Annealing, …

Some optimization can go a long way

- Before throwing lots of cores at a problem it's worthwhile spending some time optimizing your code.
- Vectorization:
  - Explicit loops (for i in (1,2,3) …) are very slow in interpreted languages like Matlab and Python.
  - Much faster to use operations on vectors since those are heavily optimized in underlying C code.

```
i = 0;
tic
for t = 0:.01:100
  i = i + 1;
  y(i) = sin(t);
end
toc
```

Elapsed time is 0.003418 seconds.

```
tic
x = sin(0:.01:100 );
toc
```

Elapsed time is 0.001414 seconds.

Compiling Code for Speed

- Interpreted languages usually come with a big speed penalty .
  - One way to avoid this is to program in a compiled language like C or Fortran.
  - Downside: you have to learn C or Fortran.
  - Programming in C or Fortran is a bit harder. E.g. need to know about pointers, memory allocation, …
  - You lose the ease of use / debugging of a Matlab or Python environment
- Various options:
  - Matlab → Mex functions
  - Stata → Mata
  - Julia → NKOTB
  - Python → Numba

Compiling Python: Numba

```python
import numpy as np
from numba import njit, prange

A = np.random.rand(10**7)

''' Simple Python Implementation '''
def fun_python(A):
    s = 0
    for i in range(A.shape[0]):
        s += A[i]
    return s
```

Speed Comparison:

- Python only -- Elapsed Time: 1.909
- Numba plain -- Elapsed Time: 0.012
- Numba Prange -- Elapsed Time: 0.003

```python
''' Numba – Just in Time Compilation'''
@njit
def fun_numba(A):
    s = 0
    for i in range(A.shape[0]):
        s += A[i]
    return s

''' Numba – parallel w/ prange '''
@njit(parallel=True)
def fun_numba_prange(A):
    s = 0
    for i in prange(A.shape[0]):
        s += A[i]
    return s
```

Compiling Python: Numba

- Numba can dramatically speed up your code.
- Real world example:
  - Structural life-cycle model of labor supply
  - Numba runs about 25 times faster than pure Python or Matlab code.
- Numba gets close to the performance of writing directly in C or Fortran but with all the benefits of doing so in a Python environment.

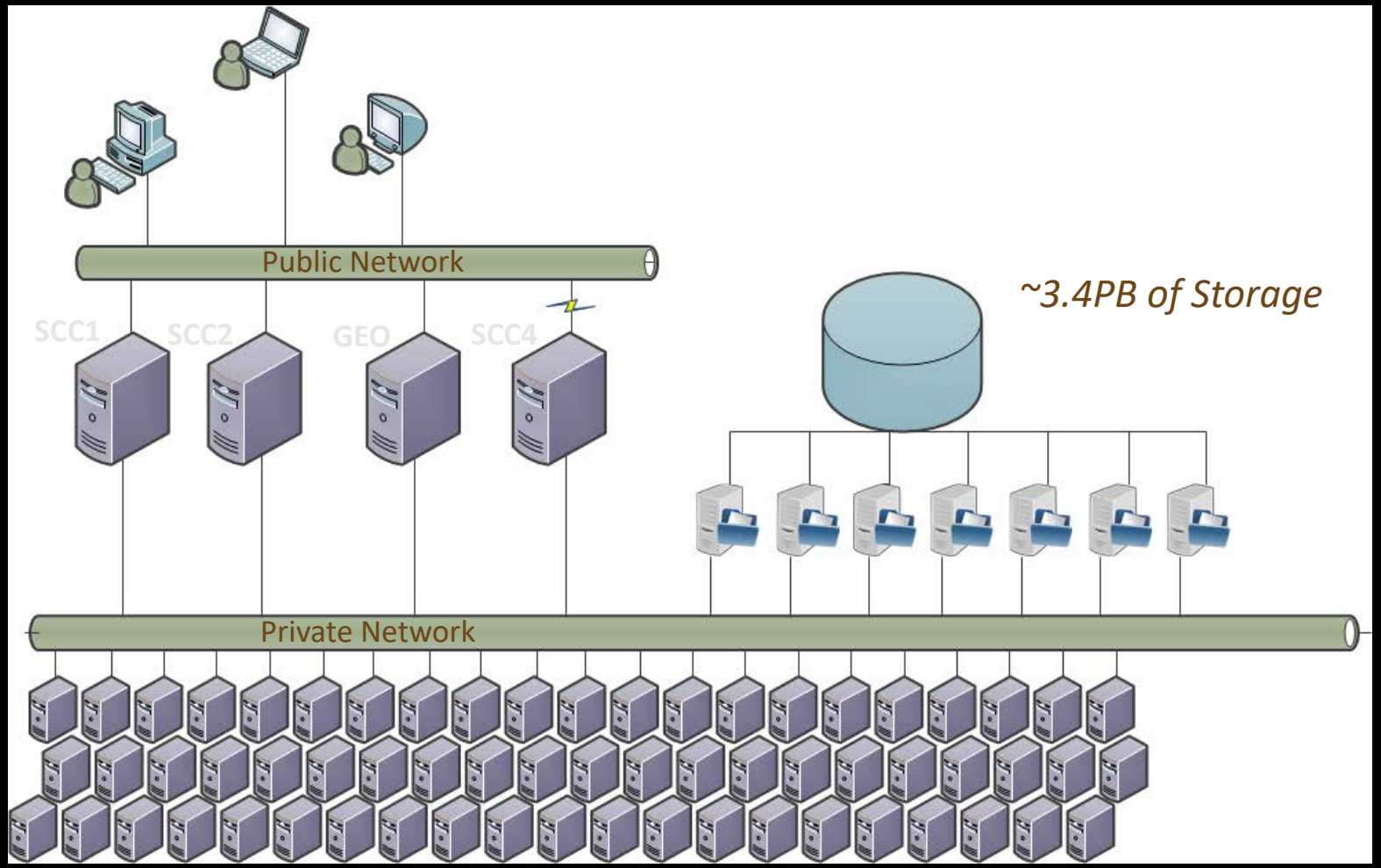
						Runtime Ratios	
Steps	Runs	Matlab	Pure Python	Numba		Matlab / Pure	Matlab / Numba
1	100	0.938	0.950	0.037		0.99	25.39
6	10	8.988	6.022	0.748		1.49	12.01
12	10	312.820	41.768	11.219		7.49	27.88

How to get Access

- Graduate students cannot have their own account.
- You can be added to a faculty account, or to the department account.
  - Use faculty accounts for *joint work or research assistance* with a faculty member.
  - Use the department account for *your own work*.
- To get access:
  Send e-mail to the RCS Liaison with your full name, your login name (Kerberos), and the e-mail address that you want to use.
  Currently Stephen Terry is the RCS Liaison.

SCC organization

Login nodes



Compute nodes

Around 900 nodes with ~12,000 CPUs and ~200 GPUs

Remote Access

- There are several ways (SFTP, SSH, …).
- Easiest way is browser based version:
  - scc-ondemand.bu.edu
  - → Amazing! Can access SCC from anywhere (phone, ipad, laptop,…) without extra software!
- You can open interactive versions of most software environments right there in your browser!
- You can also upload and download filed, access a terminal, start scripts, …

SCC Queue system

- The SCC uses a centralized job management system to allocate computing resources called “qsub”.
- You submit a job to qsub specifying the resources you need and the script to execute.
  - E.g. “need 16 cores, 96 GB of RAM, and want to run structuralEst.m using matlab”
- This is done using the “qsub” command and by writing a bash script that provides some additional settings.
- This adds your job to the queue and once the resources are available your job starts automatically.

Typical workflow might be:

- Develop your script (do file, matlab, python, …) either on your local machine or in an interactive session.
  - E.g.: start_structural_model.m
- Write a bash script that calls the matlab script.
  - E.g.: run_batch
- Submit the bash script to the SCC Queuing system “qsub”.
  - E.g.: qsub -pe omp 16 ./run_batch
  - Note “-pe omp 16” means that you are requesting 28 computing cores.
- This will submit the job request to qsub where it now lines up in the queue. As soon as a node with 16 cores is available the job will start running.
  - You can configure the “run_batch” file so that you receive an email when the job starts and finishes.

Example Set-up for SCC Job Submission

Command to submit script to qsub:

```
qsub -pe omp 28 ./run_batch
```

File: run_batch

```
#!/bin/bash -l
#$ -pe omp 28
# set default value for n; override with qsub -v at runtime
#$ -M johannes.schmieder@gmail.com
#$ -m beas

#$ -N StructEst_Test

# Load the newest version of matlab on SCC
module load matlab
# Additional qsub options here . . .
matlab -nodisplay -r "runBatchJob($NSLOTS); exit"
```

File: runBatchJob -- Matlab file to start main program

```
function runBatchJob(n, nslots)
    % redirects  /.matlab PCT temp files to TMPDIR on the compute
    % node to avoid inter-node (compute node <--> login node) I/O
    myCluster = parcluster('local') % cores on compute node to be
"local"
    if getenv('ENVIRONMENT')    % true if this is a batch job
      myCluster.JobStorageLocation = getenv('TMPDIR')  % points to
TMPDIR
    end

    % Create a parallel pool with the number of CPU cores requested
on the cluster
    parpool(myCluster, nslots)

    % Run main matlab code
    estimate_structural_model

    % Shut down parallel pool
    delete(gcp('nocreate'));
end
```

Example Set-up for SCC Job Submission

- See the queue:
  - qstat
- See the jobs submitted under your username:
  - qstat -u <username>
- Delete a job:
  - qdel jobNumber.

Resource Limits

- Cores: most nodes have 8, 16 or 28 cores. You should write your code targeting these numbers and being mindful of what you need.
  - Requesting 17 cores makes no sense since you are effectively blocking a 28 core machine.
  - Requesting 16 cores if you only run StataMP (8-core) also makes no sense.
- Wall clock time limit:
  - On the shared computing nodes jobs will automatically terminate after 12 hours.
  - You can request longer run times, but there are fewer nodes that offer that.
  - Write your code so that not everything is lost if it does not finish by then.

A little Linux goes a long way …

- The cluster runs on Linux.
- Worthwhile to work through a little tutorial.
- Learn some basic commandline commands:
  - ls
  - Cp
  - Mv

Conclusion

- There is some learning curve but a huge payoff.
- The research computing service team offers great support and documentation:
  - http://www.bu.edu/tech/support/research/
  - http://www.bu.edu/tech/support/research/system-usage/running-jobs/
  - By reading online you can learn a lot.
  - RCS also offers classes for scientific computing taught be experts (Matlab, Python, R, GPU programming, Linux, …)
- Ultimately you learn by doing and experimenting.