北京大学
校园行
高性能计算系列讲座

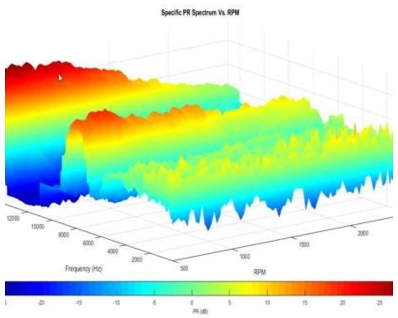
MATLAB Parallel Computing & Big Data

阮卡佳

Practical application of parallel computing

- Why parallel computing?
  - Need faster insight to bring competitive products to market quickly
  - Accelerate your research or product development
  - Computing infrastructure is broadly available (multicore desktops, GPUs, clusters, and clouds)

- Why parallel computing with MATLAB and Simulink
  - Accelerate workflows with minimal to no code changes to your original code
  - Focus on your engineering and research, not the computation



Automotive Test Analysis and Visualization
3-4 months of development time saved

Heart Transplant Studies
4 weeks reduced to 5 days
6X speedup in process time

Design and Build Wave Energy Farm
Sensitivity studies accelerated 12x

Discrete-Event Model of Fleet Performance
Simulation time reduced from months to hours
20X faster simulation time
Linkage with Neural Network Toolbox

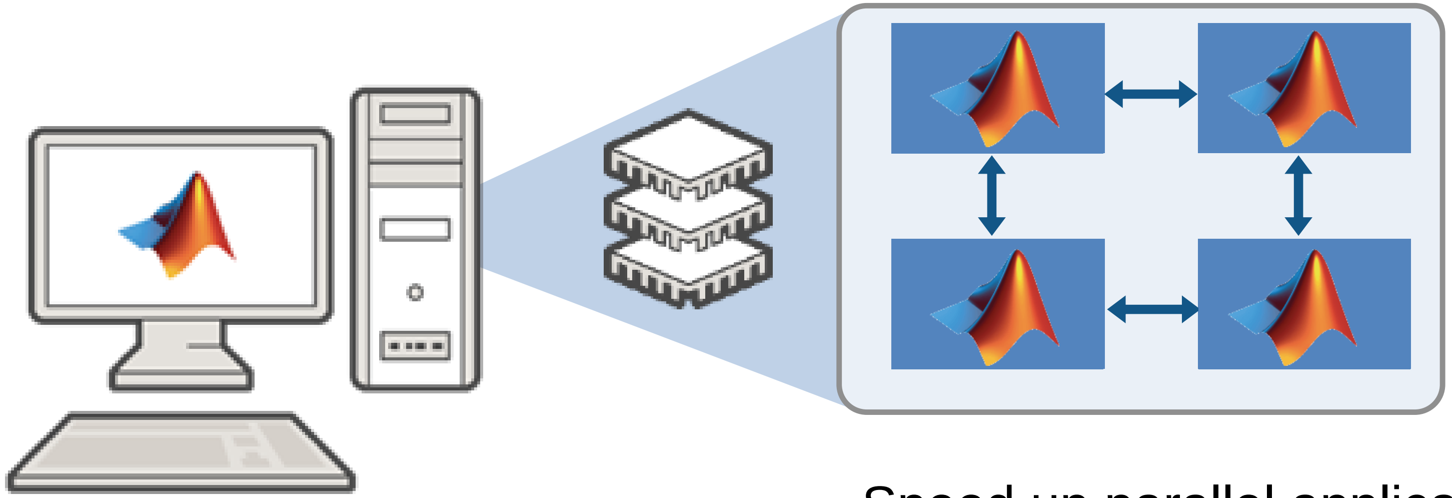
Calculating Derived Market Data
Implementation time reduced by months
Updates loaded 8X faster

MATLAB Parallel Computing

Parallel Computing Paradigm

Multicore Desktops

Multicore Desktop with GPUs



Parallel Computing Toolbox

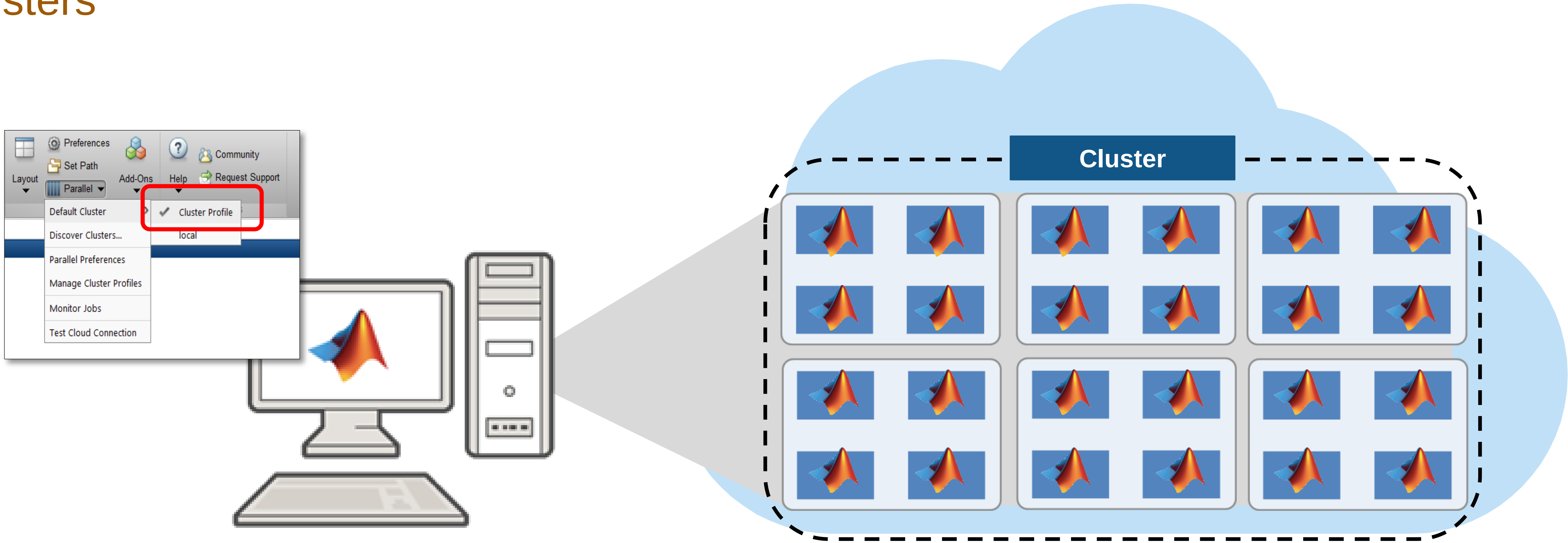
Speed up parallel applications

Take advantage of GPUs

Prototype code for your cluster

Parallel Computing Paradigm

Clusters



Accelerating MATLAB and Simulink Applications

Parallel-enabled toolboxes

Simple programming constructs

Advanced programming constructs

Accelerating MATLAB and Simulink Applications

Ease of Use

Parallel-enabled toolboxes

Simple programming constructs

Advanced programming constructs

Greater Control

Parallel-enabled Toolboxes (MATLAB® Product Family)

Enable parallel computing support by setting a flag or preference

Image Processing

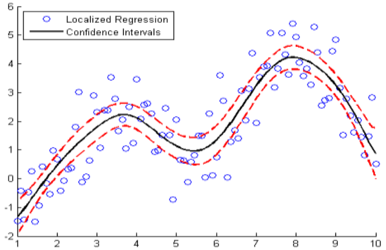
Batch Image Processor, Block Processing, GPU-enabled functions

Original Image of Peppers

Recolored Image of Peppers

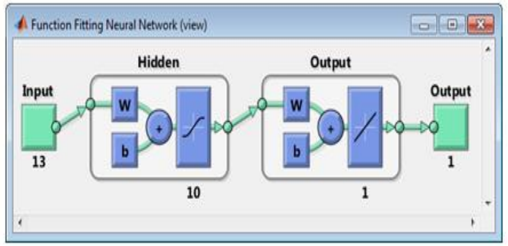
Statistics and Machine Learning

Resampling Methods, k-Means clustering, GPU-enabled functions



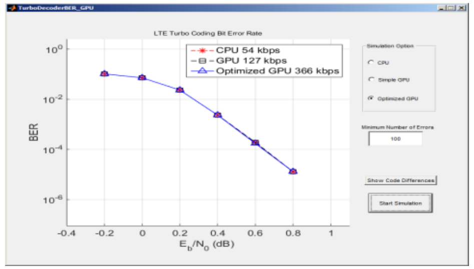
Deep Learning

Deep Learning, Neural Network training and simulation



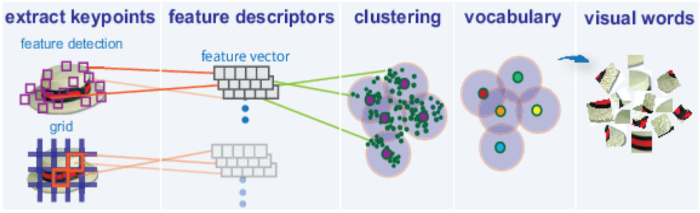
Signal Processing and Communications

GPU-enabled FFT filtering, cross correlation, BER simulations



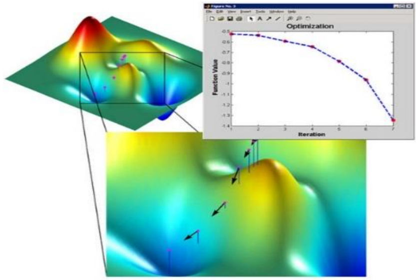
Computer Vision

Bag-of-words workflow



Optimization

Estimation of gradients

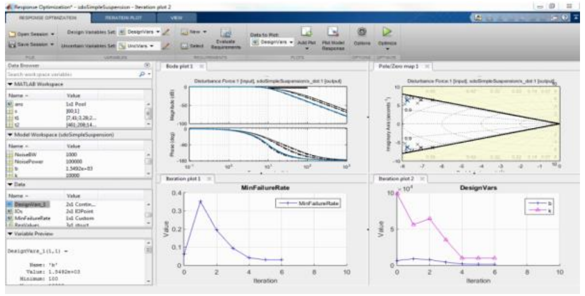


[Other Parallel-enabled Toolboxes]

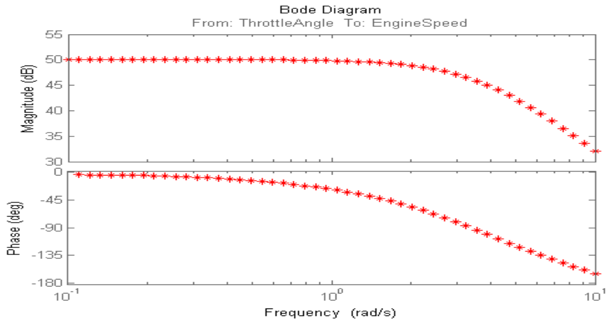
Parallel-enabled Toolboxes (Simulink® Product Family)

Enable parallel computing support by setting a flag or preference

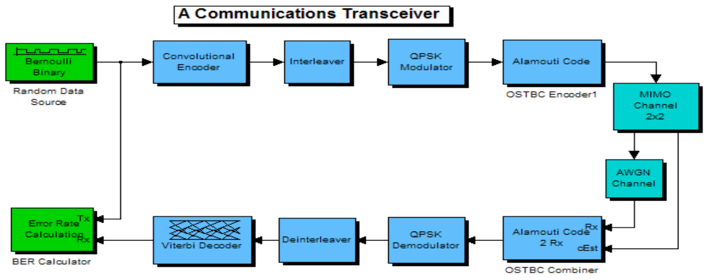
Simulink Design Optimization
Response optimization, sensitivity analysis, parameter estimation



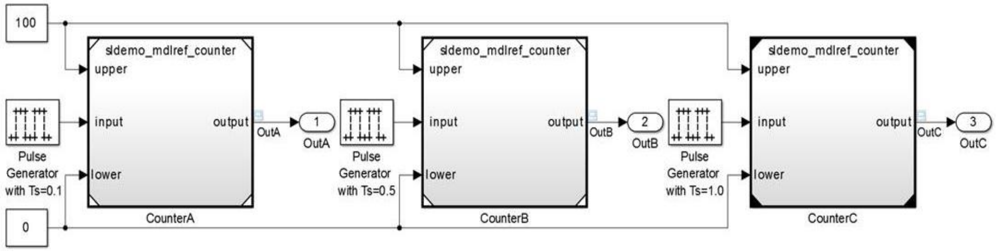
Simulink Control Design
Frequency response estimation



Communication Systems Toolbox
GPU-based System objects for Simulation Acceleration



Simulink/Embedded Coder
Generating and building code



Other Parallel-enabled Toolboxes

Accelerating MATLAB and Simulink Applications

Parallel-enabled toolboxes

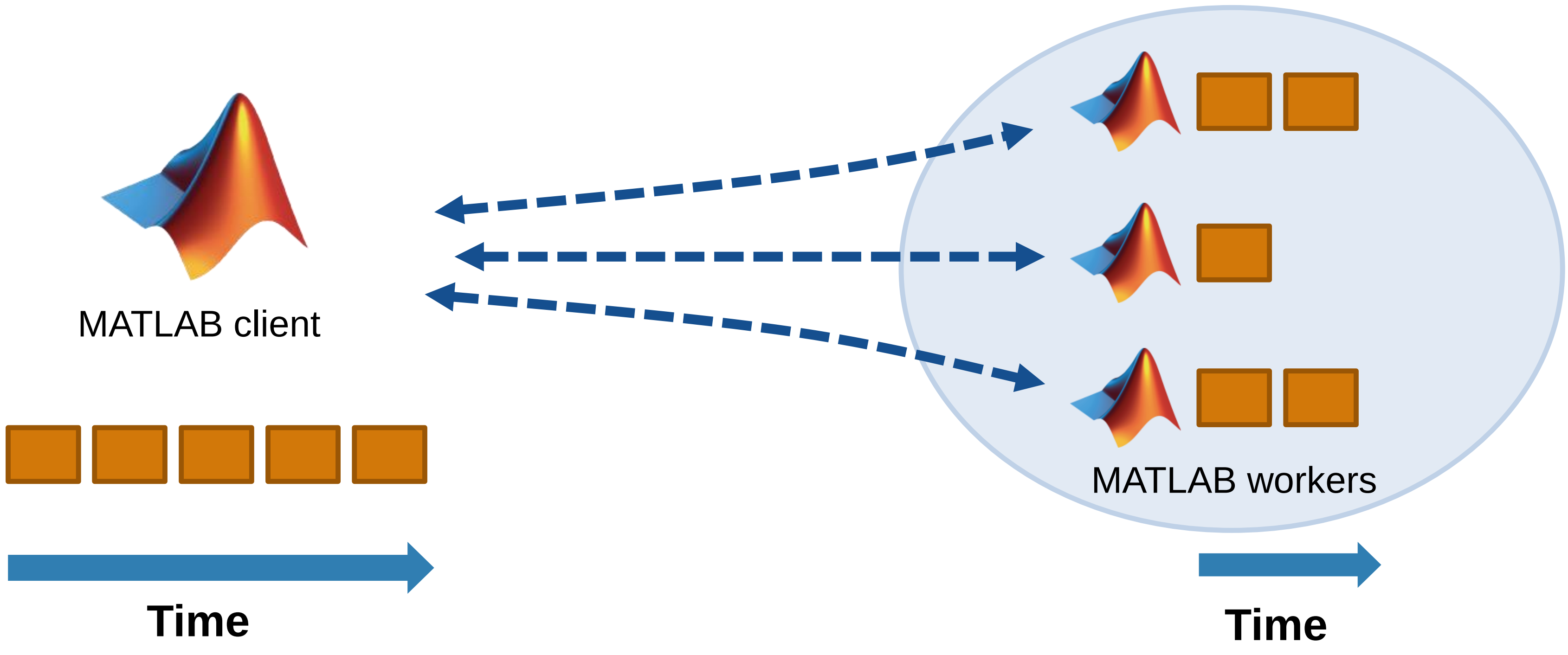
Simple programming constructs

Advanced programming constructs

Explicit Parallelism: Independent Tasks or Iterations

Simple programming constructs: `parfor, parfeval`

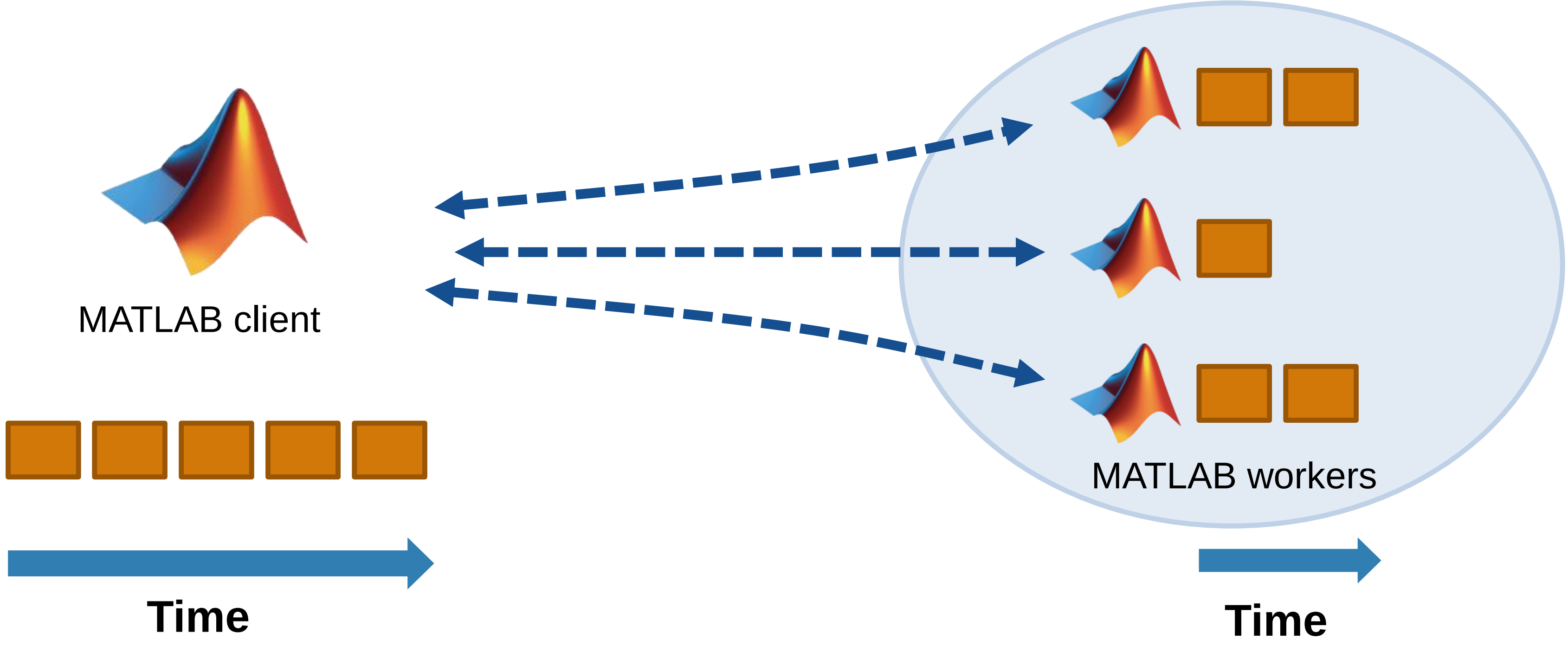
- Examples: parameter sweeps, Monte Carlo simulations
- No dependencies or communications between tasks



Explicit Parallelism: Independent Tasks or Iterations

```
for i = 1:5
      y(i) = myFunc(myVar(i));
end
```

```
parfor i = 1:5
      y(i) = myFunc(myVar(i));
end
```



Demo: Parameter Sweep

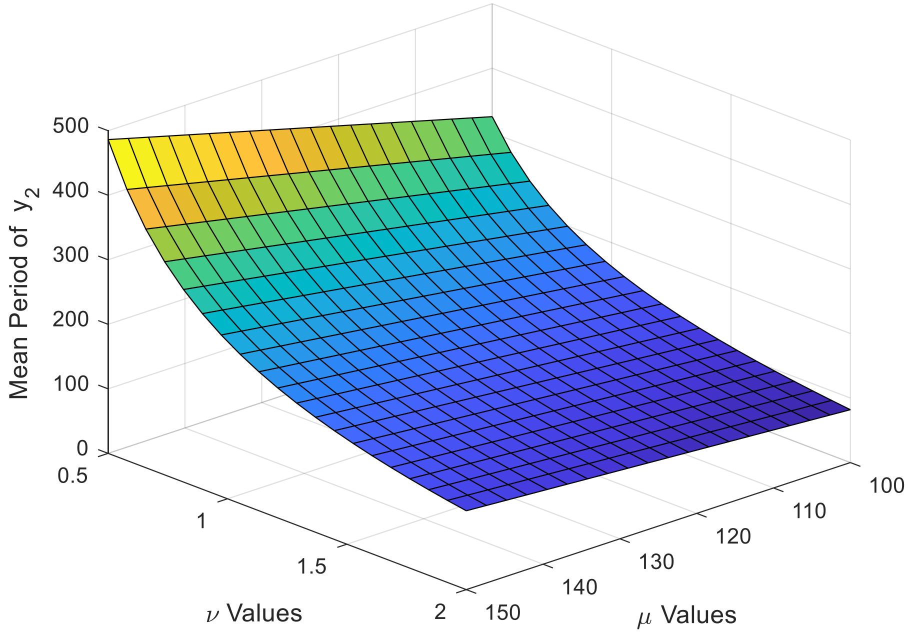
Using `parfor` with MATLAB for parameter sweep

Parameter sweep of Van Der Pol oscillator

- System of ODE used to challenge stiff solvers

$$\dot{y}_1 = \nu y_2$$
$$\dot{y}_2 = \mu(1 - y_1^2)y_2 - y_1$$

- Goal is to compute mean period of oscillator
- Parameters investigated: ν and μ
- Sweeping over ranges $[100, 150]$ and $[0.5, 2]$ of μ and ν respectively

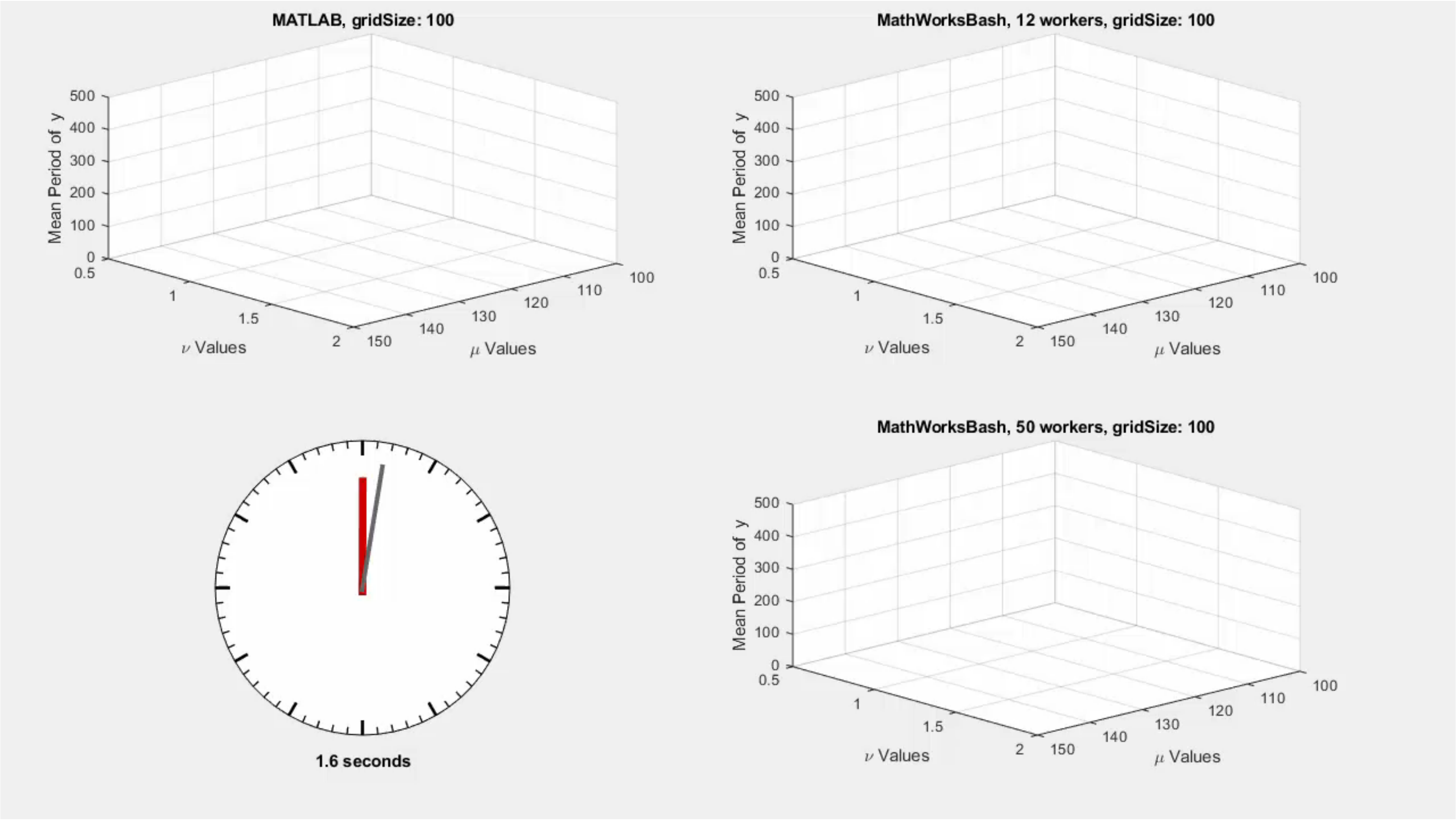


Speeding up in the cloud – same code, three environments

MATLAB Desktop

12 local workers Desktop

50 cloud workers AWS Cluster



parfor Limitation

Noninteger loop variable

```
parfor x = 0:0.1:1
    parfor y = 2:10
        A(y) = A(y-1) + ...
    end
end
```

Nested parallel loops

Dependent loop body

parfor Limitation

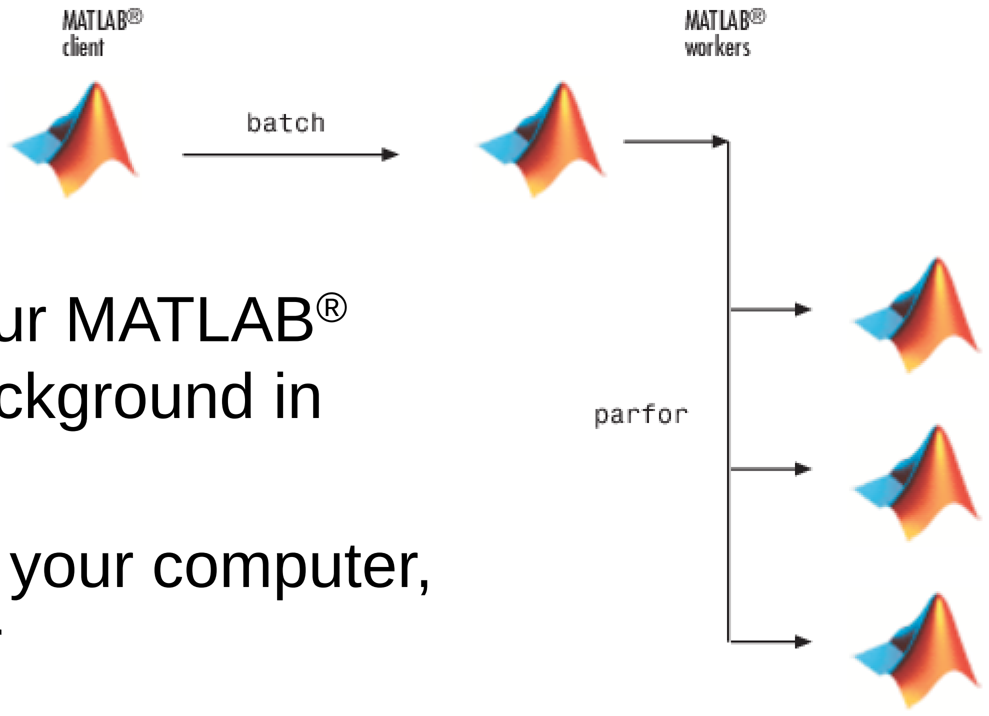
```
parfor iOuter = ...
    parfor iInner = ...
        ...
    end
end
```

```
parfor iOuter = ...
    for iInner = ...
        ...
    end
end
```

```
for iOuter = ...
    parfor iInner = ...
        ...
    end
end
```

`batch` simplifies offloading computations to a compute cluster

```
>>job = batch('mywave')
```

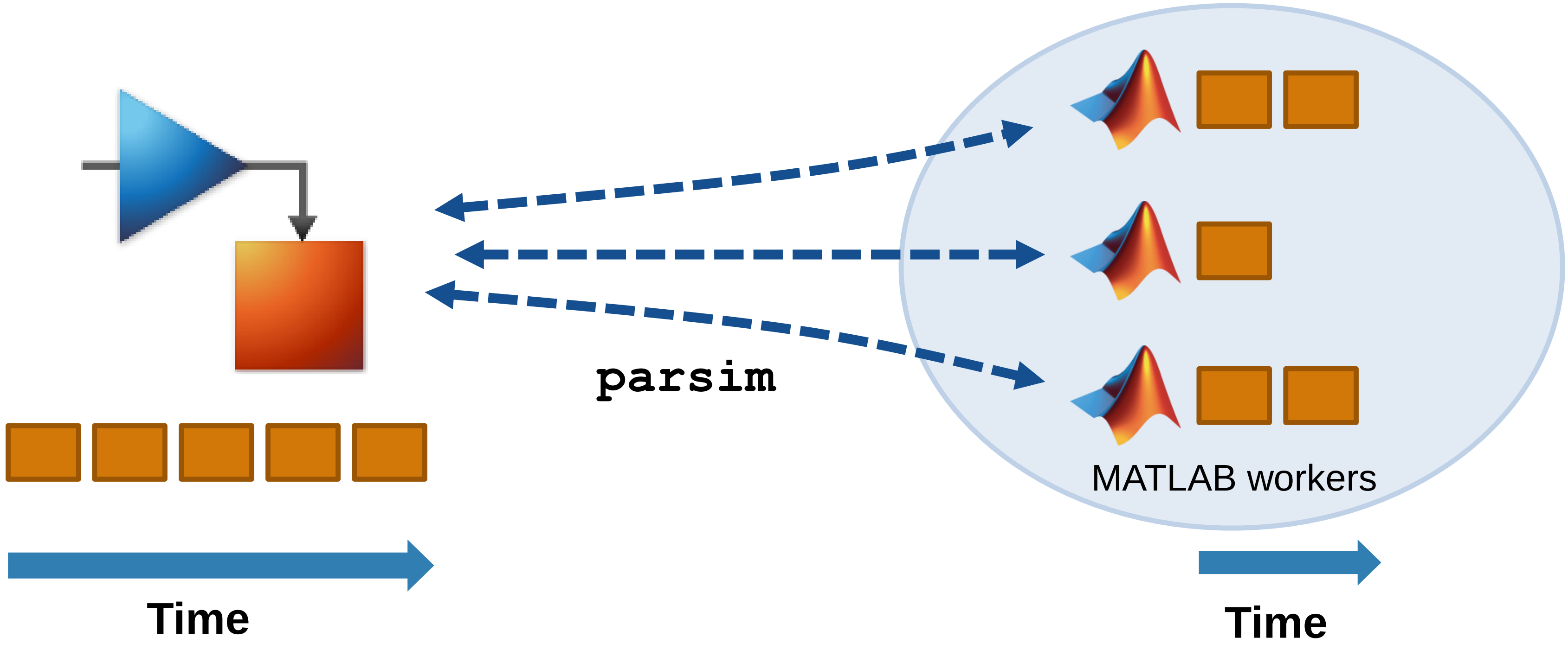


- To offload work from your MATLAB® session to run in the background in another session
- Submit jobs, shut down your computer, and access results later

Run Multiple Simulations in Parallel

- Run independent Simulink simulations in parallel using the `parsim` function

```
for i = 10000:-1:1
    in(i) = Simulink.SimulationInput(my_model);
    in(i) = in(i).setVariable(my_var, i);
end
out = parsim(in);
```



PKU 超算使用指南

1. PKU 超算 —— MATLAB 单节点并行

http://hpc.pku.edu.cn/docs/pdf/matlab.pdf

- 适用场景
  - 最多单节点32核并行，适合于计算密集型 MATLAB 代码
  - 可申请多个节点，节点之间计算独立

- 步骤
  1. 在本机开发 M 脚本
  2. 提交至集群（已有 matlab.sh 模板）

gpfs/share/example/app/MATLAB

1 MATLAB 代码

```
parpool('local',32)
% parfor i = 1:2048
A = zeros(100,1);
parfor i = 1:100
    A(i) = sin(i*2*pi/20);
end
save myResult.mat A
```

2 Shell 脚本

```
#!/bin/bash
#SBATCH -o job.%j.%N.out
#SBATCH --partition=C032M0128G
#SBATCH --qos=low
#SBATCH -J matlabJOB
#SBATCH --get-user-env
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --mail-user=xxxx@163.com
#SBATCH --mail-type=end
#SBATCH --time=1:00:00

module load matlab/R2019a
matlab -nodesktop -nosplash -nodisplay -r abc
```

2. PKU 超算 —— MATLAB 多节点并行
http://hpc.pku.edu.cn/docs/pdf/matlab.pdf

- 适用场景
  - 可进行多核跨节点并行，适合于计算和数据密集型 MATLAB 代码
  - 可申请多个节点，节点之间可实现数据交互
  - 每个节点最多 32 核，如申请 10 个节点，则为 319 核并行（有一个节点调度）

2. PKU 超算 —— MATLAB 多节点并行

http://hpc.pku.edu.cn/docs/pdf/matlab.pdf

- 步骤
  1. 在本机开发 M 脚本
  2. 在集群，创建提交脚本（SubmitTemplate_new.m）
  3. 提交至集群（已有 matlab_mps.sh 模板）

gpfs/share/example/app/MATLAB

1 MATLAB 代码

```
% parfor i = 1:2048
A = zeros(100,1);
parfor i = 1:100
    A(i) = sin(i*2*pi/20);
end
save myResult.mat A
```

2 Submit MATLAB 脚本

```
c.AdditionalProperties.JobName = 'matlabJOB';
c.AdditionalProperties.JotPriority = 'low';
c.AdditionalProperties.Mail = '***@163.com';
c.AdditionalProperties.Partition = 'C032M0128G
c.AdditionalProperties.NodeNum = '2';
c.AdditionalProperties.WorkerPerNode = '32';
c.AdditionalProperties.WallTime = '1:00:00';

j = batch(c,'abc_mps','Pool',63)      % Run a job f
exit
```

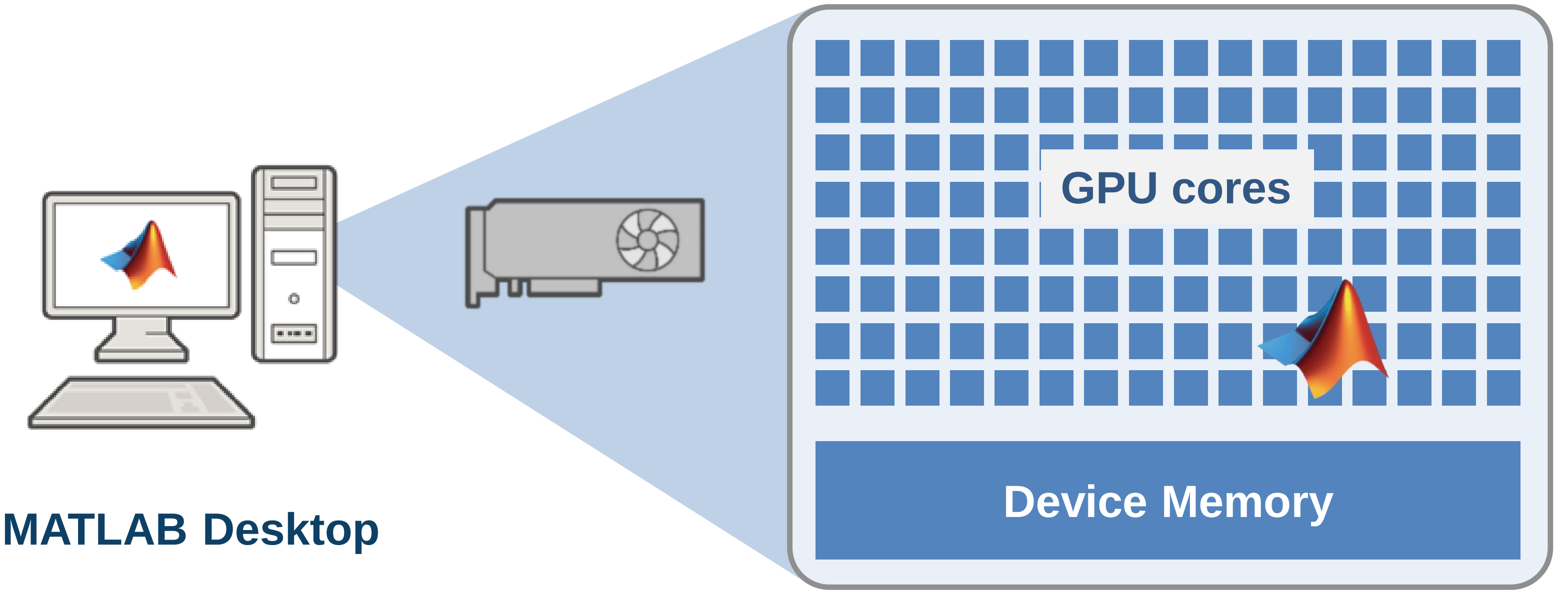
3 Shell 脚本

```
#!/bin/bash
#SBATCH -o job.%j.%N.out
#SBATCH --partition=C032M0128G
#SBATCH --qos=low
#SBATCH -J matlabJOB
#SBATCH --get-user-env
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --mail-type=end
#SBATCH --time=1:00:00

module load matlab/R2019a
matlab -nodesktop -nosplash -nodisplay -r SubmitTemplates_new -log LogFile.txt
```

Accelerate Application with GPU

Perform MATLAB computing on NVIDIA CUDA-enabled GPUs



Speed-up using NVIDIA GPUs

- Ideal Problems
  - Massively Parallel and/or Vectorized operations
  - Computationally Intensive
- 500+ GPU-enabled MATLAB functions
- Simple programming constructs
  - `gpuArray, gather`

Transfer Data To GPU From Computer Memory

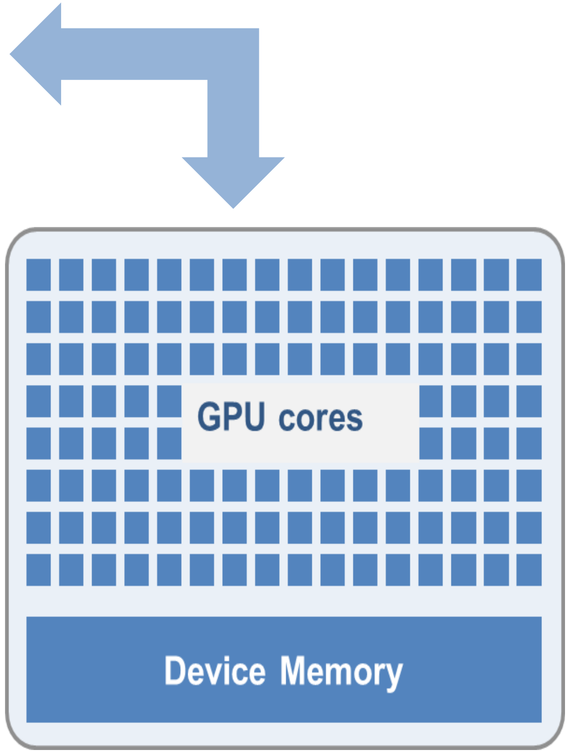
```
A=gpuArray(A);
```

Perform Calculation on GPU

```
X=exping(A);
```

Gather Data or Plot

```
X=gather(X)
```



GPU Enabled Toolboxes and Applications

Deep Learning

Statistics and Machine Learning

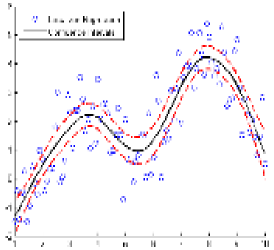
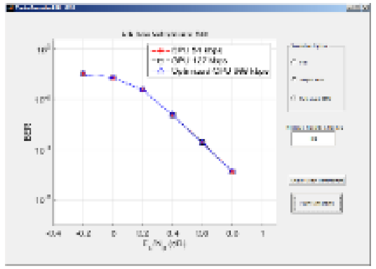
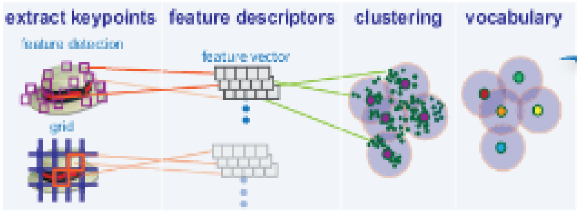


Image Processing

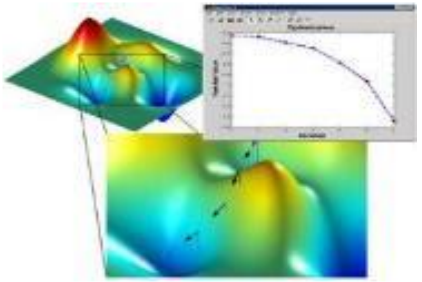
Signal Processing and Communications



Computer Vision

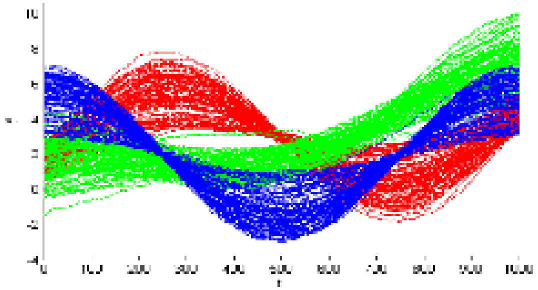


Optimization



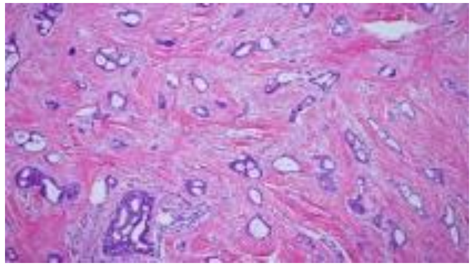
[Other Parallel-enabled Toolboxes]

Examples: GPU Accelerated MATLAB



10x speedup

K-means clustering algorithm

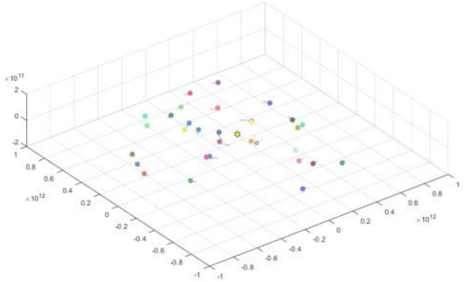


14x speedup

template matching routine

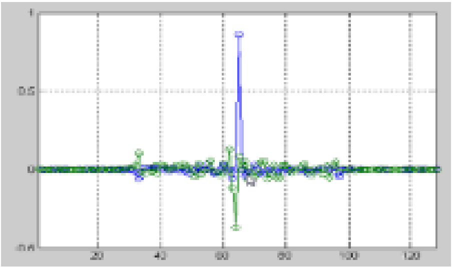
12x speedup

using Black-Scholes model



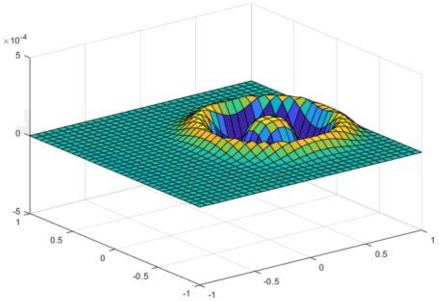
44x speedup

simulating the movement of celestial objects



4x speedup

adaptive filtering routine



77x speedup

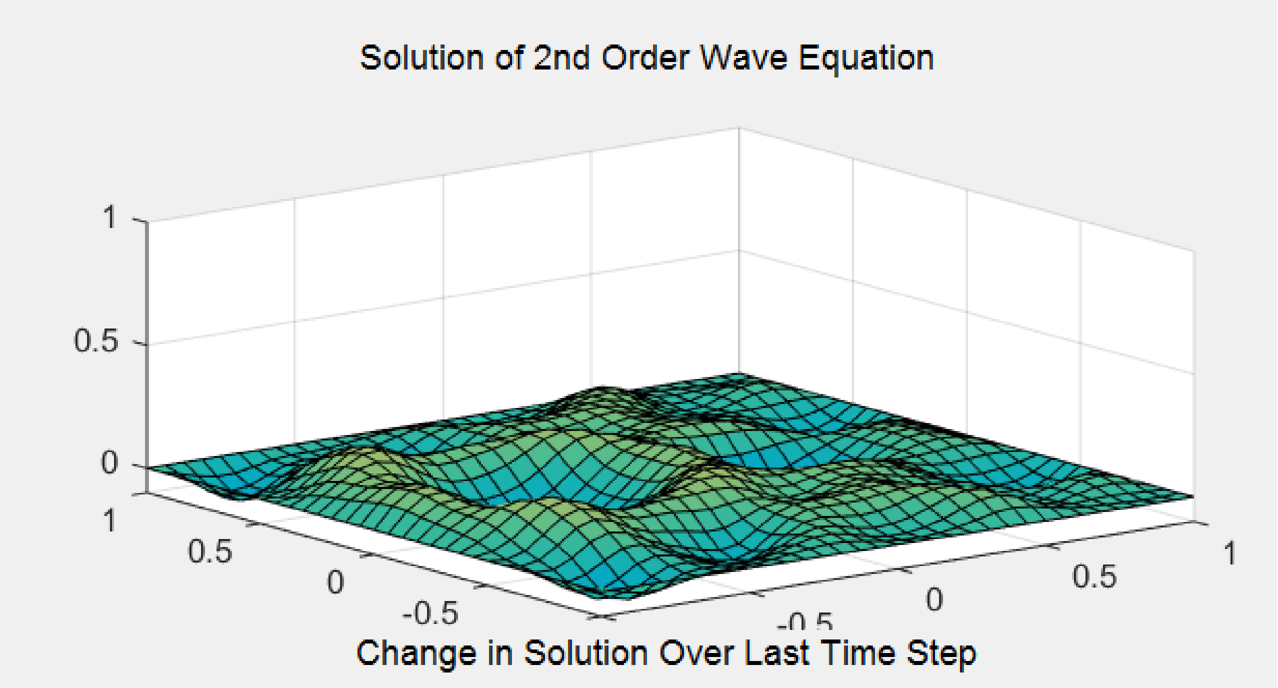
wave equation solving

NVIDIA Titan V GPU, Intel® Core™ i7-8700T Processor (12MB Cache, 2.40GHz)

Demo: Wave Equation

Accelerating scientific computing in MATLAB with GPUs

- **Objective:** Solve 2nd order wave equation with spectral methods
- **Approach:**
  - Develop code for CPU
  - Modify the code to use GPU computing using `gpuArray`
  - Compare performance of the code using CPU and GPU



Big Data

Running into “Big Data” Issues?

- “Out of memory”
  - Running out of address space
- Performance
  - Takes too long to process all of your data
- Slow processing (swapping)
  - Data too large to be efficiently managed between RAM and virtual memory

How big is big?

What does “Big Data” even mean?

“Any collection of data sets so large and complex that it becomes difficult to process using … traditional data processing applications.”

(Wikipedia)

“Any collection of data sets so large that it becomes difficult to process using traditional MATLAB functions, which assume all of the data is in memory.”

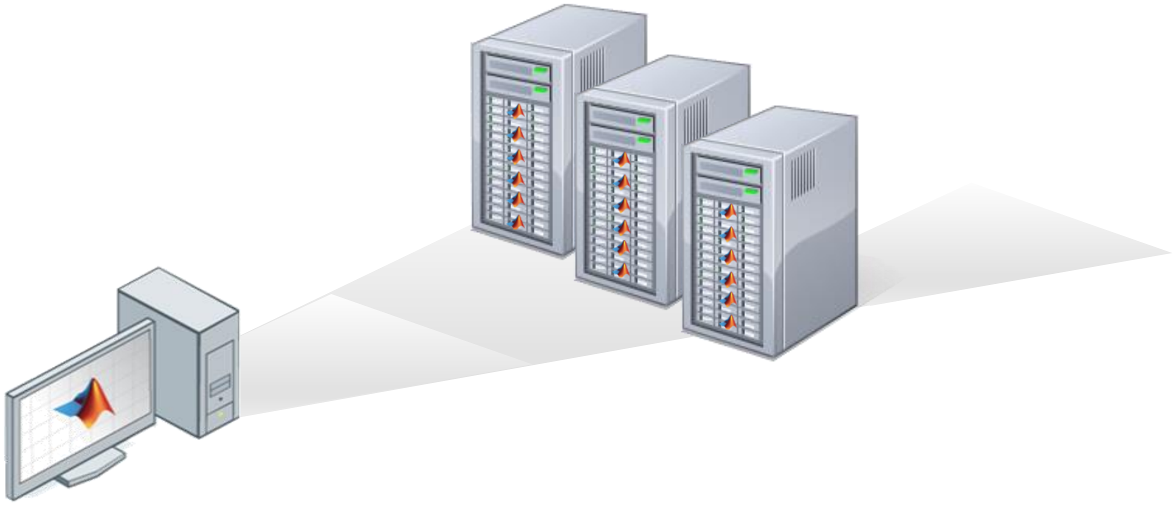
(MATLAB)

How big is big?

Sizes of data in this talk

- Most of our data lies somewhere in between the extremes
  - <   1GB can typically be handled in memory on one machine (small data)
  - >100GB typically requires processing in pieces using many machines (big data)

- 10GB might be too much for one laptop / desktop ("**inconveniently large**")



Big problems

So what's the big problem?

- Standard tools won't work
- Getting the data is hard; processing it is even harder
- Need to learn new tools and new coding styles
- Have to rewrite algorithms

We want to let you:

- Prototype algorithms quickly using small data
- Scale up to huge data-sets running on large clusters
- Use the same MATLAB code for both

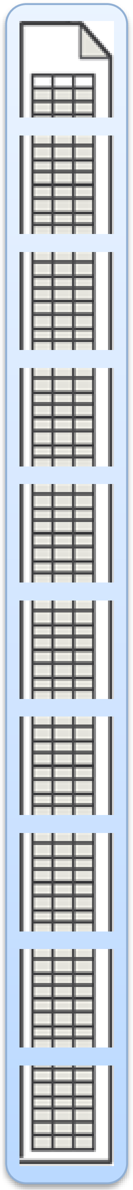
tall arrays

New solution in R2016b

Quick overview (detail later!):

- Treat data in multiple files as one large table/array
- Write normal array / table code
- Behind the scenes operate on pieces

`tall array`



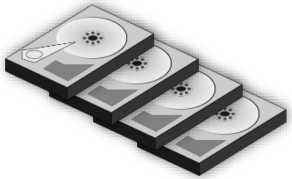
Working with data arrays in MATLAB

Take the calculation to where the data is.

Normal array – calculation happens in main memory:

```
x = rand(...)
x_norm = (x - mean(x)) ./ std(x)
```

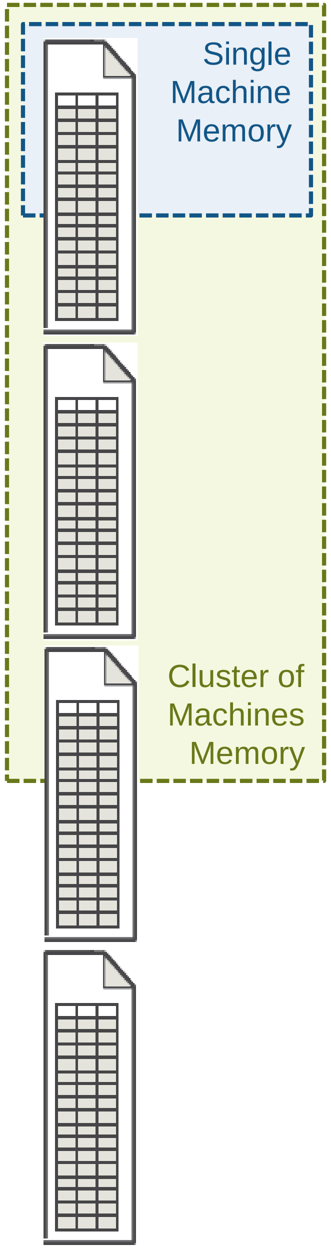
Tall array – calculation is performed by stepping through files:



```
x = tall(...)
x_norm = (x - mean(x)) ./ std(x)
```

tall arrays

New solution in R2016b



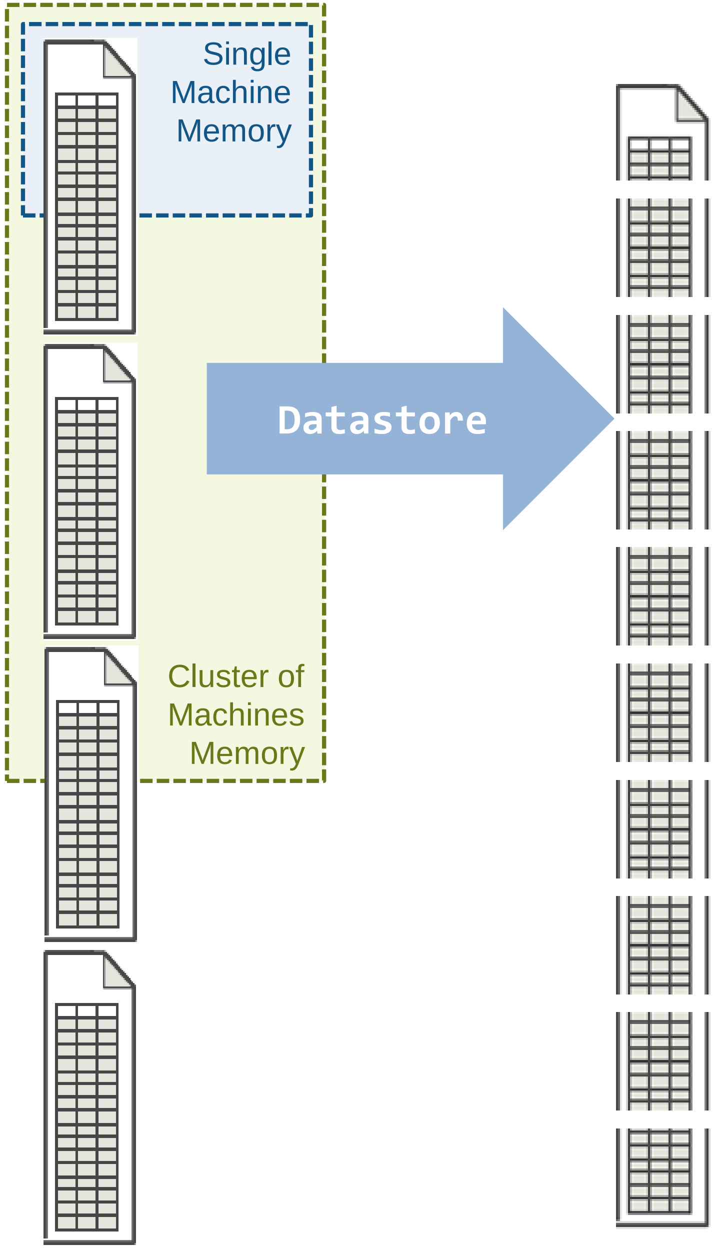
One or more files

tall arrays

New solution in R2016b

- Use datastore to define file-list

```
>> ds = datastore('*.csv')
```



One or more files

tall arrays

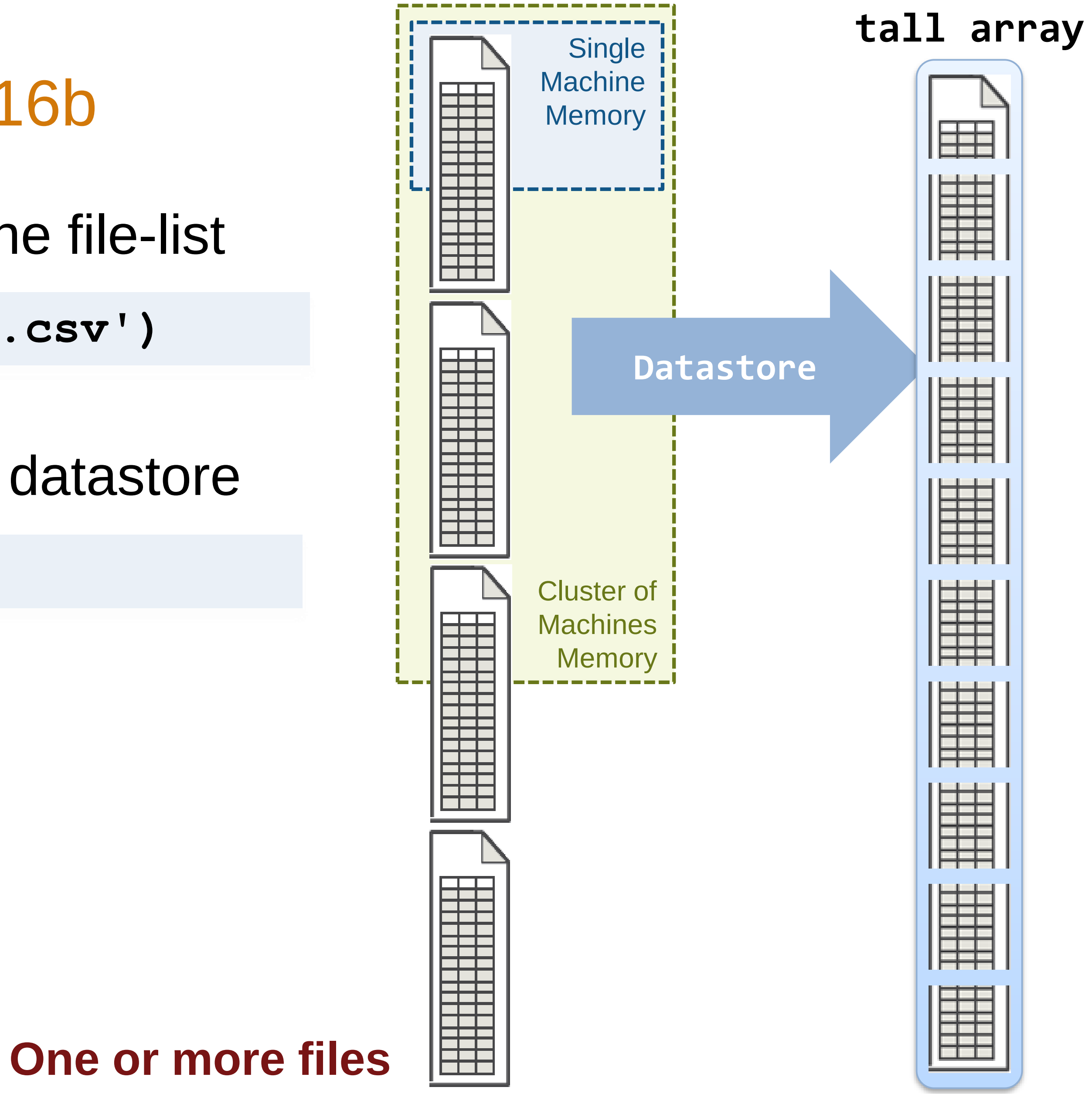
New solution in R2016b

- Use datastore to define file-list

```
>> ds = datastore('*.csv')
```

- Create tall table from datastore

```
>> tt = tall(ds)
```



tall arrays

New solution in R2016b

- Use datastore to define file-list

```
>> ds = datastore('*.csv')
```

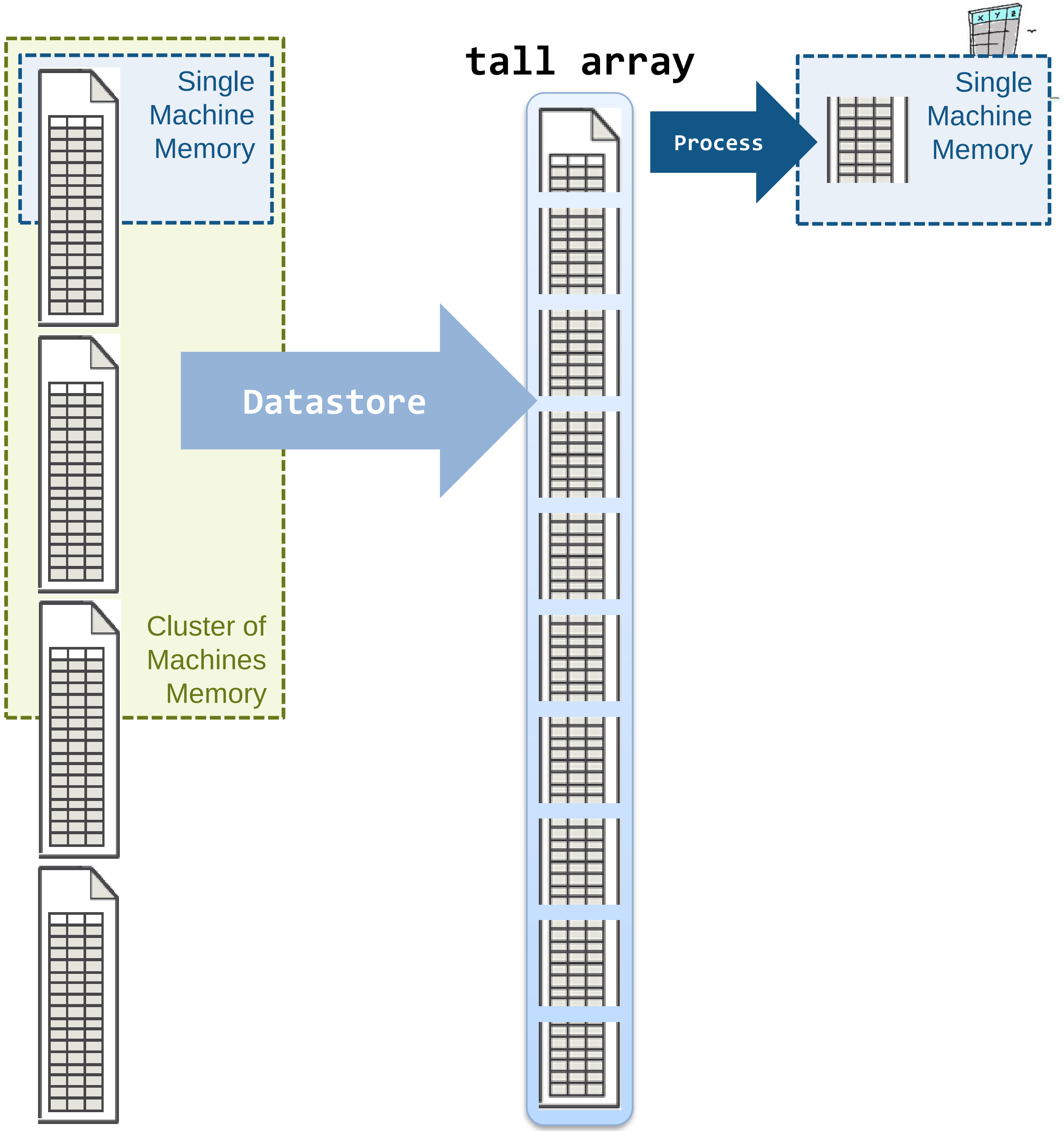
- Create tall table from datastore

```
>> tt = tall(ds)
```

- Operate on whole tall table just like ordinary table

```
>> summary(tt)
>> max(tt.EndTime - tt.StartTime)
```

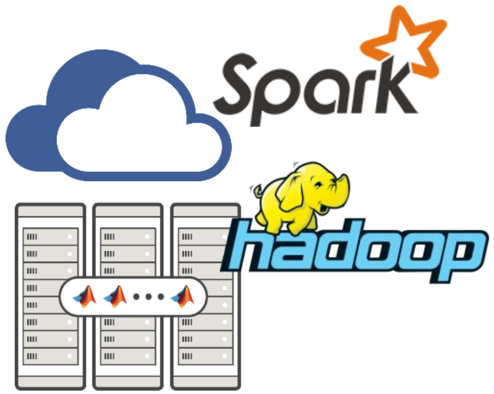
One or more files

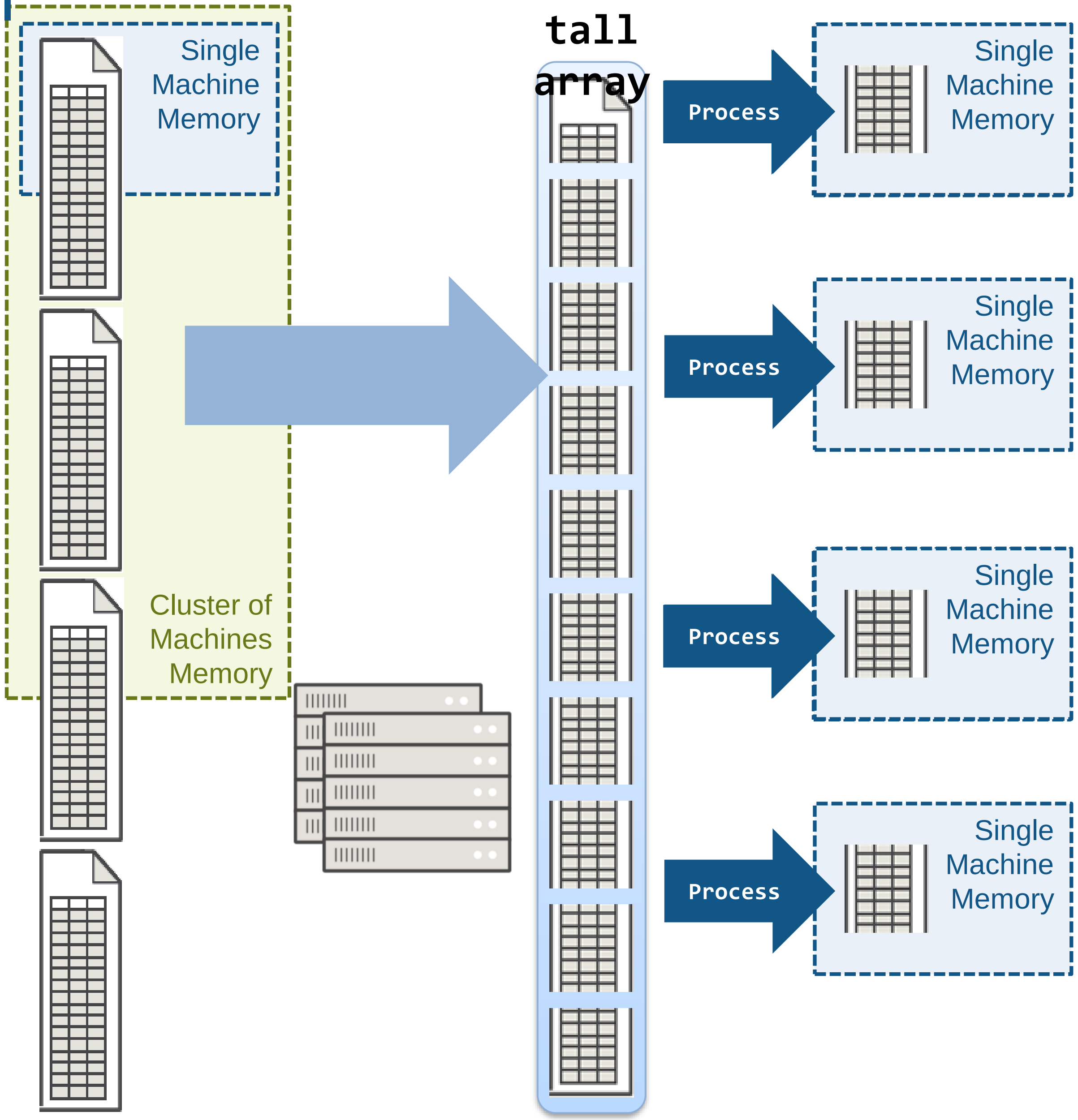


Speed up `tall` arrays with parallel

- With Parallel Computing Toolbox, process several “chunks” at once

- Can scale up to clusters with MATLAB Parallel Server



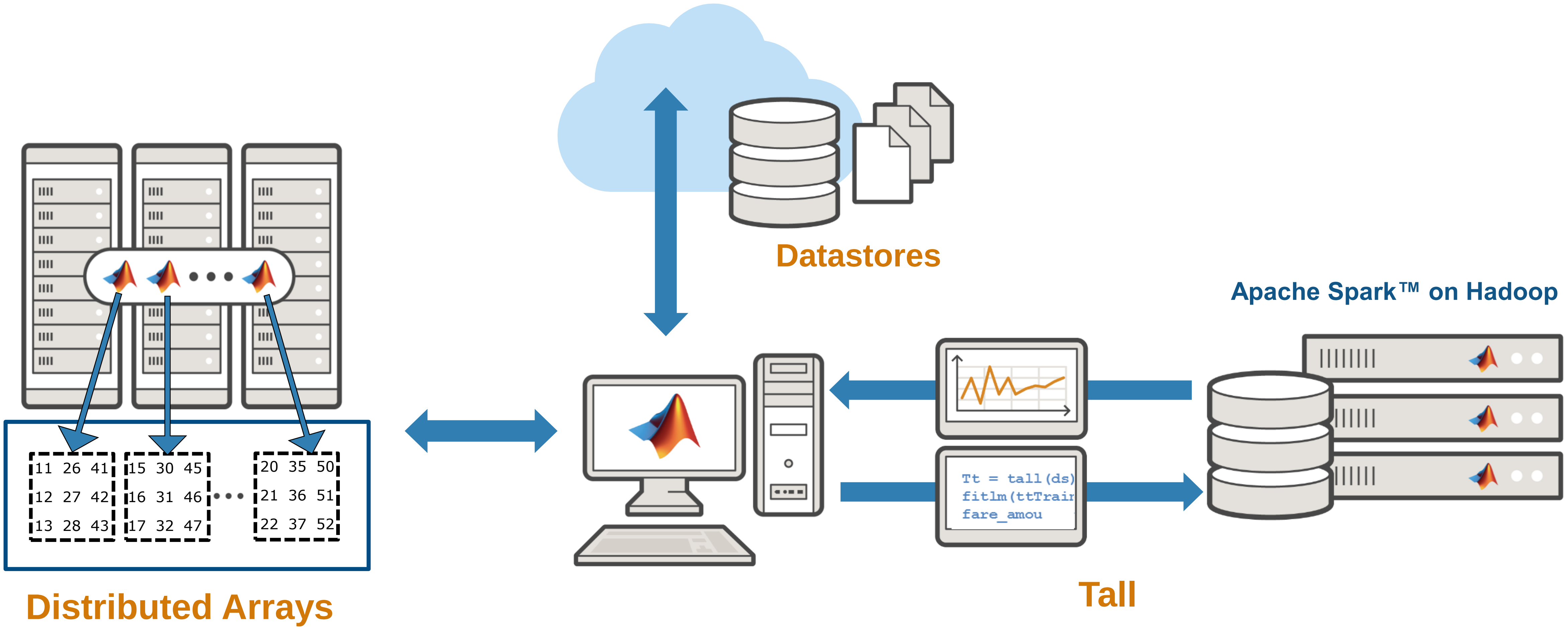


Example Use Case: Dealing with Big Data in MATLAB

- **Goal:** Create a model to predict the cost of a Taxi Ride in New York City.

- **Considerations:**
  - Raw data are .csv taxi ride log files
  - File size ranges from 22 – 26MB
  - The full data set contains > 2 million rows
  - Start with linear regression (to facilitate prediction)
  - Scale up initial work

Extend big data capabilities in MATLAB with parallel computing



Summary

- Easily develop parallel MATLAB applications without being a parallel programming expert
- Speed up the execution of your MATLAB applications using additional hardware
- Develop parallel applications on your desktop and easily scale to a cluster when needed

Why MATLAB?

Accelerating the Pace of Engineering and Science