

# Parallel Computing with MATLAB

## Hands-On Workshop for HPC.NRW

**Steve Schäfer**

MathWorks Academia Group

[steves@mathworks.com](mailto:steves@mathworks.com)

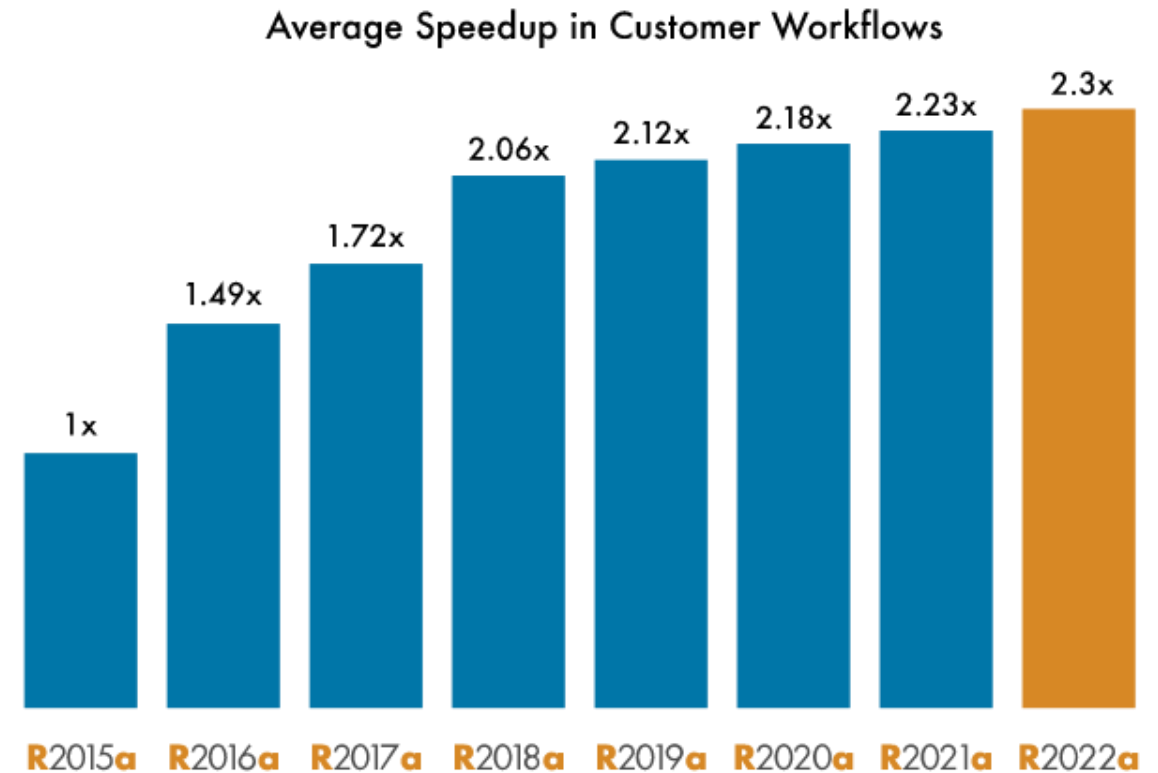
# Agenda

- Accelerating Serial MATLAB Code
- Introduction to Parallel Computing with MATLAB
- Speeding up computation with the **Parallel Computing Toolbox**
- Using GPUs with MATLAB
- Scaling up to a Cluster using **MATLAB Parallel Server**
- Overview of Big Data Capabilities in MATLAB (Optional)

# Accelerating Serial MATLAB Code

## Run MATLAB code faster by ...

- installing the latest release **R2022a**
  - Incremental improvements each release
  - Examples: Faster differential equation solvers and reading of images, render plots using less memory
  - Increased speed of MATLAB startup
- using built-in functions and data-types:
  - Regular performance improvements
  - These are extensively documented and tested with each other; constantly updated.



# Optimize your code before parallelizing for best performance



Try using functions instead of scripts. Functions are generally faster.



Instead of resizing arrays dynamically, **pre-allocate** memory.



Create a new variable rather than assigning data of a different type to an existing variable.



**Vectorize** — Use matrix and vector operations instead of for-loops.



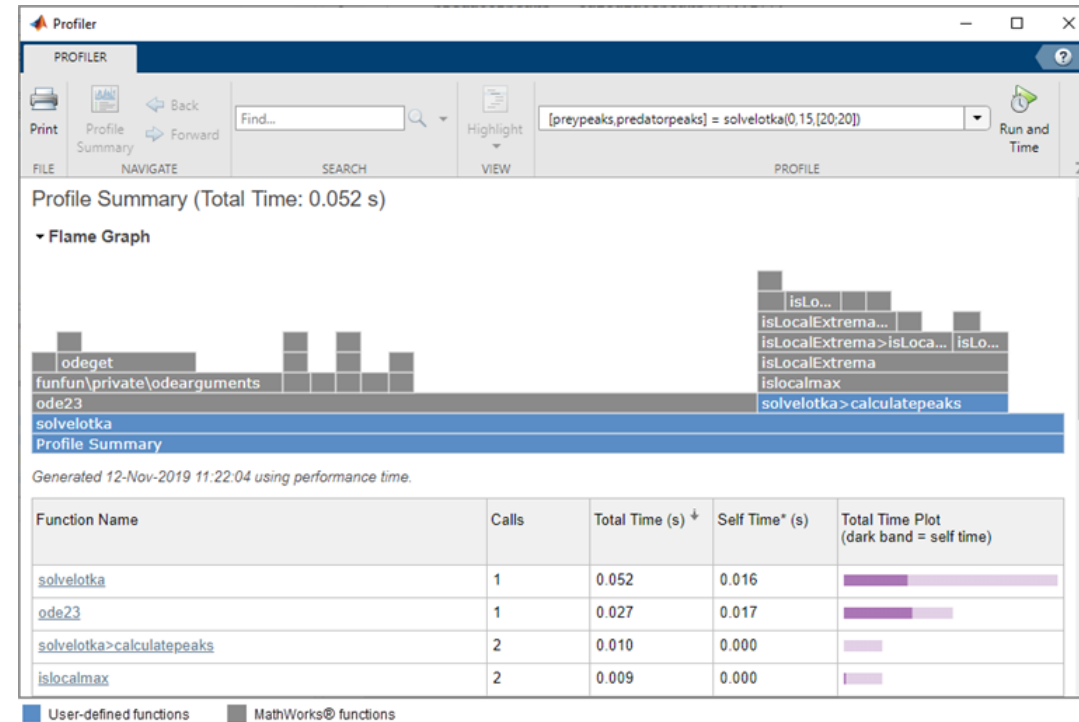
Avoid printing too much data on the screen, reuse existing graphics handles.



Avoid programmatic use of `cd`, `addpath`, and `rmpath` when possible.

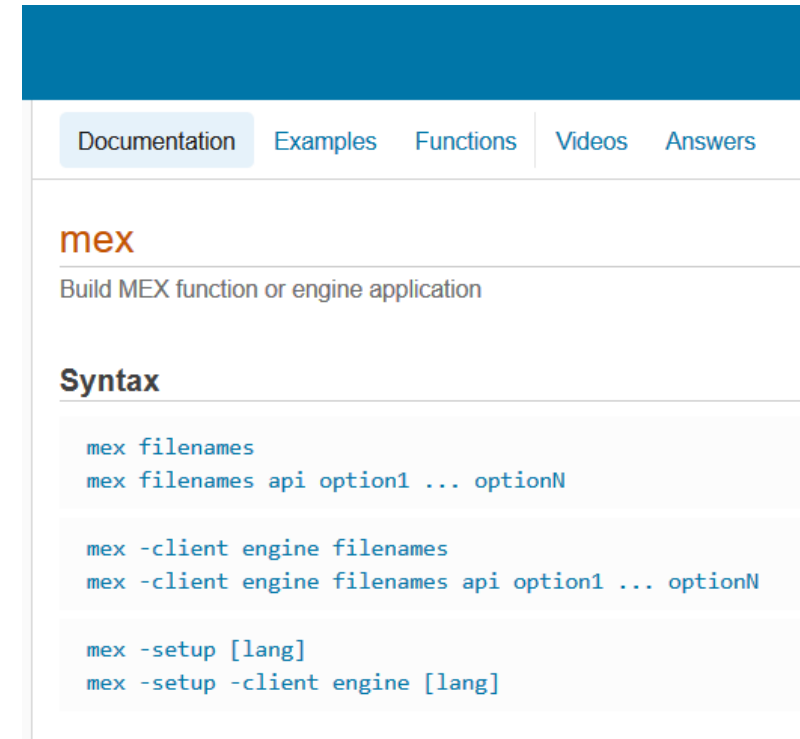
# Optimize your code before parallelizing for best performance

- Use `tic` & `toc` to
  - time your code executions
- Use [MATLAB Profiler](#) to
  - analyse the execution time
  - Identify bottlenecks.



# Optimize your code before parallelizing for best performance

- Replace code with MEX functions (Advanced)
  - Generate MATLAB Executable ([MEX](#)) C/C++ or CUDA code from a function.
  - Use MATLAB Coder & GPU Coder Apps to generate code more easily.
  - [Lots of supported functions](#)
  - Massive speed-up for certain applications



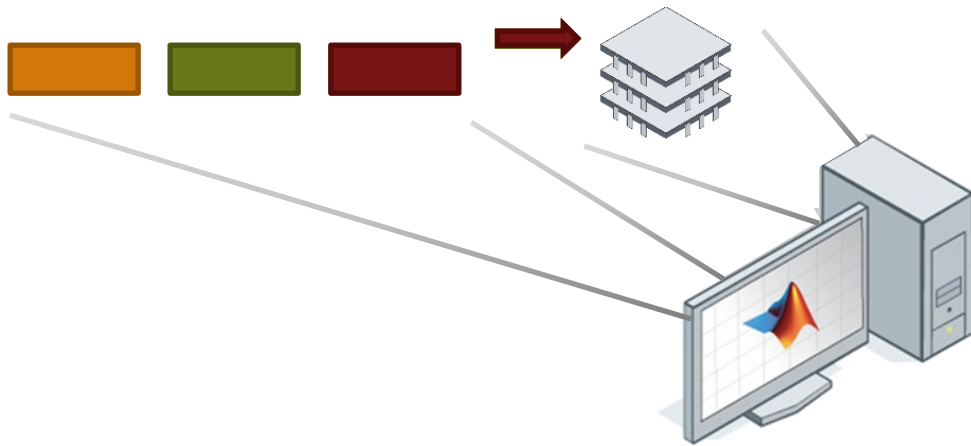
The screenshot shows the MATLAB documentation page for the `mex` command. The page has a blue header with navigation tabs for Documentation, Examples, Functions, Videos, and Answers. The `mex` command is highlighted in orange. Below the title, it says "Build MEX function or engine application". The "Syntax" section lists several command-line options: `mex filenames`, `mex filenames api option1 ... optionN`, `mex -client engine filenames`, `mex -client engine filenames api option1 ... optionN`, `mex -setup [lang]`, and `mex -setup -client engine [lang]`.

## Now for something different:

- So far we've mostly talked about using only one core of your computer
  - But your CPU probably has many cores (2-16+), which you can utilise.
  - You may also have access to a GPU, which has hundreds of cores.
  - Or a powerful workstation or HPC Cluster or an AWS EC2 instance with multiple cores.
- Now we'll look at how to utilise these.
  - We are using the **Parallel Computing Toolbox** on your local machine
  - and **MATLAB Parallel Server** for (remote) cluster or cloud computing

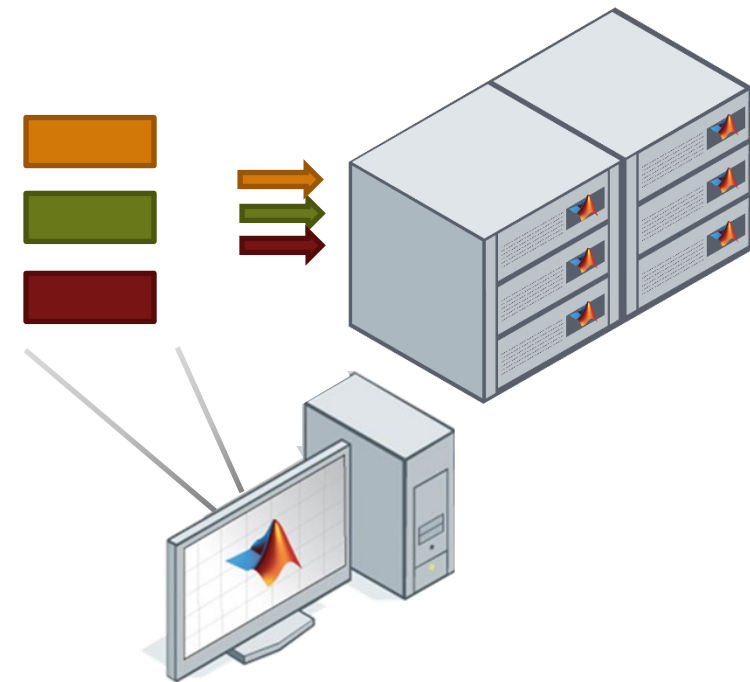
# What is Parallel Computing?

## Serial



Code executes in sequence

## Parallel

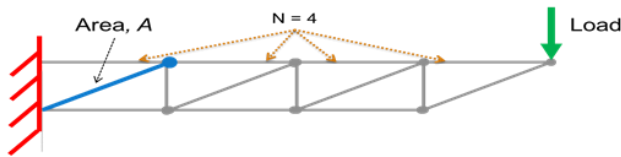


Code executes in parallel

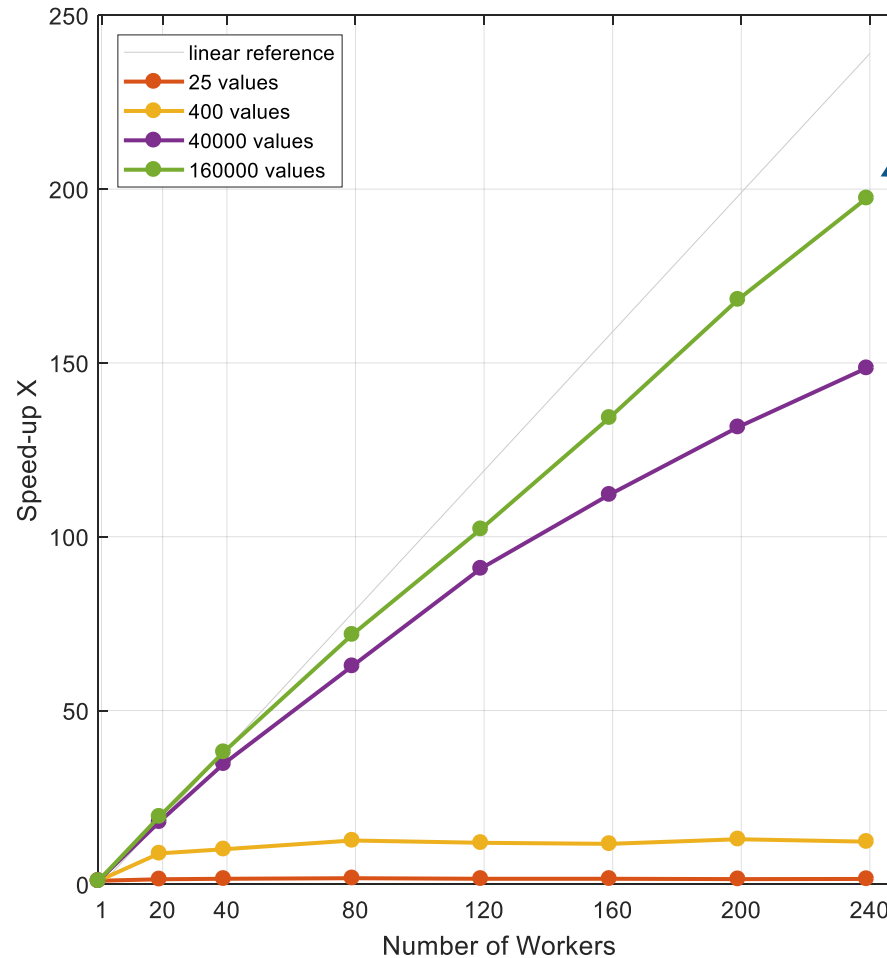
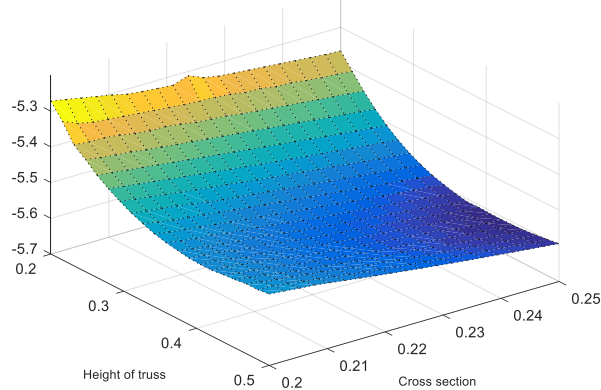
# Why is Parallel Computing useful?

Great potential for accelerating certain types of applications

$$M\ddot{x} + C\dot{x} + Kx = F$$



Log of Maximum Y Deflection (12 segments)



200x faster!

Workers in pool	Compute time (minutes)			
	160e3 values	40e3 values	400 values	25 values
1	70	17	0.19	0.02
20	3.6	0.9	0.02	0.01
40	1.8	0.5	0.02	0.01
80	1.0	0.3	0.02	0.01
160	0.5	0.2	0.02	0.01
240	0.4	0.1	0.02	0.01

Processor: Intel Xeon E5-class v2  
16 physical cores per node  
MATLAB R2017a

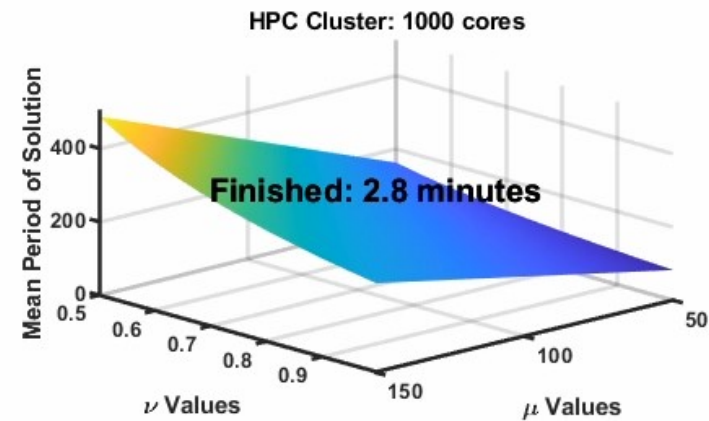
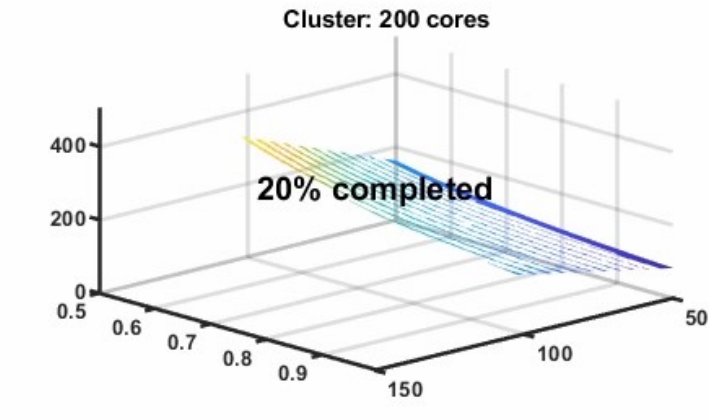
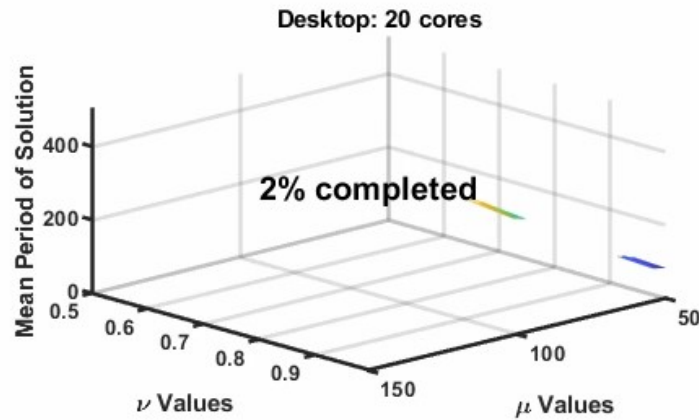
# What types of problems can Parallel Computing be used for?

- Large problems that can be easily broken down into lots of smaller ones, which are then solved at the same time
- “Embarrassingly Parallel”
  - Term originally coined by [Cleve Moler](#), who created the first version of MATLAB in 1984

## Some Examples:

- Mesh-based solutions for Partial Differential Equations (PDEs)
- Independent Simulations with different parameters
- Discrete Fourier Transforms, with each harmonic calculated independently

# Parameter Sweep for a van der Pol Oscillator (a common ODE): Speeding up the same code in three different environments



# When to use Parallel Computing?

## Some questions to consider

- Do you need to solve larger problems faster?
- Have you already optimized your serial code?
- Can your problem be solved in parallel?
  
- If so, do you have access to:
  - A multi-core or multi-processor computer?
  - A graphics processing unit (GPU)?
  - Access to an HPC Cluster or AWS?

# University of Cambridge Accelerates Neuroimaging Data Analysis to Study Healthy Cognitive Aging

## Challenge

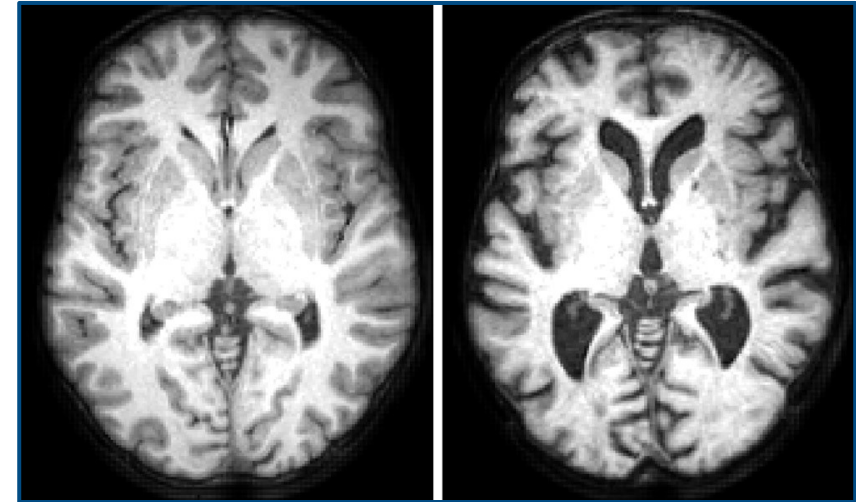
Analyze terabytes of MRI and MEG imaging data for a study on cognitive abilities in old age

## Solution

Use MATLAB to process the data on a high-performance computing cluster and to apply advanced statistical, optimization, and machine learning techniques

## Results

- Multistep image processing pipeline automated
- Data analysis accelerated on a 1200-core cluster
- Key influences on late-life cognitive health identified



Structural MRI images of the brain of a typical healthy 19-year old (left) and typical healthy 86-year old (right).

*“MATLAB Parallel Server not only reduces processing time; it also lowers the barrier to entry into parallel computing—an important consideration given the wide range of technical abilities among our scientists.”*

*- Richard Henson, University of Cambridge*

# NASA Langley Accelerates Acoustic Data Analysis with GPU Computing

## Challenge

Accelerate the analysis of sound recordings from wind tunnel tests of aircraft components

## Solution

- Use Parallel Computing Toolbox to process acoustic data
- Cut processing time by running computationally intensive operations on a GPU

## Results

- GPU computations completed 40 times faster
- Algorithm GPU-enabled in 30 minutes
- Processing of test data accelerated



Wind tunnel test setup featuring the Hybrid Wing Body model (inverted), with 97-microphone phased array (top) and microphone tower (left).

*“Our **legacy code took up to 40 minutes** to analyze a single wind tunnel test. The addition of GPU computing with Parallel Computing Toolbox **cut it to under a minute**. It took 30 minutes to get our MATLAB algorithm working on the GPU—no low-level CUDA programming was needed.”*  
- Christopher Bahr, research aerospace engineer at NASA

# Virgin Orbit Simulates LauncherOne Stage Separation Events

## Challenge

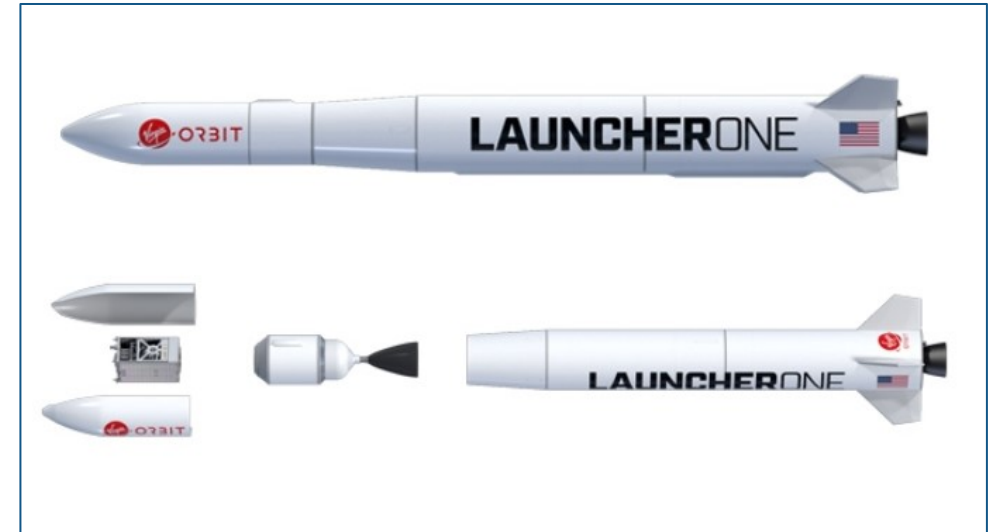
Simulate separation events for LauncherOne spacecraft

## Solution

- Use MATLAB, Simulink, and Simscape Multibody to model components and automate Monte Carlo simulations
- Used Parallel Computing Toolbox to run simulations in parallel on multicore processors

## Results

- Simulations completed 10 times faster
- Simulation set up times cut by up to 90%
- Hardware designs informed by simulation results



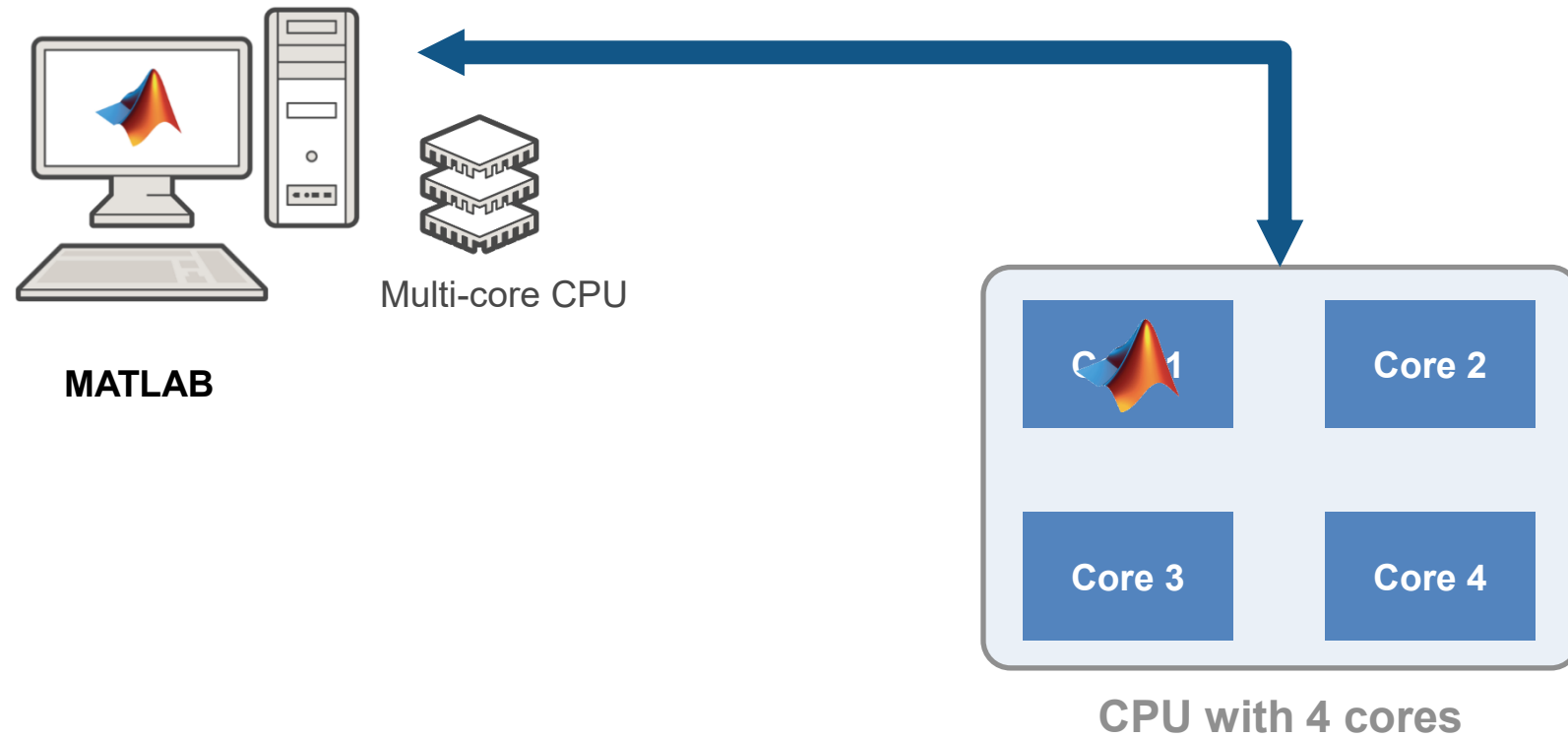
Virgin Orbit's LauncherOne vehicle assembled (top), with exploded view showing the fairing, payload, and first and second stages (bottom).

*"With Simulink, we can employ simplifying assumptions and parallel processing to reduce simulation times from days to hours...Just as important, we can automate the simulations so they run in the background or overnight, and have the results waiting for us in the morning."*

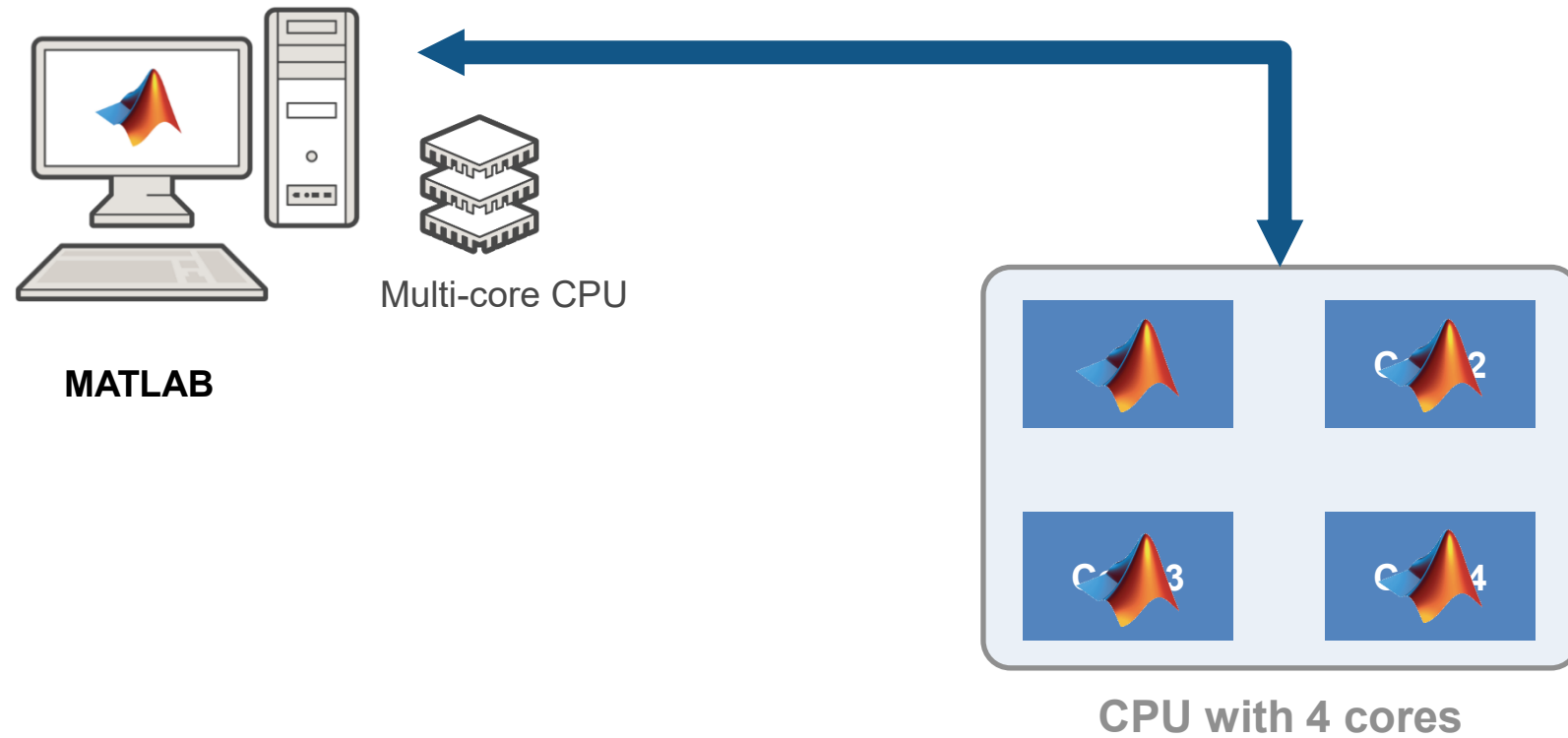
*- Patrick Harvey, Associate Engineer at Virgin Orbit*

# Introduction to Parallel Computing with MATLAB

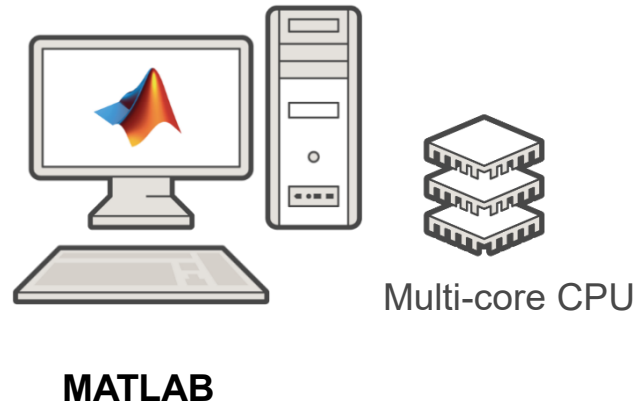
# Most of your MATLAB code runs on one core



# Run multiple iterations by utilizing multiple CPU cores



# MATLAB has built-in multithreading



## MATLAB Multicore



### Run MATLAB on multicore and multiprocessor machines

MATLAB® provides two main ways to take advantage of multicore and multiprocessor computers. By using the full computational power of your machine, you can run your MATLAB applications faster and more efficiently.

#### Built-in Multithreading

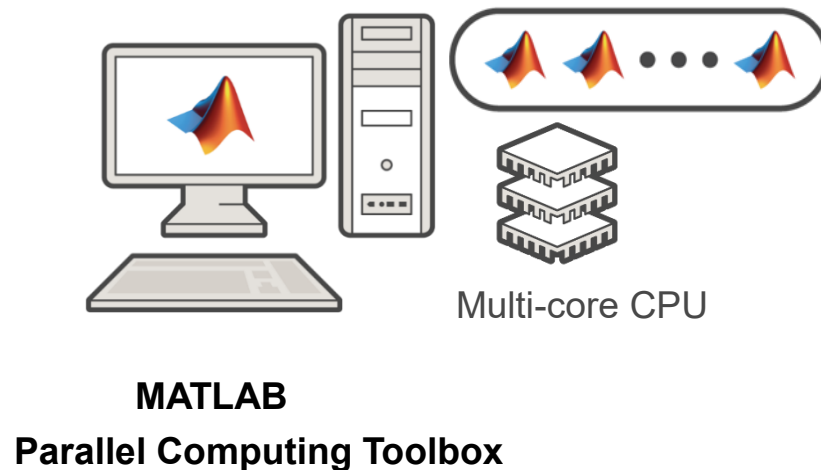
Linear algebra and numerical functions such as `fft`, `\(mldivide)`, `eig`, `svd`, and `sort` are multithreaded in MATLAB. Multithreaded computations have been on by default in MATLAB since Release 2008a. These functions automatically execute on multiple computational threads in a single MATLAB session, allowing them to execute faster on multicore-enabled machines. Additionally, many functions in Image Processing Toolbox™ are multithreaded.

#### Parallelism Using MATLAB Workers

You can run multiple MATLAB workers (MATLAB computational engines) on a single machine to execute applications in parallel, with [Parallel Computing Toolbox™](#). This approach allows you more control over the parallelism than with built-in multithreading, and is often used for coarser grained problems such as running parameter sweeps in parallel.

[MATLAB multicore](#)

# MATLAB workers execute applications in parallel



## MATLAB Multicore



### Run MATLAB on multicore and multiprocessor machines

MATLAB® provides two main ways to take advantage of multicore and multiprocessor computers. By using the full computational power of your machine, you can run your MATLAB applications faster and more efficiently.

#### Built-in Multithreading

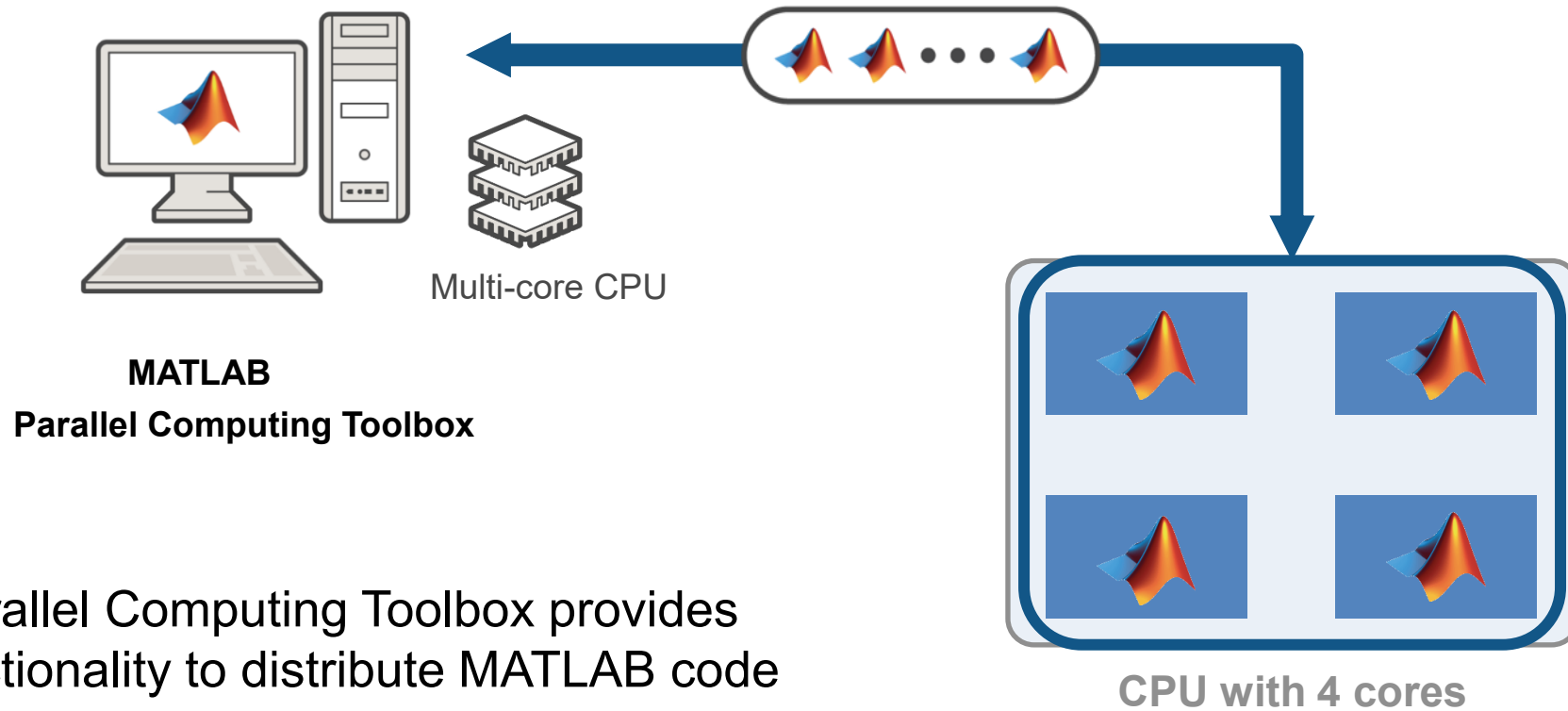
Linear algebra and numerical functions such as `fft`, `\(mldivide)`, `eig`, `svd`, and `sort` are multithreaded in MATLAB. Multithreaded computations have been on by default in MATLAB since Release 2008a. These functions automatically execute on multiple computational threads in a single MATLAB session, allowing them to execute faster on multicore-enabled machines. Additionally, many functions in Image Processing Toolbox™ are multithreaded.

#### Parallelism Using MATLAB Workers

You can run multiple MATLAB workers (MATLAB computational engines) on a single machine to execute applications in parallel, with [Parallel Computing Toolbox™](#). This approach allows you more control over the parallelism than with built-in multithreading, and is often used for coarser grained problems such as running parameter sweeps in parallel.

# MATLAB + Parallel Computing Toolbox

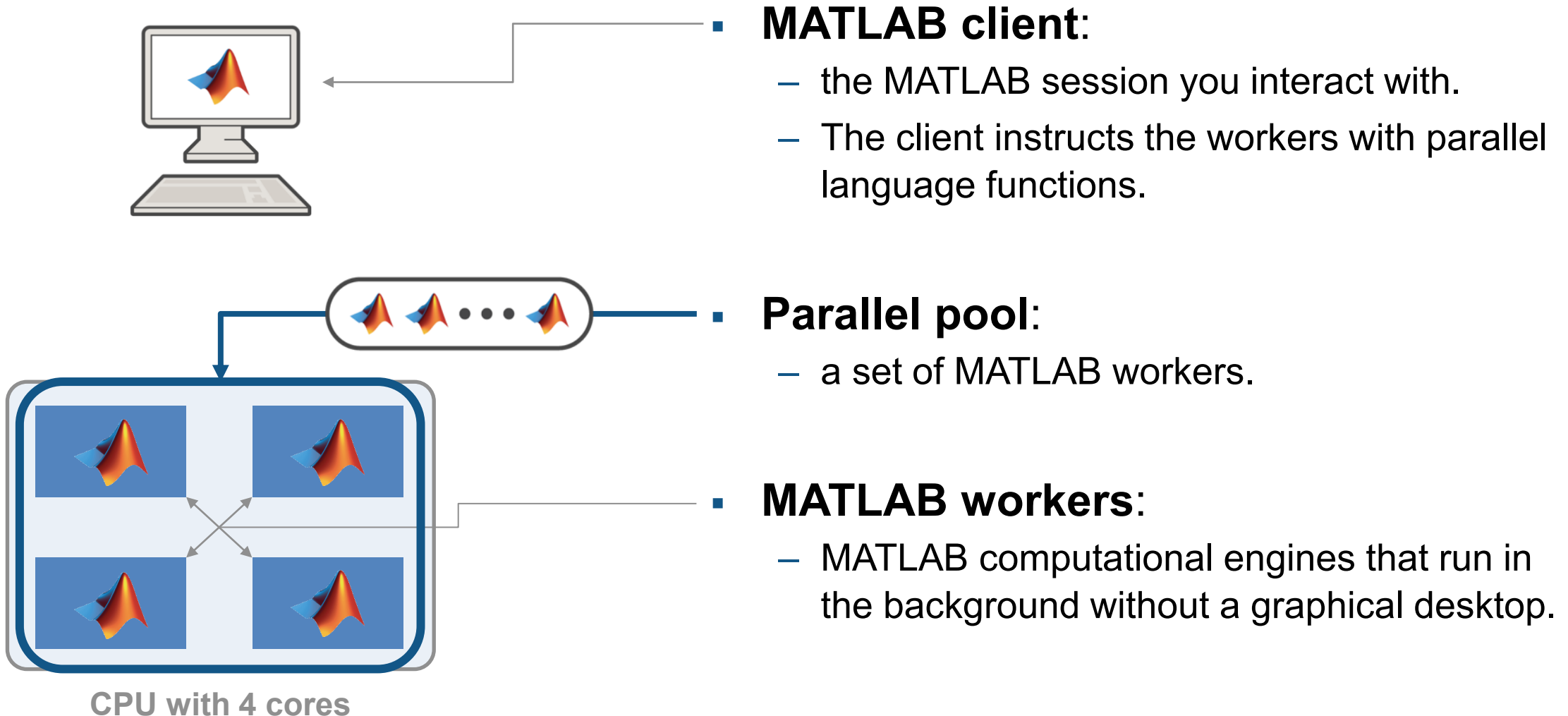
Leverage multiple cores on your machine with explicit parallel techniques



- The Parallel Computing Toolbox provides the functionality to distribute MATLAB code across multiple MATLAB worker.

# MATLAB + Parallel Computing Toolbox

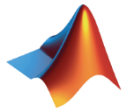
## Terminology



# MATLAB + Parallel Computing Toolbox

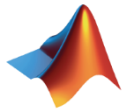
Only simple modifications to your code required

Three good commands to know:



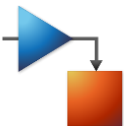
**for** → **parfor**

(parallel for-loop)



**feval** → **parfeval**

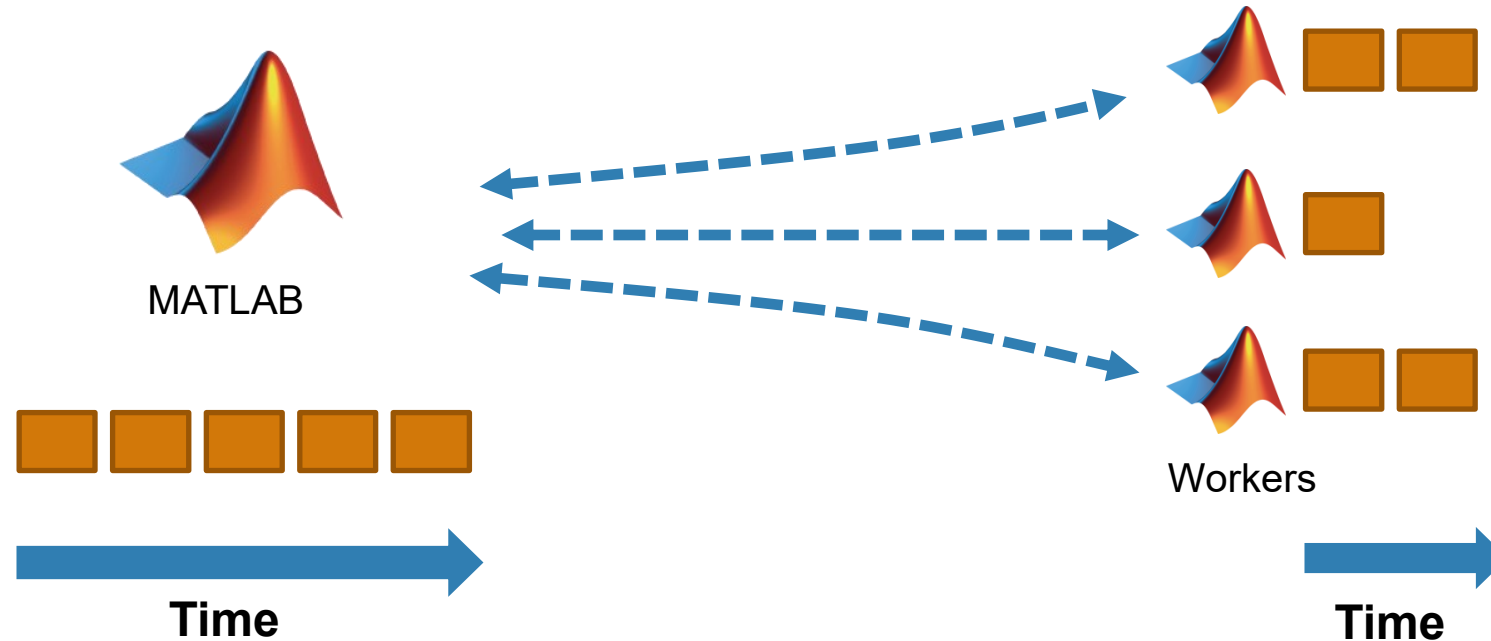
(parallel function evaluations)



**sim** → **parsim**

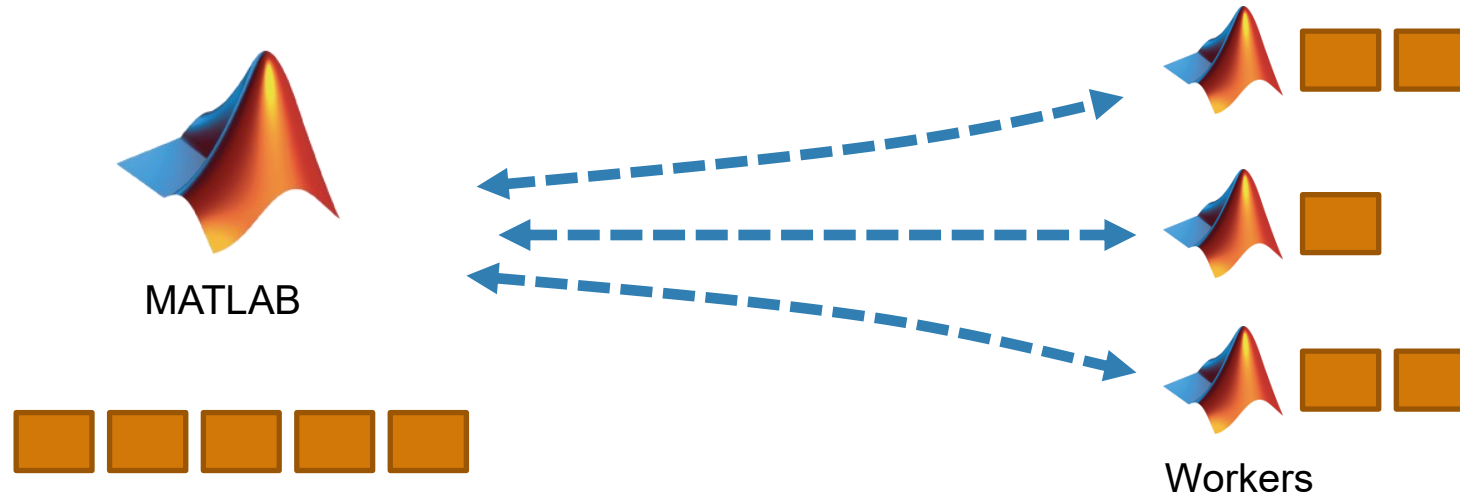
(parallel Simulink runs)

# Explicit parallelism with `parfor`



- Run iterations in parallel
- Examples: parameter sweeps, Monte Carlo simulations

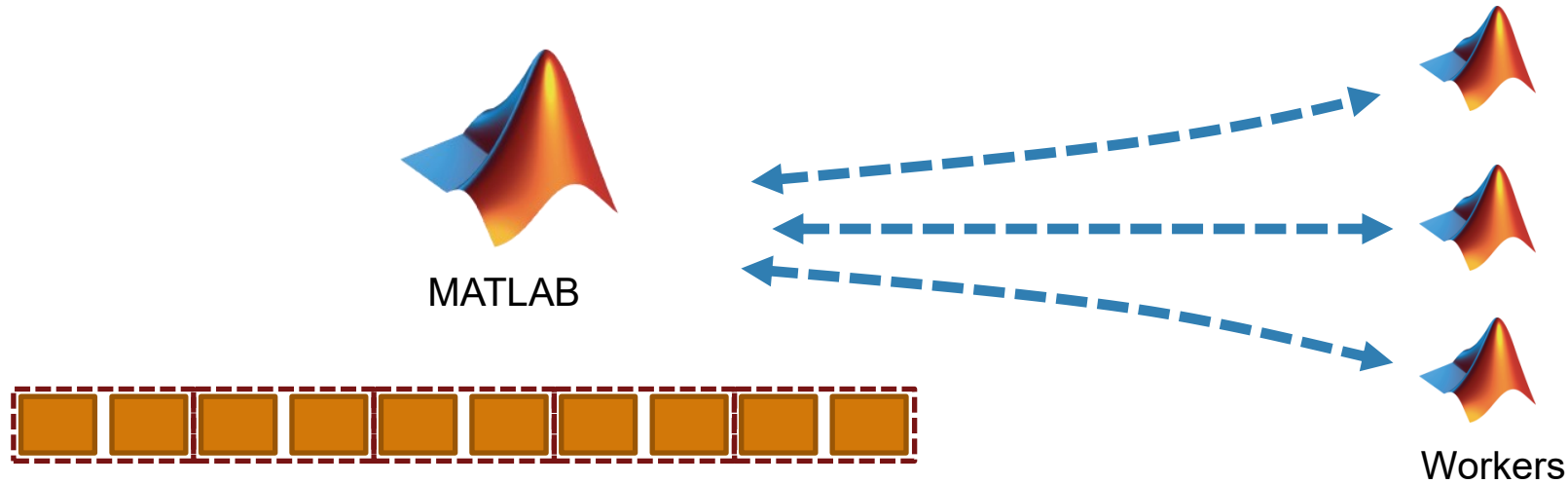
# Explicit parallelism with `parfor`



```
a = zeros(5, 1);  
b = pi;  
for i = 1:5  
    a(i) = i + b;  
end  
disp(a)
```

```
a = zeros(5, 1);  
b = pi;  
parfor i = 1:5  
    a(i) = i + b;  
end  
disp(a)
```

# Explicit parallelism with `parfor`



```
a = zeros(10, 1);
```

```
b = pi;
```

```
parfor i = 1:10
```

```
    a(i) = i + b;
```

```
end
```

```
disp(a)
```

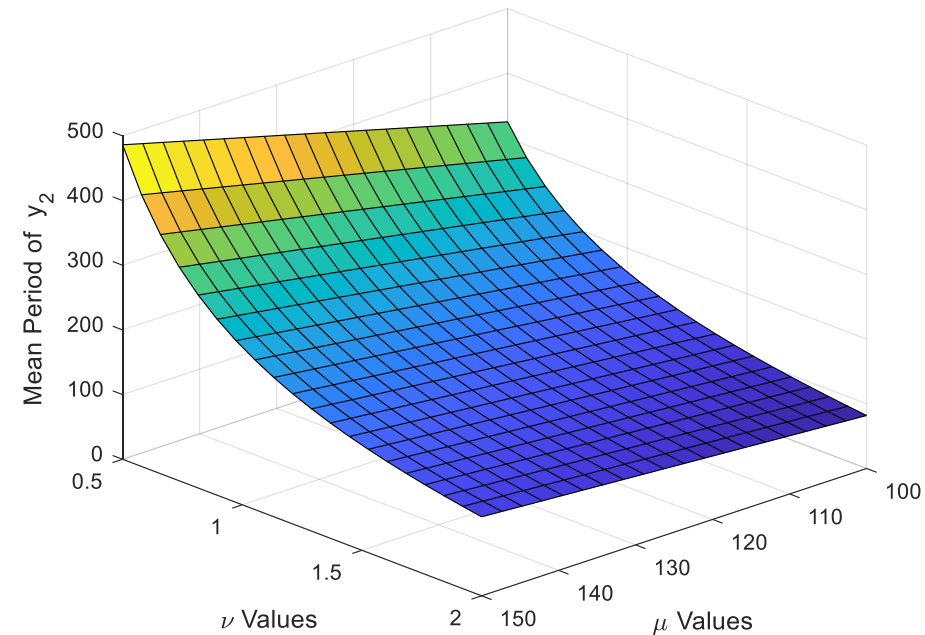
# Speed up a parameter sweep using `parfor`

## Demo: Parameter sweep for van der Pol oscillator

- System of ODEs

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

- Compute mean period of  $y$
- Use `parfor`, study impact of  $\nu, \mu$

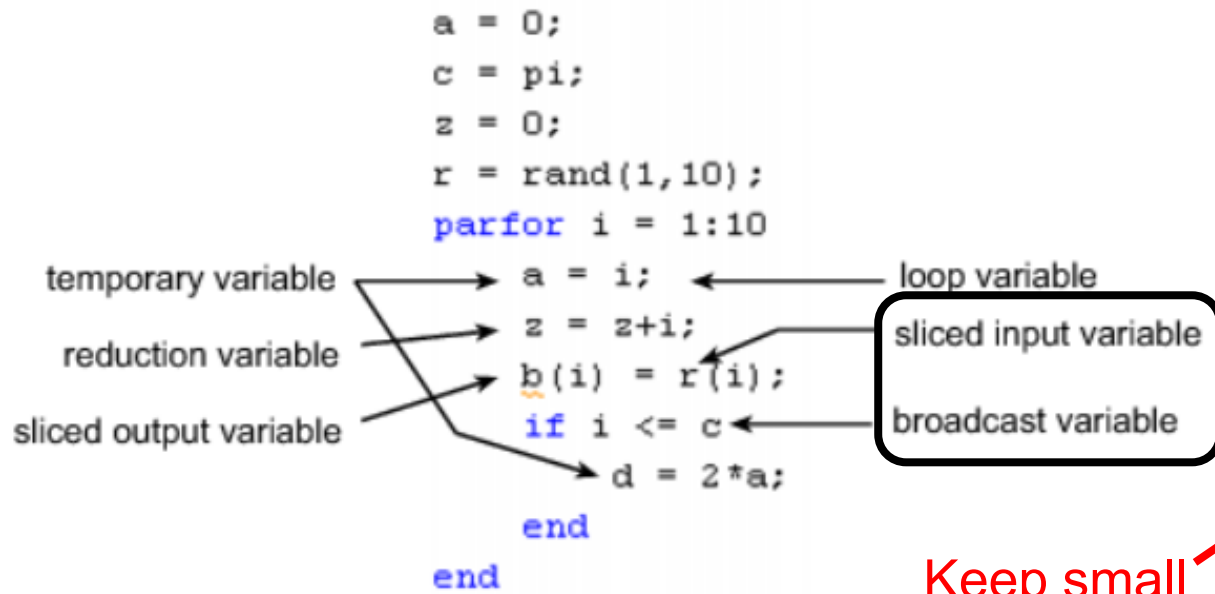


# Hands-On Exercise: Introduction to `parfor`

## Factors that govern speedup of `parfor` loops

- May not be much speedup when computation time is too short
- Execution may be slow because of:
  - Memory limitations (RAM)
  - File access limitations
- Implicit multithreading
  - MATLAB uses multiple threads for speedup of some operations
  - Use Resource Monitor or similar on serial code to check on that
- Unbalanced load due to iteration execution times
  - Avoid some iterations taking multiples of the execution time of other iterations

# Optimizing parfor



Type	Category
sliced input	input
broadcast	input
reduction	output
sliced output	output
loop	only exist on worker
temporary	only exist on worker

Use more

Keep small

## Execute additional code as iterations complete

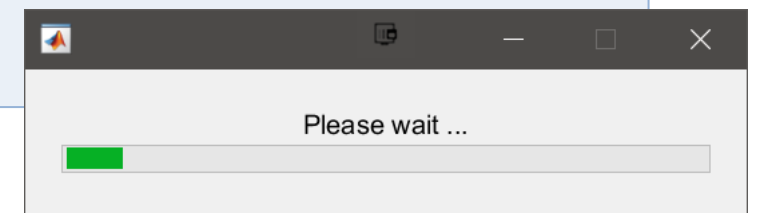
- Send data or messages from parallel workers back to the MATLAB client
- Retrieve intermediate values and track computation progress

```
function a = parforWaitbar
    D = parallel.pool.DataQueue;
    h = waitbar(0, 'Please wait ...');
    afterEach(D, @nUpdateWaitbar)

    N = 200;
    p = 1;

    parfor i = 1:N
        a(i) = max(abs(eig(rand(400)))));
        send(D, i)
    end

    function nUpdateWaitbar(~)
        waitbar(p/N, h)
        p = p + 1;
    end
end
```



# Execute functions in parallel asynchronously using `parfeval`



- Asynchronous execution on parallel workers
- Useful for “needle in a haystack” problems

```
for idx = 1:10
    f(idx) = parfeval(@magic,1,idx);
end

for idx = 1:10
    [completedIdx,value] = fetchNext(f);
    magicResults{completedIdx} = value;
end
```

# Hands-On Exercise: Introduction to `parfeval`

# Automatic parallel support (MATLAB)

Enable parallel computing support by setting a flag or preference

(..., 'UseParallel', true)

## Image Processing

Batch Image Processor, Block Processing, GPU-enabled functions

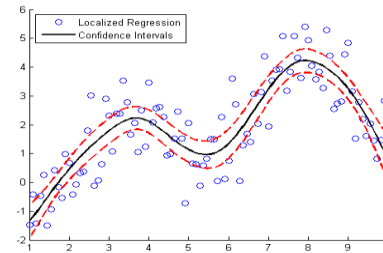


Original Image of Peppers

Recolored Image of Peppers

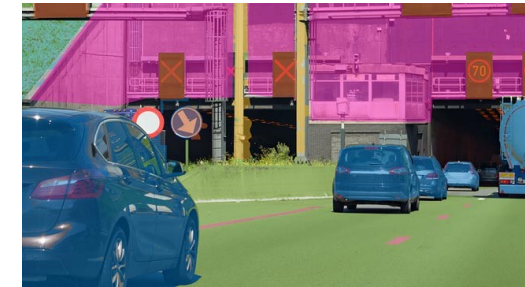
## Statistics and Machine Learning

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



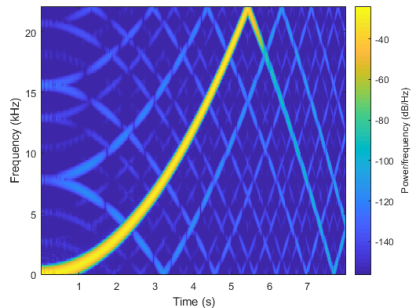
## Deep Learning

Deep Learning, Neural Network training and simulation



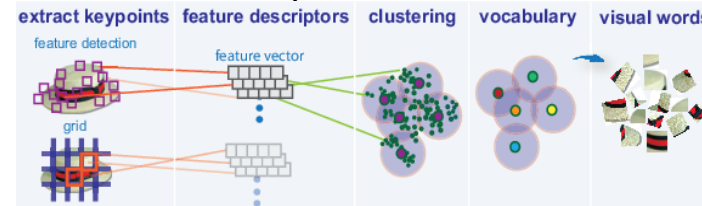
## Signal Processing and Communications

GPU-enabled FFT filtering, cross correlation, BER simulations



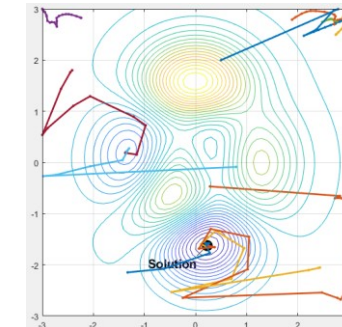
## Computer Vision

Bag-of-words workflow, object detectors



## Optimization and Global Optimization

Estimation of gradients, parallel search

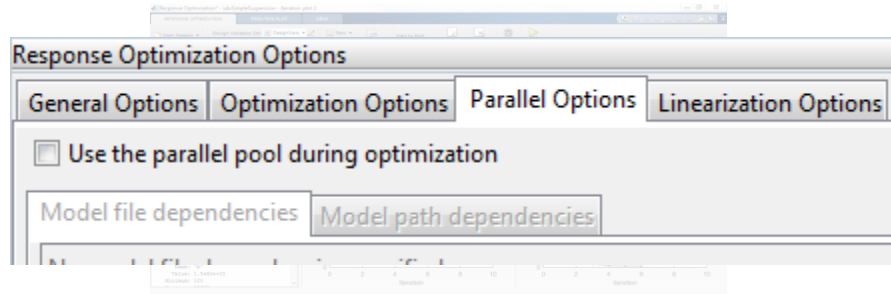


# Automatic parallel support (*Simulink*)

Enable parallel computing support by setting a flag or preference

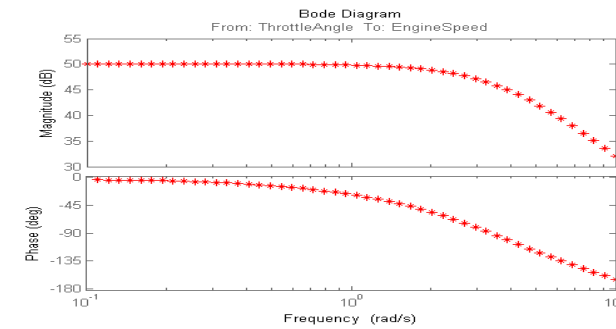
## Simulink Design Optimization

Response optimization, sensitivity analysis, parameter estimation



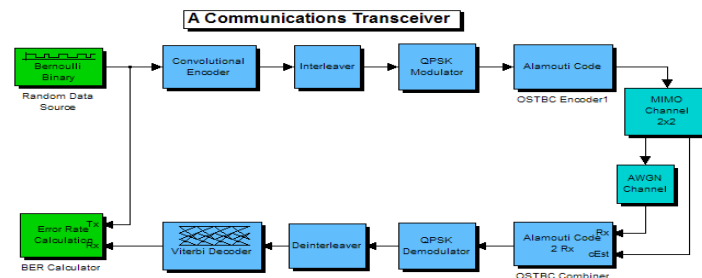
## Simulink Control Design

Frequency response estimation



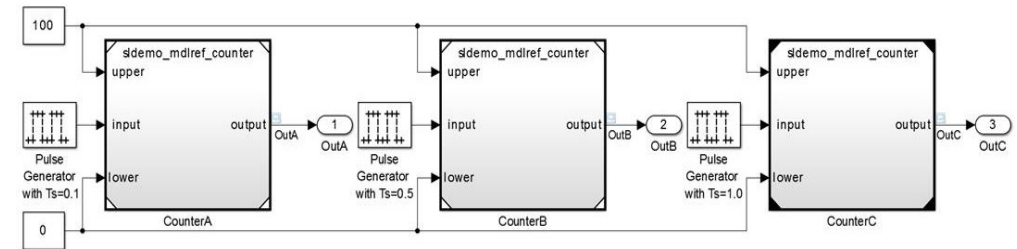
## Communication Systems Toolbox

GPU-based System objects for Simulation Acceleration



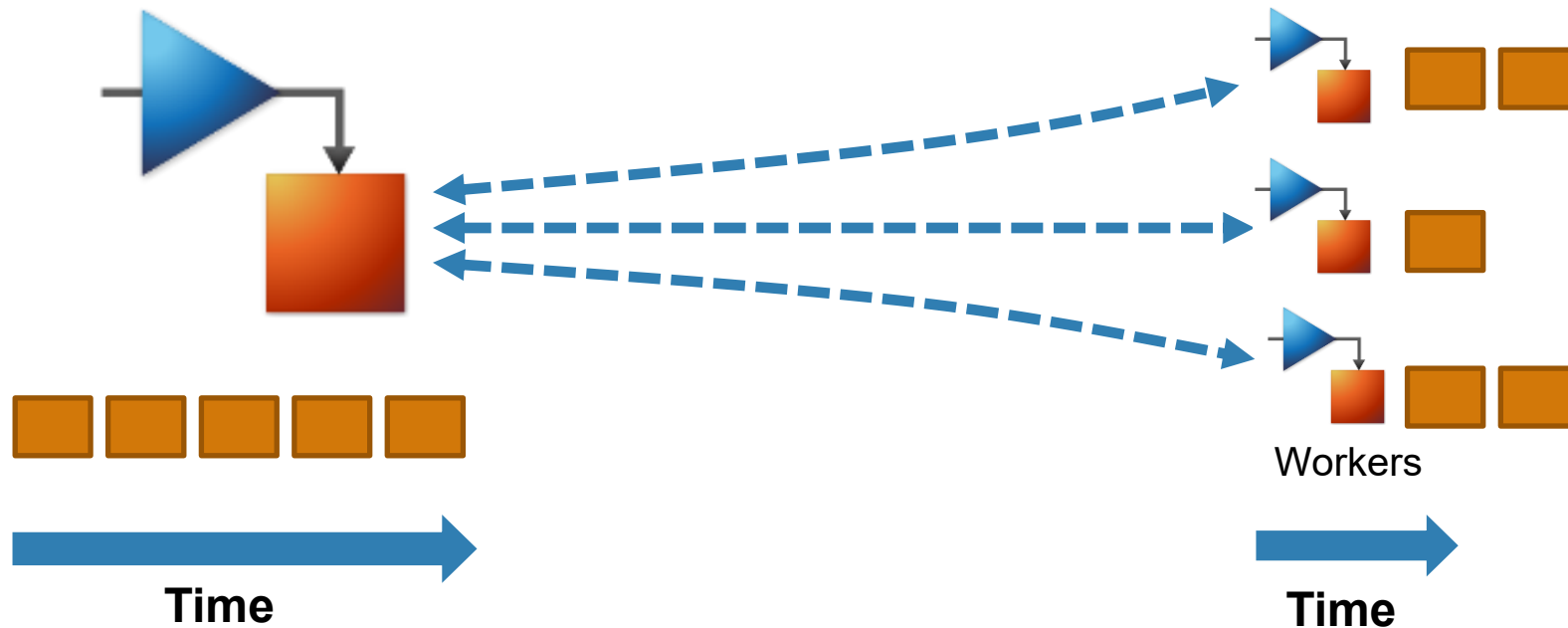
## Simulink/Embedded Coder

Generating and building code



[Additional automatic parallel support](#)

# Run multiple Simulink simulations in parallel with `parsim`

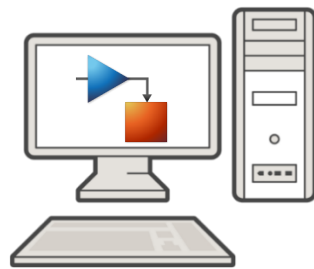


- Run independent Simulink simulations in parallel using the `parsim` function

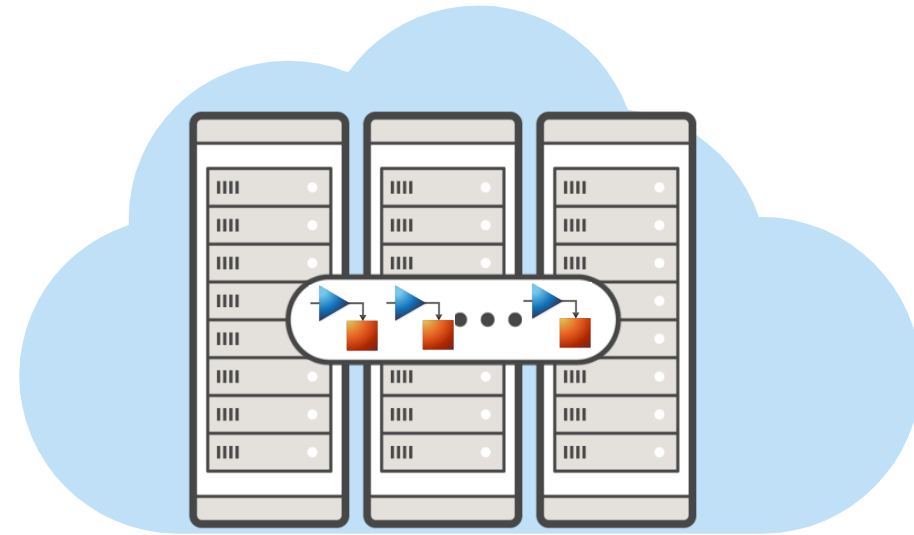
```
for i = 10000:-1:1
    in(i) = Simulink.SimulationInput(my_model);
    in(i) = in(i).setVariable(my_var, i);
end
out = parsim(in);
```

# Benefits of using `parsim`

- Run multiple simulations on your machine or clouds and clusters
- Transfer base workspace variables to workers
- Automatically transfer all files to workers
- Automatically return file logging data
- Automatically manage build folders
- Display progress
- Manage errors



Desktop



Multicore

Cluster

# Profile Simulink performance

The screenshot displays the Simulink Profiler interface for a model named 'sldemo\_antiwindup'. The model is titled 'Anti-Windup PID Control Demonstration' and consists of the following components:

- Input:** A step function  $r(t)$ .
- Summing Junction:** The input  $r(t)$  is summed with feedback from the output  $y(t)$  to produce the error signal  $e(t)$ .
- Controller:** A PID controller block labeled 'PID(s)' that takes  $e(t)$  as input and produces the control signal  $u(t)$ .
- Plant Actuator:** A block labeled 'SAT(u)' that saturates the control signal  $u(t)$ .
- Process:** A 'First Order Process' block with the transfer function  $\frac{1}{10s + 1}$ .
- Output:** A 'Dead Time' block followed by a gain of 1, resulting in the final output  $yout$ .

Below the model, the Profiler Report is shown for the run 'sldemo\_antiwindup @ 04-Nov-2019 09:33:52'. It contains two tables of performance data:

Path	Plot (Dark Band = Self Time)	Total Time (s)	Self Time (s)	Number of Calls
sldemo_antiwindup		5.631	1.237	18013
Scope		3.684	3.684	6004
Scope1		0.694	0.694	6004
PID Controller		0.005	0.000	0
Step		0.004	0.004	6002

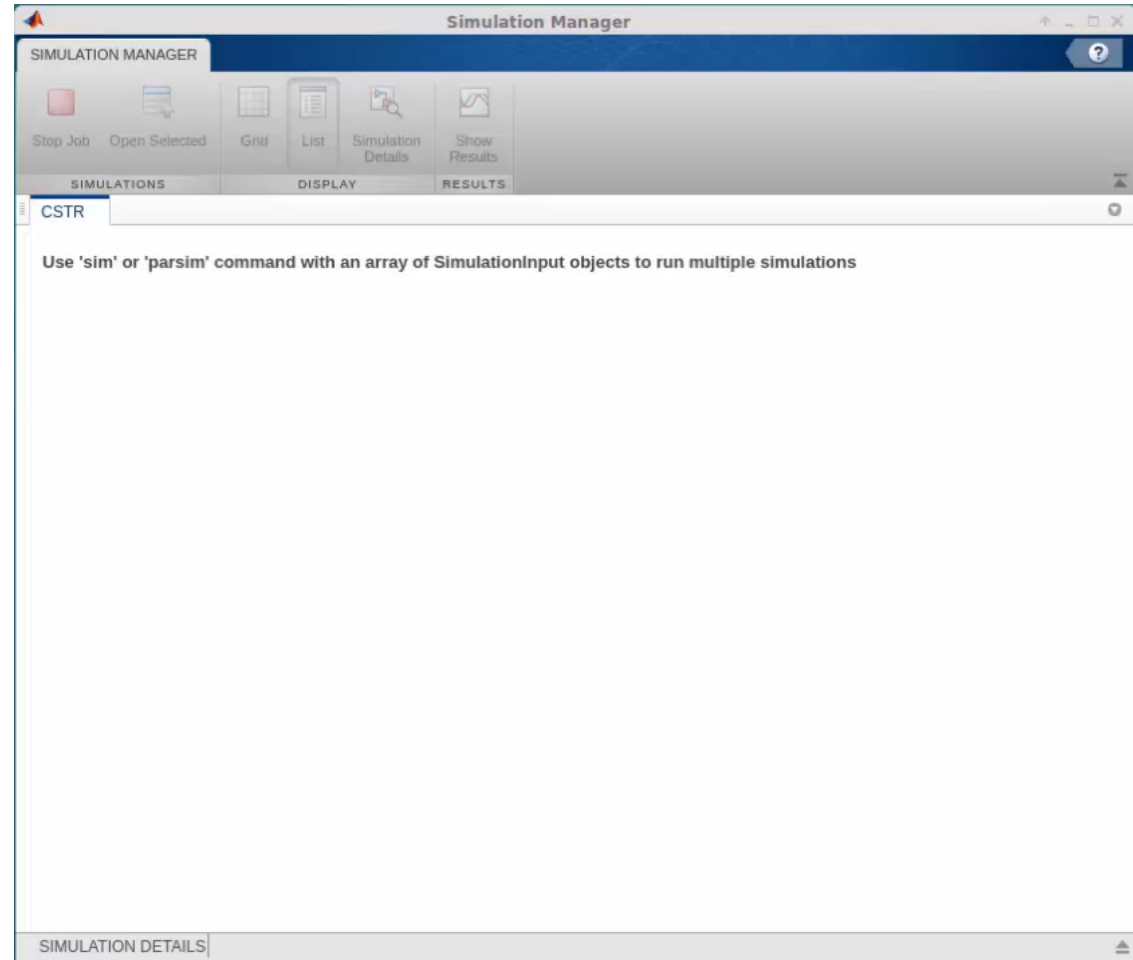
  

Path	Time Plot (Dark Band = Self Time)	Total Time (s)	Self Time (s)	Number of Calls
...emo_antiwindup		14.982	0.076	1
compilePhase		9.351	9.351	1
simulationPhase		4.872	0.610	1
...puts.Major		4.227	0.001	2001
...s.Major		3.533	3.533	2001
...s.Major		0.686	0.686	2001

[How Profiler Captures Performance Data](#)

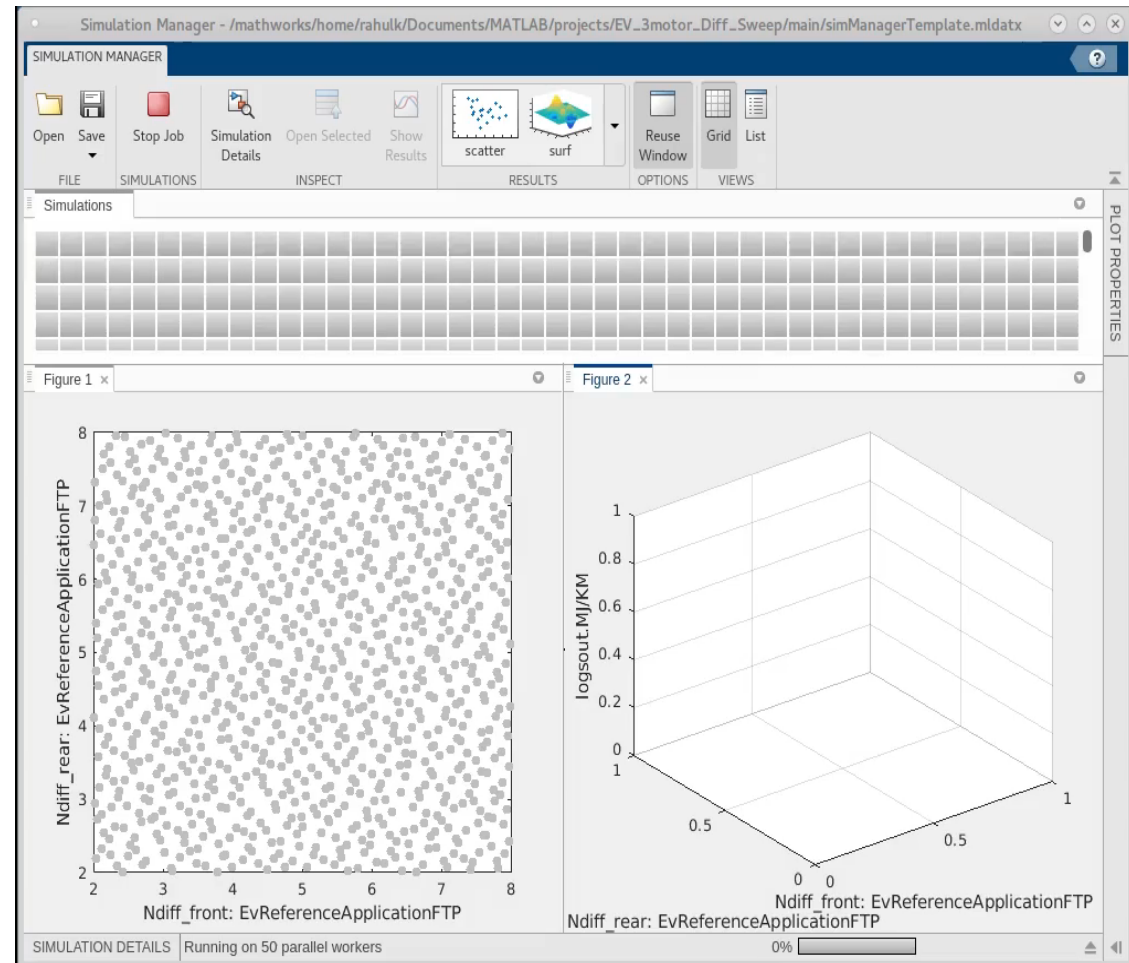
# Monitor multiple simulations at once with Simulation Manager

- View the progress of the simulations
- Examine simulation settings and diagnostics
- View simulation results in the Simulation Data Inspector



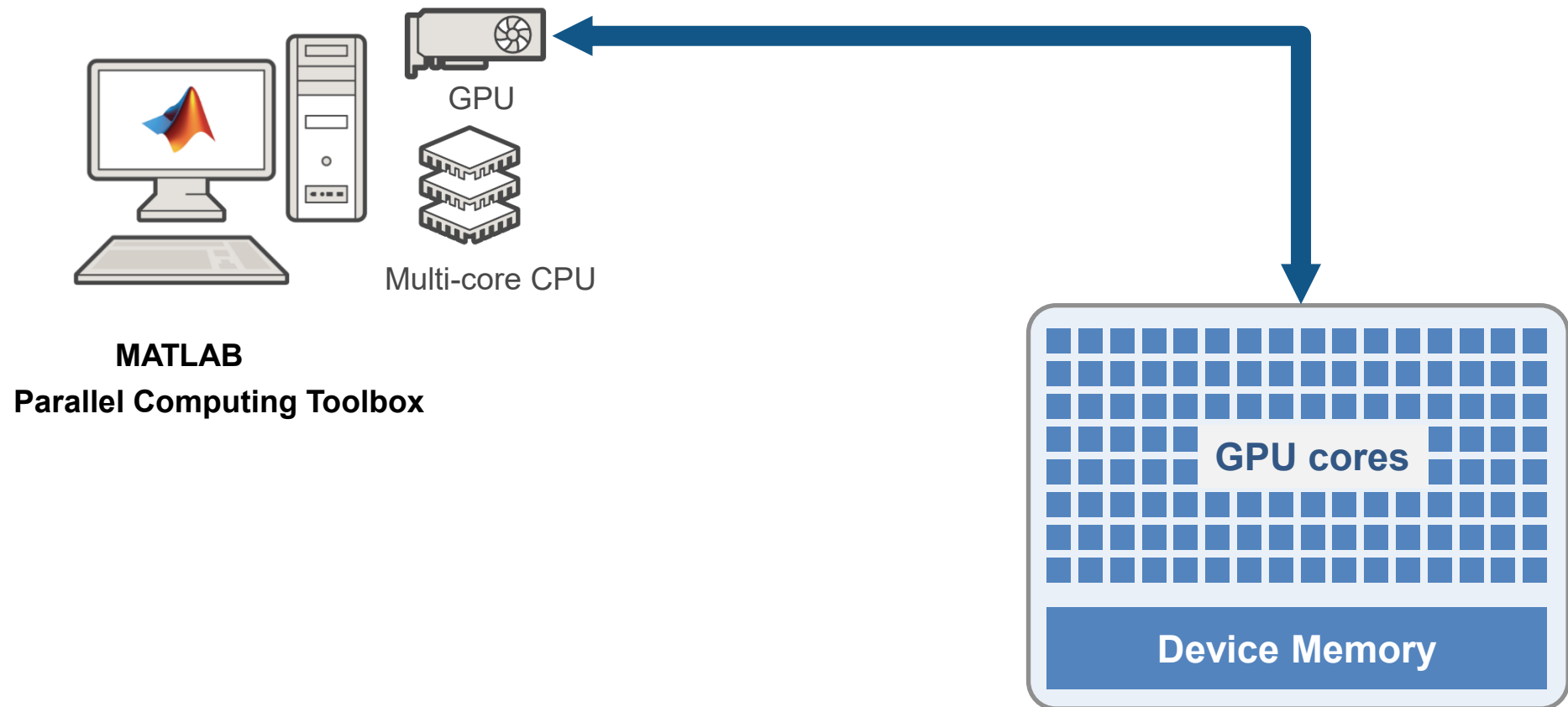
# Simulation Manager can also be used to inspect the variation in outputs with parameters

- Visualize simulation results as the simulations are running



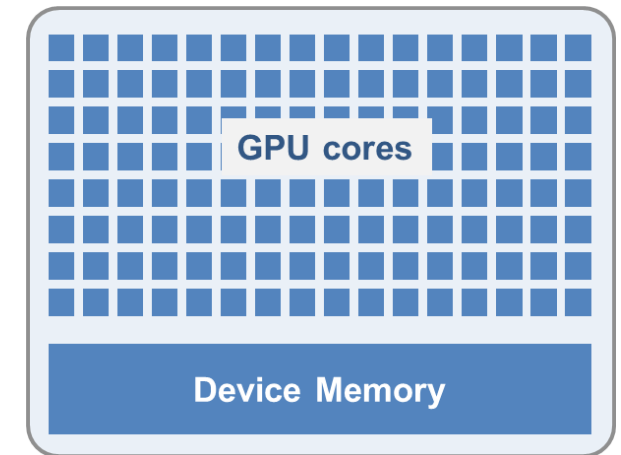
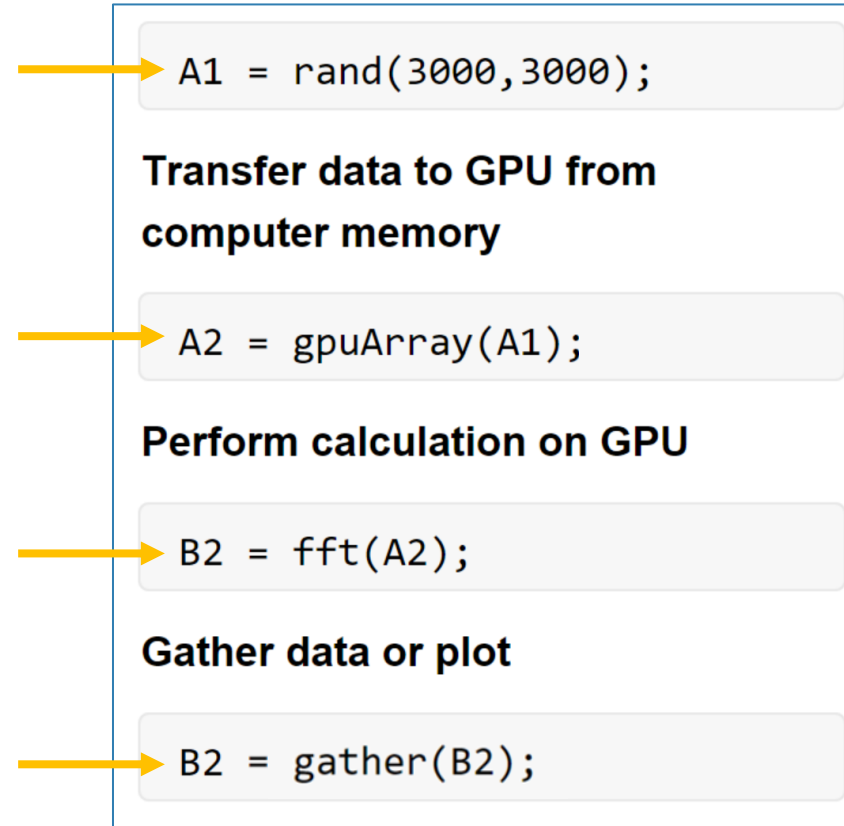
# Accelerate applications with NVIDIA GPUs

# Using NVIDIA GPUs with the Parallel Computing Toolbox



# Leverage your GPU to accelerate your MATLAB code

- Ideal Problems
  - massively parallel and/or vectorized operations
  - computationally intensive
- 999+ GPU-supported functions ([documentation](#))
- Use `gpuArray` and `gather` to transfer data between CPU & GPU



# How do I know if I have a supported GPU?

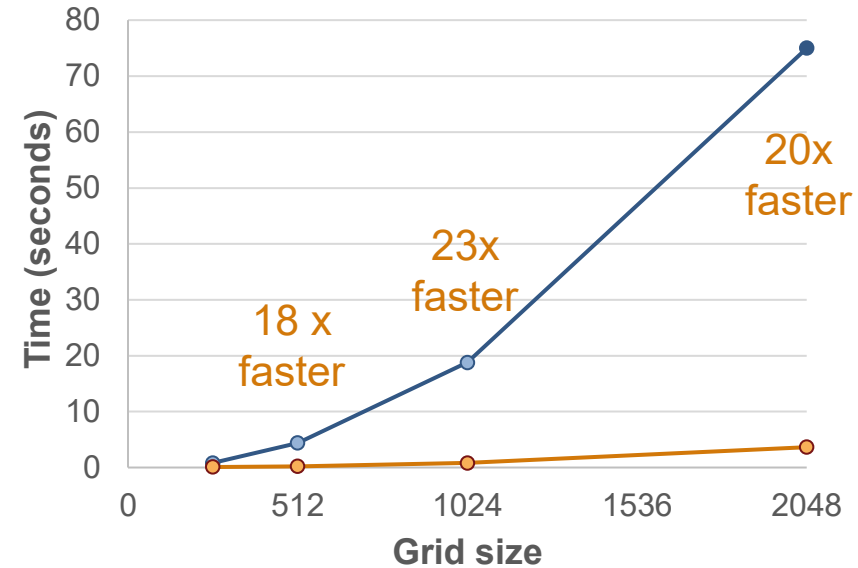
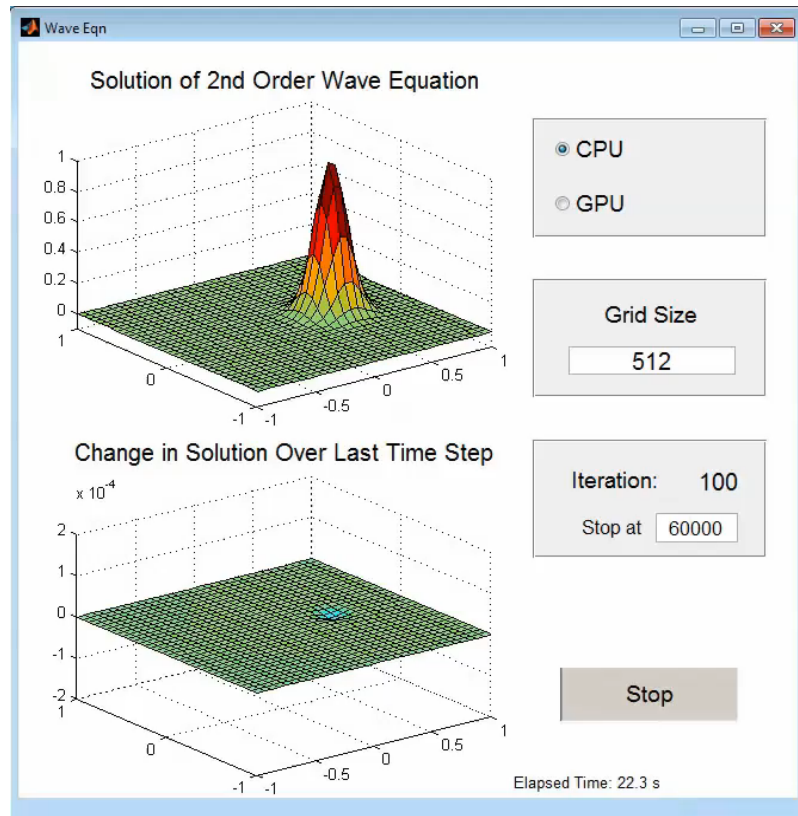
- In MATLAB, type:  
**>> gpuDevice**
- If you see a CUDA Device, you are good to go.
  - The key number to note is the 'ComputeCapability'
  - See [Support for NVIDIA GPU architecture by MATLAB release](#)

```
>> gpuDevice
ans =
    CUDADevice with properties:
        Name: 'Tesla V100-DGXS-16GB'
        Index: 1
        ComputeCapability: '7.0'
        SupportsDouble: 1
        DriverVersion: 10
        ToolkitVersion: 10
        MaxThreadsPerBlock: 1024
        MaxShmemPerBlock: 49152
        MaxThreadBlockSize: [1024 1024 64]
        MaxGridSize: [2.1475e+09 65535 65535]
        SIMDWidth: 32
        TotalMemory: 1.6908e+10
        AvailableMemory: 1.6130e+10
        MultiprocessorCount: 80
        ClockRateKHz: 1530000
        ComputeMode: 'Default'
        GPUOverlapsTransfers: 1
        KernelExecutionTimeout: 1
        CanMapHostMemory: 1
        DeviceSupported: 1
        DeviceSelected: 1

>> gpuDeviceCount
ans =
    4
```

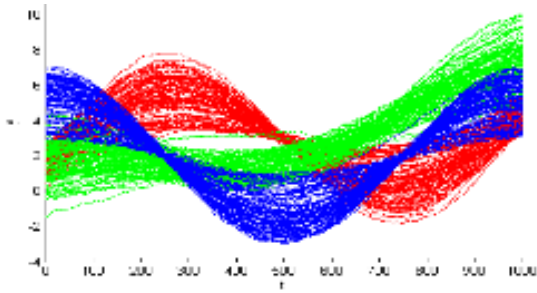
# Run Same Code on CPU and GPU

## Demo: Solving 2nd Order Wave Equation

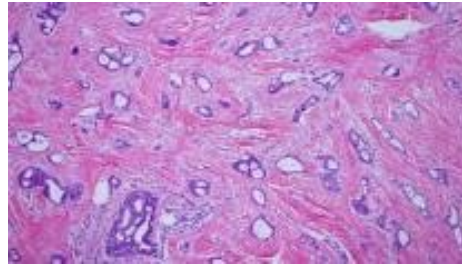


CPU	GPU
Intel(R) Xeon(R) W3550 3.06GHz 4 cores memory bandwidth 25.6 Gb/s	NVIDIA Tesla K20c 706MHz 2496 cores memory bandwidth 208 Gb/s

# Speeding up MATLAB Applications with GPUs



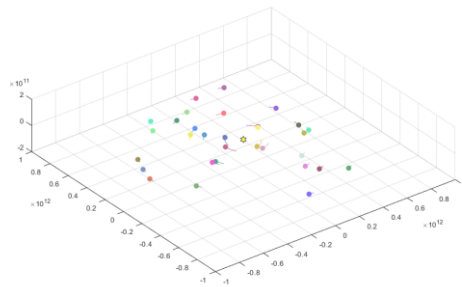
**10x speedup**  
*K-means clustering algorithm*



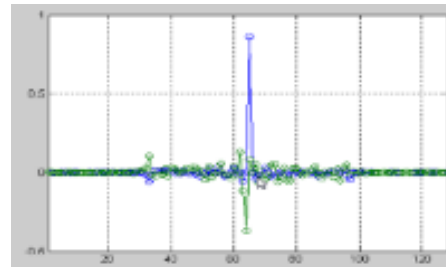
**14x speedup**  
*template matching routine*



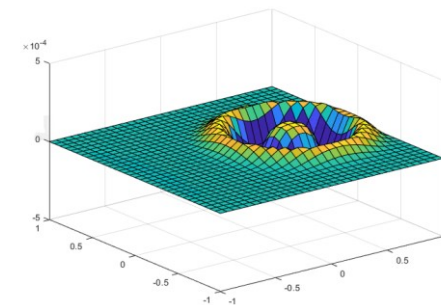
**12x speedup**  
*using Black-Scholes model*



**44x speedup**  
*simulating the movement of celestial objects*



**4x speedup**  
*adaptive filtering routine*

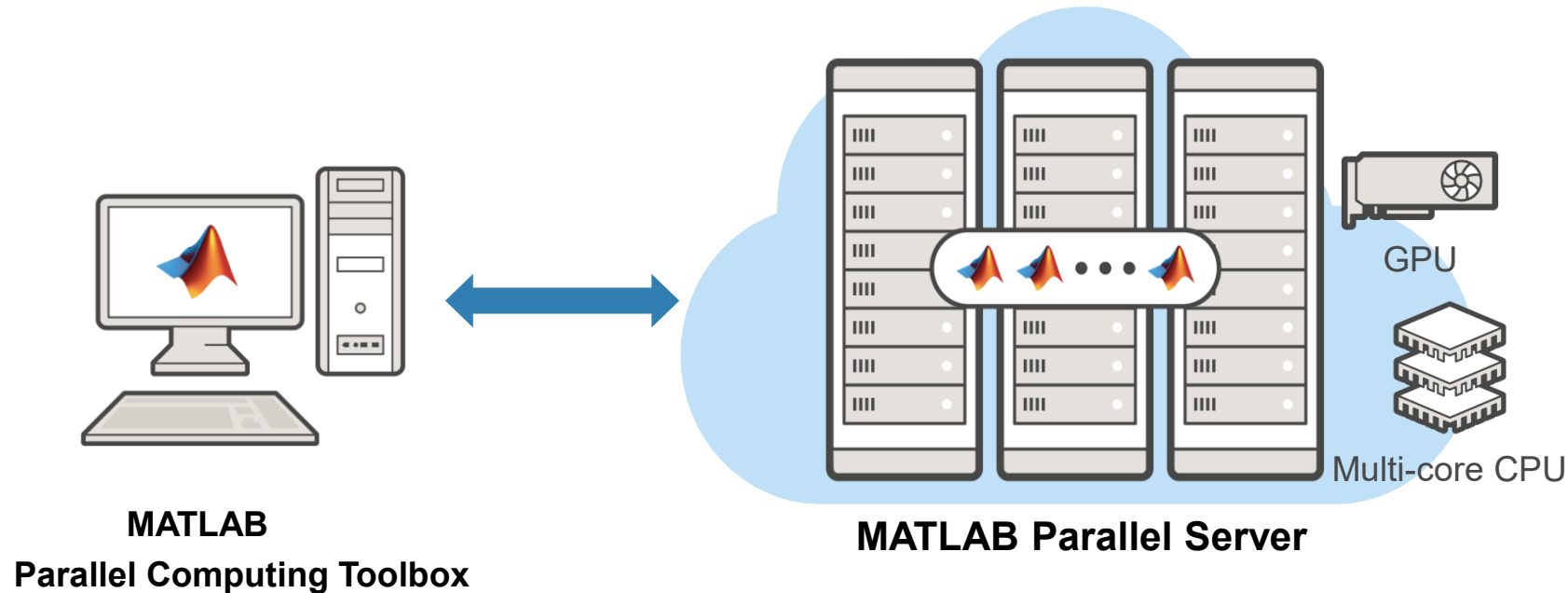


**23x speedup**  
*wave equation solving*

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

# Scaling to Cluster with MATLAB Parallel Server (Outlook)

# Parallel computing on your desktop, clusters, and clouds

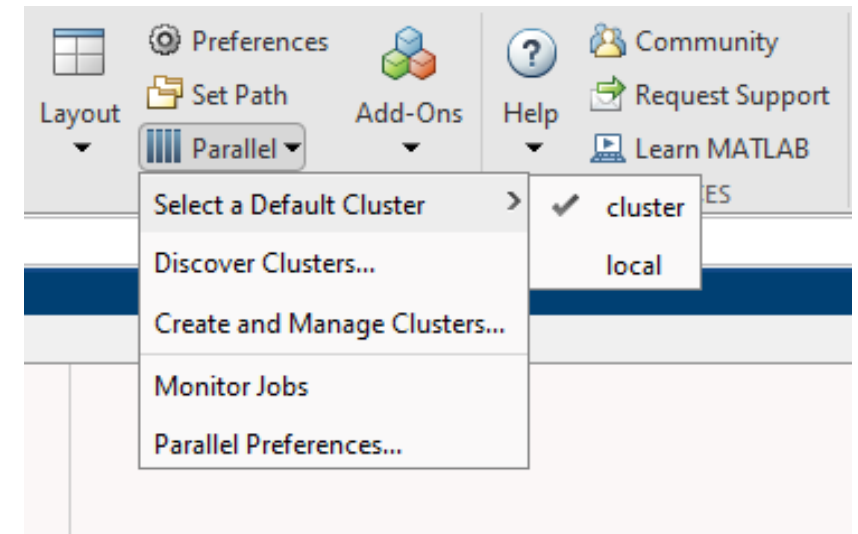


- Prototype and develop on the desktop
- Integrate with your infrastructure
- Access directly through MATLAB

# Scale to clusters and clouds

With MATLAB Parallel Server, you can...

- Use more hardware with minimal code change
- Submit to on-premise or cloud clusters
- Support cross-platform submission
  - Windows client to Linux cluster



# Interactive parallel computing

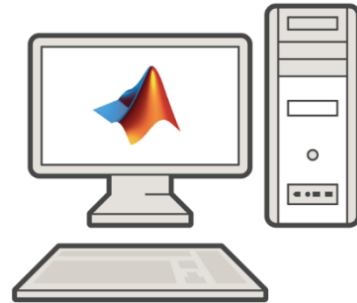
## Leverage cluster resources in MATLAB

```
>> parpool(myCluster, 3)
```

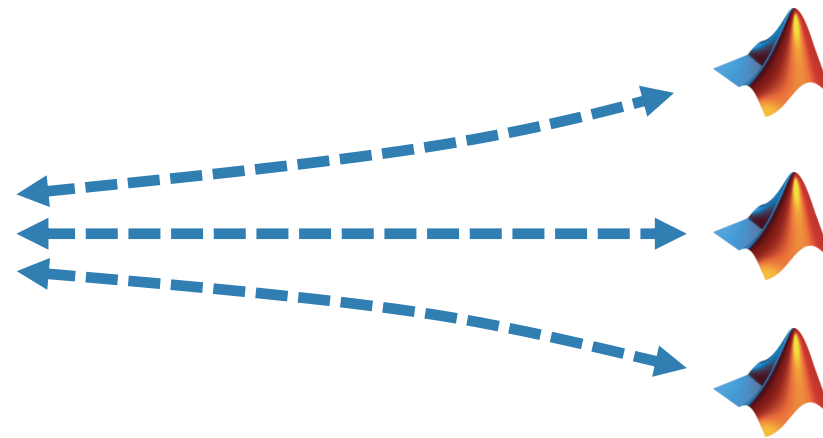
```
>> myScript
```

```
myScript.m:
```

```
a = zeros(5, 1);  
b = pi;  
parfor i = 1:5  
    a(i) = i + b;  
end
```



**MATLAB**  
Parallel Computing Toolbox

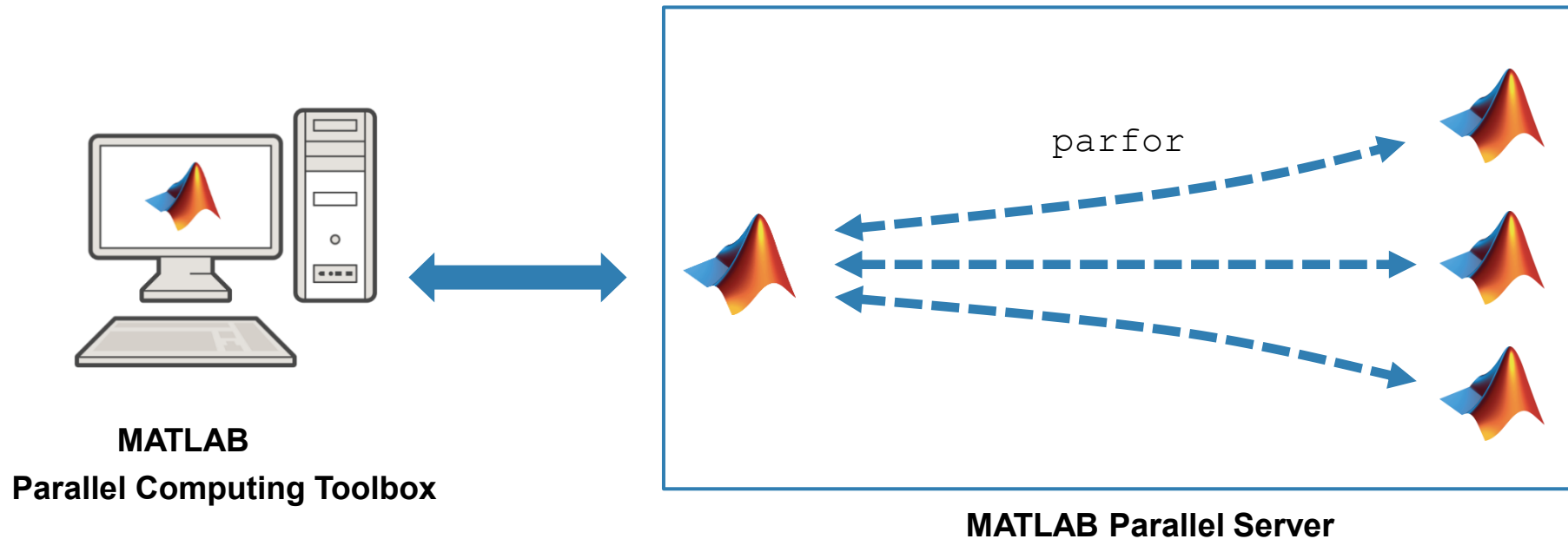


**MATLAB Parallel Server**

# batch simplifies offloading computations

Submit MATLAB jobs to the cluster

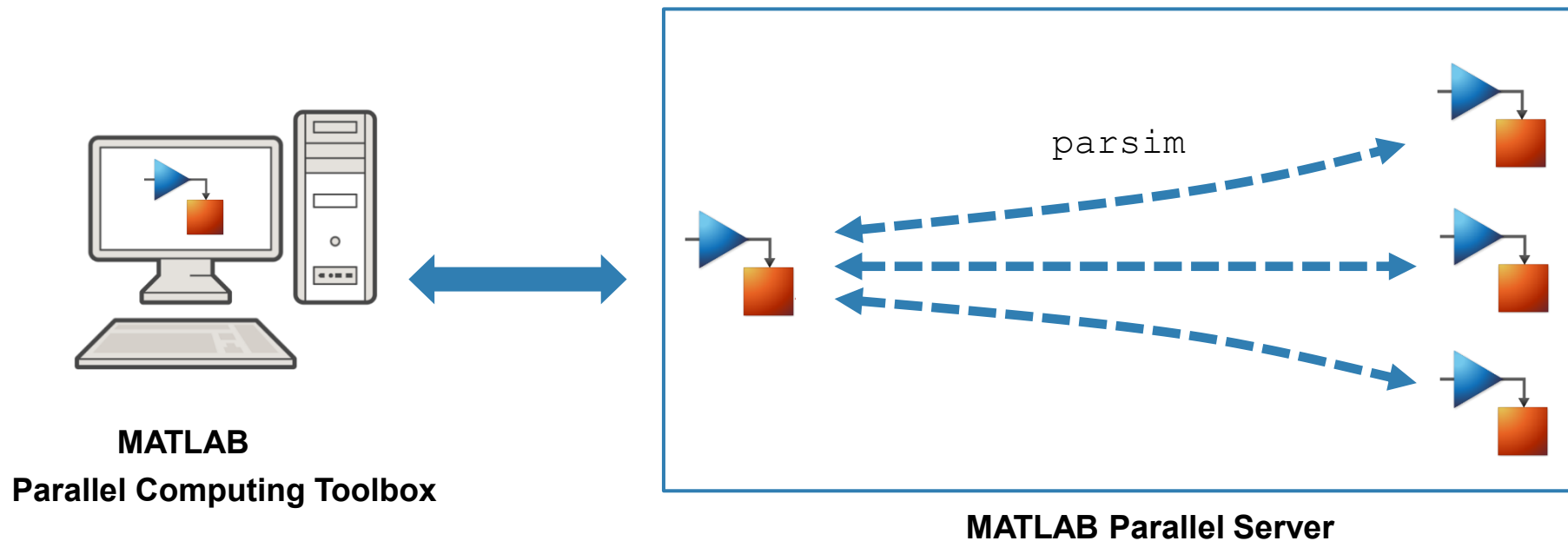
```
>> job = batch(myCluster, 'myScript', 'Pool', 3)
```



# batch simplifies offloading simulations

## Submit Simulink jobs to the cluster

```
>> job = batchsim(myCluster,in,'Pool',3)
```



# Speed up a parameter sweep using `parfor` on a cluster with MATLAB Parallel Server

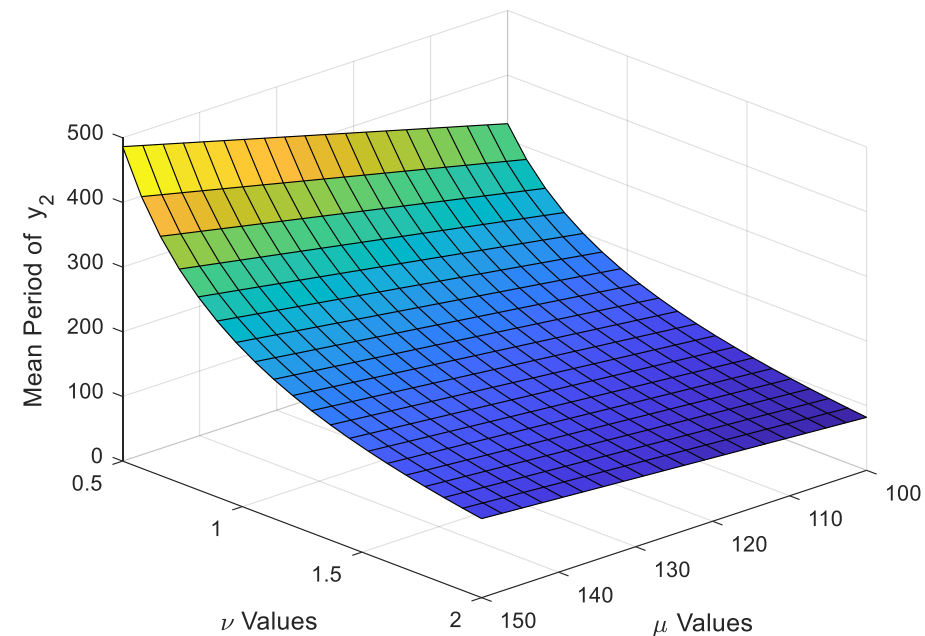
## Demo: Parameter sweep for van der Pol oscillator

- System of ODEs

$$\dot{y}_1 = \nu y_2$$

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

- Compute mean period of  $y$
- Use `parfor`, study impact of  $\nu, \mu$

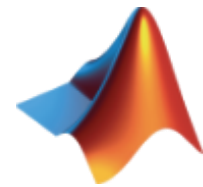


# Working with Big Data (Optional)

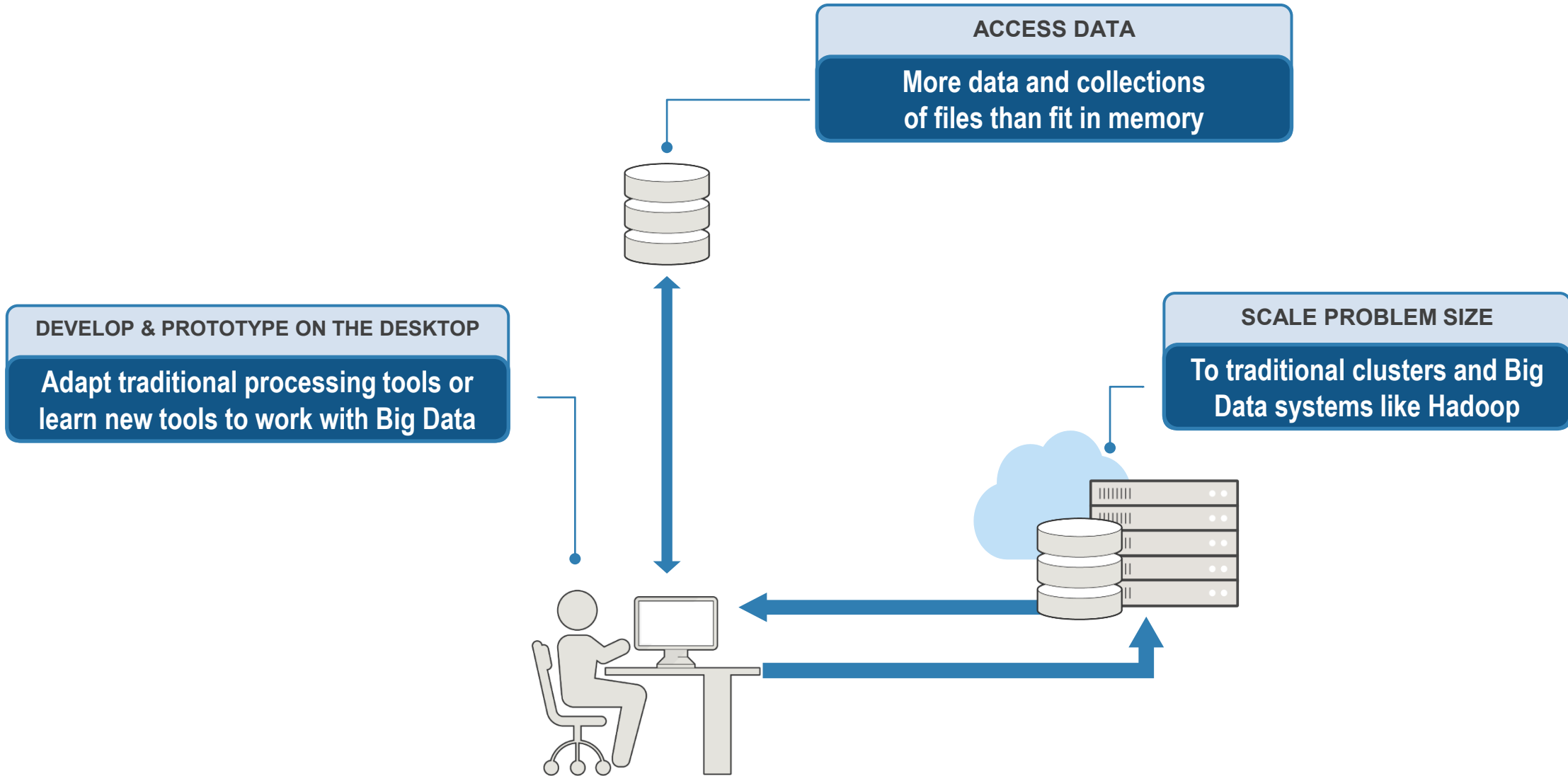
## Big Data capabilities in MATLAB

Wouldn't it be nice if we could:

- Easily access data however it is stored?
- Prototype algorithms quickly using small data sets?
- And then scale up to big data sets running on large clusters?
- **All using the same intuitive MATLAB syntax we are used to?**



# Big data workflow



# Access data with datastore

- **For:**
  - Handling collections of files or large files
- **Provides:**
  - Preview and configure I/O properties
  - Read data into memory (*all at once, or incrementally*)
  - Transform data one file at a time for data engineering workflows
  - Combine with tall arrays to analyze the entire out-of-memory dataset with few code changes

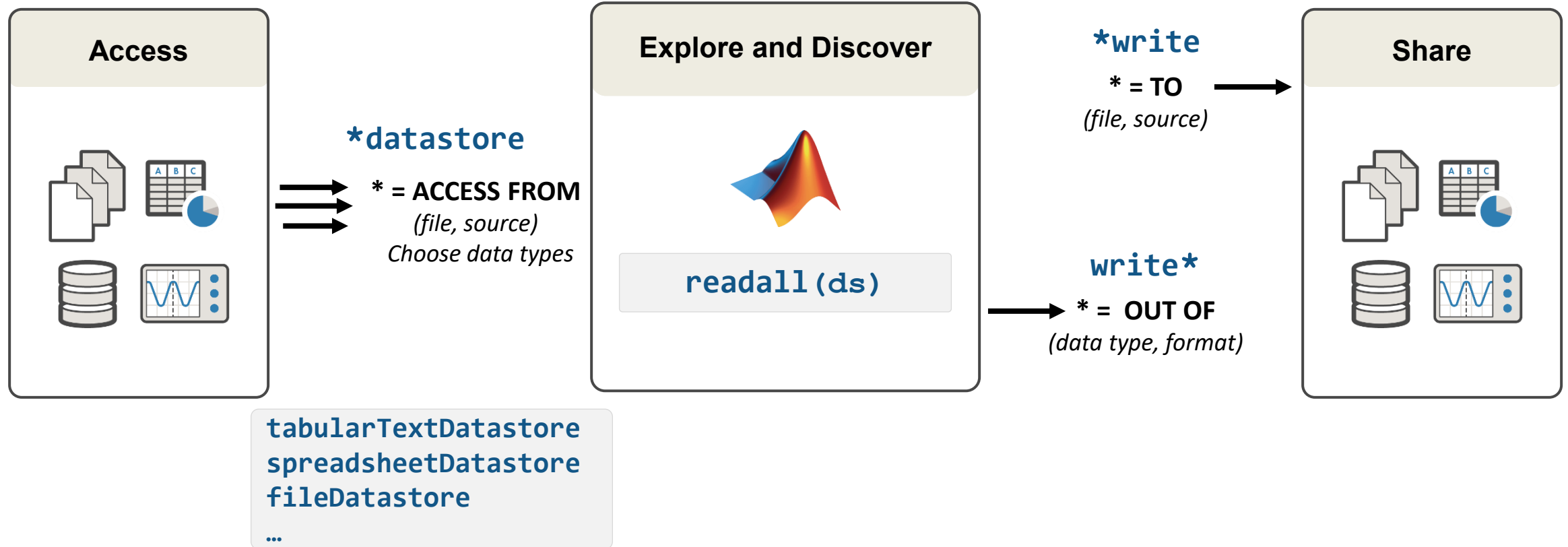
```
ds = datastore("data_flat/*.csv")

    Files: {
        'C:\Marketing\WhatsNew\datastores\data_1'
        'C:\Marketing\WhatsNew\datastores\data_2'
        'C:\Marketing\WhatsNew\datastores\data_3'
        ... and 86 more
    }
    FileEncoding: 'UTF-8'
AlternateFileSystemRoots: {}
    PreserveVariableNames: false
    ReadVariableNames: true
    VariableNames: {'TimeStamp', 'TimeZone', 'Name' ... and 86 more}
    DatetimeLocale: en_US

Text Format Properties:
    NumHeaderLines: 0
    Delimiter: ','
    RowDelimiter: '\r\n'
    TreatAsMissing: ''
    MissingValue: NaN

Advanced Text Format Properties:
    TextscanFormats: {'%{MM/dd/yyyy HH:mm:ss}D', '%q', '%q'}
```

# Use datastores for reading collections of files into memory



[Select Datastore for File Format or Application](#)

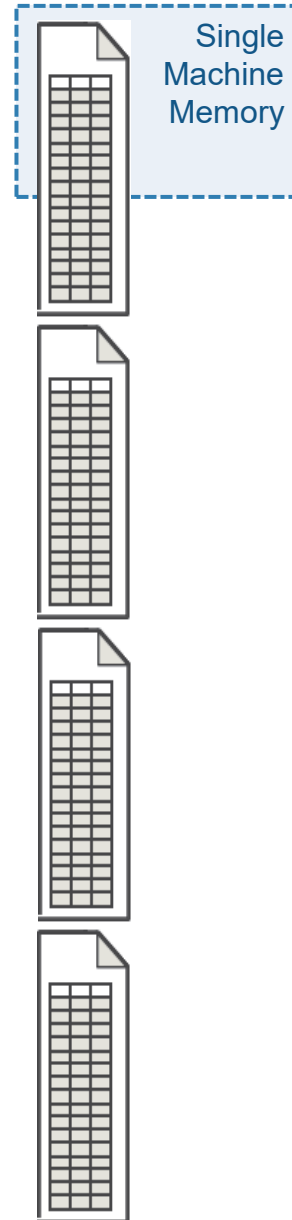
## tall arrays

- Data type designed for data that doesn't fit into memory
- Lots of observations (hence "tall")
- Looks like a normal MATLAB array
  - Supports numeric types, tables, datetimes, strings, etc.
  - Supports several hundred functions for basic math, stats, indexing, etc.
  - Statistics and Machine Learning Toolbox support (clustering, classification, etc.)



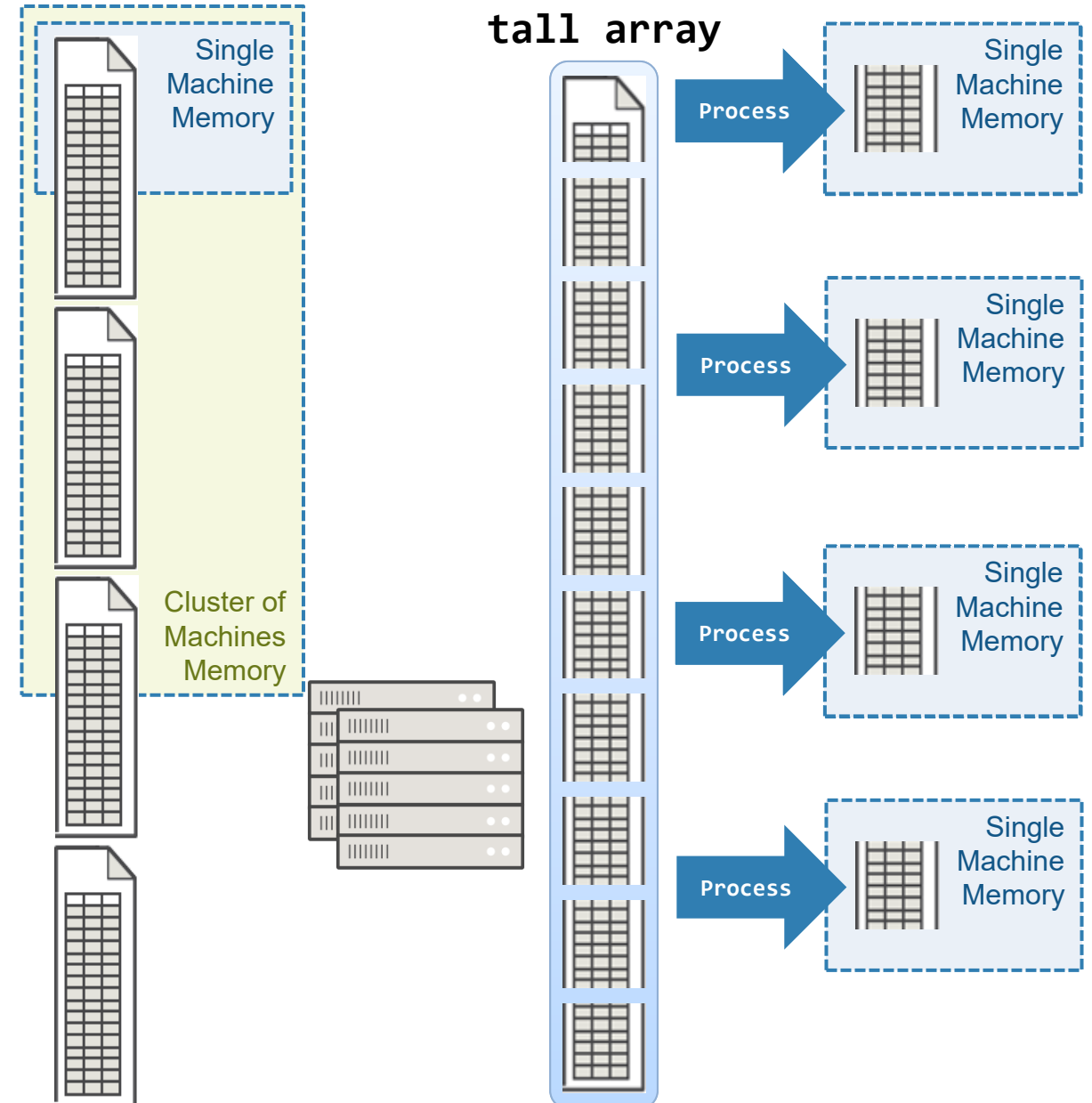
## tall arrays

- Automatically breaks data up into small “chunks” that fit in memory
- Tall arrays scan through the dataset one “chunk” at a time
- Processing code for tall arrays is the same as ordinary arrays



## tall arrays

- With Parallel Computing Toolbox, process several “chunks” at once
- Can scale up to clusters with MATLAB Parallel Server



# Big Data Without Big Changes

## One file

### Access Data

```
measured = readtable('PumpData.csv');  
measured = table2timetable(measured);
```

### Preprocess Data

#### Select data of interest

```
measured = measured(timerange(seconds(1),seconds(2)), 'Speed')
```

#### Work with missing data

```
measured = fillmissing(measured, 'linear');
```

#### Calculate statistics

```
m = mean(measured.Speed);  
s = std(measured.Speed);
```

## One hundred files

### Access Data

```
measured = datastore('PumpData*.csv');  
measured = tall(measured);  
measured = table2timetable(measured);
```

### Preprocess Data

#### Select data of interest

```
measured = measured(timerange(seconds(1),seconds(2)), 'Speed')
```

#### Work with missing data

```
measured = fillmissing(measured, 'linear');
```

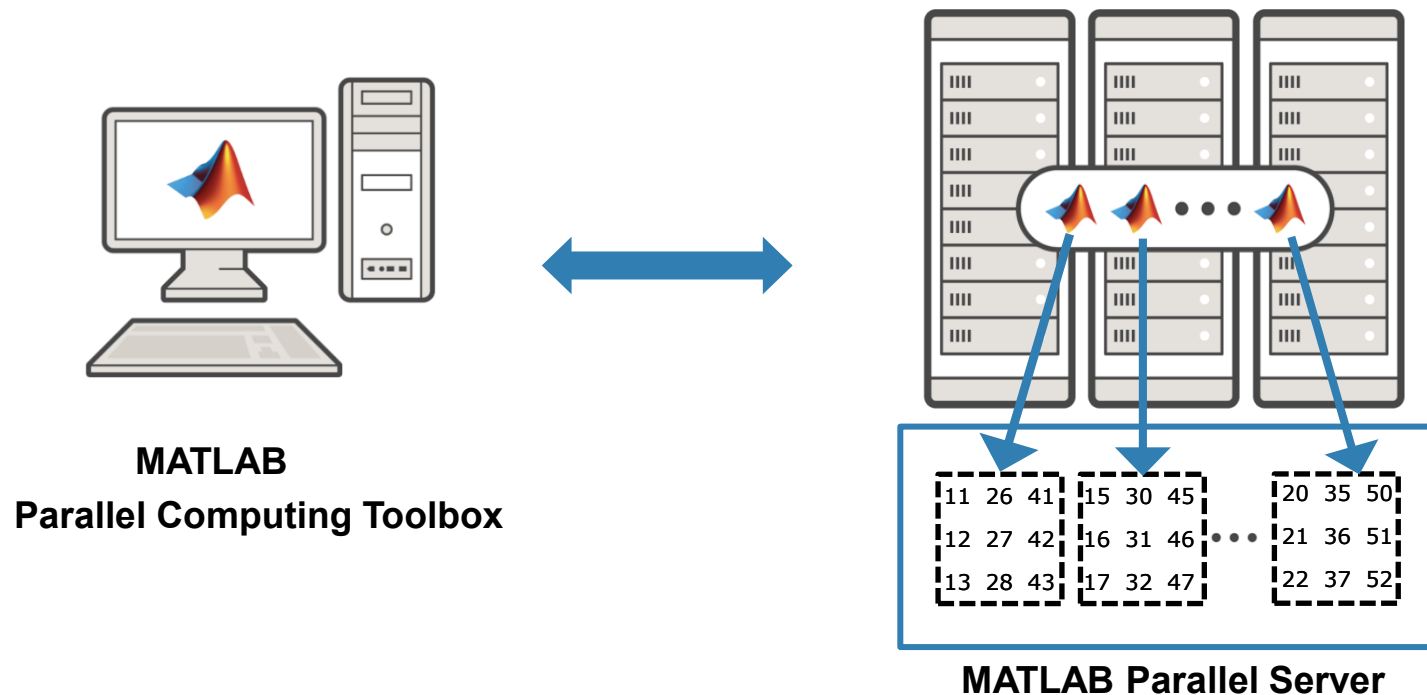
#### Calculate statistics

```
m = mean(measured.Speed);  
s = std(measured.Speed);
```

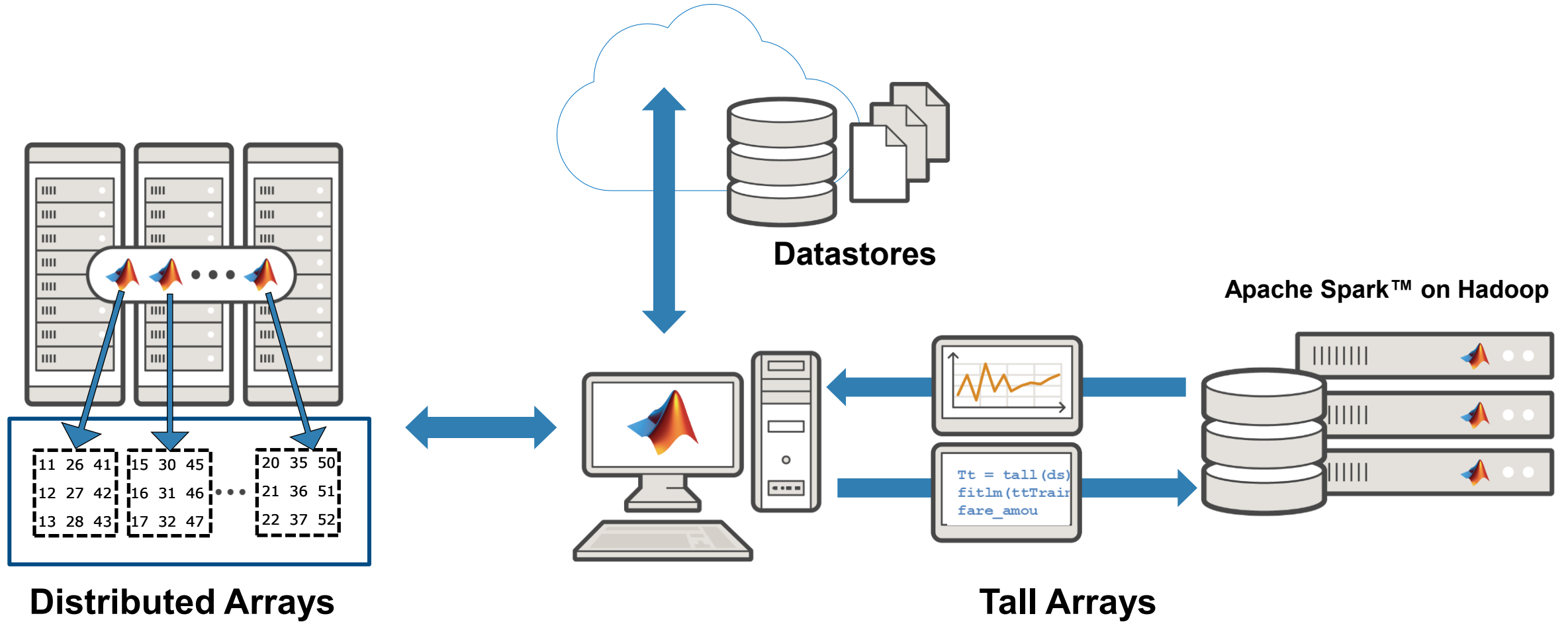
```
[m,s] = gather(m,s);
```

# distributed arrays

- Keep large datasets in-memory, split among workers running on a cluster
- Common Actions: Matrix Manipulation & Linear Algebra and Signal Processing
- Several hundred MATLAB functions overloaded for distributed arrays



# Big Data Capabilities in MATLAB with Parallel Computing

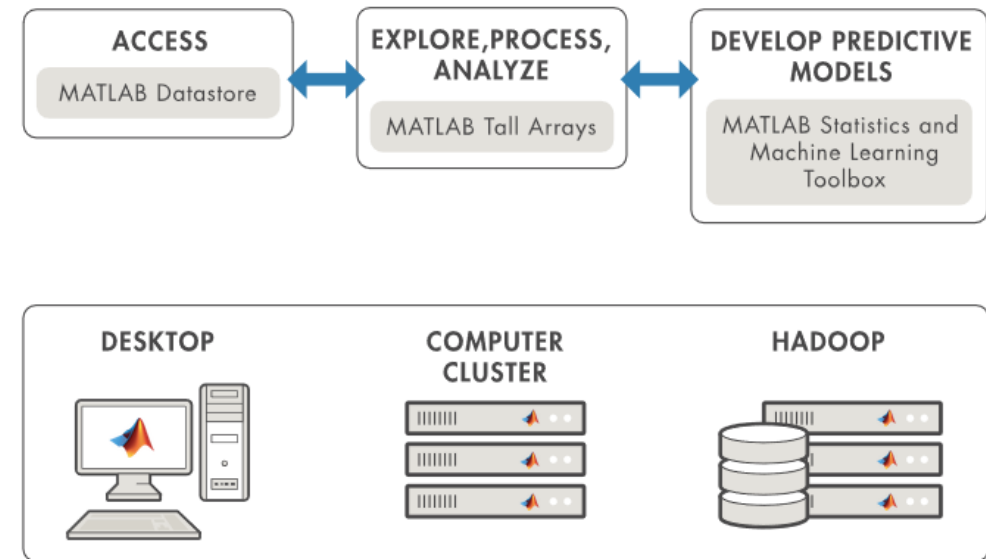


## Summary – Working with Big Data in MATLAB

- Use **datstores** to manage data processing from large collections of files.
- Use **Tall Arrays** to process files too big to fit in memory.
- Use **Distributed Arrays** and **GPU Arrays** to parallelize problems for solving on multiple workers at once.
- Use **Parallel Computing Toolbox** (on Desktop) or **MATLAB Parallel Server** (on clusters) to scale-up solutions.

# Learn More on Big Data

- [Strategies for Efficient Use of Memory](#)
- [Resolving "Out of Memory" Errors](#)
- [Big Data with MATLAB](#)
- [MATLAB Tall Arrays in Action](#)



# Summary

# Run MATLAB on Multicore Machines



- Built-in multithreading (implicit)
  - Automatically enabled in MATLAB
  - Multiple computational threads in a single MATLAB session
  - Functions such as `fft`, `eig`, `svd`, and `sort` are multithreaded in MATLAB
  - Additionally, many image processing functions are multithreaded
- Parallel computing using explicit techniques
  - Multiple computation engines (workers) controlled by a single session
  - High-level constructs to let you parallelize MATLAB applications
  - Perform MATLAB computations on GPUs
  - Scale parallel applications beyond a single machine to clusters and clouds

# Scaling MATLAB applications and Simulink simulations

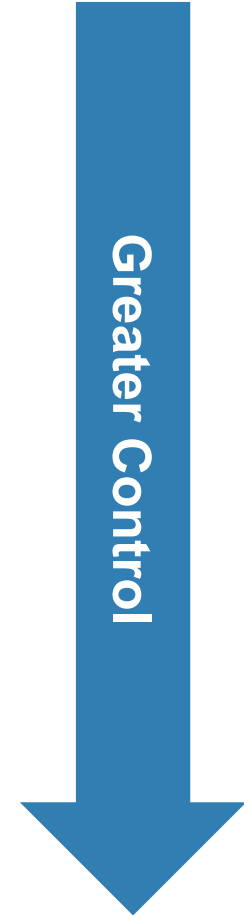


## **Automatic parallel support in toolboxes**

`(..., 'UseParallel', true)`

Common programming constructs

Advanced programming constructs



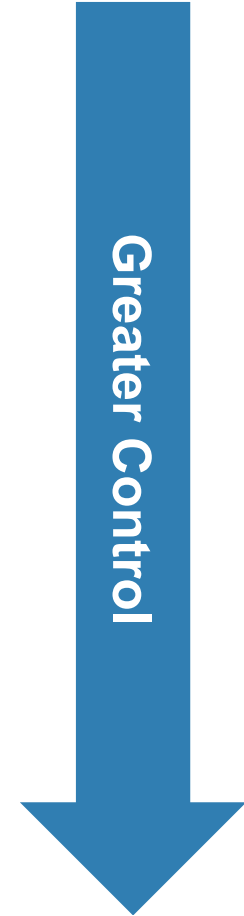
# Scaling MATLAB applications and Simulink simulations



Automatic parallel support in toolboxes

**Common programming constructs**  
(`parfor`, `parfeval`, `parsim`, ...)

Advanced programming constructs



# Scaling MATLAB applications and Simulink simulations



Automatic parallel support in toolboxes

Common programming constructs

**Advanced programming constructs**

(`spmd`, `parfevalOnAll`, ...)



# Summary

- Use **Parallel Computing Toolbox** on the Desktop to speed up your computationally intensive applications using multiple CPU cores or GPUs.
- Scale up to Clusters or Cloud using **MATLAB Parallel Server**
- Use Big Data capabilities such as Tall and Distributed Arrays, Datastores to further scale up solutions.

**Steve Schäfer**

*MathWorks Academia Group*

[steves@mathworks.com](mailto:steves@mathworks.com)

