Workshop: Parallel Computing with MATLAB

October 24, 2023

Dr. Mihaela Jarema
Academia Group
mjarema@mathworks.com

Agenda

- **Part I – Parallel Computing with MATLAB on the Desktop**
  - **Parallel Computing Toolbox**
  - **MATLAB Online**
- Part II – Parallel Computing with MATLAB on the JSC Cluster (6.11.2023)
  - MATLAB Parallel Server

Why use parallel computing?

Save time with parallel computing by carrying out computationally and data-intensive problems in parallel (simultaneously)

- distribute your tasks to be executed in parallel
- distribute your data to solve big data problems

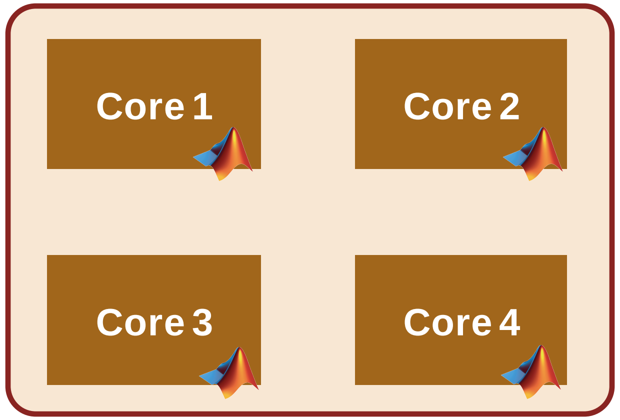
on your compute cores and GPUs, or scaled up to clusters and cloud computing

Why use parallel computing with MATLAB?

Save time with parallel computing by carrying out computationally and data-intensive problems in parallel (simultaneously)

- distribute your tasks to be executed in parallel
- distribute your data to solve big data problems

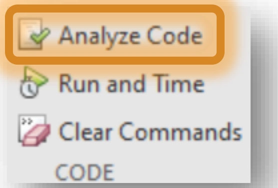
on your compute cores and GPUs, or scaled up to clusters and cloud computing with minimal code changes, so you can focus on your research use case.



CPU with 4 cores

! Before going parallel, make sure you optimize your serial code for best performance

- Use the **Code Analyzer** to automatically check your code for coding (and performance) problems.




```
1   tic
2   x = 0;
3   for k = 2:1e6
4       x(k) = x(k-1)+1;
5   end
6   toc
```

Elapsed time is 0.091058 seconds.

Line 4: Variable appears to change size on every loop iteration (within a script). Consider preallocating for speed. Details ▾

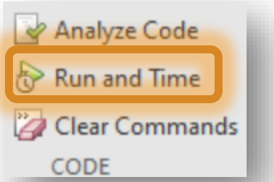
```
1   tic
2   x = zeros(1,1e6);
3   for k = 2:1e6
4       x(k) = x(k-1)+1;
5   end
6   toc
```

Elapsed time is 0.018484 seconds.

! Preallocating the maximum amount of space required for the array instead of letting MATLAB repeatedly reallocate memory for the growing array is about 5 times faster!

! Before going parallel, make sure you optimize your serial code for best performance

- Use the **Profiler** to find the code that runs slowest and evaluate possible performance improvements.




```
rng(1)
x = rand(1,1e6);
for k = 1:numel(x)
    if x(k)<.5
        x(k) = 0;
    end
end
```

! Use vectorization (matrix and vector operations) instead of for-loops!

Time	Calls	Line	
0.008	1	1	`rng(1)`
0.010	1	2	`x = rand(1,1e6);`
< 0.001	1	3	`for k = 1:numel(x)`
0.041	1000000	4	`if x(k)<.5`
0.016	499837	5	`x(k) = 0;`
0.032	1000000	6	`end`
0.032	1000000	7	`end`

Time	Calls	Line	
0.005	1	1	`rng(1)`
0.010	1	2	`x = rand(1,1e6);`
0.007	1	3	`x(x<.5) = 0;`

! Before going parallel, make sure you optimize your serial code for best performance with efficient programming practices

Pre-allocate memory instead of letting arrays be resized dynamically.

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

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

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

Place independent operations outside loops to avoid redundant computations.

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

Techniques to improve performance

! Before going parallel, make sure you optimize your serial code for best performance with efficient programming practices

(Advanced) Replace code with MEX functions

Documentation | Examples | Functions | Videos | Answers

mex

Build MEX function or engine application

Syntax

```
mex filenames
mex filenames api option1 ... optionN
```

```
mex -client engine filenames
mex -client engine filenames api option1 ... optionN
```

```
mex -setup [lang]
mex -setup -client engine [lang]
```

Scaling MATLAB applications and Simulink simulations

Ease of Use

Automatic parallel support in toolboxes

Common programming constructs

Advanced programming constructs

Greater Control

Take advantage of your multicore and multiprocessor computers automatically

with no programming effort
using built-in multithreading

just by setting a flag/preference
* with Parallel Computing Toolbox

MATLAB Multicore

What Is MATLAB Multicore?

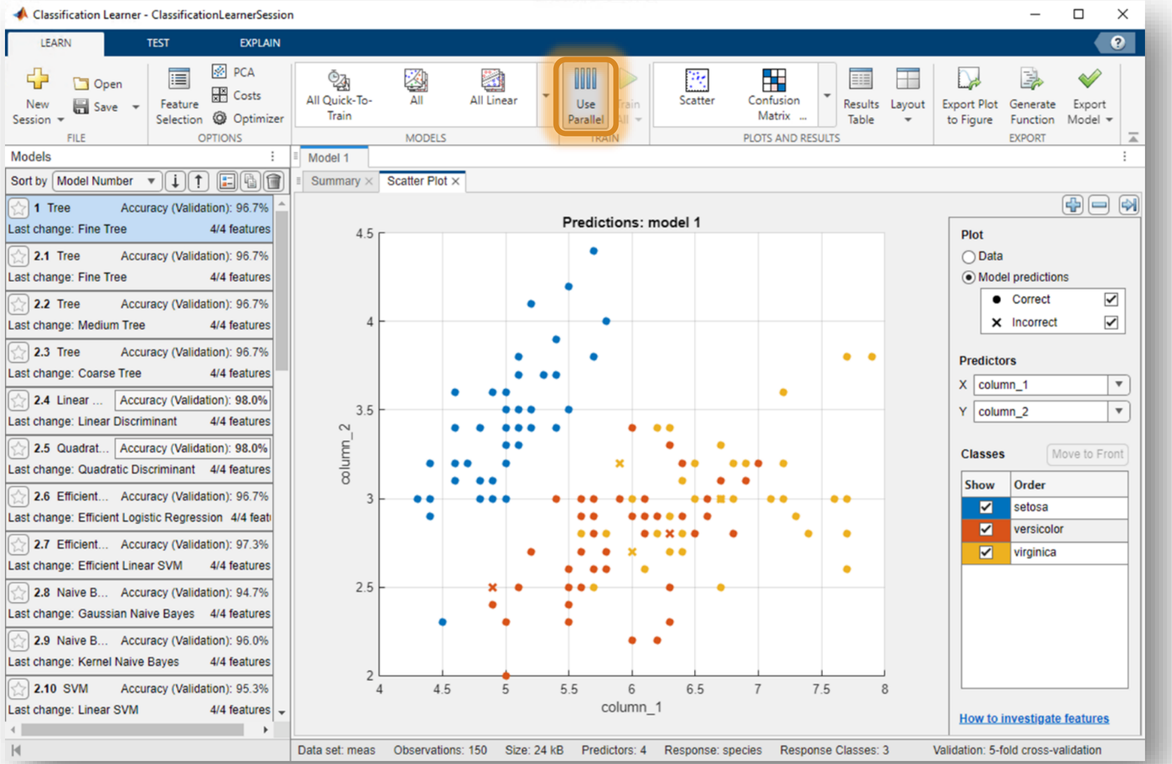
MATLAB® provides two main ways to take advantage of multicore and multiprocessor computers. By using the full computational power of your machine, you can run your MATLAB applications faster and more efficiently.

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.



Scaling MATLAB applications and Simulink simulations

Ease of Use

Automatic parallel support in toolboxes

Common programming constructs
(`parfor`, `parfeval`, `parsim`, …)

Advanced programming constructs

Greater Control

Scale further with parallel computing

MATLAB Multicore

What Is MATLAB Multicore?

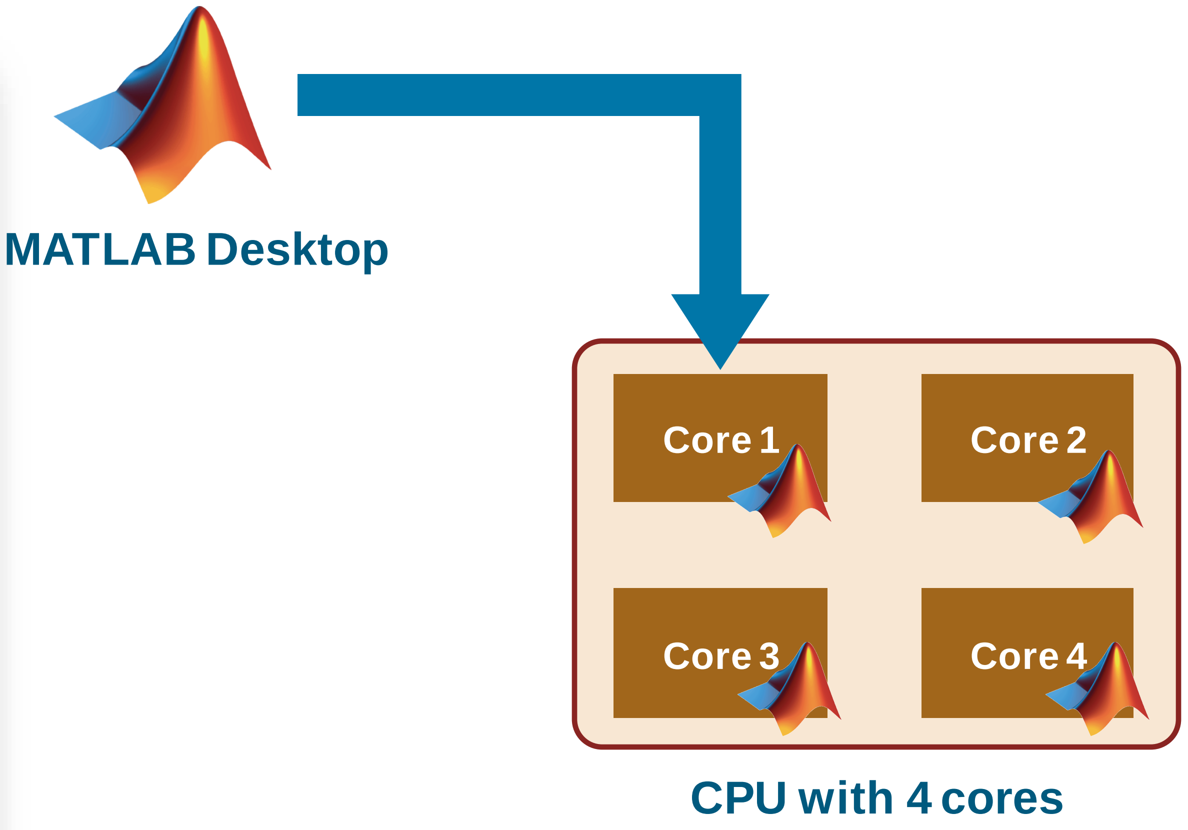
MATLAB® provides two main ways to take advantage of multicore and multiprocessor computers. By using the full computational power of your machine, you can run your MATLAB applications faster and more efficiently.

Built-in Multithreading

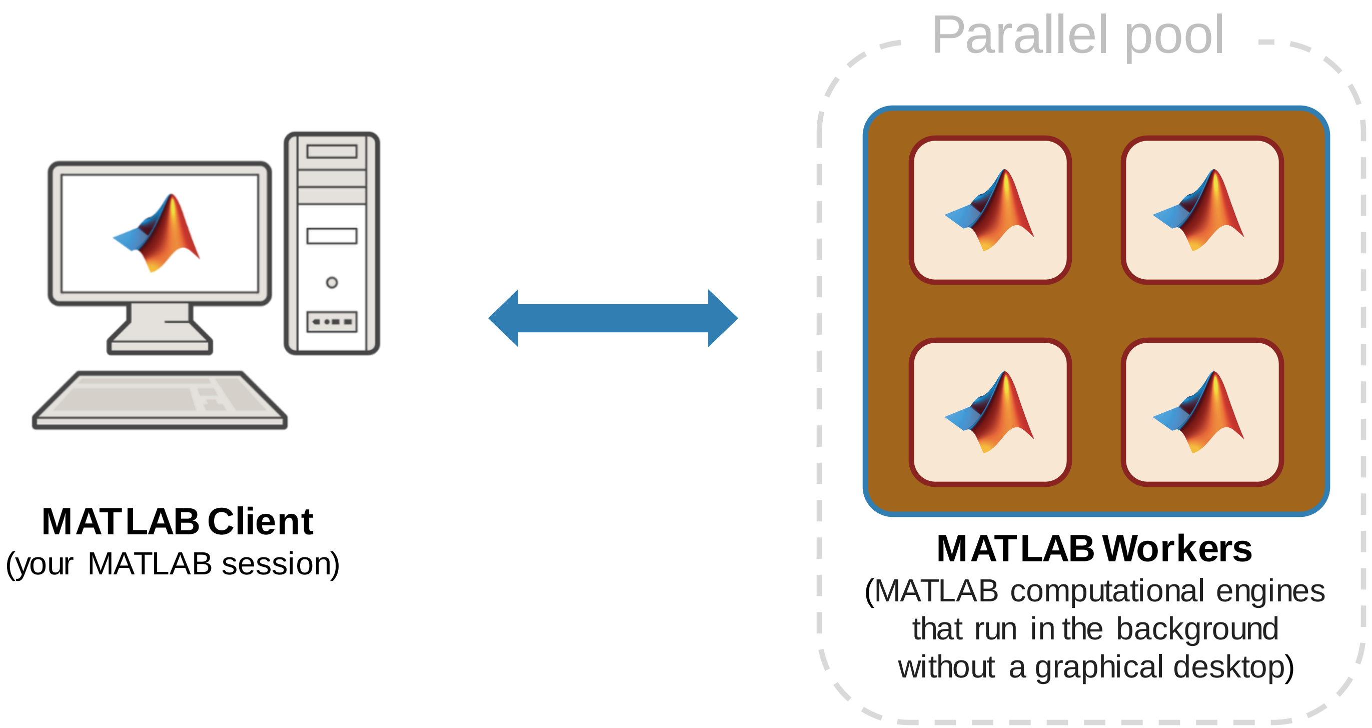
Linear algebra and numerical functions such as `fft`, `\` (`mldivide`), `eig`, `svd`, and `sort` are multithreaded in MATLAB. Multithreaded computations have been on by default in MATLAB since Release 2008a. These functions automatically execute on multiple computational threads in a single MATLAB session, 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.

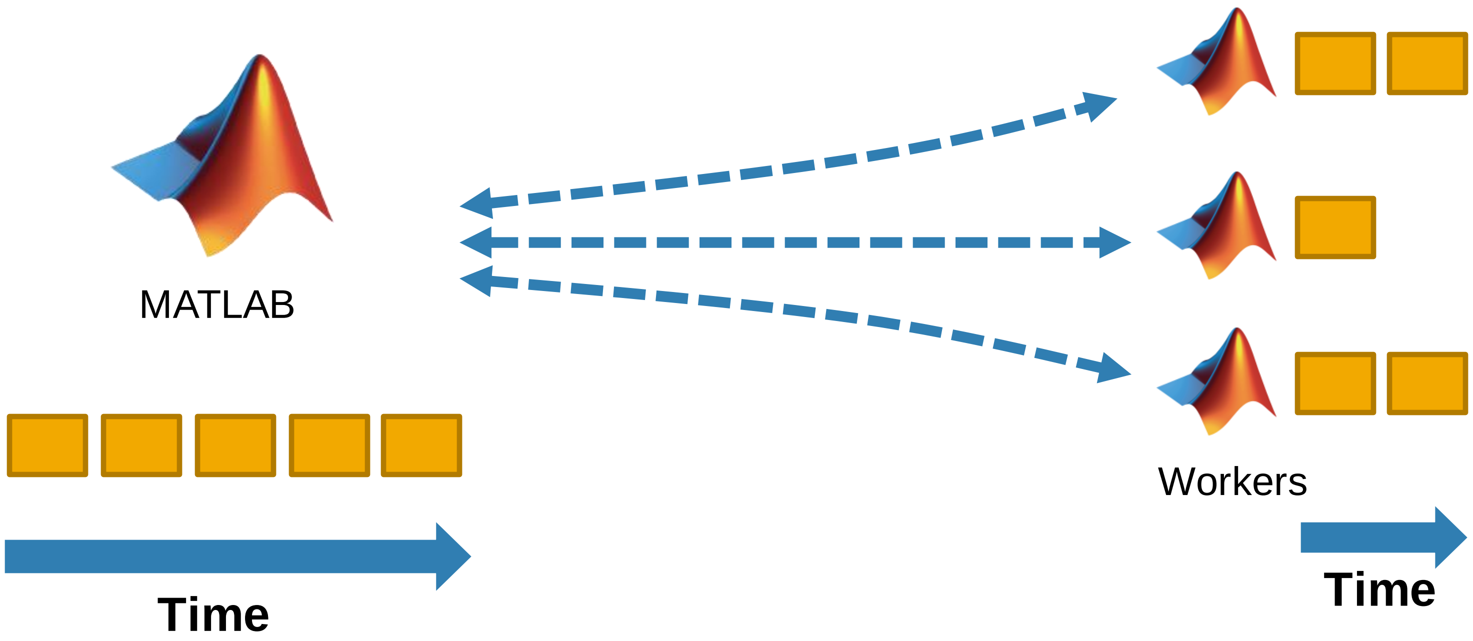


Run multiple iterations by utilizing multiple CPU cores



Explicit parallelism using `parfor` (parallel `for`-loop)

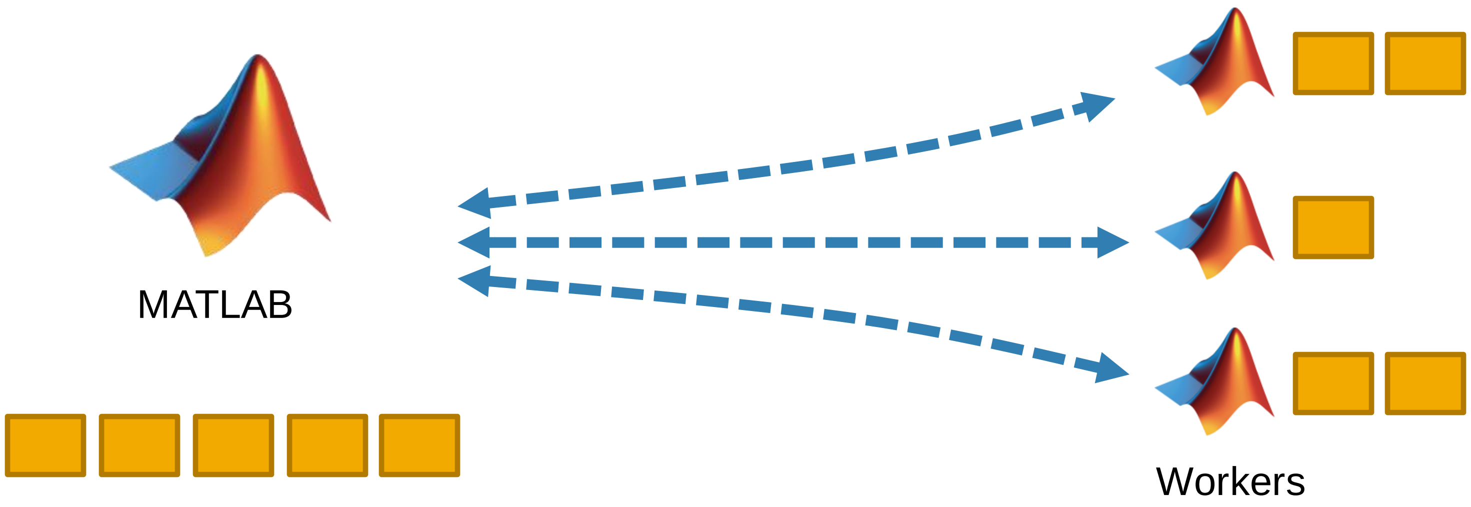
- Run iterations in parallel
- Examples: parameter sweeps, Monte Carlo simulations



Explicit parallelism using `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)
```



Hands-On Exercise: Convert a simple `for` to `parfor`

Getting Started with `parfor`

In this exercise, we will convert a simple `for`-loop into a `parfor`-loop and understand the basic differences between the two types of loops.

```
doc parfor
```

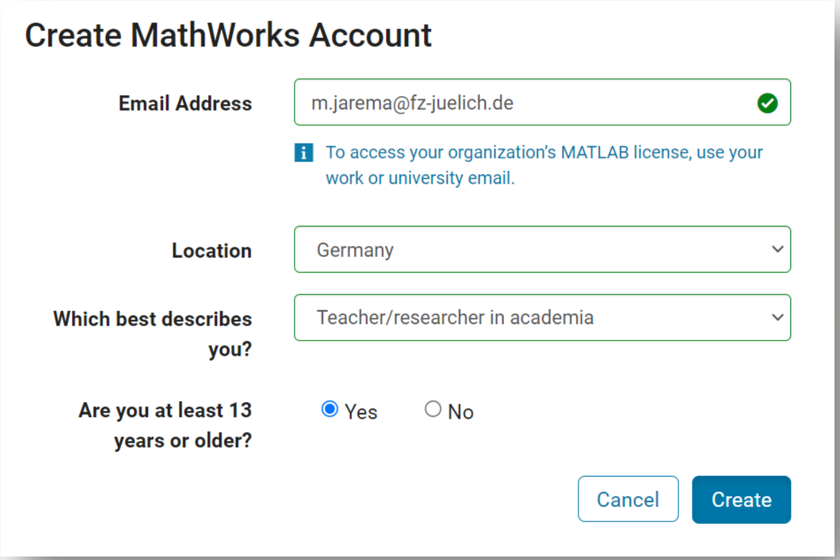
How much time will the following `for`-loop take?

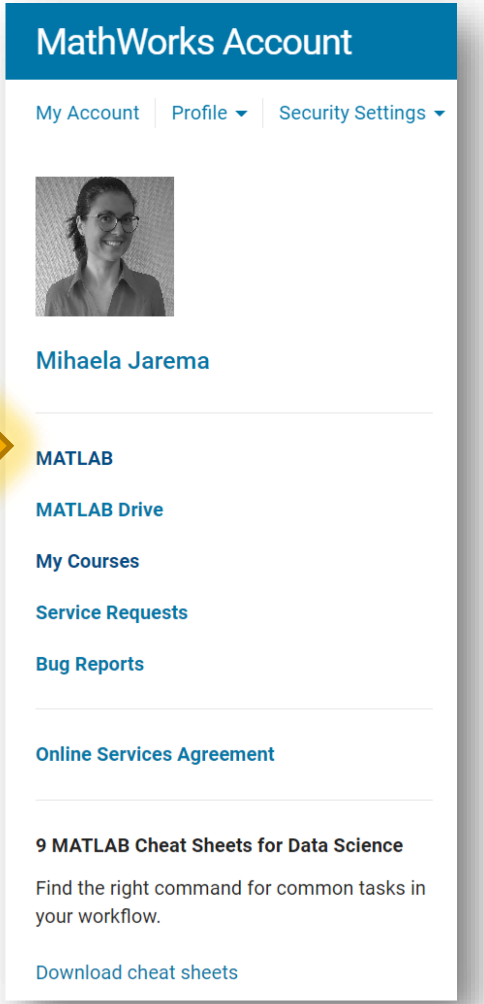
This is an example of a basic `for`-loop; in each iteration, `pause(1)` stops MATLAB execution for one second and then displays the index `idx` of the iteration. Since there are 10 iterations, the `for`-loop takes about 10 seconds (the added `tic` and `toc` measure the time elapsed) and the indices are displayed sequentially, from 1 to 10.

```
tic
for idx = 1:10
    pause(1)
    disp(idx)
end
```

```
     1

     2

     3
```

Step 1: Create a free MathWorks account with your **@fz-juelich.de** email address



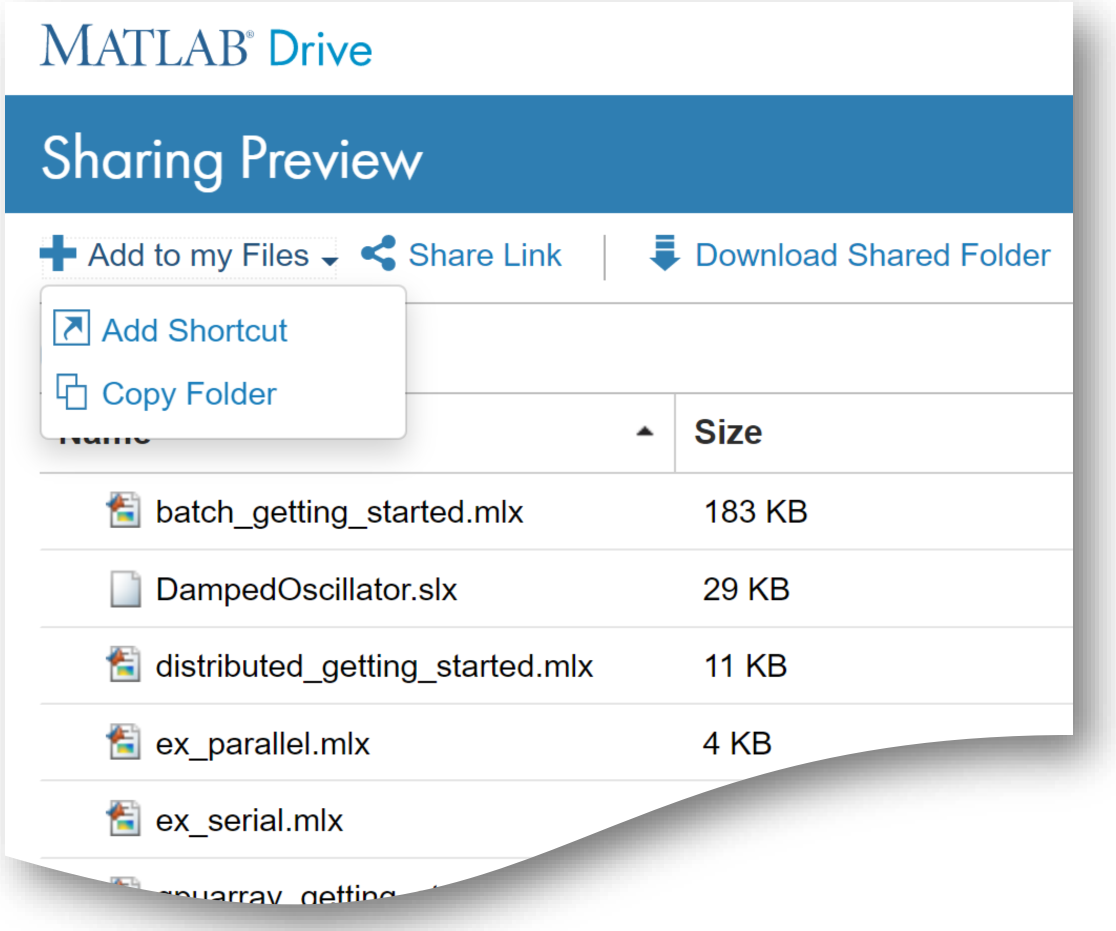


mathworks.com/mwaccount/register

Setup: Step 2 – Copy materials via MATLAB Drive

Click **Add to my Files** and select **Copy Folder**.

*For use on your MATLAB Desktop, click **Download Shared Folder** instead.*



https://drive.matlab.com/sharing/8281b8fd-b766-4bbb-b3aa-2da63bcd037e

Setup: Step 3 – Activate the workshop license and launch MATLAB Online

Access MATLAB for your Parallel Computing Workshop

MathWorks is pleased to provide a special license to you as a course participant to use for your Parallel Computing Workshop. This is a limited license for the duration of your course and is intended to be used only for course work and not for government, research, commercial, or other organization use.

Course Name:	DE-MUC_2023-10-24_Parallel_Computing
Organization:	MathWorks Parallel Computing
Starting:	24 Oct 2023
Ending:	25 Oct 2023

Access MATLAB Online

mathworks.com/licensecenter/classroom/4200902/

Common problems when you try to convert `for`-loops to `parfor`-loops

! **Noninteger loop variables**

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

! **Nested parallel loops**

! **Dependent loop body**

Use Code Analyzer to fix problems when converting `for`-loops to `parfor`-loops

! `parfor`-loop iterations have no guaranteed order and one loop iteration cannot depend on a previous iteration → You might need to modify your code to use `parfor`!

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

No warnings found.

```
1 a = zeros(5,1);
2 b = pi;
3 parfor i = 1:5
4     a(i) = a(i-1) + b;
5 end
6 disp(a)
```

Line 3: The PARFOR loop cannot run due to the way variable 'a' is used. Details ▼

Hands-On Exercise: Convert `for`-loops into `parfor`-loops

Convert `for`-Loops Into `parfor`-Loops

In some cases, you must modify the code to convert `for`-loops to `parfor`-loops. These example focus on diagnosing and fixing `parfor`-loop problems

Hint: Review the code analyzer messages and check the documentation for additional help.

Ensure `parfor`-loop variables are consecutive increasing integers

Correct the following `parfor`-loop to make sure the loop variables are consecutive increasing integers.

Hint: You can fix these errors by converting the loop variables into a valid range.

```
parfor x = 0:0.1:1
    disp(x)
end
```

Ensure there are no nested `parfor`-loops

You cannot nest a `parfor`-loop inside another `parfor`-loop. The reason is that the workers in a parallel pool cannot start or

Because parallel processing incurs overhead, you must choose carefully whether you want

loop

Consider parallel overhead* in deciding when to use `parfor`

`parfor` can be useful ☺

- `for`-loops with loop iterations that take long to execute
- `for`-loops with **many** loop iterations that take a short time, e.g., parameter sweep

`parfor` might not be useful ☹

- `for`-loops with loop iterations that take a short time to execute

Check mathworks.com/help/parallel-computing/improve-parfor-performance.html for ways to improve `parfor` performance.

* Parallel overhead: time required for communication, coordination, and data transfer from client to workers and back.

Run code in parallel

Synchronously with `parfor`

- You wait for your loop to complete to obtain your results
- Your MATLAB client is blocked from running any new computations
- You cannot break out of loop early

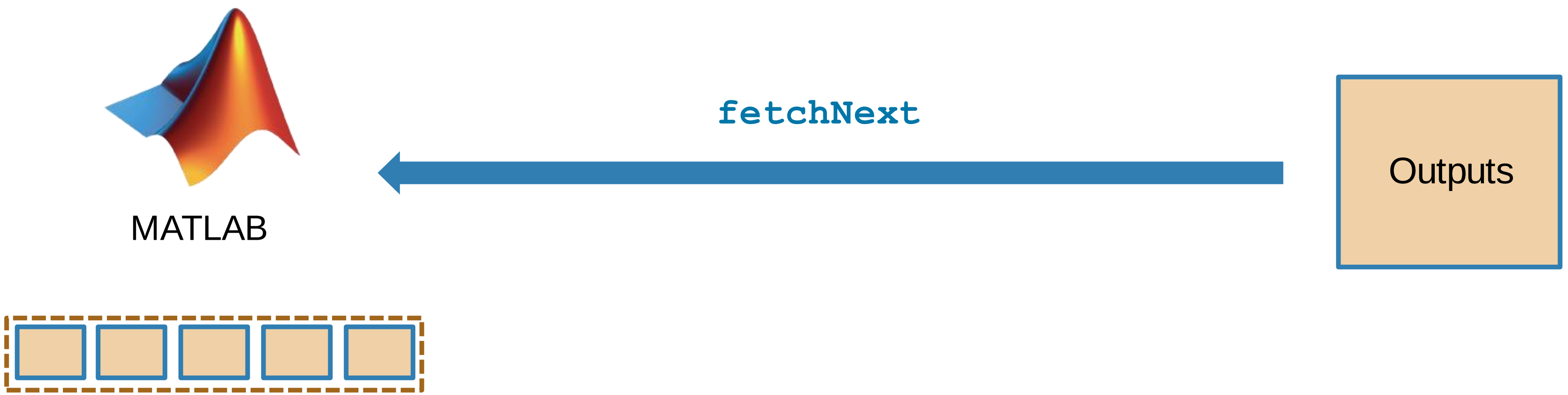
Asynchronously with `parfeval`*

- You can obtain intermediate results
- Your MATLAB client is free to pursue other computations
- You can break out of loop early

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

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

* Runs **function** on parallel workers

Execute functions in parallel asynchronously using `parfeval`



Asynchronous execution on parallel workers

```
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: Use `parfeval` to run functions in the background

Getting Started with `parfeval`

This exercise shows how to use `parfeval` to run functions in the background.

Set up parallel environment

Create a parallel pool `p` with two workers.

```
clear
delete(gcp('nocreate'))
p = parpool(2);
```

```
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 2).
```

Add work to queue

When you use `parfeval` to run functions in the background, this creates an object called *future* for each function and adds the object to the pool queue.

Let's use `parfeval` by instructing workers to execute the function `pause(1)` twice, thus simulating two
each take one second to compute.

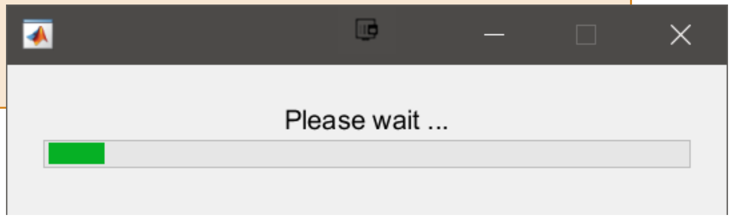
- The inputs to `parfeval` are the function handle (@) to th

Execute additional code as `parfor/parfeval` 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
```



Hands-On Exercise: Use `parfeval` to run functions in the background

Execute additional code as `parfeval` iterations complete

When you offload computations to workers using `parfeval`, you can use `afterEach` and `afterAll` to automatically invoke functions on each or all of the results of `parfeval` computations.

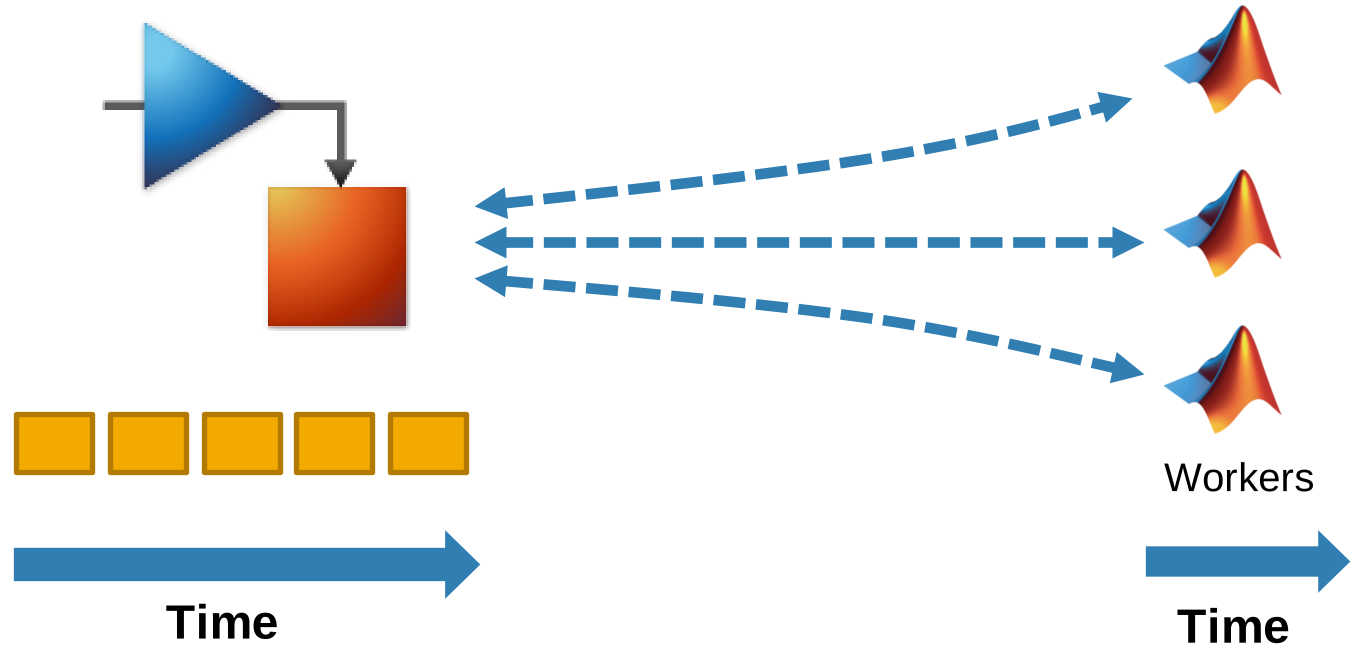
Call `afterEach` on `parfeval` computations

Let's use `parfeval` to compute random vectors in the workers, and `afterEach` to display the largest element in each of those vectors after they are created. `afterEach` executes the function handle on the output of each future when they become ready.

```
p = gcp;
numOutputs = 1;
input1 = 1e6;
input2 = 1;
f(10) = parallel.FevalFuture;
for idx = 1:10
     f(idx) = parfeval(@rand, numOutputs, input1, input2);
end
```

Use `afterEach` to compute the largest elements of these vectors as th

Run multiple 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);
```

Scaling MATLAB applications and Simulink simulations

Ease of Use

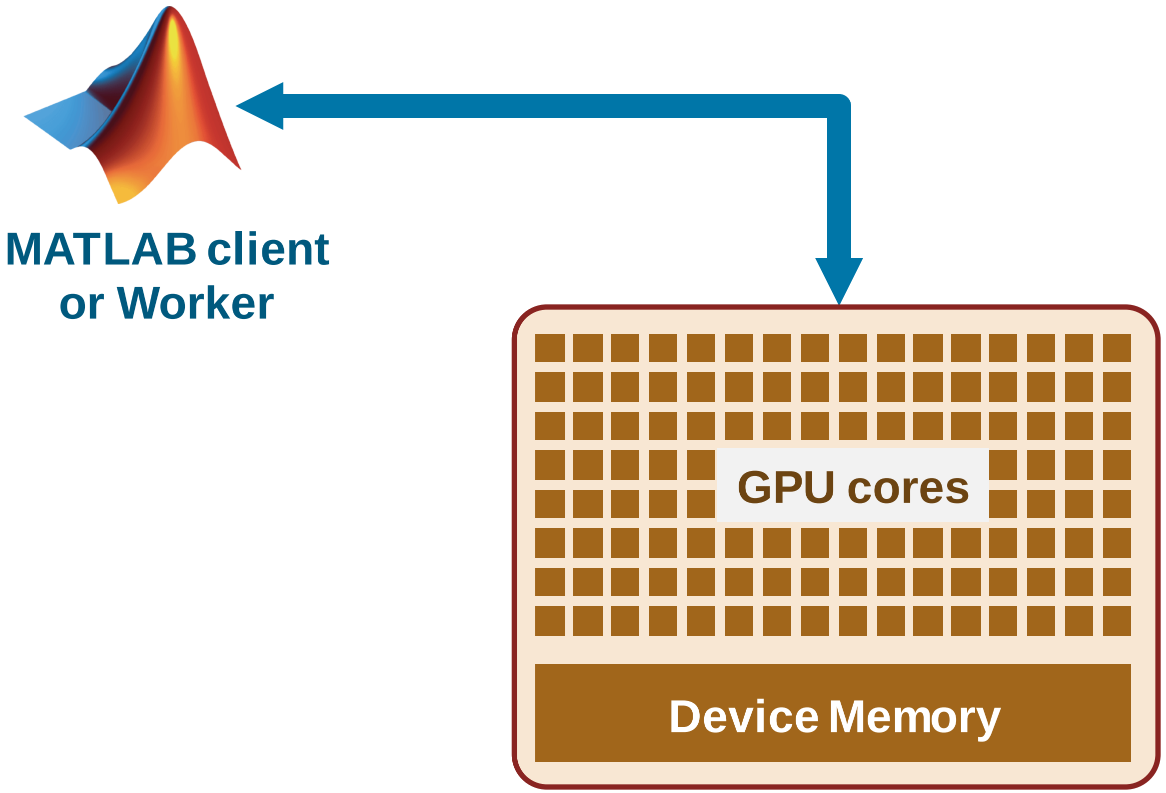
Automatic parallel support in toolboxes

Common programming constructs

Advanced programming constructs
(`spmd`, `etc.`)

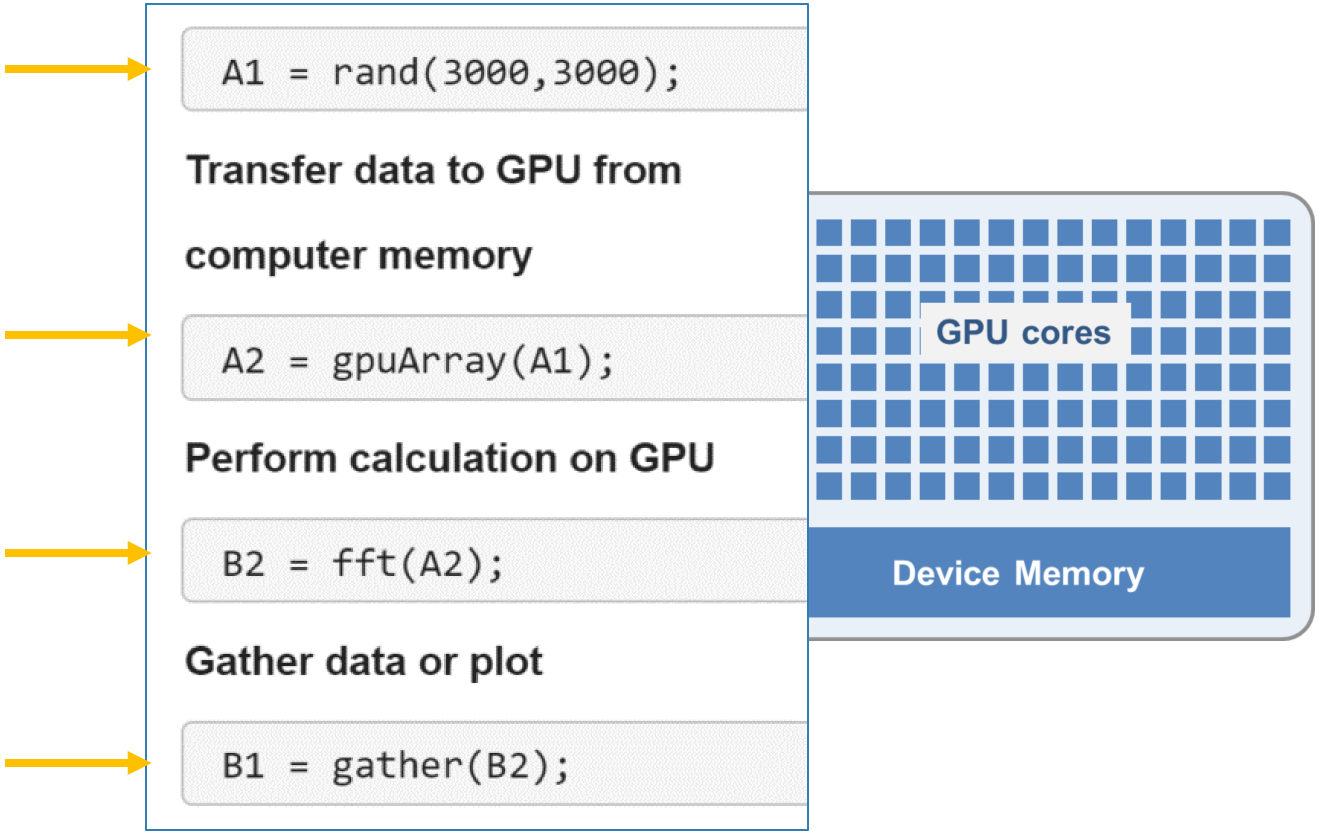
Greater Control

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
- 500+ GPU-supported functions
- Use `gpuArray` and `gather` to transfer data between CPU and GPU



Hands-On Exercise: Offload computations to your GPU

Getting started with GPU Computing in MATLAB

This exercise will demonstrate how to speed up computations using your computer's GPU.

We'll start with an algorithm initially written to run on the CPU. If all the functions that you want to use are supported on the GPU, you can simply use `gpuArray` to transfer input data to the GPU, and call `gather` to retrieve the output data from the GPU. With some minor changes to the code, we'll be able to offload the computation to the GPU.

Run a computation on the CPU

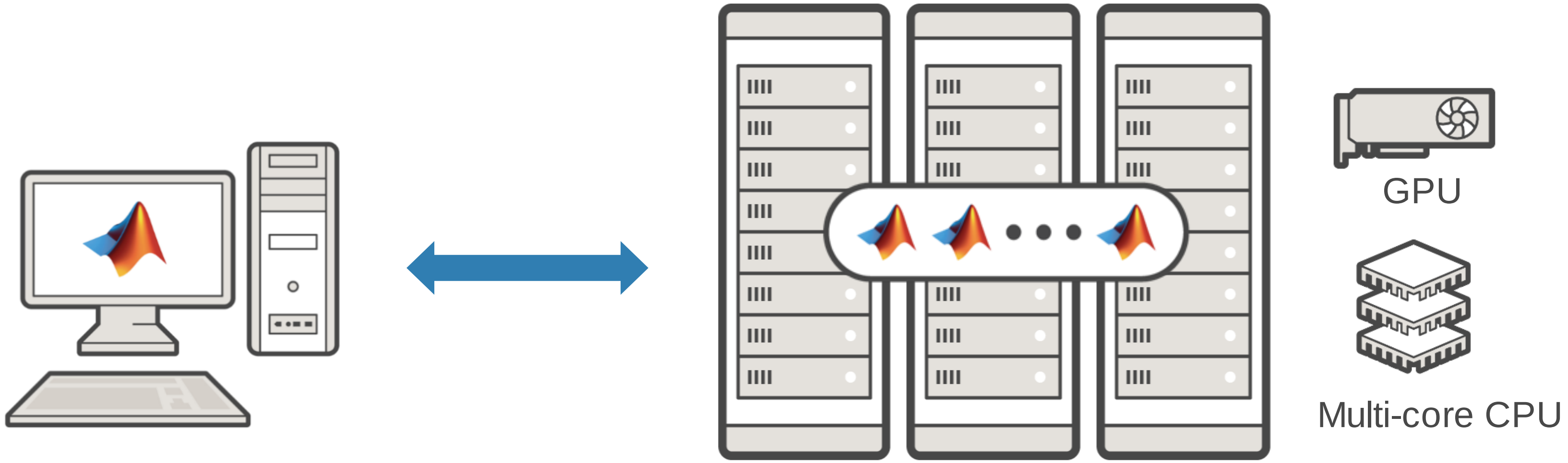
Compute the `fft` on a random matrix:

```
N = 8192;
matrix_cpu = rand(N,N);

tic
out_cpu = fft(matrix_cpu);
time_cpu = toc;

disp(['FFT time on CPU: ' num2str(time_cpu)])
```

```
FFT time on CPU: 0.24305
```

...ur GPU device

Parallel computing on your desktop, clusters, and clouds



MATLAB
Parallel Computing Toolbox

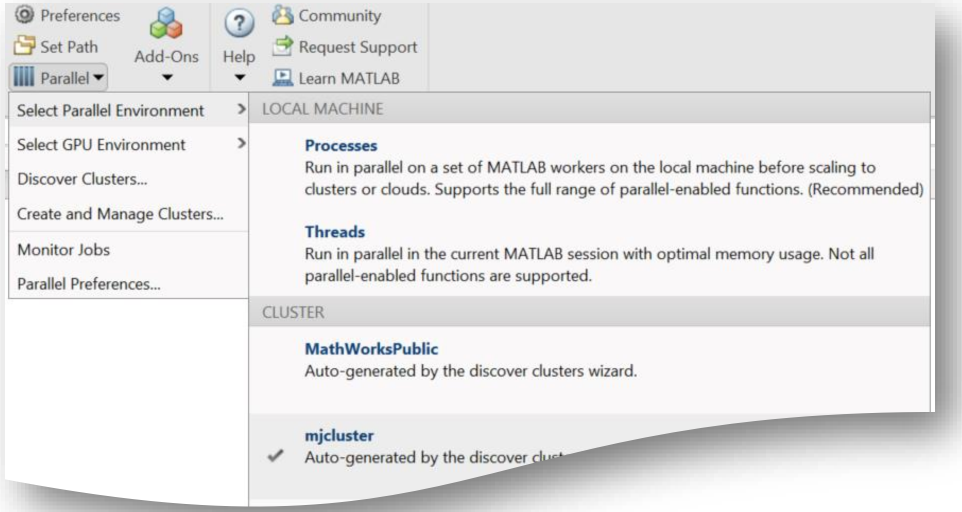
MATLAB Parallel Server

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

Scale to clusters and clouds

With MATLAB Parallel Server, you can…

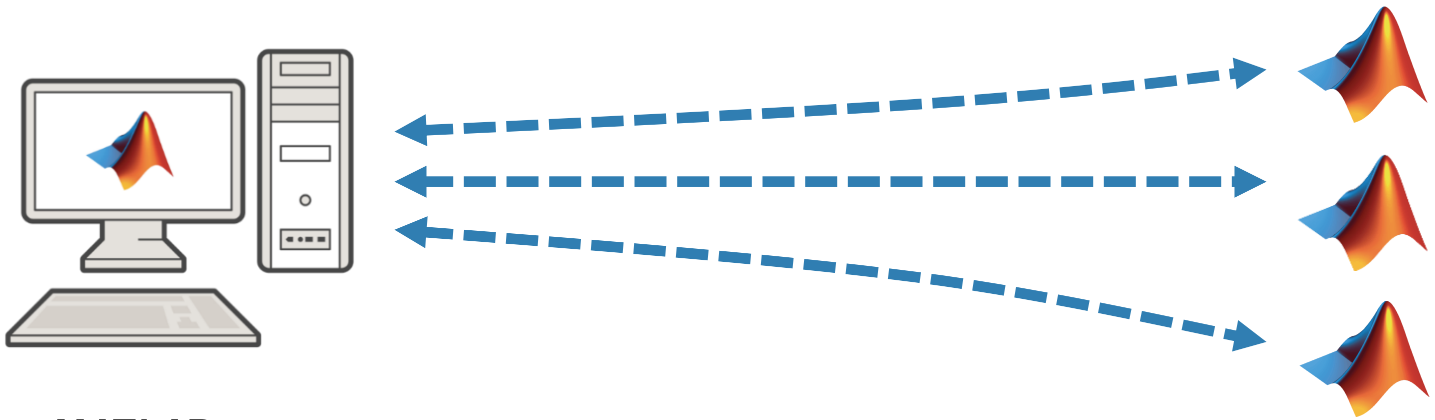
- Change hardware with minimal code change
- Submit to on-premise or cloud clusters
- Support cross-platform submission
  - Windows client to Linux cluster



Interactive parallel computing

Leverage cluster resources in MATLAB

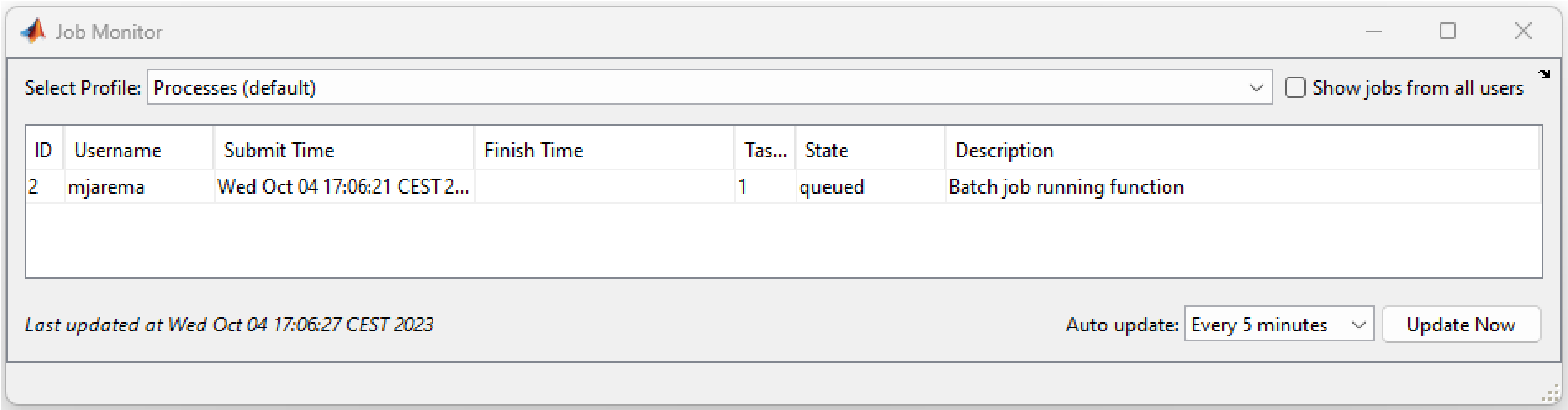
```
>> parpool('cluster', 3);
>> myscript
```



MATLAB
Parallel Computing Toolbox

myscript.m:

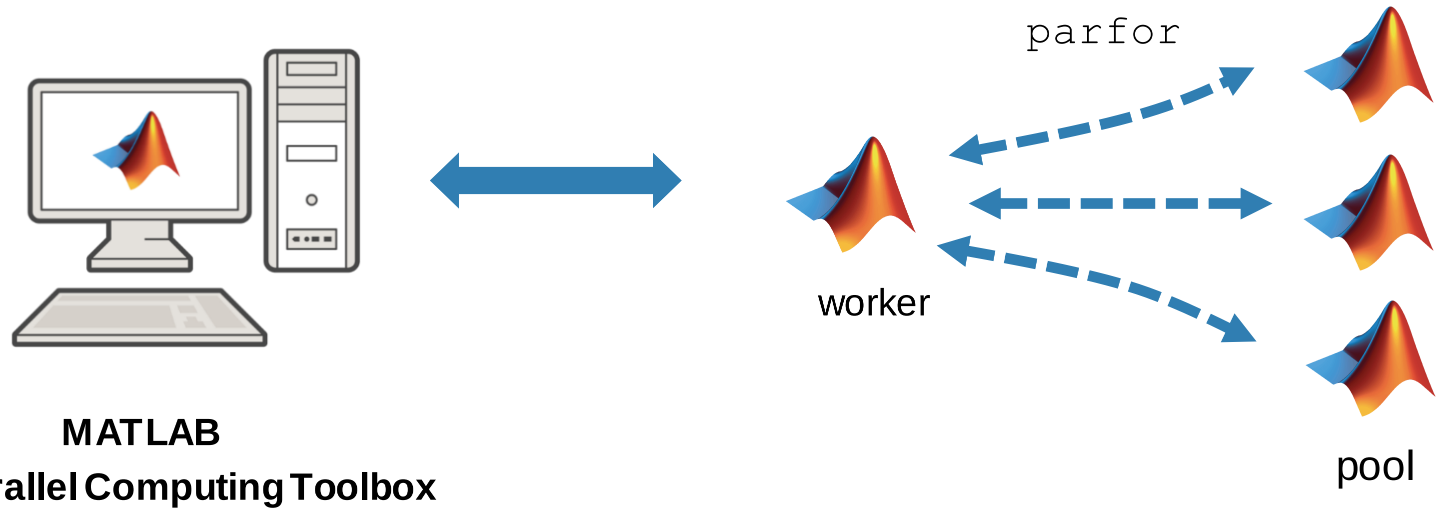
```
a = zeros(5, 1);
b = pi;
parfor i = 1:5
   a(i) = i + b;
end
```



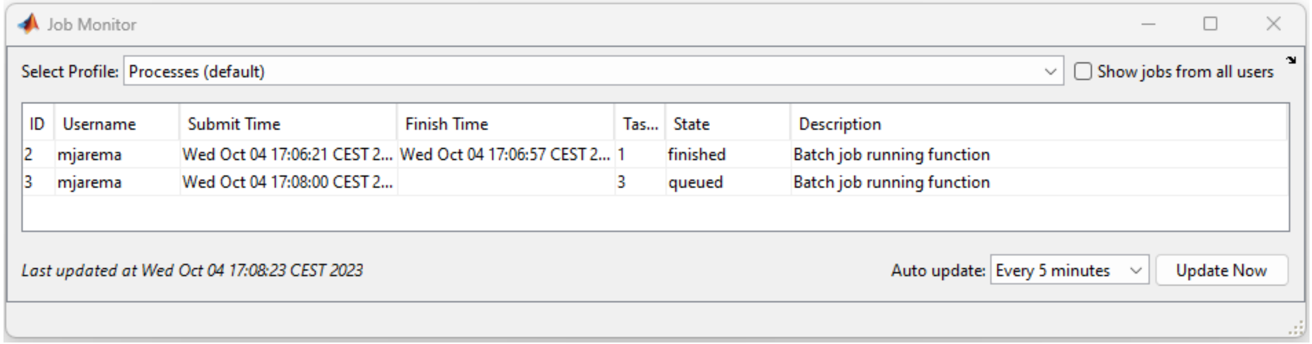
`batch` simplifies offloading computations

Submit MATLAB jobs to the cluster

```
>> job = batch('myscript','Pool',3);
```




```
>> j.State
ans =
    'running'
>> j.diary
Warning: The diary of this batch job might be incomplete
because the job is still running.
--- Start Diary ---

Analyzed 1 image.
Analyzed 2 images.
Analyzed 3 images.
Analyzed 4 images.

--- End Diary ---
```



Hands-On Exercise: Use `batch` to offload serial and parallel computations

Getting Started with `batch`

We can use `batch` jobs to offload the execution of long-running computations in the background and carry out other tasks while the batch job is processing. `batch` does not block MATLAB and we can continue working while computations take place. When we submit batch jobs to another computer or cluster, we can even close MATLAB on the client, and retrieve results later.

In this exercise, we will submit batch jobs from MATLAB to our local machine. The workers will run on the same machine as the client, but the same workflow can be used to submit jobs to a remote compute cluster or the cloud, freeing up our local resources.

Run a `batch` job to offload a serial computation

`batch` runs our code on a local worker or a cluster worker, but does not require a parallel pool. Close a parallel pool, if one is open.

```
delete(gcp('nocreate'))
```

Use the batch command to offload the function `ex_serial` to one MATLAB worker session that runs in the background, while we can continue using MATLAB as computations take place.

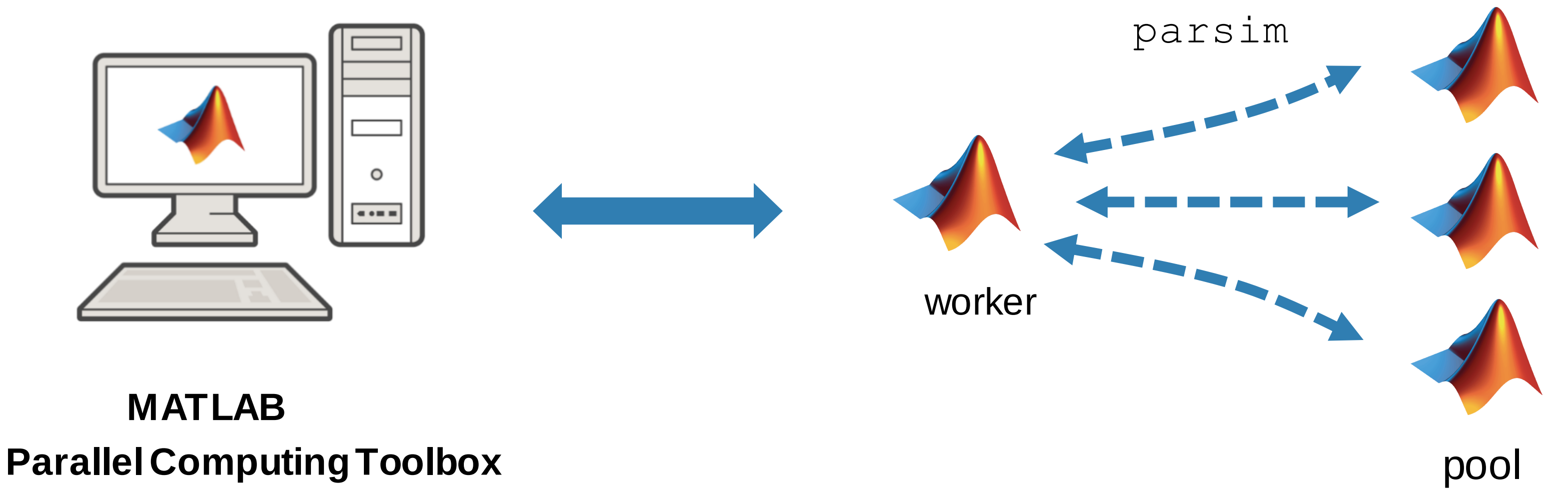
The function performs `N` trials of computing the largest eigenvalue for an `M-by-M` random matrix and

```
numOutputs = 1;
```

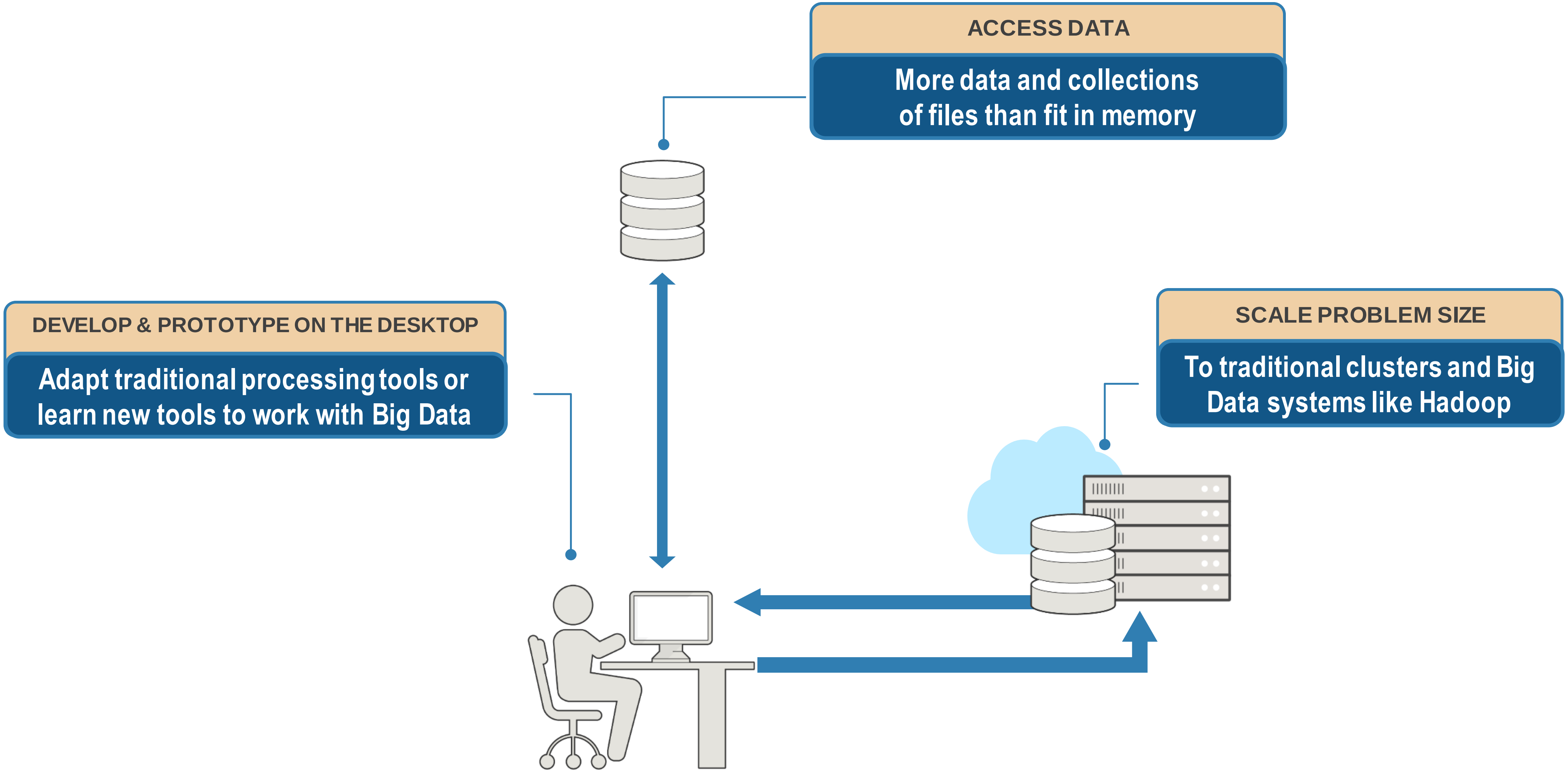
`batch` simplifies offloading simulations

Submit Simulink jobs to the cluster

```
job = batchsim(in,'Pool',3);
```

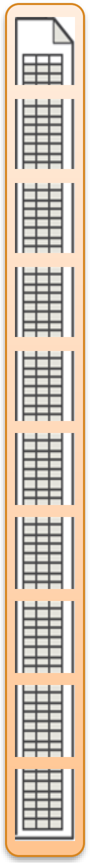


Big Data Workflows



`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.)



Hands-On Exercise: Use `tall` arrays for Big Data

Getting Started with `tall` arrays for Big Data

In this exercise we'll use `tall` arrays to work with large data sets that have more rows than might fit into memory.

We can work with many operations and functions as we would with in-memory MATLAB arrays, but most results are evaluated only when we request them explicitly using `gather`, write a tall array to disk, or visualize the tall array. MATLAB automatically optimizes the queued calculations by minimizing the number of passes through the data.

Access Data: Create Datastore from Sample File(s)

The comma-separated text file `airlinesmall.csv` contains departure and arrival information about individual airline flights from the years 1987 through 2008, stored in a tabular manner.

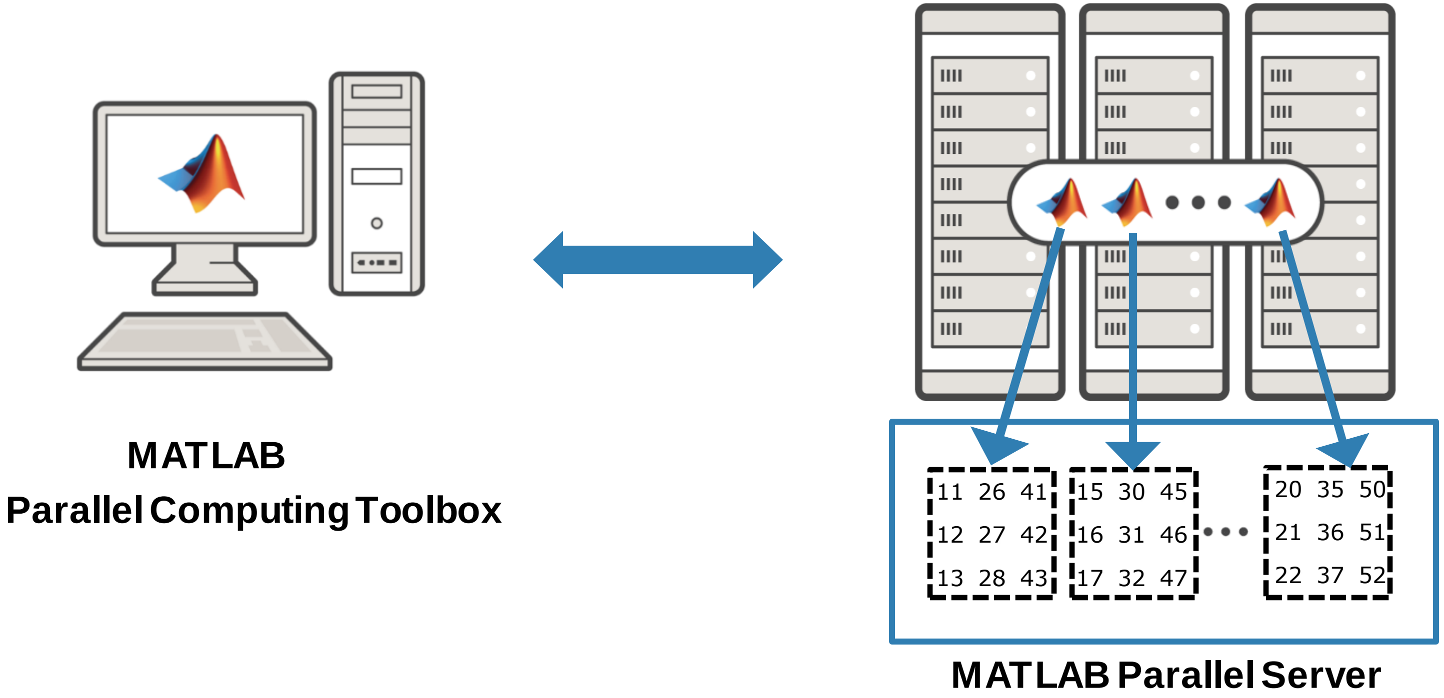
Tabular data that contains a mix of numeric and text data, as well as variable and row names, is best represented in MATLAB as a **table** (a container that stores column-oriented data in variables), because table variables can have different data types and sizes as long as all variables have the same number of rows.

If a file can be imported and processed in its entirety on our computer, we could also i

programmatically, by reading the comma-separated text fil

distributed arrays

- Distribute large matrices across workers running on a cluster
- Support includes matrix manipulation, linear algebra, and signal processing
- Several hundred MATLAB functions overloaded for distributed arrays



Working with distributed arrays

Further Resources

- MATLAB Documentation
  - MATLAB → Software Development Tools → Performance and Memory
  - Parallel Computing Toolbox
- Parallel and GPU Computing Tutorials
  - https://www.mathworks.com/videos/series/parallel-and-gpu-computing-tutorials-97719.html
- Parallel Computing with MATLAB
  - https://www.mathworks.com/solutions/parallel-computing.html

Choose your solution to accelerate your code

Top 5 **MATLAB** Acceleration Techniques

1. Adopt Efficient (Serial) Programming Practices
2. Leverage Existing Optimized Algorithms
3. Use Parallel Computing including GPUs
4. Use Parallel Computing
5. Generate C Code from MATLAB Code
6. All of the Above