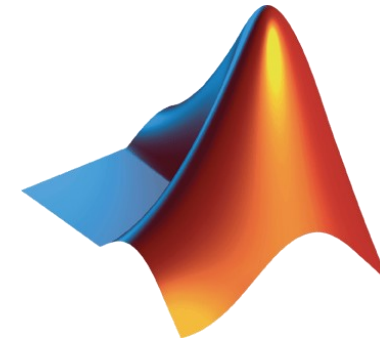


Workshop: Parallel Computing With MATLAB (Part I)

UNIVERSITÄT
DUISBURG
ESSEN

Raymond Norris
Application Engineer, MathWorks
May 2021



Agenda

- Part I – Parallel Computing with MATLAB on the Desktop
 - Parallel Computing Toolbox
 - MATLAB Online
- Part II – Scaling MATLAB to magnitUDE
 - MATLAB Parallel Server
 - MATLAB Desktop

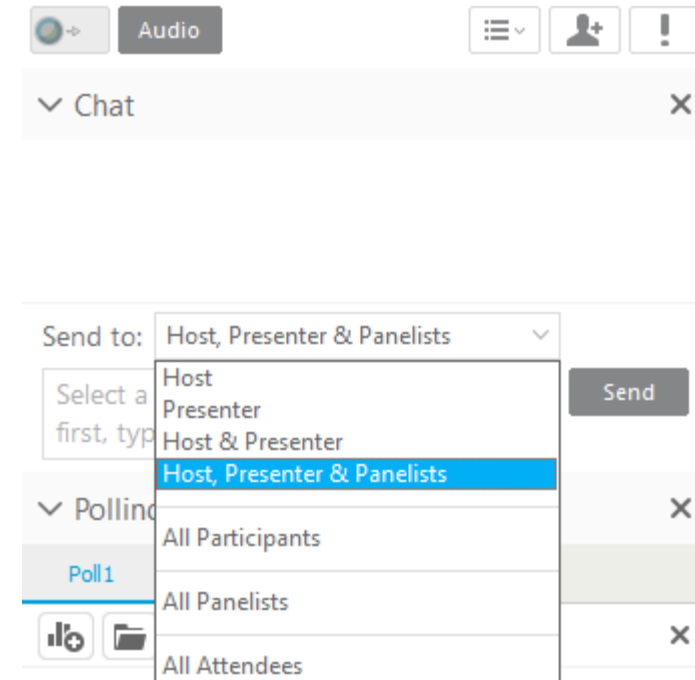
Agenda

- Part I – Parallel Computing with MATLAB on the Desktop
 - Parallel Computing Toolbox
 - MATLAB Online
- Part II – Scaling MATLAB to magnitUDE
 - MATLAB Parallel Server
 - MATLAB Desktop

Chatting

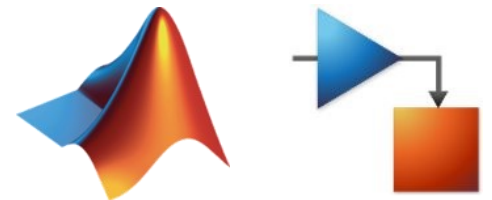
- Send to at least the *Host, Presenter & Panelists*
- Ideally, send to *All Attendees*

Participants



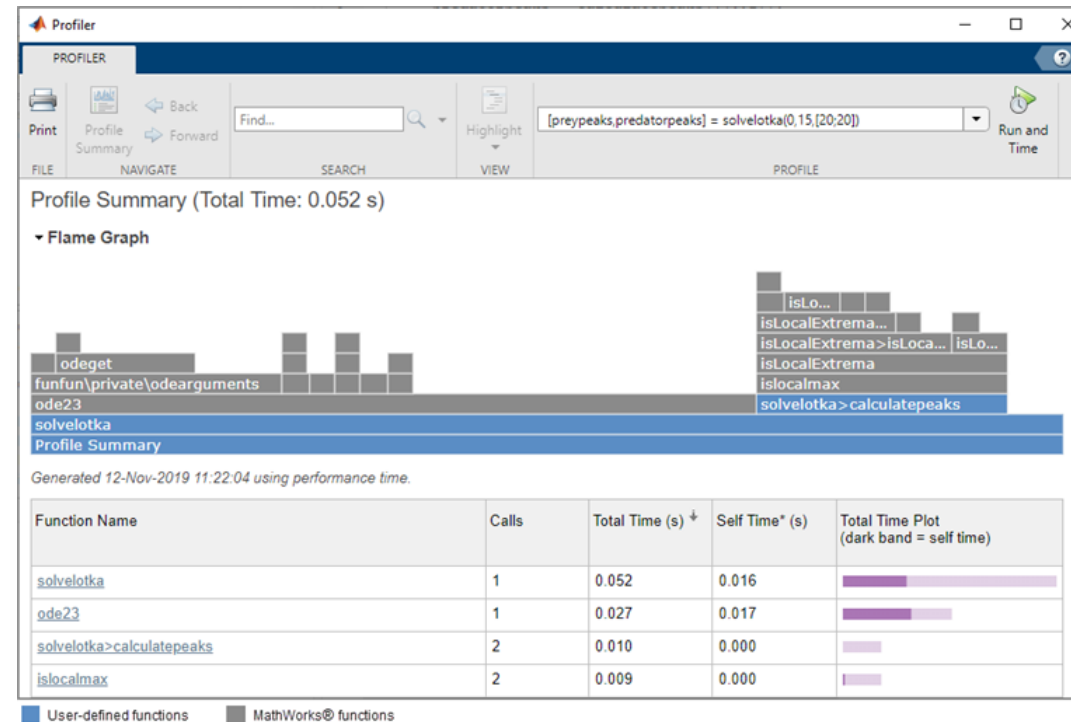
Save time and tackle increasingly complex problems

- Reduce computation time by using available compute cores and GPUs
- Scale and accelerate workflows with minimal code changes
- Scale computations to clusters and clouds
- Focus on your engineering and research, not the computation



Optimize your code before parallelizing for best performance

- Find bottlenecks with profiler



Optimize your code before parallelizing for best performance

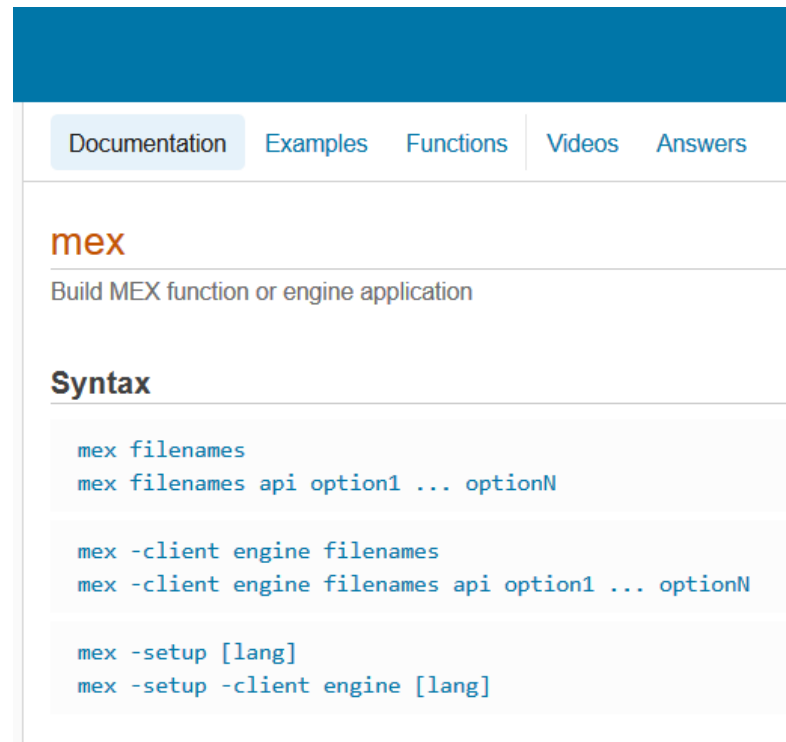
- Implement effective programming techniques

```
59 | % Process each image using preprocessImage
60 | for imgInd = 1:numel(imds.Files)
61 |     fprintf('Processing image %i', imgInd)
62 |     inImageFile = imds.Files{imgInd};
63 |     t(imgInd) = imgDep + imgInd;
64 |     % Output has the same sub-directory structure as input
65 |     % outImageFileWithExtension = strrep(inImageFile, inDir, outDir);
66 |     [~,name,ext] = fileparts(inImageFile);
67 |     outImageFileWithExtension = fullfile(tempdir, [name ext]);
68 |     % Remove the file extension to create the template output file name
69 |     [path, filename,~] = fileparts(outImageFileWithExtension);
70 |     outImageFile = fullfile(path,filename);
71 |
```

⚠ Line 63: The variable 't' appears to change size on every loop iteration. Consider preallocating for speed. [Details](#)

Optimize your code before parallelizing for best performance

- (Advanced) Replace code with MEX functions



The screenshot shows the MATLAB documentation page for the `mex` command. The page has a blue header bar and a navigation menu with tabs for Documentation, Examples, Functions, Videos, and Answers. The main content area is titled `mex` and describes it as a command to "Build MEX function or engine application". Below this, there is a "Syntax" section with three code blocks showing the command syntax for building MEX functions, using a client engine, and setting up the environment.

```
Documentation Examples Functions Videos Answers
```

mex

Build MEX function or engine application

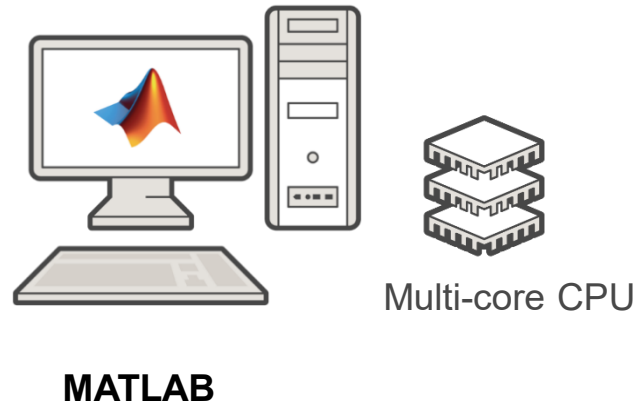
Syntax

```
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]
```


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

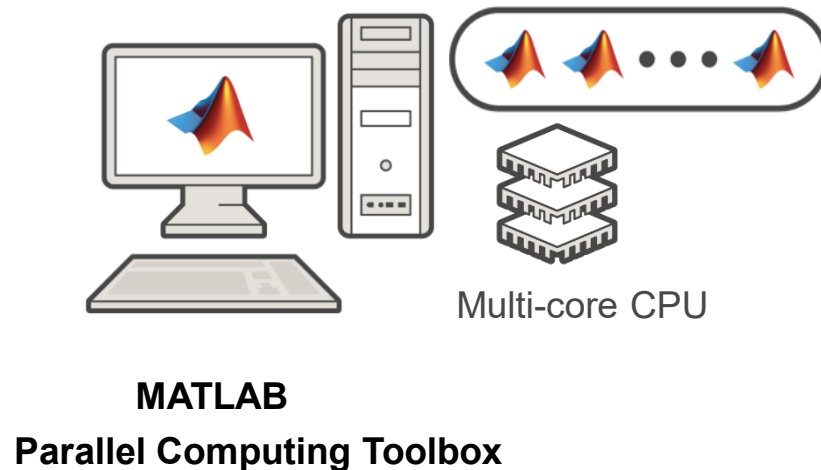
Linear algebra and numerical functions such as `fft`, `\(mldivide)`, `eig`, `svd`, and `sort` are multithreaded in MATLAB. Multithreaded computations have been on by default in MATLAB since Release 2008a. These functions automatically execute on multiple computational threads in a single MATLAB session,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

Scale further with parallel computing



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.

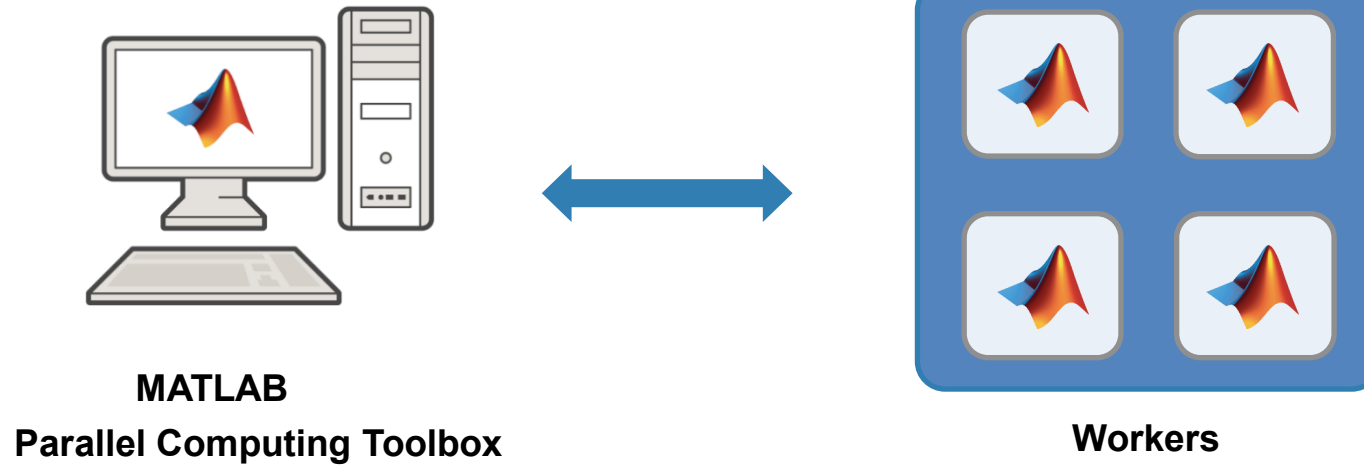
Built-in Multithreading

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.

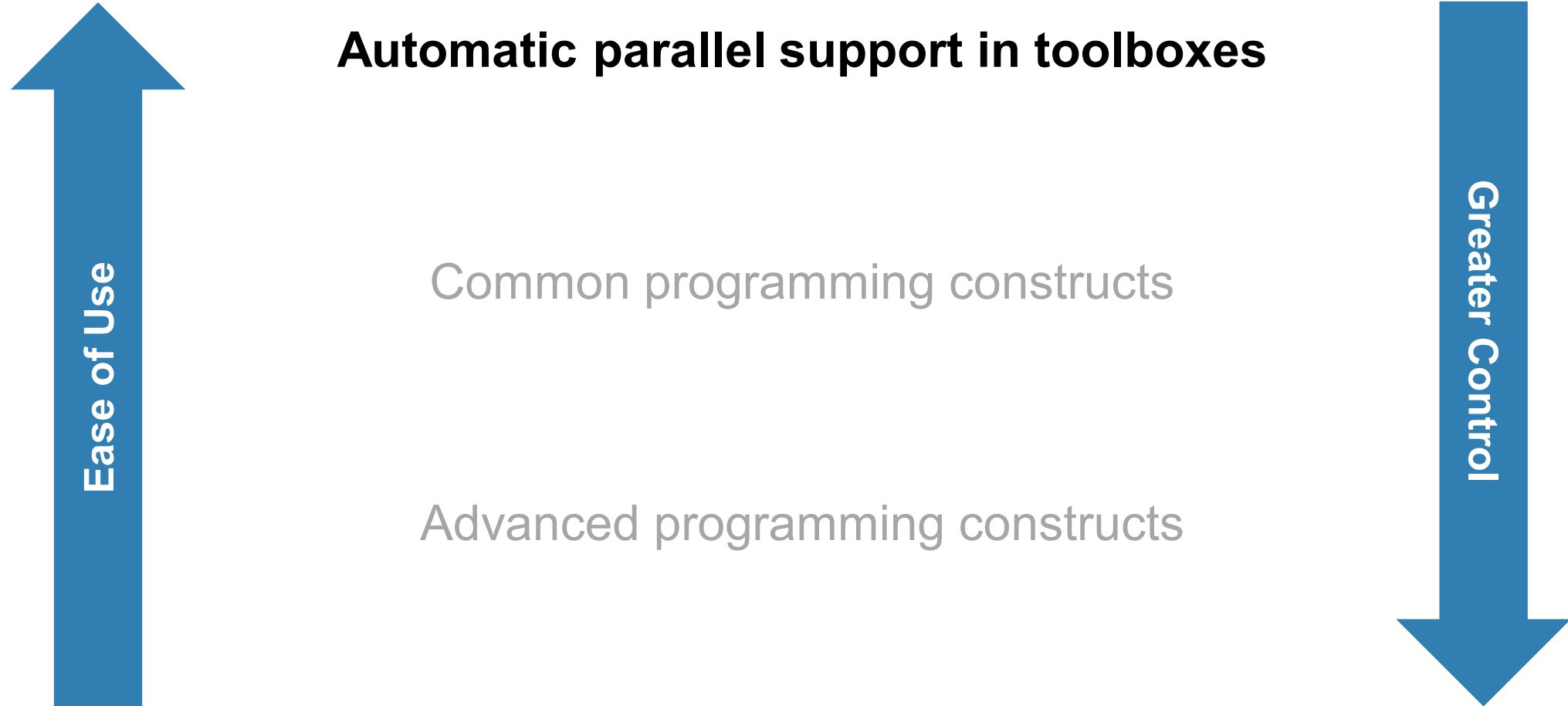
Parallelism Using MATLAB Workers

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.

Run multiple iterations by utilizing multiple CPU cores



Scaling MATLAB applications and Simulink simulations



Scaling MATLAB applications and Simulink simulations



Automatic parallel support in toolboxes

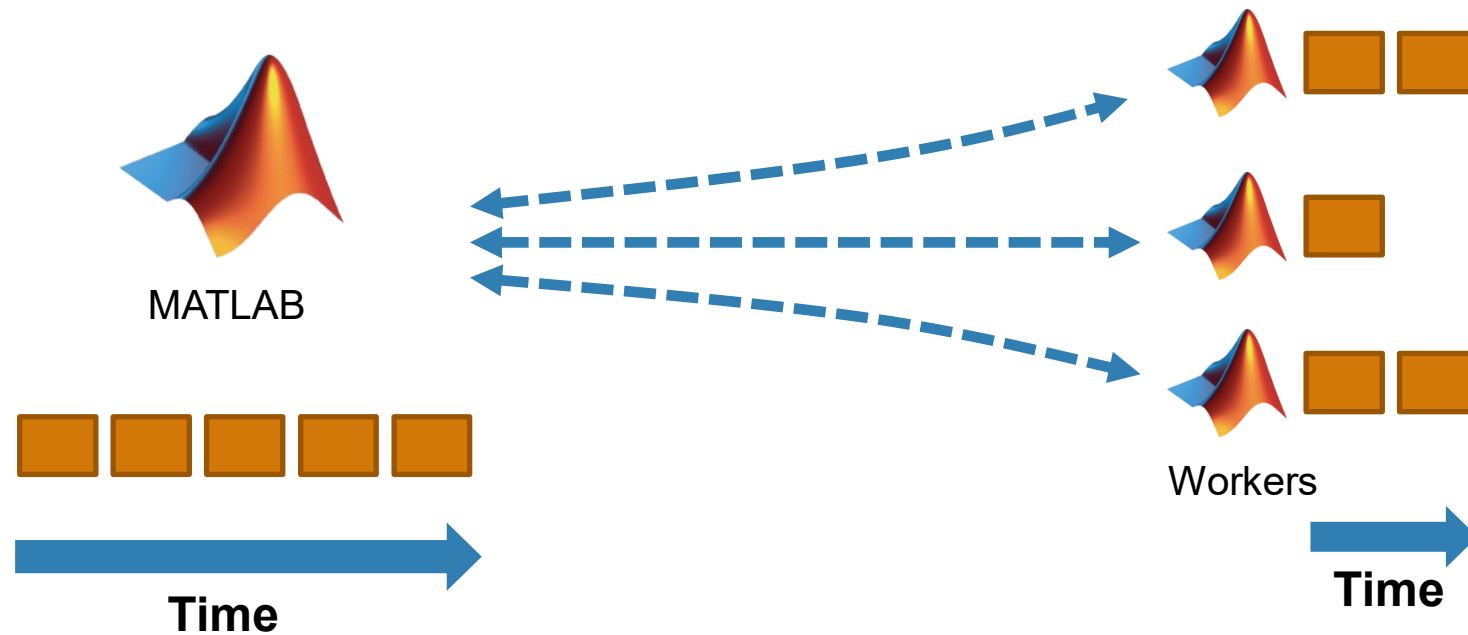
Common programming constructs
(`parfor`, `parfeval`, `parsim`, ...)

Advanced programming constructs



Parallelism using `parfor`

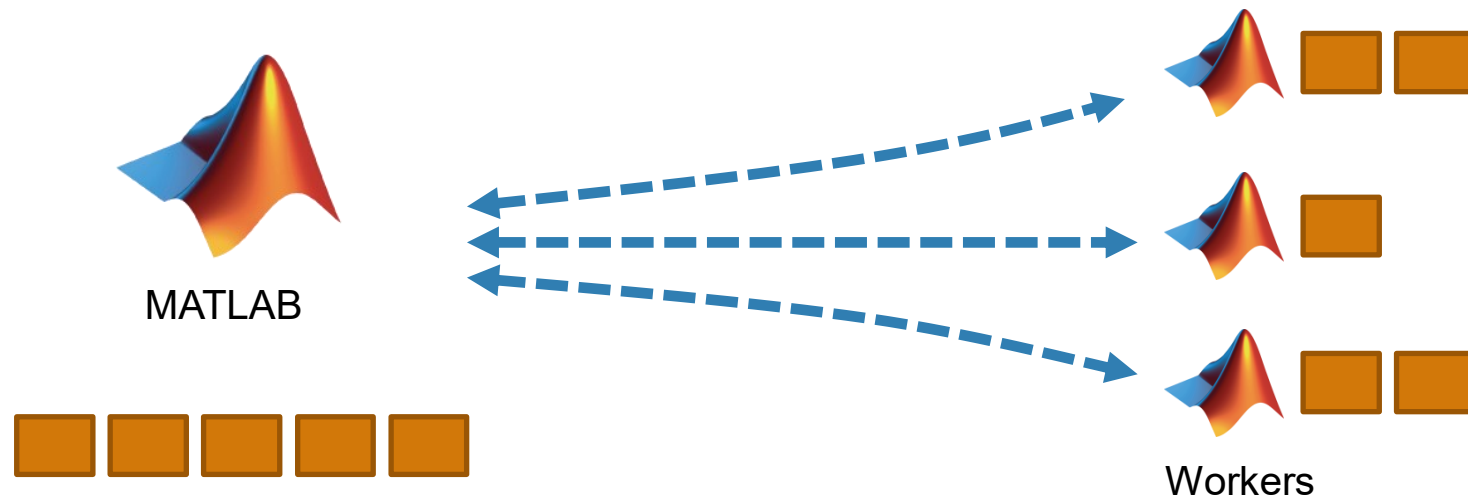
- Run iterations in parallel
- Examples: parameter sweeps, Monte Carlo simulations



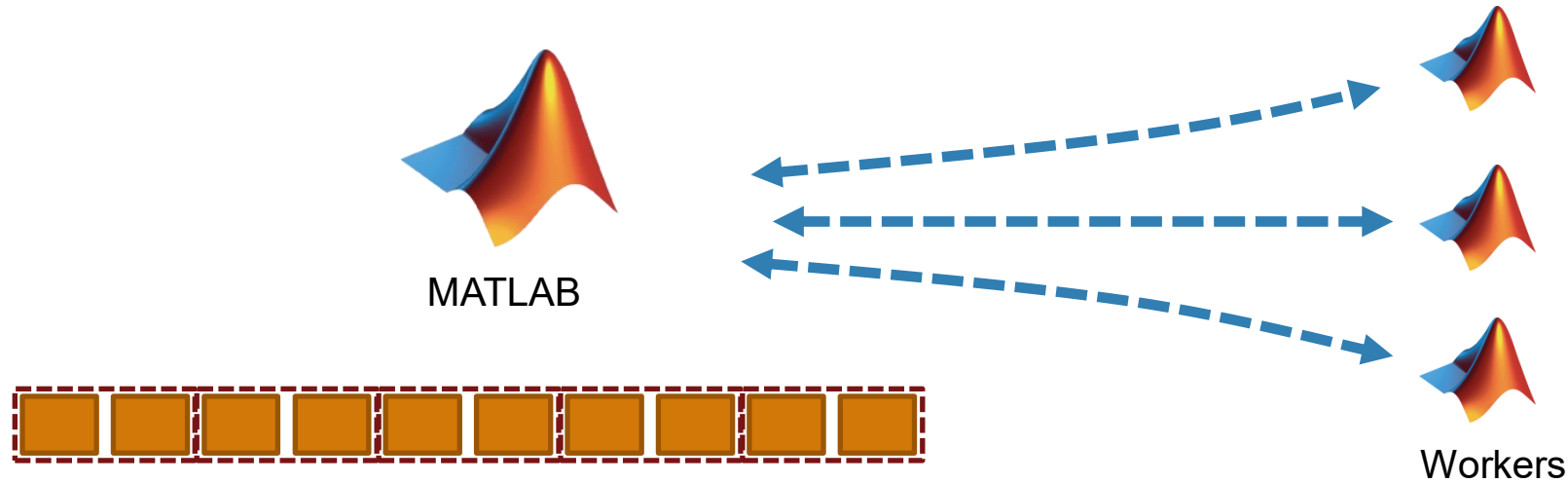
Parallelism using `parfor`

```
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)
```

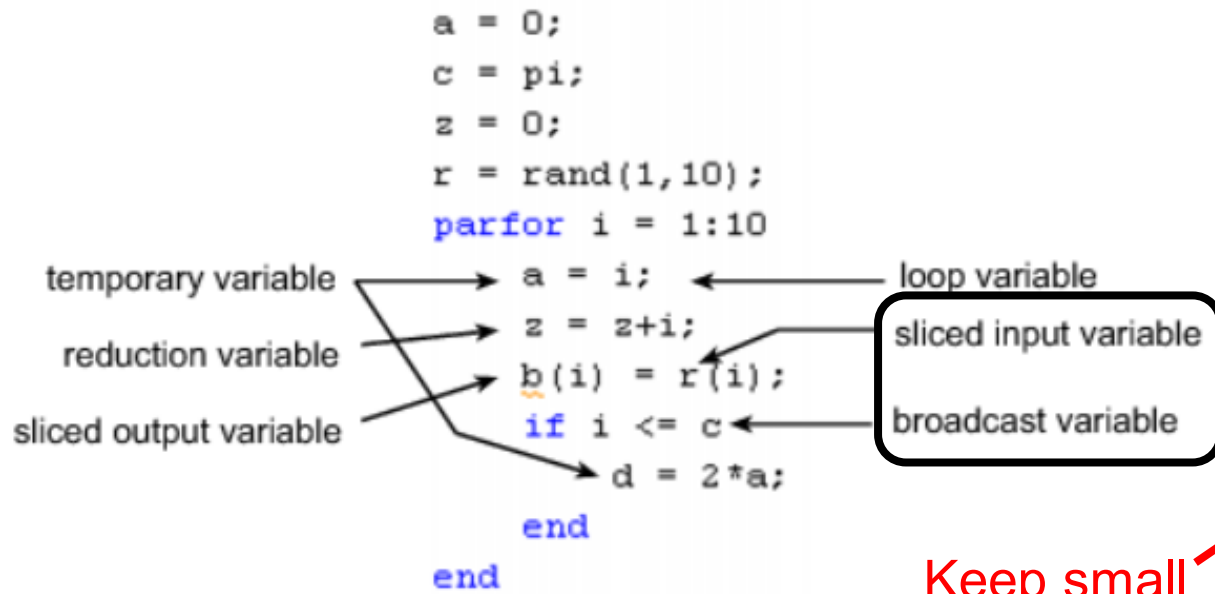


Parallelize for loops with independent iterations



```
a = zeros(10, 1);  
b = pi;  
parfor i = 1:10  
    a(i) = i + b;  
end  
disp(a)
```


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

Parallelism using `parfor`

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

No warnings found.
(Using Default Settings)

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

Line 4: In a PARFOR loop, variable 'a' is indexed in different ways, potentially causing dependencies between iterations.

Execute additional code as iterations complete

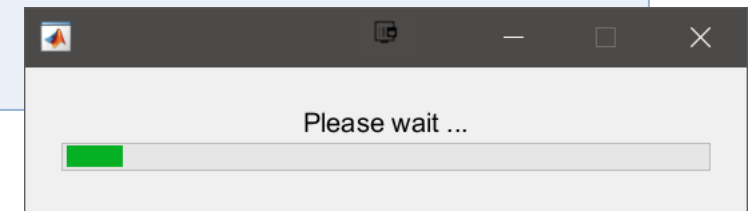
- Send data or messages from parallel workers back to the MATLAB client
- Retrieve intermediate values and track computation progress

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

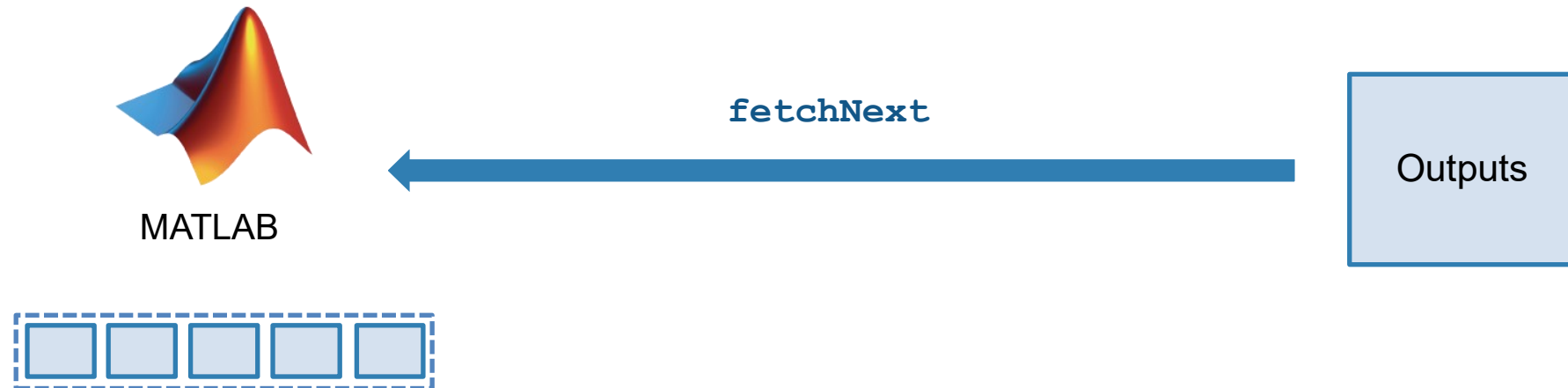
    N = 200;
    p = 1;

    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`

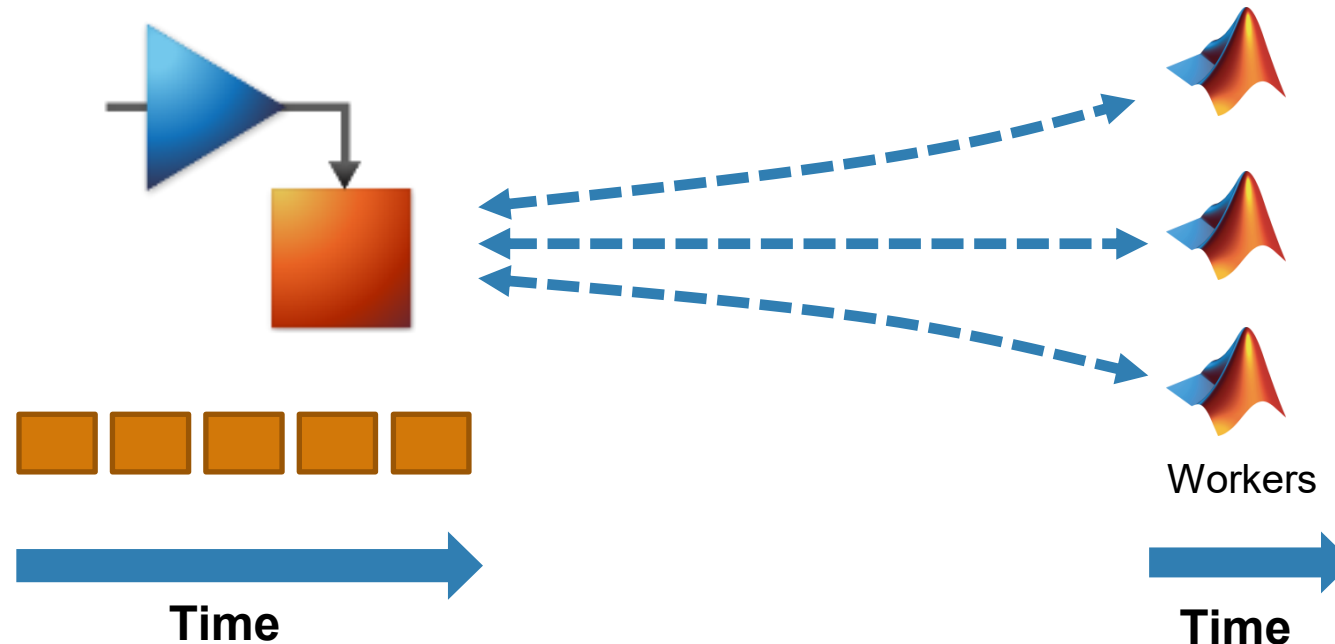


- Asynchronous execution on parallel workers
- Useful for “needle in a haystack” problems

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

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

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



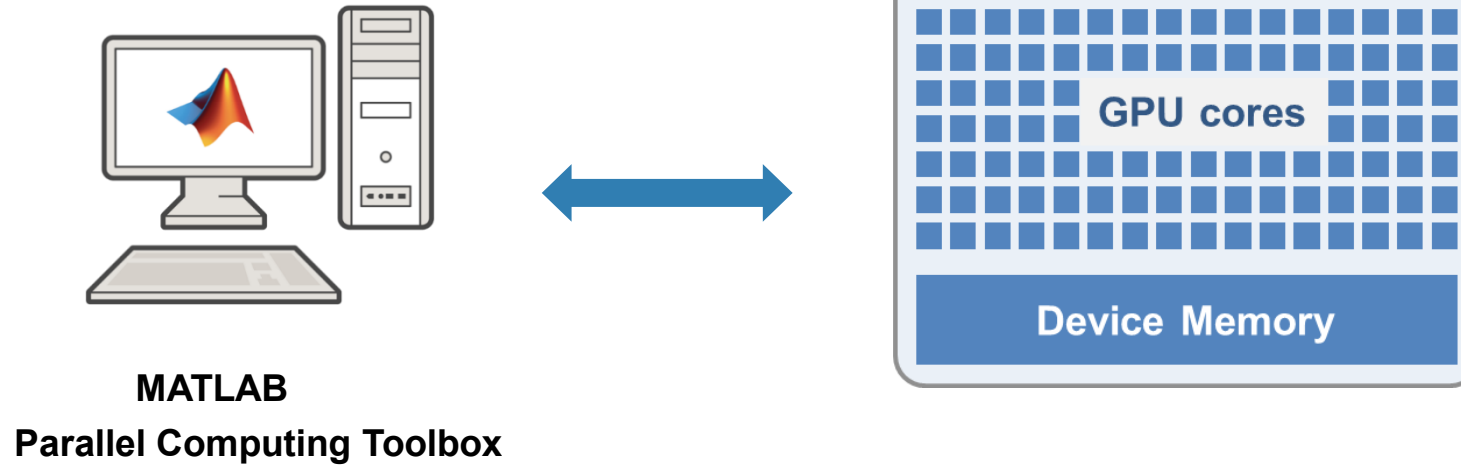
Automatic parallel support in toolboxes

Common programming constructs

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

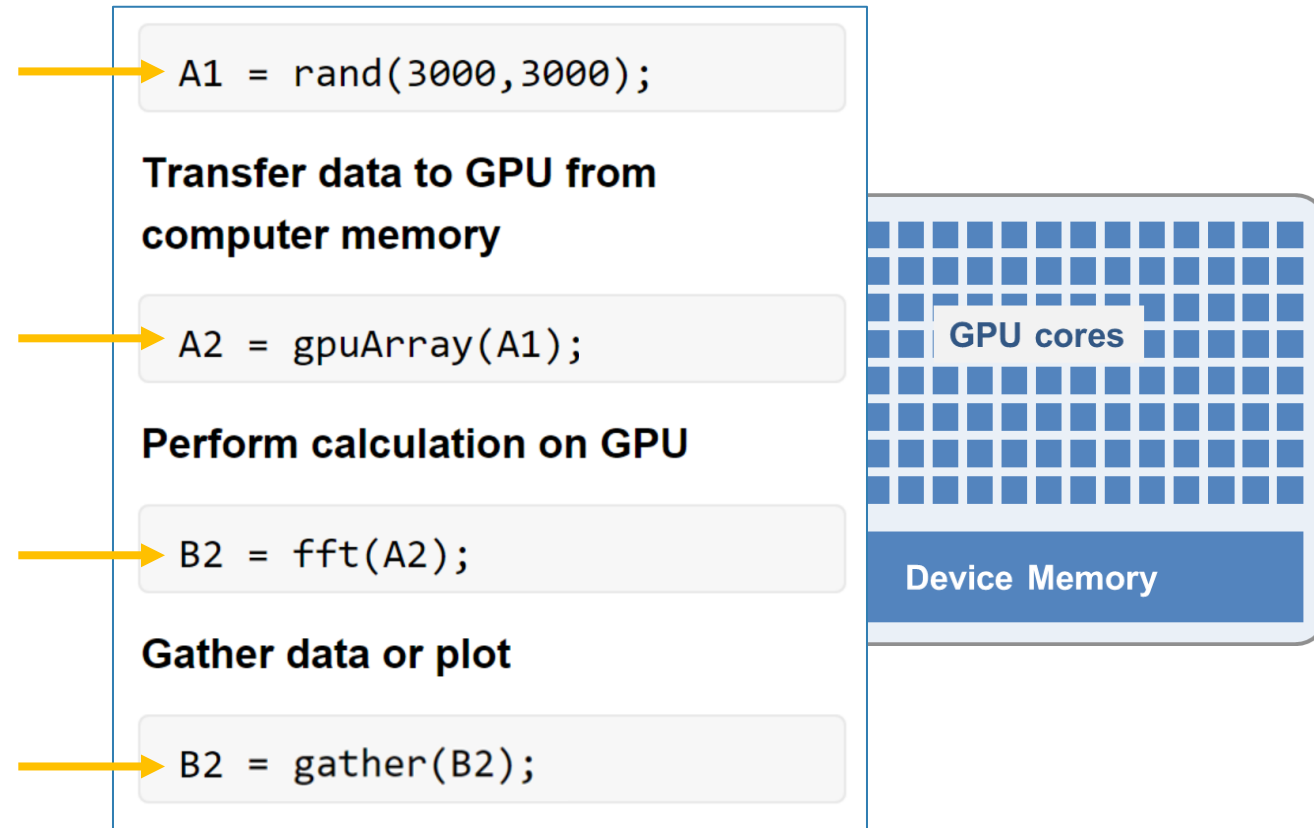


Leverage NVIDIA GPUs without learning CUDA

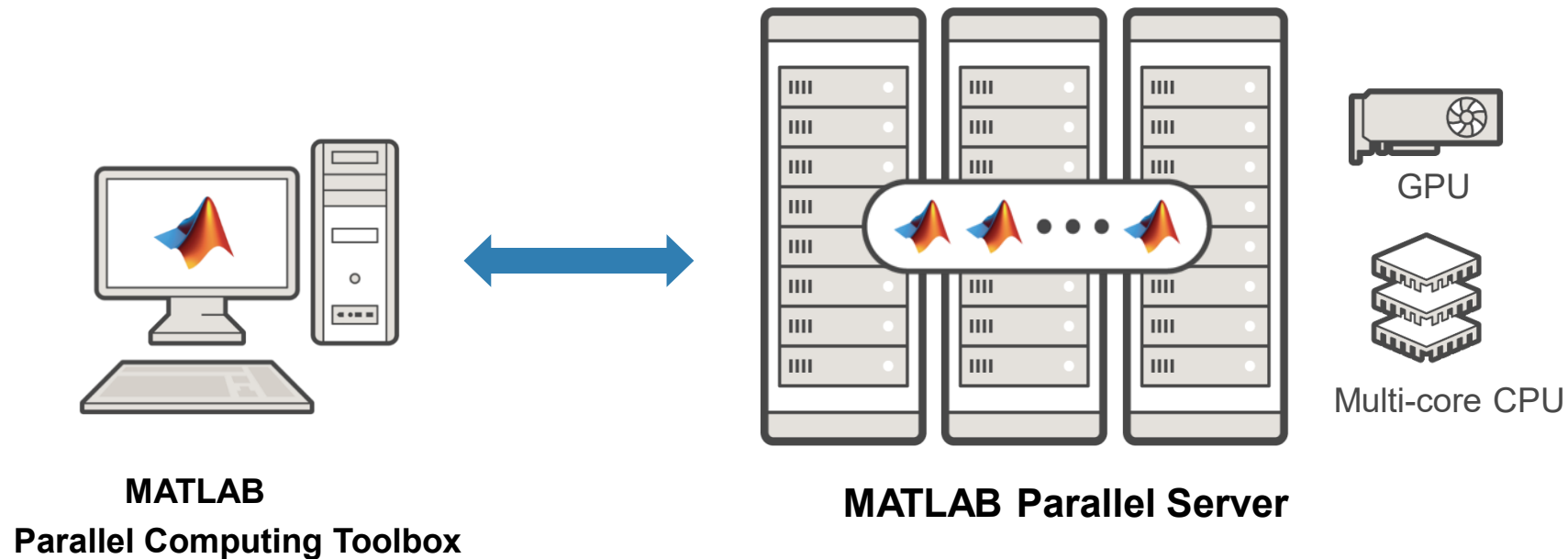


Leverage your GPU to accelerate your MATLAB code

- 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



Parallel computing on your desktop, clusters, and clouds

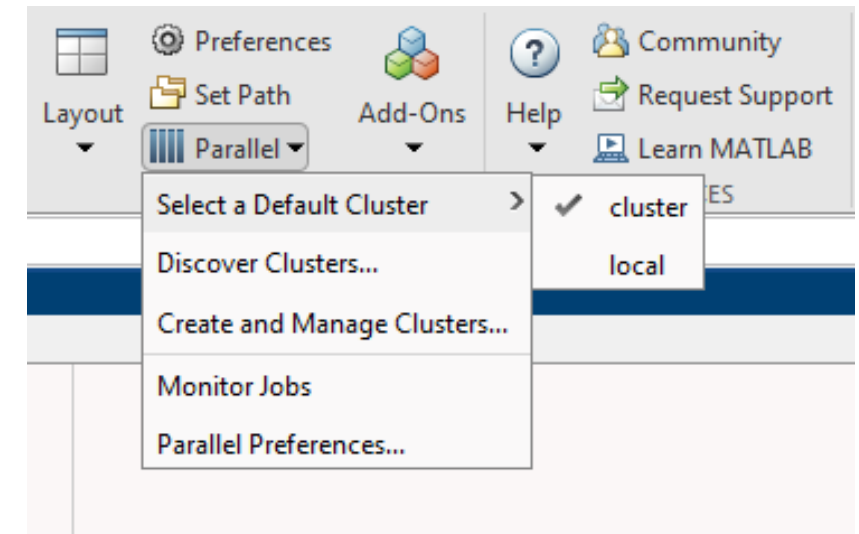


- Prototype on the desktop
- Integrate with infrastructure
- Access directly through MATLAB

Scale to clusters and clouds

With MATLAB Parallel Server, you can...

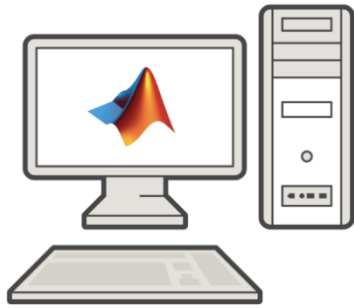
- Change hardware with minimal code change
- Submit to on-premise or cloud clusters
- Support cross-platform submission
 - Windows client to Linux cluster



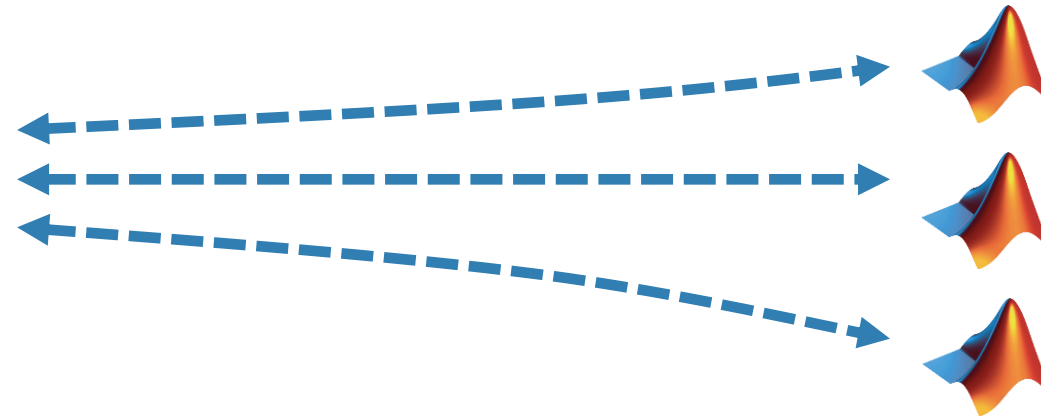
Interactive parallel computing

Leverage cluster resources in MATLAB

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

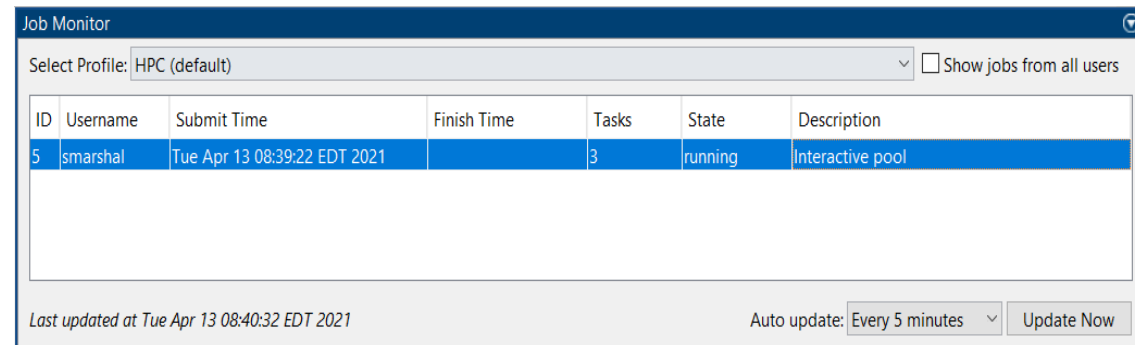


MATLAB
Parallel Computing Toolbox



myscript.m:

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



Job Monitor

Select Profile: HPC (default) Show jobs from all users

ID	Username	Submit Time	Finish Time	Tasks	State	Description
5	smarshal	Tue Apr 13 08:39:22 EDT 2021		3	running	Interactive pool

Last updated at Tue Apr 13 08:40:32 EDT 2021

Auto update: Every 5 minutes

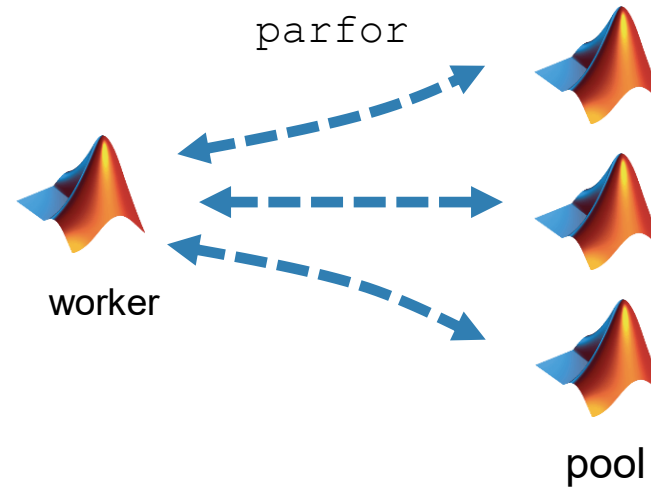
batch simplifies offloading computations

Submit MATLAB jobs to the cluster

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



MATLAB
Parallel Computing Toolbox



```
>> 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 ---
```

ID	Username	Submit Time	Finish Time	Tasks	State	Description
6	smarshal	Tue Apr 13 08:41:53 EDT 2021		4	running	Batch job running script
7	smarshal	Tue Apr 13 08:41:53 EDT 2021		4	running	Batch job running script
8	smarshal	Tue Apr 13 08:41:54 EDT 2021		4	queued	Batch job running script
9	smarshal	Tue Apr 13 08:41:55 EDT 2021		4	queued	Batch job running script

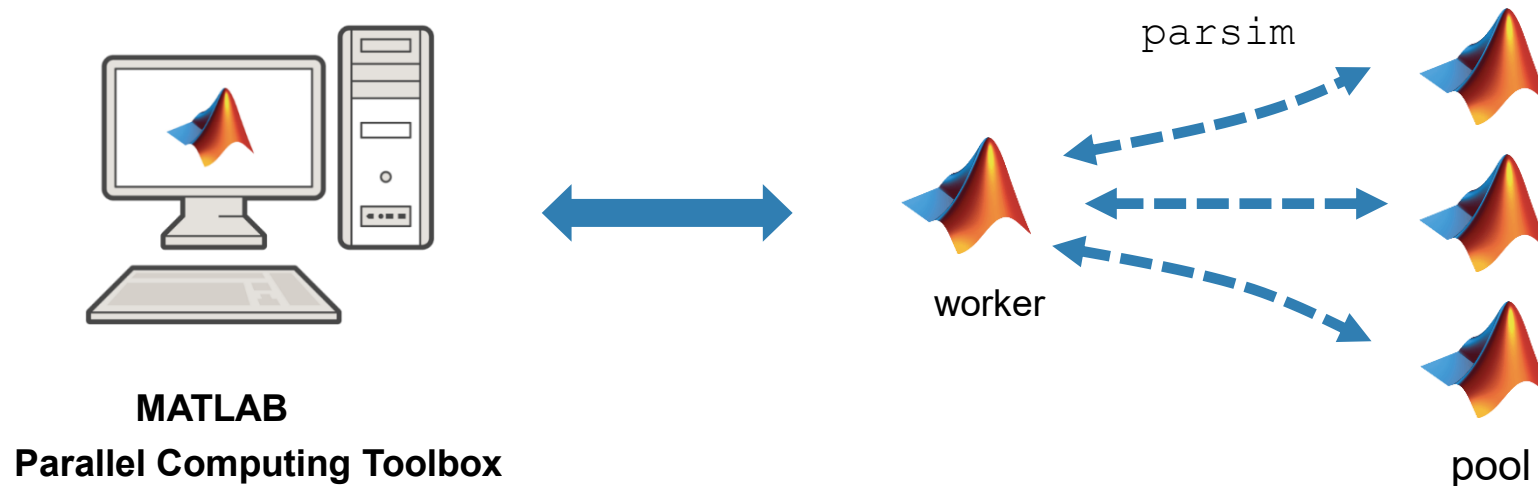
Last updated at Tue Apr 13 08:40:32 EDT 2021

Auto update: Every 5 minutes

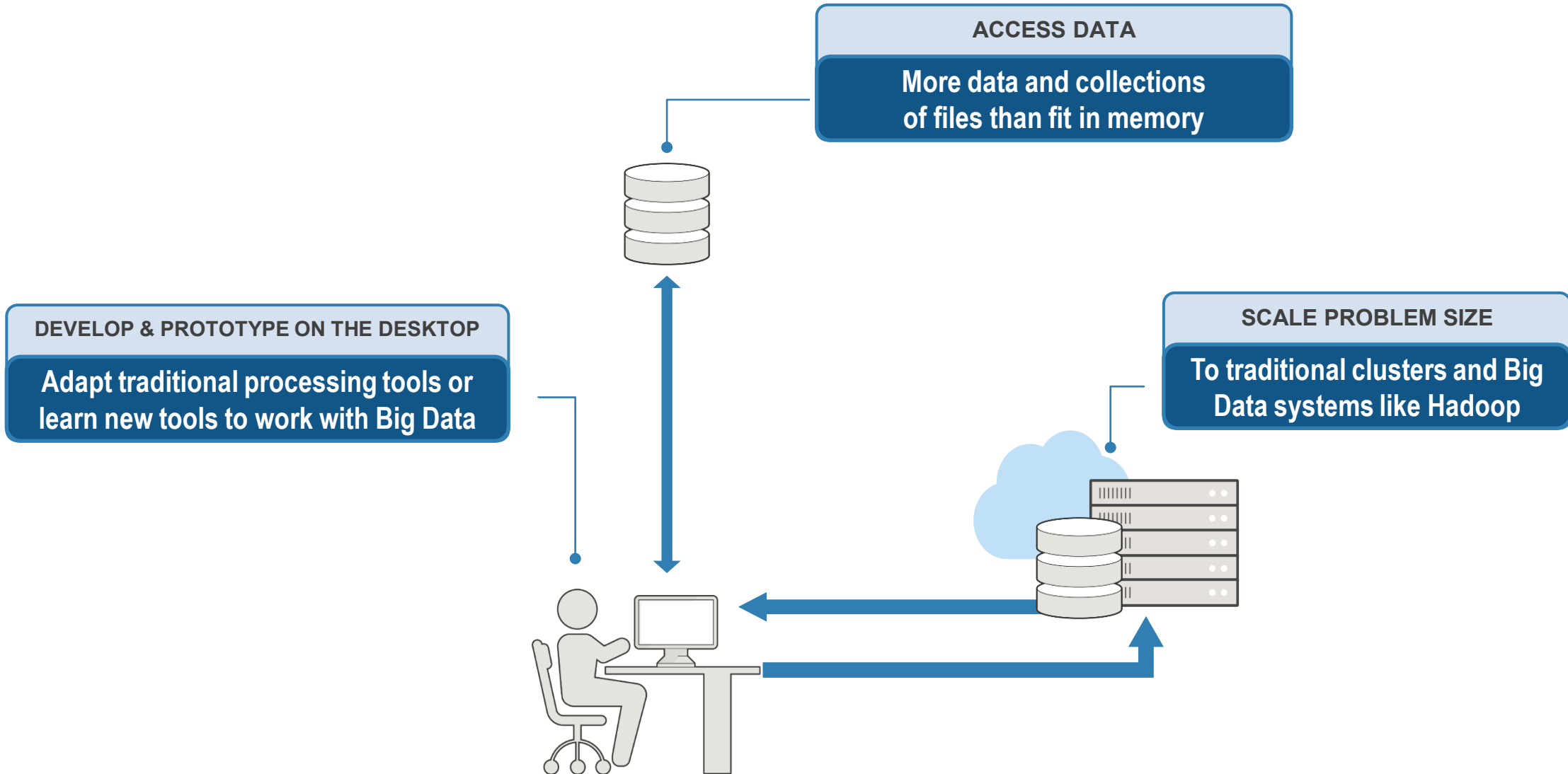
batch simplifies offloading simulations

Submit Simulink jobs to the cluster

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



Big data workflows



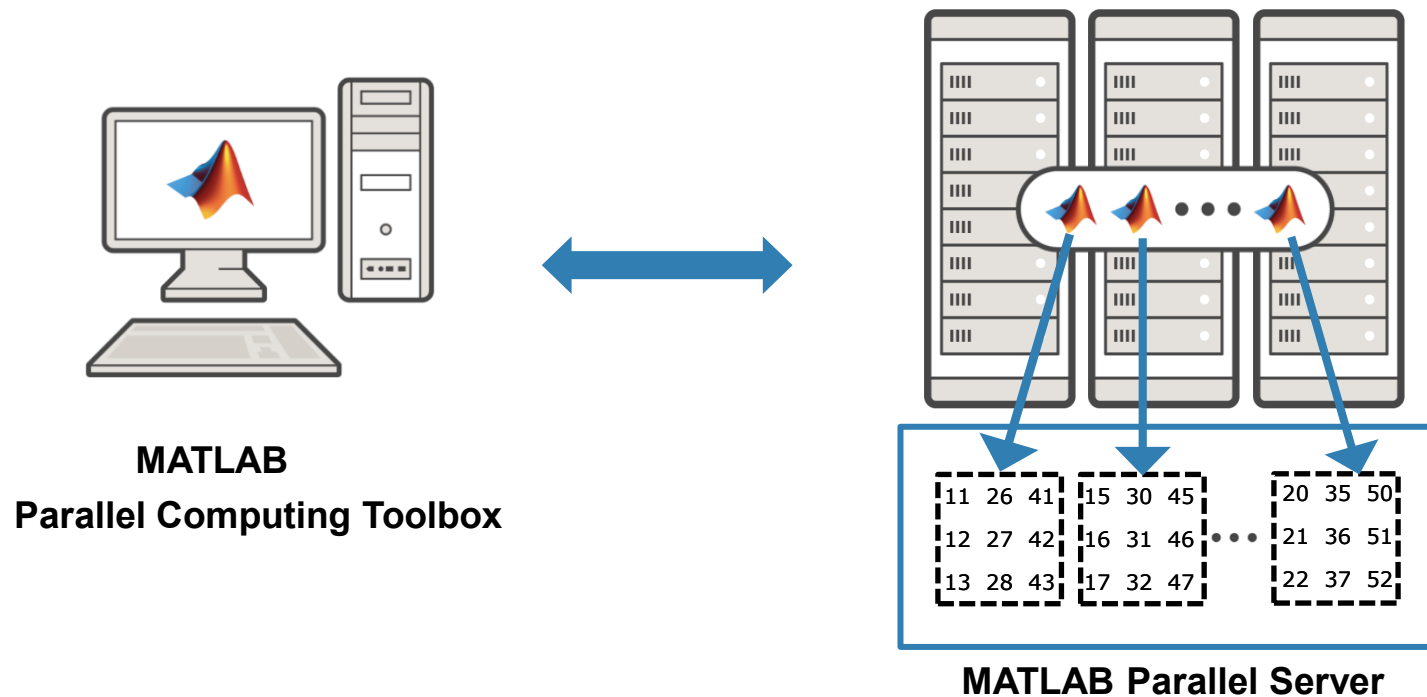
tall arrays

- New 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.)



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



tall arrays vs. distributed arrays

- `tall` arrays are useful for out-of-memory datasets with a “tall” shape
 - Can be used on a desktop, cluster, or with Spark/Hadoop
 - Low-level alternatives are MapReduce and MATLAB API for Spark
- `distributed` arrays are useful for in-memory datasets on a cluster
 - Can be any shape (“tall”, “wide”, or both)
 - Low-level alternative is SPMD + [gop](#) (Global operation across all workers)

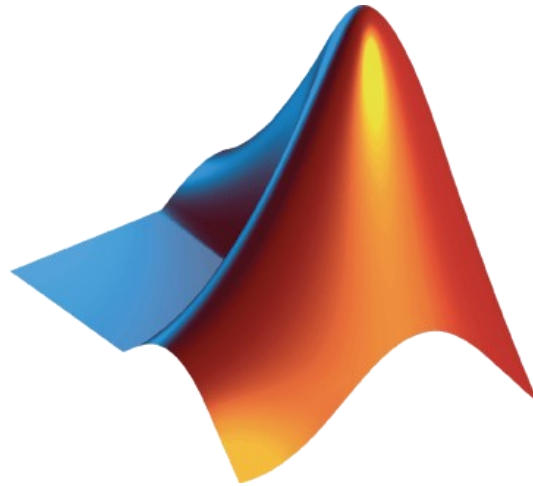
	Tall Array	Distributed Array
Support Focus	Data Analytics, Statistics and Machine Learning	Linear Algebra, Matrix Manipulations
Data Shape	“Tall” only	“Tall”, “wide” or both
Prototype on Desktop	✓	✓
Helps on Desktop	✓	✗
Run on HPC	✓	✓
Run on Spark/Hadoop	✓	✗
Fault Tolerant	✓	✗

Resources

- MATLAB Documentation
 - [MATLAB → Advanced Software Development → 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>



Download Instructions

- <https://drive.matlab.com/sharing/8bd444df-1344-429e-aa71-1da85bf81870>
 - Click on **Add to my Files**
 - Click **Copy Folder**
 - Log into MATLAB Drive with MathWorks Account

- https://www.mathworks.com/licensecenter/classroom/PC_3442250
 - Enter MathWorks email address
 - Click **Next**
 - Click **Access MATLAB Online** (maybe prompted to sign-in again)
 - In Current Folder, double click on **Parallel Computing Workshop**
 - Right click on **startWorkshop**, select **Run**

Start Workshop

```
>> startWorkshop
```

```
MATLAB version verified.
```

```
Parallel Computing Toolbox is licensed.
```

```
Parallel Computing Toolbox is installed.
```

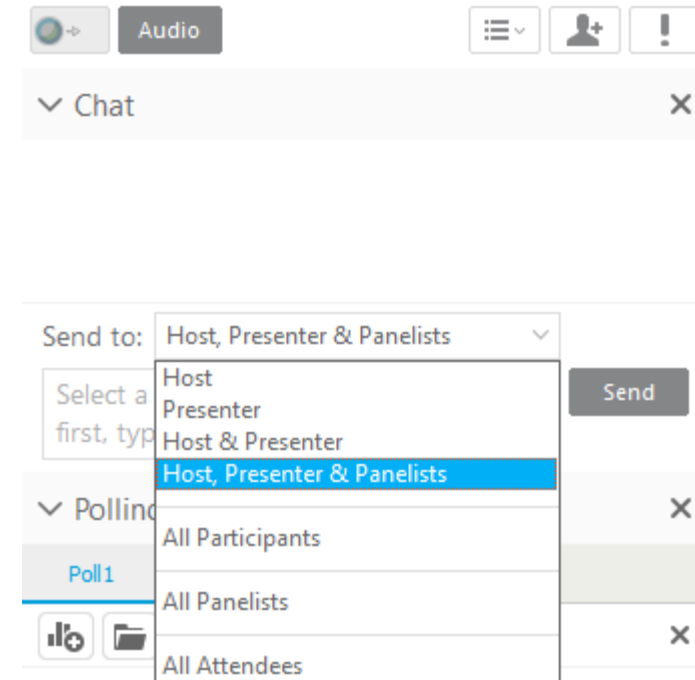
```
Parallel Computing Workshop content successfully added to MATLAB path.
```

```
Review WorkshopInstructions to get started with the workshop.
```

```
>>
```

Chatting

- Send to at least the *Host, Presenter & Panelists*
- Ideally, send to *All Attendees*



Agenda

- Part I – Parallel Computing with MATLAB on the Desktop
 - Parallel Computing Toolbox
 - MATLAB Online
- Part II – Scaling MATLAB to magnitUDE
 - MATLAB Parallel Server
 - MATLAB Desktop
- Next Week
 - Thursday, May 27th 2-5pm
 - MATLAB / PCT installed (R2020b or R2021a)
 - magnitUDE account

Required: MATLAB & Parallel Computing Toolbox

```
>> license('test', 'distrib_computing_toolbox')
```

```
ans =
```

```
1
```

R2021a

R2020b