Workshop: Parallel Computing with MATLAB and Scaling to HPCC

Raymond Norris
MathWorks

Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting

What's Not Being Covered Today?

- Data Parallel
- MapReduce
- MPI
- Simulink

Let's Define Some Terms

cli·ent noun \ˈklī-ənt

1 : MATLAB session that submits the job

com·mu·ni·cate job *adjective* \kə-ˈmyü-nə-ˌkāt\ \ˈjäb

1 : a job composed of tasks that communicate with each other, running at the same time

in·de·pen·dent job *adjective* \ˌin-də-ˈpen-dənt\ \ˈjäb

1 : a job composed of independent tasks, with no communication, which do not need to run at the same time

lab *noun* \ˈlab

1 : see worker

…a Few More Terms

MAT·LAB pool *noun* \mat-lab\ \ˈpül

1 : a collection of workers

SPMD abbreviation

1 : Single Program Multiple Data

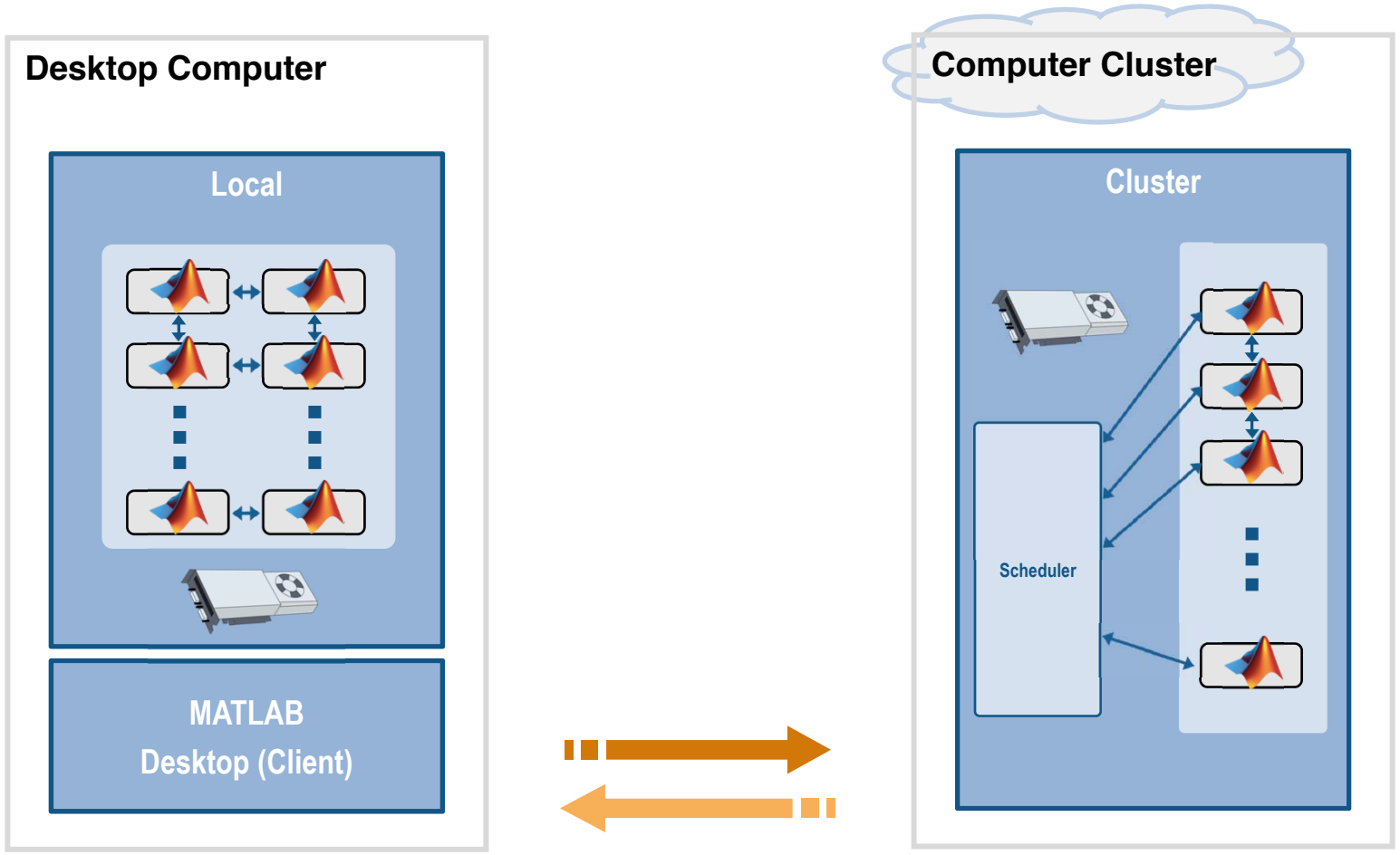
MDCS abbreviation

1 : MATLAB Distributed Computing Server

worker *noun* \ˈwər-kər

1 : headless MATLAB session that performs tasks

MATLAB Parallel Computing Solution



Parallel Computing Toolbox

MATLAB Distributed Computing Server

Typical Parallel Applications

- Massive `for` loops (`parfor`)
  - Parameter sweep
    - Many iterations
    - Long iterations
  - Monte-Carlo simulations
  - Test suites
- One-Off Batch Jobs

Task Parallel Applications

- Partition Large Data Sets (`spmd`)

Data Parallel Applications

Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting

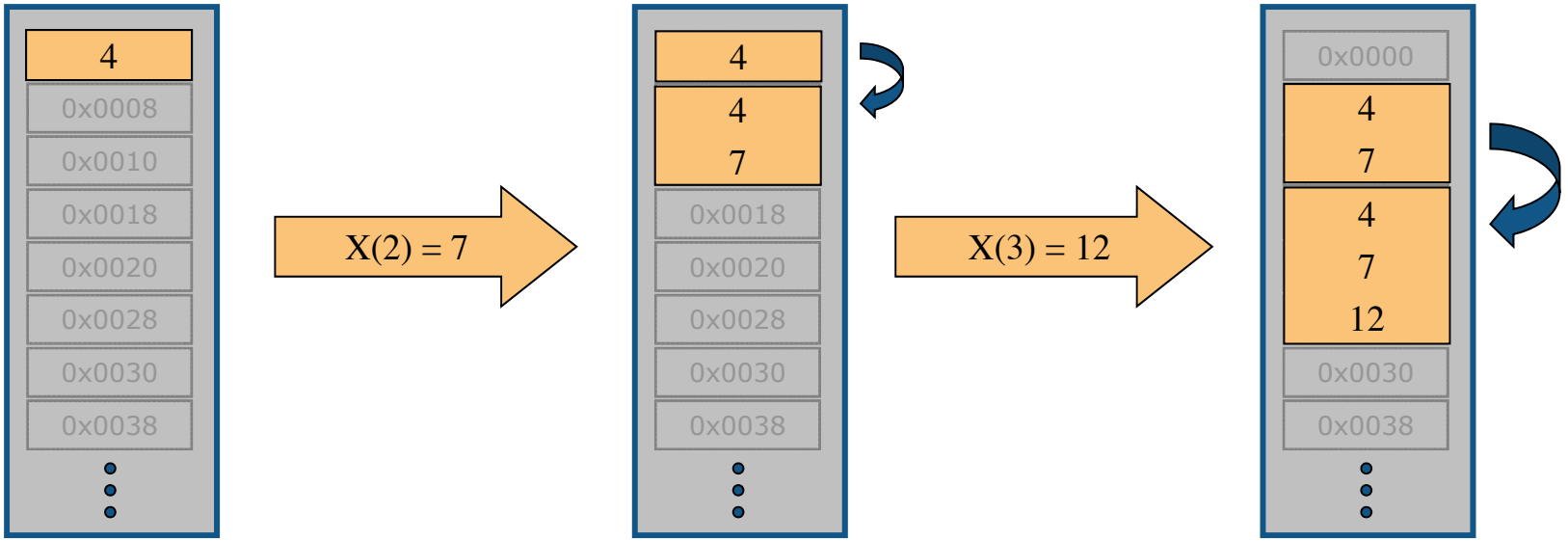
But Before We Get Started…

- Do you preallocate your matrices?

Effect of Not Preallocating Memory

```
>> x = 4;
>> x(2) = 7;
>> x(3) = 12;
```

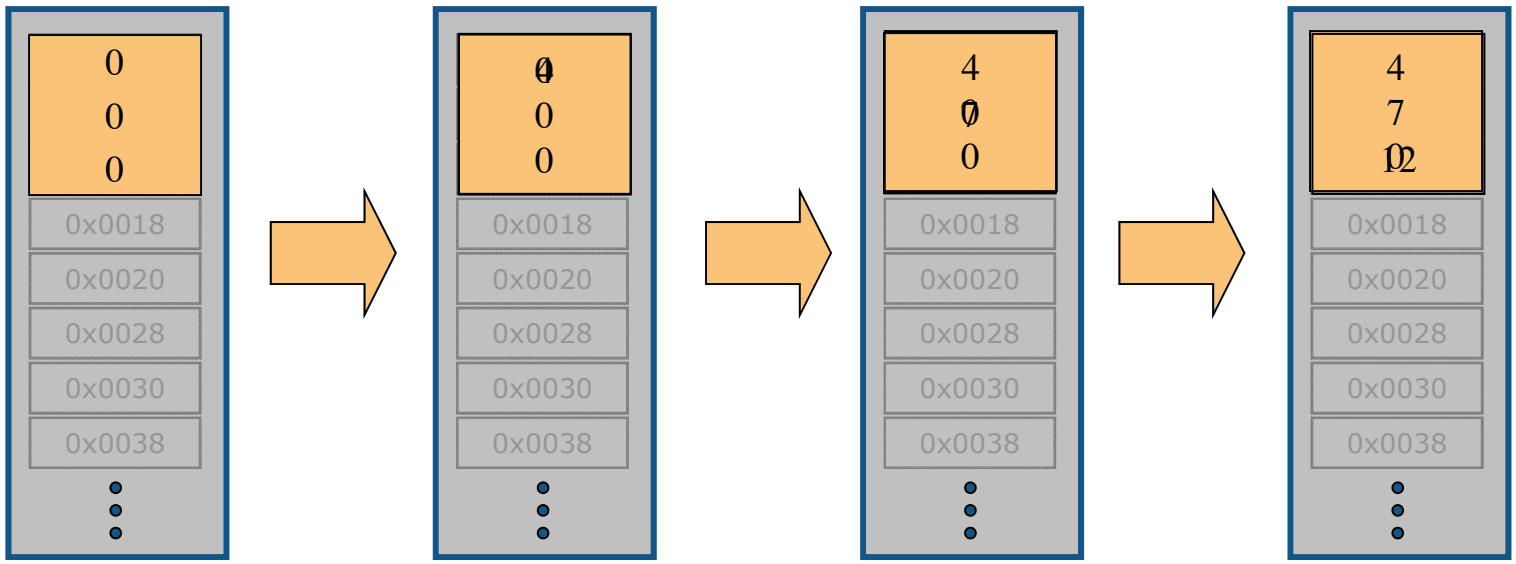
Resizing Arrays is Expensive



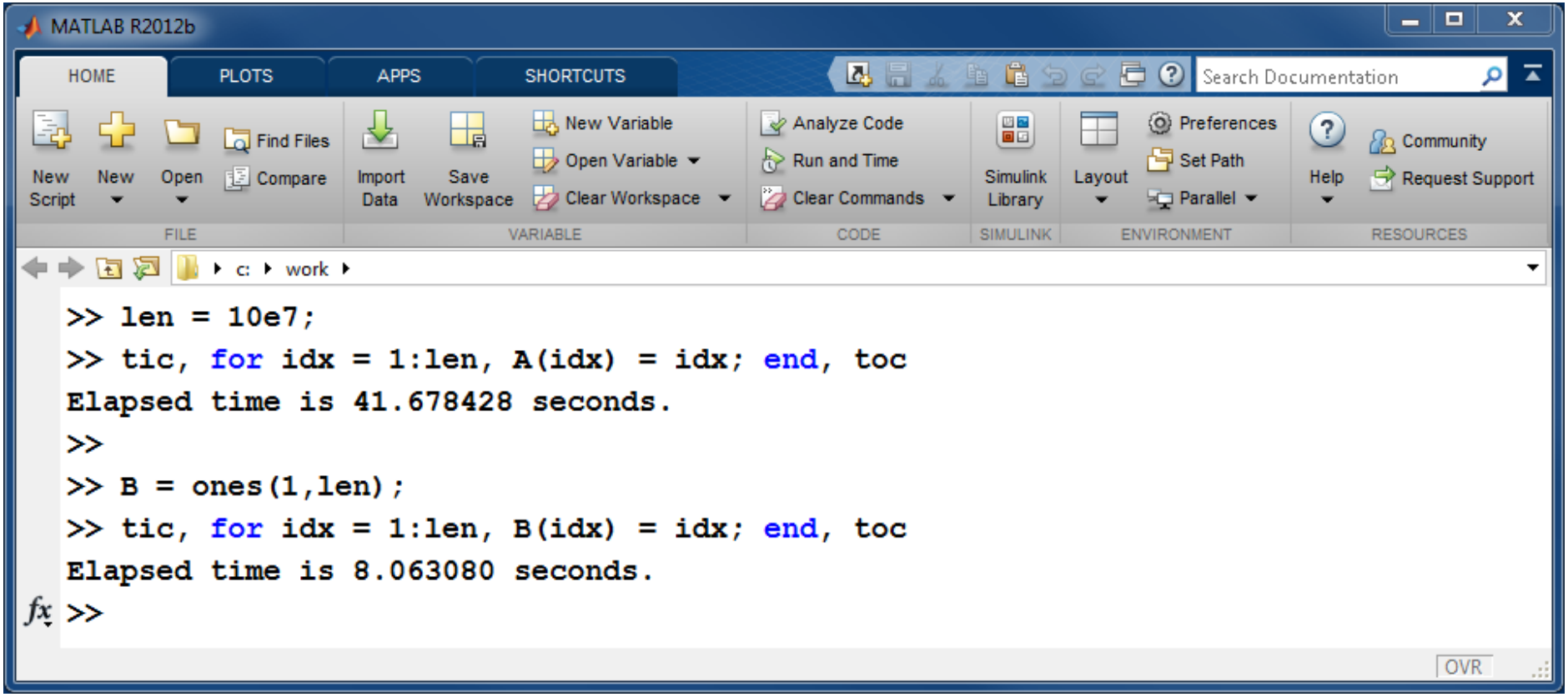
Benefit of Preallocation

```
>> x = zeros(3,1);
>> x(1) = 4;
>> x(2) = 7;
>> x(3) = 12;
```

Reduced Memory Operations

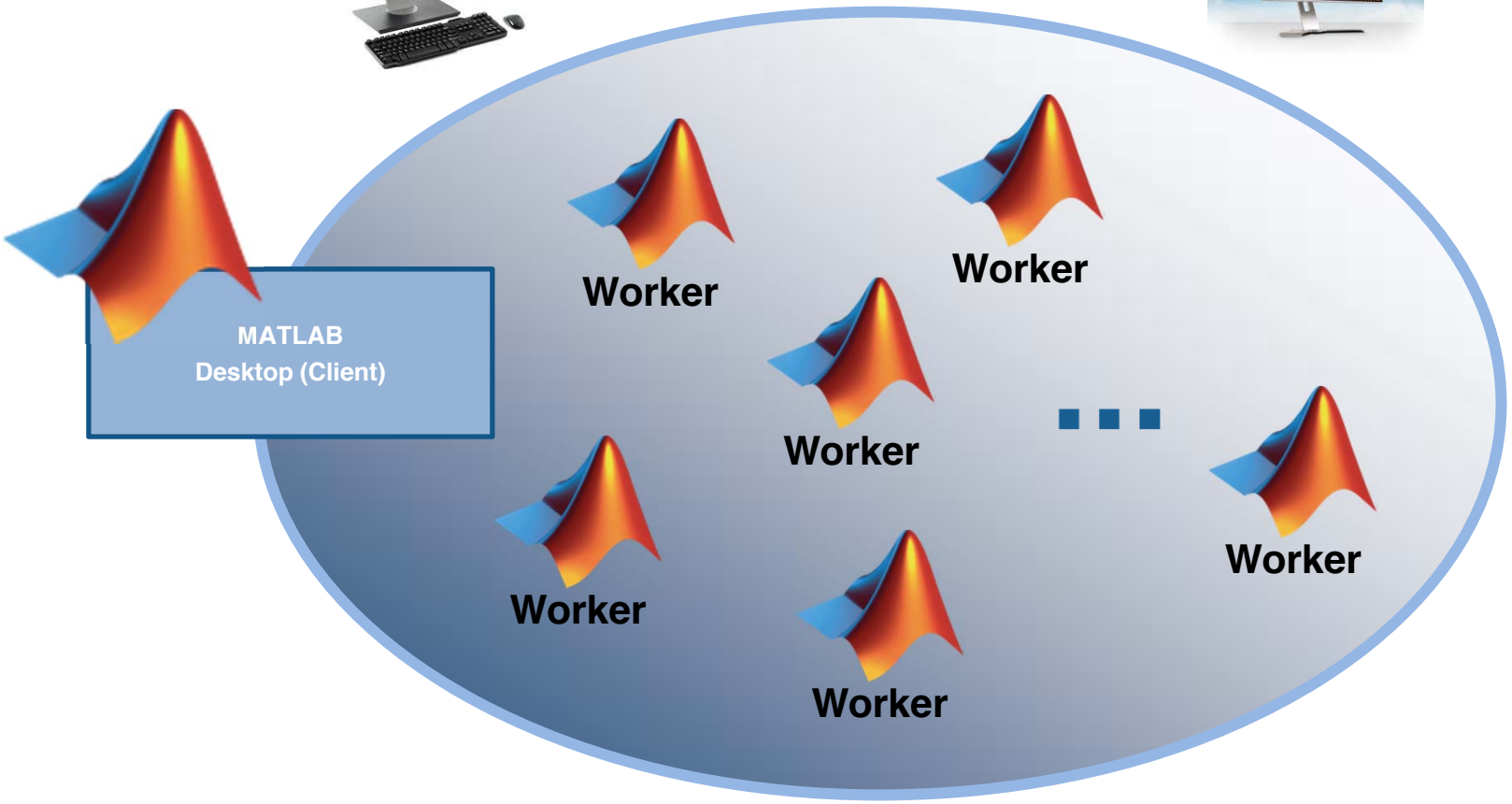


Let's Try It…


MATLAB R2012b
HOME
PLOTS
APPS
SHORTCUTS
Search Documentation
New Script
New
Open
Find Files
Compare
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
Analyze Code
Run and Time
Clear Commands
Simulink Library
Layout
Preferences
Set Path
Parallel
Help
Community
Request Support
FILE
VARIABLE
CODE
SIMULINK
ENVIRONMENT
RESOURCES
c: ▸ work ▸
>> len = 10e7;
>> tic, for idx = 1:len, A(idx) = idx; end, toc
Elapsed time is 41.678428 seconds.
>>
>> B = ones(1,len);
>> tic, for idx = 1:len, B(idx) = idx; end, toc
Elapsed time is 8.063080 seconds.
>>
OVR

Getting Started With the MATLAB Pool

The MATLAB Pool



Connecting to HPCC to Run MATLAB

```
ssh -X USERNAME@hpc-login1.usc.edu

## For bash users
% cp ~matlab/setup_matlab.sh ~/
% source setup_matlab.sh

## For tcsh users
% cp ~matlab/setup_matlab.csh ~/
% source setup_matlab.csh

% matlab_local ## or matlab_cluster

ssh -X COMPUTE-NODE
. /usr/usc/matlab/2013a/setup.[c]sh
% matlab &
```

Only for today's seminar

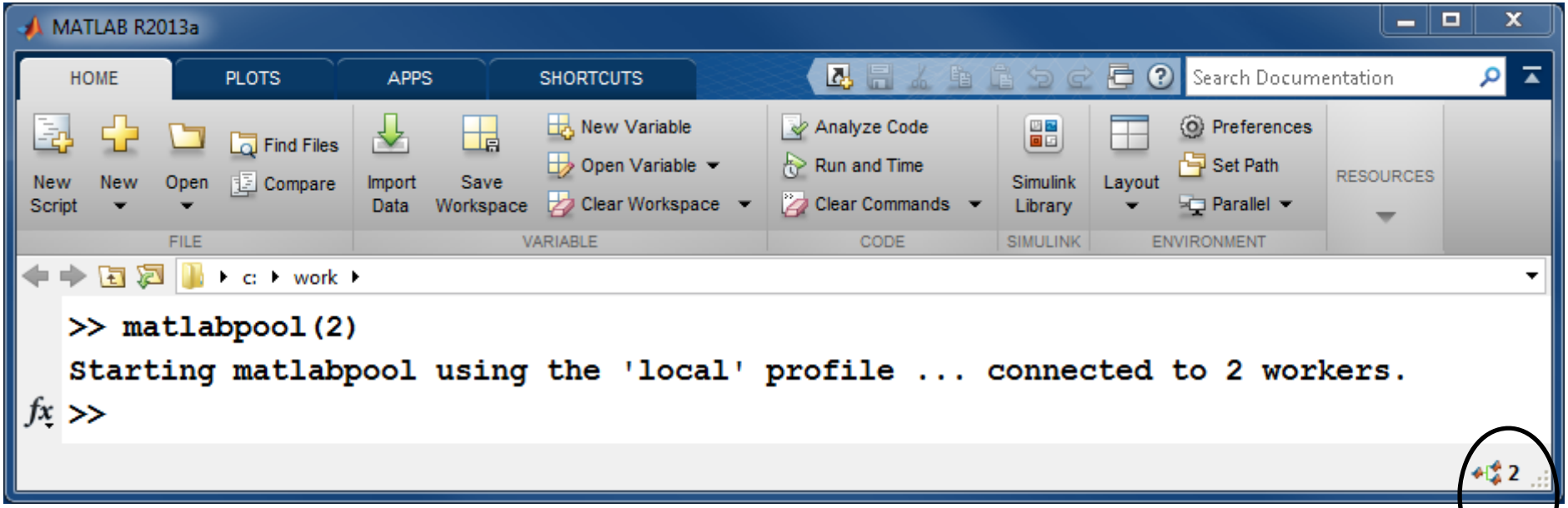
To be updated on the Wiki

Starting a MATLAB Pool…

Bring up the Windows Task Manager or Linux *top*

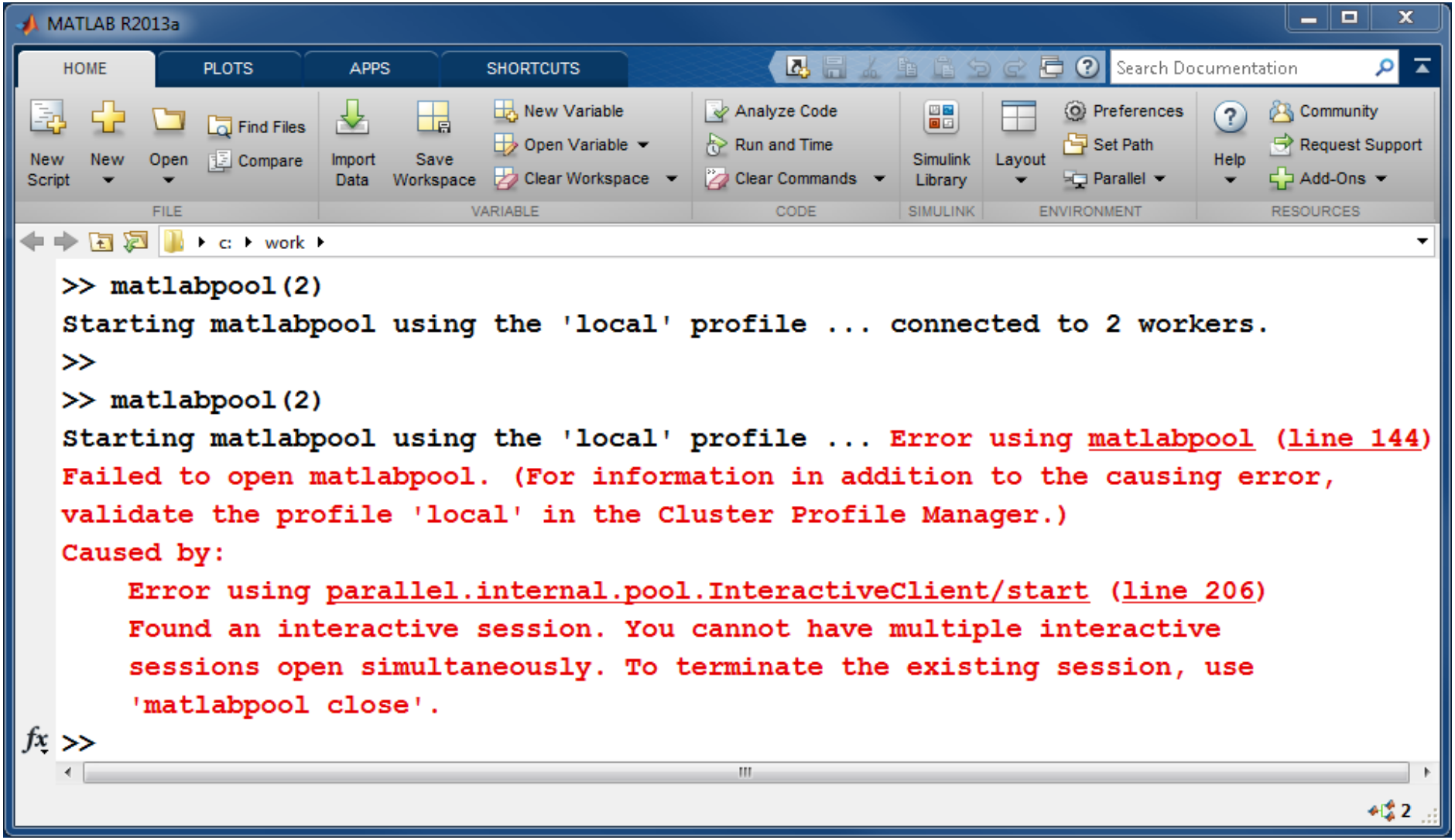
Start MATLAB

Open a MATLAB pool with two workers using the local profile



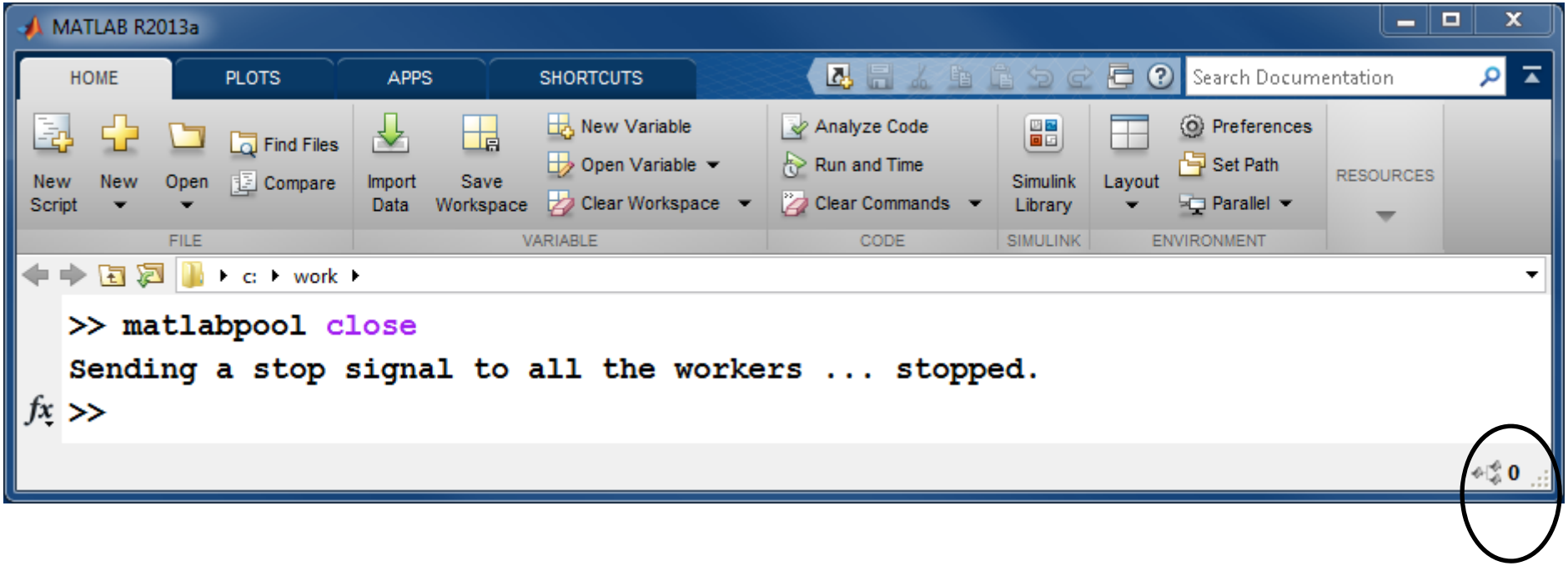
Maximum of 12 local workers

One MATLAB Pool at a Time



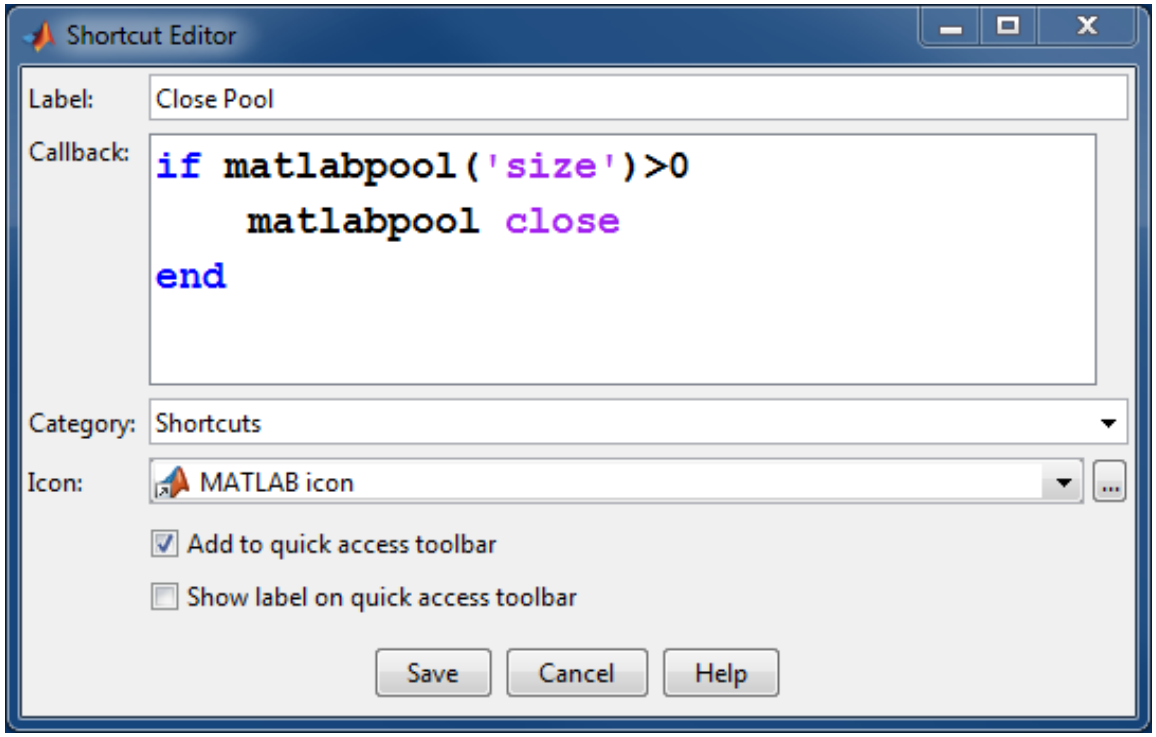
Even if you have not exceeded the maximum number of workers, you can only open one MATLAB pool at a time

Stopping a MATLAB Pool

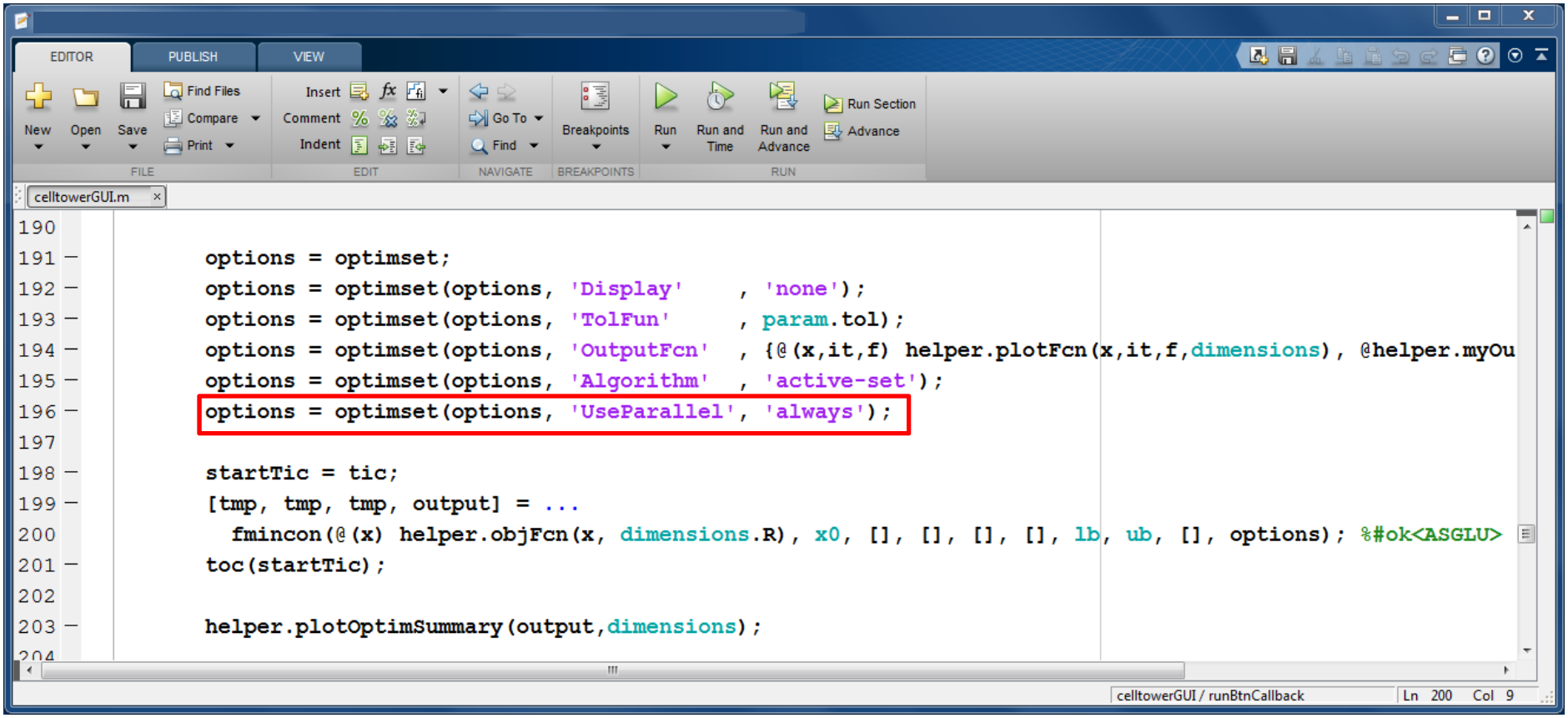


Add Shortcut for Starting the MATLAB Pool

Add Shortcut for Stopping the MATLAB Pool



Toolbox Support for Parallel Computing



Products That Support PCT

- Bioinformatics Toolbox
- Communications System Toolbox
- Embedded Coder
- Global Optimization Toolbox
- Image Processing Toolbox
- Model-Based Calibration Toolbox
- Neural Network Toolbox
- Optimization Toolbox
- Phased Array System Toolbox
- Robust Control Toolbox
- Signal Processing Toolbox
- Simulink
- Simulink Coder
- Simulink Control Design
- Simulink Design Optimization
- Statistics Toolbox
- SystemTest

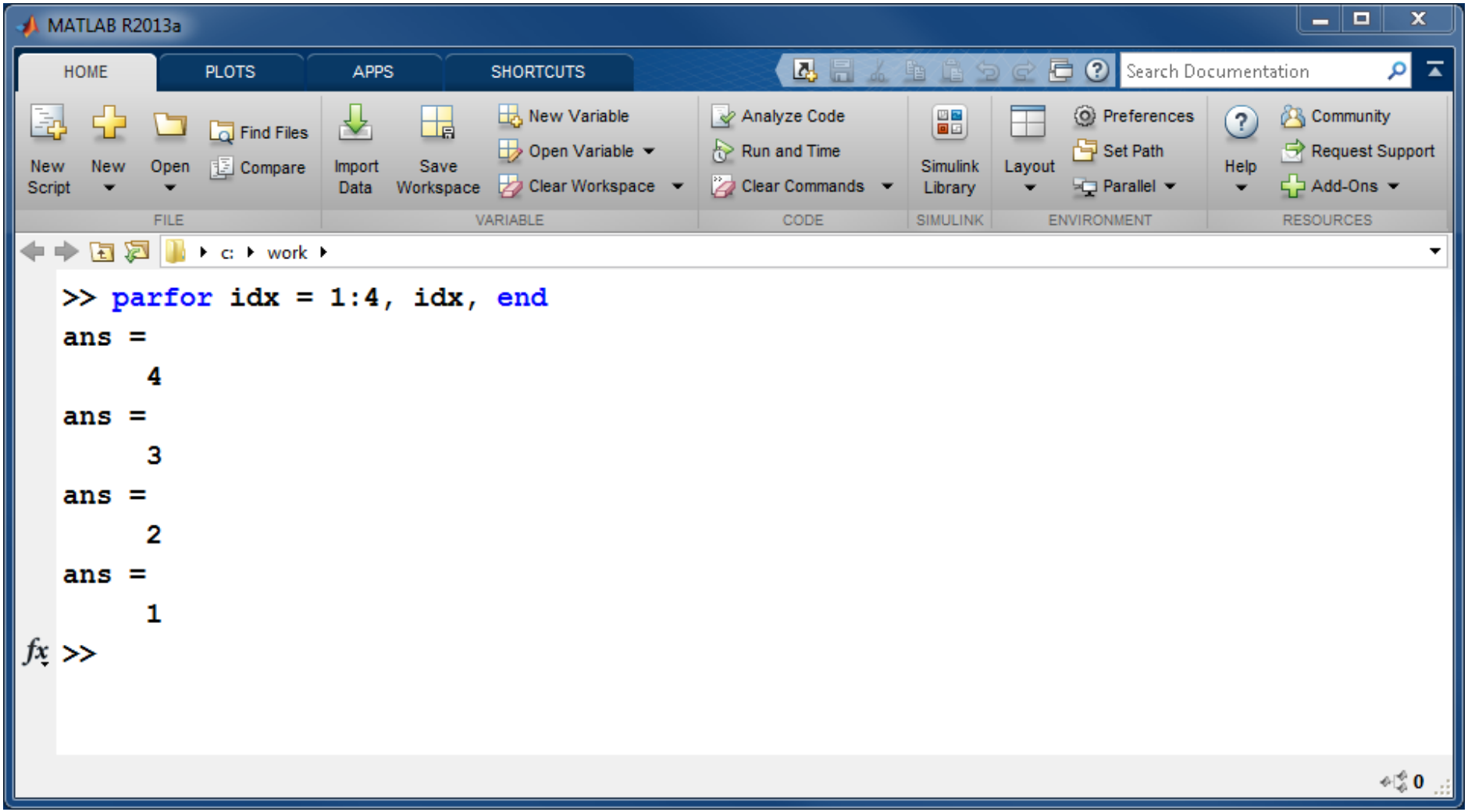
http://www.mathworks.com/products/parallel-computing/builtin-parallel-support.html

`parfor`: The Parallel `for` Loop

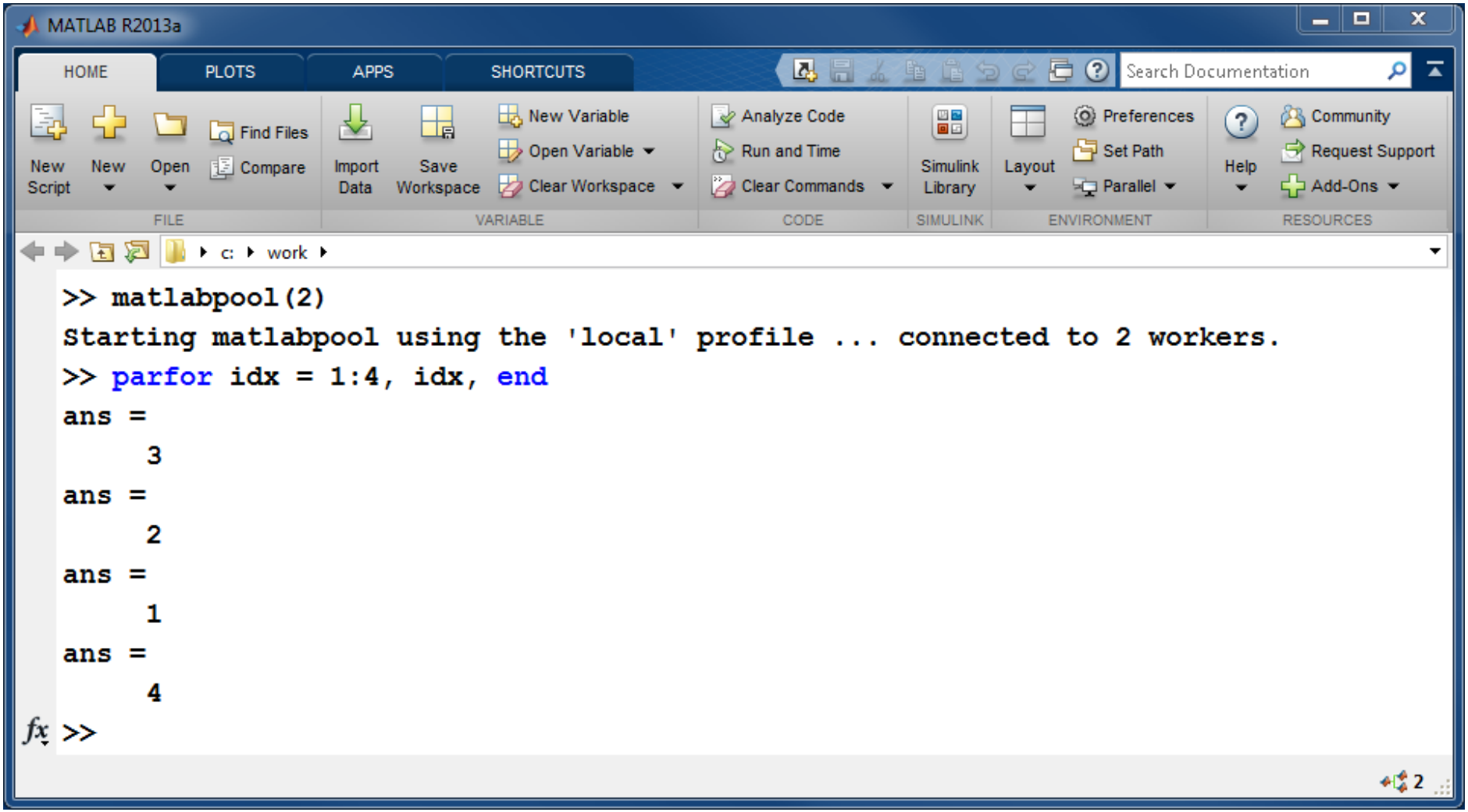
Using the `parfor` Construct

- In order to convert a `for` loop to a `parfor` loop, the `for` loop must at least be:
  - Task independent
  - Order independent

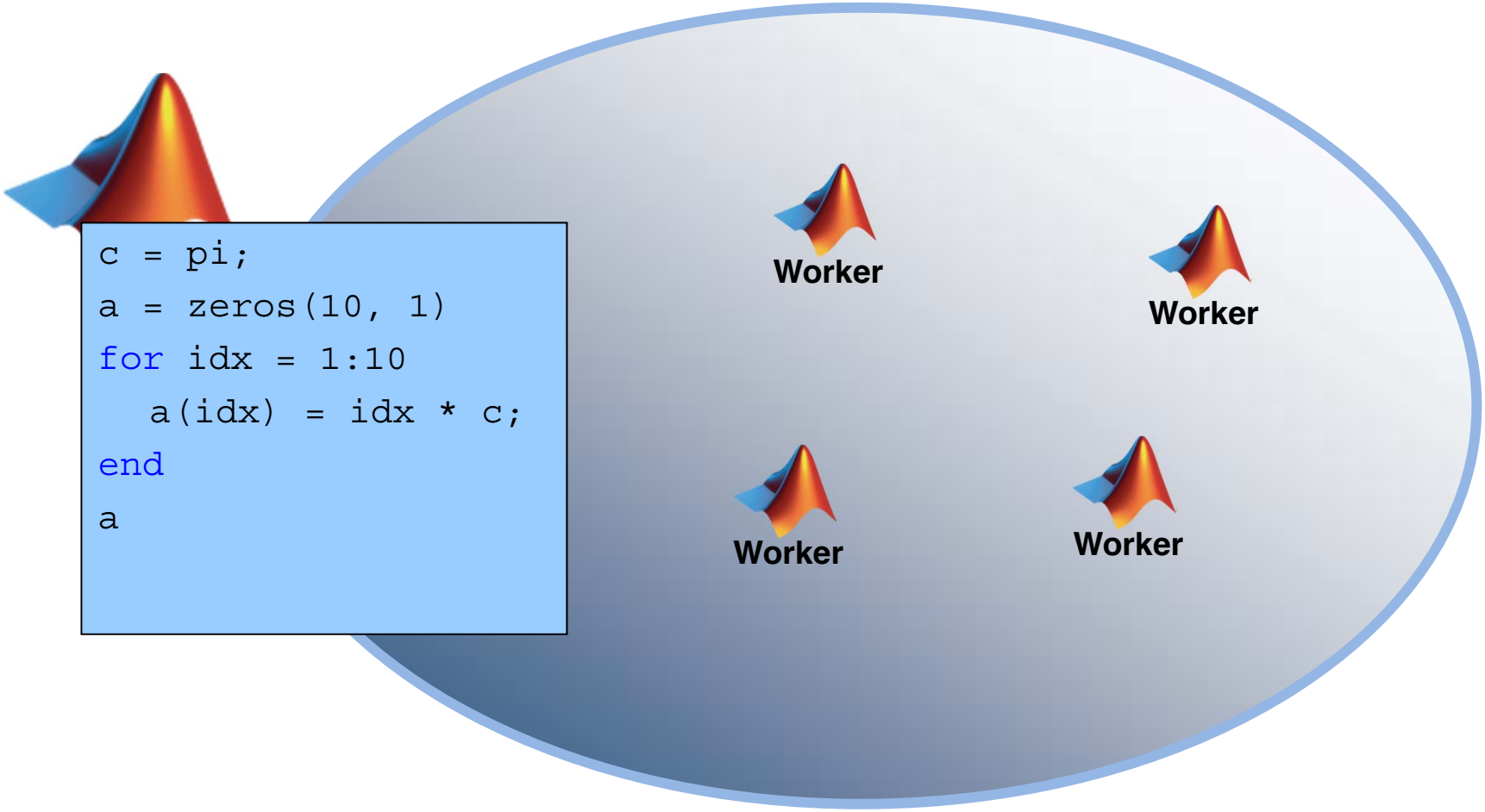
Order Independent?



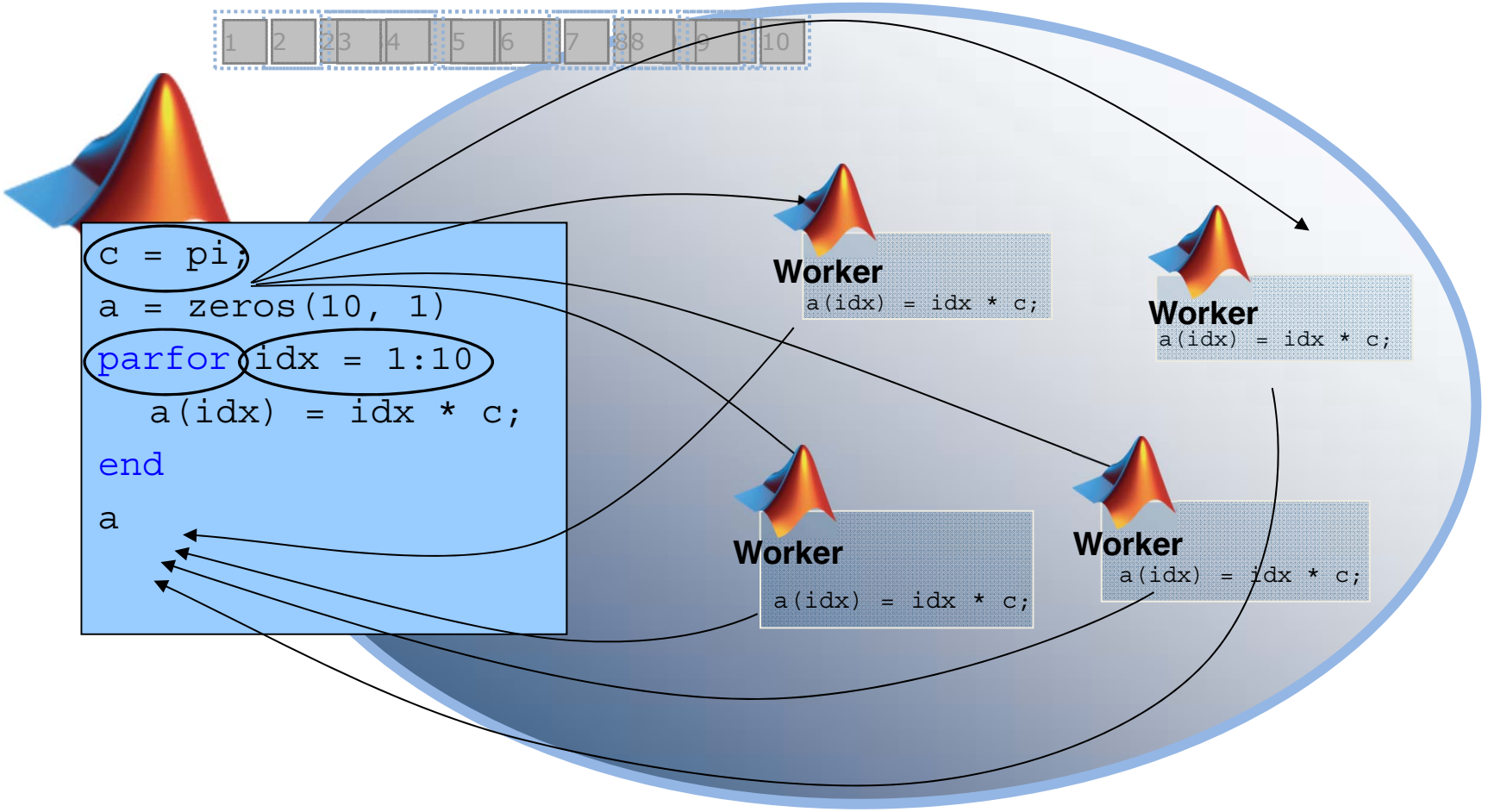
What If a MATLAB Pool Is Running?



The Mechanics of `parfor` Blocks



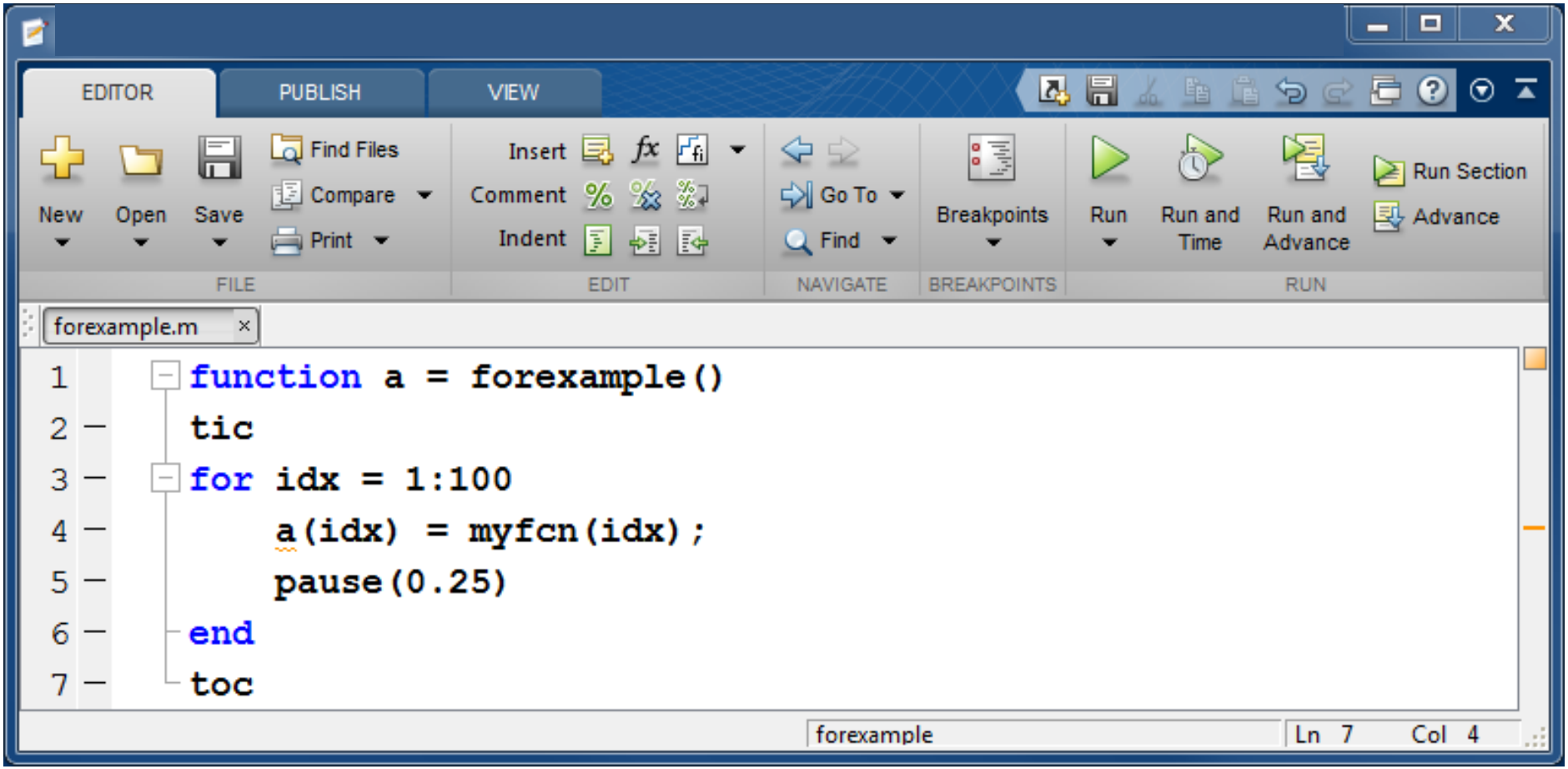
The Mechanics of `parfor` Blocks



Auto-load balancing Pool of MATLAB Workers

Example: Hello, World!

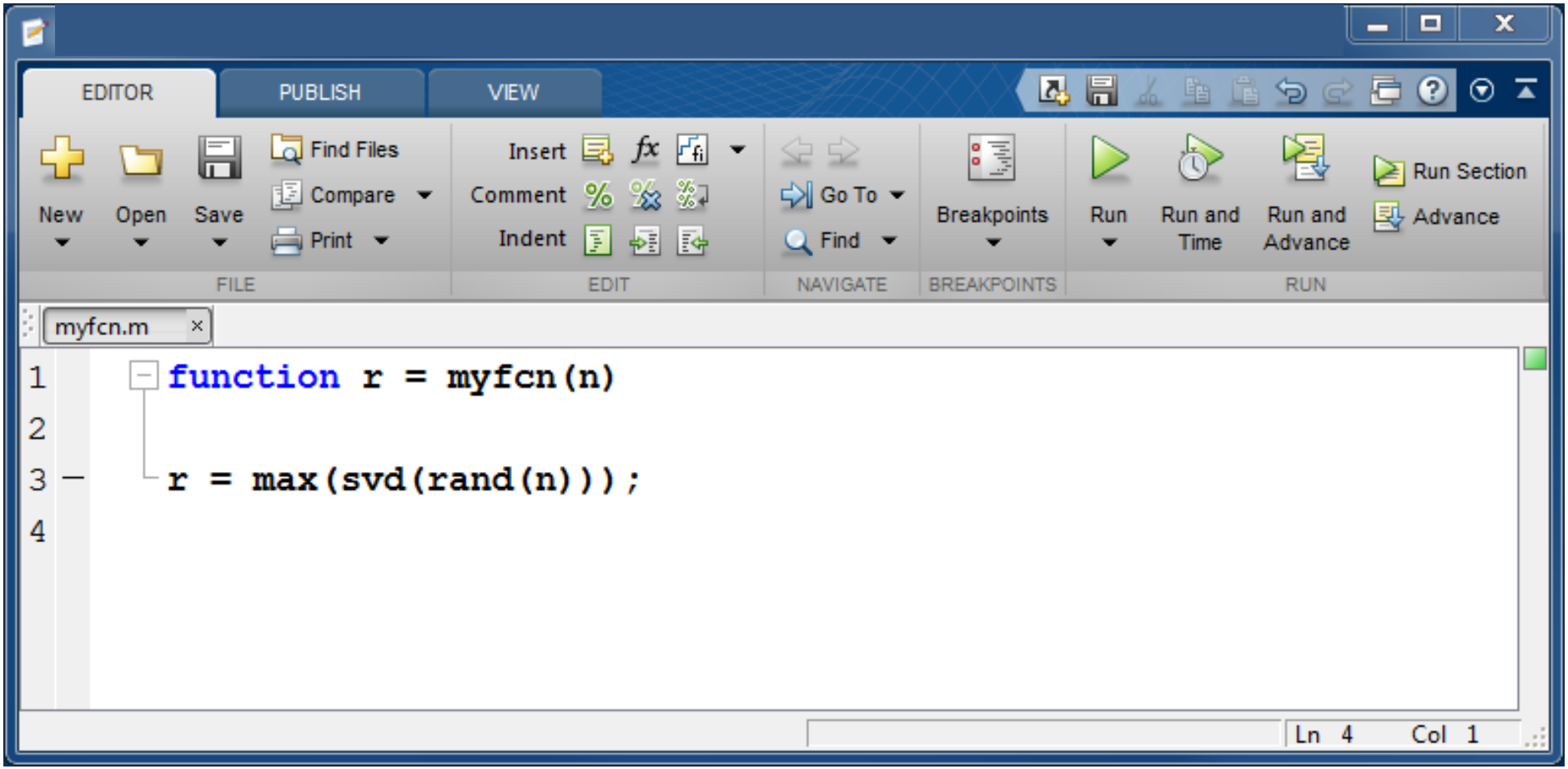
1. Code the example below. Save it as **forexample.m**




```
>> forexample
```

Example: Hello, World! (2)

2. Code the helper function. Save it as **myfcn.m** . Time and run it.

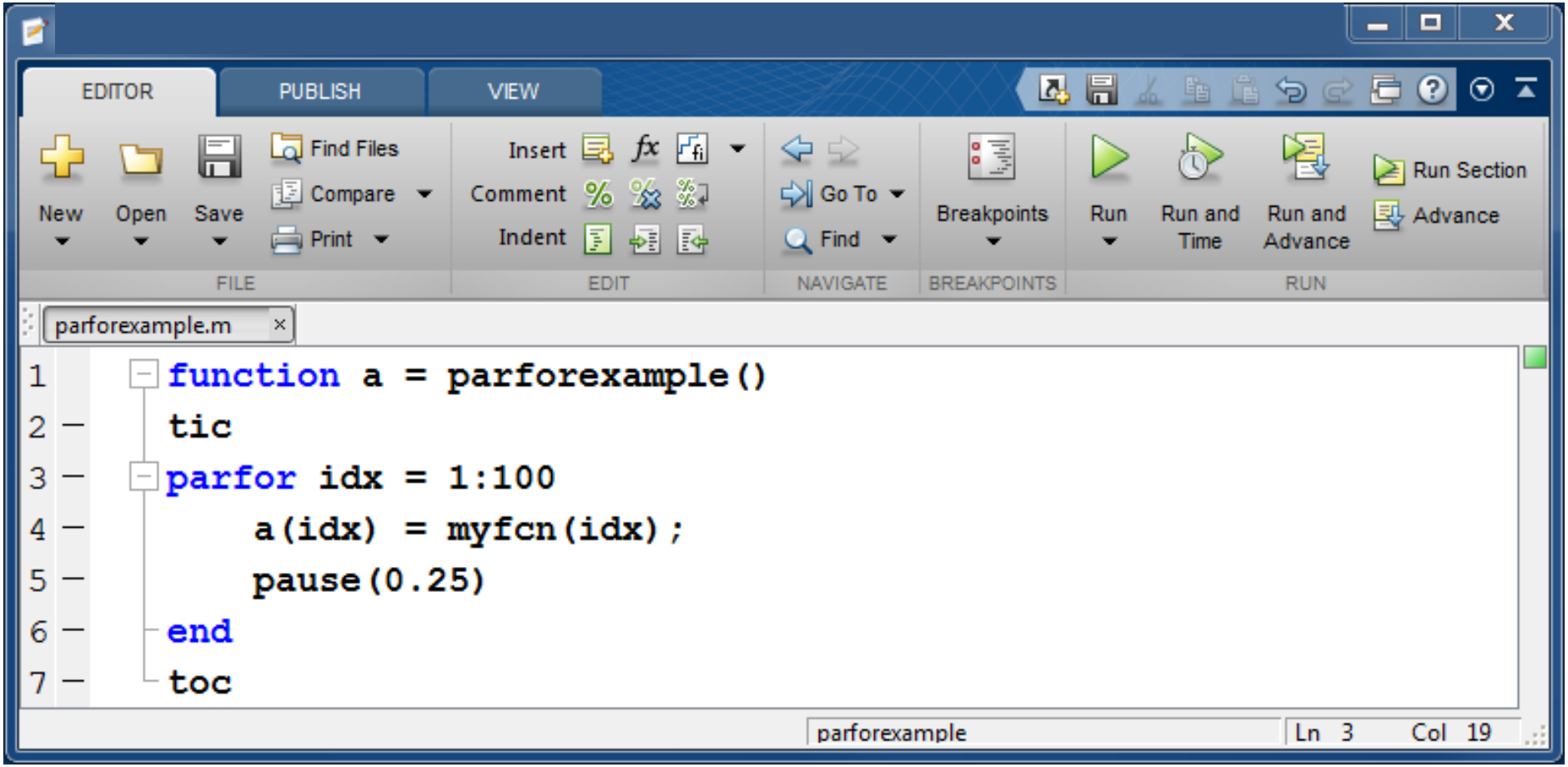



```
>> myfcn
```

Example: Hello, World! (3)

3. Parallelize the `for` loop and save it as **parforexample.m**

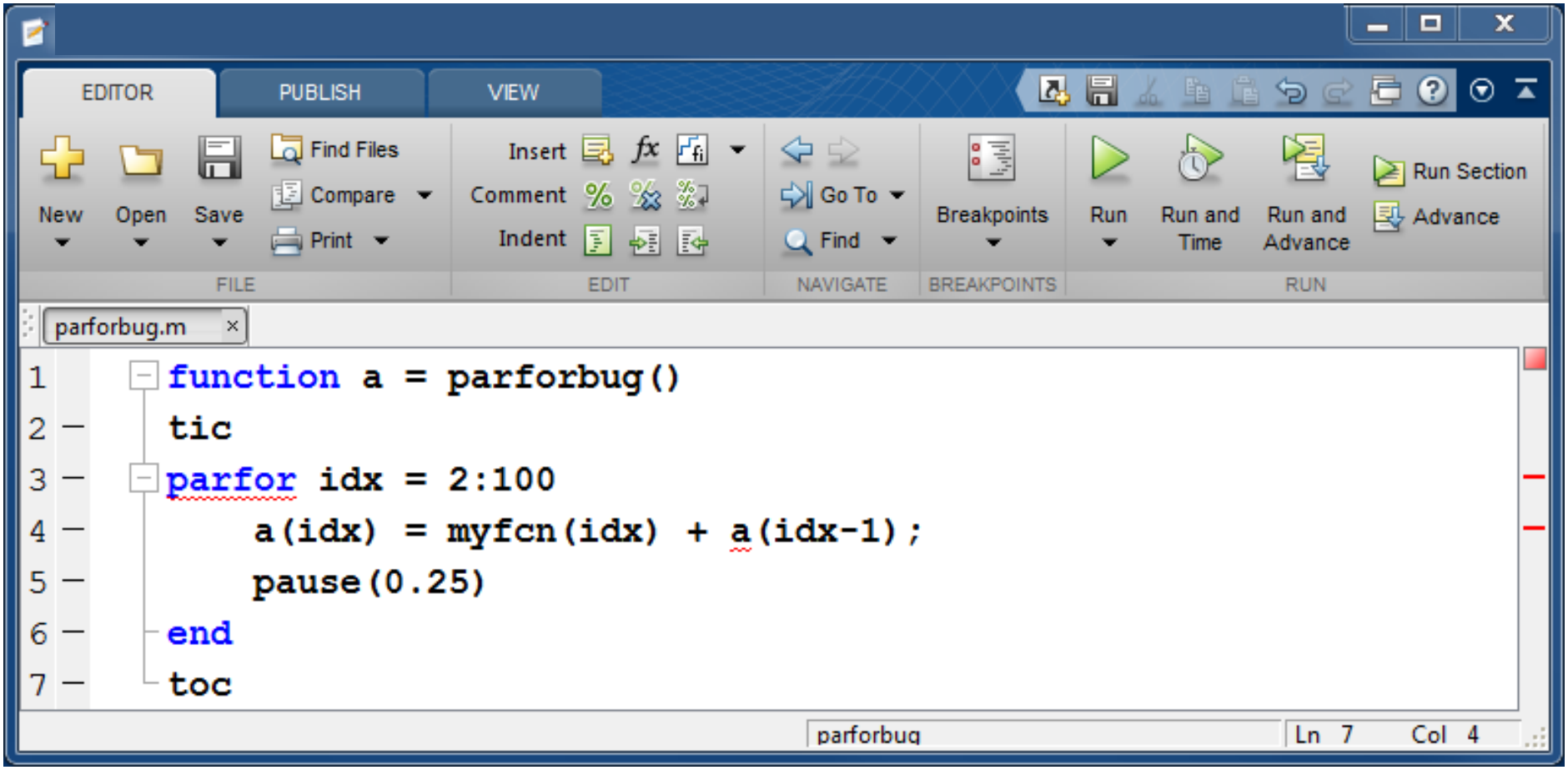
4. Start a MATLAB pool and run it. Change the size of the Pool. What speed ups do you get?




```
>> parforexample
```

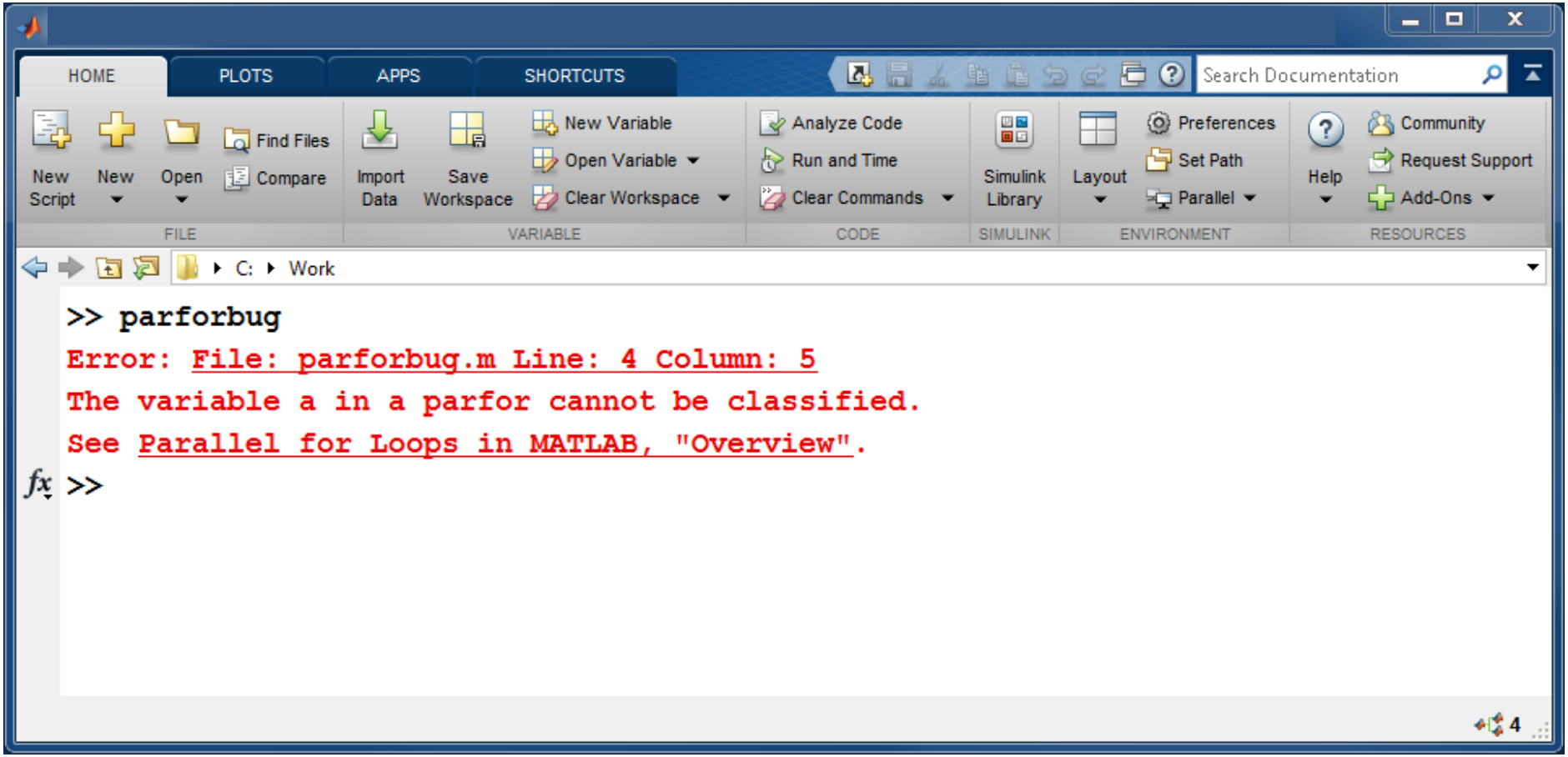
Example: Break It (1)

5. Add a dependency to the `parfor` loop. Look at the code analyzer messages.




```
>> parforbug
```

Example: Break It (2)



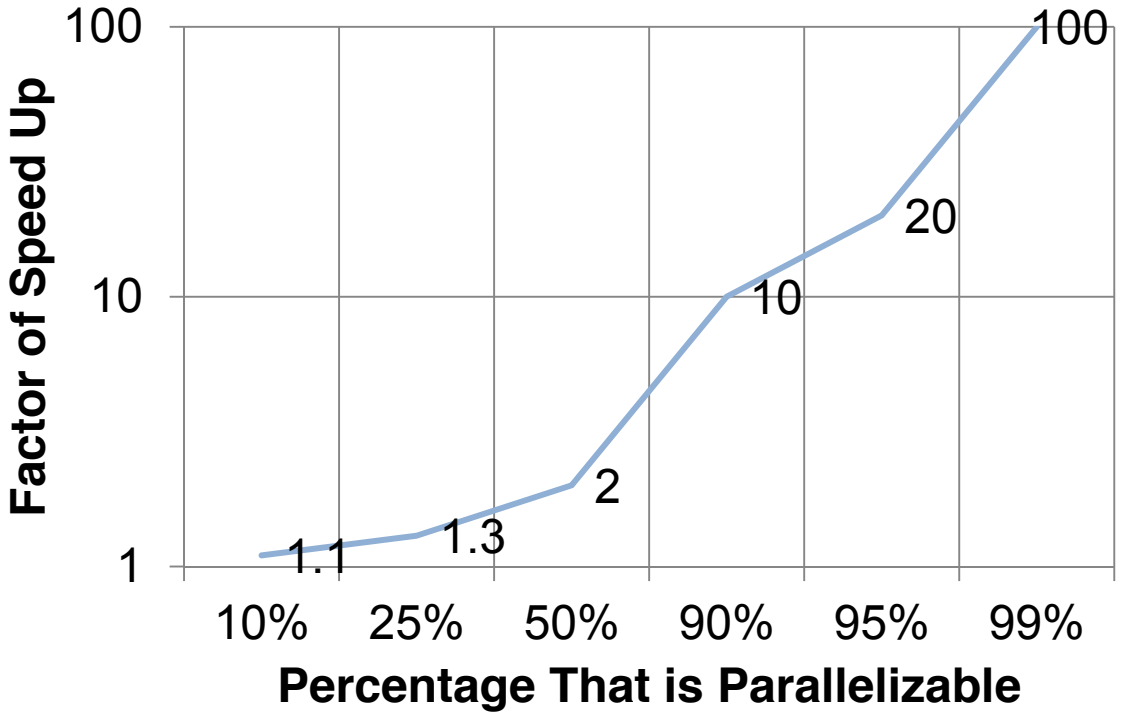
The variable `a` cannot be properly classified

Constraints

- The loop variable cannot be used to index with other variables
- No inter-process communication. Therefore, a `parfor` loop cannot contain:
  - break and return statements
  - global and persistent variables
  - nested functions
  - changes to handle classes
- Transparency
  - Cannot “introduce” variables (e.g. `eval`, `load`, `global`, etc.)
  - Unambiguous Variables Names
- No nested `parfor` loops or `spmd` statement

This is Great! Should I Get Linear Improvement?

- Not exactly
  - Too little work, too much data
- Are you calling BLAS or LAPACK routines?
- What are you timing?
  - MATLAB Profiler
- Amdahl's Law
  - $SU(N) = \frac{1}{(1-P)+\frac{P}{N}}$



Optimizing a `parfor` Loop

- Should I pre-allocate a matrix?
  - There is no significant speedup, if any, in pre-allocating the matrix
- Should I pre-assign large matrices before the `parfor`?
  - Yes, if they're going to be referenced after the `for` loop (to be explained why later)
  - Otherwise, do all the large creation on the workers
  - So if I have a `for` loop with 100 iterations and 10 workers, are each of the matrices create 10 times? Or 100 times?
    - 100 times. See later for minimizing this.

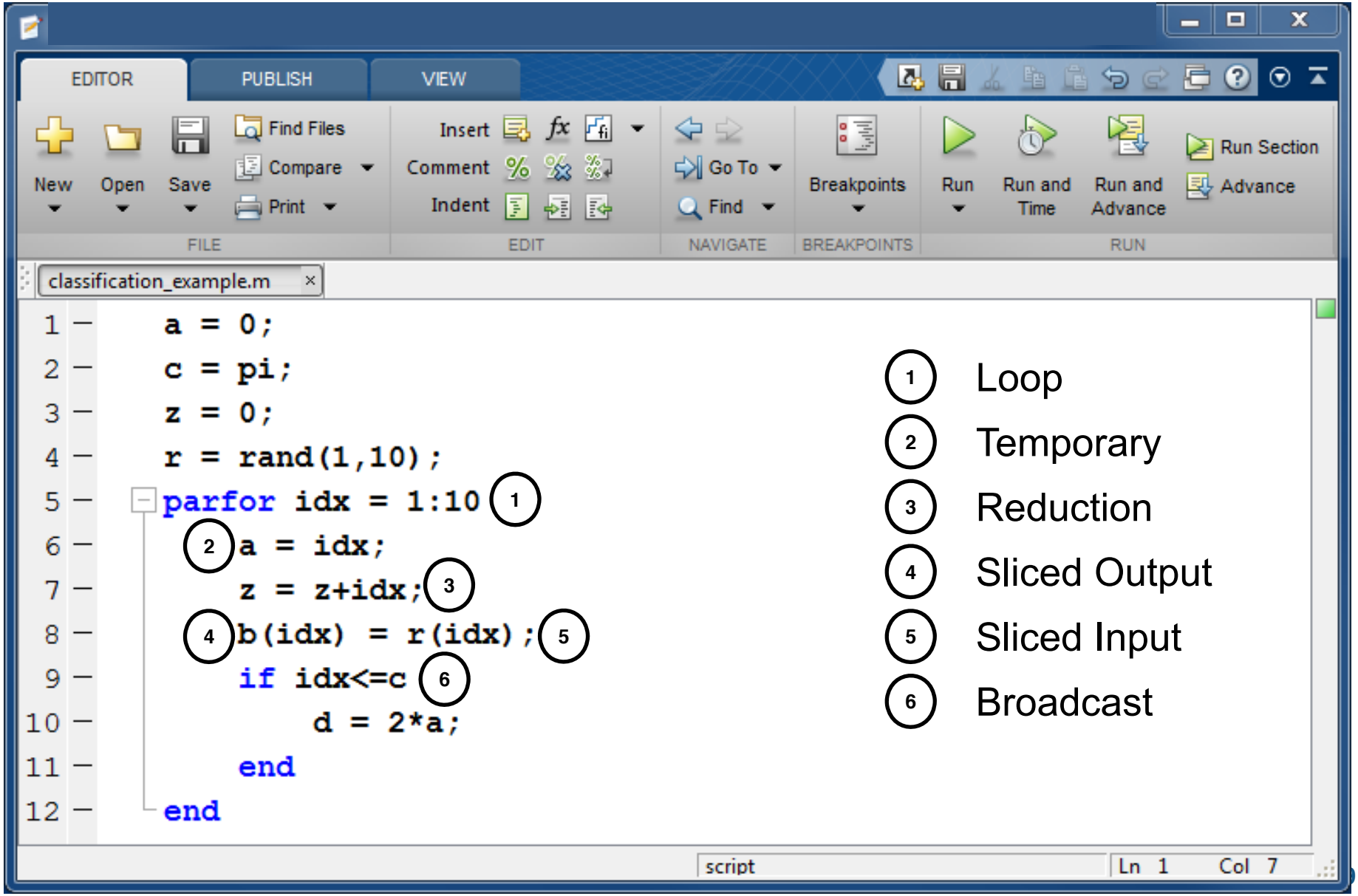
`parfor` Variable Classification

- All variables referenced at the top level of the `parfor` must be resolved and classified

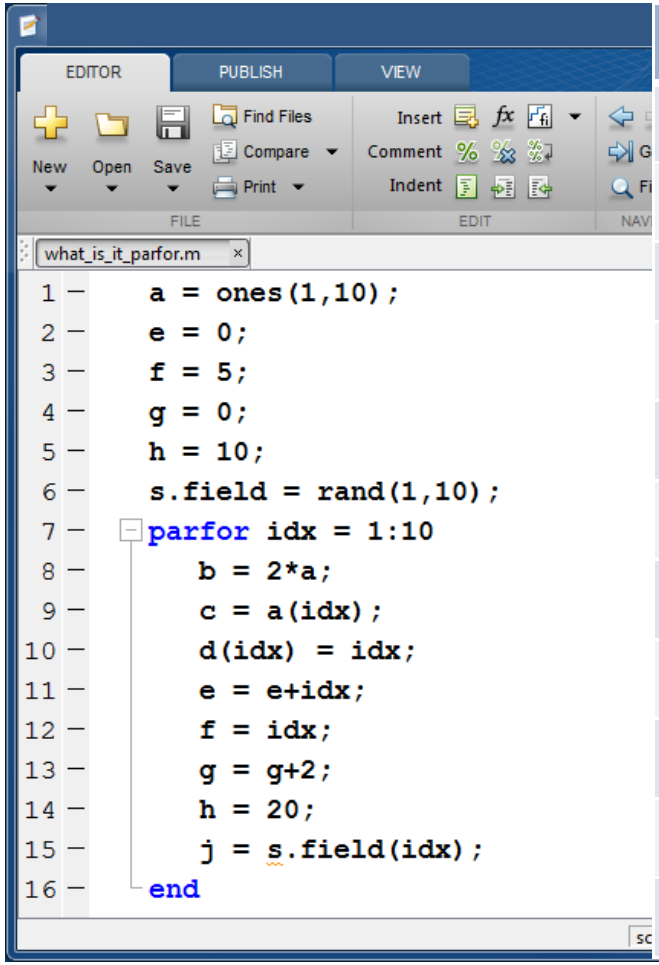
Classification	Description
Loop	Serves as a loop index for arrays
Sliced	An array whose segments are operated on by different iterations of the loop
Broadcast	A variable defined before the loop whose value is used inside the loop, but never assigned inside the loop
Reduction	Accumulates a value across iterations of the loop, regardless of iteration order
Temporary	Variable created inside the loop, but unlike sliced or reduction variables, not available outside the loop

```
>> web([docroot '/distcomp/advanced-topics.html#bq_of7_-1'])
```

Variable Classification Example



After the `for` loop, what is the type and the value of each variable?



Variable	Type	Value
a	broadcast	ones(1:10)
b	temp	undefined
c	temp	undefined
d	sliced	1:10
e	reduction	55
f	temp	5
g	reduction	20
h	temp	10
j	temp	0.0000 + 1.0000i
s	broadcast	rand(1,10)
idx	loop	undefined

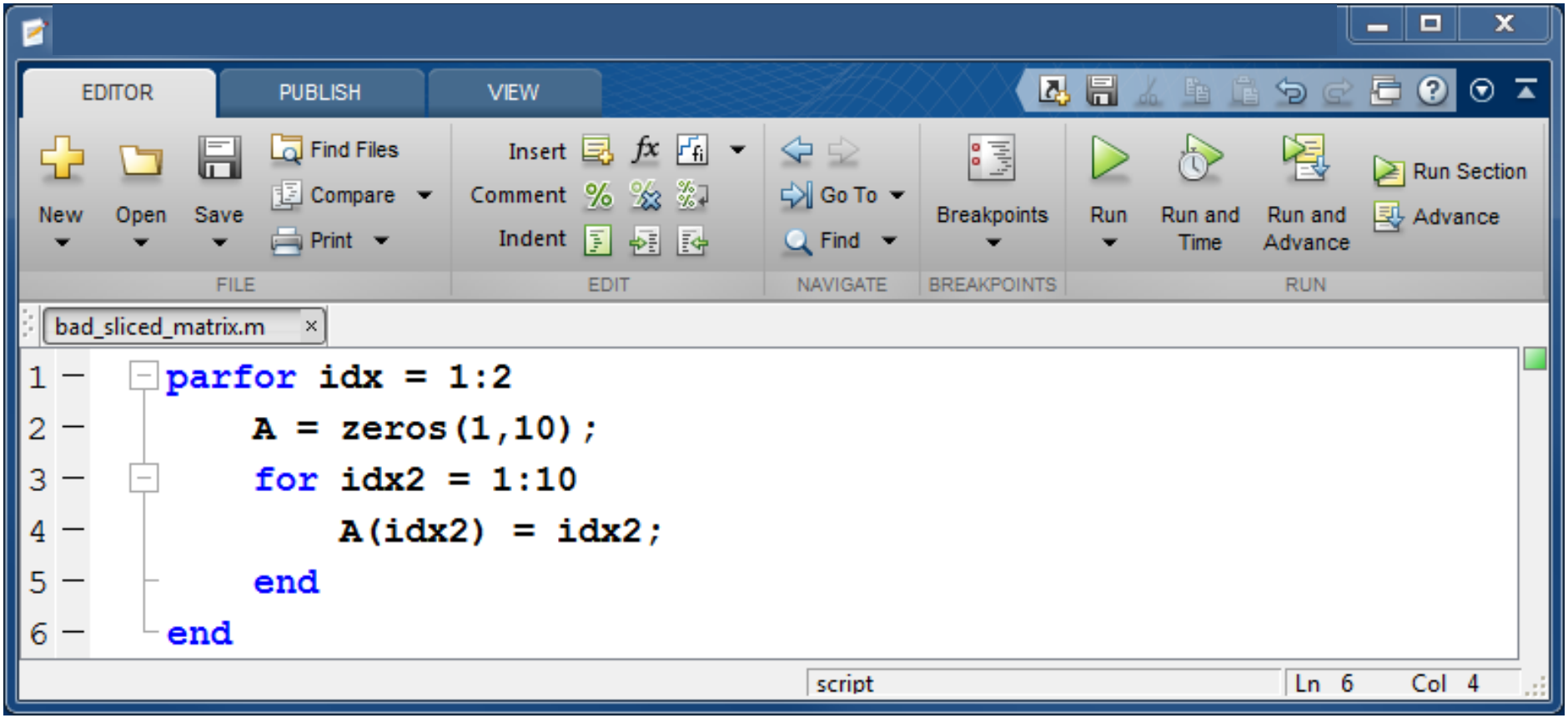
```
>> what_is_it_parfor
```

Sliced Variables

- An indexed variables, parceled out to each worker
  - Indexing at the first level only and for () or {}
  - Within the list of indices for a sliced variable, one of these indices is of the form i, i+k, i-k, k+i, or k-i, where i is the loop variable and k is a constant or a simple (non-indexed) broadcast variable; and every other index is a constant, a simple broadcast variable, colon, or end

Not Valid	Valid
A(i+f(k),j,:,3)	A(i+k,j,:,3)
A(i,20:30,end)	A(i,:,end)
A(i,:,s.field1)	A(i,:,k)

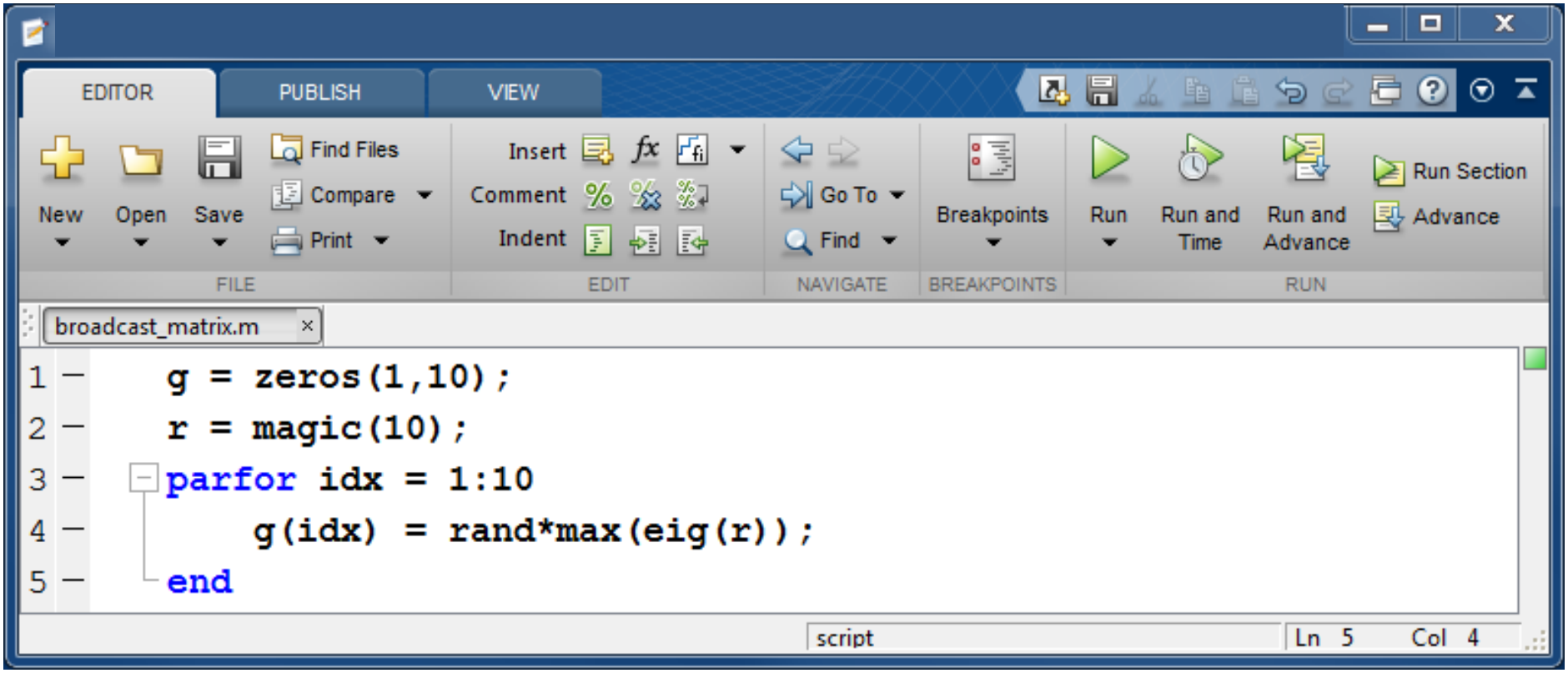
Implications of Sliced Variables



What is the value of `A`?

```
>> bad_sliced_matrix
```

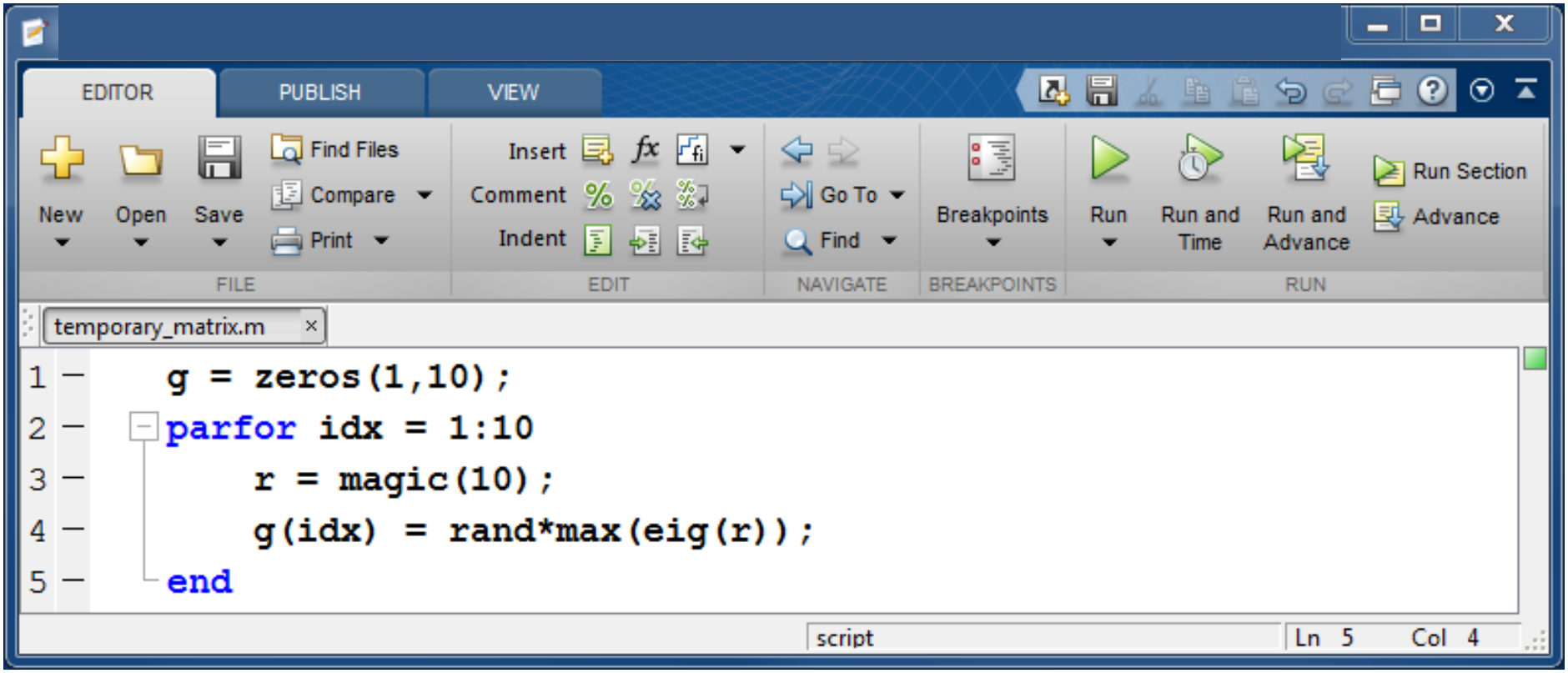
Implications of Broadcast Variables



The entire data set `r` is broadcast to each worker

```
>> broadcast_matrix
```

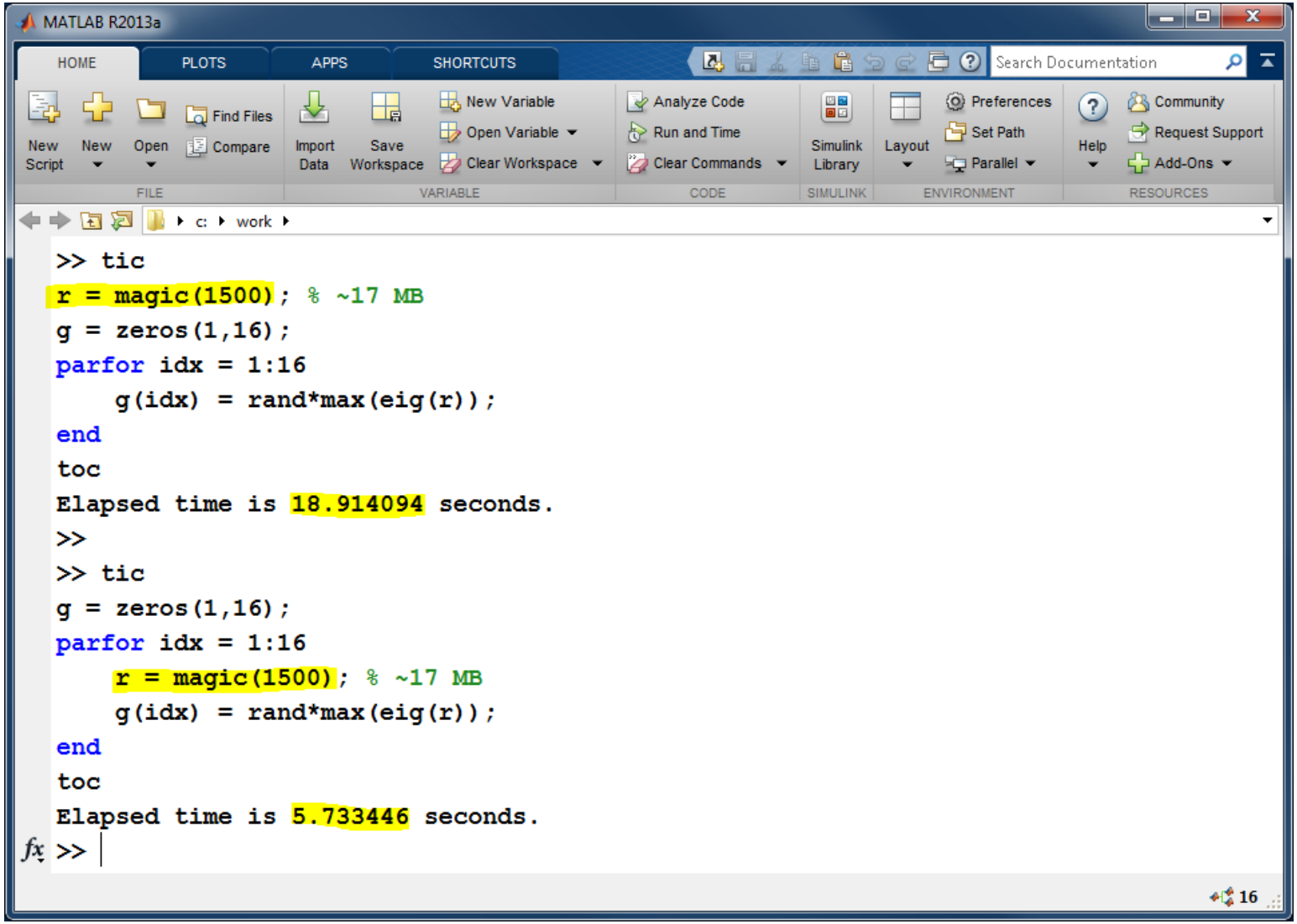
Implications of Broadcast Variables



Could you create `r` on the workers instead?

```
>> temporary_matrix
```

Implications of Broadcast Variables



Implications of Reductions Variables

- Variable appears on both sides of assignment
- Same operation must be performed on variable for all iterations
- Reduction function must be associative and commutative

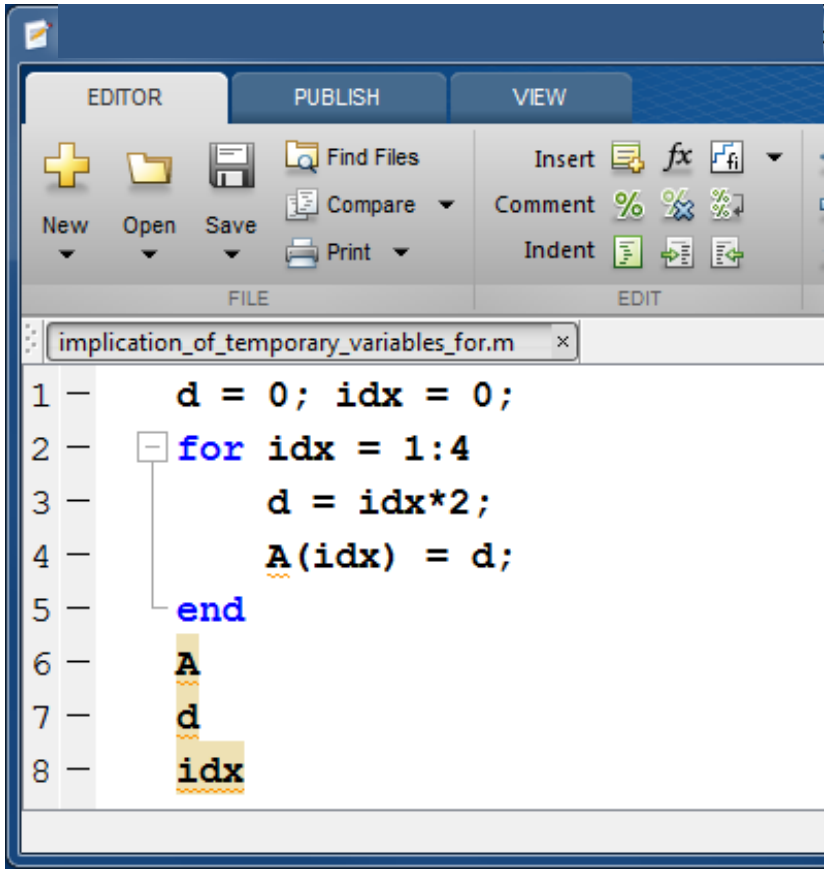
Implications of Reduction Variables

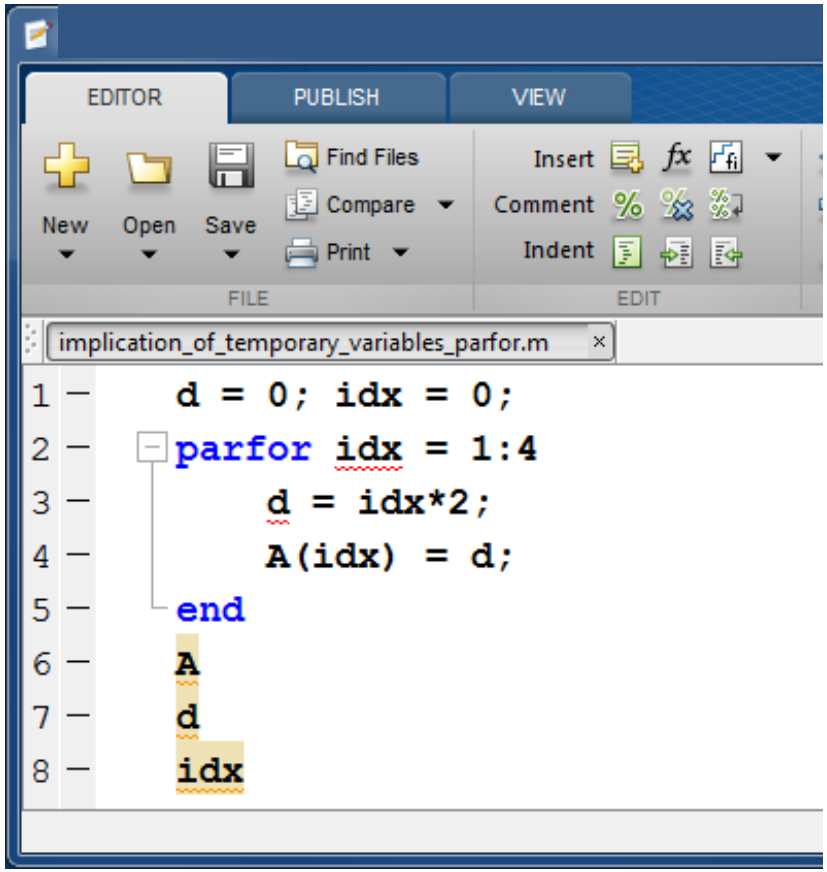
```
>> x = 0;
parfor idx = 1:10
    x = x+idx;
end
x
```

```
>> x2 = [];
parfor idx = 1:10
    x2 = [x2 idx];
end
x2
```

```
>> x3 = 0;
parfor idx = 1:32
    if idx<16
        x3 = x3*idx;
    else
        x3 = x3+idx;
    end
end
x3
Error: Different reduction functions are used for the same variable x3.
See Parallel for Loops in MATLAB, "Basic Rules for Reduction Variables".
```

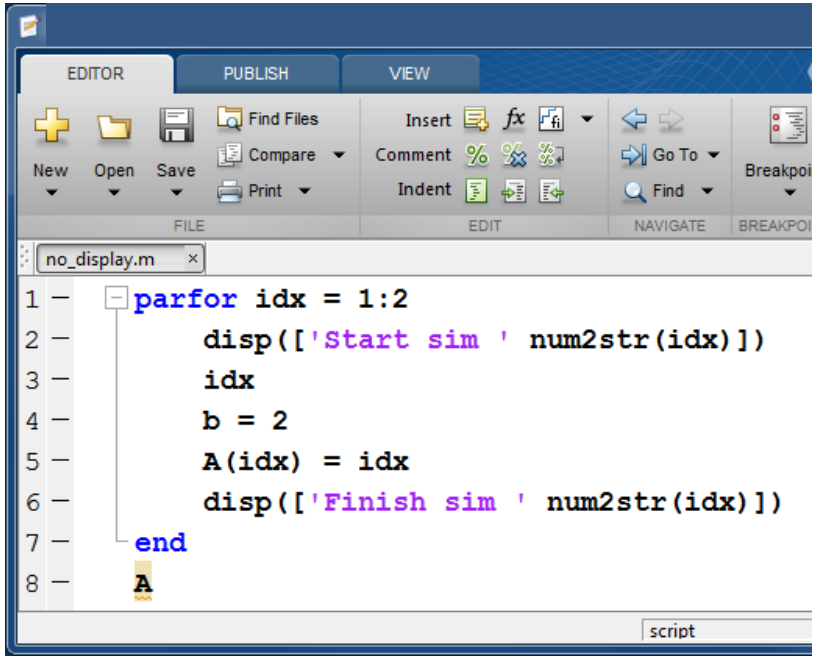
Implications of Temporary Variables

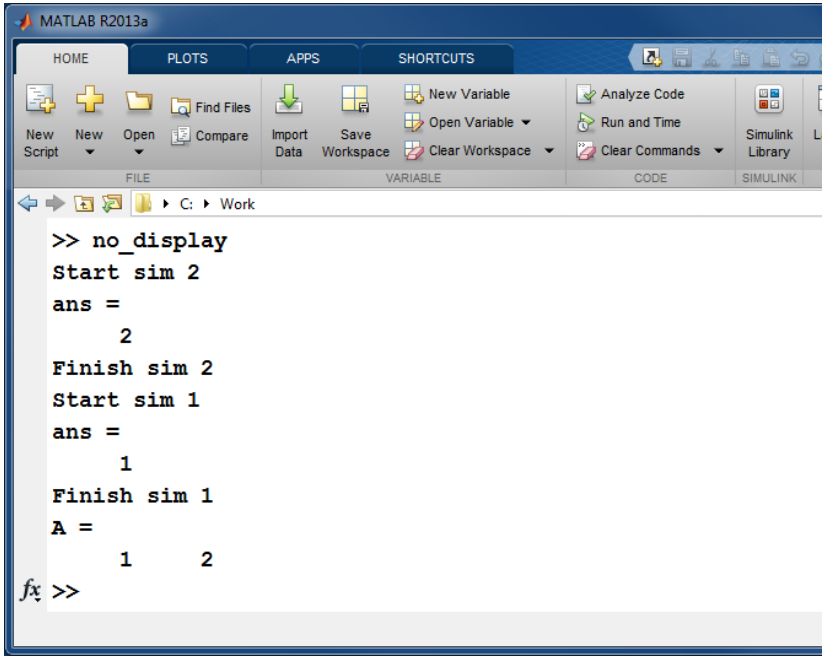




What is the value of `A`? `d`? `idx`?

Variable Assignments Are Not Displayed When Running a `parfor`



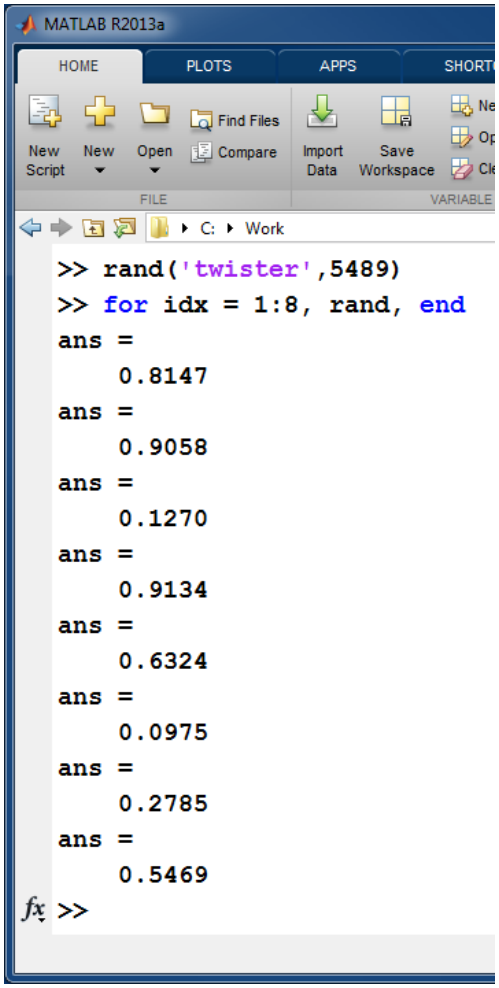


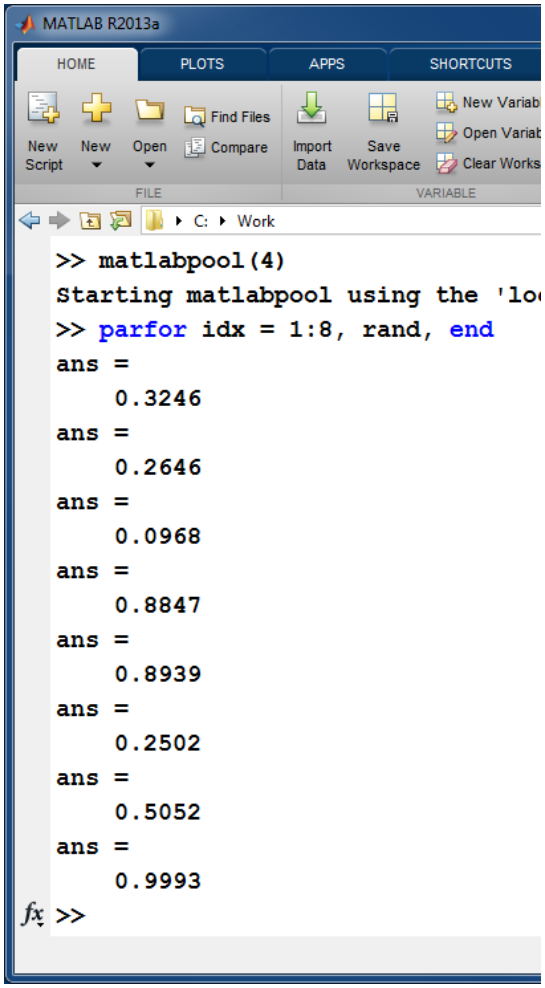

```
>> no_display
```

`rand` in `parfor` Loops (1)

- MATLAB has a repeatable sequence of random numbers
- When workers are started up, rather than using this same sequence of random numbers, the `labindex` is used to seed the RNG

`rand` in `parfor` Loops (2)



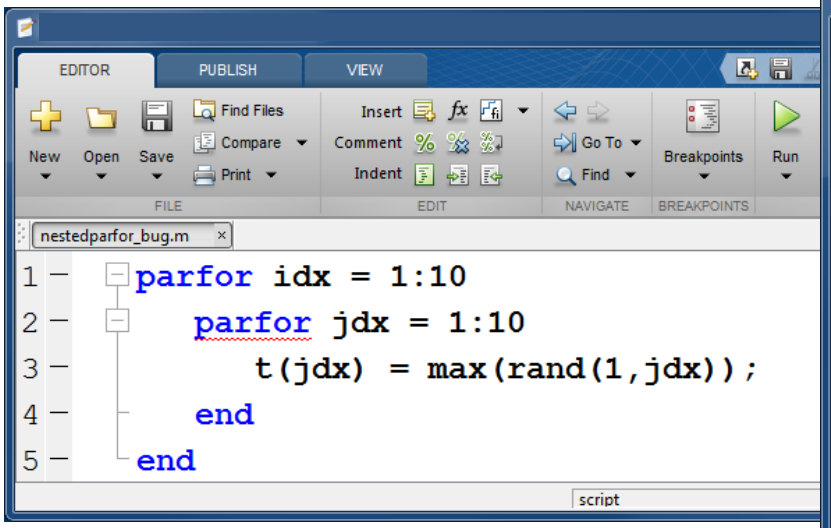


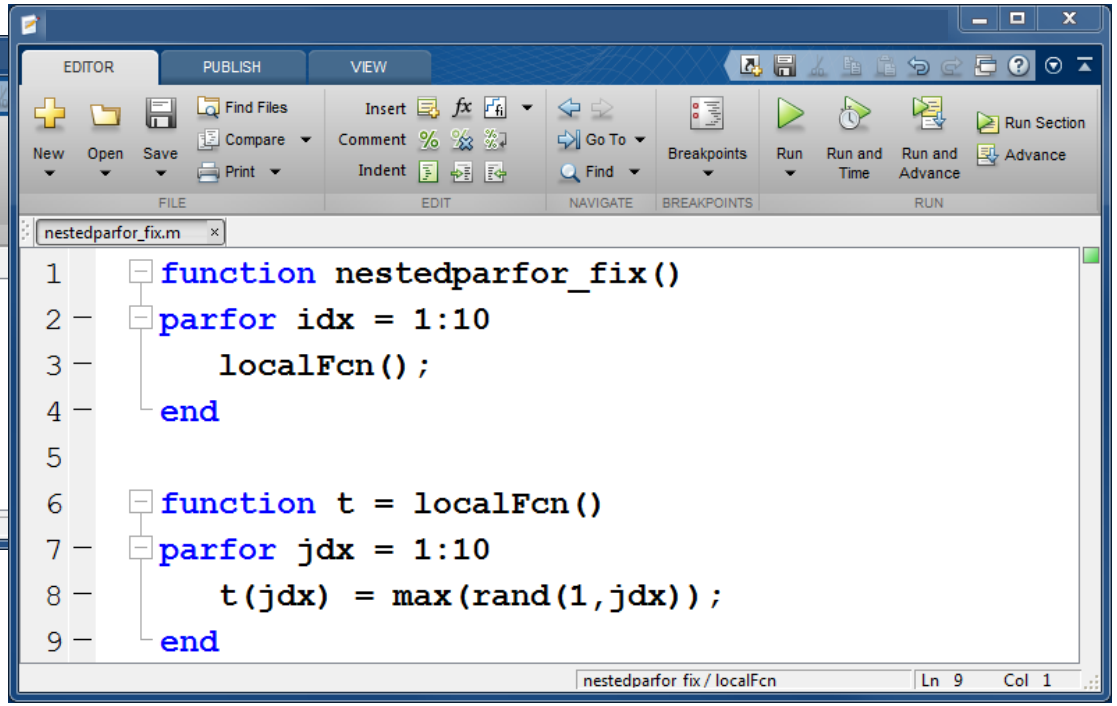
Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting

What If My `parfor` Has a `parfor` In It?

- MATLAB runs a static analyzer on the immediate `parfor` and will error out nested `parfor` loops. However, functions called from within the `parfor` that include `parfor` loops are treated as regular `for` loops

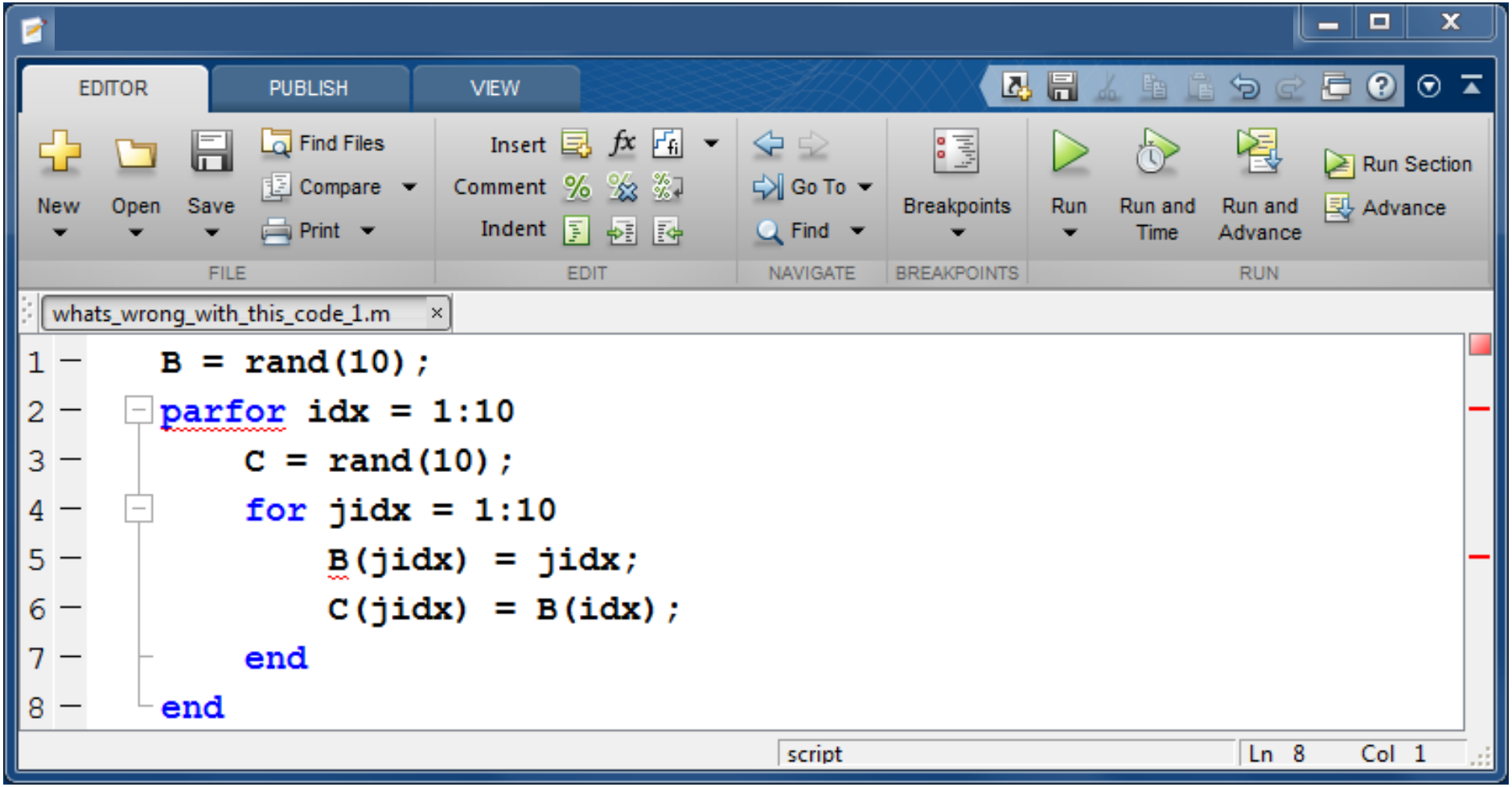





```
>> nestedparfor_bug
>> nestedparfor_fix
```

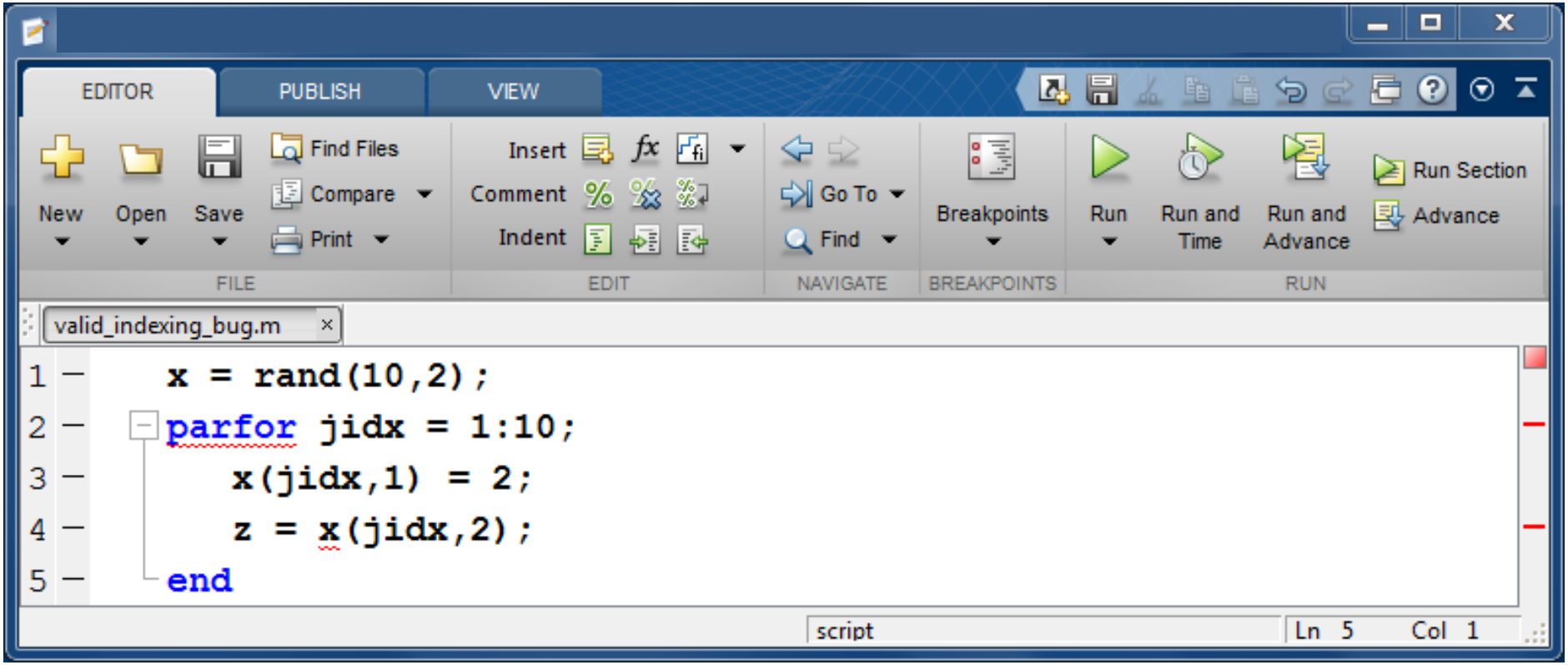
What's Wrong With This Code?

Why can we index into `C` with `jidx`, but not `B`?




```
>> whats_wrong_with_this_code
```

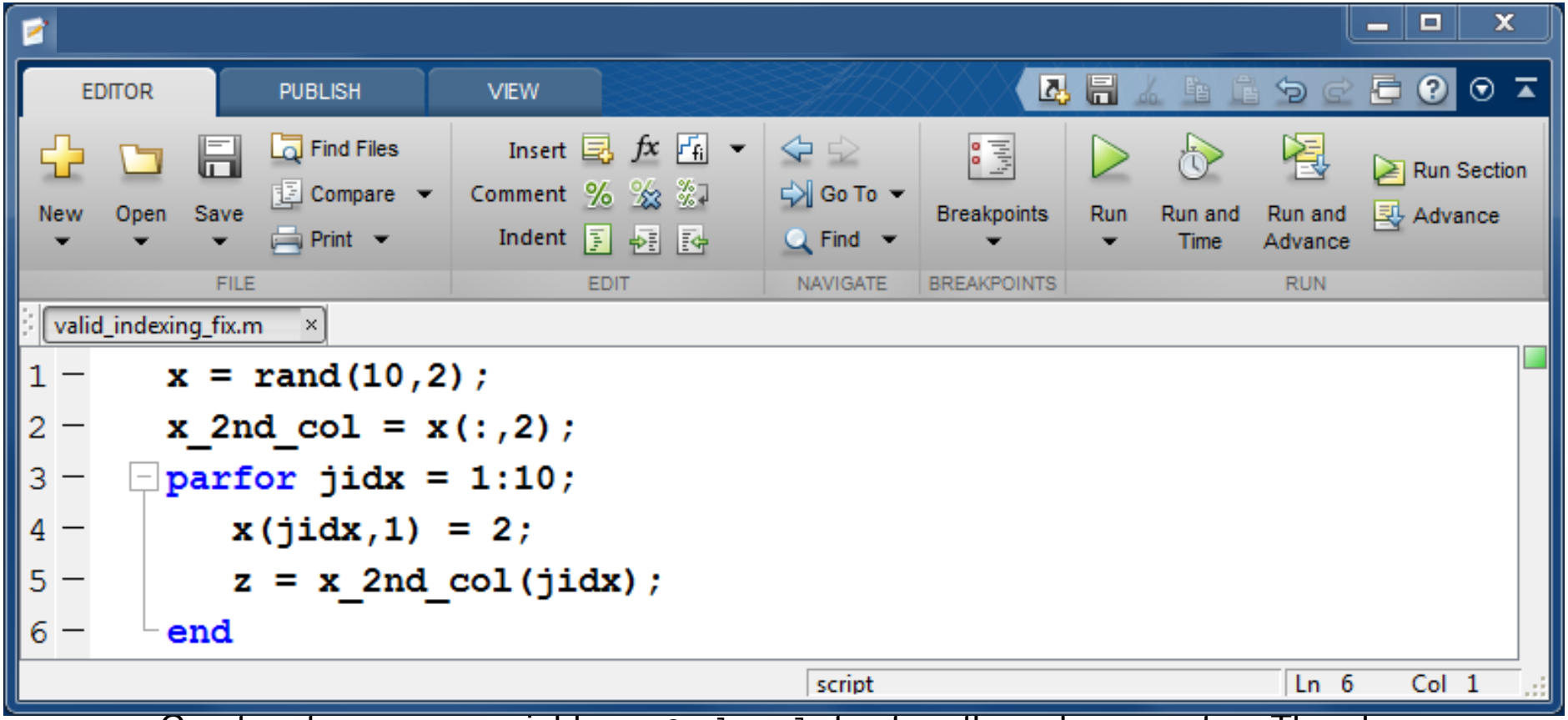
`parfor` issue: Indexing With Different Expressions



How can we avoid indexing into `x` two different ways?

```
>> valid_indexing_bug
```

`parfor` issue: Solution

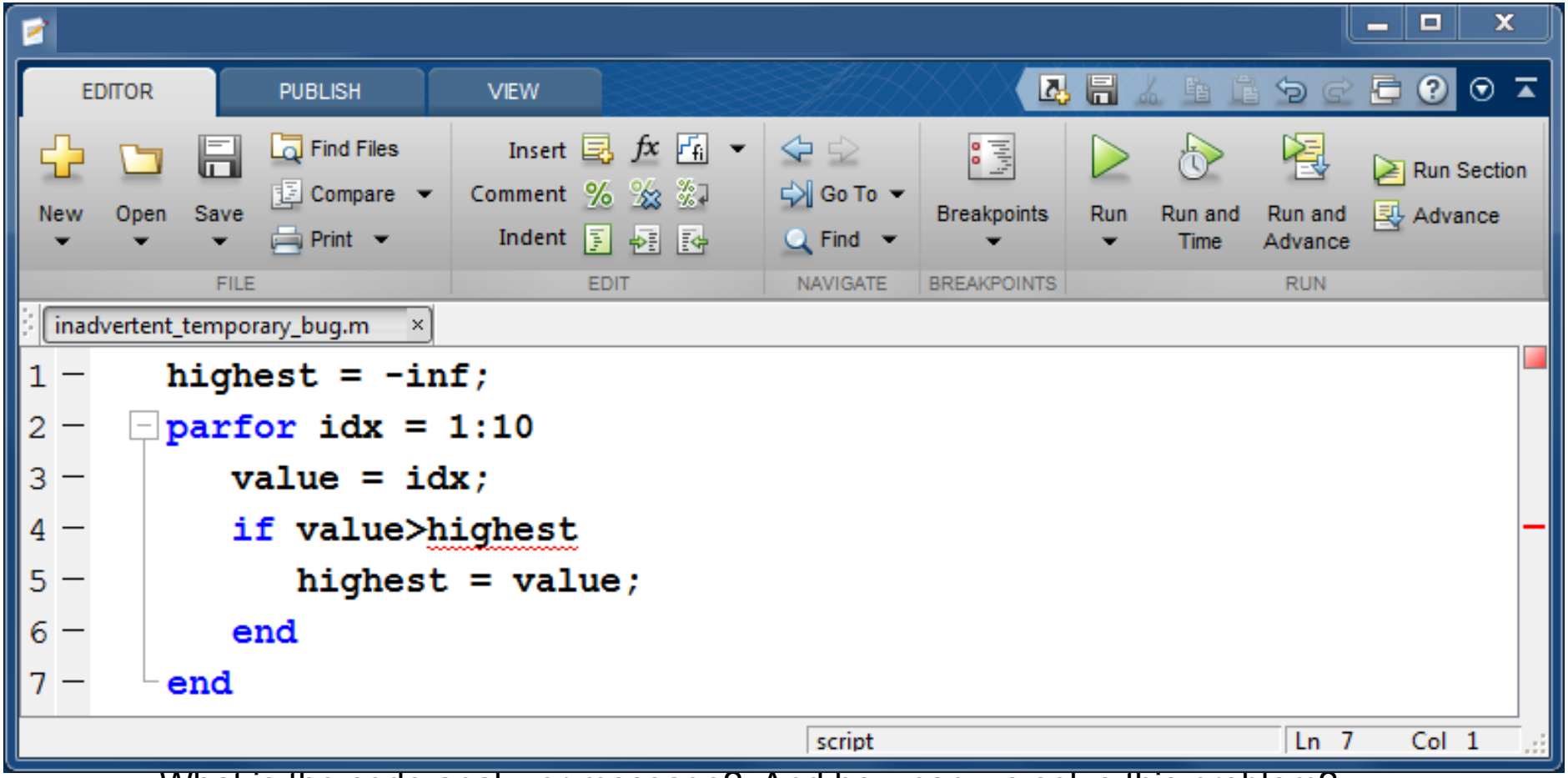


Create a temporary variable, `x_2nd_col`, to store the column vector. Then loop into the vector using the looping index, `jidx`, rather than the into a matrix.

Note: This doesn't scale very well if we needed to index into `x` many ways.

```
>> valid_indexing_fix
```

`parfor` issue: Inadvertently Creating Temporary Variables

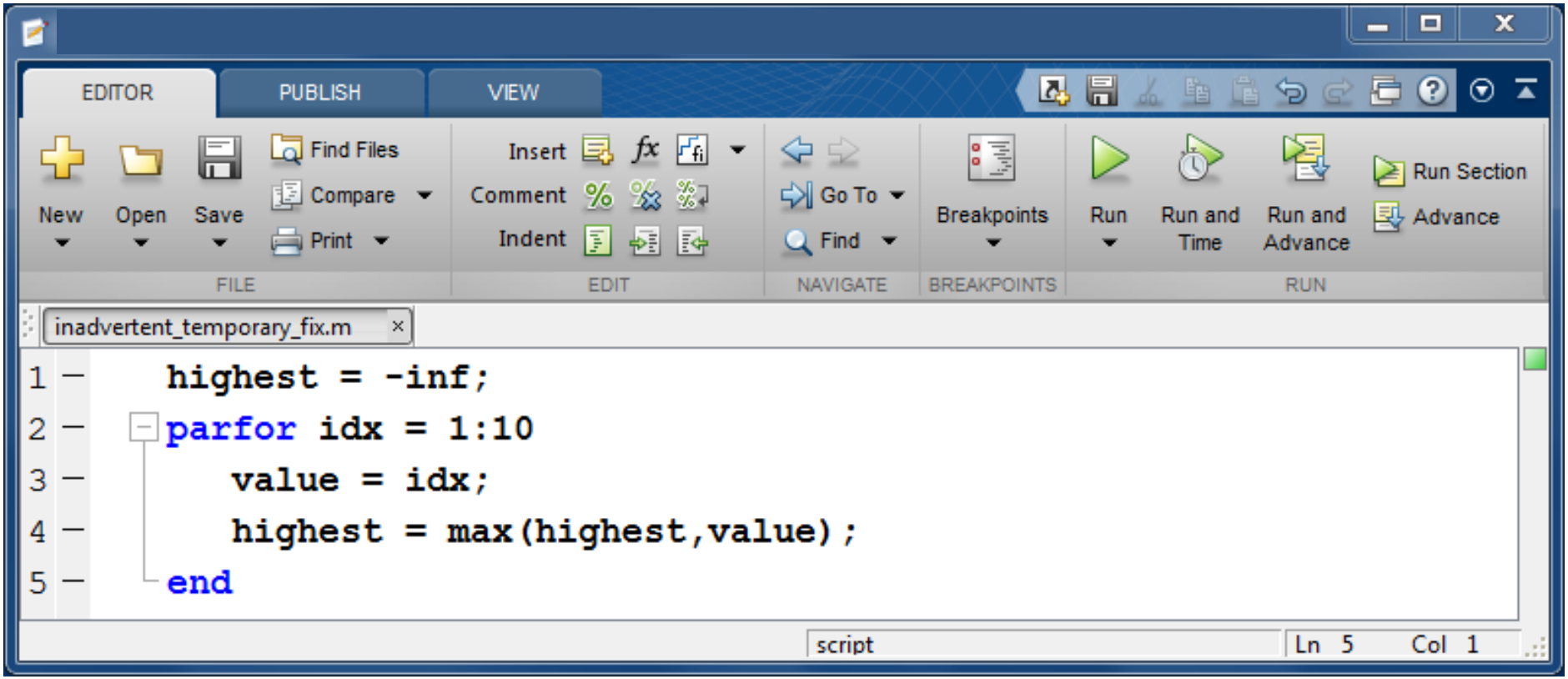


What is the code analyzer message? And how can we solve this problem?

Why does the code analyzer think `highest` is a temporary variable?

```
>> inadvertent_temporary_bug
```

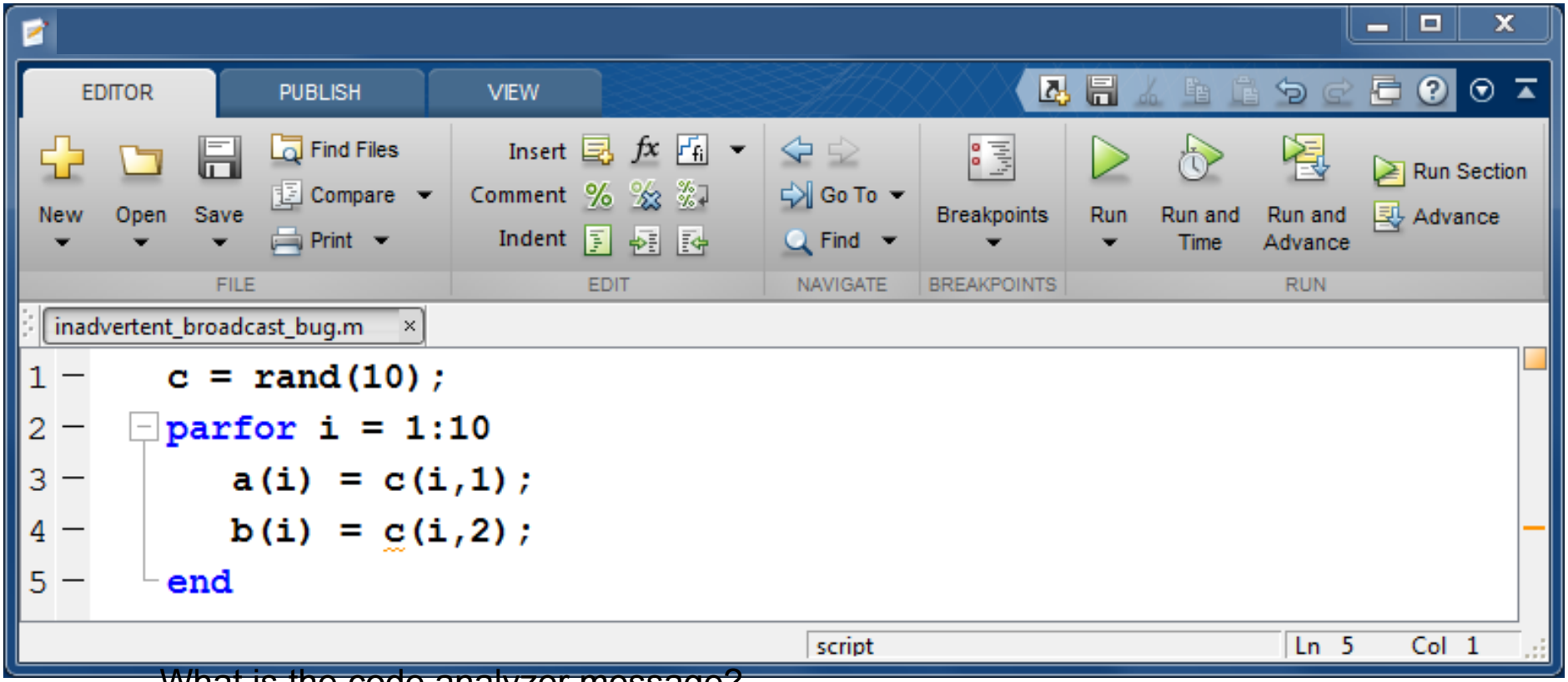
`parfor` issue: Solution



Assign `highest` to the result of a reduction function

```
>> inadvertent_temporary_fix
```

`parfor` issue: Inadvertently Creating Broadcast Variables



What is the code analyzer message?

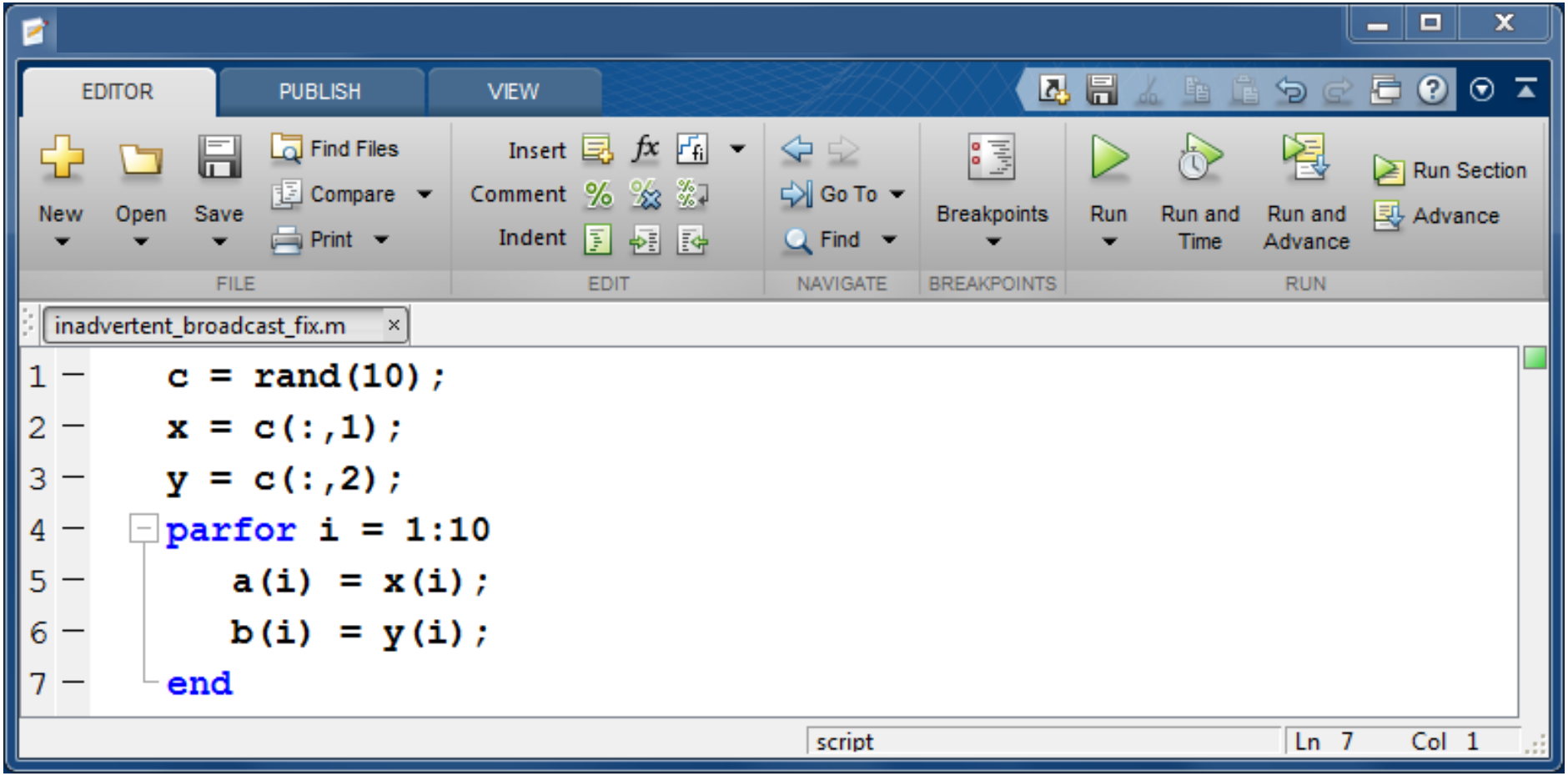
Why isn't `c` a sliced variable? What kind is it?

How can we make it sliced?

If we didn't have the `b` assignment, would `c` be sliced?

```
>> inadvertent_broadcast_bug
```

`parfor` issue: Solution



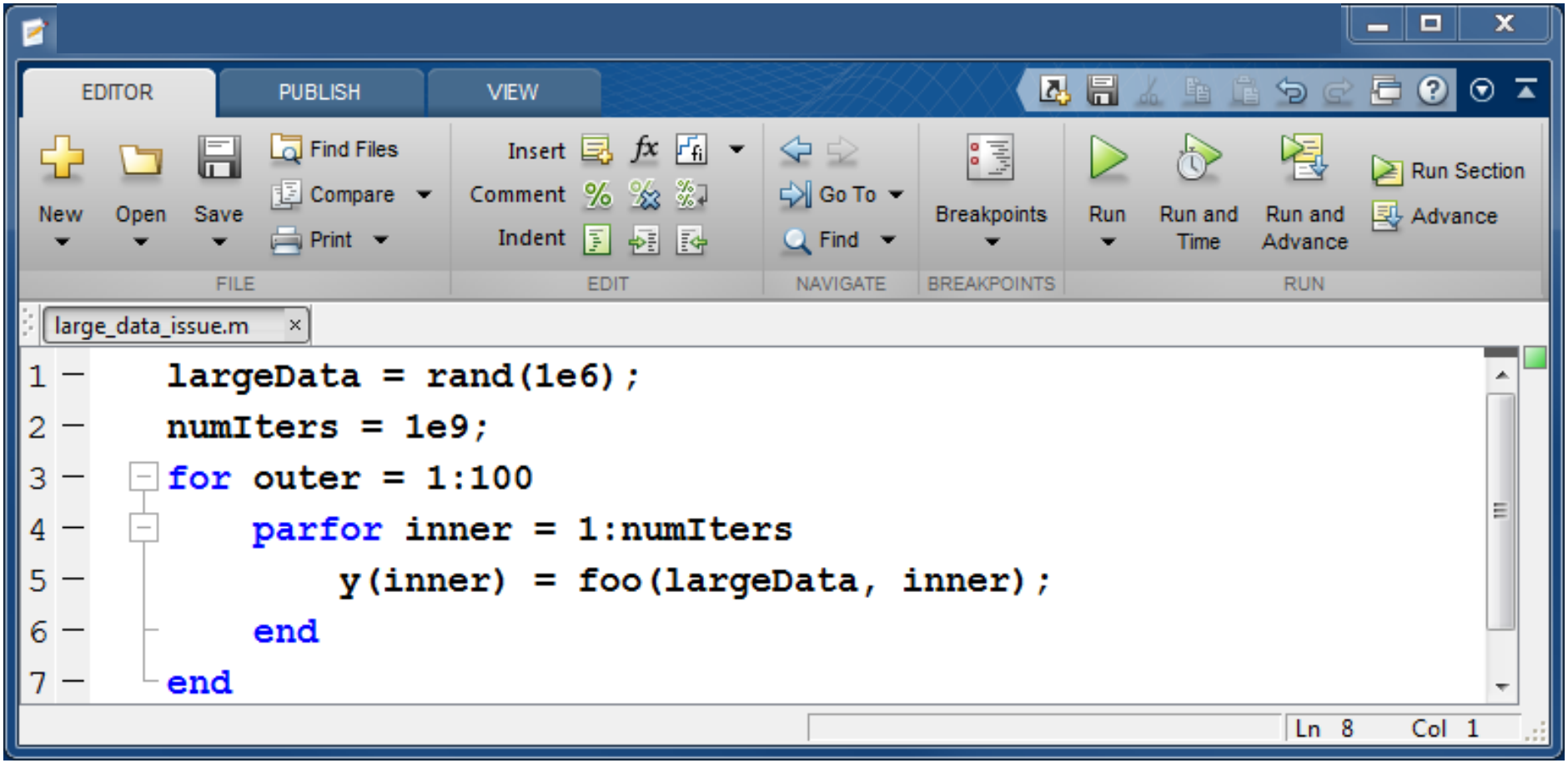
Create the additional variables `x` and `y`, which are sliced

```
>> inadvertent_broadcast_fix
```

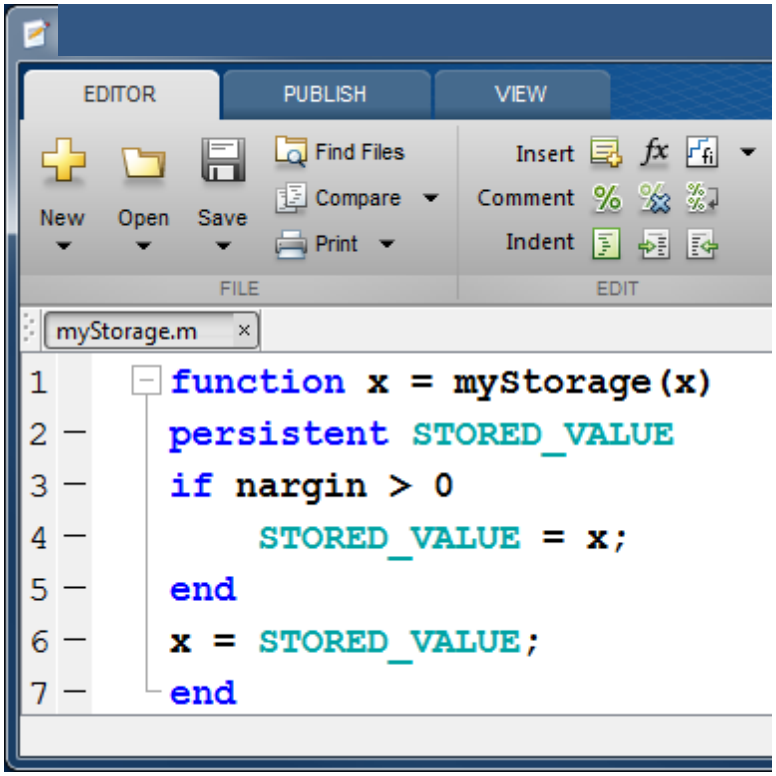
Persistent Storage (1)

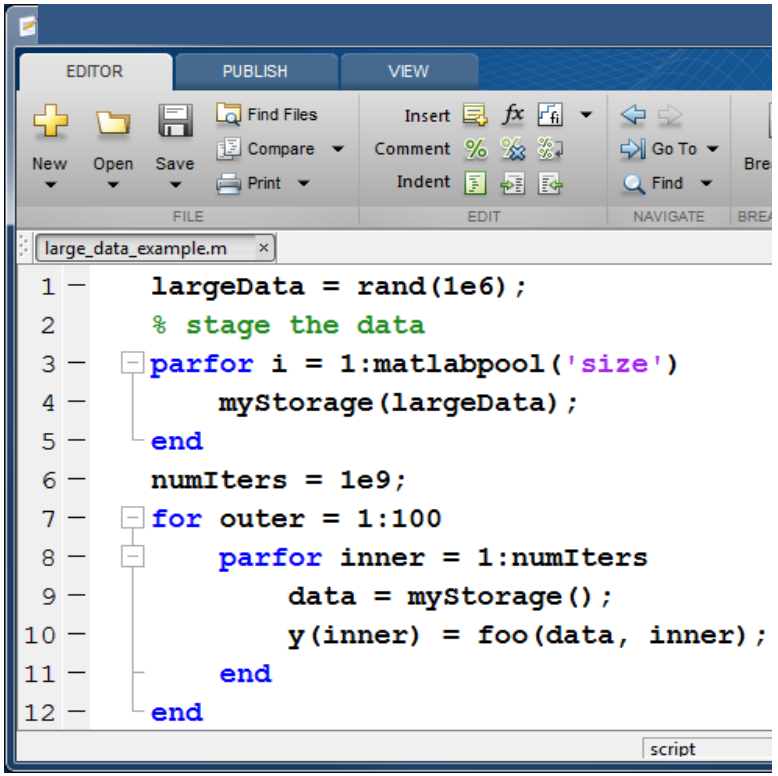
- I cannot convert the outer loop into `parfor` because it's in someone else's top level function. However, if I convert the inner loop into `parfor` in the straightforward manner, we end up sending large data to the workers N times.

Persistent Storage (2)



Solution: Persistent Storage





Store the value in a persistent variable in a function

Best Practices for Converting `for` to `parfor`

- Use code analyzer to diagnose `parfor` issues
- If your `for` loop cannot be converted to a `parfor`, consider wrapping a subset of the body to a function
- If you modify your `parfor` loop, switch back to a for loop for regression testing
- Read the section on classification of variables

```
>> docsearch 'Classification of Variables'
```

Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting

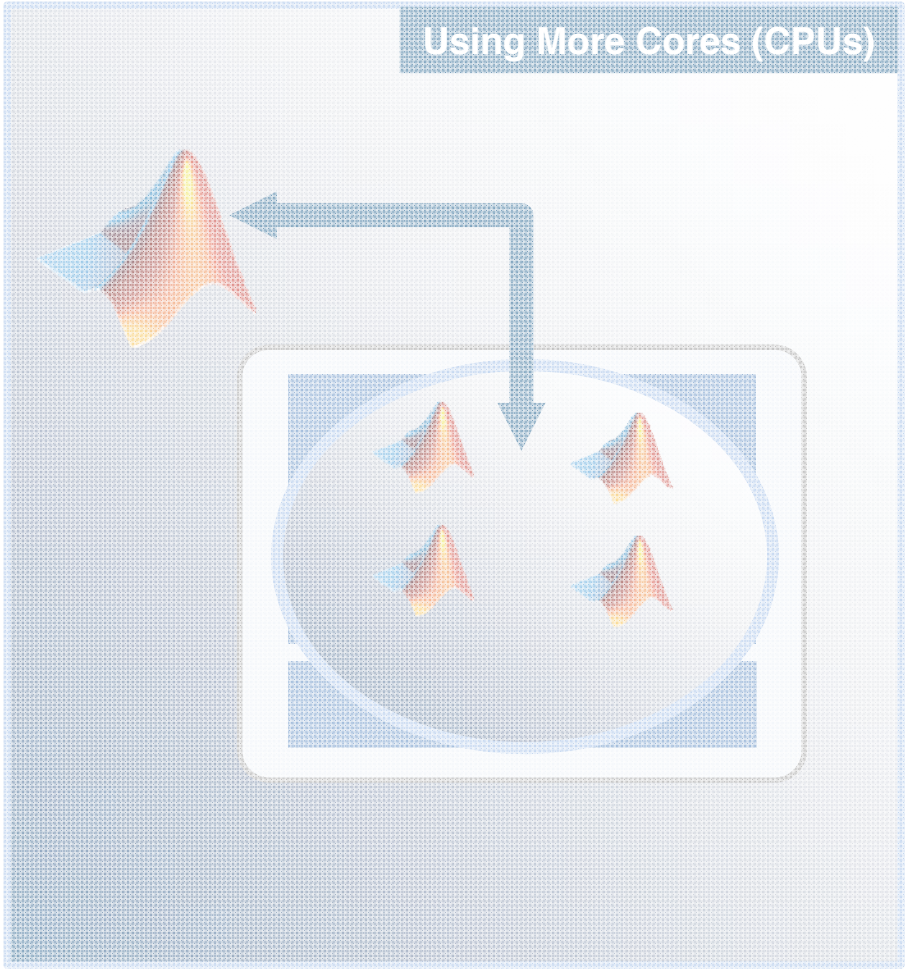
What is a Graphics Processing Unit (GPU)

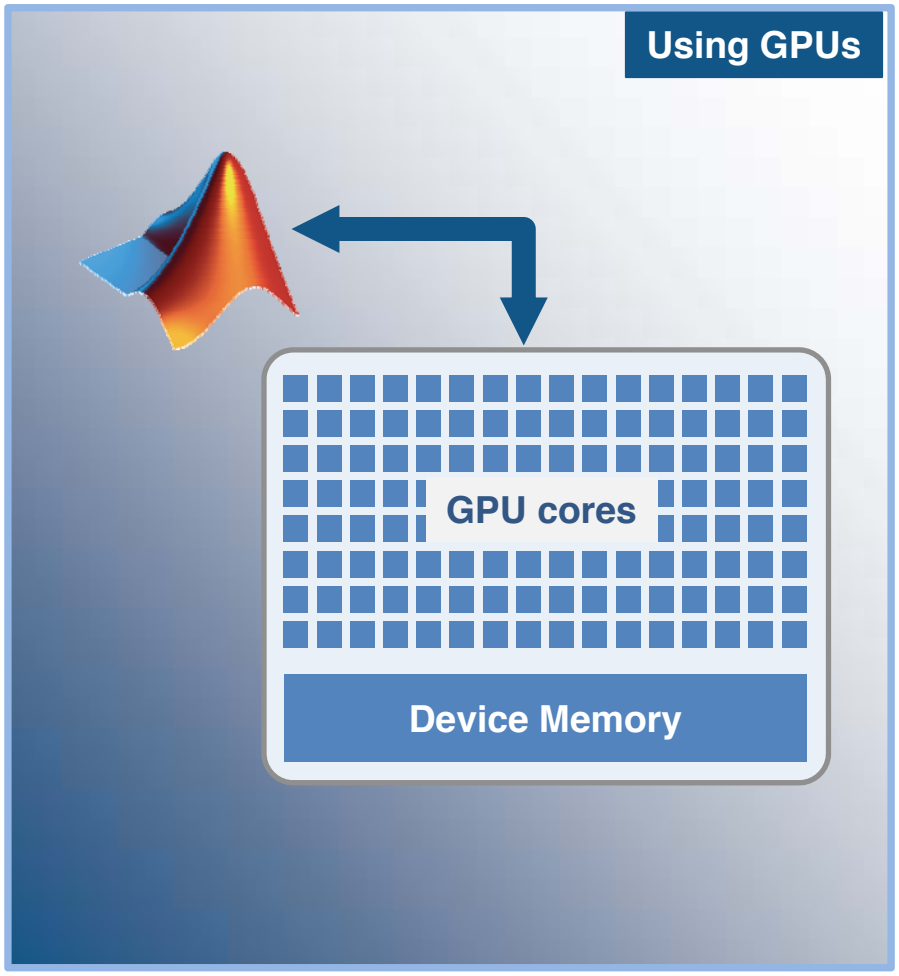
- Originally for graphics acceleration, now also used for scientific calculations
- Massively parallel array of integer and floating point processors
  - Typically hundreds of processors per card
  - GPU cores complement CPU cores
- Dedicated high-speed memory

- blogs.mathworks.com/loren/2013/06/24/running-monte-carlo-simulations-on-multiple-gpus

* Parallel Computing Toolbox requires NVIDIA GPUs with Compute Capability 1.3 or higher, including NVIDIA Tesla 20-series products. See a complete listing at www.nvidia.com/object/cuda_gpus.html

Performance Gain with More Hardware





Programming Parallel Applications (GPU)

- Built-in support with Toolboxes

Programming Parallel Applications (GPU)

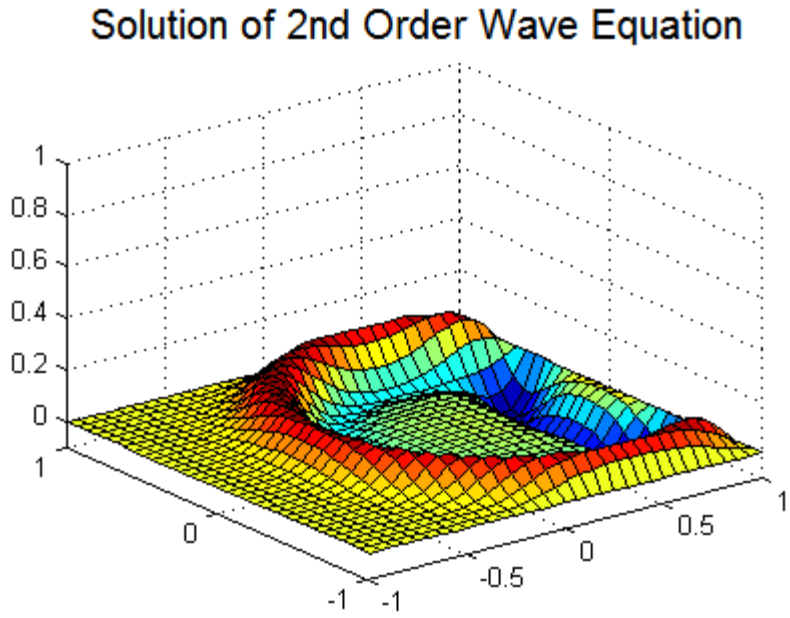
- Built-in support with Toolboxes
- Simple programming constructs:
  `gpuArray, gather`

Example: Solving 2D Wave Equation

GPU Computing

- Solve 2nd order wave equation using spectral methods:

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

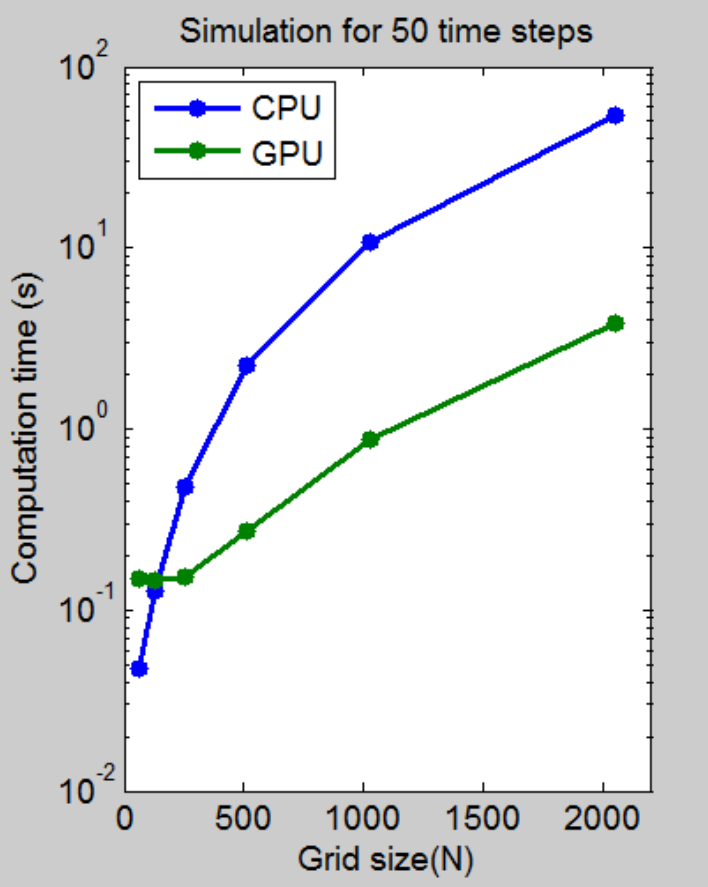


- Run both on CPU and GPU
- Using `gpuArray` and overloaded functions

www.mathworks.com/help/distcomp/using-gpuarray.html#bsloua3-1

Benchmark: Solving 2D Wave Equation

CPU vs GPU



Grid Size	CPU (s)	GPU (s)	Speedup
64 x 64	0.05	0.15	0.32
128 x 128	0.13	0.15	0.88
256 x 256	0.47	0.15	3.12
512 x 512	2.22	0.27	8.10
1024 x 1024	10.80	0.88	12.31
2048 x 2048	54.60	3.84	14.22

Intel Xeon Processor W3690 (3.47GHz),
NVIDIA Tesla K20 GPU

Programming Parallel Applications (GPU)

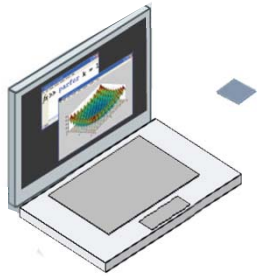
- Built-in support with Toolboxes
- Simple programming constructs:
  `gpuArray, gather`
- Advanced programming constructs:
  `arrayfun, bsxfun, spmd`
- Interface for experts:
  `CUDAKernel, MEX support`

www.mathworks.com/help/releases/R2013a/distcomp/executing-cuda-or-ptx-code-on-the-gpu.html

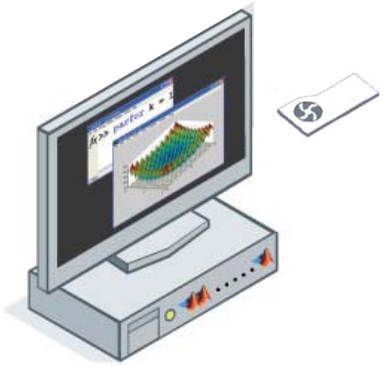
www.mathworks.com/help/releases/R2013a/distcomp/create-and-run-mex-files-containing-cuda-code.html

GPU Performance – not all cards are equal

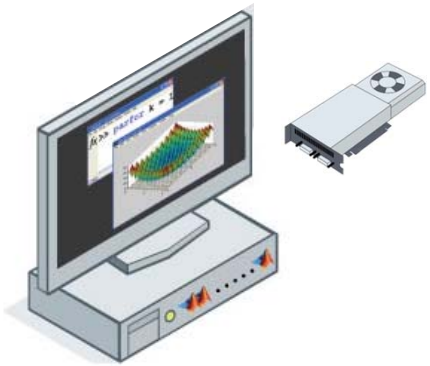
- Tesla-based cards will provide best performance
- Realistically, expect 4x to 15x speedup (Tesla) vs CPU
- See GPUBench on MATLAB Central for examples
  www.mathworks.com/matlabcentral/fileexchange/34080-gpubench



Laptop GPU
GeForce



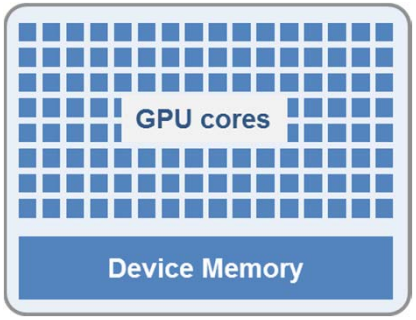
Desktop GPU
GeForce / Quadro



High Performance Computing GPU
Tesla / Quadro

Criteria for Good Problems to Run on a GPU

- **Massively parallel:**
  - Calculations can be broken into hundreds or thousands of independent units of work
  - Problem size takes advantage of many GPU cores



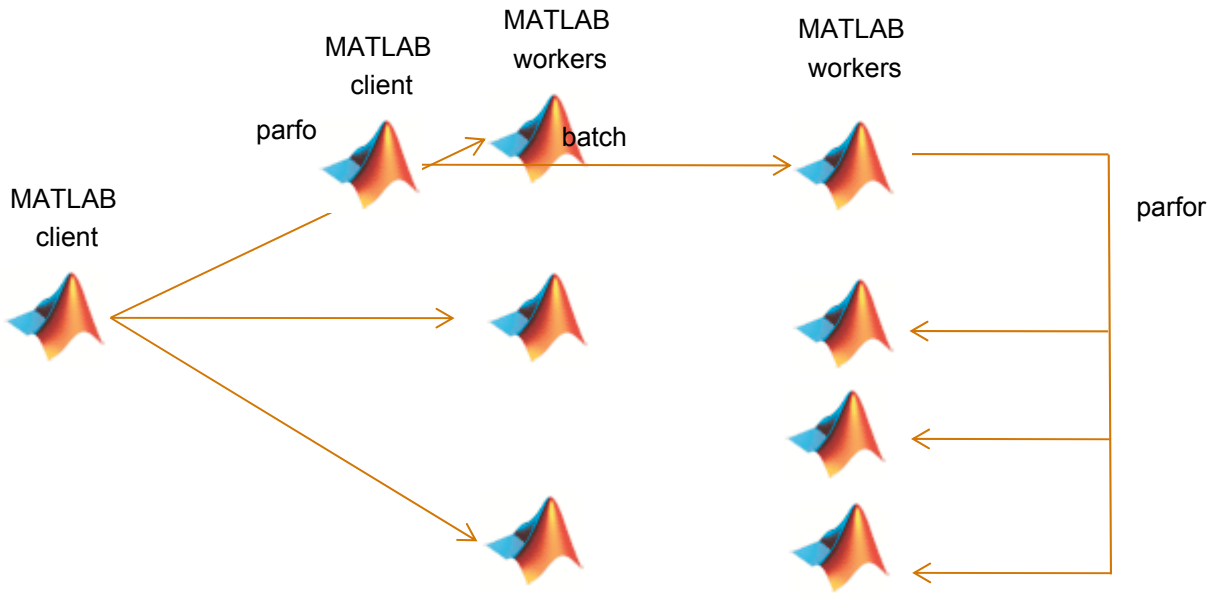
- **Computationally intensive:**
  - Computation time significantly exceeds CPU/GPU data transfer time

- **Algorithm consists of supported functions:**
  - Growing list of Toolboxes with built-in support
    - www.mathworks.com/products/parallel-computing/builtin-parallel-support.html
  - Subset of core MATLAB for `gpuArray`, `arrayfun`, `bsxfun`
    - www.mathworks.com/help/distcomp/using-gpuarray.html#bsloua3-1
    - www.mathworks.com/help/distcomp/execute-matlab-code-elementwise-on-a-gpu.html#bsnx7h8-1

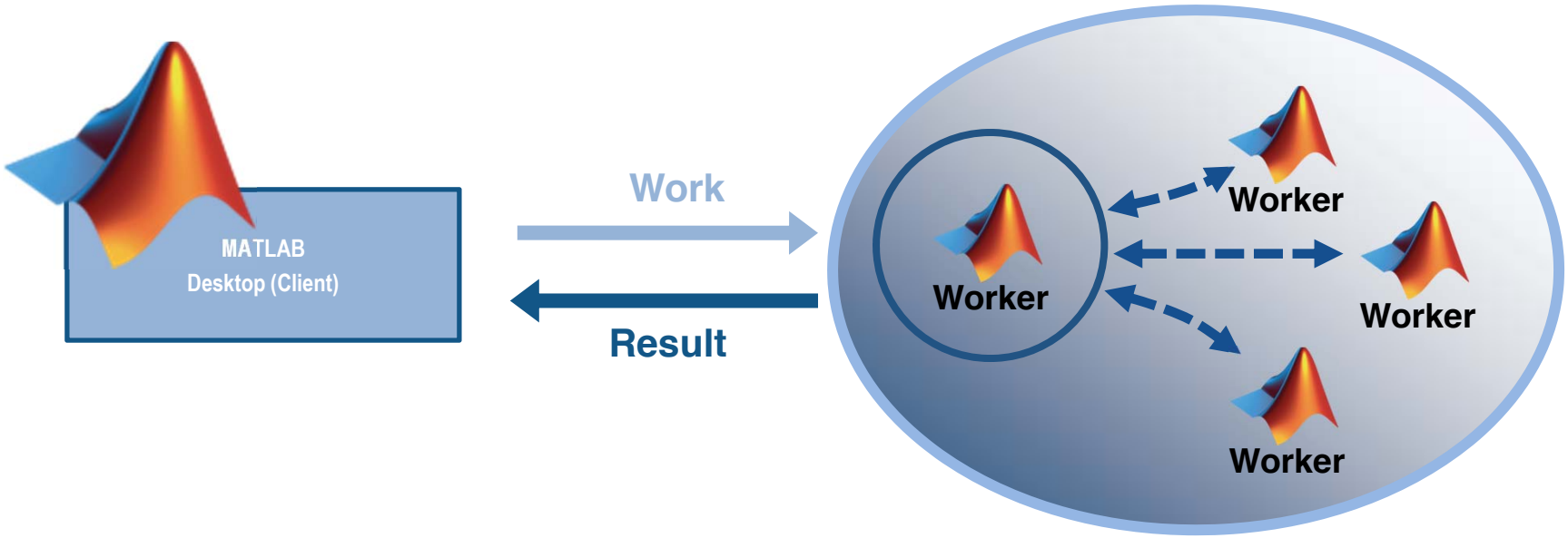
Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting

Migrating from Local to Cluster



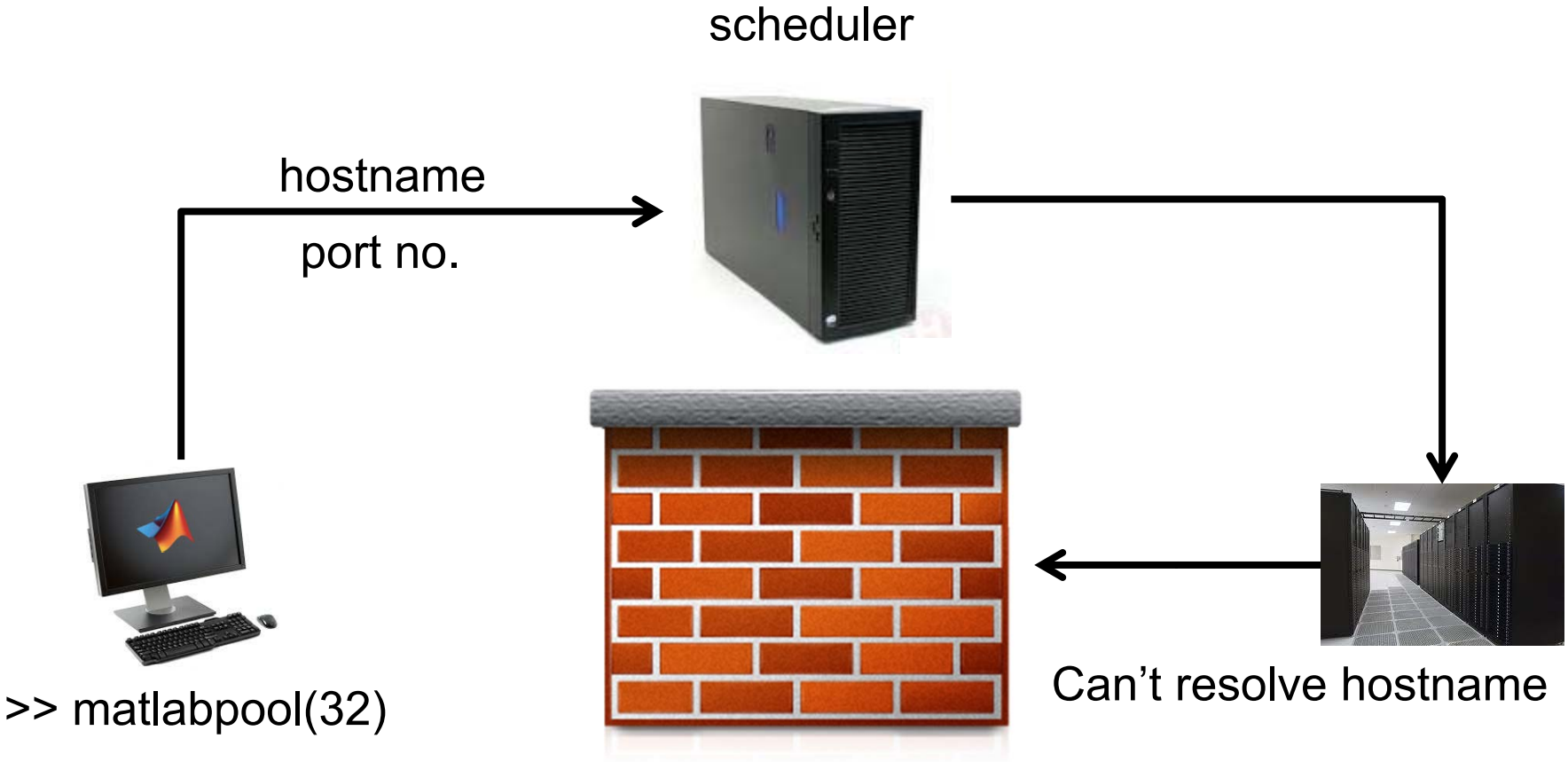
Offload Computations with `batch`



Can't I Just Use `matlabpool` to Connect to the Cluster/Cloud?

- MATLAB pool
  - So long as the compute nodes can reach back to your local desktop, then yes, you can run jobs on the cluster using `matlabpool`
  - Recall, the MATLAB Client is blocked
  - Cannot run other parallel jobs
  - Consumes MDCS licenses while the pool is open, even if they aren't being used
- Batch
  - Ideal if:
    - the local desktop is not reachable from the cluster, or
    - if I want shutdown my desktop, or
    - if I want submit multiple jobs at once

Why Can't I Open a MATLAB Pool to the Cluster?

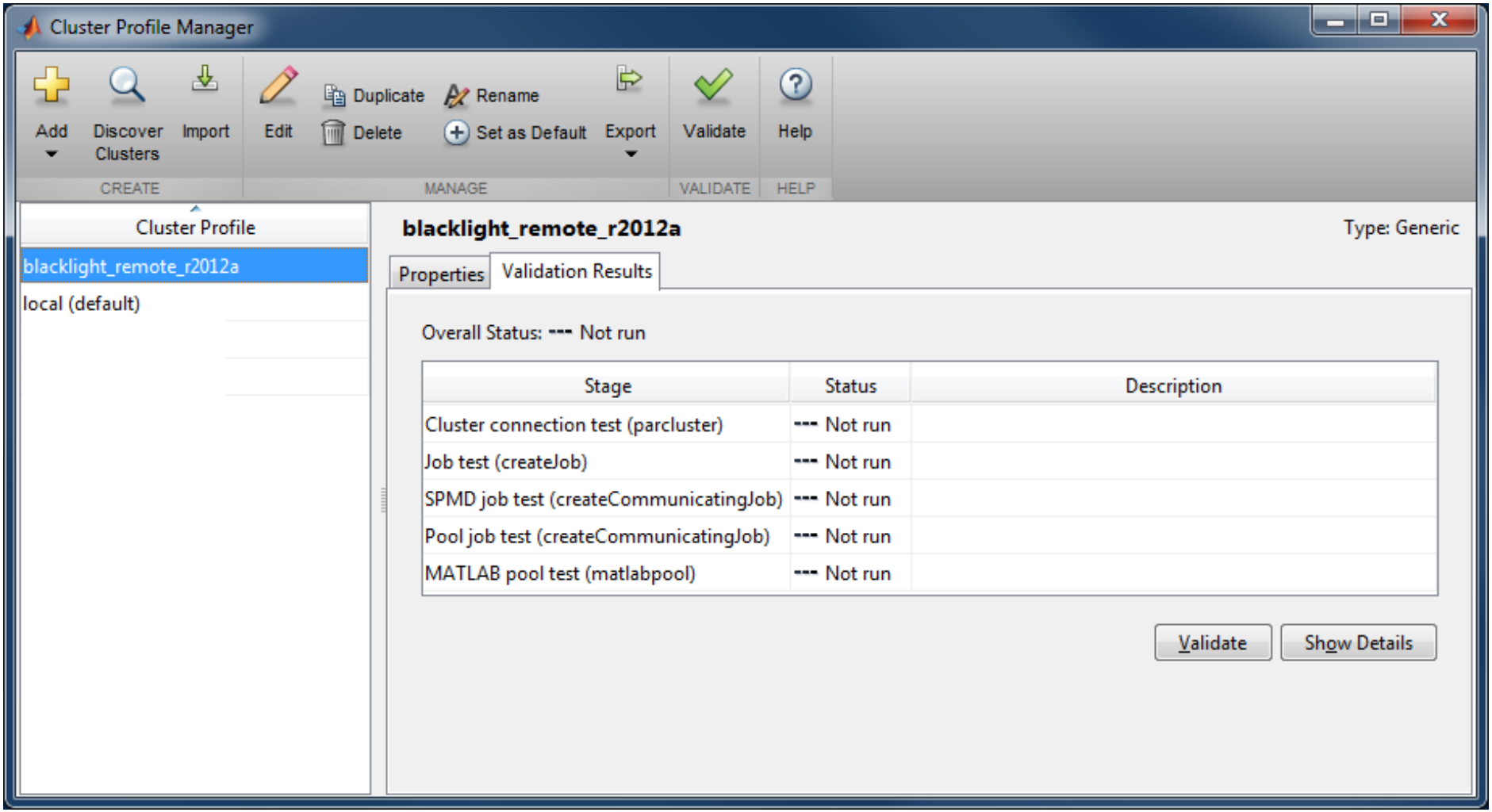



```
>> pctconfig('hostname','12.34.56.78')
```

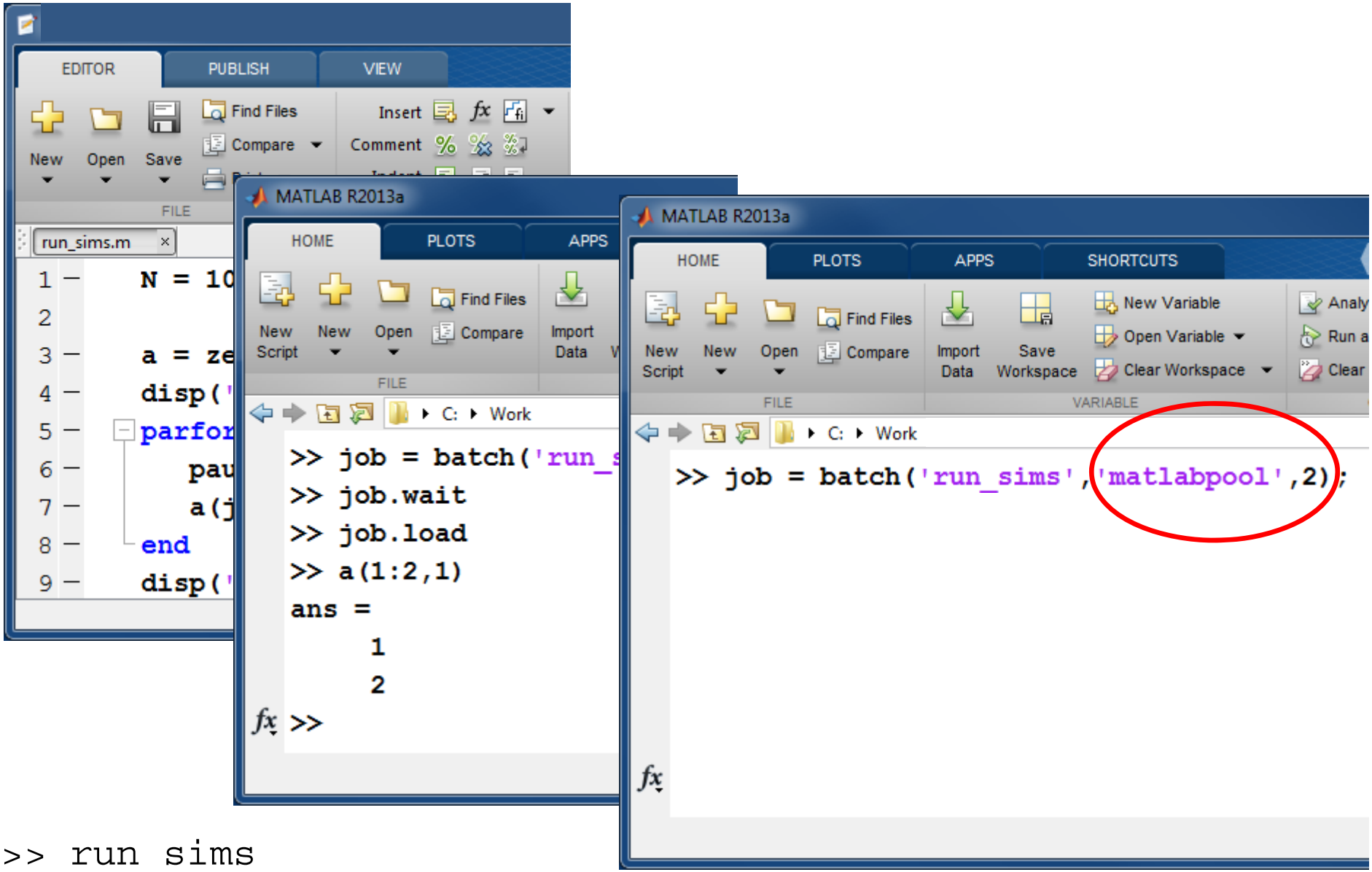
Profiles

- Think of cluster profiles like printer queue configurations
- Managing profiles
  - Typically created by Sys Admins
  - Label profiles based on the version of MATLAB
    - E.g. *hpcc_local_r2013a*
- Import profiles generated by the Sys Admin
  - Don't modify them with two exceptions
    - Specify the JobStorageLocation
    - Setting the ClusterSize
- Validate profiles
  - Ensure new profile is properly working
  - Helpful when debugging failed jobs

Import and Validating a Profile

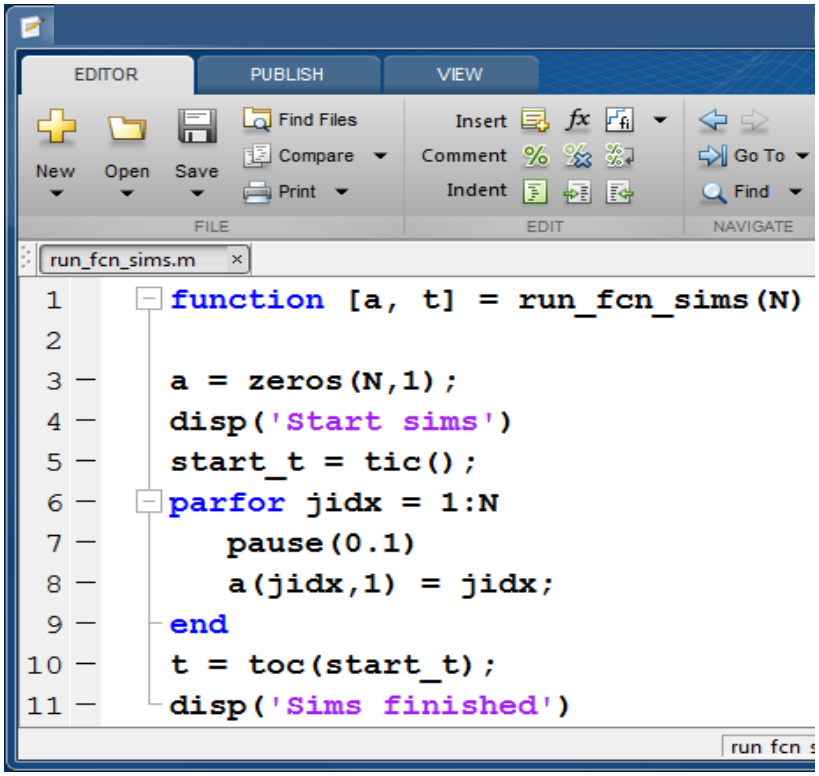


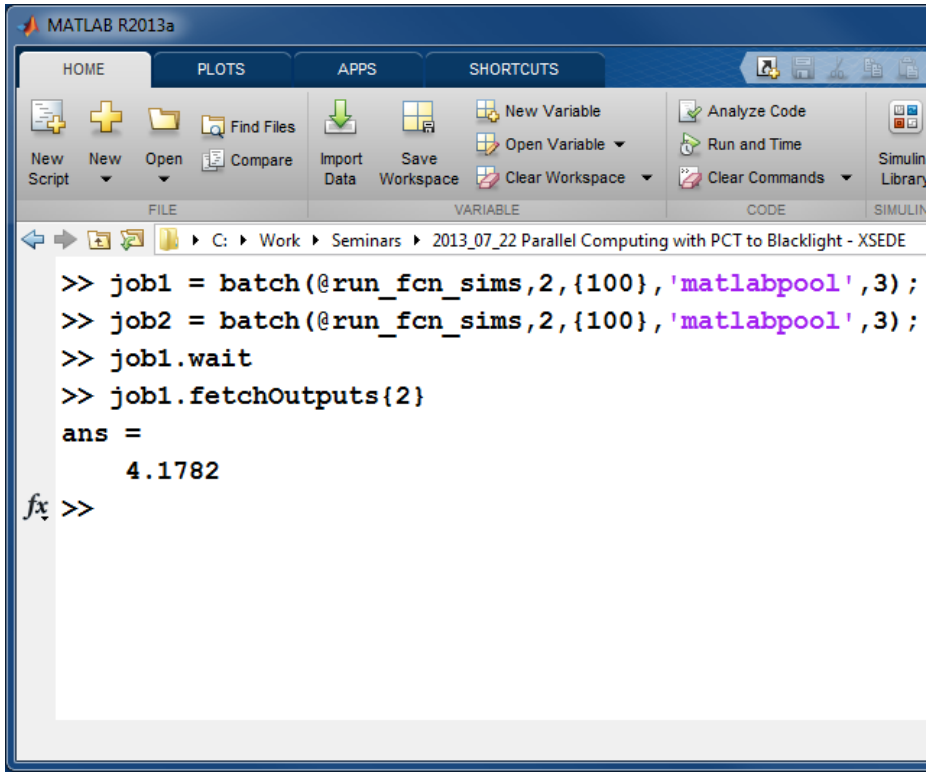
Submitting Scripts with `batch`




```
>> run_sims
```

Submitting Functions with `batch`





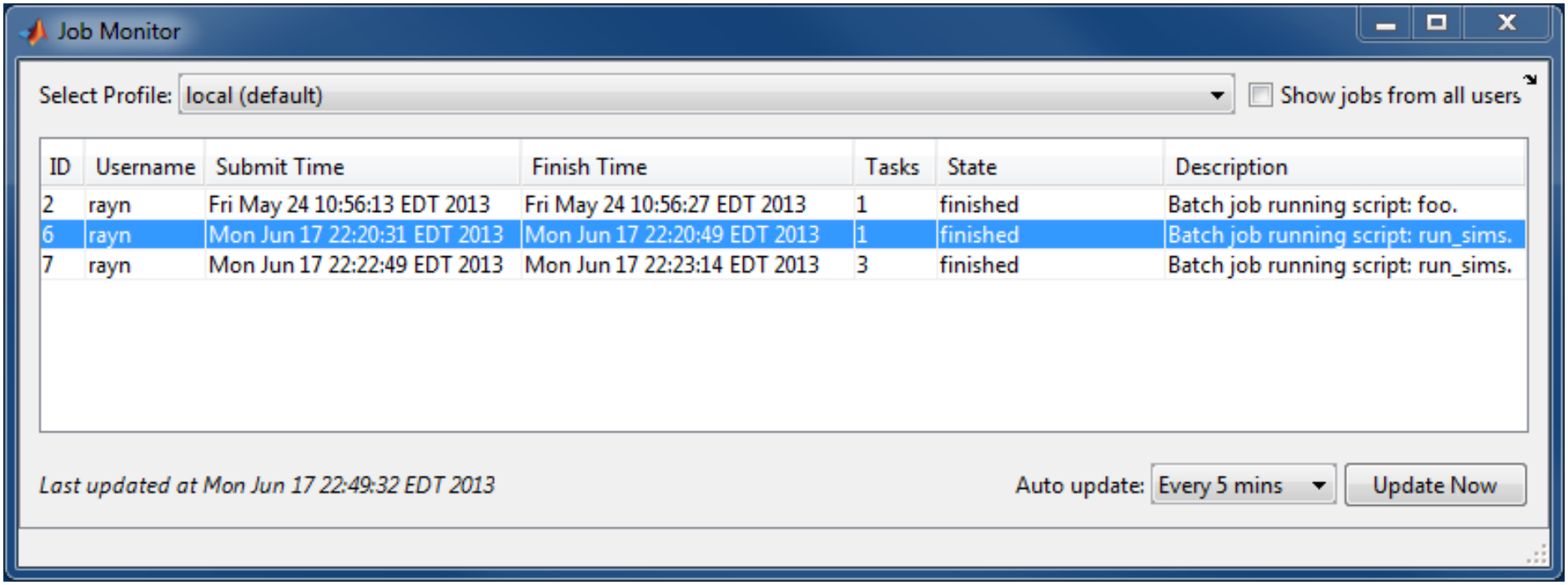

```
>> run_fcn_sims
```

Fixing the `batch` Warning Message

```
Warning: Unable to change to requested working directory.
Reason :Cannot CD to C:\Work (Name is nonexistent or not
a directory).
```

- Call `batch` with CurrentFolder set to '.'
- `job = batch(….,'CurrentFolder','.');`

How Can I Find Yesterday's Job?

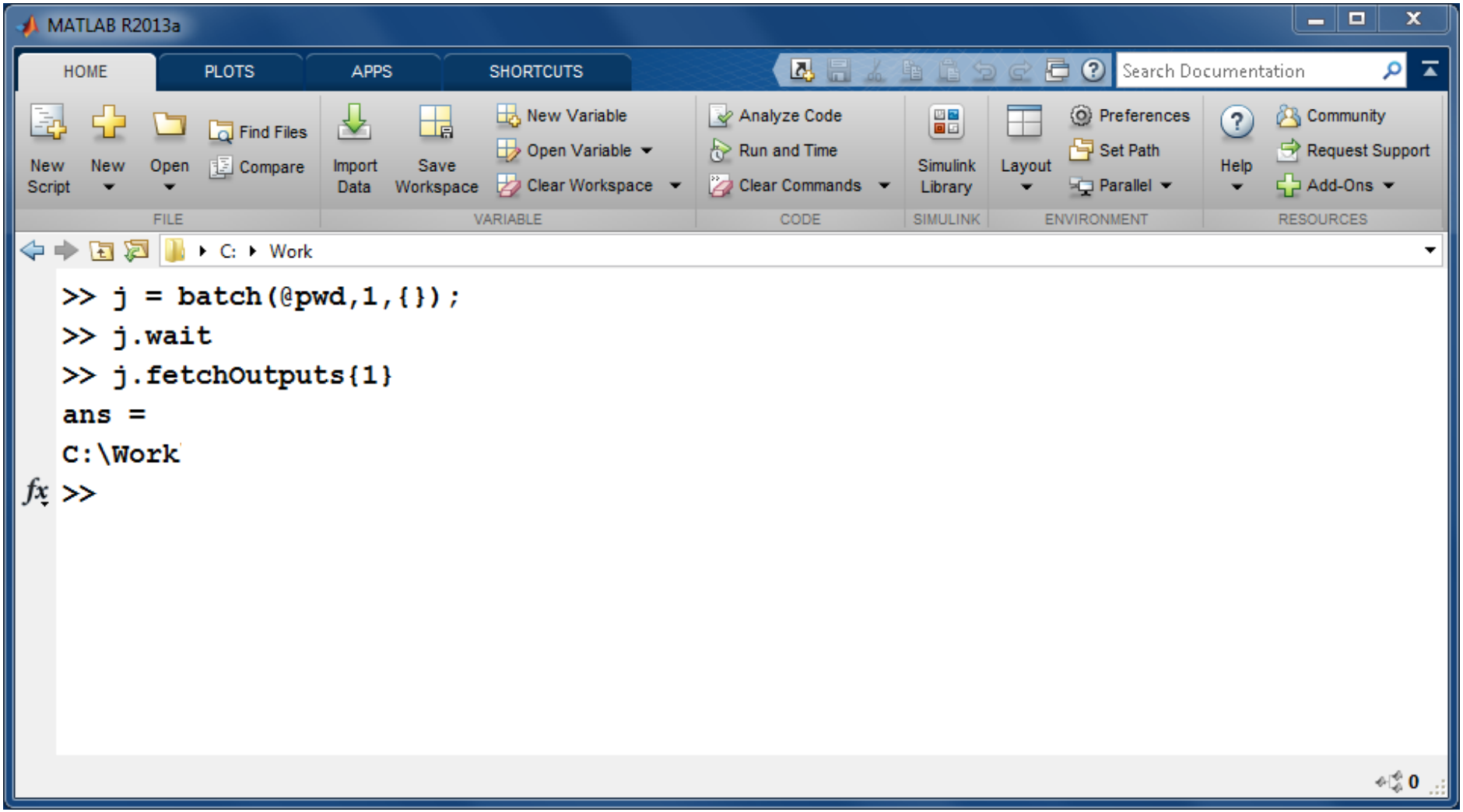


Job Monitor

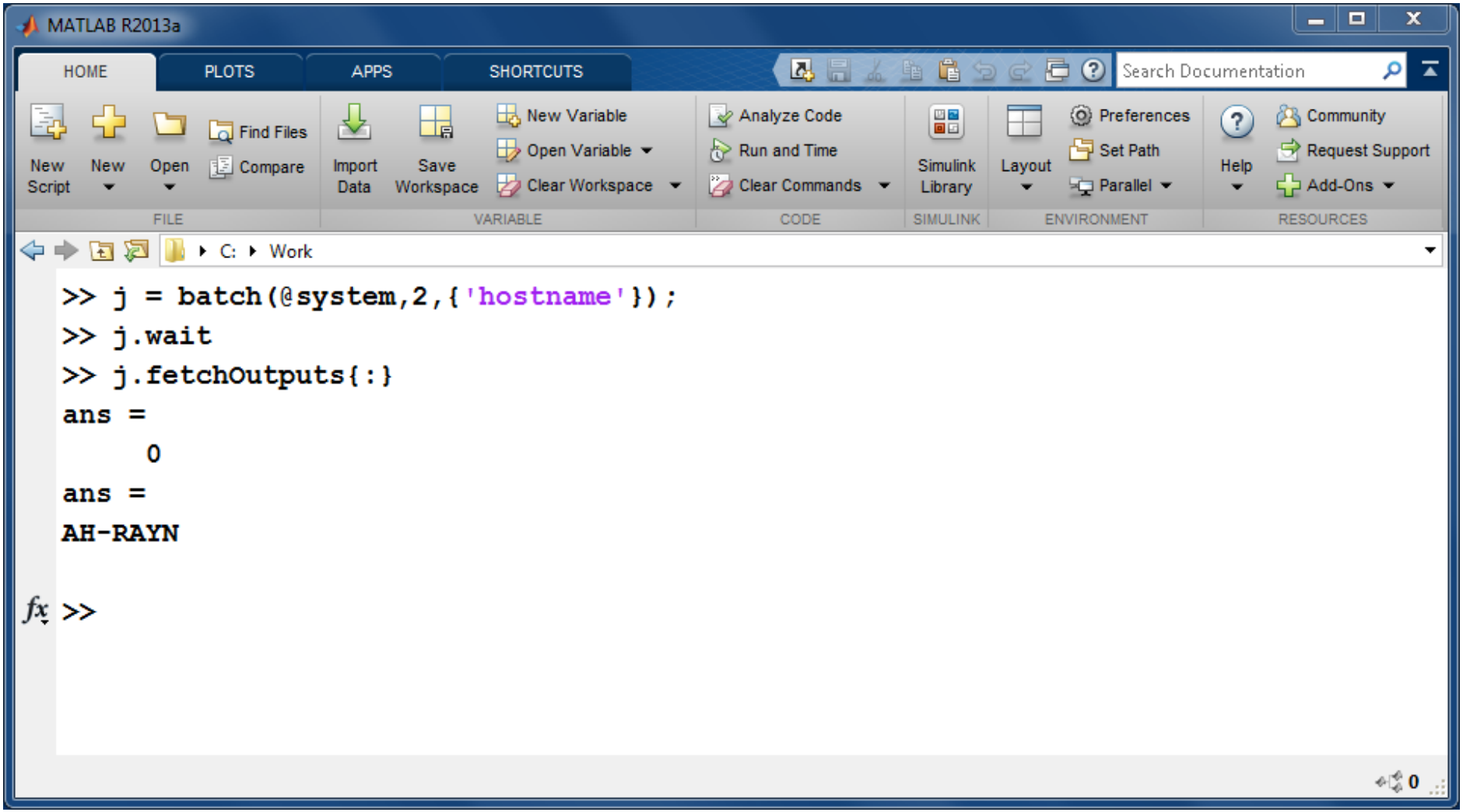
Final Exam: What Final Exam?

- Choose one of the following:
  - ➔ **Submit a job that determines the MATLAB directory your task ran in**
  - Submit a job that determines the machine that ran your task
    - Hint: system(), hostname.exe
- Clear your MATLAB workspace and get a handle to the job you ran above

Final Exam: Solution (1)



Final Exam: Solution (2)



Recommendations

- Profile your code to search for bottlenecks
- Make use of **code analyzer** when coding `parfor` and `spmd`
- Display the correct amount of verbosity for debugging purposes
- Implement an error handler, including capture of calls to 3rd party functions – **don't assume calls to libraries succeed**
- Beware of multiple processes writing to the same file
- Avoid the use of global variables
- **Avoid hard coding path and filenames** that don't exist on the cluster
- Migrate from scripts to functions
- Consider whether or not you'll need to recompile your MEX-files
- After migrating from `for` to `parfor`, switch back to for to make sure nothing has broken
- If calling rand in a for loop, while debugging call rand('seed',0), to get consistent results each time
- When calling matlabpool/batch, parameterize your code

Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting

Troubleshooting and Debugging

- Object data size limitations
  - Single transfers of data between client and workers

System Architecture	Maximum Data Size Per Transfer (approx.)
64-bit	2.0 GB
32-bit	600 MB

- Tasks or jobs remain in Queued state even thought cluster scheduler states it's finished
  - Most likely MDCS failed to startup
- No results or job failed
  - job.load or job.fetchOutputArguments{:}
  - job.Parent.getDebugLog(job)

System Support

System Requirements

- Maximum 1 MATLAB worker / CPU core
- Minimum 1 GB RAM / MATLAB worker
- Minimum 5 GB of disk space for temporary data directories
- GPU
  - CUDA-enabled NVIDIA GPU w/ compute capability 1.3 or above http://www.nvidia.com/content/cuda/cuda-gpus.html
  - Latest CUDA driver http://www.nvidia.com/Download/index.aspx

What's New In R2013a?

- GPU-enabled functions in Image Processing Toolbox and Phased Array System Toolbox
- More MATLAB functions enabled for use with GPUs, including `interp1` and `ismember`
- Enhancements to MATLAB functions enabled for GPUs, including `arrayfun`, `svd`, and `mldivide` (\)
- Ability to launch CUDA code and manipulate data contained in GPU arrays from MEX-functions
- **Automatic detection and transfer of files required for execution in both batch and interactive workflows**
- More MATLAB functions enabled for distributed arrays

Training: Parallel Computing with MATLAB

- Two-day course introducing tools and techniques for distributing code and writing parallel algorithms in MATLAB. The course shows how to increase both the speed and the scale of existing code using PCT.
  - Working with a MATLAB pool
  - Speeding up computations
  - Task-parallel programming
  - Working with large data sets
  - Data-parallel programming
  - Increasing scale with multiple systems
  - **Prerequisites:** *MATLAB Fundamentals*

- mathworks.com/training