Accelerating and Parallelizing MATLAB Code on HPC Infrastructure

Francesca Perino – Sam Marshalik - Sergio Obando Quintero
Application Engineering Team

Choose a Parallel Computing Solution

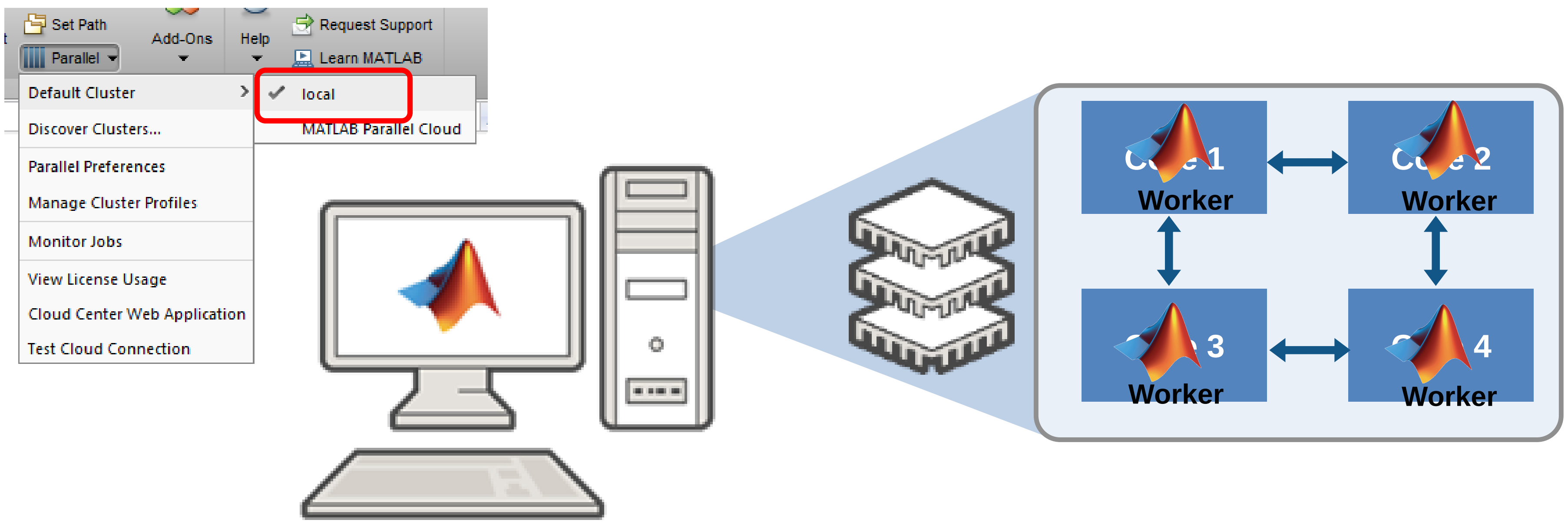
- Do you want to process your data faster?
- Do you want to offload to a cluster?
- Do you want to scale up your big data calculation?

Practical Application of Parallel Computing

- Why parallel computing?
  - Need faster insight on more complex problems with larger datasets
  - Computing infrastructure is broadly available (multicore desktops, GPUs, clusters)
- Why parallel computing with MATLAB
  - Leverage computational power of more hardware
  - Accelerate workflows with minimal to no code changes to your original code
  - Focus on your engineering and research, not the computation

Parallel Computing Paradigm

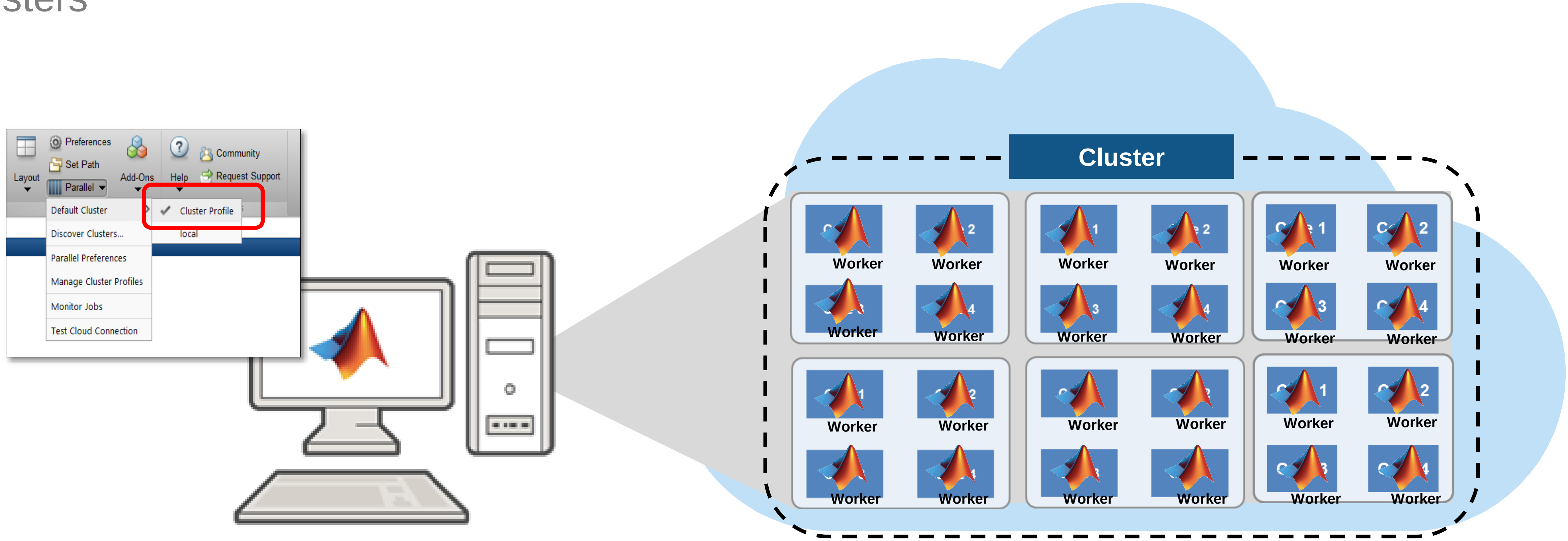
Multicore Desktops



MATLAB multicore

Parallel Computing Paradigm

Clusters



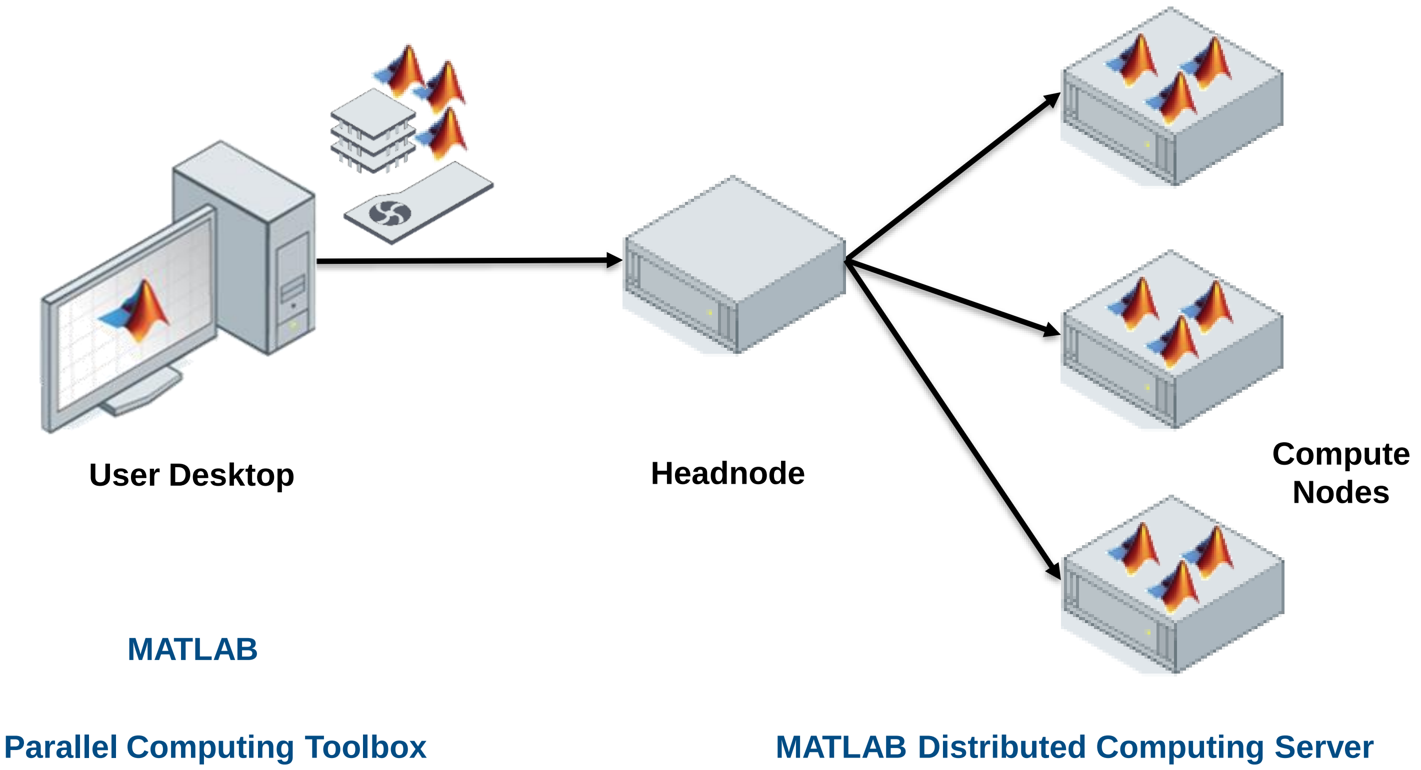
Migrate execution to a cluster environment

MATLAB
Parallel Computing Toolbox

MATLAB Distributed Computing Server

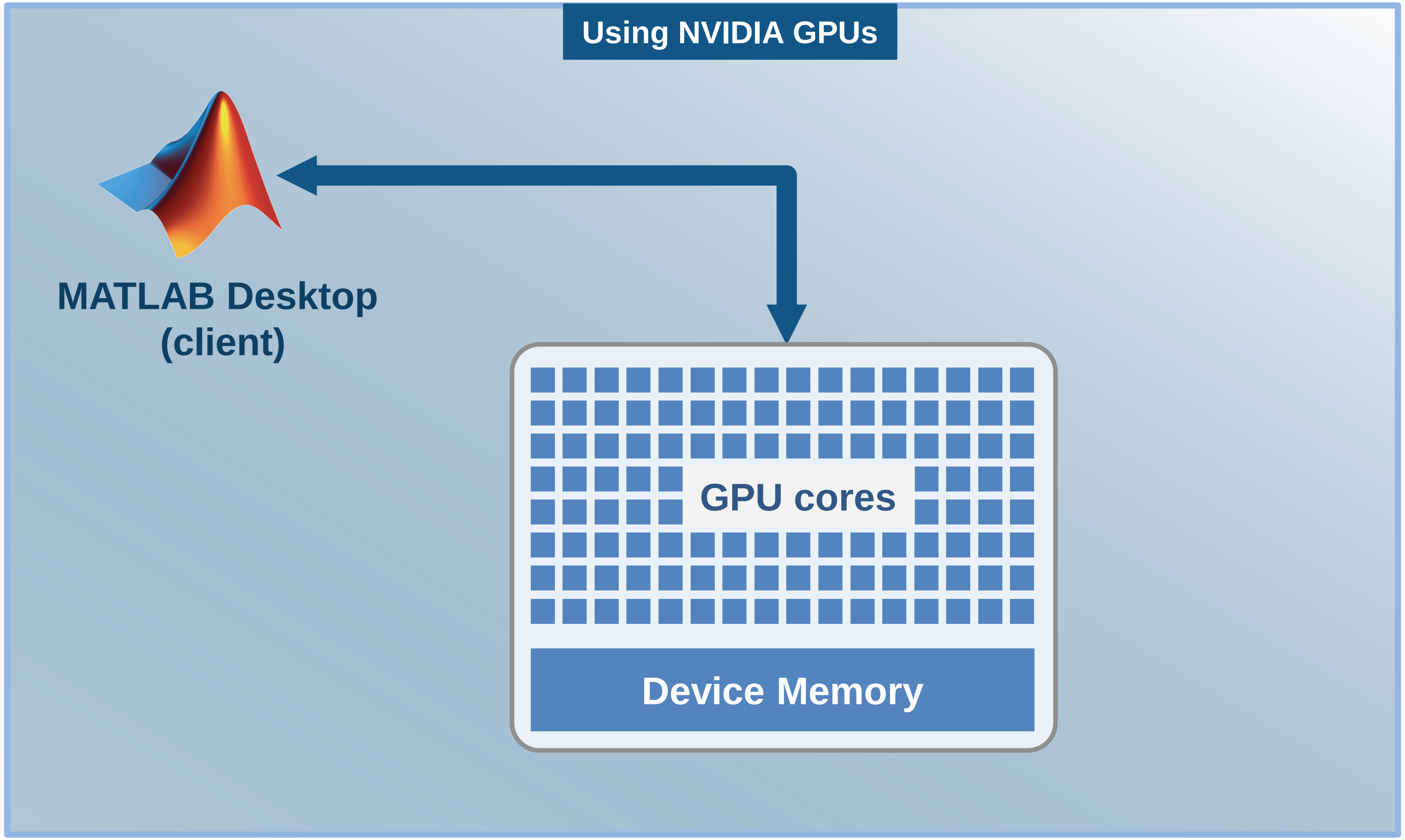
Cluster Computing Paradigm

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



Parallel Computing Paradigm

NVIDIA GPUs



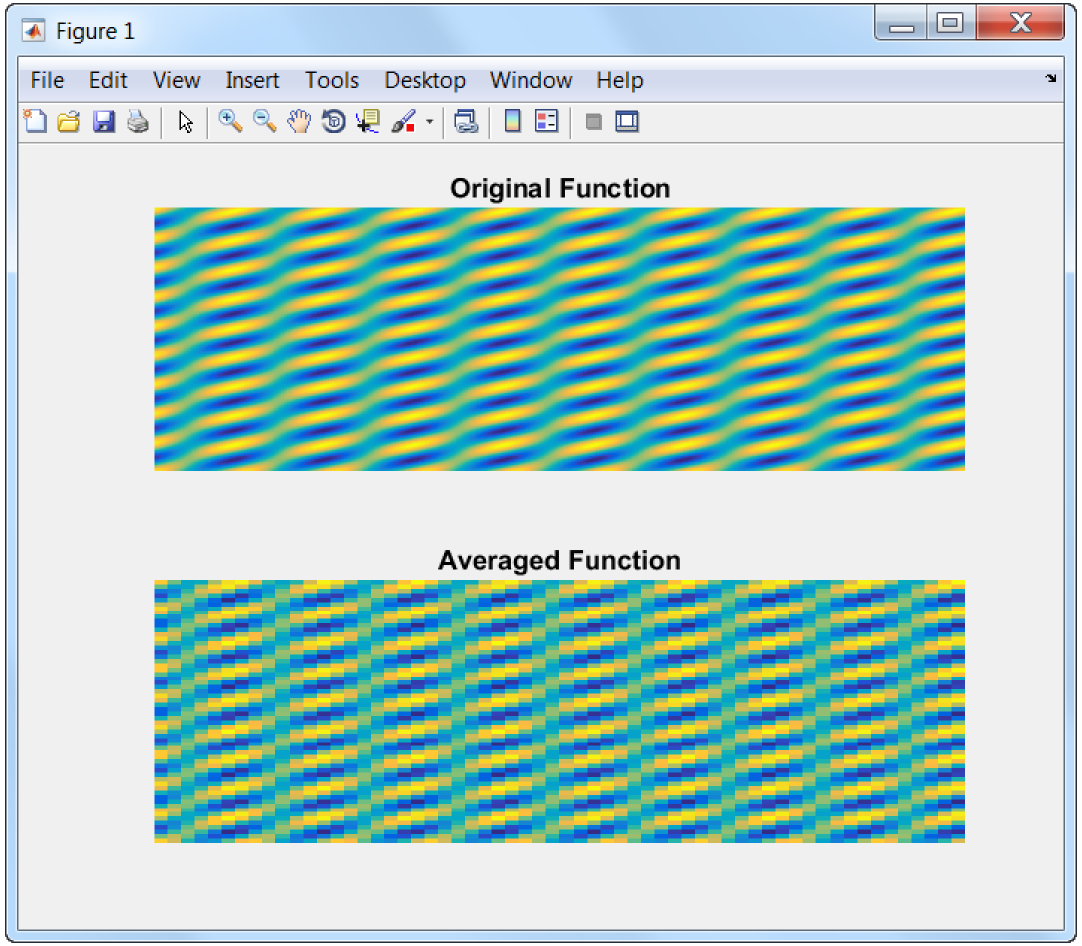
Steps for writing a MATLAB parallel code

1. Best practices in programming ▪ Identify bottlenecks (e.g. Profiler, Code analyzer) ▪ Vectorization & pre-allocation
2. Better algorithms ▪ Different algorithmic approach to solve the same problem ▪ The most recent MATLAB release
3. More processors, cores, and GPUs ▪ Utilize high level parallel constructs (e.g. `parpool`, `parfor`) ▪ Scale to clusters, grids, and clouds

web(fullfile(docroot, 'matlab/matlab_prog/techniques-for-improving-performance.html'))

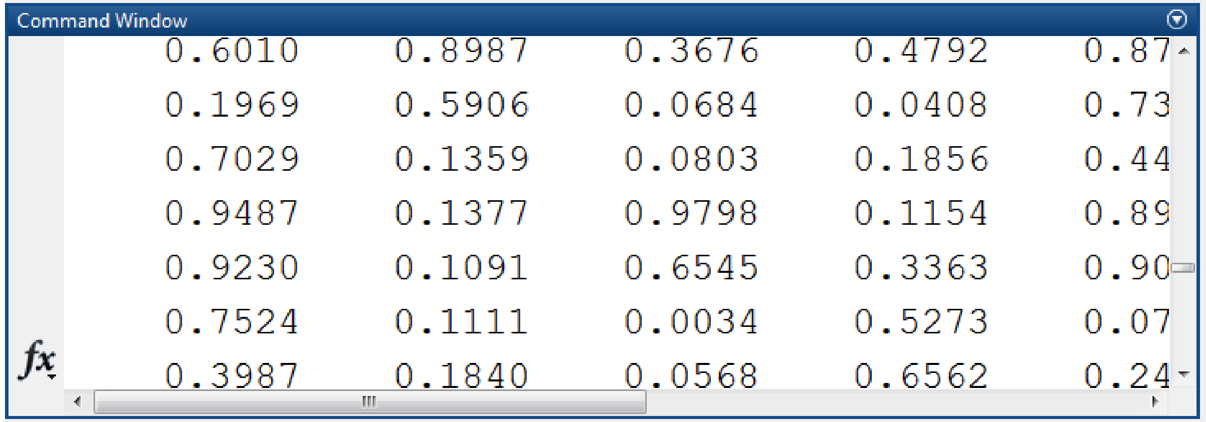
Example: Block Processing Images

- Calculate a function at grid points
- Take the mean of larger blocks



Best Practices

- Profile your code
- Minimize file I/O
- Reuse existing graphics components
- Avoid printing to Command Window



Access Multiple Files to Import Specific Columns

```matlab
tic
for ii = length(D):-1:1    % Takes around 2 minutes to process
    fullname        = [Location '\' D(ii).name];
    [~, name, ~]    = fileparts(fullname); % Strips off ".xls"
    Engine          = importExcelData(fullname);
    [val, index]    = min(Engine.BSFC);
    EngineMin       = Engine(index,:);
    EngineMin.Name = categorical({name}); % Add the name
    AllEngines(ii,:) = EngineMin;
end
toc
```

```matlab
%% Import the data
data = [xlsread(workbookFile, sheetName, sprintf('A%d:A%d',startRow(1),endRow(1))),...
    xlsread(workbookFile, sheetName, sprintf('G%d:G%d',startRow(1),endRow(1))),...
    xlsread(workbookFile, sheetName, sprintf('U%d:U%d',startRow(1),endRow(1)))];
for block=2:length(startRow)
    tmpDataBlock = [xlsread(workbookFile, sheetName, sprintf('A%d:A%d',startRow(block),endRow(block))),...
        xlsread(workbookFile, sheetName, sprintf('G%d:G%d',startRow(block),endRow(block))),...
        xlsread(workbookFile, sheetName, sprintf('U%d:U%d',startRow(block),endRow(block)))];
    data = [data;tmpDataBlock]; %#ok<AGROW>
end
```

Access Multiple Files to Import Specific Columns

☰ CONTENTS

spreadsheetDatastore

Create SpreadsheetDatastore object for collections of spreadsheet data

Syntax

```
ssds = spreadsheetDatastore(location)
ssds = spreadsheetDatastore(location,Name,Value)
```

Description

`ssds = spreadsheetDatastore(location)` creates a datastore from the collection of data specified by locatio collections of data that are too large to fit in memory. After creating a `SpreadsheetDatstore` object, you can read ways. See Using SpreadsheetDatastore Objects for more information.

`ssds = spreadsheetDatastore(location,Name,Value)` specifies additional parameters for `ssds` using one or For example, `spreadsheetDatastore(location,'FileExtentions',{'.xlsx','.xls'})` specifies which files depending on the file extensions.

```
%% Method #2:
% we use |spreadsheetDatastore|
tic
dsEngine = ...
    spreadsheetDatastore('data','FileExtensions',{'.xls','.xlsm'});
dsEngine.ReadSize = 'file';
%%
dsEngine.SelectedVariableNames = {'SPEED','LOAD','BSFC'};
%%
ii = 1;
while hasdata(dsEngine)
    Engine           = read(dsEngine);
    [val, index]     = min(Engine.BSFC);
    EngineMin        = Engine(index,:);
    [~,name]         = fileparts(dsEngine.Files{ii});
    EngineMin.Name   = categorical({name}); % Add the name
    AllEngines(ii,:) = EngineMin;
    ii = ii+1;
end
toc
```

Steps for writing a MATLAB parallel code

1. Best practices in programming ▪ Identify bottlenecks (e.g. Profiler, Code analyzer) ▪ Vectorization & pre-allocation
2. Better algorithms ▪ Different algorithmic approach to solve the same problem ▪ The most recent MATLAB release
3. More processors, cores, and GPUs ▪ Utilize high level parallel constructs (e.g. `parpool`, `parfor`) ▪ Scale to clusters, grids, and clouds

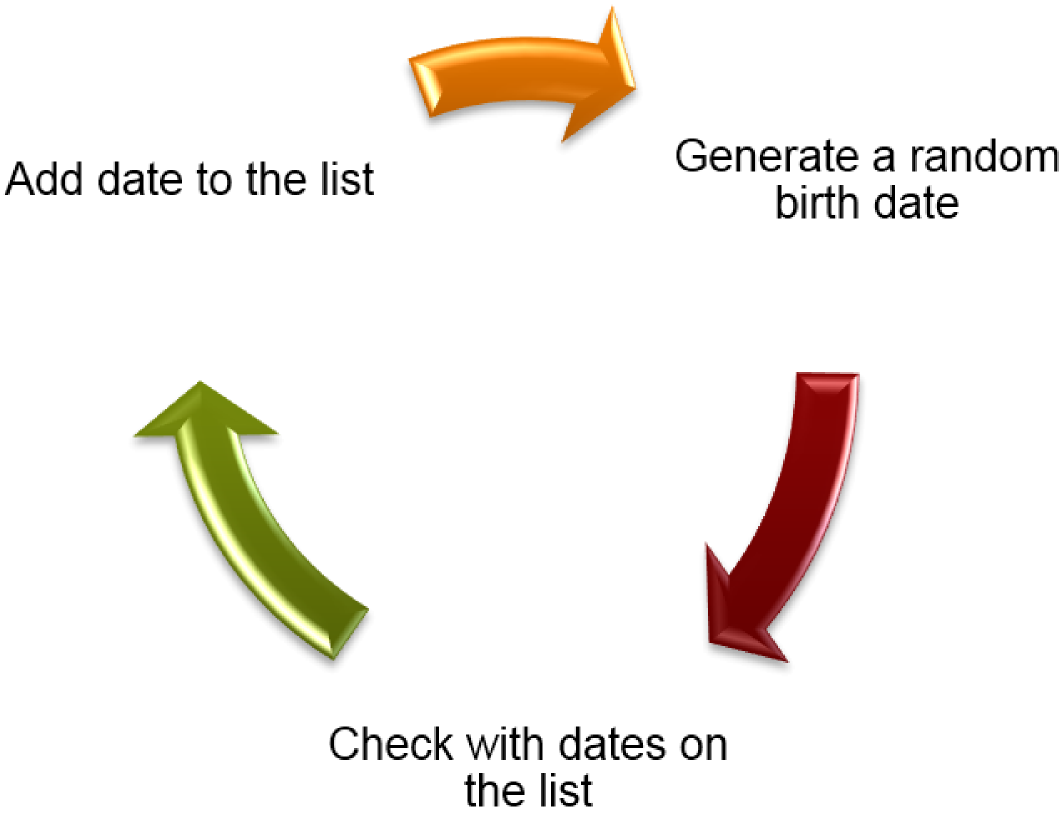
web(fullfile(docroot, 'matlab/matlab_prog/techniques-for-improving-performance.html'))

Exercise: Birthday Paradox

- What is the probability that in a group of 23 randomly selected individual, at least two of them will share the same birthday?

Exercise: Birthday Paradox Implementation

- Profile `runBirthdaySum.m`
- Edit `runBirthdayUnique1.m`
  - **TODO:** without a FOR loop create a list with a random birthday for each member in the group
- Edit `runBirthdayVec.m`
  - **TODO:** try a different algorithmic approach based on |sort| to solve the same problem



Parallel and Distributed Computing with MATLAB

Steps for writing a MATLAB parallel code

1. **Best practices in programming**
   - Identify bottlenecks (e.g. Profiler, Code analyzer)
   - Vectorization & pre-allocation
2. **Better algorithms**
   - Different algorithmic approach to solve the same problem
   - The most recent MATLAB release
3. **More processors, cores, and GPUs**
   - Utilize high level parallel constructs (e.g. `parpool`, `parfor`)
   - Scale to clusters, grids, and clouds

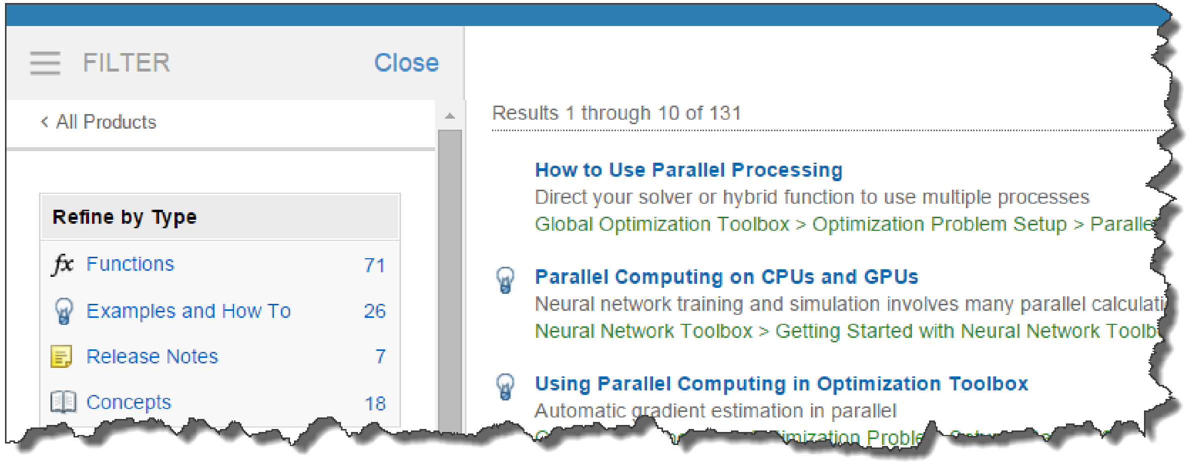
web(fullfile(docroot, 'matlab/matlab_prog/techniques-for-improving-performance.html'))

Programming Parallel Applications

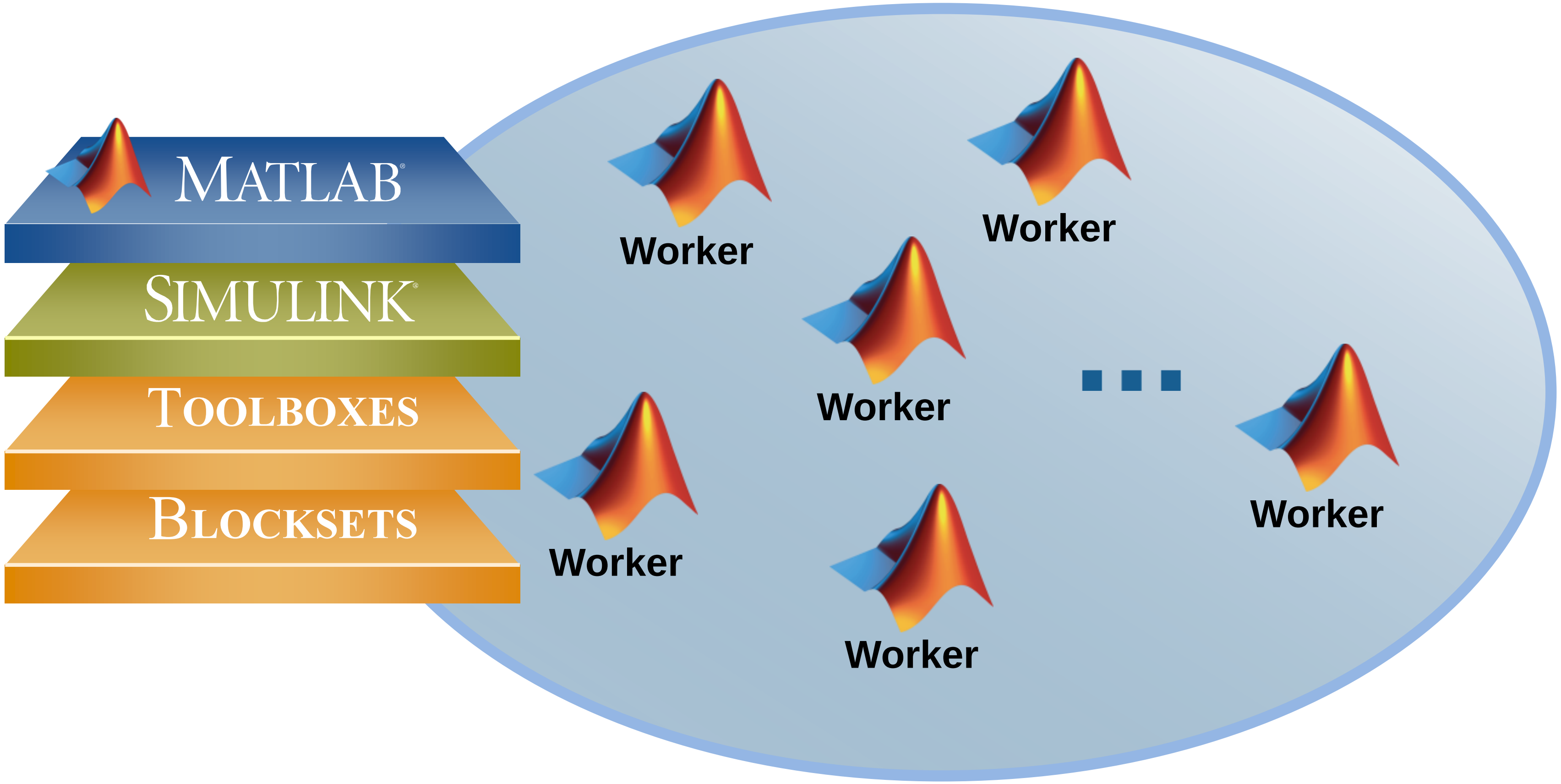
- Built-in multithreading
  - Automatically enabled in MATLAB since R2008a
  - Multiple threads in a single MATLAB computation engine

- Parallel-enabled MATLAB Toolboxes
  - Enable parallel computing support by setting a flag or preference

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



Parallel Computing



Parallel-enabled Toolboxes (MATLAB® Product Family)

Enable acceleration by setting a flag or preference

Image Processing

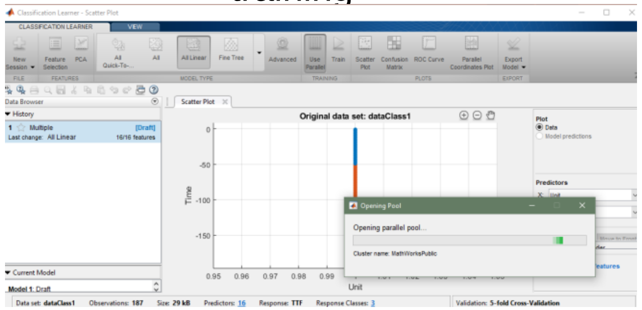
Batch Image Processor, Block Processing, GPU-enabled functions

Original Image of Peppers

Recolored Image of Peppers

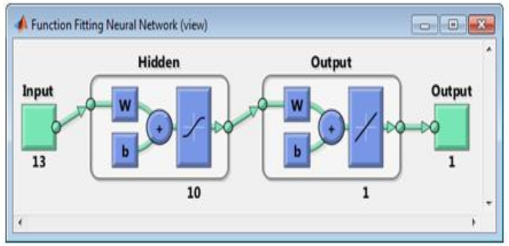
Statistics and Machine Learning

GPU-enabled functions, parallel training



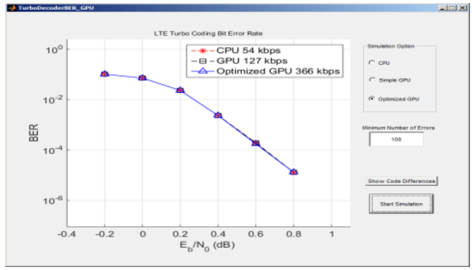
Neural Networks

Deep Learning, Neural Network training and simulation



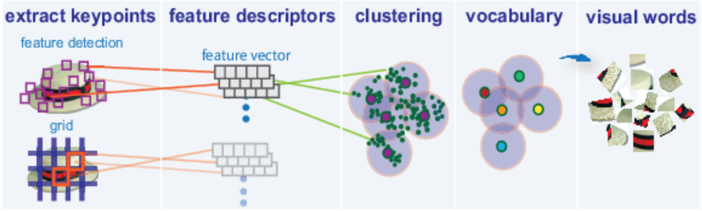
Signal Processing and Communications

GPU-enabled FFT filtering, cross correlation, BER simulations



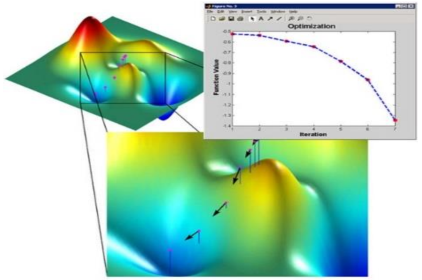
Computer Vision

Bag-of-words workflow



Optimization

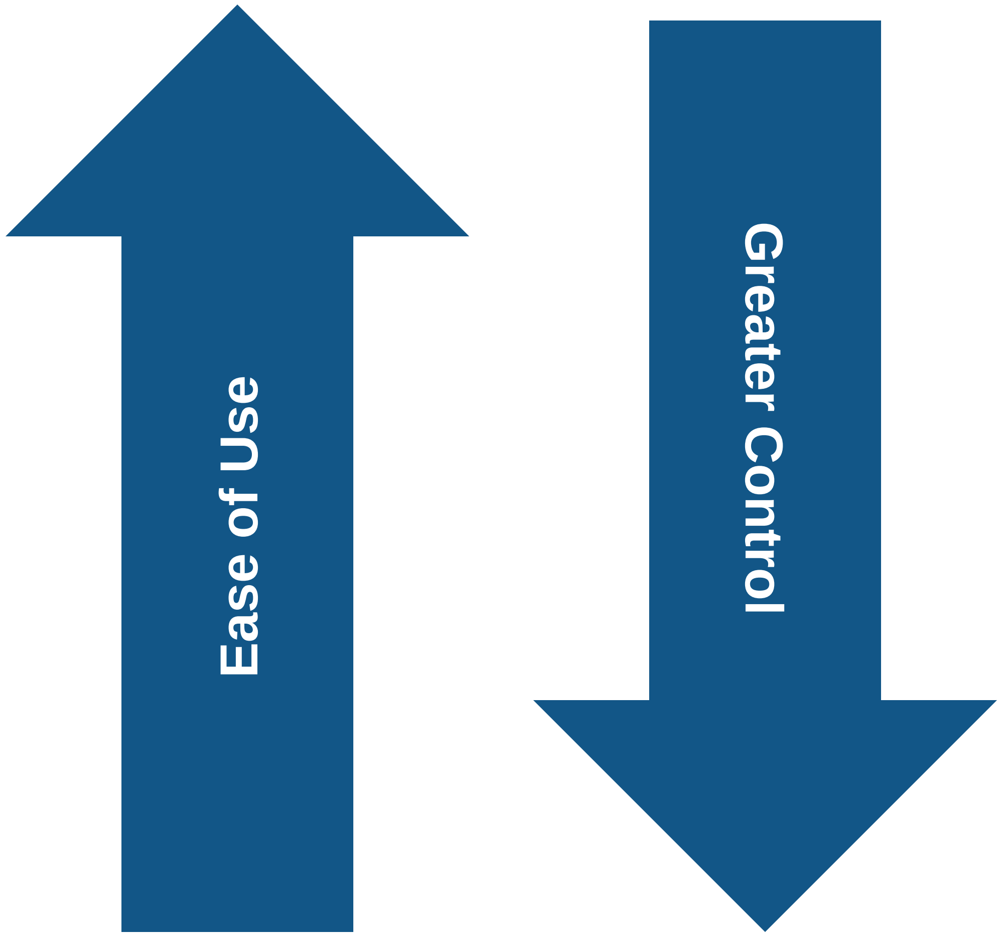
Estimation of gradients



[Other Parallel-enabled Toolboxes]

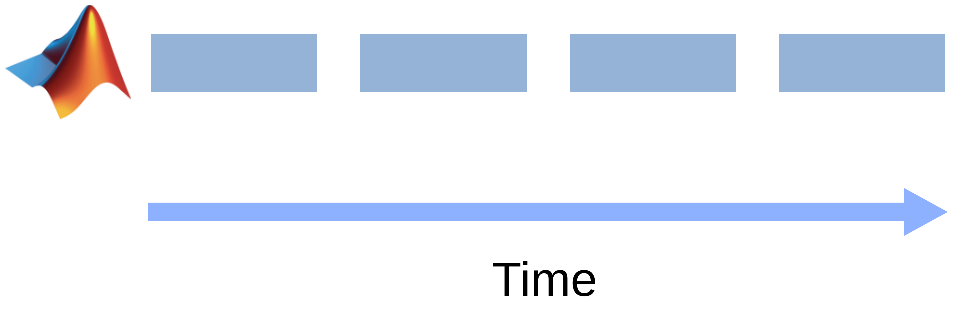
Programming Parallel Applications

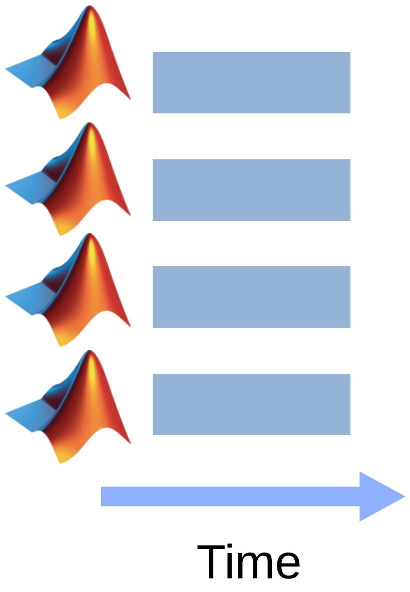
- Built in support
  - `..., 'UseParallel', true)`
- Simple programming constructs
  - `parfor, batch`



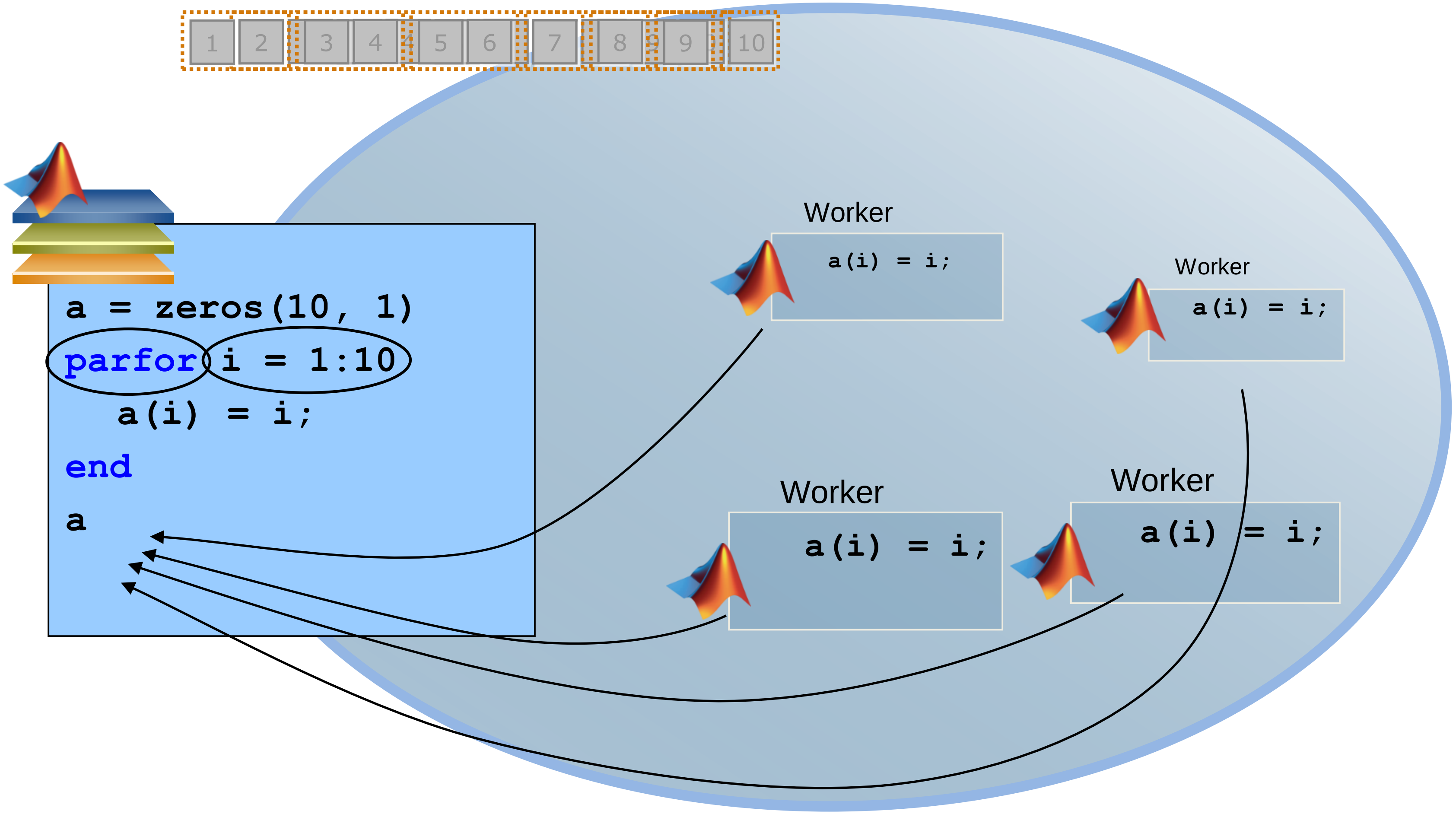
Embarrassingly Parallel: Independent Tasks or Iterations

- No dependencies or communication between tasks
- Examples:
  - Monte Carlo simulations
  - Parameter sweeps
  - Same operation on many files





Mechanics of `parfor` Loops



Example: Estimate π using the Buffon-Laplace method

	trials	pi_value	diff
1	10	2.5000	0.6416
2	100	3.1250	0.0166
3	1000	3.0541	0.0875
4	10000	3.1401	0.0015
5	100000	3.1572	0.0156
6	1000000	3.1390	0.0025
7	10000000	3.1420	0.0004
8	100000000	3.1417	0.0001

non-dimensional length of grid side, x

```
a = 1;
```

non-dimensional length of grid side, y

```
b = 1;
```

non-dimensional length of needle,

```
nLength = 0.5;
```

number of needles considered

```
nNeedles = 10.^(1:8) ;
```

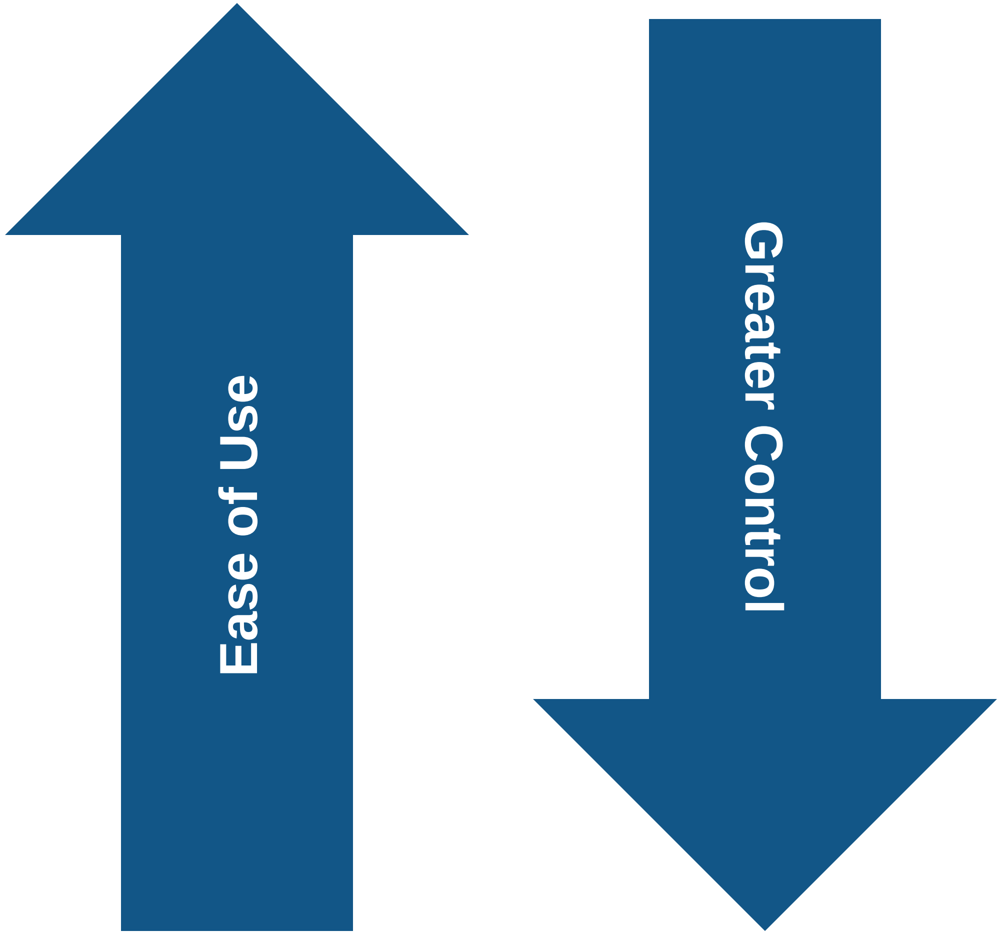
```
for i = 1:length(nNeedles)
    pi_N(i)    = calcPI(nNeedles(i),nLength,a,b);
end
```

Factors Governing the Speedup of `parfor` Loops

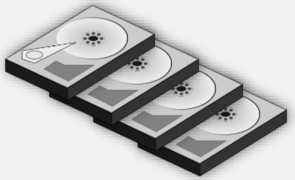
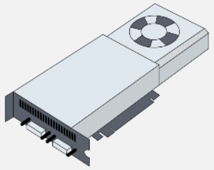
- No speedup because computation time too short
- Execution may be slow because of
  - Memory limitations (RAM)
  - File access limitations
- Implicit multithreading
  - MATLAB uses multiple threads for speedup of some operations
  - Use Task Manager or similar on serial code to check on that
- Unbalanced load due to iteration execution times
  - Avoid some iterations taking multiples of the execution time of other iterations.

Programming Parallel Applications

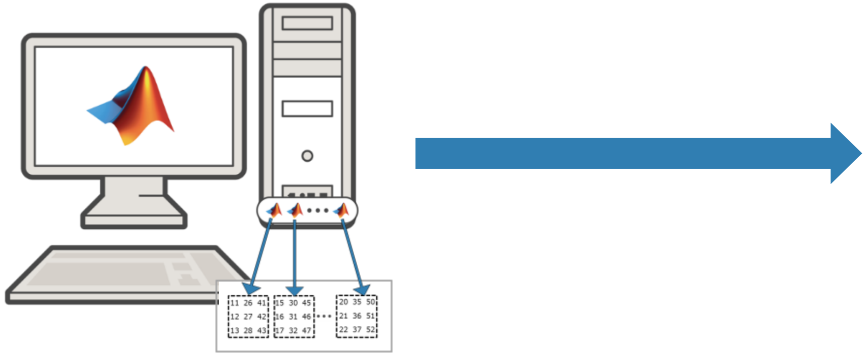
- Built in support
  - `..., 'UseParallel', true)`
- Simple programming constructs
  - `parfor, batch`
- Full control of parallelization
  - `spmd, parfeval`

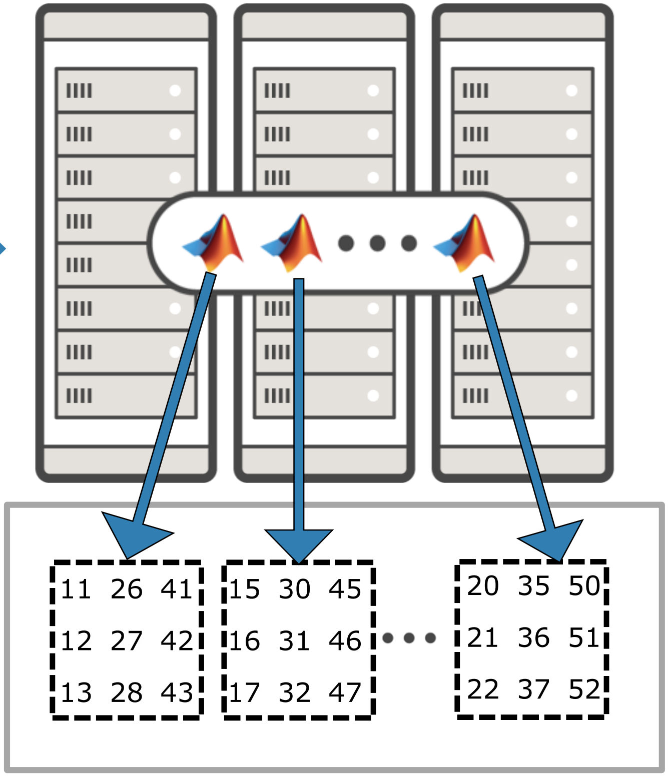


Datatypes for Scaling Data
Represent data *not* in “normal” memory

Datatype		Memory Location	Use case
`tall`		Disks	Pre-processing, statistics, machine learning
`distributed`		Cluster	Sparse and dense numerics
`gpuArray`		GPU	GPU computations

Distributed Arrays





MATLAB and Parallel Computing Toolbox

```
parpool('local')
x = A\b;
% prototype with small A,b
% A,b are distributed arrays
```

MATLAB Distributed Computing Server

```
parpool(<cluster>)
x = A\b;
% For large A,b
% A,b are distributed arrays
```

Develop applications once, change run environment by changing the profile

Example: Estimate π using the Buffon-Laplace method

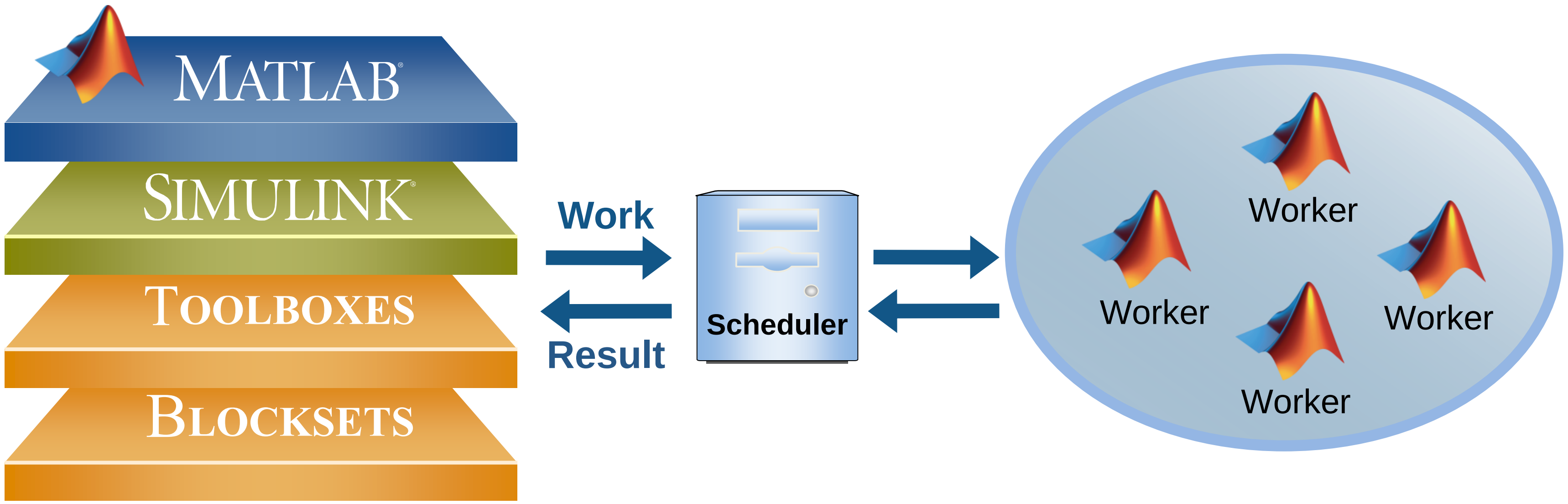
- We want to speed up the estimation of π for 10^9 trials
  - Define a 10^9-by-1 codistributed arrays, distributed by columns with a uniform partition scheme.

```
›x0 = a * rand(nNeedles,1,codistributor);
›y0 = b * rand(nNeedles,1,codistributor);
›phi= 2 * pi * rand(nNeedles,1,codistributor);
```

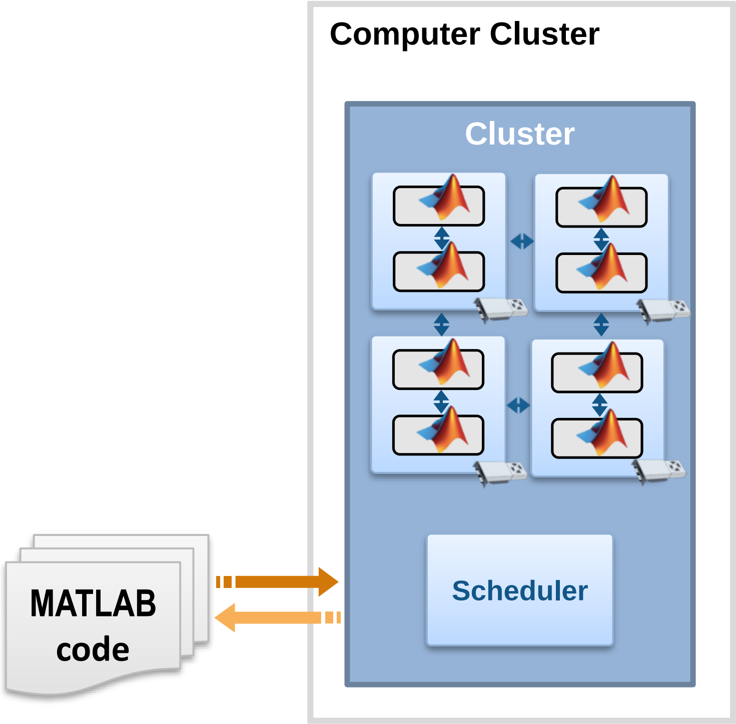
  - Create on *x* workers

```
spmd
    piN = spmdCalcPI(nNeedles,a,b,nLength);
end
```

Offloading Computations



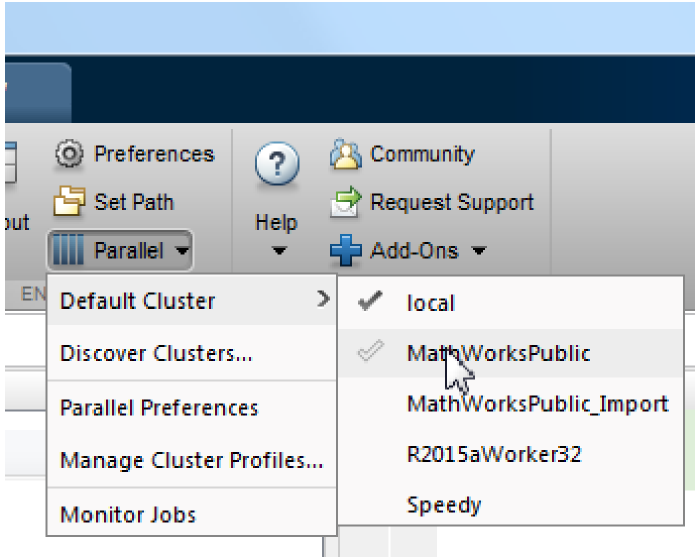
Offloading Computations



- Send desktop code to cluster resources
  - No parallelism required within code
  - Submit directly from MATLAB
- Leverage supplied infrastructure
  - File transfer / path augmentation
  - Job monitoring
  - Simplified retrieval of results
- Scale offloaded computations

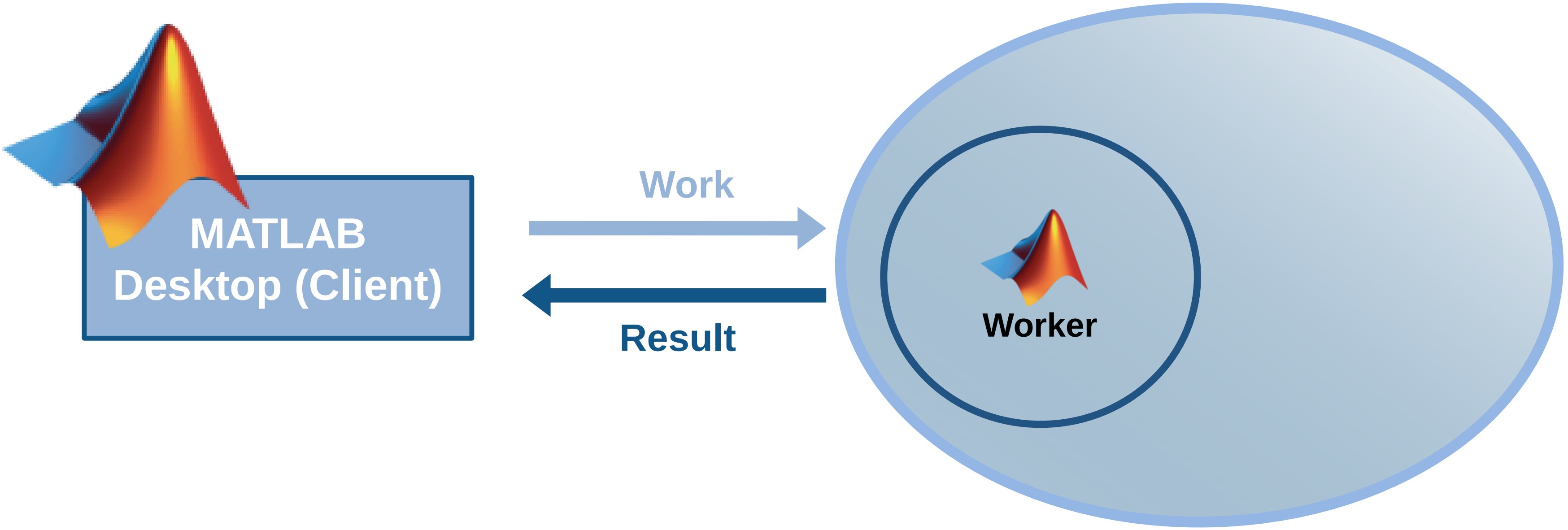
Migrate to Cluster / Cloud

- Use MATLAB Distributed Computing Server
- Change hardware without changing algorithm



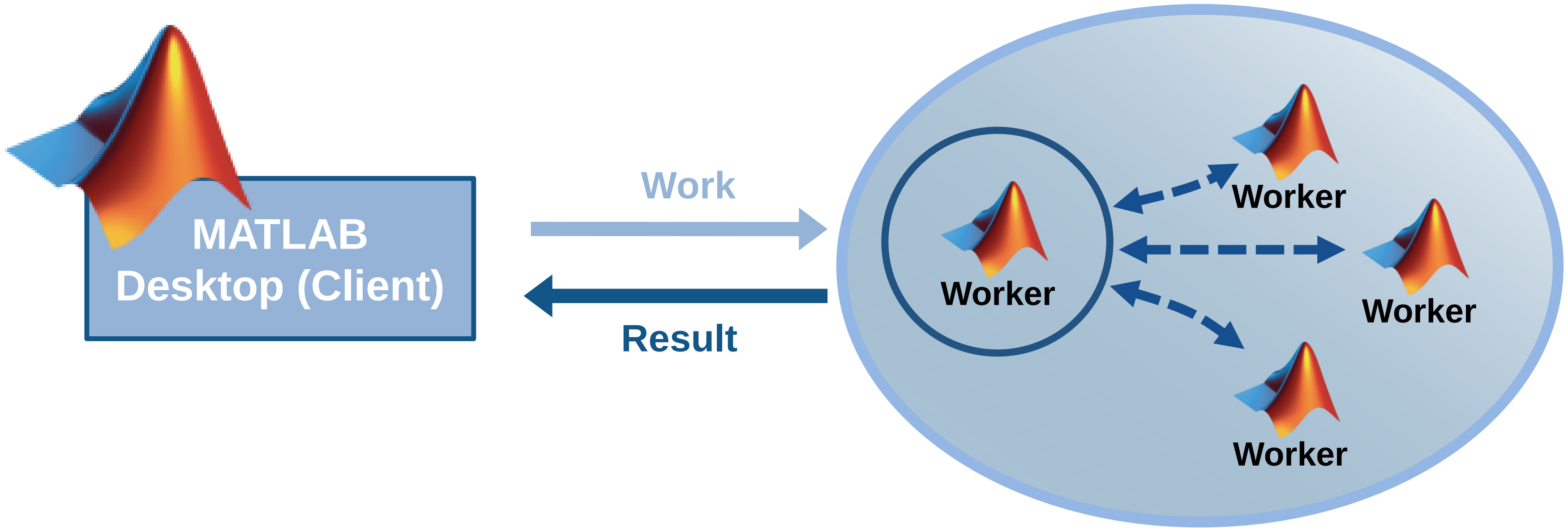
Offloading Serial Computations with `batch`

- Offload the computation to a workstation targets compute-intensive applications



`batch(…)`

Offload and Scale Computations with `batch` with a Parallel Pool




```
batch(…, 'Pool',…)
```

- batch jobs are particularly suitable when you are working on a compute cluster.

Estimate π using the Buffon-Laplace method

Run MATLAB script or function on a worker in the cluster specified by the default cluster profile:

```
c = parcluster()
j = batch(c,@batchCalcPI,1,{nNeedles,nLength,a,b},...
    'Pool',length(nNeedles),...
    'AttachedFiles','calcPI.m')
```

Wait for the job to finish. To see your batch job's status or to track its progress, use the Job Monitor, as described in Job Monitor

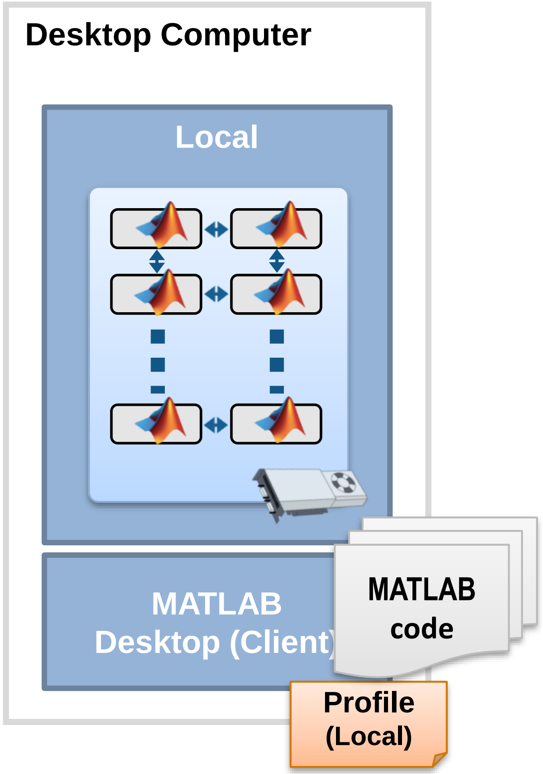
```
wait(j)
elapsedTime = j.FinishDateTime-j.StartDateTime
```

Get results into a cell array

```
pi_N = fetchOutputs(j);
pi_diff = abs(pi-pi_N{1});
pi_table = table(nNeedles',pi_N{1}',pi_diff','VariableNames',{'trials','pi_value','diff'})
```

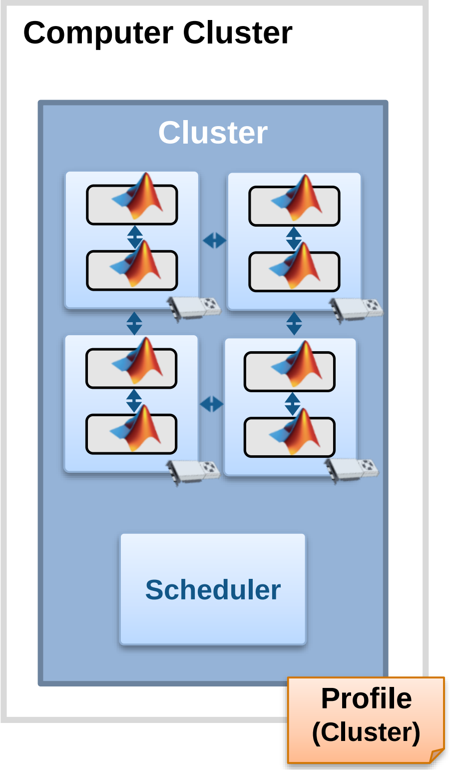
Use MATLAB Distributed Computing Server

1. Prototype code



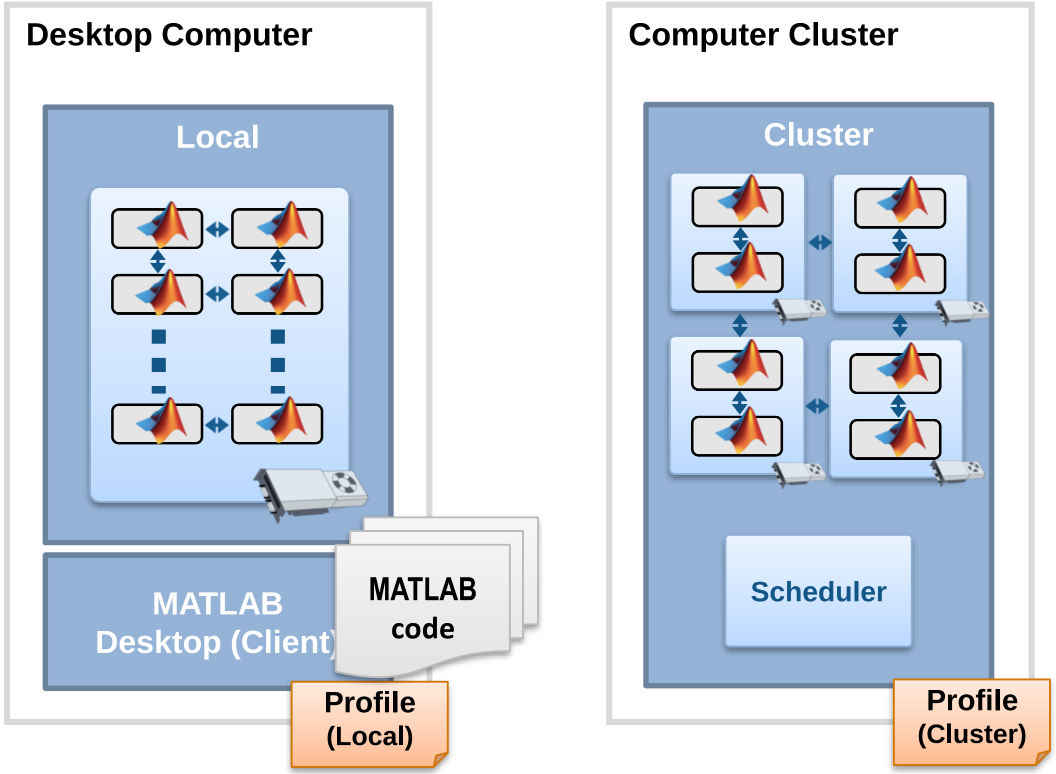
Use MATLAB Distributed Computing Server

1. Prototype code
2. Get access to an enabled cluster



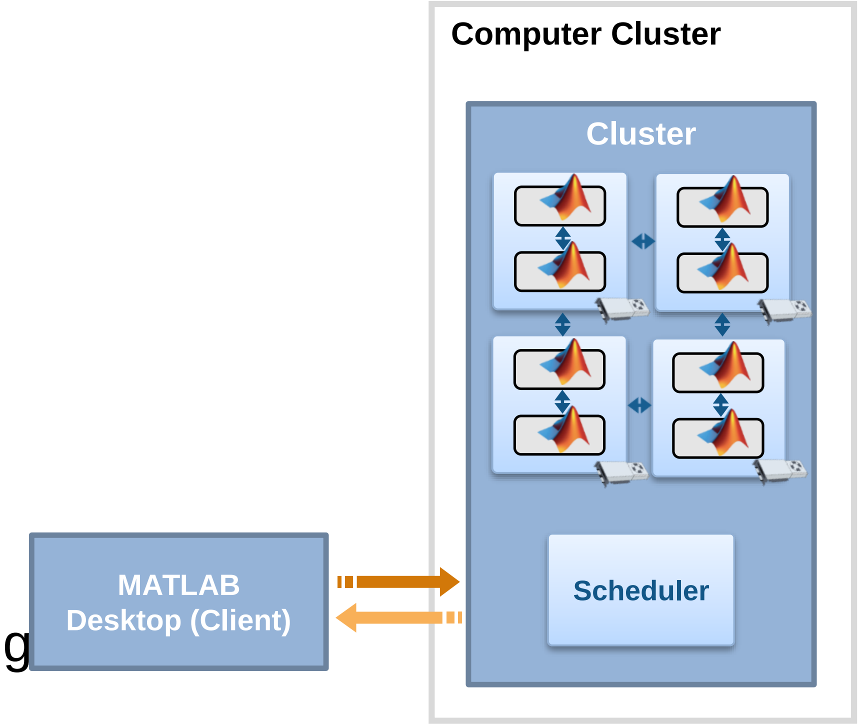
Use MATLAB Distributed Computing Server

1. Prototype code
2. Get access to an enabled cluster
3. Switch cluster profile to run on cluster resources



Take Advantage of Cluster Hardware

- Offload computation:
  - Free up desktop
  - Access better computers
- Scale speed-up:
  - Use more cores
  - Go from hours to minutes
- Scale memory:
  - Utilize tall arrays and distributed arrays
  - Solve larger problems without re-coding alg



Summary

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

Parallel Computing with MATLAB – Beyond PARFOR

Well-known features

- parallel-enabled toolboxes
- `parfor/parsim`
- `gpuArray`

Full spectrum of support

- batch submission, jobs and tasks
  `batch, createJob, createTask`
- asynchronous queue for feval
  `parfeval`
- parallel support for big data
  `tall, mapreduce`
- distributed arrays (“global arrays”)
  `distributed, codistributed`
- message passing
  `labSend, labReceive`

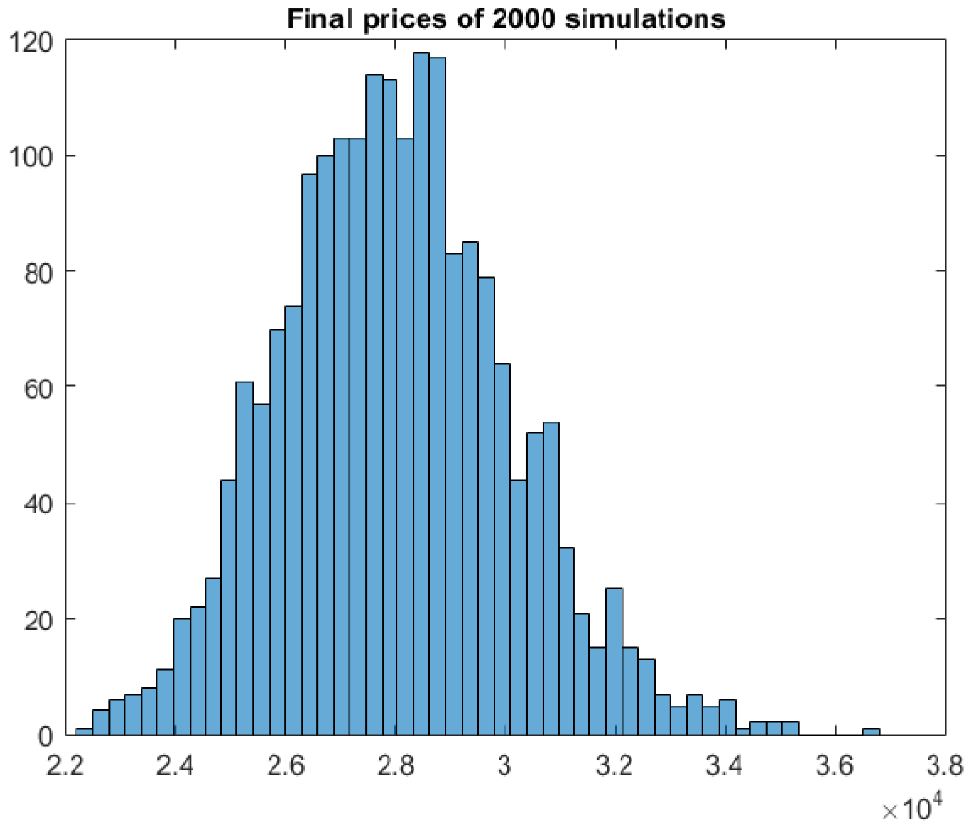
tutorials

Some Other Valuable 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 on the Cloud with MATLAB
  - http://www.mathworks.com/products/parallel-computing/parallel-computing-on-the-cloud/

Monte Carlo Price Simulation I

1. Inspect original code in `gdpSim.m`.
2. Vectorize the code performance by eliminating a `for`-loop. Compare timings and results.
3. Eliminate the remaining loop. Again, compare timings and results.

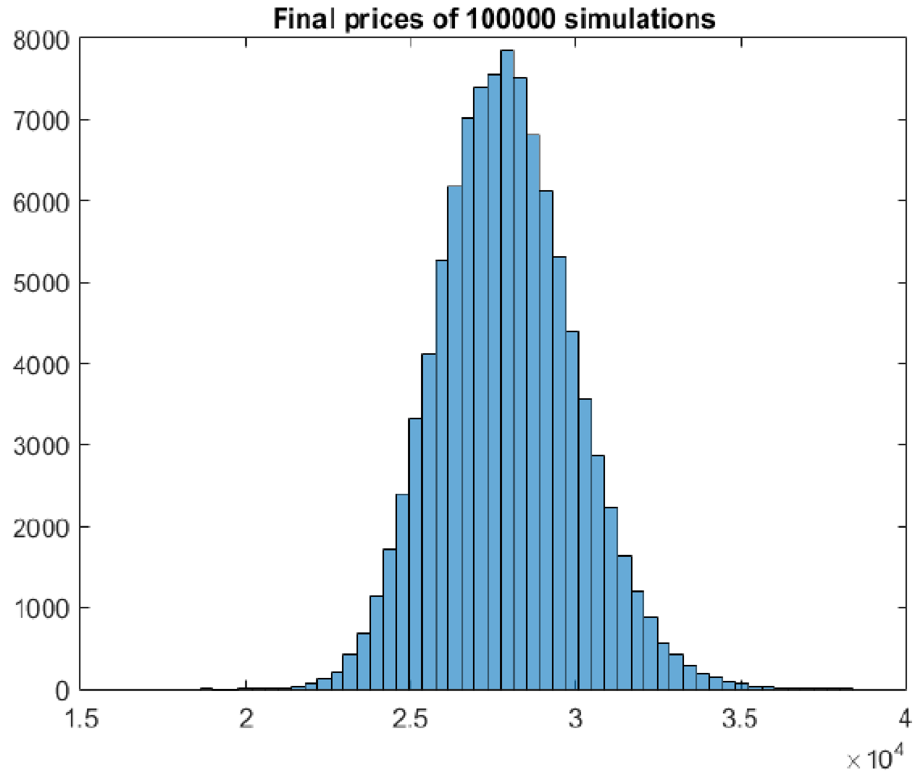


Monte Carlo Price Simulation III

1. Inspect initial code in `gdpSimVec.m`.

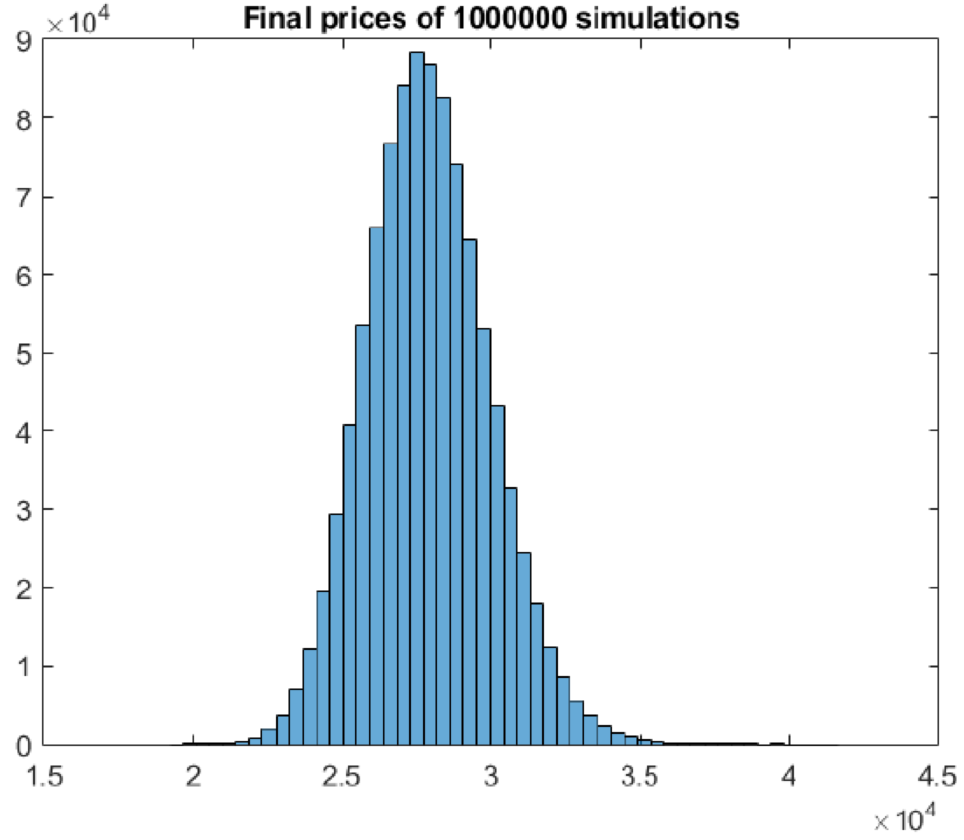
2. Accelerate the code performance by parallelizing the `for`-loop.

3. Run in parallel pool and compare timings.

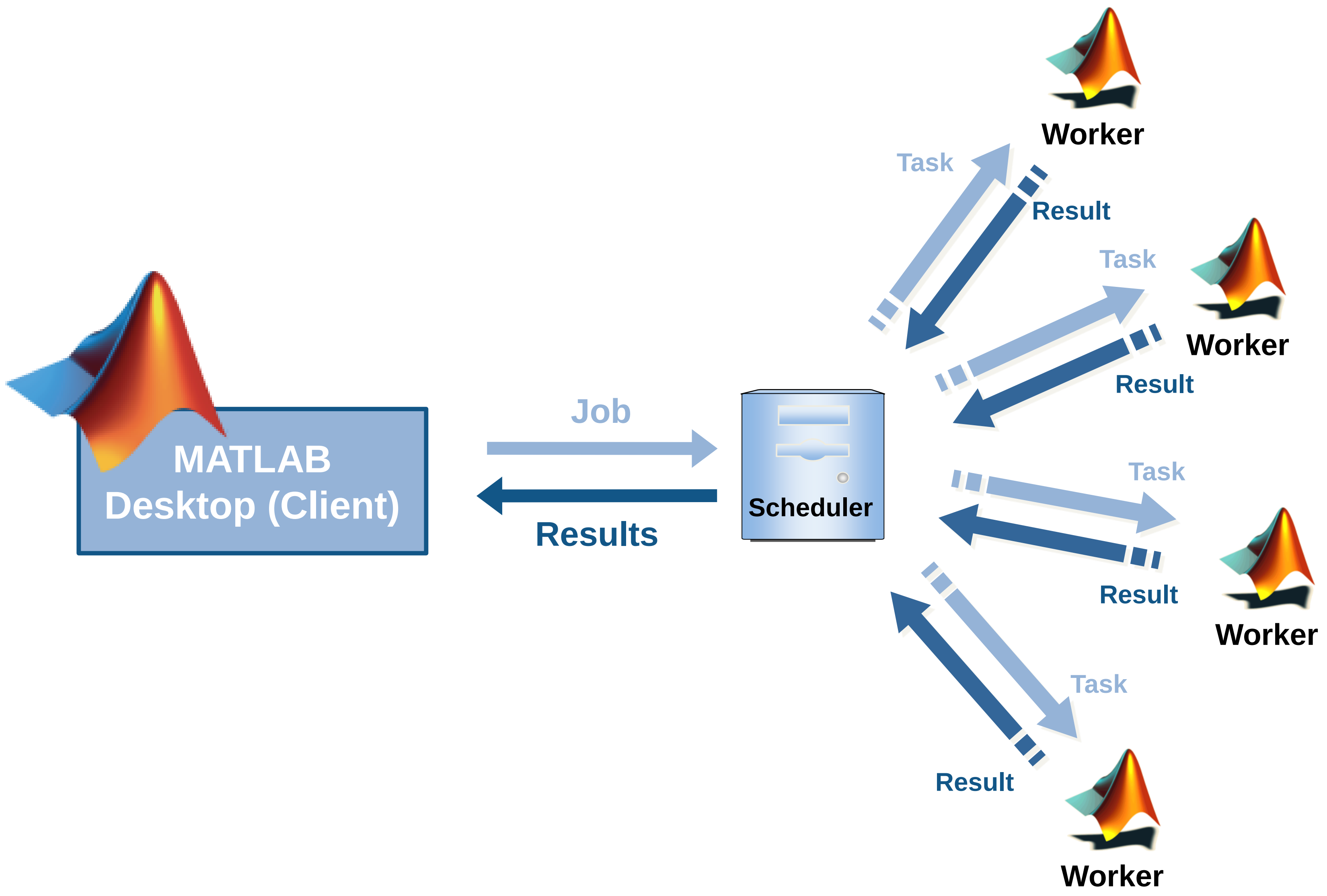


Monte Carlo Price Simulation IV

1. Inspect initial code in `gdpSimPar.m` and `gdpSimMat.m`.
2. Combine parallelization and vectorization.
3. Compare timings.



Scheduling Jobs and Tasks



Example: Scheduling different solvers on the same ODE system

```
sched = parcluster()
```

Create job

```
job = createJob(sched);
job.AutoAttachFiles = false;
myAttachedFiles= {'springDampSolver45.m', 'springDampSolver23.m', ...
                             'springDampSolverAna.m'};
job.AttachedFiles = myAttachedFiles;
```

Create tasks in job

```
task1 = createTask(job,@springDampSolver45,2,{m,k,b,totalTime});
task2 = createTask(job,@springDampSolver23,2,{m,k,b,totalTime});
task3 = createTask(job,@springDampSolverAna,2,{m,k,b,totalTime});
```

Submit job

```
submit(job)
wait(job)
```