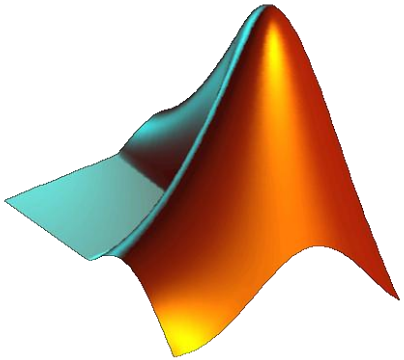
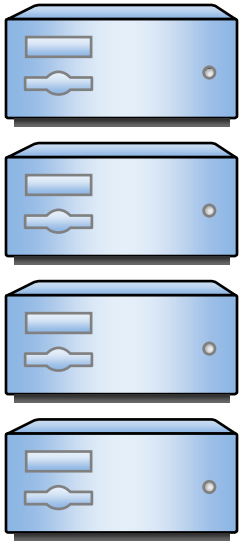
Parallel Computing with MATLAB





Sarah Wait Zaranek, PhD Senior Application Engineer

Jamie Winter Senior Account Manager

Some Questions to Consider

- Do you want to speed up your algorithms?
- Do you have datasets too big to fit on your computer?

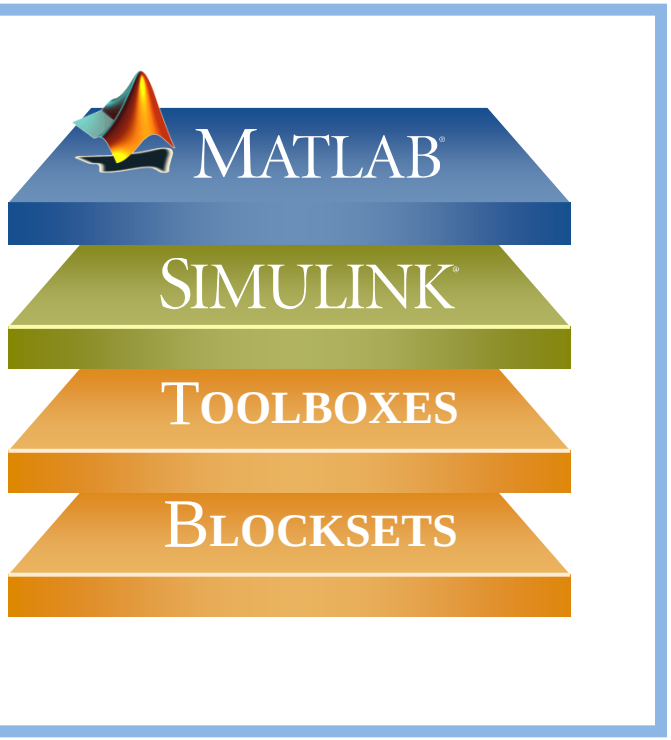
If so…

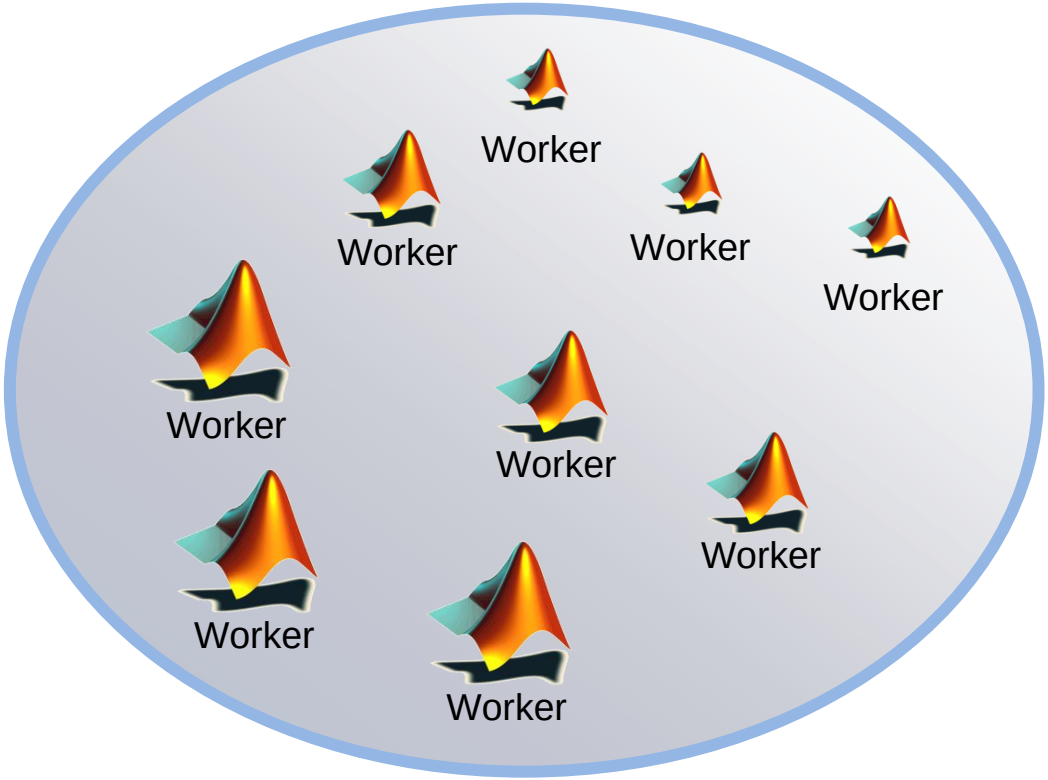
- Do you have a multicore or multiprocessor desktop machine?
- Do you have access to a computer cluster?

Solving Big Technical Problems

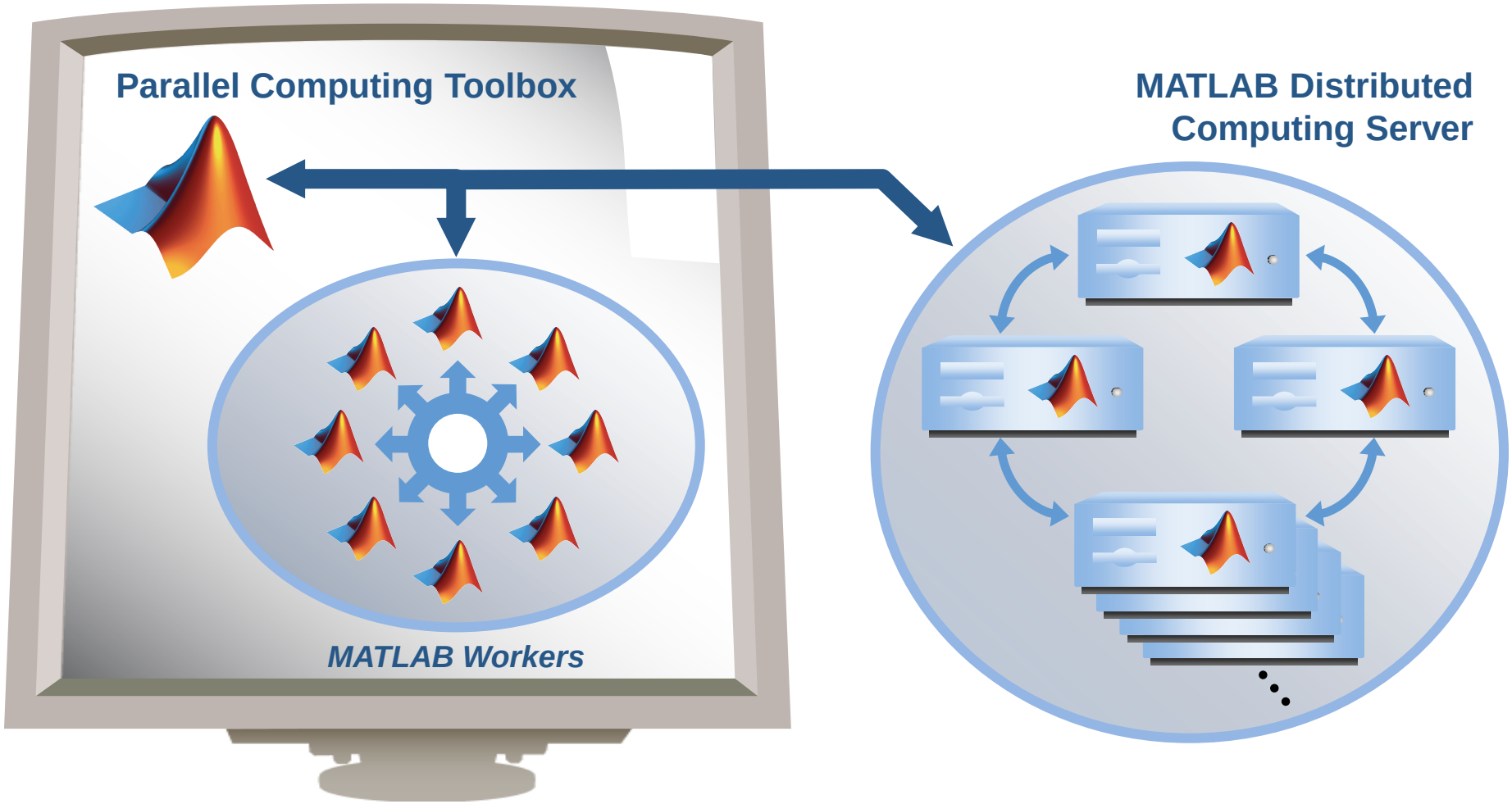
Challenges	You could…	Solutions
Long running Computationally intensive	Wait	Run similar *tasks* on independent processors in *parallel*
Large data set	Reduce size of problem	Load *data* onto multiple machines that work together in *parallel*

Parallel Computing with MATLAB





Parallel Computing with MATLAB



Programming Parallel Applications

Level of control	*Level of effort*
Minimal	**None**
Some	**Straightforward**
Extensive	**Involved**

Parallel Computing with MATLAB

Level of effort

None

- Built-in Toolbox Support

Straightforward

Task Parallel

- `parfor`
- `job` and `tasks`

Data Parallel

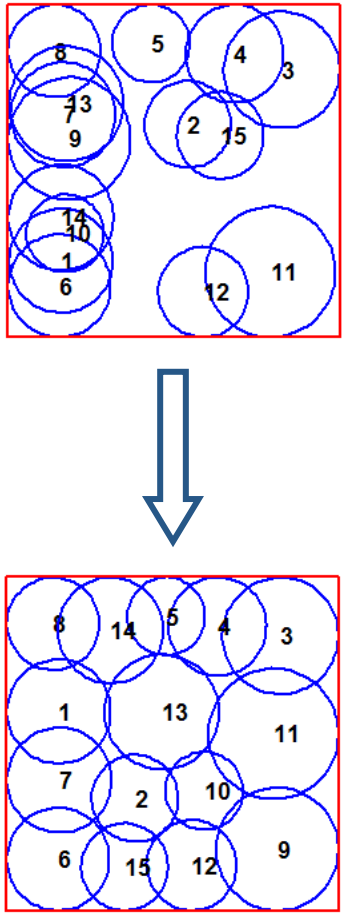
- `distributed`
- `spmd`

Involved

- MATLAB and MPI

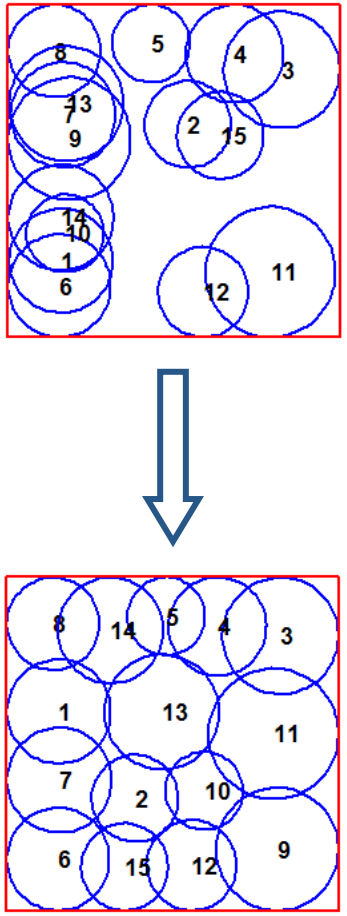
Example: Optimizing Tower Placement

- Determine location of cell towers
- Maximize coverage
- Minimize overlap



Summary of Example

- Enabled built-in support for Parallel Computing Toolbox in Optimization Toolbox
- Used a pool of MATLAB workers
- Optimized in parallel using `fmincon`



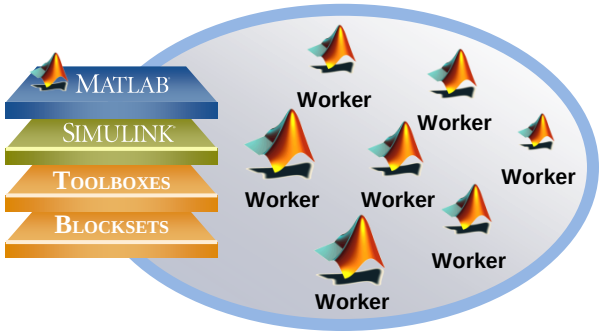
Parallel Computing Support in Optimization Toolbox

- Functions:
  - `fmincon`
    - Finds a constrained minimum of a function of several variables
  - `fminimax`
    - Finds a minimax solution of a function of several variables
  - `fgoalattain`
    - Solves the multiobjective goal attainment optimization problem

- Functions can take finite differences in parallel in order to speed the estimation of gradients

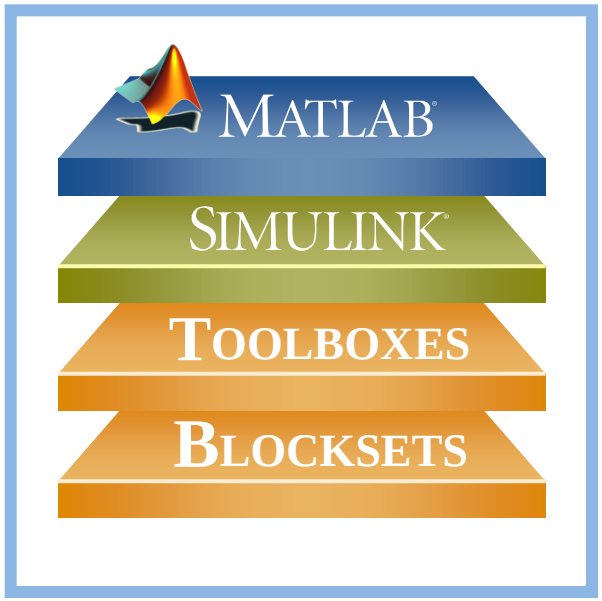
Tools Providing Parallel Computing Support

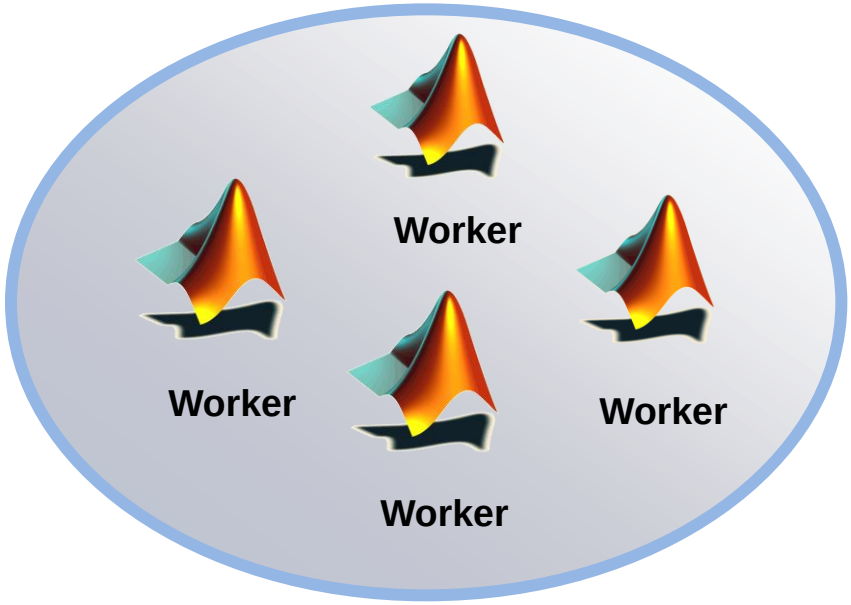
- Optimization Toolbox
- GADS Toolbox
- Statistics Toolbox
- SystemTest
- Simulink Design Optimization
- Bioinformatics Toolbox
- Model-Based Calibration Toolbox
- …



Directly leverage functions in Parallel Computing Toolbox

Task Parallel Applications

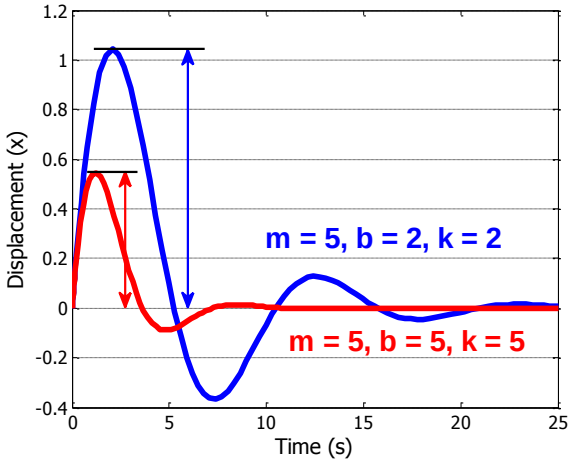




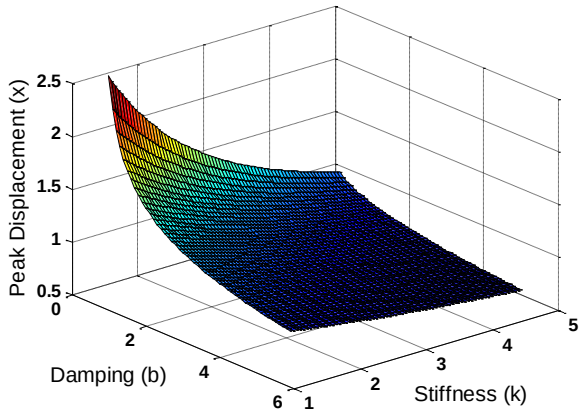
Example: Parameter Sweep of ODEs

- Solve a 2nd order ODE

$$\overbrace{m}^{5}\ddot{x} + \underbrace{b}_{1,2,\ldots}\dot{x} + \underbrace{k}_{1,2,\ldots}x = 0$$

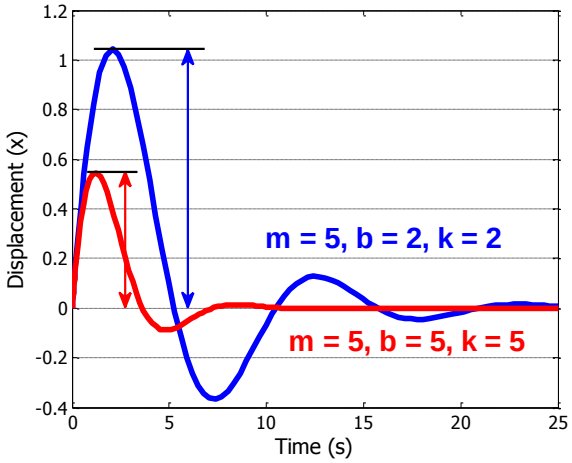


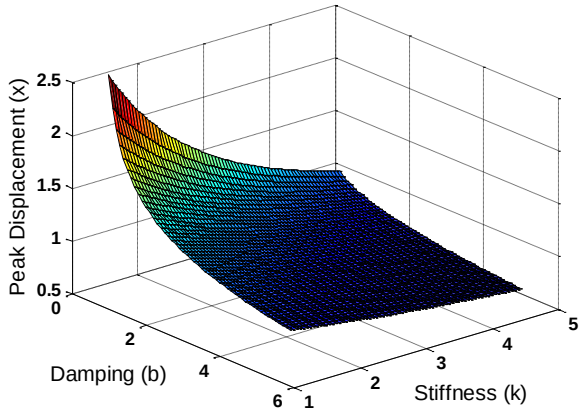
- Simulate with different values for ***b*** and ***k***
- Record peak value for each run
- Plot results



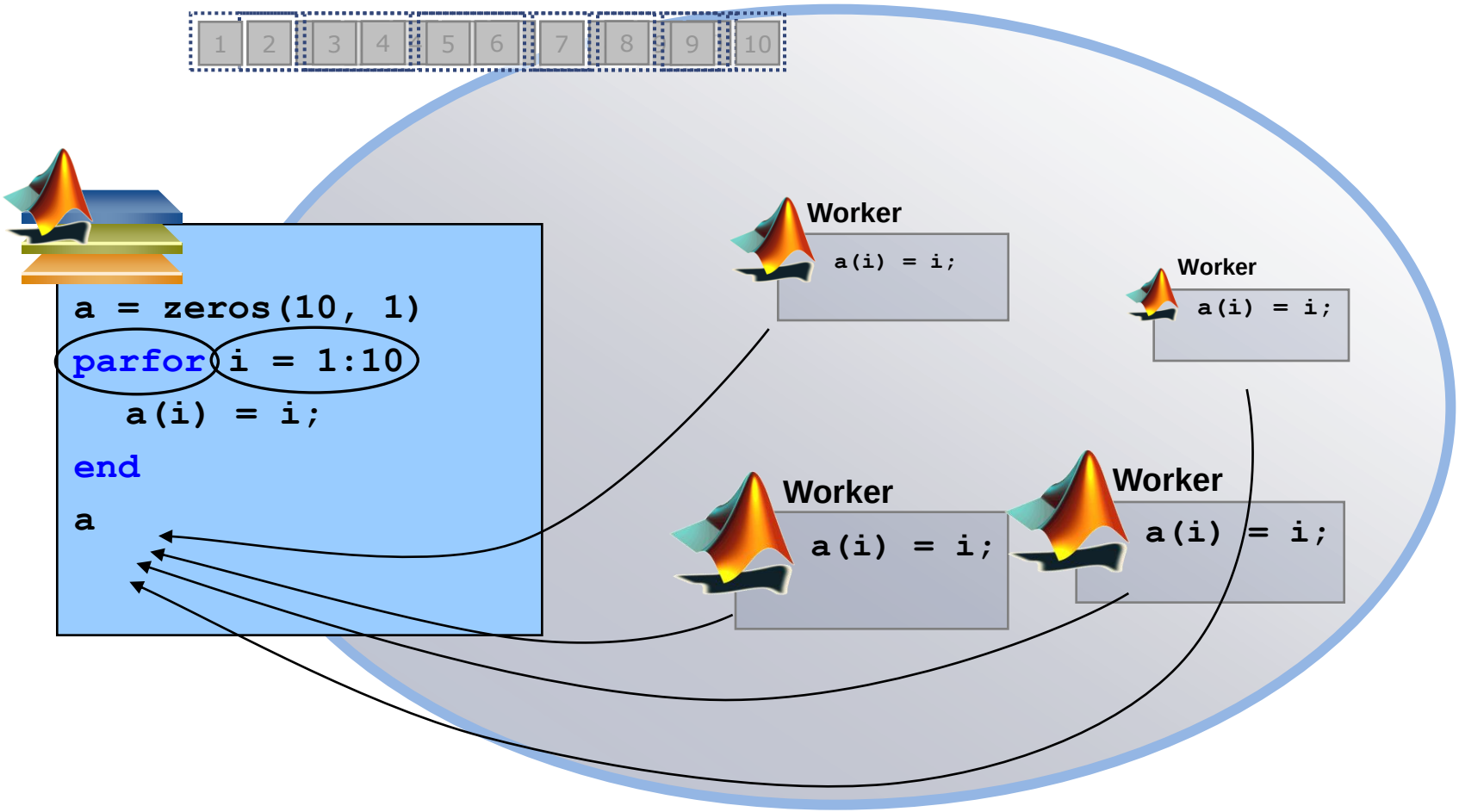
Summary of Example

- Mixed task-parallel and serial code in the same function
- Ran loops on a pool of MATLAB resources
- Used M-Lint analysis to help in converting existing `for`-loop into `parfor`-loop





The Mechanics of `parfor` Loops



Pool of MATLAB Workers

Converting `for` to `parfor`

- Requirements for `parfor` loops
  - Task independent
  - Order independent
- Constraints on the loop body
  - Cannot “introduce” variables (e.g. `eval`, `load`, `global`, etc.)
  - Cannot contain `break` or `return` statements
  - Cannot contain another `parfor` loop

Advice for Converting `for` to `parfor`

- Use M-Lint to diagnose `parfor` issues
- If your `for` loop cannot be converted to a `parfor`, consider wrapping a subset of the body to a function
- Read the section in the documentation on classification of variables
- http://blogs.mathworks.com/loren/2009/10/02/using-parfor-loops-getting-up-and-running/

Parallel Computing Tools Address…

Task-Parallel

Long computations

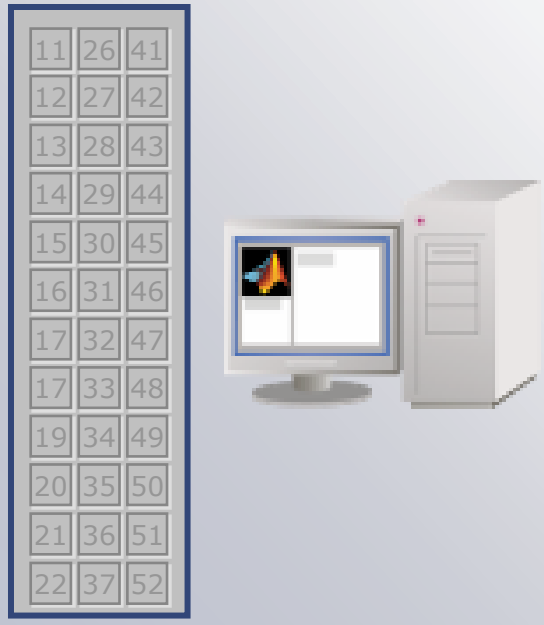
- Multiple independent iterations

```
parfor i = 1 : n
    % do something with i
end
```

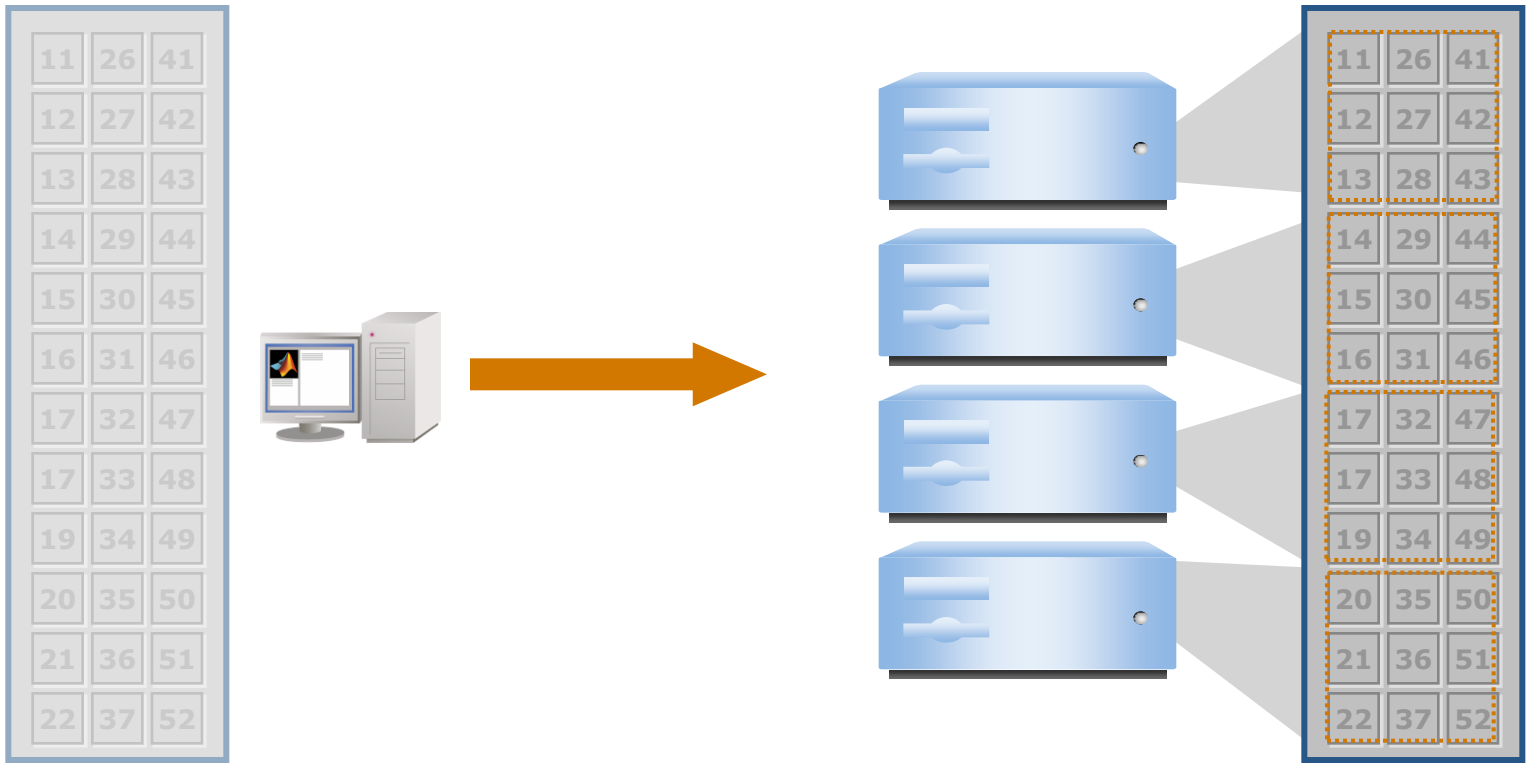
- Series of tasks

Data-Parallel

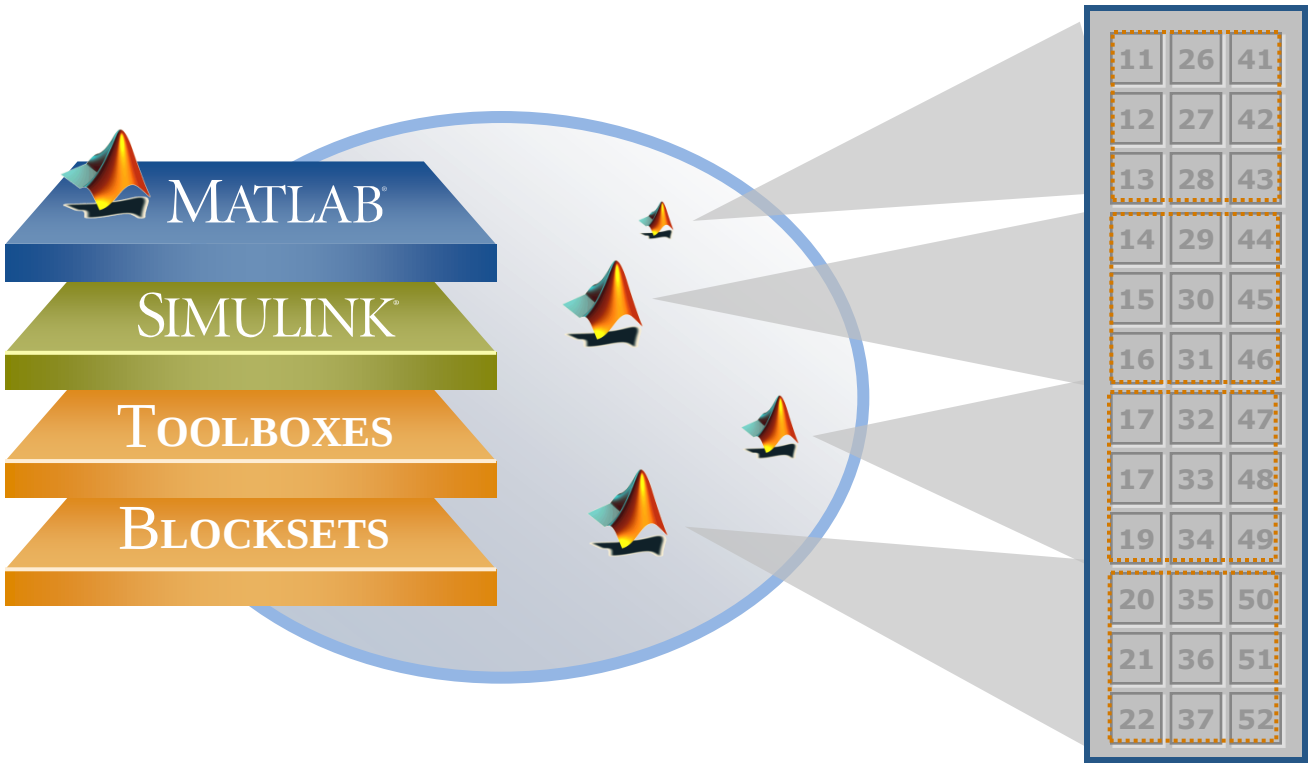
Large data problems



Large Data Sets (Data Parallel)



Client-side Distributed Arrays



Remotely Manipulate Array from Desktop

Distributed Array Lives on the Cluster

Distributed Arrays and Parallel Algorithms

- Distributed arrays
  - Store segments of data across participating workers
  - Can be created and manipulated directly from the client.
  - Create from any built-in class in MATLAB
    - Examples: doubles, sparse, logicals, cell arrays, and arrays of structs
- Parallel algorithms for distributed arrays
  - Matrix manipulation operations
    - Examples: indexing, data type conversion, and transpose
  - Parallel linear algebra functions, such as **`svd`** and **`lu`**

Enhanced MATLAB Functions That Operate on Distributed Arrays

Type of Function	Function Names
Data functions	cumprod, cumsum, fft, max, min, prod, sum
Data type functions	cast, cell2mat, cell2struct, celldisp, cellfun, char, double, fieldnames, int16, int32, int64, int8, logical, num2cell, rmfield, single, struct2cell, swapbytes, typecast, uint16, uint32, uint64, uint8
Elementary and trigonometric functions	abs, acos, acosd, acosh, acot, acotd, acoth, acsc, acscd, acsch, angle, asec, asecd, asech, asin, asind, asinh, atan, atan2, atand, atanh, ceil, complex, conj, cos, cosd, cosh, cot, cotd, coth, csc, cscd, csch, exp, expm1, fix, floor, hypot, imag, isreal, log, log10, log1p, log2, mod, nextpow2, nthroot, pow2, real, reallog, realpow, realsqrt, rem, round, sec, secd, sech, sign, sin, sind, sinh, sqrt, tan, tand, tanh
Elementary matrices	cat, diag, eps, find, isempty, isequal, isequalwithequalnans, isfinite, isinf, isnan, length, ndims, size, tril, triu
Matrix functions	chol, eig, lu, norm, normest, svd
Array operations	all, and, any, bitand, bitor, bitxor, ctranspose, end, eq, ge, gt, horzcat, ldivide, le, lt, minus, mldivide, mrdivide, mtimes, ne, not, or, plus, power, rdivide, subsasgn, subsindex, subsref, times, transpose, uminus, uplus, vertcat, xor
Sparse matrix functions	full, issparse, nnz, nonzeros, nzmax, sparse, spfun, spones
Special functions	dot

MPI-Based Functions in Parallel Computing Toolbox™

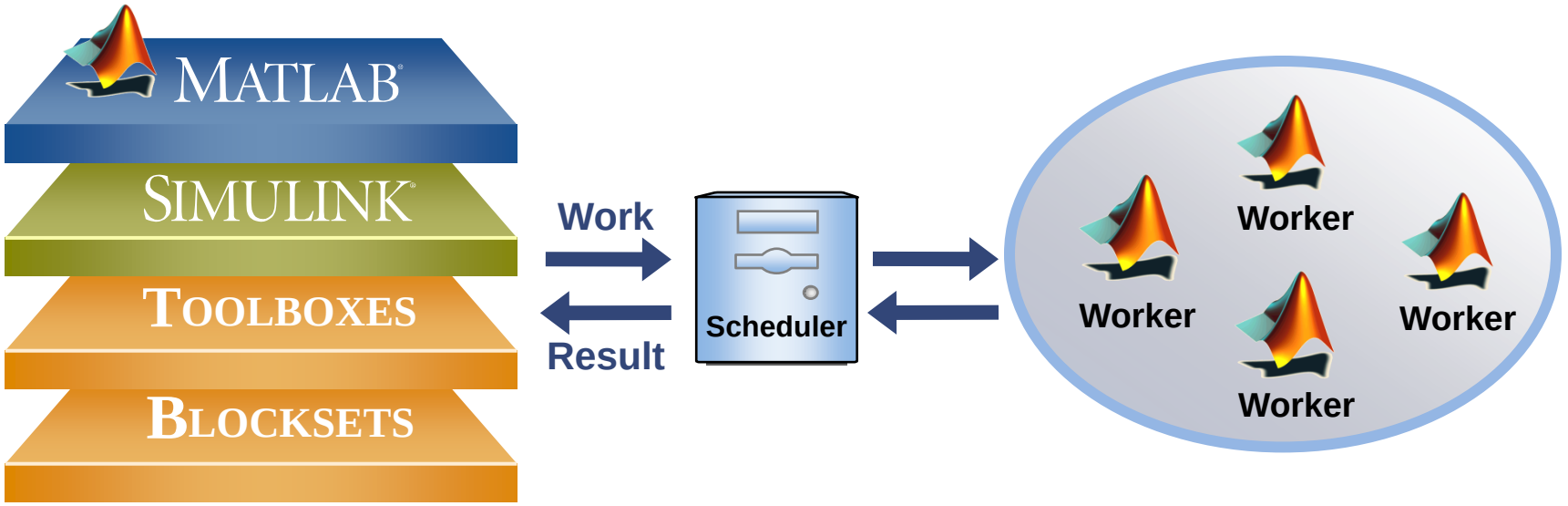
Use when a high degree of control over parallel algorithm is required

- High-level abstractions of MPI functions
  - **`labSendReceive`**, **`labBroadcast`**, and others
  - Send, receive, and broadcast any data type in MATLAB
- Automatic bookkeeping
  - Setup: communication, ranks, etc.
  - Error detection: deadlocks and miscommunications
- Pluggable
  - Use any MPI implementation that is *binary*-compatible with MPICH2

Interactive to Scheduling

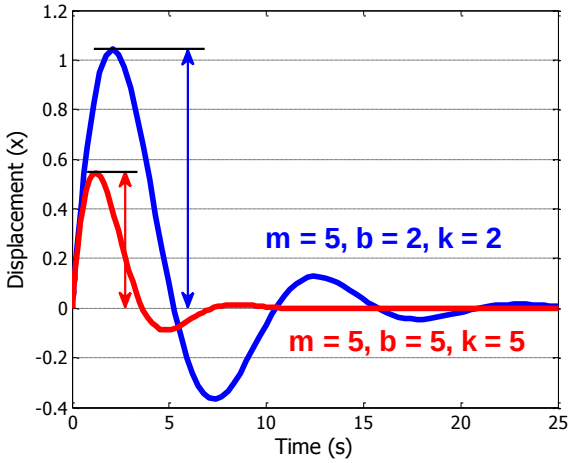
- Interactive
  - Great for prototyping
  - Immediate access to MATLAB workers
- Scheduling
  - Offloads work to other MATLAB workers (local or on a cluster)
  - Access to more computing resources for improved performance
  - Frees up local MATLAB session

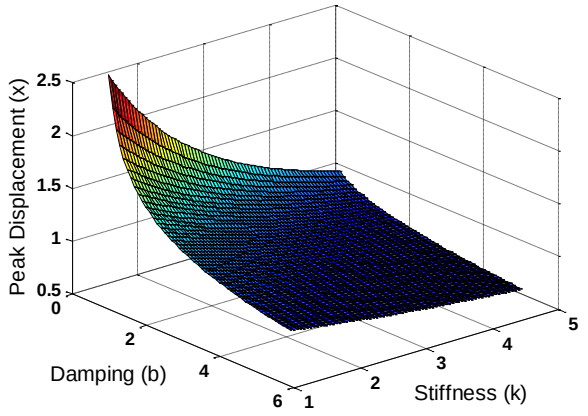
Scheduling Work



Example: Schedule Processing

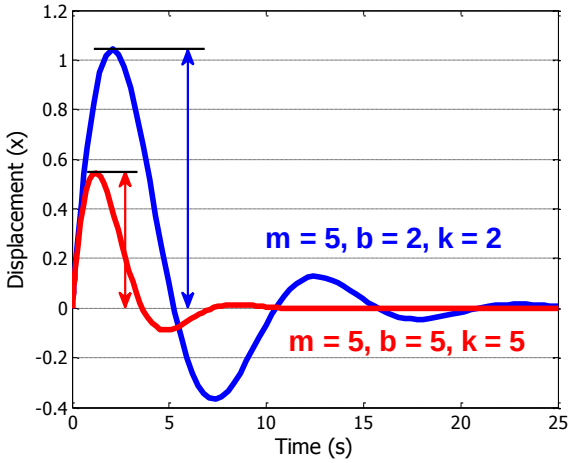
- Offload parameter sweep to local workers
- Get peak value results when processing is complete
- Plot results in local MATLAB

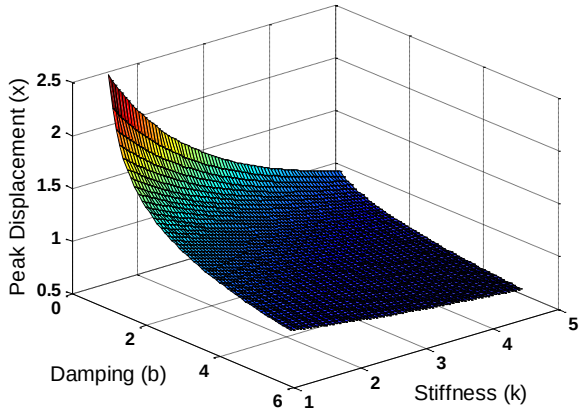




Summary of Example

- Used `batch` for off-loading work
- Used `matlabpool` option to off-load and run in parallel
- Used `load` to retrieve worker's workspace

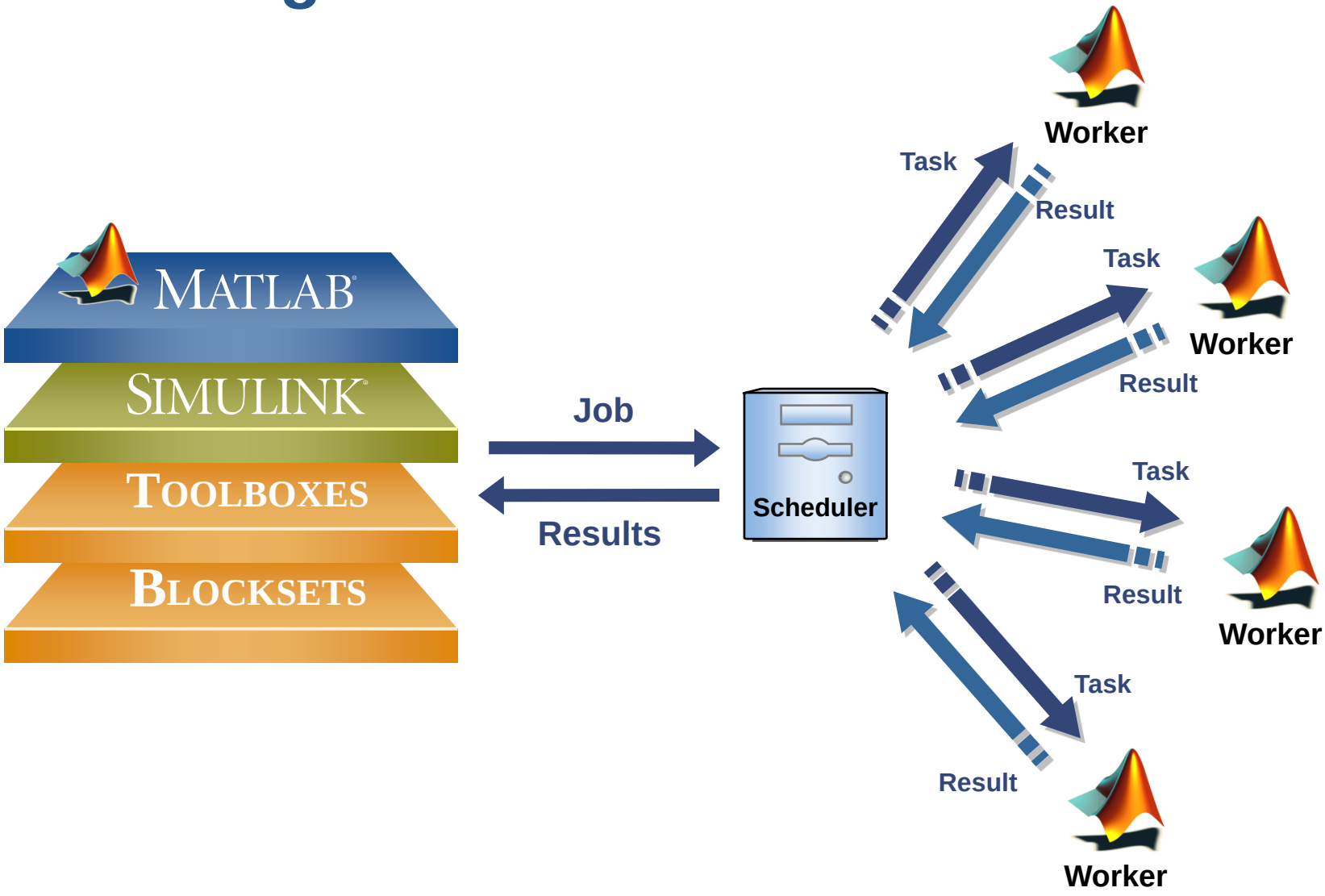




Task-Parallel Workflows

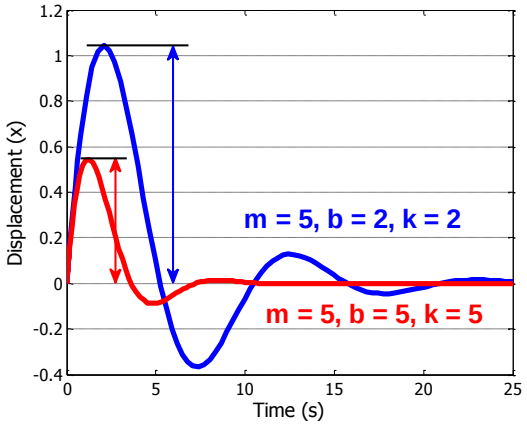
- `parfor`
  - Multiple independent iterations
  - Easy to combine serial and parallel code
  - Workflow
    - Interactive using `matlabpool`
    - Scheduled using `batch`
- `jobs/tasks`
  - Series of independent tasks; not necessarily iterations
  - Workflow ➔ Always scheduled

Scheduling Jobs and Tasks



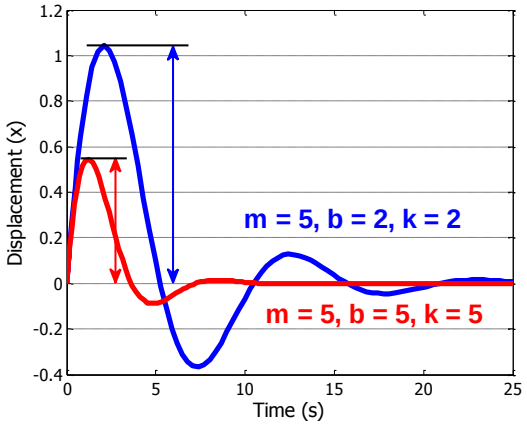
Example: Scheduling Independent Simulations

- Offload three independent approaches to solving our previous ODE example
- Retrieve simulated displacement as a function of time for each simulation
- Plot comparison of results in local MATLAB



Summary of Example

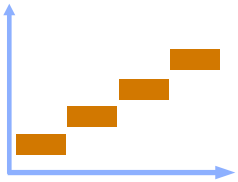
- Used `findResource` to find scheduler
- Used `createJob` and `createTask` to set up the problem
- Used `submit` to off-load and run in parallel
- Used `getAllOutputArguments` to retrieve all task outputs

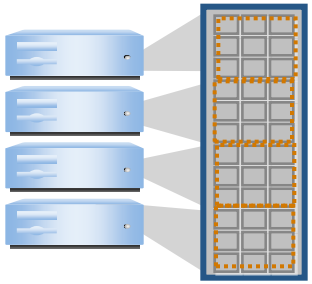


Factors to Consider for Scheduling

- There is always an overhead to distribution
  - Combine small repetitive function calls
- Share code and data with workers efficiently
  - Set job properties (`FileDependencies, PathDependencies`)
- Minimize I/O
  - Enable `Workspace` option for `batch`
- Capture command window output
  - Enable `CaptureDiary` option for `batch`

Options for Scheduling Jobs



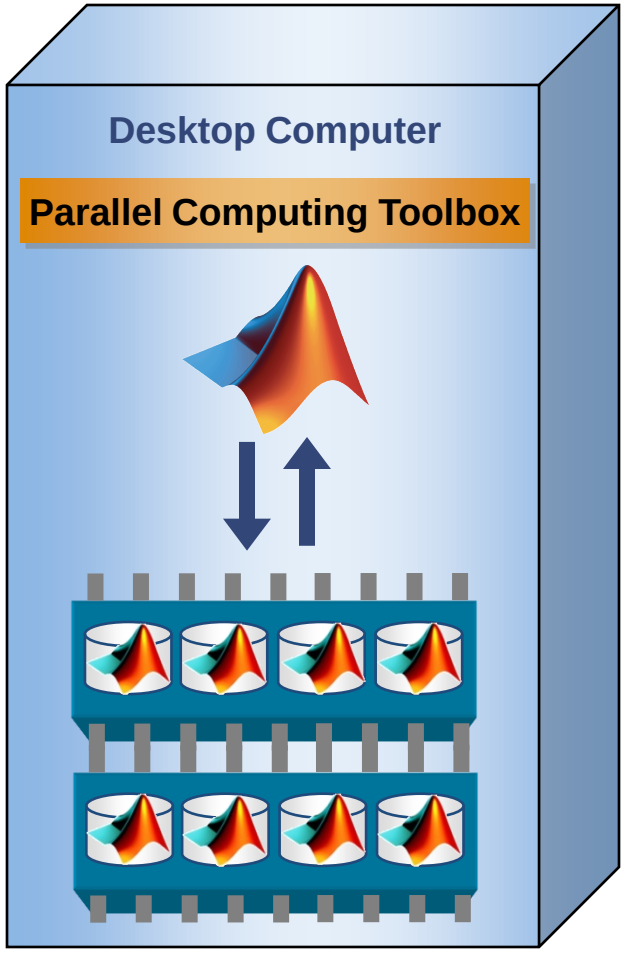


Task Parallel	Data Parallel

`>> createMatlabPoolJob`
or
`>> batch`
	`>> createMatlabPoolJob`
	or
	`>> batch`

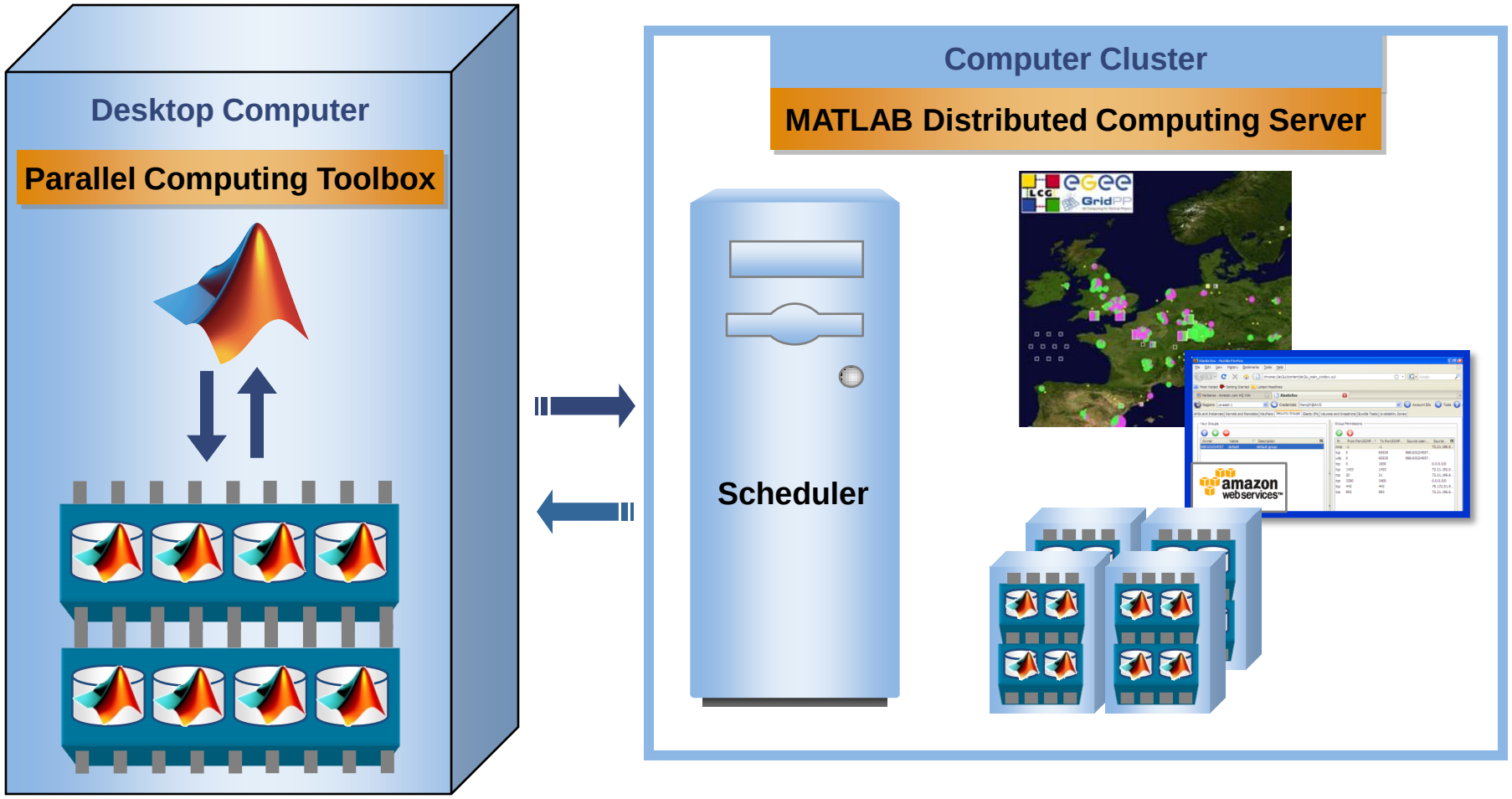
`>> createJob(...)` `>> createTask(...)`	`>> createParallelJob`

Run 8 Local Workers on Desktop



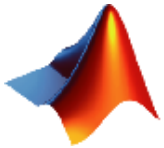
- Rapidly develop parallel applications on local computer
- Take full advantage of desktop power
- Separate computer cluster not required

Scale Up to Clusters, Grids and Clouds



Support for Schedulers

Direct Support



Platform

Microsoft Windows Compute Cluster Server 2003

Windows HPC Server 2008

TORQUE

Open API for others

GRID ENGINE

UNIVA UD

Summary

- Speed up algorithms without code changes
- Develop parallel code interactively
  - Task-parallel applications for faster processing
  - Data-parallel applications for handling large data sets
- Schedule your programs to run

MathWorks Resources Available

- Stay connected – register for a MathWorks Account on **www.mathworks.com**
- Join newsgroups, exchange links, and more at MATLAB Central – **www.mathworks.com/matlabcentral**
- Get started using MATLAB and Simulink – check out our video tutorials at **www.mathworks.com/academia/student_center/tutorials/launchpad.html**

Do you want to know more?

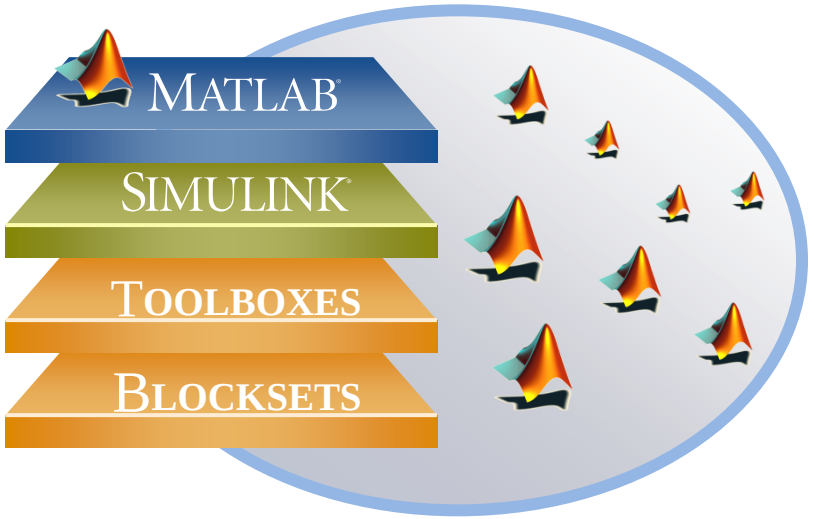
More info:

Jamie.Winter@mathworks.com

(508) 647 – 7463

Composite Arrays

- Created from desktop
- Stored on workers
- Syntax *similar* to cell arrays

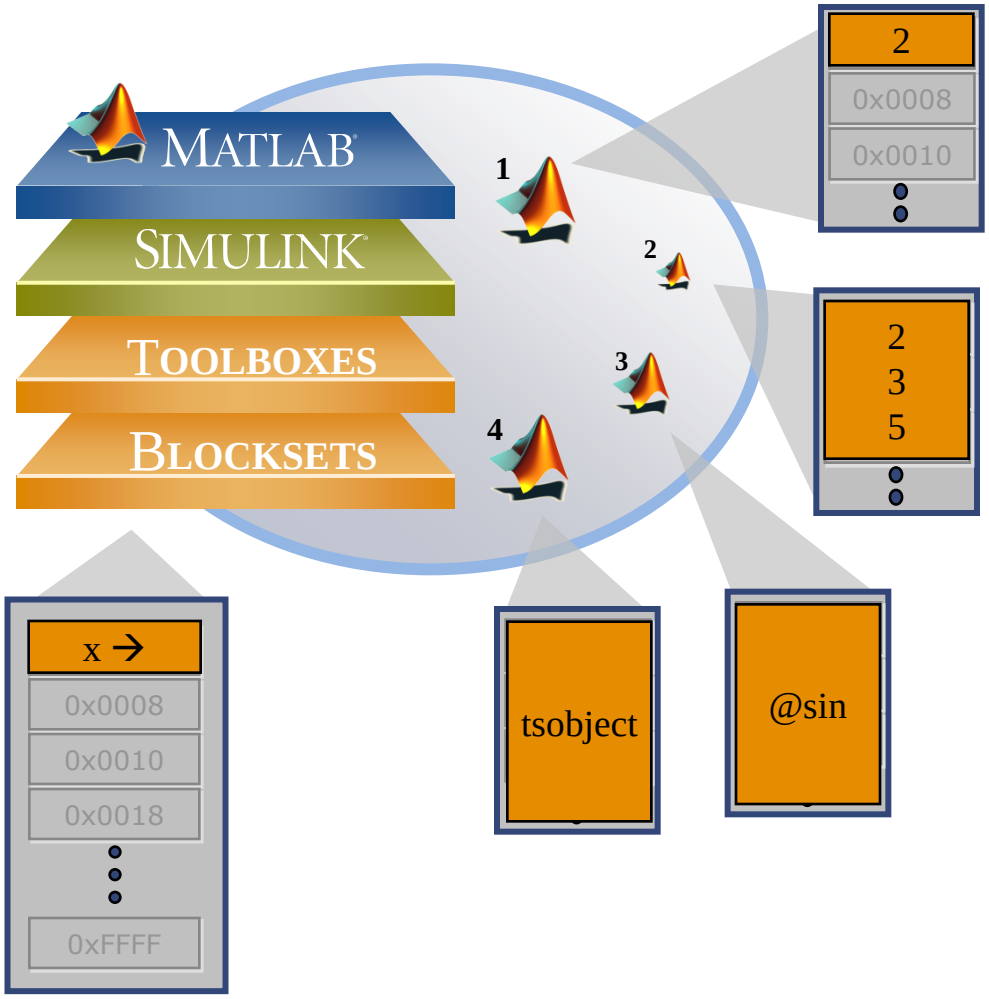


Composite Array in Memory

```
>> matlabpool open 4

>> x = Composite(4)

>> x{1} = 2
>> x{2} = [2, 3, 5]
>> x{3} = @sin
>> x{4} = tsobject()
```



A mental model for SPMD … END