Carl von Ossietzky Universität Oldenburg

Betriebseinheit für technisch-wissenschaftliche Infrastruktur

Introduction to High-Performance Computing

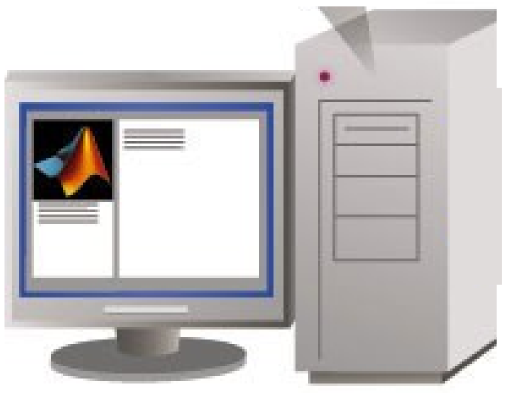
Session 08

Matlab Distributed Compute Server (MDCS)

Introduction to MDCS

(MDCS was renamed Matlab Parallel Server in 2019a)

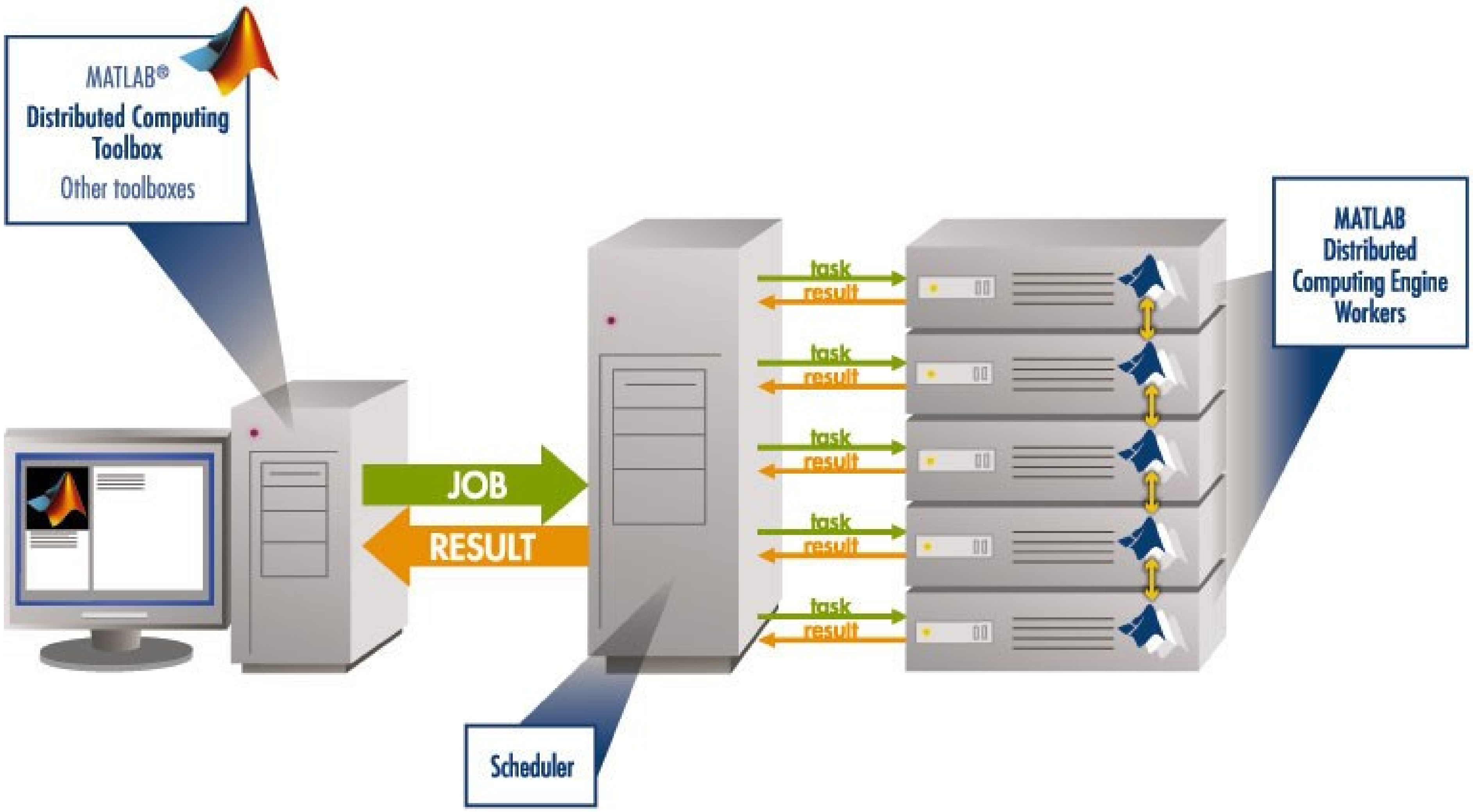
What is MDCS?



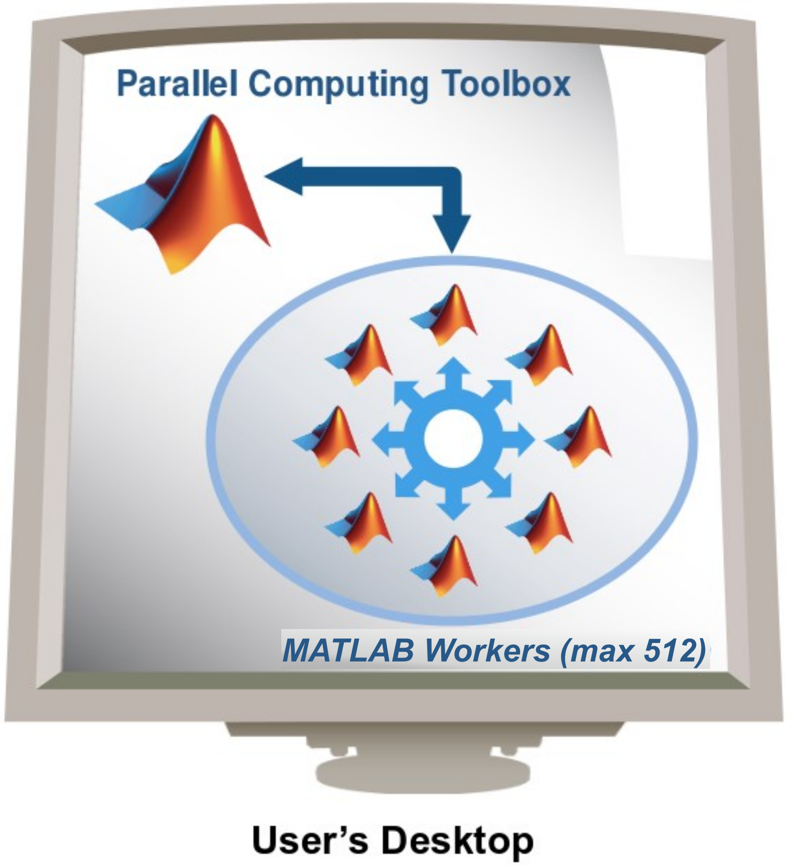
Matlab on your desktop computer:

- you are limited by the compute power of your local machine
  - memory
  - CPU speed
- you can only run one job at a time
- your machine may become unusable while your Matlab job is running

What is MDCS?


MATLAB®
Distributed Computing Toolbox
Other toolboxes
JOB
RESULT
task
result
task
result
task
result
task
result
task
result
Scheduler
MATLAB Distributed Computing Engine Workers

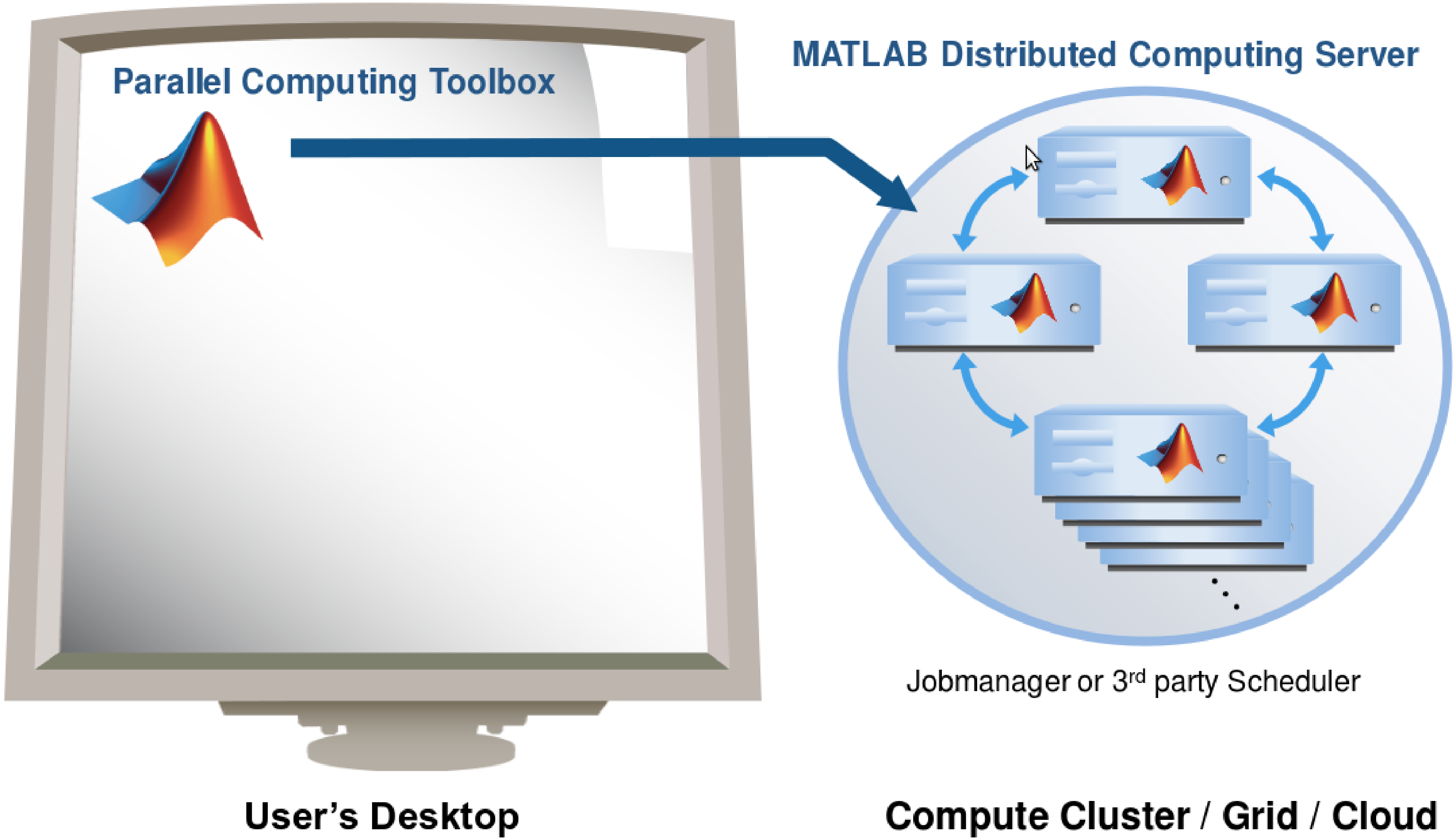
Parallel Computing with Matlab



(taken from MathWorks marketing)

- easily experiment with explicit parallelism on multicore machines
- rapidly develop parallel applications on local computer
- take full advantage of desktop power, incl. GPUs
- separate compute cluster not required

Parallel Computing with Matlab



(taken from MathWorks marketing)

What is MDCS

- MDCS allows you to off-load Matlab programs to a compute server
- simplified workflow
  - you can develop and test your application locally before submitting jobs, also in parallel
  - results are automatically returned to your local machine for post-processing
- the Parallel Computing Toolbox provides utilities for parallelization
  - task-parallel
  - data-parallel

Why to use MDCS on the Cluster?

- easy to use
  - work on your local computer within known Matlab environment
  - files (scripts, data, results) are automatically transfered
  - no need to learn about job scripts (but it helps to know a little)
- parallelization across multiple nodes
  - make use of distributed memory
  - use parallel threads (CPU cores) for each worker

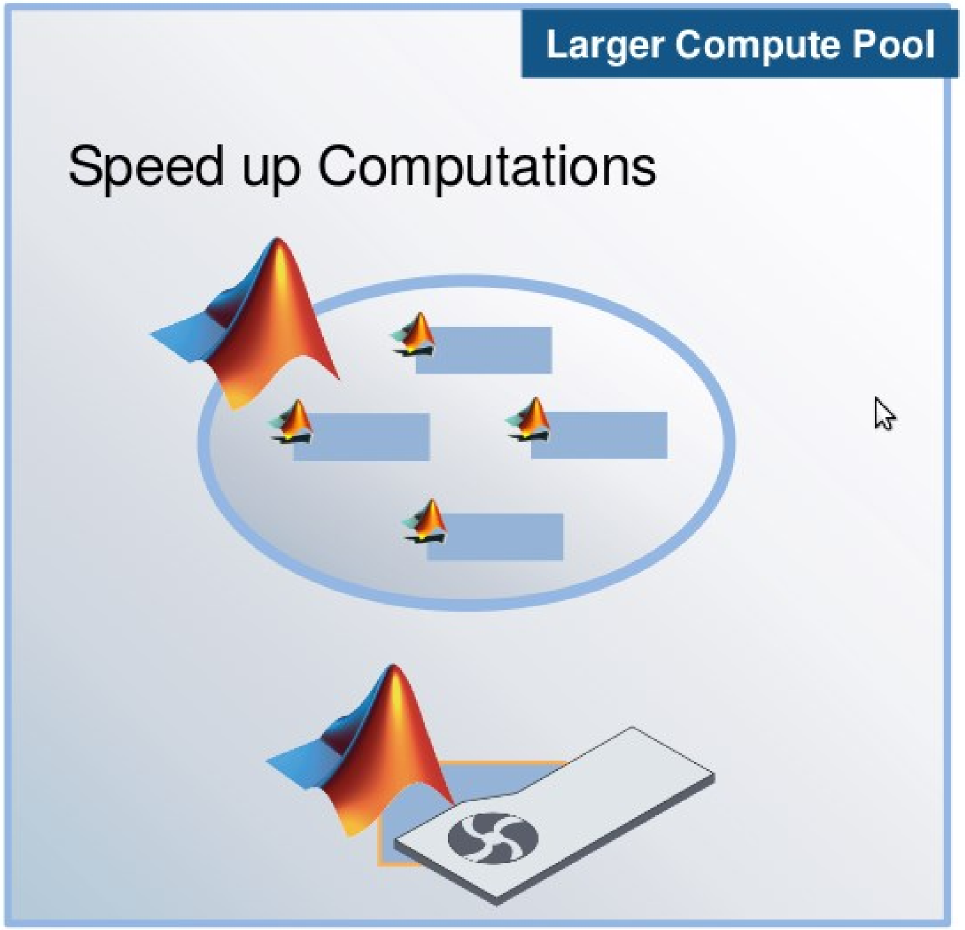
```
>> maxNumCompThreads(1);     % set the number of threads to 1
>> a = rand(4096); b = rand(4096);  % create to matrices
>> tic;c=a*b;toc             % compute and time matrix multiplication
Elapsed time is 3.633846 seconds.
>> maxNumCompThreads(4);     % set the number of threads to 4
>> tic;c=a*b;toc             % compute and time matrix multiplication
Elapsed time is 1.019613 seconds.
>> maxNumCompThreads('automatic');     % set the number of threads to automatic
>> tic;c=a*b;toc             % compute and time matrix multiplication
Elapsed time is 0.257363 seconds.
```

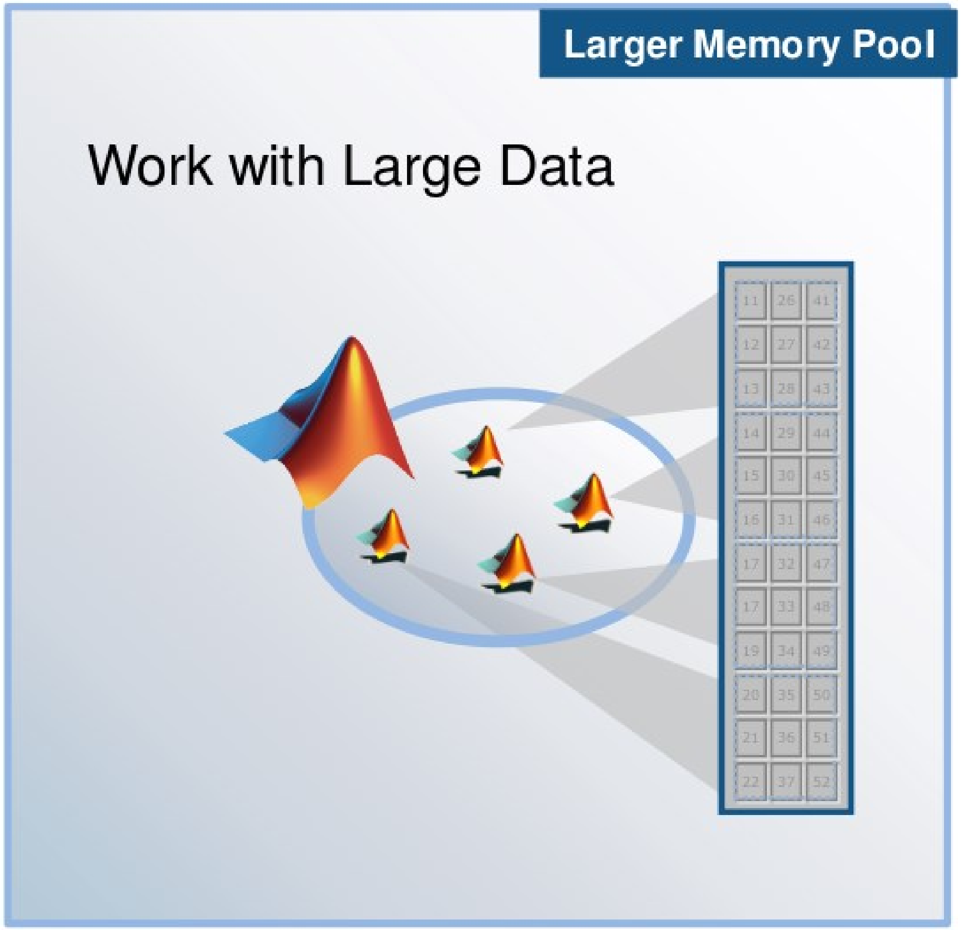
MDCS Licenses

- MDCS on the HPC cluster includes 272 worker licenses
  - Matlab used to be limited to 200 licenses, now Campus license
  - for fair sharing not more than 36 MDCS licenses should be used per job and at most two jobs per user (hard limit)
  - check license use on the cluster:

```
[abcd1234@carl]$ scontrol show license mdcs
LicenseName=mdcs
    Total=272 Used=47 Free=225 Remote=no
```

Parallel Computing with Matlab

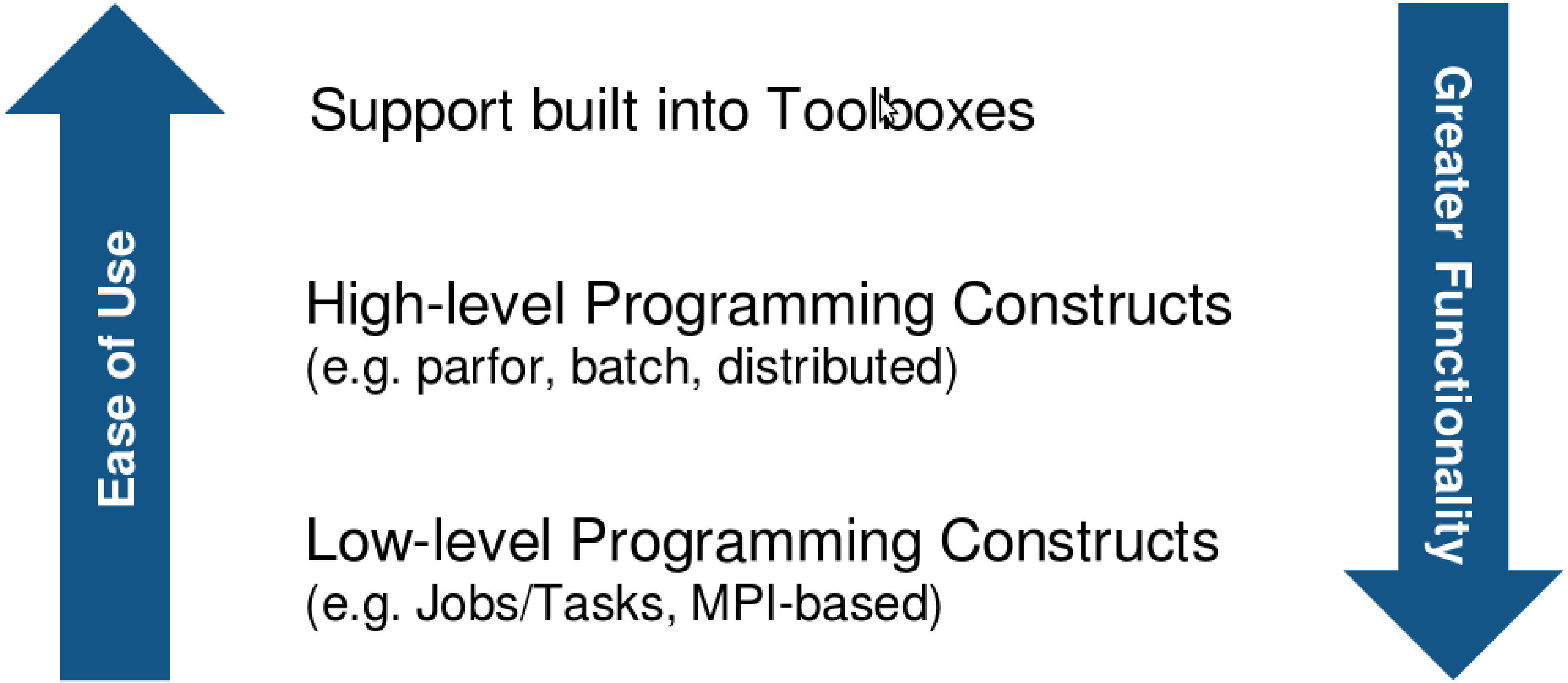




Betriebseinheit für technisch-wissenschaftliche Infrastruktur

Parallel Computing with Matlab

Three levels of Integration:



Parallel Computing Support in Toolboxes

- Optimization Toolbox
- Global Optimization Toolbox
- Statistics Toolbox
- Simulink Design Optimization
- Bioinformatics Toolbox
- Communications Toolbox
- Model-Based Calibration Toolbox
- ... and more

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

Configuration of MDCS

Using Matlab on CARL/EDDY

https://wiki.hpcuser.uni-oldenburg.de/index.php?title=MATLAB_2016

- there are three ways of running Matlab on the compute nodes:
  1. interactively with **`srun`** (or with **`srun.x11`** if you need GUI)

     ```
     $ module load MATLAB
     $ srun -p carl.p --ntasks 1 --cpus-per-task 24 matlab -nodisplay -nojvm
     >> a=rand(4096); b=rand(4096);
     >> …
     ```

  2. as a job with **`sbatch`**
     - job script contains: **`matlab -nodisplay -nojvm -batch myprogam`**
     - Matlab program provided as **`myprogram.m`**
     - use **`-r`** instead of **`-batch`** for older Matlab versions (before 2019a) and terminate program with **`quit()`**
  3. using MDCS
     - most convenient and recommended way
     - only option to use more than one compute node
     - requires configuration of local computer

Using MDCS on CARL/EDDY

- before you can use MDCS a few preparations are needed (only needed to be done once)
  - Matlab needs to be installed (see local web page) on your local machine, version must match to version on cluster (e.g. R2019b)
  - your local machine must be able to login to CARL/EDDY via ssh
    - Linux/Mac have ssh per default, for Windows you can use PuTTY
    - if you are not in the university network you also need to connect to a VPN (see HPC-Wiki for details)
  - a number of files (from a zipped archive from the HPC-Wiki) have to copied to a local directory (for older versions of Matlab you may need root/admin access for this step)
  - a parallel configuration has to be setup with Matlab

see https://wiki.hpcuser.uni-oldenburg.de/index.php?title=Configuration_MDCS_2016

Configuration of MDCS Cluster Profile

- the remote system is described in the cluster profile

`JobStorageLocation`:	local directory for job data, e.g. `C:\Users\name\Documents\MATLAB\2019b\JobData`
`RemoteJobStorageLocation`:	local directory for job data, e.g. on `$WORK` `/gss/work/abcd1234/MATLAB/2019b/JobData`

- directories are sync'd at job submission and after the job has completed
- existing workspace is copied at job submission (can affect submission time)
- workspace of main process is copied back (can affect job load time), use e.g. `clear bigvar1 bigvar2;` (and save in separate files if needed)

Configuration of MDCS Cluster Profile

- the remote system is described in the cluster profile

`JobStorageLocation`:	local directory for job data, e.g. `C:\Users\name\Documents\MATLAB\2019b\JobData`
`RemoteJobStorageLocation`:	local directory for job data, e.g. on **`$WORK`** `/gss/work/abcd1234/MATLAB/2019b/JobData`
`NumWorkers`:	set to 36 for fair sharing
`NumThreads`:	set to 1 (default), can be changed when useful

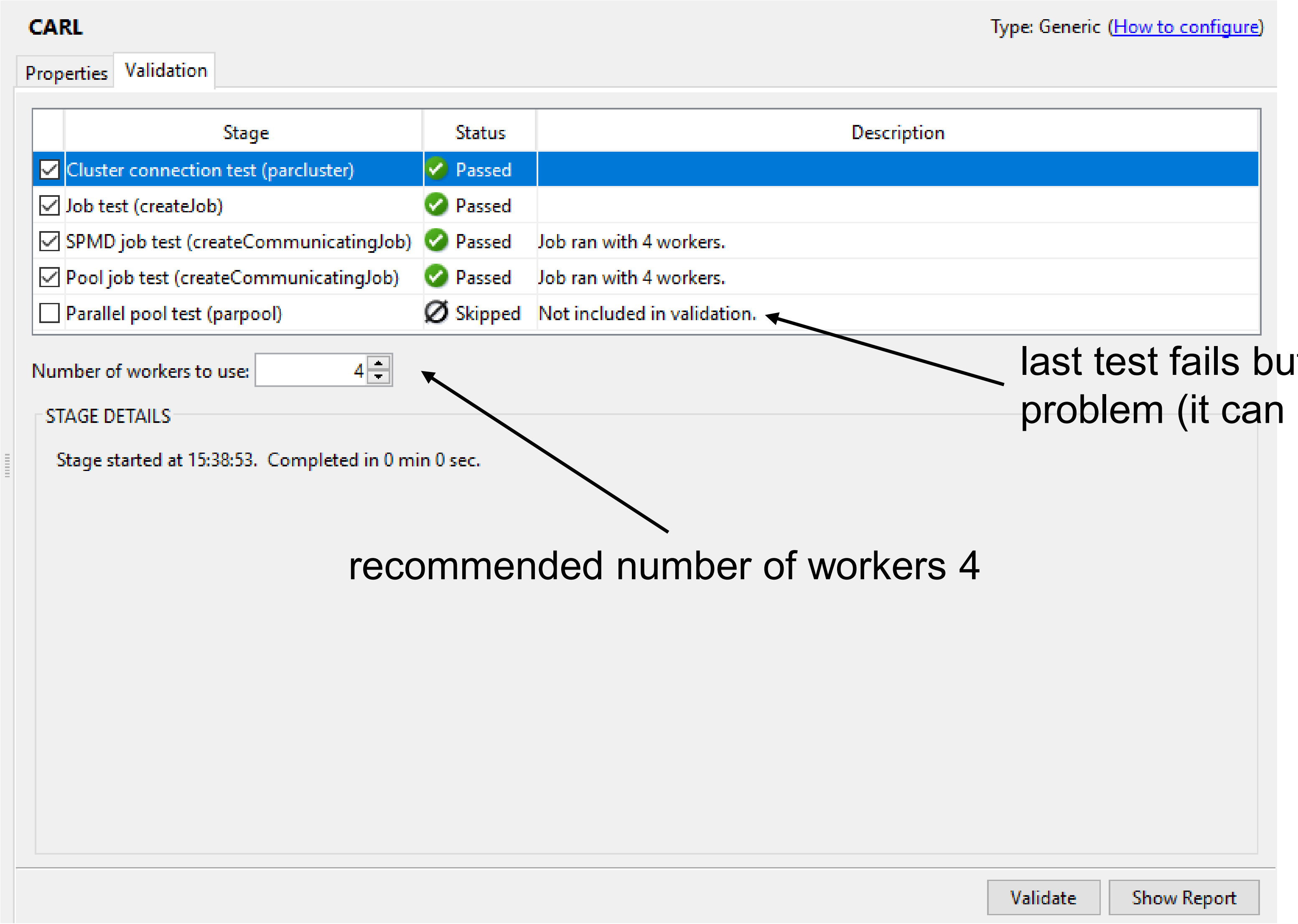
  - change with e.g.: **`sched.NumThreads=4;`**
  - maximum number of threads is the number of CPU cores in a node
  - total number of cores allocated is **`(worker+1)*NumThreads`**
  - benchmark your code to determine a good number of threads per worker.

Configuration of MDCS Cluster Profile

- the remote system is described in the cluster profile

`JobStorageLocation`:	local directory for job data, e.g. **`C:\Users\name\Documents\MATLAB\2019b\JobData`**
`RemoteJobStorageLocation`:	remote directory for job data, e.g. on **`$WORK`** **`/gss/work/abcd1234/MATLAB/2019b/JobData`**
`NumWorkers`:	set to 36 for fair sharing
`NumThreads`:	set to 1 (default), can be changed when useful
`AdditionalProperties`:	set at least **`ClusterHost`** and **`RemoteJobStorageLocation`** (see above), addtional options for password-free login are described in HPC wiki

Validation of MDCS Cluster Profile



last test fails but that is not a problem (it can be skipped)

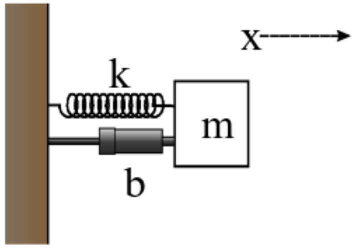
recommended number of workers 4

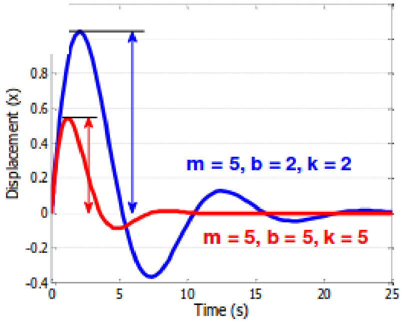
Basic Example for Using MDCS

Using MDCS on CARL/EDDY

- once you have completed the setup you can submit jobs to the cluster
  - example parameter sweep for 2^{nd}-order ODE (taken from the HPC-Wiki)
  - dampened oscillator

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





  - simulate with different values for *b* and *k*
  - record peak value for each run

Betriebseinheit für technisch-wissenschaftliche Infrastruktur

2^{nd}-order ODE for example

odesystem.m

```
function dy = odesystem(t, y, m, b, k)
% 2nd-order ODE
%
%   m*X'' + b*X' + k*X = 0
%
% --> system of 1st-order ODEs
%
%   y  = X'
%   y' = -1/m * (k*y + b*y')
% Copyright 2009 The MathWorks, Inc.

dy(1) = y(2);
dy(2) = -1/m * (k * y(1) + b * y(2));

dy = dy(:); % convert to column vector
```

Parameter Sweep: serial Matlab code

paramSweep_batch.m

```
%% Initialize Problem
m      =         5;  % mass
bVals = 0.1:.1:15;  % damping values (step .1)
kVals = 1.5:.1:15;  % stiffness values (step .1) damping
[kGrid, bGrid] = meshgrid(bVals, kVals);
peakVals = nan(size(kGrid));

%% Parameter Sweep
tic;

for idx = 1:numel(kGrid)
  % Solve ODE
  [T,Y] = ode45(@(t,y) odesystem(t, y, m, bGrid(idx), kGrid(idx)), ...
    [0, 25], ...  % simulate for 25 seconds
    [0, 1]);      % initial conditions

  % Determine peak value
  peakVals(idx) = max(Y(:,1));
end

t1 = toc;
```

Parameter Sweep: parallel Matlab code

paramSweep_batch.m

```
%% Initialize Problem
m     =          5;  % mass
bVals = 0.1:.1:15;  % damping values (step .1)
kVals = 1.5:.1:15;  % stiffness values (step .1) damping
[kGrid, bGrid] = meshgrid(bVals, kVals);
peakVals = nan(size(kGrid));

%% Parameter Sweep
tic;

parfor idx = 1:numel(kGrid)
  % Solve ODE
  [T,Y] = ode45(@(t,y) odesystem(t, y, m, bGrid(idx), kGrid(idx)), ...
    [0, 25], ...   % simulate for 25 seconds
    [0, 1]);       % initial conditions

  % Determine peak value
  peakVals(idx) = max(Y(:,1));
end

t1 = toc;
```

Using MDCS on CARL/EDDY

- submitting jobs to the cluster

```
>> sched = parcluster('CARL');
>> job = batch(sched, 'paramSweep_batch', 'Pool', 7, ...
               'AttachedFiles', {'odesystem.m'});
```

  - first command creates a handle **`sched`** for the cluster using the available configuration
  - second command creates a job and sends it to the cluster
    - Matlab script is executed on the cluster
    - requests a pool of workers (number of processes is +1 for master)
    - uses default resources unless modified
    - files can be attached explicitly but Matlab also automatically attaches needed files (if it can find them and if not disabled)
    - job handle **`job`** contains additional information

Using MDCS on CARL/EDDY

- checking the status of a job

```
>> job.State
```

  - answer can be e.g. 'queued', 'running', or 'finished'
  - alternatively, use the job monitor

- retrieving the results from a completed job

```
>> jobData = load(job);
```

  - the structure **jobData** holds the workspace from the main process
  - further processing can be done locally, e.g. creating a plot

```
>> figure;
>> f=surf(jobData.bVals, jobData.kVals, jobData.peakVals);
>> set(f,'LineStyle','none');
>> set(f,'FaceAlpha',0.5);
>> xlabel('Damping'); ylabel('Stiffness'); zlabel('Peak Displacement');
>> view(50, 30);
```

Using MDCS on CARL/EDDY

- changing resource allocation

```
> sched.AdditionalProperties.runtime='0:30:00';
> sched.AdditionalProperties.memory='4G';
> remove(sched.AdditionalProperties, 'memory');
```

  - changes maximum runtime and memory per worker
  - remove previous setting to get default
  - older Matlab versions use a different format (see HPC wiki)

- path-dependency as alternative to attaching files
  - use addpath within script (.m-files)
  - use AdditionalPath property of scheduler object
  - use absolute path names
  - copy files to the cluster before submitting job

Using MDCS on CARL/EDDY

- recovering jobs
  - it is possible to terminate the local Matlab session while jobs are running (or waiting on the cluster)
  - to reconnect

```
>> sched   = parcluster('CARL');
>> sched.Jobs                     % to list available jobs
>> job     = sched.Jobs(1)        % to get job information
>> jobData = load(job);
```

- deleting jobs permanently

```
>> delete(sched.Jobs(1));         % delete first job in list
>> delete(sched.Jobs);            % delete all jobs in list
```

  - careful, this removes files from your local computer and cannot be undone
  - you can also use the Job Monitor for this

Monitoring Jobs and Error Tracking

- Matlab Job Monitor for basic information
  - may show warnings and/or errors
  - in the basic example a warning is shown: **`"<Dir>" not found in path`**, can be avoided by adding **`'AutoAddClientPath', false`** to the **`batch-command`**
- use **`squeue`** and **`sacct`** for additional information from SLURM
- job handle can be used to get information about errors
- Matlab diary for additional log output
- files in the job directory