

Distributed Computing in the Engineering Workflow

Loren Dean

Director of Engineering, MATLAB Products
The MathWorks

MathWorks

Aerospace and Defense Conference '07



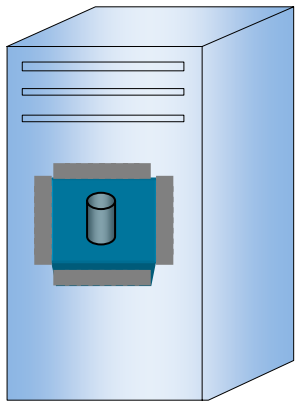
Agenda



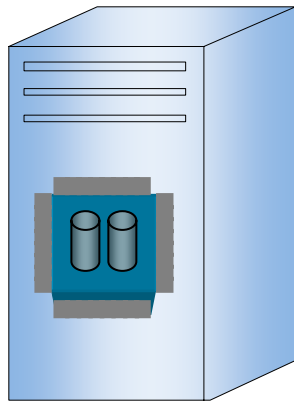
An important trend impacting the Engineering workflow

- Task parallel applications
- Data parallel applications

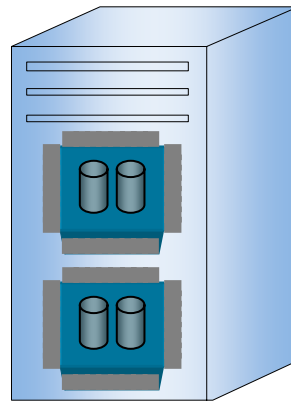
Market trend: from single processor to grids



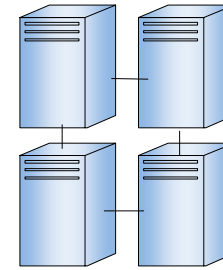
Single processor



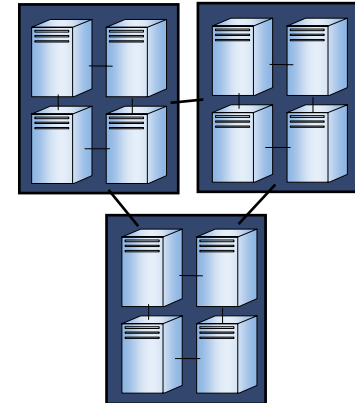
Multi-core



Multiprocessor



Clusters



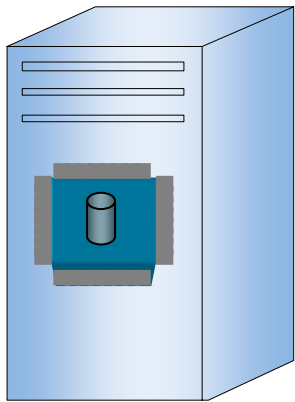
Grids

Why is this important?

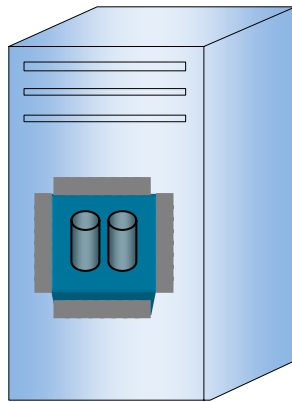
- Technical computing
 - Modeling and analysis involves numerous runs
 - Monte carlo or similar applications very common
 - Complexity of algorithms causes longer execution times
 - Data sets are increasing in size

- Model-based design
 - Simulation is done prior to real-world implementation
 - Many scenarios tested
 - Optimal solutions can be found earlier
 - Simulations are growing in complexity and size
 - Simulation time increases
 - Rsim and other targets are only part of the solution

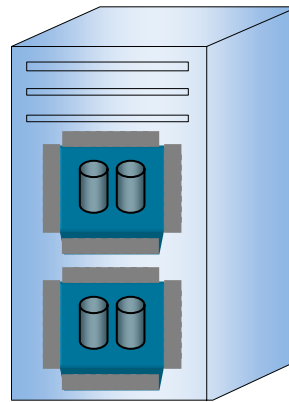
MATLAB addresses the market trend



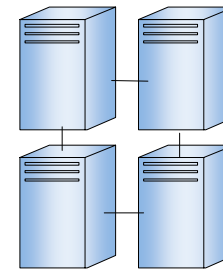
Single processor



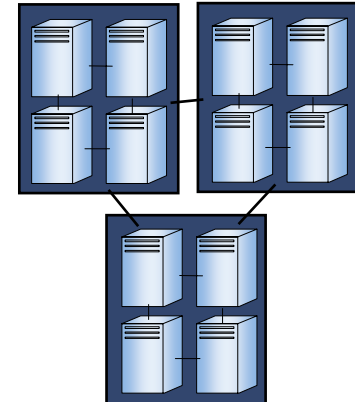
Multi-core



Multiprocessor



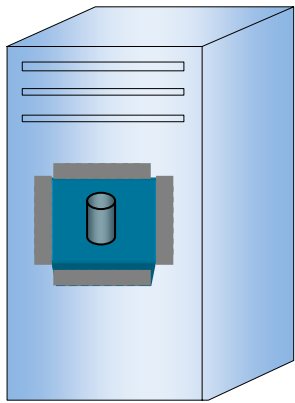
Clusters



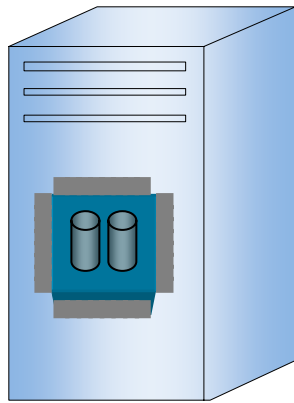
Grids

**MATLAB
in R2006b**

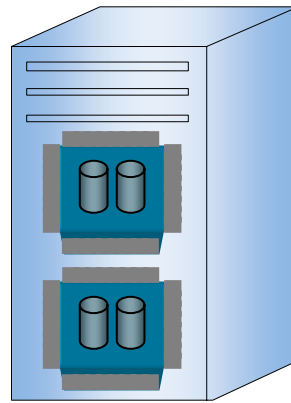
MATLAB addresses the market trend



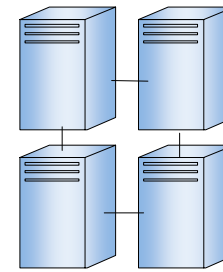
Single processor



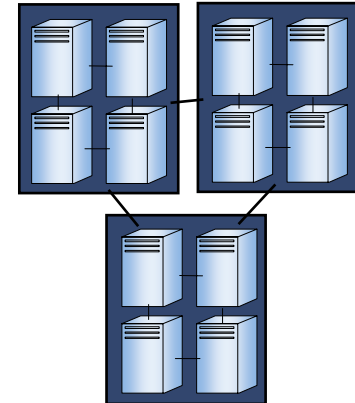
Multi-core



Multiprocessor



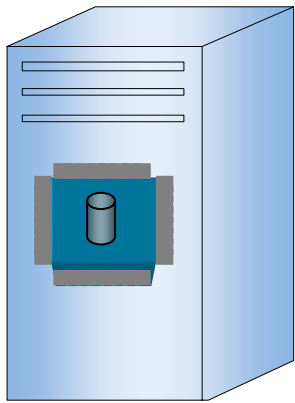
Clusters



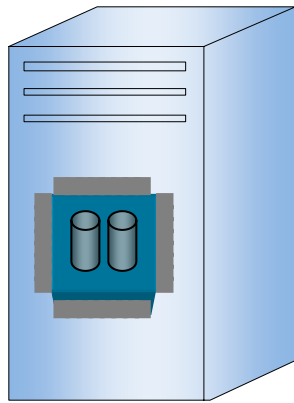
Grids

MATLAB
in R2007a

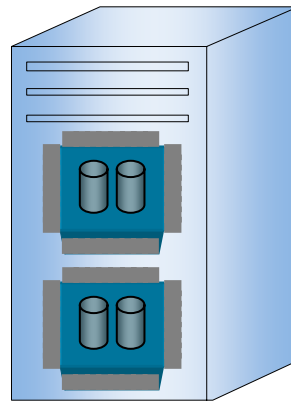
MATLAB to Distributed Computing



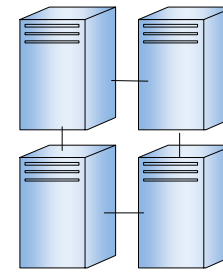
Single processor



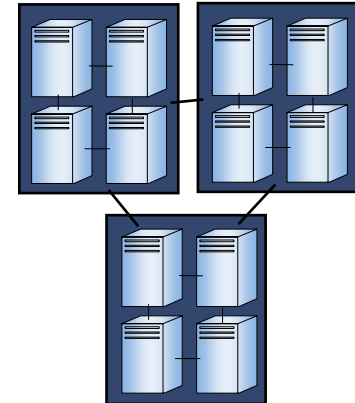
Multi-core



Multiprocessor



Clusters



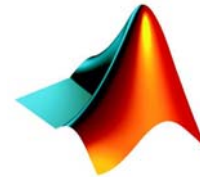
Grids

Distributed Computing Toolbox
in R2007a

MATLAB Distributed
Computing Engine

What is happening because of this trend?

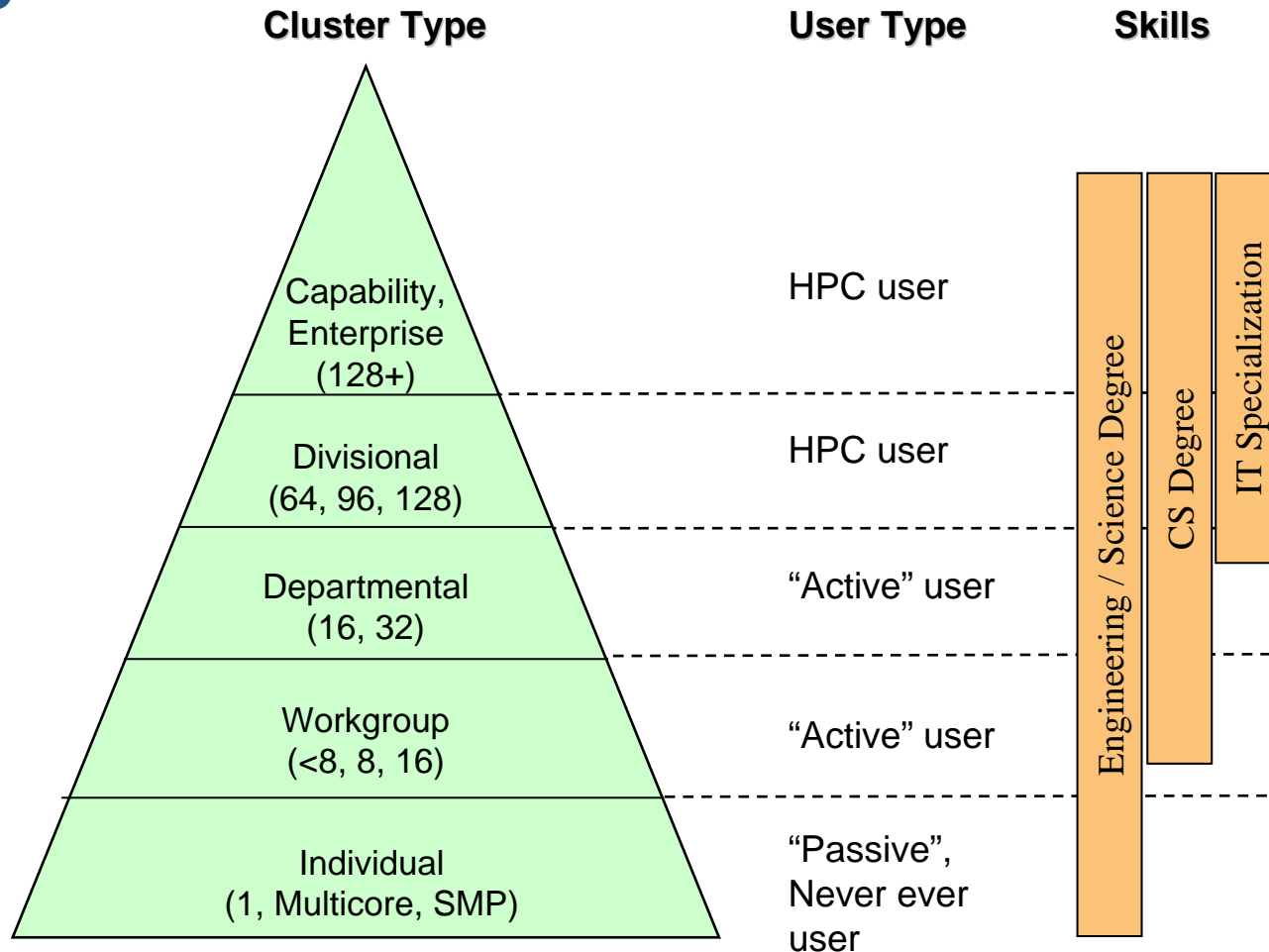
<p>Interactive programming - re-use of existing applications</p>
<p>Distributed/parallel engineering software tools becoming available</p>
<p>Schedulers and Operating Systems providing foundation services</p>
<p>Low-cost hardware</p>



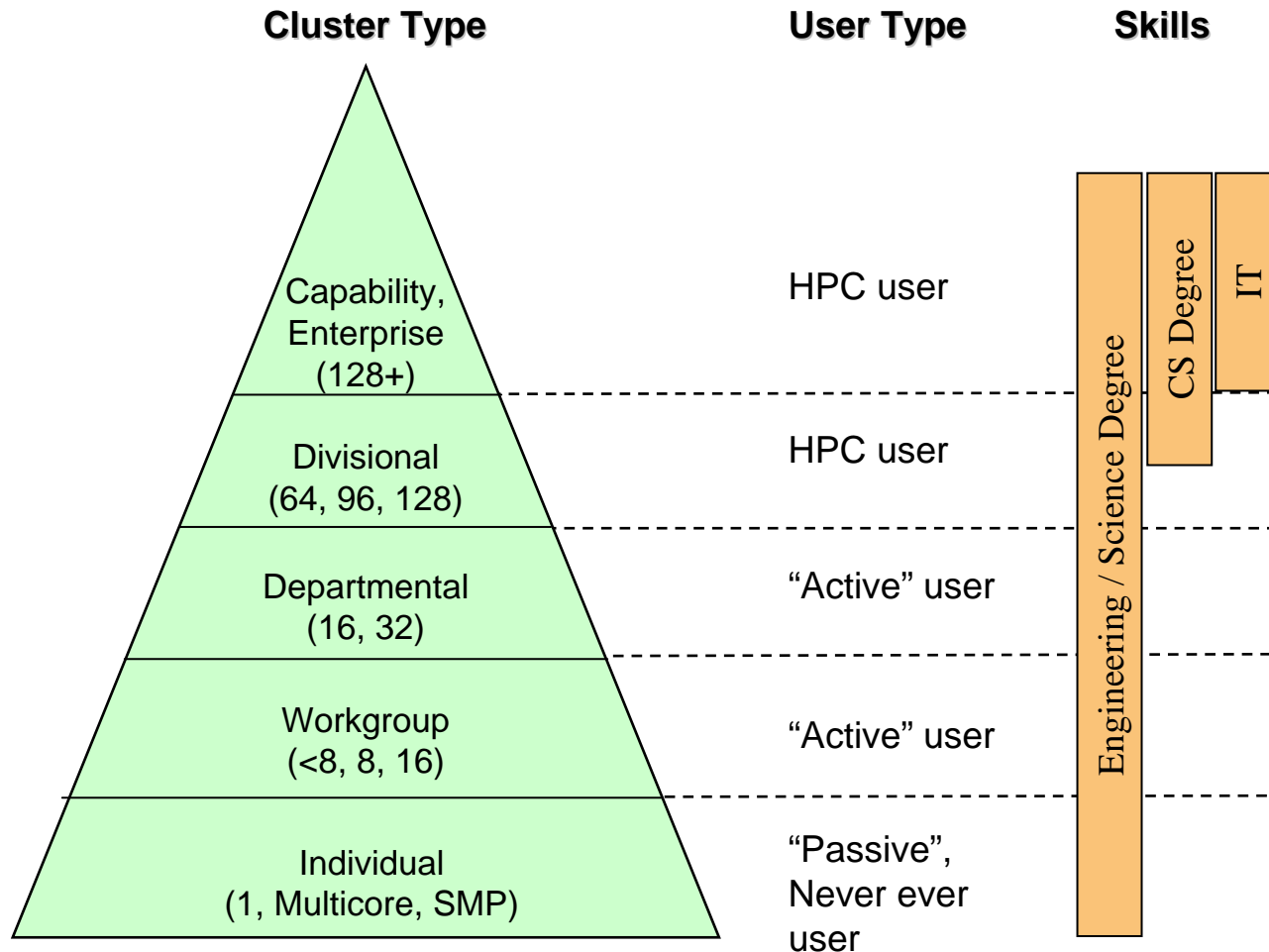
ScaLAPACK



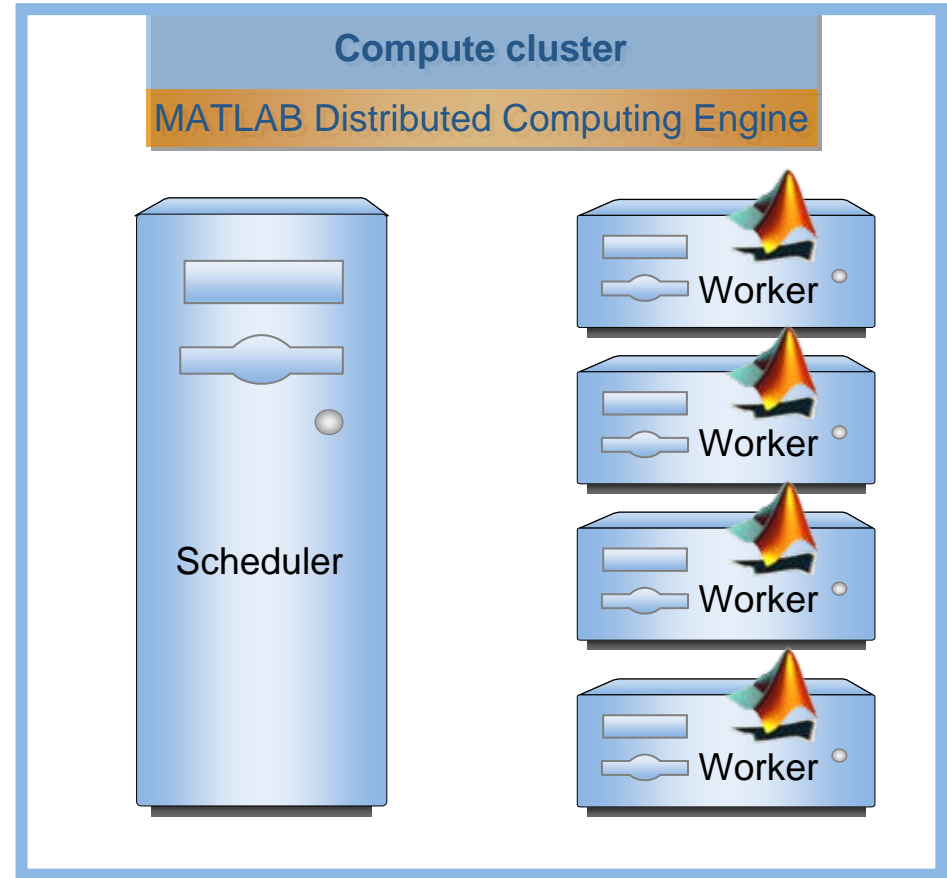
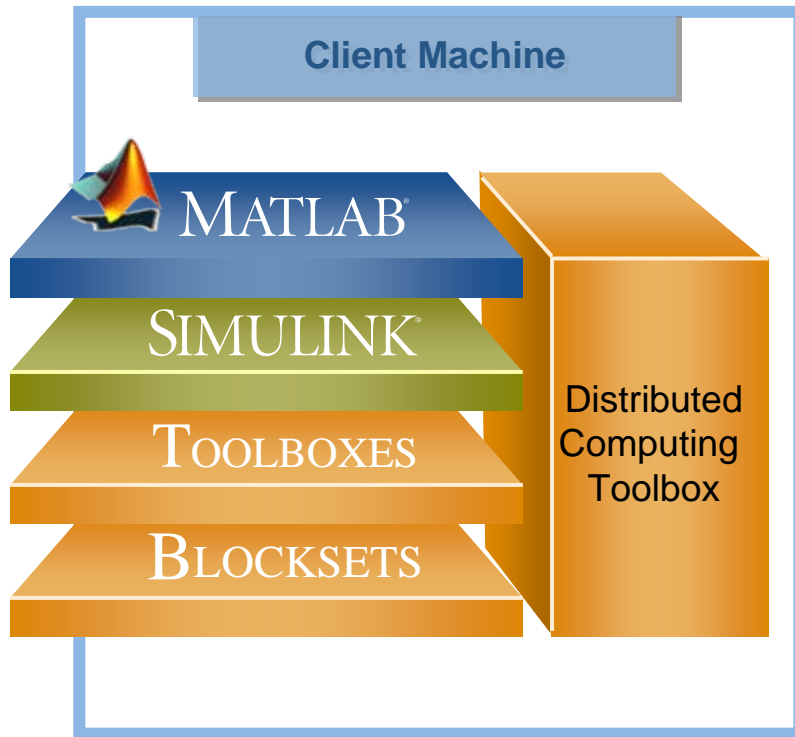
Skills required for distributed computing today



Skills that would be preferred



Distributed computing solution



Agenda

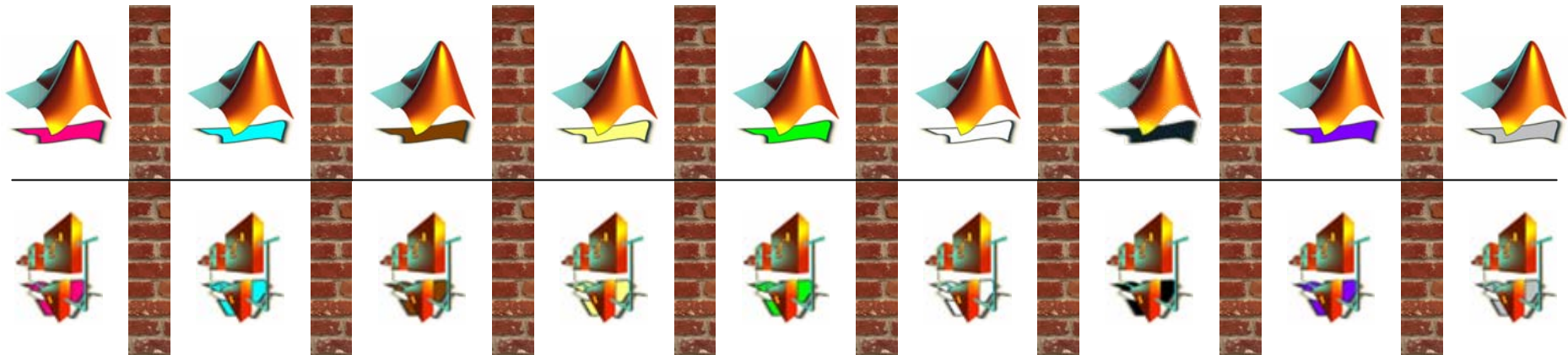
- An important trend impacting the Engineering workflow



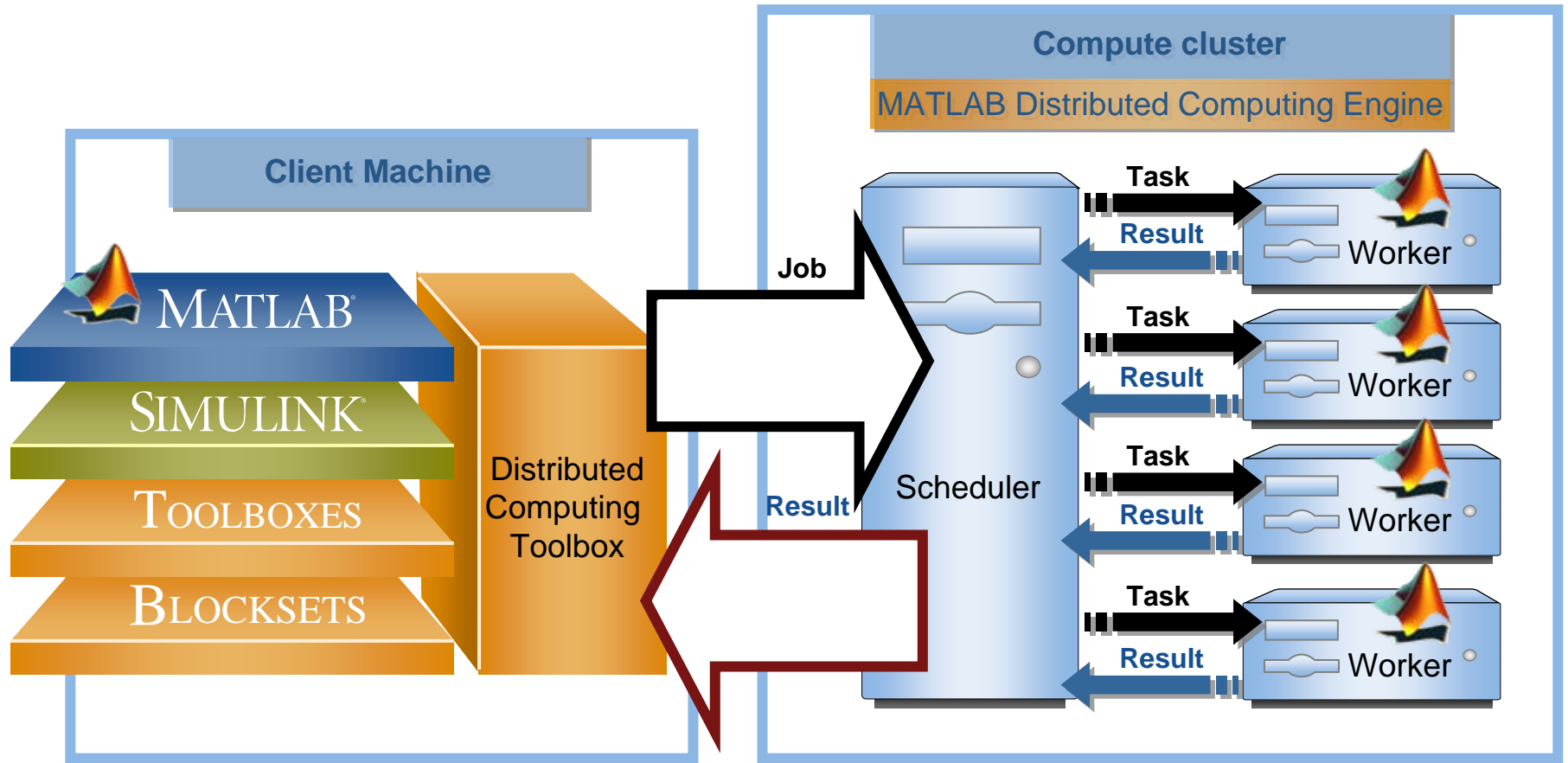
Task parallel applications

- Data parallel applications

Multiple independent problems

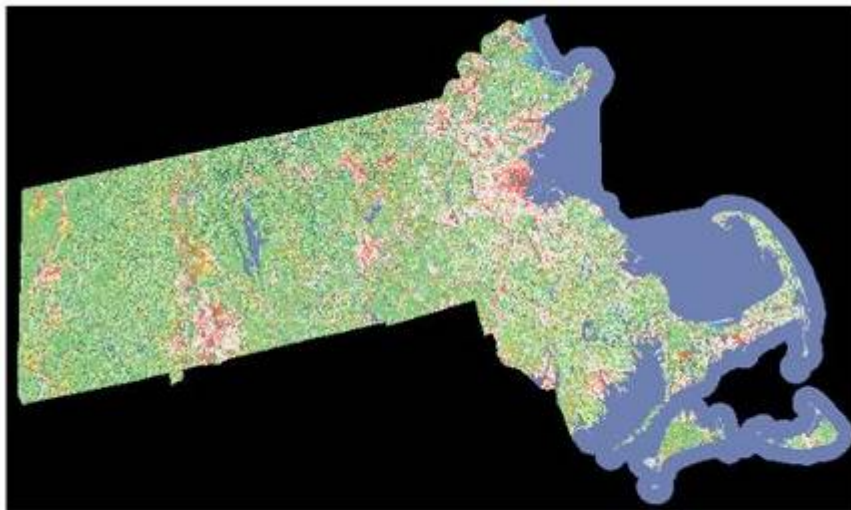


Task parallel applications



Example: Land Classification

- National Land Cover Dataset (NLCD) from U.S. Geological Survey – 30GB
- “Where are wetlands, forests etc concentrated?”
- “How does the distribution compare with other datasets?”



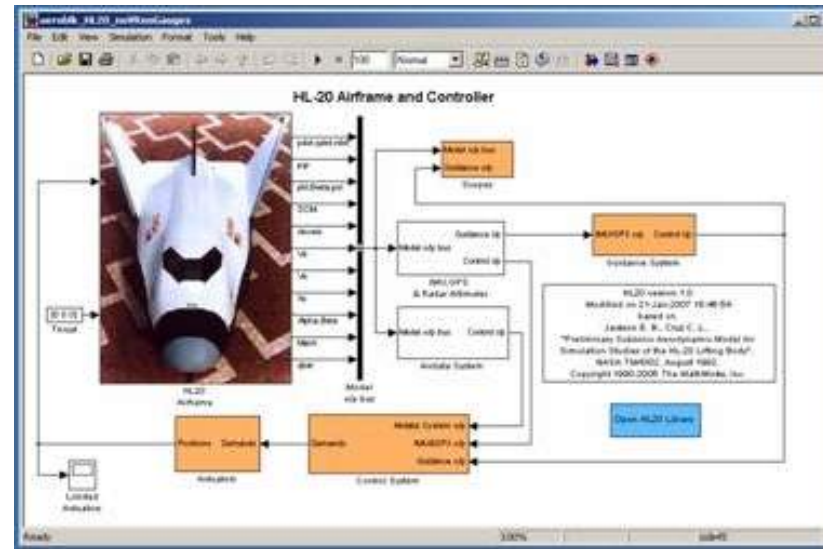
From sequential to distributed: MATLAB

```
1  function results = main(var1, var2)
2
3
4
5
6
7
8
9  nSims = 1000;
10 out = cell(1, nSims);
11
12 for ii = 1:nSims
13     out{ii} = myFunction(ii, var1, var2);
14 end
15
16
17
18
19
20
21 results = postprocessing(out);
22
```


From sequential to distributed: MATLAB

```
1  function results = main(var1, var2)
2
3  jm = findResource('scheduler', 'type', 'jobmanager');
4
5  job = createJob(jm, ...
6      'FileDependencies', 'myFunction.m', ...
7      'PathDependencies', {'\\myPath\myFolder\data'});
8
9  nSims = 1000;
10 out = cell(1, nSims);
11
12 for ii = 1:nSims
13     createTask(job, @myFunction, 1, {ii, var1, var2});
14 end
15
16
17
18
19
20
21 results = postprocessing(out);
22
```

From sequential to distributed: Simulink



1. Divide the Monte Carlo simulations such that each processor executes a full Simulink simulation or RSIM target.
 Eg., one simulation per altitude
2. Create a Task Function that uses MATLAB commands to call the Simulink model you want to execute

Agenda

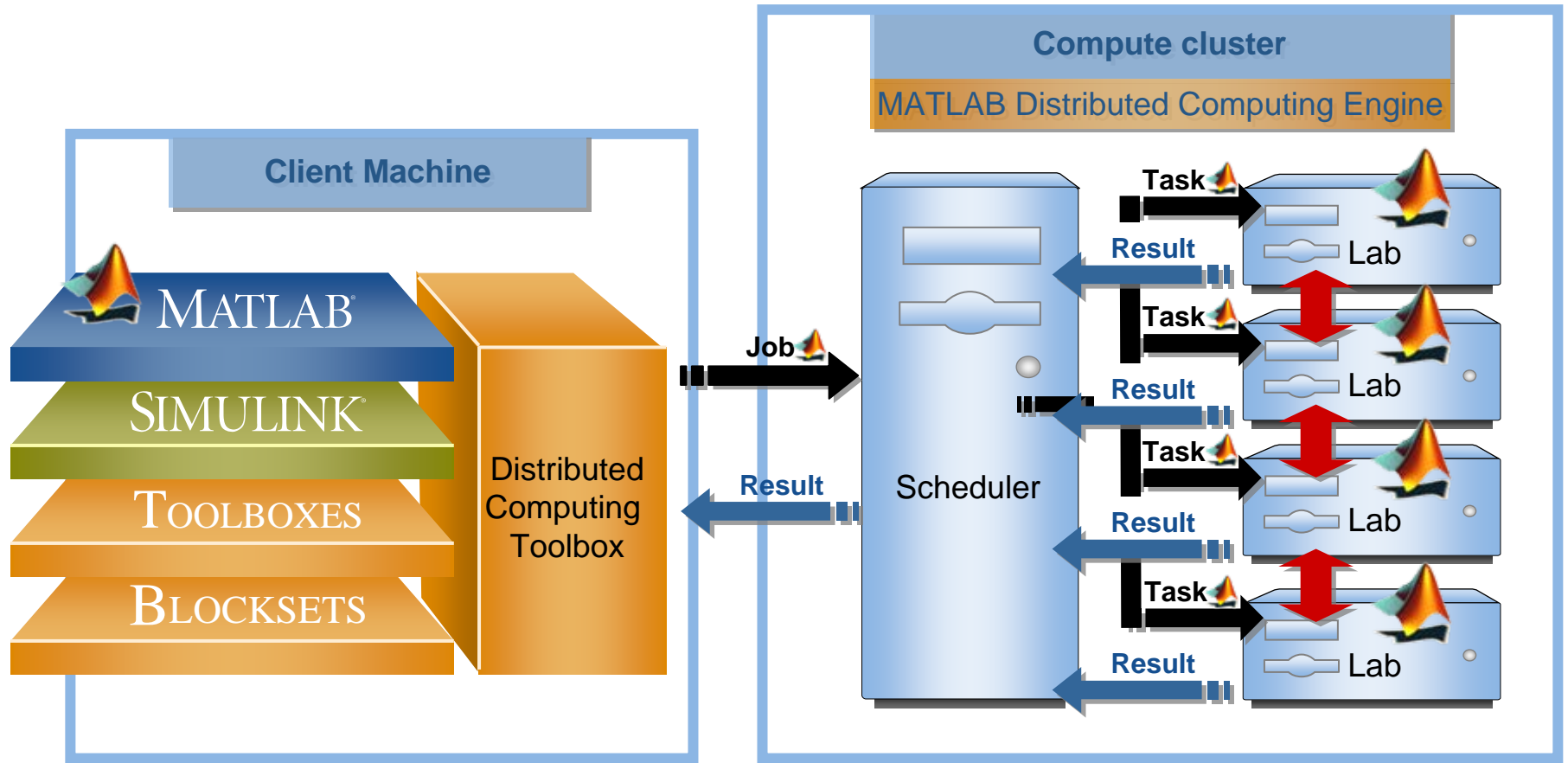
- A little history and context setting
- Task parallel solutions



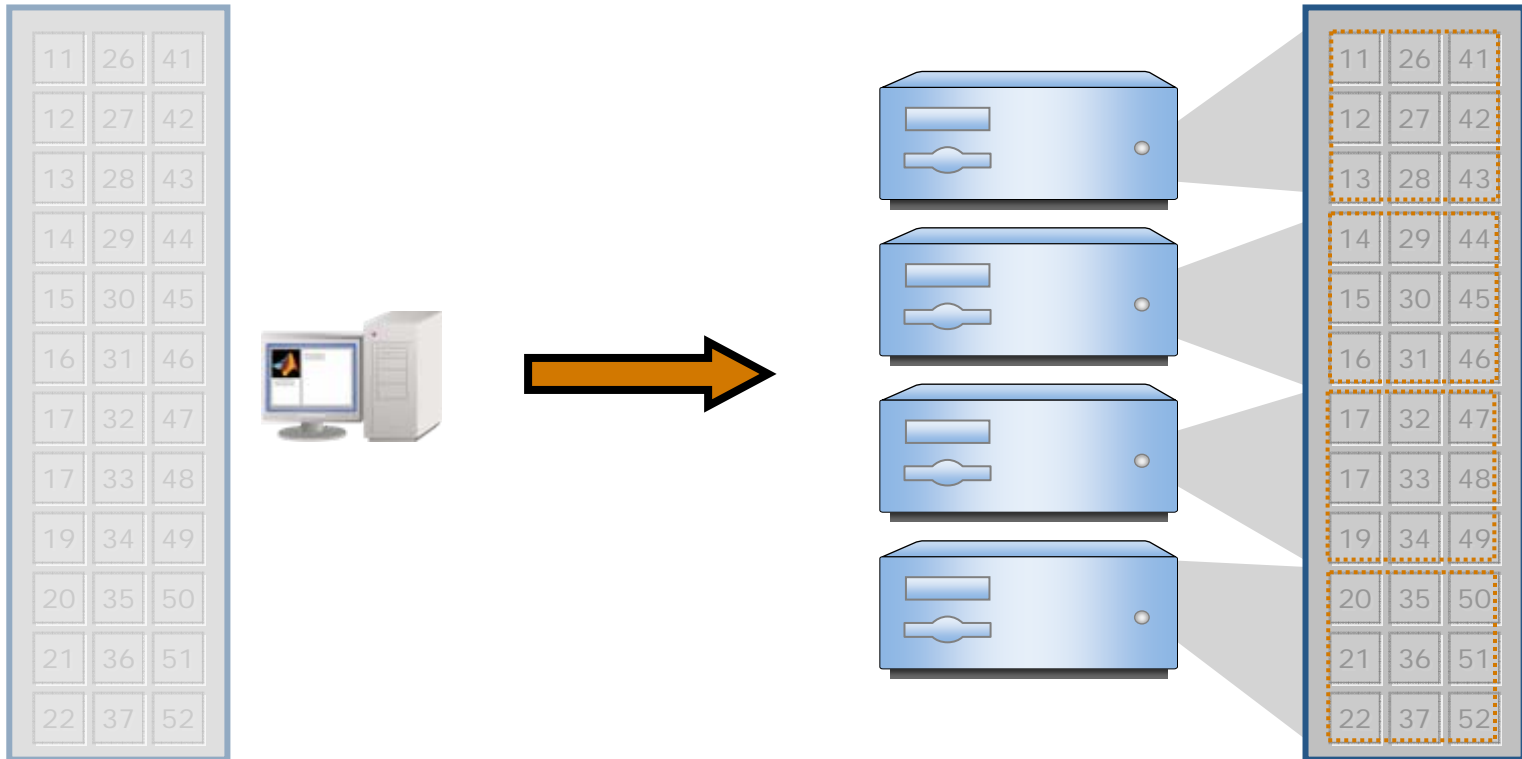
Data parallel solutions

Data parallel applications

(interactive and batch)



Large Memory Requirements



Transposing a Distributed Matrix

```

integer d(1:n1), b(m,n)
do j=1,n1
  j = j + (j-1)*q1
enddo

subroutine table(mv,nv,map,comm2,comm)
  implicit none
  include "mpi.h"
  integer mv, nv, comm, comm2, coord(2), proc1, proc2, ierr, px, qx
  integer map(0:mv-1)

do px=0,mv-1
  do qx=0,nv-1
    coord(1)=px
    coord(2)=qx
    call MPI_Cart_rank(comm2,coord,proc2,ierr)
    coord(1)=qx
    coord(2)=px
    call MPI_Cart_rank(comm,coord,proc1,ierr)
  enddo
enddo

subroutine final_assembly(d,m1,n1,comm,Master,tag,m,n,p,seq)
  implicit none
  include "mpi.h"
  integer b(m,n), d(m1,n1), seq, status(MPI_STATUS_SIZE)
  integer cc(2,0:5)

data cc/1,1,1,2,1,3,2,1,2,2,2,3/

call MPI_Isend(d,m1*n1,MPI_INTEGER,Master,tag,comm,
  & cc(1,ierr)
  & cc(2,ierr)

c**The Master assembles the final (transposed) matrix from local copies
if(Me.eq.Master) then
  do source=0,p-1
    call MPI_Recv(d,m1*n1,MPI_INTEGER,source,tag,comm,
  & status,ierr)
    jx=cc(1,source)
    kx=cc(2,source)
    call assembly(d,m1,n1,b(m,n),jx,kx)
  enddo
write(*,'(S15)') ((b(i,j),j=1,n),i=1,m)
enddo

subroutine merge_matrices(st,m1,n1,dest,tag,comm,d,source)
  implicit none
  include "mpi.h"
  integer m1, n1, dest, tag, comm, source, ierr
  integer st(m1,n1), d(m1,n1)
  integer seq(2), status(MPI_STATUS_SIZE,2)

call MPI_Isend(st, n1*m1, MPI_INTEGER, dest, tag,
  & comm, seq(1), ierr)
call MPI_Recv(d, n1*m1, MPI_INTEGER, source, tag,
  & comm, seq(2), ierr)
call MPI_Finalize(2, seq, status, ierr)

enddo

```

Using FORTRAN and MPI

Using Distributed Arrays

$$P \gg E = D'$$

```

File Edit Text Go Desktop Window Help
1 function [Y, locX] = ptrans(rowsX, colsX)
2 % PARALLEL MATRIX TRANSPOSE
3 % We will simulate a matrix evenly distributed by columns
4 % After transpose each lab must hold a block of columns
5 % of the transposed matrix.
6
7 % Number of cols locally stored on each lab
8 loccols = colsX/numlabs;
9
10 % Number of rows locally stored on each lab
11 locrows_t = rowsX/numlabs;
12
13 % This is the local part of our simulated matrix
14 locX = rand(rowsX, loccols);
15
16 % Transposed local matrix
17 locX_t = transpose(locX);
18
19 % Preallocate Local part of the transposed matrix
20 Y = zeros(rowsX, loccols_t);
21
22 % This part remains on the same lab for all the labs
23 unchanged = locX_t(:, (labindex - 1) * loccols_t + 1 : labindex * loccols_t);
24 Y((labindex - 1) * loccols + 1 : labindex * loccols, :) = unchanged;
25
26 % Now exchange data with other labs
27 % First generate the sequence in which labs exchange data
28 labToFrom = circshift(1 : numlabs, [0 labindex - 1]);
29 labToFrom(labToFrom == labindex) = labToFrom(1);
30 labToFrom(1) = [];
31
32 for labctr = labToFrom
33   senddata = locX_t(:, (labctr - 1) * loccols_t + 1 : labctr * loccols_t);
34   recvdata = labSendReceive(labctr, labctr, senddata);
35   Y((labctr - 1) * loccols + 1 : labctr * loccols, :) = recvdata;
36 end
37
38

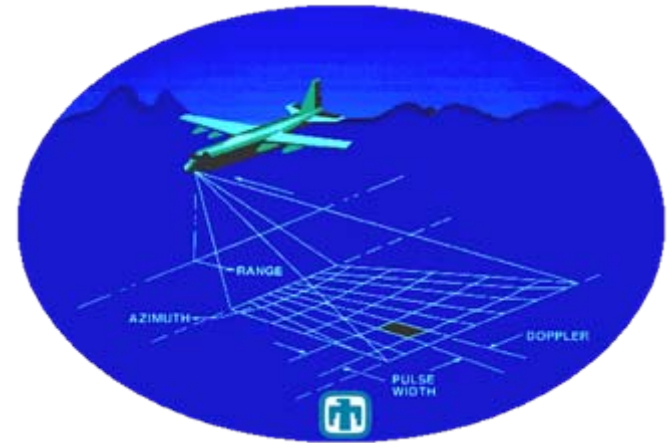
```

Using MATLAB and MPI

Example:

Image Formation Algorithms: Synthetic Aperture Radar (SAR)

- Description
 - SAR is a sophisticated method of post-processing radar data
 - Size and processing requirements demand lots of memory
- Approach
 - Processing SAR images involves interdependent 2-D operations
 - Distribute image across the cluster



From sequential to distributed: MATLAB

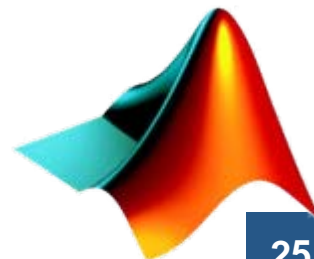
```

1  function ImageOut = cztproc_single
2
3  %Read in SAR image data
4  load sarimage.mat;
5  [I,N] = size(fftImage);
6
7  im_dist = distribute(fftImage,1);
8
9  tic;
10 %*****
11 % azimuth processing
12 %*****
13 nfft = power(2,nextpow2(2*N-1));
14 w = exp(-j*2*pi/N);
15 kk = (-N+1):N-1 .';
16 kk2 = (kk.^2) ./ 2;
17 ww = w .^ (kk2); % <----- Chirp filter is 1./ww
18 nn = (0:(N-1))';
19 aa = ww(N+nn);
20
21 for i = 1:I,
22     % Perform azimuth CZT
23     x = im_dist(i,:);
24     y = (x.' ) .* aa;
25
26     fy = fft( y, nfft );
27     fv = fft( 1 ./ ww(1:(2*N-1)), nfft ); % <----- Chirp filter.
28     fy = fy .* fv;
29     g = ifft( fy );
30
31     g = [g( N:(2*N-1), : ) .* ww( N:(2*N-1) )].'; %#ok<NBR&K>
32
33     im_dist(i,:) = g;
34 end
35
    
```

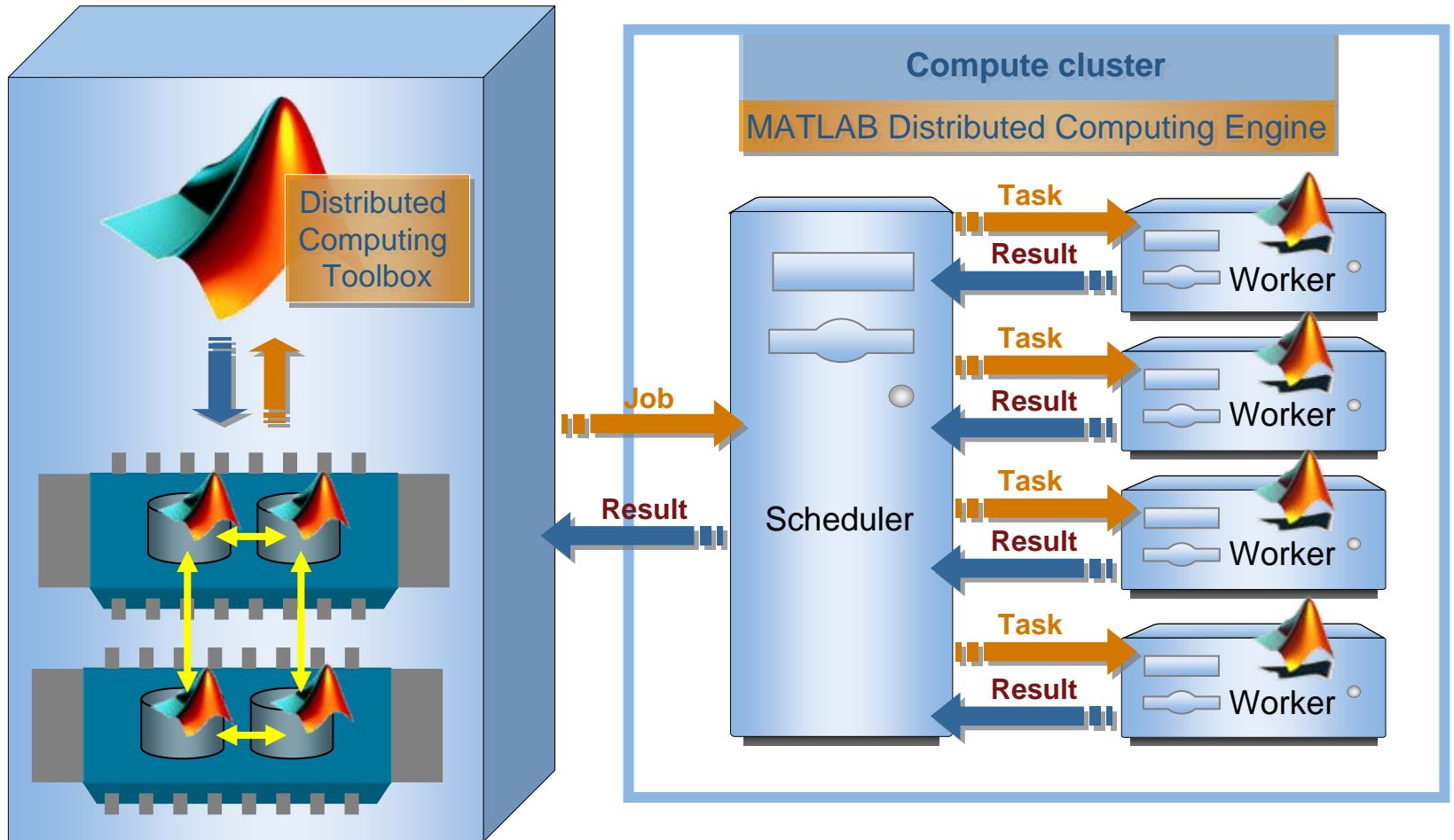

From sequential to distributed: MATLAB

```

1  function ImageOut = cztproc_single
2
3  %Read in SAR image data
4  load sarimage.mat;
5  [I,N] = size(fftImage);
6
7  im_dist = distribute(fftImage,1);
8
9  tic;
10 %*****
11 % azimuth processing
12 %*****
13 nfft = power(2,nextpow2(2*N-1));
14 w = exp(-j*2*pi/N);
15 kk = (-N+1):N-1 .';
16 kk2 = (kk.^2) ./ 2;
17 ww = w .^ (kk2); % <----- Chirp filter is 1./ww
18 nn = (0:(N-1))';
19 aa = ww(N+nn);
20
21 parfor i = 1:I,
22     % Perform azimuth CZT
23     x = im_dist(i,:);
24     y = (x.') .* aa;
25
26     fy = fft(y, nfft);
27     fv = fft(1 ./ ww(1:(2*N-1)), nfft); % <----- Chirp filter.
28     fy = fy .* fv;
29     g = ifft(fy);
30
31     g = [g(N:(2*N-1), : ) .* ww(N:(2*N-1) )].'; %#ok<NBRAK>
32
33     im_dist(i,:) = g;
34 end
35
    
```



Distributed Computing in the Engineering Workflow



Why is this important?

- Technical computing
 - Modeling and analysis involves numerous runs
 - Monte carlo or similar applications very common
 - Complexity of algorithms causes longer execution times
 - Data sets are increasing in size

- Model-based design
 - Simulation is done prior to real-world implementation
 - Many scenarios tested
 - Optimal solutions can be found earlier
 - Simulations are growing in complexity and size
 - Simulation time increases
 - Rsim and other targets are only part of the solution

Summary

- Hardware trends are impacting everybody
- Understanding and creating distributed applications will be an important skill for anybody in the fields of Computer Science or Technical Computing
- MathWorks provides a market-leading solution for distributed applications
- Demo available in exhibit hall