Towards High-Level Programming Support for Scientific Computing on Clusters

Hans P. Zima

Institute for Software Science, University of Vienna, Vienna, Austria

and

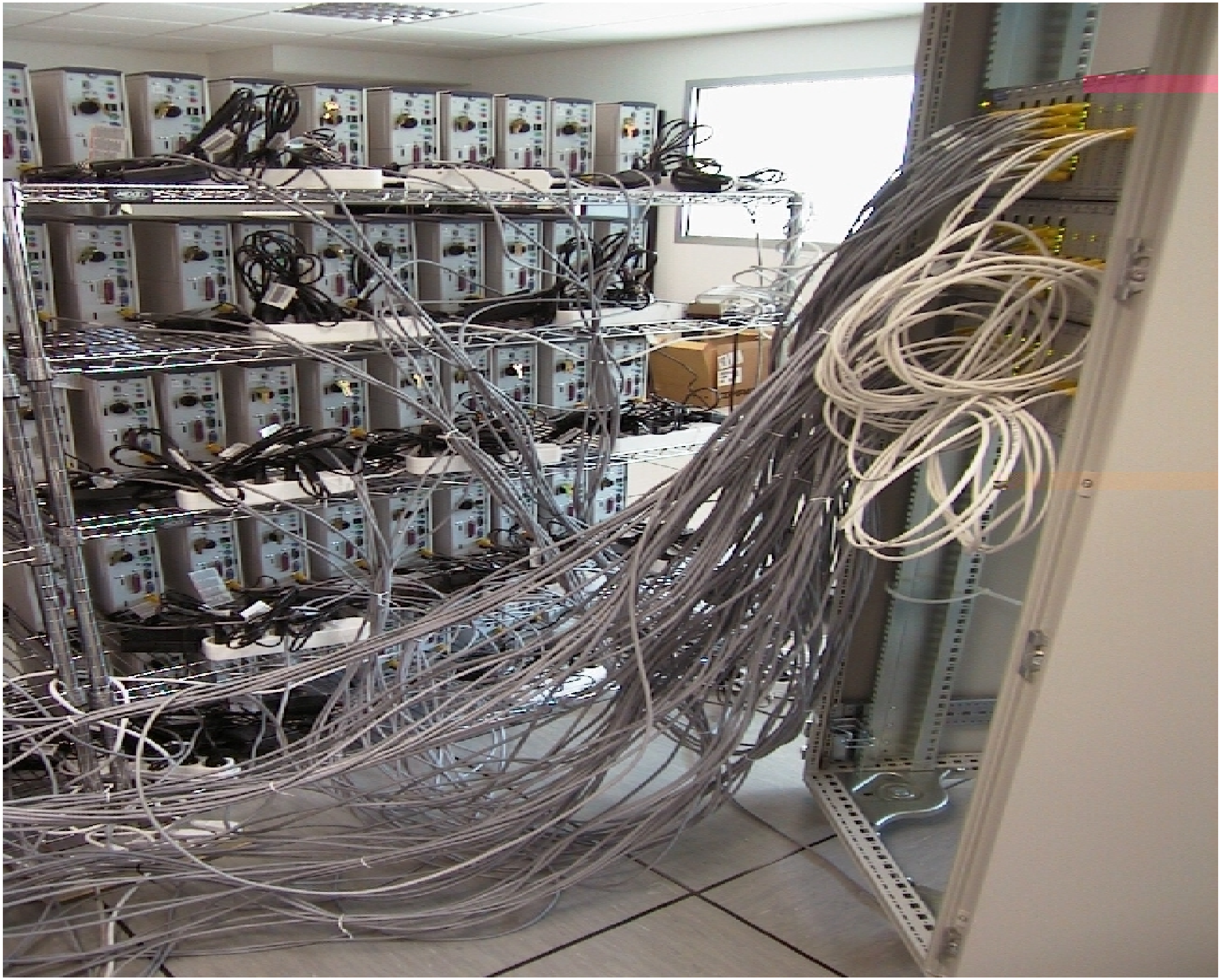
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, USA

zima@cacr.caltech.edu

IEEE International Conference on Cluster Computing -- Cluster2001
Newport Beach, California, USA

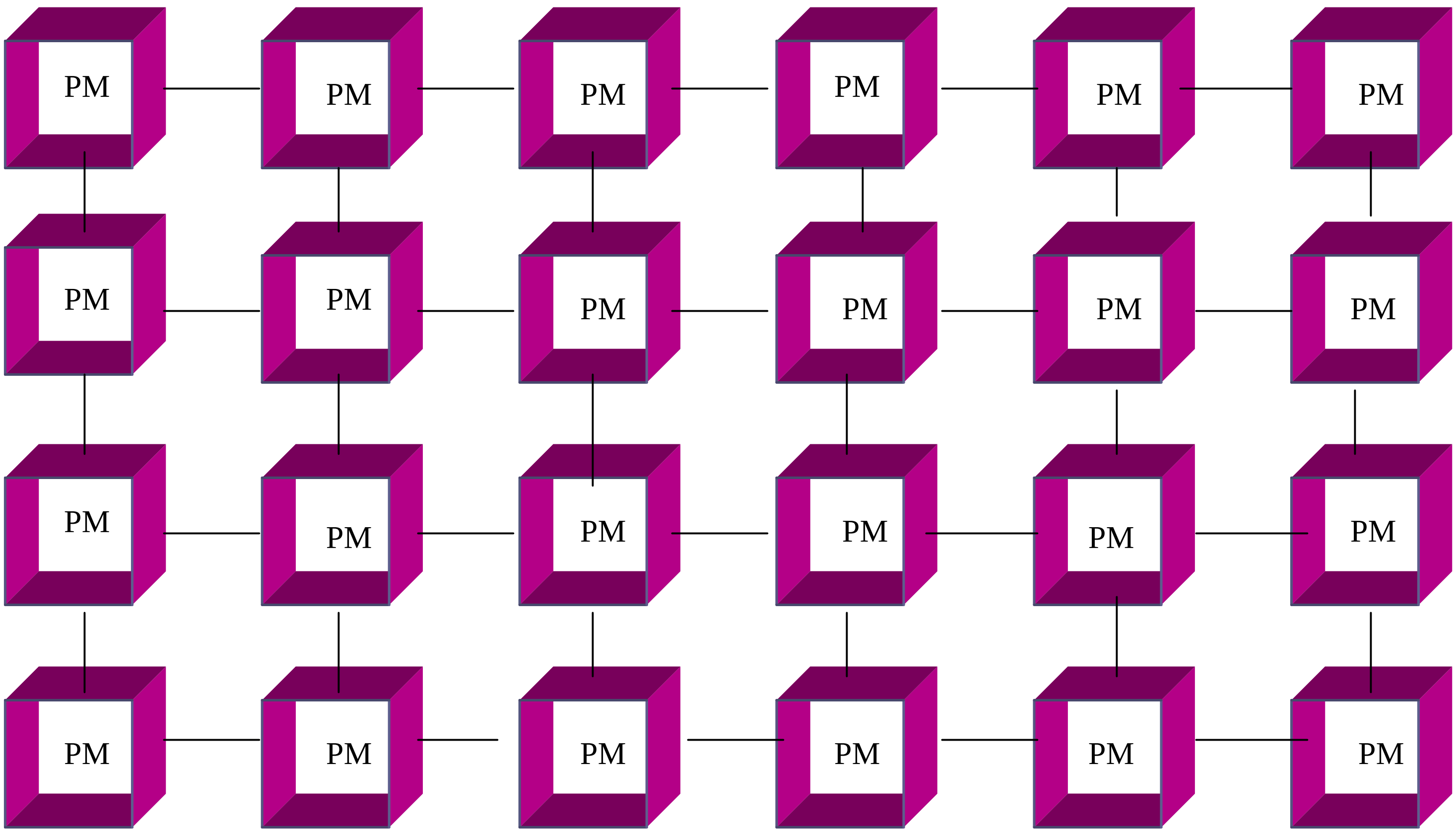
OUTLINE

- 1 Introduction
- 2 Distribution and Alignment Control
- 3 Irregular and Dynamic Problems
- 4 Generalizing the Concept of Distribution
- 5 Future Aspects
- 6 Conclusion



Source: i-Cluster, University of Grenoble, France

Abstract Cluster Architecture



Programming Models

- **What is the right model for parallel programming of clusters?** ***-- A viable compromise must be found between the conflicting goals of***
  - **expressivity**
  - **portability**
  - **performance**
- ***For performance-oriented programming, the*** **message-passing model** ***has been traditionally adopted.***

This Talk ...

- *discusses high-level programming support for data-intensive scientific parallel programming of clusters that can provide sufficient* **performance,** *including irregular and dynamic problems*
- *focuses on* **high-level specification** *of*
  - **distribution of data and work**
  - **affinity relationships**
  - **communication**

The Problem: **Running a single large-scale scientific application in parallel on a cluster. Can the application programmer be provided with higher-level support than message passing without inacceptable loss of performance?**

Initial Assumptions

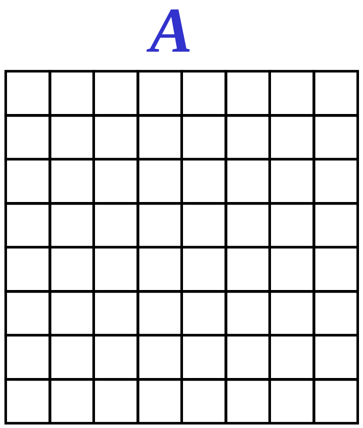
- ***Homogeneous cluster***
- ***Single-processor nodes***
- ***Each node operates in a separate address space***
- ***Abstract network topology***

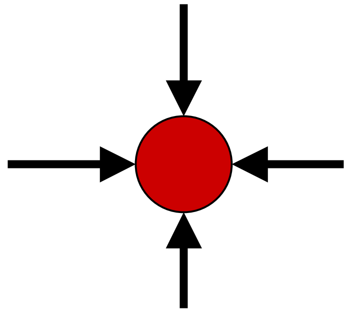
MPI vs. HPF

An Almost Trivial Example (Jacobi)

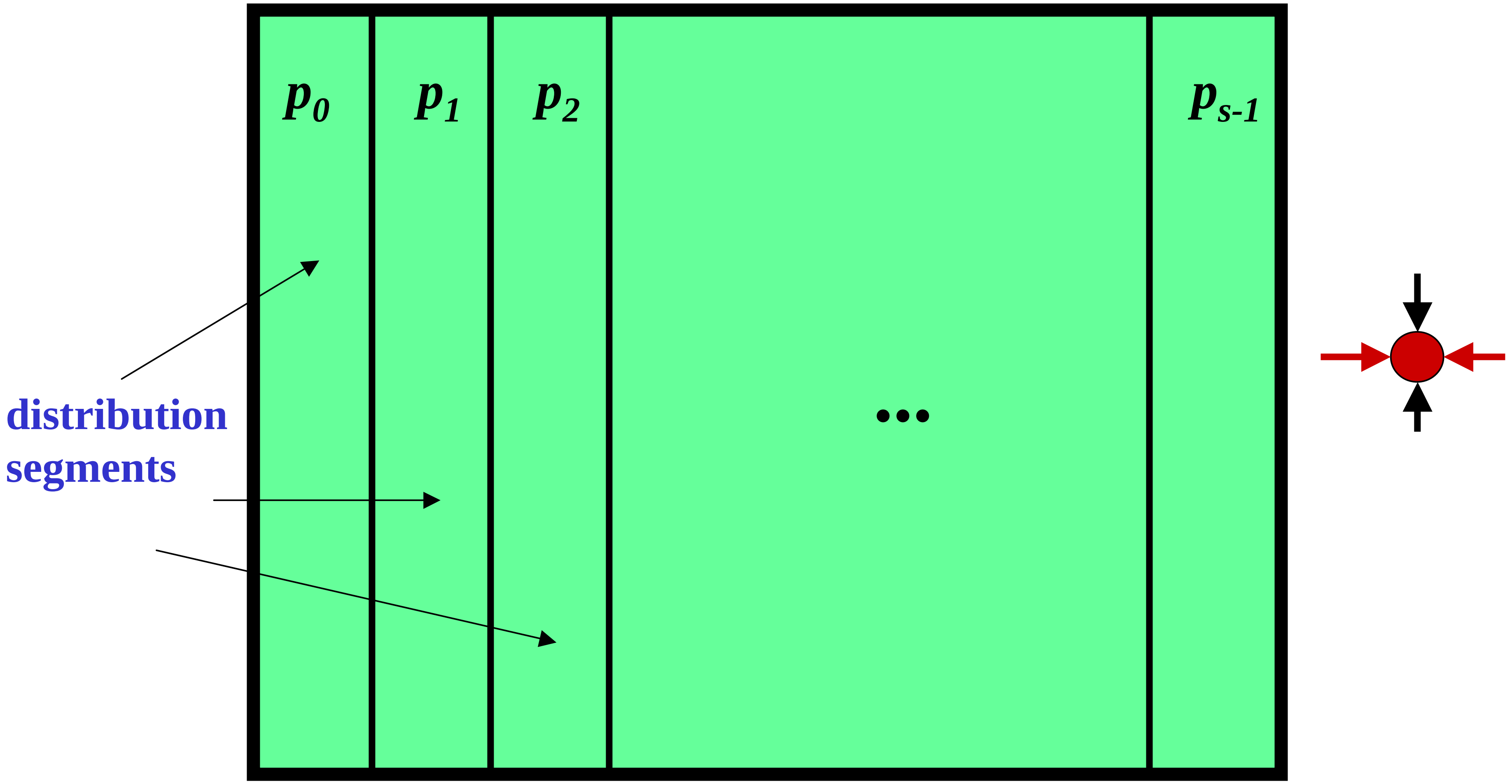
Sequential Jacobi Kernel

```
do while (.not. converged)
   do  J=1,N
      do  I=1,N
         B(I,J) = 0.25 * (A(I-1,J)+A(I+1,J)+
                          A(I,J-1)+A(I,J+1))
      end do
    end do
   A(1:N,1:N) = B(1:N,1:N)
   ...
end do
```

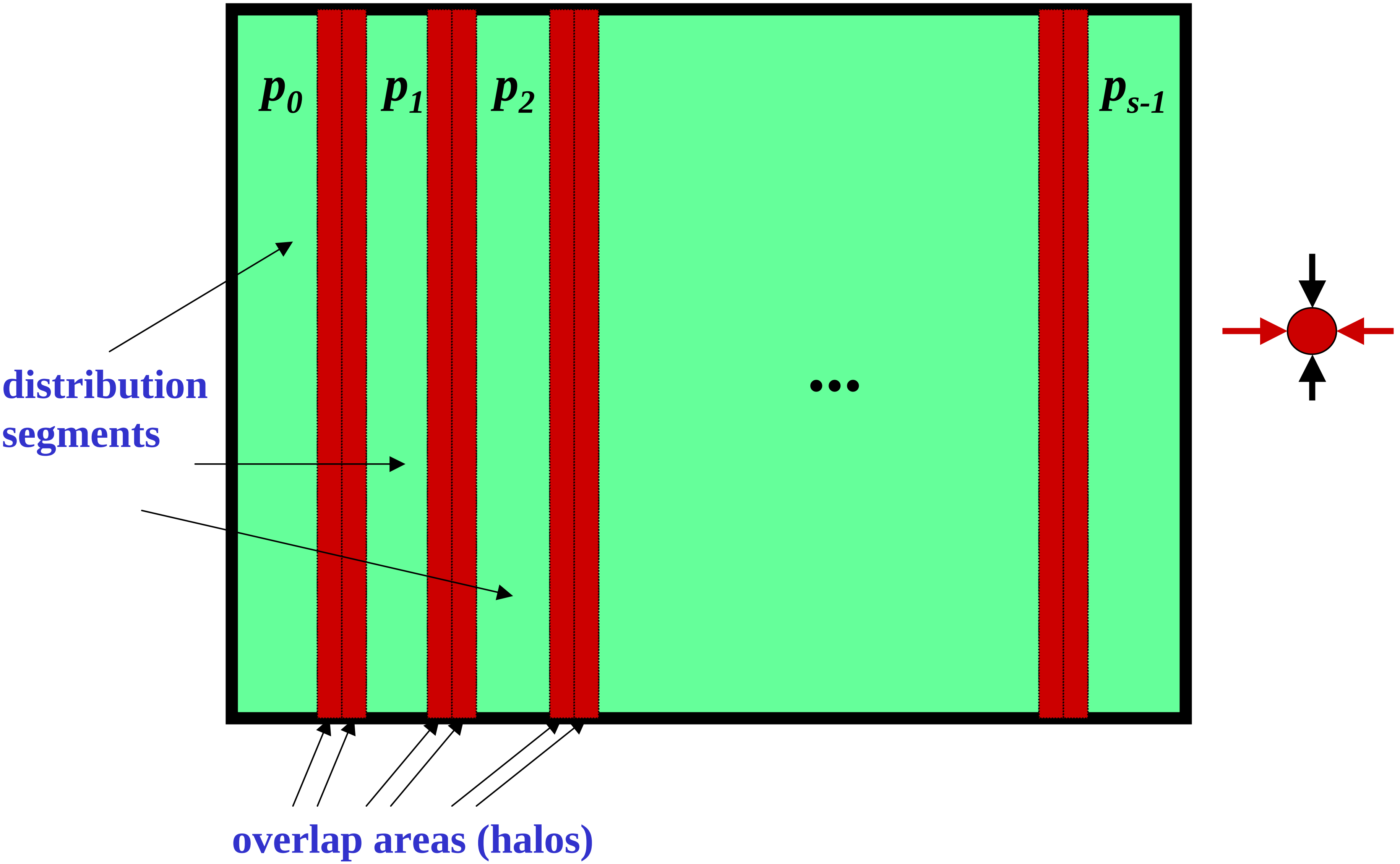




Column-block distribution of a 2D-matrix



Column-block distribution of a 2D-matrix

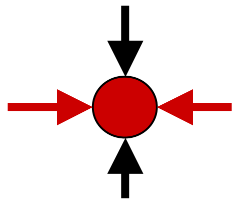


Parallel Jacobi with MPI

initialize MPI

```
do while (.not. converged)
   do  J=1,M
      do  I=1,N
          B(I,J) = 0.25 * (A(I-1,J)+A(I+1,J)+
                          A(I,J-1)+A(I,J+1))
      end do
   end do
   A(1:N,1:N) = B(1:N,1:N)
```

local computation



```
if (MOD(myrank,2) .eq. 1) then call MPI_SEND(B(1,1),N,…,myrank-1,..)
                              call MPI_RCV(A(1,0),N,…,myrank-1,..)
                              if (myrank .lt. s-1 then
                                call MPI_SEND(B(1,M),N,…,myrank+1,..)
                                call MPI_RCV(A(1,M+1),N,…,myrank+1,..)
                              endif
                          else  …
                       …
```

Parallel Jacobi with HPF

processors *P(NUMBER_OF_PROCESSORS)*
distribute(*,BLOCK) onto *P :: A, B*

```
do while (.not. converged)
    do  J=1,N
        do  I=1,N
            B(I,J) = 0.25 * (A(I-1,J)+A(I+1,J)+
                            A(I,J-1)+A(I,J+1))
        end do
    end do
    A(1:N,1:N) = B(1:N,1:N)
```

global computation

Communication is automatically generated by the compiler

Observations

- *The HPF code is far simpler than the MPI code*
- *The compilers can generate from that HPF code an*

The Problem: Irregular, Dynamic, and Adaptive Aplications

- **Examples:** ***applications working with semi-structured grid collections, unstructured grid codes, spectral transform codes***

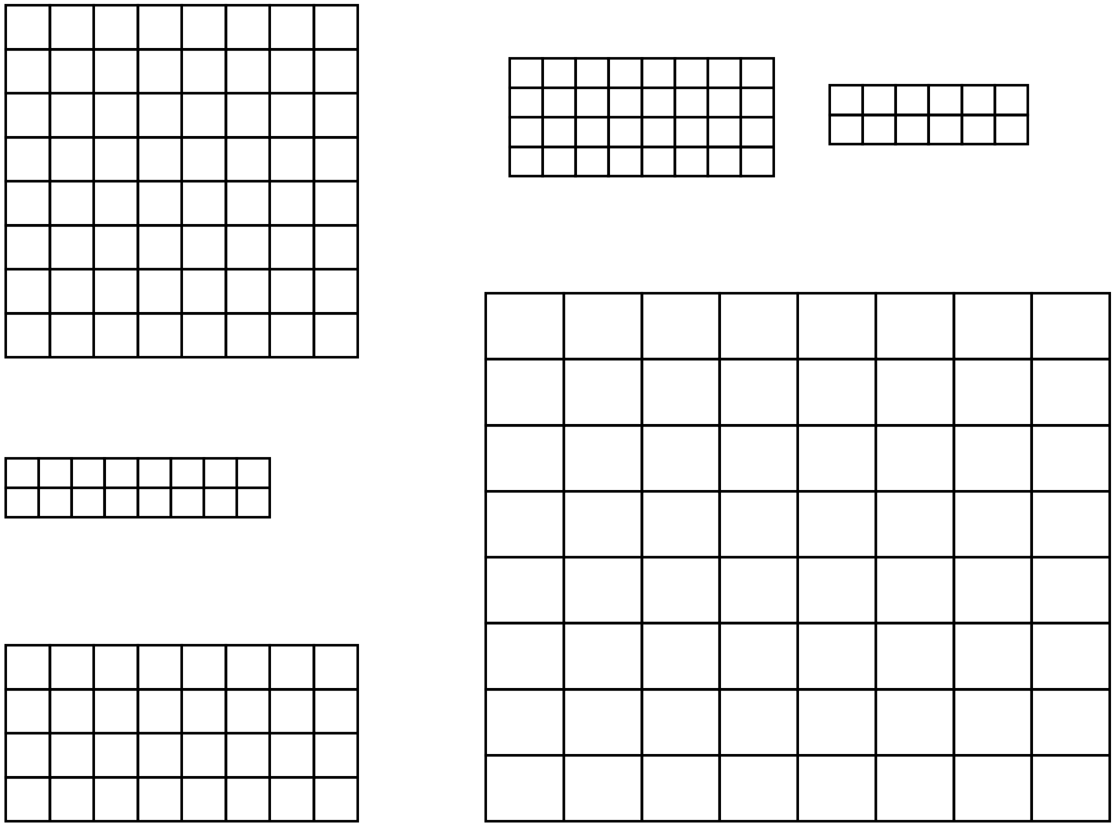
- **Typical features:** ***Irregular and/or dynamically varying data structures, data access and dependence patterns, data distribution, work patterns and work distributions***

Semi-structured Grid Collections

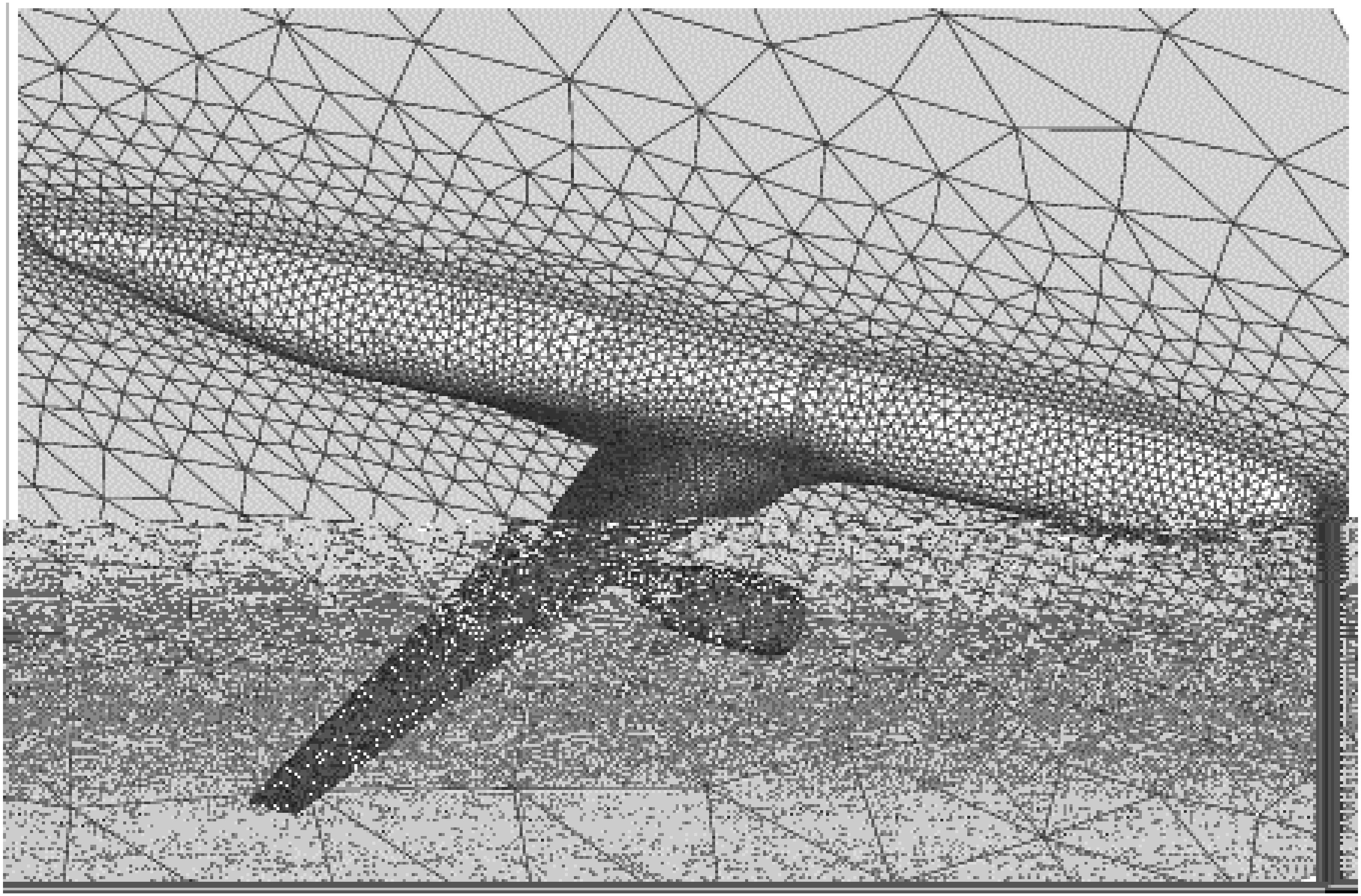
- ***Irregularly structured sets of structured grids that can be processed in parallel:***
  - **multiblock**
  - **parallel multigrid**
  - **structured AMR (SAMR)**

- ***In order to exploit two-level parallelism and achieve load balancing, it is necessary to***
  - **distribute grids to subsets of processors**
  - **allow irregular distributions**

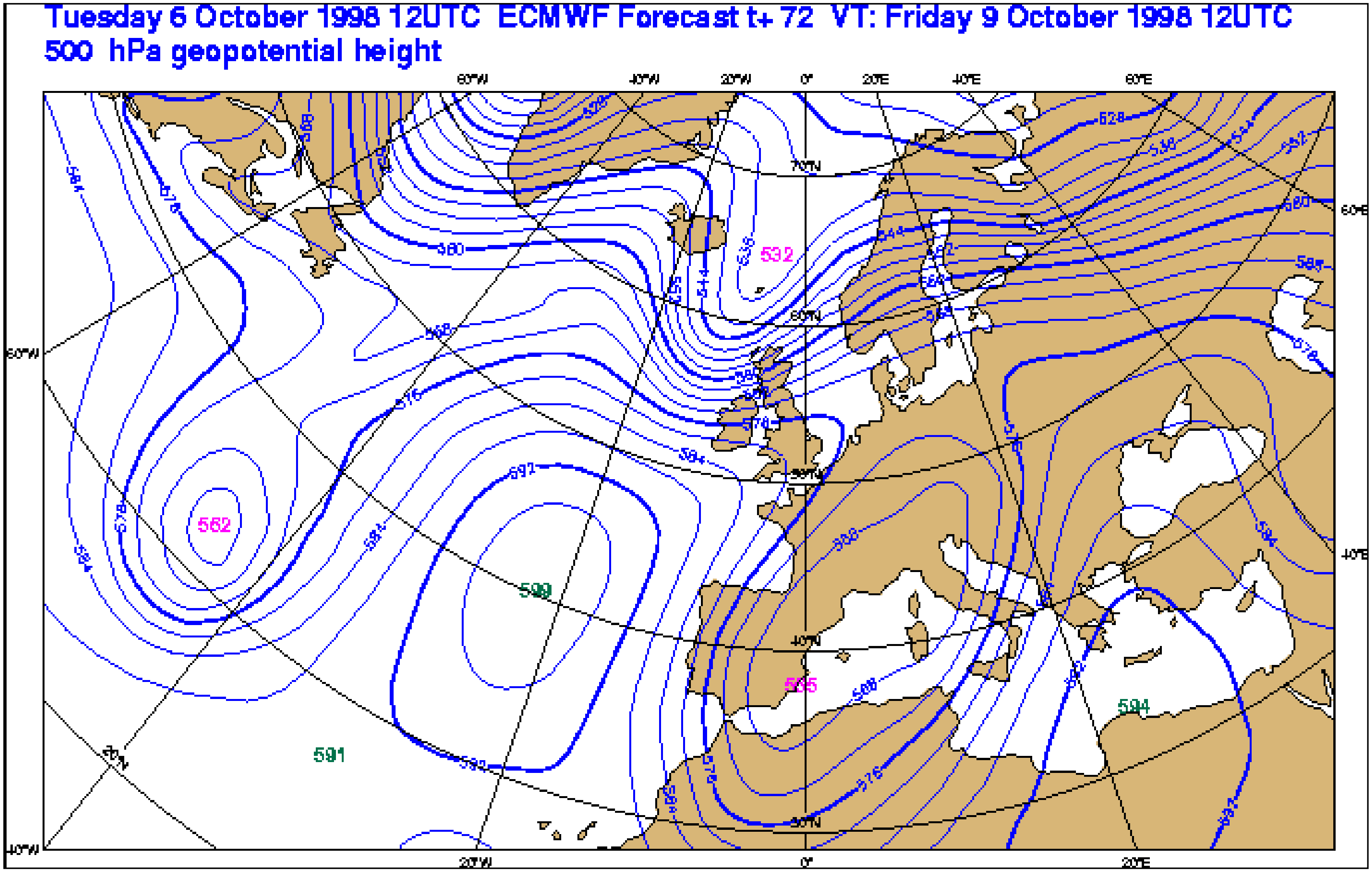
Example: A Multiblock Grid Collection



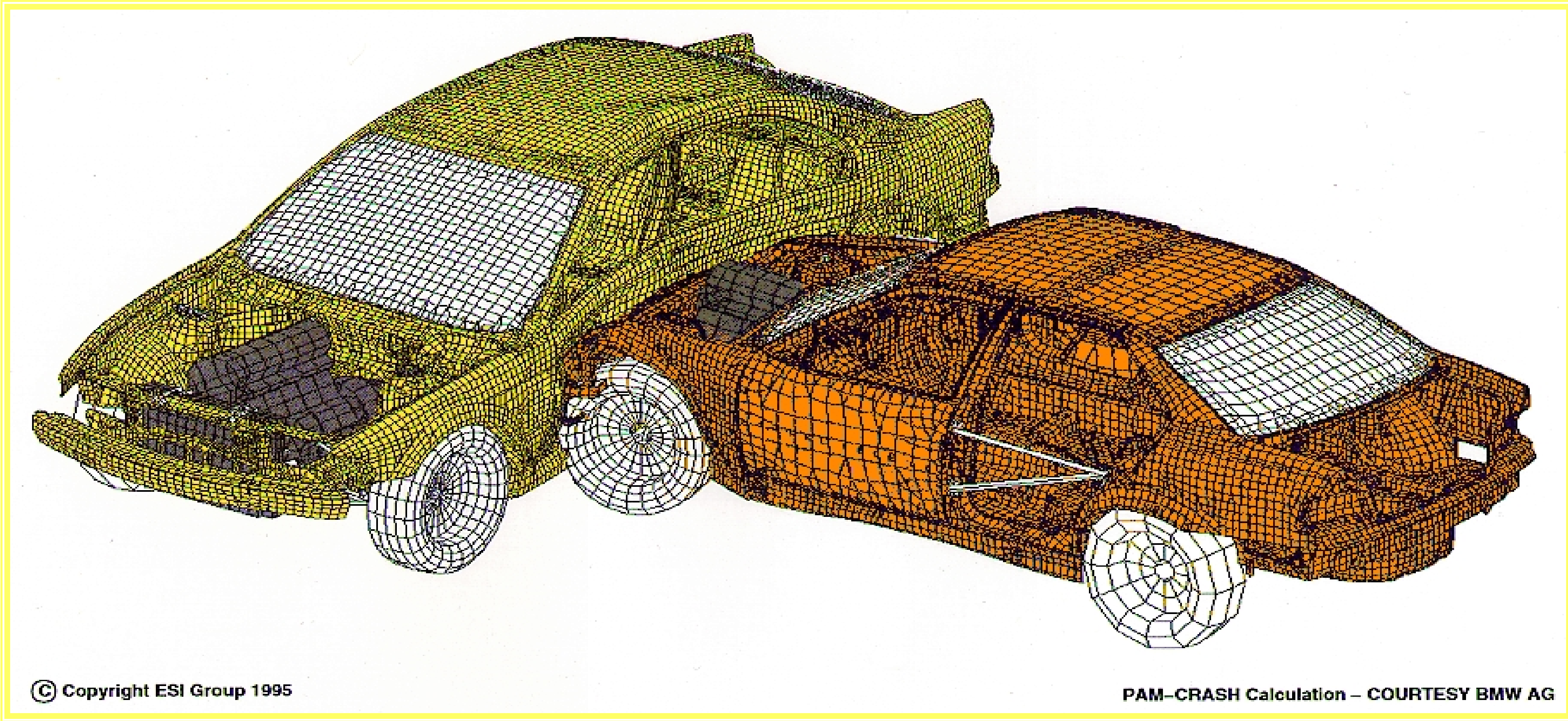
An Unstructured Grid



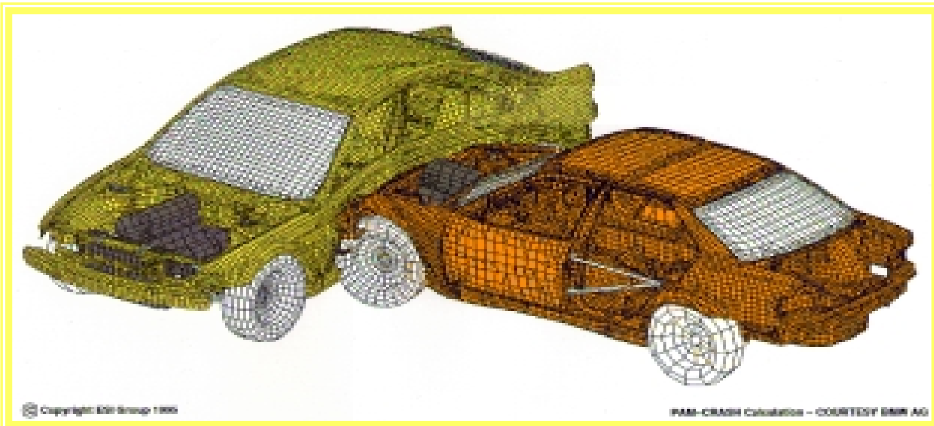
Source: Dimitri Mavriplis, ICASE, NASA Langley Research Center



Hans P. Zima
Cluster2001, Newport Beach,
California


© Copyright ESI Group 1995
PAM–CRASH Calculation – COURTESY BMW AG

Finite Element Code for Crash Simulation



Access to unstructured meshes requires at least 2 levels of indirection.

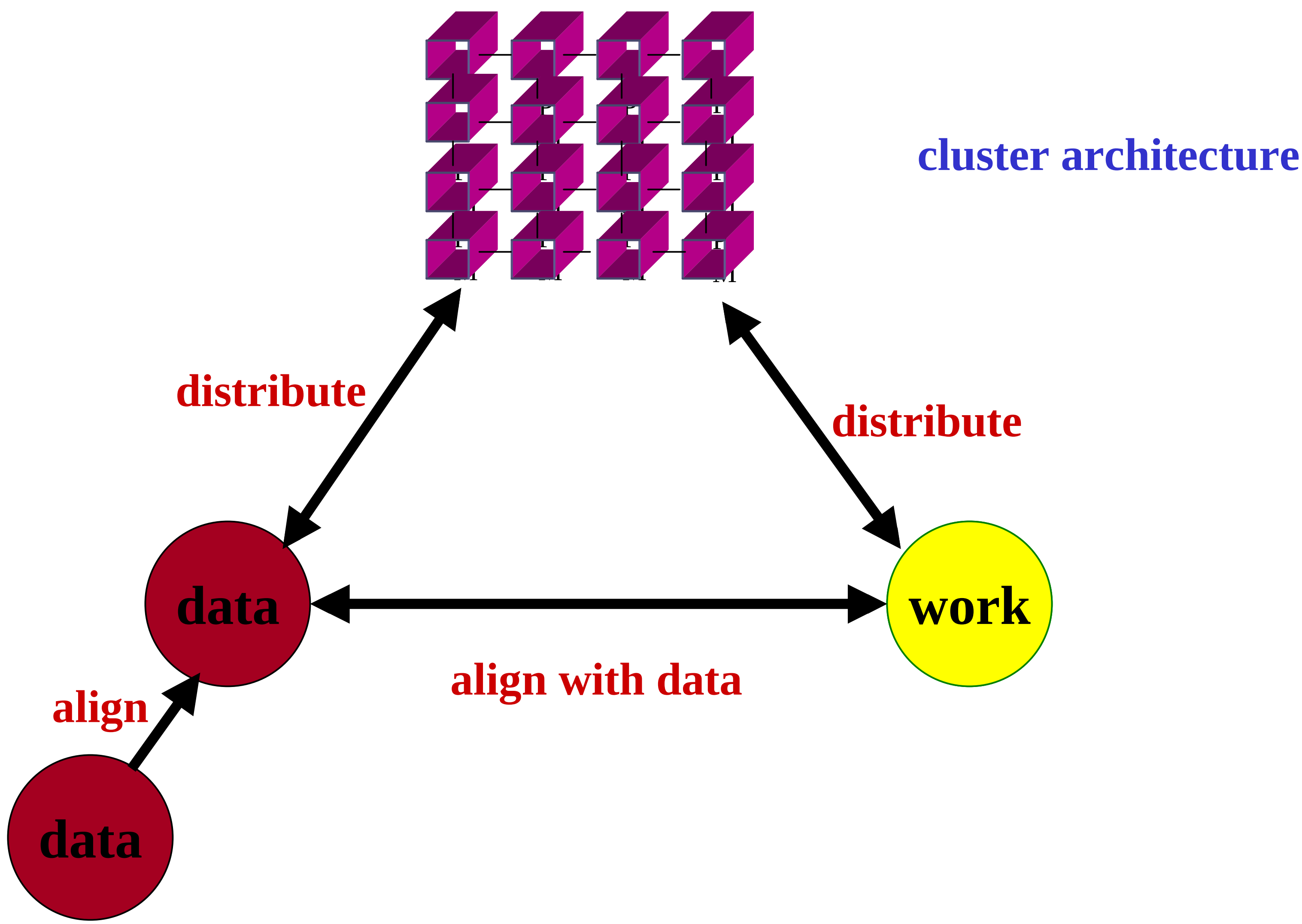
access patterns cannot be analyzed at compile-time

```
REAL :: X(3,N_NODES), F(6,N_NODES), ...
INTEGER :: IX(4,N_ELEMS) !mesh connectivity
...
do i = 1, N_ELEMS
   do i = 1, 4
      F(:,IX(K,I)) = ...+F(:,IX(K,I))+ ...
   end do
end do
```

Why Simple Distribution Strategies are not Adequate for Irregular Problems

- *Regular data distributions may not reflect* **locality in physical space**
- *Regular data distributions may not support* **load balancing** *for irregularly distributed objects*
- *the* **owner-computes paradigm** *does not allow expression of* **affinity** *between data and work distribution*
- *static distribution strategies cannot reflect dynamic* **changes of data and computation structures**

Distribution and Alignment Control



Requirements for a More General Distribution Model

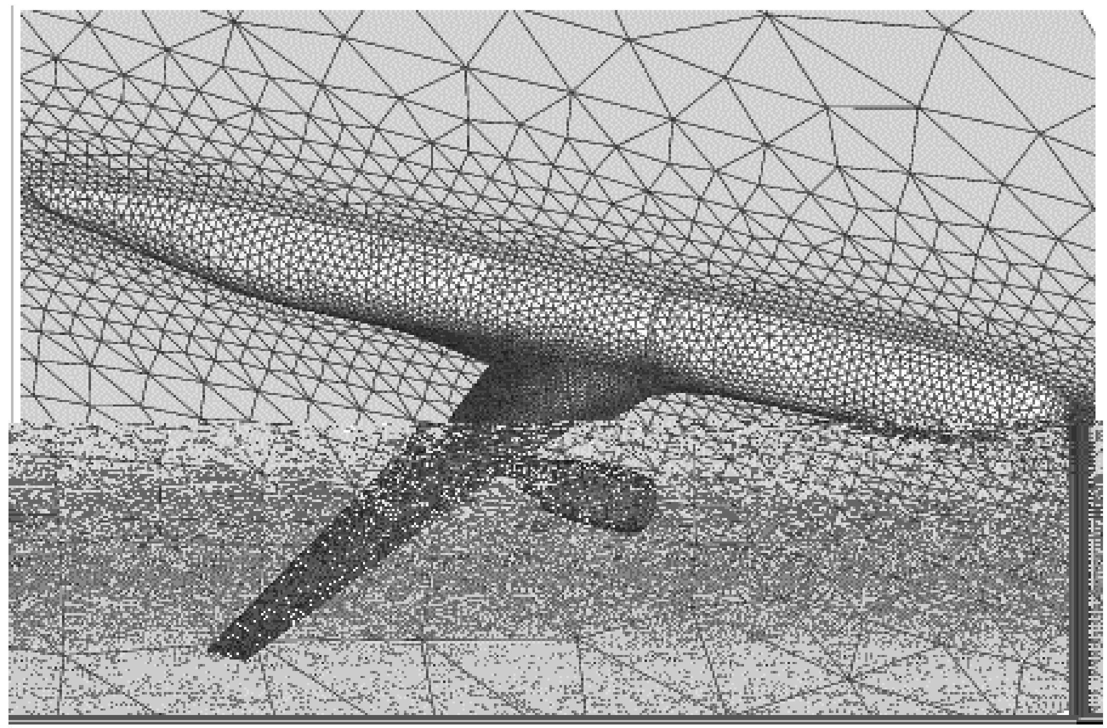
- ***Generalize mappings*** index space ---> processors
- ***Allow distributions to subsets of processors***
- ***Generalize affinity specifications***
  - **data alignment**
  - **work / data alignment**
- ***Allow distributions and alignments to change dynamically***
- ***Improved (high-level) control of communication***

Data Distribution for Unstructured Grids

Problem: ***index space locality does not reflect locality in 3D space***

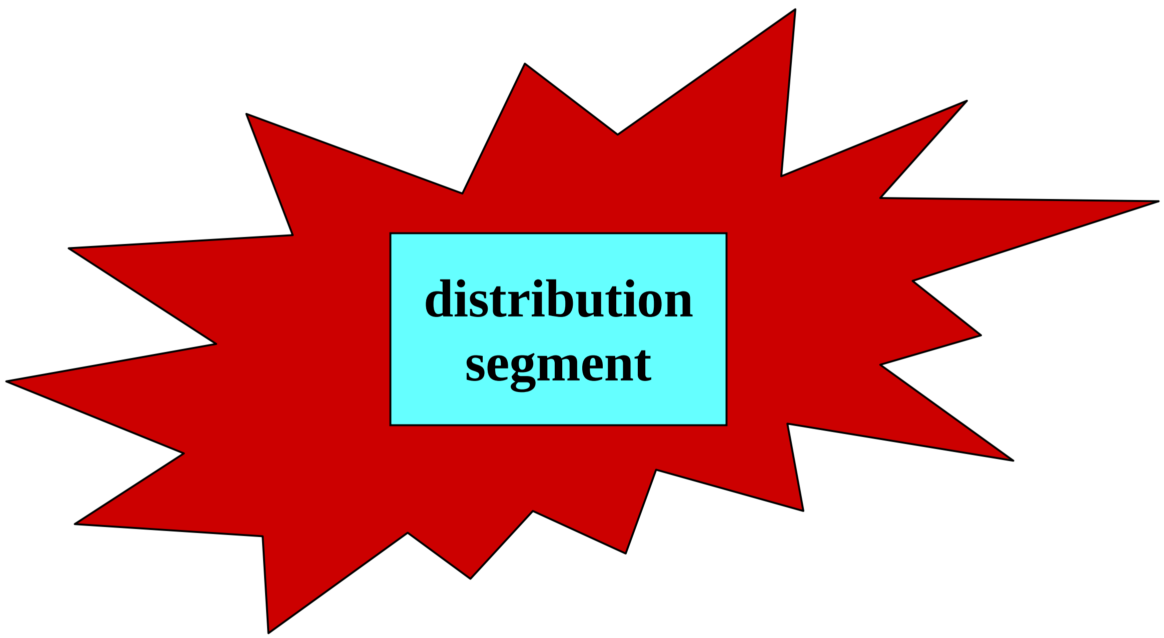
Solution 1: ***indirect distribution***

Solution 2: ***general block distribution combined with reordering of elements***

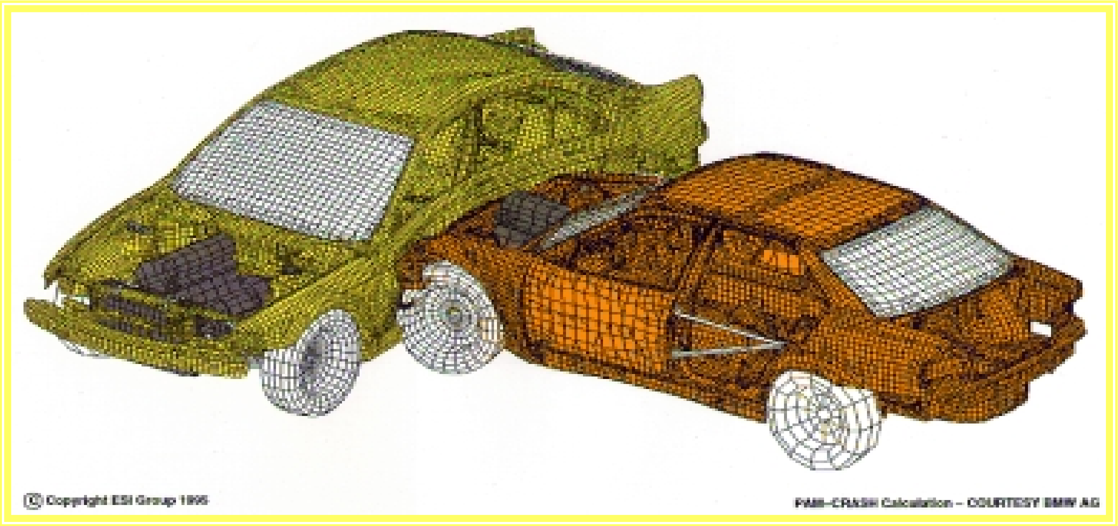


Two Methods for Communication Control

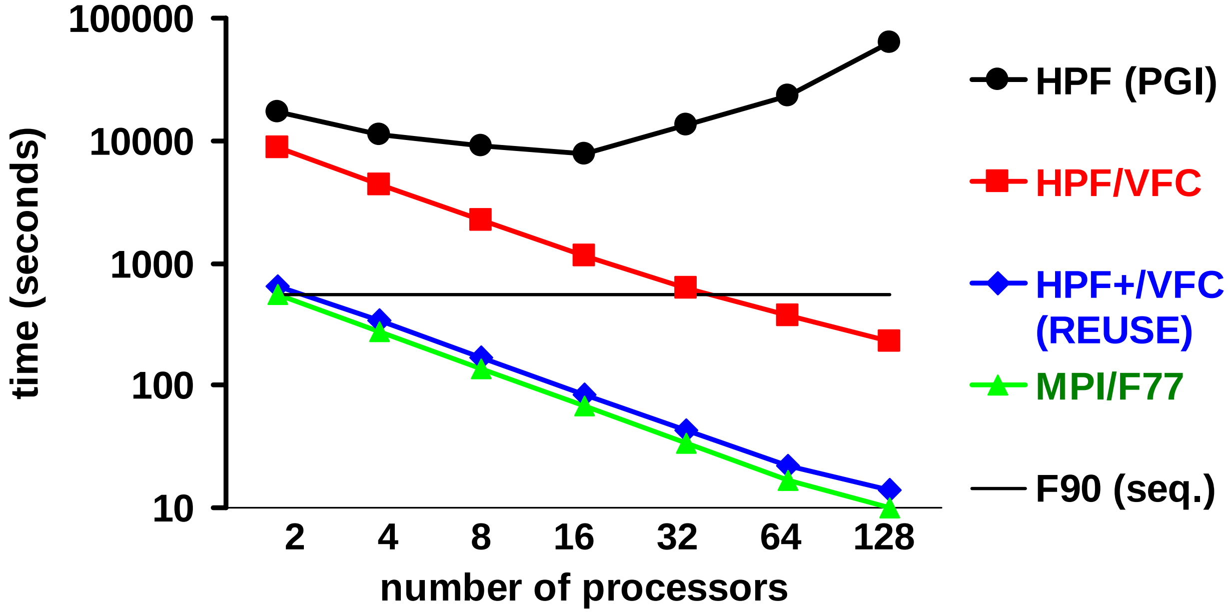
- **Schedule Reuse Control:** ***make communication schedules explicitly accessible objects and control their evaluation and reuse***
- **Halo management:** ***allow explicit control of irregular communication halos (ghost regions)***



The Effect of Schedule Reuse and Halo Management



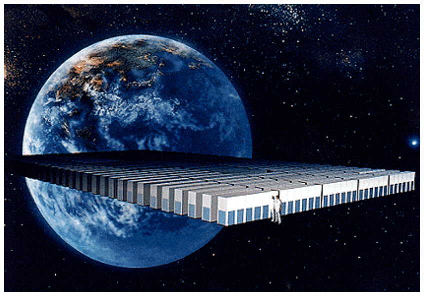
FEM crash simulation kernel



Language Extensions

(HPF/Earth Simulator)

- ☐ Schedule Reuse
- ☐ Halo Control
- ☐ Purest proc's



Example: Sparse Matrix Vector Product

- Generate the matrix in **distributed sparse format**
  - sparse representation (e.g., compressed row storage)
  - data distribution across memories
- **For each distribution segment,** execute a separate thread computing a (parameterized) local matrix-vector product
- **Combine** the local vectors determined by the threads (global reduction)

MRD/CRS Sparse Matrix Distribution

0	**53**	0	0	0	0	0	0
0	0	0	0	0	0	**21**	0
19	0	0	0	0	0	0	**16**
0	0	0	0	0	**72**	0	0
0	0	0	**17**	0	0	0	0
0	0	0	0	**93**	0	0	0
0	0	0	0	0	0	**13**	0

0	0	0	0	**44**	0	0	**19**
0	**23**	**69**	0	**37**	0	0	0
27	0	0	**11**	0	0	**64**	0

D^0	C^0	R^0
53	2	1
19	1	2
17	4	2
93	5	3
		3
		4
		5
		5

D^1	C^1	R^1
21	2	1
16	3	1
72	1	2
13	2	3
		4
		4
		4
		5

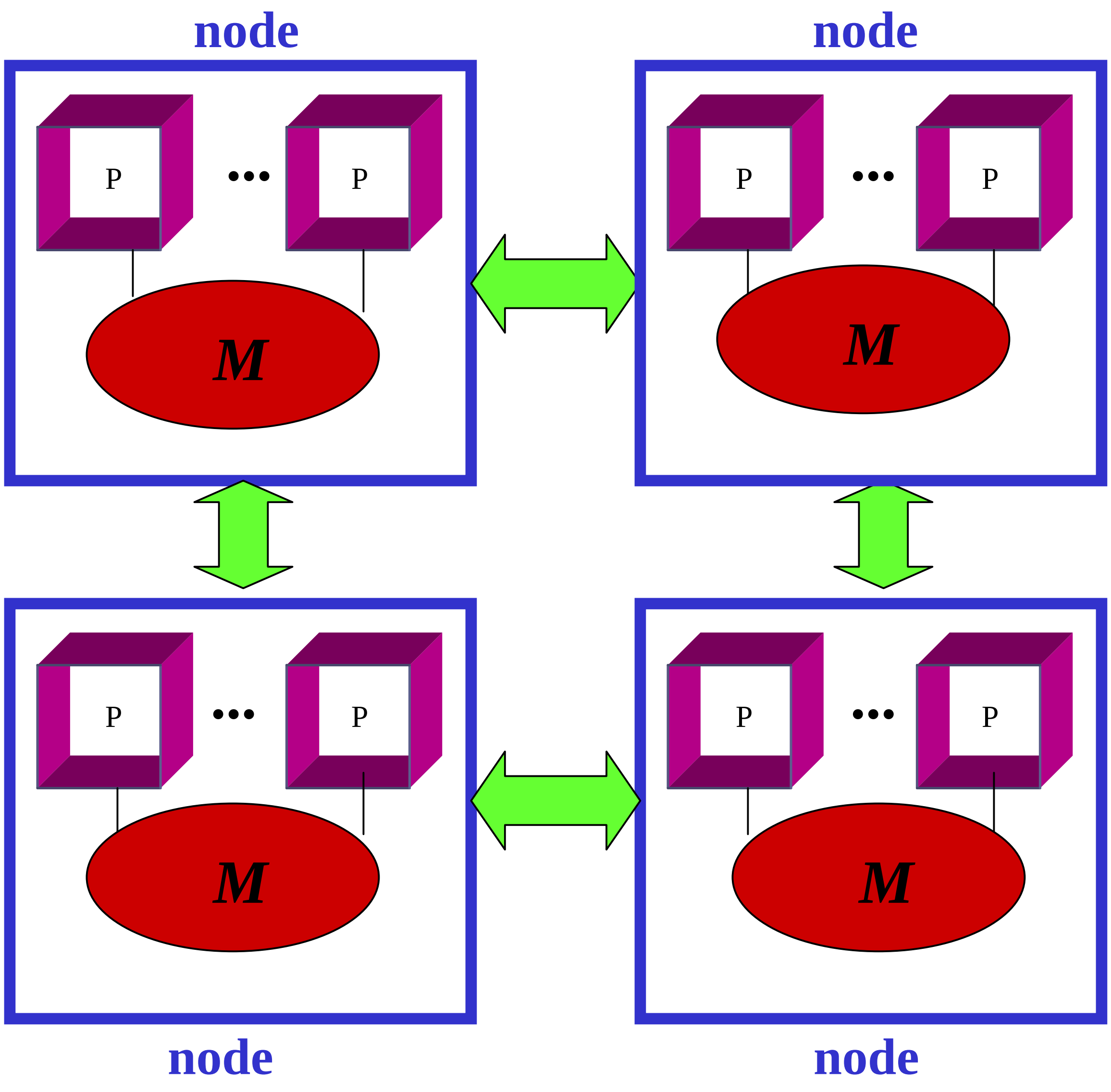
D^2	C^2	R^2
23	2	1
69	3	1
27	1	3
11	4	5

D^3	C^3	R^3
44	1	1
19	4	3
37	1	4
64	3	5

Distributed Sparse Matrix-Vector

```
integer :: NP = number_of_processors()
processors :: P(NP)
real, sparse(CRS(D,C,R,q,L1,U1,L2,U2,...)) :: A(N,M)
     ...
method   mat_vec_loc(u,D,C,R)
real :: D(q(u)); integer I,K,C(q(u)), R(L1(u):U1(u)+1)
do I = L1(u),U1(u)
   TS(u,1:N) = 0.0
   do  K = R(I), R(I+1)-1
         TS(u,I) =  TS(u,I) + D(K) * B(C(K)+L2(u))
    end do
end do
end mat_vec_loc
  …
do independent  u=1:NM, on home (A(L1(u):U1(u),L2(u):U2(u)))
     call mat_vec_loc(u,D(u,:),C(u,:),R(u,:))
  ...
```

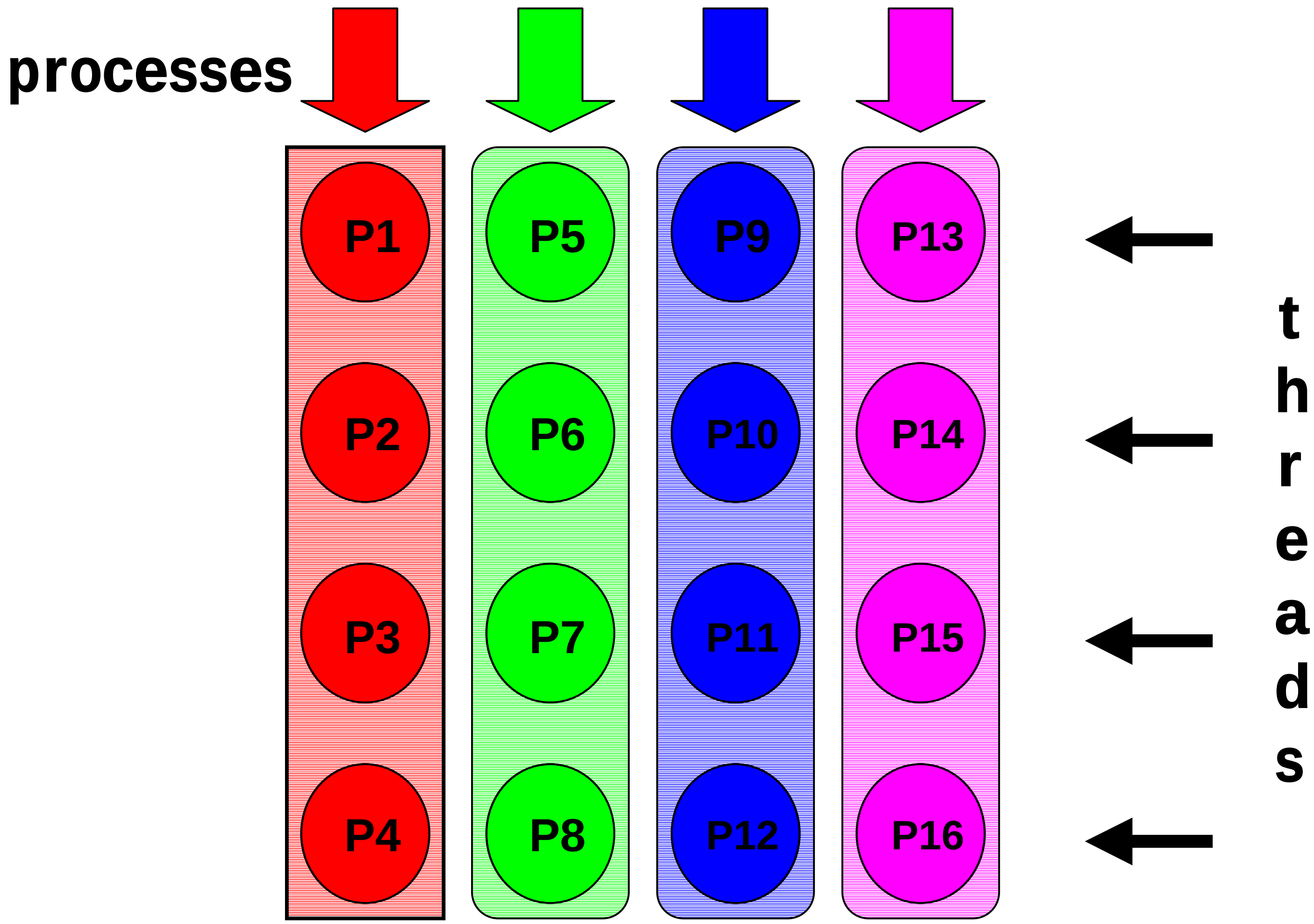
SMP Clusters



Programming Approaches to SMP Clusters

- *MPI only*
- *OpenMP with HPF-like distribution directives*
- *HPF with OpenMP-extrinsics option*
- *HPF-like approach based on topology specification*

Specification of Cluster Topology




```
processors r(4,4)
nodes n(4)
distribute r(*,block)onto n
```

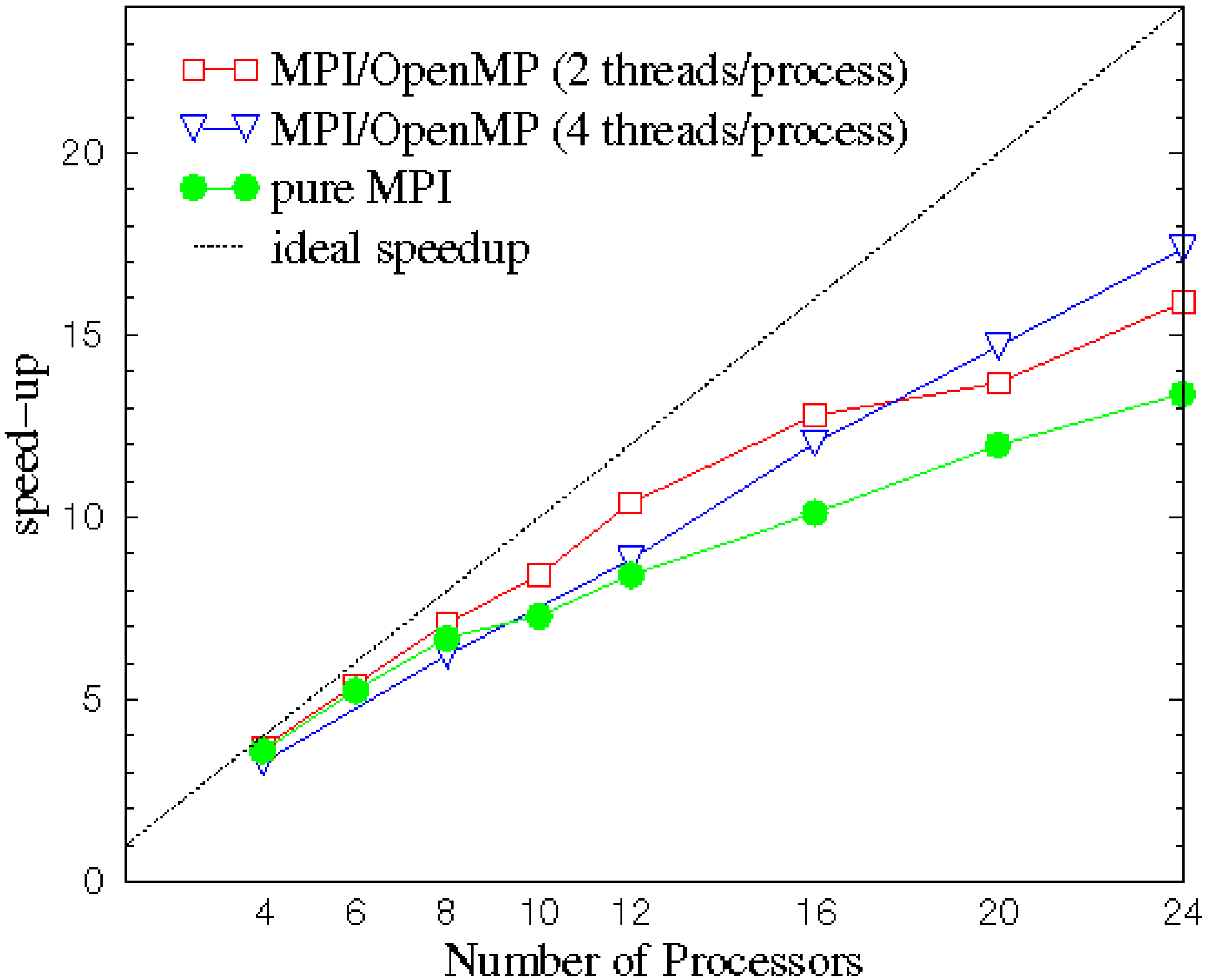
Example:

```
processors r(4,4)
nodes n(4)
distribute r(*,block) onto n

real A(N,M)
distribute(block, block) onto r :: A
```

VFC: Pure MPI vs. OpenMP/MPI

3D Medical Image Reconstruction Kernel on a 6x4 PC Cluster

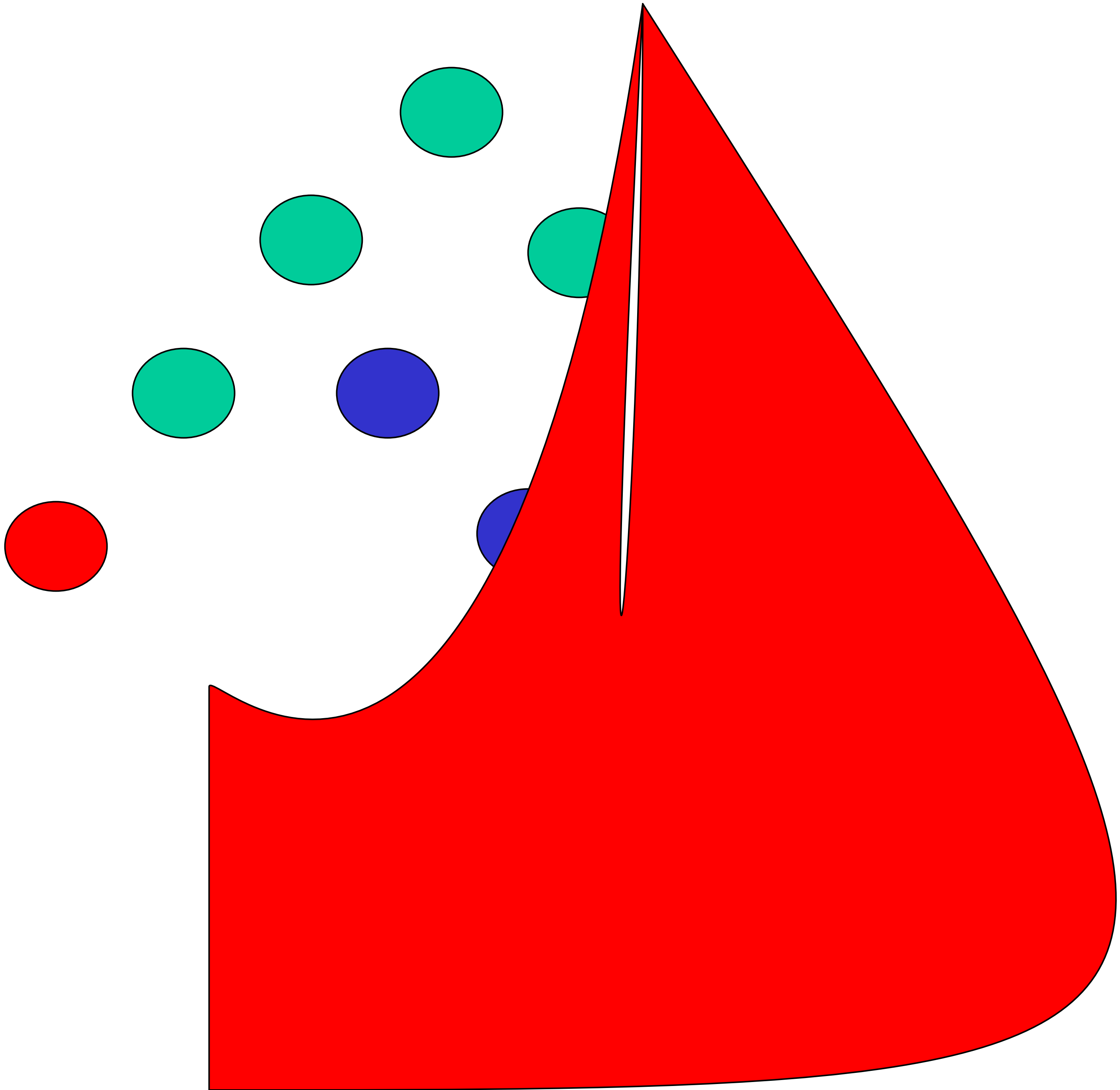


Future Aspects

- ***Generalizing distribution in an object-based framework***
- ***Trends in compiling***
- ***(Semi-)automatic performance tuning***

Generalized Data Distributions

- ***Distribution of arbitrary data structures***
- ***User-defined specifications***
- ***Distributions as first-class objects with methods***
- ***Affinity control***



Distributions as Objects I

- **Distribution object:** a mapping δ: **I --> P(J)**
  - **I:** data structure index domain
  - **J:** processor index domain

- **Distribution type:** a parameterized specification **Δ(I,J,...),** where **I** and **J** represent index domains. An *instantiation*, parameterized by an array index domain, $\mathbf{I}^A$, and a processor index domain, $\mathbf{I}^P$, yields a distribution object $\delta = \Delta(\mathbf{I}^A, \mathbf{I}^P, \ldots)$.

- Distribution types may be *intrinsic* or *user-defined*.

Example: a General Block Distribution

real *A(N1,N2)*
processors :: *P(8)*
distribute(*,*H*) :: *A*

I=[1:N1,1:N2]   P=[1:8]

δ(*,1:h2-1) = {p1}

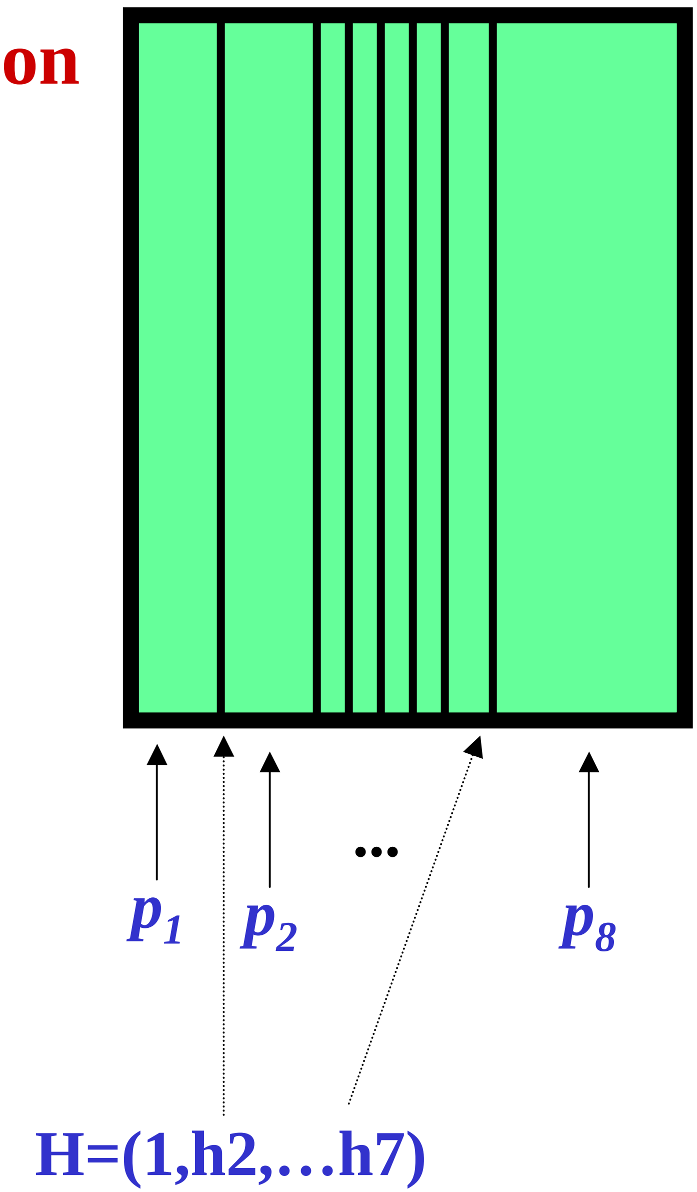
... ***distribution***

δ(*, h7:N2) = {p8}

λ(p1) = [1:N1,1:h2-1]

... ***segments***

λ(p8) = [1:N1, h7:N2]



Example: User-defined Distribution Type

Indirect Distribution

```
dist_type   indirect(map)
array       A(:)
processors  R(:)
integer     map(:)

for every  I=1,size(A)
        map A(I) to R(map(I))
endfor
```

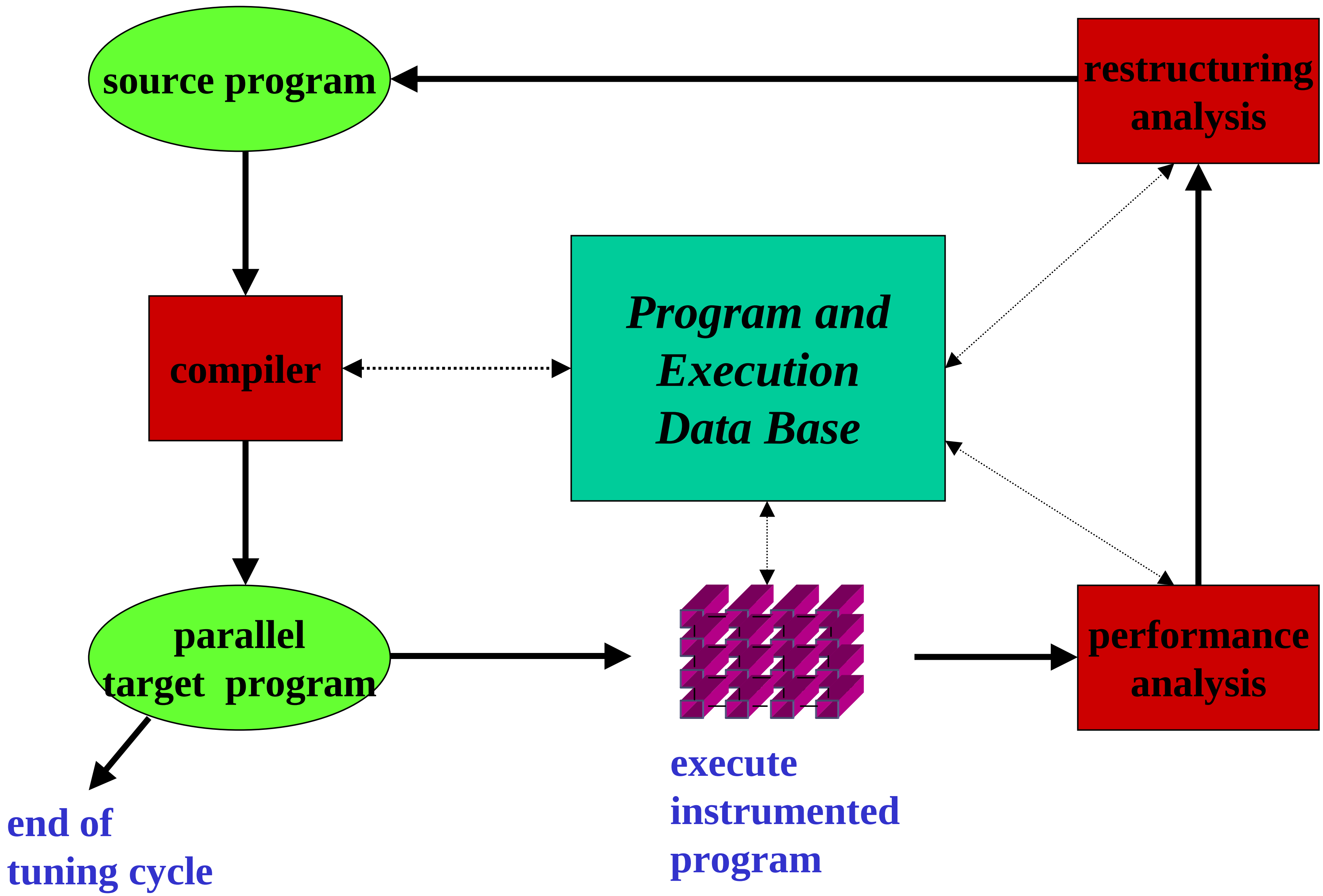
Distributions as Objects II

- ***Intrinsic methods for distribution objects:***
  - **DISTRIBUTE**(δ,A,P) *generate a distributed data structure*
  - **OWNER**(δ,A,P,i) *determine the owner of A(i)*
  - **SEGMENT**(δ,A,P,p) *determine the distribution segment of processor p*
  - **REDISTRIBUTE**(δ,A,P, δ') redistribute based on δ'
  - **INC_REDISTRIBUTE**(δ,A,P,inc) redistribute incrementally based on inc
  - **...**
- **Alignment** ***can be understood in this framework as a class of special distribution constructors***

Some Trends in Compiling

- **runtime compilation:** ***interaction compiler-runtime system***
- **feedback-oriented compilation:** ***interaction of compiler with dynamic performance analysis subsystem***
- **self-adapting software** ***---> Jack Dongarra's talk***
  - ***Atlas***
  - ***PhiPac***
  - ***FFTW***
  - **...**
- **intelligent analysis and restructuring**

Iterative Performance Tuning



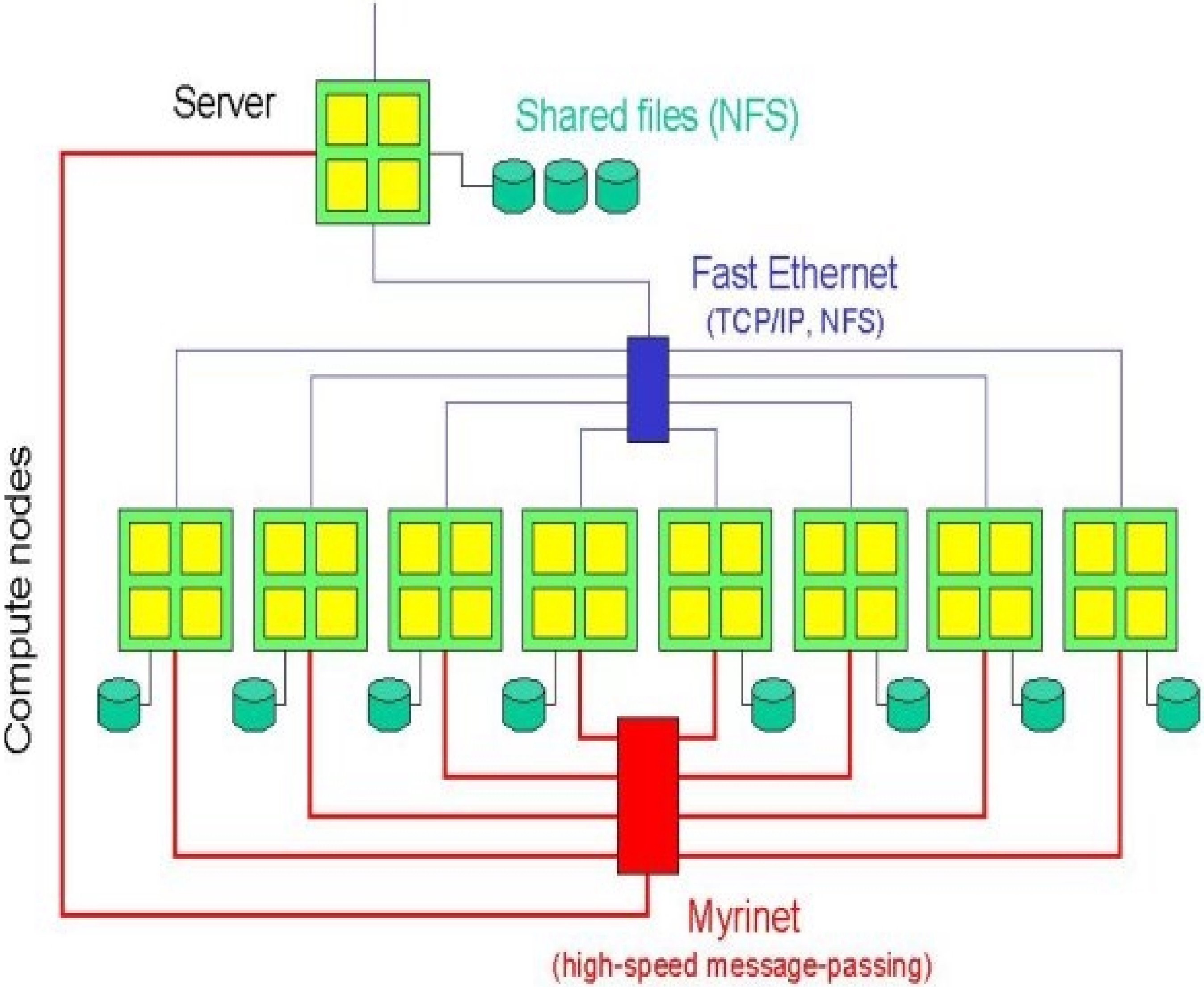
European Cluster Activities

A Few Examples

- ZAMpano Cluster, Juelich, Germany, 8x4 PIII
- Parnass2 Cluster, Bonn, Germany, 72x2 PII
- Gravitor II Cluster, Geneva, Switzerland, 132 PII/PIII
- i-Cluster, Grenoble, France, 216 PIII
- PC2 Cluster, Paderborn, Germany http://www.upb.de/pc2
- SARA Beowulf Cluster,Amsterdam #63 http://www.sara.nl/beowulf
- SUN HPC 4500 400MHz Cluster,Defense, Stockholm, 896 p,#57
- CLIC PIII Cluster, Chemnitz, Germany 528p, #156

http://www.tu-chemnitz.de/urz/anwendungen/CLIC

Juelich ZAMpano Cluster

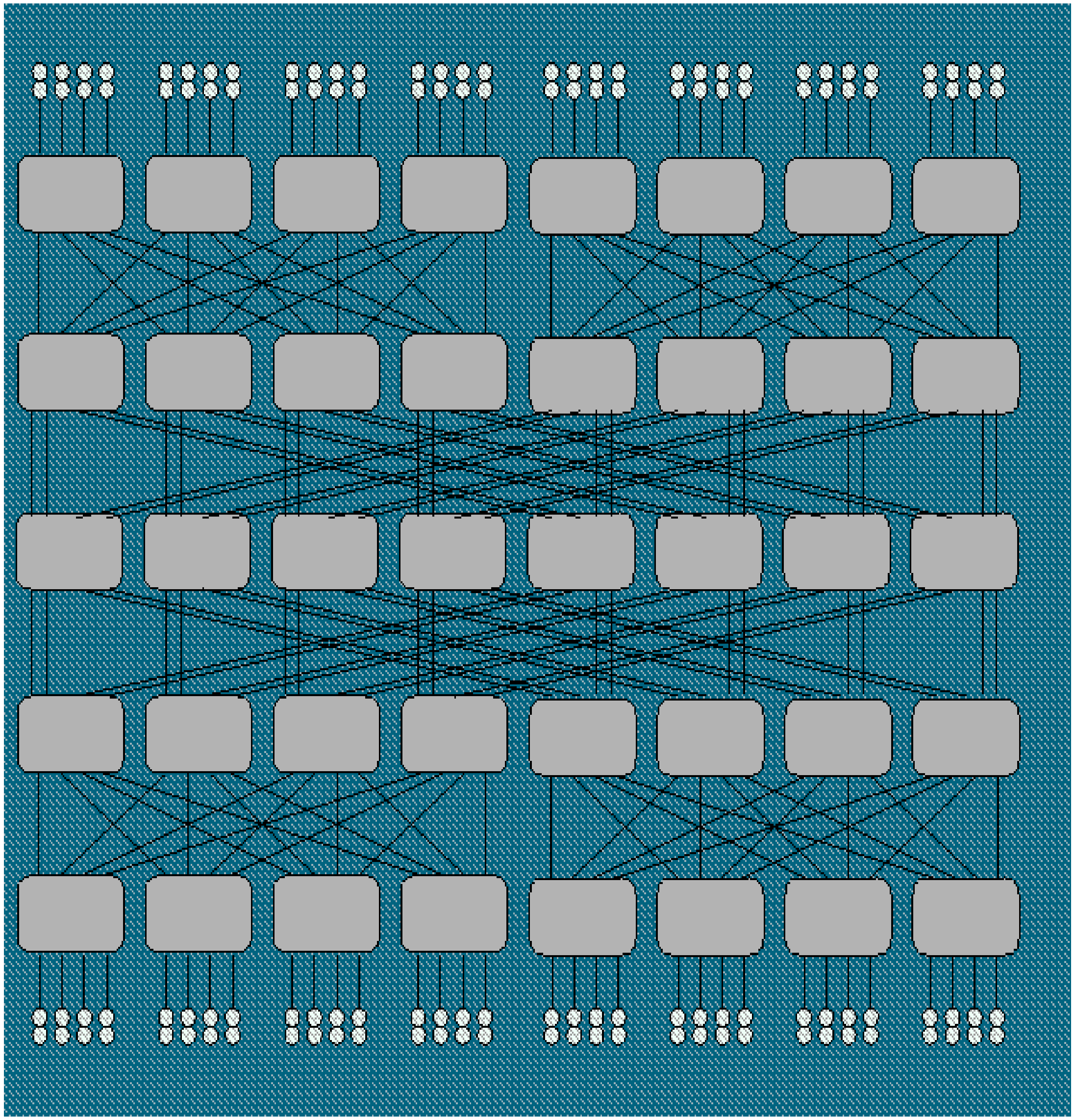


Research focuses on hierarchical systems:

-programming models
-parallel libraries
-performance tools
-application porting
-grid computing

http://zampano.zam.kfa-juelich.de

Bonn Parnass2 Cluster



144 Intel Pentium 2
400 MHz (dual), Myrinet

Applications:

-parallel Navier-Stokes
-molecular dynamics
-algebraic multigrid
-adaptive parallel multigrid
-adaptive parallel sparse grids

http://wissrech.iam.uni-bonn.de/research/projects/parnass2

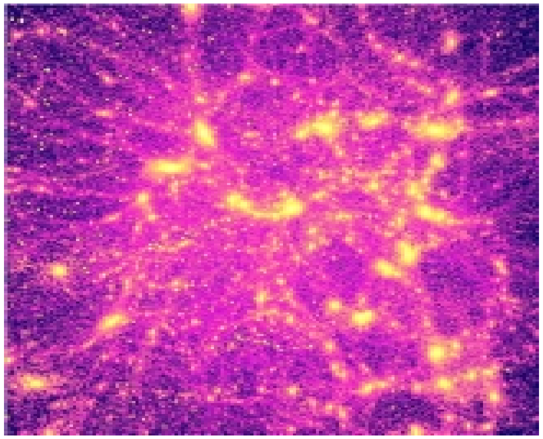
Geneva Gravitor II Cluster

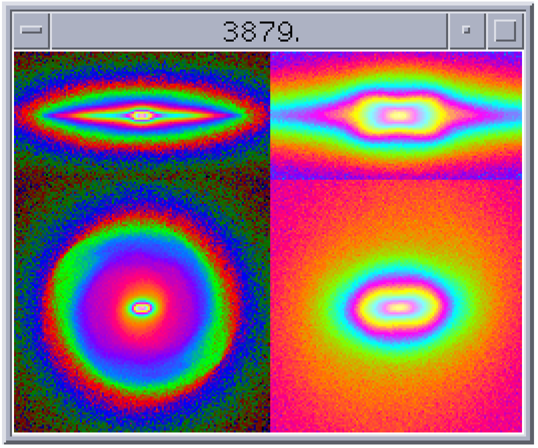
132 Intel Pentium 2 / Pentium 3 (heterogeneous)

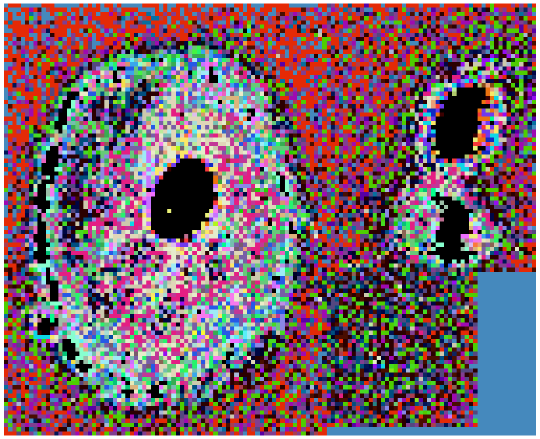
Applications:
Astrophysical Simulations

-accretion of satellit galaxies
-structure of the Milky Way
-fusion of disk galaxies
-pulsars search by data mining
-formation of rotating stars
-dark matter in galaxies

http://obswww.unige.ch/~pfennige/gravitor/gravitor_e.html

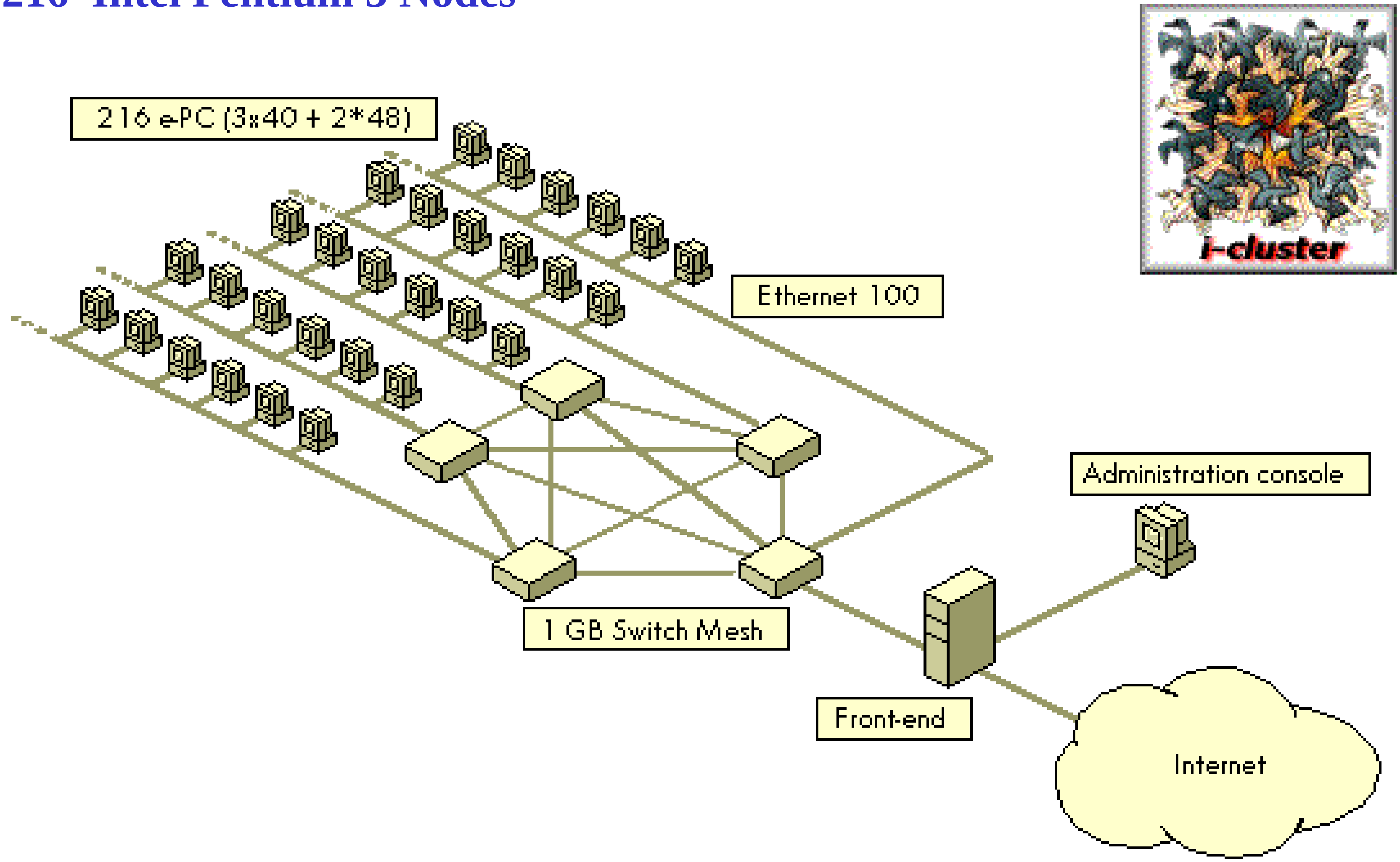






i-Cluster Grenoble

216 Intel Pentium 3 Nodes



http://www-id.imag.fr/Grappes/icluster/materiel.html

Conclusion

- ***Future developments in languages, compilers, and tools will support the transition to higher-level programming***
- ***Hardware developments may include***
  - **new node architectures**
  - **massive parallelism with 1000s of nodes becoming standard**
- ***A uniform and efficient high-level programming model for clusters with hierarchical parallelism is a research problem for some time to come***