ENS Lyon, France

29 March 2006

From high-level parallel programming to high-level grid programming

Marco Aldinucci
Dept. of Computer Science, University of Pisa, Italy
& ISTI - CNR, Pisa, Italy

ENS Lyon, France

29 March 2006

From high-level parallel programming to high-level grid programming

Marco Aldinucci

Dept. of Computer Science, University of Pisa, Italy

& ISTI - CNR, Pisa, Italy

Préparation

- Je suis Italien
  - je ne parle pas Français ... je le compris a petit, pardonnez moi
- Fatalement, l'Anglais est le langage des conférences d'informatique ...
  - ils est important pour connaître autres pays et autres expériences
  - les prochaine diapo en Anglaise ... peut-être ...

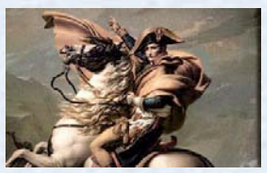
how many mistakes?

Pisa, où il est?

Leonardo Fibonacci (1170 - 1250)
MCLXX -> 1170; 0,1,1,2,3,5,8 ...

Galileo Galilei (1564 - 1642)
falling bodies, telescope, ...

just 100.000 people,
40.000 are students
(they enjoy a lot)



Napoleon (1769 - 1821)
founded ENS Pisa
(just one in Italy,
just 20 students every year, 1810
2 of them in Computer Science)

Ulisse Dini (1845 - 1918)
F(x,y)=0 -> y=f(x) (locally)

CPU

Enrico Fermi (1901 - 1954)

CEP, Pisa, Italy, 1957,
3 first computer built in Italy

The MIM seminar

- Part 0: kidding, up to now
  - just to make you aware I speak another language or two, but not French
- Part I: a very short introduction
  - no prerequisites, almost all of you already know what I'll say
- Part II: high-level parallel programming
  - little prerequisites, some of you might know what I'll say
  - might give you some hints for your research
- Part III: high-level parallel programming in Grid with dynamic Quality of Service control
  - conference level, technical, ASSIST environment (our research)
  - lot of open problems both theoretical and practical (not sure I will have the time to present them)

Part I

- Parallel programming
  - very short introduction
  - low-level mechanisms & libraries
- I'll run quite fast here
  - stop me in any moment if needed
  - pleeease don't be timid

IN SUPREMÆ DIGNITATIS 1343

Traditional // prog. models

- In charge to the programmer:
  - Defining logically / physically parallel activities
  - Scheduling and mapping of parallel activities
  - Communication / shared memory access handling
  - Synchronization

    ...
  - Load balancing
  - Fault tolerance

IN SUPREMÆ DIGNITATIS 1343

Defining parallel activities

- Goal: define {/ / activities} potentially parallel
  - Concurrent activities ☞ parallel
  - Logically shared data ☞ shared data/communications
- Implicit models
  - Derive parallel activities from plain sequential code
  - Data flow analysis ☞ independent activities
- Explicit models
  - Threads
  - Processes

IN SUPREMÆ DIGNITATIS 1343

Interaction models

- Shared memory
  - Synchronization (locks, semaphores, monitors, …)
  - "Native" data representation
- Message passing
  - Synchronization (send/receive, barriers, …)
  - Data representation (XDR, marshalling, …)
  - Wide range of communication mechanisms:
    - Send/receive
    - RPC/RMI

IN SUPREMÆ DIGNITATIS 1343

State of the art tools

- Shared memory
  - POSIX threads (and derivatives)
  - JAVA threads (and derivatives)

- Message passing
  - TCP/IP socket API
  - MPI, PVM, ...
  - (RPC)

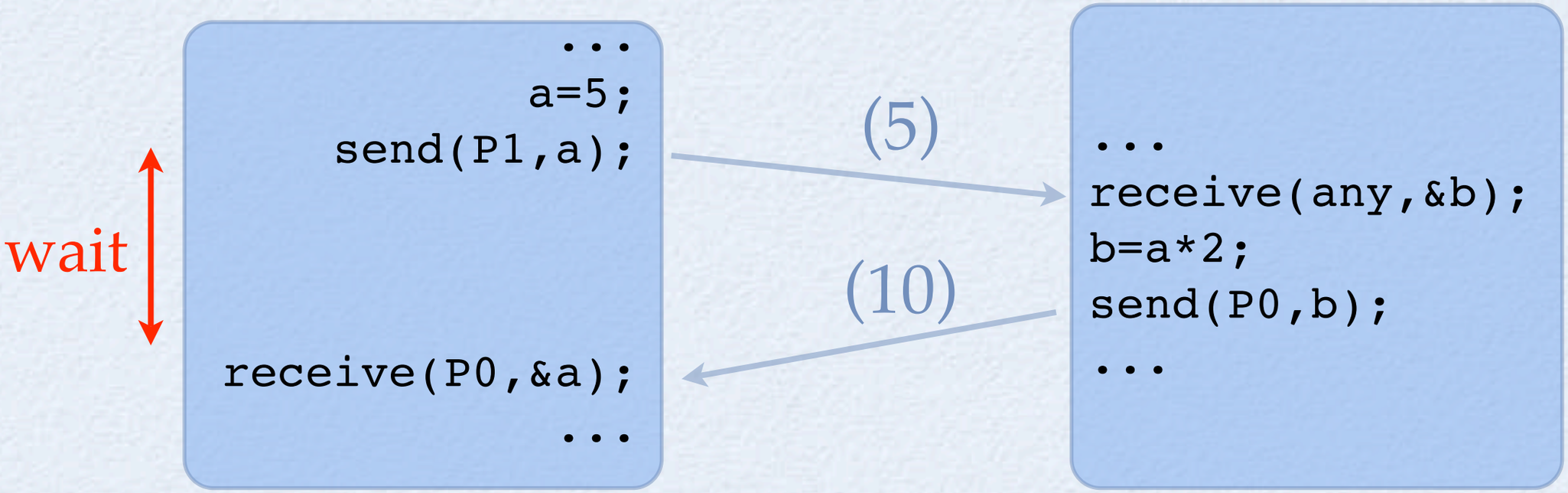
Shared memory models

- Processes + System V semaphores
- Threads (e.g. Java, POSIX)
  - Extends Thread - or - implements Runnable
  - public void run() { /* body of thread */}
  - synchronizations: monitor
  - public synchronized int incr() {…}
    - … while(cond) { … wait(); … }
    - … notify();
    - … notifyAll();
- Distributed Shared Memories ...

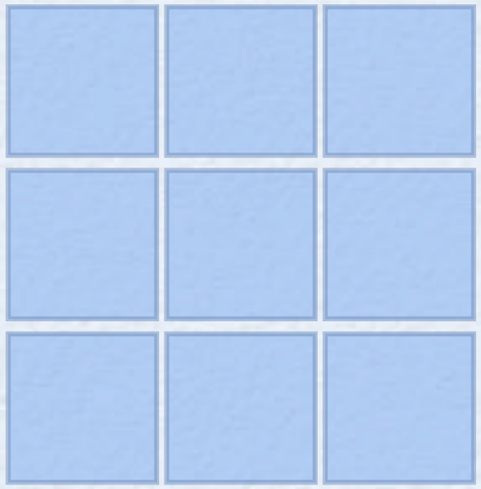
message passing (sync)

```
...
a=1;
send(P1,a);
receive(P0,&a);
...
```

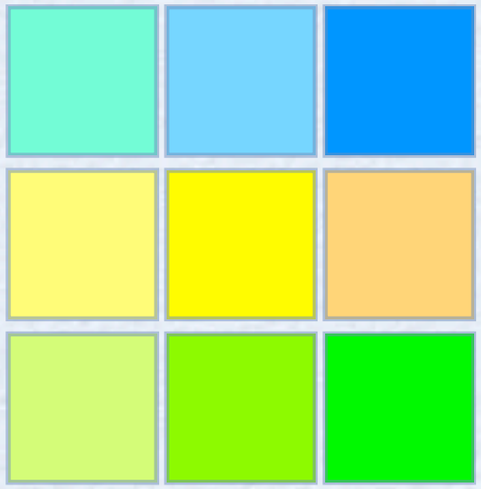
```
...
receive(any,&b);
b=a*2;
send(P0,b);
...
```



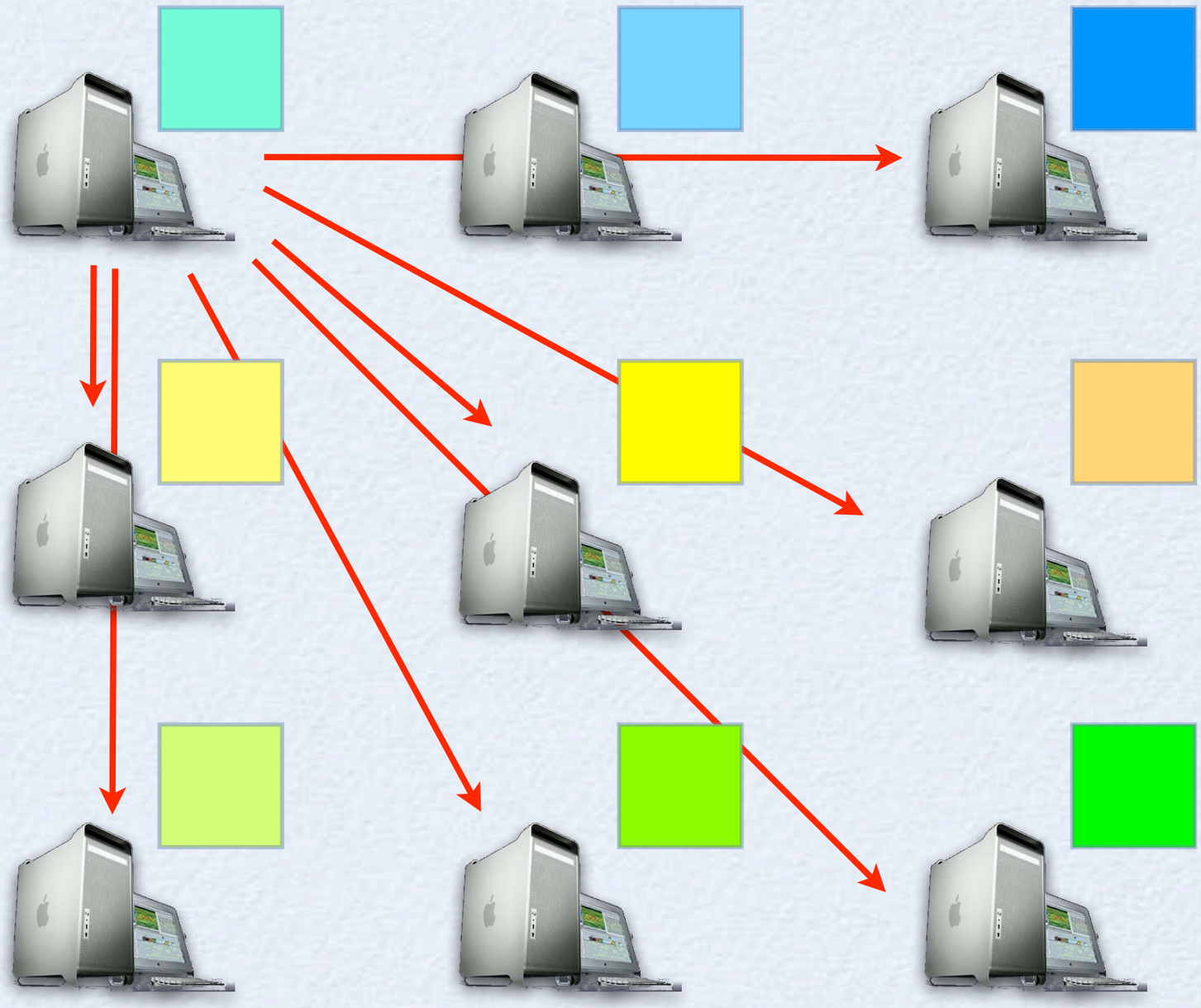
scatter, then gather



scatter, then gather

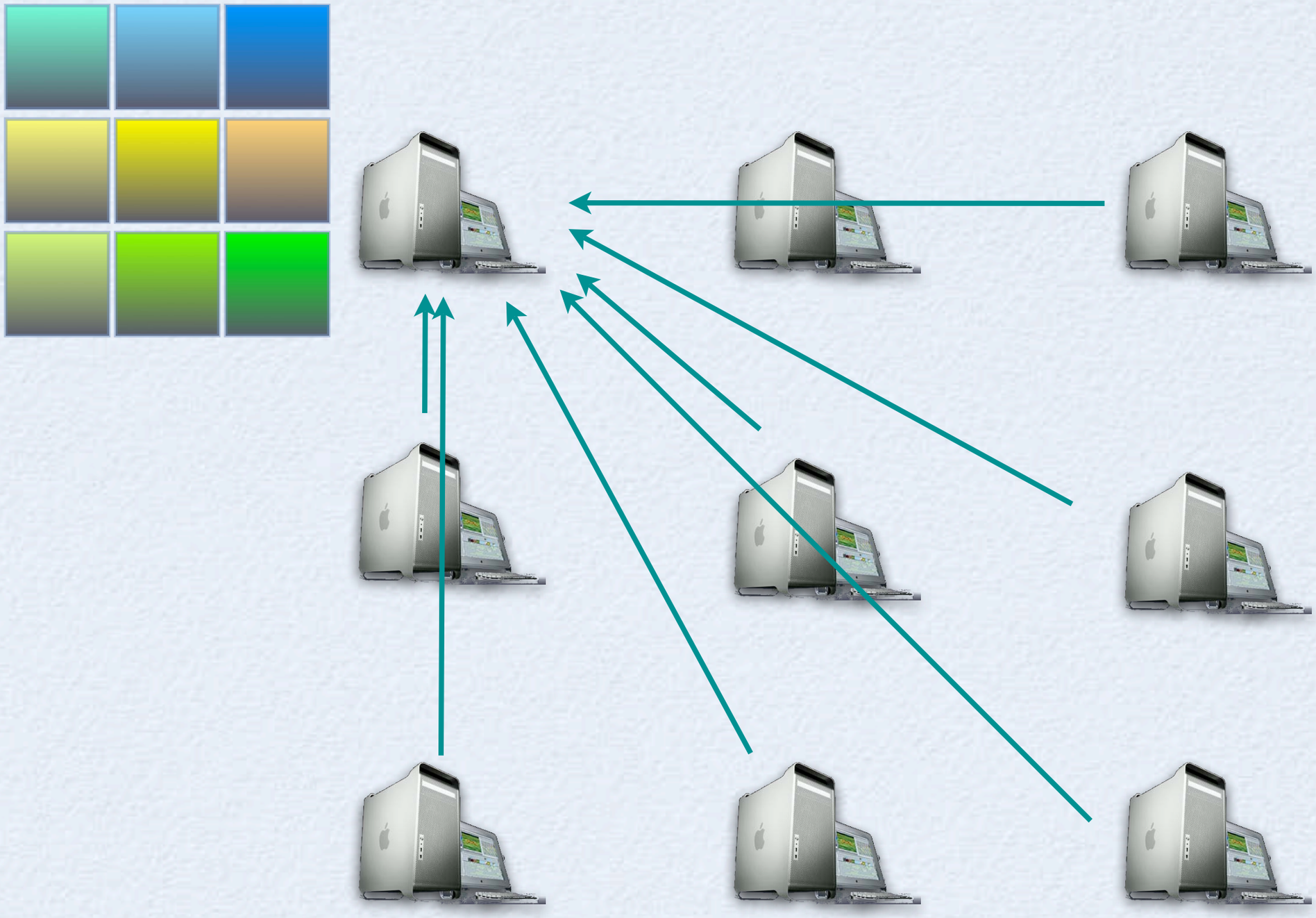


scatter, then gather



scatter, then gather

scatter, then gather



the MPI code

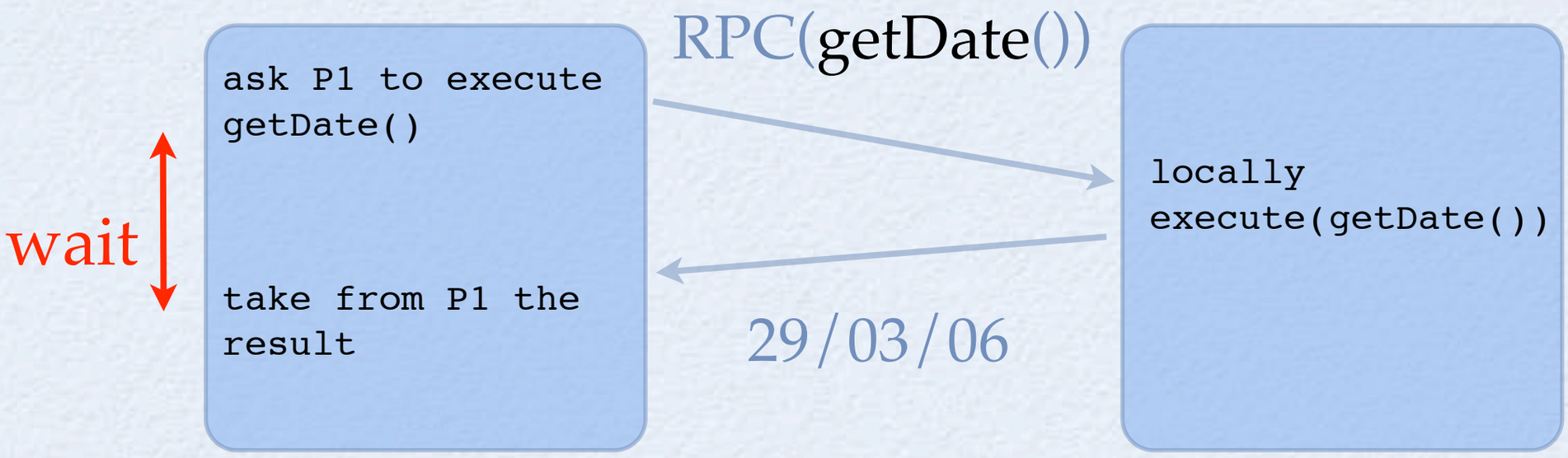
```
#include <stdio.h>
#include "mpi.h"
#define MAXPROC 8    /* Max number of procsses */
#define NAMELEN 80   /* Max length of machine name */
#define LENGTH 24    /* Lengt of send buffer is divisible by 2, 4, 6 and 8
*/

main(int argc, char* argv[]) {
  int i, j, np, me;
  const int nametag  = 42;    /* Tag value for sending name */
  const int datatag  = 43;    /* Tag value for sending data */
  const int root = 0;         /* Root process in scatter */
  MPI_Status status;          /* Status object for receive */

  char myname[NAMELEN];             /* Local host name string */
  char hostname[MAXPROC][NAMELEN];  /* Received host names */

  int x[LENGTH];        /* Send buffer */
  int y[LENGTH];        /* Receive buffer */

  MPI_Init(&argc, &argv);               /* Initialize MPI */
  MPI_Comm_size(MPI_COMM_WORLD, &np);   /* Get nr of processes */
  MPI_Comm_rank(MPI_COMM_WORLD, &me);   /* Get own identifier */

  gethostname(&myname, NAMELEN);    /* Get host name */

  if (me == 0) {    /* Process 0 does this */

    /* Initialize the array x with values 0 .. LENGTH-1 */
    for (i=0; i<LENGTH; i++) {
      x[i] = i;
    }

    /* Check that we have an even number of processes and at most MAXPROC
*/
    if (np>MAXPROC || np%2 != 0) {
      printf("You have to use an even number of processes (at most %d)\n",
MAXPROC);
      MPI_Finalize();
      exit(0);
    }

    printf("Process %d on host %s is distributing array x to all %d
processes\n\n", \
           me, myname, np);

    /* Scatter the array x to all proceses, place it in y */
    MPI_Scatter(&x, LENGTH/np, MPI_INT, &y, LENGTH/np, MPI_INT, root, \
                MPI_COMM_WORLD);

    /* Print out own portion of the scattered array */
    printf("Process %d on host %s has elements", me, myname);
    for (i=0; i<LENGTH/np; i++) {
      printf(" %d", y[i]);
    }
    printf("\n");

    /* Receive messages with hostname and the scattered data */
    /* from all other processes */
    for (i=1; i<np; i++) {
      MPI_Recv (&hostname[i], NAMELEN, MPI_CHAR, i, nametag,
MPI_COMM_WORLD, &status);
      MPI_Recv (&y, LENGTH/np, MPI_INT, i, datatag, MPI_COMM_WORLD,
&status);
      printf("Process %d on host %s has elements", i, hostname[i]);
      for (j=0; j<LENGTH/np; j++) {
        printf(" %d", y[j]);
      }
      printf("\n");
    }

    printf("Ready\n");

  } else { /* all other processes do this */

    /* Check sanity of the user */
    if (np>MAXPROC || np%2 != 0) {
      MPI_Finalize();
      exit(0);
    }

    /* Receive the scattered array from process 0, place it in array y */
    MPI_Scatter(&x, LENGTH/np, MPI_INT, &y, LENGTH/np, MPI_INT, root, \
                MPI_COMM_WORLD);
    /* Send own name back to process 0 */
    MPI_Send (&myname, NAMELEN, MPI_CHAR, 0, nametag, MPI_COMM_WORLD);
    /* Send the received array back to process 0 */
    MPI_Send (&y, LENGTH/np, MPI_INT, 0, datatag, MPI_COMM_WORLD);

  }

  MPI_Finalize();
  exit(0);
}
```


IN SUPREMÆ DIGNITATIS 1343

RPC (sync)

```
public class DateClient {
    public static void main ...
    ...
    DateServer dateServer =
   (DateServer)Naming.lookup("rmi://" +
    args[0] + "/DateServer");

    Date when = dateServer.getDate();

    ...
}}
```

```
public static void main (...
  ...
  DateServerImpl dateS = new DateServerImpl();
  Naming.bind("DateServer", dateS);
}
```



P0 (client, active)

P1 (server, passive)

RPC example: Java RMI

```
// REMOTE INTERFACE

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Date;

public interface DateServer extends Remote {
    public Date getDate() throws RemoteException;
}

// CLIENT
import java.rmi.RMISecurityManager;
import java.rmi.Naming;
import java.util.Date;

public class DateClient {
    public static void main (String args[]) throws
Exception {
   if (args.length != 1)
       throw new RuntimeException("Syntax: DateClient
<hostname>");

   System.setSecurityManager(new RMISecurityManager());

   DateServer dateServer = (DateServer)Naming.lookup
("rmi://" + args[0] + "/DateServer");

   Date when = dateServer.getDate();

   System.out.println(when);
    }
}
```

```
// SERVER
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;
import java.rmi.Naming;
import java.util.Date;

public class DateServerImpl extends UnicastRemoteObject
implements DateServer {
    public DateServerImpl() throws RemoteException {

    }

    public Date getDate() {
   return new Date();
    }

    public static void main (String args[]) throws
Exception {
   System.setSecurityManager(new RMISecurityManager());
   DateServerImpl dateS = new DateServerImpl();
   Naming.bind("DateServer", dateS);
    }
}
```

Too complex? Not enough ...

- lot of code for a so simple paradigms
- lot of static/lunch-time assumptions
  - n. of Processing Elements and their names
  - size of the matrix, number of blocks, order of distribution
- lot of architectural assumptions
  - no firewalls, homogenous (data types) and reliable machines and net, ...
- performances, load balancing?
  - depends on the regularity of the computation
  - depends on the actual load of the machines
- Is it possible to address these problems?
  - Yes of course, by adding more and more code ...

Low-level // programming

- Usually libraries
  - shared-memory (e.g. POSIX threads, DSM, ...)
  - message passing (e.g. POSIX sockets, MPI, PVM, ...)
  - orchestration code mixed with application code (e.g. mapping, scheduling, data distribution, fault-tolerance, caching, ...)
- Time consuming
  - programming, debugging
  - performance tuning
- Tailored for specific architectures
  - difficult to be ported on different platforms
  - not a good investment ...

IN SUPREMÆ DIGNITATIS 1343

Part II

- High-level parallel programming
  - what kind of problems it address
  - an overview of some environments
    - BSP (I'll not show you, Frederic Loulergue already did it)
    - HPF (just for historical reasons, people no longer believe in it ...)
    - OpenMP
    - design patterns and skeletons (...)
    - components (not shown)

IN SUPREMÆ DIGNITATIS 1343

High Performance Fortran

- Extension of the Fortran90
  - pragma for declaring parallelism
  - foremost paradigms of parallelism:
    - FORALL, DO INDEPENDENT
  - computes-owner rule
  - extremely difficult build a good compiler
    - data dependencies are entangled by indexes
- The project can be considered trespassed
  - but very important, at least to know what concepts are very very difficult to implement

HPF: example 1

An Example of FORALL

Initially,

a = [0, 1, 2, 3, 4]

b = [0, 10, 20, 30, 40]

c = [-1, -1, -1, -1, -1]

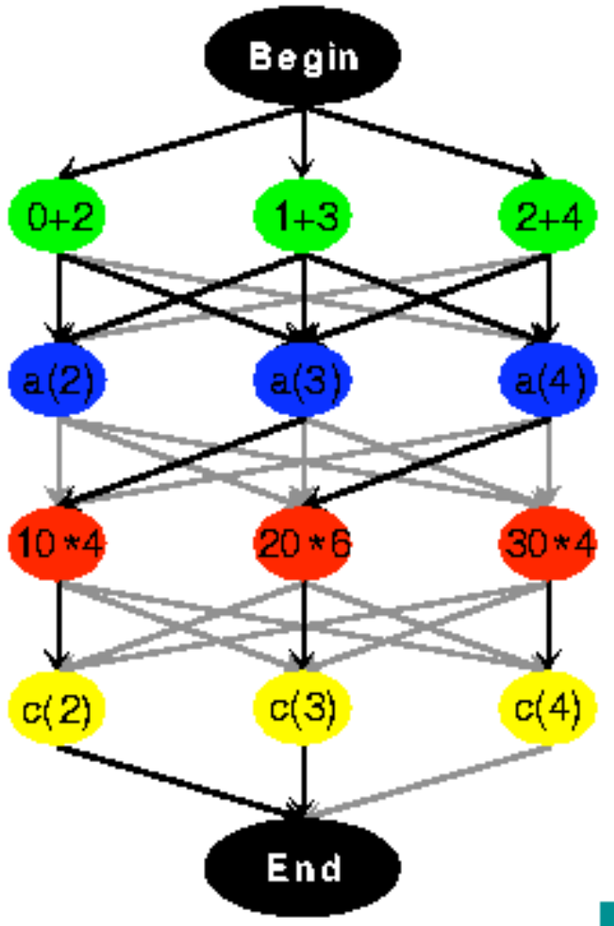
```
FORALL ( i = 2:4 )
   a(i) = a(i-1) + a(i+1)
   c(i) = b(i) * a(i+1)
END FORALL
```

Afterwards,

a = [0, 2, 4, 6, 4]

b = [0, 10, 20, 30, 40]

c = [-1, 40, 120, 120, -1]



CRPC

So, what is the problem?

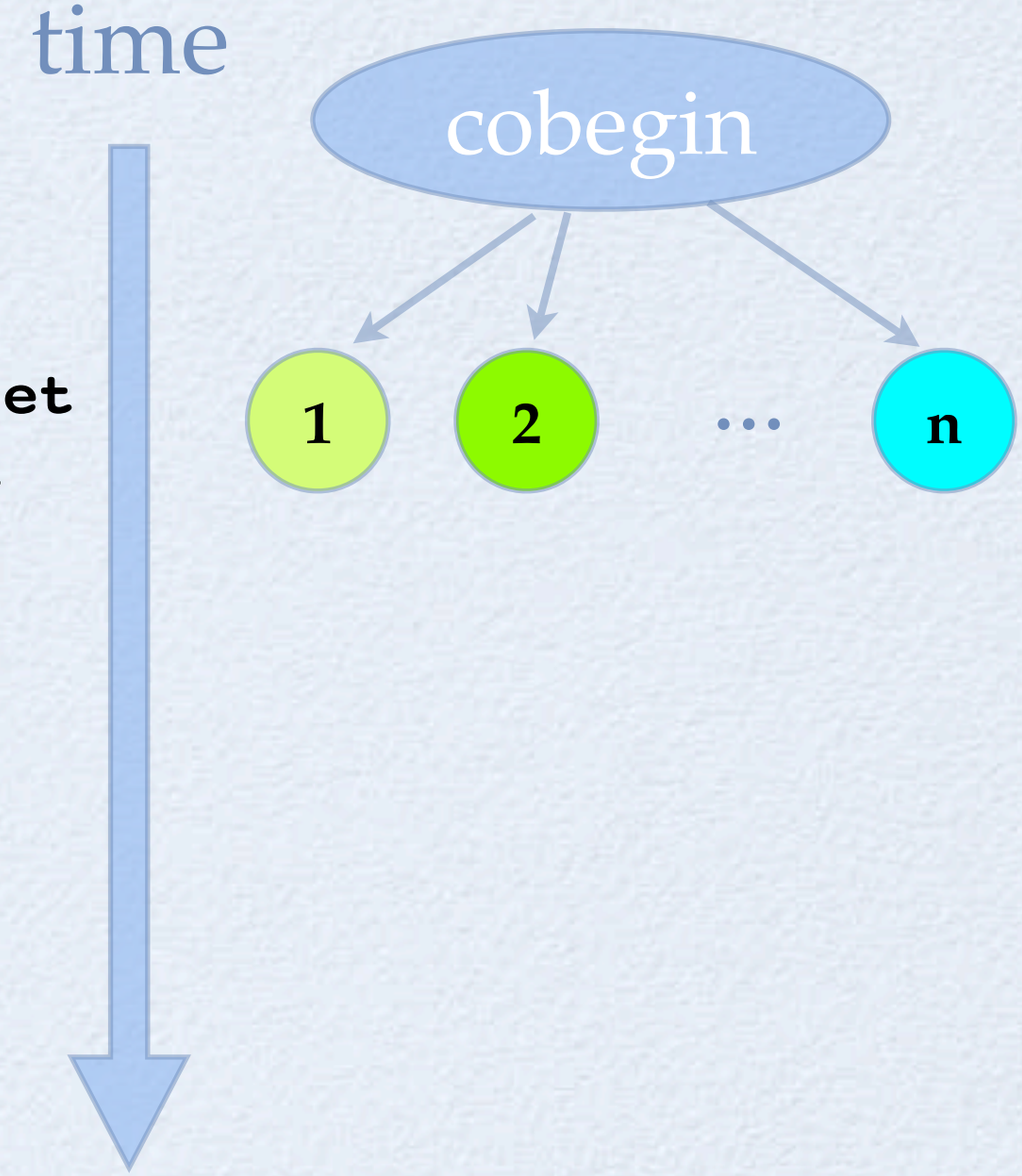
- In many cases the "arrows" shown in the previous slide are neither known at compile time (e.g. a[i] = b[f(i)], f function) nor stable across iterations
- thus, it is almost impossible to automatically derive good mapping of data onto processors
- thus performance may become rapidly disappointing

OpenMP

- Thought for shared memory machines
  - The “arrows” problem no longer exist (arrows exists but simply cost less because of the shared memory)
  - no mapping problem (because of the shared memory)
- main target: parallelization of loops
  - co-begin/co-end model
- Core elements of OpenMP:
  - thread creation
  - work load distribution(work sharing)
  - data environment management
  - thread synchronization

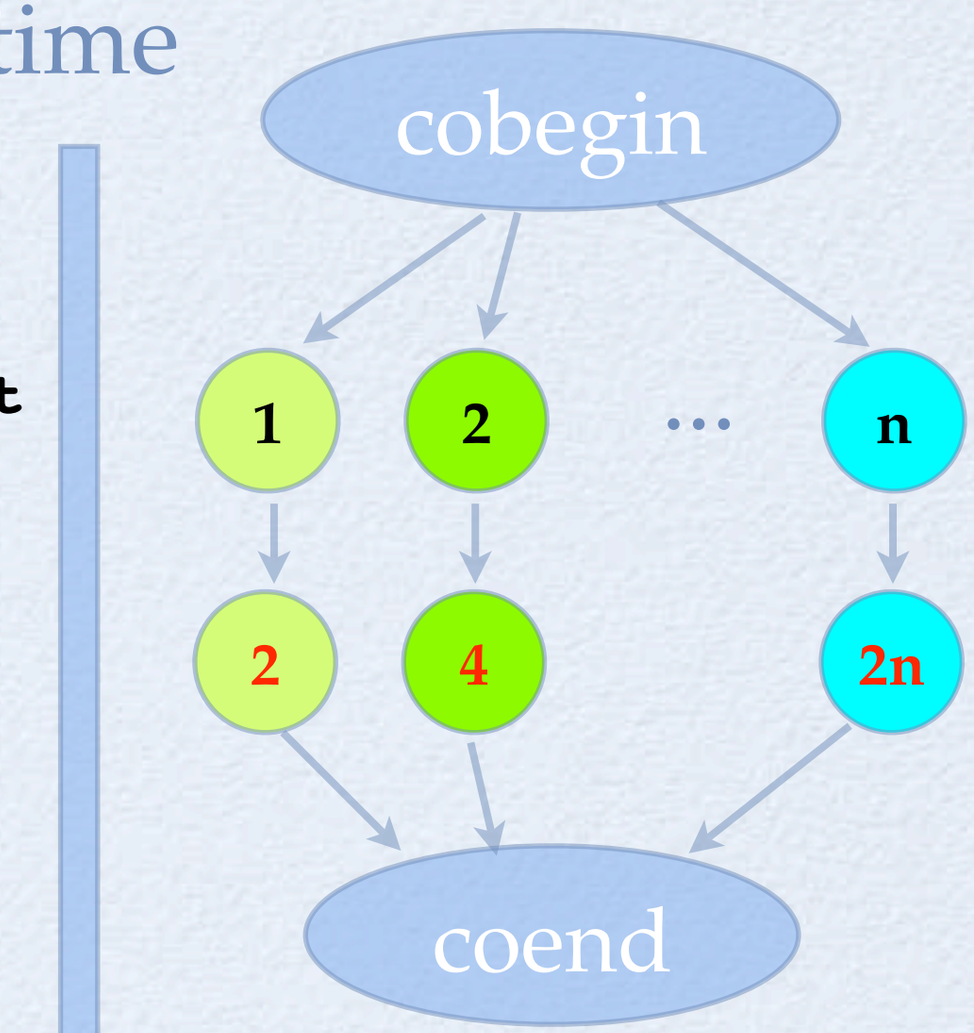
OpenMP: Parallel for

```
#include <stdio.h>
#ifdef _OPENMP
/* using conditional compilation to let
sequential compilers ignore the omp.h
header*/
#include <omp.h>
#endif
#define n 100000
void main()
{
int a[n];
int i;
#pragma omp parallel
#pragma omp for
  for (i=0;i<n;i++) a[i]= 2*i;
}
```



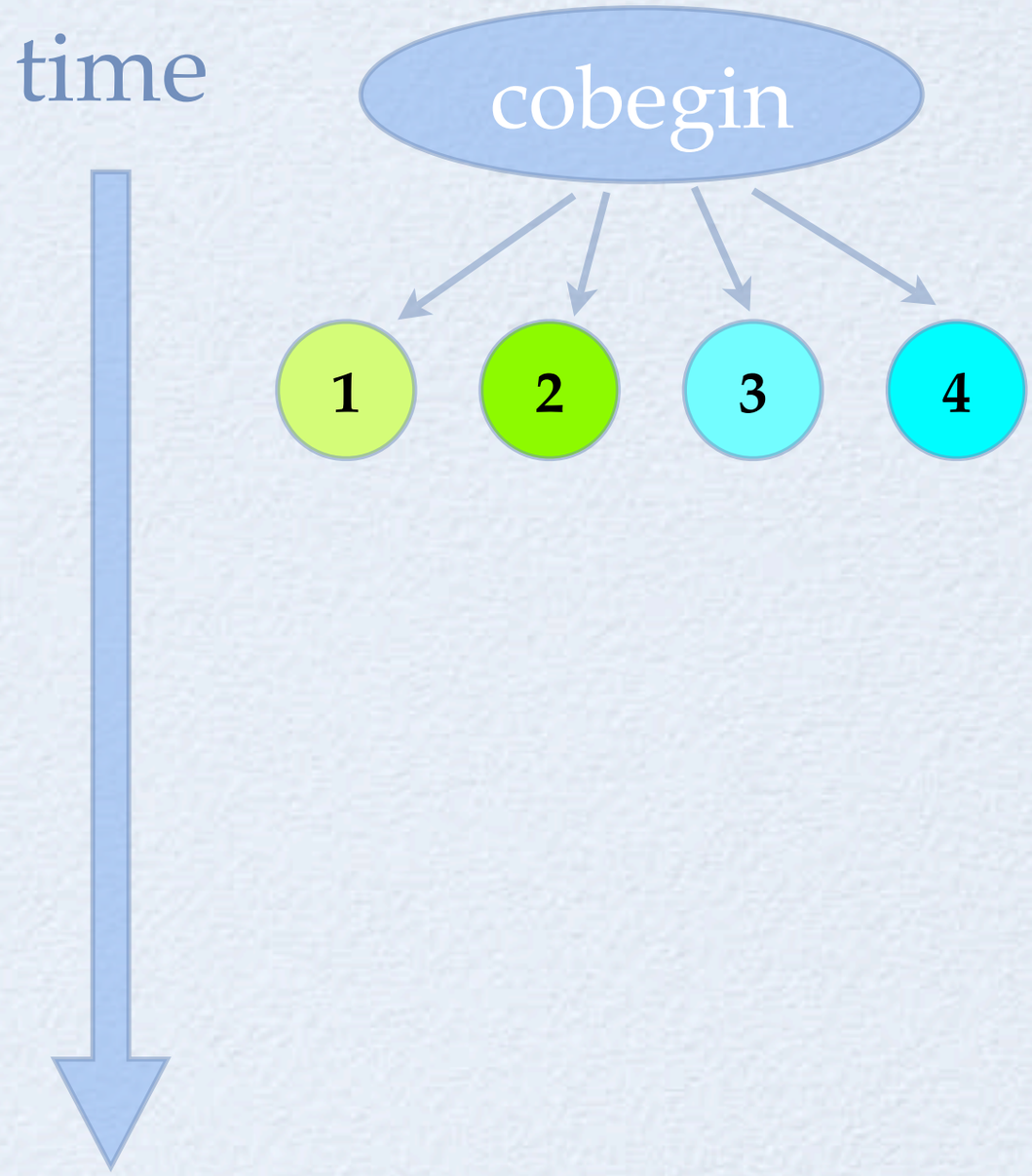
OpenMP: Parallel for

```
#include <stdio.h>
#ifdef _OPENMP
/* using conditional compilation to let
sequential compilers ignore the omp.h
header*/
#include <omp.h>
#endif
#define n 100000
void main()
{
int a[n];
int i;
#pragma omp parallel
#pragma omp for
  for (i=0;i<n;i++) a[i]= 2*i;
}
```



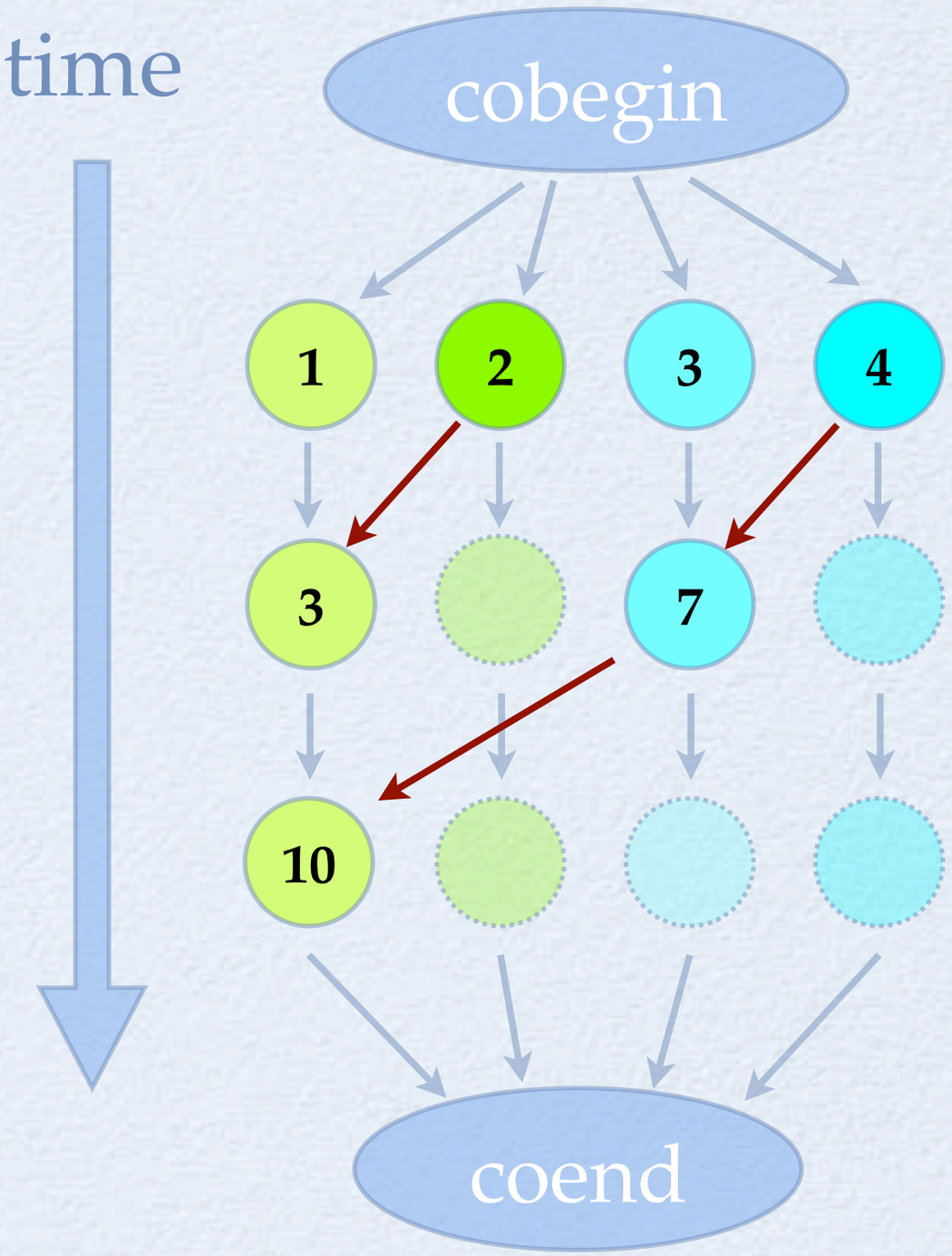
OpenMP: reduction

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {

int   i, n;
float a[100], b[100], sum;

/* Some initializations */
n = 100;
for (i=0; i < n; i++)
  a[i] = b[i] = i * 1.0;
sum = 0.0;

#pragma omp parallel for reduction(+:sum)
  for (i=0; i < n; i++)
    sum = sum + (a[i] * b[i]);

printf("   Sum = %f\n",sum);

}
```



OpenMP: reduction

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {

int   i, n;
float a[100], b[100], sum;

/* Some initializations */
n = 100;
for (i=0; i < n; i++)
  a[i] = b[i] = i * 1.0;
sum = 0.0;

#pragma omp parallel for reduction(+:sum)
  for (i=0; i < n; i++)
    sum = sum + (a[i] * b[i]);

printf("   Sum = %f\n",sum);

}
```



Balance …

- simple: need not deal with message passing as MPI does
- data layout and decomposition is handled automatically by directives.
- incremental parallelism: can work on one portion of the program at one time, no dramatic change to code is needed.
- a unified code for both serial and parallel applications: OpenMP constructs are treated as comments when sequential compilers are used.
- Original (serial) code statements need not, in general, be modified when parallelized with OpenMP. This reduces the chance of inadvertently introducing bugs.

- currenty only run efficiently in shared-memory multiprocessor platforms
- requires a compiler that supports OpenMP. Visual C++ 2005 supports it, and so do the Intel compilers for their x86 and IPF product series. GCC 4.2 will support OpenMP, though it is likely that some distributors will add OpenMP support already to their GCC 4.1 based system compilers.
- low parallel efficiency: rely more on parallelizable loops, leaving out a relatively high percentage of a non-loop code in sequential part.

Skeletons: the principle

The new system presents the user with a selection of independent "algorithmic skeleton", each of which describes the structure of a particular style of algorithm, in the way in which "higher order functions" represent general computational frameworks in the context of functional programming languages. The user must describe a solution to a problem as an instance of the appropriate skeleton.

(Cole 1988)

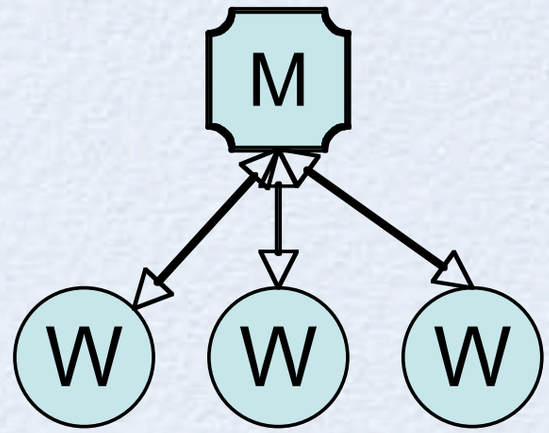
The principle (rephrased)

- Abstract parallelism exploitation pattern by parametric code (higher order function)
- Provide user mechanism to specify the parameters (sequential code, extra parameters)
- Provide (user protected) state-of-the-art implementation of each parallelism exploitation pattern
- In case, allow composition
  - Fundamental, second time property of skeletons systems

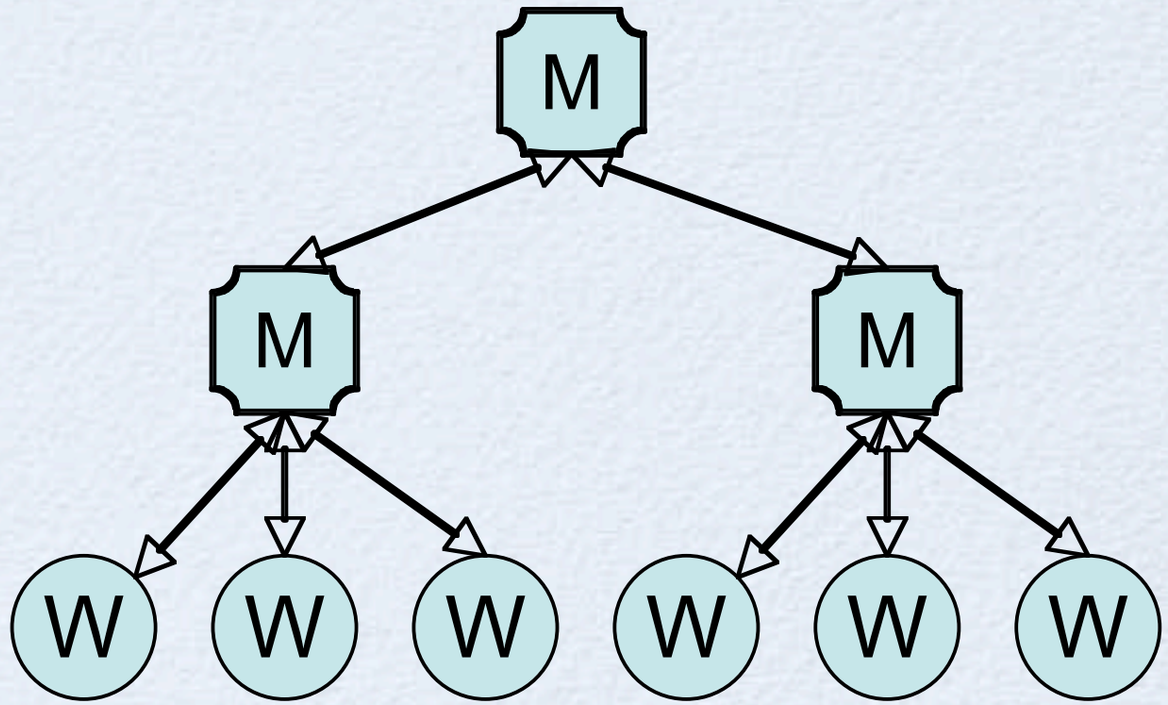
IN SUPREMÆ DIGNITATIS 1343

Example: task farm

- Parameters:
  - Worker code
  - Parallelism degree (computed?)



- Known implementation
  - Master slave pattern
  - Possibly distributed master



- Composite worker
  - Master to master optimizations

other examples ...

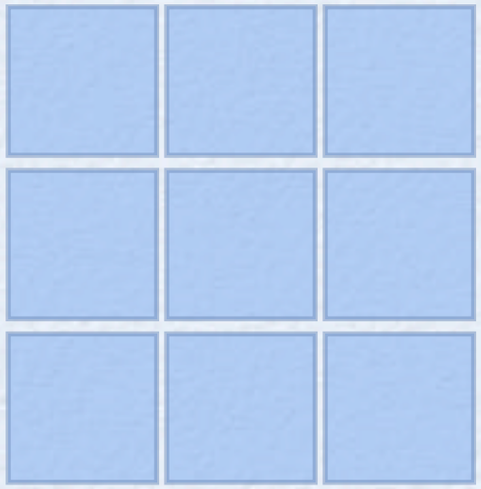
- Data parallel
  - map, fold, reduce,
  - haloswap
  - Divide&Conquer

  ...
- Control parallel
  - farm
  - pipeline
  - DAG, graph, ...

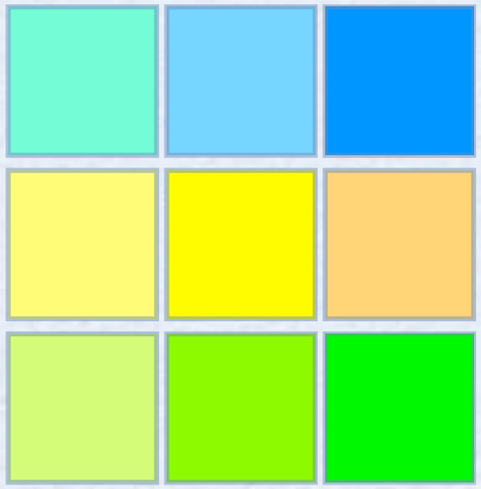
map

- functionally: apply the same function to each of the partitions of a domain
  - well known in functional programming
- parallel behavior: once data is partitioned, the partitions can be independently crunched
  - depending on initial data layout, a the map may be trammeled by a scatter-gather pair

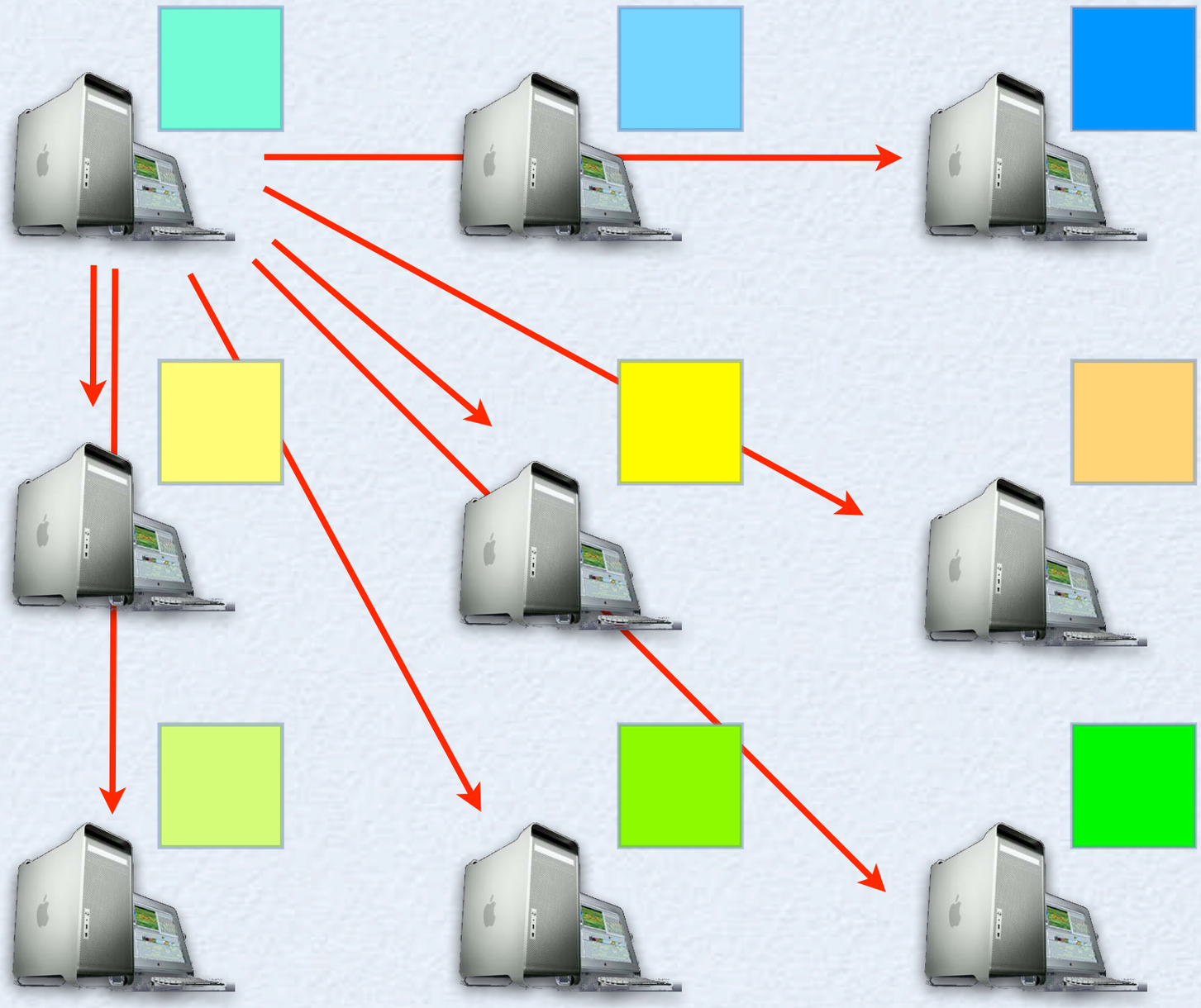
scatter-map-gather



scatter-map-gather

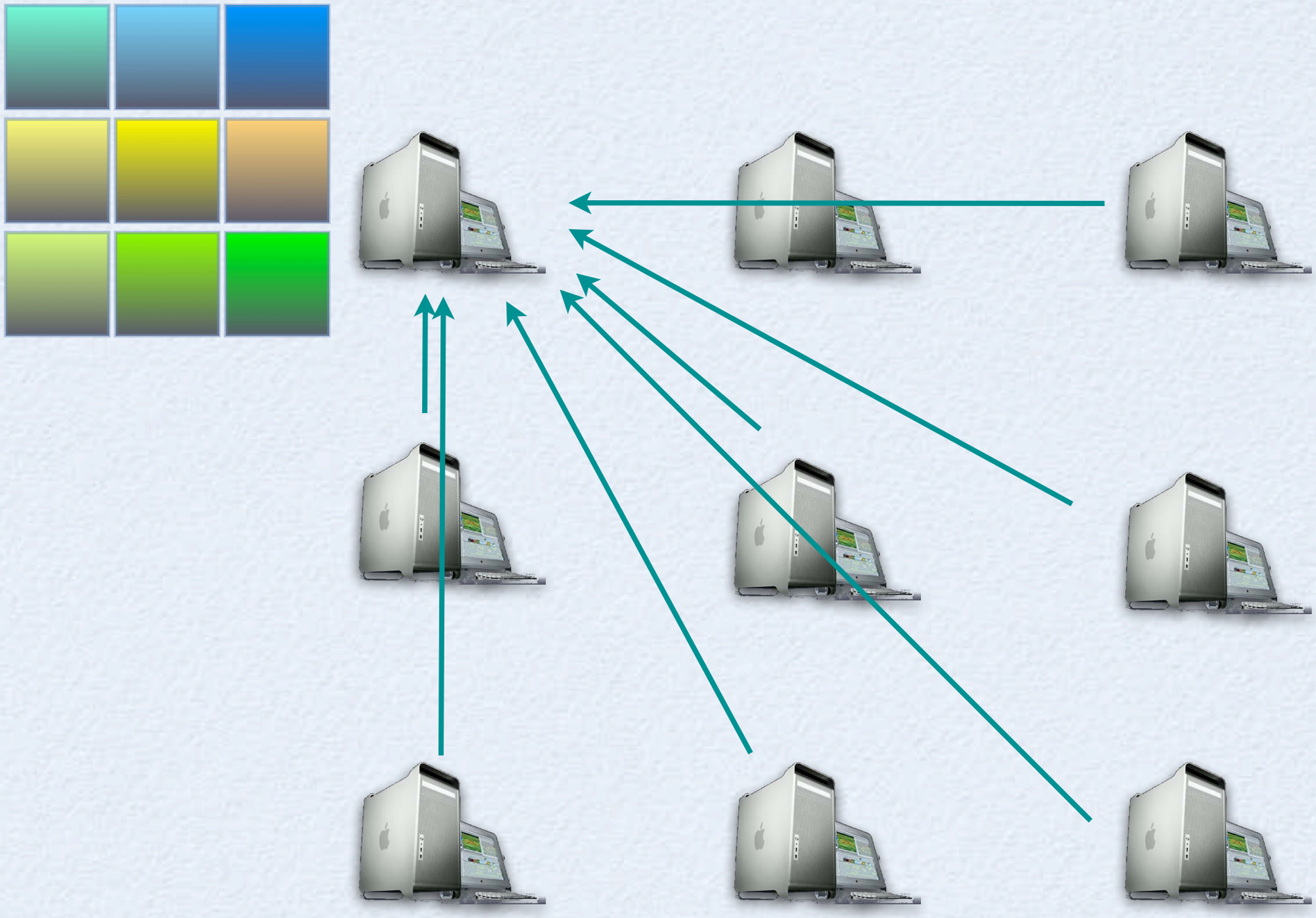


scatter-map-gather



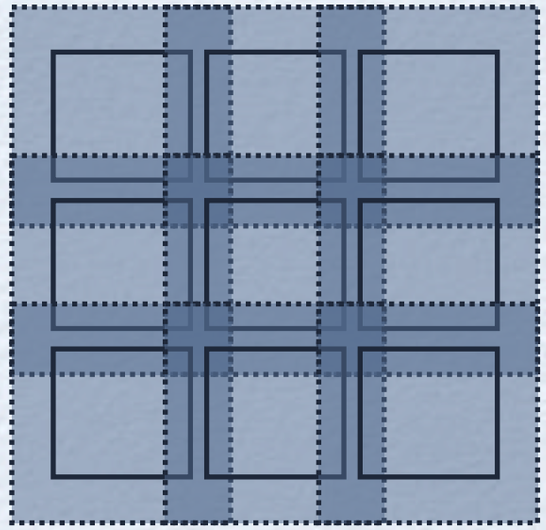
scatter-map-gather

scatter-map-gather



haloswap

- similar to map, but the initial data is divided in parts which are not partitions
  - some data (halo) appears in more than one parts
  - in the case the data is kept in distributed form, some more communications are needed
  - since data in halos is replicated, it should be somehow kept coherent (usually just one PE can write it)



Since high-level ...

- we know the semantics
  - functional behavior
  - parallel behavior
- it can be used to
  - provide good implementation
  - optimize programs
  - develop tools to tune (statically, dynamically) the program to the running environment

EXEC RULES ($\rightarrow$)

$\overline{1}.$ $\overline{\mathsf{seq}}\ f\ x^{\ell} \xrightarrow{\ell} f\ x^{\ell}$

$\overline{2}.$ $\overline{\mathsf{farm}}\ \overline{\Delta}\ x^{\ell} \xrightarrow{\ell} \overline{\Delta}\ x^{\mathcal{O}(\ell,x)}$

$\overline{3}.$ $\overline{\mathsf{pipe}}\ \overline{\Delta}_1\overline{\Delta}_2\ x^{\ell} \xrightarrow{\ell} \overline{\Delta}_2\ \mathcal{R}^{\mathcal{O}(\ell,x)}\ \overline{\Delta}_1\ x^{\ell}$

$\overline{4}.$ $\overline{\mathsf{comp}}\ \overline{\Delta}_1\overline{\Delta}_2\ x^{\ell} \xrightarrow{\ell} \overline{\Delta}_2\ \overline{\Delta}_1\ x^{\ell}$

$\overline{5}.$ $\overline{\mathsf{map}}\ p^{-1}\overline{\Delta}\ p\ x^{\ell} \xrightarrow{\ell} p^{-1}\ (\alpha\ \overline{\Delta})\ p\ x^{\ell}$

$\overline{6}.$ $\overline{\mathsf{d\&c}}\ t\ p^{-1}\ \overline{\Delta}\ p\ x^{\ell} \xrightarrow{\ell} \begin{cases} \overline{\Delta}\ x^{\ell} & \textit{iff}\ \ (t\ x) \\ p^{-1}\ (\ \alpha\ (\ \overline{\mathsf{d\&c}}\ t\ p^{-1}\ \overline{\Delta}\ p\ ))\ p\ x^{\ell} & \textit{otherwise} \end{cases}$

$\overline{7}.$ $\overline{\mathsf{while}}\ t\ \overline{\Delta}\ x^{\ell} \xrightarrow{\ell} \begin{cases} \overline{\Delta}\ x^{\ell} & \textit{iff}\ \ (t\ x) \\ \overline{\mathsf{while}}\ t\ \overline{\Delta}\ x^{\ell} & \textit{otherwise} \end{cases}$

$$\frac{\overline{\Delta}\ x^{\ell_2} \xrightarrow{\ell_2} y^{\ell_3}}{\mathcal{R}^{\ell_1}\ \overline{\Delta}\ x^{\ell_2} \xrightarrow{\ell_2} y^{\Phi(\ell_1\ell_2,0)}}\ \textit{relabel} \qquad \frac{\overline{\Delta}_1\ x^{\ell} \xrightarrow{\ell} y^{\jmath}}{\overline{\Delta}_2\ \overline{\Delta}_1\ x^{\ell} \xrightarrow{\ell} \overline{\Delta}_2\ y^{\jmath}}\ \textit{context}$$

$$\frac{p\ x^{\ell} = \langle\ y_1^{\ell_1}, \cdots, y_n^{\ell_n}\ \rangle \quad \overline{\Delta}\ y_i^{\ell_i} \xrightarrow{\ell_i} z_i^{\ell_i} \quad p^{-1}\ \langle\ z_1^{\ell_1}, \cdots, z_n^{\ell_n}\ \rangle = z \quad \begin{array}{c} i = 1..n \\ \Psi(\ell, x) = \ell_1 \cdots \ell_n \end{array}}{p^{-1}\ (\alpha\ \overline{\Delta})\ p\ x^{\ell} \xrightarrow{\ell_1,\cdots,\ell_n} z^{\ell_1,\cdots,\ell_n}}\ dp$$

$$\frac{\overline{\Delta}_i\ x_i^{\ell_i} \xrightarrow{\ell_i} y_i^{\jmath_i} \quad \forall i\ 1 \le i \le n \quad \wedge \quad \Gamma_1 \not\rightarrow \quad \wedge \quad \exists\ i,j\ 1 \le i,j \le n,\ \ell_i = \ell_j \Rightarrow i = j}{\Gamma_1 : \overline{\Delta}_1\ x_1^{\ell_1} : \cdots : \overline{\Delta}_n\ x_n^{\ell_n} : \Gamma \xrightarrow{\ell_1,\cdots,\ell_n} \Gamma_1 : y_1^{\jmath_1} : \cdots : y_n^{\jmath_n} : \Gamma}\ sp$$

M. Aldinucci and M. Danelutto, Computer Languages, Elsevier, 2006

Skeletons: evolution

Cole PhD (1988)

Fixed degree DC, Iterative combination, Cluster Task queue

Darlington (1992)

Pipeline, Farm, RaMP, DMPA

SCL Fortran S

P3L (1991)

Pipeline, Farm, Map, Reduce

SkIE ASSIST Lithium OcamlP3L

BMF (‘80)

map fold reduce prefix + algebra

MALLBA (‘00)

Combinatorial optimisation

Serot (1999)

Skipper (→MDF)

Skillicorn (mid ‘90)

Gorlatch (late ‘90)

HOC (early ‘00)

Kuchen Skil (1998)

Muesli (2002)

Pipeline, Farm, Parallel array + collectives

eSkel (2002)

Parametric skeletons + Give/Take

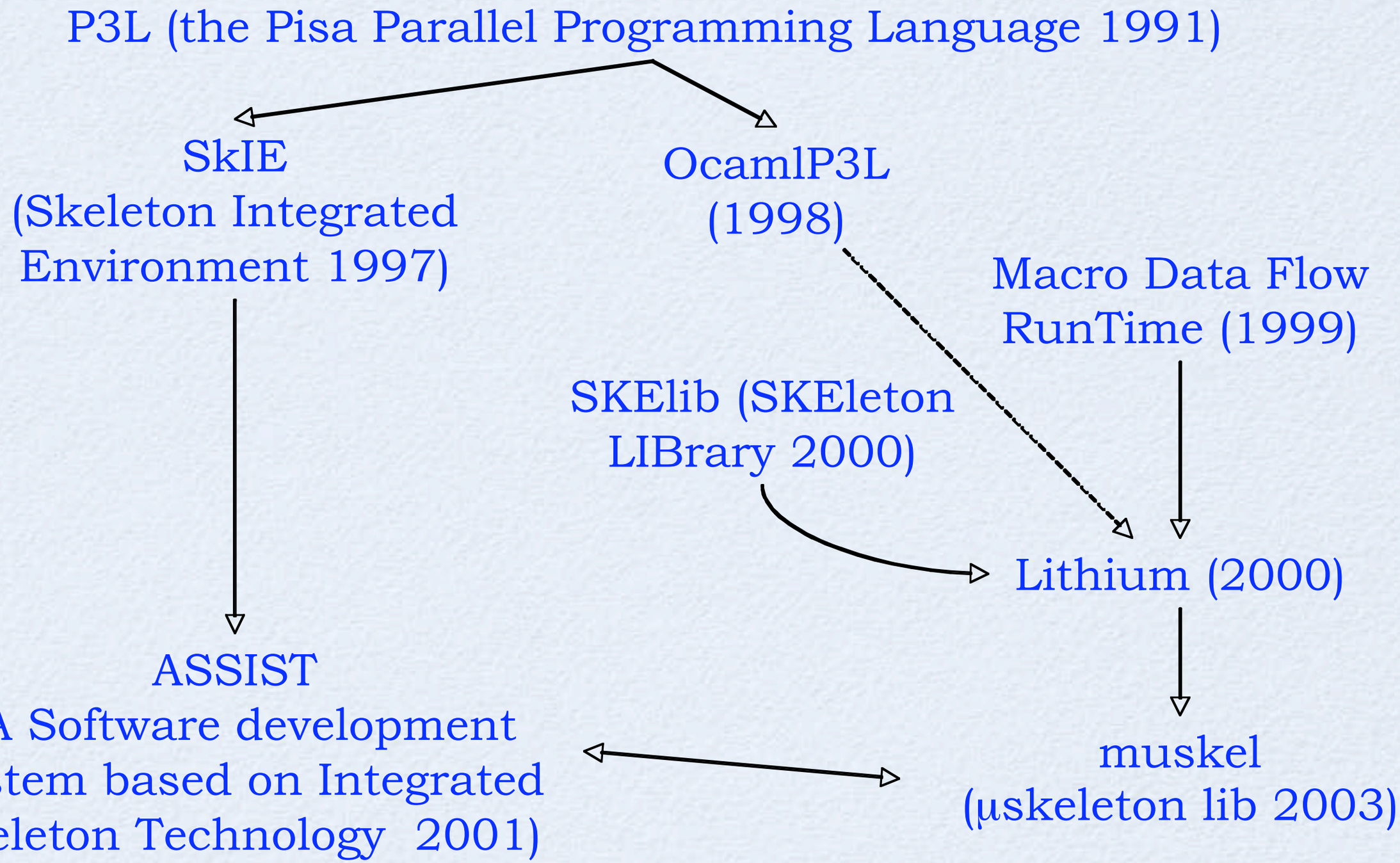
eSkel2 (2005)

M. Cole, A. Benoit

Skeletons & guest languages

- skeletons are “design patterns” (and vice-versa)
  - not fully correct, but please enable me to use this approximation (Cole 2001)
- they can be realized in any language
  - implementations exist in C, C++, Java, Ocaml ...
  -

Skeletons in Pisa



Part III

- Grids
- Why Grids are really different from clusters
  - the need of QoS control
  - the need of adaptive programs
- ASSIST (University of Pisa)

What is the Grid

- *"... coordinated resource sharing and problem solving in dynamic, multi institutional virtual organizations."* (Foster, Anatomy of the Grid)

- *"1) coordinates resources that are not subject to centralized control …"*
  "2) … using standard, open, general-purpose protocols and interfaces"
  "3) … to deliver nontrivial qualities of service." (Foster, What is the Grid?)

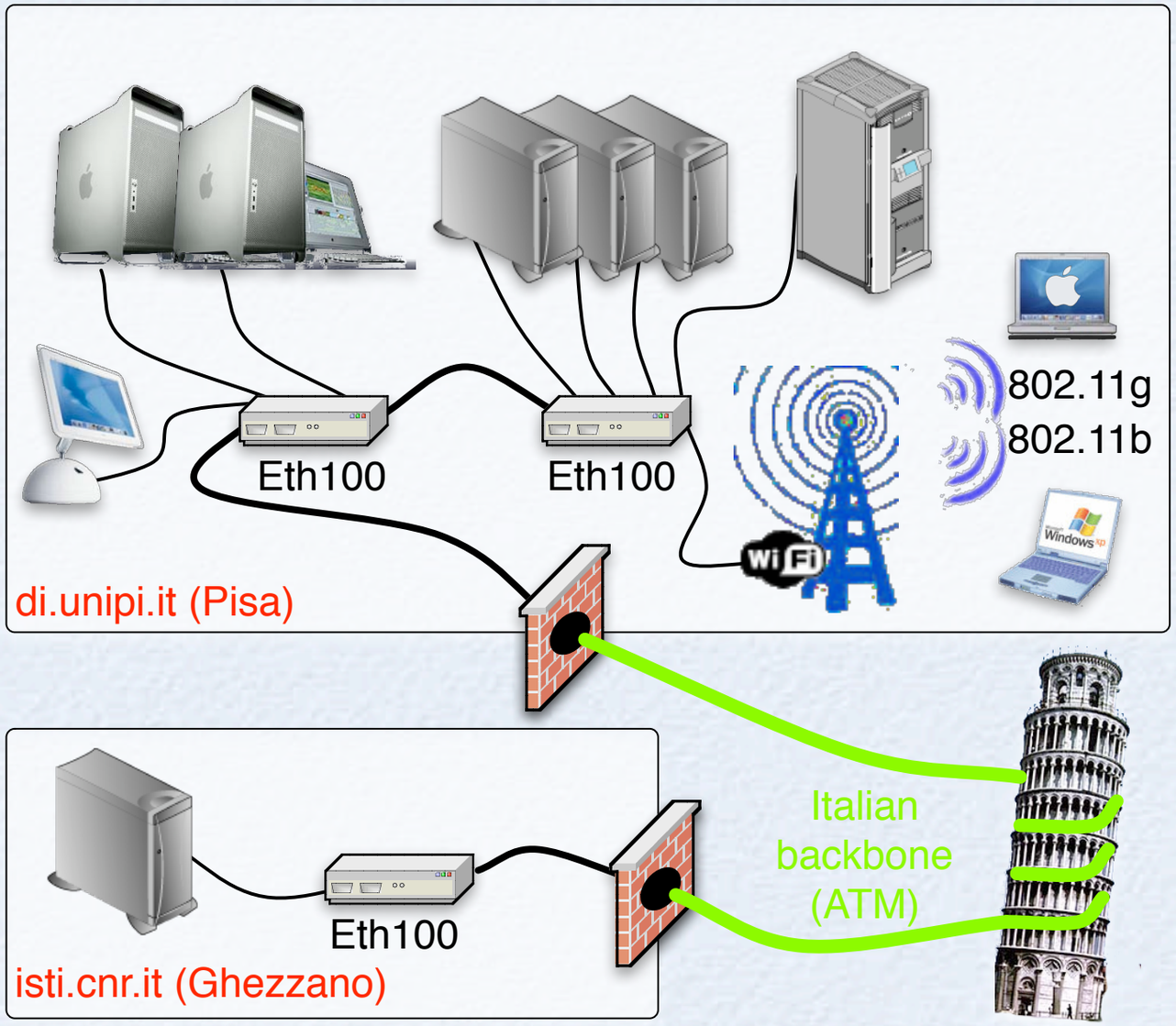
Grid features 1

- Heterogeneity:
  - machines are heterogeneous: different HW, OS, power ... networks are multi-tier, each tier is different (networks are heterogeneous as well).
  - protocols to guarantee interoperability (middleware, SOKU)
- Complexity
  - most interesting apps. are inherently distributed. Due to the scale is progressively more difficult to ensure good speedups, and correctness
  - no way to do it with low-level approaches. High-level tools needed.
- QoS
  - apps are required to exhibit a pre-determined QoS. In many interesting cases the QoS change along the run (e.g. catastrophes management)

Grid features 2

- Dynamicity:
  - platform, networks, and services become unavailable, change performances, fail-stop, ... and do it during the run. *And do it for sure, is it not a remote possibility (Gannon, Kennedy, Kesselman, Dongarra, ... GrADS@Rice Univ.)*
  - correctness as well as performance control become dynamic proprieties
  - the application should be ready to react to that, in other words it should be adaptive.
- No adaptivity means no Grid
  - this our idea (and also the idea of several partners of CoreGRID, Grids@Rice, ...)

May such HW be a Grid?

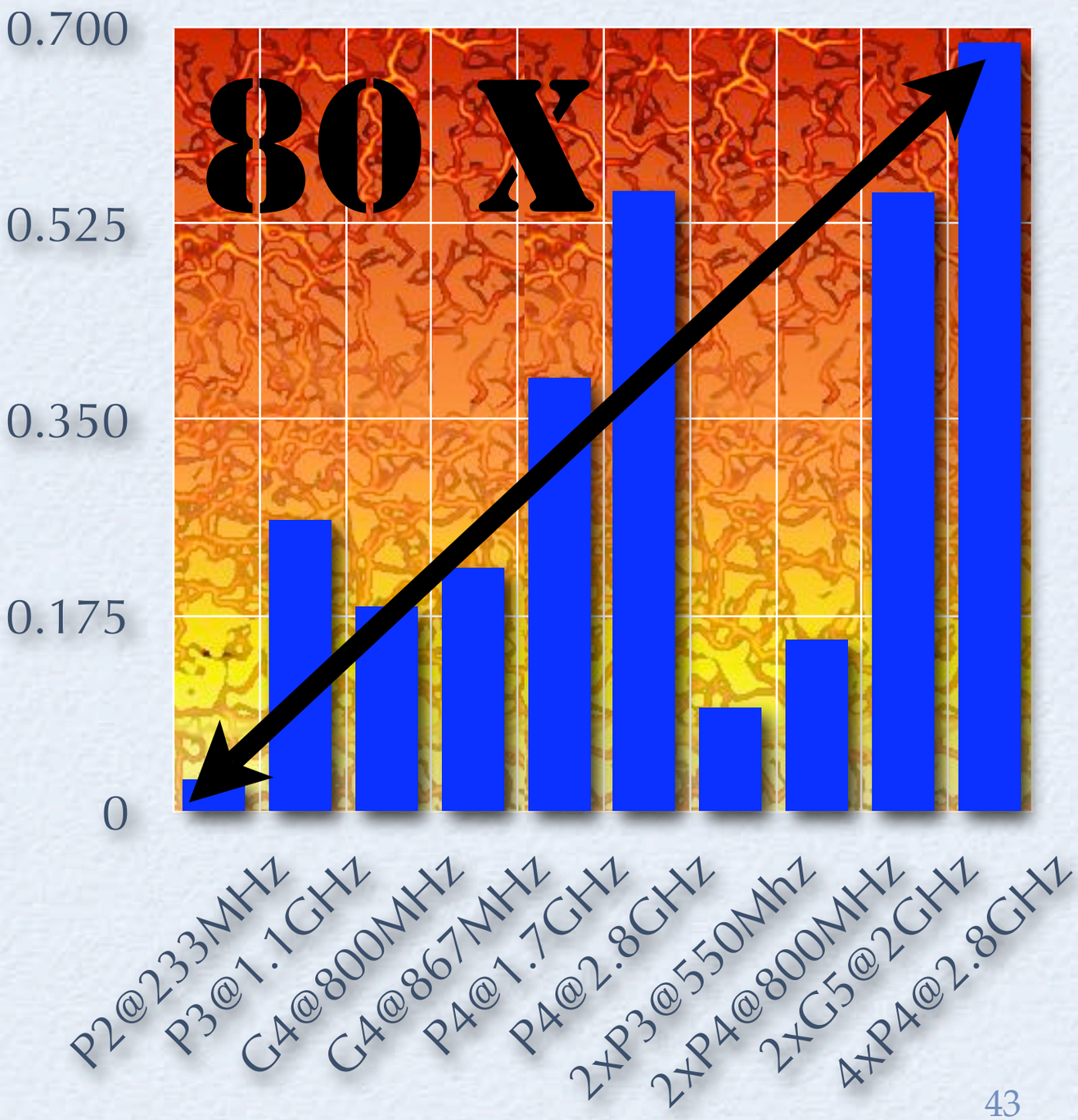


- Boxes have different powers
  - (46:1 max ratio)
- Net performance
  - Two Firewalls
  - ATM, Eth100, WiFi 11/54
- Operating Systems
  - Linux, MacOSX, Windows
- HW architecture
  - Single CPU and SMP
  - P2, P3, P4, HTP4, G4, G5

Many aspects rethought

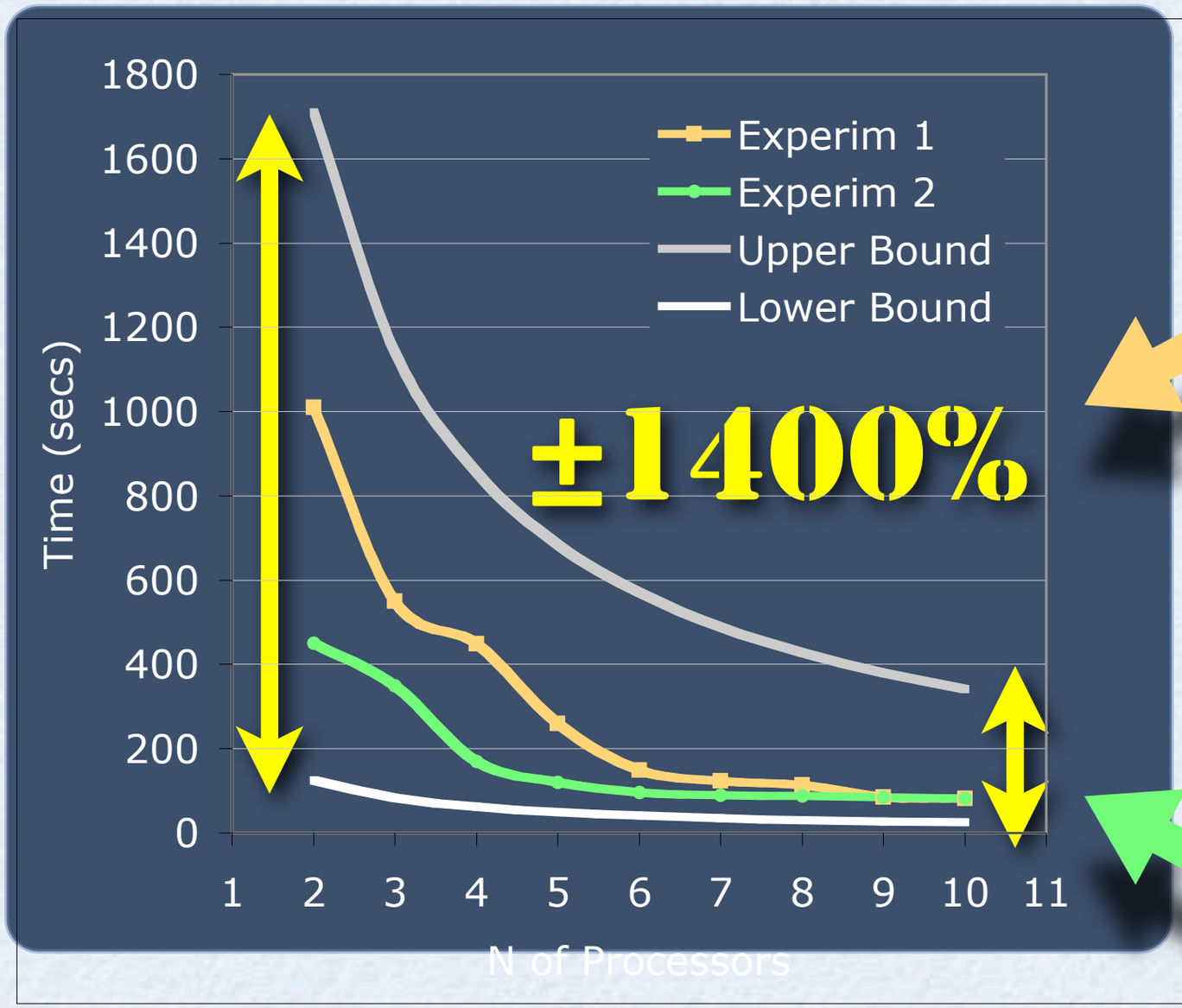
- Virtualization of resources
  - needed for adaptivity (Globus not enough)
    - ASSIST ➠ Virtual Process
    - ProActive ➠ Active Object
- Performance prediction
  - scheduling, mapping.... static/lunch time informations not reliable (look an example)

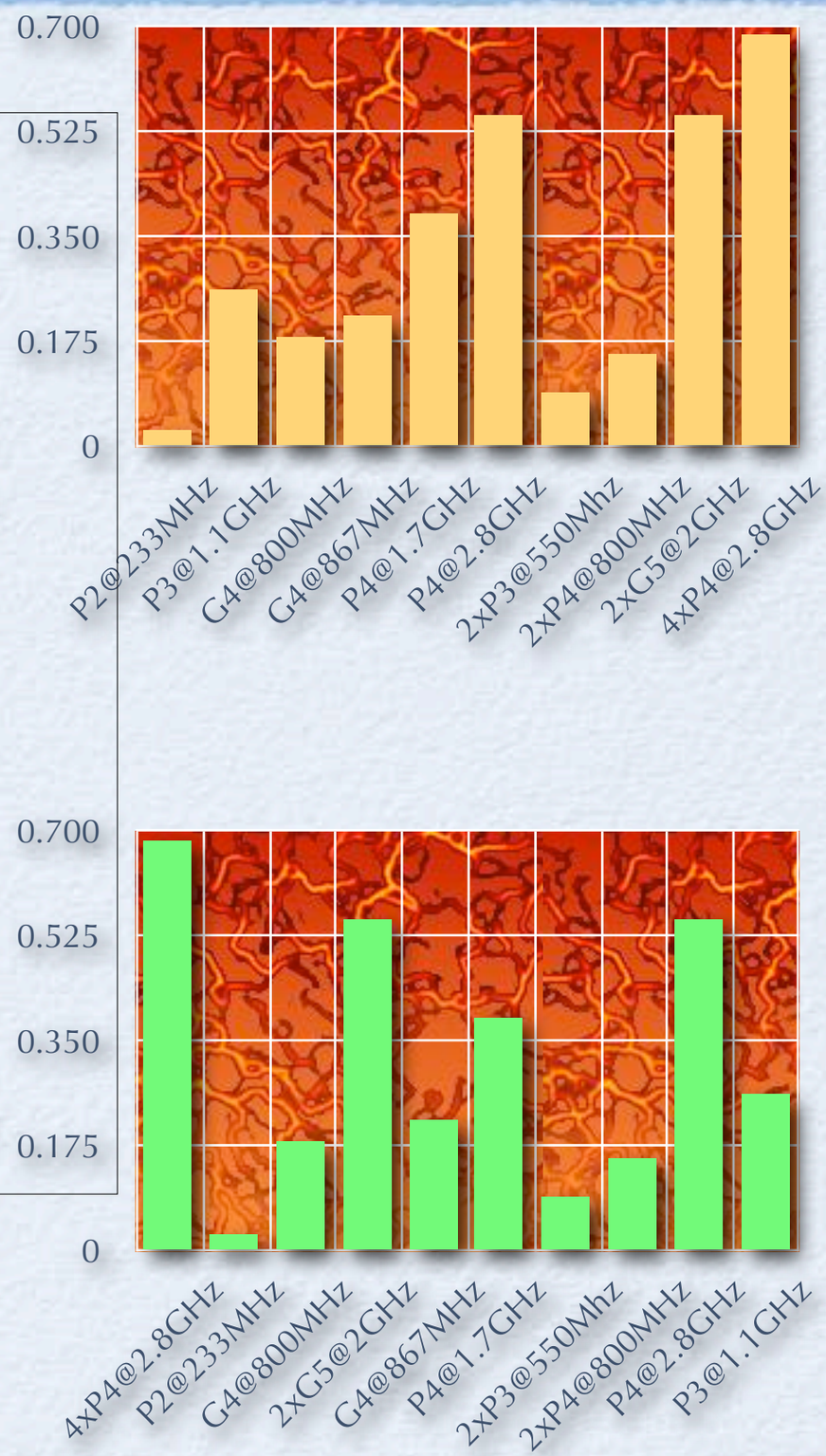
Boxes performances



- Grid platforms are supposed to exploit different "power" (in the meaning of Aristotelic power / act)
- and net bandwidth
- both of them may rapidly change over time

Performance metrics





Performance metrics

REJECTED

Outline

- Motivating ...
  - high-level programming for the grid
  - application adaptivity for the grid
- ASSIST basics & adaptivity in ASSIST
  - mechanisms
  - demo & some experiments
- Components & QoS
  - autonomic managers
  - QoS contracts
- Concluding remarks

// progr. & the grid

- concurrency exploitation, concurrent activities set up, mapping/scheduling, communication/synchronization handling and data allocation, ...
- manage resources heterogeneity and unreliability, networks latency and bandwidth unsteadiness, resources topology and availability changes, firewalls, private networks, reservation and jobs schedulers, ...

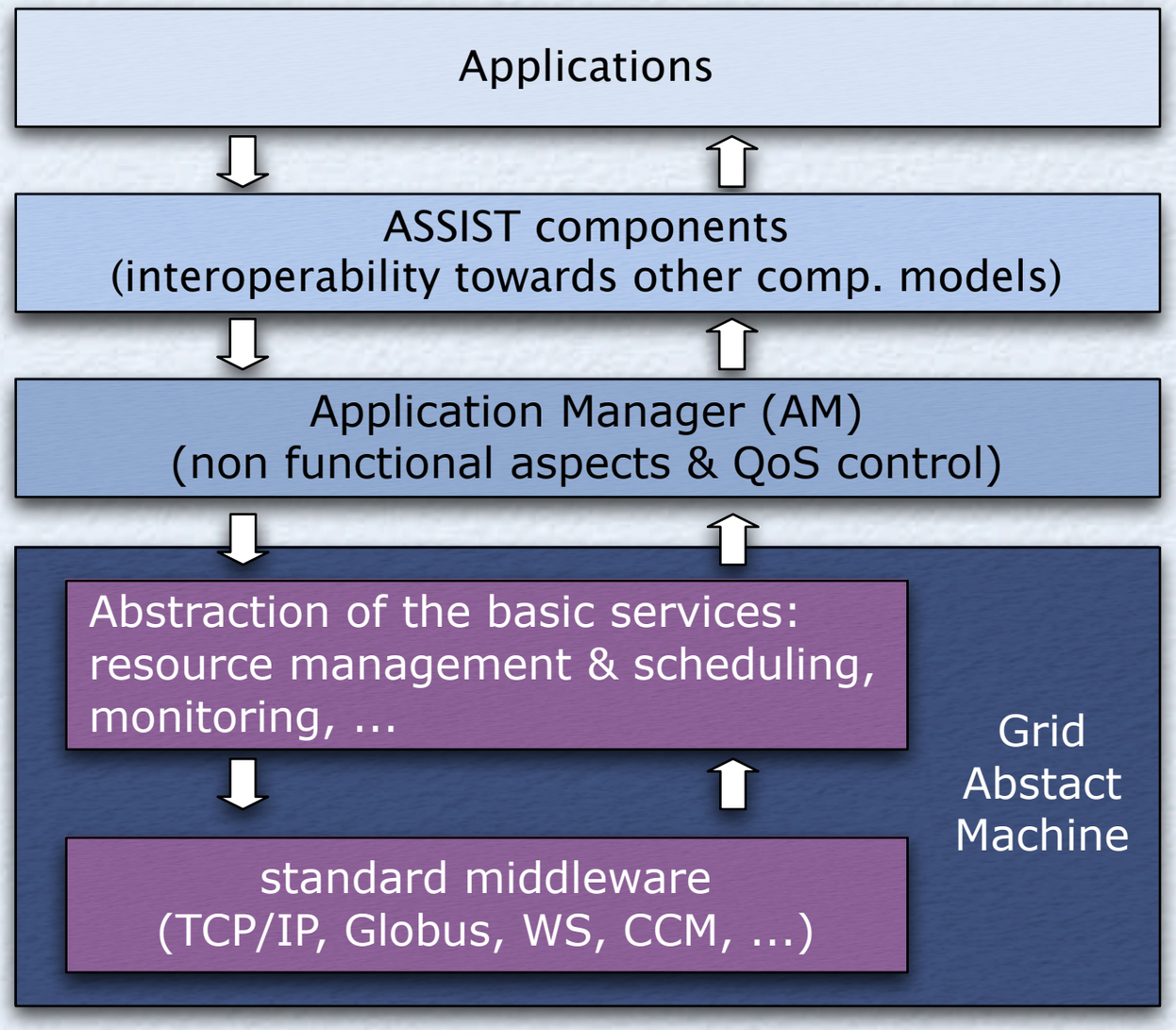
... and a non trivial QoS for **applications**

not easy leveraging only on middleware

D. Gannon et al. opened the way (GrADS@Rice)

ASSIST idea

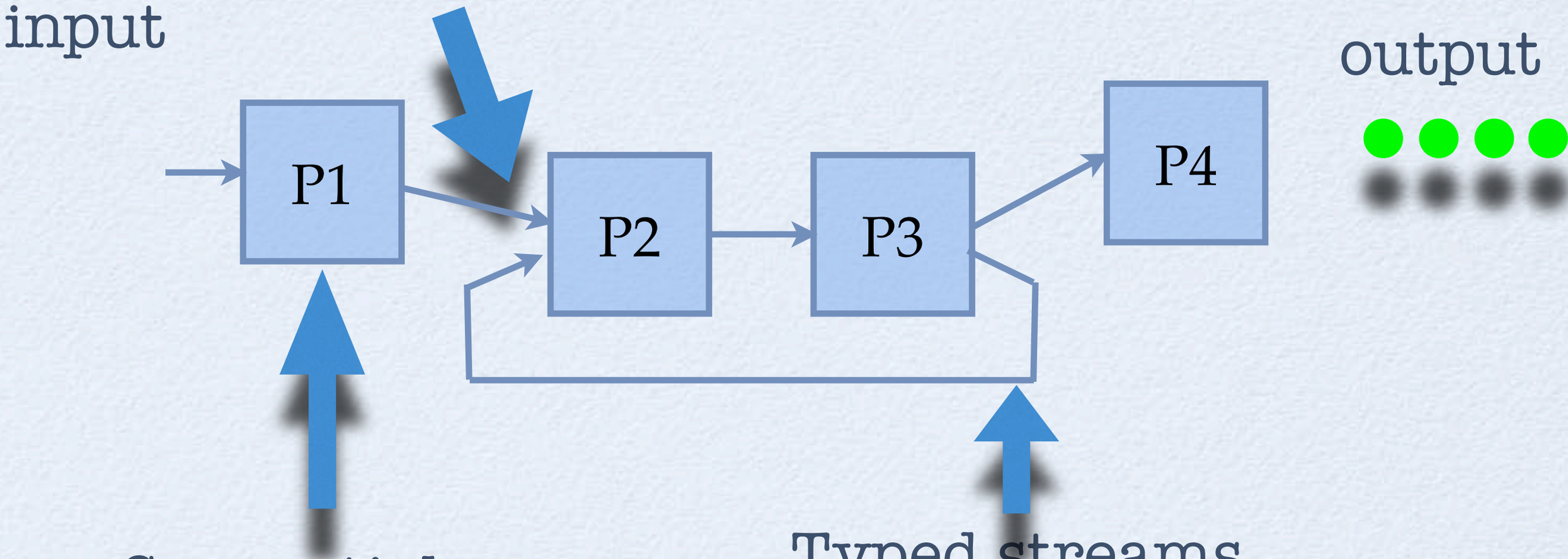
"moving most of the Grid specific efforts needed while developing high-performance Grid applications from programmers to grid tools and run-time systems"



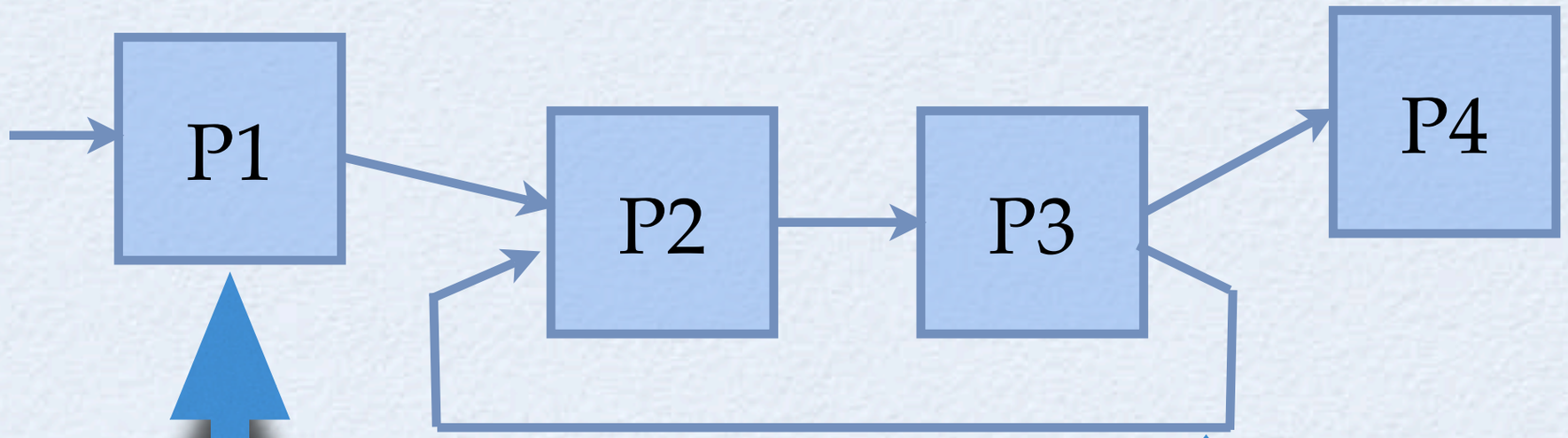
ASSIST is a high-level programming environment for grid-aware // applications.
Developed at Uni. Pisa within several national & EU projects.
First version in 2001. Open source under GPL.

app = graph of modules

Programmable, possibly nondeterministic input behaviour



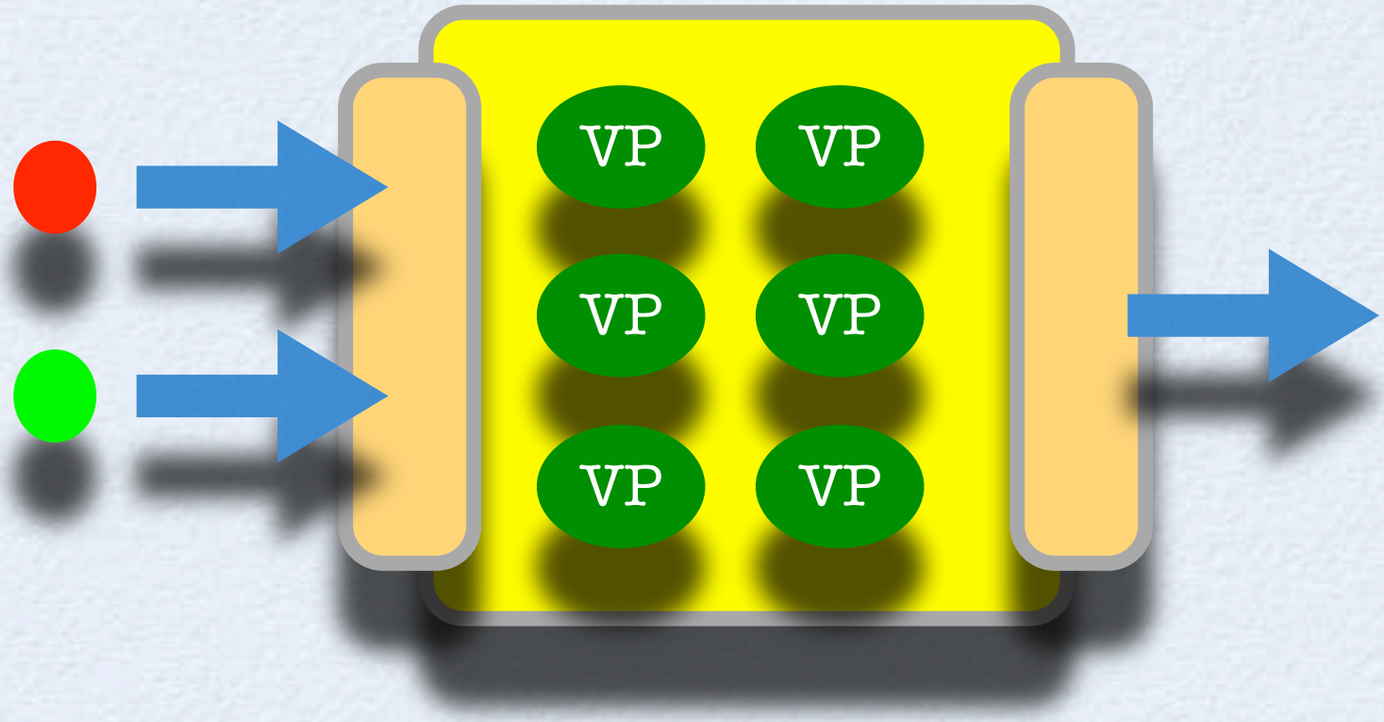
native + standards



ASSIST native or wrap
(MPI, CORBA, CCM, WS)

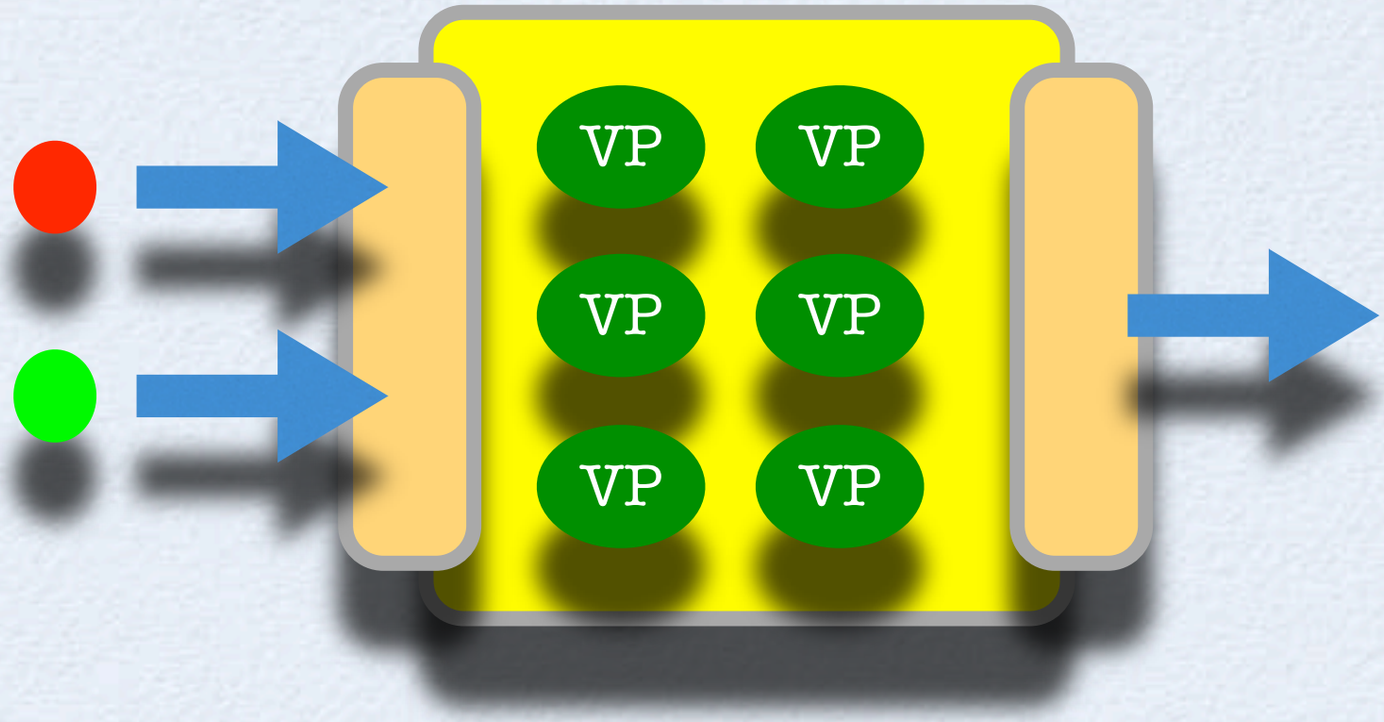
TCP/IP, Globus,
IIOP CORBA,
HTTP/SOAP

ASSIST parmod



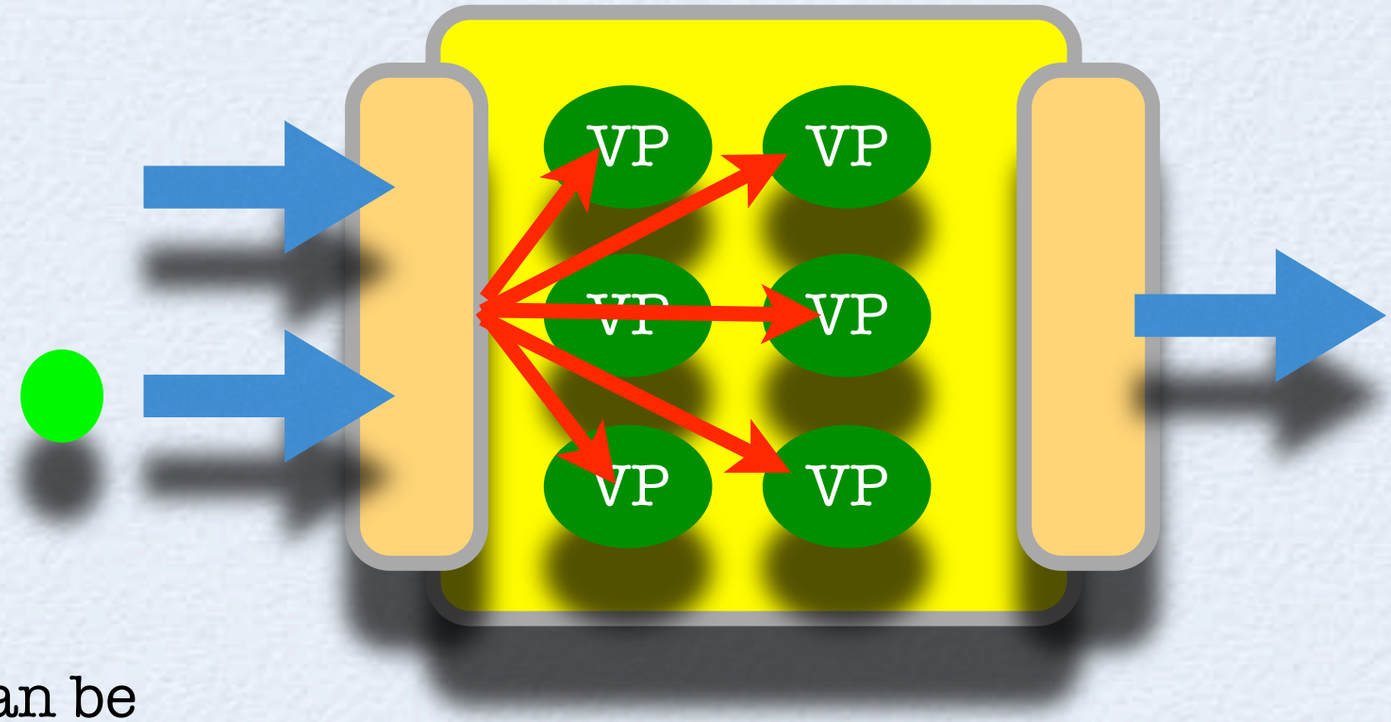
ASSIST parmod

An “input section” can be programmed in a CSP-like way



ASSIST parmod

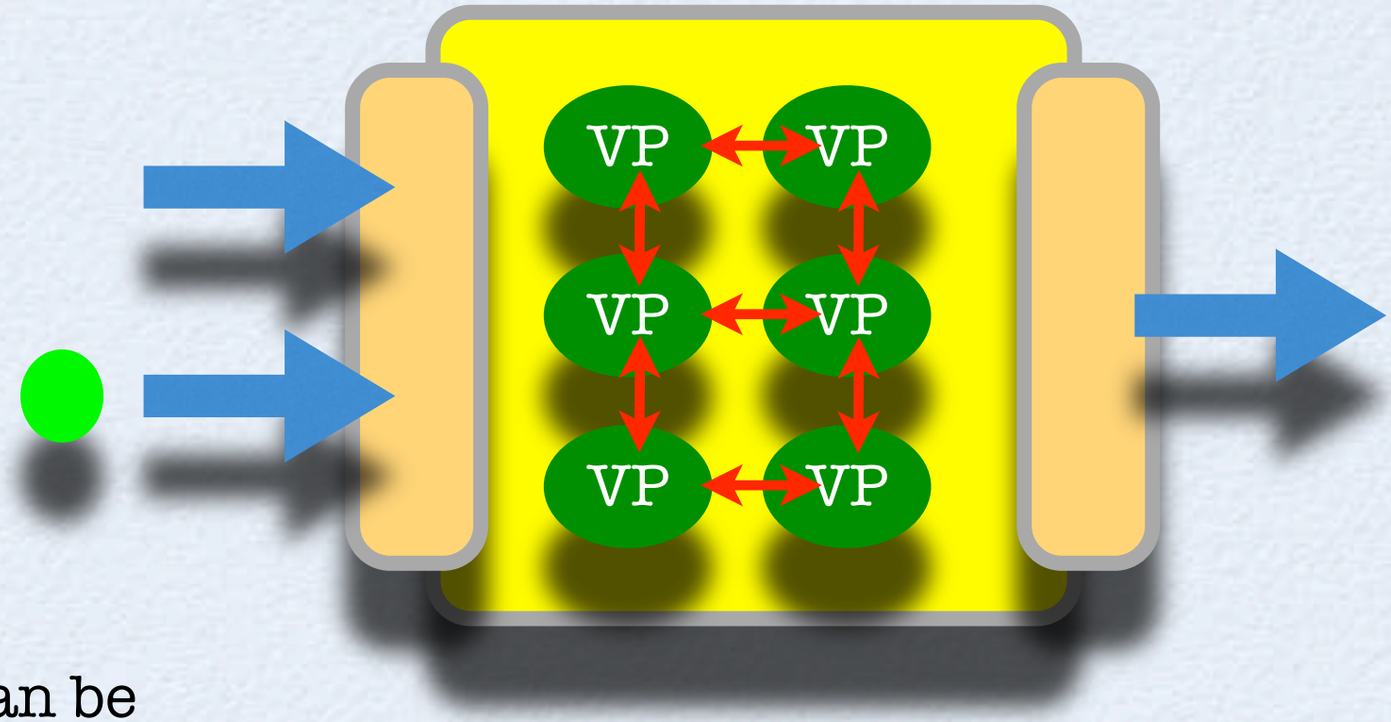
An “input section” can be programmed in a CSP-like way



Data items can be distributed (scattered, broadcasted, multicasted) to a set of **Virtual Processes** which are named accordingly to a topology

ASSIST parmod

An “input section” can be programmed in a CSP-like way



Data items can be distributed (scattered, broadcasted, multicasted) to a set of **Virtual Processes** which are named accordingly to a topology

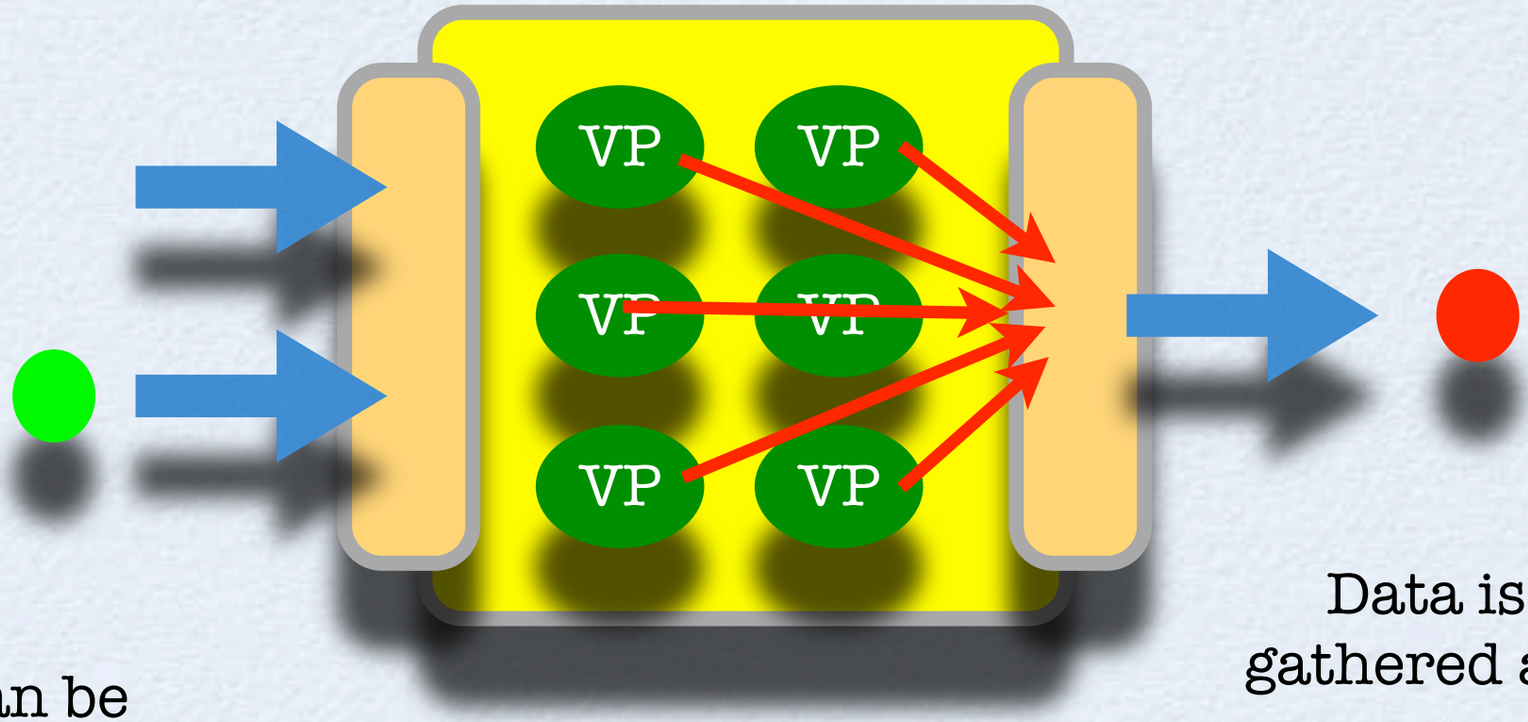
Data items partitions are elaborated by VPs, possibly in iterative way

```
while(...)
   forall VP(in, out)
   barrier
```

data is logically shared by VPs (owner-computes)

ASSIST parmod

An “input section” can be programmed in a CSP-like way



Data items can be distributed (scattered, broadcasted, multicasted) to a set of **Virtual Processes** which are named accordingly to a topology

Data items partitions are elaborated by VPs, possibly in iterative way

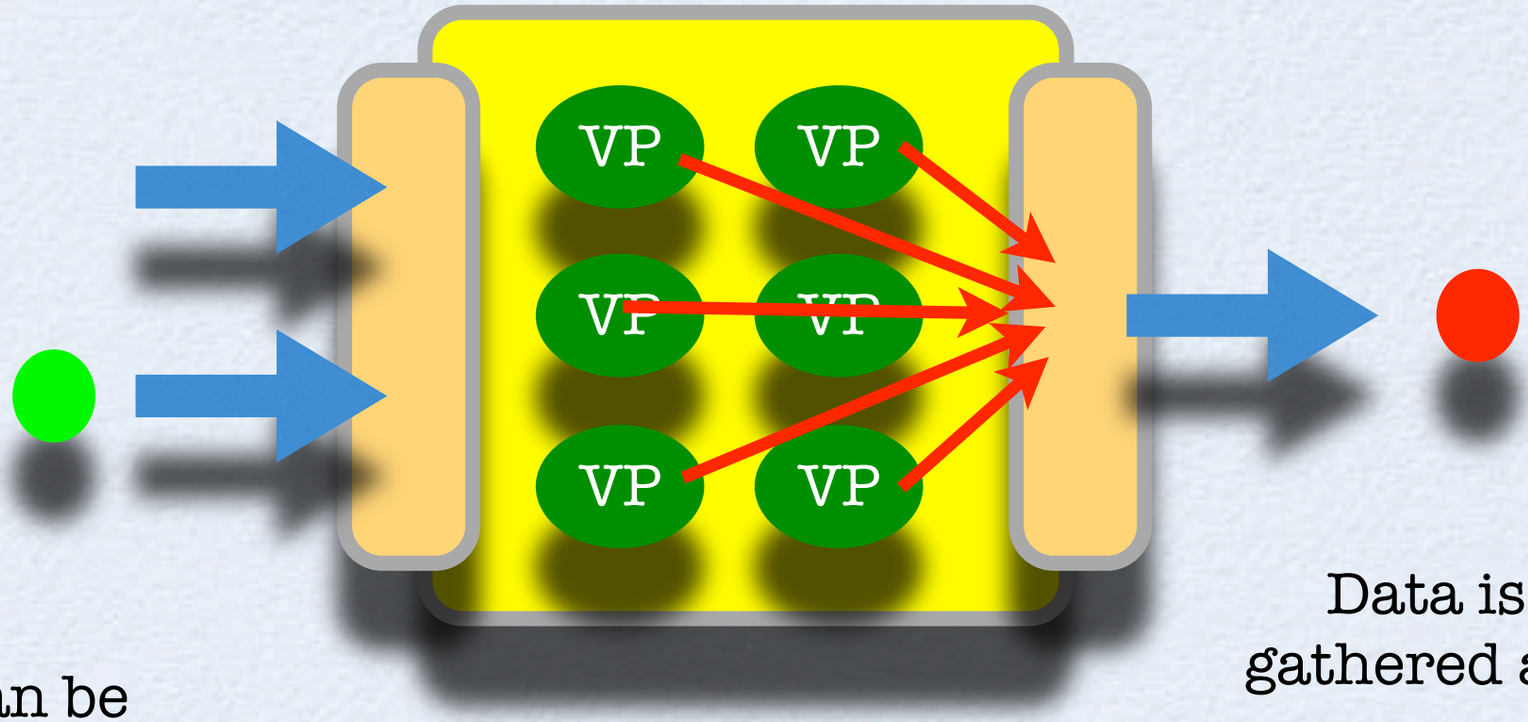
```
while(...)
   forall VP(in, out)
   barrier
```

data is logically shared by VPs (owner-computes)

Data is eventually gathered accordingly to an user defined way

ASSIST parmod

An “input section” can be programmed in a CSP-like way



Data items can be distributed (scattered, broadcasted, multicasted) to a set of **Virtual Processes** which are named accordingly to a topology

Data items partitions are elaborated by VPs, possibly in iterative way

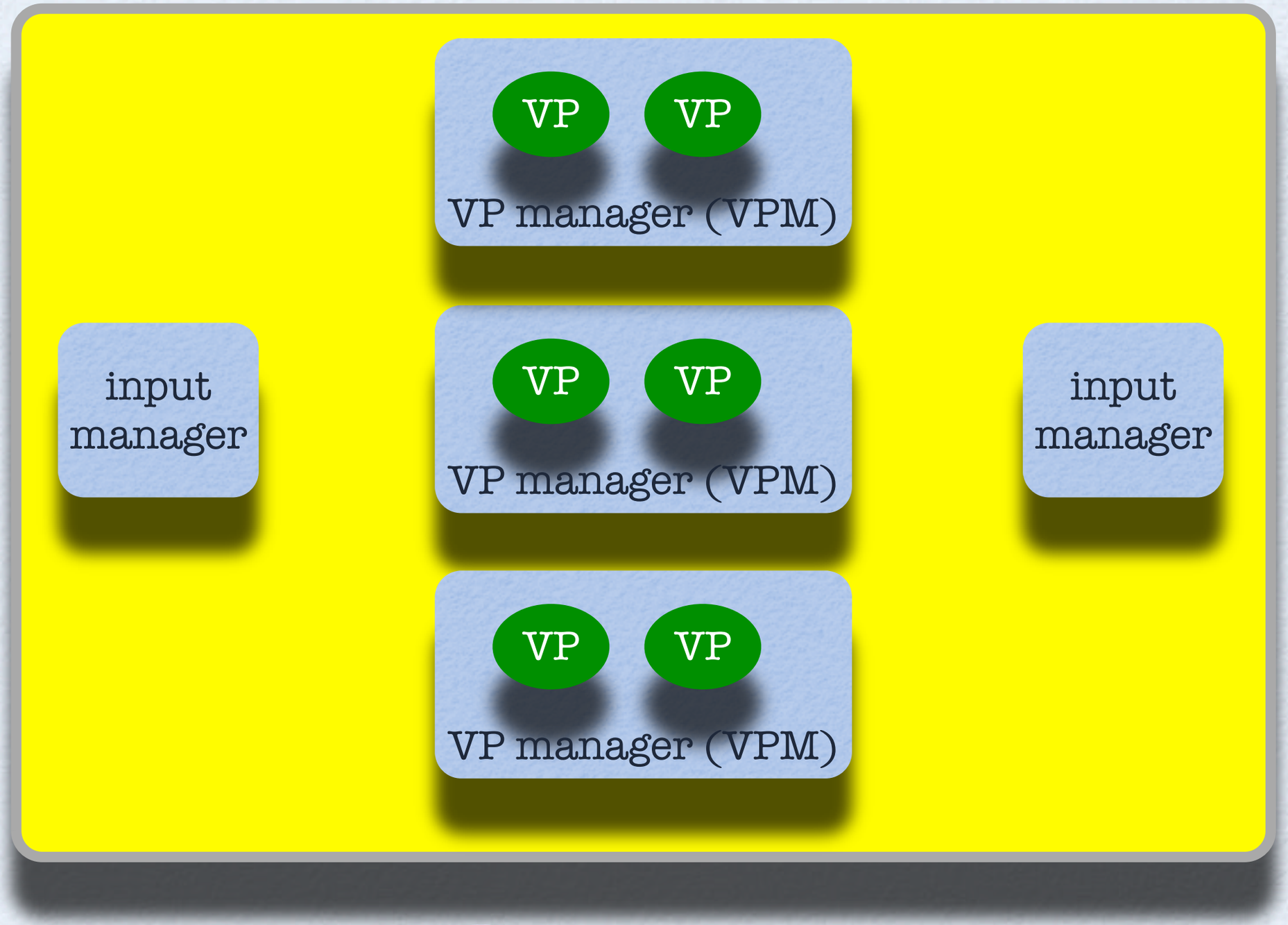
```
while(...)
   forall VP(in, out)
   barrier
```

data is logically shared by VPs (owner-computes)

Data is eventually gathered accordingly to an user defined way

Easy to express standard paradigms (skeltons), such as **farm, deal, haloswap, map, apply-to-all, forall**, ...

parmod implementation



processes VP Virtual Processes

Compiling & running

QoS contract

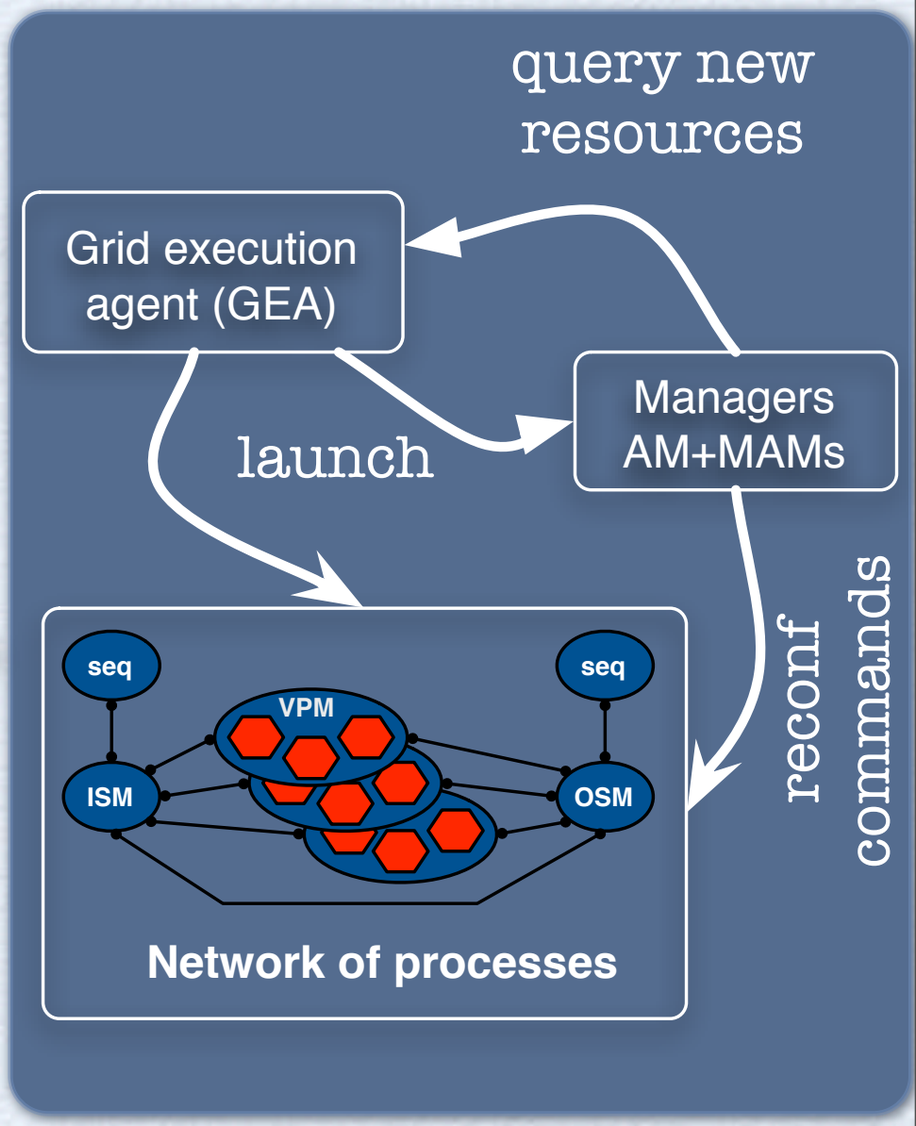
ASSIST compiler

resource description XML

Run

ASSIST program

executable code (linux, mac, M$win)



Application adaptivity

- Adaptivity aims to dynamically **control** program configuration (e.g. parallel degree) and mapping
  - for performance (high-performance is a natural sub-target)
  - for fault-tolerance (enable to cope with unsteadiness of resources, and some kind of faults)

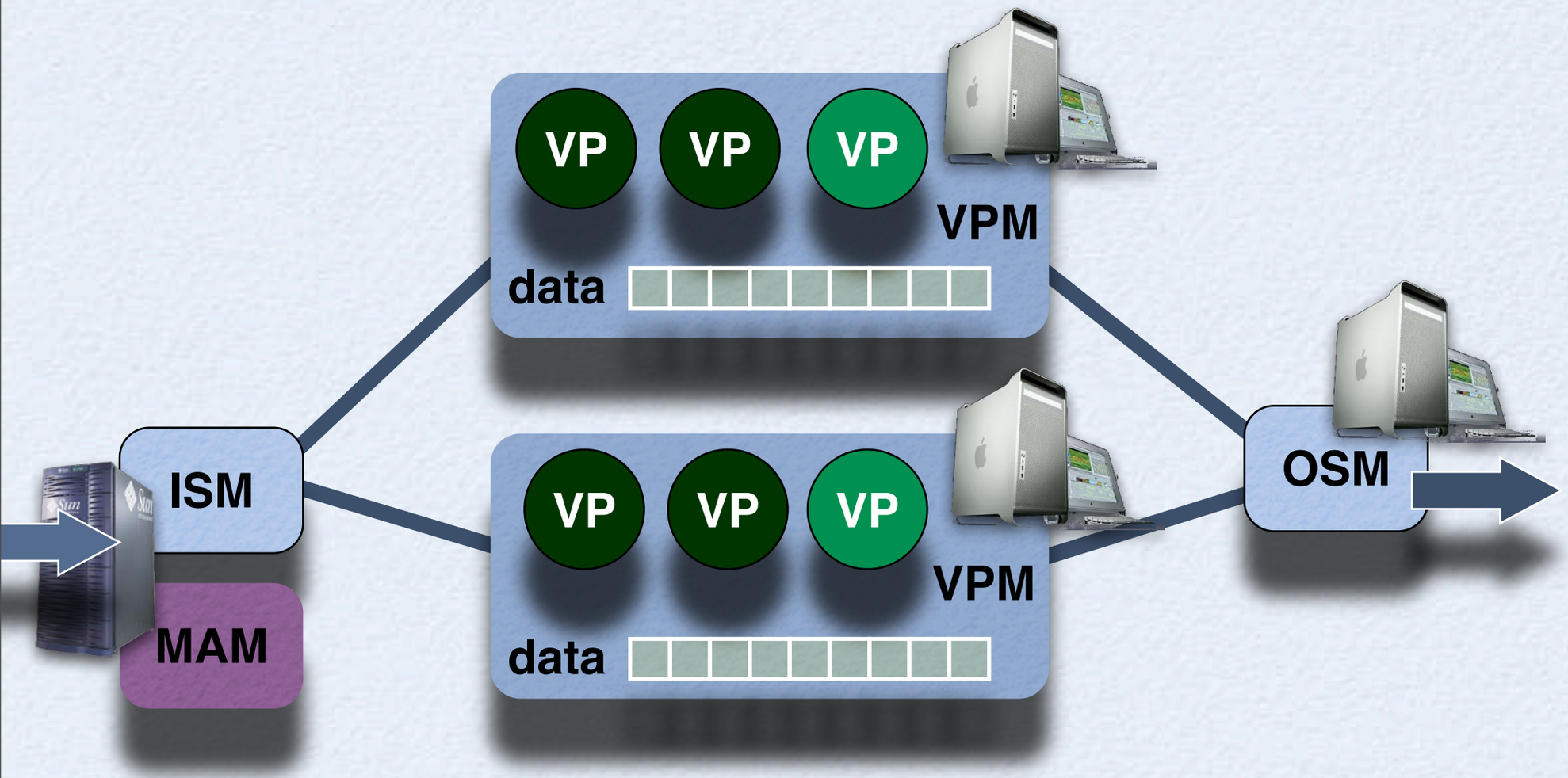
Adaptivity recipe (ingredients)

1. Mechanism for adaptivity
   - reconf-safe points
     - in which points a parallel code can be safely reconfigured?
   - reconf-safe point consensus
     - different parallel activities may not proceed in lock-step fashion
   - add/remove/migrate computation & data
2. Managing adaptivity
   - QoS contracts
     - Describing high-level QoS requirement for modules/applications
   - "self-optimizing" modules/components
     - under the control of an autonomic manager

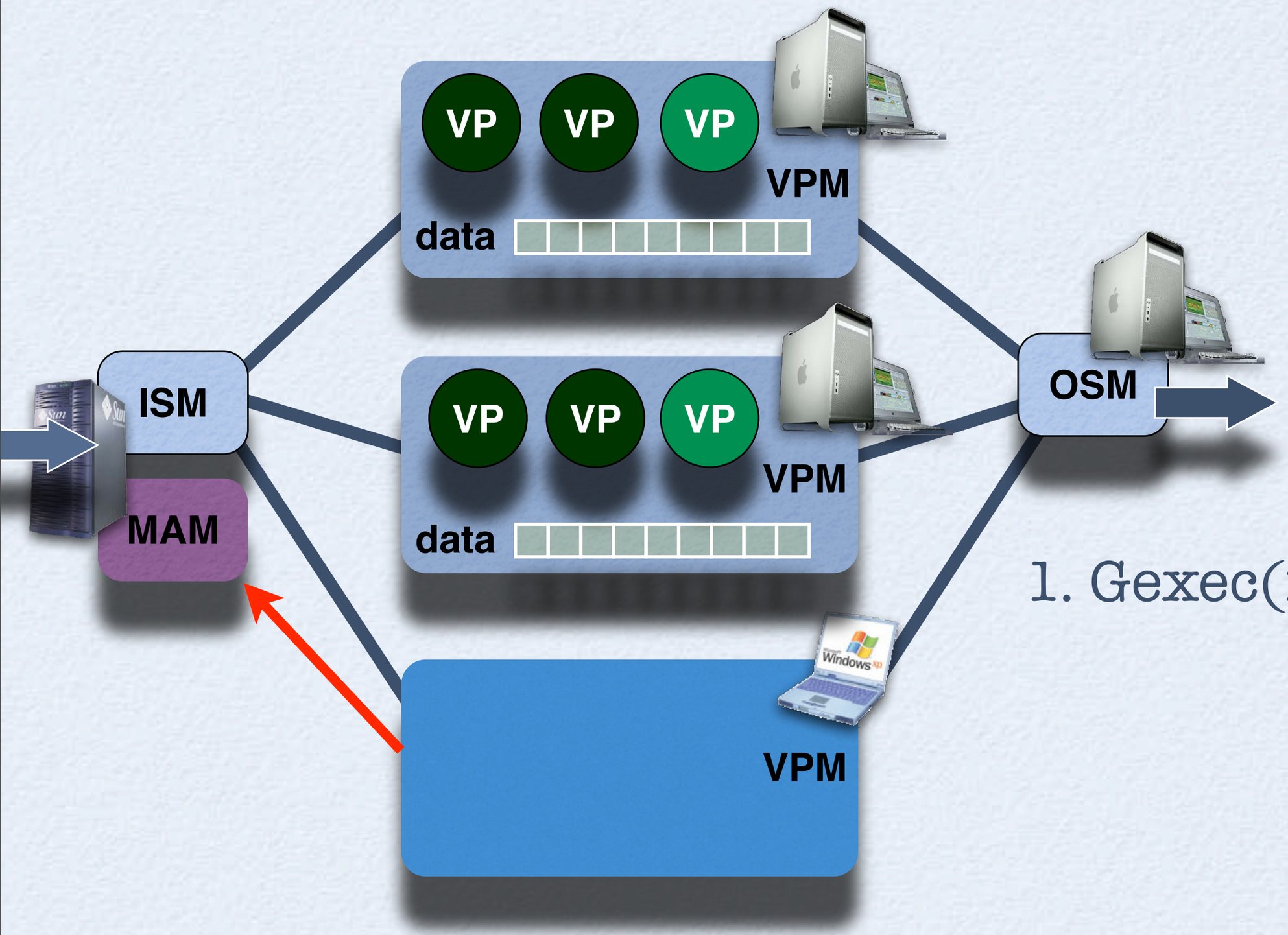
Mechanisms

- At parmod level
  - add/remove/migrate VPs
  - very low-overhead due to knowledge coming from high-level semantics + suitable compiling tools
- At component level
  - create/destroy/wire/unwire parallel entities
  - medium/large overhead due to underlying API for staging, run, ...
- Not addressed in this talk (see references in the paper: Europar 05, ParCo 05, ...), I just show a short demo

adaptivity: a working ex.

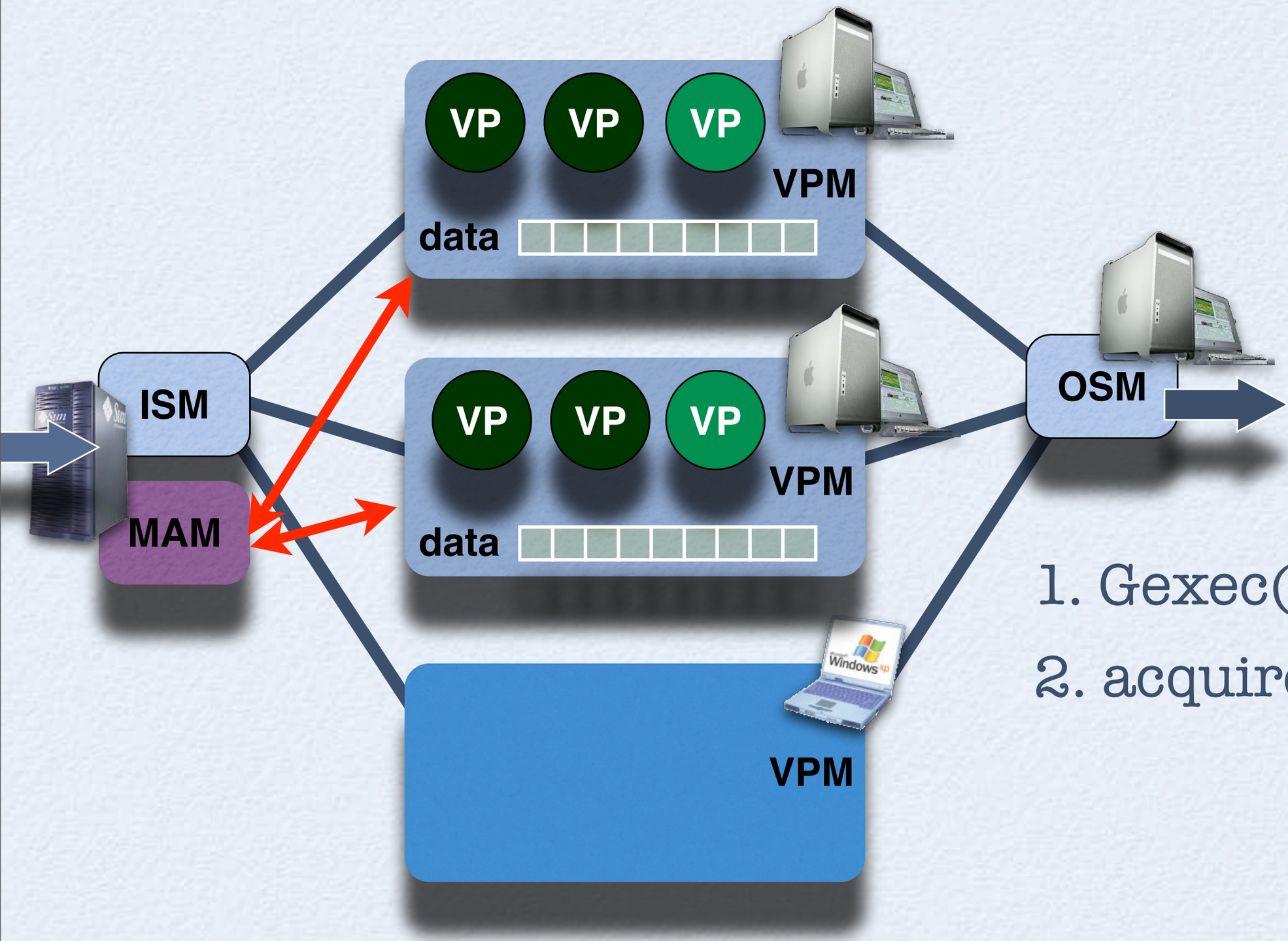


adaptivity: a working ex.



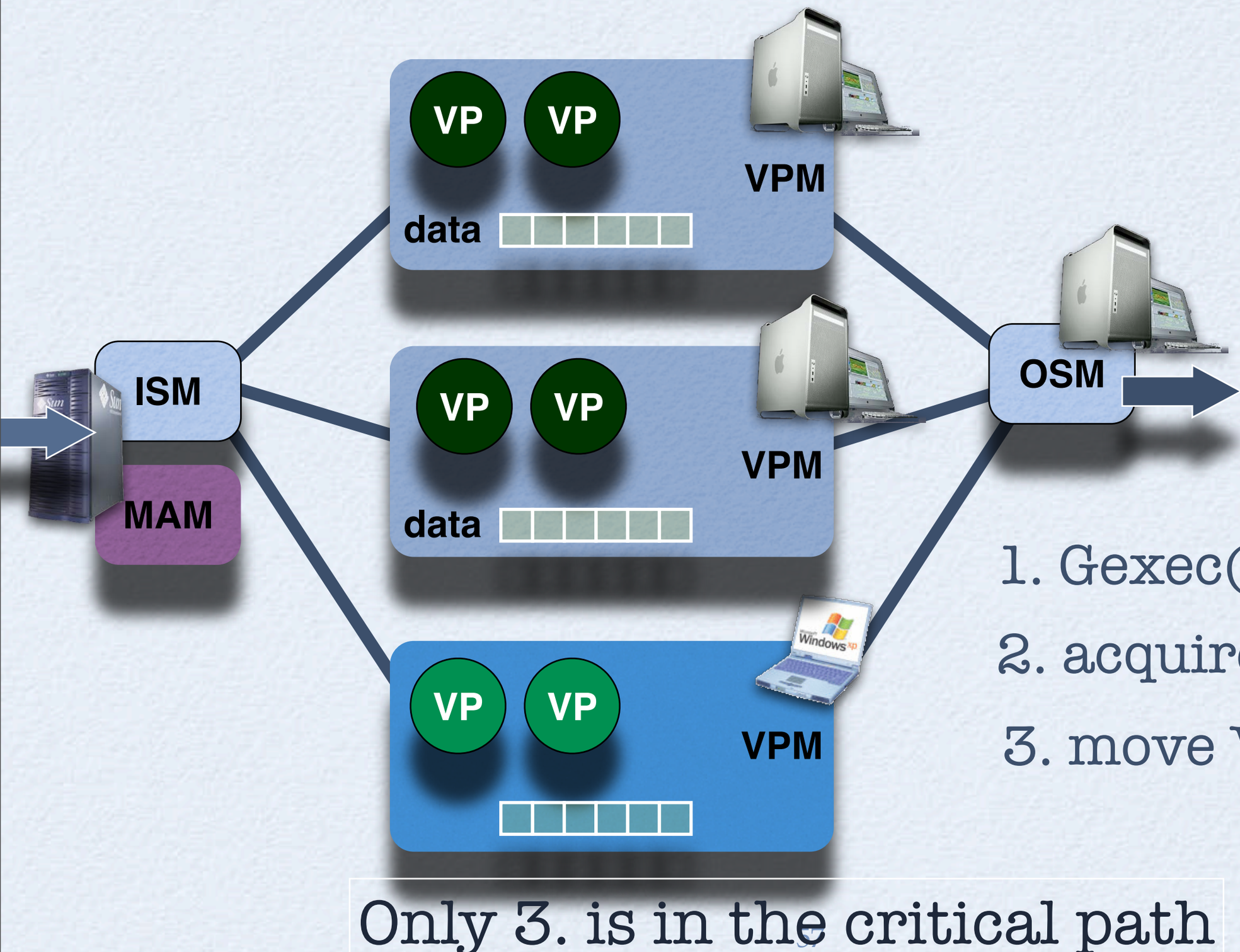
1. Gexec(newPE, VPM)

adaptivity: a working ex.



1. Gexec(newPE, VPM)
2. acquire consensus

adaptivity: a working ex.



1. Gexec(newPE, VPM)
2. acquire consensus
3. move VP and data

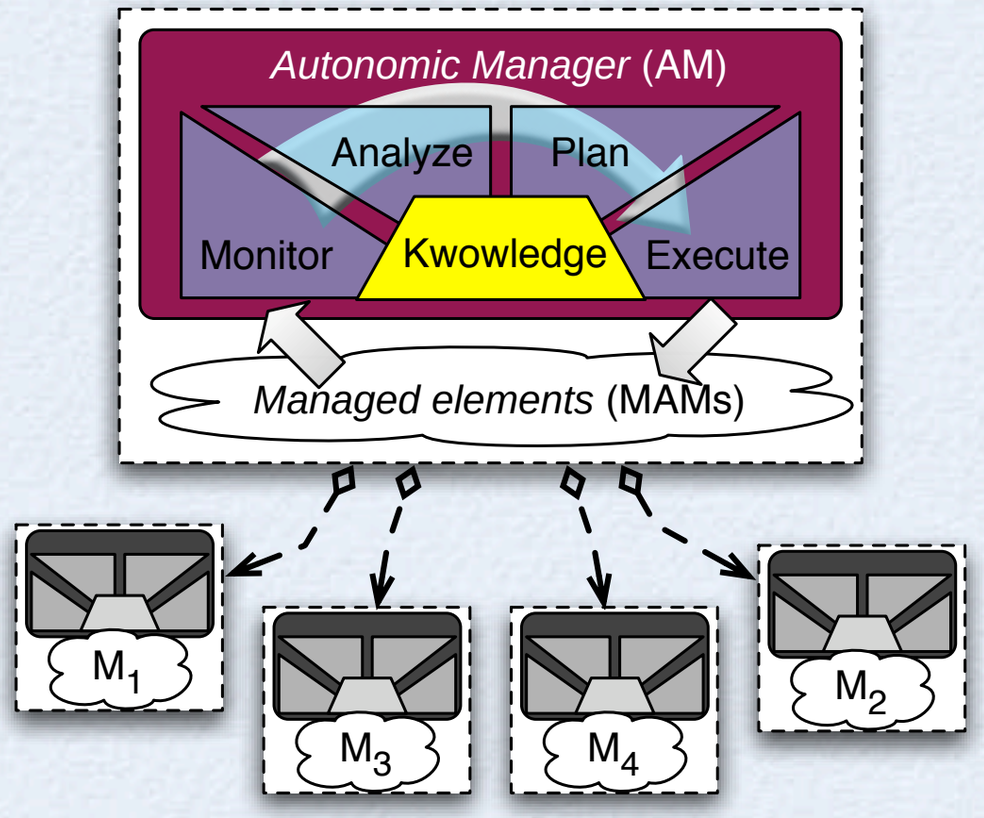
Only 3. is in the critical path

overhead? (mSecs)

parmod kind	Data-parallel (with shared state)						Farm (without shared state)					
reconf. kind	add PEs			remove PEs			add PEs			remove PEs		
# of PEs involved	1→2	2→4	4→8	2→1	4→2	8→4	1→2	2→4	4→8	2→1	4→2	8→4
R_l on-barrier	1.2	1.6	2.3	0.8	1.4	3.7	–	–	–	–	–	–
R_l on-stream-item	4.7	12.0	33.9	3.9	6.5	19.1	~ 0	~ 0	~ 0	~ 0	~ 0	~ 0
R_t	24.4	30.5	36.6	21.2	35.3	43.5	24.0	32.7	48.6	17.1	21.6	31.9

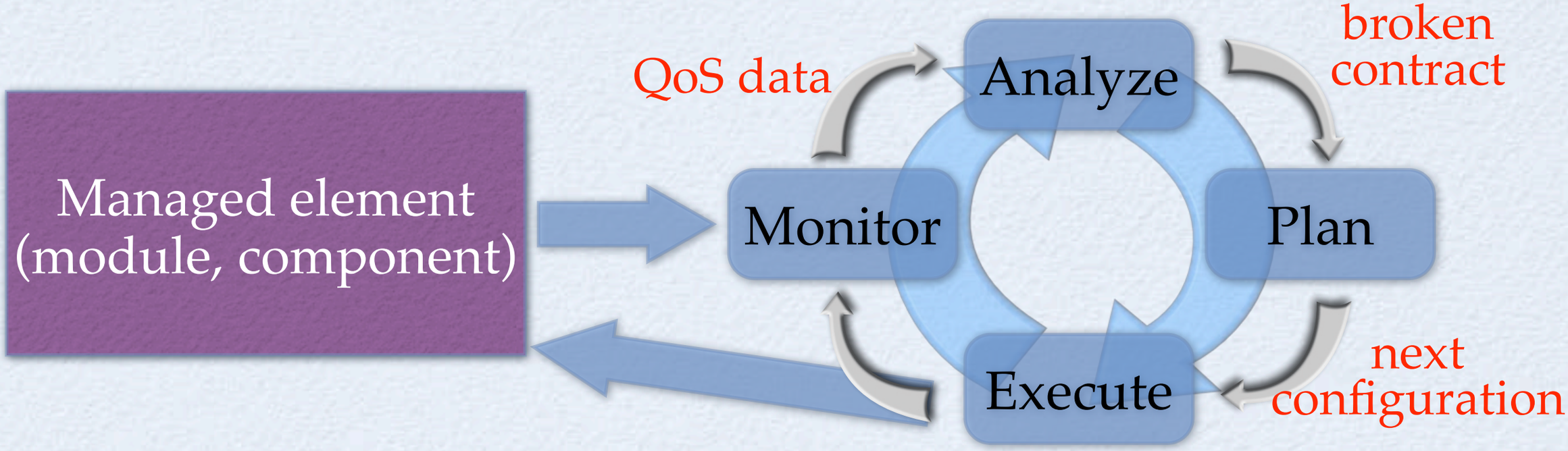
GrADS papers reports overhead in the order of hundreds of seconds (K. Kennedy et al. 2004), this is mainly due to the stop/restart behavior, not to the different running env.

Autonomic Computing



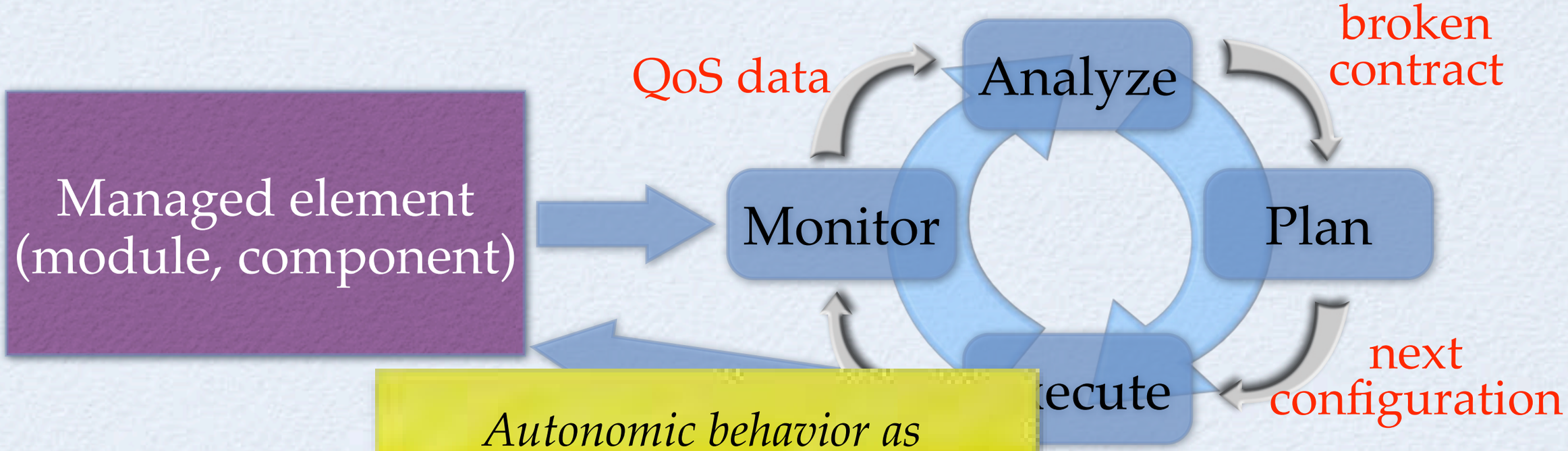
- AC emblematic of a vast hierarchy of self-governing systems, many of which consist of many interacting, self-governing components that in turn comprise a number of interacting, self-governing components at the next level down.
- IBM "invented" it in 2001 (control with self-awareness, from human body autonomic nervous system)
  - self-optimization, self-healing, self-protection, self-configuration = self-management
- control loop, of course, exists from mid of last century

Autonomic behavior



- monitor: collect execution stats: machine load, VPM service time, input/output queues lenghts, ...
- analyze: instanciate performance models with monitored data, detect broken contract, in and in the case try to indivituate the problem
- plan: select a (predefined or user defined) strategy to reconvey the contract to valid status. The strategy is actually a list of mechanism to apply.
- execute: leverage on mechanism to apply the plan

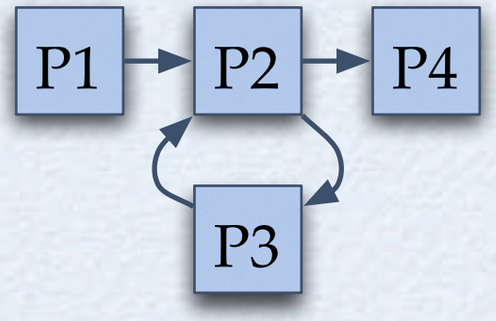
Autonomic behavior



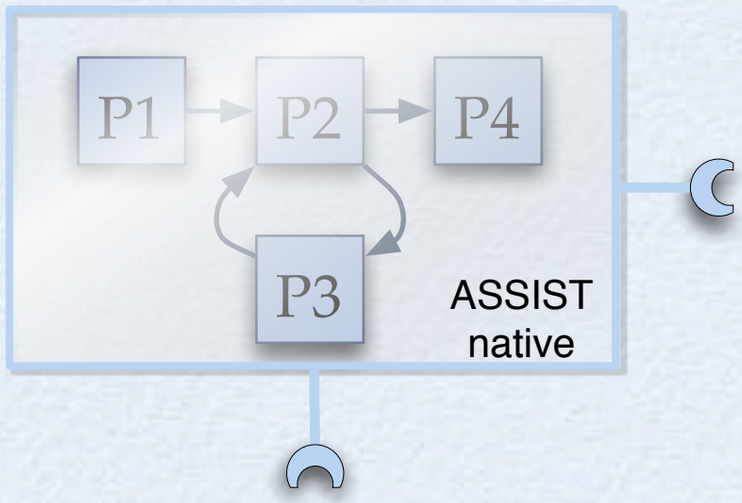
Autonomic behavior as been included in NGG2/3 (Next Generation Grid) EU founding recommendation as prerequisite for Grid computing

- **monitor**: collect service time, input/output queues lenghts,
- **analyze**: instanci ored data, detect broken contract, in and i lem
- **plan**: select a (pre reconvey the contract to valid status. The strategy pply.
- **execute**: leverage on mechanism to apply the plan

ASSIST & components

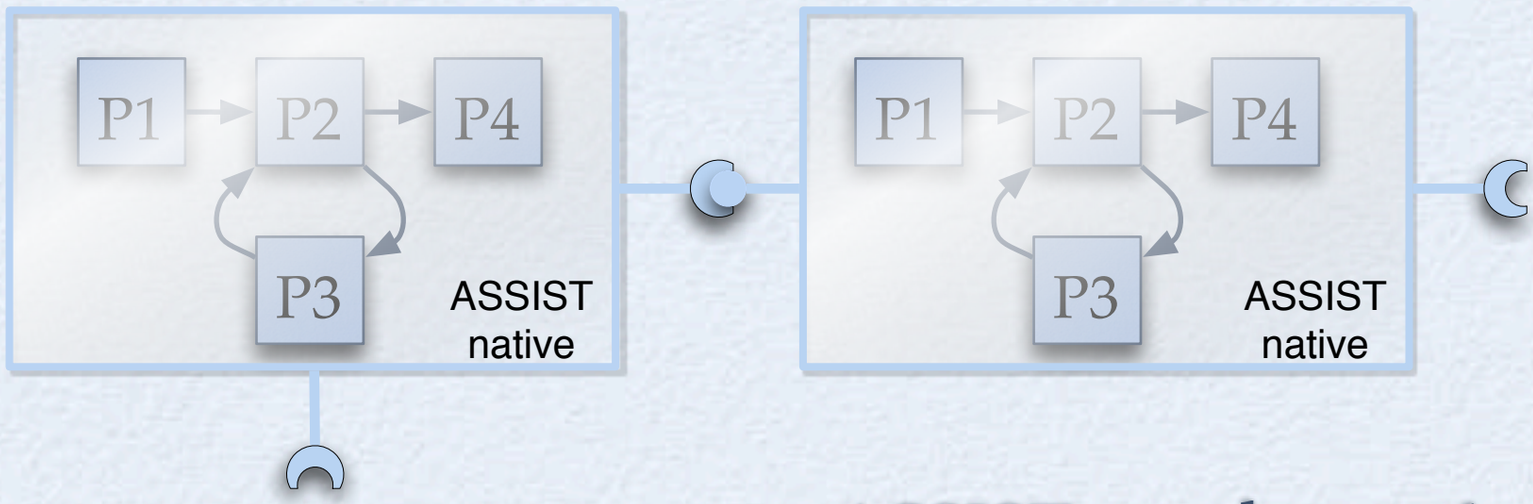


ASSIST & components



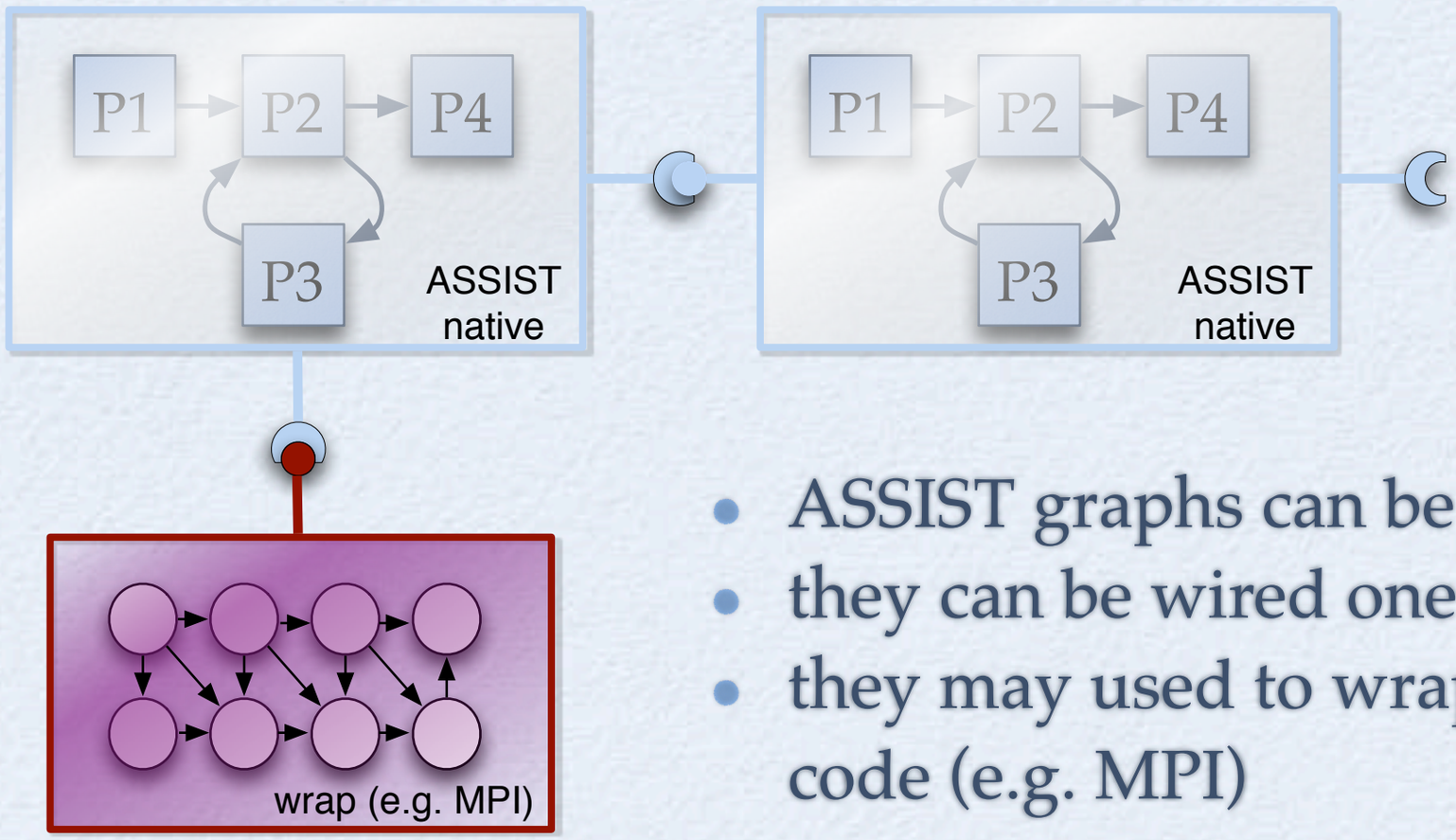
- ASSIST graphs can be enclosed in components

ASSIST & components



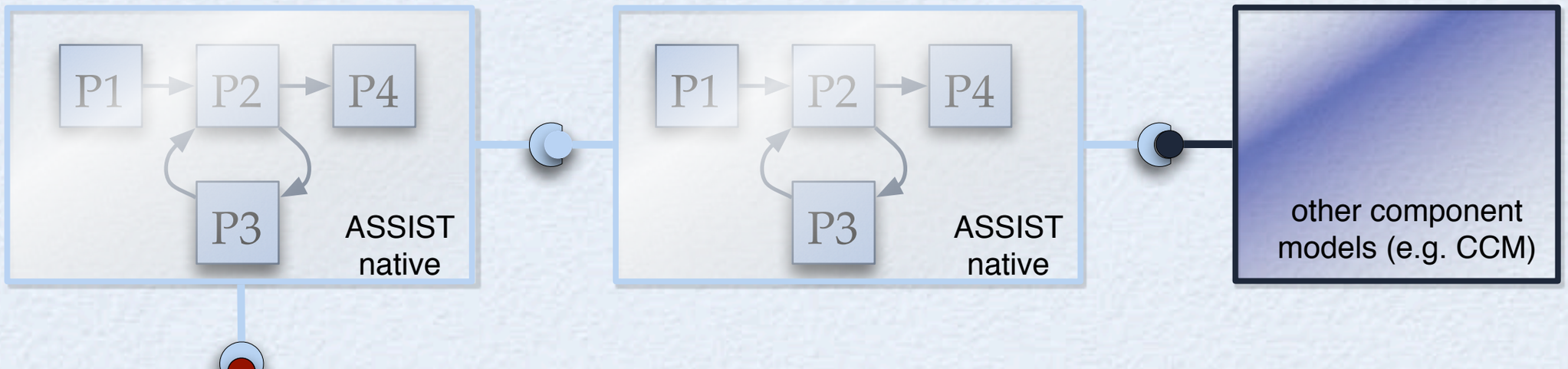
- ASSIST graphs can be enclosed in components
- they can be wired one another

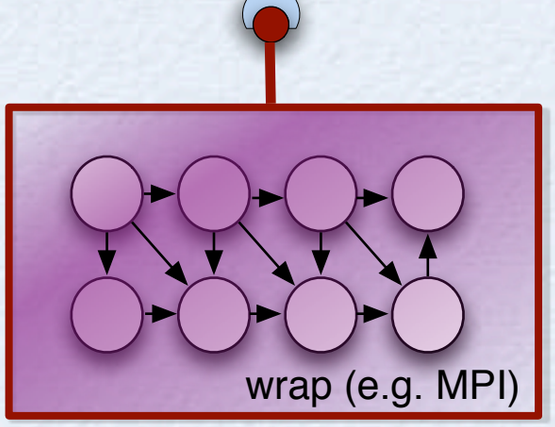
ASSIST & components



- ASSIST graphs can be enclosed in components
- they can be wired one another
- they may used to wrap sequential or parallel code (e.g. MPI)

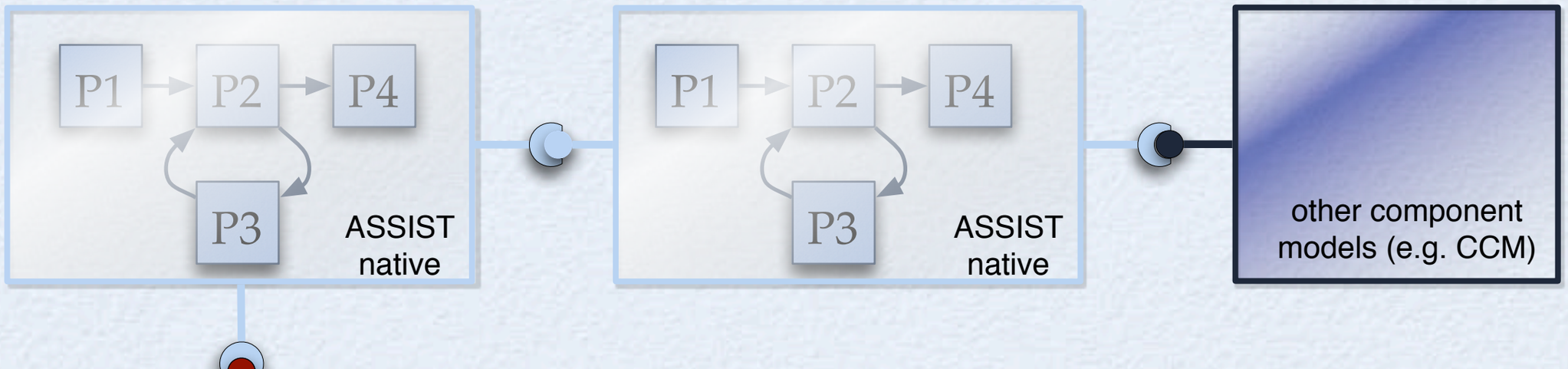
ASSIST & components

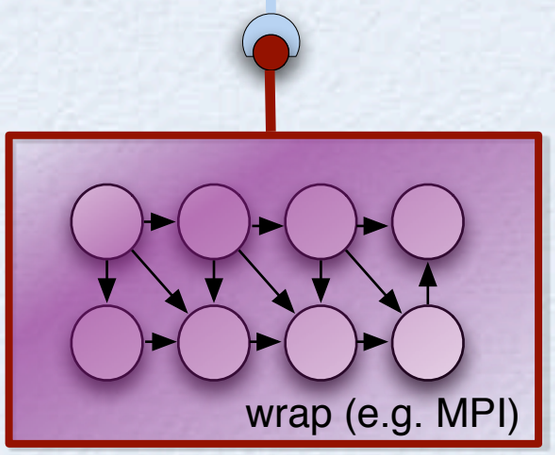




- ASSIST graphs can be enclosed in components
- they can be wired one another
- they may used to wrap sequential or parallel code (e.g. MPI)
- they can be wired to other legacy components (e.g. CCM)

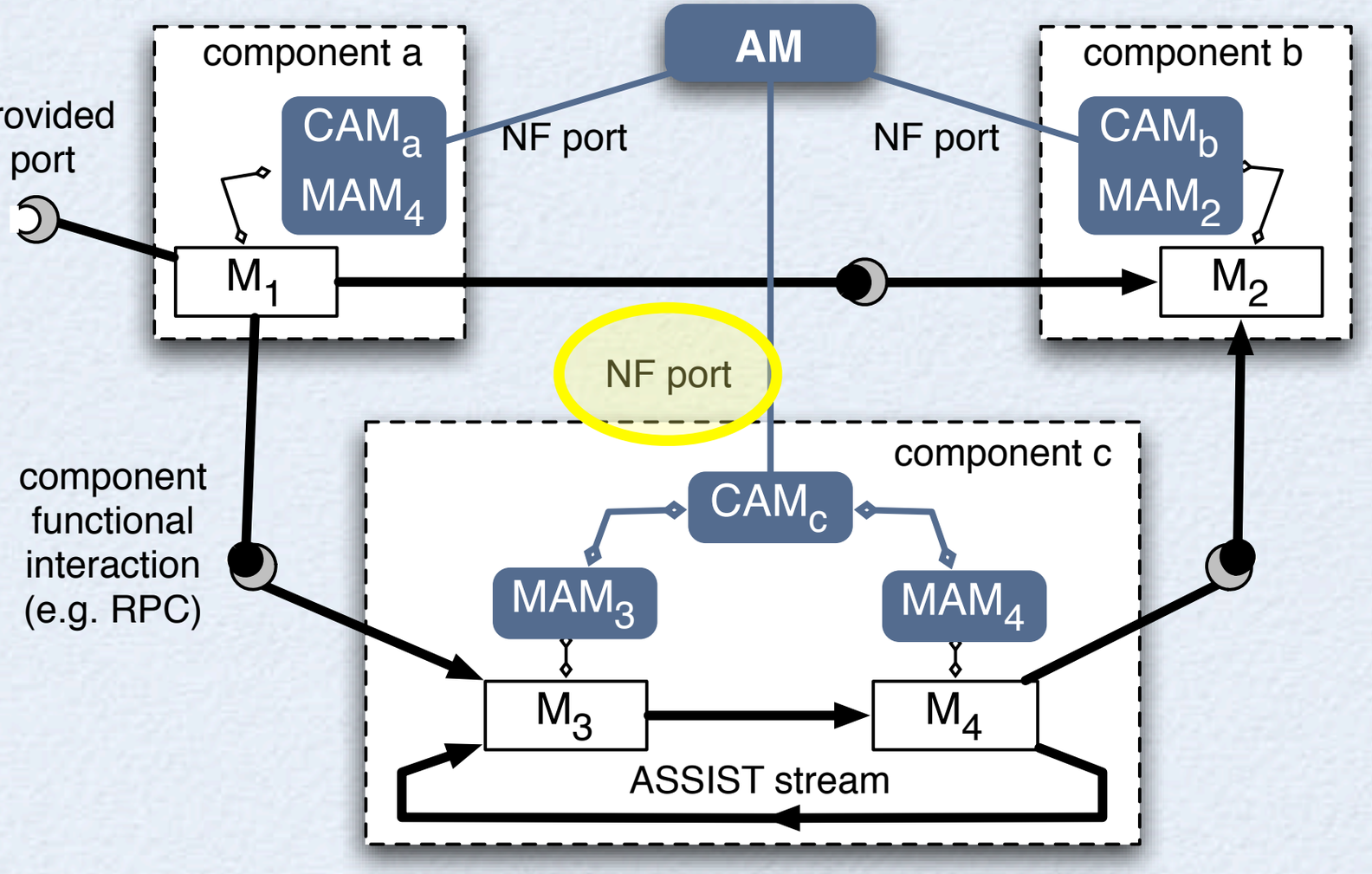
ASSIST & components





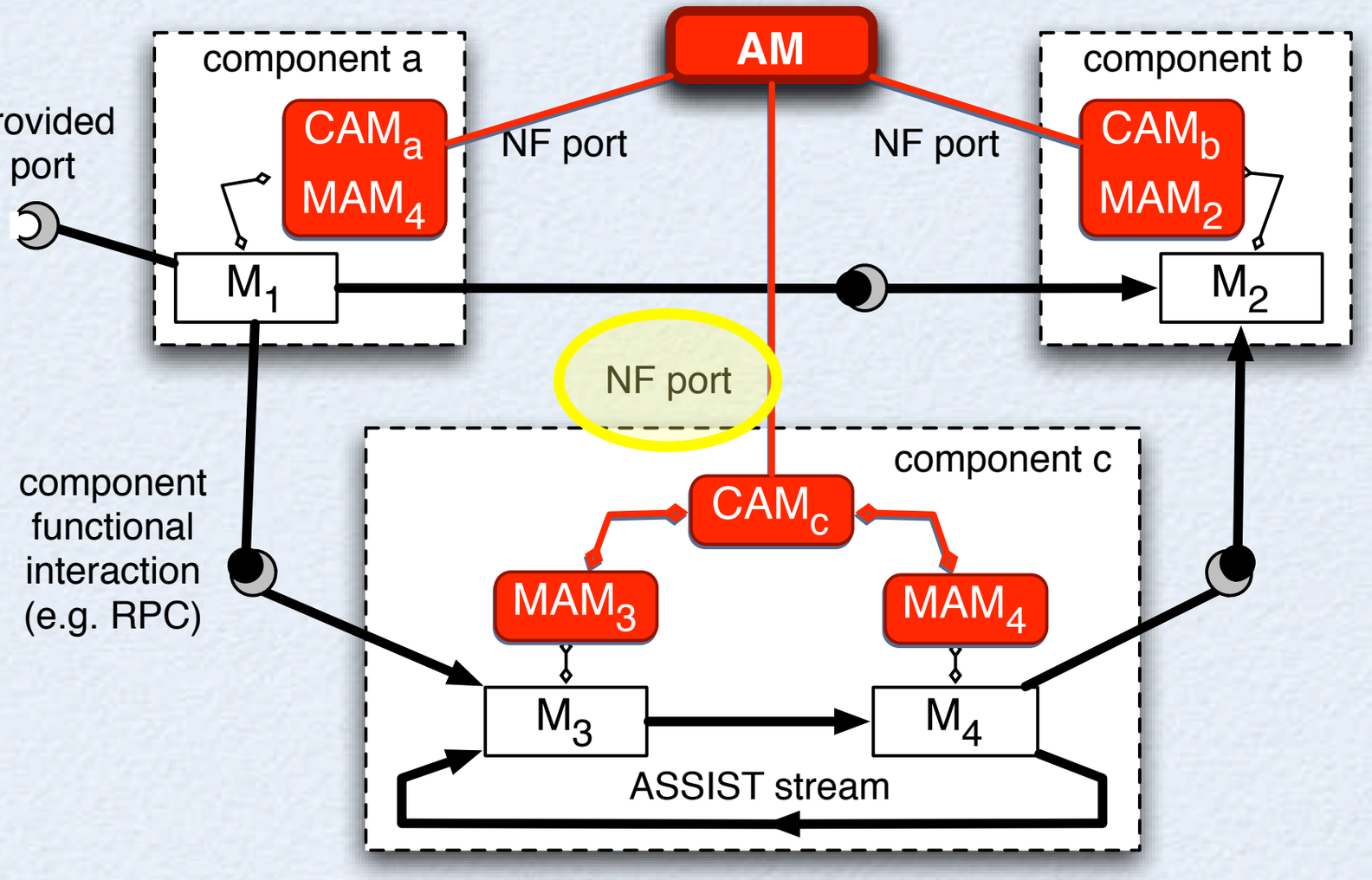
- ASSIST graphs can be enclosed in components
- they can be wired one another
- they may used to wrap sequential or parallel code (e.g. MPI)
- they can be wired to other legacy components (e.g. CCM)
- currently *native component model*, already converging in the forthcoming GCM (authors involved in CoreGRID NoE, WP3)

managed components



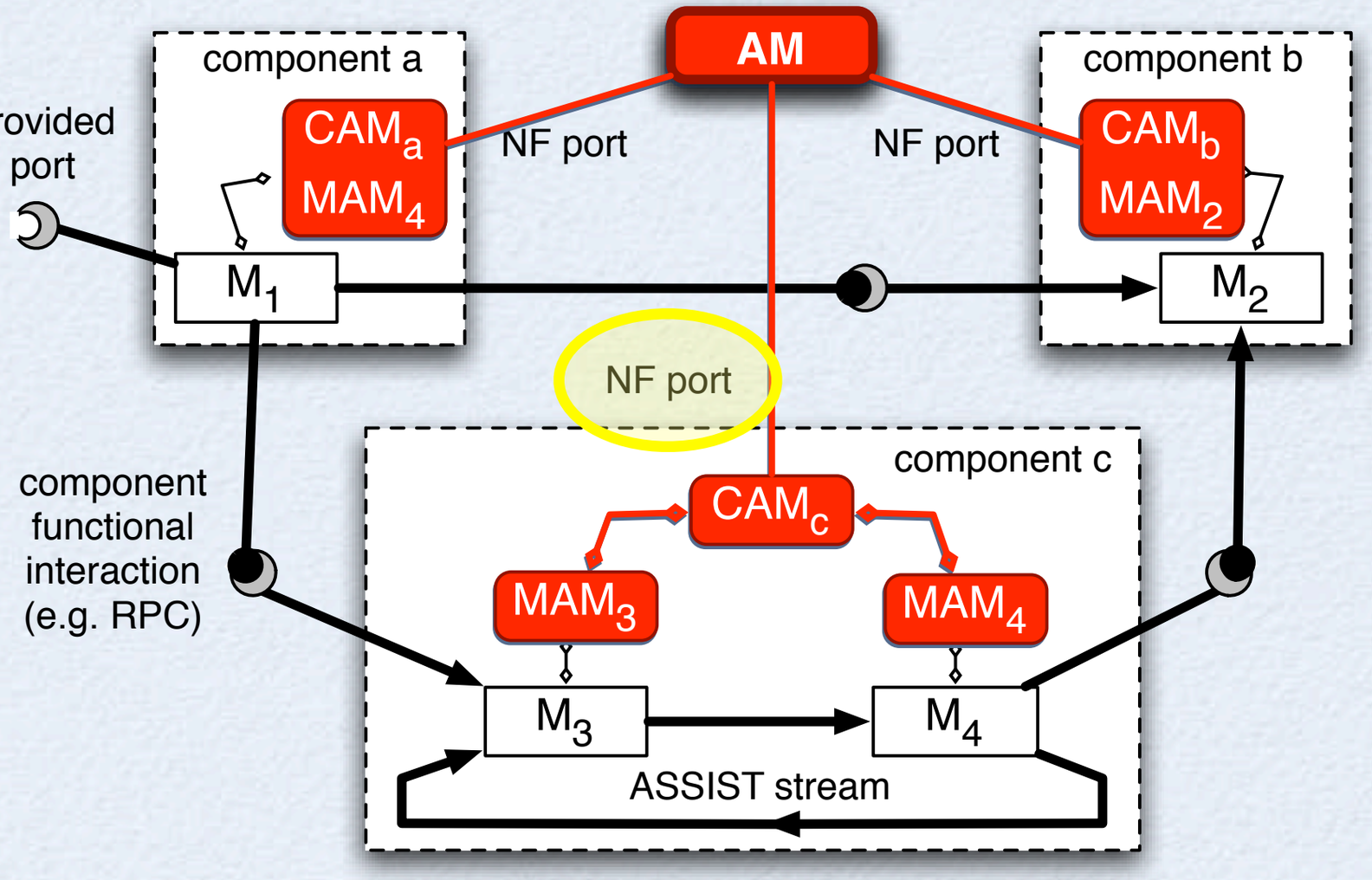
- modules and components are controlled by managers
- managers implements NF-ports

managed components



- modules and components are controlled by managers
- managers implements NF-ports

managed components

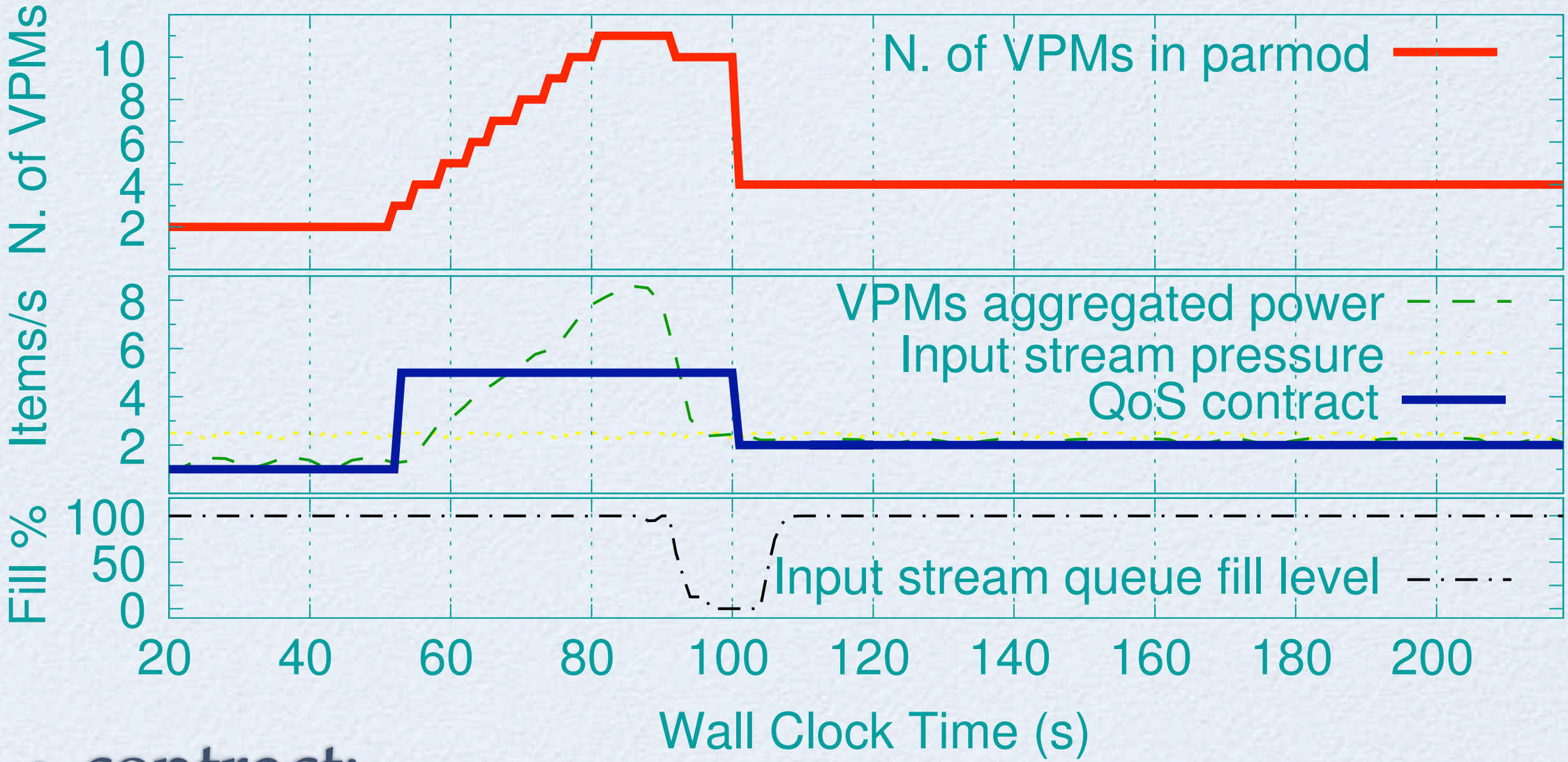


- modules and components are controlled by managers
- managers implements NF-ports
- the distributed coordination of managers enable the managing of the application as whole (the top manager being the Application Manager)

QoS contract
(of the experiment I'll show you in a minute)

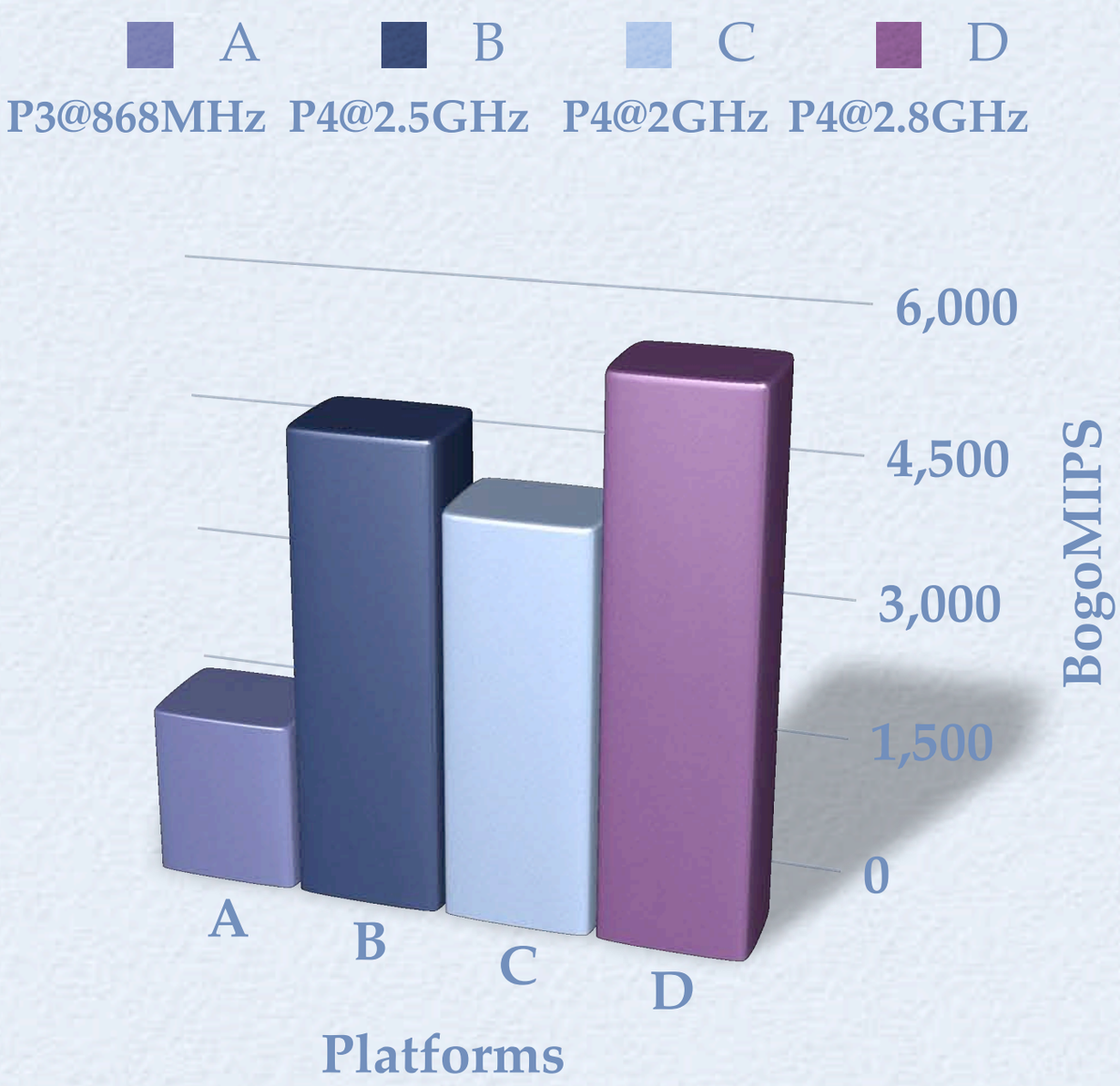
Perf. features	QL_i (input queue level), QL_o (input queue level), T_{ISM} (ISM service time), T_{OSM} (OSM service time), N_w (number of VPMs), $T_w[i]$ (VPM$_i$ avg. service time), T_p (parmod avg. service time)
Perf. model	$T_p = \max\{T_{ISM}, \sum_{i=1}^{n} T_w[i]/n, T_{OSM}\}$, $T_p < K$ (goal)
Deployment	arch = (i686-pc-linux-gnu $\vee$ powerpc-apple-darwin*)
Adapt. policy	goal_based

experiment: stateless farm

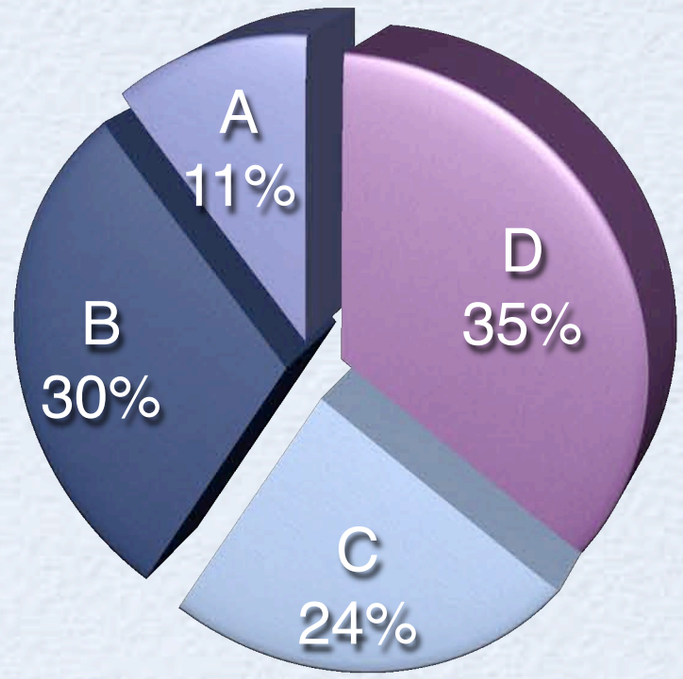


- contract:
  - keep a given service time
  - contract change along the run

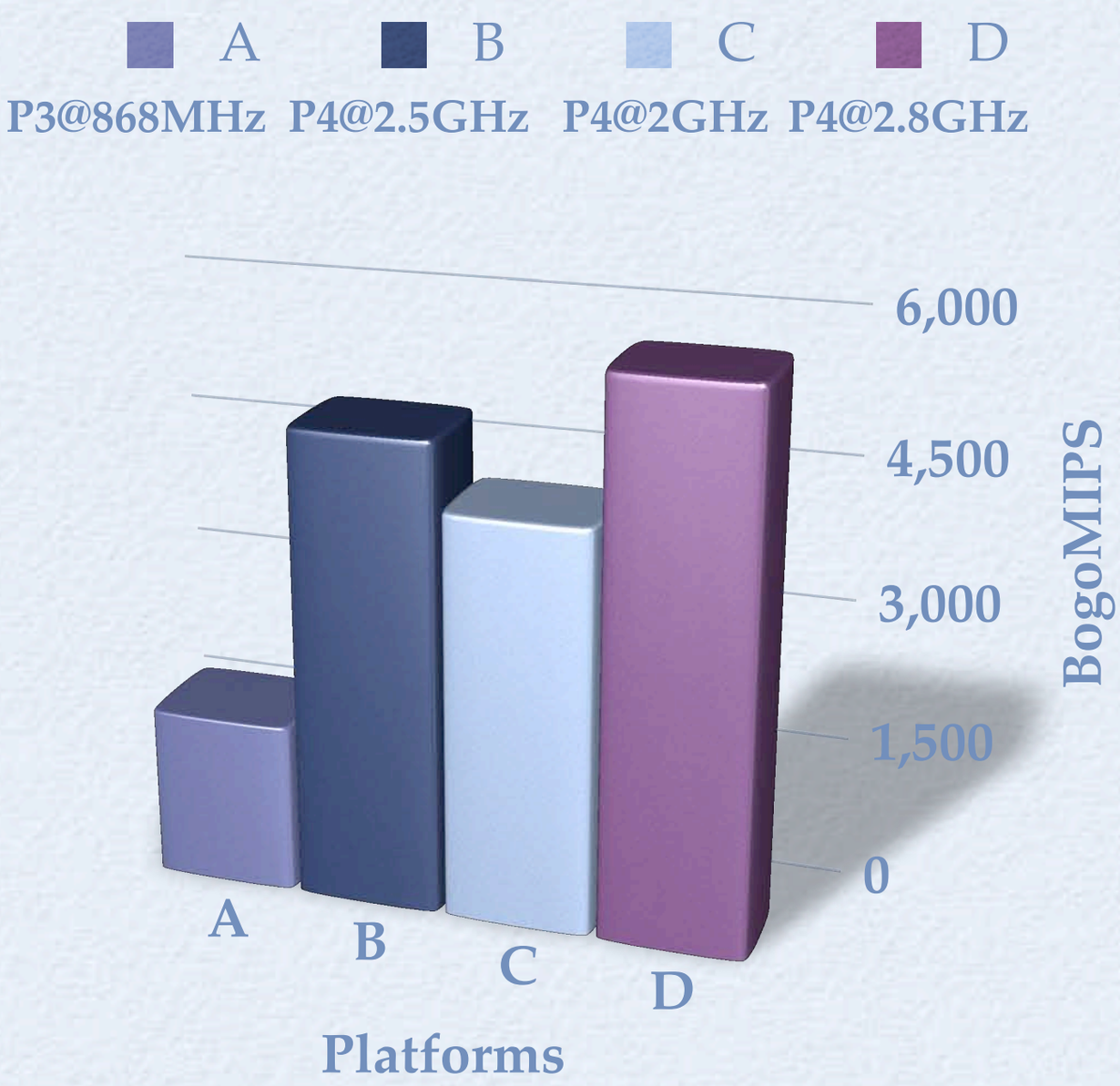
Experimenting heterogeneity



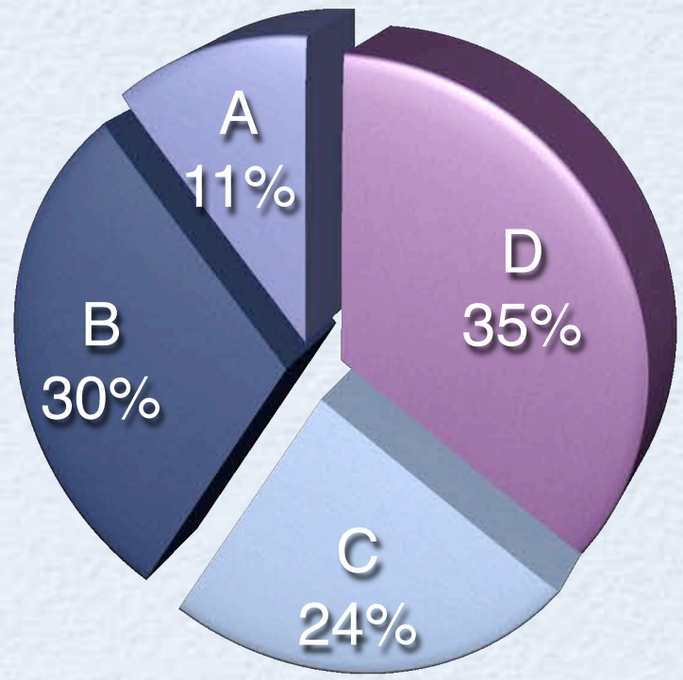
Expected work balance among platforms



Experimenting heterogeneity

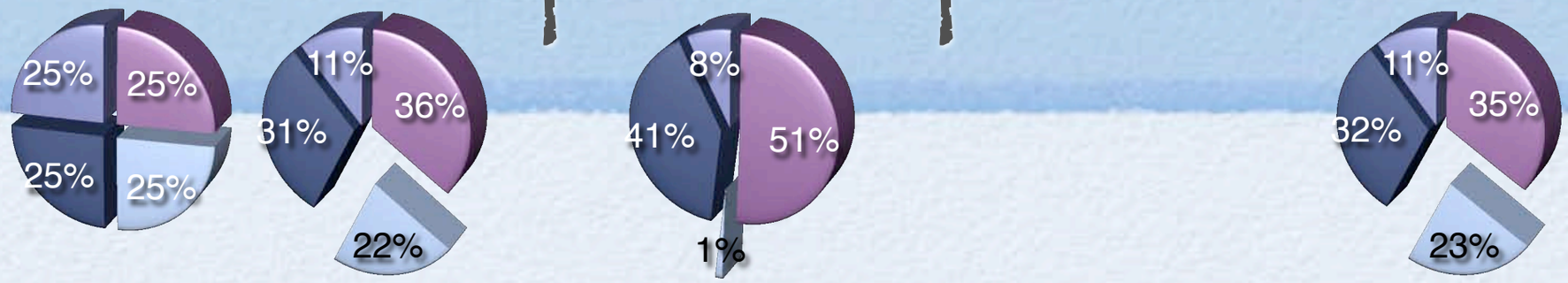


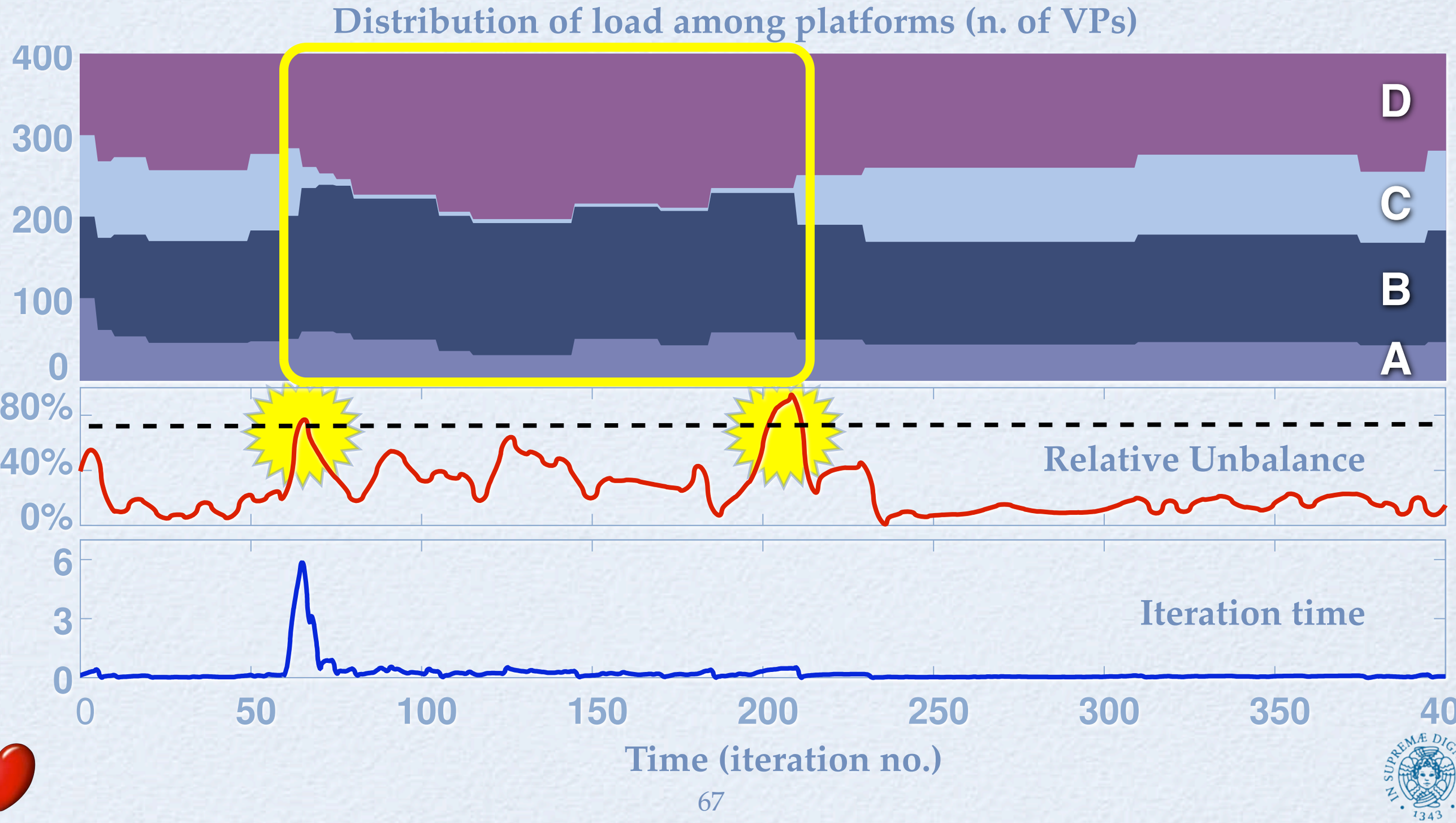
Expected work balance among platforms



Not only Intel+linux: similar experiments has been run on Linux, Mac, Win, and a mixture of them

Data-par experiment (STP)





Conclusions 1/2

- Application adaptivity in ASSIST
  - complex, but trasparent (no burden for the programmers)
    - they should just define they QoS requirements
    - QoS models are automatically generated from program structure (and don't depend on seq. funct.)
- dynamically controlled, efficiently managed
  - catch both platforms unsteadiness and code irregular behavior in running time
  - performance models not critical, reconfiguration does not stop the application
  - key feature for the grid

Conclusions 2/2

- ASSIST cope with
  - grid platform unsteadiness
  - interoperability with standards
    - and rely on them for many features
- high-performance
- app deployment problems on grid
  - private networks, job schedulers, firewalls, ...
- QoS of the whole application through hierarchy of managers

ASSIST is open source under GPL

`http://www.di.unipi.it/Assist.html`