ENS Lyon, France

29 March 2006

Toward autonomic QoS in Grid-aware applications: the ASSIST experiment

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

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

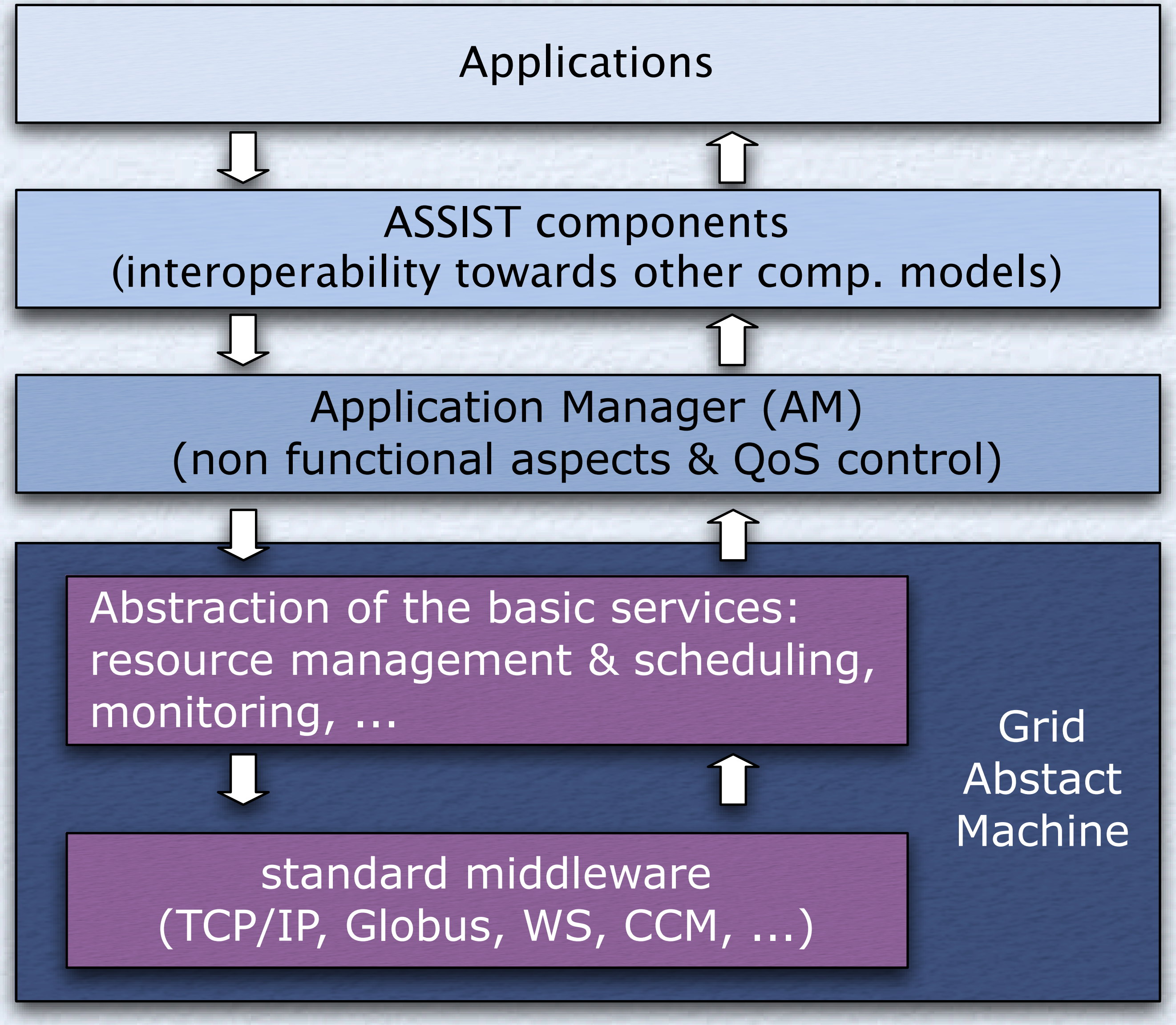
- concurrency exploitation, concurrent activities set up, mapping/scheduling, communication/synchronization handling and data allocation, ...
- manage resources heterogeneity and unreliability, networks latency and bandwidth unsteadiness, resources topology and availability changes, firewalls, private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

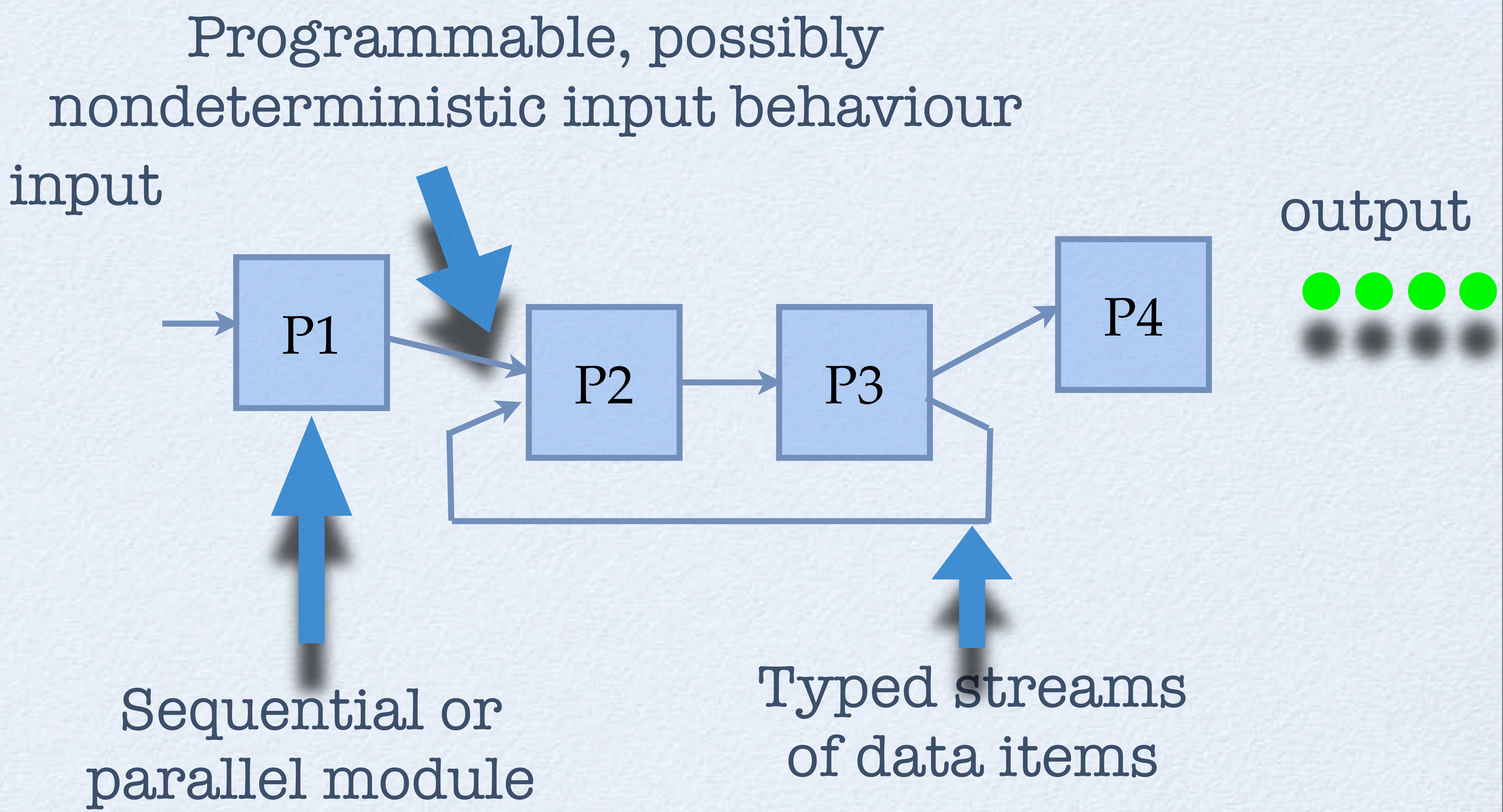
ASSIST idea



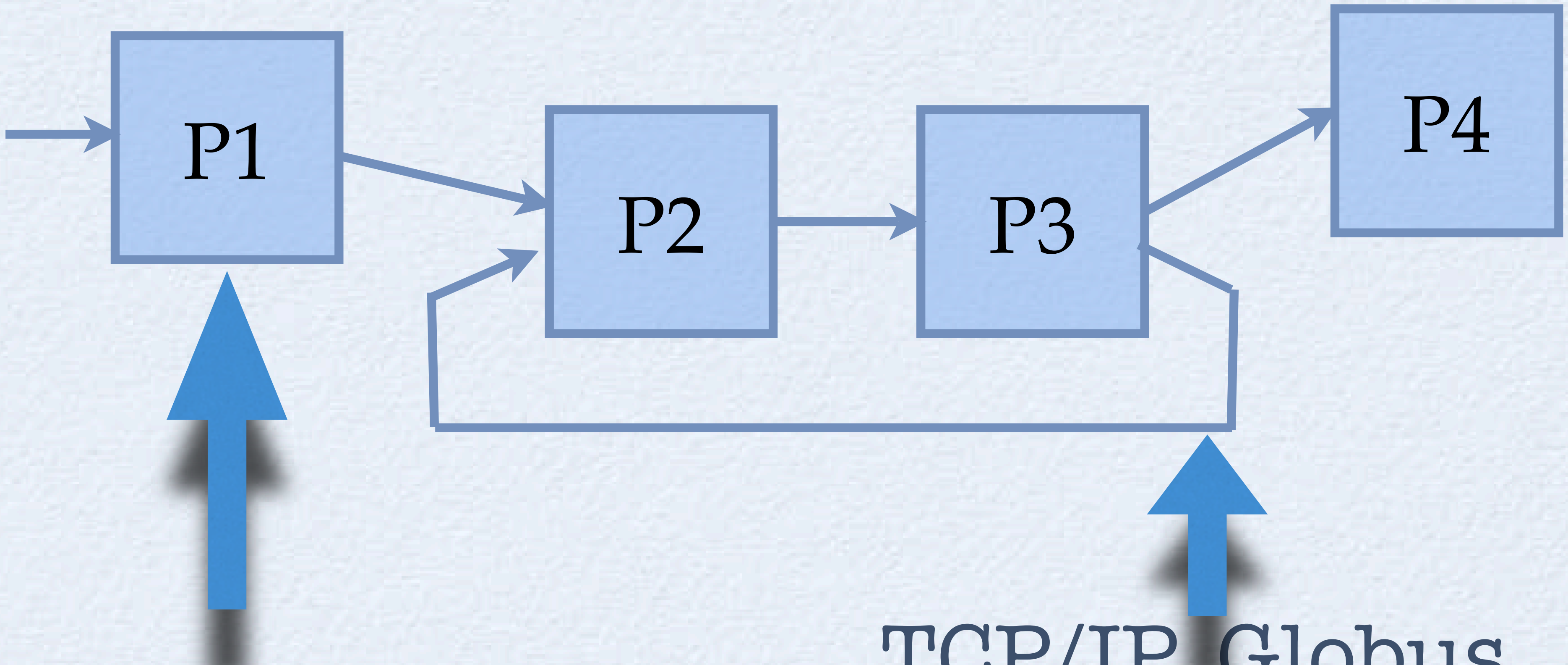
"moving most of the Grid specific efforts needed while developing high-performance Grid applications from programmers to grid tools and run-time systems"

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



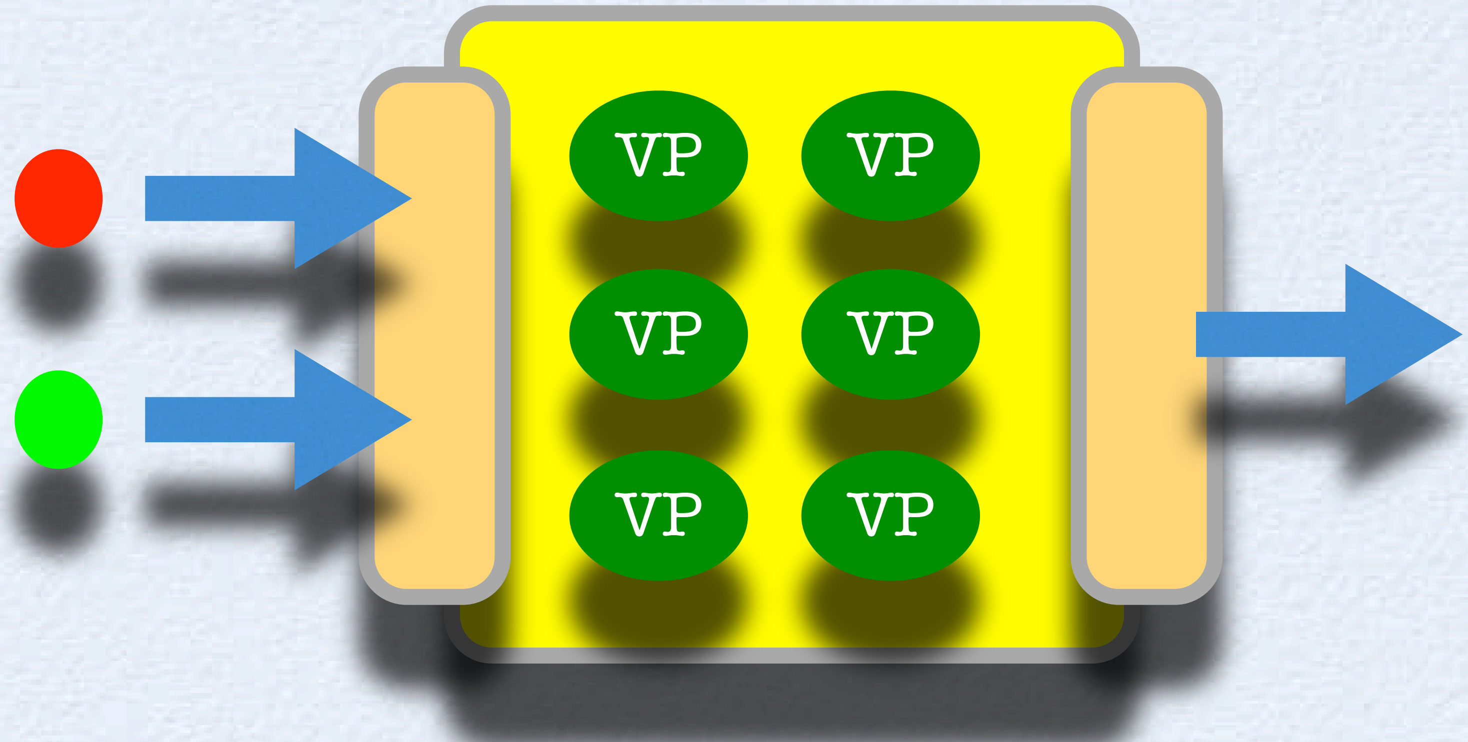
native + standards



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

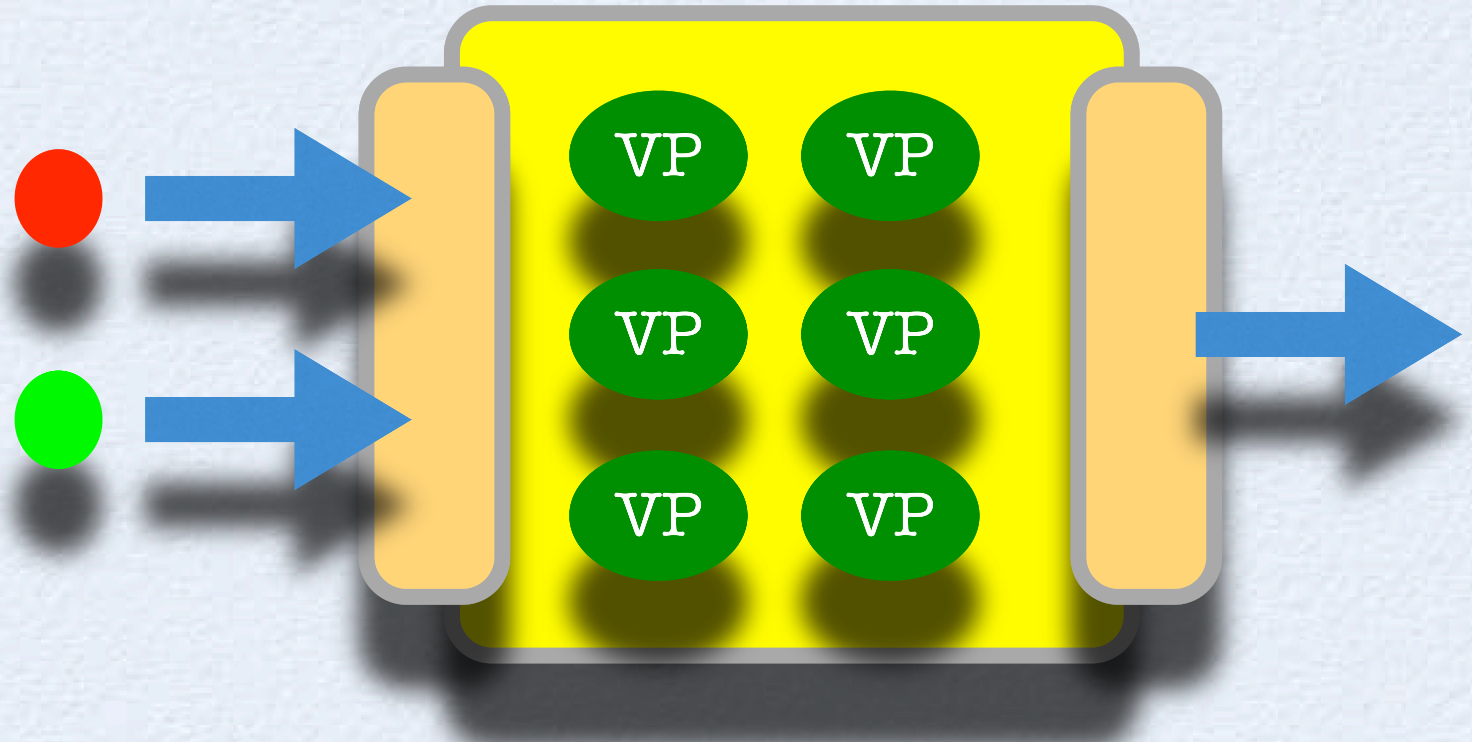
TCP/IP, Globus, IIOP CORBA, HTTP/SOAP

ASSIST parmod



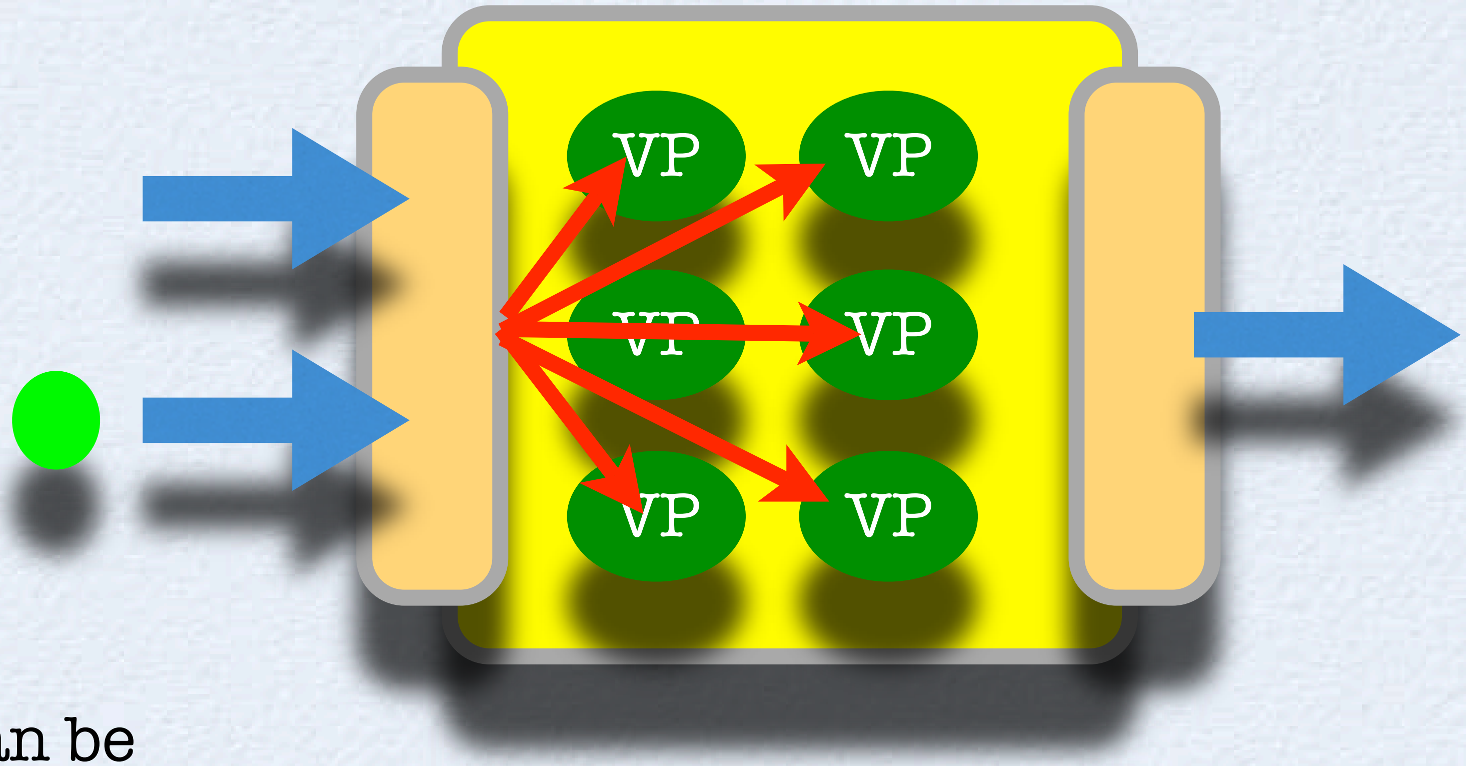
ASSIST parmod

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



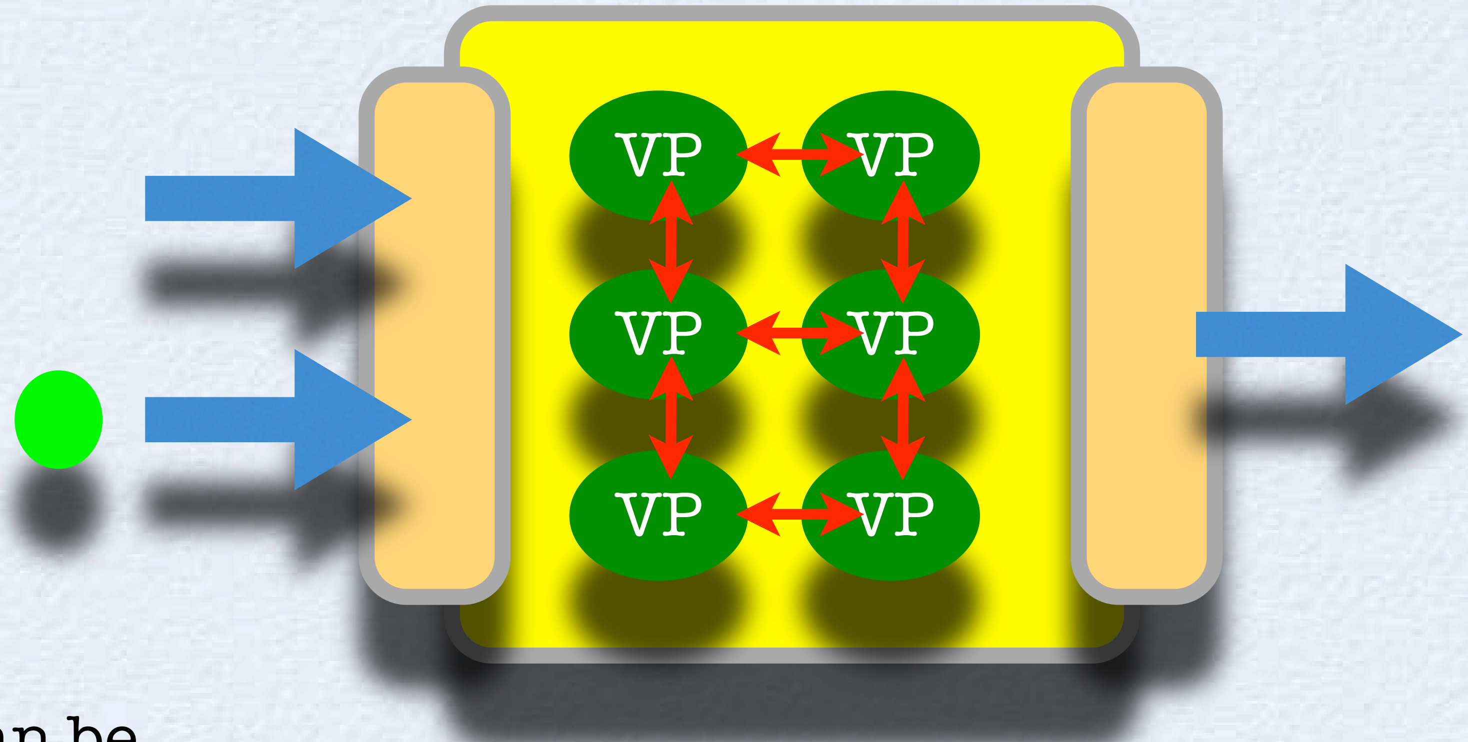
ASSIST parmod

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

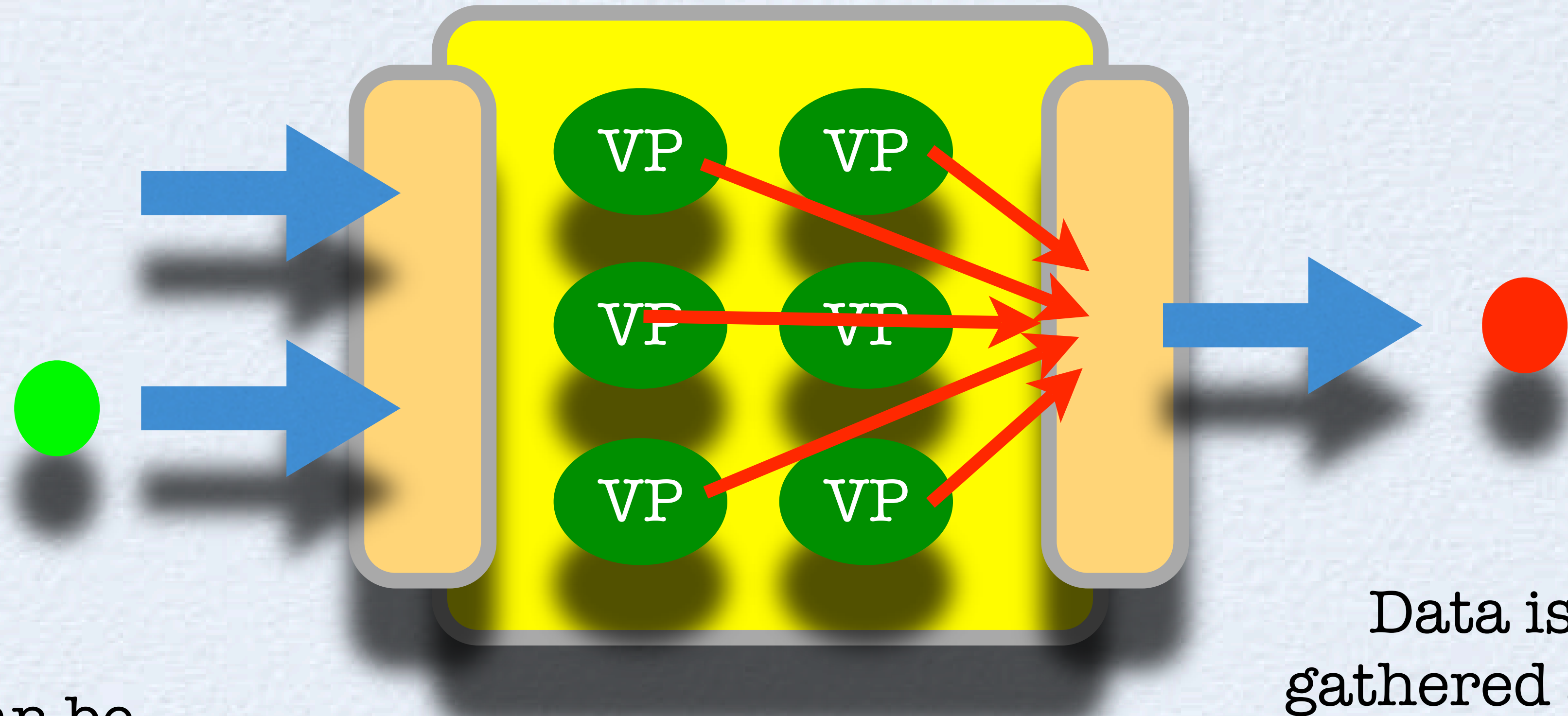
Data items can be distributed (scattered, broadcasted, multicasted) to a set of **Virtual Processes** which are named accordingly to a topology

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 is eventually gathered accordingly to an user defined way

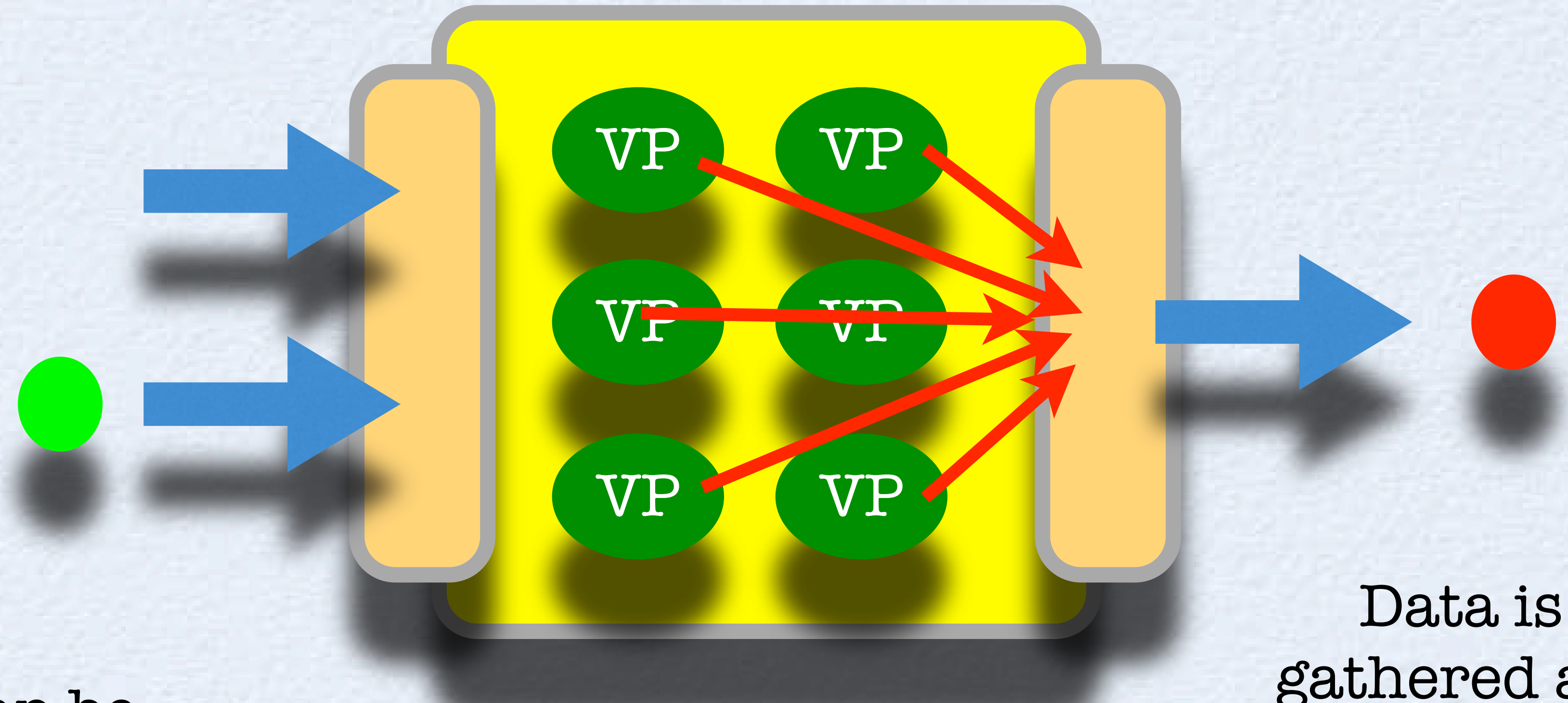
Data items can be distributed (scattered, broadcasted, multicasted) to a set of **Virtual Processes** which are named accordingly to a topology

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 is eventually gathered accordingly to an user defined way

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

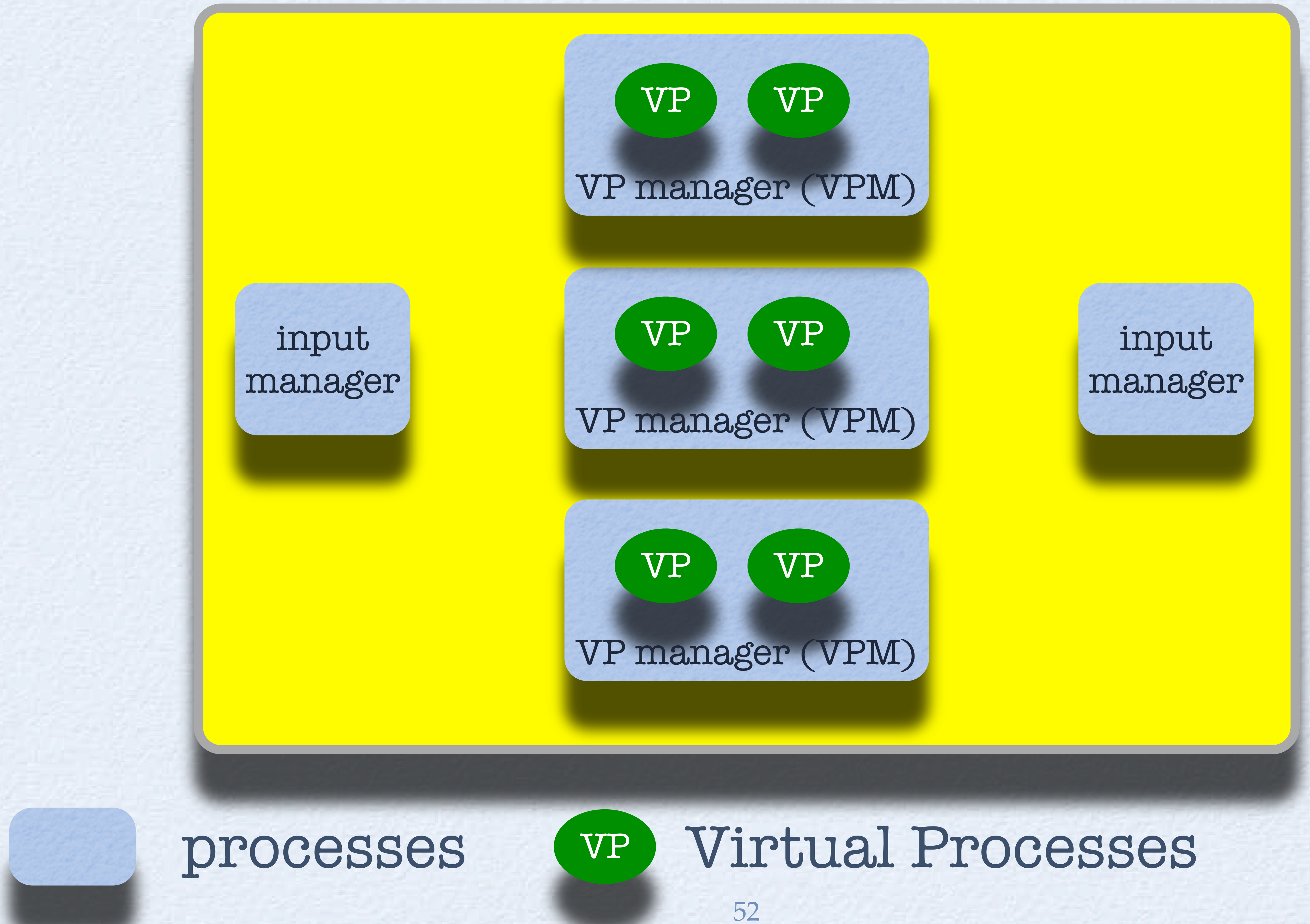
Data items partitions are elaborated by VPs, possibly in iterative way

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

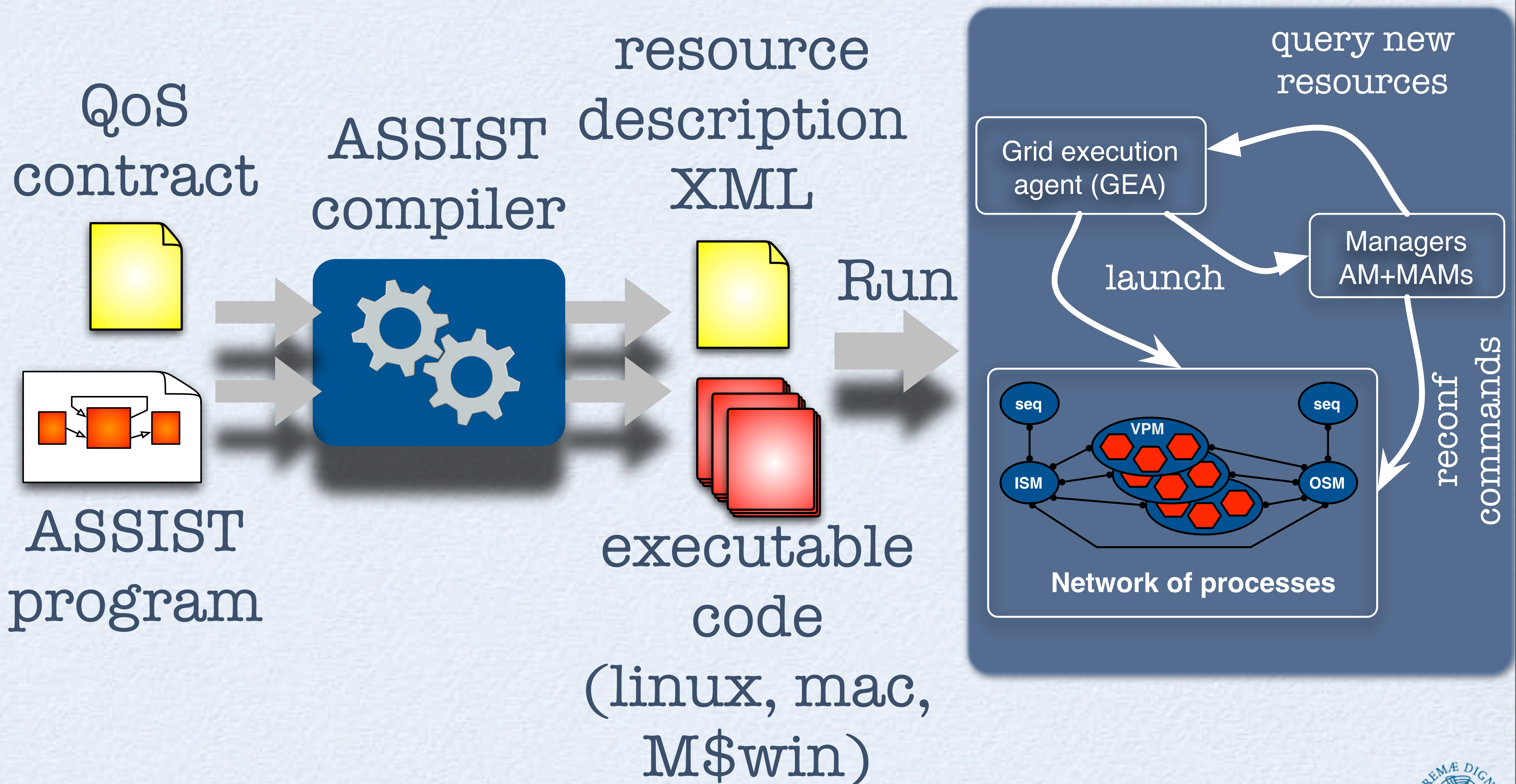
data is logically shared by VPs (owner-computes)

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

parmod implementation



Compiling & running



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)

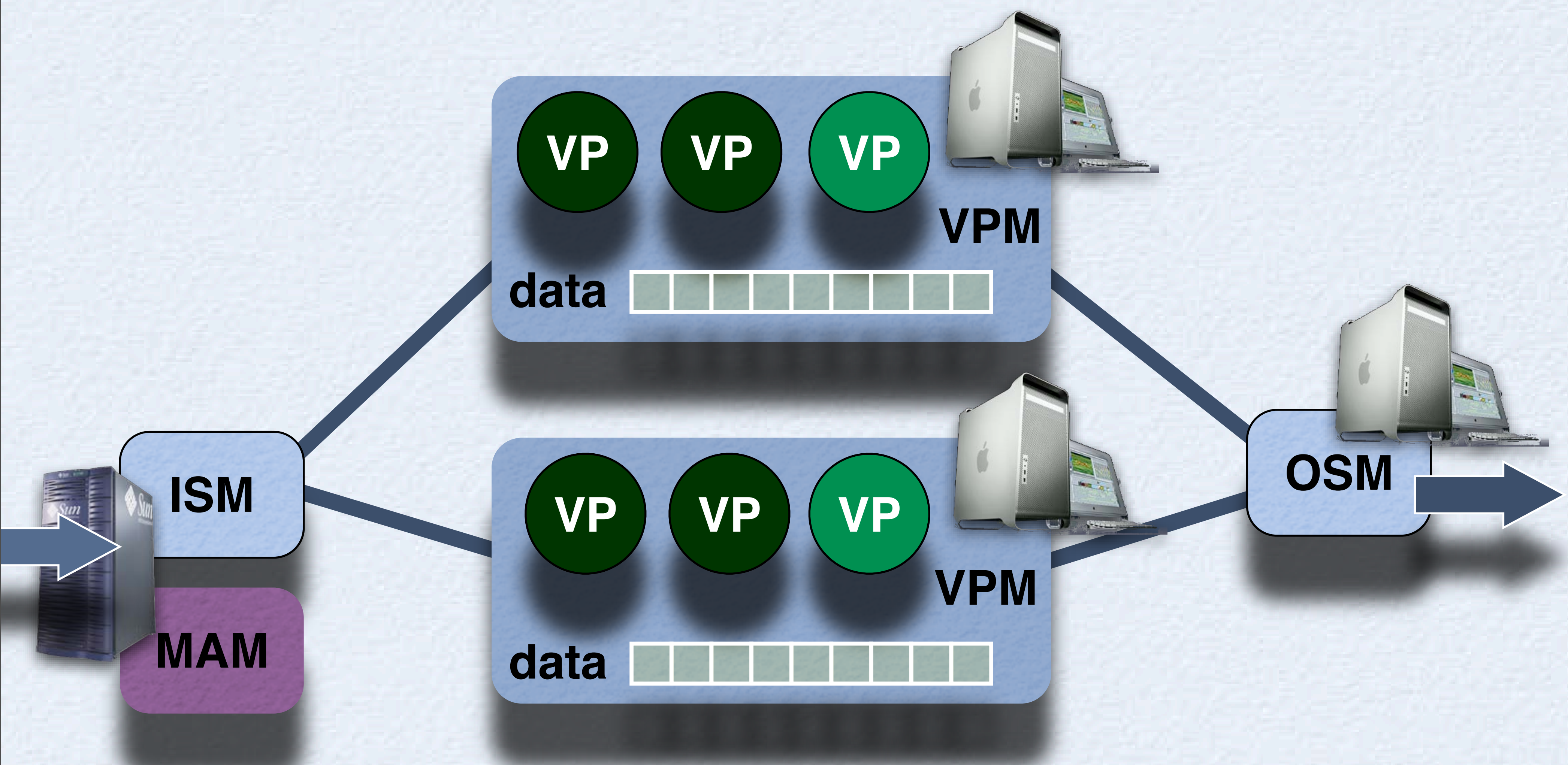
Adaptivity recipe (ingredients)

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

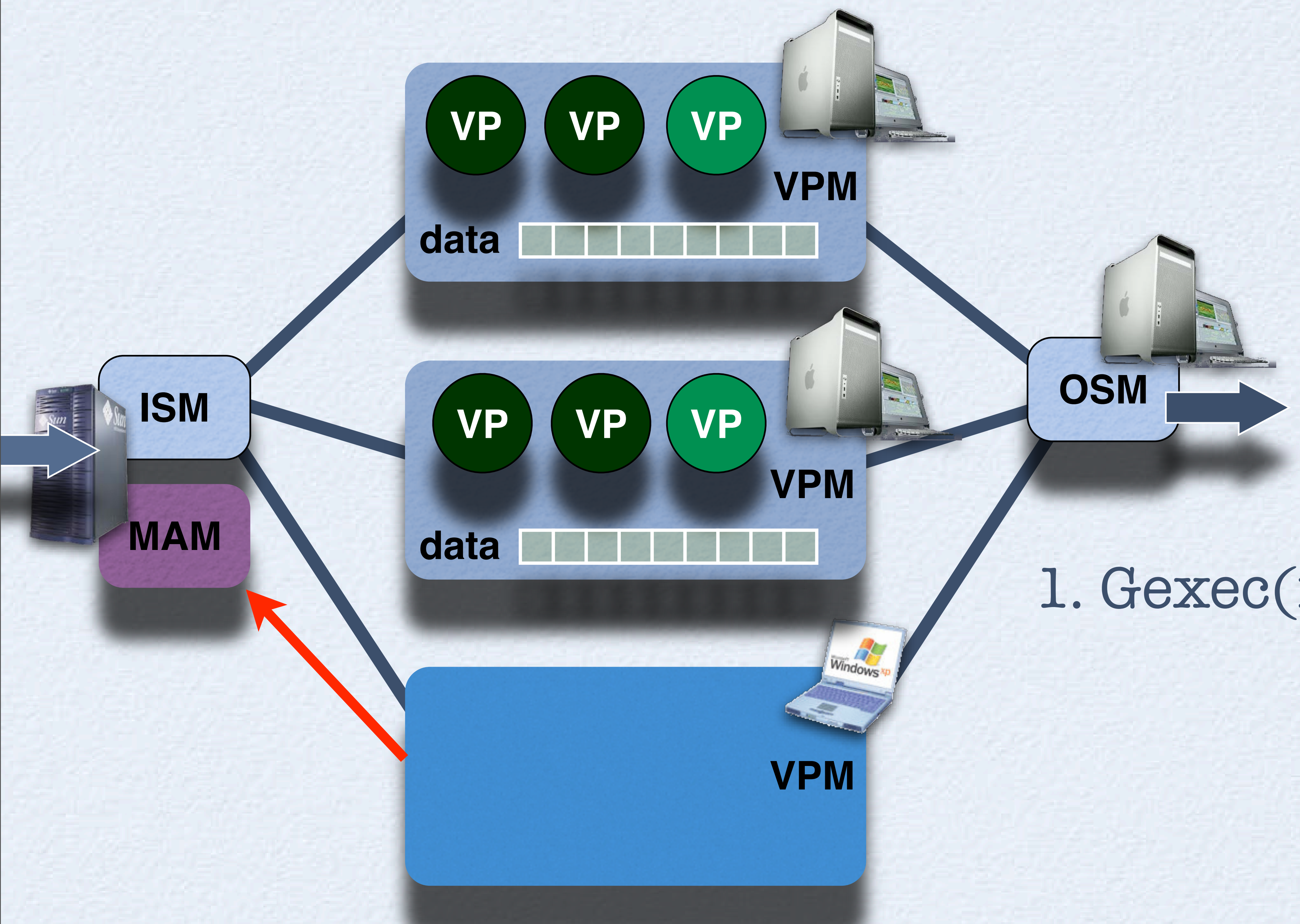
Mechanisms

- 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.

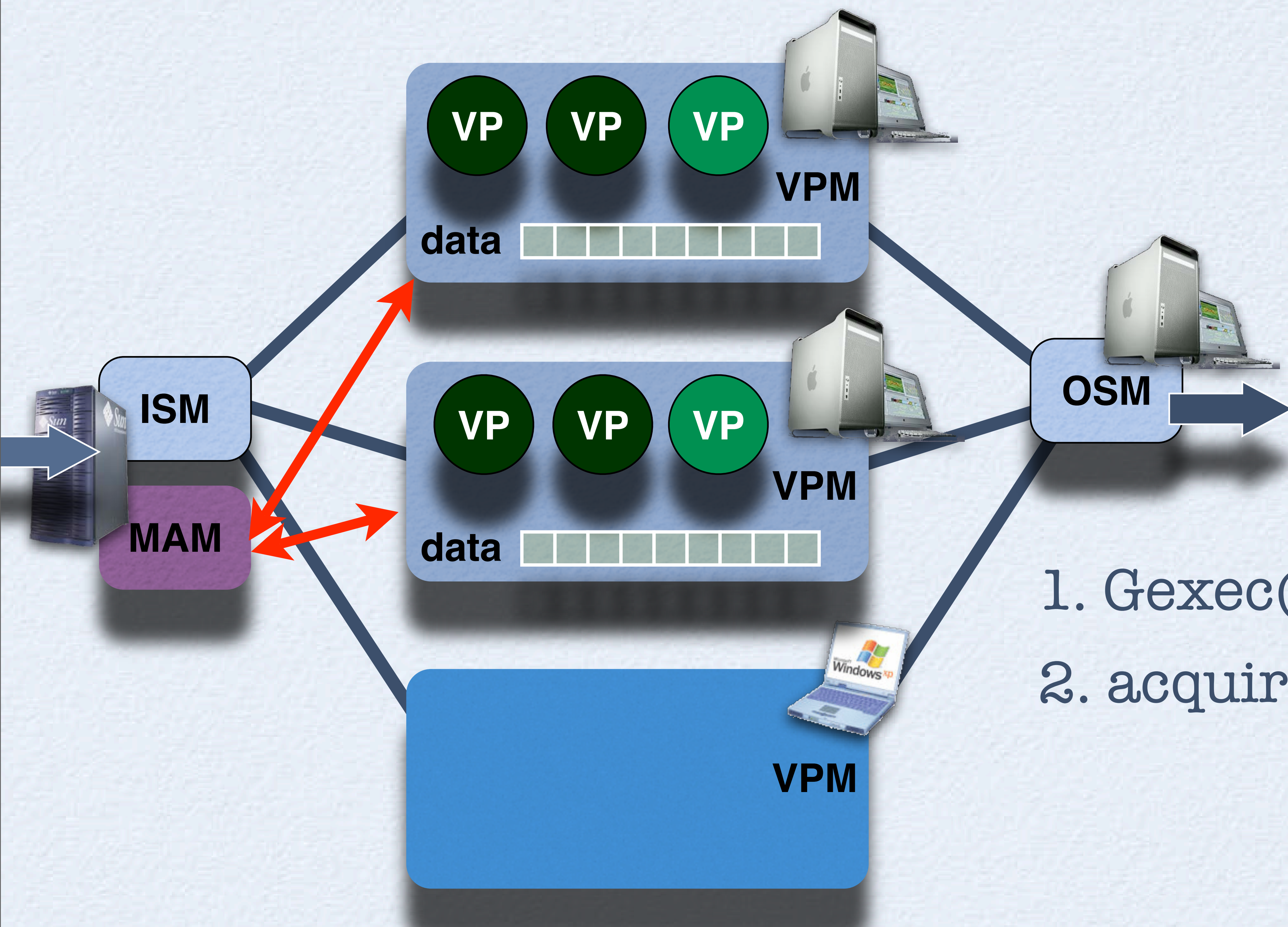


adaptivity: a working ex.



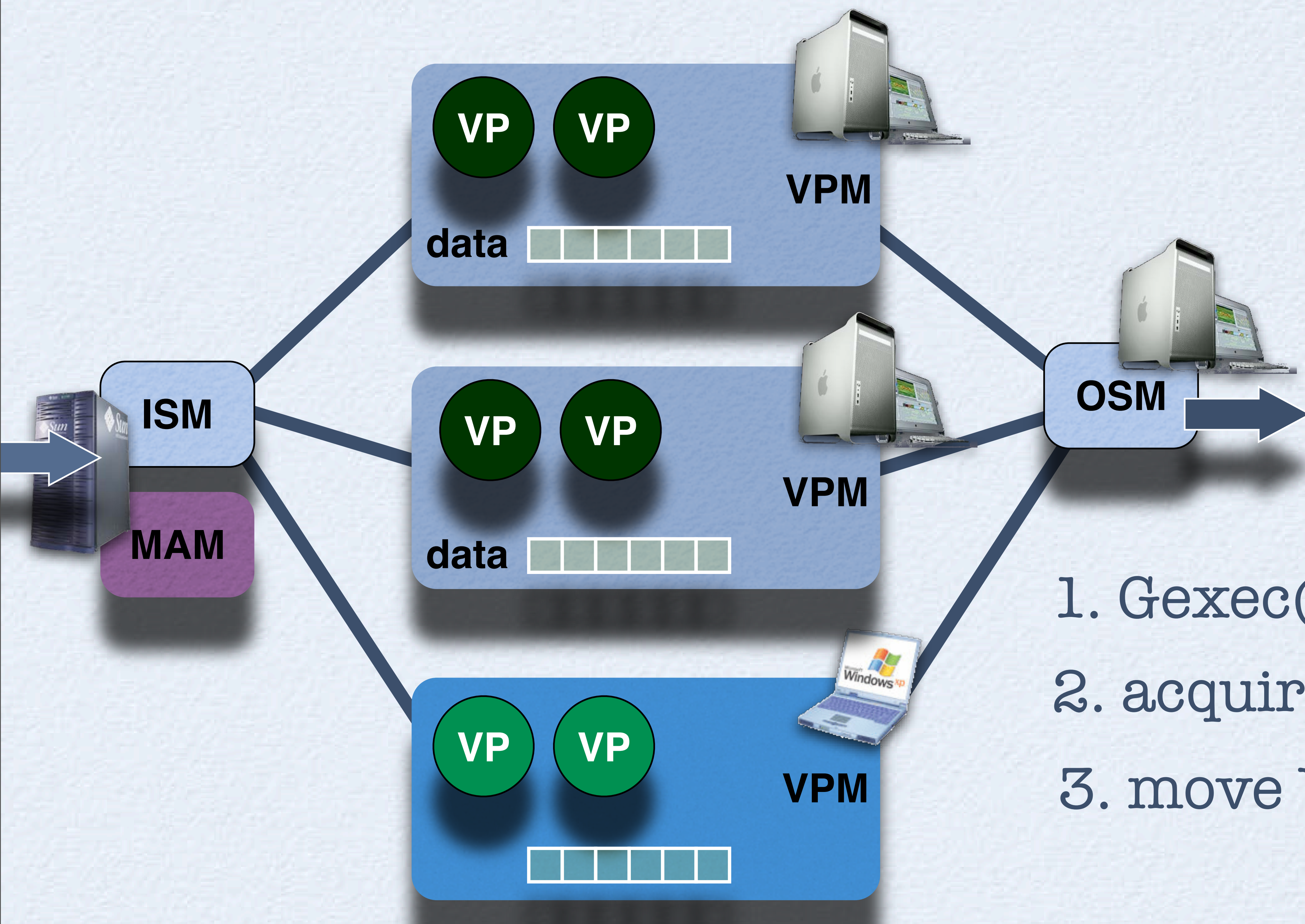
1. Gexec(newPE, VPM)

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

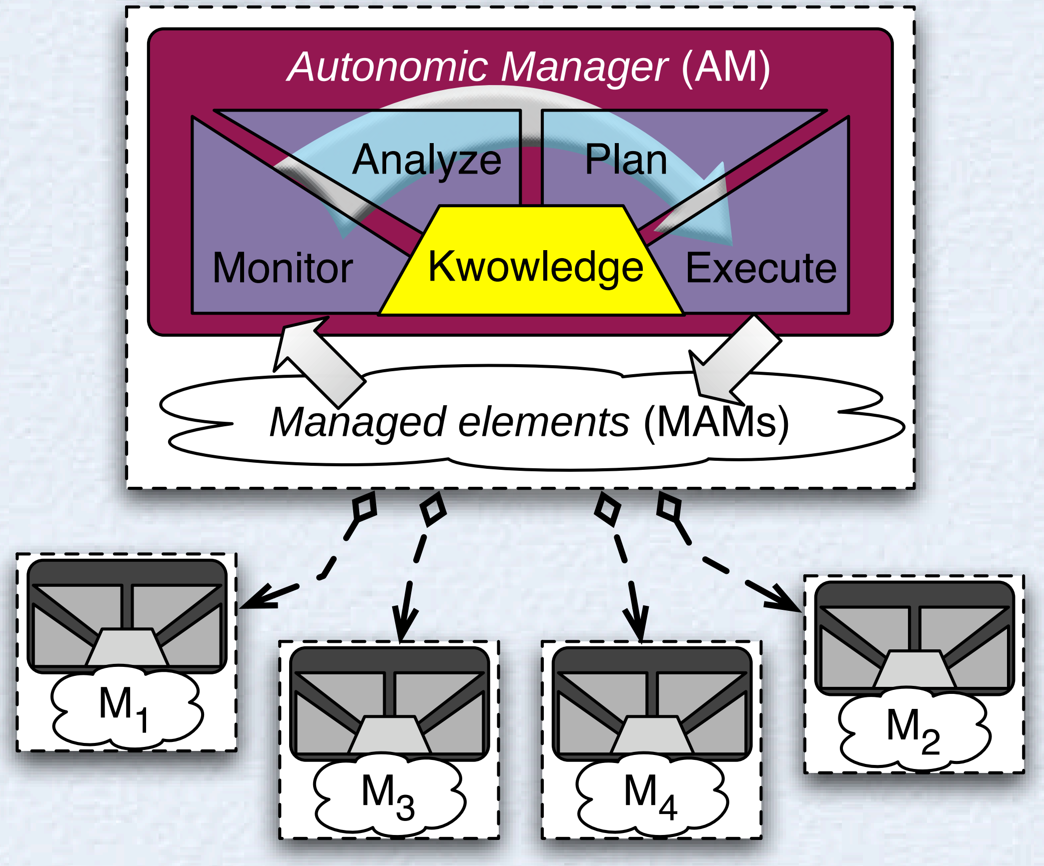
Only 3. is in the critical path

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

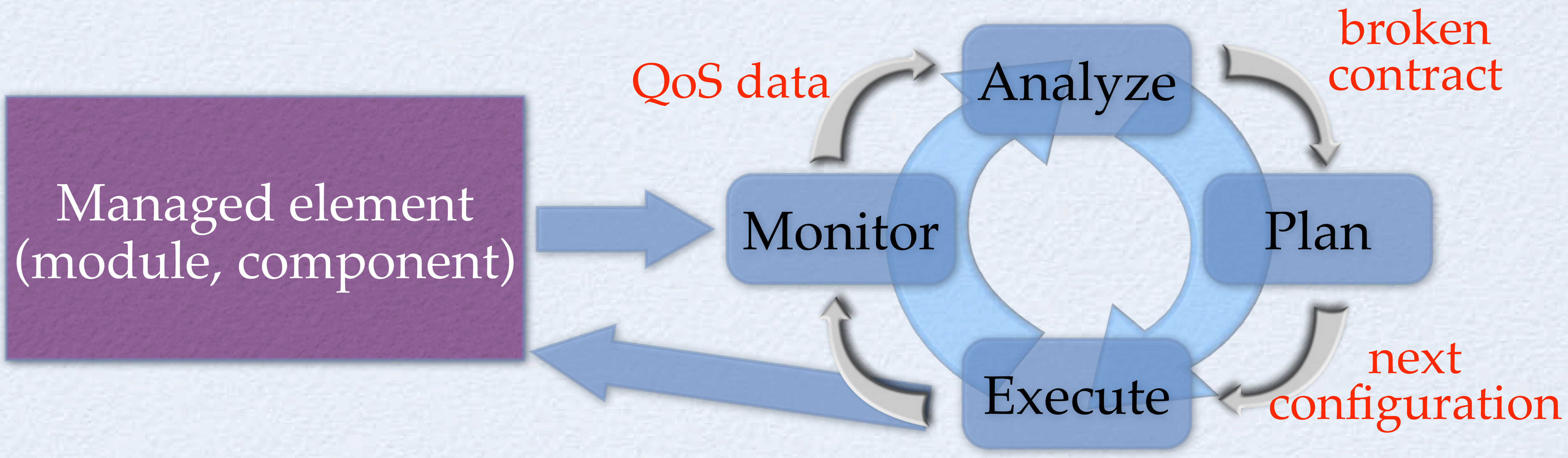
GrADS papers reports overhead in the order of hundreds of seconds (K. Kennedy et al. 2004), this is mainly due to the stop/restart behavior, not to the different running env.

Autonomic Computing



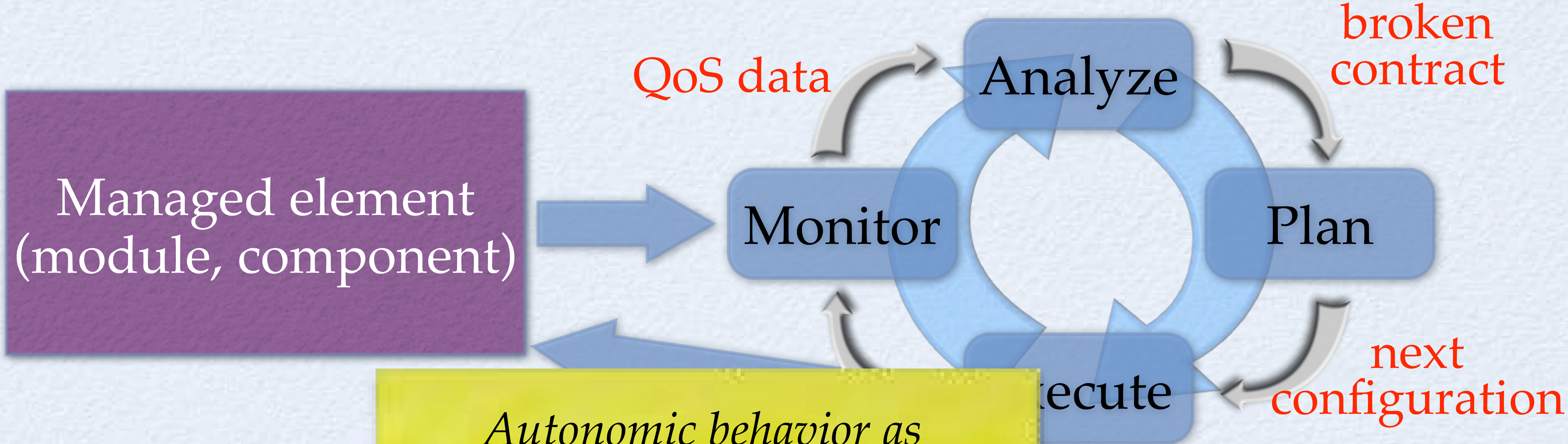
- AC emblematic of a vast hierarchy of self-governing systems, many of which consist of many interacting, self-governing components that in turn comprise a number of interacting, self-governing components at the next level down.
- IBM "invented" it in 2001 (control with self-awareness, from human body 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

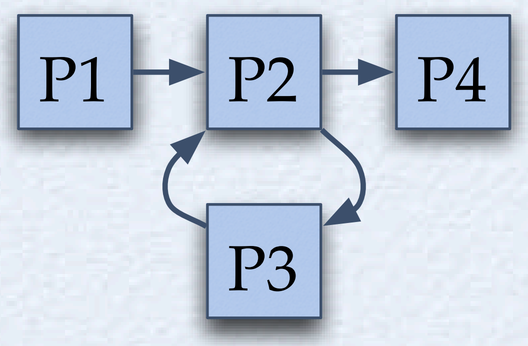
Autonomic behavior



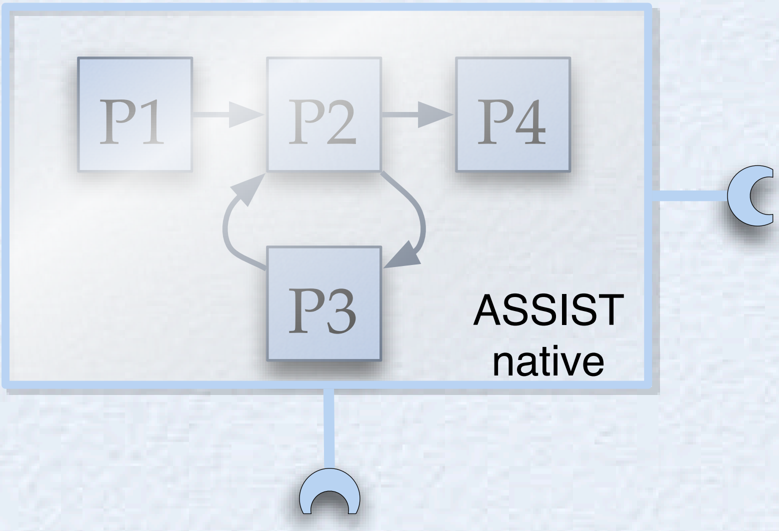
Autonomic behavior as been included in NGG2/3 (Next Generation Grid) EU founding recommendation as prerequisite for Grid computing

- **monitor**: collect … service time, input/output queues lenghts, …
- **analyze**: instanci… …red 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

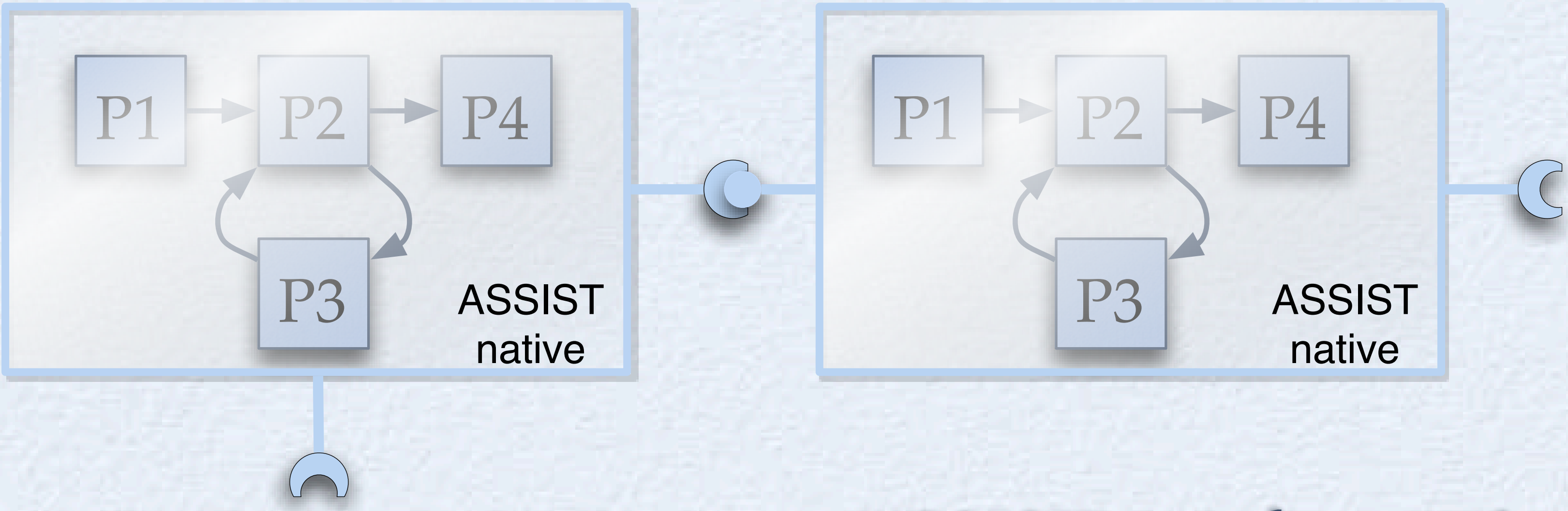


ASSIST & components



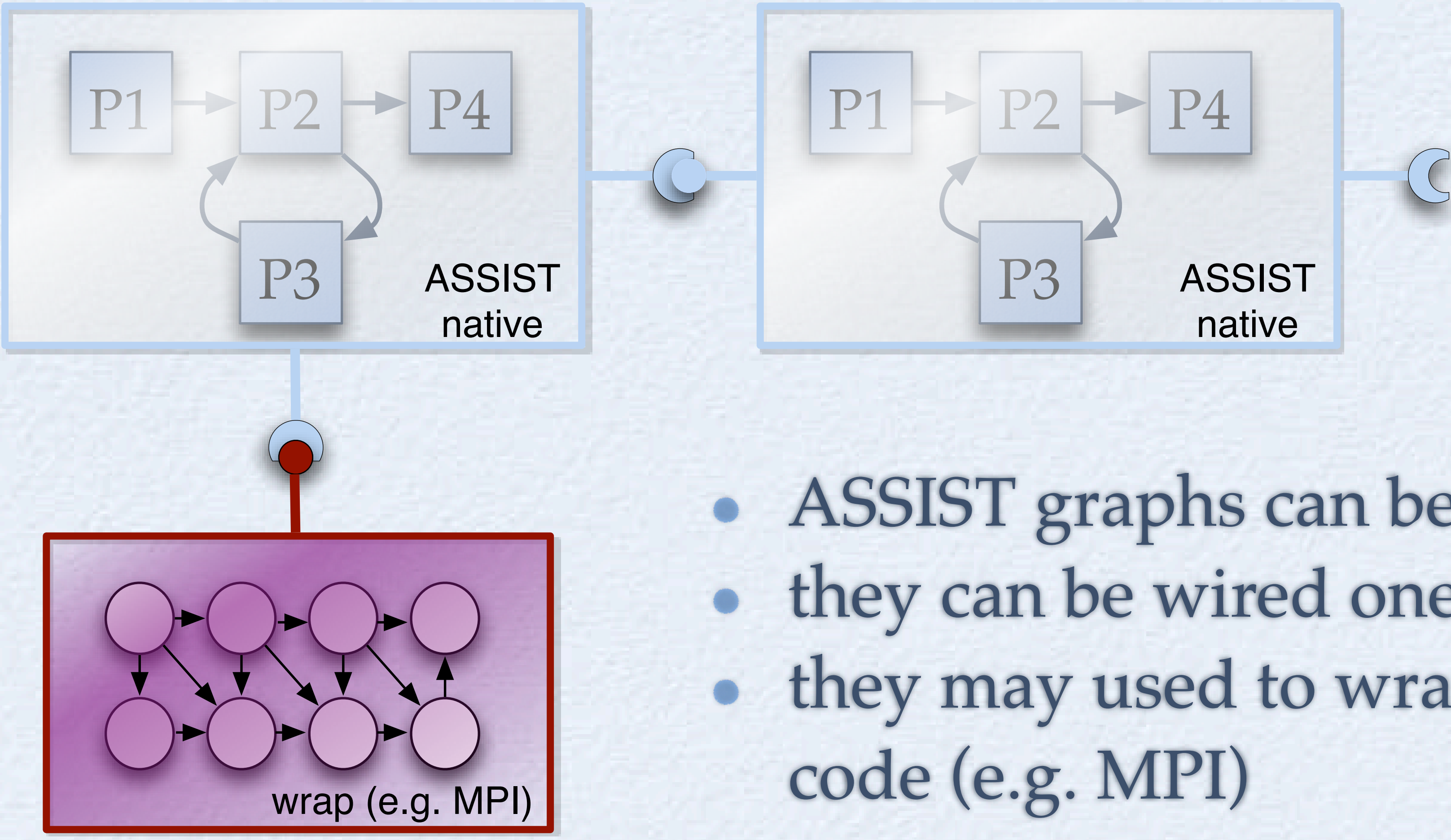
- ASSIST graphs can be enclosed in components

ASSIST & components



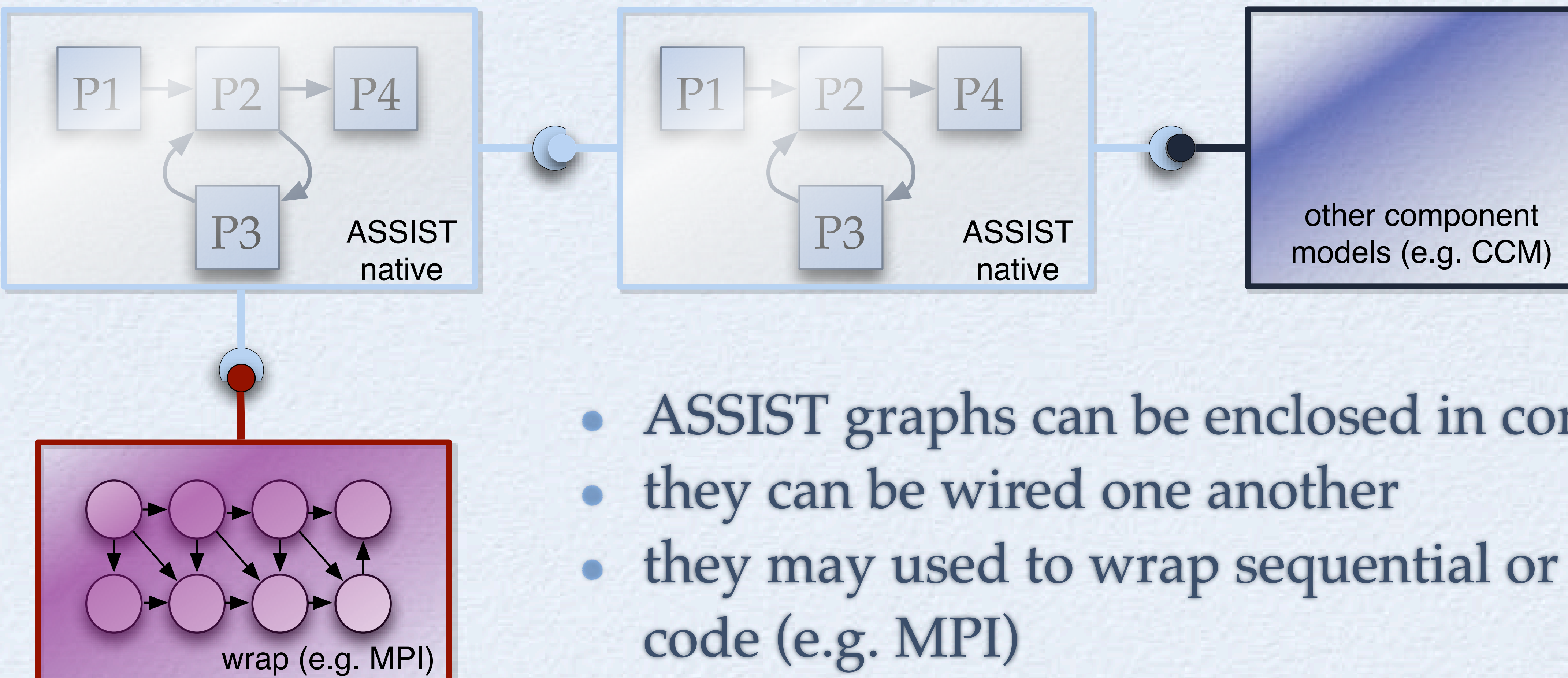
- ASSIST graphs can be enclosed in components
- they can be wired one another

ASSIST & components



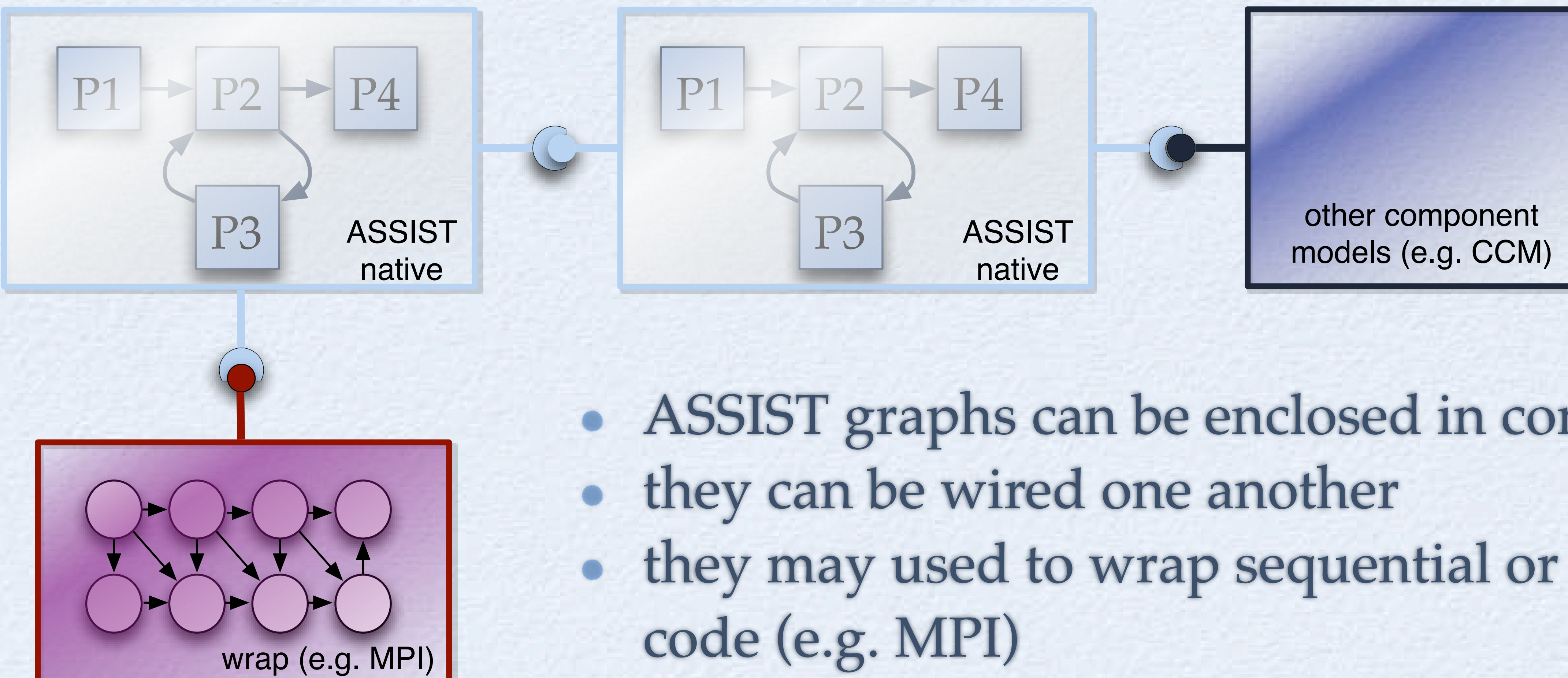
- ASSIST graphs can be enclosed in components
- they can be wired one another
- they may used to wrap sequential or parallel code (e.g. MPI)

ASSIST & components



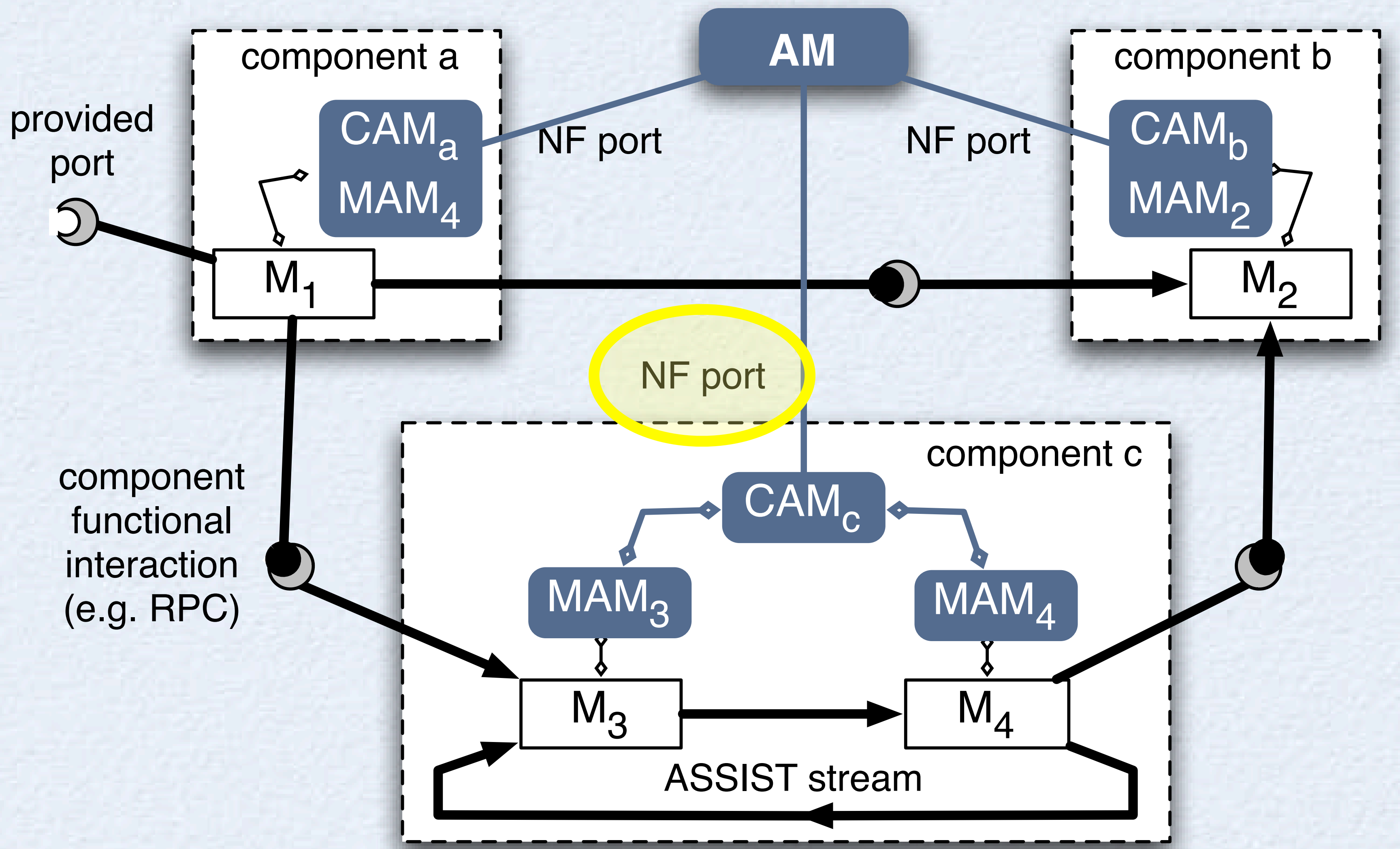
- ASSIST graphs can be enclosed in components
- they can be wired one another
- they may used to wrap sequential or parallel code (e.g. MPI)
- they can be wired to other legacy components (e.g. CCM)

ASSIST & components



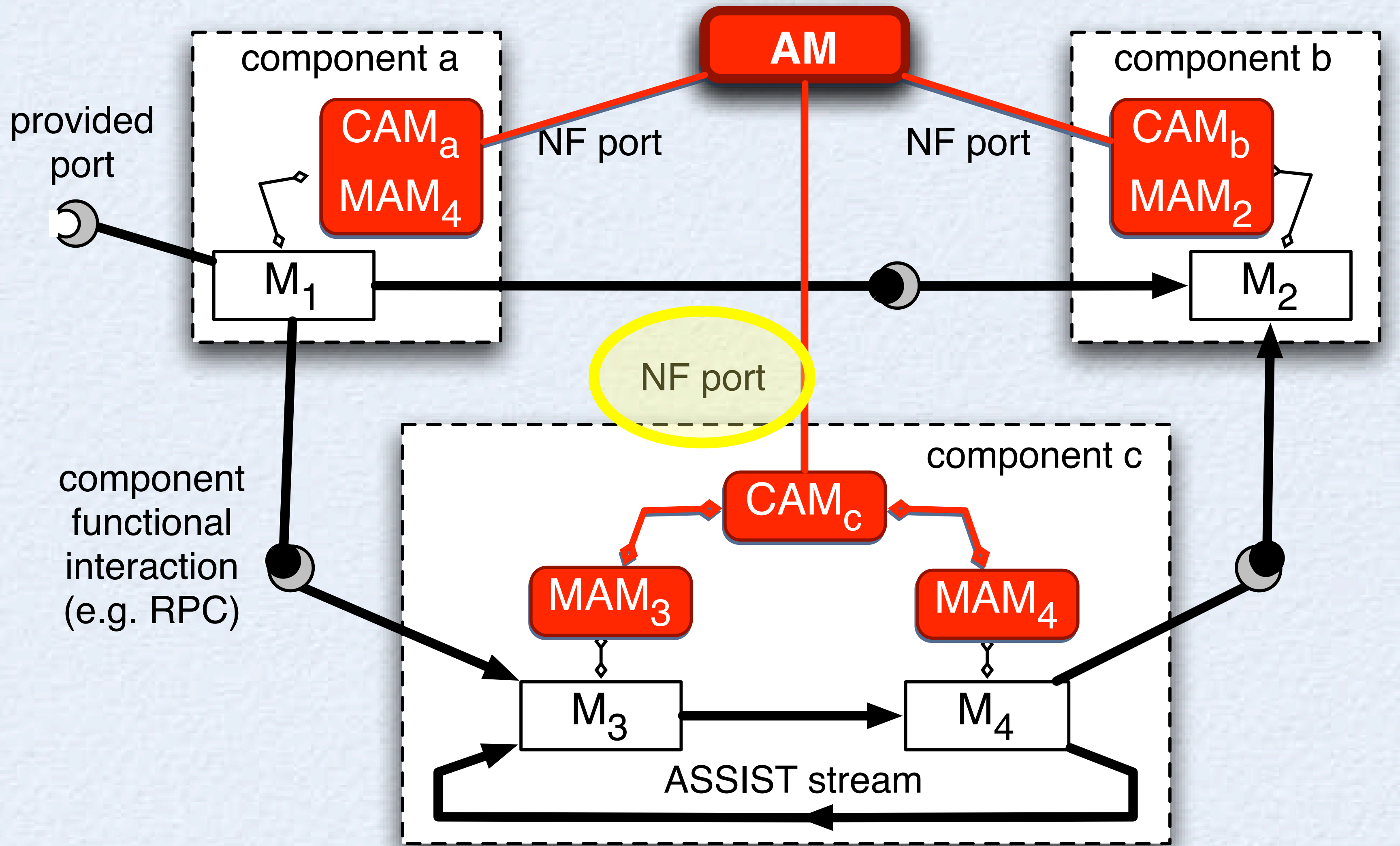
- ASSIST graphs can be enclosed in components
- they can be wired one another
- they may used to wrap sequential or parallel code (e.g. MPI)
- they can be wired to other legacy components (e.g. CCM)
- currently *native component model*, already converging in the forthcoming GCM (authors involved in CoreGRID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

managed components



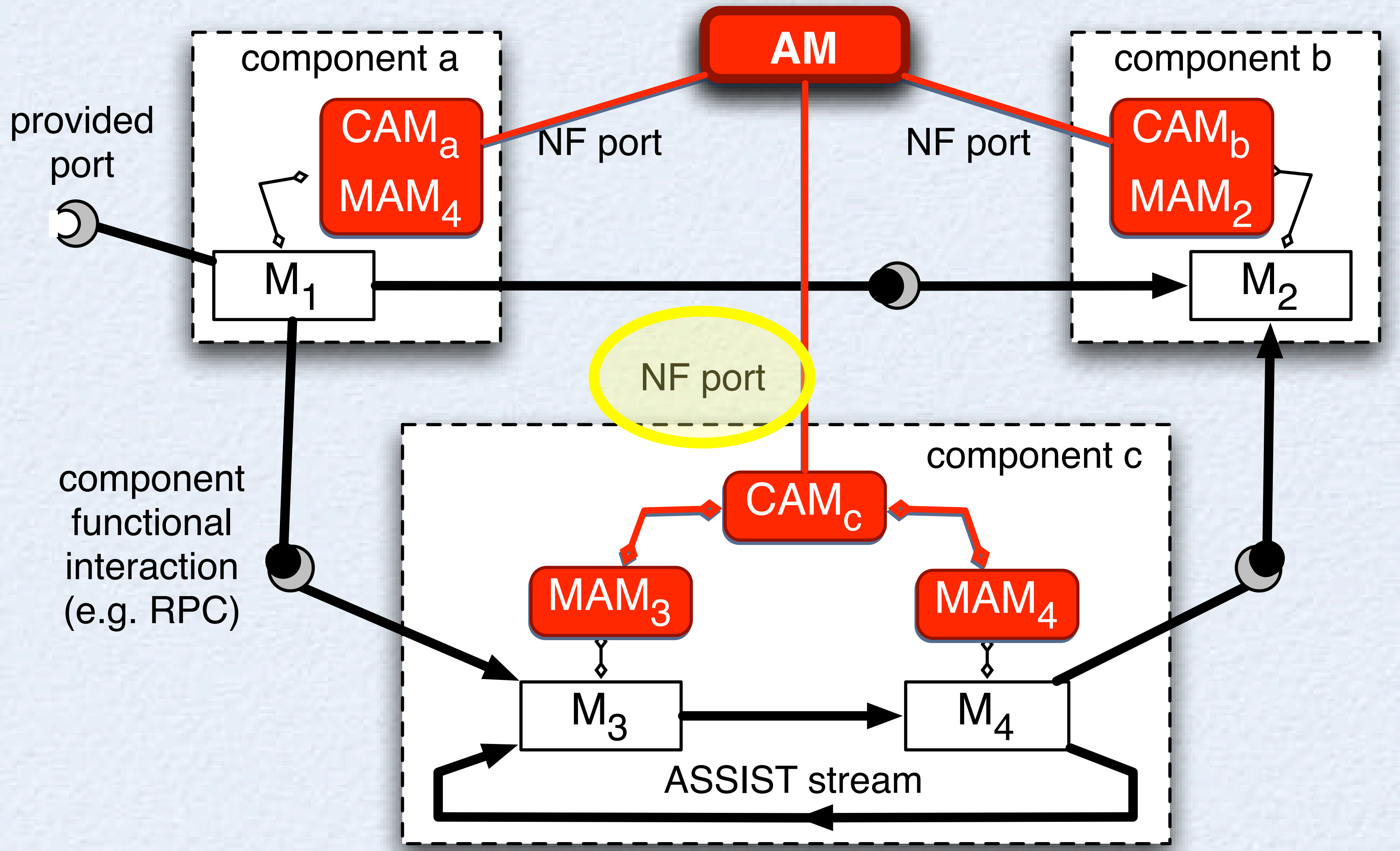
- modules and components are controlled by managers
- managers implements NF-ports

managed components



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

managed components

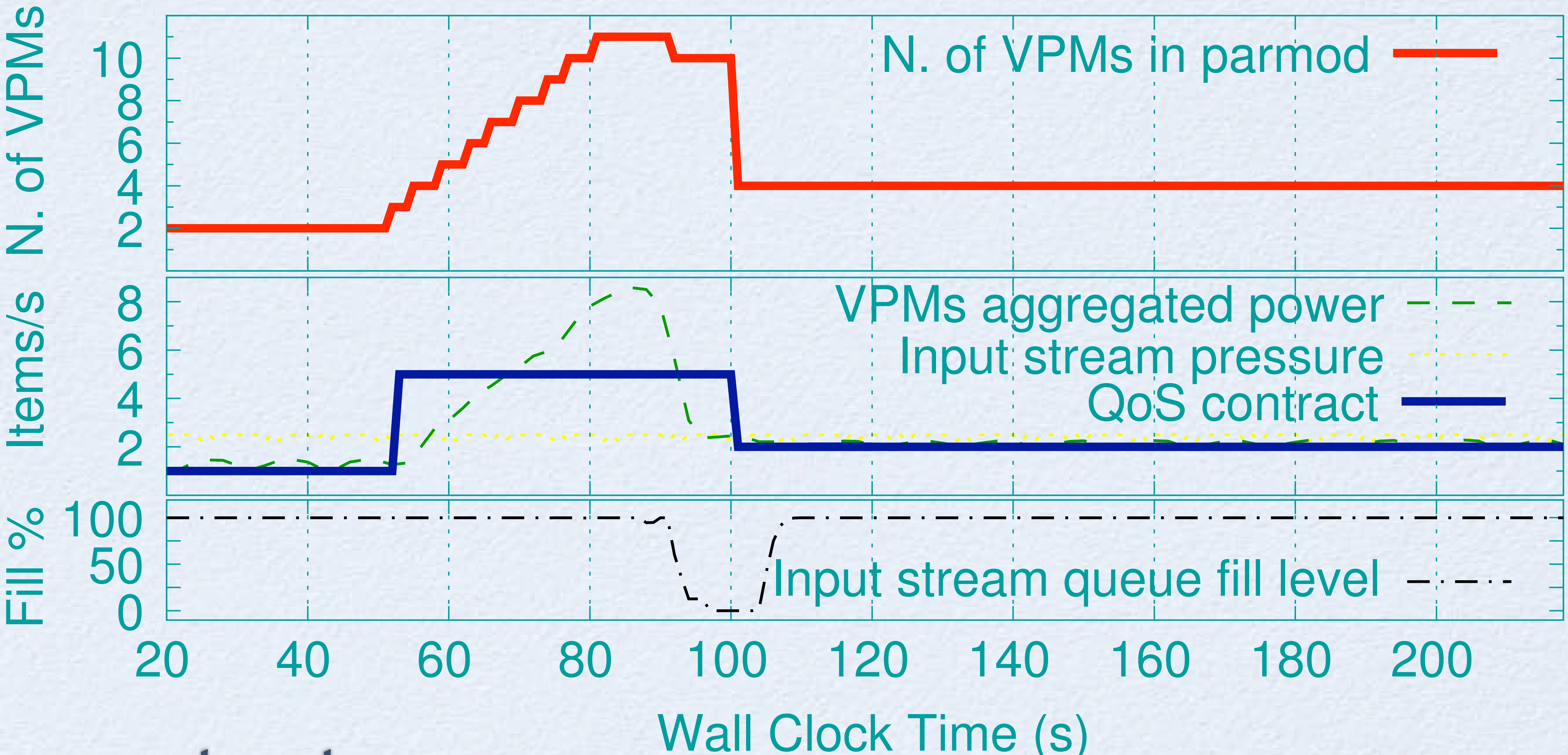


- modules and components are controlled by managers
- managers implements NF-ports
- the distributed coordination of managers enable the managing of the application as whole (the top manager being the Application Manager)

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

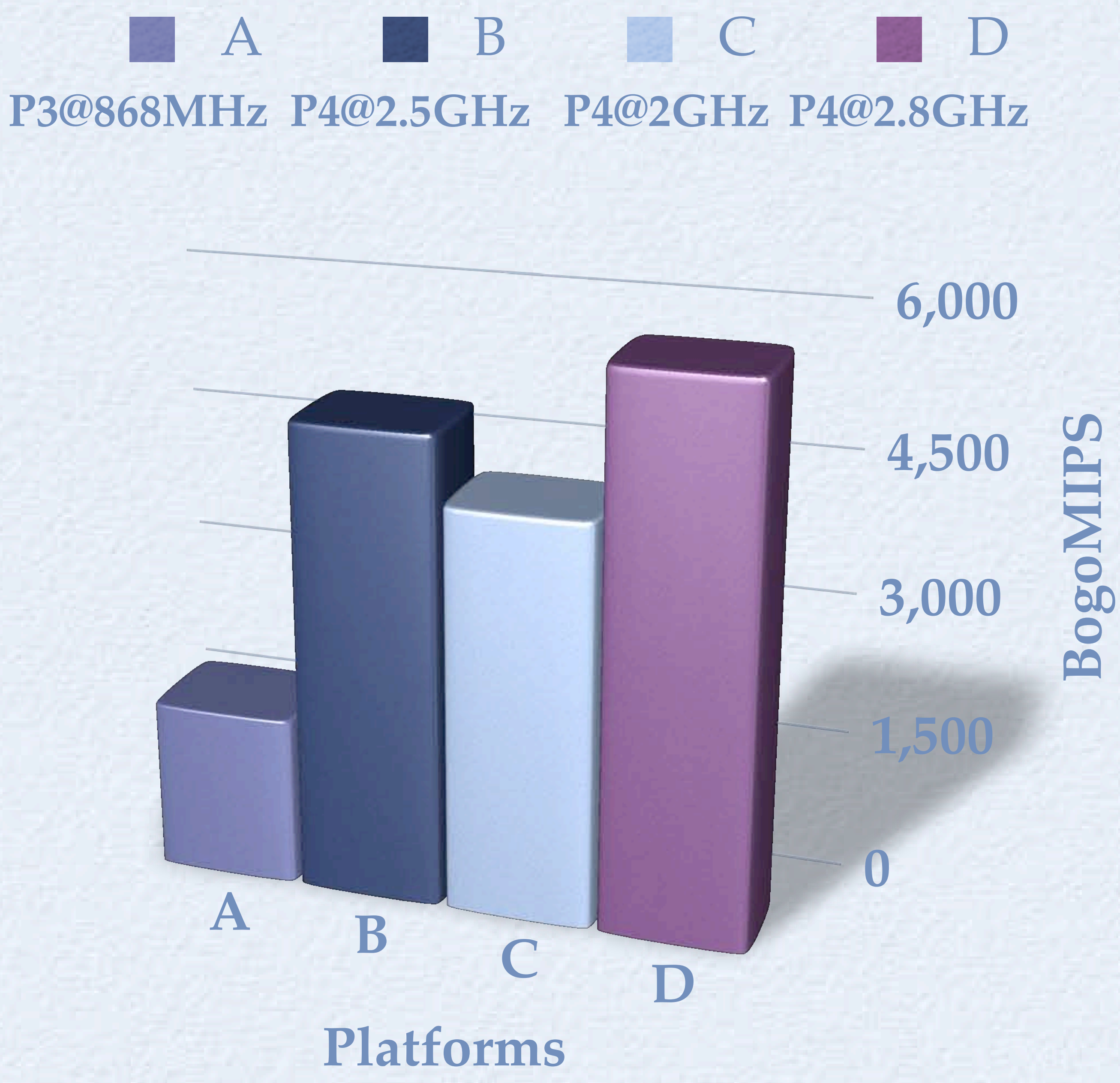
Perf. features	QL_i (input queue level), QL_o (input queue level), T_{ISM} (ISM service time), T_{OSM} (OSM service time), N_w (number of VPMs), $T_w[i]$ (VPM$_i$ avg. service time), T_p (parmod avg. service time)
Perf. model	$T_p = \max\{T_{ISM}, \sum_{i=1}^{n} T_w[i]/n, T_{OSM}\}$, $T_p < K$ (goal)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

experiment: stateless farm

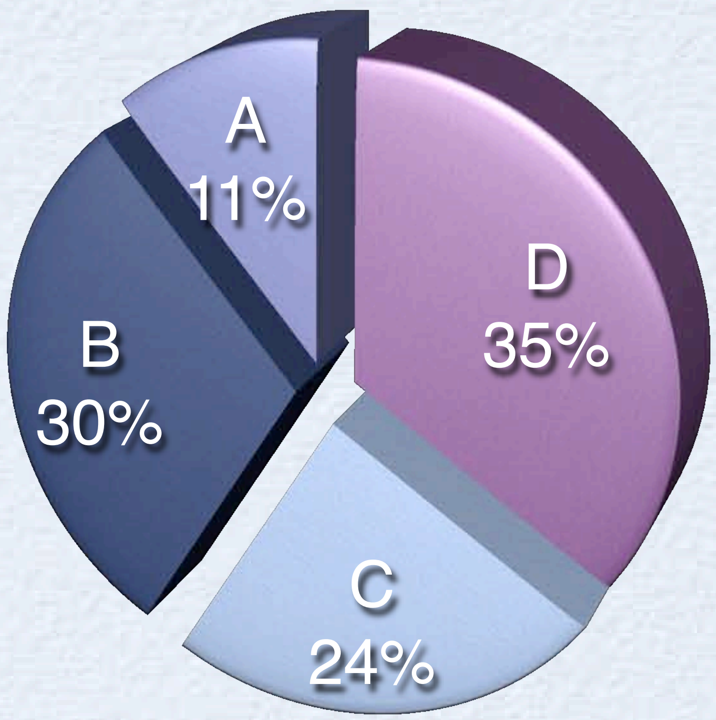


- contract:
  - keep a given service time
  - contract change along the run

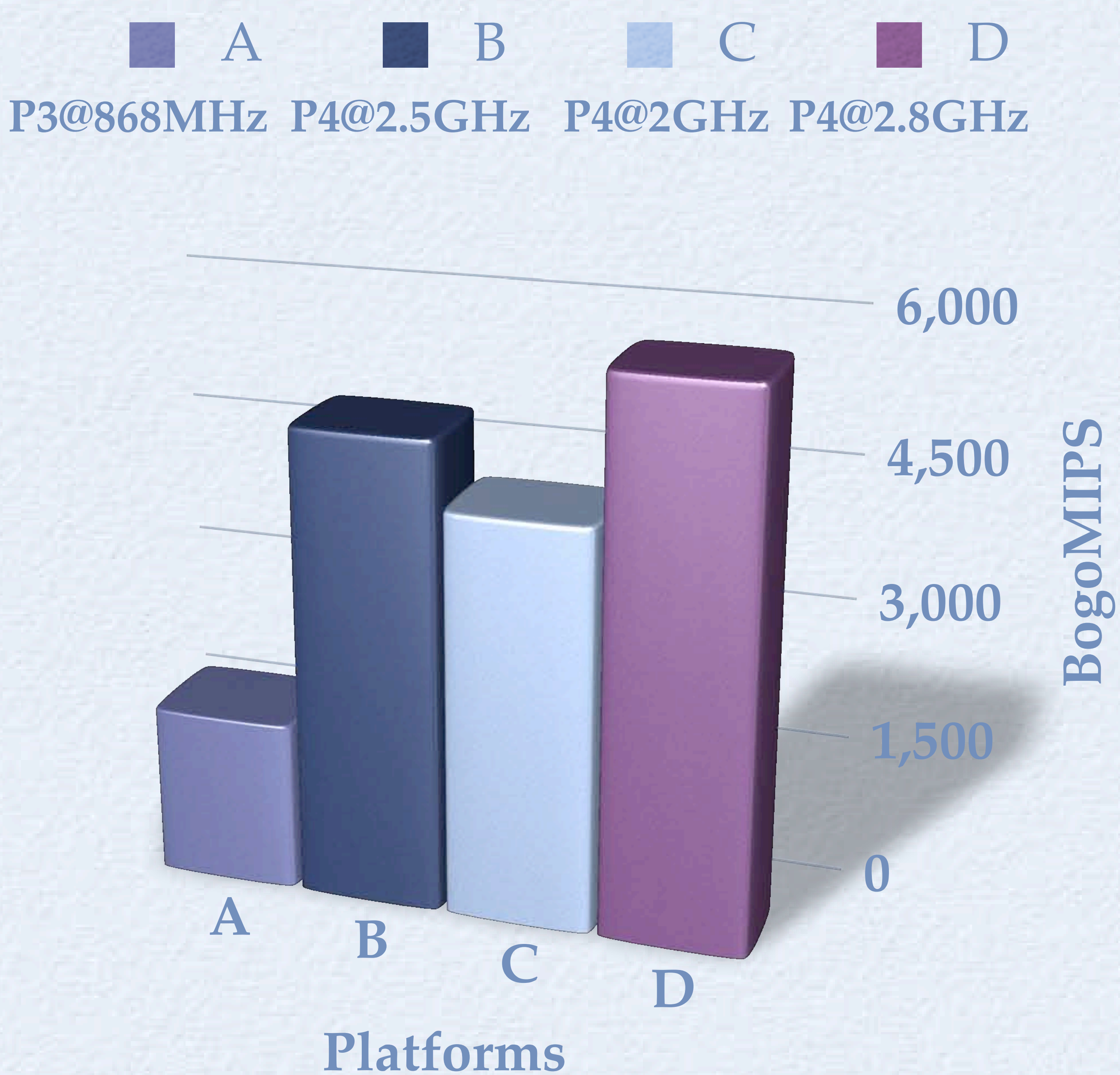
Experimenting heterogeneity



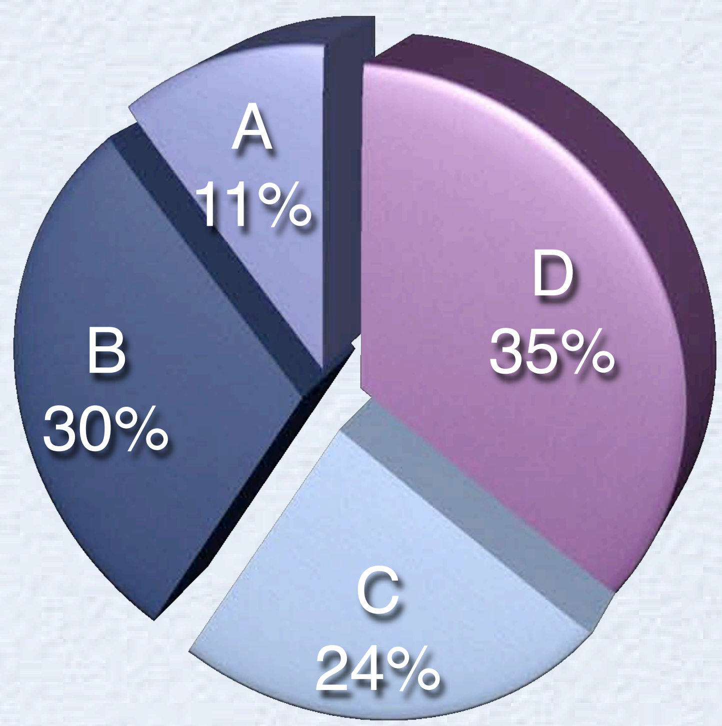
Expected work balance among platforms



Experimenting heterogeneity

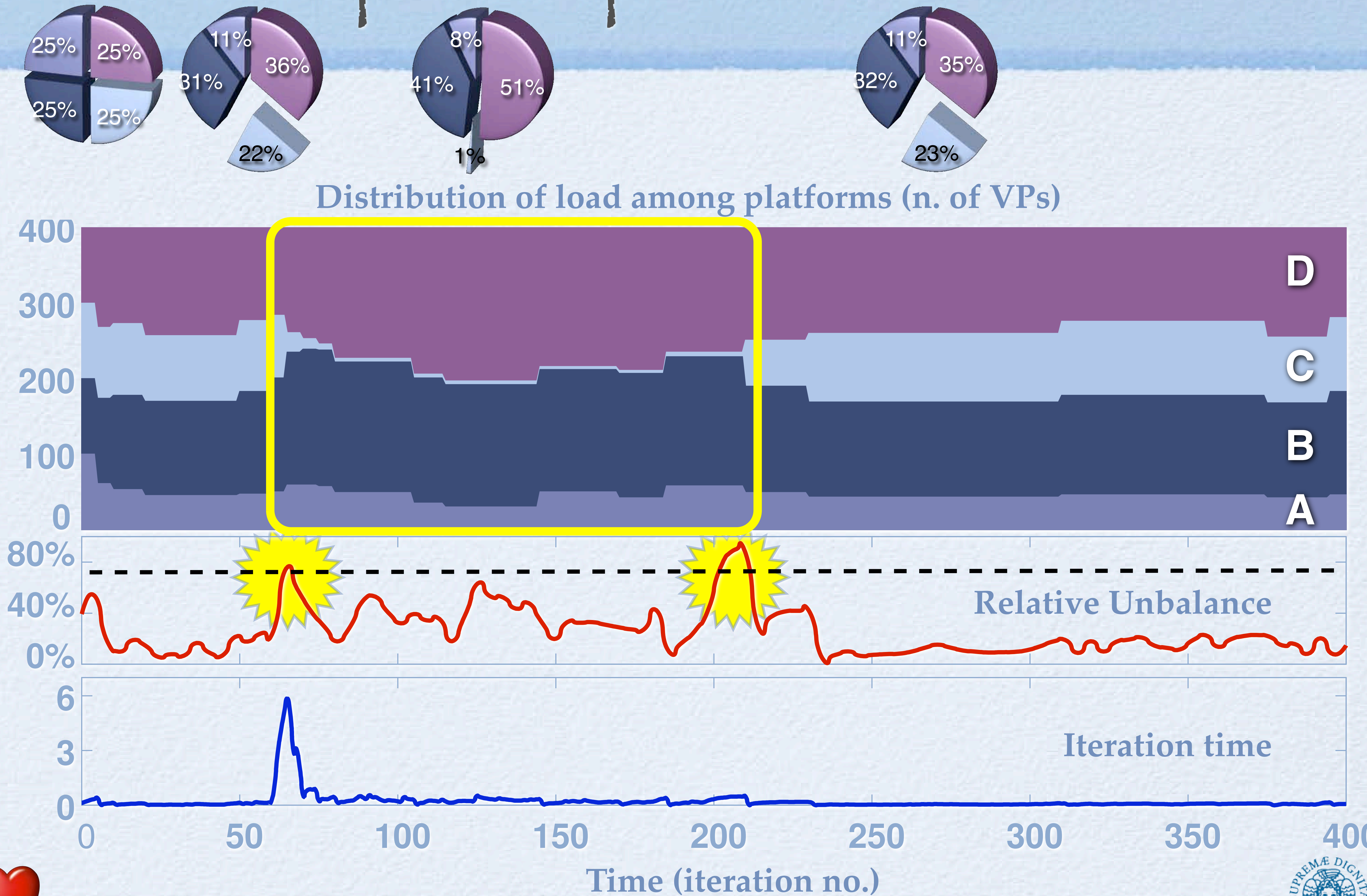


Expected work balance among platforms



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

Thank you

ASSIST is open source under GPL

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