Dagstuhl Seminar 04451 - Future Generation Grids 2004
November 4th, 2004

Rendering Grid Heterogeneity Harmless

Marco Aldinucci
ISTI - National Research Council
Pisa - Italy

with S. Campa, M. Danelutto, C. Zoccolo

Dagstuhl Seminar 04451 - Future Generation Grids 2004
November 4th, 2004

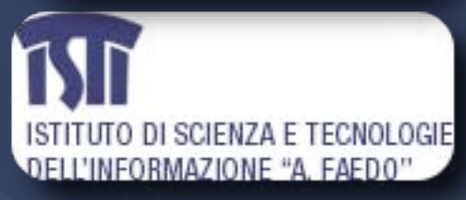
Rendering Grid *(performance)* Heterogeneity *(mostly)* Harmless

Marco Aldinucci

ISTI - National Research Council

Pisa - Italy

with S. Campa, M. Danelutto, C. Zoccolo



Outline

- Motivation
- Grid as collection of heterogeneous resources
  - Presenting experimental results
  - A simple, even simplistic model
  - Defining the asymptotic performance
- Detect Grid current status and react
- Re-distributing work & load through WS

Motivation

- Researchers in the Grid community hardly agree
  - programming model (and either if it should exists)
  - components (and either if they are an useful vehicle)
  - legacy code existence …
- but them all agree
  - **THE GRID IS A HIGHLY HETEROGENEOUS, HIGHLY DYNAMIC EXECUTION ENVIRONMENT**

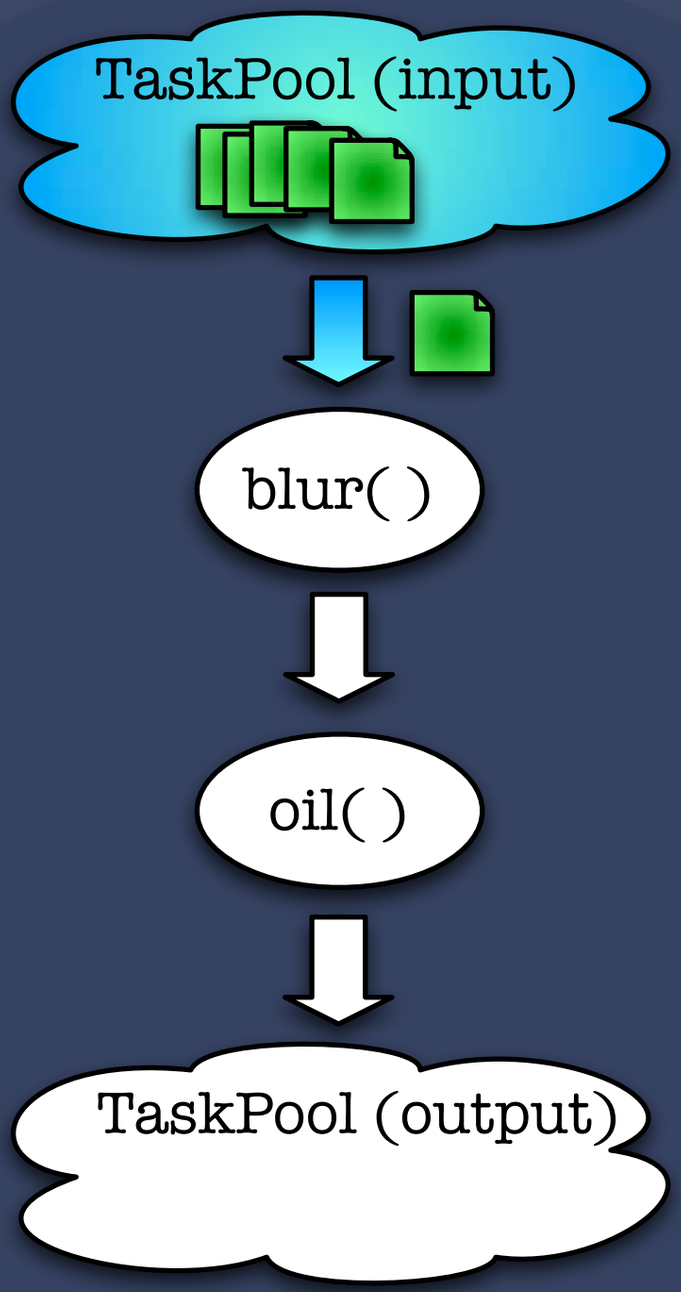
However …

	Experimental results on >2 PEs	Performance figures for >2 heterogeneous PEs
EuroPar 04 (Grid & P2P) LNCS 3149	2–4/20	1? (as far I known)
Grid Computing 04 LNCS 3165	3/30	No (as far I known)
Grid & Cooperative Computing 04 LNCS 3251	6/150	No (as far I known)

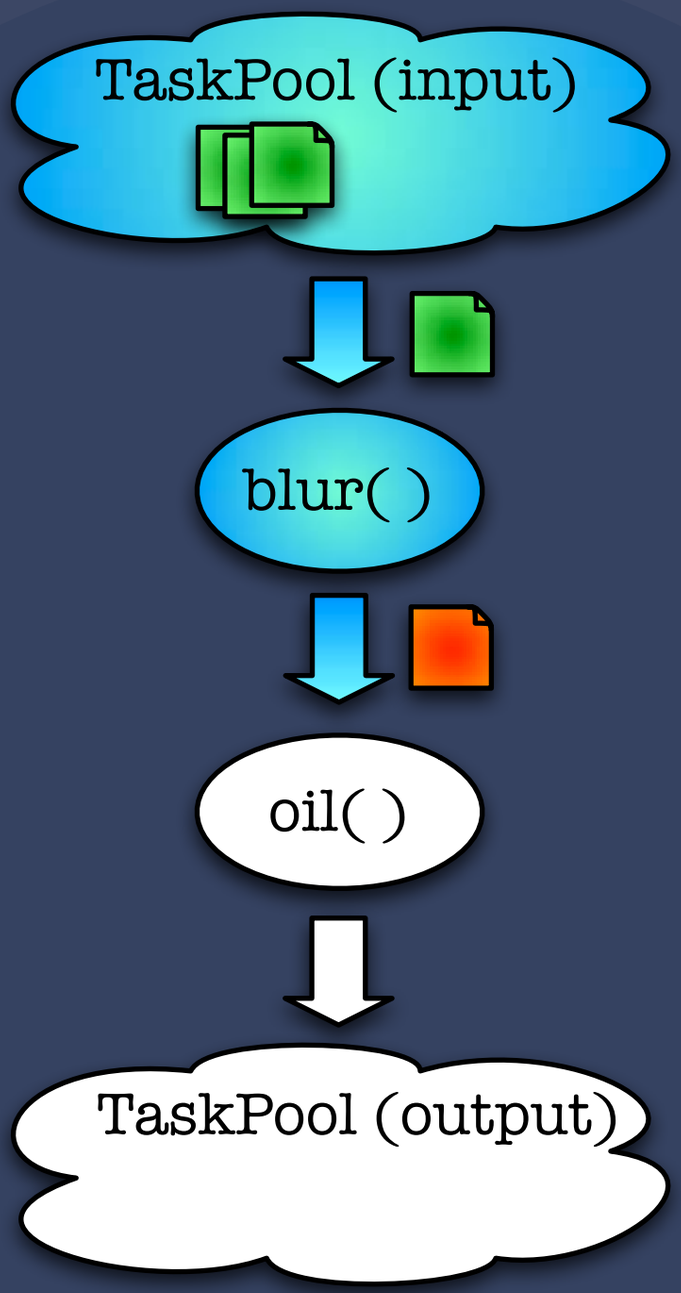
- How many platforms GTx supports?
- Java seems to be the panacea for heterogeneity:
  - Maybe we relying too much on Sun's researchers
- Look at conference proceedings:
  - few of them present experimental results
  - very few of them present result for heterogeneous environments
- we agreed on heterogeneity, thought

Testbed:

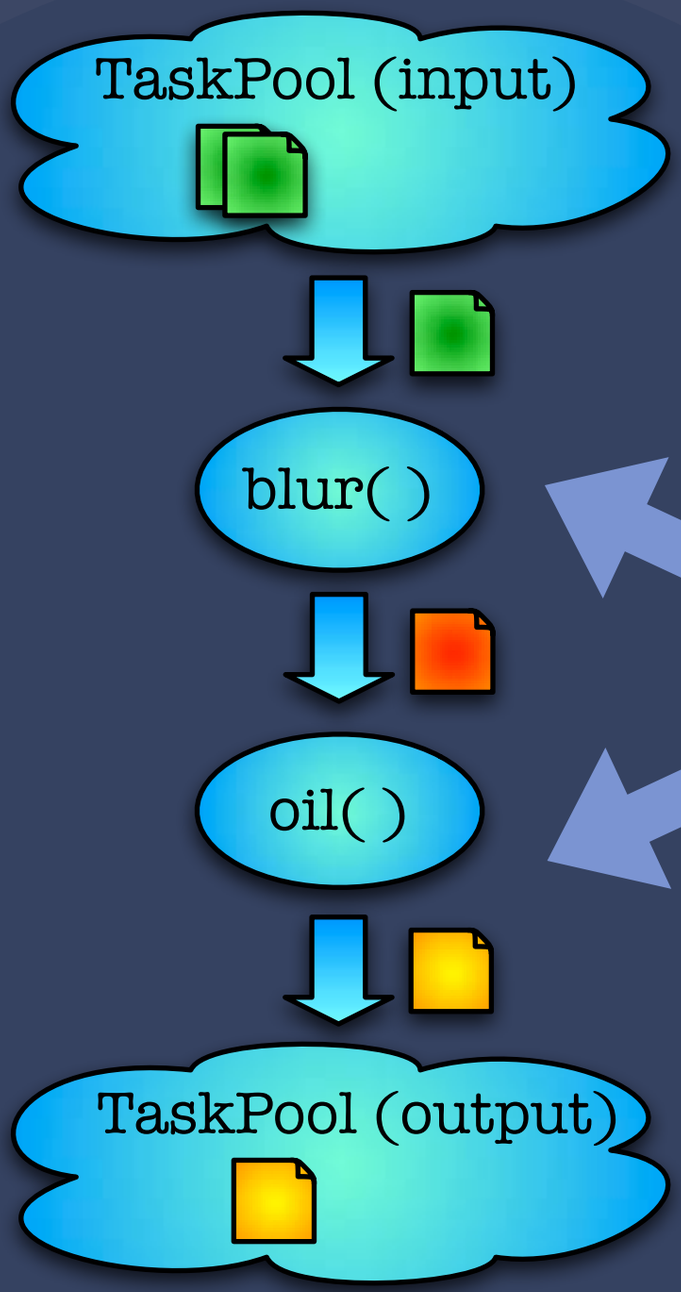
```
wait(4*365*24*60*60);
unfortunately();
```



blur( ) it



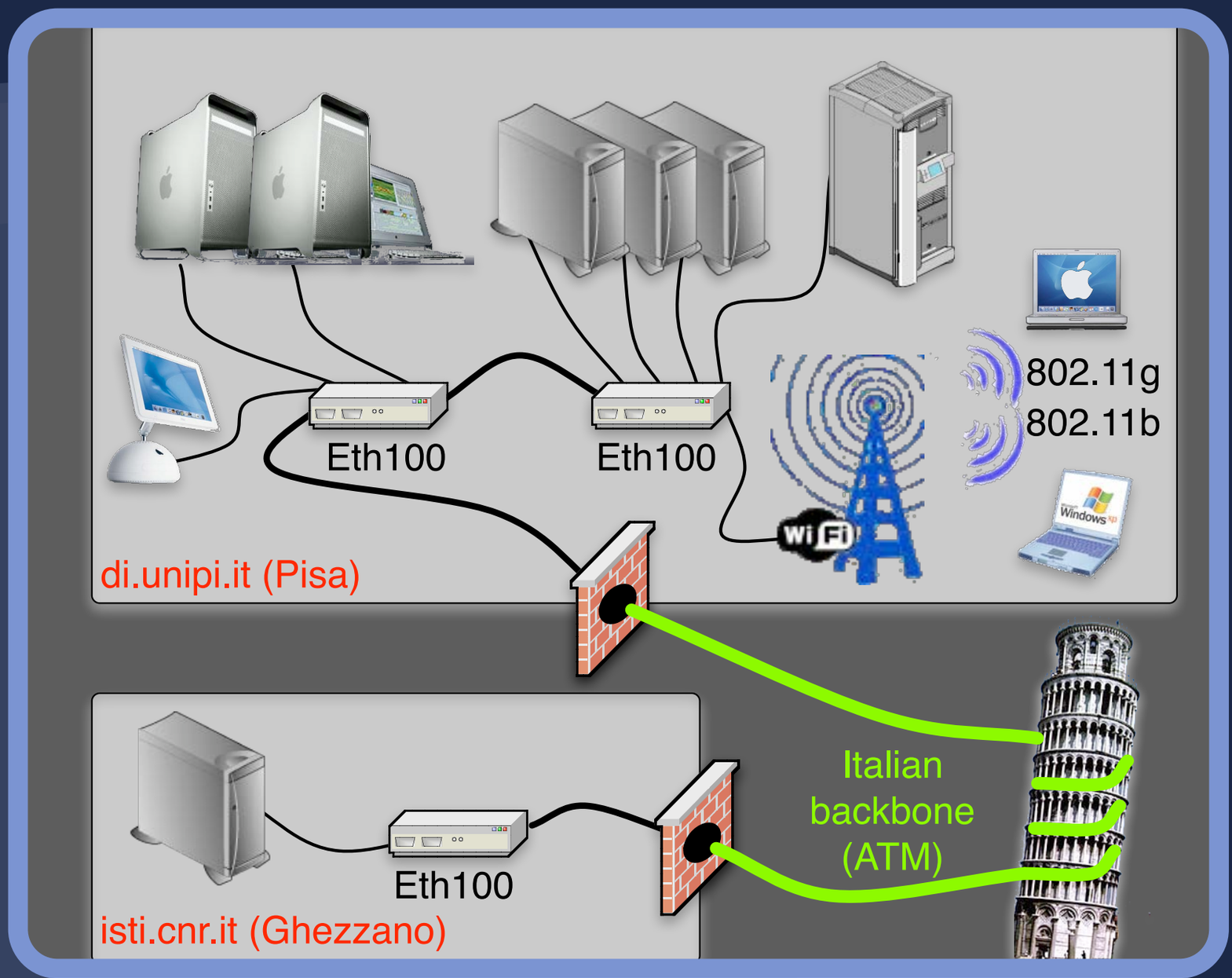
oil( ) it



These can be parallel as well

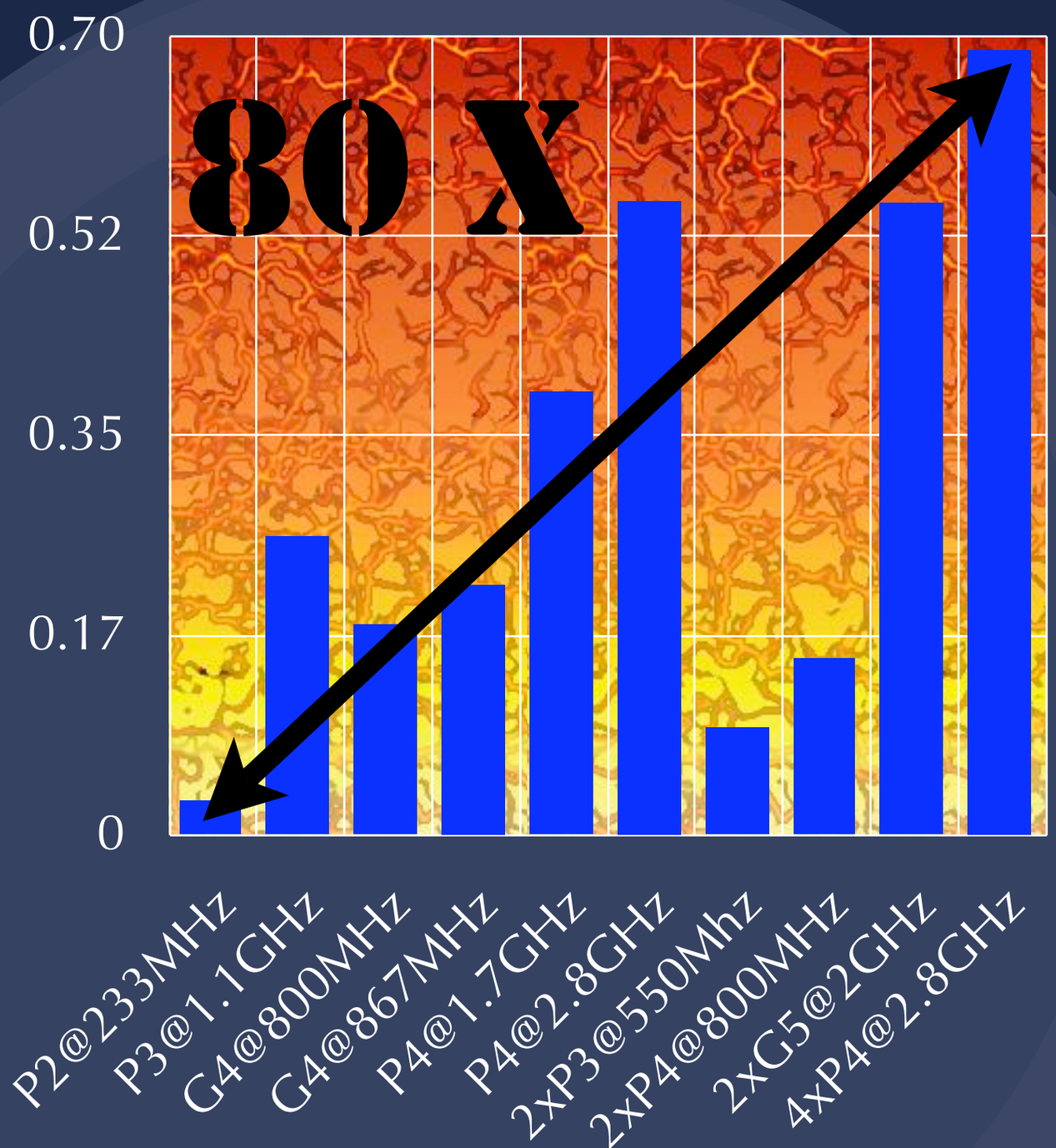
Why speedup is important

- We would like deploy HPC applications on Grid
  - not just seti@home
  - they may have time/performance/memory/... critical requirements
- Known in advance what I can expect from my run, at least as asymptotically optimal curve
  - speedup for example (widely used in COW)
  - any measure able to give informations on the *quality* of the algorithm, implementation, configuration, ...



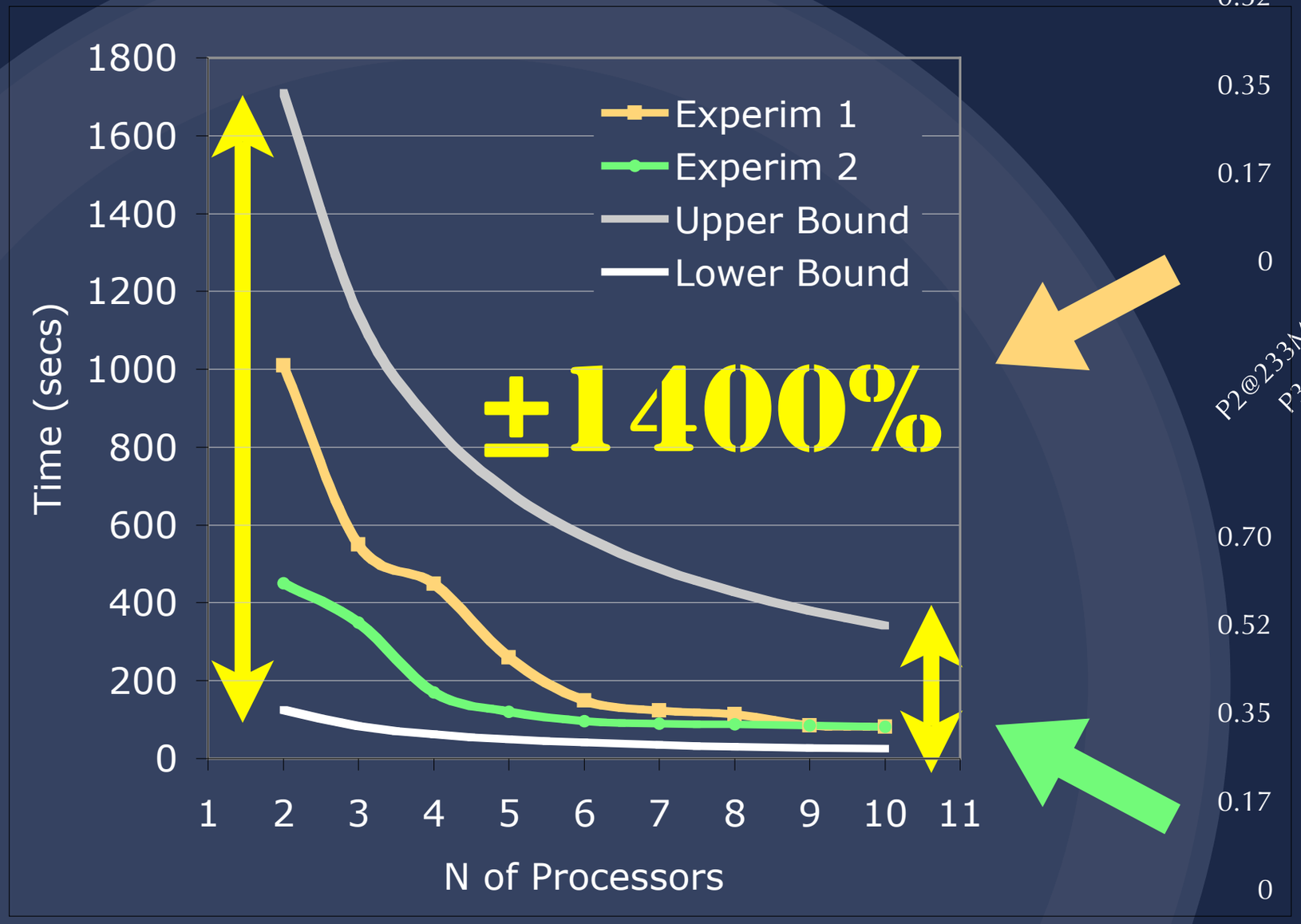
Experimental env: a home-made Grid

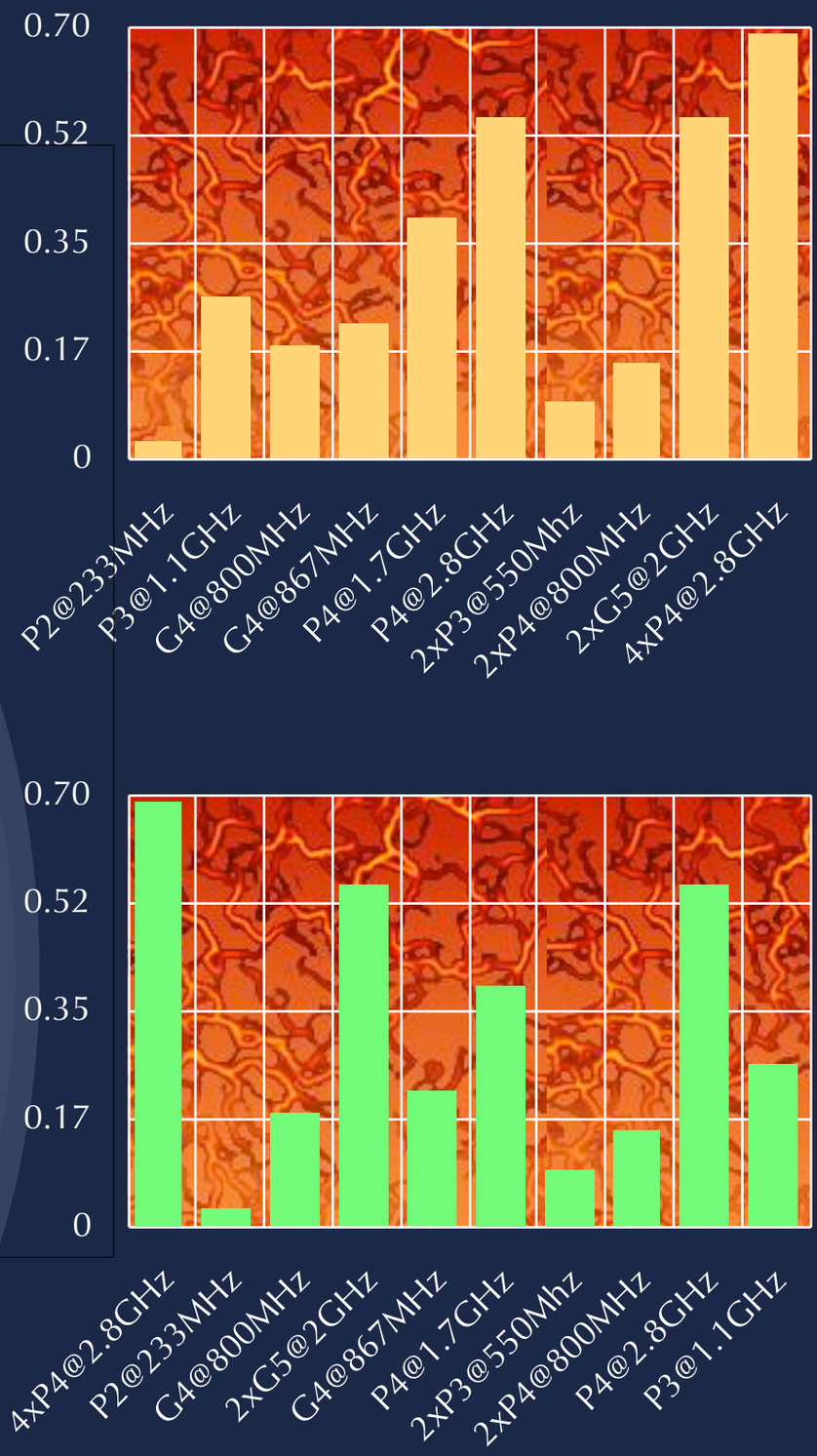
BogoPower



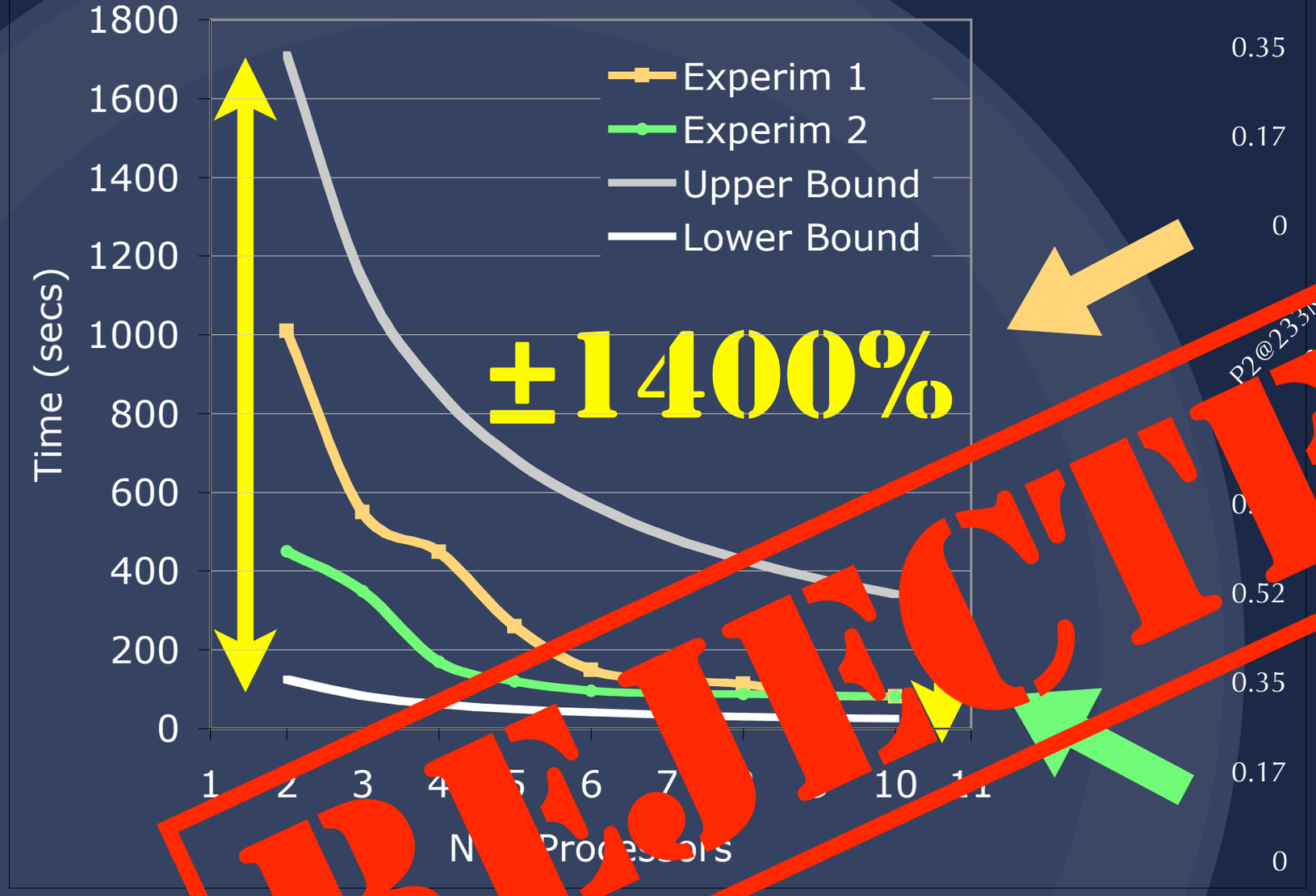
- BogoPower:
  - Models machine power on (tasks/sec) on a single PE
  - neglect net performance
- What speedup means in this scenario?
  - another metric is needed …

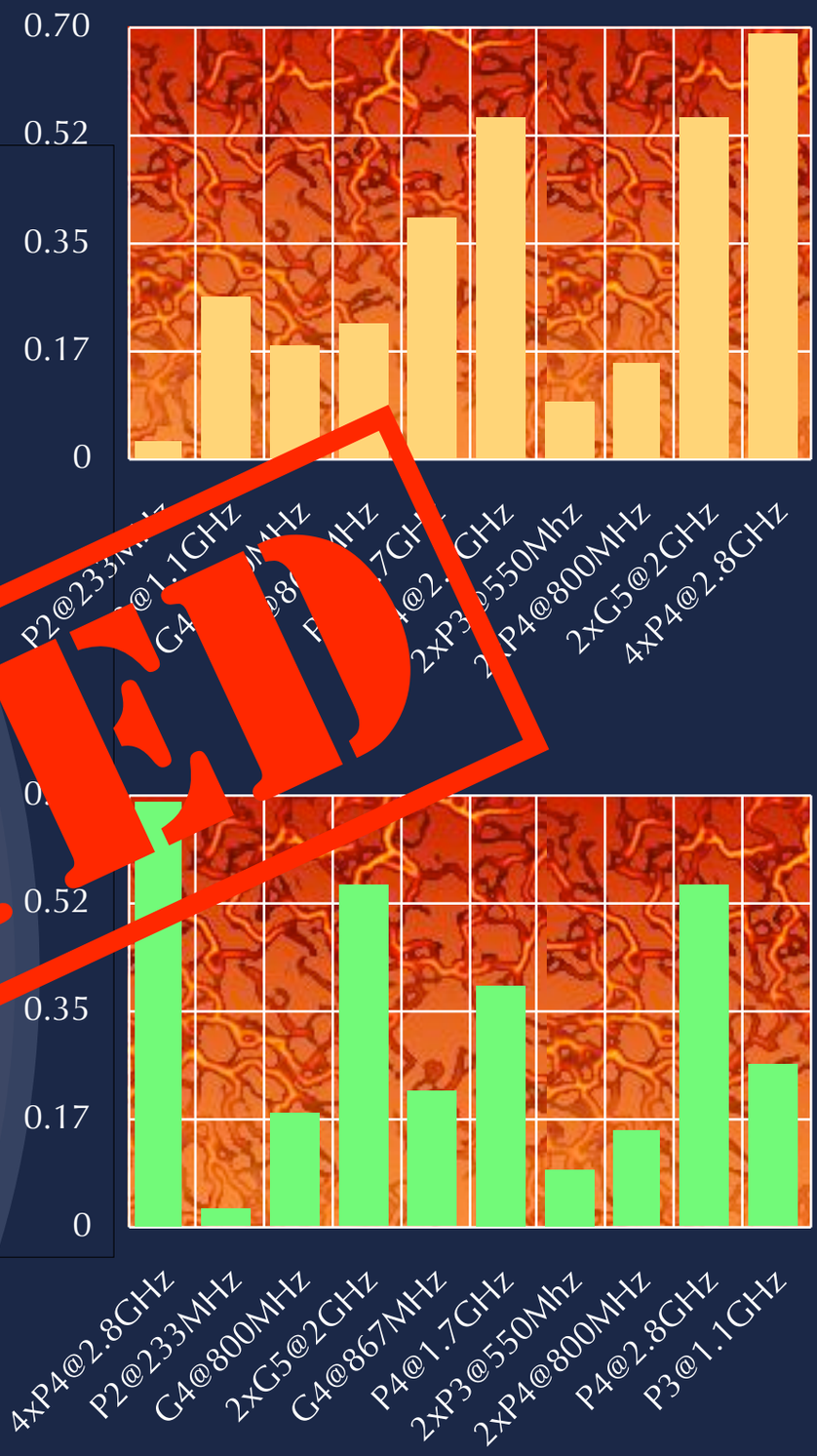
Two experiments





Two experiments





REJECTED

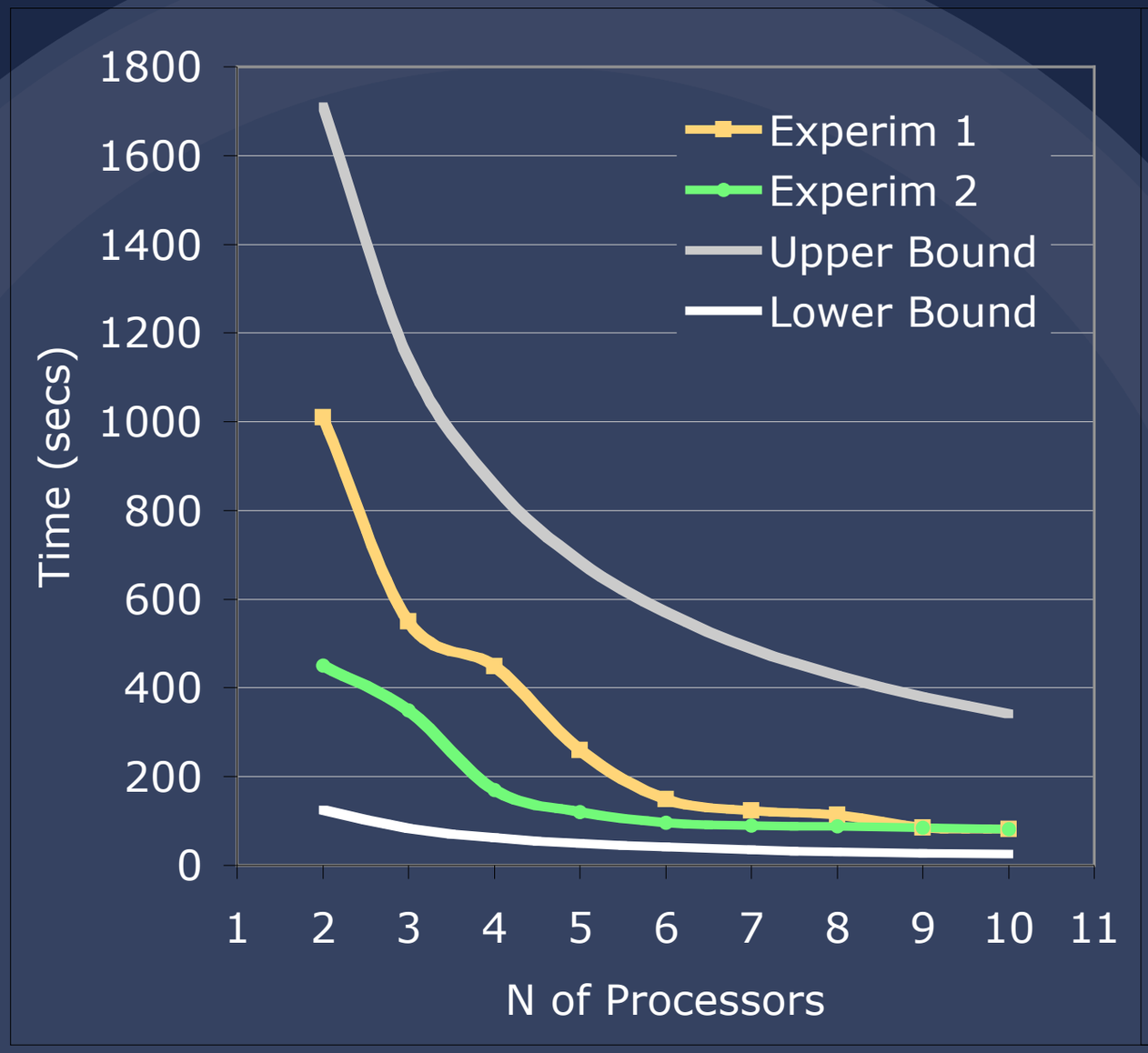
Speedup … ?


10
8
6
4
2
0
Ideal
1 2 3 6 7 8 9 10
45
36
27
13
0
Ideal
1 2 3 4 5 6 7 8 9 10
REJECTED

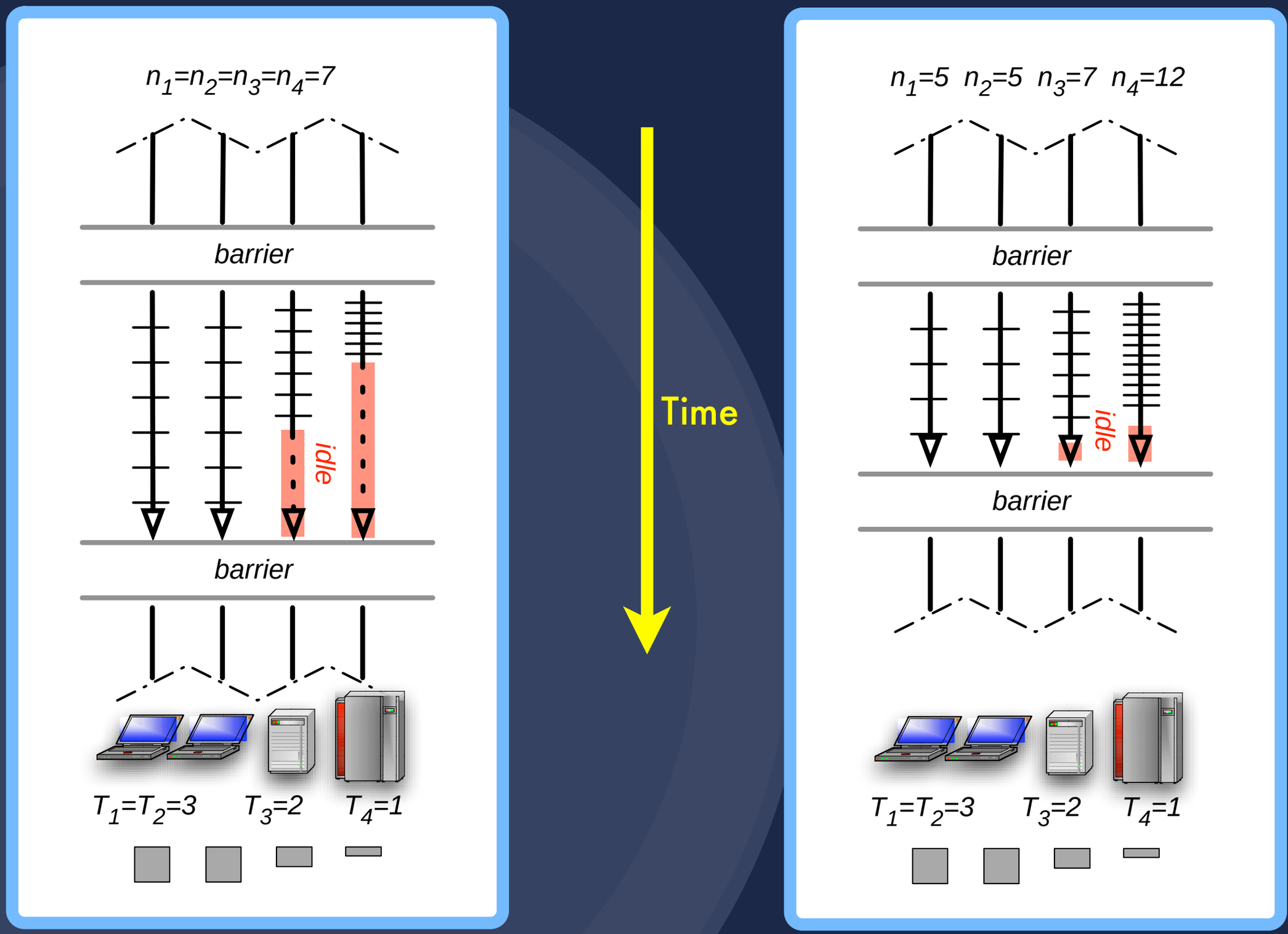
As simple as speedup

- Speedup does not give any information
  - does not provide any reference curve, i.e. an upper bound for algorithm and implementation quality
- It can be replaced with another simple measure
  - with the same features in order to keep the intuition
  - suitable for heterogeneous (in power) envs
- BogoPower can be used (sometime)

Two experiments revised



Naive scheduling (and not)



Describing sub-optimal Performance

- Suppose to have an idea of the performance T (time) of a given task T on a given platform
  - i.e. platform BogoPower - it maybe figured out from any suitable measure of performance, e.g. GridBench, GGF BenchGroup, ...
  - if task haven't constant time consider the average of a bulk of tasks
  - dynamically adapt knowledge through monitoring, adjusted by current load
- compute a scheduling, miming on-demand policy
  - that is sub-optimal, but easy to compute, to understand and to present as "ideal" performance in a paper

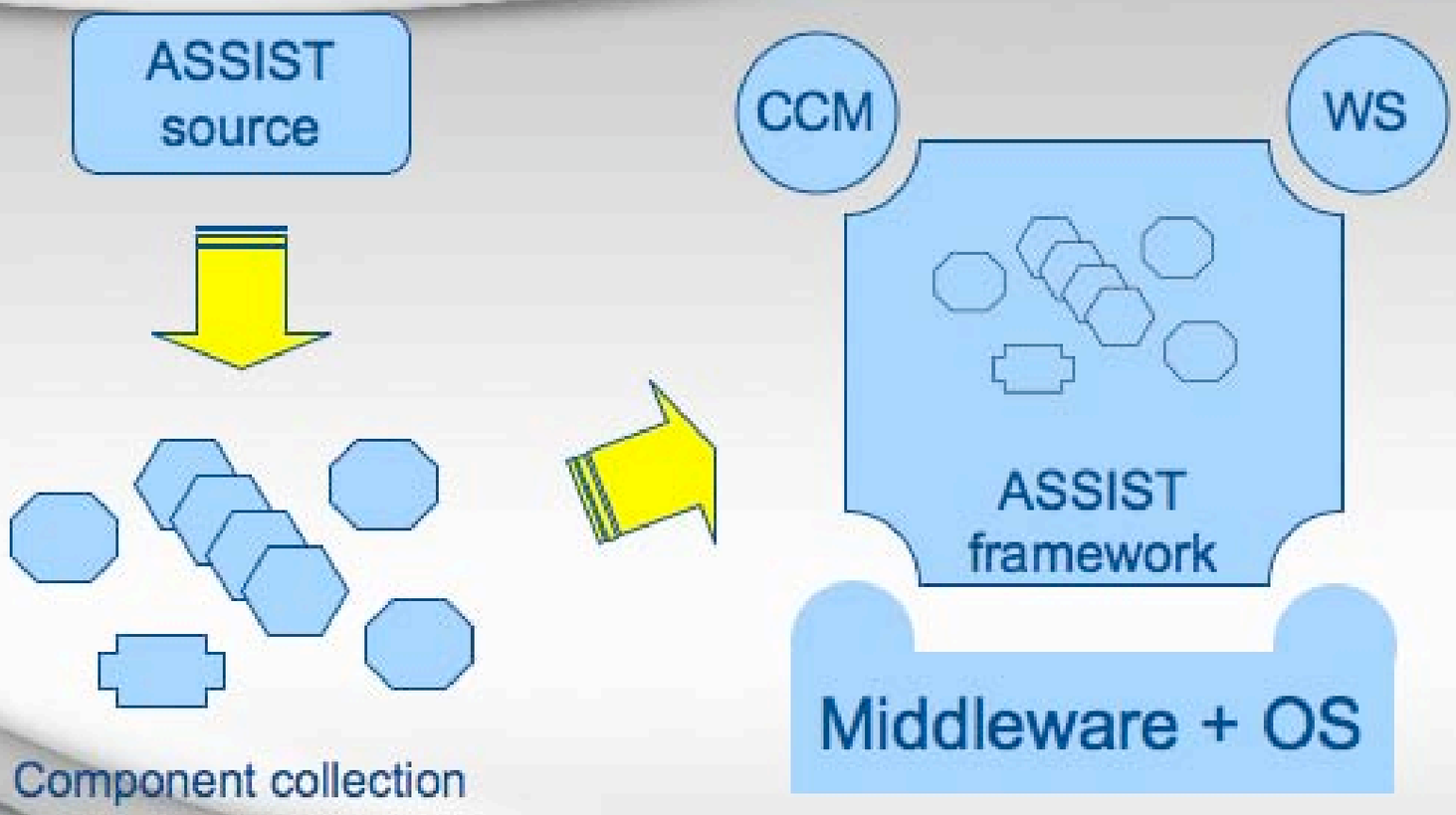
$$n_i = \frac{N \; H(T_1, \cdots, T_n)}{n \; T_i} \quad \begin{cases} N = \# \text{ of tasks} \\ H = \text{Harmonic Mean} \\ T_i = \text{Time for 1 task on } PE_i \\ n_i = \text{optimal number of tasks for } PE_i \end{cases}$$

Outline

- Motivation
- Grid as collection of heterogeneous resources
- Detect Grid current status and react
  - The ASSIST framework
  - A service to find them, a GTx to bring them all and in the darkness bind them, a model to rule them all …
- Re-distributing work & load through WS

It is not a joke,
it is *e-fantasy* !

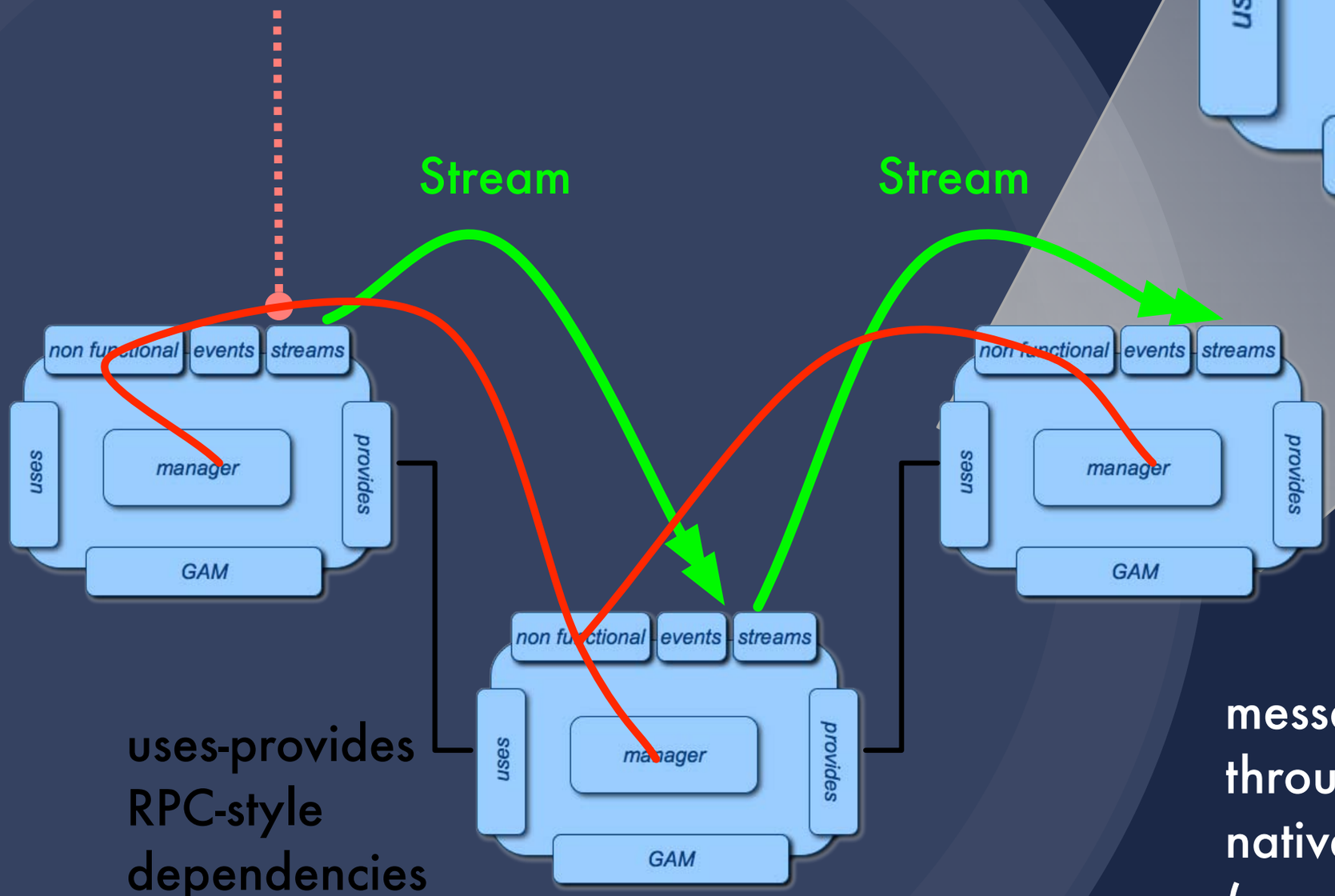
The ASSIST way



from Danelutto's yesterday talk

Application Manager

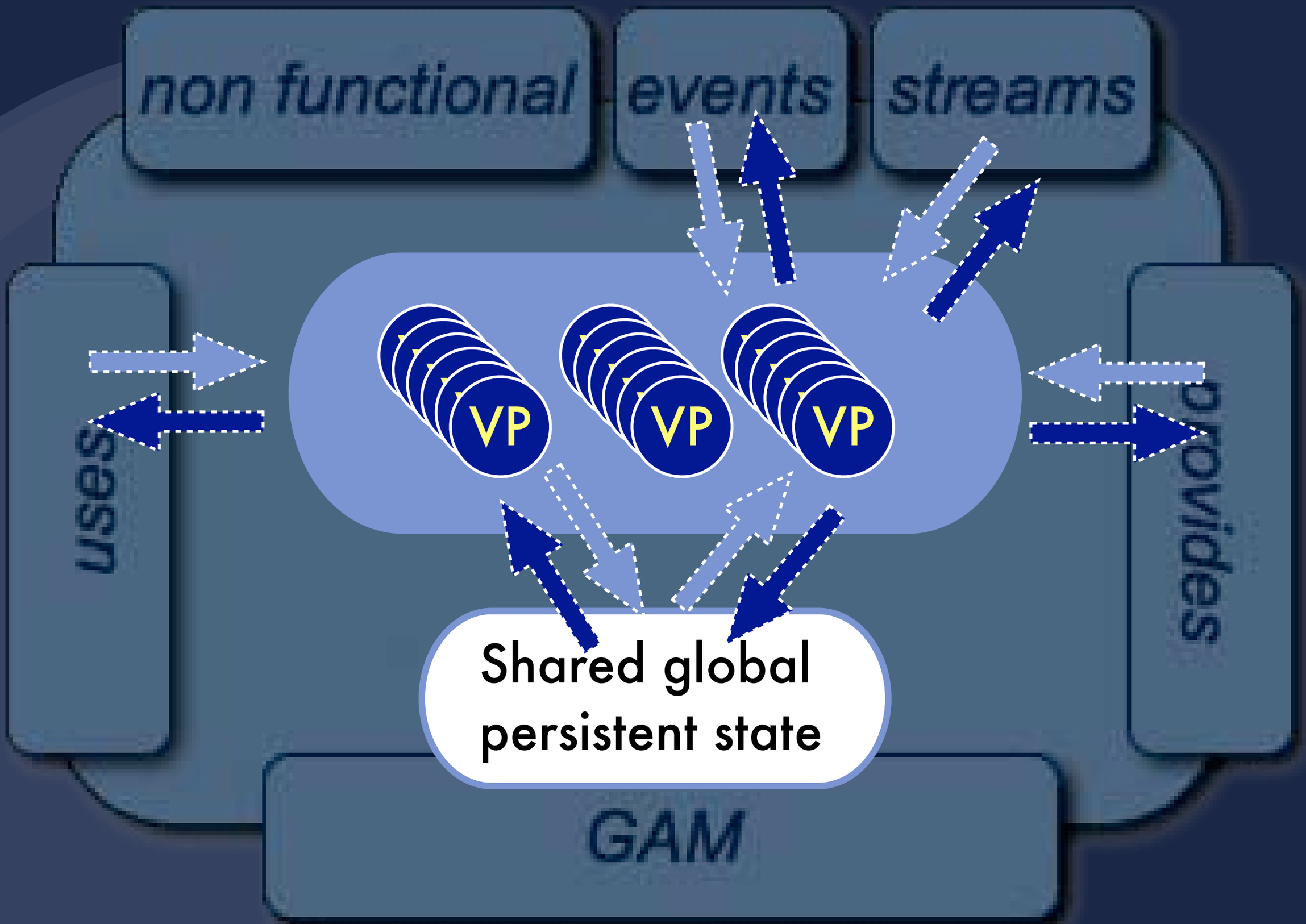
Managers may interact through non functional interfaces realizing a global, distributed control for the application.
I.e. the **Application Manager** (AM)



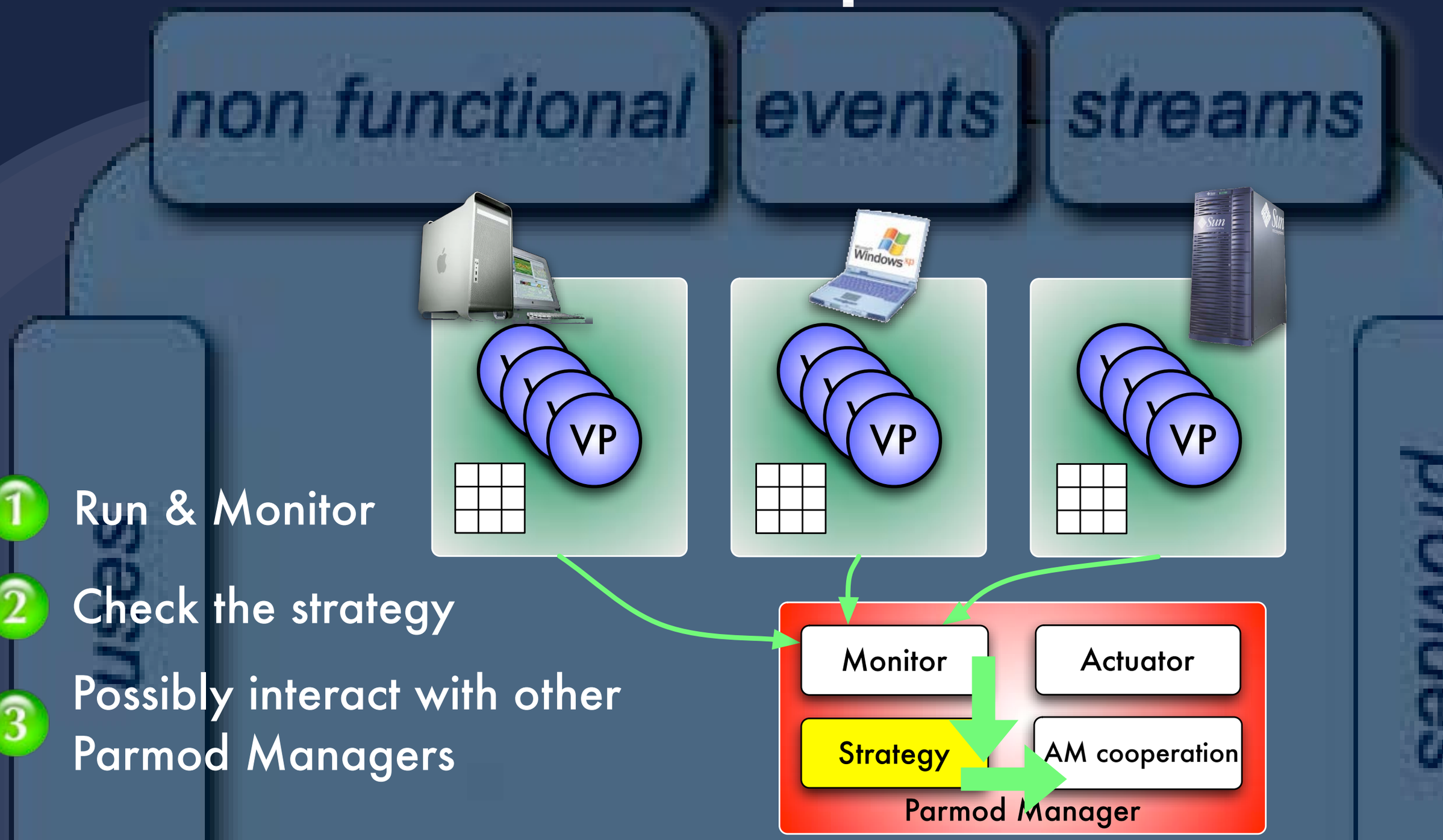
uses-provides
RPC-style
dependencies

messages may flow
through different media:
native, WS/SOAP, Corba streams
(compiler provides the full support)

Parmod component

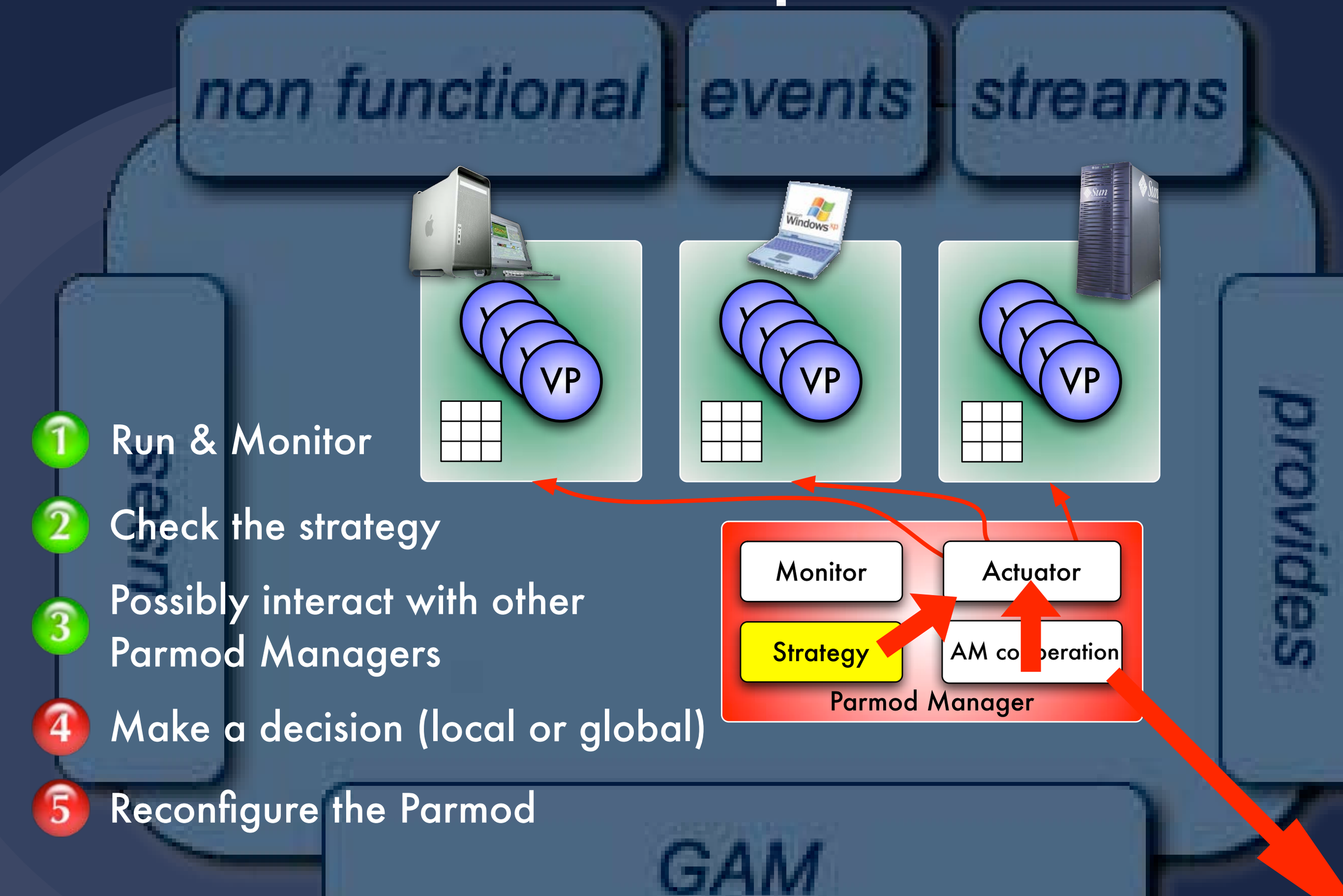


Parmod component



1. Run & Monitor
2. Check the strategy
3. Possibly interact with other Parmod Managers

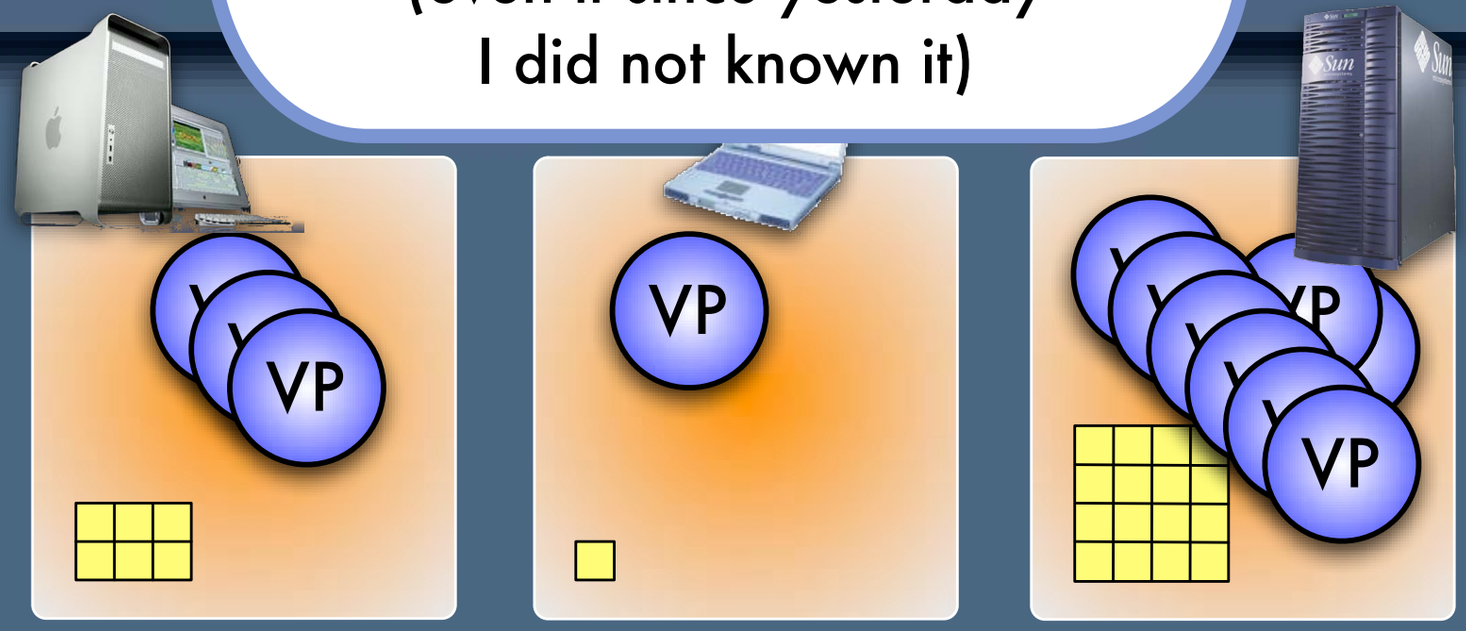
Parmod component

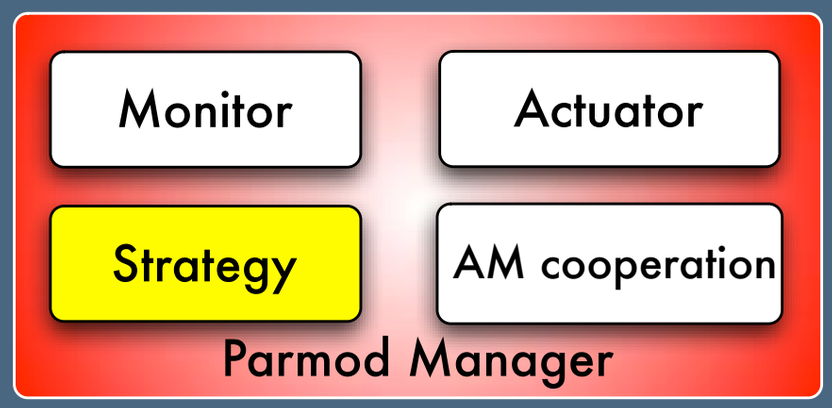


1. Run & Monitor
2. Check the strategy
3. Possibly interact with other Parmod Managers
4. Make a decision (local or global)
5. Reconfigure the Parmod

Parmod component

This maybe is
e-autonomous-computing
(even if since yesterday
I did not known it)



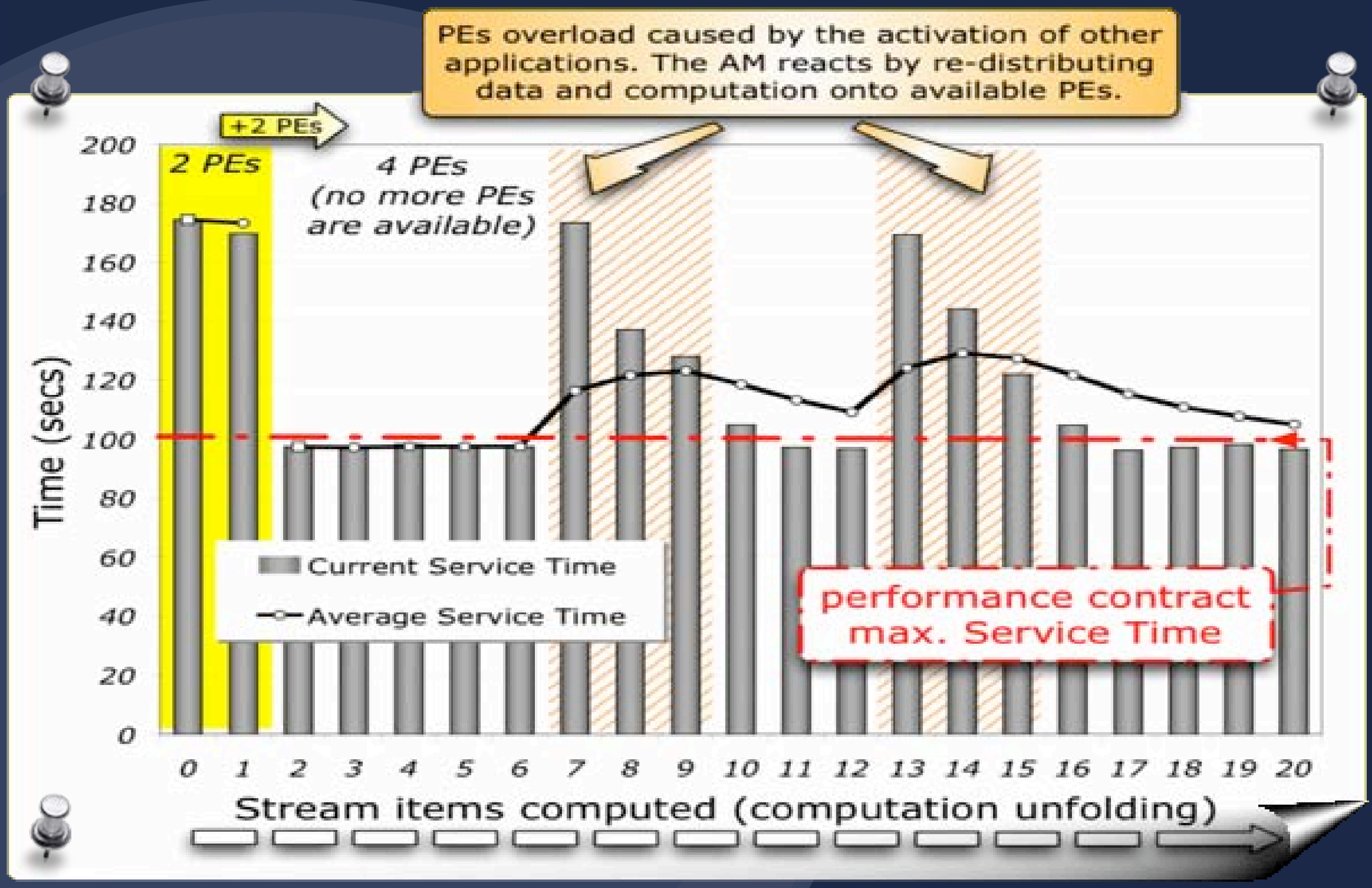


1. Run & Monitor
2. Check the strategy
3. Possibly interact with other Parmod Managers
4. Make a decision (local or global)
5. Reconfigure the Parmod
6. Parmod reconfigured !

Two key issues

2. Check the strategy
   - simulate possible scenarios by using the suitable model
     - respect the performance contract: service time, resources, …
   - e.g. the one I've presented seems quite efficient for HPF "do parallel" or BSP style computations
   - we already working to support other paradigms
5. Reconfigure the parmod
   - keep the shared state in a "storage component" that is distributed, persistent, WS accessible, high-performance
   - E.g. HOC / WS-HOC already available as part of ASSIST

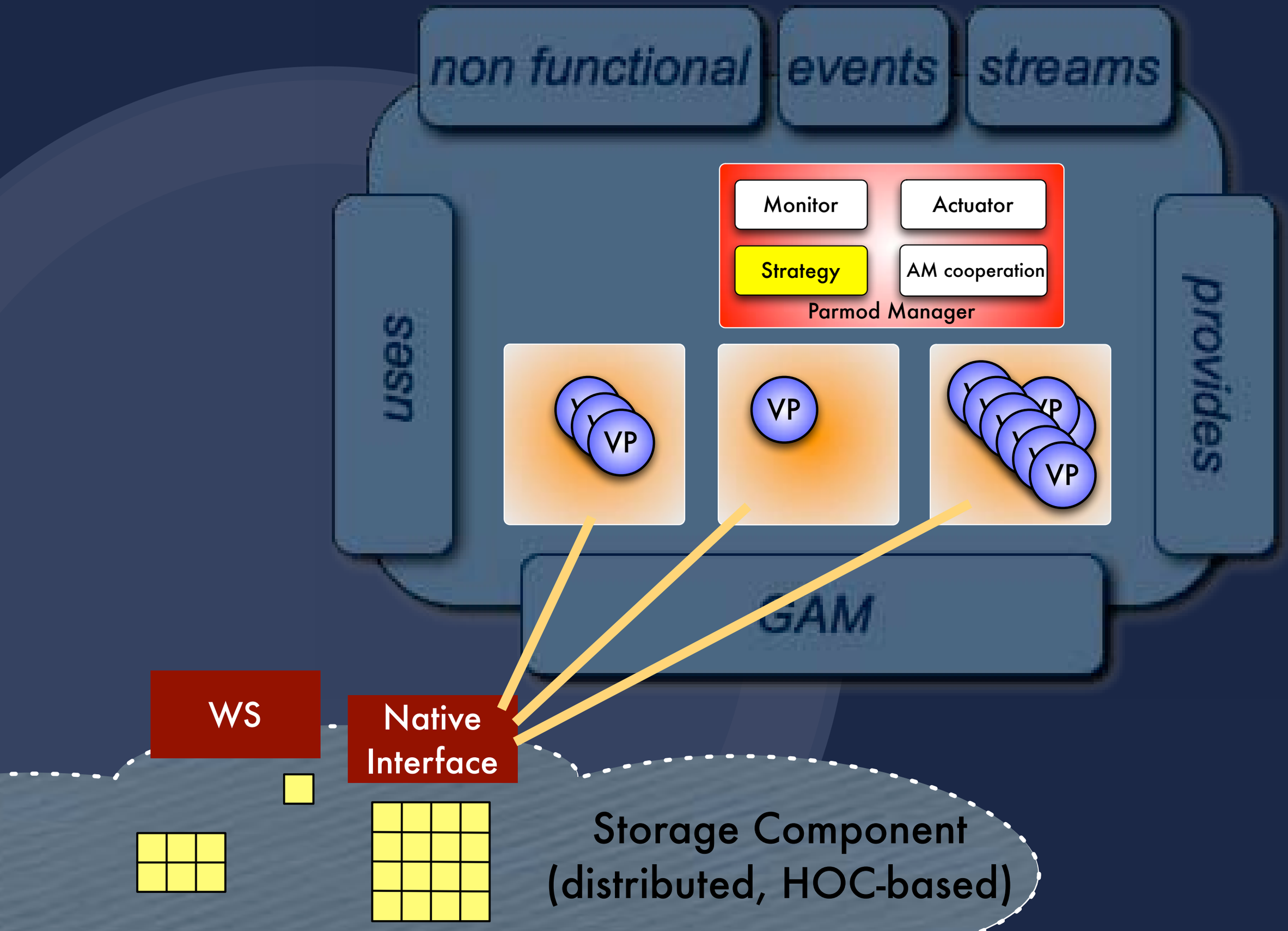
… and it work


PEs overload caused by the activation of other applications. The AM reacts by re-distributing data and computation onto available PEs.
+2 PEs
2 PEs
4 PEs
(no more PEs are available)
Time (secs)
200
180
160
140
120
100
80
60
40
20
0
Current Service Time
Average Service Time
performance contract
max. Service Time
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Stream items computed (computation unfolding)

Outline

- Motivation
- Grid as collection of heterogeneous resources
- Detect Grid current status and react
- Re-distributing work & load through WS
  - decouple management of data e computation
  - the “storage component” idea

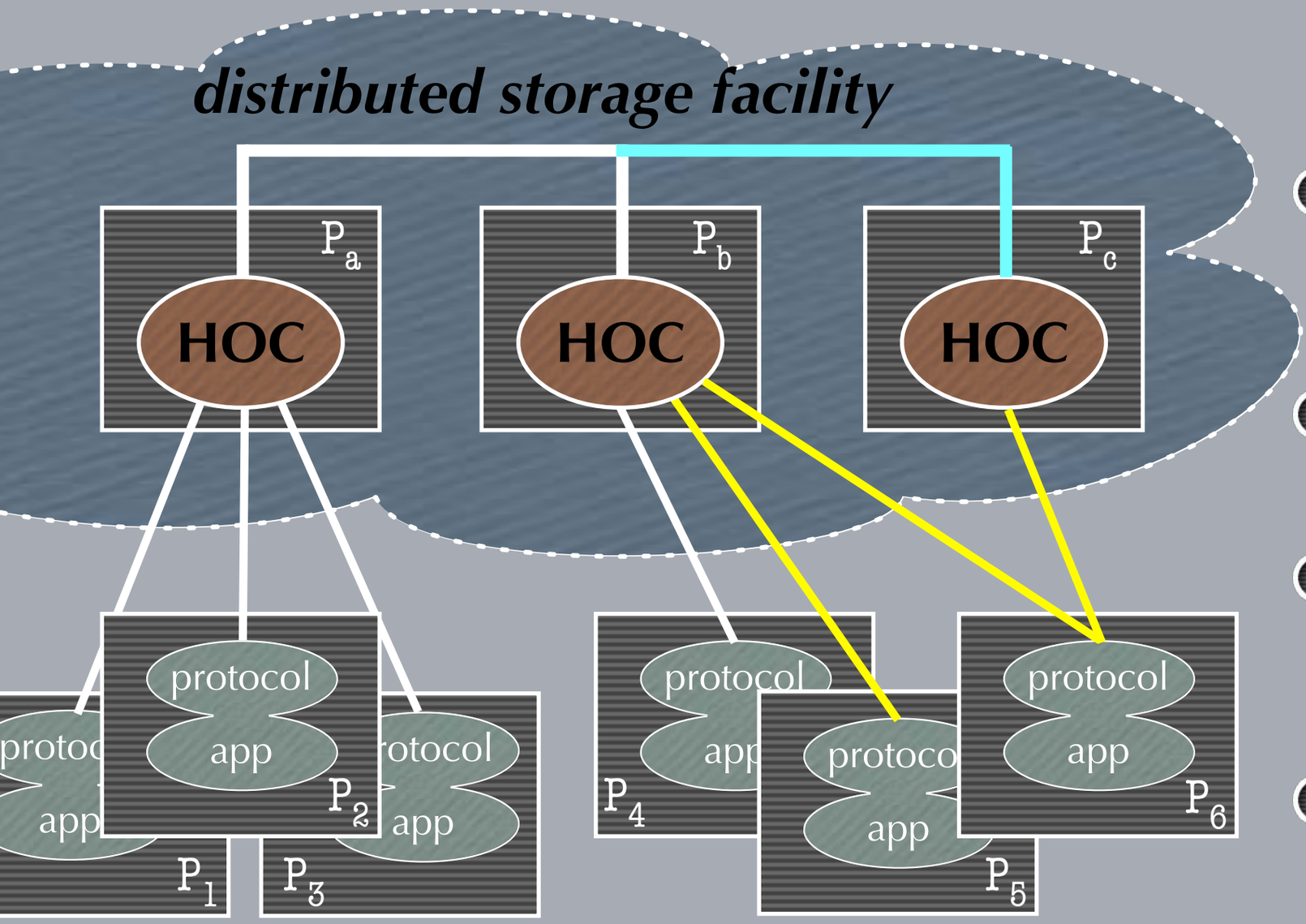
Redistribute data



HOC (Herd of Object Caches)

- A very basic storage facility
  - No hardwired policies for deployment, allocation, data coherence, ...
  - pluggable into different, third-party applications/frameworks
- proving ***data management*** as external service for applications
  - implemented as high-throughput distributed server
- decoupling computational and storage management in (distributed) application design
  - enforcing a structured development
- and exploiting persistency, scalability, re-configurability

Permanent, shared storage facility



- a facility (distributed server) providing permanent, shared storage to apps (clients)
- clients may dynamically join/ leave the storage facility
- HOC set may be hotly enlarged/ reduced on need - storage room change accordingly
- interaction with HOCs may be delegated to application-specific protocol

Why using HOC

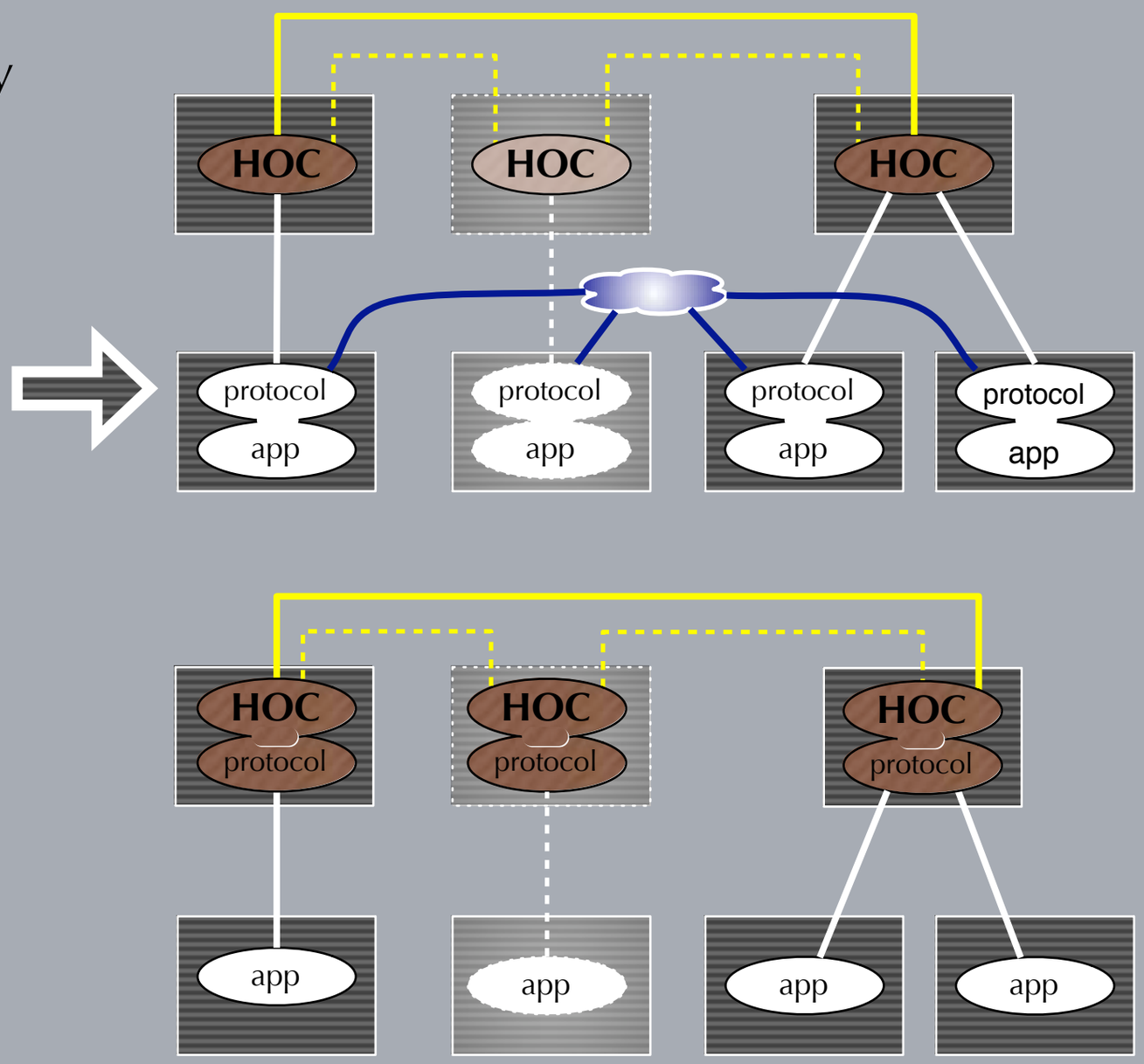
- is efficient (because essential)
  - HOC provide few primitives and no policies for data integrity (e.g. coherence, consistency, ...)
  - these are application specific and may be deployed upon HOC (at the **protocol** level)
- is a basic building block for broad class of applications
  - may be considered a storage component
  - massive storage, out-of-core applications, high-throughput data servers, shared memory support
  - extendible with application-specific primitives
- enhances both memory size and throughput by means of parallelism

… using HOC

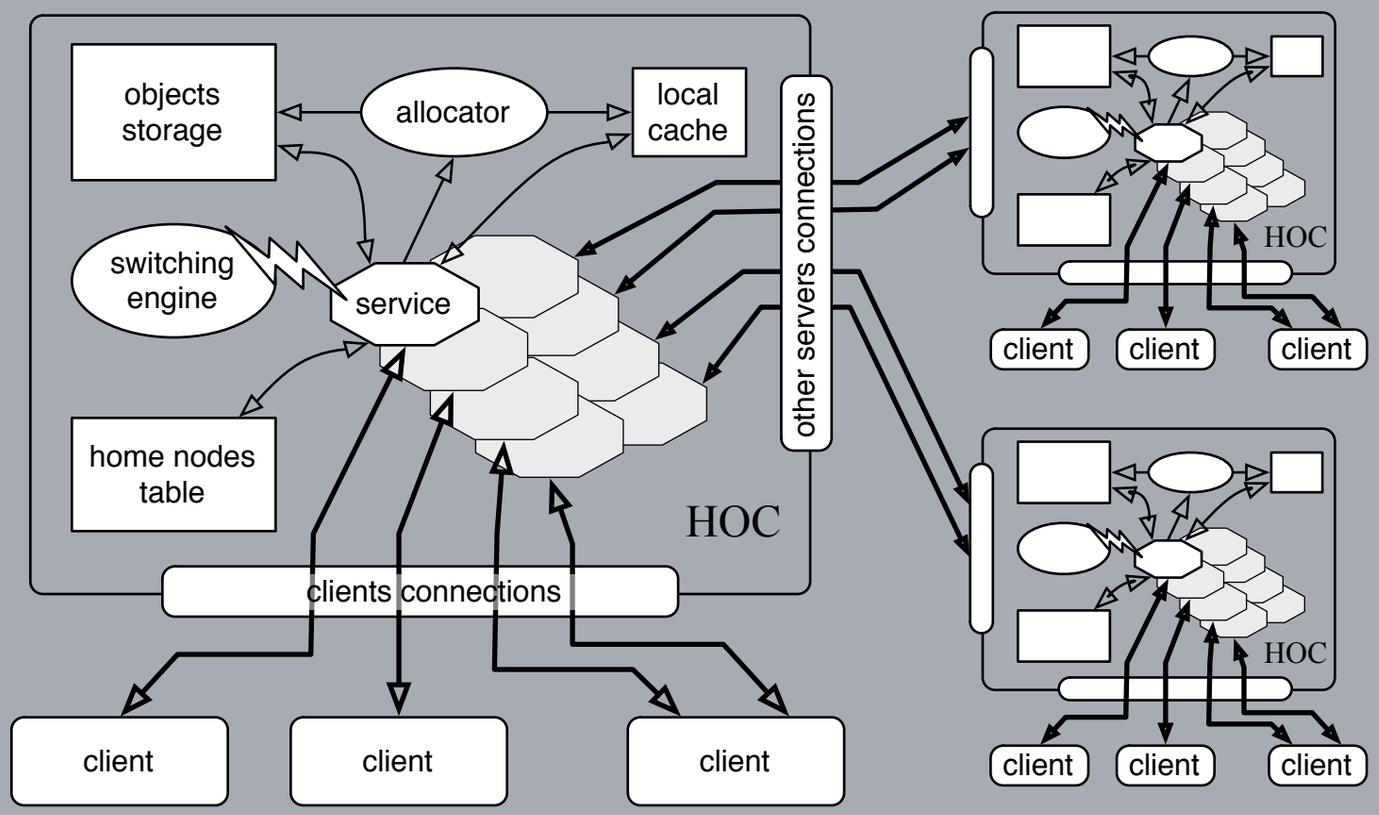
- protocol enforces application requirements on data integrity acting as mediator between the application and HOC
- it is linked to the application and use HOC API
- e.g. Apache module

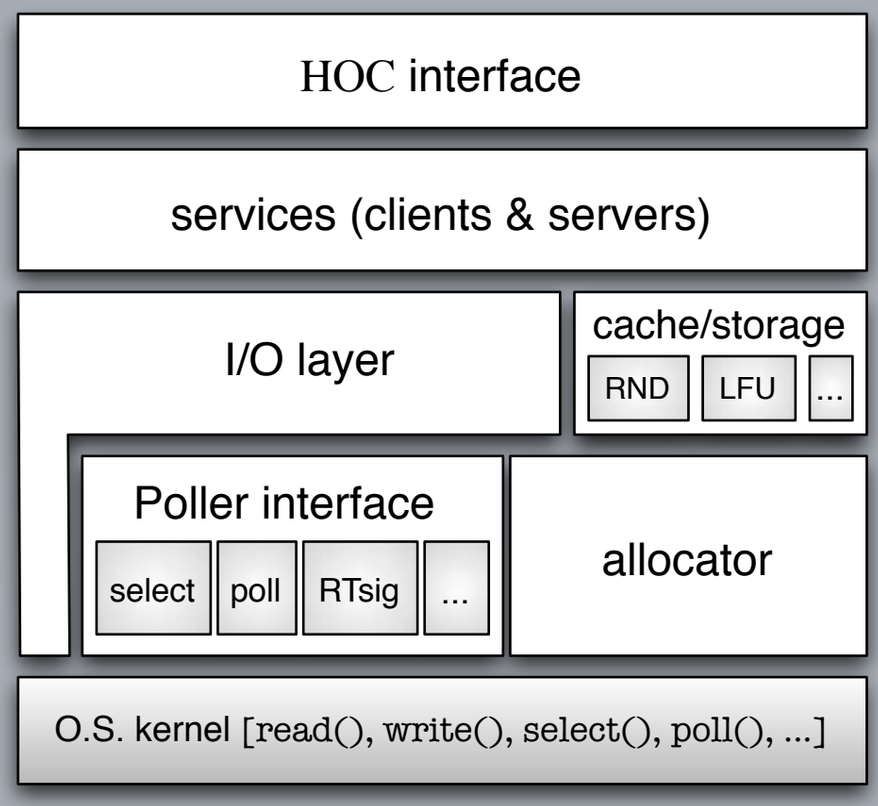
protocol may actually is a distributed application (e.g. reaching consensus, cache invalidation, ...)

HOC API may also be easily extended (provided some knowledge of HOC internals)



HOC internals





- C++, single-threaded, manage concurrent connections using non-blocking I/O based **services** (each of them being a state machine managing a single connection)
  - supporting both level-triggered (`select`, `poll`, ...) and edge-triggered (`RTsignal`, `kqueue`, ...) I/O events
- object storage may be managed either as a memory or a cache, remote objects may be cached in a separate write-through cache. Policies are configurable.
- tested on Linux, MacOS X, and heterogeneous cluster of them

HOC API

Why does the web work so well?
A language with few verbs (get, put, post) ...
Gannon said ... (Europar04, invited talk)

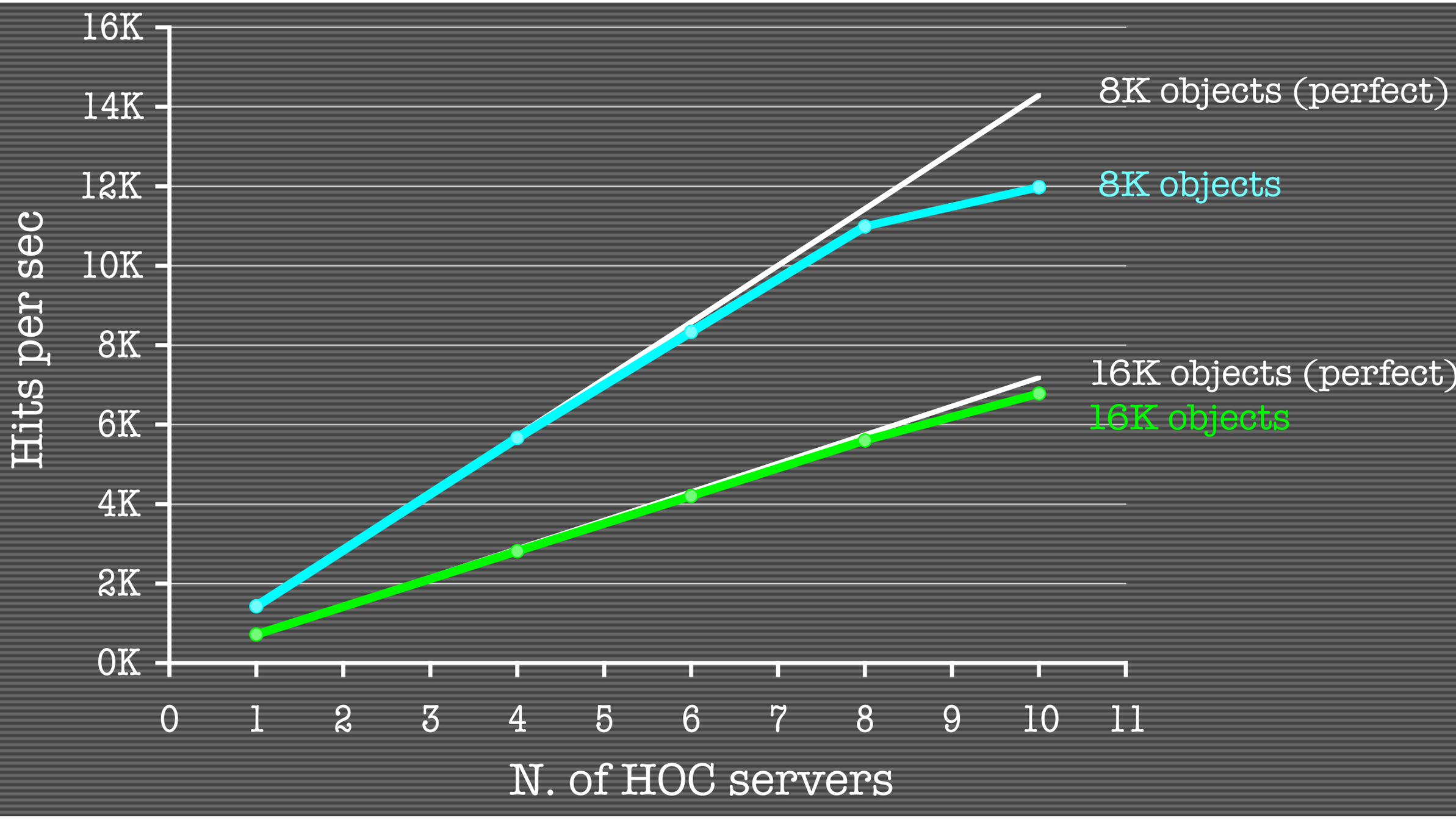
We also believe on such philosophy. As matter of a fact HOC have a four operations API

- `get`, `put`, `remove` arbitrary length objects. Each object is identified by a key and a home node
- `execute(key, op, data)` remotely execute method **`op`** with parameter **`data`** on object identified by **`key`**

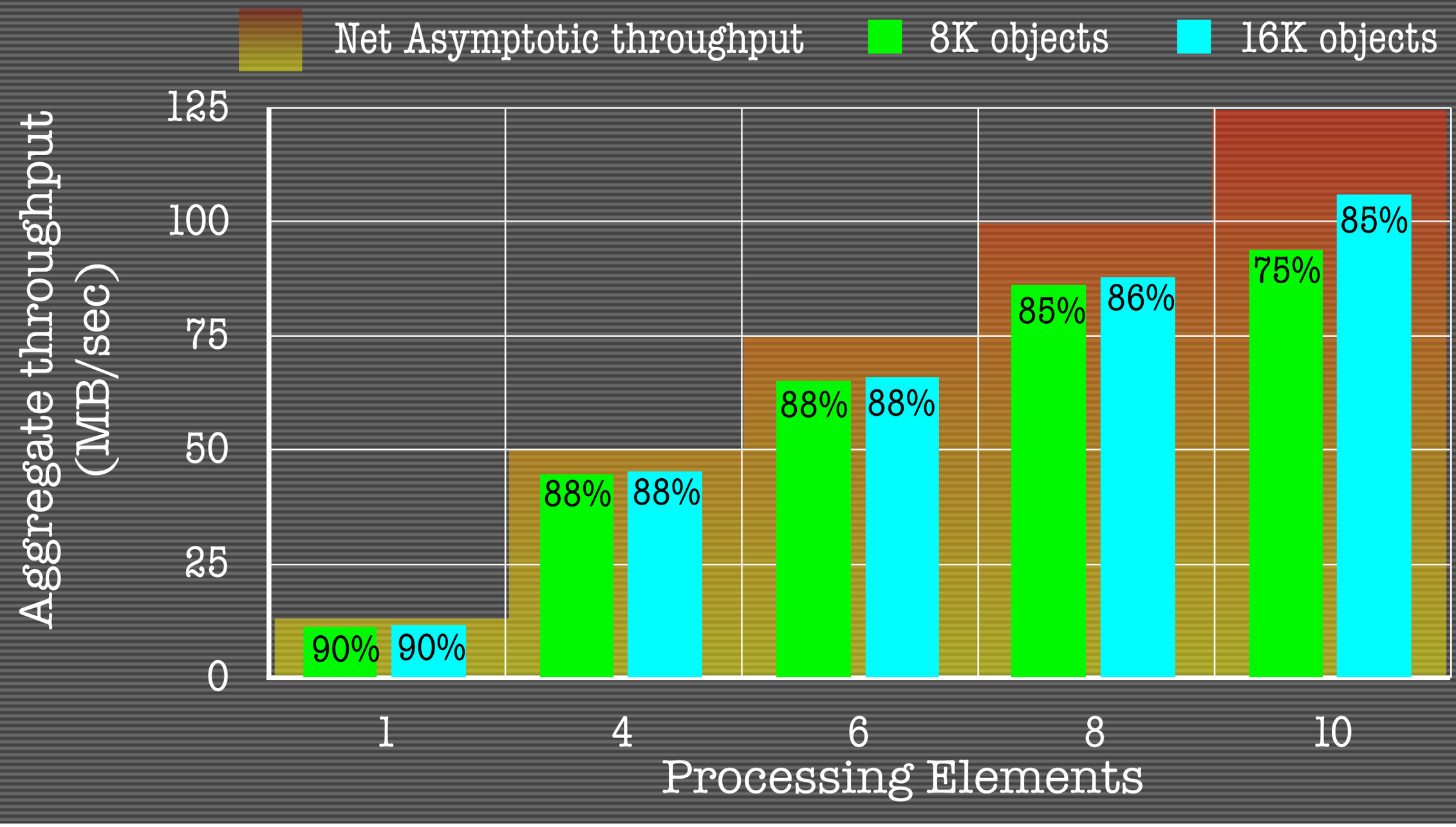
Performance figures (1PE)

Arch/Net/OS	concurrent connections	Msg size (Bytes)	Replies/Sec	net throughput (Bytes/Sec)	net throughput w.r.t. ideal
P4@2GHz Mem 512MB GigaEth Linux ker. 2.4.22	2048	1 M	91	91 M	96%
	3072	512	20 M	10 M	11%
P3@800MHz Mem 1GB FastEth Linux ker. 2.4.18	1024	8 K	1429	11.2 M	90%
	1024	16 K	718	11.2 M	90%

Speedup (Hit per sec VS N. servers)



Sustained aggregate throughput



Summarizing

- HOC is a building block for storage-oriented components
  - distributed caches, distributed memories, parallel repositories
  - configurable, hot-pluggable,
- very good performances
  - close-to-ideal net throughput over thousands of concurrent connections
  - close-to-ideal speedup

Conclusions

- A simple model able to describe what we can expect from our Grid applications
  - Usable as “Ideal performance” slope in papers
- A first effort toward a serious AM
  - A very ongoing work, as asked by Alexander
- Exploit the potentiality of ASSIST+WS+StorageComponent

THANK YOU!

QUESTIONS?

Some references

- M. Aldinucci, M. Coppola, M. Danelutto, M. Vanneschi, C. Zoccolo.
  ASSIST as a research framework for high-performance Grid programming environments.
  Grid Computing: Software Environments and Tools, Springer, 2004, to appear.
- M. Aldinucci, S. Campa, M. Coppola, M. Danelutto, D. Laforenza, D. Puppin, L. Scarponi, M. Vanneschi, C. Zoccolo.
  Components for high-performance Grid programming in the Grid.it project.
  In Component Models and Systems for Grid Applications. Proc. of the Workshop on Component Models and Systems for Grid Applications, June 26, 2004 held in Saint Malo, France. Springer, 2005, to appear.
- M. Aldinucci, S. Campa, S. Magini, P. Pesciullesi, L. Potiti, R. Ravazzolo, M. Torquati, C. Zoccolo.
  Targeting interoperability and heterogeneous architectures in ASSIST.
  In Proc. of Intl. Conference EuroPar2003: Parallel and Distributed Computing, Pisa, Italy, LCNS n. 3149,Springer, August 2003.
- M. Aldinucci, S. Campa, P. Ciullo, M. Coppola, S. Magini, P. Pesciullesi, L. Potiti, R. Ravazzolo, M. Torquati, M. Vanneschi, C. Zoccolo.
  The Implementation of ASSIST, an Environment for Parallel and Distributed Programming.
  In Proc. of Intl. Conference EuroPar2003: Parallel and Distributed Computing, Klagenfurt, Austria, LNCS n. 2790, Springer, August 2003.
- M. Aldinucci, M. Torquati.
  Accelerating Apache farms through ad-HOC distributed scalable objects repository.
  In Proc. of Intl. Conference EuroPar2003: Parallel and Distributed Computing, Pisa, Italy, LNCS n. 3149, Springer, August 2004
- M. Aldinucci, M. Danelutto, J. Dünnweber, S. Gorlatch.
  Optimization techniques for implementing parallel skeletons in Grid.
  Parallel Processing Letters, World Scientific, Singapore, 2005, submitted, to appear (maybe).