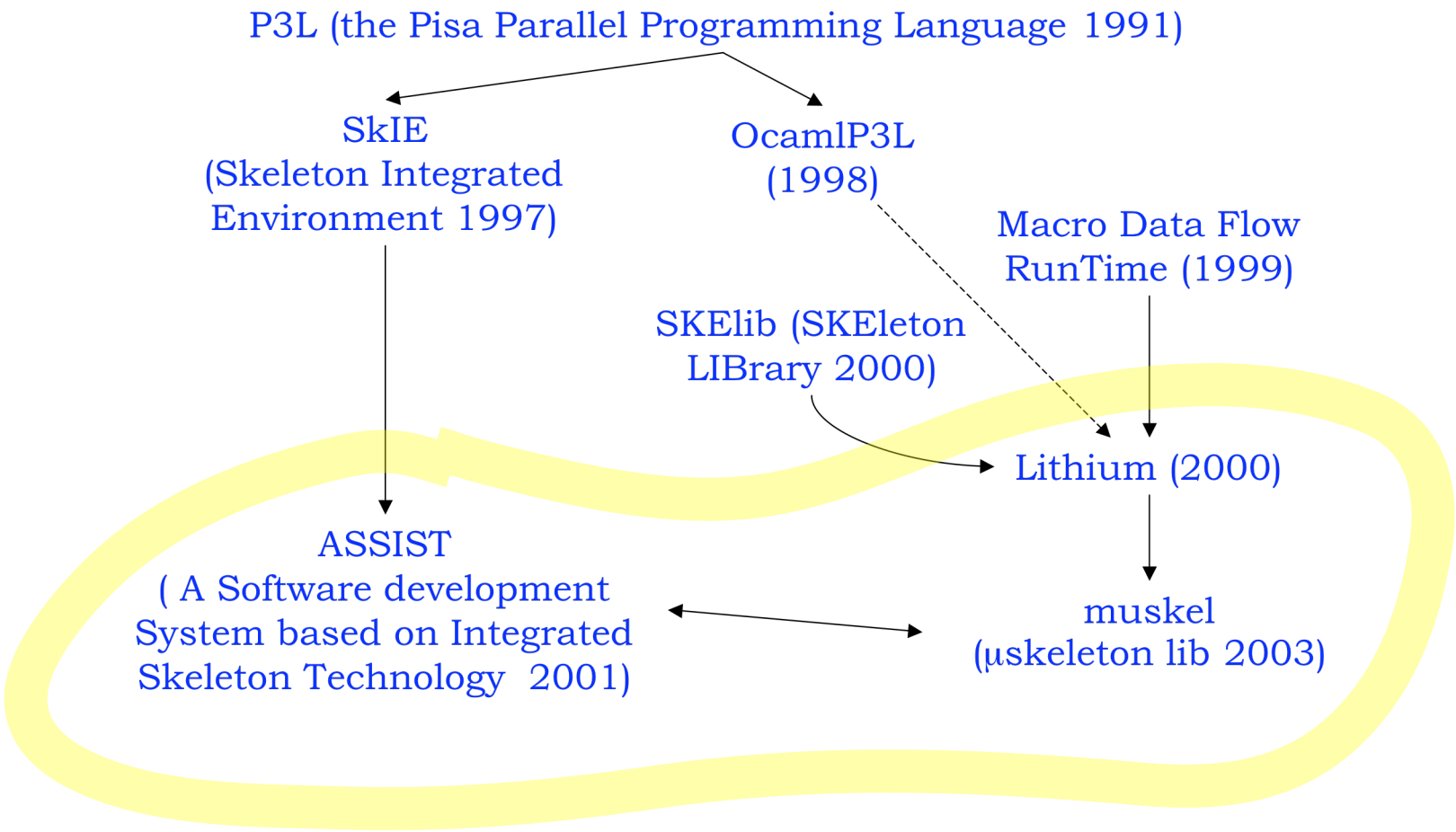
Heterogeneity



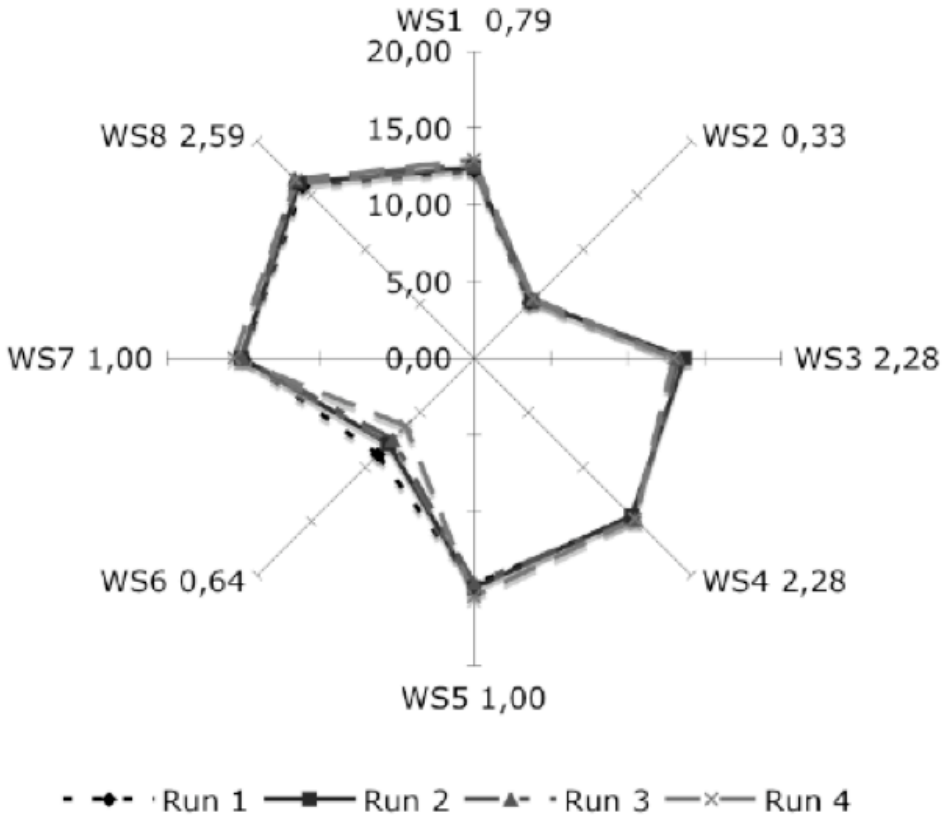
Canonical handling

- Supported by virtual machine
  - Java ☞ Data + code portability through JVM
  - Web services ☞ XML/SOAP/HTML
  - Easier to achieve, less performant (but JIT)
- Supported by compiler tools
  - Cross compiling code for different architectures
  - Dynamic linking to libraries (hopefully)
  - XDR needed somewhere (DVSM ?)
  - Loading time decisions about code to load
  - Harder to implement, much better performance

Support via Virtual machine

- Sample code: task farm
- Code for master and worker written in Java
- Bytecode dynamically loaded on remote processors
- Data exchange via serialization (internal XDR!)
  - Performance advancement vs. XML based formats

Support via virtual machine (exp)



- Tasks per PE
- Heterogeneous
- Production WS !
- WSx YYY (BogoMIPS)
- Different runs

- muskel (2004)

Support via compiler

- Compile time, step 1:
  - Generation of suitable (high level) intermediate code
    - Data interchange code (marshall/unmarshall)
    - Platform independent comm and sync primitives
  - Generation of cross-makefiles
- Load time:
  - Target node kind
- Compile time, step 2:
  - JIT? (☜ right after the loading policies eval)
  - Cross compile vs. ssh compile
- Load time, step 2:
  - Stage needed libraries

Support via compiler (2)

Linux

gcc

0x00010000

Different padding schema

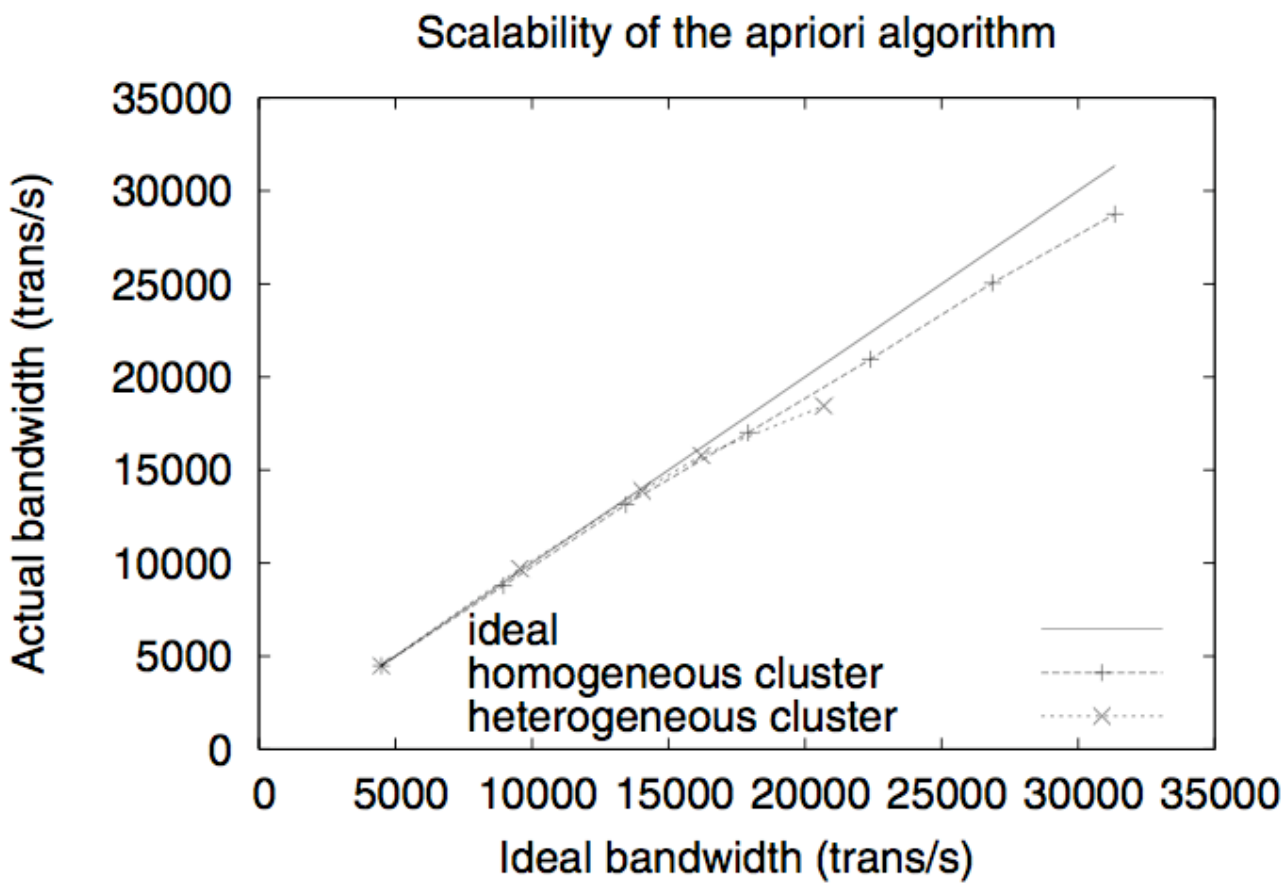
TCP/IP

gcc

MacOSX BSD

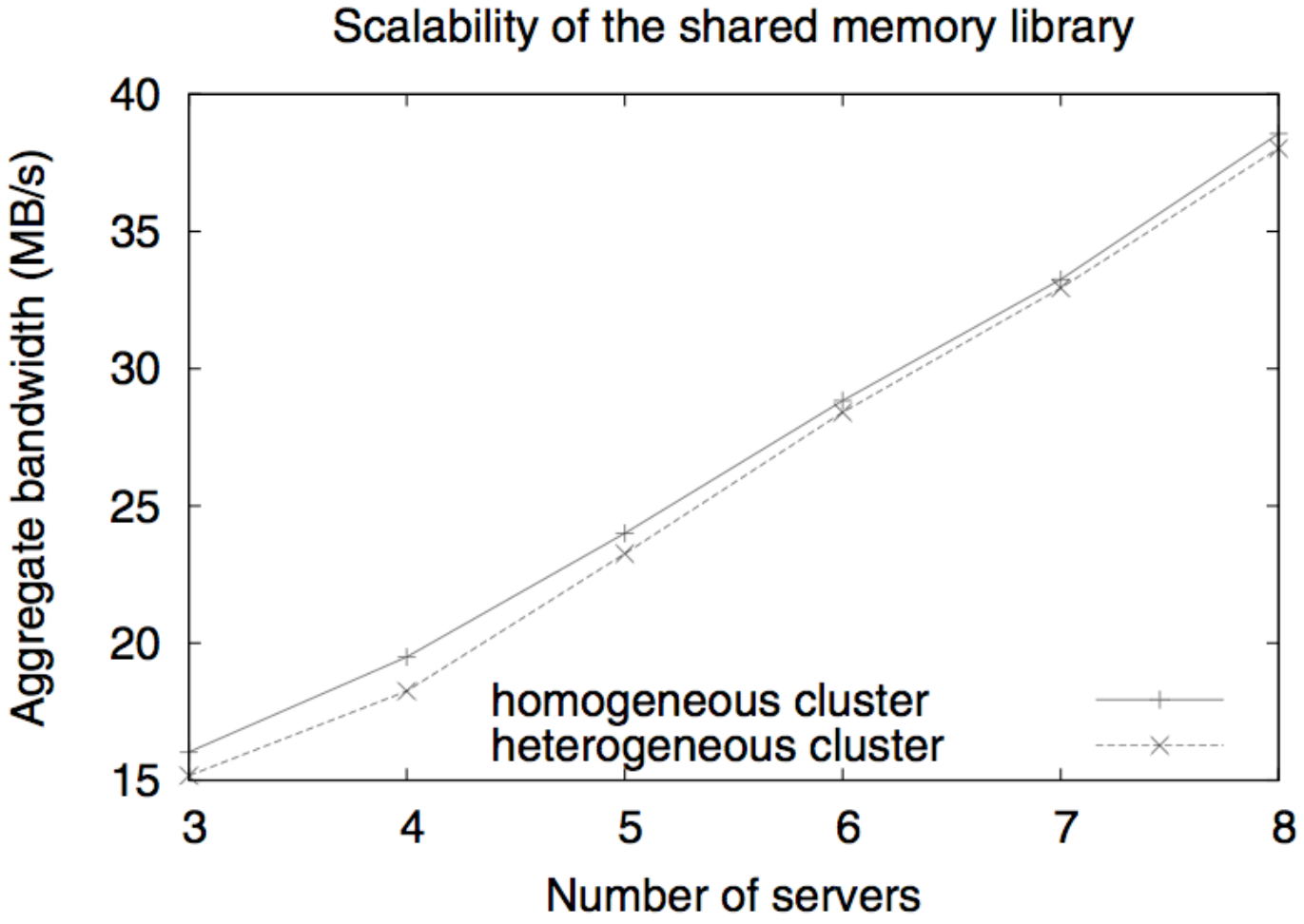
0x00000001

Support via compiler (exp results)



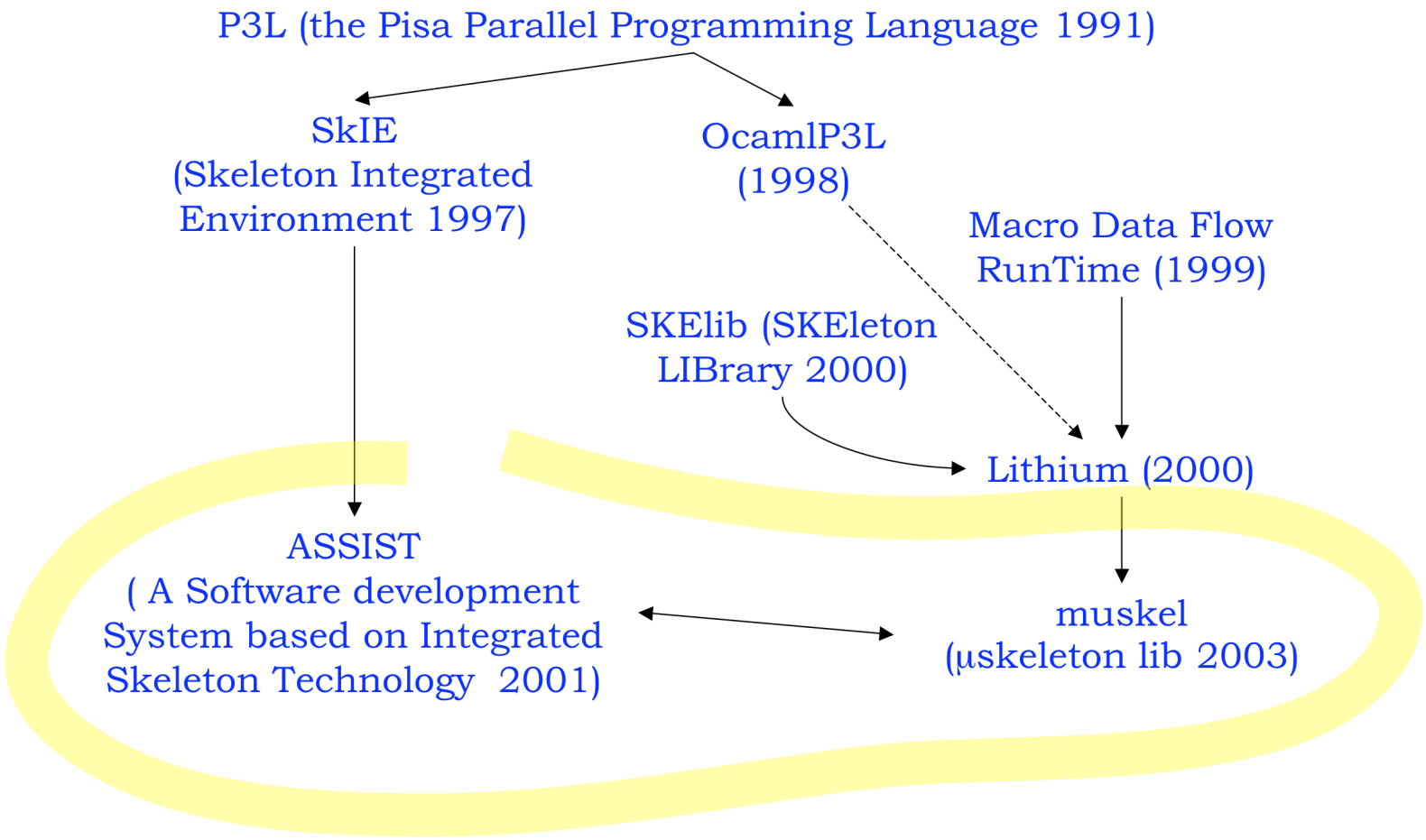
EuroPar04 ASSIST

Support via compiler (exp res 2)



EuroPar04 ASSIST

Dynamicity/Adaptation



Dynamicity sources

- System dependent
  - Load of single, shared machines
  - Heterogeneous machine collections
    - OS react differently to events
  - Network load (connectivity, bandwidth)
  - (node & link) faults
    - (handled separately ☞ fault tolerance)
- Application dependent
  - Hot spots (e.g. stream of “heavy” tasks)
  - Bad, (“system aware”) application coding

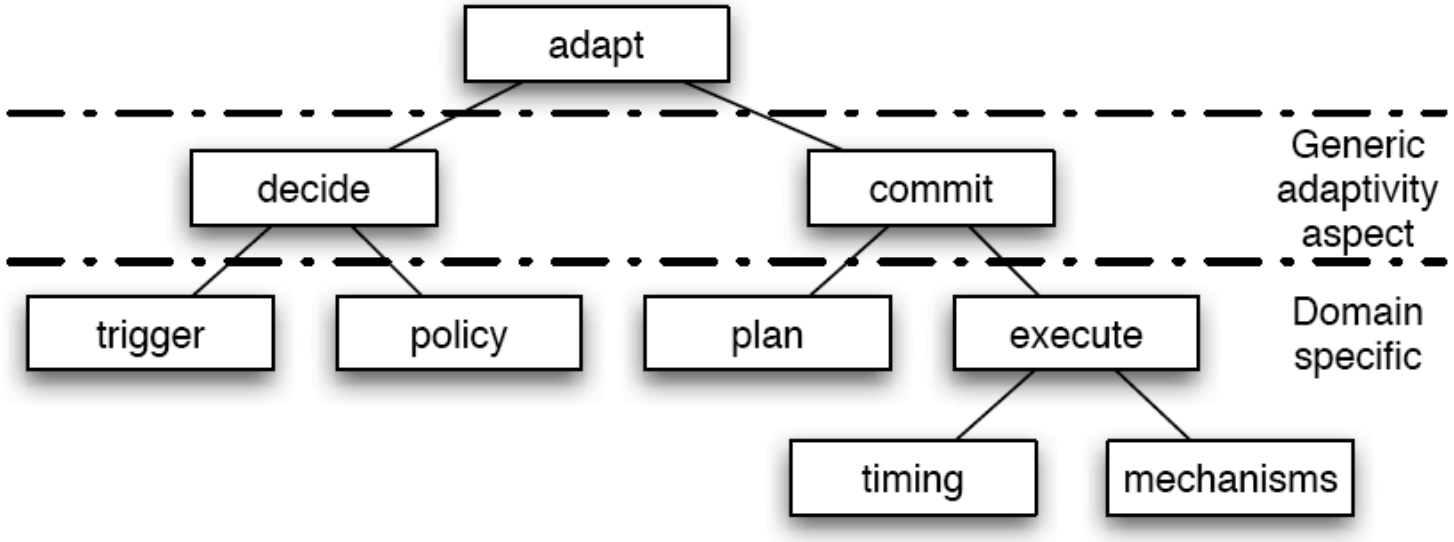
Dynamicity effects

- Resource under-utilization
- Resource over-utilization

- Impaired load balancing policies
- Degraded performance
- Degraded efficiency

Dynamicity handling: adaptation

- Several phases
  - Recognize problem
  - Devise a solution
  - Plan execution
  - Commit & exec
- Each with its own implications
- Each with its own problems/solutions



Adaptivity: decide phase

- Triggering application
  - Monitoring
    - Planned at compile time
    - Automatically handled vs. user driven
    - Target architecture proper mechanisms
  - Policy
    - Intervene on the parallism degree
    - Restructure data distribution
    - ...
    - Possibly supported by performance models

Adaptivity: commit phase

- Make a plan
  - Current computation involved as less as possible
  - Exploit structured parallel programming model features to intervene in the right points
  - Plan a complete set of actions
- Commit
  - Decide which mechanism
  - Apply them according to the plan

Adaptivity: case study (decide)

- Data parallel skeleton
- Initial partition of data across PEs
  - Load balancing achieved
- Several PEs overloaded (different user burst)
- Option 1): look for other PEs, recruit and redistributed
- Option 2): redistribute

- Performance contract driven (!)

Performance contract

- User supplied (top level skeleton/application)
- Several kind of:
  - Service time
  - Completion time
  - Deadline
  - QoS
- Sub-contracts derived for application sub-components
  - Relies on performance models

Adaptivity: case study (commit)

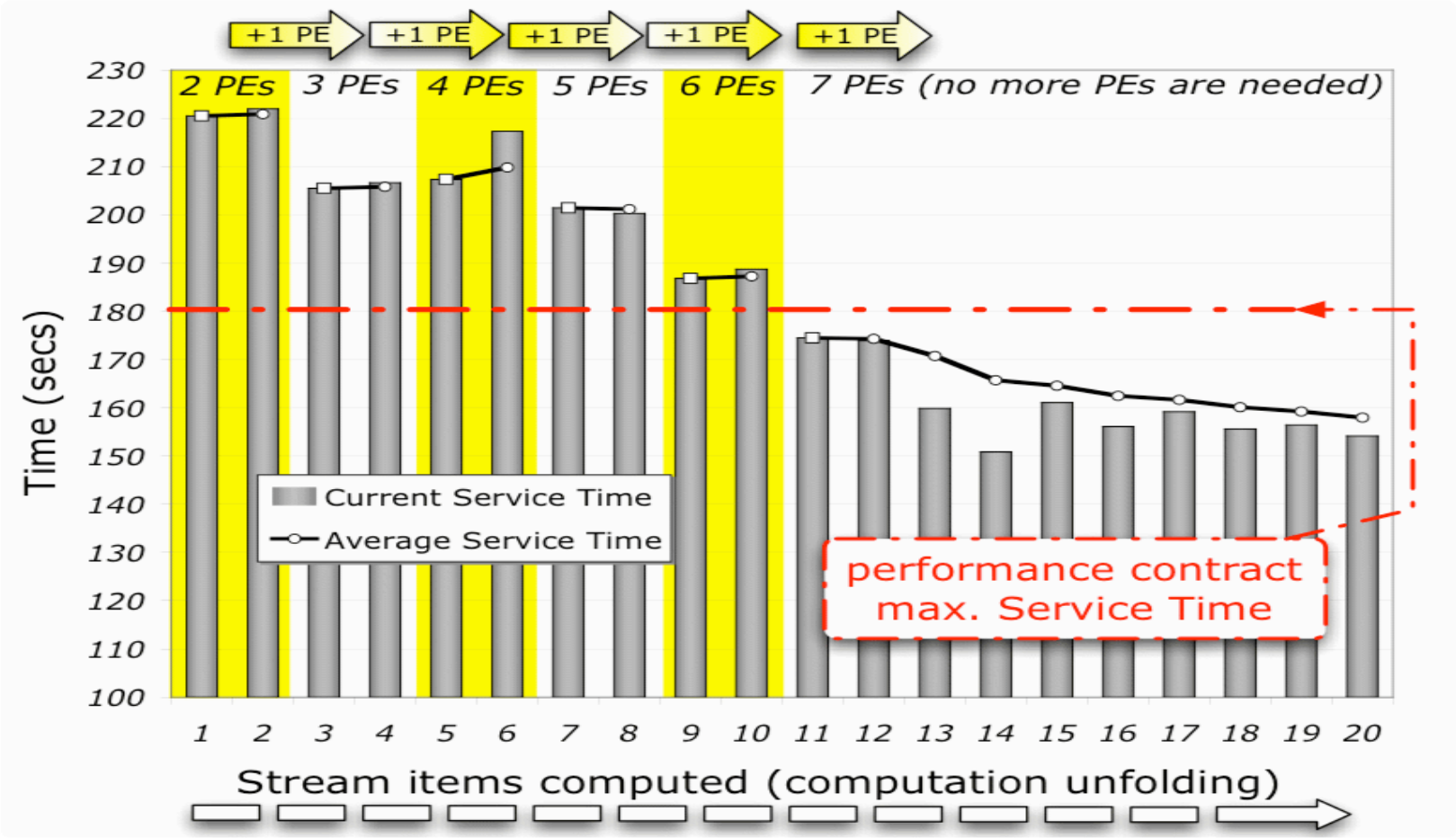
- Embarrassingly parallel data parallel skeleton

- Stop a process with excess load
- Restart with smaller data partition item
- Stop process with low load
- Move data & restart

- Data parallel skeleton with stencil

- Wait for barrier
- Stop all process involved
- Exchange data
- Restart processes

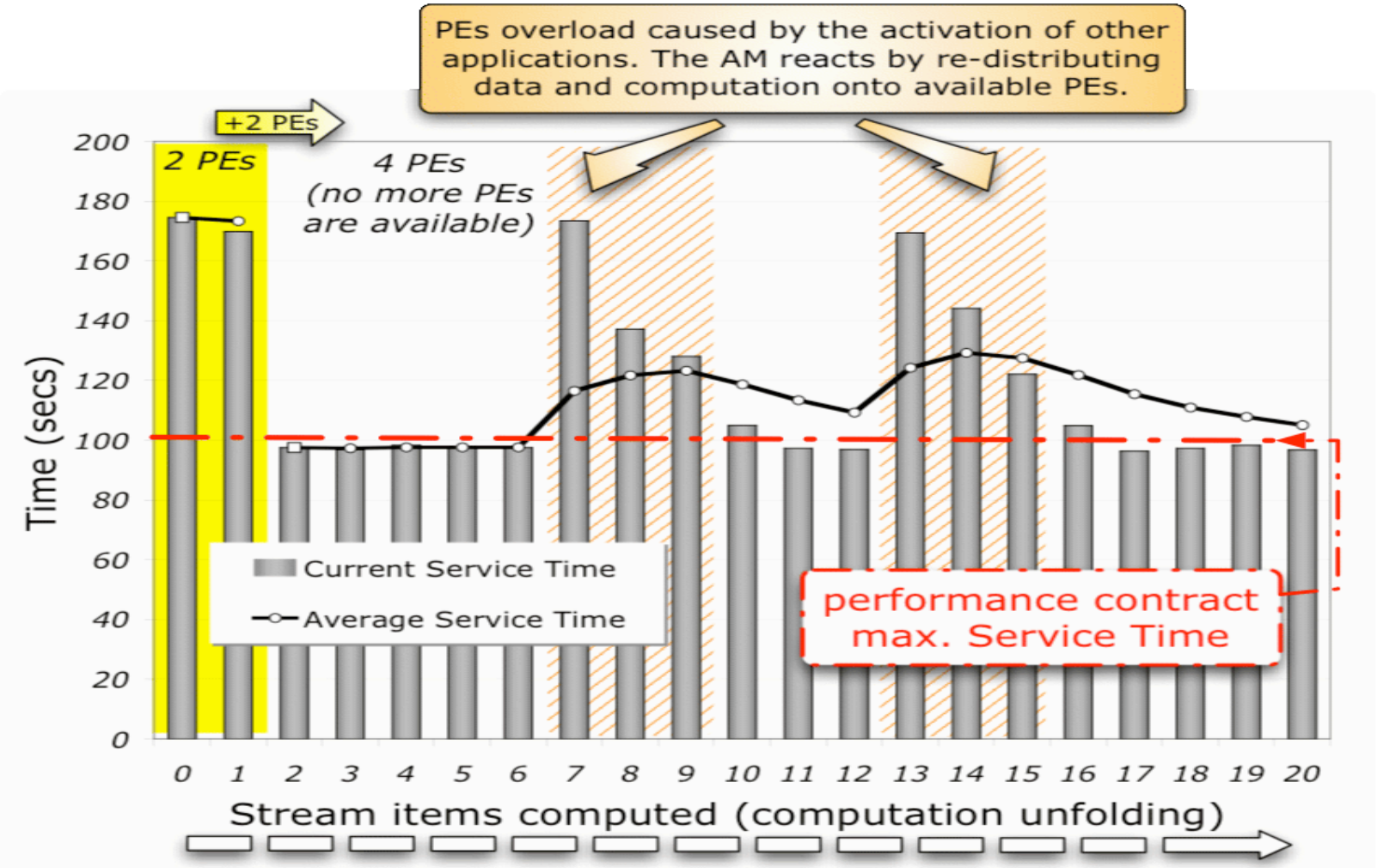
Adaptivity: experiments (ASSIST)



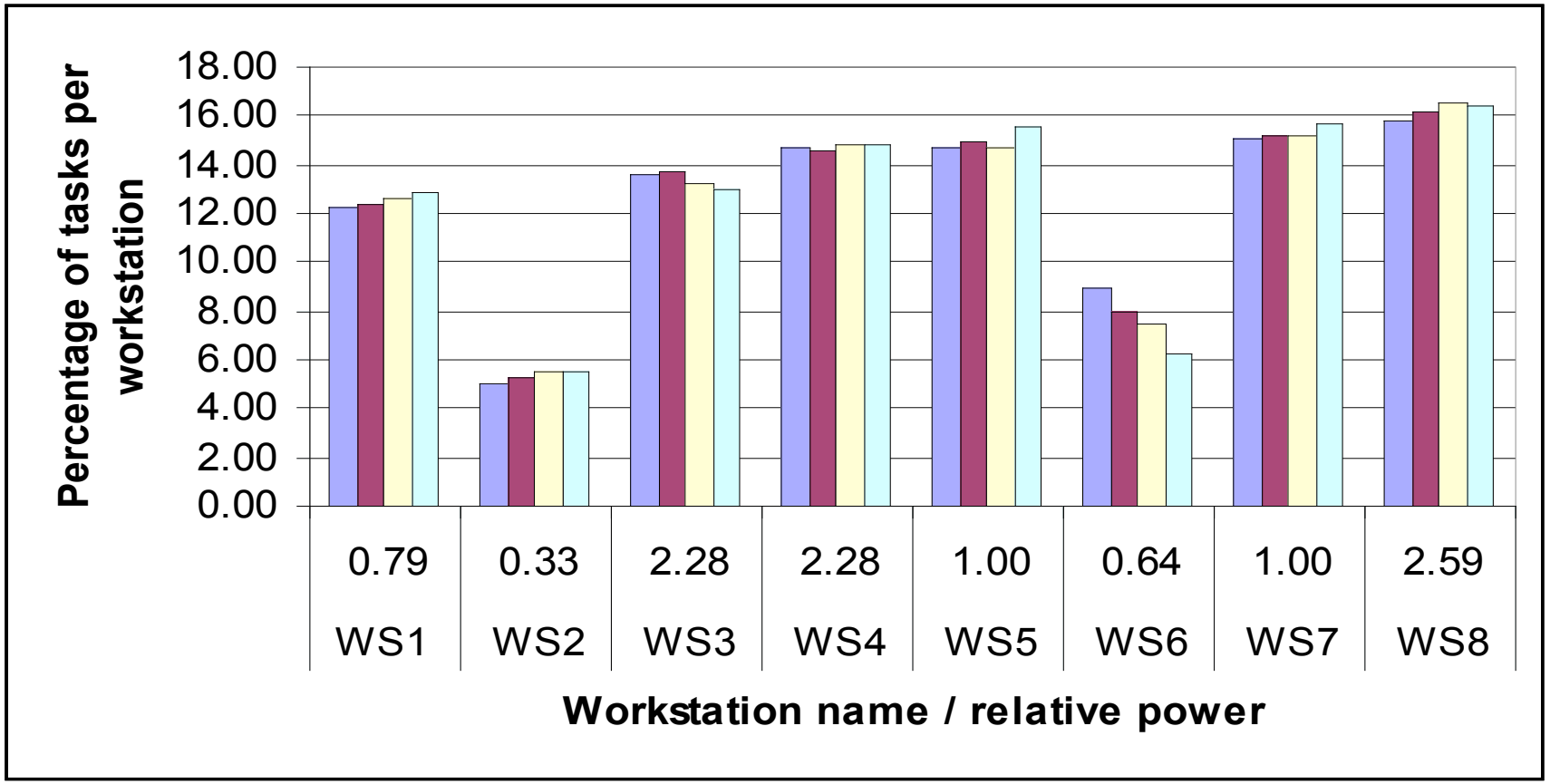
. Aldinucci, et al. Components for high performance Grid programming in Grid.it. *Proc. of the Workshop on Component Models and Systems for Grid Applications*, CoreGRID series. Springer Verlag, January 2005.

Adaptivity: experiments (ASSIST)

. Aldinucci, et al. Components for high performance Grid programming in Grid.it. *Proc. of the Workshop on Component Models and Systems for Grid Applications*, CoreGRID series. Springer Verlag, January 2005.



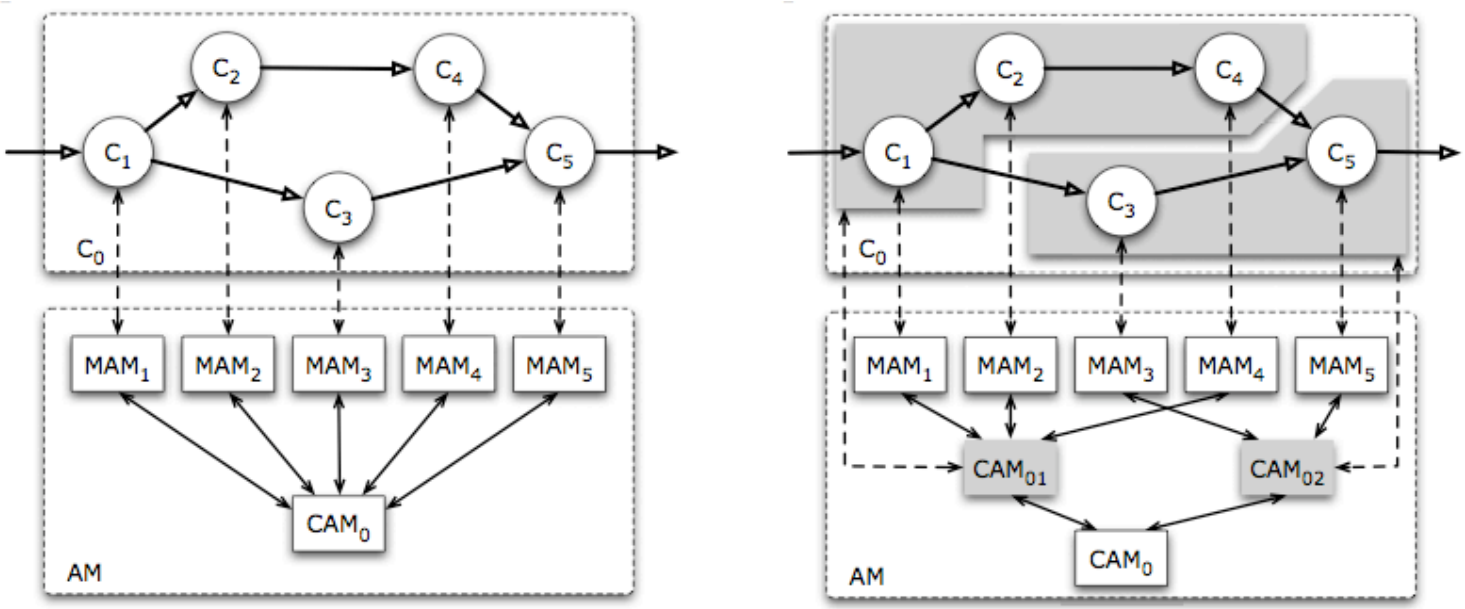
Adaptivity: experiments (muskel)



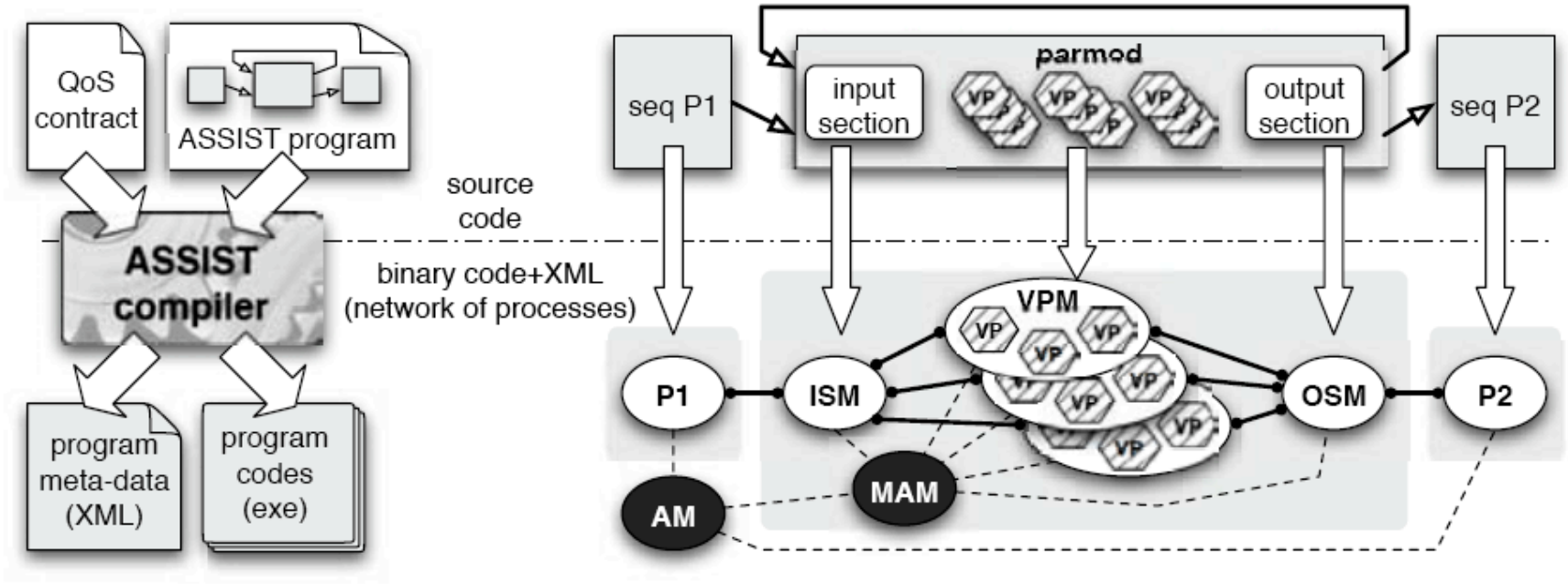
M. Danelutto, *QoS in parallel programming through application managers*
Euromicro Conference on Parallel, Distributed and Network-based processing, Lugano, Feb. 2005

Adaptivity: ASSIST

- Each parmod has its own *manager*
  - Takes care of optimizing performance
- Parmods in a generic graph
  - Graph *manager* optimizes overall computation

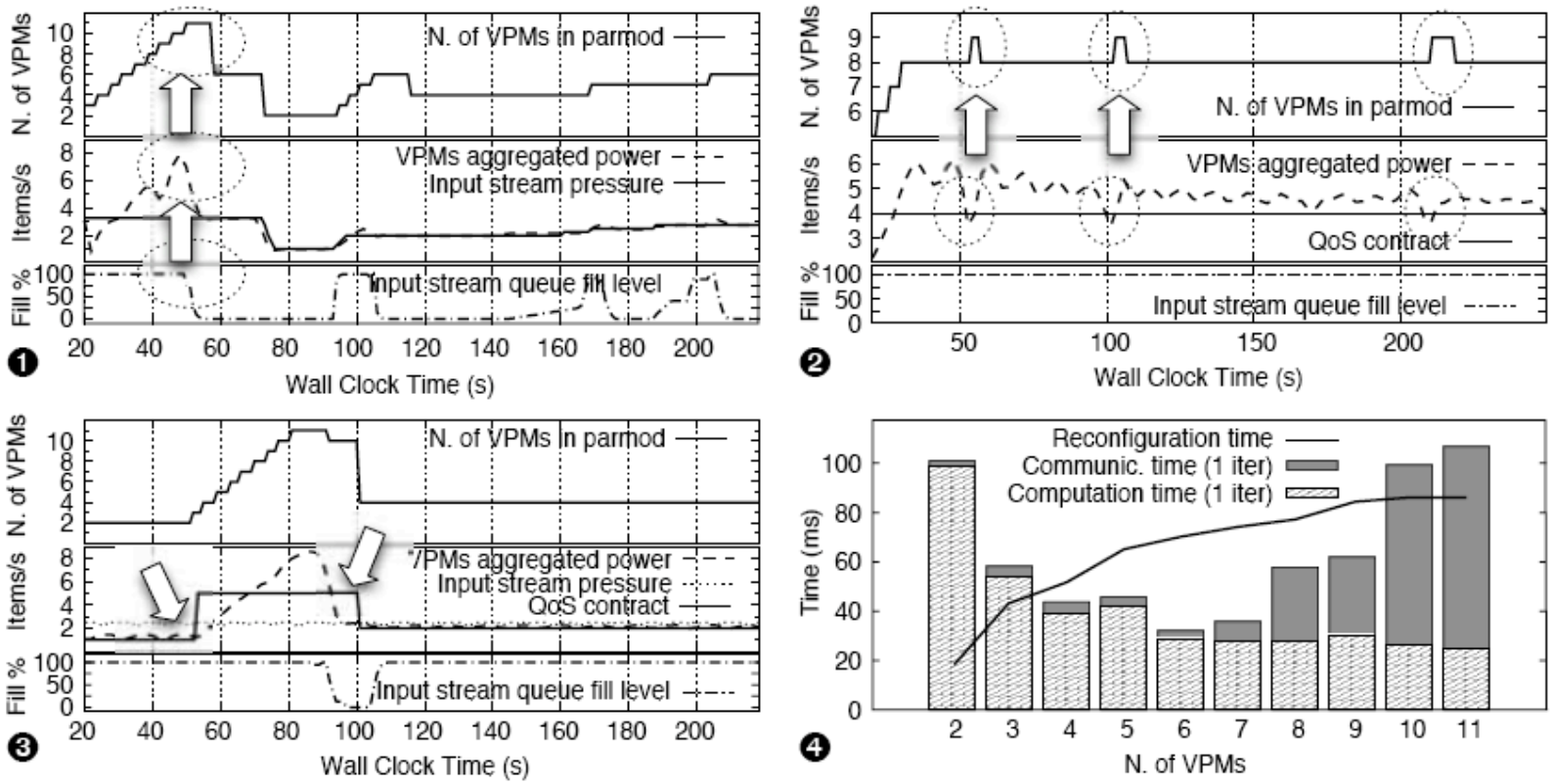


Implementation



Aldinucci et al. *Dynamic reconfiguration of grid-aware applications in ASSIST,* Europar 2005

Adaptivity: recent results(ASSIST)



Aldinucci et al. *Dynamic reconfiguration of grid-aware applications in ASSIST,* Europar 2005