Grid programming with components:
an advanced COMPonent platform
for an effective invisible grid

GCM Non-Functional Features and ProActive

Marco Aldinucci
&
M. Danelutto, S. Campa,
D. LaforenzA, N. Tonellotto, P.Dazzi

UniPisa & ISTI-CNR

e-mail: aldinuc@di.unipi.it

Outline

- Not really Proactive user case
  - Bringing some ideas
  - Proposed for GCM (CoreGRID/GridCOMP)
  - Experienced with ASSIST
  - Also, currently experimenting using ProActive
- Proactive User case
  - Already described last monday
  - I repeat if time

GridComp MODEL key points

- Hierarchic model
  - Expressiveness
  - Structured composition
- Interactions among components
  - Collective/group
  - Configurable/programmable
  - Not only RPC, but also stream/event
- NF aspects and QoS control
  - Autonomic computing paradigm

GCM implementation aspects (in my viewpoint at least)

- Membrane is an active object
  - Centralized implementation
- Controller are components
  - One possible choice, among the others
  - Lightweight components
- Communication protocol
  - Asynchronous communications
  - Krakow feedback. Rodolfo Toledo, Eric Tanter, Jose Piquer: USING REFLEXD FOR A GRID SOLUTION TO THE N-QUEENS PROBLEM: A CASE STUDY. CoreGRID Integration Workshop, Karkow, October 2006

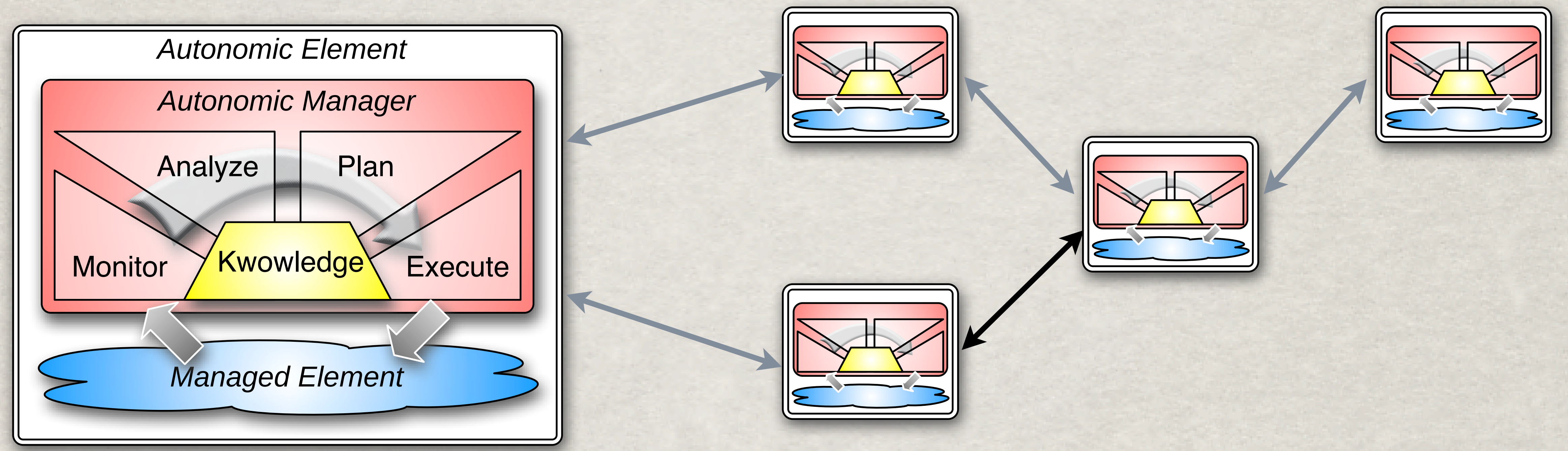
Autonomic Computing Paradigm (AC)

- Aims to tackle the complexity of QoS management providing self-managing components, i.e. :
  - Self-configuring
  - Self-optimizing
  - Self-healing
  - Self-protection
- Basically control loops
  - Basic theory dates back to last mid-century decade
  - Recently re-vamped and propelled by IBM

AC Bare Bones

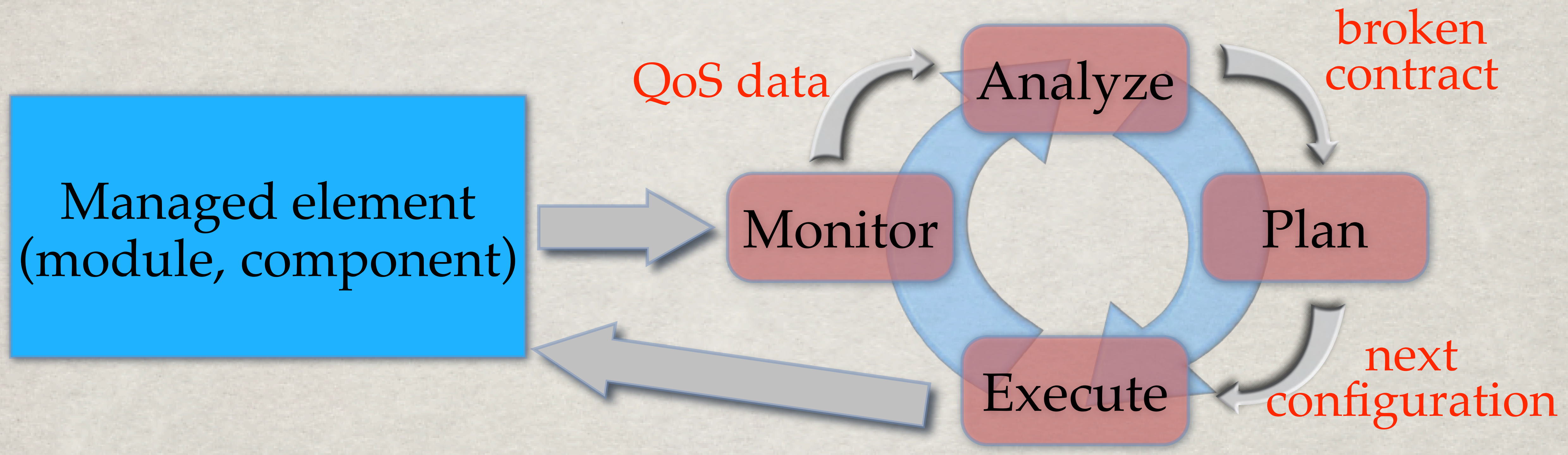
- A complex system is usually set up by distinct elements
  - composed in horizontal fashion (i.e. used_by/provided_to)
  - nested in vertical fashion (i.e. implemented_by)
- AC idea:
  - Each entity exhibits certain self-management capability
  - At each level, entities cooperate to self-manage their aggregation
  - Each level subsumes capability at the next level down

An AC Element & its "horizontal" Companions



- AC element
  - Managed Element
  - Autonomic Manager
- AC elements co-operate to achieve a common goal
  - Possibly with dynamic patterns along running time

Insulated AC Element Cycle



- **Monitor**: collect execution stats: machine load, service time, input/output queues lengths, ...
- **Analyze**: instantiate performance models with monitored data, detect broken contract, in and in the case try to individuate the problem
- **Plan**: select a (predefined or user defined) strategy to re-convey the contract to valid status. The strategy is actually a list of mechanism to apply.
- **Execute**: leverage on mechanism to apply the plan

AC Element - ASSIST Experience

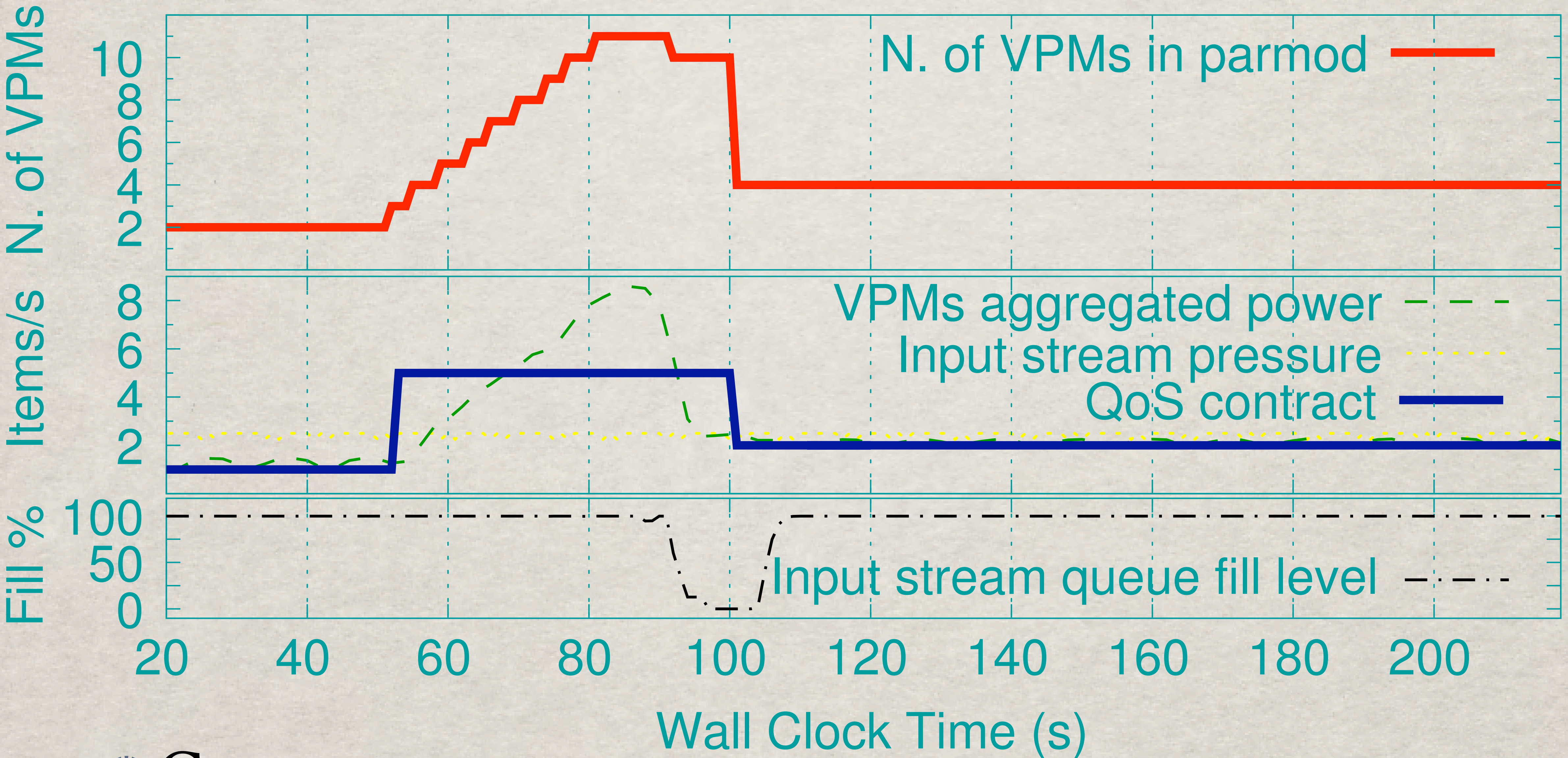
- Some experiences already done
  - Based on QoS contracts
  - Autonomic parmod
  - Autonomic supercomponents
    - Higher order components
    - DAG, Farm

M. Aldinucci and M. Danelutto. Algorithmic skeletons meeting grids. *Parallel Computing*, 32(7-8): 449–462, 2006.
M. Aldinucci, M. Danelutto, M. Vanneschi. Autonomic QoS in ASSIST Grid-aware components. In *Euromicro PDP 2006: Parallel Distributed and network-based Processing*, IEEE, Montbéliard, France, February 2006.
M. Aldinucci, C. Bertolli, S. Campa, M. Coppola, M. Vanneschi, L. Veraldi, C. Zoccolo. Self-Configuring and Self-Optimising Grid Components in the GCM model and their ASSIST implementation. In HPC-GECO/Compframe 2006 (held in conjuction with HPDC-15), IEEE, Paris, France, June 2006.
M. Aldinucci, A. Petrocelli, E. Pistoletti, M. Torquati, M. Vanneschi, L. Veraldi, and C. Zoccolo. Dynamic reconfiguration of grid-aware applications in ASSIST. In J. C. Cunha, and P. D. Medeiros, editors, Proc. of *11th Intl Euro-Par 2005: Parallel and Distributed Computing*, volume 3648 of *LNCS*, Lisboa, Portugal. Springer Verlag, August 2005.
....

QoS contract Example (ASSIST)

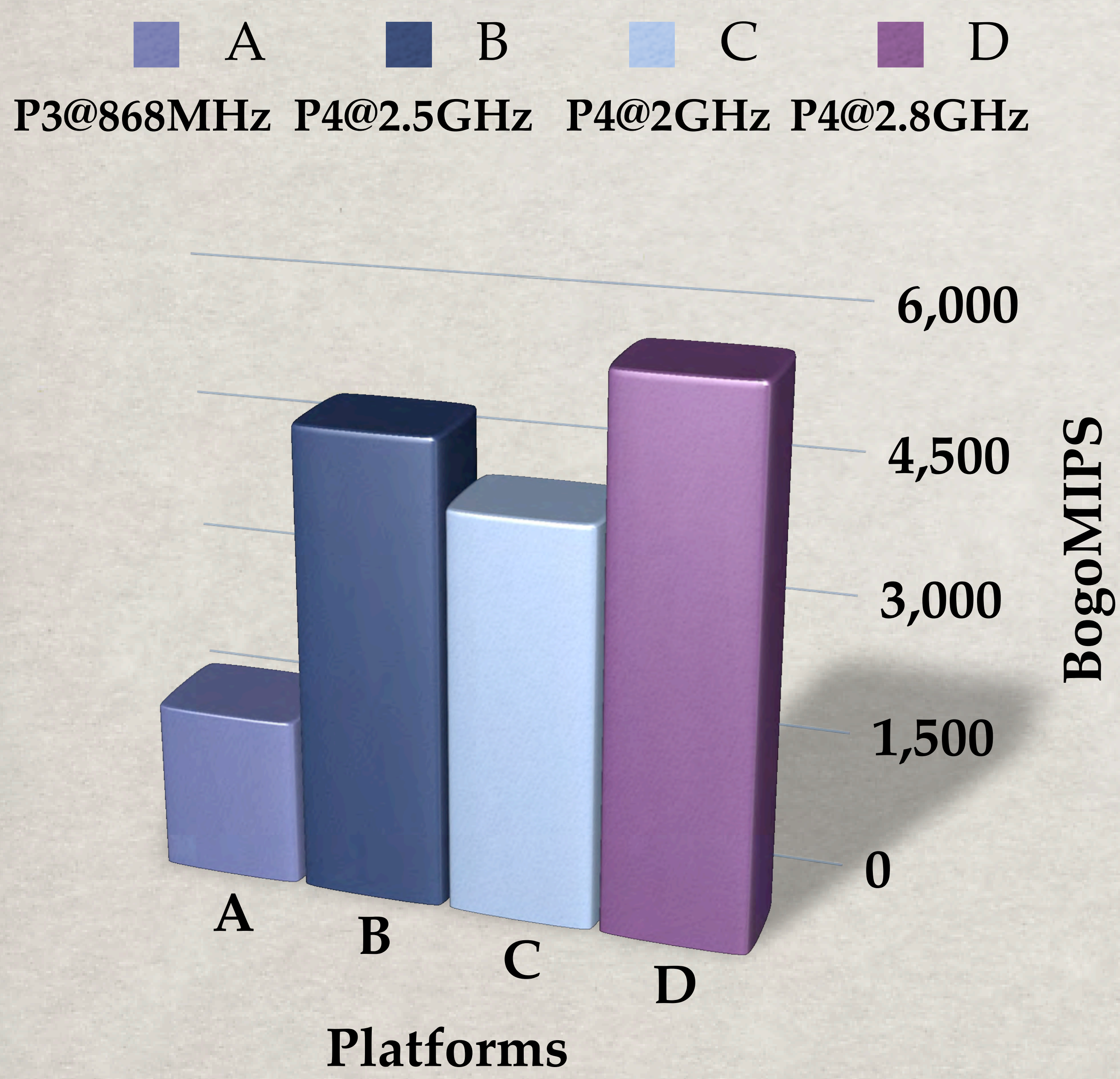
Perf. features	QL_i (input queue level), QL_o (input queue level), T_{ISM} (ISM service time), T_{OSM} (OSM service time), N_w (number of VPMs), $T_w[i]$ (VPM$_i$ avg. service time), T_p (parmod avg. service time)
Perf. model	$T_p = \max\{T_{ISM}, \sum_{i=1}^{n} T_w[i]/n, T_{OSM}\}$, $T_p < K$ (goal)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

Exp 1: Stateless FARM

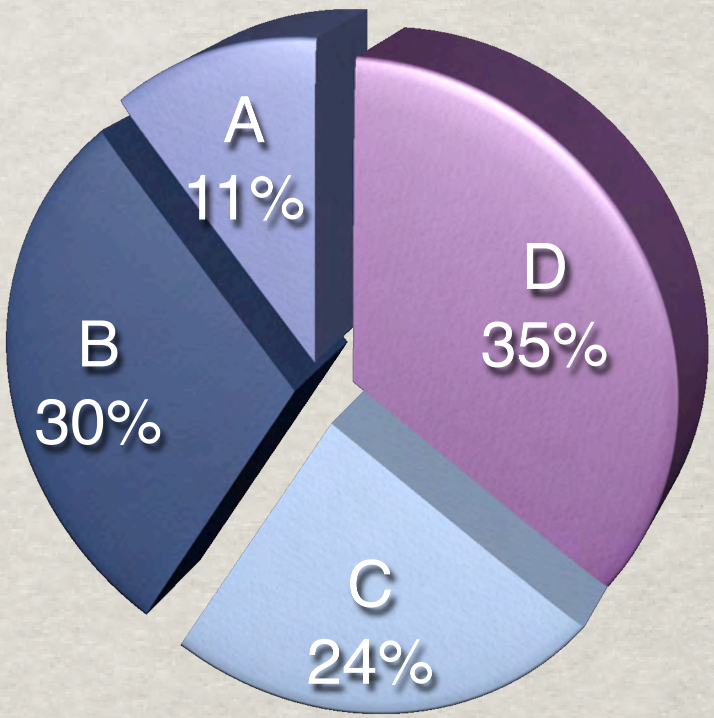


- Contract:
  - keep a given service time
  - contract change along the run

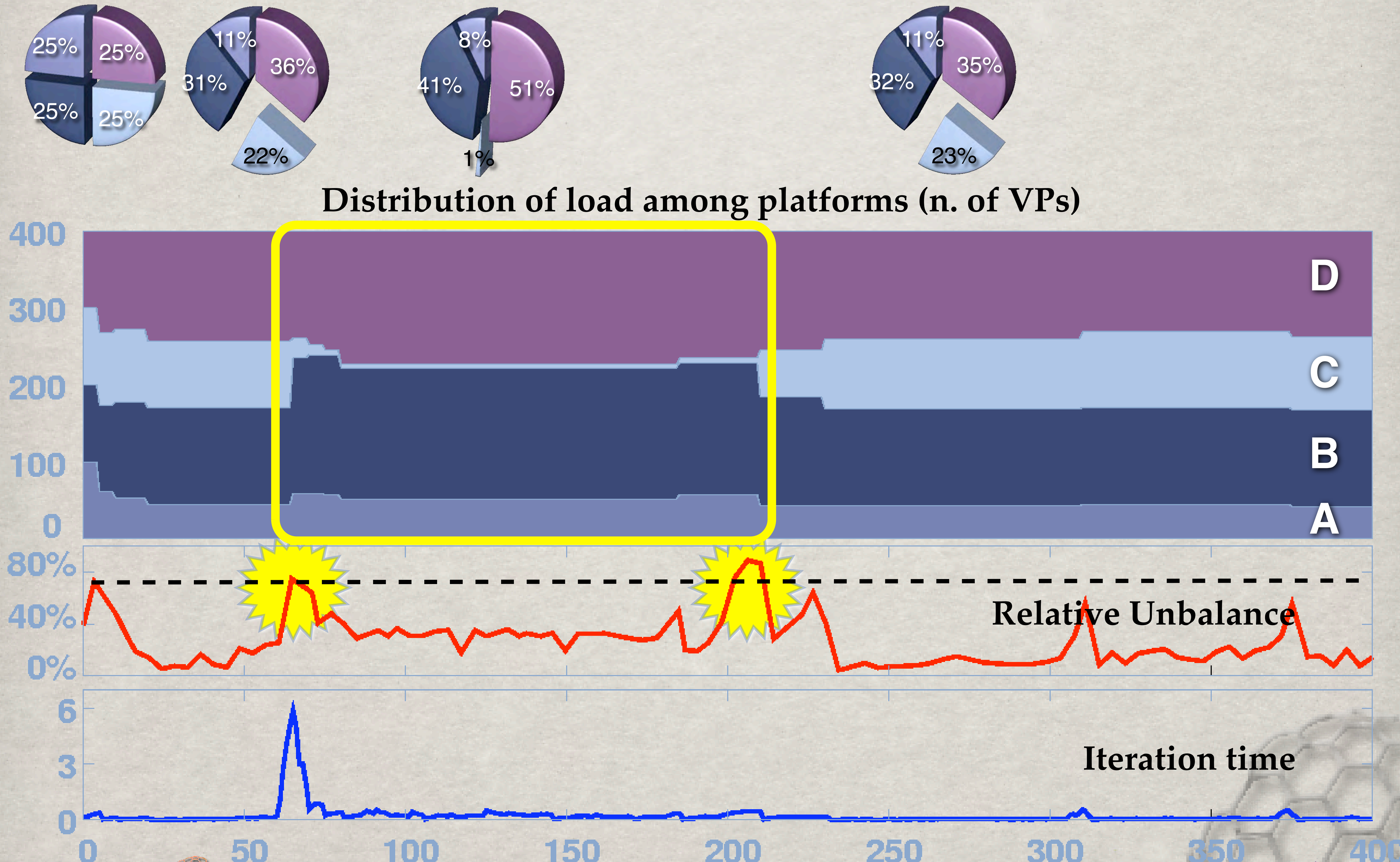
Exp 2: Data-Parallel(STP)



Expected work balance among platforms



Exp 2: Data-Parallel(STP)

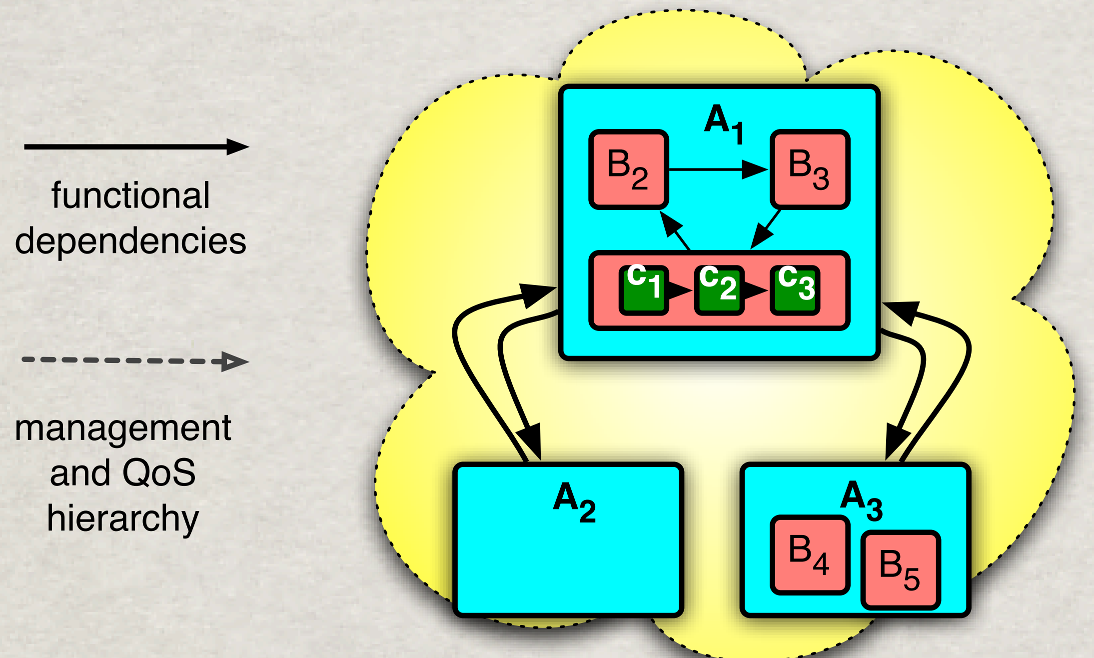


Overhead? (mSecs)

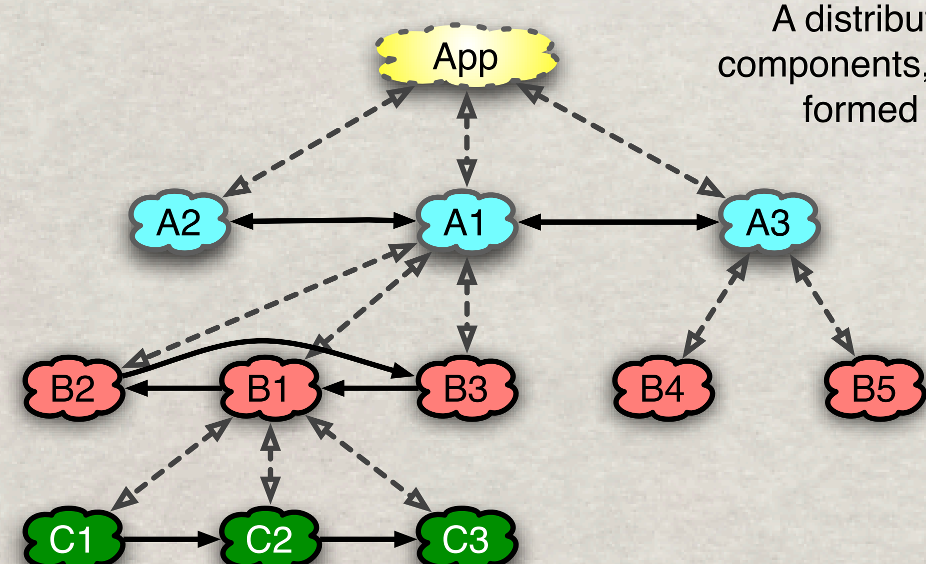
parmod kind	Data-parallel (with shared state)						Farm (without shared state)					
reconf. kind	add PEs			remove PEs			add PEs			remove PEs		
# of PEs involved	1→2	2→4	4→8	2→1	4→2	8→4	1→2	2→4	4→8	2→1	4→2	8→4
R_l on-barrier	1.2	1.6	2.3	0.8	1.4	3.7	–	–	–	–	–	–
R_l on-stream-item	4.7	12.0	33.9	3.9	6.5	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.

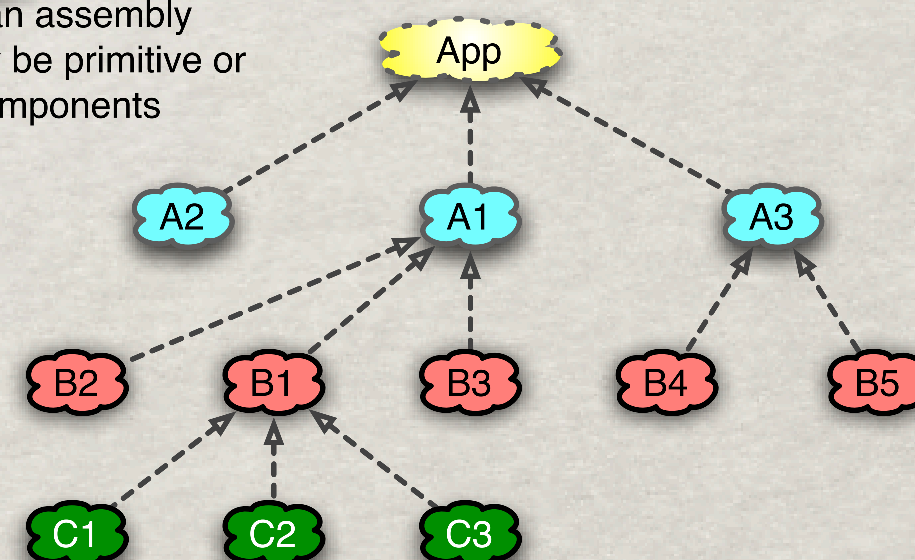
Vertical Composition



A distributed App is an assembly components, which may be primitive or formed by other components



The QoS of a component depends by its nested components and their functional relations. Components may include either sequential or distributed code



Provided QoS can be synthesized in a bottom-up fashion, while requested QoS imposed in top-down fashion. Application management can be distributed along the hierarchy to improve management locality

Autonomic Cycle & Vertical

- Autonomic cycle manage some further points
  - Accepts new QoS contracts from father manager
  - Raises locally unmanageable contract violations
  - At each level, implements cooperation with other partners
- Formalization is an open problem

Horizontal & Vertical orchestration

- Open problems
- A satisfactory formalization is missing
  - how describe QoS proprieties
  - Describe distributed parametric analysis strategies & reconfiguration plans
    - How to generate them automatically, how to enforce locality of actions
- Some experiences already done with ASSIST, some promising ideas
  - Exploiting structured orchestration of activities (super-components)

Rationale

- AC promising
- Something can be already done
  - Experiences in ASSIST given good feedbacks in terms of reactivity, low-overhead, ...
  - Documented in literature
- Several, very interesting open problems
  - At the border with Global Computing community
  - Very interesting for EU VII FP

CoreGRID GCM NF features

- Autonomic behavior
  - EU 7 FP, NGG3, blah blah ...
- Renewed proposal based on:
  - Fractal style level of compliance
  - Passive or active vertical interaction

Fractal Conformance levels

Minor (κ)		1		1		1		1	2	3
Major (Θ)	0	0	1	1	2	2	3	3	3	3
Component			✓	✓	✓	✓	✓	✓	✓	✓
Interface					✓	✓	✓	✓	✓	✓
Component Type Interface Type							✓	✓	✓	✓
Attribute, Content, Binding LifeCycle Controller		✓		✓		✓		✓	✓	✓
Factory									✓	✓
Template										✓

Conformance level Θ.κ

Fractal Conformance levels Rephrased and GCM

- Major (Θ) $\geq$ 1 $\Leftrightarrow$ "it is a component"
  - Minor (κ) $\geq$ 1 $\Leftrightarrow$ "it exhibits AC, CC, BC, LC"
    - Minor (κ) =2&3 have a bit uneven meaning (F, T)
- Add another counter describing NF behavior $\Theta.\kappa.\alpha$ (as partial function)
  - α=0 $\perp$, only if (Θ<1 or κ<1) (observationally undecidable)
  - α=1 No autonomicity
  - α=2 Passive autonomicity (low-level, server only NF intf)
  - α=3 Active autonomicity (high-level, client/server NF intf)

Some Aspect still not Clear

- Main concerns
  - How much the model should be specified?
    - Not that much, at the end this is why we adopted Fractal ...
    - It should be a Model not the specification of an implementation
      - OO Model is not Java specification
    - Membrane
  - Fractal/ProActive implementation
    - Maps 1:1 to GCM reference implementation?
    - Are group communications implemented by controllers?
    - Controllers=components? *(in which component model?)*
    - How controllers interoperate and how are programmed?
    - Is membrane admitting a distributed implementation?

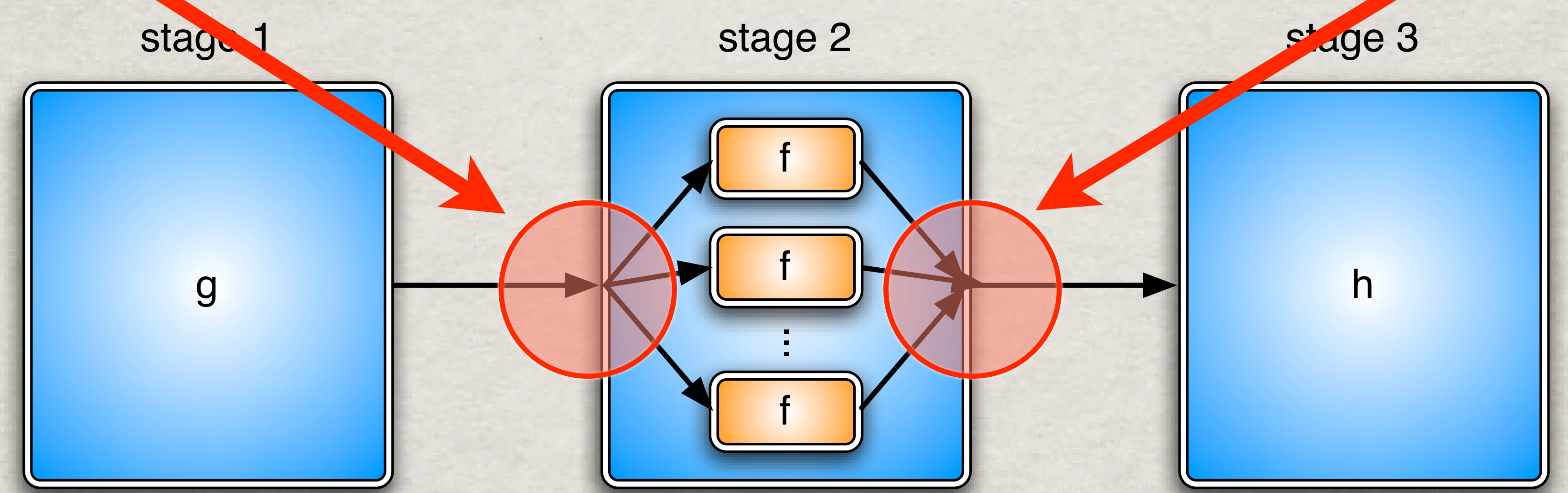
Our Fractal/ProActive experience (First 6 months)

- Understanding
  - Install, learn, understand Fractal & ProActive
  - Understand Fractal/Proactive architecture
    - Documentation; not layered architecture
  - Fractal interoperability
    - Proactive vs Julia implementations
    - AOP with Fractlet
- Case study
  - Self-optimizing only (performance)
  - pipe(S1, Farm(S2), S3)
  - Fractal/ProActive features to support NF control

Self-optimizing Pipe(g,Farm(f),h)

User programmable unicast

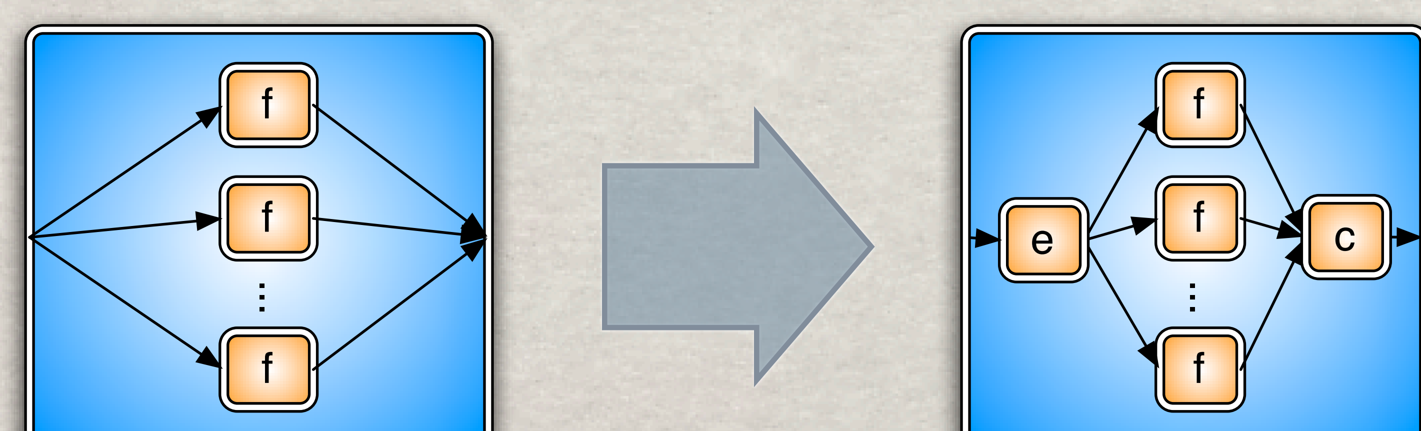
Collects from any



- A simple three stages application, working on a data stream (e.g. video frames)
  - pipe performance max(Tg,Tfarm(f),Th)
  - farm performance Tf/#n, n variable along run
- Self-optimizing w.r.t. nodes power along time

Farm

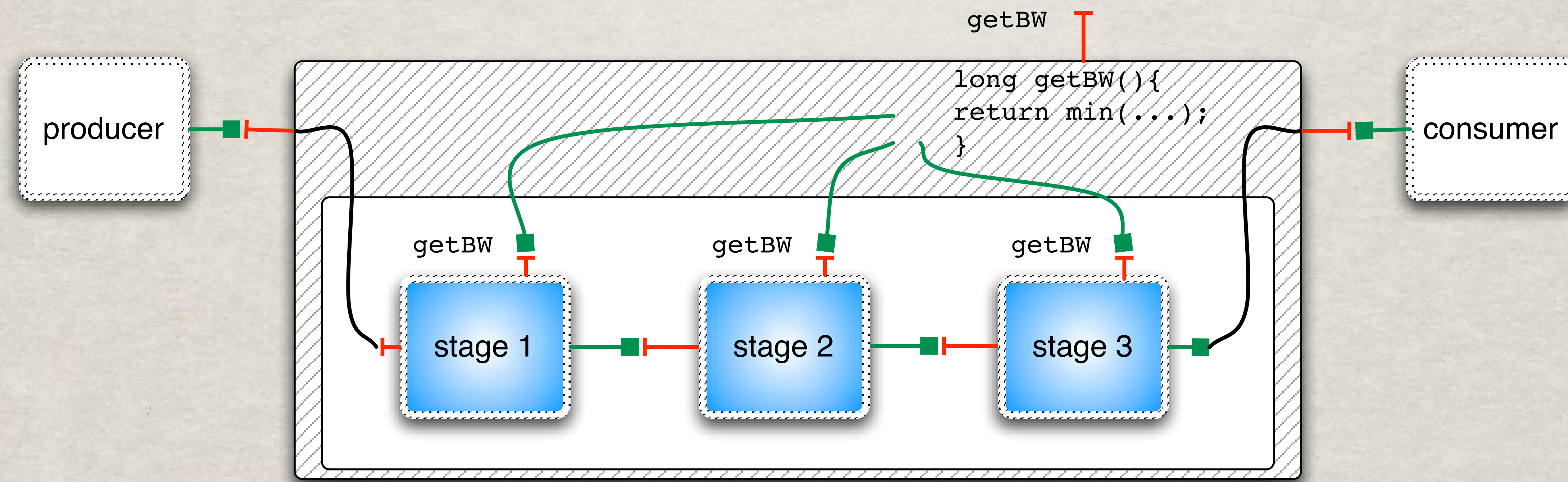
- A clean implementation needs:
  - Unicast “programmable” communications
    - send to a single ID in a set, collect from any (not all)
    - probably not excluded by GCM specification, not clear our to implement in the current version
  - Distributed implementation of the membrane
    - is it a single Active Objects?
- Currently two inner components act as distributor and collector



Pipe

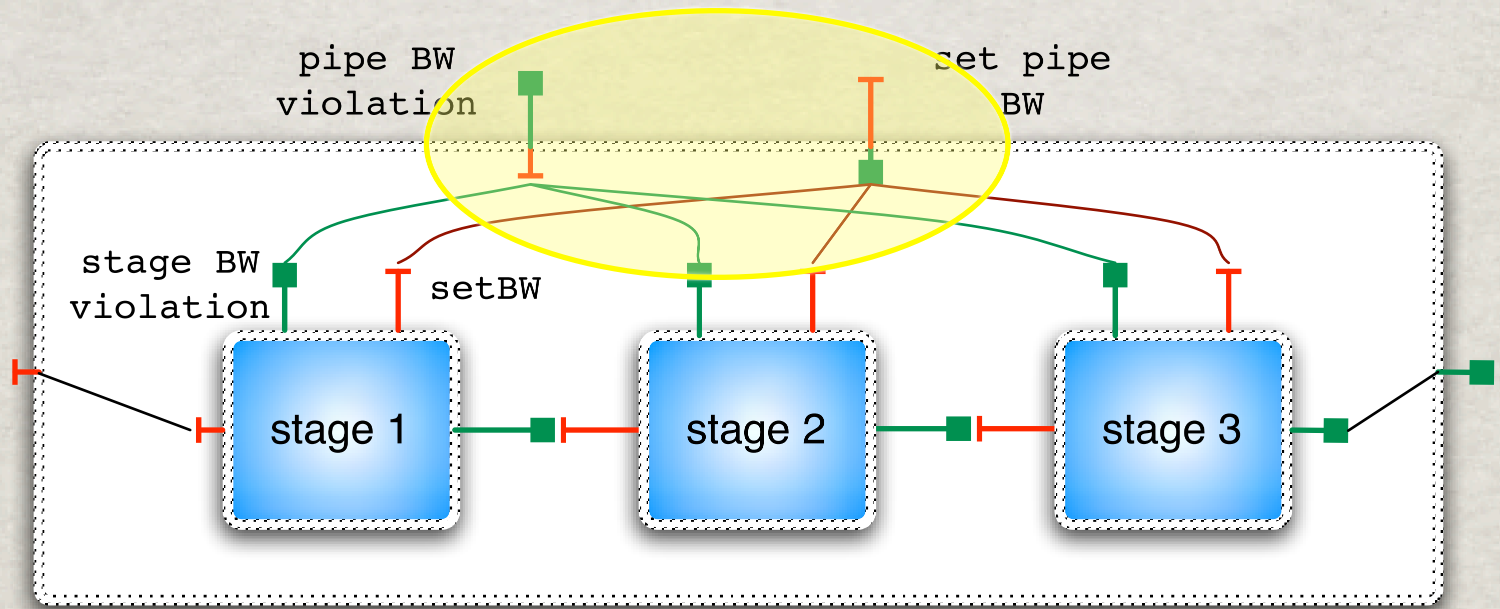
- Two versions
  - Passive inner components
    - Each component exposes server NF interface (GetBandwidth)
    - They are periodically polled from a controller in the membrane, which then expose a GetBandwidth server port for the pipe component
    - Implementation pretty tricky, polling is programmed at hand within the controller
  - Active inner components
    - How to open server ports on the membrane toward the inner part (import-binding)? Is it possible?
    - We simulated with a functional component
  - Both versions expose all ports through a single JVM
    - Membrane and Active Objects

Pipe with Passive NF stages

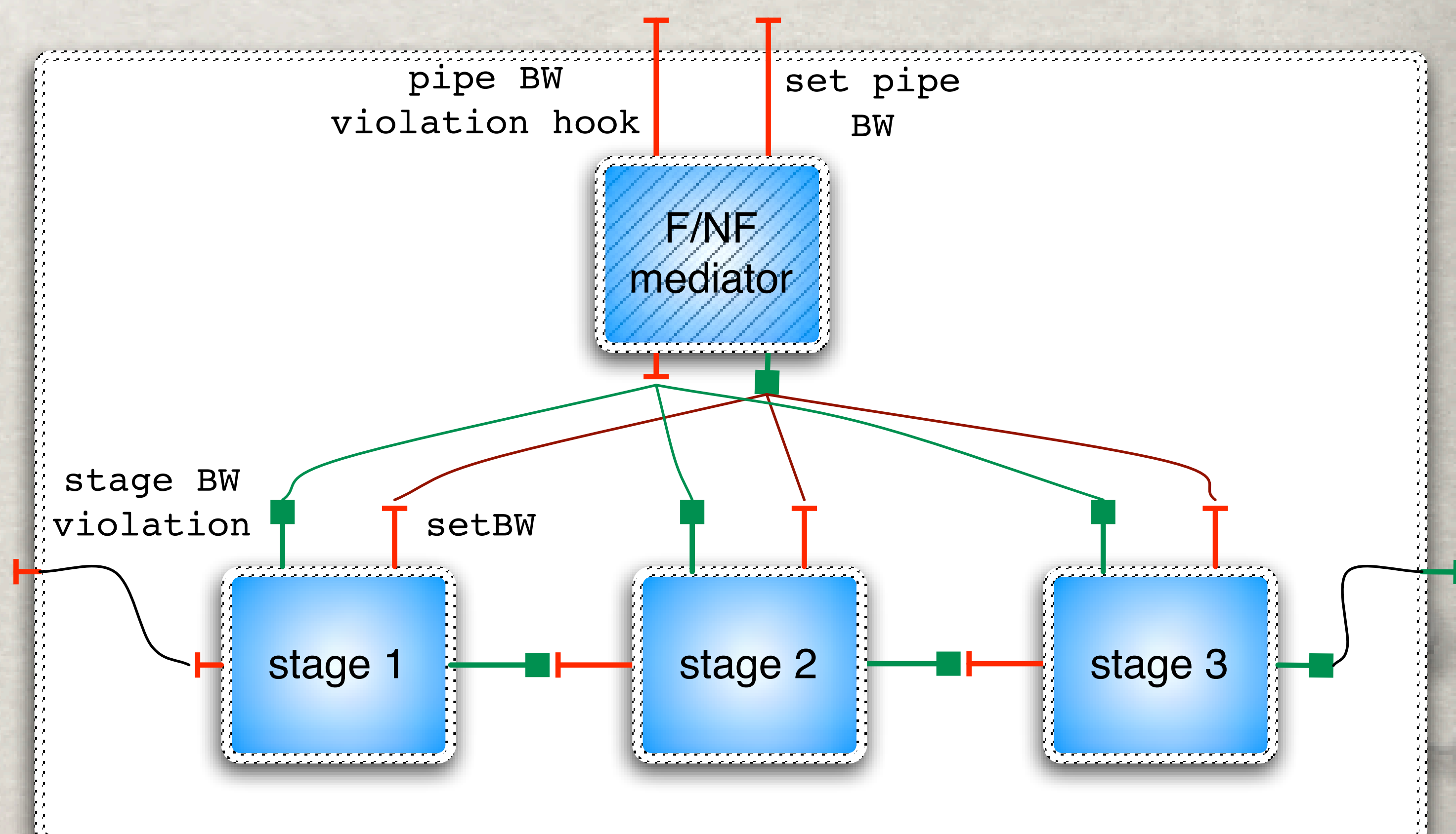


- Implemented, works
  - Overheads not yet measured
- Managing code completely up to the user
  - NF binding programmatically described

Pipe with Active NF Stages



- Not succeed to express this
  - Maybe not impossible, but we don't succeeded in several weeks
  - Can be simulated by inserting an functional component (explicit manager)
  - Import/export bindings for NF controllers appears under-specified (-studied, -implemented



Points needing further investigation

- Programming controllers
  - GCM specification should be refined
- Interactions among controllers
  - Ports exposed by controllers, toward in and out
  - Interaction among ports
- Mapping membrane & controllers
  - VN, ActiveObjects, JVM, nodes, ...
- Low-level points
  - Sent to Proactive Q&A

Conclusion

- High-level research issues
  - Formalization of QoS property ongoing
  - Interaction among managers is still a black hole
- Implementation issues
  - Middleware expressiveness/effectiveness tradeoff can (should?) be improved
  - Low-level issues submitted to Proactive Q&A
  - Layering of features
    - In our idea, some of middleware features may require a promotion to QoS features (e.g. load balancing, communication synchronicity, group communication semantics, security ...) because they are supposed to be dependent by semantics of GCM application not on ProActive