Components, GCM, and Behavioural skeletons

Marco Aldinucci
University of Pisa, Italy
(CoreGRID REP Programme)

M. Danelutto, S. Campa
University of Pisa, Italy

P. Kilpatrick
QUB, UK

N. Tonellotto, P.Dazzi
ISTI-CNR, Italy

November 22th, 2007
QUB, Belfast, UK

Outline

- Prelude
  - Uni. Pisa and the HPC lab.
- Motivation
  - why adaptive and autonomic management
  - why skeletons
- Behavioural Skeletons
  - parametric composite component with management
  - functional and non-functional description
  - families of behavioural skeletons
- GCM implementation
  - preliminary experiments and performances

Pisa Computer Science Department & Parallel arch. Lab

- Computer Science Dept.
  - First in Italy (estab. 1968)
  - Research and teaching
    - Bachelor, master, and PhD programme
    - ~ 70 tenures + lot of fellows
- Parallel architecture lab. (current)
  - 1 Full Prof. (M. Vanneschi)
  - 1 Associate Prof. (M. Danelutto)
  - 2 Researchers (**M. Aldinucci**, M. Coppola)
  - 1 PostDoc (S. Campa)
  - 2 Phd students (M. Meneghin, C. Bertolli),
  - 2 senior engineers (M. Torquati, R. Ravazzolo)
  - 4 junior engineers + several master students (in thesis)

Participation in Projects (1997-2007)

Ongoing

- IN.SY.EME **(MIUR-IT FIRB)** Integrated System for Emergency - Jul. 2007, 36 m
- FRIMP **(Cassa di Risparmio di Pisa)** Software for Network Processors Feb. 2007, 24 m
- VirtuaLinux **(Eurotech SpA)** Roboust Virtual Clutering - Nov. 2006, 6 m
- BEinGRID **(EU-IP, 6th FP)** The Grid infrastructure for the Retail Management Experiment - Jun. 2006, 18 m
- XtreemOS **(EU-IP, 6th FP)**: Building and Promoting a Linux-based Operating System to Support Virtual Organisations for Next Generation Grids - Jun. 2006, 48 m
- GridComp **(EU-STREP, 6th FP)** Grid Component Model - June 2006, 30 m
- SFIDA **(MIUR FAR-ICT)**: Innovative platform supporting collaborative-business for Small-Medium Enterprises - Sept. 2007, 24 m
- CoreGrid **(EU-Network of Excellence, 6th FP)**: Foundations, Software Infrastructures and Applications for large scale distributed, Grid and Peer-to-Peer Technologies - 2004, 48 m

Completed

- Galieo Pisa-ParisVII/INRIA (Exchange Programme) 2004 - 2006
- MOPROSCO Pisa-ParisVII/INRIA (Exchange Programme) 2005 - 2007
- Grid.it (MIUR FIRB) 2003 - 2006
- GridCoord (EU-Special Action, 6th FP) 2004 - 2006
- Vigoni Pisa-Berlino/Muenster (Exchange Programme) 2003 - 2005
- SAIB (Ricerca Industriale MIUR) 2002 - 2004
- Law 449/97 year 2000 (strategic projects MIUR-CNR) 2002 - 2004
- Law 449/97 year 1999 (strategic projects MIUR-CNR) 2002 - 2004
- ASI-PQE2000 (MIUR) 2001- 2002
- Agenzia2000 (MIUR) 2000-2002
- Vigoni Pisa-Passau (Exchange Programme) 1998 - 2000
- MOSAICO (MIUR 40%) 1998 - 2000
- PQE2000 (CNR, ENA, INFN, Alenia Spazio) 1997 - 2000

Scientific Productivity of the Lab (1997-2007)

- Research & dissemination
  - 21 intl. journals (8 A-class), 35 intl. conferences (20 A-class), 26 intl. workshops & symposium, 12 parts of books, served as editors for several journal and books, 2 large conferences organised (400+ attendees), several invited talks
- Software (open source & copyrighted)
  - 2 full programming environments for parallel languages
    - with language compiler: SkiE, ASSIST
  - several libraries for parallel programming
    - on top of Java, C, C++, Fortran, MPI, ACE, sockets, shmem, ...
  - servers and applications
    - distributed shared memory & storage, web server farm, // datamining, ...
  - cluster virtualization, cluster robustness, storage virtualization
    - VirtuaLinux

CGM 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

Why Autonomic Computing

- // programming & the grid
  - concurrency exploitation, concurrent activities set up, mapping/ scheduling, communication/synchronisation 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, ...

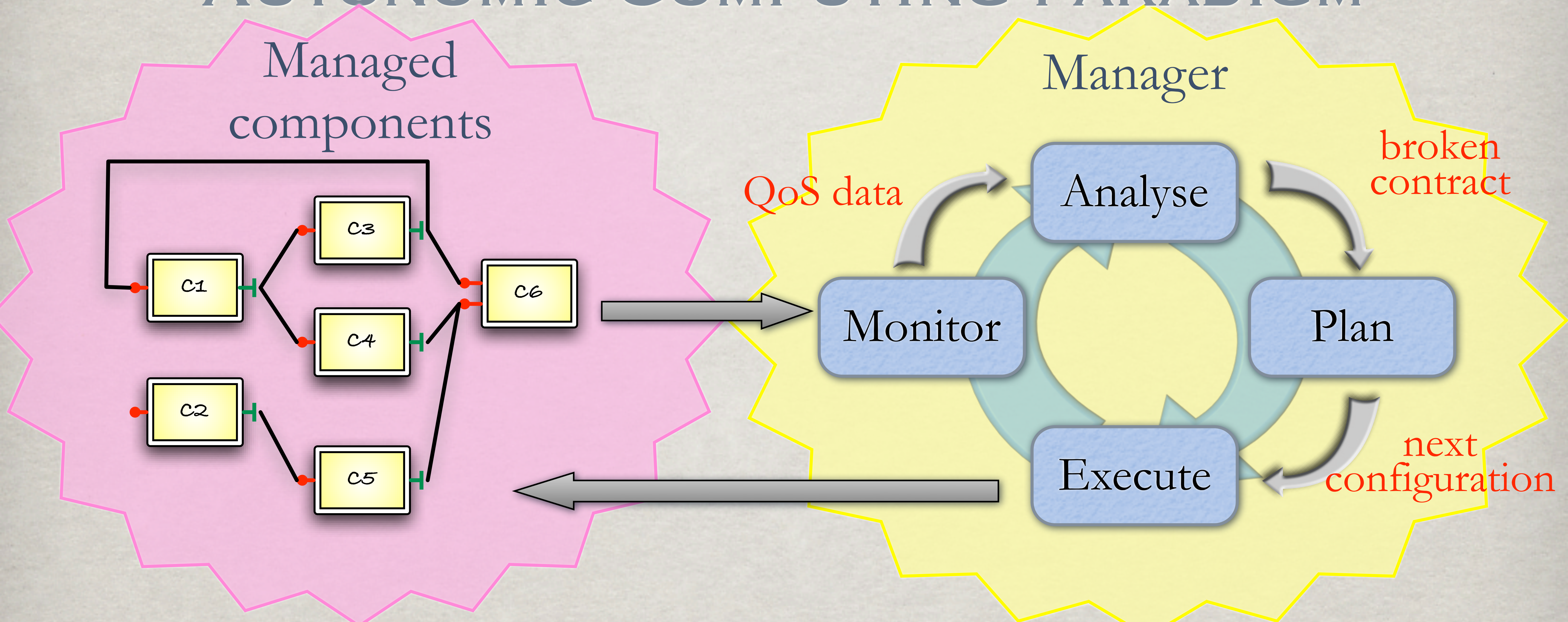
... and a non trivial QoS for **applications**

not easy leveraging only on middleware

our approach:

high-level methodologies + tools

Autonomic Computing paradigm



- monitor: collect execution stats: machine load, service time, input/output queues lengths, ...
- analyse: instantiate performance models with monitored data, detect broken contract, in and in the case try to detect the cause of the problem
- plan: select a (predefined or user defined) strategy to re-convey the contract to validity. The strategy is actually a "program" using execute API
- execute: leverage on mechanism to apply the plan

Why skeletons 1/2

- Management is difficult
  - Application change along time (ADL not enough)
  - How "describe" functional, non-functional features and their inter-relations?
  - The low-level programming of component and its management is simply too complex
- Component reuse is already a problem
  - Specialising component yet more with management strategy would just worsen the problem
  - Especially if the component should be reverse engineered to be used (its behaviour may change along the run)

Why skeletons 2/2

- Skeletons represent patterns of parallel computations (expressed in GCM as graphs of components)
- Exploit the inherent skeleton semantics
  - thus, restrict the general case of skeleton assembly
  - graph of any component ➠ parametric networks of components exhibiting a given property
  - enough general to enable reuse
  - enough restricted to predetermine management strategies
- Can be enforced with additional requirements
  - E.g.: Any adaptation does not change the functional semantics

Behavioural Skeletons idea

- Represent an evolution of the algorithmic skeleton concept for component management
  - abstract parametric paradigms of component assembly
  - specialized to solve one or more management goals
    - self-configuration/optimization/healing/protection.
- Are higher-order components
- Are not exclusive
  - can be composed with non-skeletal assemblies via standard components connectors
    - overcome a classic limitation of skeletal systems

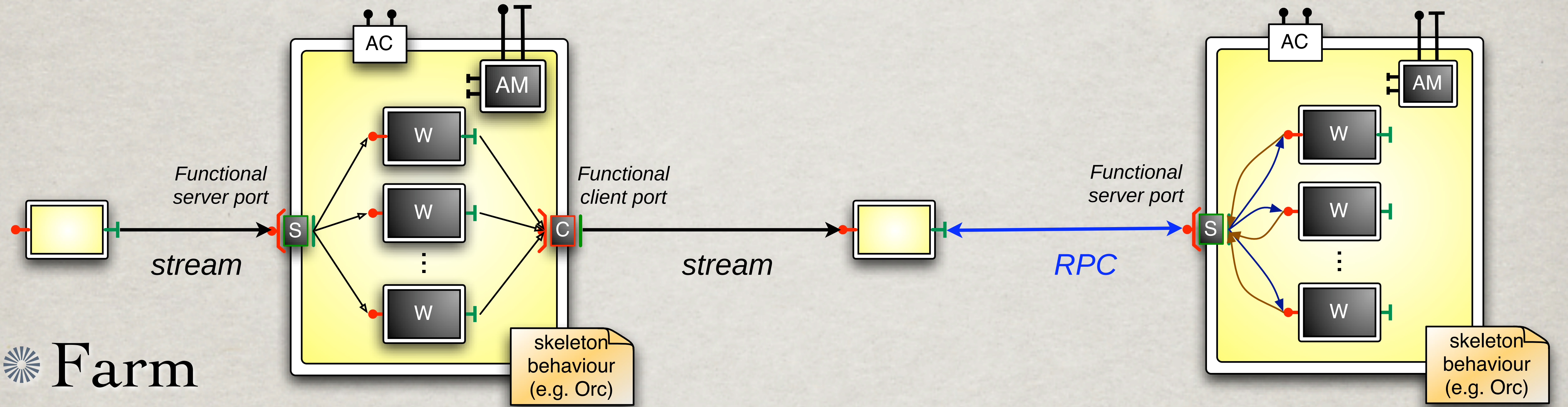
Behavioural Skeletons proprieties

- Expose a description of its functional behaviour
- Establish a parametric orchestration schema of inner components
- May carry constraints that inner components are required to comply with
- May carry a number of pre-defined plans aiming to cope with a given self-management goal
- Carry an implementation (they are factories)

Be-Skeletons families

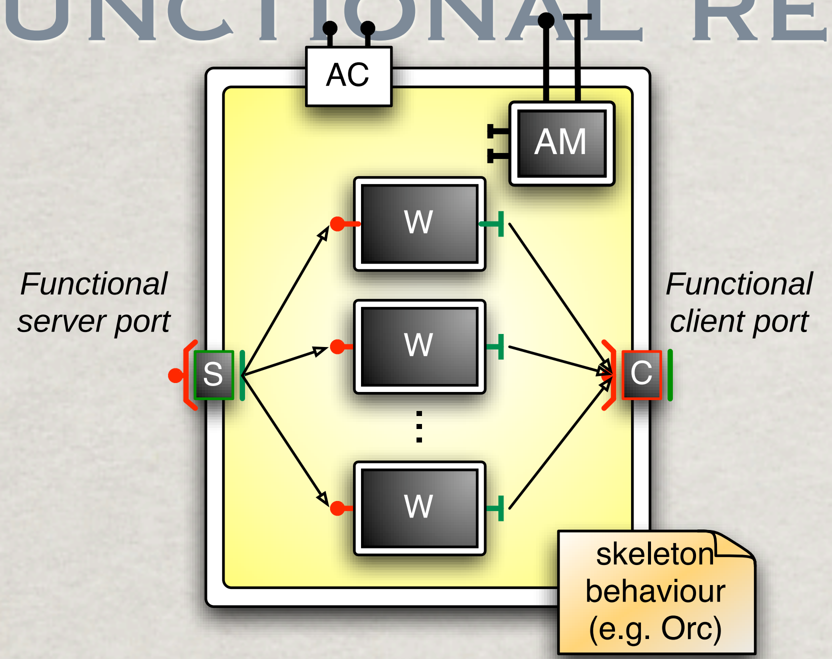
- Functional Replication
  - Farm/parameter sweep (self-optimization)
  - Simple Data-Parallel (self-configuring map-reduce)
  - Active/Passive Replication (self-healing)
- Proxy
  - Pipeline (coupled self-protecting proxies)
- Wrappers
  - Facade (self-protection)
- Many others can be borrowed from Design Patterns

Functional replication



- Farm
  - S = unicast, C = from_any, W = stateless inner component
- Data Parallel
  - S = scatter, C = gather, W = stateless inner component
- Fault-tolerant Active Replication
  - S = broadcast, C = get_one_in_a_set, W= stateless inner ...
- ...

Functional replication



Functional behaviour description (orchestration)

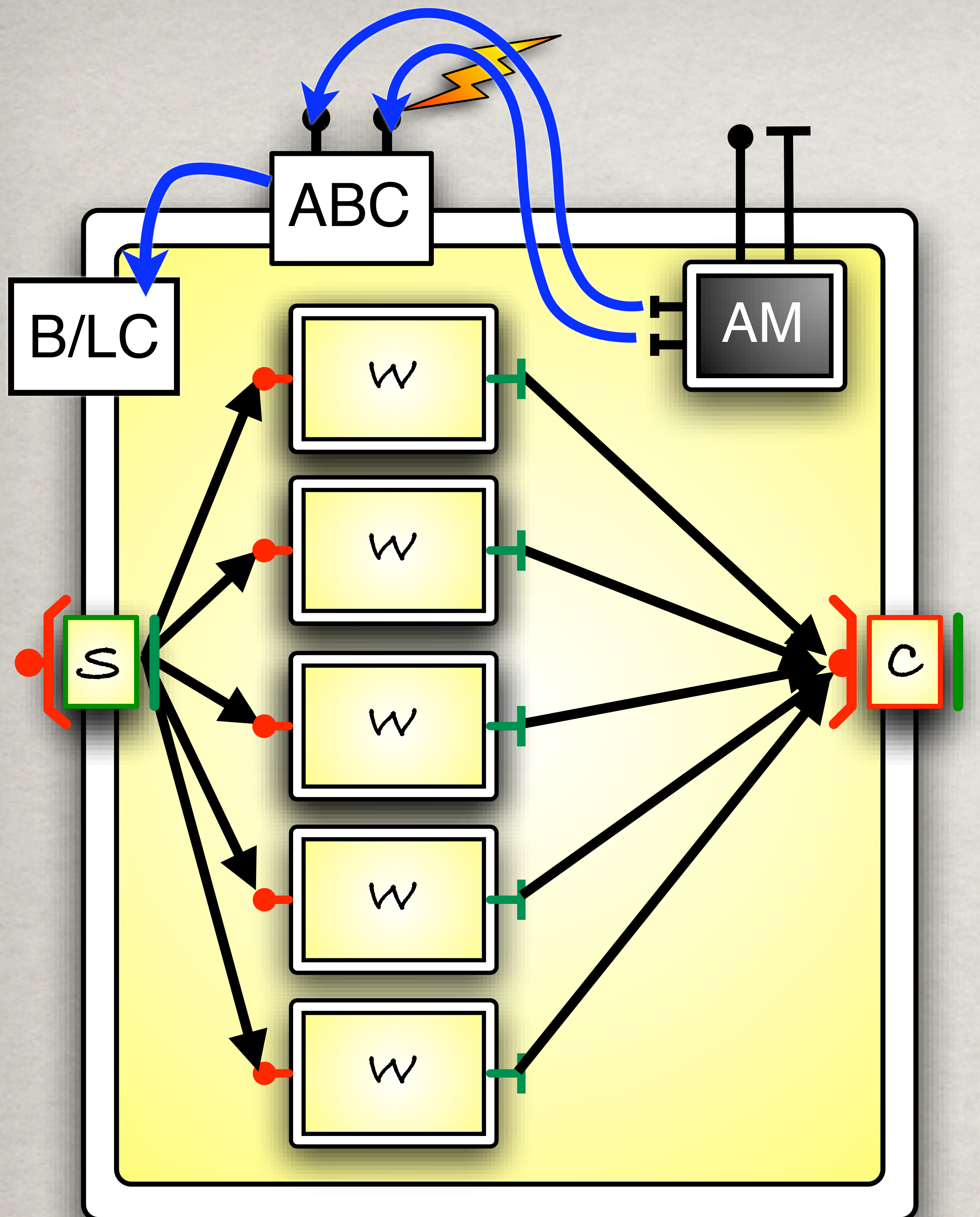
$$system(data, S, G, W, in, out, N) \triangleq$$
$$S(data, in) \mid (| \; i : 1 \le i \le N : W_i(in_i, out_i)) \mid C(out)$$
$$W_i(in_i, out_i) \triangleq$$
$$in_i.get > tk > process(tk) > r > (out_i.put(r) \mid W_i(in_i, out_i))$$

- Meant to parametrically expose all allowed adaptation
  - Any AM policy that does not change this semantics is *correct*
  - As an example changing ***i*** in this schema is correct
  - Functional semantics is invariant from ***i***, non-functional one is not (and changing ***i*** means changing the number of Ws for self-* purposes

GridCOMP

CoreGRID

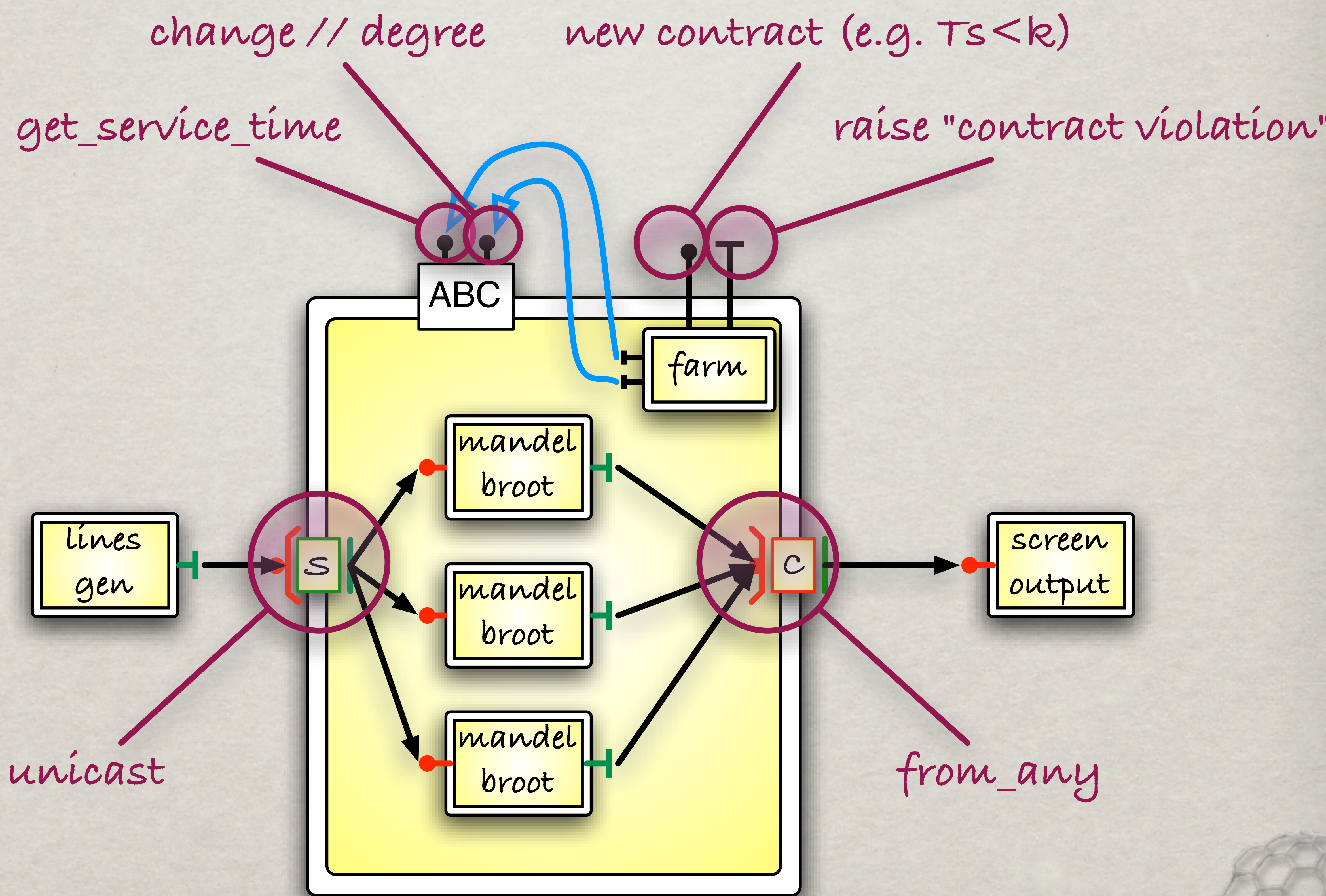
GCM IMPLEMENTATION

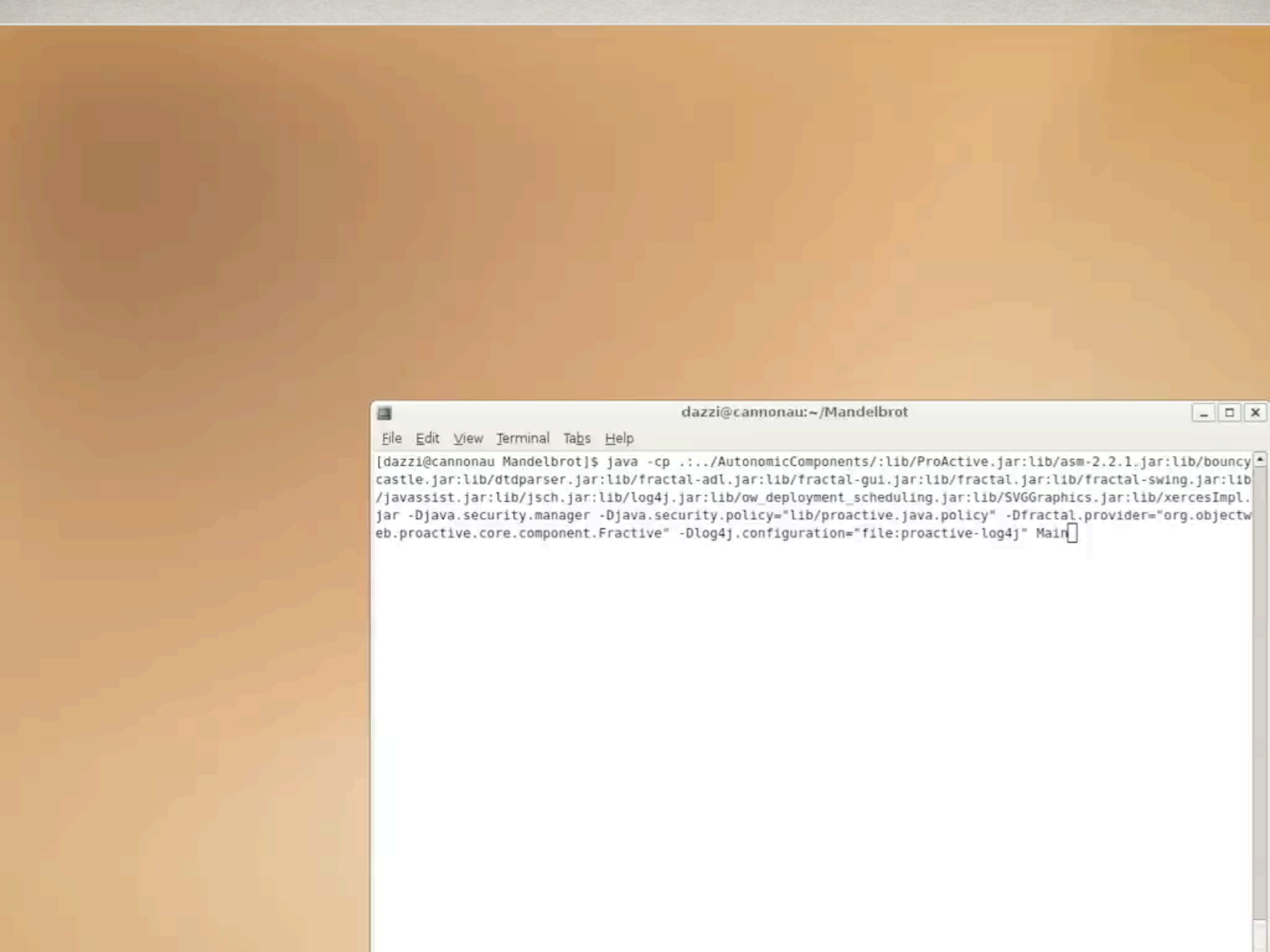


ABC = Autonomic Behaviour Controller (implements mechanisms)
AM = Autonomic Manager (implements policies)
B/LC = Binding + Lifecycle Controller

1. Choose a schema (.e.g. functional replication) ABC API is chosen accordingly
2. Choose an inner component (compliant to Be-Ske constraints)
3. Choose behavior of ports (e.g. unicast/from_any, scatter/gather)
4. Wire it in your application. Run it, then trigger adaptations
5. Possibly, automatize the process with a Manager

Farm example (Mandelbroot)

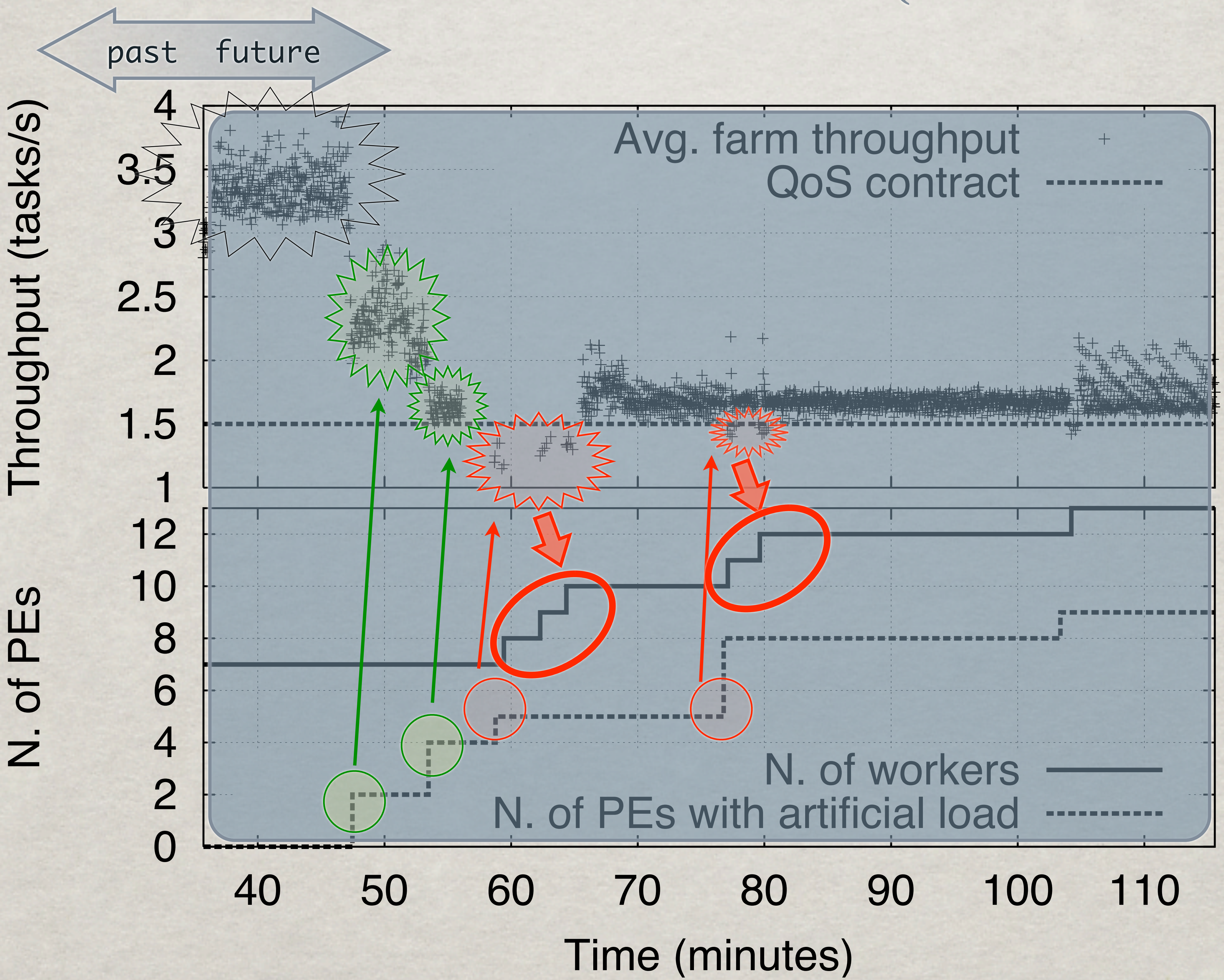



dazzi@cannonau:~/Mandelbrot
File Edit View Terminal Tabs Help
[dazzi@cannonau Mandelbrot]$ java -cp .:../AutonomicComponents/:lib/ProActive.jar:lib/asm-2.2.1.jar:lib/bouncy
castle.jar:lib/dtdparser.jar:lib/fractal-adl.jar:lib/fractal-gui.jar:lib/fractal.jar:lib/fractal-swing.jar:lib
/javassist.jar:lib/jsch.jar:lib/log4j.jar:lib/ow_deployment_scheduling.jar:lib/SVGGraphics.jar:lib/xercesImpl.
jar -Djava.security.manager -Djava.security.policy="lib/proactive.java.policy" -Dfractal.provider="org.objectw
eb.proactive.core.component.Fractive" -Dlog4j.configuration="file:proactive-log4j" Main

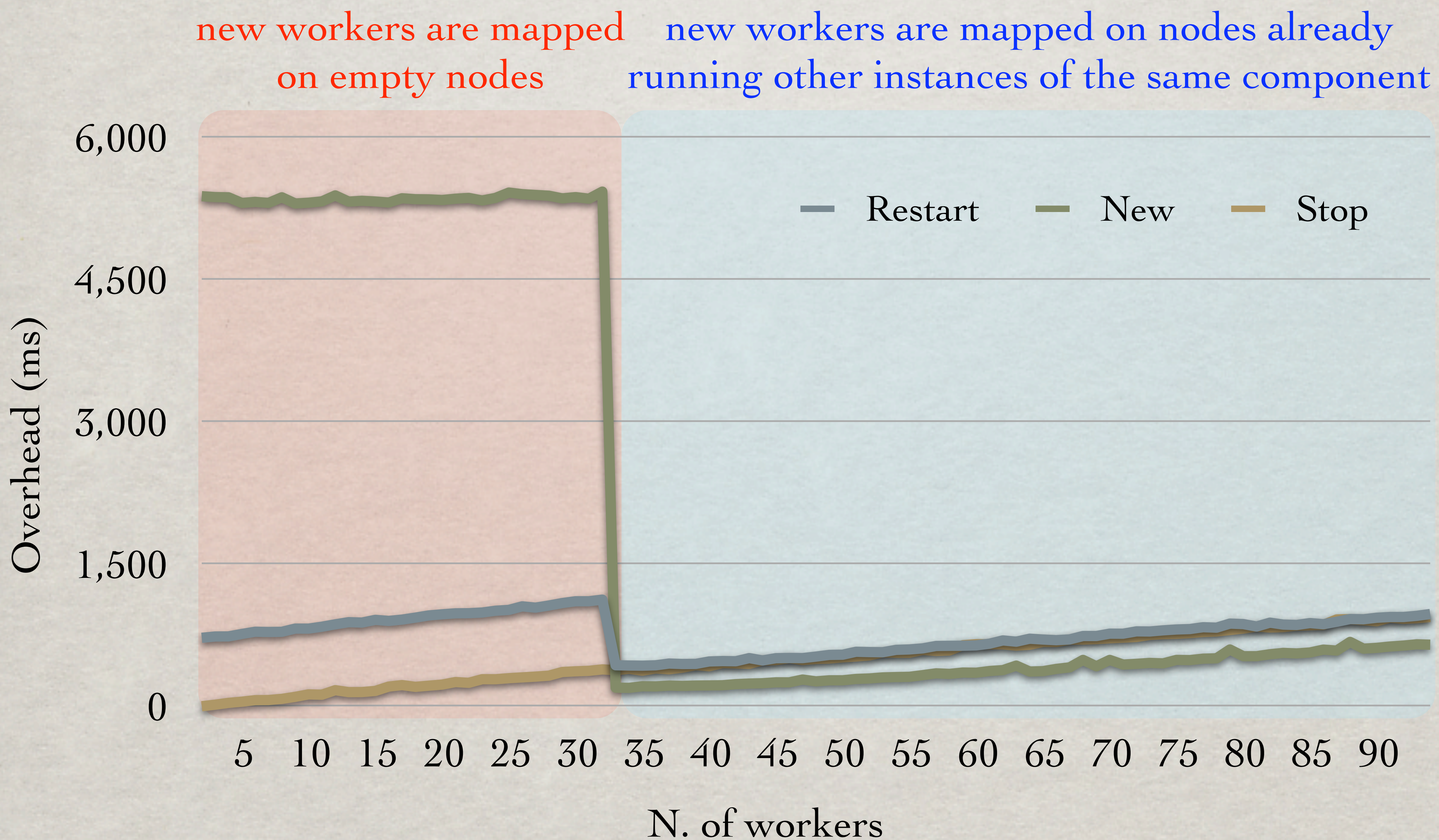
Not just farm (i.e. param sweep)

- Many other skeletons already developed for GCM
  - some mentioned before
- Easy extendible to stateful variants
  - imposing inner component expose NF ports for state access
- Policies not discussed here
  - expressed with a `when-event-if-cond-then-action` list of rules
  - some exist, work ongoing ...

Typical Log of a Run (Explained)



Overheads


new workers are mapped
on empty nodes
new workers are mapped on nodes already
running other instances of the same component
6,000
4,500
3,000
1,500
0
Overhead (ms)
Restart
New
Stop
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
N. of workers

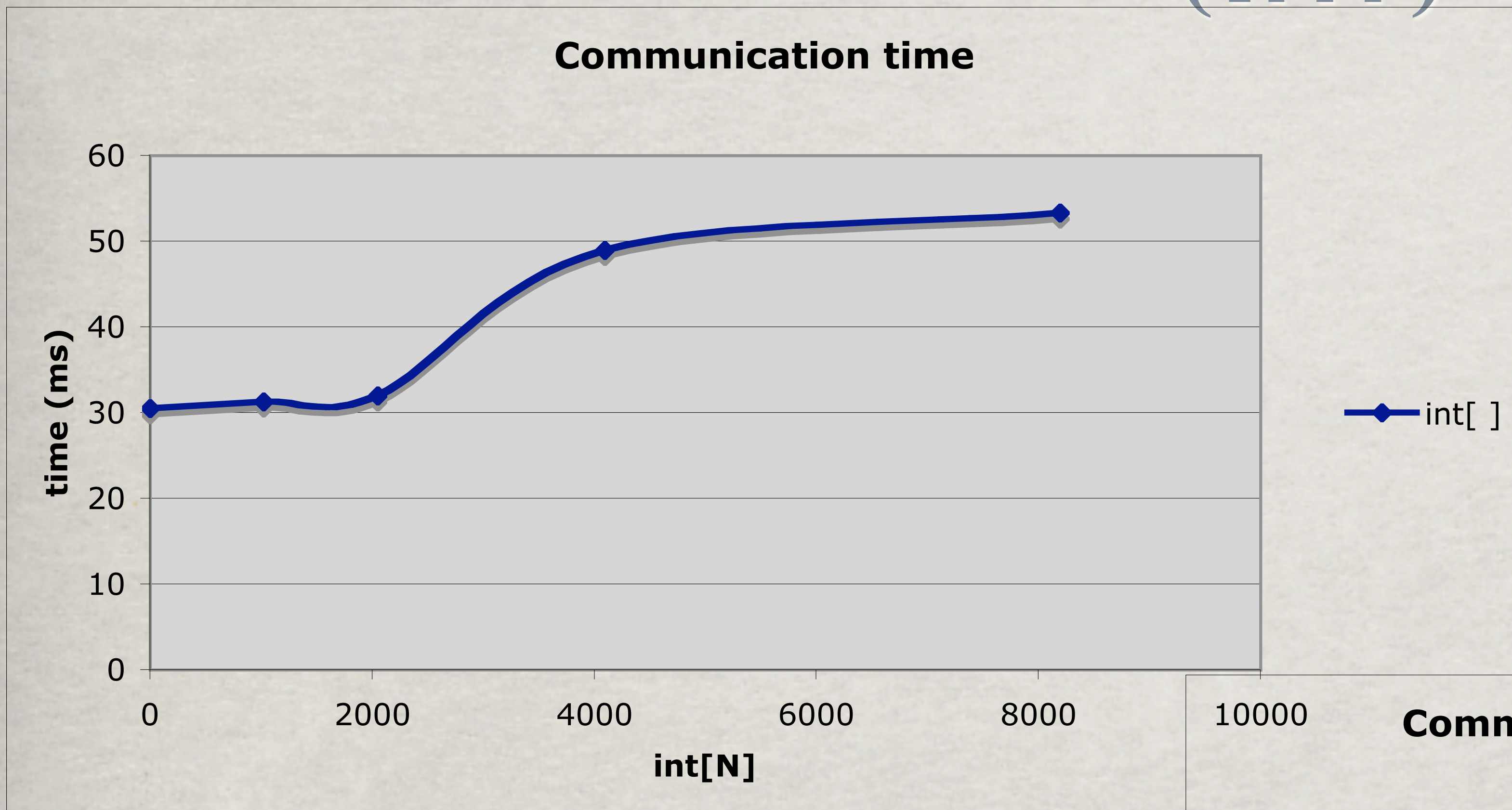
Proactive/Java Appears quite heavyweight w.r.t. other approaches

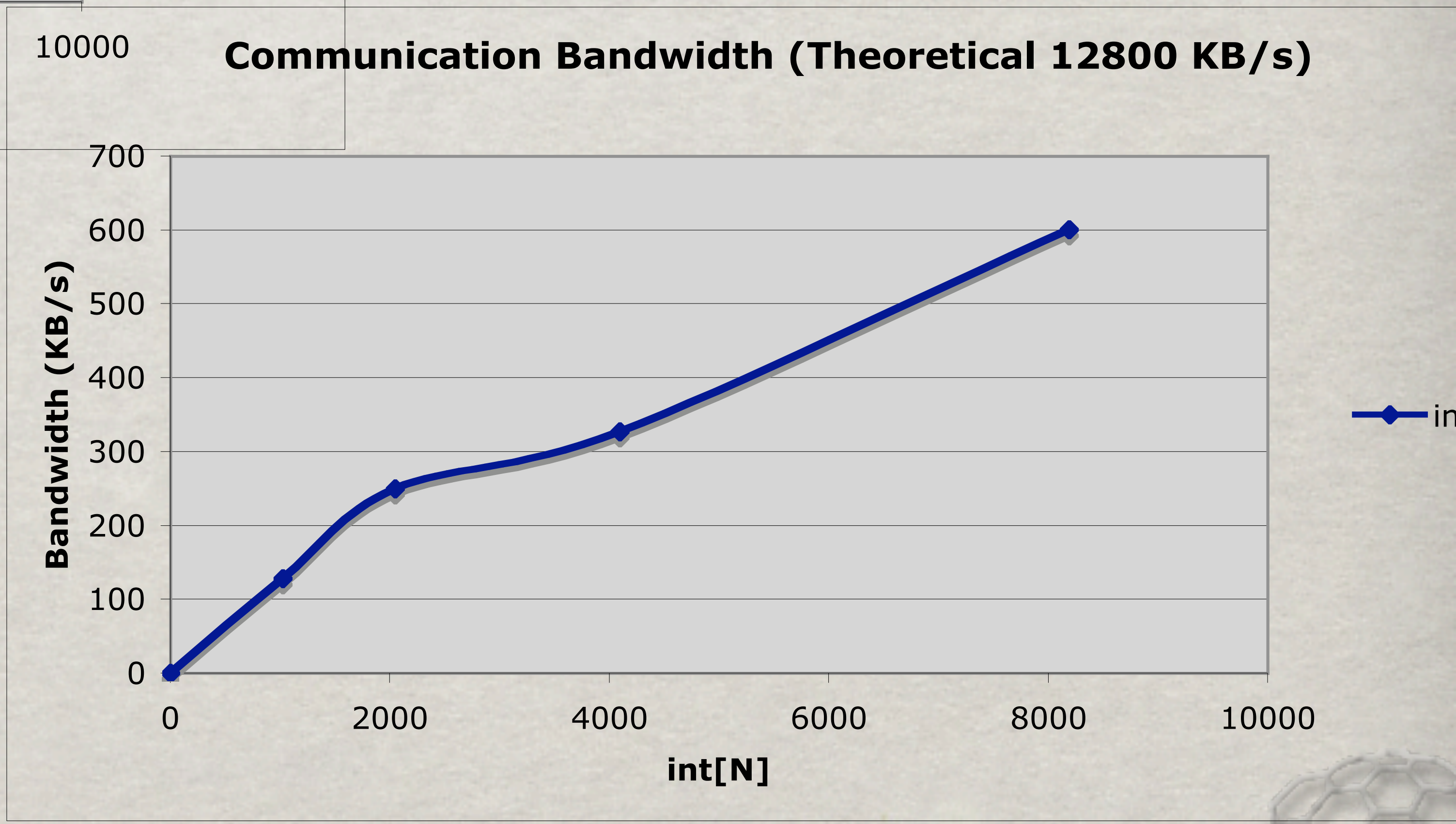
ASSIST/C++ overheads (ms)

M. Aldinucci, A. Petrocelli, E. Pistoletti, M. Torquati, M. Vanneschi, L. Veraldi, and C. Zoccolo. Dynamic reconfiguration of grid-aware applications in ASSIST. Euro-Par 2005, vol. 3648 of LNCS, Lisboa, Portugal. Springer Verlag, August 2005.

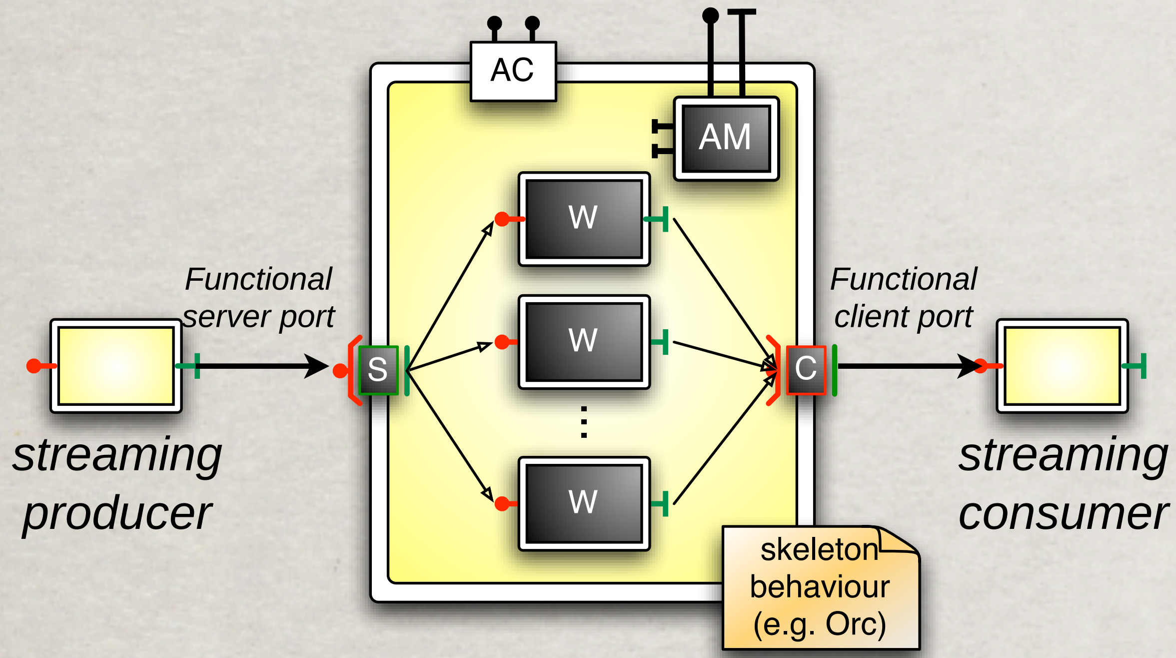
parmod kind	Data-parallel (with shared state)						Farm (without shared state)					
reconf. kind	add PEs			remove PEs			add PEs			remove PEs		
# of PEs involved	1→2	2→4	4→8	2→1	4→2	8→4	1→2	2→4	4→8	2→1	4→2	8→4
R_l on-barrier	1.2	1.6	2.3	0.8	1.4	3.7	–	–	–	–	–	–
R_l on-stream-item	4.7	12.0	33.9	3.9	6.5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

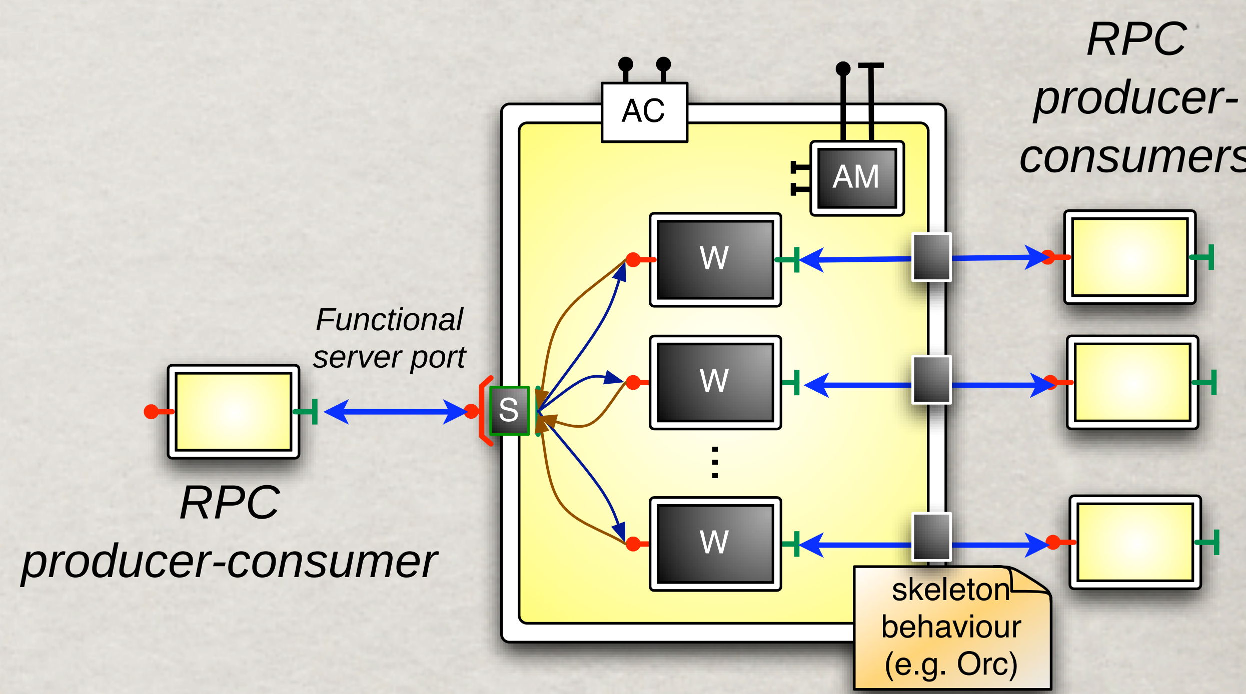
Proactive Communication Time (Int)



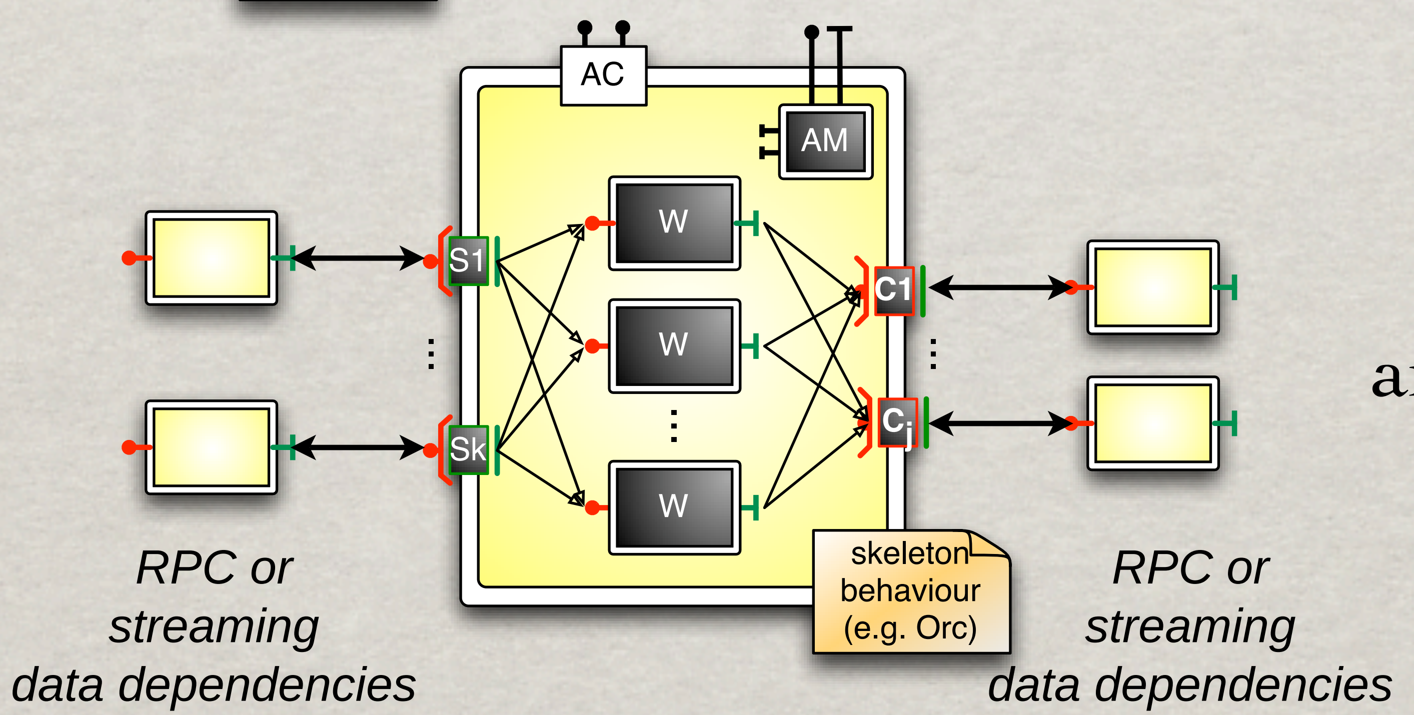


Variations and Flavours





or in general ...



and even more ...

Abstracting Out Variants

- *n* client and *y* server ports
  - synchronous and/or asynchronous
  - stream and/or RPC
  - programmable, possibly nondeterministic, relations among ports
    - wait for an item on port_A ***and/or*** one item on port_B
    - in general, any CSP expression
- But ... be careful, this is the **ASSIST** model
  - all features described above + distributed membrane + autonomicity, QoS contracts, limited hierarchy depth (i.e. 2)
  - sophisticated C++ implementation, language not easy to modify
- GCM should be *enough* expressive and *not too* complex
  - we consider ASSIST as the complexity asymptote

Conclusions

- Behavioural Skeletons
  - templates with built-in management for the App designer
  - methodology for the skeleton designer
    - management can be changed/refined
    - just prove your own management is correct against skeleton functional description
  - can be freely mixed with standard GCM components
    - because once instanced, they are standard
- Already implemented on GCM
  - not happy about GCM runtime performances (can be improved)
    - We also implemented in ASSIST with different performances