GridCOMP
Effective Components for the Grids

WP3 - Perspective roadmap for autonomic management

Marco Aldinucci
Dept. Computer Science – University of Pisa
Programming Model Institute - CoreGRID

aldinuc@di.unipi.it

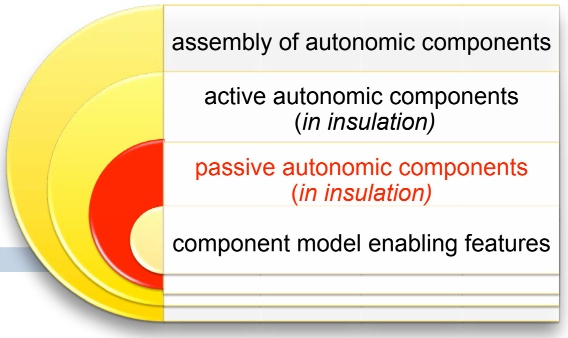
Unipi & ISTI-CNR working group

M. Danelutto, D. Laforenza, M. Aldinucci, N. Tonellotto, S. Campa, P. Dazzi, G. Zoppi

Outline

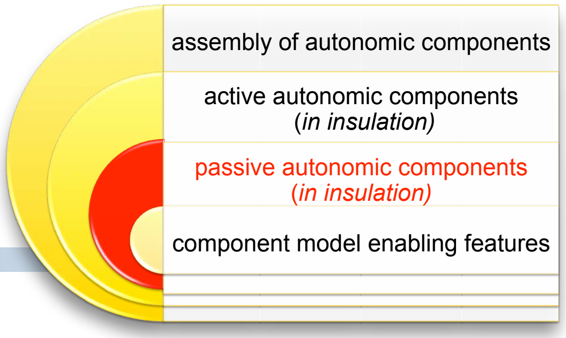
assembly of autonomic components	• distributed policies • orchestration of policies • formal tools, abstraction tools
active autonomic components *(in insulation)*	• behavioural skeletons • management policies
passive autonomic components *(in insulation)*	• behavioural skeletons • reconfiguration operations • component specific and infrastructural monitoring sensors
component model enabling features	• lightweight primitives • dynamic binding • suitable life-cycle control

Passive Autonomic Components



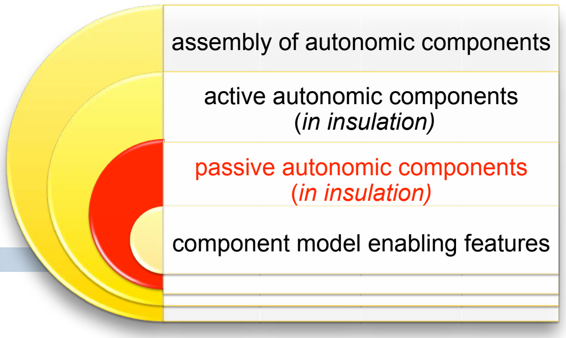
- BeSke are parametric composite components
  - extend controllers with ABC & MC
    - non-functional ports for reconfiguration & monitoring
  - implements a modified LC
    - because the manager is a component that cannot be stopped
  - currently provide several flavours of inner components replication
    - stateless farm, stateless farm with initialization, stateless dataparallel, stateful dataparallel
  - enough to cover use cases
- accomplishment state: 95%
  - remaining 5% is mostly related to code reengineering

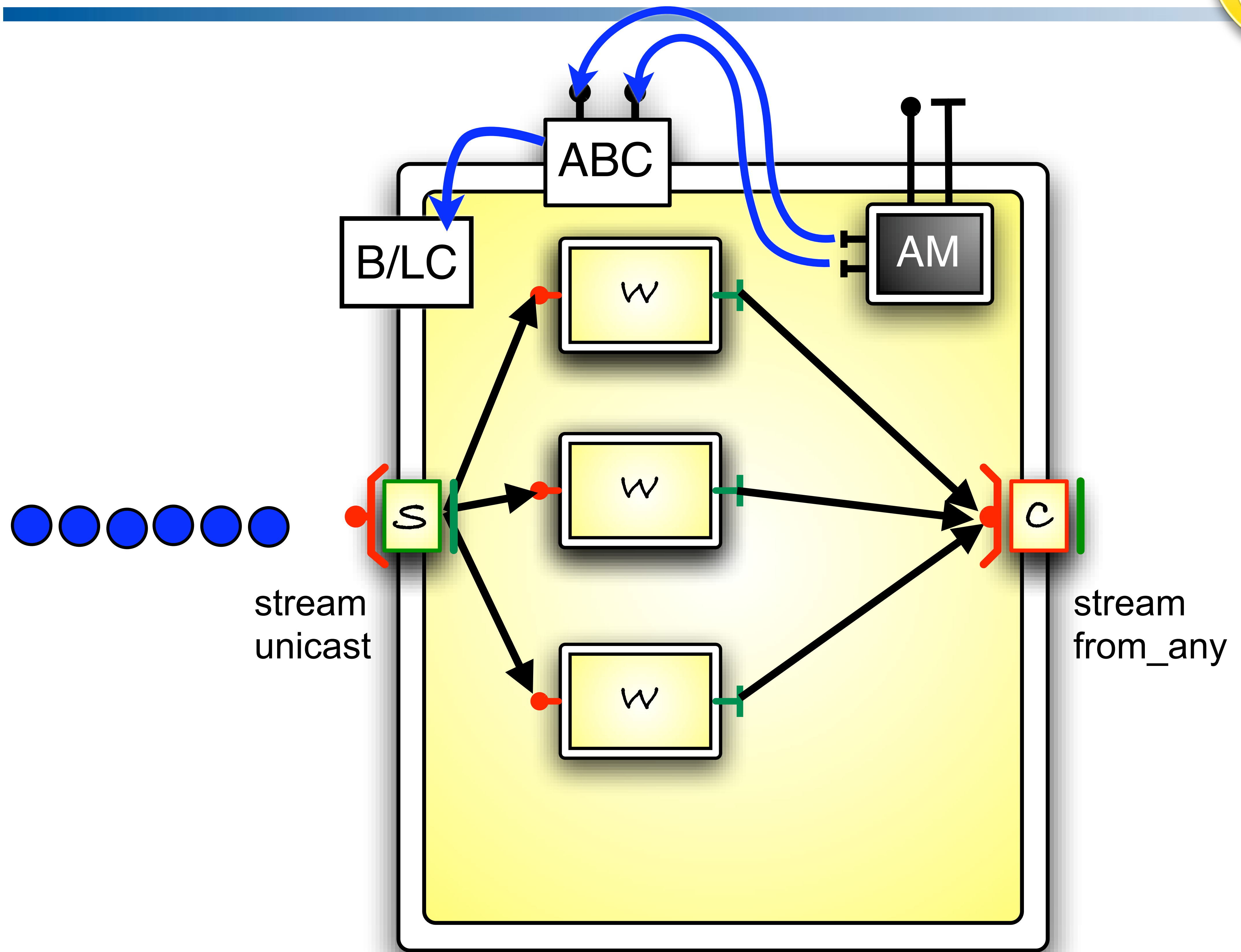
Farm (functional replication)



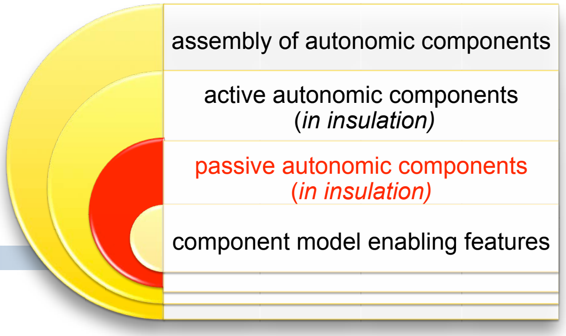
- farm BeSke
  - stateless
  - composite with a pair of **streaming** unicast/from_any ports
  - support composite workers
  - provide parallelism degree change, load-balancing, etc ...
  - autonomic policy for performance assessed
    - for performance at least (in insulation)
  - support several use case
    - e.g. Atos
    - just wrap the sequential code in a primitive component, and farm it out
  - see Nicola's talk & demo

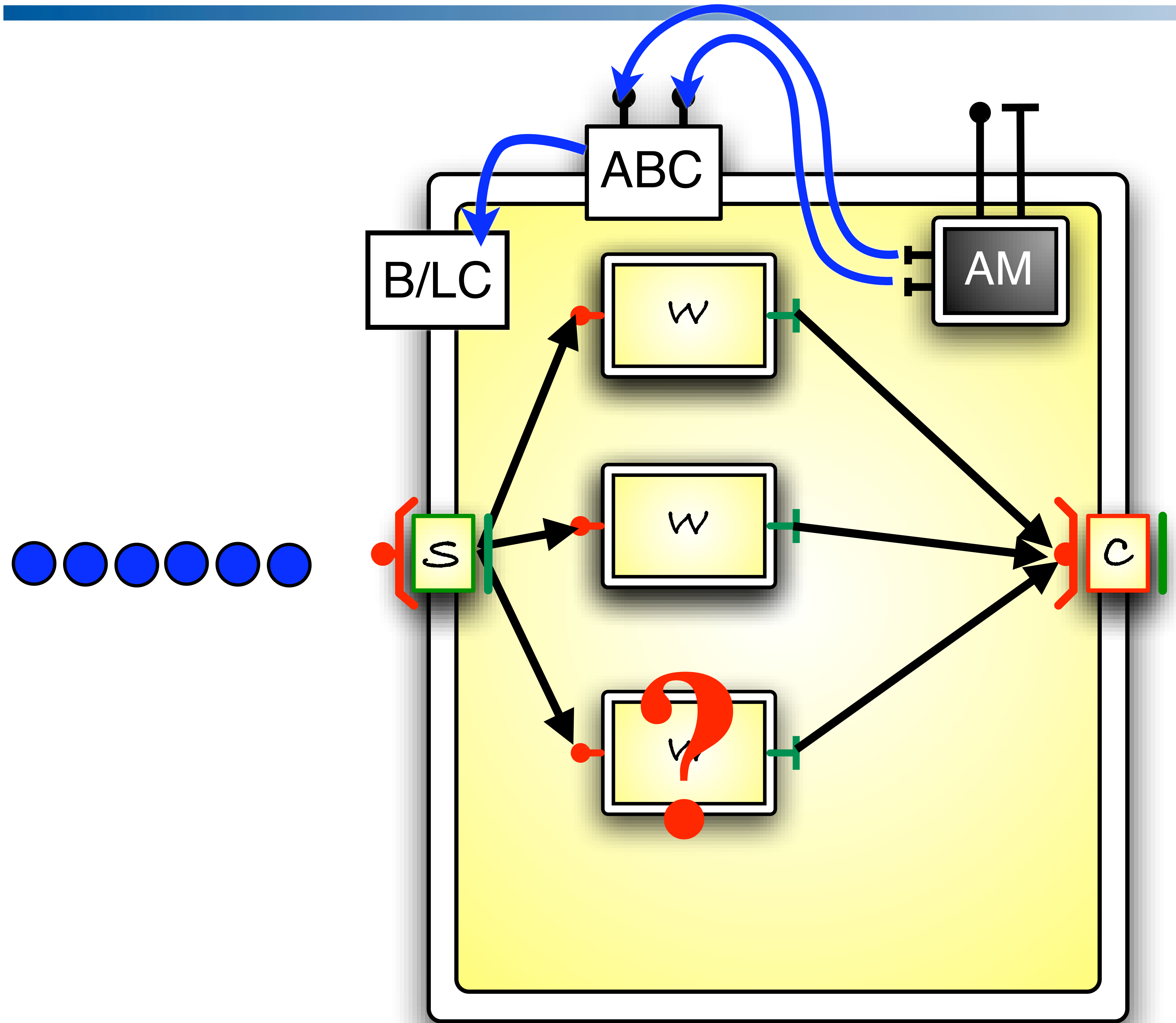
farm (in theory ...)





farm (in reality ...)

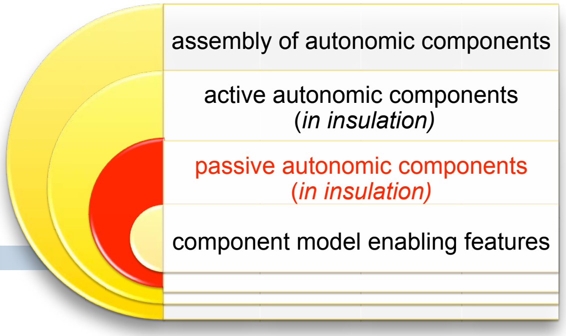




Notes

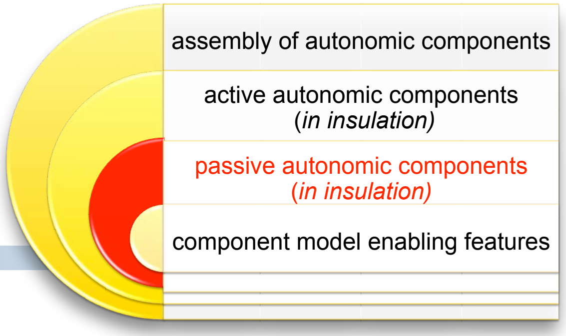
- rebalancing operation is logically not needed and it has been introduced because tasks cannot be transiently stopped at the composite membrane
- not sure it works with non-streaming ports (not void return type)

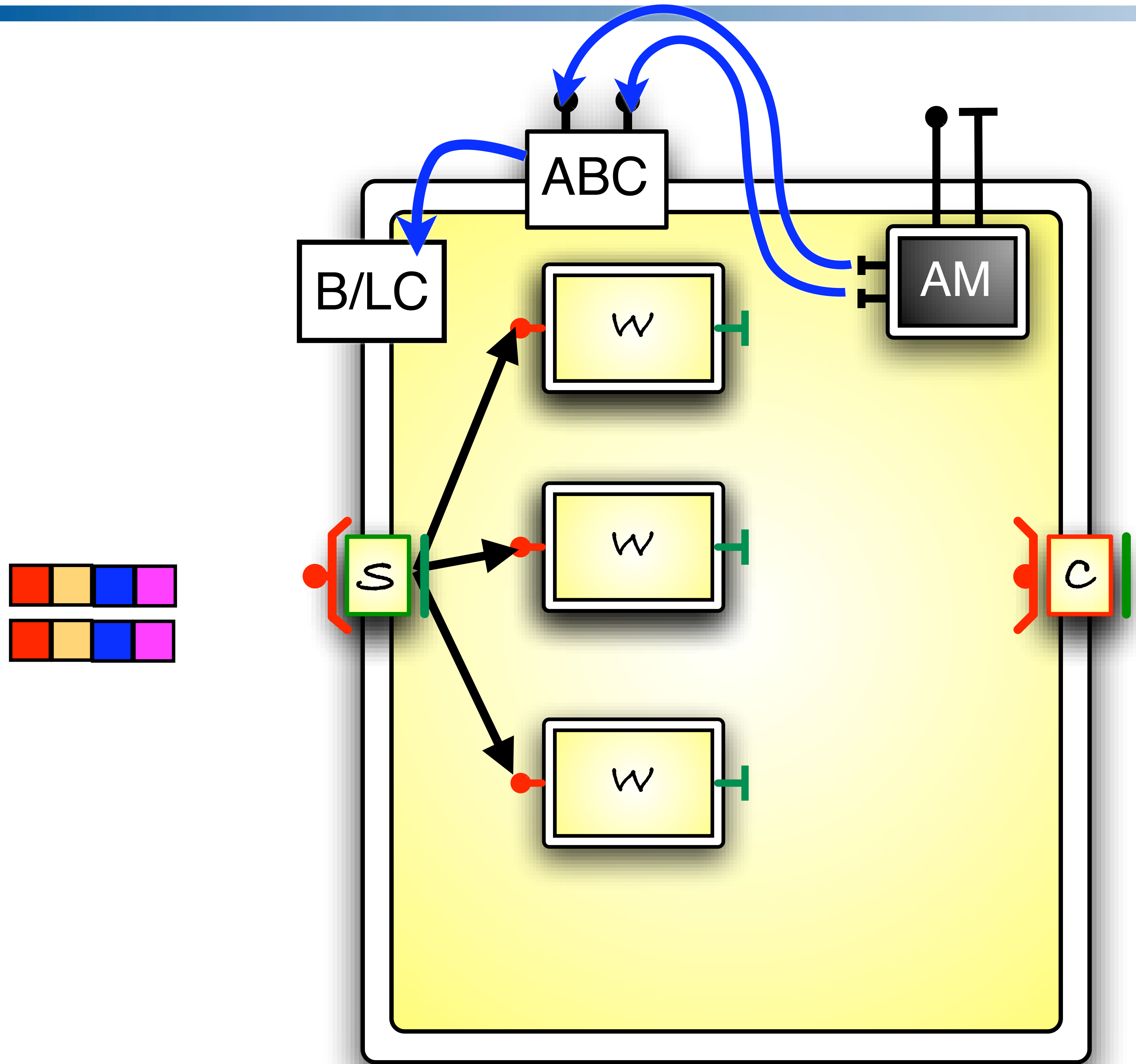
Data parallel



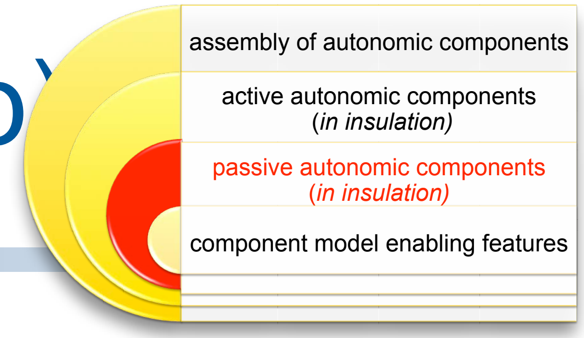
- data parallel BeSke family
  - stateless and stateful
  - composite exhibiting any number of multicast-gather ports
  - they are bound to inner components according to a fixed predefined parametric pattern
    - e.g. server: *scatter* client: *gather*
    - e.g. server: *scatter* and *broadcast* client: *none*
  - autonomic policy for performance assessed
    - for performance at least (in insulation, not yet implemented)
  - supports use case (e.g. IBM)
  - stateful DP provides data redistribution according to user defined&implemented set/get redistribution ports
- see Sonia’s talk & demo

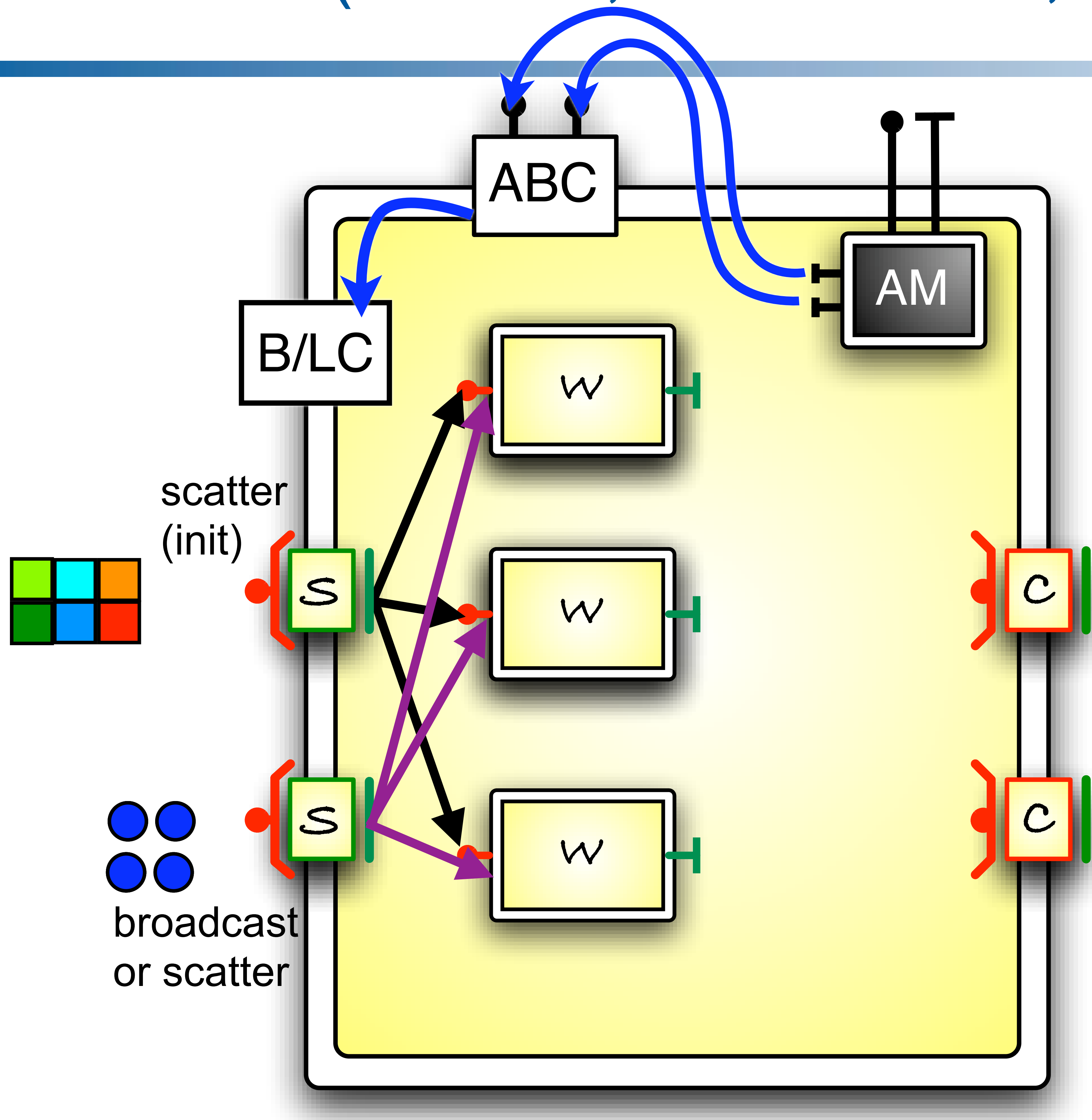
Data Parallel (stateless, pure map)





Data Parallel (stateful, distrib state, map)

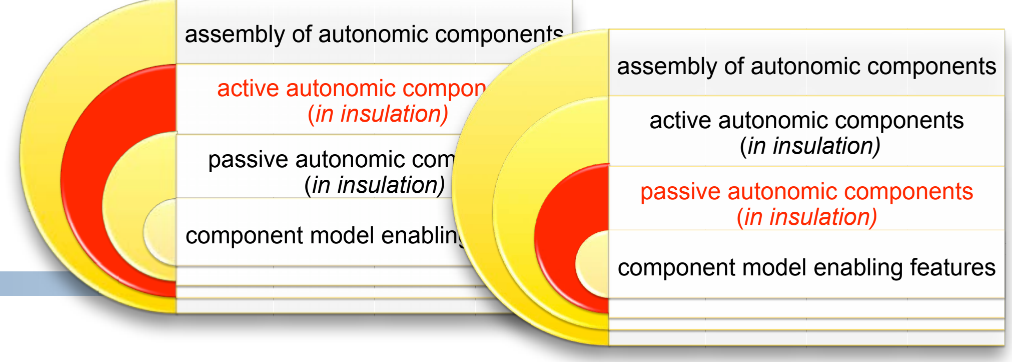


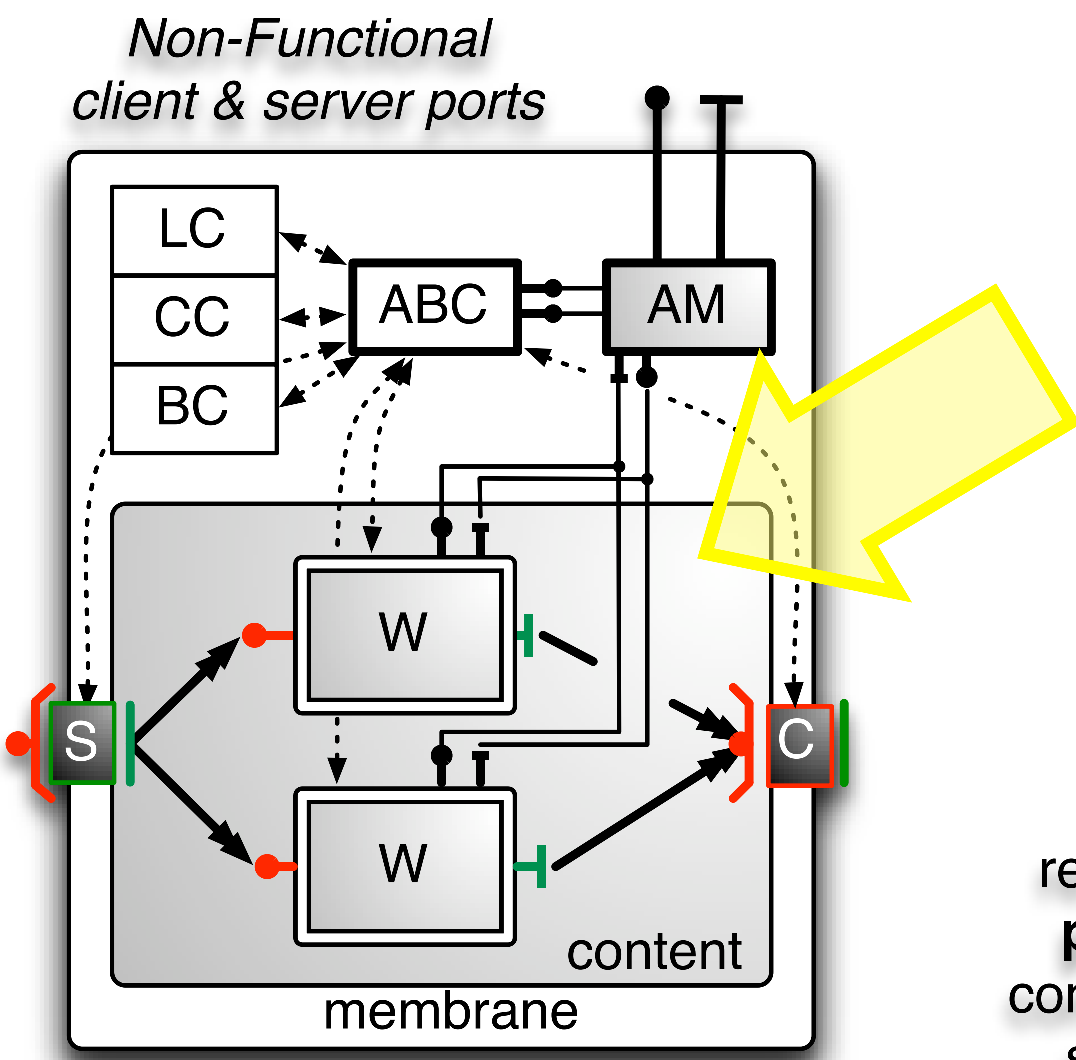


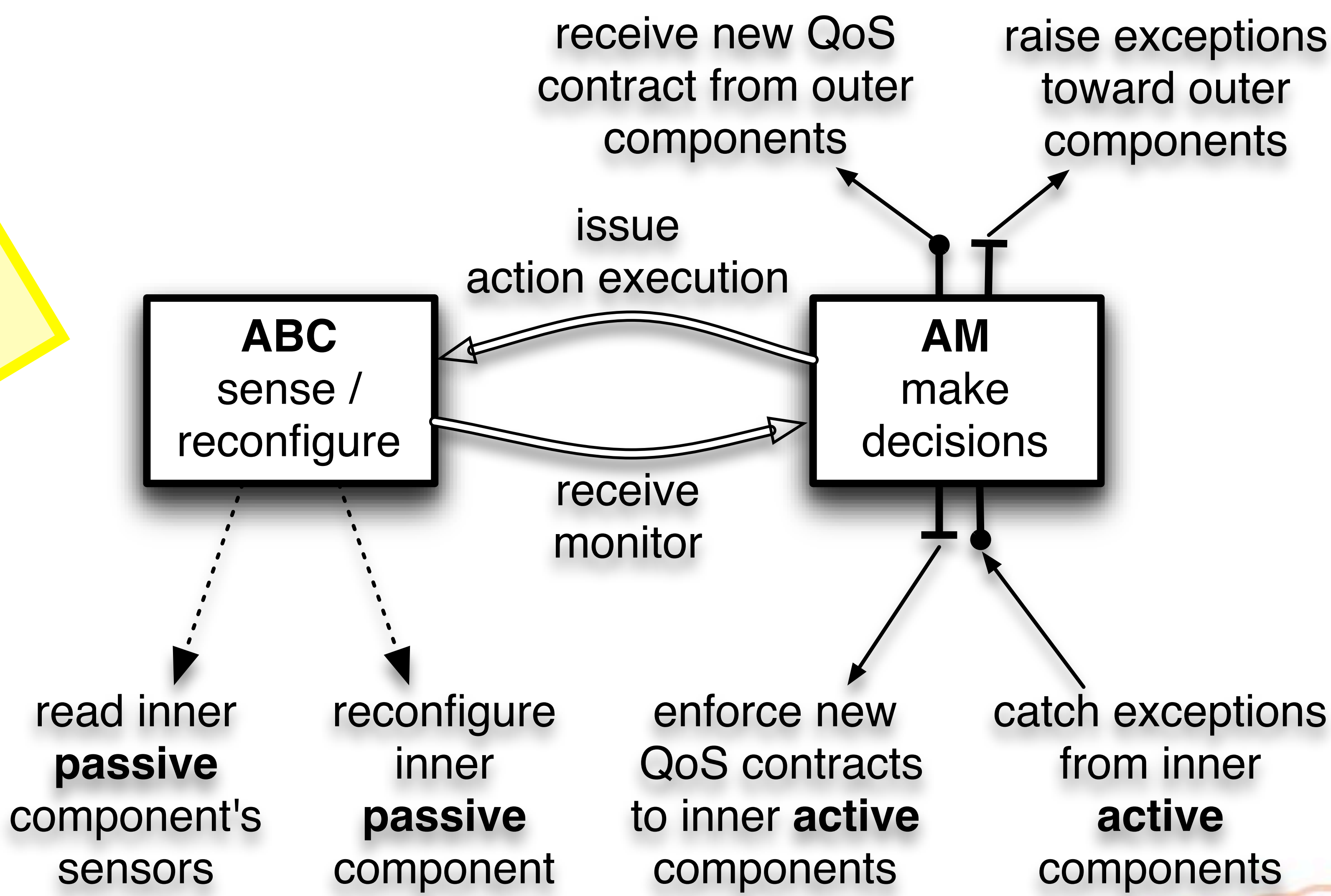
Notes

- any number of server and client ports (either RPC or stream, in theory)
- the model cannot (structurally) enforce init happens before requests on other ports
- port reconfiguration and data redistribution should be atomic (no tasks should be distributed in the middle. We are not sure we can enforce with proactive
- data redistributions are functional requests (they should be because they are related to business logic) but the inner components execute them by way of a NF port in such a way they can be executed in stopped state (workaround)
- enqueued task cannot be cancelled (IBM use case)

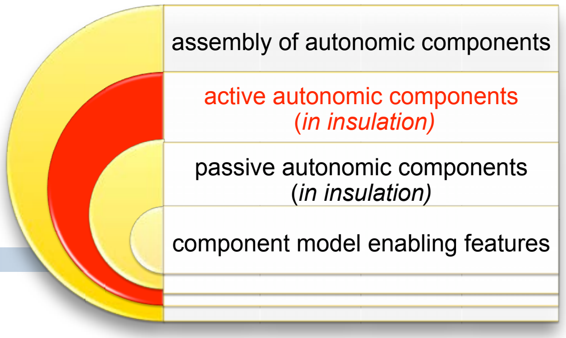
A more defined picture





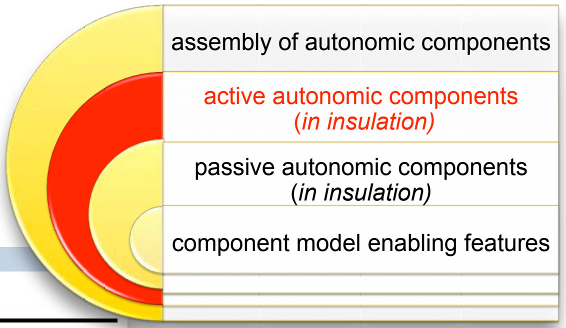


Active Autonomic Components



- Active = passive + manager
  - non-functional internal and external client and server ports
    - supporting event based orchestration
  - the manager defines a local control policy + co-ordination with other managers
    - **when**-*event*-**if**-*cond*-**then**-*actions*
      - where actions can be invocations to the ABC or messages toward other managers
    - reaction engine can be implemented via JBoss rules
      - already experimented with behavioural skeletons in SCA
        - *M. Aldinucci, M. Danelutto, G. Zoppi, P. Kilpatrick. Advances in autonomic components & services. Currently submitted*

Active management policies - examples



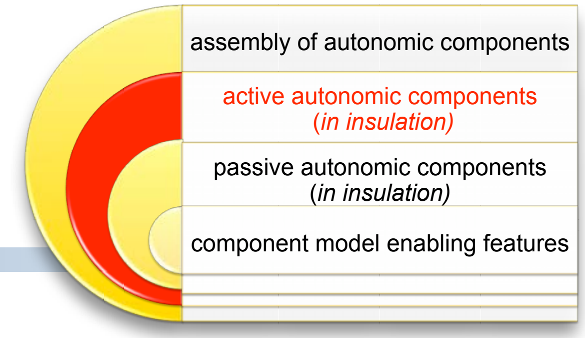
Component	Manager Contract	m_i
C_1	active (pipe) $\mathcal{CP} = K_{\mathrm{low}} \le T_{\mathrm{self}} \le K_{\mathrm{high}}$	$K_{\mathrm{low}}, K_{\mathrm{high}}$ constants; $T_{C_2}, T_{C_3}, T_{C_4}$ monitored $T_{\mathrm{self}} = \max\{T_{C_2}, T_{C_3}, T_{C_4}\}$ $\mathcal{CP}_{C_2} = \mathcal{CP}_{C_3} = \mathcal{CP}_{C_4} = \mathcal{CP}$
C_3	active (farm) $\mathcal{CP} = (\mathcal{CP}_{\mathrm{super}}) \wedge (IT_{\mathrm{self}} \le T_{\mathrm{self}})$	IT_{self} = request inter-arrival time; n_{self} = #workers let C_j children of C_3, $1 \le j \le n$: T_{C_j} monitored $T_{\mathrm{self}} = \sum_{j=1..n} T_{C_j}/n$; $\mathcal{CP}_{C_j}$ = optimise(T_{C_j}); $\mathcal{G} = 1/n_{\mathrm{self}}$
C_5	active (pipe) $\mathcal{CP} = \mathcal{CP}_{\mathrm{super}}$	$T_{C_6}, T_{C_7}, T_{C_8}$ monitored $T_{\mathrm{self}} = \max\{T_{C_6}, T_{C_7}, T_{C_8}\}$ $\mathcal{CP}_6 = \mathcal{CP}_7 = \mathcal{CP}_8$ = null
$C_{2,4,6,7,8}$	passive (none)	provide $T_{C_{2,4,6,7,8}}$ via NF port (respectively)

Component	Manager Contract	m_i
pipe	$\mathcal{CP} = K_{\mathrm{low}} \le T_{\mathrm{self}} \le K_{\mathrm{high}}$	$K_{\mathrm{low}}, K_{\mathrm{high}}$ constants; $\forall C_f \in$ children(self): T_{C_f} monitored (service time); $T_{\mathrm{self}} = \max_{\forall C_f}\{T_{C_f}\}$; $\forall C_f : \mathcal{CP}_{C_f} = \mathcal{CP}$
farm	$\mathcal{CP} = (\mathcal{CP}_{\mathrm{super}}) \wedge (IT_{\mathrm{self}} \le T_{\mathrm{self}})$	IT_{self} monitored (request inter-arrival time); $\forall C_f \in$ children(self): T_{C_f} monitored (service time); n_{self} = #workers; $T_{\mathrm{self}} = \sum_{j=1..n} T_{C_j}/n$; $\mathcal{CP}_{C_f}$ = optimise(T_{C_f}); $\mathcal{G} = 1/n_{\mathrm{self}}$

	Plan	Expected Cost	Expected Benefit
$\mathrm{PL}_{\mathrm{F}_1}$	move the slower worker C_w on a faster platform, if any	h = cost(stop(C_w);deploy(C_w);start(C_w))	decrease service time. $T_F(\circlearrowleft_{t+h}) = T_{C_x}(\circlearrowleft_t)\Delta$, $0 \le \Delta \le 1$ speed difference between the platforms
$\mathrm{PL}_{\mathrm{F}_2}$	increase parallelism degree (allocate k new workers)	h = cost(deploy(C_x);start(C_w)) for k instances	decrease service time. $T(\circlearrowleft_{t+h}) = T(\circlearrowleft_t)n/(n+k)$
$\mathrm{PL}_{\mathrm{F}_3}$	decrease parallelism degree (deallocate k workers)	h = cost(stop(C_w)) for k instances	increase service time. $T(\circlearrowleft_{t+h}) = T(\circlearrowleft_t)(n+k)/n$
$\mathrm{PL}_{\mathrm{F}_4}$	raise violation		
$\mathrm{PL}_{\mathrm{P}_1}$	move stage (C_s) with maximum T to a faster resource, if any	h = cost(stop(C_s); deploy(C_s); start(C_s))	decrease service time. $T_P(\circlearrowleft_{t+h}) = T_{C_P}(\circlearrowleft_t)\Delta$, $0 \le \Delta \le 1$ speed difference between the platforms
$\mathrm{PL}_{\mathrm{P}_2}$	collapse fastest adjacent stages (C_s, C_{s+1})	h = cost(stop(C_s); deploy(C_s); start(C_s)) for C_s, C_{s+1}	decrease resource usage $n = n - 1$, increase service time
$\mathrm{PL}_{\mathrm{P}_3}$	raise violation		

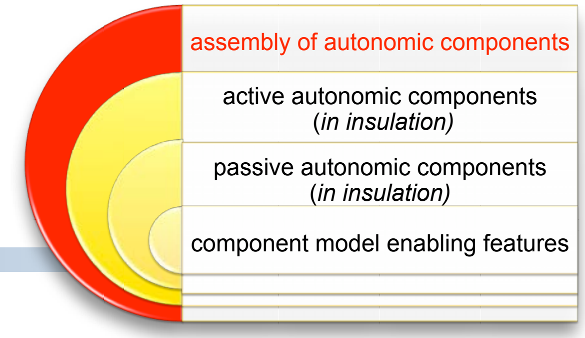
M. Aldinucci, M. Danelutto, P. Kilpatrick. Towards hierarchical management of autonomic components: a case study. CoreGRID Technical Report TR-0127, March 2008

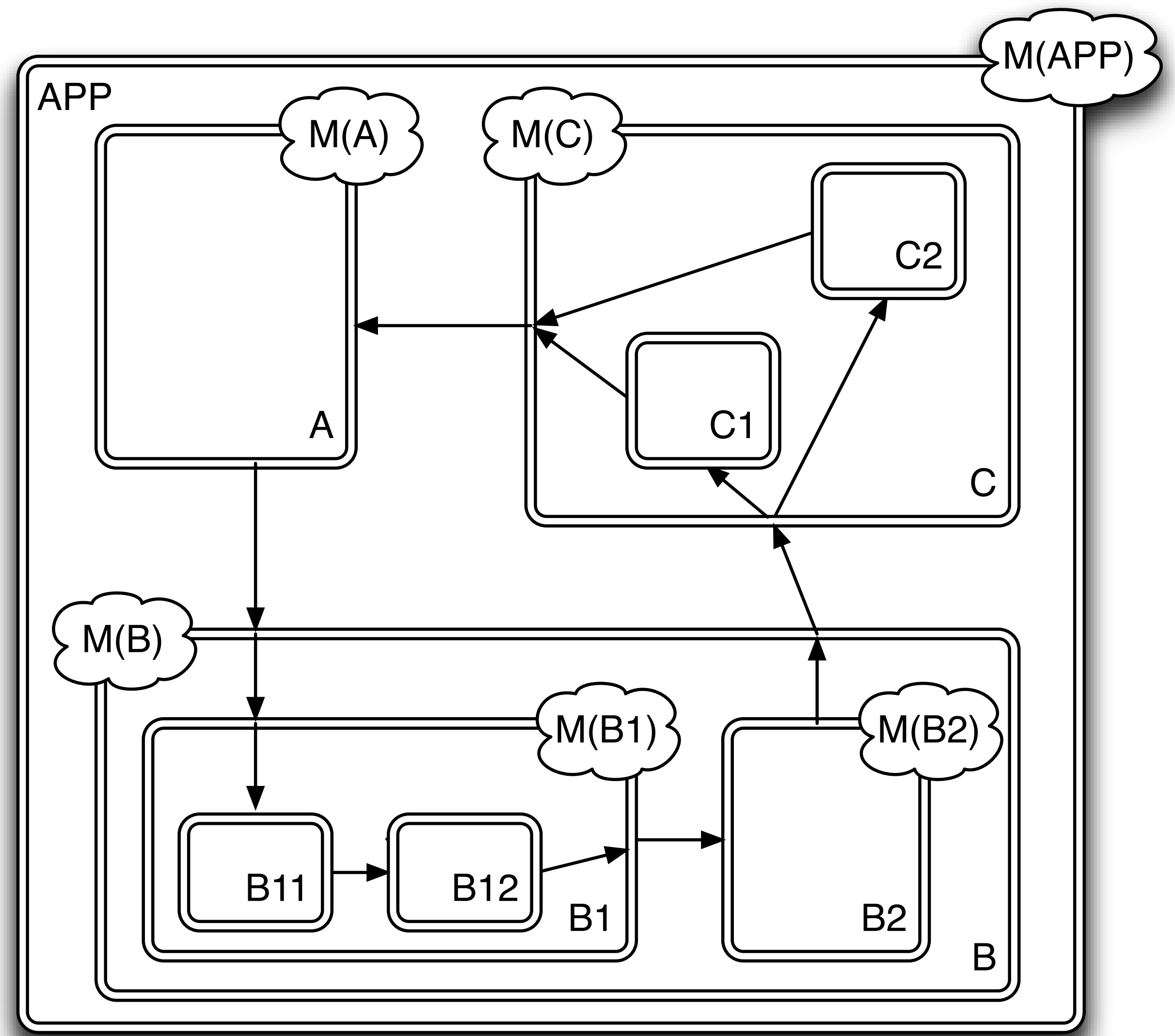
Active management - resume

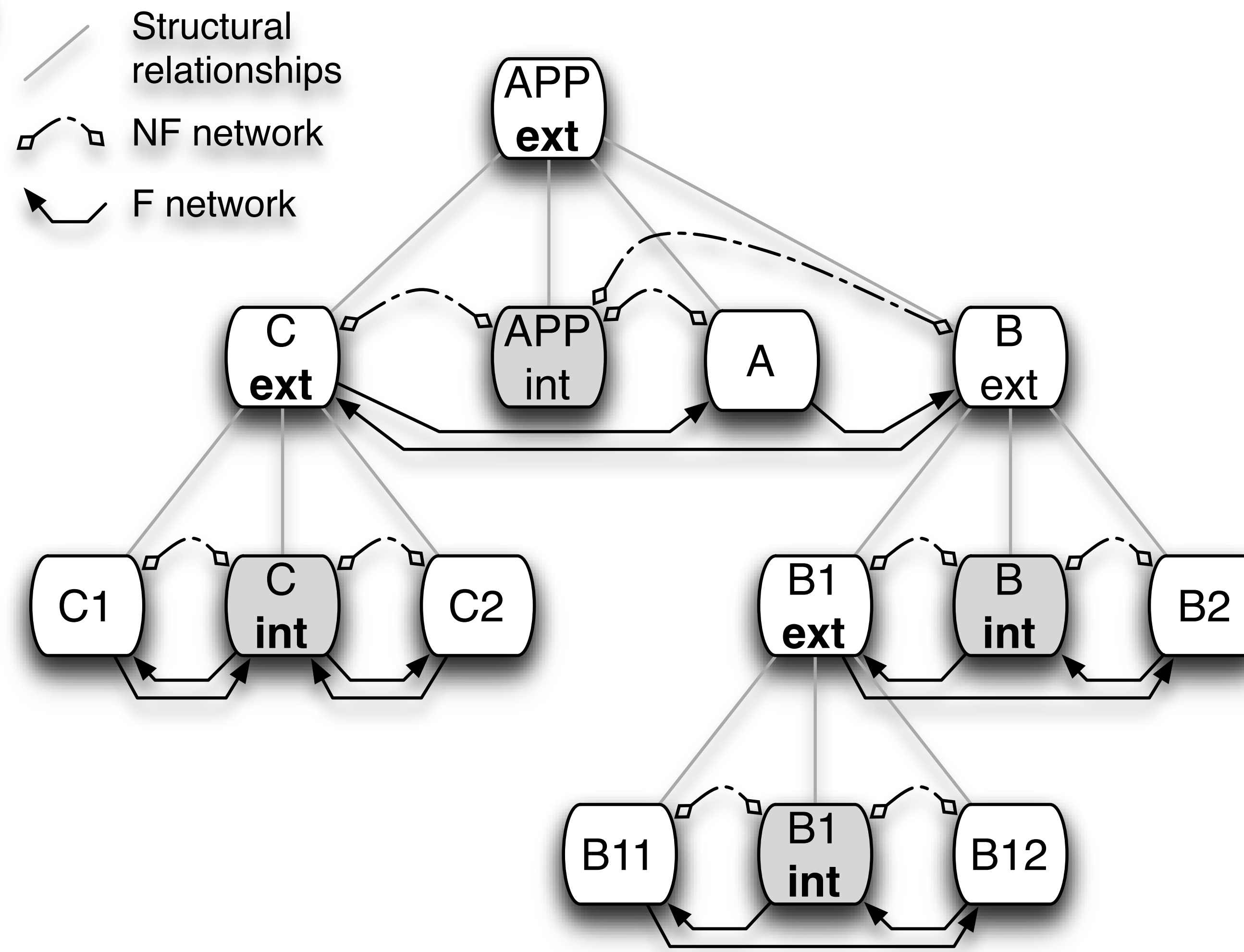


- accomplishment state: 75%
- for the next deliverable
  - equip current demo with a simple manager
    - working in insulation
    - some of those already have in reality
    - code reengineering work mostly
    - farm and data parallel supported
- for the end of the project
  - example of manager coordination
    - probably miming IBM use case
  - dynamic pluggable QoS contracts
    - defined as mobile beans

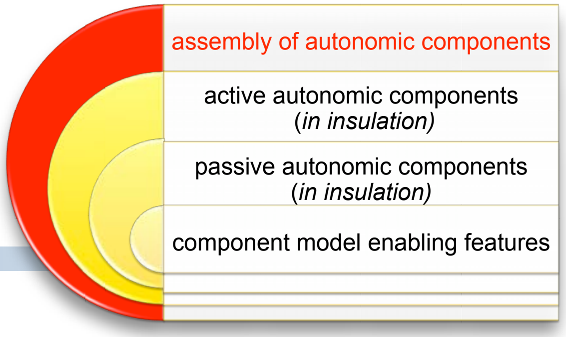
Management orchestration - concepts





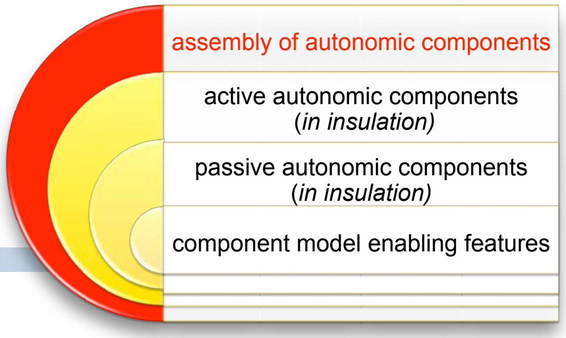


Management orchestration - references



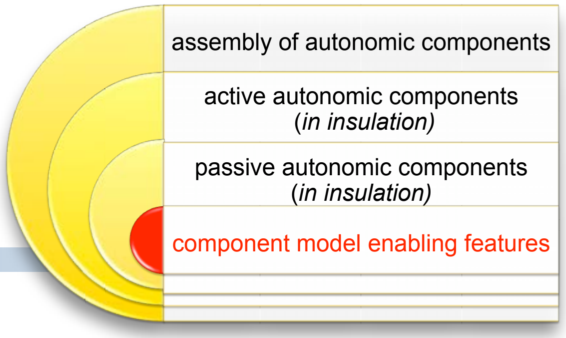
- accomplishment state: 50%
  - so high just because we don't plan to fully investigate the problem within project lifetime
  - we plan just a single experiment
    - maybe related to IBM use case
  - in general, we still have many open problems in this regard

GCM Component model thoughts

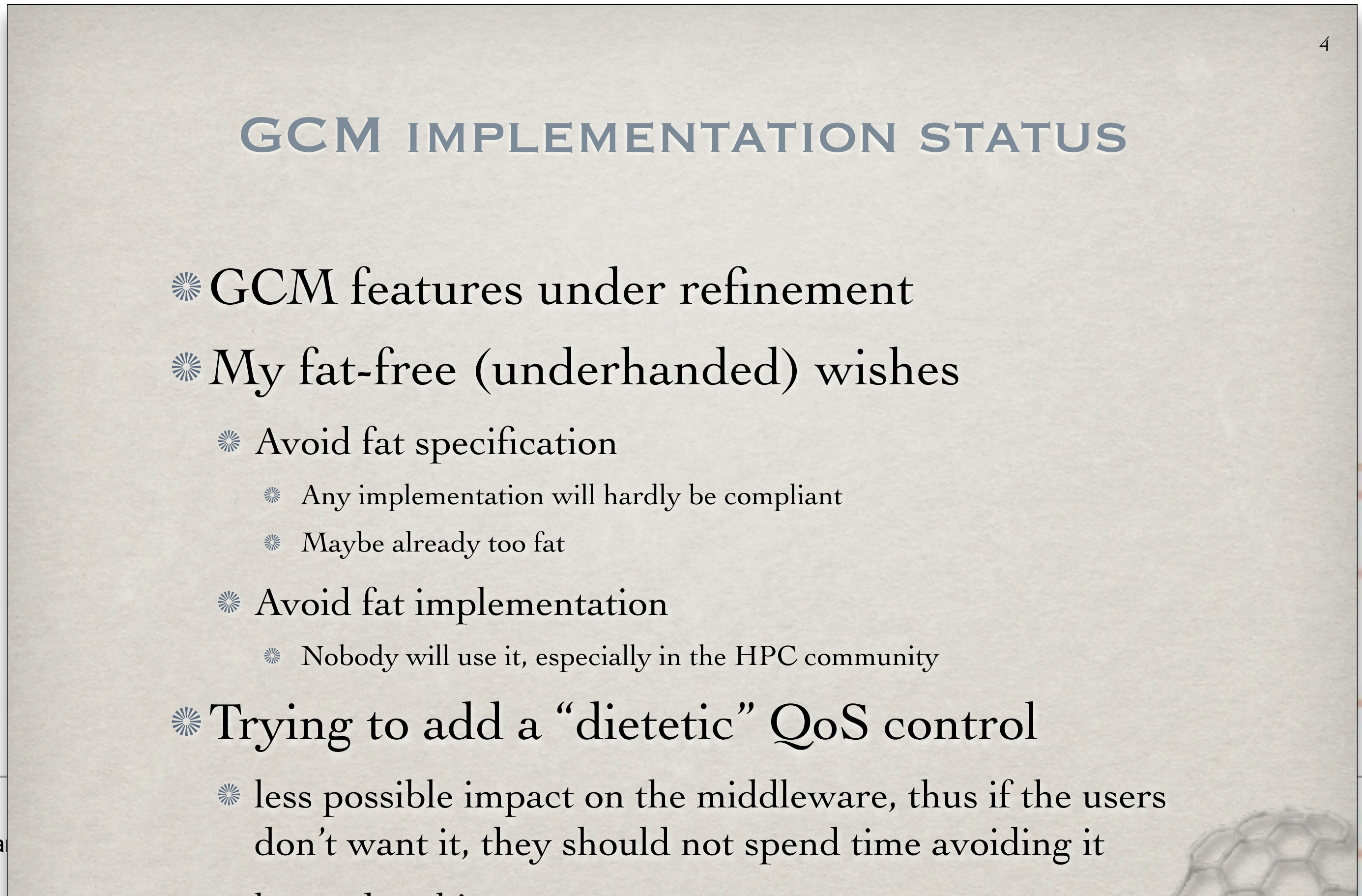


- possible technical flaws
  - non-uniformity of involved languages
    - ADL, xml, Java, Proactive (or other middleware)
  - lack of static tools
    - correctness of ADL files - really error prone
    - factories & automatic generation of non-creative code
      - how many copy-paste do you need to develop an application?
      - do components actually ease the development?
  - dynamic reservation & deployment support
    - cluster reservation mechanism should be rethought in the light of reconfiguration needs?
    - G5K
- conceptual flaws
  - what is the pragmatic of the composite?
  - is reuse really exploited?

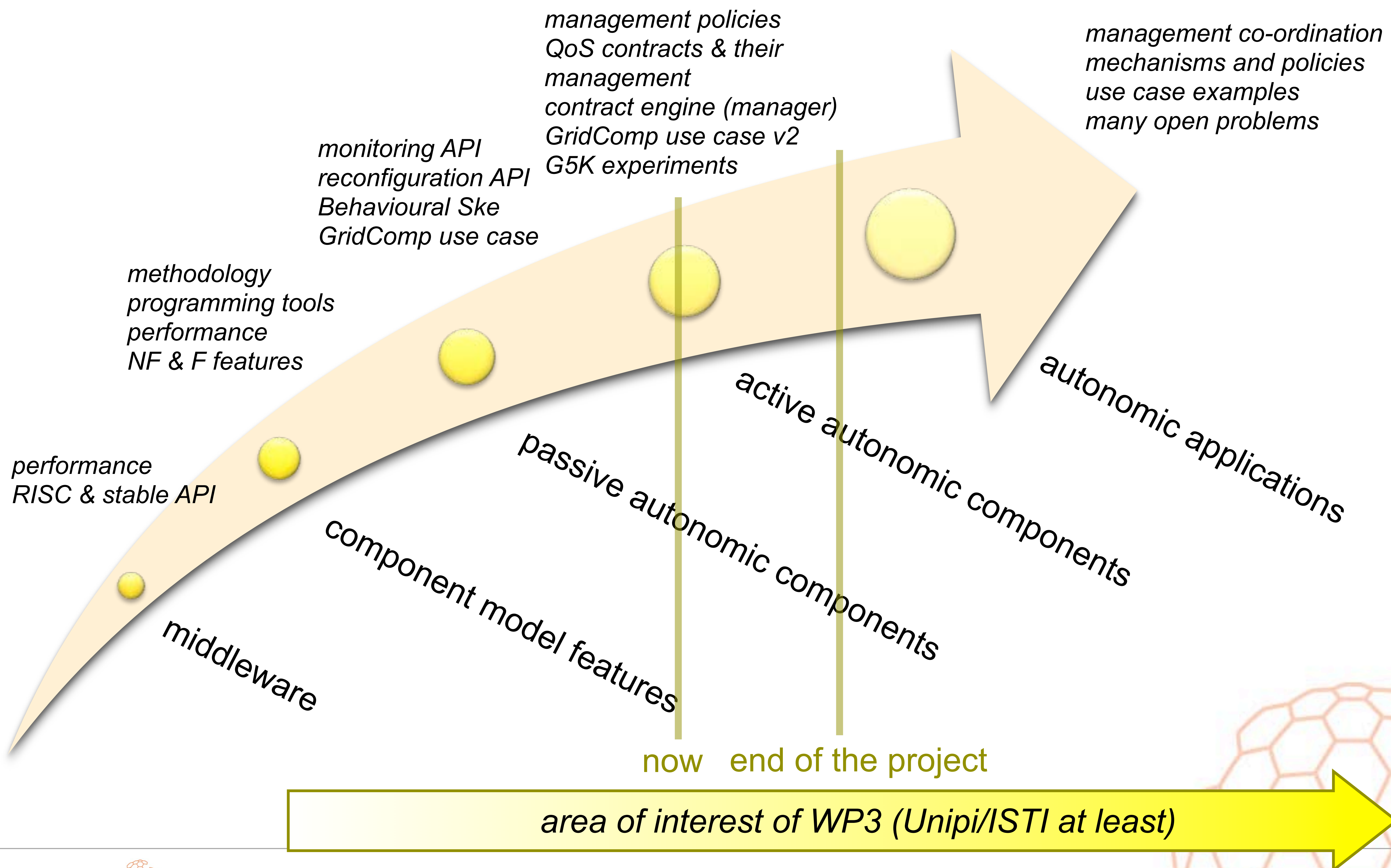
GCM/Proactive thoughts & feedbacks



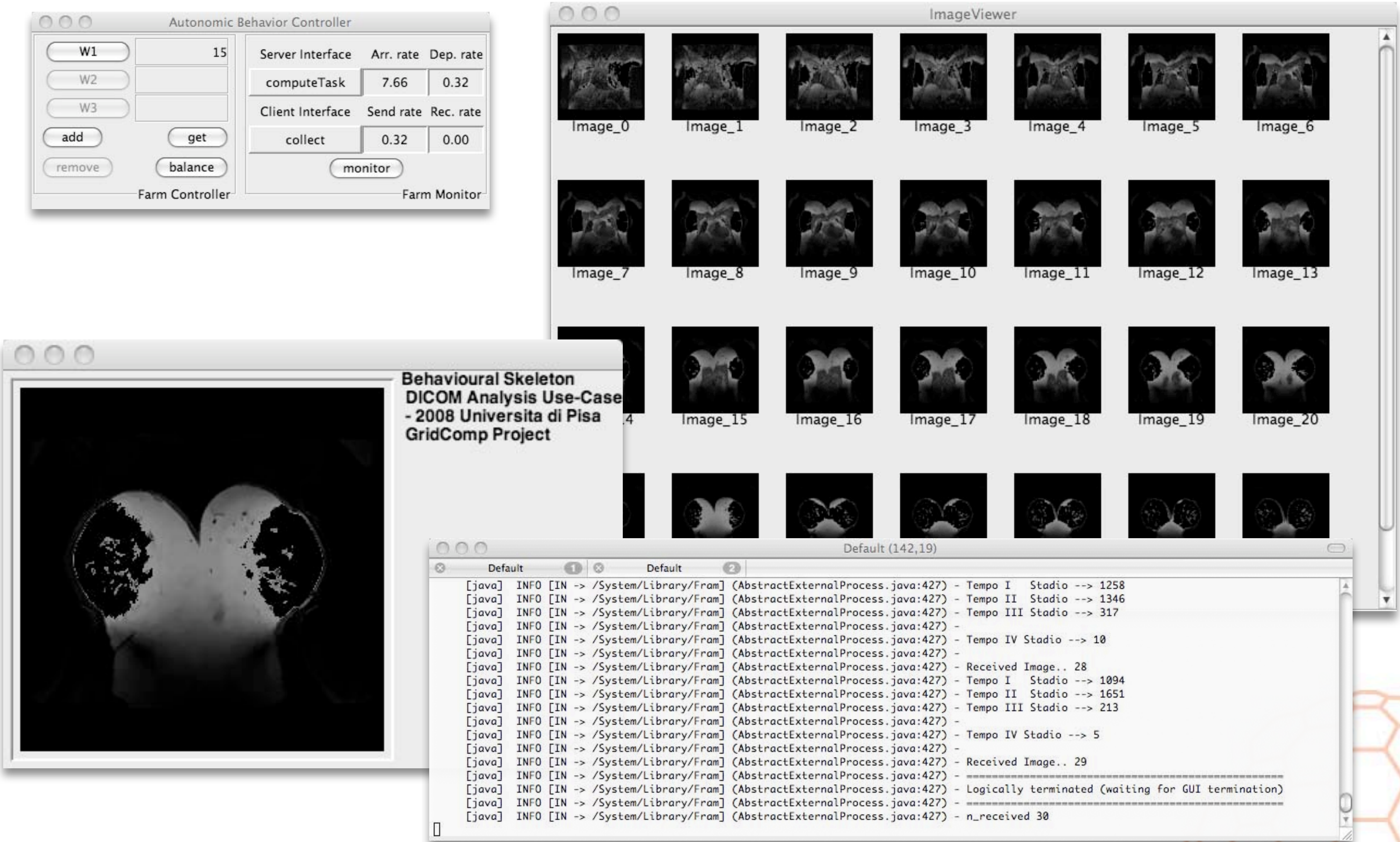
- scarce separation between component model and its run-time support (proactive)
  - the component run-time support is difficult to extend
  - a deep run-time knowledge is strictly required to develop any non-toyish application
- asynchronous communications + queues impair run-time reactivity
  - events (e.g. monitor) refers to current state, but any structural reconfiguration will be applied to non yet enqueued requests
    - e.g. task cancel IBM fingerprint
  - 90% of development effort consists in avoiding unwanted behaviour related to active object model
- life-cycle: stopped/running states probably not enough
  - proper destruction mechanism, states related to reconfiguration (reconf-safe),
- monitor still missing
  - no real possibility to experiment autonomic policies without it
- GCM has a pretty fat middleware
  - as said in Palma 2007
  - hard to know all the features (then hard to exploit them)
- Proactive versions
  - 3.9_beta
  - any porting to a new version will be complex at this point, apart for expected changes (e.g. monitor)



Resume and perspectives



Demo



Autonomic Behavior Controller
W1
15
W2
W3
add
get
remove
balance
Farm Controller
Server Interface
Arr. rate
Dep. rate
computeTask
7.66
0.32
Client Interface
Send rate
Rec. rate
collect
0.32
0.00
monitor
Farm Monitor
ImageViewer
Image_0
Image_1
Image_2
Image_3
Image_4
Image_5
Image_6
Image_7
Image_8
Image_9
Image_10
Image_11
Image_12
Image_13
Image_15
Image_16
Image_17
Image_18
Image_19
Image_20
Behavioural Skeleton
DICOM Analysis Use-Case
- 2008 Universita di Pisa
GridComp Project
Default (142,19)
Default
Default
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) - Tempo I Stadio --> 1258
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) - Tempo II Stadio --> 1346
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) - Tempo III Stadio --> 317
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) -
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) - Tempo IV Stadio --> 10
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) -
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) - Received Image.. 28
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) - Tempo I Stadio --> 1094
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) - Tempo II Stadio --> 1651
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) - Tempo III Stadio --> 213
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) -
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) - Tempo IV Stadio --> 5
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) -
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) - Received Image.. 29
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) - ==
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) - Logically terminated (waiting for GUI termination)
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) - ==
[java] INFO [IN -> /System/Library/Fram] (AbstractExternalProcess.java:427) - n_received 30

Resume and perspectives

- passive autonomic components
  - almost all planned features have been implemented
    - LC, ABC, MC, and patches to several controllers
    - behavioural skeletons (functional replication stateless and stateful)
- active autonomic components
  - implementation of management policies for performance (farm and DP)
  - QoS contracts as JBoss rules (static and dynamic)
    - not originally planned, already experimented in SCA
- orchestration of management
  - working on the design
  - formalisation of reconfiguration and QoS contracts
  - formalisation of orchestration strategies
    - uniformly supporting important concepts for HPC and pervasive
      - locality, fault-tolerance, geographic position, ...
- all use cases are supported by BeSke