Towards a Formal Semantics for Autonomic Components

Marco Aldinucci
University of Pisa, Italy

Emilio Tuosto
Univ. of Leicester, U.K.

CoreGRID Symposium
August 25-26, 2008, Las Palmas de Gran Canaria, Canary Island, Spain

Outline

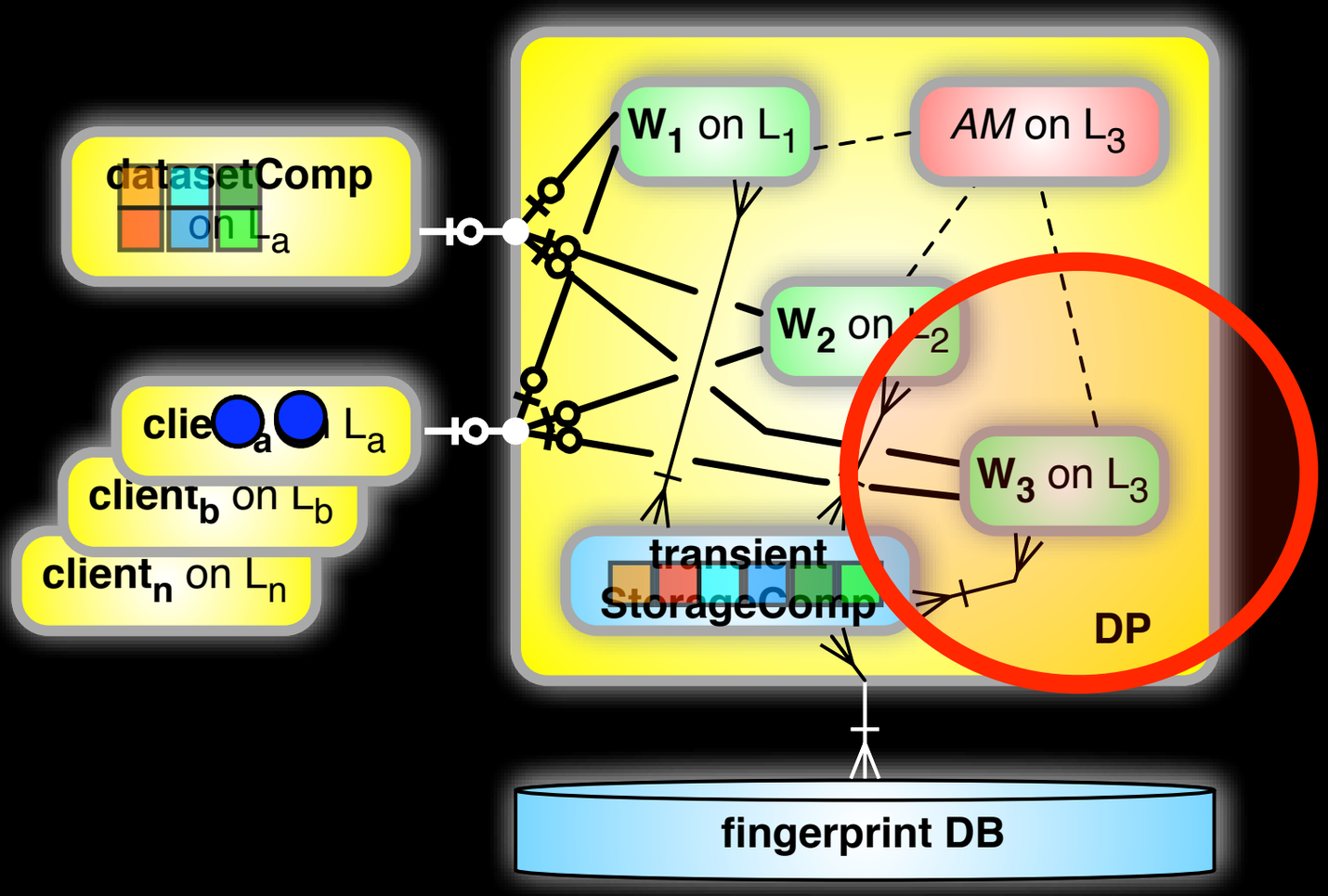
- A motivating demo
  - brand new, not in the paper
- A SHR-based semantics for GCM app adaptation
  - just the graphical representation, here
- Management
  - JBoss-based first-order logic contracts
  - hierarchical management
- Evolutions
  - already ongoing ...

Terminology - GCM

- GCM - Grid Component Model
  - Defined in CoreGRID developed in GridCOMP
- Features
  - hierarchical components (Fractal-derived)
  - collective communications and component interaction patterns
  - autonomic management of notable (parallel) composite components
  - advanced programming models
    - e.g. behavioural skeletons
  - recently **ETSI standardised**
    - deployment & others

A motivating example

IBM fingerprint recognition app (mockup, GridComp)



7) **AM** *reacts (e.g. increasing // degree): copying* **W***1; bindings (external,* **AM**, **StorageComp***) should be preserved; DB partitions (***W***x state) should be redistributed via StorageComp*

6) **AM** *may sense a changed answer time (e.g. increased), due to a dataset size/kind and/or platform status change*

5) *repeat 2-3-4 ... 2-3-4 ...*

4) **clients** *get the answer OR(***W***1,***W***2,...)*

3) *each worker matches the fingerprint against its DB partition*

2) *clients broadcast requests to all workers*

1) *references to DB slices are scattered*

RPC or dataflow bindings ---- management bindings data sharing port bindings

demo

Global vs local view

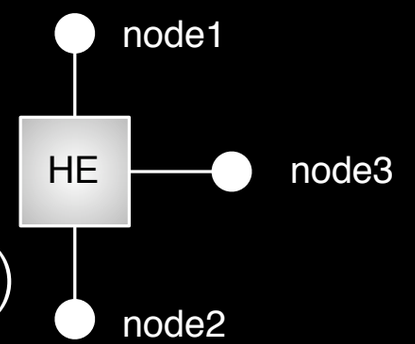
- This is a global view of the application
  - somehow adaptation are seen by way of the sky-cam components are created, bindings are drawn ... but who do it ?
- This global behaviour should be achieved as the coordination of local operations
  - local operations (independently specified)
  - user-defined (local) policies
    - determining a global behaviour (e.g. which adaptation should be preferred, among all possible)

Semantics rationale

- Formal specification is important for autonomic app
  - ADL in not enough; it describes a **static** view
  - adaptation operation should be **automatically** managed
  - correctness (and other) should be enforced beyond across adaptation
- We consider dynamically evolving component assemblies
  - suitable representation for adaptive and autonomic applications
  - by using SHR as specification methodology

Terminology - SHR

- SHR - Synchronised Hyperedge Replacement
  - One of the theoretical tools of (IP - 6FP)
- Features
  - hyperedges, i.e. edges connecting many nodes
    - from now on exchange edges (boxes) with nodes (circles)
    - send/recv, unicast, multicast, broadcast, scatter, gather, ...
  - rewrite by way of synchronised ***context-free*** productions
  - productions impose conditions on adjacent nodes
  - components' behaviour ***independently*** specified by productions
  - global transitions as application of ***compatible*** productions
  - i.e. enables local adaptations



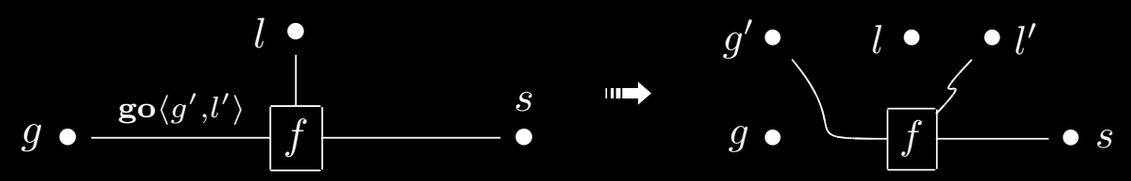
Some adaptation operations

Migration	`go`	keep the external state (if any)
	`start`	start from a fresh external state
Replication	`share`	component replica share external state with source component
	`copy`	component replica is created with a fresh external state
Kill	`kill`	kill the component (detach bindings, garbage collect, ...)

Adaptation ops as SHR axioms

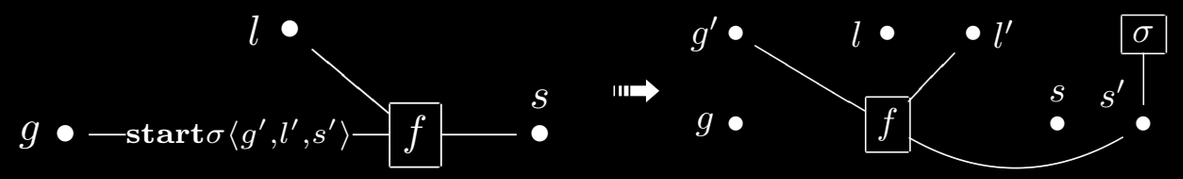
s, s' = external states; g, g' = managers; l, l' = locations

move component f from l to l' (keep state)



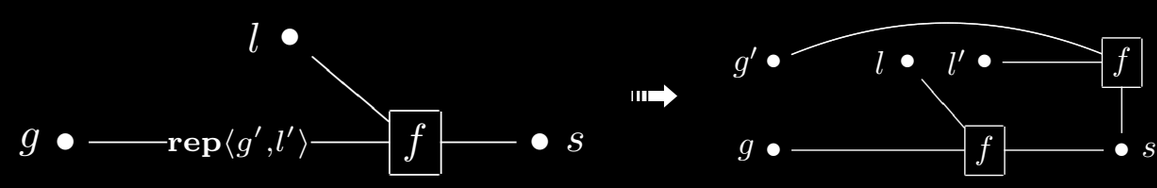
e.g. g=g'

move component f from l to l' (fresh state)



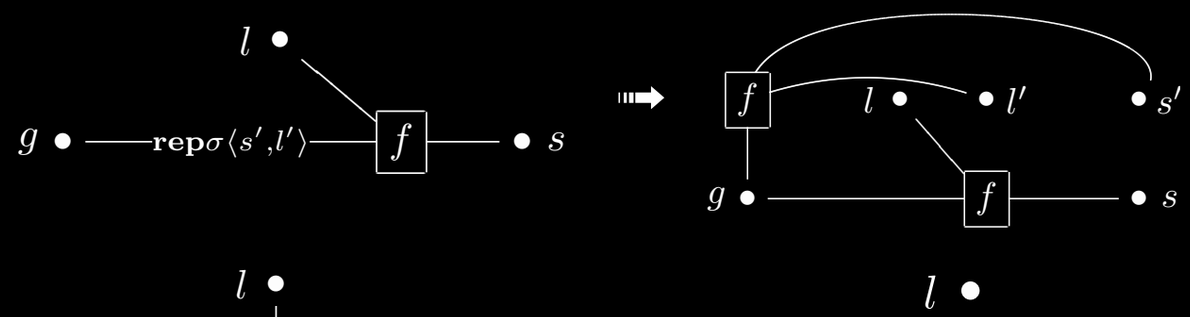
e.g. g=g', s ≠ s'

replicate component (keep state, change location)



e.g. g=g'

replicate component (fresh state, change location)



e.g. s ≠ s'

kill component

SHR Inference rules…in one slide

Parallel

$$\frac{\Gamma \vdash G_1 \xrightarrow{\Lambda} \Phi \vdash G_2 \qquad \Gamma' \vdash G_1' \xrightarrow{\Lambda'} \Phi' \vdash G_2' \qquad (\Gamma \cup \Phi) \cap (\Gamma' \cup \Phi') = \emptyset}{\Gamma, \Gamma' \vdash G_1 | G_1' \xrightarrow{\Lambda \cup \Lambda'} \Phi, \Phi' \vdash G_2 | G_2'}$$

The system can do whatever disjoint subsystems do

Restrict

$$\frac{\Gamma, x \vdash G_1 \xrightarrow{\Lambda} \Gamma, x \vdash G_2 \qquad \Lambda(x) = \epsilon \vee \Lambda(x) = \tau}{\Gamma \vdash \nu x\ G_1 \xrightarrow{\Lambda \setminus \{x\}} \Gamma \vdash \nu x\ G_2}$$

The system can do any transition not requiring any synchronisations on restricted node

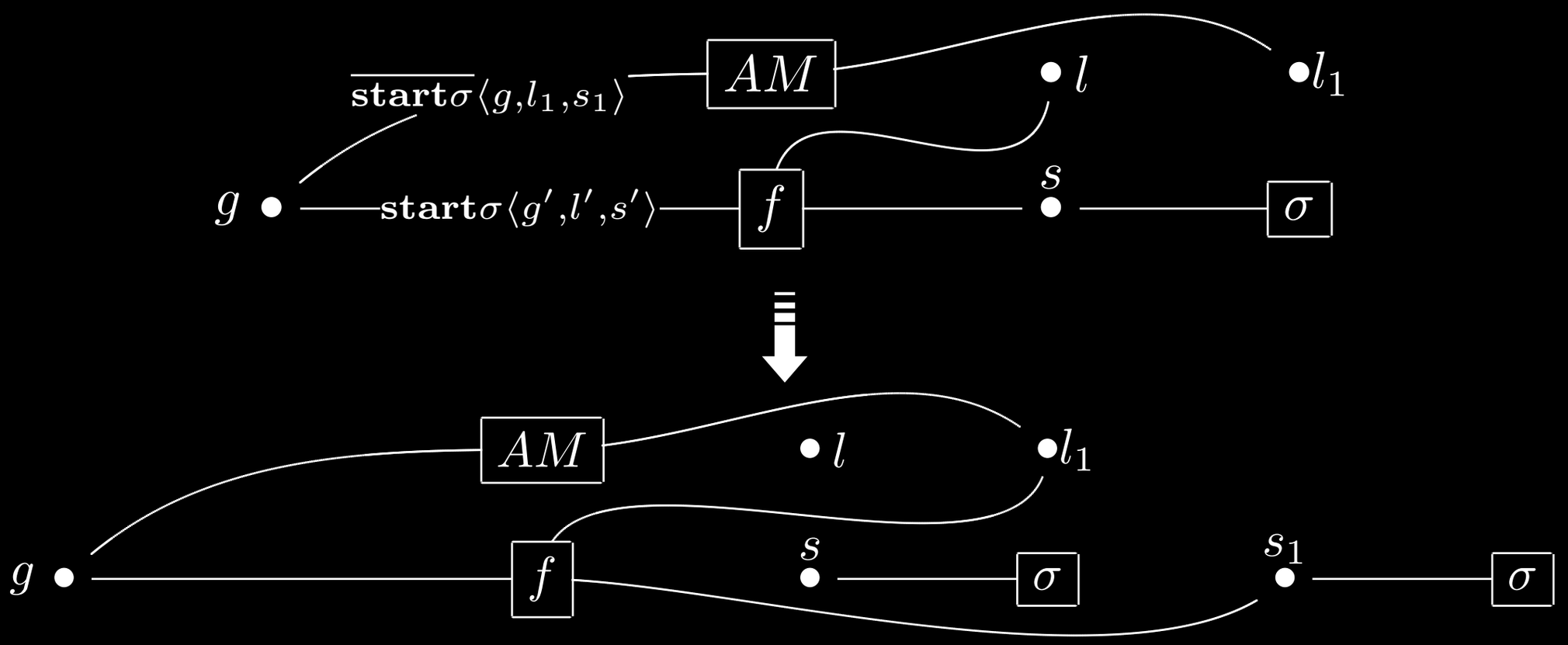
Merge

$$\frac{\Gamma, x, y \vdash G_1 \xrightarrow{\Lambda} \Phi \vdash G_2}{\Gamma[x/y] \vdash G_1[x/y] \xrightarrow{\Lambda, \{x, \tau,\}} \Phi[x/y] \vdash \nu U\ G_2[x/y]\rho}$$

x and y can be fused provided that they perform *compatible* synchronisation actions

CoreGRID GridCOMP SENSORIA

Example



AM asks component f to change location and attach to a new external state (application of start rule)

Observe that hyperedges can be used to represent very different concepts/attributes (e.g. **location**, **store**, **manager** hooks)

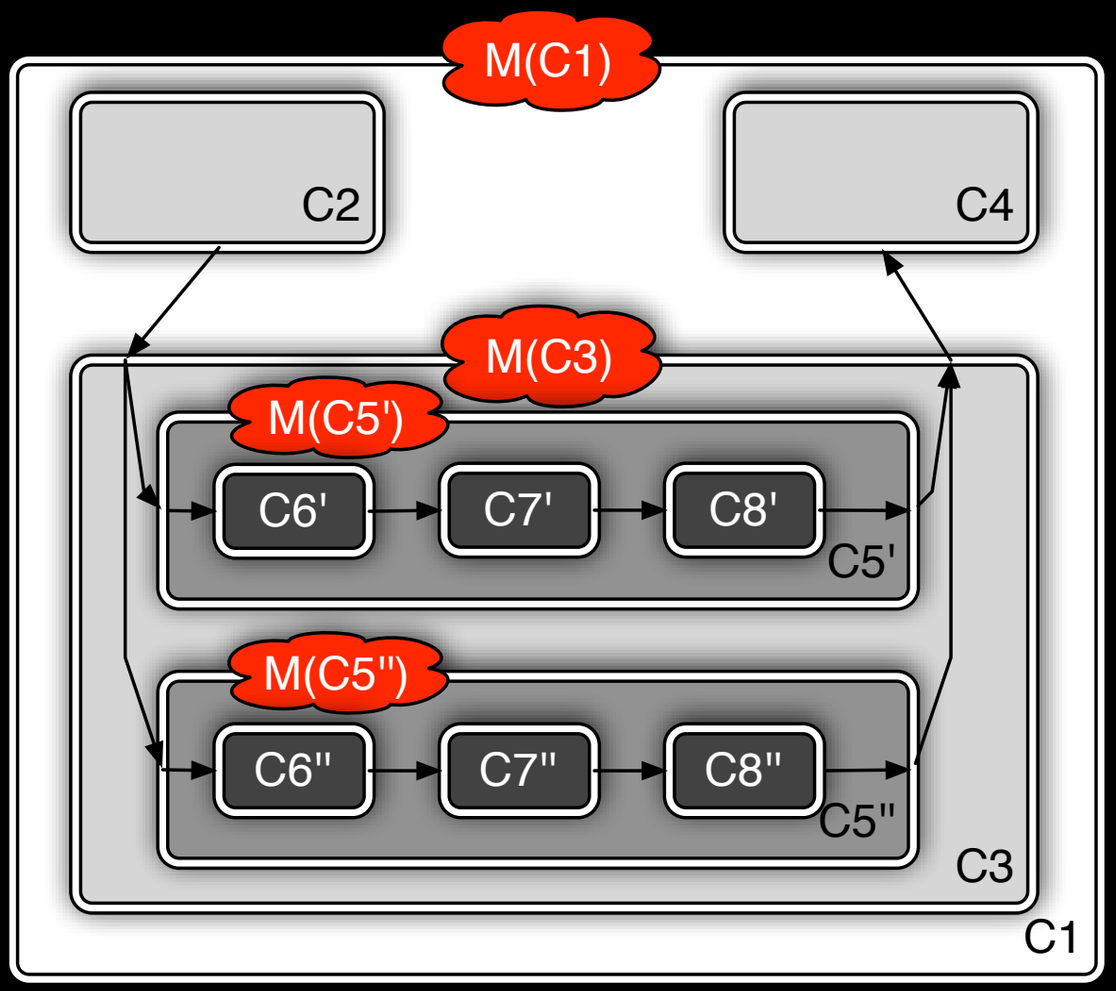
Driving adaptations

- Managers drive the adaptation process
  - choose among all possible adaptations
  - in a distributed way
- Implementing concepts in GCM
  - when-event-if-cond-then-act list of rules
  - where act either an adaptation or a message to a set of companion managers
  - as JBoss Drools
    - first order logic
      - *maybe not fuzzy enough*

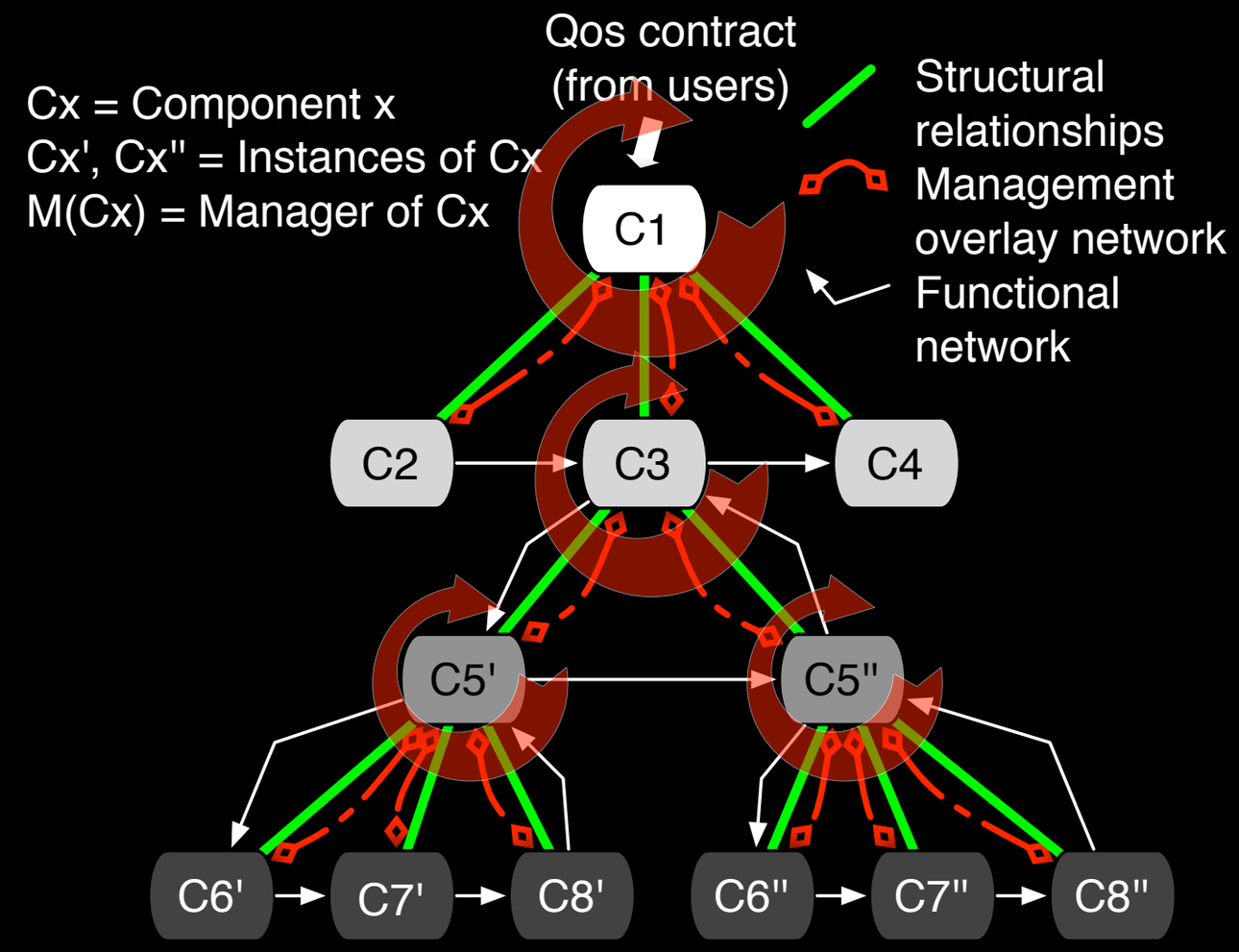
A simple contract

```
rule "CheckInterArrivalRate"
  salience 5
  when
     $arrivalBean : ArrivalRateBean( value <  ManagersConstants.LOW_PERF_LEVEL)
  then
     $arrivalBean.setData(ManagersConstants.notEnoughTasks_VIOL);
     $arrivalBean.fireOperation(ManagerOperation.RAISE_VIOLATION);
     System.out.println( "InterArrivalTime not enough - Raising a violation");
end
rule "CheckRateLow"
  when
     $departureBean : DepartureRateBean( value < ManagersConstants.LOW_PERF_LEVEL )
     $parDegree: NumWorkerBean(value <= ManagersConstants.MAX_NUM_WORKERS)
  then
 $departureBean.fireOperation(ManagerOperation.    REPLICATE_SHARE);
     $departureBean.fireOperation(ManagerOperation.BALANCE_LOAD);
     System.out.println( "Adding "+ManagersConstants.ADD_WORKERS+ "workers");
end
rule "CheckRateHigh"
  when
     $departureBean : DepartureRateBean( value > ManagersConstants.HIGH_PERF_LEVEL )
     $parDegree: NumWorkerBean(value > ManagersConstants.MIN_NUM_WORKERS)
  then
     $departureBean.fireOperation(ManagerOperation.KILL);
     $departureBean.fireOperation(ManagerOperation.BALANCE_LOAD);
     System.out.println( "Rate "+$departureBean.getValue()+" (Removing 1 workers)");
end
```

Orchestration of managers (overlay)



Cx = Component x
Cx', Cx" = Instances of Cx
M(Cx) = Manager of Cx



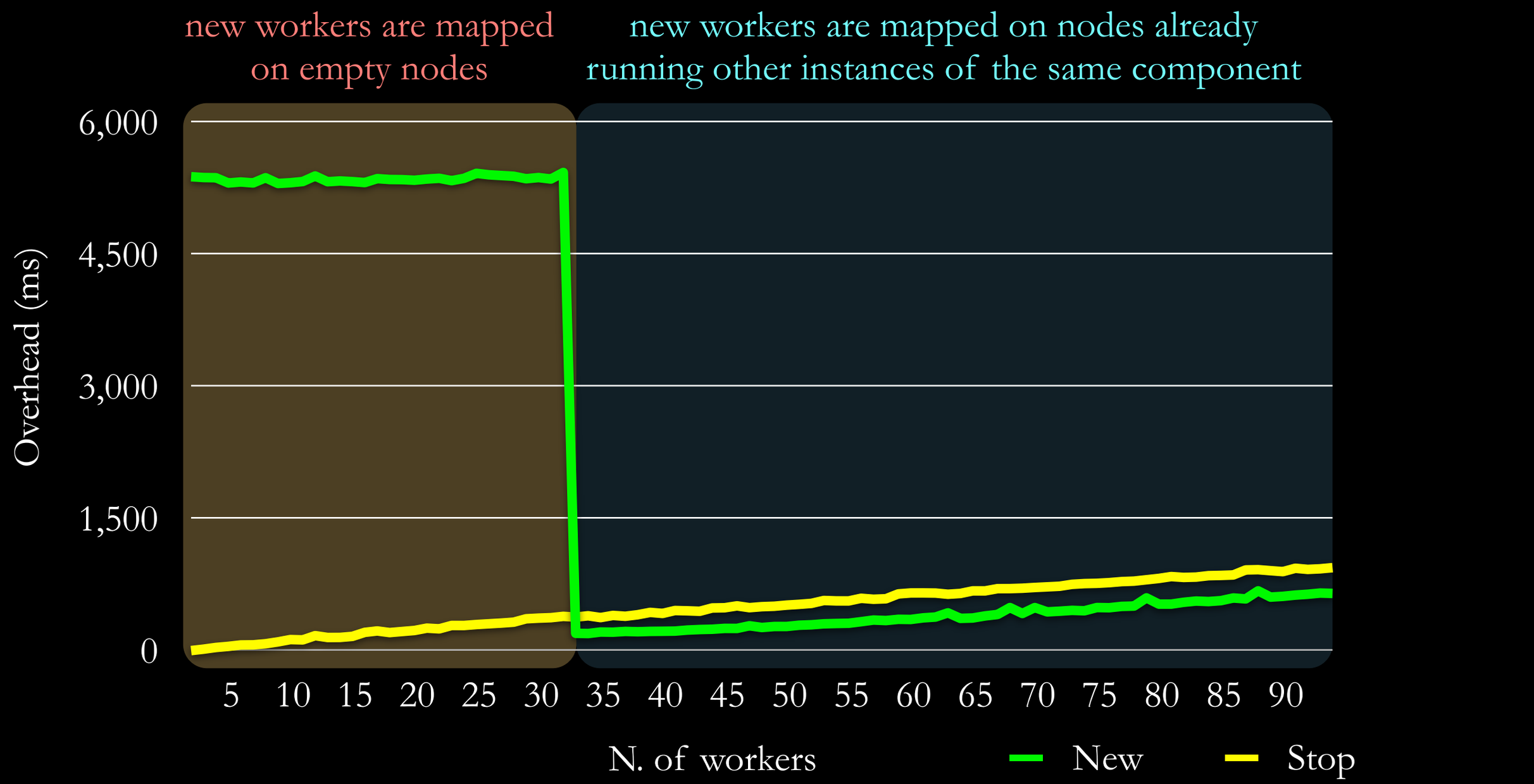
Conclusion

- We introduced few component adaptation operations
  - existing in GCM
  - able to capture typical adaptation patterns of many grid apps
- We detail a semantics for these operations
  - based on SHR
  - suitable for the description of component concurrent semantics and the run-time evolution of assemblies
- We discuss the appropriateness of the level of abstraction chosen to describe adaptation operations
  - to support the design of component-based applications and their *autonomic management*

Future works

- Present work represent just a starting point
  - establishing a common background with other communities
  - understanding the feasibility of the approach
- Currently working at exploiting the formalisation for
  - proving interesting properties ...
  - setting up a framework to derive optimised adaptation protocols
    - for example let us consider **adaptation overhead ...**

Analysis: Overheads (GCM/Proactive)



Analysis: Overhead (Alternative Impl)

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.

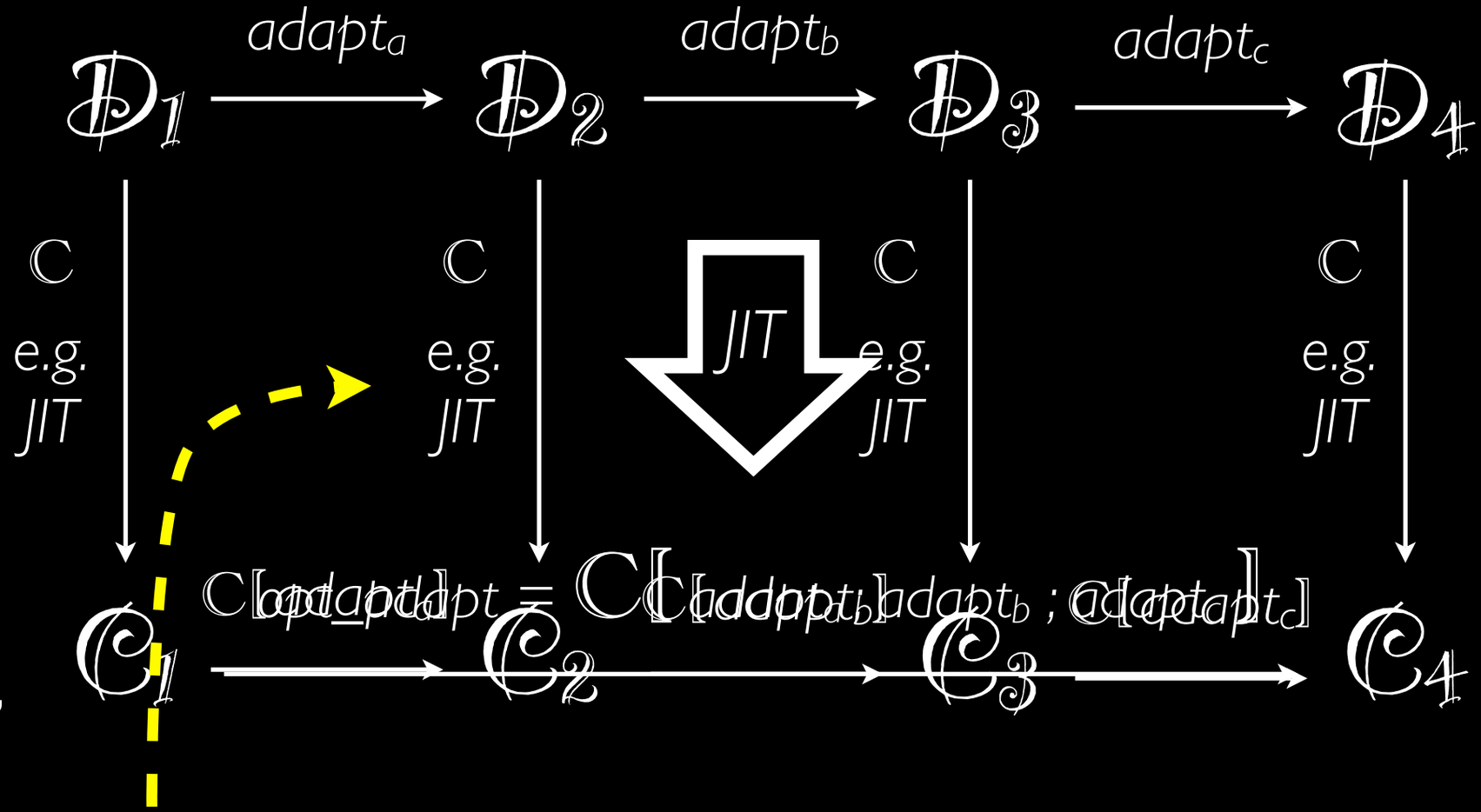
parmod kind	Data-parallel (with shared state)						Farm (without shared state)					
reconf. kind	add PEs			remove PEs			add PEs			remove PEs		
# of PEs involved	1→2	2→4	4→8	2→1	4→2	8→4	1→2	2→4	4→8	2→1	4→2	8→4
R_l on-barrier	1.2	1.6	2.3	0.8	1.4	3.7	–	–	–	–	–	–
R_l on-stream-item	4.7	12.0	33.9	3.9	6.5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

It is just C++ against Java?

- No, unfortunately it is not so simple ...
  - dynamic class loading (red vs blue zone of the previous chart), dynamic introspection, dynamic binding
  - generic data serialisation, shared data alignment
  - JIT, code factories, etc.
  - **non optimised protocols**
    - look-ahead resource recruiting, pre-deployment, atomic multicast (replica management), consensus (reconf-safe-points)
    - **sequence of reconfiguration operations**
      - add + rebalance + move means:
        stop-add-**start**-**stop**-rebalance-**start**-**stop**-move-**start**
        for each involved component - 4 of them can be avoided
- I.e. we should find the right static-dynamic trade-off

Dynamically compiling adaptations

source
designer specify adaptation operation semantics at the highest possible level



binary
operations are really adaptation protocols
lifecycle, resource recruiting, creation, binding, ...)

we need it for the "env loop - outer AC loop" (see invited talk)
i.e. for dynamically introducing new contracts or adaptations

CoreGRID GridCOMP SENSORIA

Thank you !