Skeleton “Componentization”

CCP A.A. 2008-2009
M. Danelutto
Maggio 2009

Goal

- starting from a well know parallel computation pattern
- derive a suitable component schema
- analyse the implementation
  - issues
  - peculiarities
  - ...

The pattern: task farm

- Collection of input tasks to be processed, independently, to get a collection of results

- First step: design an abstract implementation
  - pick a model from the known ones
    - e.g. master-worker

Abstract implementation

- master-worker
  - single vs. multiple master
  - RPC vs. stream semantics
  - dynamic vs. static number of workers
- scheduling policy

Abstract implementation

- master-worker
  - **single** vs. multiple master
  - RPC vs. **stream semantics**
  - dynamic vs. **static** number of workers
  - scheduling policy: **round robin**

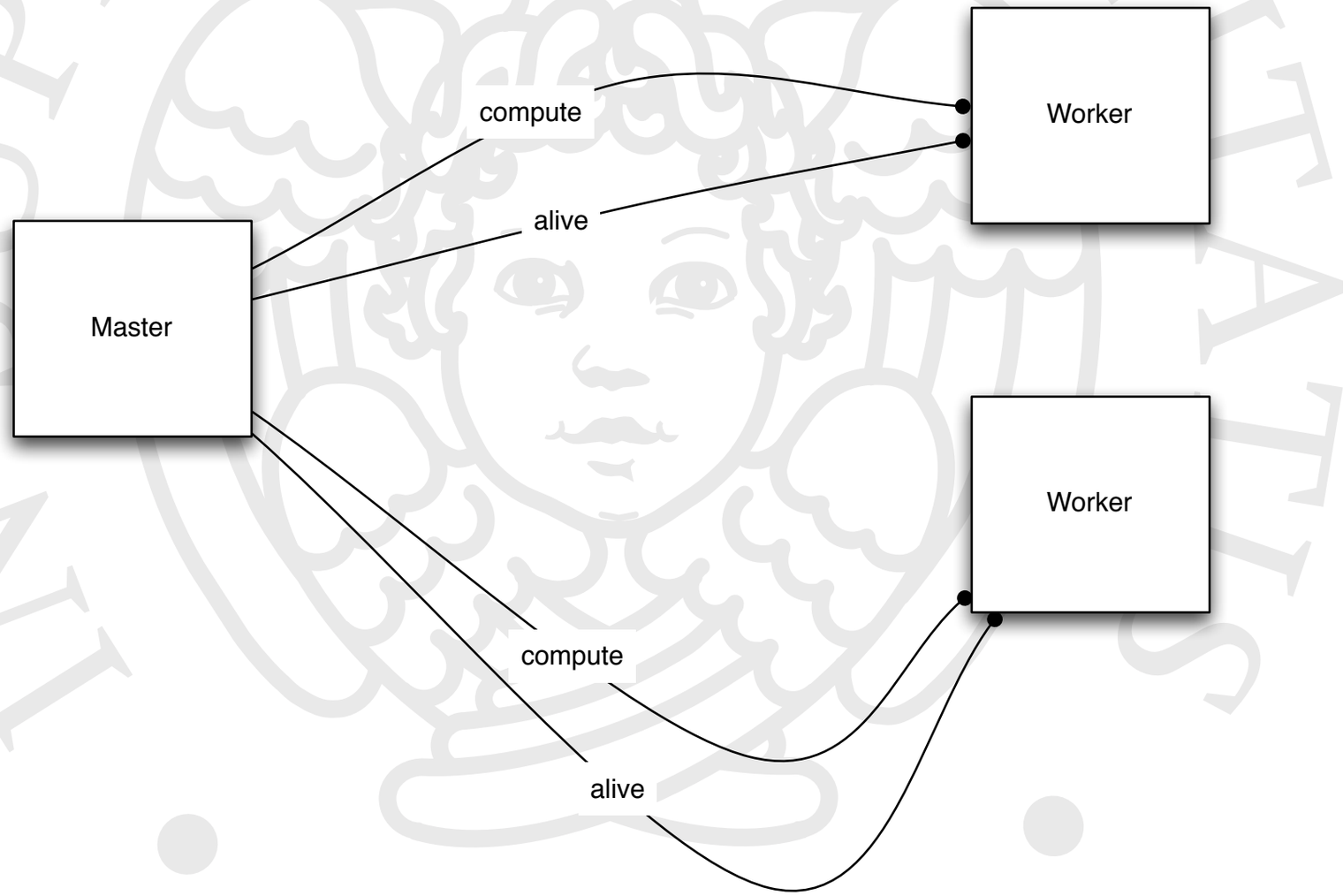
Identification of “gross” components

- master
  - single instance
  - interfaces
    - provides: void computeTask(Task)
    - uses: void deliverResult(Result)
- worker
  - multiple instances
  - interfaces
    - provides: Result compute(Task)

Identification of functionalities

- master
  - receiving tasks
  - scheduling tasks to workers
  - delivering results
- worker
  - receiving tasks
  - computing
  - delivering results

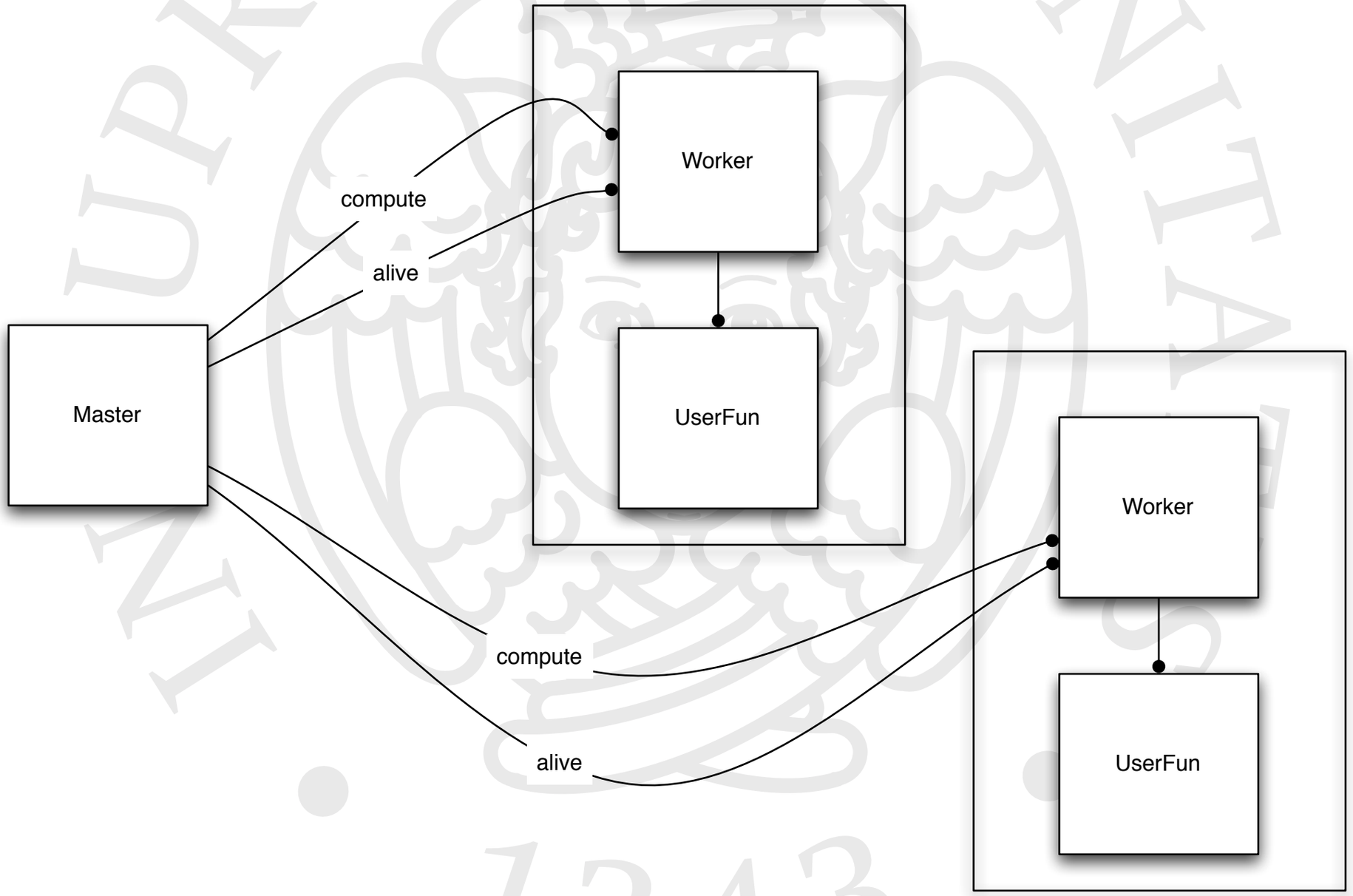
Sample schema



Futher worker refinement

- interaction with the manager
- interaction with the “user function”
  - which is not necessarily a worker
- interface: provides
  - Result compute(Task)
  - boolean alive()
- interface: uses
  - Result compute(Task)

Sample schema



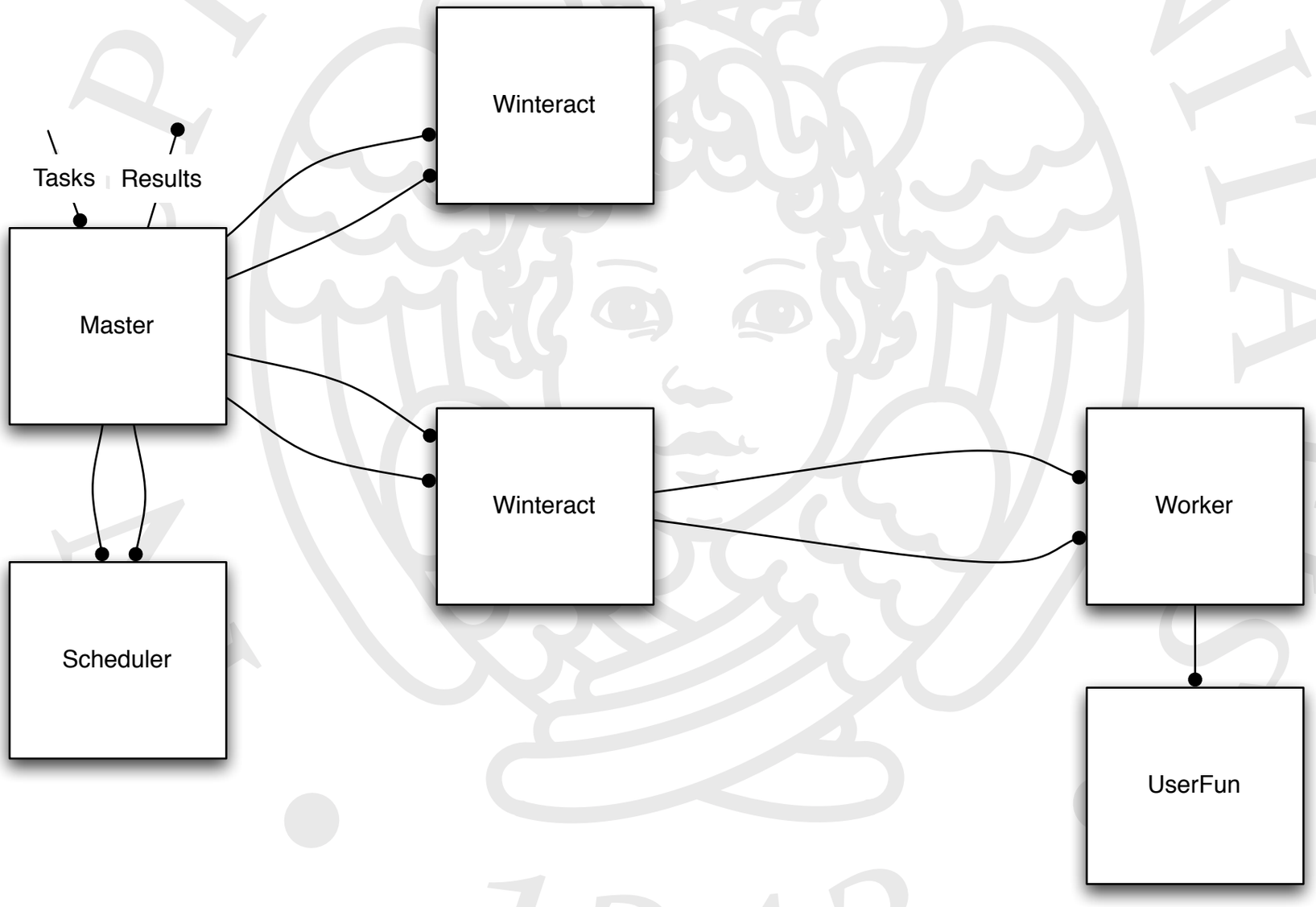
Further master refinement

- Master
  - scheduling policy
    - scheduling component
    - can be changed
  - interfaces
    - provides: void storeWorkers(Collection<Worker>) Worker getNext(Task)
  - implemented policies
    - round robin (e.g.)

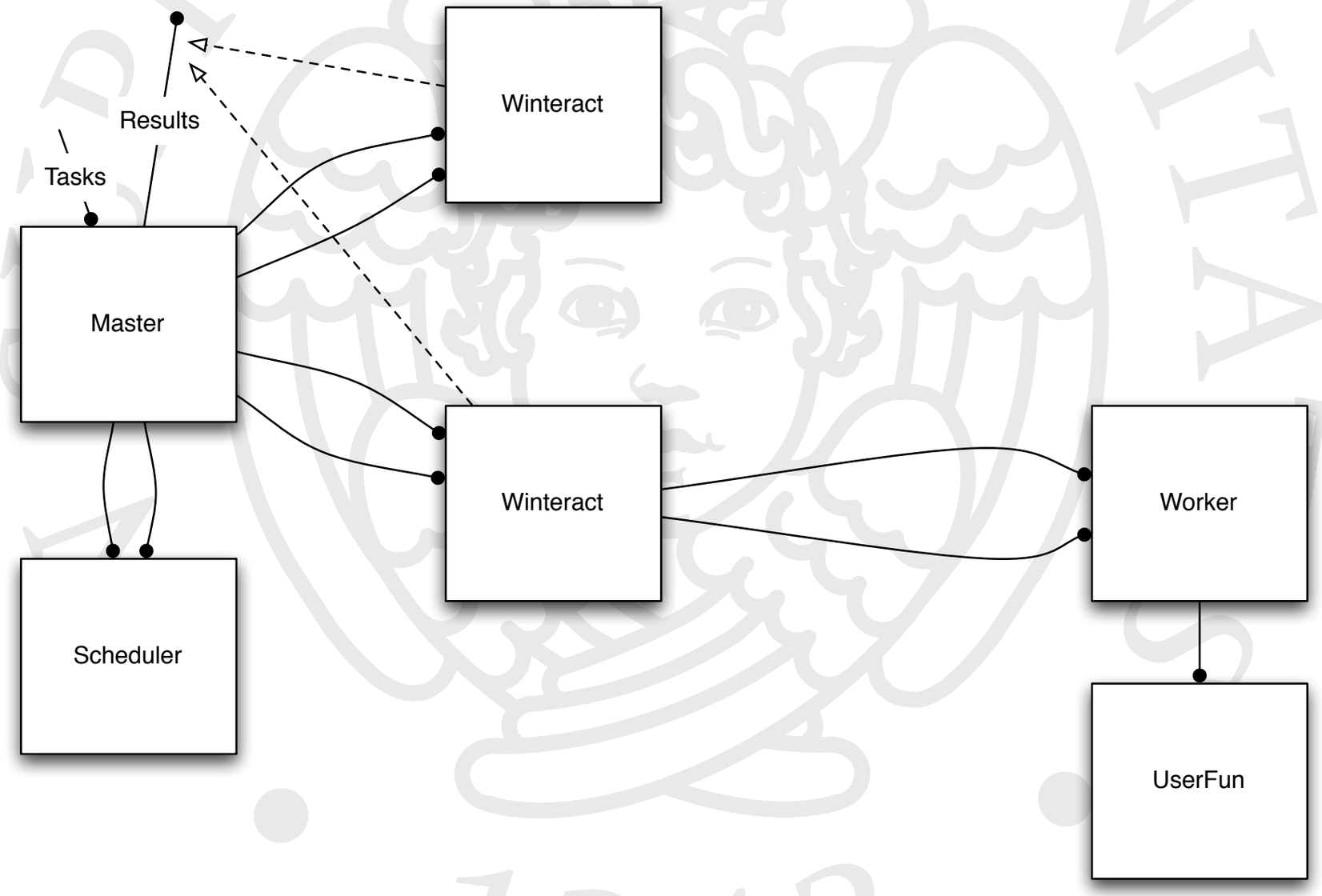
Further master refinement

- Winteraction
  - manage single worker interactions
  - interface: provides
    - void setWorker(Worker)
    - Result compute(Task)
- interface: uses
  - Result compute(Task) *towards worker*
  - boolean alive() *towards worker*
  - void raiseFailure() *towards master*

General view



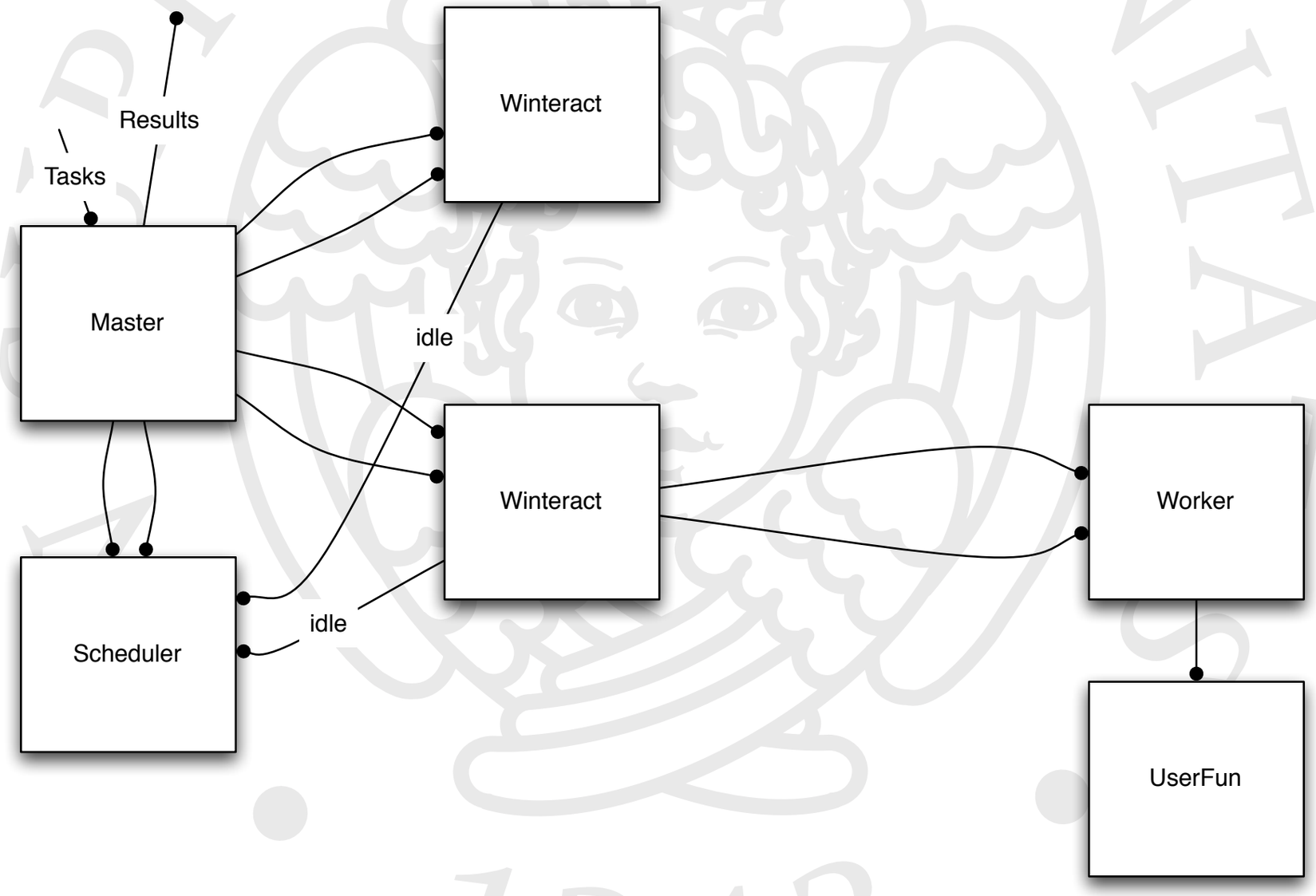
Reducing bottlenecks (master res deliv)



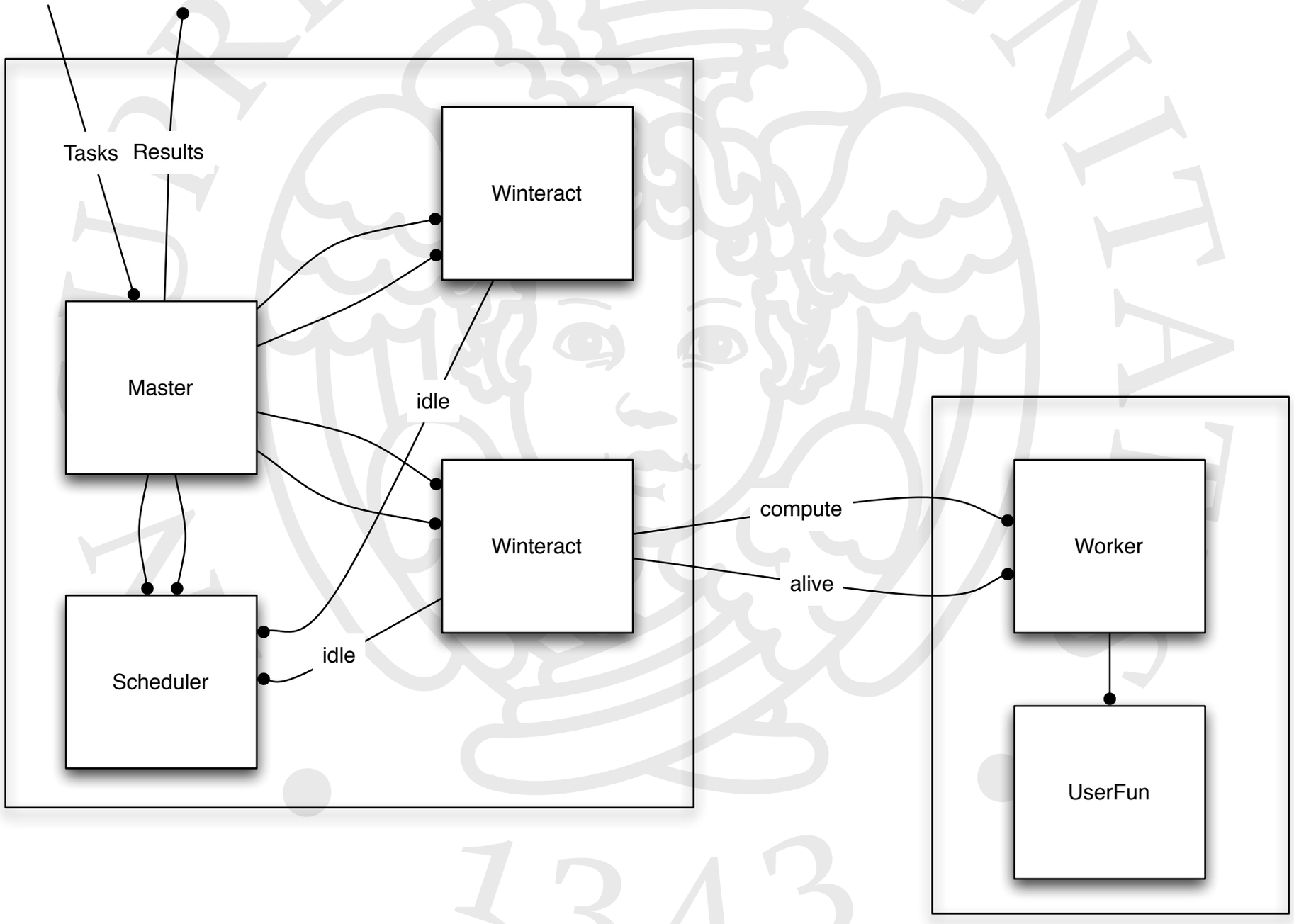
Changing scheduling policy

- Local policy
  - round robin (static, dynamic)
  - random scheduling
- Global policy
  - auto scheduling
  - requires information on idle workers
  - two possibilities
    - modify the interface Winteract and attach Sched comp.
    - modify the interface Master

Changing scheduling policy (2)



Component assembly



ADLs

- no way to have parametric interfaces (e.g. Winteract[i] on master)
  - master with n workers
    - has n copies of Winteract in the assembly
    - has n use/client interfaces
- possibility to use collective interfaces (defined in GCM)
  - scatter, multicast (for tasks)
  - gather (for results)
  - “rigid” semantics ...

Option 1

- programmatically
  - generate ADLs with suitable number of items
    - for(i=0; ...) *add interface, add w sub-component, ...*
- that is

  a)pre processing phase

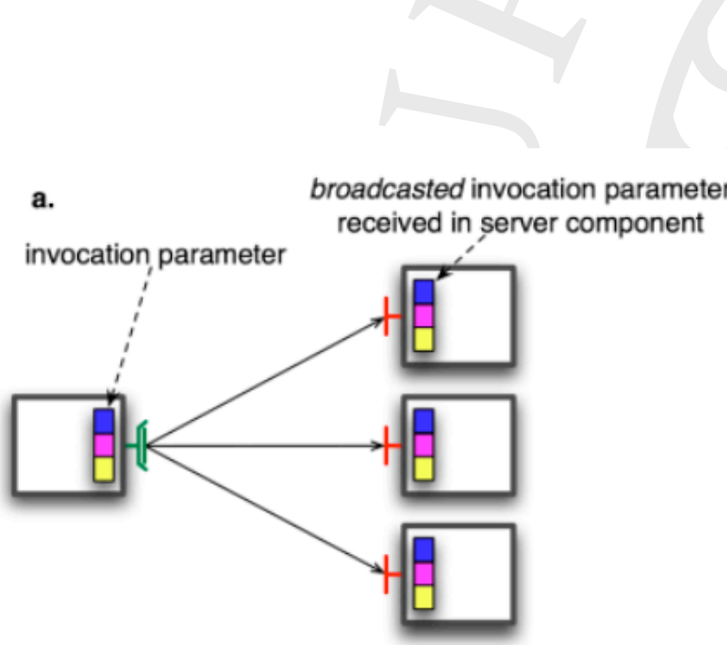
  b)component assembly run

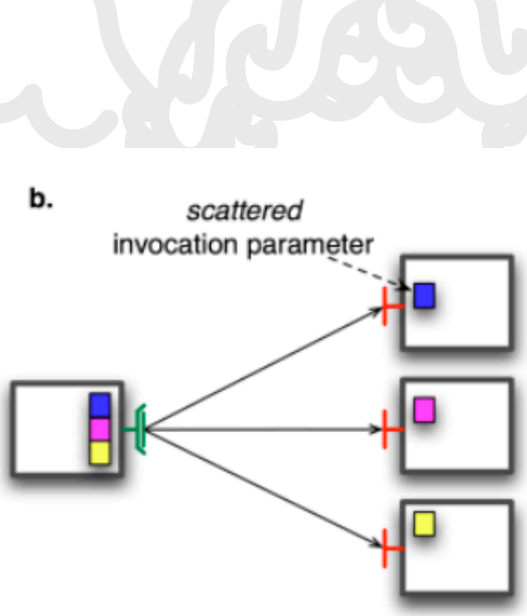
Option 2

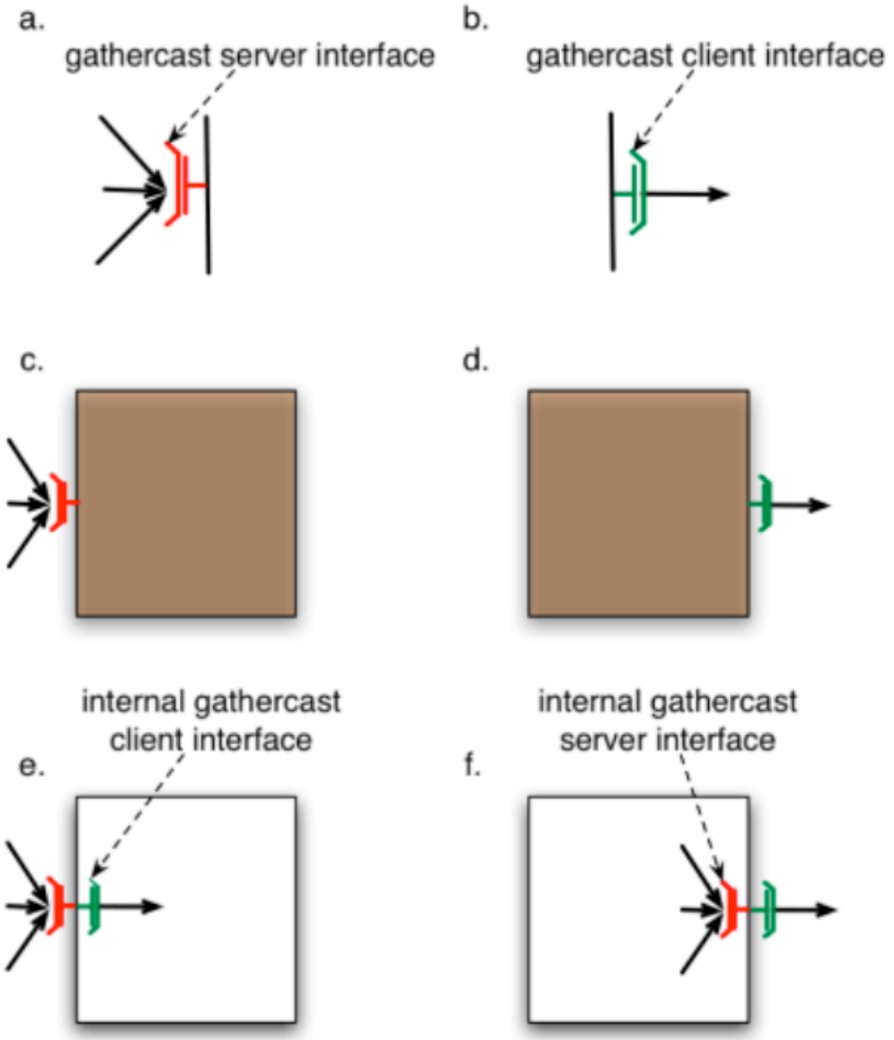
- compose components without ADL
  - programmatically compose components
  - no need to modify ADL “on the fly”
  - still needed to have proper number of interfaces in the component

Collective interfaces (GCM)

- Definition of 1-to-N and N-to-1 interfaces

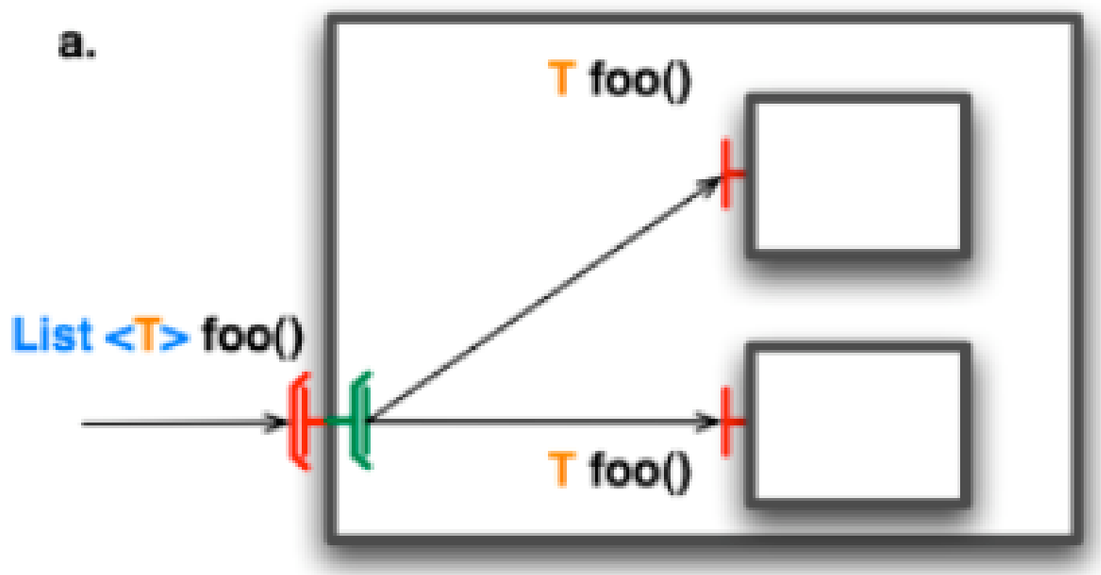






Scatter/Multicast

- in type
  - List<T>
- out type
  - collection of T ports

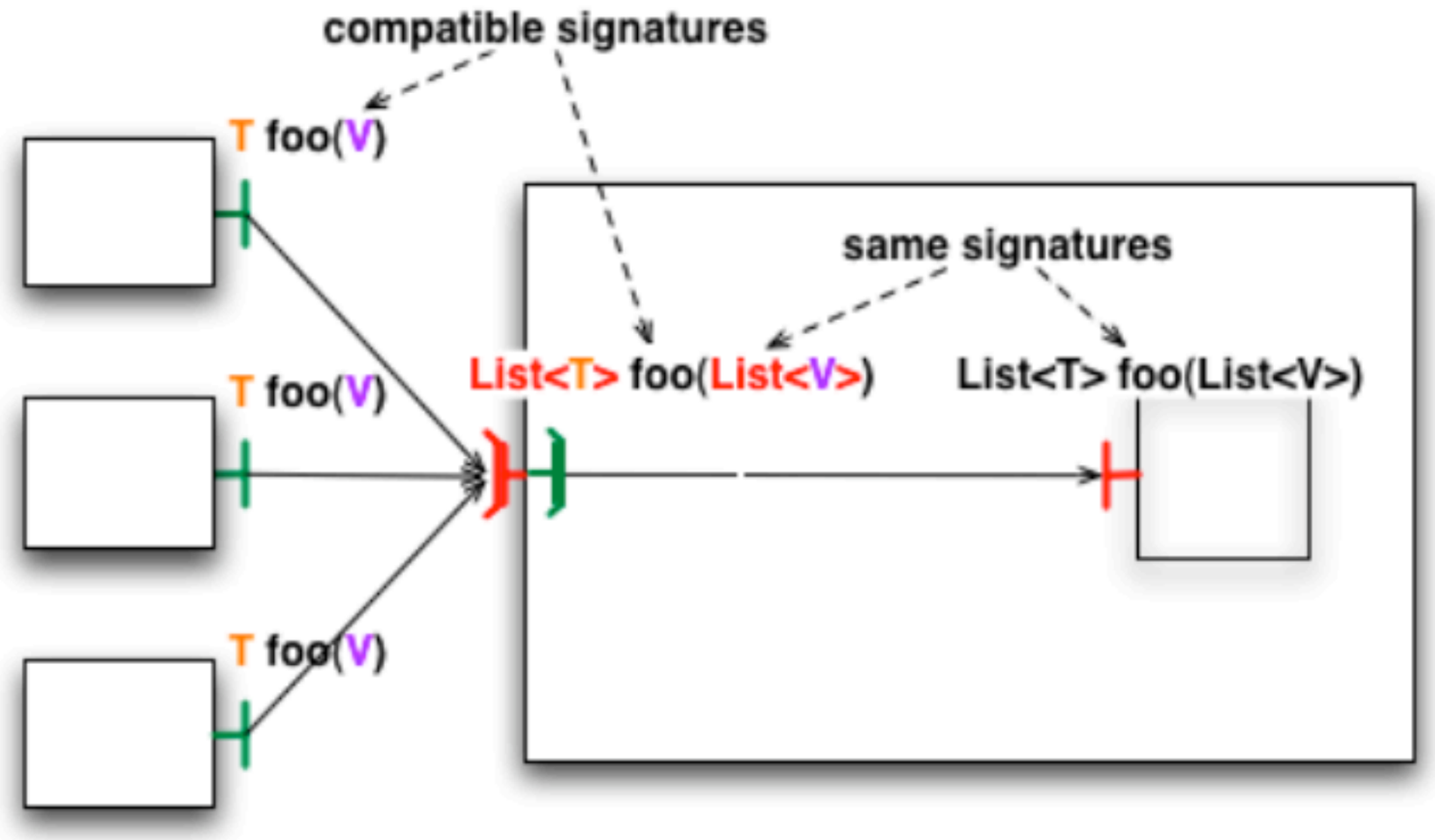


Scattering/multicasting policies

- BROADCAST
- ONE_TO_ONE (scatter)
- ROUND_ROBIN
- RANDOM
- UNICAST
  - specialized through proper multicast controller (user defined)

Gathercast

- out type
  - V
- in type
  - LIST<V>



Sample multicast code

- see
  - src/Examples/org/objectweb/proactive/examples/components/userguide/multicast
- in ProActive distribution (4.0.2)