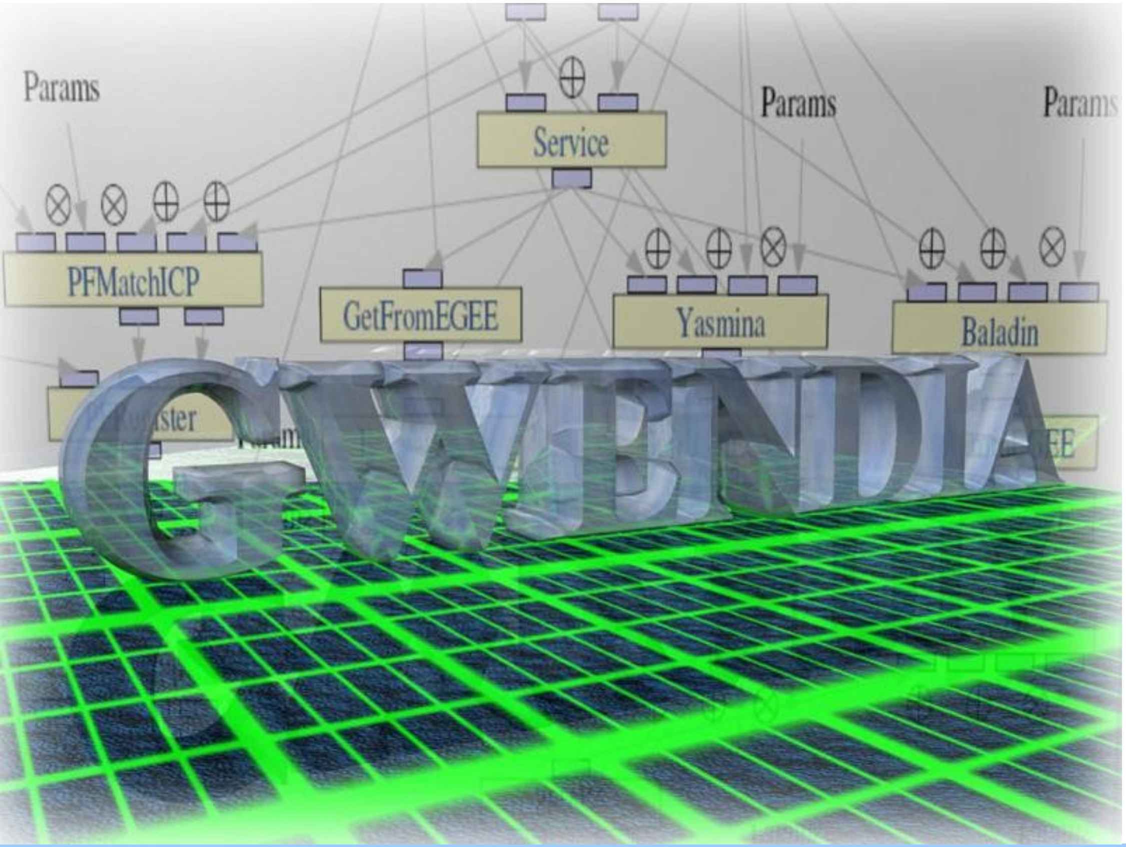
Software composition for scientific workflows



Johan Montagnat
CNRS / UNS, I3S, MODALIS
April 17, 2009

Financé par ANR

Grid infrastructures

egee Enabling Grids for E-sciencE | Open Science Grid | NAREGI

- **Scientific production**
  - Europe: EGEE (www.eu-egee.org), USA: OSG (www.opensciencegrid.org), NAREGI (www.naregi.org)...

EGEE European infrastructure:

> 250 computing centers in 45 countries

> 80 000 CPU cores

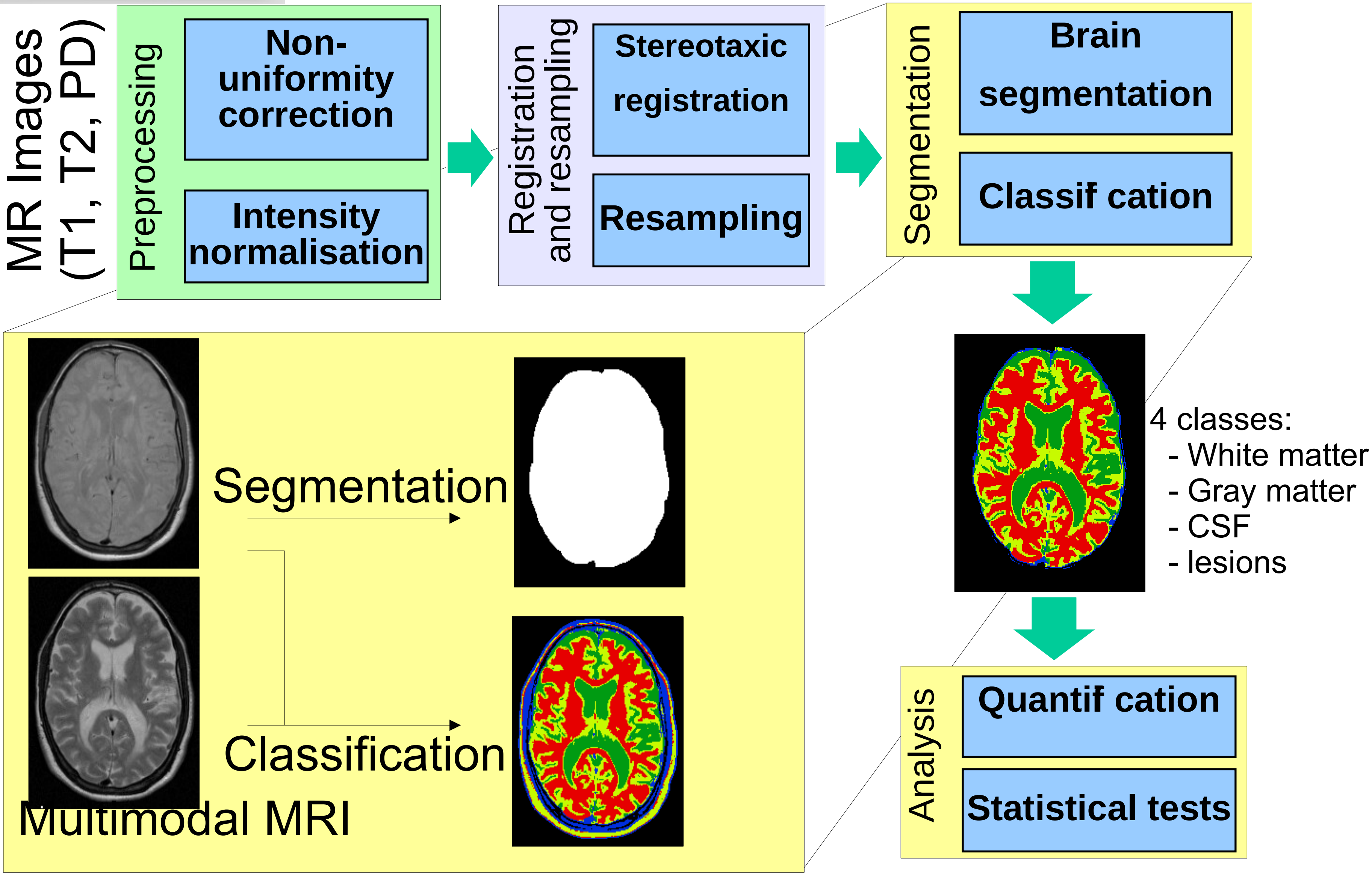
> 13 000 users

> 250 000 jobs / day

- **behavior**
  - Very large scale (WAN)
  - Coarse grain parallelism, loose coupling
  - High heterogeneity, low reliability, large latencies

Images analysis pipelines

Grid Workflow Efficient Enactment for Data Intensive Applications



Images analysis pipelines

Grid Workflow Efficient Enactment for Data Intensive Applications

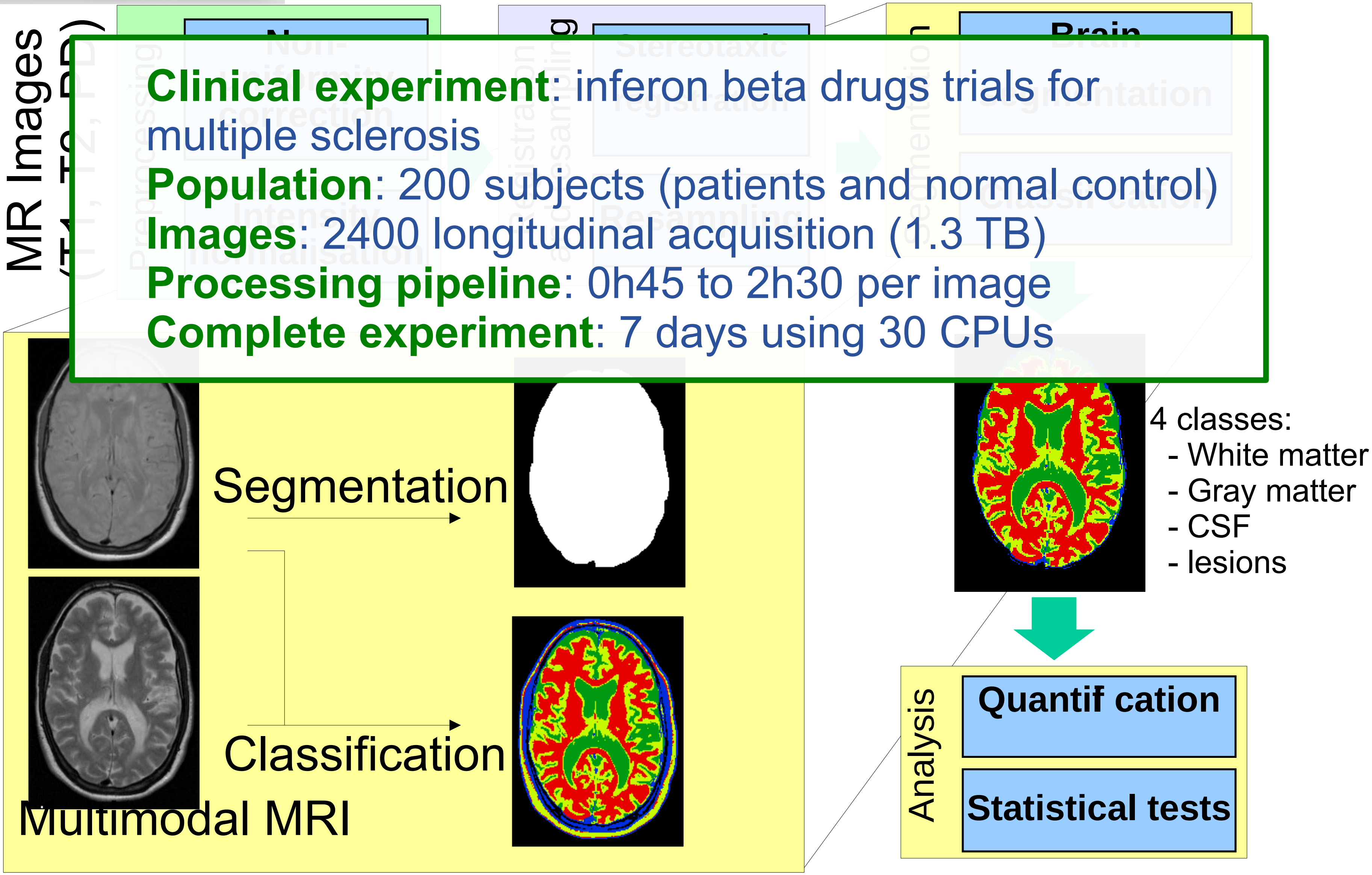
Clinical experiment: inferon beta drugs trials for multiple sclerosis

Population: 200 subjects (patients and normal control)

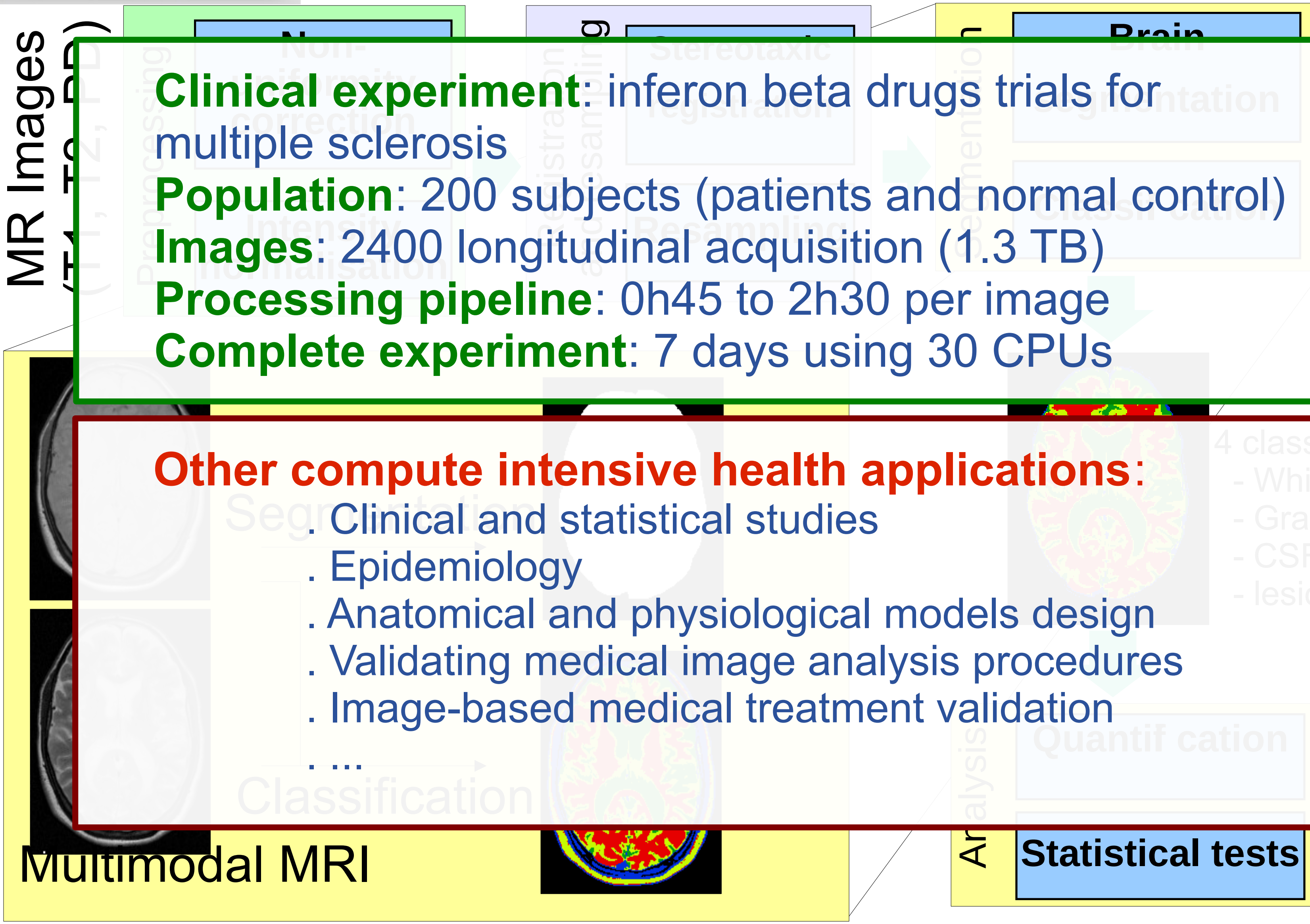
Images: 2400 longitudinal acquisition (1.3 TB)

Processing pipeline: 0h45 to 2h30 per image

Complete experiment: 7 days using 30 CPUs



Images analysis pipelines



Clinical experiment: inferon beta drugs trials for multiple sclerosis
Population: 200 subjects (patients and normal control)
Images: 2400 longitudinal acquisition (1.3 TB)
Processing pipeline: 0h45 to 2h30 per image
Complete experiment: 7 days using 30 CPUs

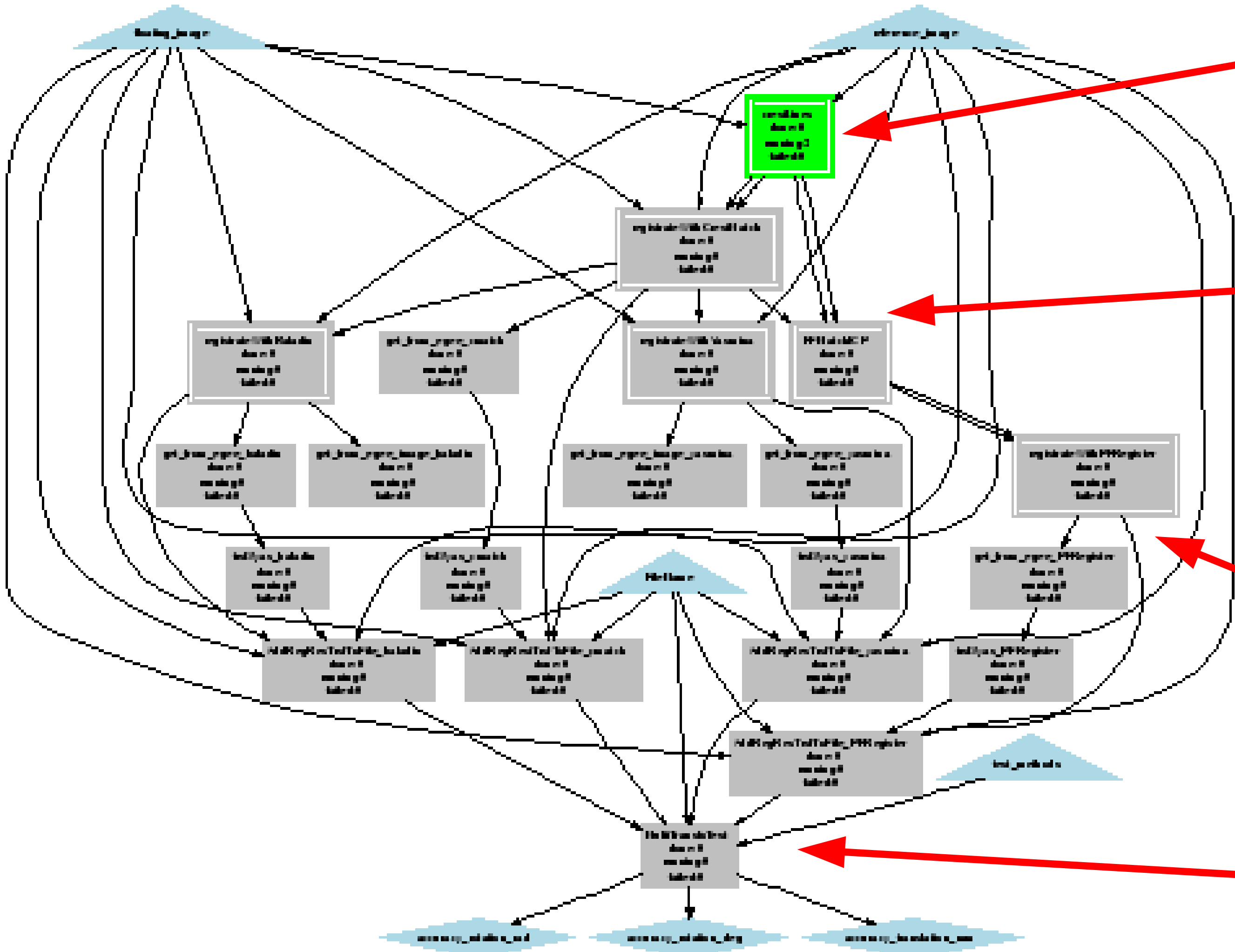
Other compute intensive health applications:

. Clinical and statistical studies
. Epidemiology
. Anatomical and physiological models design
. Validating medical image analysis procedures
. Image-based medical treatment validation
. ...

Scientific application workflows

Grid Workflow Efficient Enactment for Data Intensive Applications

- **Application logic description**
  - Coarse grain, data defined independently

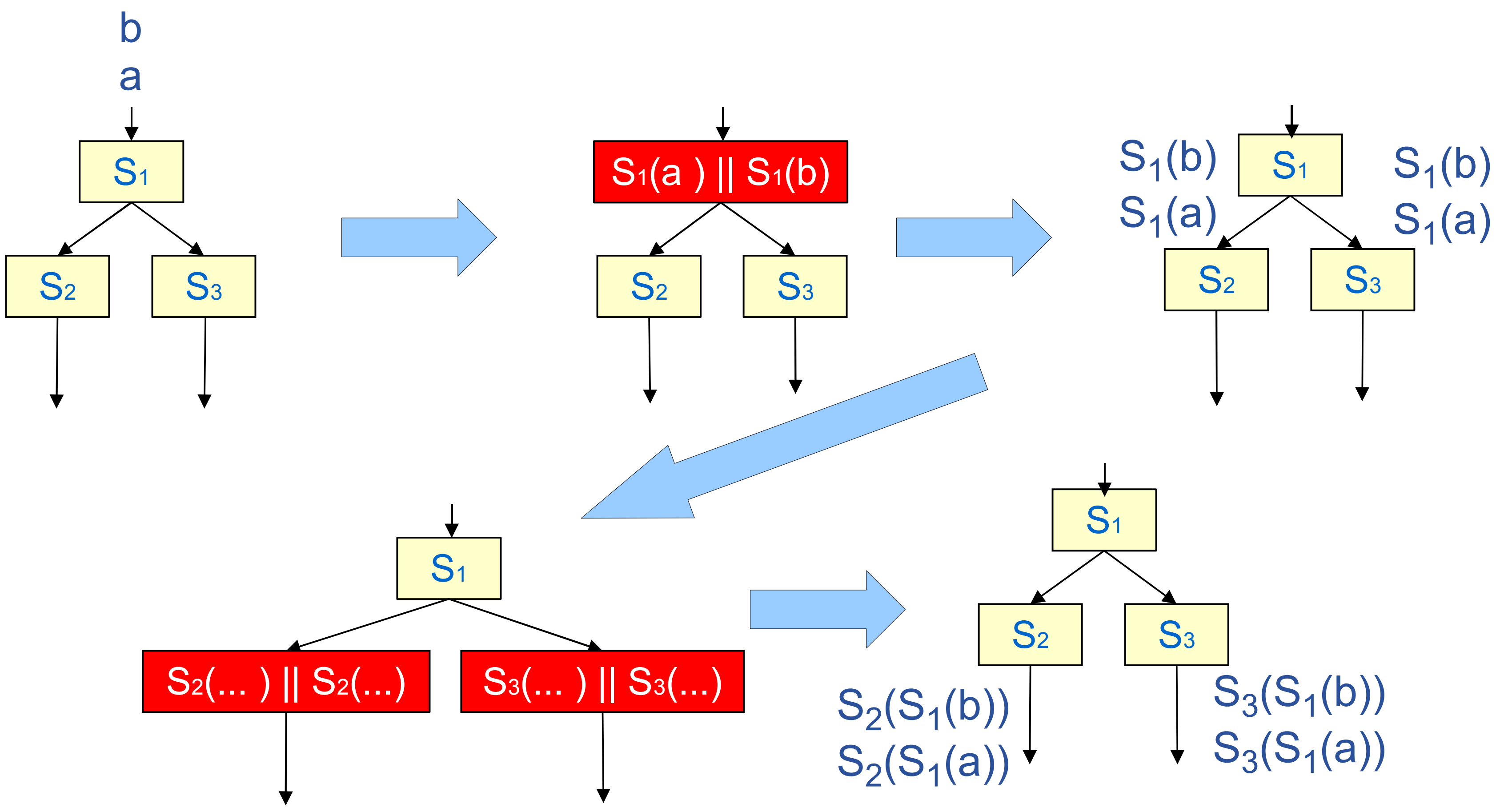


Data intensive medical imaging

- **Application community**
  - Compute and data intensive applications
  - Non-expert end users
  - Distributed (medical centers)
- **Coarse grain parallelism**
  - Grid computing
  - Massive data parallelism
- **Platform independence**
  - Common representation / submission interface to
    - Different grids
    - Multiple grids
- **Pipelines are pure data flows**
  - Successive image processing filters
  - Data intensive and **data driven**

Dynamic data flow composition

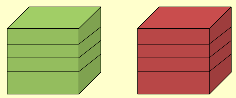
- **Data flow resolved dynamically**
- **Services invoked multiple times**



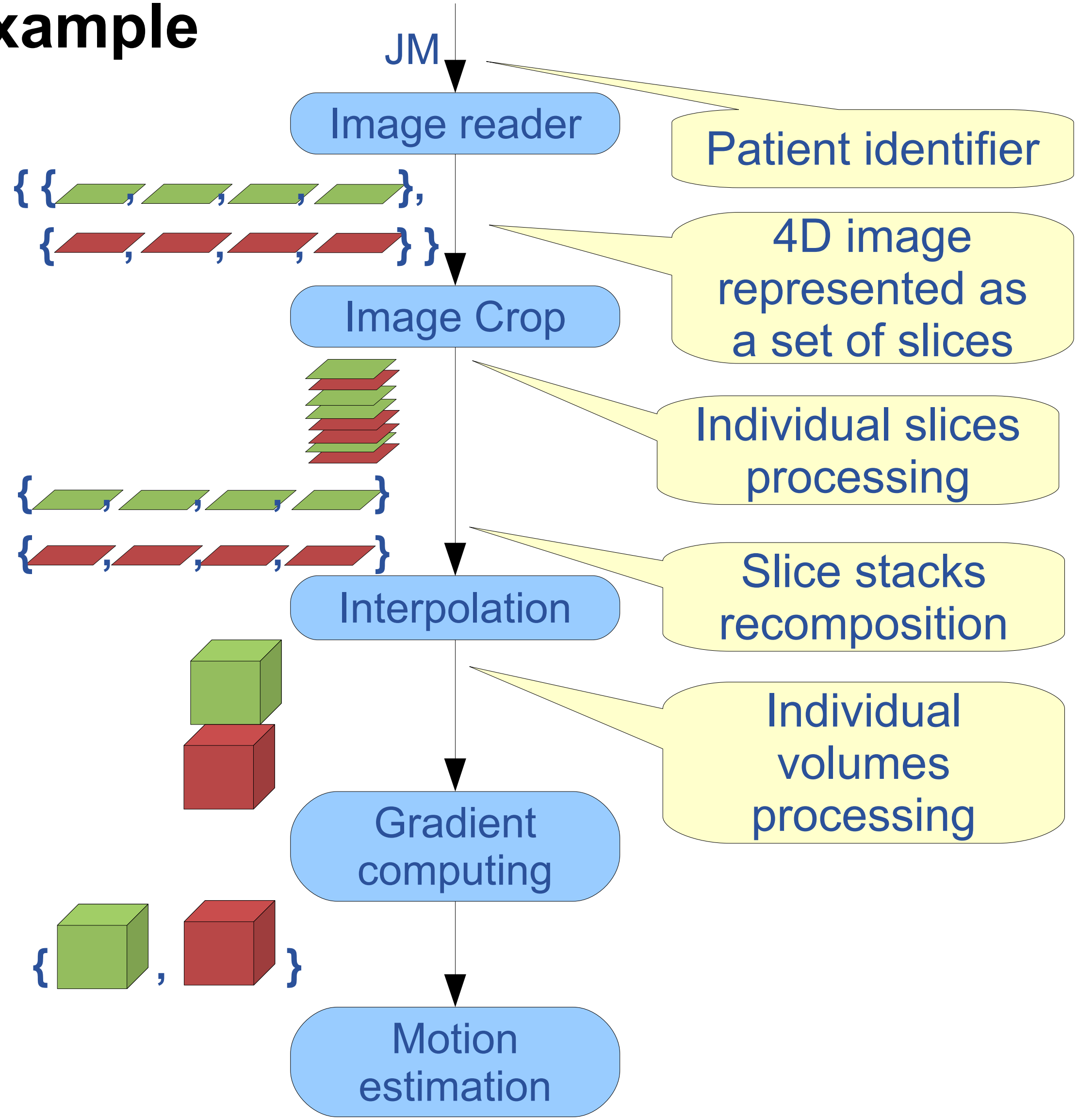
Complex data flows

Grid Workflow Efficient Enactment for Data Intensive Applications

- **Cardiac sequences example**

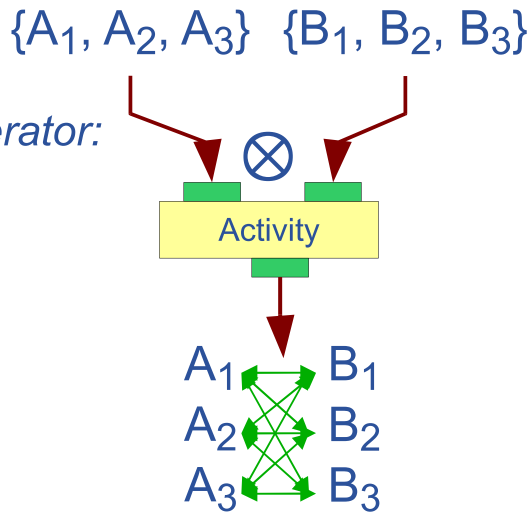


The input data set is a 4D image composed of 2 volumes (labelled green and red). Each volume is composed of 4 slices.



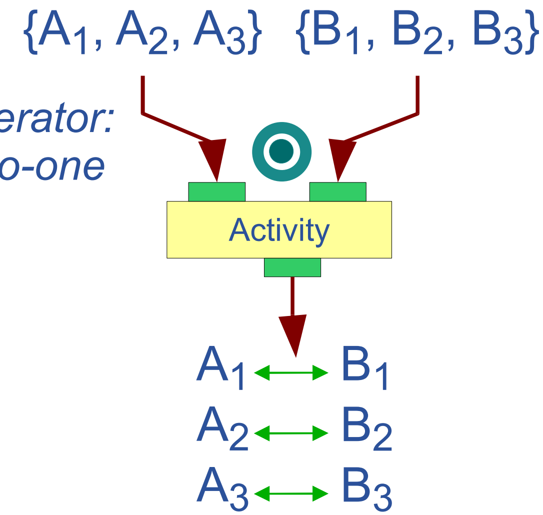
Data flows composition

cross iterator: all-to-all



$\mathbf{A} \otimes \mathbf{B}$

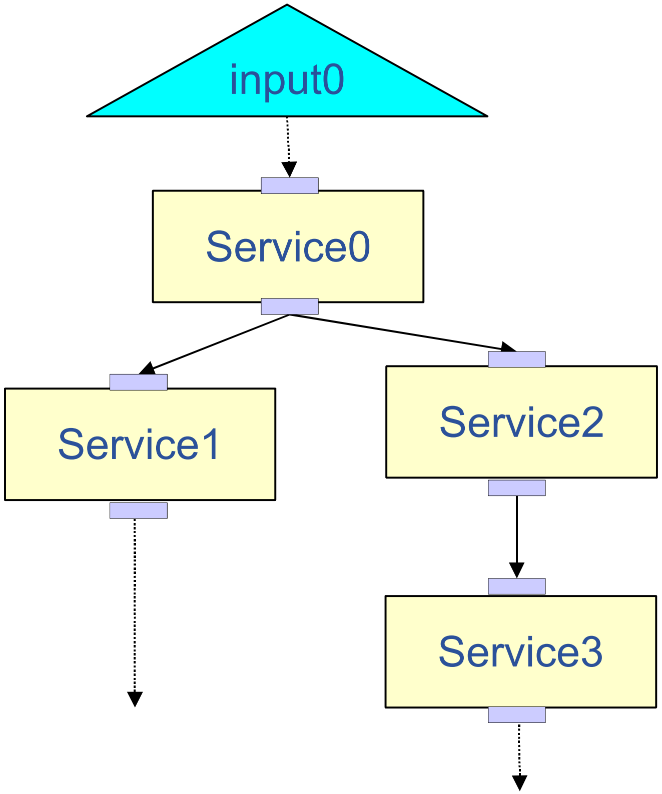
dot iterator: one-to-one

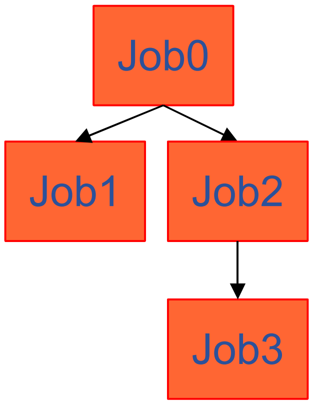


$\mathbf{A} \odot \mathbf{B}$

Scufl data flow definition

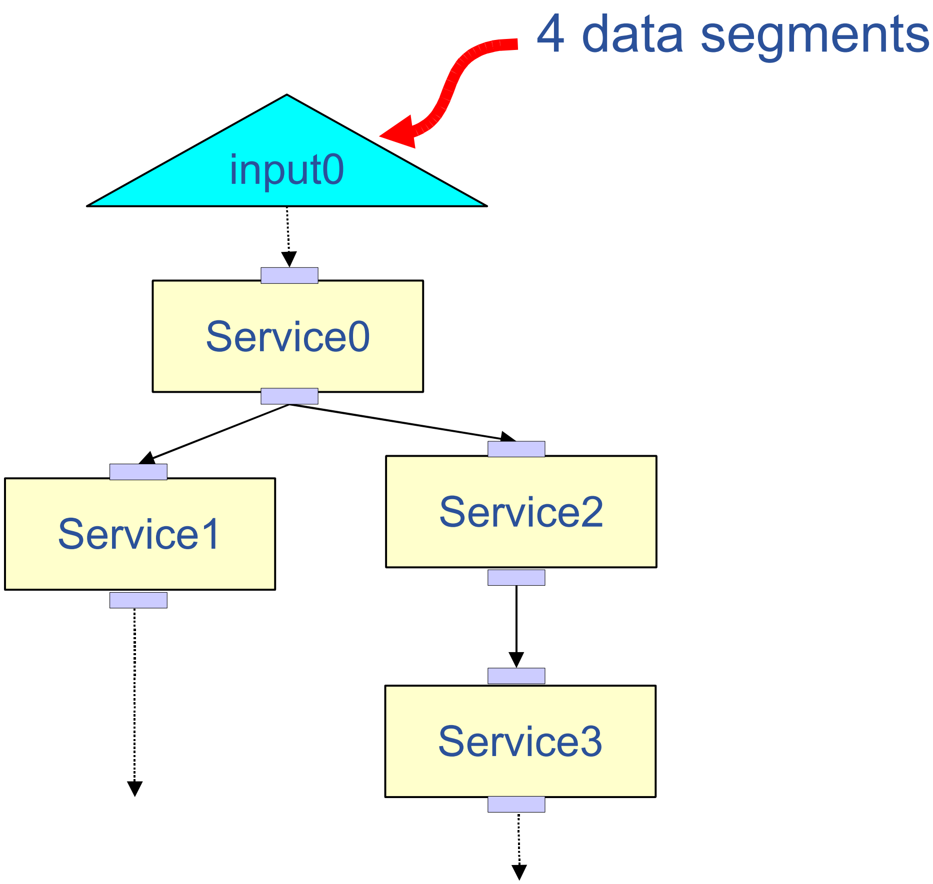
- **Graph of services (+ data)** **DAG of tasks**

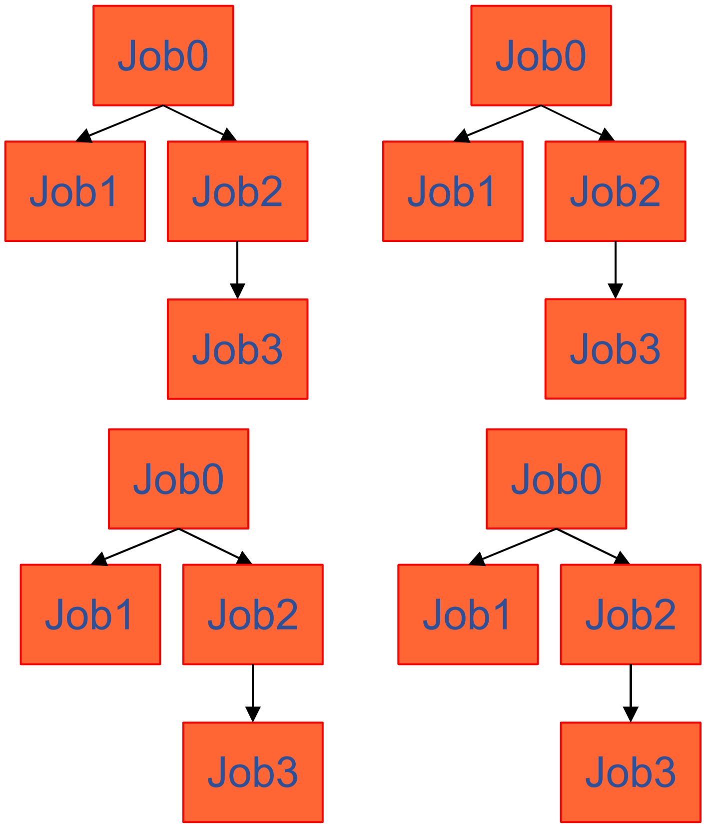




Scufl data flow definition

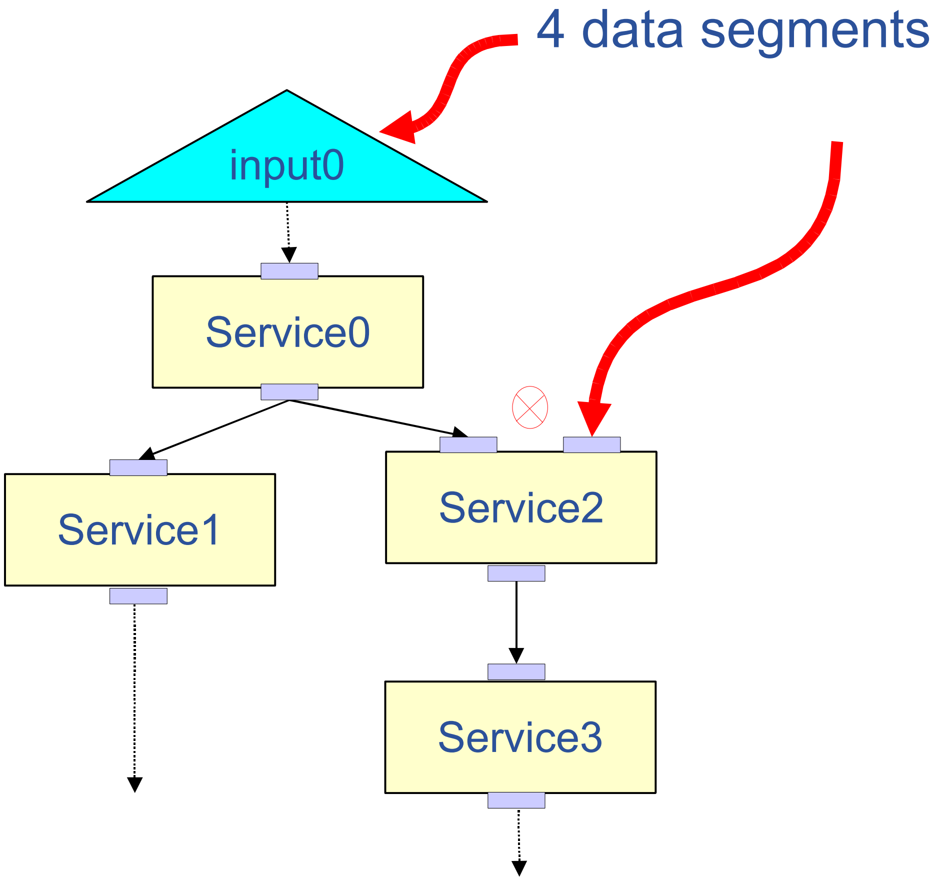
- **Graph of services (+ data)** **DAG of tasks**

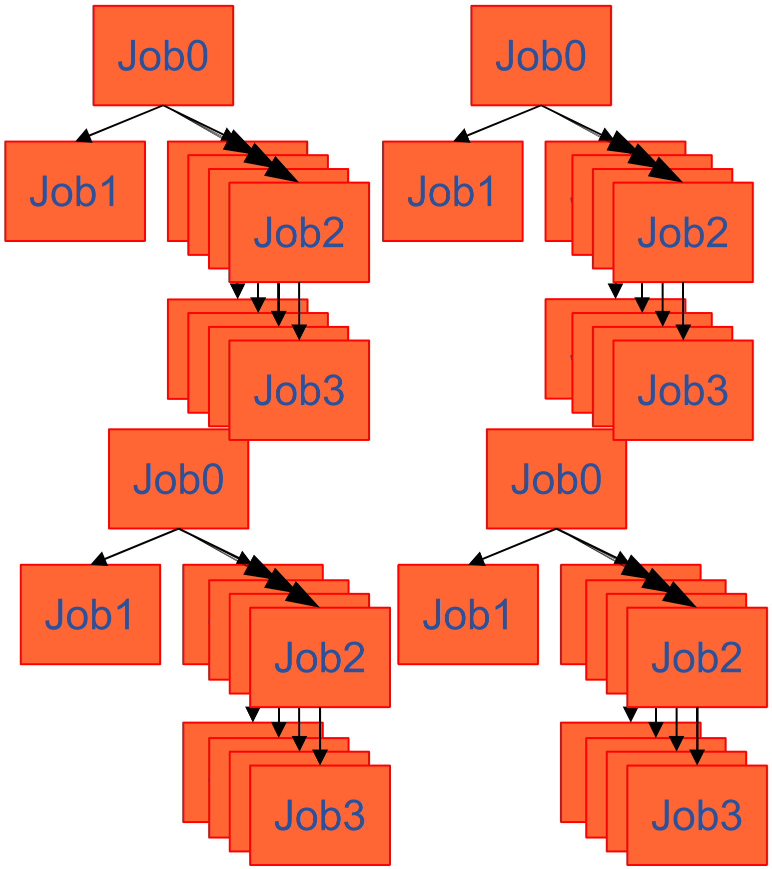




Scufl data flow definition

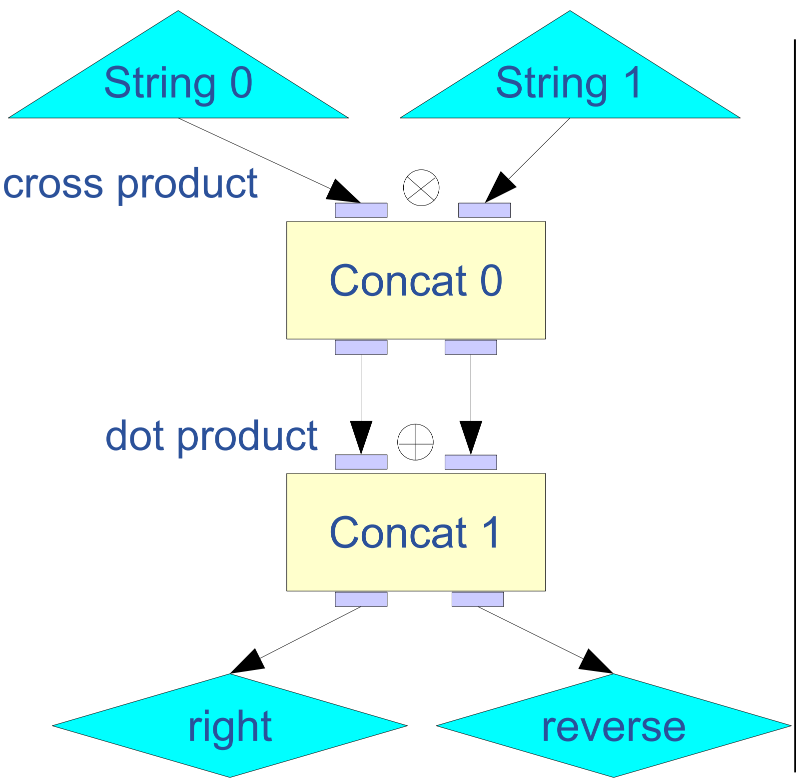
- **Graph of services (+ data)** **DAG of tasks**



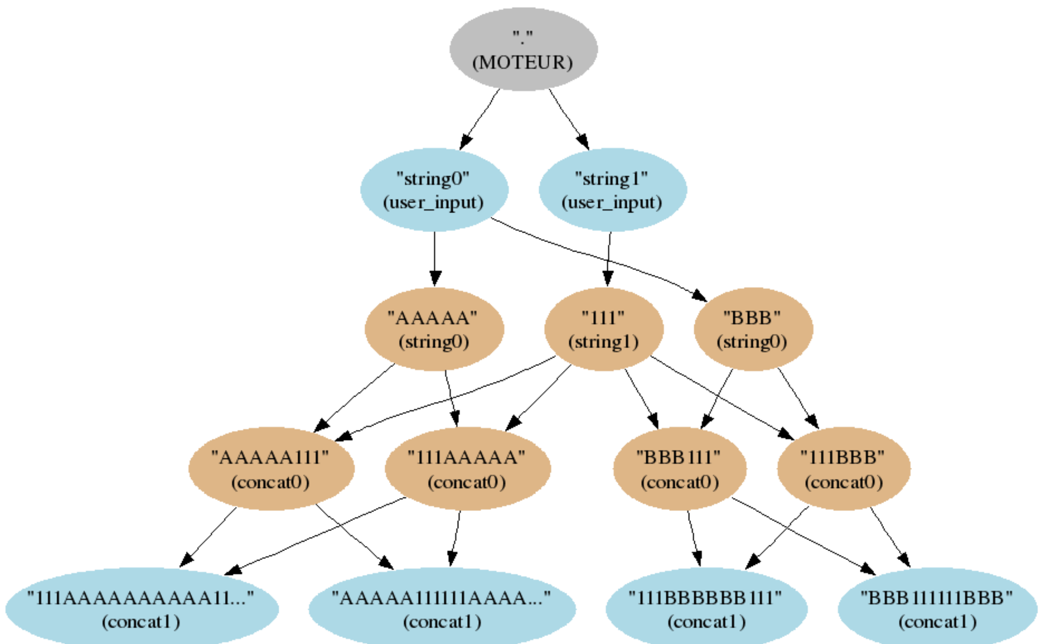


Computation history

- **This data representation allows to:**
  - Handle dot products iteration strategies if data segments are puzzled
  - Retrieve results provenance



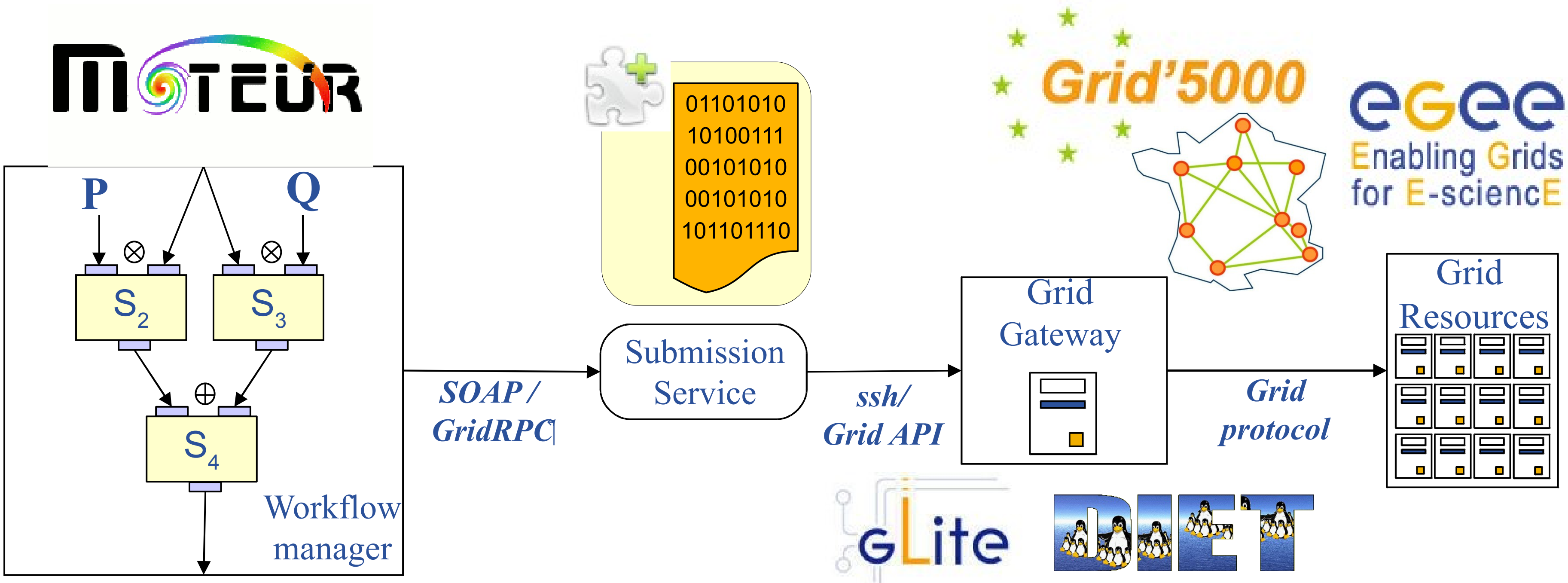
Data graph



MOTEUR workfklow manager

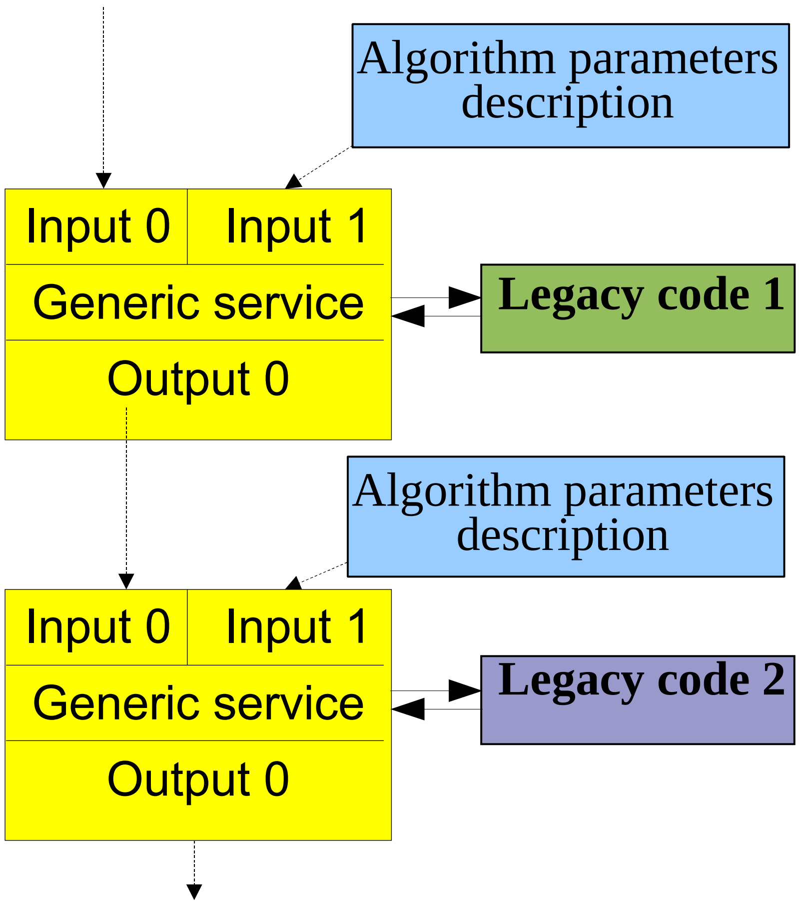
- **Open source workflow enactor**
  - Code + docs + tutorial: http://egee1.unice.fr/MOTEUR
  - Developped at the I3S CNRS laboratory
  - With the support of French national projects
    - AGIR: http://www.agir.org
    - GWENDIA: http://gwendia.polytech.unice.fr
- **Targets**
  - Ease of use, flexibility, service-oriented approach
  - Performance, transparent exploitation of application parallelism
- **Supports**
  - Scufl language (from myGrid/Taverna)
  - Service based invocation (WS)
  - Grid middlewares (EGEE / Grid'5000)

Grid enactment



- **Batch-oriented infrastructure: no deployment**
  - EGEE / gLite middleware
  - Grid'5000 / OAR, using idle resources
- **Service oriented infrastructure: pre-deployment**
  - Web Services
  - DIET Services

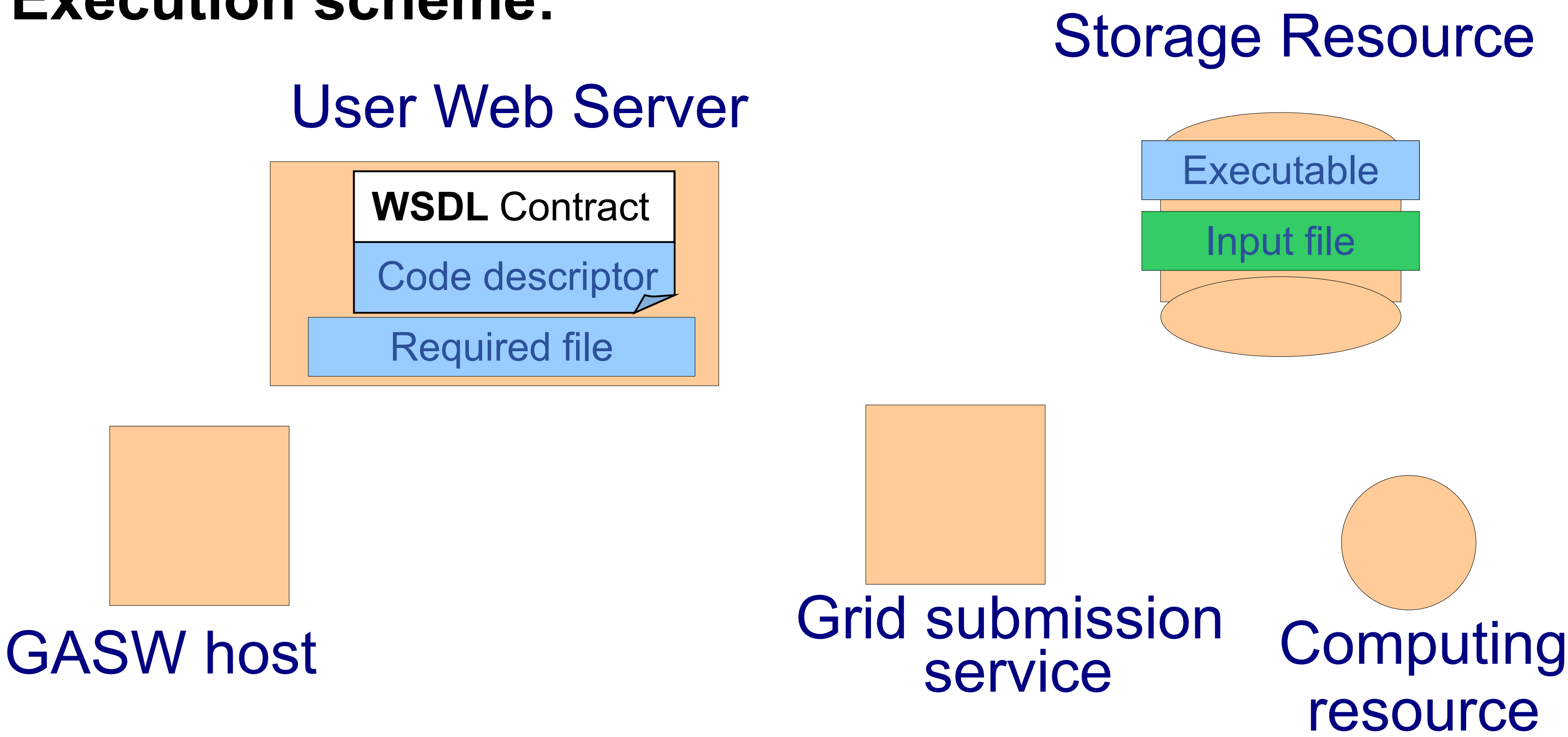
- **Provide service wrapper to non instrumented code**
- **Handle data transfers (references to grid data)**




```
<description>
  <executable name="CrestLines.pl">
    <access type="URL">
         <path value="http://colors.unice.fr:80/"/>
    </access>
    <value value="CrestLines.pl"/>
    <input name="image" option="-im1">
         <access type="LFN" />
    </input>
    <input name="scale" option="-s"/>
    <output name="crest_lines" option="-c2">
         <access type="LFN" />
    </output>
    <sandbox name="convert8bits">
        <access type="URL">
           <path value="http://colors.unice.fr:80/"/>
        </access>
        <value value="Convert8bits.pl"/>
    </sandbox>
  </executable>
</description>
```

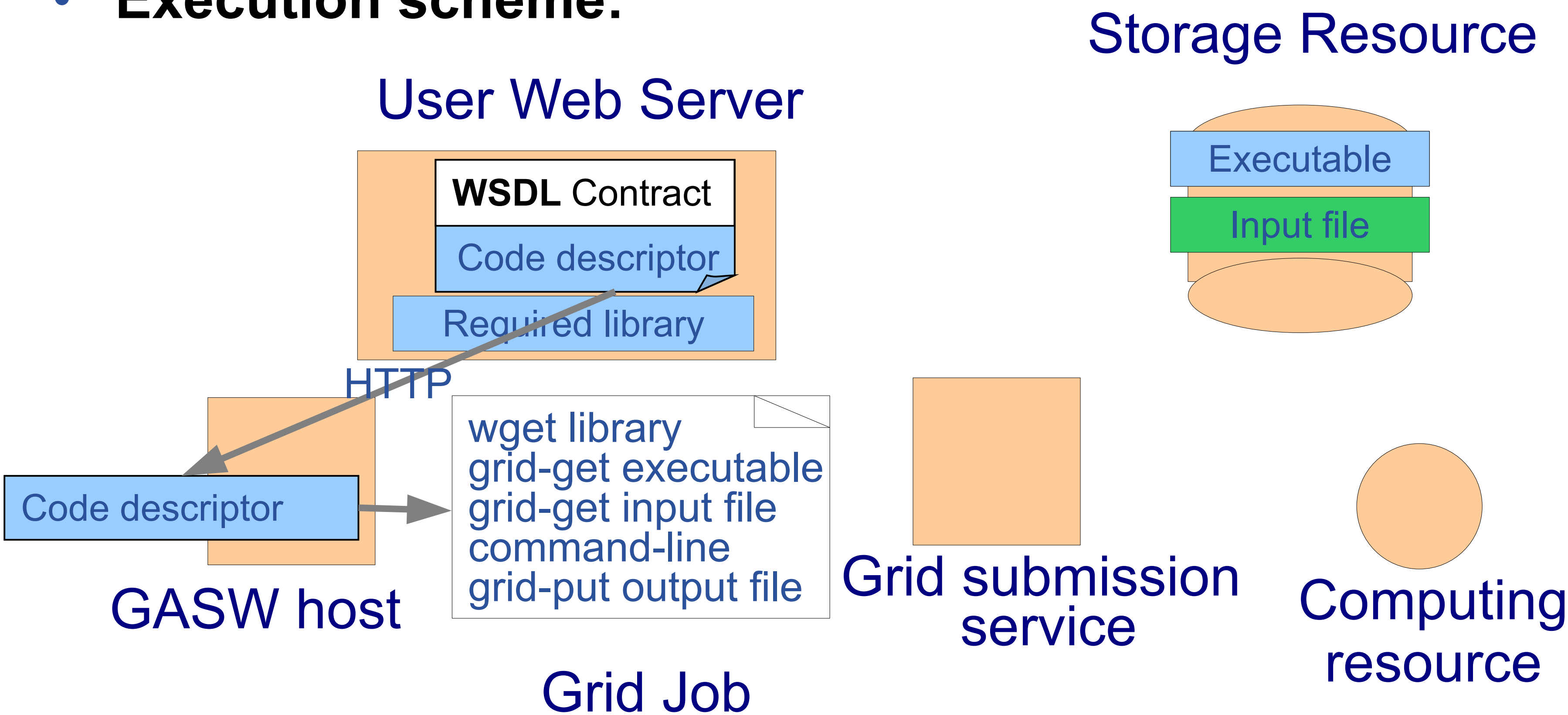
GASW Service

- **Generic Application Service Wrapper**
  - Provide service wrapper to non instrumented code
  - Handle data transfer (references to grid data)
- **Execution scheme:**



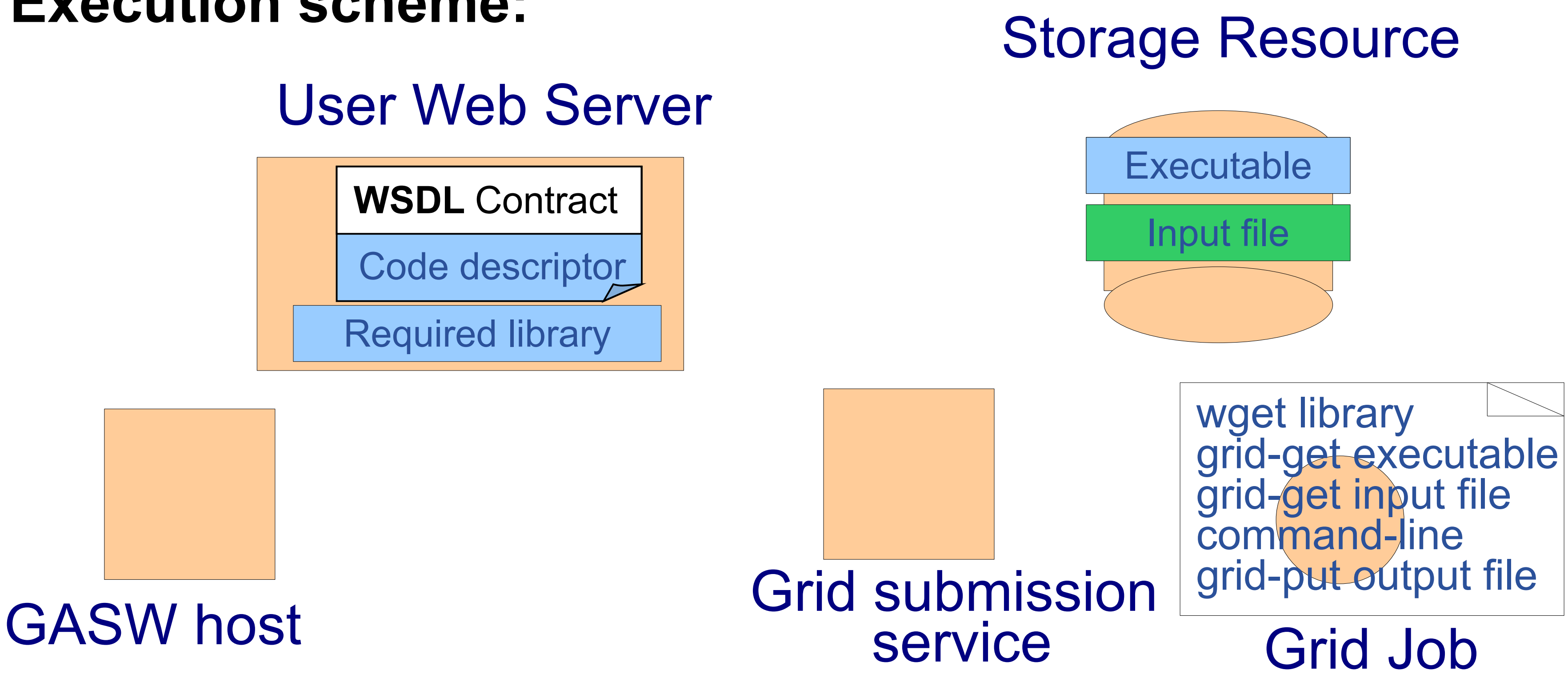
Dynamic wrapping

- **Generic Application Service Wrapper**
  - Provide service wrapper to non instrumented code
  - Handle data transfer (references to grid data)
- **Execution scheme:**



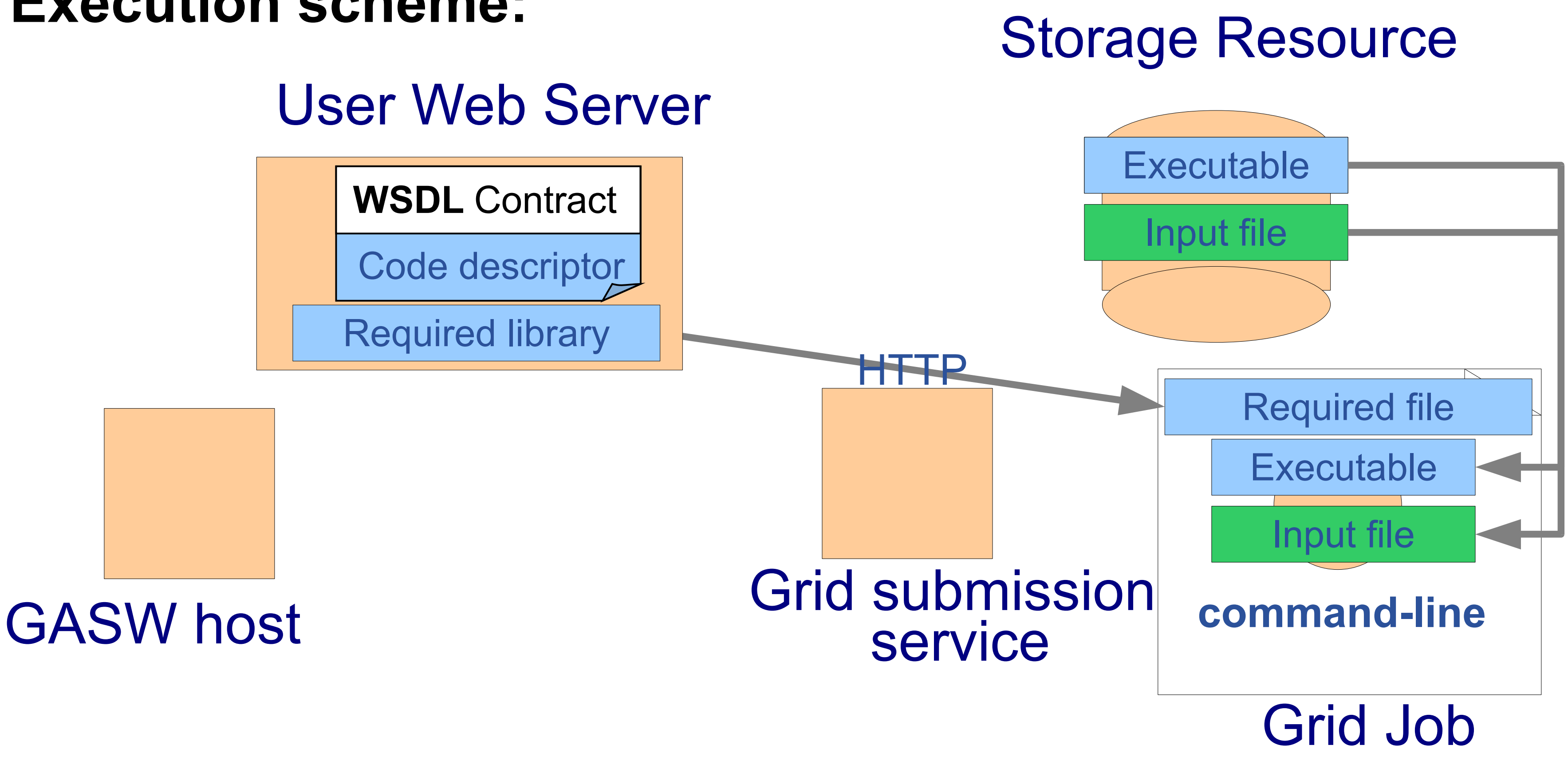
Dynamic wrapping

- **Generic Application Service Wrapper**
  - Provide service wrapper to non instrumented code
  - Handle data transfer (references to grid data)
- **Execution scheme:**



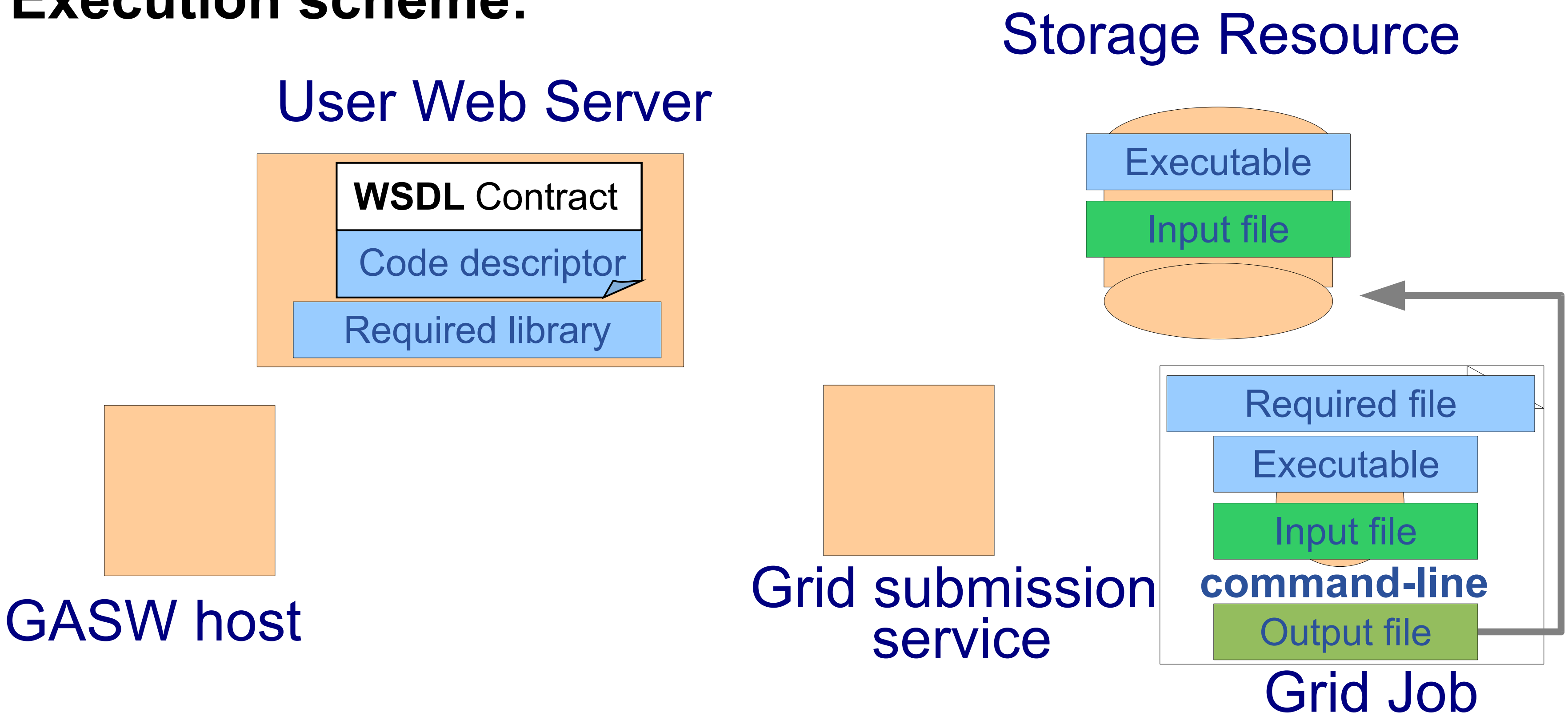
Dynamic wrapping

- **Generic Application Service Wrapper**
  - Provide service wrapper to non instrumented code
  - Handle data transfer (references to grid data)
- **Execution scheme:**



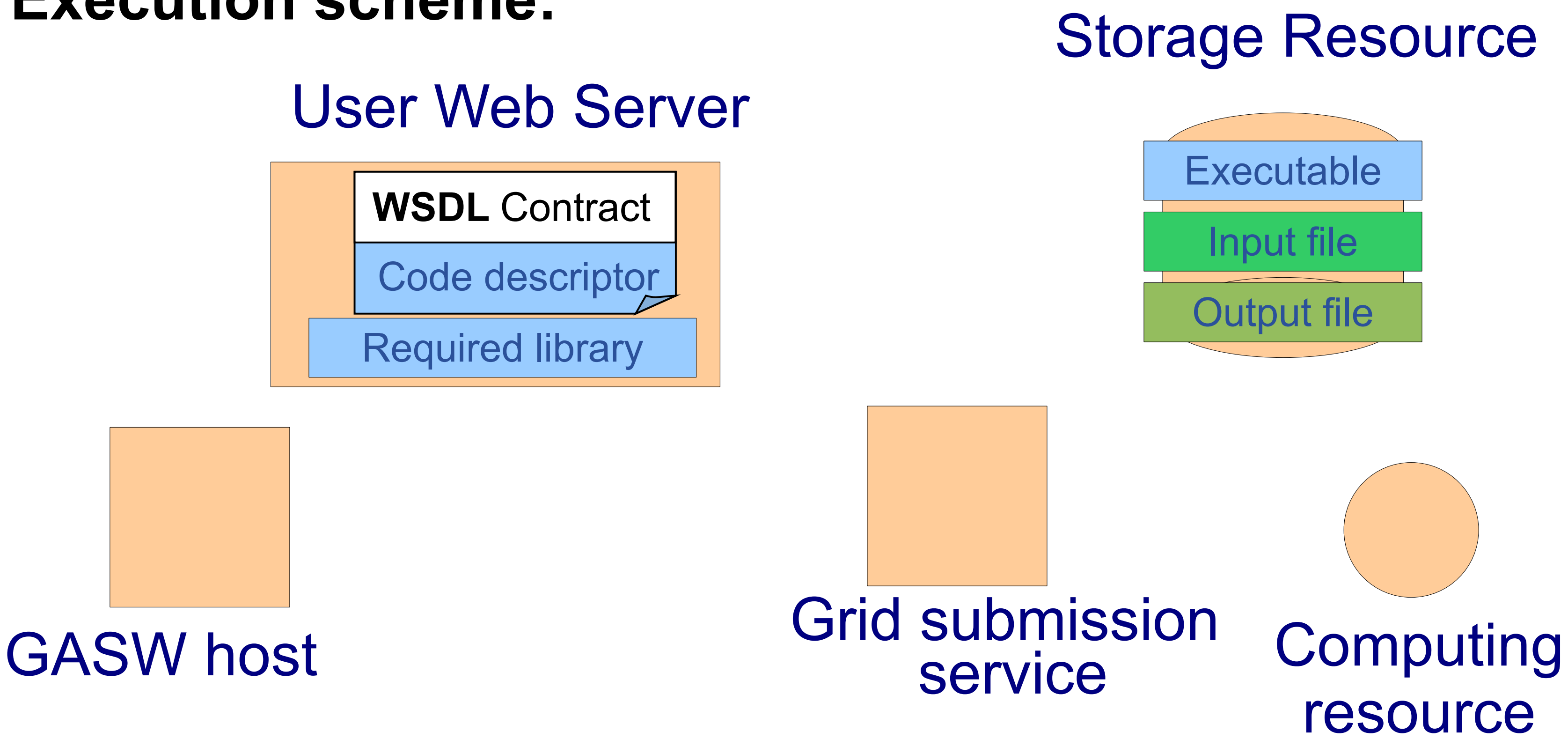
Dynamic wrapping

- **Generic Application Service Wrapper**
  - Provide service wrapper to non instrumented code
  - Handle data transfer (references to grid data)
- **Execution scheme:**



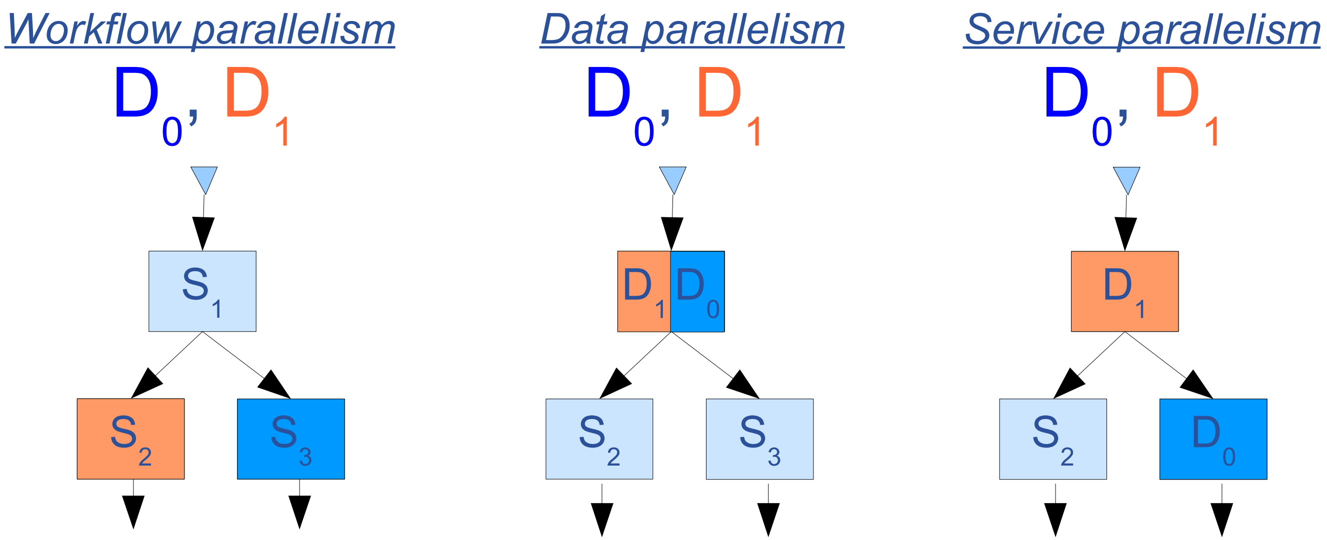
Dynamic wrapping

- **Generic Application Service Wrapper**
  - Provide service wrapper to non instrumented code
  - Handle data transfer (references to grid data)
- **Execution scheme:**



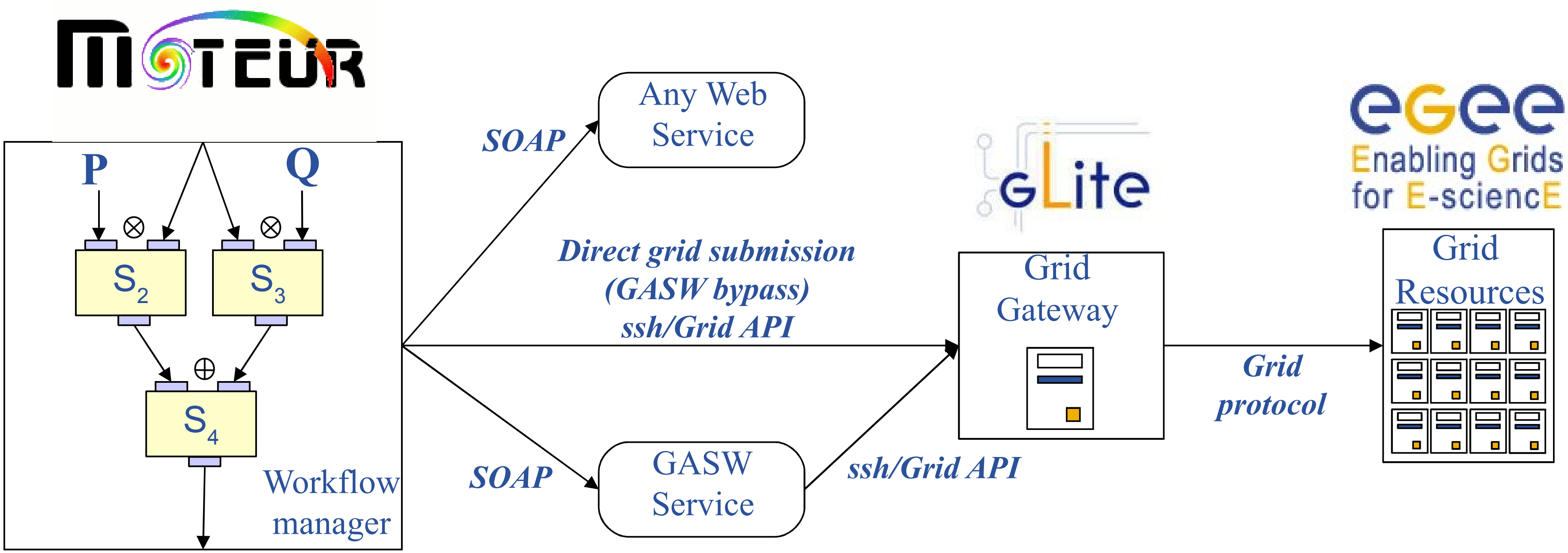
Parallel execution

- **A workflow naturally provides application parallelization**
- **3 kinds of parallelism exploited**



- **Data sets are composed dynamically**

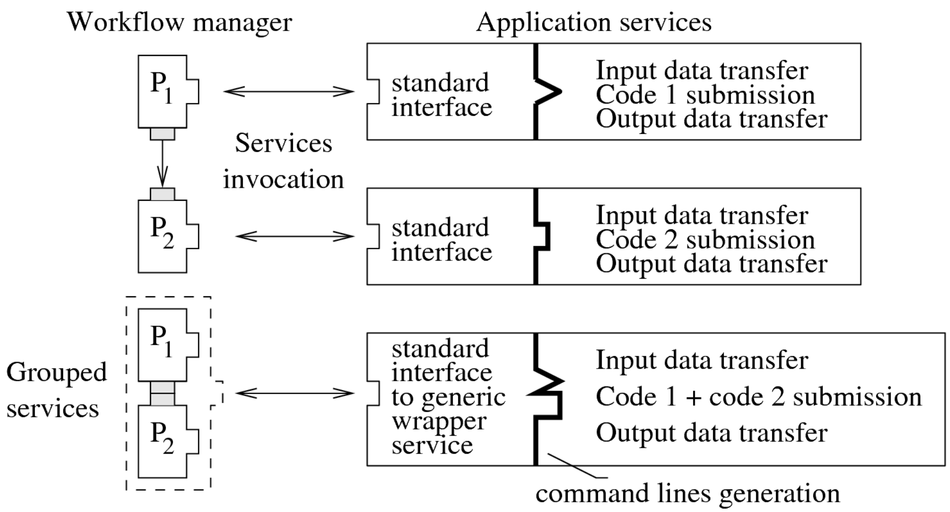
Cheating with SOA



- **GASW service is recognized by MOTEUR**
  - Not a real blackbox
- **Submission can be directly made to the grid**
  - Grid mode: Web Service call bypassed
  - No difference from a user point of view (no change in Scufl)

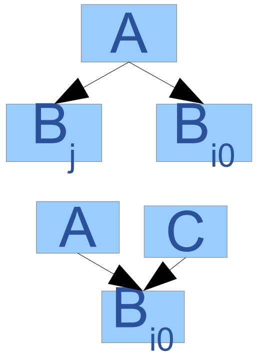
Optimizing the execution through dynamic code wrapping

- **High latencies strongly penalizes job execution**
  - Latency ≈ 5 minutes +/- 5 minutes on the EGEE production grid
- **High variability requires to reduce the number of jobs**
  - The higher the number of submitted jobs, the higher the probability to get outliers
- **Services grouping reduces the number of job submissions**



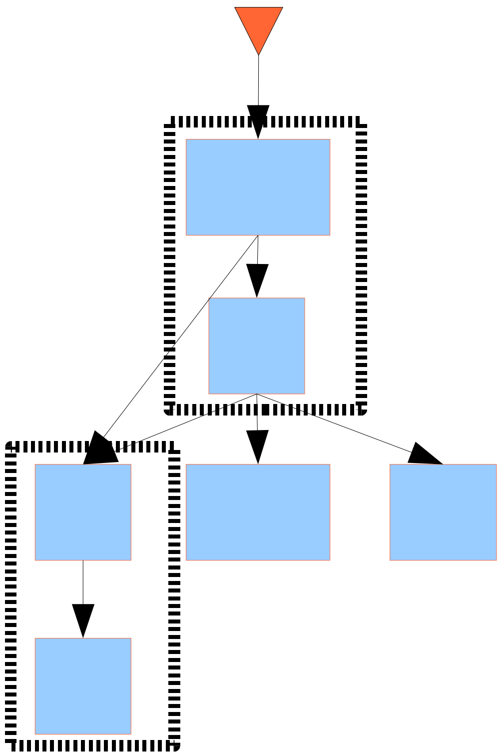
Grouping rule

- **Goal: find a grouping rule that does not break parallelism**
- **Let A be a service of the workflow and {B0,...Bn} its children**
- **For grouping A and Bi0: no parallelism loss <=>**
  - (1) Bi0 is an ancestor of every Bj
  - (2) Every ancestor of Bi0 is an ancestor of A (or A itself)
- **No parallelism loss =>**
  - ¬(1) => parallelism between Bj and Bi0 is broken
  - ¬(2) => parallelism between A and C is broken
- **(1) & (2) => no parallelism loss**
- **This rule is recursively applied on the workflow graph**

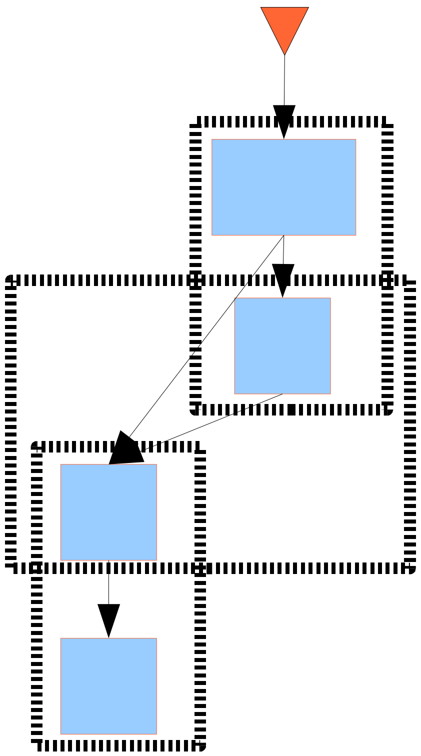


Job Grouping Experiments

- 6 services – 2 groups
- 4 job submissions/input data set submission/input data set

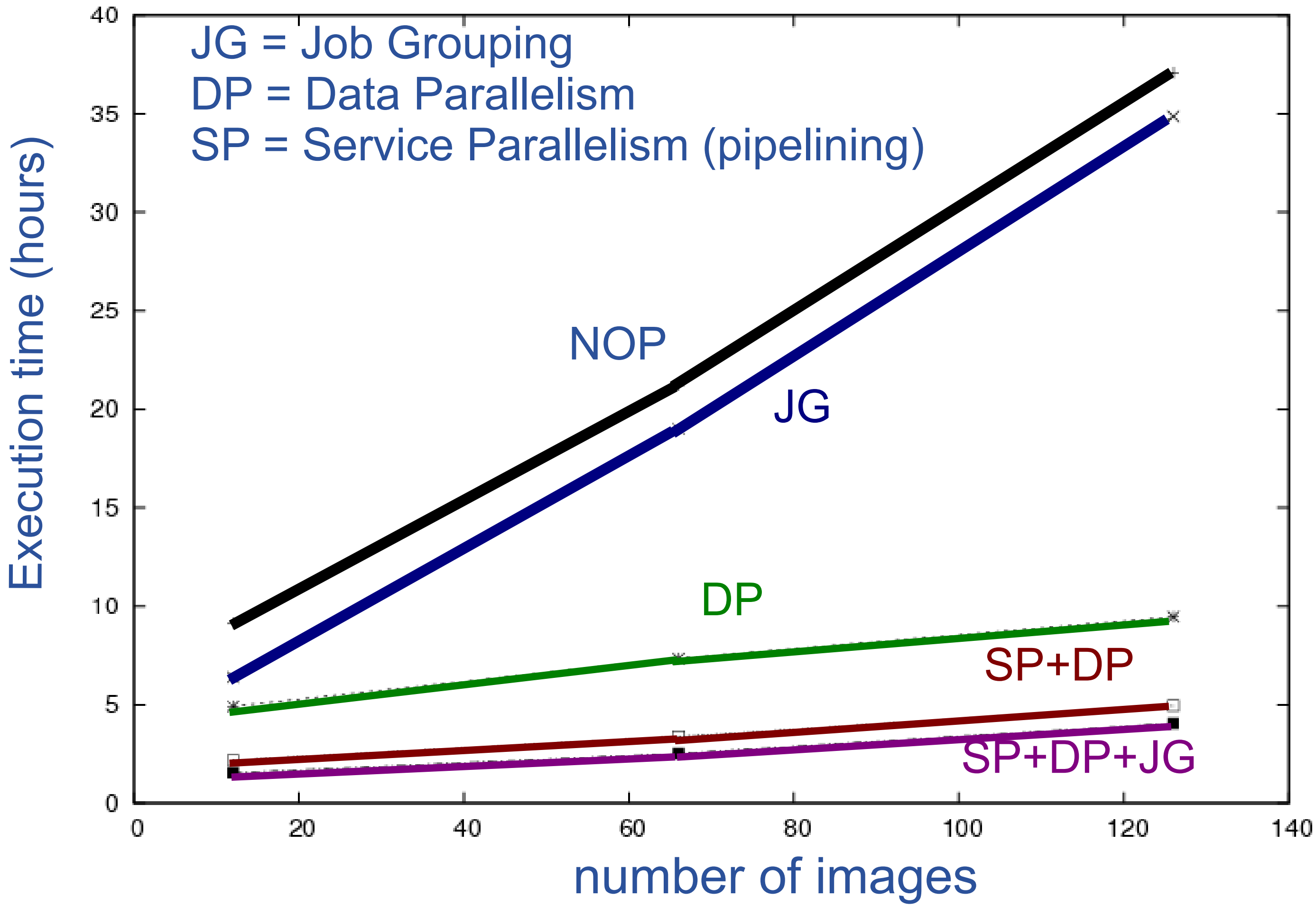


- 4 services - 3 groups
- Recursive application



Performance results

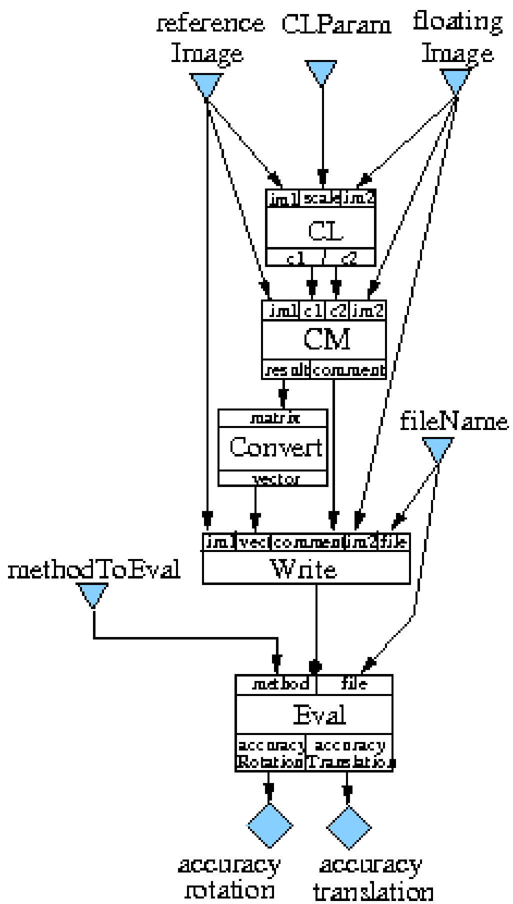
- **On the EGEE infrastructure**

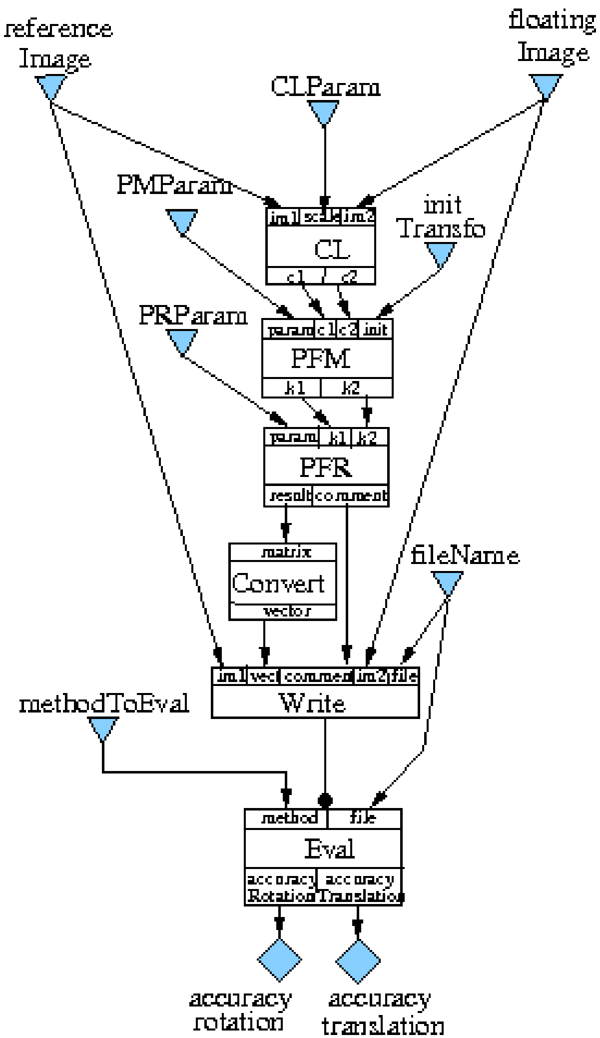


Workflows transformation

Grid Workflow Efficient Enactment for Data Intensive Applications

- **Two basic registration orchestrations to be merged**

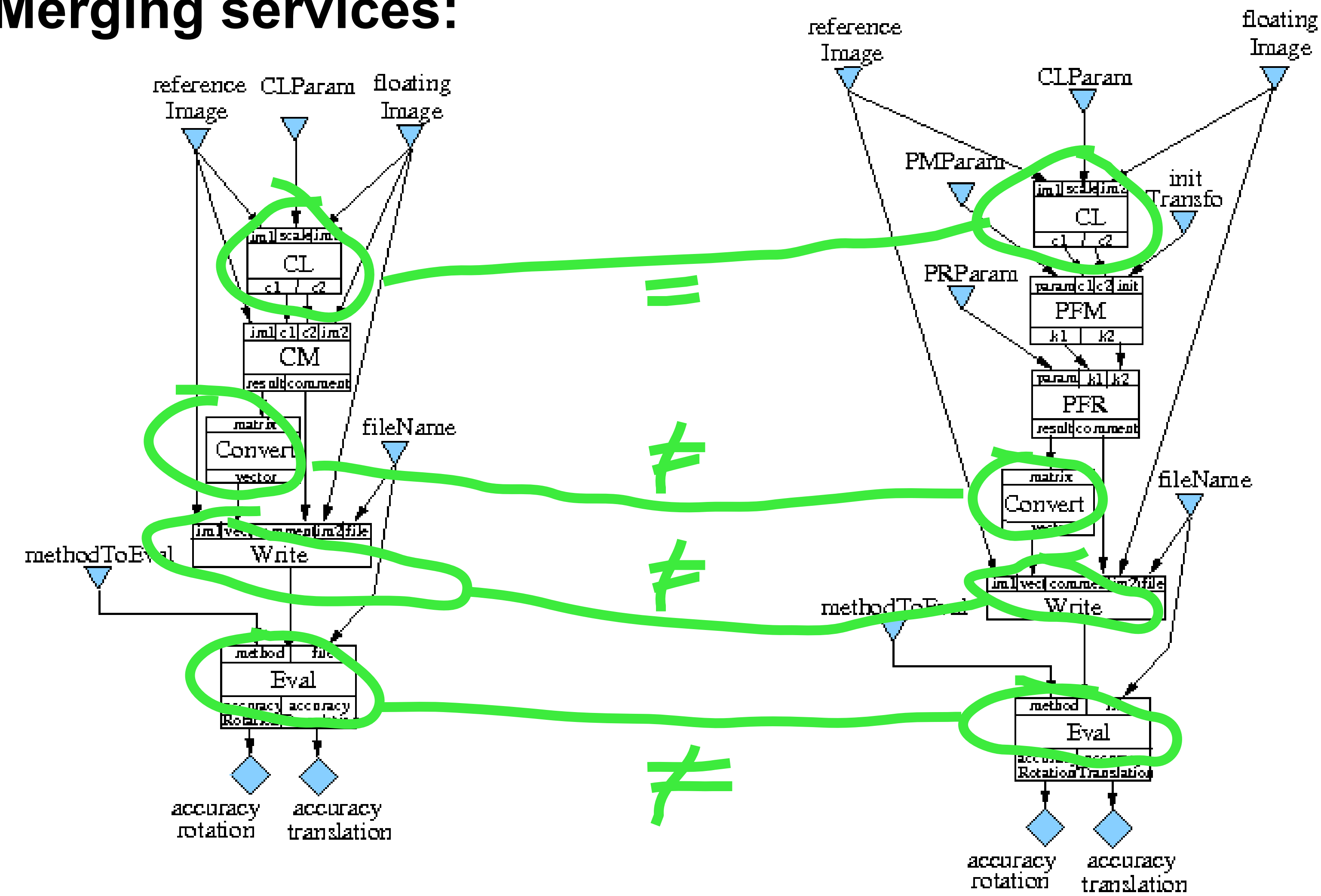




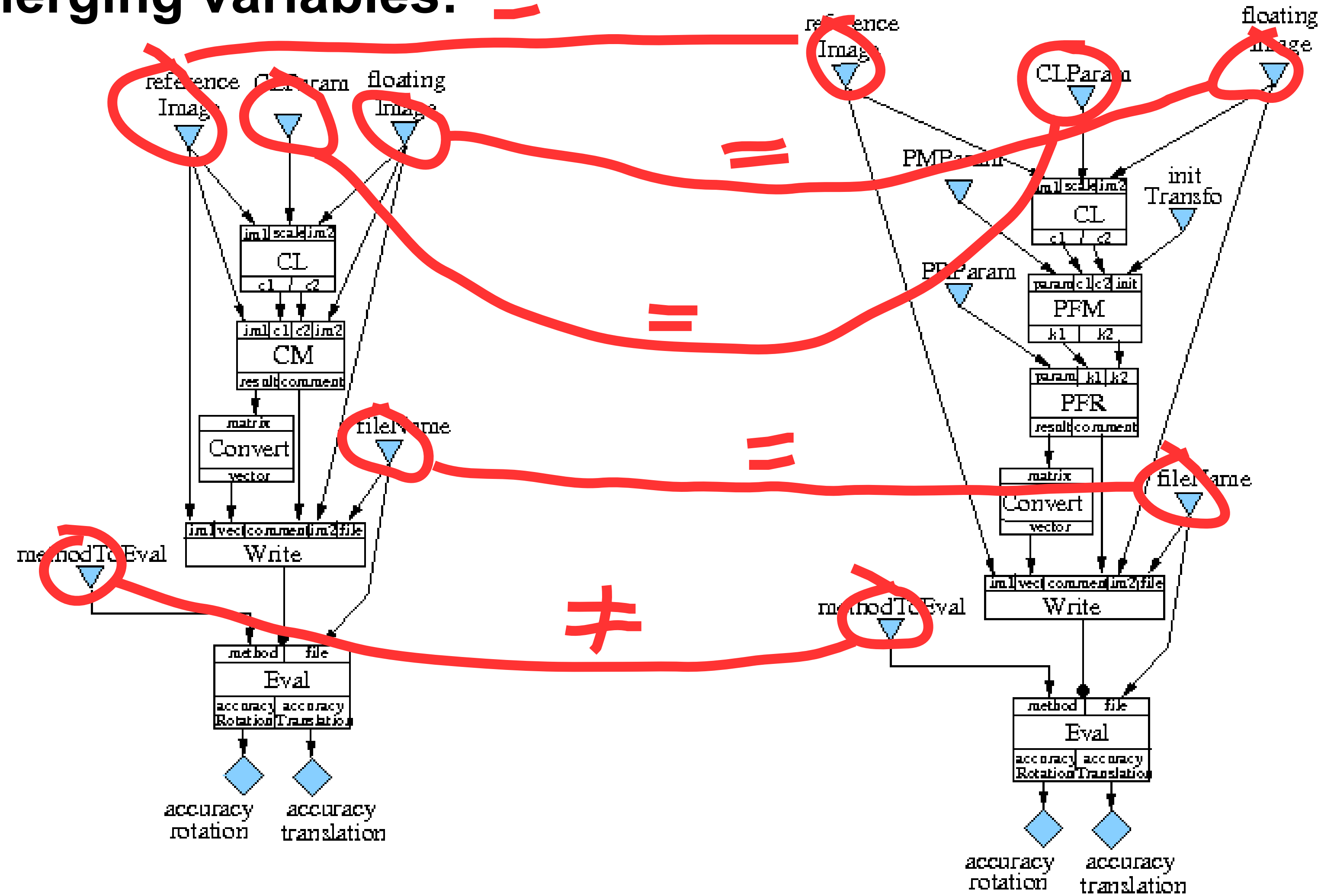
Workflows transformation

Grid Workflow Efficient Enactment for Data Intensive Applications

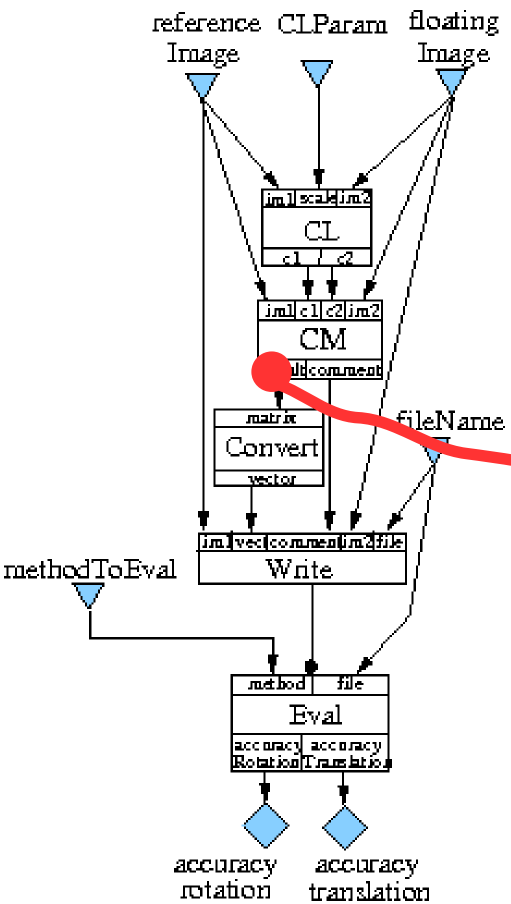
- **Merging services:**

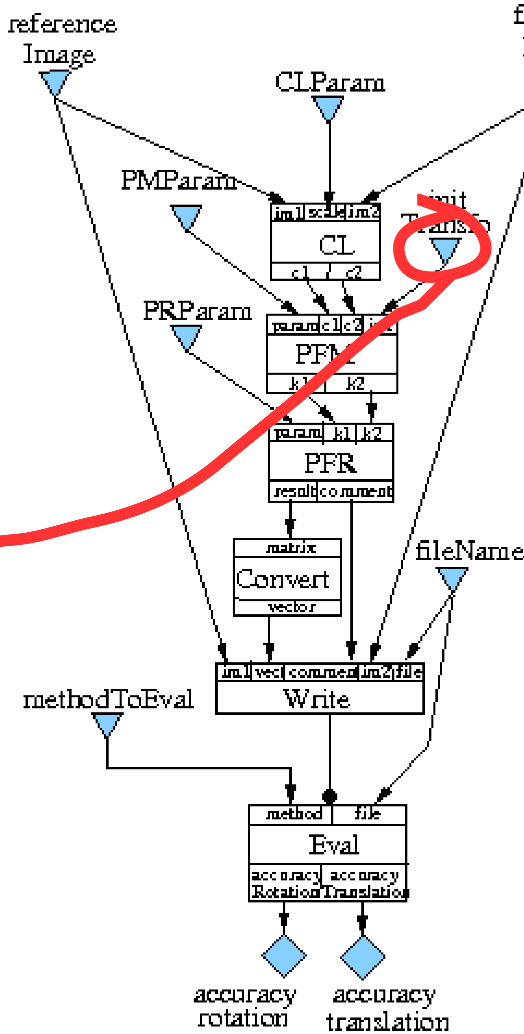


- **Merging variables:**



- **Merging variables (II):**

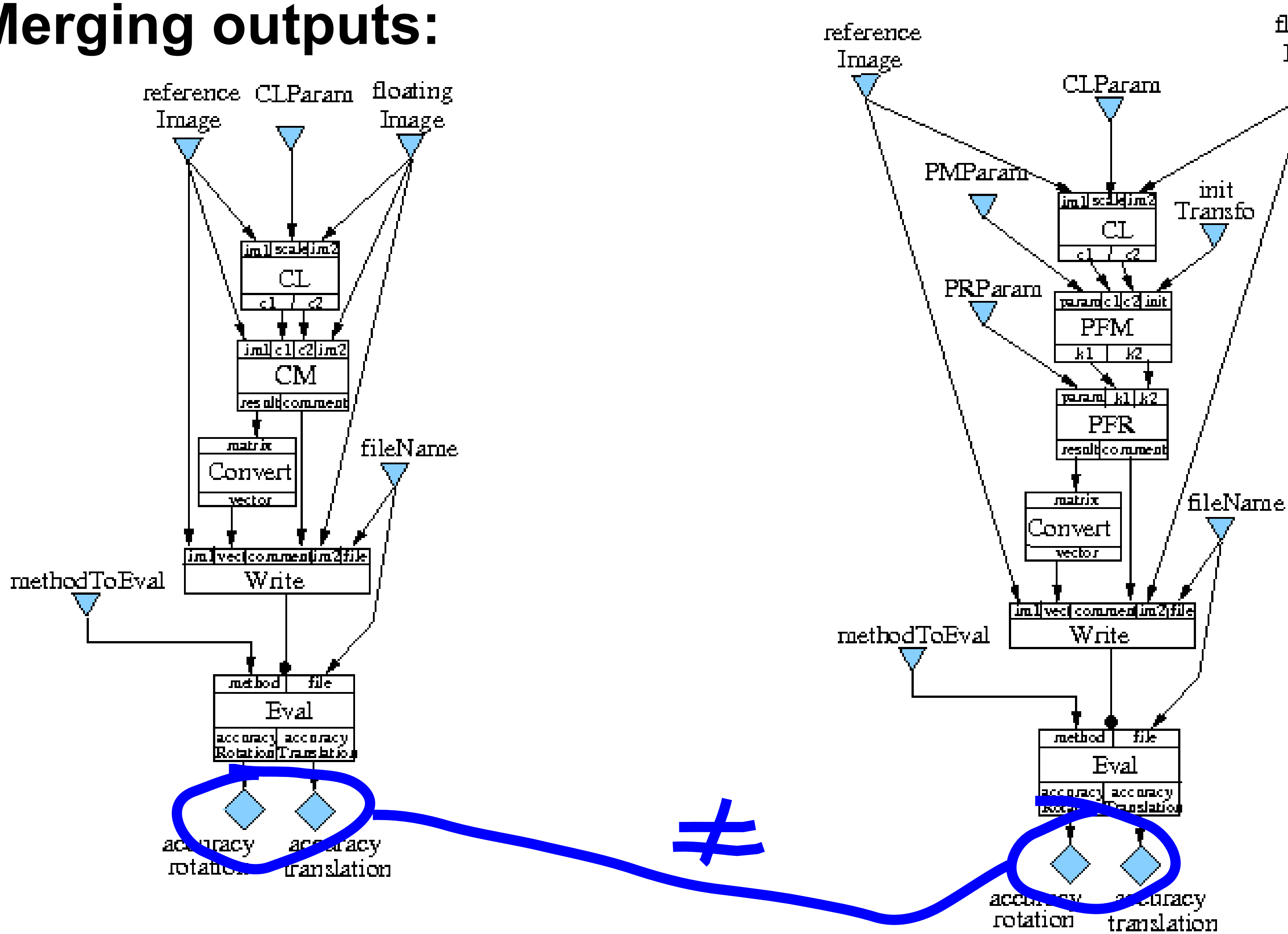




Workflows transformation

Grid Workflow Efficient Enactment for Data Intensive Applications

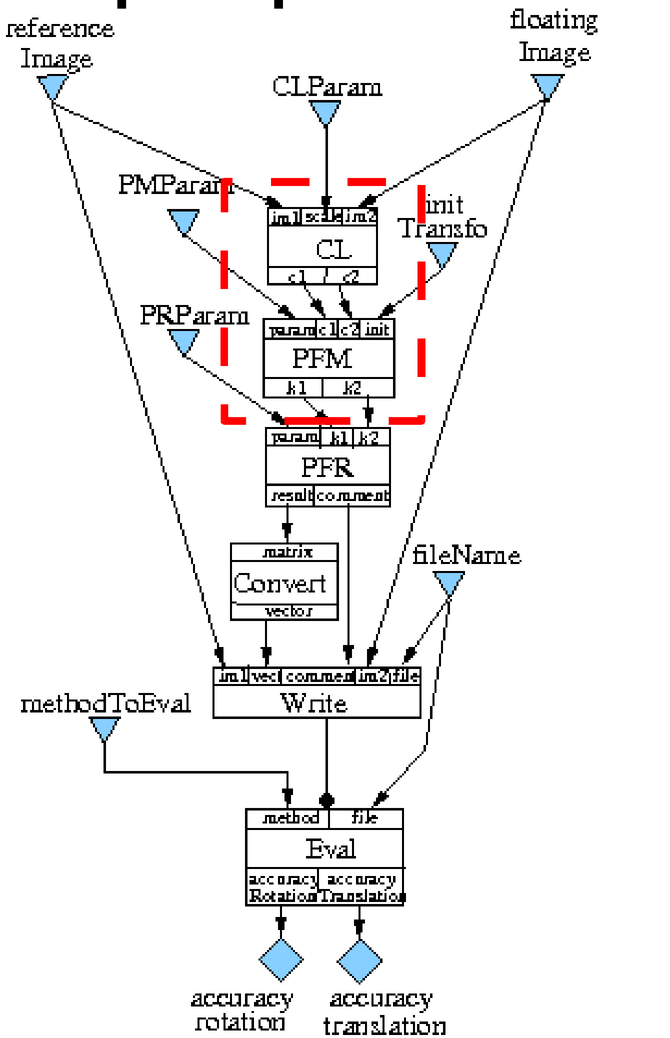
- **Merging outputs:**



- **A fully automatic merging procedure is not suitable**

Orchestration Model Supporting Merging

- **Graph representation**



- **OMSM representation in Prolog [C. Nemo et al, SCC'07]**

%Orchestration declaration
orch(pfm,[im0,im1,fN,method,transfo,cl,pm,pr],
[i1,i2,i3,i4,...]).

% Instructions
instr(i1,[im0,im1,cl],clOut,invoke(cl)).
instr(i2,[clOut],c1,assign(pfmC1)).
instr(i3,[clOut],c2,assign(pfmC2)).
instr(i4,
[pm,pfmC1,pfmC2,transfo],pfmOut,invoke(pfm)).
...

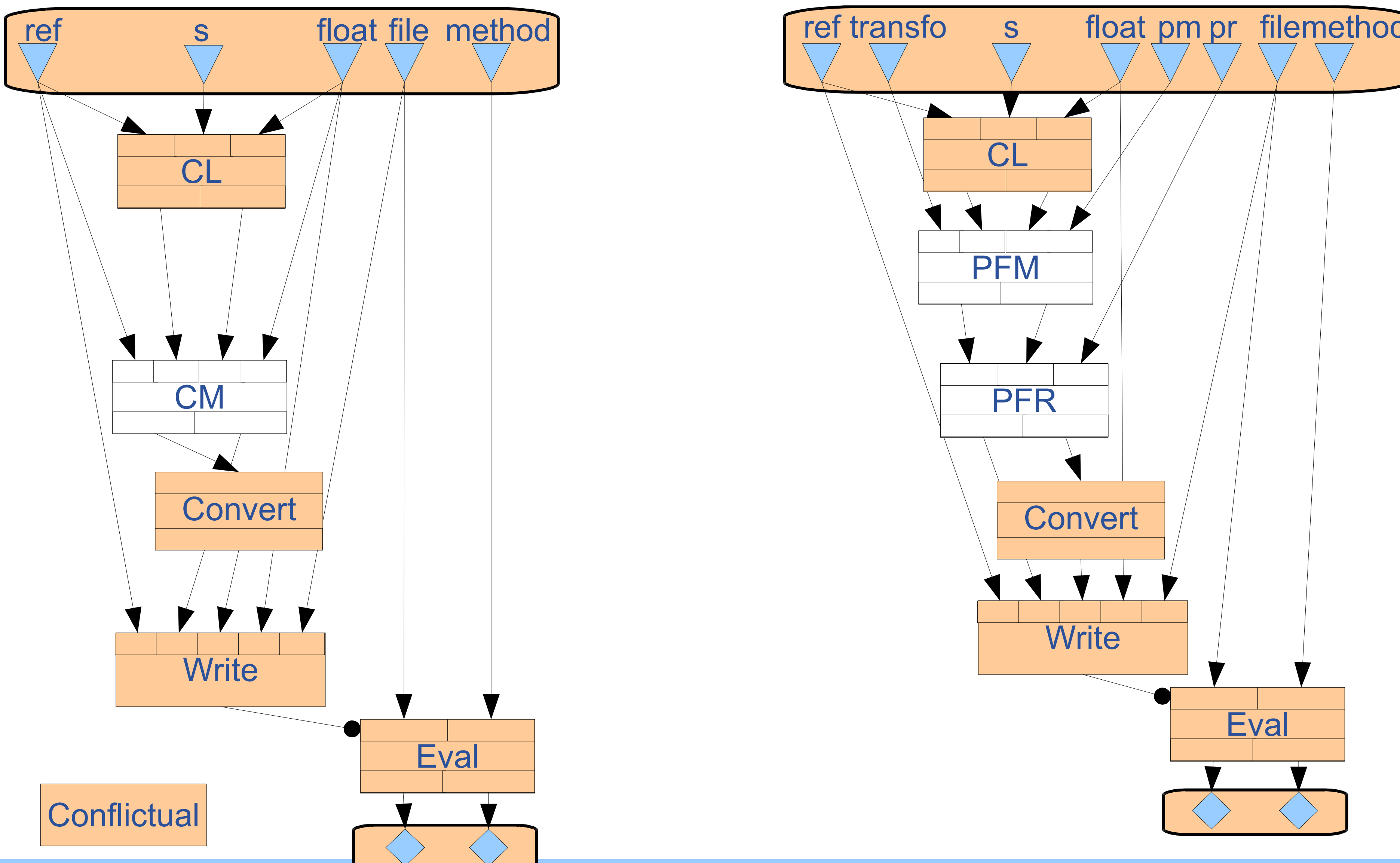
% Constraints
pred(i1,i2).
pred(i1, i3).
pred(i2, i4).
pred(i3, i4).
...

Conflicts detection

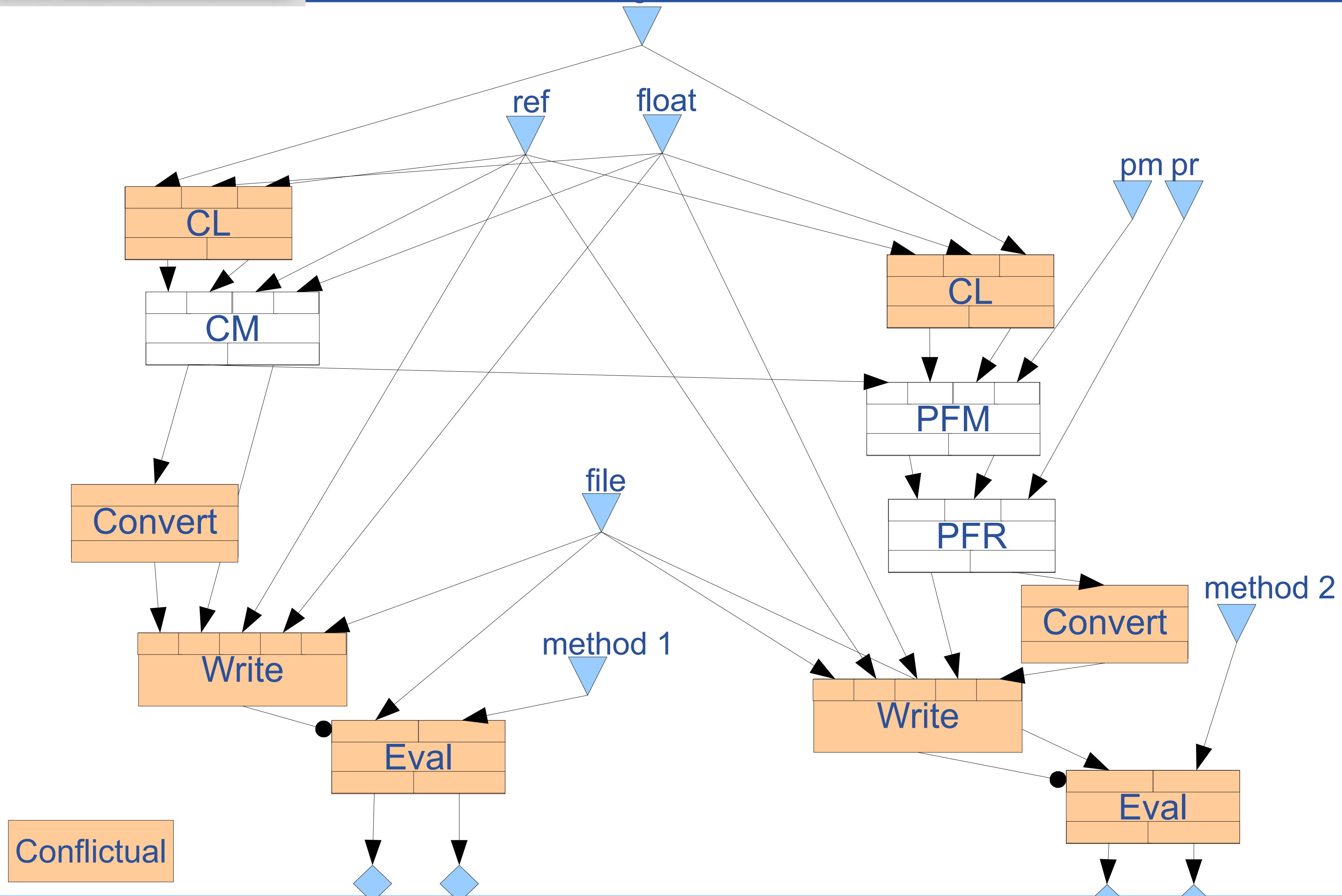
- **Consistency properties:**
  - P1: orchestrations have at most one invocation to a given basic service
    - Multiple invocations are wrapped in a complex invocation
  - P2: orchestrations have at most one input and one output
    - Multiple input / outputs are encapsulated in a structure
  - P3: orchestrations satisfy a set of constraints:
    - No concurrent write access to the variables
    - No cycle in instructions invocation order
- **Potential unification points are detected because they break either P1 or P2**
- **If an orchestration breaks P3, it is rejected from the process**

Conflicts detected

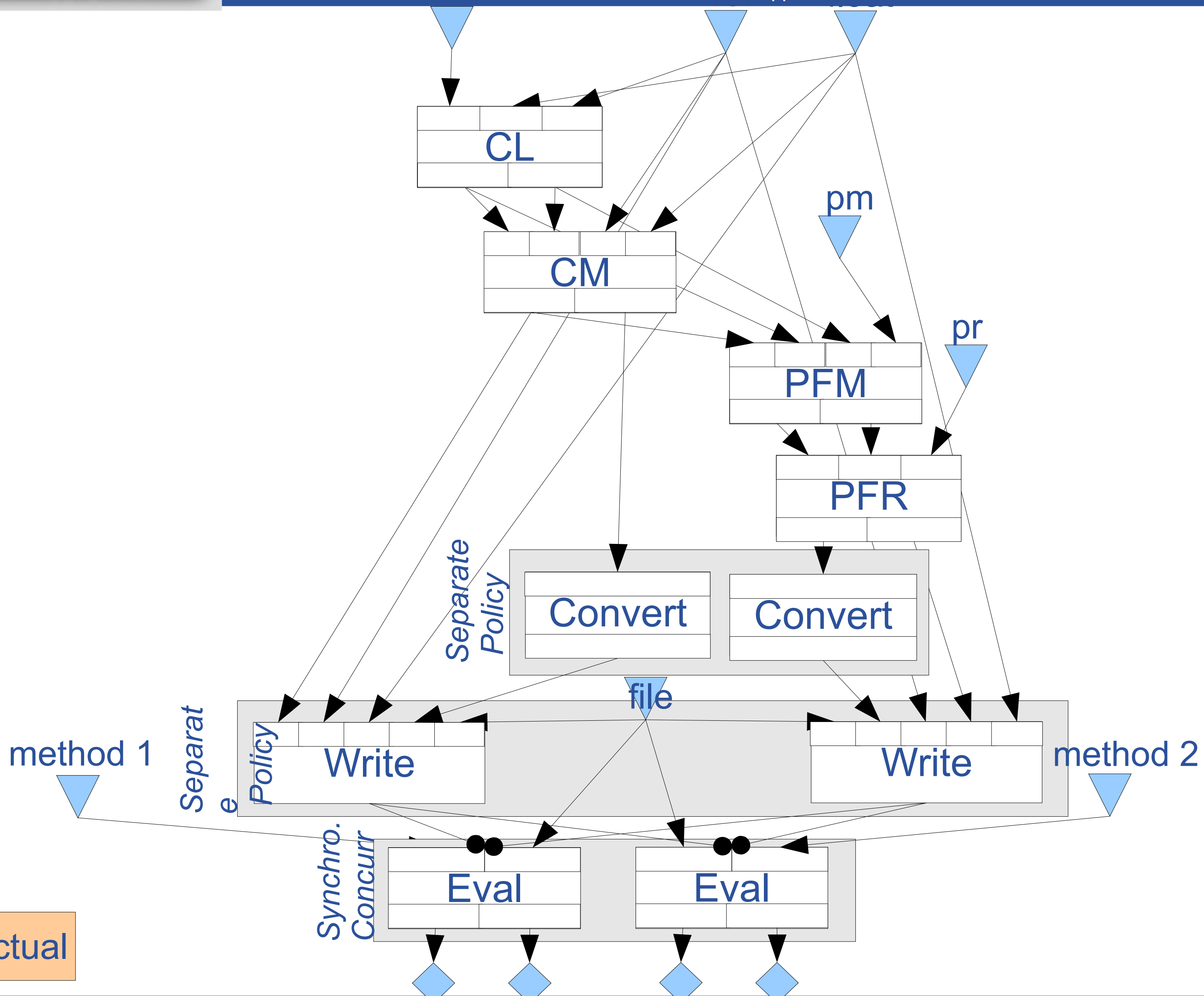
Grid Workflow Efficient Enactment for Data Intensive Applications



Inputs / outputs merging



Services merging



Conflictual

Conclusions

- **Scientific workflows**
  - Implicit parallelism
  - Complex data flows
- **Software composition**
  - Code reuse
  - Heterogeneous codes assembling
- **Preoccupations**
  - Performance
  - Ease of use
  - Separate application logic and data sets
  - Compact representation of complex processes