Monitoring and Performance Analysis in Distributed Systems

Roland Wismüller

LRR-TUM
Institut für Informatik, Technische Universtität München
Germany

e-mail: wismuell@in.tum.de

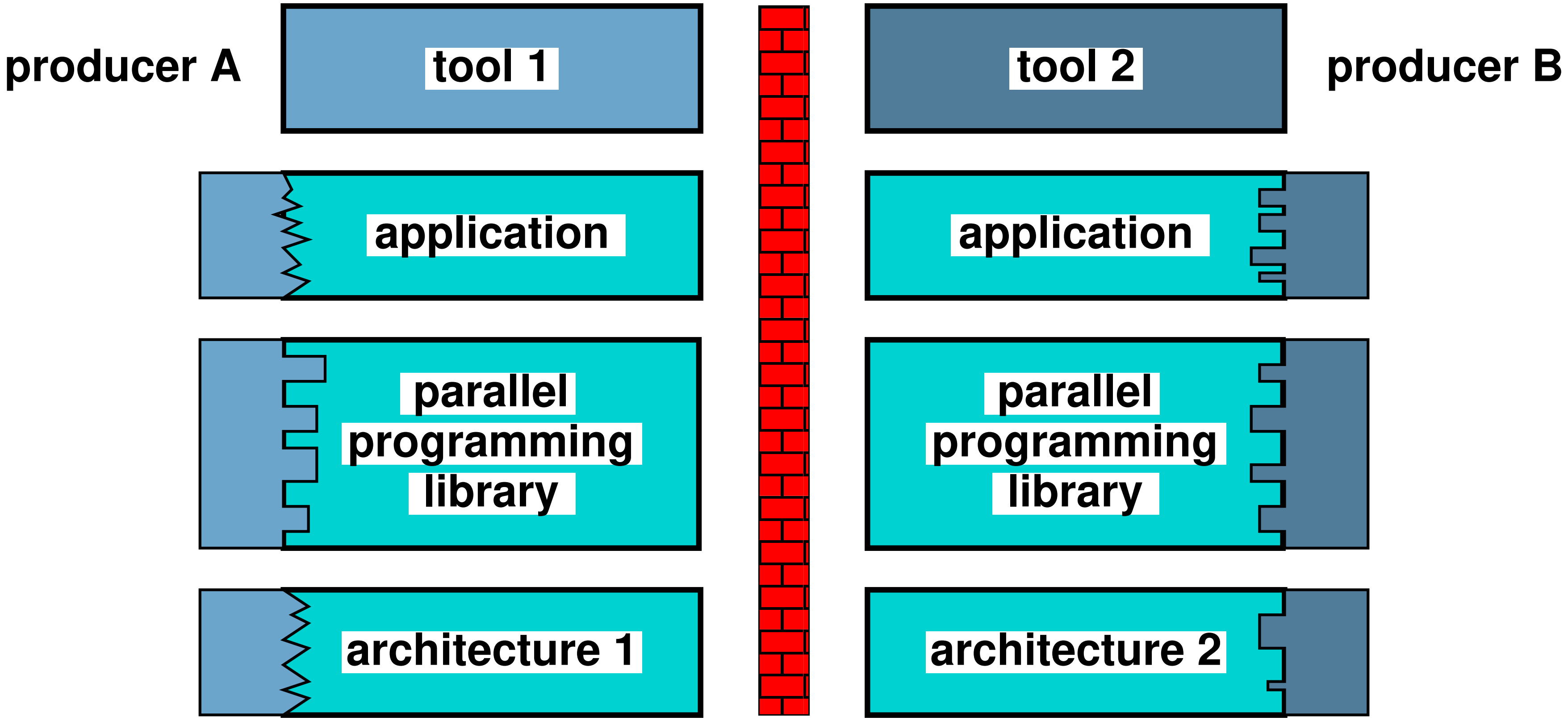
Agenda

- Monitoring of distributed systems
  - OMIS – On-line Monitoring Interface Specification
  - OCM – An Implementation of OMIS
- Distributed automatic performance analysis
  - Concepts from the APART working group
  - A possible implementation on the Grid

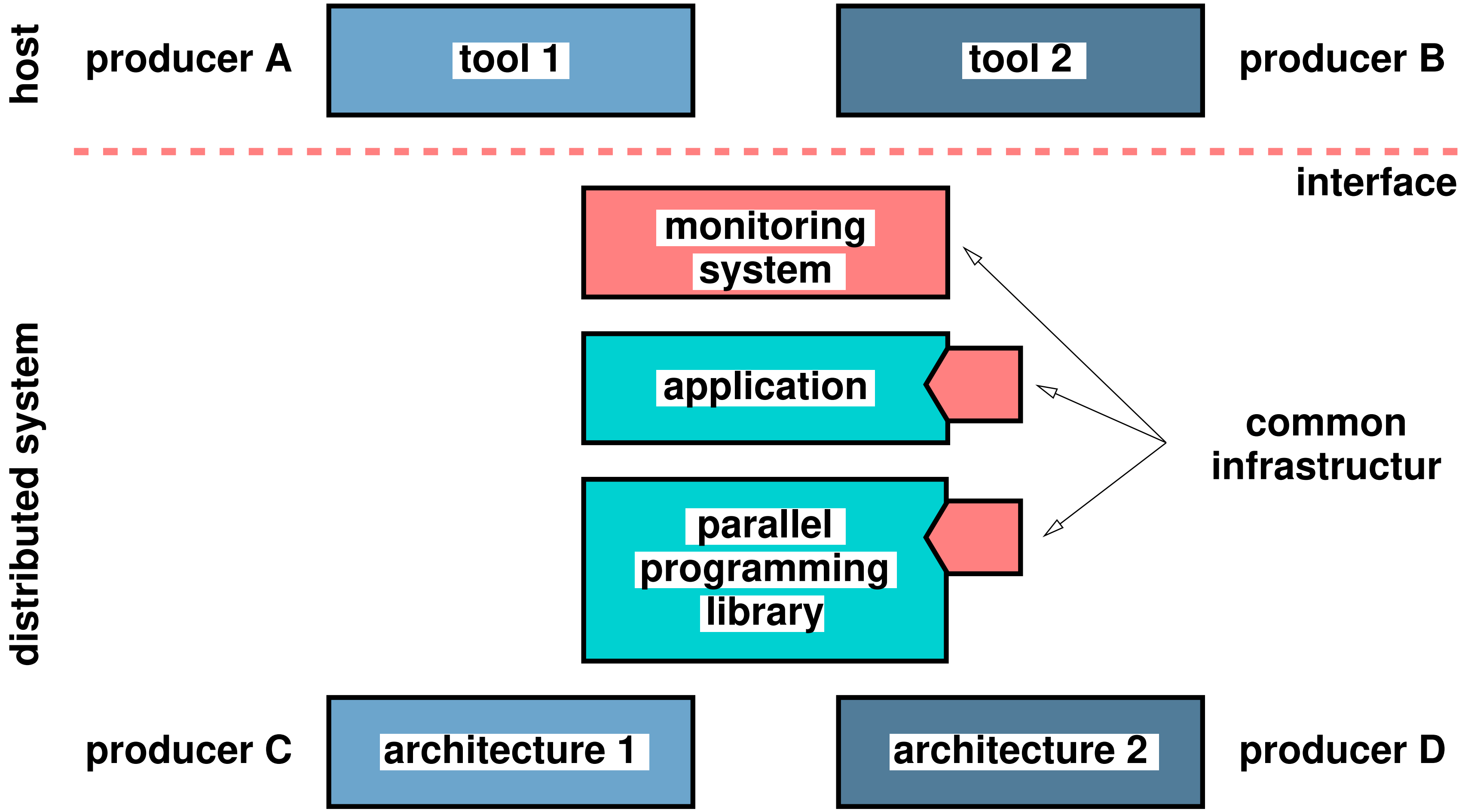
Motivation

- On-line tools need an infrastructure for **observation** and **manipulation** of program executions
  - e.g. debugging, performance analysis, load balancing, resource management, adaptive run-time systems, ...
- Currently:
  - proprietary monitoring systems
  - no standard infrastructure exists
- Consequences:
  - new tool $\Rightarrow$ new monitoring system
  - limited portability, no interoperation

Current Tool Design



Tool Design using OMIS



On-line Monitoring Interface Specification

The interface should ...

- support on-line monitoring
  - adjust at run-time what should be monitored
  - modify target's run-time behavior if desired
- be platform independent
  - platform $=$ hardware, operating system, progr. library
  - supports development of uniform tool environments
- be usable for any kind of run-time tool
  - observing and/or manipulating tools
  - interactive / automatic, centralized / distributed

On-line Monitoring Interface Specification

The interface should ...

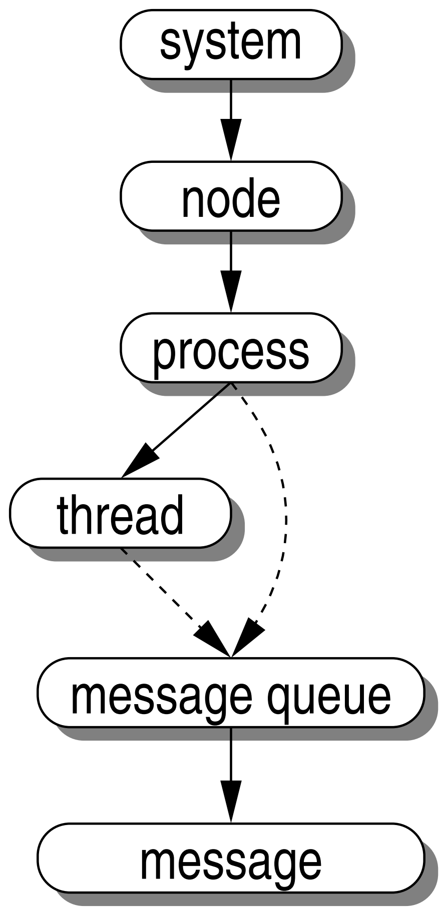
- support interoperable tools
  - sets of independent, small tools that can be used together
  - mechanisms for coordination between tools
- be efficient
  - minimal intrusion (CPU usage, communication)
  - scalability

☞ Meeting all of these requirements at the same time is hard

OMIS: Basic Concepts

Object based approach

- observed system is a hierarchical set of objects
  - classes: nodes, processes, threads, messages, and message queues
  - node/process model suitable for DMPs, NOWs, SMPs, and SMP clusters
- access via abstract identifiers (tokens)
- services observe and manipulate objects
  - OMIS core: platform independent serv.
  - others: platform specific extensions
- tools define their own view of the observed system



OMIS: Basic Concepts

Event / action paradigm

- monitoring requests consist of two parts
  - event definition specifies when something has to be done
  - action list specifies what has to be done
  - empty event definition $\Rightarrow$ execute action list immediately
  - ☞ tools differ mainly in the required event/action relations
  - ☞ events needed by different tools are fairly the same
  - ☞ actions may be more diverse $\Rightarrow$ extensions
- actions are autonomously executed by the monitoring system
  - ☞ monitoring system can be ‘programmed’ by a tool
- event context parameters: pass event information to actions

OMIS: Basic Concepts

Scalability

- services operate on sets of objects
- token expansion
  - object token $\Rightarrow$ set of all contained objects

Location transparency

- necessary for services on object sets
- token localization:
  - object token $\Rightarrow$ containing object

OMIS: Basic Concepts

Extensibility

- motivation:
  - tools may require special actions and/or events
  - execute parts of a tool in a distributed way
  - platform specific services and/or objects
- distributed tool extensions
  - independent of target system and OMIS implementation
  - OMIS defines both external and internal interface
- monitor extensions
  - may depend on target system and OMIS impl.
  - OMIS only defines the external interface

OMIS: Basic Concepts

OMIS Request Examples

```
thread_reached_addr([p_14,p_52], 0x1234):
     thread_stop([]) thread_get_backtrace($thread, 0)
```

- When (a thread in) one of the specified processes reaches the breakpoint, stop the whole application and return the stack backtrace

```
thread_has_started_lib_call([p_21], "pvm_send"):
     pa_counter_local_increment(pa_lc_1, $par5)
```

- When (a thread in) process p_21 starts a send call, add the message size (event context paramerter par5) to a counter

A More Complex Example

```
thread_has_ended_lib_call([a_], "pvm_recv") :
        pa_counter_global_increment(pa_c_1, 1)

thread_creates_proc([a_]) :
        node_attach([$new_node])
        proc_attach([$new_proc])

:   pa_counter_global_read([pa_c_1], 1)
```

➠ Count the total number of `pcm_recv` calls in all processes

A Still More Complex Example

```
:   pa_ac_create()

:   array_insert_index(p_1,pa_ac_1)

thread_has_ended_lib_call([a_], "pvm_recv") :
       pa_counter_local_increment(pa_ac_1<$proc>, 1)

thread_creates_proc([a_]) :
       node_attach([$new_node])
       proc_attach([$new_proc])
       aray_insert_index($new_proc, pa_ac_1)

:   pa_counter_local_read([pa_ac_1], 1)
```

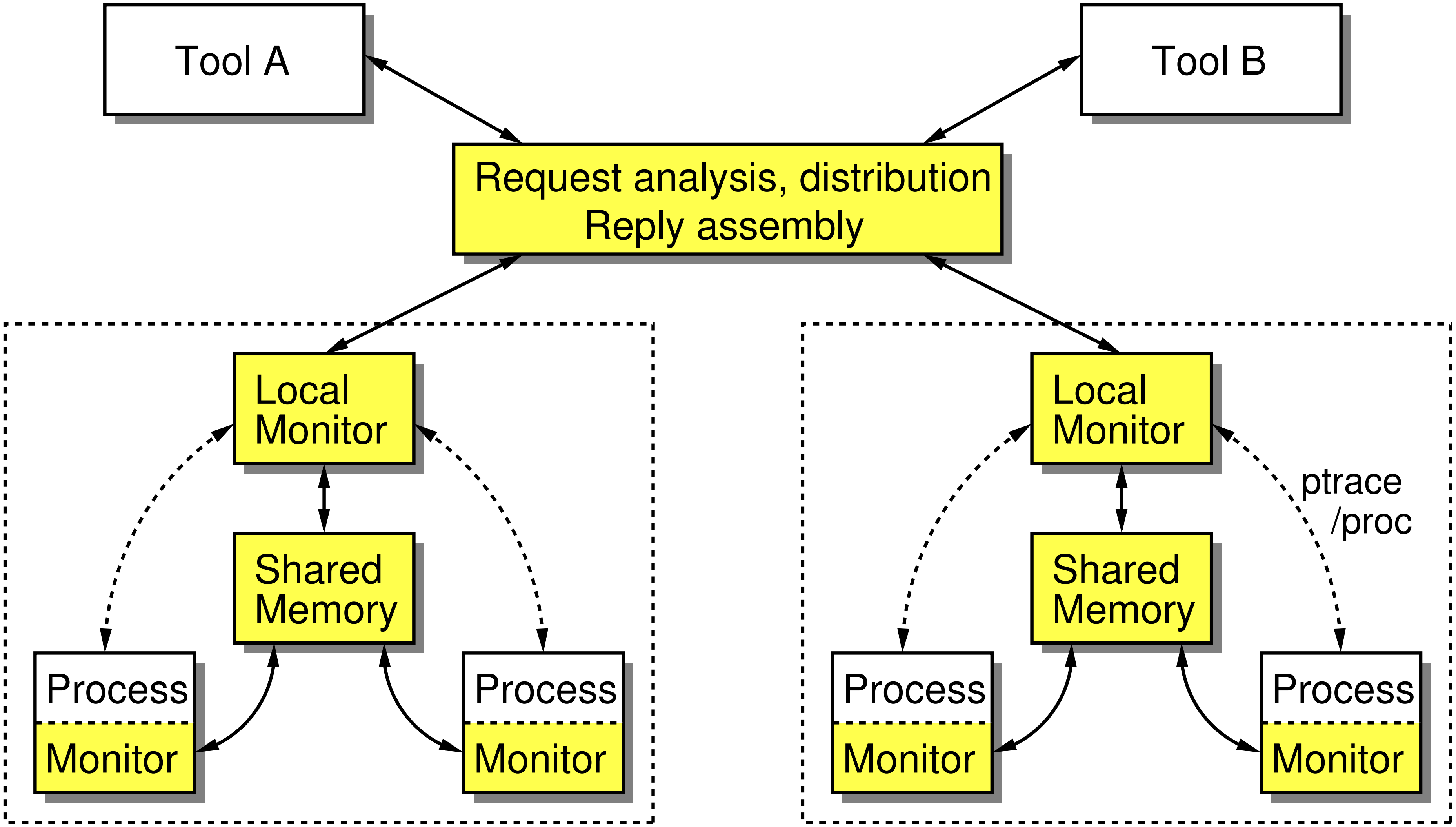
- Count the number of `pcm_recv` calls for each process

OCM – An Implementation of OMIS

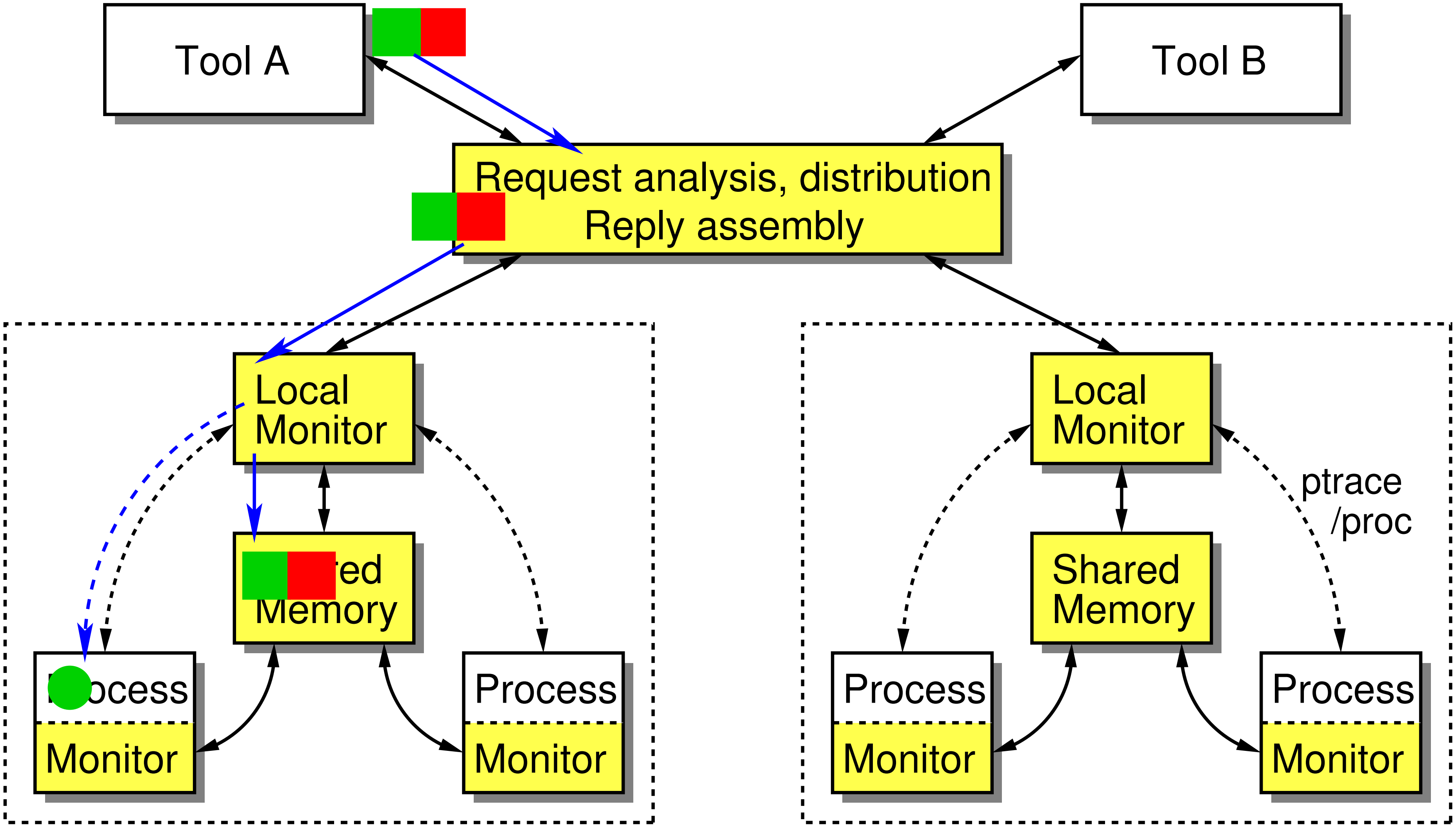
OCM = OMIS compliant monitoring system

- reference implementation of OMIS
- supports PVM and MPI on NOWs (Solaris, Irix, Linux, ...)
- first tools are available (debugger, visualizer, performance analyzer)
- ports to other platforms:
  - Java
  - Grid (see Marian Bubak's talk)
  - SCI-Cluster (SMiLE)

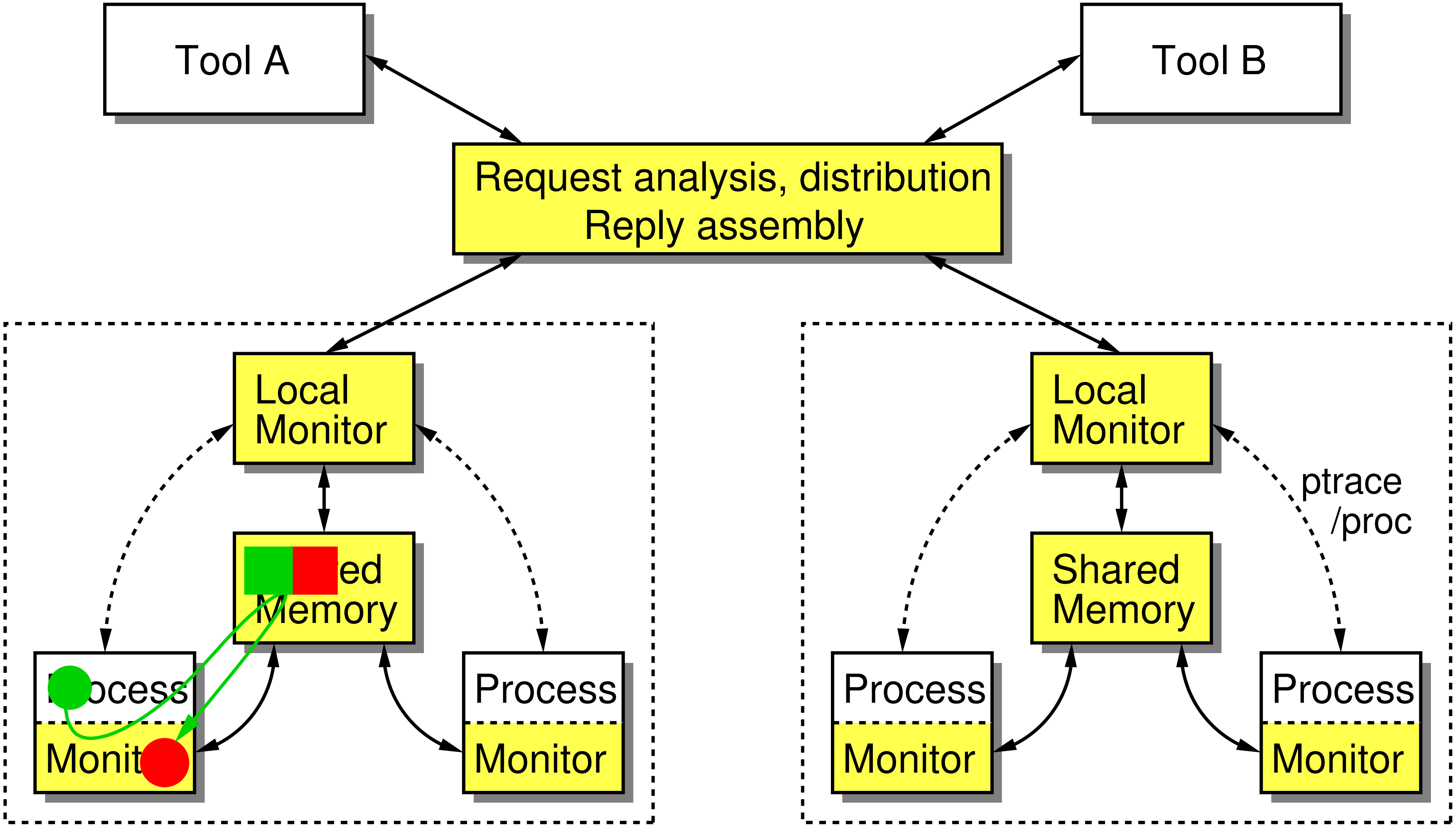
Structure of the OCM



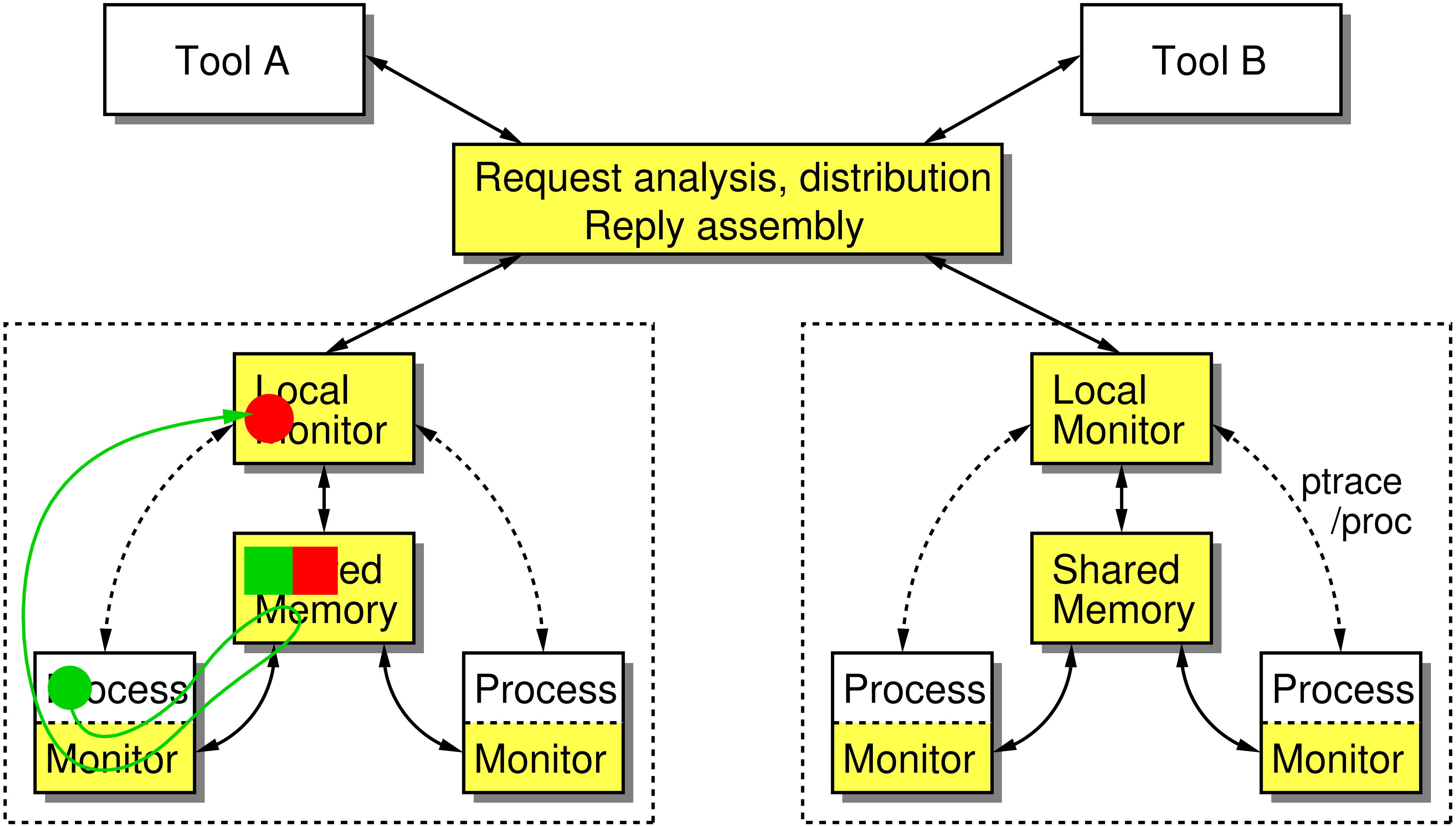
Request Processing in the OCM



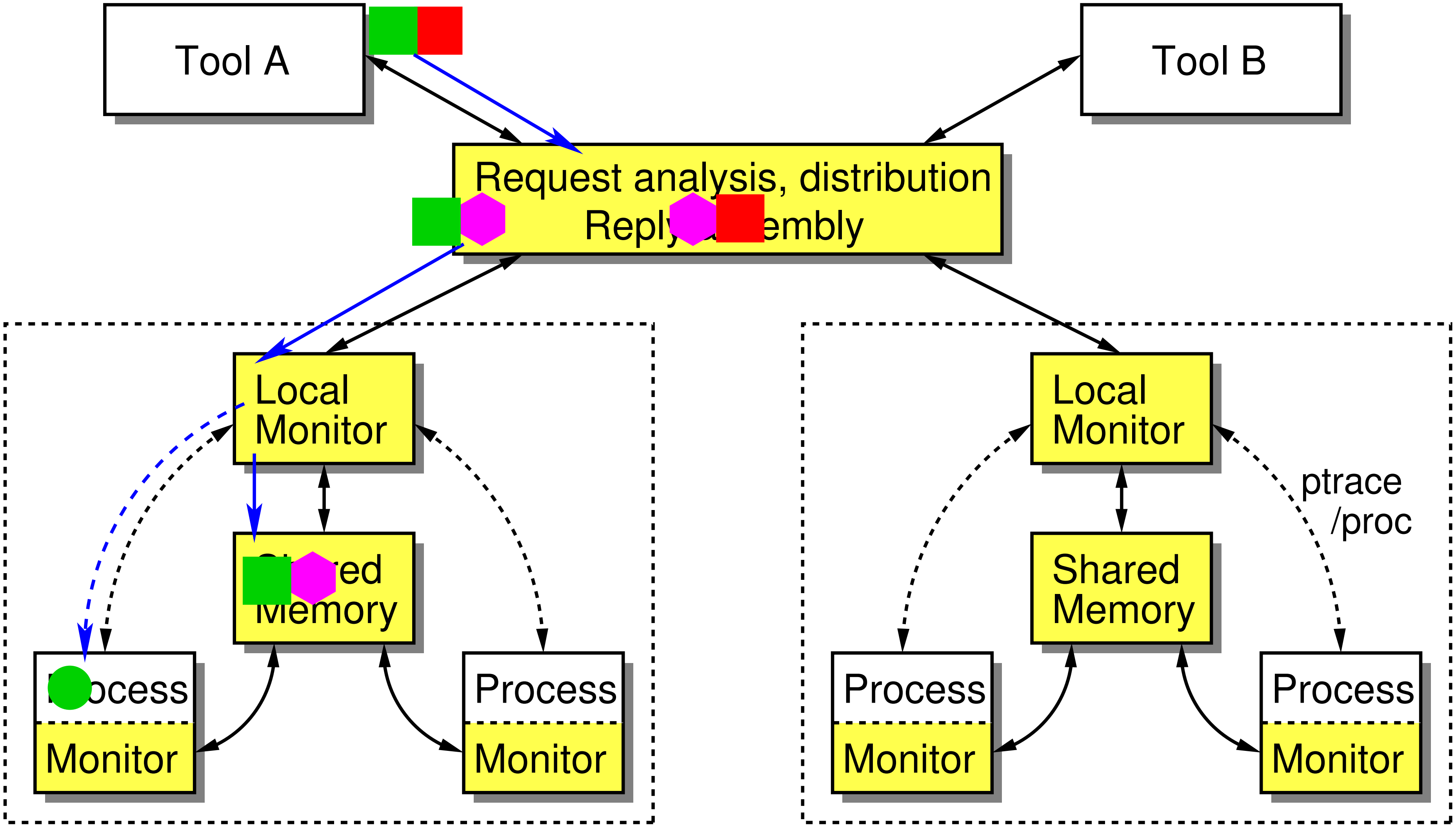
Request Processing in the OCM



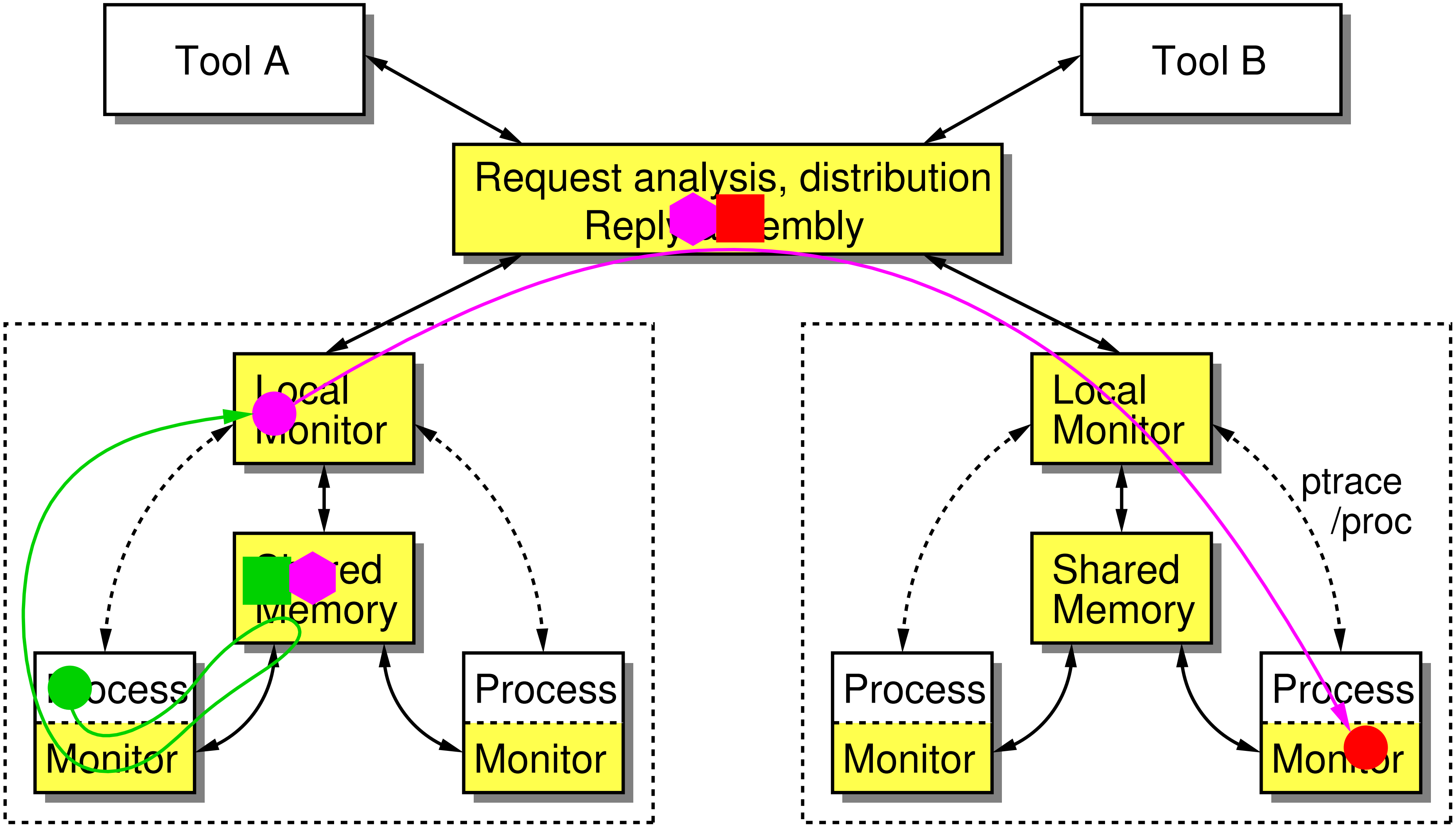
Request Processing in the OCM



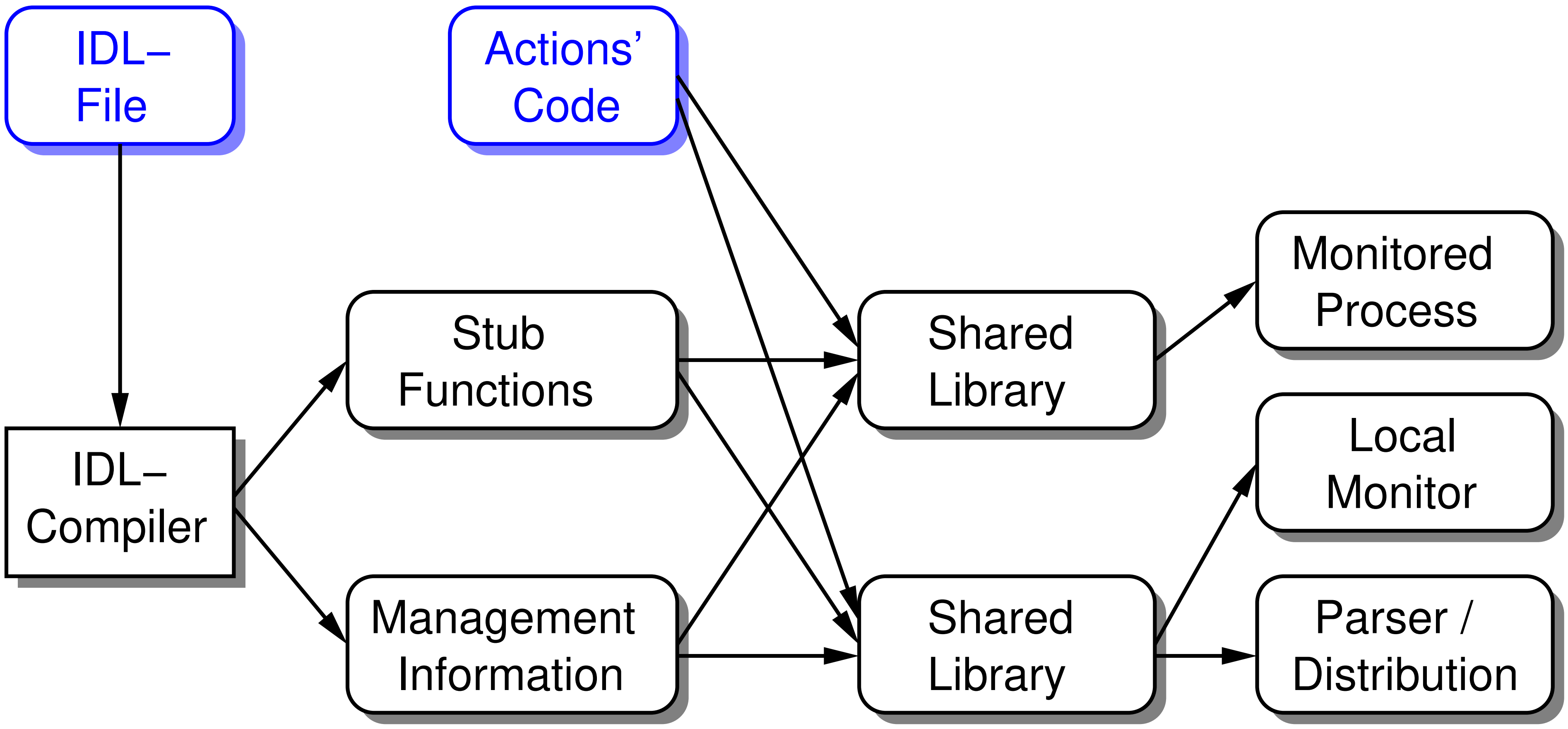
Request Processing in the OCM



Request Processing in the OCM



Extending the OCM: new Actions



OMIS/OCM for the Grid

Why OMIS/OCM for the Grid?

- on-line monitoring
  - optimization, resource management, steering
- uniform interface on all platforms
  - tools for heterogeneous systems
- distributed request processing
  - efficiency in distributed systems
- support for interoperability
  - collaborative use of tools ?

Why Automatic Performance Analysis for the Grid?

- user friendliness
- users do not have/need knowledge about execution platforms
- tracing and off-line analysis is prohibitively expensive
- programs can not be rerun in the same environment
  - no repeated tracing / profiling
- batch-oriented usage of the Grid
  - interactive tools are difficult to apply
- execution environment not known a-priori
  - no a-priori optimization
  - on-line tuning? (e.g. optimal number of workers)

Concepts from APART

The APART Working Group

- ESPRIT working group (1999-2000, 2001-2004)
- coordinates projects on automatic performance analysis:
  - Aurora (Univ. Vienna)
  - Peridot (Techn. Univ. Munich)
  - OVALTINE/ADAPT (Univ. Manchester)
  - KOJAK (FZJ)
  - Kappa-Pi (Univ. Autònoma de Barcelona)
  - [Paradyn (Univ. Madison)]

Concepts from APART

The APART Approach

- object oriented performance data model
  - available performance data
  - different kinds and sources, e.g. profiles, traces, ...
  - make use of existing monitoring tools
- formal specification of performance properties
  - possible bottlenecks in an application
  - specific to programming paradigm
  - APART specification language (ASL)
- (specification of analysis process)

Concepts from APART

Property Specification in ASL

```
PROPERTY TEMPLATE CostPerProcess <float CostFunc(MPISummary)>
    (Region r, Experiment e, Process p, Region RankBasis)
{
    LET  cost = CostFunc(summary(r,e,p))
    IN   CONDITION: cost > 0;
         CONFIDENCE: 1;
         SEVERITY: cost/duration(RankBasis,e);
}

float IoCostFunc(MPISummary rs) = rs.IoTime;
float CommCostFunc(MPISummary rs) = rs.CommTime;

PROPERTY CostPerProcess <IoCostFunc> IoCostPerProcess;
PROPERTY CostPerProcess <CommCostFunc> CommCostPerProcess;
```

Concepts from APART

Property Specification in ASL

```
PROPERTY TEMPLATE PropertyOnAllProcesses
    <PROPERTY x(Region, Experiment, Process, Region)>
    (Region r, Experiment e, Region RankBasis)
{
    CONDITION:  Forall p IN e.processors SUCH THAT
                   condition(x(r,e,p,RankBasis)) AND
                   confidence(x(r,e,p,RankBasis)) ==1;
    CONFIDENCE: 1;
    SEVERITY:   max(severity(x(r,e,p,RankBasis))
                    WHERE p in e.processes)
}

PROPERTY PropertyOnAllProcesses <IoCostPerProcess> IoCost;
PROPERTY PropertyOnAllProcesses <CommCostPerProcess> CommCost;
```

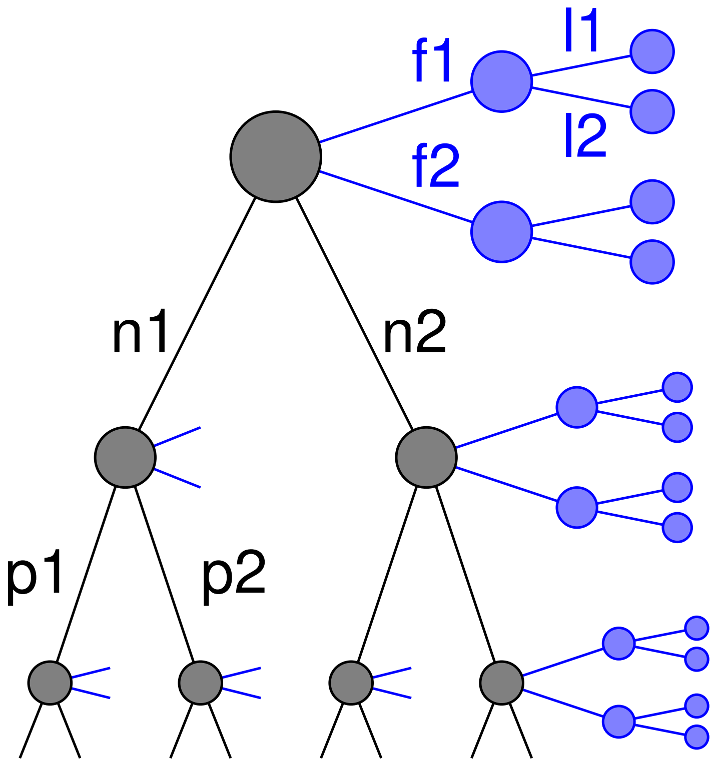
Concepts from APART

“Granularity” of Properties

- location
  - whole system, cluster, machine, process, thread
- program structure (region)
  - program, module, function, loop, basic block, statement
- time
  - complete execution, execution phases

$\Rightarrow$ **Multidimensional Tree of Properties**

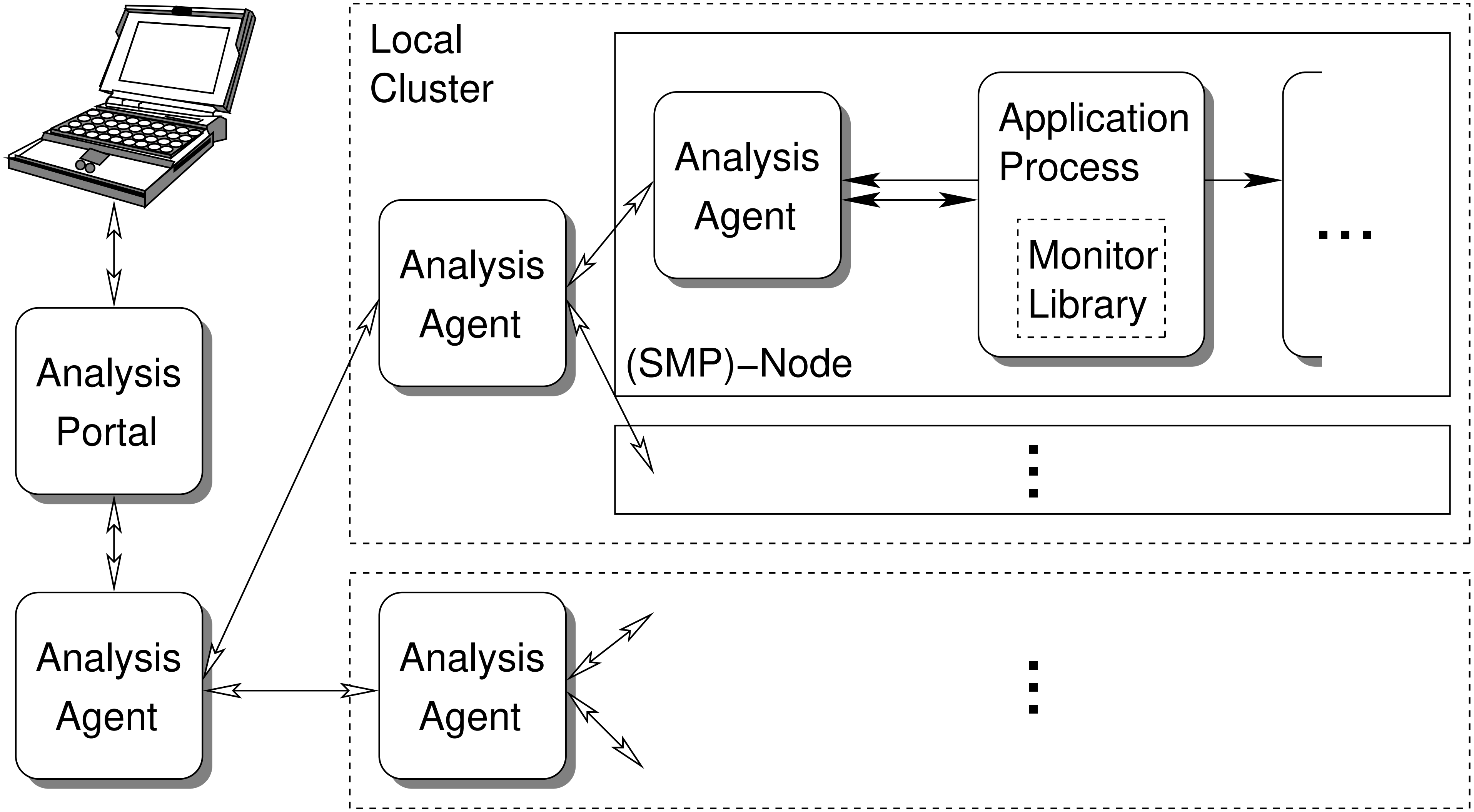
Tree of Properties



- search strategy?
  - top-down (hypothesis driven)
  - bottom-up (data driven)
- in addition to refinement
  - properties with "distributed" conditions
    e.g. load imbalance

A Possible Implementation on the Grid

Agent Hierarchy



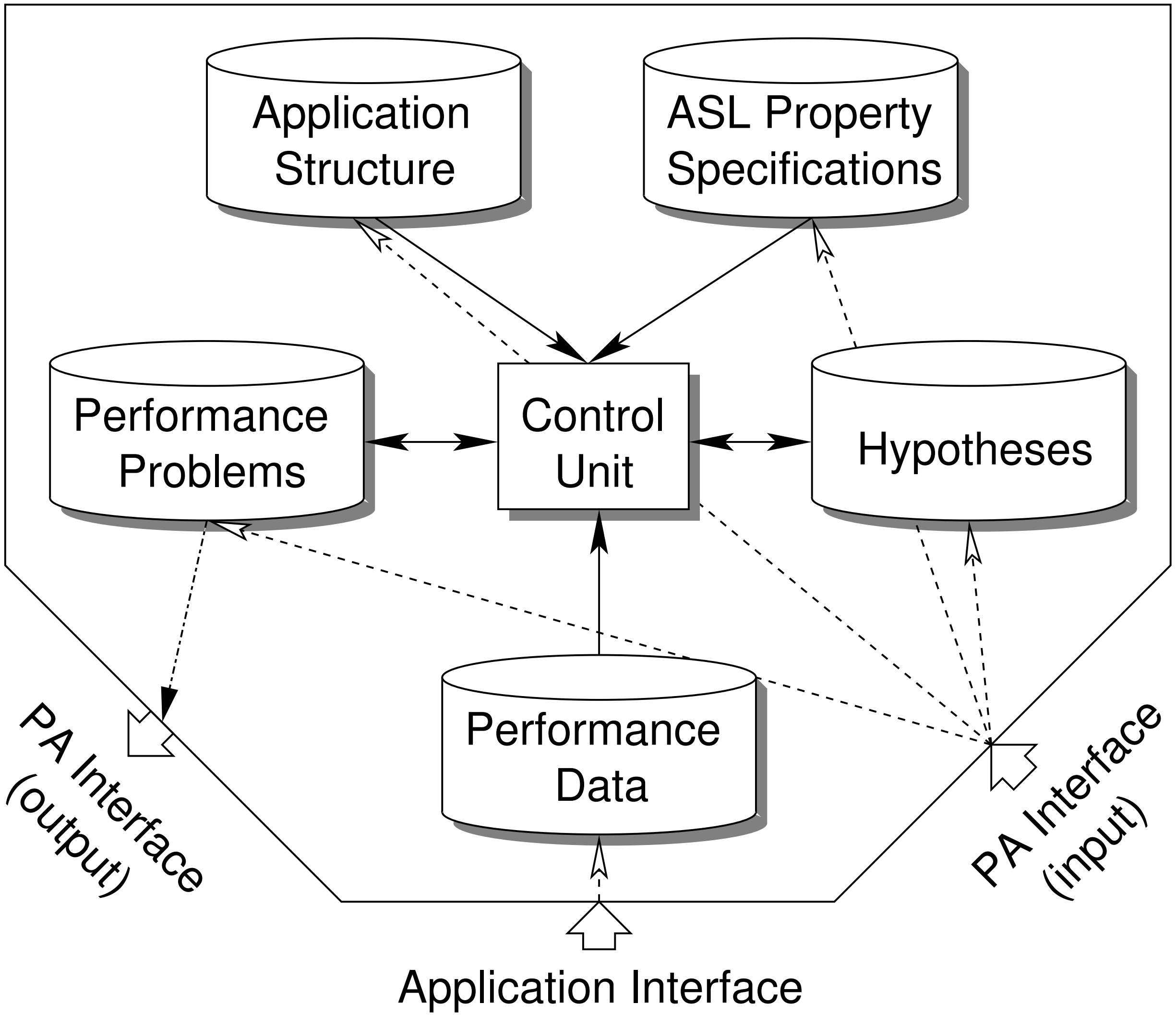
A Possible Implementation on the Grid

Communication

- towards leaves:
  - ASL property specifications
  - program structure information
  - evaluation requests for hypotheses
  - priorities for hypotheses
- towards root:
  - performance problems
  - partially evaluated hypotheses

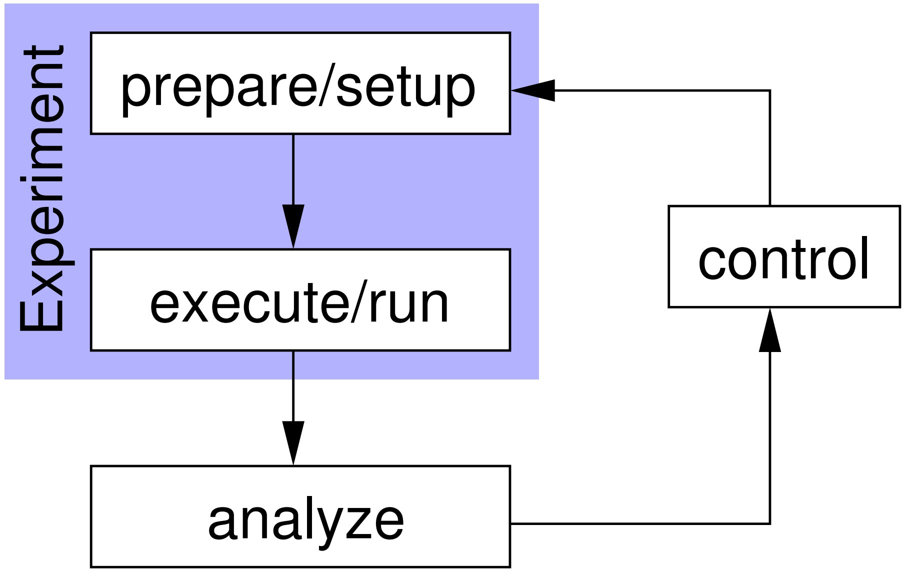
A Possible Implementation on the Grid

Structure of an Agent



A Possible Implementation on the Grid

Performance Experiments in APART



Types of experiments

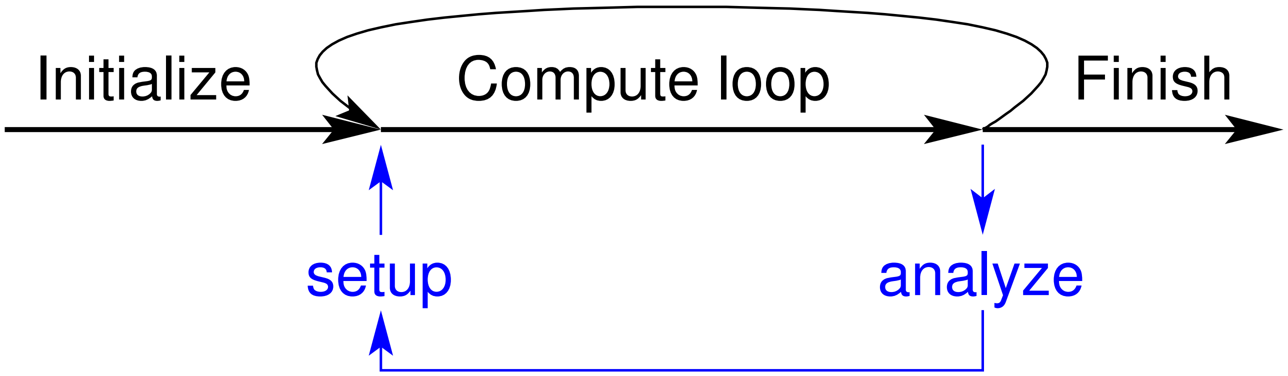
- multiple complete executions
  - e.g. for scalability analysis
- single complete execution
  - off-line monitoring
- part of an execution
  - on-line monitoring

For the Grid

- on-line monitoring (amount of data, repeatability)
- needs knowledge on execution phases

A Possible Implementation on the Grid

Execution Phases



- refinement of hypotheses only possible for repeating phases
- ideally: phases marked by programmer
- fallbacks:
  - loop information from compiler
  - extry / exit of “large” functions

A Possible Implementation on the Grid

Program Pattern Information

- optional information on application's design patterns
  - e.g. SPMD, master/worker, pipeline, divide & conquer, ...
- allows to derive more special performance properties, e.g.
  - *master is overloaded*
  - *pipeline stalled by stage xyz*
- incremental on-line specification of patterns
  - mapping to symbolic information compiled into executable
    e.g. line numbers, function names, ...

Status

Automatic Performance Analysis

- APART: ASL and principal design
- Peridot: design of on-line analysis agents
- ESPRIT proposal: extraction of high level performance properties on the Grid

OMIS/OCM

- OMIS 2.0 is available (`http://wwwbode.in.tum.de/~omis`)
- OCM will be released under GPL
- current work (ICS-AGH, Krakow):
  OCM for the Grid, Java-Extension for OMIS