# Performance Analysis on the Grid

## Activities in the CrossGrid Project

Roland Wismüller

LRR-TUM, Technische Universität München
Institute for Software Science, University of Vienna

# CrossGrid / Applications

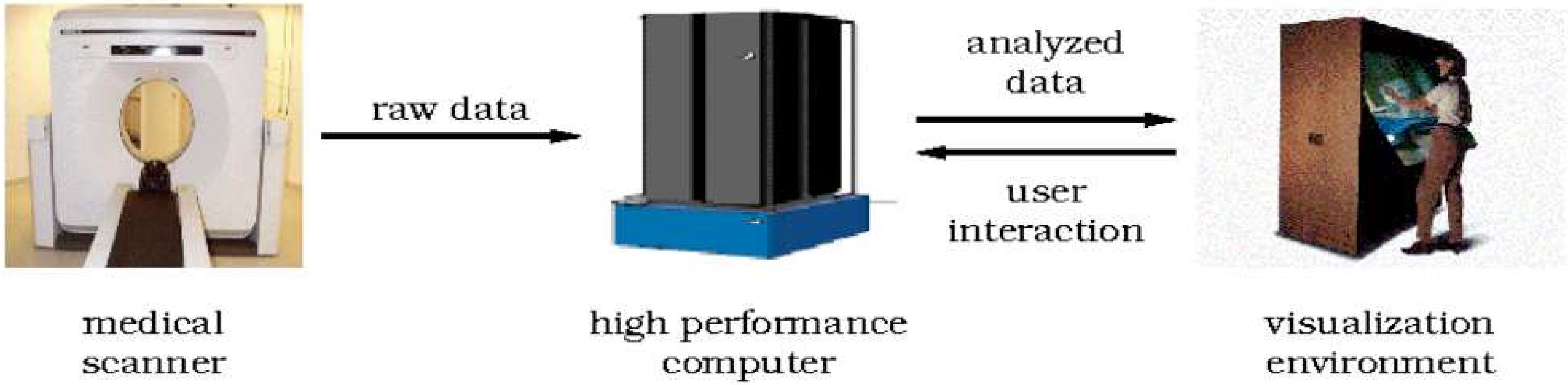
## CrossGrid

- Funded by EU: 03/2002 - 02/2005
- Goal: extend Grid towards *interactive* applications
- `http://www.eu-crossgrid.org`

## CrossGrid Applications

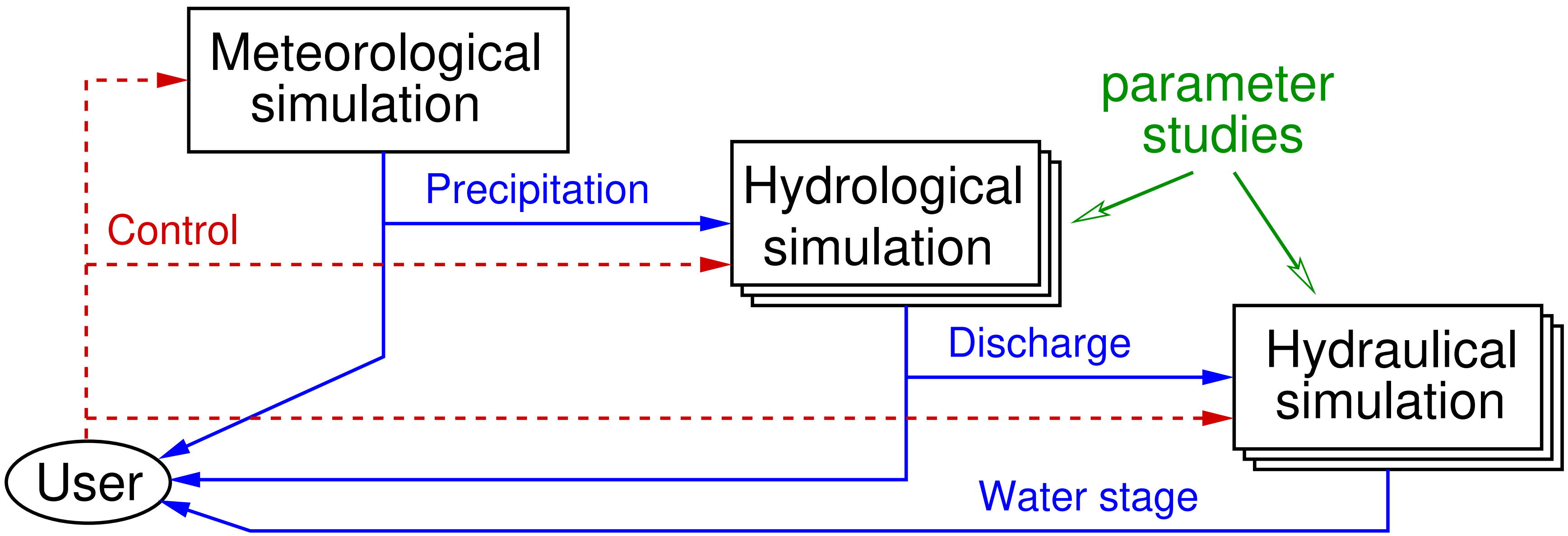
- Simulation of vascular blood flow
- Flooding crisis support
- Data analysis in High Energy Physics
- Weather forecast and air pollution

# Simulation of vascular blood flow




- Interactive visualization and simulation
  - response times are critical
  - 0.1 sec (head movement) to 5 min (change in simulation)
- Performance analysis
  - response time and its breakdown
  - performance data for specific interactions

# Flooding crisis support




- Interactive steering
  - start / kill simulations (based on new data / performance)
- Performance analysis
  - convergence behavior, progress of computation
  - infrastructure related performance data

# On-line Performance Analysis

## Properties

- User defines performance *measurements* at run-time
- Results are values evolving over time
- Selective run-time instrumentation
- Counters and (integrating) timers, *very* limited tracing

## Motivation

- Correlate performance data with interactions
- On-line data necessary for steering
- Reduce the amount of measurement data
- Allow to start monitoring *after* application started

# Special Requirements

- Integration with Grid infrastructure
  - e.g. job submission / scheduling
- Acquire infrastructure related performance data
  - steering / resource selection
  - assessment of application's performance data
- Focus on interaction of distributed application components
  - components are already optimized (assumption!)
- Provide data meaningful in the context of an application
  - take into account application specific information

# Application specific information

## Examples

- How much disk I/O was caused by a specific user interaction?
- How much of the response time is due to
  - communication between client and server,
  - computation within client / server,
  - communication inside the server,
  - ...?
- What is the current convergence rate of my solver?
  (When will the result be available?)

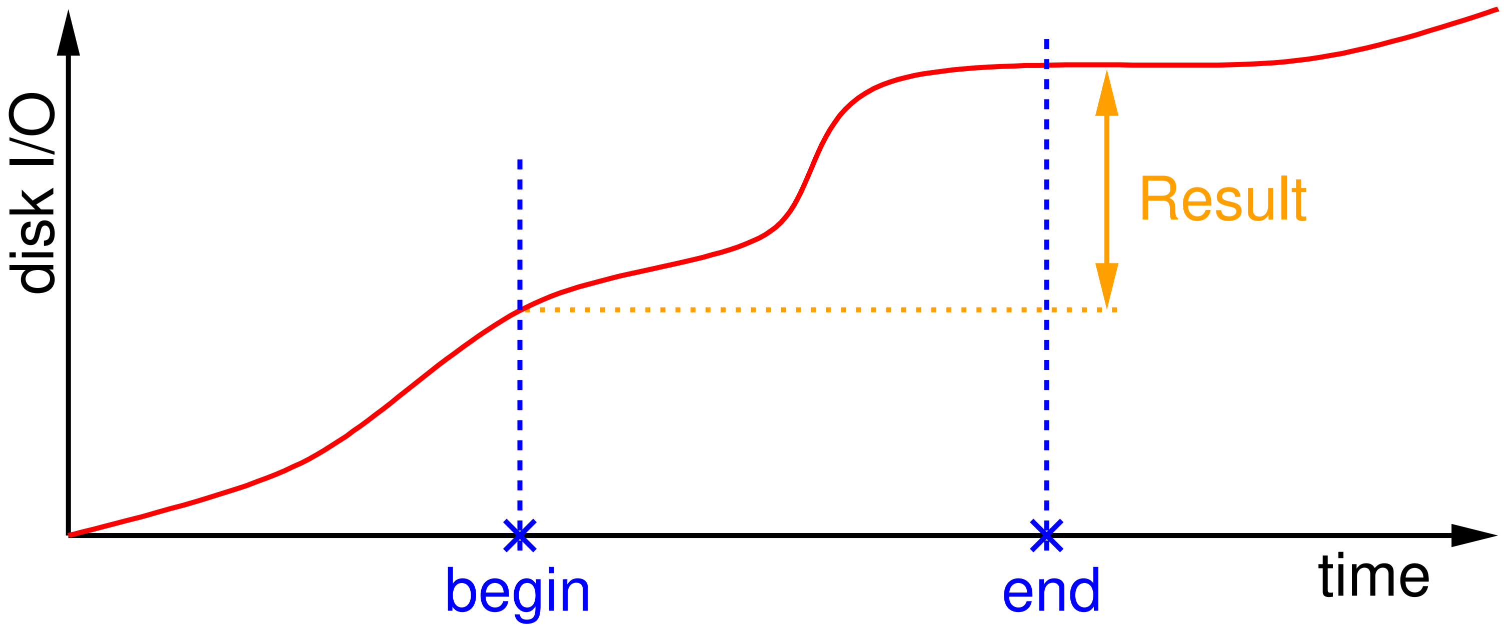
# Approach (1)

- Performance analysis tool G-PM offers “standard” metrics
  - e.g. CPU time, communication time, disk I/O; node load
  - automatic instrumentation
- Application programmer provides a minimum amount of information
  - relevant events inside application
  - association between events
  - relevant data computed by the application
  - manual source code instrumentation (*probes*)
- G-PM allows to define new metrics
  - based on existing ones and application specific information

# Approach (2)

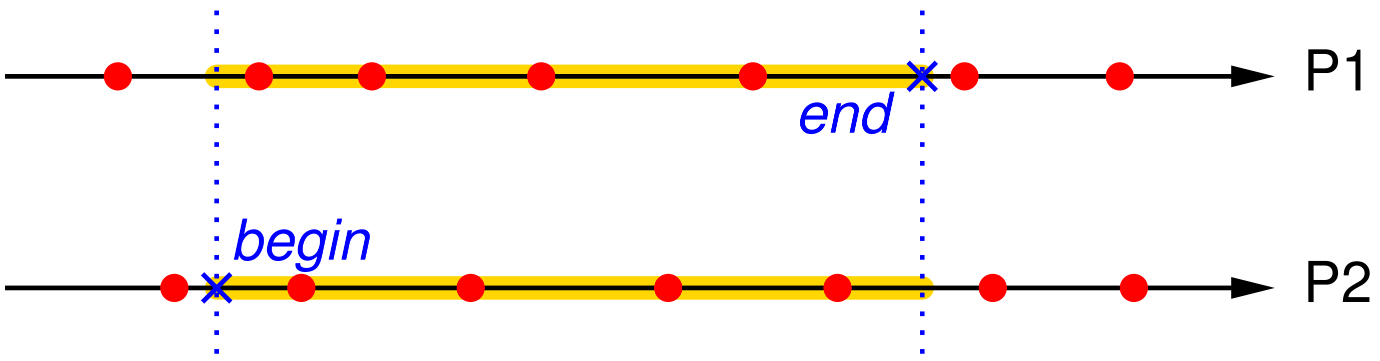
## Disk I/O caused by an interaction (naive)

- Define measurement “disk I/O”
- Probes mark beginning and end of an interaction
- Result = disk I/O (end) - disk I/O (begin)



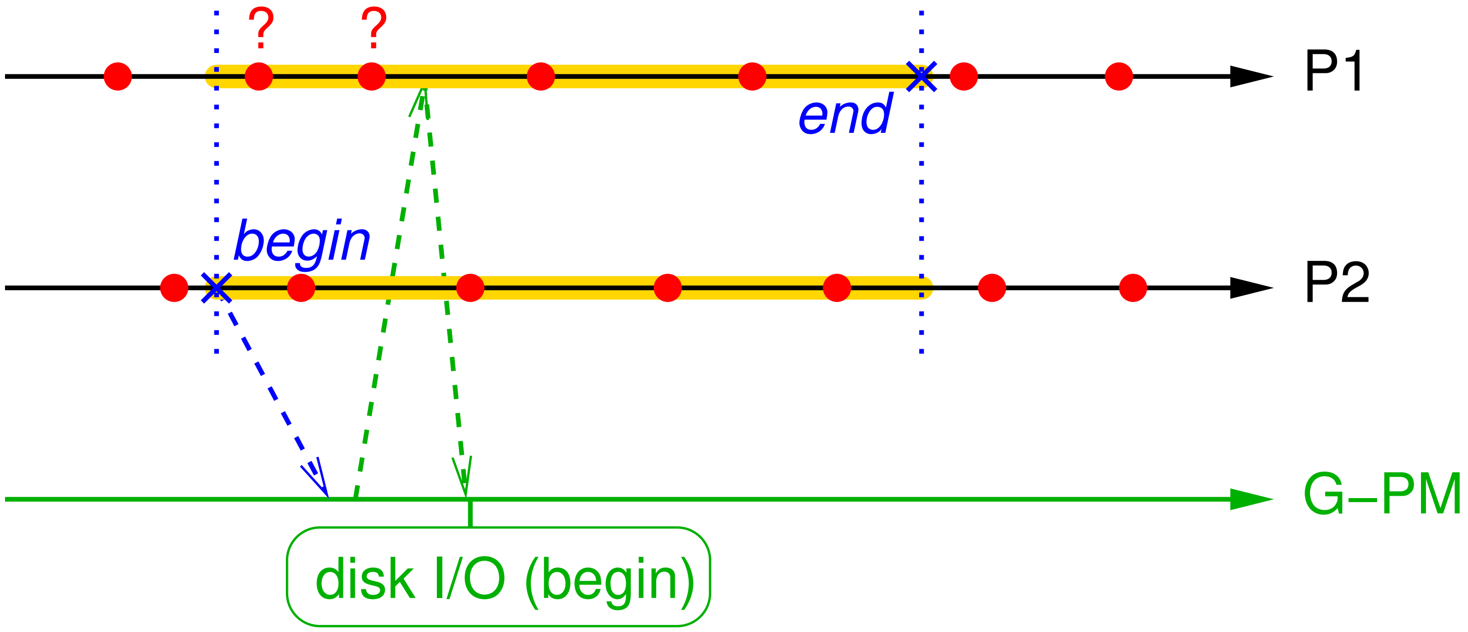
# Problems (1)

- We have a distributed system
- We can't read the measurement result instantaneously



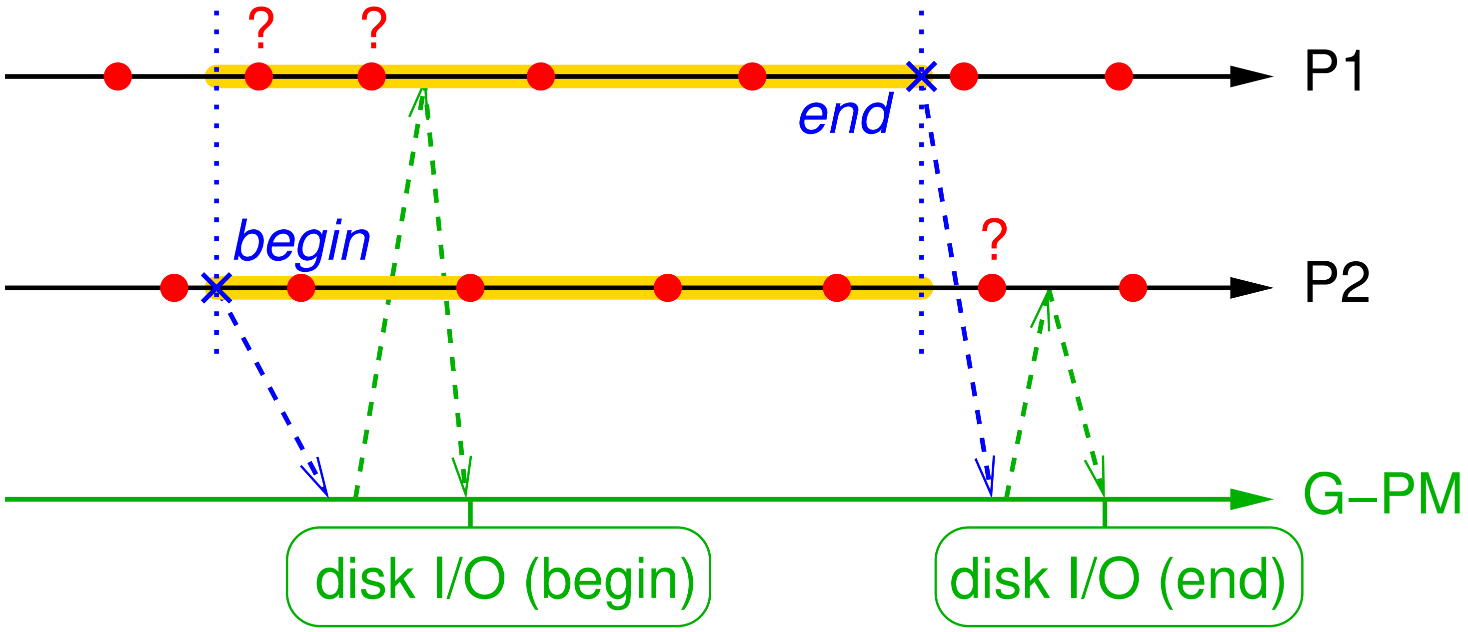
# Problems (1)

- We have a distributed system
- We can't read the measurement result instantaneously



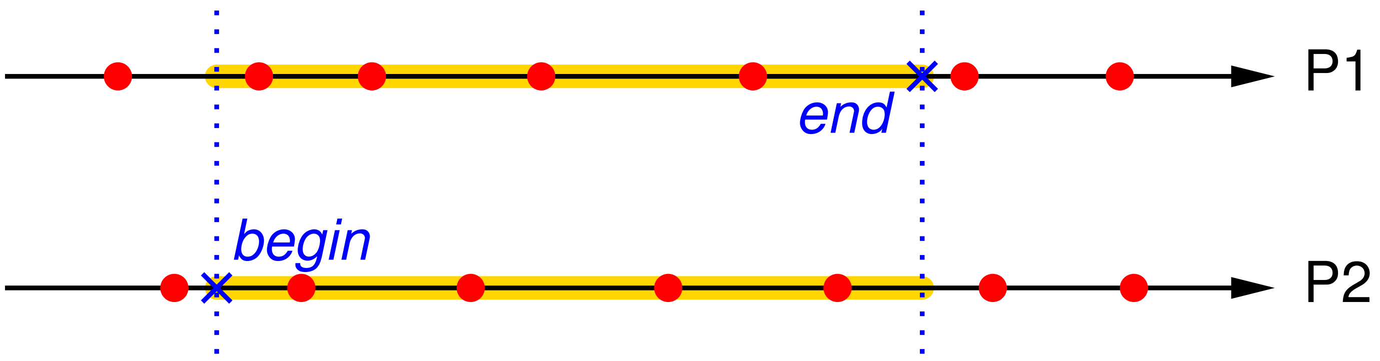
# Problems (1)

- We have a distributed system
- We can't read the measurement result instantaneously



# Problems (1)

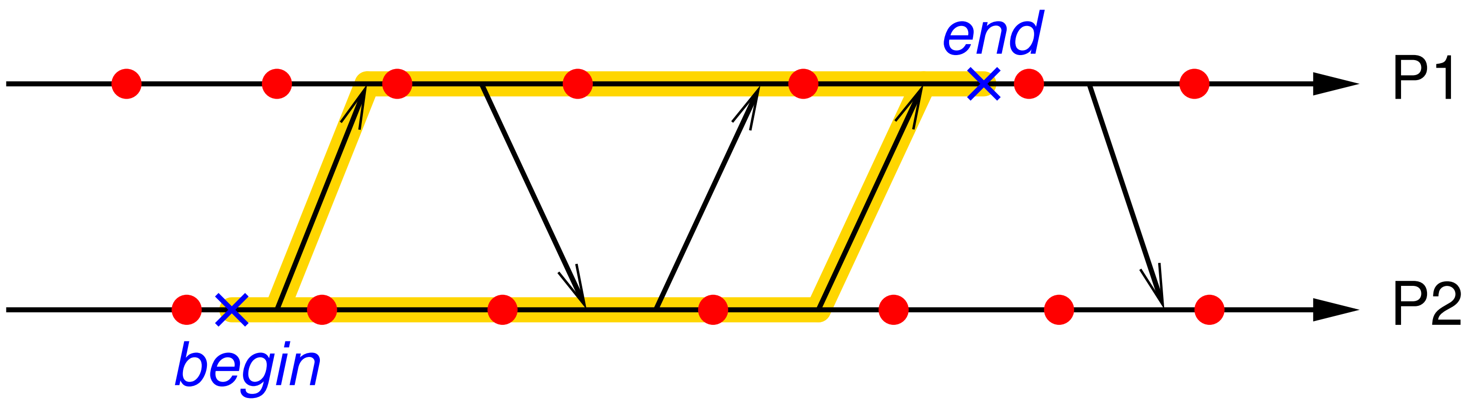
- We have a distributed system
- We can't read the measurement result instantaneously




- But: This is not what we need!
- We should consider only I/O *caused* by the interaction

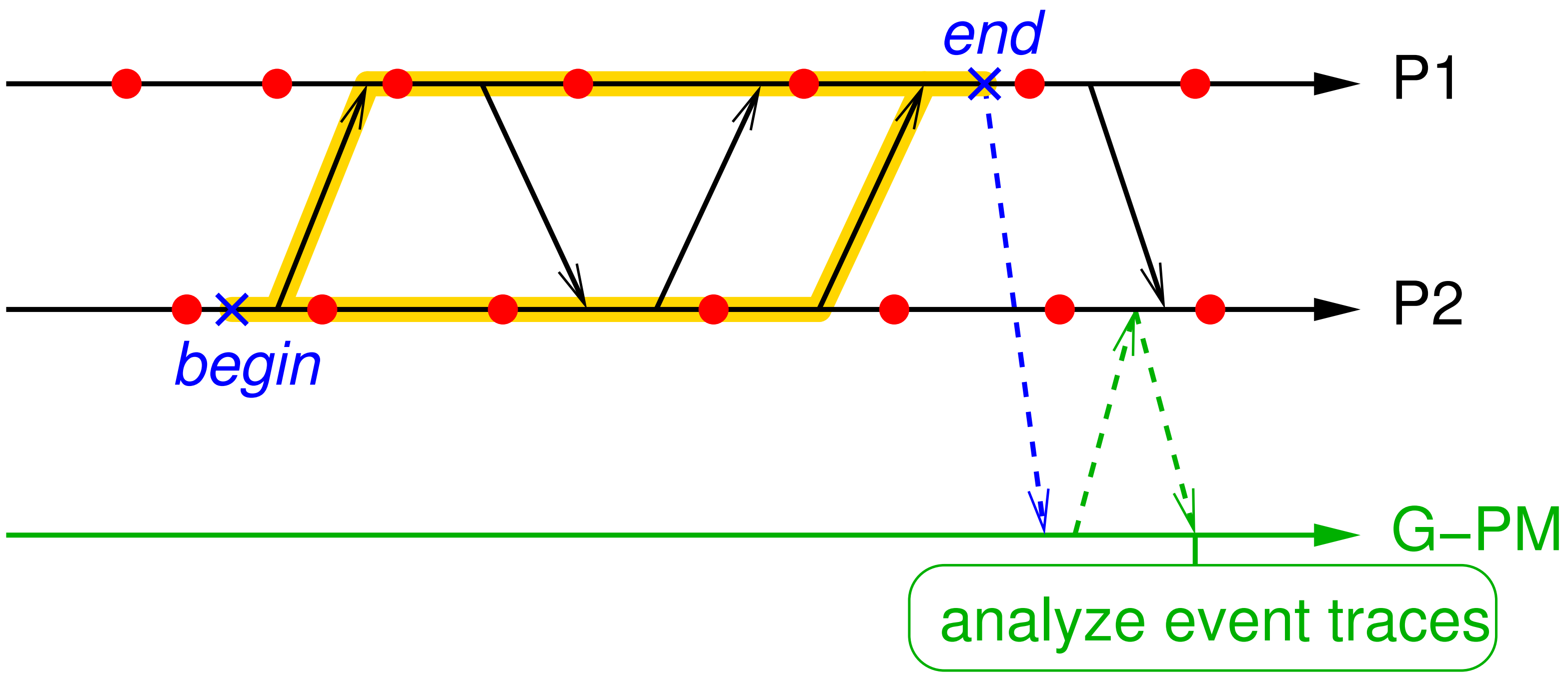
# Problems (1)

- Use causal relationships (*happended before*, $\rightarrow$)
- Consider only I/O events causally "between" *begin* and *end*



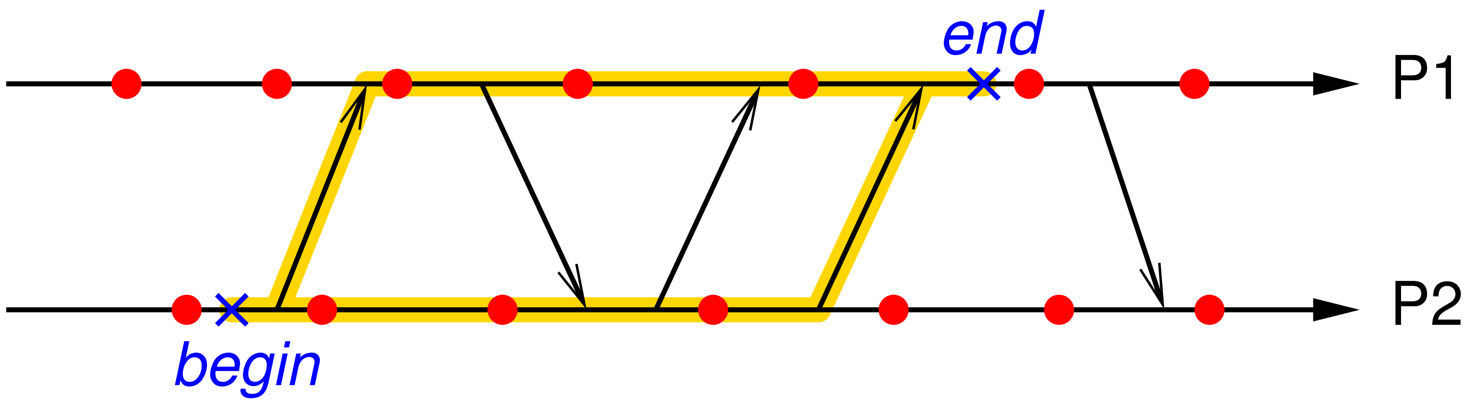
# Problems (1)

- Use causal relationships (*happended before*, $\rightarrow$)
- Consider only I/O events causally "between" *begin* and *end*



# Problems (1)

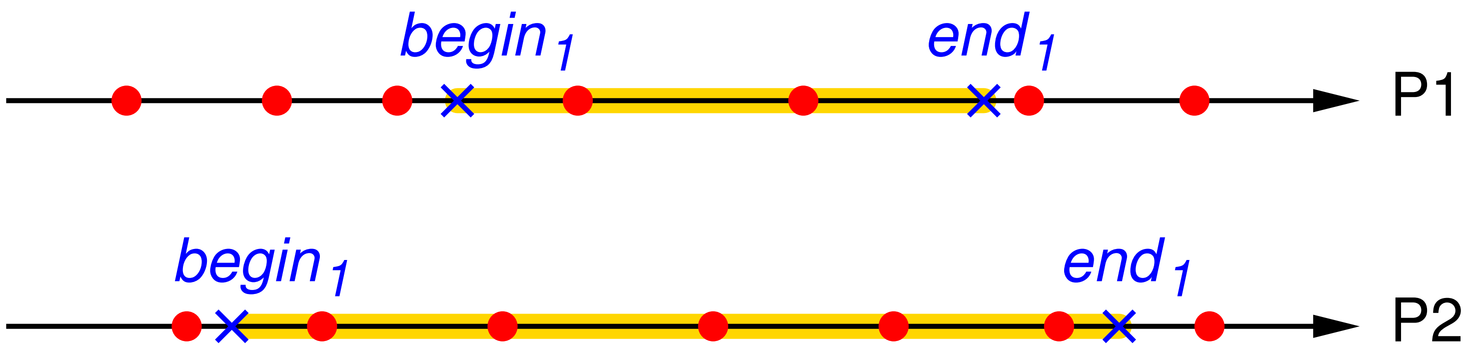
- Use causal relationships (*happended before*, $\rightarrow$)
- Consider only I/O events causally "between" *begin* and *end*




- But: This is not what we need, either!
- $a \rightarrow b$ does not imply that $b$ is actually caused by $a$

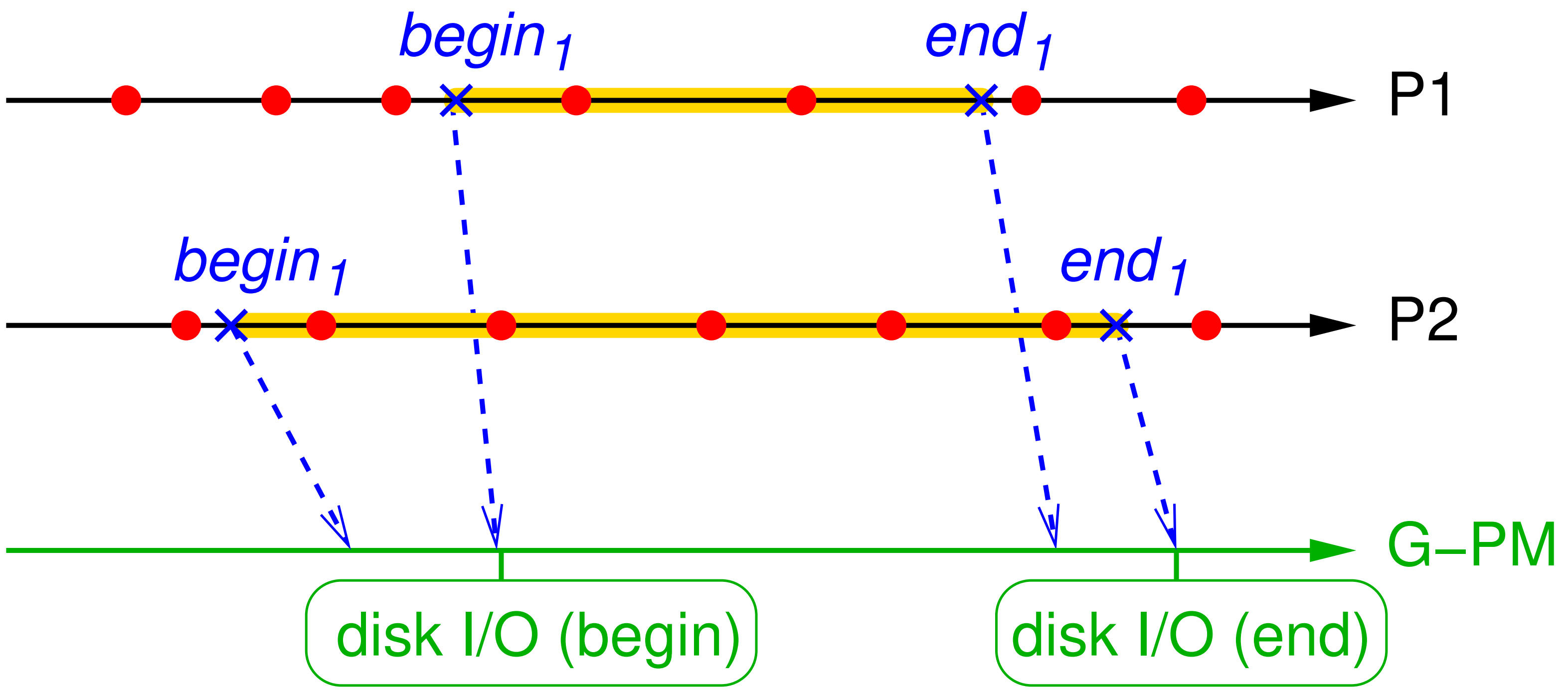
# Problems (1)

- The user must provide this information!
- Corresponding *begin*s and *end*s for each sequential process
- Correspondence indicated by matching IDs passed to probes (*virtual time*)



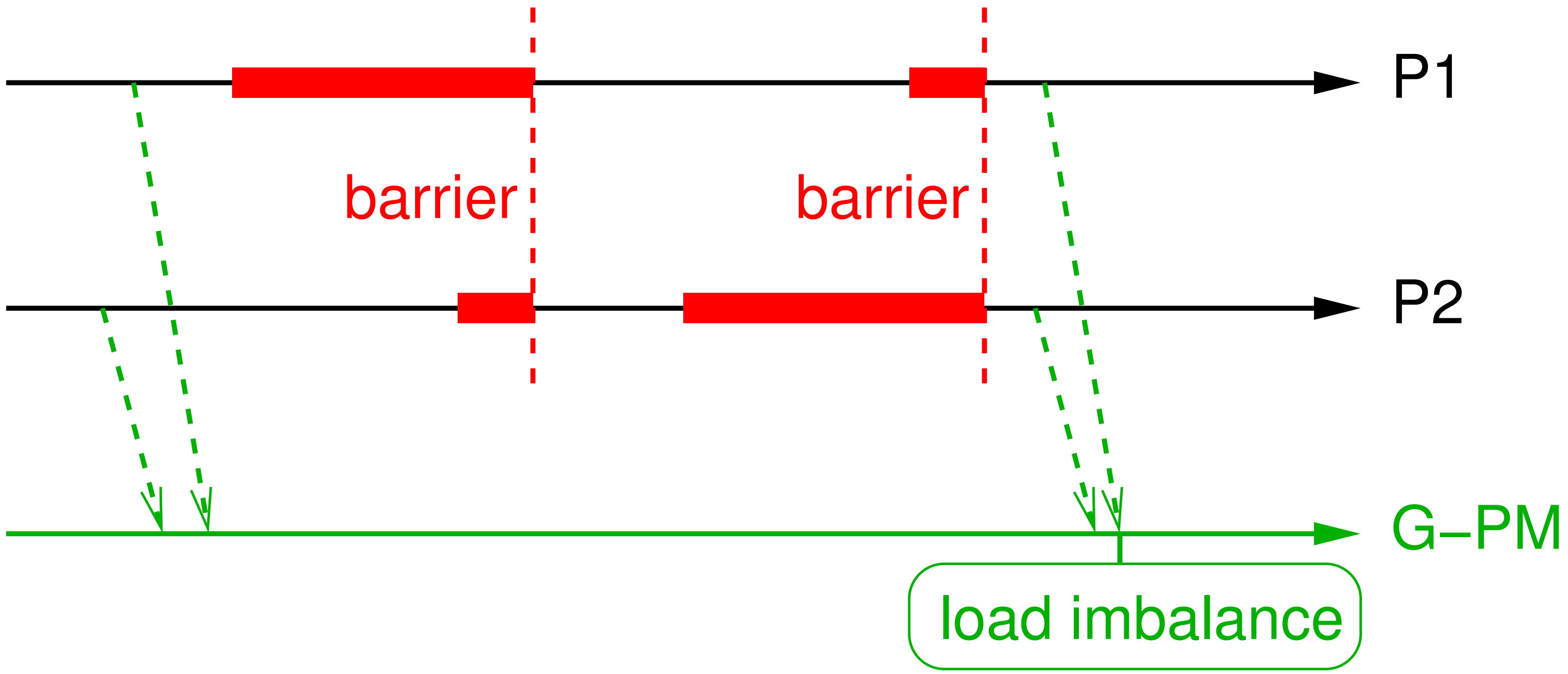
# Problems (1)

- The user must provide this information!
- Corresponding *begin*s and *end*s for each sequential process
- Correspondence indicated by matching IDs passed to probes (*virtual time*)



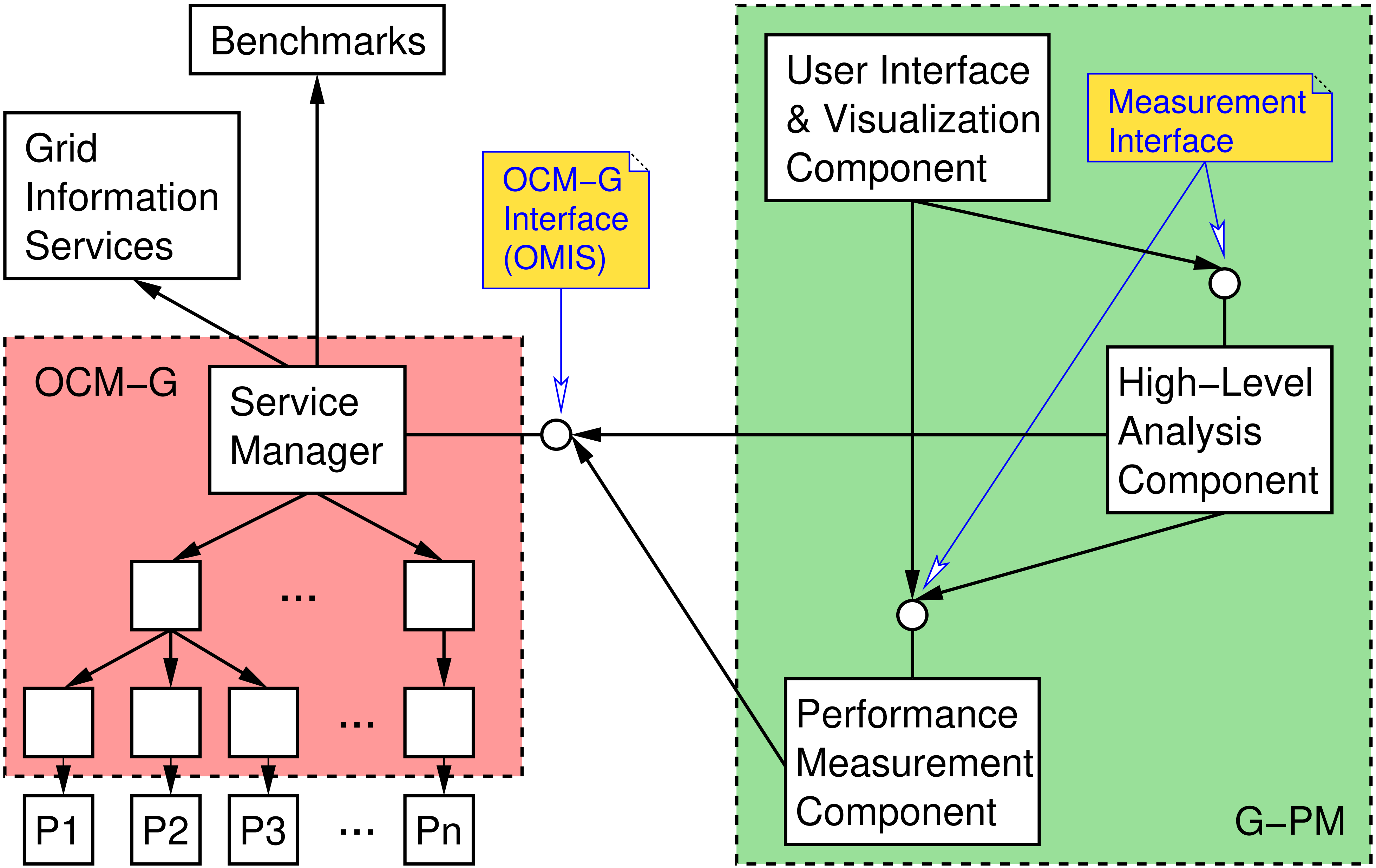
# Problems (2)

- Interesting effects may be hidden by summaries
- E.g. load imbalance (waiting time at a barrier):

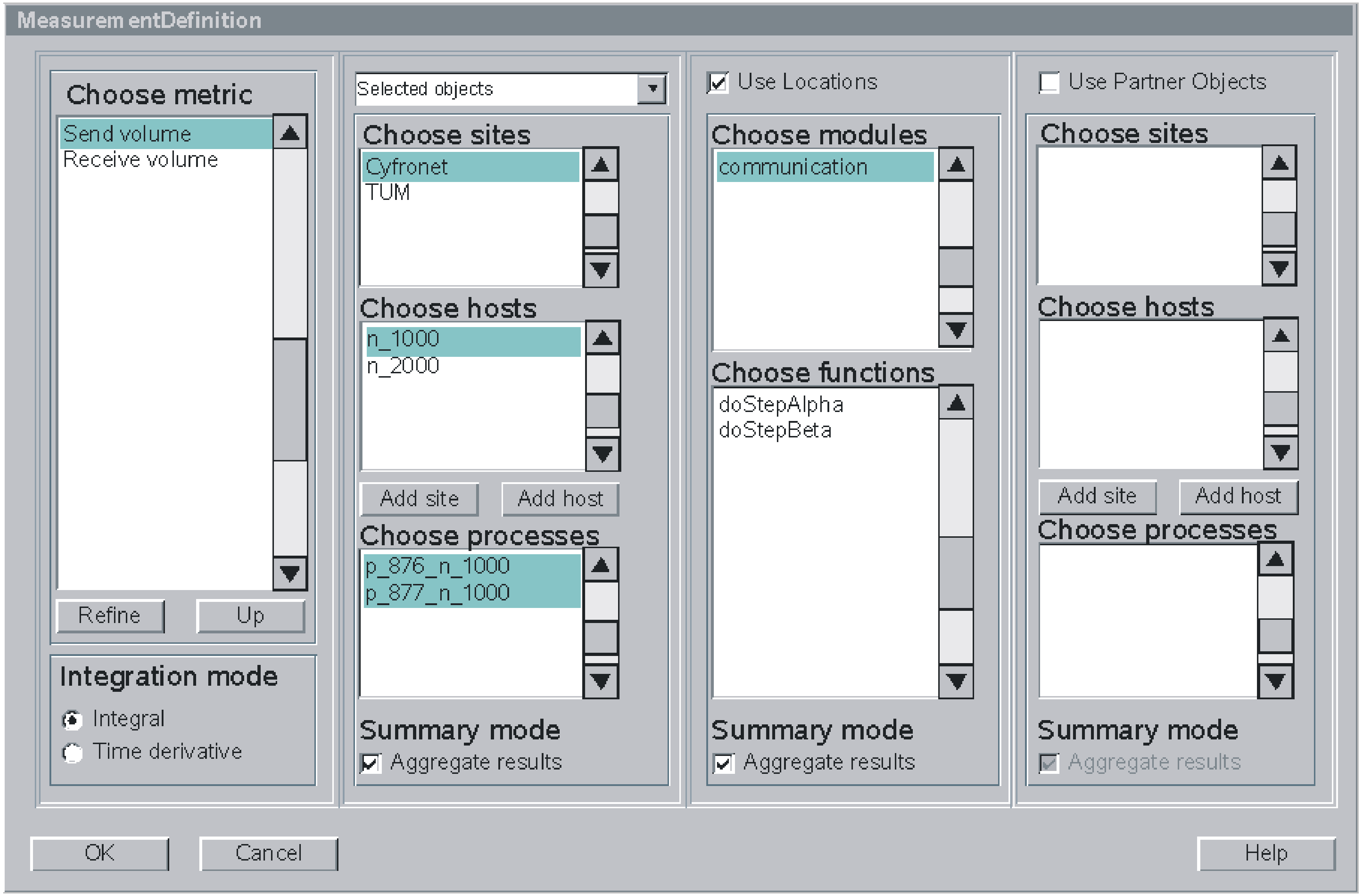



- Solution: read a "trace" of timer values: one value for each execution of the barrier (*virtual time*)

# Design: Components



# Design: User Interface



# Design: Measurement Interface

- Measurement specification consists of:
  - Metrics: *what to measure*
  - Set of objects: *where to measure*
  - Set of partner objects
  - Set of program regions

  } *restrictions*
- Two main (abstract) base classes:
  - *Metrics*: knows how to implement a measurement
  - *ActiveMeasurement*: knows how to read it
- Many different implementations:
  - PMC: sampled, function, aggregated functions, ...
  - HLAC: different strategies based on metrics definition

# Design: Metrics Specification Language

- Specification must (implicitly) specify the necessary actions
  - when a measurement is defined
  - when a probe is executed
  - when a measurement is read
- Existing languages:
  - ASL (Apart), JavaPSL (U. Vienna): performance properties
  - MDL (U. Madison, Paradyn): low level measurements
  - EARL (Research Center Jülich): trace analysis
- None of them seems to be appropriate for G-PM:
  - different goals and focus
  - in G-PM, language is exposed to user ($\Rightarrow$ must be simple)

# Design: Metrics Specification Language

## Use a declarative language?

```
my_IO_volume(Process[] procs, File[] fil, Region[] reg)
{
  vol[p][vt] =   IO_volume(p, fil, reg) AT end(p, vt)
               - IO_volume(p, fil, reg) AT begin(p, vt);
  result[vt] = SUM(vol[p][vt] WHERE p IN procs);
  return result[];
}
```

- Uses data flow principle and unification
- Problem: how to compile / interpret such a language?

# Conclusions

- Interactive Grid applications demand
  - on-line performance analysis
  - application specific measurements
- Basic idea:
  - provide set of standard metrics
  - allow for application specific events and data
  - provide specification language to define new metrics
- Simple idea, but many problems to solve

# Status

- Finished:
  - coarse design of monitoring system
  - coarse design of G-PM
  - specification of GUI and measurement interface
- To do:
  - first implementation of OCM-G, PMC, UIVC
  - demonstrators for HLAC
  - design of metrics specification language
  - implementation of a compiler/interpreter
  - optimization of measurements