State Based Visualization of PVM Applications

Roland Wismüller

Lehrstuhl für Rechnertechnik
und Rechnerorganisation (LRR-TUM)
Technische Universität München

Motivation

- Debugging/test of parallel programs: often information on the global state of the communication / process system is needed:
  - which tasks exist, how are they interacting?
  - which tasks are blocked in barriers / receive calls?
  - which groups do exist, which tasks are members?
  - what is the contents of the tasks' receive queues?
- Parallel debuggers don't allow to answer these questions easily.

⇒ State based visualization tool VISTOP
  - global view on program execution at task level
  - coarse grained stepping through program execution
  - goal: integration with parallel debugger

VISTOP: Overview

Visualization strategy:

- program execution presented as sequence of snapshots
- snapshot shows consistent global state
- stepping from one snapshot to the next
- state transitions can be animated

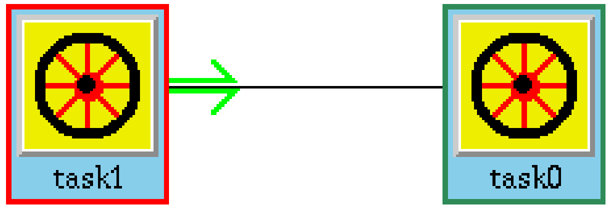
Different views on program state:

- Concurrency View: interaction of tasks, increased scalability by hierarchical approach
- Object Creation View: dynamic creation of tasks
- System View: mapping from tasks to hosts
- Group Manager: groups and group members

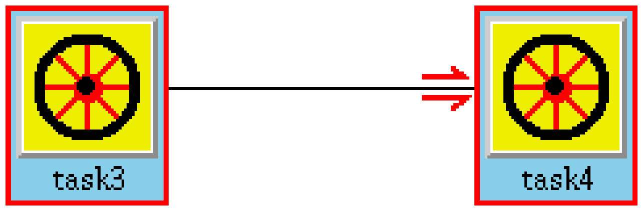
Visualization Patterns in Concurrency View

Task | with timeout | with error | Barrier | No explicit group object

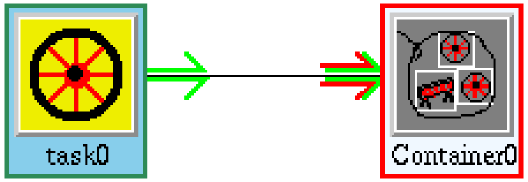
Task1 is sending to Task0:

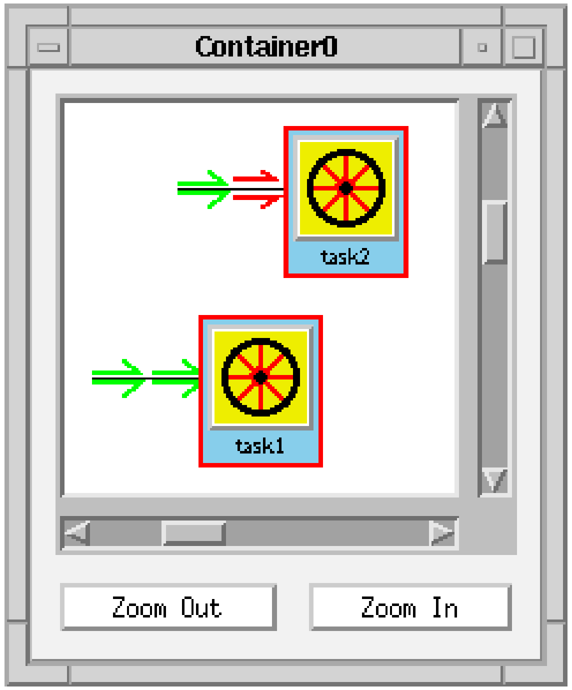


Task4 waits to receive from Task3:

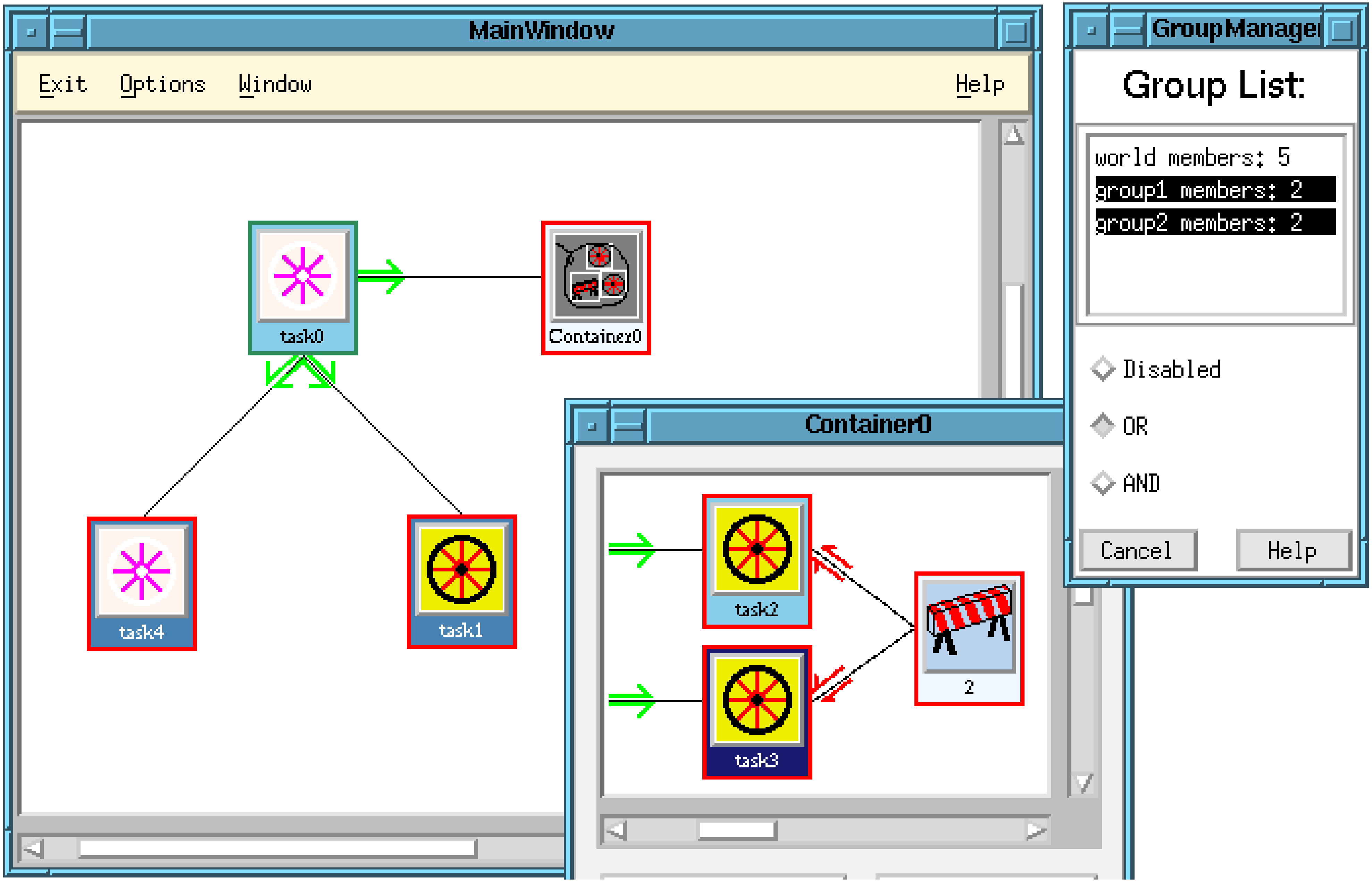


Hierarchical Objects (Container)

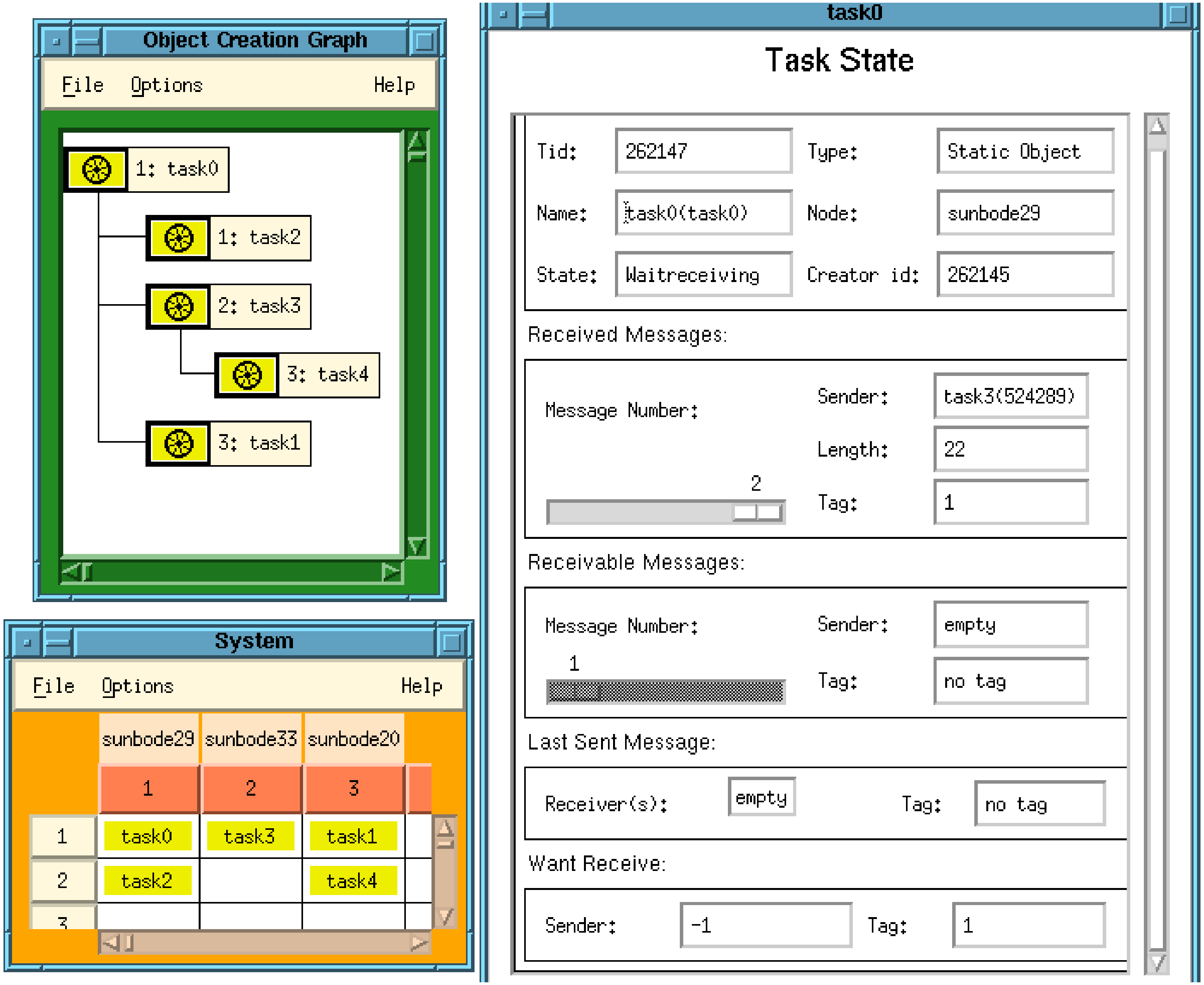




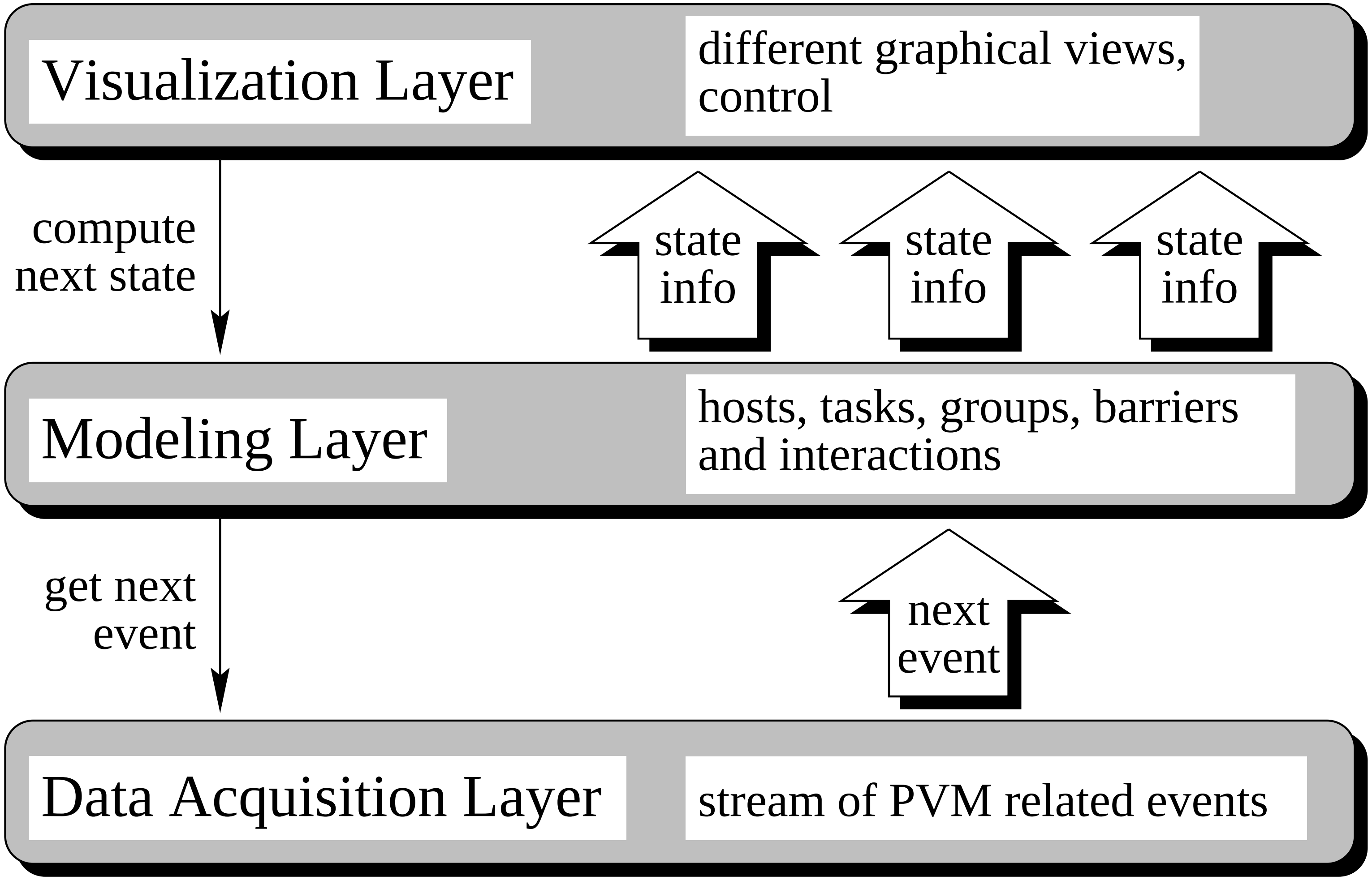
VISTOP: Concurrency View



VISTOP: Other Views



Structure of VISTOP



Data Acquisition Layer (1)

Tasks:

- monitor relevant events during program execution
- forward events to modeling layer in suitable order
- provide initial state for modeling layer

Final implementation:

- based on OMIS compliant monitoring system

$\leadsto$ allows on-line usage

$\leadsto$ allows integration with parallel debugger

Current prototype:

- uses PVM built-in trace facility (via XPVM)

$\leadsto$ off-line usage

Data Acquisition Layer (2)

Problems:

- broadcast event doesn't contain information on actual receiver tasks
- no event "message inserted into receive queue"
  - $\rightsquigarrow$ impossible to decide whether receive call blocks
  - $\rightsquigarrow$ impossible to compute receive queue
  - $\rightsquigarrow$ receive operations with sender wildcard can not be visualized as desired

$\Rightarrow$ modified PVM library

Modeling Layer

Task:

- compute new program state from old state and next event
  $\leadsto$ both forward and backward

Implementation:

- state machine modeling behavior of PVM

  $\leadsto$ additional information for backward stepping

- collection of C++ objects for

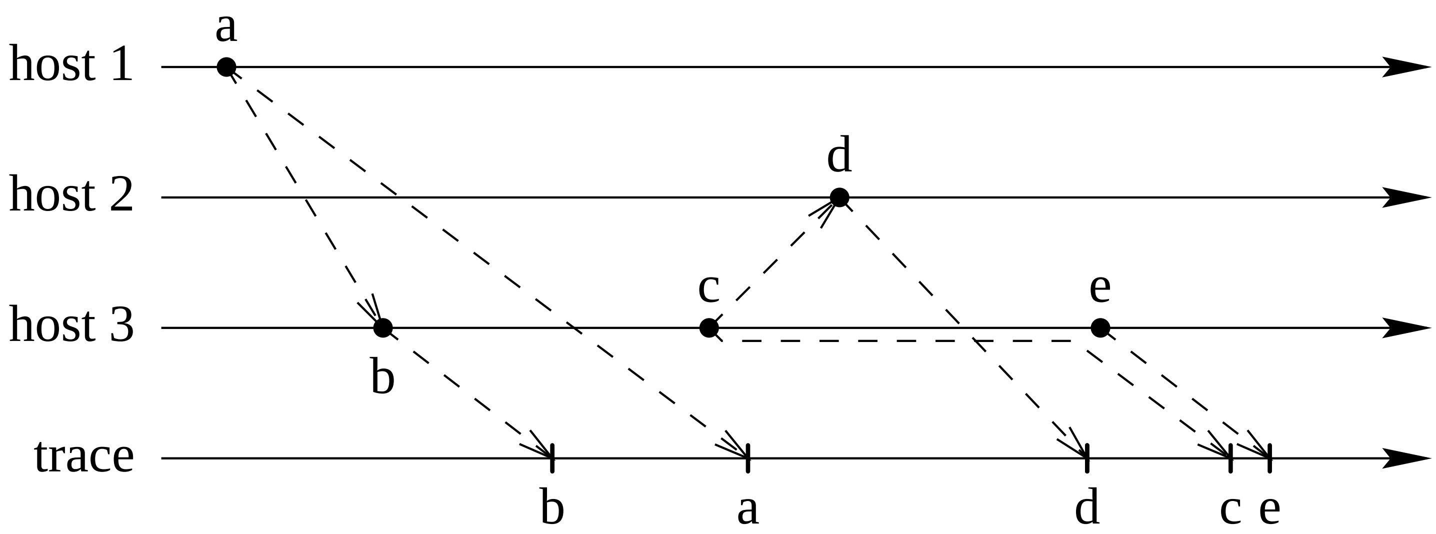
  – hosts, tasks, groups, barriers
  – interactions between these objects

- detailed error checking (e.g. unpacking, PVM calls)

Ordering of Events (1)

Problem:

- in distributed systems: trace may contain events in an inconsistent order



$\Rightarrow$ data acquisition layer must re-sort events

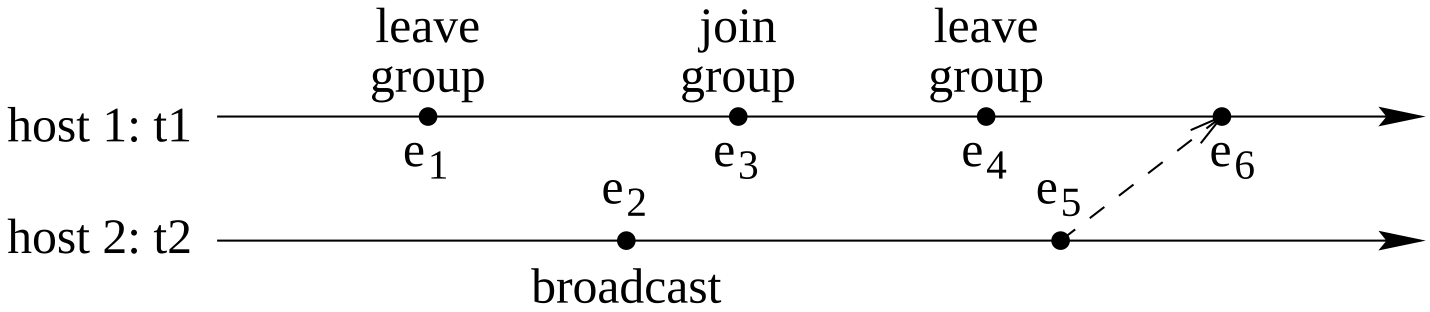
$\leadsto$ no global time $\Rightarrow$ other criteria

- events on a single node sorted correctly
- causal dependencies ($e \rightarrow e'$, if e.g. e = send, e' = receive)

Ordering of Events (2)

Additional problem: consistency of states and events

- some events can only occur in a specific state
- example: broadcast to a group, actually sent to task $t1$



- sequence $e_1, e_2, e_3, e_4, e_5, e_6$ obviously incorrect

$\Rightarrow$ select e as next event, only if

  - there is no unprocessed event e' with $e' \rightarrow e$
  - e can happen in the currently modeled state

$\leadsto$ wrong choice may lead into a deadlock!

Ordering of Events (3)

Algorithm:

$E = \{\, e \mid \text{e is next unprocessed event on some node} \,\}$

$E = E - \{\, e \mid \text{there is an unprocessed event } e' \text{ with } e' \rightarrow e$
$\qquad \text{or } e \text{ cannot happen in the current state} \,\}$

$E_1 = \{\, e \in E \mid e \text{ does not “hinder” any unprocessed } e' \text{ with } e \,||\, e' \,\}$

if $E_1 \neq \emptyset$, choose $e \in E_1$

else $E_2 = E - \{\, e \mid \text{the negative effect of } e \text{ on some event } e' \text{ is}$
$\qquad \text{not undone by a later event } \overline{e} \text{ with } e' \not\rightarrow \overline{e} \,\}$

if $|E_2| = 1$, choose $e \in E_2$

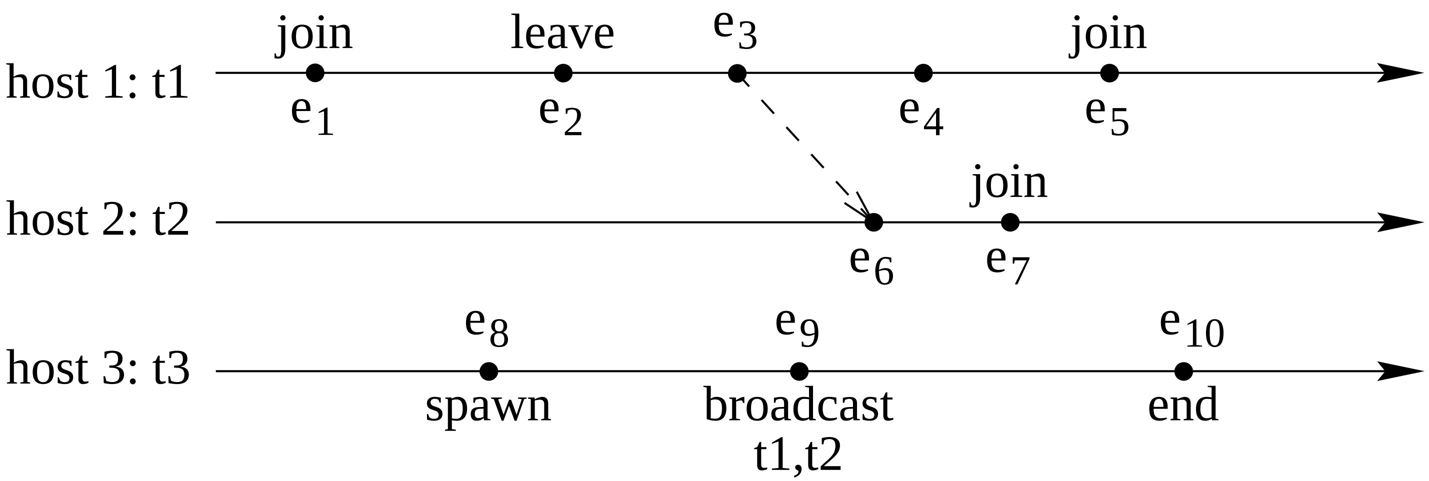
else examine all state dependent events e' with $e \,||\, e'$
for some relevant state changing event $e \in E$

determine *cuts* where e' can be executed

determine a sequence of these cuts

Ordering of Events (4)

Example:



$$E = \{e_1, e_6, e_8\} \rightsquigarrow E_1 = \{e_1, e_8\} \rightsquigarrow e_1$$
$$E = \{e_2, e_6, e_8\} \rightsquigarrow E_1 = \{e_8\} \rightsquigarrow e_8$$
$$E = \{e_2, e_6, e_9\} \rightsquigarrow E_1 = \emptyset \rightsquigarrow E_2 = \{e_2\} \rightsquigarrow e_2$$

If $e_4 =$ "send to $t3$, failed since $t3$ not existent":

$$E = \{e_1, e_6, e_8\} \rightsquigarrow E_1 = \{e_1, e_8\} \rightsquigarrow e_1$$
$$E = \{e_2, e_6, e_8\} \rightsquigarrow E_1 = \emptyset \rightsquigarrow E_2 = \{e_2, e_8\} \rightsquigarrow e_2$$

Conclusion and Future Work

Results:

- state based visualization can be a helpful tool for parallel program development
- PVM built-in traces not sufficient
- modeling of global states is relatively simple, the hard part is the ordering of events

Ongoing Work:

$\leadsto$ further development of sorting algorithm

$\leadsto$ utilization of OMIS compliant monitoring system
($\Rightarrow$ on-line usage)

$\leadsto$ integration with parallel debugger, checkpointer, and deterministic execution controller
($\Rightarrow$ THE TOOL-SET for PVM)