Using Monitoring Techniques to Support the Cooperation of Software Components

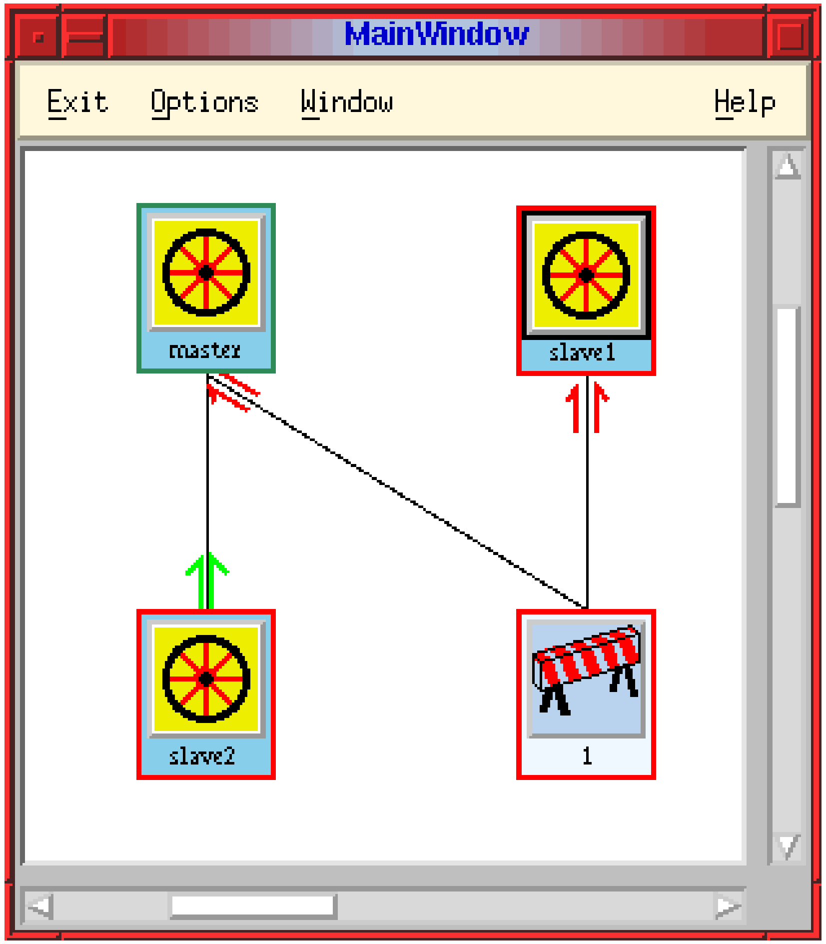
Roland Wismüller

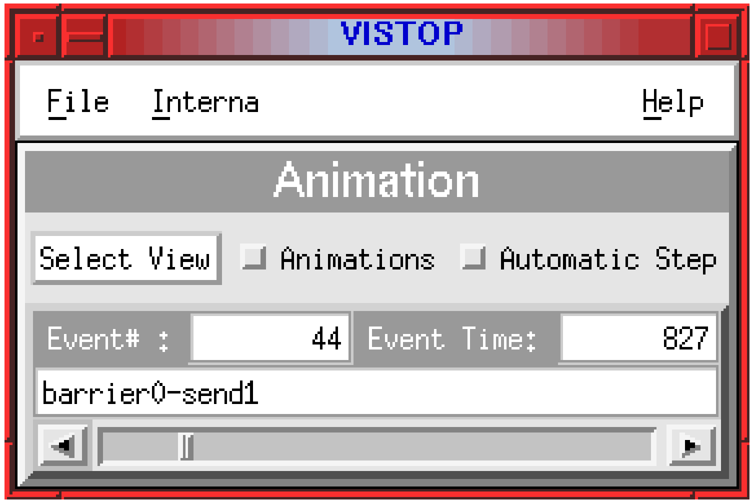
LRR-TUM
Technische Universität München
Munich, Germany

1. Motivation: A Tool Environment

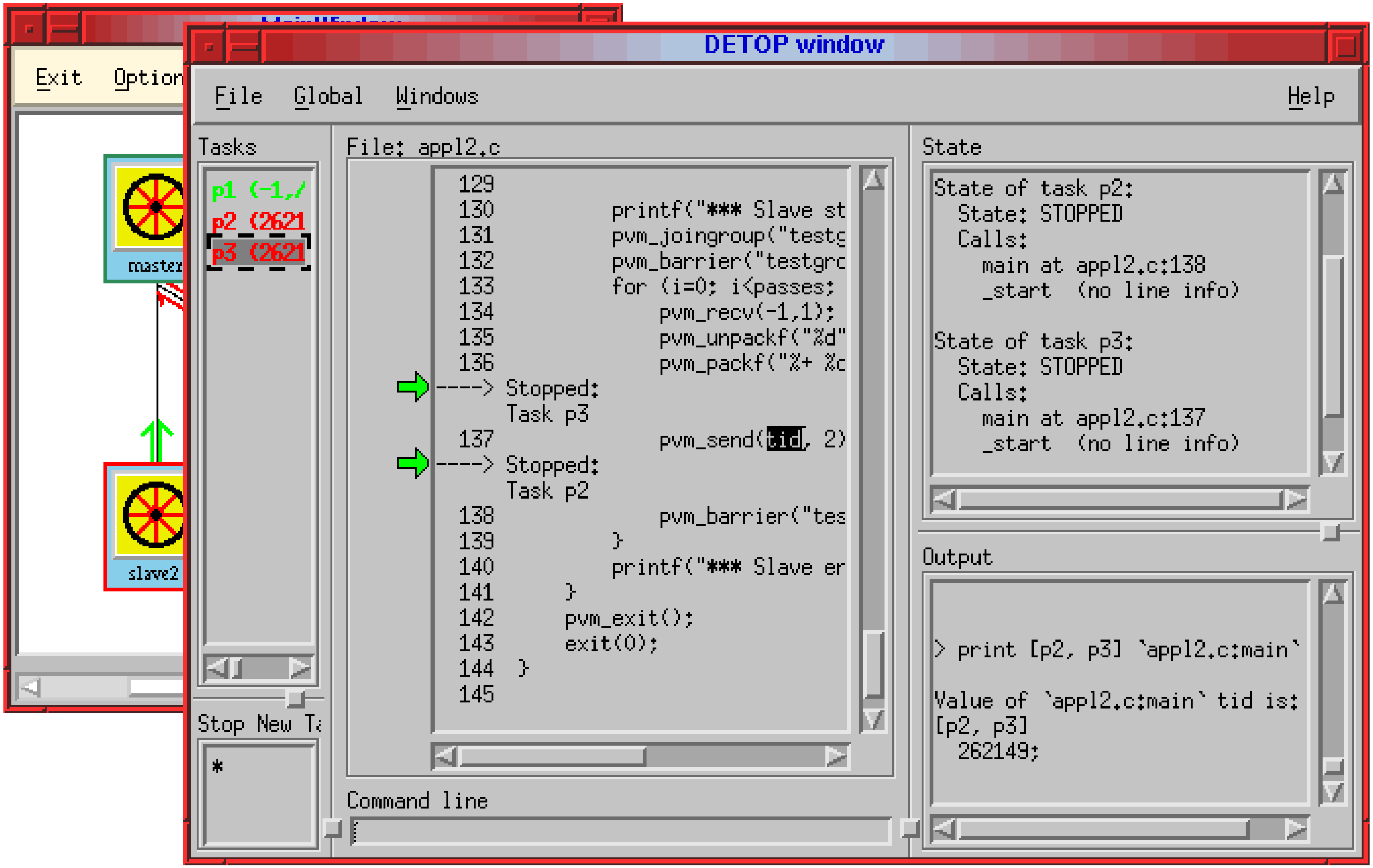
Starting point:

- development of on-line tools at LRR-TUM
  - source level debugger DETOP
  - program flow visualizer VISTOP
  - performance analysis, deterministic execution, checkpointing, ...
- desire: combine these tools
  - e.g. visualizer + debugger
    - overview of program execution
    - detailed inspection


MainWindow
Exit Options Window Help
master
slave1
slave2
1

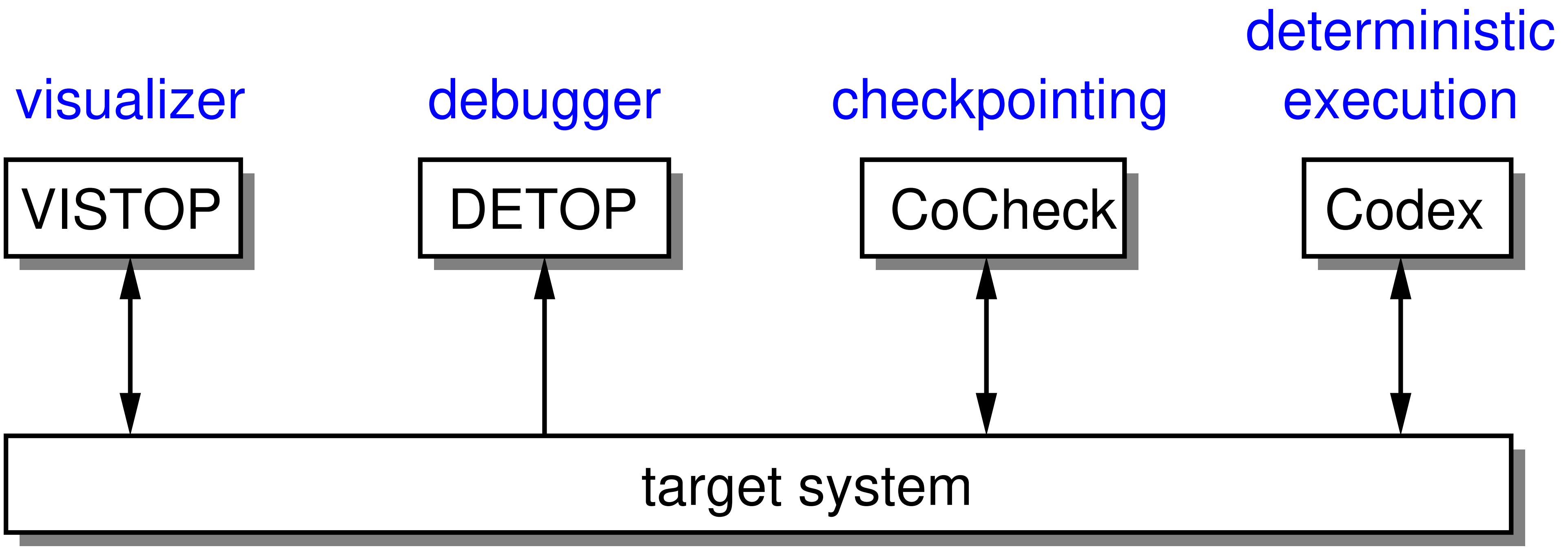

VISTOP
File Interna Help
Animation
Select View
Animations
Automatic Step
Event# : 44
Event Time: 827
barrier0-send1

1. Motivation: A Tool Environment

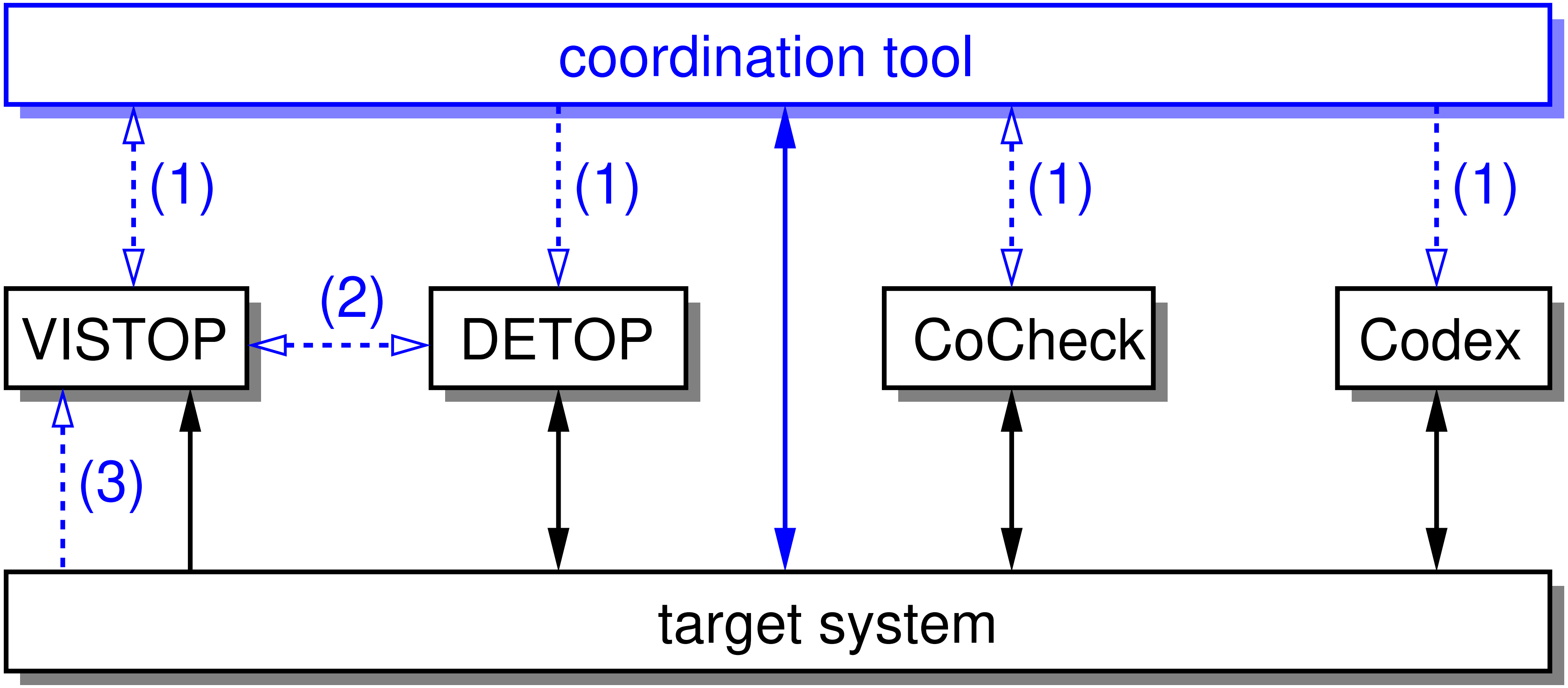


1. Motivation: A Tool Environment

The environment's structure



The environment's structure



2. Cooperation of Software Components

In the tool environment:

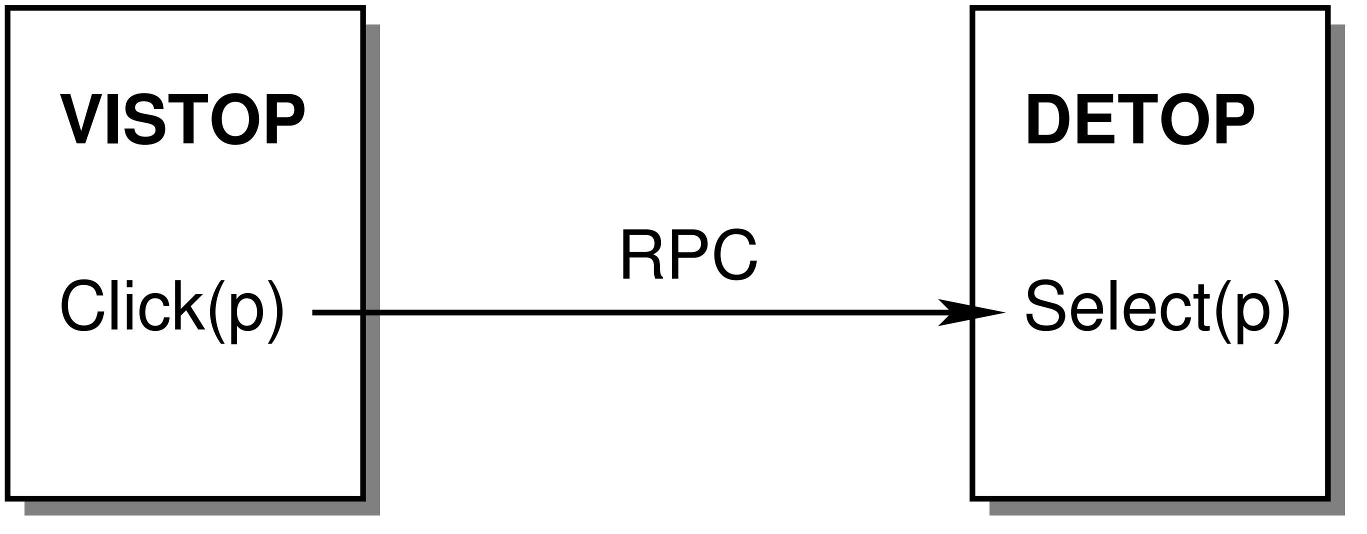
- e.g. synchronized selection, consistent displays
- steering of tools by coordination tool

General problem:

- creating software environments from *existing* components
  - with user interaction / GUIs
  - with active, independent behavior (in contrast to servers)
  - developed independently, for stand-alone use
  - without modifying the components' source code

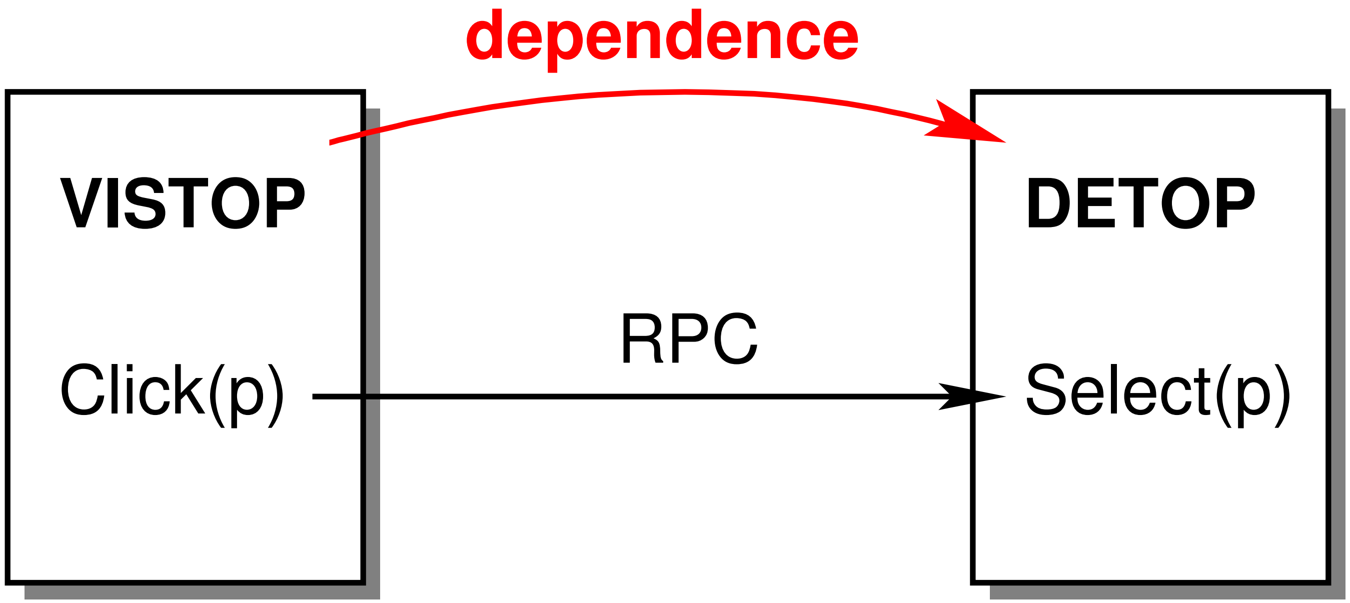
Hardwiring

- e.g. PVM, MPI, RPC, CORBA, PCTE



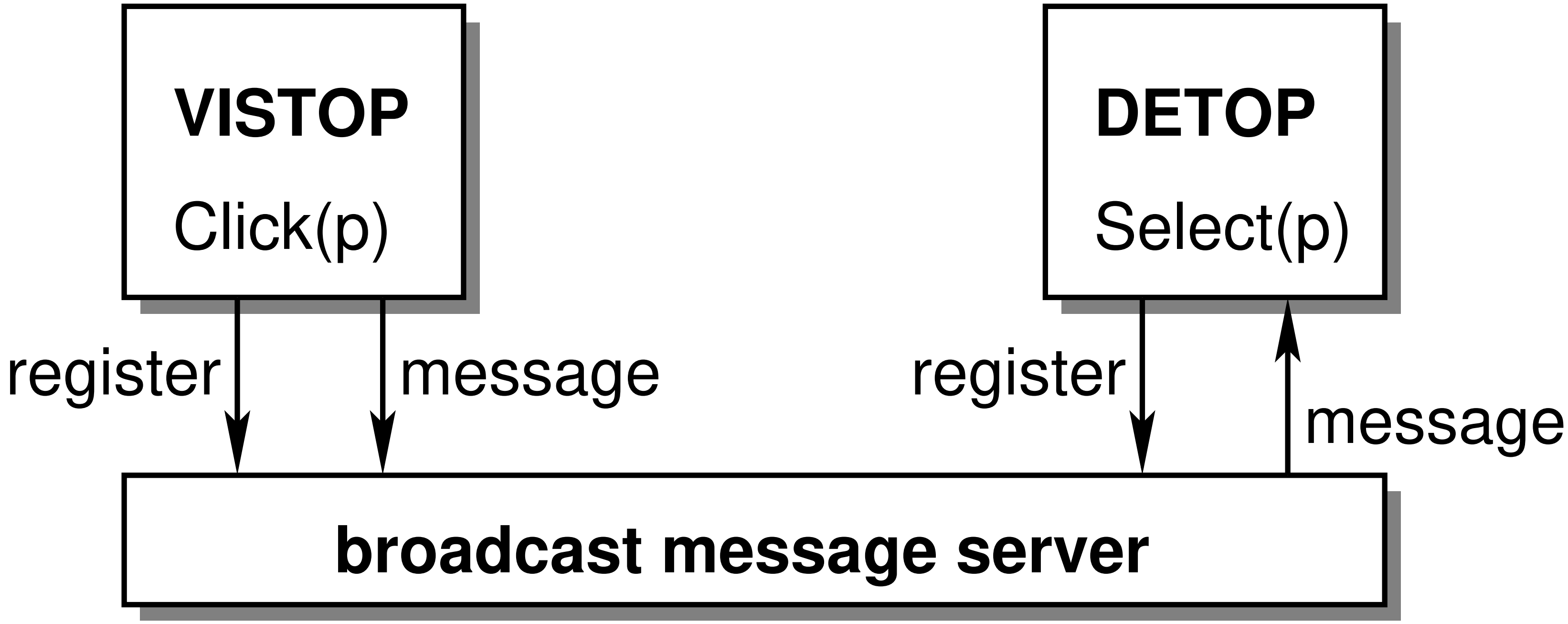
Hardwiring

- e.g. PVM, MPI, RPC, CORBA, PCTE



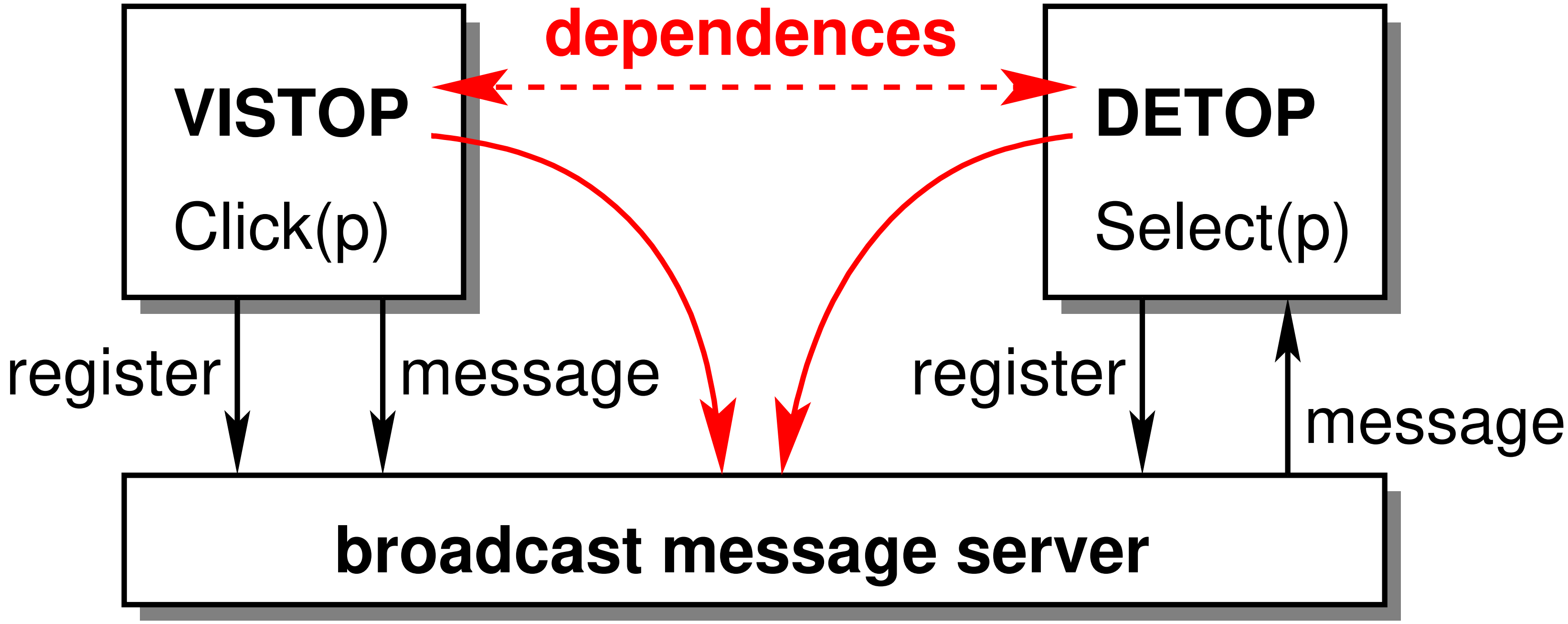
Broadcast message server

- e.g. ToolTalk, SoftBench



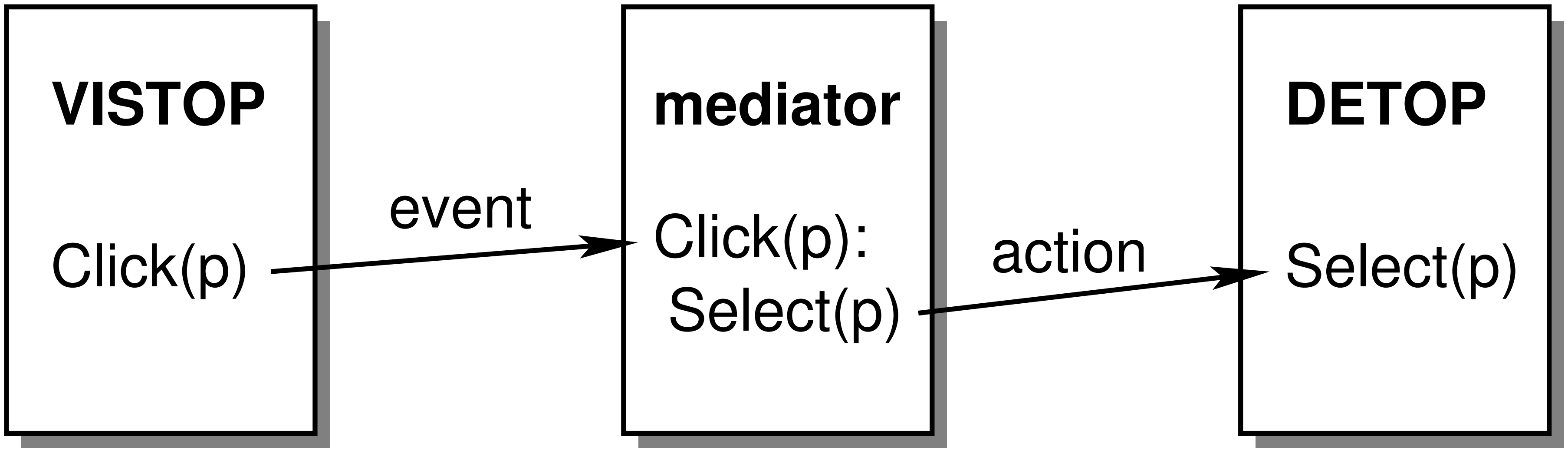
Broadcast message server

- e.g. ToolTalk, SoftBench



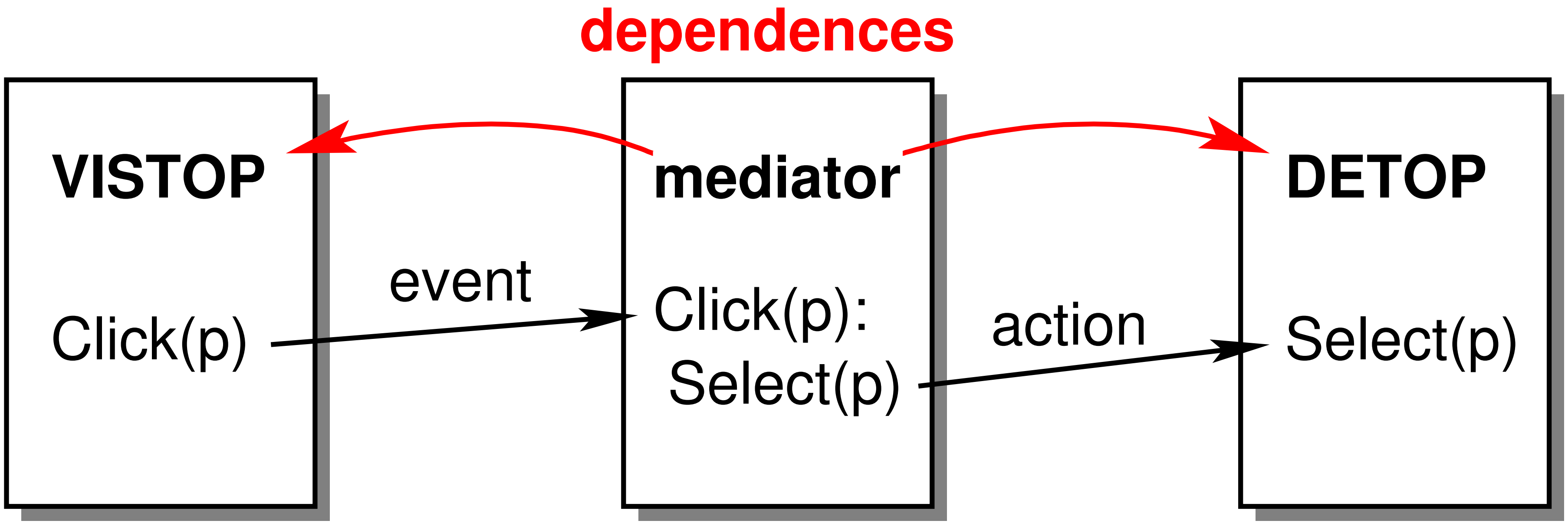
Mediators / interactors

- e.g. ToolBus, Manifold, CHAIMS, TIL



Mediators / interactors

- e.g. ToolBus, Manifold, CHAIMS, TIL

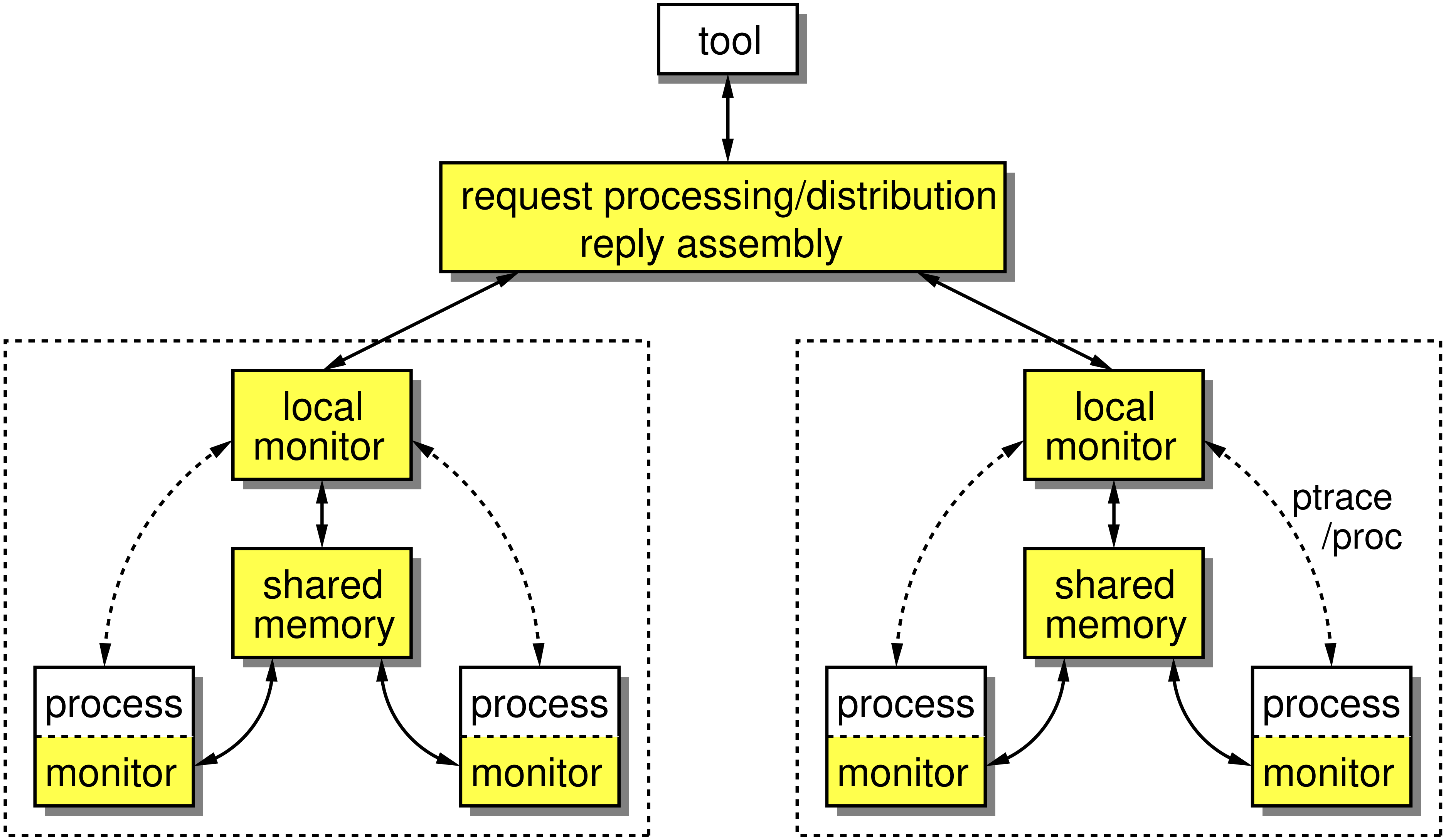


2. Cooperation of Software Components

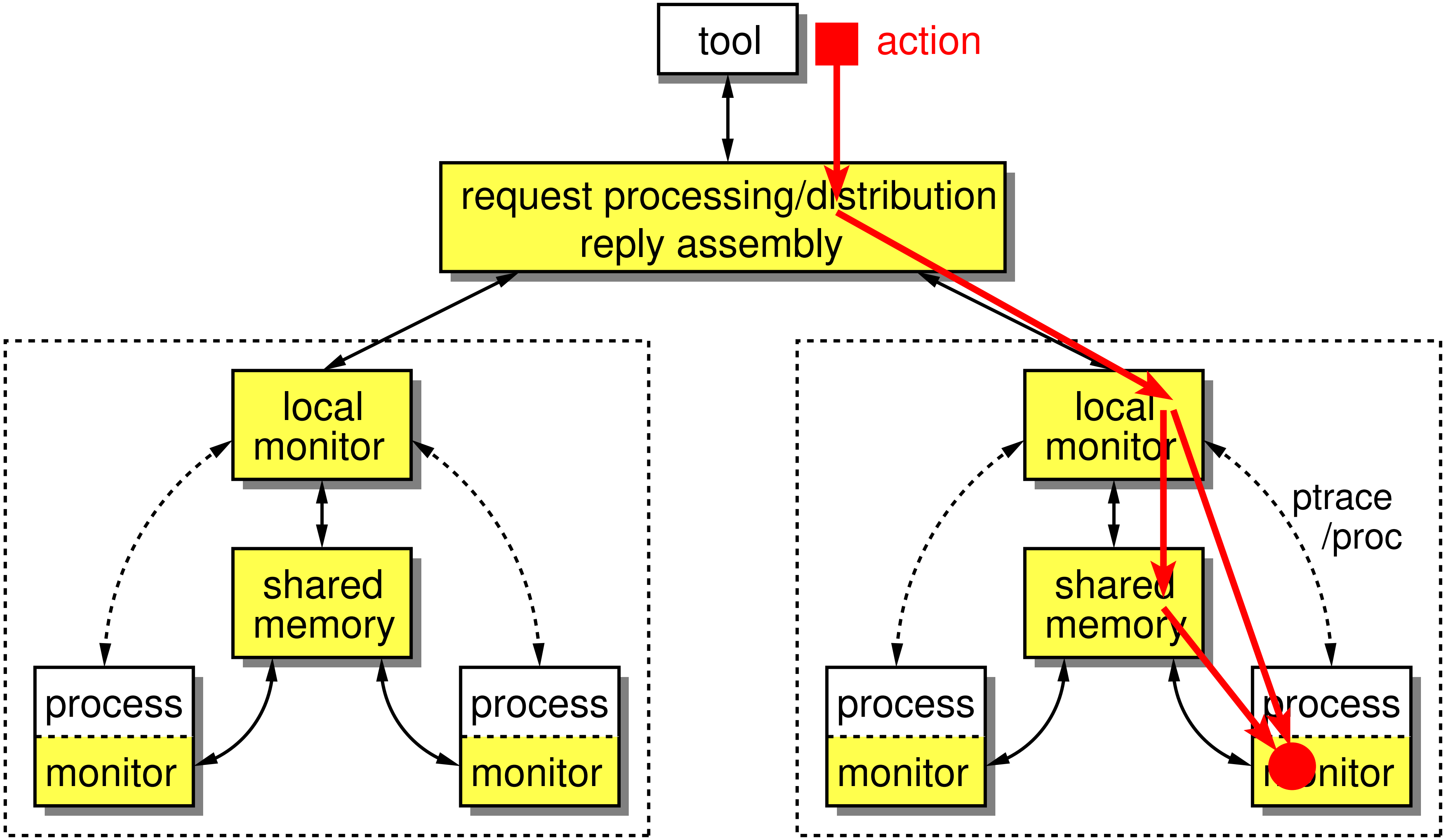
Problem:
components must already provide external interfaces

- completeness / appropriateness ?
  „[*anticipating*] *the events needed in an object's interface* [...] *is unreasonably hard*" (Sullivan, 1994)
- dependence on other software components
  - even with mediators (middleware!)
- components usually developed for stand-alone use
  - integration into environment requires modifications

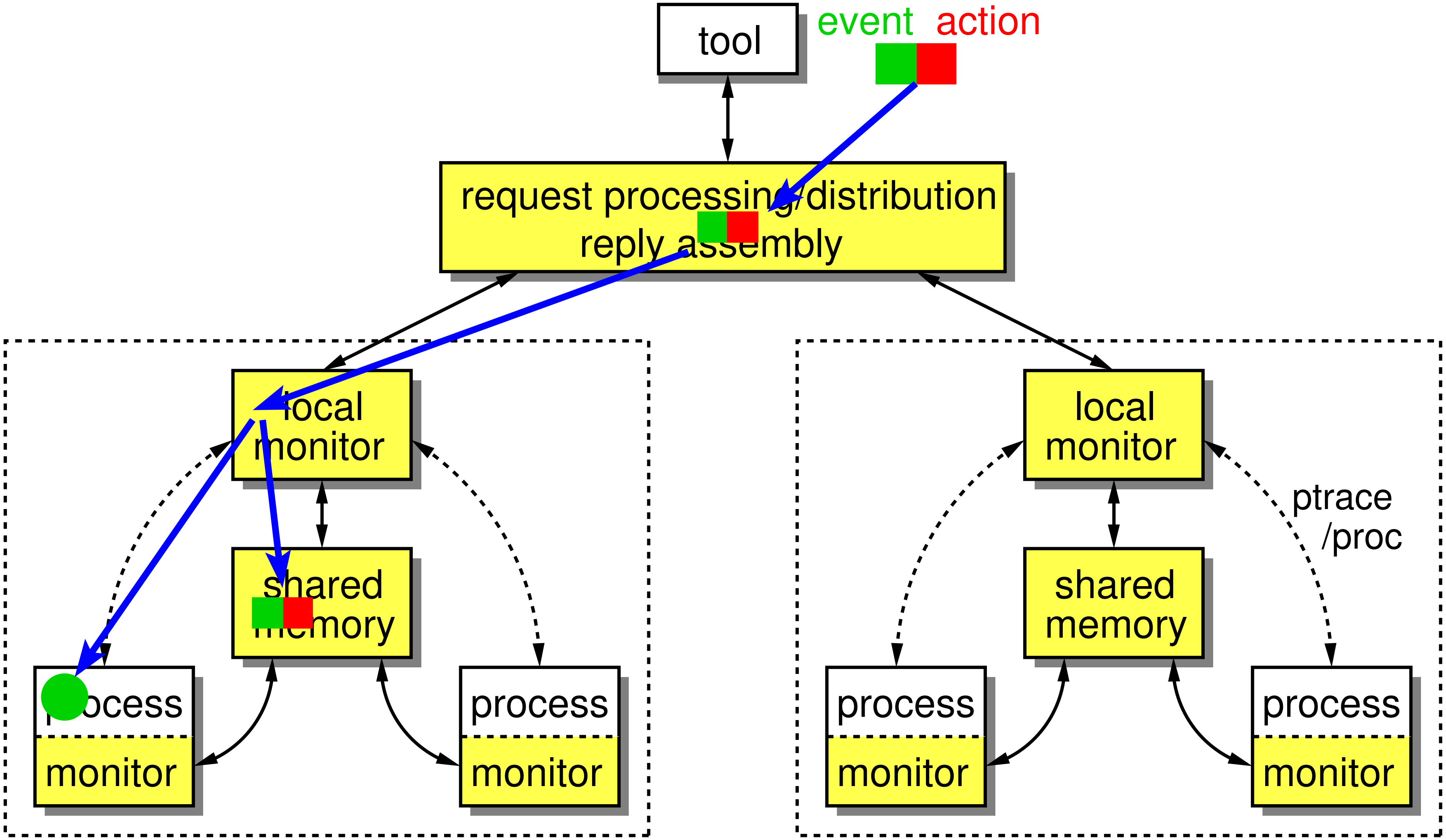
Operation of the OCM



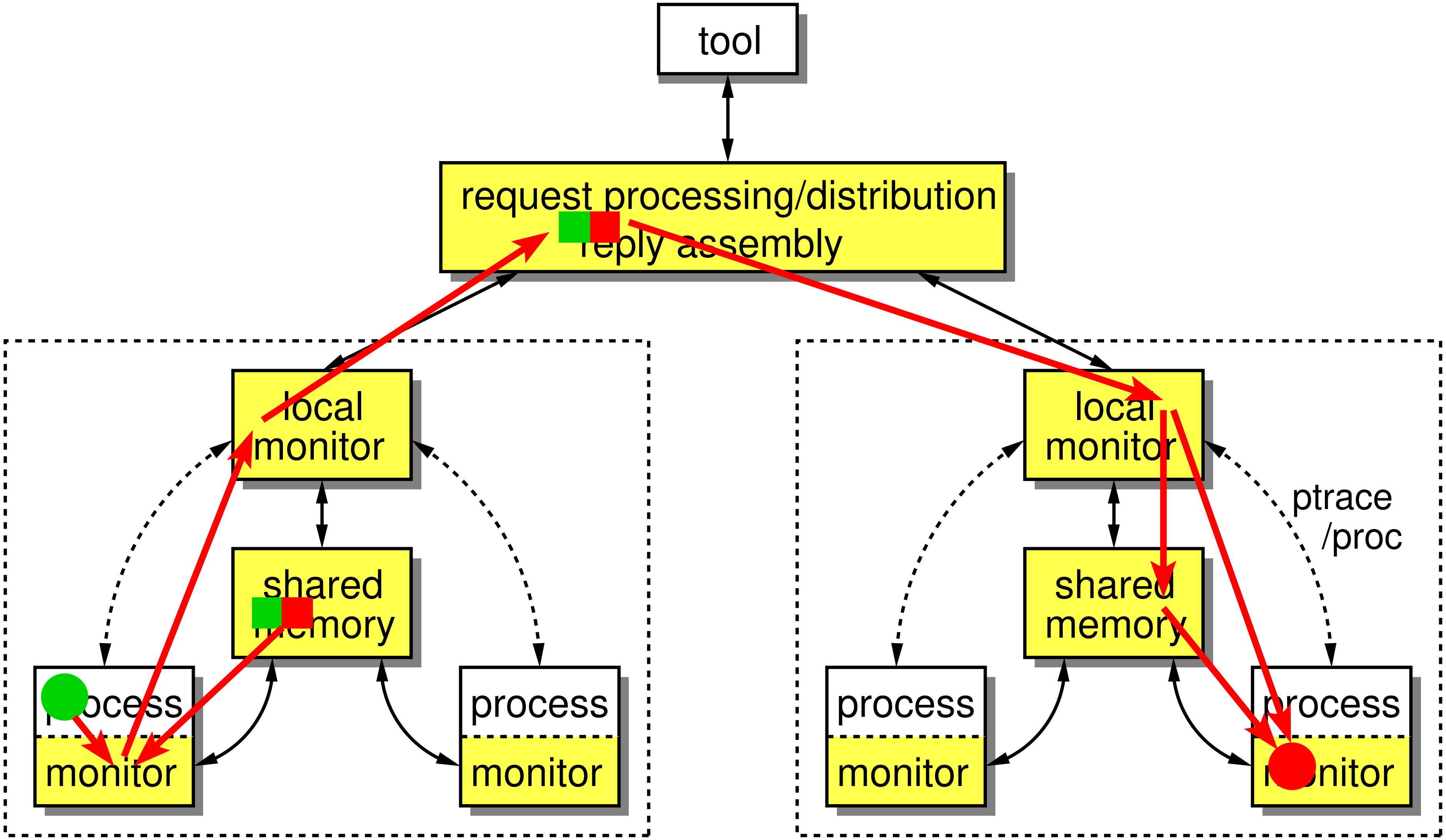
Operation of the OCM



Operation of the OCM



Operation of the OCM



Special properties of the OCM:

- monitoring of events in processes
- execution of actions in processes

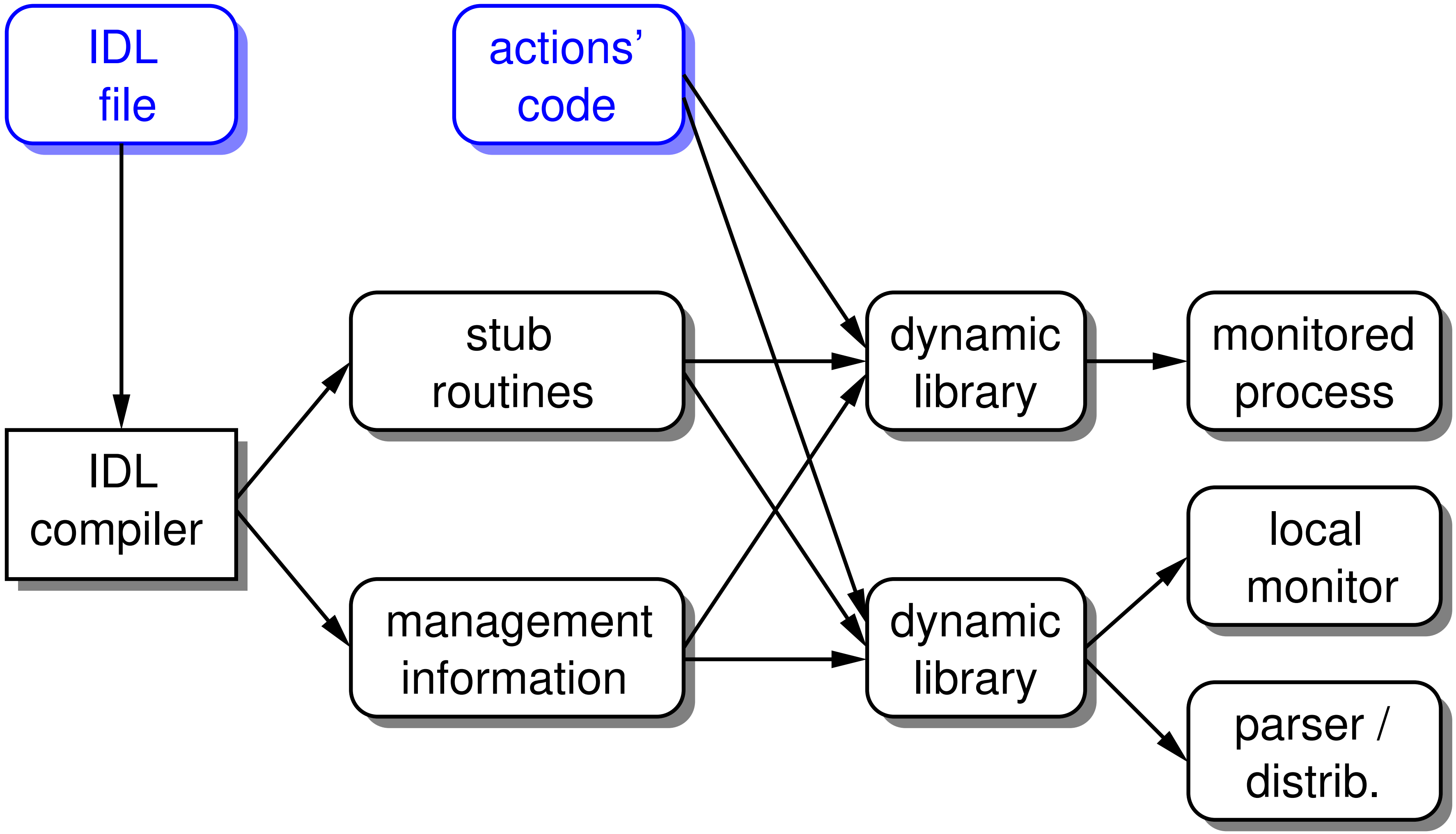
without a need to specially prepare these processes

- implemented via
  - dynamic instrumentation (*dyninst API*)
    - cooperation with B. P. Miller, UW Madison
  - inferior RPC
    - manipulation of stack and program counter
    - synchronized with a local event

Special properties of the OCM:

- OCM's interface is dynamically extensible by
  - new events
  - new actions
- allows to define the component's interfaces at runtime
- implemented via
  - interface description language
  - dynamic linking
  - inferior RPC

Including new actions

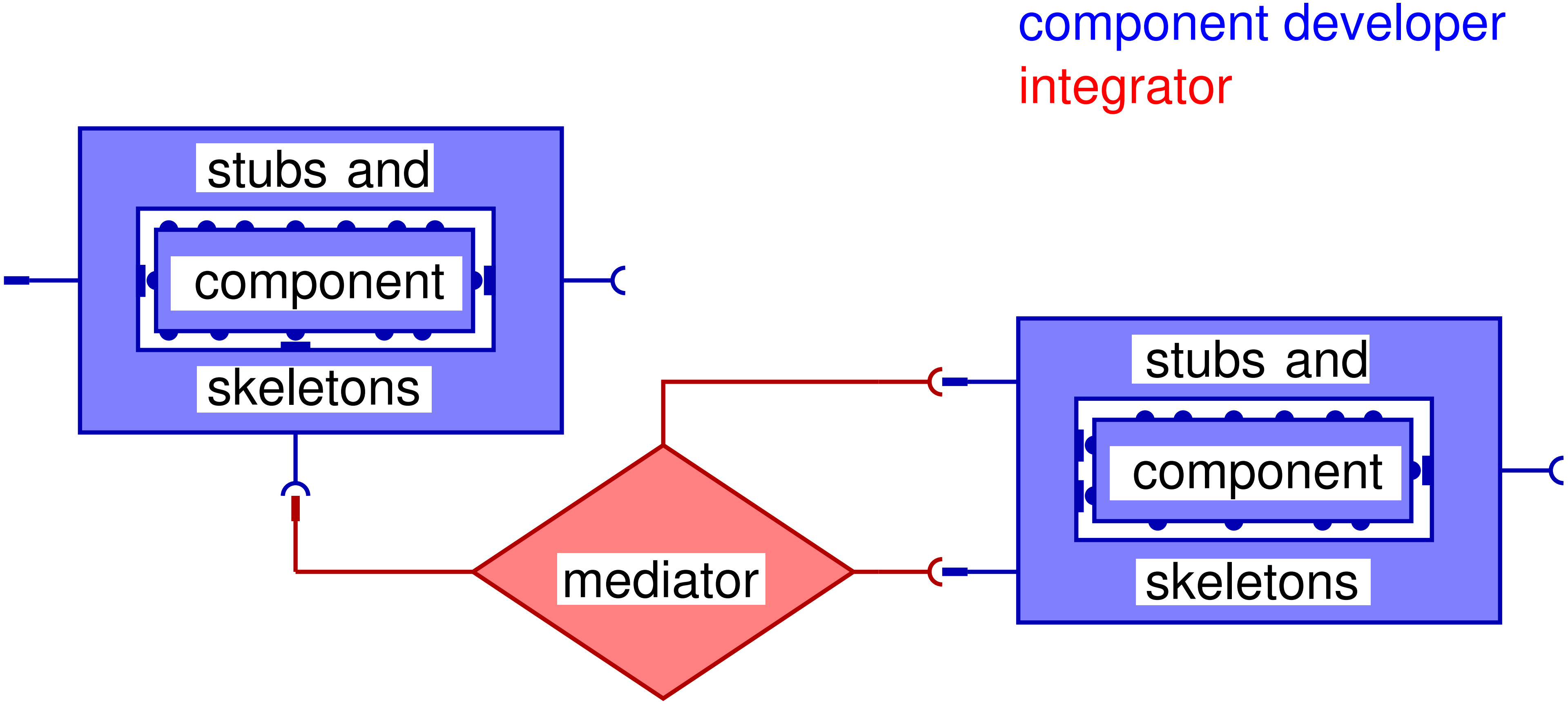


4. Result

Monitoring system $=$ dynamic middleware

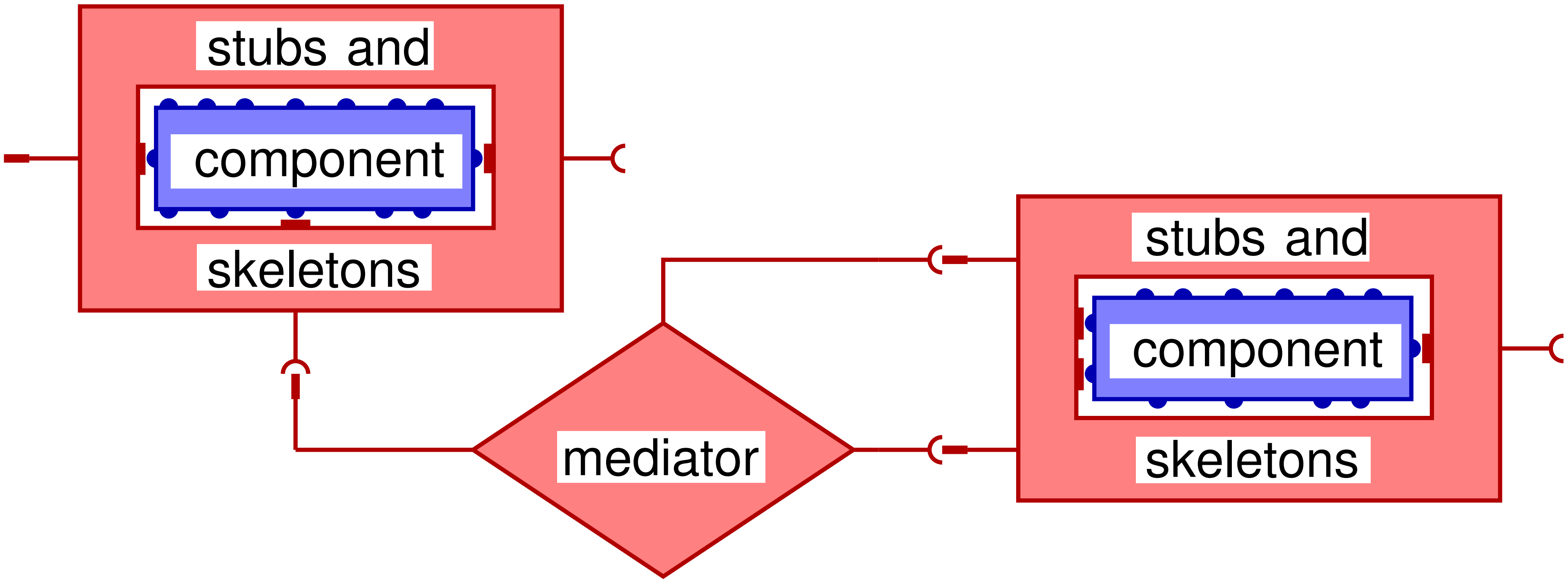
- mediator concept
  - behavioral rules can be programmed at runtime
- interface adapters dynamically linked into components
  - adapter provides external interface
  - adapter code can use all symbols visible for the linker
    - global variables / objects
    - global functions
    - public and protected methods
  - events: start / end of a function or method
  - actions: can use component like a programming library

Traditional approach



Dynamic linking with interface adapters

component developer
integrator



Flexible (re-)use of components:

- larger (internal) interface available
- interface is easily / efficiently adaptable

Simplified development of components:

- no need to explicitly export the interface
  - no unnecessary ’ballast’ in the code
  - no dependence on a concrete middleware

But: interfaces per se / documentation must be available!

5. Practical Experience

Prototype of tool environment

- checkpointing and automatic restart
- consistency VISTOP $\leftrightarrow$ DETOP
- 3 events + 10 actions
  - only one action with more complex adapter
  - no modification of the tools' source code (however, some modification was necessary for data integration)
- efficiency is sufficient, but should be improved (interaction needs 1.7 ms compared to 0.33 ms with PVM)

6. Conclusion

Cooperation of software components:

- mediator approach
- dynamic linking with interface adapters

Future work

- monitoring:
  separate middleware aspects and monitoring services
- cooperation:
  integrate concepts with a coordination language