Koordination verteilter Software-Komponenten durch Online-Monitorsysteme

Roland Wismüller

LRR-TUM
Technische Universität München

Agenda

1. Verteilte Komponenten
2. Umgebungen interoperabler Komponenten
   - ➥ konkret: interoperable Laufzeit-Werkzeuge
   - ➥ Problem: Verhaltensintegration
3. Monitoring verteilter Systeme
4. Koordination durch Monitorsysteme
5. Praktische Erfahrungen
6. Fazit

1. Verteilte Komponenten

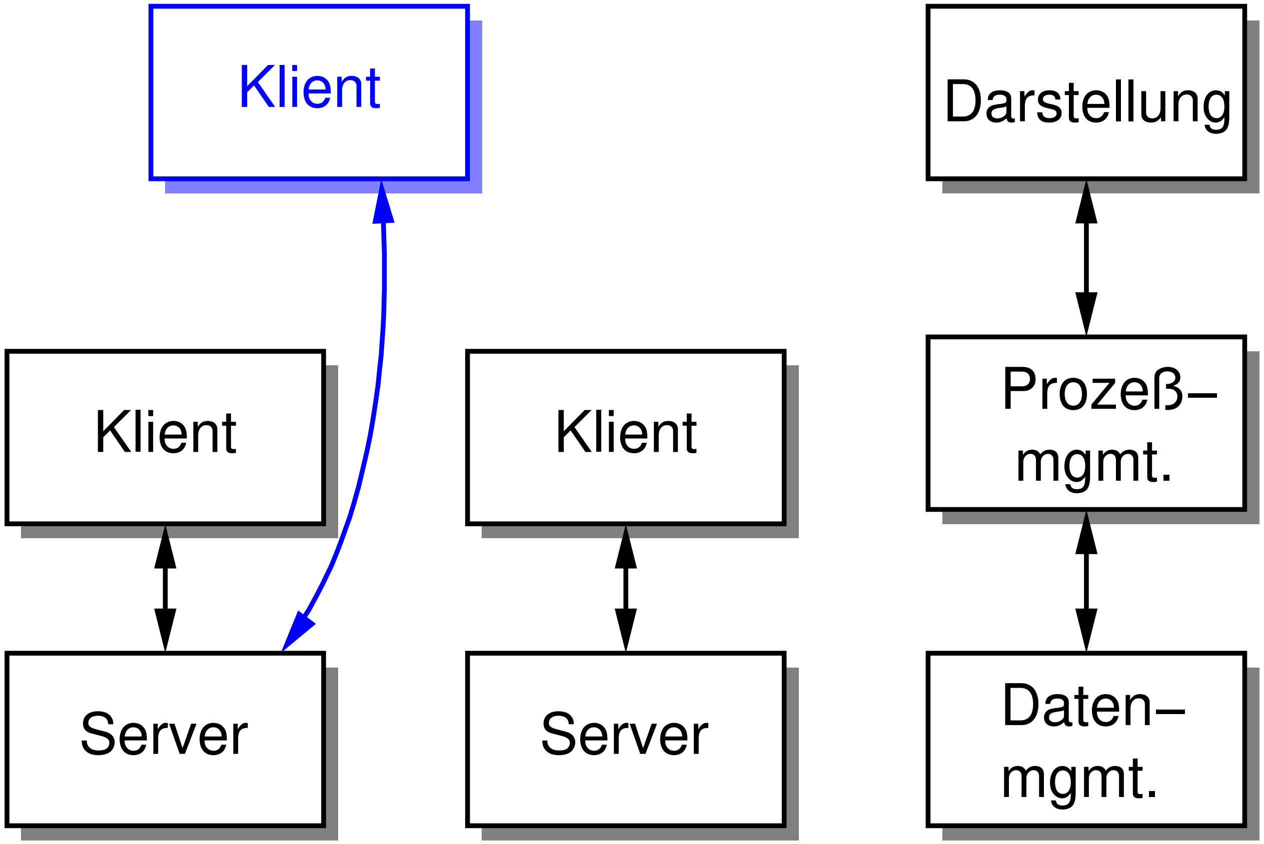
Ziele:

- Vereinfachte Entwicklung großer Systeme
- Wiederverwendbare Software-Komponenten
- Offene Systeme

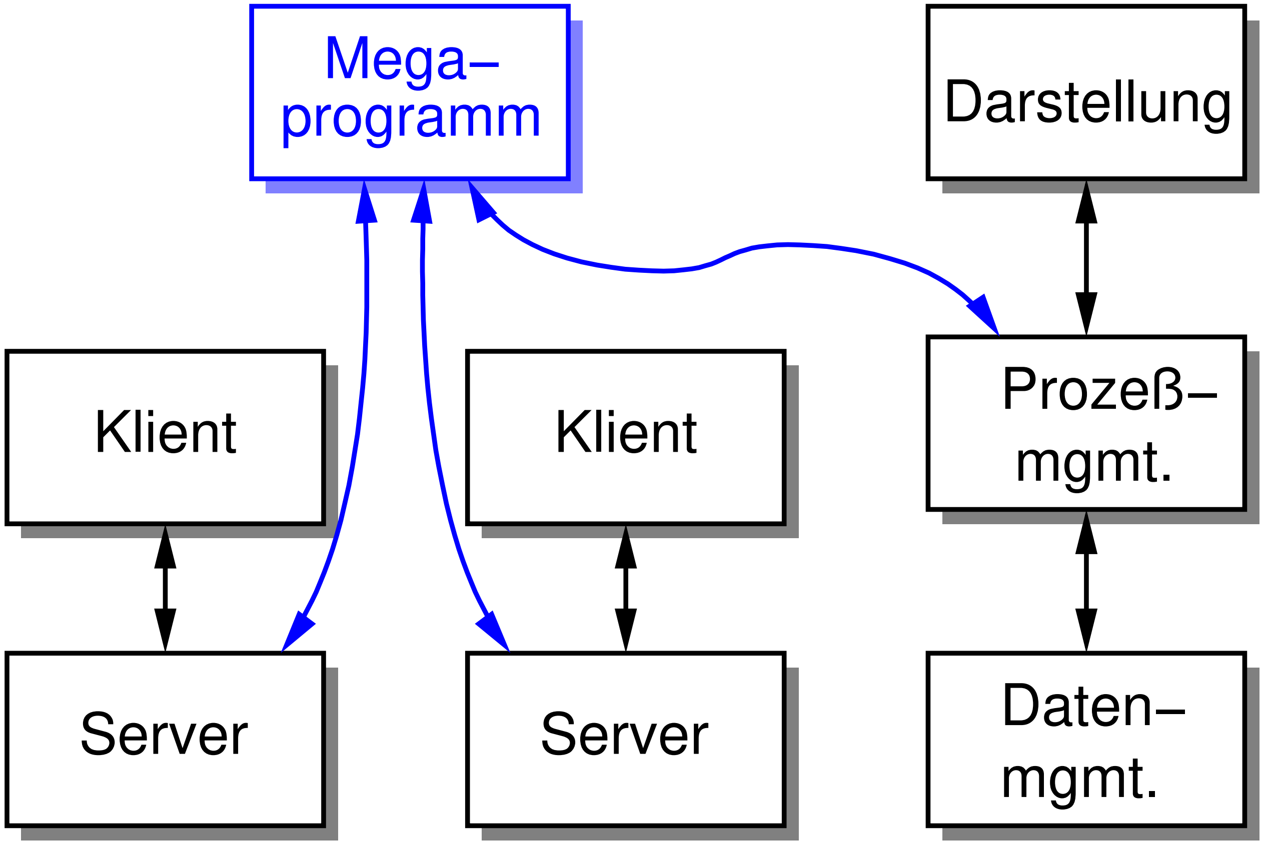
„Klassische“ Szenarien:

- *Client / server*
- *Three-tier* Architekturen
- *Megaprogramming*

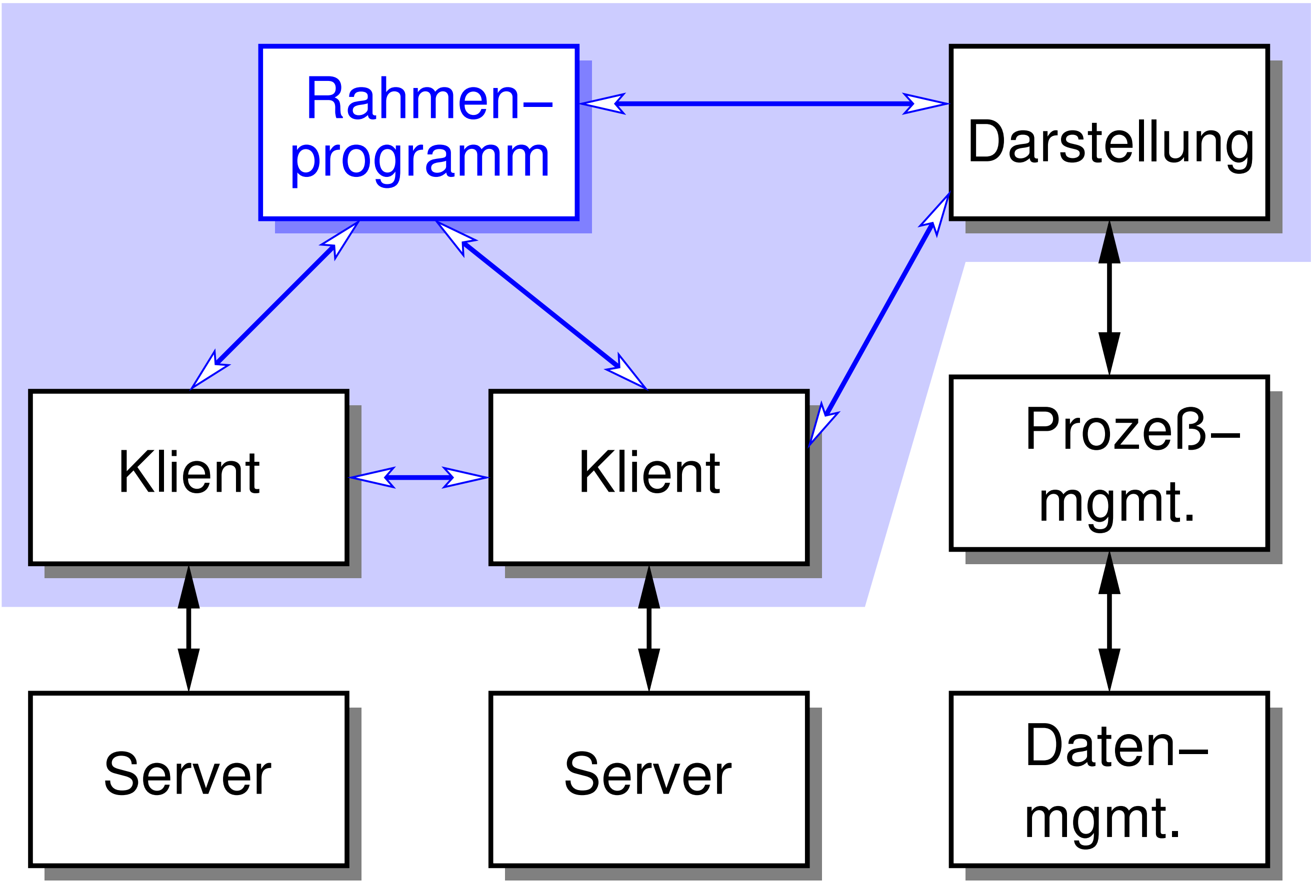
Client/server und *three-tier* Architekturen



Megaprogramming



Interoperable, interaktive Komponenten



Beispiele:

- CASE-Umgebungen
  - Editoren für verschiedene Sichten
  - Browser
  - Compiler
  - Debugger
- Werkzeuge zur Laufzeituntersuchung
  - Beobachtung/Steuerung unterschiedlicher Aspekte eines Programmablaufs

2. Umgebungen interoperabler Komponenten

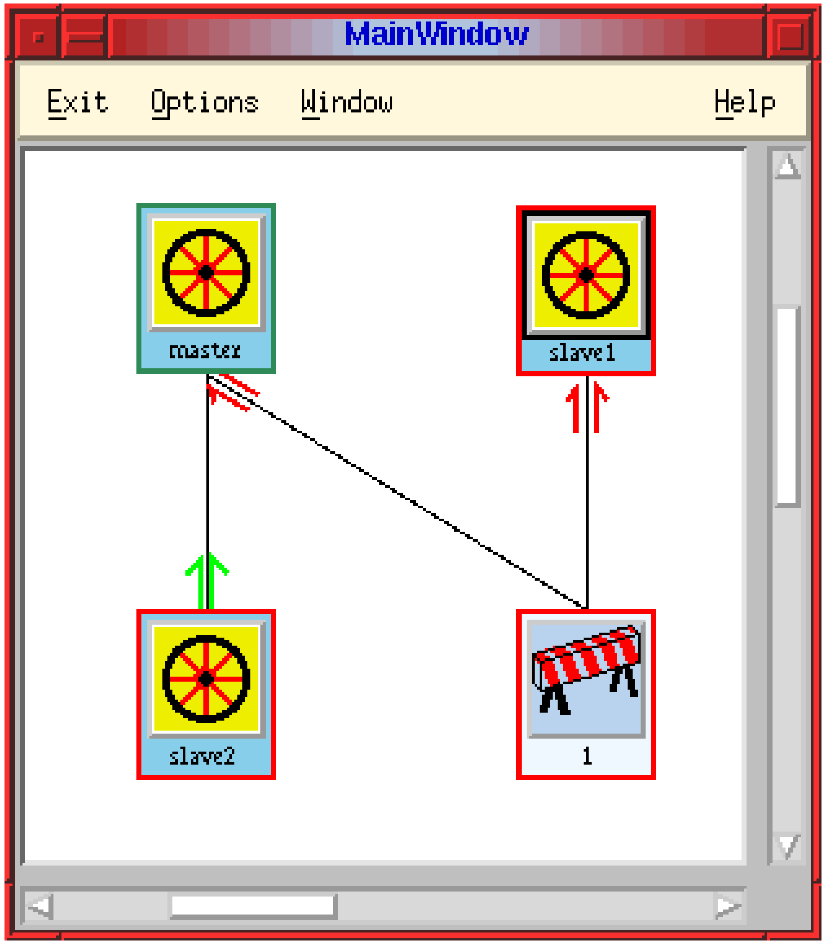
Fragestellung:

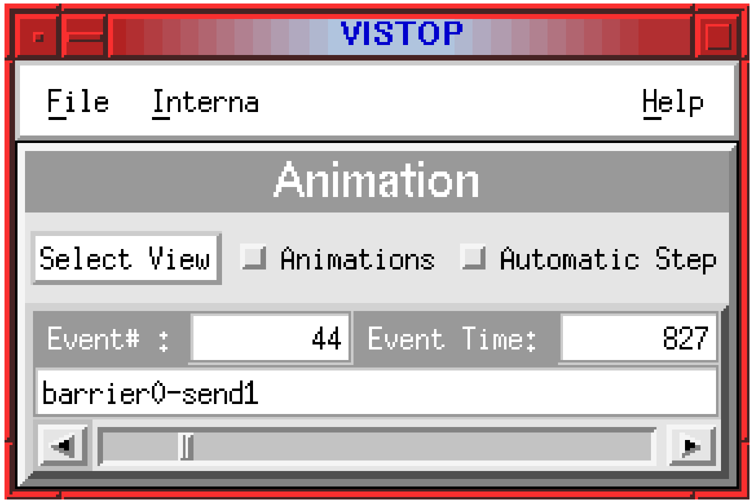
- Erstellung von Software-Umgebungen aus vorhandenen Komponenten
  - mit Benutzerinteraktion
  - aktiv, mit eigenständigem Verhalten
    (im Ggs. zu rein reaktiven Servern)
  - unabhängig voneinander entwickelt und nutzbar
  - keine Modifikation der Komponenten
- Unterstützende Methoden und Infrastrukturen

2.1. Eine Umgebung interoperabler Werkzeuge

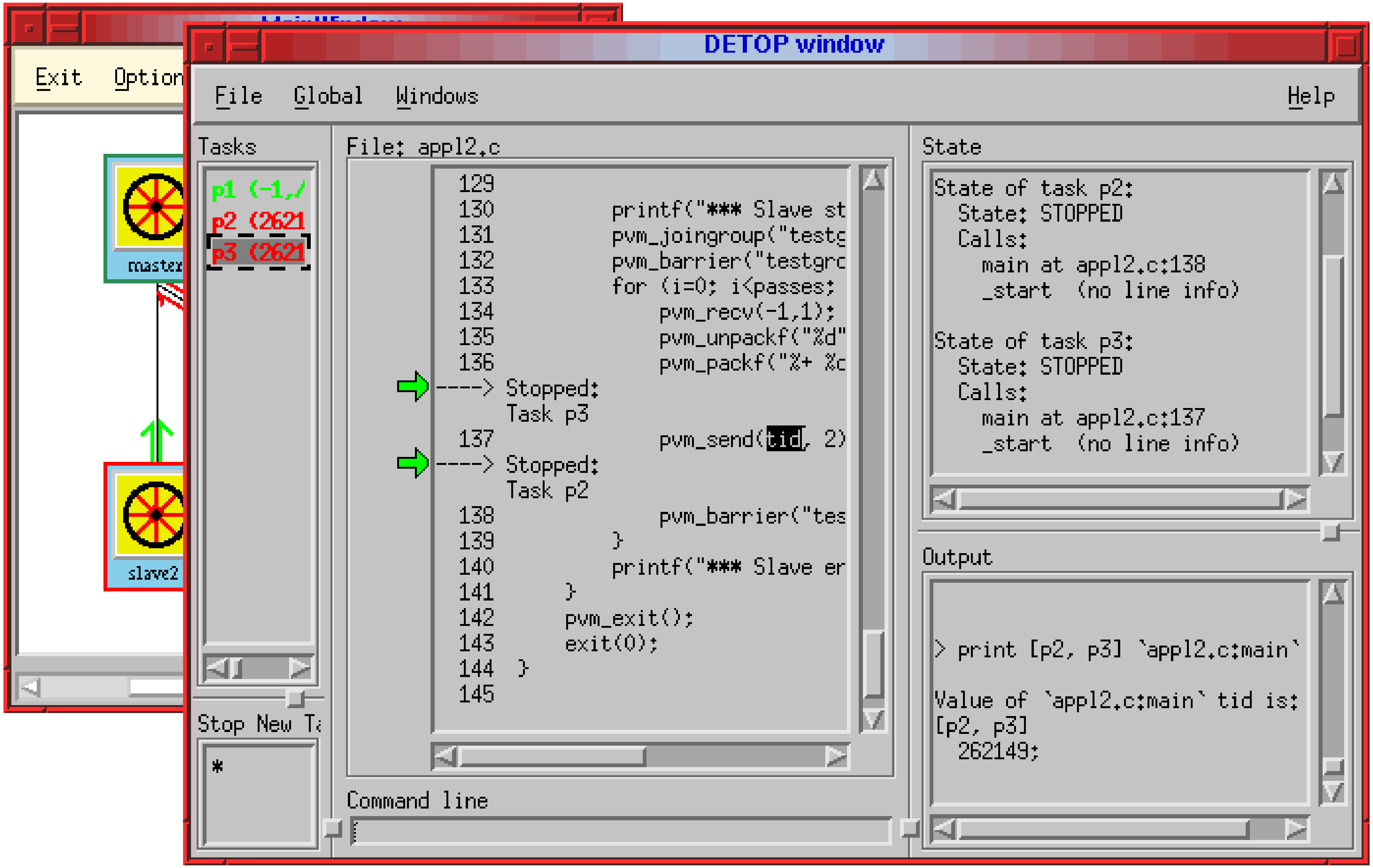
Motivation:

- Entwicklung von Online-Werkzeuge am LRR-TUM
  - Quellsprach-Debugger DETOP
  - Ablaufvisualisierer VISTOP
  - Leistungsanalyse, deterministische Ausführung, Sicherungspunkterstellung, ...
- Wunsch:
  - Kombination Visualisierer + Debugger
    - Überblick über Ablaufverhalten
    - detaillierte Untersuchung


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

2.1. Eine Umgebung interoperabler Werkzeuge



2.1. Eine Umgebung interoperabler Werkzeuge

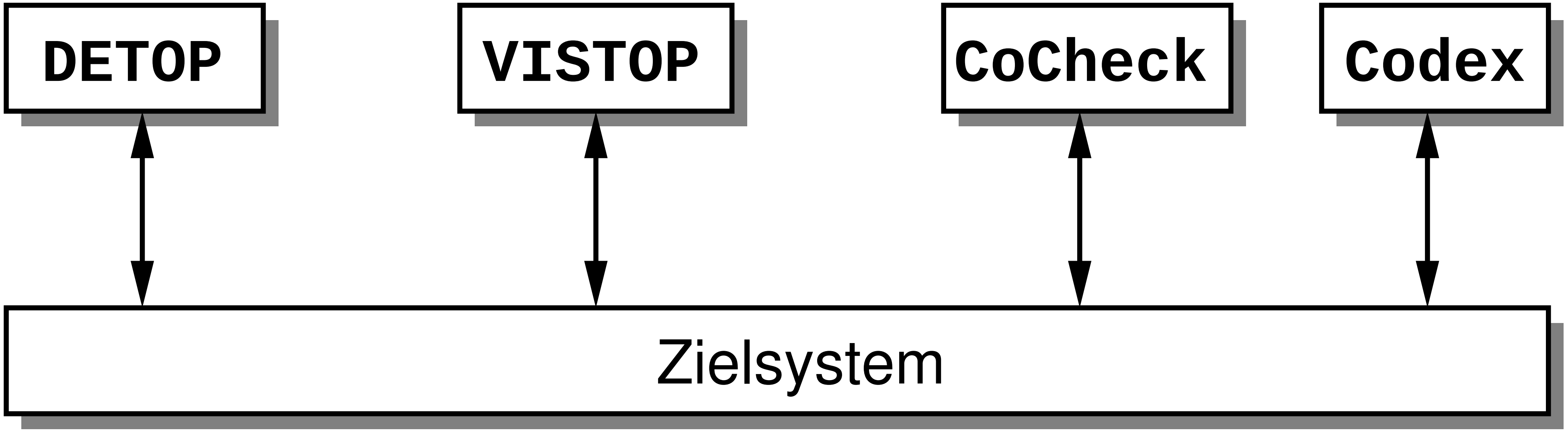
Schwierigkeit:

- VISTOP zeigt bereits vergangene Zustände
- ☞ Zur Untersuchung mit DETOP:
  Rücksetzen des Zielprogramms

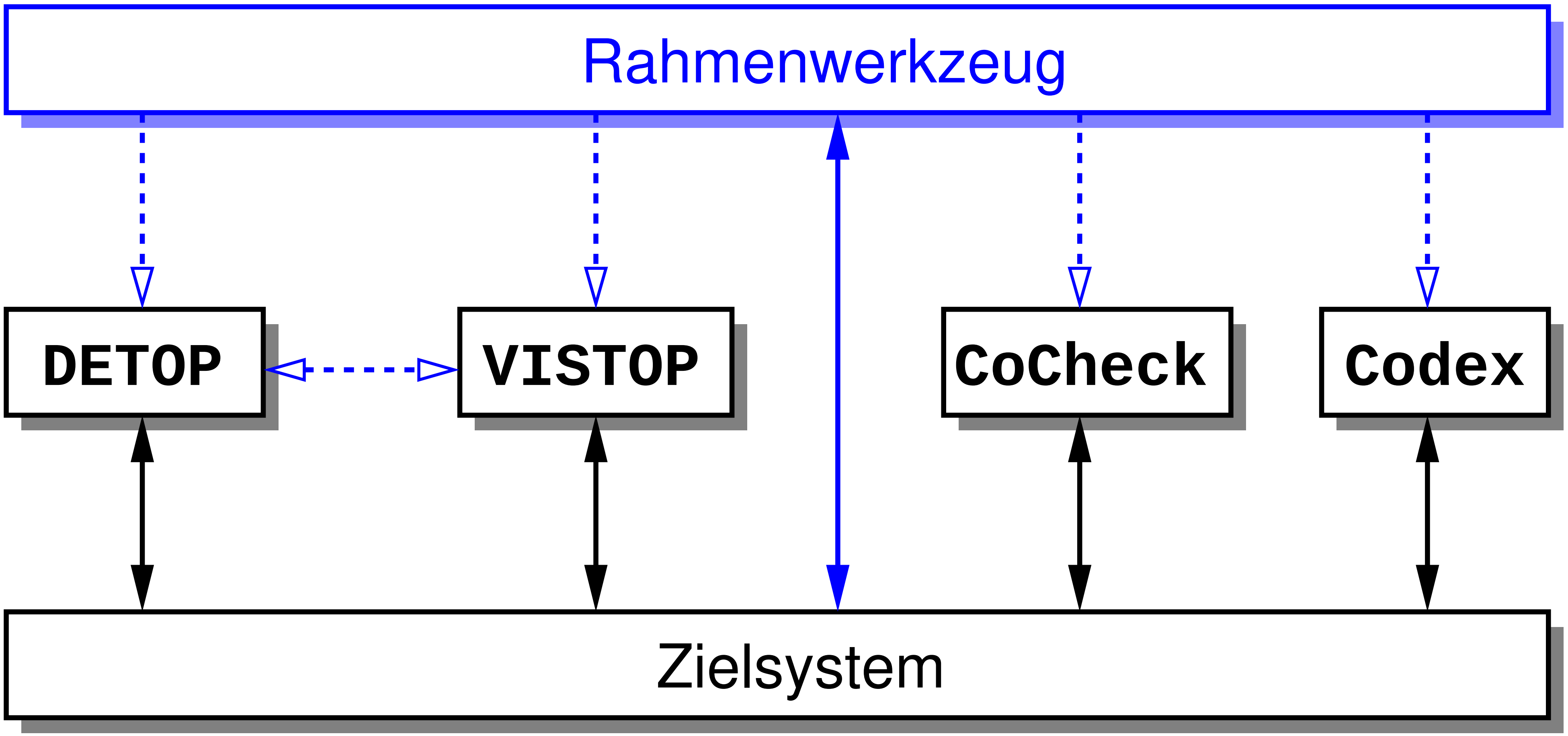
Vorgehensweise:

- Periodische Erstellung von Sicherungspunkten
- Rücksetzen auf geeigneten Sicherungspunkt
- Automatische Weiterführung zum Zielzustand
  - deterministische Ausführung notwendig

Aufbau der Werkzeugumgebung



Aufbau der Werkzeugumgebung



Konfliktfreier Zugriff auf gemeinsame Daten

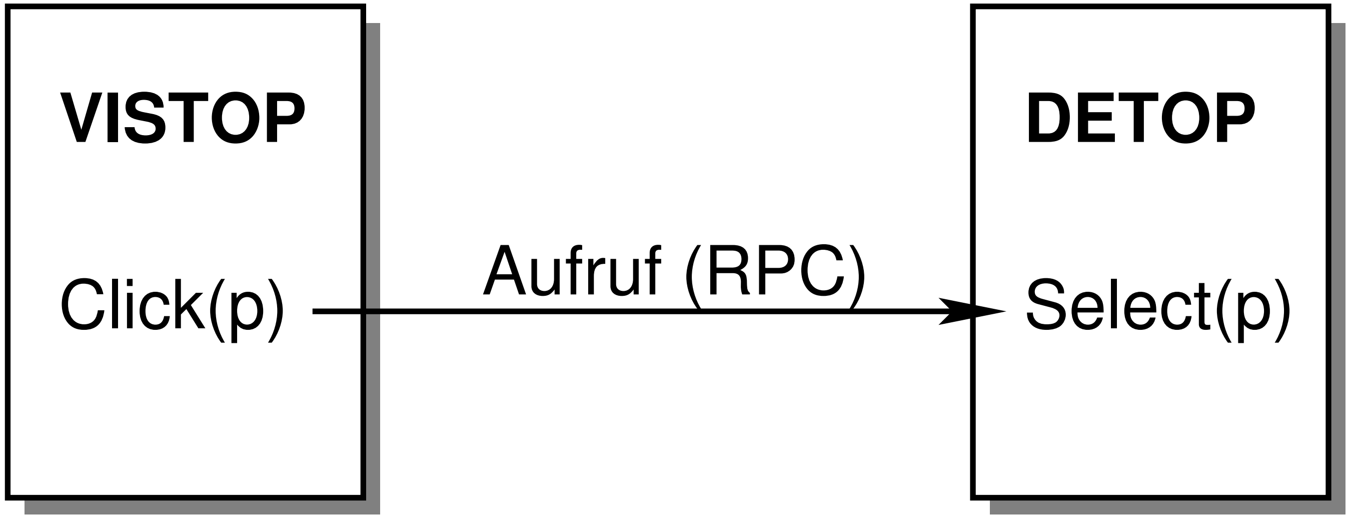
- ☞ Datenintegration
- ➥ hier: gemeinsames Monitorsystem (OMIS/OCM)
  - ➥ Erweiterungen zur Erkennung und Behandlung von Konflikten
    - ➥ Konfliktursache: Verändern des Zielsystemzustands
  - ➥ Ziel: Werkzeuge benötigen kein Wissen übereinander
- ➥ (bei CASE-Umgebungen: gem. Datenbanksystem, z.B. PCTE)

Kooperation zwischen den Komponenten

- Verhaltensintegration (Steuerungs-/Präsentationsintegration)
- Beispiele:
  - konsistente Informationsdarstellung
  - synchronisierte Auswahl
  - *click back*
- Im weiteren Sinne auch:
  - Realisierung komplexer Funktionen durch Rahmenprogramm

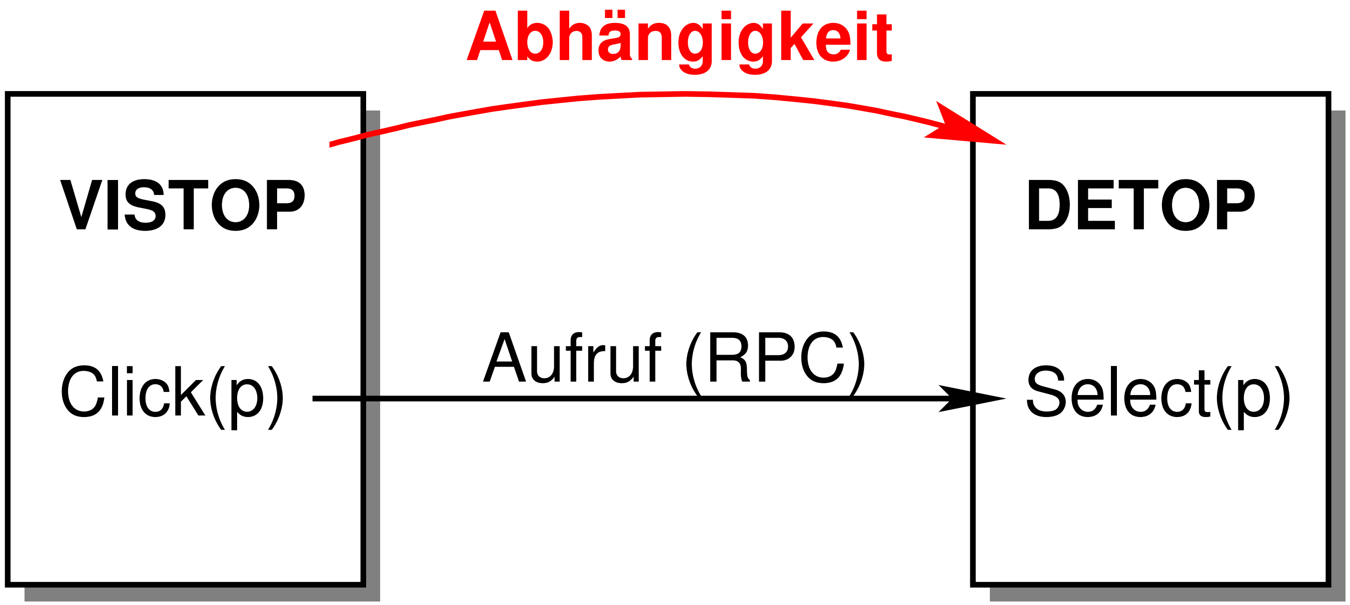
Feste Verdrahtung

- Z.B. RPC, CORBA, PCTE



Feste Verdrahtung

- Z.B. RPC, CORBA, PCTE

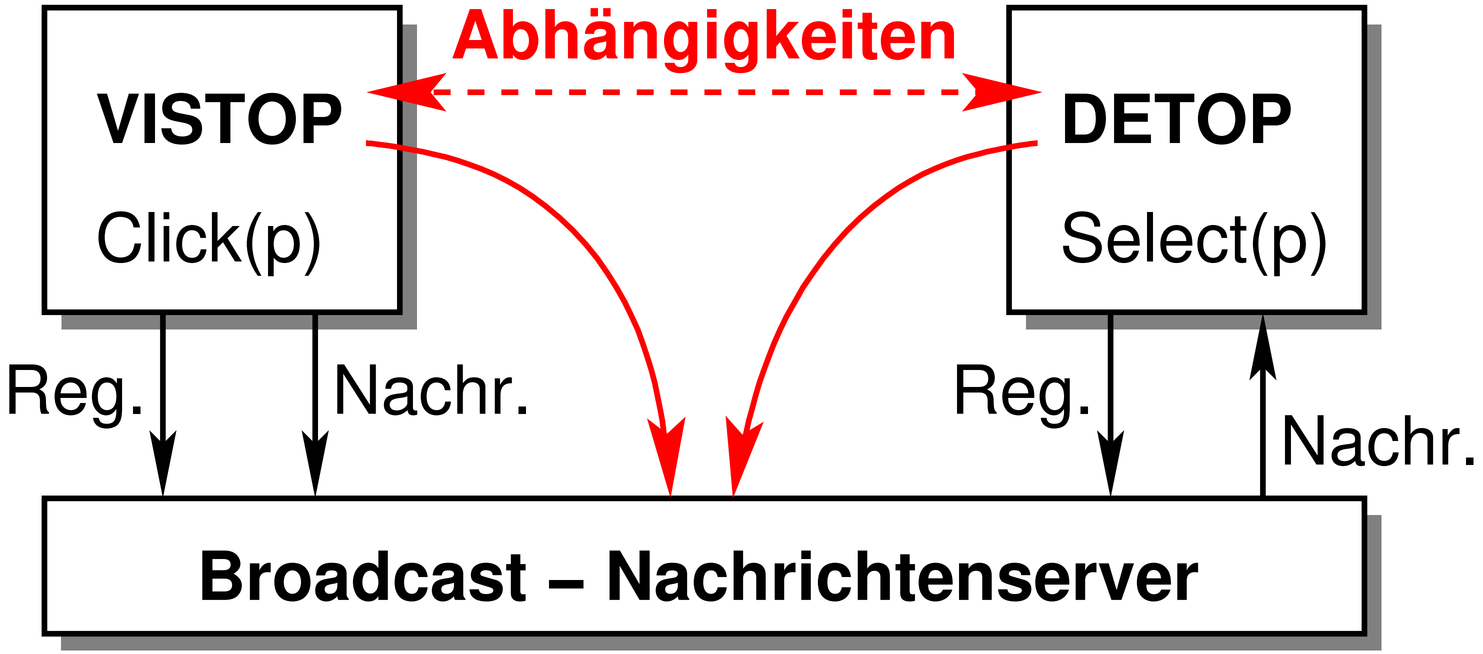


Broadcast-Nachrichtenserver

- Z.B. ToolTalk, SoftBench

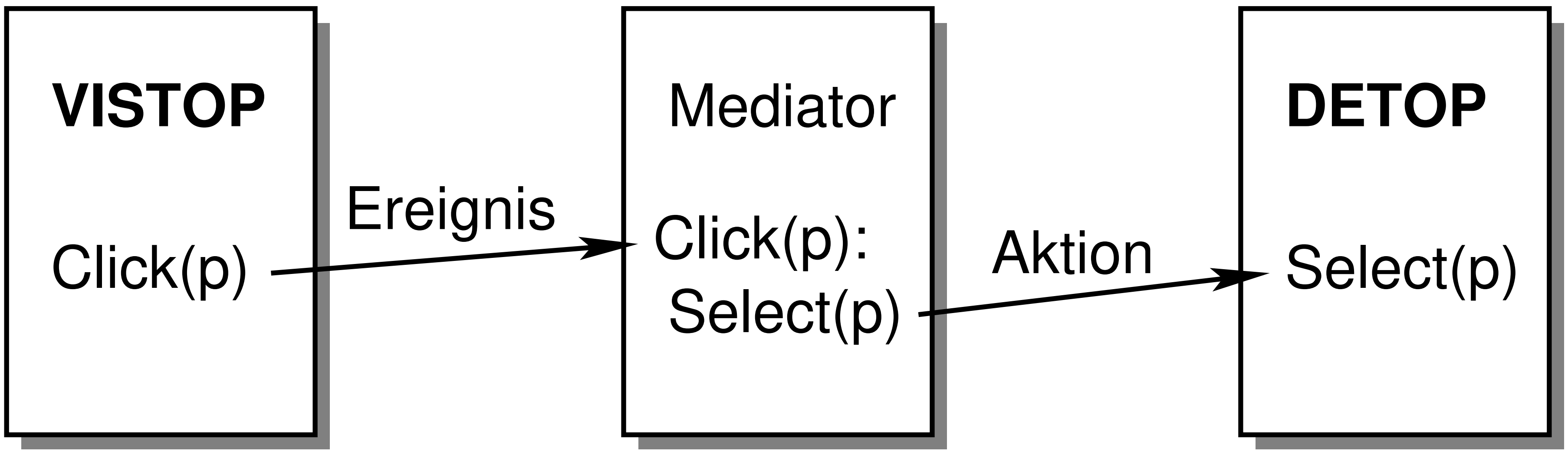
Broadcast-Nachrichtenserver

- Z.B. ToolTalk, SoftBench



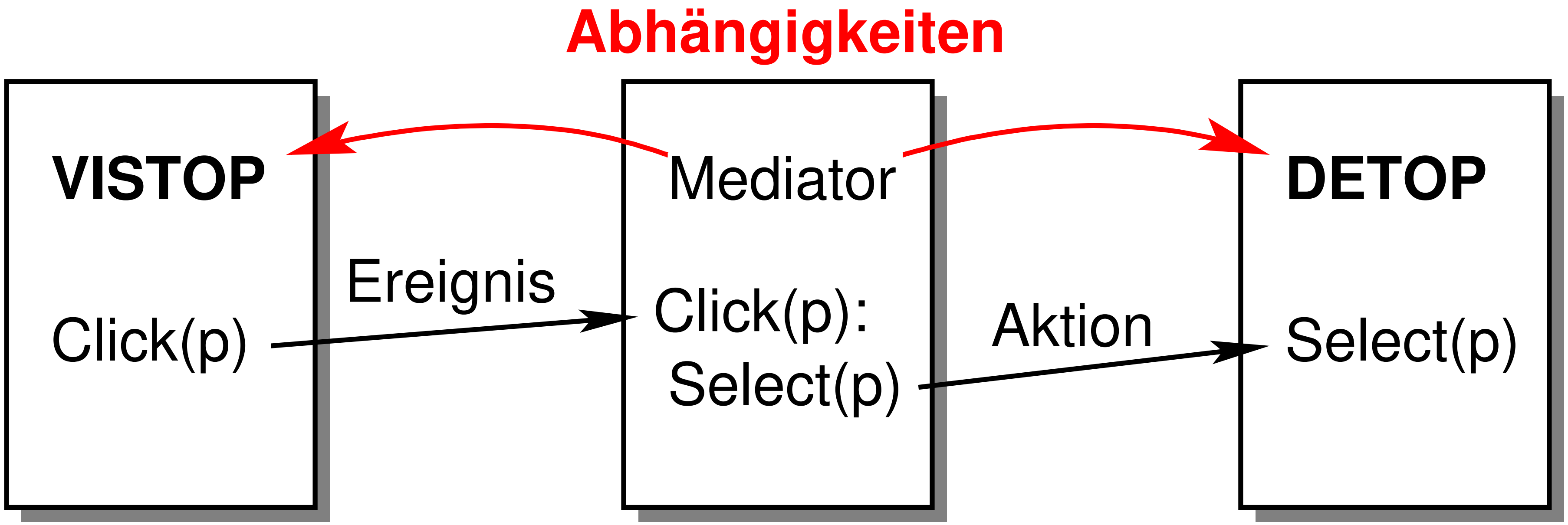
Mediatoren / Interaktoren

- Z.B. ToolBus, Manifold, CHAIMS, TIL



Mediatoren / Interaktoren

- Z.B. ToolBus, Manifold, CHAIMS, TIL



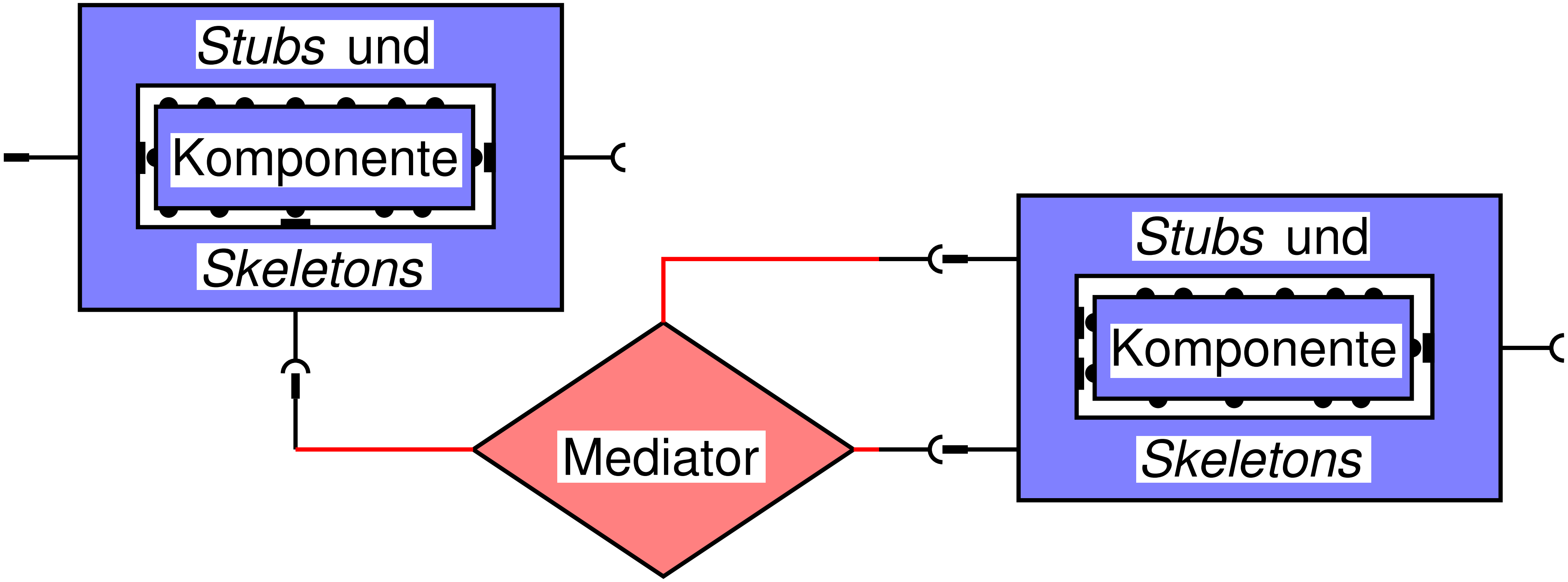
Komponenten müssen a-priori Schnittstellen bereitstellen

- Vollständigkeit / Angemessenheit?
  „[*anticipating*] *the events needed in an object's interface* [...] *is unreasonably hard*" (Sullivan, 1994)
- Abhängigkeit von Drittsoftware
  - selbst bei Mediatoren (Middleware!)
- Komponenten meist nur für isolierte Nutzung entwickelt
  - Einsatz in Umgebung erfordert Modifikation

☞ Dynamisches Einbinden der Middleware-Schnittstellen?

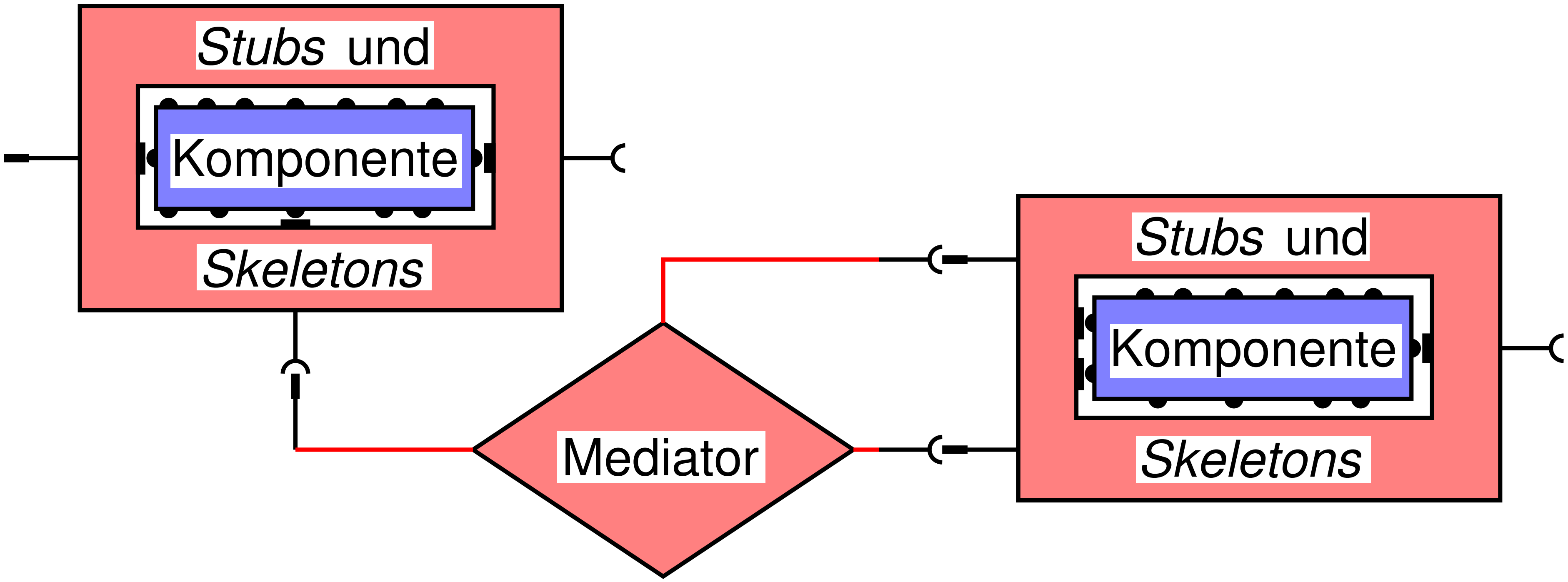
Traditionelles Vorgehen

Komponentenersteller
Integrierer



Dynamisches Einbinden von Schnittstellen

Komponentenersteller
Integrierer



2.5. Diskussion

Einfachere Entwicklung von Komponenten:

- Kein expliziter Export von Schnittstellen
- Kein unnötiger 'Ballast' im Code
- Keine Abhängigkeit von einer Middleware

Flexiblere Nutzung von Komponenten:

- Mächtigere Schnittstelle nutzbar
- Anpassung der Schnittstelle möglich

Schnittstelle an sich / Dokumentation muß vorhanden sein!

OMIS: *On-line Monitoring Interface Specification*

- Motivaton:
  Werkzeuge benötigen Komponente zur
  - Online-Beobachtung
  - Steuerung

  paralleler und verteilter Systeme
- Ziele:
  - universelle Monitoring-Schnittstelle
  - plattform- und werkzeugunabhängig
  - effizient und skalierbar
  - gleichzeitig durch mehrere Werkzeuge nutzbar

OMIS: *On-line Monitoring Interface Specification*

- Konzepte:
  - objektbasiertes Modell des Zielsystems
  - programmierbare Ereignis/Aktions-Relationen

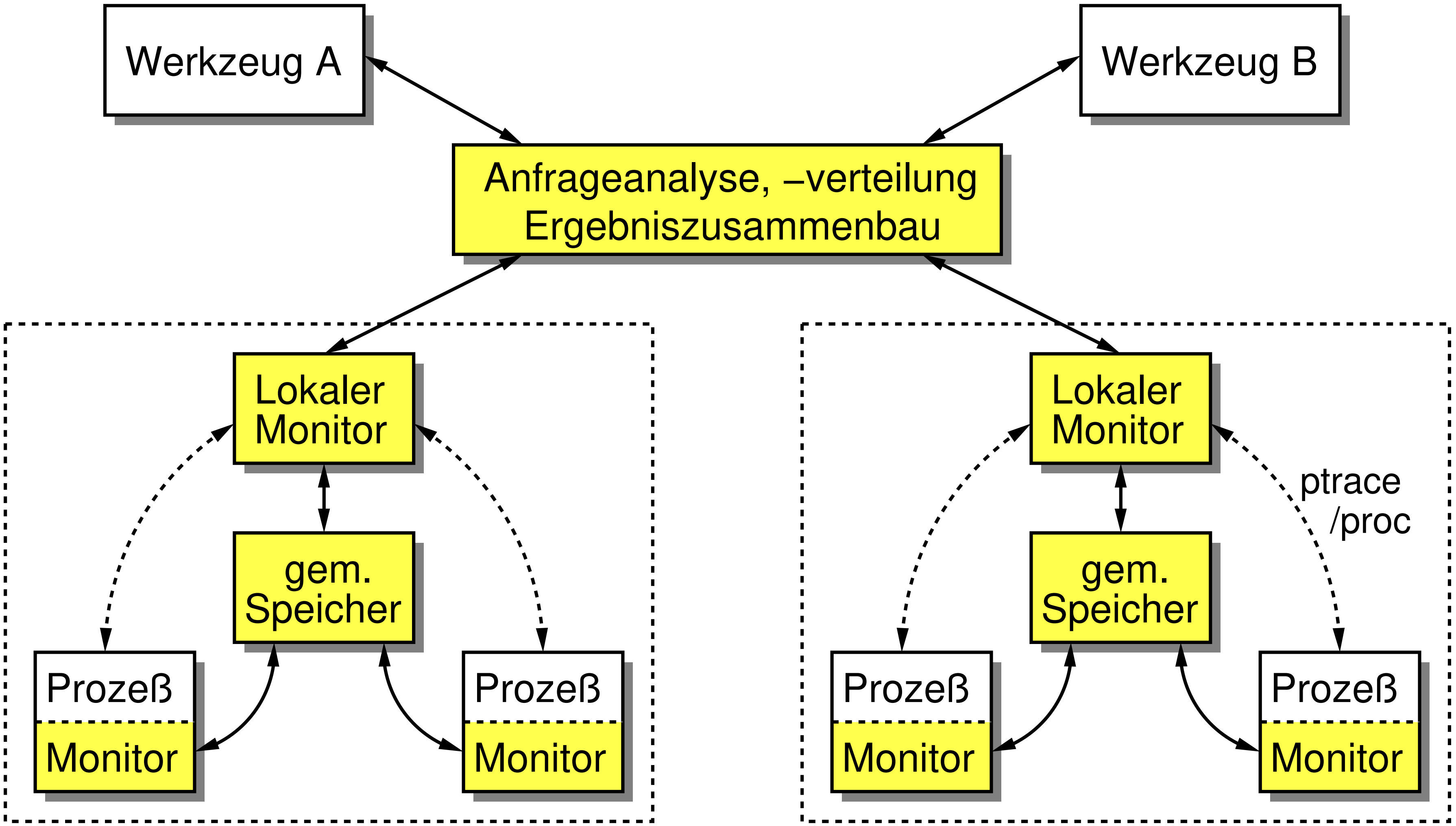
```
thread_started_lib_call([p_1],``pvm_send'') :
    thread_stop([$node])
    thread_get_backtrace([$thread],0)
```

  - Objektmengen
  - Ortstransparenz
  - Erweiterbarkeit

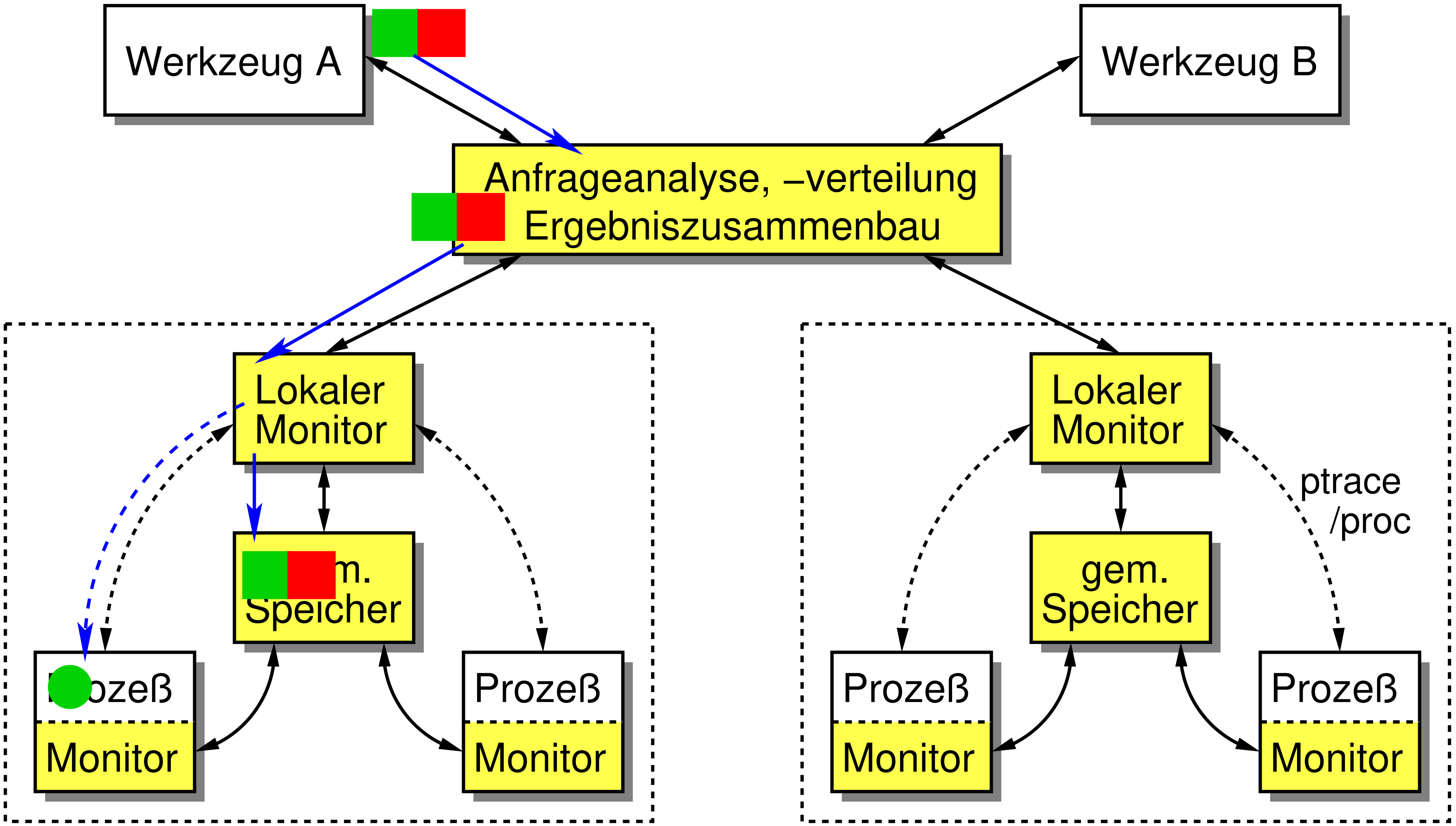
OCM: *OMIS Compliant Monitoring system*

- Realisierung der OMIS-Schnittstelle
- Zielsystem: Netze von Arbeitsplatzrechnern
  - Solaris, Irix, Linux, AIX, ...
- Unterstützung für PVM und MPI
  - MPI: Kooperation mit ICS-AGH, Krakau (Marian Bubak)

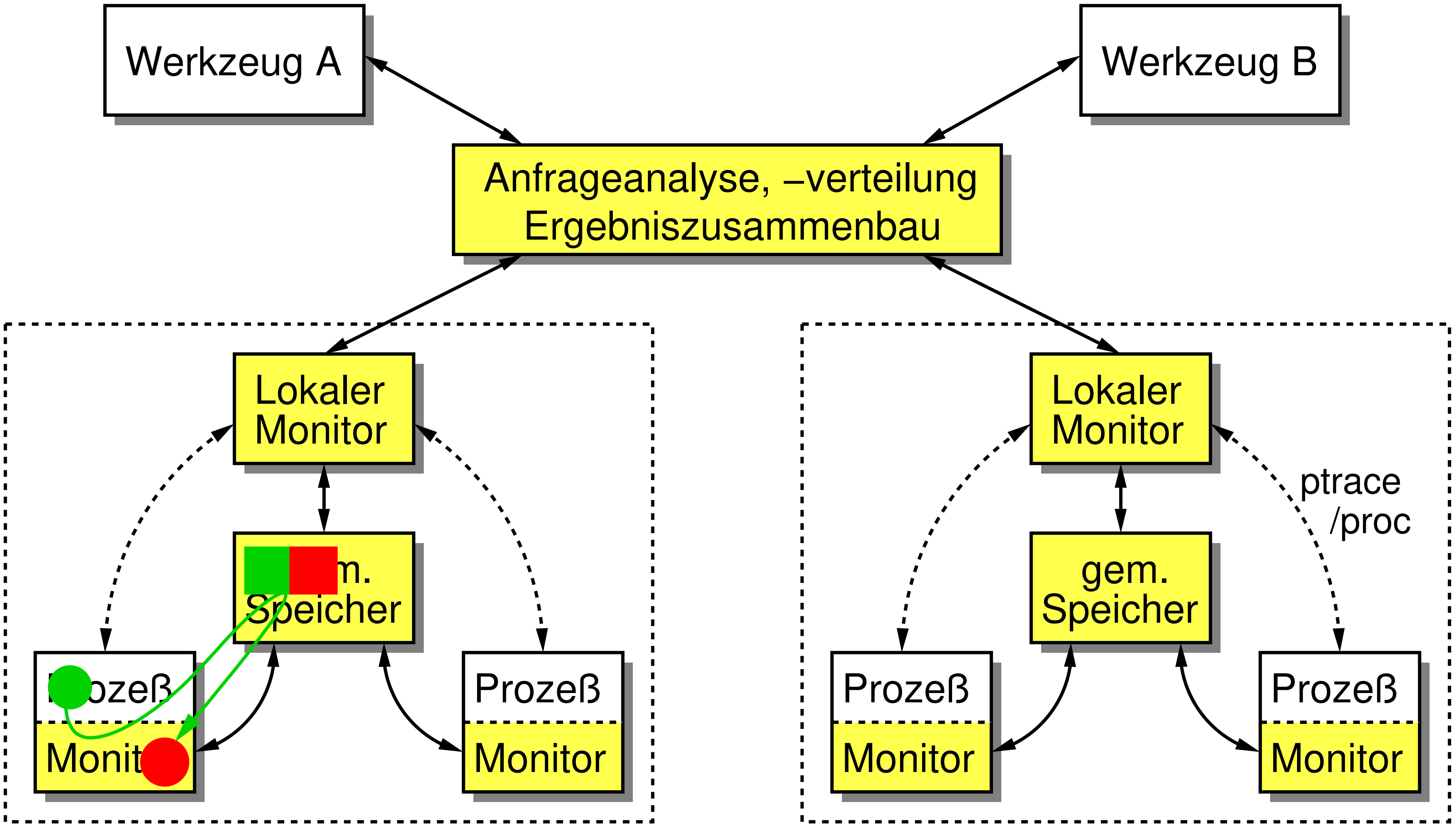
Aufbau des OCM



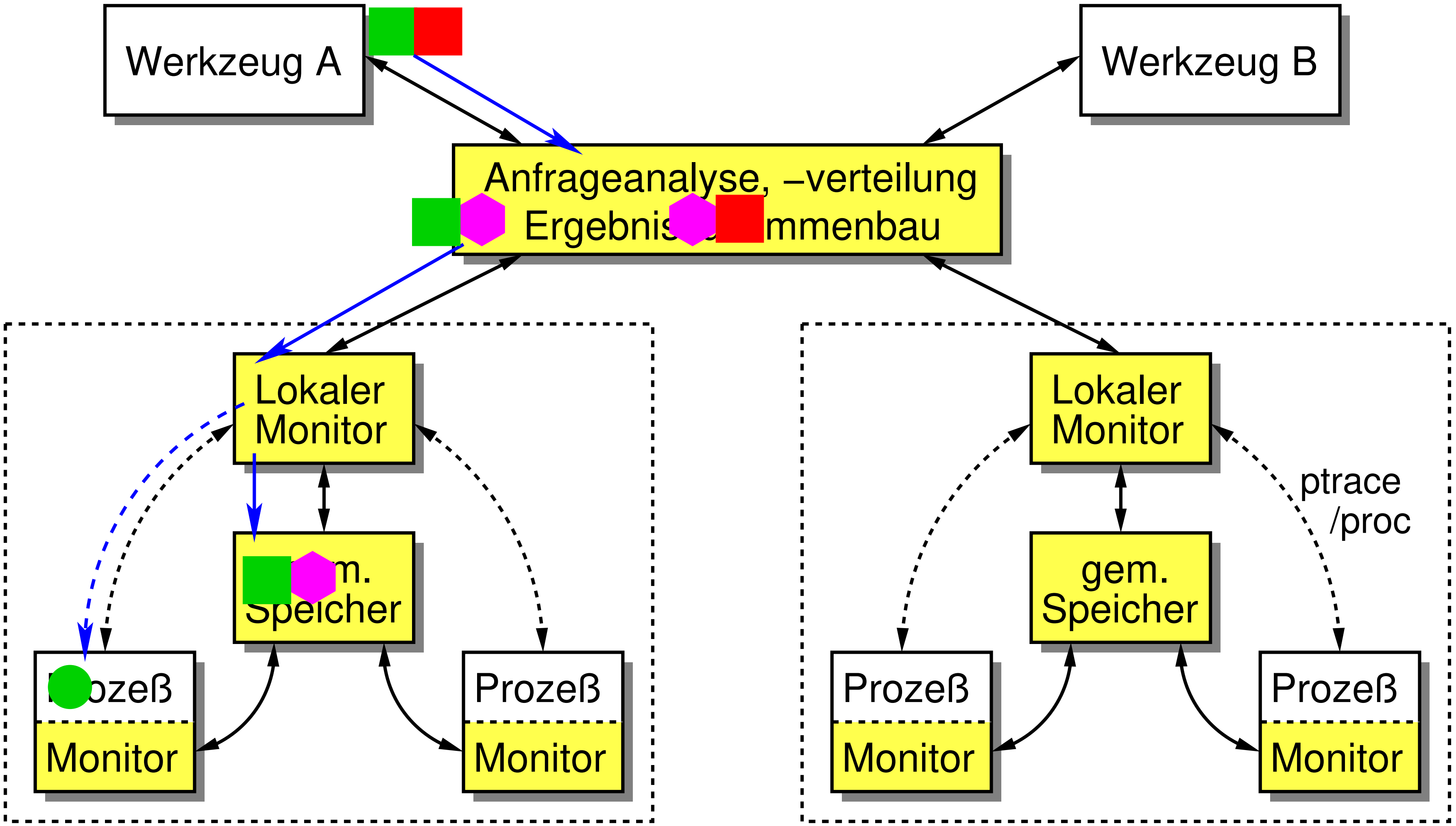
Anfrageverteilung und -abarbeitung im OCM



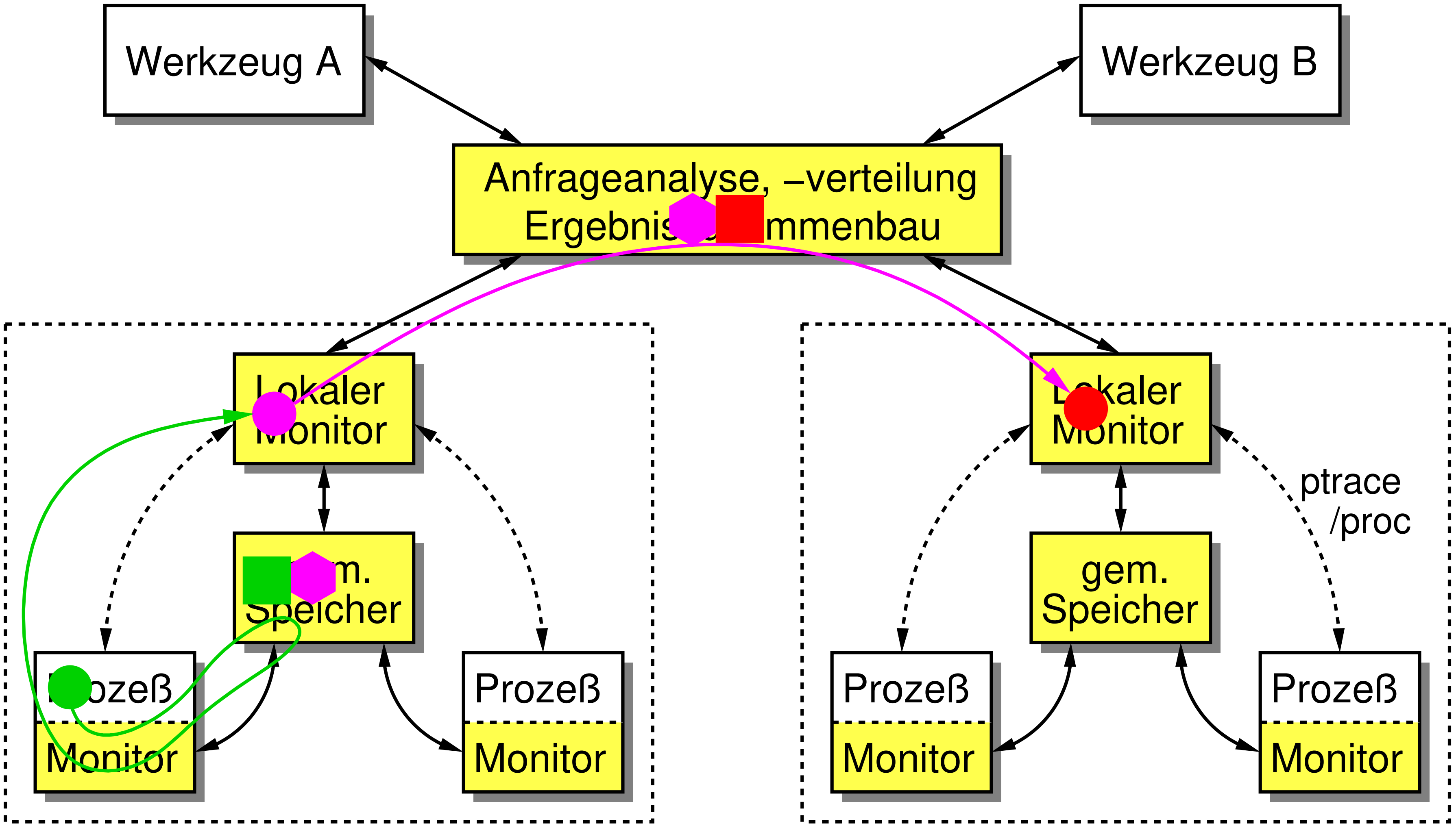
Anfrageverteilung und -abarbeitung im OCM



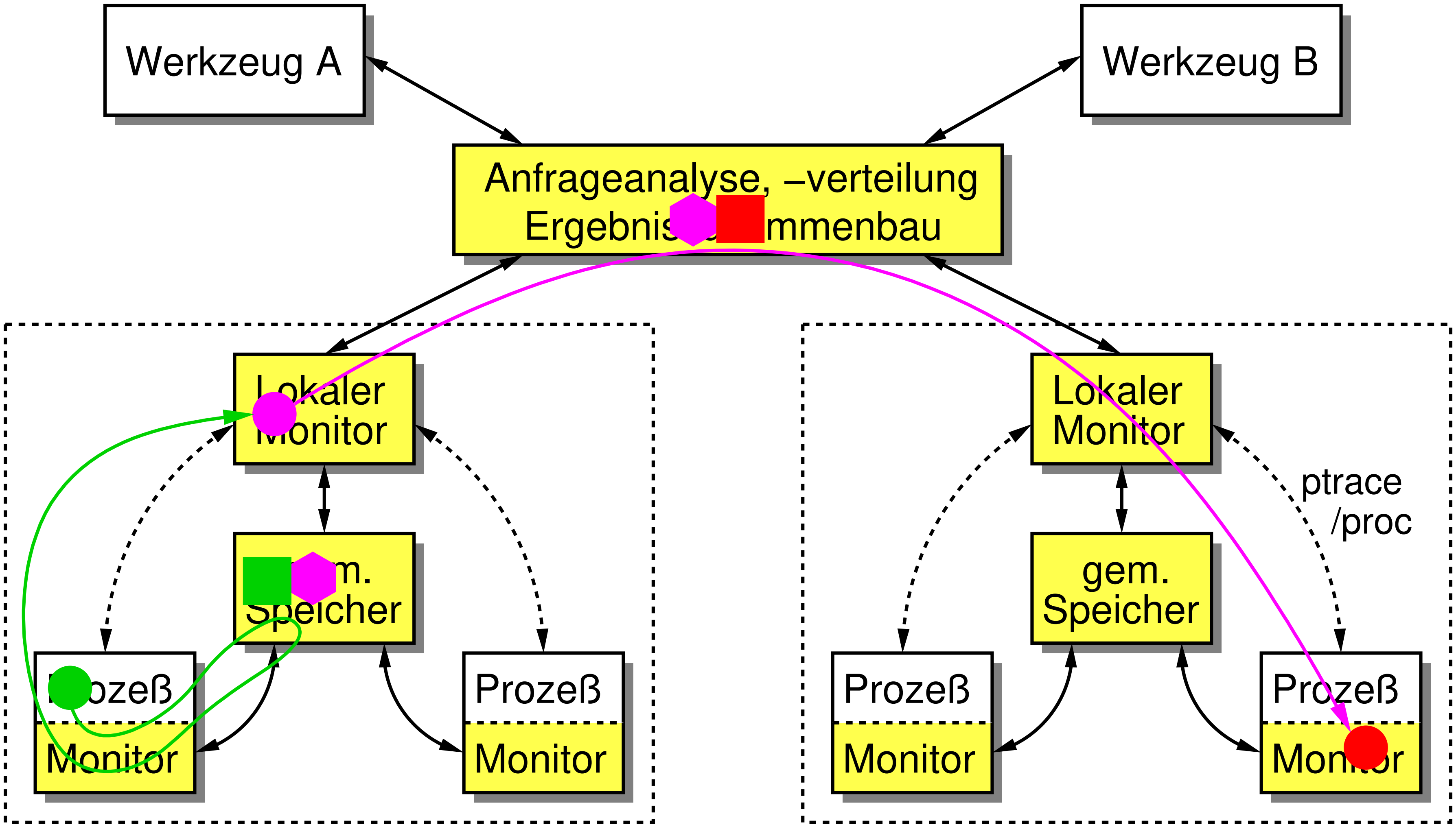
Anfrageverteilung und -abarbeitung im OCM



Anfrageverteilung und -abarbeitung im OCM



Anfrageverteilung und -abarbeitung im OCM



3.1. Zur Realisierung des OCM

Relevante Funktionen:

- Einbringen
  - der Ereignisinstrumentierung
  - des Aktionscodes
  - der Monitorteile

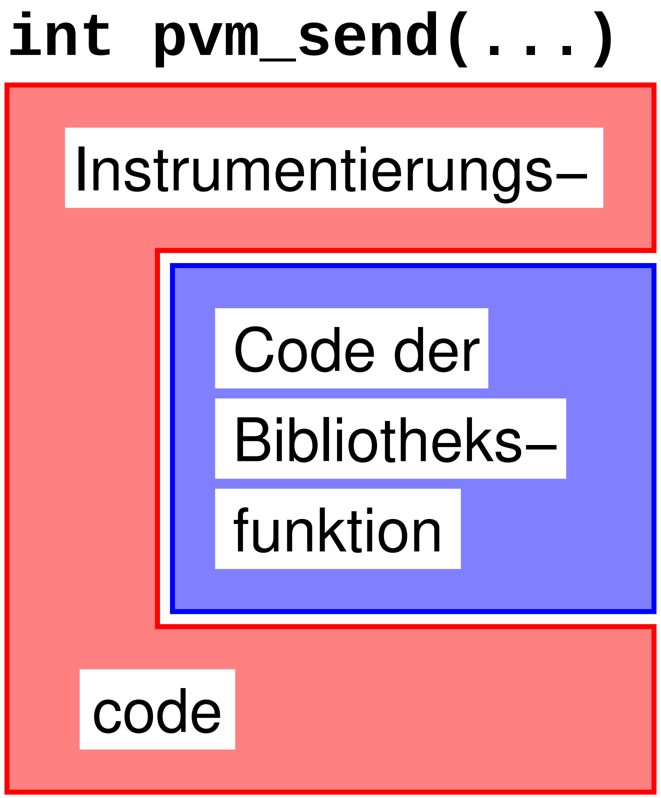
  in die Prozesse
  ohne spezielle Vorbereitung
- Aufruf von Aktionen in überwachten Prozessen
- Dynamische Erweiterbarkeit der OCM-Schnittstelle

Mit Betriebssystemunterstützung:

- Überwachung von Signalen
- Traps

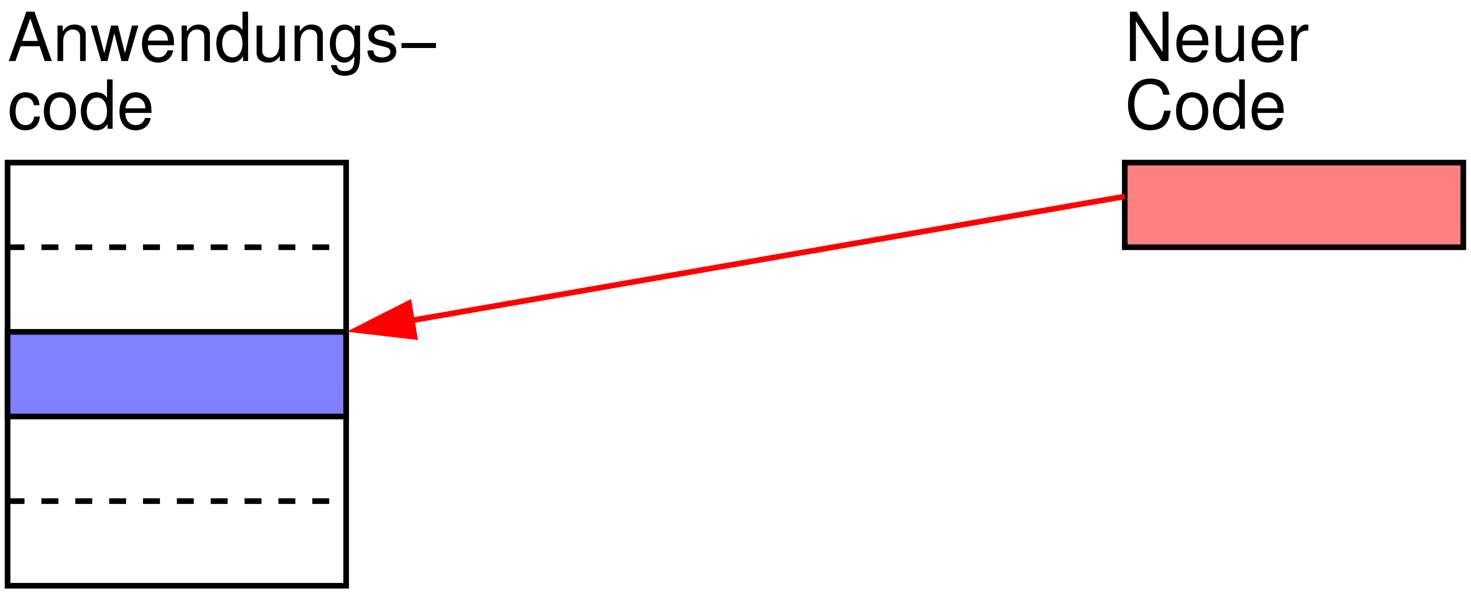
Zusätzlich: *binary wrapping*

- Überwachung von Bibliotheksaufrufen
- Einfach und schnell
- Neues Binden notwendig (statisch / dynamisch)



Erweiterung: *dyninst*

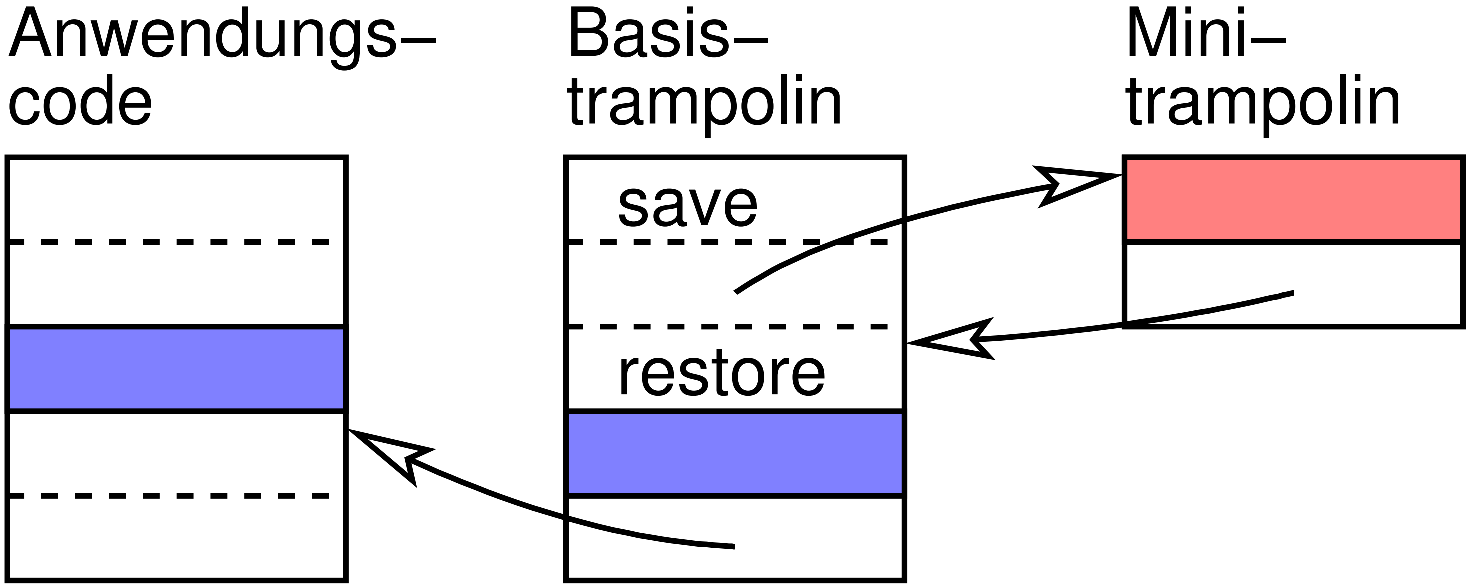
- Dynamische Codemodifikation
- Keine spezielle Vorbereitung notwendig



- Kooperation mit UW Madison (Bart Miller)

Erweiterung: *dyninst*

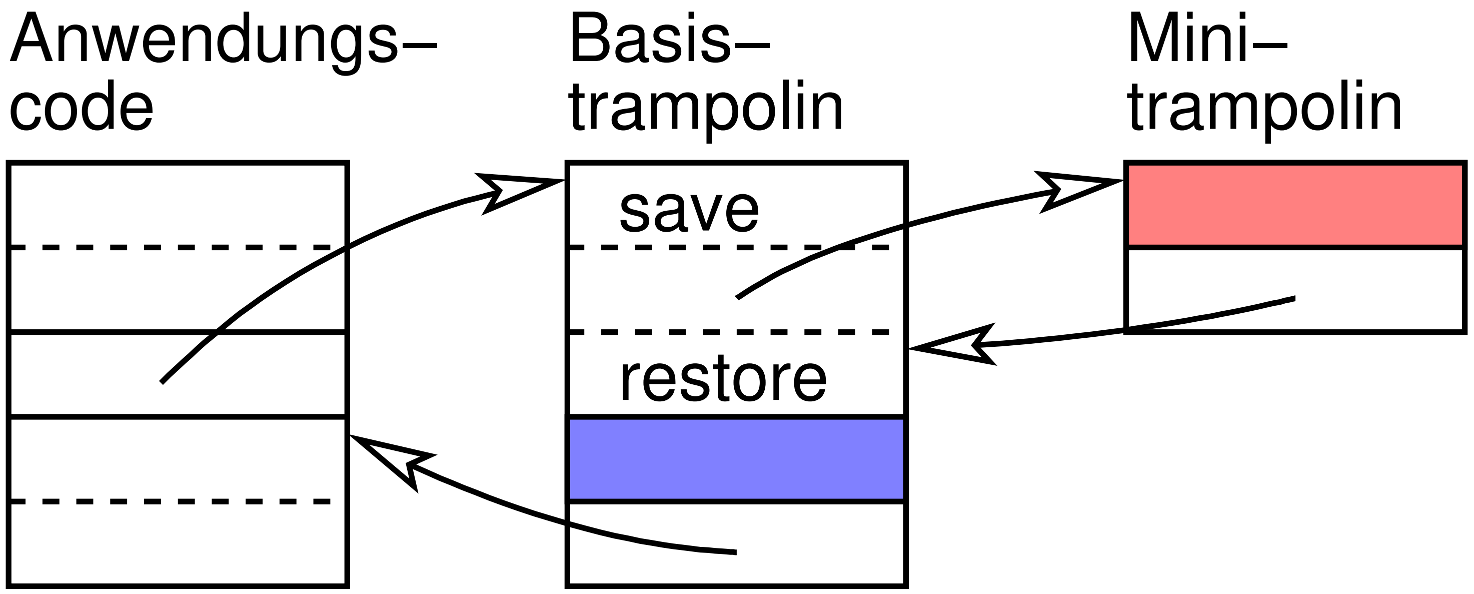
- Dynamische Codemodifikation
- Keine spezielle Vorbereitung notwendig



- Kooperation mit UW Madison (Bart Miller)

Erweiterung: *dyninst*

- Dynamische Codemodifikation
- Keine spezielle Vorbereitung notwendig



- Kooperation mit UW Madison (Bart Miller)

3.1.2. Aktionen in überwachten Prozessen

Inferior RPC

- Sichern des Prozessorstatus
- Manipulation des Befehlszählers / Kellers
- Anhalten des Prozesses bei Rückkehr
- Restaurieren des Prozessorstatus

Schwierigkeit

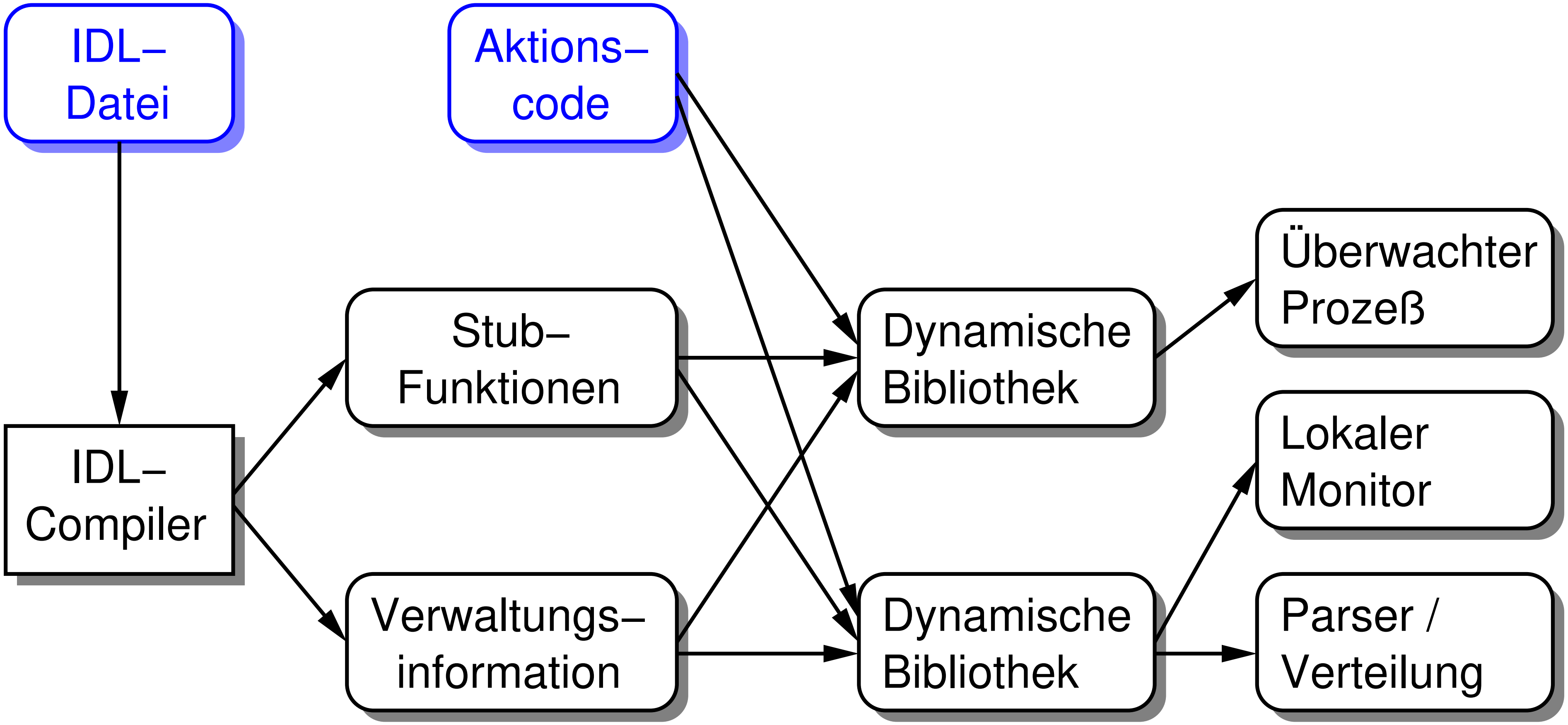
- Aufruf entspricht Unterbrechung
  - Aufruf an ’sicherer’ Stelle: Auslösen durch lokales Ereignis

3.1.3. Dynamische Erweiterbarkeit des OCM

Situation:

- OCM muß Schnittstellen der Ereignisse und Aktionen kennen
  - syntaktische / semantische Analyse der Anfragen
- Information bei Start des OCM noch nicht verfügbar
  - werkzeugspezifische Ereignisse / Aktionen
  - prozeßspezifische Ereignisse / Aktionen

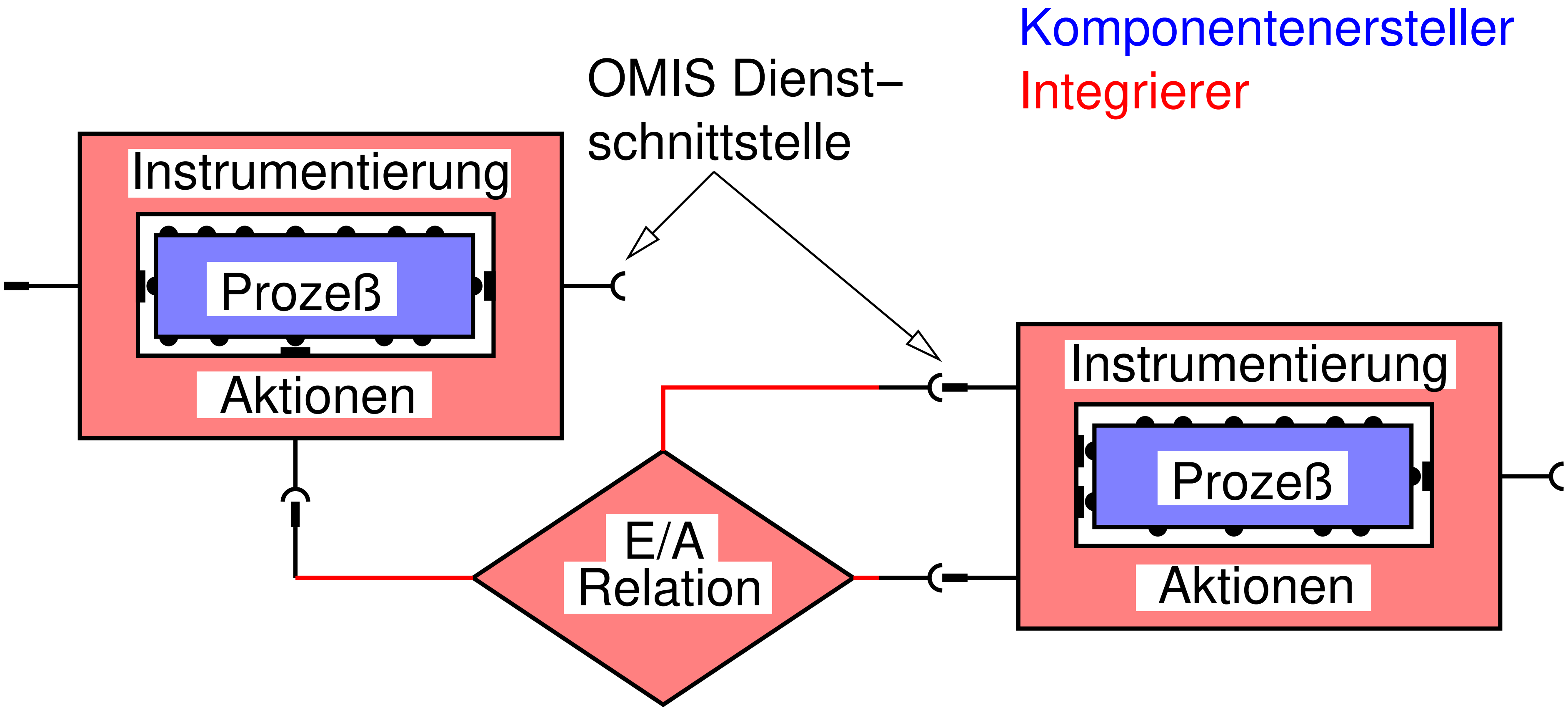
Einbinden neuer Aktionen



Beobachtung: Monitorsystem = Middleware

- Ereignisorientiert
- Verhaltensregeln dynamisch programmierbar
- Dynamisches Anbinden von Schnittstellen(adaptern)
  - alle für Binder sichtbaren Symbole nutzbar
    - globale Variable / Objekte
    - globale Funktionen
    - öffentliche und geschützte Methoden
  - Ereignisse: Beginn / Ende einer Funktion bzw. Methode
  - Aktionen: nutzen Komponente wie Programmierbibliothek

Der OCM als „dynamische“ Middleware



Prototyp der Werkzeugumgebung

- Sichern des Programmzustands, automatisches Rücksetzen
- Konsistenz VISTOP $\leftrightarrow$ DETOP
- 3 Ereignisse + 10 Aktionen
  - nur eine Aktion mit komplexerem Anpassungscode
  - ohne Modifikation des Werkzeug-Quellcodes
- Aber: Modifikation für Datenintegration erforderlich (Konflikte durch Verletzung impliziter Annahmen)
- ☞ Werkzeuge müssen Vorhandensein anderer Werkzeuge prinzipiell berücksichtigen, nicht aber konkrete Kooperation

6. Fazit und Ausblick

Problem interoperabler Komponenten:

- Vorhandensein von Middleware-Schnittstellen
- Vorhersehen der notwendigen Ereignisse und Aktionen

Monitoring-Techniken

$\Rightarrow$ dynamisches Einbinden von Schnittstellen

- Integrierer kann Komponenten wie Bibliothek nutzen

Ähnlichkeit zwischen Monitorsystemen und Middleware

- ☞ Systematische Trennung von Middleware und Monitordiensten

6. Fazit und Ausblick

Prototyp zeigt:

- Ansatz / Realisierungstechniken praxistauglich
- Nutzung noch unkomfortabel
  - Integration mit Koordinationssprache

Am Horizont:

- Dynamische *plug-ins* / Rekonfigurierbarkeit
  - Unterstützung durch Betriebssysteme und Compiler
  - Nutzung innerhalb von Betriebssystemen