# Laufzeit-Werkzeuge
# für parallele und verteilte Systeme

Roland Wismüller

LRR-TUM
Technische Universität München

## Werkzeuge zur Laufzeitanalyse

- *Beobachten*, *Analysieren* und *Steuern* des Ablaufs eines verteilten Hard- und Softwaresystems (Zielsystem)
- Hier betrachtet: *on-line* Werkzeuge
- Beispiele:
  - Quellsprach-Debugging
  - Visualisierung des Programmablaufs
  - Deterministische Ablaufsteuerung
  - Leistungsanalyse
  - Sicherungspunkterstellung und Prozeßmigration
  - Lastausgleich

# Agenda

1. Einführung
2. Werkzeug-Entwicklung am LRR-TUM
3. Online-Monitoring
4. Interoperable Werkzeuge
5. Zusammenfassung und Ausblick

# 2.1. Quellsprach-Debugging

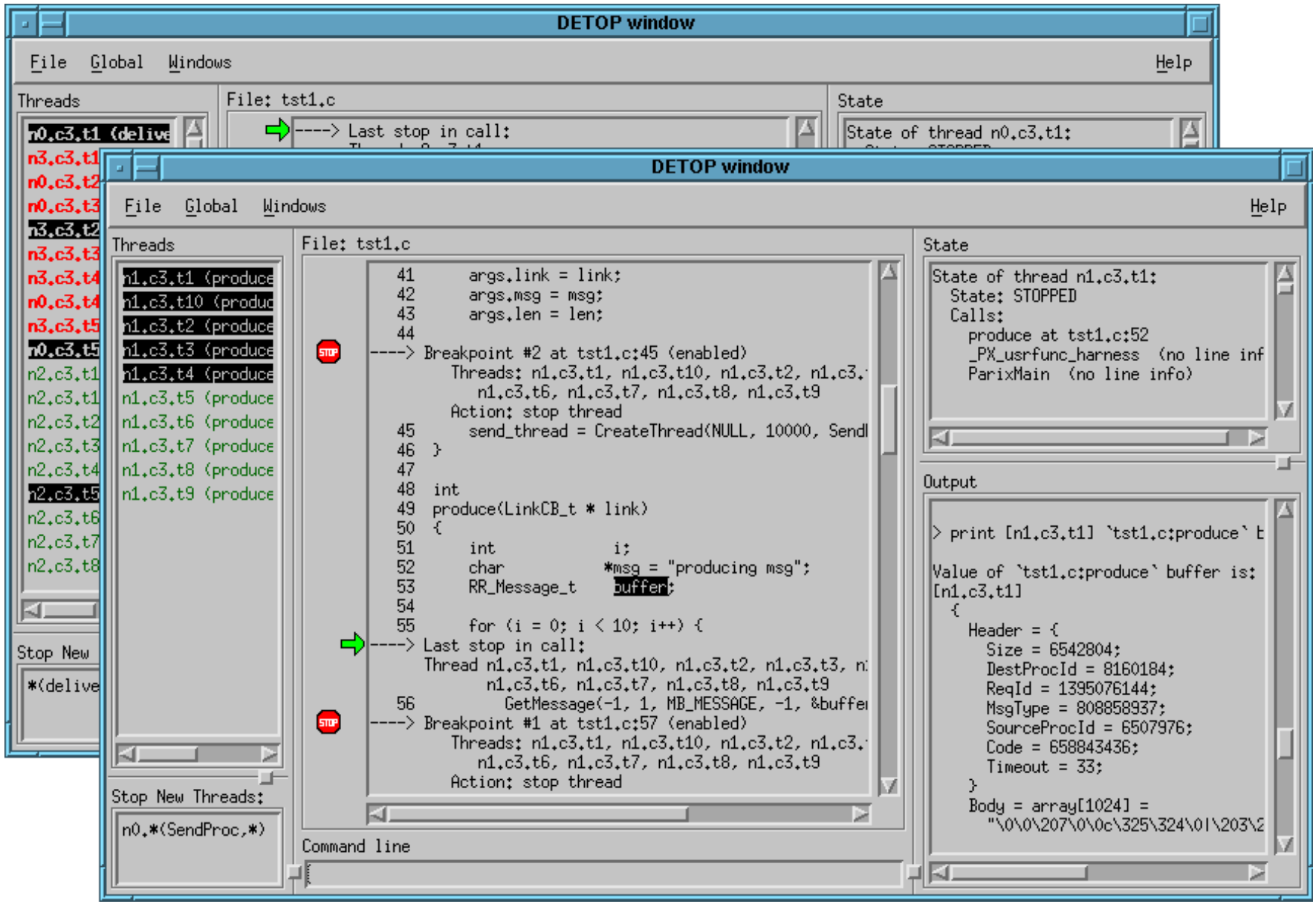
## Verteilter Debugger DETOP

- Realisiert für MMK, PARIX, PVM, MPI, HPF (SPiDER)

## Fragestellungen

- Effiziente Mechanismen zum Monitoring von Threads
- Skalierbarkeit
- Compiler-Optimierungen
- Mehrstufige Übersetzung / Parallelisierung

# 2.1. Quellsprach-Debugging



## Compiler-Optimierungen

| Quellprogramm | | Zielprogramm | |
|---|---|---|---|
| | | 1234: | store ...,r0 |
| 10 | v = ... | | ... |
| 11 | w = ... | 1236: | store ...,r0 |
| 12 | v = ... | | ... |
| | | 1239: | store ...,r1 |

## Compiler-Optimierungen

**Quellprogramm**

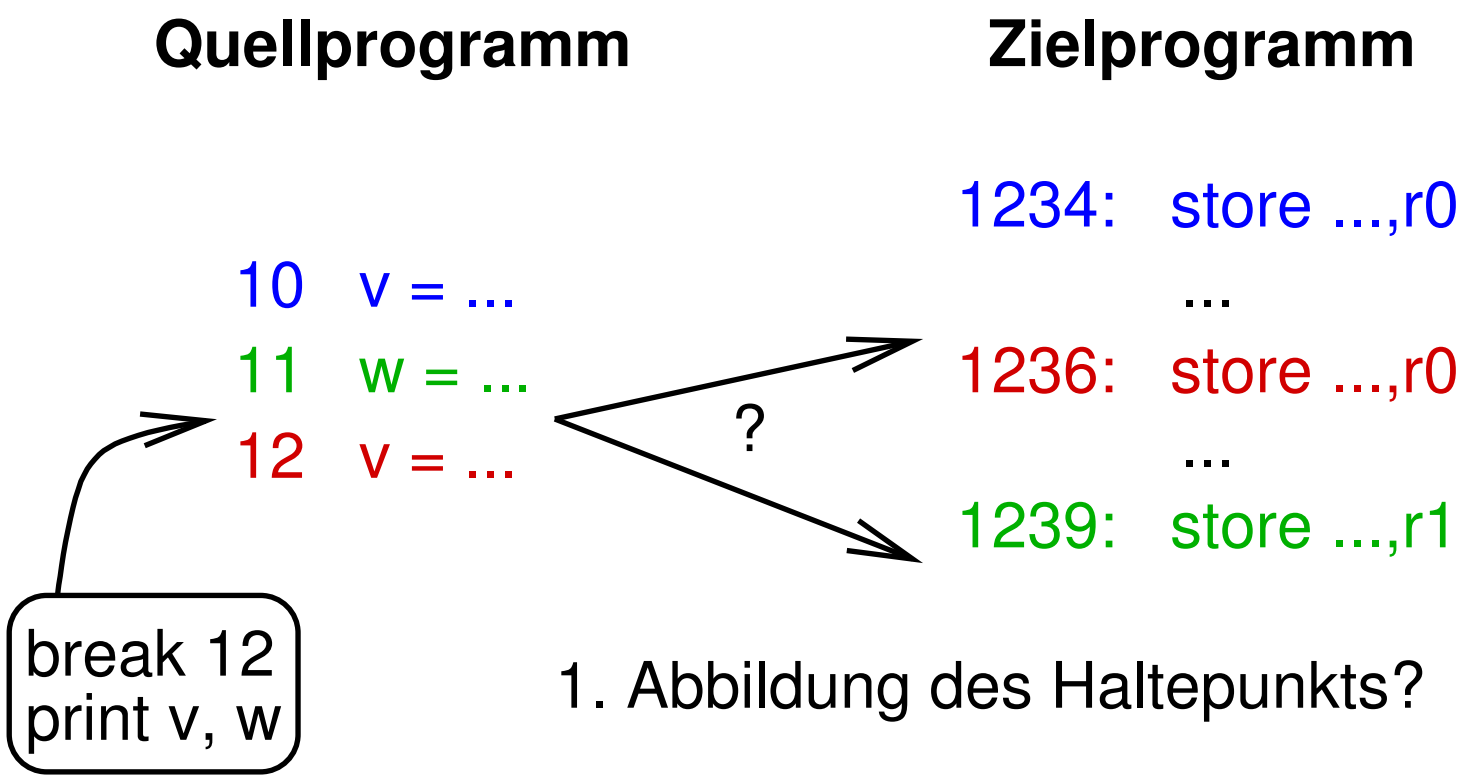
```
10   v = ...
11   w = ...
12   v = ...
```

break 12
print v, w

**Zielprogramm**

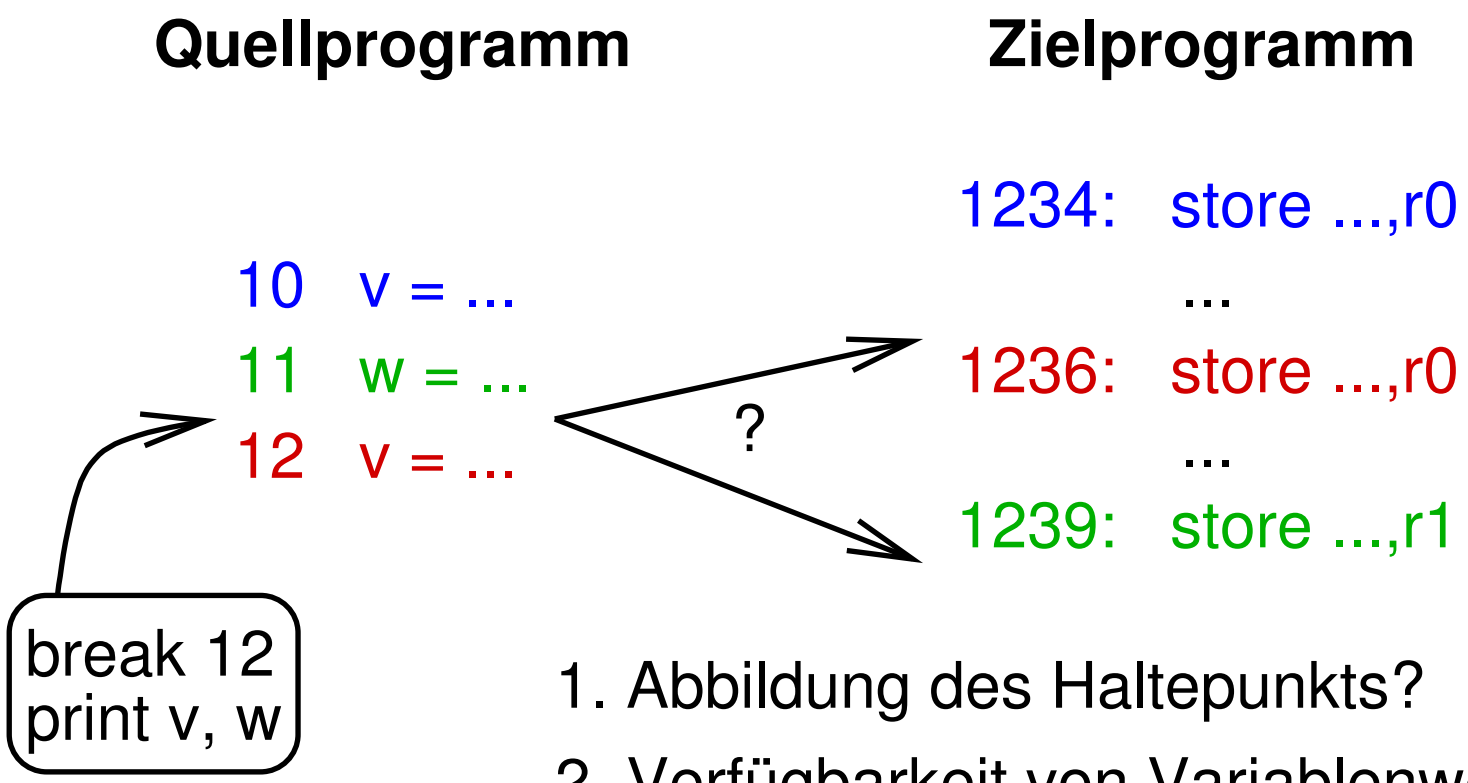
```
1234:   store ...,r0
          ...
1236:   store ...,r0
          ...
1239:   store ...,r1
```

## Compiler-Optimierungen



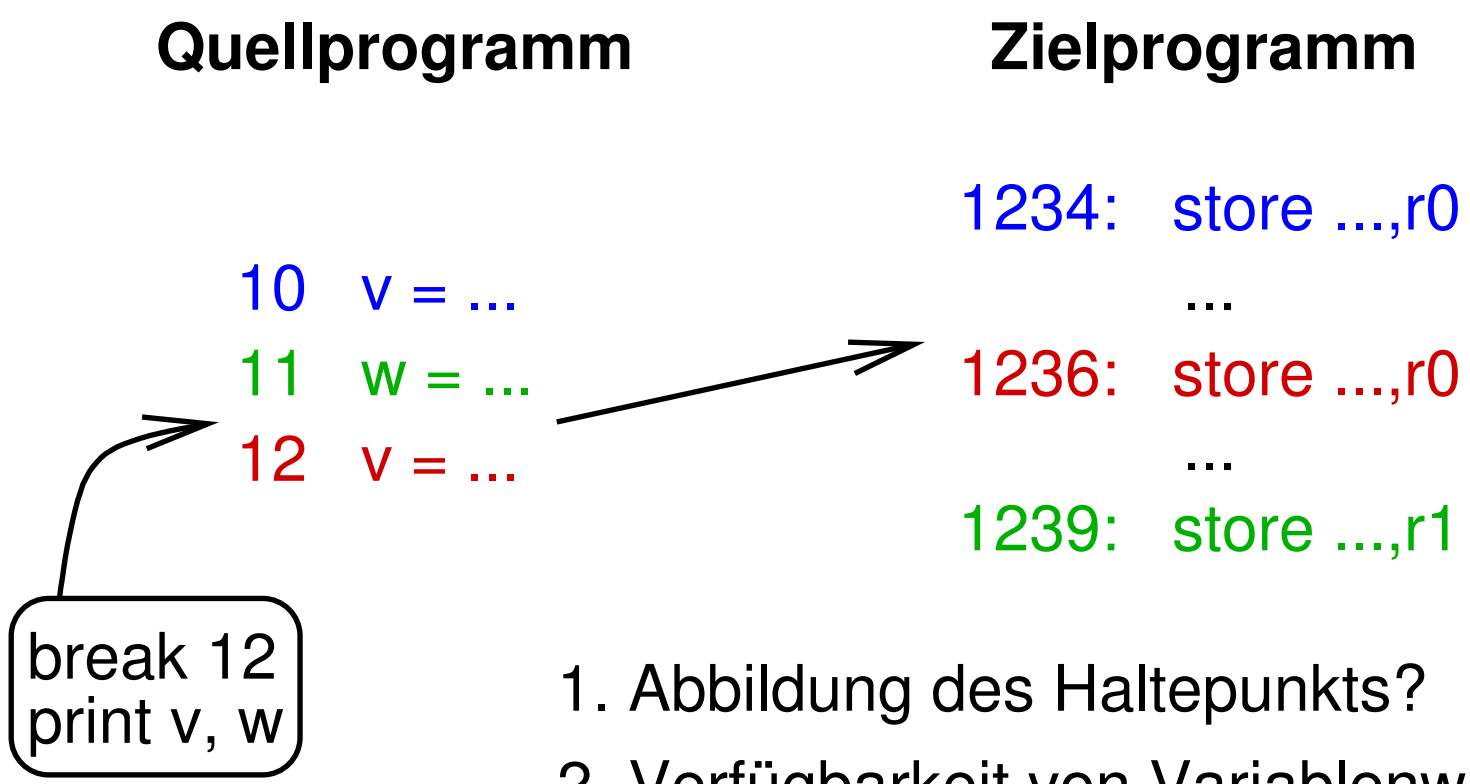

1. Abbildung des Haltepunkts?

## Compiler-Optimierungen



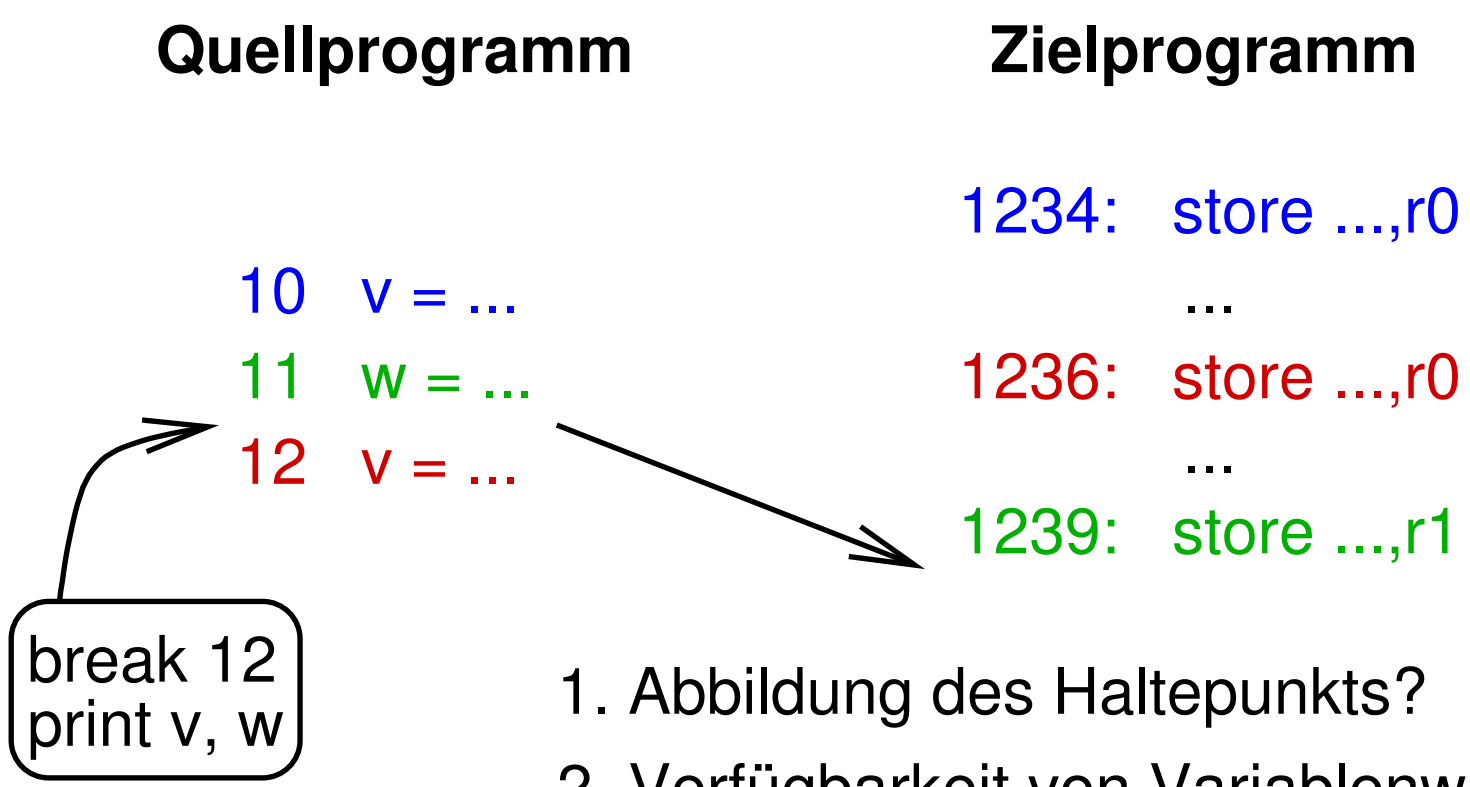

1. Abbildung des Haltepunkts?
2. Verfügbarkeit von Variablenwerten?

## Compiler-Optimierungen



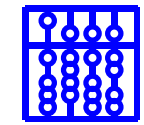

1. Abbildung des Haltepunkts?
2. Verfügbarkeit von Variablenwerten?
   - r0 enthält Wert von v
   - Wert von w nicht verfügbar

## Compiler-Optimierungen




1. Abbildung des Haltepunkts?
2. Verfügbarkeit von Variablenwerten?
   - r1 enthält Wert von w
   - Wert von v nicht verfügbar

# 2.2. Visualisierung des Programmablaufs



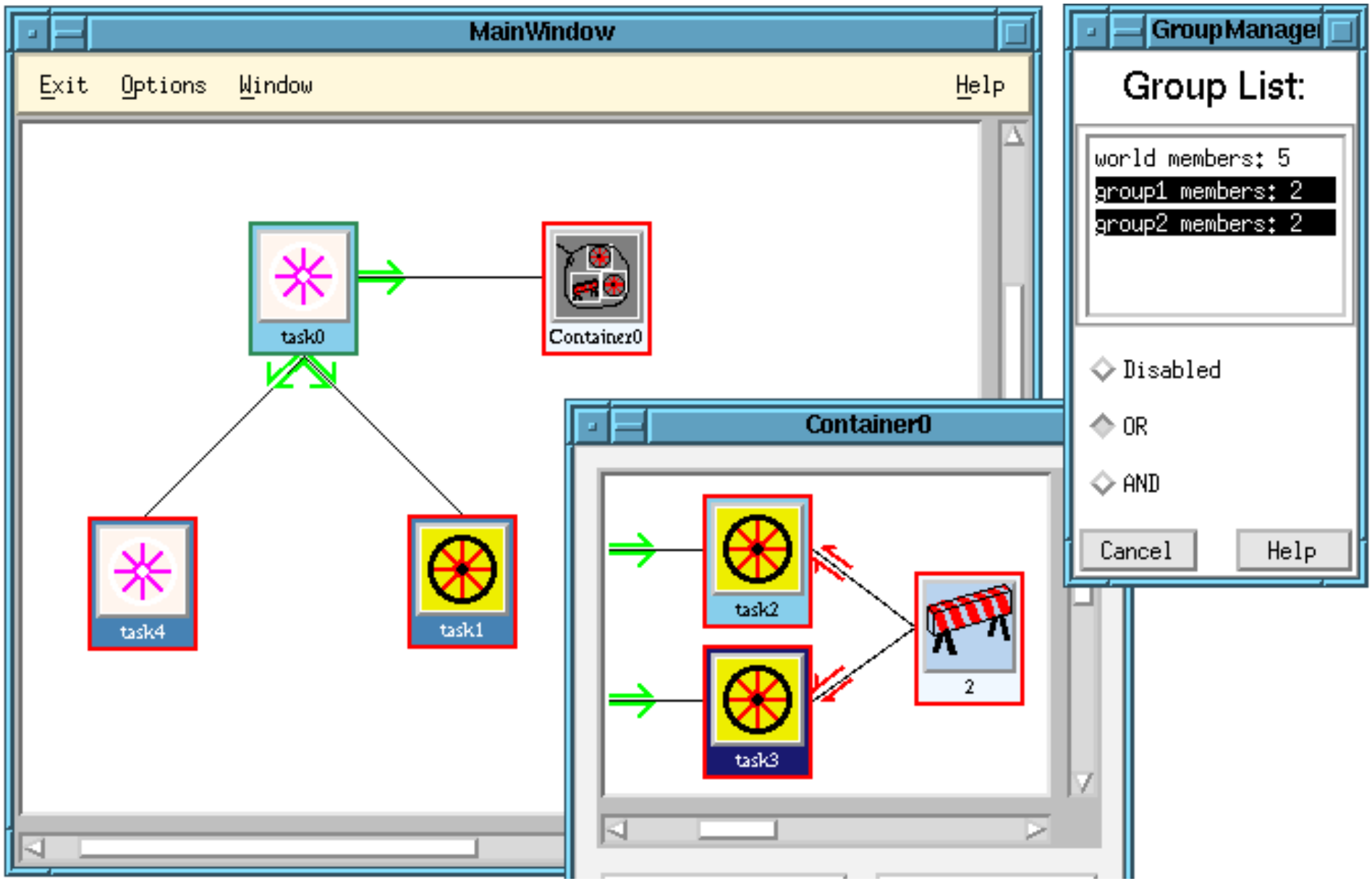
## Visualisierer VISTOP

- zeigt (mögliche) Folgen globaler Zustände
  - Kommunikation, Synchronisation, Prozeßgruppen
- Realisiert für MMK und PVM

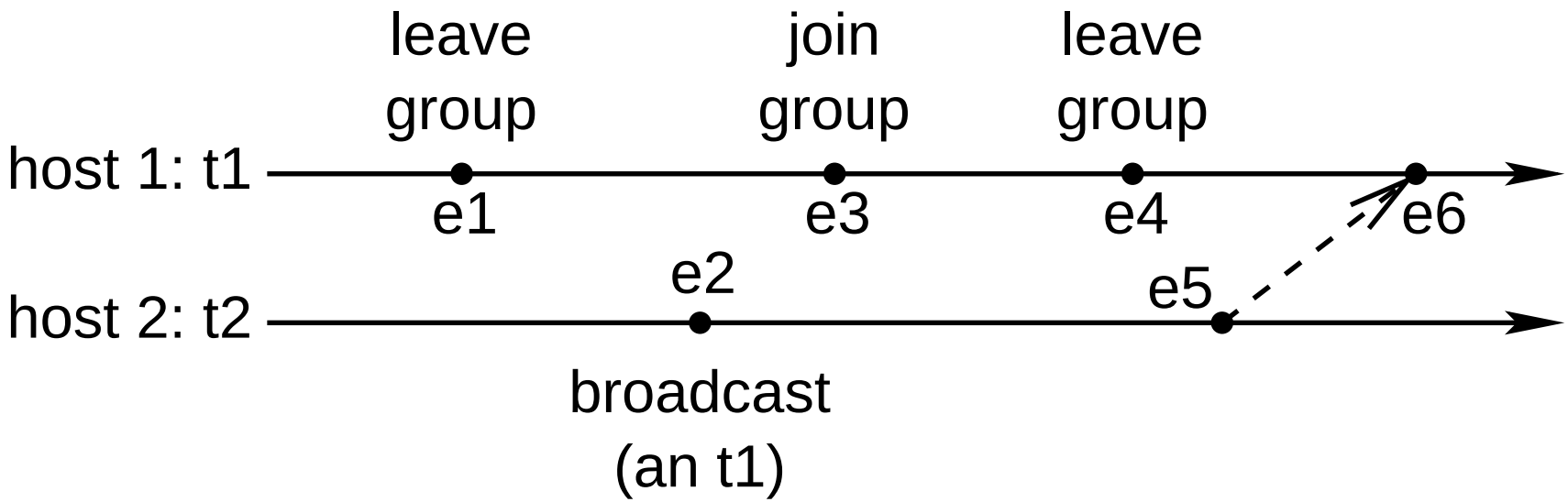
## Fragestellungen

- Visualisierungsmetaphern
- Datenerfassungstechniken
- Berechnung konsistenter globaler Zustände

# 2.2. Visualisierung des Programmablaufs

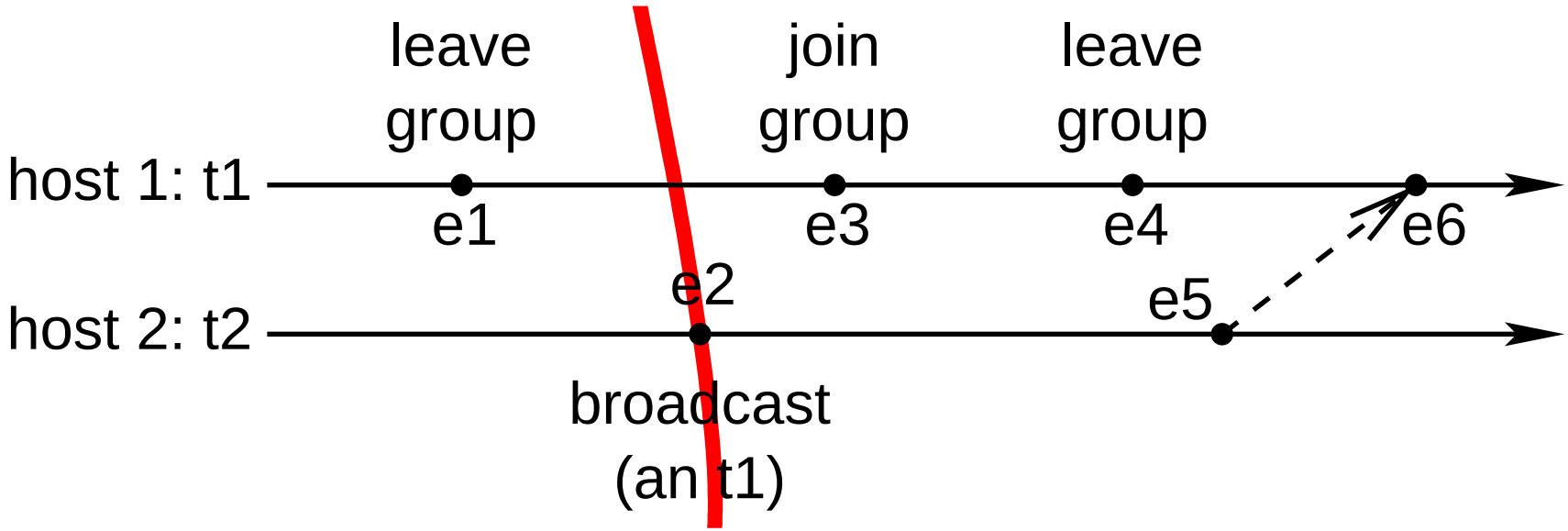


## Berechnung konsistenter globaler Zustände



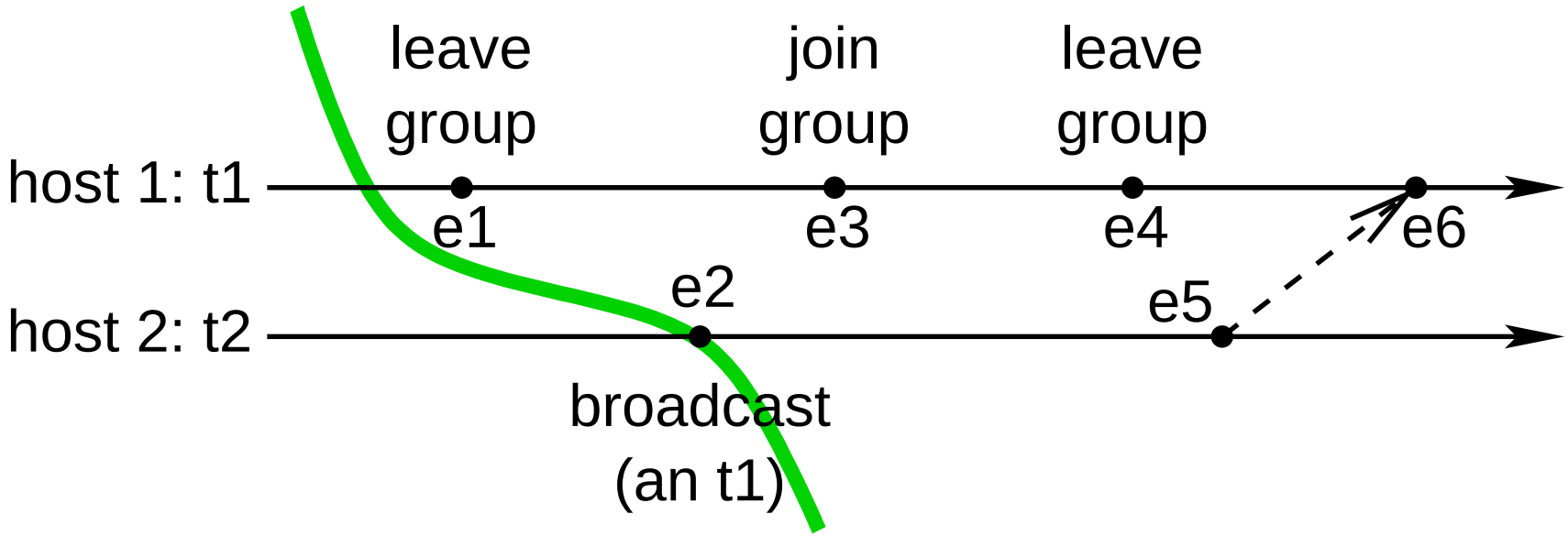

- Reihenfolge $e1, e2, e3, e4, e5, e6$ offenbar inkonsistent
- Reihenfolgen $e2, e1, e3, e4, ...$ und $e1, e3, e2, e4, ...$ konsistent
- Keine Kausalitätsbeziehung zwischen $e2, e1$ bzw. $e2, e3$

## Berechnung konsistenter globaler Zustände



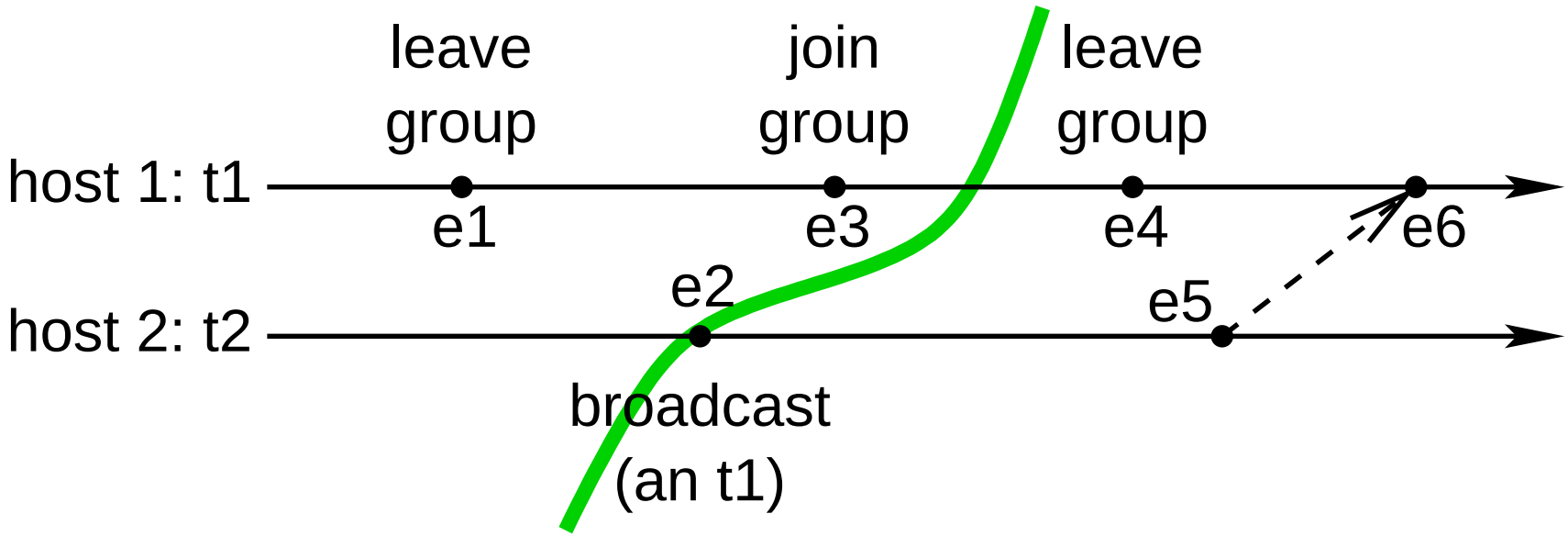

- Reihenfolge $e1, e2, e3, e4, e5, e6$ offenbar inkonsistent
- Reihenfolgen $e2, e1, e3, e4, ...$ und $e1, e3, e2, e4, ...$ konsistent
- Keine Kausalitätsbeziehung zwischen $e2, e1$ bzw. $e2, e3$

## Berechnung konsistenter globaler Zustände



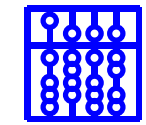

- Reihenfolge $e1, e2, e3, e4, e5, e6$ offenbar inkonsistent
- Reihenfolgen $e2, e1, e3, e4, ...$ und $e1, e3, e2, e4, ...$ konsistent
- Keine Kausalitätsbeziehung zwischen $e2, e1$ bzw. $e2, e3$

## Berechnung konsistenter globaler Zustände




- Reihenfolge $e1, e2, e3, e4, e5, e6$ offenbar inkonsistent
- Reihenfolgen $e2, e1, e3, e4, ...$ und $e1, e3, e2, e4, ...$ konsistent
- Keine Kausalitätsbeziehung zwischen $e2, e1$ bzw. $e2, e3$

# 2.3. Deterministische Ablaufsteuerung



## Ziel: Testunterstützung

- Nicht nur *Replay*
- Erzwingen möglicher Abläufe durch Zugriffsmuster für gemeinsame Objekte

## Ablaufsteuerungs-Werkzeug codex

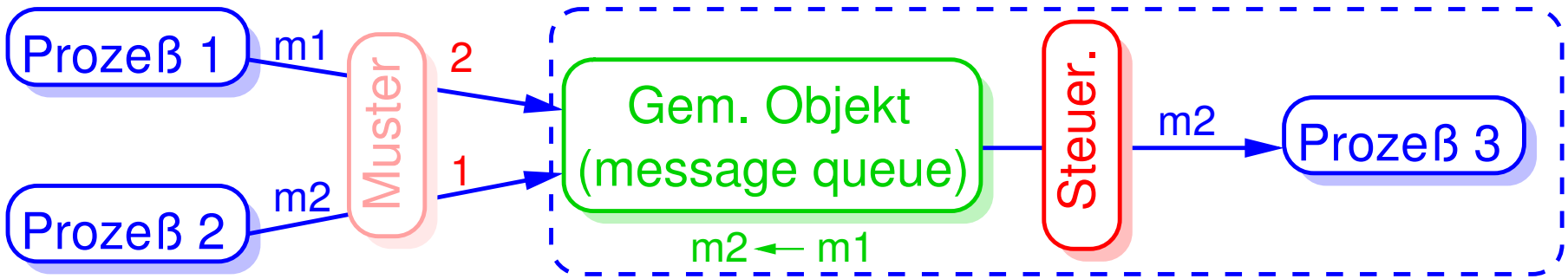
- Realisiert für PVM

## Fragestellungen

- Spezifikation von Zugriffsmustern
- Effizientes Erzwingen der Zugriffsmuster

## Effizientes Erzwingen der Zugriffsmuster

- Für PVM: Spezielle Semantik der gemeinsamen Objekte (*message queues*)
- Konzeptuell: Muster steuert Reihenfolge der Schreibzugriffe (Sendeaufrufe)
- Realisierung: Umordnung der Nachrichten erst beim Empfang



## Leistungsanalyse-Werkzeuge PATOP und TATOO

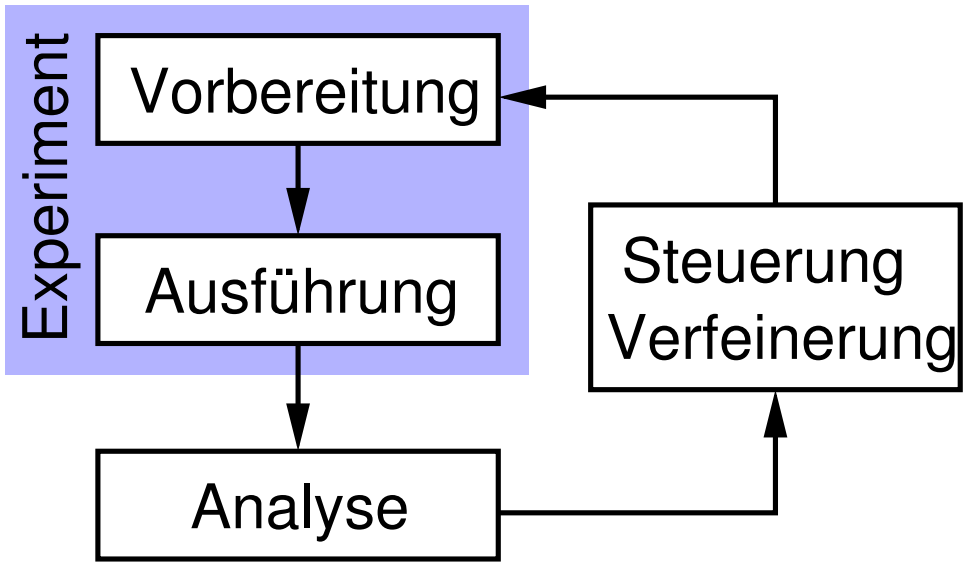
- Online- und Offline-Analyse
- Realisiert für MMK, PARIX, PVM, MPI, HPF, ...

## Fragestellungen

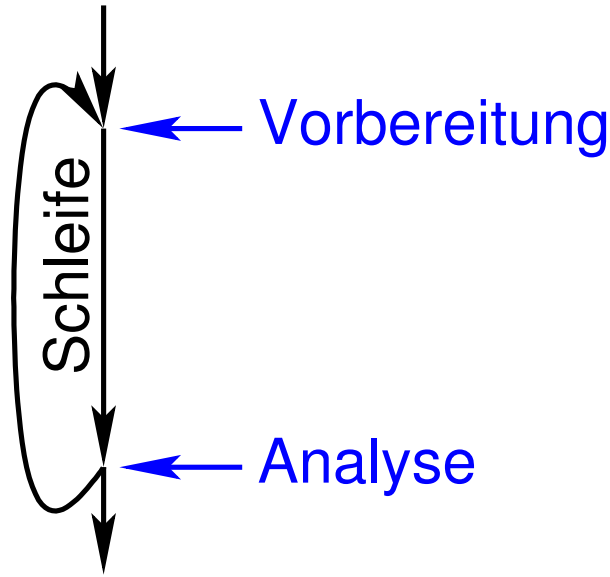
- Online-Analyse mit verteilter Auswertung
- Leistungsanalyse in nicht-exklusiven Systemen
- Automatische Online-Leistungsanalyse

## Automatische Online-Leistungsanalyse

**Performance–Experiment**




**Online–Experiment**




- Verfeinerung von Hypothesen nur für wiederholte Phasen möglich
- Werkzeug benötigt Strukturinformation (Programmierer, Compiler)

## Werkzeuge CoCheck, LoMan, LMC

- CoCheck: Sicherungspunkte und Migration für PVM
- LoMan: Lastverwalter auf Basis von CoCheck
- LMC: Lastverwaltung für CORBA

## Fragestellungen

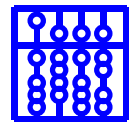
- Protokolle (konsistente Sicherungspunkte, Migration)
- Lastmessung und -bewertung
- Kombination von Initialplazierung, Migration und Replikation

## Motivation

- Jedes Online-Werkzeug benötigt verteilte Infrastruktur zur
  - Beobachtung
  - Manipulation

  von Programmabläufen
- Bisher:
  - Werkzeuge besitzen spezielle, proprietäre Monitorsysteme
  - Keine Standard-Schnittstellen
- Probleme:
  - Entwicklungs- und Portierungsaufwand
  - Interoperabilität

# 3.1. Monitoring-Schnittstelle



## Anforderungen an eine Standard-Schnittstelle

- Unterstützung für Online-Monitoring
- Plattformunabhängigkeit
- Nutzbarkeit für verschiedenste Werkzeuge
- Unterstützung interoperabler Werkzeuge
- Effiziente, skalierbare Implementierung

## Umsetzung:

- OMIS (On-line Monitoring Interface Specification)

## Konzepte von OMIS

- Objektbasiertes Modell des Zielsystems
  - Objekte mit abstrakten Identifikatoren
  - Dienste auf diesen Objekten
  - Werkzeuge definieren ihre Sicht des Zielsystems
- Ereignis/Aktions-Modell
  - Anfrage = Ereignisdefinition + Aktionsliste
  - Übergabemechanismus für Ereignisinformation
- Skalierbarkeit: Dienste auf Objektmengen
- Ortstransparenz
- Erweiterbarkeit

## OMIS Beispielanfragen

```
thread_reached_addr([p_14,p_52], 0x124):
     thread_stop([a_])
```

Wenn ein Thread in einem der Prozesse die Adresse `0x124` erreicht, stoppe alle überwachten Threads.

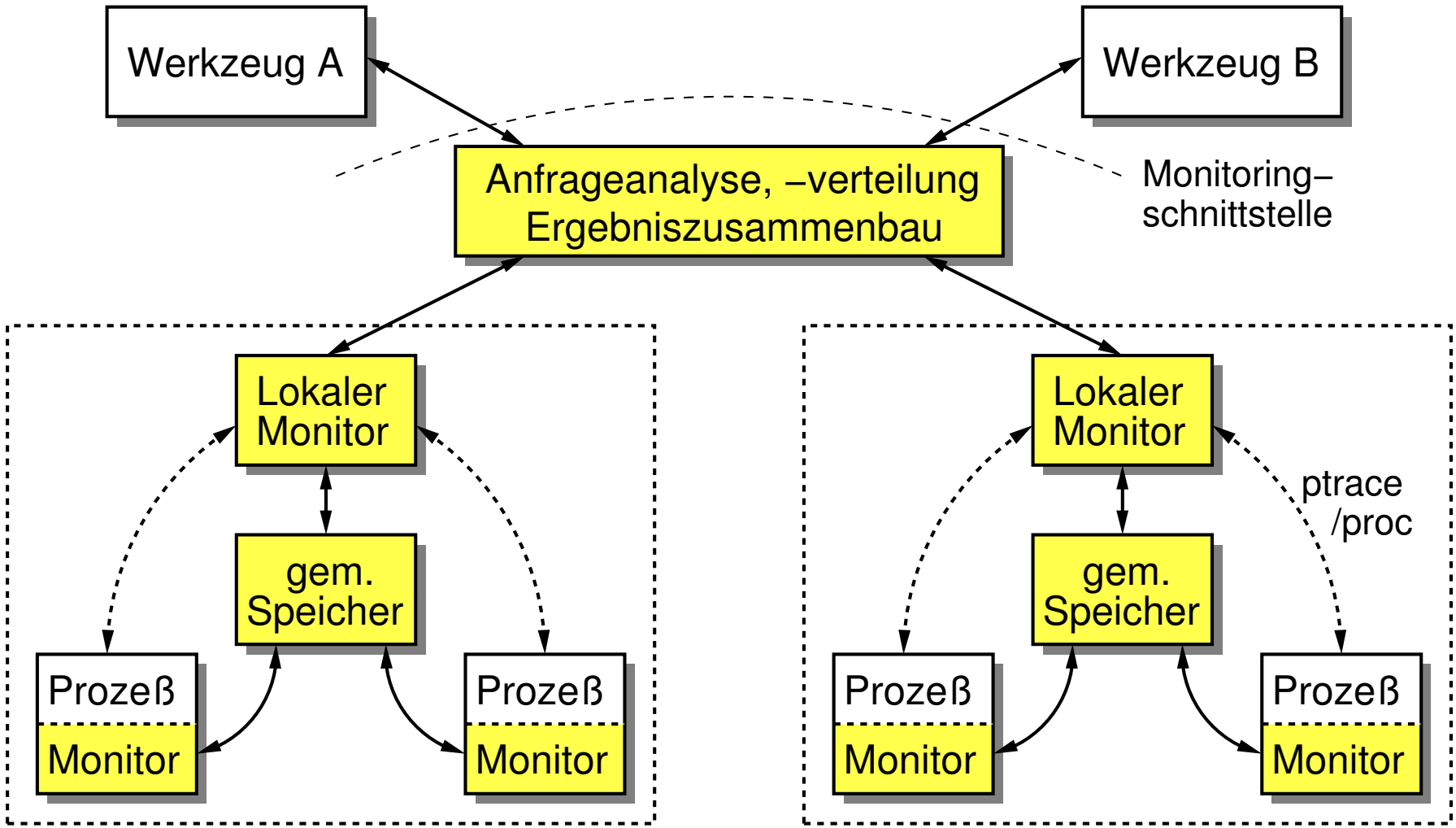
```
thread_has_ended_lib_call([a_], "pvm_recv") :
     pa_counter_global_increment(pa_c_1, 1)
thread_creates_proc([a_]) :
     node_attach([$new_node]) proc_attach([$new_proc])
: pa_counter_global_read([pa_c_1], 1)
```

Anzahl der Aufrufe von `pvm_recv` in allen Threads (Prozessen)

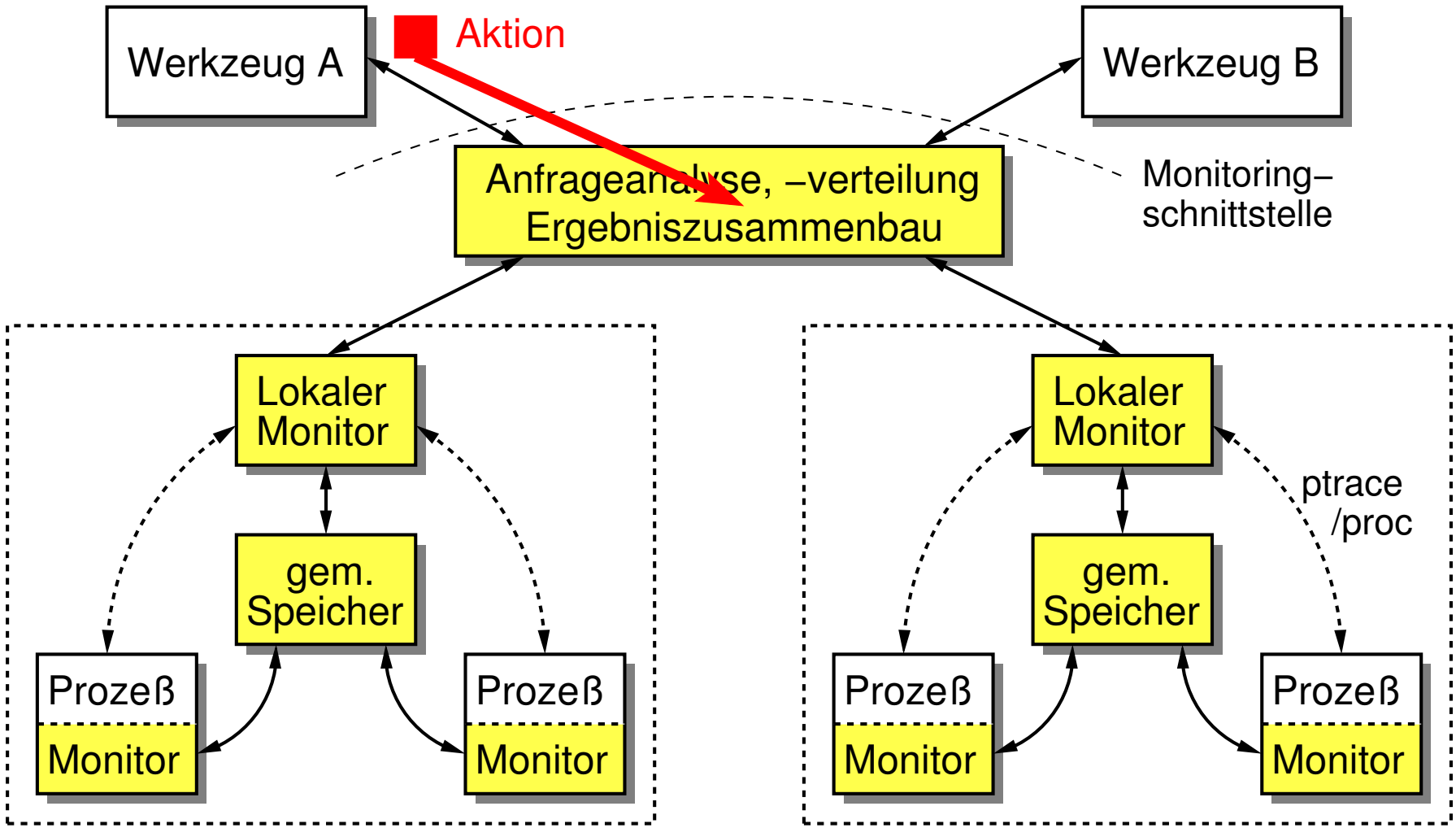
## Implementierung von OMIS: OCM

- Für PVM und MPI auf UNIX-Clustern
  (Solaris, Irix, Linux, ...)
- Erste Werkzeuge: DETOP, VISTOP, CoCheck, PATOP
- Portierungen auf andere Plattformen
  - DSM-Systeme (SMiLE-Cluster am LRR-TUM, HW-Monitor)
  - SMP-Systeme (Thread-Monitoring)
  - Java / RMI
  - Grid-Umgebungen

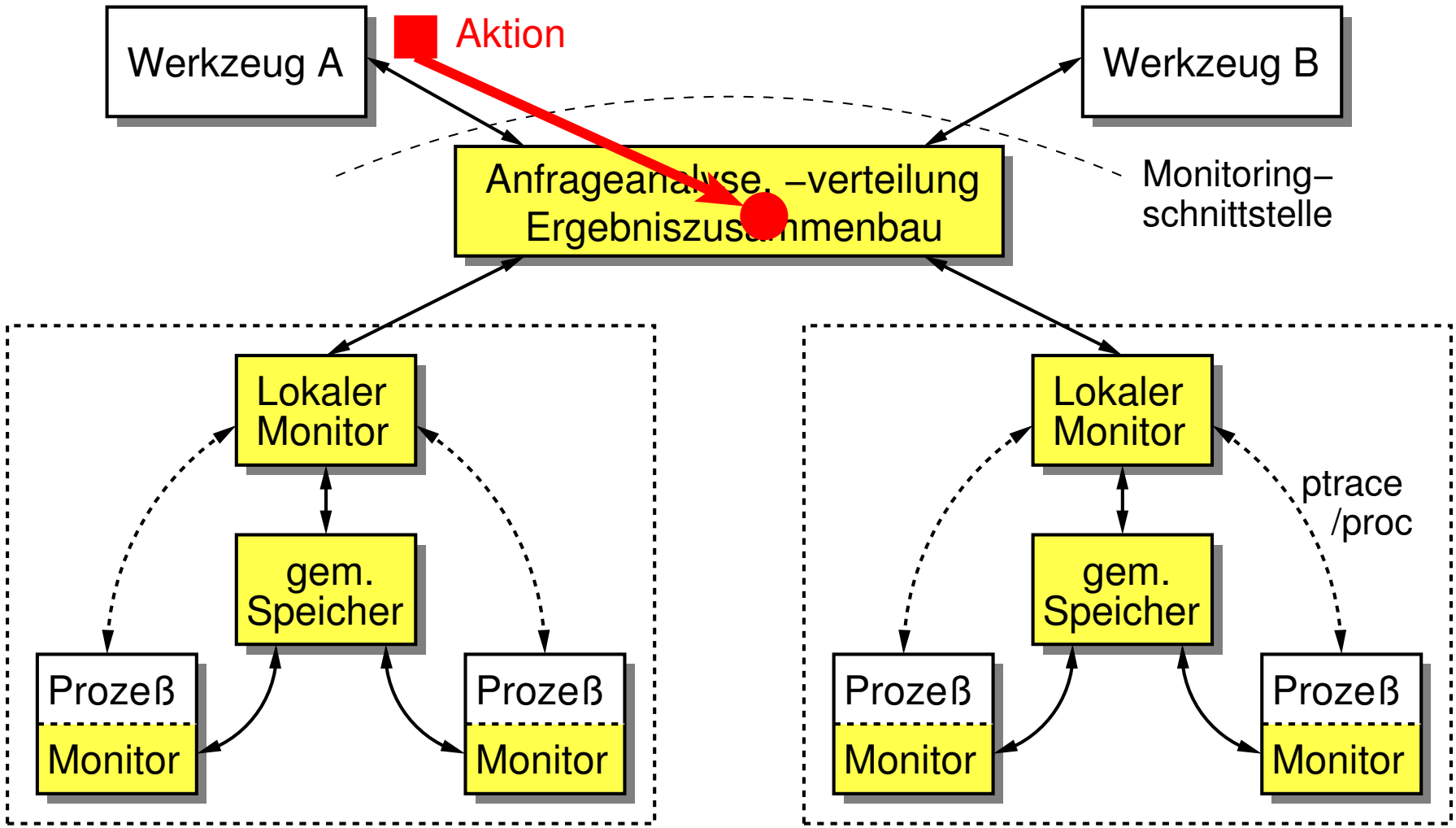
## Struktur des OCM



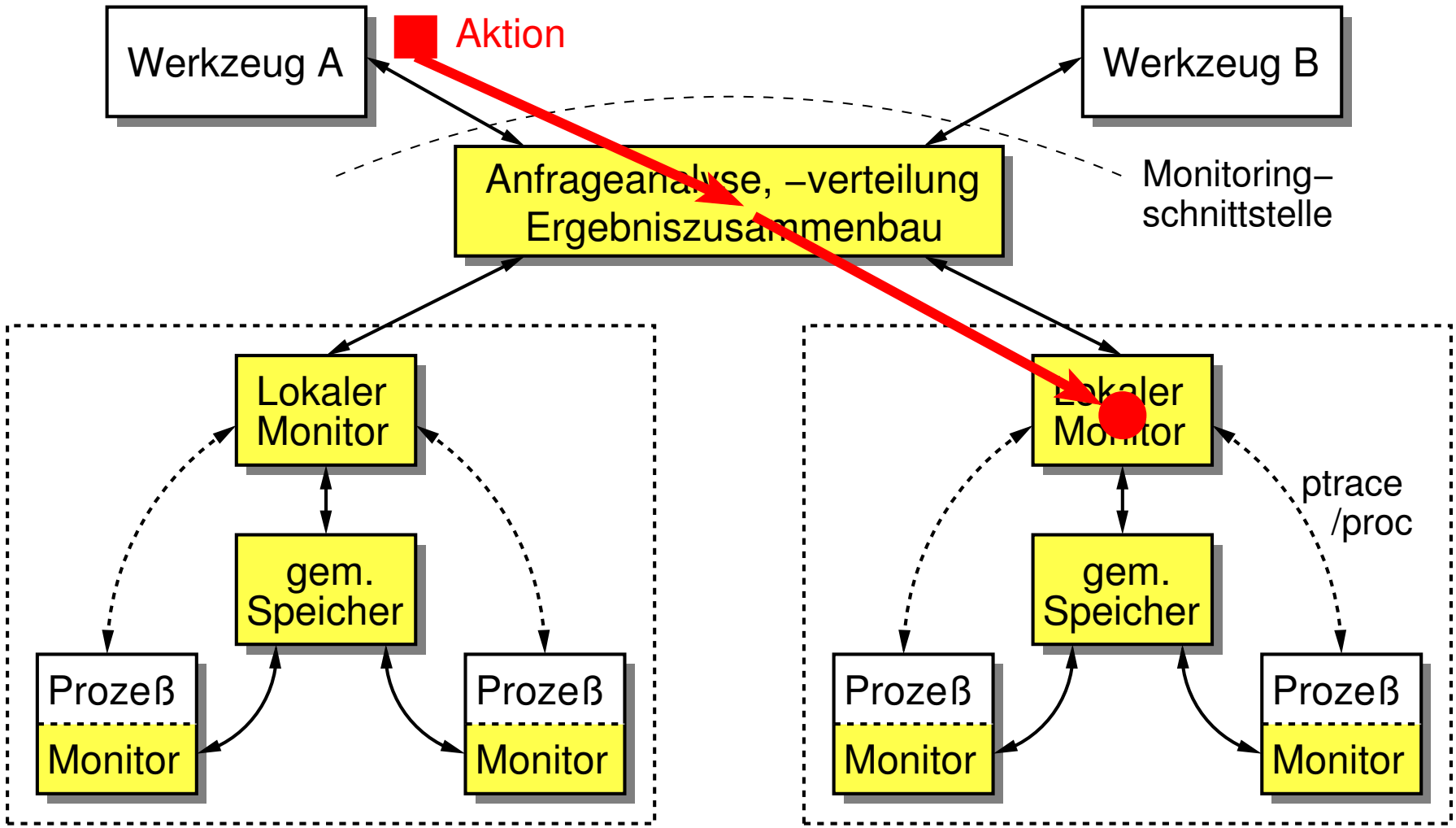
## Anfrageabarbeitung im OCM



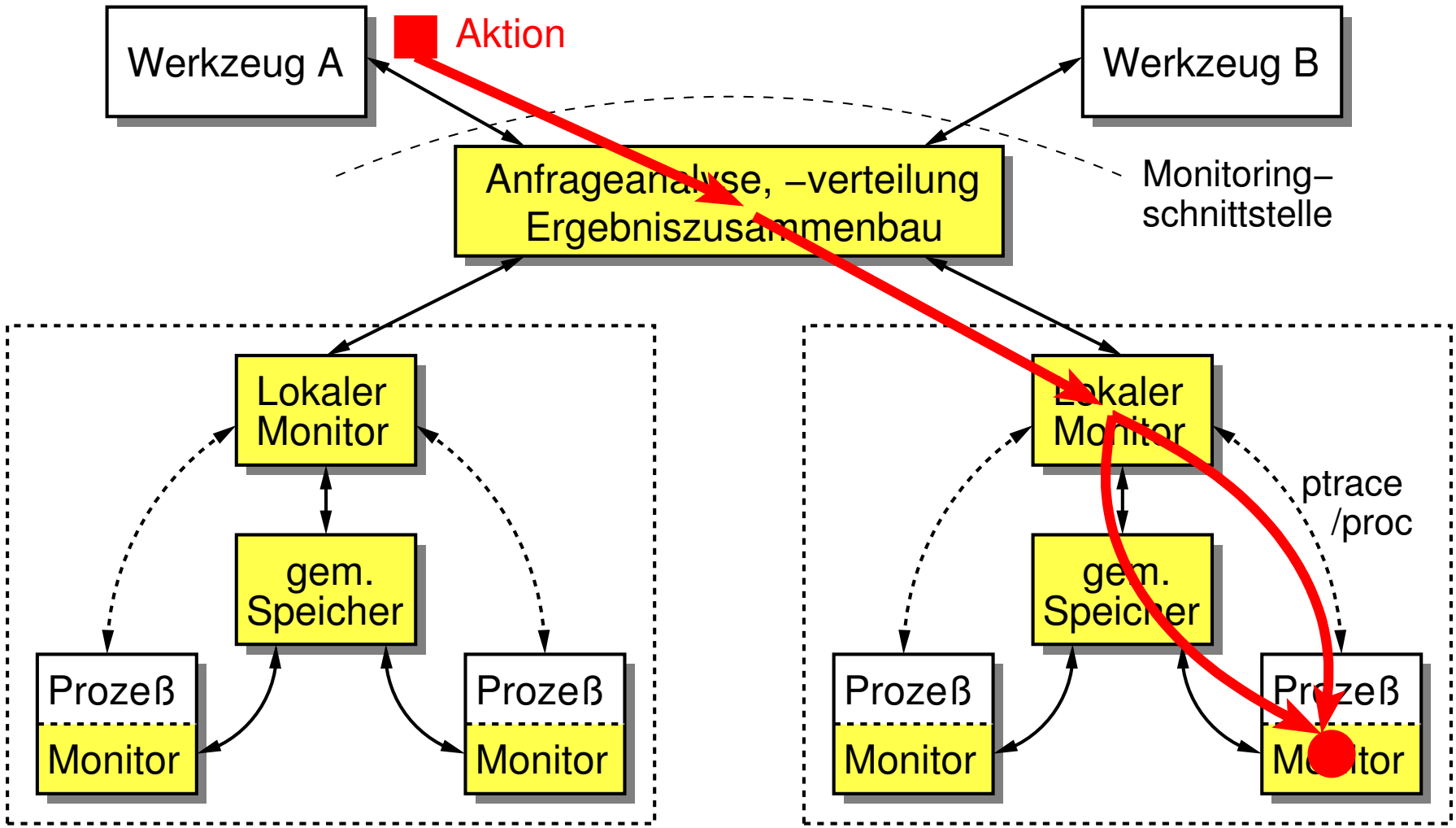
## Anfrageabarbeitung im OCM



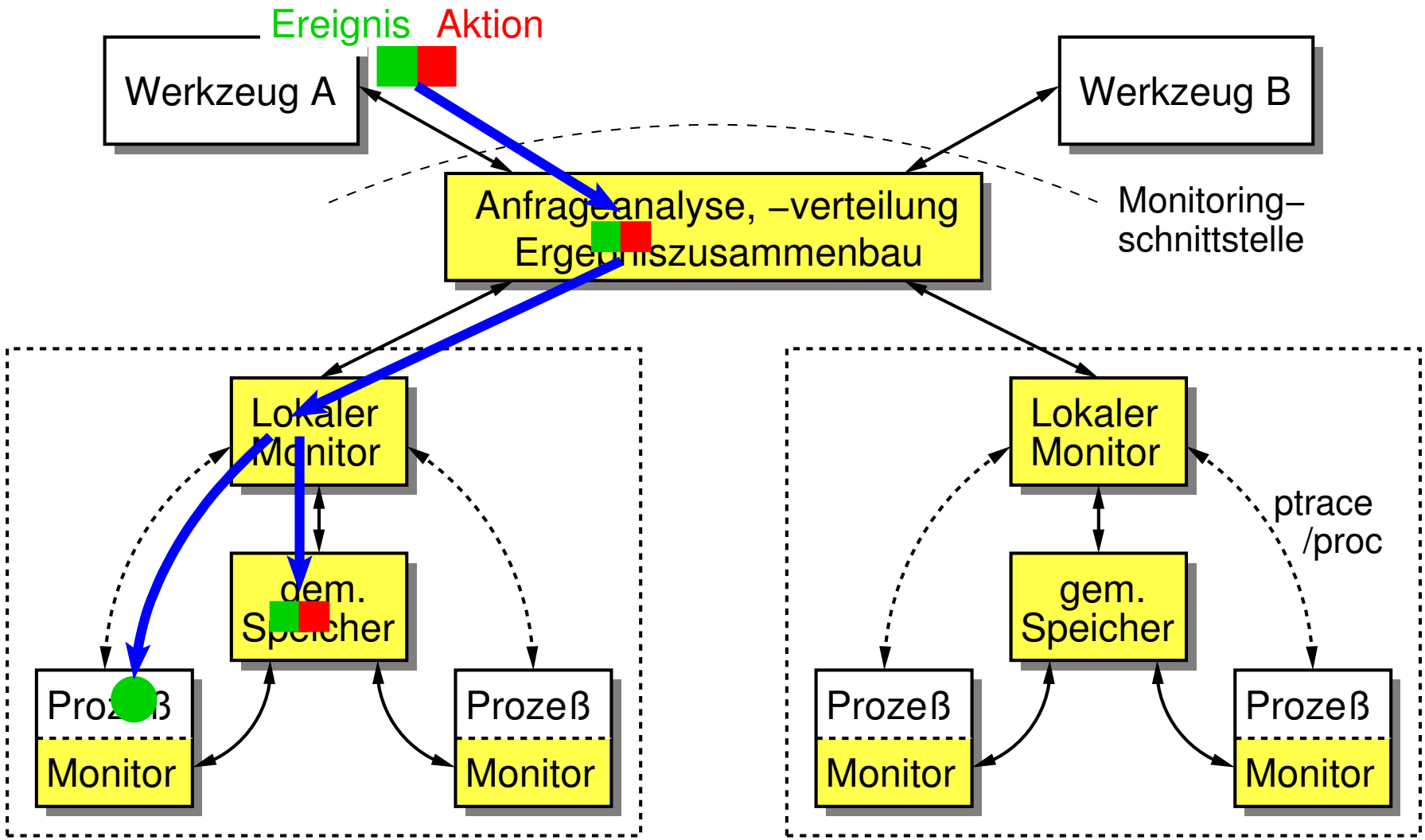
## Anfrageabarbeitung im OCM



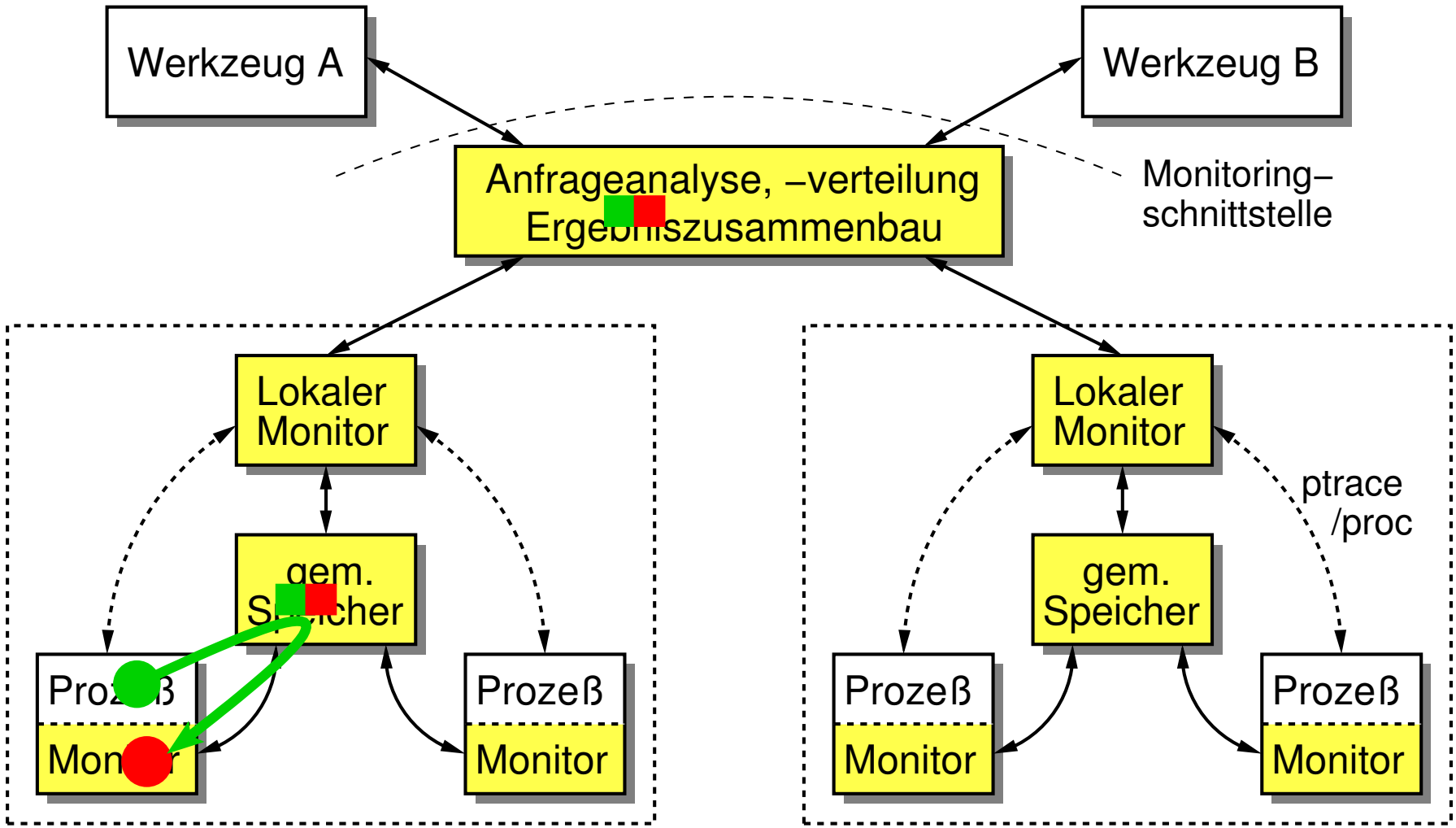
## Anfrageabarbeitung im OCM



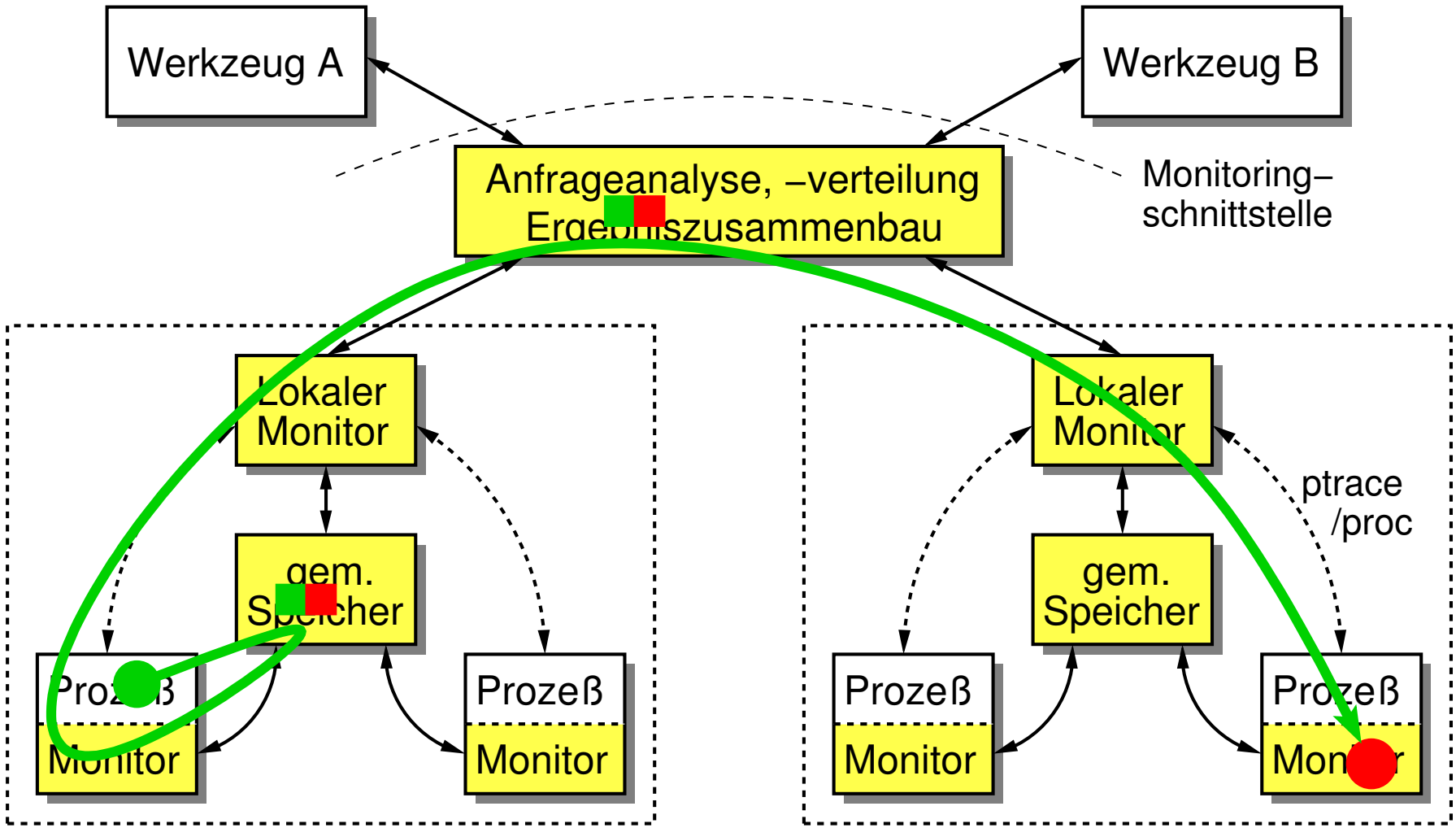
## Anfrageabarbeitung im OCM



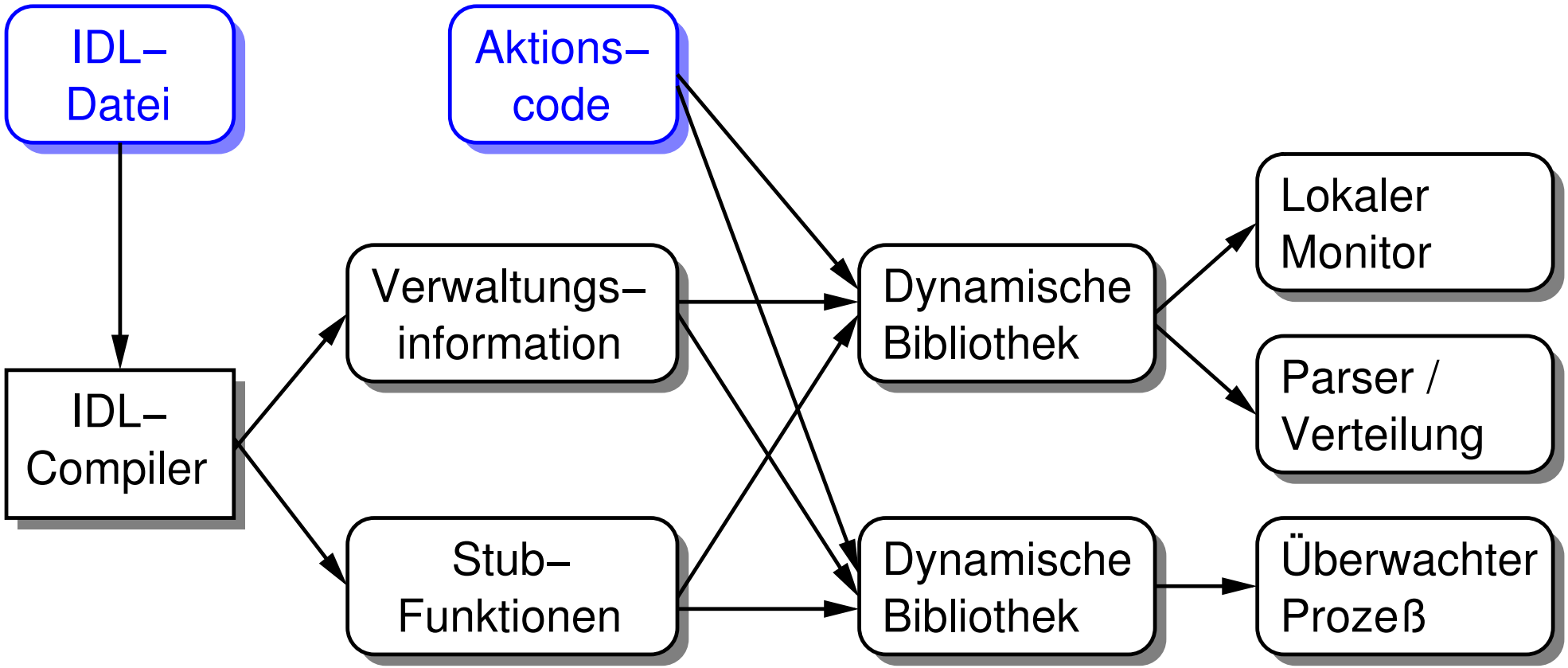
## Anfrageabarbeitung im OCM



## Anfrageabarbeitung im OCM



## Erweiterung des OCM um neue Aktionen



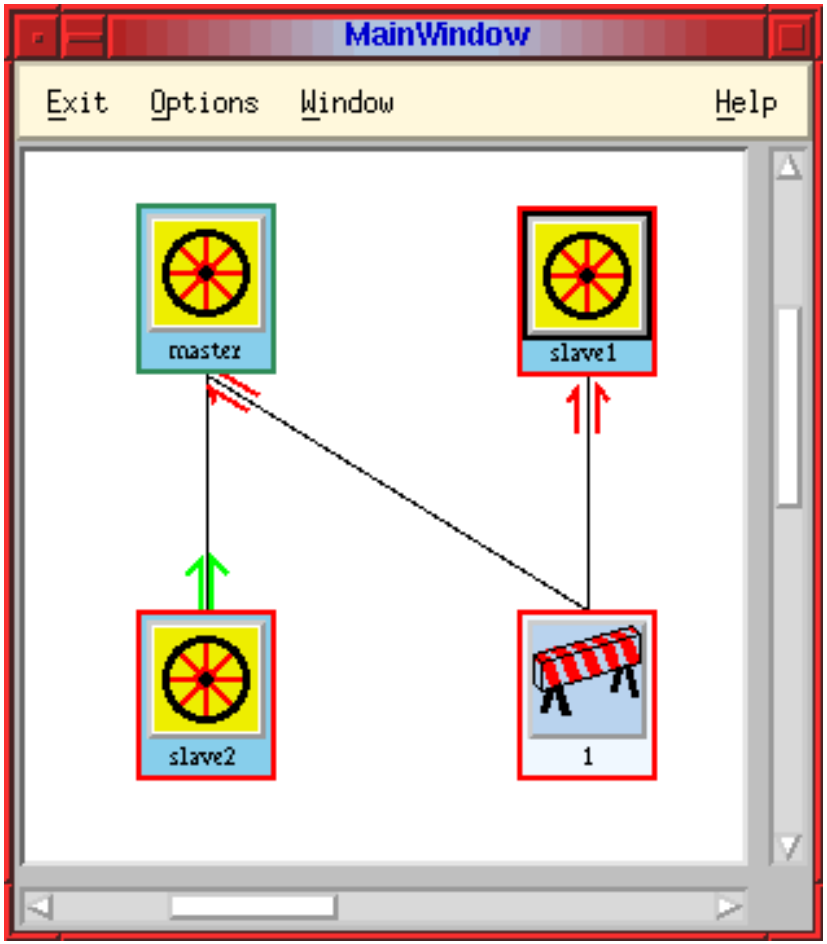
## Motivation: Kombination von Werkzeugen

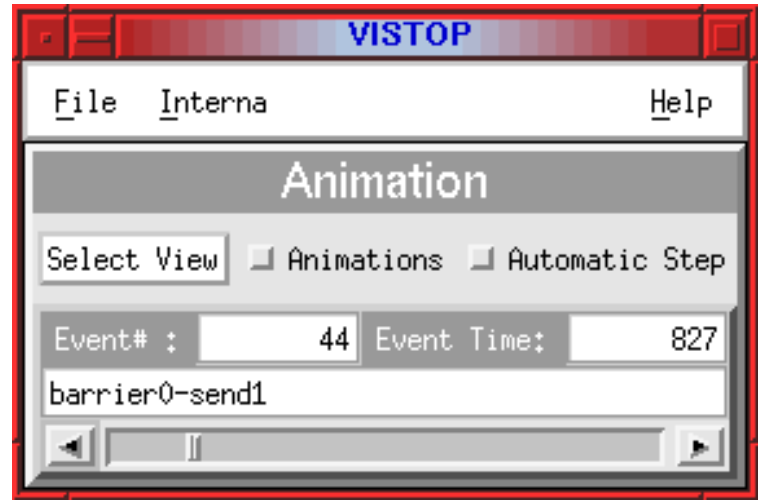
- z.B. Visualisierer + Debugger
  - Überblick über Ablaufverhalten
  - detaillierte Untersuchung

## Problem:

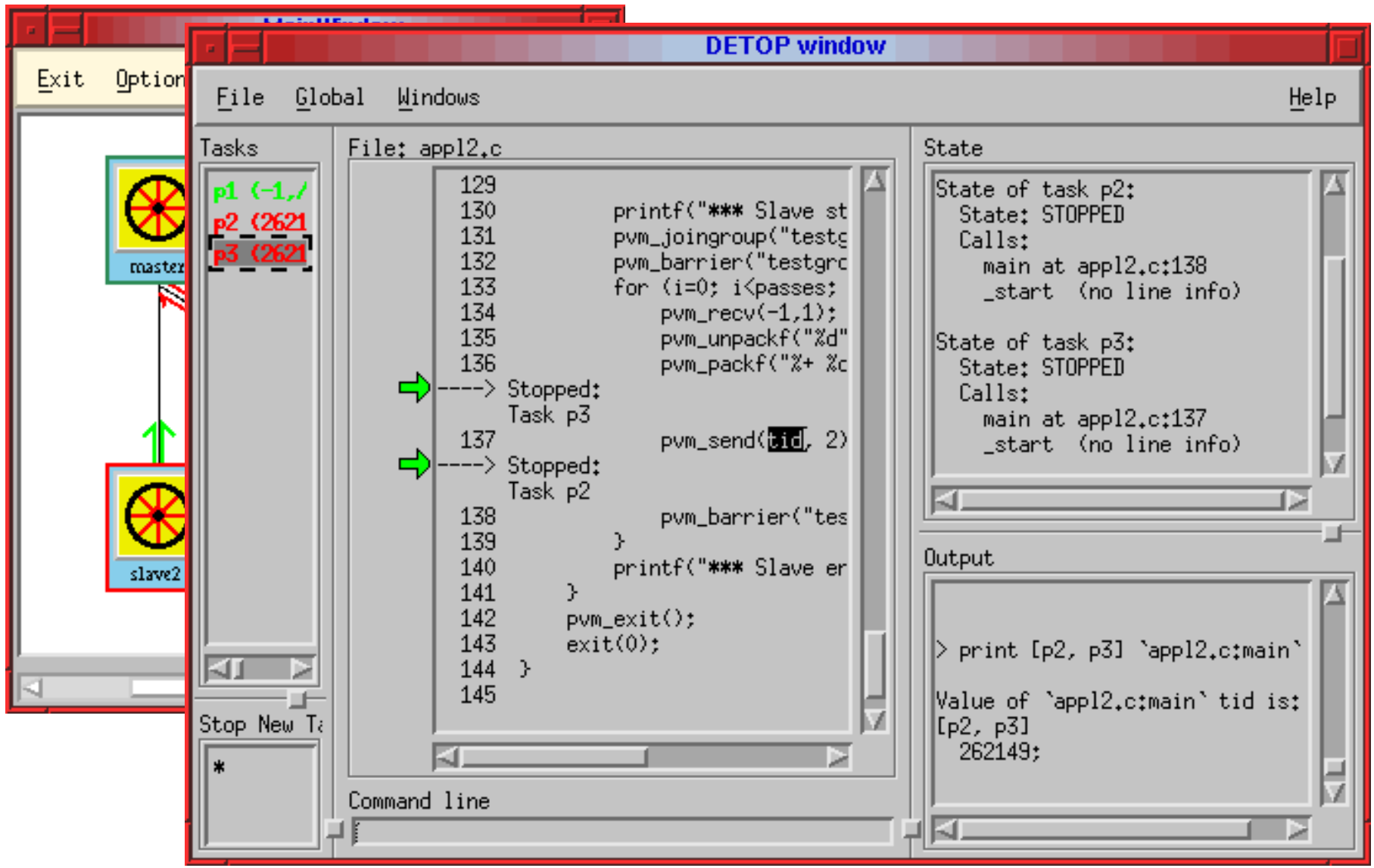
- Daten- und Verhaltensintegration notwendig
- Keine Abhängigkeit der Werkzeuge
- Möglichst keine Modifikation der Werkzeuge

# 3. Interoperable Werkzeuge

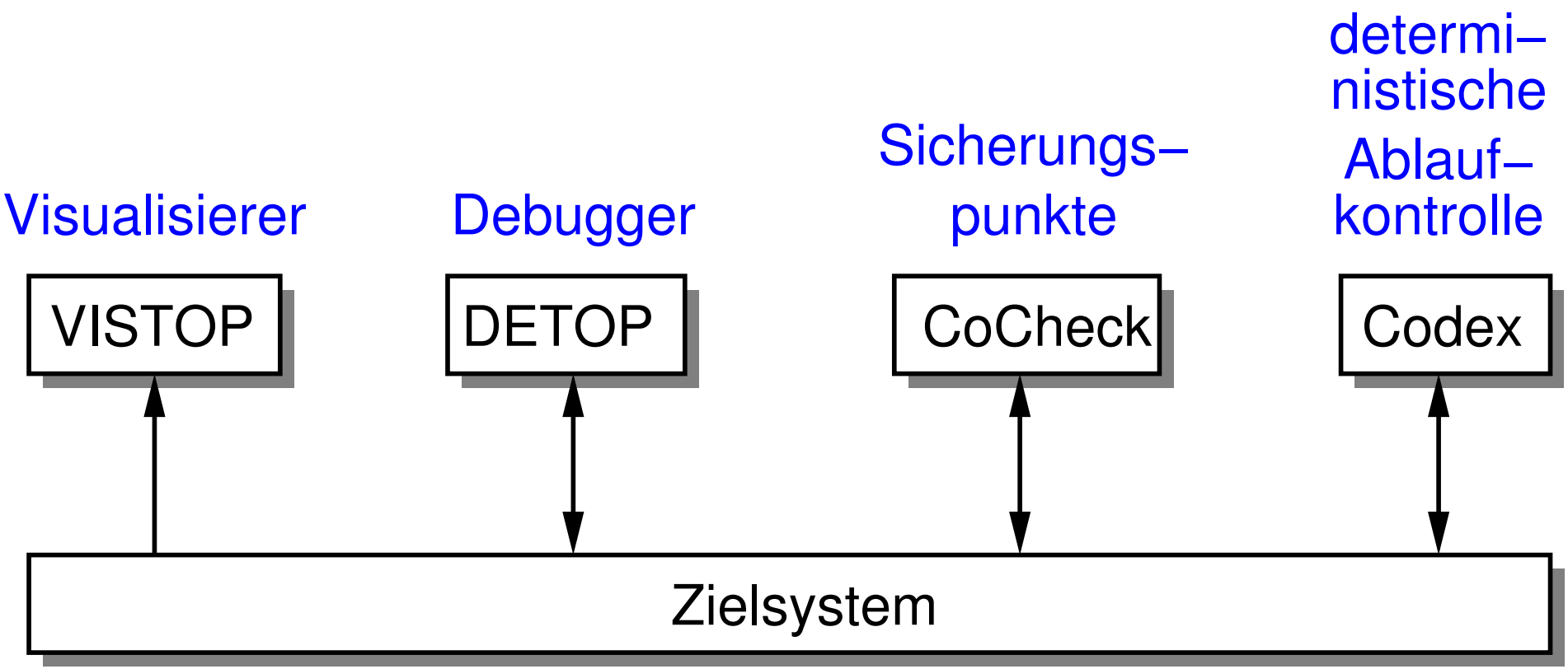


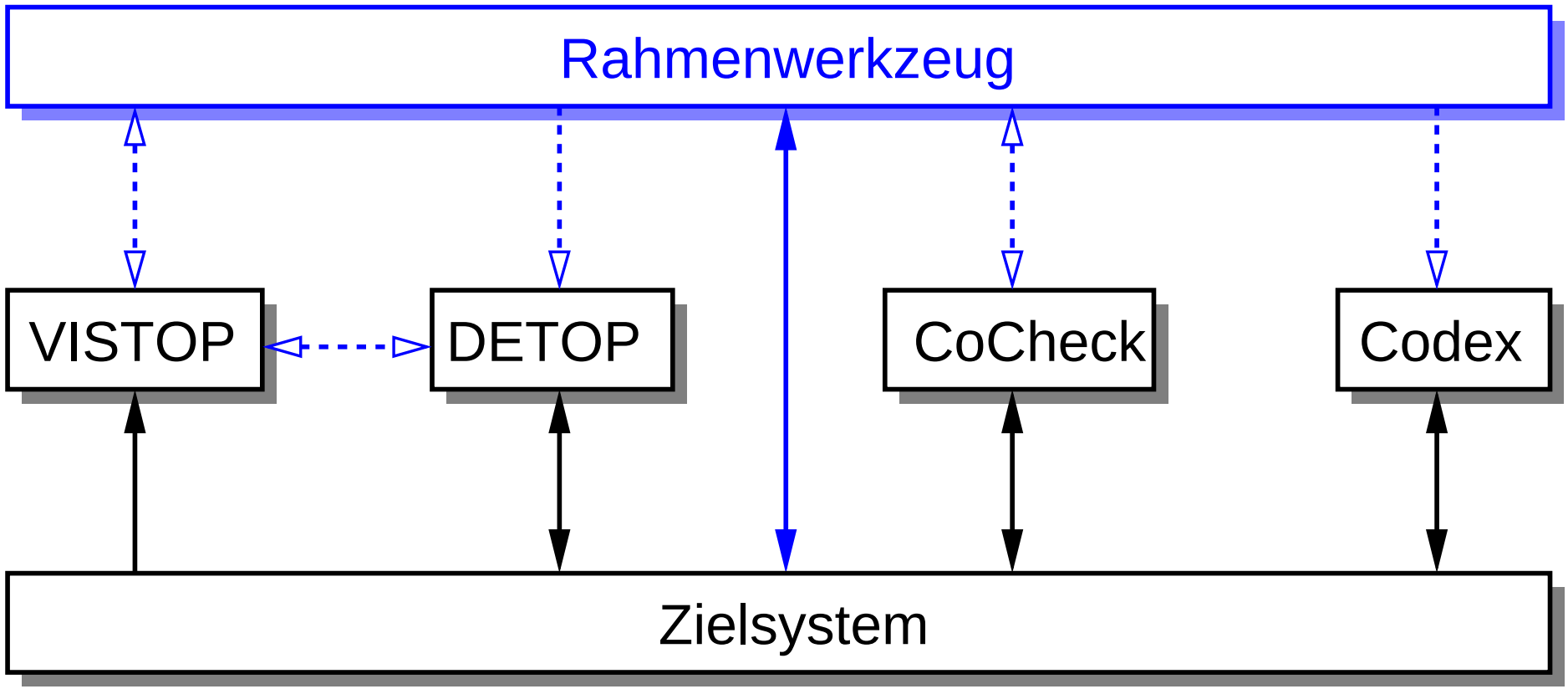
# 3. Interoperable Werkzeuge



## Aufbau der Werkzeugumgebung



## Aufbau der Werkzeugumgebung



## Problem: Konflikte bei Zugriff auf gemeinsames Zielsystem

- Instrumentierung / Betriebssystem-Schnittstellen
  - Gelöst durch gemeinsames Monitorsystem (OMIS/OCM)
- Explizite Modifikation des Zielsystemzustands
  - DETOP hält Zielprozeß an
    $\Rightarrow$ VISTOP muß von der Änderung erfahren
  - CoCheck ändert Task-ID der Prozesse
    $\Rightarrow$ andere Werkzeuge dürfen Änderung nicht sehen

## Erkennung und Auflösung von Konflikten

- Lösungsansatz:
  - Zielsystemobjekt = Menge von Zustandskomponenten mit Schreib-/Leseoperationen
  - Operationen mit Attribut: Beobachtungsebene
  - Konflikterkennung und -auflösung anhand der Attribute
- Konfliktauflösung:
  - Beobachtung von Schreibzugriffen
  - Abfangen von Lesezugriffen
- Zusätzlich berücksichtigt:
  Überlappung von Zustandskomponenten, Erweiterbarkeit

# 3.2. Verhaltensintegration

## Ziel: Kooperation zwischen Werkzeugen

- Z.B. synchronisierte Auswahl, konsistente Darstellung
- Im weiteren Sinne: Steuerung durch Rahmenprogramm

## Allgemeinere Fragestellung:

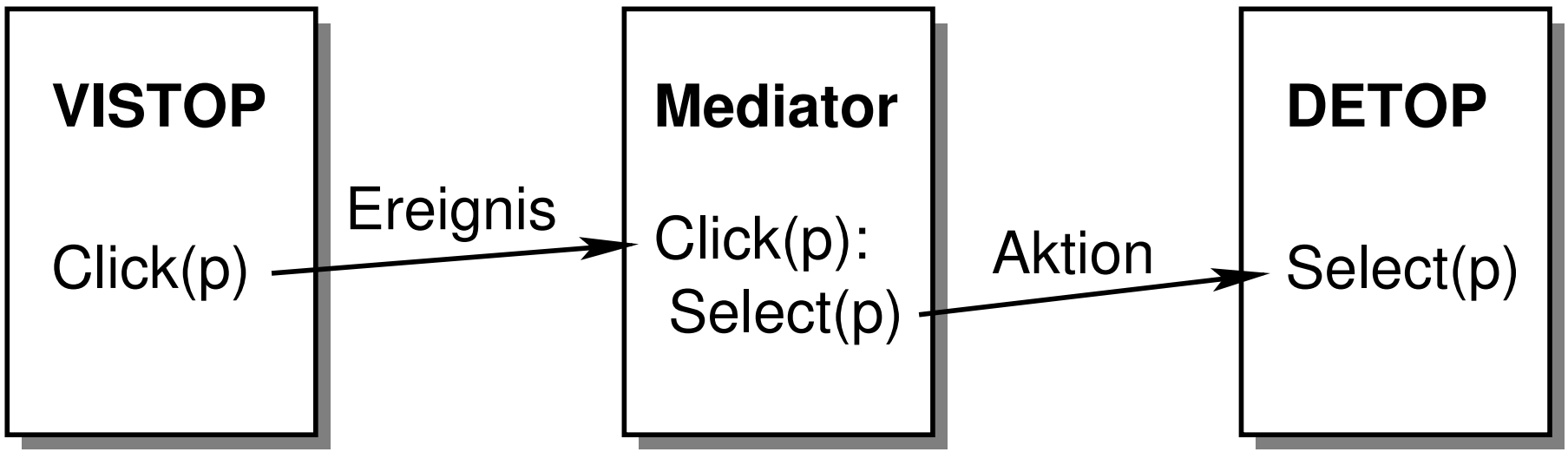
- Erstellung von Software-Umgebungen aus vorhandenen, aktiven Komponenten ( = Programme mit Benutzerinteraktion)

## Derzeit bester Ansatz:

- Mediatoren (bzw. Interaktoren)

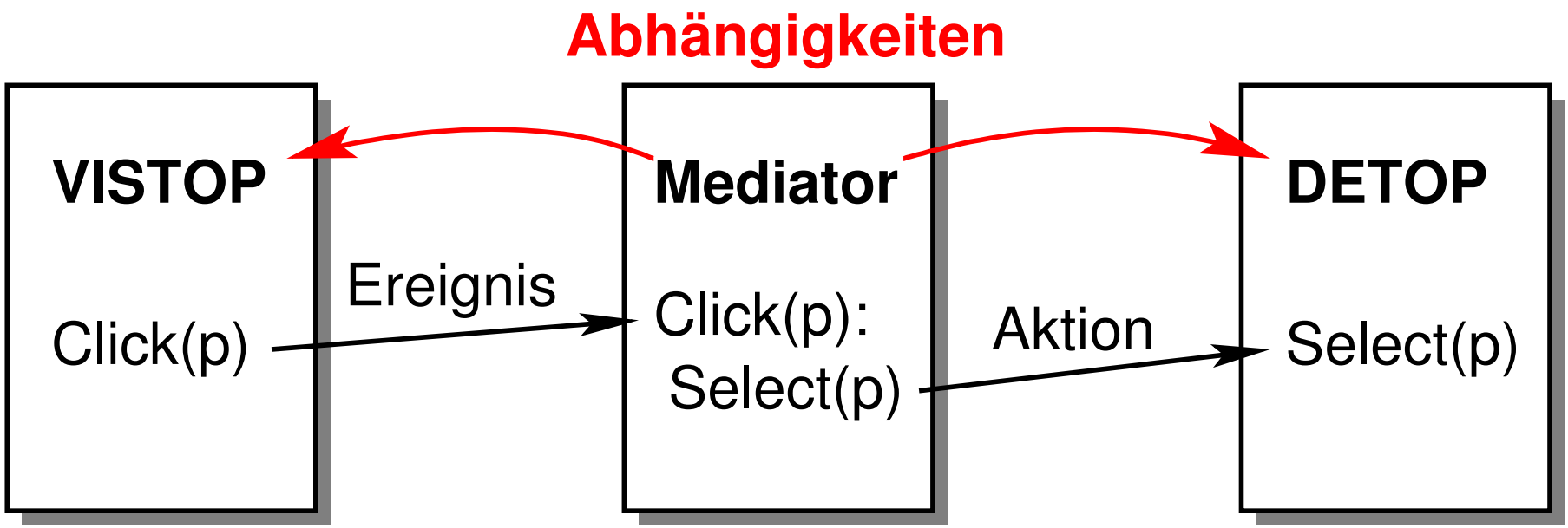
## Mediatoren / Interaktoren

- Mediator ist zusätzliche Komponente
- Überwacht Ereignisse, führt zugehörige Aktionen aus

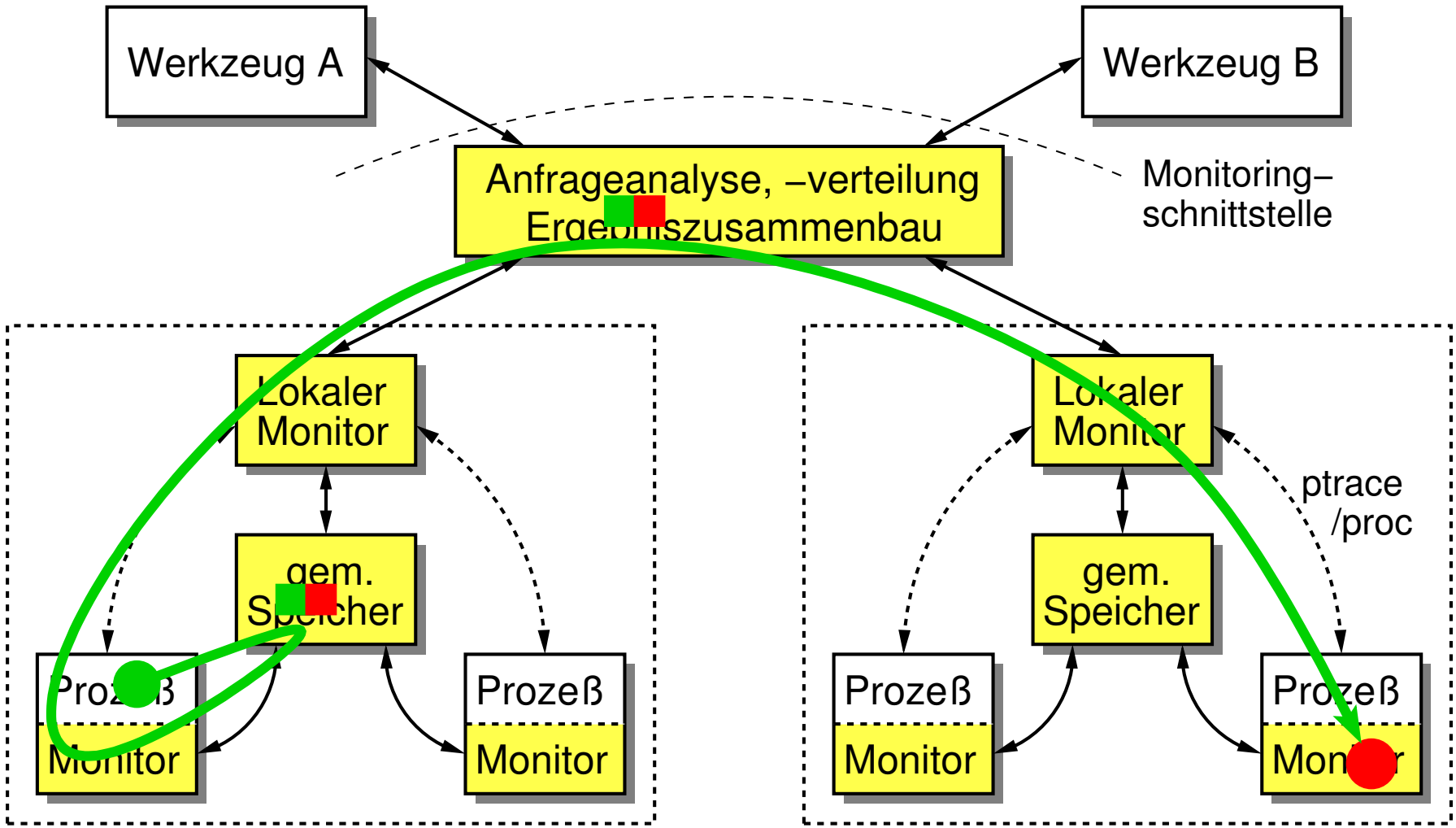


## Mediatoren / Interaktoren

- Mediator ist zusätzliche Komponente
- Überwacht Ereignisse, führt zugehörige Aktionen aus



## Anfrageabarbeitung im OCM



## Spezielle Eigenschaften des OCM:

- Überwachung von Ereig–nissen in Prozessen
- Aufruf von Aktionen in Prozessen

} ohne spezielle Vorbereitung dieser Prozesse

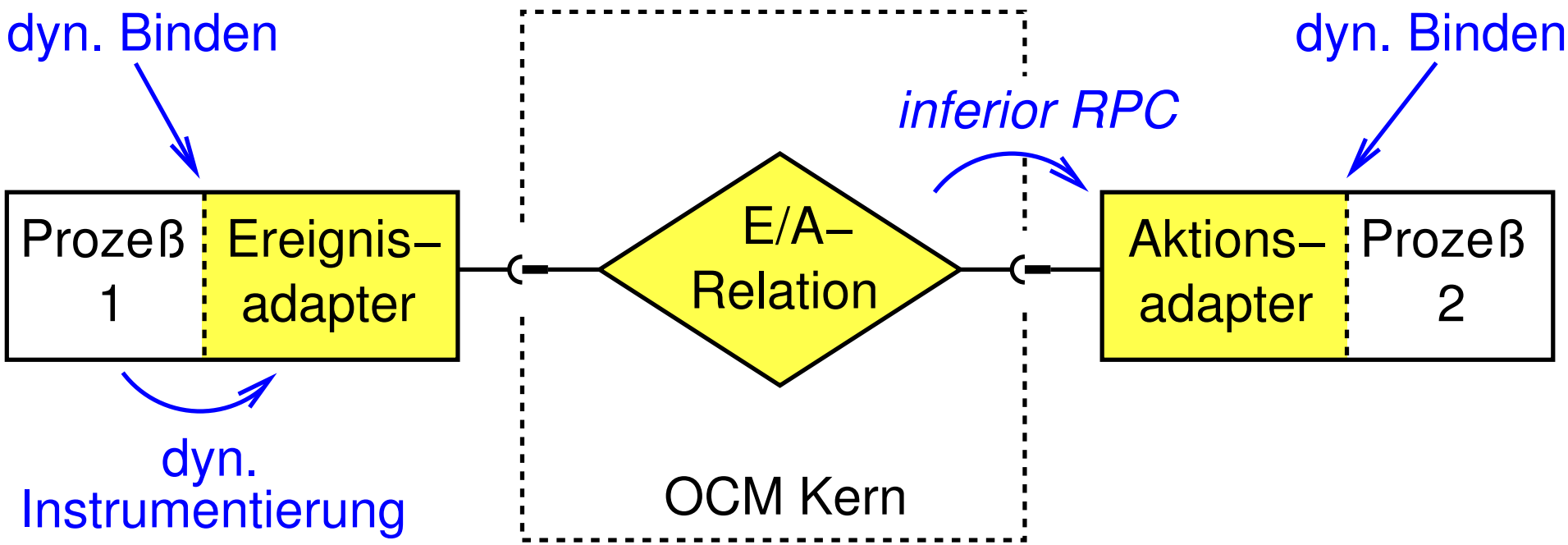
- Menge der Ereignisse und Aktionen zur Laufzeit erweiterbar

## Löst wichtiges Problem:

- Komponenten müssen a-priori Schnittstellen bereitstellen
  - Vorhandensein?
  - Vollständigkeit / Angemessenheit?
  - Abhängigkeit von Middleware

## Zur Realisierung:

- Kombination von dynamischer Instrumentierung, dynamischem Binden und *inferior RPC*



## Verallgemeinerung: Monitorsystem = dynamische Middleware

- Mediatorkonzept
  - 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

## Konsequenzen:

- Flexiblere Nutzung von Komponenten
  - Mächtigere (interne) Schnittstelle nutzbar
  - Einfache und effiziente Anpassung der Schnittstelle
- Vereinfachte Entwicklung von Komponenten
  - Kein expliziter Export von Schnittstellen
    - kein unnötiger ’Ballast’ im Code
    - keine Abhängigkeit von einer Middleware
- Schnittstelle an sich / Dokumentation muß vorhanden sein!

## Prototyp der Werkzeugumgebung

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

# 4. Zusammenfassung und Ausblick

## Laufzeit-Werkzeuge

- Beispiele, Forschungsthemen

## Monitorsysteme

- Universelle Schnittstelle
- Effiziente Implementierung

## Interoperable Laufzeit-Werkzeuge

- Motivation, Beispiel
- Datenintegration
- Verhaltensintegration

# 4. Zusammenfassung und Ausblick

## Künftige Arbeiten

- Automatische Leistungsanalyse interaktiver Anwendungen in Grid-Umgebungen (Esprit-Projekt CrossGrid)
- Interaktion zwischen Werkzeugen und Compilern
  - Debugging / Leistungsanalyse von HPF+ Programmen
  - Adaptive Laufzeitsysteme
- Konsolodierung des OCM, weitere Zielplattformen
- Ausbau der Konzepte zur Verhaltensintegration
  - Übertragung in andere Anwendungsbereiche
  - Integration mit Koordinationssprache