Interoperable Laufzeit-Werkzeuge und Online-Monitorsysteme

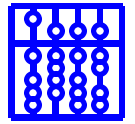
Roland Wismüller

LRR-TUM
Technische Universität München

Agenda

1. Eine Umgebung interoperabler Werkzeuge
   - ↪ Beispiel und Anforderungen
2. Datenintegration
3. Verhaltensintegration
4. Verhaltensintegration durch das Monitorsystem
5. Praktische Erfahrungen
6. Fazit

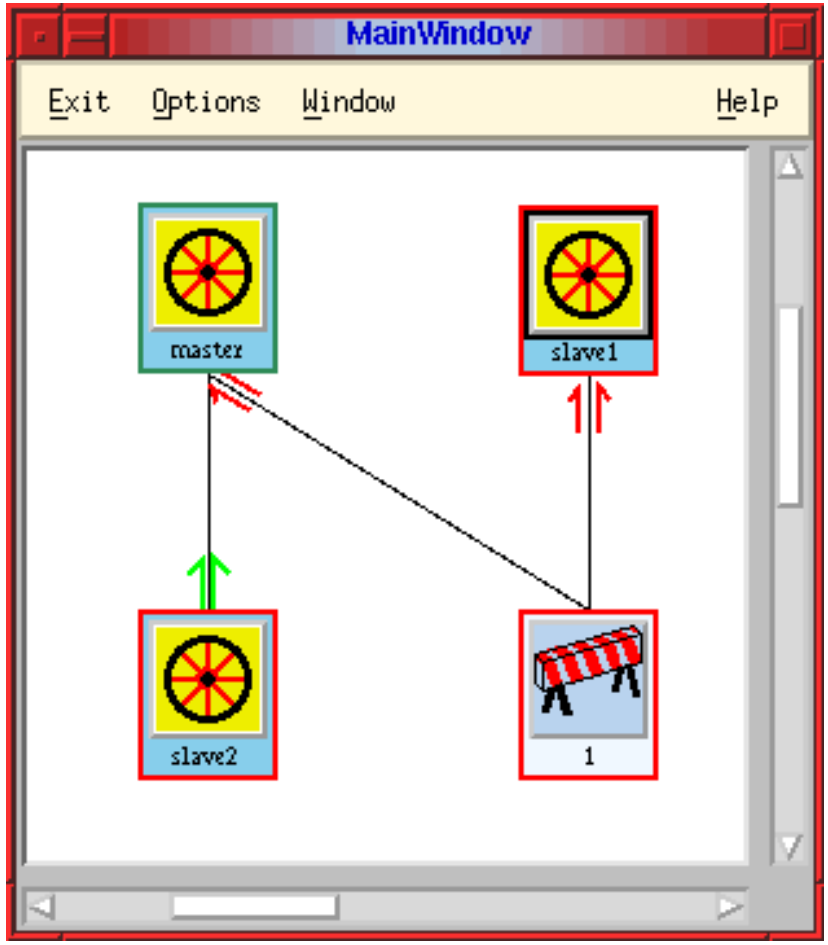
1. Eine Umgebung interoperabler Werkzeuge

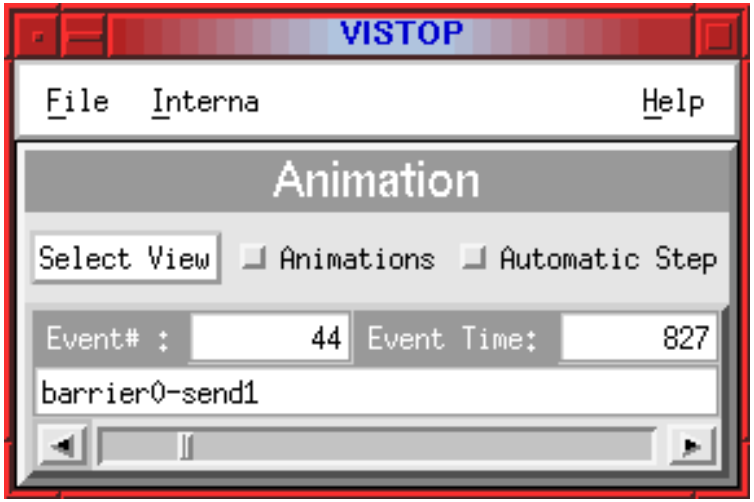


Motivation:

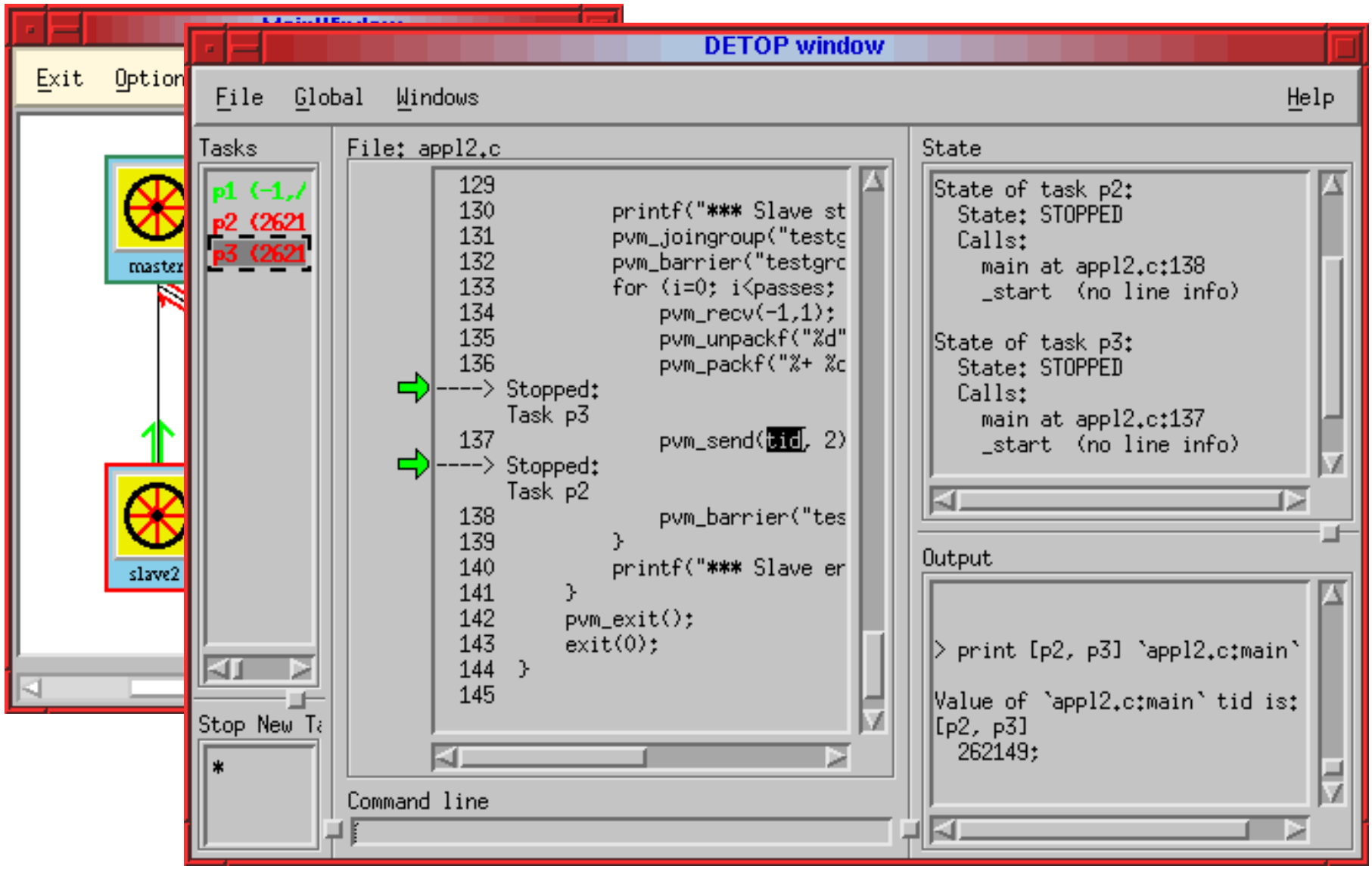
- Entwicklung von Online-Werkzeuge am LRR-TUM
  - Quellsprach-Debugger DETOP
  - Ablaufvisualisierer VISTOP
  - Leistungsanalyse, deterministische Ausführung, Sicherungspunkterstellung, ...
- Wunsch: Kombination von Werkzeugen
  - z.B. Visualisierer + Debugger
    - ☞ Überblick über Ablaufverhalten
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

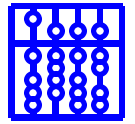
1. Eine Umgebung interoperabler Werkzeuge





1. Eine Umgebung interoperabler Werkzeuge





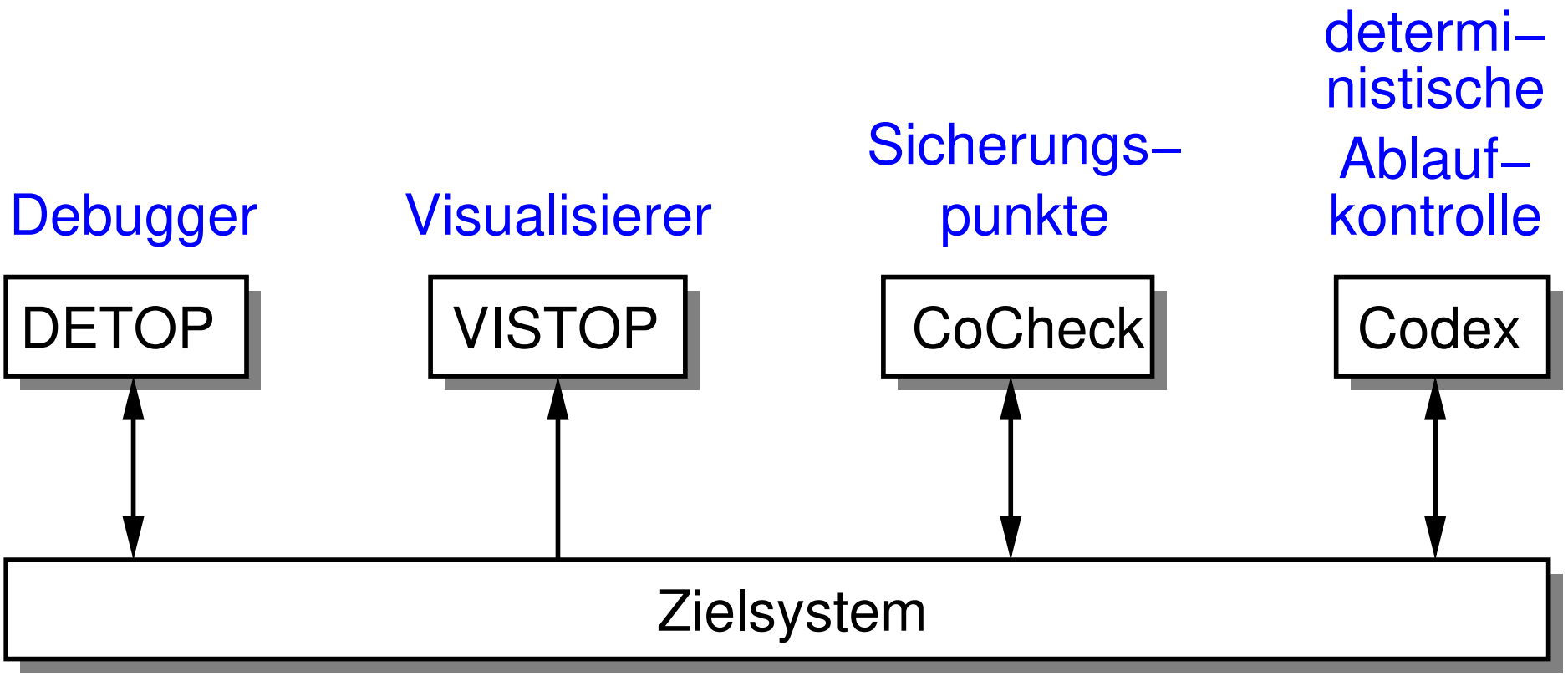
Schwierigkeit:

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

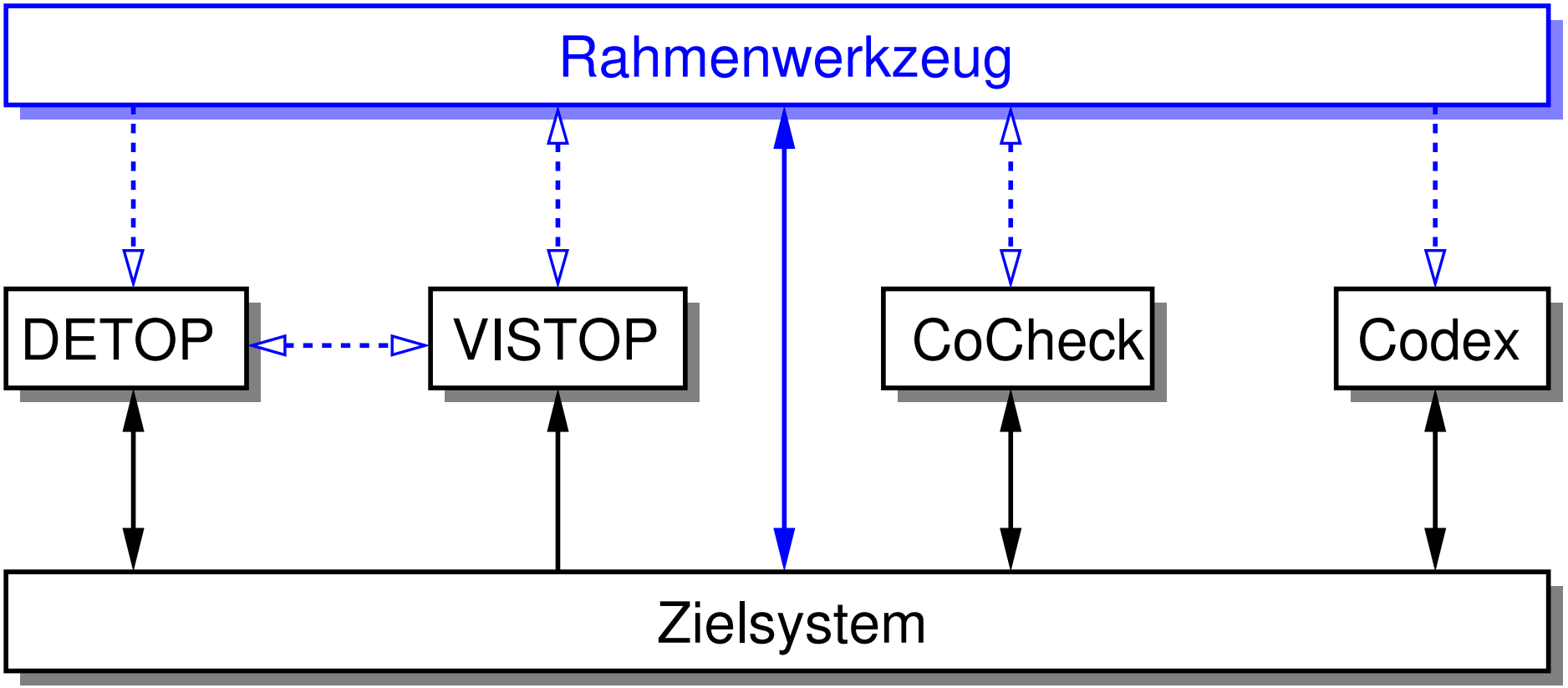
Vorgehensweise:

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

Aufbau der Werkzeugumgebung



Aufbau der Werkzeugumgebung



Ziel: Konfliktfreier Zugriff auf gemeinsame Daten

- I.d.R. nur erreichbar durch gemeinsame Zugriffsdienste
  - z.B. Betriebssystem, Datenbanksystem
- Hier: gemeinsame Monitorschnittstelle zur
  - Online-Beobachtung
  - Steuerung

  paralleler und verteilter Systeme
- Projekt OMIS (*On-line Monitoring Interface Specification*)
  - gestartet 1995 mit T. Ludwig und V. Sunderam, Emory

Ziele von OMIS

- Werkzeug- und Plattformunabhängigkeit
- Gleichzeitige Nutzung durch mehrere Werkzeuge

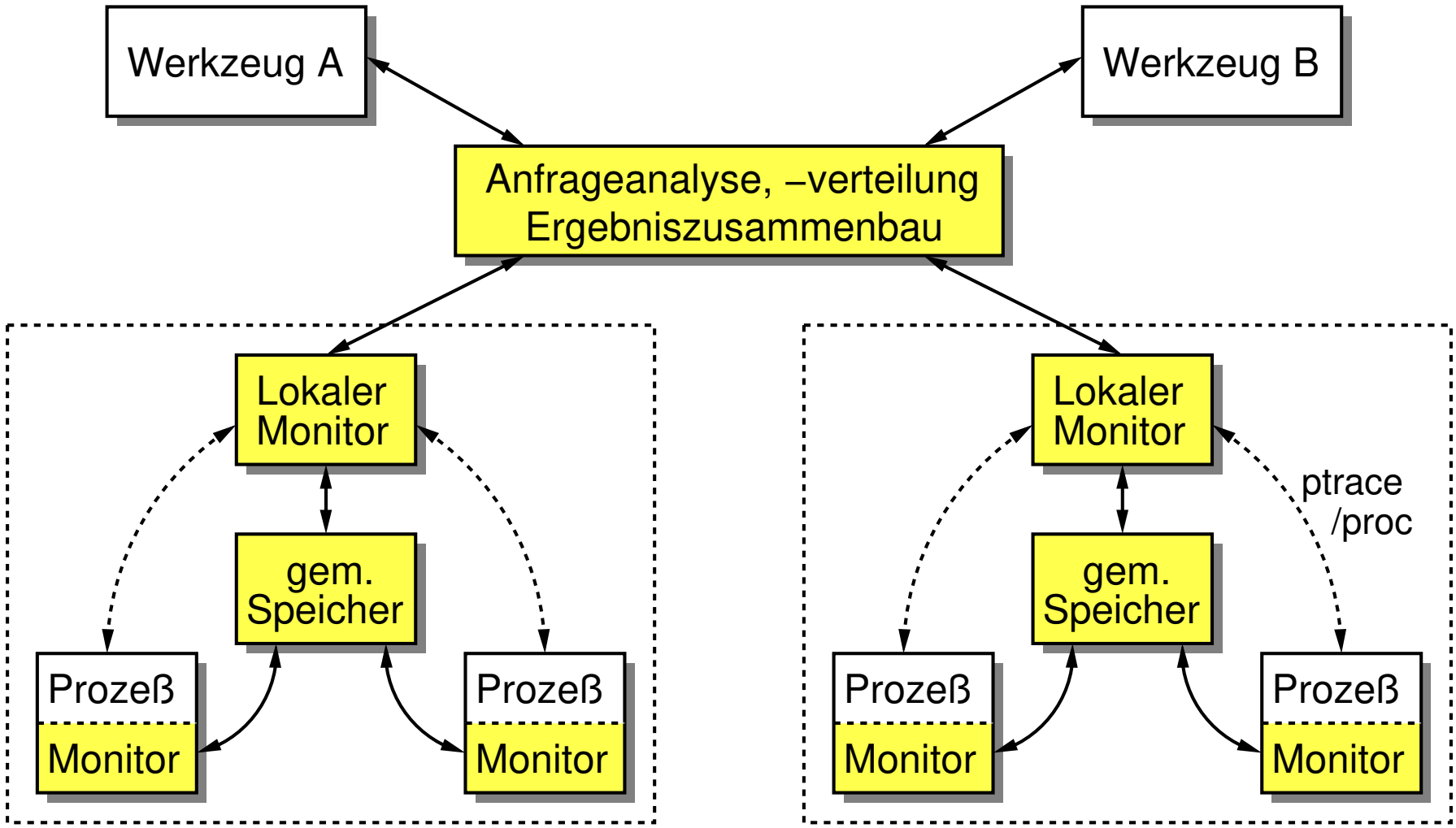
Konzepte

- Objektbasiertes Modell des Zielsystems
- Programmierbare Ereignis/Aktions-Relationen

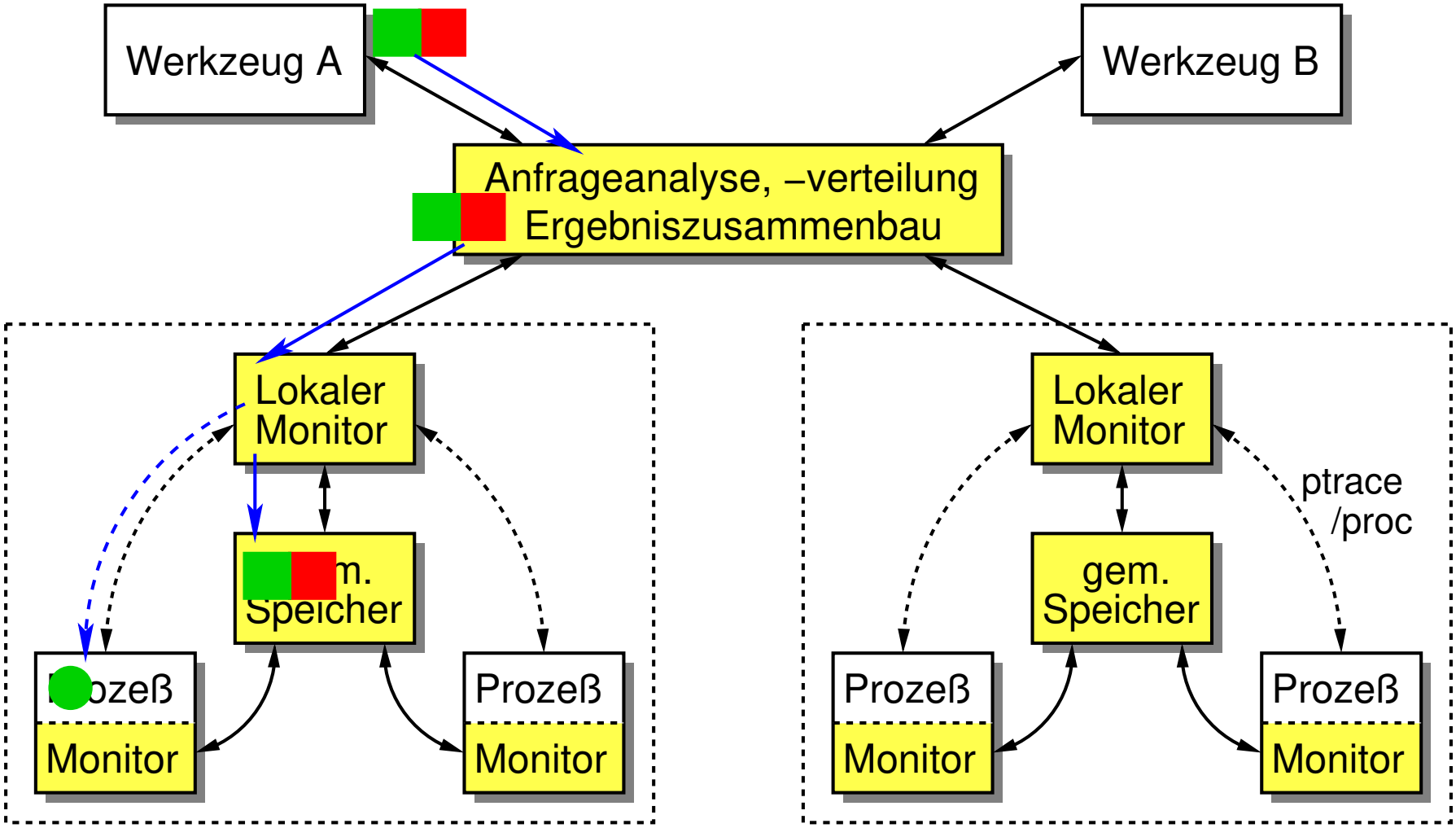
```
thread_started_lib_call([p_1],``pvm_send'') :
    thread_stop([$node])
    thread_get_backtrace([$thread],0)
```

- Objektmengen, Ortstransparenz
- Erweiterbarkeit

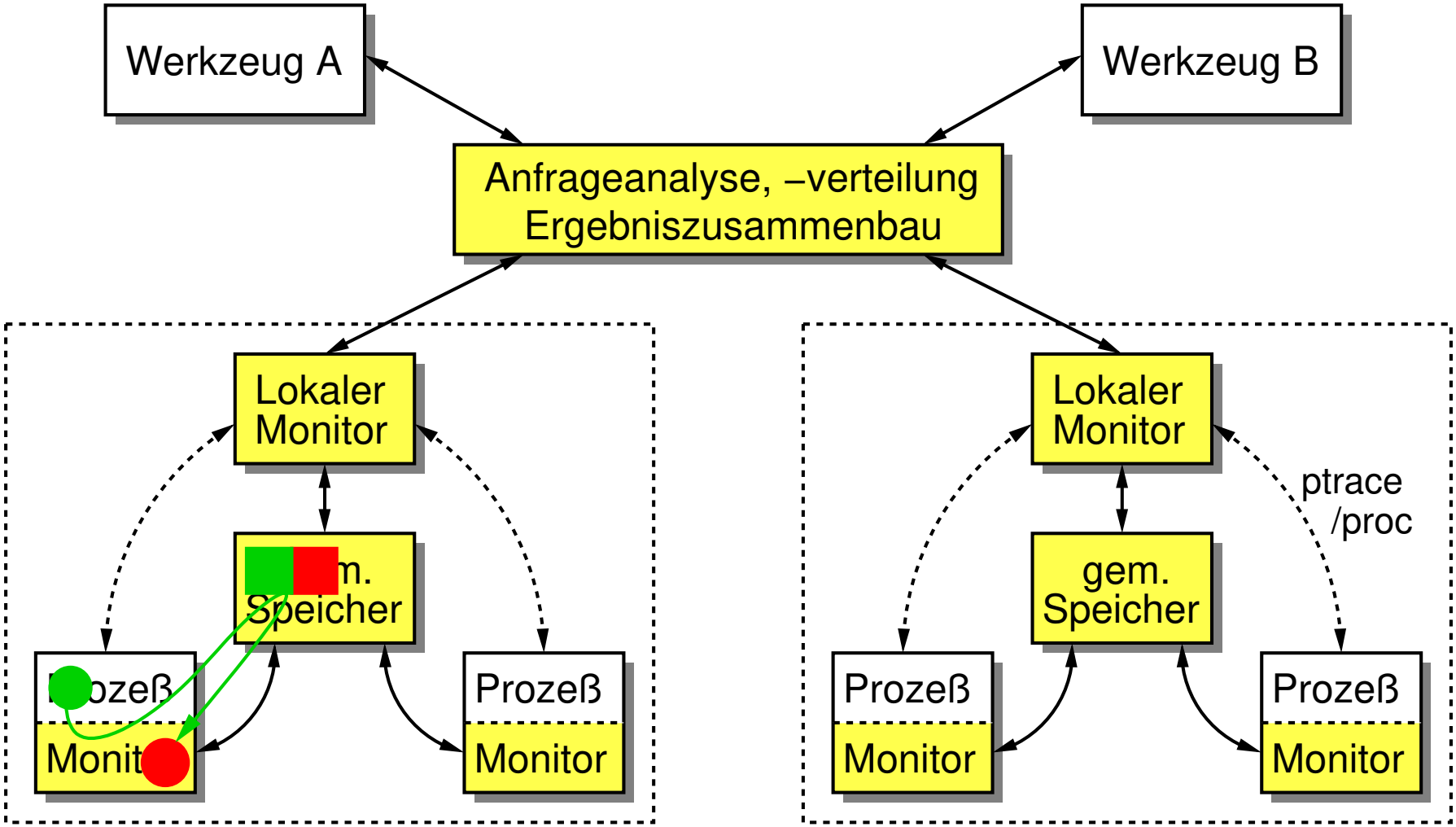
Aufbau des OCM (*OMIS Compliant Monitoring system*)



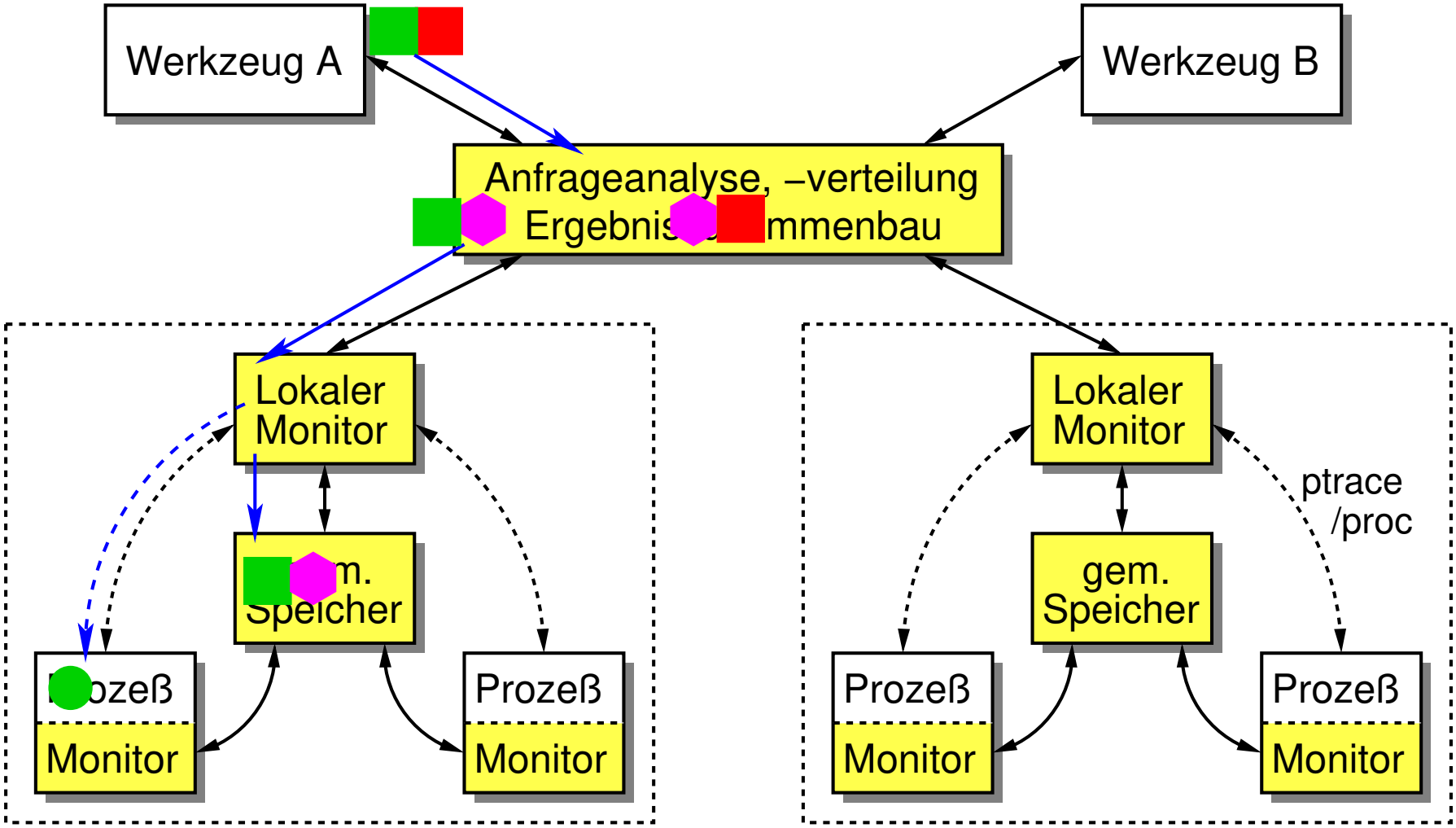
Anfrageverteilung und -abarbeitung im OCM



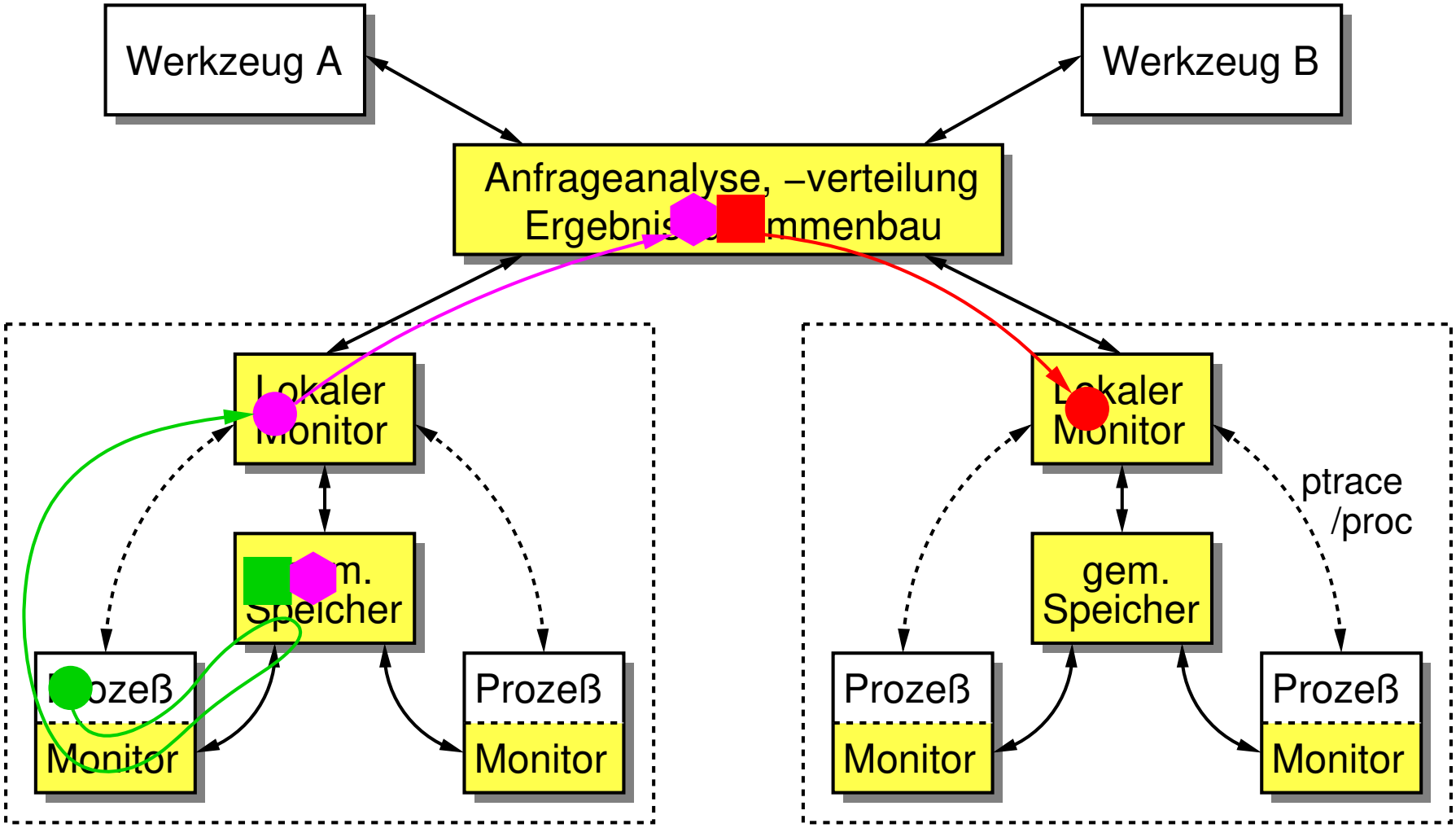
Anfrageverteilung und -abarbeitung im OCM



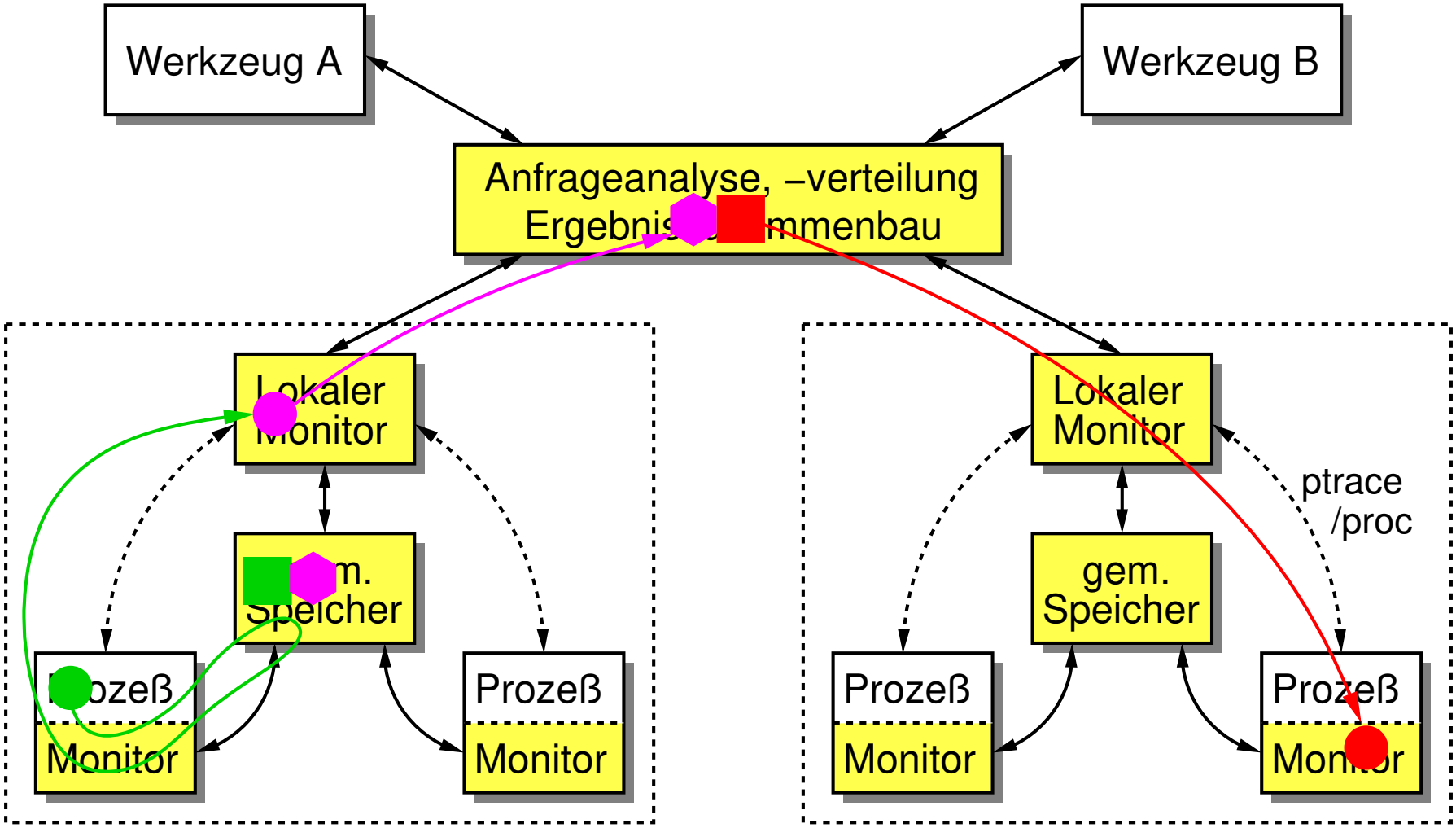
Anfrageverteilung und -abarbeitung im OCM



Anfrageverteilung und -abarbeitung im OCM



Anfrageverteilung und -abarbeitung im OCM



Konflikte

- Ursache: Verändern des Zielsystemzustands
- Beispiele:
  - 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 der Konflikte, ohne daß Werkzeuge Wissen übereinander benötigen?

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
- Probleme: Überlappung von Zustandskomponenten, Erweiterbarkeit

Ziel: Kooperation zwischen den Komponenten

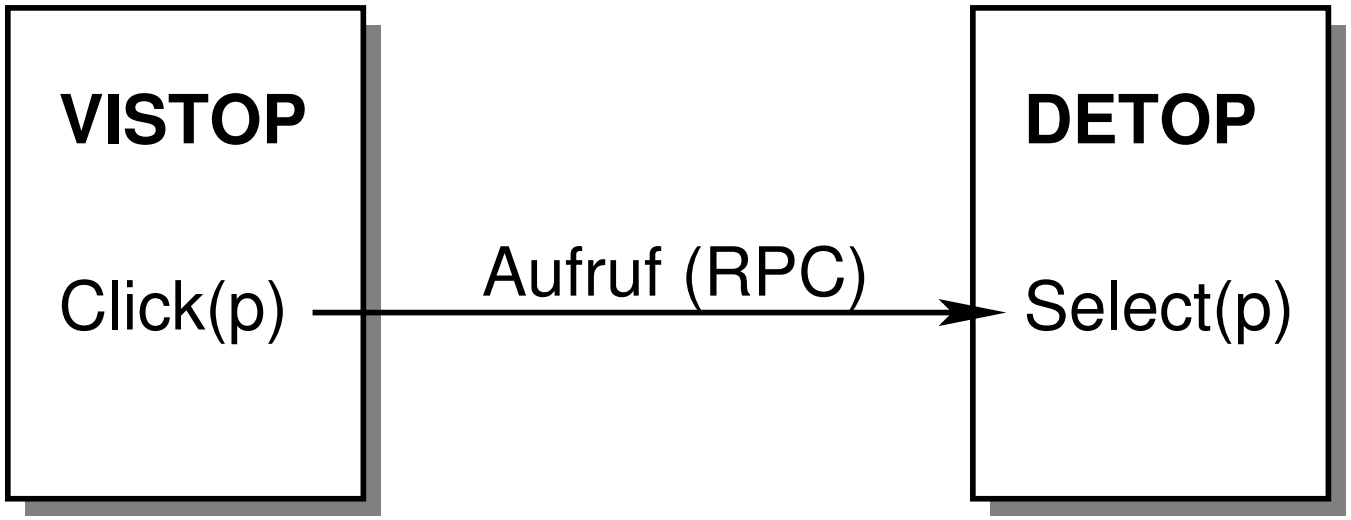
- z.B. synchronisierte Auswahl, konsistente Darstellung
- Im weiteren Sinne: Steuerung durch Rahmenprogramm

Allgemeine Fragestellung:

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

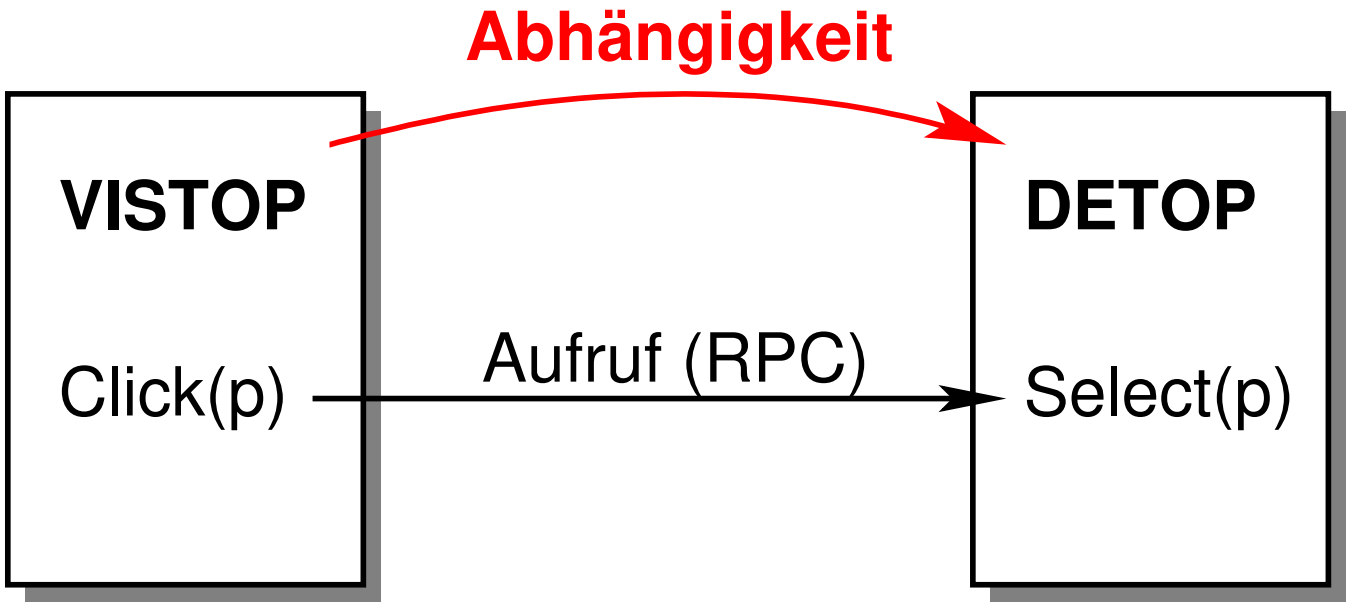
Feste Verdrahtung

- Z.B. RPC, CORBA, PCTE



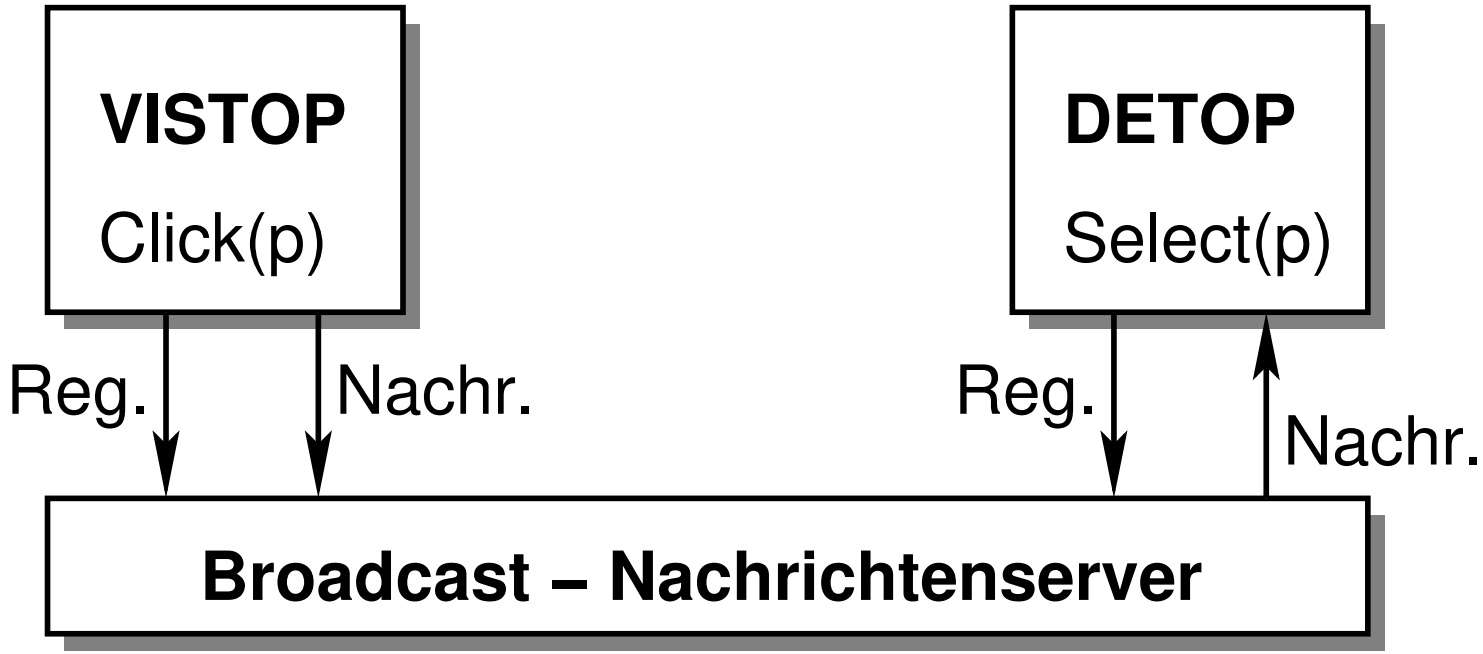
Feste Verdrahtung

- Z.B. RPC, CORBA, PCTE



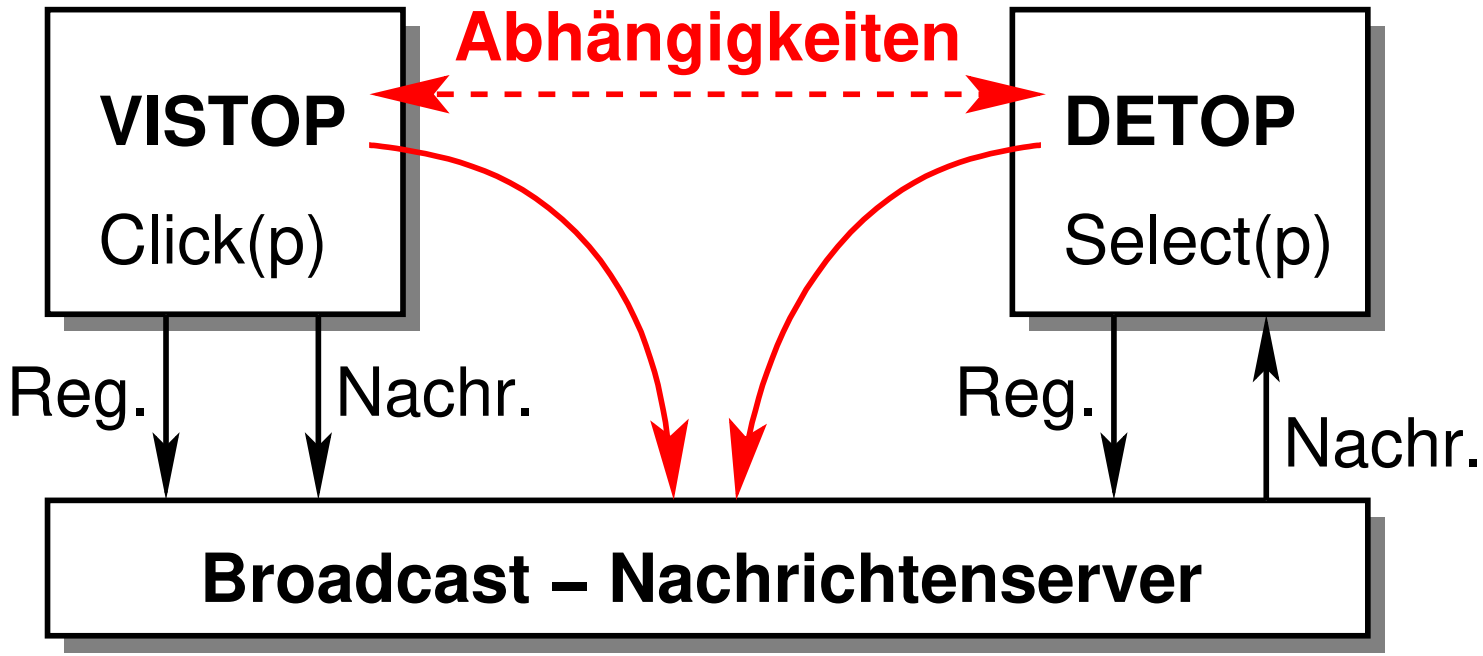
Broadcast-Nachrichtenserver

- Z.B. ToolTalk, SoftBench



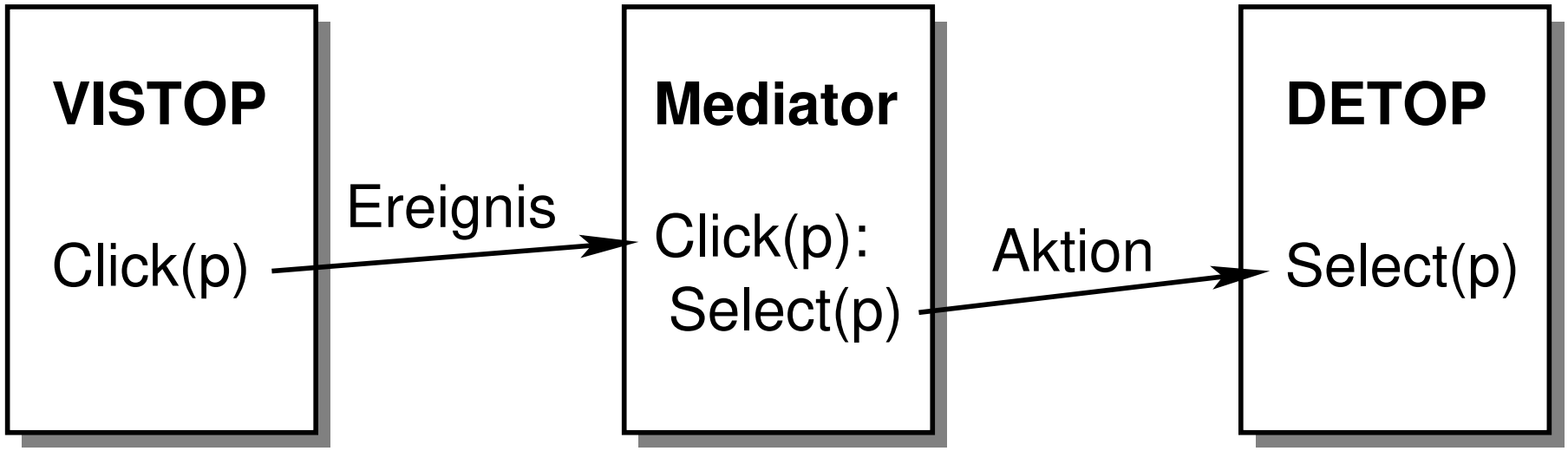
Broadcast-Nachrichtenserver

- Z.B. ToolTalk, SoftBench



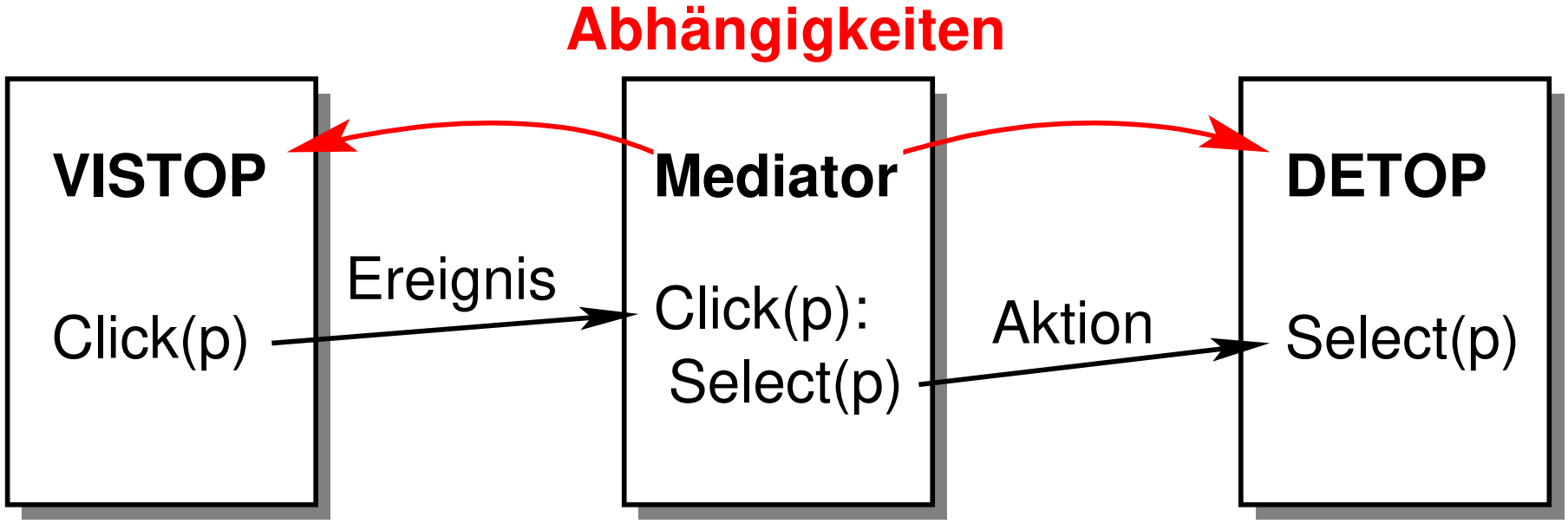
Mediatoren / Interaktoren

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



Mediatoren / Interaktoren

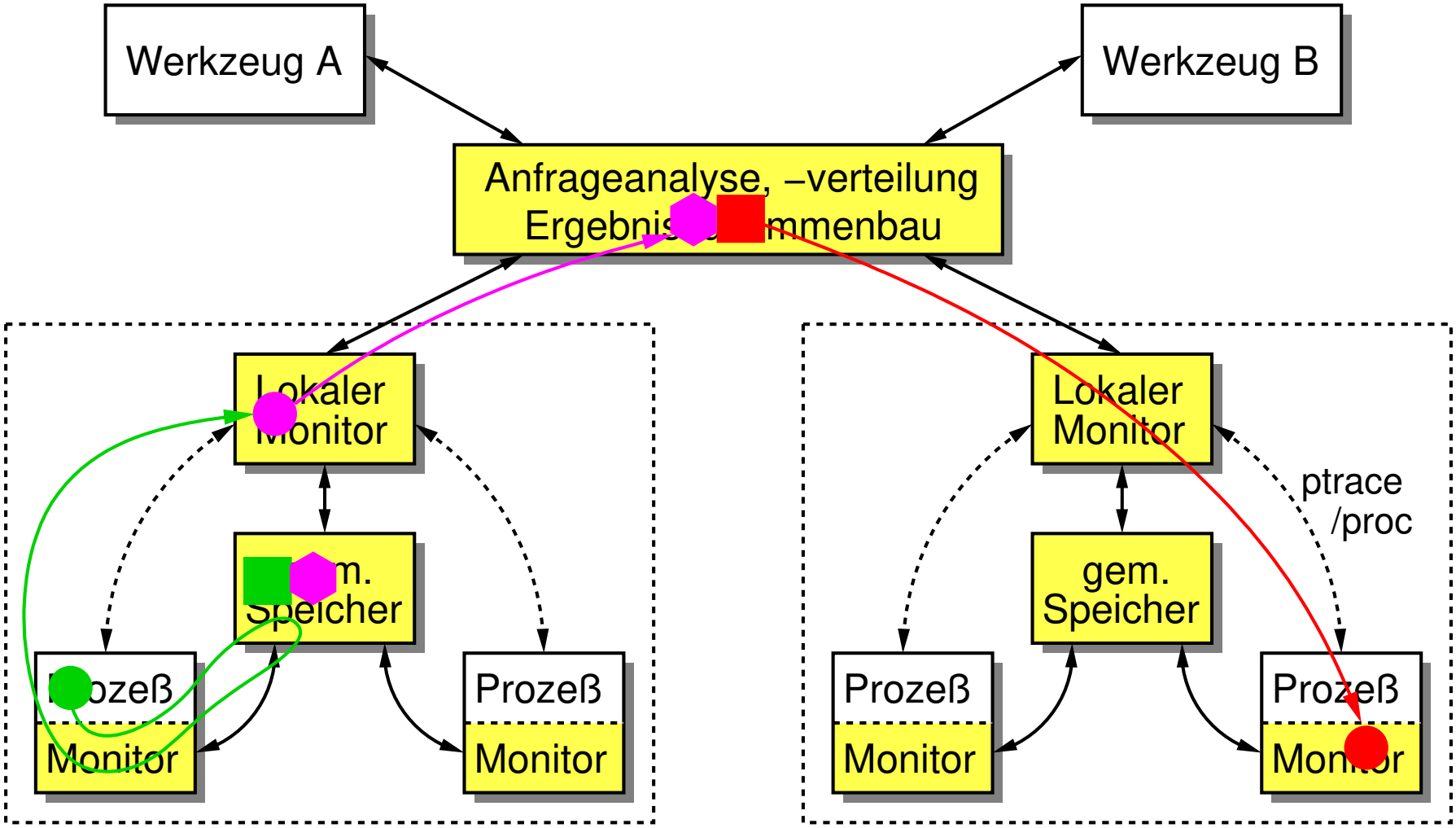
- Z.B. ToolBus, Manifold, CHAIMS, TIL



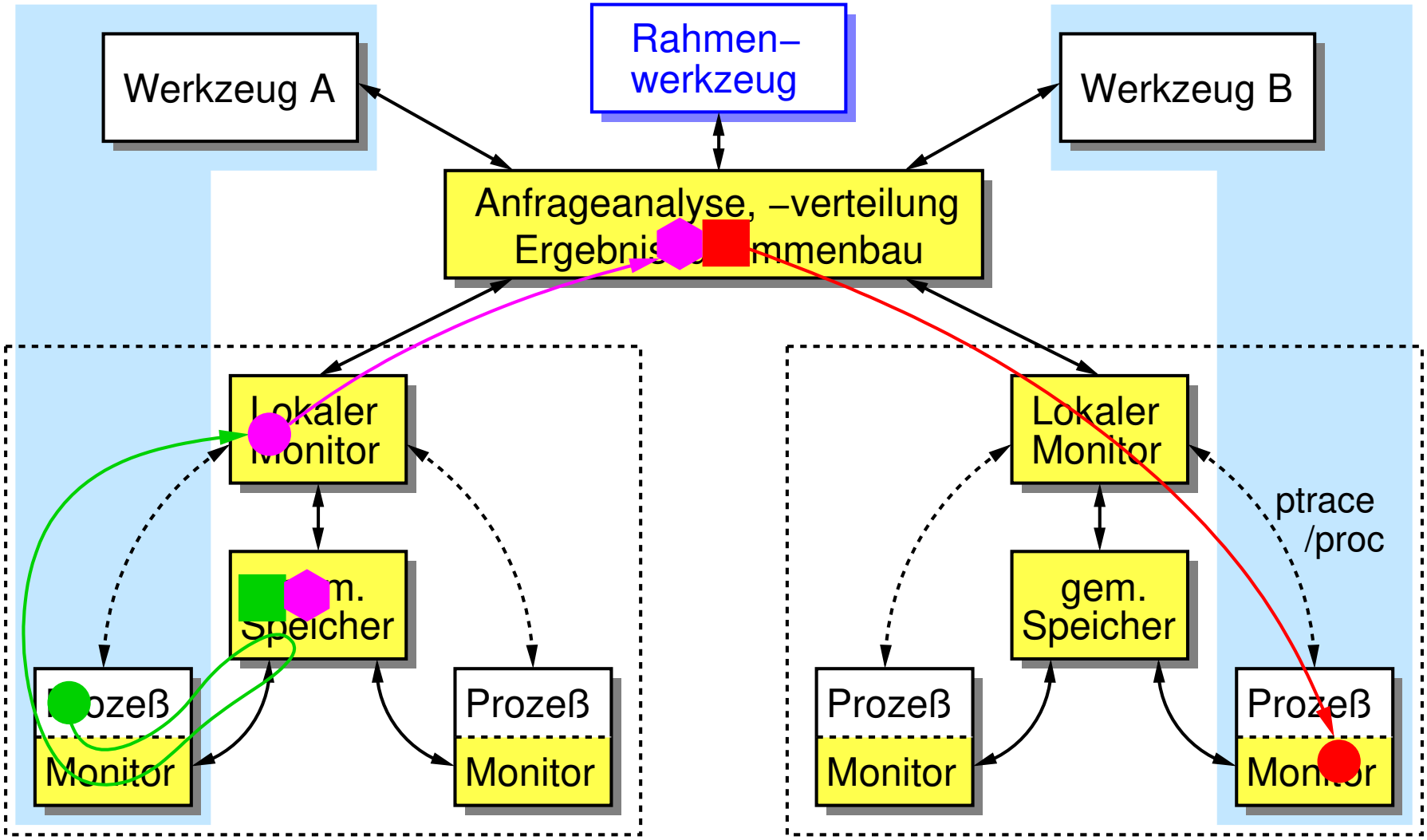
Problem:
Komponenten müssen a-priori Schnittstellen bereitstellen

- 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

Anfrageverteilung und -abarbeitung im OCM



Der OCM als Mediator

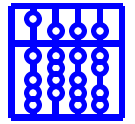


Spezielle Eigenschaften des OCM:

- Überwachung von Ereignissen in Prozessen
- Aufruf von Aktionen in Prozessen

} ohne spezielle Vorbereitung dieser Prozesse

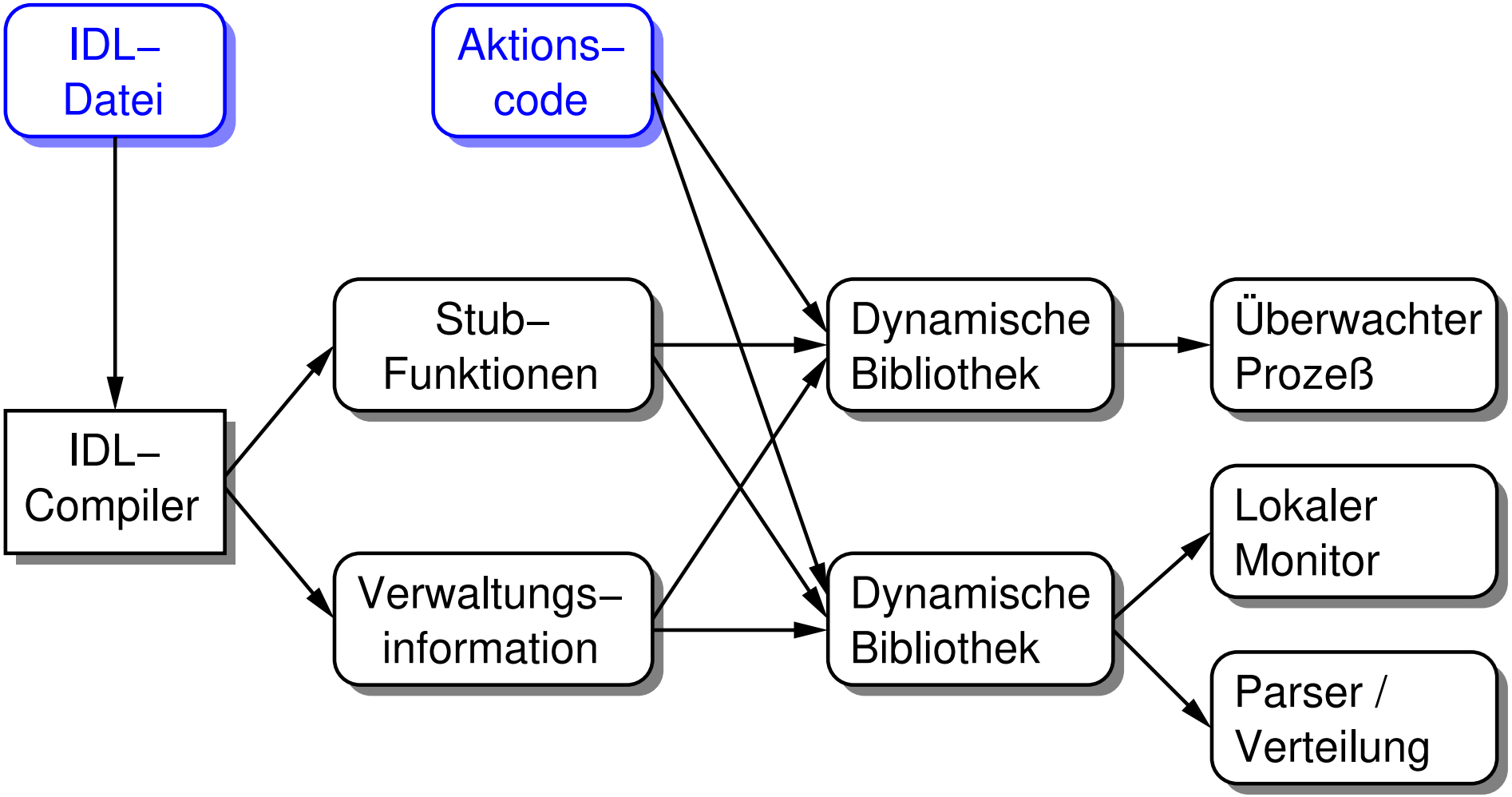
- Realisierung durch
  - dynamische Instrumentierung (*dyninst API*)
    - Kooperation mit B. P. Miller, UW Madison
  - *inferior RPC*
    - Stack- und Befehlszählermanipulation
    - synchronisiert durch lokales Ereignis



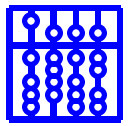
Spezielle Eigenschaften des OCM:

- Dynamische Erweiterbarkeit der OCM-Schnittstelle um
  - neue Ereignisse
  - neue Aktionen
- Definition der Komponentenschnittstellen zur Laufzeit
- Realisierung durch
  - Schnittstellen-Beschreibungssprache
  - dynamisches Binden
  - *inferior RPC*

Einbinden neuer Aktionen



4.1. Resultat

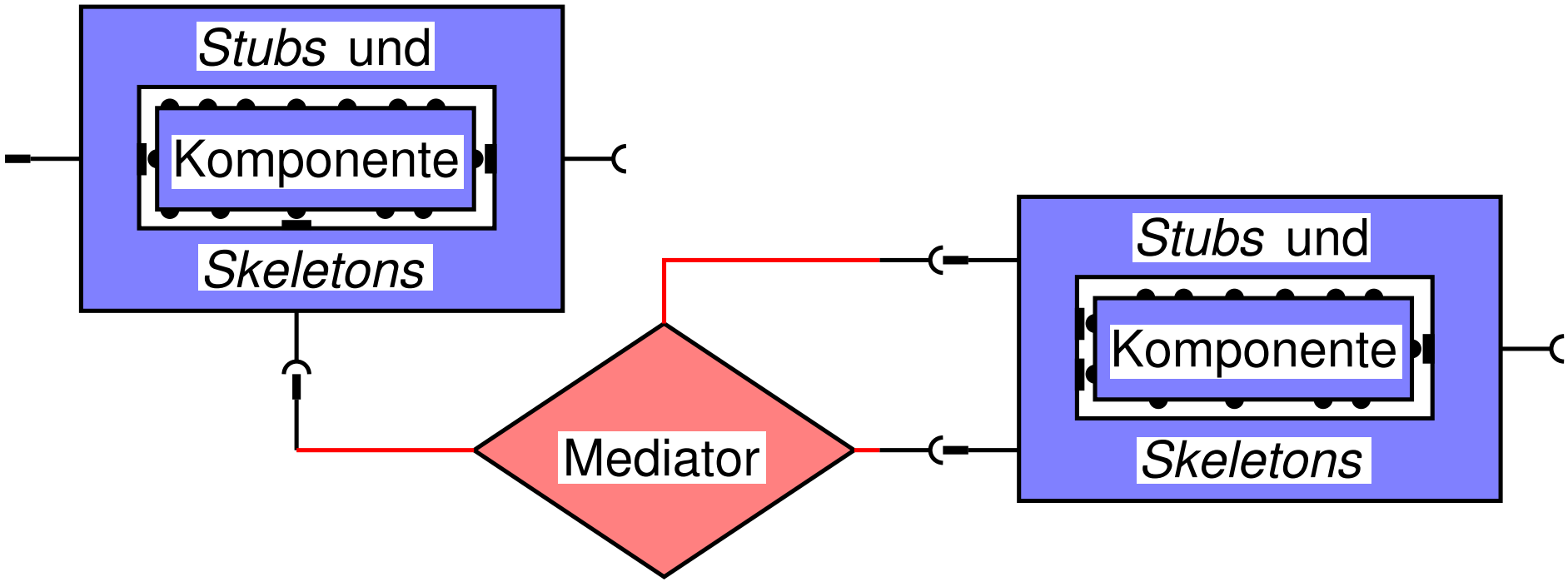


Monitorsystem = dynamische Middleware

- 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

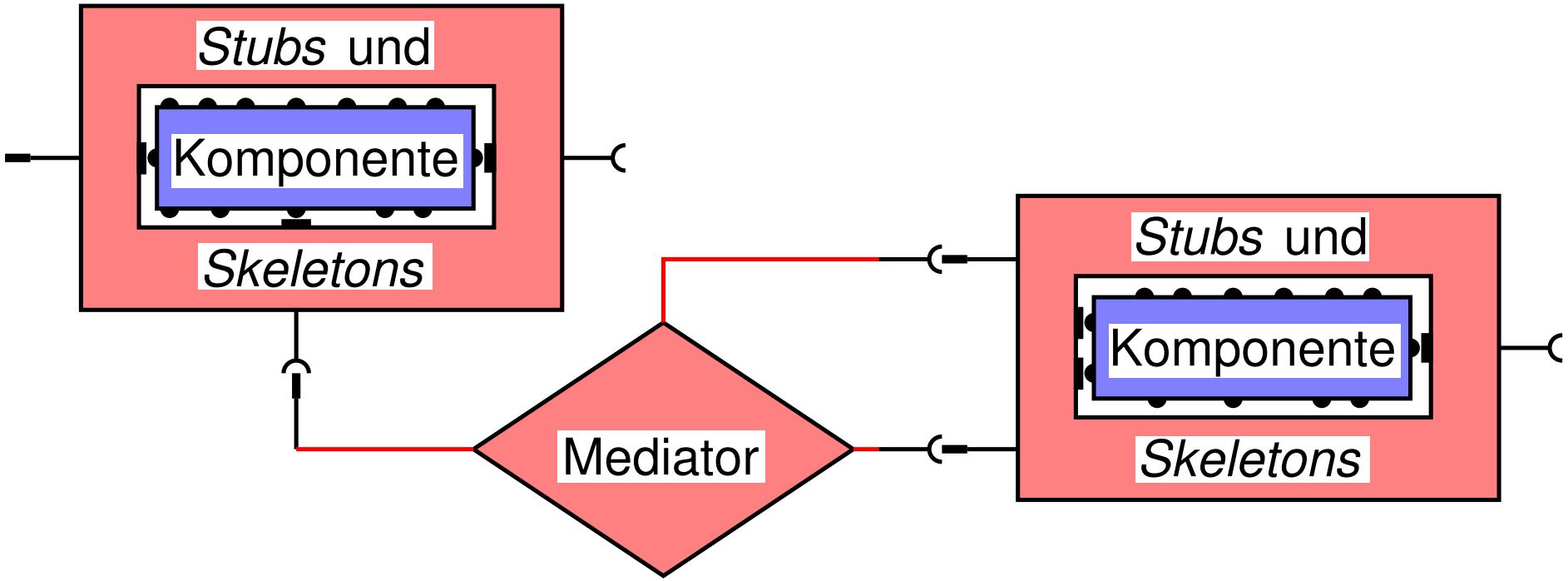
Traditionelles Vorgehen

Komponentenersteller
Integrierer

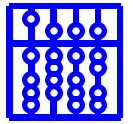


Dynamisches Anbinden von Schnittstellen

Komponentenersteller
Integrierer



4.2. Diskussion



Flexiblere Nutzung von Komponenten:

- Mächtigere Schnittstelle nutzbar
- Anpassung der Schnittstelle möglich

Einfachere 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
- 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

Interoperable Laufzeit-Werkzeuge

- Verbesserter Funktionsumfang durch Synergien
- Einfachere Realisierung durch Modularität

Datenintegration

- Gemeinsames Monitorsystem
- Konflikterkennung und -auflösung

Verhaltensintegration

- Mediator-Ansatz
- Dynamisches Einbinden von Schnittstellen

6. Fazit und Ausblick

Ähnlichkeit zwischen Monitorsystemen und Middleware

- Systematische Trennung von Middleware und Monitordiensten

Prototyp der Werkzeugumgebung

- Ansatz / Realisierungstechniken praxistauglich
  - Aufbau modularer Werkzeugumgebungen
- Verhaltensintegration noch unkomfortabel
  - Integration mit Koordinationssprache