Konzepte zur flexiblen Kooperation von Anwendungsprogrammen

Roland Wismüller

Institut für Softwarewissenschaft, Universität Wien
LRR-TUM, Technische Universität München

Agenda

1. Kooperation von Anwendungsprogrammen

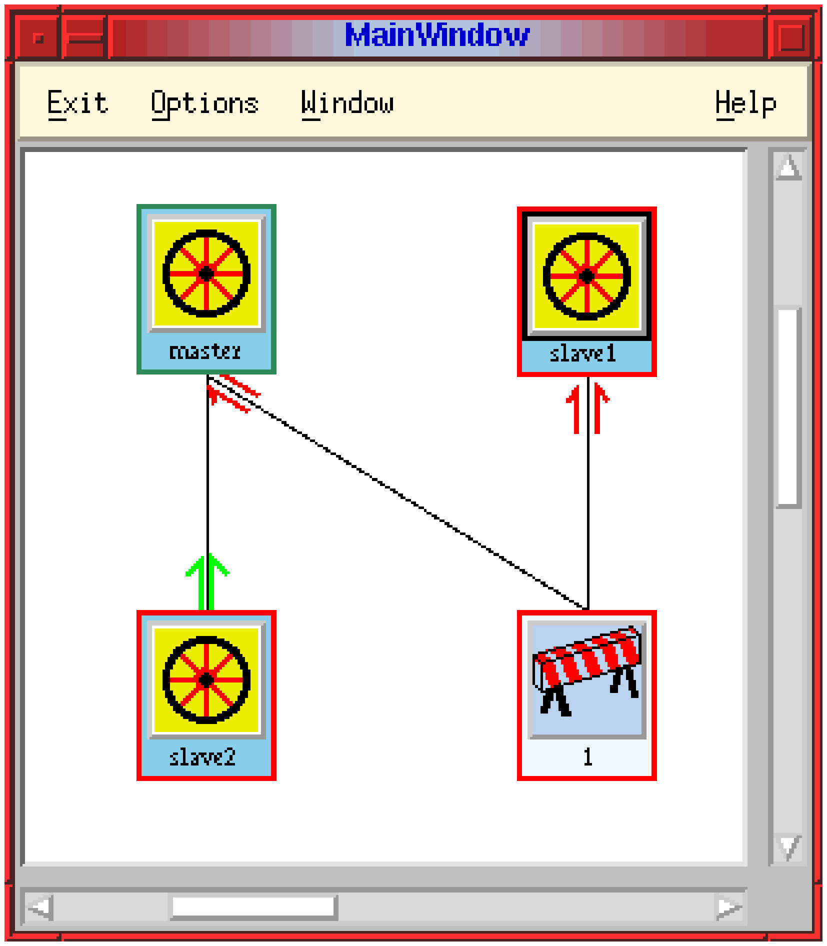
1.1 Motivation:

- Anwendungs„integration“: lockere Zusammenarbeit von Anwendungsprogrammen
  - z.B. `emacs`, `latex`, `ghostview`
  - z.B. Vorlesungsdatenbank, Institutsraumplaner
- Anwendungsprogramme als spezielle Komponenten
  - i.a. mit Benutzerinteraktion
  - aktiv, mit eigenständigem Verhalten (im Ggs. zu Servern)
  - unabhängig voneinander entwickelt und nutzbar
  - primär für isolierte Nutzung vorgesehen
  - Nebenbedingung: keine Modifikation der Komponenten

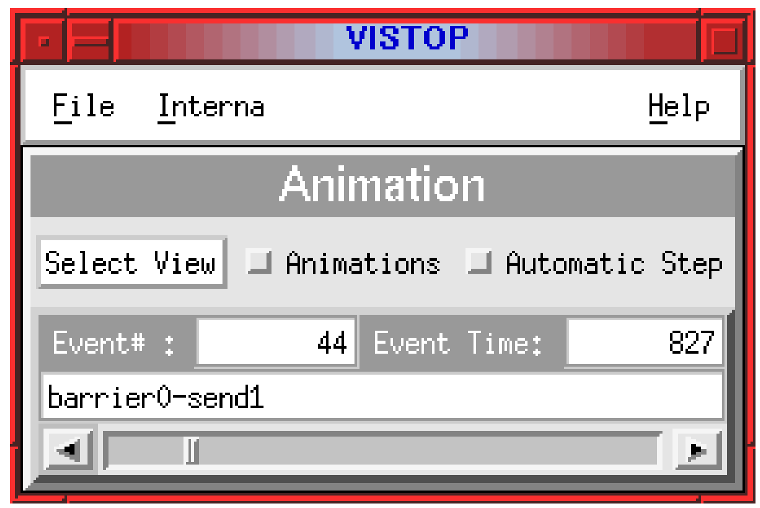
1.1 Motivation ...

Konkreter Hintergrund am LRR-TUM:

- Entwicklung von Laufzeit-Werkzeugen
  - Quellsprach-Debugger DETOP
  - Ablaufvisualisierer VISTOP
  - Leistungsanalyse, deterministische Ausführung, Sicherungspunkterstellung, ...
- Wunsch: kombinierte Verwendung der Werkzeuge
  - z.B. Visualisierer + Debugger
    - Überblick über Ablaufverhalten
    - detaillierte Untersuchung

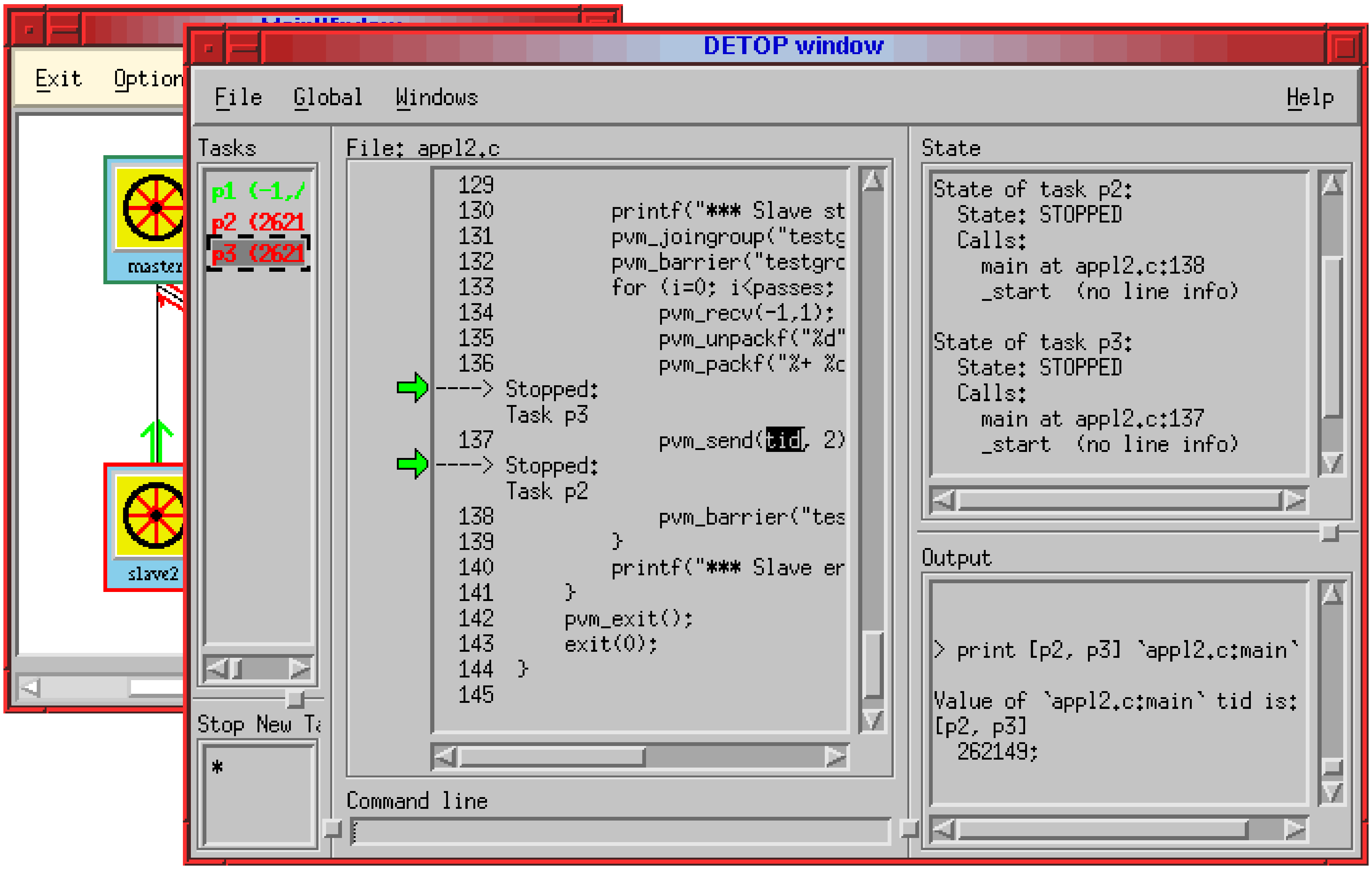


MainWindow
Exit Options Window Help
master
slave1
slave2
1

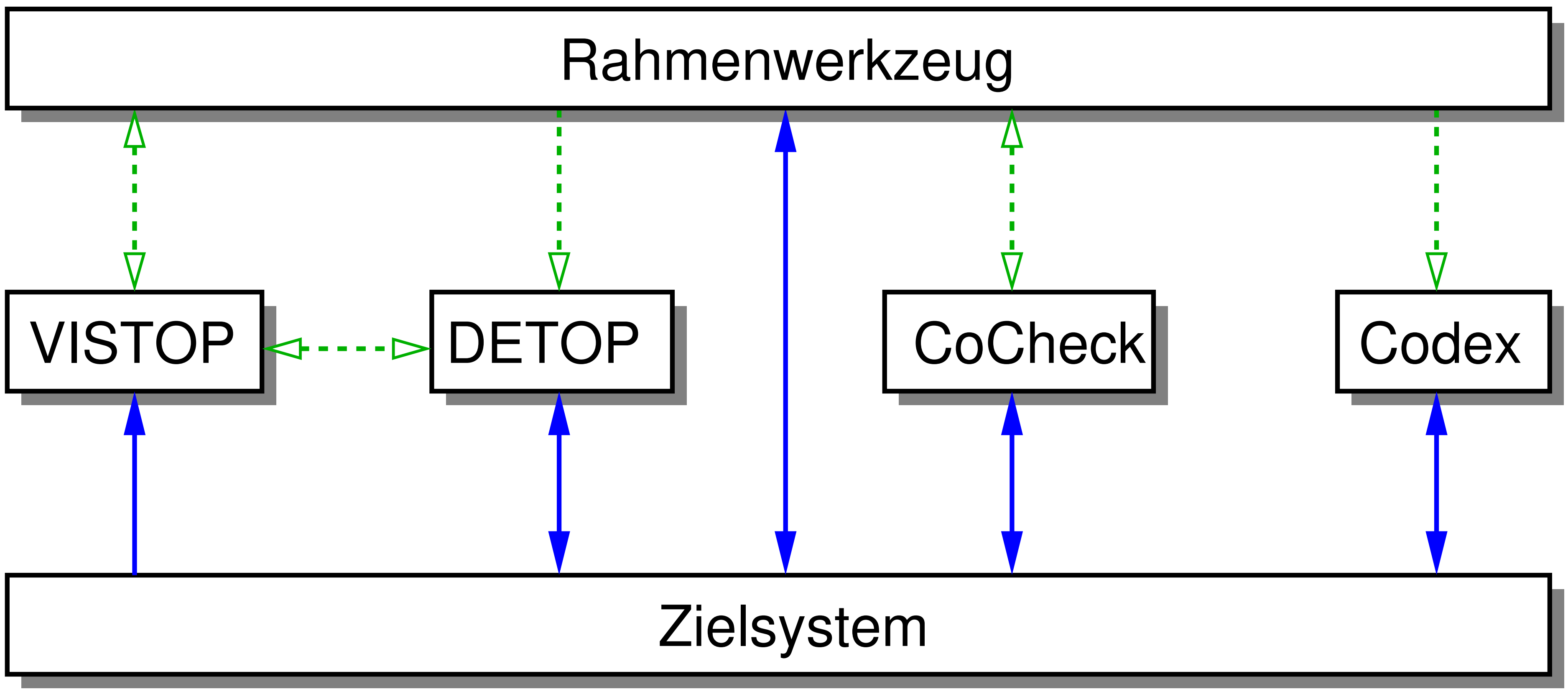


VISTOP
File Interna Help
Animation
Select View
Animations
Automatic Step
Event# :
44
Event Time:
827
barrier0-send1

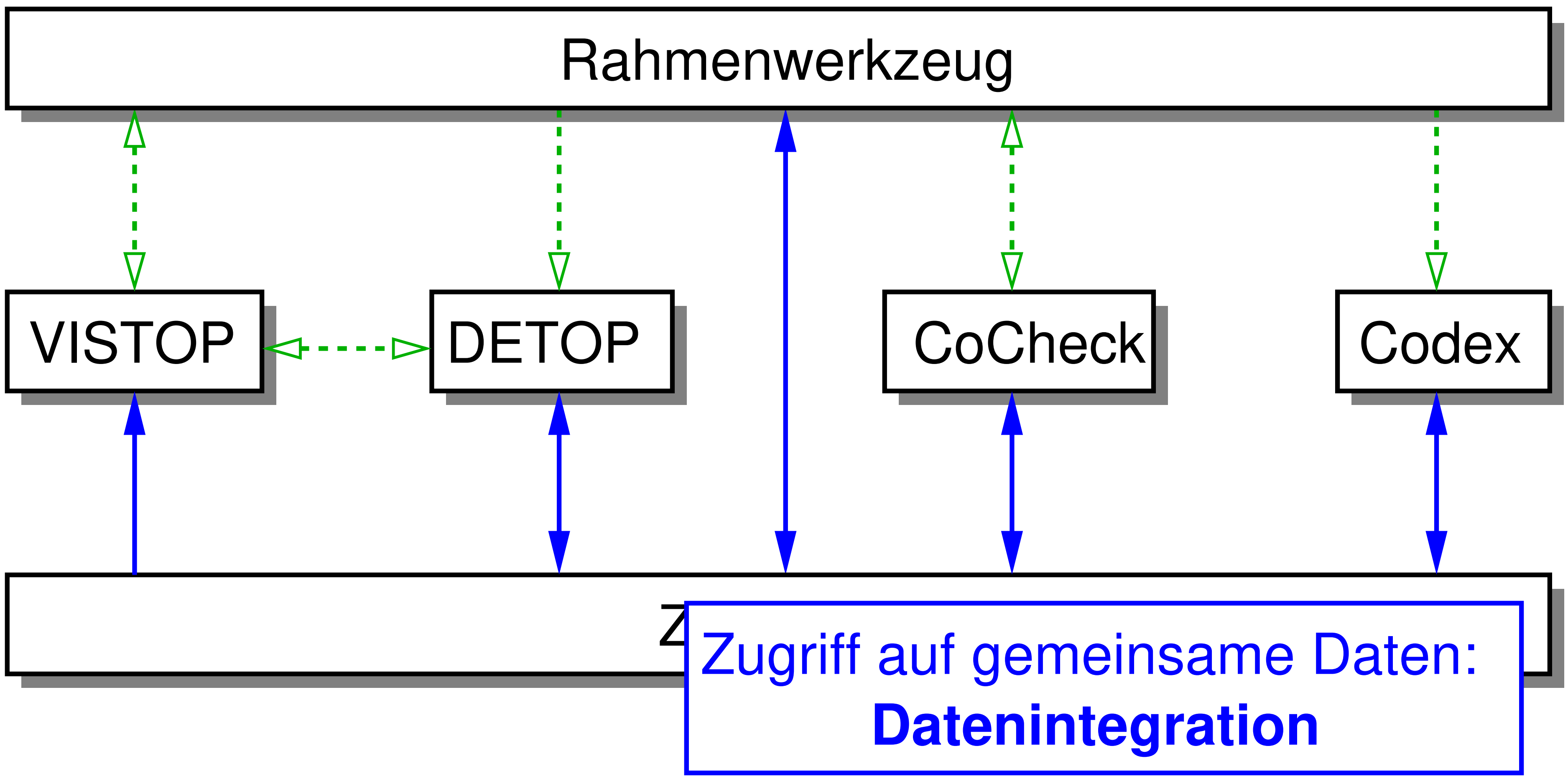
1.1 Motivation ...



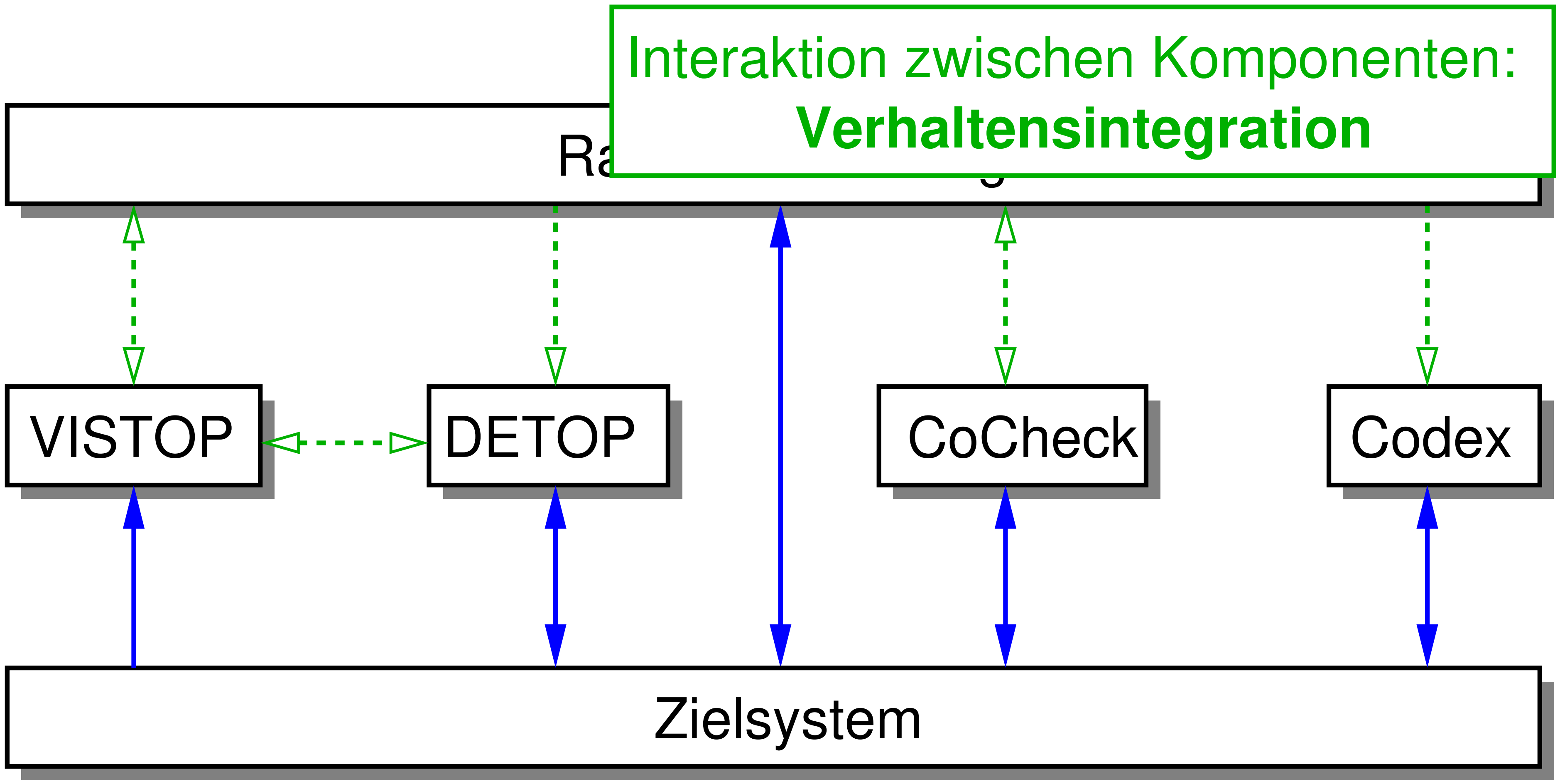
Aufbau der Werkzeugumgebung



Aufbau der Werkzeugumgebung



Aufbau der Werkzeugumgebung



1.2 Datenintegration

Problem: Zugriff auf gemeinsame Daten

- Evtl. „exklusive“ Schnittstelle
- Konflikte durch Modifikationen (Konsistenz, Transparenz)

Im Fall der Werkzeugumgebung:

- Gemeinsame Daten = Zielsystem
- Prozeßkontroll-Schnittstelle ist exklusiv
- Lösung: Online-Monitorschnittstelle zur Beobachtung und Steuerung paralleler und verteilter Systeme
  - Projekt OMIS (*On-line Monitoring Interface Specification*)

1.2 Datenintegration

Ziele von OMIS

- Werkzeug- und Plattformunabhängigkeit
- Gleichzeitige Nutzung durch mehrere Werkzeuge

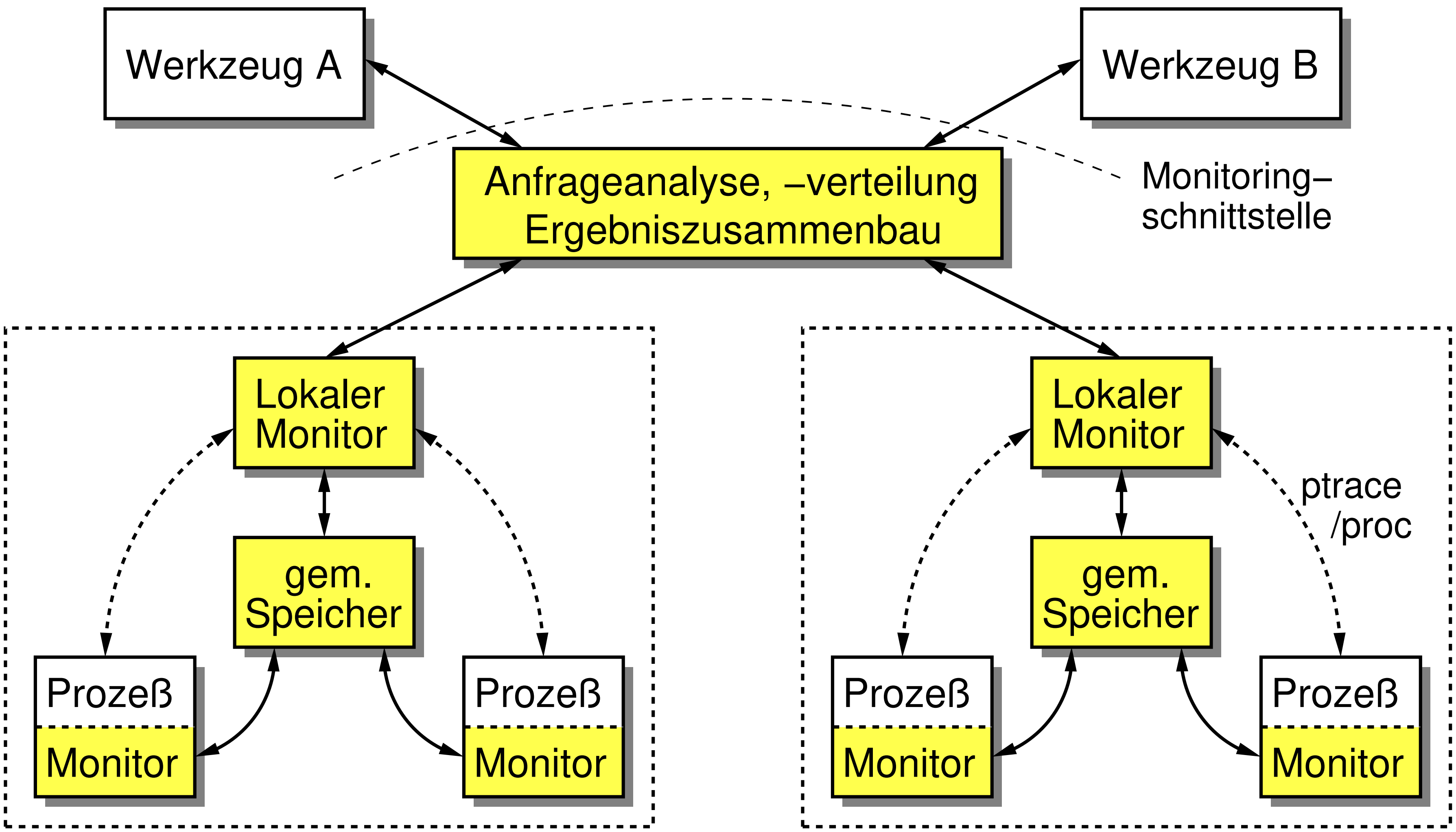
Konzepte

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

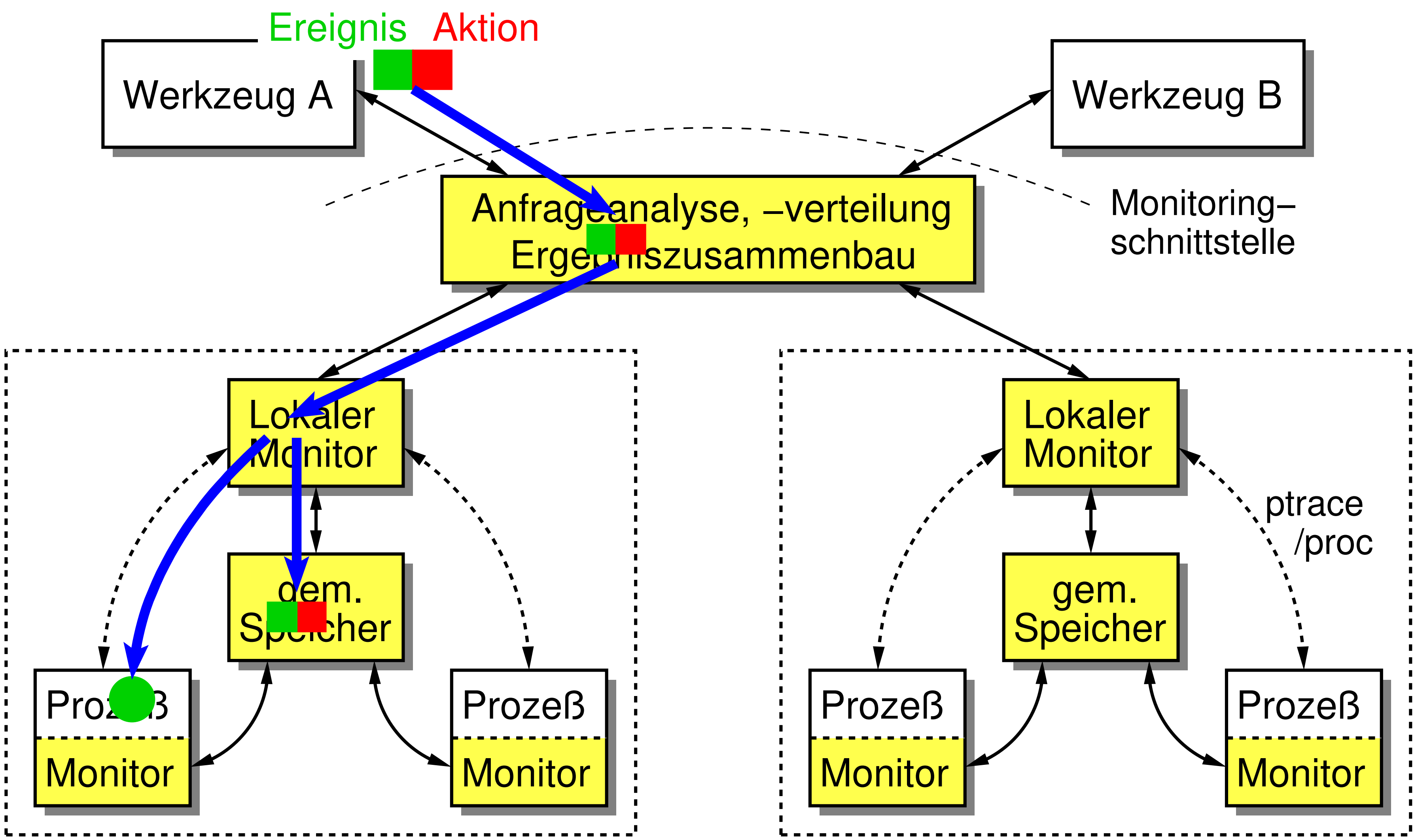
```
thread_started_lib_call([p_1],"pvm_send") :
    thread_stop([$node])
    thread_get_backtrace([$thread],0)
```

- Objektmengen, Ortstransparenz
- Erweiterbarkeit

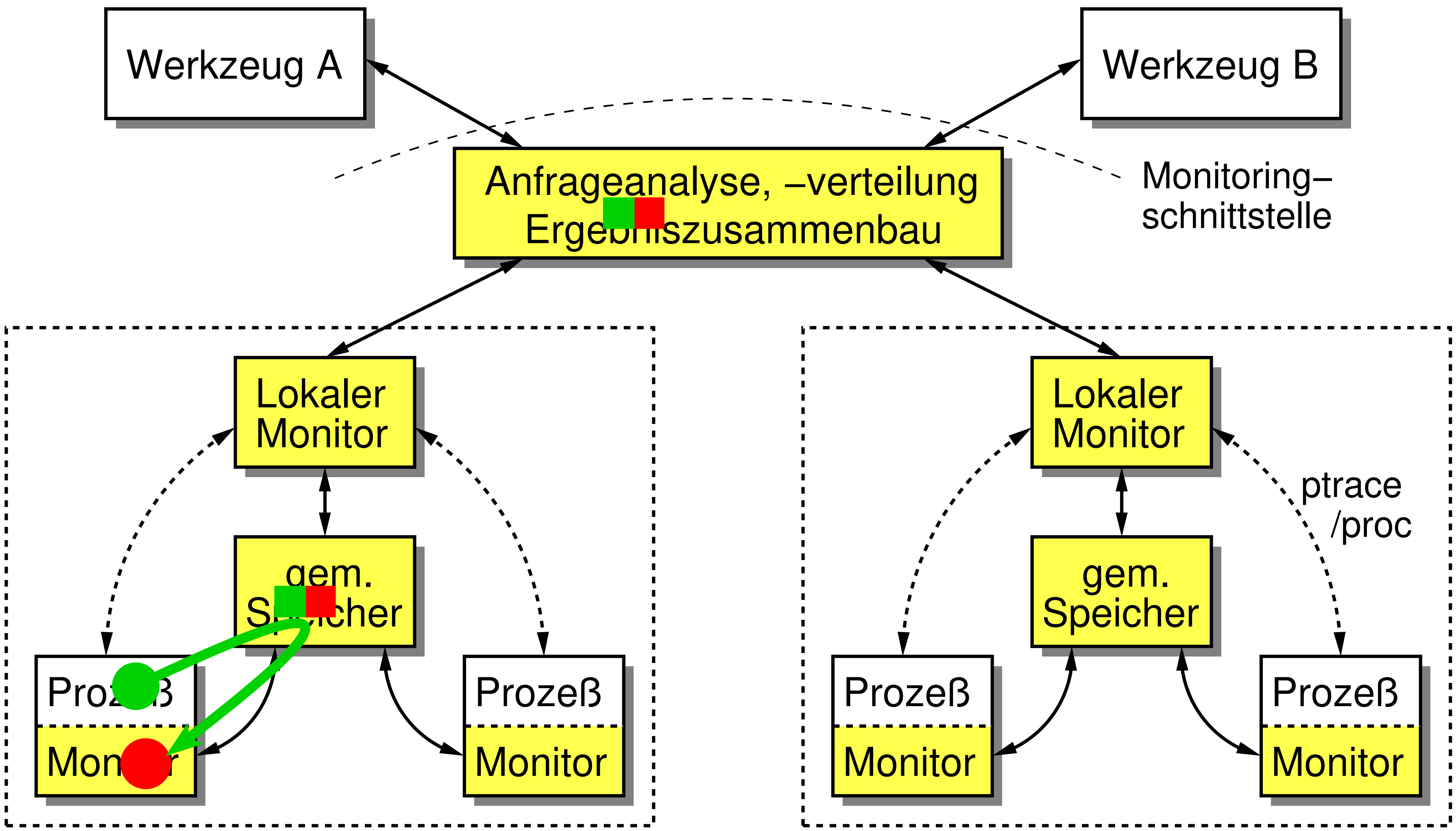
Aufbau des OCM (*OMIS Compliant Monitoring system*)



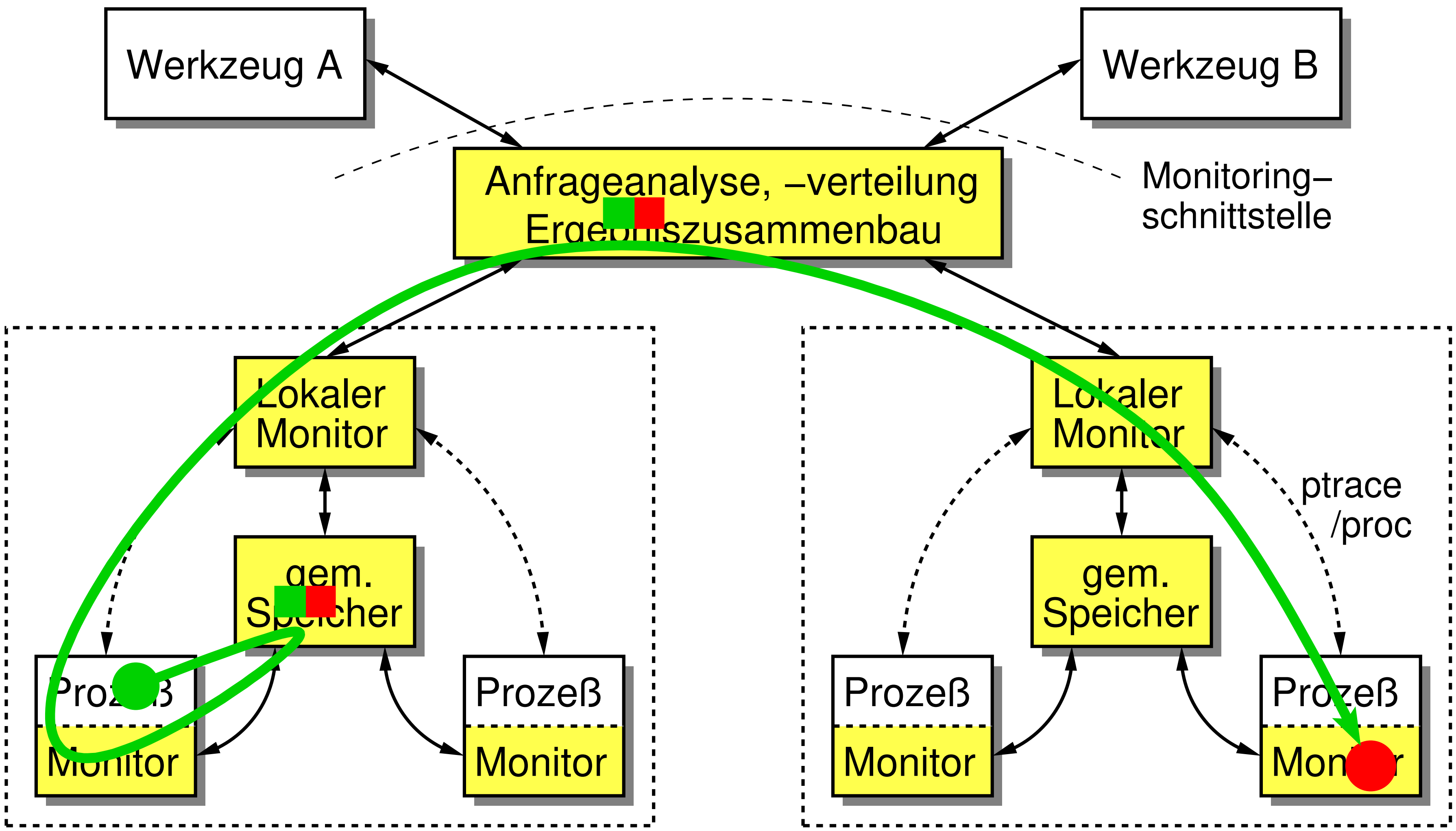
Anfrageabarbeitung im OCM



Anfrageabarbeitung im OCM



Anfrageabarbeitung im OCM



Verbleibende Konflikte: explizite Zustandsmodifikationen

- 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?
- Ansatz:
  - genaue Modellierung der Zielsystemobjekte
  - Konflikterkennung und -auflösung anhand von Attributen der Zugriffe

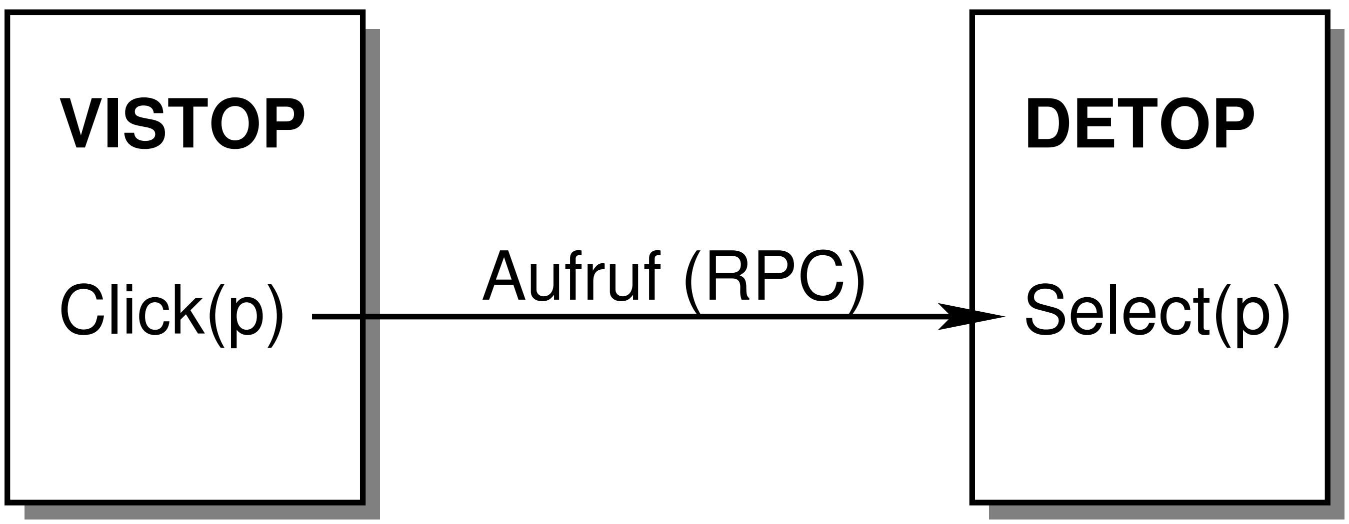
1.3 Verhaltensintegration

Ziel: Interaktion zwischen den Komponenten

- Ereignis in einer Komponente soll zu Aktion in anderer Komponente führen
- Im Beispiel der Werkzeugumgebung etwa:
  - synchronisierte Auswahl: Auswahl von Prozessen in VISTOP $\Rightarrow$ Auswahl derselben Prozesse in DETOP
  - konsistente Darstellung: Rückkehr zu vergangenem Zustand in VISTOP $\Rightarrow$ Anzeige von DETOP als ungültig kennzeichnen
  - Spezialfall: Aufruf von Aktionen durch das Rahmenwerkzeug

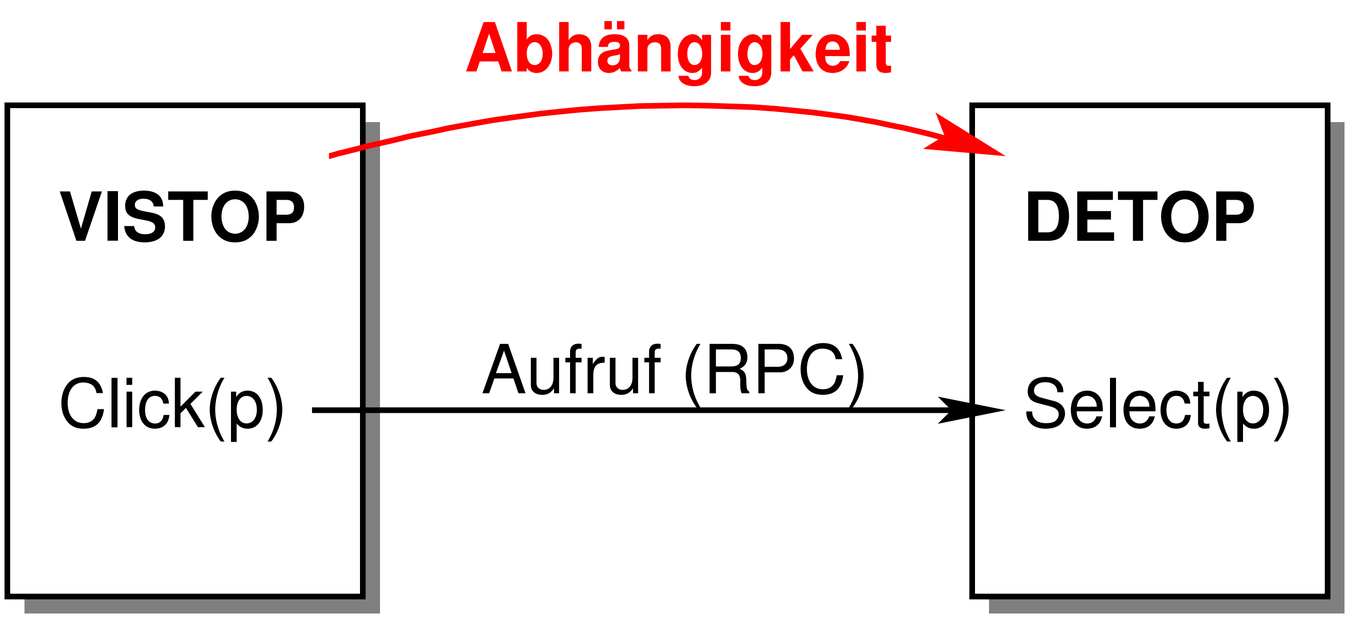
Einfachster Ansatz: Feste Verdrahtung

- Direkter Aufruf notwendiger Aktionen in anderer Komponente
- Z.B. mittels RPC, RMI, CORBA, PCTE



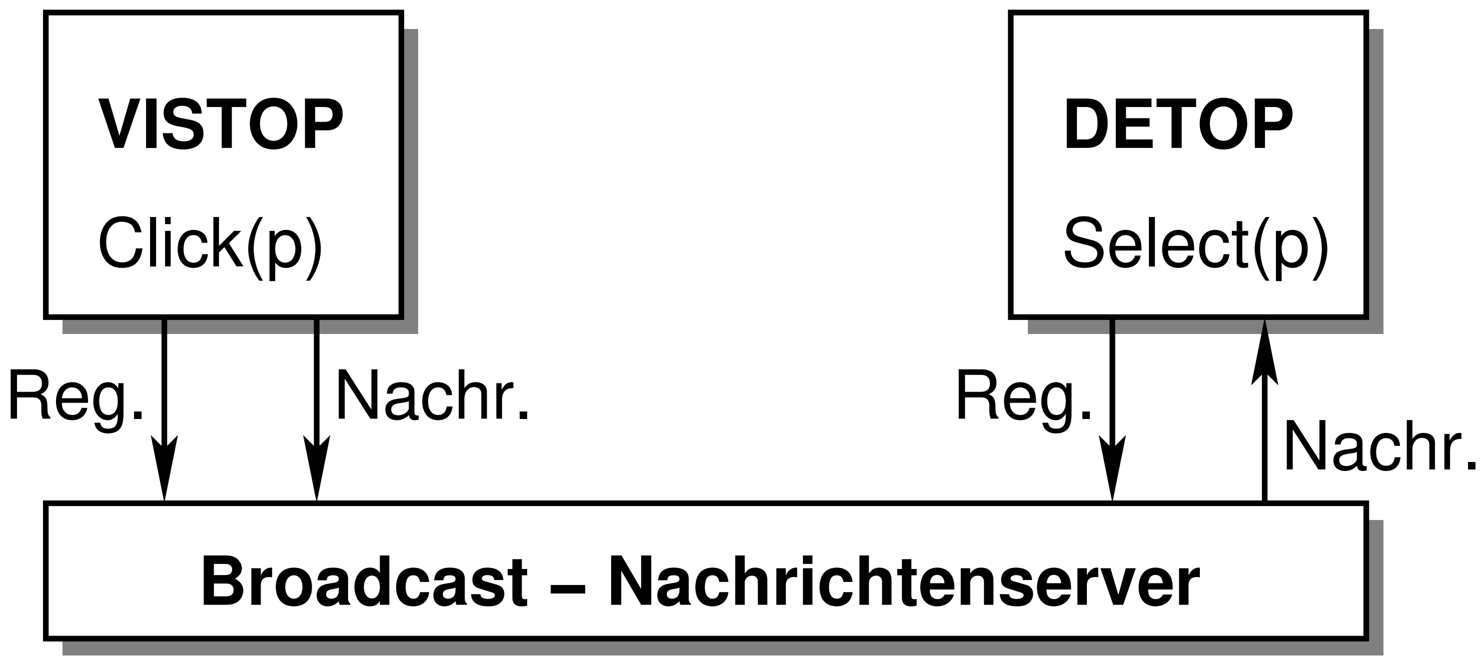
Einfachster Ansatz: Feste Verdrahtung

- Direkter Aufruf notwendiger Aktionen in anderer Komponente
- Z.B. mittels RPC, RMI, CORBA, PCTE



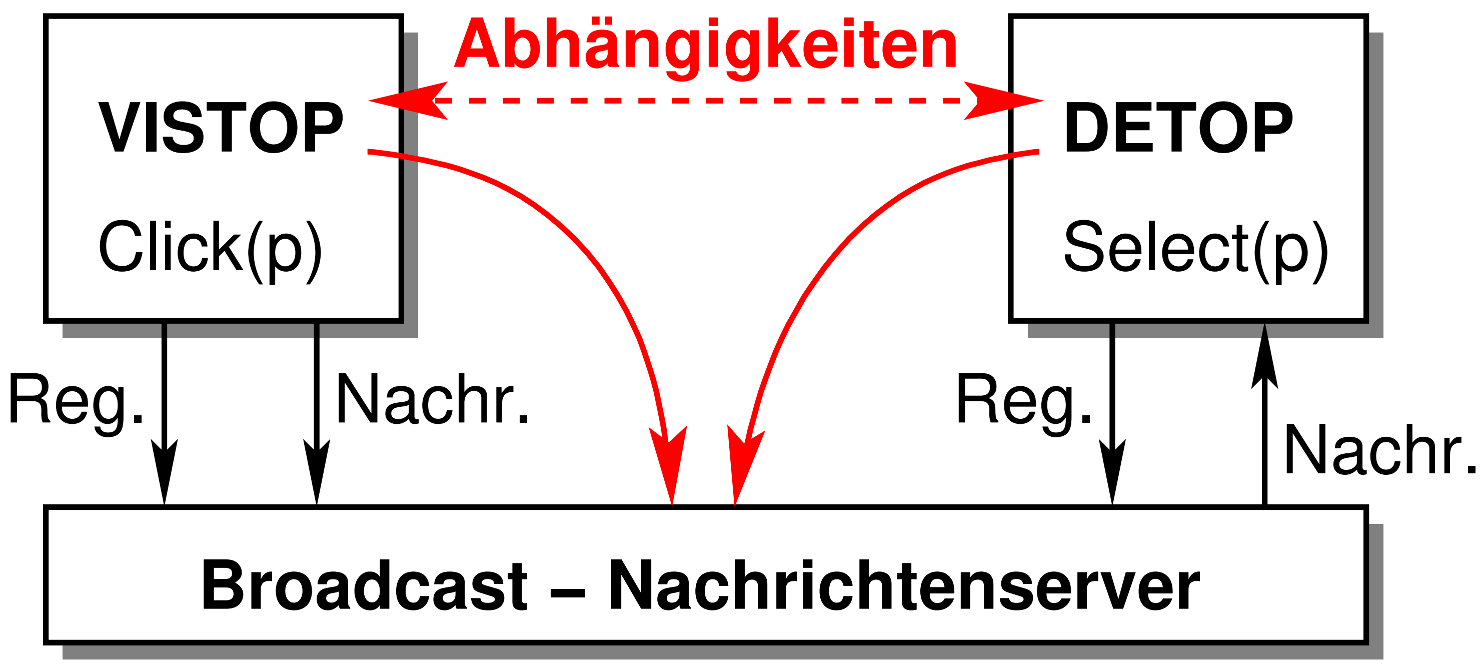
Broadcast-Nachrichtenserver

- Nachricht wird an alle registrierten Komponenten gesendet
- Z.B. ToolTalk, SoftBench



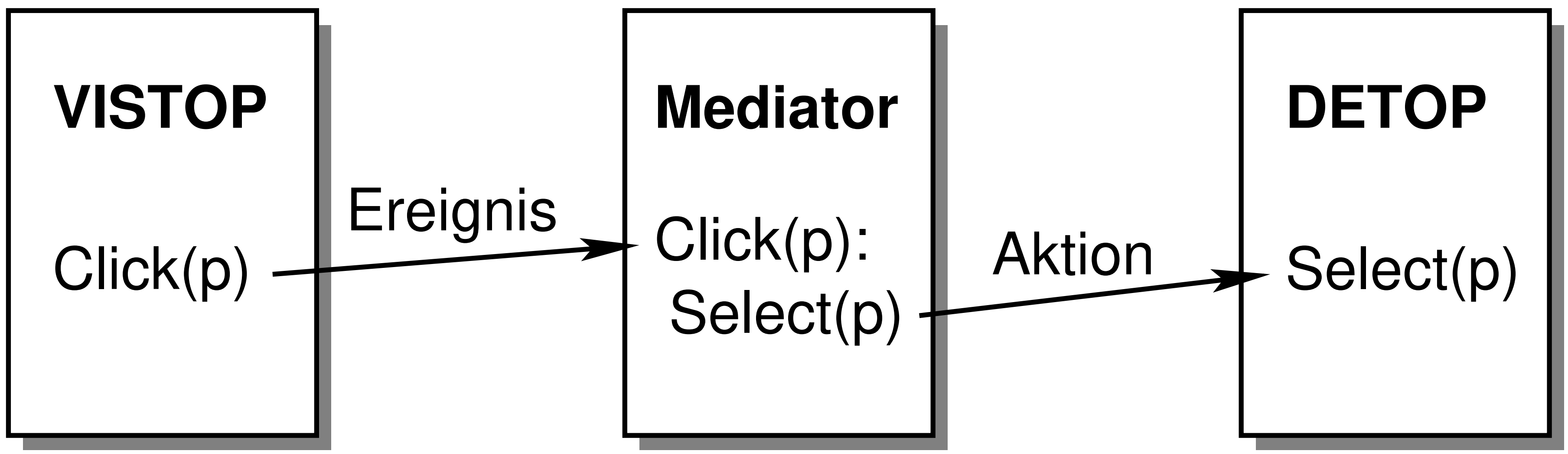
Broadcast-Nachrichtenserver

- Nachricht wird an alle registrierten Komponenten gesendet
- Z.B. ToolTalk, SoftBench



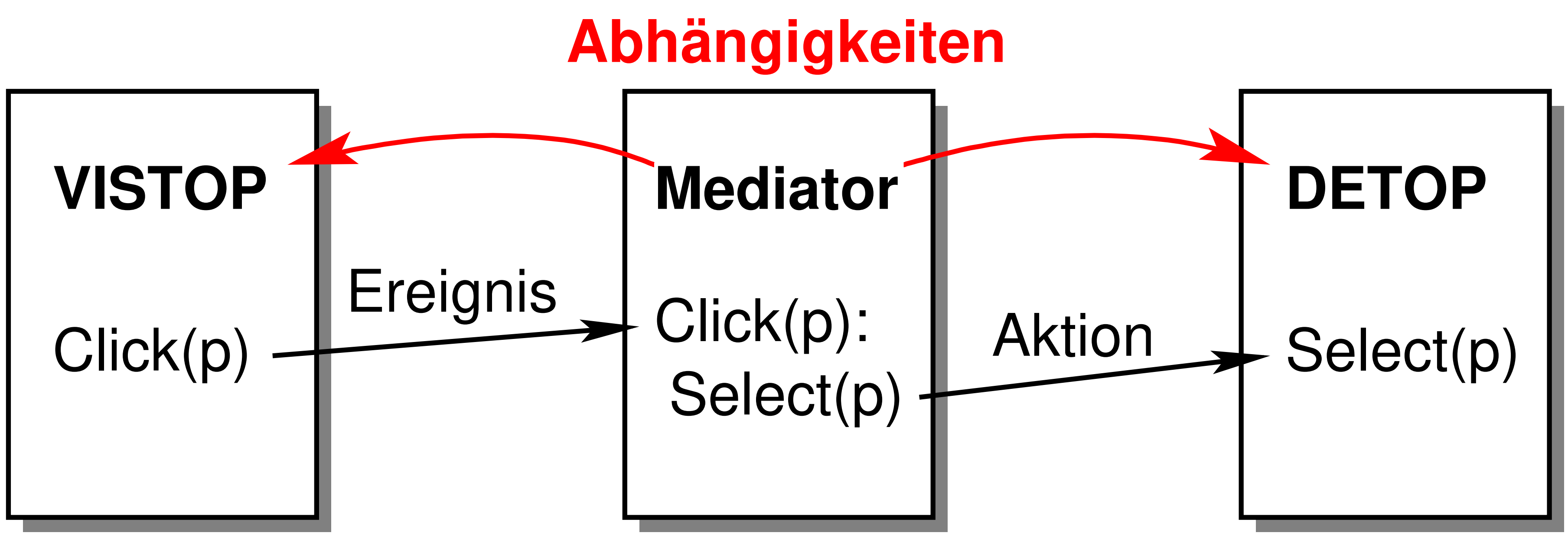
Mediatoren / Interaktoren

- Mediator überwacht Ereignis, ruft zugeordnete Aktion auf
- Z.B. ToolBus, Manifold, CHAIMS, TIL



Mediatoren / Interaktoren

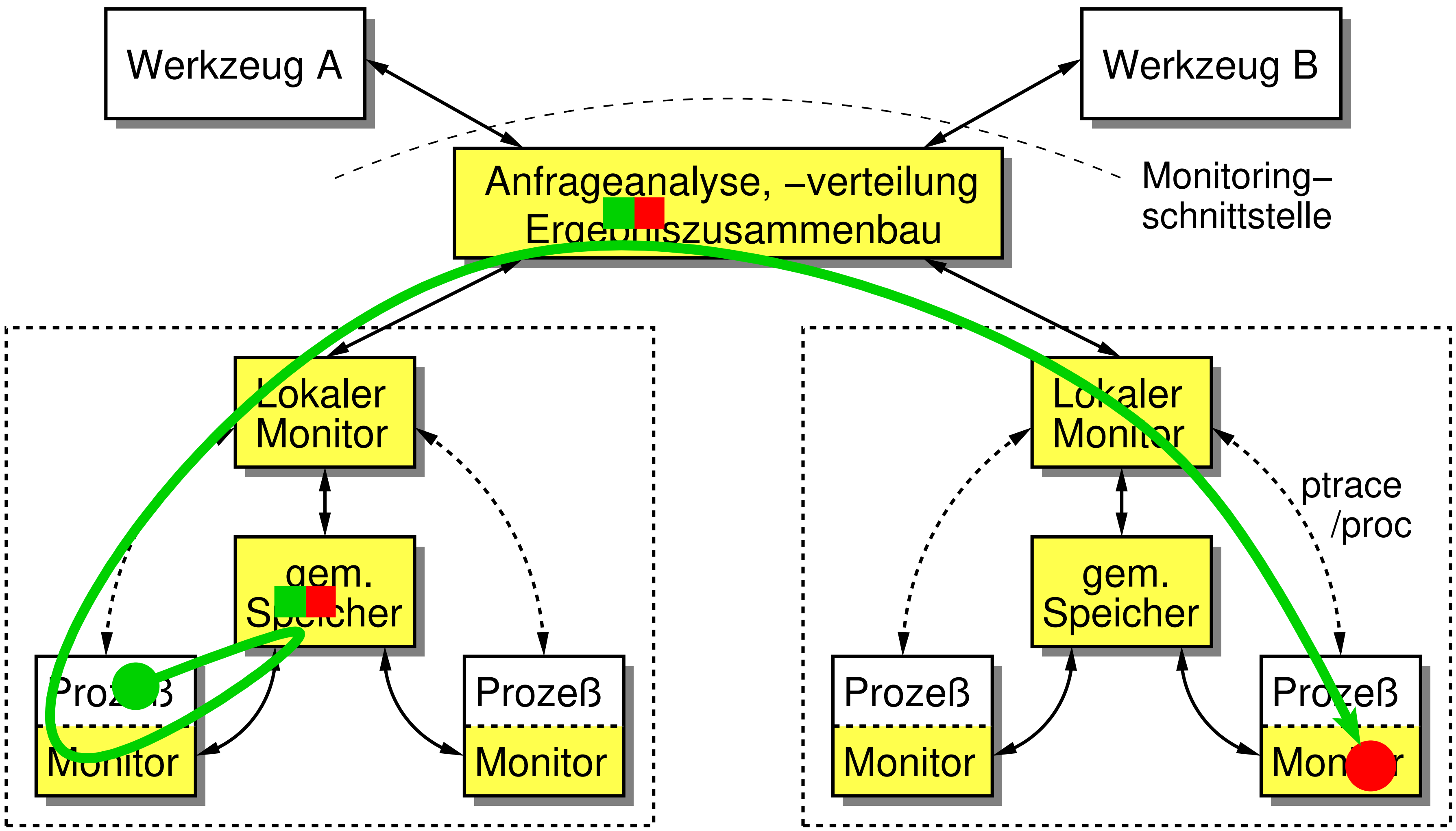
- Mediator überwacht Ereignis, ruft zugeordnete Aktion auf
- Z.B. ToolBus, Manifold, CHAIMS, TIL



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
  - Export der Schnittstellen für feste Middleware
- Komponenten meist nur für isolierte Nutzung entwickelt
  - Einsatz in Umgebung erfordert Modifikation

Anfrageabarbeitung im OCM



Spezielle Eigenschaften des OCM:

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

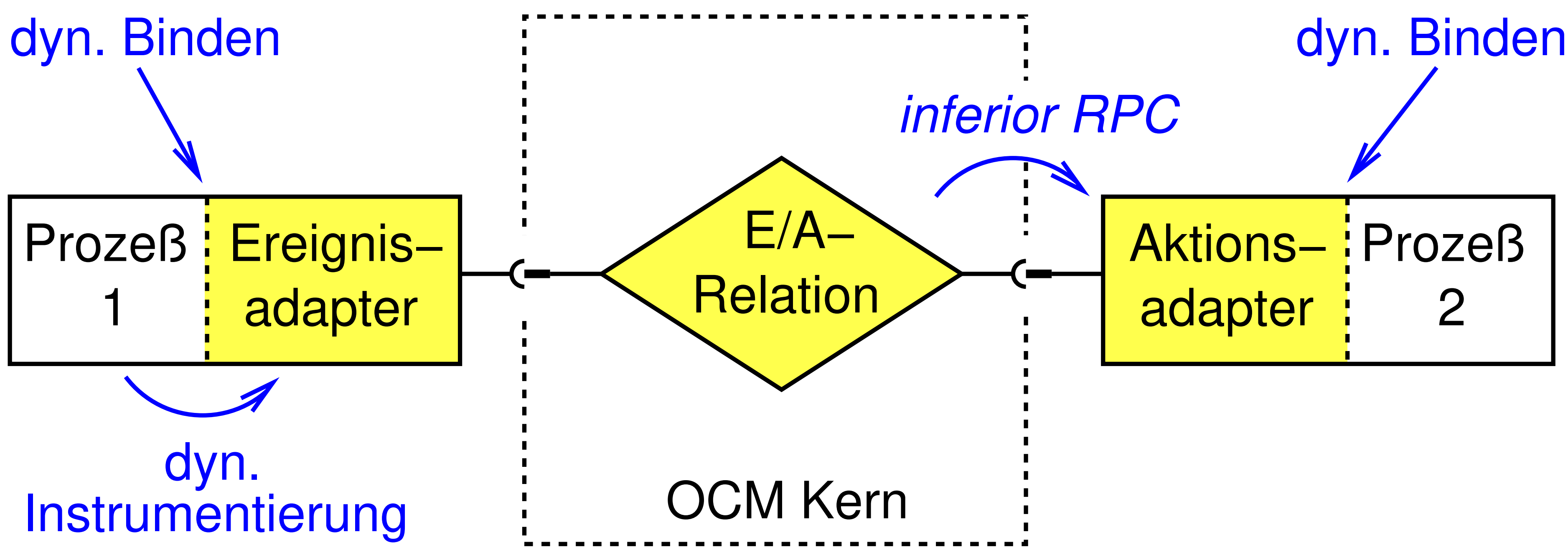
$\Big\}$ ohne spezielle Vorbereitung dieser Prozesse

- Menge der Ereignisse und Aktionen zur Laufzeit erweiterbar

$\Longrightarrow$ Mediatorkonzept auch ohne explizitem Export von Schnittstellen realisierbar

Zur Realisierung:

- Dynamische Instrumentierung
- *Inferior RPC* (synchronisiert durch lokales Ereignis)
- Dynamisches Binden



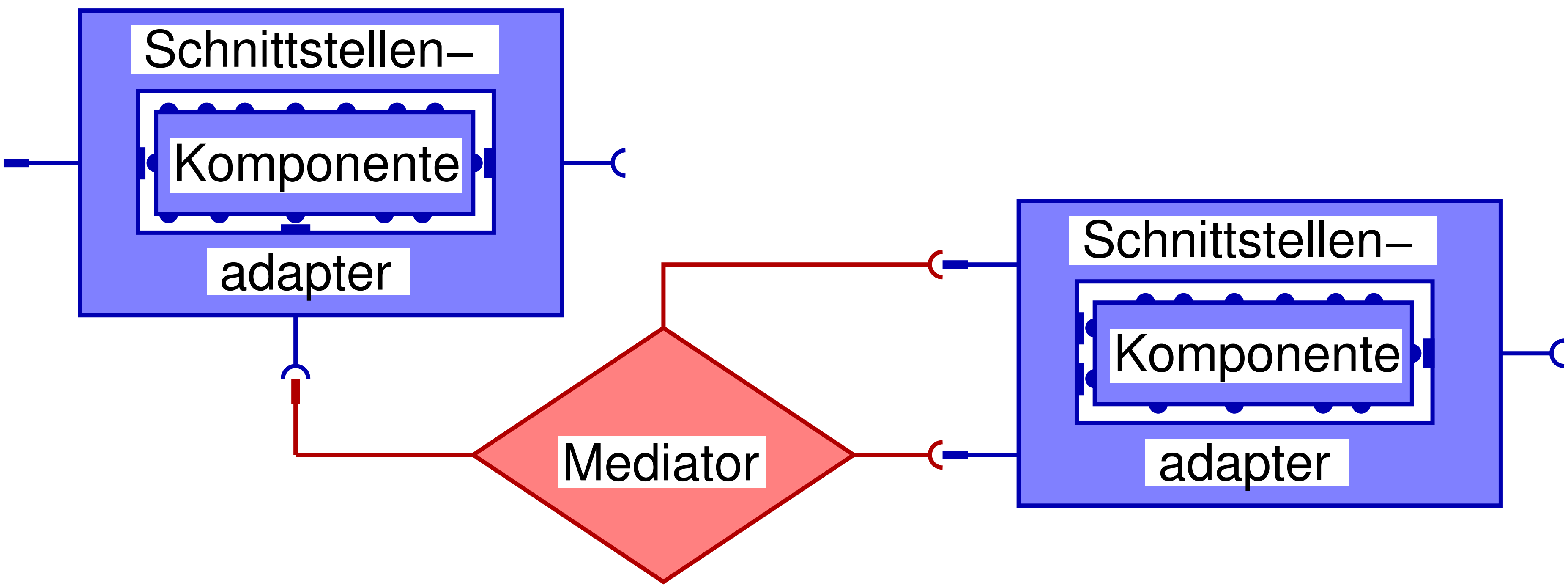
1.5 Resultat

Monitorsystem = dynamische Middleware

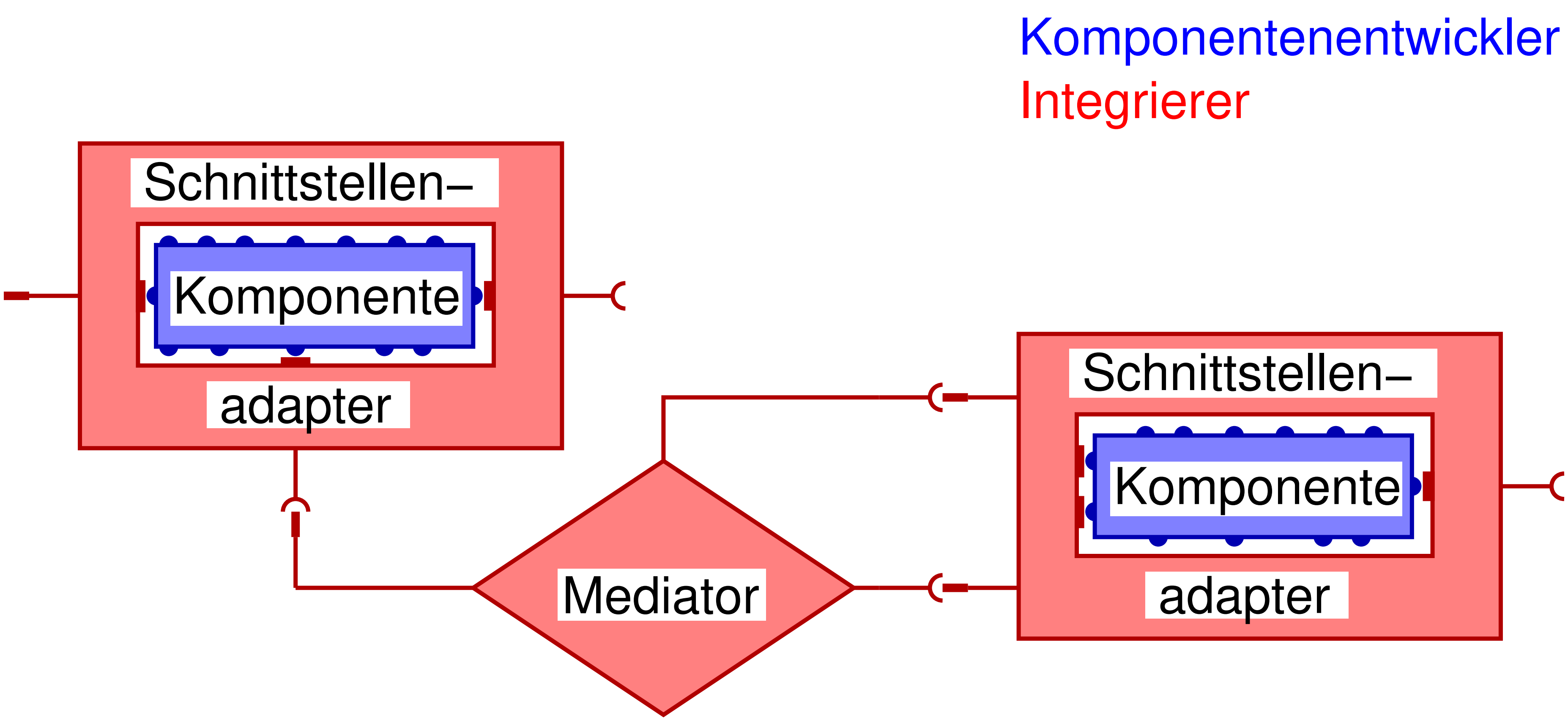
- Mediatorkonzept
  - Verhaltensregeln dynamisch programmierbar
- Dynamisches Anbinden von Schnittstellen(adaptern)
  - alle für Binder sichtbaren Symbole nutzbar
    - globale Variable / Objekte
    - globale Funktionen
    - öffentliche und geschützte Methoden
  - Ereignisse: Beginn / Ende einer Funktion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

Traditionelles Vorgehen

Komponentenentwickler
Integrierer



Dynamisches Anbinden von Schnittstellen



1.5 Resultat ...

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

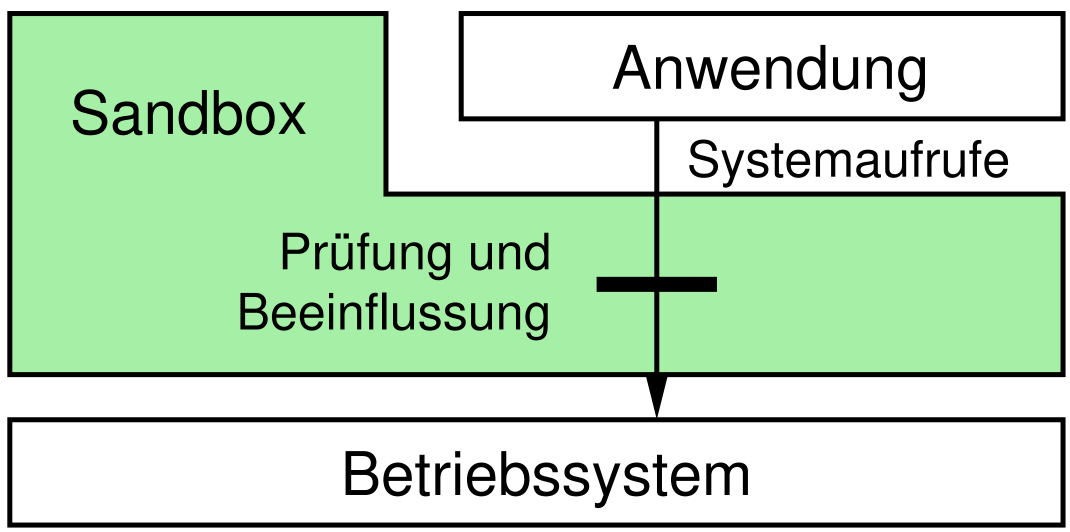
☞ Bei Komponenten mit gemeinsamen Daten:
Möglichkeit einer gleichzeitigen Nuzung muß beim Entwurf berücksichtigt werden, konkrete Kooperation noch nicht

Motivation

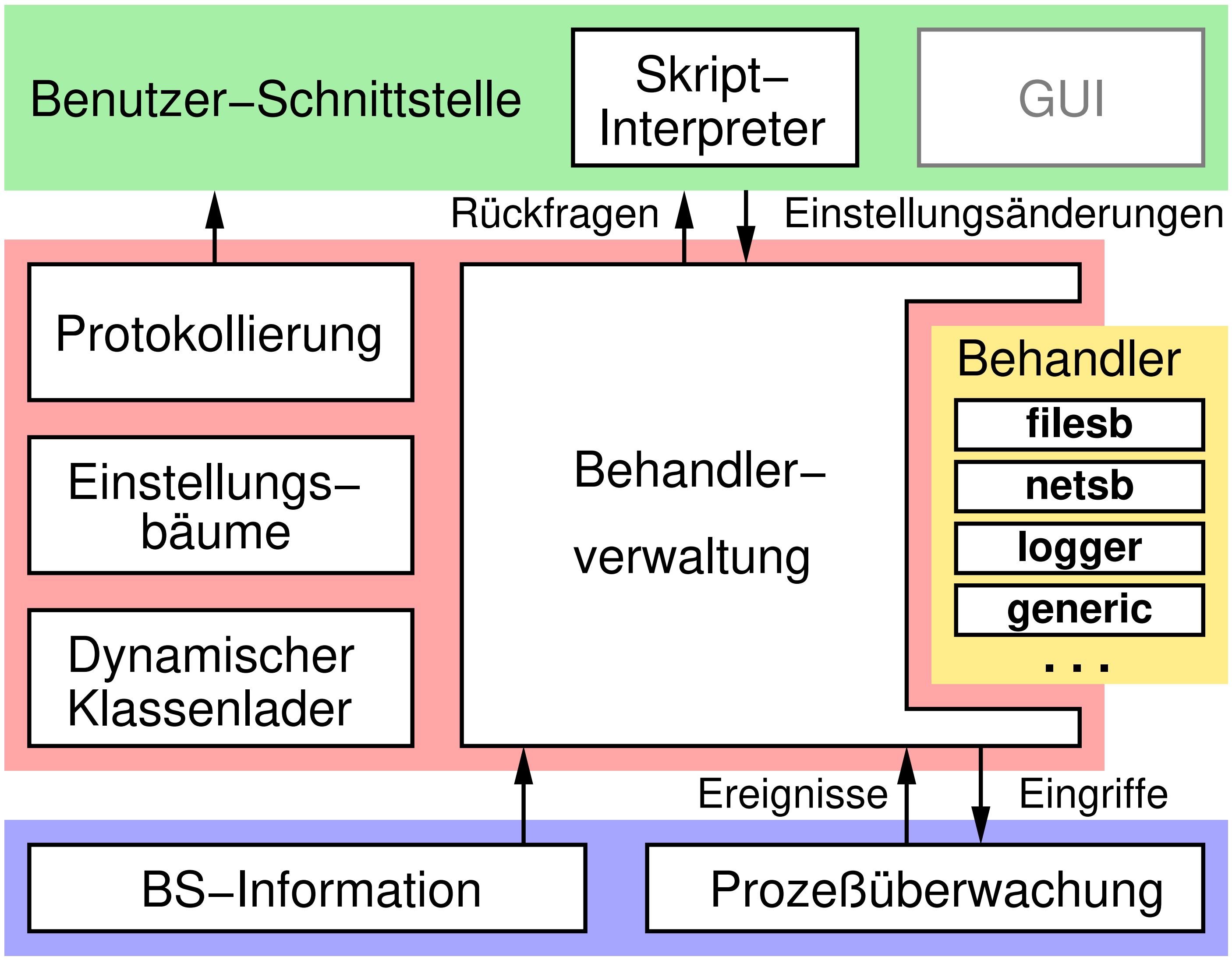
- Sicherheit bei der Ausführung unbekannter Software, z.B. mobilem Code
- *Community Access Services* in Grid-Umgebung: Sicherheit bei gemeinsamer Nutzung einer Kennung
- Problem: Betriebssystem schützt nur Benutzer gegeneinander
  - kein Schutz der Resourcen eines Benutzers vor *eigenen* Prozessen
- Lösung:
  - Ausführung von Prozessen in gesicherter Umgebung („*Sandbox*“), die Zugriffsrechte einschränkt

Zur Realisierung

- Basis: Zugriff auf Resourcen nur über Systemaufrufe
- Einschränkung der Zugriffsrechte durch Überwachung aller Systemaufrufe (Prozeßkontrolle)
  - Bei Rechteverletzung: Abbruch des Aufrufs / Programms



Architektur der Sandbox



Stand des Projekts

- Sandbox für Linux einsatzbereit
  - normaler Benutzerprozeß (keine `root`-Rechte)
  - Überwachung beliebiger Prozesse und deren Kinder
- Überwachung von Datei- und Netzwerkzugriffen
  - Rechte für reguläre Mengen von Dateien definierbar
  - Netzzugriffe über Protokollfamilie, Host, Port gefiltert
  - Laufzeit-Overhead: 10-100%
- Weitere Arbeiten:
  - graphische Bedienoberfläche
  - Konverter für *Capability*-Sprachen
  - Portierung auf andere Unix-Systeme

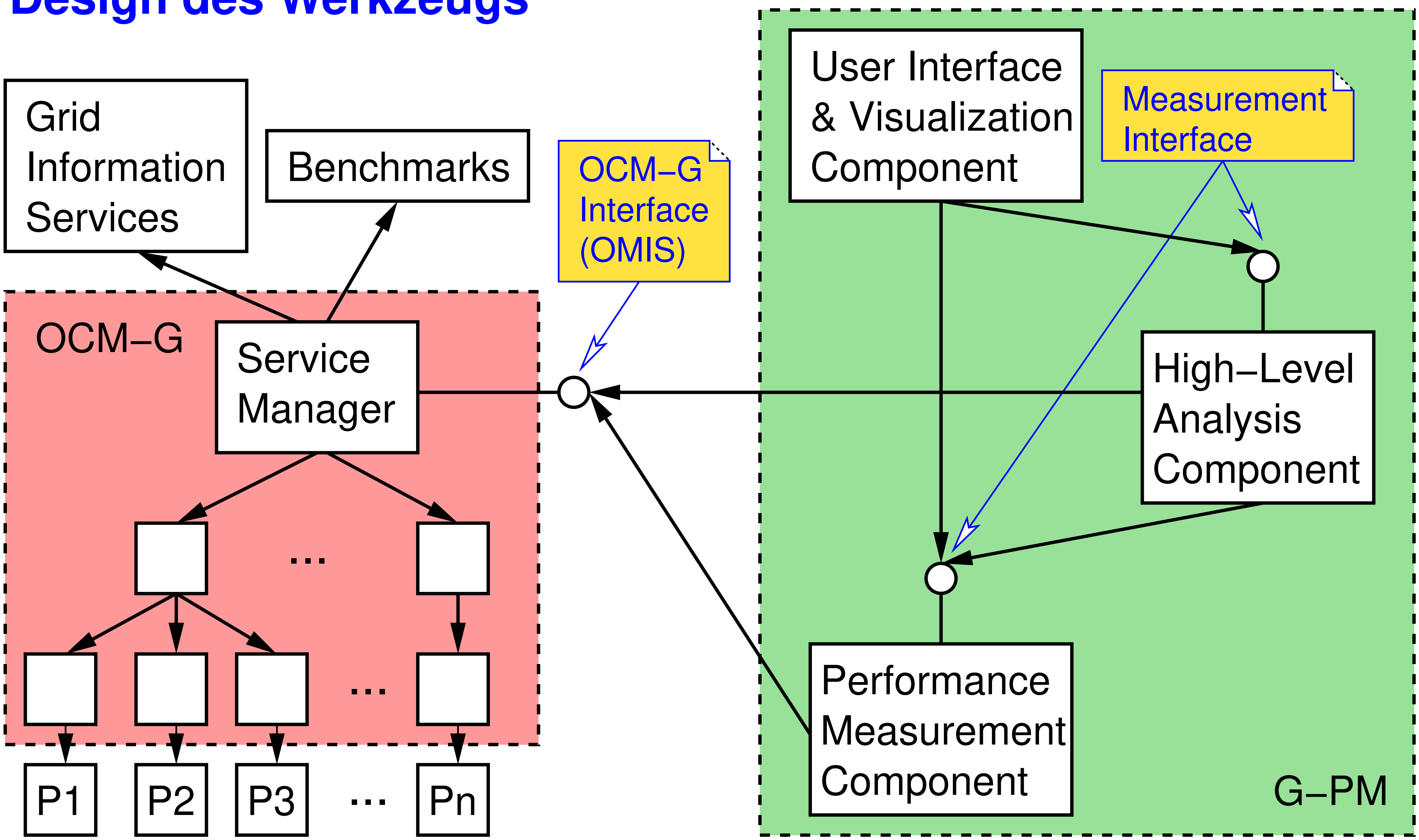
Motivation

- EU IST-Projekt CrossGrid:
  - interaktive Anwendungen im Grid
    - Simulation des Blutflusses in Beinarterien
    - Flut-Vorhersage
    - Hochenergiephysik
    - Wetter- und Klimasimulation
- Anforderungen an Leistungsanalyse u.a.:
  - Messung der Antwortzeit und ihrer Zusammensetzung
  - Leistungsdaten für spezifische Interaktionen
  - Messung des Konvergenzverhaltens
  - Leistungsdaten der Rechner-Infrastruktur

Ansatz

- Online-Leistungsanalyse
- Messung von Standardmetriken
  - z.B. CPU-Zeit, Kommunikationsvolumen, E/A; Knotenlast
  - automatische Instrumentierung
- Minimale manuelle Instrumentierung der Anwendung (Proben)
  - relevante Ereignisse innerhalb der Anwendung
  - Zusammengehörigkeit von Ereignissen
  - anwendungsspezifische Daten
- Benutzerdefinierbare Metriken
  - auf Basis existierender Metriken und Proben

Design des Werkzeugs



Stand des Projekts

- CrossGrid: März 2002 - Februar 2005
- Erste Entwurfsphase abgeschlossen
- Erster Prototyp bis Januar 2003
- Weitere Arbeiten:
  - Entwurf einer Metrik-Definitionssprache
  - Realisierung eines Compilers / Interpreters
  - Optimierung der Messungen (verteilte Auswertung)

4. Fazit und Ausblick

Anwendungsintegration

- Mediatoren + dynamisches Einbinden von Schnittstellen(adaptern)
  - vereinfachte (verteilte) Kooperation von Anwendungen
- Integration mit Koordinationssprache
- Unterstützung durch Compiler und Betriebssystem

Grid-Computing

- Sicherheits-Werkzeug
  - Schutz d. Ressourcen / d. Grid-Benutzer gegeneinander
- Integration mit *Community Access Services*
  - Praktische Umsetzung der Grundidee des Grids

Grid-Computing ...

- Leistungsanalyse-Werkzeug
  - anwendungsorientierte Analyse interaktiver Jobs
- Praktische Erfahrungen
- Selbstadaptierende Anwendungen

Parallele Programmierung

- „Natürlicher“ Parallelismus + Sequentialisierung statt sequentielles Programm + Parallelisierung
- Trennung von Funktion und Ausführungsstrategie