Prozeßkontrolle

Dynamische Überwachung und Modifikation ablaufender Programme

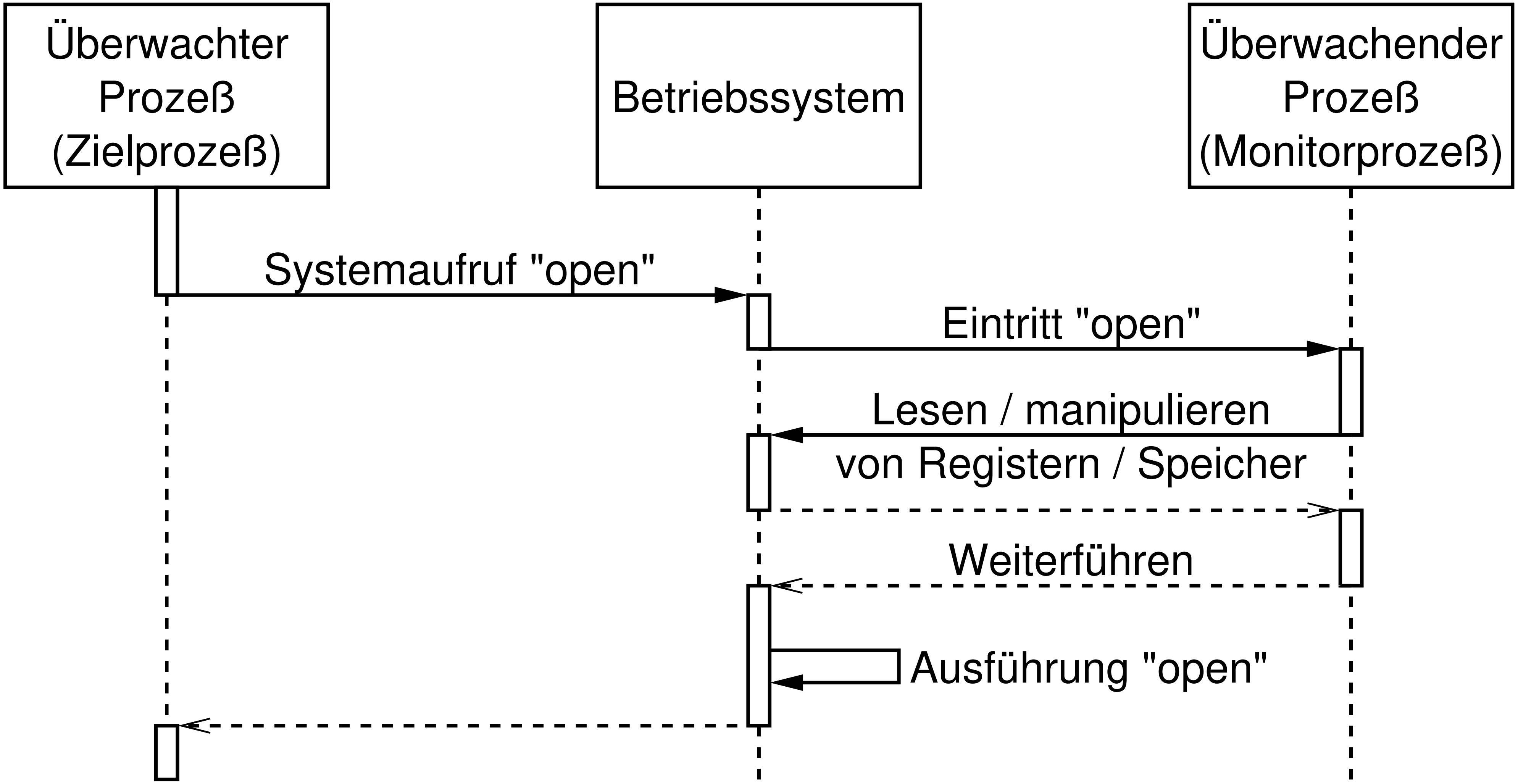
Roland Wismüller

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

Prozeßkontrolle

- Spezielle Form der Interprozeßkommunikation
- Ein Prozeß überwacht / steuert Ausführung eines anderen:
  - Systemaufrufe
  - Signale
  - Zugriff auf Speicher und Register
  - Anhalten / Beenden
- Ursprüngliche Motivation: Debugging
- Realisiert durch Betriebssystemdienste, z.B. in UNIX:
  - `ptrace`: Systemaufruf-Schnittstelle
  - `proc`: Dateisystem-Schnittstelle

Ablaufbeispiel



1. Einführung: Prozeßkontrolle

Höhere Dienste

- Dynamische Instrumentierung
  - Einfügen von Code in den Zielprozeß
  - effiziente Überwachung beliebiger Ereignisse
- *Inferior RPC*
  - Aufruf einer beliebigen Funktion im Zielprozeß
  - Manipulation von Registern, Keller und Befehlszähler
- Aufruf des dynamischen Binders durch *inferior RPC*
  - Einbinden größerer Codeteile in Zielprozeß
  - dabei Auflösung von Referenzen

1. Einführung: Prozeßkontrolle

Einsatz: Werkzeuge zur „Bearbeitung“ von Programmabläufen

- Fehlersuche
  - Protokoll-Generierung, z.B. Systemaufrufe
  - Ablaufvisualisierung (v.a. parallele Programme)
  - On-line Debugger: Haltepunkte, Zugriff auf Variable
- Leistungsanalyse
  - Anzahl / Laufzeit von Funktionsaufrufen
- Transparente Sicherungspunkt-Erstellung
- Abgesicherte (eingeschränkte) Ausführungsumgebung
- Integration von Anwendungsprogrammen

Agenda

Prozeßkontrolle

Dynamische Überwachung und Modifikation ablaufender Programme

1. Einführung: Prozeßkontrolle
2. Erweiterung: Verteiltes Monitorsystem
3. Anwendung: Abgesicherte Ausführungsumgebung
4. Anwendung: Middleware zur Verhaltensintegration
5. Fazit und Ausblick

2. Verteiltes Monitorsystem

Motivation

- Einfache, effiziente Nutzung der Prozeßkontrolle
- Erweiterung auf parallele und verteilte Systeme
- Simultane Nutzbarkeit durch mehrere Werkzeuge

Vorgehen

- Definition einer Online-Monitorschnittstelle
  - Projekt OMIS (*On-line Monitoring Interface Specification*)
- Realisierung des Monitorsystems
  - Projekt OCM (*OMIS Compliant Monitoring system*)
  - Zielsysteme: NOWs, PVM und MPI

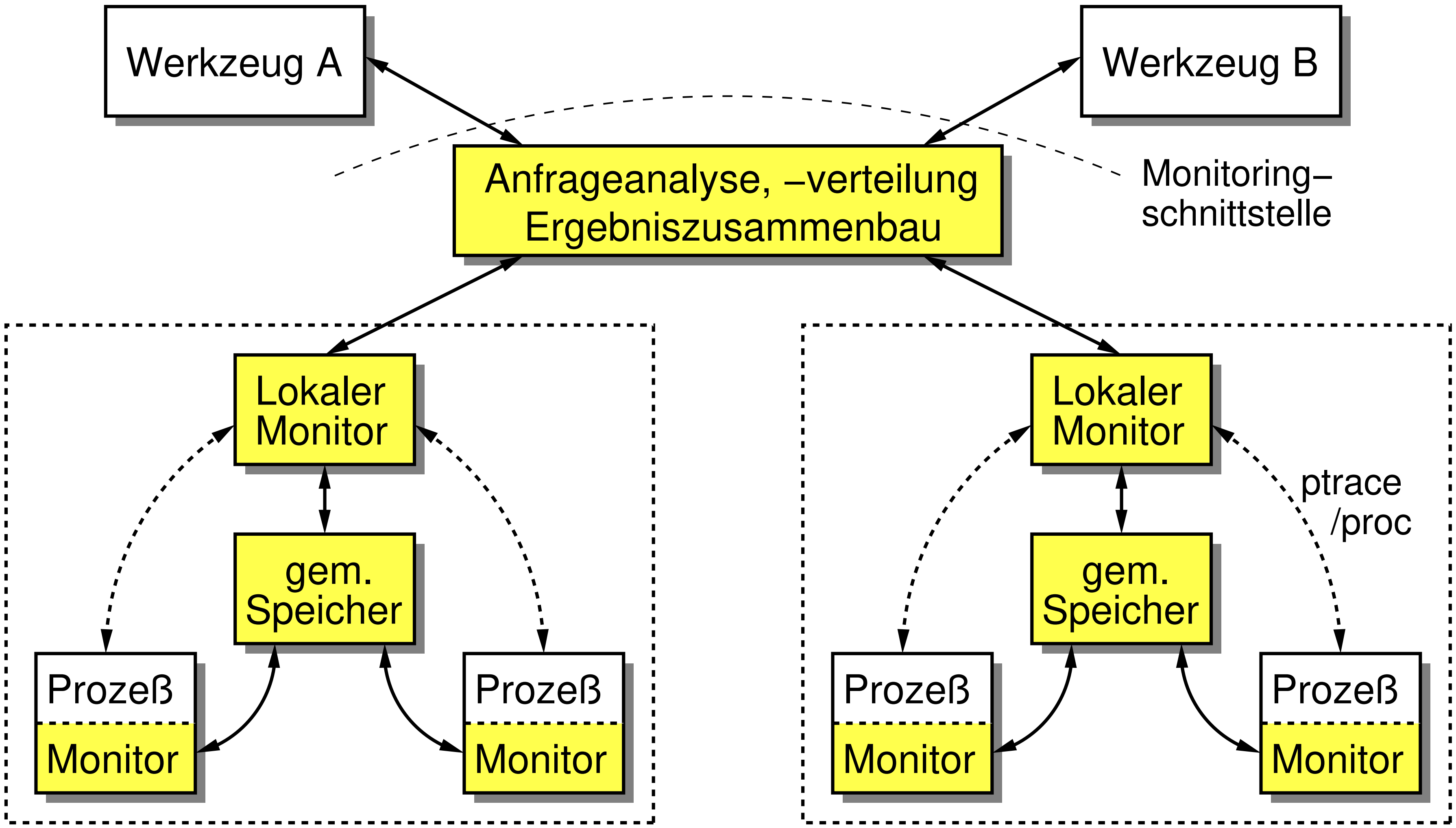
Konzepte von OMIS

- Objektbasiertes Modell des Zielsystems
  - Knoten, Prozesse, Threads, ...
- Programmierbare Ereignis/Aktions-Relationen

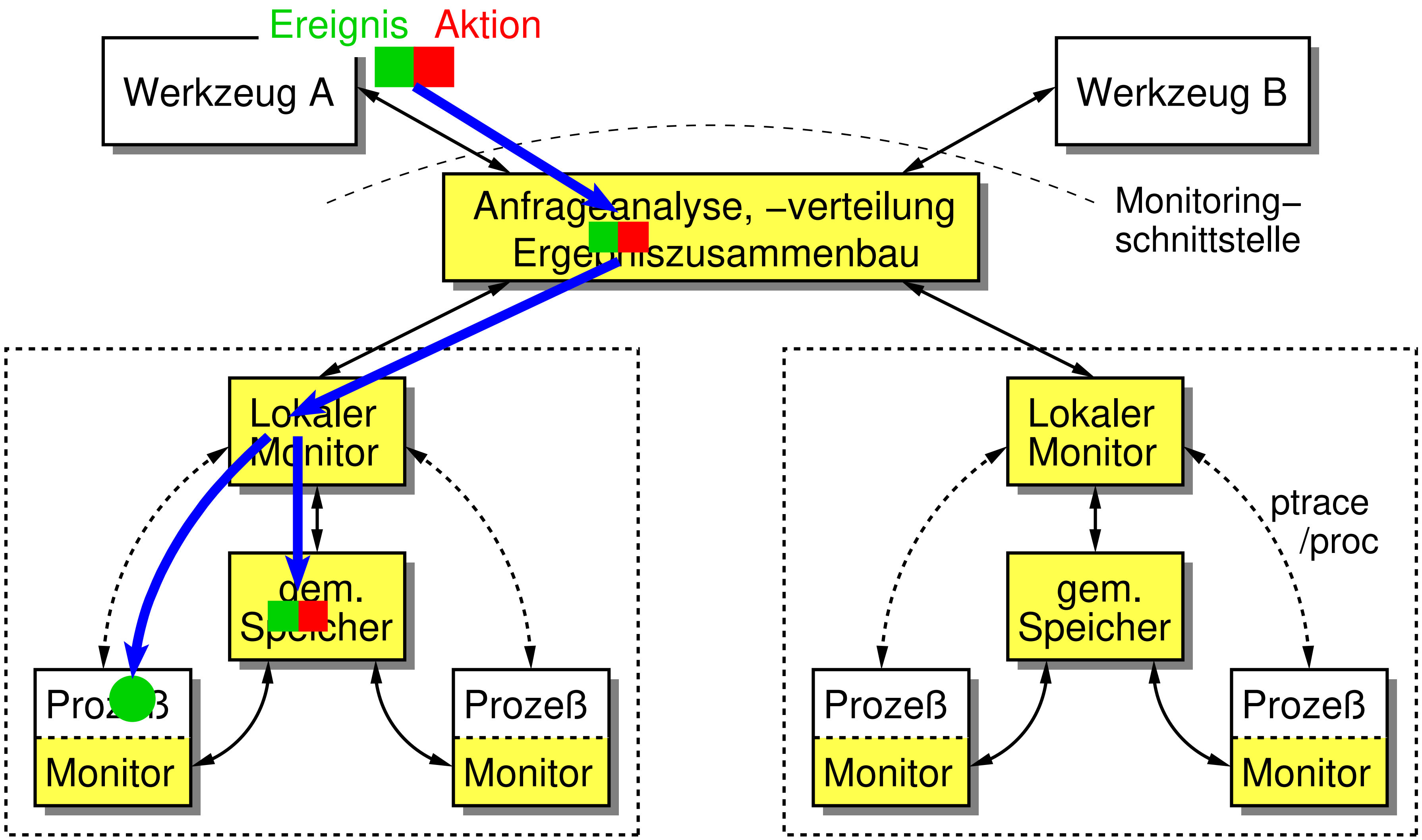
```
thread_started_lib_call([p_1],"pvm_send") :
    thread_stop([$node])
    thread_get_backtrace([$thread],0)
```

- Objektmengen, Ortstransparenz
  - einfache Nutzung, Skalierbarkeit, Optimierungen
- Erweiterbarkeit
  - (dynamisches) Einbinden neuer Objekte und Dienste

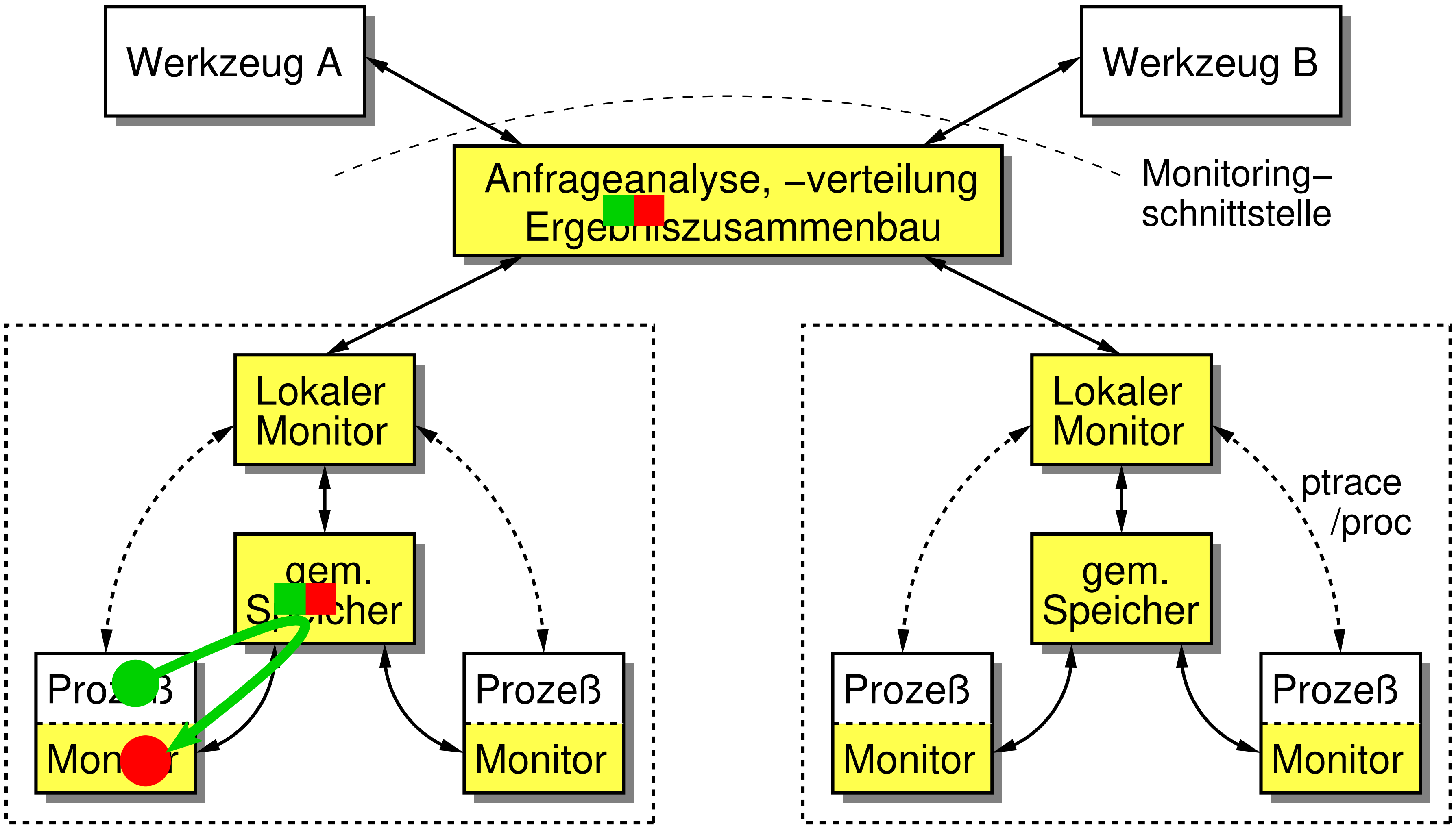
Architektur des OCM



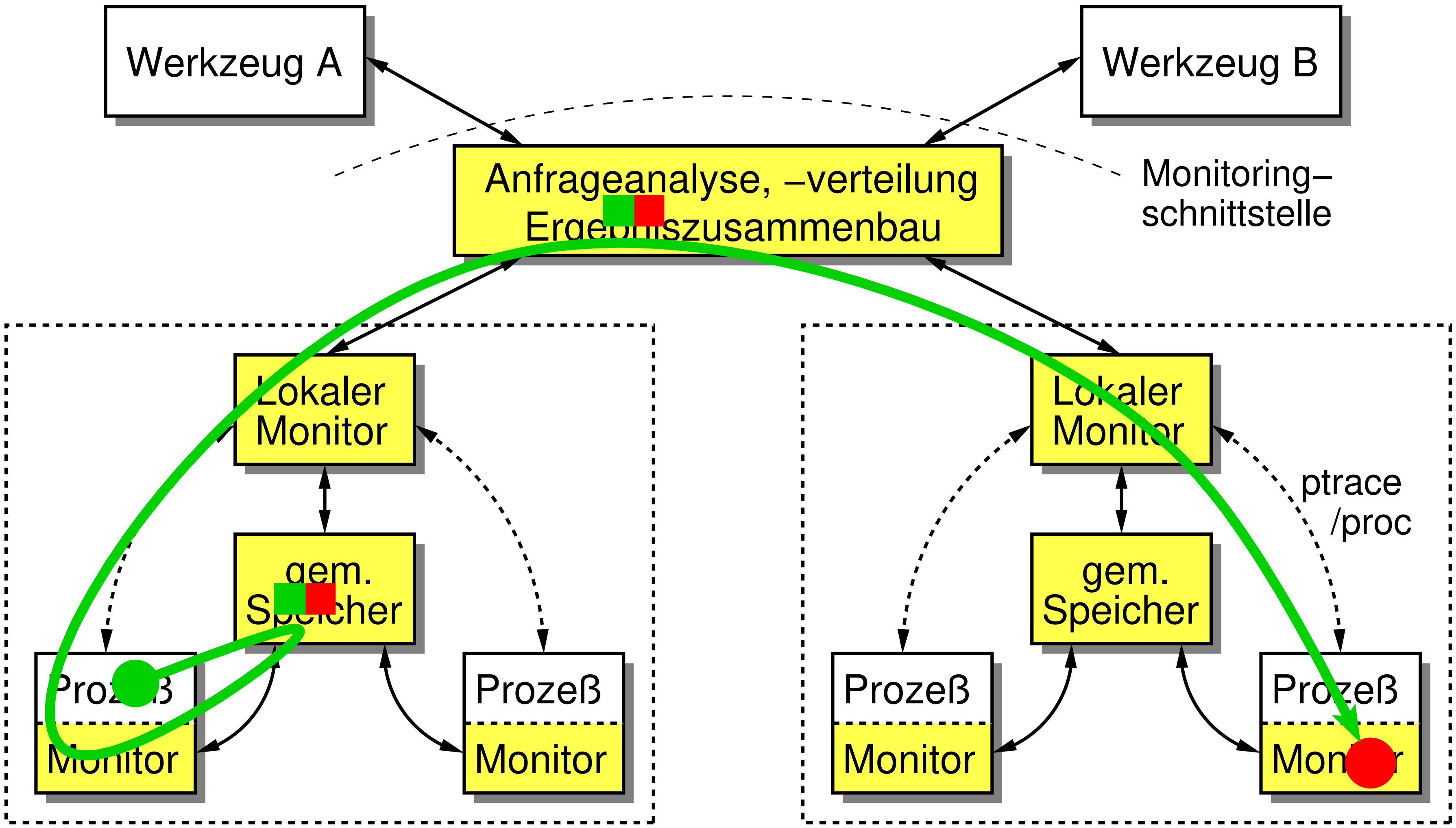
Anfrageabarbeitung im OCM



Anfrageabarbeitung im OCM



Anfrageabarbeitung im OCM



2. Verteiltes Monitorsystem

Ergebnisse

- OCM verfügbar für NOWs (Solaris, Linux, Irix)
  - Unterstützung für parallele Programme unter PVM und MPI
  - mehrere Werkzeuge auf Basis des OCM verfügbar

- Projekte (mit ICS-AGH Krakau):
  - Unterstützung für verteilte Java-Anwendungen (KBN)
  - Unterstützung für Multithreading auf SMPs (KBN)
  - Einsatz zur Online-Leistungsanalyse interaktiver Grid-Anwendungen (EU CrossGrid)

3. Abgesicherte Ausführungsumgebung

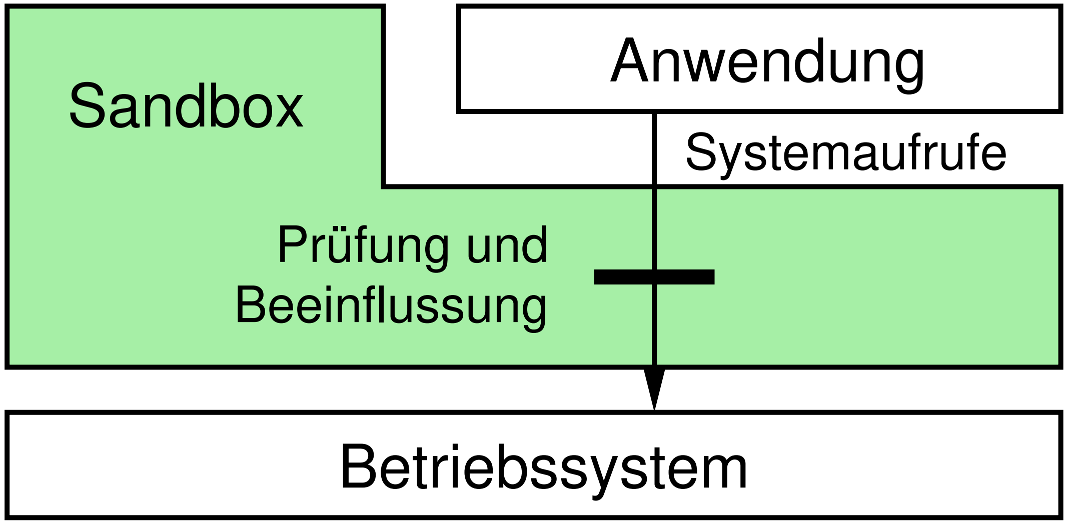
Motivation

- Betriebssystem schützt nur Benutzer gegeneinander
  - kein Schutz der Resourcen eines Benutzers vor *eigenen* Prozessen
- Problemsituationen:
  - Ausführung unbekannter Software ($\rightarrow$ Viren, Trojaner)
  - Grid computing: gemeinsame Nutzung einer Kennung (*Community Access Services*)
- Lösung:
  - Ausführung in gesicherter Umgebung („*Sandbox*“)
  - Umgebung schränkt Zugriffsrechte ein

3. Abgesicherte Ausführungsumgebung

Zur Realisierung

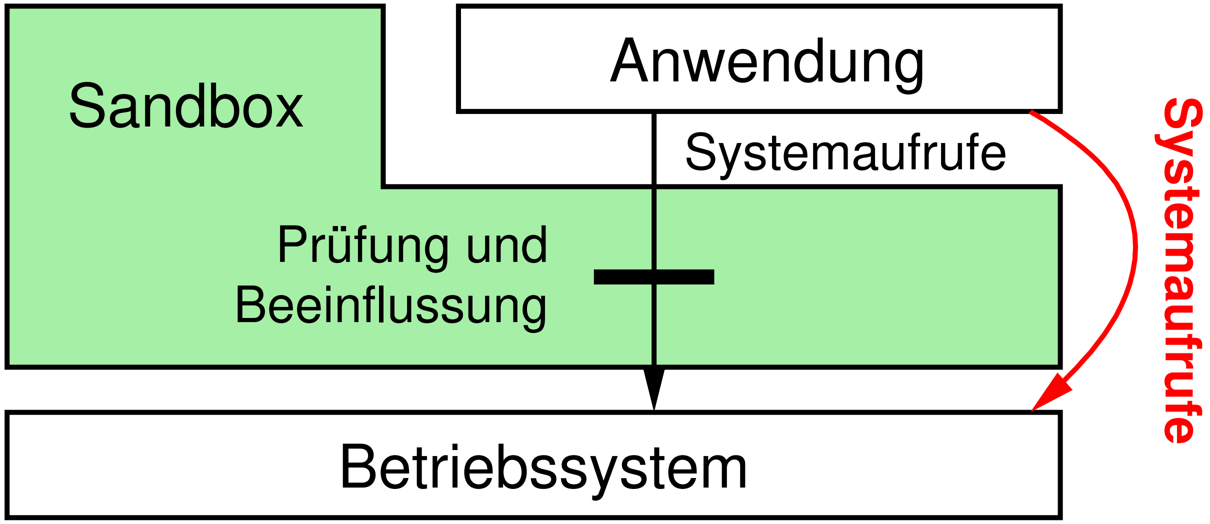
- Basis: Zugriff auf Resourcen nur über Systemaufrufe
- Einschränkung der Zugriffsrechte durch Überwachung aller Systemaufrufe (Prozeßkontrolle)
  - Bei Rechteverletzung: Abbruch des Aufrufs / Programms



3. Abgesicherte Ausführungsumgebung

Zur Realisierung

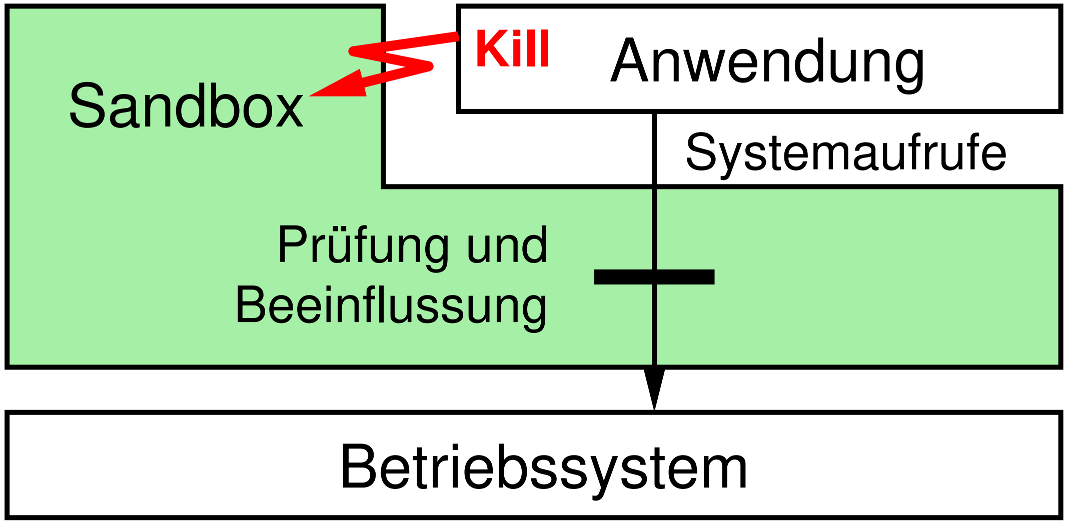
- Basis: Zugriff auf Resourcen nur über Systemaufrufe
- Einschränkung der Zugriffsrechte durch Überwachung aller Systemaufrufe (Prozeßkontrolle)
  - Bei Rechteverletzung: Abbruch des Aufrufs / Programms



3. Abgesicherte Ausführungsumgebung

Zur Realisierung

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

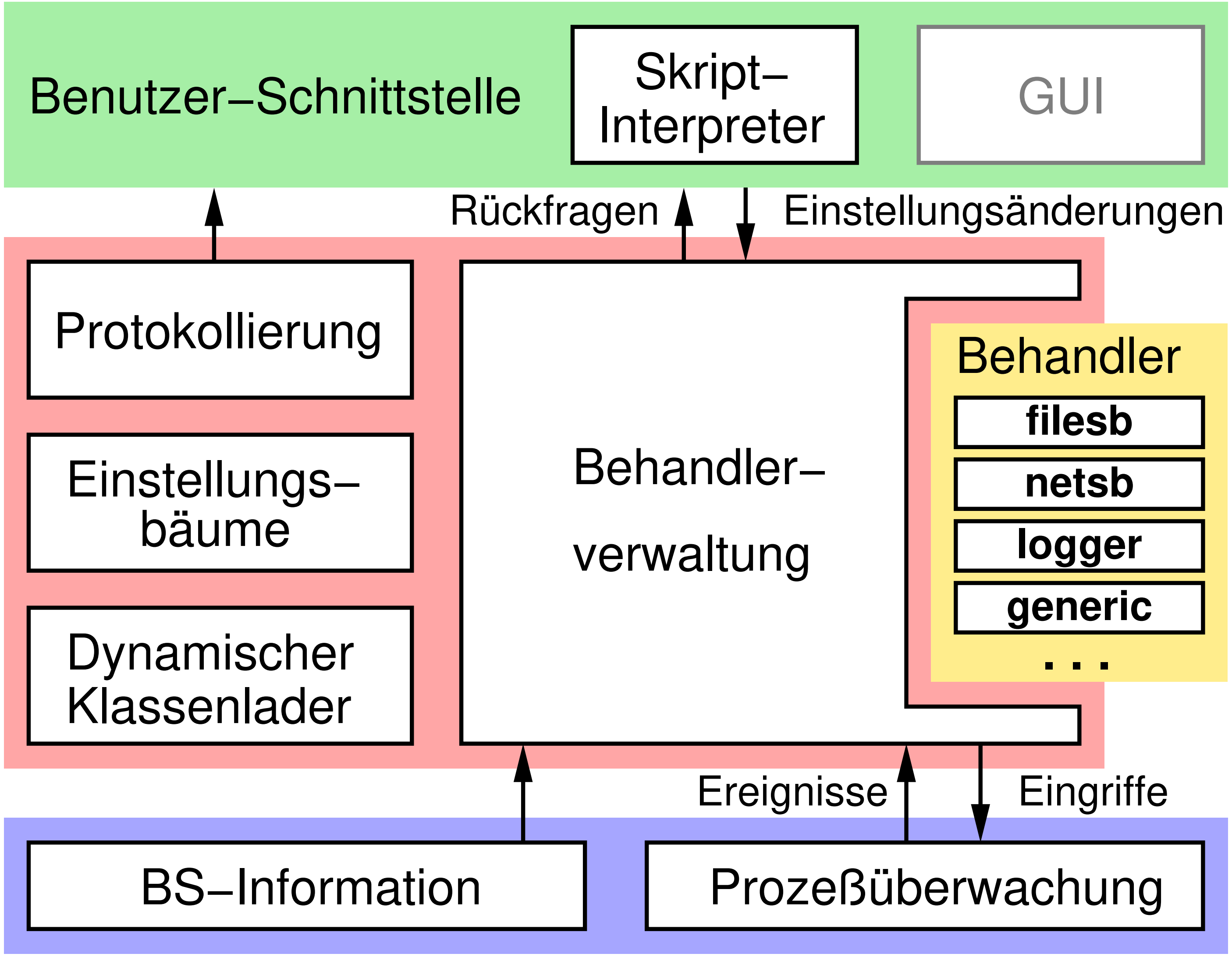


3. Abgesicherte Ausführungsumgebung

Realisierungsprobleme

- Sicherheit
  - Lückenlose Überwachung (Kindprozesse, „Übergriffe“)
- Systemabhängigkeit
  - Prozeßkontroll-Schnittstellen, Systemaufrufe
- Konfigurierbarkeit
  - Abbildung von Rechten (*Capabilities*) auf Systemaufrufe
- Erweiterbarkeit
  - Einbinden weiterer Überwachungen

Architektur der Sandbox



3. Abgesicherte Ausführungsumgebung

Ergebnis

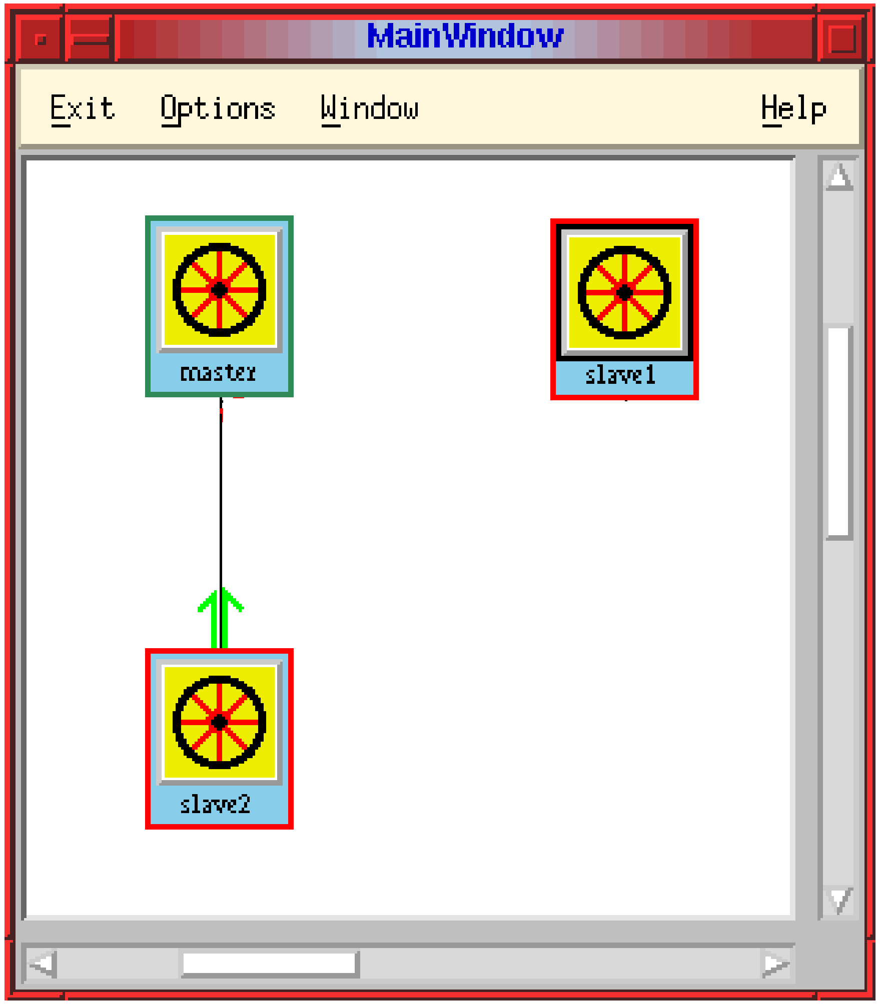
- Sandbox für Linux einsatzbereit
  - frei konfigurierbar, „ausbruchssicher“
  - normaler Benutzerprozeß (keine `root`-Rechte)
- Überwachung von Dateizugriffen
  - Rechte für reguläre Mengen von Dateien definierbar
  - gegenüber UNIX verfeinertes Rechtemodell
- Überwachung von Netzwerkzugriffen
  - Filterung über Protokollfamilie, Host, Port
- Laufzeit-Overhead: 10-100%

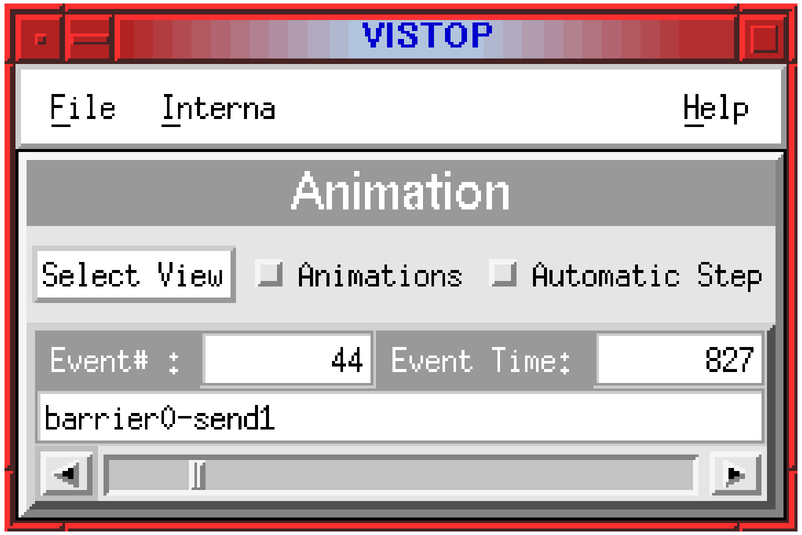
4. Middleware zur Verhaltensintegration

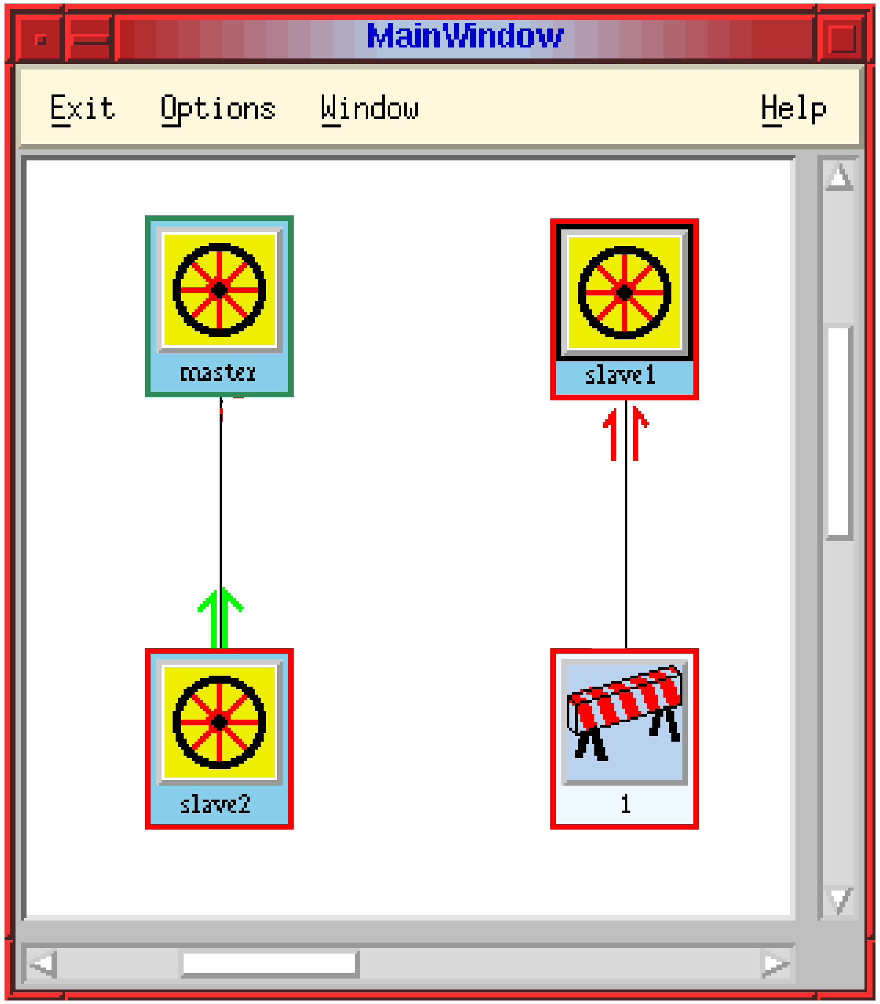
Motivation

- Arbeitsumgebung aus mehreren Einzelanwendungen
  - Wunsch: (lose) Zusammenarbeit der Anwendungen
- Konkreter Hintergrund am LRR-TUM:
  - Entwicklung verschiedenster Online-Werkzeuge
    - Quellsprach-Debugger, Ablaufvisualisierer, Leistungsanalyse, deterministische Ausführung, Sicherungspunkterstellung, ...
  - Wunsch: Kombination von Visualisierer + Debugger
    - Überblick über Ablaufverhalten
    - detaillierte Untersuchung

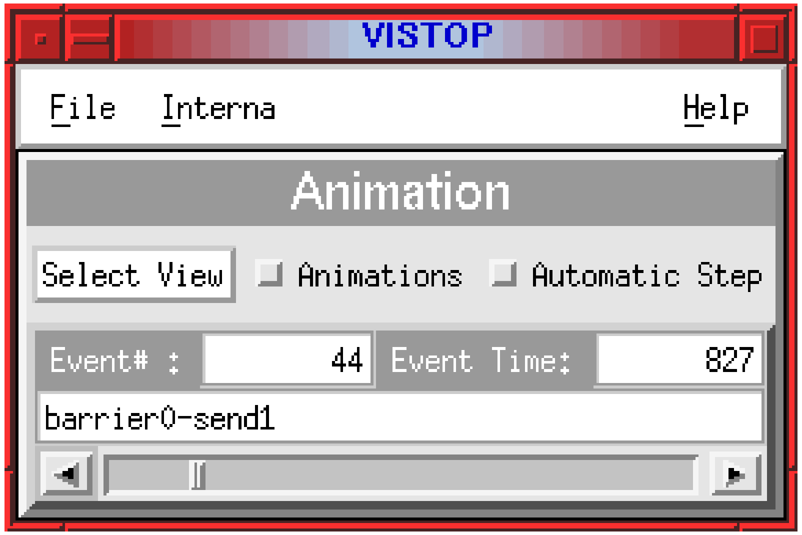
4. Middleware zur Verhaltensintegration



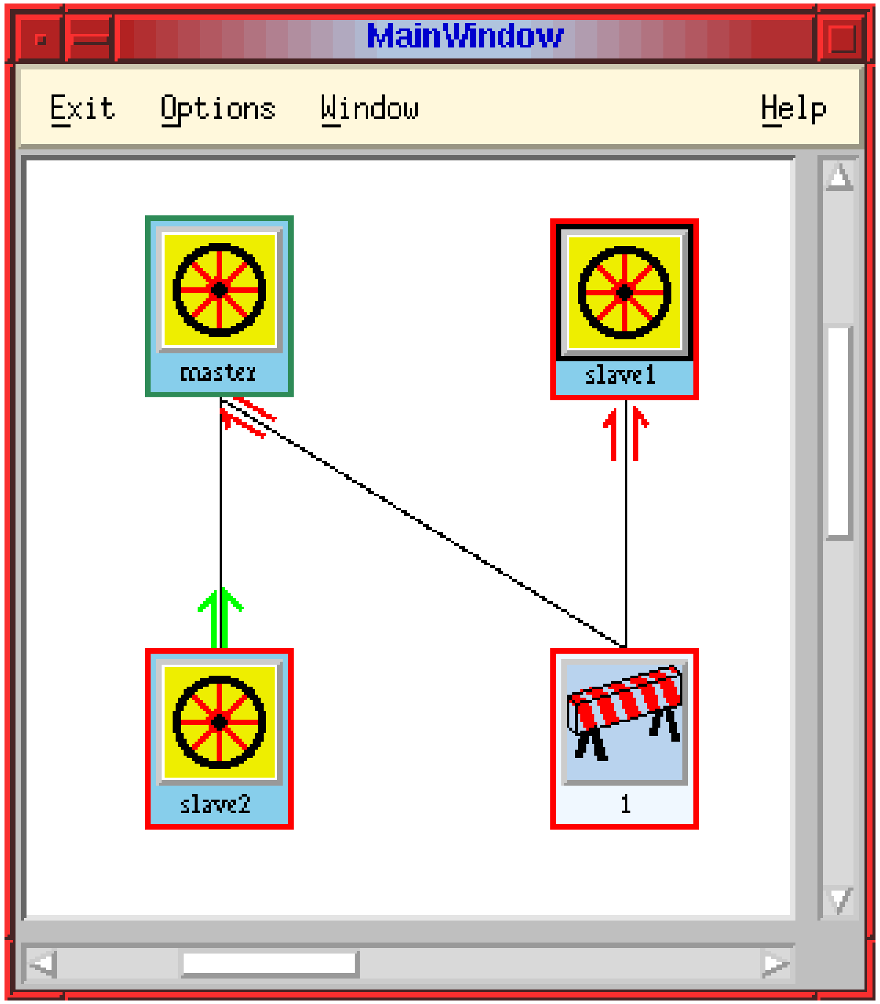


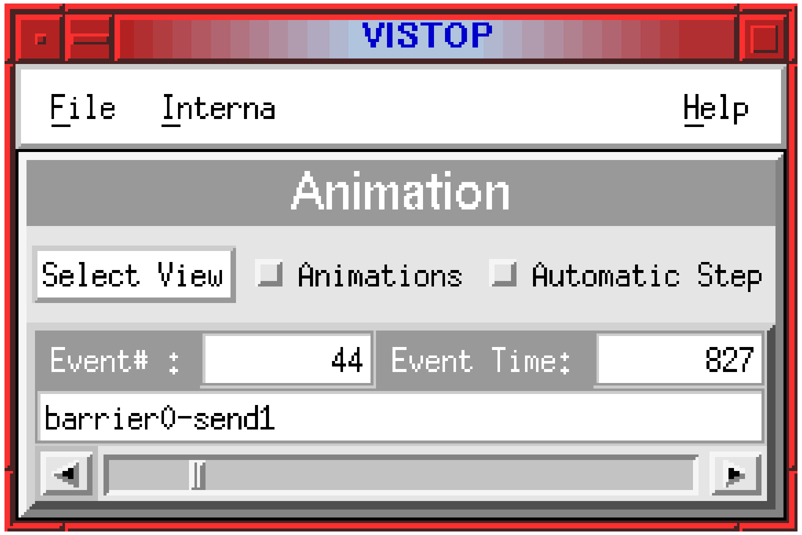


MainWindow
Exit
Options
Window
Help
master
slave1
slave2
1

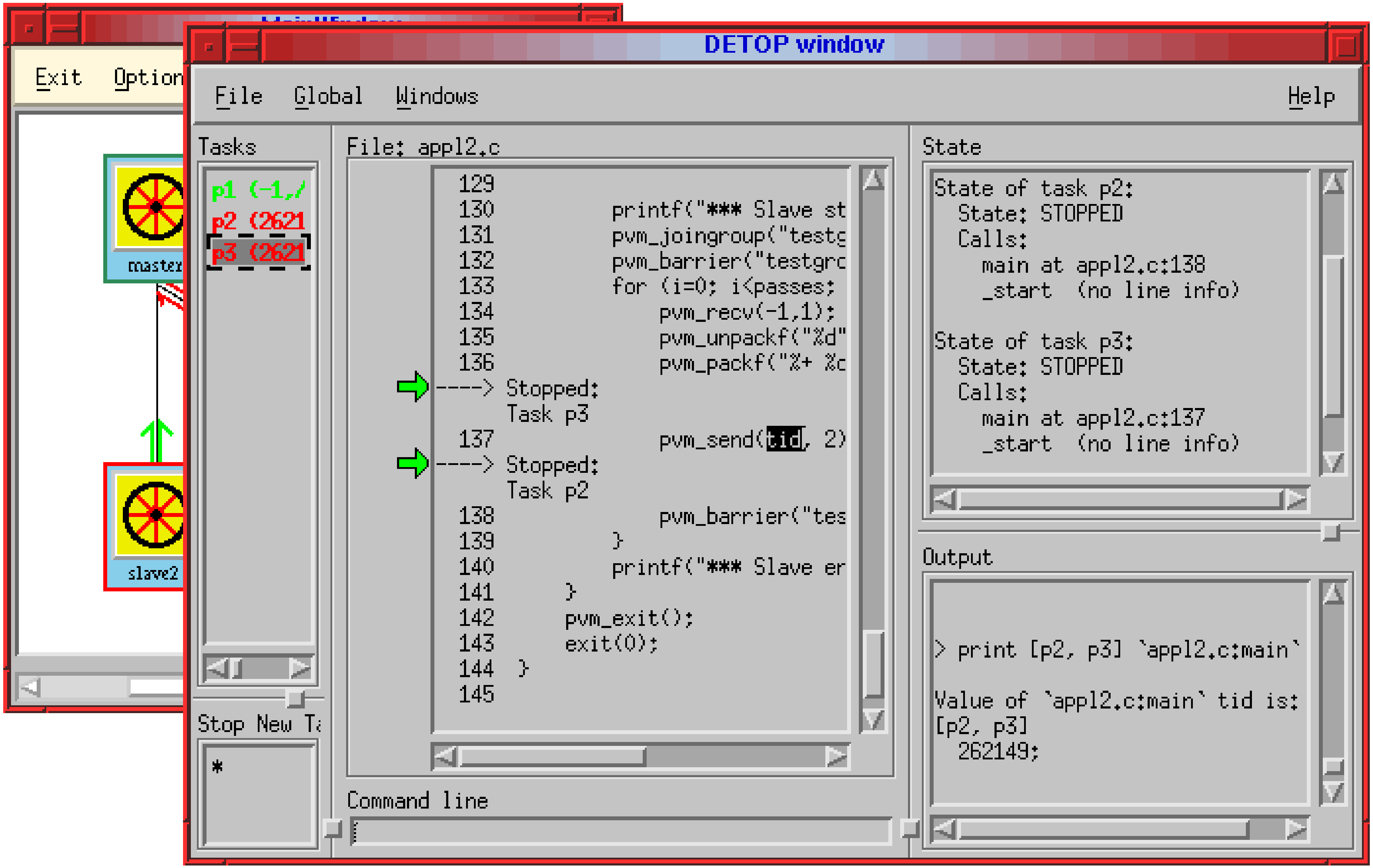


VISTOP
File
Interna
Help
Animation
Select View
Animations
Automatic Step
Event# :
44
Event Time:
827
barrier0-send1

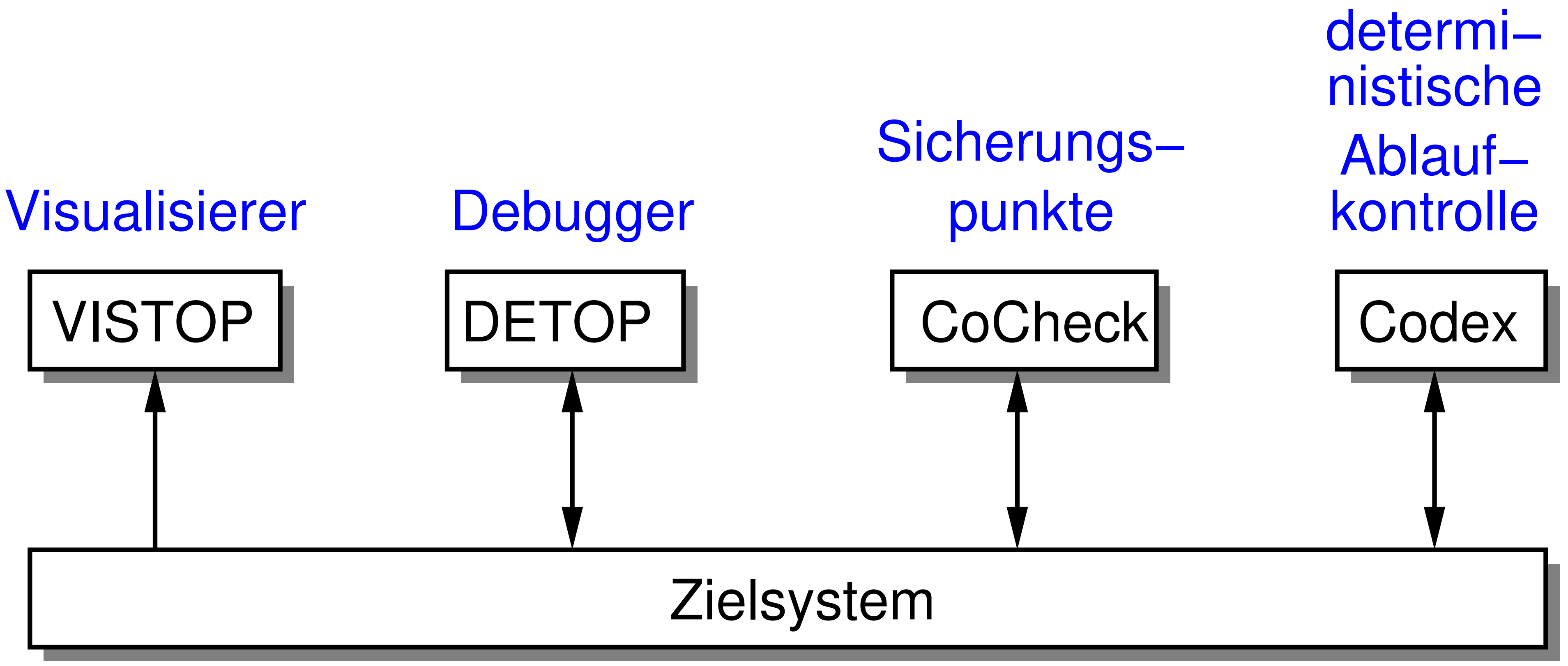

MainWindow
Exit Options Window Help
master
slave1
slave2
1


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

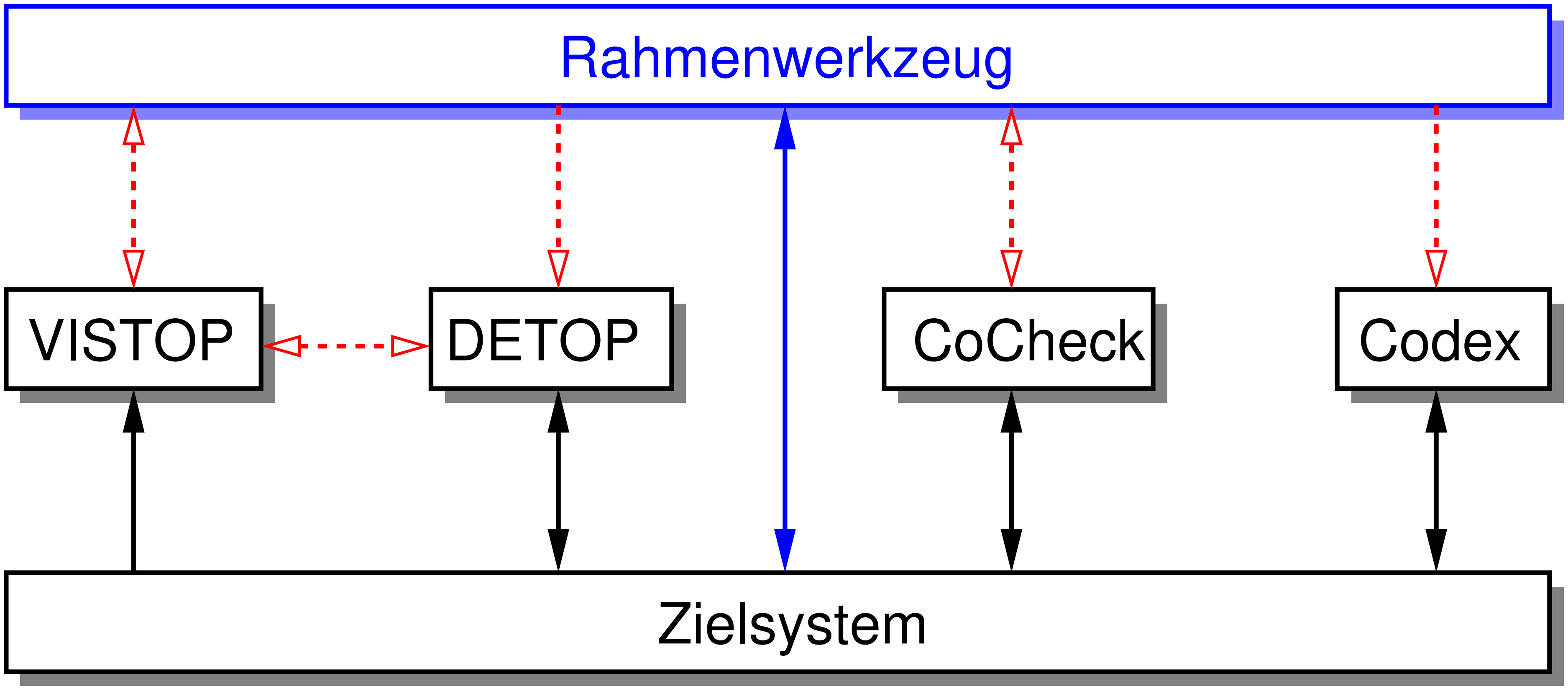
4. Middleware zur Verhaltensintegration



Aufbau der Werkzeugumgebung



Aufbau der Werkzeugumgebung



4.1. Verhaltensintegration

Ziel: Kooperation zwischen Anwendungsprogrammen

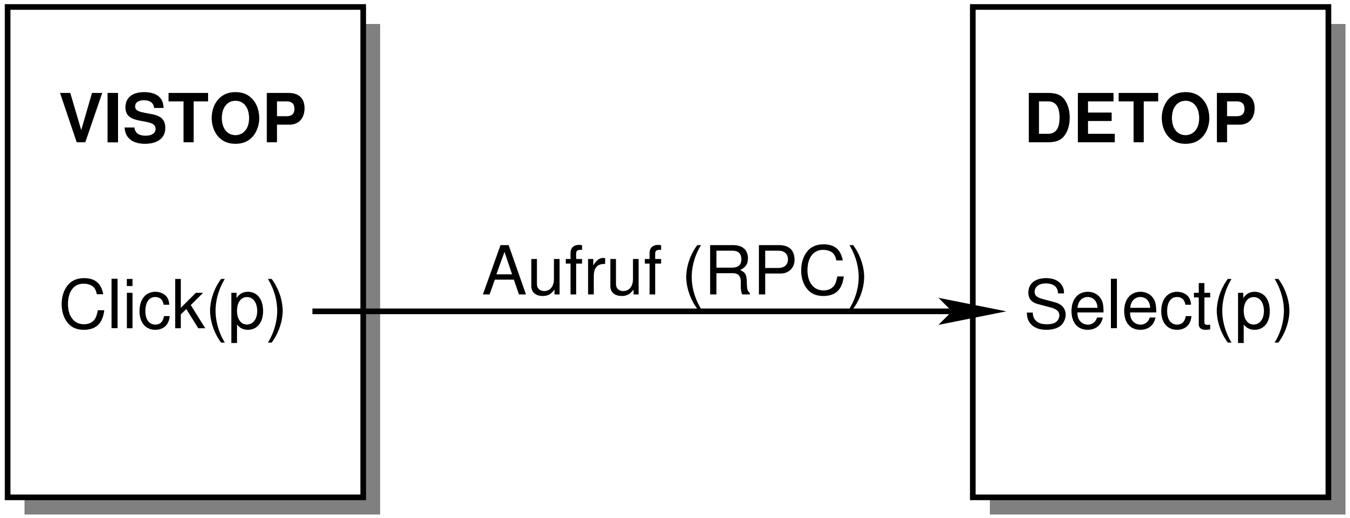
- Synchronisierte Objektauswahl, konsistente Darstellung
- Steuerung der Werkzeuge durch das Rahmenwerkzeug

Allgemeine Fragestellung:

- Erstellung von Software-Umgebungen aus vorhandenen Anwendungsprogrammen (= Komponenten)
  - aktiv, mit eigenständigem Verhalten (im Ggs. zu Servern)
  - i.a. mit Benutzerinteraktion
  - unabhängig voneinander entwickelt, unabhängige Nutzung
  - i.a. keine Modifikation der Komponenten möglich

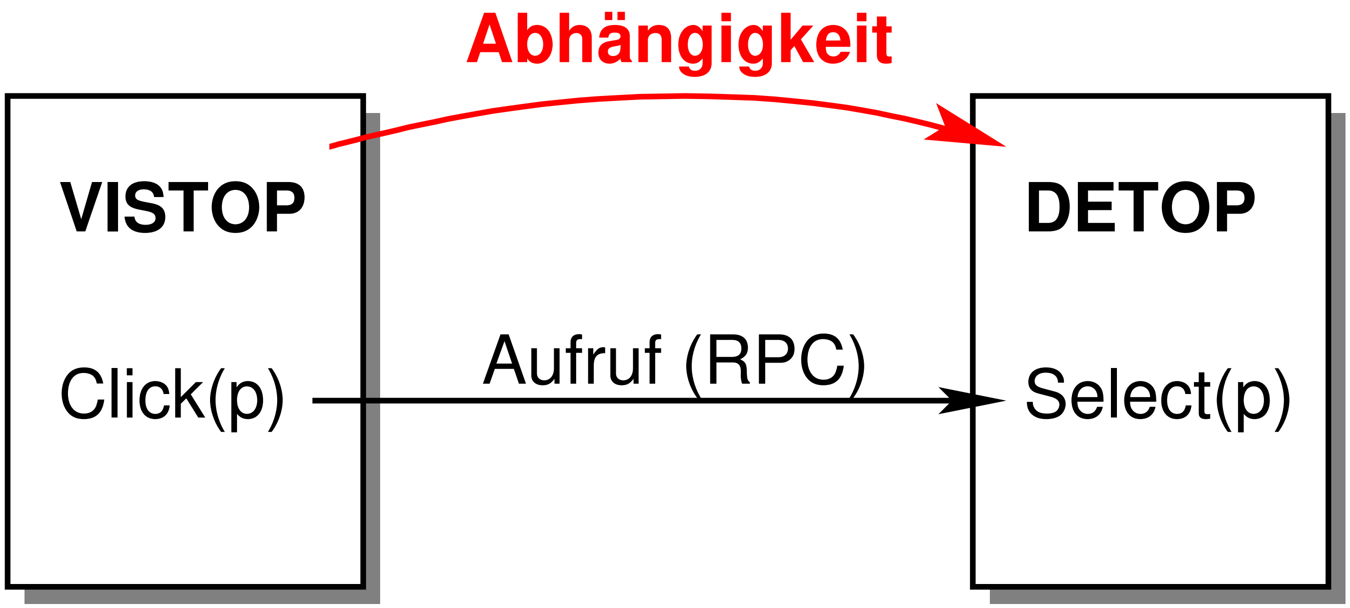
Einfachster Ansatz: Feste Verdrahtung

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



Einfachster Ansatz: Feste Verdrahtung

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

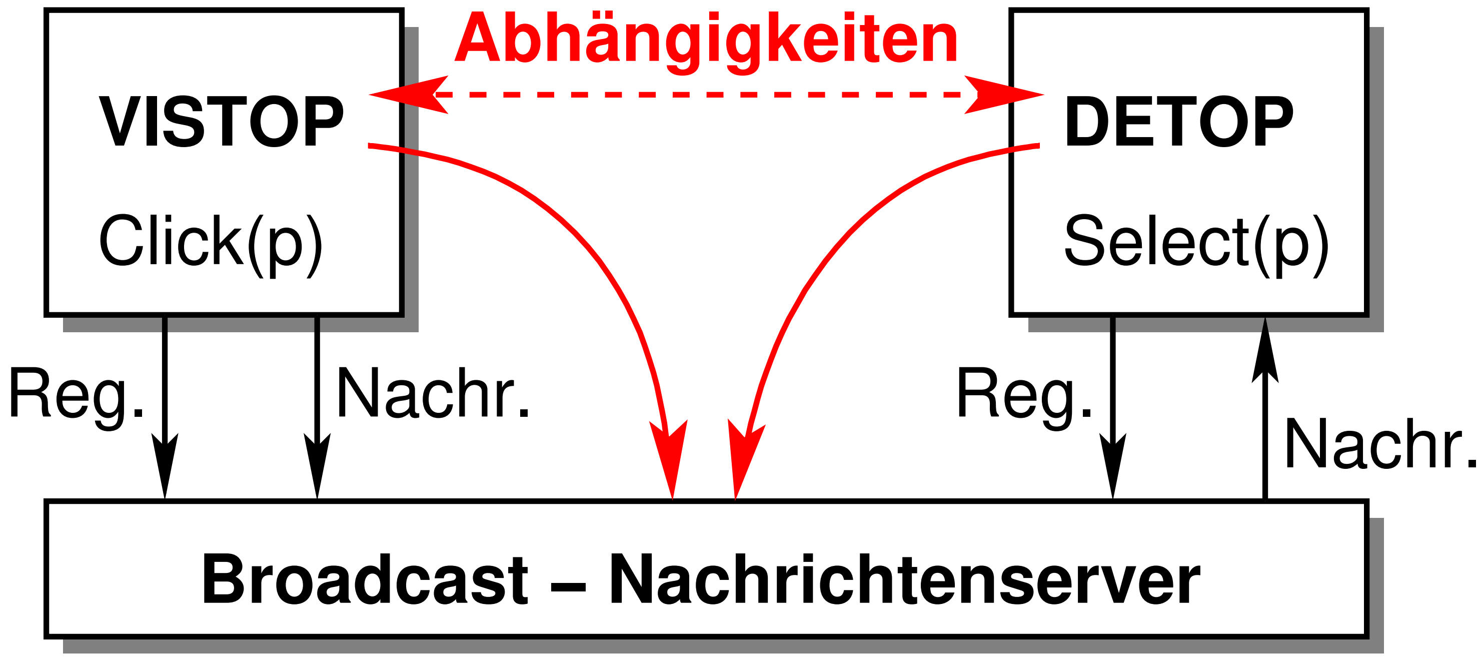


Broadcast-Nachrichtenserver

- Zentraler Server verteilt Nachrichten
- Z.B. ToolTalk, SoftBench

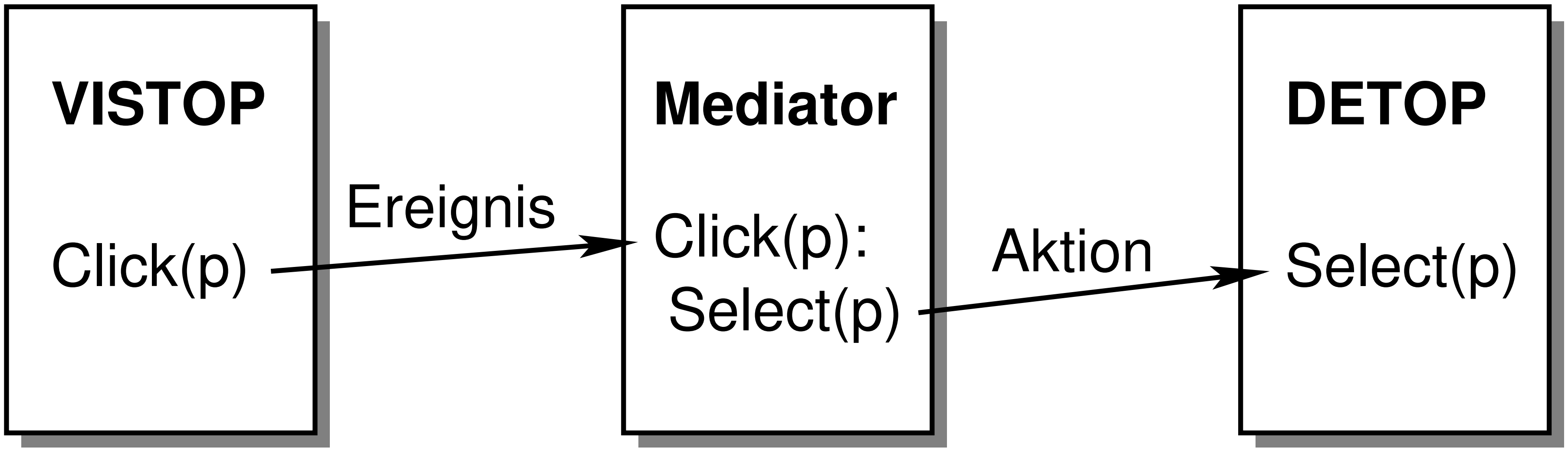
Broadcast-Nachrichtenserver

- Zentraler Server verteilt Nachrichten
- Z.B. ToolTalk, SoftBench



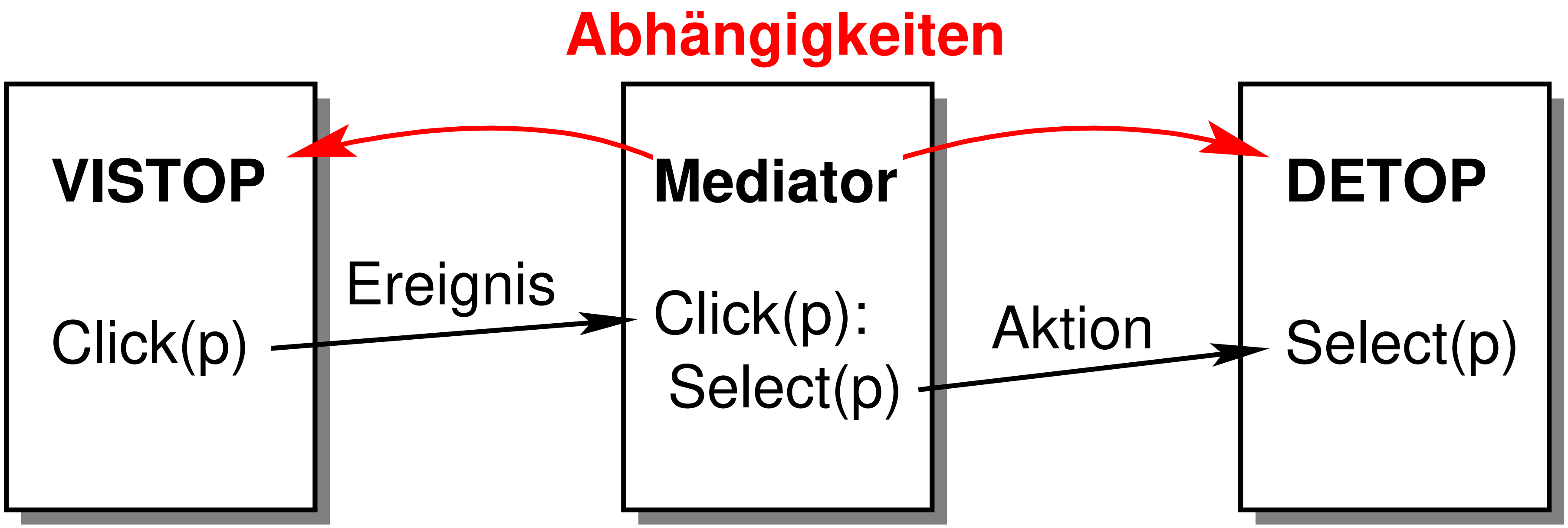
Mediatoren / Interaktoren

- 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

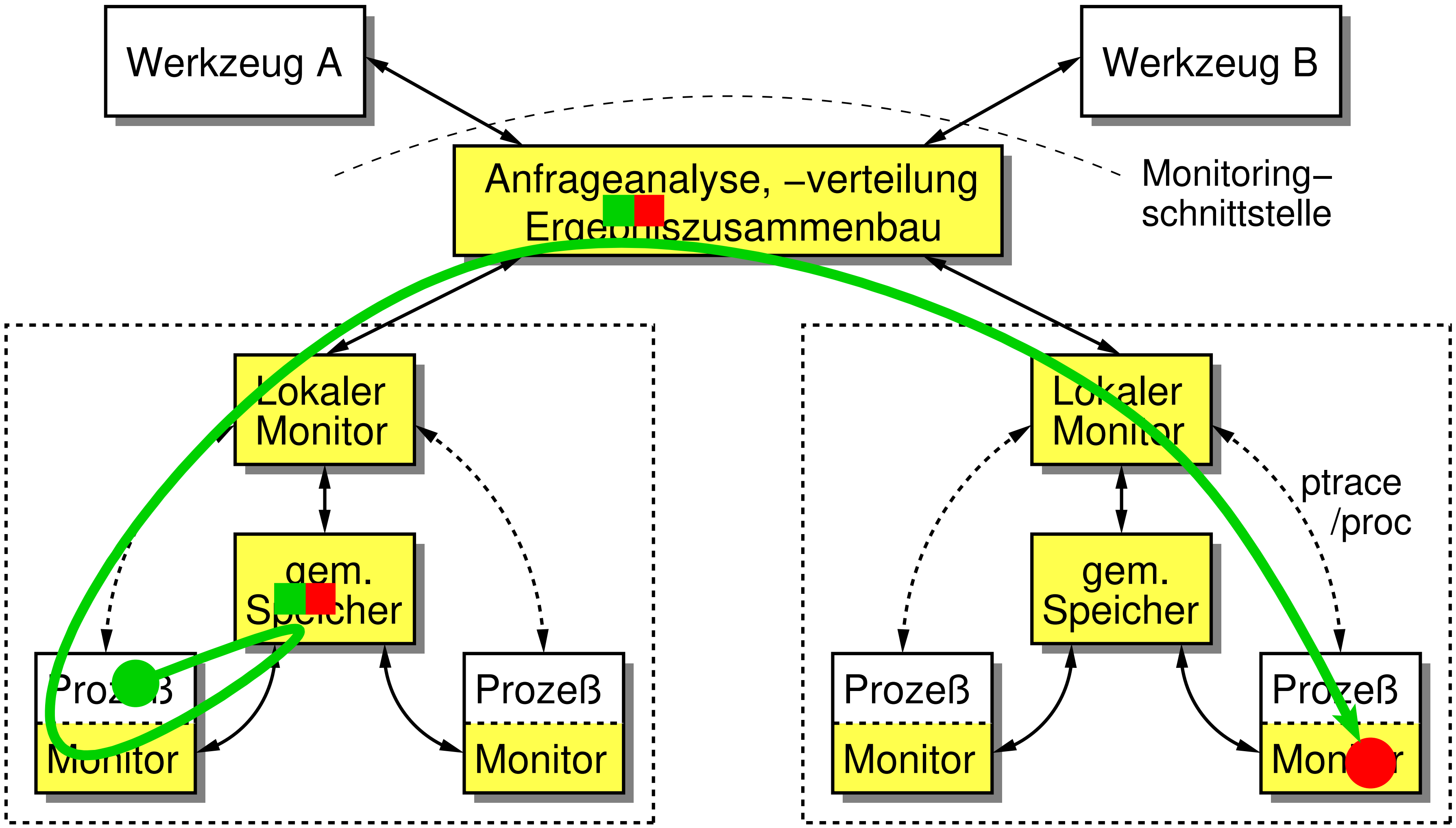


4.1. Verhaltensintegration

Problem:
Komponenten müssen Schnittstellen a-priori 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
- Hier: Komponenten für isolierte Nutzung entwickelt
  - Integration in Umgebung erfordert Modifikation

Anfrageabarbeitung im OCM



4.2. Verhaltensintegration durch den OCM

Spezielle Eigenschaften des OCM:

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

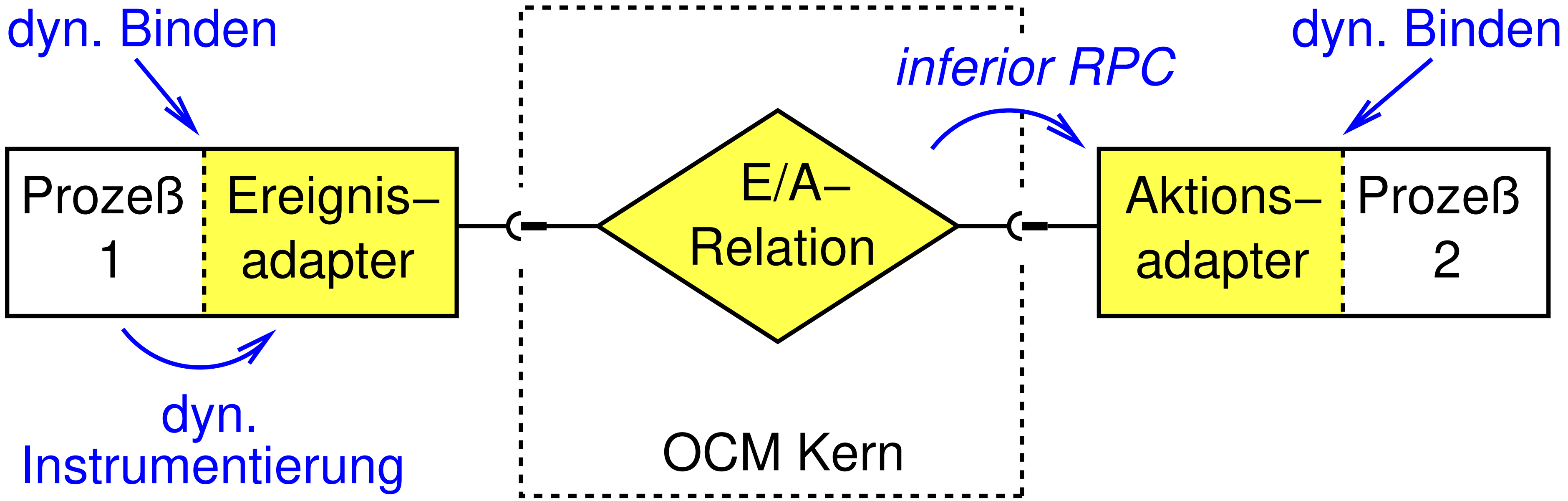
ohne spezielle Vorbereitung dieser Prozesse

- Menge der Ereignisse und Aktionen zur Laufzeit erweiterbar

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

Zur Realisierung:

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

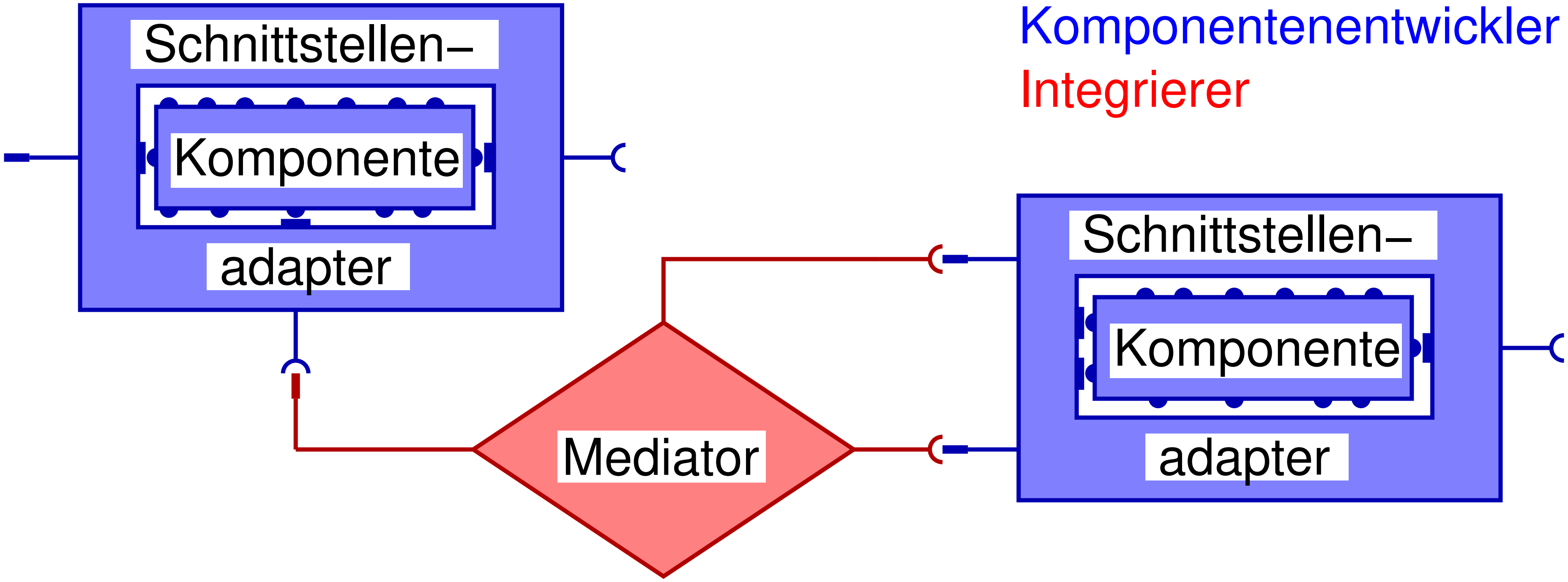


4.3. Ergebnis

Prozeßkontrolle ermöglicht Verhaltensintegration durch Mediatoren ohne a-priori Erfordernis von Schnittstellen

- Schnittstellen(adapter) werden zur Laufzeit angebunden
  - 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
- Schnittstellen an sich / Dokumentation müssen vorhanden sein!

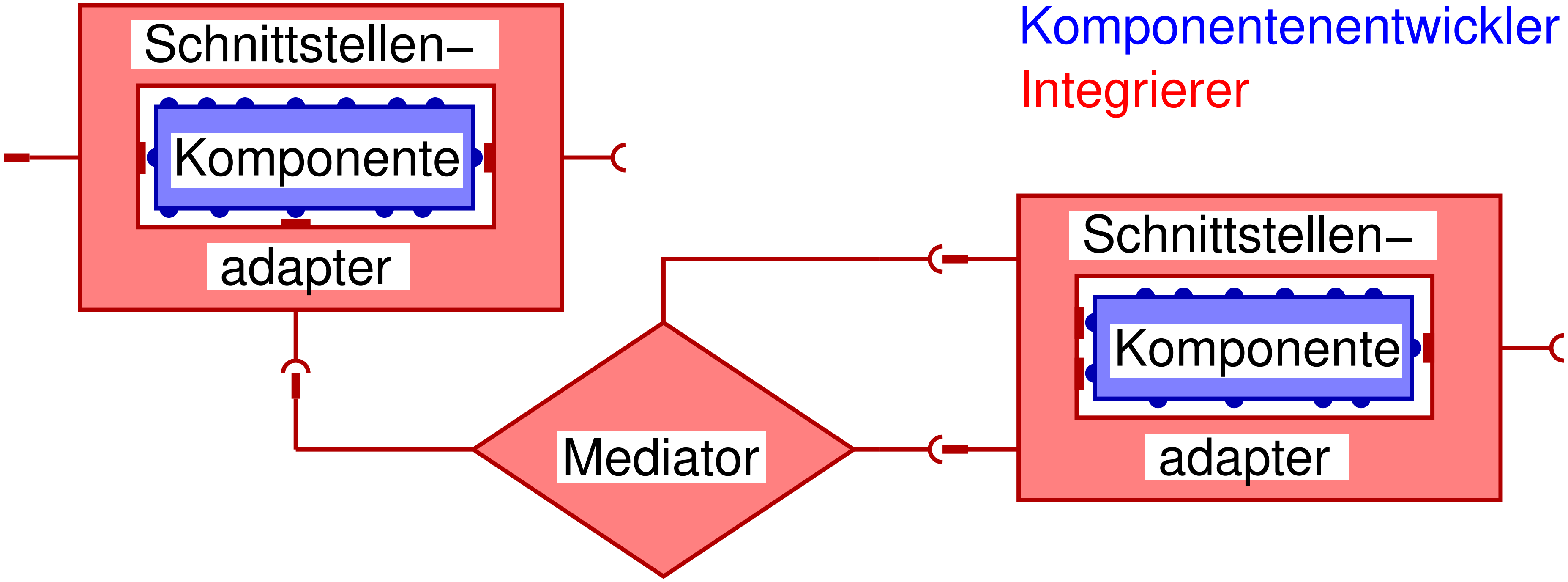
Traditionelles Vorgehen



- Schnittstellenadapter meist unnötiger „Ballast“ im Code
- Abhängigkeit von (Vorhandensein der) Middleware

4.3. Ergebnis

Dynamisches Anbinden von Schnittstellen



- Mächtigere (interne) Schnittstelle nutzbar
- Einfache und effiziente Anpassung der Schnittstelle möglich

4.3. Ergebnis

Praktischer Einsatz

- Prototyp der Werkzeugumgebung
  - Sichern des Programmzustands, automatisches Rücksetzen
  - Konsistenz VISTOP $\leftrightarrow$ DETOP
- Zur Verhaltensintegration: 3 Ereignisse + 10 Aktionen
  - eine Aktion mit komplexerem Schnittstellenadapter
  - ohne Modifikation des Werkzeug-Quellcodes
- Voraussetzung: Datenintegration durch gemeinsames Monitorsystem
  - muß beim Entwurf der Komponenten berücksichtigt werden

5. Fazit und Ausblick

Prozeßkontrolle

- Programme in Ausführung sind Objekte, die bearbeitet werden können
- Mehr Anwendungen als nur Programmanalyse

Weitere Arbeiten

- Weiterentwicklung des OCM
- Einsatz der Sandbox im Grid (*Community Access Services*)
- Komfortable Umgebung zur Verhaltensintegration
- Unterstützung / Nutzung in Betriebssystemen