Interoperable Laufzeit-Werkzeuge für parallele und verteilte Systeme

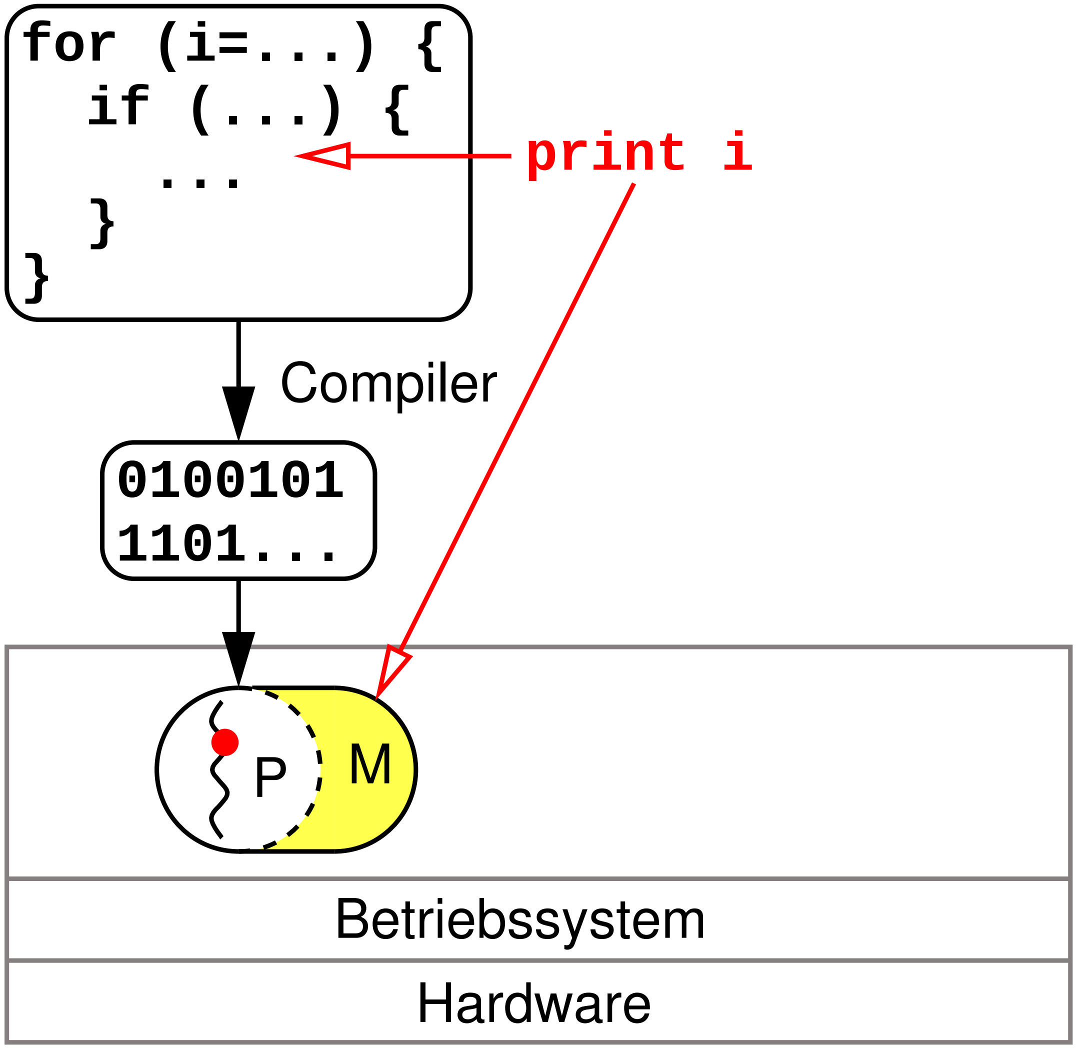
Roland Wismüller

LRR-TUM
Technische Universität München

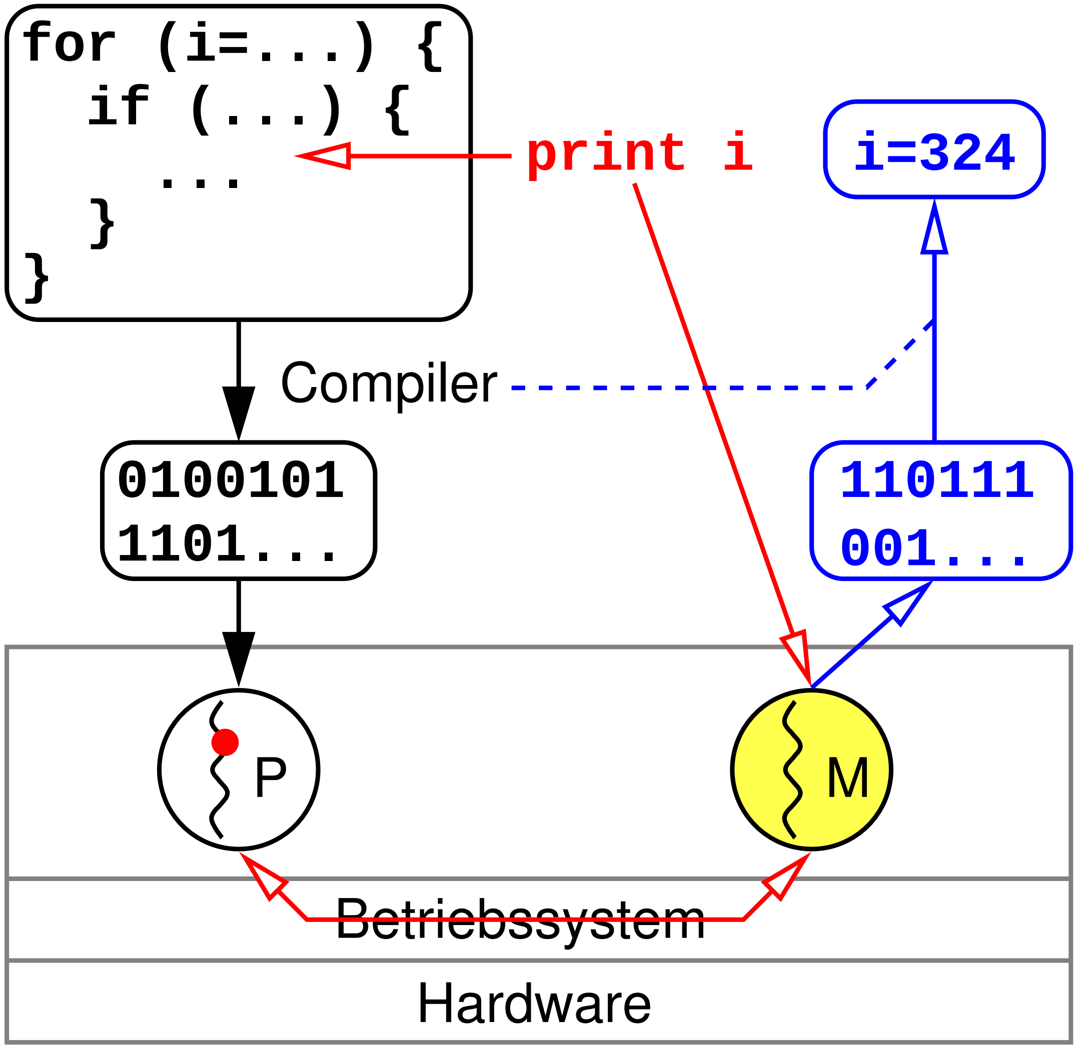
1.1. Werkzeuge zur Laufzeitanalyse (Laufzeit-Werkzeuge)

- Laufzeit-Werkzeuge *beobachten*, *analysieren* und *steuern* den Ablauf eines Soft- und Hardwaresystems (Zielsystem)
- Beispiele:
  - Leistungsanalyse-Werkzeug
    - wie lange dauert die Ausführung einer Funktion?
    - welche Datenmenge wird gelesen/geschrieben?
  - Debugger
    - welche Anweisungen werden ausgeführt?
    - Inhalte von Variablen?
  - deterministische Ablaufsteuerung
    - Beobachtung / Vertauschung asynchroner Ereignisse

1.2. Monitoring und Instrumentierung



1.2. Monitoring und Instrumentierung

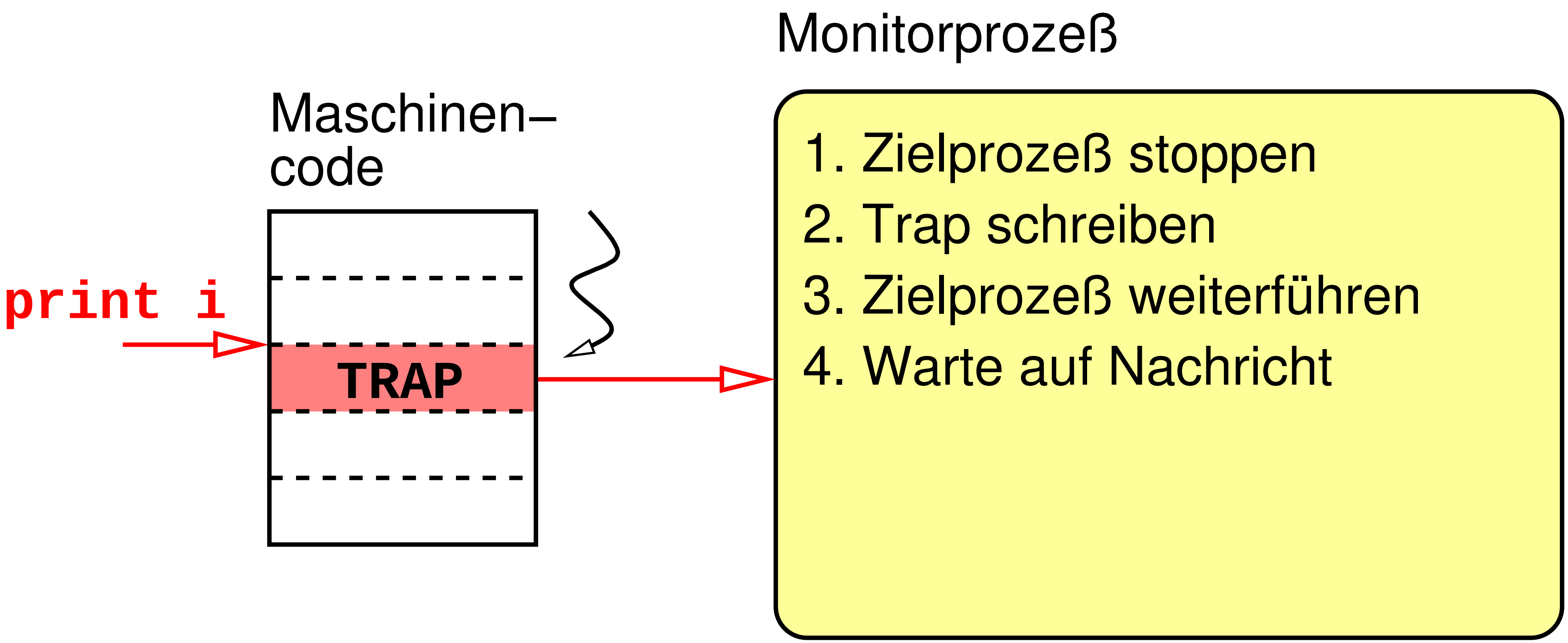


1.2. Monitoring und Instrumentierung

- Monitoring: Mechanismus der Beobachtung/Steuerung
  - in beobachtete Prozesse integriert
  - separate Monitorprozesse (Monitorsystem)
- Instrumentierung: Einbringen von Sonden in das beobachtete Programm
  - statisch: im Quellcode des Programms
  - dynamisch: im (laufenden) Maschinencode
- Auswertung der gewonnenen Information
  - off-line: nach Ende des beobachteten Programms
  - on-line: noch während der Ausführung

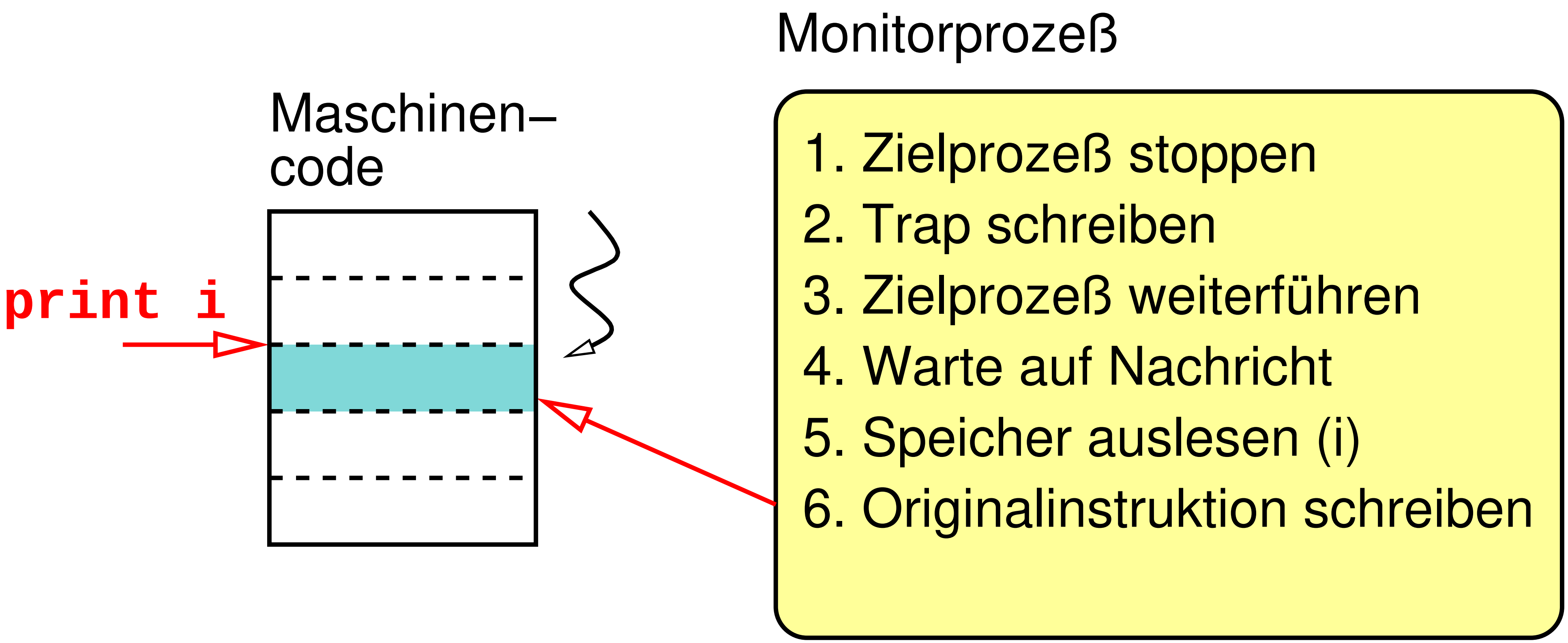
1.3. Dynamische Instrumentierungstechniken

➥ Trap-Befehle und Betriebssystemdienste



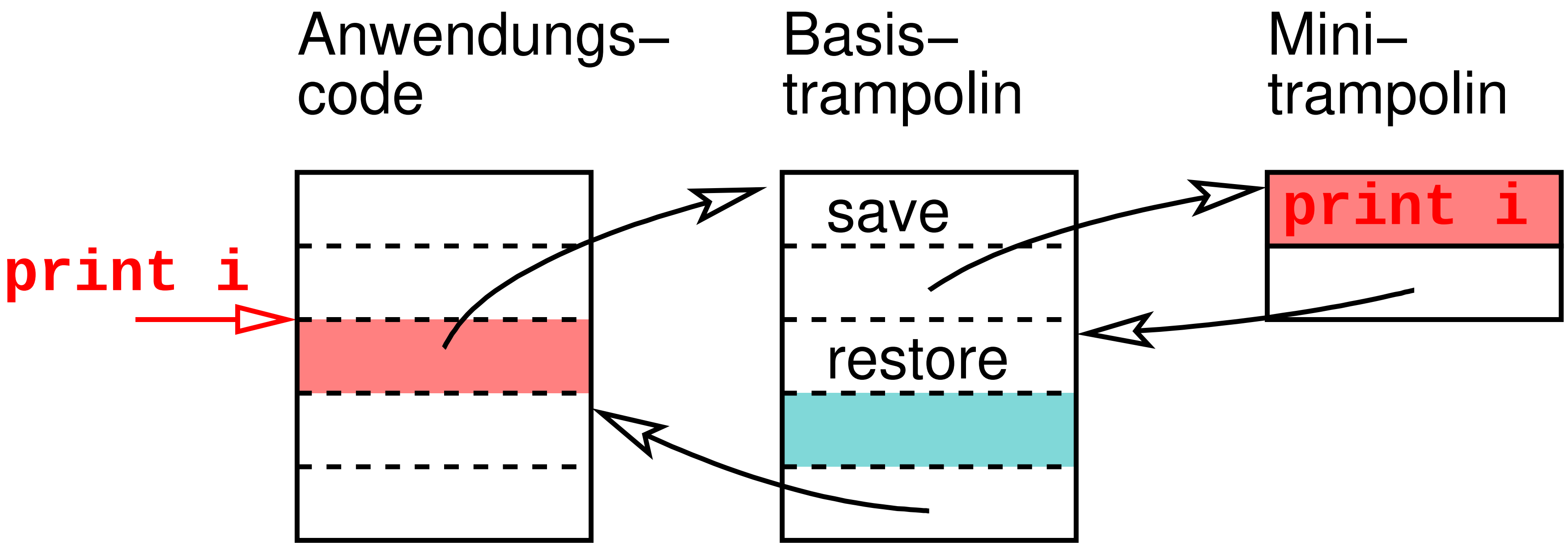
1.3. Dynamische Instrumentierungstechniken

- Trap-Befehle und Betriebssystemdienste



1.3. Dynamische Instrumentierungstechniken

- Dynamische Instrumentierung mit *dyninst*



1.3. Instrumentierungstechniken

- Dynamische Instrumentierung mit *dyninst*
  - Einbinden komplexerer Aktionen
    - Nutzung des dynamischen Binders
    - Aufruf im Zielprozeß durch *inferior RPC*
  - *inferior RPC*
    - Sichern des Prozessorstatus
    - Manipulation von Befehlszähler und Keller (Unterprogrammaufruf)
    - Anhalten des Prozesses bei Rückkehr aus Unterprogramm
    - Restaurieren des Prozessorstatus

Interoperable Laufzeit-Werkzeuge für parallele und verteilte Systeme

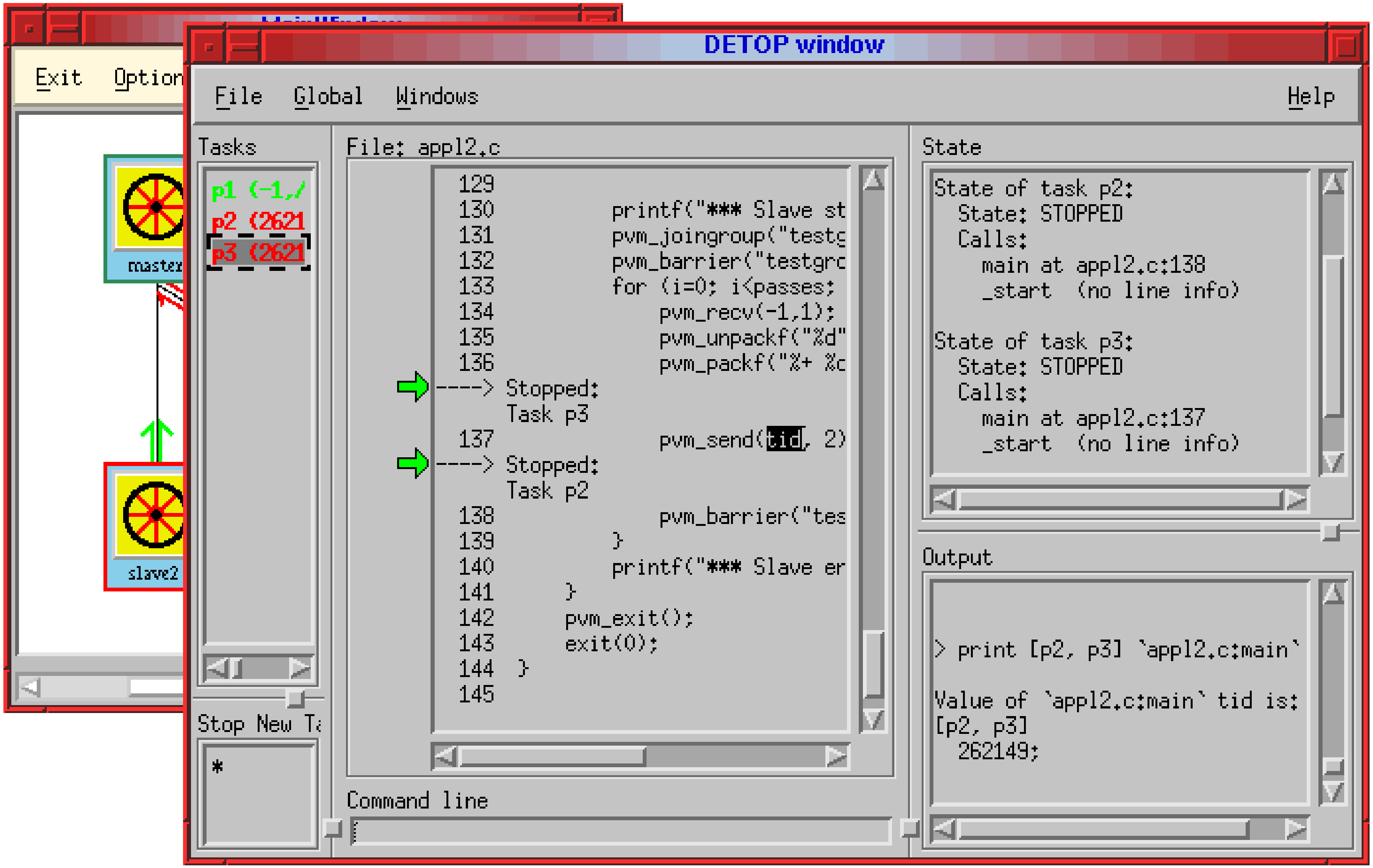
- Einführung
- Motivation: Eine Werkzeugumgebung
- Daten- und Verhaltensintegration
- Resultat und praktische Erfahrungen
- Fazit und Ausblick

2. Motivation: Eine Werkzeugumgebung

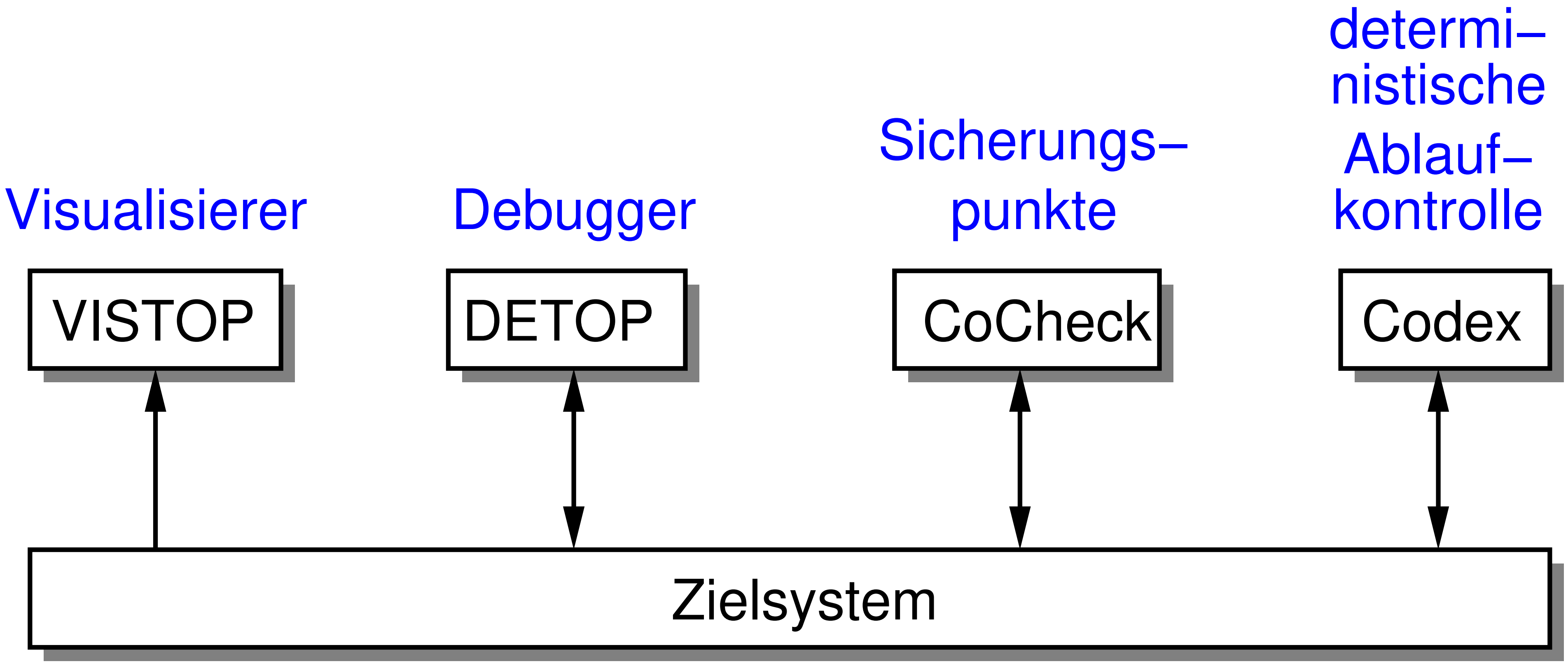
Hintergrund:

- 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

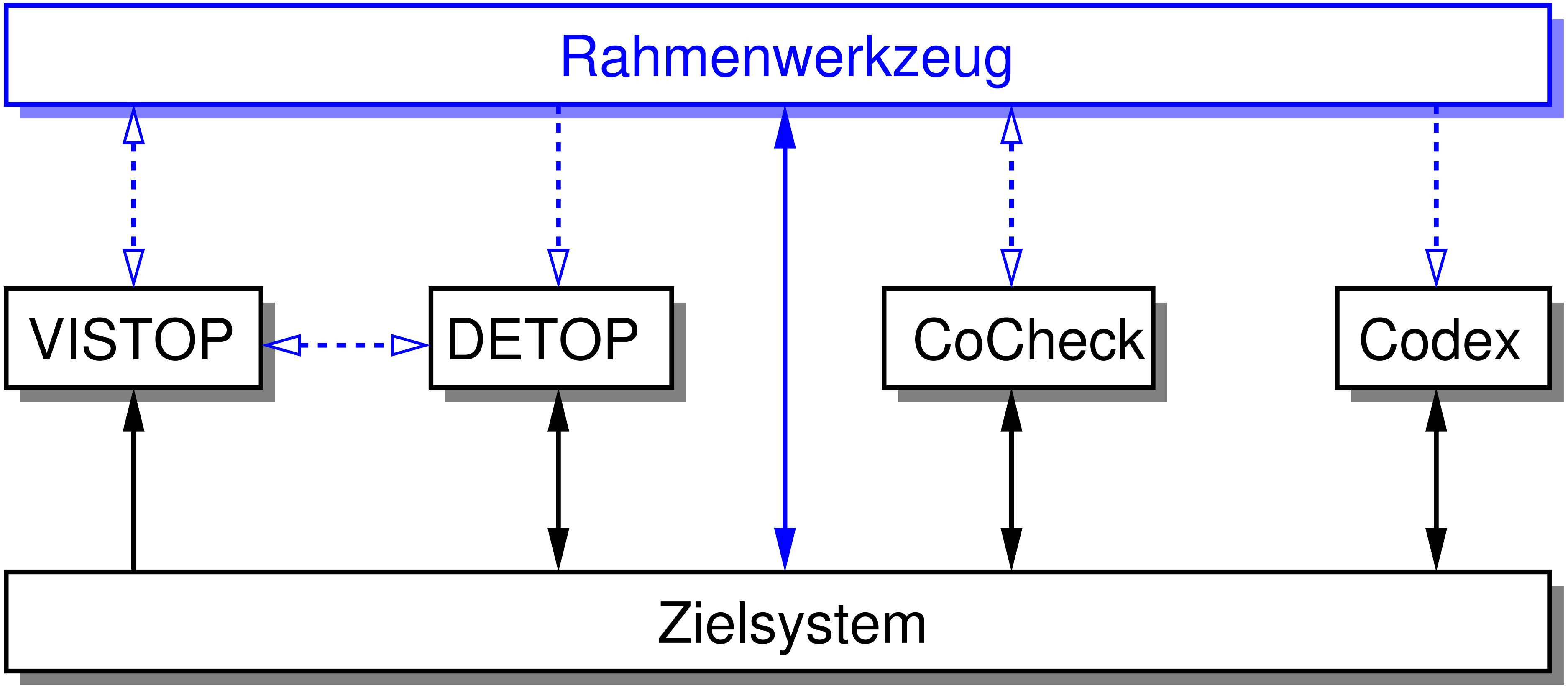
2. Motivation: Eine Werkzeugumgebung



Aufbau der Werkzeugumgebung



Aufbau der Werkzeugumgebung



3. Datenintegration

Problem: Zugriff auf gemeinsames Zielsystem

- Exklusive Betriebssystem-Schnittstellen
- Konflikte bei Modifikationen
  - Instrumentierung, Steuerung

Lösung:

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

3. Datenintegration

Ziele von OMIS

- Entkopplung von Werkzeug und Monitorsystem
- Gleichzeitige Nutzung durch mehrere Werkzeuge

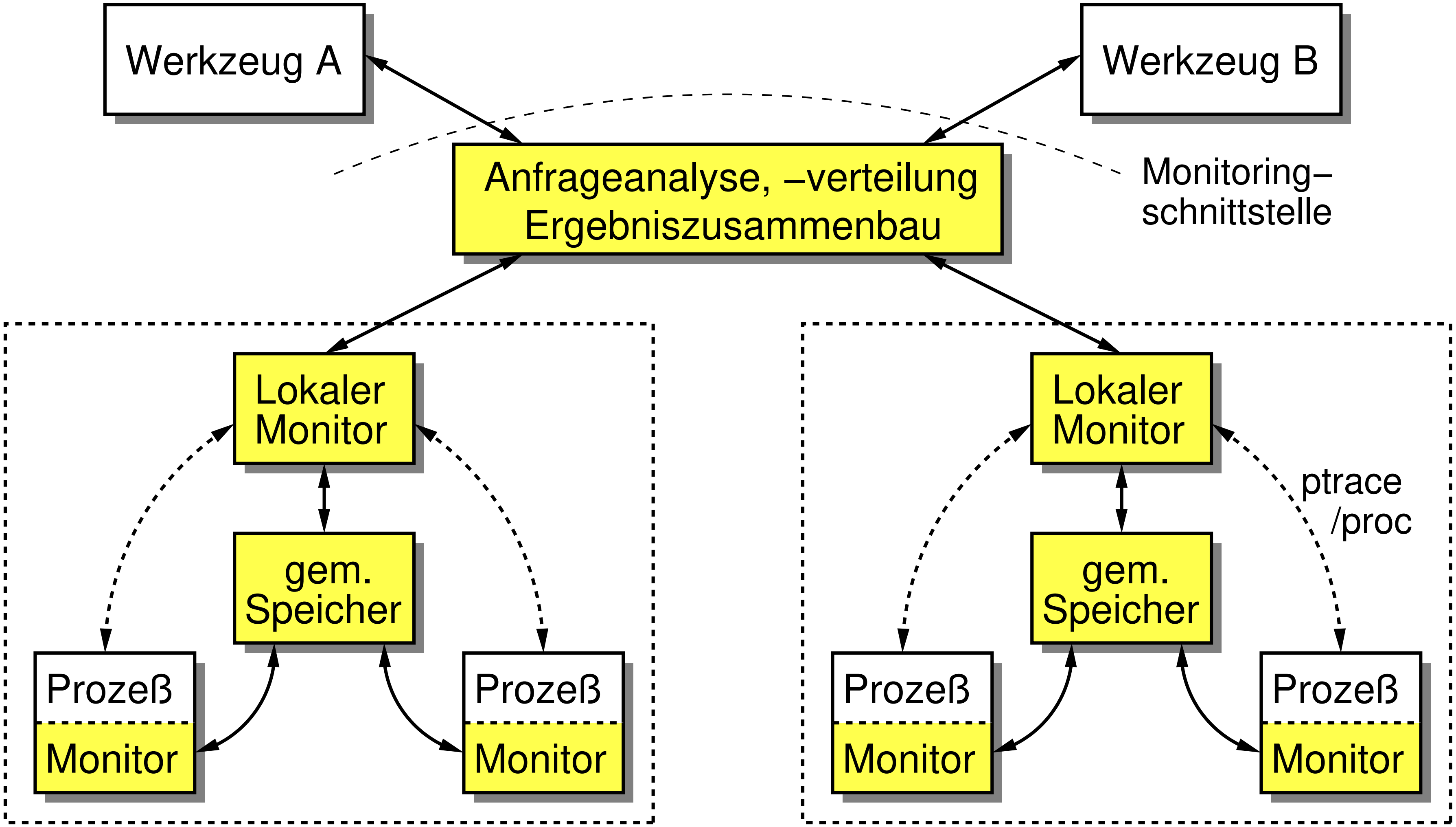
Konzepte

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

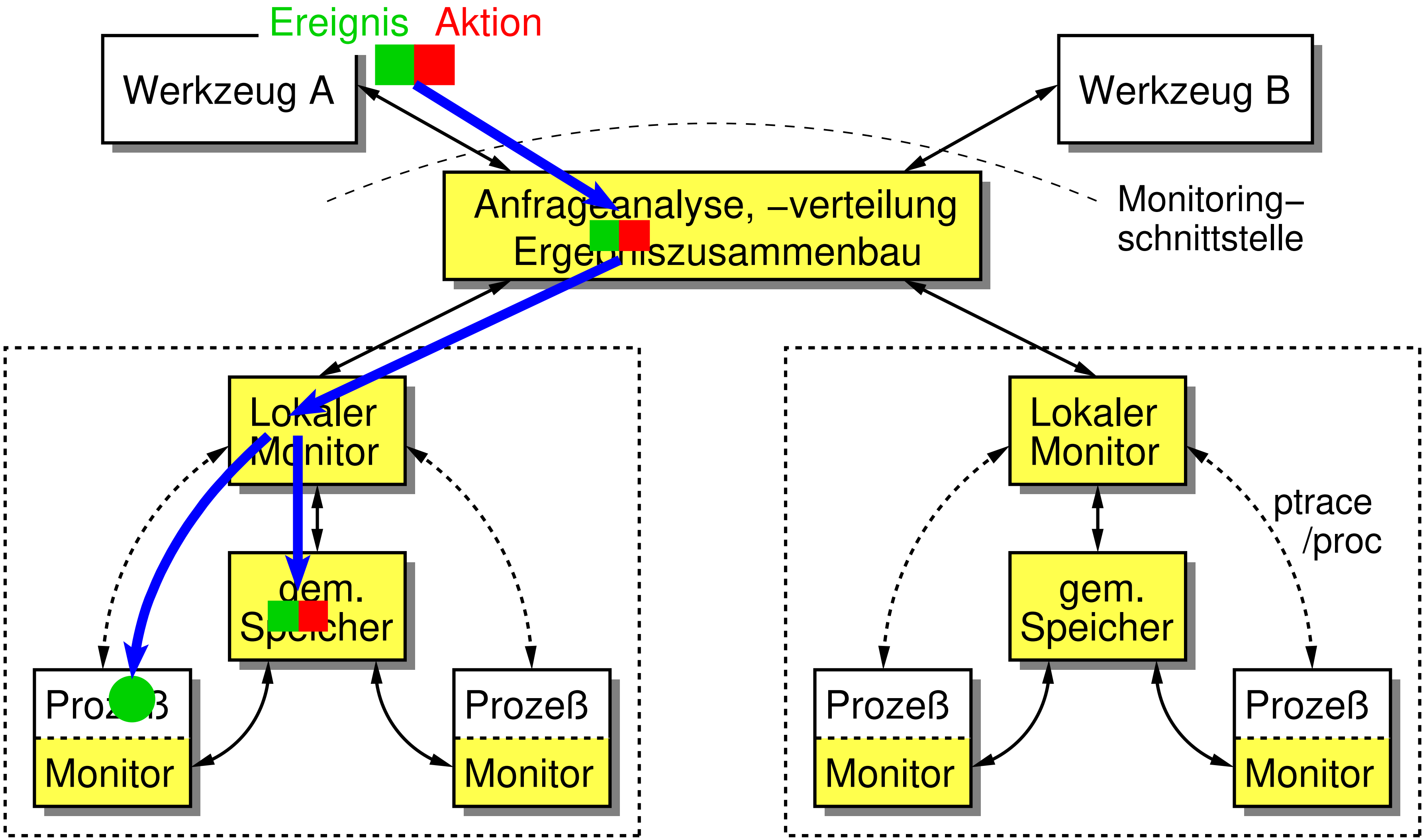
```
thread_started_lib_call([p_1],"pvm_send") :
    thread_stop([$node])
    thread_get_backtrace([$thread],0)
```

- Objektmengen, Ortstransparenz
- Erweiterbarkeit

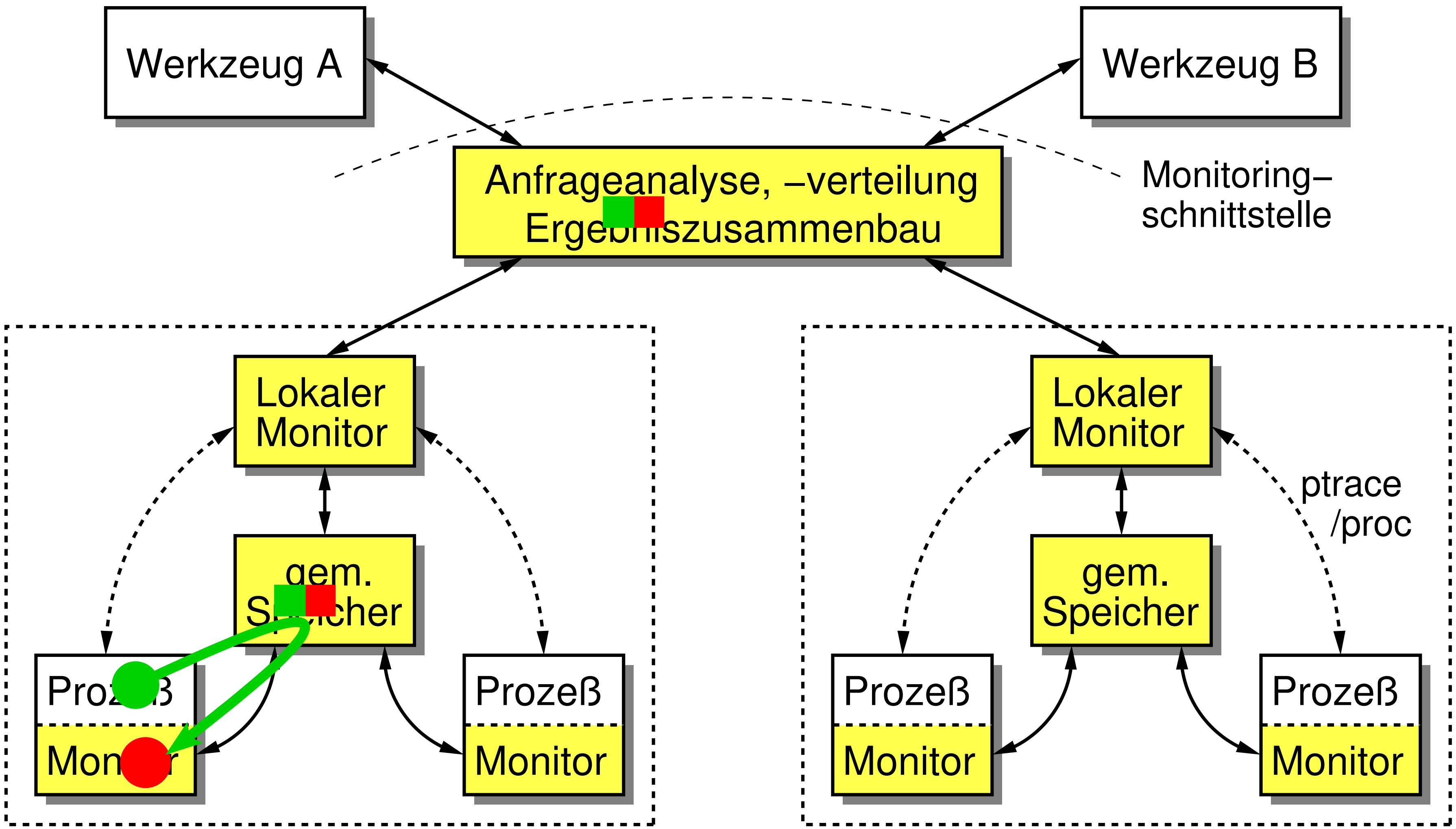
Aufbau des OCM (*OMIS Compliant Monitoring system*)



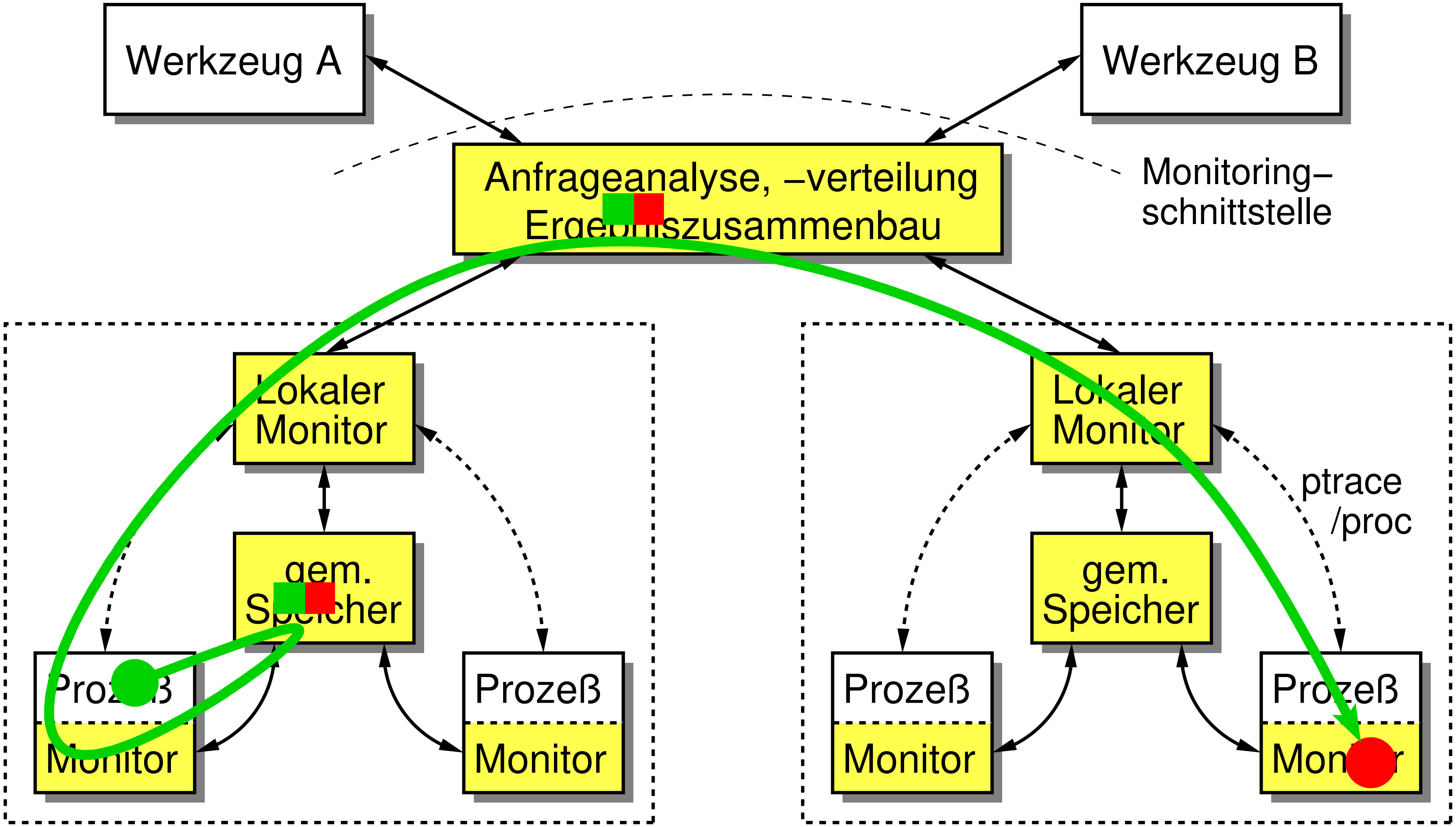
Anfrageabarbeitung im OCM



Anfrageabarbeitung im OCM



Anfrageabarbeitung im OCM



3. Datenintegration

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

4. Verhaltensintegration

Ziel: Kooperation zwischen den Komponenten

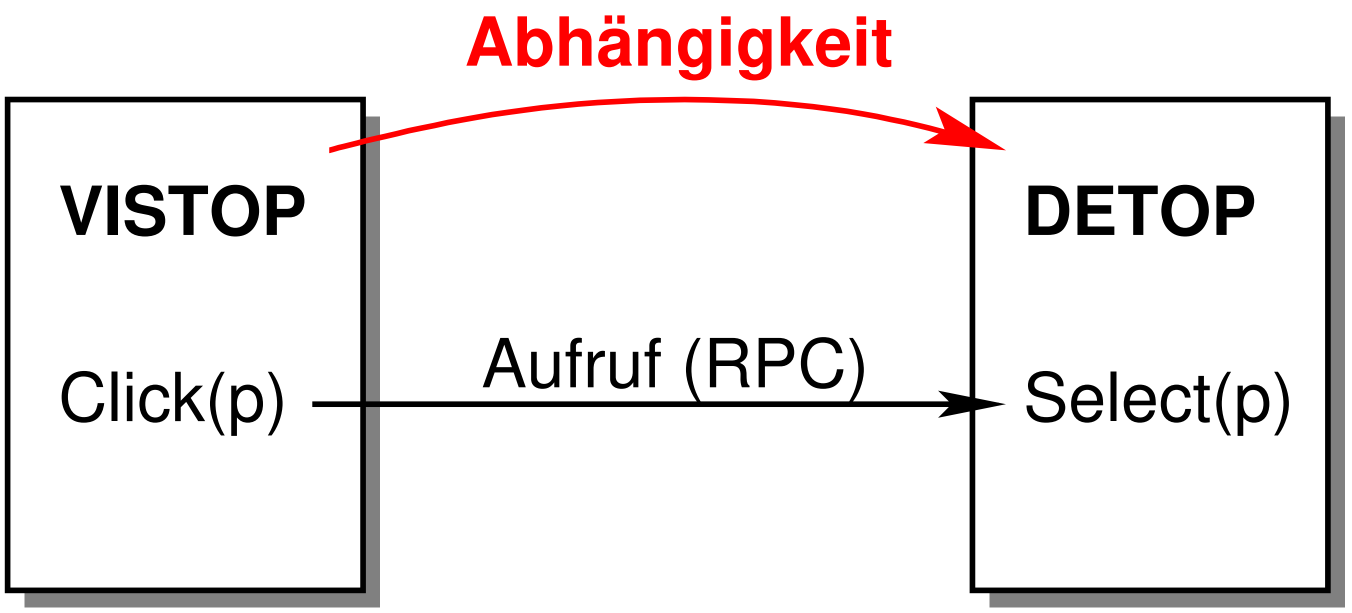
- Z.B. synchronisierte Auswahl, konsistente Darstellung
- Im weiteren Sinne: Steuerung durch Rahmenprogramm

Allgemeine Fragestellung:

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

Einfachster Ansatz: Feste Verdrahtung

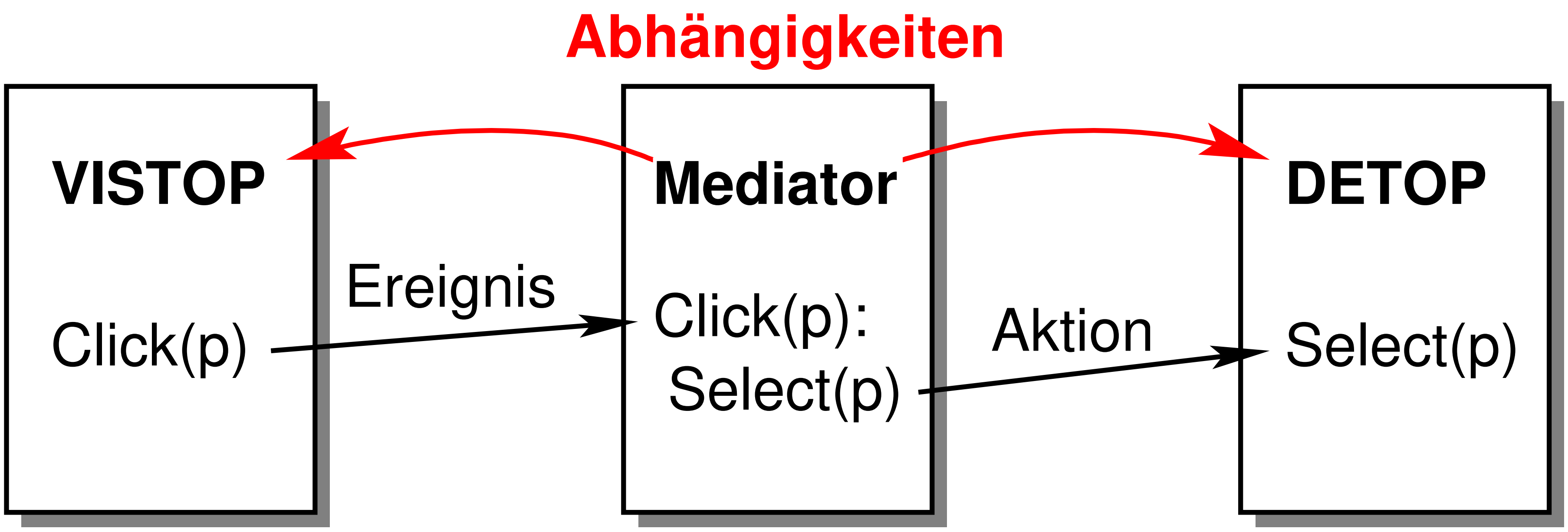
- Direkter Aufruf notwendiger Aktionen in anderer Komponente
- Z.B. mittels RPC, RMI, CORBA, PCTE



4. Verhaltensintegration

Mediatoren / Interaktoren

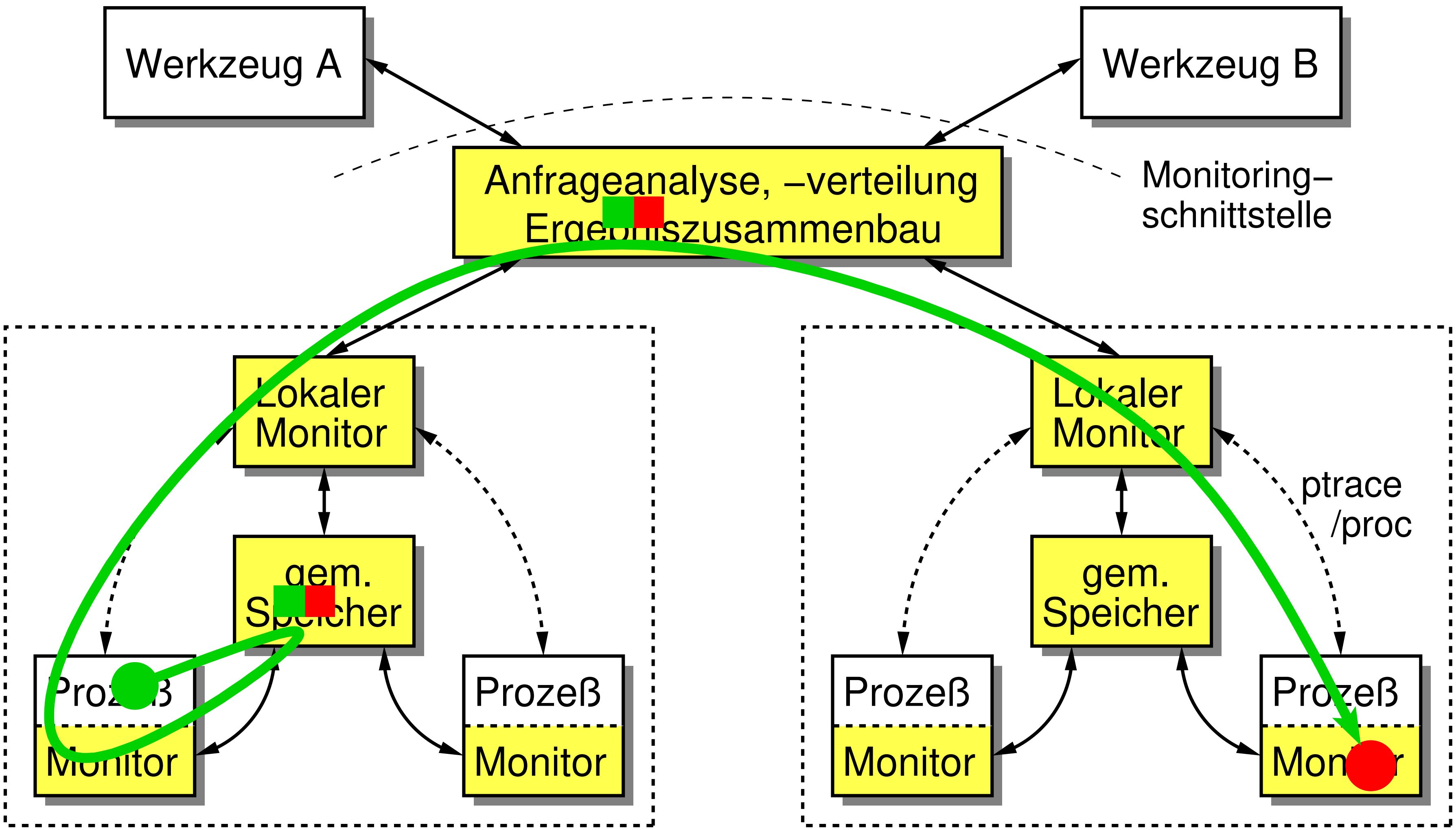
- Mediator überwacht Ereignis, ruft zugeordnete Aktion auf
- Z.B. ToolBus, Manifold, CHAIMS, TIL



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



4.1. Verhaltensintegration durch den OCM

Spezielle Eigenschaften des OCM:

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

$\}$ ohne spezielle Vorbereitung dieser Prozesse

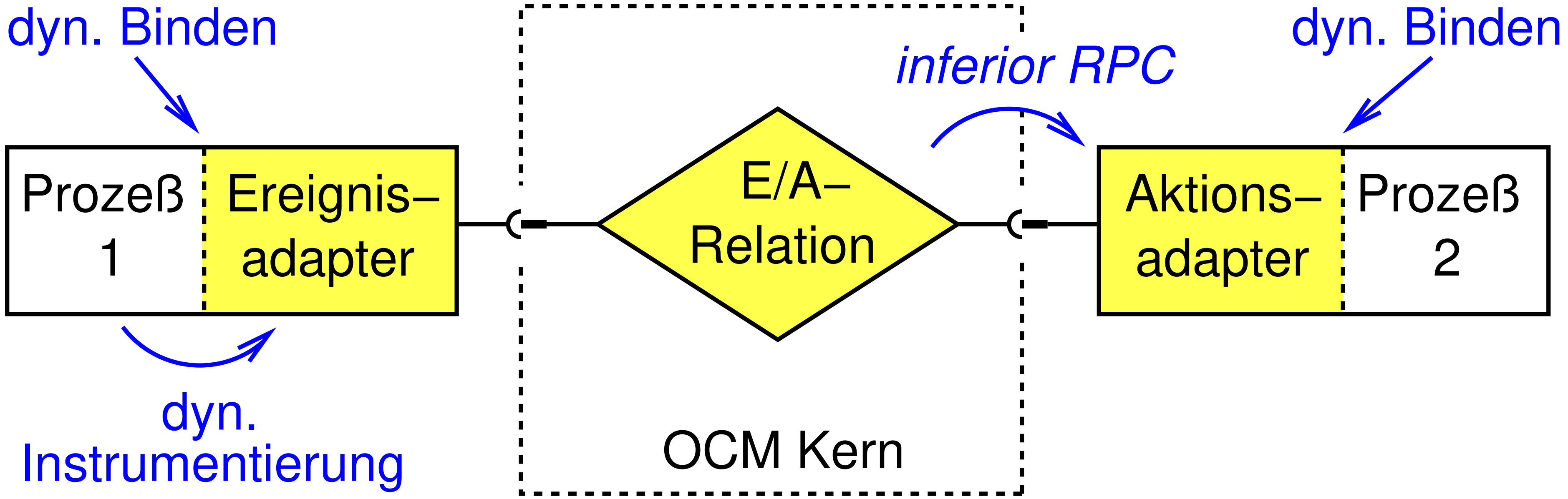
- Menge der Ereignisse und Aktionen zur Laufzeit erweiterbar

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

4.1. Verhaltensintegration durch den OCM

Zur Realisierung:

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



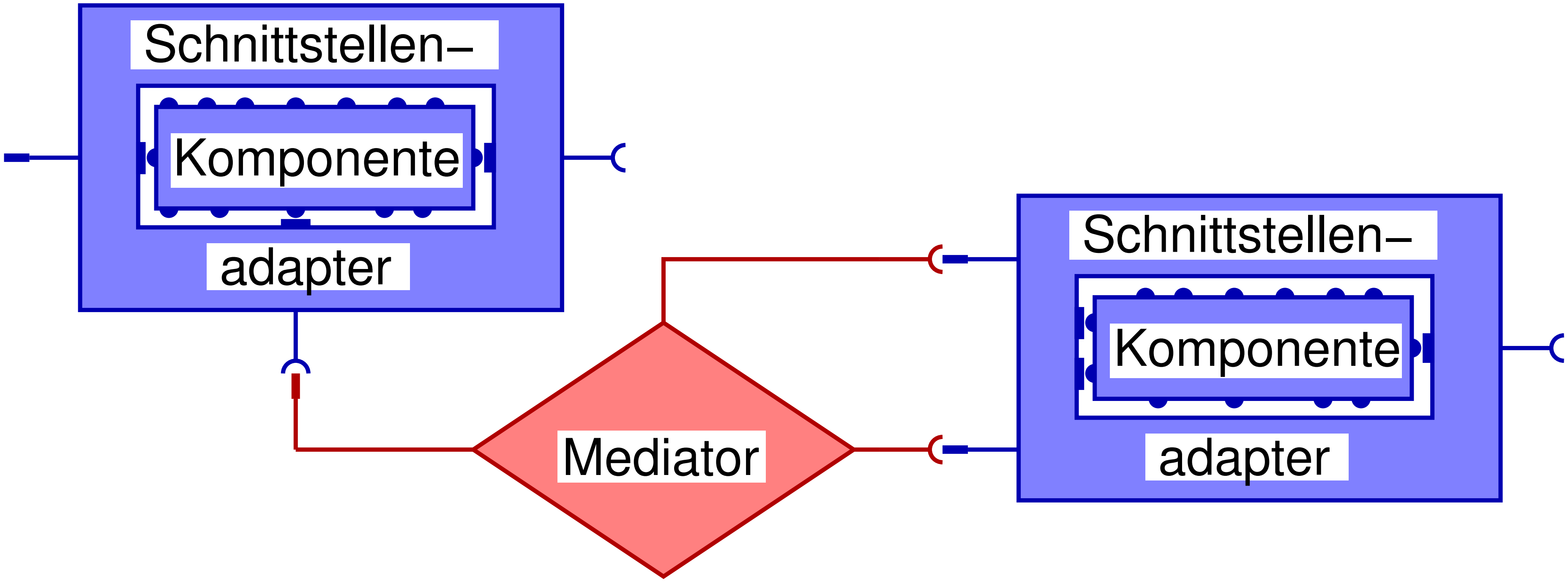
5. Resultat

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

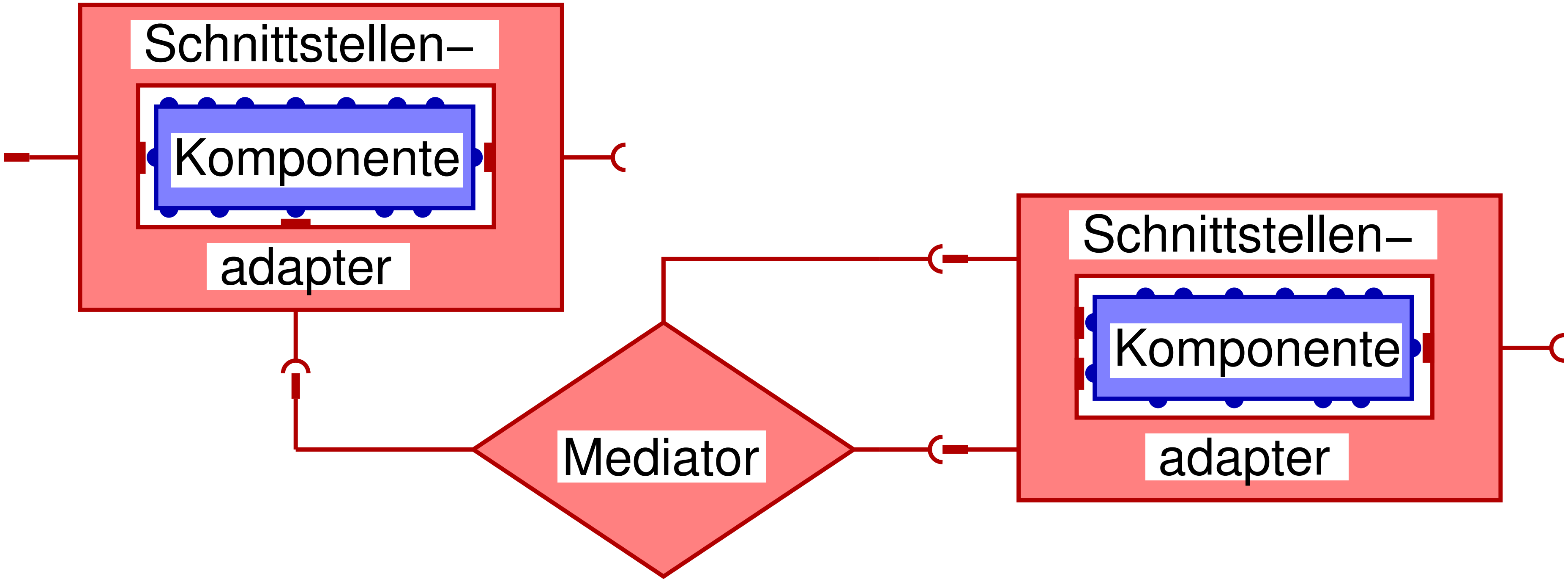
Traditionelles Vorgehen

Komponentenentwickler
Integrierer



Dynamisches Anbinden von Schnittstellen

Komponentenentwickler
Integrierer



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!

5.1. Praktische Erfahrungen

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

6. Fazit und Ausblick

Verhaltensintegration

- Mediator-Ansatz
- Dynamisches Einbinden von Schnittstellen(adaptern)

Weitere Arbeiten

- Übertragung der Konzepte in andere Anwendungsbereiche
- Integration mit Koordinationssprache
- Unterstützung durch / Nutzung in Betriebssystemen
- Systematische Trennung von Middleware und Monitordiensten