Interoperable Werkzeuge zur Laufzeitanalyse verteilter Systeme:

Eine Fallstudie

Roland Wismüller

Lehrstuhl für Rechnertechnik und Rechnerorganisation
LRR-TUM
Technische Universität München
e-mail: wismuell@in.tum.de
www: www.in.tum.de/∼wismuell

Agenda

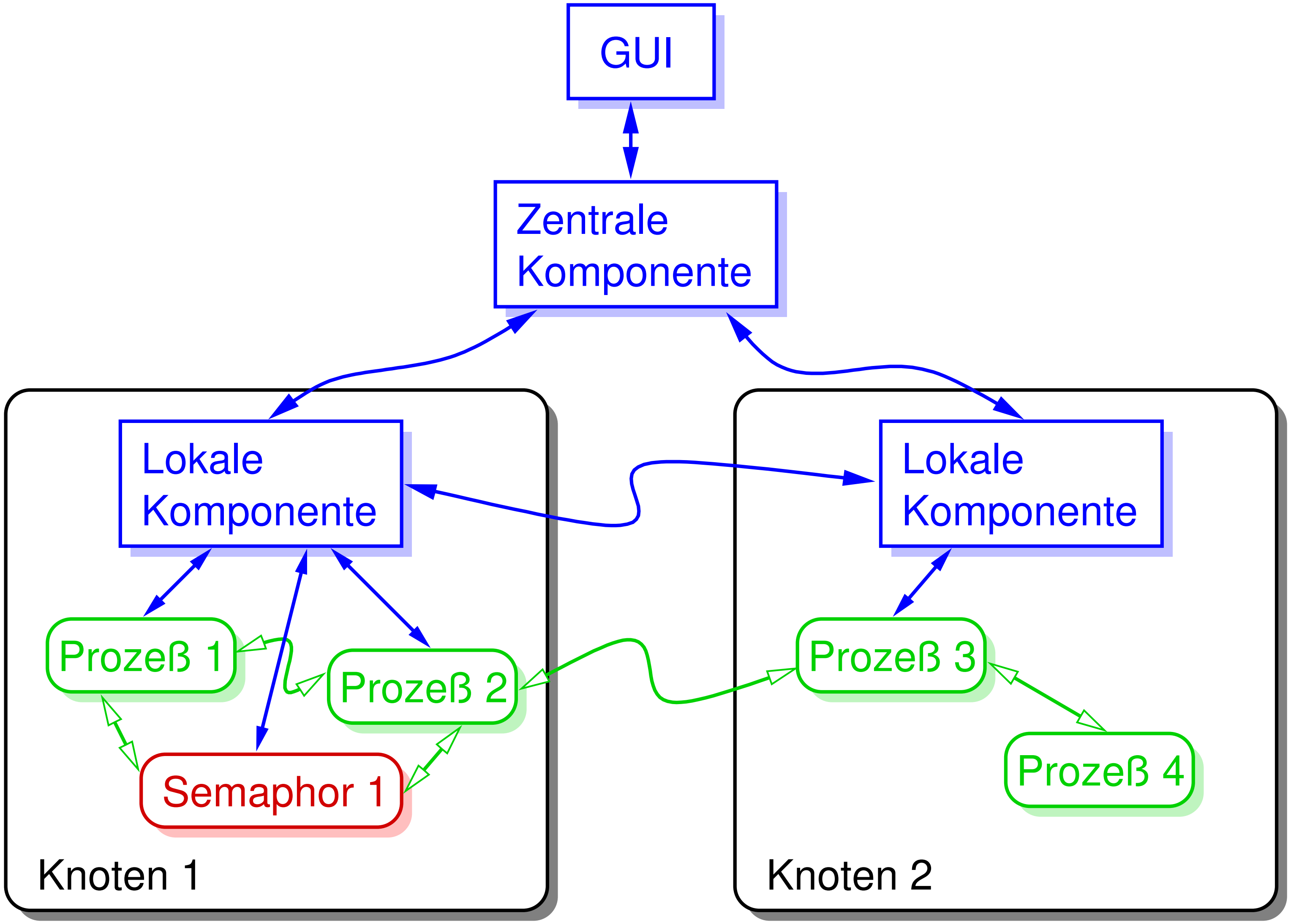
- Einführung
  - Werkzeuge zur Laufzeitanalyse, Interoperabilität
- Ein konkretes Szenario:
  - Debugger mit wahlfreiem Zugriff auf Programmzustände
  - dazu benötigte Einzelwerkzeuge
- Interaktionen der Werkzeuge
  - Anforderungen an eine unterstützende Infrastruktur
  - existierende Infrastrukturen
- Umsetzung der Anforderungen im Monitorsystem OCM
- Zusammenfassung

Einführung

Werkzeuge zur Laufzeitanalyse

- *Beobachten*, *Analysieren* und *Steuern* des Ablaufs eines verteilten Hard- und Softwaresystems (Zielsystem)
- Hier betrachtet: *on-line* Werkzeuge
- Typisch:
  - ereignisorientierte Arbeitsweise
  - Aufbau:
    - verteilte Überwachungs- und Steuerkomponenten
    - zentrale Analysekomponente (z.T. auch verteilt)
    - zentrales Benutzerinterface (interaktive Werkzeuge)

Typischer Aufbau eines Werkzeugs



Interoperabilität

- **Schillernder Begriff:**
  - „mehrsprachige“ Software (z.B. C und Fortran)
  - Kommunikation verteilter Programme (z.B. CORBA)
  - Datenbankmodelle (relational, OO)
  - ...
- **Gemeinsamkeit:**

  Interoperabilität ist die Fähigkeit von unabhängigen (nicht speziell zu diesem Zweck entworfenen) Software-Komponenten, auf syntaktischer und semantischer Ebene zu kooperieren.

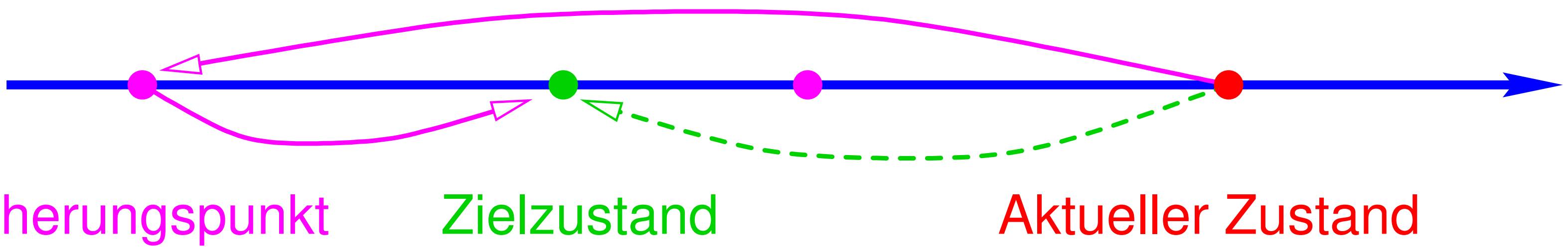
Einführung ...

Interoperabilität bei Laufzeitwerkzeugen

- Mehrere Werkzeuge kooperieren bei der Überwachung / Steuerung desselben Systems
- **Motivation:**
  - für Benutzer:
    - gleichzeitige Nutzung von Werkzeugen für unterschiedliche Aufgaben
    - Zusatznutzen durch kombinierten Einsatz
  - für Entwickler:
    - bessere Modularisierung
    - Ausfaktorisieren von Zielsystemabhängigkeiten

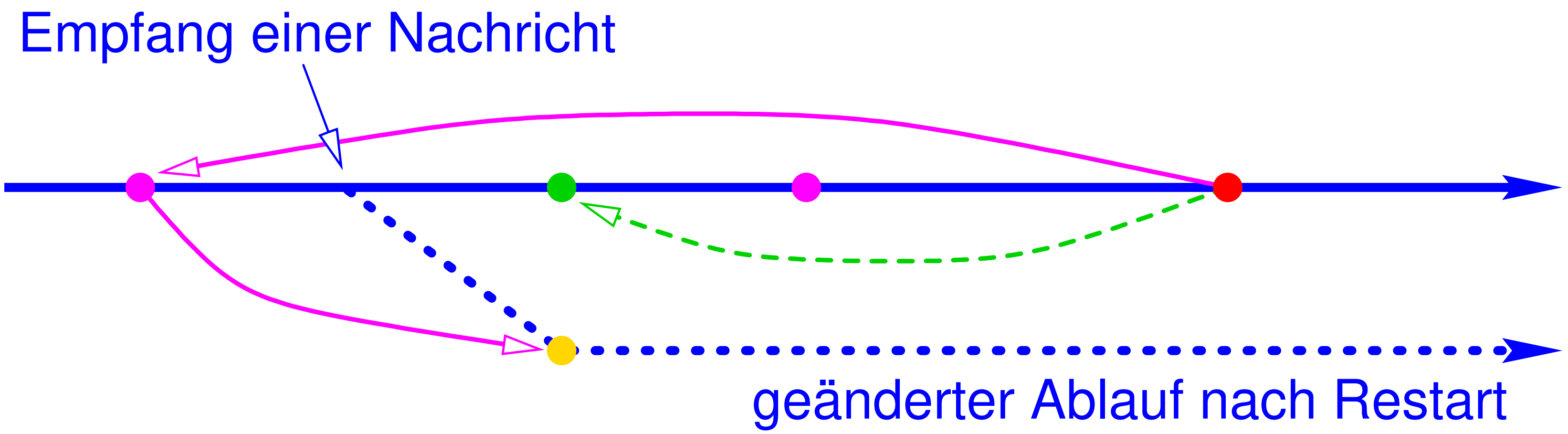
Ein konkretes Szenario

- **Ziel:**
  - effiziente Fehlersuche in lang laufenden Programmen
  - Debugger mit wahlfreiem Zugriff auf (frühere) Programmzustände
- **Realisierungsidee:**
  - periodisches Sichern des Programmzustands
  - Zurücksetzen auf früheren Sicherungspunkt und automatisches „Anfahren“ des gewünschten Zustands



Ein konkretes Szenario ...

- **Realisierung:** durch Kombination existierender Werkzeuge
  - paralleler Debugger
  - Sicherungspunkt-Generator
  - Ablaufvisualisierer zur Definition des Zielzustands
  - Werkzeug zur deterministischen Ablaufkontrolle



Der Debugger DETOP

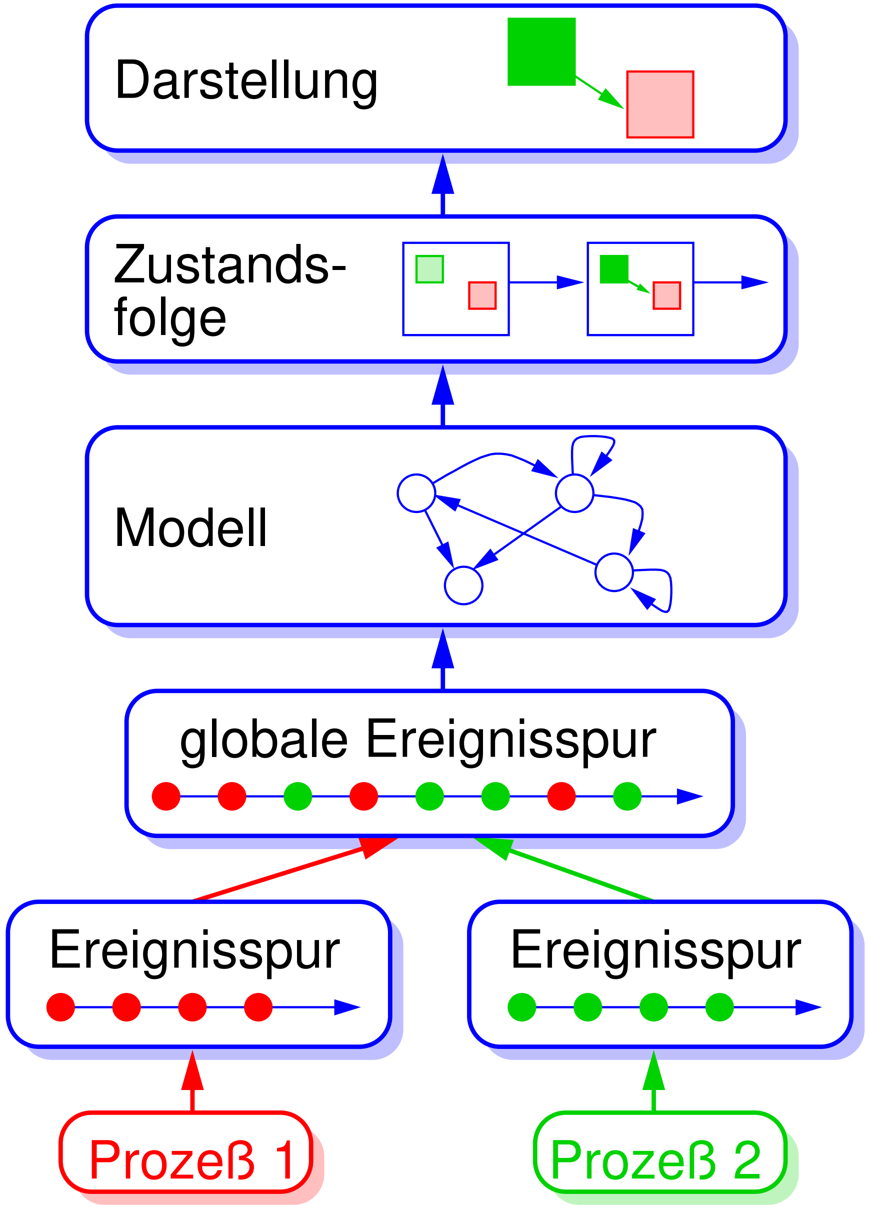
- **Ziel:**
  - Debugging von Prozeßmengen
- **Realisierung:**
  - verteilter Aufbau gemäß vorgestelltem Schema
  - lokale Komponenten (Monitorsystem) nutzen BS-Schnittstellen
    - zur Prozeßüberwachung (z.B. Signale)
    - zur Prozeßsteuerung (z.B. Anhalten)
    - zum Lesen / Schreiben im Adreßraum überwachter Prozesse
  - keine Kenntnis des Programmiermodells (PVM)
    - Ausnahme: Task-Identifikatoren

Sicherungspunkterstellung mit CoCheck

- **Ziel:**
  - konsistente Sicherungspunkte kommunizierender Prozesse
- **Basis:**
  - Sicherungspunktgenerator für unabhängige Prozesse
  - Protokoll zum Leeren der Kommunikationsverbindungen
    - speichert alle Nachrichten im Prozeßadreßraum
- **Realisierung:**
  - Abfangen von PVM-Aufrufen
    - Empfangen der gespeicherten Nachrichten
    - Umsetzen der Task-Identifikatoren

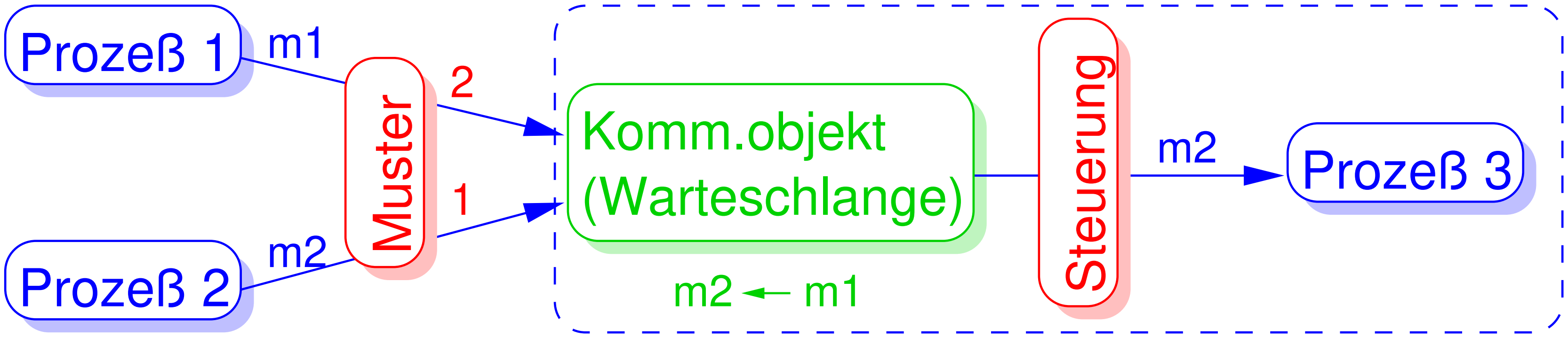
Der Ablaufvisualisierer VISTOP

- **Ziel:**
  - zustandsorientierte Visualisierung des Programmablaufs
- **Realisierung:**
  - Aufzeichnung von PVM-Aufrufen
  - Modellierung des PVM-Zustands
  - Darstellung eines ausgewählten Zustands



Ablaufsteuerung durch Codex

- **Ziel:**
  - deterministische Ablaufsteuerung durch Vorgabe von Zugriffsmustern auf Kommunikationsobjekten
- **Realisierung:**
  - Abfangen von PVM-Empfangsaufrufen
    - Prüfen, welche Nachricht laut Muster empfangen werden müsste
    - Empfangen dieser Nachricht durch PVM-Aufruf



Interaktionen der Werkzeuge

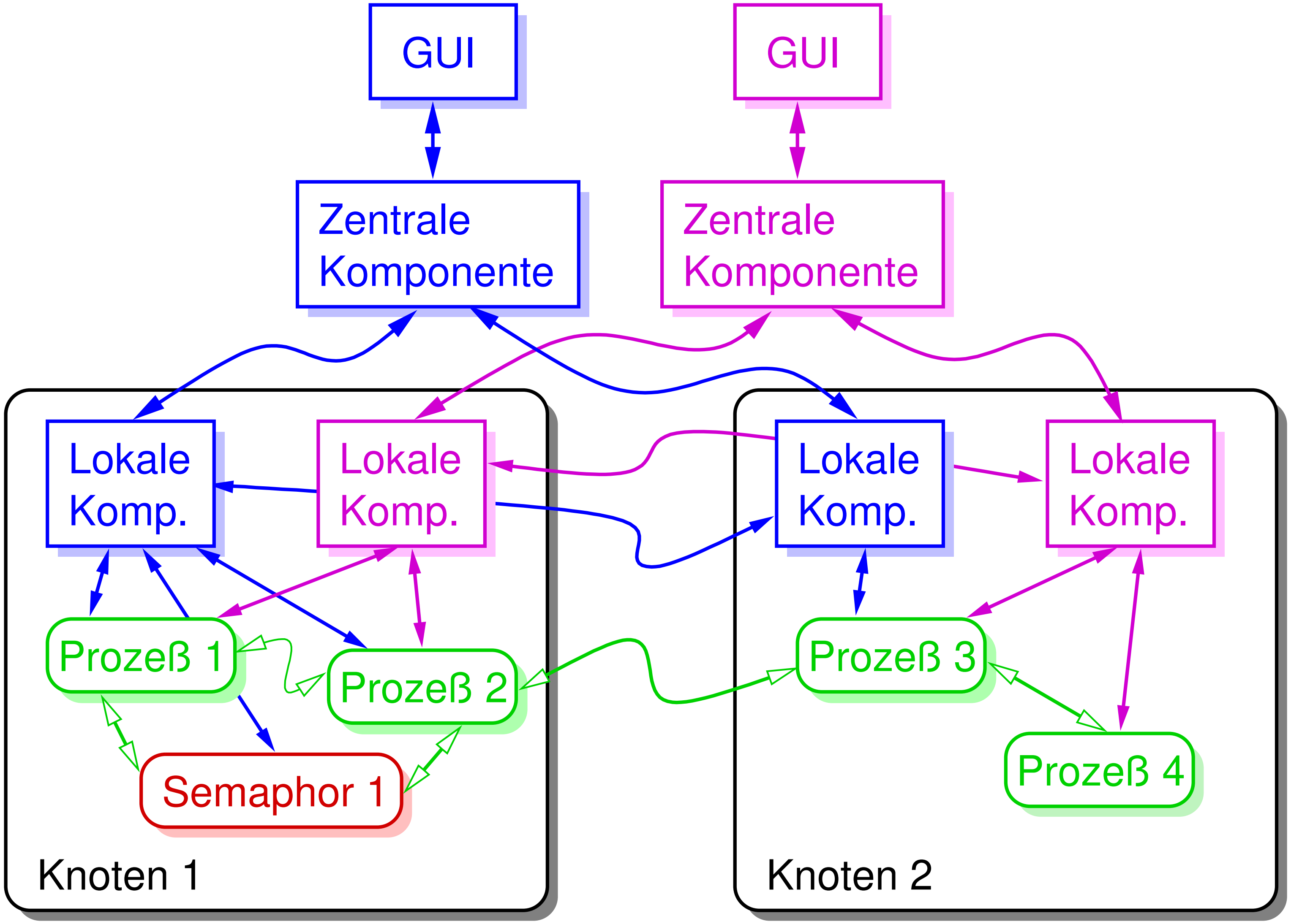
- **Im folgenden:**
  - vier Beispiele für Wechselwirkungen zw. Werkzeugen
- **Daraus:**
  - Klassifikation der Arten von Wechselwirkungen
  - Anforderungen an Infrastrukturen
- **Nicht explizit betrachtet:**
  - Fragen der Benutzerschnittstelle

Interaktionen der Werkzeuge ...

- **Beispiel 1**:
  - CoCheck, VISTOP und Codex überwachen PVM-Aufrufe in denselben Prozessen
  - CoCheck und Codex modifizieren Parameter der Aufrufe

Interaktionen der Werkzeuge ...

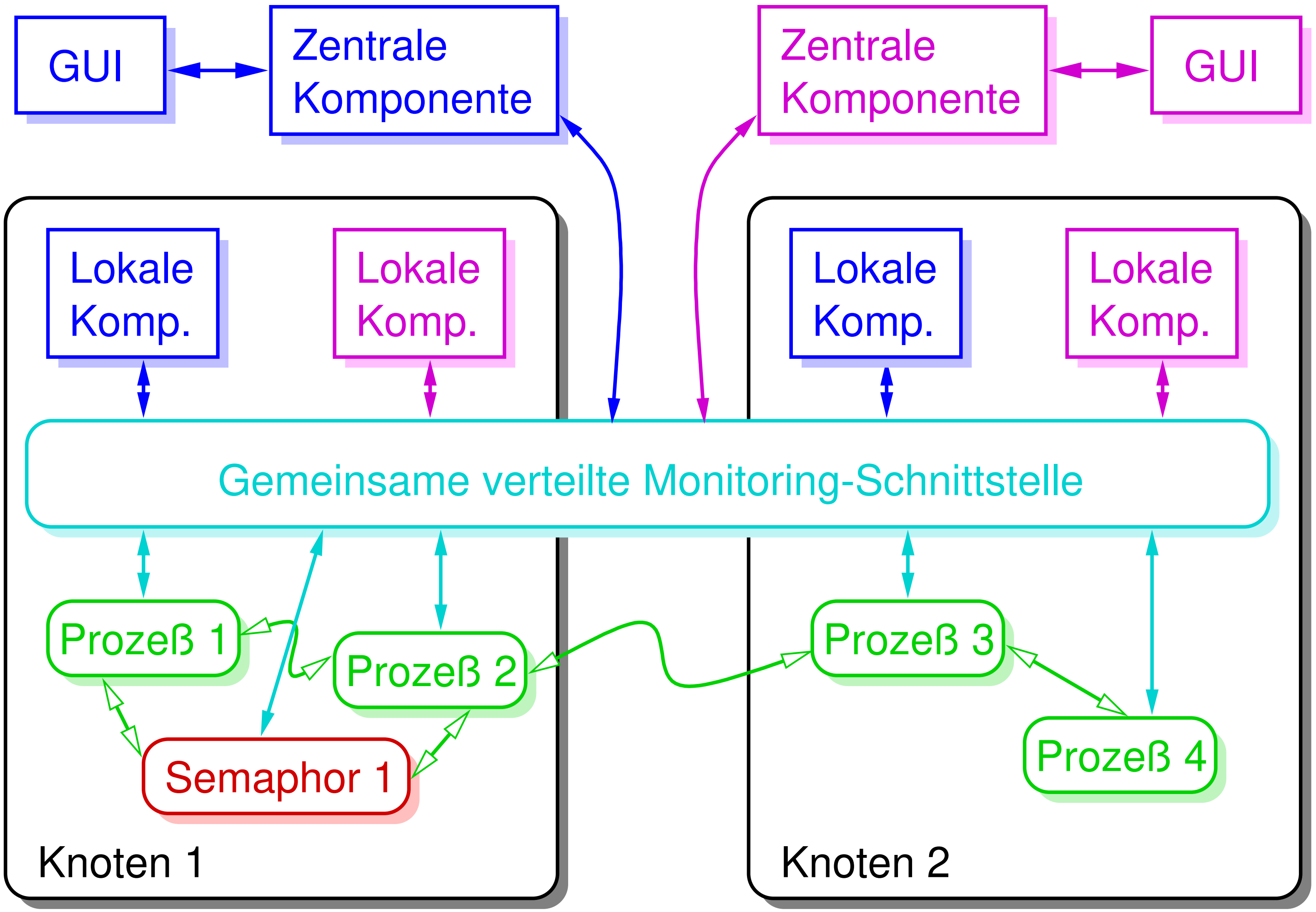
Mehrere Werkzeuge am selben Zielsystem



Interaktionen der Werkzeuge ...

- *Analogie zu Betriebssystemen:*
  - Zielsystem $\longleftrightarrow$ Festplatte
  - freiwillige Koordination funktioniert nicht!
- **Anforderung 1:**
  - gemeinsame, offene Schnittstelle
    zur Strukturierung und Koordination der Zugriffe
    auf das Zielsystem

Gemeinsame Schnittstelle zum Zielsystem



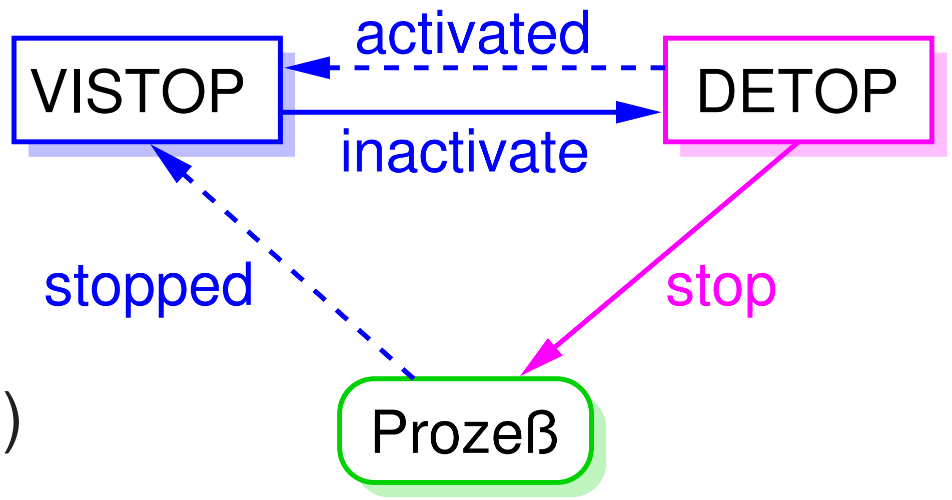
Interaktionen der Werkzeuge ...

- **Beispiel 2:**
  - Werkzeuge sollten konsistente Sicht des Systems zeigen
  - Problem:
    - DETOP zeigt immer aktuellen Zustand
    - VISTOP kann beliebigen vergangenen Zustand zeigen
  - Lösungsidee:
    - wenn DETOP aktiv: VISTOP zeigt letzten Zustand an
    - beim Zurückblättern in VISTOP: DETOP inaktivieren
  - wegen Pufferung:
    - beim Anhalten eines Prozesses: VISTOP muß Ereignisspur-Puffer lesen

Interaktionen der Werkzeuge ...

- **Anforderung 2:**
  - Melden von Ereignissen und Auslösen von Aktionen in anderen Werkzeugen
  - direkte Interaktionen

- **Anforderung 3:**
  - Überwachung von Zustandsänderungen an Objekten (ausgelöst durch andere Werkzeuge)
  - indirekte Interaktionen



Interaktionen der Werkzeuge ...

- **Beispiel 3:**
  - Restart durch CoCheck führt zu neuen Task-Identifikatoren
  - andere Werkzeuge sollen weiterhin die alten Identifikatoren sehen
  - Ermittlung der Identifikatoren über Monitor-Schnittstelle
- **Anforderung 4:**
  - Mechanismus, um Zugriffe anderer Werkzeuge abzufangen
  - Modifikation der Anfragen bzw. Ergebnisse

Interaktionen der Werkzeuge ...

- **Beispiel 4:**
  - VISTOP modelliert Nachrichtenwarteschlangen
  - Codex verletzt FIFO-Semantik der Warteschlangen
    - Empfangsreihenfolge $\neq$ Eintragsreihenfolge
  - Verletzung einer Grundannahme von VISTOP
- Erfordert wesentliche Anpassungen in den Werkzeugen
- Grenze der Unterstützung durch Infrastruktur!

Anforderungen an eine Infrastruktur

- *„Mehrbenutzerfähige“ Schnittstelle zum Monitoring*
  - Koordination von Zugriffen auf das Zielsystem
- *Überwachung von Ereignissen und Aufruf von Aktionen in anderen Werkzeugen*
  - direkte Interaktionen
- *Benachrichtigung bei Änderung von Objektzuständen*
  - indirekte Interaktionen
- *Abfangen und Modifizieren von Objektzugriffen*
  - Aufrechterhaltung von Sichten

Existierende Infrastrukturen

	Corba	ToolTalk	PCTE	DPCL	DAMS	OCM	DBSe
Monitorschnittstelle	-	-	o	+	+	+	-
Mehrbenutzerfähig	+	+	+	+	+	+	+
Koordination	(+)	-	+	-	?	+	+
Direkte Interaktionen	+	+	o	-	-	-	-
Ereignisse	o	+	-	-	-	-	-
Aktionen	+	+	-	-	-	-	-
Benachrichtigung	-	-	o	-	+	o	-
Sichten	-	-	-	-	-	-	o

Umsetzung der Anforderungen im OCM

Das verteilte Monitorsystem OCM

- Grundlagen:
  - Objektmodell des überwachten Systems
  - Ereignis-/Aktions-Schema
    - Dienste: Ereigniserkennung, Zugriff auf Objekte
    - Kombination in bedingten Anfragen, z.B.:

```
thread_has_been_stopped([p_1]) :
    thread_get_backtrace([$thread], 0)
```

- Realisierung:
  - verteilte Server-Prozesse, hierarchische Struktur
  - Ereigniserkennung und Aktionen auch in den überwachten Prozessen

Umsetzung der Anforderungen im OCM ...

„Mehrbenutzerfähige“ Schnittstelle zum Monitoring

- Server-Aufbau des OCM
- Eigene „Umgebung“ für jedes Werkzeug
  - z.B. Definition des überwachten Systems
- Synchronisation:
  - Einzelaktionen auf einem Objekt ununterbrechbar
  - atomare Multicast-Kommunikation
  - globale Sperren auf Anforderung

Umsetzung der Anforderungen im OCM ...

Ereignisse und Aktionen in anderen Werkzeugen

- OCM unterstützt Ereignisse/Aktionen im Kontext überwachter Prozesse
- Registrierung erfolgt erst zur Laufzeit
- Notwendige Erweiterung:
  - Werkzeuge in das Objektmodell aufnehmen
  - Grenze zwischen überwachtem und überwachendem System aufgelöst!

Umsetzung der Anforderungen im OCM ...

Benachrichtigung bei Änderung von Objektzuständen

- Realisierbar durch Ereignisdienste
- Z.Zt. nur für Spezialfälle, z.B. `thread_has_been_stopped`

Abfangen und Modifizieren von Objektzugriffen

- Im OCM derzeit nicht möglich

Gemeinsamer Kern: Erkennung von Objektzugriffen

- Überwachung des Aufrufs von Diensten nicht ausreichend
  - Dienste greifen auf mehrere Objektkomponenten zu

Umsetzung der Anforderungen im OCM ...

Erkennung von Objektzugriffen

- Ziel: *universeller* Dienst
- Lösungskonzept:
  - jede Zustandskomponente eines Objekts hat eindeutigen Bezeichner (Selektor)
  - Zugriff erfolgt nur über diese Selektoren; Methoden:
    - write(object, selector, value)
    - read(object, selector) $\mapsto$ value
  - keine Überlappung der Zustandskomponenten
- Definition der relevanten Zugriffe durch Muster über (Objekt $\times$ Selektor $\times$ Zugriffsart $\times$ Wert) möglich

Umsetzung der Anforderungen im OCM ...

Erkennung von Objektzugriffen

- Anforderungen:
  - Selektoren beschreiben Semantik der Komponenten
    - hierarchische Gliederung, z.B. `memory.1234`
  - Manipulation des Zielsystems ausschließlich durch Schreiben auf Zustandskomponenten
    - Anhalten eines Prozesses: `write(p123,stopped,True)`
- Probleme:
  - dynamische Erweiterbarkeit (Objekttypen, Komponenten)
  - implizite Änderung von Zustandskomponenten durch BS-Aufrufe
  - Effizienz der Implementierung?

Zusammenfassung

- Kombination von Werkzeugen ist ein guter Weg zur Realisierung höherer Funktionalität
- Interoperabilität ohne unterstützende Infrastruktur nicht realisierbar
  - mehrbenutzerfähige Monitoring-Schnittstelle
  - Ereignisse und Aktionen in anderen Werkzeugen
  - Benachrichtigung bei Änderung von Objektzuständen
  - Abfangen von Objektzugriffen
- OCM bietet Chancen zur Realisierung dieser Anforderungen
- Zukunft: Umsetzung der Konzepte, Bewertung