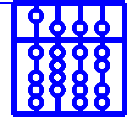
Effiziente parallele Programmierung auf vernetzten Arbeitsplatzrechnern

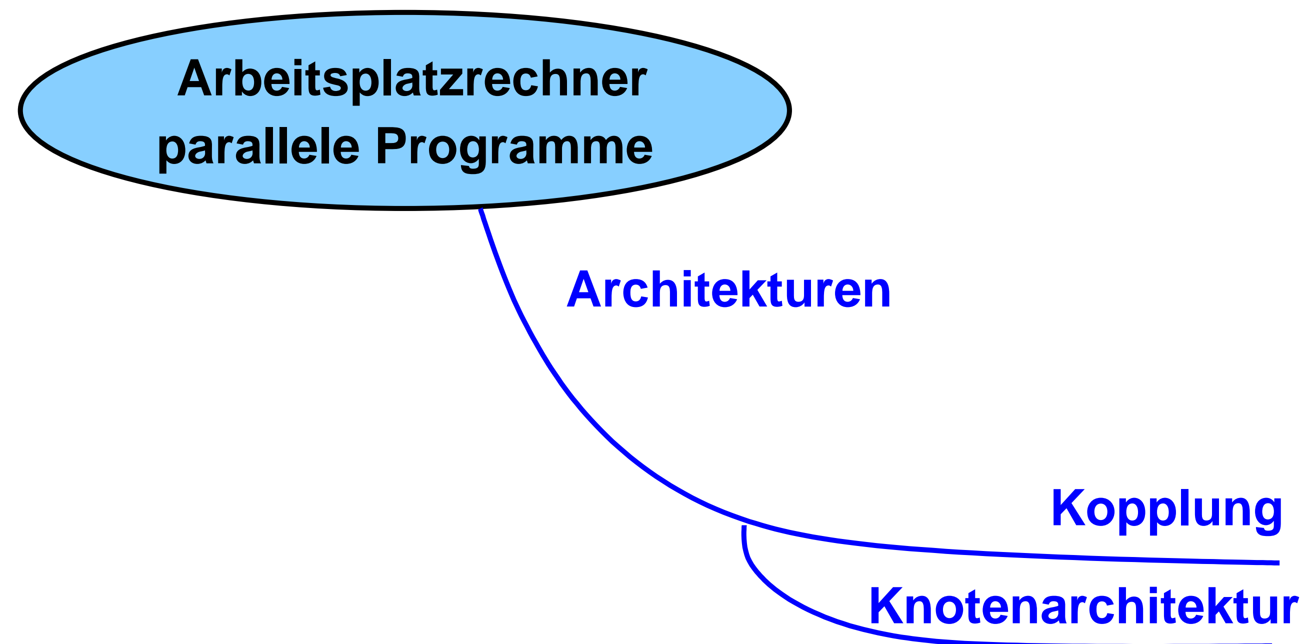
Thomas Ludwig

Lehrstuhl für Rechnertechnik und
Rechnerorganisation (LRR-TUM)
Institut für Informatik
Technische Universität München

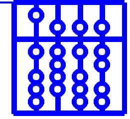
ludwig@in.tum.de
www.in.tum.de/∼ludwig

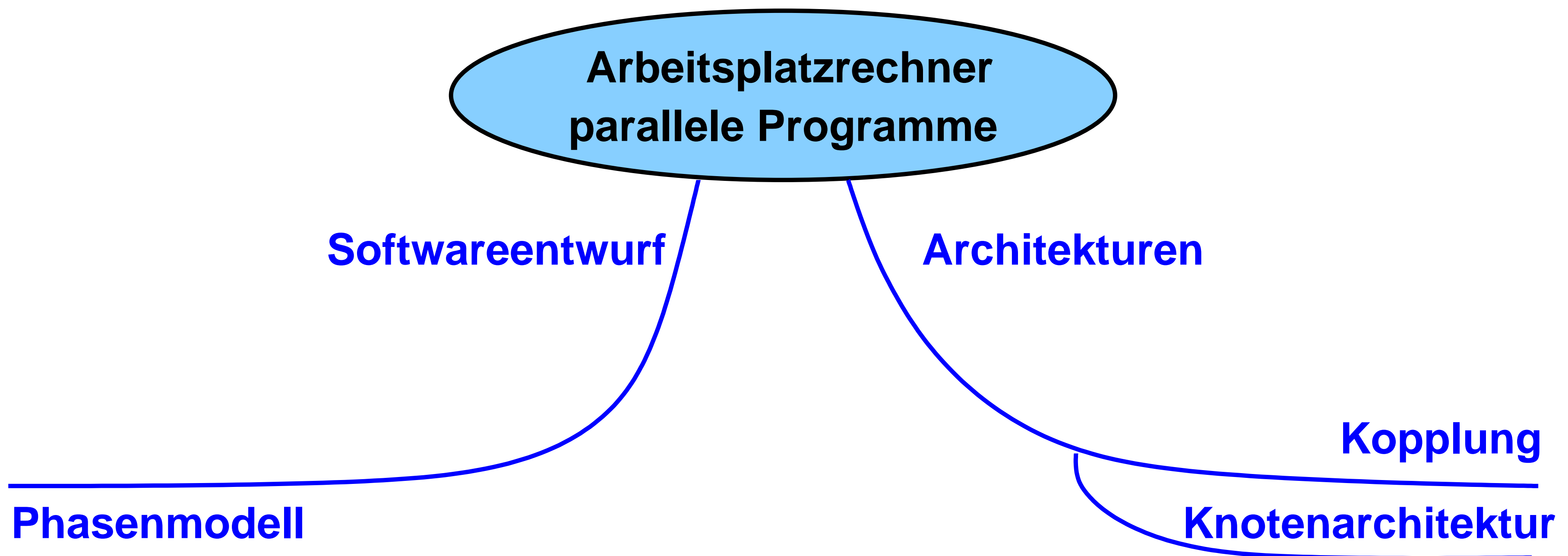
© Thomas Ludwig, TU München




Arbeitsplatzrechner
parallele Programme
Architekturen
Kopplung
Knotenarchitektur

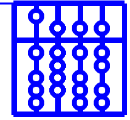
Parallele Programme und Arbeitsplatzrechner



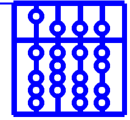


© Thomas Ludwig, TU München

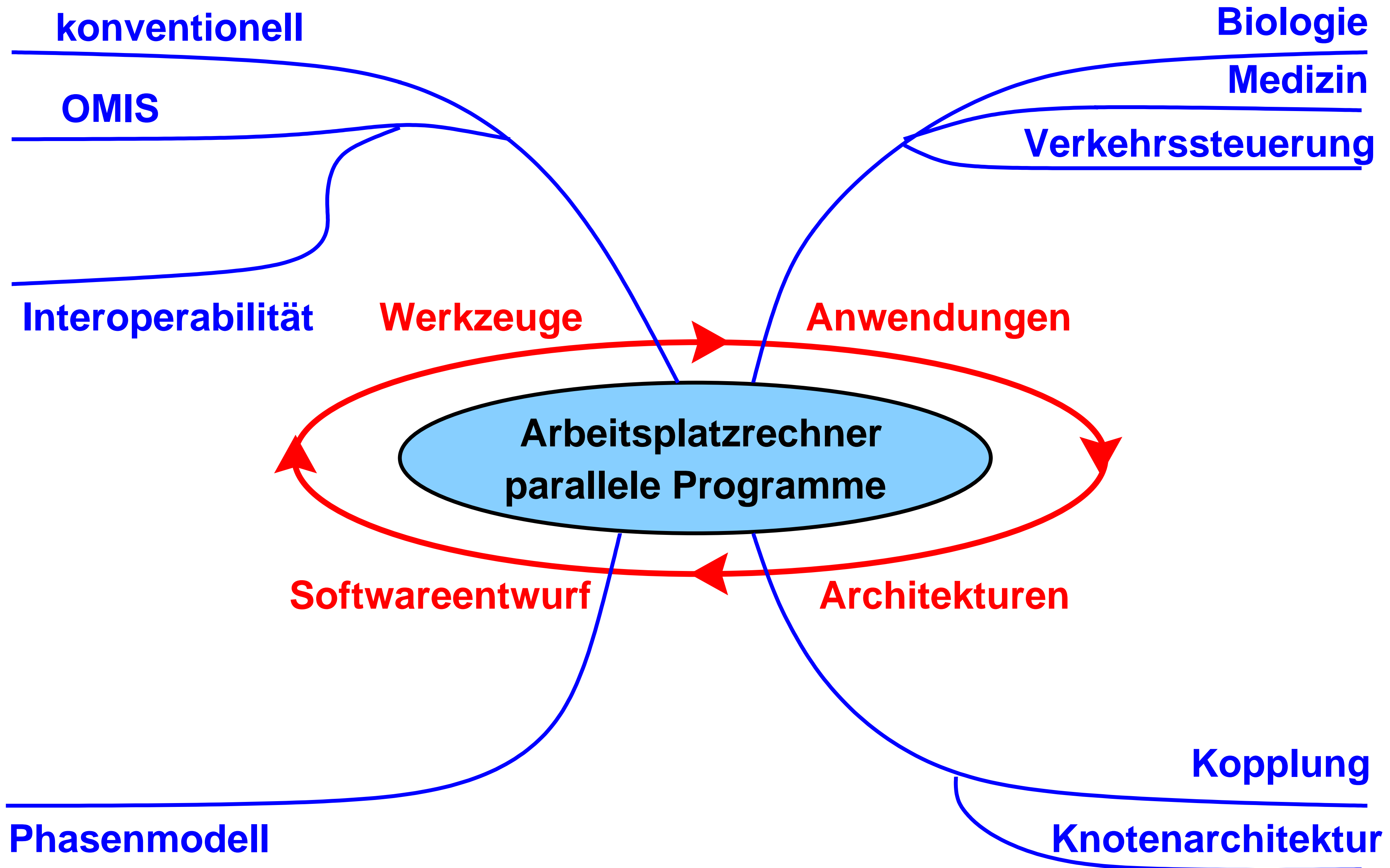
Parallele Programme und Arbeitsplatzrechner



Parallele Programme und Arbeitsplatzrechner

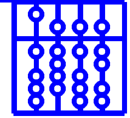


Parallele Programme und Arbeitsplatzrechner



© Thomas Ludwig, TU München

Historie



verteilte und parallele Programmierung

parallele Ein-/Ausgabe

heterogene Prog'modelle

Topologien

1986 → 1992 → 1996 → 2000

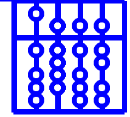
1986: Topologien — 80286, Speicherkopplung

1992: heterogene Prog'modelle — SUN / IBM, Ethernet / ATM

1996: parallele Ein-/Ausgabe — SUN, Ethernet

2000: verteilte und parallele Programmierung — Linux SMPs, FastEthernet, SCI-Kopplung

Moderne vernetzte Arbeitsplatzrechner



Hardwareentwicklung

- Steigende Leistung der Prozessoren
- Steigende Leistung der Verbindungsnetze

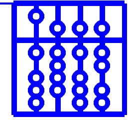
Rechnernetze vs. Parallelrechner

Neueste TOP500-Liste

- Platz 62: Sandia National Lab, Cplant Project
  233 GFlops mit 580 Alpha-Prozessoren; RedHat Linux, Myrinet,
- Platz 54: ZIB, T3E-900
  254 GFlops mit 404 Alpha-Prozessoren

© Thomas Ludwig, TU München

Entwicklungsmethodik paralleler Programme



Ausgangspunkt

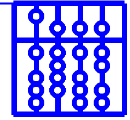
- Neuentwicklungen (10%)
- Aufarbeitung sequentieller Programme (90%)
  Sogenannter Legacy-Code oder Dusty-Deck-Code
  Meist nur Quellcode vorhanden; kein Know-How

Bei Neuentwicklungen

- Möglicherweise Spezifikation des Parallelismus
- Einsatz neuer Paradigmen
  z.B. objektorientierte und skelettorientierte Programmierung

© Thomas Ludwig, TU München

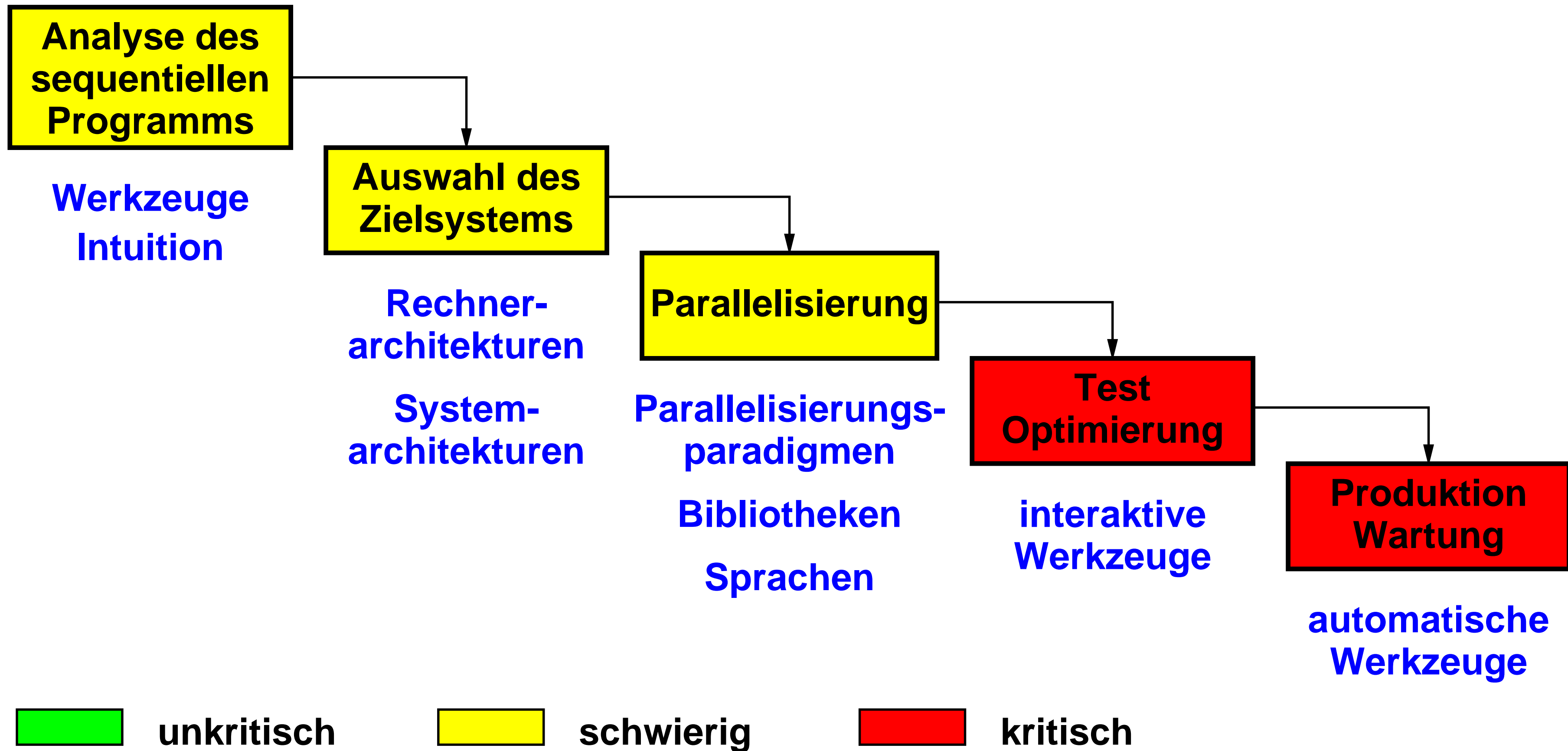
Entwicklungsmethodik paralleler Programme...



Aufarbeitung sequentieller Programme

- **Analyse des sequentiellen Programms**
  - Werkzeuge
  - Intuition
- **Auswahl des Zielsystems**
  - Rechner-architekturen
  - System-architekturen
- **Parallelisierung**
  - Parallelisierungs-paradigmen
  - Bibliotheken
  - Sprachen
- **Test Optimierung**
  - interaktive Werkzeuge
- **Produktion Wartung**
  - automatische Werkzeuge

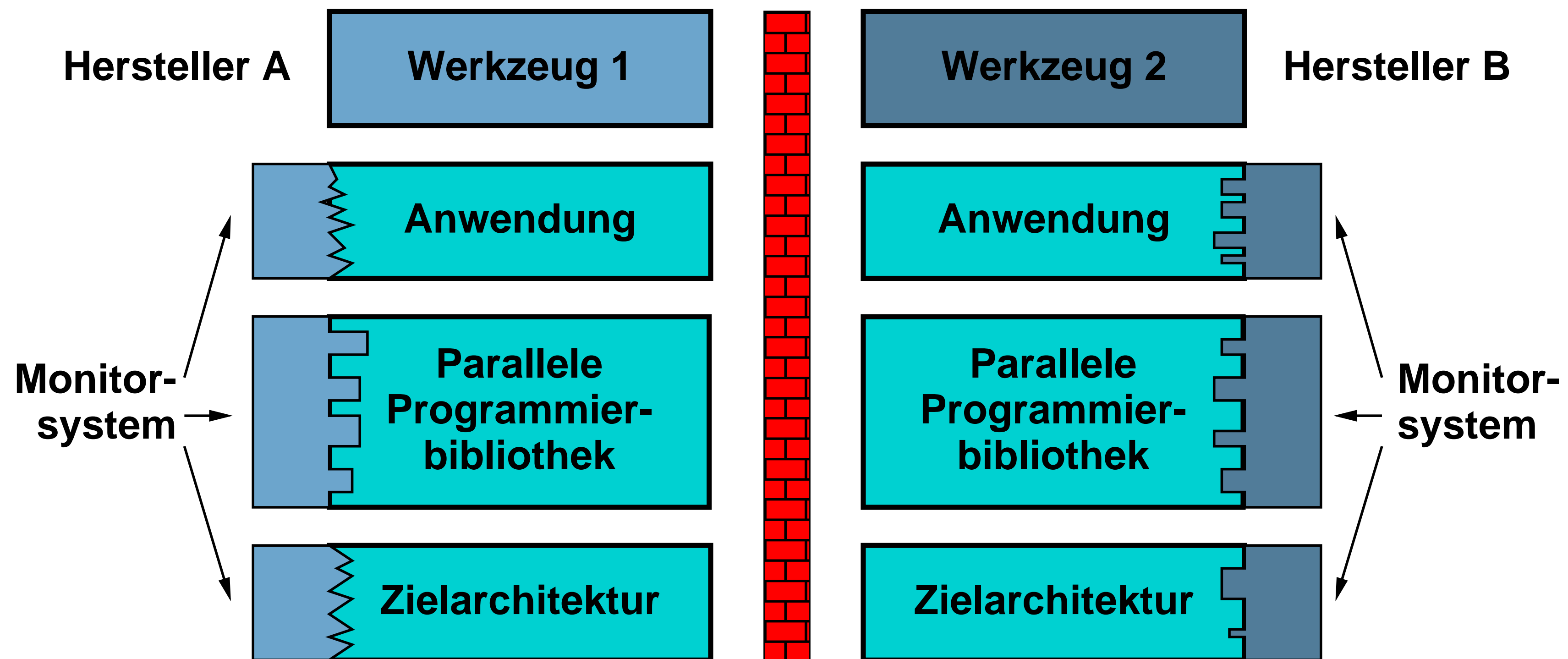
Bewertung der Entwicklungsmethodik



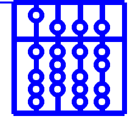
© Thomas Ludwig, TU München

Situation der Entwicklungswerkzeuge

- Kein theoretisches Fundament (Betriebssysteme, verteilte Systeme)
- Ad-hoc-Entwurf mit monolithischem Aufbau



Problemstellungen



Aufwendige Werkzeugkonstruktion

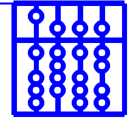
- Ständige Neuentwicklung bereits vorhandener Funktionalitäten
- Benutzer hat schlechte Auswahl
- Konstruktion paralleler Software erschwert

Mangelhafte Konzepte bei den Werkzeugen

- Keine Kooperation möglich
  Grund: Werkzeuge wissen nichts voneinander
- Keine gemeinsame Verwendung möglich
  Grund: Werkzeuge verwenden verschieden modifizierte Systemkomponenten

© Thomas Ludwig, TU München

Ansatz für eine methodische Werkzeugkonstruktion



Prinzip: Schnittstelle zwischen Werkzeug und Monitorsystem einziehen

Folge: Monitorsystem nur einmal realisieren; beliebige Werkzeuge aufsetzen

Die früheren Probleme sind damit eliminierbar

- ➥ Werkzeuge verwenden inkompatible Systemmodifikationen
- ➥ Werkzeuge wissen nichts voneinander

Gemeinsame Schnittstelle für alle Werkzeuge

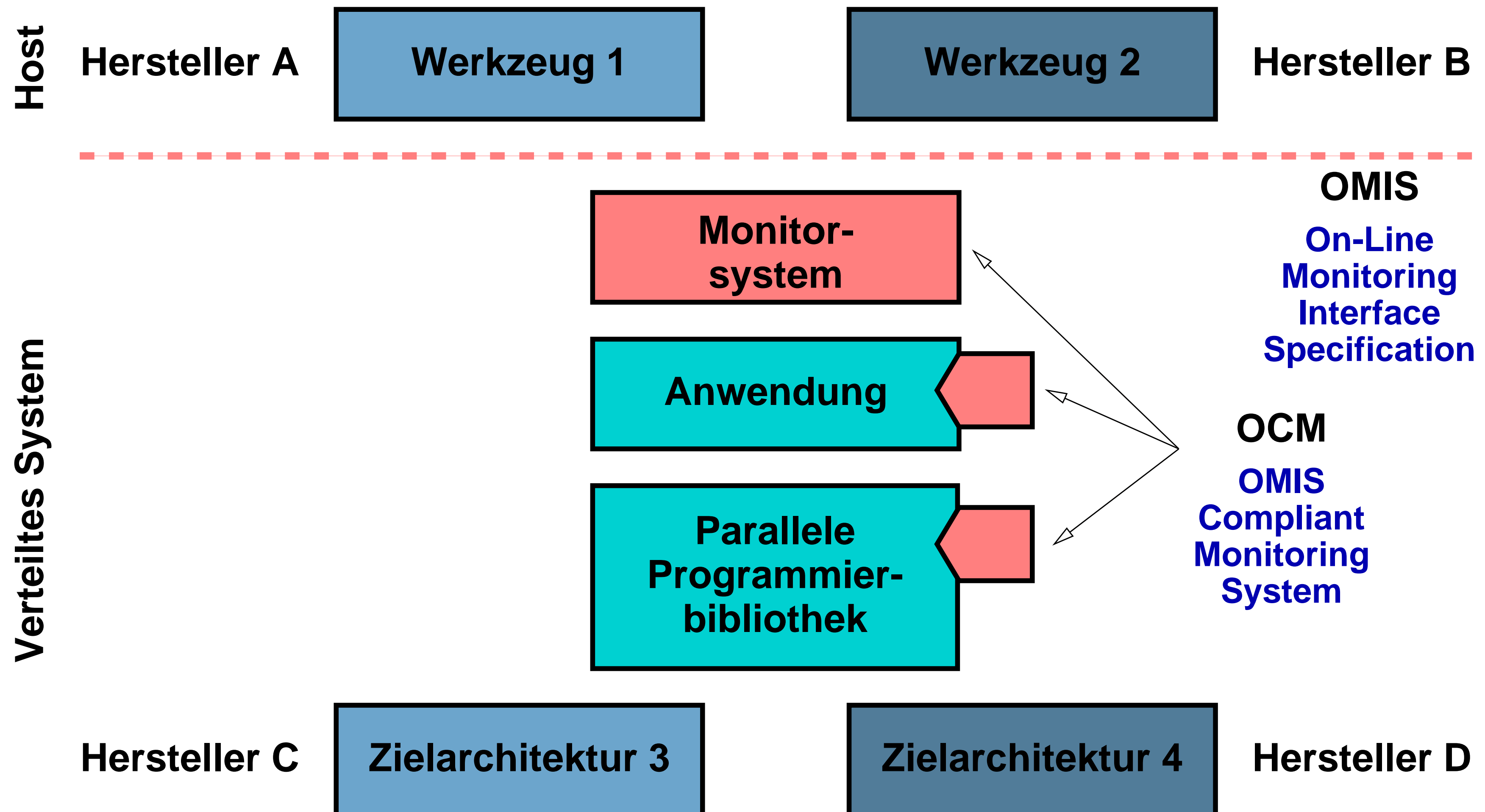
- ➥ Basis für Werkzeugintegration und Werkzeuginteroperabilität

Gemeinsames Monitorsystem auf verschiedenen Architekturen

- ➥ Basis für universelle Werkzeuge

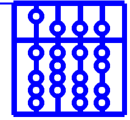
Methodische Werkzeugkonstruktion ...

Schnittstellenbasierte Werkzeugkonstruktion



© Thomas Ludwig, TU M¨unchen

Anforderungen an die Schnittstelle



Unterstützung von Interoperabilität und Universalität

- Mehrere Werkzeuge nebenläufig anwendbar
- Werkzeuge auf mehreren Zielarchitekturen ablauffähig

Allgemeingültigkeit und Flexibilität

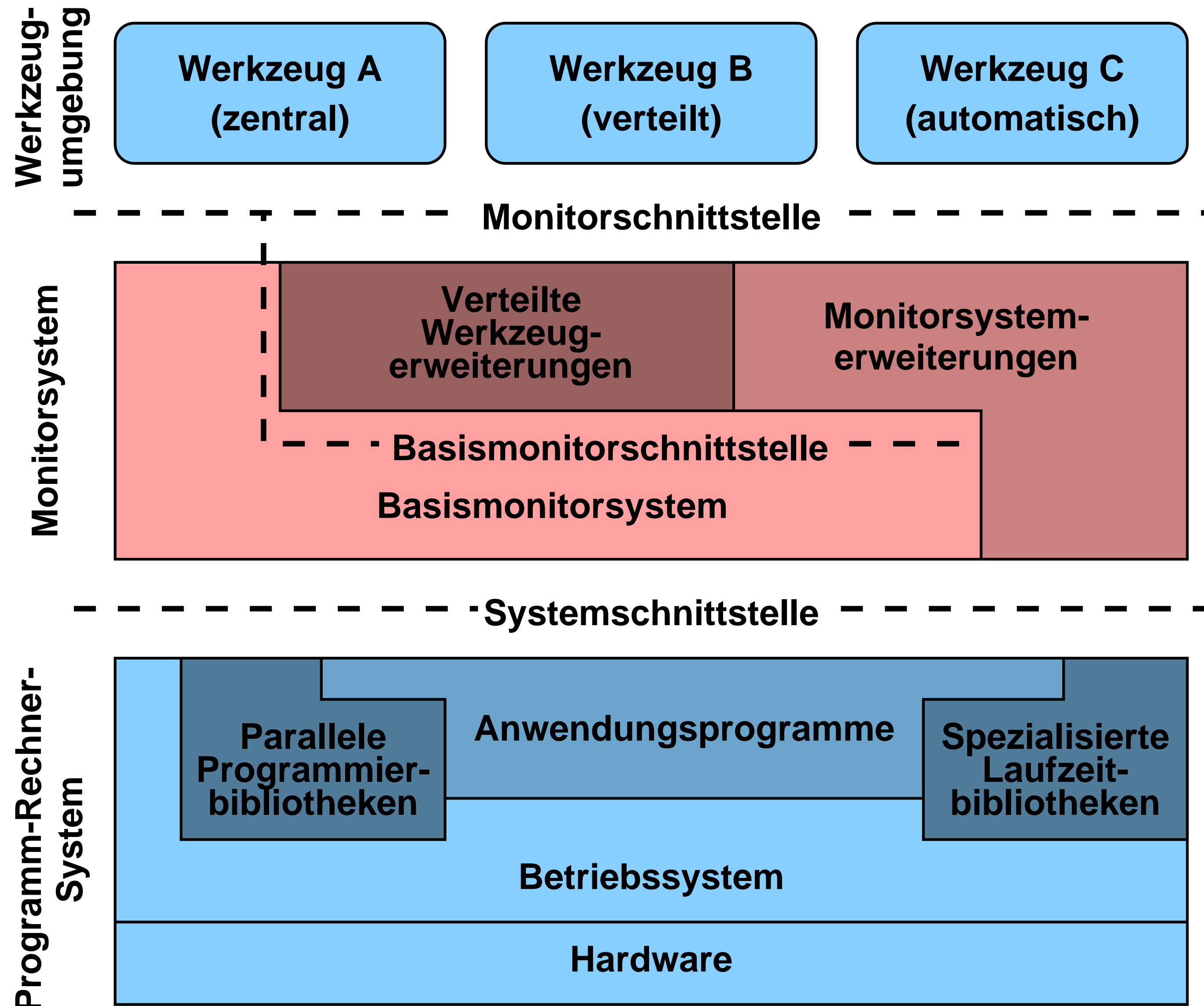
- Verschiedene Werkzeuge zu unterschiedlichen Zwecken
- Verschiedene Werkzeugarchitekturen
- Verschiedene programmiersprachliche Objekte

Erweiterbarkeit

- Neue Werkzeugkonzepte
- Neue zu überwachende Objekte

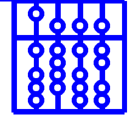
©Thomas Ludwig, TU München

Das Systemmodell



© Thomas Ludwig, TU M¨unchen

Objektbasierte Schnittstelle



Objektklassen

- System, Knoten
- Prozesse, Threads
- Nachrichtenpuffer, Nachrichten
- Monitorobjekte

Diensteklassen

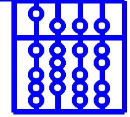
- Informationsdienste (z.B. liefere Liste der Prozesse)
- Manipulationsdienste (z.B. halte Prozeß an)
- Benachrichtigungsdienste (z.B. melde, daß Prozeß terminiert)

Sicht des Werkzeugs

- Monitorsystem ist paralleler Server, der Dienste an Objekten anbietet

© Thomas Ludwig, TU M¨unchen

Objektbasierte Schnittstelle ...



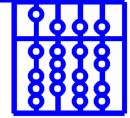
Arbeitsprinzip: Ereignis/Aktions-Modell

- **Ereignis** = interessierender Zustandsübergang im überwachten System
- **Aktion** = erwünschte Beobachtung oder Manipulation des System

Dienstanforderungen

- Verhalten des Monitorsystems ist durch Ereignis/Aktions-Relationen bestimmt
- Programmierung der Relationen über die definierte Schnittstelle
- Aktionen jeweils ausgelöst, wenn das Ereignis erkannt wird
- Leere Ereignisdefinition $\Rightarrow$ unbedingte Aktion

Funktionalität der Monitorschnittstelle



Syntax der Schnittstellenfunktion

```
Omis_reply
omis_request( char * request ,
              void (* callback)(Omis_reply reply, void *param),
              void *param,
              Omis_flags flags )
```

Syntax der Ereignis/Aktions-Relationen

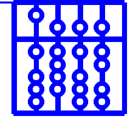
request	::=	[*event_definition*] : *action_list*
event_definition	::=	*service_name* (*parameters*)
action_list	::=	*action* | *action* [;] *action_list*
action	::=	*service_name* (*parameters*)

Semantische Details

- Dienste arbeiten auf Einzelobjekten und Objektmengen
- Dienste arbeiten auf Objekten beliebiger Hierarchiestufen
- Aktionen können potentiell nebenläufig ausgeführt werden

© Thomas Ludwig, TU München

Beispiel Leistungsanalyse



Gewünschte Information: Verweilzeit im Sendeaufruf und
Anzahl gesendete Datenelemente

Was ist zu tun

- Starten und Stoppen einer Stoppuhr
- Verwendung eines inkrementierenden Zählers

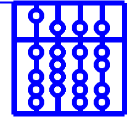
```
thread_has_started_lib_call([p_21],”MPI_Send”) :
    pt_integrator_start(pt_i_1)  pt_counter_add(pt_c_1,$par5)
```

- Wenn Prozeß p_21 einen Sendeaufruf startet: aktiviere einen integrierenden Zähler und addiere die Anzahl Datenelemente ($par5) auf einen Zähler

```
thread_has_ended_lib_call([p_21],”MPI_Send”) : pt_integrator_stop(pt_i_1)
```

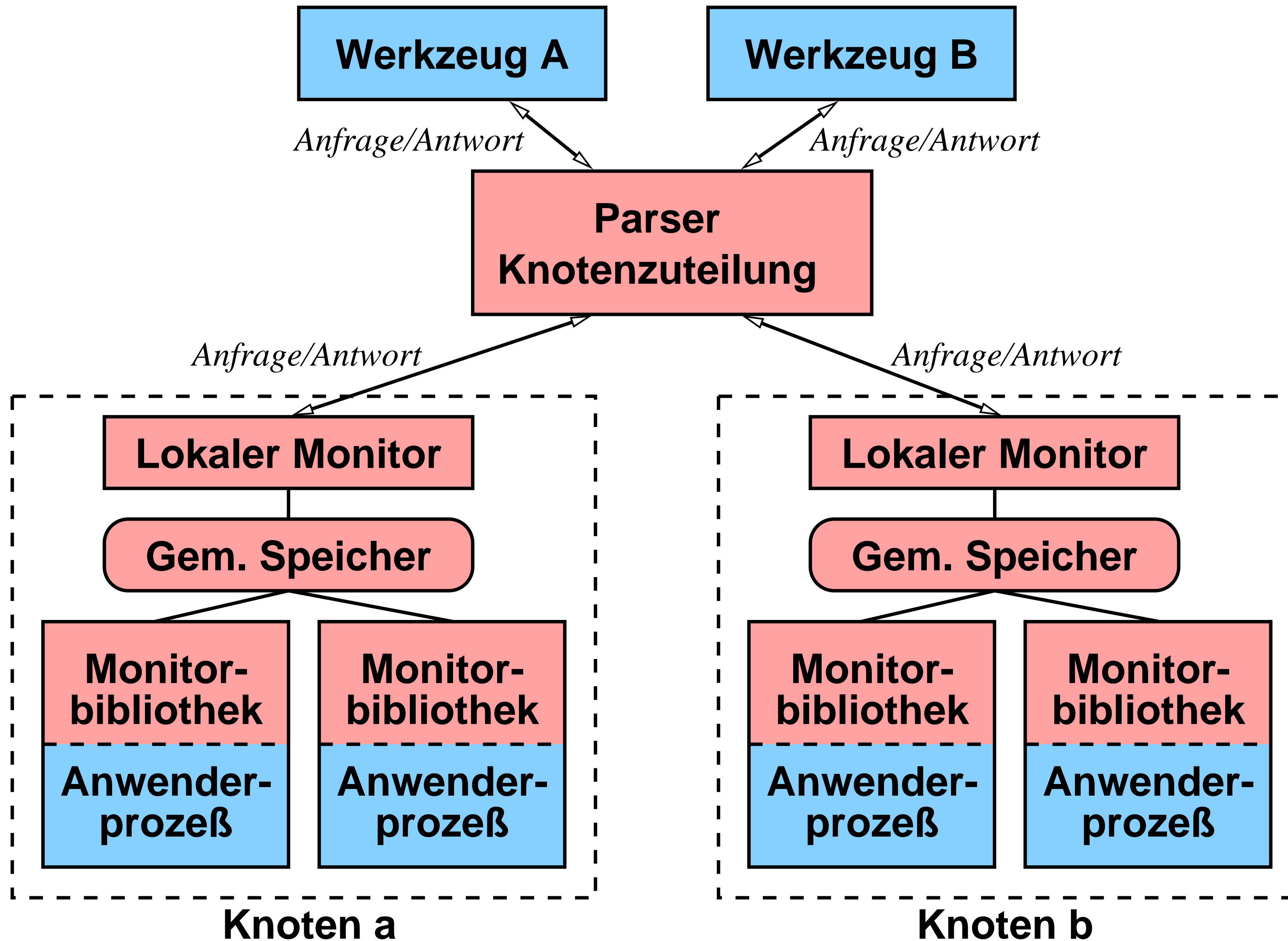
- Wenn der Prozeß den Aufruf beendet: halte den integrierenden Zähler an

Entwurfskonzepte des Monitorsystems



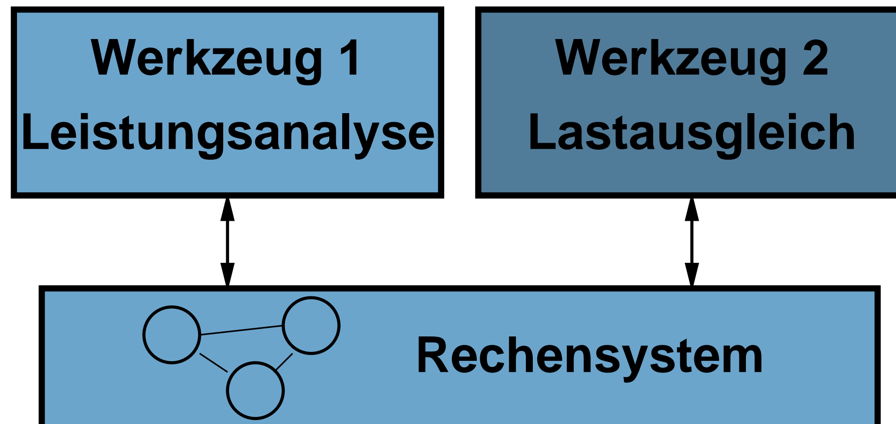
- Das Monitorsystem wird aus kommunizierenden Monitoren gebildet, jeweils einem pro Rechnerknoten der virtuellen Maschine
- Globale Aktionsspezifikationen werden zur Definitionszeit an die Knoten verteilt.
- Kommunikation zwischen Monitoren mittels Nachrichtenaustausch
- Im ersten Schritt: Zentrales Parsen der Dienstanforderungen
- Ereignisgetriebenes Ausführungsmodell der (Einzelknoten-)Monitore
- Ereigniserkennung und Aktionsausführung sowohl im Monitor als auch in der Anwendung
- Kommunikation zwischen knotenlokalen Monitorkomponenten über gemeinsamen Speicher und Signale

Grobstruktur des Monitorsystems



© Thomas Ludwig, TU München

Das Konzept der Interoperabilität



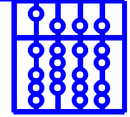
Problem

- Mehr als ein Werkzeug kennt den Zustand von O
- Mindestens ein Werkzeug verändert den Zustand von O
- Frage: Was machen die anderen Werkzeuge?

Notwendige Voraussetzung

- Einheitliche Schnittstelle für alle Werkzeuge
- Gemeinsames Monitorsystem

© Thomas Ludwig, TU München



Das Konzept der Interoperabilität ...

Zielvorstellung

- Mehrere Werkzeuge können nebenläufig konsistent genutzt werden

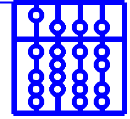
Varianten in obigem Beispiel aus Werkzeugsicht

- Leistungsanalyse weiß nichts vom Lastausgleich
- Leistungsanalyse kennt Migration als Zustandswechsel

Varianten aus Benutzersicht

- Werkzeug visualisiert die Prozeßmigration
  (Erwünschte Sicht eines Werkzeugentwicklers)
- Werkzeug verdeckt die Prozeßmigration
  (Erwünschte Sicht eines Anwendungsprogrammierers)

Modellierung der Interoperabilität



Ausgangspunkt: Objektsicht des Systems

Frage: Wer greift wann wie auf Objekte zu?

Zugriffsarten: Lesend, modifizierend

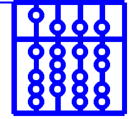
Zugriffskonflikt

- Inkompatibler zeitlich überlappender Zugriff zweier Werkzeuge auf ein gemeinsames Objekt O, wobei mindestens ein Zugriff modifizierend ist.

  Möglichkeiten: RW, WR, WW

Mögliche Folgen von Zugriffskonflikten

- Abbruch des Programms
- Falsche Informationen im Werkzeug
- Abbruch des Werkzeuglaufs



Konzept konflktfreier Werkzeuge

Konsistenzproblem

- Werkzeug A bearbeitet ein Objekt, das von Werkzeug B später modifiziert wird.

  Situationen: RW und WW

- Lösung: Werkzeug bittet um Benachrichtigung, wenn der Zustand eines Objekts verändert wird

Transparenzproblem

- Werkzeug B greift auf ein Objekt zu, das von Werkzeug A modifiziert wurde

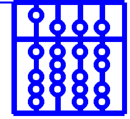
  Situationen: WR und WW

- Lösung: noch nicht endgültig erarbeitet

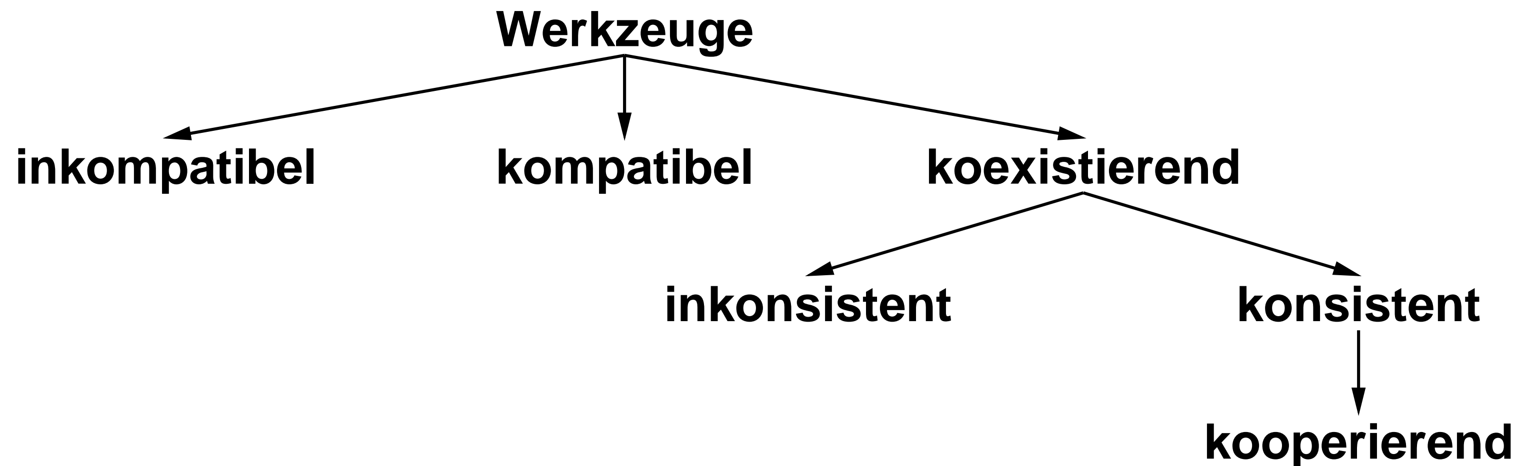
  Transaktionsmechanismen und Sperren: zu hohe Zusatzlast

  Abfangen von Zugriffen anderer Werkzeuge (eigene Modifikationen verdecken)

© Thomas Ludwig, TU M¨unchen

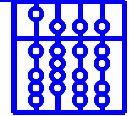


Neue Werkzeugeklassifikation



Beispiel: Leistungsanalyse / Lastausgleich

- Inkonsistent koexistierend: Leistungsanalysator ist verwirrt durch Verschwinden und Auftauchen von Prozessen
- Konsistent koexistierend: Leistungsanalysator erkennt Verschiebungen und visualisiert sie falls gewünscht. Ansonsten verdeckt er sie
- Kooperierend: Leistungsanalysator gibt selber Anstoß zur Lastverwaltung; Lastverwalter beauftragt Leistungsanalysator mit der Visualisierung der Verwaltungseffekte



Stand der Technik

PCTE — Portable Common Tool Environment (Esprit 1983)

➥ Interoperabilität bei Software-Entwurfswerkzeugen

ToolTalk (Sun Microsystems)

➥ Interoperabilität für Programme unter X Windows

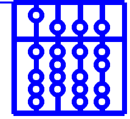
DAMS (Universidade Nova de Lisboa, Lissabon, Portugal)

➥ Einheitliches Monitorsystem mit Ansätzen zu Interoperabilität
keine Konfliktauflösung

DPCL — Dynamic Probe Class Library (IBM u.a.)

➥ Monitorschnittstelle ohne Konzepte zur Interoperabilität

Erweiterung und Übertragung der Konzepte

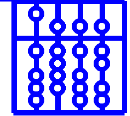


Verteilter gemeinsamer Speicher

- Hier wichtig: Adreßraumbereiche
- Erweiterung von OMIS um die Objektklasse „Speicherbereich“
- Erweiterung von OMIS um neue Dienste zu dieser Objektklasse
- Anpassung des Monitorsystems an diesen Dienste

CORBA-basierte Programmsysteme

- Übertragung der Schnittstellenkonzepte in den Bereich der Middleware-Systeme
- Adaption einzelner Komponenten an CORBA



Status der Arbeiten

Modell der schnittstellenbasierten Werkzeugkonstruktion
1996: OMIS 1.0

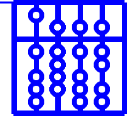
Konzepte zur Interoperabilität
1997: OMIS 2.0

Entwurf und Realisierung eines verteilten Monitorsystems
1997-1999 OCM

Entwurf und Realisierung interoperabler Online-Werkzeuge
Herbst 1999: Fehlersuche und Sicherungspunktgenerator

Übertragung der Konzepte auf andere Zielarchitekturen
OMIS 3.0 MiMo (Middleware Monitoring System)

Anwendungsbeispiel: Verkehrssteuerung



SEEDS - Simulation Environment for the Evaluation of Distributed Traffic Control Systems (Esprit IV)

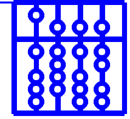
Aufgabenstellung

- Verteilter Simulator für Flughafenbodenverkehr

Anwendungsgebiete

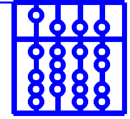
- Entwurf neuer Flughäfen, Optimierung bestehender
- Optimierung des Verkehrsdurchsatzes unter bestimmten Randbedingungen
- Training der Controller

Anwendungsbeispiel: Verkehrssteuerung...



Fragestellungen

- Auswahl geeigneter Rechner und Vernetzung
- Analyse des Datenverkehrs im Netz
- Einhalten von Echtzeit-Bedingungen
- Auswahl geeigneter Programmiermodelle (CORBA, DIS)



Anwendungsbeispiel: Computertomographie

Aufgabenstellung

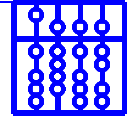
- Parallele Rekonstruktion bei PET und fMrI
- Einbindung in die klinische Routine

Anwendungsgebiete

- Untersuchung von Stoffwechselvorgängen im Gehirn
- Statistische Analyse bei Patientengruppen

© Thomas Ludwig, TU München

Anwendungsbeispiel: Computertomographie...



Forschungsfragen

- Auswahl geeigneter Rechner und Vernetzungen
  Doppelprozessor PC's mit FastEthernet und SCI-Vernetzung
- Vergleich verschiedener Programmiermodelle
  PVM, Treadmarks, CORBA
- Kopplung parallele und verteilte Programmierung

Künftig

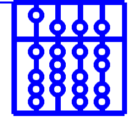
- Einbindung einer effizienten parallelen Ein-/Ausgabe
- Hardwarevernetzung für parallele Ein-/Ausgabe

Förderung

- Stifterverband; 1 offene Stelle

© Thomas Ludwig, TU München

Anwendungsbeispiel: Bioinformatik



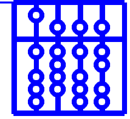
Aufgabenstellung

- Integration bestehender Algorithmen und Werkzeuge zur Phylogenie in ein vereinheitlichtes Werkzeug
- Effiziente Verwaltung von Sequenzinformationen
- Parallelisierung langlaufender Algorithmen zur Ausrichtung und Stammbaumanalyse

Anwendungsgebiete

- Phylogenetische Analysen bei Bakterien
- Sondendesign und DNA-Chips
- Genomforschung bei Meeresbakterien

© Thomas Ludwig, TU München



Anwendungsbeispiel: Bioinformatik...

Forschungsfragen

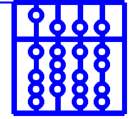
- ➥ Integration bestehender Werkzeuge (Interoperabilität)
- ➥ Vernetzte Arbeitsumgebungen
- ➥ Parallelisierung rechenintensiver Algorithmen
- ➥ Entwurf effizienter Ein-/Ausgabekomponenten in HW und SW

Förderungen

- ➥ BMBF, Bayerische Forschungsstiftung; 3 offene Stellen

© Thomas Ludwig, TU München

Zusammenfassung und Ausblick



Parallele Programme und vernetzte Arbeitsplatzrechner

- Es ist leichter, ein System zusammenzubauen, als es effizient zu nutzen
- Insbesondere im Bereich Online-Werkzeuge und Unterstützung für Ein-/Ausgabe gibt es erhebliche Defizite

Künftige Entwicklung

- Integration von parallelem Rechnen und verteilter Programmierung
- Zunehmende Wichtigkeit der effizienten parallelen Ein-/Ausgabe
- Ausgeweitete Einsatzgebiete