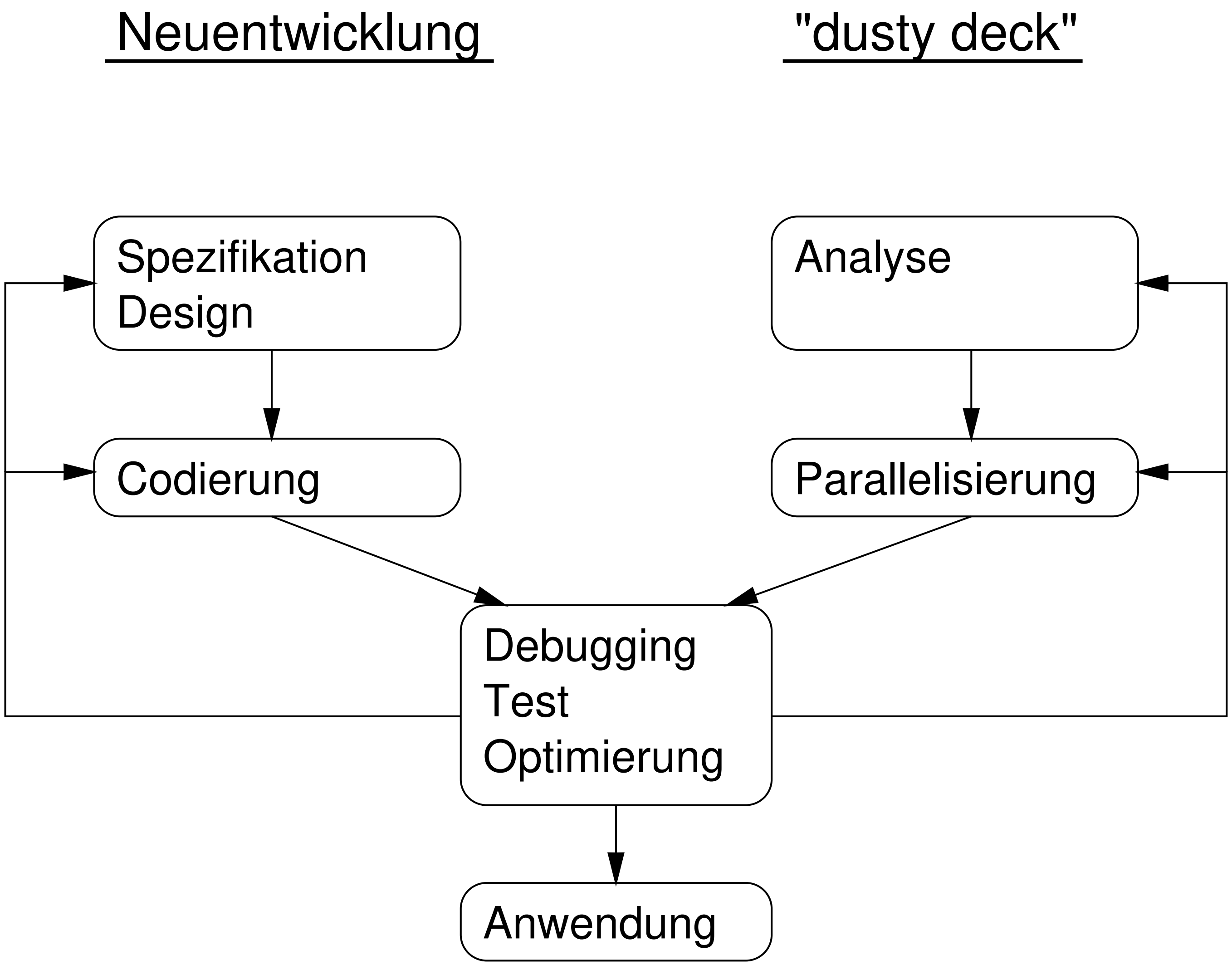
Werkzeuge für die parallele Programmierung

Roland Wismüller

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

Entwicklungszyklus paralleler Programme



Beispiele für Werkzeuge (1)

Spezifikation und Design:

- Formale Methoden: z.B. FOCUS
  - Systemkomponenten spezifiziert als stromverarbeitende Funktionen
  - Eigenschaften beschrieben durch logische Prädikate über Strömen
  - Gesamtsystem entsteht durch Kompositionsregeln
- Graphische Methoden: z.B. Hamlet, Trapper, Hence
  - Graphische Spezifikation der Prozeß- und Kommunikationsstruktur
    $\leadsto$ Coderahmen für die Implementierung
    $\leadsto$ Simulation des Systems (Hamlet)
  - Darstellung von Laufzeitinformation in der Spezifikation
  - Probleme: Replikation, Dynamik, Konsistenz der Implementierung

Beispiele für Werkzeuge (2)

Codierung:

- Parallele Programmierbibliotheken (PVM, MPI, BLAS, ...)
- Codegeneratoren in Designwerkzeugen
- Parallele Sprachen
  - HPF: Fortran90 + Array-Verteilung

```
!HPF$ PROCESSORS P(4,4)
      REAL A(100,200)
!HPF$ DISTRIBUTE A(BLOCK,BLOCK) ONTO P
      ...
      a(i,j) = a(i+1,j-k)
```

  - Parallele C-Dialekte auf CSP-Basis
  - Parallele objektorientierte Sprachen
    Objekte im System verteilt, Spracherweiterungen für Plazierung und Synchronisation, Zugriff transparent

Beispiele für Werkzeuge (3)

Analyse / Parallelisierung:

- Interaktive Werkzeuge (z.B. Forge, MIMDizer)
  - Variablen-Browser: Definition/Benuzung von (globalen) Variablen (Common-Blöcke / Equivalence)
  - Programmstruktur: Prozeduraufrufgraphen ...
  - Schleifenanalyse: Abhängigkeiten zwischen Schleifeniterationen

  $\Rightarrow$ Datenaufteilung, Code-Umstrukturierung, Übersetzerdirektiven

- Automatisch parallelisierende Compiler
  - (V)LIW-Compiler (feingranularer Parallelismus)
  - Compiler für Systeme mit gemeinsamem Speicher

    $\rightsquigarrow$ Schleifenparallelität, Aufteilung auf Threads und Synchronisation
  - Compiler für Systeme mit verteiltem Speicher

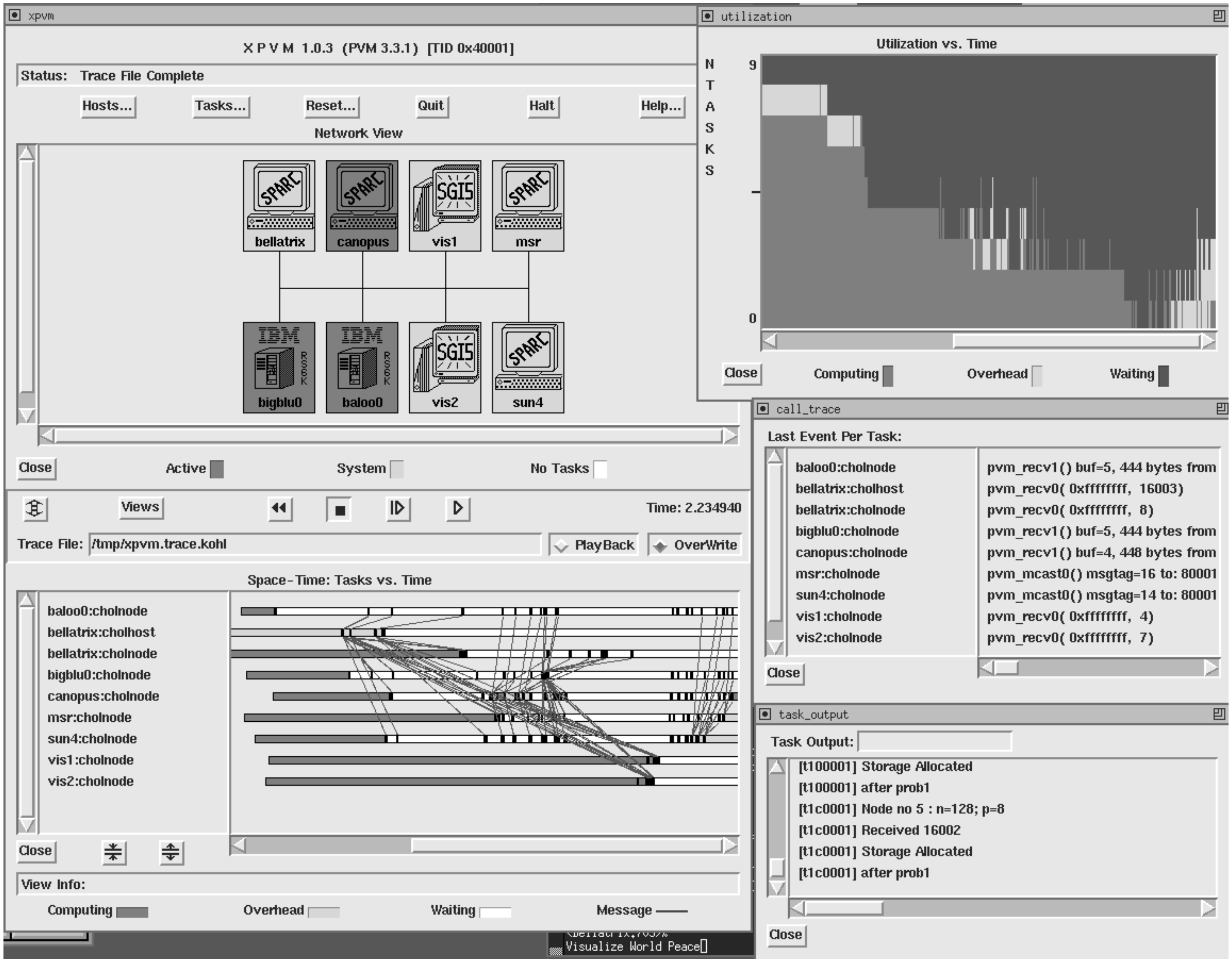
    $\rightsquigarrow$ Datenaufteilung (Forschung)

Beispiele für Werkzeuge (4)

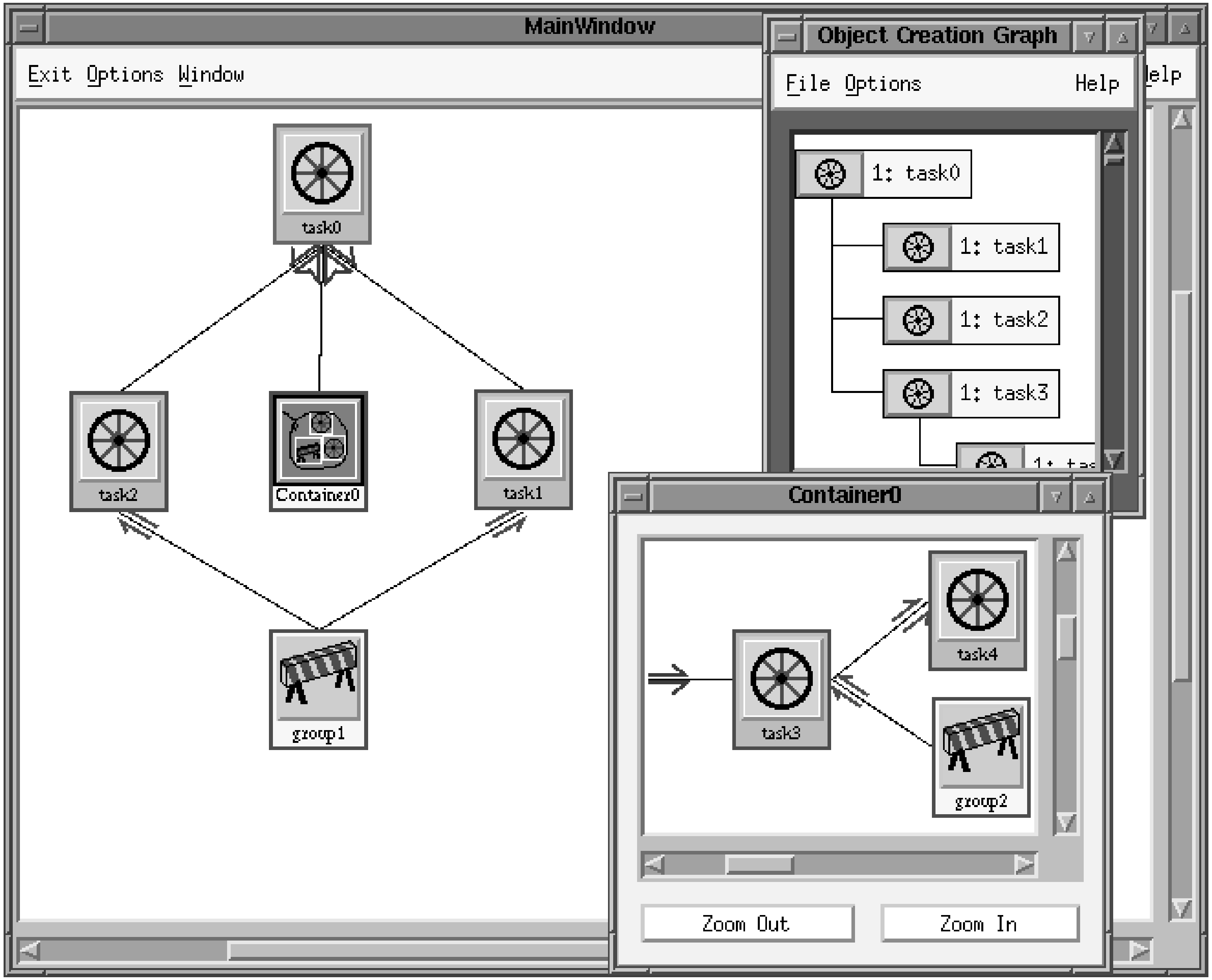
Debugging, Test, Optimierung:

- Statische Analysatoren
  - *Race*-Erkennung
  - *Performance Prediction*
- Interaktive Beobachtungswerkzeuge
  - Analyse (und Steuerung) eines Programmablaufs
    $\leadsto$ XPVM
    $\leadsto$ The Tool-set
- Automatische Laufzeit-Werkzeuge
  - *Race*-Erkennung („*on-the-fly*“)
  - *Replay*, Deterministische (Wieder-)Ausführung

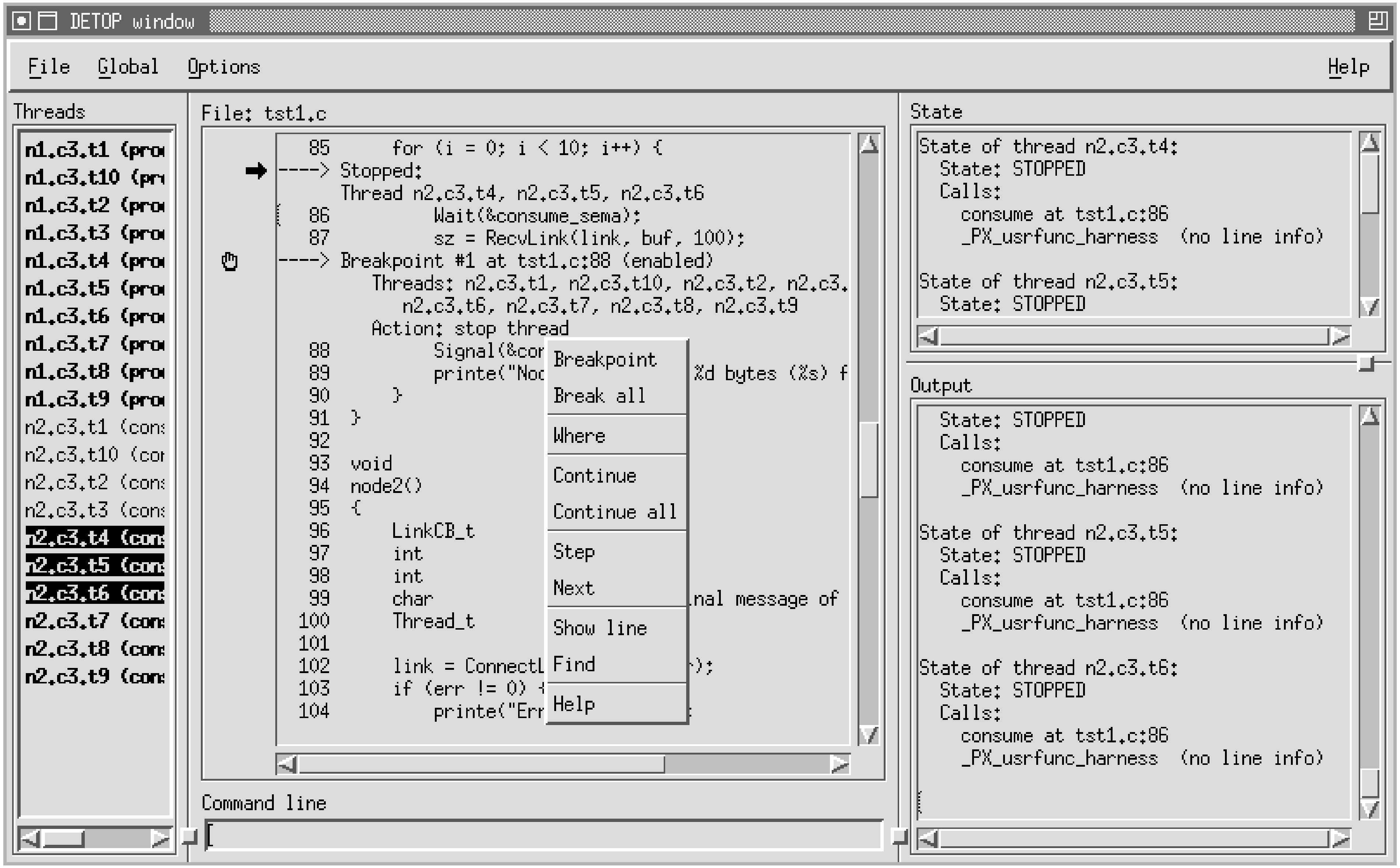
XPVM


xpvm
X P V M 1.0.3 (PVM 3.3.1) [TID 0x40001]
Status: Trace File Complete
Hosts...
Tasks...
Reset...
Quit
Halt
Help...
Network View
bellatrix
canopus
vis1
msr
bigblu0
baloo0
vis2
sun4
Close
Active
System
No Tasks
Views
Time: 2.234940
Trace File: /tmp/xpvm.trace.kohl
PlayBack
OverWrite
Space-Time: Tasks vs. Time
baloo0:cholnode
bellatrix:cholhost
bellatrix:cholnode
bigblu0:cholnode
canopus:cholnode
msr:cholnode
sun4:cholnode
vis1:cholnode
vis2:cholnode
Close
View Info:
Computing
Overhead
Waiting
Message
Visualize World Peace
utilization
Utilization vs. Time
NTASKS
9
0
Close
Computing
Overhead
Waiting
call_trace
Last Event Per Task:
baloo0:cholnode pvm_recv1() buf=5, 444 bytes from
bellatrix:cholhost pvm_recv0(0xffffffff, 16003)
bellatrix:cholnode pvm_recv0(0xffffffff, 8)
bigblu0:cholnode pvm_recv1() buf=5, 444 bytes from
canopus:cholnode pvm_recv1() buf=4, 448 bytes from
msr:cholnode pvm_mcast0() msgtag=16 to: 80001
sun4:cholnode pvm_mcast0() msgtag=14 to: 80001
vis1:cholnode pvm_recv0(0xffffffff, 4)
vis2:cholnode pvm_recv0(0xffffffff, 7)
Close
task_output
Task Output:
[t100001] Storage Allocated
[t100001] after prob1
[t1c0001] Node no 5 : n=128; p=8
[t1c0001] Received 16002
[t1c0001] Storage Allocated
[t1c0001] after prob1
Close

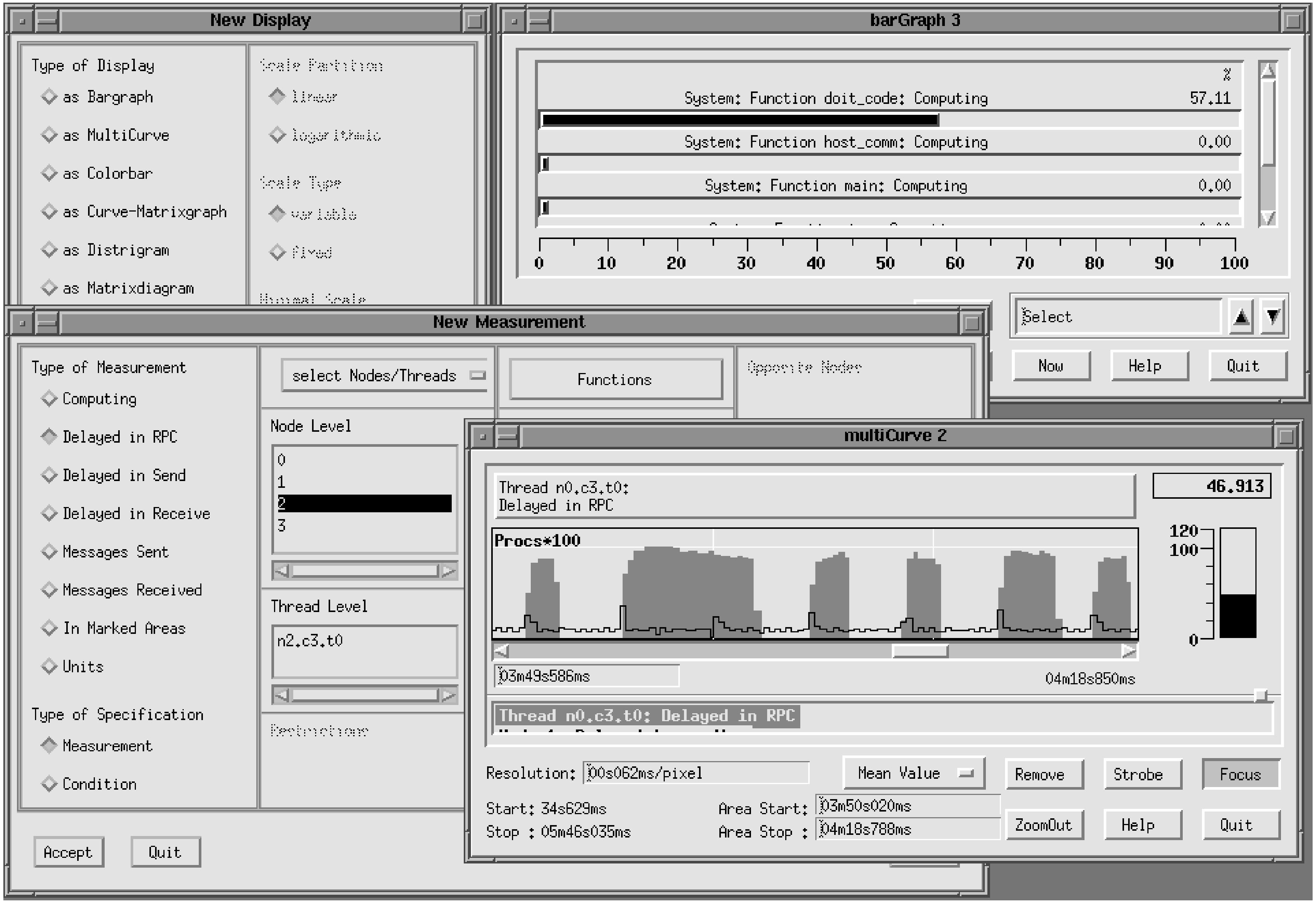
VISTOP


MainWindow
Exit Options Window
task0
task2
Container0
task1
group1
Object Creation Graph
File Options Help
1: task0
1: task1
1: task2
1: task3
Container0
task4
task3
group2
Zoom Out
Zoom In

DETOP


DETOP window
File
Global
Options
Help
Threads
n1.c3.t1 (pro
n1.c3.t10 (pr
n1.c3.t2 (pro
n1.c3.t3 (pro
n1.c3.t4 (pro
n1.c3.t5 (pro
n1.c3.t6 (pro
n1.c3.t7 (pro
n1.c3.t8 (pro
n1.c3.t9 (pro
n2.c3.t1 (con
n2.c3.t10 (co
n2.c3.t2 (con
n2.c3.t3 (con
n2.c3.t4 (con
n2.c3.t5 (con
n2.c3.t6 (con
n2.c3.t7 (con
n2.c3.t8 (con
n2.c3.t9 (con
File: tst1.c
85 for (i = 0; i < 10; i++) {
----> Stopped:
Thread n2.c3.t4, n2.c3.t5, n2.c3.t6
86 Wait(&consume_sema);
87 sz = RecvLink(link, buf, 100);
----> Breakpoint #1 at tst1.c:88 (enabled)
Threads: n2.c3.t1, n2.c3.t10, n2.c3.t2, n2.c3.
n2.c3.t6, n2.c3.t7, n2.c3.t8, n2.c3.t9
Action: stop thread
88 Signal(&cor
89 printe("Noc
%d bytes (%s) f
90 }
91 }
92
93 void
94 node2()
95 {
96 LinkCB_t
97 int
98 int
99 char
nal message of
100 Thread_t
101
102 link = Connectl
);
103 if (err != 0) {
104 printe("Err
Breakpoint
Break all
Where
Continue
Continue all
Step
Next
Show line
Find
Help
Command line
State
State of thread n2.c3.t4:
State: STOPPED
Calls:
consume at tst1.c:86
_PX_usrfunc_harness (no line info)
State of thread n2.c3.t5:
State: STOPPED
Output
State: STOPPED
Calls:
consume at tst1.c:86
_PX_usrfunc_harness (no line info)
State of thread n2.c3.t5:
State: STOPPED
Calls:
consume at tst1.c:86
_PX_usrfunc_harness (no line info)
State of thread n2.c3.t6:
State: STOPPED
Calls:
consume at tst1.c:86
_PX_usrfunc_harness (no line info)

PATOP


New Display
Type of Display
as Bargraph
as MultiCurve
as Colorbar
as Curve-Matrixgraph
as Distrigram
as Matrixdiagram
barGraph 3
%
System: Function doit_code: Computing 57.11
System: Function host_comm: Computing 0.00
System: Function main: Computing 0.00
0 10 20 30 40 50 60 70 80 90 100
Select
Now
Help
Quit
New Measurement
Type of Measurement
Computing
Delayed in RPC
Delayed in Send
Delayed in Receive
Messages Sent
Messages Received
In Marked Areas
Units
Type of Specification
Measurement
Condition
select Nodes/Threads
Functions
Node Level
0
1
2
3
Thread Level
n2.c3.t0
Accept
Quit
multiCurve 2
Thread n0.c3.t0:
Delayed in RPC
46.913
Procs*100
120
100
0
03m49s586ms
04m18s850ms
Thread n0.c3.t0: Delayed in RPC
Resolution: 00s062ms/pixel
Mean Value
Remove
Strobe
Focus
Start: 34s629ms
Stop : 05m46s035ms
Area Start: 03m50s020ms
Area Stop : 04m18s788ms
ZoomOut
Help
Quit

Beispiele für Werkzeuge (5)

Anwendung:

- Start und Steuerung: z.B. PVM Konsole
- Ressourcen-Verwaltung: z.B. Condor, Codine
  - *Batch-Queueing*
  - Zuteilung der Ressourcen
  - Überwachung der Ressourcen-Nutzung, Abrechnung
- Lastausgleich, Fehlertoleranz
  - Basis: *Checkpointing*

    $\leadsto$ Erfassung (und Sicherung) aller/einzelner Prozeßzustände

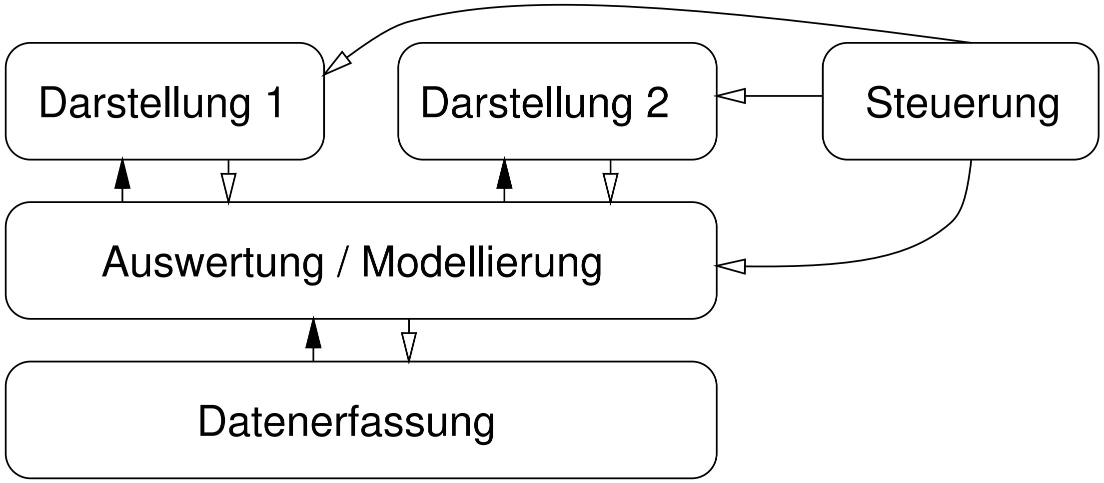
    $\leadsto$ Weiterführung aller/einzelner Prozesse ab *checkpoint*
  - Problem: Kommunikation (Netzzustand, Routing)

Werkzeuge zur Laufzeitbeobachtung

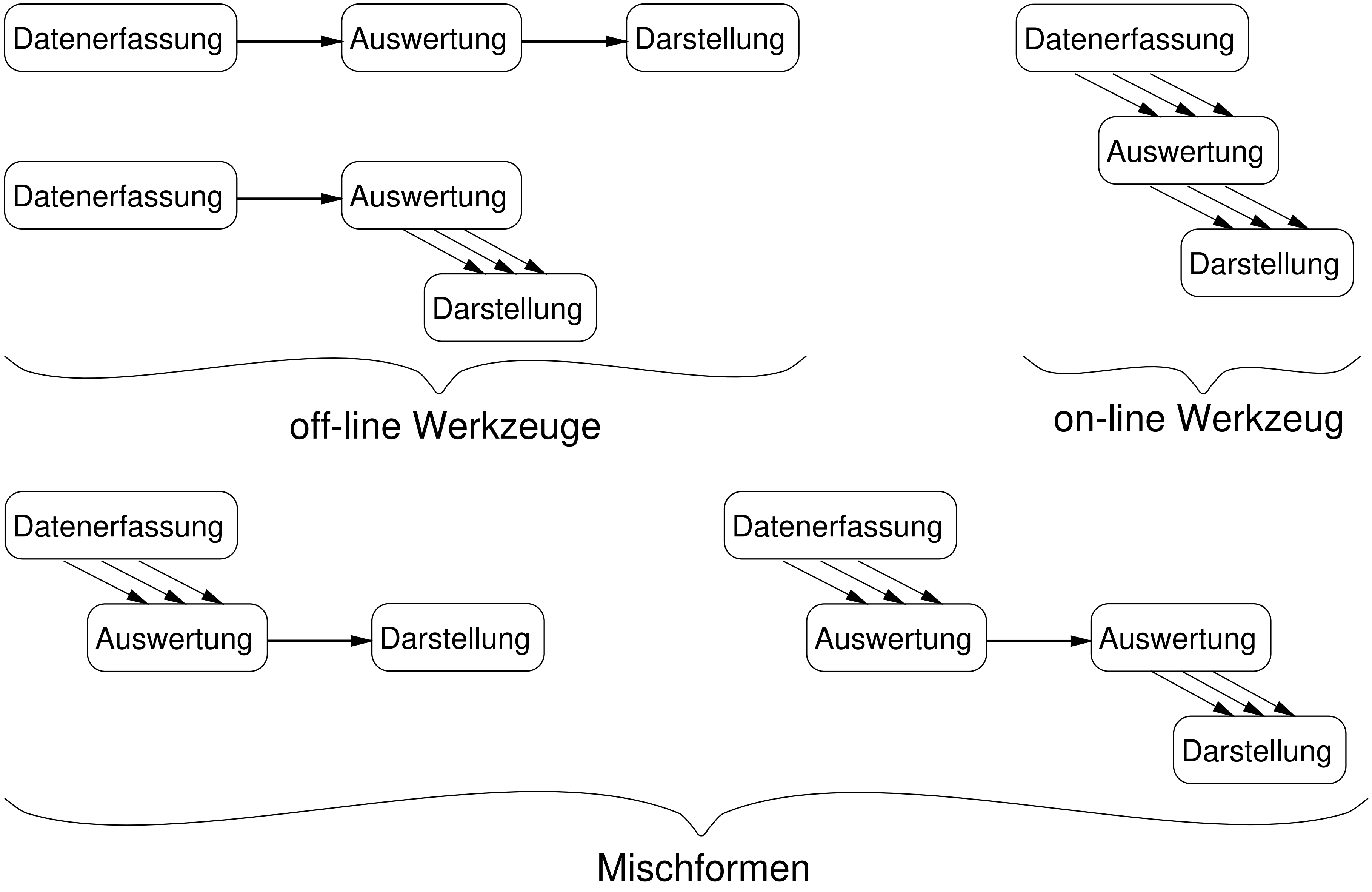
Aufgaben der Werkzeuge:

- Erfassung von <u>Daten</u> über das Ablaufverhalten eines parallelen Programms
- <u>Auswertung</u> und <u>Darstellung</u> dieser Daten
  - Angepaßt an jeweilige Problemstellung
  - Angepaßt an Denkweise und -kategorien des Benutzers
  - $\leadsto$ Wesentliche Unterschiede der Werkzeugen
- Evtl. auch Modifikation / Steuerung des Programmablaufs

Struktur:



Zeitliche Anordnung der Phasen



Aufgaben der Phasen

		Debugger	Visualisierer	Leistungsanalyse
Datenerfassung	Zustand	Prozesse Prozeßzustände Speicherinhalte Registerinhalte Keller Warteschlangen	---	Prozesse
	Ereignisse	Haltepunkte Signale Prozeßerzeugung Interaktion	Prozeßerzeugung Interaktion	Interaktion Code-Abschnitte
Auswertung		Umsetzung des Sprachniveaus Zusammenfassen identischer Daten	Modellierung des Globalzustands aus Ereignissen	statistische Auswertung zeitliche Korrelation
Darstellung		i.W. textuell z.T. graphisch	graphisch	graphisch z.T. textuell

Ein spezielles Problem: Verteiltheit

Situation:

- Reales System besteht aus asynchron arbeitenden unabhängigen Einheiten
  - $\leadsto$ Keine eindeutige Reihenfolge von Ereignissen
  - $\leadsto$ Kein eindeutiger globaler Zustand
- Werkzuge wollen/müssen ein zentral getaktetes (synchrones) System mit wohldefiniertem globalen Zustand zeigen
  - $\leadsto$ Einfacheres Denkmodell (sequentielle Schritte, Schritte intern parallel)
  - $\leadsto$ Sinnvolles Werkzeug zwangsläufig zentral
    - $\Rightarrow$ Darstellung von Ereignissen zwangläufig geordnet
    - $\Rightarrow$ Zentralisierte Darstellung assoziiert mit globalem Zustand

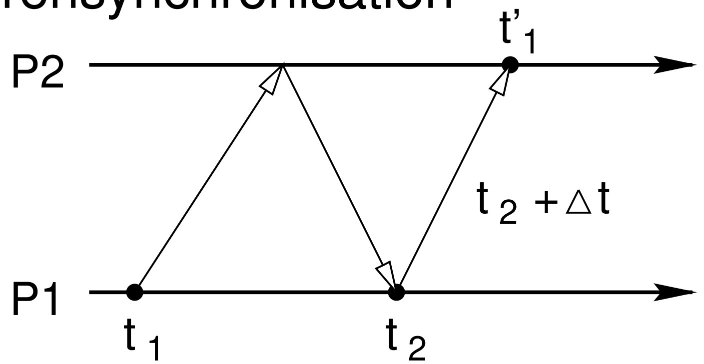
$\Rightarrow$ Einführung einer globalen Zeit im verteilten System
Globalzustand $Z(t) = (Z_1(t), Z_2(t), ..., Z_n(t))$

Erzeugung einer globalen Zeit (1)

1. Approximation einer globalen Zeit auf Basis realer Zeit

Vorgehensweisen:

- Uhrenstart mit *broadcast*-Nachricht
  - $\Delta t \sim$ Nachrichtenlaufzeit
  - Uhrendrift
- Uhrensynchronisation



$\triangle t = (t_2 - t_1) / 2$ Nachrichtenlaufzeit

P2 stellt bei t'_1 seine Uhr auf $t_2 + \triangle t$

  - Schwankungen der Nachrichtenlaufzeit $\Rightarrow$ statistische Methoden
  - Uhrendrift $\Rightarrow$ mehrfache Synchronisation, Abschätzung der Drift
  - Voraussetzung: Drift konstant (in der Praxis erfüllt)

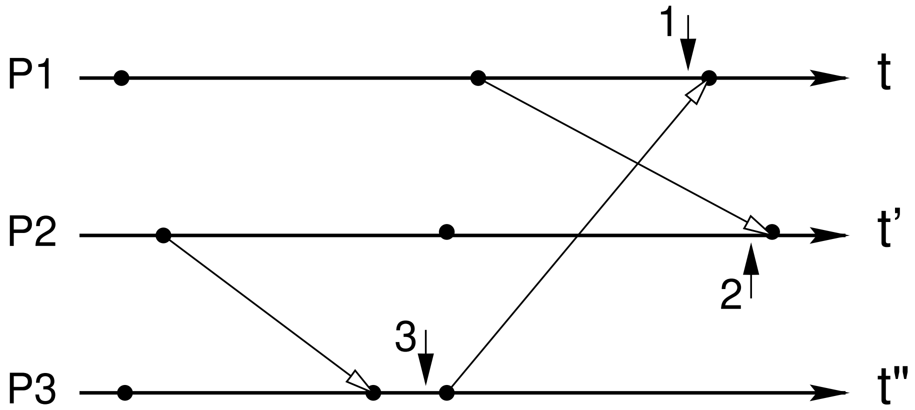
Erzeugung einer globalen Zeit (2)

2. Virtuelle globale Zeit auf Basis der Kausalität

Kausalität:

- Abhängigkeit zwischen Ereignissen in verschiedenen Komponenten, z.B. Empfang einer Nachricht erst nach dem Senden möglich
- Wichtig für logisches Programmverhalten (Debugging, Visualisierung)

Zentralisiertes Verfahren: Auswahl des nächsten Ereignisses

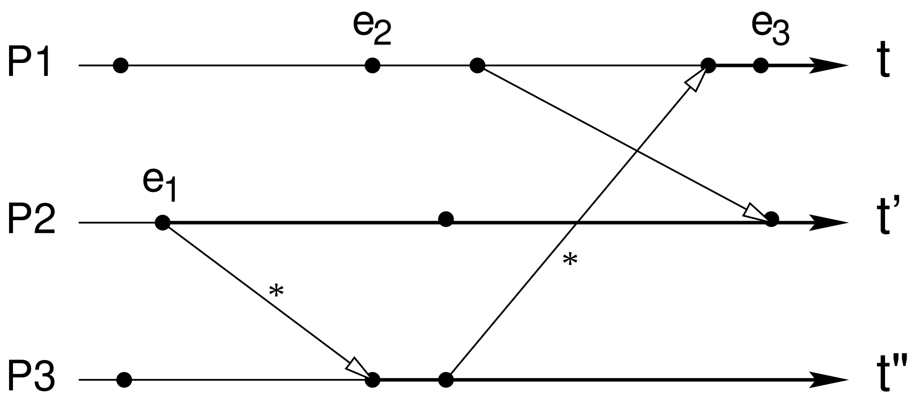


- Pro Komponente ein Zeiger auf (lokal) nächstes Ereignis
- Wähle Ereignis, das nicht von unbearbeitetem Ereignis abhängt
- Zeigerpositionen definieren globalen Zustand (*cut*)

Erzeugung einer globalen Zeit (3)

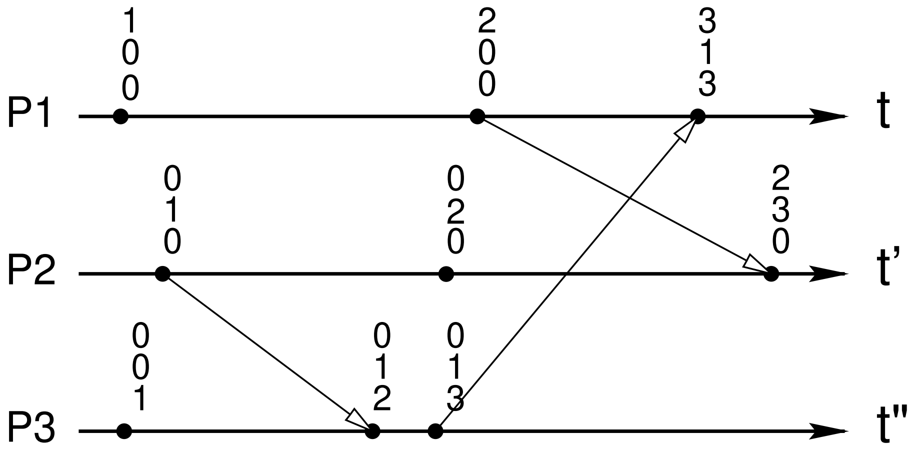
Verteilte Verfahren:

- Markierung: Ordnen einzelner Ereignisse



  - e_i nach e_1 g.d.w. Komponente markiert, wenn e_i eintritt
  - Verfolgen einer kausalen Abhängigkeit

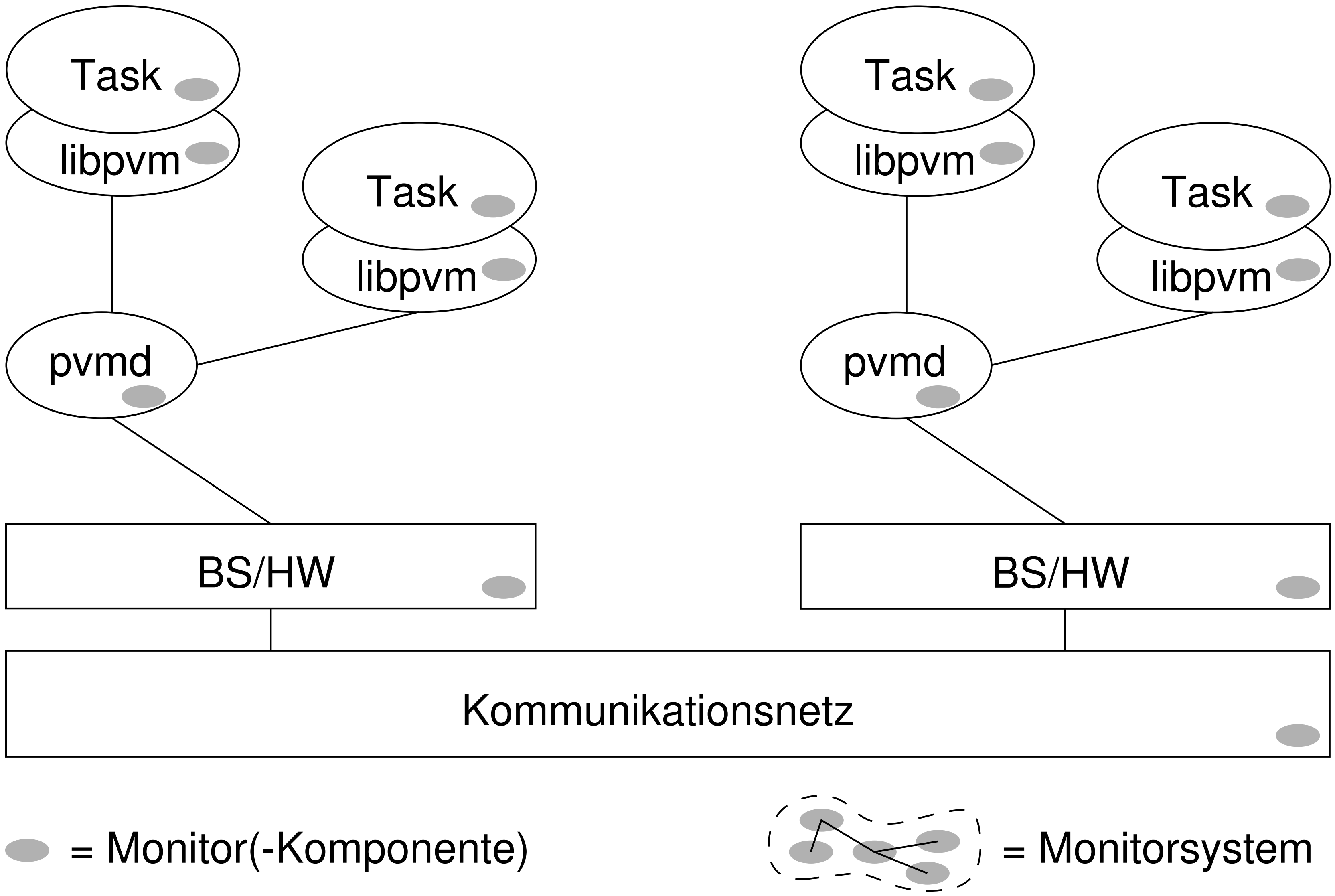
- Vektor-Zeitstempel: Ordnen aller Ereignisse



  - e_i nach e_j g.d.w. $vt(e_i) > vt(e_j)$
  - Problem: Dynamik $\Rightarrow$ Baum-Zeitstempel

Datenerfassung in parallelen Systemen

Typisches paralleles System (PVM):



Instrumentierung: Tasks

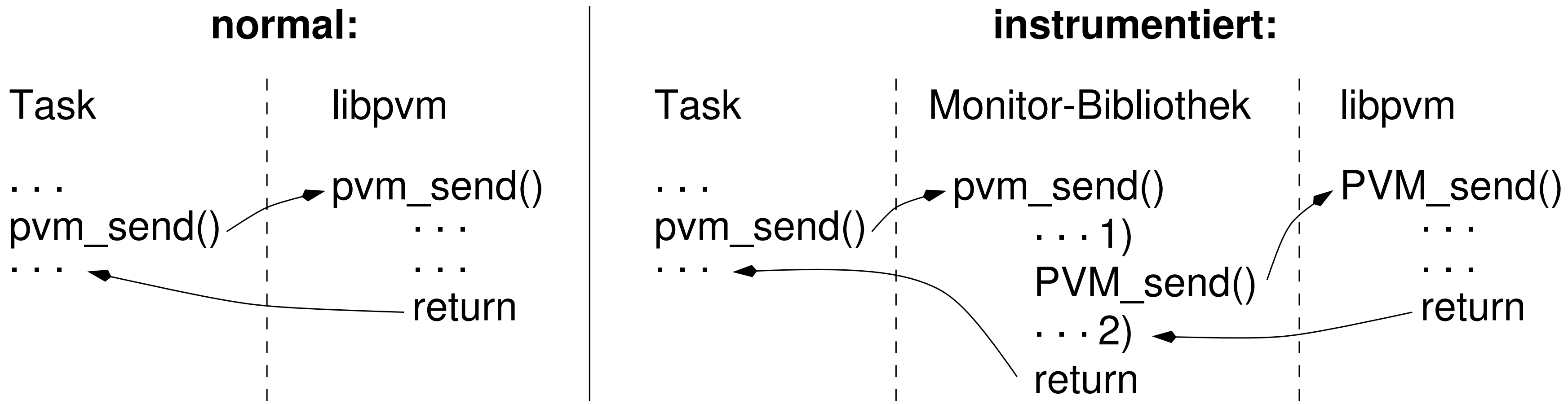
Instrumentierung von Tasks:

- Statische Instrumentierung des Quelltextes von Tasks
  - Sehr portabel und flexibel
  - Meist durch Präprozessor ⇒ Neuübersetzung nötig
- Ohne Quelltextänderung: Betriebssystem-Dienste
  - Prozeßzustand
    - Prozesser/Schedulingzustand: `/dev/kmem` oder `/proc`
    - Speicher/Register/Keller: `ptrace()` oder `/proc`
  - Ereignisse
    - Signale: Abfangen über `ptrace()` oder `/proc`
    - Andere: Einsetzen von Traps oder Sprüngen in den Maschinencode (zur Laufzeit, über `ptrace()` oder `/proc`)

Instrumentierung: Programmiermodell (1)

Instrumentierung des Programmiermodells (Bibliothek, Daemon):

- *Wrapping*-Funktionen



An den Stellen 1) und 2) sind *hooks* eingefügt:

```
if (mon_pvm_send0)
    mon_pvm_send0(...);
```

Der Funktionszeiger `mon_pvm_send0` wird vom Monitorsystem zur Laufzeit besetzt ⇒ flexible Auswertung möglich

Instrumentierung: Programmiermodell (2)

- Bewertung der *Wrapping*-Technik:
  - \+ Keine Modifikation der Bibliothek
    (jedoch Umbenennung der Funktionen nötig, außer bei MPI)
  - \+ Unabhängig von Implementierung
  - \+ Effizient
  - – Nur eingeschränkte Information erfaßbar

$\leadsto$ Detailliertere Information über *hooks* in der Bibliothek bzw. im Daemon

- – Modifikation der Programmierbibliothek
- – Abhängigkeit von Implementierung und Versionen

$\Rightarrow$ Monitoring-Schnittstelle

Instrumentierung: BS, HW, Netz

Betriebssystem, Hardware:

- „Standard"-Schnittstellen des Betriebssystems
  (`/dev/kmem, /proc, ptrace()`)
- Einige Information aus Performancegründen nicht detailliert erfaßbar, z.B. Speicherzugriffe, Scheduling, Cache-Verhalten

  ⇒ Bereitstellung vorverarbeiteter Daten

  ⇒ Hardware-Monitore (z.B. im Prozessor, im Speicherinterface)

Kommunikationsnetz:

- „Standardisierte" *Management Information Bases* in den Netzkomponenten
  - Nur statistische Information (Fehlerrate, Kollisionsrate, ...)
  - Schwer mit einzelnen Anwendungen korrelierbar

Beispiel: Datenerfassung in XPVM

Ereignisse:

- Start/Ende von PVM-Aufrufen
- Einige spezielle Ereignisse (z.B. Taskerzeugung)

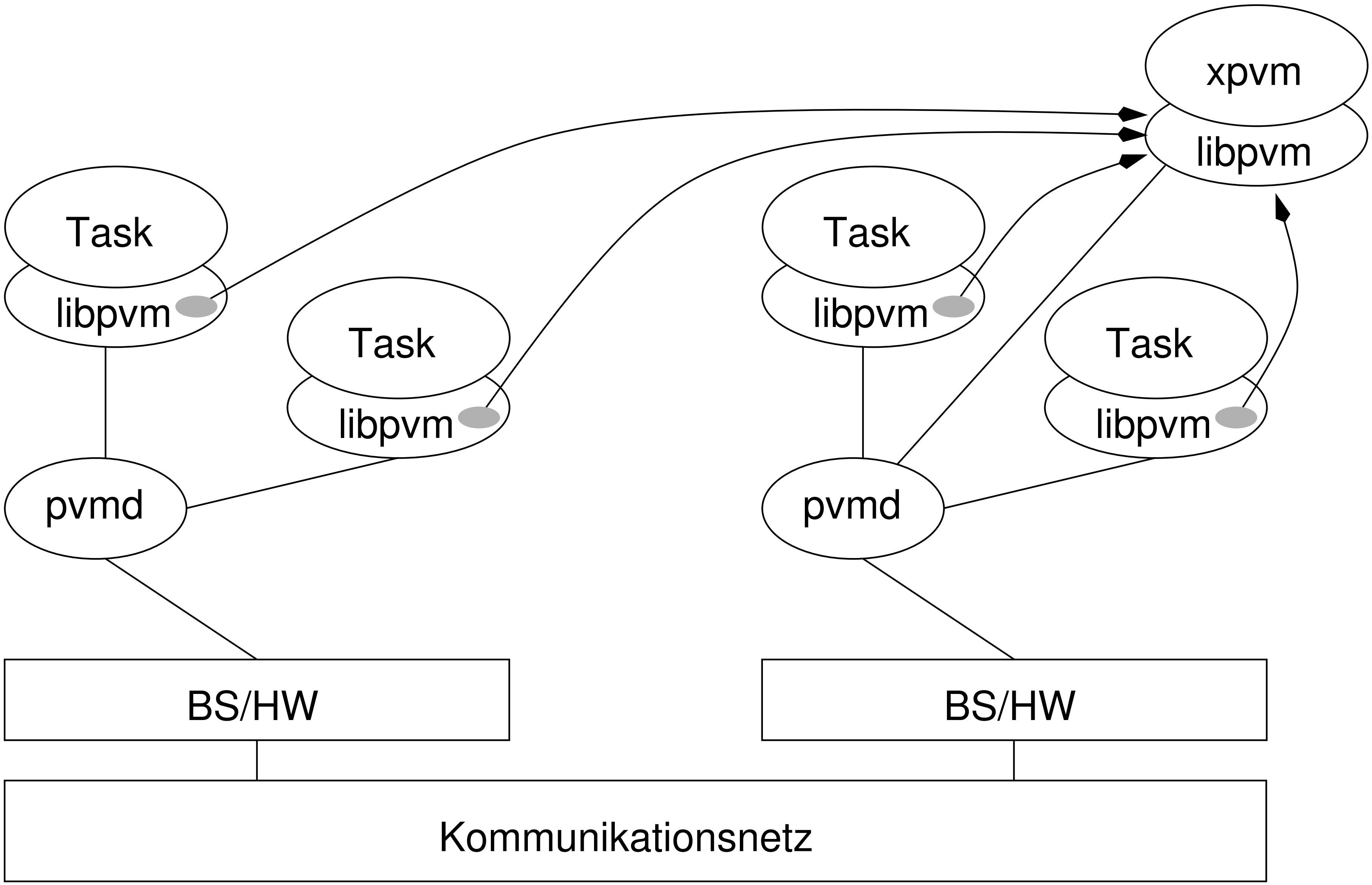
Instrumentierung: In PVM Bibliothek integriert

```
if (trace_this_kind_of_event) {
    pack time stamp and parameters into PVM message;
    send message to tracing task;
}
```

Bewertung:

+ Einfach, portabel
– Eingeschränkt: nur Ereignisse aus Programmierbibliothek
– Inflexibel: Ereignisse immer an zentrale Komponente gemeldet
– Keine knotenübergreifenden Funktionen

Datenerfassung in XPVM



Beispiel: OMIS (1)

OMIS: *On-line Monitoring Interface Specification*

Ziel: Universelle Monitor-Schnittstelle für Laufzeit-Werkzeuge

Basis: Ereignis/Aktions-Konzept

- Dienste des Monitorsystems aufgeteilt in Ereignisse und Aktionen
- Werkzeug kann Zuordnung frei programmieren:
  „Immer wenn ... passiert, führe ... aus“, z.B.
  - Task x beginnt zu senden: starte Timer y
  - Task x hört auf zu senden: stoppe Timer y
- Bewertung:
  - + Jede Monitorkomponente kann autonom auf Ereignisse reagieren ($\Rightarrow$ Performance)
  - + Flexibilität
  - – Implementierungsaufwand

Beispiel: OMIS (2)

Knotenübergreifende Ereignis/Aktions-Definitionen:

- z.B. globales Anhalten, Schreiben in zentrale Spurdatei
- Erweiterung des Ereignis/Aktions-Konzepts durch *Crosstrigger*
  - neues lokales Ereignis: Eintreffen eines *Crosstriggers*
  - neue lokale Aktion: Absenden eines *Crosstriggers*

  $\leadsto$ *Crosstrigger* realisierbar durch Nachricht
- Beispiel: globales Anhalten, wenn Prozeß x sendet

Knoten 0	Knoten 1
start_send(x): sendCT(1); stop()	CT(1): stop()

- in OMIS: automatisches Einfügen der *Crosstrigger*

  [0] start_send(x): [0,1] stop()

Datenerfassung bei OMIS