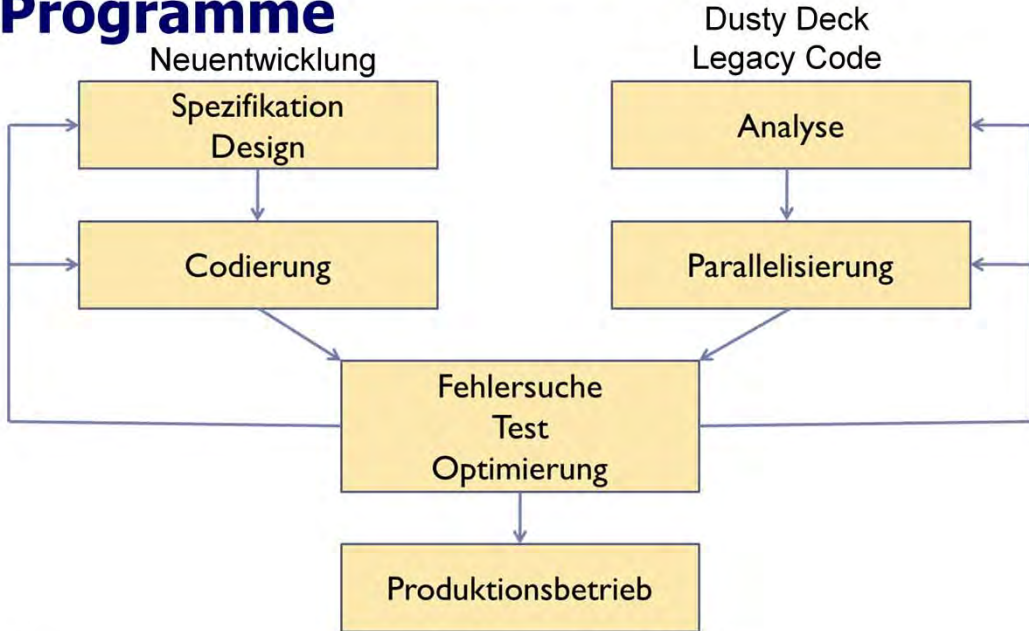




# Fehlersuche

1. Entwicklungszyklus paralleler Programme
2. Fehlersuche
3. Häufige Fehlerquellen
4. Problemstellungen
5. Werkzeugunterstützung
6. Offline-Werkzeuge
7. Laufzeit-Debugger
8. Konzepte paralleler Debugger
9. Deterministische Ablaufkontrolle

# 1. Entwicklungszyklus paralleler Programme





## 2. Fehlersuche (Debugging)

Aufspüren von Fehlerzuständen und die Beseitigung ihrer Ursachen

### 4 Schritte

- Test, Regressionstest
- Erkennen der Fehlerwirkung
- Schließen auf die Fehlerursache
- Beseitigen der Fehlerursache

# Debugging?

9/9

0800 Antam started  
 1000 " stopped - antam ✓  
 1300 (032) MP-MC  $\begin{cases} 1.2700 & 9.032847025 \\ 2.130476415 & 9.037846895 \end{cases}$  correct  
 (033) PRO 2  $\begin{cases} 2.130476415 \\ 2.130676415 \end{cases}$  correct

Relays 6-2 in 033 failed special speed test  
 in relay " " test.

Relay  
 2145  
 2170

1100 Relays changed  
 Started Cosine Tape (Sine check)  
 1525 Started Multi-Adder Test.

1545



Relay #70 Panel F  
 (moth) in relay.

First actual case of bug being found.  
 1630 Antam started.  
 1700 closed down.

# Beispiel

```
foo (a, b, x, &result);  
    /* Ursache: '&' vergessen  
       Compiler-Warning: pointer from  
                          integer without a cast */  
  
void foo (int a, int b, int *x,  
          int *result)  
{   *x = a+b;  
    /* segmentation violation */  
}
```

# 3. Häufigste Fehlerquellen

## Sequentielle Programmierung

- Schnittstellenprobleme (Typen, Zeiger auf Parameter, ...)
- Zeiger und dynamische Speicherverwaltung
- Logische und arithmetische Fehler

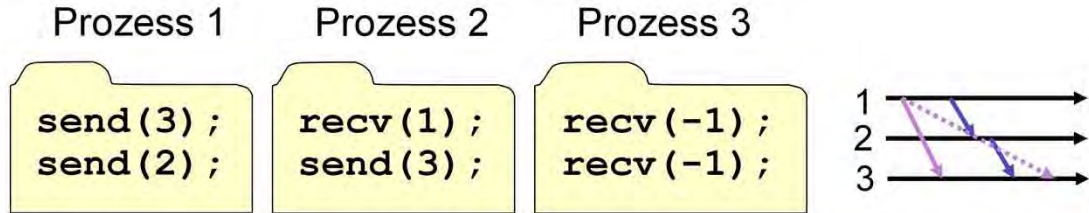
## Parallele Programme

- Kommunikationsfehler (Protokolle)
- Überholvorgänge (*races*)
- Verklemmungen (*deadlocks*)

# Häufigste Fehlerquelle: Überholvorgänge

Definition: Ein Überholvorgang entsteht durch unsynchronisierte, modifizierende Zugriffe auf gemeinsame Objekte (Adressbereiche, Nachrichtenpuffer)

Beispiel:



Konsequenz: Nichtdeterminismus,  
Nichtreproduzierbarkeit (schwer feststellbar)



# Häufigste Fehlerquelle: Verklemmung

Definition: Bei einer Verklemmung warten Prozesse blockierend auf Ereignisse anderer Prozesse, die auch blockiert sind

Beispiel:

Prozess 1

```
recv (... , 2 , ... ) ;  
send (... , 2 , ... ) ;
```

Prozess 2

```
recv (... , 1 , ... ) ;  
send (... , 1 , ... ) ;
```

Konsequenz: Programm bleibt hängen (leicht feststellbar)



## 4. Problemstellungen

Zusätzlich zu den normalen Problemen der Fehlersuche

- Erkennen einer Fehlerwirkung
- Suchen der Fehlerursache
  - Nichtreproduzierbarkeit der Fehlerwirkung
  - Nichtdeterminismus der Programmausführung
  - Ursache: Zeitabhängigkeit **nach** der Fehlerursache
- Unübersichtlichkeit: viele Prozesse
- Physische Verteiltheit
- Dynamik: Knoten- und Prozessmengen variieren potentiell



# Erkennen einer Fehlerwirkung

## Normalerweise

- Zustand des Programms entspricht nicht der Spezifikation (am Ende / mittendrin)
- Vergleich mit Testdaten

## Probleme bei parallelen Programmen

- Ergebnisse nicht nachrechenbar
- Ablauf abhängig von der Prozessorzahl
- Ablauf abhängig von zeitlichen Verhältnissen
- Häufig nicht bitidentisch nachrechenbar

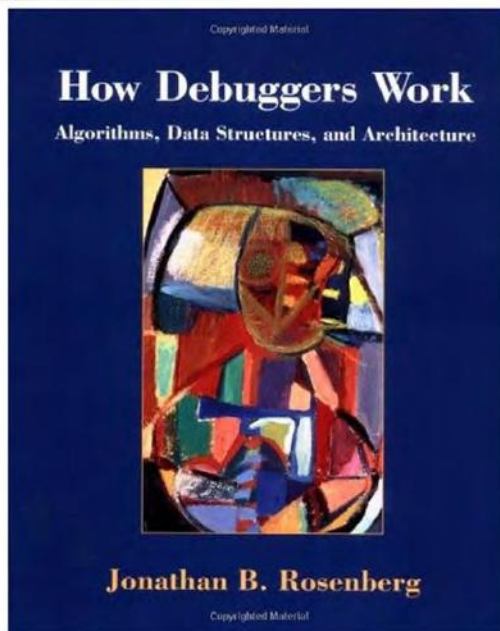
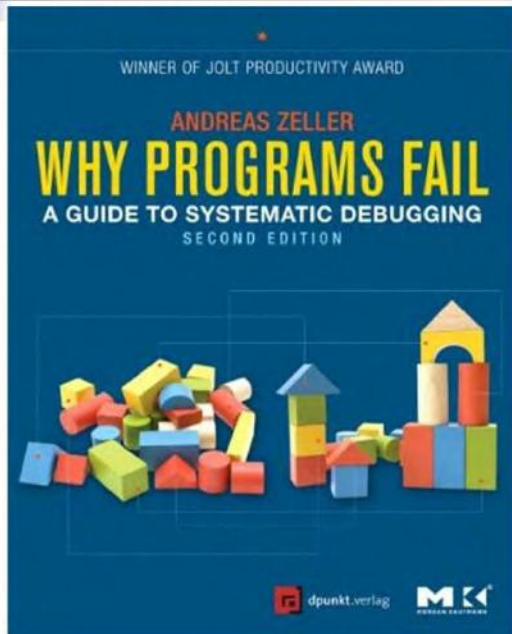
→ Falsche Berechnungen schwer erkennbar

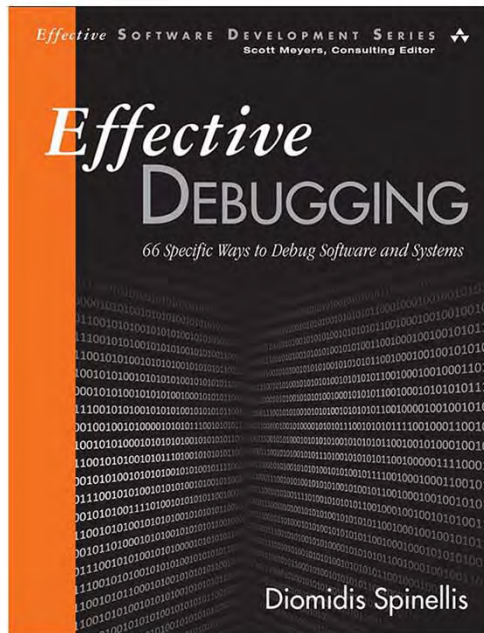
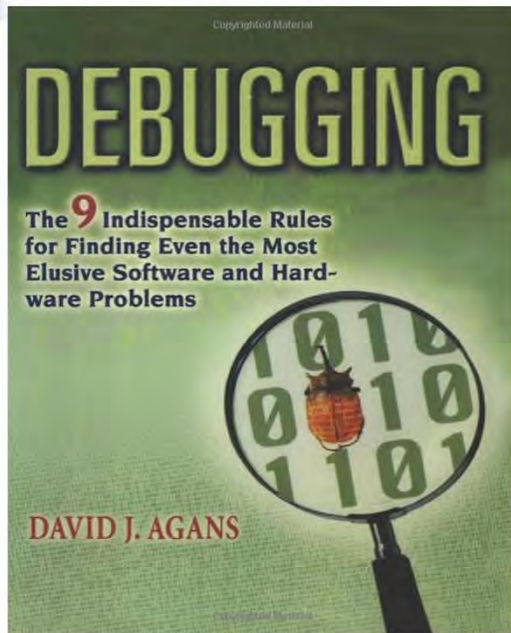


# Fehlertypen

---

- Heisenbug
  - Verschwindet, wenn man versucht, ihn zu suchen
- Bohrbug
  - Beständiger, zuverlässiger Fehler (eher selten)
- Mandelbug
  - Hohe Komplexität lässt ihn chaotisch erscheinen
- Schroedinbug
  - Taucht erstmals auf, nachdem jemand den Programmcode gelesen hat und feststellte, dass das nie hat laufen können. Danach läuft es auch nicht mehr.





O'REILLY\* Urheberrechtlich geschütztes Material



Weniger schlecht  
programmieren

Kathrin Passig &  
Johannes Jander

Urheberrechtlich geschütztes Material



## 5. Werkzeugunterstützung

- Statische Analyse
- printf() und WRITE
- Debugger
  - Spurbasierte Werkzeuge (offline)
  - Laufzeit-Werkzeuge (online)
- Ablaufkontrolle und Sicherungspunkte



## Analyse des Programmtextes vor/zur Übersetzungszeit

- Sequentielle Aspekte
  - Strikte Typ- und Parameterprüfung
  - Erweiterte semantische Tests
  - Einsatz spezieller Werkzeuge (siehe Liste)
  - Gute ANSI-C-Compiler, Option -Wall (alle Warnungen)
- Parallele Aspekte
  - Erkennen möglicher Überholvorgänge
  - Prüfung auf Verklemmungsfreiheit

= Forschungsthemen (bisher ungelöst)



# printf() und WRITE

**Die** Werkzeuge zur Fehlersuche schlechthin!

Bei parallelen Programmen aber:

- Zuordnung zu einzelnen Prozessen schwierig  
Zeichen-/zeilenweises Mischen möglich
- Bei Netzen: Umleitung in ein gemeinsames Fenster?!
- Sortierung oft unmöglich, da keine globale Zeit
- Korrekte kausale Ordnung der Ausgaben nicht gewährleistet



# Offline-/Online-Werkzeuge

Automatische Fehlerprüfung nach oder zur Laufzeit  
Sequentielle Aspekte

- Dynamische Speicherverwaltung

Parallele Aspekte

- Parameterprüfung bei Programmierbibliothek
- *Race*-Erkennung (Forschungsthema)

Vorbereitung der Anwendung (Alternativen)

- Präprozessor und Neuübersetzung
- Binden mit speziell instrumentierter Bibliothek
- Instrumentierung der Binärdatei

## 6. Spurbasierte Werkzeuge (offline)

### Merkmale

- Aufzeichnung relevanter Ereignisse des Programmlaufs
- Betrachtung der Spur durch „Browser“  
offline und auch near-online möglich
- Im Prinzip: automatisiertes `printf()`

### Aufgezeichnete Ereignisse

- Aufruf und Rückkehr der Funktionen der Programmierbibliothek  
Lokale Zeit, Dauer, Parameter
- Zum Teil auch benutzerdefinierte Ereignisse möglich

## Darstellungsarten

- Raum-Zeit-Diagramme, Gantt-Diagramme  
Darstellung einzelner Prozeßzustände  
Knoten- und/oder prozeßorientierte Darstellung  
Gut: globaler Überblick  
Schlecht: Globale Ordnung meist trügerisch
- Folge von Schnappschüssen  
Darstellung des globalen Zustands zu bestimmten Zeiten

X P V M 1.1.1 (PVM 3.3.11) [TID 0x40001]

Help: Move Pointer to Select Query, Double Click Button to Hold Info

Hosts...

Tasks...

Views...

Reset...

Quit

Halt

Help...



Time: 2.245225

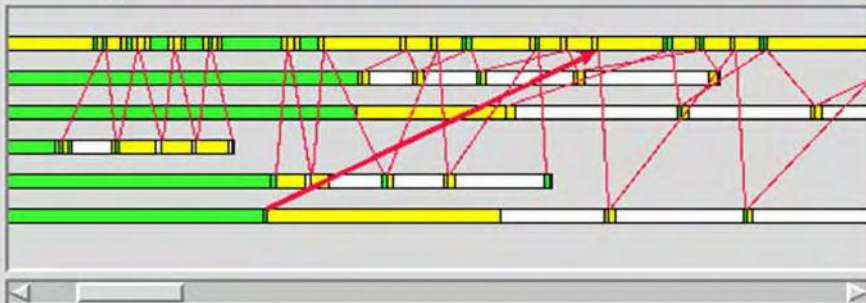
Trace File: /tmp/xpvm.trace.wismuell

PlayBack

OverWrite

Space-Time: Tasks vs. Time

atroland:hello2  
atroland:hello\_other2  
atroland:hello\_other2  
atroland:hello\_other2  
atroland:hello\_other2  
atroland:hello\_other2



Close



View Info: Message: from 40028 to 40023, 0.208900 to 0.320792, code=1 nbytes=32

Computing

Overhead

Waiting

Message





# Spurbasierte Werkzeuge...

## Steuerung

- VCR-ähnliche Elemente: Start, Stop, Vor, Zurück, Einzelschritt
- Meist Auswahl relevanter Knoten und Prozesse

## Vorbereitung

- Präprozessor und Neuübersetzung
- Binden mit instrumentierter Bibliothek
- Laufzeitoption der Programmierbibliothek

## Bewertung

- Für globalen Überblick und zur Überwachung der Kommunikation





## 7. Laufzeit-Debugger (online)

### Vorgehen

- Anhalten des Programms an interessanten Stellen
- Inspizieren des Programmzustandes
- Fortsetzen (oder Neustart) des Programms
- Schrittweise Programmabarbeitung

### Bei erkanntem Fehler

- Hypothese zur Fehlerursache
- Neustart des Programms und Überprüfen der Hypothese

## Typischer Funktionsumfang

- Anhalten des Programms  
Bedingt und/oder unbedingt
- Inspizieren des Programmzustandes  
Prozeduraufrufkeller, Parameter, Variablen
- Modifikation des Programmzustandes  
Setzen von Variablen, Veränderung des Codes(!)
- Ausführungskontrolle
  - Start und Stop
  - Einzelschritt (Anweisungen, Prozeduren)

# 8. Konzepte paralleler Debugger

## Eigenschaften paralleler Programme

- Mehrere Aktivitätsträger  
Prozesse, Threads; evtl. mehrere Binärformate
- Dynamik  
Zur Laufzeit Änderungen der Knoten, Prozesse, ...
- Interaktion  
Kommunikation und Synchronisation zwischen Prozessen
- Verteiltheit  
Verteilte Information; kein globaler Systemzustand

Berücksichtigung dieser Eigenschaften sehr unterschiedlich  
Kein Standard auf dem Gebiet in Sicht

# Umgang mit mehreren Prozessen/Threads

Zwei Methoden:

## Fenstertechnik und Prozessmengen

- Pro Prozess ein Fenster  
Unabhängiger sequentieller Debugger pro Fenster  
Leicht zu entwickeln; schwierige Benutzung bei vielen Prozessen  
=> (v.a.) für funktionsparallele Programme
- Ein einziges Fenster für alle Prozesse  
Auswahl eines Prozesses zur Fehlersuche  
Kommandos für Prozessmengen  
=> (v.a) für datenparallele Programme
- Mehrere Fenster für beliebige Teilmengen von Prozessen  
=> für beliebige Programme (DETOP)

Problem: Umgang mit höheren Prozessanzahlen

- Kommandos für Gruppen von Prozessen
- Zusammenfassen identischer Ergebnisse verschiedener Prozesse
- Einsatz geeigneter graphischer Darstellungen

Problem: Debugging dynamisch generierter Prozesse

- Stoppen aller neu erzeugten Prozesse; manuelle Auswahl

## Überwachung von Kommunikation und Synchronisation

- Möglich durch Haltepunkte auf Bibliotheksfunktionen
- Meist keine weitergehende Unterstützung, wie z.B.
  - Ausgabe wartender Prozesse
  - Status von Nachrichtenwarteschlangen
  - Haltepunkte auf Nachrichten
- Ausweg

Gleichzeitige Benutzung spurbasierter Werkzeuge (i.a. nur lesender Zugriff) und von Spezialwerkzeugen (message queue manager, mqm)





# Verteiltheit

---

Daten der Anwendung sind verteilt

- Spezialwerkzeuge liefern globale Sicht

Gemeinsame Daten verteilter Prozesse

- Beispiele: MPI-Gruppen, gemeinsame Speichersegmente
- Problem: Einfrieren des Zustandes bei Erreichen des Haltepunktes
- Meist nur Anhalten eines Prozesses unterstützt
- Wenn globales Anhalten unterstützt, dann nie sofort(!)  
=> Zustandsveränderungen sind möglich

Globale Ereigniserkennung (Forschungsthema)

- Verknüpfung von Ereignissen in verschiedenen Prozessen
- Z.B. Ereignisse a und b sind kausal abhängig/unabhängig



# Beispiel DETOP (1993)

The image displays two overlapping windows of the DETOP (1993) debugger interface.

**Top Window:**

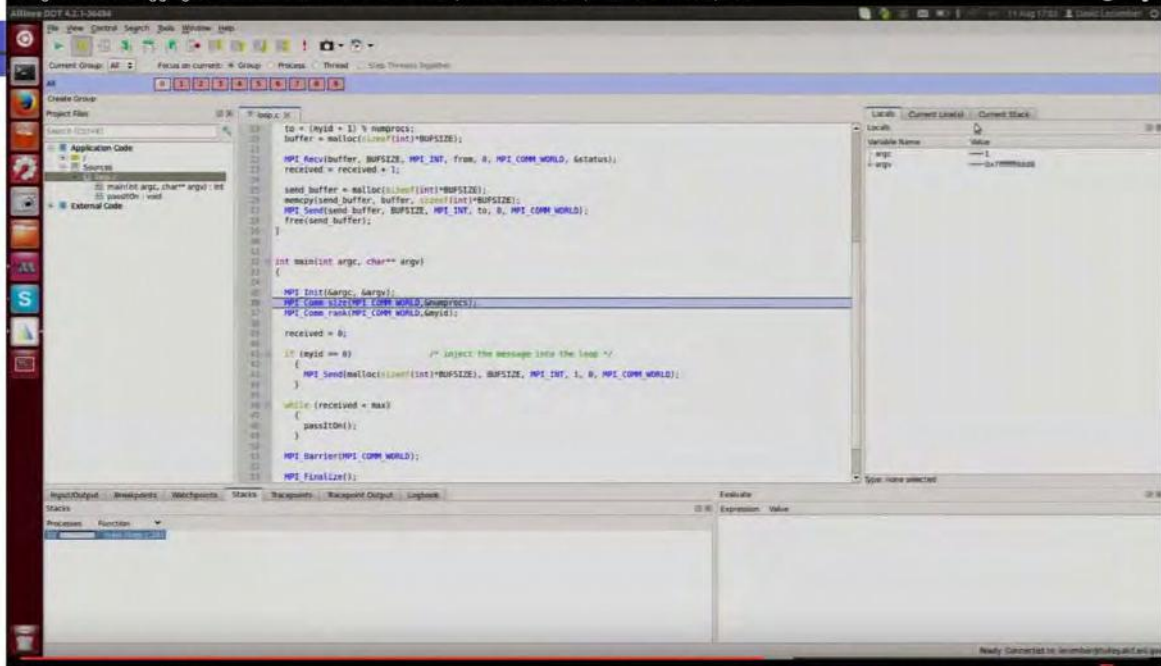
- File:** tst1.c
- Threads:** A list of threads including n0.c3.t1 (delivered), n3.c3.t1, n0.c3.t2, n3.c3.t2, n3.c3.t3, n3.c3.t4, n0.c3.t5, n2.c3.t1, n2.c3.t2, n2.c3.t3, n2.c3.t4, n2.c3.t5, n2.c3.t6, n2.c3.t7, and n2.c3.t8.
- State:** Last stop in call: ...
- State of thread n0.c3.t1:** ...

**Bottom Window:**

- File:** tst1.c
- Threads:** A list of threads including n1.c3.t1 (produce), n1.c3.t10 (produce), n1.c3.t2 (produce), n1.c3.t3 (produce), n1.c3.t4 (produce), n1.c3.t5 (produce), n1.c3.t6 (produce), n1.c3.t7 (produce), n1.c3.t8 (produce), and n1.c3.t9 (produce).
- Code:** Source code for tst1.c is displayed, showing a loop for (i = 0; i < 10; i++) { ... } and a call to produce(LinkCB\_t \* link).
- Breakpoints:** Breakpoint #2 at tst1.c:45 (enabled) and Breakpoint #1 at tst1.c:57 (enabled) are shown.
- State:** State of thread n1.c3.t1: STOPPED. Calls: produce at tst1.c:52, \_PX\_usrfunc\_harness (no line info), ParixMain (no line info).
- Output:** > print [n1.c3.t1] `tst1.c:produce` t. Value of `tst1.c:produce` buffer is: [n1.c3.t1] { Header = { Size = 6542804; DestProcId = 8160184; ReqId = 1395076144; MsgType = 808858937; SourceProcId = 6507976; Code = 659843436; Timeout = 33; } Body = array[1024] = ...

## The Distributed Debugging Tool (DDT)

*"DDT, the Distributed Debugging Tool is a comprehensive graphical debugger for scalar, multi-threaded and large-scale parallel applications that are written in C, C++ and Fortran."* Alinea



<https://www.youtube.com/watch?v=G5QCmvtwnik>

MPI startet offenbar eigene Threads

# Breakpoint im Ozeanmodell MPIOM

Erste ausführbare  
Zeile in Funktion

Standardausgabe  
und -fehler aller  
Prozesse

The screenshot shows the Allinea Distributed Debugging Tool interface. The main window displays the source code of the 'mo\_boundsexch' function. A breakpoint is set at line 102, 'ts = p\_time'. A dialog box indicates the process stopped at this breakpoint. The 'Input/Output' window shows the standard output and error for all processes. The 'Evaluate' window shows the current line of code and the values of variables 'p\_time' and 'ts'.

Current Line(s)

Variable Name	Value
p_time	0x9000000
ts	127384756

Input/Output

Currently Displaying: All

Process 1: tp10140\_mpiom\_timestep\_do\_not\_open\_for\_write  
Process 1: created average stream: file\_stream(name: "tp10140\_mpiom\_timestep\_do\_not", format: "m2", average:  
Process 1: tp10140\_mpiom\_monitoring\_monitor\_open\_for\_write  
Process 1: created average stream: file\_stream(name: "tp10140\_mpiom\_monitoring\_monitor", format: "m2", average:  
Process 1: MPI2IN: current date: 0000-01-01 00:00:00 begin of timestep (day): 1 (run): 1  
Process 1: MPI2: read feedback: 0000-01-01 from file:  
Process 1: MPI2IN: current date: 0000-01-01 00:00:00 begin of timestep (day): 2 (run): 2


Type here ('Enter') to send: More

Evaluate

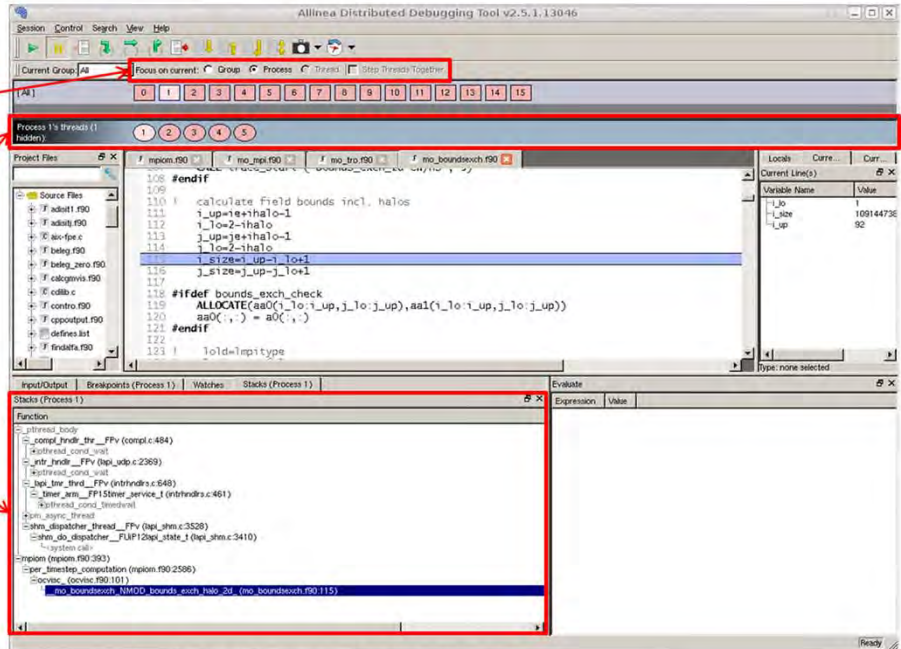
Expression	Value
type: none selected	

Werte von  
Ausdrücken in  
aktueller Zeile





- und  
sicht  
eln mit







## 8. Deterministische Ablaufkontrolle

### Problem des Nichtdeterminismus

- Erzwingen einer deterministischen Abarbeitungsreihenfolge der Kommunikation bei wiederholter Programmausführung  
Programm dadurch evtl. verlangsamt  
Varianten der Reihenfolgen systematisch testbar  
(Bei sequentiellen Programmen kein Problem!)

# Deterministische Ablaufkontrolle...

Funktionsweise eines Werkzeugs hierzu:

- Erster Programmlauf

Aufzeichnen der Reihenfolge des Eintreffens von Nachrichten bei Empfängern

- Weitere Programmläufe (*deterministic replay*)

Verwendung der Informationen aus dem ersten Programmlauf

Die Reihenfolge, wie Nachrichten beim Empfänger ankommen, wird gesteuert: zu früh eintreffende werden zurückgestellt

# Sicherungspunkte

## Problem der Zykluszeit

- Bei Fehler muss das Programm vom Anfang wiederholt werden, um nach der Fehlerursache zu suchen

## Vorgehensweise

- Zyklisches Erstellen von Sicherungspunkten
- Im Fehlerfall: Auswahl eines geeigneten Sicherungspunktes und Wiederanlauf des Programms von diesem Zeitpunkt aus.
- Evtl. gekoppelt mit Ablaufkontrolle

- Unterscheide Fehlerursache und Fehlerwirkung
- Typische Fehler paralleler Programme
  - Überholvorgänge, Verklemmungen
- Probleme
  - Nichtreproduzierbarkeit, Nichtdeterminismus
  - Unübersichtlichkeit, physische Verteiltheit, Dynamik
- Spurbasierte Werkzeuge für einen globalen Überblick und Prüfung der Kommunikation
- Haltepunktbasierte Debugger für Detailuntersuchungen
- Ablaufkontrolle beseitigt Nichtdeterminismus
- Sicherungspunkte verkürzen den Testzyklus



# Fehlersuche

## Die wichtigsten Fragen

- Welche Schritte umfasst die Fehlersuche?
- Was sind die häufigsten Fehlerquellen in Programmen?
- Was versteht man unter einer Verklemmung?
- Was versteht man unter einem Überholvorgang?
- Welche Problemstellungen gibt es bei parallelen Programmen?
- Welche Kategorien der Werkzeugunterstützung unterscheiden wir?
- Wie stellen spurbasierte Werkzeuge typischerweise ihre Informationen dar?
- Welche Funktionen bietet ein Laufzeit-Debugger?
- Was ist deterministische Ablaufkontrolle und wie funktioniert sie?
- Was ist hierbei der Sinn von Sicherungspunkten?