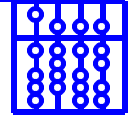


Synergetic Tool Environments

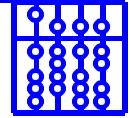
Thomas Ludwig, Jörg Trinitis, Roland Wismüller

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

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



- ↳ **Current Situation and Problems**
- ↳ **Methodical Tool Construction**
- ↳ **A Common Interface for Tools**
- ↳ **The Concept of Interoperability**
- ↳ **The Model of Interoperability**
- ↳ **Conflict Free Tools**
- ↳ **Tool Classification**
- ↳ **Debugging and Checkpointing**
- ↳ **Summary and Future Work**



Special problems of parallel programs

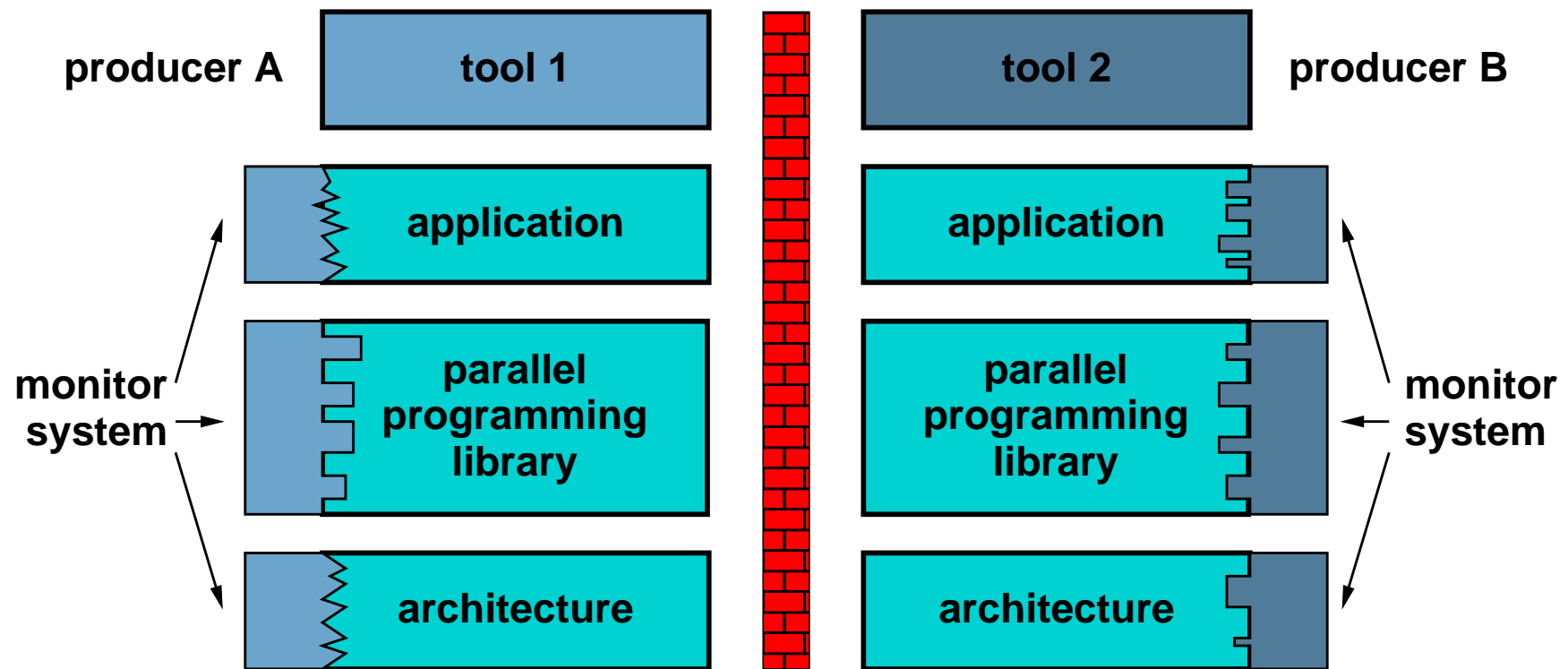
- ➔ non-determinism because of communication
- ➔ non-reproducibility because of time dependent behaviour
- ☞ Makes parallel programs difficult to develop

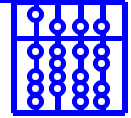
Necessary: on-line tools for direct program manipulation

- ➔ Debugging, performance analysis
- ➔ Deterministic program execution, computational steering
- ➔ Checkpointing mechanisms and load management and balancing

Nowadays tool construction principles:

- ➡ Hardly a theoretical basis (operating systems, distributed systems)
- ➡ Ad hoc construction with monolithic conception





Complicated tool construction

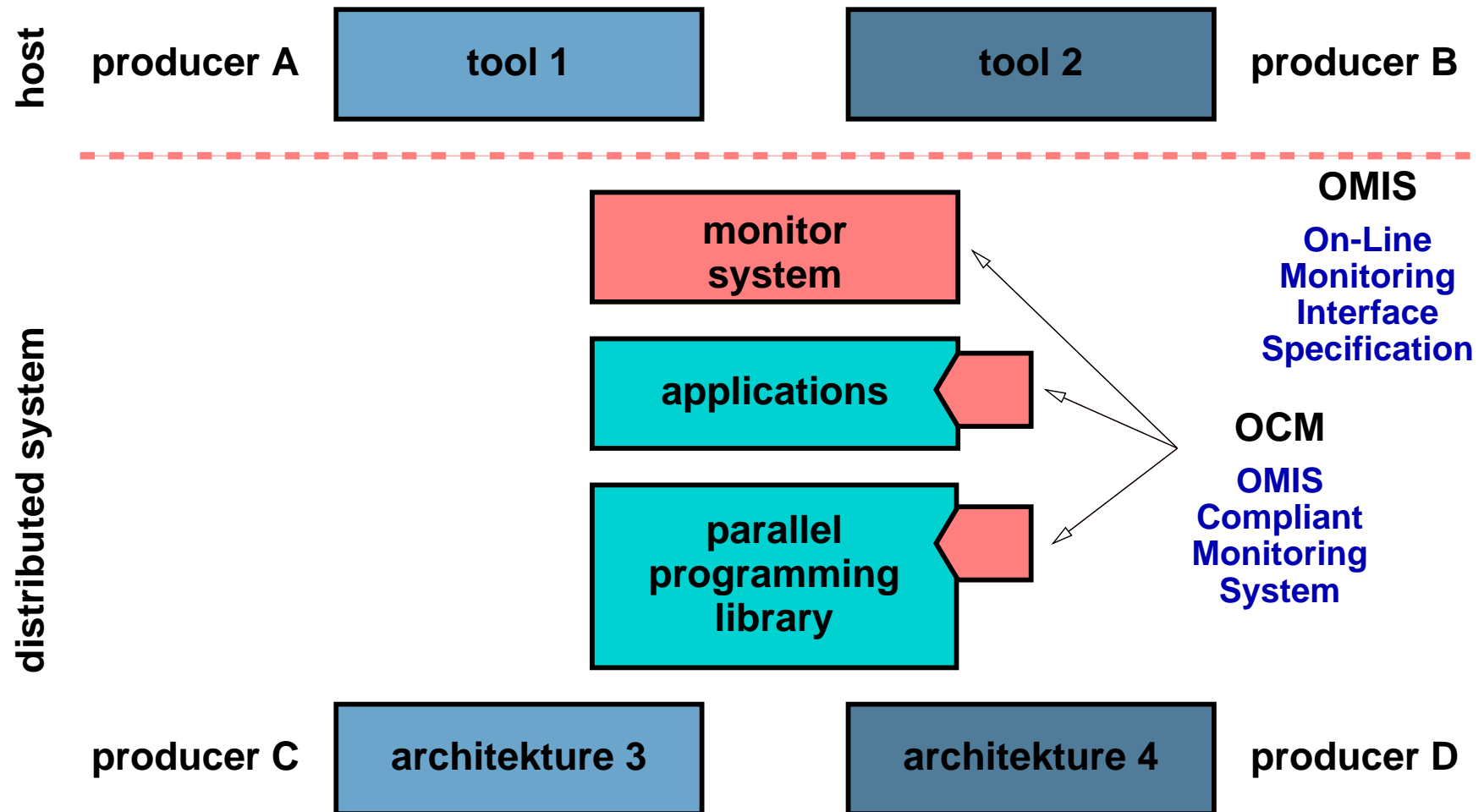
- ➔ Permanent re-development of already existing functionality
- ➔ Small selection available for users
- ➔ Development of parallel software not optimal supported

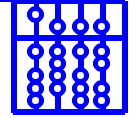
Poor tool concepts

- ➔ No cooperation between tools possible
Reason: Tools do not know of each other
- ➔ No concurrent usage supported
Reason: Tools use different infra structure components that are incompatible to each other

Methodical Tool Construction

Interface-based tool construction





Object classes

- System, node, process, thread, message buffer, message, monitor object

Service classes

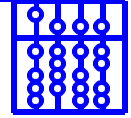
- Services for information, manipulation and events

Working principle: event/action model

- **Event** = change of state in the programm system
- **Action** = information or manipulation service requests

Requests

- Behaviour of the monitoring system is controlled by event/action-relations
- Relations are programmed via the interface



Syntax of the interface function

Omis_reply

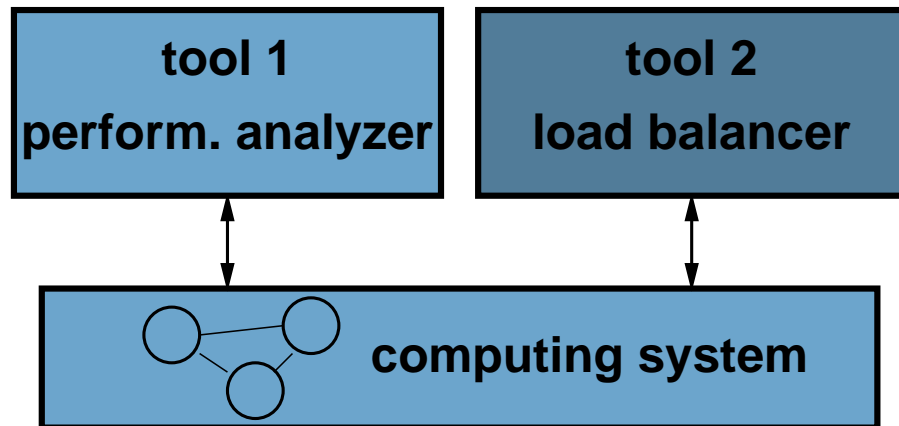
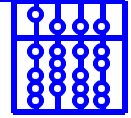
```
omis_request( char * request ,
              void (* callback)(Omis_reply reply, void *param),
              void *param,
              Omis_flags flags )
```

Syntax of event/action-relations

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

Semantic details

- ➔ Services work on single objects and object sets
- ➔ Services accept object references on any hierarchy level
- ➔ Actions can be performed concurrently

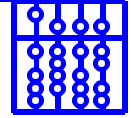


Prerequisite

- ➔ Uniform interface for all tools
- ➔ Shared monitoring system

Problem

- ➔ More than one tool knows the state of object o
- ➔ At least one tool modifies the state of O
- ➔ Question: what about the other tools?



Starting point: object view of the system

Question: who accesses objects when in which way?

Access modes: reading, writing

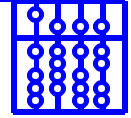
Access conflict

☞ Incompatible overlapping access of two tools to a shared object where at least one access modifies the object

Variants: RW, WR, WW

Possible consequences of access conflicts

- ➔ Program crash
- ➔ Wrong information in the tool
- ➔ Crash of tool session

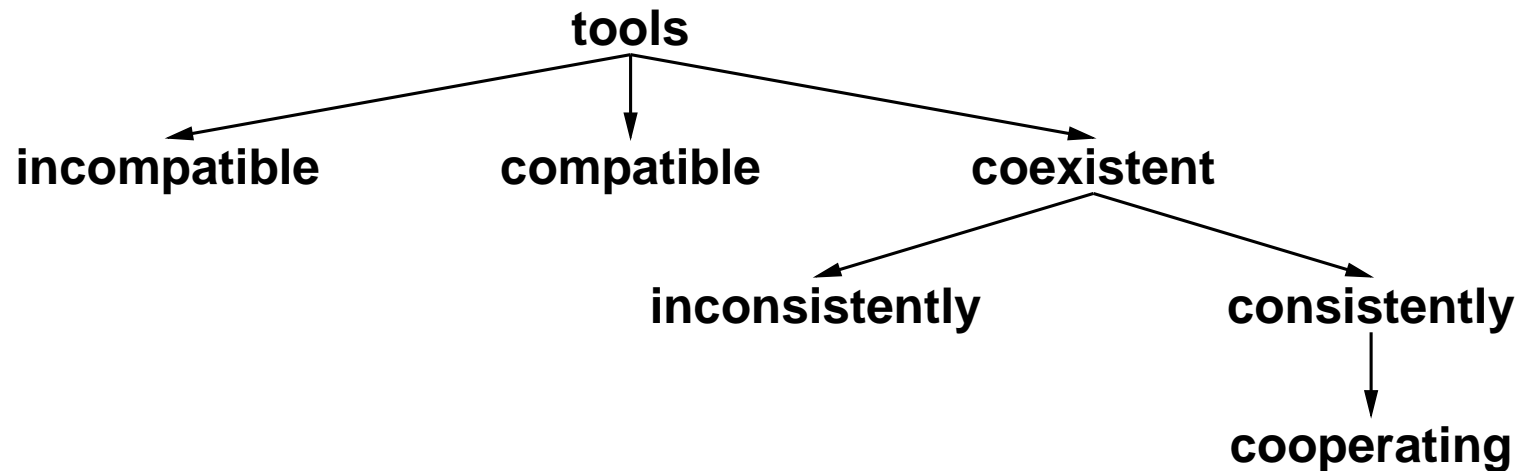
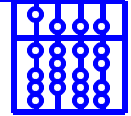


Consistency problem

- ➔ Tool A reads an object that will be modified by tool B later on
Situations: RW und WW
- ➔ Solution: Tool asks for notification when the state of an object gets modified

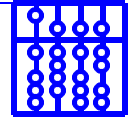
Transparency problem

- ➔ Tool B reads an object that was modified by tool A
Situations: WR und WW
- ➔ Solution: not completely fixed currently
Transaction mechanisms and locks: high overhead
Catch accesses from other tools (modification hiding)



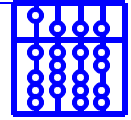
Example: Performance Analysis and load balancing

- 👉 Inconsistently coexisting: performance analyzer gets confused by the appearing and disappearing processes
- 👉 Consistently coexisting: performance analyzer recognizes migrations and visualizes them if the user wants that. Otherwise it hides them.
- 👉 Cooperating: performance analyzer requests load management; load management requests performance analyzer to visualize certain load indices



Synergy

- ➡ Shorter debugging cycle possible: checkpoint and restore debuggee
- ➡ Conditional checkpoints
- ➡ Predicate selected checkpoints
- ➡ Checking hypotheses against checkpoints
- ➡ Reverse execution
- ➡ Undoing debugger actions
- ➡ Interrupting debugging sessions



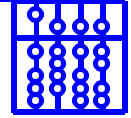
Internal problems and solutions

- ☞ Restore modifies application state
 - Problem: consistency
 - debugger must be notified

- ☞ Checkpointer uses an internal checkpointing and consistency mechanism

- ☞ Even checkpointing might modify application state
 - Problem: transparency
 - side effects must be hidden
 - clever implementation (atomic services, signal handler)
 - access interception

- ☞ Remaining problem: non-deterministic applications
 - future: code integration

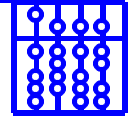


Tool construction

- ➔ Methodical tool construction can be based on a common interface
- ➔ A standard monitor implementation can be used by different developer groups
- ➔ Object classes and services can be expanded

Interoperability

- ➔ Interoperability requires a common infrastructure for the monitoring system
- ➔ Concurrent object access is the key issue with interoperability
- ➔ Interoperability can be divided into several classes, cooperation being the most complex one
- ➔ Issue is also recognized in other projects: DPCL (IBM), DAMS (Univ. Lisbon, Portugal)



Interoperable tools

- ➔ Debugger, performance analyser and visualizer (beta)
- ➔ Debugger and checkpointing system (beta)
- ➔ Integration of computational steering and deterministic execution control

Adaptations

- ➔ Adaptations to new architectures: distributed memory systems
- ➔ Adaptations to CORBA environments