DETOP – An Interactive Debugger for PowerPC Based Multicomputers

Roland Wismüller
Technische Universität München
Institut für Informatik

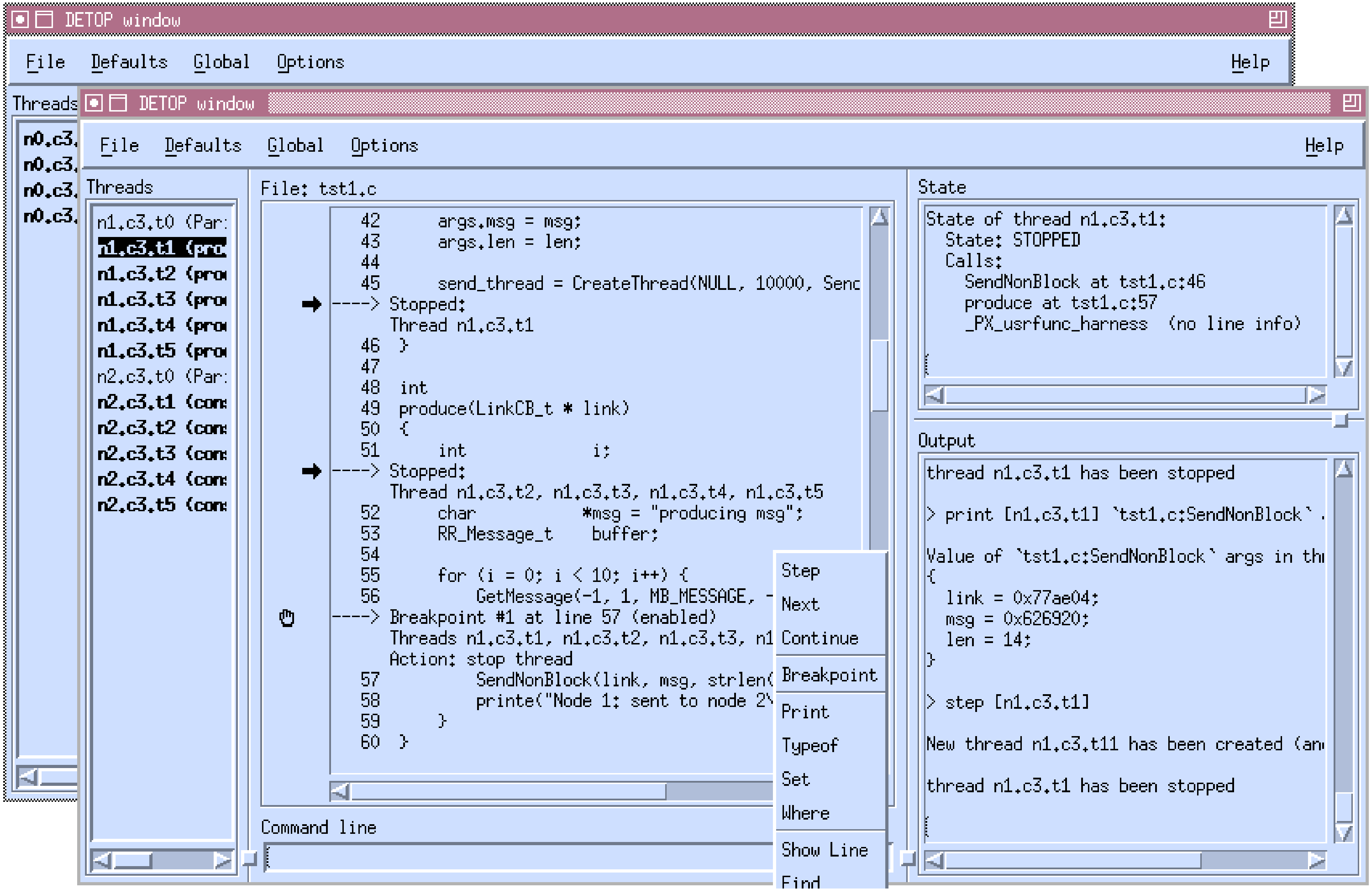
18.05.1995

Requirements for Run-Time Tools

1. ease of use
   - ⇝ graphical interface + command line
   - ⇝ global view and detailed views
2. general usability
   - ⇝ programming languages
   - ⇝ programming models: multi-threaded, dynamic
3. minimal intrusion
   - ⇝ overhead in time (probe effect)
   - ⇝ overhead in space (no virtual memory)
4. scalability
   - ⇝ data collection and analysis
   - ⇝ data representation

The Parallel Debugger DETOP

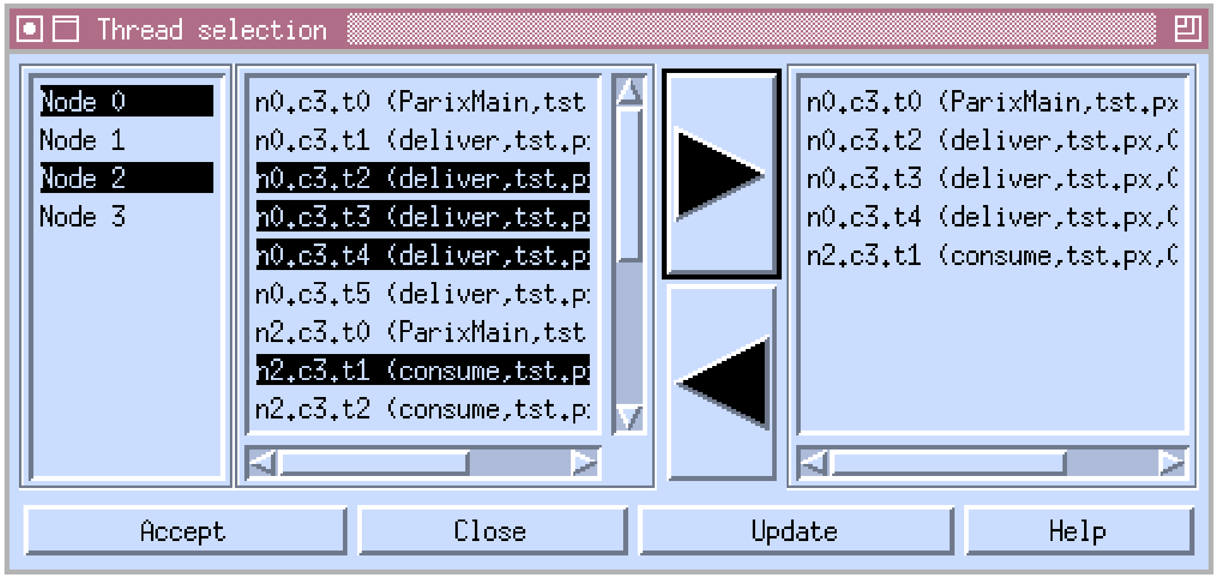
- Developed by TUM in cooperation with Parsytec
- Targets: PowerXplorer and GC/PowerPlus
- Highlights:
  - ⇝ easy-to-use graphical interface based on Motif
  - ⇝ multiple debugger windows with multiple threads per window
  - ⇝ C and Fortran 77
  - ⇝ full support for PARIX (dynamic creation of threads and contexts)
  - ⇝ provides the most important debugger commands
    - execution control, inspect/modify program state, breakpoints
      - ⇝ applied to a set of selected threads
    - configuration and on-line help
  - ⇝ based on event/action paradigm


DETOP window
File Defaults Global Options Help
Threads
File: tst1.c
42 args.msg = msg;
43 args.len = len;
44
45 send_thread = CreateThread(NULL, 10000, Senc
----> Stopped:
Thread n1.c3.t1
46 }
47
48 int
49 produce(LinkCB_t * link)
50 {
51 int i;
----> Stopped:
Thread n1.c3.t2, n1.c3.t3, n1.c3.t4, n1.c3.t5
52 char *msg = "producing msg";
53 RR_Message_t buffer;
54
55 for (i = 0; i < 10; i++) {
56 GetMessage(-1, 1, MB_MESSAGE, -
----> Breakpoint #1 at line 57 (enabled)
Threads n1.c3.t1, n1.c3.t2, n1.c3.t3, n1
Action: stop thread
57 SendNonBlock(link, msg, strlen(
58 printf("Node 1: sent to node 2\
59 }
60 }
Step
Next
Continue
Breakpoint
Print
Typeof
Set
Where
Show Line
Find
Command line
State
State of thread n1.c3.t1:
State: STOPPED
Calls:
SendNonBlock at tst1.c:46
produce at tst1.c:57
_PX_usrfunc_harness (no line info)
Output
thread n1.c3.t1 has been stopped
> print [n1.c3.t1] `tst1.c:SendNonBlock`
Value of `tst1.c:SendNonBlock` args in thi
{
link = 0x77ae04;
msg = 0x626920;
len = 14;
}
> step [n1.c3.t1]
New thread n1.c3.t11 has been created (an
thread n1.c3.t1 has been stopped
n1.c3.t0 (Par:
n1.c3.t1 (pro
n1.c3.t2 (pro
n1.c3.t3 (pro
n1.c3.t4 (pro
n1.c3.t5 (pro
n2.c3.t0 (Par:
n2.c3.t1 (con
n2.c3.t2 (con
n2.c3.t3 (con
n2.c3.t4 (con
n2.c3.t5 (con

Debugging Dynamic Parallel Programs

Identification of threads:

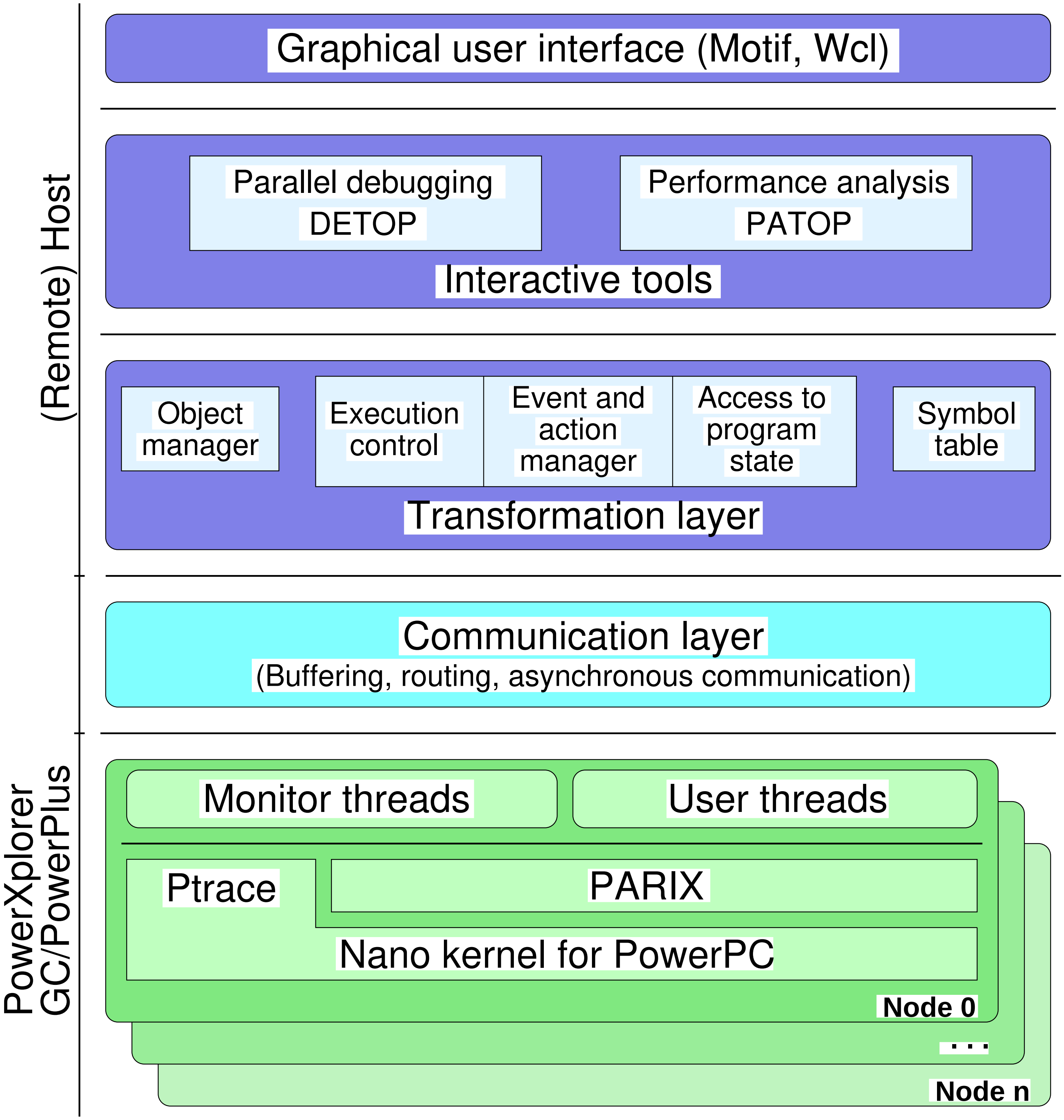
- global identifier (*n0.c4.t5*)
- name of root function
- name of executable
- thread pointer



Debugging of dynamic threads:

- debugger option: stop all dynamically created threads
- thread creation event: user can define actions to be performed
- threads specified by pattern:
  - ⇝ matching threads are automatically added to debugger window
  - ⇝ breakpoints for threads that do not yet exist

Debugger Architecture



Debugger Interface

- Based on X-Windows and Motif

  $\leadsto$ Widget Creation Library (Wcl) for rapid prototyping

- Object-oriented design

  Most important class: “list of text objects”

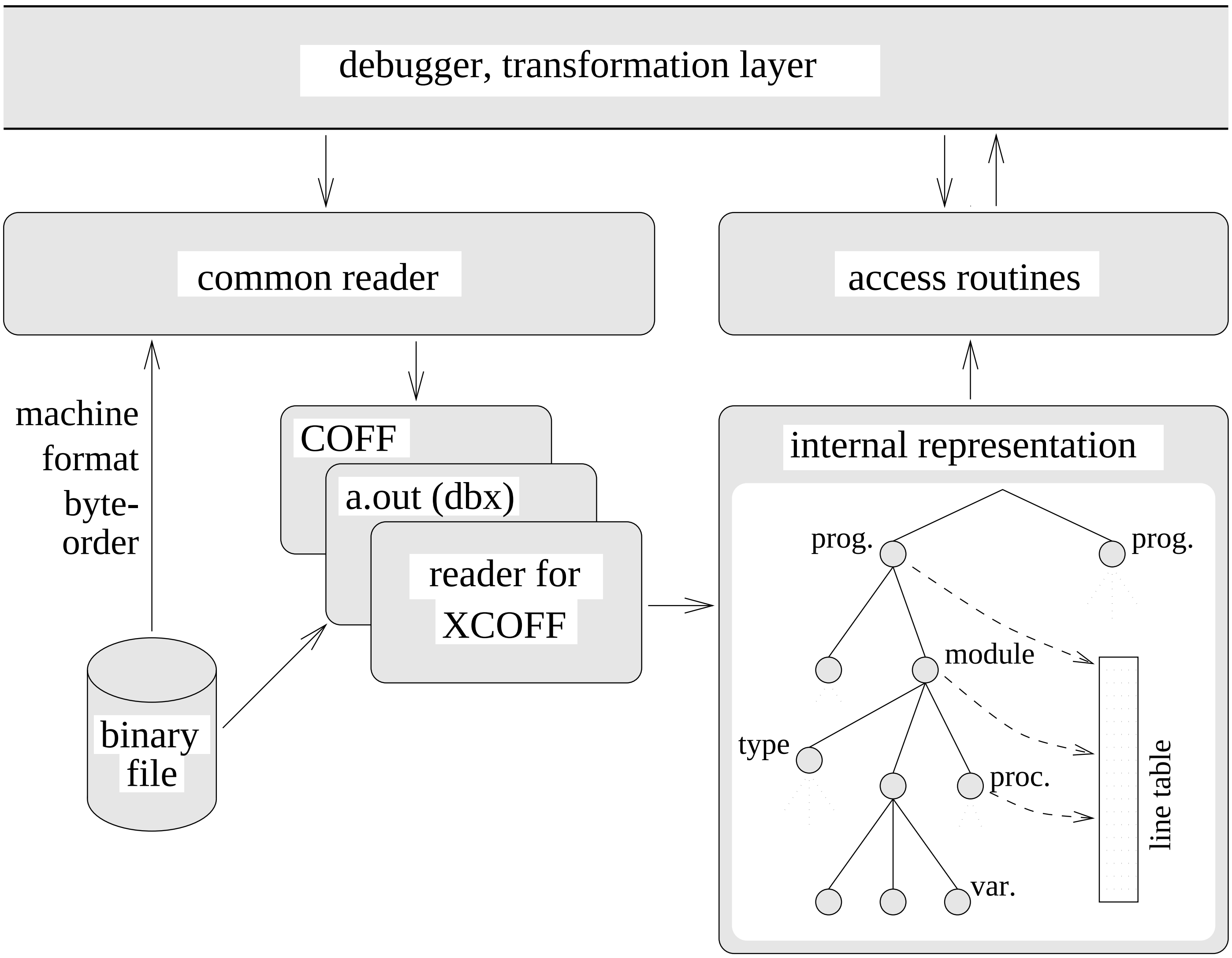
  $\leadsto$ used for source code listing, output window, and state display

  $\leadsto$ provides selection mechanism and context sensitive popup-menus

  $\Rightarrow$ functions offered by menu change according to the *meaning* of the selected item

  $\leadsto$ information can be updated in-place, e.g. in the state display

Symbol Table Management



Access routines behave like a pattern matcher:

root × *name* × *class* × *subclass* × *search_type*

$\longrightarrow$ *set of symbol descriptors*

$\leadsto$ determine set of symbols with given properties

$\leadsto$ search for a specific symbol

Communication layer

Communication between debugger and monitor via TCP/IP sockets

$\leadsto$ full remote debugging

Message buffering

$\leadsto$ pack multiple requests into a single message, if possible

Asynchronous communication and evaluation

$\leadsto$ user interface is not blocked while commands execute

$\leadsto$ commands for different nodes can be executed concurrently

$\leadsto$ evaluation of results is based on data-flow graphs and callback functions

- data buffering, parameter passing between callbacks
- callbacks with multiple sources of input (modes: AND, OR)
- dynamic and static graphs for single requests and actions, respectively

Monitoring System

- one monitor on each node, plus communication between monitors
- operates at machine level (no symbol tables on target system)
- based on a programmable interface and event/action scheme:
  - pipeline of commands

    $\{$view[] $\{$inspect[?]$\}$ $\}$ $\leadsto$ all threads on node and their state
  - global commands

    ga$\{$view[] $\{$combine[?]$\}$ $\}$ $\leadsto$ list of all threads, combine results
  - events and actions

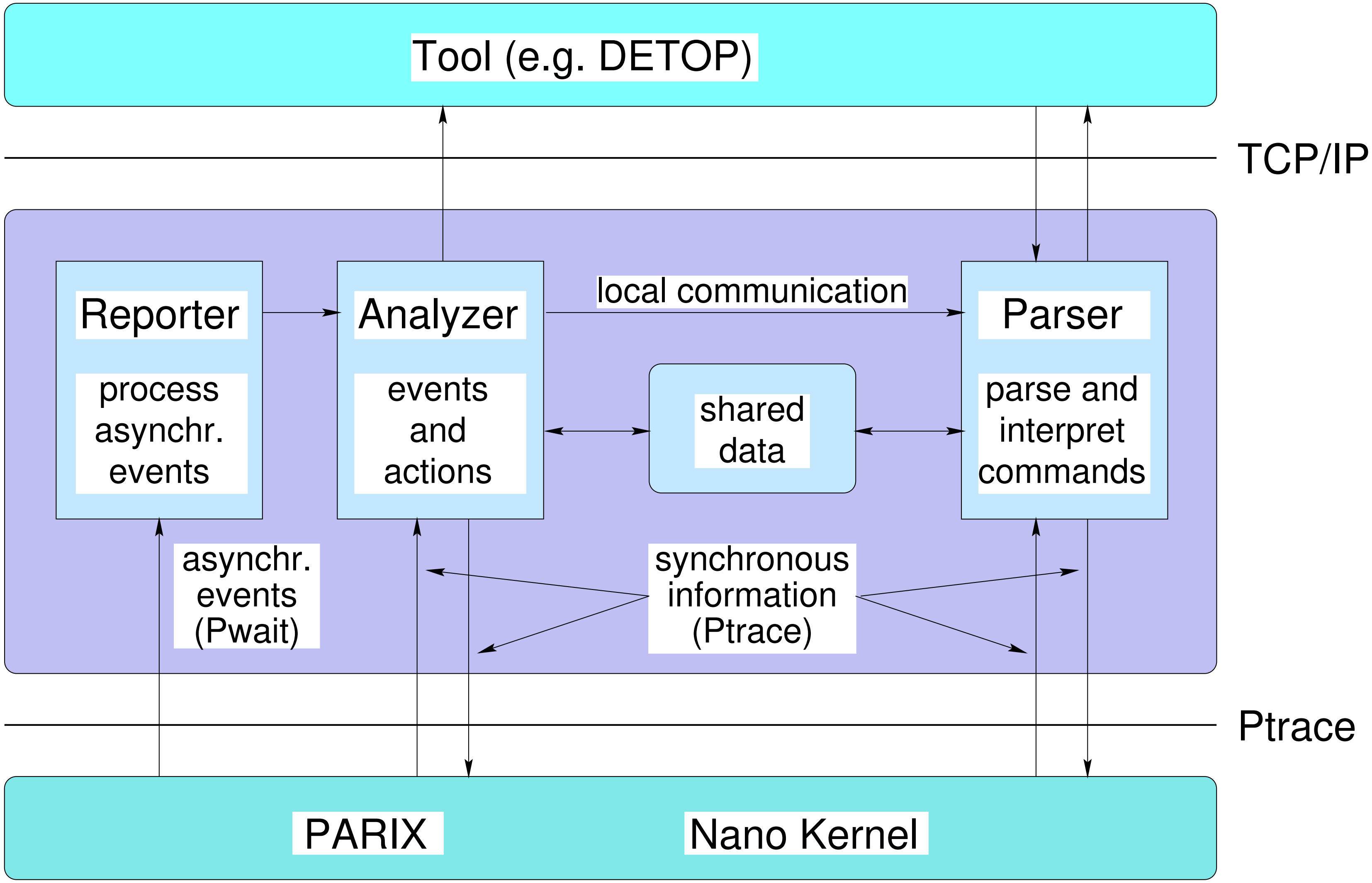
    $\{$exec_event[... ,4]$\}$ $\leadsto$ define event # 4

    4$\{$stop[] inspect[?]$\}$ $\leadsto$ stop all threads on node, determine state of thread that triggered the event

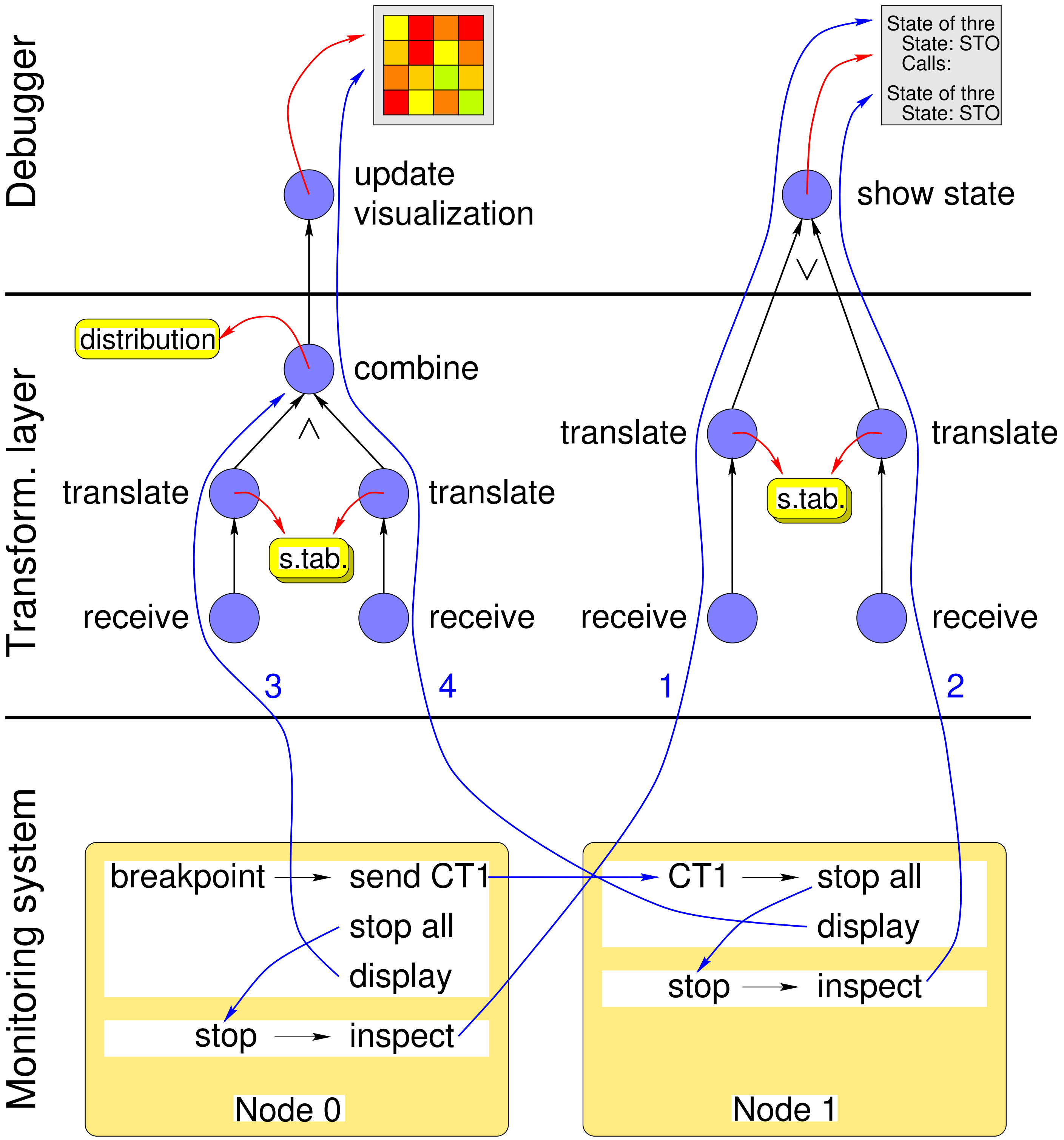
  $\leadsto$ every monitor command can be used as an action

  $\leadsto$ crosstrigger signals for global events and remote actions

Monitor Architecture



An Example


Debugger
Transform. layer
Monitoring system
State of thre
State: STO
Calls:
State of thre
State: STO
update
visualization
show state
distribution
combine
translate
translate
translate
translate
s.tab.
s.tab.
receive
receive
receive
receive
3
4
1
2
breakpoint
send CT1
stop all
display
stop
inspect
Node 0
CT1
stop all
display
stop
inspect
Node 1

Conclusion

- DETOP is easy to use
  - \+ context sensitive menus + optional command line
  - \+ multiple windows for different views
  - – visualizers
- it supports a very large class of programs
  - \+ C and Fortran , PARIX
  - \+ data and functional parallelism, dynamic parallelism
  - – C++, other programming models (PVM, MPI)
- it causes only minimal intrusion
  - \+ small, but powerful monitoring system
  - \+ programmable monitor interface, event/action scheme
  - – deterministic (re-)execution
- it is usable for large systems
  - \+ distributed event recognition, data acquisition and preprocessing
  - \+ multiple threads per window, dynamic selection of threads
  - – more comprehensive presentation of data

Future Work

- debugging of thread communication (using special breakpoints), distributed breakpoints
- full support of event/action scheme at the level of the user interface
- visualization of program state and data
- support for C++
- future versions of PowerXplorer, GC/PowerPlus
- adaption to PVM
  - on top of PARIX
  - for networks of workstations (as part of THE TOOL-SET for PVM)