KOJAK – A Tool Set for Automatic Performance Analysis of Parallel Programs

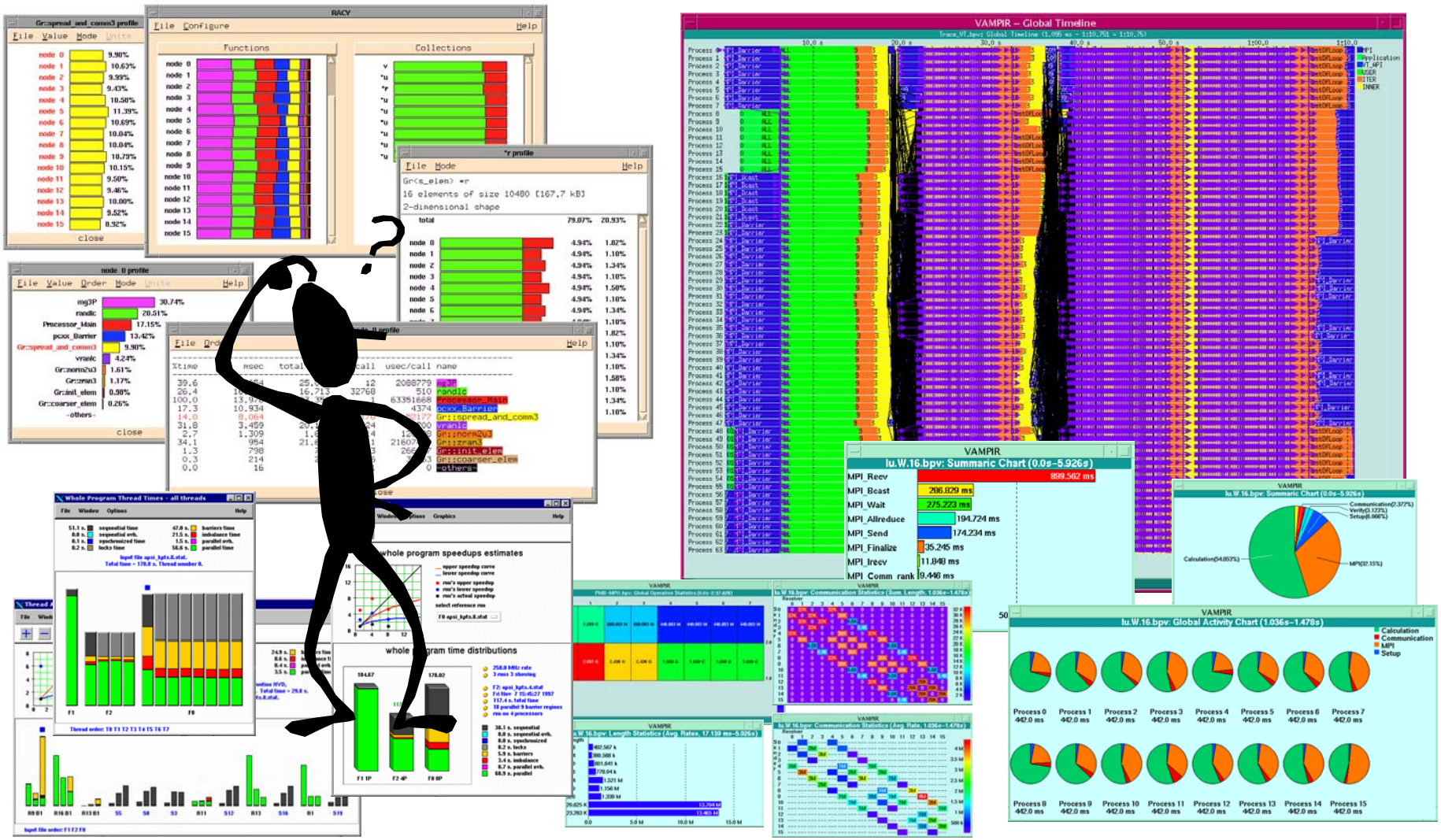
Felix Wolf

University of Tennessee
Computer Science Department
Innovative Computing Laboratory (ICL)
Knoxville, Tennessee
wolf@icl.edu

Bernd Mohr

Forschungszentrum Jülich
Zentralinstitut für Angewandte Mathematik
John von Neumann - Institut für Computing (NIC)
Jülich, Germany
b.mohr@fz-juelich.de

Motivation

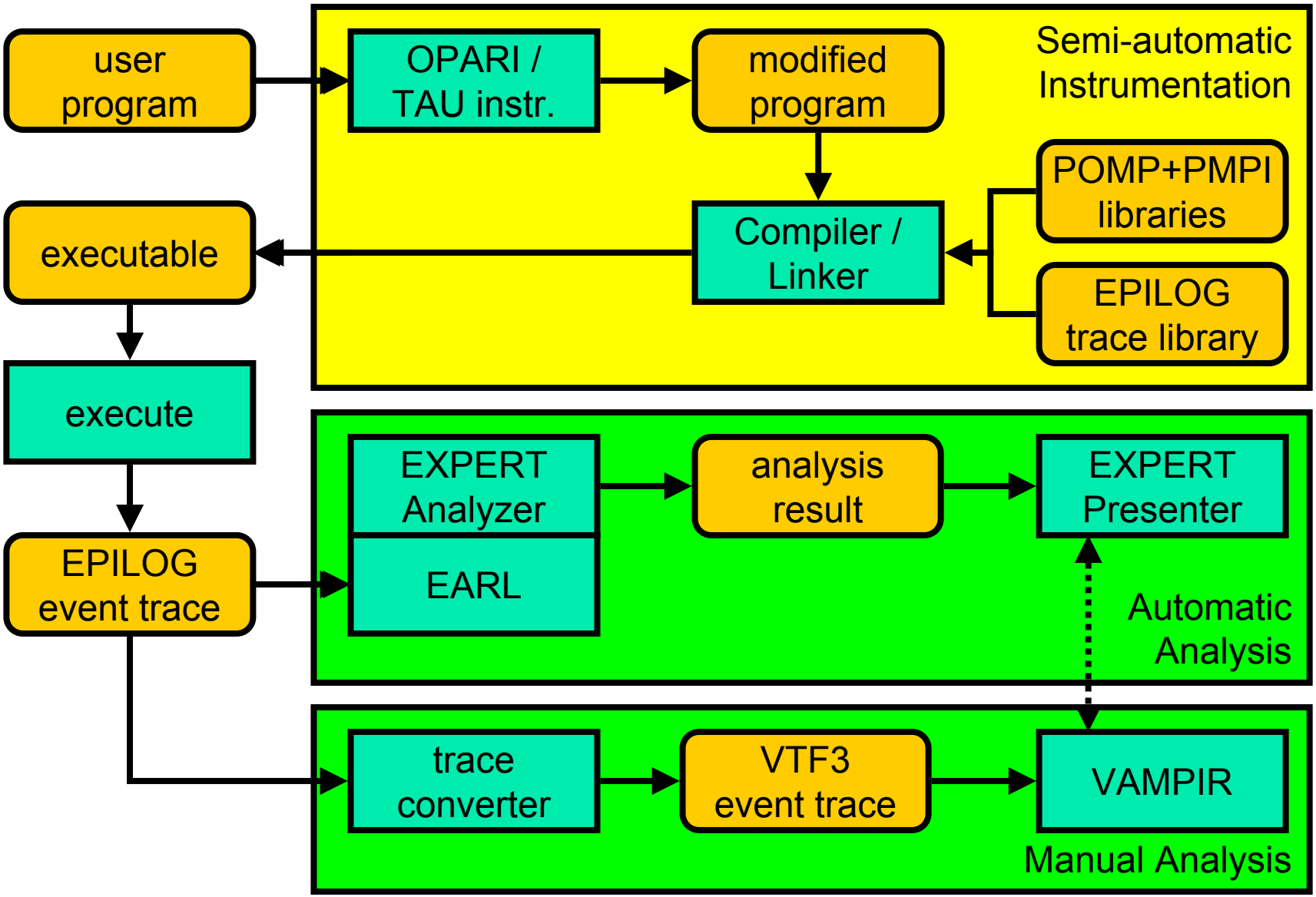

RACY
File Configure Help
Functions
Collections
node 0 profile
File Value Order Mode Help
mg3P
randlc
Processor_Main
pcxx_Barrier
Gr::spread_and_comm3
vranlc
Gr::norm2u3
Gr::zran3
Gr::init_elem
Gr::coarser_elem
-others-
close
VAMPIR – Global Timeline
VAMPIR
lu.W.16.bpv: Summaric Chart (0.0s–5.926s)
MPI_Recv 899.562 ms
MPI_Bcast 286.029 ms
MPI_Wait 275.223 ms
MPI_Allreduce 194.724 ms
MPI_Send 174.234 ms
MPI_Finalize 35.245 ms
MPI_Irecv 11.848 ms
MPI_Comm_rank 9.446 ms
VAMPIR
lu.W.16.bpv: Global Activity Chart (1.036s–1.478s)
Calculation
Communication
MPI
Setup
Process 0 442.0 ms
Process 1 442.0 ms
Process 2 442.0 ms
Process 3 442.0 ms
Process 4 442.0 ms
Process 5 442.0 ms
Process 6 442.0 ms
Process 7 442.0 ms
Process 8 442.0 ms
Process 9 442.0 ms
Process 10 442.0 ms
Process 11 442.0 ms
Process 12 442.0 ms
Process 13 442.0 ms
Process 14 442.0 ms
Process 15 442.0 ms
whole program speedups estimates
whole program time distributions

The KOJAK Project

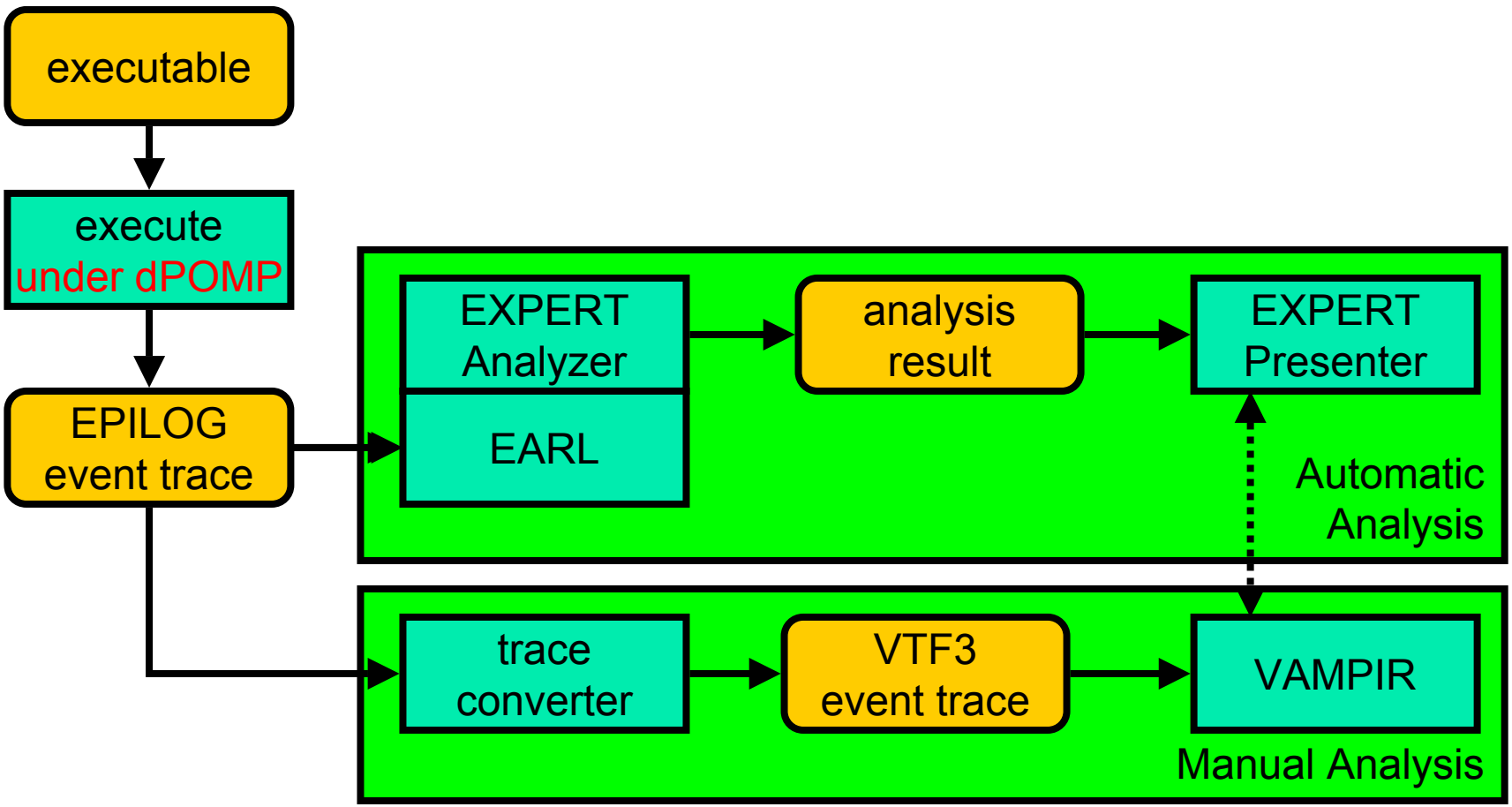
- Kit for Objective Judgement and Automatic Knowledge-based detection of bottlenecks

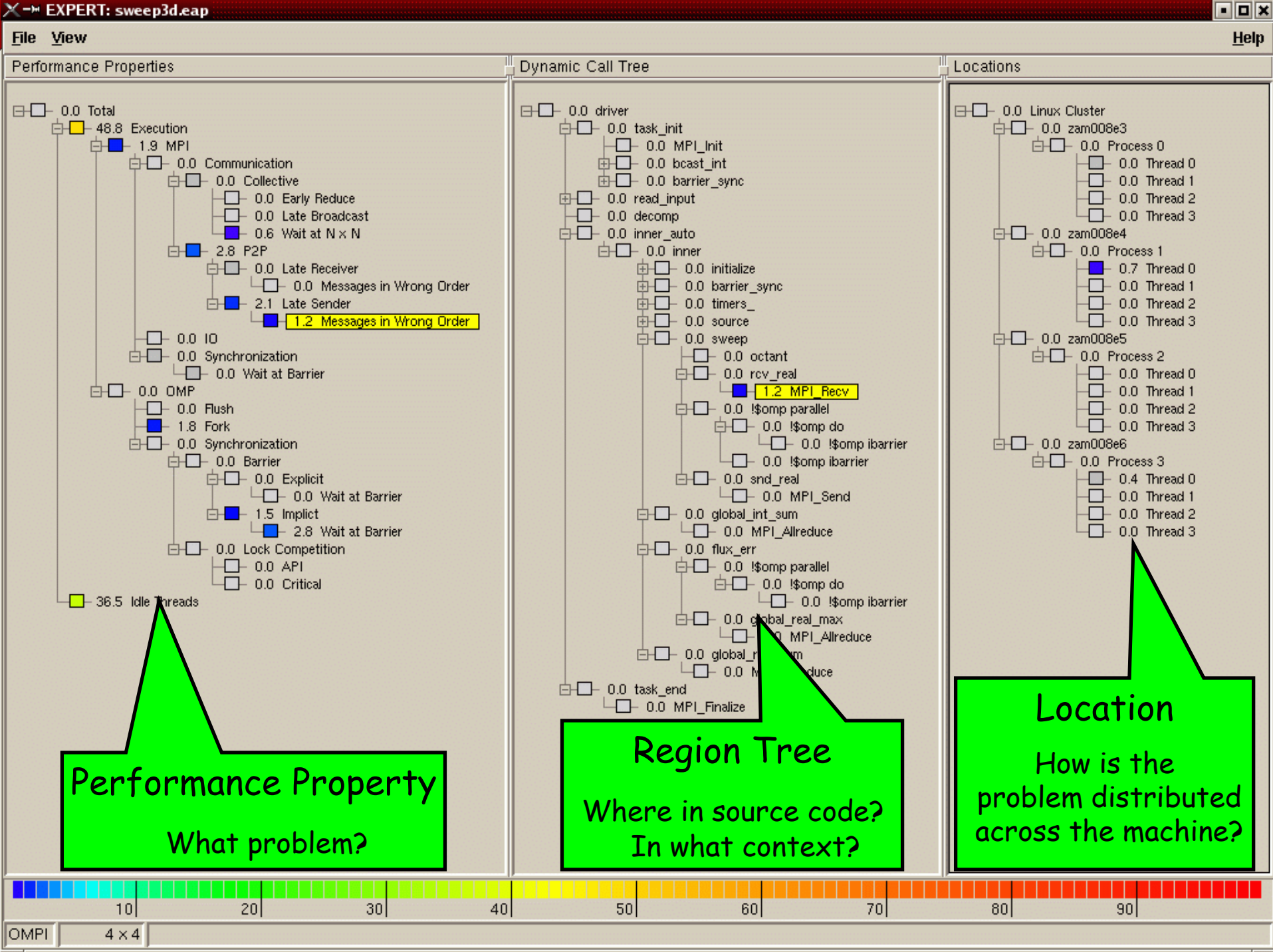
- Long-term goals
  - Design and Implementation of a Portable, Generic, and Automatic Performance Analysis Environment
- Current focus
  - Event-based Specification of Performance Properties
  - MPI, OpenMP, Hybrid (OpenMP + MPI) programming model
  - Parallel computers with SMP nodes
  - Development of research prototypes

KOJAK Architecture



KOJAK Architecture on IBM for OpenMP




EXPERT: sweep3d.eap
File View Help
Performance Properties
Dynamic Call Tree
Locations
0.0 Total
48.8 Execution
1.9 MPI
0.0 Communication
0.0 Collective
0.0 Early Reduce
0.0 Late Broadcast
0.6 Wait at N x N
2.8 P2P
0.0 Late Receiver
0.0 Messages in Wrong Order
2.1 Late Sender
1.2 Messages in Wrong Order
0.0 IO
0.0 Synchronization
0.0 Wait at Barrier
0.0 OMP
0.0 Flush
1.8 Fork
0.0 Synchronization
0.0 Barrier
0.0 Explicit
0.0 Wait at Barrier
1.5 Implict
2.8 Wait at Barrier
0.0 Lock Competition
0.0 API
0.0 Critical
36.5 Idle Threads
0.0 driver
0.0 task_init
0.0 MPI_Init
0.0 bcast_int
0.0 barrier_sync
0.0 read_input
0.0 decomp
0.0 inner_auto
0.0 inner
0.0 initialize
0.0 barrier_sync
0.0 timers_
0.0 source
0.0 sweep
0.0 octant
0.0 rcv_real
1.2 MPI_Recv
0.0 !$omp parallel
0.0 !$omp do
0.0 !$omp ibarrier
0.0 !$omp ibarrier
0.0 snd_real
0.0 MPI_Send
0.0 global_int_sum
0.0 MPI_Allreduce
0.0 flux_err
0.0 !$omp parallel
0.0 !$omp do
0.0 !$omp ibarrier
0.0 task_end
0.0 MPI_Finalize
0.0 Linux Cluster
0.0 zam008e3
0.0 Process 0
0.0 Thread 0
0.0 Thread 1
0.0 Thread 2
0.0 Thread 3
0.0 zam008e4
0.0 Process 1
0.7 Thread 0
0.0 Thread 1
0.0 Thread 2
0.0 Thread 3
0.0 zam008e5
0.0 Process 2
0.0 Thread 0
0.0 Thread 1
0.0 Thread 2
0.0 Thread 3
0.0 zam008e6
0.0 Process 3
0.4 Thread 0
0.0 Thread 1
0.0 Thread 2
0.0 Thread 3
Performance Property
What problem?
Region Tree
Where in source code?
In what context?
Location
How is the problem distributed across the machine?
10
20
30
40
50
60
70
80
90
OMPI
4 x 4

Usage on JUMP

Step 0: One-time Initialization of the KOJAK Tool Enviroment

```
% module load kojak vampir
```

Step 1: Semi-Automatic Instrumentation of the Application

- Manually instrument program with init/begin/end POMP directives
- Prepend all compile and link commands with kinst-pomp

Step 2. Run instrumented executable

Step 3. Analyzing the Measured Event Trace

```
% cl-analyzer a.elg
```

Step 4. Viewing the Analyzer Results

```
% Presenter a.eap
```

Step 5. Convert Event Trace to VAMPIR Format. [OPTIONAL]

```
% elg2vtf3 a.elg ; vampir a.vpt
```

Manual Instrumentation

- Insert once as the first executable line of the main program

Fortran:	C/C++:
!$POMP INST INIT	#pragma pomp inst init

- Also, at least the main program/function has to be instrumented
  - INST BEGIN/END directive to mark any user-defined regions
  - All but last exit have to be instrumented by INST ALTEND

Fortran:	C/C++:
!$POMP INST BEGIN(name) ... [!$POMP INST ALTEND(name)] ... !$POMP INST END(name)	#pragma pomp inst begin(name) ... [#pragma pomp inst altend(name)] ... #pragma pomp inst end(name)