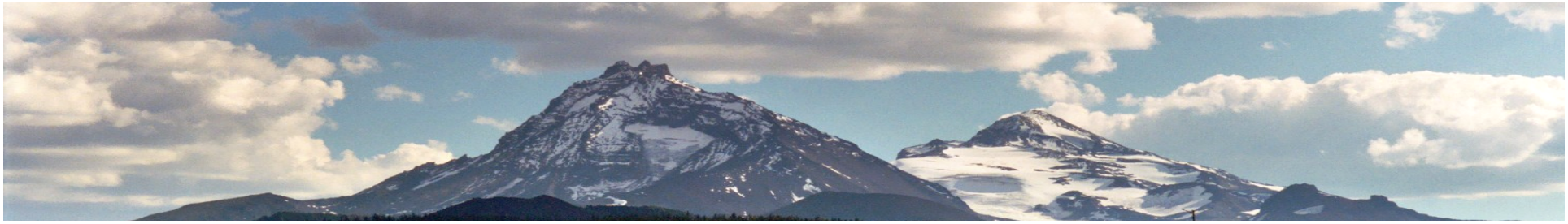


Lecture 5:
Parallel Tools Landscape – Part 2

Allen D. Malony
Department of Computer and Information Science

UNIVERSITY OF OREGON

Performance and Debugging Tools

Performance Measurement and Analysis:

- ✓ ~~Open|SpeedShop~~
- ✓ ~~HPCToolkit~~
- ✓ ~~Vampir~~
- ✓ ~~Scalasca~~
- ❍ Periscope
- ❍ mpiP
- ❍ Paraver
- ❍ PerfExpert

Modeling and prediction

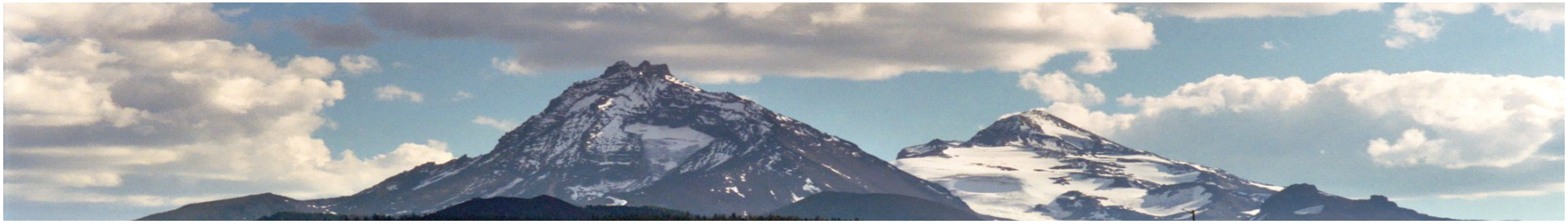
- ❍ Prophesy
- ❍ MuMMI

Autotuning Frameworks

- ❍ Active Harmony
- ❍ Orio and Pbound

Debugging

- ❍ Stat



Periscope

Michael Gerndt

Technische Universität München (Germany)

http://www.lrr.in.tum.de/~periscope

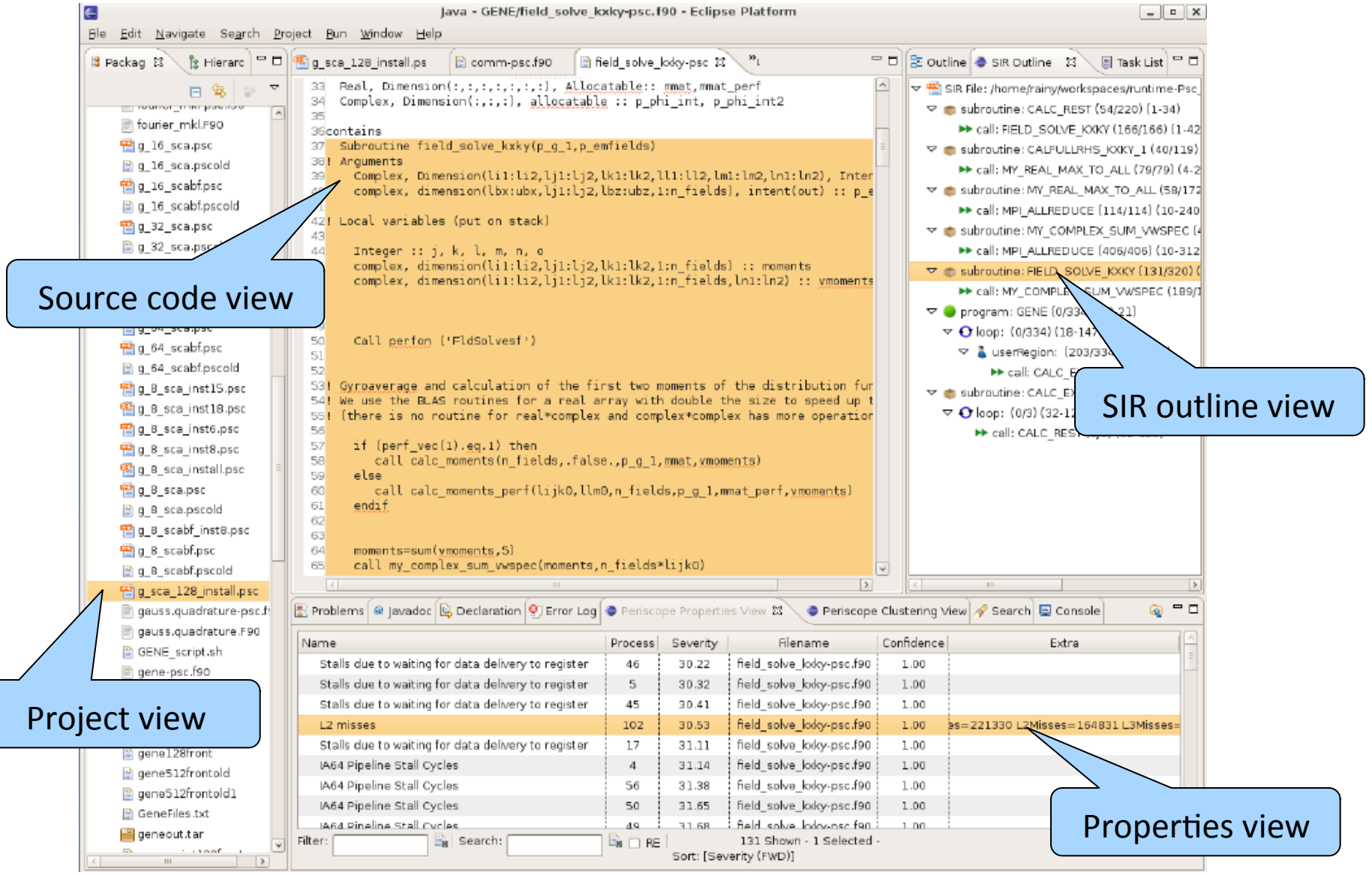
Periscope

- Automated profile-based performance analysis
  - Iterative on-line performance analysis
    - Multiple distributed hierarchical agents
  - Automatic search for bottlenecks based on properties formalizing expert knowledge
    - MPI wait states, OpenMP overheads and imbalances
    - Processor utilization hardware counters
  - Clustering of processes/threads with similar properties
  - Eclipse-based integrated environment
- Supports
  - SGI Altix Itanium2, IBM Power and x86-based architectures
- Developed by TU Munich
  - Released as open-source

Periscope Properties and Strategies

- MPI
  - Excessive MPI communication time
  - Excessive MPI time due to many small messages
  - Excessive MPI time in receive due to late sender
  - …
- OpenMP
  - Load imbalance in parallel region/section
  - Sequential computation in master/single/ordered region
  - ...
- Hardware performance counters (platform-specific)
  - Cycles lost due to cache misses
    - high L1/L2/L3 demand load miss rate
  - Cycles lost due to no instruction to dispatch
  - ...

Periscope Plug-in to Eclipse Environment



Benchmark Instrumentation

- Instrumentation, either manually or using Score-P
- Change compiler to Score-P wrapper script
- Compile and link

Periscope Online Analysis

- Periscope is started via its frontend and automatically starts application and hierarchy of analysis agents

```
% psc_frontend --help
Usage: psc_frontend <options>
  [--help]              (displays this help message)
  [--quiet]             (do not display debug messages)
  [--registry=host:port]  (address of the registry service, optional)
  [--port=n]             (local port number, optional)
  [--maxfan=n]           (max. number of child agents, default=4)
  [--timeout=secs]       (timeout for startup of agent hierarchy)
  [--delay=n]            (search delay in phase executions)
  [--appname=name]
  [--apprun=commandline]
  [--mpinumprocs=number of MPI processes]
  [--ompnumthreads=number of OpenMP threads]
…
  [--strategy=name]
  [--sir=name]
  [--phase=(FileID,RFL)]
  [--debug=level]
```

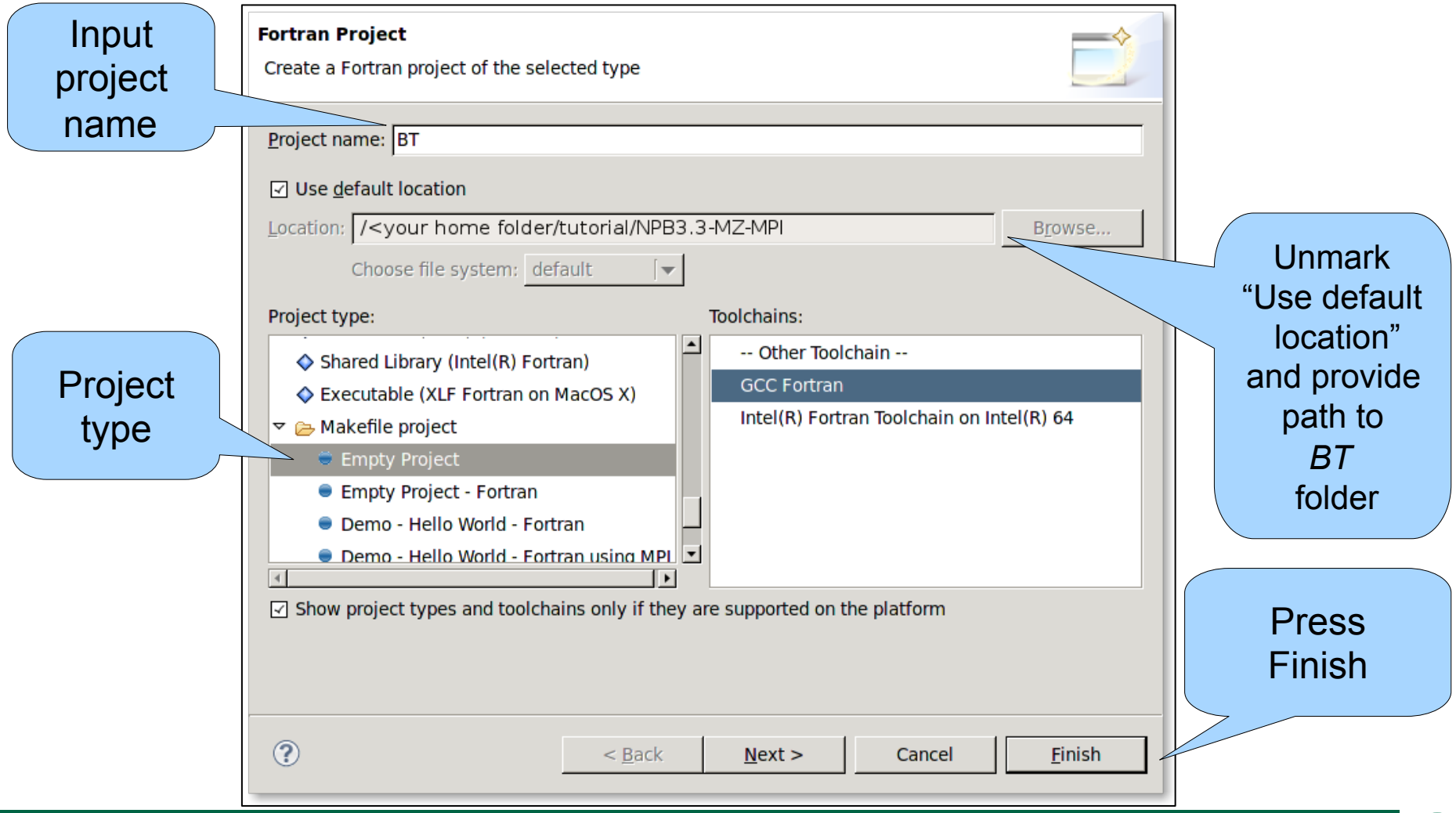
Periscope Online Analysis Strategies

- Experiment with strategies
  - Strategies: MPI, OMP, scalability_OMP
- Experiment with OpenMP thread counts

```
livetau$ psc_frontend --apprun=./bt-mz_B.4 --strategy=MPI
--mpinumprocs=4 –ompnumthreads=1 –phase="OA_phase"

[psc_frontend][DBG0:fe] Agent network UP and RUNNING. Starting search.

 NAS Parallel Benchmarks 3.3 -- BT Benchmark
 [...]
 Time step 200
 BT Benchmark Completed.

---------
End Periscope run! Search took 60.5 seconds (33.3 seconds for startup)
```

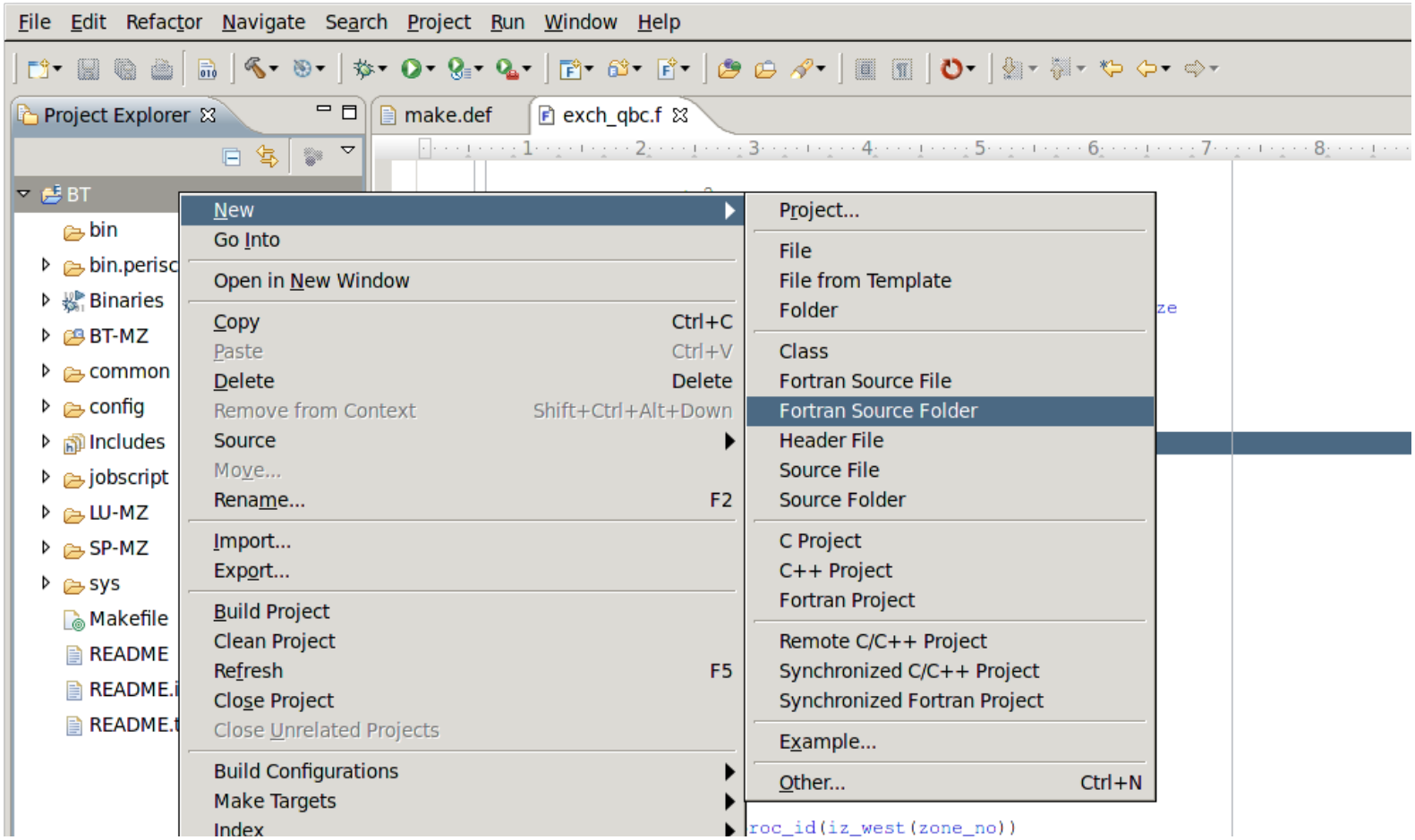
Creating Fortran Project

- File->New->Project... → Fortran->Fortran Project

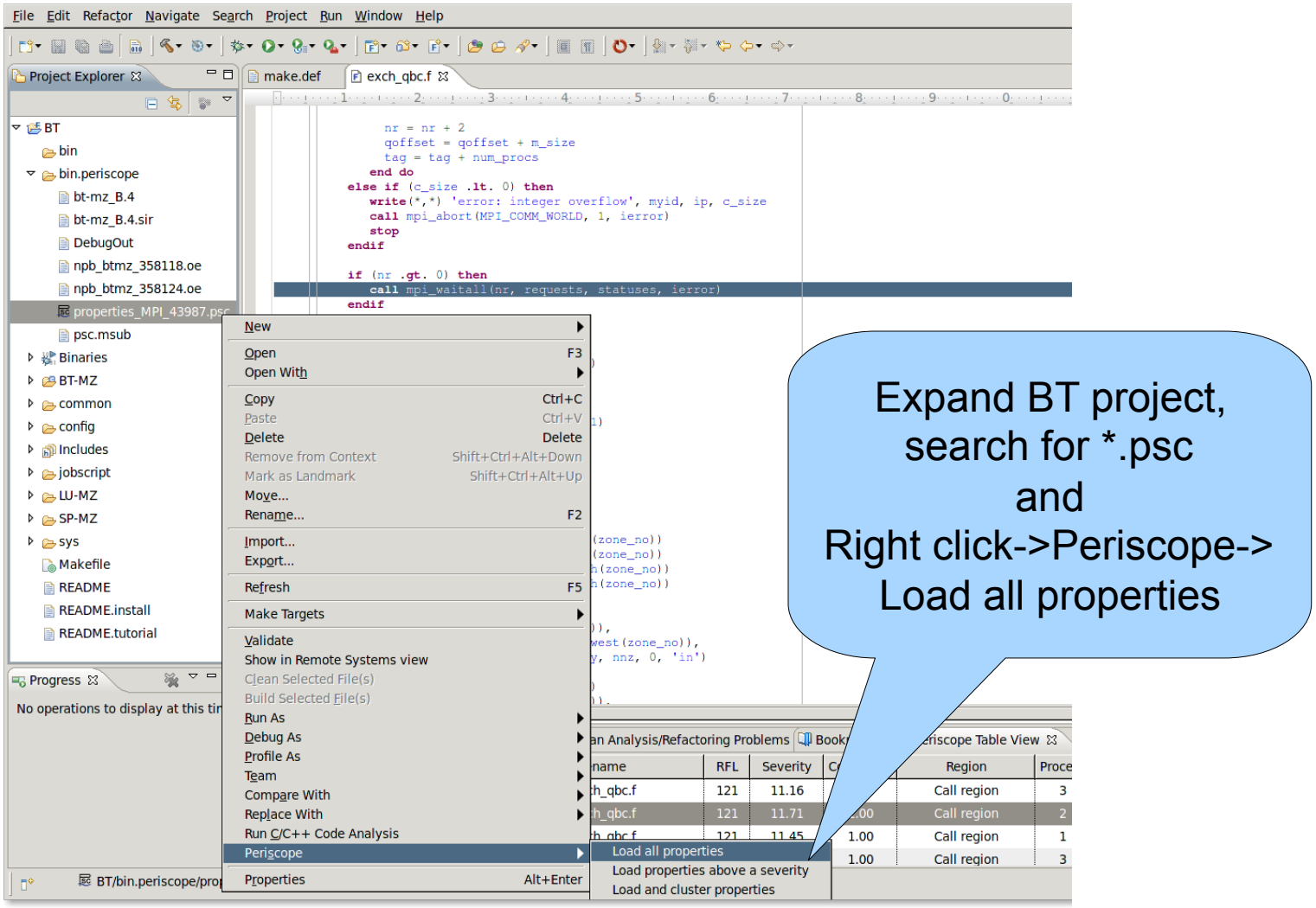


Add BT-MZ as a Source Folder

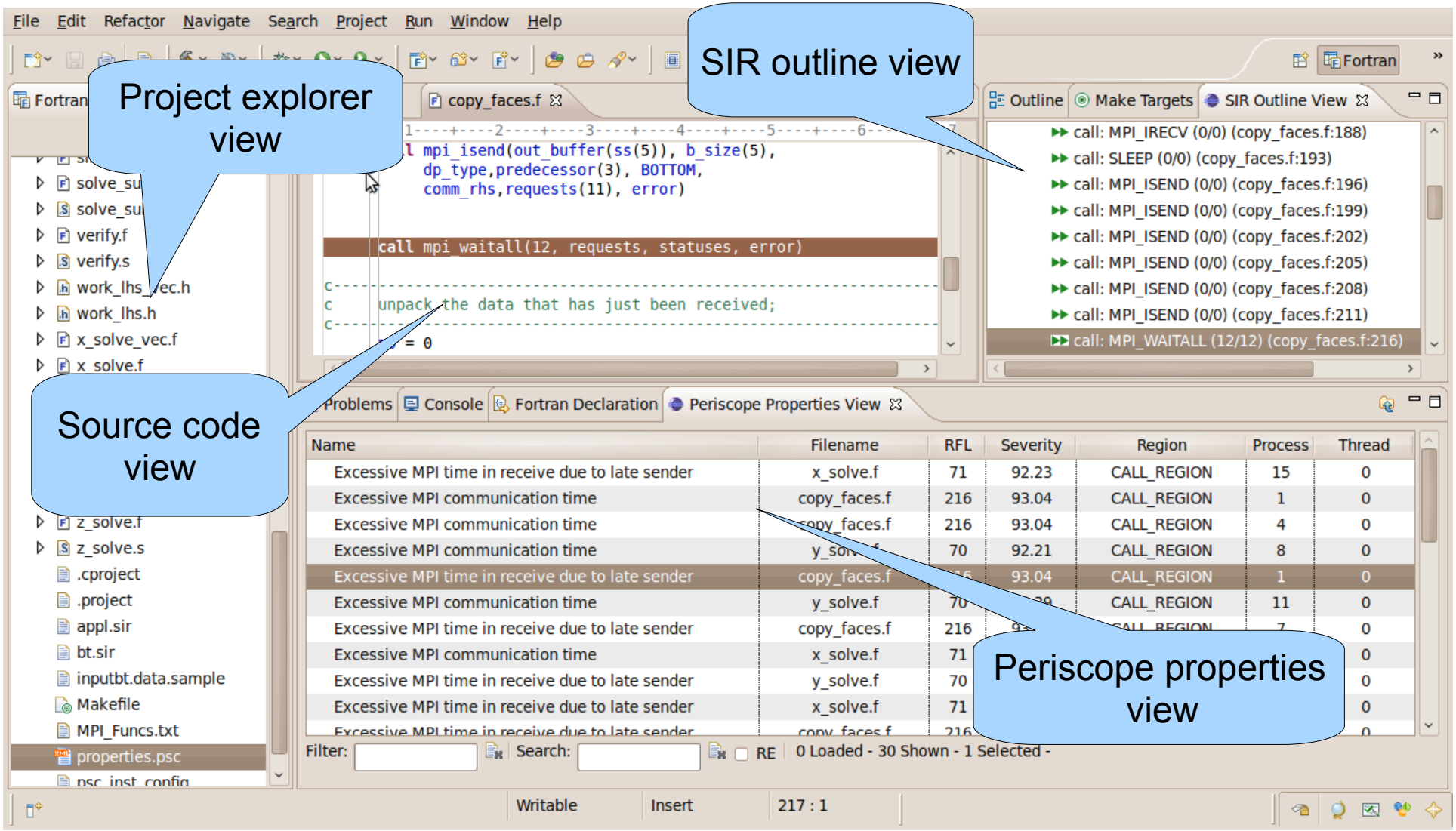
- Right-click -> File-> New -> Fortran Source Folder



Loading Properties



Periscope GUI



Periscope GUI Report Exploration Features

- Multi-functional table is used in the GUI for Eclipse for the visualization of bottlenecks
  - Multiple criteria sorting algorithm
  - Complex categorization utility
  - Searching engine using Regular Expressions
  - Filtering operations
  - Direct navigation from the bottlenecks to their precise source location using the default IDE editor for that source file type (e.g. CDT/Photran editor).

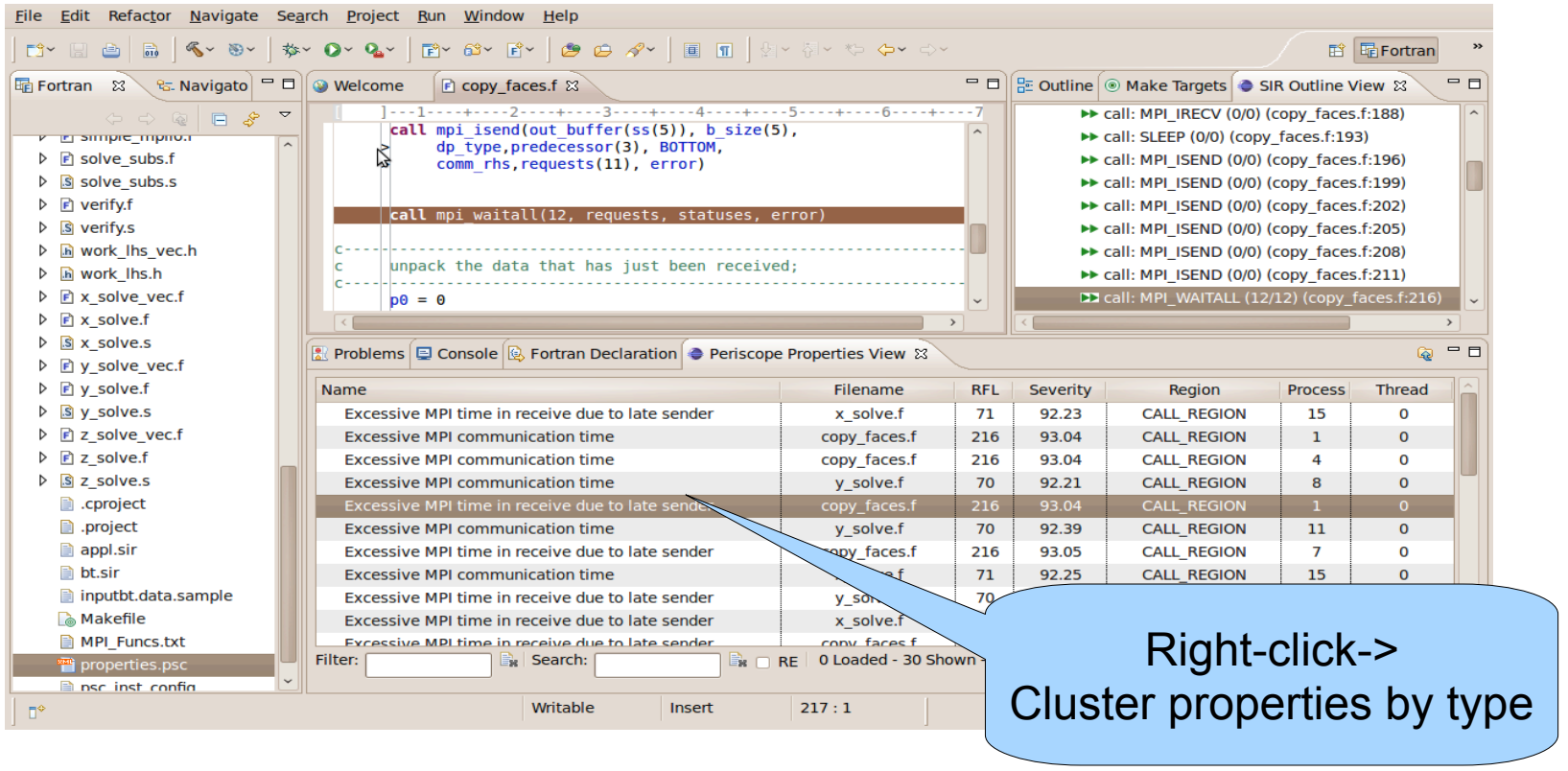
- SIR outline view shows a combination of the standard intermediate representation (SIR) of the analysed application and the distribution of its bottlenecks. The main goals of this view are to assist the navigation in the source code and attract developer's attention to the most problematic code areas.

Periscope GUI Report Exploration Features

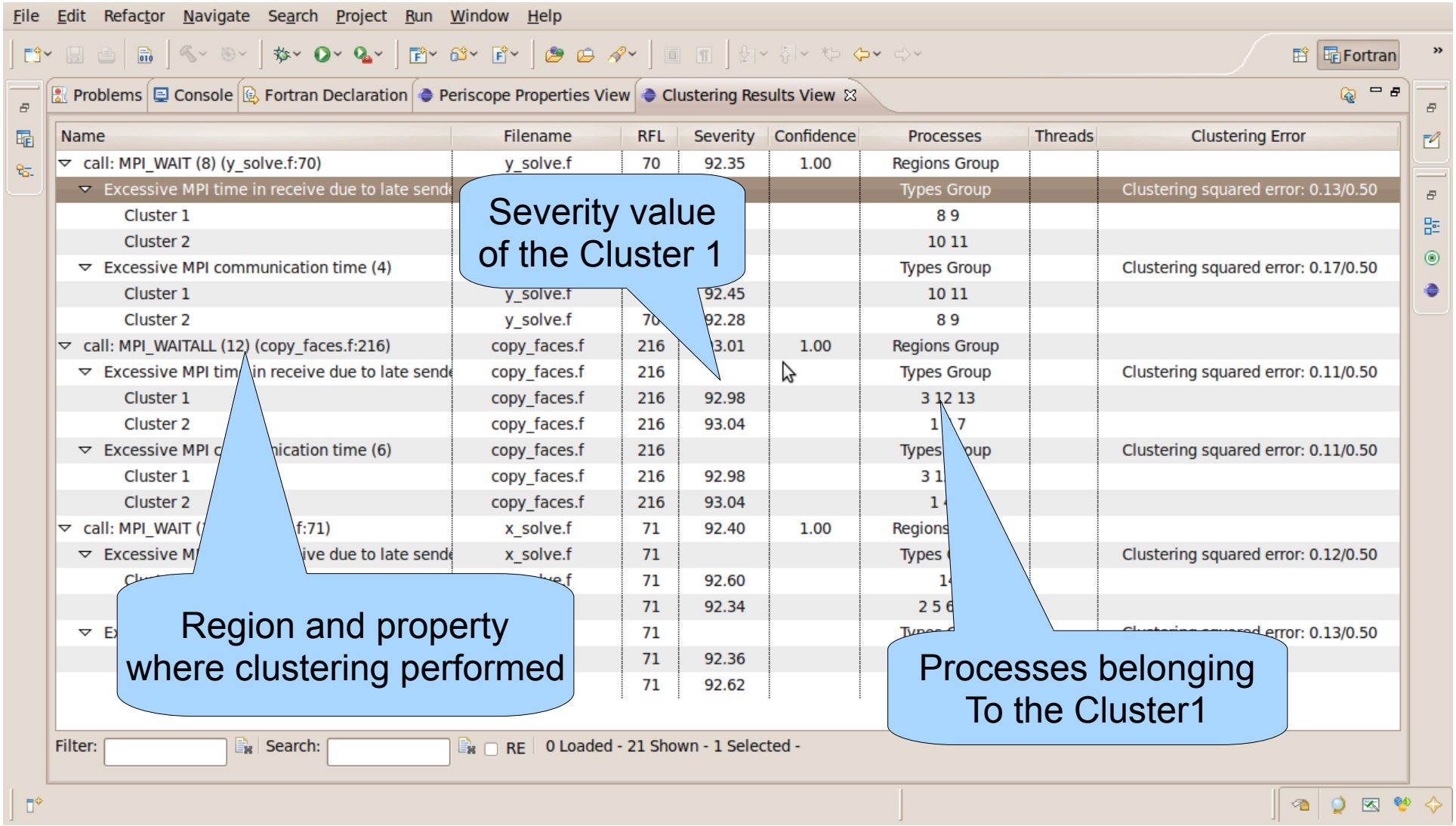
- Multi-functional table is used in the GUI for Eclipse for the visualization of bottlenecks
  - Multiple criteria sorting algorithm
  - Complex categorization utility
  - Searching engine using Regular Expressions
  - Filtering operations
  - Direct navigation from the bottlenecks to their precise source location using the default IDE editor for that source file type (e.g. CDT/Photran editor)
- SIR outline view shows a combination of the standard intermediate representation (SIR) of the analysed application and the distribution of its bottlenecks
- Main goals of this view are to assist the navigation in the source code and attract developer's attention to the most problematic code areas.

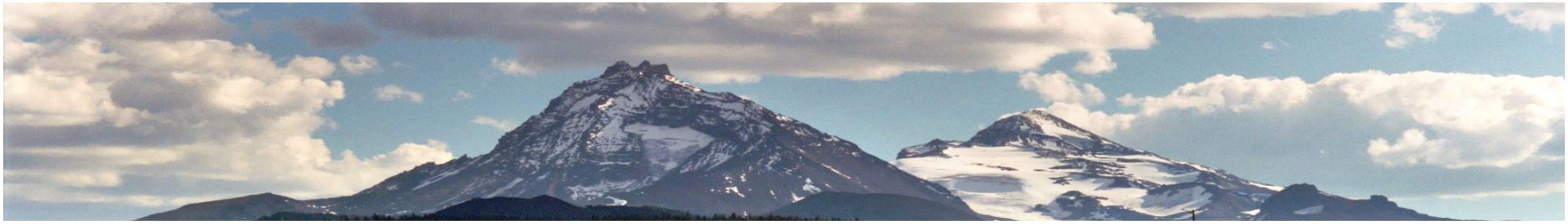
Properties Clustering

- Clustering can effectively summarize displayed properties and identify a similar performance behaviour possibly hidden in the large amount of data



Properties Clustering





mpiP: Lightweight, Scalable MPI Profiling

Jeff Vetter

Oak Ridge National Laboratory (USA)

http://mpip.sourceforge.net

So, You Need to Look at a New Application …

- Scenarios
  - New application development
  - Analyze/Optimize external application
  - Suspected bottlenecks
- First goal is an overview of …
  - Communication frequency and intensity
  - Types and complexity of communication
  - Source code locations of expensive MPI calls
  - Differences between processes

Basic Principle of Profiling MPI

- Intercept all MPI API calls
  - Using wrappers for all MPI calls
- Aggregate statistics over time
  - Number of invocations
  - Data volume
  - Time spent during function execution
- Multiple aggregations options/granularity
  - By function name or type
  - By source code location (call stack)
  - By process rank

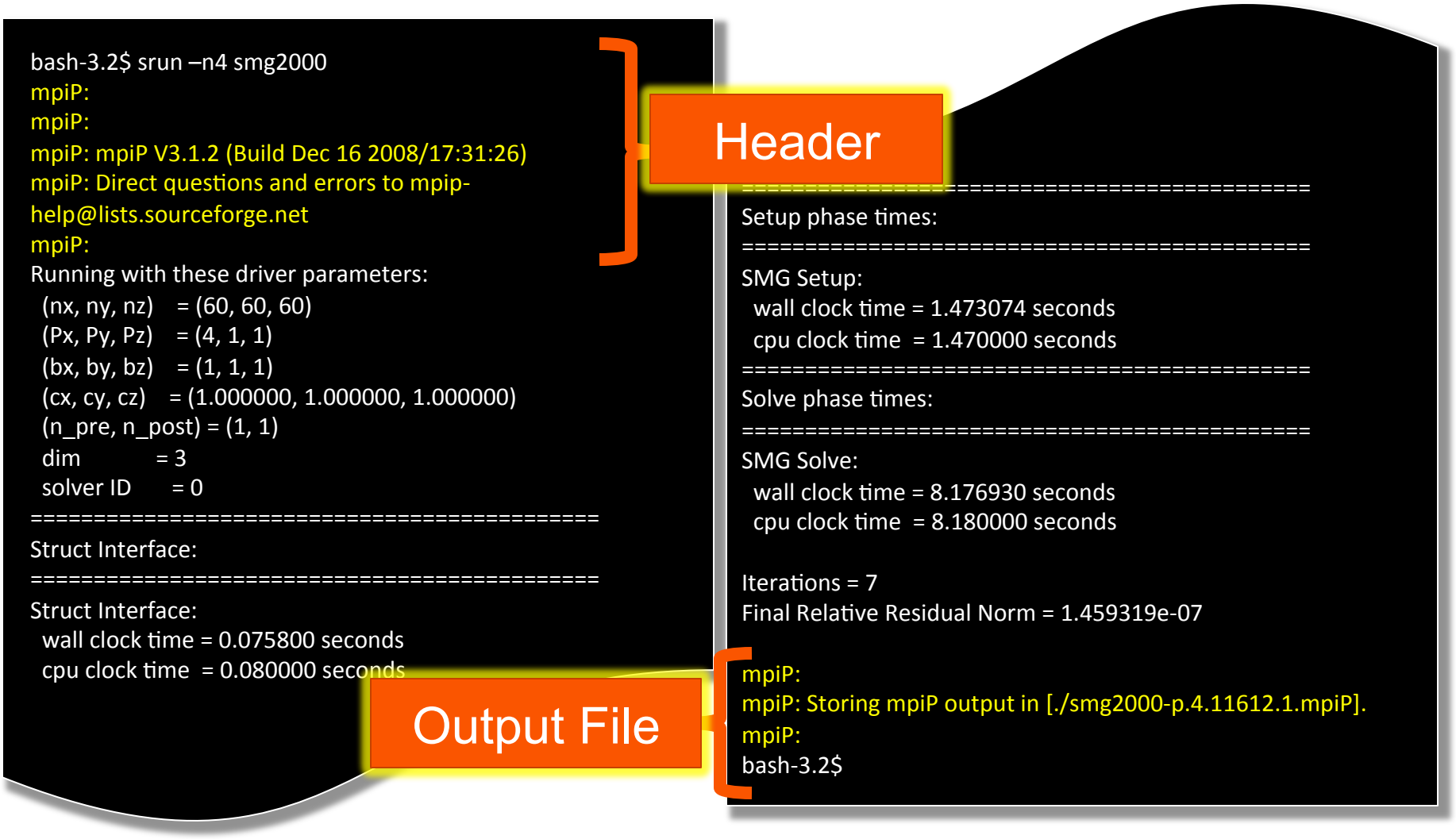
mpiP: Efficient MPI Profiling

- Open source MPI profiling library
  - Developed at LLNL, maintained by LLNL & ORNL
  - Available from sourceforge
  - Works with any MPI library
- Easy-to-use and portable design
  - Relies on PMPI instrumentation
  - No additional tool daemons or support infrastructure
  - Single text file as output
  - Optional: GUI viewer

Running with mpiP 101 / Experimental Setup

- mpiP works on binary files
  - Uses standard development chain
  - Use of "-g" recommended
- Run option 1: Relink
  - Specify libmpi.a/.so on the link line
  - Portable solution, but requires object files
- Run option 2: library preload
  - Set preload variable (e.g., LD_PRELOAD) to mpiP
  - Transparent, but only on supported systems

Running with mpiP 101 / Running



mpiP 101 / Output – Metadata

```
@ mpiP
@ Command : ./smg2000-p -n 60 60 60
@ Version                  : 3.1.2
@ MPIP Build date          : Dec 16 2008, 17:31:26
@ Start time               : 2009 09 19 20:38:50
@ Stop time                : 2009 09 19 20:39:00
@ Timer Used               : gettimeofday
@ MPIP env var             : [null]
@ Collector Rank           : 0
@ Collector PID            : 11612
@ Final Output Dir         : .
@ Report generation        : Collective
@ MPI Task Assignment      : 0 hera27
@ MPI Task Assignment      : 1 hera27
@ MPI Task Assignment      : 2 hera31
@ MPI Task Assignment      : 3 hera31
```

mpiP 101 / Output – Overview

```
---------------------------------------------------------
@--- MPI Time (seconds) ---------------------------------
---------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0       9.78       1.97    20.12
   1        9.8       1.95    19.93
   2        9.8       1.87    19.12
   3       9.77       2.15    21.99
   *       39.1       7.94    20.29
---------------------------------------------------------
```

mpiP 101 / Output – Callsites

```
---------------------------------------------------------------------------
@--- Callsites: 23 ---------------------------------------------------------
---------------------------------------------------------------------------
 ID Lev File/Address        Line Parent_Funct                    MPI_Call
  1   0 communication.c     1405 hypre_CommPkgUnCommit           Type_free
  2   0 timing.c             419 hypre_PrintTiming               Allreduce
  3   0 communication.c      492 hypre_InitializeCommunication   Isend
  4   0 struct_innerprod.c   107 hypre_StructInnerProd           Allreduce
  5   0 timing.c             421 hypre_PrintTiming               Allreduce
  6   0 coarsen.c            542 hypre_StructCoarsen             Waitall
  7   0 coarsen.c            534 hypre_StructCoarsen             Isend
  8   0 communication.c     1552 hypre_CommTypeEntryBuildMPI     Type_free
  9   0 communication.c     1491 hypre_CommTypeBuildMPI          Type_free
 10   0 communication.c      667 hypre_FinalizeCommunication     Waitall
 11   0 smg2000.c            231 main                            Barrier
 12   0 coarsen.c            491 hypre_StructCoarsen             Waitall
 13   0 coarsen.c            551 hypre_StructCoarsen             Waitall
 14   0 coarsen.c            509 hypre_StructCoarsen             Irecv
 15   0 communication.c     1561 hypre_CommTypeEntryBuildMPI     Type_free
 16   0 struct_grid.c        366 hypre_GatherAllBoxes            Allgather
 17   0 communication.c     1487 hypre_CommTypeBuildMPI          Type_commit
 18   0 coarsen.c            497 hypre_StructCoarsen             Waitall
 19   0 coarsen.c            469 hypre_StructCoarsen             Irecv
 20   0 communication.c     1413 hypre_CommPkgUnCommit           Type_free
 21   0 coarsen.c            483 hypre_StructCoarsen             Isend
 22   0 struct_grid.c        395 hypre_GatherAllBoxes            Allgatherv
 23   0 communication.c      485 hypre_InitializeCommunication   Irecv
---------------------------------------------------------------------------
```

mpiP 101 / Output – per Function Timing

```
---------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ---
---------------------------------------------------------------
Call                 Site       Time    App%    MPI%      COV
Waitall                10    4.4e+03   11.24   55.40     0.32
Isend                   3   1.69e+03    4.31   21.24     0.34
Irecv                  23        980    2.50   12.34     0.36
Waitall                12        137    0.35    1.72     0.71
Type_commit            17        103    0.26    1.29     0.36
Type_free               9       99.4    0.25    1.25     0.36
Waitall                 6       81.7    0.21    1.03     0.70
Type_free              15       79.3    0.20    1.00     0.36
Type_free               1       67.9    0.17    0.85     0.35
Type_free              20       63.8    0.16    0.80     0.35
Isend                  21         57    0.15    0.72     0.20
Isend                   7       48.6    0.12    0.61     0.37
Type_free               8       29.3    0.07    0.37     0.37
Irecv                  19       27.8    0.07    0.35     0.32
Irecv                  14       25.8    0.07    0.32     0.34
...
```

mpiP Output – per Function Message Size

```
---------------------------------------------------------------------
@--- Aggregate Sent Message Size (top twenty, descending, bytes) ------
---------------------------------------------------------------------
Call                 Site       Count      Total       Avrg  Sent%
Isend                   3      260044    2.3e+08        885  99.63
Isend                   7        9120   8.22e+05       90.1   0.36
Isend                  21        9120   3.65e+04          4   0.02
Allreduce               4          36        288          8   0.00
Allgatherv             22           4        112         28   0.00
Allreduce               2          12         96          8   0.00
Allreduce               5          12         96          8   0.00
Allgather              16           4         16          4   0.00
```

mpiP 101 / Output – per Callsite Timing

```
---------------------------------------------------------------------
@--- Callsite Time statistics (all, milliseconds): 92 ---------------
---------------------------------------------------------------------
Name                Site Rank  Count      Max     Mean      Min   App%   MPI%
Allgather             16    0      1    0.034    0.034    0.034   0.00   0.00
Allgather             16    1      1    0.049    0.049    0.049   0.00   0.00
Allgather             16    2      1     2.92     2.92     2.92   0.03   0.16
Allgather             16    3      1        3        3        3   0.03   0.14
Allgather             16    *      4        3      1.5    0.034   0.02   0.08

Allgatherv            22    0      1     0.03     0.03     0.03   0.00   0.00
Allgatherv            22    1      1    0.036    0.036    0.036   0.00   0.00
Allgatherv            22    2      1    0.022    0.022    0.022   0.00   0.00
Allgatherv            22    3      1    0.022    0.022    0.022   0.00   0.00
Allgatherv            22    *      4    0.036   0.0275    0.022   0.00   0.00

Allreduce              2    0      3    0.382    0.239    0.011   0.01   0.04
Allreduce              2    1      3     0.31    0.148    0.046   0.00   0.02
Allreduce              2    2      3    0.411    0.178    0.062   0.01   0.03
Allreduce              2    3      3     1.33    0.622    0.062   0.02   0.09
Allreduce              2    *     12     1.33    0.297    0.011   0.01   0.04

...
```

mpiP 101 / Output – per Callsite Message Size

```
---------------------------------------------------------------------
@--- Callsite Message Sent statistics (all, sent bytes) ------------------
---------------------------------------------------------------------
Name                 Site Rank    Count       Max      Mean       Min       Sum
Allgather              16    0        1         4         4         4         4
Allgather              16    1        1         4         4         4         4
Allgather              16    2        1         4         4         4         4
Allgather              16    3        1         4         4         4         4
Allgather              16    *        4         4         4         4        16

Allgatherv             22    0        1        28        28        28        28
Allgatherv             22    1        1        28        28        28        28
Allgatherv             22    2        1        28        28        28        28
Allgatherv             22    3        1        28        28        28        28
Allgatherv             22    *        4        28        28        28       112

Allreduce               2    0        3         8         8         8        24
Allreduce               2    1        3         8         8         8        24
Allreduce               2    2        3         8         8         8        24
Allreduce               2    3        3         8         8         8        24
Allreduce               2    *       12         8         8         8        96

...
```

Fine Tuning the Profile Run

- mpiP Advanced Features
  - User controlled stack trace depth
  - Reduced output for large scale experiments
  - Application control to limit scope
  - Measurements for MPI I/O routines
- Controlled by MPIP environment variable
  - Set by user before profile run
  - Command line style argument list

mpiP Parameters

Param.	Description	Default
-c	Concise Output / No callsite data	
-f dir	Set output directory	
-k n	Set callsite stack traceback size to n	1
-l	Use less memory for data collection	
-n	Do not truncate pathnames	
-o	Disable profiling at startup	
-s n	Set hash table size	256
-t x	Print threshold	0.0
-v	Print concise & verbose output	

Controlling the Stack Trace

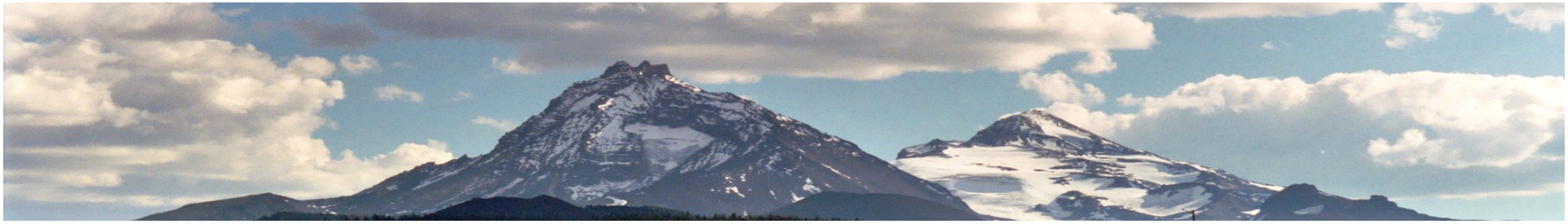
- Callsites are determined using stack traces
  - Starting from current call stack going backwards
  - Useful to avoid MPI wrappers
  - Helps to distinguishes library invocations
- Tradeoff: stack trace depth
  - Too short: can't distinguish invocations
  - Too long: extra overhead / too many call sites
- User can set stack trace depth

Platforms

- Highly portable design
  - Built on top of PMPI, which is part of any MPI
  - Very few dependencies
- Tested on many platforms, including
  - Linux (x86, x86-64, IA-64, MIPS64)
  - BG/L & BG/P
  - AIX (Power 3/4/5)
  - Cray XT3/4/5 with Catamount and CNL
  - Cray X1E

mpiPView: The GUI for mpiP

- Optional: displaying mpiP data in a GUI
- Implemented as part of the Tool Gear project
- Reads mpiP output file
- Provides connection to source files
- Usage process
- First: select input metrics
- Hierarchical view of all callsites
- Source panel once callsite is selected
- Ability to remap source directories



Paraver

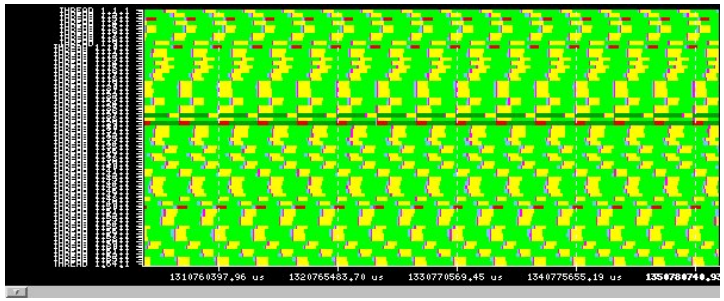
Jesus Labarta

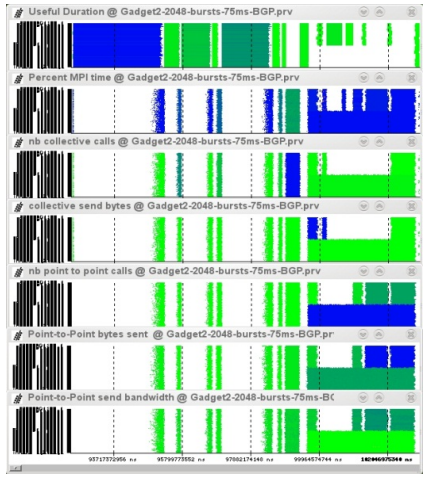
Barcelona Supercomputing Center (Spain)

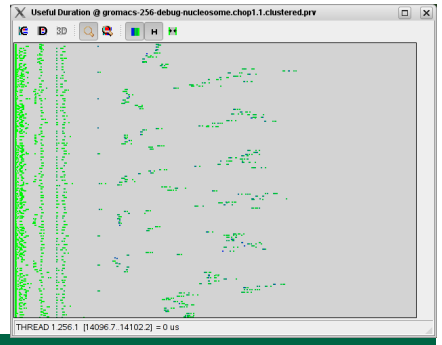
http://www.bsc.es/paraver

BSC Tools

- Since 1991
- Based on traces
- Open Source
  - http://www.bsc.es/paraver
- Core tools:
  - Paraver (paramedir) – offline trace analysis
  - Dimemas – message passing simulator
  - Extrae – instrumentation
- Focus
  - Detail, flexibility, intelligence







Extrae

- Major BSC instrumentation package
- When / where
  - Parallel programming model runtime
    - MPI, OpenMP, pthreads, OmpSs, CUDA, OpenCL, MIC…
    - API entry/exit, OpenMP outlined routines
  - Selected user functions
  - Periodic samples
  - User events
- Additional information
  - Counters
    - PAPI
    - Network counters
    - OS counters
  - Link to source code
    - Callstack

How does Extrae intercept your app?

- LD_PRELOAD
  - Works on production binaries
  - Specific library for each combination of runtimes
  - Does not require knowledge on the application
- Dynamic instrumentation
  - Works on production binaries (based on Dyninst)
  - Just specify functions to be instrumented.
- Other possibilities
  - Link instrumentation library statically
  - OmpSs (instrumentation calls injected by compiler + linked to library)

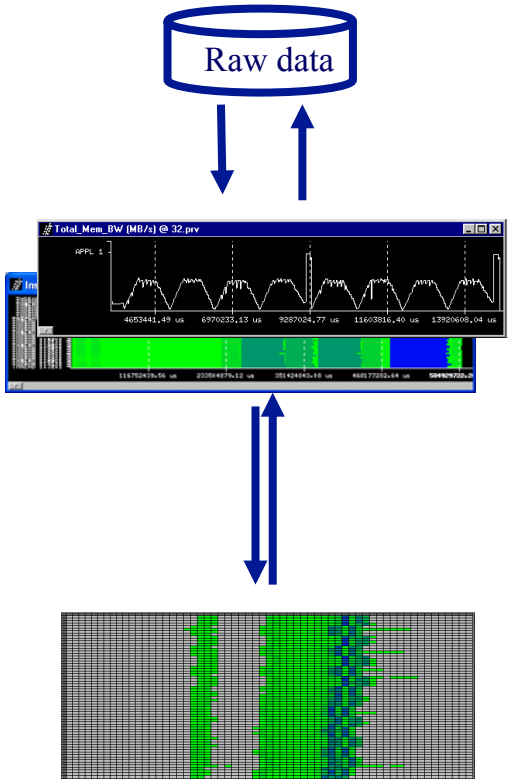
Programming model	Library
Serial	libseqtrace
Pure MPI	libmpitrace[f][1]
Pure OpenMP	libomptrace
Pure Pthreads	libpttrace
CUDA	libcudatrace
MPI + OpenMP	libompitrace[f] [1]
MPI + Pthreads	libptmpitrace[f] [1]
Mpi + CUDA	libcudampitrace[f] [1]

[1] for Fortran codes

How to use Extrae?

- Adapt job submission script
  - Specify LD_PRELOAD library and xml instrumentation control file
- Specify the data to be captured in the .xml instrumentation control file
- Run and get the trace …

Paraver Performance Data Browser



Timelines

2/3D tables (Statistics)

Trace visualization/analysis

+ trace manipulation

Goal = Flexibility

No semantics

Programmable

Comparative analyses

Multiple traces

Synchronize scales

Paraver Mathematical Foundation

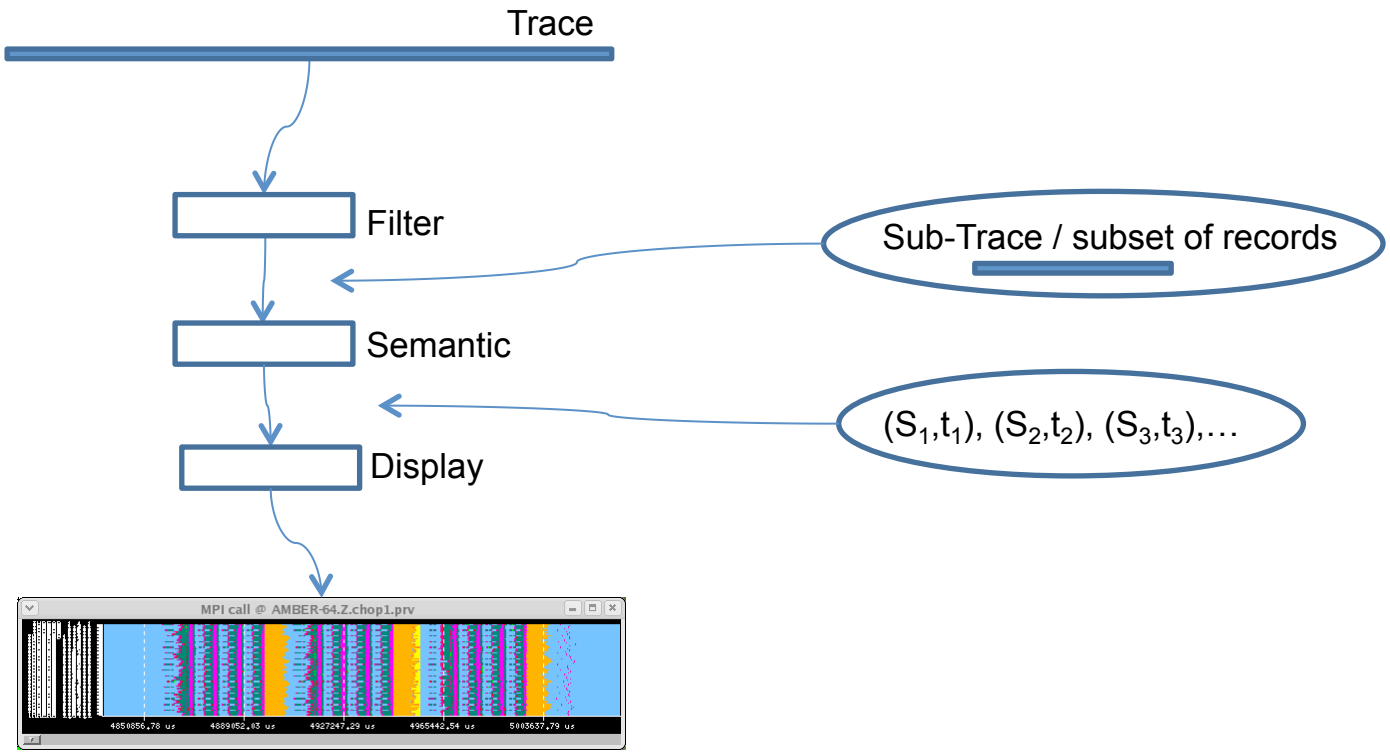
- Every behavioral aspect/metric described as a function of time
  - Possibly aggregated along
    - the process model dimension (**thread**, process, application, workload)
    - The resource model dimension (core, node, system)
  - Language to describe how to compute such functions of time (GUI)
    - Basic operators (from) trace records
    - Ways of combining them
- Those functions of time can be rendered into a 2D image
  - Timeline
- Statistics can be computed for each possible value or range of values of that function of time
  - Tables: profiles and histograms

Paraver Mathematical Foundation

$$s(t) = S_i, t \in [t_i, t_{i+1}), i \in \mathbb{N}$$

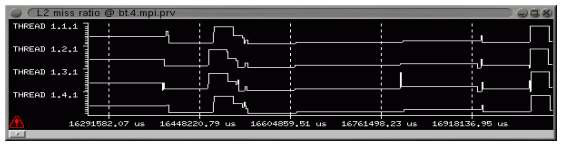
Function of time

Series of values

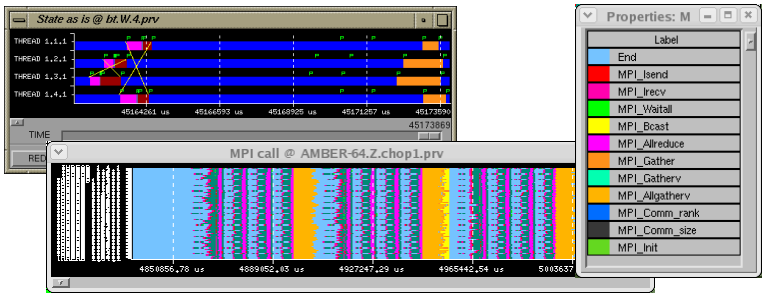


Timelines

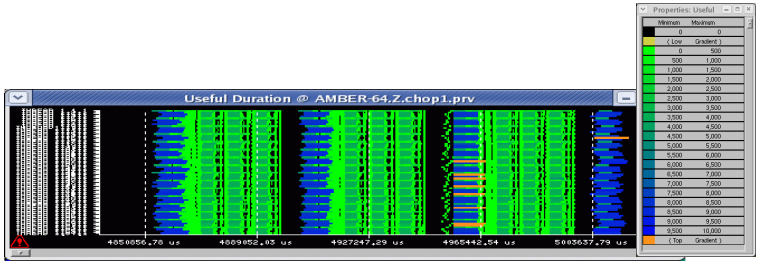
- Representation
  - Function of time



  - Colour encoding



  - Not null gradient
    - Black for zero value
    - Light green → Dark blue



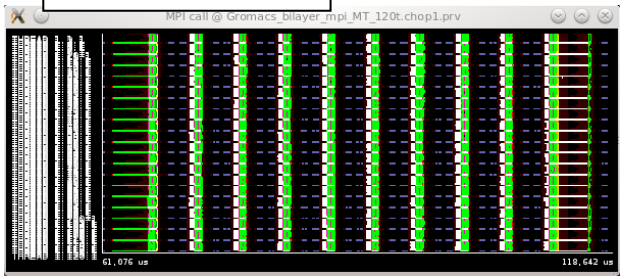
- Non linear rendering to address scalability

J. Labarta, et al.: "Scalability of tracing and visualization tools", PARCO 2005

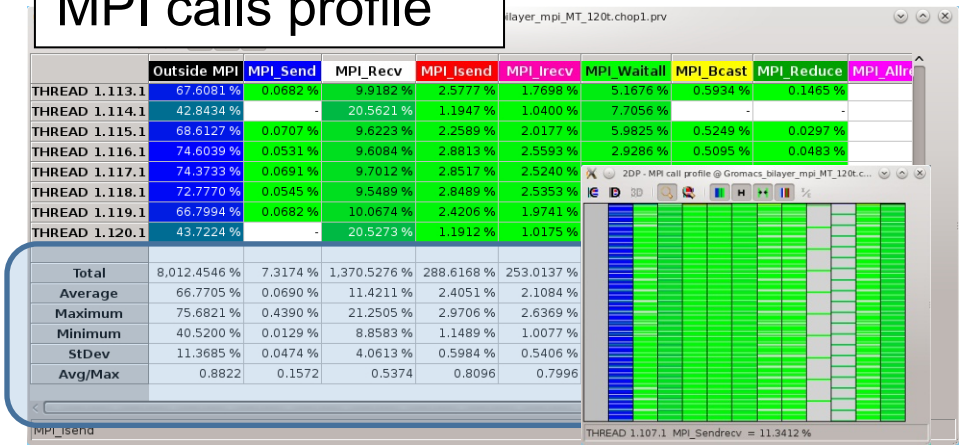
Tables: Profiles, histograms, correlations

- From timelines to tables

MPI calls

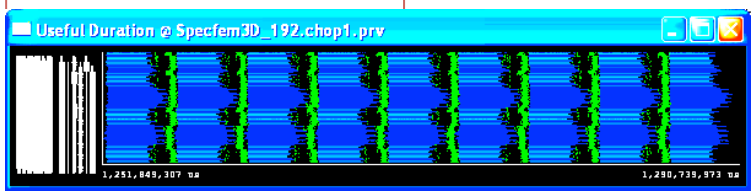


MPI calls profile

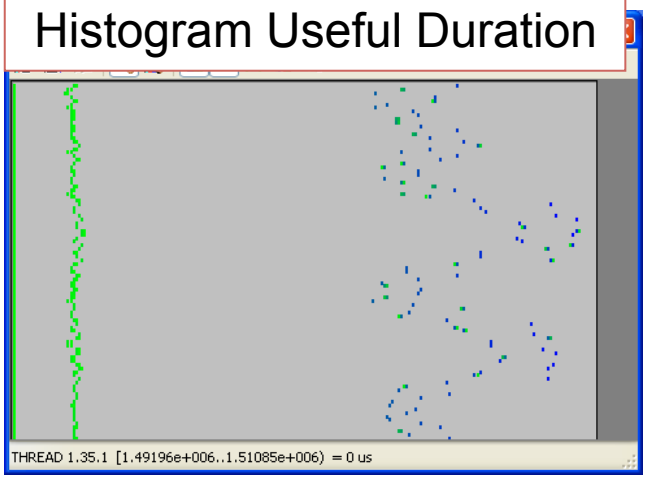


	Outside MPI	MPI_Send	MPI_Recv	MPI_Isend	MPI_Irecv	MPI_Waitall	MPI_Bcast	MPI_Reduce
THREAD 1.113.1	67.6081 %	0.0682 %	9.9182 %	2.5777 %	1.7698 %	5.1676 %	0.5934 %	0.1465 %
THREAD 1.114.1	42.8434 %	-	20.5621 %	1.1947 %	1.0400 %	7.7056 %	-	-
THREAD 1.115.1	68.6127 %	0.0707 %	9.6223 %	2.2589 %	2.0177 %	5.9825 %	0.5249 %	0.0297 %
THREAD 1.116.1	74.6039 %	0.0531 %	9.6084 %	2.8813 %	2.5593 %	2.9286 %	0.5095 %	0.0483 %
THREAD 1.117.1	74.3733 %	0.0691 %	9.7012 %	2.8517 %	2.5240 %			
THREAD 1.118.1	72.7770 %	0.0545 %	9.5489 %	2.8489 %	2.5353 %			
THREAD 1.119.1	66.7994 %	0.0682 %	10.0674 %	2.4206 %	1.9741 %			
THREAD 1.120.1	43.7224 %	-	20.5273 %	1.1912 %	1.0175 %			
Total	8,012.4546 %	7.3174 %	1,370.5276 %	288.6168 %	253.0137 %			
Average	66.7705 %	0.0690 %	11.4211 %	2.4051 %	2.1084 %			
Maximum	75.6821 %	0.4390 %	21.2505 %	2.9706 %	2.6369 %			
Minimum	40.5200 %	0.0129 %	8.8583 %	1.1489 %	1.0077 %			
StDev	11.3685 %	0.0474 %	4.0613 %	0.5984 %	0.5406 %			
Avg/Max	0.8822	0.1572	0.5374	0.8096	0.7996			

Useful Duration



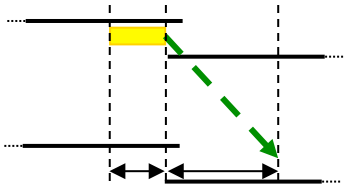
Histogram Useful Duration



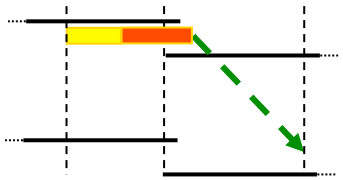
Dimemas: Coarse grain, Trace driven simulation

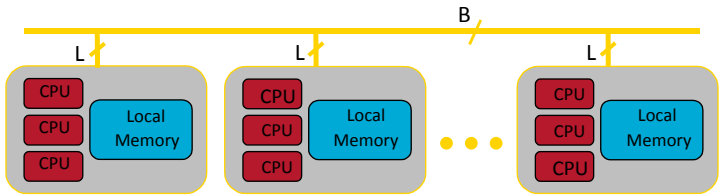
- Simulation: Highly non linear model
  - Linear components
    - Point to point communication
    - Sequential processor performance
      - Global CPU speed
      - Per block/subroutine

$$T = \frac{MessageSize}{BW} + L$$

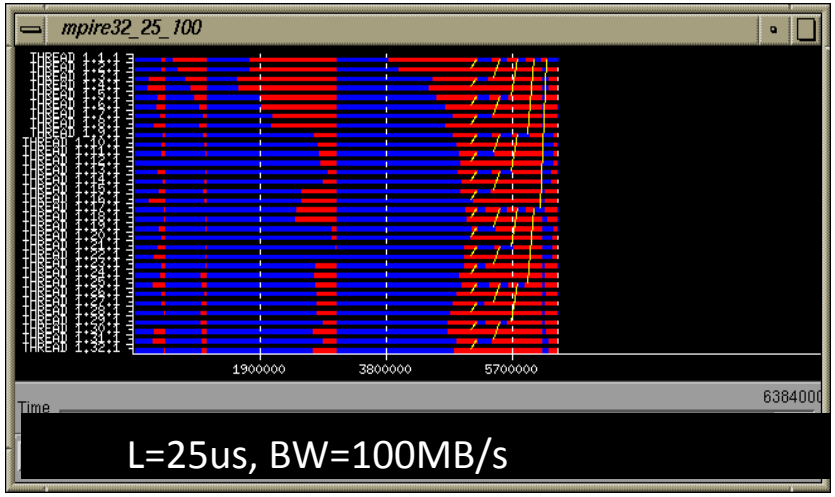


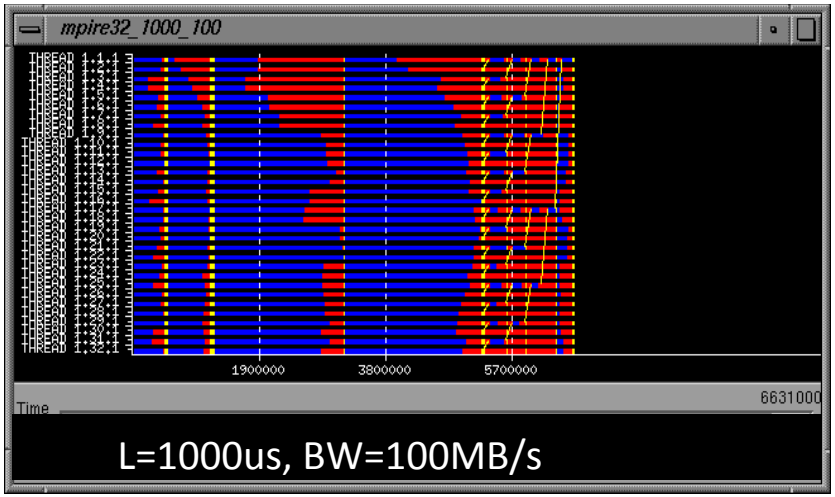
  - Non linear components
    - Synchronization semantics
      - Blocking receives
      - Rendezvous
    - Resource contention
      - CPU
      - Communication subsystem
        - » links (half/full duplex), busses

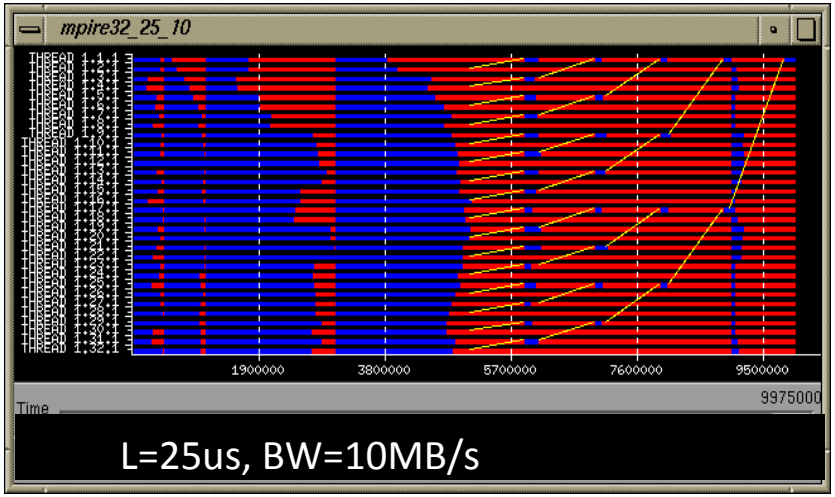




MPIRE 32 tasks, no network contention



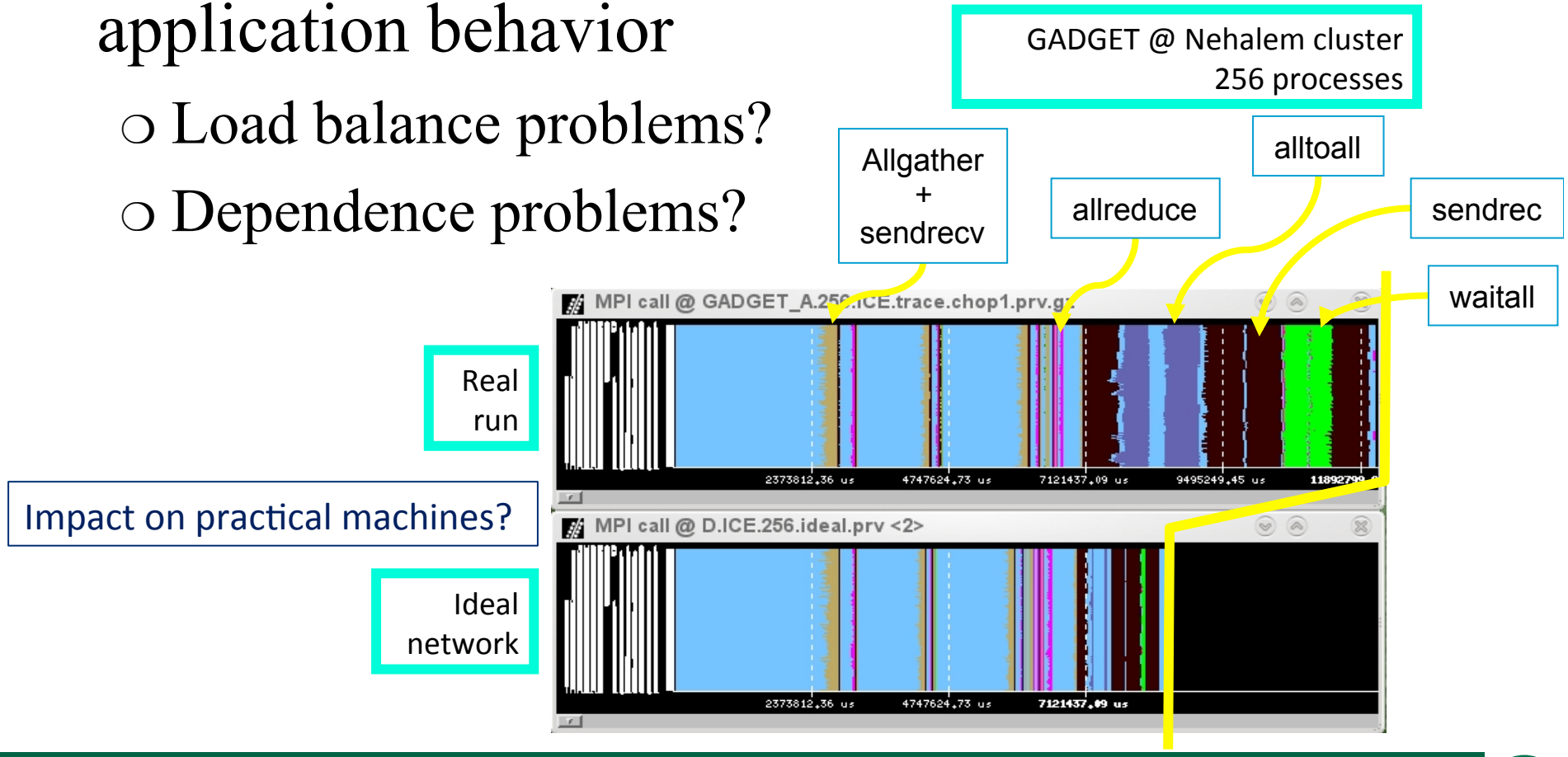


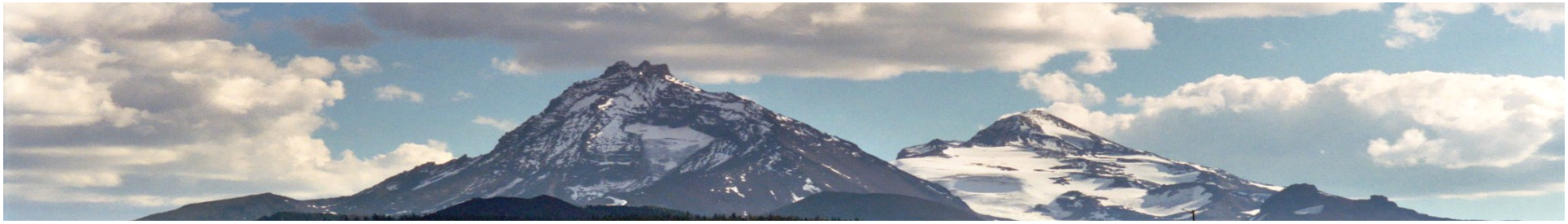


All windows same scale

Ideal Machine

- The impossible machine: $BW = \infty, \quad L = 0$
- Actually describes/characterizes Intrinsic application behavior
  - Load balance problems?
  - Dependence problems?





Prophesy / MuMMI

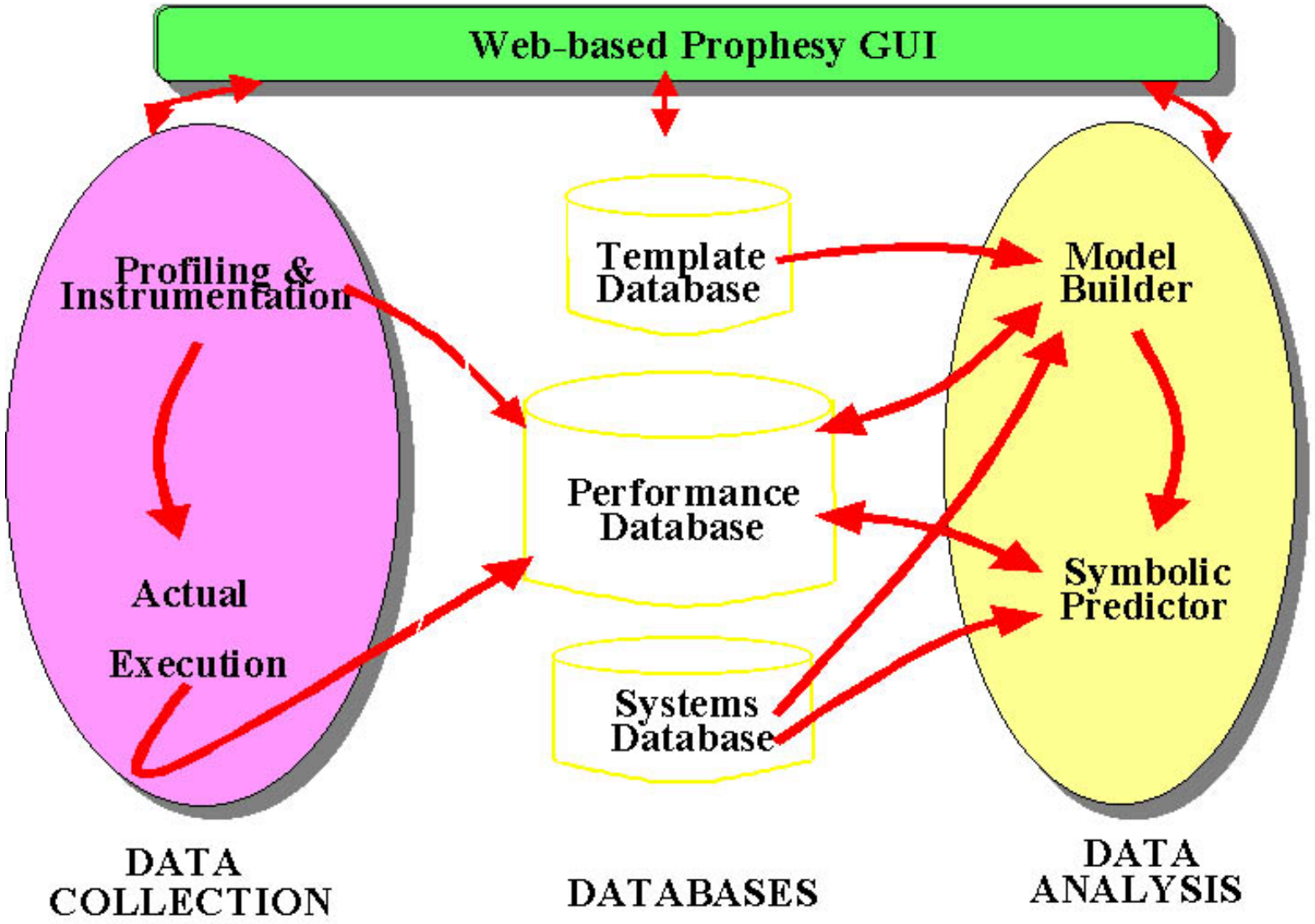
Valerie Taylor
Texas A&M University (USA)
Argonne National Laboratory (USA)
http://prophesy.cs.tamu.edu

PROPHESY

Prophesy

- Infrastructure for analyzing and modeling the performance of parallel and distributed applications
- Relational database with:
  - performance data
  - system features
  - application details
- System can be used to
  - Develop models based upon significant data
  - Identify the most efficient implementation of a given function based upon the given system configuration
  - Explore the various trends implicated by the significant data
  - Predict the performance on a different system

Prophesy Architecture



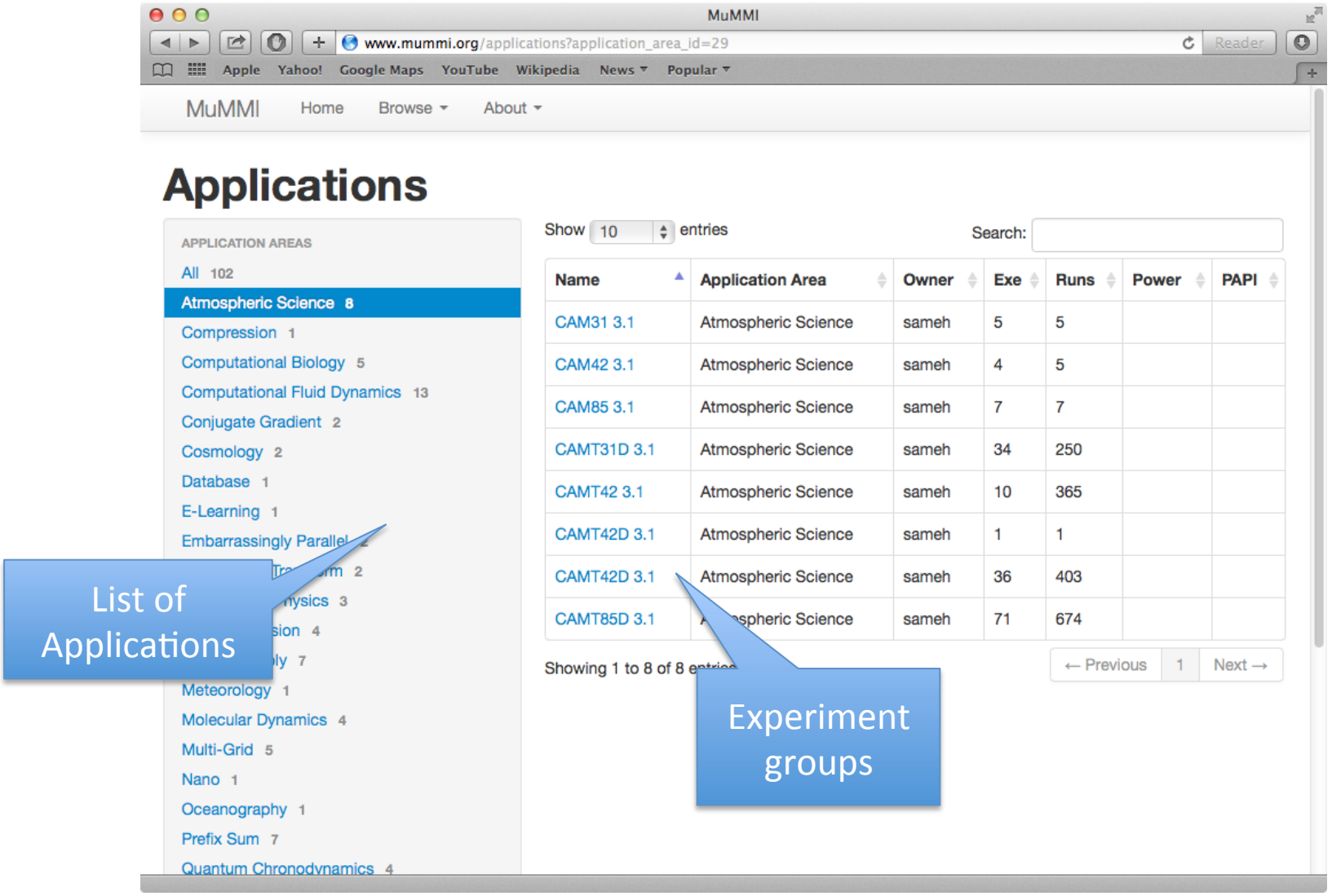
Prophesy Workflow

- Application is instrumented, executed, data is automatically transferred to Prophesy database
- Prophesy generates an analytical model with coefficients based on performance data, system data and model templates
- All web-enabled
- Linear / non-linear models used to:
  - Predict performance on new compute platform
  - Experiment with system feature modifications

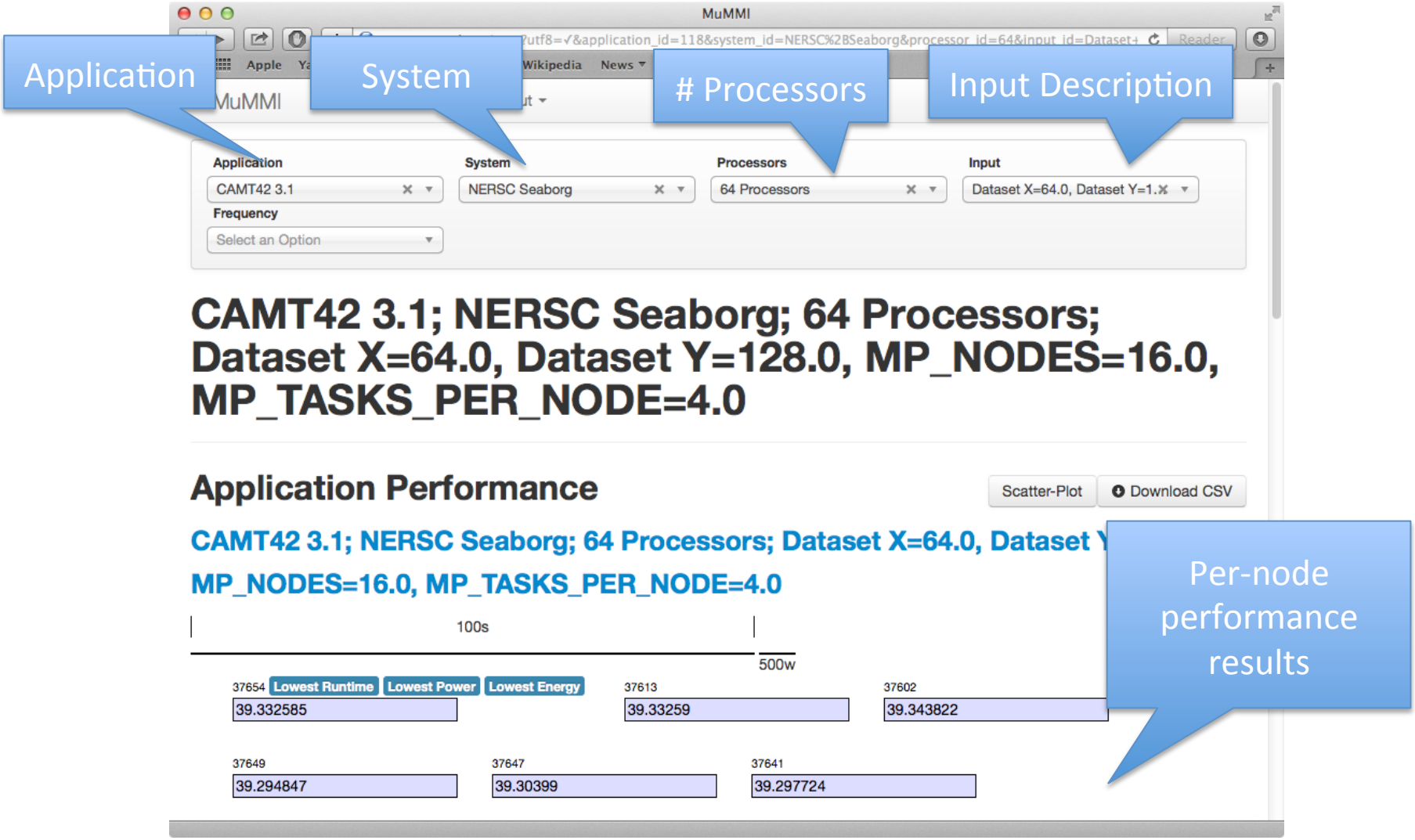
MuMMI

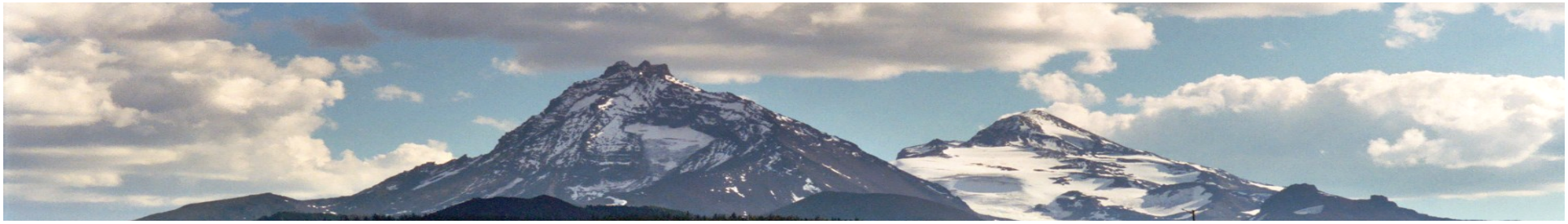
- The **M**ultiple **M**etrics **M**odeling **I**nfrastructure (**MuMMI**) project
- An infrastructure for analyzing and modeling the performance *and* **power consumption** of parallel and distributed applications
- http://www.mummi.org
- MuMMI allows for the development of linear as well as nonlinear models
- Models, when combined with data from the system database, can be used by the prediction engine to predict the performance on a different compute platform
- Models can give insight into which machine may perform the best for the given implementation of the kernel and what happens when one changes different features of the system

MuMMI Application Browser



MuMMI Sample Application





PerfExpert

Jim Browne

Texas Advanced Computing Center (USA)

University of Texas at Austin (USA)

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

Goals for PerfExpert

- Make use of the tool as simple as possible
- Start with only chip/node level optimization
- Make it adaptable across multiple architectures
- Design for extension to communication and I/O performance
- How?
  - Formulate the performance optimization task as a workflow of tasks
  - Leverage the state-of-the-art
  - Automate the entire workflow

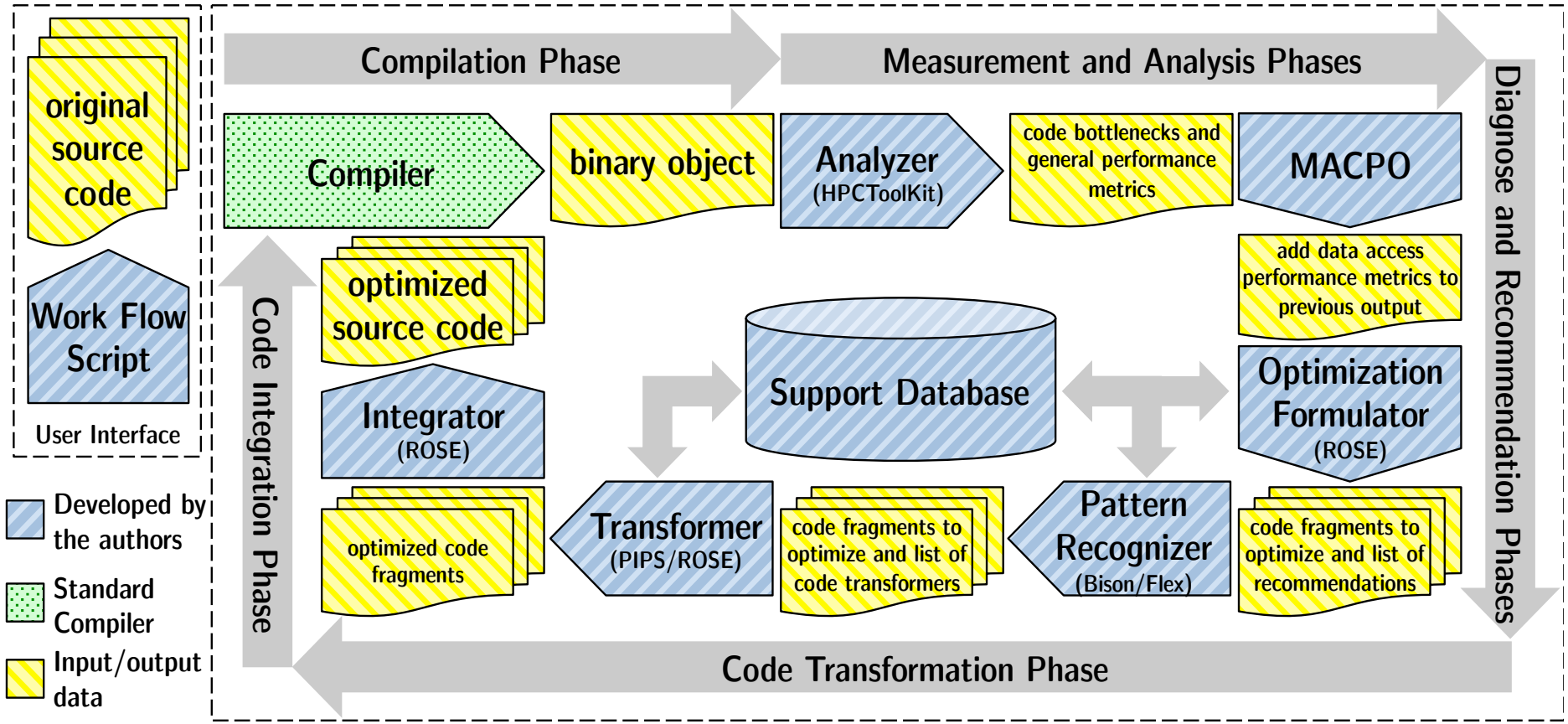
PerfExpert: Introduction

- Four stages of automatic performance optimization:
  - Measurement and attribution
  - Analysis, diagnosis and identification of bottlenecks
  - Selection of effective optimizations
  - Implementation of optimizations
- Components:
  - HPCToolkit
  - MACPO (based on ROSE)
  - Bison, Flex, PIPS

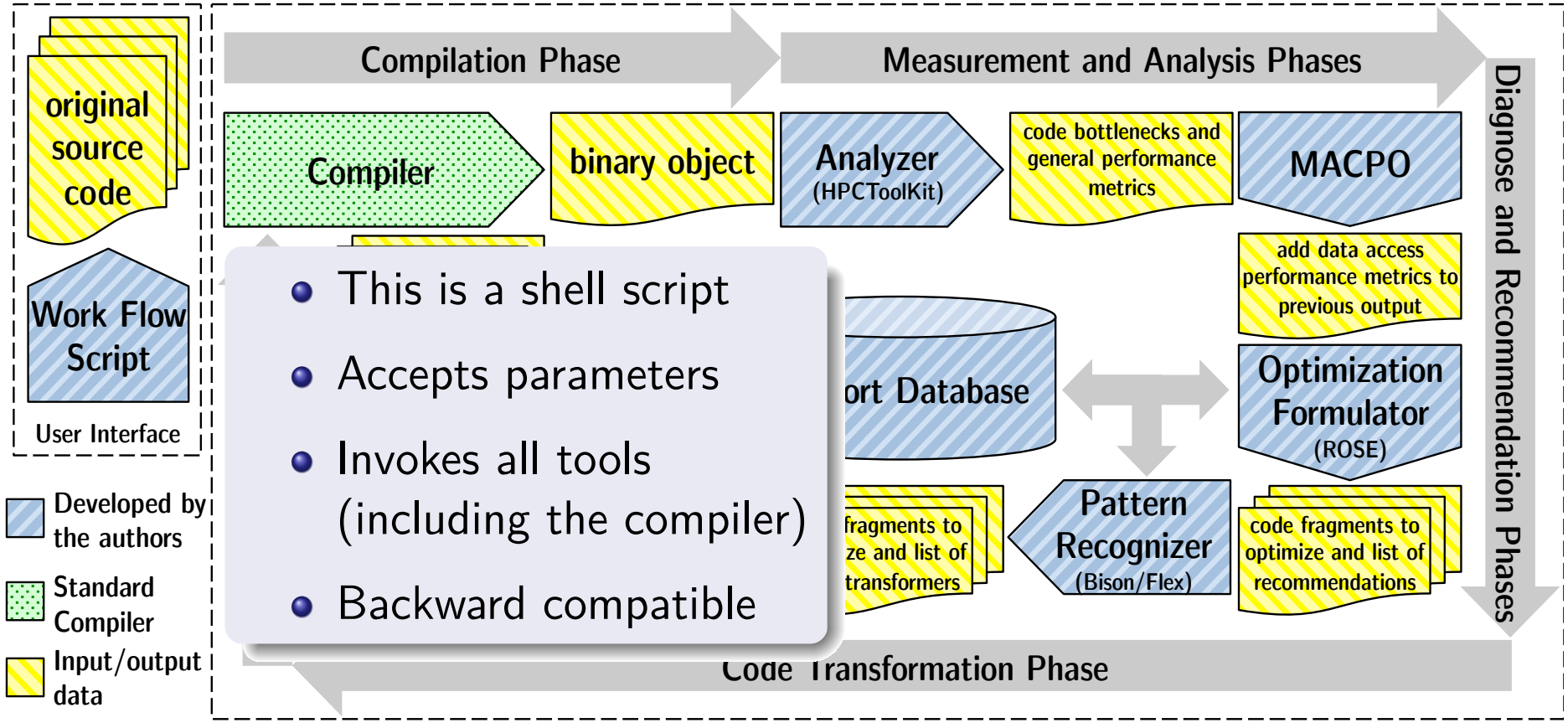
PerfExpert Performance Report

- Identification of bottlenecks by relevance
- Performance analysis based on performance metrics
- Recommendations for optimization
- Three possible outputs:
  - Performance report only
  - List of recommendations
  - Fully automated code transformation

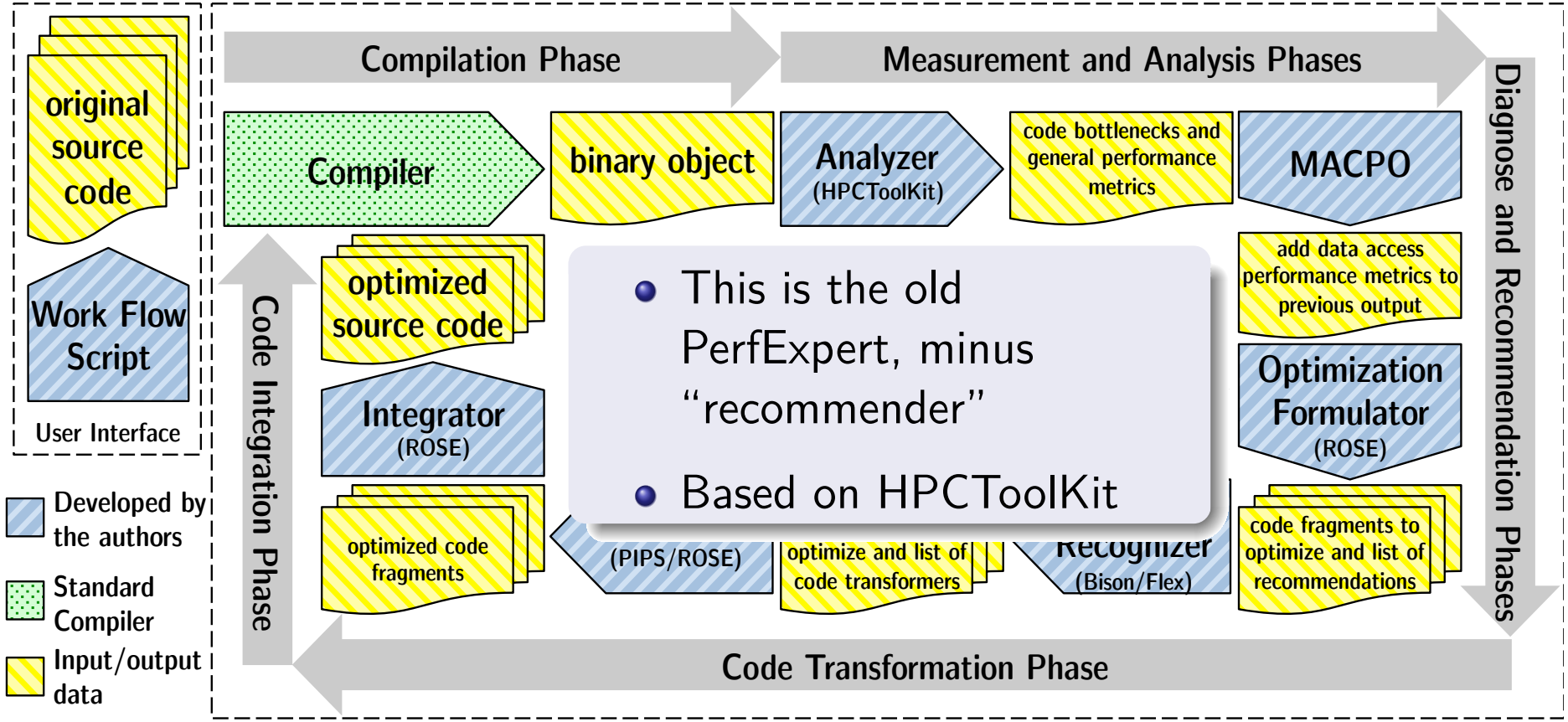
PerfExpert: The Big Picture



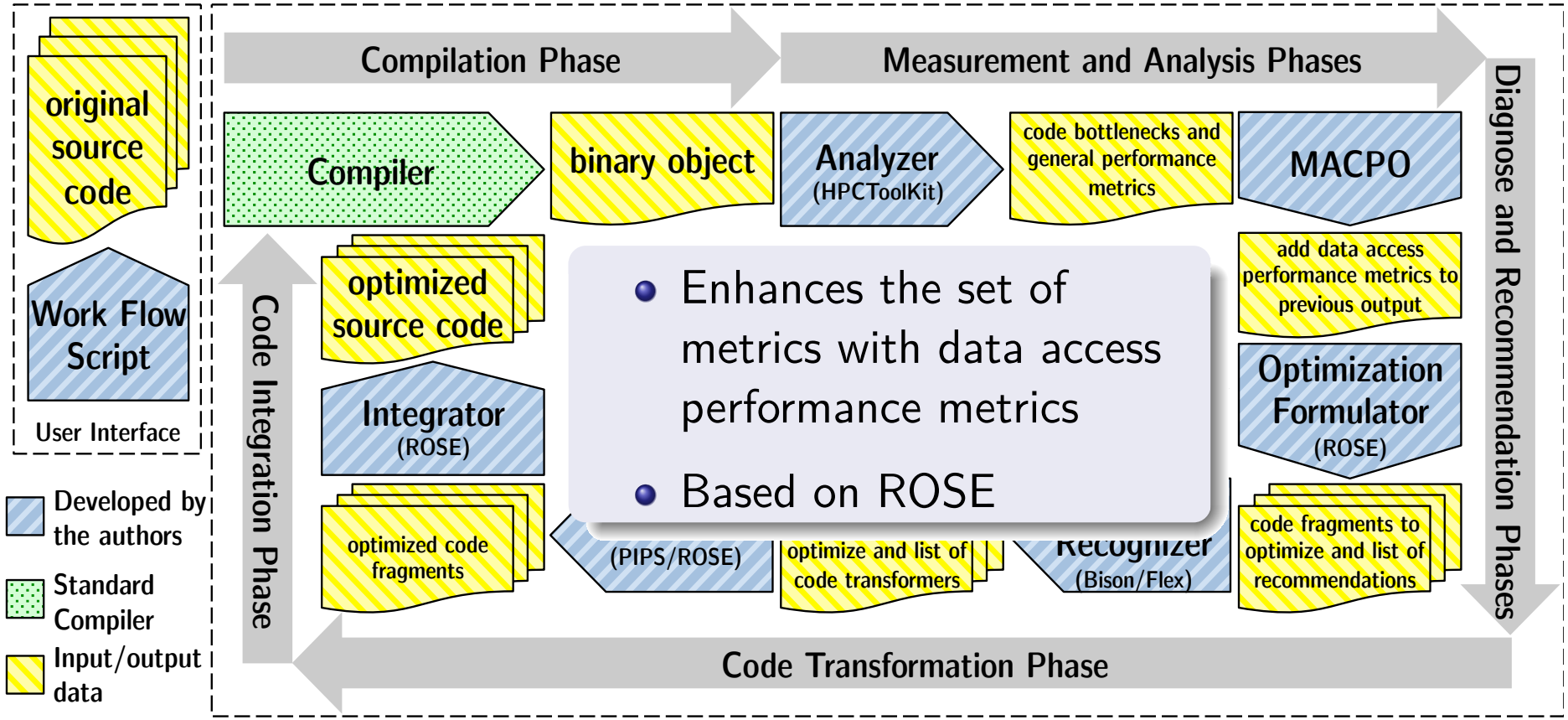
PerfExpert: Work Flow Script



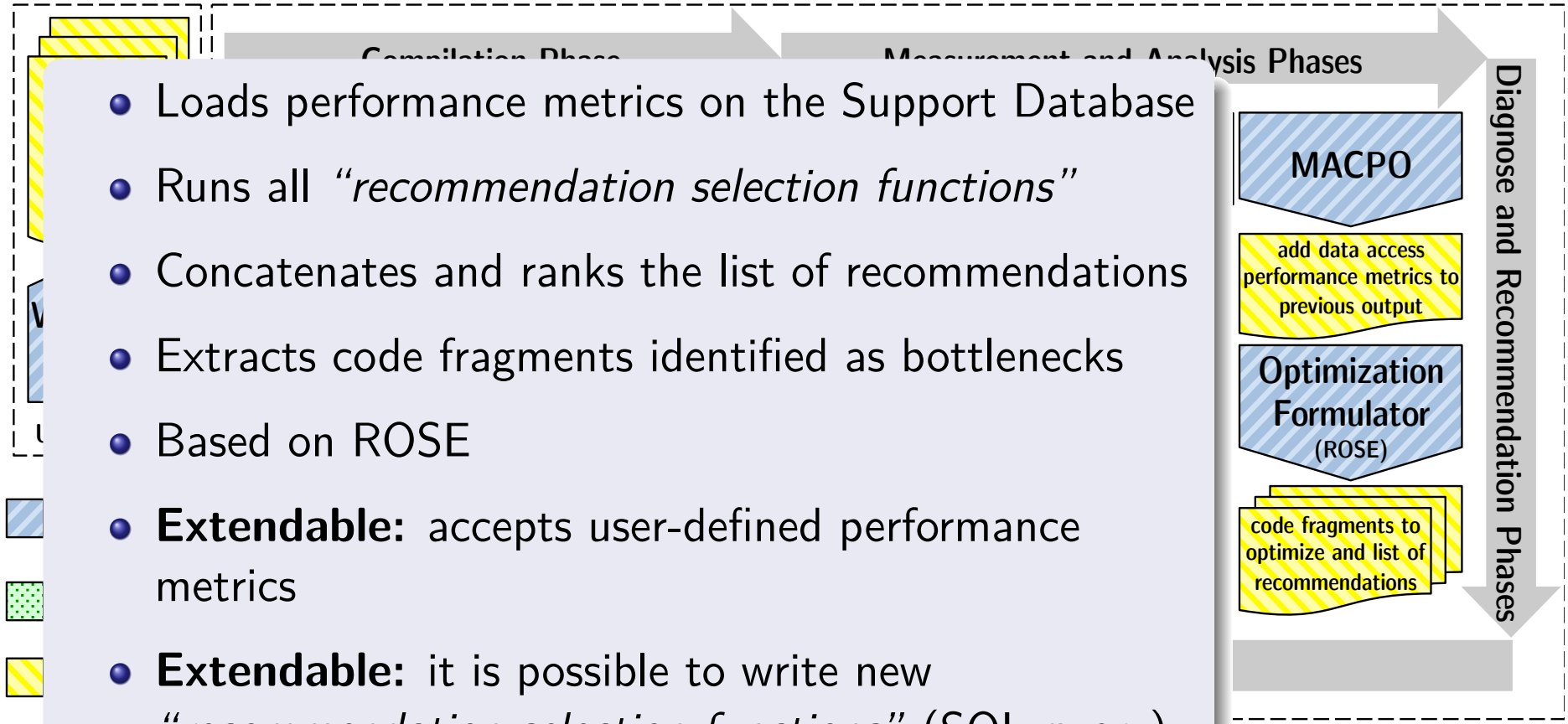
PerfExpert: Analyzer



PerfExpert: MACPO



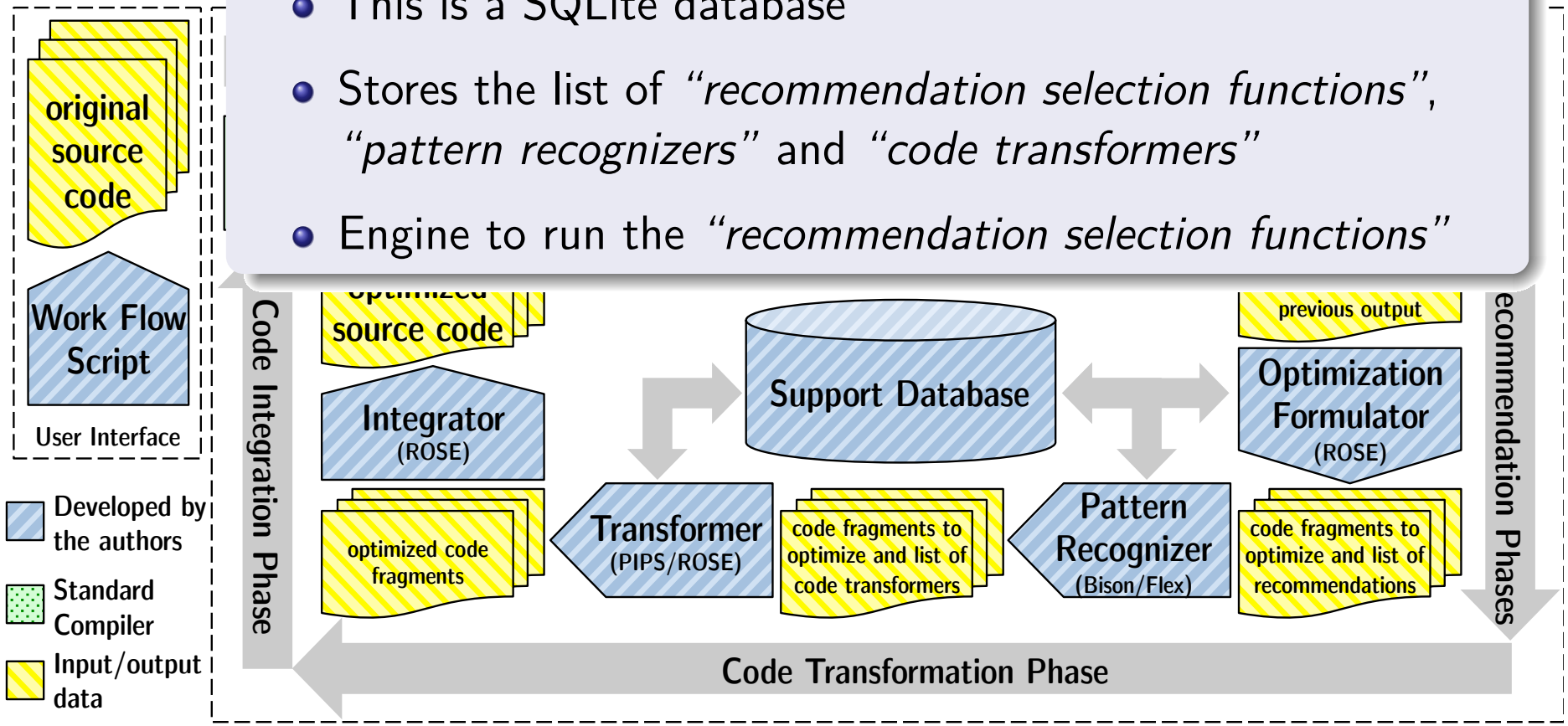
PerfExpert: Optimization Formulator



- Loads performance metrics on the Support Database
- Runs all *"recommendation selection functions"*
- Concatenates and ranks the list of recommendations
- Extracts code fragments identified as bottlenecks
- Based on ROSE
- **Extendable:** accepts user-defined performance metrics
- **Extendable:** it is possible to write new *"recommendation selection functions"* (SQL query)

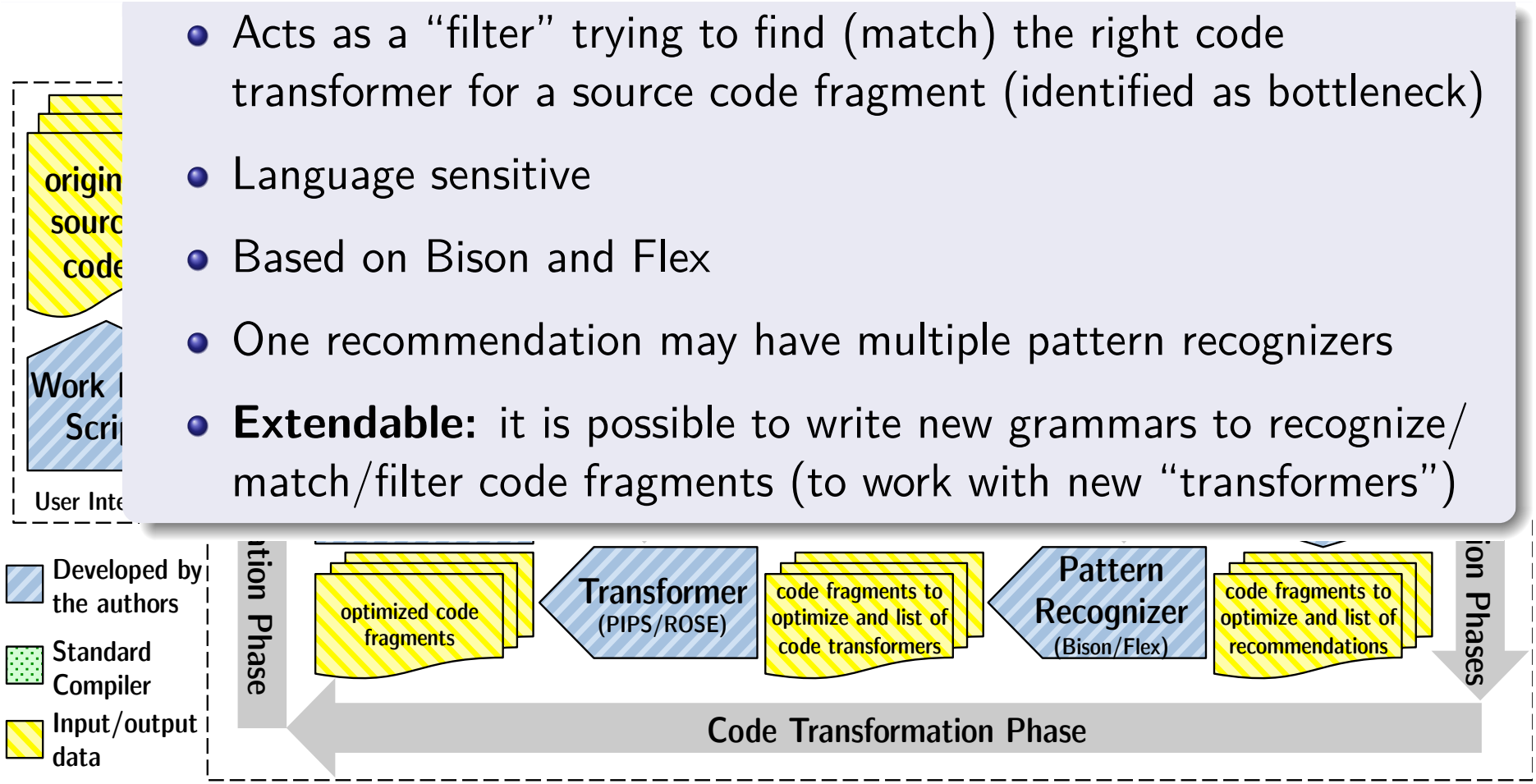
PerfExpert: Support Database

- This is a SQLite database
- Stores the list of *"recommendation selection functions"*, *"pattern recognizers"* and *"code transformers"*
- Engine to run the *"recommendation selection functions"*



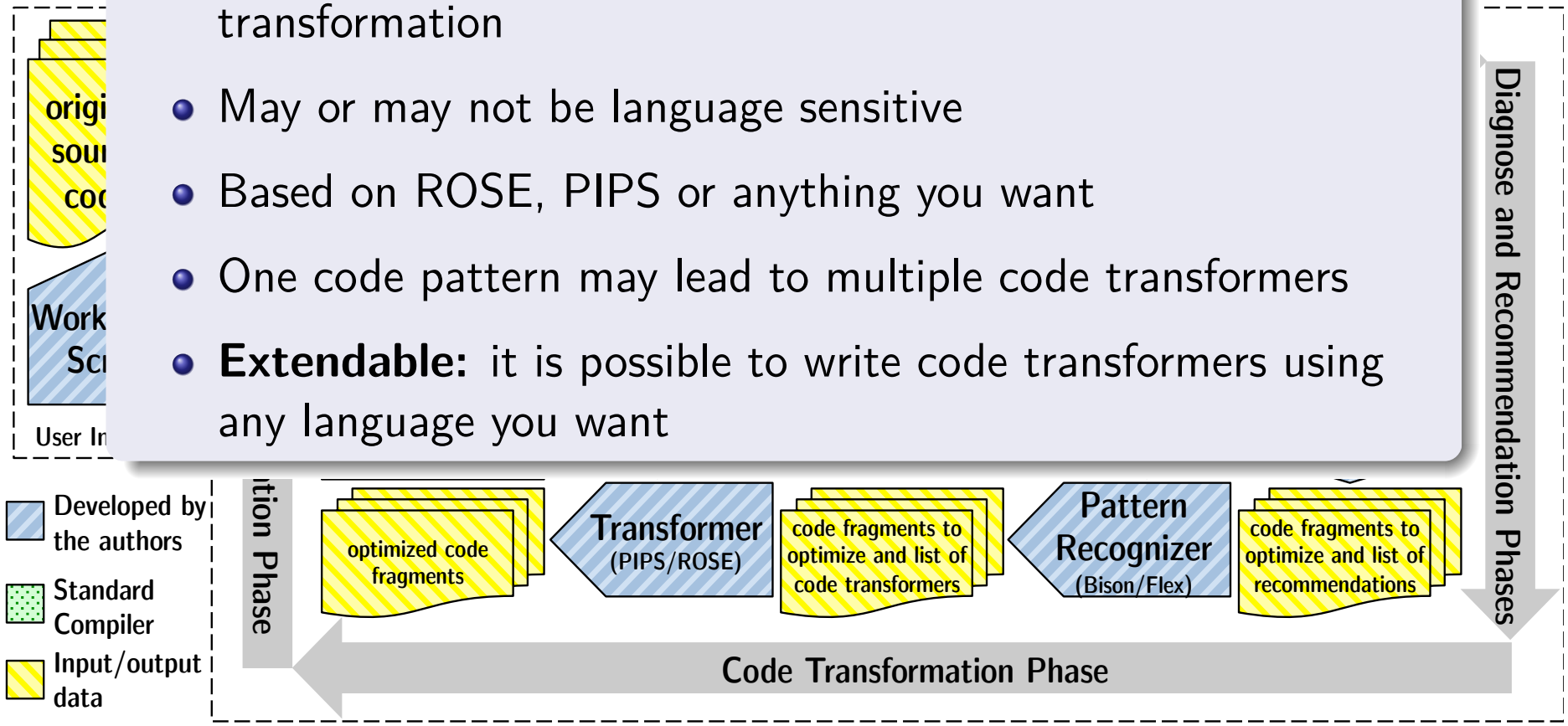
PerfExpert: Pattern Recognizer

- Acts as a "filter" trying to find (match) the right code transformer for a source code fragment (identified as bottleneck)
- Language sensitive
- Based on Bison and Flex
- One recommendation may have multiple pattern recognizers
- **Extendable:** it is possible to write new grammars to recognize/ match/filter code fragments (to work with new "transformers")

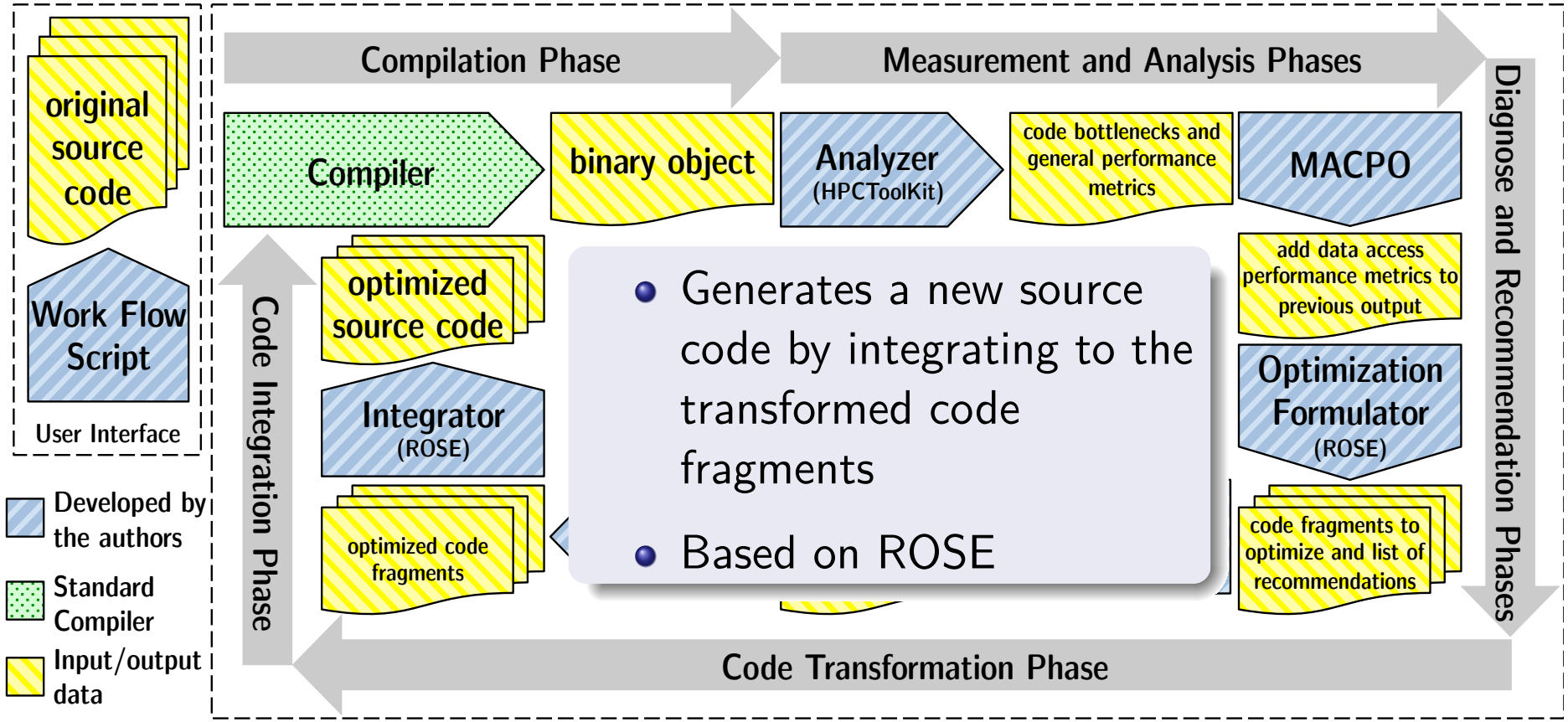


PerfExpert: Transformer

- Implements the recommendation by applying source code transformation
- May or may not be language sensitive
- Based on ROSE, PIPS or anything you want
- One code pattern may lead to multiple code transformers
- **Extendable:** it is possible to write code transformers using any language you want

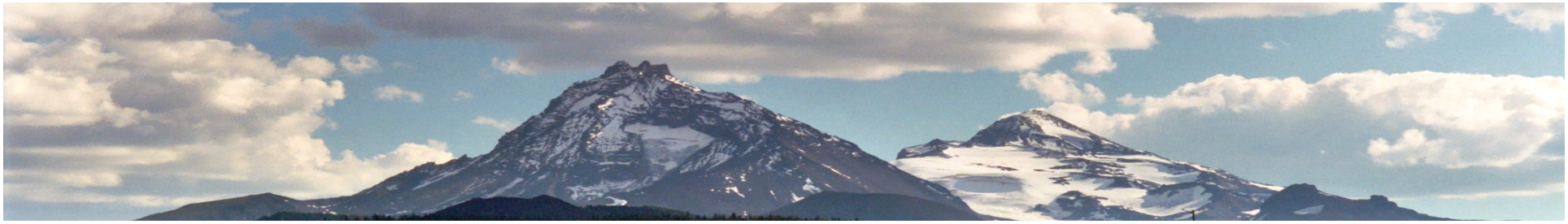


PerfExpert: Integrator



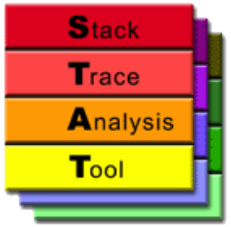
PerfExpert: Key Points for Design

- Each piece of the tool chain can be updated/ upgraded individually
- It is flexible: you can add new metrics as well as plug new tools to measure application performance
- It is extendable: new recommendations, transformations and strategies to select recommendations
- Multi-language, multi-architecture, open-source and built on top of well-established tools (HPCToolKit, ROSE, PIPS, etc.)
- Easy to use and lightweight



STAT: Stack Trace Analysis Tool

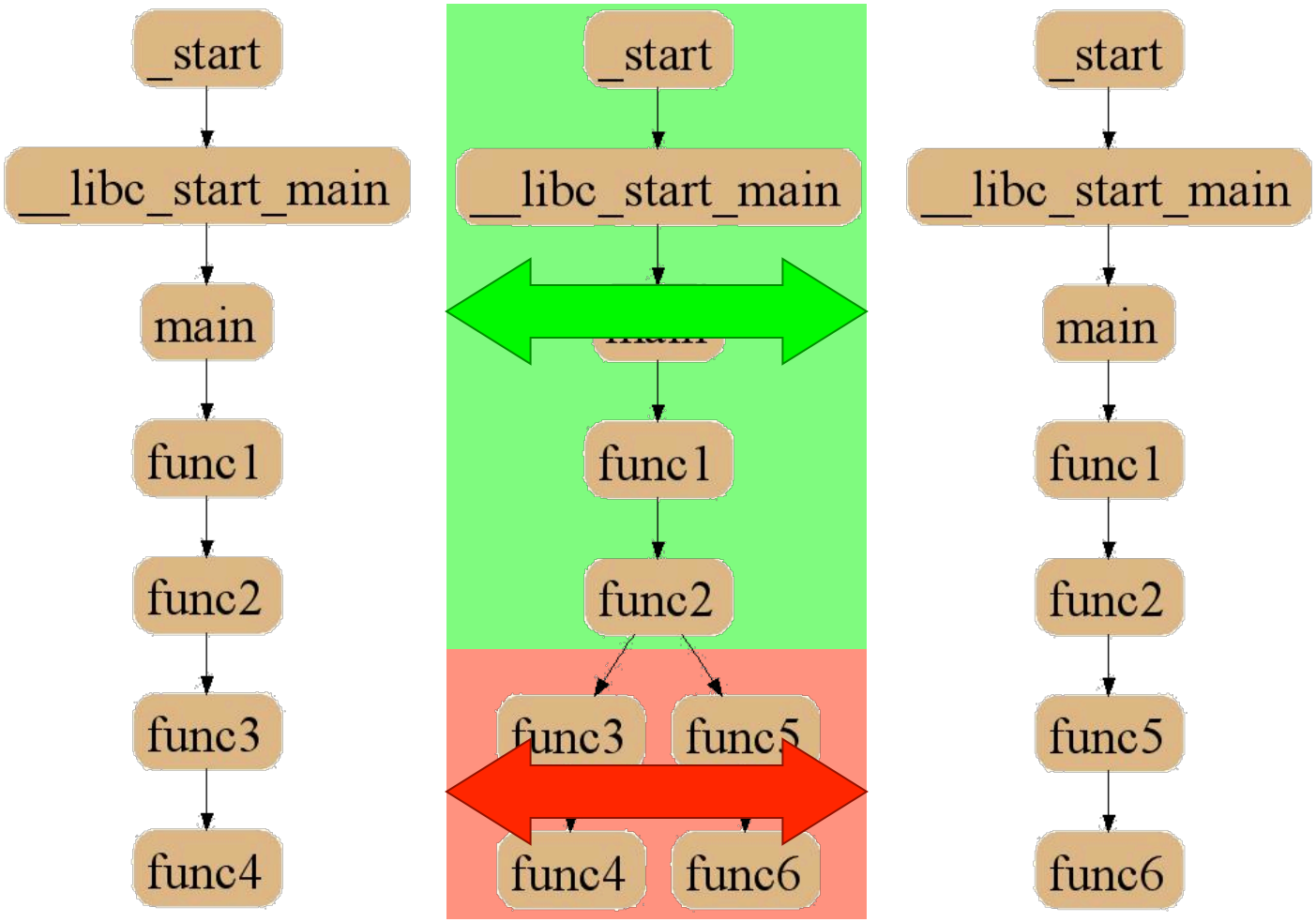
Martin Schulz, Bart Miller, Dorian Arnold
Lawrence Livermore National Laboratory (USA)
University of Wisconsin (USA)
University of New Mexico (USA)



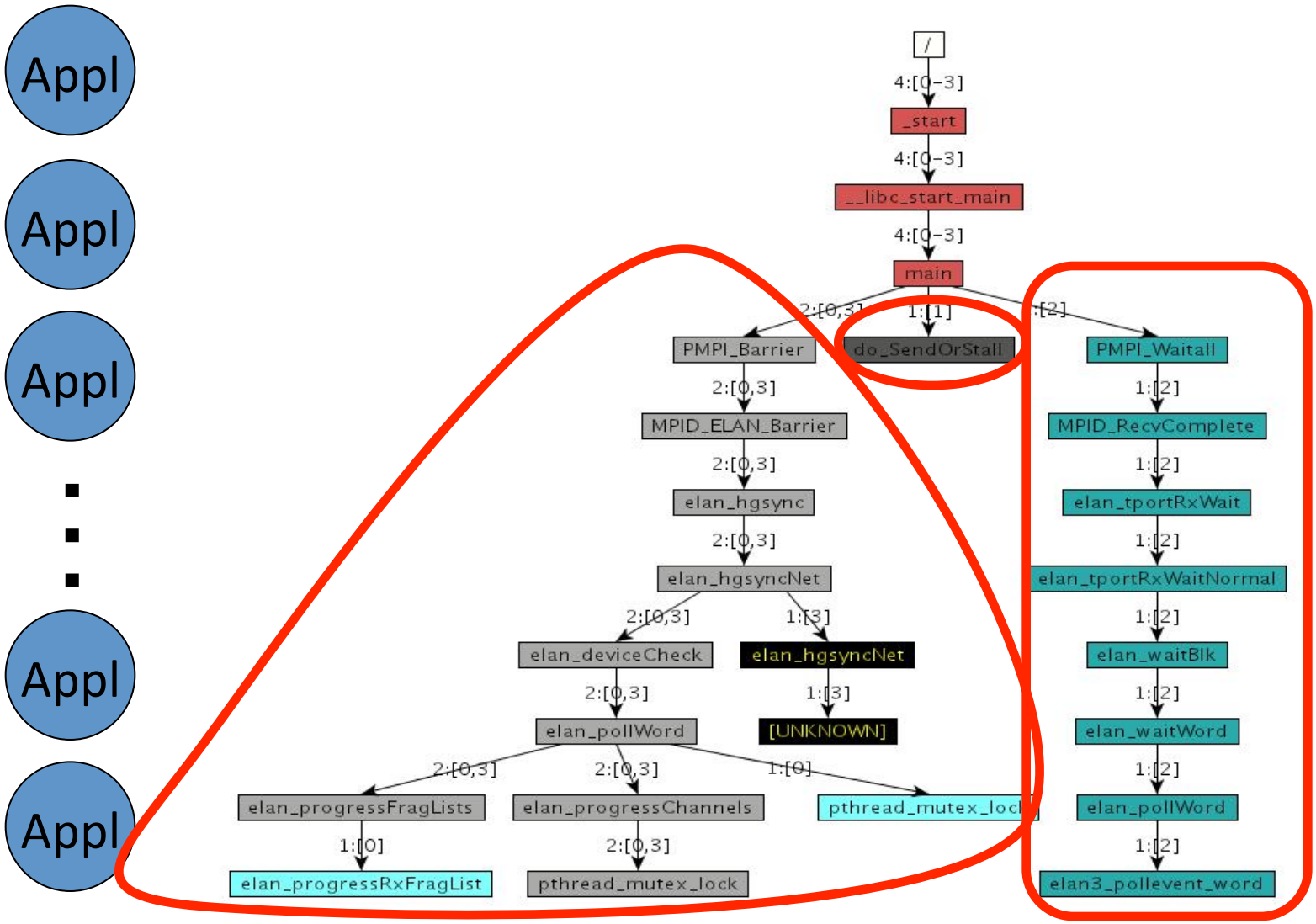
STAT: Aggregating Stack Traces for Debugging

- Existing debuggers don't scale
  - Inherent limits in the approaches
  - Need for new, scalable methodologies
- Need to pre-analyze and reduce data
  - Fast tools to gather state
  - Help select nodes to run conventional debuggers on
- Scalable tool: STAT
  - Stack Trace Analysis Tool
  - Goal: Identify equivalence classes
  - Hierarchical and distributed aggregation of stack traces from all tasks
  - Stack trace merge <1s from 200K+ cores

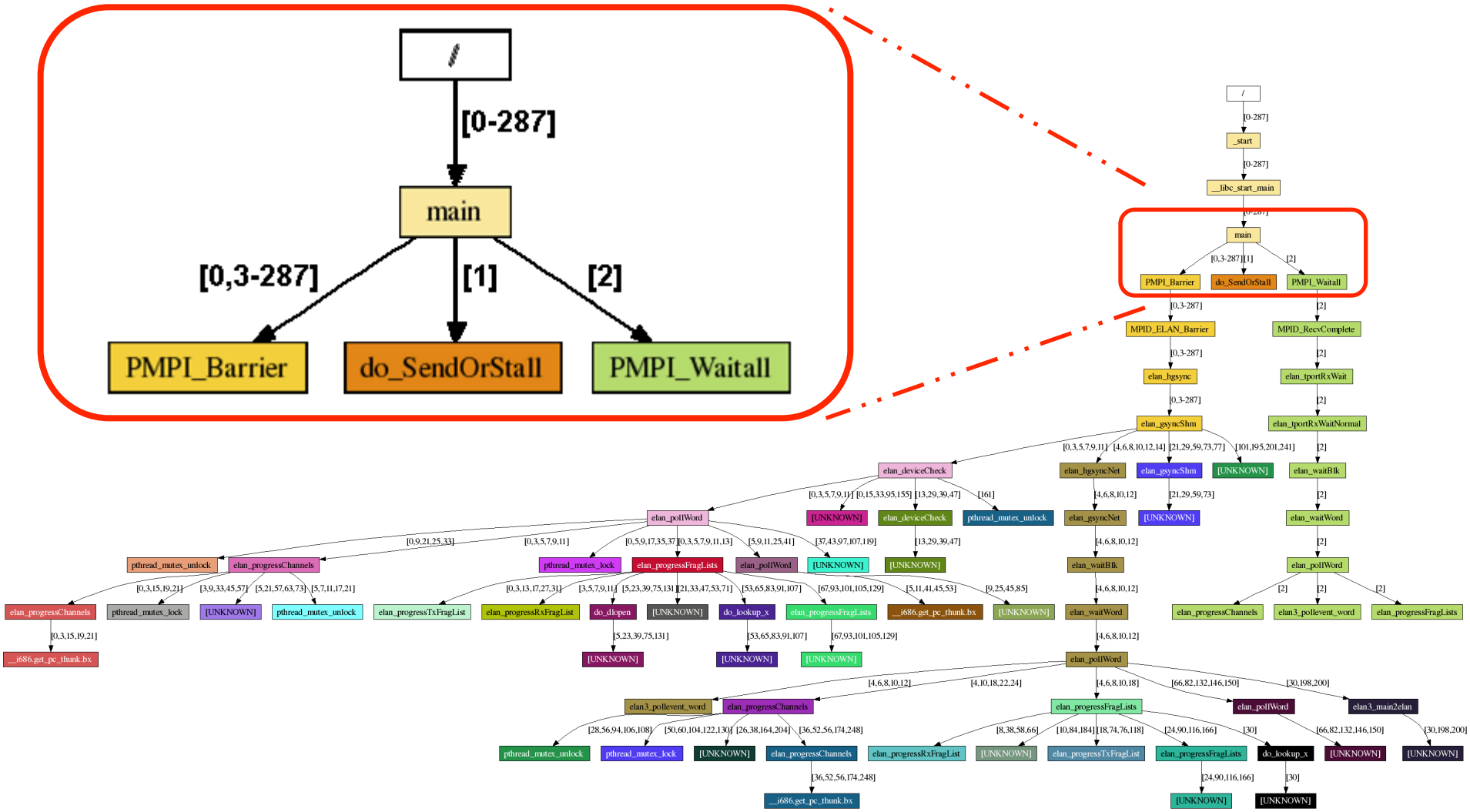
Distinguishing Behavior with Stack Traces

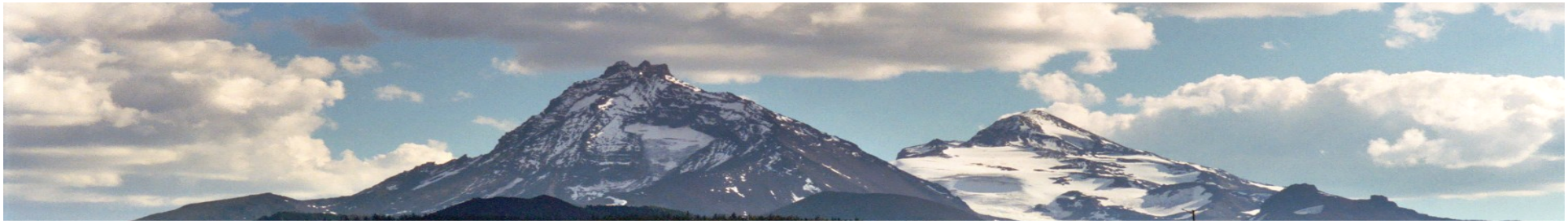


3D-Trace Space/Time Analysis



Scalable Representation





Active Harmony

Jeff Hollingsworth
University of Maryland (USA)
University of Wisconsin (USA)

http://dyninst.org/harmony

Why Automate Performance Tuning?

- Too many parameters that impact performance.
- Optimal performance for a given system depends on:
  - Details of the processor
  - Details of the inputs (workload)
  - Which nodes are assigned to the program
  - Other things running on the system
- Parameters come from:
  - User code
  - Libraries
  - Compiler choices

Automated Parameter tuning can be used for adaptive tuning in complex software.

Automated Performance Tuning

- Goal: Maximize achieved performance
- Problems:
  - Large number of parameters to tune
  - Shape of objective function unknown
  - Multiple libraries and coupled applications
  - Analytical model may not be available
  - Shape of objective function could change during execution!
- Requirements:
  - Runtime tuning for long running programs
  - Don’t run too many configurations
  - Don’t try really bad configurations

I really mean don't let people hand tune

- Example, a dense matrix multiply kernel
- Various Options:
  - Original program: 30.1 sec
  - Hand Tuned (by developer): 11.4 sec
  - Auto-tuned of hand-tuned: 15.9 sec
  - Auto-tuned original program: 8.5 sec
- What Happened?
  - Hand tuning prevented analysis
    - Auto-tuned transformations were then not possible

Active Harmony

- Runtime performance optimization
  - Can also support training runs
- Automatic library selection (code)
  - Monitor library performance
  - Switch library if necessary
- Automatic performance tuning (parameter)
  - Monitor system performance
  - Adjust runtime parameters
- Hooks for Compiler Frameworks
  - Integrated with USC/ISI Chill

Basic Client API Overview

- Only five core functions needed
  - Initialization/Teardown (4 functions)

```
hdesc_t *harmony_init (appName, iomethod)
int harmony_connect (hdesc, host, port)
int harmony_bundle_file (hdesc, filename)
harmony_bind_int(hdesc, varName, &ptr)
```

  - Critical loop analysis (2 functions)

```
int harmony_fetch (hdesc)
int harmony_report (hdesc, performance)
```

  - Available for C, C++, Fortran, and Chapel

Harmony Bundle Definitions

- External bundle definition file

```
[rchen@brood00 code_generation]$ cat bundle.cfg
harmonyApp gemm {
  { harmonyBundle TI { int {2 500 2} global } }
  { harmonyBundle TJ { int {2 500 2} global } }
  { harmonyBundle TK { int {2 500 2} global } }
  { harmonyBundle UI { int {1   8 1} global } }
  { harmonyBundle UJ { int {1   8 1} global } }
}
```

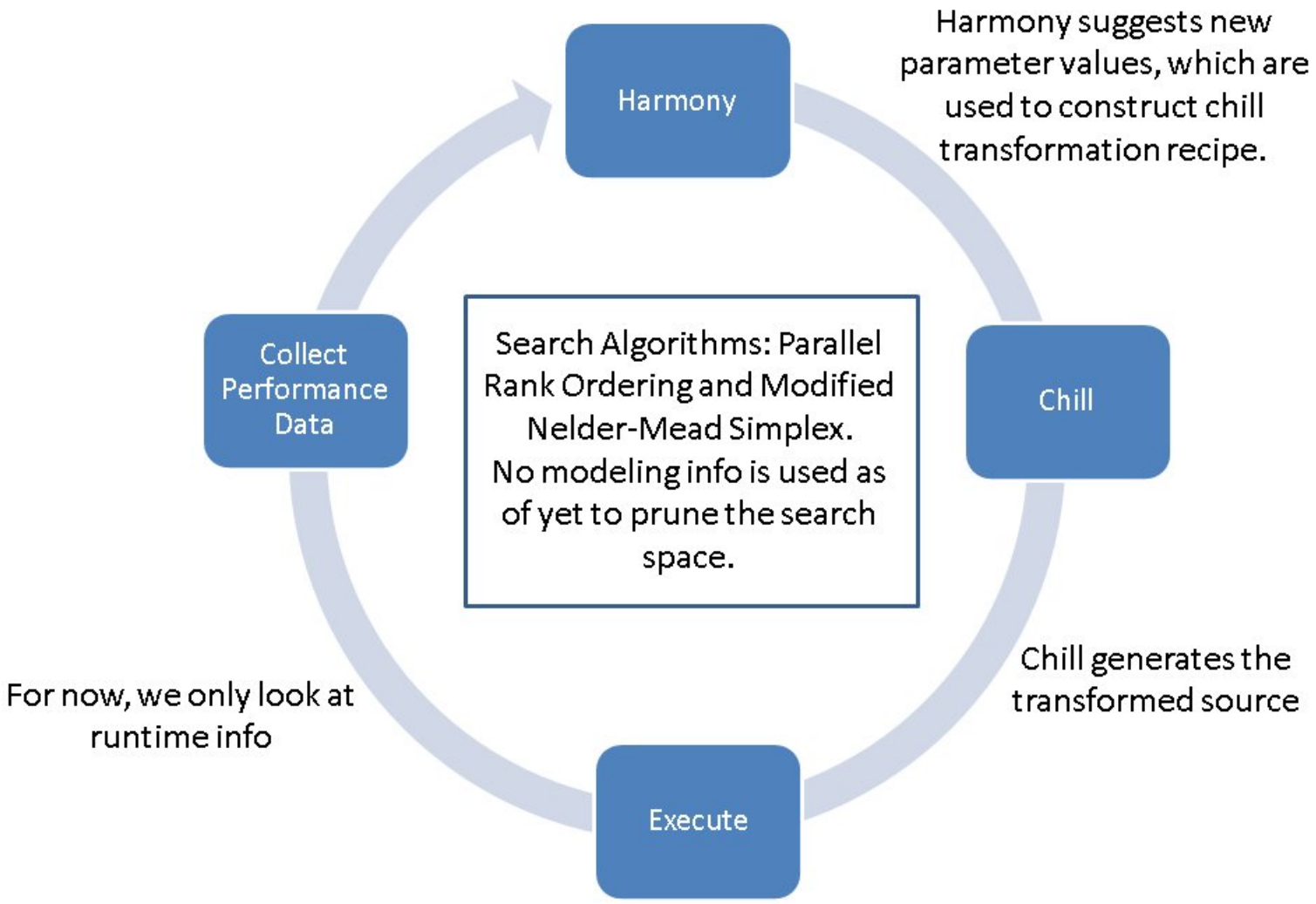
Bundle Name Value Class Value Range Bundle Scope

- Equivalent client API routines available

A Bit More About Harmony Search

- Pre-execution
  - Sensitivity Discovery Phase
  - Used to **Order** not **Eliminate** search dimensions
- Online Algorithm
  - Use Parallel Rank Order Search
    - Variation of Rank Oder Algorithm
      - Part of the class of Generating Set Algorithms
    - Different configurations on different nodes

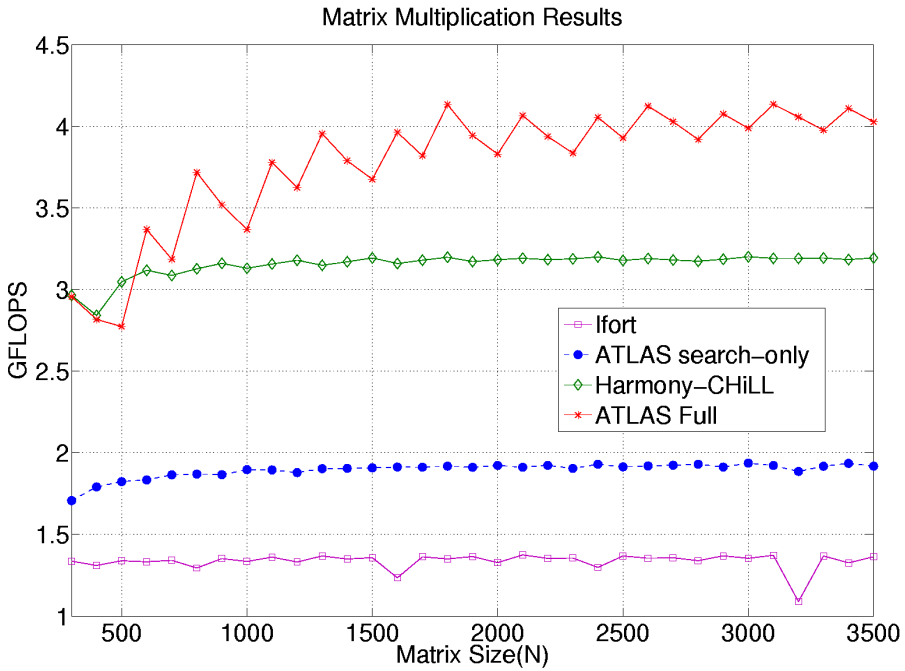
Integrating Compiler Transformations

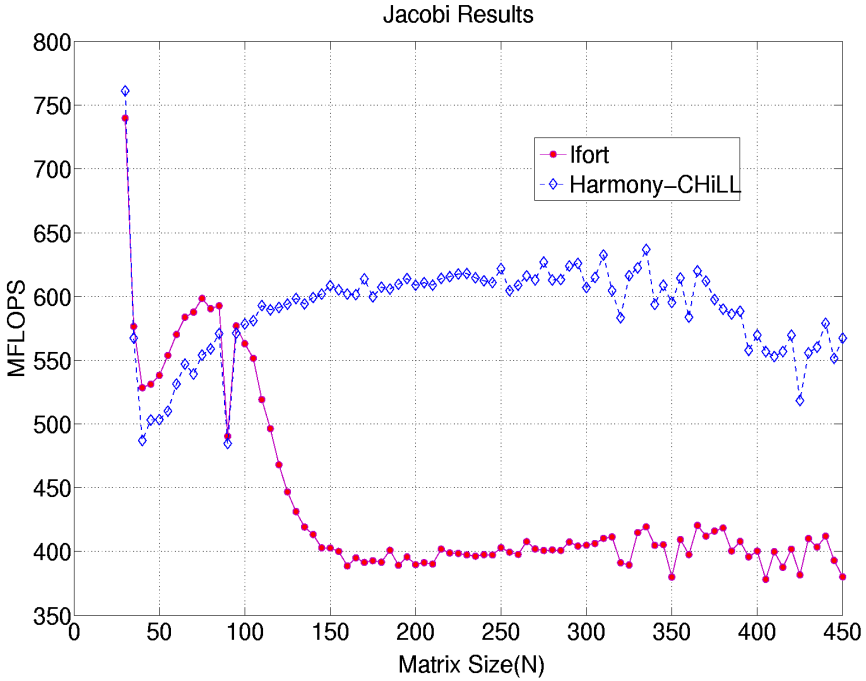


Auto-tuning Compiler Transformations

- Described as a series of Recipes
- Recipes consist of a sequence of operations
  - Permute([stmt],order): change the loop order
  - Tile(stmt,loop,size,[outer-loop]): tile loop at level loop
  - Nnroll(stmt,loop,size): unroll stmt's loop at level loop
  - Datacopy(stmt,loop,array,[index]):
    - Make a local copy of the data
  - Split(stmt,loop,condition): split stmt's loop level loop into multiple loops
  - Nonsingular(matrix)

CHiLL + Active Harmony

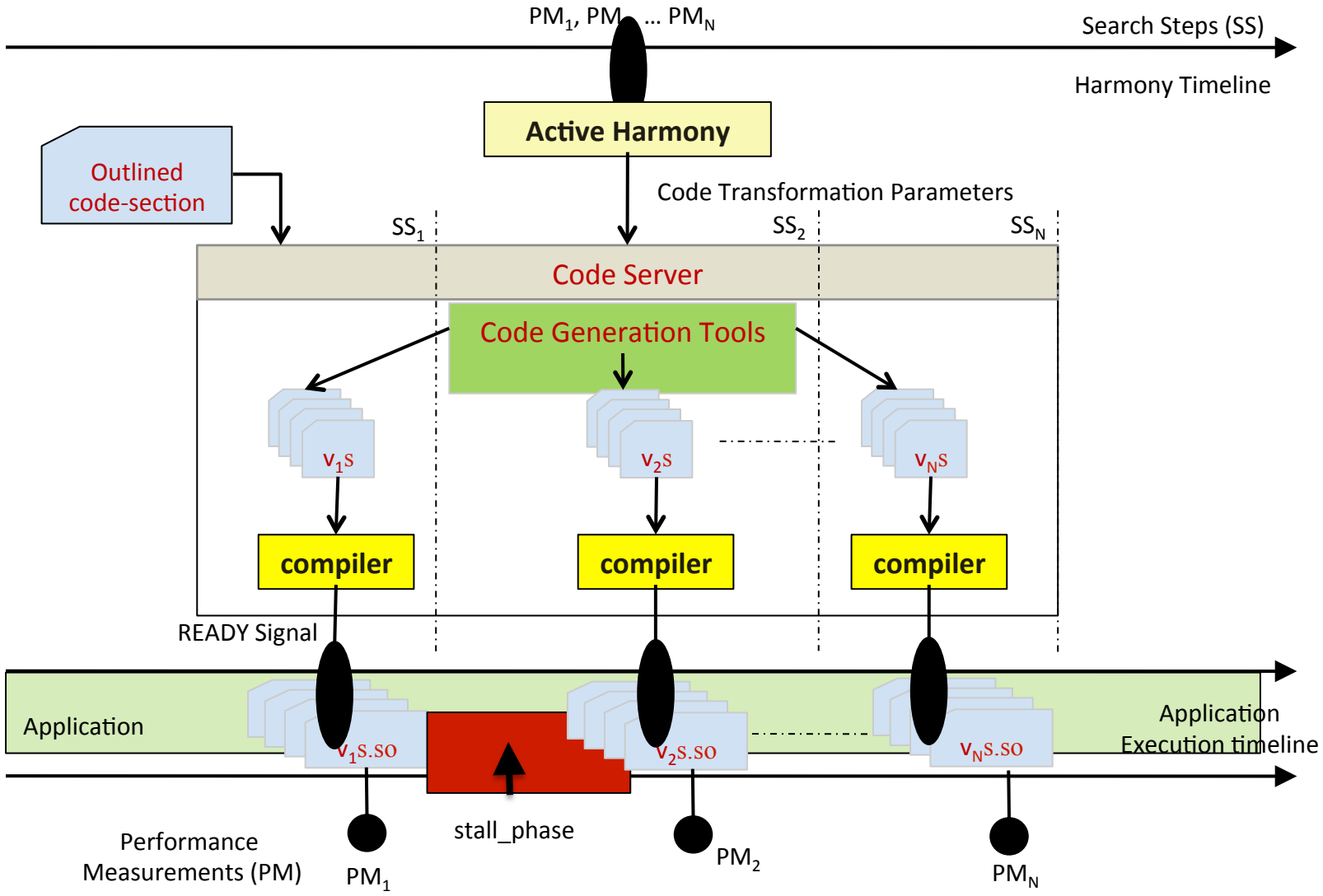




Generate and evaluate different optimizations that would have been prohibitively time consuming for a programmer to explore manually.

Ananta Tiwari, Chun Chen, Jacqueline Chame, Mary Hall, Jeffrey K. Hollingsworth, "A Scalable Auto-tuning Framework for Compiler Optimization," IPDPS 2009, Rome, May 2009.

Compiling New Code Variants at Runtime



Auto-Tuning for Communication

- Optimize overlap
  - Hide communication behind as much computation as possible
  - Tune how often to call poll routines
    - Messages don't move unless call into MPI library
    - Calling too often results in slowdown
- Portability
  - No support from special hardware for overlap
- Optimize local computation
  - Improve cache performance with loop tiling
- Parameterize & Auto-Tuning
  - Cope with the complex trade-off regarding the optimization techniques