# PerfExpert

**Jim Browne, Ashay Rane and Leo Fialho**

## Petascale Tools Workshop
## Madison WI, 2013


# Agenda



# Overview: why PerfExpert?

## Problem: HPC systems operate far below peak

- Chip/node architectural complexity is growing rapidly
- Performance optimization for these chips requires deep knowledge of architectures, code patterns, compilers, etc.

## Performance optimization tools

- Powerful in the hands of experts
- Require detailed performance and system expertise
- HPC application developers are domain experts, not computer gurus

## Many HPC programmers/users do not use your tools

(seriously)

# Goal for PerfExpert: democratize optimization!

## Subgoals:

- Make use of the tool as simple as possible
- Start with only chip/node level optimization
- Make it adaptable across multiple architectures

## How to accomplish?

- Formulate the performance optimization task as a workflow of subtasks
- Leverage the state-of-the-art: build on the best available tools for the subtasks to minimize the effort and cost of development
- Automate the entire workflow

# Introduction

## The four stages of automatic performance optimization:

- Measurement and attribution (1)
- Analysis, diagnosis and identification of bottlenecks (2)
- Selection of effective optimizations (3)
- Implementation of optimizations (4)

## Use of State-of-the-Art:

- HPCToolkit/Intel VTune, **MACPO** based on ROSE (1)
- **PerfExpert Team** (2 and 3)
- **PerfExpert Team** based on ROSE, PIPS, Bison and Flex (4)

TACC THE UNIVERSITY OF TEXAS AT AUSTIN

# Introduction

## Uniqueness of PerfExpert:

- Nearly complete optimization first three stages of optimization for chip/node level
- Framework for implementing optimizations is complete and several optimizations are completed
- Integrates code segment focused and data structure based measurements (**MACPO**)
  - Code segment local measurement
  - Data structure specific traces
  - More accurate (associative) cache models
  - Strides by data structure and code segment
  - Architecture "independent" metrics

# What can PerfExpert provide to you?

## Performance report:

- Identification of bottlenecks by relevance
- Performance analysis based on performance metrics
- Recommendations for optimization

## There are three possible outputs:

- Performance report only
- List of recommendations
- Fully automated code transformation

TACC THE UNIVERSITY OF TEXAS AT AUSTIN

# Performance Report

```
Loop in function compute() at mm.c:8 (99.8% of the total runtime)
================================================================================
ratio to total instrns       % 0.........25...........50.........75........100
   - floating point      :    100 ********************************************
   - data accesses       :     25 ************
* GFLOPS (% max)         :     12 ******
   - packed              :      0 *
   - scalar              :     12 ******
--------------------------------------------------------------------------------
performance assessment      LCPI good.....okay.....fair.....poor.....bad....
* overall                :    3.0 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
upper bound estimates
* data accesses          :    9.6 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
   - L1d hits            :    0.9 >>>>>>>>>>>>>>>>>
   - L2d hits            :    1.8 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
   - L2d misses          :    6.9 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
* instruction accesses   :    0.1 >
   - L1i hits            :    0.0 >
   - L2i hits            :    0.0 >
   - L2i misses          :    0.1 >
* data TLB               :    4.6 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
* instruction TLB        :    0.0 >
* branch instructions    :    0.1 >>
   - correctly predicted :    0.1 >>
   - mispredicted        :    0.0 >
* floating-point instr   :    5.1 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
   - fast FP instr       :    5.1 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
   - slow FP instr       :    0.0 >
```

# List of Recommendations

```
#---------------------------------------------
# Recommendations for mm.c:8
#---------------------------------------------
# This is a possible recommendation for this code segment
#
Description:  change the order of loops
Reason:  this optimization may improve the memory access
pattern and make it more cache and TLB friendly
Pattern Recognizers:  c_loop2 f_loop2
Code example:
loop i {
  loop j {...}
}
=====> loop j {
  loop i {...}
}
```

# Fully Automated Code Transformation

## Before:

```
void compute() {
 register int i, j, k;




 for (i = 0; i < 1000; i++)


  for (j = 0; j < 1000; j++)

   for (k = 0; k < 1000; k++)
    c[i][j] += (a[i][k] * b[k][j]);
}
```

## After:

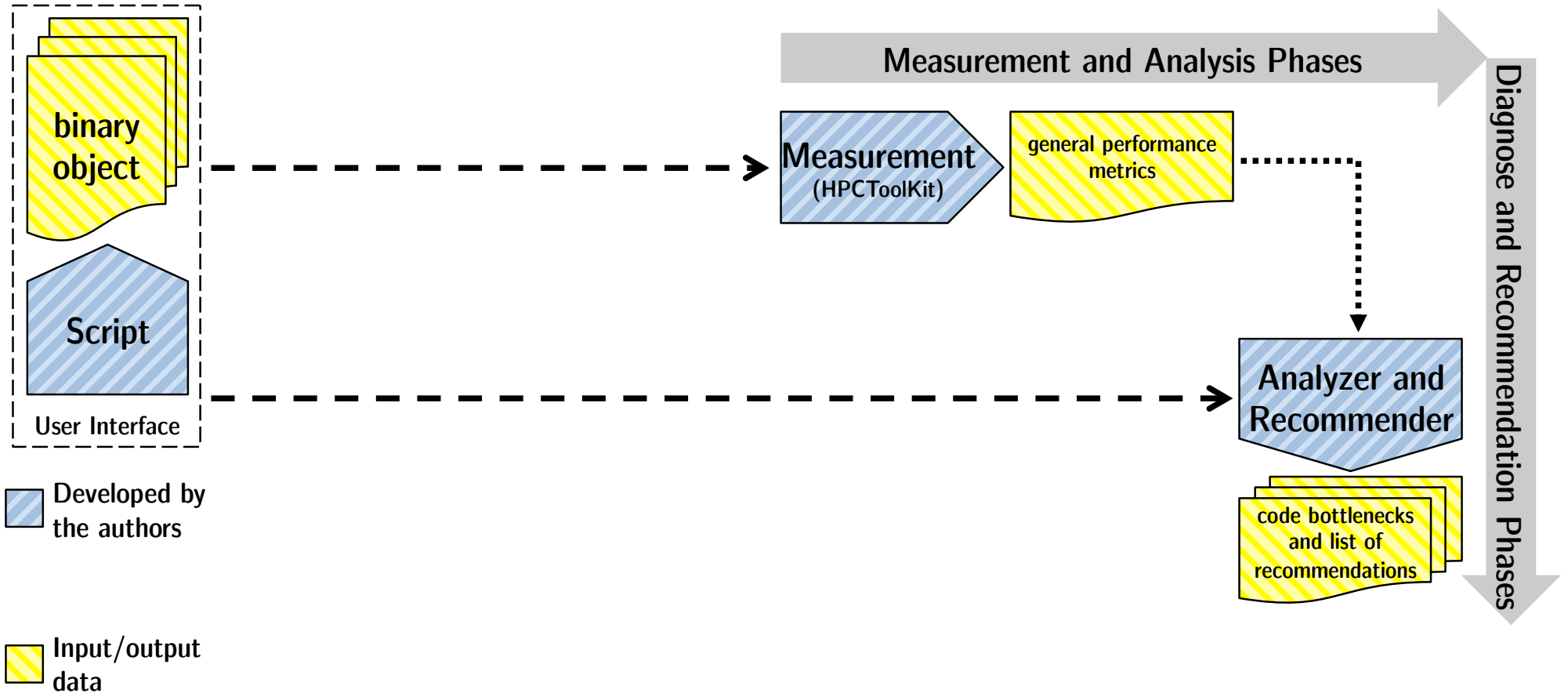
```
void compute() {
 register int i, j, k;
 //PIPS generated variable
 register int jp, kp;
 /* PERFEXPERT: start work here */
 /* PERFEXPERT: grandparent loop */
 loop_6:
 for (i = 0; i <= 999; i++)
  /* PERFEXPERT: parent loop */
  loop_7:
  for(jp = 0; jp <= 999; jp += 1)
   /* PERFEXPERT: bottleneck */
   for(kp = 0; kp <= 999; kp += 1)
   c[i][kp] += a[i][jp]*b[jp][kp];
}
```

# Agenda

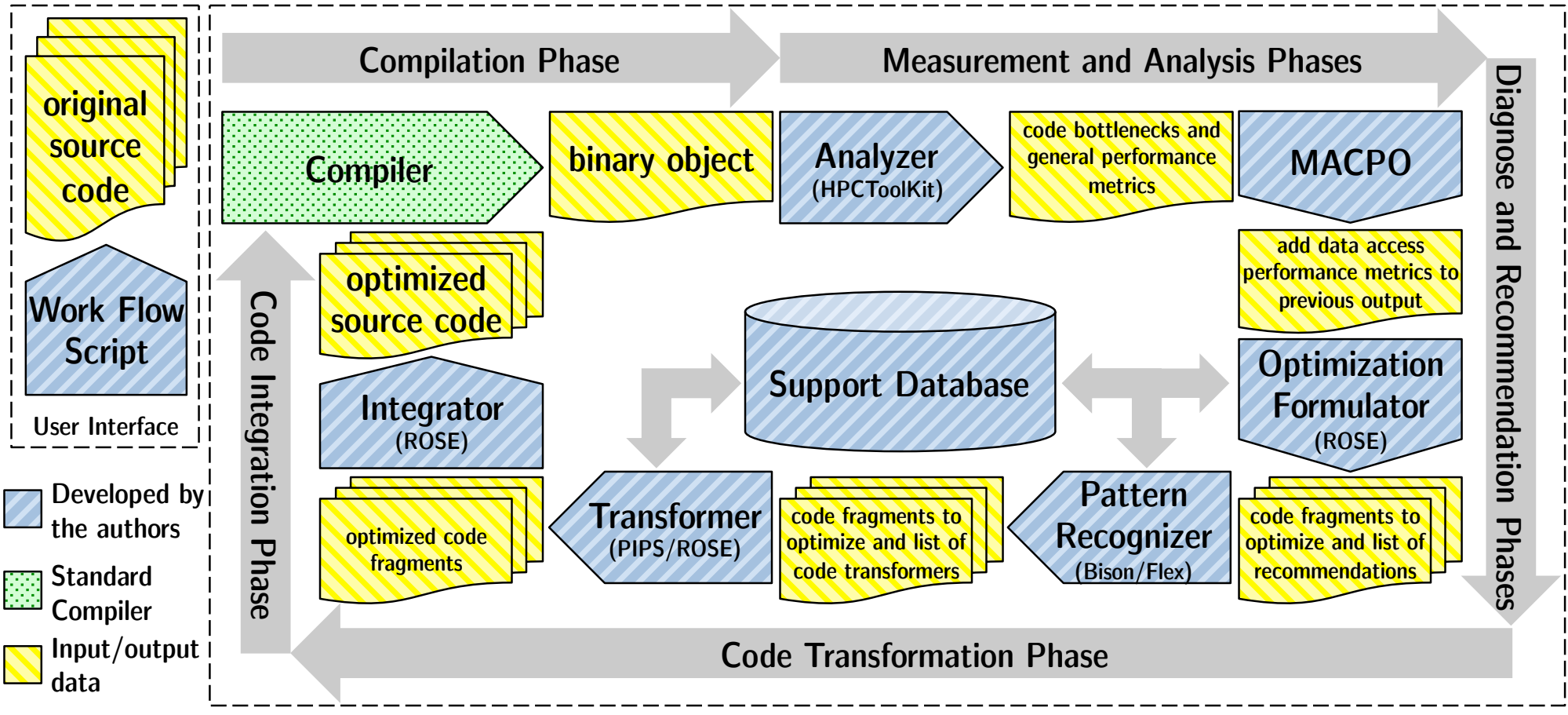


TACC THE UNIVERSITY OF TEXAS AT AUSTIN

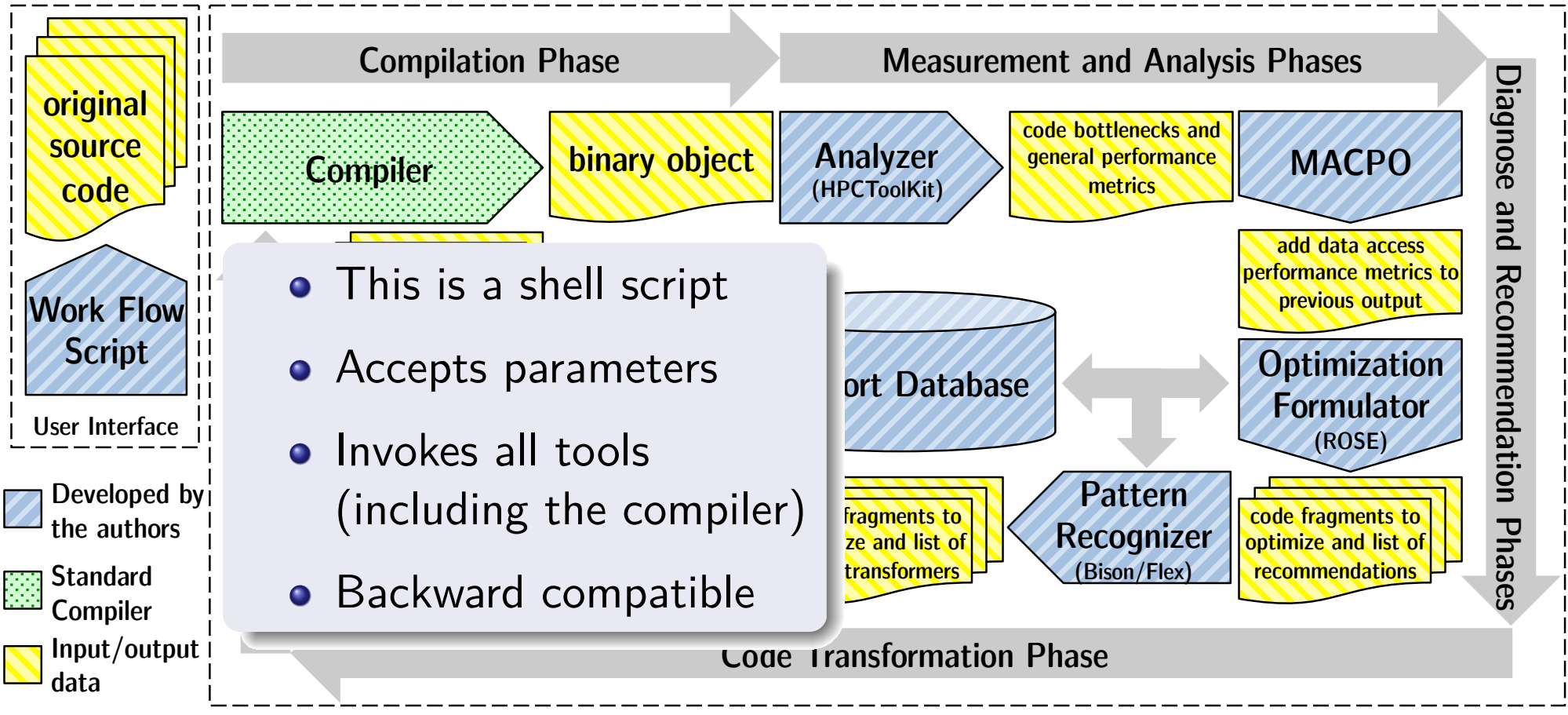
# Current Version: The Big Picture



# New Version: The Big Picture

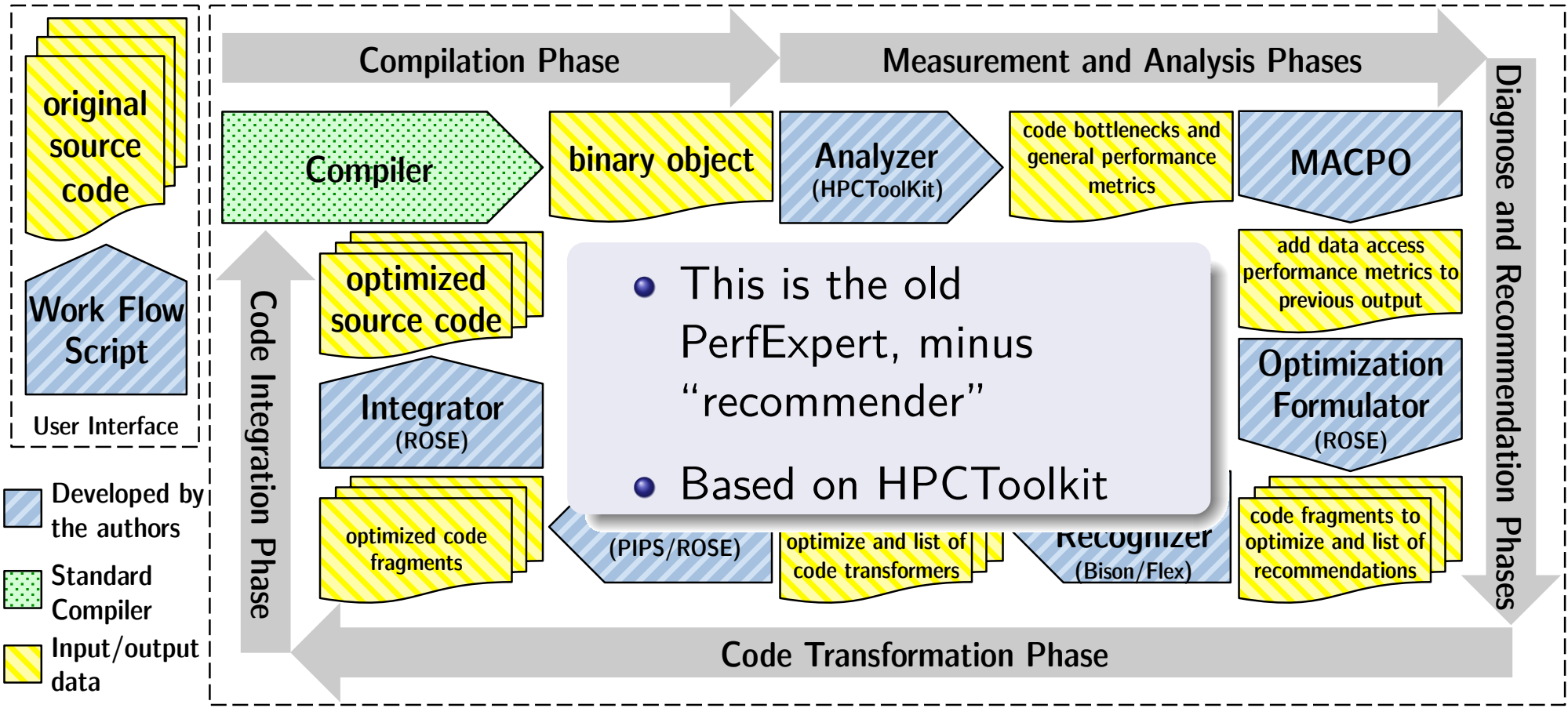


# New Version: Work Flow Script

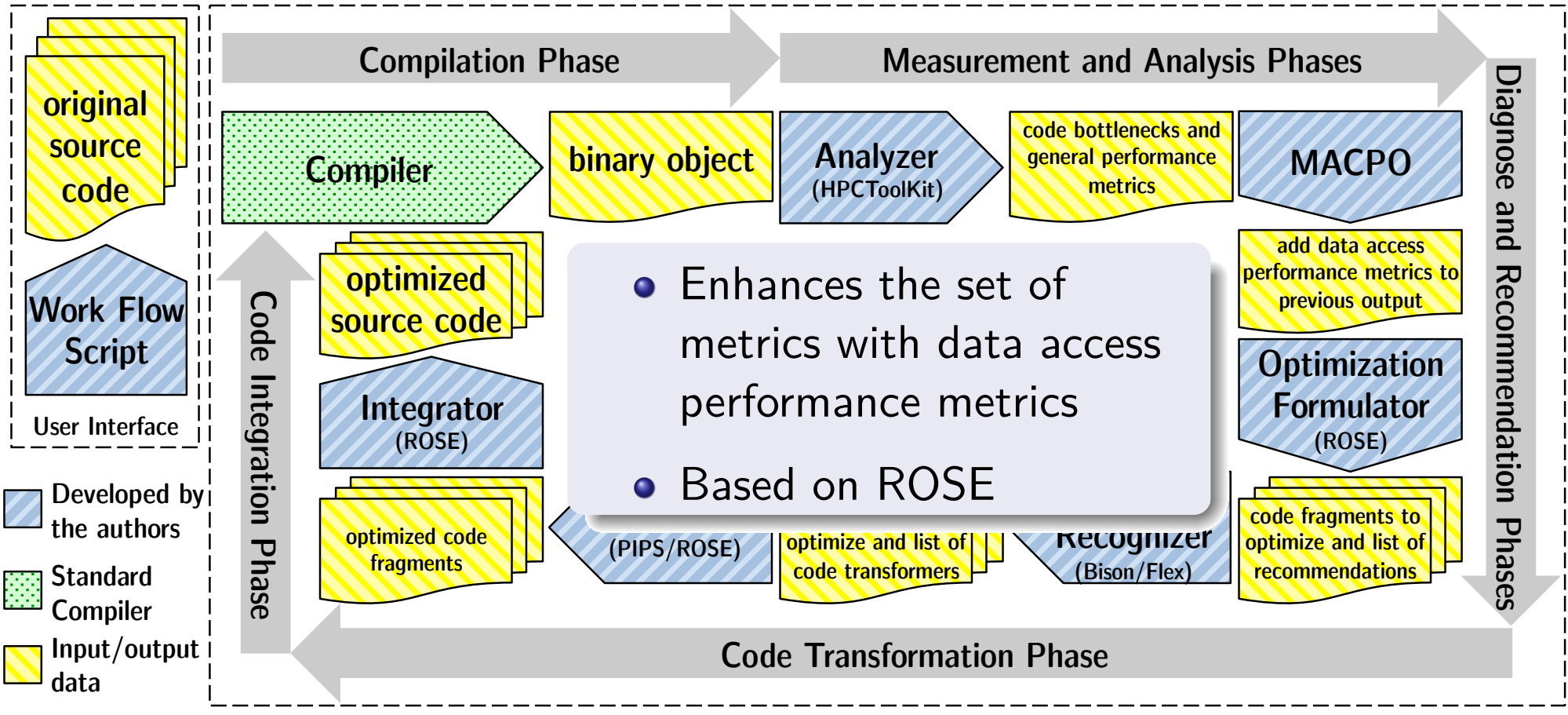



- This is a shell script
- Accepts parameters
- Invokes all tools (including the compiler)
- Backward compatible

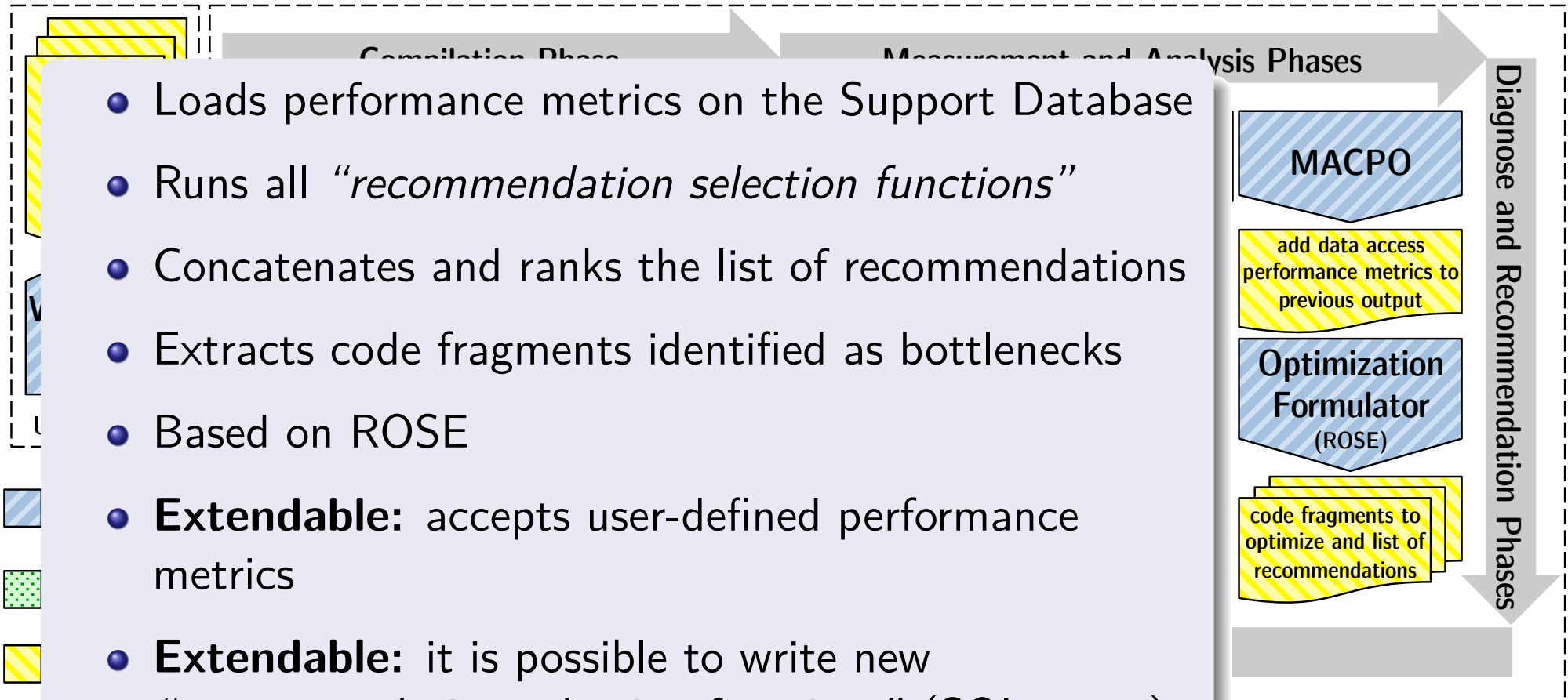
# New Version: Analyzer



# New Version: MACPO



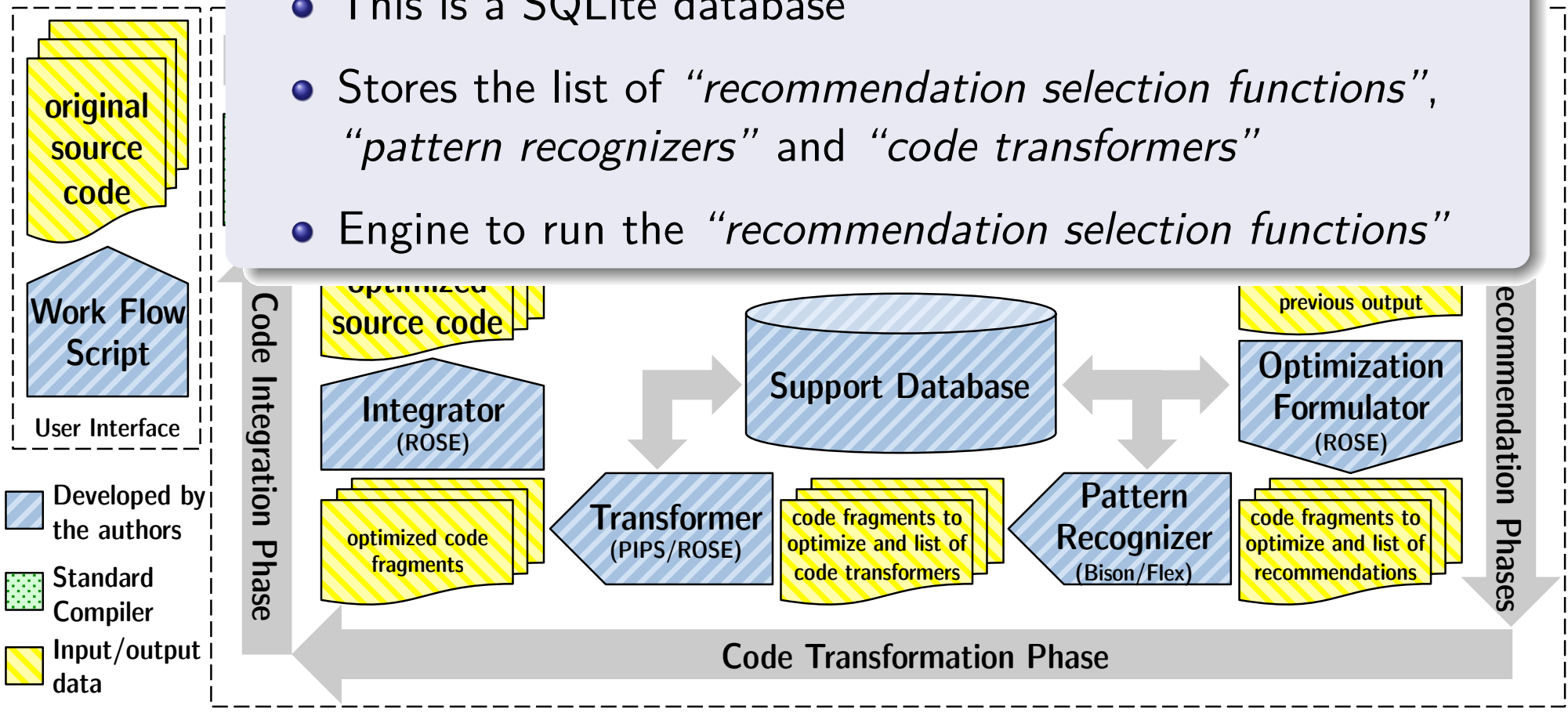
# New Version: Optimization Formulator




- 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)

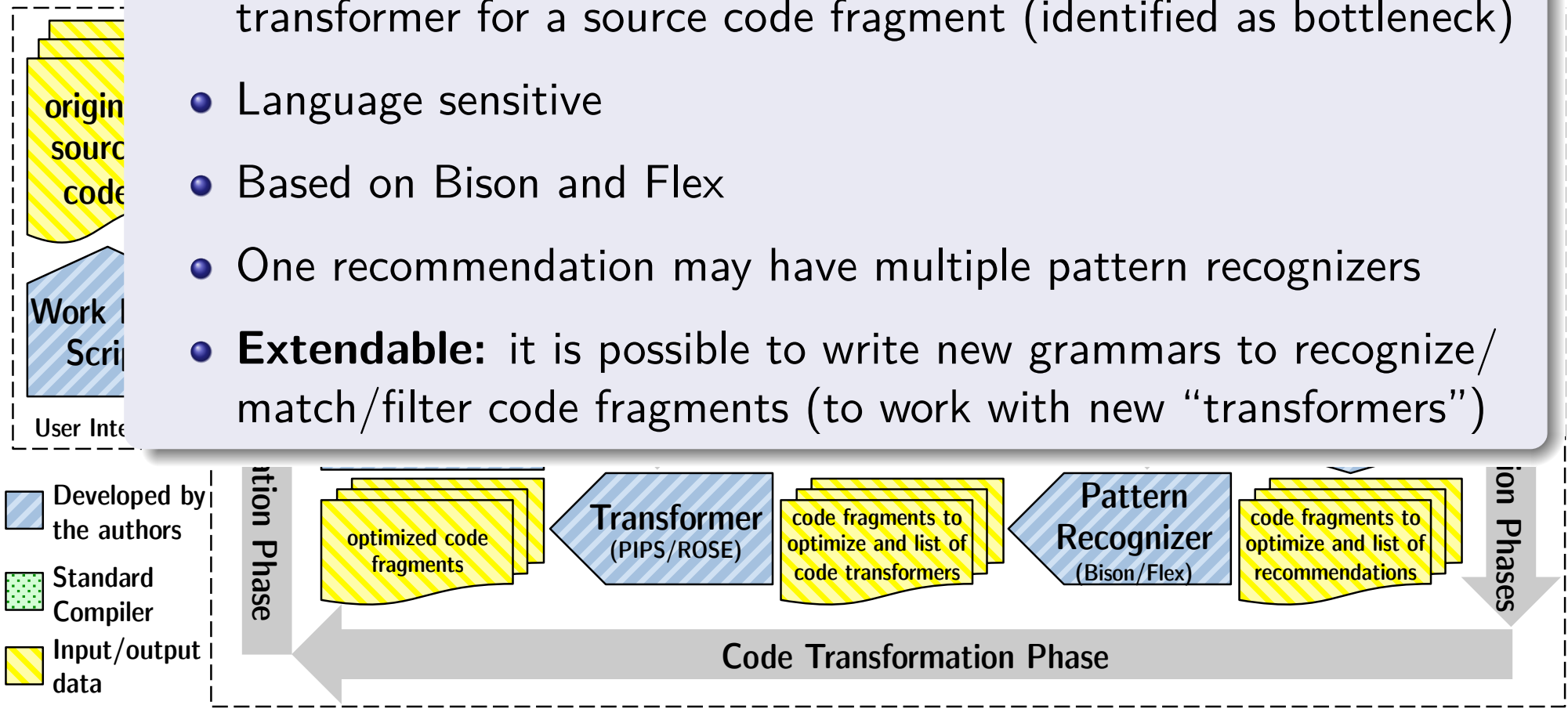
# New Version: Support Database

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



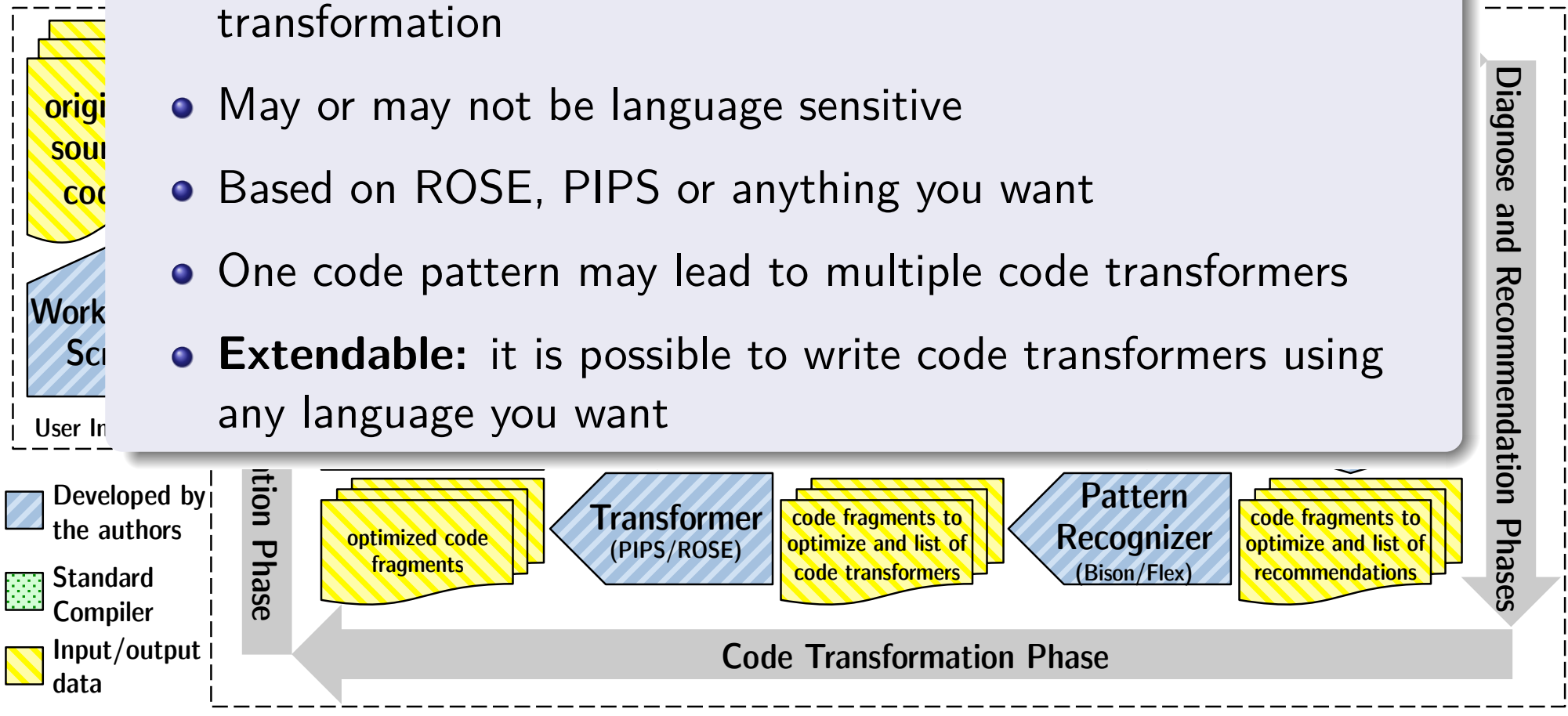
# New Version: Pattern Recognizer

- 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")

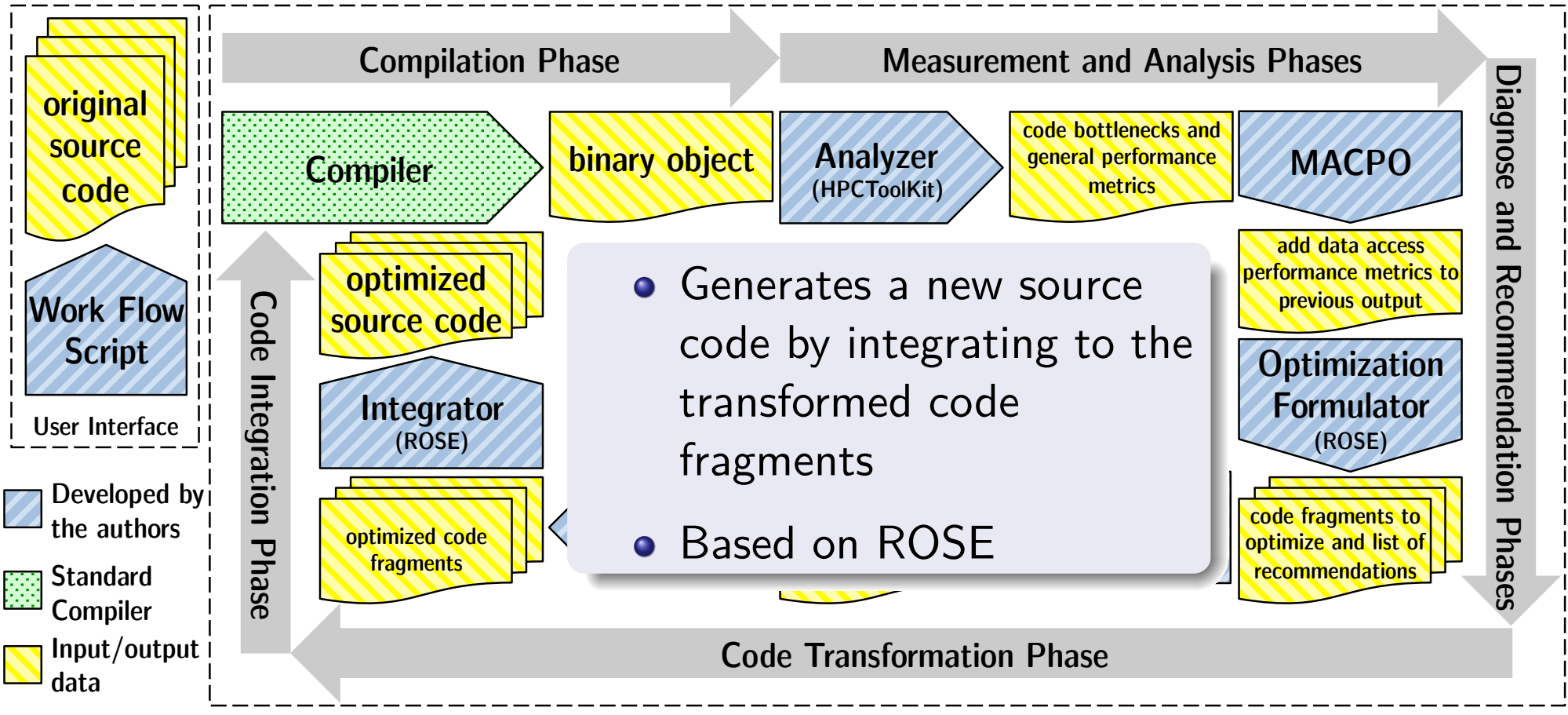


# New Version: Transformer

- 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



# New Version: Integrator



# Agenda



# Understanding PerfExpert Analysis

## On the The Analysis Report...

- The more "expensive" comes first
- Tells user where the slow code sections are as well as why they perform poorly
- Every function or loop which takes more than 1% of the execution time is analyzed (default value)
- Yes, we rely on performance metrics (but not only and not the raw ones)
- No, we do not rely on hardware specs
- If you are not using properly the node PerfExpert may conclude everything is fine (use a representative workload)

# Performance Report

```
Loop in function compute() at mm.c:8 (99.8% of the total runtime)
================================================================================
ratio to total instrns       % 0.........25...........50.........75........100
   - floating point      :    100 ********************************************
   - data accesses       :     25 ************
* GFLOPS (% max)         :     12 ******
   - packed              :      0 *
   - scalar              :     12 ******
--------------------------------------------------------------------------------
performance assessment      LCPI good.....okay.....fair.....poor.....bad....
* overall                :    3.0 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
upper bound estimates
* data accesses          :    9.6 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
   - L1d hits            :    0.9 >>>>>>>>>>>>>>>>>
   - L2d hits            :    1.8 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
   - L2d misses          :    6.9 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
* instruction accesses   :    0.1 >
   - L1i hits            :    0.0 >
   - L2i hits            :    0.0 >
   - L2i misses          :    0.1 >
* data TLB               :    4.6 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
* instruction TLB        :    0.0 >
* branch instructions    :    0.1 >>
   - correctly predicted :    0.1 >>
   - mispredicted        :    0.0 >
* floating-point instr   :    5.1 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
   - fast FP instr       :    5.1 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
   - slow FP instr       :    0.0 >
```

# Metrics used by PerfExpert

## Architecture Characteristics

- Memory access latency: L1, L2, L3 and main memory (based on micro-benchmarks)
- Memory topology and size (based on hwlock)
- Branch latency and missed branch latency (based on micro-benchmarks)
- Float-point operation latency (based on micro-benchmarks)
- Micro-architecture (in progress)

## Source Code

- Language (C, C++, Fortran)
- File name and line number
- Type (loop or function)
- Function name and "deepness"
- Representativeness (percentage of execution time)

# Metrics used by PerfExpert

## Execution Performance

- Raw data (PAPI)
- LCPI: local cycles per instruction (PerfExpert Analyzer)

## Data Access Performance (from MACPO)

- Access strides and the frequency of occurrence (*)
- Presence or absence of cache thrashing and the frequency (*)
- Estimated cost (cycles) per access (*)
- NUMA misses (*)
- Reuse factors for data caches (*)
- Stream count

(*) *per variable*

# Extending PerfExpert

## Adding Performance Metrics

- Dynamically loaded into the support database
- We treat everything (most of them, actually) as metrics

## Some Example Metrics

```
code.section_info=Loop in function compute() at mm.c:8
code.filename=mm.c
code.line_number=8
code.type=loop
code.function_name=compute
code.representativeness=99.8
perfexpert.ratio.data_accesses=0.25
perfexpert.instruction_accesses.L2i_hits=0.002
perfexpert.branch_instructions.mispredicted=0.0
perfexpert.floating-point_instr.fast_FP_instr=5.073
perfexpert.data_accesses.L2d_hits=1.846
...
```

# Extending PerfExpert

## Recommendation Selection Functions

- Is is just a SQL query
- You can use as many functions as you want
- We already have some strategies on how to rank recommendations
- A recommendation may lead to several pattern recognizers

## A Simple Recommendation Selection Function Example

```
SELECT recommendation FROM t_rec WHERE
metric.A > 'this' AND metric.B <= 'that'
ORDER BY score DESC;
```

# Extending PerfExpert

## Pattern Recognizers

- Any program which returns 0 or 1
- Language sensitive
- A pattern recognizer may lead to several code transformers

## A Simple Grammar (Byson/Flex)

```
nested_iteration_statement
:  WHILE '(' exp ')' WHILE '(' exp ')' stmnt
| WHILE '(' exp ')' '' WHILE '(' exp ')' stmnt ''
| DO DO stmnt WHILE '(' exp ')' ';' stmnt WHILE '(' exp ')' ';'
| DO '' DO stmnt WHILE '(' exp ')' ';' '' WHILE '(' exp ')' ';'
| FOR '(' exp_stmnt exp_stmnt ')' FOR '(' exp_stmnt exp_stmnt ')' stmnt
| FOR '(' exp_stmnt exp_stmnt ')' '' FOR '(' exp_stmnt exp_stmnt ')' stmnt ''
| FOR '(' exp_stmnt exp_stmnt exp ')' FOR '(' exp_stmnt exp_stmnt exp ')' stmnt
| FOR '(' exp_stmnt exp_stmnt exp ')' '' FOR '(' exp_stmnt exp_stmnt exp ')' stmnt '' ;
```

# Extending PerfExpert

## Code Transformers

- Any program which returns 0 or 1
- May be language sensitive

## A Simple TPIPS script

```
create c_loop2 ../source/mm.c
activate INTERPROCEDURAL_SUMMARY_PRECONDITION
activate TRANSFORMERS_INTER_FULL
activate PRECONDITIONS_INTER_FULL
setproperty SEMANTICS_FIX_POINT_OPERATOR ‘‘derivative’’
module compute
apply LOOP_INTERCHANGE
loop_8
apply UNSPLIT[%PROGRAM]
close
quit
```

# Agenda



# Conclusions

## Why is this performance optimization "architecture" strong?

- Each piece of the tool chain can be updated/upgraded individually
- **It is extendable**: metrics, performance measurement and analysis phases, 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!

- This is the first end-to-end open-source performance optimization tool (as far as we know)
- It will become more and more powerful as new recommendations, transformations and features are added
- There is no "big code" (to increase in complexity until it become unusable or too hard to maintain)

# Next Steps

## Major Goals

- Improve analysis based on the data access (in progress)
- Increase the number of recommendations and possible code transformations (continuously)
- New algorithms for recommendations selection (in progress)
- Add support to MIC architecture (in progress)
- Add support to MPI-related recommendations (medium term)
- Add support to MPI-related code transformations (long term)

## Minor Goals

- Support "Makefile"-based source code/compilation tree (done!)
- Make the required packages installation process easier (done!)
- Add a test suite based on established benchmark codes (in progress)
- Easy-to-use interface to manipulate the support database (medium term)

# Thank You

fialho@utexas.edu

http://www.tacc.utexas.edu/perfexpert