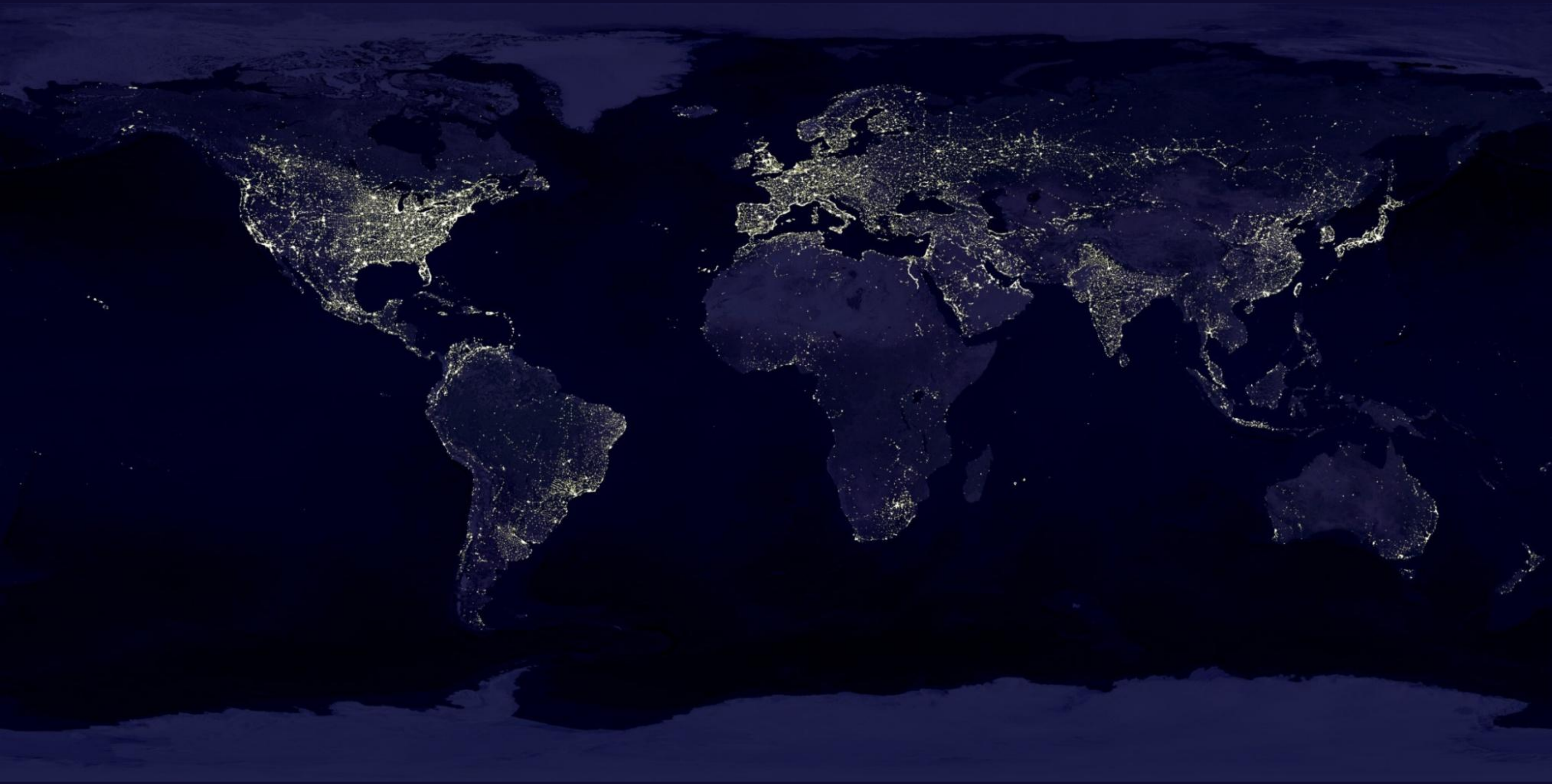
Collective Mind infrastructure and repository to crowdsource auto-tuning



Grigori Fursin

INRIA, France

HPSC 2013, Taiwan

March 2013

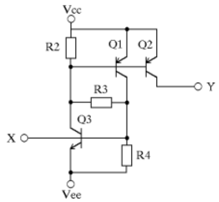
Background

- Collective Mind approach combined with social networking, expert knowledge and predictive modeling
- Collective Mind framework basics
  - Plugin-based type-free and schema-free infrastructure
  - Unified web interfaces (similar to WEB 2.0 concept)
  - Portable file (json) based repository
  - Auto-tuning and predictive modeling scenarios
- Demo
- Conclusions and future works

Motivation: back to basics

End users

Task



Service/application providers

(supercomputing, cloud computing, mobile systems)

Solution

User requirements:

most common:

minimize all costs (time, power consumption, price, size, faults, etc)

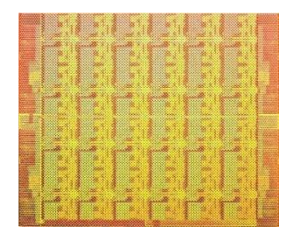
guarantee real-time constraints (bandwidth, QOS, etc)

Decision
(depends on user requirements)

Available choices (solutions)

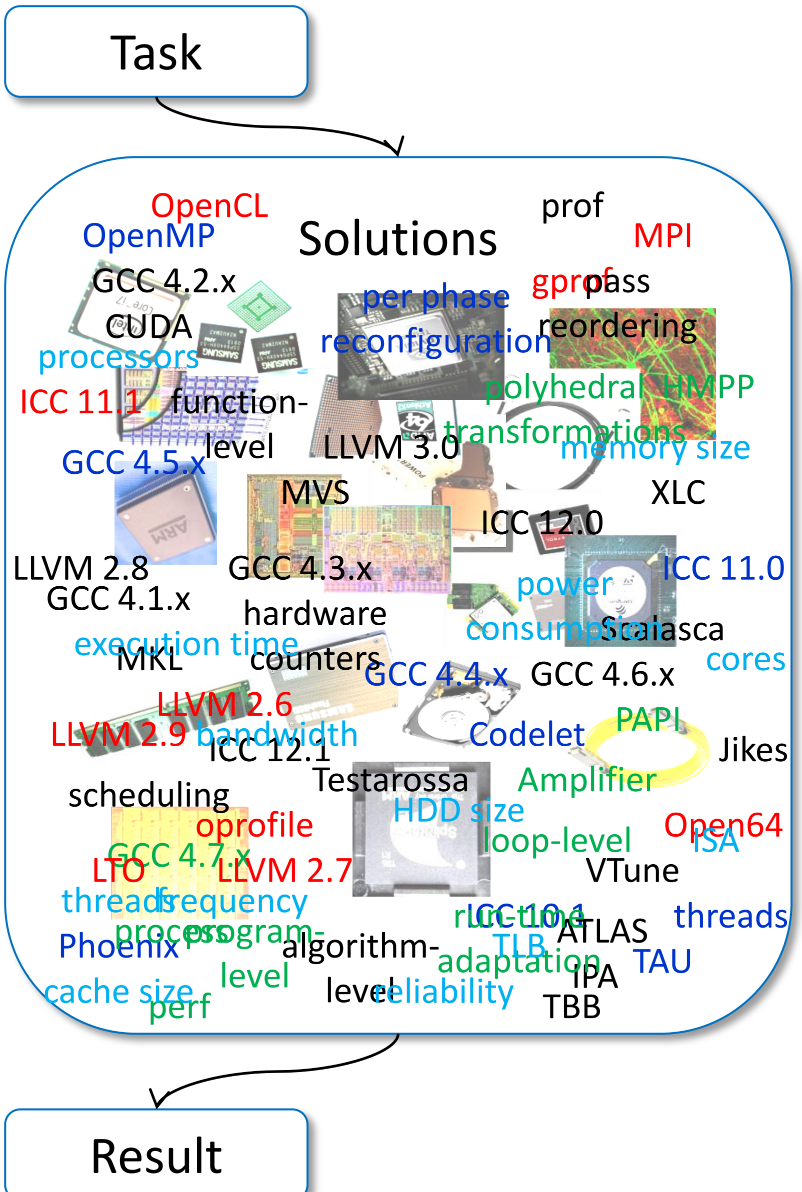
Should provide choices and help with decisions

Hardware and software designers



Result

Challenges



Clean up this mess!

Simplify analysis, tuning and modelling of computer systems for non-computer engineers

Bring together researchers from interdisciplinary communities

Understanding computer systems' behavior: a physicist's approach

Task

Treat computer system as a black box

Result

Understanding computer systems' behavior: a physicist's approach

Task

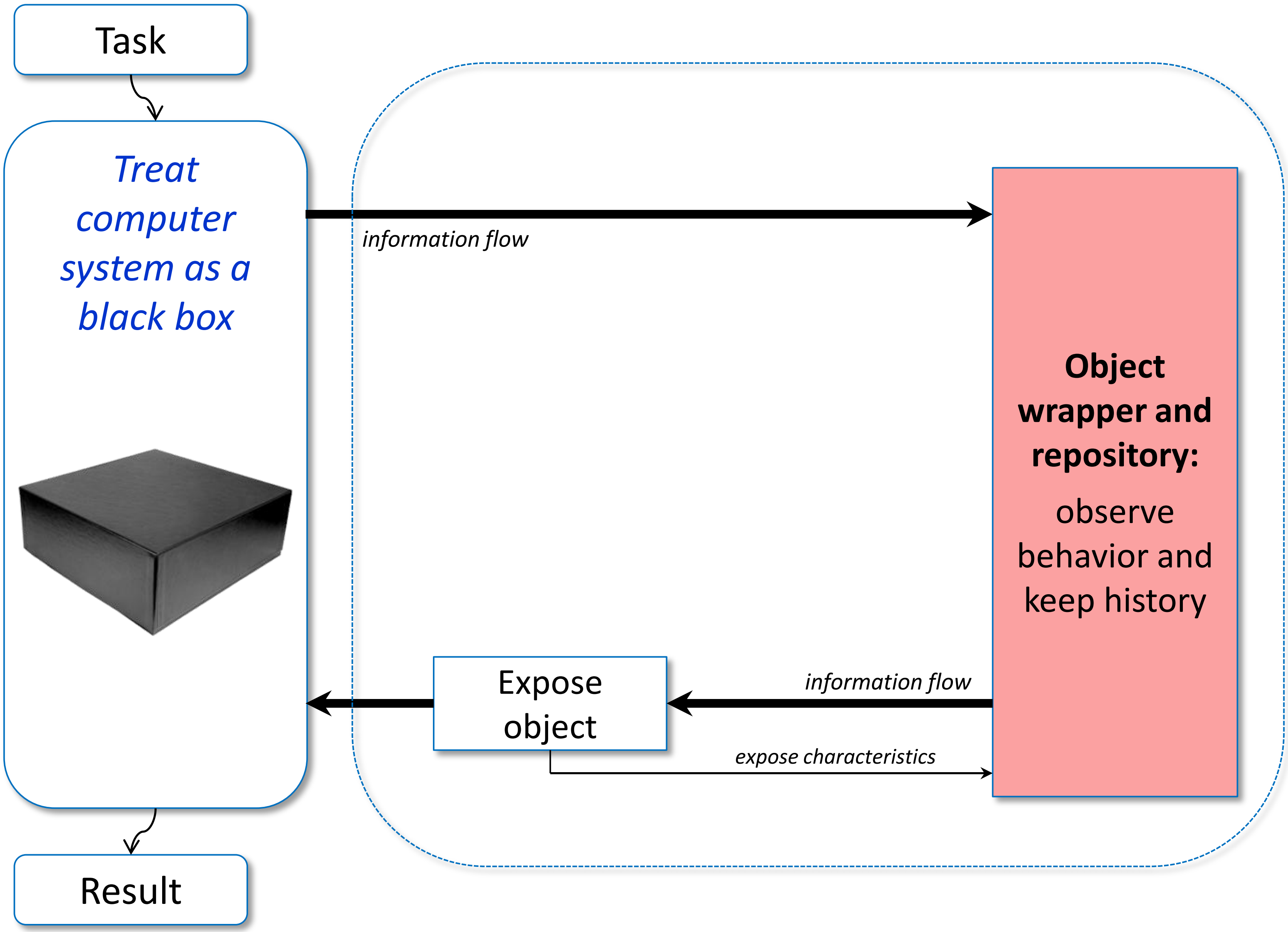
Treat computer system as a black box

information flow

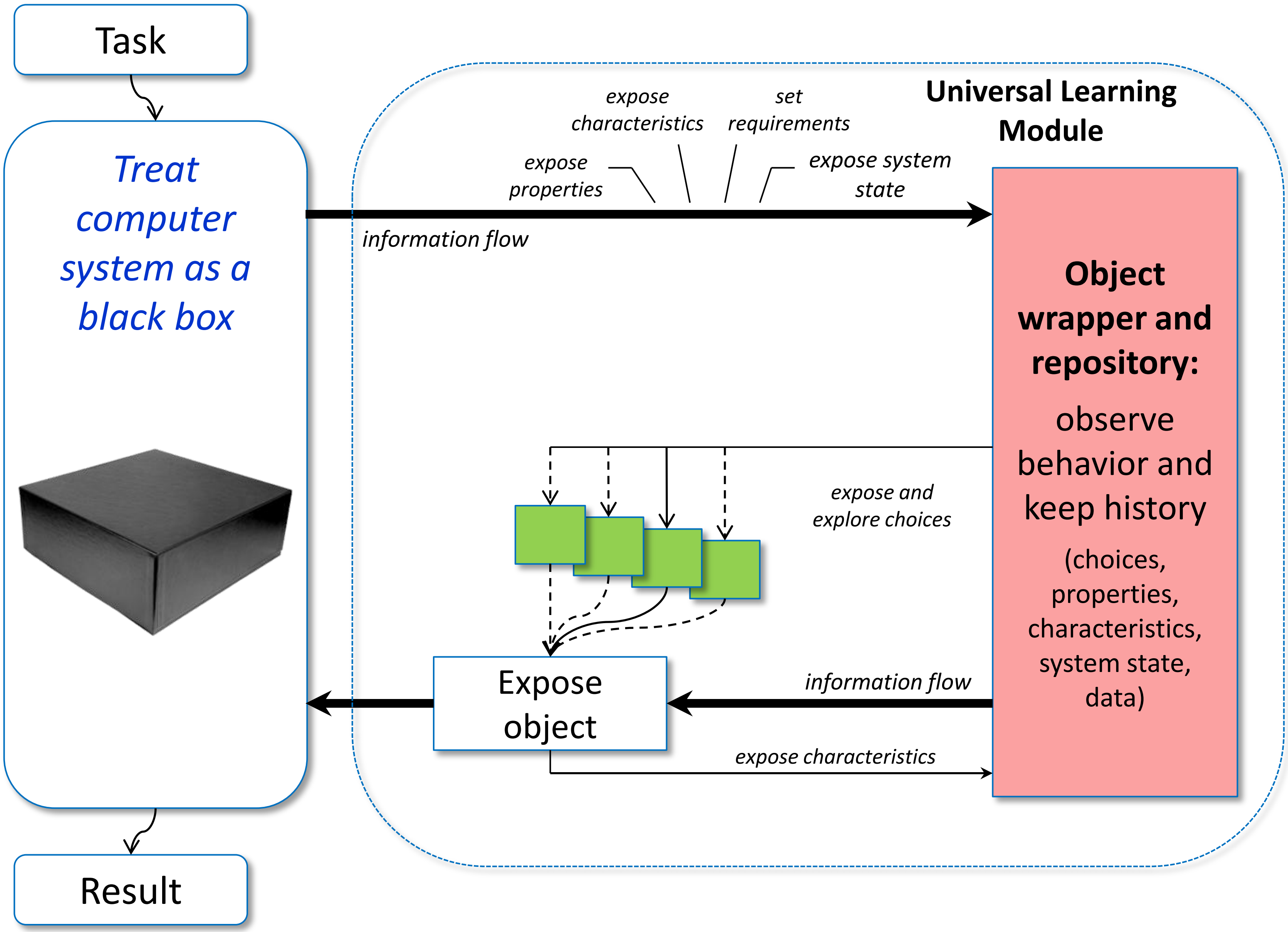
Expose object

Result

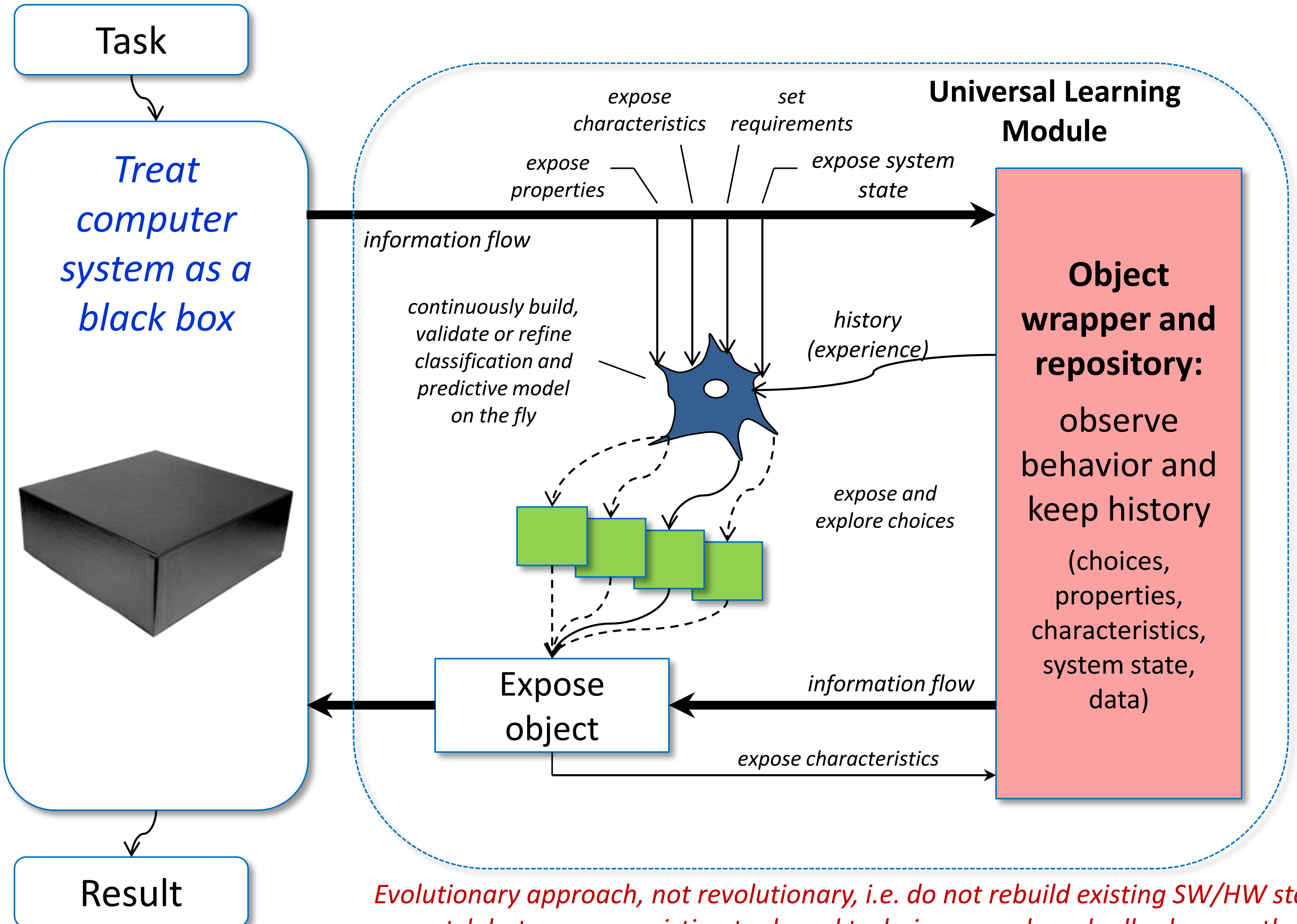
Observe system



Gradually expose properties, characteristics, choices



Classify, build models, predict behavior

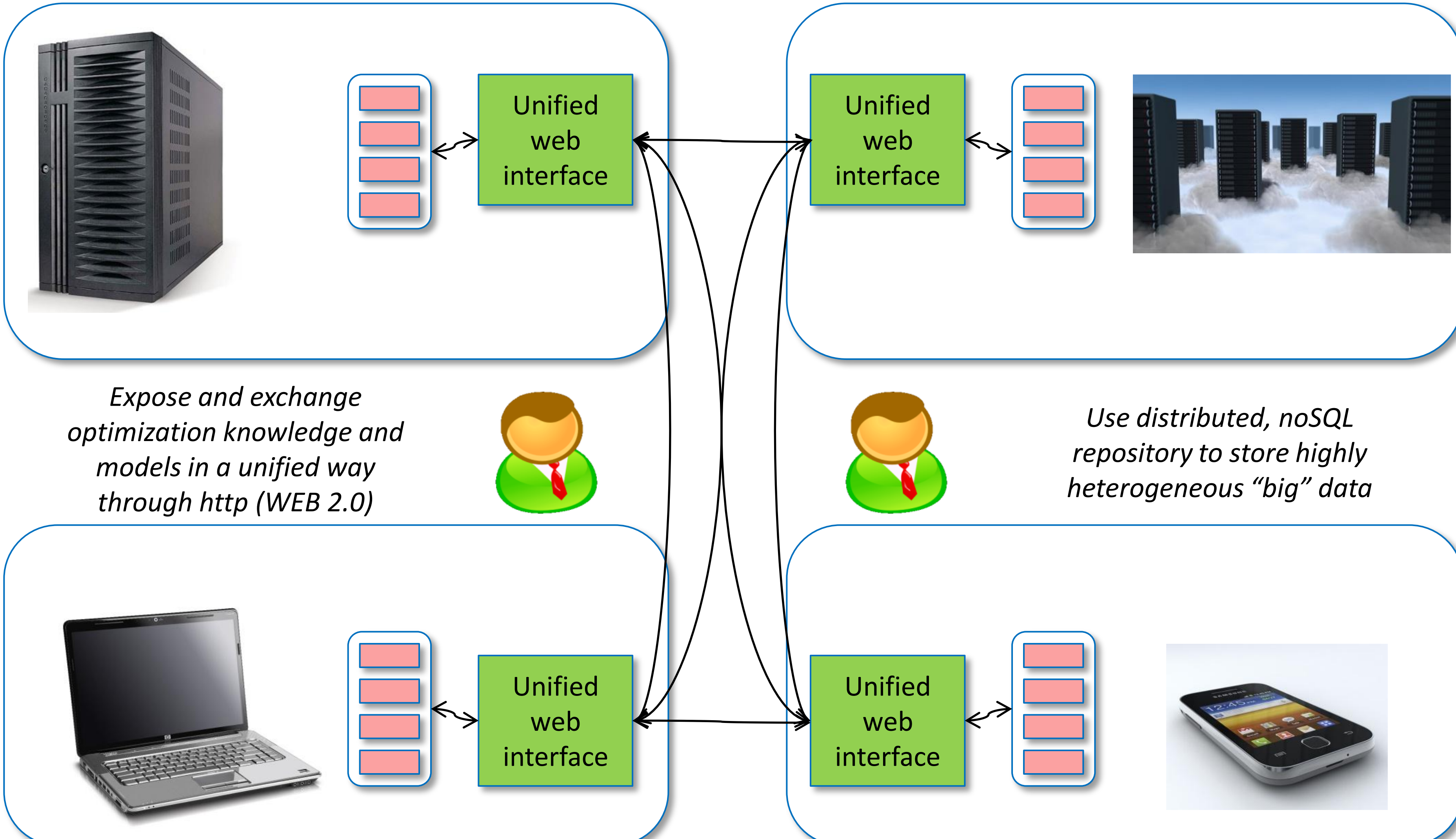


Evolutionary approach, not revolutionary, i.e. do not rebuild existing SW/HW stack from scratch but wrap up existing tools and techniques, and gradually clean up the mess!

Transparently crowdsource learning of a behavior of any existing mobile, cluster, cloud computer system

Extrapolate collective knowledge to build faster and more power efficient computer systems

Build self-tuning machines using agent-based models

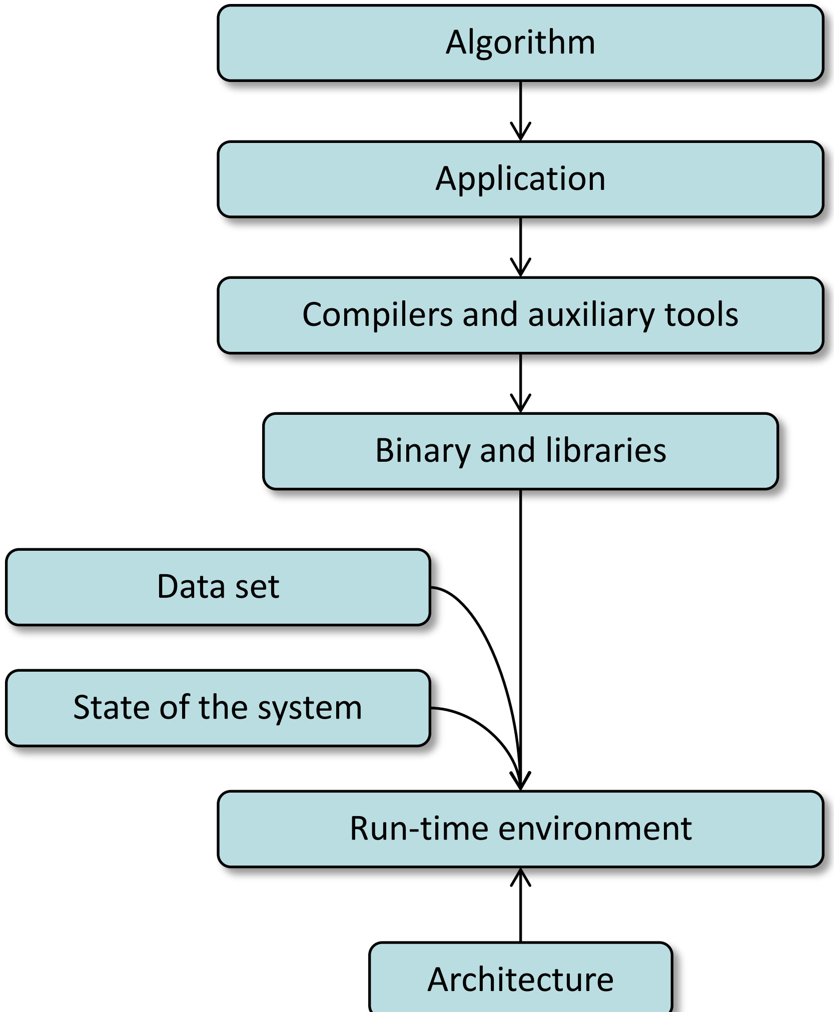


Gradual decomposition, parameterization, observation and exploration of a system

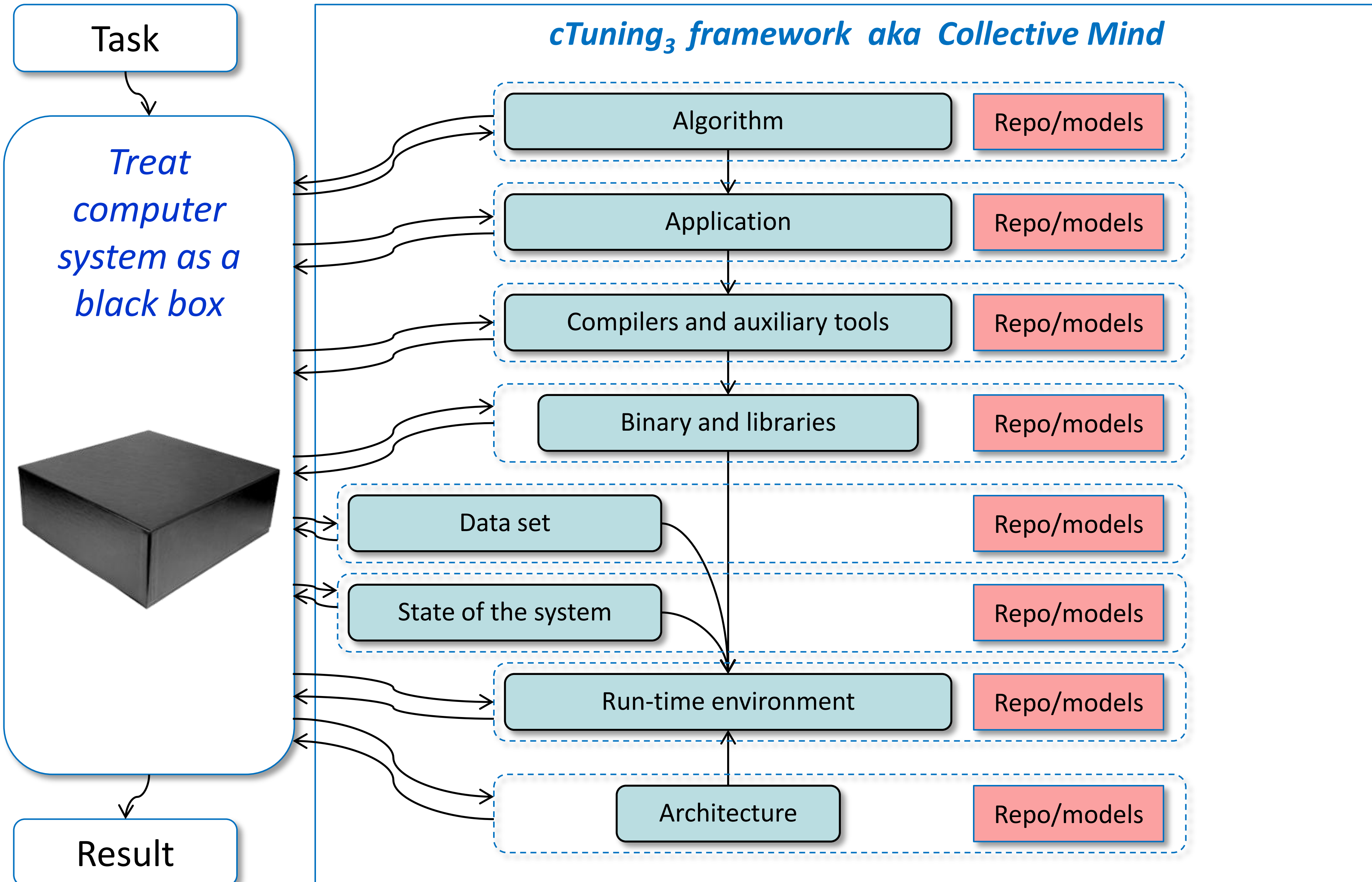
Task

Treat computer system as a black box

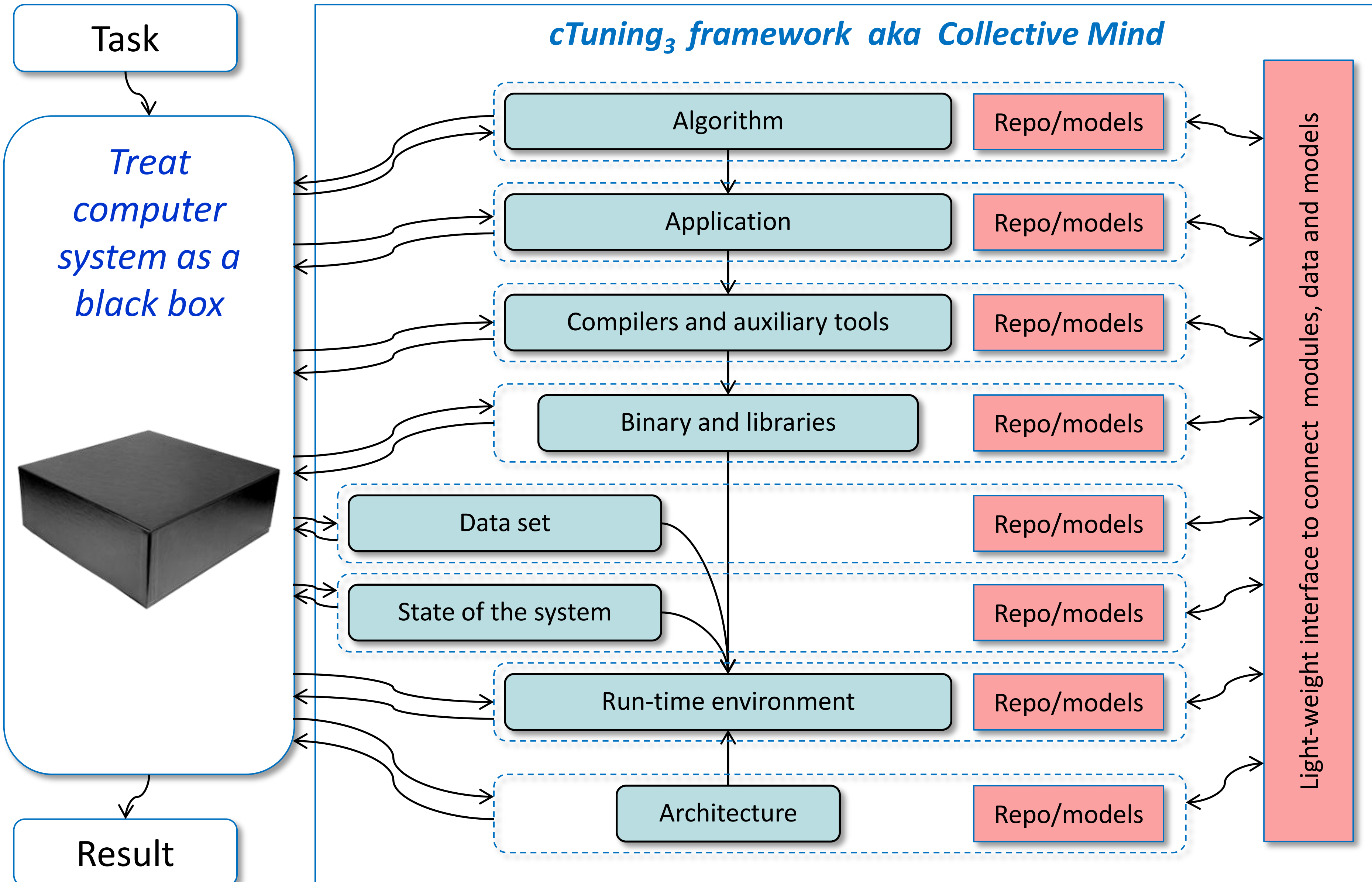
Result



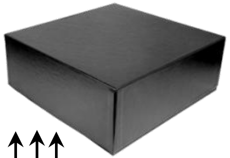
Gradual decomposition, parameterization, observation and exploration of a system



Gradual top-down decomposition, parameterization, observation and exploration of a system



Example of characterizing/explaining behavior of computer systems



		Gradually expose some characteristics	Gradually expose some properties/choices
Compile Program		time …	compiler flags; pragmas …
Run code	Run-time environment	time; CPI, power consumption …	pinning/scheduling …
	System	cost;	architecture; frequency; cache size…
	Data set	size; values; description …	precision …
Analyze profile		time; size …	instrumentation; profiling …

Combine expert knowledge with automatic detection!

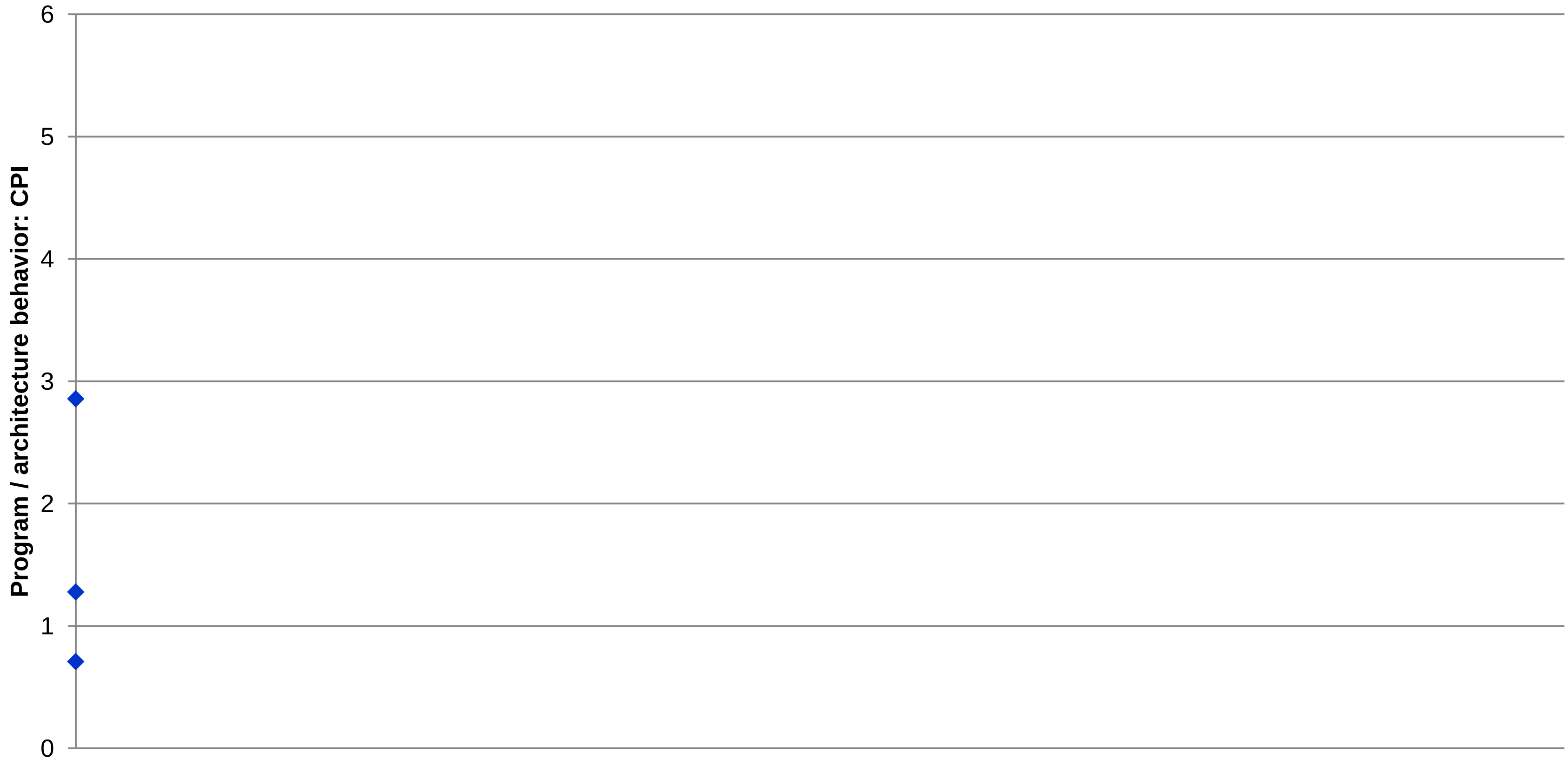
Start from coarse-grain and gradually move to fine-grain level!

Start coarse-grain decomposition of a system (detect coarse-grain effects first). Add universal learning modules.

Example of characterizing/explaining behavior of computer systems

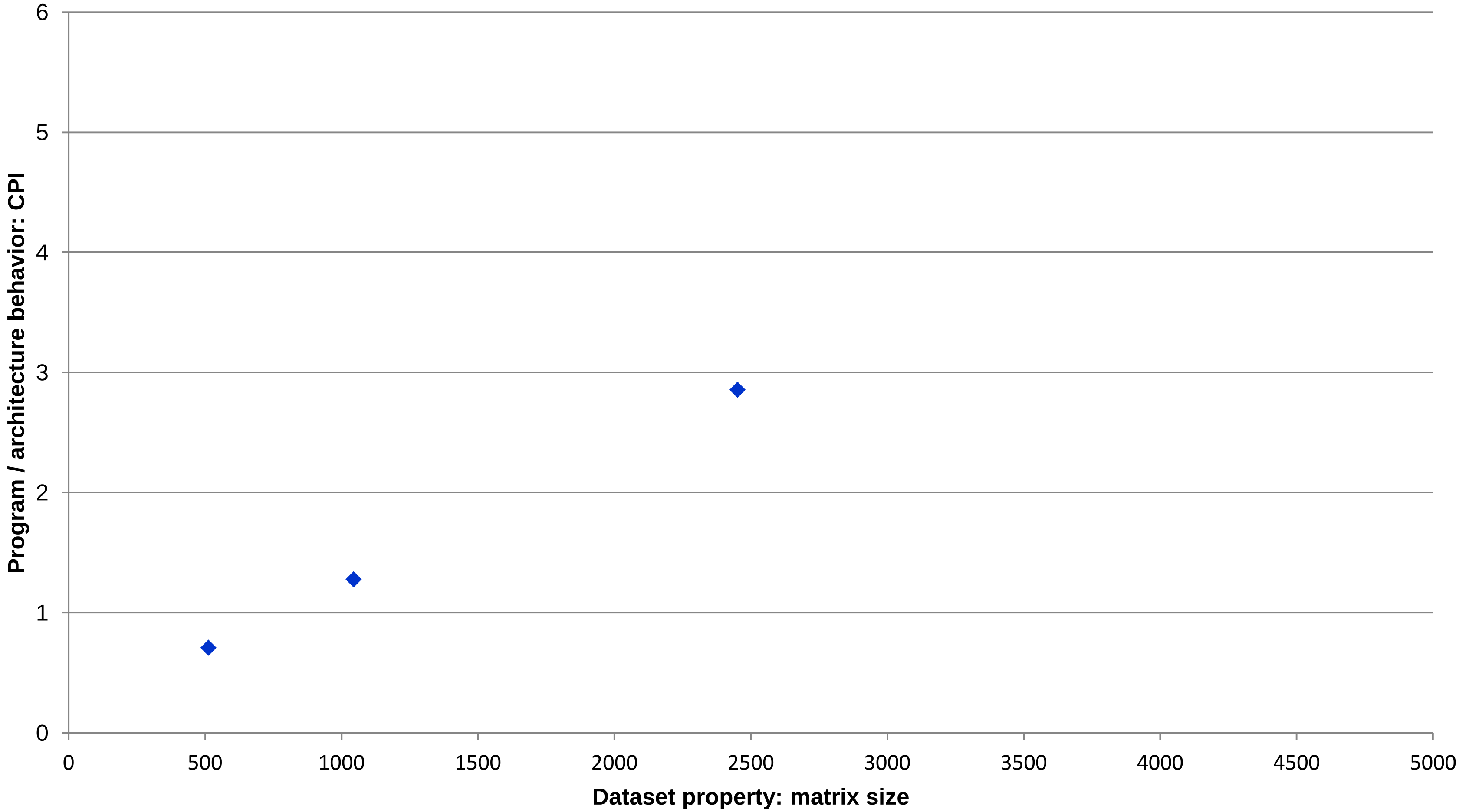
How we can explain the following observations for some piece of code (“codelet object”)?

(LU-decomposition codelet, Intel Nehalem)



Example of characterizing/explaining behavior of computer systems

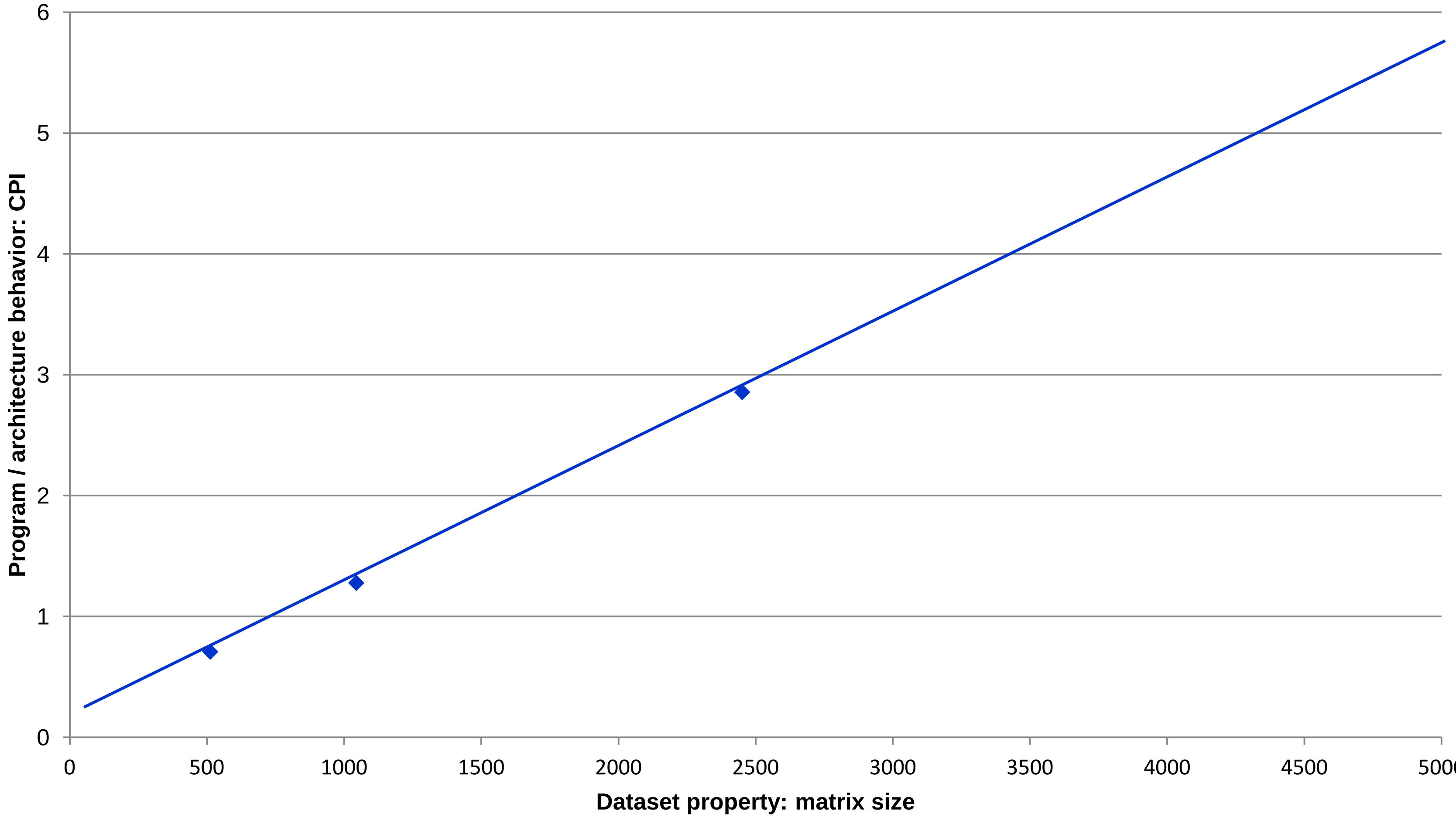
Add 1 property: matrix size



Example of characterizing/explaining behavior of computer systems

Try to build a model to correlate objectives (CPI) and features (matrix size).

Start from simple models: linear regression (detect coarse grain effects)

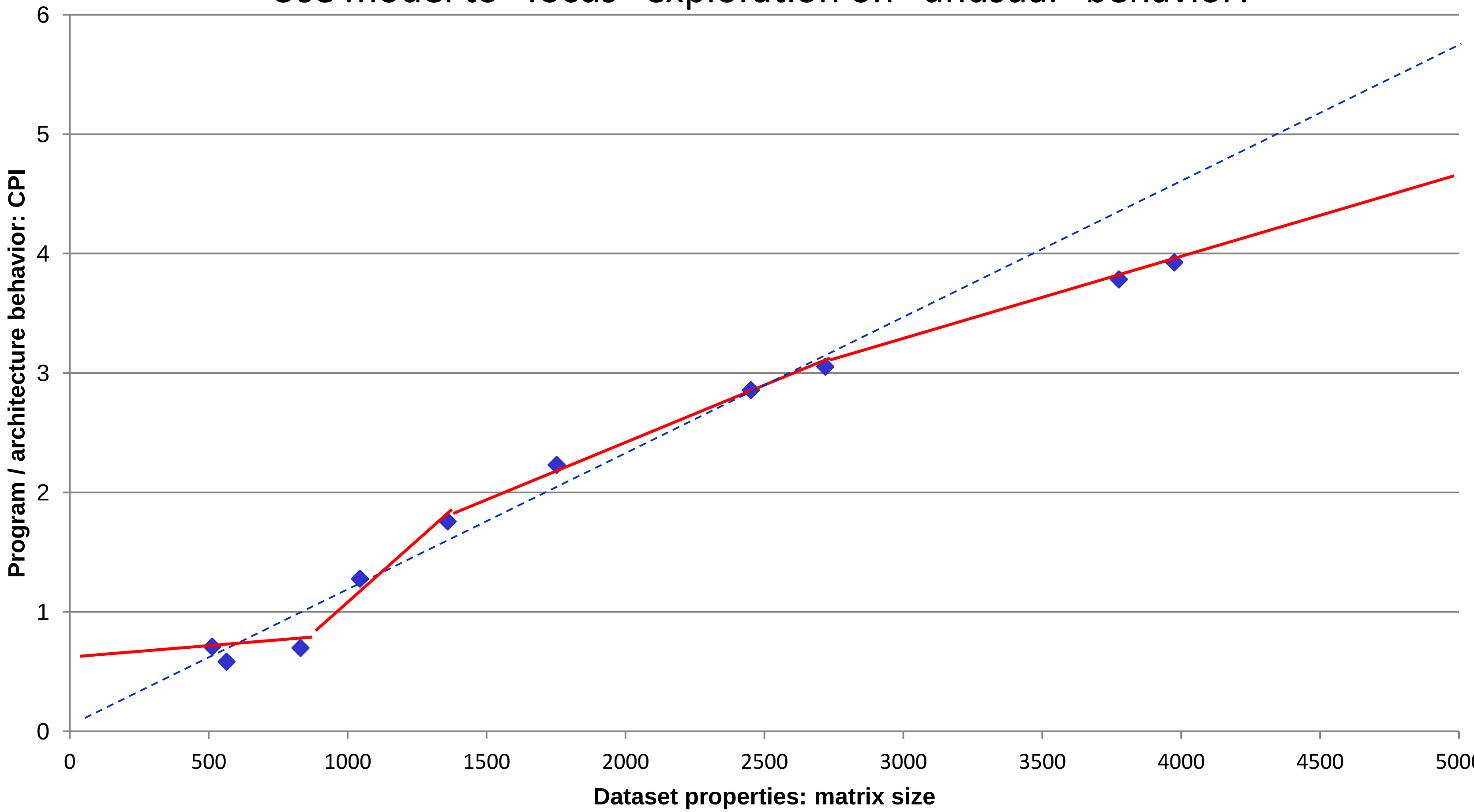


Example of characterizing/explaining behavior of computer systems

If more observations, validate model and detect discrepancies!

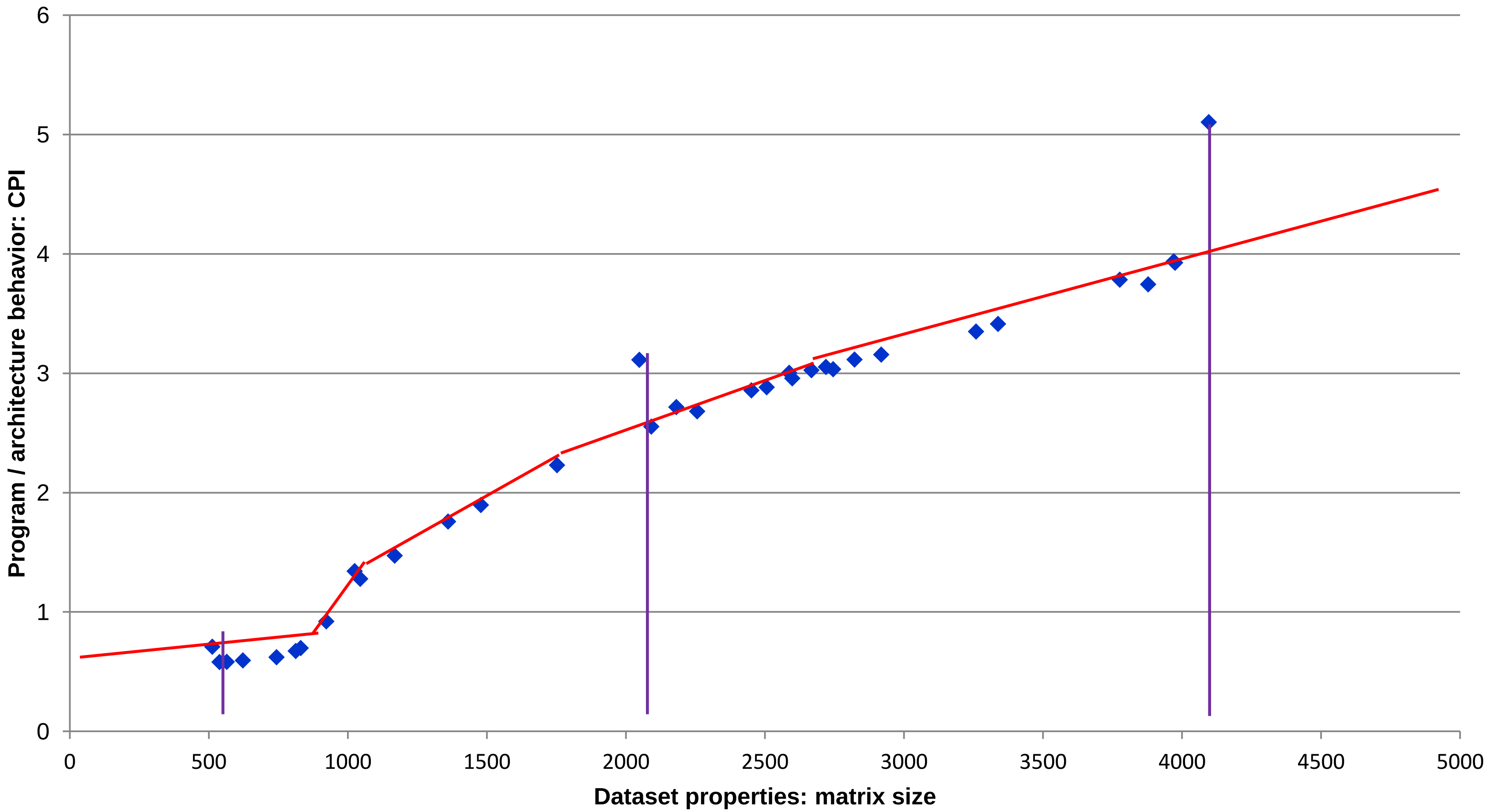
Continuously retrain models to fit new data!

Use model to “focus” exploration on “unusual” behavior!



Example of characterizing/explaining behavior of computer systems

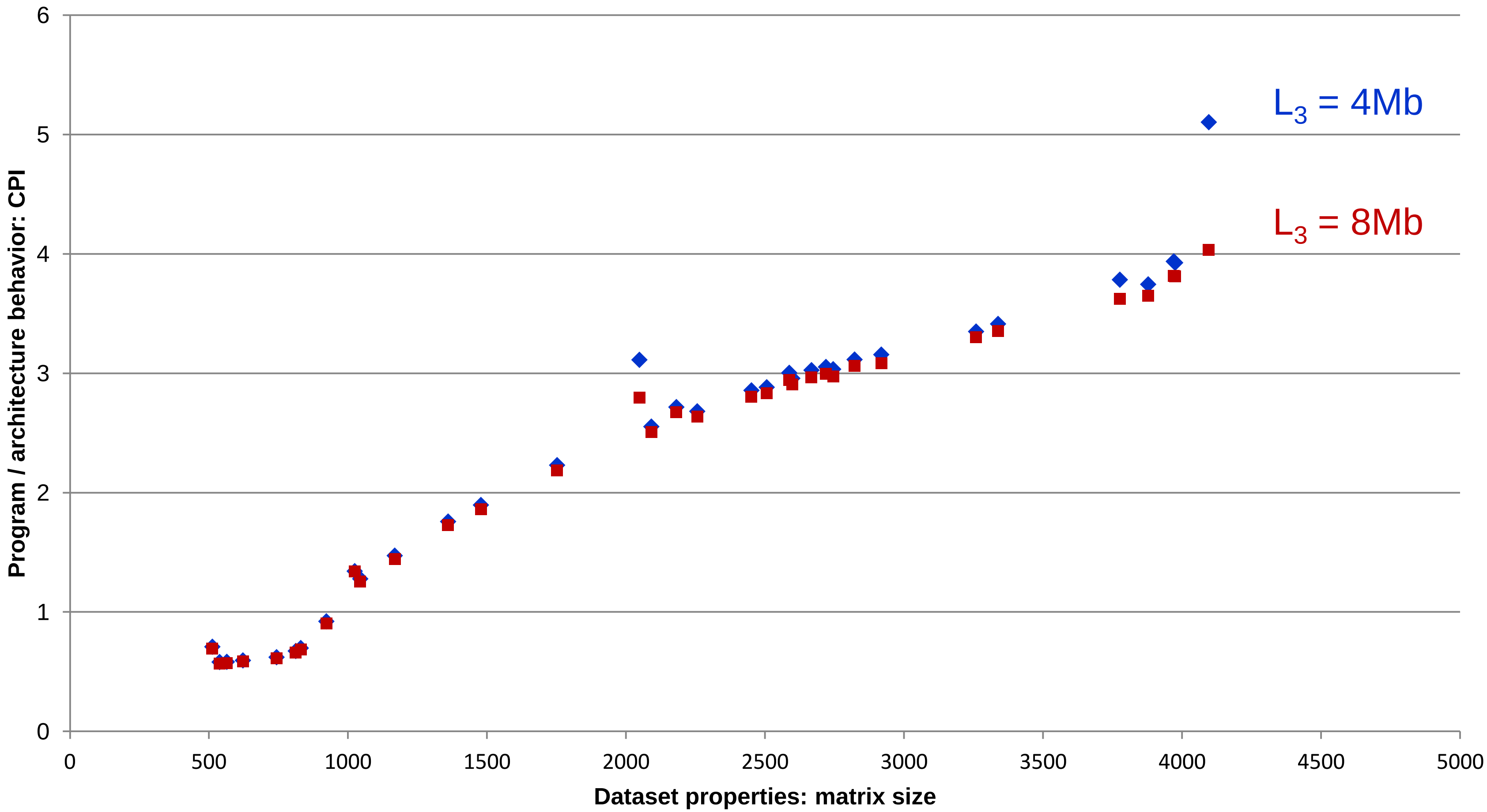
Gradually increase model complexity if needed (hierarchical modeling).
For example, detect fine-grain effects (singularities) and characterize them.



Example of characterizing/explaining behavior of computer systems

Start adding more properties (one more architecture with twice bigger cache)!

Use automatic approach to correlate all objectives and features.



Example of characterizing/explaining behavior of computer systems

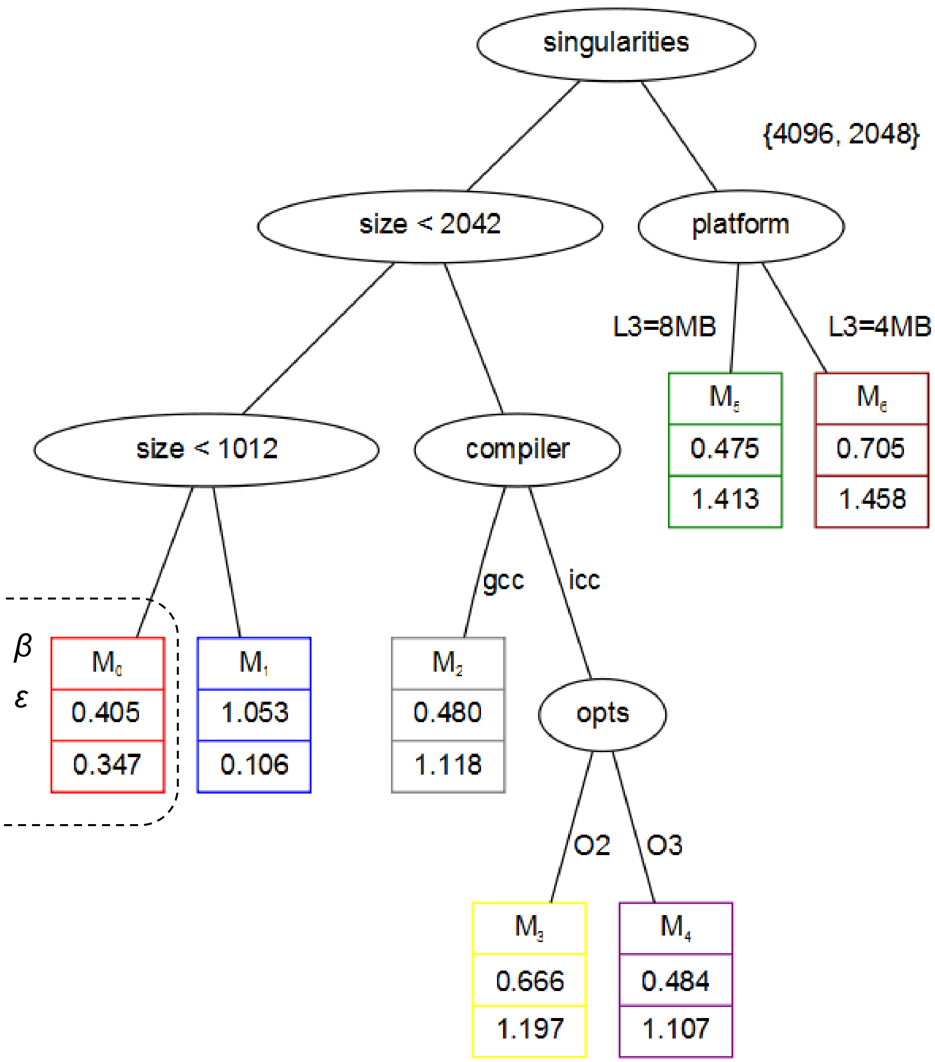
Continuously build and refine classification (decision trees for example) and predictive models on all collected data to improve predictions.

Continue exploring design and optimization spaces (evaluate different architectures, optimizations, compilers, etc.)

Focus exploration on unexplored areas, areas with high variability or with high mispredict rate of models

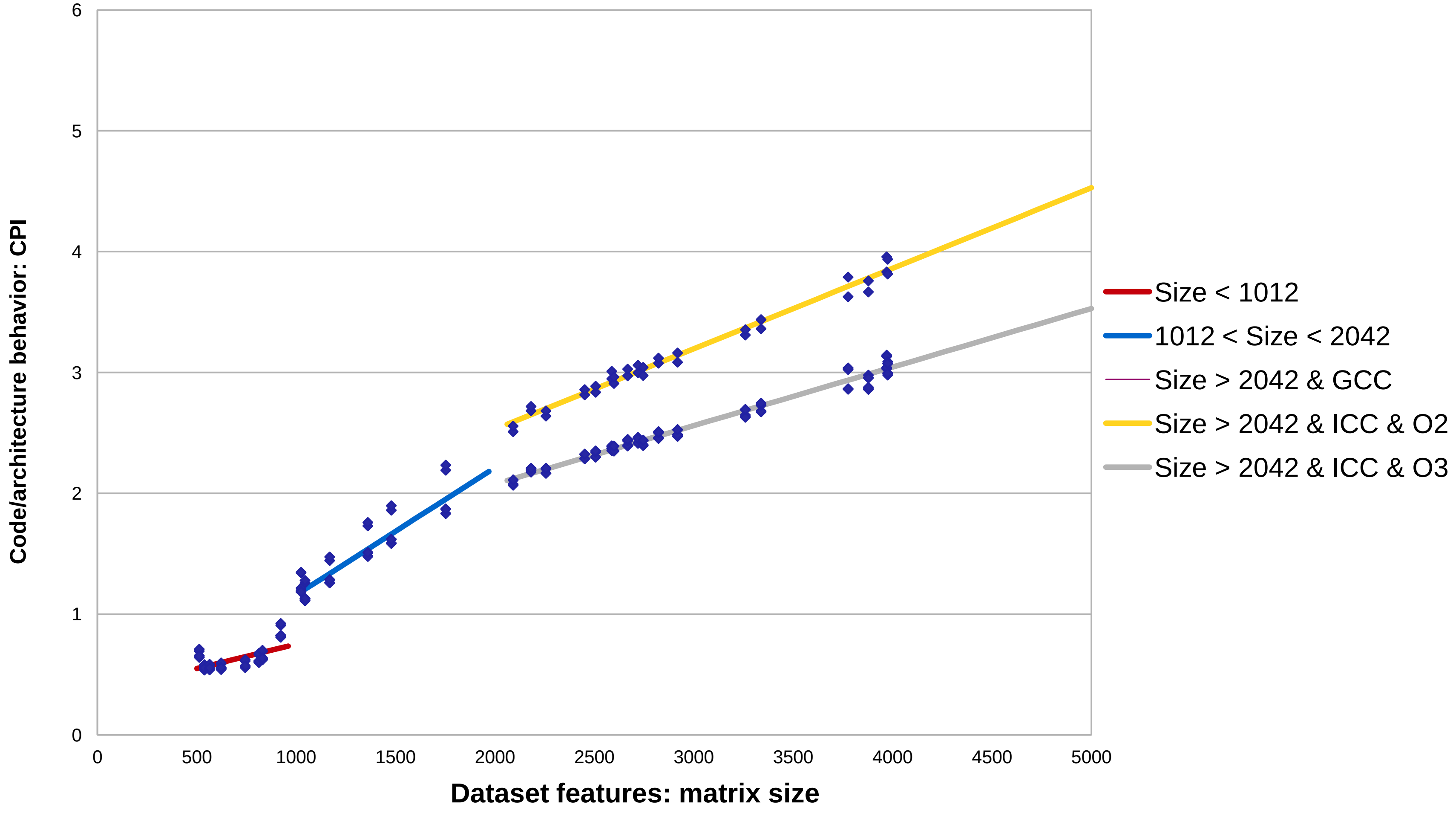
cM predictive model module

CPI = $\varepsilon + 1000 \times \beta \times$ data size



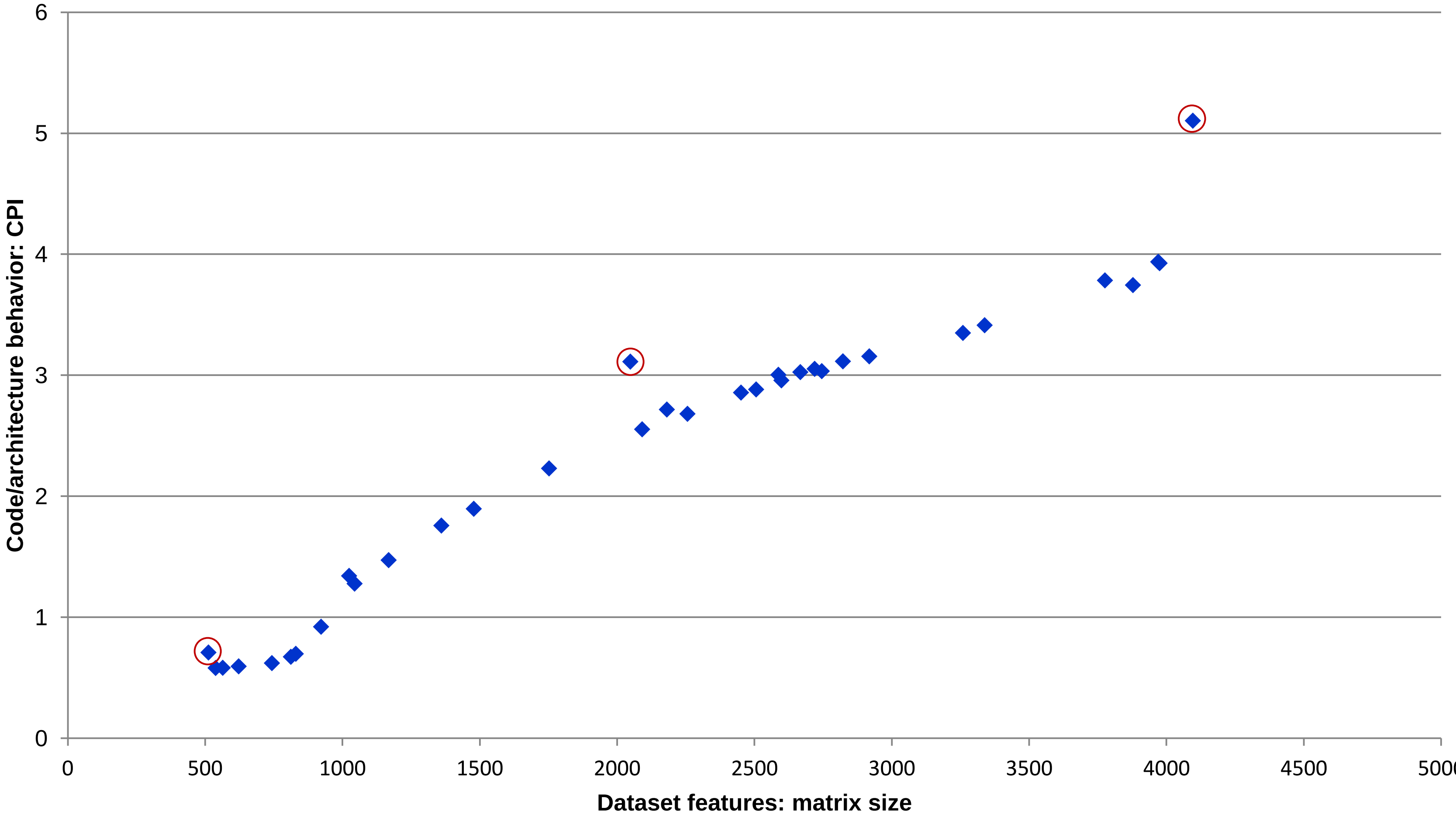
Model optimization and data compaction

Optimize decision tree (many different algorithms)
Balance precision vs cost of modeling = ROI (coarse-grain vs fine-grain effects)
Compact data on-line before sharing with other users!



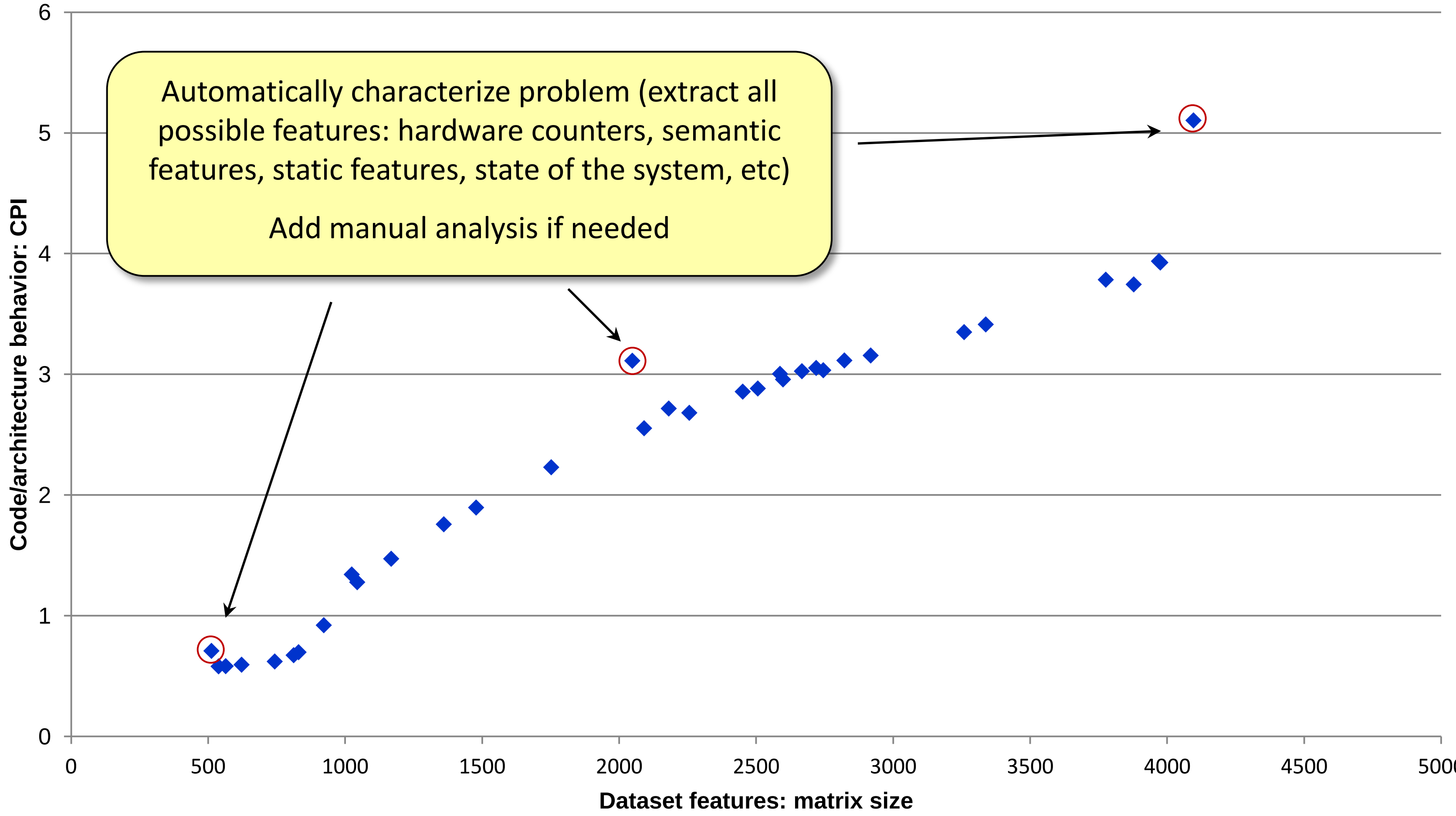
Extensible and collaborative advice system

Collaboratively and continuously add expert advices or automatic optimizations.



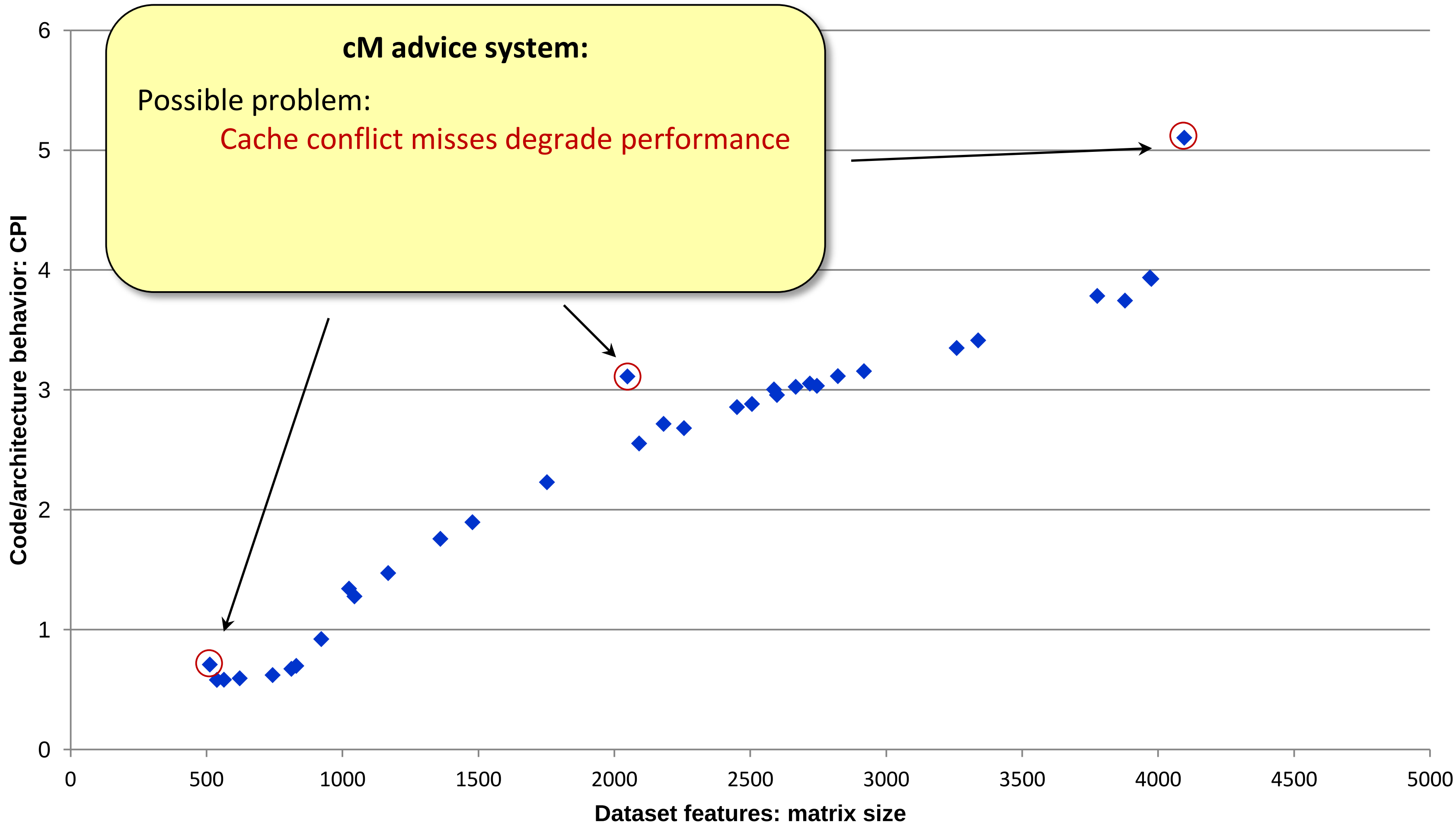
Extensible and collaborative advice system

Collaboratively and continuously add expert advices or automatic optimizations.



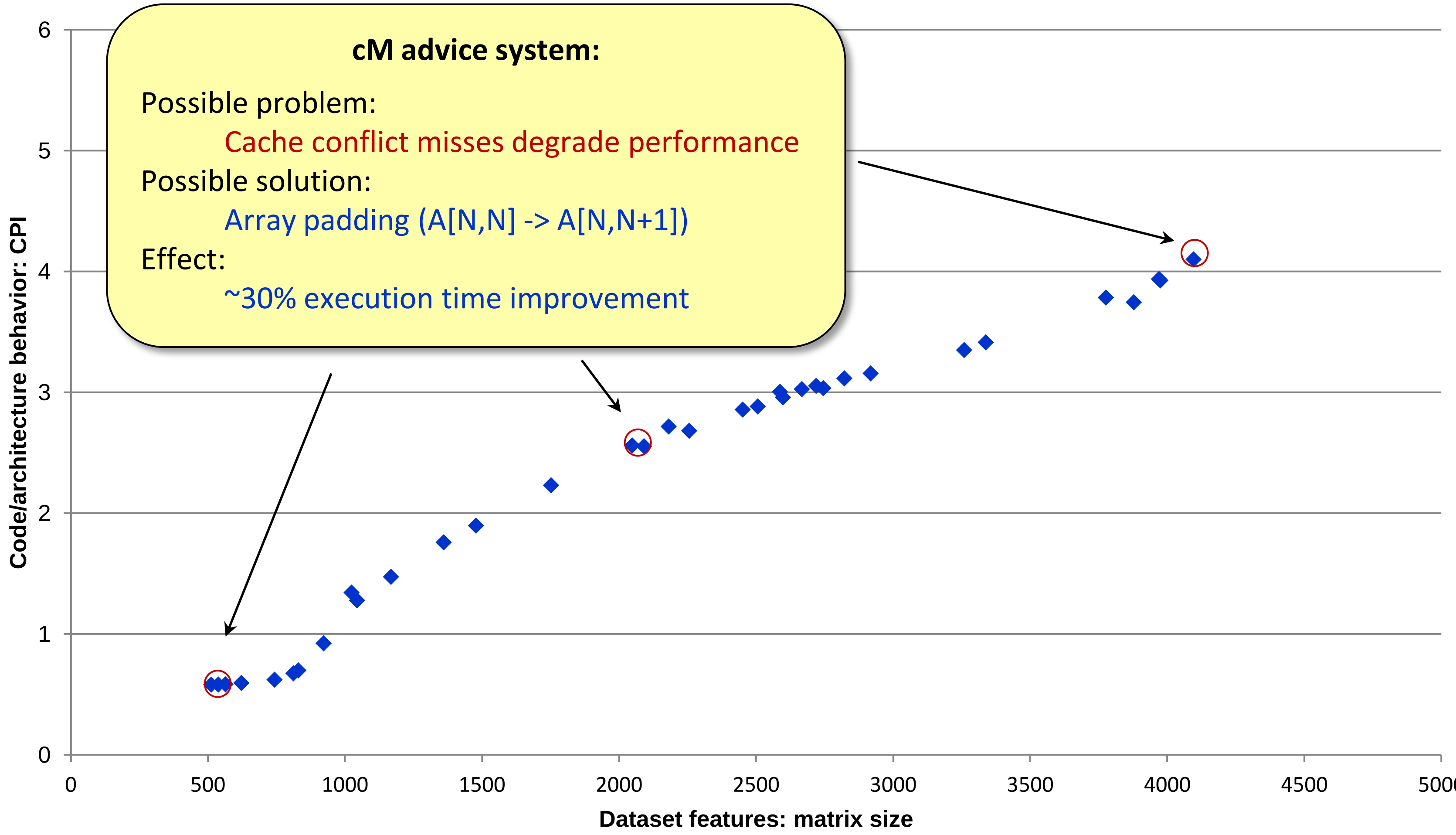
Extensible and collaborative advice system

Collaboratively and continuously add expert advices or automatic optimizations.



Extensible and collaborative expert system

Collaboratively and continuously add expert advices or automatic optimizations.

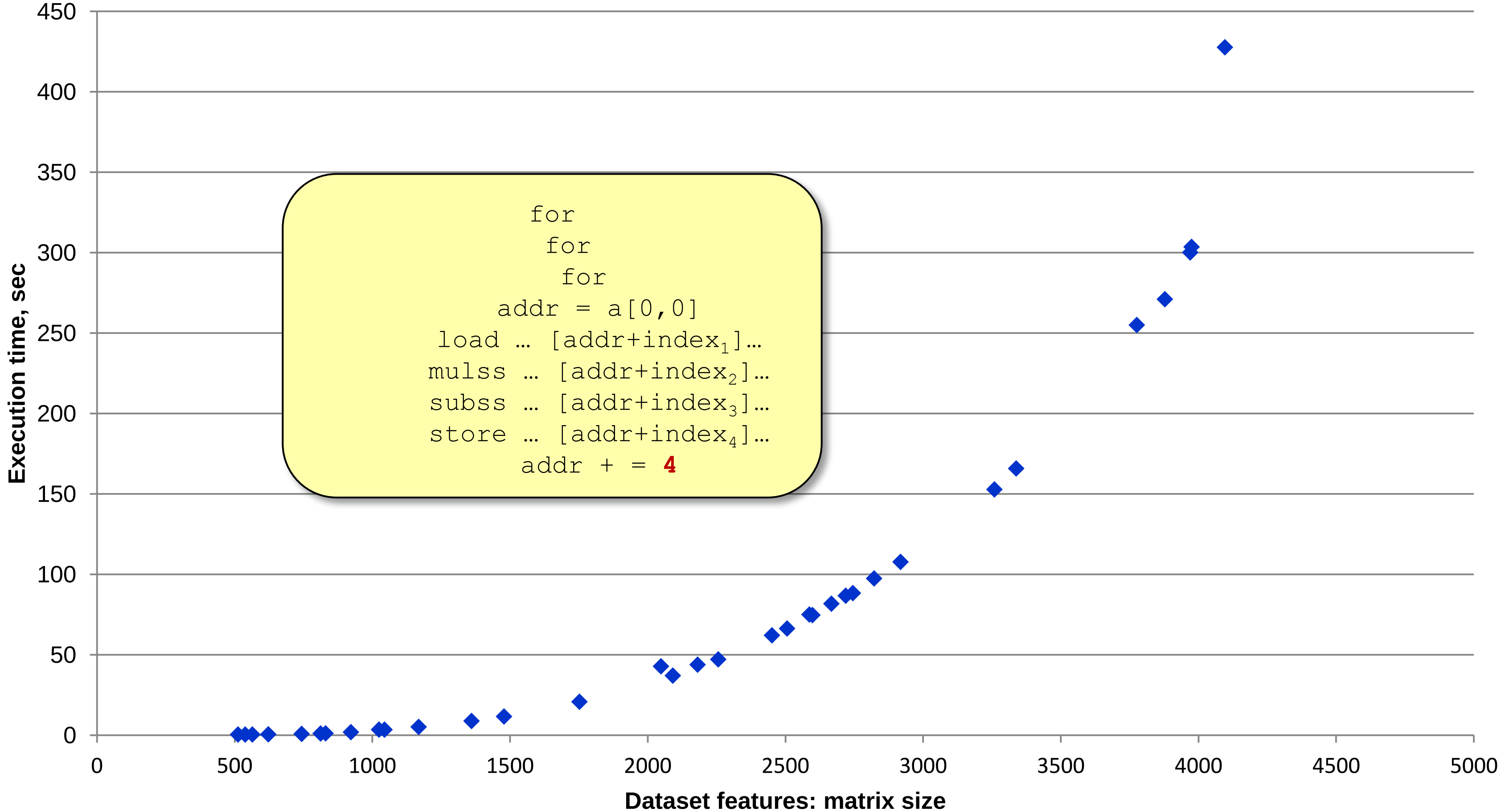


System reaction to code changes: physicist's view

Add dynamic memory characterization through semantically non-equivalent modifications.

For example, convert all array accesses to scalars to detect balance between CPU/memory accesses.

Intentionally change/break semantics to observe reaction in terms of performance/power etc!



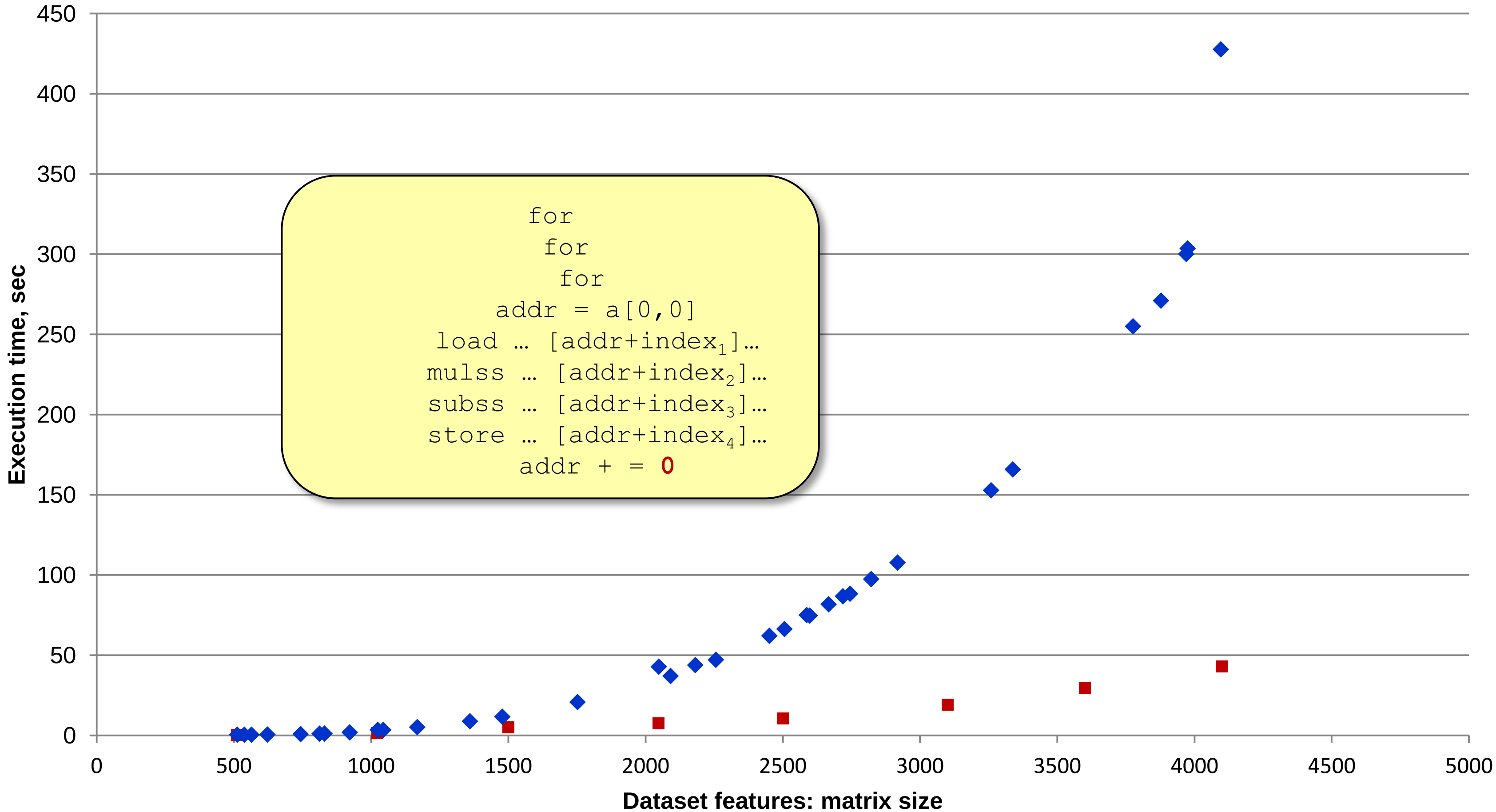
Grigori Fursin, Mike O'Boyle, Olivier Temam, and Gregory Watts. ***Fast and Accurate Method for Determining a Lower Bound on Execution Time.*** *Concurrency Practice and Experience, 16(2-3), pages 271-292, 2004*

System reaction to code changes: physicist's view

Add dynamic memory characterization through semantically non-equivalent modifications.

For example, convert all array accesses to scalars to detect balance between CPU/memory accesses.

Intentionally change/break semantics to observe reaction in terms of performance/power etc!

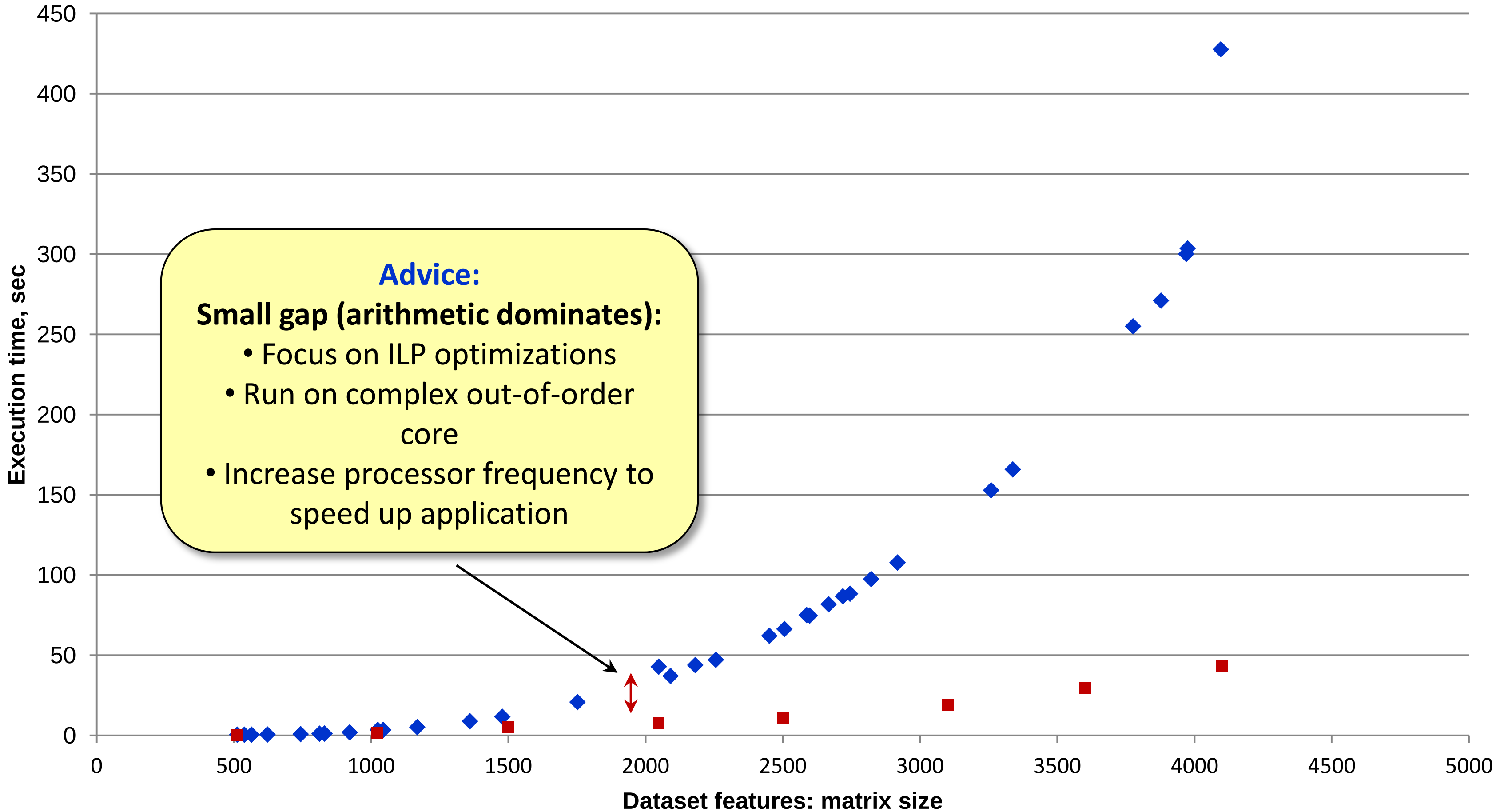


Grigori Fursin, Mike O'Boyle, Olivier Temam, and Gregory Watts. ***Fast and Accurate Method for Determining a Lower Bound on Execution Time.*** *Concurrency Practice and Experience, 16(2-3), pages 271-292, 2004*

System reaction to code changes: physicist's view

Expert or automatic advices based on additional information in the repository!

Focus optimizations to speed up search: which/where?

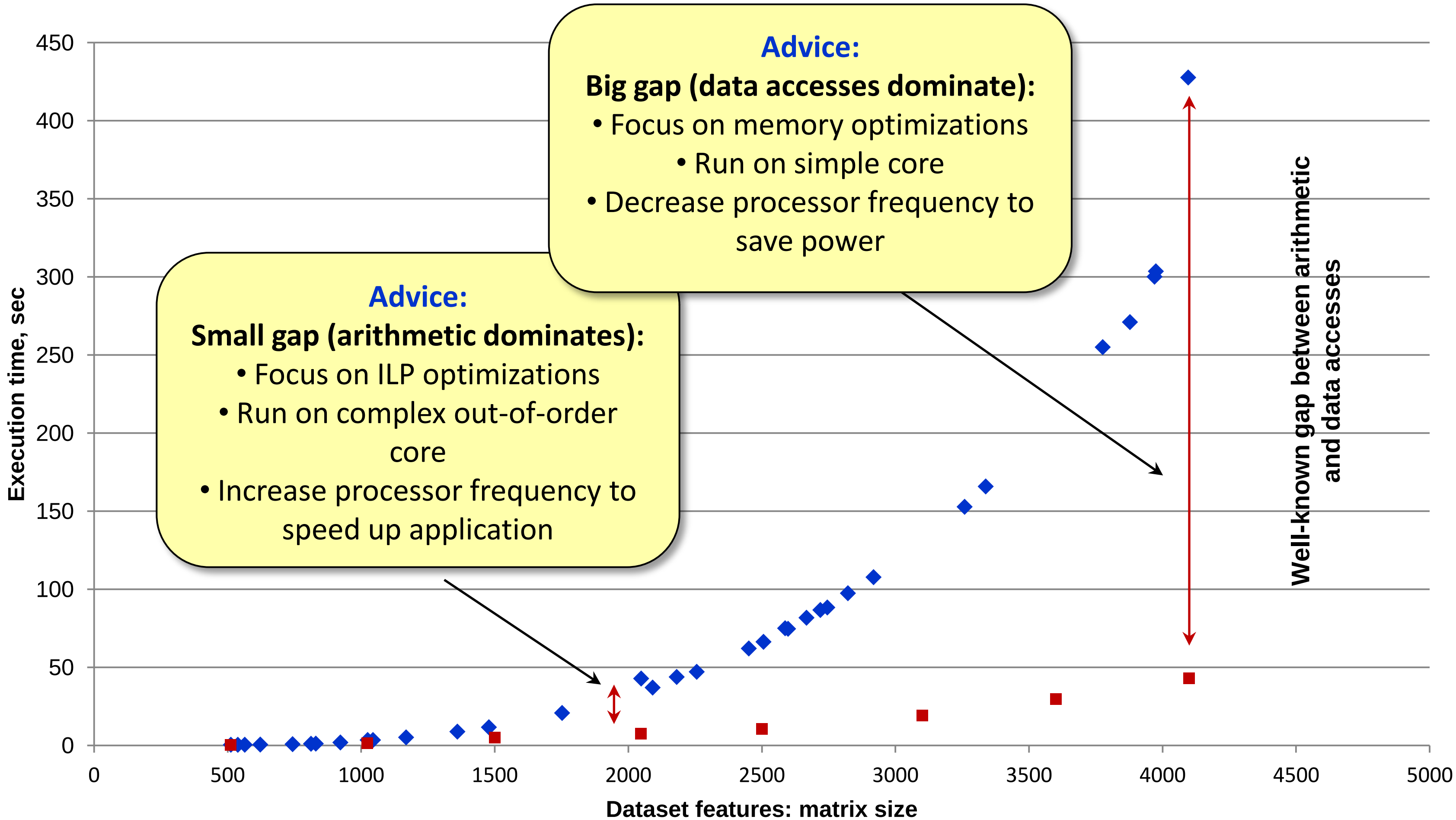


Grigori Fursin, Mike O'Boyle, Olivier Temam, and Gregory Watts. ***Fast and Accurate Method for Determining a Lower Bound on Execution Time.*** *Concurrency Practice and Experience, 16(2-3), pages 271-292, 2004*

System reaction to code changes: physicist's view

Expert or automatic advices based on additional information in the repository!

Focus optimizations to speed up search: which/where?

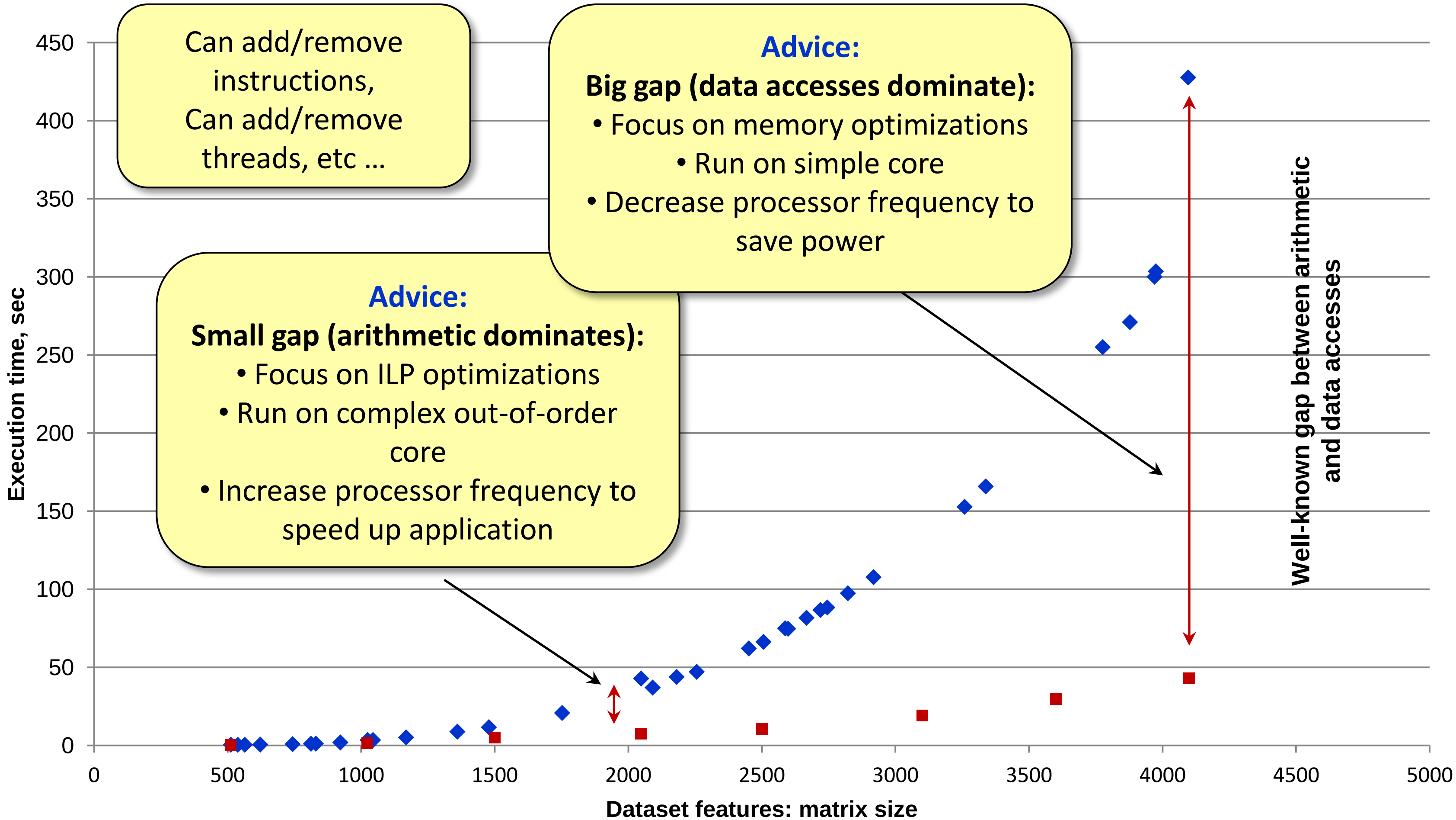


Grigori Fursin, Mike O'Boyle, Olivier Temam, and Gregory Watts. ***Fast and Accurate Method for Determining a Lower Bound on Execution Time.*** *Concurrency Practice and Experience, 16(2-3), pages 271-292, 2004*

System reaction to code changes: physicist's view

Expert or automatic advices based on additional information in the repository!

Focus optimizations to speed up search: which/where?



Grigori Fursin, Mike O'Boyle, Olivier Temam, and Gregory Watts. ***Fast and Accurate Method for Determining a Lower Bound on Execution Time.*** *Concurrency Practice and Experience, 16(2-3), pages 271-292, 2004*

Implementation in open-source cTuning[1] framework

cd [application_directory]

make CC=**icc** CC_OPTS=**-fast**

or

icc -**fast** *.c

time ./a.out < [**my_dataset**] > [output]

record "-fast", execution time

Implementation in open-source cTuning[1] framework

cd [application_directory]

make CC=**icc** CC_OPTS=**-fast**

or

icc -**fast** *.c

time ./a.out < [**my_dataset**] > [output]

record "-fast", execution time

ccc-comp build="make" compiler=icc opts="-fast"

ccc-comp compiler=icc opts="-fast"

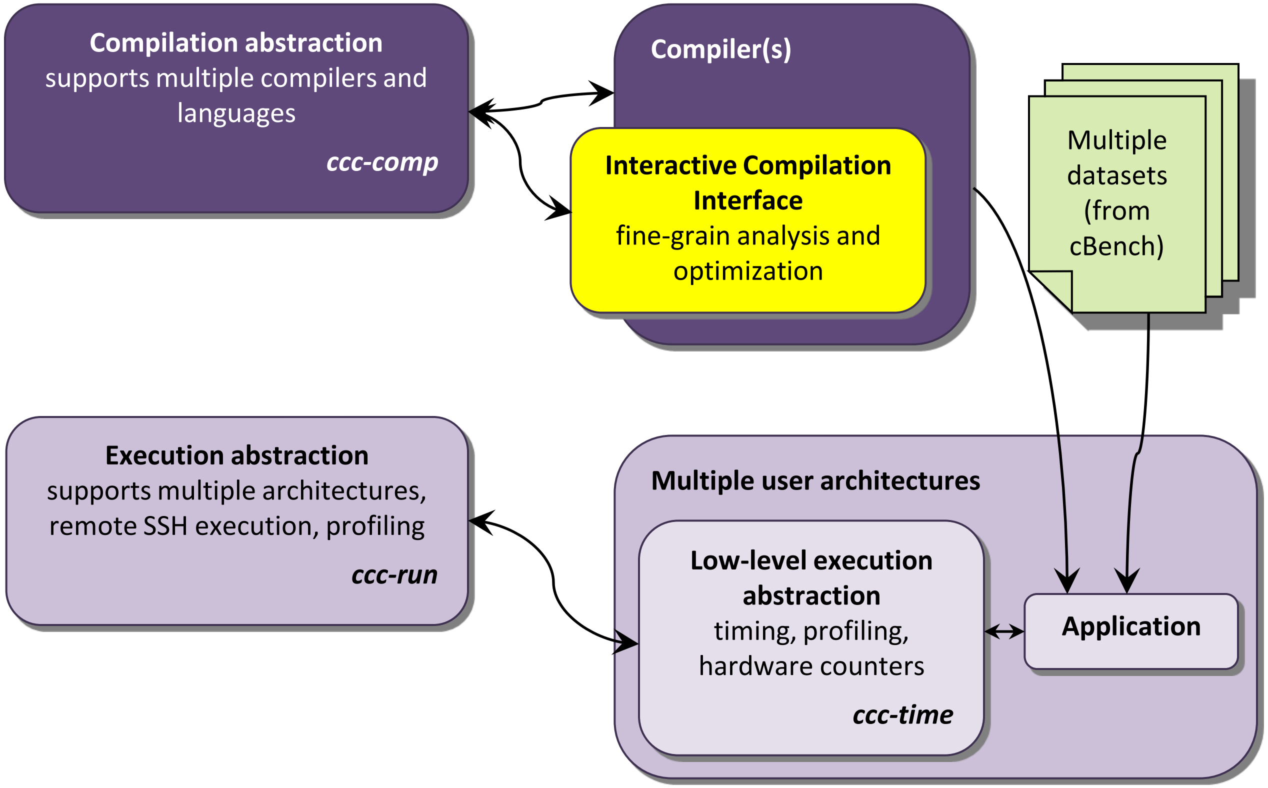
ccc-run prog=./a.out cmd="< [my dataset]"

ccc-time <cmd>

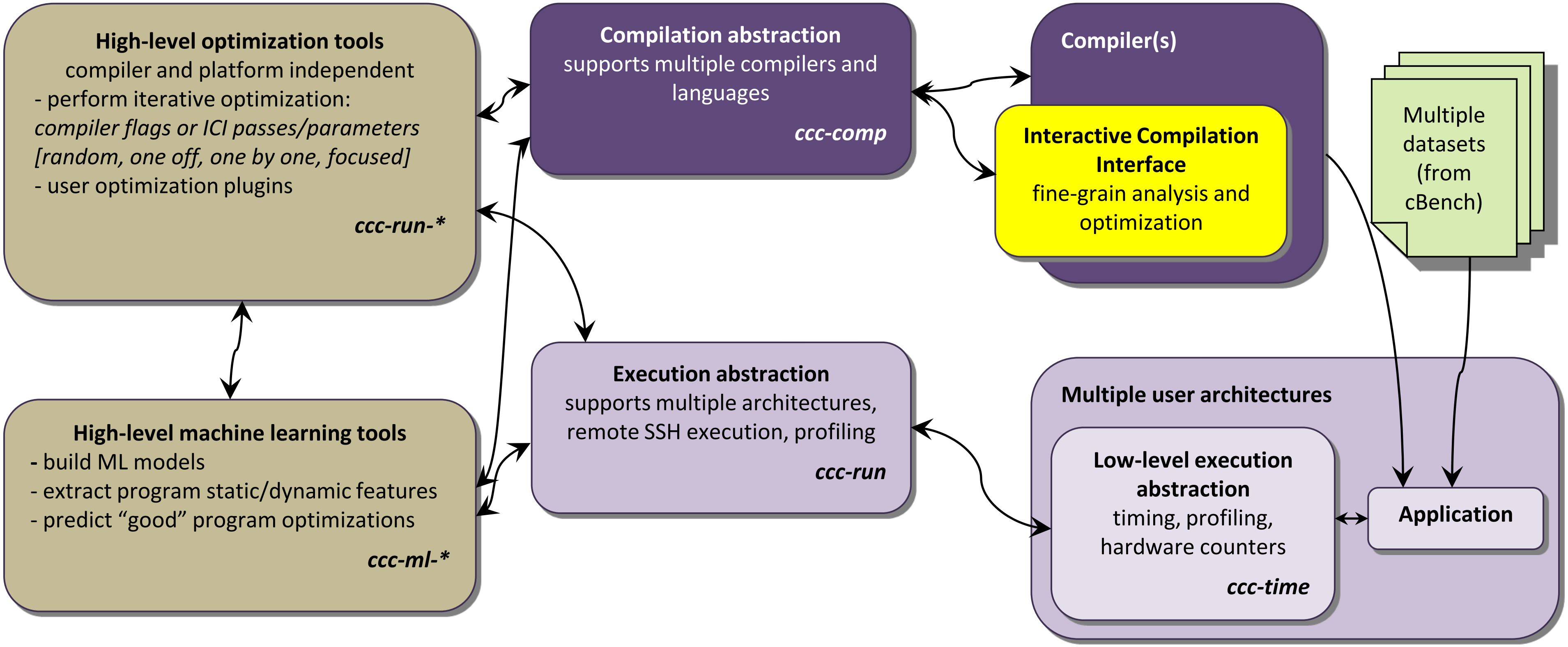
ccc-record-stats <file_with_stats>

- *Low level platform-dependent plugins in C*
- *Communication through text file or directly through MySQL database*
- *High level platform-independent exploration or analysis plugins in PHP*
- *Web services at cTuning.org as plugins in PHP*

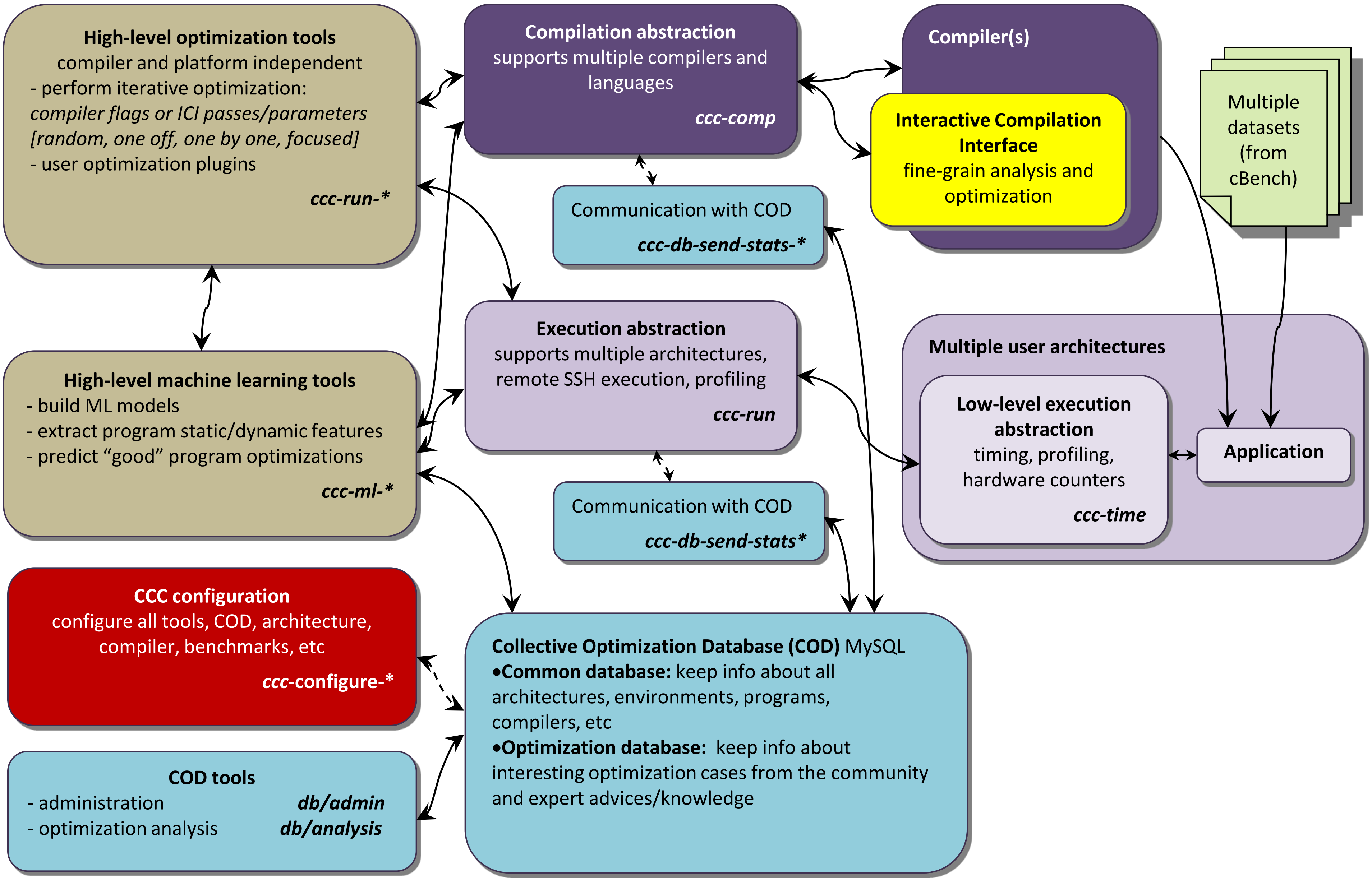
Implementation in open-source cTuning[1] framework



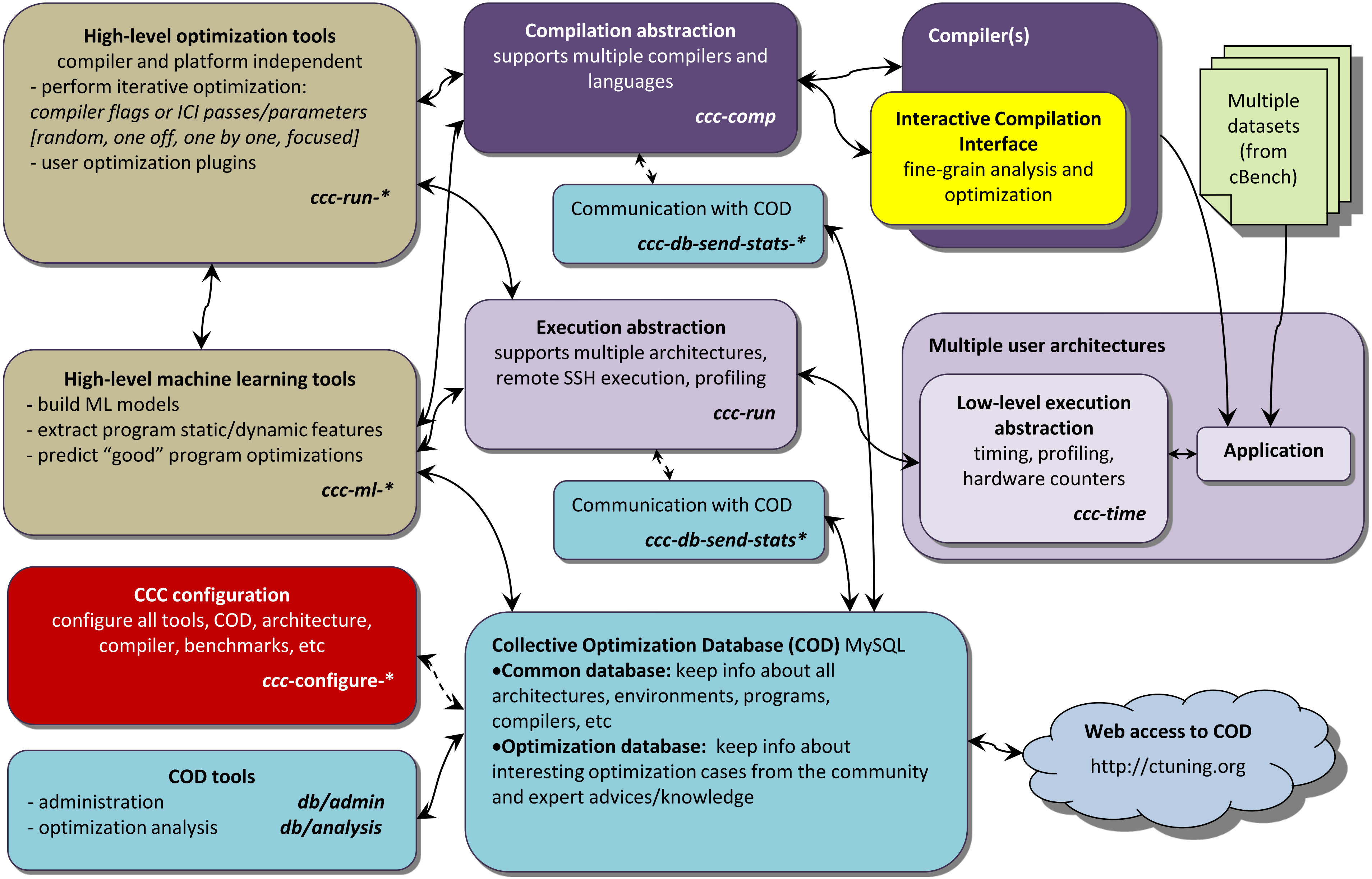
Implementation in open-source cTuning[1] framework



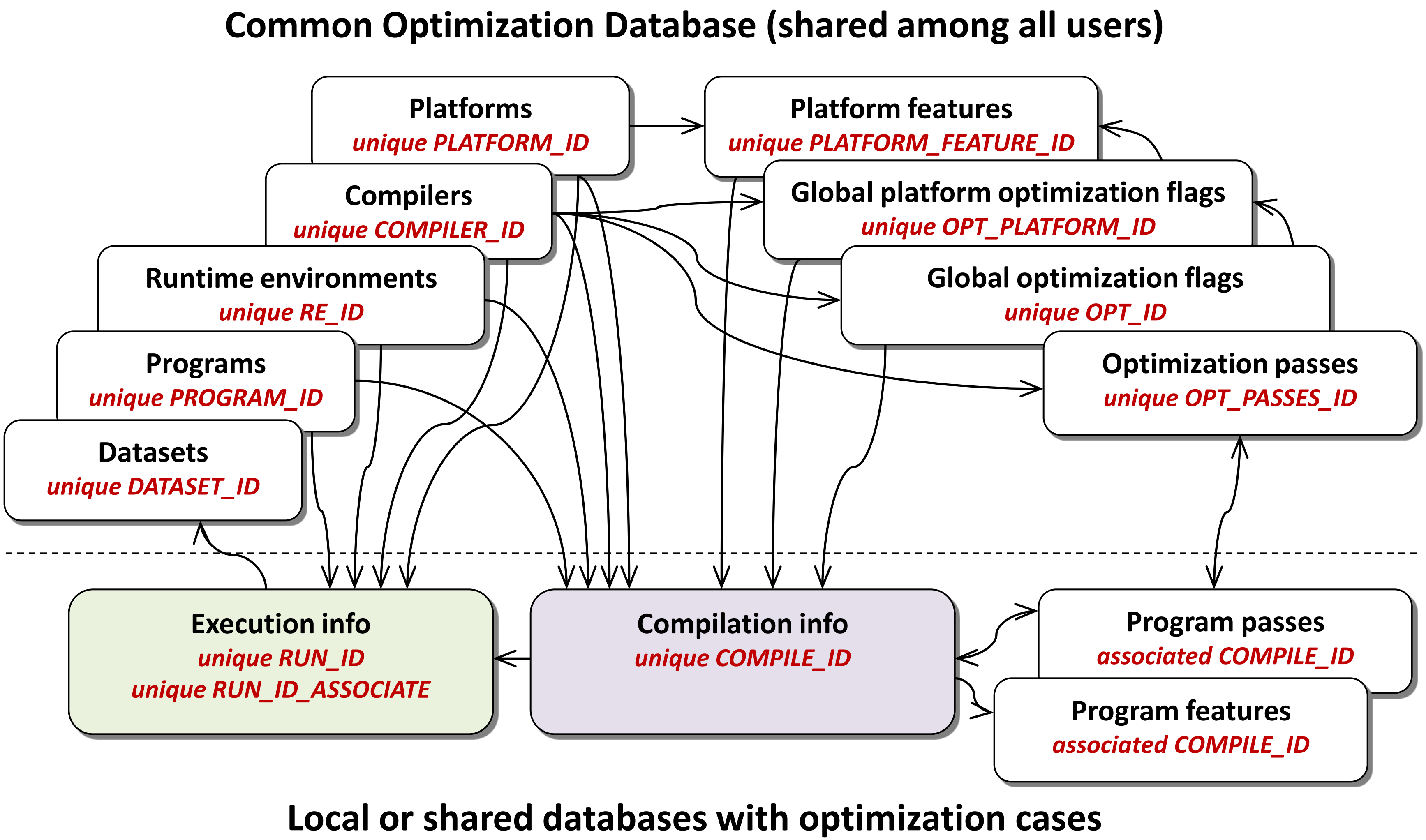
Implementation in open-source cTuning[1] framework



Implementation in open-source cTuning$_1$ framework



MySQL-based Collective Optimization Database



Problems with $cTuning_1$

- Difficult to extend (C, various hardwired components, need to change schema and types in MySQL)
- No convenient way of sharing modules, benchmarks, data sets, models (manual, csv files, emails, etc)
- Problems with repository scalability
- Complex, hardwired interfaces

cTuning$_3$ aka Collective Mind framework basics

cd [application_directory]

make CC=**icc** CC_OPTS=**-fast**

or

icc -**fast** *.c

time ./a.out < [**my_dataset**] > [output]

record "-fast", execution time

cTuning$_3$ aka Collective Mind framework basics

cd [application_directory]

make CC=**icc** CC_OPTS=**-fast**

or

icc -fast *.c

time ./a.out < [**my_dataset**] > [output]

record "-fast", execution time

E nd-users or cM developers CMD

Universal cM FE

cM kernel

python

cM plugins (modules)

code.source *build*

compiler *build*

code *run*

…

python or any other language

cTuning$_3$ aka Collective Mind framework basics

cd [application_directory]

make CC=**icc** CC_OPTS=**-fast**

or

icc -fast *.c

time ./a.out < [**my_dataset**] > [output]

record "-fast", execution time

End-users or cM developers CMD

Universal cM FE

cM kernel

python

cM plugins (modules)

code.source *build*

compiler *build*

code *run*

...

python or any other language

cm [module name] [action] (param$_1$=value$_1$ param$_2$=value$_2$ … -- *unparsed command line*)

cm code.source build ct_compiler=icc13 ct_optimizations=-fast

cm compiler build -- icc -fast *.c

cm code run os=android binary=./a.out dataset=image-crazy-scientist.pgm

Should be able to run on any OS (Windows, Linux, Android, MacOS, etc)!

cTuning$_3$ aka Collective Mind framework basics

Simple and minimalistic high-level cM interface - **one function (!)**

should be easy to connect to any language if needed

schema and type-free (only strings) -

easily extended when needed for research (agile methodology)!

(python dictionary) *output* = cm_kernel.access ((python dictionary) *input*)

Input: {

cm_run_module_uoa	- cM plugin name (or some UID)
cm_action	- cM plugin action (function)
parameters	- *(module and action dependent)*

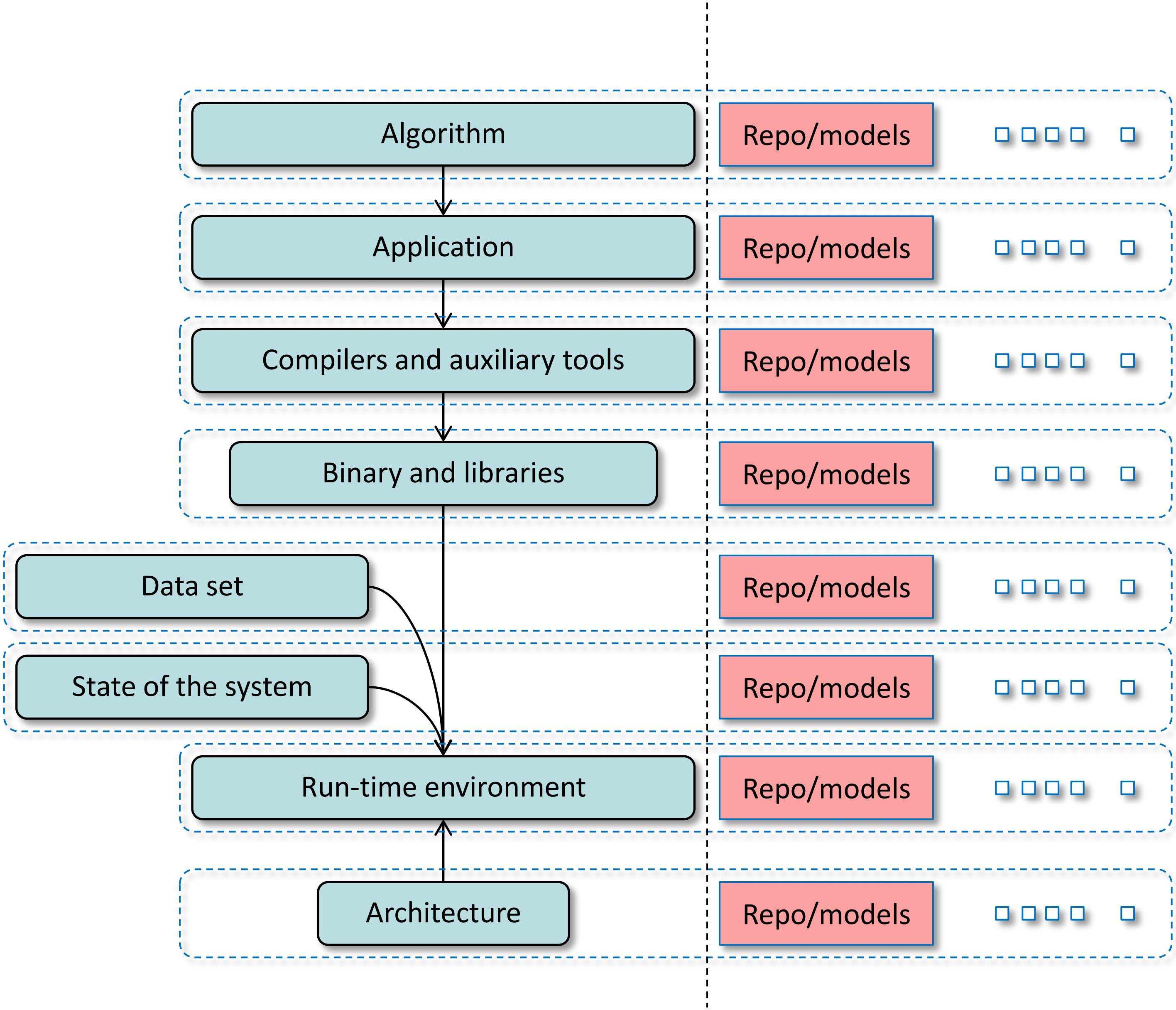
}

Output: {

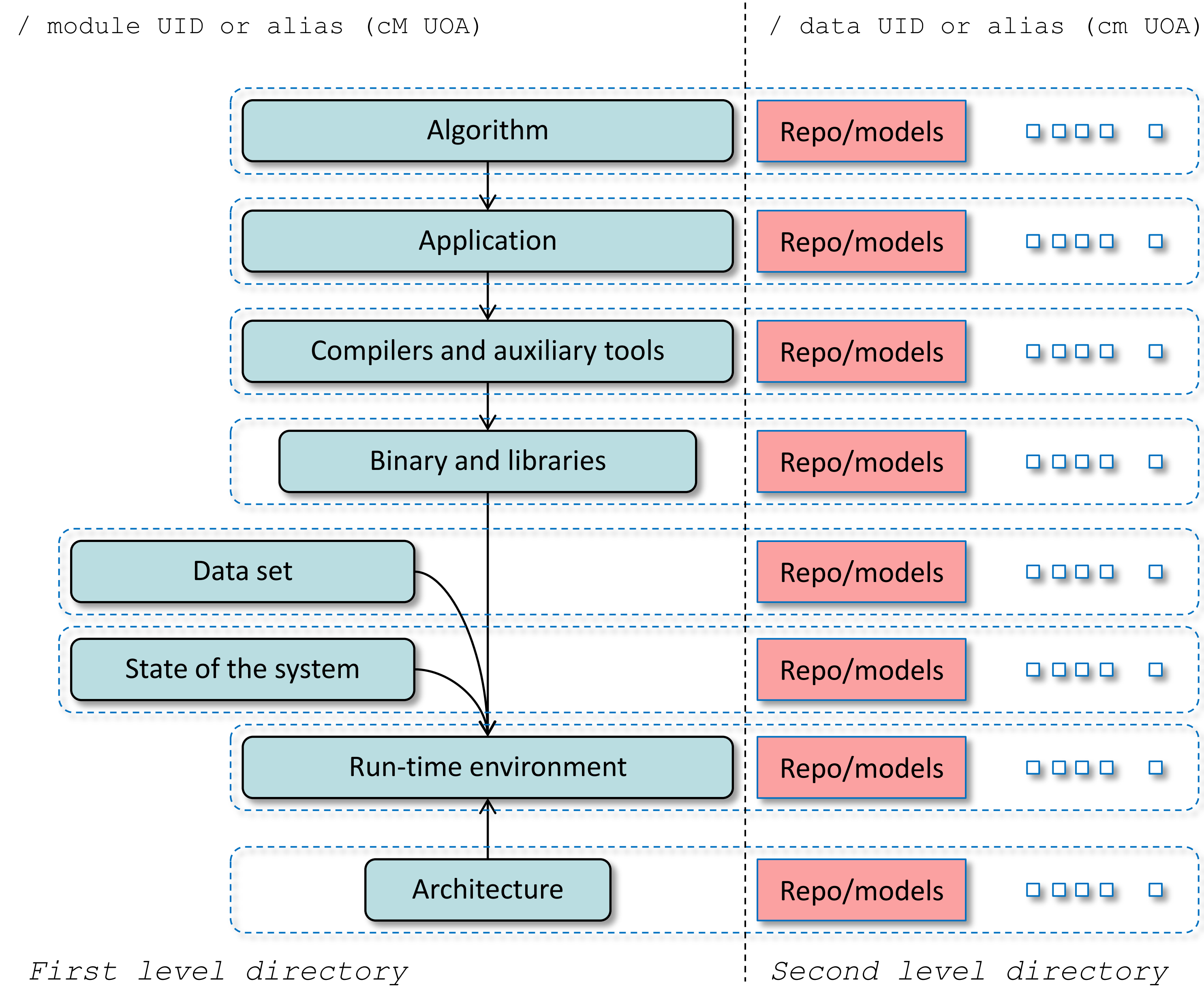
cm_return	- if 0, success if >0, error if <0, warning
cm_error	- if cm_return>0, error message
parameters	- *(module and action dependent)*

}

Collective Mind Repository basics



Collective Mind Repository basics



Very flexible and portable

Easy to access, edit and move data

Can be per application, experiment, architecture, etc

Can be easily shared (through web, SVN, GIT, FTP)

Schema-free extensible data meta-representation

cM uses **JSON** as internal data representation format:

JSON or JavaScript Object Notation, is a text-based open standard designed for human-readable data interchange (from Wikipedia)

- very intuitive to use and modify
- nearly native for python and php;   simple libraries for Java, C, C++, …
- easy to index with powerful indexing services (cM uses ElasticSearch)

cM records input and output of the module for reproducibility!

Data is referenced by CID:

`(Repository UID:) Module UID: Data UID`

Schema-free extensible data representation

Example of JSON entry ***ctuning.compiler:icc-12.x-linux***

```
{
  "all_compiler_flags_desc": {

    "##base_flag": {
      "type": "text"
      "desc_text": "compiler flag: -O3",
      "field_size": "7",

      "has_choice": "yes",
      "choice": [
        "-O0", "-O1", "-O2", "-Os", "-O3", "-fast"
      ],
      "default_value": "-O3",

      "explorable": "yes",
      "explore_level": "1",
      "explore_type": "fixed",
      "forbid_disable_at_random": "yes"
    },
    …
  }
…
}
```

Universal modules/functions

Task

Treat computer system as a black box

Result

python module that has a UID and alias
(for example, "code" / 688dee2e7014f1fb)

Input dictionary that can be represented as JSON
*{"run_cmd":'image',
"binary_name":"a.out",
"run_set_env":[],
"run_output_files":[],
"work_dir":"",
"run_time_out":"",
"code_deps":[],
"run_os_uoa":"",
...}*

Function
(for example "run")

invoke object, observe behavior and keep history

Execute code

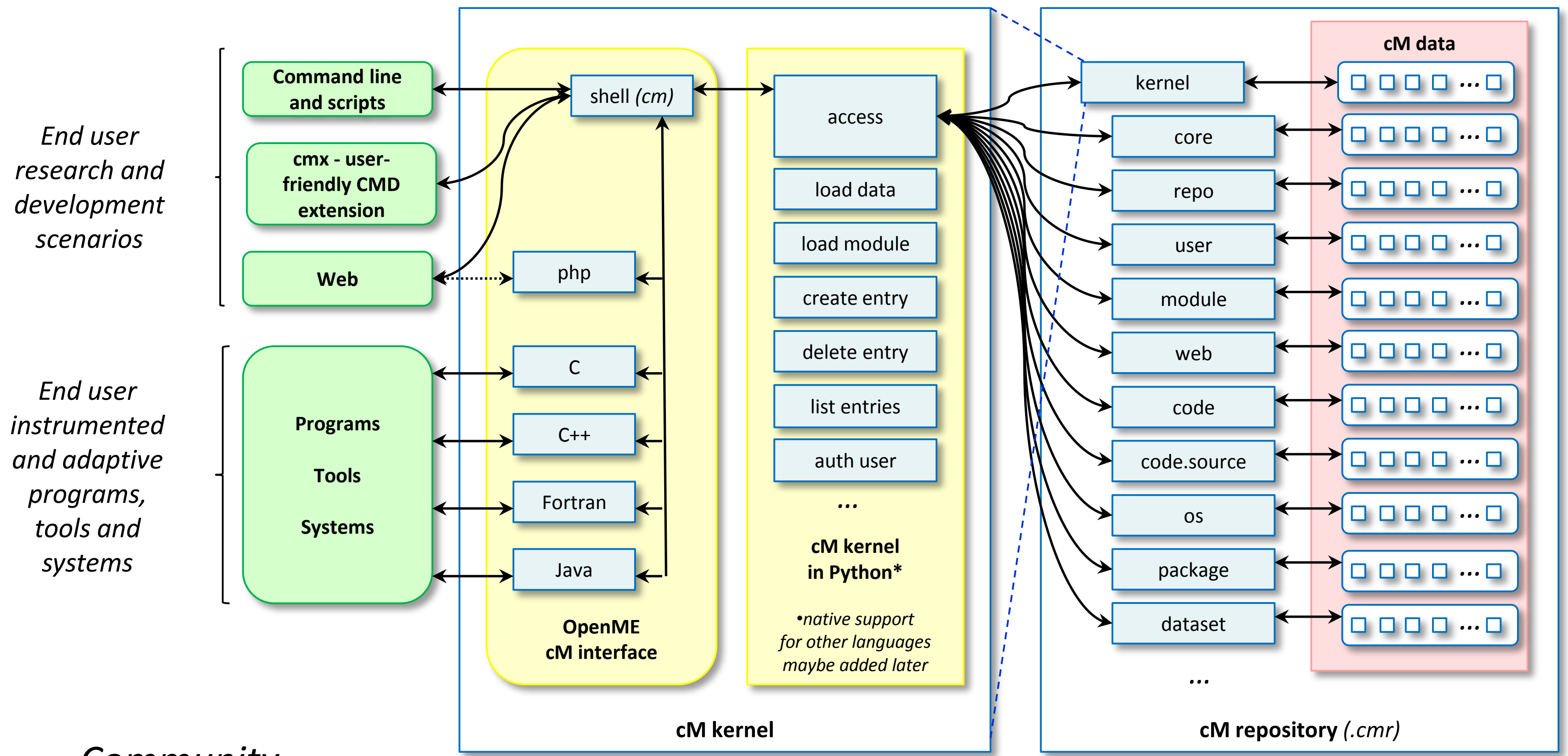
Output dictionary that can be represented as JSON
*{"cm_return":',
"failed_run":"",
"run_time_by_module":"",
...}*

Save input/output and related files

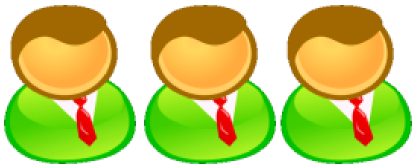
gradually describe input and output (choices, characteristics, etc)

Index input and output for fast search

Collective Mind overall structure



Community or workgroup

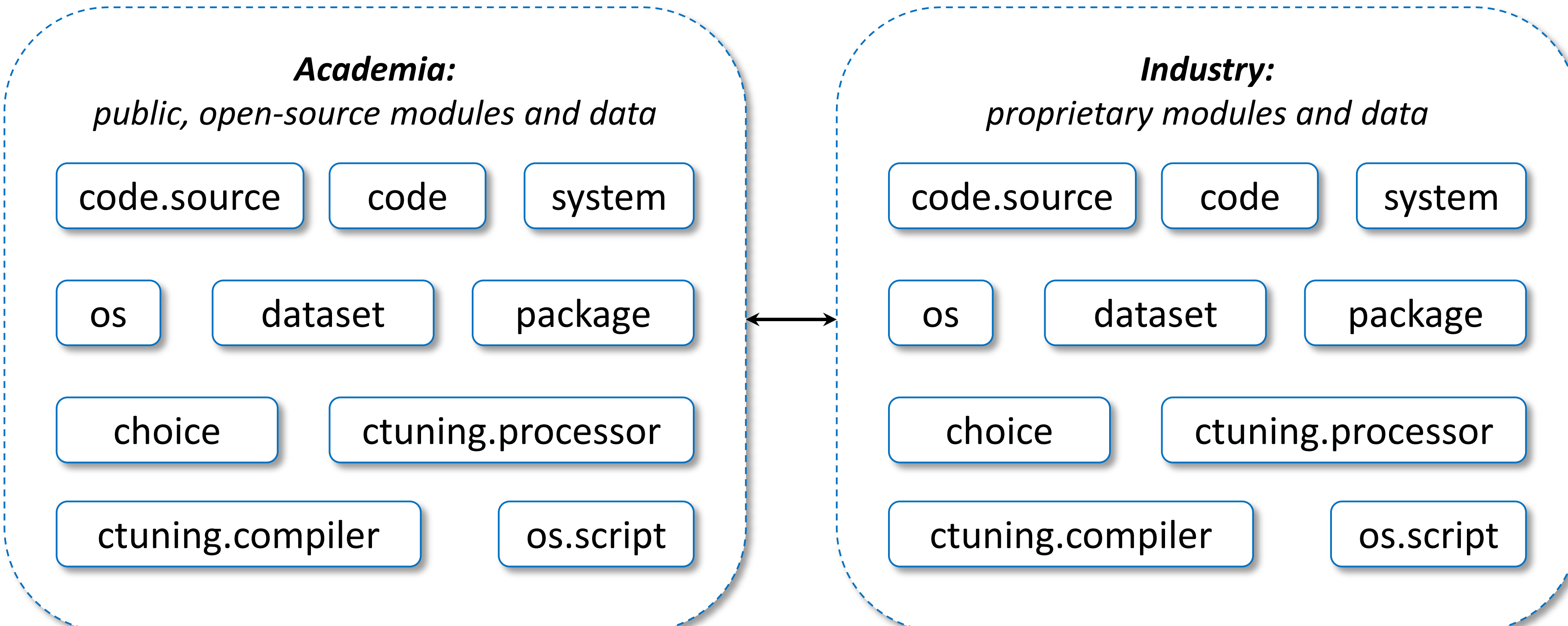


- Gradually add more modules, interfaces and data depending on user/project/company needs
- Gradually add more parameters
- Gradually expose choices, properties, characteristics

Collaborative, reproducible experiments: research LEGO

- Continuously adding "basic blocks" (modules)
- Adding tools, applications, datasets
- Gradually stabilize interfaces

Users can start connecting modules and data together to prepare experimental pipelines with various observation, characterization, auto-tuning and predictive scenarios!



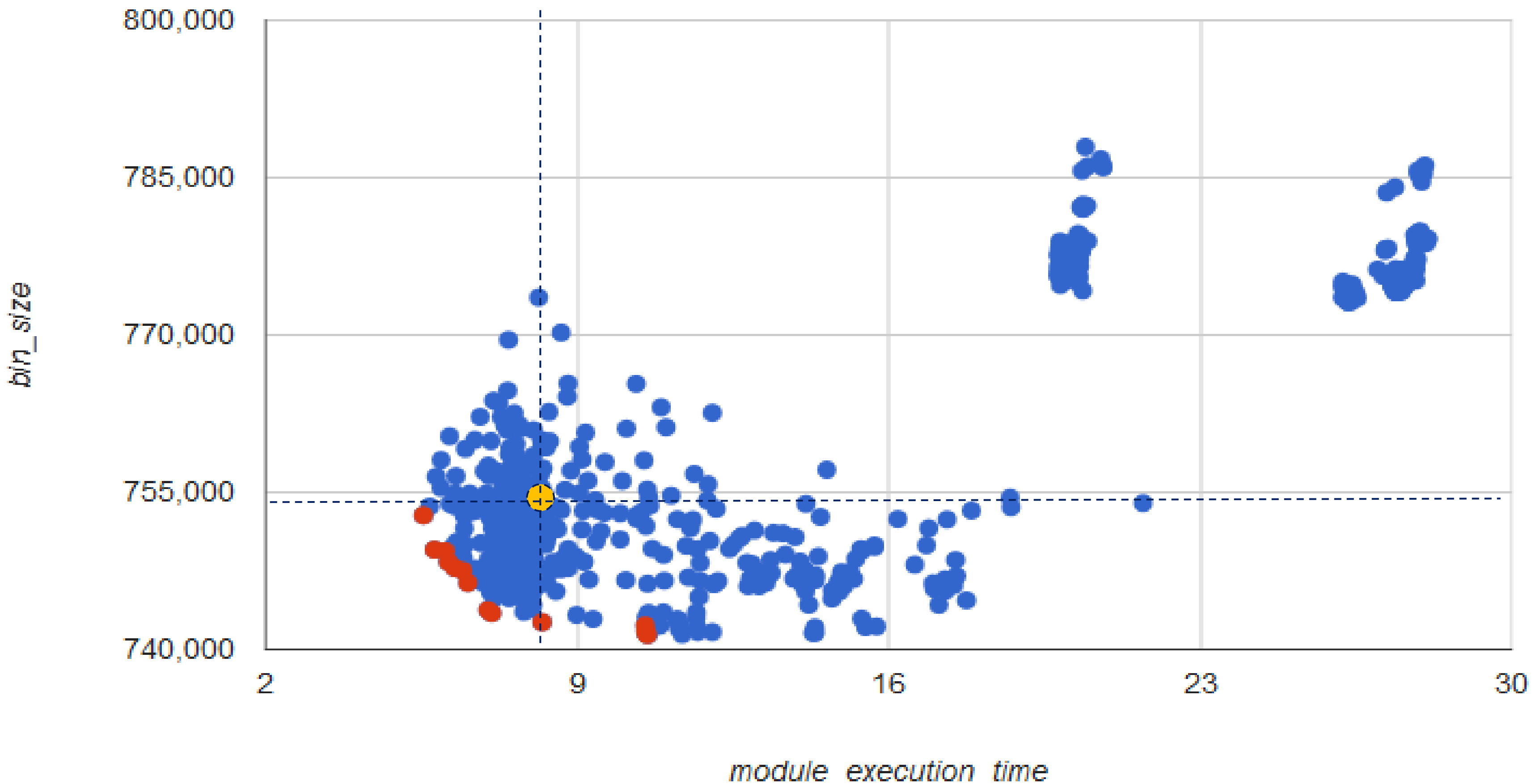
Experimental pipelines for auto-tuning and modeling

- **Init pipeline**
- Detected system information
- Initialize parameters
- Prepare dataset
- **Clean program**
- **Prepare compiler flags**
- Use compiler profiling
- Use cTuning CC/MILEPOST GCC for fine-grain program analysis and tuning
- Use universal Alchemist plugin (with any OpenME-compatible compiler or tool)
- Use Alchemist plugin (currently for GCC)
- **Build program**
- Get objdump and md5sum (if supported)
- Use OpenME for fine-grain program analysis and online tuning (build & run)
- Use 'Intel VTune Amplifier' to collect hardware counters
- Use 'perf' to collect hardware counters
- Set frequency (in Unix, if supported)
- Get system state before execution
- **Run program**
- Check output for correctness (use dataset UID to save different outputs)
- Finish OpenME
- Misc info
- **Observed characteristics**
- Observed statistical characteristics
- **Finalize pipeline**

Currently prepared experiments

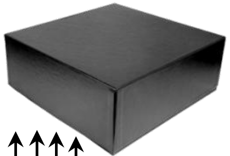
- Polybench - numerical kernels with exposed parameters of all matrices in cM
  - CPU: 28 prepared benchmarks
  - CUDA: 15 prepared benchmarks
  - OpenCL: 15 prepared benchmarks
- cBench - 23 benchmarks with 20 and 1000 datasets per benchmark
- Codelets - 44 codelets from embedded domain (provided by CAPS Entreprise)
- SPEC 2000/2006
- Description of 32-bit and 64-bit OS: Windows, Linux, Android
- Description of major compilers: GCC 4.x, LLVM 3.x, Open64/Pathscale 5.x, ICC 12.x
- Support for collection of hardware counters: perf, Intel vTune
- Support for frequency modification
- Validated on laptops, mobiles, tables, GRID/cloud - can work even from the USB key

Visualize and analyze optimization spaces



Program:	*cBench: susan corners*	Processor:	*ARM v6, 830MHz*
Compiler:	*Sourcery GCC for ARM v4.6.1*	OS:	*Android OS v2.3.5*
System:	*Samsung Galaxy Y*	Data set:	*MiDataSet #1, image, 600x450x8b PGM, 263KB*

Gradually increase granularity and complexity

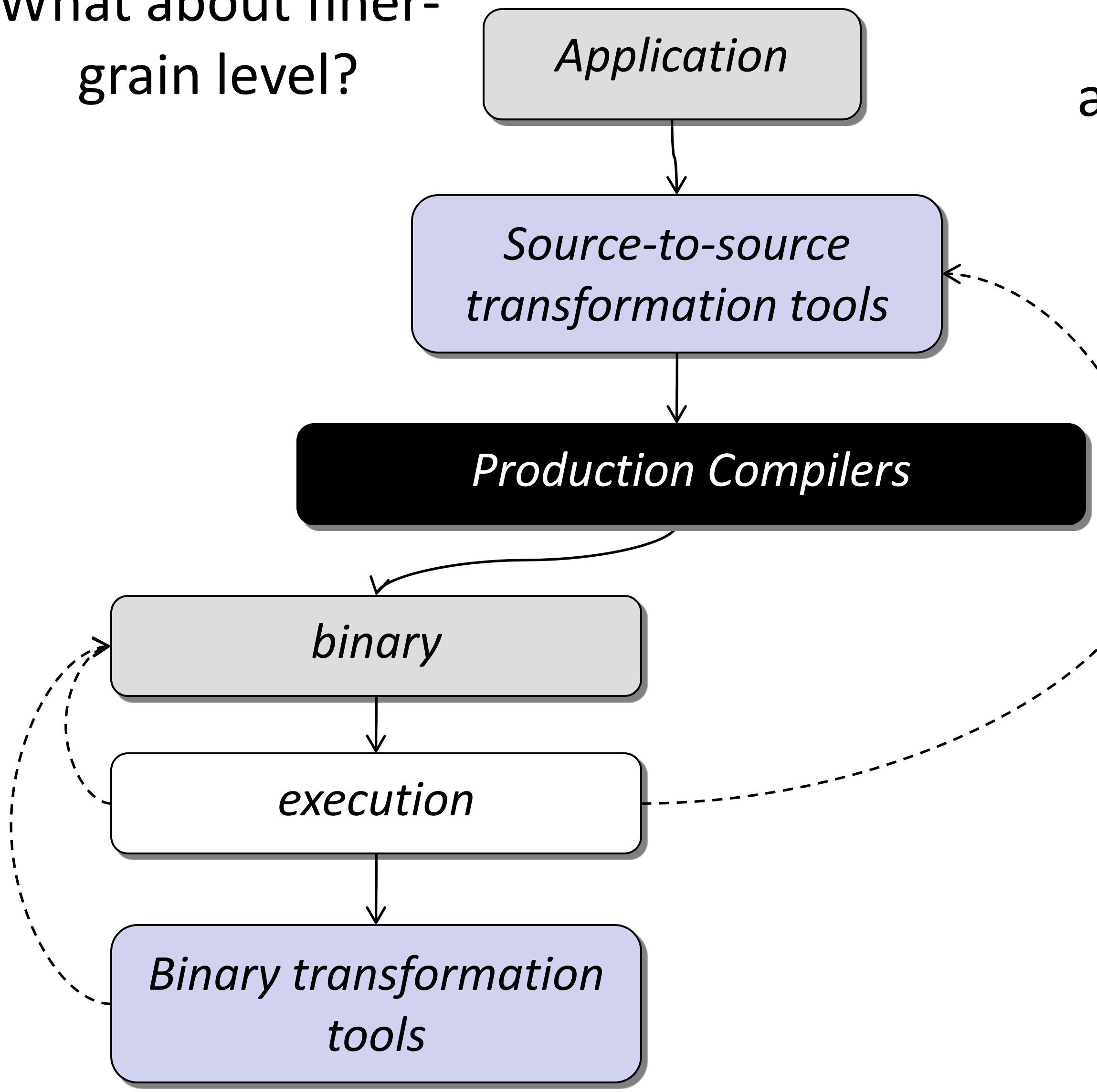


				Gradually expose some characteristics	Gradually expose some choices
	Algorithm selection			(time) productivity, variable-accuracy, complexity …	Language, MPI, OpenMP, TBB, MapReduce …
Compile Program				time …	compiler flags; pragmas …
	Code analysis & Transformations			time; memory usage; code size …	transformation ordering; polyhedral transformations; transformation parameters; instruction ordering …
		Process			
		Thread			
		Function			
		Codelet			
		Loop			
			Instruction		
Run code	Run-time environment			time; power consumption …	pinning/scheduling …
	System			cost; size …	CPU/GPU; frequency; memory hierarchy …
	Data set			size; values; description …	precision …
	Run-time analysis			time; precision …	hardware counters; power meters …
	Run-time state			processor state; cache state …	helper threads; hardware counters …
Analyze profile				time; size …	instrumentation; profiling …

Coarse-grain vs. fine-grain effects: depends on user requirements and expected ROI

Interactive compilers, tools and applications

What about finer-grain level?



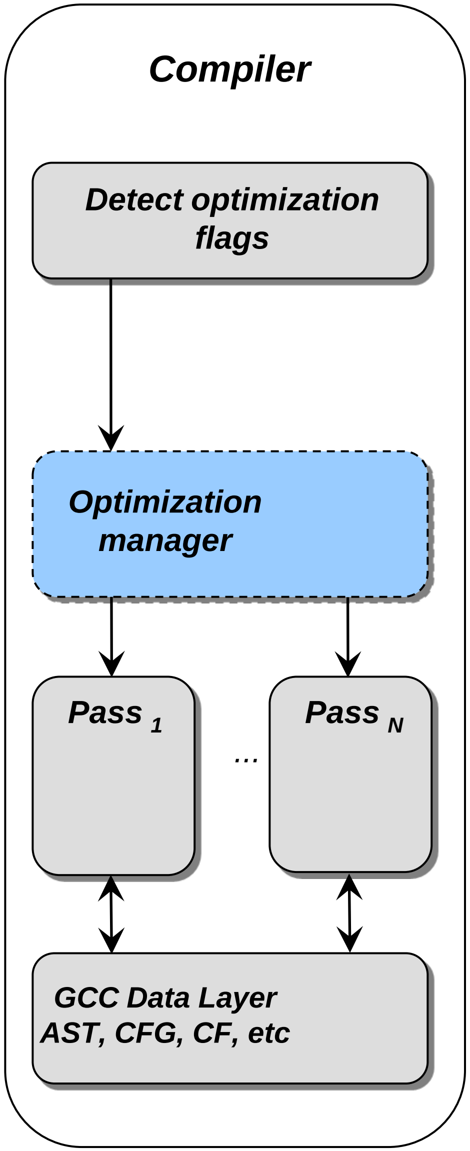
Traditional compilation, analysis and optimization

Often internal compiler decisions are not known or there is no precise control even through pragmas.

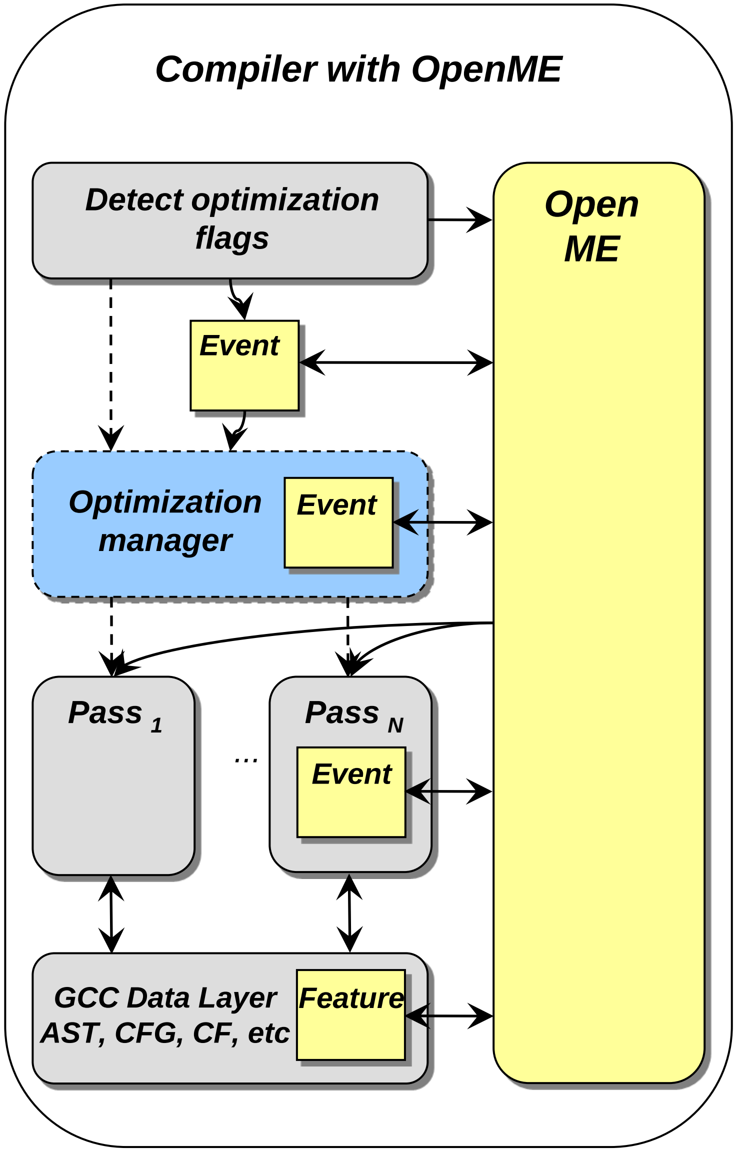
Interference with internal compiler optimizations complicates program analysis and characterization.

Current pragma based auto-tuning frameworks are very complex.

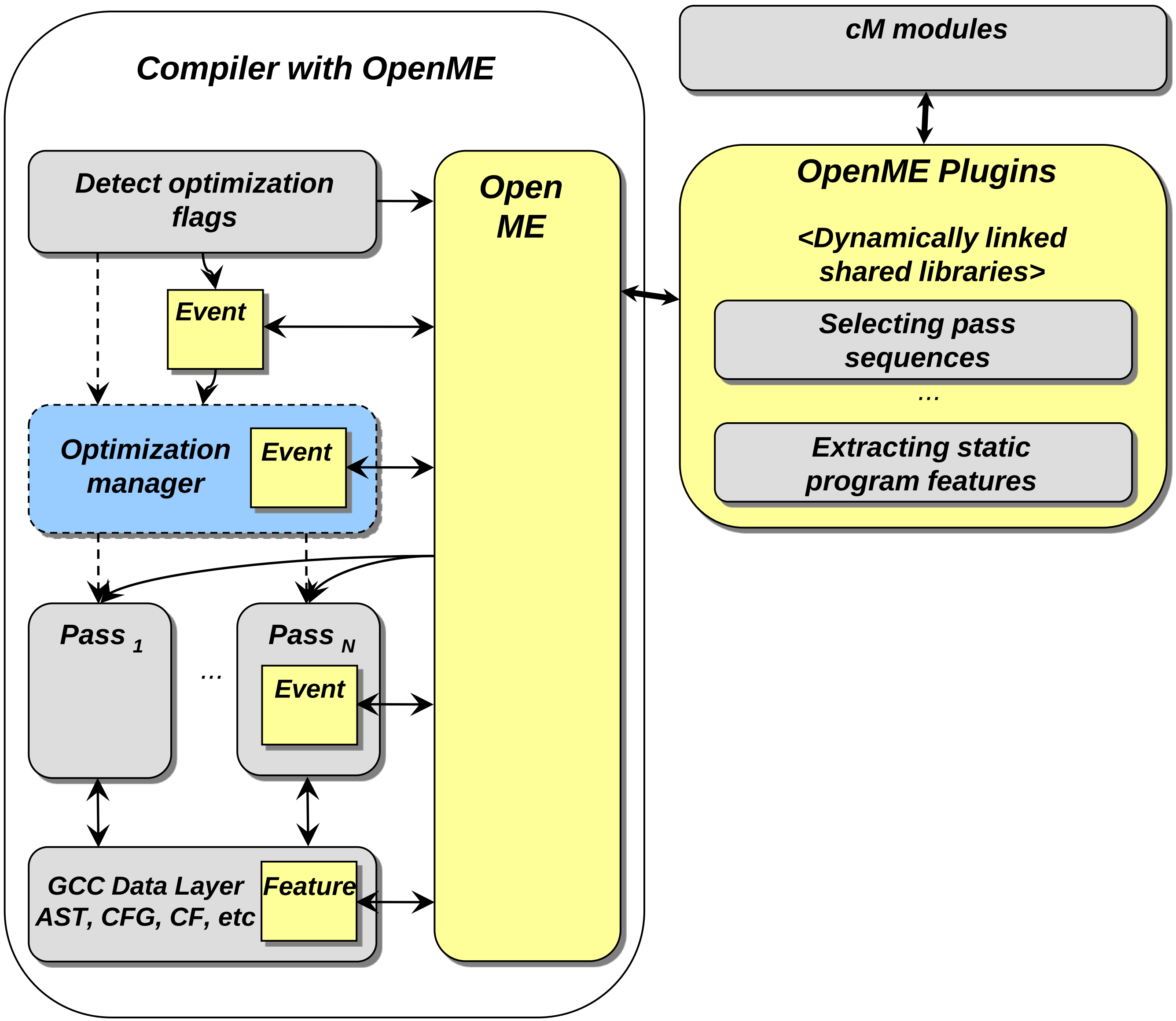
OpenME - interactive plugin and event-based interface to "open up" applications and tools



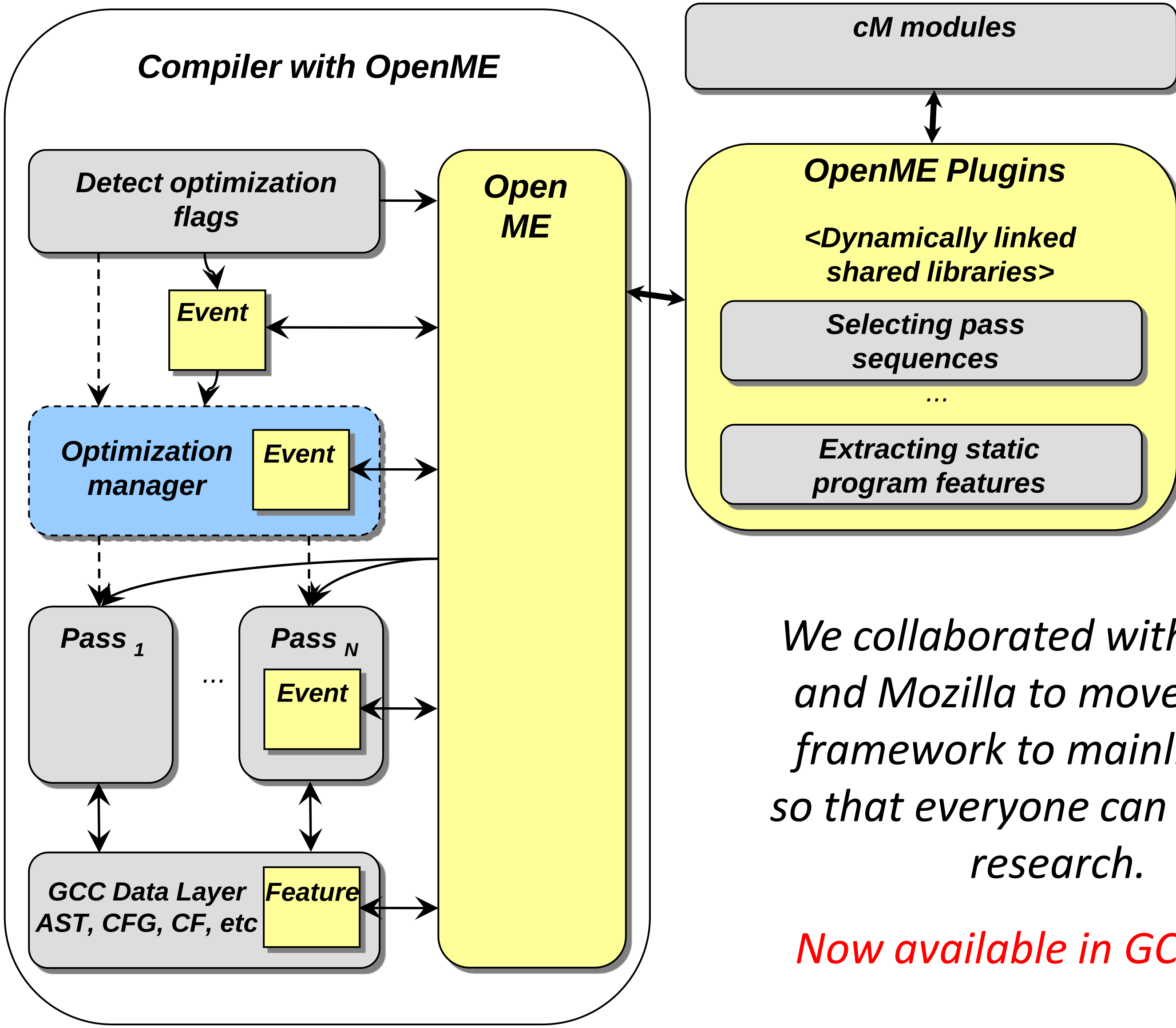
OpenME - interactive plugin and event-based interface to "open up" applications and tools



OpenME - interactive plugin and event-based interface to "open up" applications and tools



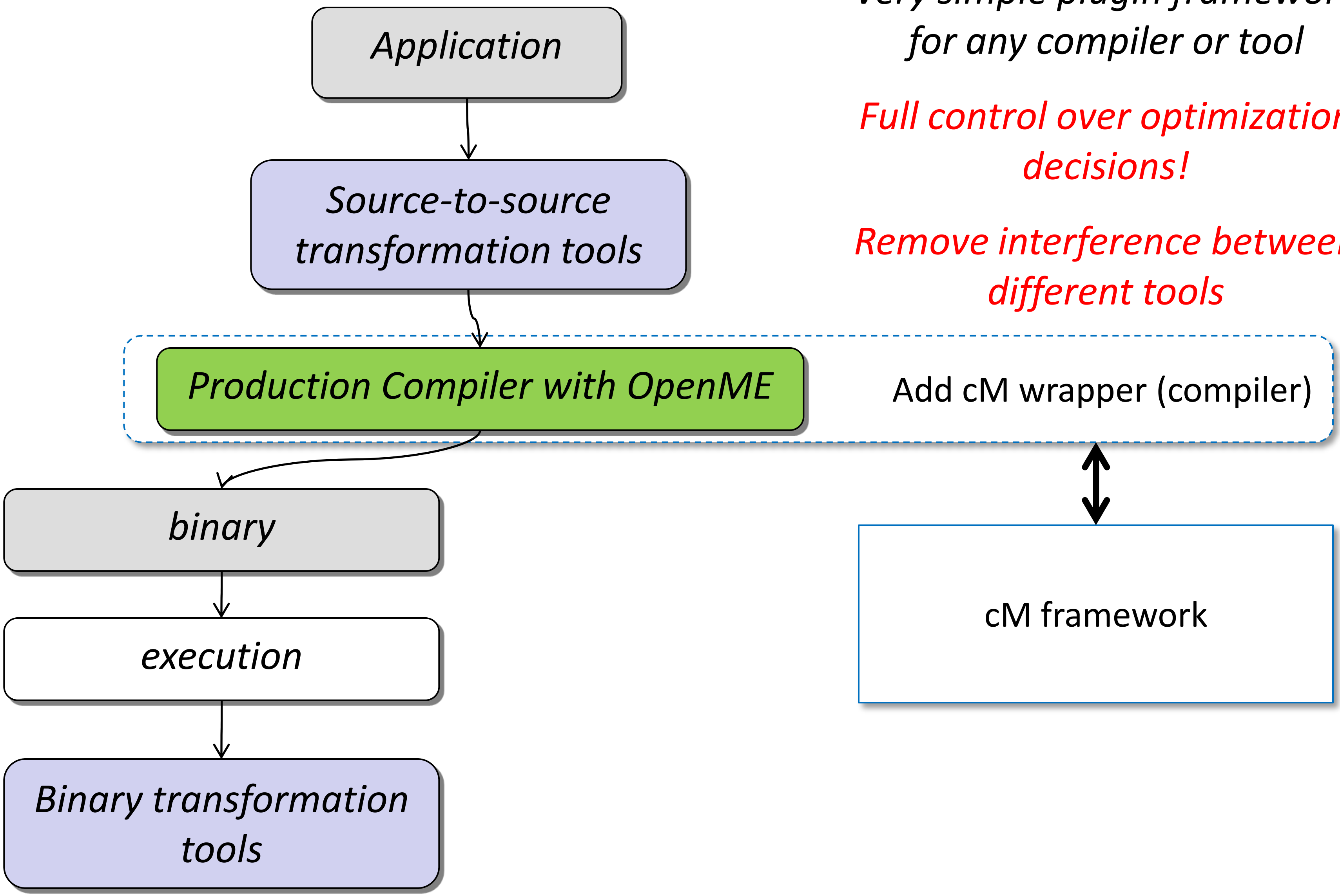
OpenME - interactive plugin and event-based interface to "open up" applications and tools



We collaborated with Google and Mozilla to move similar framework to mainline GCC so that everyone can use it for research.

Now available in GCC >=4.6

OpenME - interactive plugin and event-based interface to "open up" applications and tools



Very simple plugin framework for any compiler or tool

Full control over optimization decisions!

Remove interference between different tools

Example of OpenME for LLVM 3.2

OpenME: 3 functions only!

- *openme_init(…)* ***- initialize/load plugin***
- *openme_callback(char* event_name, void* params)* ***- call event***
- *openme_finish(…)* ***- finalize (if needed)***

tools/clang/tools/driver/cc1_main.cpp

```
#include "openme.h“
…
int cc1_main(const char **ArgBegin, const char **ArgEnd,
             const char *Argv0, void *MainAddr) {

  openme_init("UNI_ALCHEMIST_USE", "UNI_ALCHEMIST_PLUGINS", NULL, 0);
  …
 // Execute the frontend actions.
 Success = ExecuteCompilerInvocation(Clang.get());
 openme_callback("ALC_FINISH", NULL);
  …
}
```

Example of OpenME for LLVM 3.2

lib/Transforms/Scalar/LoopUnrollPass.cpp

```
#include <cJSON.h>
#include "openme.h“
…
bool LoopUnroll::runOnLoop(Loop *L, LPPassManager &LPM) {

  struct alc_unroll   {
    const char *func_name;
    const char *loop_name;
    cJSON *json;
    int factor;
   } alc_unroll;
…
alc_unroll.func_name=(Header->getParent()->getName()).data();
alc_unroll.loop_name=(Header->getName()).data();
openme_callback("ALC_TRANSFORM_UNROLL_INIT", &alc_unroll);
…
 // Unroll the loop.
alc_unroll.factor=Count;
openme_callback("ALC_TRANSFORM_UNROLL", &alc_unroll);
Count=alc_unroll.factor;

if (!UnrollLoop(L, Count, TripCount, UnrollRuntime, TripMultiple, LI, &LPM))
   return false;
…
}
```

Example of OpenME for LLVM 3.2

Alchemist plugin (.so/dll object) - in development
for online/interactive analysis, tuning and adaptation

```
#include <cJSON.h>
#include <openme.h>

int openme_plugin_init(struct openme_info *ome_info) {
  …
   openme_register_callback(ome_info, "ALC_TRANSFORM_UNROLL_INIT", alc_transform_unroll_init);
  openme_register_callback(ome_info, "ALC_TRANSFORM_UNROLL", alc_transform_unroll);
  openme_register_callback(ome_info, "ALC_TRANSFORM_UNROLL_FEATURES", alc_transform_unroll_features);
  openme_register_callback(ome_info, "ALC_FINISH", alc_finish);
  …
}

extern void alc_transform_unroll_init(struct alc_unroll *alc_unroll){
   …
}

extern void alc_transform_unroll(struct alc_unroll *alc_unroll) {
  …
}
…
```

Example of OpenME for OpenCL/CUDA C application

- **2mm.c / 2mm.cu**

```
...
#ifdef OPENME
#include <openme.h>
#endif
...

int main(void) {
...
#ifdef OPENME
  openme_init(NULL,NULL,NULL,0);
  openme_callback("PROGRAM_START", NULL);
#endif
...
#ifdef OPENME
openme_callback("ACC_KERNEL_START", NULL);
#endif

cl_launch_kernel();
  or
mm2Cuda(A, B, C, D, E, E_outputFromGpu);

#ifdef OPENME
openme_callback("ACC_KERNEL_END", NULL);
#endif
...
```

```
...
#ifdef OPENME
 openme_callback("KERNEL_START", NULL);
#endif

mm2_cpu(A, B, C, D, E);

#ifdef OPENME
  openme_callback("KERNEL_END", NULL);
#endif

#ifdef OPENME
openme_callback("PROGRAM_END", NULL);
#endif
...
}
```

Example of OpenME for Fortran application

- **matmul.F**

```
 PROGRAM MATMULPROG
…

 INTEGER*8 OBJ, OPENME_CREATE_OBJ_F
CALL OPENME_INIT_F(""//CHAR(0), ""//CHAR(0), ""//CHAR(0), 0)
CALL OPENME_CALLBACK_F("PROGRAM_START"//CHAR(0))

…
 CALL OPENME_CALLBACK_F("KERNEL_START"//CHAR(0));
DO I=1, I_REPEAT
  CALL MATMUL
END DO
CALL OPENME_CALLBACK_F("KERNEL_END"//CHAR(0));

…
 CALL OPENME_CALLBACK_F("PROGRAM_END"//CHAR(0))
END
```

Next steps

1) Prepare pre-release around May/June 2013 (BSD-style license) - **ASK for preview!**
2) Reproduce my past published research within new framework:
   - Add “classical” classification and predictive models
   - Add various exploration strategies (random, focused)
   - Add run-time adaptation scenarios (CUDA/OpenCL scheduling, pinning, etc)
   - Add co-design scenarios
3) Use framework for analysis and auto-tuning of industrial applications
4) Help to customize framework for industrial usages (consulting)
5) Applying for new funding (academic and industrial)
6) Continue virtual collaborative cTuning Lab to build community:
   - Public repository to share applications, datasets, models at cTuning.org:
   - New publication model for reproducible research
   - Community R&D discussion
     http://groups.google.com/group/collective-mind
   - Collect data from Android mobiles

Acknowledgements

- PhD students and postdocs (my Intel Exascale team)

  Abdul Wahid Memon, Pablo Oliveira, Yuriy Kashnikov

- Colleague from NCAR, USA

  Davide Del Vento and his colleagues/interns

- Colleagues from IBM, CAPS, ARC (Synopsys), Intel, Google, ARM, ST
- Colleagues from Intel (USA)

  David Kuck and David Wong

- cTuning community:

- EU FP6, FP7 program and HiPEAC network of excellence

  http://www.hipeac.net

Main references

• Grigori Fursin. **Collective Tuning Initiative: automating and accelerating development and optimization of computing systems.** Proceedings of the GCC Summit'09, Montreal, Canada, June 2009

• Grigori Fursin and Olivier Temam. **Collective Optimization: A Practical Collaborative Approach.** ACM Transactions on Architecture and Code Optimization (TACO), December 2010, Volume 7, Number 4, pages 20-49

• Grigori Fursin, Yuriy Kashnikov, Abdul Wahid Memon, Zbigniew Chamski, Olivier Temam, Mircea Namolaru, Elad Yom-Tov, Bilha Mendelson, Ayal Zaks, Eric Courtois, Francois Bodin, Phil Barnard, Elton Ashton, Edwin Bonilla, John Thomson, Chris Williams, Michael O'Boyle. **MILEPOST GCC: machine learning enabled self-tuning compiler.** International Journal of Parallel Programming (IJPP), June 2011, Volume 39, Issue 3, pages 296-327

• Yang Chen, Shuangde Fang, Yuanjie Huang, Lieven Eeckhout, Grigori Fursin, Olivier Temam and Chengyong Wu. **Deconstructing iterative optimization.** ACM Transactions on Architecture and Code Optimization (TACO), October 2012, Volume 9, Number 3

• Yang Chen, Yuanjie Huang, Lieven Eeckhout, Grigori Fursin, Liang Peng, Olivier Temam, Chengyong Wu. **Evaluating Iterative Optimization across 1000 Data Sets.** PLDI'10

• Victor Jimenez, Isaac Gelado, Lluis Vilanova, Marisa Gil, Grigori Fursin and Nacho Navarro. **Predictive runtime code scheduling for heterogeneous architectures.** HiPEAC'09

Main references

• Lianjie Luo, Yang Chen, Chengyong Wu, Shun Long and Grigori Fursin. **Finding representative sets of optimizations for adaptive multiversioning applications.** SMART'09 co-located with HiPEAC'09

• Grigori Fursin, John Cavazos, Michael O'Boyle and Olivier Temam. **MiDataSets: Creating The Conditions For A More Realistic Evaluation of Iterative Optimization.** HiPEAC’07

• F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M.F.P. O'Boyle, J. Thomson, M. Toussaint and C.K.I. Williams. **Using Machine Learning to Focus Iterative Optimization.** CGO’06

•Grigori Fursin, Albert Cohen, Michael O'Boyle and Oliver Temam. **A Practical Method For Quickly Evaluating Program Optimizations.** HiPEAC’05

•Grigori Fursin, Mike O'Boyle, Olivier Temam, and Gregory Watts. **Fast and Accurate Method for Determining a Lower Bound on Execution Time.** Concurrency Practice and Experience, 16(2-3), pages 271-292, 2004

• Grigori Fursin. **Iterative Compilation and Performance Prediction for Numerical Applications.** Ph.D. thesis, University of Edinburgh, Edinburgh, UK, January 2004