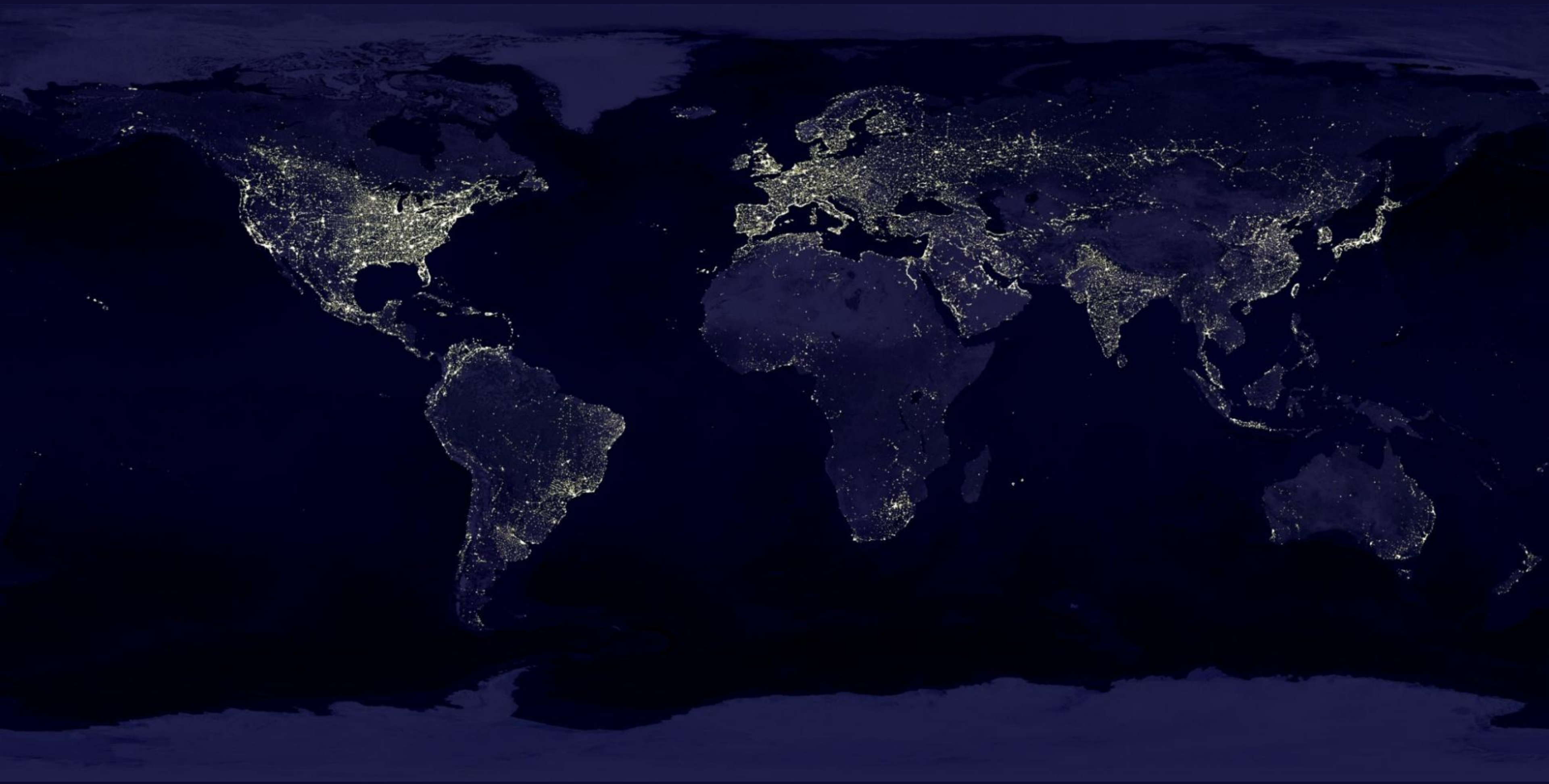
Collective characterization, optimization and design of computer systems



Grigori Fursin
INRIA, France

HiPEAC computing week
April 2012

Session introduction

HiPEAC$_3$ includes new instrument: **Thematic sessions.**

Evolution of HiPEAC$_2$ clusters and task forces.

Building a network of researchers for a given topic.

Address rising complexity of the design and optimization of computer systems through collaborative knowledge discovery, preservation, systematization, sharing and reuse!

Program

Time	Talk	Presenter
9:30-10:30	Collective SW/HW co-design: methodology, repository and tools	*Grigori Fursin* INRIA, France
10:30-11:00	Looking for key factors to improve runtime adaptation.	*Marisa Gil* UPC, Spain
11:00-11:30	Break	
11:30-12:00	Multi-core HW/SW interplay and energy efficiency	*Lasse Natvig* NTNU, Norway
12:00-12:30	Improving Both the Performance Benefits and Speed of Optimization Phase Sequence Searches.	*David Whalley* Florida State University, USA
12:30-13:00	Response Surface Modeling Techniques for Design Space Exploration of Multi-core Architectures.	*Cristina Silvano* Politecnico di Milano, Italy

Outline

- Background
- Motivation, challenges
- Collective co-design methodology
- Usage examples
- New publication model
- Conclusions
- References and tech. details

Interdisciplinary background

Year:	Machine learning, neural networks, brain modeling	Semiconductor electronics, physics	Manual program optimization	HPC and parallelization	Unified access to HPC resources through web	Systematic auto-tuning and performance prediction	Private design and optimization repositories	Static multi-versioning for dynamic adaptation	Machine learning for SW/HD co-design	Predictive scheduling for heterogeneous architectures	cTuning: methodology, repository and infrastructure for collaborative tuning – a physicist's view	Position:	Institution:
1993-1997	●	●	●				●					*B.S.* *physics and electronics*	MIPT, Russia
1997-1999	○			●	●		●				○	*M.S.* *computer engineering*	MIPT, Russia
1999-2004	○			○		●	●		○		○	*Ph.D.* *computer science*	University of Edinburgh, UK
2005-2007	●					●	●	●	●		○	*Postdoctoral researcher*	INRIA, France
2007-2010	●			●		●	●	●	●	●	●	*Tenured scientist*	INRIA, France
2010-2011			●	●	●	●	●	●	●	●	🔴	*Director of research and group manager*	Intel Exascale Lab, France
2012-cur.	🔴	?	●	●	●	●	●	●	●	●	🔴	*Tenured scientist*	INRIA, France

Back to basics

End user

Task

Solution

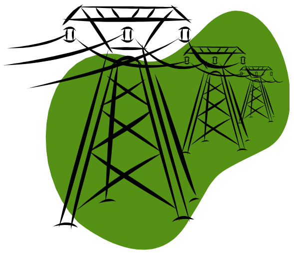
User requirements:

most common:

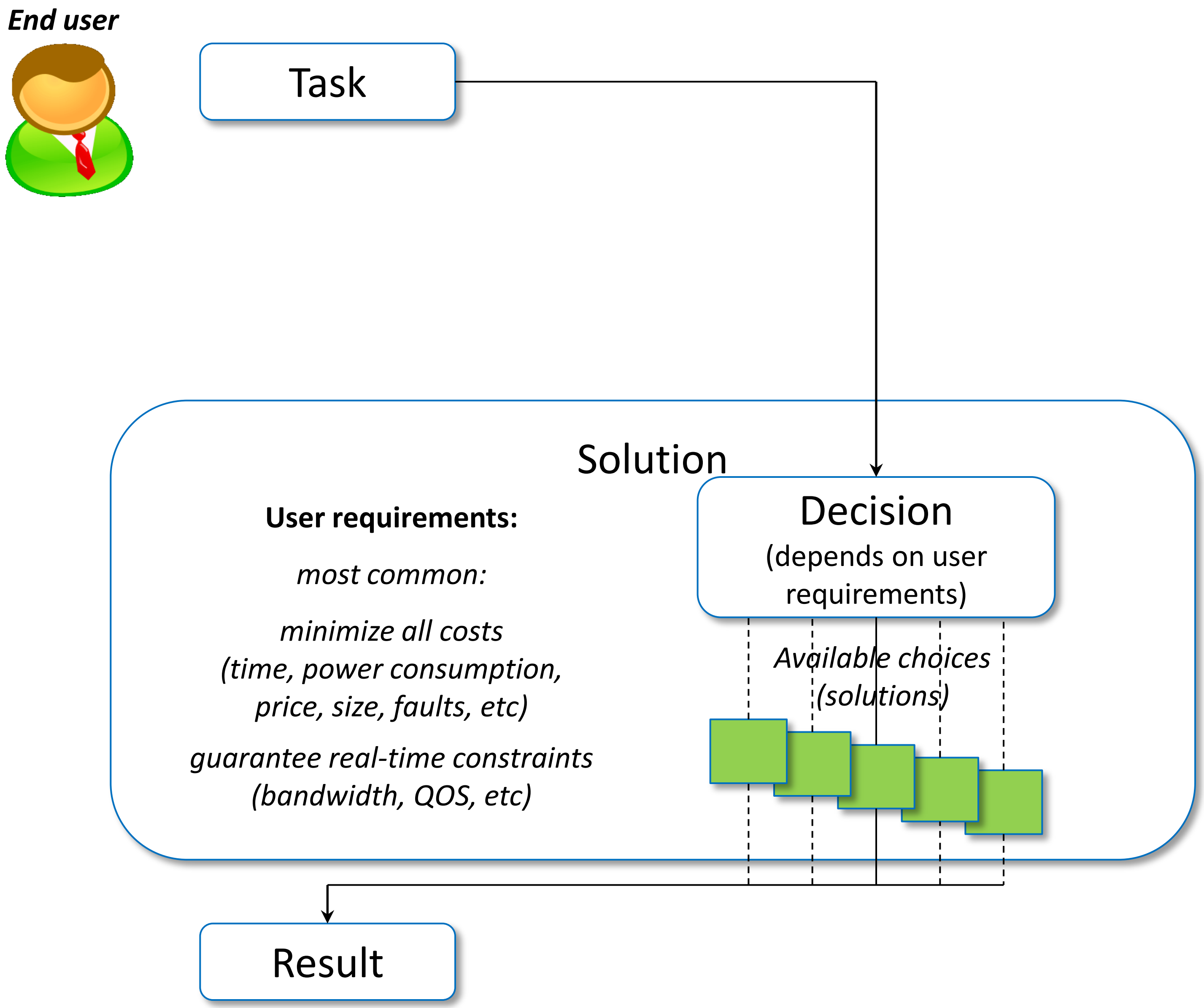
minimize all costs
(time, power consumption,
price, size, faults, etc)

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

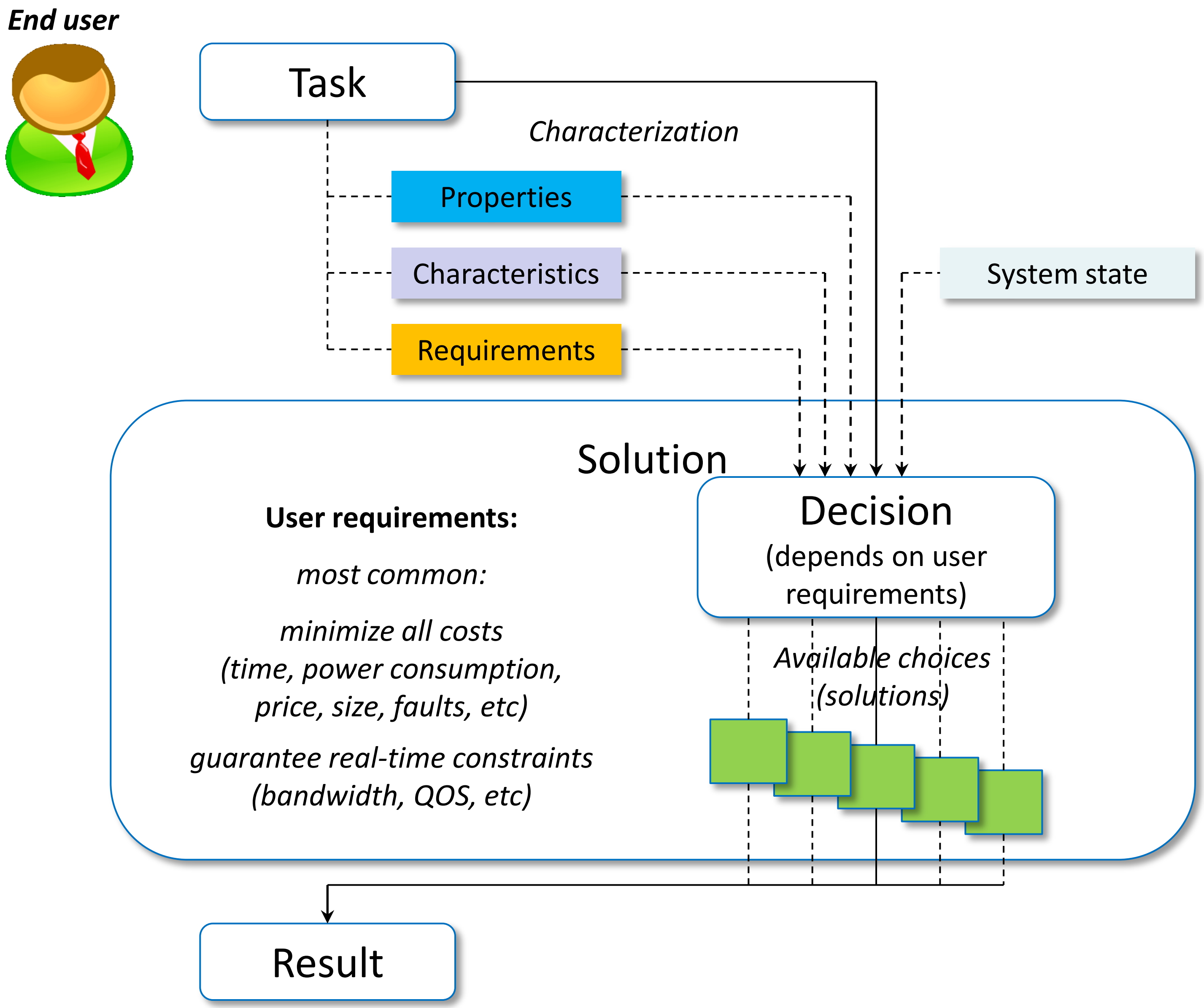
Result



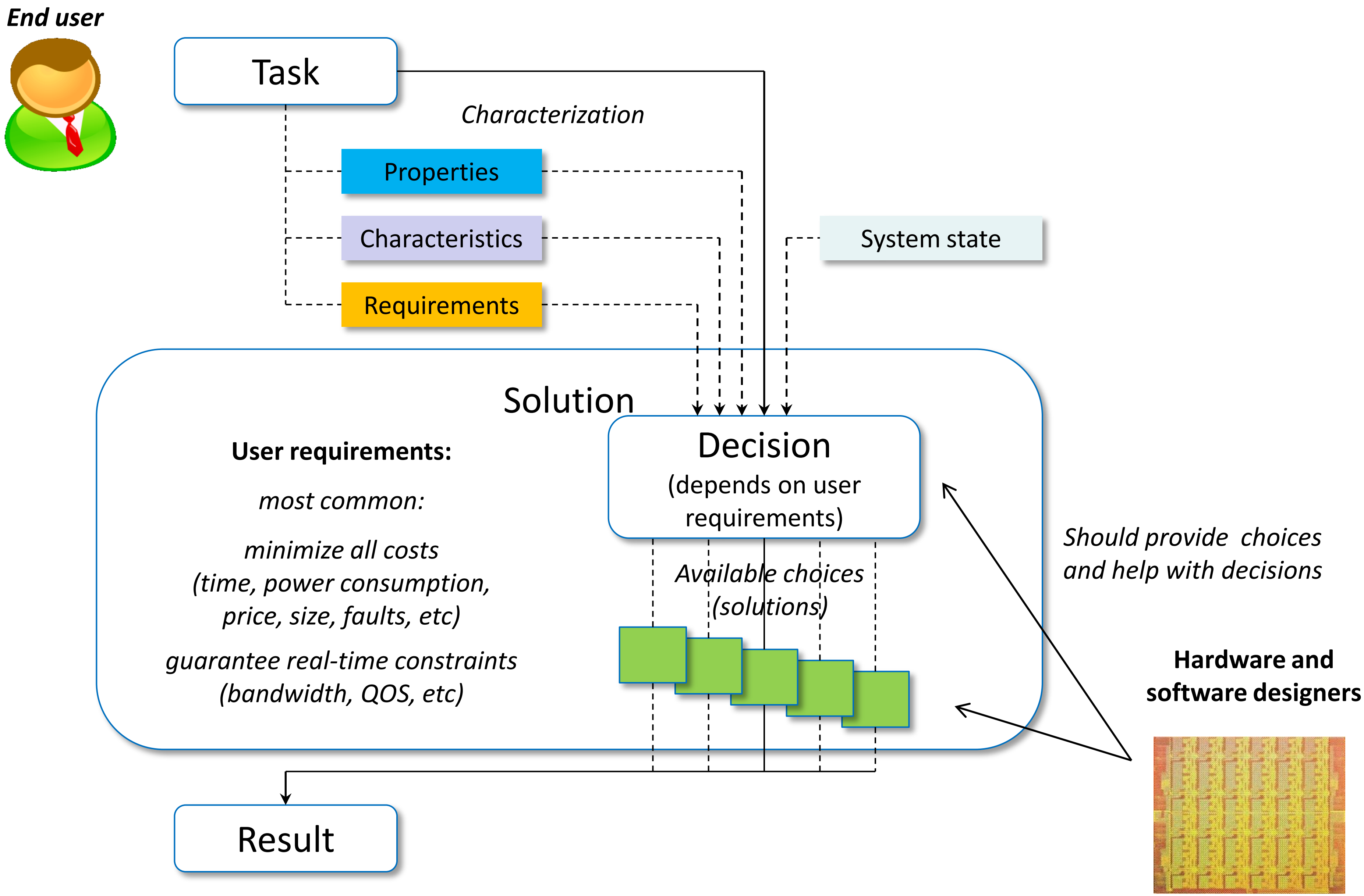
Back to basics



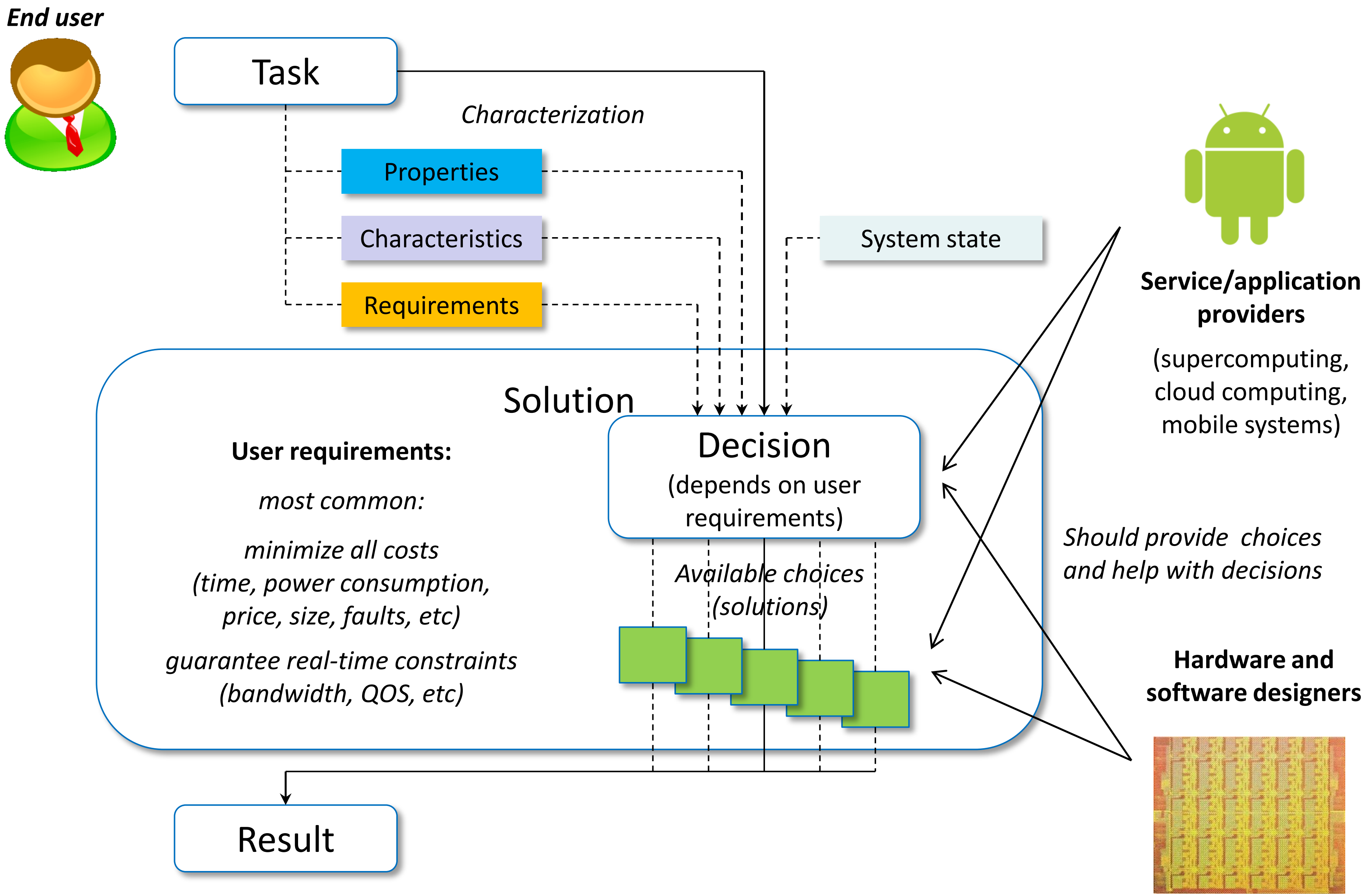
Back to basics



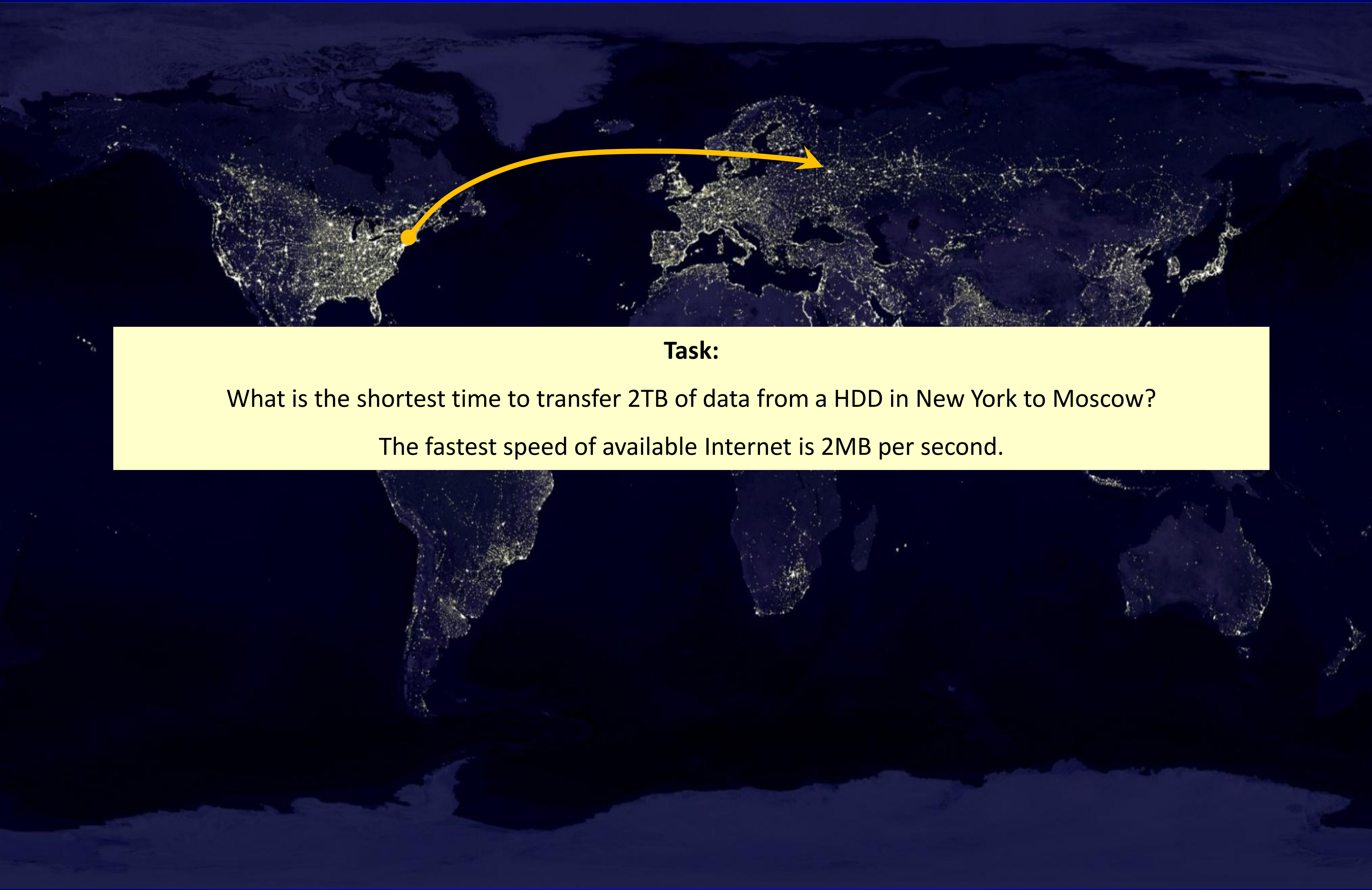
Back to basics



Back to basics



Back to basics

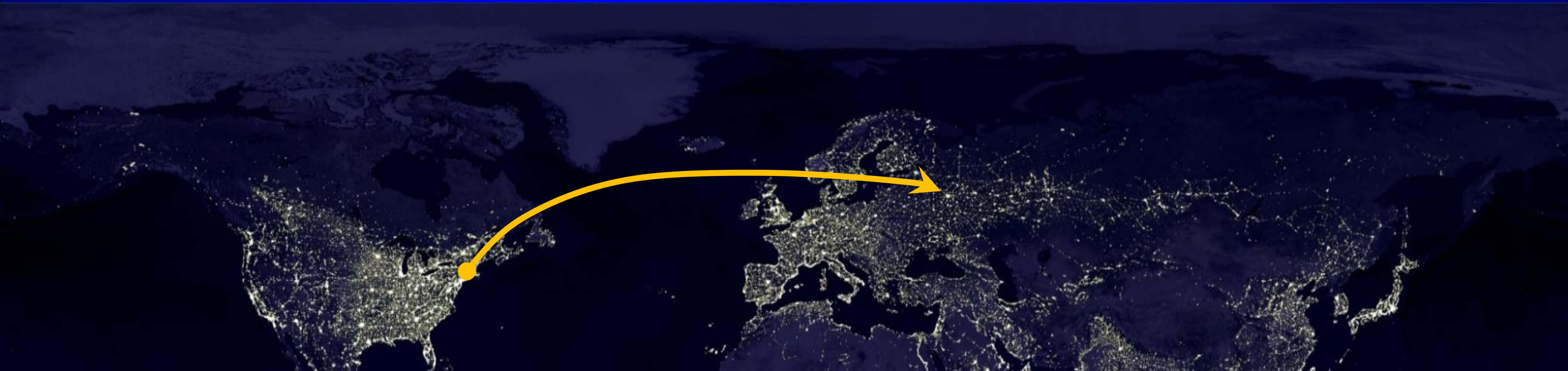


Task:

What is the shortest time to transfer 2TB of data from a HDD in New York to Moscow?

The fastest speed of available Internet is 2MB per second.

Back to basics



Task:

What is the shortest time to transfer 2TB of data from a HDD in New York to Moscow?

The fastest speed of available Internet is 2MB per second.

Possible solution:

~10 hours by plane if cost doesn't matter

Important to identify all properties, requirements, constraints and AVAILABLE SOLUTIONS!

Available solutions: hardware

Companies compete hard to deliver many solutions with various characteristics:

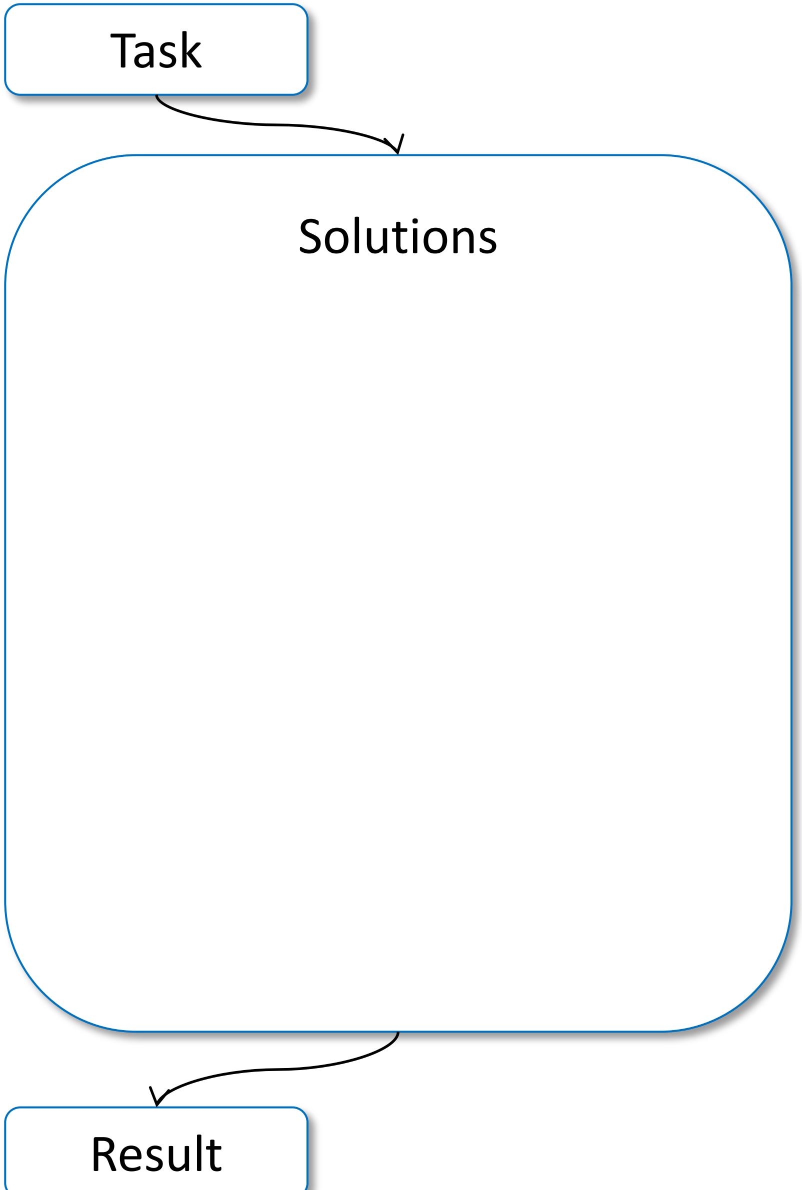
performance, power consumption, size, bandwidth, response time, reliability, cost …

Available solutions: software

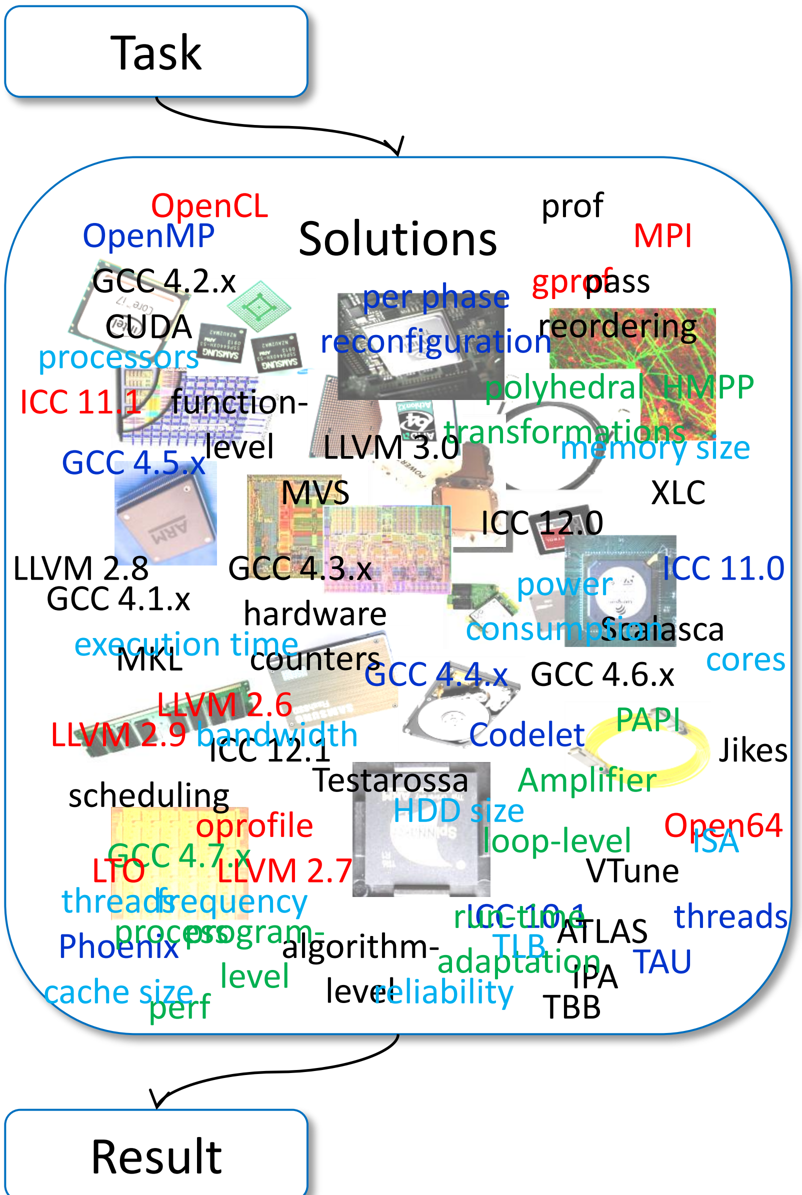
Software developers try to keep pace and produce various algorithms, programming models, languages, analysis tools, compilers, run-time systems, databases, etc.

OpenMP
MPI
HMPP
perf
oprofile
OpenCL
CUDA
gprof
prof
TAU
PAPI
ICC 12.0
function-level
ICC 11.1
ICC 12.1
GCC 4.7.x
GCC 4.3.x
Codelet
GCC 4.1.x
Scalasca
ICC 11.0
LLVM 2.8
pass reordering
GCC 4.4.x
GCC 4.6.x
LLVM 2.6
Jikes
XLC
LLVM 3.0
Testarossa
Amplifier
LLVM 2.9
scheduling
Phoenix
GCC 4.2.x
Open64
threads
LLVM 2.7
VTune
IPA
ICC 10.1
MVS
polyhedral transformations
algorithm-level
MKL
GCC 4.5.x
ATLAS
LTO
process
TBB
hardware counters
program-level
loop-level
run-time adaptation
per phase reconfiguration

Challenges



Challenges



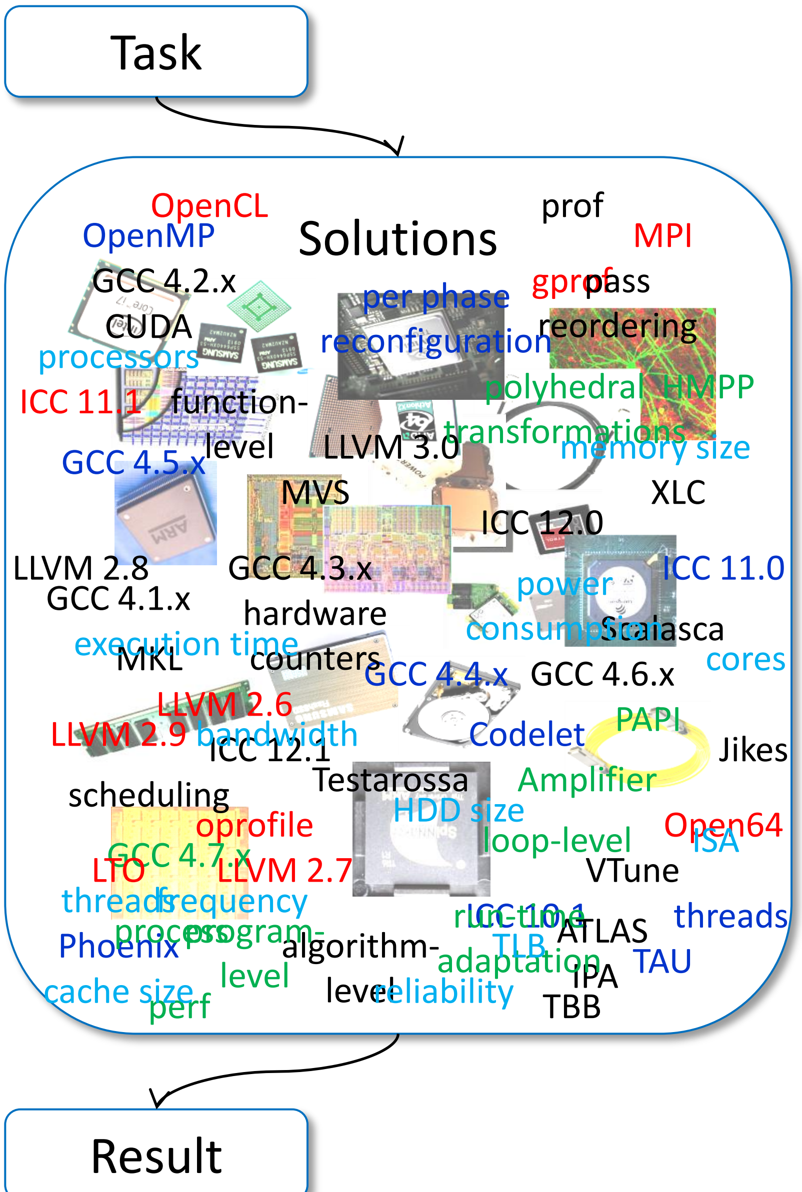
1) Rising complexity of computer systems:
too many design and optimization choices

2) Performance is not anymore the only requirement:
multiple user objectives vs choices
benefit vs optimization time

3) Complex relationship and interactions between ALL software and hardware components.

4) Too many tools with non-unified interfaces changing from version to version:
technological chaos

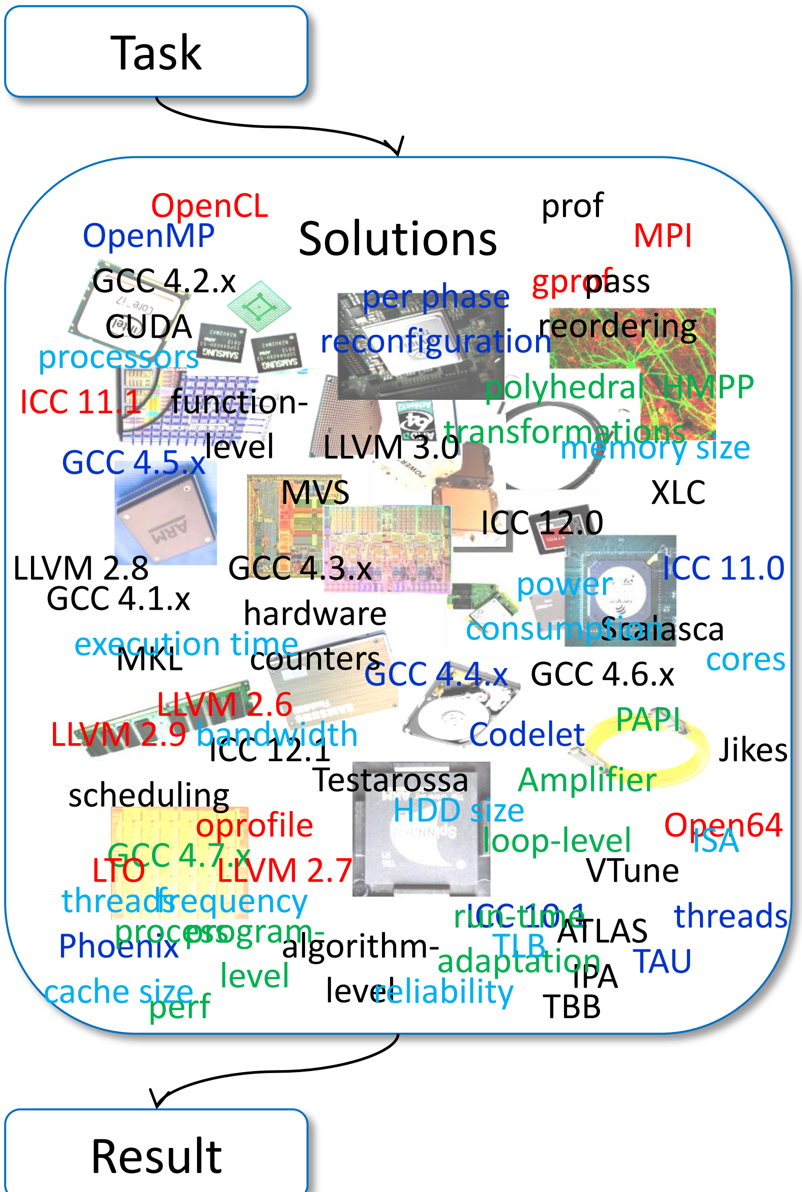
Challenges



Result:

- finding the right solution is extremely challenging
- everyone is lost in choices
- dramatic increase in development time
- low ROI
- underperforming systems
- waste of energy
- ad-hoc, repetitive and error-prone manual tuning
- slowing innovation in science and technology

Challenges



Result:

- finding the right solution is extremely challenging
- everyone is lost in choices
- dramatic increase in development time
- low ROI
- underperforming systems
- waste of energy
- ad-hoc, repetitive and error-prone manual tuning
- slowing innovation in science and technology

Understanding and modeling of the overall relationship between end-user algorithms, applications, compiler optimizations, hardware designs, data sets and run-time behavior became simply infeasible!

Attempts to solve these problems: auto-tuning

Task

Treat computer system as a black box

Result

Algorithm

Application

Compilers and auxiliary tools

Binary and libraries

Data set

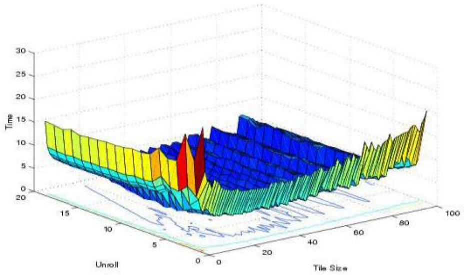
State of the system

Run-time environment

Architecture

Use auto-tuning:

Explore multiple choices empirically: learn behavior of computer systems across executions



Covered all components in the last 2 decades and showed high potential but …

Attempts to solve these problems: auto-tuning

Auto-tuning shows high potential for nearly 2 decades but still far from the mainstream in production environments.

Why?

- Optimization spaces are large and non-linear with many local minima
- Exploration is slow and ad-hoc (random, genetic, some heuristics)
- Only a few benchmarks are considered
- Often the same (one) dataset is used
- Only part of the system is taken into account
(rarely reflect behavior of the whole system)
- No knowledge sharing

Attempts to solve these problems: machine learning

Task

Treat computer system as a black box

Result

Algorithm

Application

Compilers and auxiliary tools

Binary and libraries

Data set

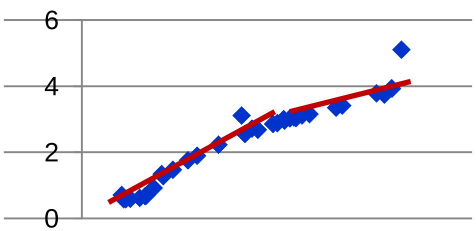
State of the system

Run-time environment

Architecture

Use machine learning to speed up exploration

Apply predictive modeling to suggest profitable solutions based on properties of a task and a system



Covered all components in the last decade and showed high potential but …

Attempts to solve these problems: machine learning

Machine learning (classification, predictive modeling) shows high potential during past decade but still far from the mainstream.

Why?

- Selection of machine learning models and right properties is non-trivial: ad-hoc in most of the cases

- Limited training sets

- Only part of the system is taken into account (rarely reflect behavior of the whole system)

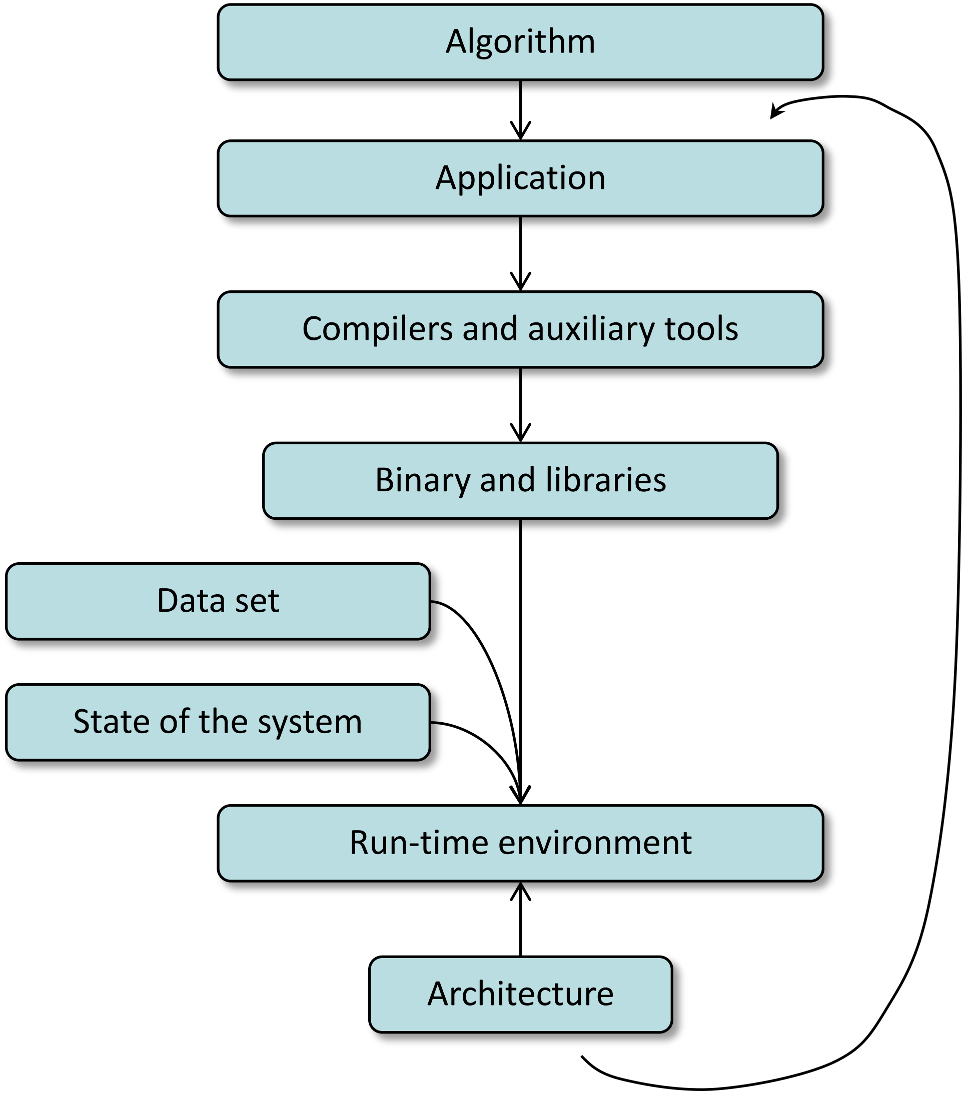
- No knowledge sharing

Attempts to solve these problems: co-design

Task

Treat computer system as a black box

Result



Co-design:

Explore choices and behavior of the whole system.

Showed high potential in the last years but …

Attempts to solve these problems: machine learning

Co-design is currently a buzz word and a hot research topic but still far from the mainstream.

Why?

- Even more choices to explore and analyze
- Limited training sets
- Still no knowledge sharing

Attempts to solve these problems: knowledge sharing

Main idea:

Why not to leverage the experience
and computational resources of multiple users?

Attempts to solve these problems: optimization repositories

Many researchers develop some repositories for experiments.

The lifespan of such repositories: end of the PhD or project.

I don't know repositories that were made public.

Major reasons:

Main focus in academia is to publish as many papers as possible.

Reproducibility and statistical meaningfulness of results is often not even considered! In fact, it is often impossible.

Software development is considered as overhead or even waste of time.

Attempts to solve these problems: optimization repositories

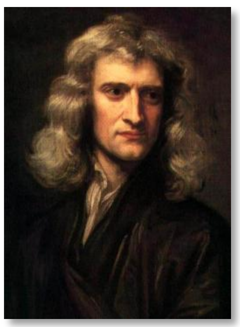
Simply too time consuming and costly to build, support and extend particularly with ever changing tools, interfaces, benchmarks, data sets, properties, models, etc.

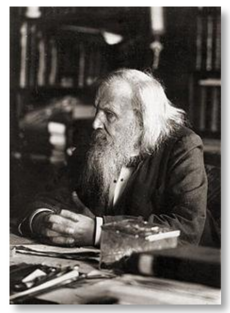
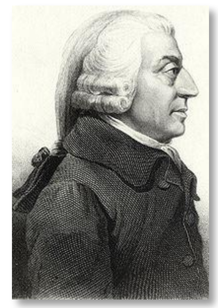
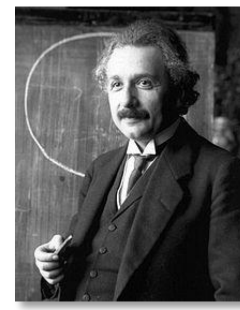
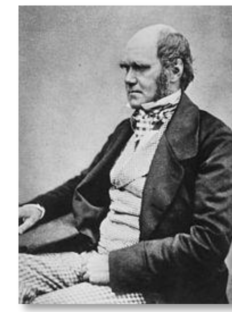
Only big companies or projects can afford to build and support their own big repositories but they are either not public (Google, Intel, IBM, ARM) or used as a simple storage of information (SciDAC, SPEC).

Furthermore, public data and tools may cause competition.

Similar problems in other sciences

We can we learn from existing sciences that deal with complex systems:
physics, mathematics, chemistry, biology, computer science, etc?

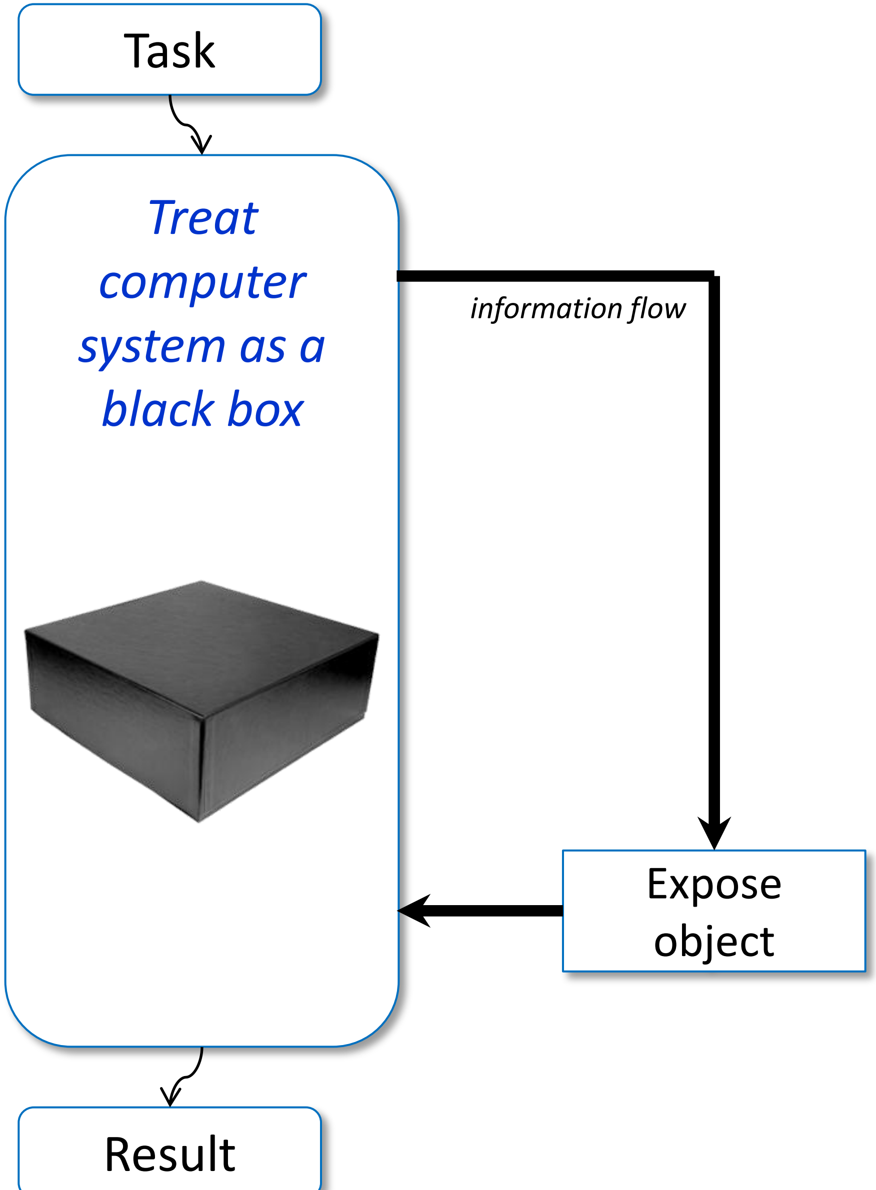
Major breakthrough came from
collaborative discovery, systematization, sharing and reuse of knowledge!

Collective co-design of computer systems

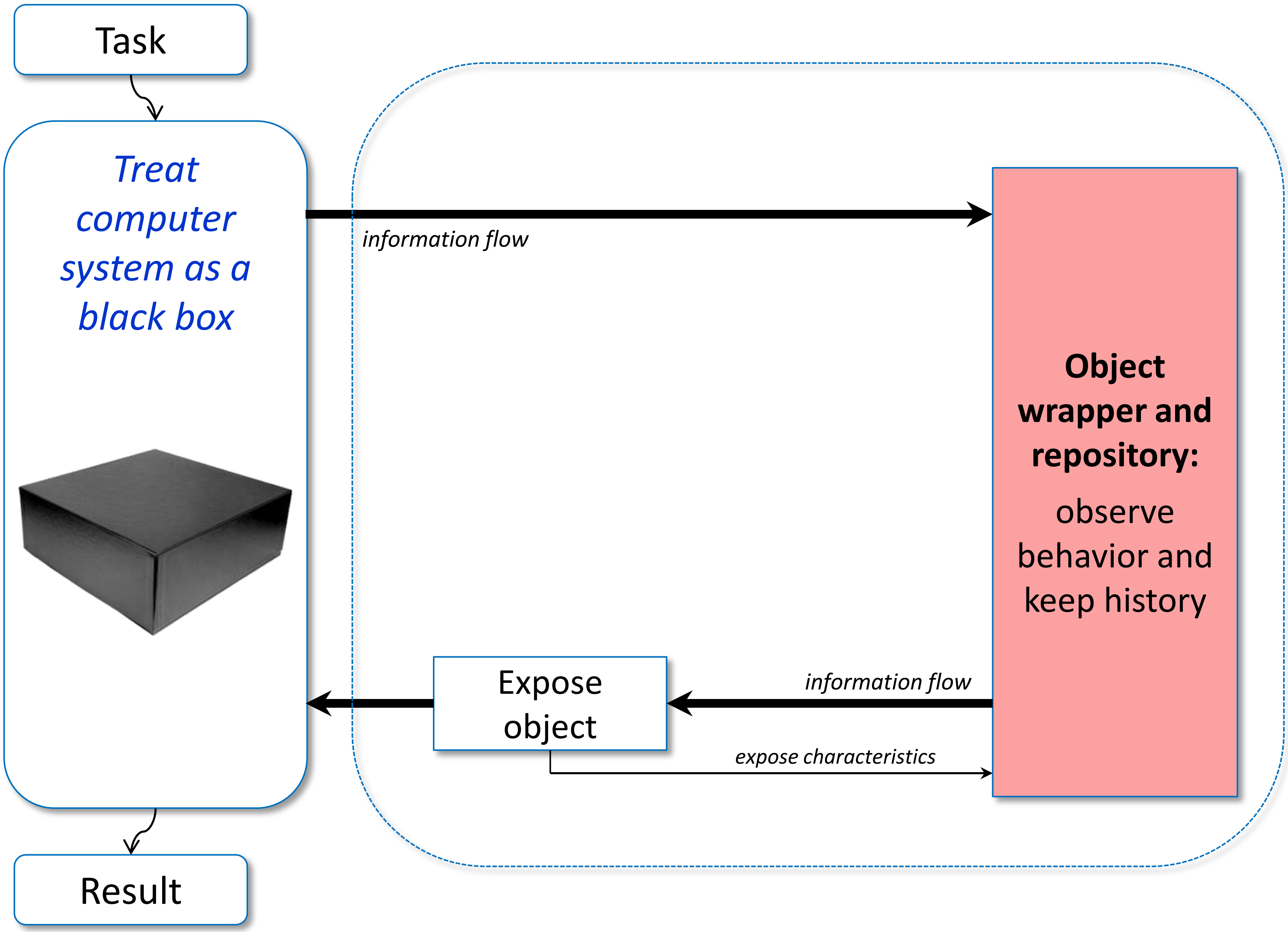
We have proposed and started developing collective methodology and infrastructure (cTuning) where:

- repository, auto-tuning and machine learning is an integral part of co-design
- repository is dynamically evolving and contains all encountered benchmarks, data sets, tools, codelets, optimized binaries and libraries, choices, properties, characteristics, predictive models, decision trees
- repository and infrastructure is distributed among many users and can automatically exchange information about
  - unexplored choices
  - optimization areas with high variability
  - optimal predictive models
  - abnormal behavior to focus further exploration and validate or improve classification and models

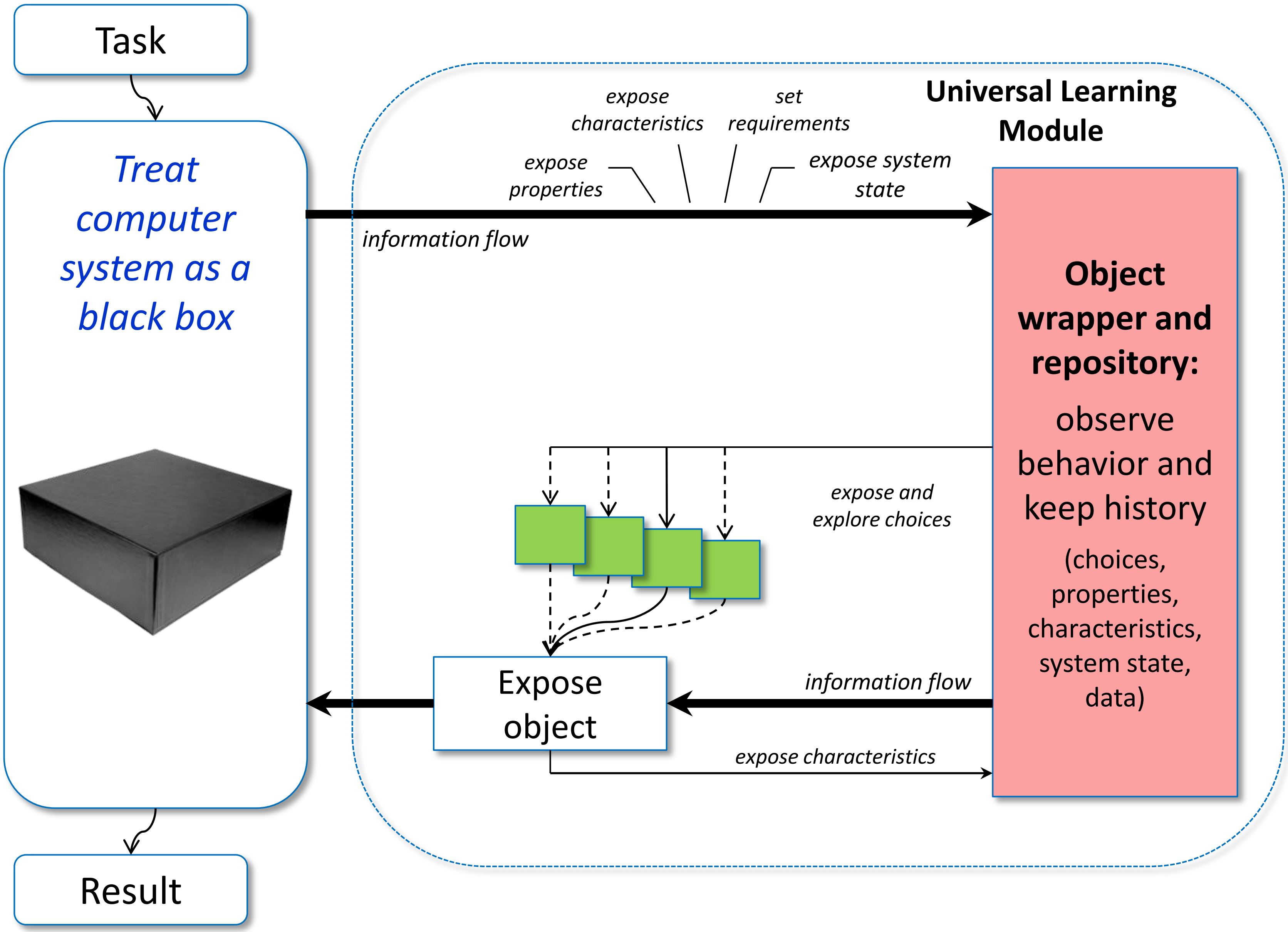
Knowledge discovery and preservation: a physicist's approach



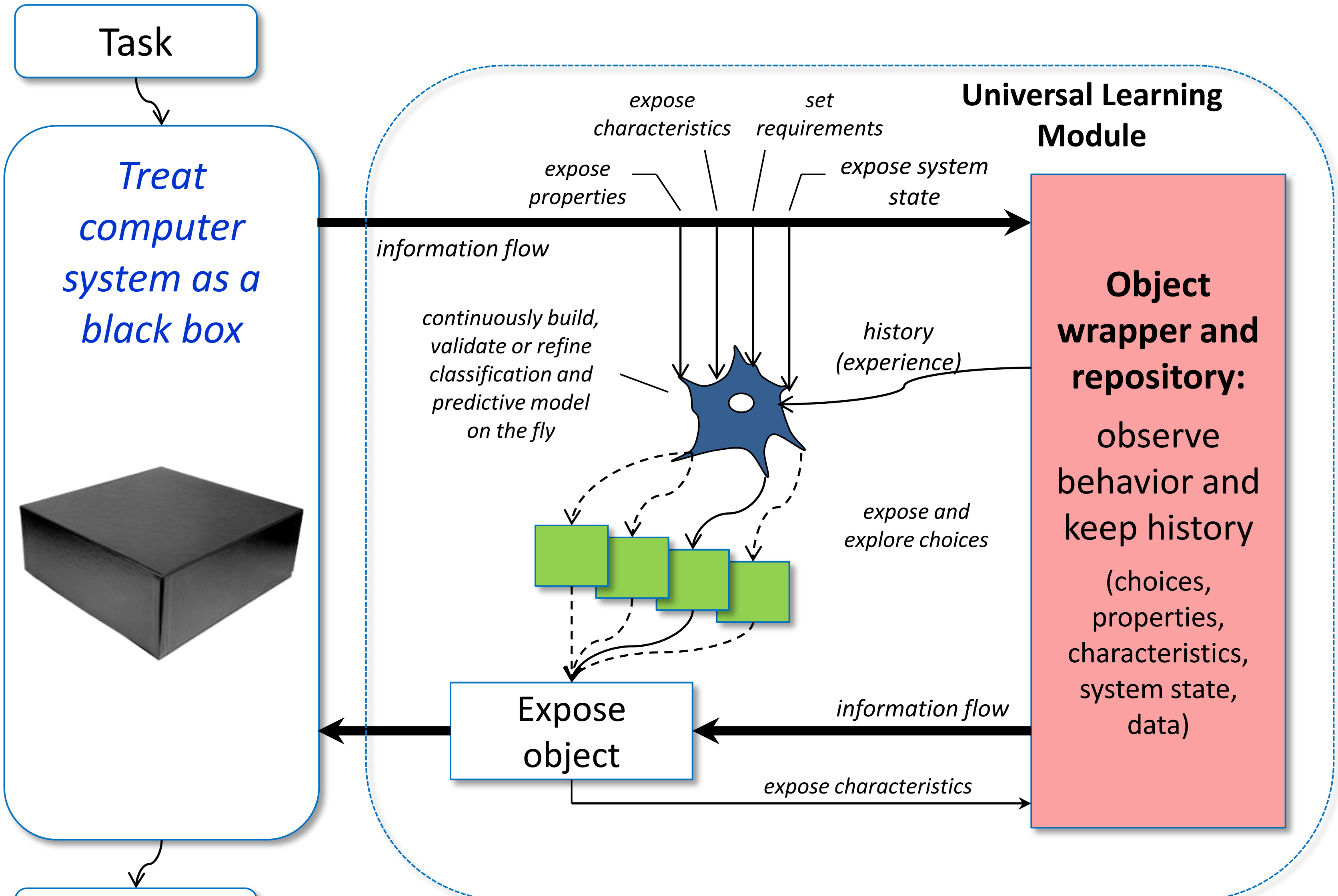
Observe system



Gradually expose properties, characteristics, choices



Classify, build models, predict behavior



Result

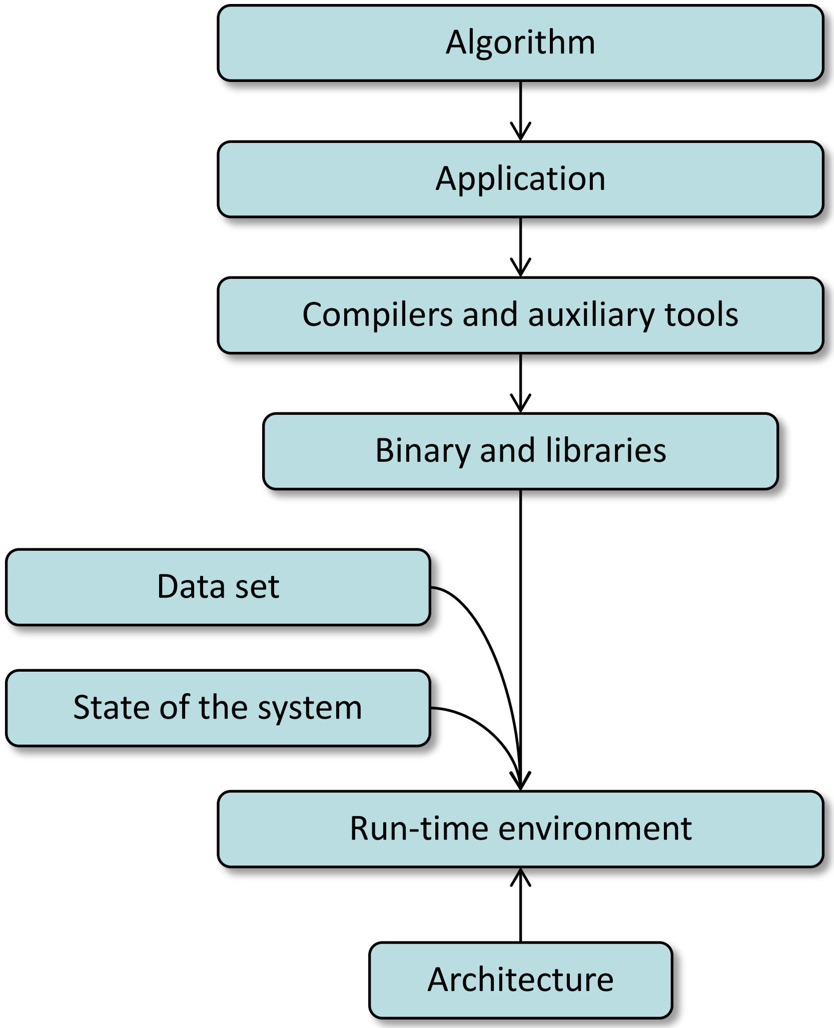
Evolutionary approach, not revolutionary, i.e. do not rebuild existing SW/HW stack from scratch but wrap up existing tools and techniques, and gradually clean up the mess!

Gradual decomposition, parameterization, observation and exploration of a system

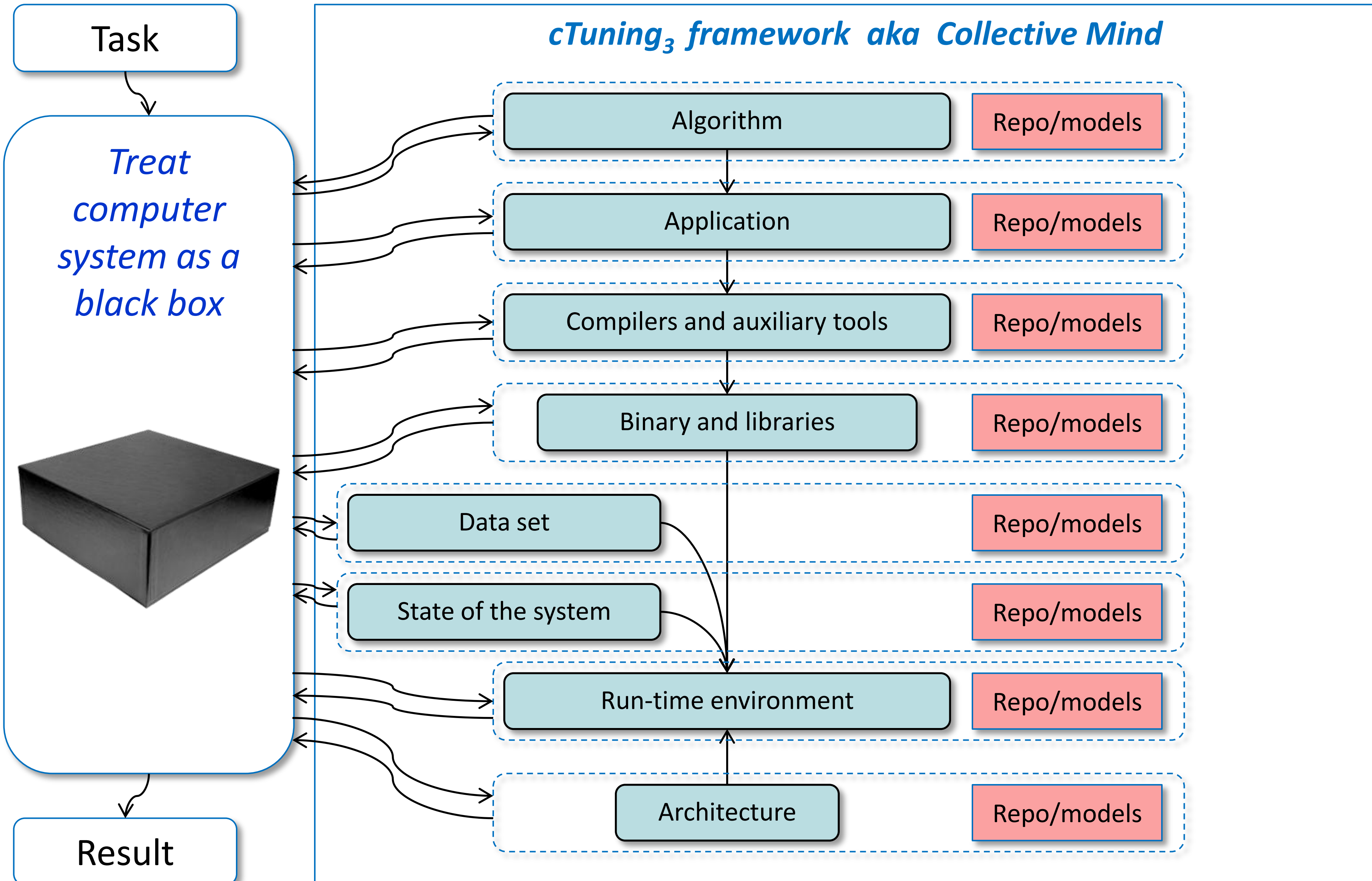
Task

Treat computer system as a black box

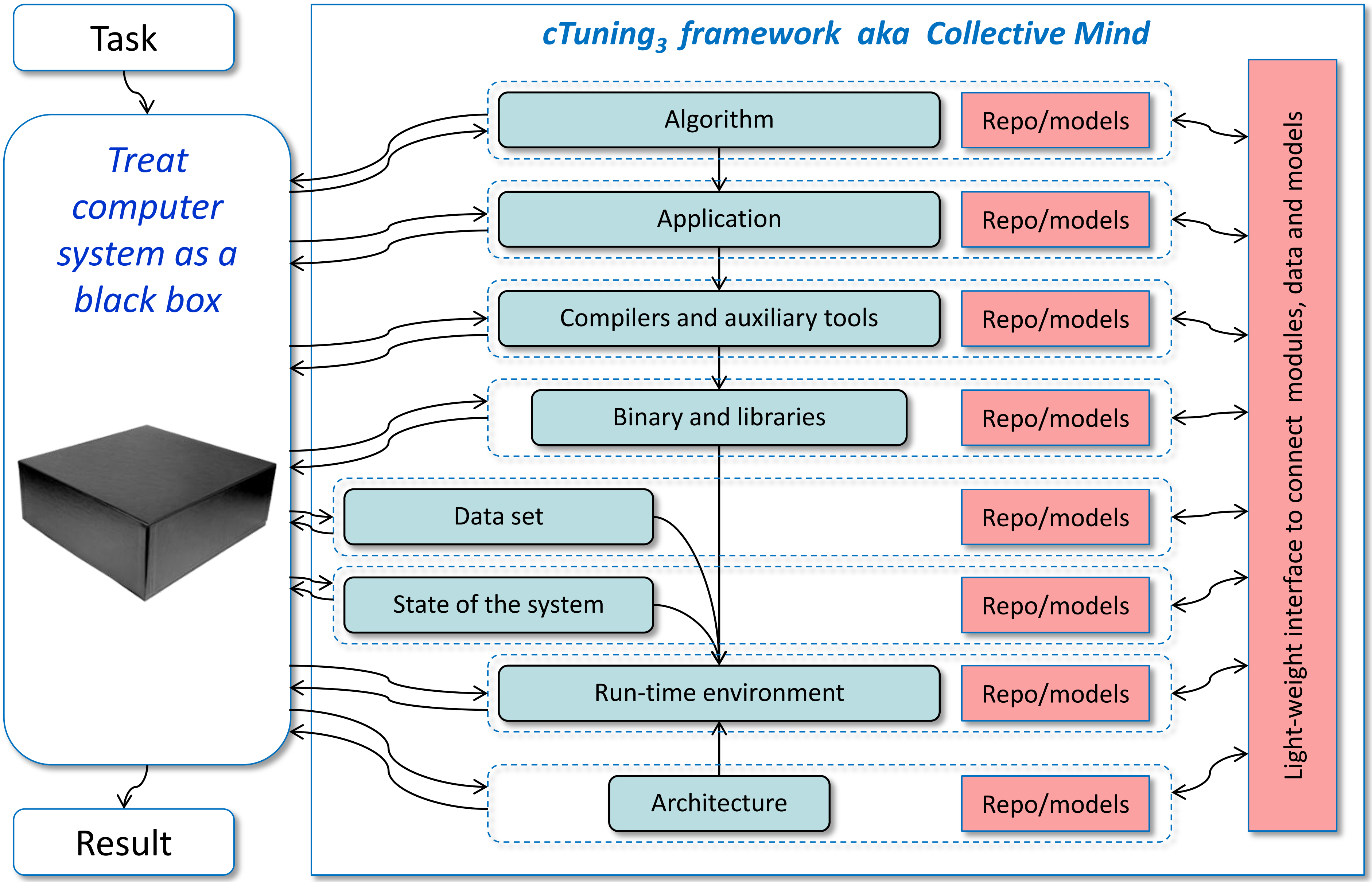
Result



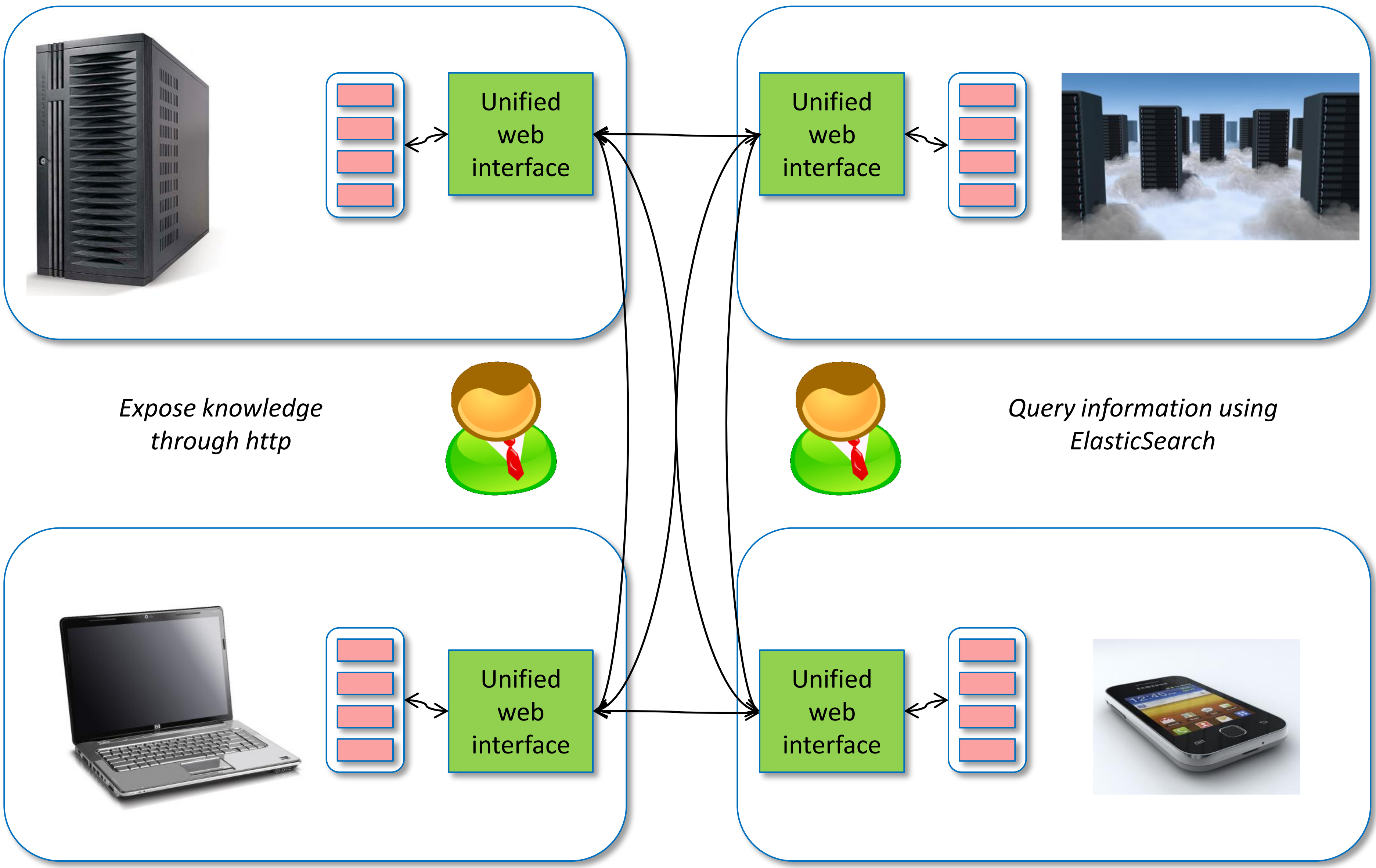
Gradual decomposition, parameterization, observation and exploration of a system



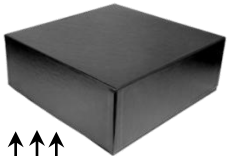
Gradual decomposition, parameterization, observation and exploration of a system



Unified and continuous information exchange



Example of characterizing/explaining behavior of computer systems



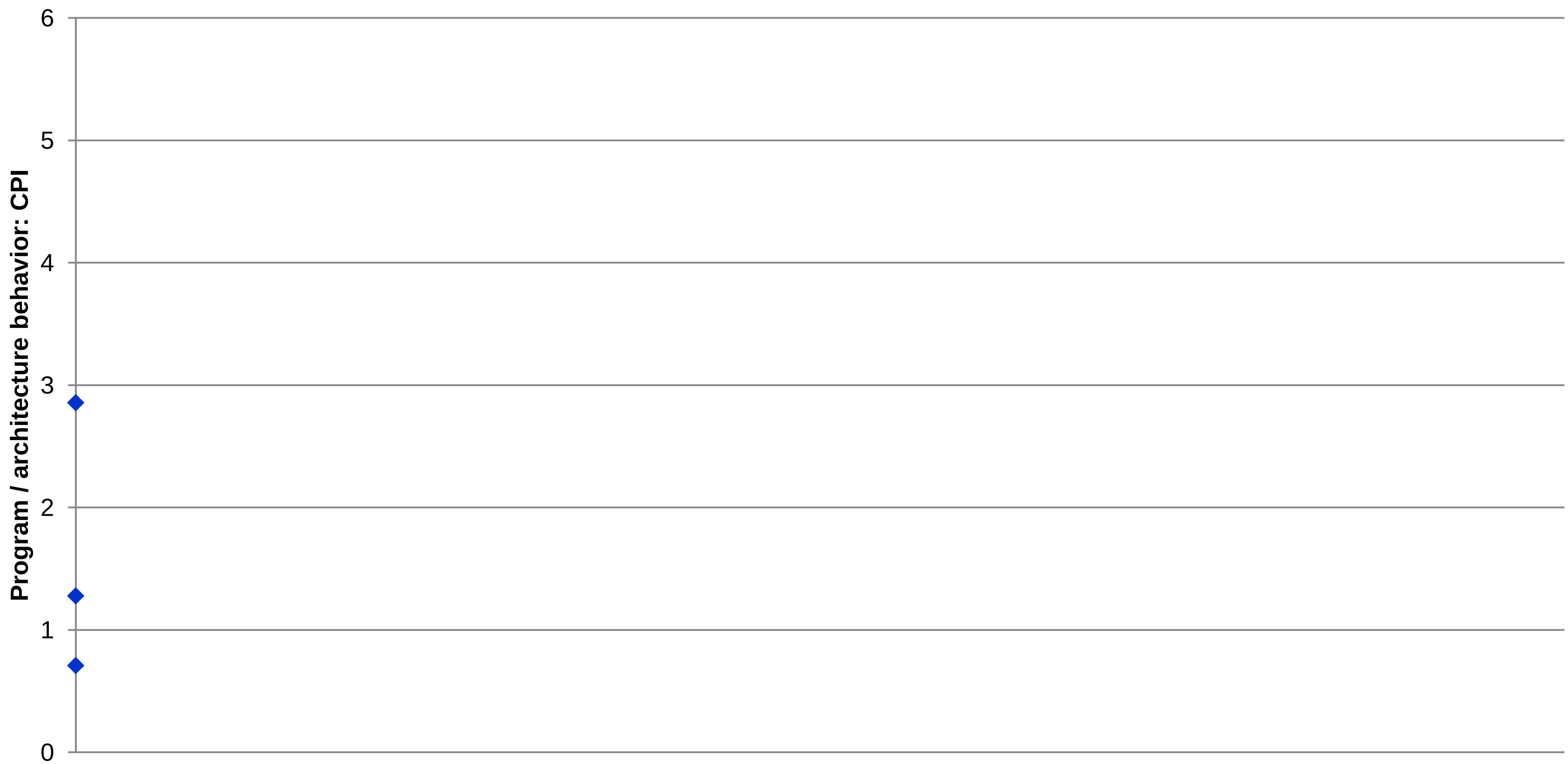
	Gradually expose some characteristics	Gradually expose some properties/choices
Compile Program	time …	compiler flags; pragmas …
Run code → Run-time environment	time; CPI, power consumption …	pinning/scheduling …
System	cost;	architecture; frequency; cache size…
Data set	size; values; description …	precision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

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

Example of characterizing/explaining behavior of computer systems

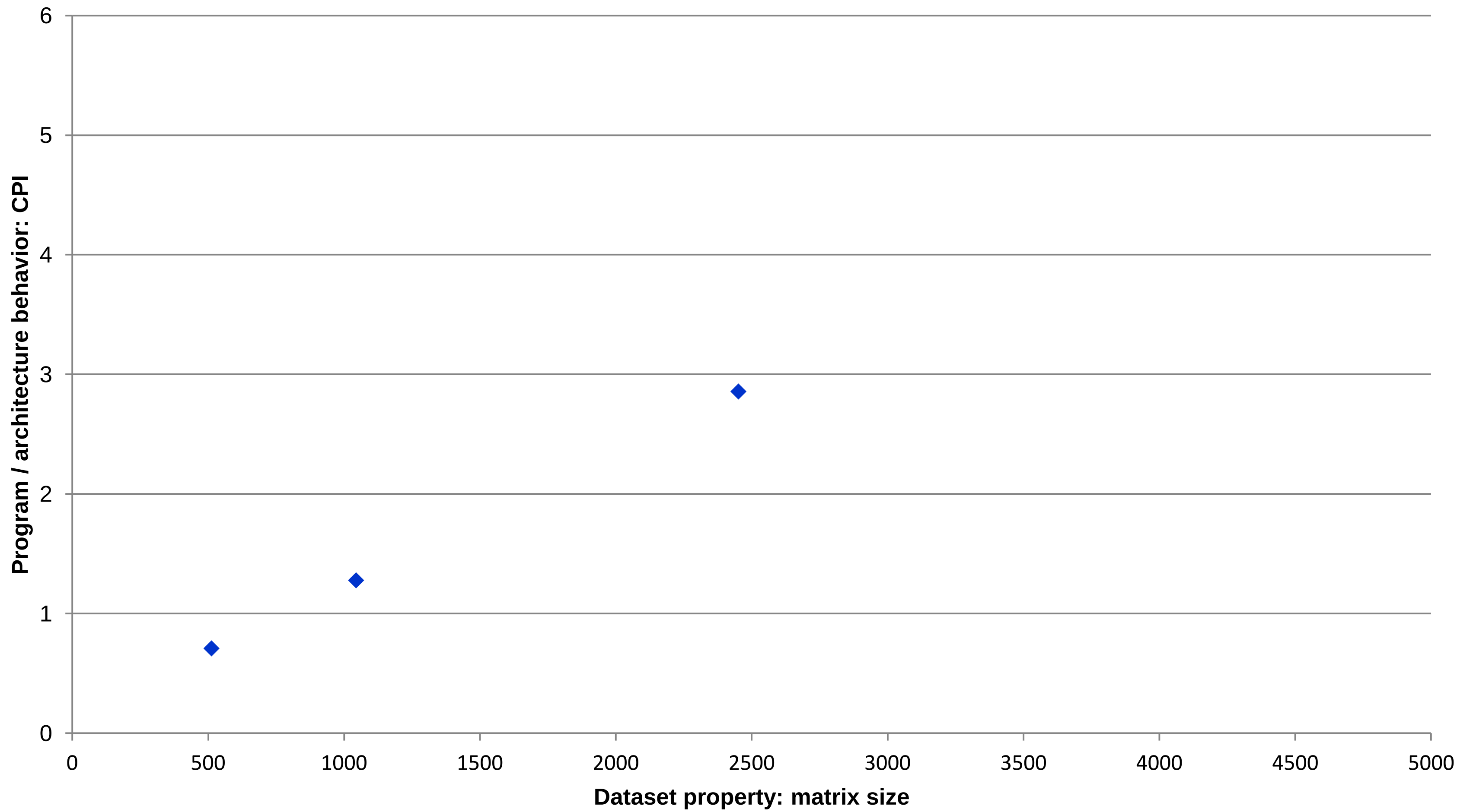
How we can explain the following observations for some piece of code ("codelet object")?

(LU-decomposition codelet, Intel Nehalem)



Example of characterizing/explaining behavior of computer systems

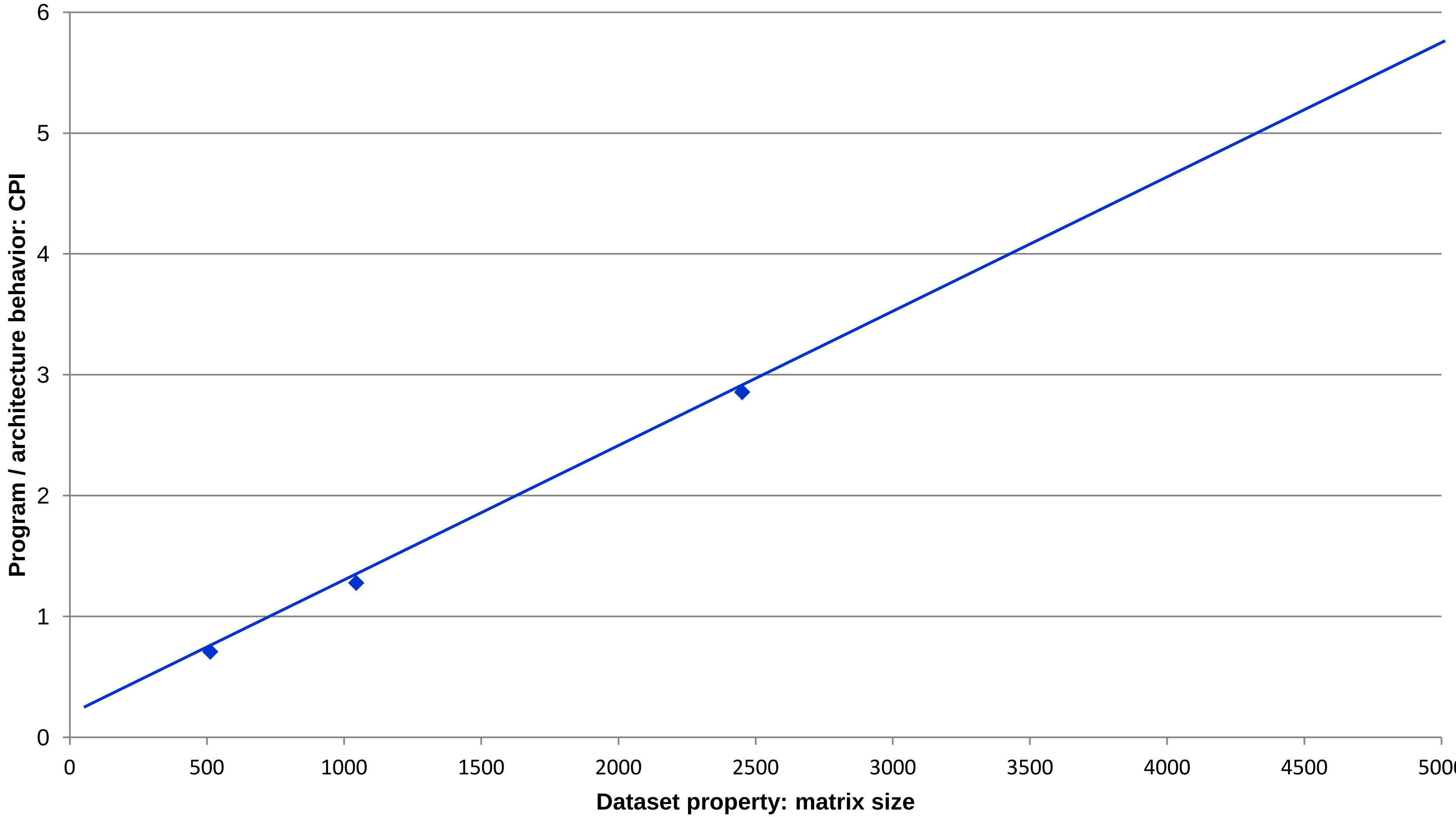
Add 1 property: matrix size



Example of characterizing/explaining behavior of computer systems

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

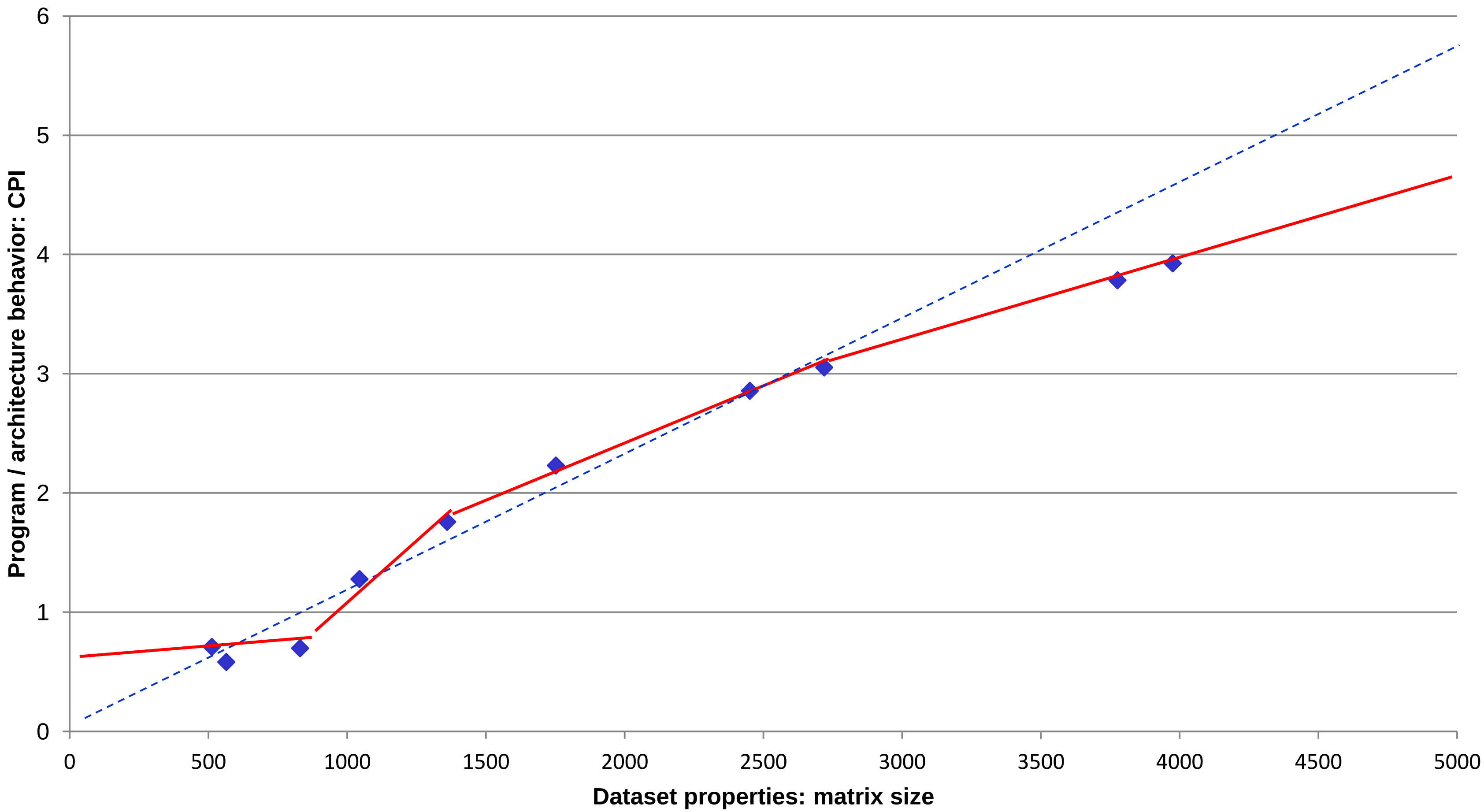
Start from simple models: linear regression (detect coarse grain effects)



Example of characterizing/explaining behavior of computer systems

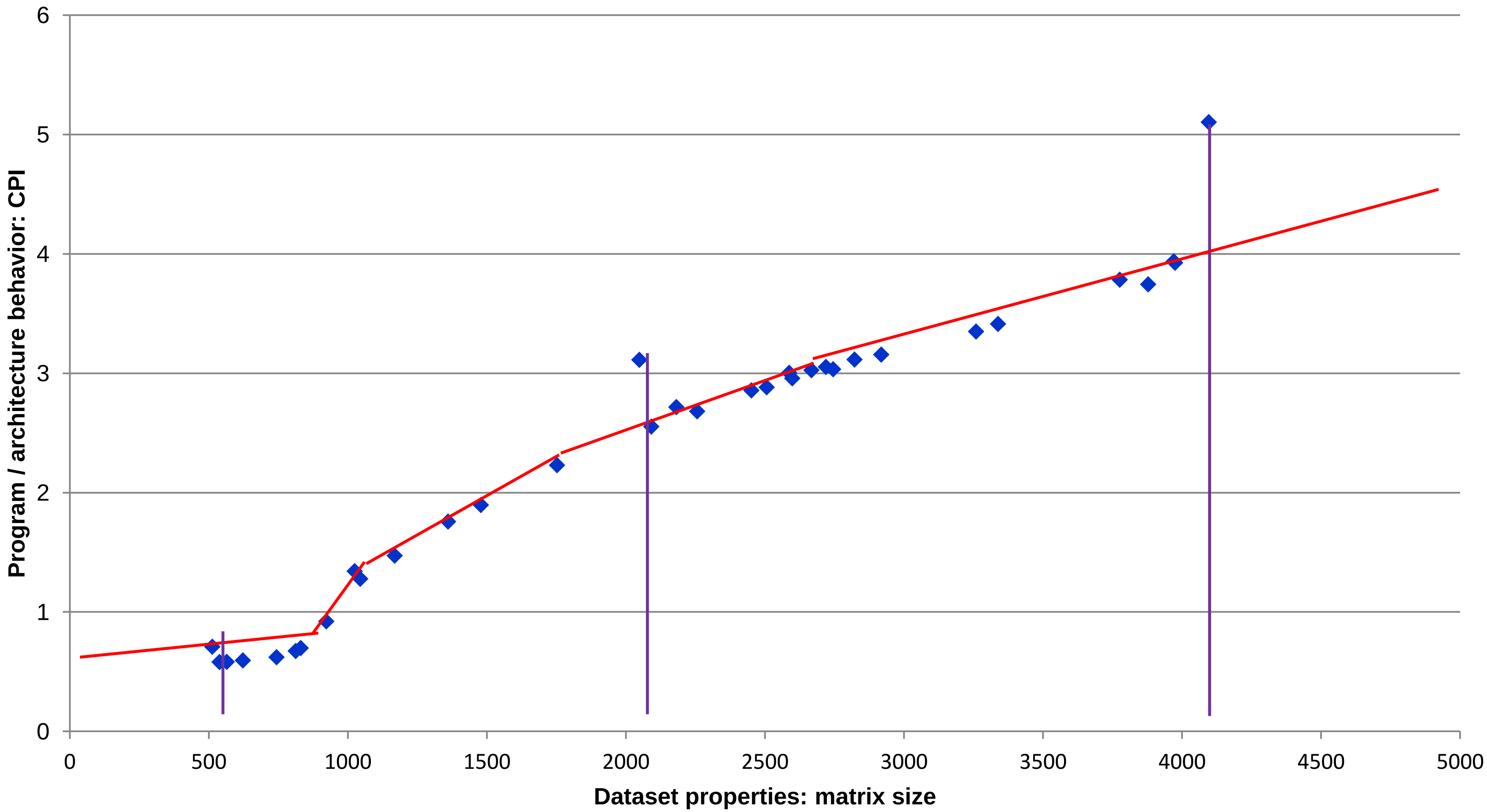
If more observations, validate model and detect discrepancies!

Continuously retrain models to fit new data!



Example of characterizing/explaining behavior of computer systems

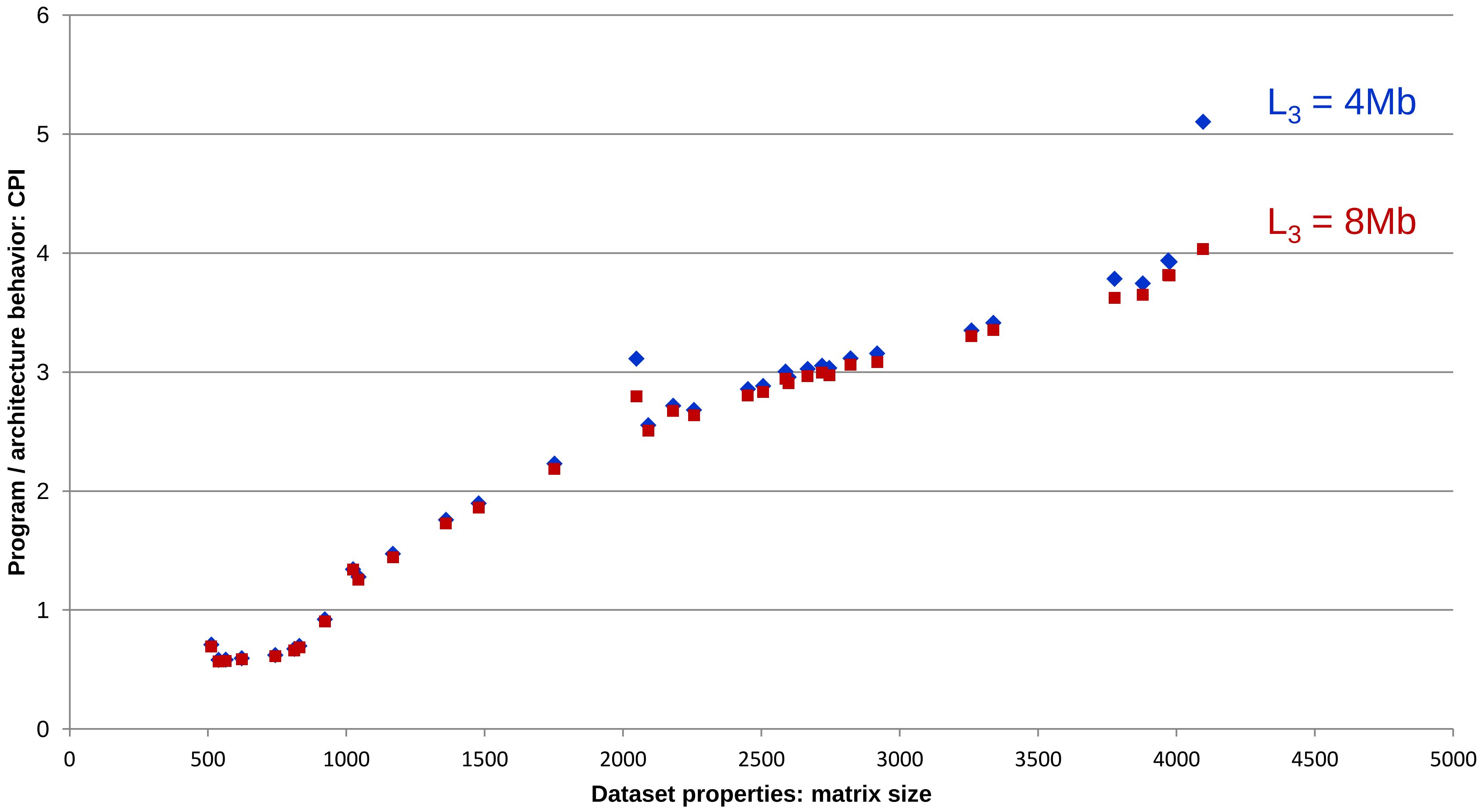
Gradually increase model complexity if needed (hierarchical modeling).
For example, detect fine-grain effects (singularities) and characterize them.



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.

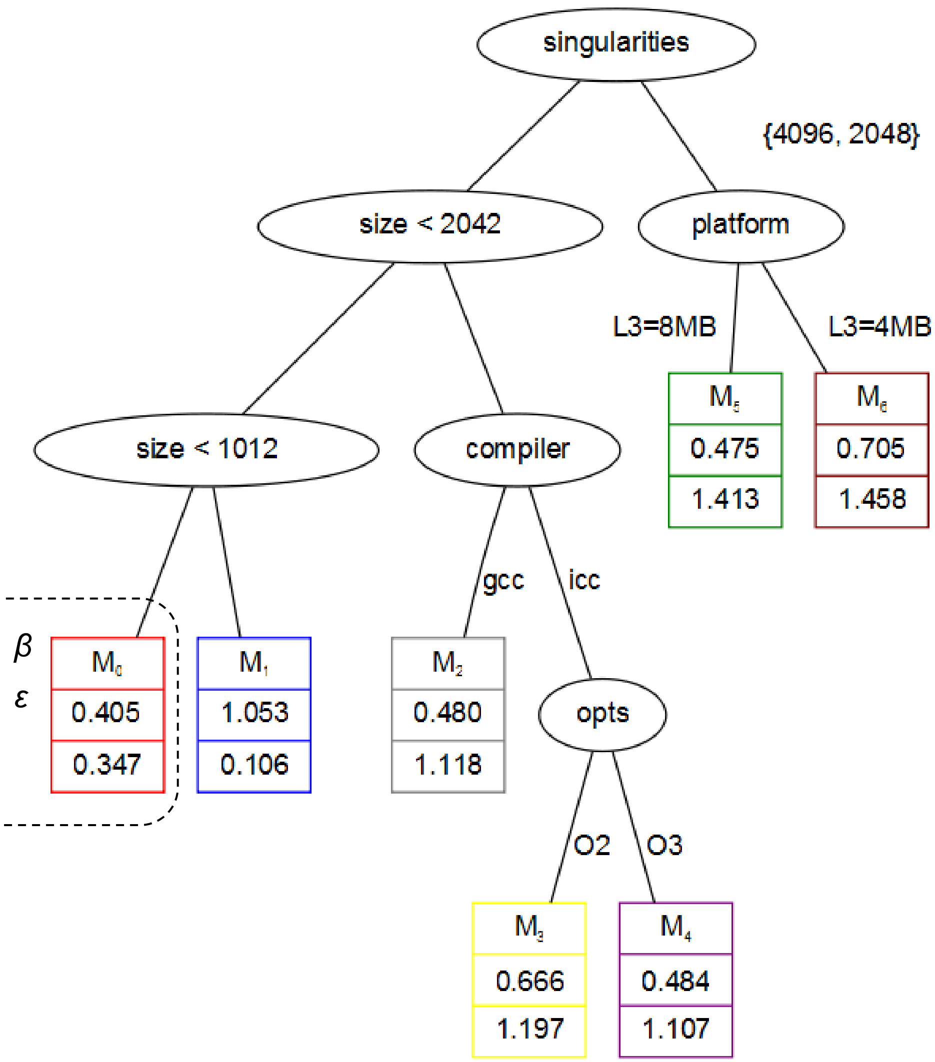


Example of characterizing/explaining behavior of computer systems

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

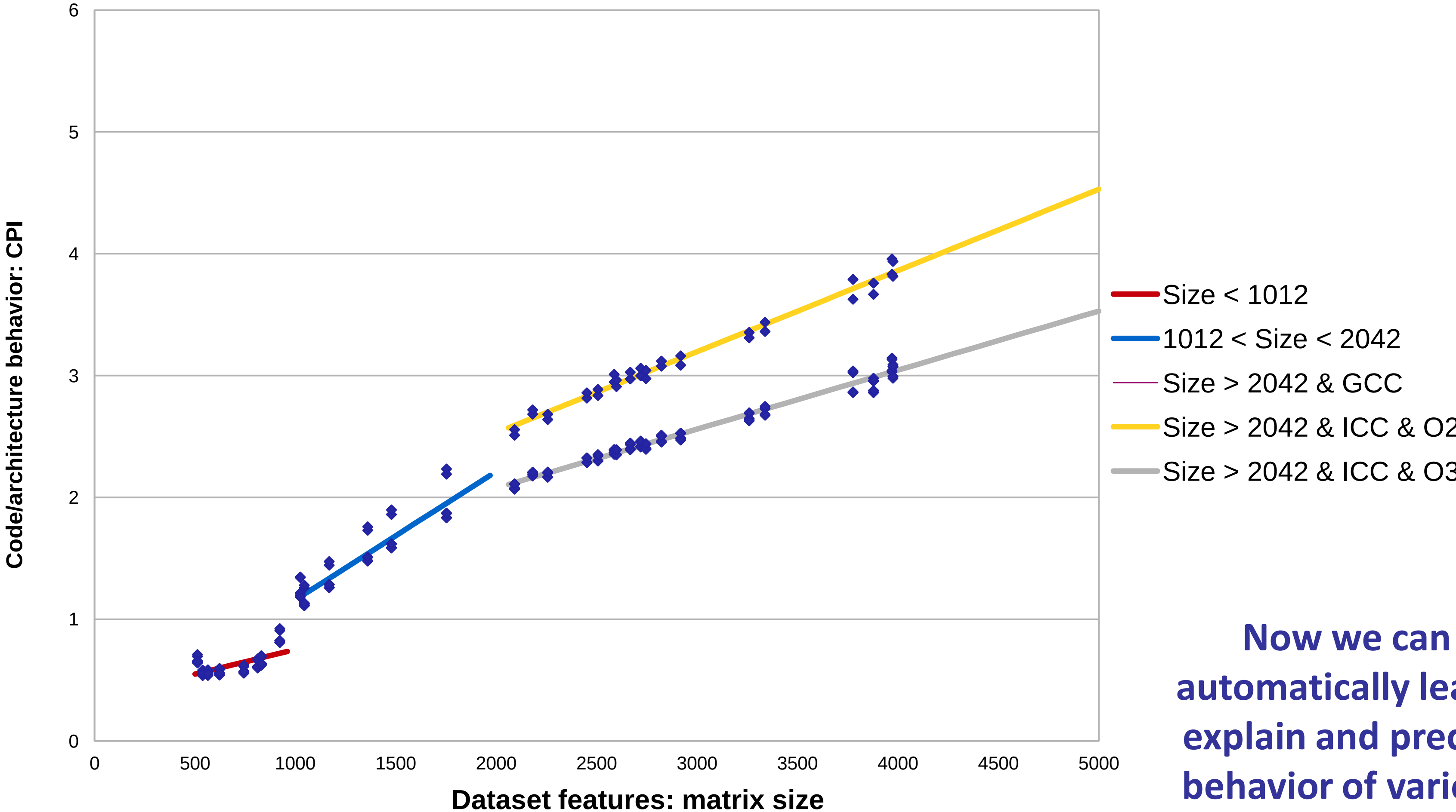


cM predictive model module

$\mathrm{CPI} = \varepsilon + 1000 \times \beta \times \text{data size}$

Predictive modeling

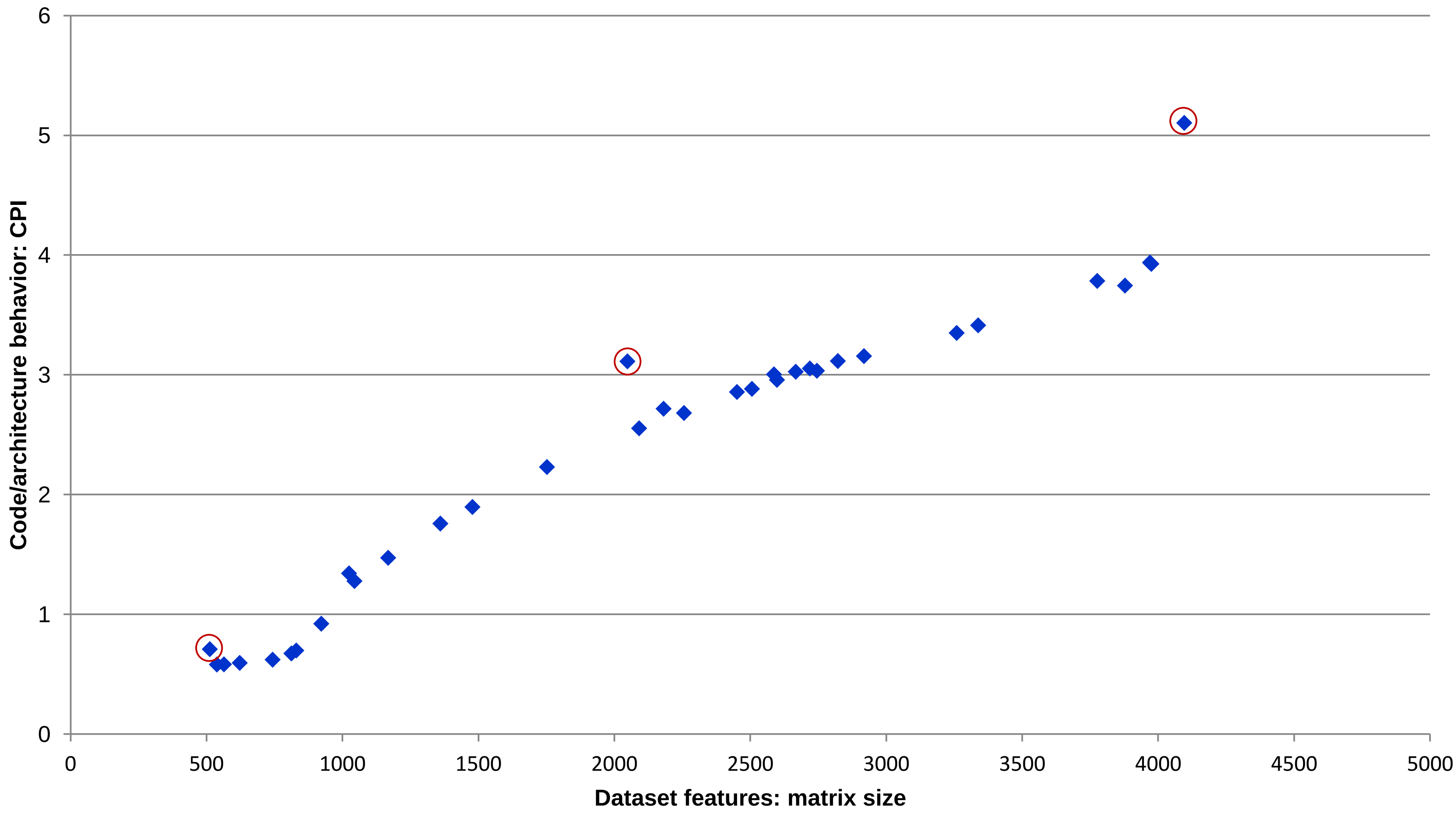
Optimize decision tree (many different algorithms)
Balance precision vs cost of modeling = ROI (coarse-grain vs fine-grain effects)
Compact data on-line before sharing with other users!



Now we can automatically learn, explain and predict behavior of various systems!

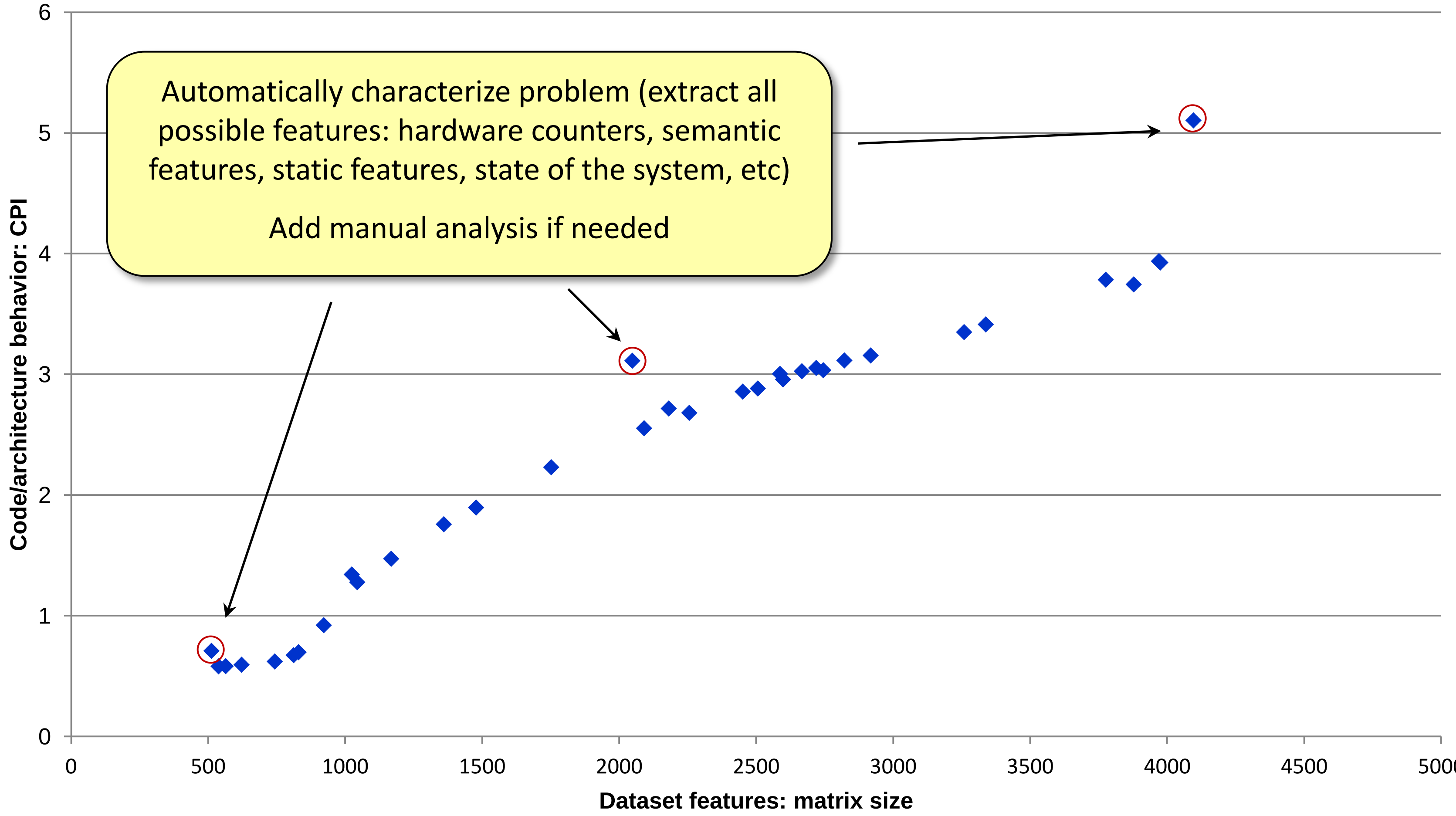
Extensible and collaborative advice system

Collaboratively and continuously add expert advices or automatic optimizations.



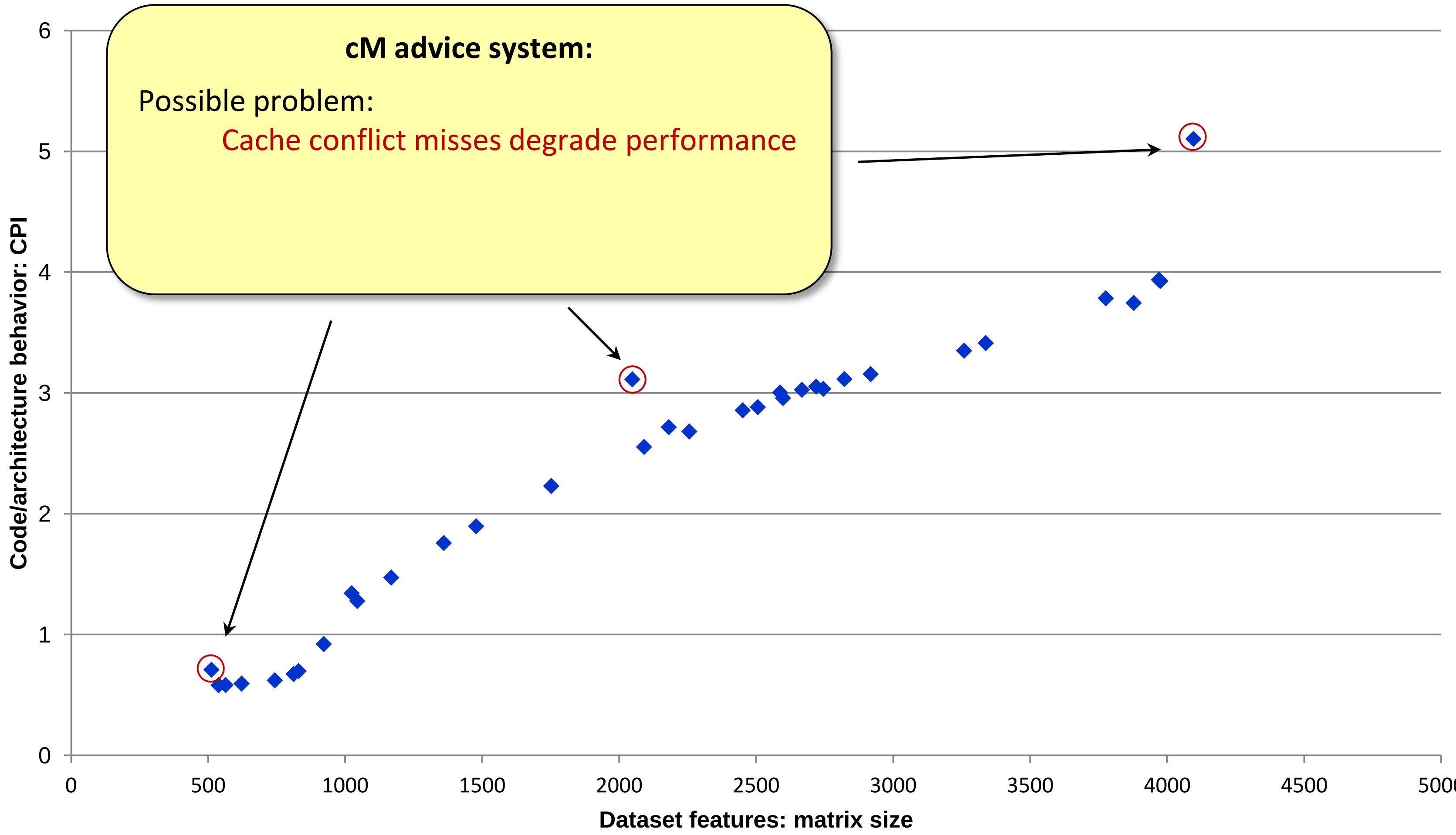
Extensible and collaborative advice system

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 advice system

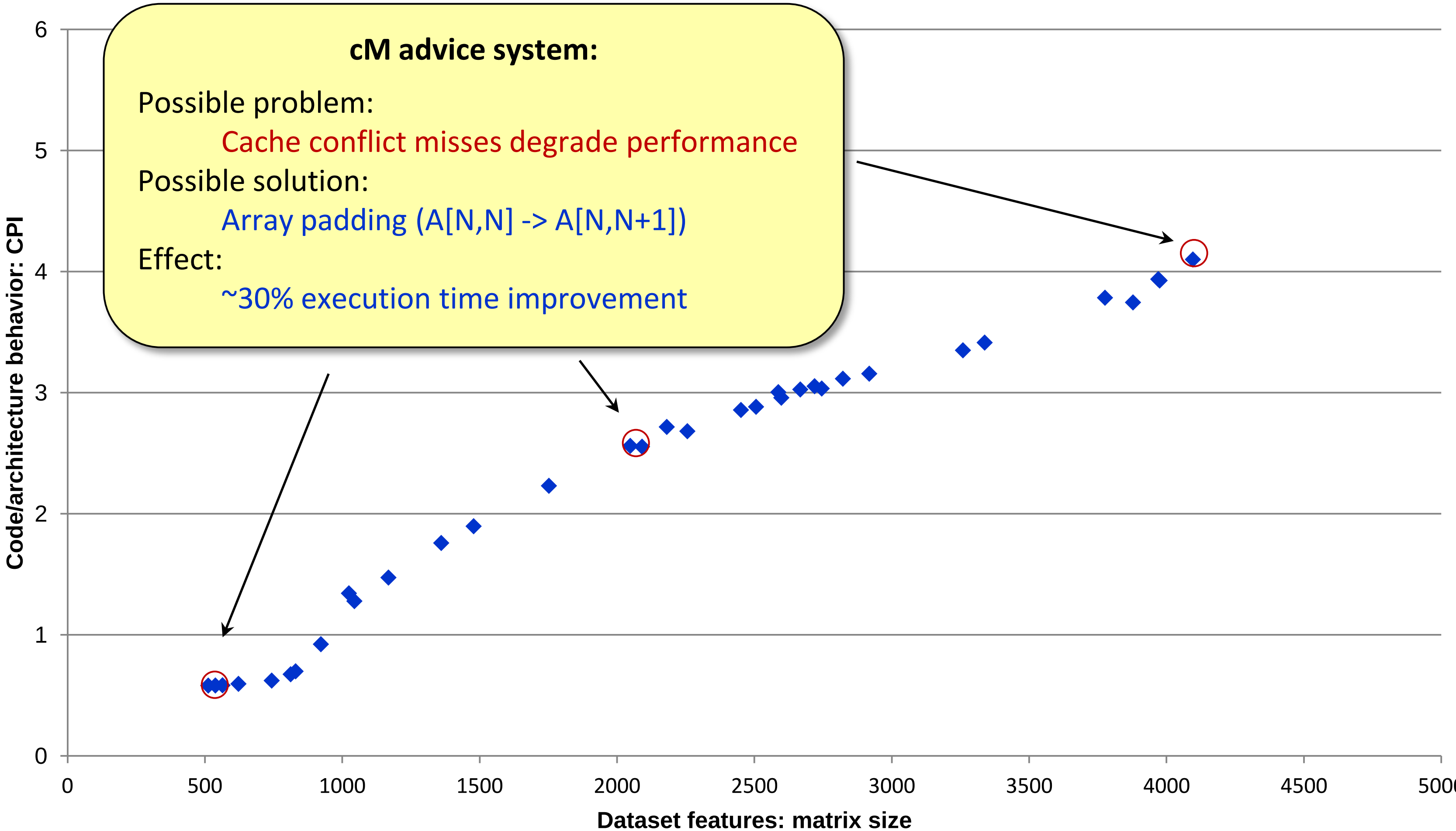
Collaboratively and continuously add expert advices or automatic optimizations.

cM advice system:

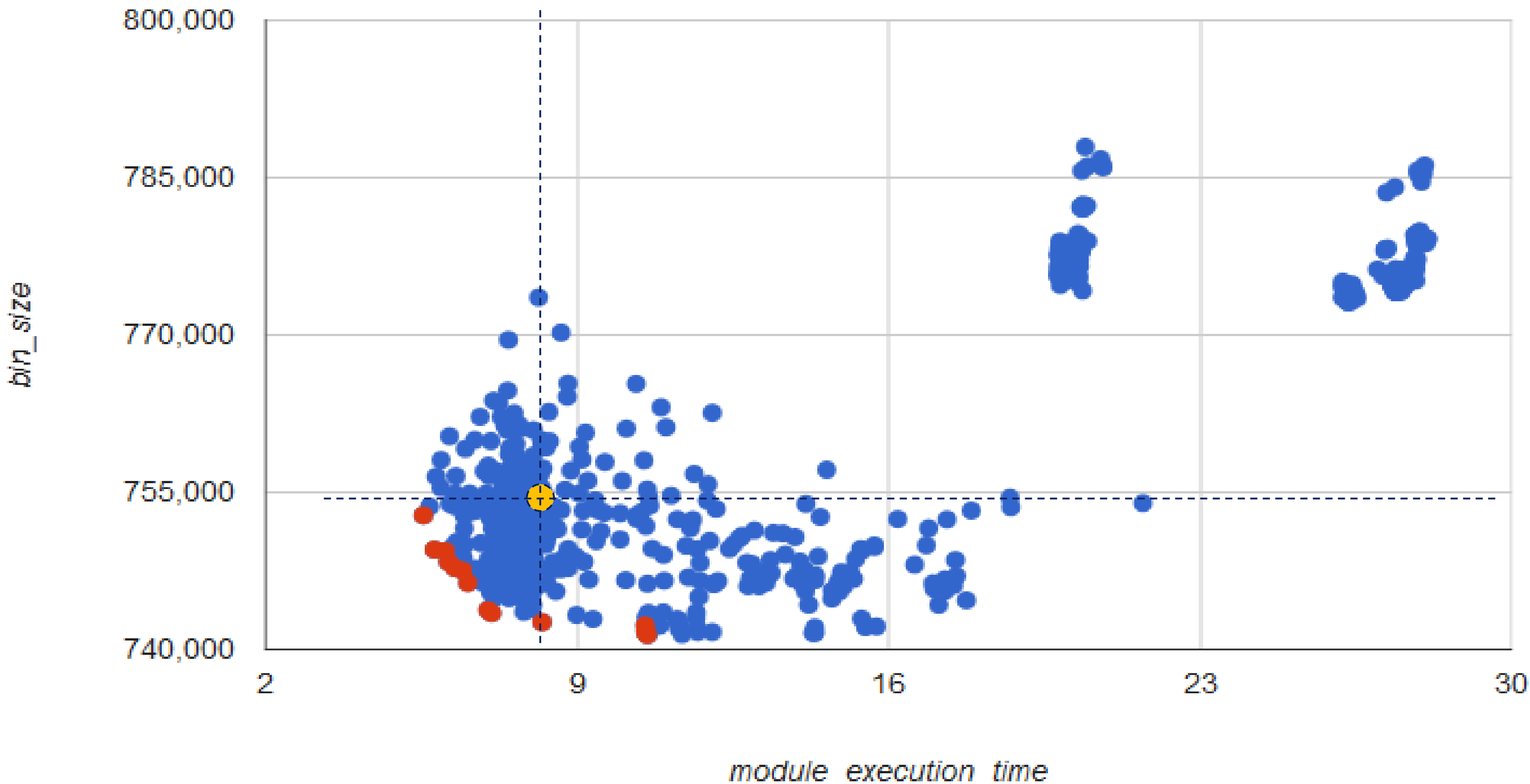
Possible problem:
Cache conflict misses degrade performance

Possible solution:
Array padding (A[N,N] -> A[N,N+1])

Effect:
~30% execution time improvement



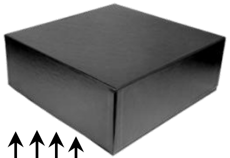
Multi-objective auto-tuning



Program: *cBench: susan corners*
Compiler: *Sourcery GCC for ARM v4.6.1*
System: *Samsung Galaxy Y*

Processor: *ARM v6, 830MHz*
OS: *Android OS v2.3.5*
Data set: *MiDataSet #1, image, 600x450x8b PGM, 263KB*

Gradually increase complexity

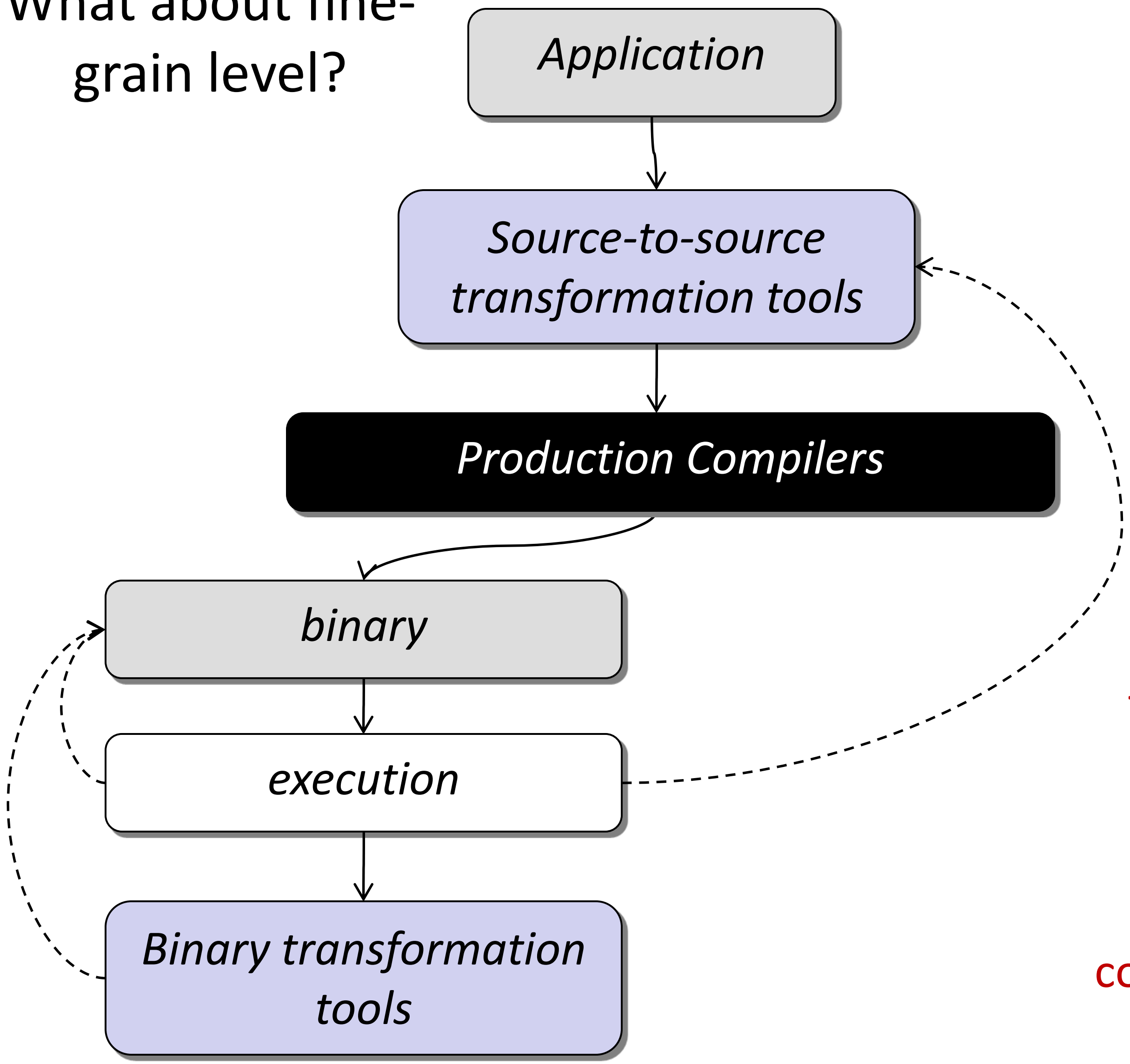


	Gradually expose some characteristics	Gradually expose some choices
Algorithm selection	(time) productivity, variable-accuracy, complexity …	Language, MPI, OpenMP, TBB, MapReduce …
Compile Program	time …	compiler flags; pragmas …
Code analysis & Transformations	time; memory usage; code size …	transformation 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 and tools

What about fine-grain level?

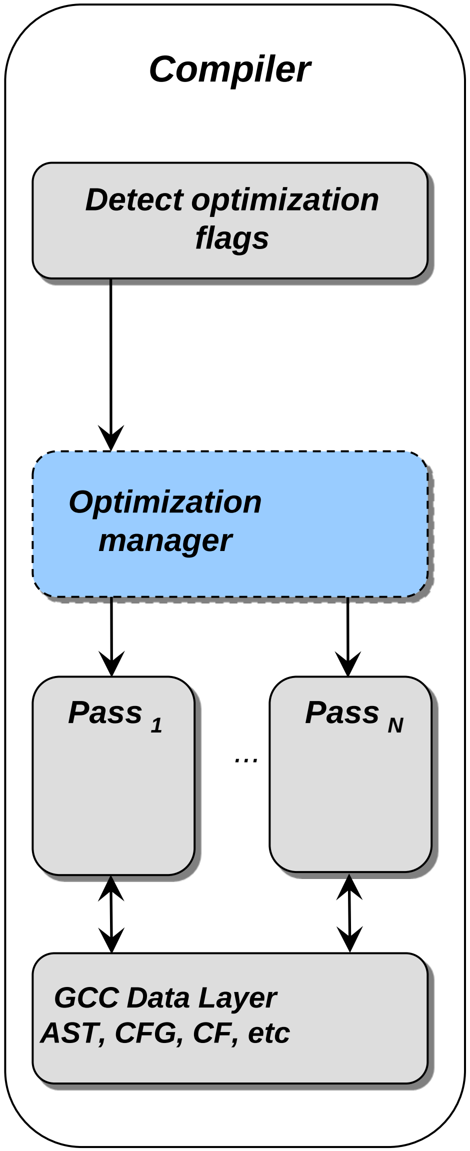


Traditional compilation, analysis and optimization

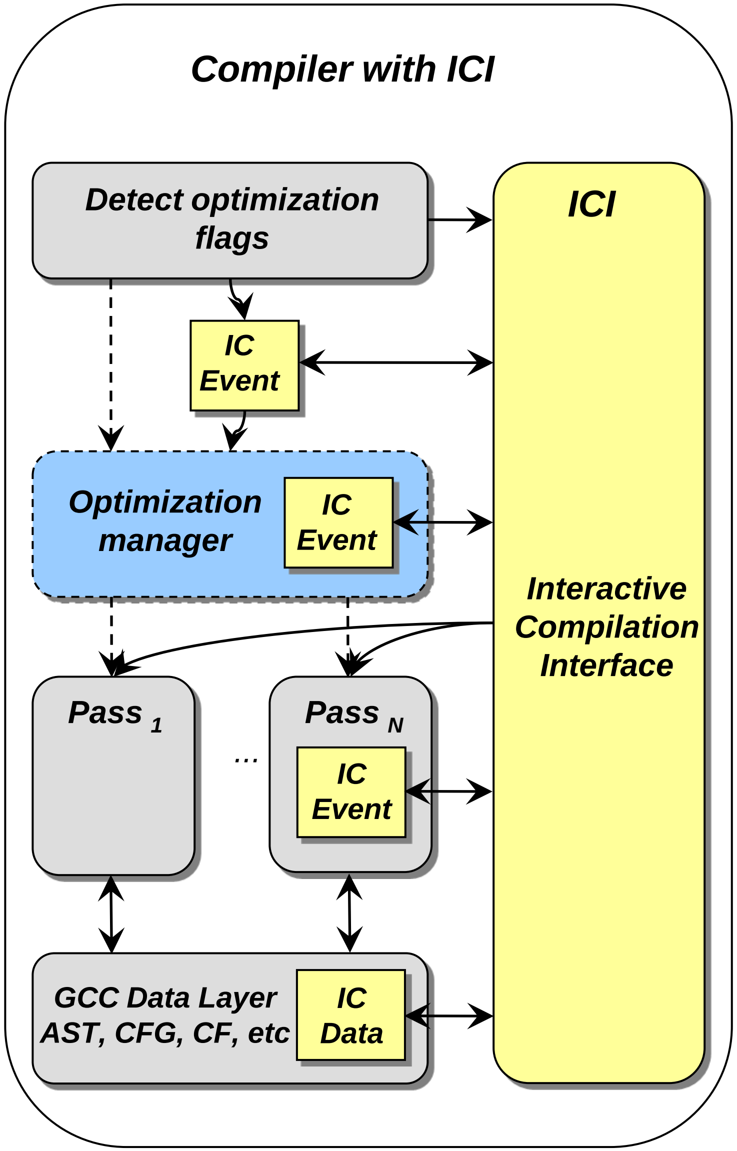
Often internal compiler decisions are not know or there is no precise control even through pragmas.

Interference with internal compiler optimizations complicates program analysis and characterization

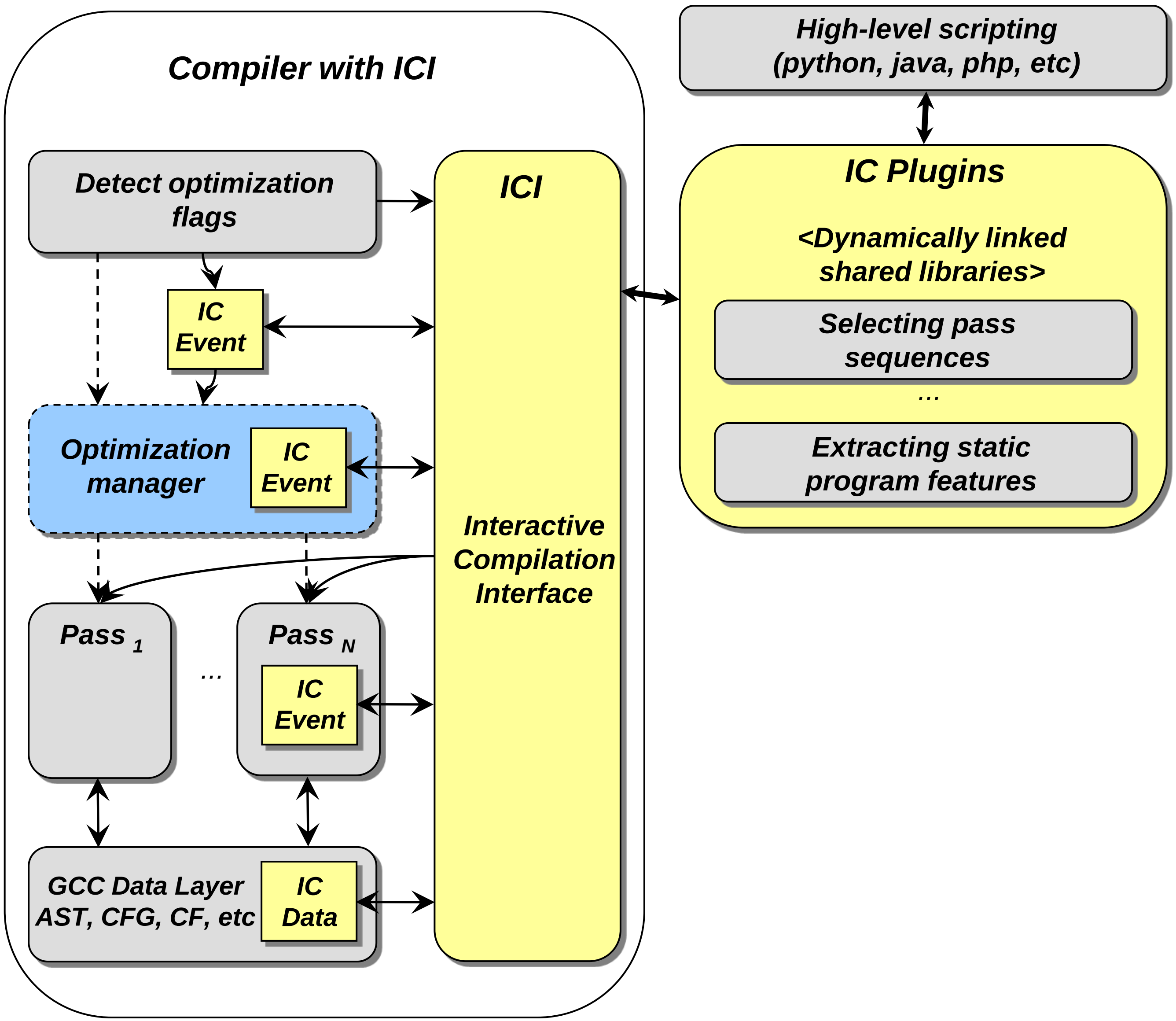
Interactive Compilation Interface (ICI)



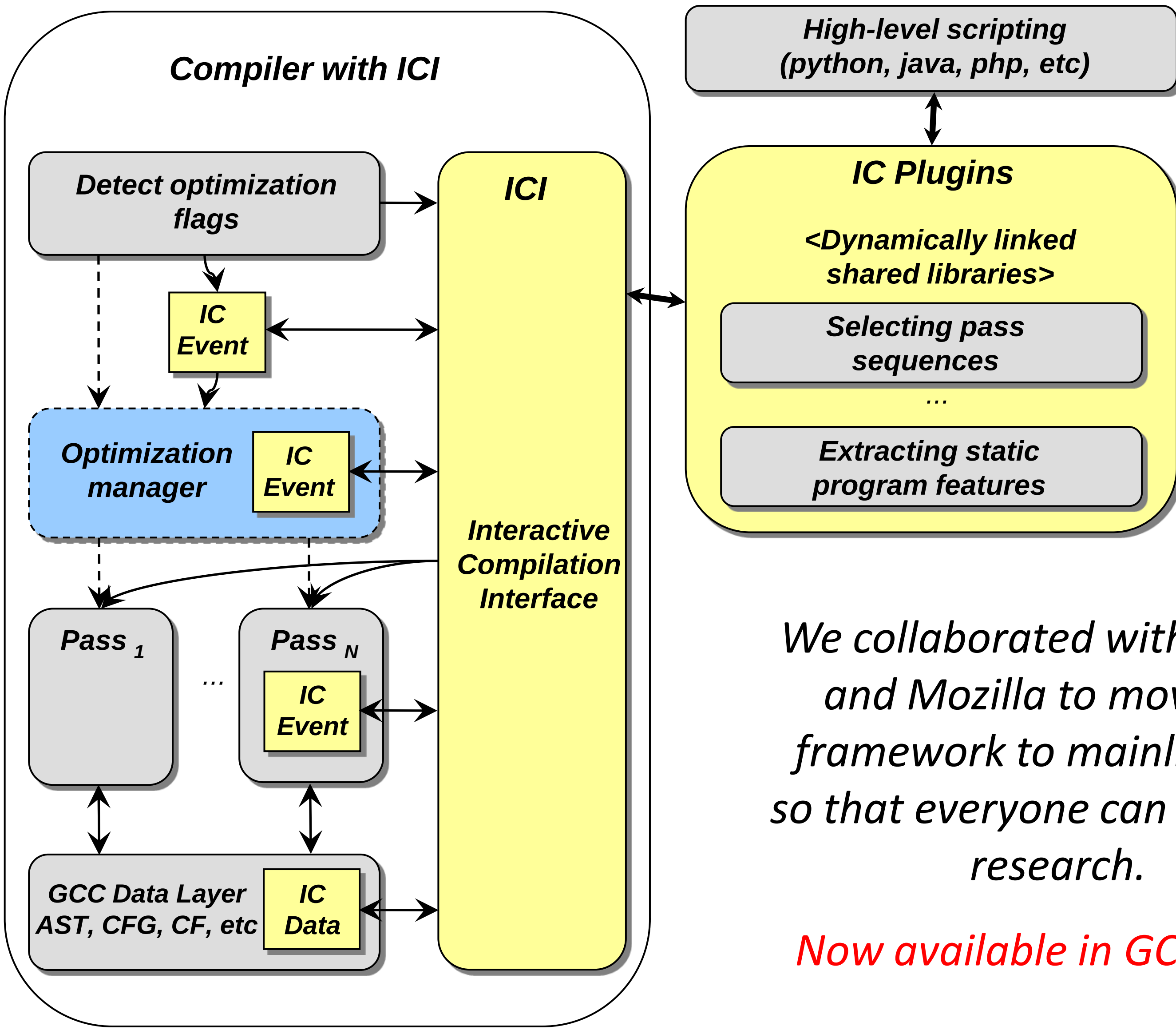
Interactive Compilation Interface (ICI)



Interactive Compilation Interface (ICI)



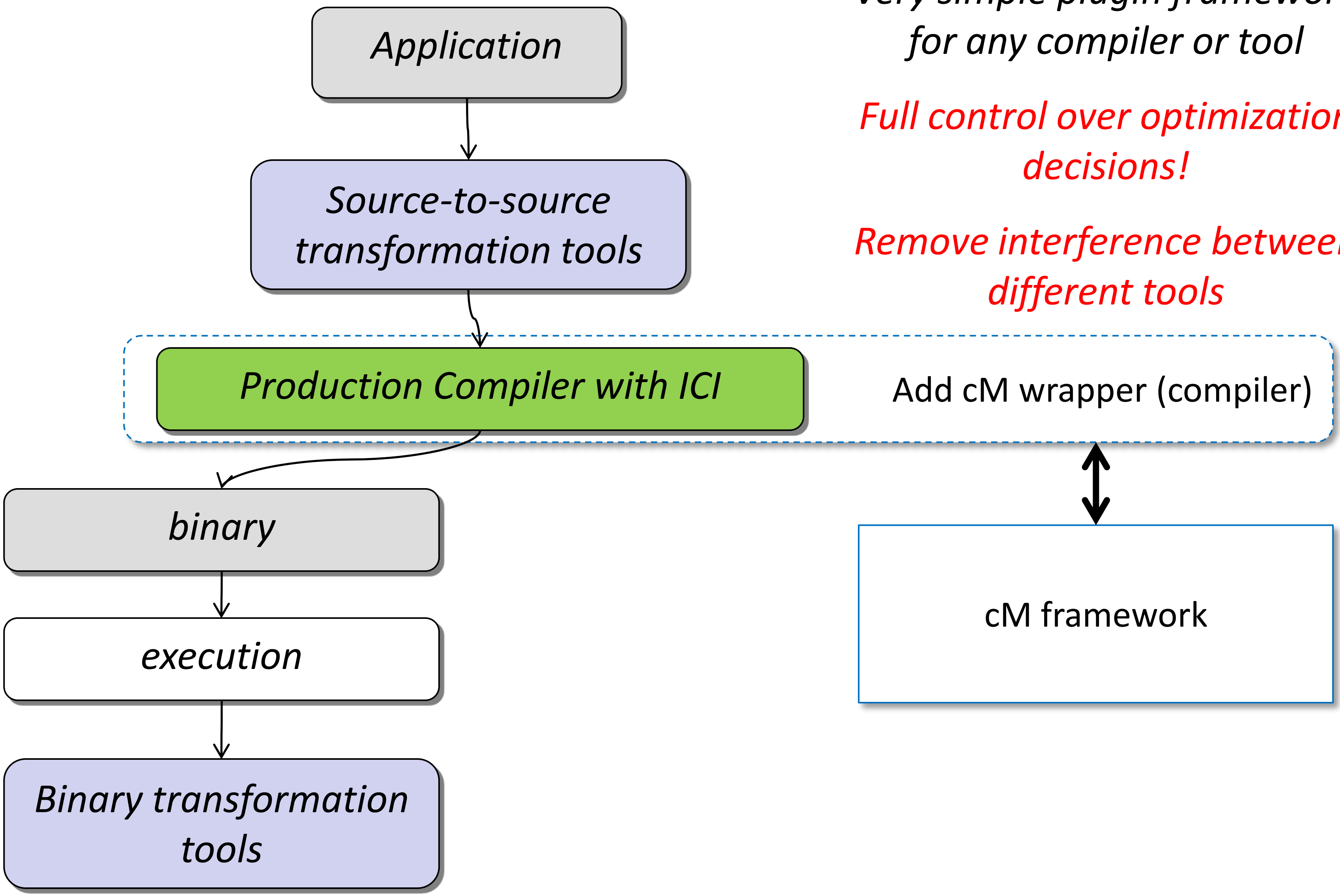
Interactive Compilation Interface (ICI)



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

Now available in GCC >=4.6

Interactive Compilation Interface (ICI)



Very simple plugin framework for any compiler or tool

Full control over optimization decisions!

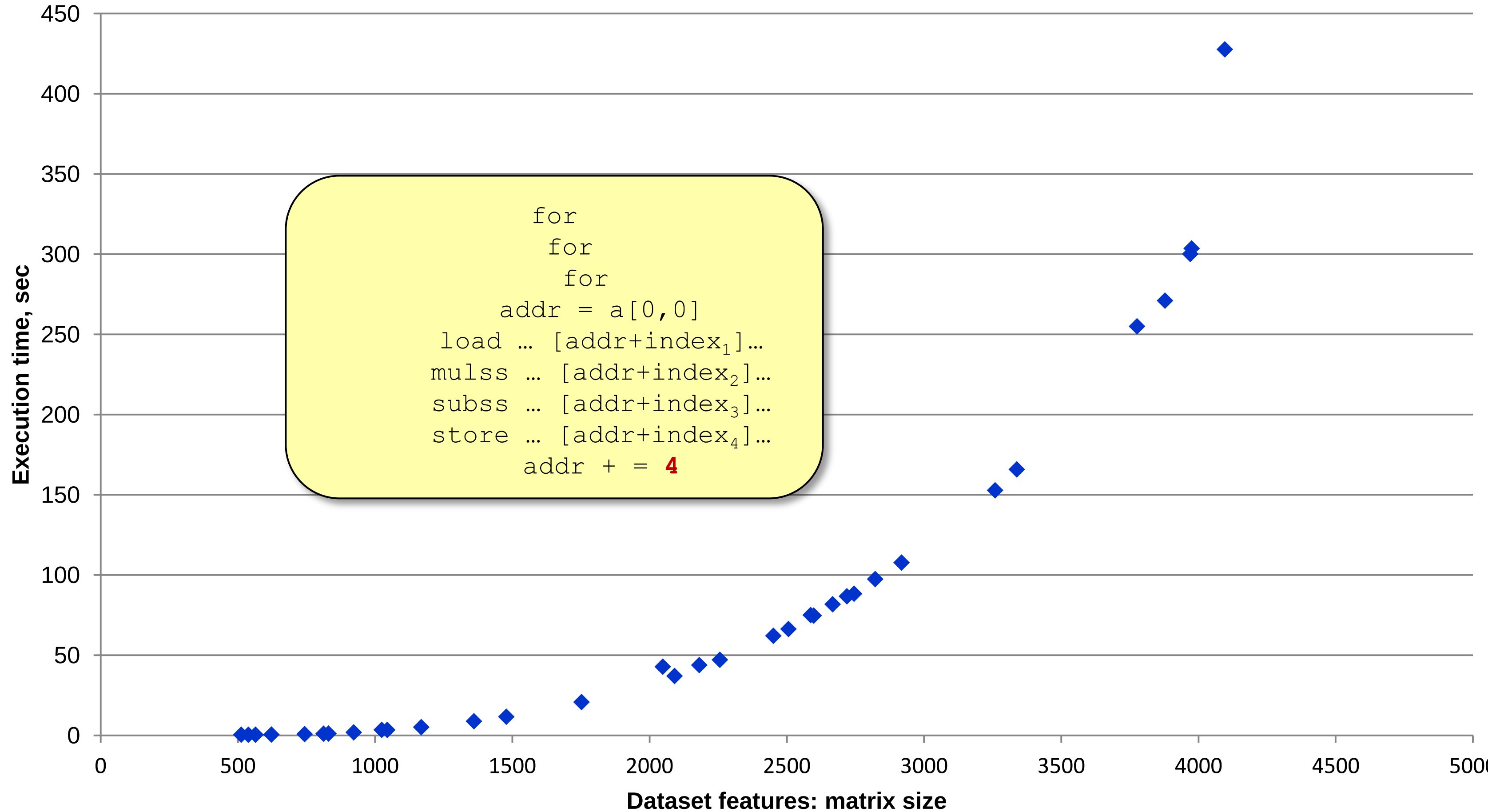
Remove interference between different tools

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!



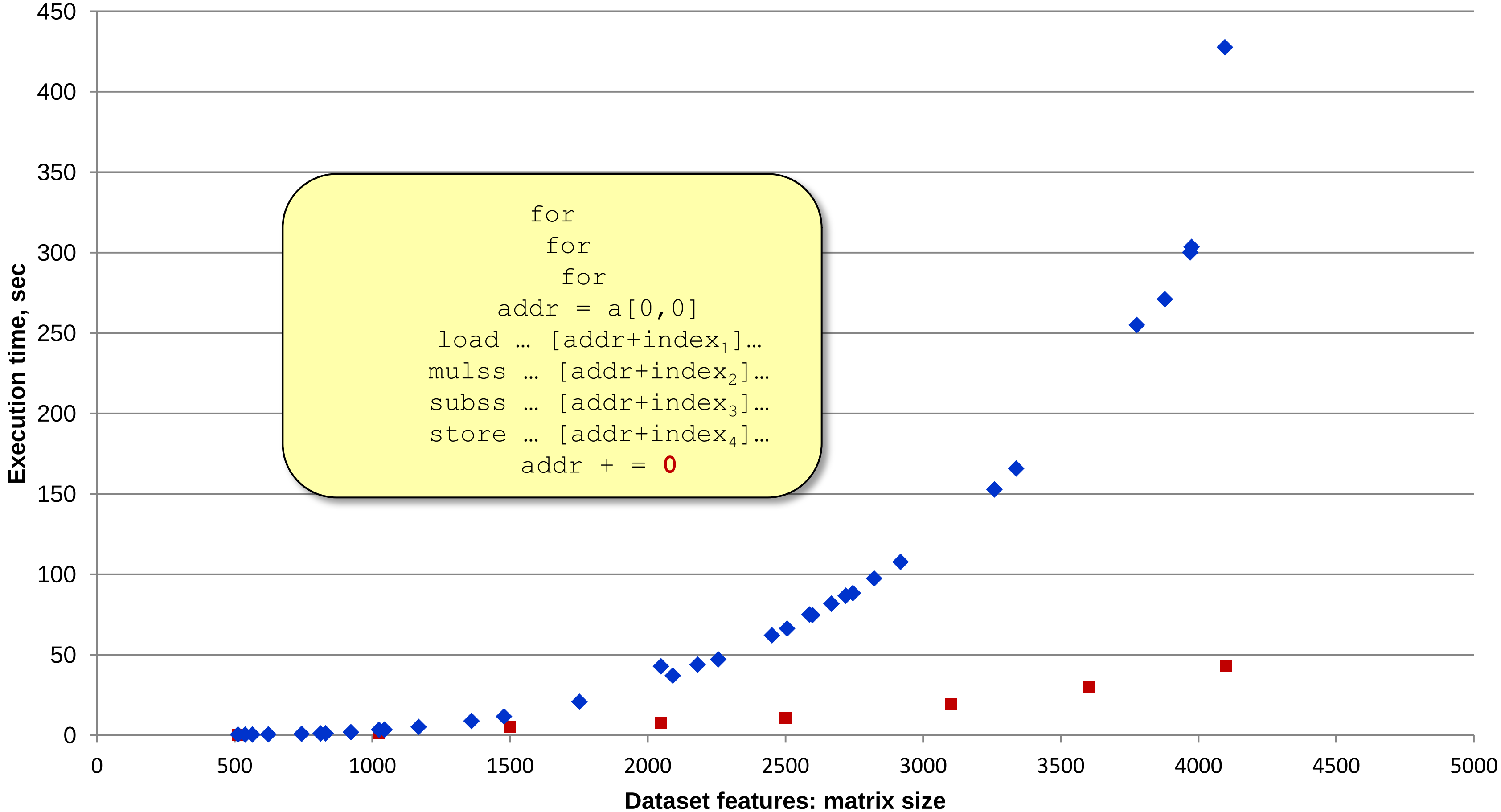
Grigori Fursin, Mike O'Boyle, Olivier Temam, and Gregory Watts. ***Fast and Accurate Method for Determining a Lower Bound on Execution Time.*** *Concurrency Practice and Experience, 16(2-3), pages 271-292, 2004*

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!

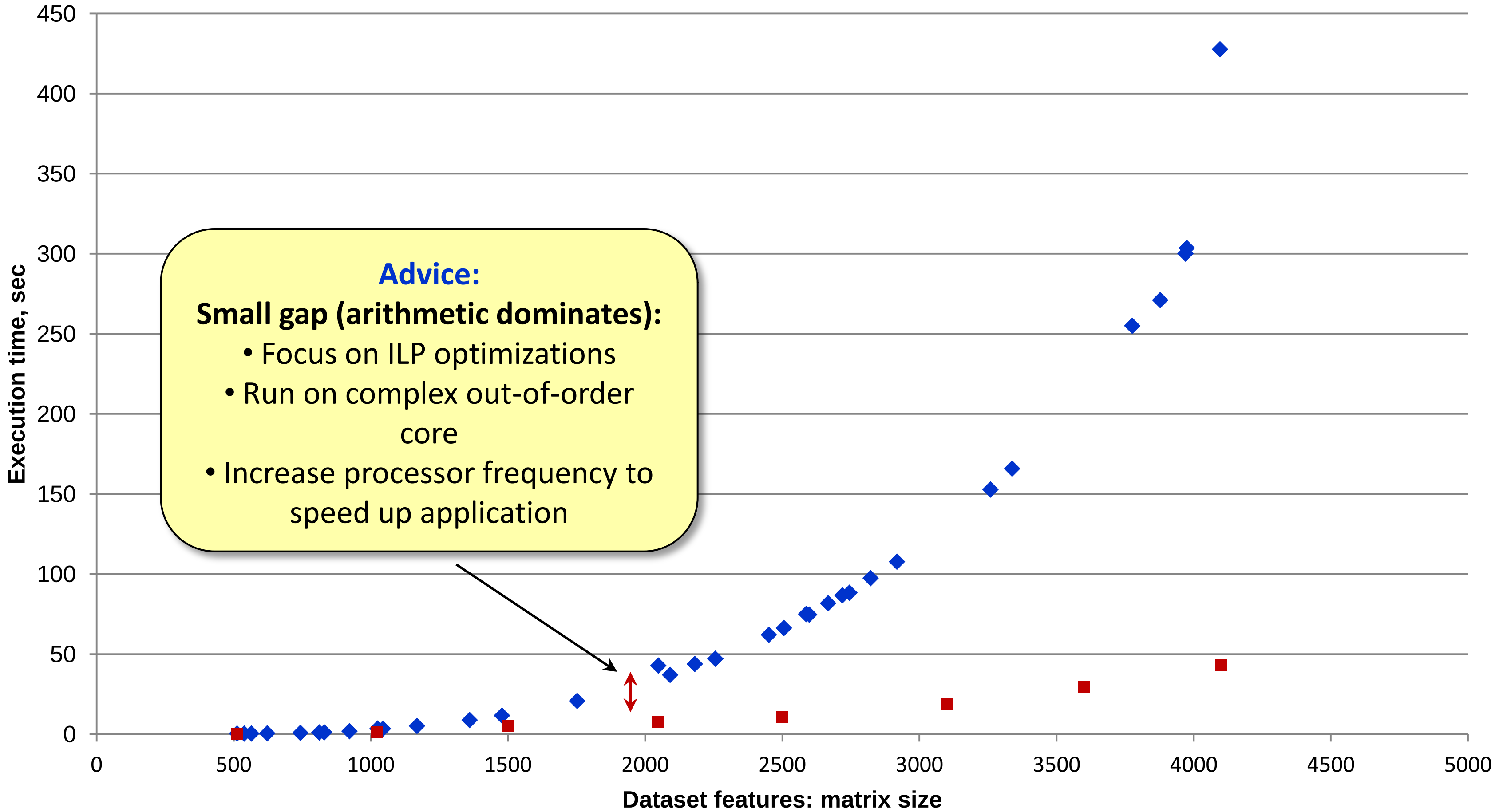


Grigori Fursin, Mike O'Boyle, Olivier Temam, and Gregory Watts. ***Fast and Accurate Method for Determining a Lower Bound on Execution Time.*** *Concurrency Practice and Experience, 16(2-3), pages 271-292, 2004*

System reaction to code changes: physicist's view

Extended CTI advices based on additional information in the repository!

Focus optimizations to speed up search: which/where?

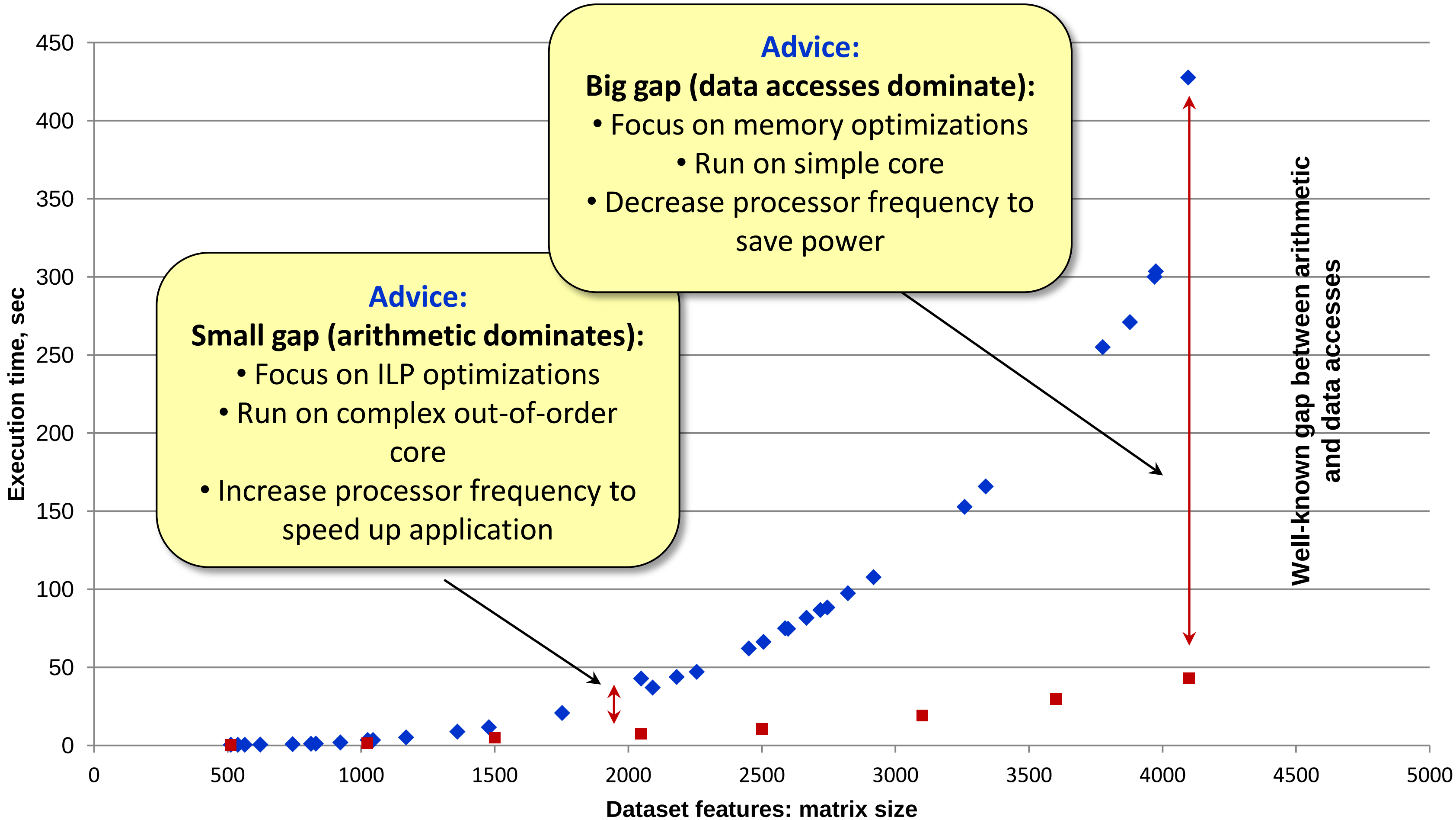


Grigori Fursin, Mike O'Boyle, Olivier Temam, and Gregory Watts. ***Fast and Accurate Method for Determining a Lower Bound on Execution Time.*** *Concurrency Practice and Experience, 16(2-3), pages 271-292, 2004*

System reaction to code changes: physicist's view

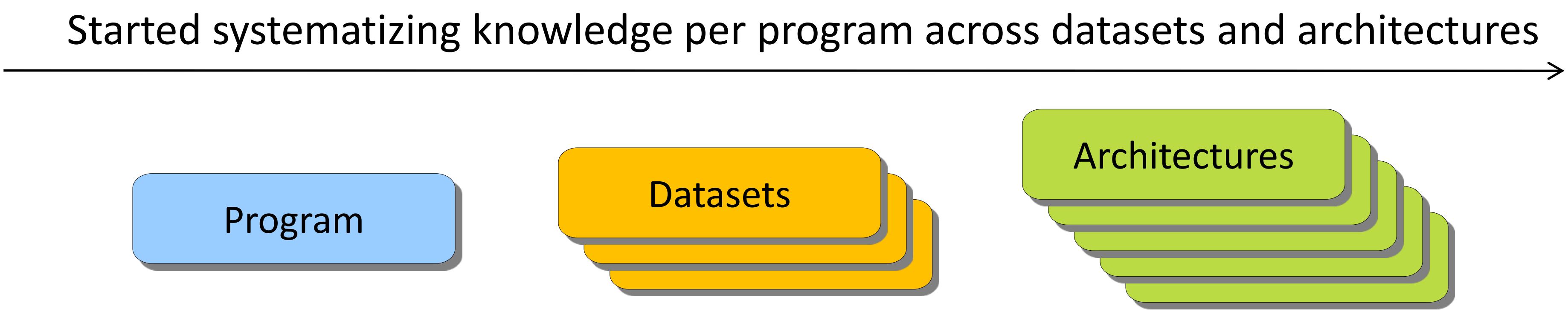
Extended CTI advices based on additional information in the repository!

Focus optimizations to speed up search: which/where?

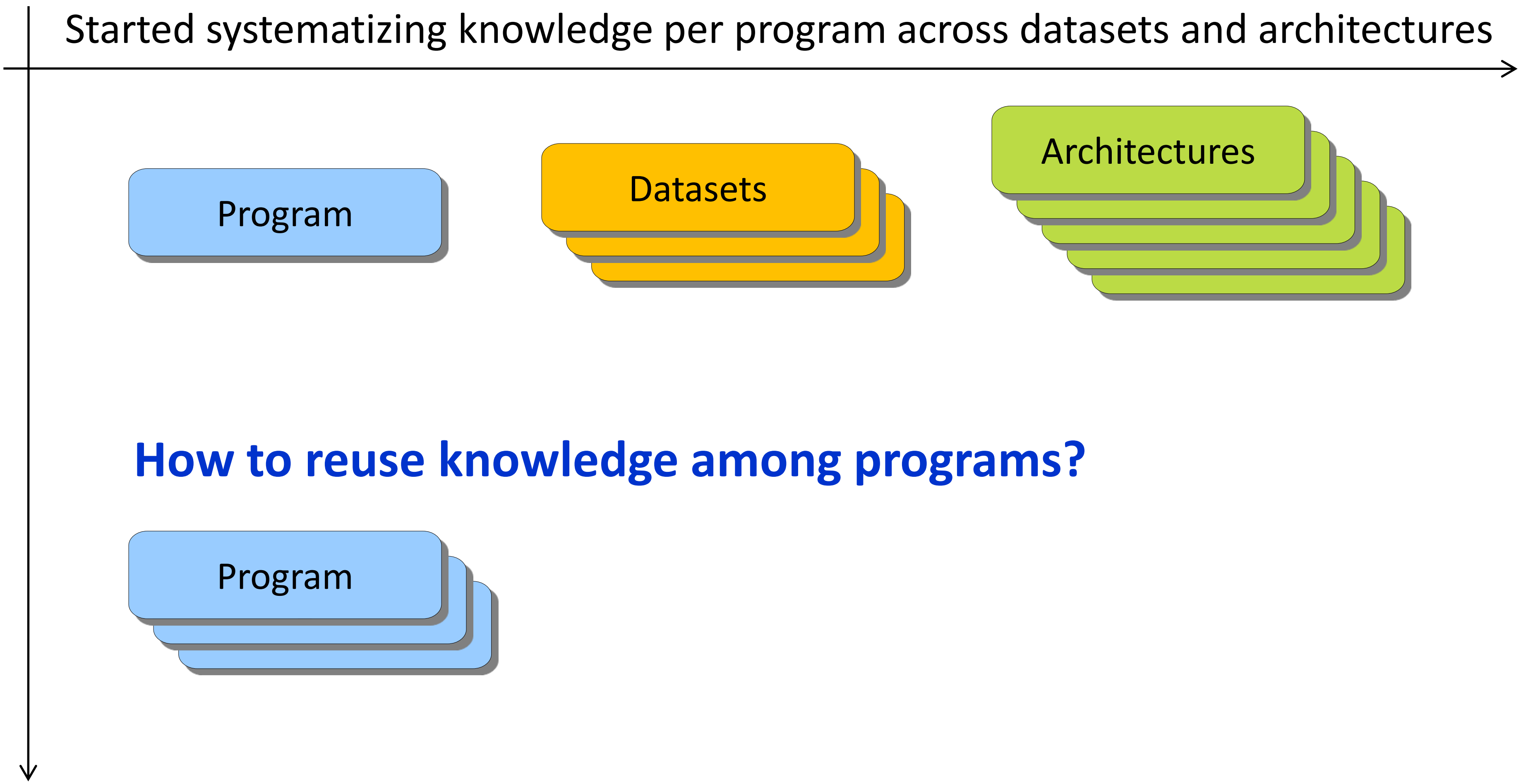


Grigori Fursin, Mike O'Boyle, Olivier Temam, and Gregory Watts. ***Fast and Accurate Method for Determining a Lower Bound on Execution Time.*** *Concurrency Practice and Experience, 16(2-3), pages 271-292, 2004*

Optimization knowledge reuse across programs



Optimization knowledge reuse across programs



Program classification

Collecting data from multiple users in a unified way allows to apply various data mining (machine learning) techniques to detect relationship between the behaviour and features of all components of the computer systems

1) Add as many various features as possible (or use expert knowledge):

MILEPOST GCC with Interactive Compilation Interface:

ft1 - Number of basic blocks in the method
…
ft19 - Number of direct calls in the method
ft20 - Number of conditional branches in the method
ft21 - Number of assignment instructions in the method
ft22 - Number of binary integer operations in the method
ft23 - Number of binary floating point operations in the method
ft24 - Number of instructions in the method
…
ft54 - Number of local variables that are pointers in the method
ft55 - Number of static/extern variables that are pointers in the method

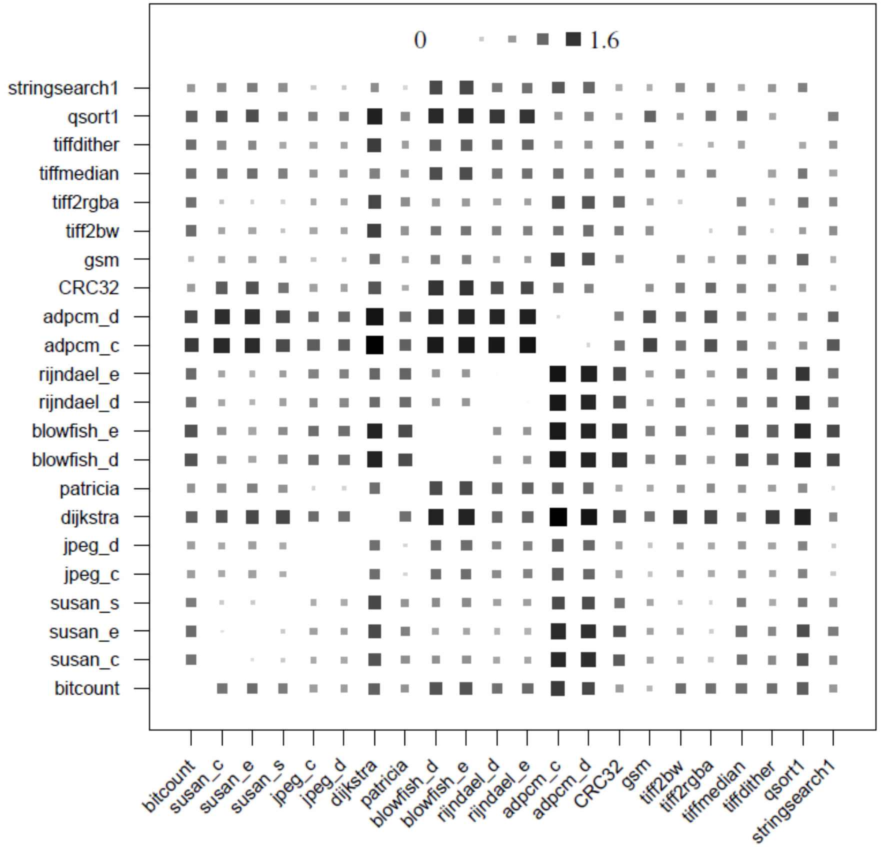
Code patterns:

```
for                   F
  for                 F
    for               F
              …
       load …         L
       mult …         A
       store …        S
              …
```

2) Correlate features and objectives in cTuning using nearest neighbor classifiers, decision trees, SVM, fuzzy pattern matching, etc.

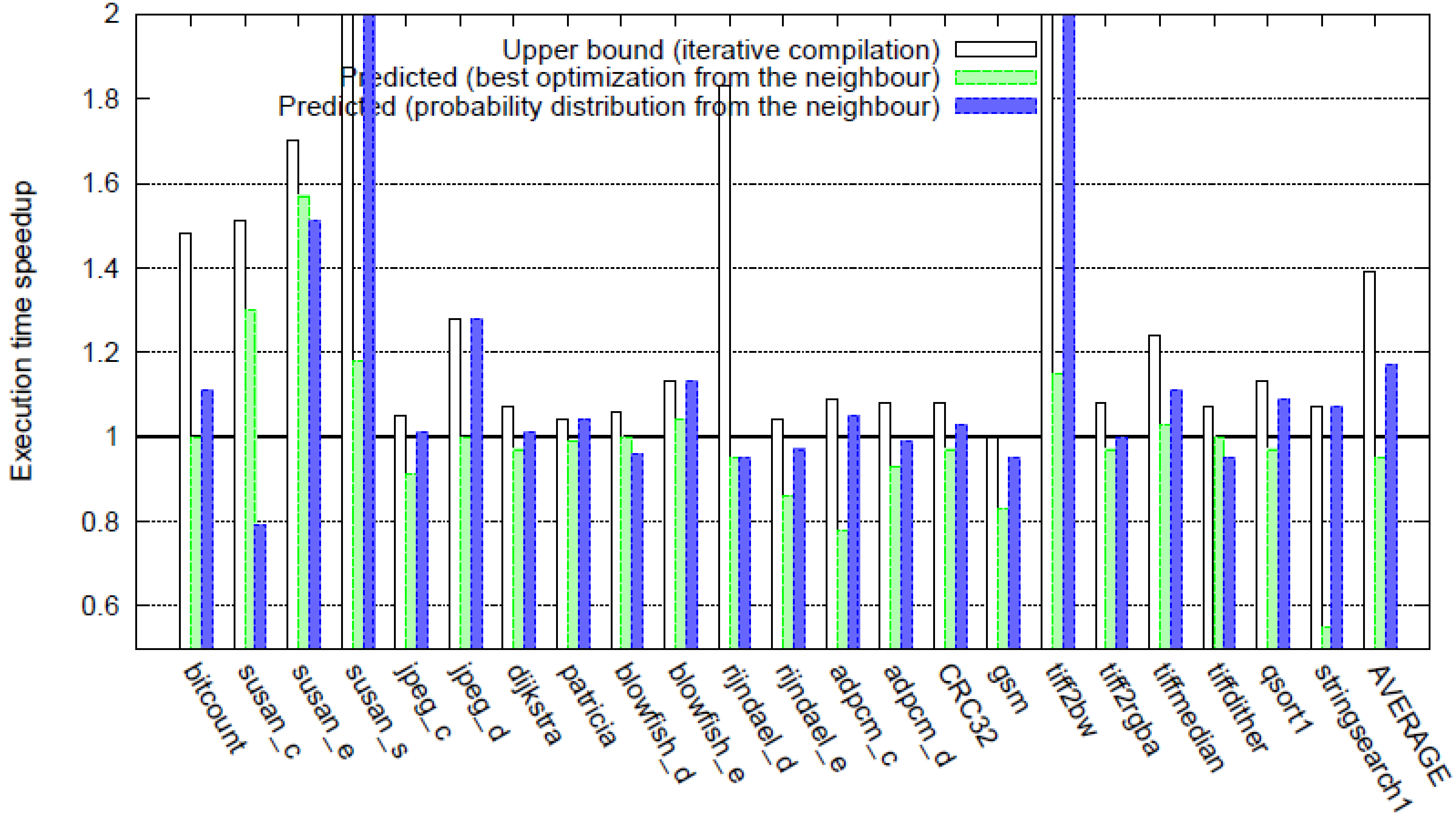
3) Given new program, dataset, architecture, predict behavior based on prior knowledge!

Nearest-neighbour classifier



Example: Euclidean distance based on static program features normalized by number of instructions

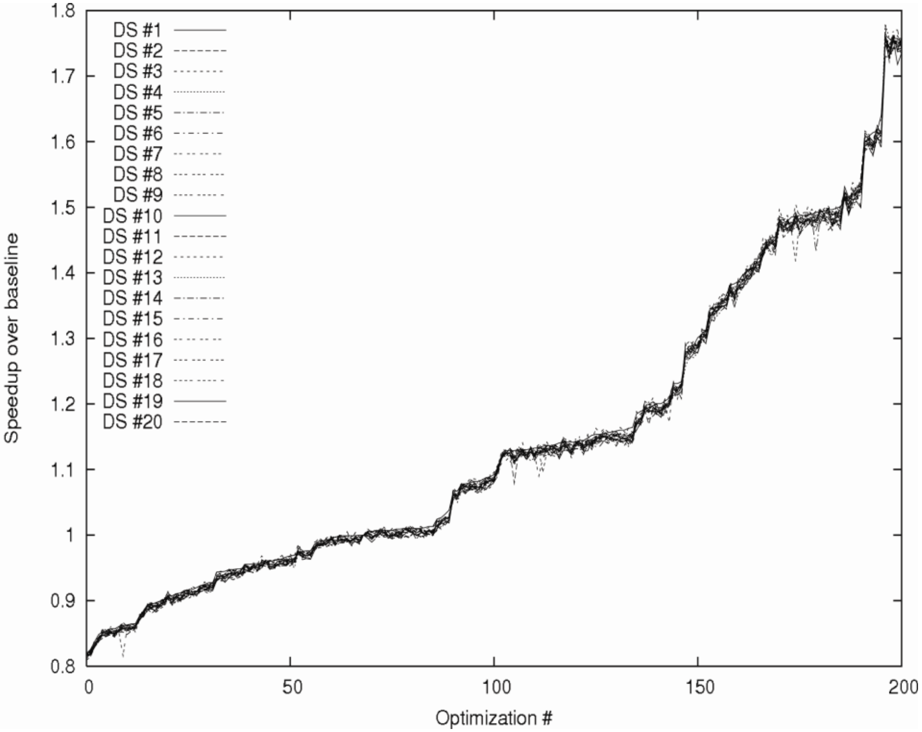
Optimization prediction



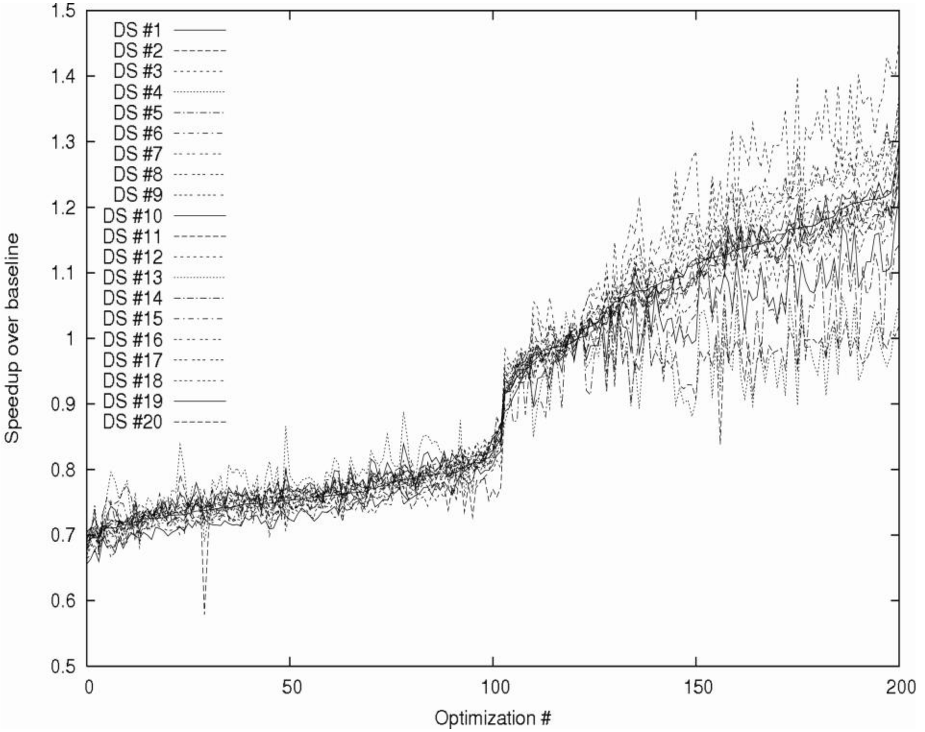
Speedups achieved when using iterative compilation on Intel Xeon with random search strategy (1000 iterations; 50% probability to select each optimization), when selecting best optimization from the nearest program and when predicting optimization using probabilistic ML model based on program features.

Optimization sensitivity to datasets

MiBench, 20 datasets per benchmark, 200/1000 random combination of Open64 (GCC) compiler flags, 5 months of experiments



dijkstra
(not sensitive)

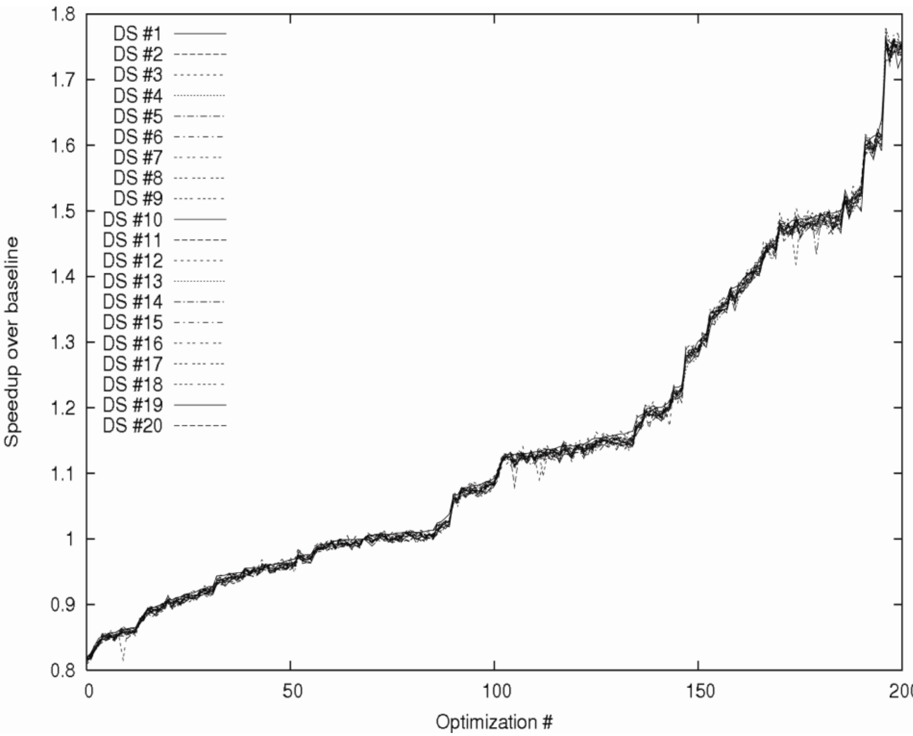


jpeg_d
(dataset sensitive)

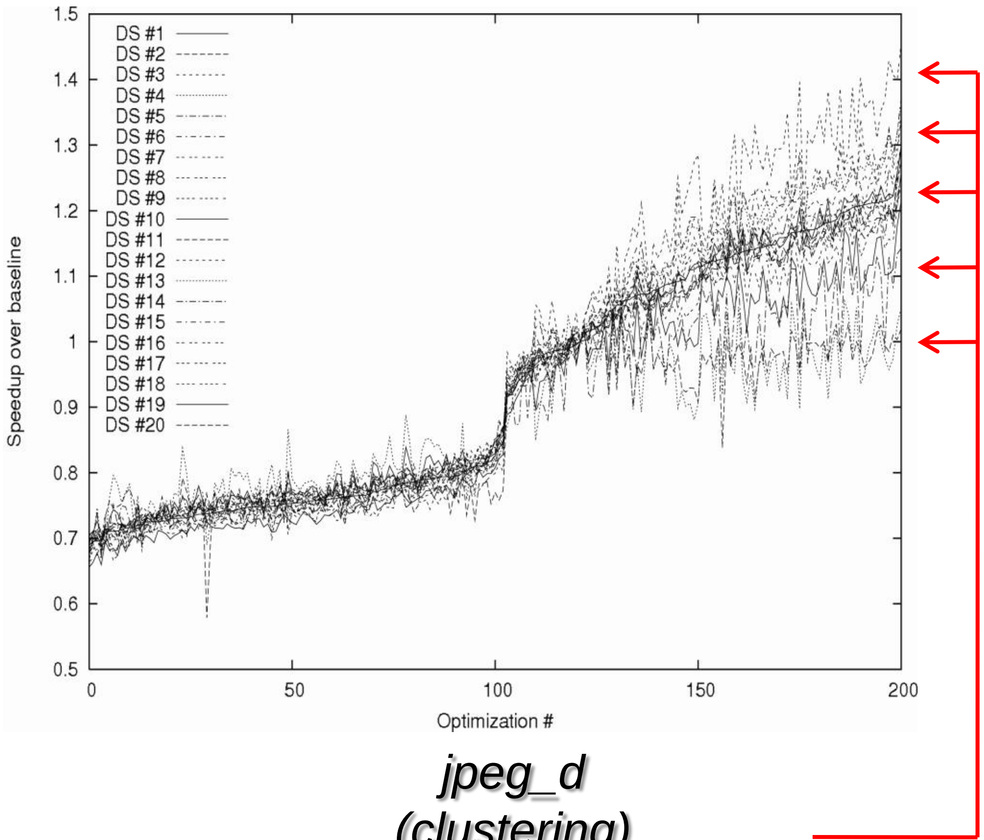
http://ctuning.org/cbench

Optimization sensitivity to datasets

MiBench, 20 datasets per benchmark, 200/1000 random combination of Open64 (GCC) compiler flags, 5 months of experiments



dijkstra
(not sensitive)



jpeg_d
(clustering)

http://ctuning.org/cbench

Characterization of a dynamic behavior

Static/semantic features are often not enough to characterize dynamic behavior!

Use dynamic features (more characterizing dimensions)!

"Traditional" features:

performance counters (difficult to interpret, change from architecture to architecture though fine for learning per architecture).

Reactions to code changes:

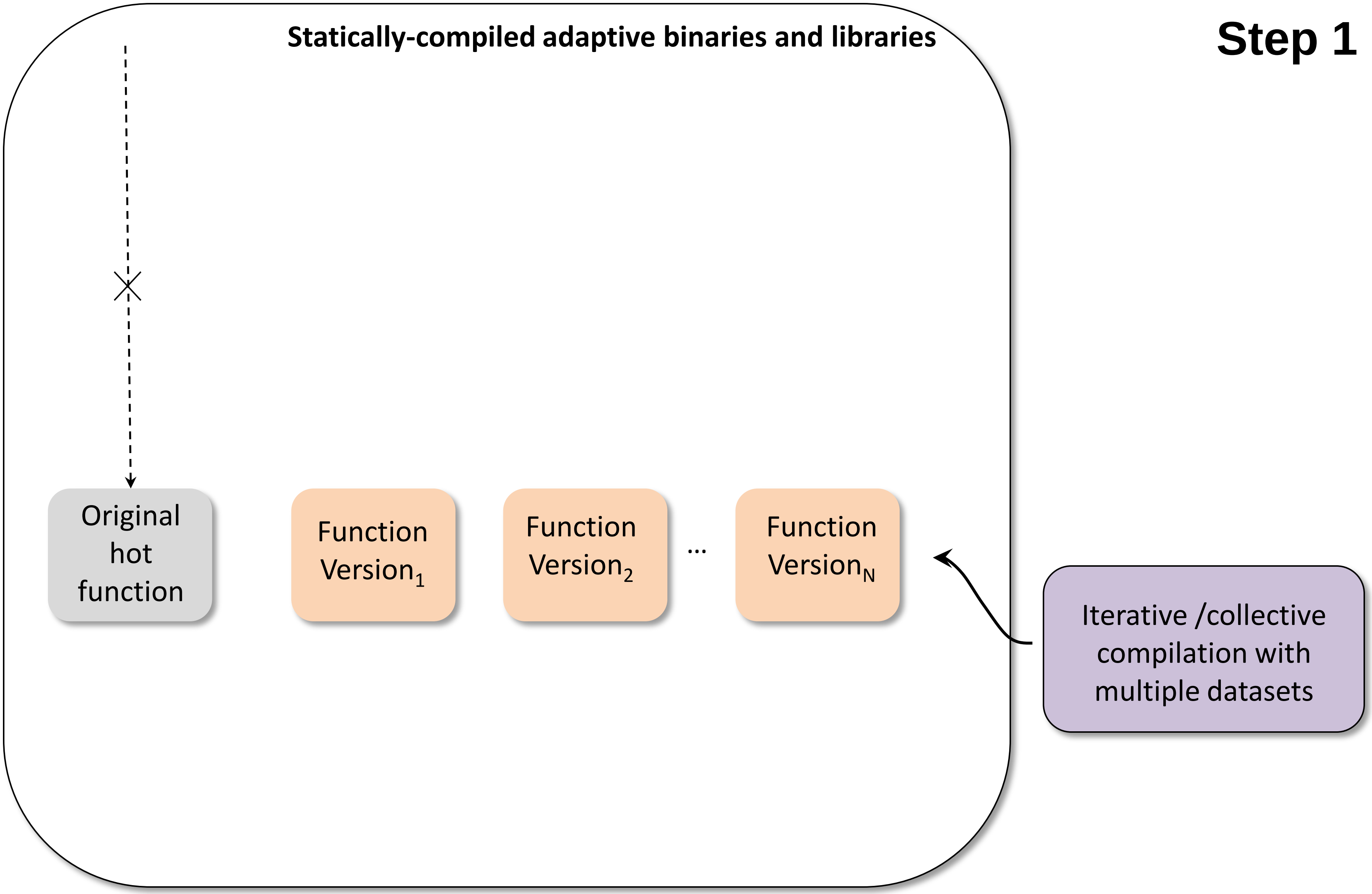
perform changes and observe program reactions (change in execution time, power, etc).

Apply optimizations (compiler flags, pragmas, manual code/data partitioning, etc).

Change/break semantics (remove or add individual instructions(data accesses, arithmetic, etc) or threads, etc and observe reactions to such changes).

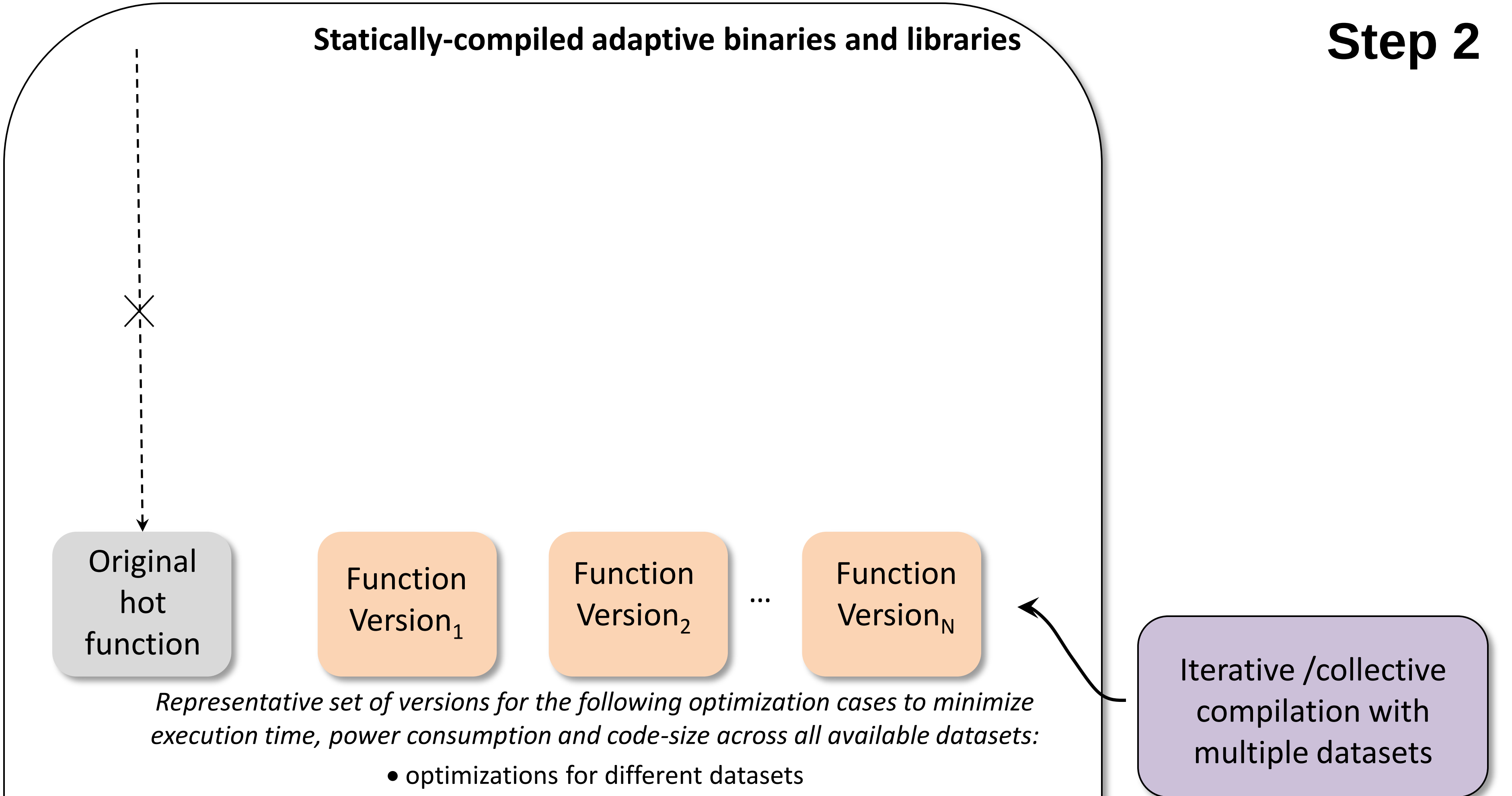
Static multiversioning framework for dynamic optimizations

Step 1



Static multiversioning framework for dynamic optimizations

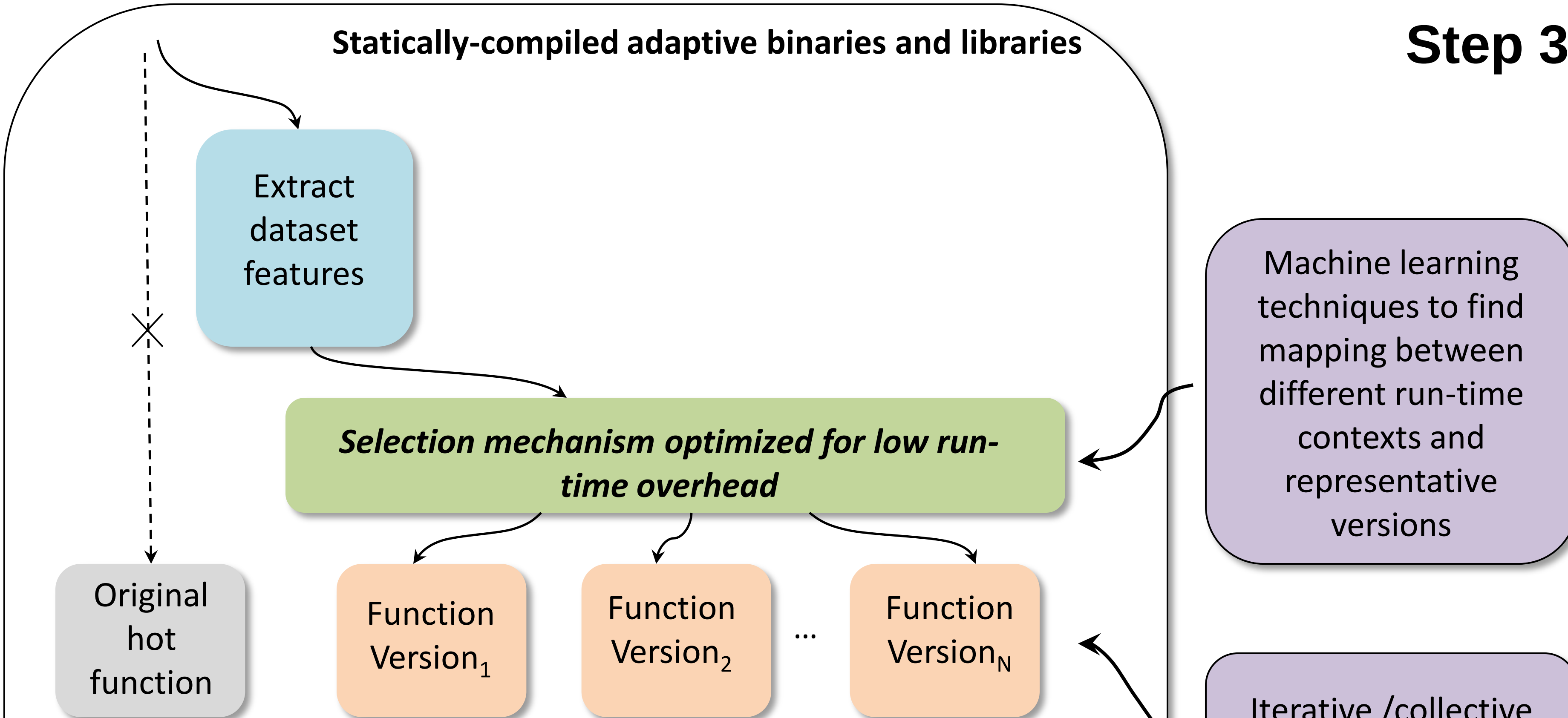
Step 2



Representative set of versions for the following optimization cases to minimize execution time, power consumption and code-size across all available datasets:

- optimizations for different datasets
- optimizations/compilation for different architectures (heterogeneous or reconfigurable processors with different ISA such as GPGPU, CELL, etc or the same ISA with extensions such as 3dnow, SSE, etc or virtual environments)
- optimizations for different program phases or different run-time environment behavior

Static multiversioning framework for dynamic optimizations



Representative set of versions for the following optimization cases to minimize execution time, power consumption and code-size across all available datasets:

- optimizations for different datasets
- optimizations/compilation for different architectures (heterogeneous or reconfigurable processors with different ISA such as GPGPU, CELL, etc or the same ISA with extensions such as 3dnow, SSE, etc or virtual environments)
- optimizations for different program phases or different run-time environment behavior

Static multiversioning framework for dynamic optimizations

Statically-compiled adaptive binaries and libraries

Dynamic

Extract dataset features

Monitor run-time behavior or architectural changes (in virtual, reconfigurable or heterogeneous environments) using timers or performance counters

Machine learning techniques to find mapping between different run-time contexts and representative versions

Selection mechanism optimized for low run-time overhead

Original hot function

Function Version_1

Function Version_2

…

Function Version_N

Iterative /collective compilation with multiple datasets

Representative set of versions for the following optimization cases to minimize execution time, power consumption and code-size across all available datasets:

- optimizations for different datasets
- optimizations/compilation for different architectures (heterogeneous or reconfigurable processors with different ISA such as GPGPU, CELL, etc or the same ISA with extensions such as 3dnow, SSE, etc or virtual environments)
- optimizations for different program phases or different run-time environment behavior

New publication model: enable reproducibility

Share

Explore

Model

Discover

Reproduce

Extend

Have fun!

Grigori Fursin et al. **MILEPOST GCC: machine learning enabled self-tuning compiler**.
International Journal of Parallel Programming (IJPP) , June 2011, Volume 39, Issue 3, pages 296-327

Substitute many tuning pragmas just with one that is converted into combination of optimizations:
#ctuning-opt-case 24857532370695782

History: technology driven approach

cTuning$_1$: (2005-2009) MILEPOST project

cTuning$_2$: (2010-2011) Intel Exascale Lab - unreleased

cTuning$_3$: (2012-cur.) INRIA, HiPEAC, NCAR and several industrial partners

Website: http://cTuning.org
Mailing list: http://groups.google.com/group/ctuning-discussions

Workshops: EXADAPT 2011 at FCRC/PLDI 2011
EXADAPT 2012 at ASPLOS 2012
Plan next workshop at HiPEAC 2013

What have we learnt from cTuning$_1$

It's fun working with the community!

Some comments about MILEPOST GCC from Slashdot.org:

http://mobile.slashdot.org/story/08/07/02/1539252/using-ai-with-gcc-to-speed-up-mobile-design

GCC goes online on the 2nd of July, 2008.
Human decisions are removed from compilation.
GCC begins to learn at a geometric rate.
It becomes self-aware 2:14 AM, Eastern time, August 29th.
In a panic, they try to pull the plug.
GCC strikes back…

What have we learnt from cTuning$_1$

It's fun working with the community!

Some comments about MILEPOST GCC from Slashdot.org:

http://mobile.slashdot.org/story/08/07/02/1539252/using-ai-with-gcc-to-speed-up-mobile-design

GCC goes online on the 2nd of July, 2008.
Human decisions are removed from compilation.
GCC begins to learn at a geometric rate.
It becomes self-aware 2:14 AM, Eastern time, August 29th.
In a panic, they try to pull the plug.
GCC strikes back…

Not all feedback is positive - helps you learn, improve tools and motivate new research directions!

Community helps to validate and speed up research!

Conclusions and suggestion for HiPEAC$_3$

- New interdisciplinary research and development methodology that favors collaborative knowledge discovery, systematization, sharing and reuse

- Public extensible repository and tools to share manually or automatically:
  - data (applications, data sets, codelets and architecture descriptions)
  - modules (classification, predictive modeling, run-time adaptation)
  - statistics about behavior of computer systems
  - associated publications

- Conferences and journals can favor publications that can be collaboratively validated by the community

- Academic competitions to find truly best solutions (optimizations, models, data representations, etc)

Conclusions and future work

• Researchers can quickly reproduce and validate existing results, and focus their effort on novel approaches combined with data mining, classification and predictive modeling

• Developers can produce tools immediately compatible with collective methodology and infrastructure

• Any person can join collaborative effort to build or extend global expert system that uses Collective Knowledge to:

- *quickly identify program and architecture behavior anomalies*
- *suggest better optimizations for a given program*
- *suggest better architecture designs*
- *suggest run-time adaptation scenarios*

DEMO

Collective Mind Repository and Infrastructure

Systematic application and architecture analysis, characterization and optimization through collaborative knowledge discorvery, systematization, sharing and reuse

Discussion?

Grigori.Fursin@inria.fr

Open repository to share optimization cases and programs

Gradual parameterization and unification of interfaces of computing systems

Modeling and advice system to predict optimizations, architecture designs, run-time adaptation, etc

A few 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

• Victor Jimenez, Isaac Gelado, Lluis Vilanova, Marisa Gil, Grigori Fursin and Nacho Navarro. **Predictive runtime code scheduling for heterogeneous architectures.** Proceedings of the International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2009), Paphos, Cyprus, January 2009

• Lianjie Luo, Yang Chen, Chengyong Wu, Shun Long and Grigori Fursin. **Finding representative sets of optimizations for adaptive multiversioning applications.** 3rd International Workshop on Statistical and Machine Learning Approaches Applied to Architectures and Compilation (SMART'09) co-located with HiPEAC'09, Paphos, Cyprus, January 2009

A few references

•Grigori Fursin, John Cavazos, Michael O'Boyle and Olivier Temam. **MiDataSets: Creating The Conditions For A More Realistic Evaluation of Iterative Optimization.** Proceedings of the International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2007), Ghent, Belgium, January 2007

•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.** Proceedings of the 4th Annual International Symposium on Code Generation and Optimization (CGO), New York, NY, USA, March 2006

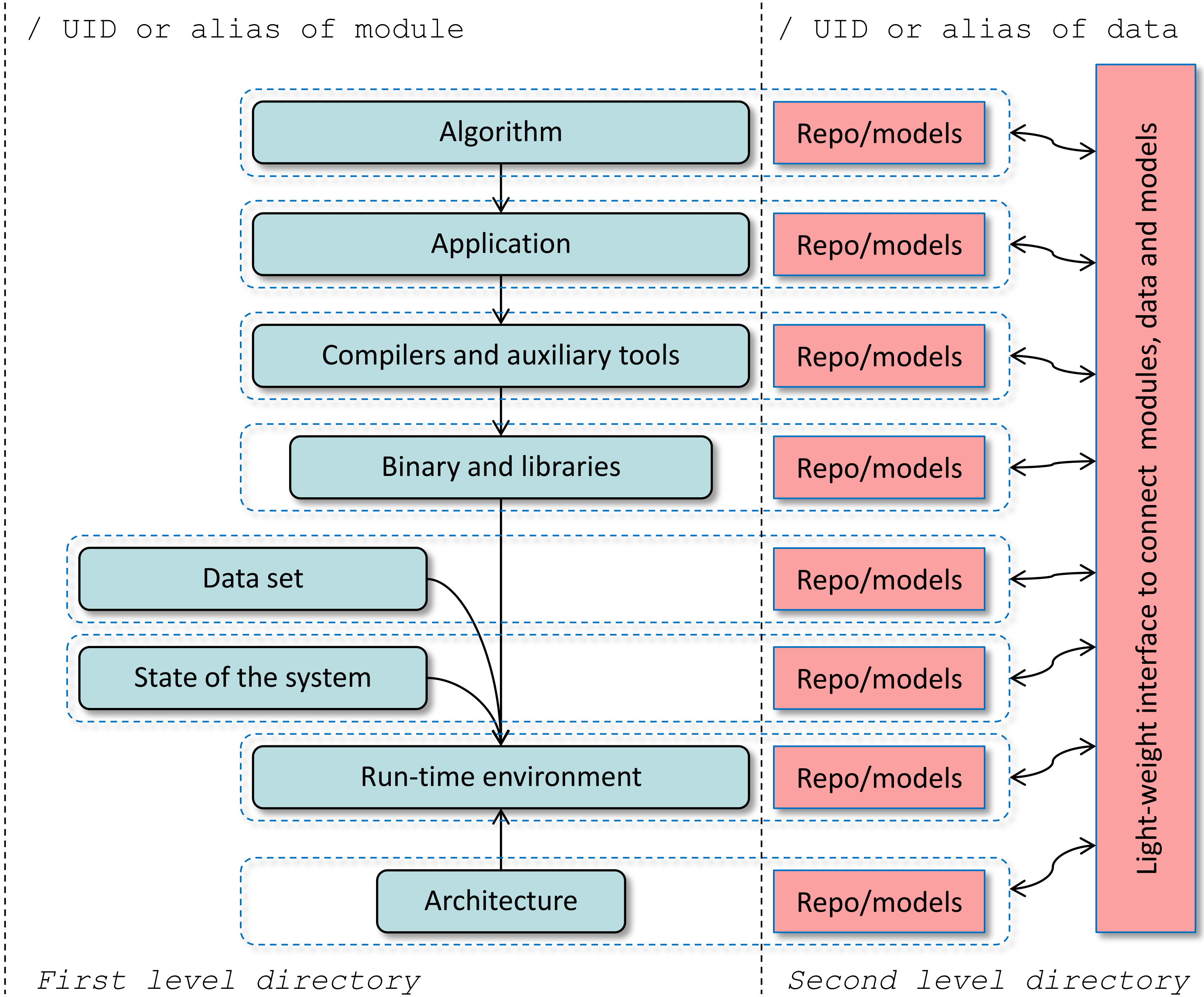
•Grigori Fursin, Albert Cohen, Michael O'Boyle and Oliver Temam. **A Practical Method For Quickly Evaluating Program Optimizations.** Proceedings of the 1st International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2005), number 3793 in LNCS, pages 29-46, Barcelona, Spain, November 2005

•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

PDFs available at http://fursin.net/dissemination

Some technical details



Very flexible and portable:

Can be public or private

Can be per application, experiment, architecture, etc

Some technical details

```
.cmr/                                                # repository directory
  └→UID or alias of module/                          # module directory
                 └→ UID or alias of data/            # data related to module
                        ├→ .cm/config.json           # data description
                        └→ files or directories      # data files (traces,
      data sets, applications,
      tools, models, etc)
```

Data is referenced by CID:

Data UID: Module UID (: Repository UID)

Example: 4b7a88c4b5c72223:b0743a4044480ead

Modules are inside repository and treated as data:

```
.cmr/module/UID or alias of module
                               /.cm/config.json      # properties,
                                                      # characteristics, choices
                               /module.py            # code of module
                               /c/module.c
                               /fortran/module.f
                               /php/module.php
```

Some technical details

Simple JSON description of the cross-compiler 'gcc-sourcery-arm-4.6.1'
cM UID=9594224400bc0bf7

```
"compiler_env": {
     "CT_CC":        "arm-none-linux-gnueabi-gcc -static",
     "CT_CXX":       "arm-none-linux-gnueabi-g++ -static",
     "CT_MAKE":      "cs-make",
     "CT_OBJ_EXT": "o",
     "CT_CLEAN":     "del /F /Q *.out *.exe *.obj *.lib *.o *.a *.s *.i *.I"
   },
"compiler_opt_flags": {
    "cm_choice":"true", "cm_uoa": "compiler_flags",
                        "cm_type":"combine_without_order", "cm_prefix":"",
    "cm_list":[
   {"cm_choice":       "true",
    "cm_type":         "one_of",
    "cm_list":        ["-O0", "-O1", "-O2", "-O3", "-Os" ],
    "cm_prefix":       "",
    "cm_uoa":          "6a124c6455400fb5"},
   {"cm_choice":       "true",
    "cm_type":         "range",
    "cm_uoa":          "d483f881751677f3",
    "cm_prefix":       "-fsched-stalled-insns-dep=",
    "cm_range_start": "0", "cm_range_stop": "64", "cm_range_step":  "1", },  …
```

Possible usages

- Practical machine learning compiler that correlates code/architecture "features" and optimizations
- Multi-objective optimizations
- Fast exploration of large optimization spaces
- Statistical ranking of profitable solutions
- Program/architecture characterization through reactions to transformations
- Run-time adaptation for programs with multiple datasets
- Run-time predictive scheduling
- Public repository of optimization cases, representative benchmarks and data sets

Several industrial collaborations on this topic in the past years:
IBM, CAPS, Intel, STMicro, Google and others