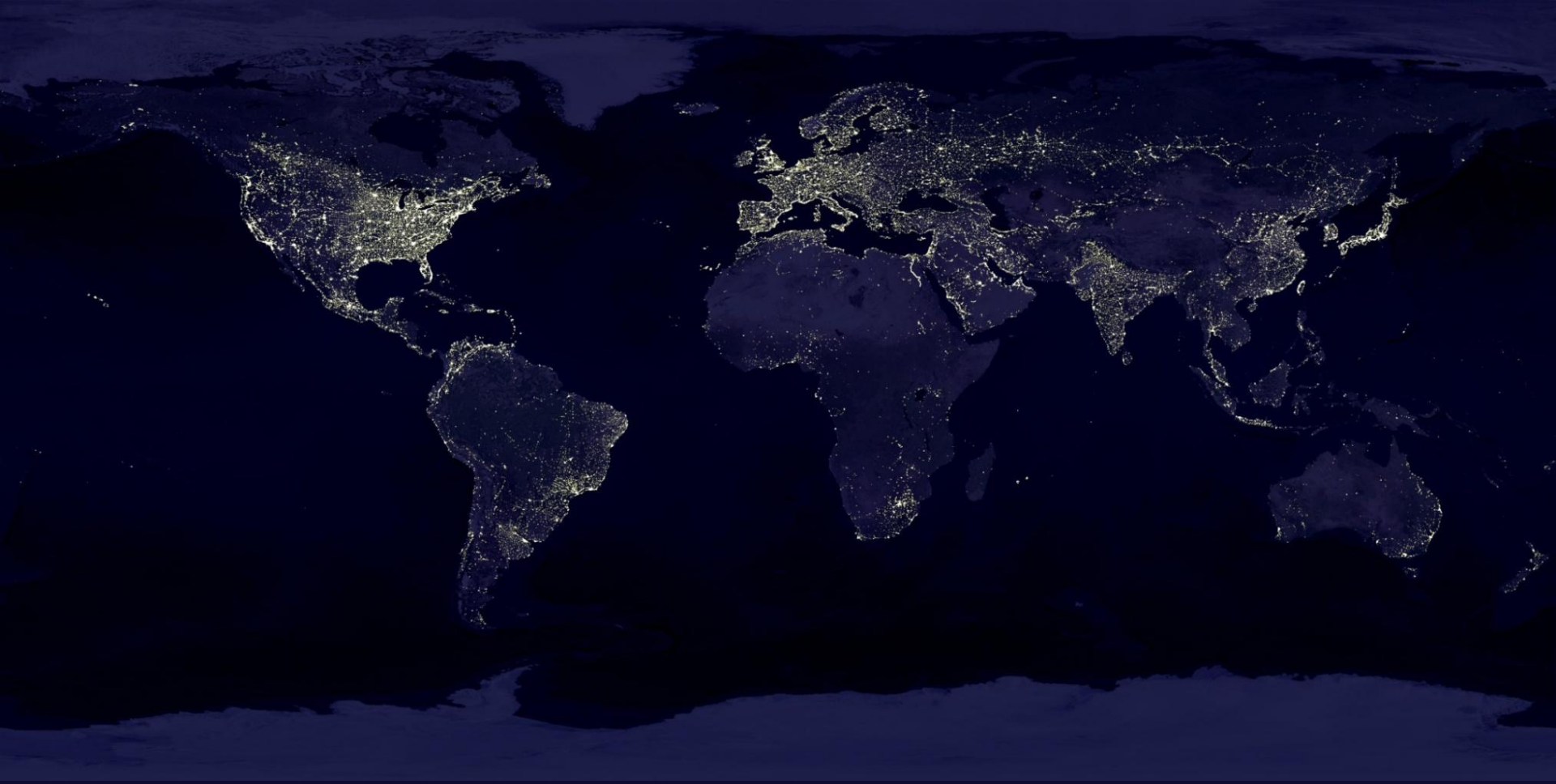


Collective Mind: community-driven systematization and automation of program optimization



Grigori Fursin

POSTALE group, INRIA,
Paris region, France

University of Tennessee, USA
May 2014

Question:

How to design next generation of faster, smaller, cheaper, more power efficient and reliable computer systems (software and hardware)?

Long term interdisciplinary vision:

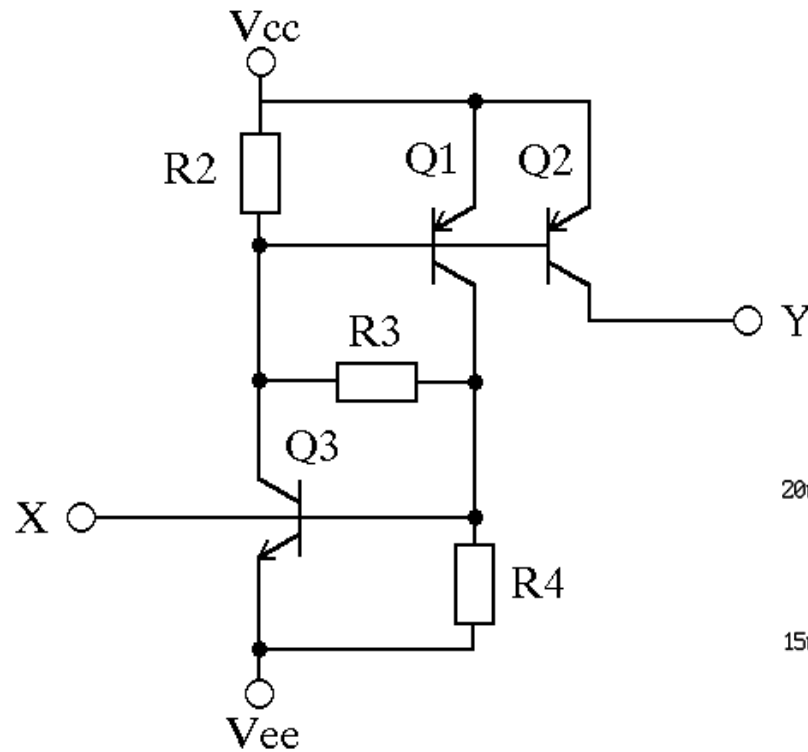
Collaborative, big-data driven, and reproducible
program optimization and hardware co-design



Continuously validated in industrial projects with
Intel, ARM, IBM, CAPS, ARC (Synopsys), STMicroelectronics

- Motivation: general problems in computer engineering
- Big-data driven program optimization and architecture co-design
- Problems with this approach
- Unifying compiler multi-objective auto-tuning
- Unifying performance modelling
- New publication model (reproducible research)
- Conclusions and future work

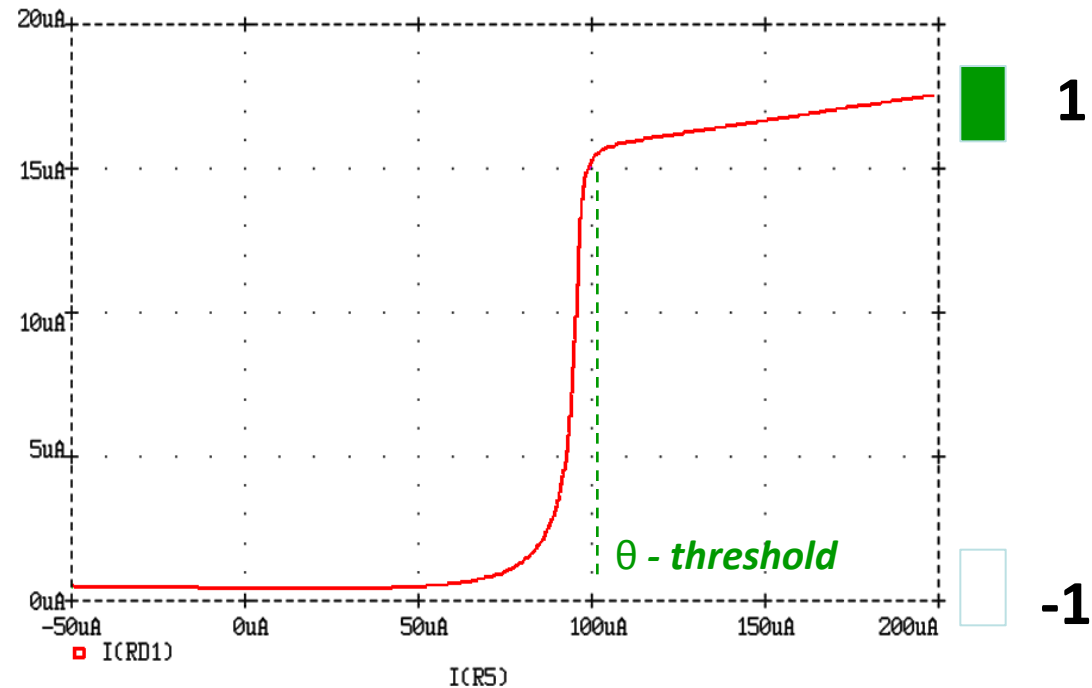
Back to 1993: sad story



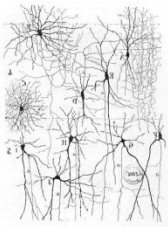
Semiconductor neural element -
base of neural accelerators
and brain-inspired computers
*Modeling and understanding
brain functions*

***Faced major problem
during modeling***

- **Too slow**
- **Unreliable**
- **Costly**



Let's tune and learn everything empirically like in physics



**End
User
task**



Available solutions

Algorithm

Application

Compilers

Binary and libraries

State of the system

Data set

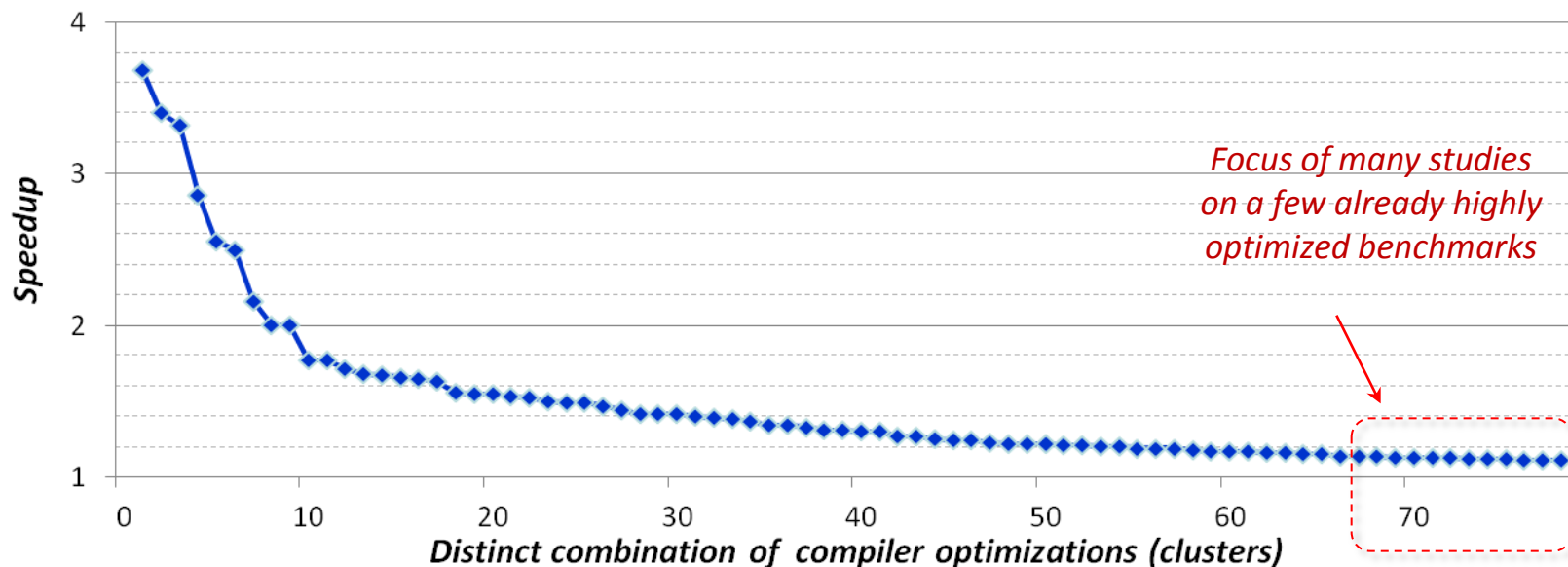
Run-time environment

Storage

Architecture

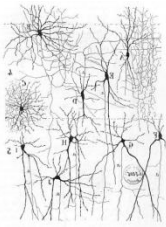
Result

From ATLAS to whole applications



Tuning 285 shared code and dataset combinations from 8 benchmarks including NAS, MiBench, SPEC2000, SPEC2006, Powerstone, UTDSP and SNU-RT

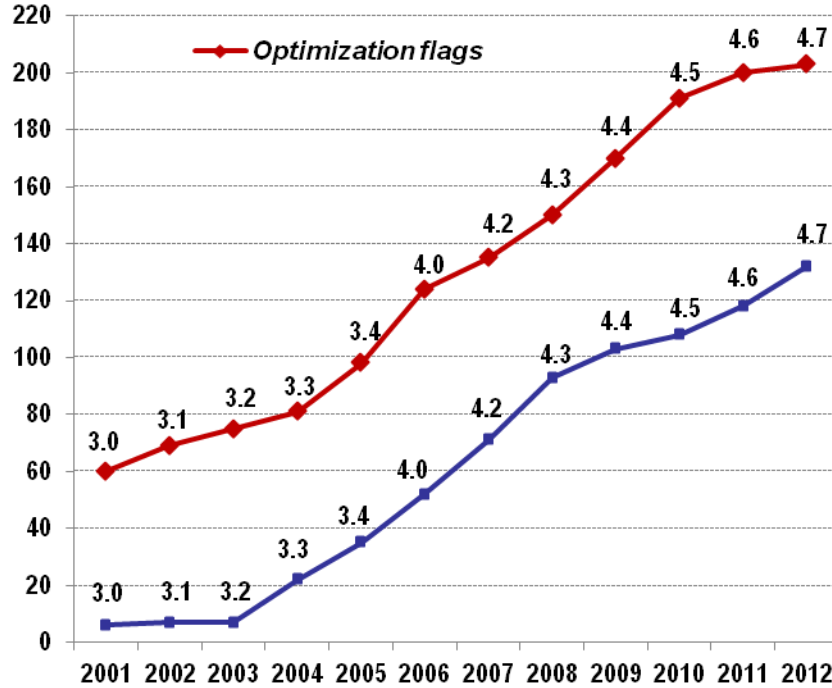
using GRID 5000; Intel E5520, 2.6MHz;
GCC 4.6.3; at least 5000 random combinations of flags



**End
User
task**



GCC optimizations



Result

Fundamental problems:

- 1) Too many design and optimization choices at all levels
- 2) Always multi-objective optimization: performance vs compilation time vs code size vs system size vs power consumption vs reliability vs return on investment
- 3) Complex relationship and interactions between ALL software and hardware components

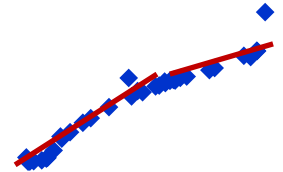
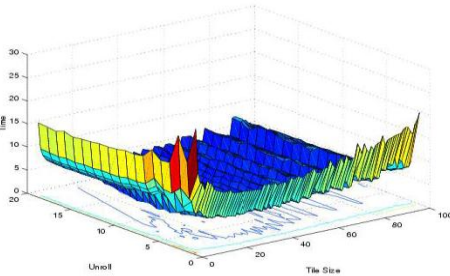
Empirical auto-tuning is too time consuming, ad-hoc and tedious to be a mainstream!

Summary of current problems

Auto-tuning, machine-learning, dynamic adaptation, co-design shows high potential for more than 2 decades but still far from the mainstream in production environments due to:

- Optimization spaces are large and non-linear with many local minima
- Exploration is slow and ad-hoc (random, genetic, some heuristics)
- Only small part of the system is taken into account (rarely reflect behavior of the whole system)
- Very limited training sets (a few benchmarks, datasets, architectures)
- Black box model doesn't help architecture or compiler designers
- Many statistical pitfalls and wrong usages of machine learning for compilation and architecture

***By the end of experiments, new tool versions are often available;
Life span of experiments and ad-hoc frameworks - end of MS or PhD project;
Researchers focus on publications rather than practical and reproducible solutions***



Can we crowdsource auto-tuning?

Got stuck with a limited number of benchmarks, datasets, architectures and a large number of optimizations and generated data;
could not validate data mining and machine learning techniques

Needed dramatically new approach!

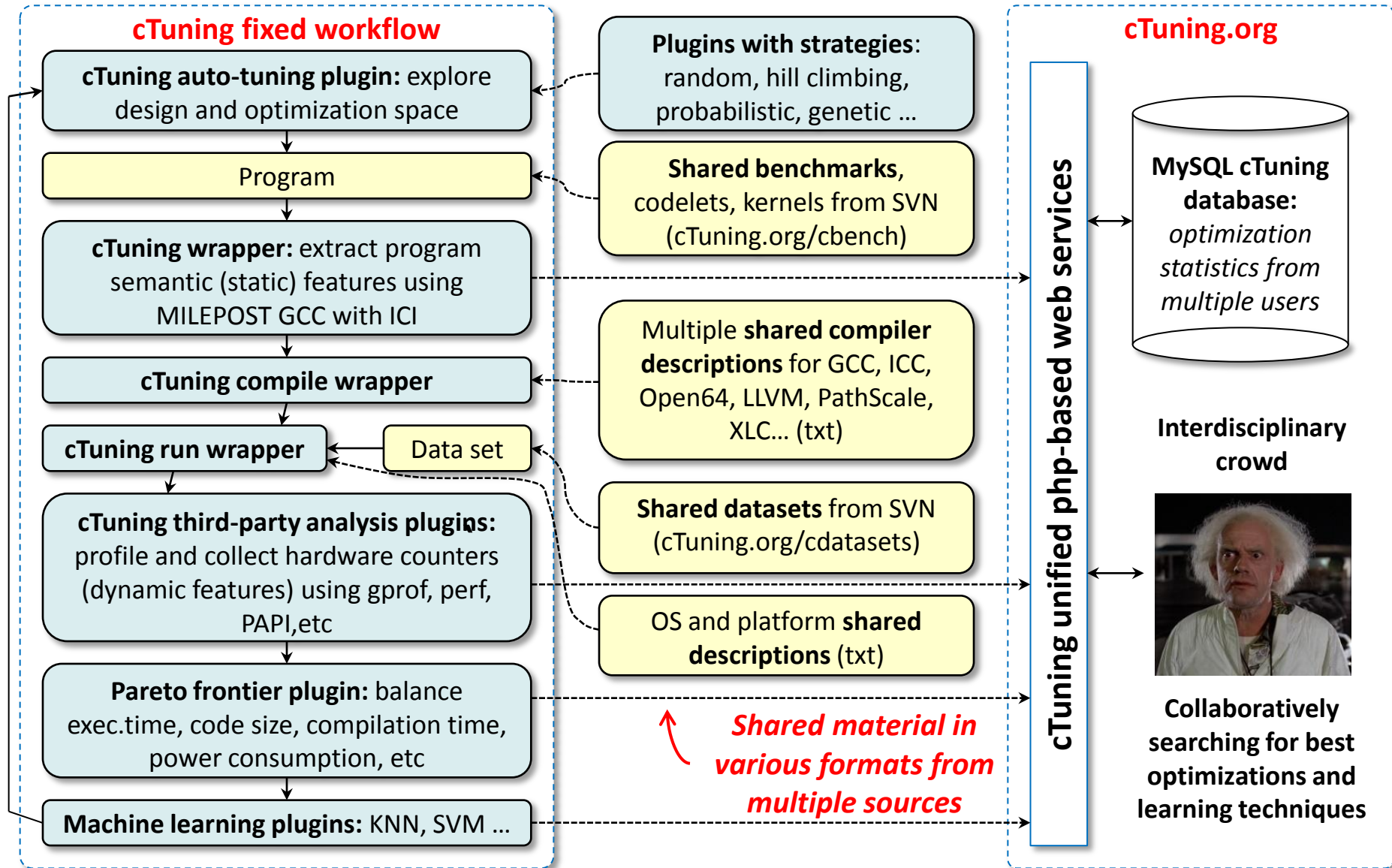
Millions of users run realistic applications on different architectures with different datasets, run-time systems, compilers, optimizations!



Can we leverage their experience and computational resources?

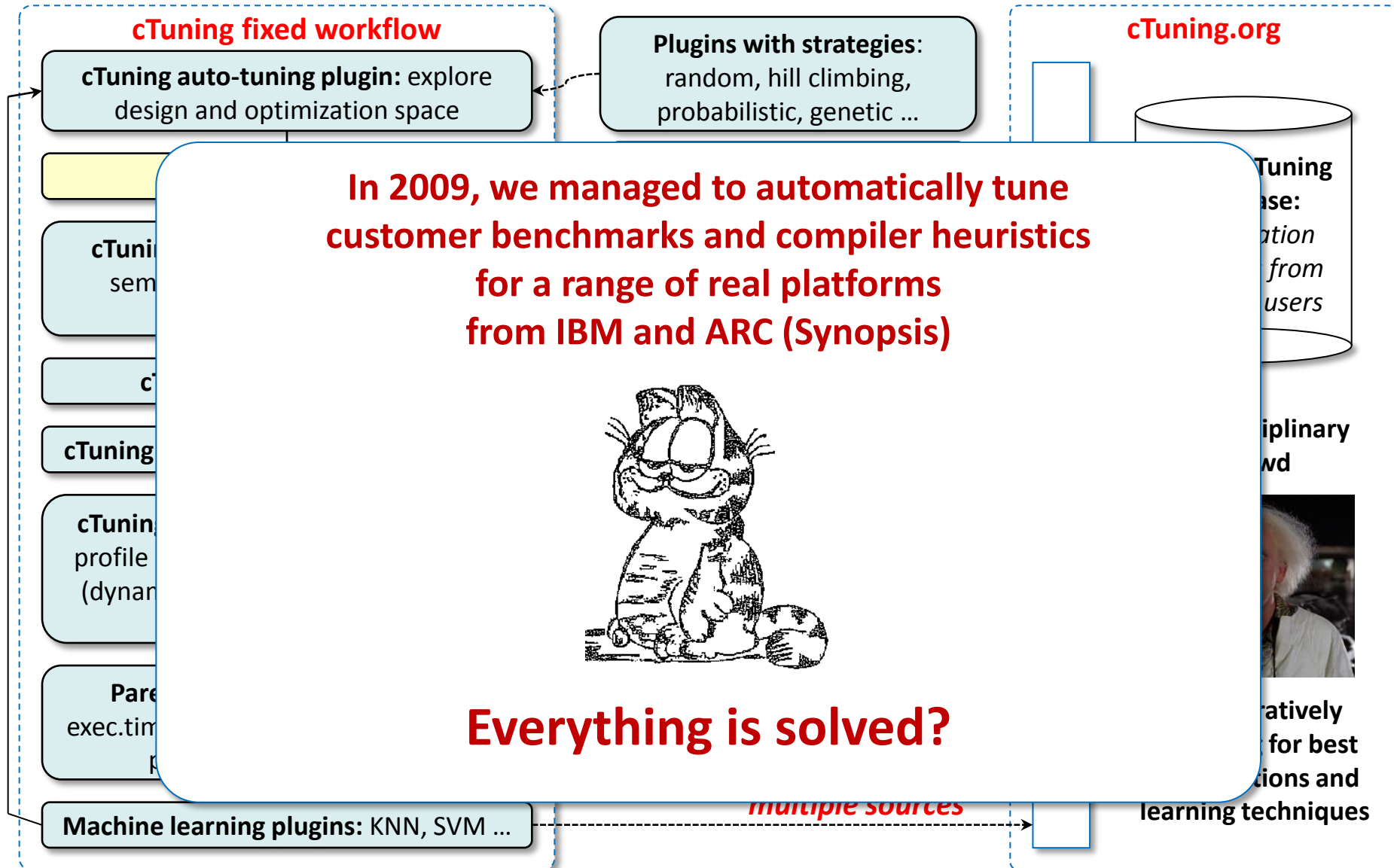
Can we connect disjoint analysis, tuning, learning tools together with a public repository of knowledge?

Easy: Plugin-based auto-tuning and learning (1999-2009)



Grigori Fursin and Olivier Temam, Collective Optimization: a practical collaborative approach, ACM TACO 7(4), 2010

Easy: Plugin-based auto-tuning and learning (1999-2009)

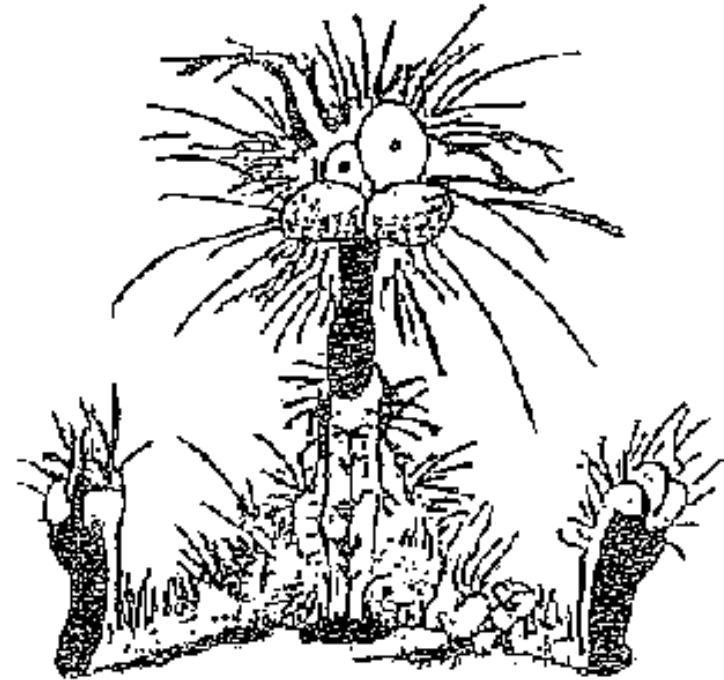


Common open-source cTuning CC framework (<http://cTuning.org/ctuning-cc>)

We also experienced a few problems

Technological chaos

LLVM 3.4 AVX
GCC 4.8.x GCC 4.1.x ATLAS
frequency LLVM 2.8 hardware perf
genetic CUDA 5.x function counters
algorithms ICC 11.1 level pass
ARM v8 MPI OpenCL reordering
OpenMP prof gprof
reliability CUDA 4.x Intel SandyBridge
GCC 4.2.x per phase
ARM v6 LLVM 3.0 reconfiguration
GCC 4.5.x MVS 2013 HMPP memory size
execution time transformations XLC
GCC 4.3.x ICC 12.0 KNN
SSE4 predictive Scalasca
scheduling ICC 11.0
MKL bandwidth GCC 4.4.x GCC 4.6.x
LLVM 2.6 SimpleScalar Codelet PAPI
LLVM 2.9 ICC 12.1 Testarossa Amplifier Jikes
process HDD size Open64
oprofile TLB loop-level ISA
GCC 4.7.x LLVM 2.7 SVM VTune
threads LTO ICC 10.1 threads
Phoenix program-level algorithm- TAU
cache size level level IPA TBB
algorithm precision

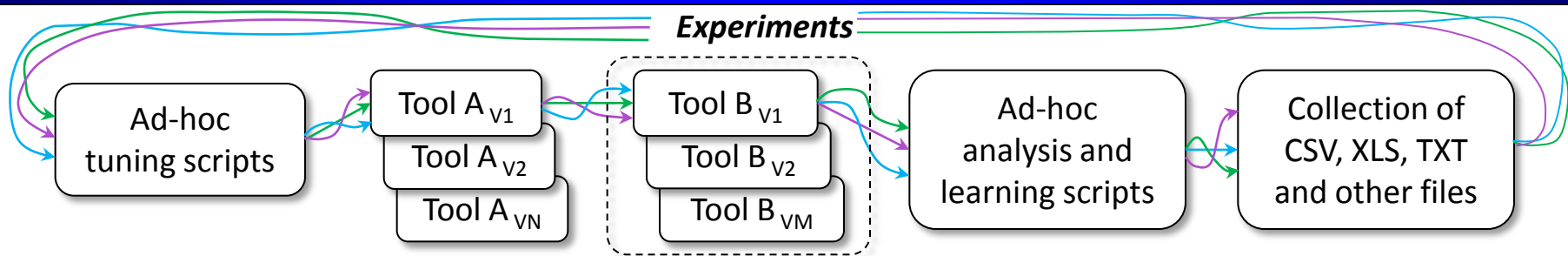


Major problems:
reproducibility, extensibility and scalability

**Many thanks to international academic
and industrial users for their feedback**

<http://cTuning.org/lab/people>

Collective Mind: towards systematic and reproducible experimentation



Hardwired experimental setups, very difficult to change, scale or share

Behavior

Choices

Features

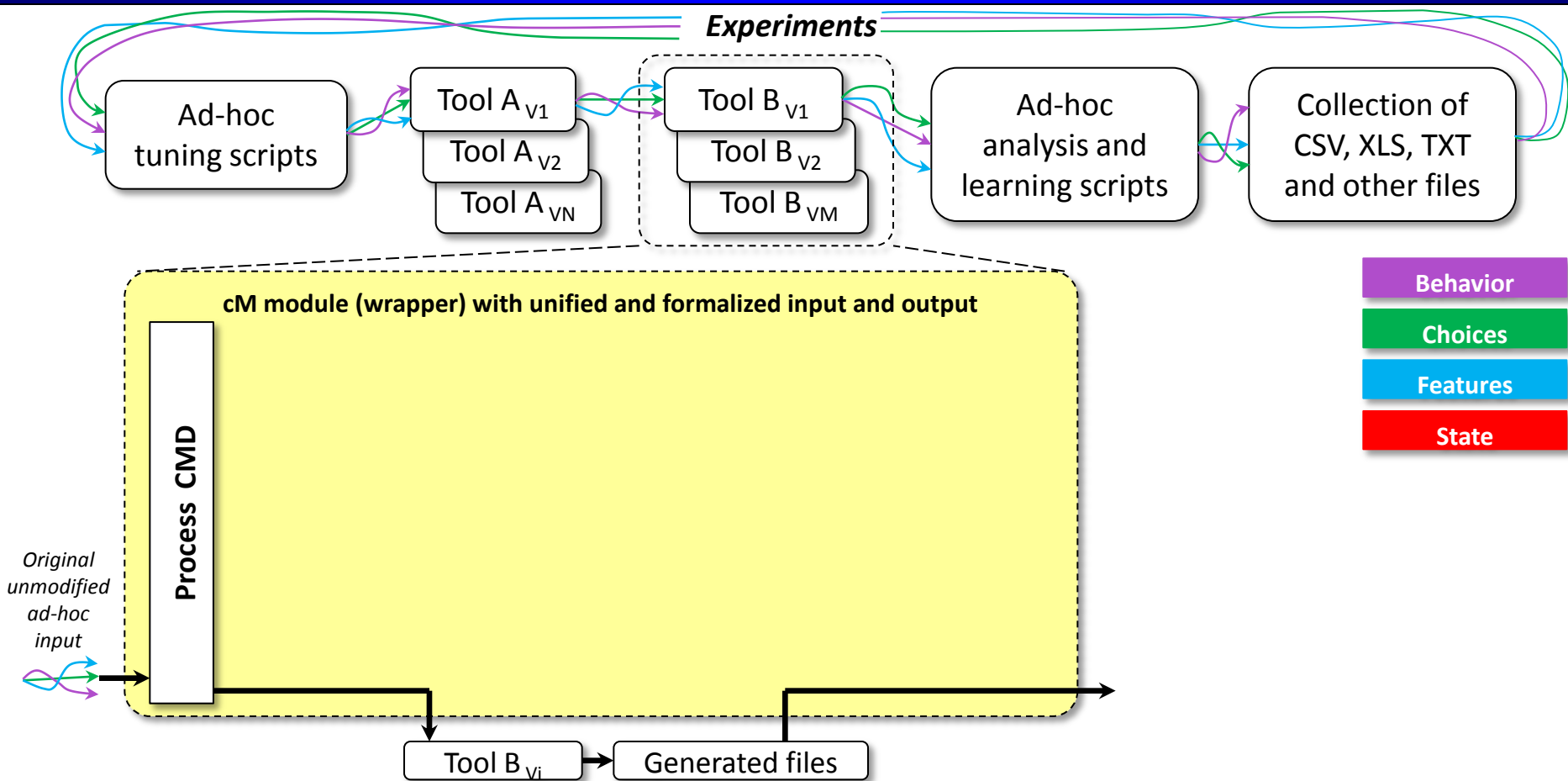
State

Motivation for Collective Mind (cM):

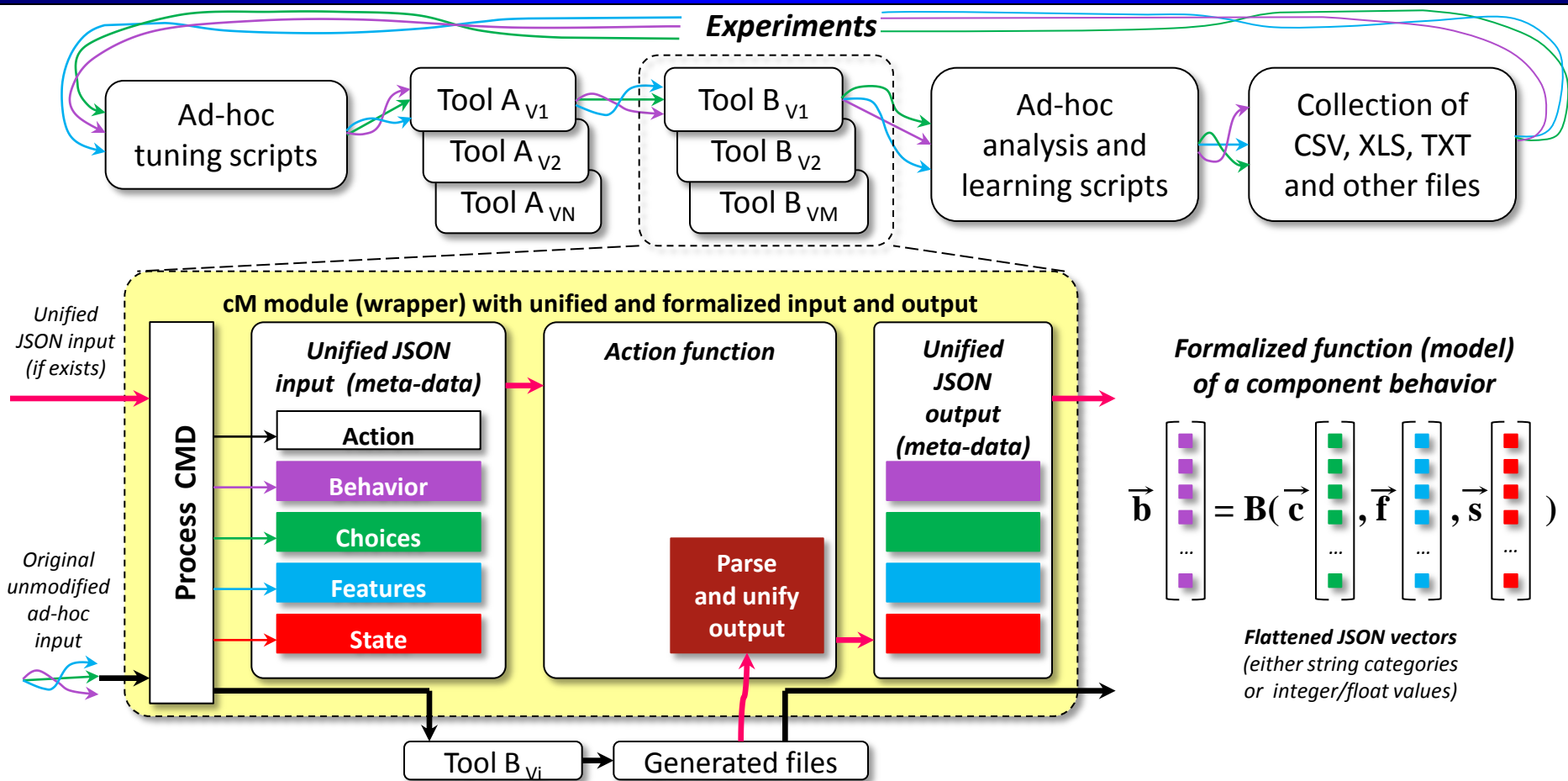
- How to preserve, share and reuse practical knowledge and experience and program optimization and hardware co-design?
- How to make auto-tuning, machine learning and run-time adaptation practical?
- How to ensure reproducibility of experimental results?

Meta description that should be exposed in the information flow for auto-tuning and machine learning

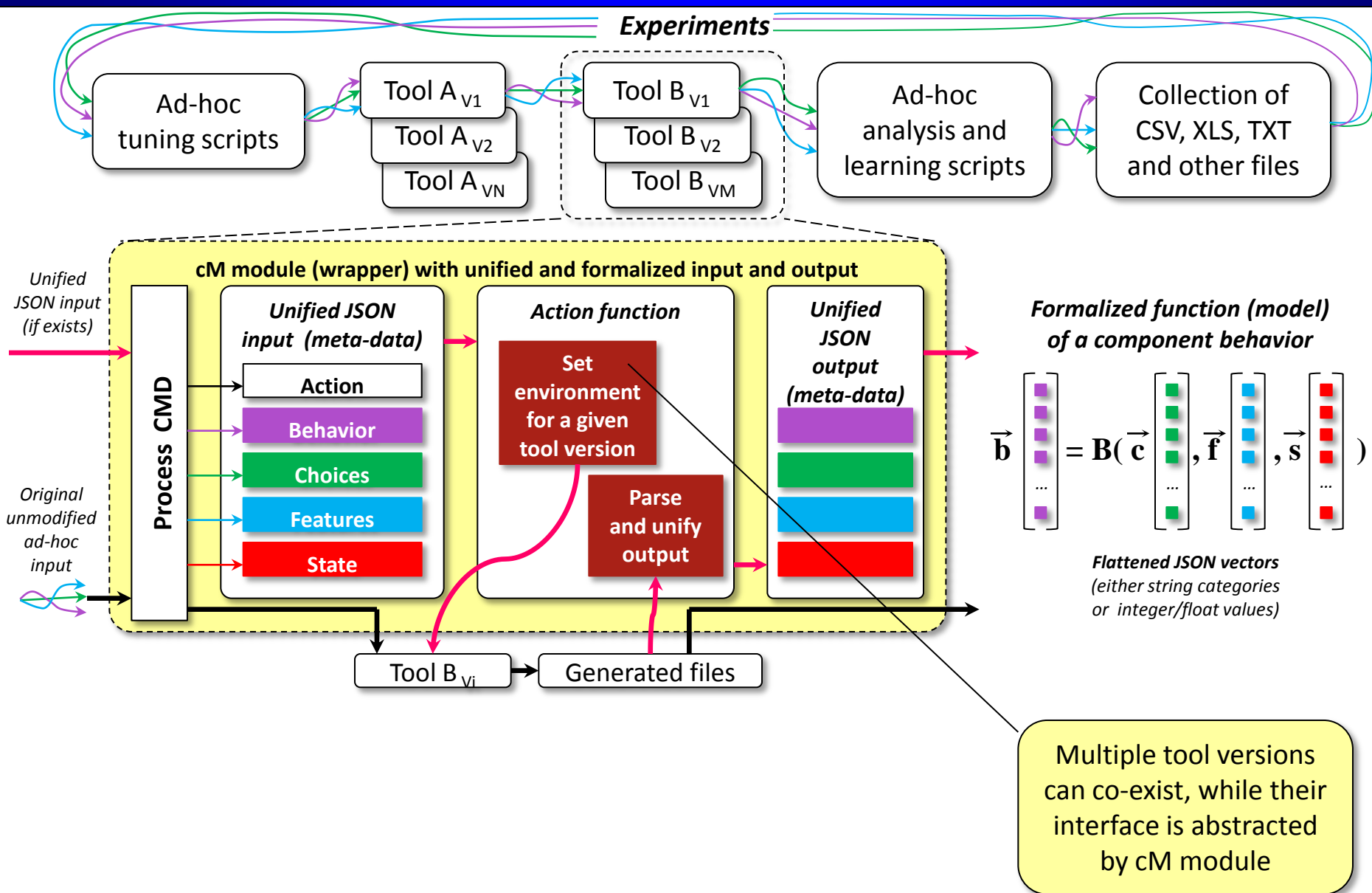
Collective Mind: towards systematic and reproducible experimentation



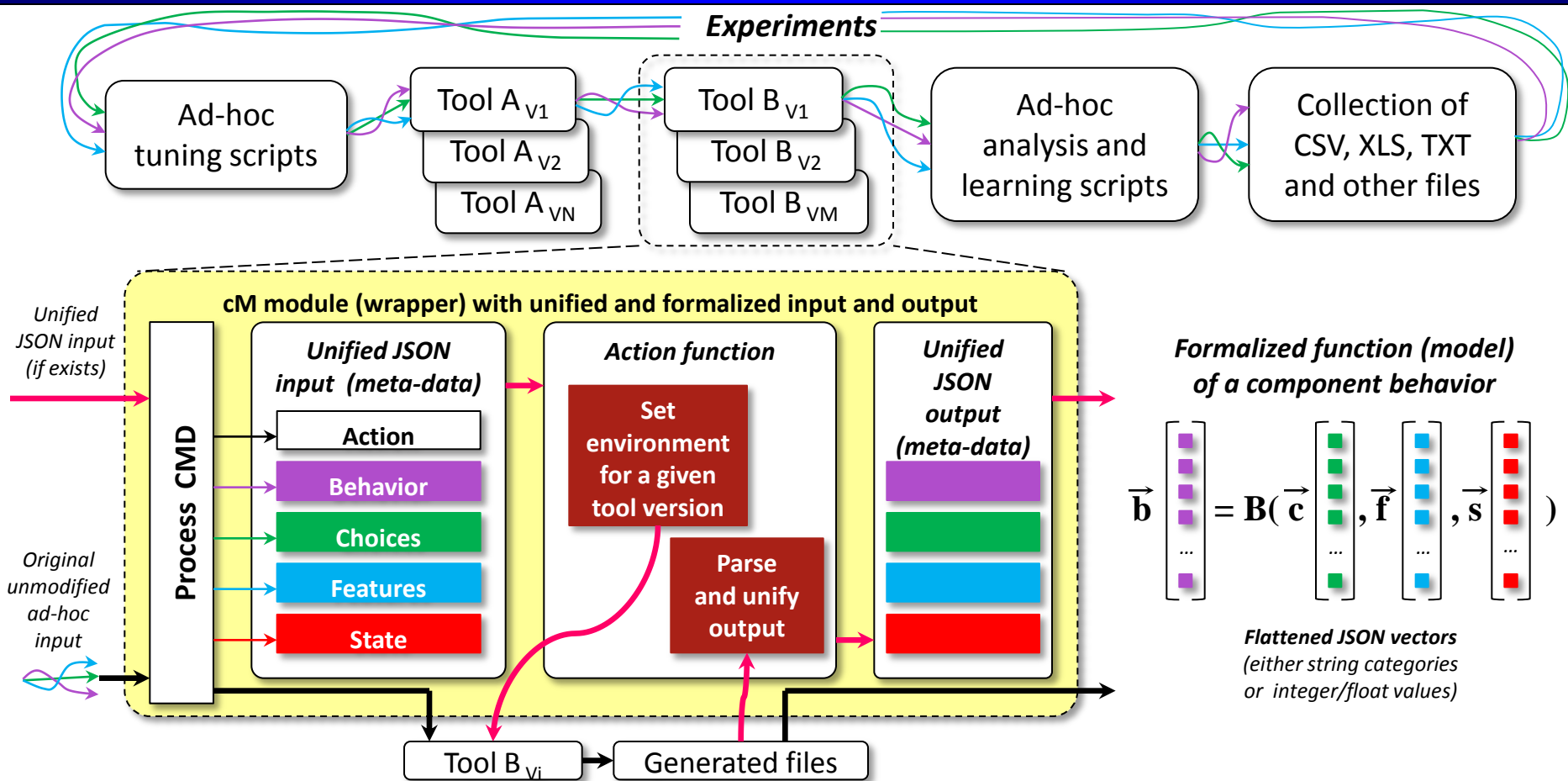
Collective Mind: towards systematic and reproducible experimentation



Collective Mind: towards systematic and reproducible experimentation



Collective Mind: towards systematic and reproducible experimentation



cm **[module name]** **[action]** (param₁=value₁ param₂=value₂ ... -- unparsed command line)

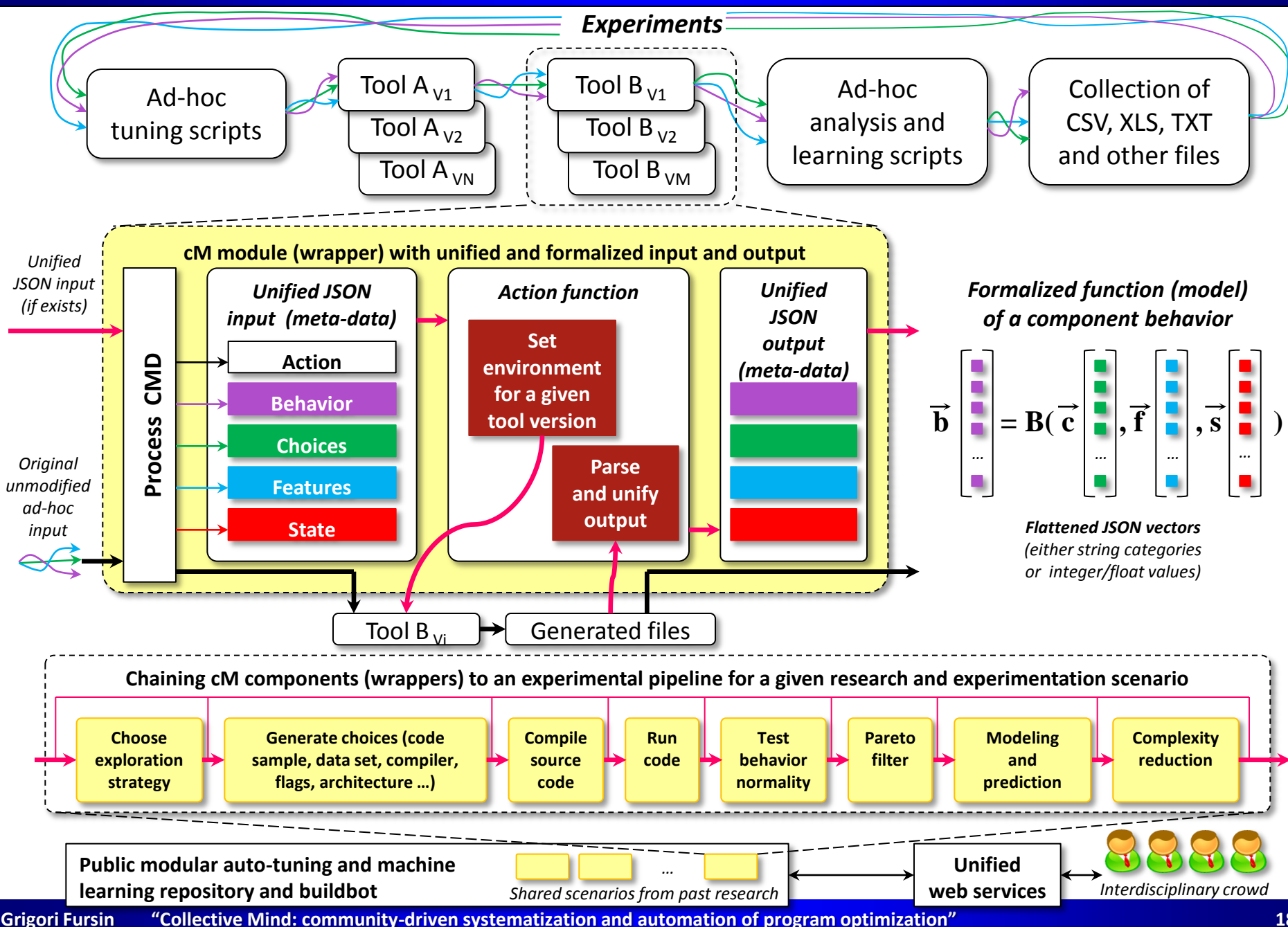
cm **compiler** **build** -- icc -fast *.c

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

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)!

Assembling, preserving, sharing and extending the whole pipeline as "LEGO"



Data abstraction in Collective Mind

cM repository directory structure:

cM module

compiler

package

dataset

module

Files, directories

GCC 4.7.1 bin

GCC 4.7.1 source

LLVM 3.4

gmp 5.0.5

mpfr 3.1.0

lapack 2.3.0

java apache commons codec 1.7

image-jpeg-0001

bzip2-0006

txt-0012

compiler

package

dataset

JSON meta-description

GCC 4.4.4

GCC 4.7.1

LLVM 3.1

LLVM 3.4

...

...

...

...

...

...

...

...

...

...

...

...

...

Compiler
flags

Installation
info

Features

Actions

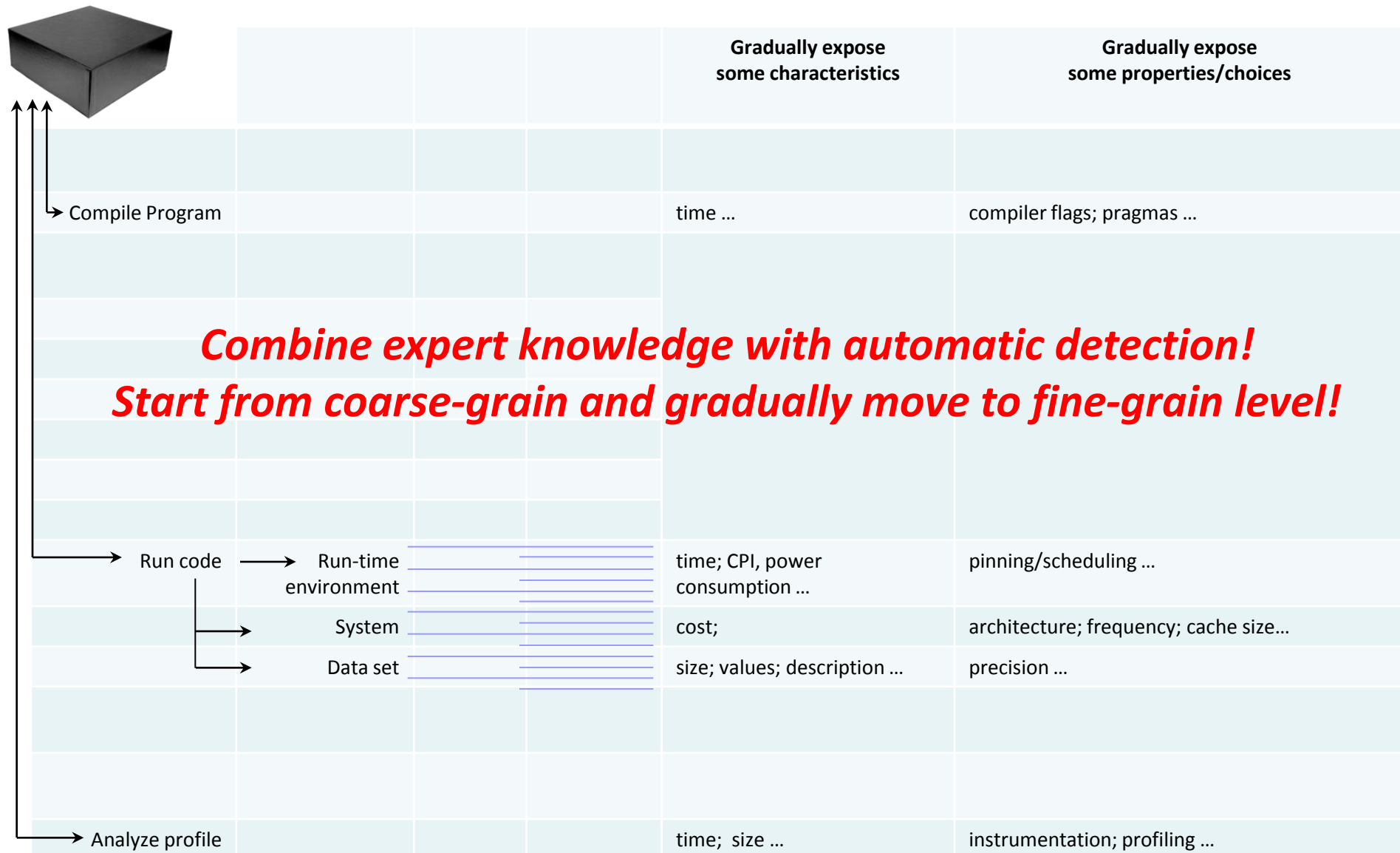
.cmr

/ module UOA

/ data UOA (UID or alias)

/ .cm / data.json

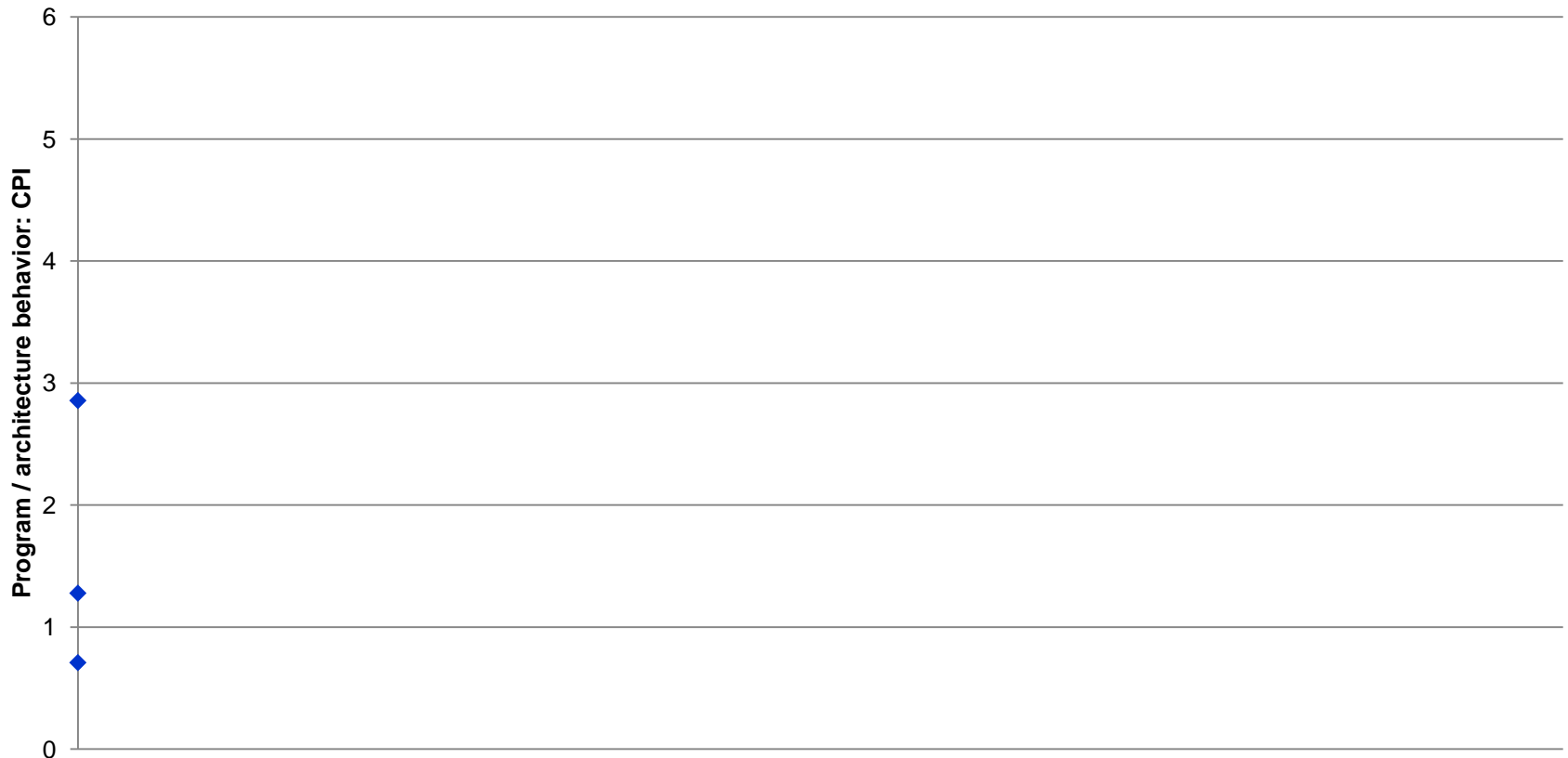
Example of collaborative learning of computer systems' behavior



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

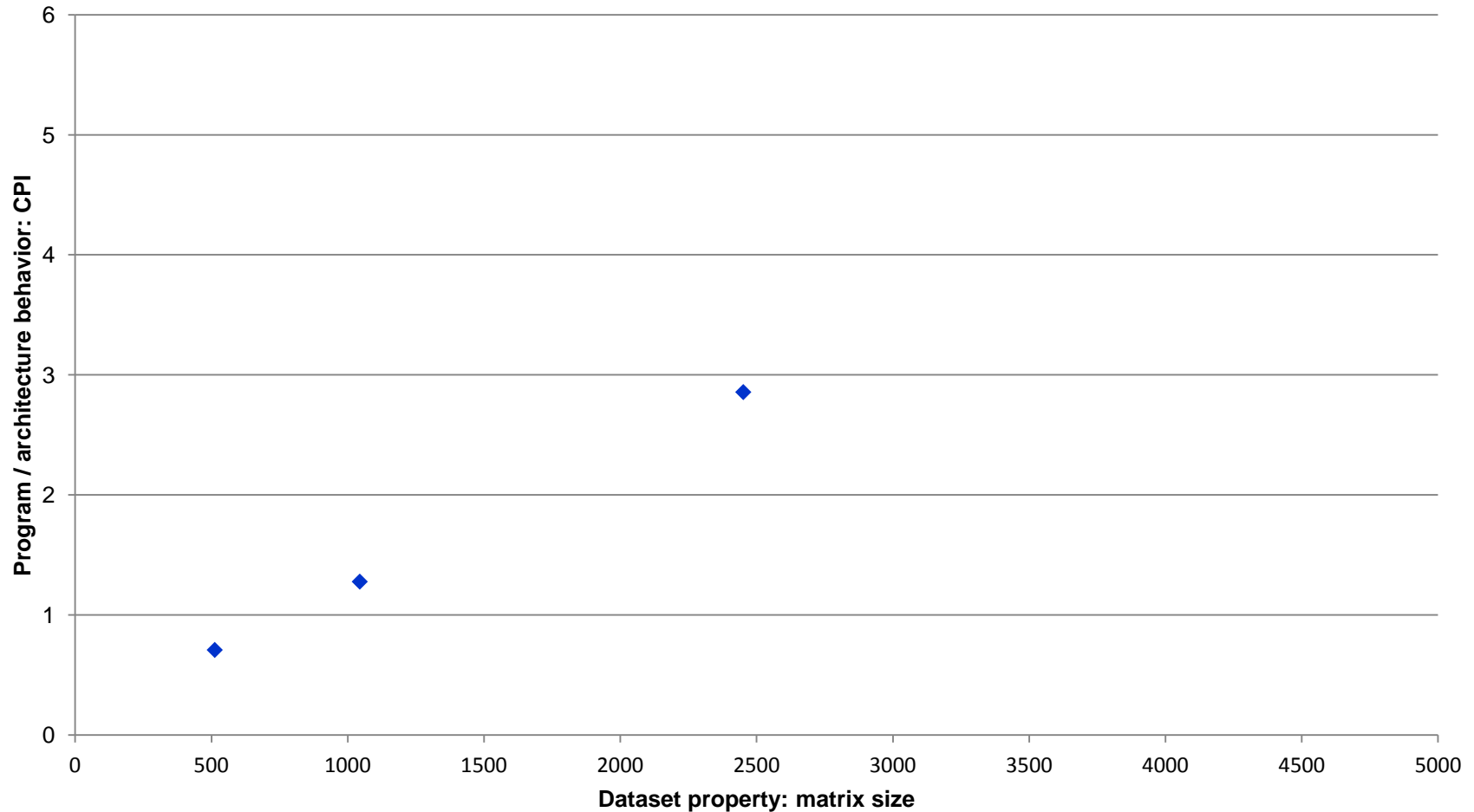
Example of characterizing/explaining behavior of computer systems

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

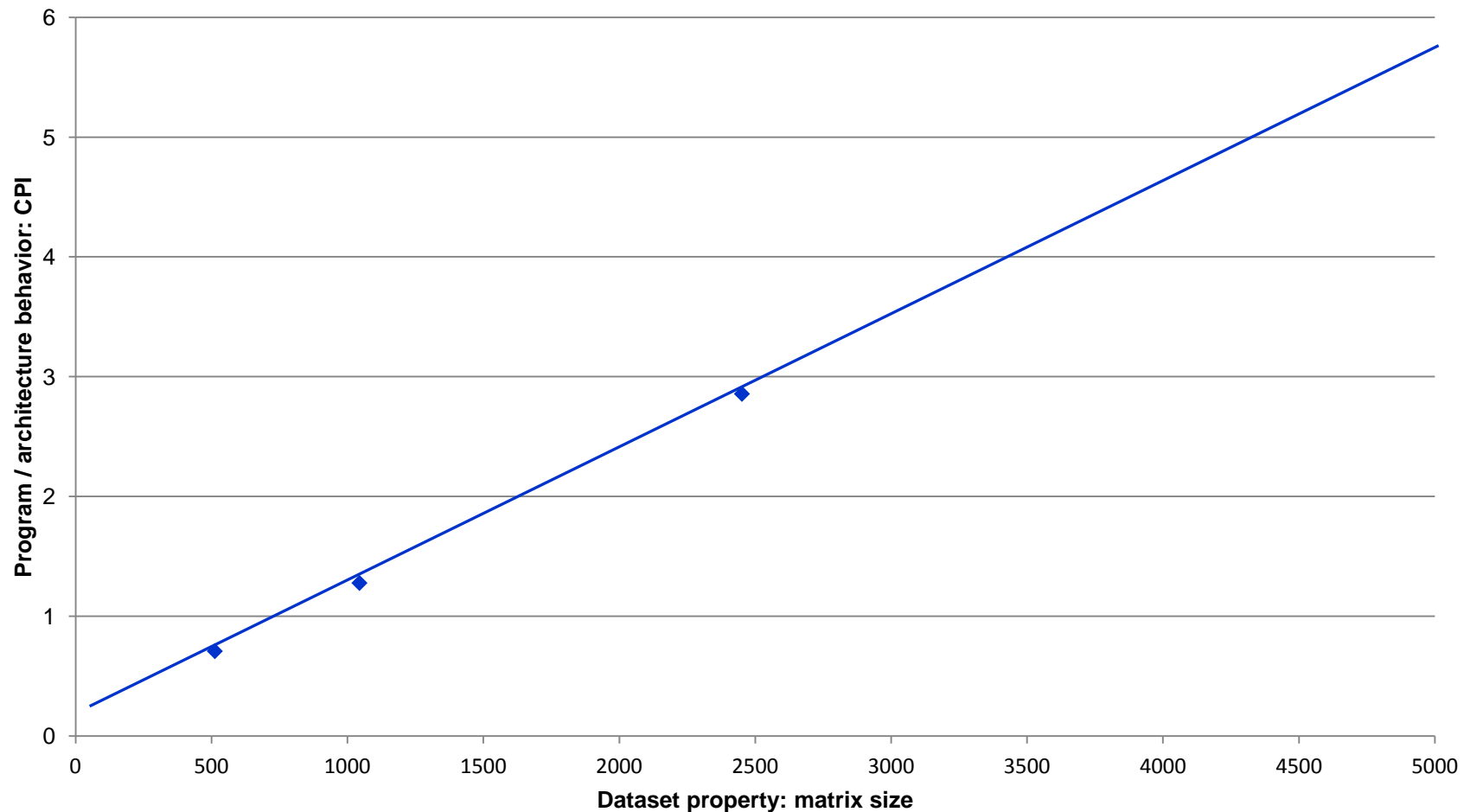
Add 1 property: **matrix size**



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)

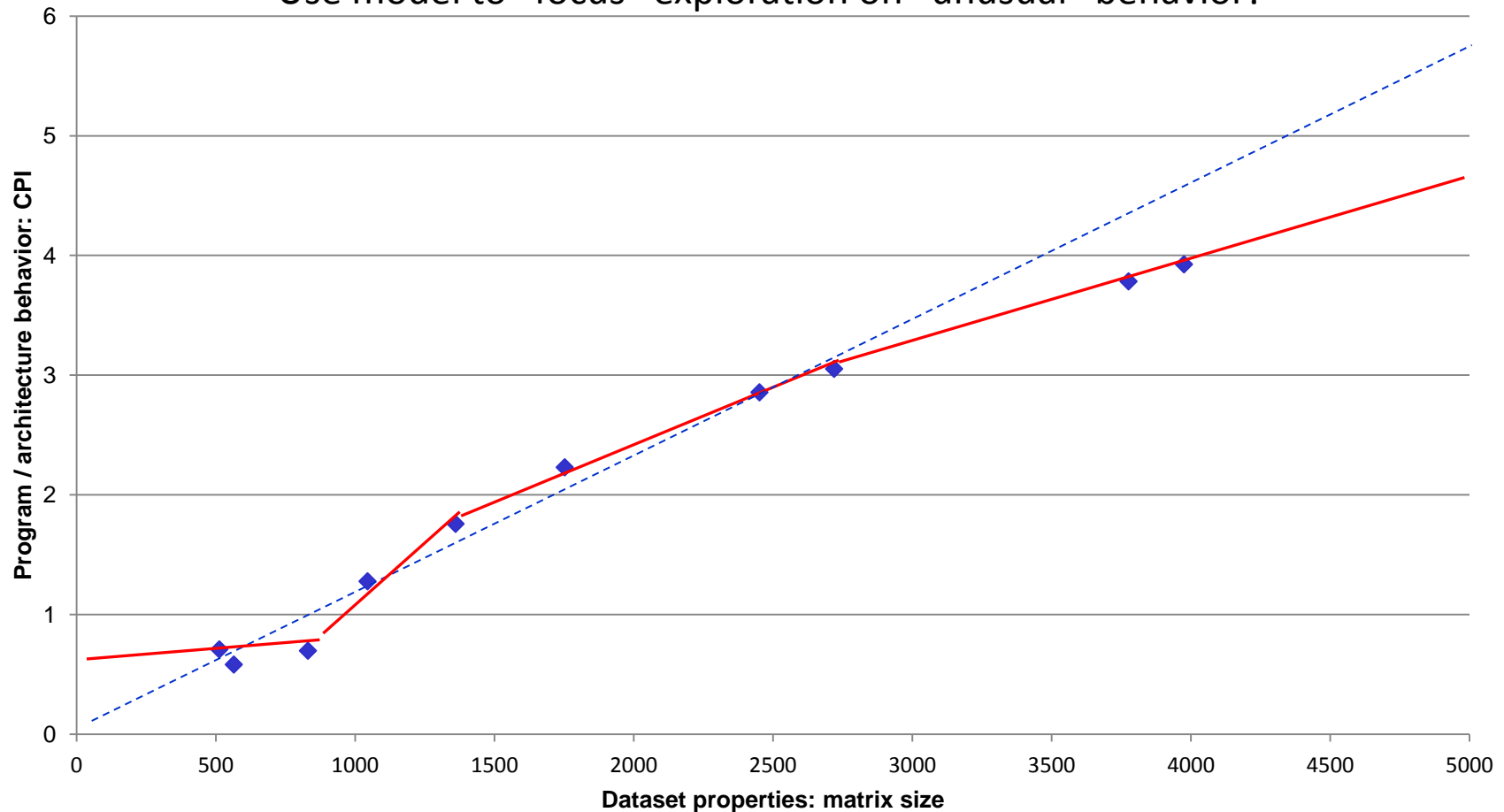


Example of characterizing/explaining behavior of computer systems

If more observations, **validate model** and **detect discrepancies!**

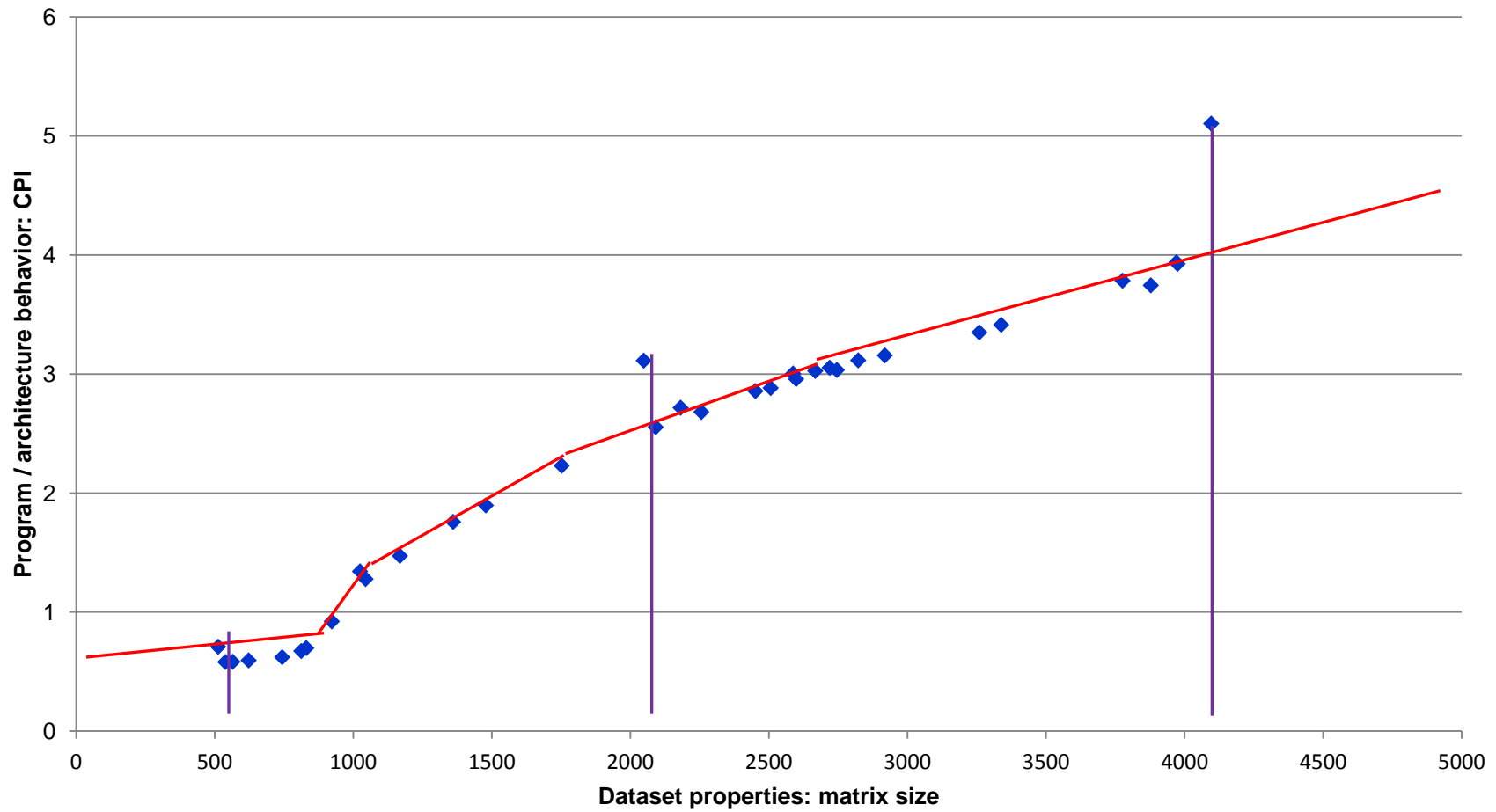
Continuously retrain models to fit new data!

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



Example of characterizing/explaining behavior of computer systems

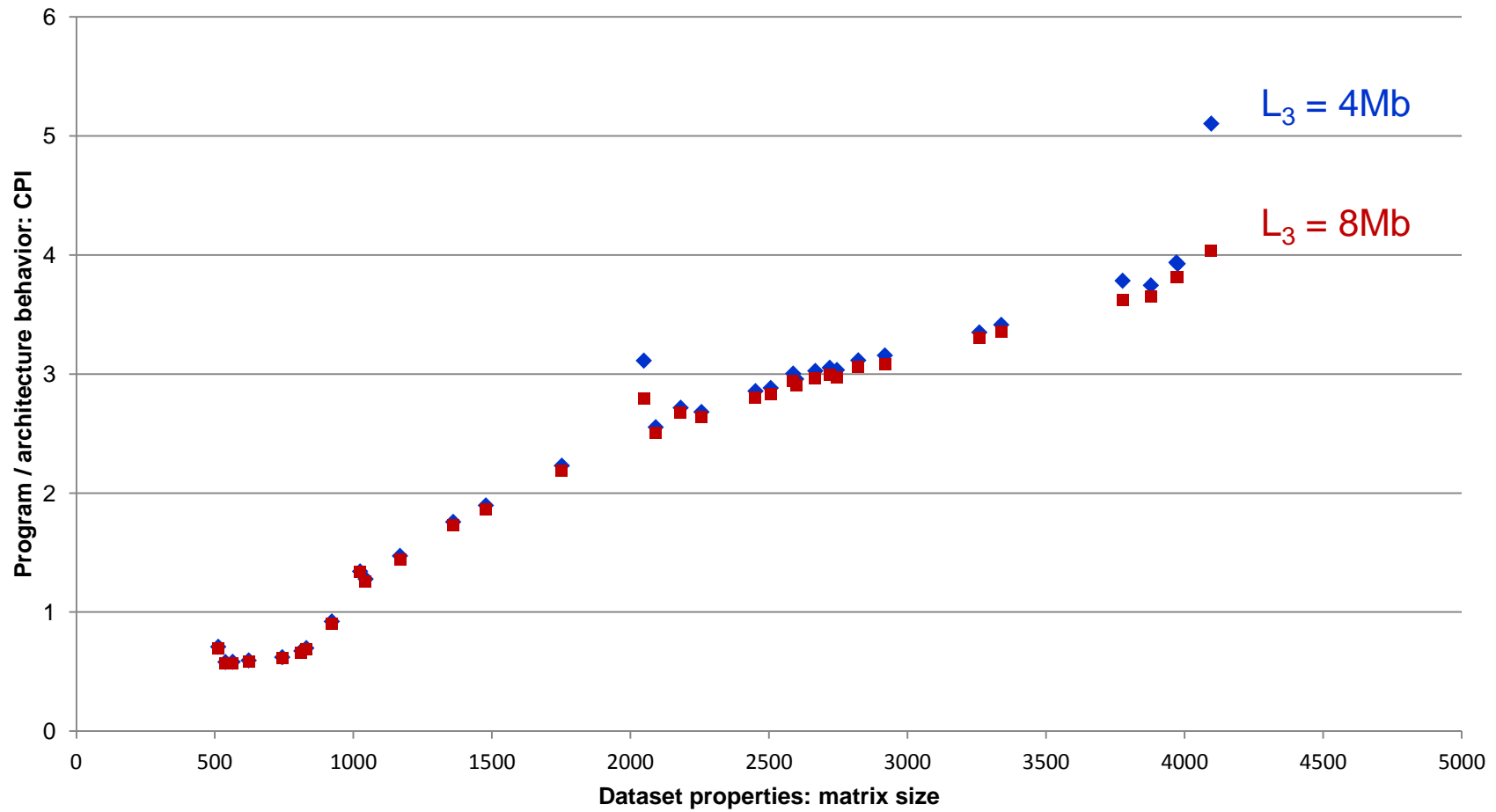
Gradually increase model complexity if needed ([hierarchical modeling](#)).
For example, detect [fine-grain effects](#) ([singularities](#)) and [characterize](#) them.



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.

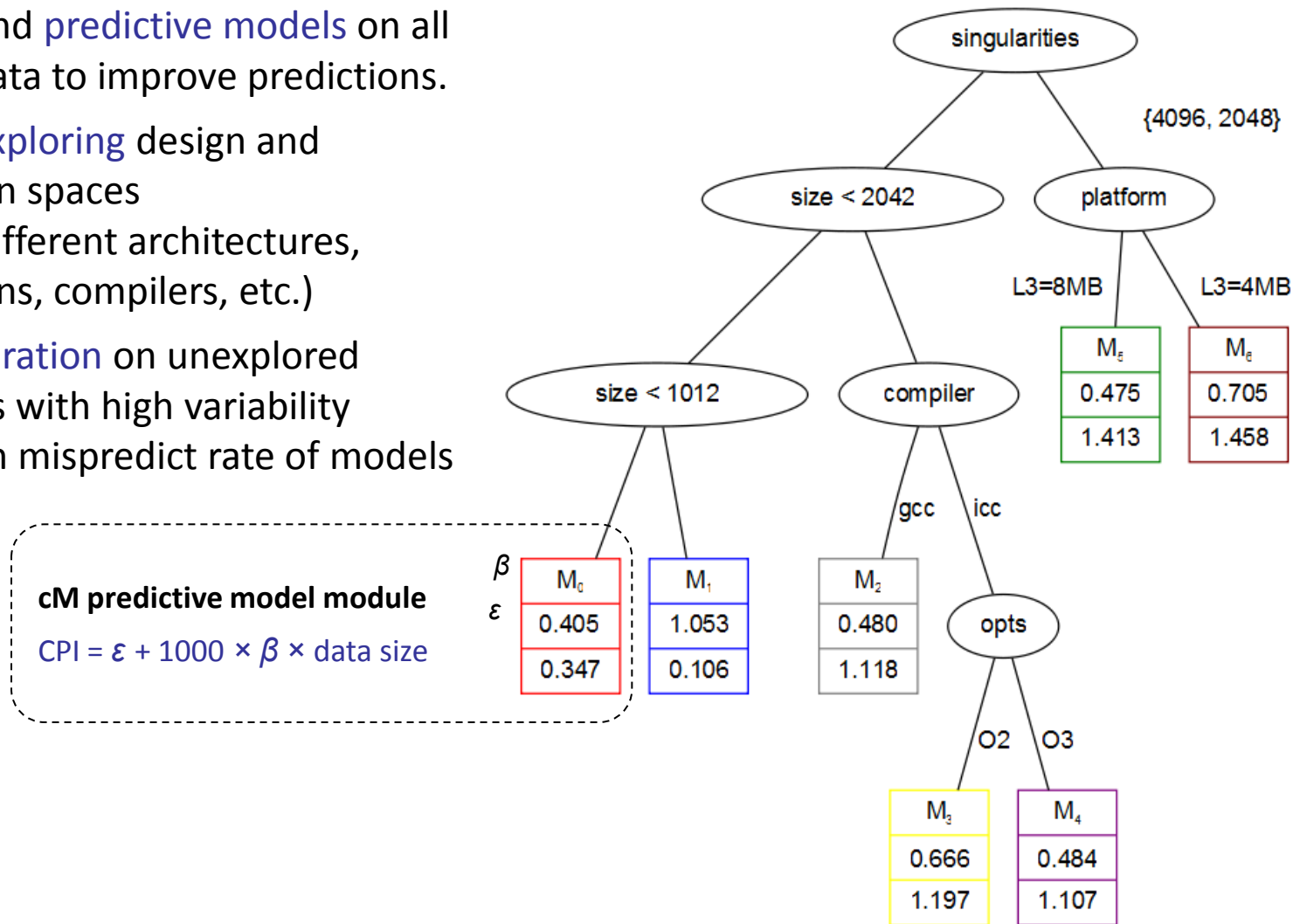


Example of characterizing/explaining behavior of computer systems

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

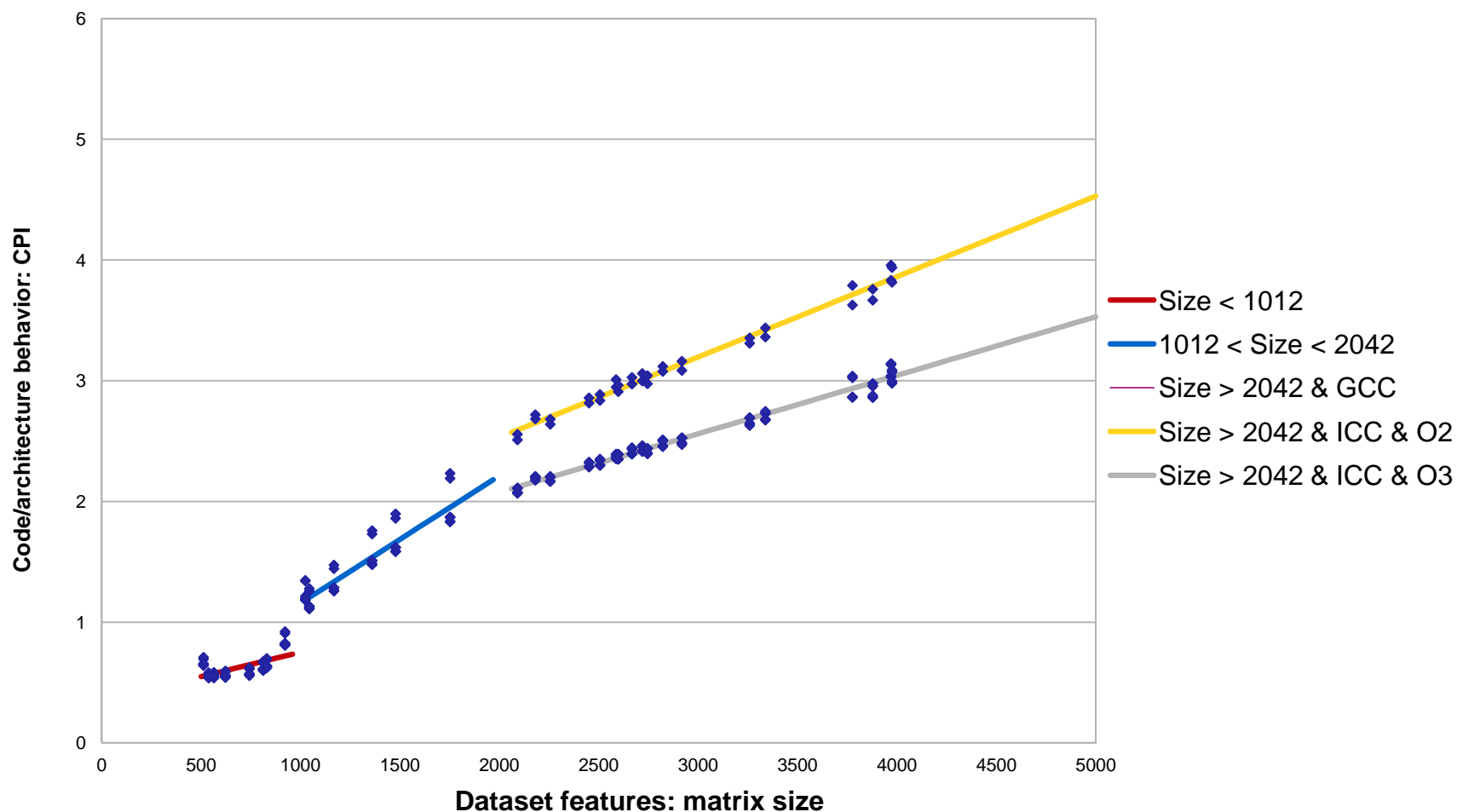


Model optimization and data compaction

Optimize decision tree (many different algorithms)

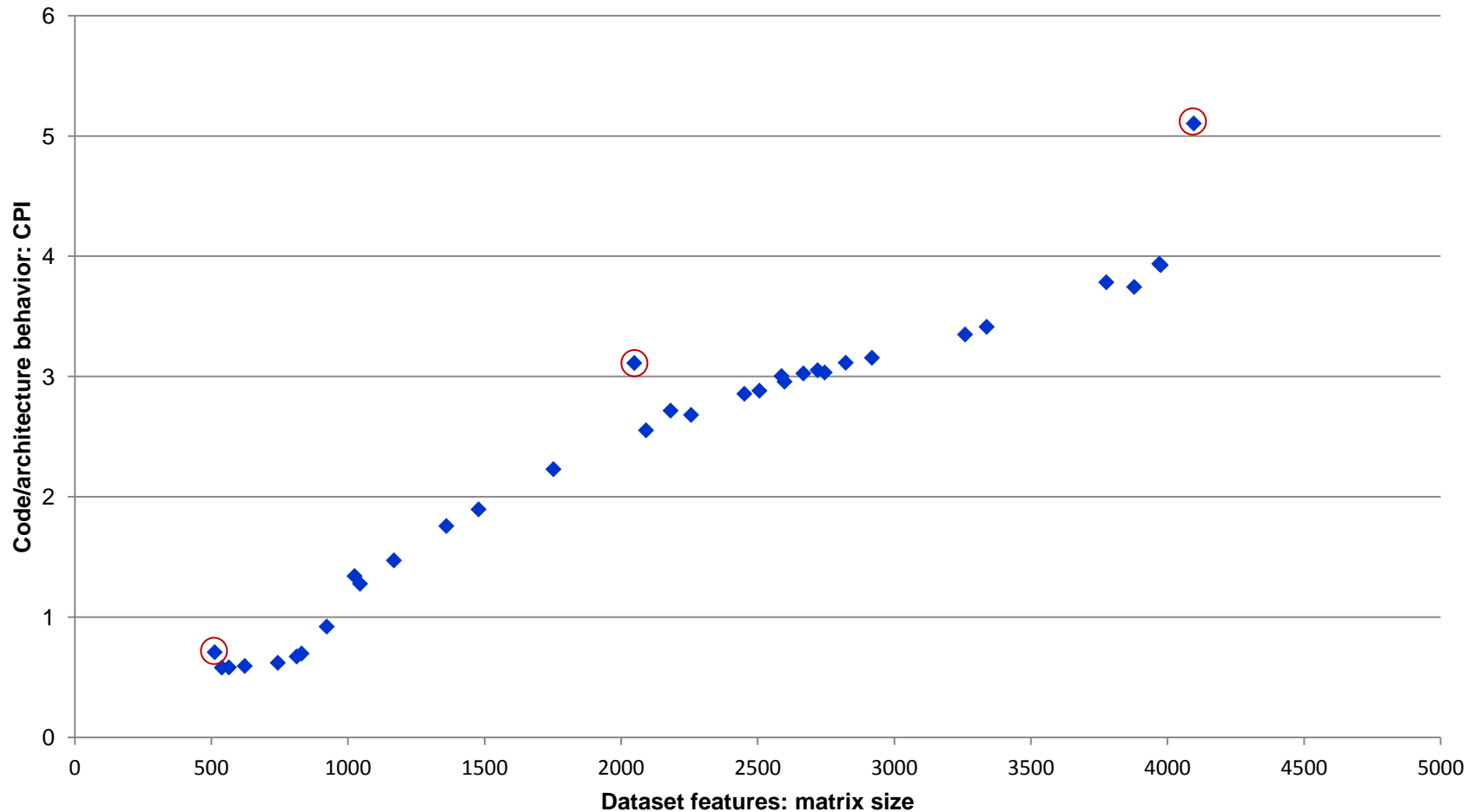
Balance precision vs cost of modeling = ROI (coarse-grain vs fine-grain effects)

Compact data on-line before sharing with other users!



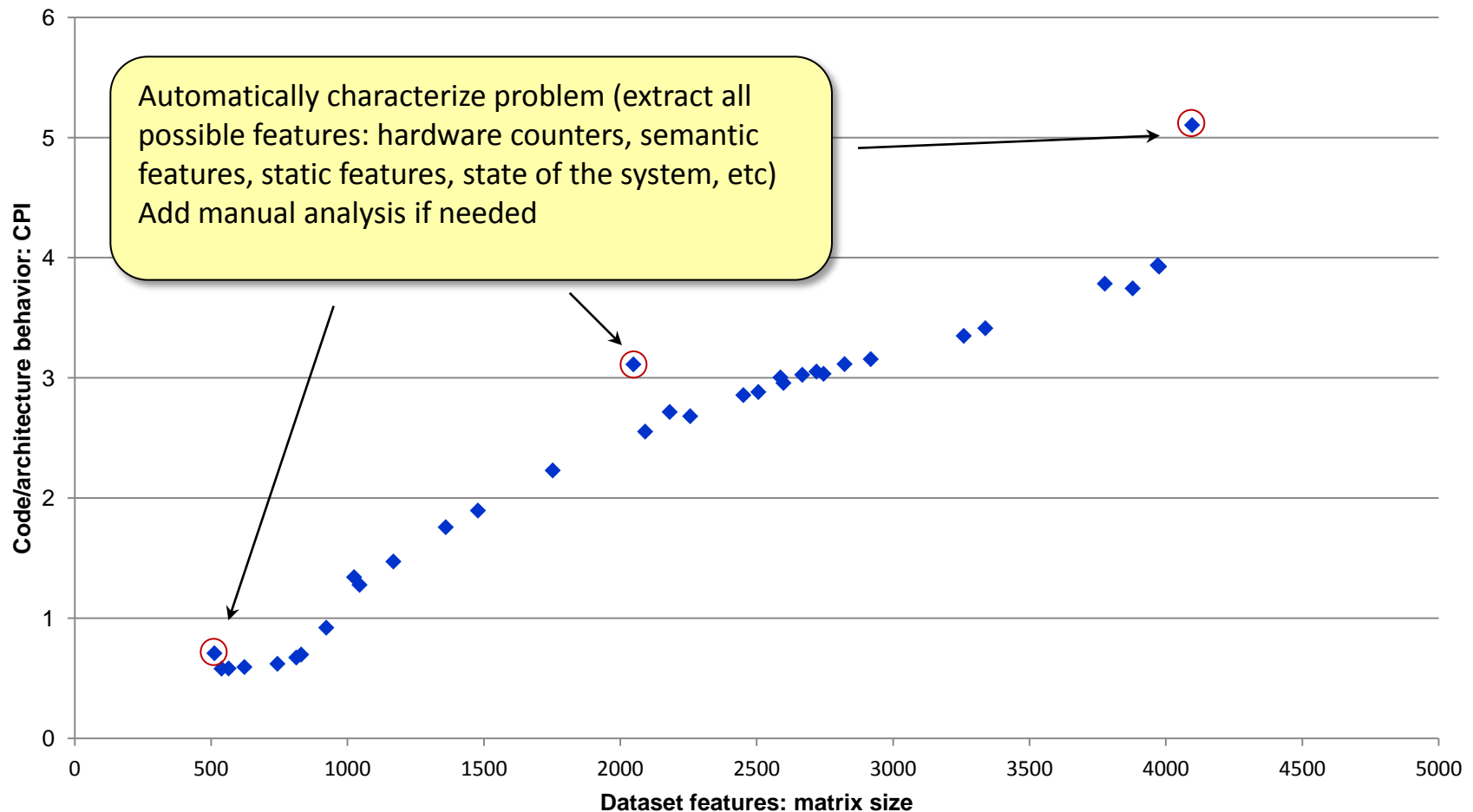
Extensible and collaborative advice system

Collaboratively and continuously add [expert advices](#) or [automatic optimizations](#).



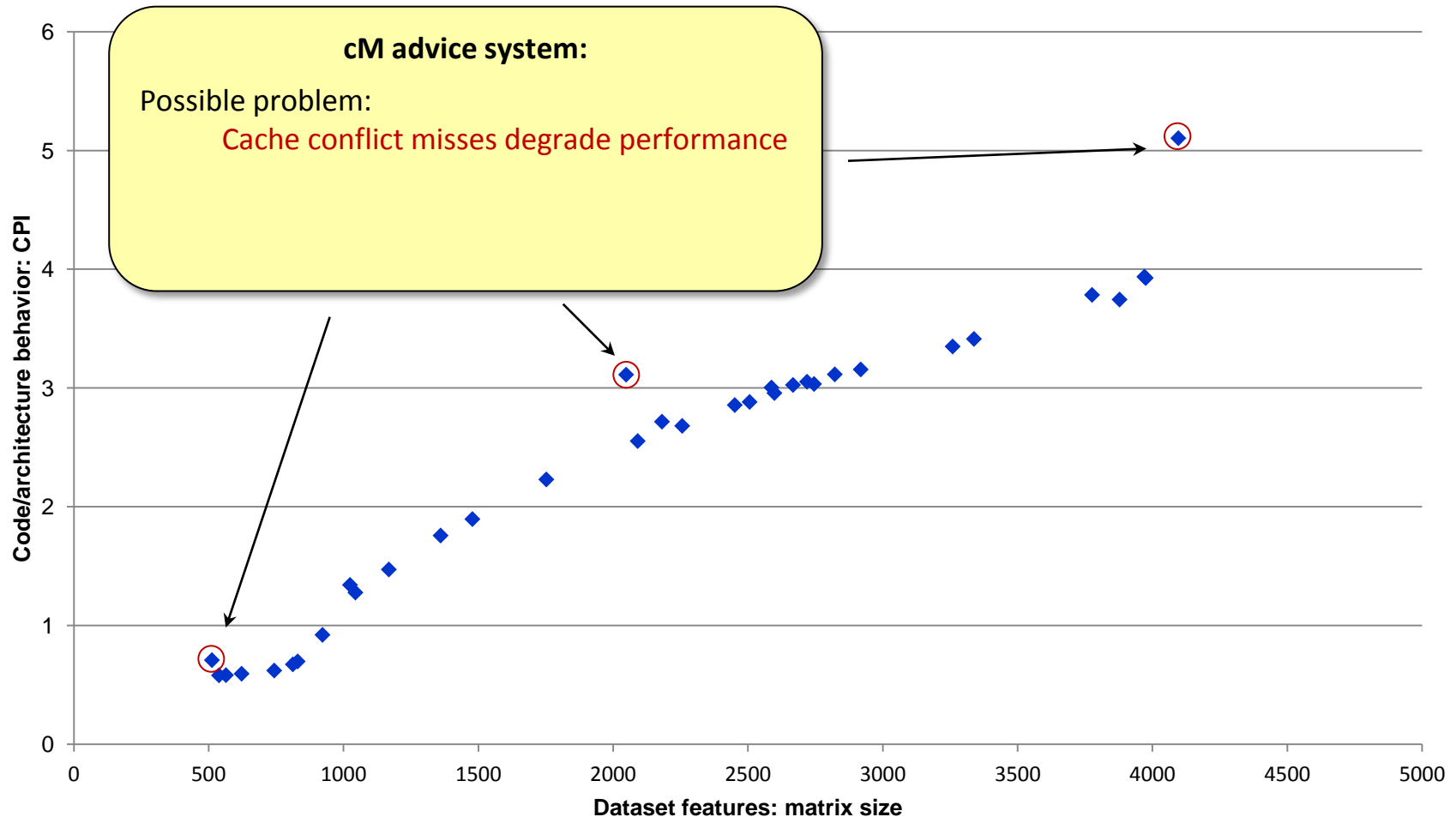
Extensible and collaborative advice system

Collaboratively and continuously add **expert advices** or **automatic optimizations**.



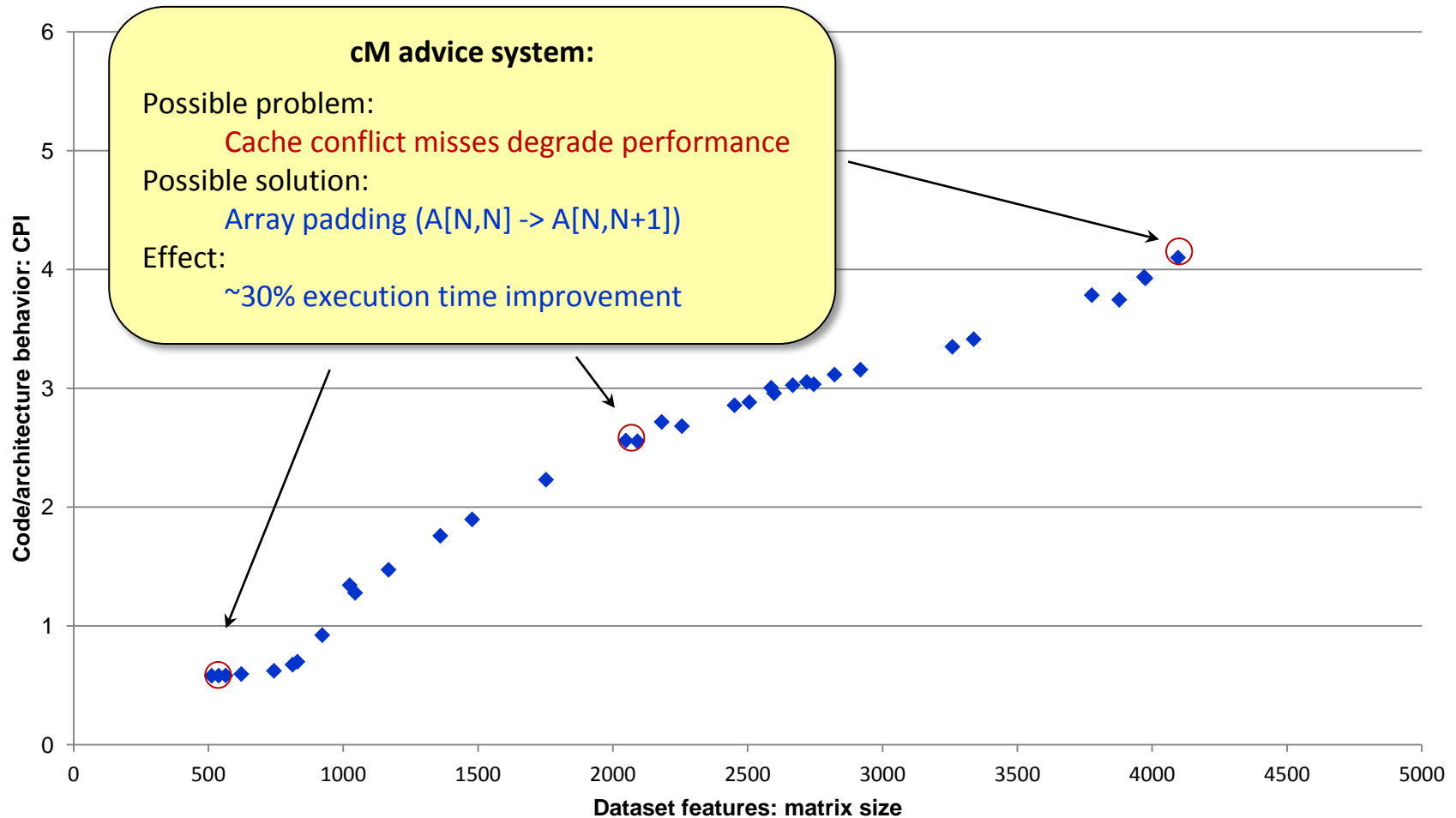
Extensible and collaborative advice system

Collaboratively and continuously add **expert advices** or **automatic optimizations**.

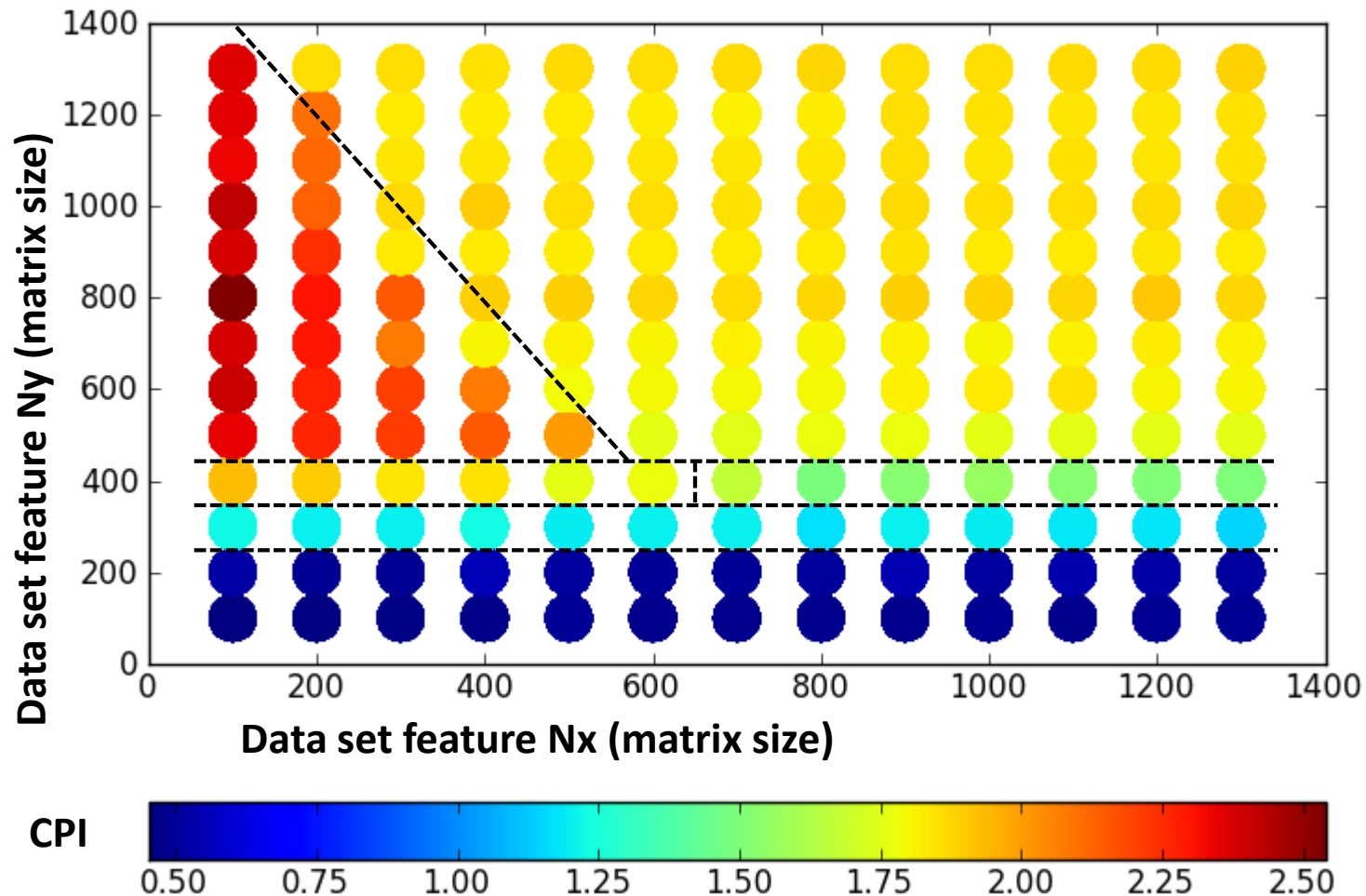


Extensible and collaborative advice system

Collaboratively and continuously add **expert advices** or **automatic optimizations**.



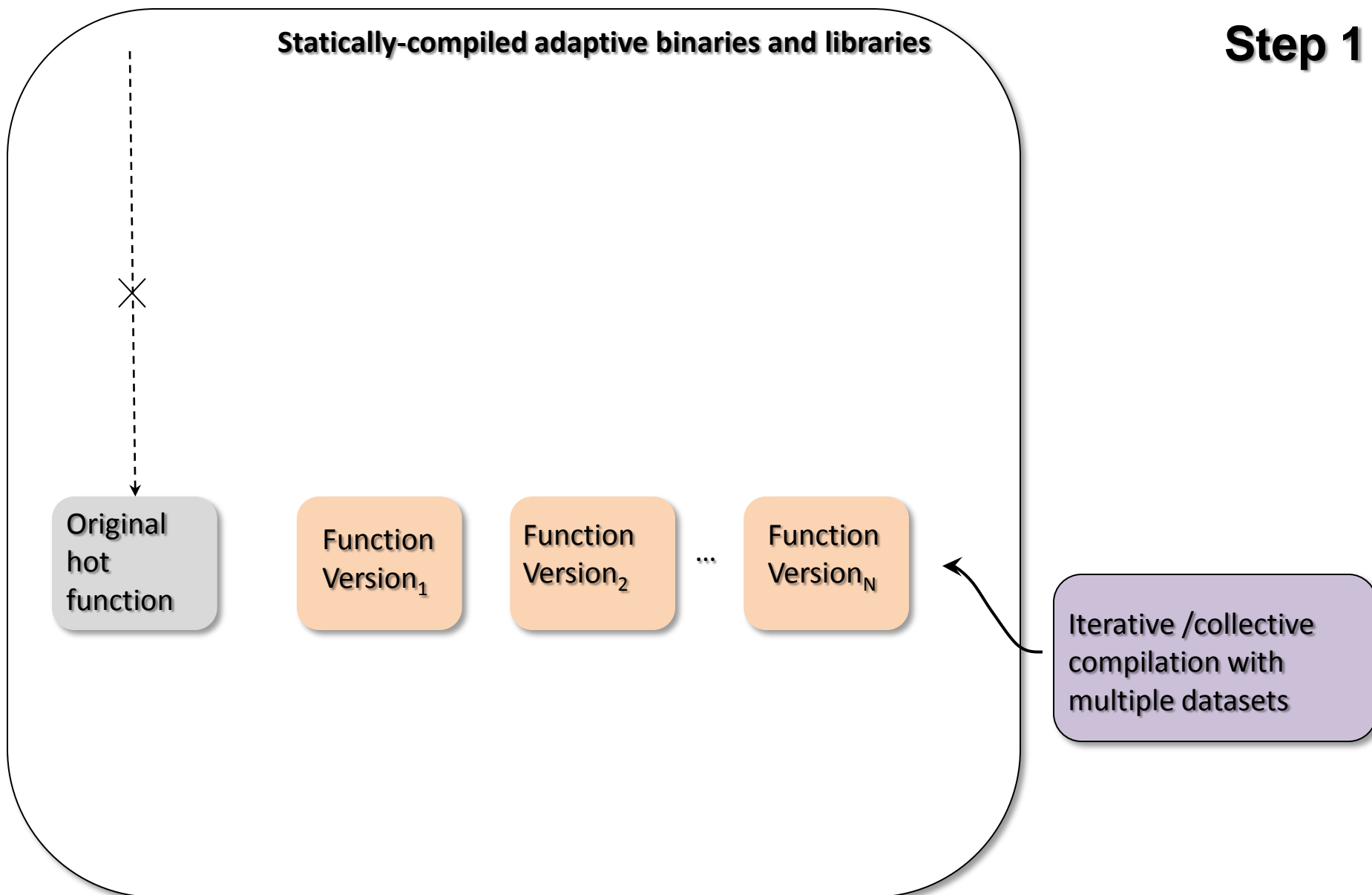
Model-driven program optimizations



Off-the-shelf models can handle some example:
matmul, Intel i5, MARS model

Static multiversioning framework for dynamic optimizations

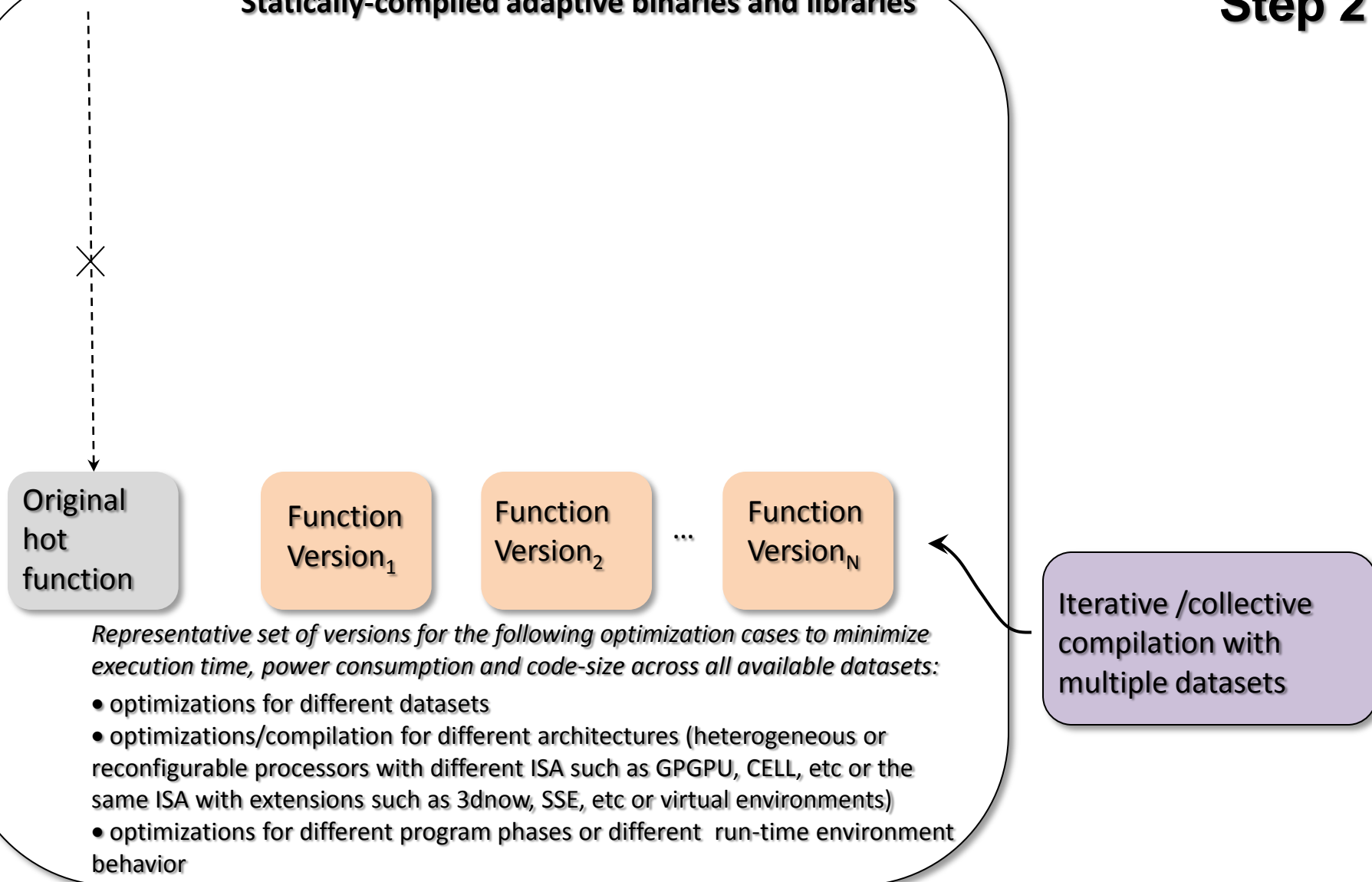
Step 1



Static multiversioning framework for dynamic optimizations

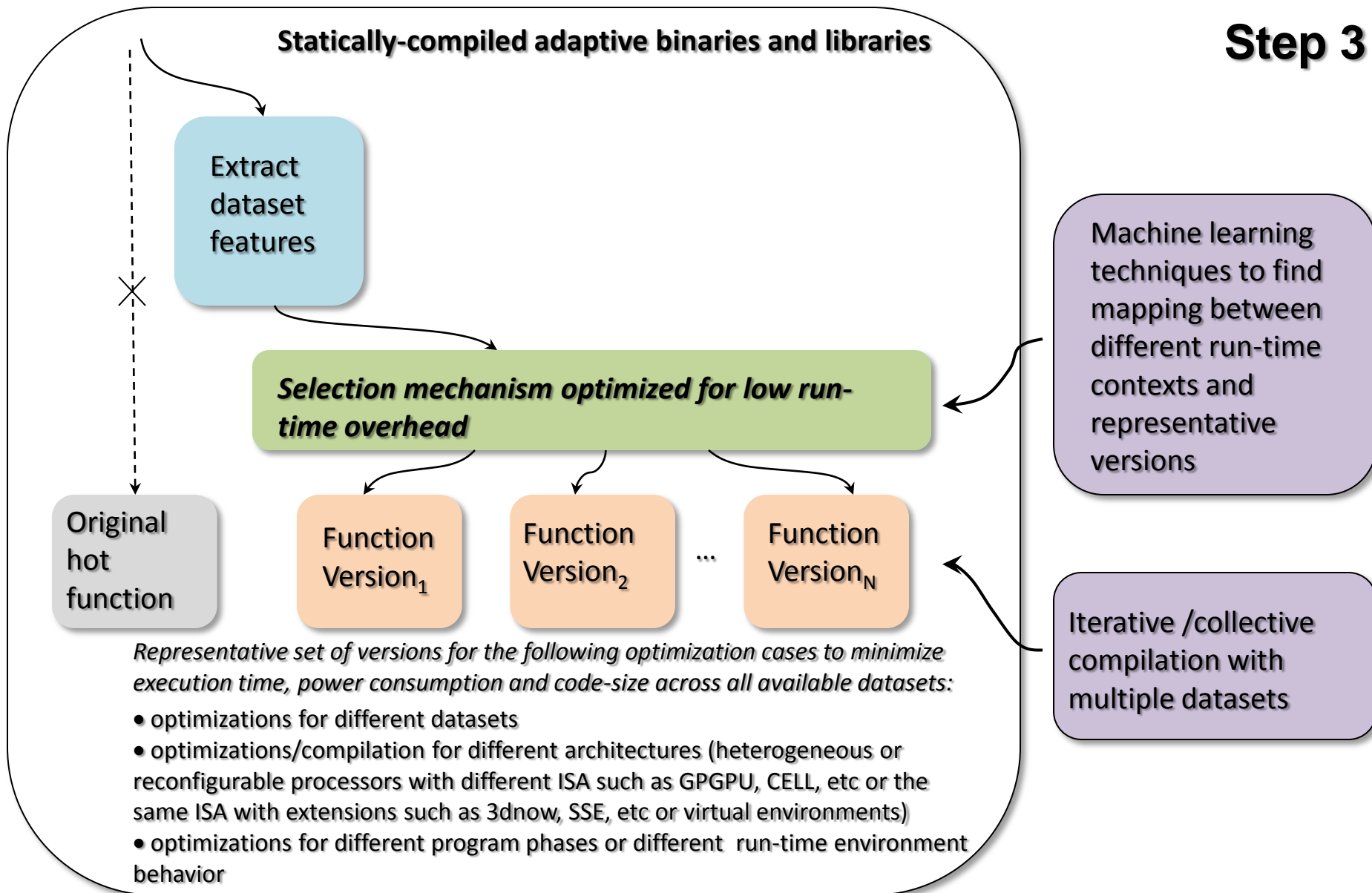
Step 2

Statically-compiled adaptive binaries and libraries

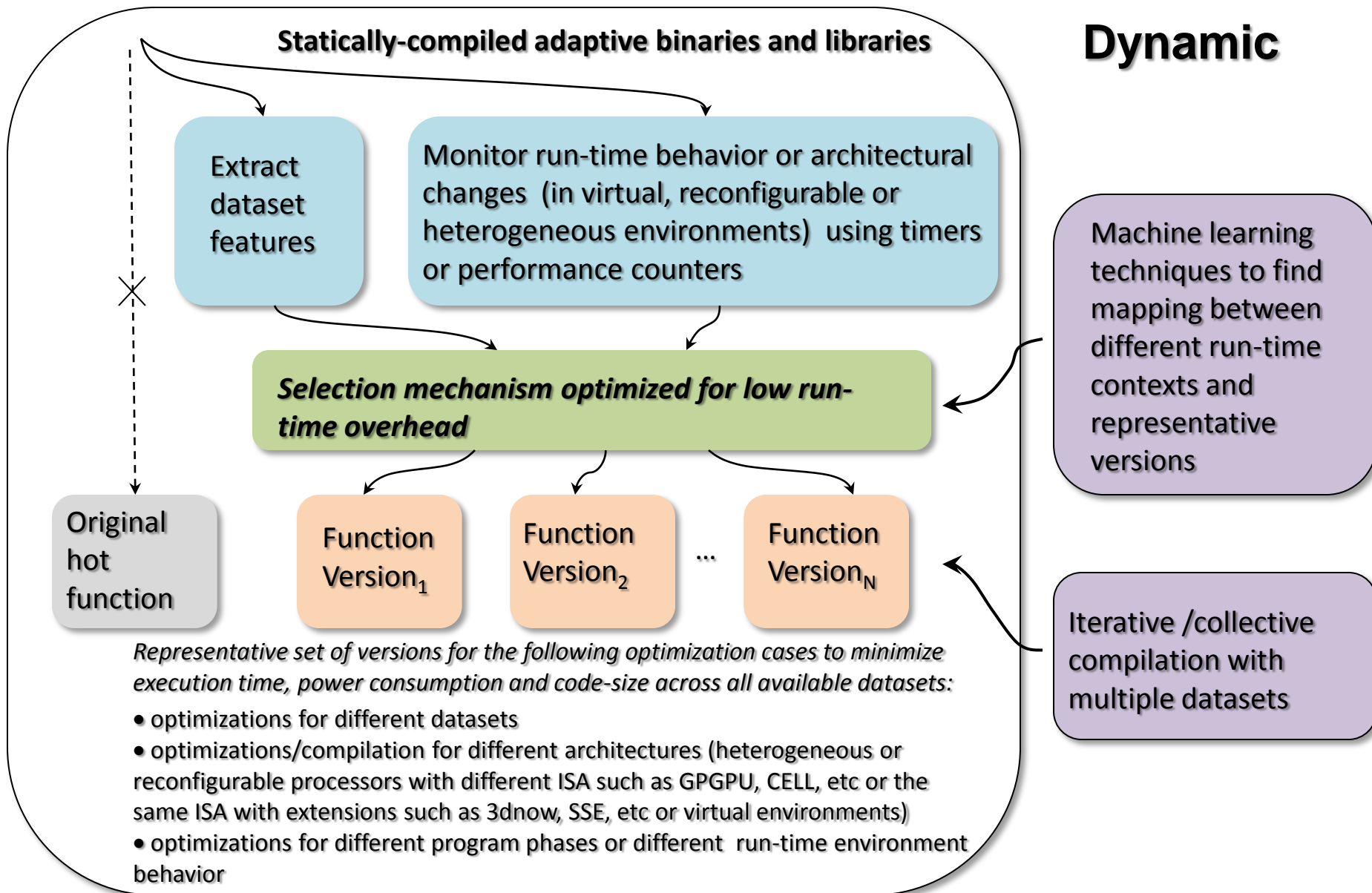


Static multiversioning framework for dynamic optimizations

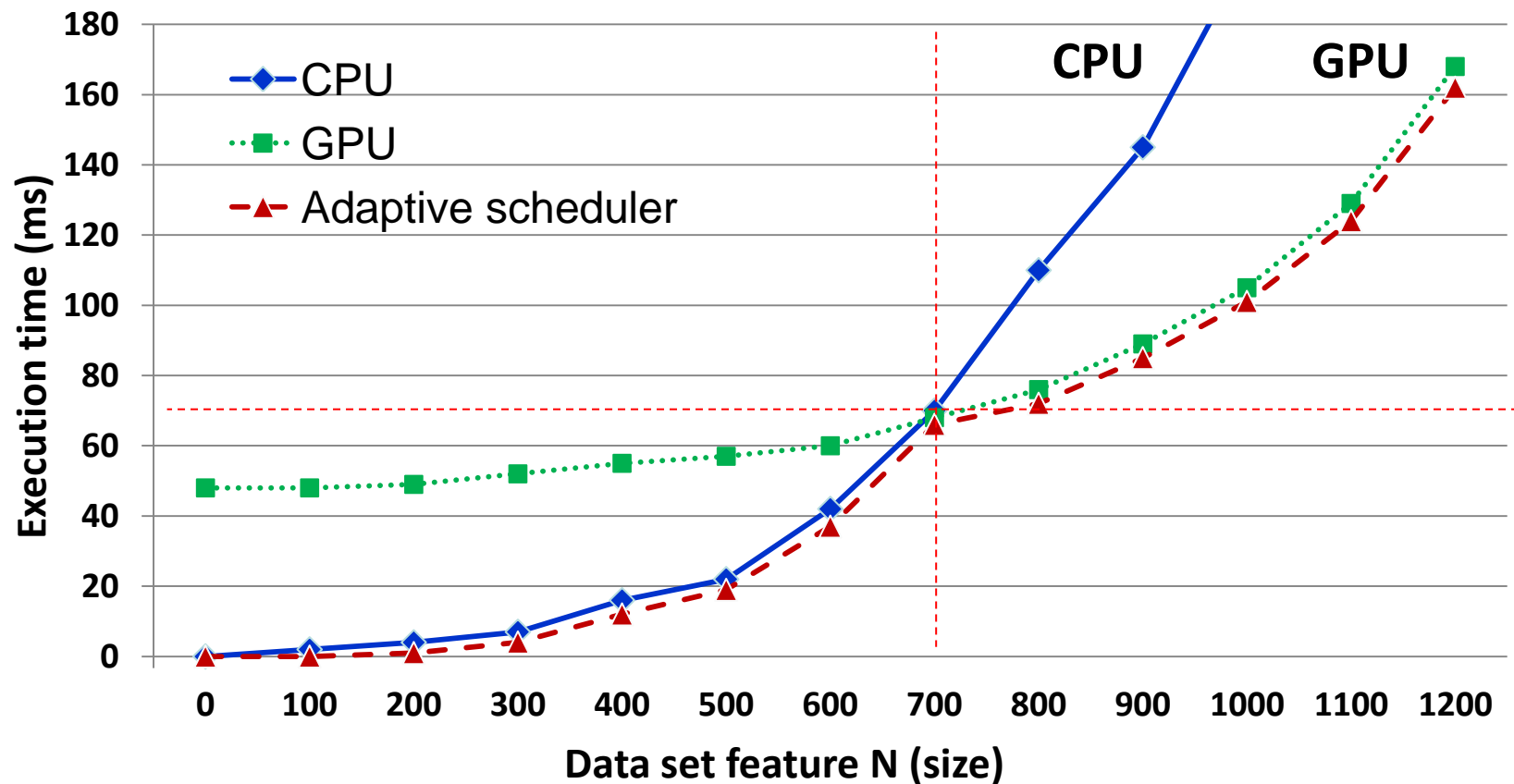
Step 3



Static multiversioning framework for dynamic optimizations



Adaptive scheduling for heterogeneous architectures



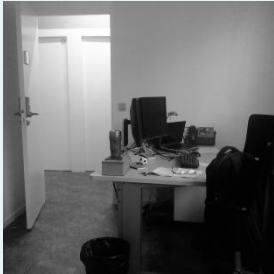
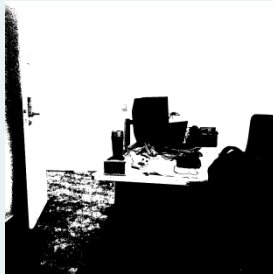

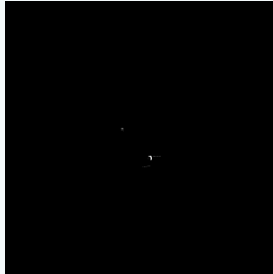
Grigori Fursin, Victor Jimenez - EU HiPEAC explorator grant (2008-2009)

Víctor J. Jiménez, Lluís Vilanova, Isaac Gelado, Marisa Gil, Grigori Fursin, Nacho Navarro,
Predictive Runtime Code Scheduling for Heterogeneous Architectures, HiPEAC 2009, pp.19-33

Tricky part: find right features

Image B&W threshold filter




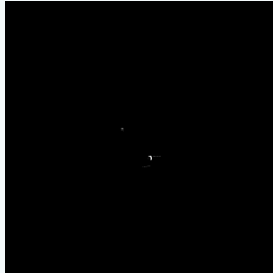
```
*matrix_ptr2++ = (temp1 > T) ? 255 : 0;
```

Class	-O3	-O3 -fno-if-conversion
Shared data set sample ₁	<i>reference execution time</i> 	no change 
Shared data set sample ₂	no change 	<i>+17.3% improvement</i> 

Tricky part: find right features

Image B&W threshold filter




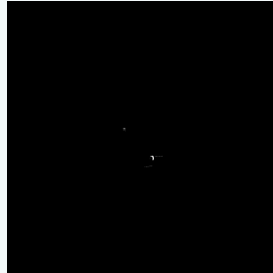
```
*matrix_ptr2++ = (temp1 > T) ? 255 : 0;
```

Class	-O3	-O3 -fno-if-conversion
Shared data set sample ₁ <i>Monitored during day</i>	<i>reference execution time</i> 	no change 
Shared data set sample ₂ <i>Monitored during night</i>	no change 	<i>+17.3% improvement</i> 

Tricky part: find right features

Image B&W threshold filter

```
*matrix_ptr2++ = (temp1 > T) ? 255 : 0;
```

Class	-O3	-O3 -fno-if-conversion
Shared data set sample ₁ <i>Monitored during day</i>	<i>reference execution time</i> 	no change 
Shared data set sample ₂ <i>Monitored during night</i>	no change 	+17.3% improvement 

Feature “**TIME_OF_THE_DAY**” related to algorithm, data set and run-time
Can’t be found by ML - simply does not exist in the system!

Can use split-compilation (cloning and run-time adaptation)

```
if get_feature(TIME_OF_THE_DAY)==NIGHT  
else
```

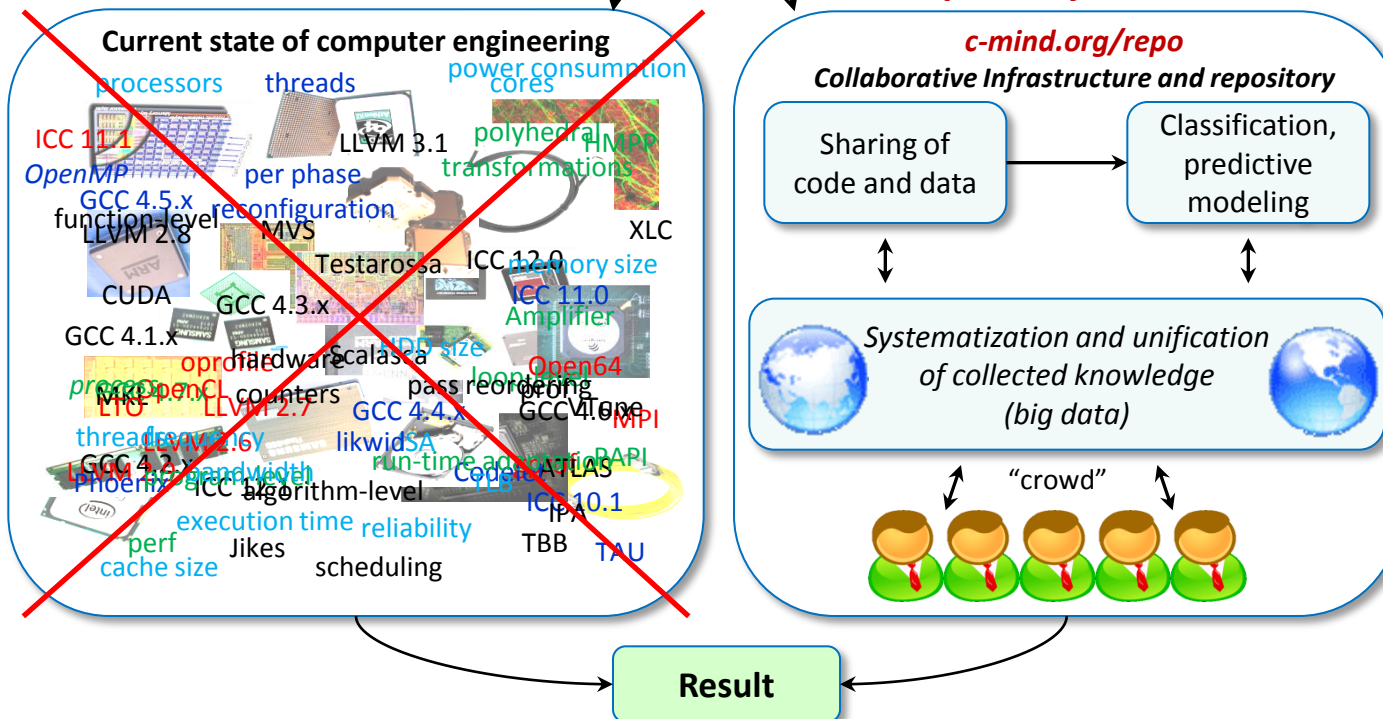
```
bw_filter_codelet_day(buffers);  
bw_filter_codelet_night(buffers);
```

My dream: making computer engineering a science!

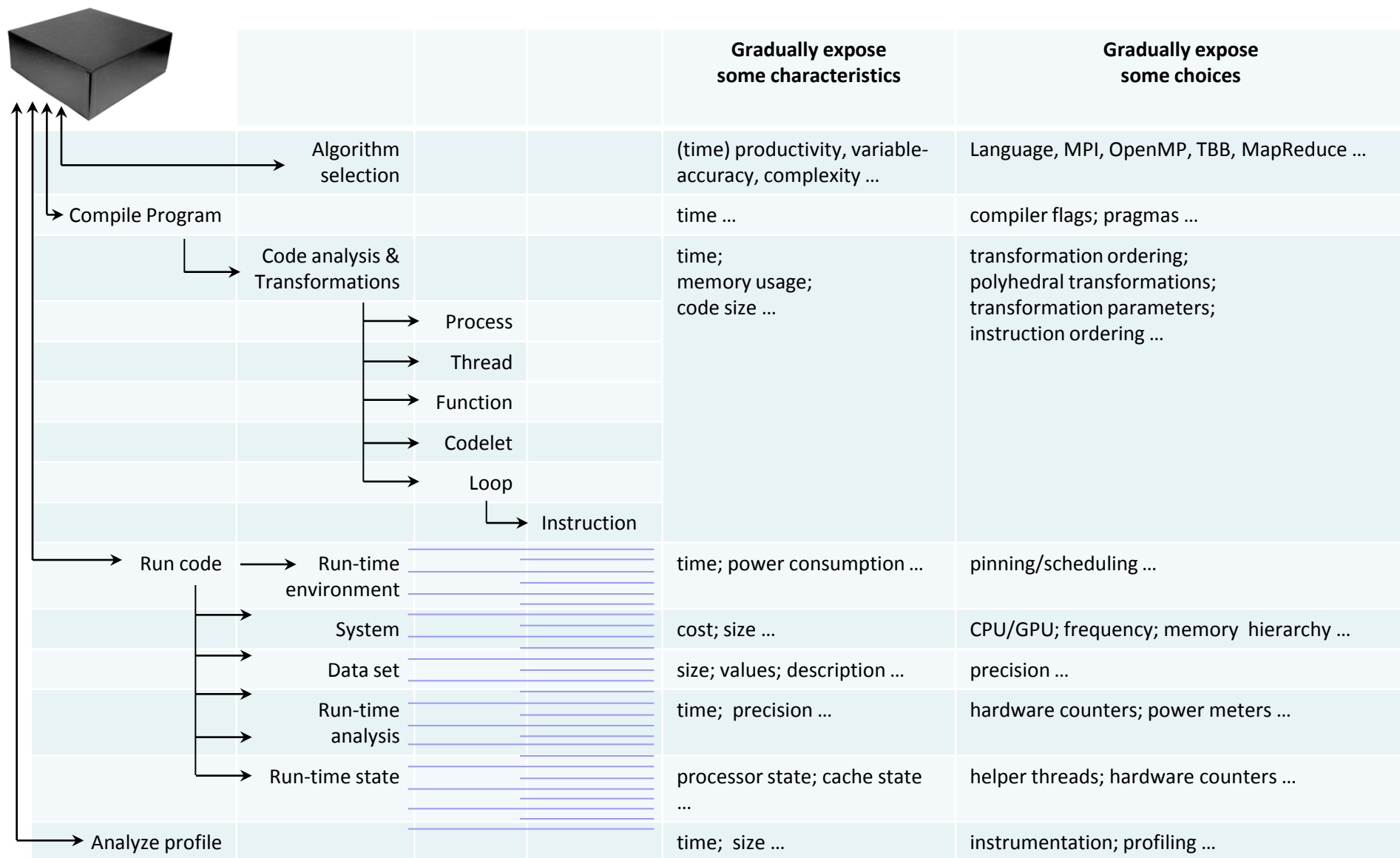
- Prototype research idea
- Validate existing work
- Perform end-user task

- **Quick, non-reproducible hack?**
- **Ad-hoc heuristic?**
- **Quick publication?**
- **No shared code and data?**

- **Share code, data with their meta-description and dependencies**
- **Systematize and classify collected optimization knowledge**
- **Develop and preserve the whole experimental pipeline**
- **Validate experimental results by the community**
- **Extrapolate collected knowledge to build faster, smaller, more power efficient and reliable computer systems**



Gradually increase granularity and complexity



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

Growing, plugin-based cM pipeline for auto-tuning and learning

<http://c-mind.org/ctuning-pipeline>



•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

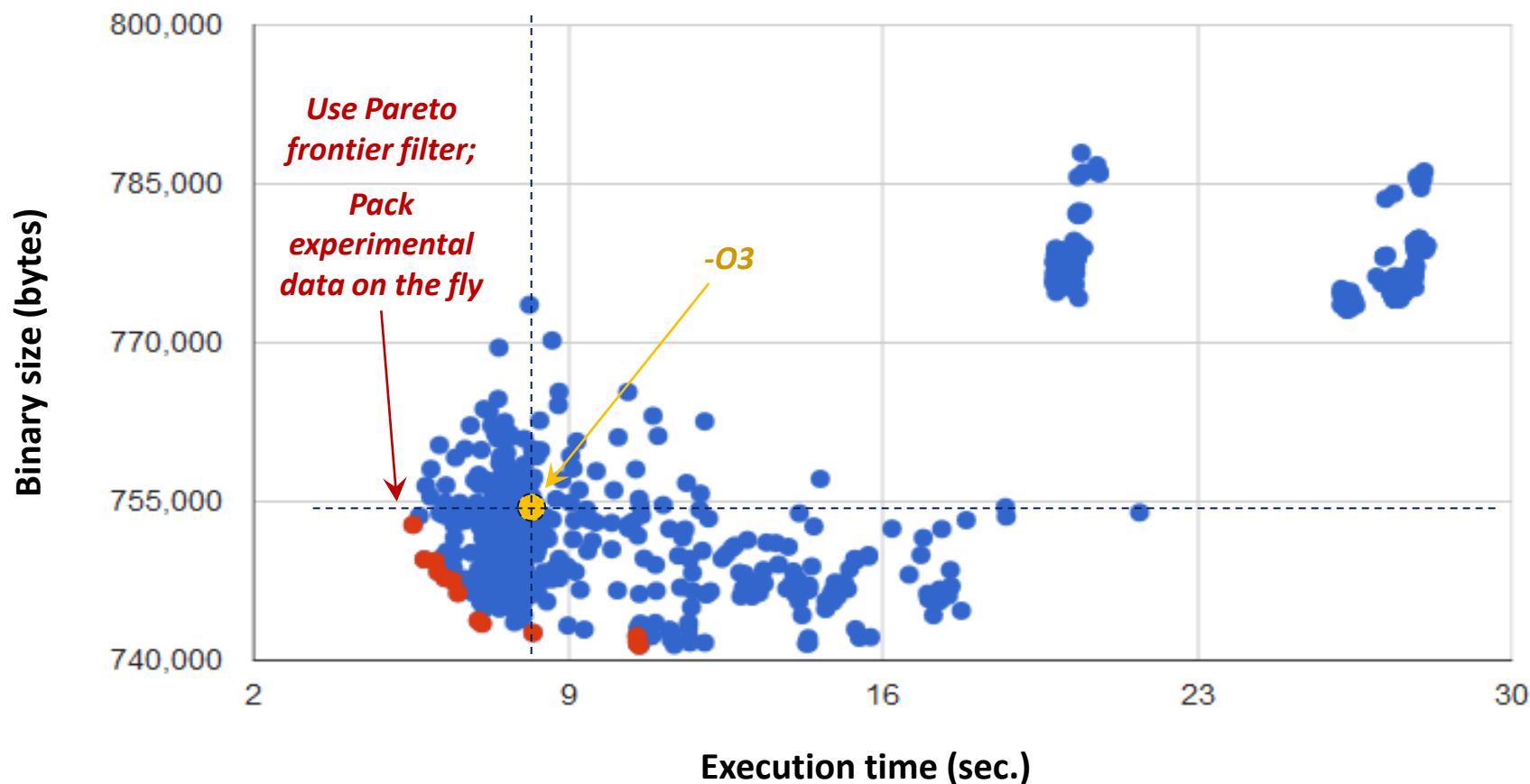
- Observed statistical characteristics

•Finalize pipeline

Our Collective Mind Buildbot and plugin-based auto-tuning pipeline supports the following shared benchmarks and codelets:

- 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 Enterprise)
- 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

Multi-objective compiler auto-tuning using mobile phones



Program: *image corner detection*
Compiler: *Sourcery GCC for ARM v4.7.3*
System: *Samsung Galaxy Y*

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

500 combinations of random flags -O3 -f(no-)FLAG

Powered by Collective Mind Node (Android Apps on Google Play)



Need new publication model in computer engineering
where results are shared and validated by community

Share benchmarks, data sets,
tools, predictive models,
whole experimental setups,
specifications, performance
tuning results, etc ...

Open access publication

<http://hal.inria.fr/hal-00685276>

Grigori Fursin, Cupertino Miranda, Olivier
Temam, Mircea Namolaru, Elad Yom-
Tov, Ayal Zaks, Bilha Mendelson, Phil
Barnard, Elton Ashton, Eric Courtois,
Francois Bodin, Edwin Bonilla, John
Thomson, Hugh Leather, Chris Williams,
Michael O'Boyle. **MILEPOST GCC:
machine learning based research
compiler.**

#ctuning-opt-case 24857532370695782

What have we learnt from cTuning

It's fun and motivating 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...

Community was interested to validate and improve techniques!

Community can identify missing related citations and projects!

Open discussions can provide new directions for research!

What have we learnt from cTuning

It's fun and motivating 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...

Community was interested to validate and improve techniques!

Community can identify missing related citations and projects!

Open discussions can provide new directions for research!

Not all feedback is positive - however unlike unfair reviews you can engage in discussions and explain your position!

Reproducibility of experimental results

Reproducibility came as a side effect!

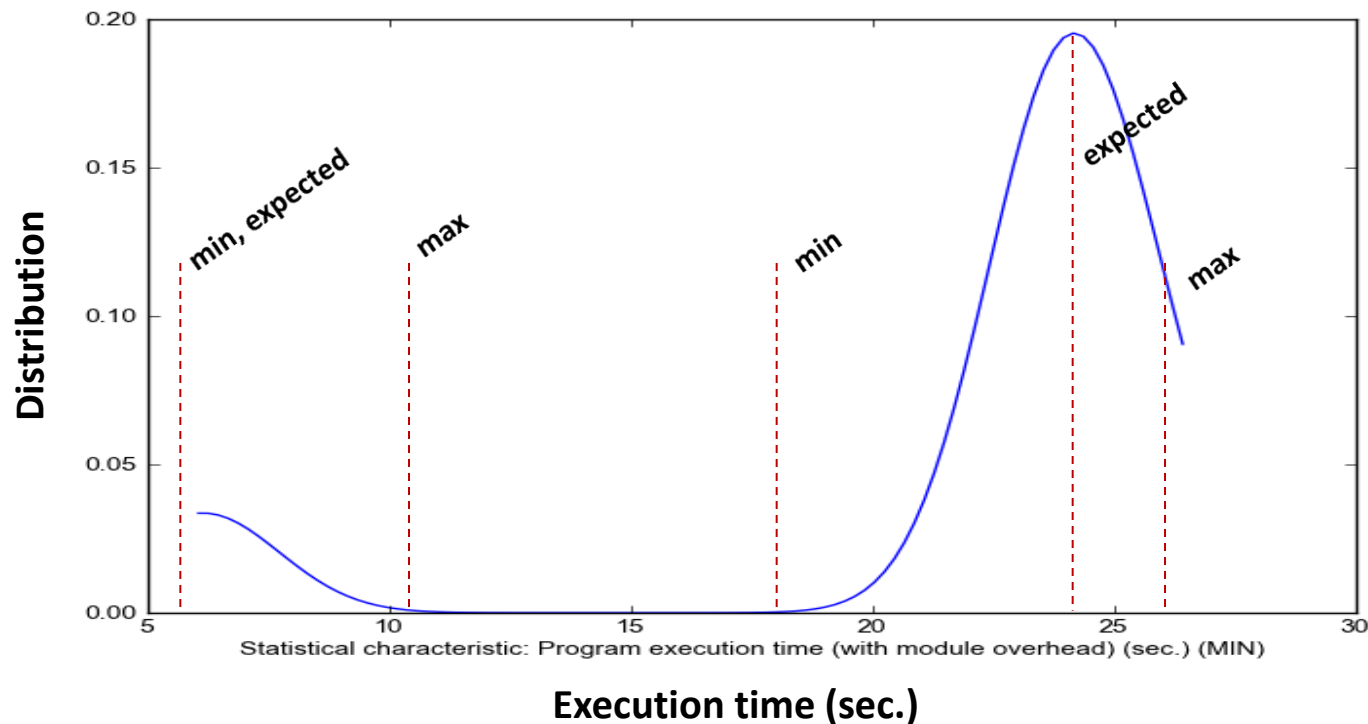
- Can preserve the whole experimental setup with all data and software dependencies
- Can perform statistical analysis (normality test) for characteristics
- Community can add missing features or improve machine learning models

Reproducibility of experimental results

Reproducibility came as a side effect!

- Can preserve the whole experimental setup with all data and software dependencies
- Can perform statistical analysis (normality test) for characteristics
- Community can add missing features or improve machine learning models

Unexpected behavior - expose to the community including domain specialists, explain, find missing feature and add to the system

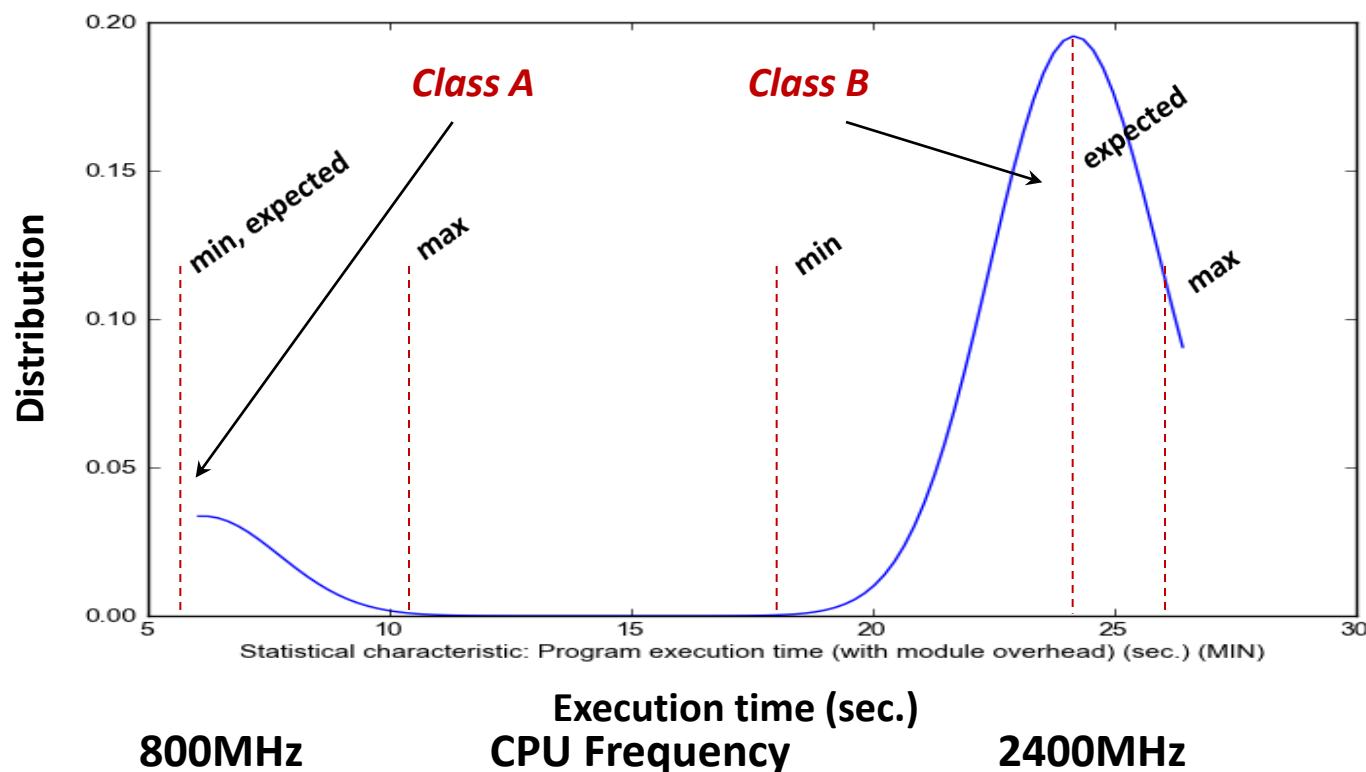


Reproducibility of experimental results

Reproducibility came as a side effect!

- Can preserve the whole experimental setup with all data and software dependencies
- Can perform statistical analysis (normality test) for characteristics
- Community can add missing features or improve machine learning models

Unexpected behavior - expose to the community including domain specialists, explain, find missing feature and add to the system



Proposal: crowdsourcing validation of publications and research results

- Submit papers to open access archives (arXiv, HAL, etc)
- Make all related research material either at the personal website or at public sharing services
- Initiate discussion at social networking sites with ranking (Reddit, SlashDot, StackExchange) or without (Google+, Facebook)
- Arrange first small program committee that monitors discussions to filter obviously wrong, unreproducible or possibly plagiarized
- Select a set of “interesting” papers and send it to a interdisciplinary program committee based on paper topics and public discussions

**See our workshop at ACM SIGPLAN TRUST’14
co-located with PLDI’14 in 2 weeks in Edinburgh!**

Proposal: crowdsourcing validation of publications and research results


- Select final papers based on public discussions and professional reviews
- Create an open access reproducible online journal with all related materials from the most interesting, advanced and highest ranked publications
- Send considerably updated papers to traditional journals (not to break current system but make open access and traditional publication models co-exist)



Plan to validate at ADAPT'15 (adapt-workshop.org)
workshop on adaptive, self-tuning computing systems

Current status and future work

- Pilot live repository for public curation of research material: <http://c-mind.org/repo>
- Infrastructure is available at SourceForge under standard BSD license: <http://c-mind.org>
- Example of crowdsourcing compiler flag auto-tuning using mobile phones: “**Collective Mind Node**” in Google Play Store
- Several publications under submission
- Raising funding to make cM more user friendly and add more research scenarios

Education	Academic research
<p>New publication model where research material and experimental results are shared, validated and reused by the community</p> <p>http://ctuning.org/reproducibility</p> <ul style="list-style-type: none">• ACM SIGPLAN TRUST 2014 @ PLDI 2014 http://c-mind.org/events/trust2014• Panel at ADAPT 2014 @ HiPEAC 2014 http://adapt-workshop.org (this year we attempted to reproduce some submitted articles)	<div></div> <ul style="list-style-type: none">• Systematizing, validating, sharing past research techniques on auto-tuning and machine learning through cM pipeline• Optimal feature and model selection• Finding representative benchmarks and data sets• Run-time adaptation and ML

Grigori Fursin, “Collective Mind: cleaning up the research and experimentation mess in computer engineering using crowdsourcing, big data and machine learning”, INRIA Tech. report No 00850880, August 2013

<http://hal.inria.fr/hal-00850880>

<http://arxiv.org/abs/1308.2410>

Acknowledgements

- Colleagues from ARM (UK): Anton Lokhmotov
- Colleagues from Intel (USA): *David Kuck and David Wong*
- Colleagues from STMicroelectronics (France):
Christophe Guillone, Antoine Moynault, Christian Bertin
- Colleagues from NCAR (USA): *Davide Del Vento and his interns*
- Colleagues from CAPS Entreprise (France): *Francois Bodin*
- cTuning/Collective Mind community:



- EU FP6, FP7 program and HiPEAC network of excellence
<http://www.hipeac.net>

Questions? Comments?