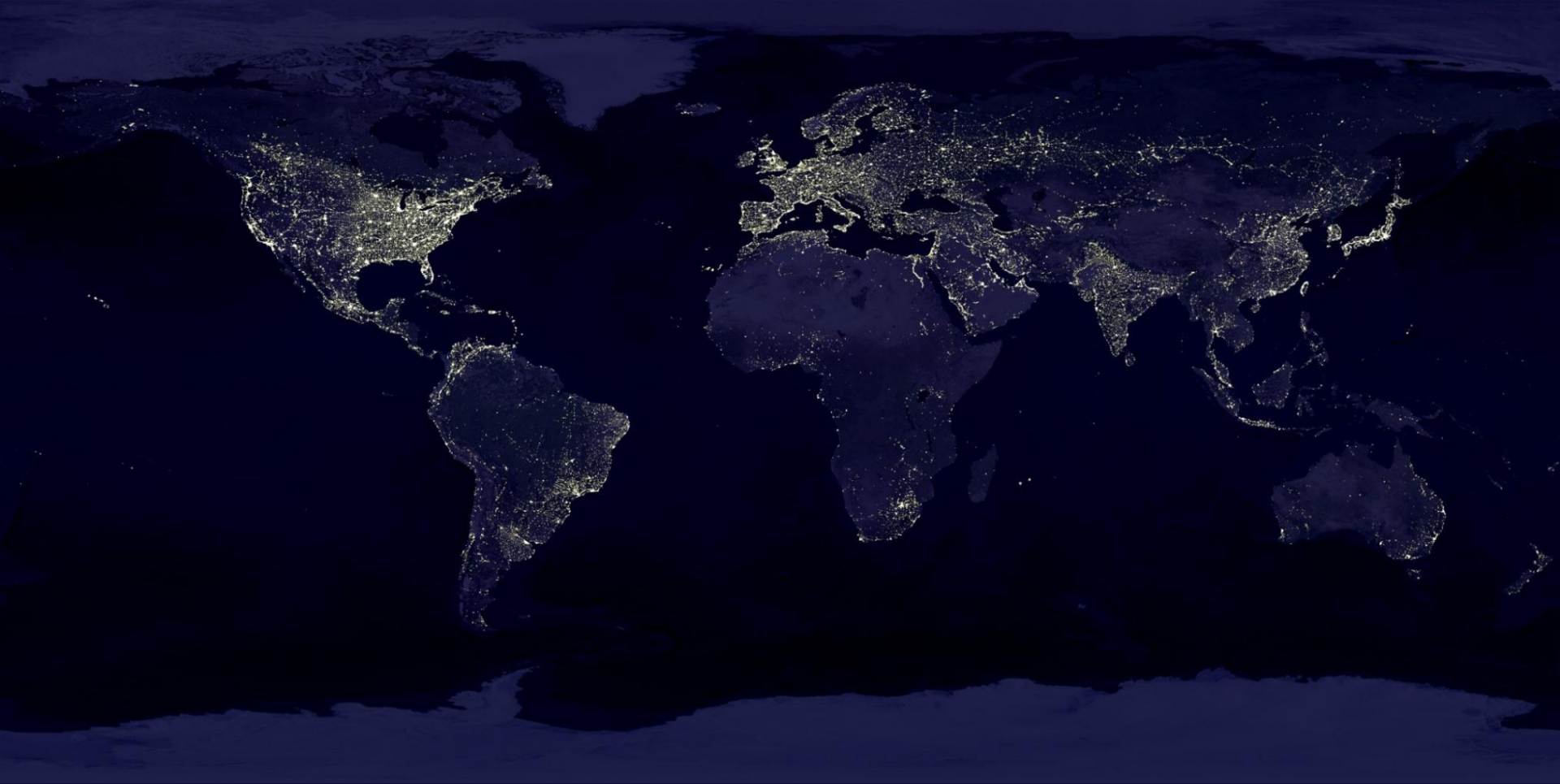


# **Collective Mind:** making auto-tuning practical using crowdsourcing and predictive modeling



**Grigori Fursin**  
**INRIA, France**

**INRIA-Illinois-ANL 10<sup>th</sup> workshop**  
**Urbana, IL, USA**  
**November 2013**

# Summary

## Challenges:

- How to abstract and unify whole system auto-tuning and modeling?
- How to predict optimizations while helping architecture or compiler designers?
- How to preserve all past tuning knowledge and extrapolate it to the new systems?

- General problems in computer engineering

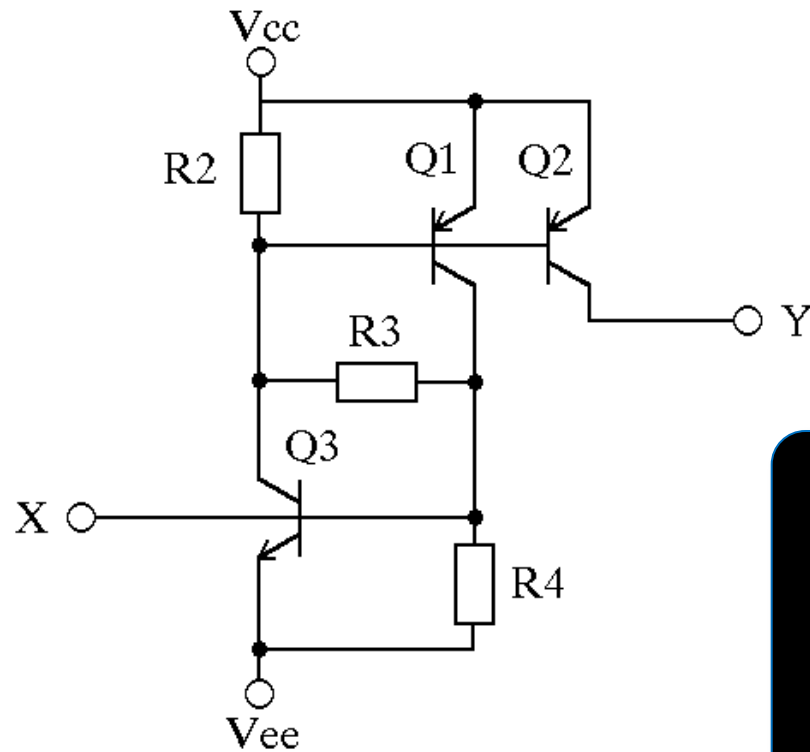
- Cleaning up research and experimental mess

- *Collective Mind Repository, infrastructure and methodology*
- *Reproducible research and experimentation*
- *Crowdsourcing, predictive modelling*

- Unifying compiler multi-objective auto-tuning

- Unifying performance modelling

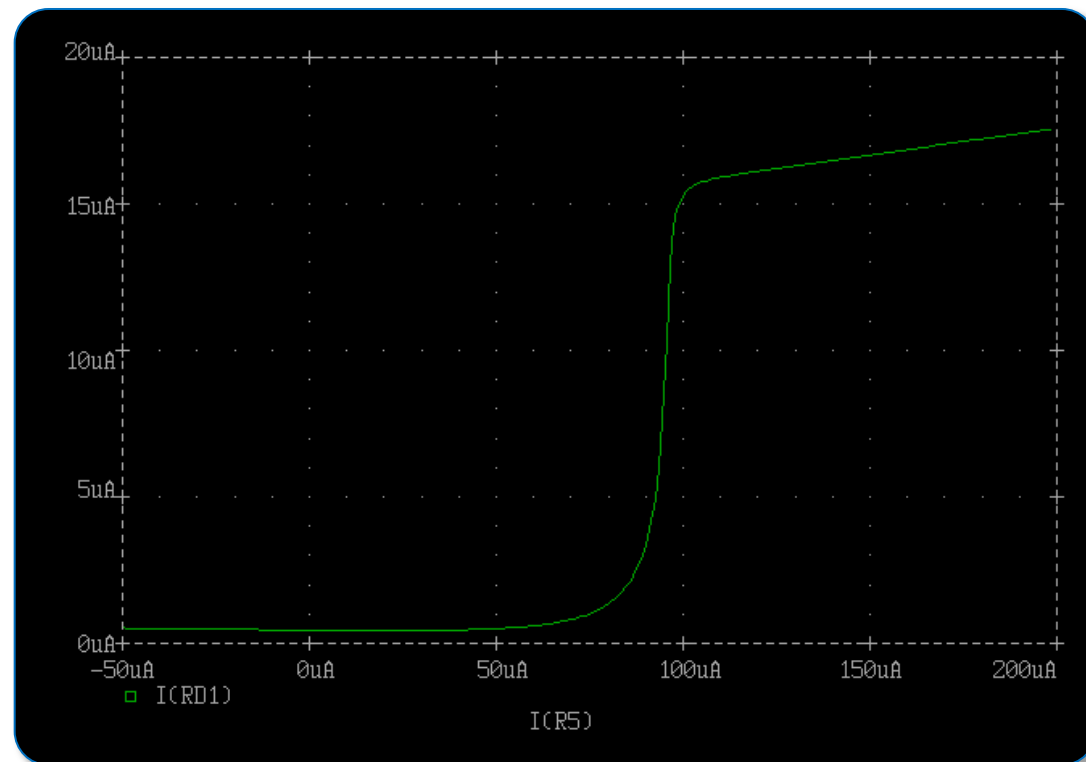
- Conclusions and future work



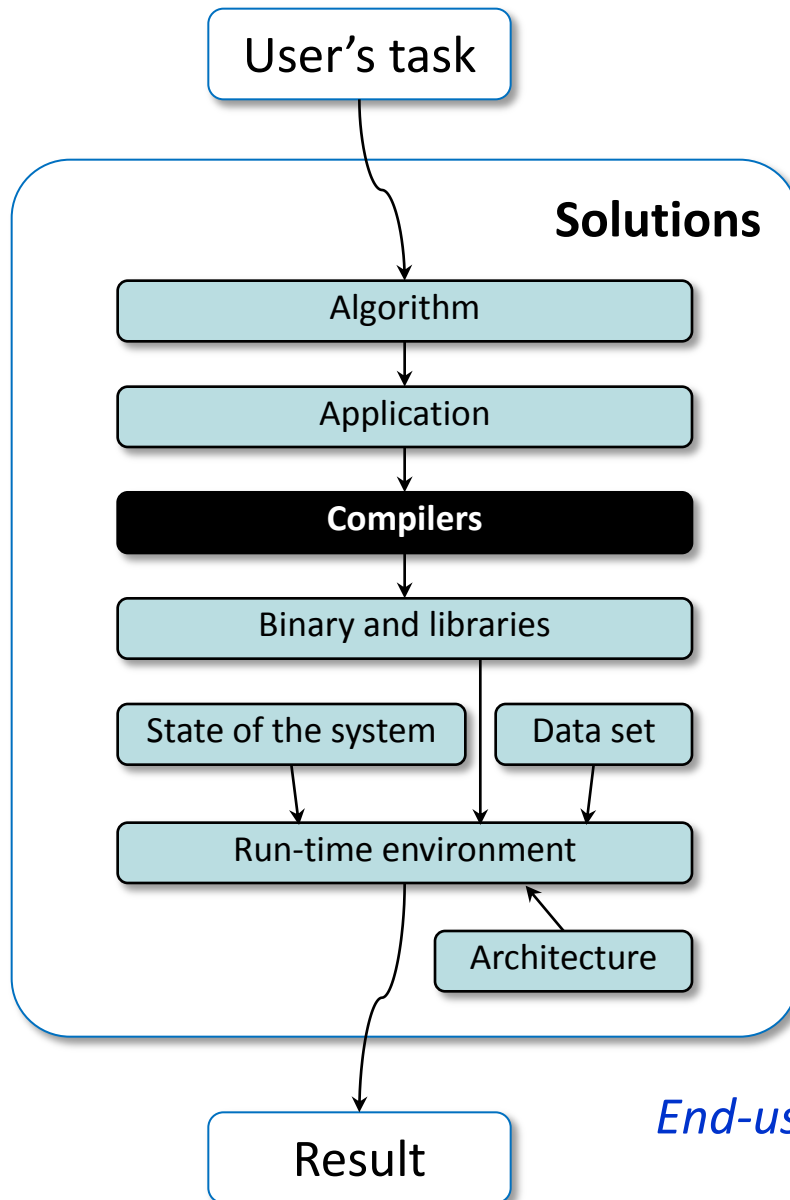
***Semiconductor neural element -***  
base of neural accelerators  
and computers  
*Modeling and understanding*  
*brain functions*

***My problem***  
***with modeling:***

- **Slow**
- **Unreliable**
- **Costly**



# Problems I have been facing since 1993



*End-users care about performance, reliability, costs.  
Technology is secondary!*

# Problems I have been facing since 1993

User's task



Result

**Delivering optimal solutions is tough:**

- 1) Rising complexity of computer systems:  
too many design and optimization choices  
at ALL levels
- 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

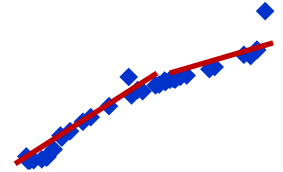
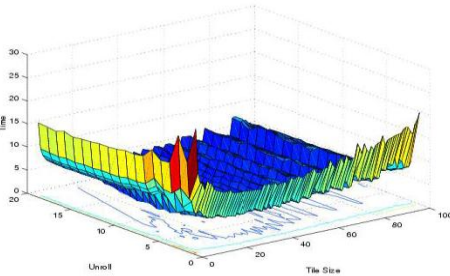
*End-users care about performance, reliability, costs.  
Technology is secondary!*

# 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***



# Compiler auto-tuning

## Major problems in my projects:

- Long training times (both auto-tuning and ML)

*1999-2005 (PhD and EU MHAOTEU project)*

4 kernels / SPEC2000, 1 datasets, 2 architectures,  
tiling/unrolling/padding, **~4 months of experiments**,  
**SHARED as CSV and thorough MySQL**

*2006-2009 (EU MILEPOST project)*

16 benchmarks, 1dataset, 3 architectures,  
GCC and ICC, 500 combinations of flags,  
**~6 months of experiments**, **SHARED through**  
**MySQL, plugin-based framework and web services**

*2009-2011 (Collective Tuning)*

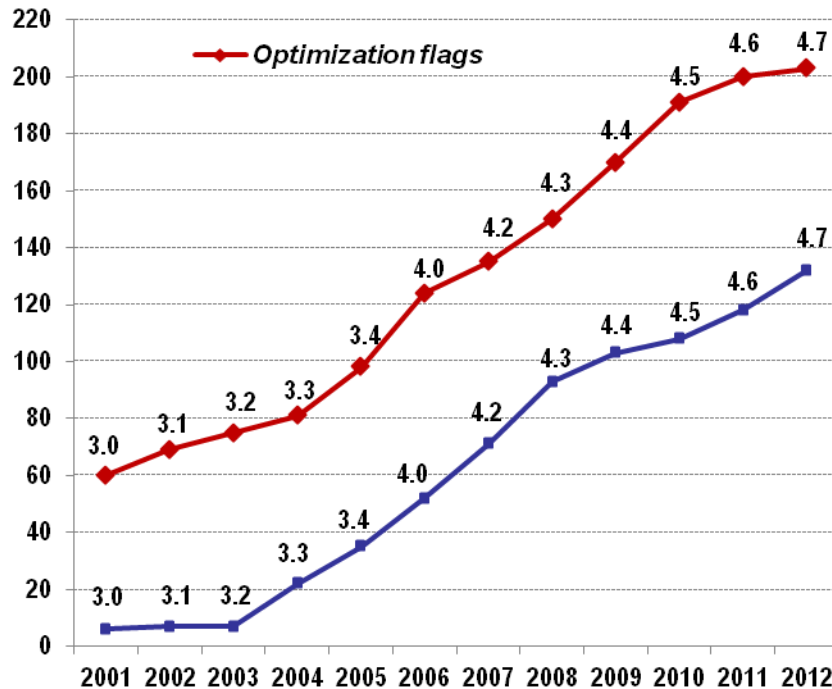
16 benchmarks, 20..1000 datasets, GRID5000 with  
16 nodes, **~10 months of experiments**,  
**SHARED through MySQL, plugin based framework**  
**and web services**

*2011-cur (Collective Mind)*

300 benchmarks, 20..1000 datasets  
GRID5000 with 100 nodes,

**Some experiments are still in progress,**  
**SHARED ONLINE**

## GCC optimization evolution



**Find empirically optimal optimizations  
in multi-dimensional space while  
balancing multiple characteristics:**

- execution time
- code size
- compilation time



# Can we crowdsource auto-tuning? My main focus since 2004

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!

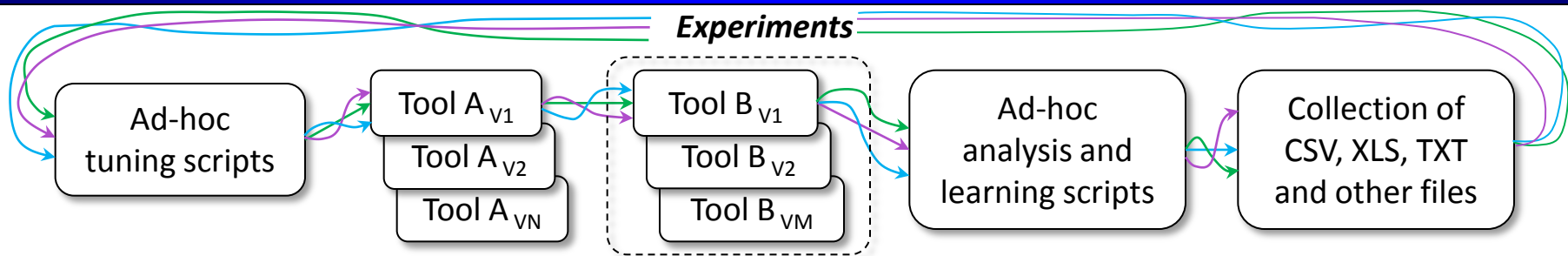


**Can we leverage their experience and computational resources?**

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



# How to implement?



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

Behavior

Choices

Features

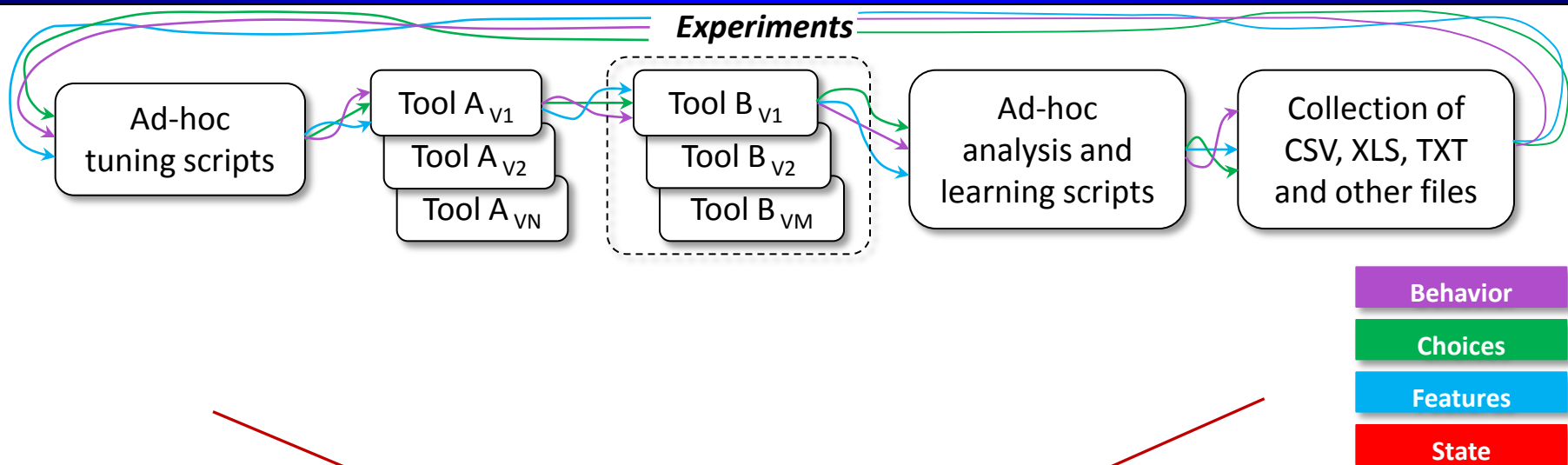
State

## Revolutionary approach:

**Let's redesign the whole system and make it tunable and adaptable?**

- Too complex and time consuming (decades)
- Community will not easily accept

# How to implement?



## ~~Revolutionary approach:~~

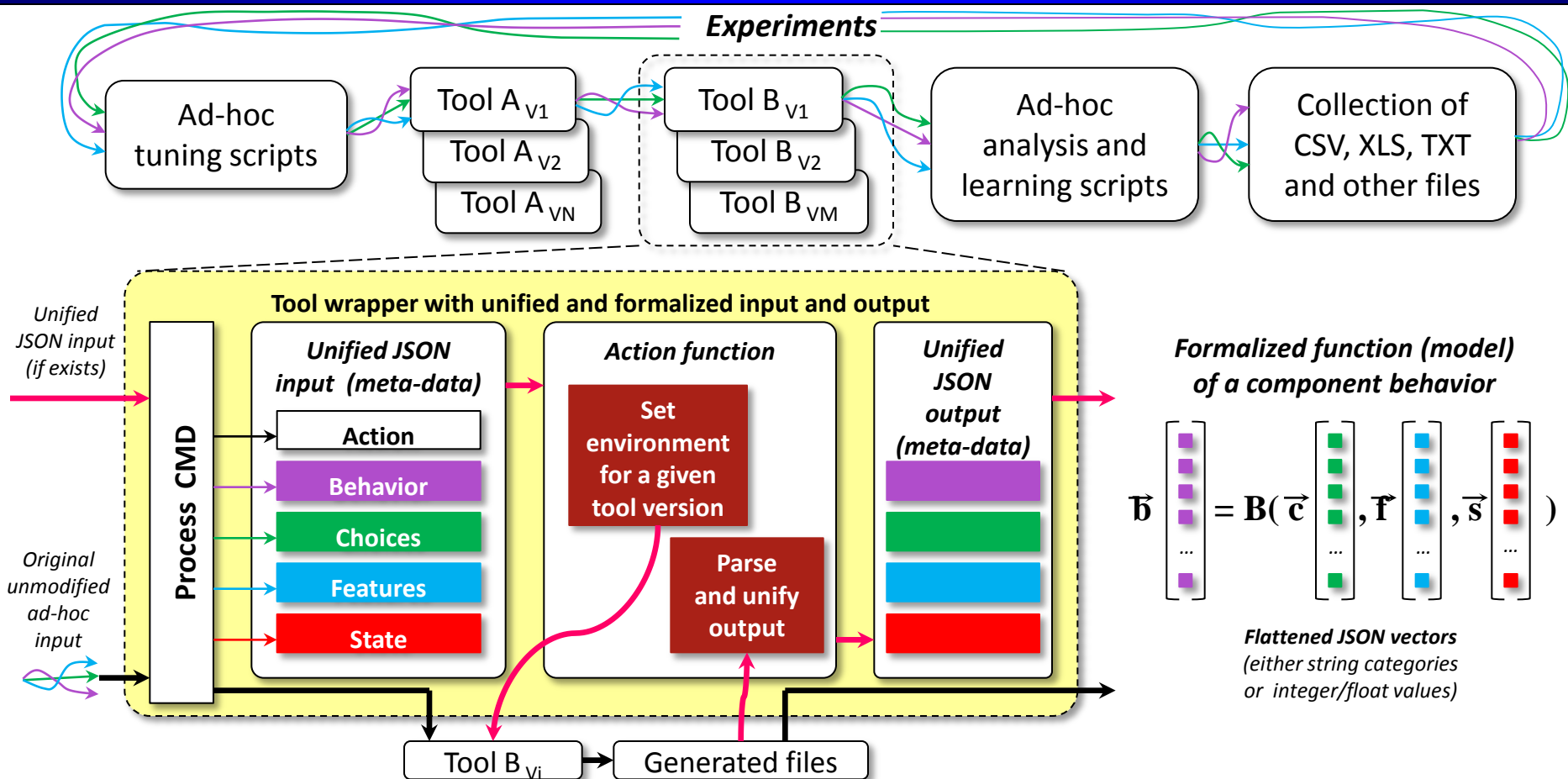
~~Let's redesign the whole system and make it tunable and adaptable?~~

- Too complex and time consuming (decades)
- Community will not easily accept

## Evolutionary agile methodology:

Gradually clean-up system and make it tunable and adaptable while involving community

# How to implement?



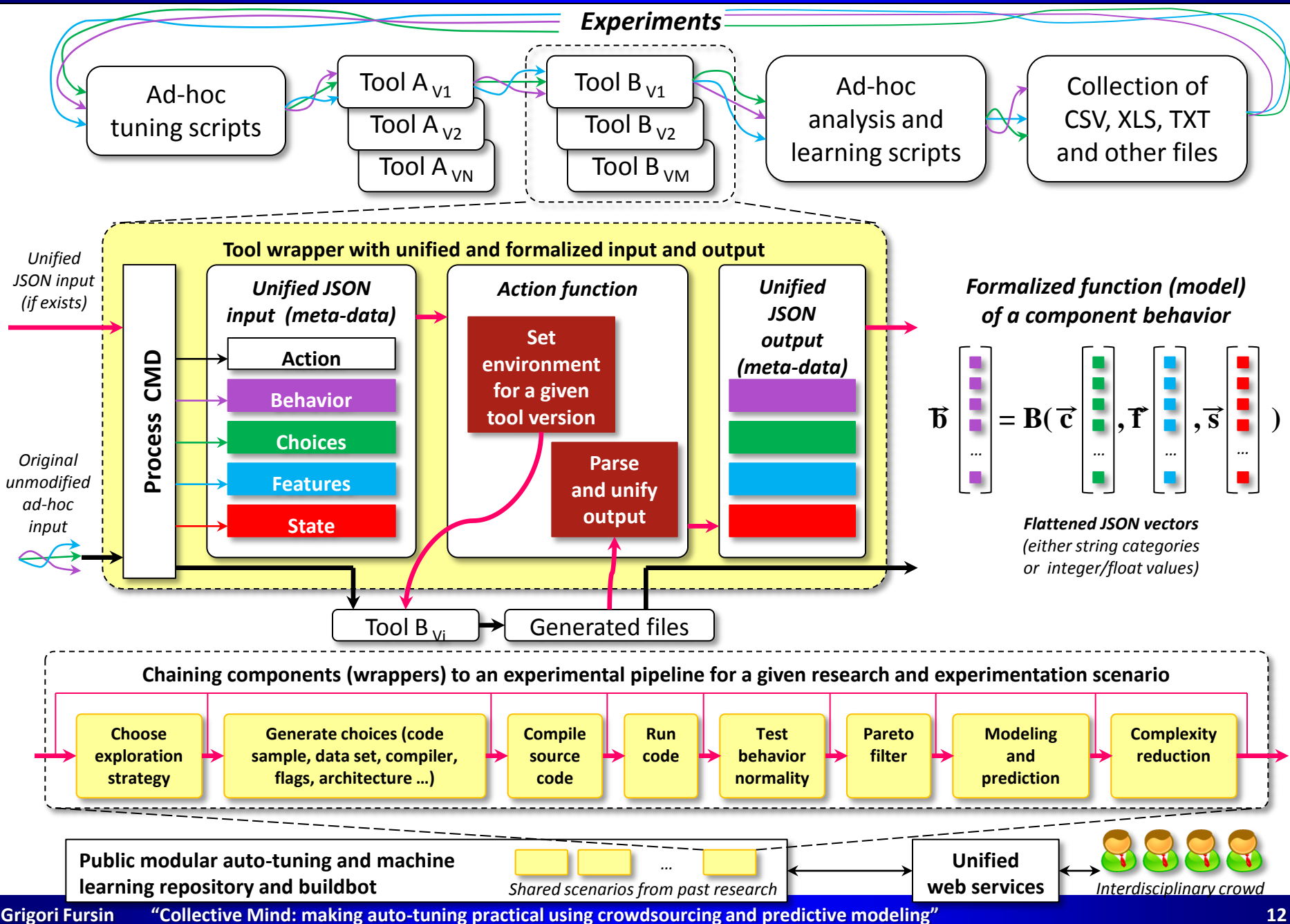
cm **[module name]** **[action]** (param<sub>1</sub>=value<sub>1</sub> param<sub>2</sub>=value<sub>2</sub> ... -- unparsed command line)

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

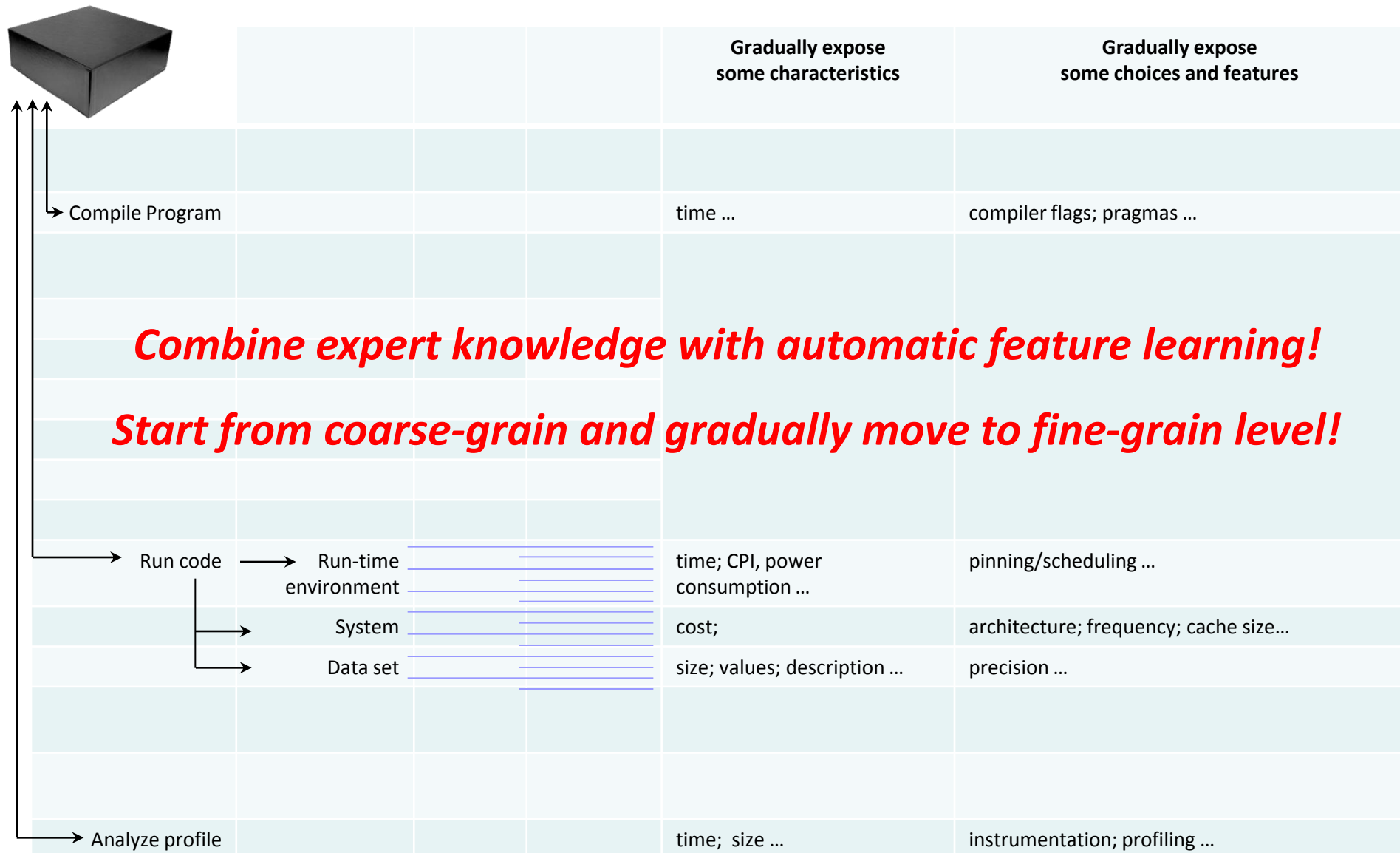
cm **code.source** build ct\_compiler=icc13 ct\_optimizations=-fast

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)!*



# Top-down decomposition and learning of computer systems



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

# 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

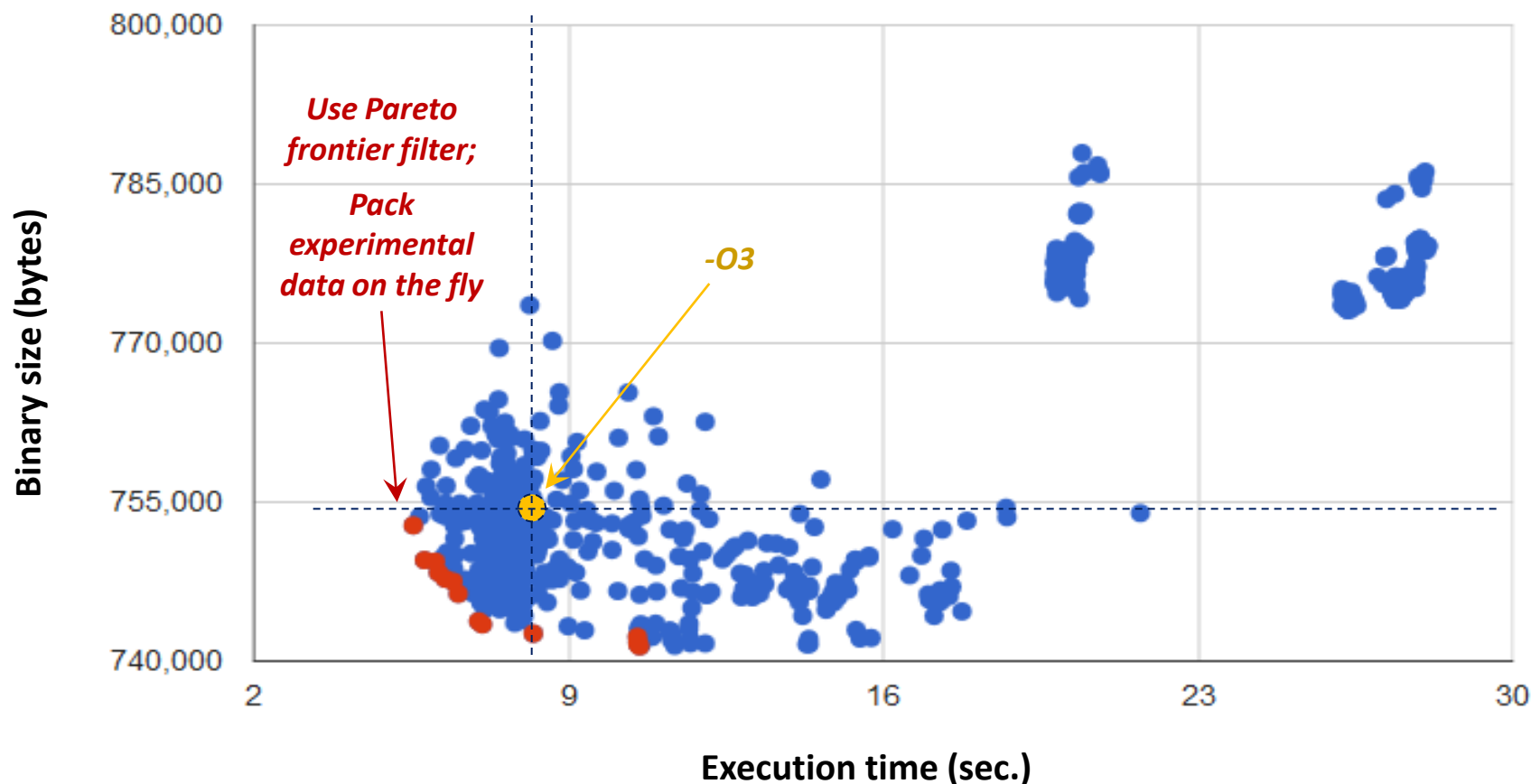
## •Finalize pipeline

***Our Collective Mind Buildbot 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)*

# Universal complexity (dimension) reduction

## Found solution

-O3 -fno-align-functions -fno-align-jumps -fno-align-labels -fno-align-loops -fno-asynchronous-unwind-tables -fno-branch-count-reg -fno-branch-target-load-optimize2 -fno-btr-bb-exclusive -fno-caller-saves -fno-combine-stack-adjustments -fno-common -fno-compare-elim -fno-conserve-stack -fno-cprop-registers -fno-crossjumping -fno-cse-follow-jumps -fno-cx-limited-range -fdce -fno-defer-pop -fno-delete-null-pointer-checks -fno-devirtualize -fno-dse -fno-early-inlining -fno-expensive-optimizations -fno-forward-propagate -fgcse -fno-gcse-after-reload -fno-gcse-las -fno-gcse-lm -fno-gcse-sm -fno-graphite-identity -fguess-branch-probability -fno-if-conversion -fno-if-conversion2 -fno-inline-functions -fno-inline-functions-called-once -fno-inline-small-functions -fno-ipa-cp -fno-ipa-cp-clone -fno-ipa-matrix-reorg -fno-ipa-profile -fno-ipa-pta -fno-ipa-pure-const -fno-ipa-reference -fno-ipa-sra -fno-ivopts -fno-jump-tables -fno-math-errno -fno-loop-block -fno-loop-flatten -fno-loop-interchange -fno-loop-parallelize-all -fno-loop-strip-mine -fno-merge-constants -fno-modulo-sched -fmove-loop-invariants -fomit-frame-pointer -fno-optimize-register-move -fno-optimize-sibling-calls -fno-peel-loops -fno-peephole -fno-peephole2 -fno-predictive-commoning -fno-prefetch-loop-arrays -fno-regmove -fno-rename-registers -fno-reorder-blocks -fno-reorder-blocks-and-partition -fno-reorder-functions -fno-rerun-cse-after-loop -fno-reschedule-modulo-scheduled-loops -fno-sched-critical-path-heuristic -fno-sched-dep-count-heuristic -fno-sched-group-heuristic -fno-sched-interblock -fno-sched-last-insn-heuristic -fno-sched-pressure -fno-sched-rank-heuristic -fno-sched-spec -fno-sched-spec-insn-heuristic -fno-sched-spec-load -fno-sched-spec-load-dangerous -fno-sched-stalled-insns -fno-sched-stalled-insns-dep -fno-sched2-use-superblocks -fno-schedule-insns -fno-schedule-insns2 -fno-short-enums -fno-signed-zeros -fno-sel-sched-pipelining -fno-sel-sched-pipelining-outer-loops -fno-sel-sched-reschedule-pipelined -fno-selective-scheduling -fno-selective-scheduling2 -fno-signaling-nans -fno-single-precision-constant -fno-split-ivs-in-unroller -fno-split-wide-types -fno-strict-aliasing -fno-thread-jumps -fno-trapping-math -fno-tree-bit-ccp -fno-tree-builtin-call-dce -fno-tree-ccp -fno-tree-ch -fno-tree-copy-prop -fno-tree-copyrename -fno-tree-cselim -fno-tree-dce -fno-tree-dominator-opts -fno-tree-dse -ftree-forwprop -fno-tree-fre -fno-tree-loop-distribute-patterns -fno-tree-loop-distribution -fno-tree-loop-if-convert -fno-tree-loop-if-convert-stores -fno-tree-loop-im -fno-tree-loop-ivcanon -fno-tree-loop-optimize -fno-tree-lrs -fno-tree-phi-prop -fno-tree-pre -fno-tree-pta -fno-tree-reassoc -fno-tree-scev-cprop -fno-tree-sink -fno-tree-slp-vectorize -fno-tree-sra -fno-tree-switch-conversion -ftree-ter -fno-tree-vect-loop-version -fno-tree-vectorize -fno-tree-vrp -fno-unroll-all-loops -fno-unsafe-loop-optimizations -fno-unsafe-math-optimizations -funswitch-loops -fno-variable-expansion-in-unroller -fno-vect-cost-model -fno-web

**Not very useful for analysis**

# Universal complexity (dimension) reduction

## Found solution

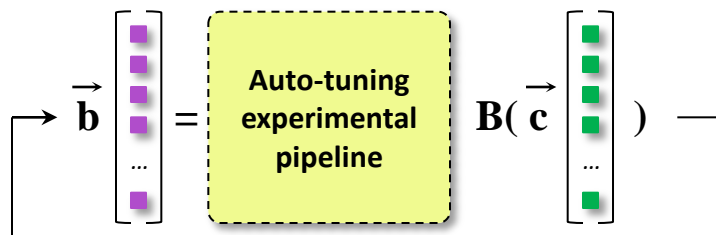
-O3 -fno-align-functions -fno-align-jumps -fno-align-labels -fno-align-loops -fno-asynchronous-unwind-tables -fno-branch-count-reg -fno-branch-target-load-optimize2 -fno-btr-bb-exclusive -fno-caller-saves -fno-combine-stack-adjustments -fno-common -fno-compare-elim -fno-conserve-stack -fno-cprop-registers -fno-crossjumping -fno-cse-follow-jumps -fno-cx-limited-range -fdce -fno-defer-pop -fno-delete-null-pointer-checks -fno-devirtualize -fno-dse -fno-early-inlining -fno-expensive-optimizations -fno-forward-propagate -fgcse -fno-gcse-after-reload -fno-gcse-las -fno-gcse-lm -fno-gcse-sm -fno-graphite-identity -fguess-branch-probability -fno-if-conversion -fno-if-conversion2 -fno-inline-functions -fno-inline-functions-called-once -fno-inline-small-functions -fno-ipa-cp -fno-ipa-cp-clone -fno-ipa-matrix-reorg -fno-ipa-profile -fno-ipa-pta -fno-ipa-pure-const -fno-ipa-reference -fno-ipa-sra -fno-ivopts -fno-jump-tables -fno-math-errno -fno-loop-block -fno-loop-flatten -fno-loop-interchange -fno-loop-parallelize-all -fno-loop-strip-mine -fno-merge-constants -fno-modulo-sched -fmove-loop-invariants -fomit-frame-pointer -fno-optimize-register-move -fno-optimize-sibling-calls -fno-peel-loops -fno-peephole -fno-peephole2 -fno-predictive-commoning -fno-prefetch-loop-arrays -fno-regmove -fno-rename-registers -fno-reorder-blocks -fno-reorder-blocks-and-partition -fno-reorder-functions -fno-rerun-cse-after-loop -fno-reschedule-modulo-scheduled-loops -fno-sched-critical-path-heuristic -fno-sched-dep-count-heuristic -fno-sched-group-heuristic -fno-sched-interblock -fno-sched-last-insn-heuristic -fno-sched-pressure -fno-sched-rank-heuristic -fno-sched-spec -fno-sched-spec-insn-heuristic -fno-sched-spec-load -fno-sched-spec-load-dangerous -fno-sched-stalled-insns -fno-sched-stalled-insns-dep -fno-sched2-use-superblocks -fno-schedule-insns -fno-schedule-insns2 -fno-short-enums -fno-signed-zeros -fno-sel-sched-pipelining -fno-sel-sched-pipelining-outer-loops -fno-sel-sched-reschedule-pipelined -fno-selective-scheduling -fno-selective-scheduling2 -fno-signaling-nans -fno-single-precision-constant -fno-split-ivs-in-unroller -fno-split-wide-types -fno-strict-aliasing -fno-thread-jumps -fno-trapping-math -fno-tree-bit-ccp -fno-tree-builtin-call-dce -fno-tree-ccp -fno-tree-ch -fno-tree-copy-prop -fno-tree-copyrename -fno-tree-cselim -fno-tree-dce -fno-tree-dominator-opts -fno-tree-dse -ftree-forwprop -fno-tree-fre -fno-tree-loop-distribute-patterns -fno-tree-loop-distribution -fno-tree-loop-if-convert -fno-tree-loop-if-convert-stores -fno-tree-loop-im -fno-tree-loop-ivcanon -fno-tree-loop-optimize -fno-tree-lrs -fno-tree-phi-prop -fno-tree-pre -fno-tree-pta -fno-tree-reassoc -fno-tree-scev-cprop -fno-tree-sink -fno-tree-slp-vectorize -fno-tree-sra -fno-tree-switch-conversion -ftree-ter -fno-tree-vect-loop-version -fno-tree-vectorize -fno-tree-vrp -fno-unroll-all-loops -fno-unsafe-loop-optimizations -fno-unsafe-math-optimizations -funswitch-loops -fno-variable-expansion-in-unroller -fno-vect-cost-model -fno-web



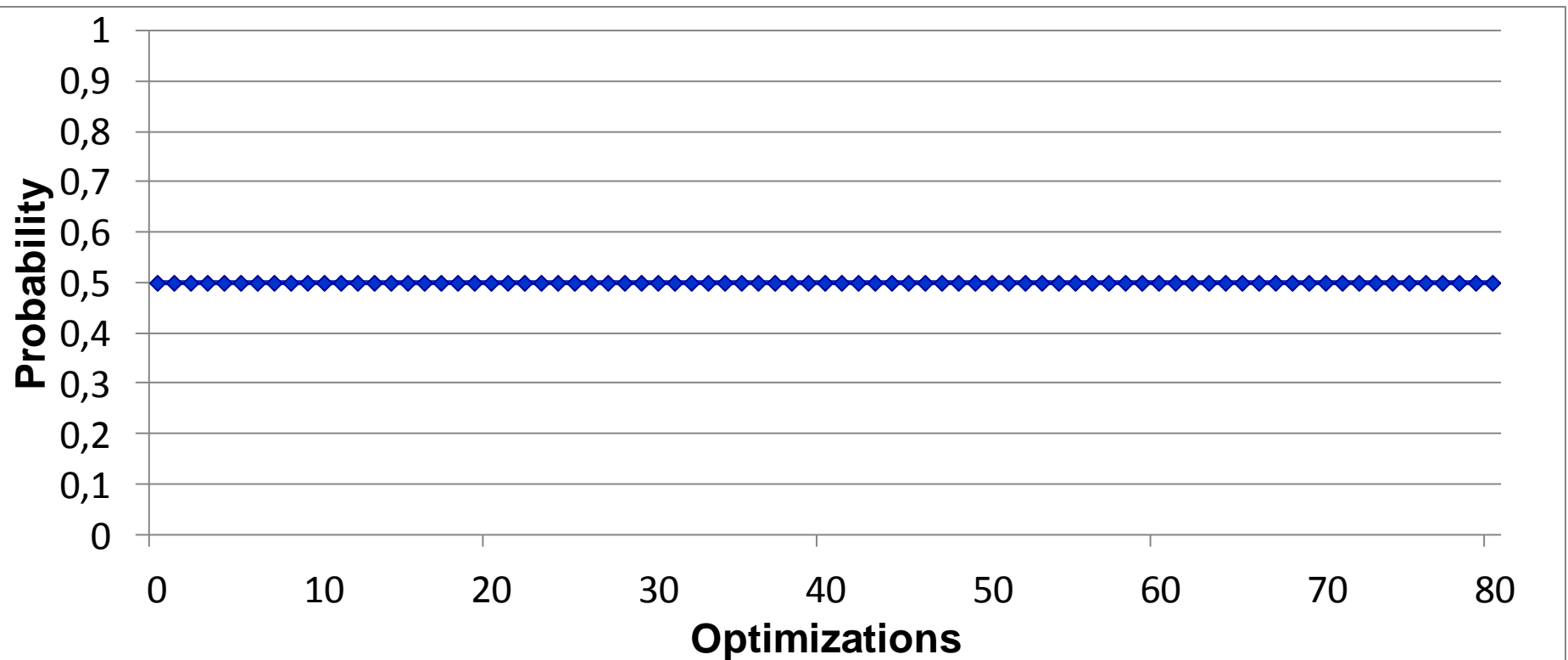
## Chain complexity reduction filter

remove dimensions (or set to default)

iteratively, ANOVA, PCA, etc...



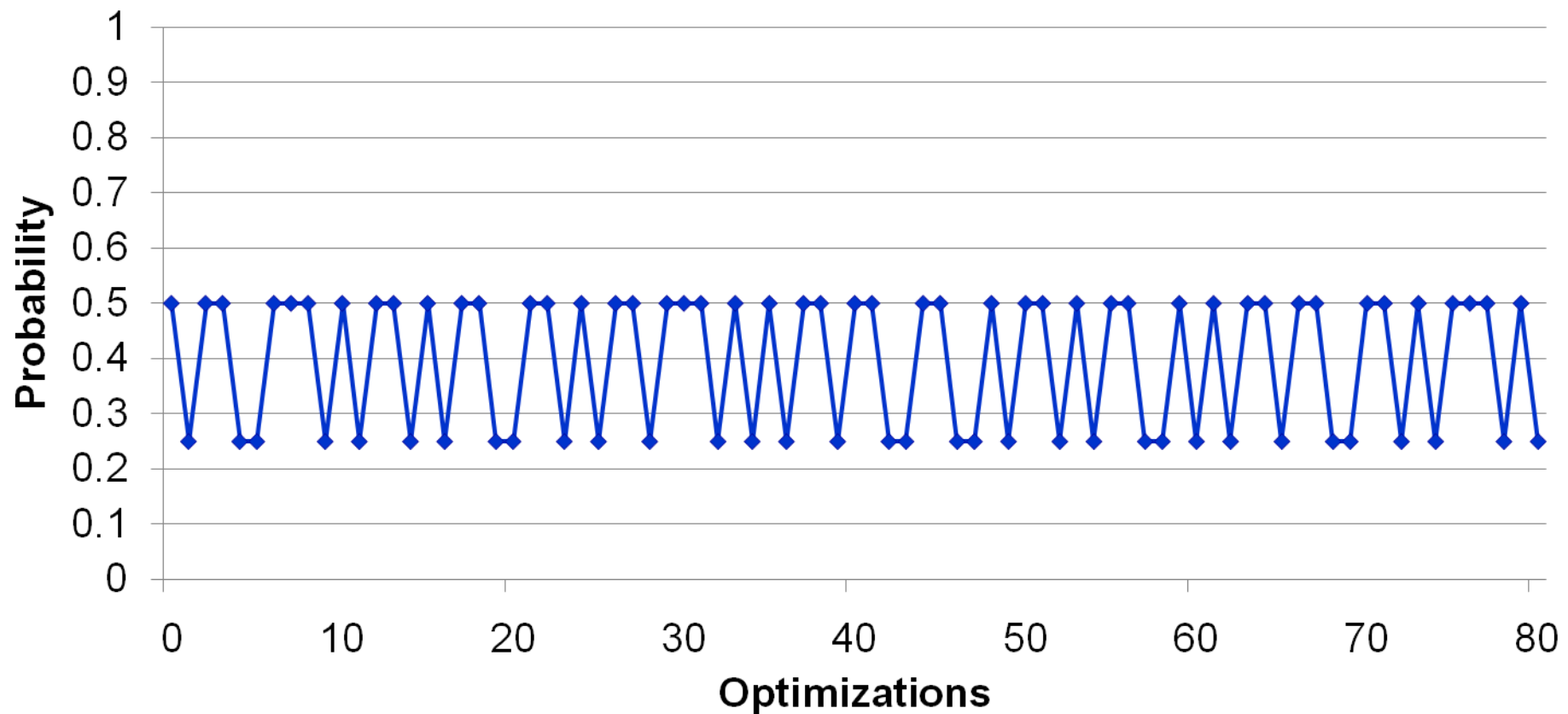
# Active learning to systematize and focus exploration



**Start: 50% probability to select optimization (uniform distribution)**

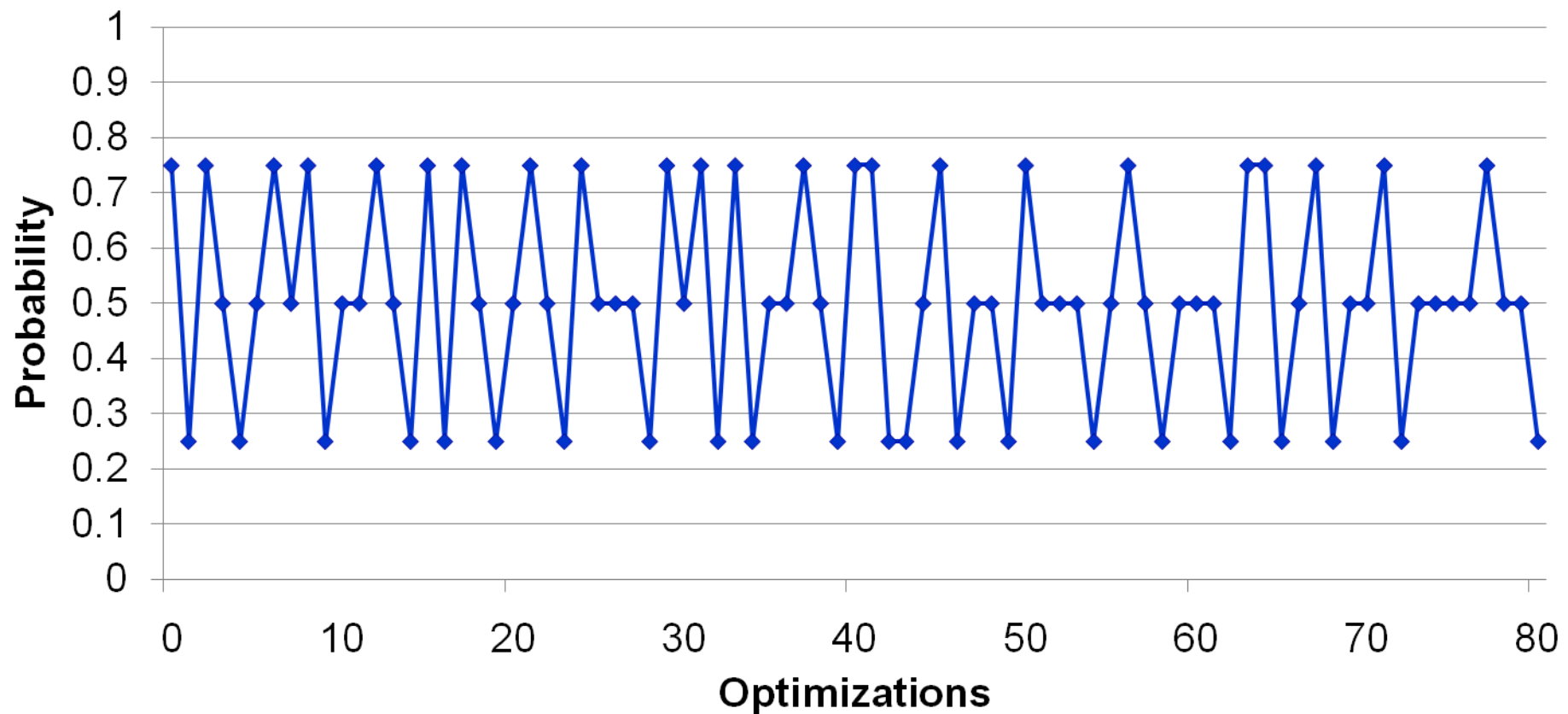
**Avoiding collection of huge amount of data -  
filtering (compacting) and learning space on the fly**

# Active learning to systematize and focus exploration



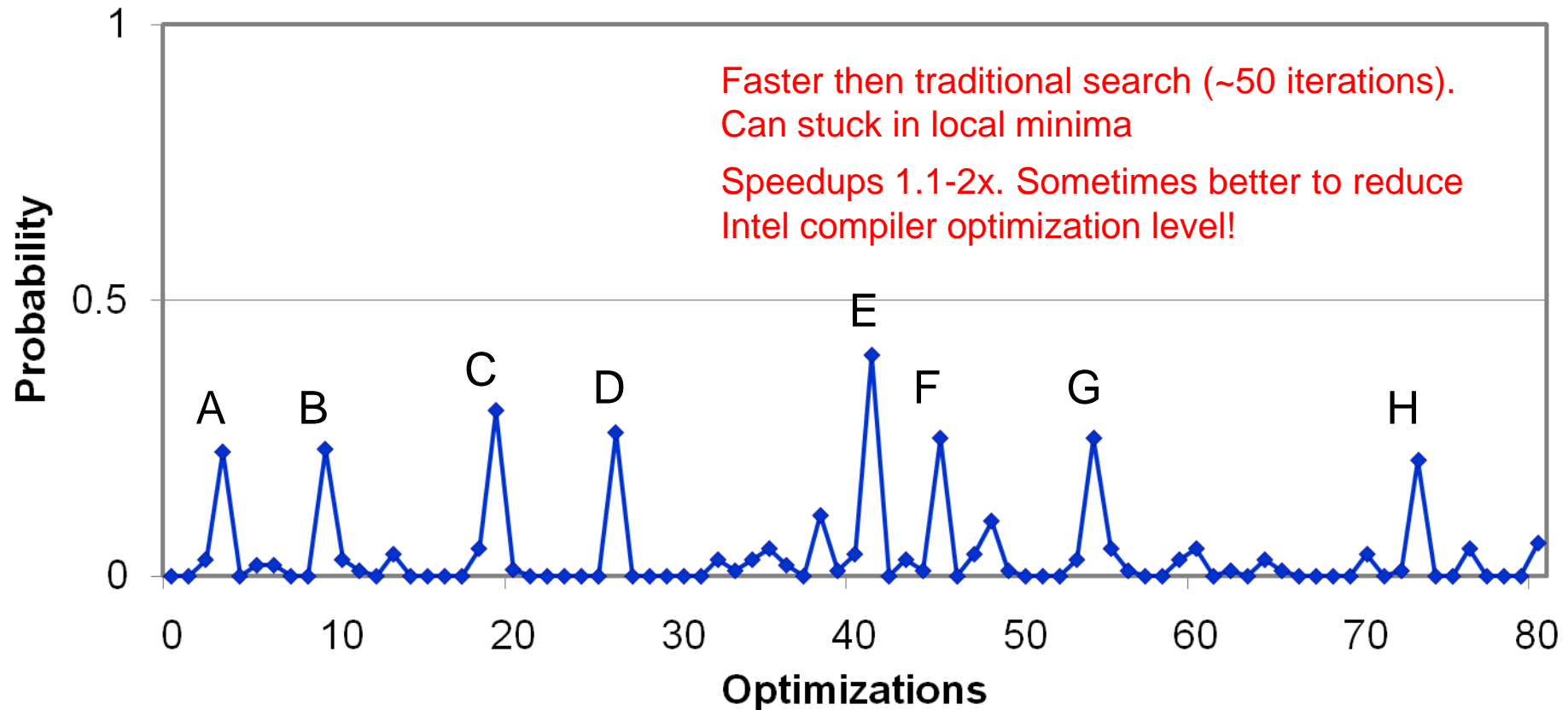
**Current random selection of optimizations increased execution time (bad):**  
*reduce probabilities of the selected optimizations*

# Active learning to systematize and focus exploration



**Current random selection of optimizations improved execution time (good):**  
*reward probabilities of the selected optimizations*

# Active learning to systematize and focus exploration



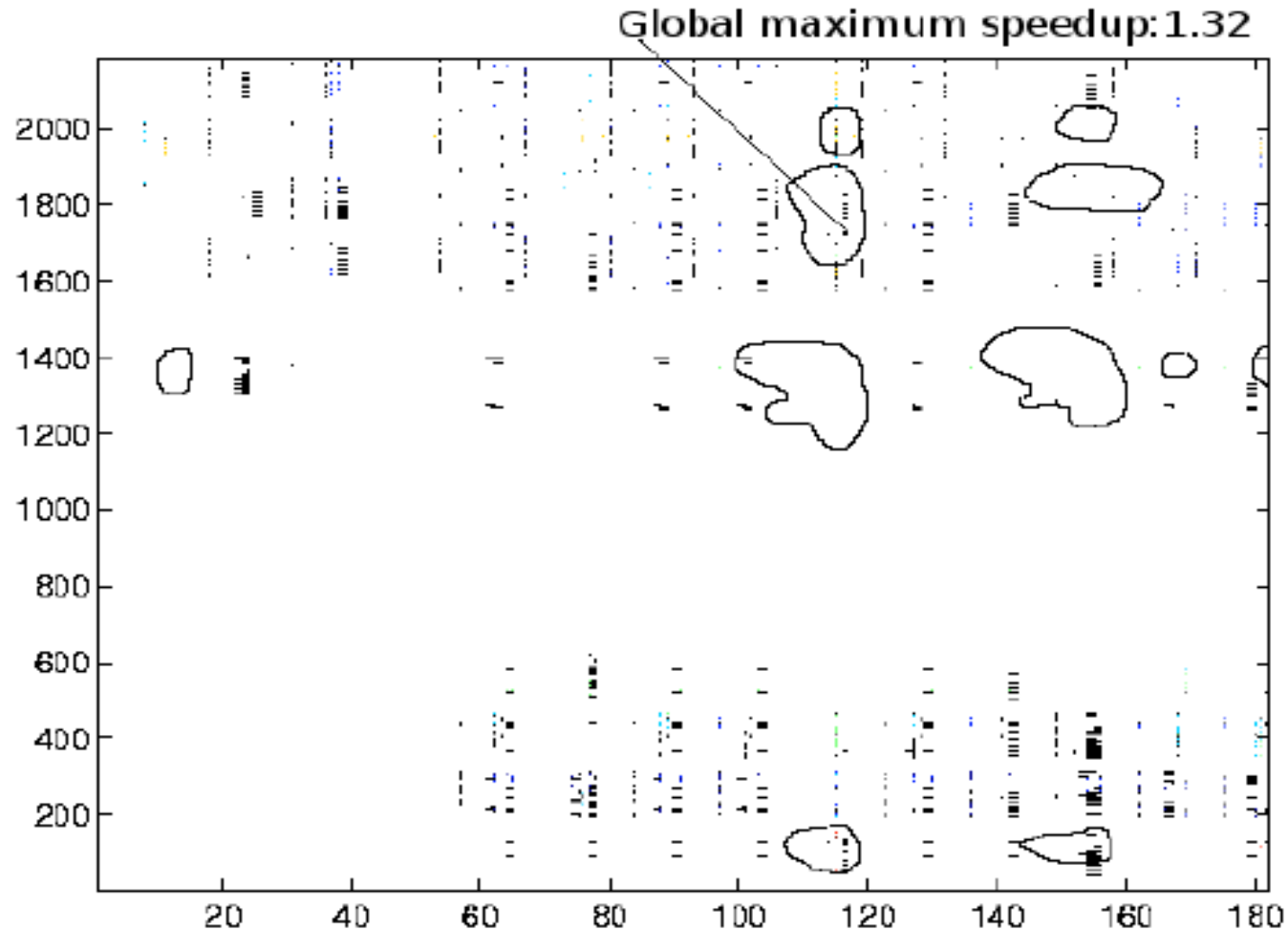
*“good optimizations” across all programs:*

A – Break up large expression trees  
B – Value propagation  
C – Hoisting of loop invariants  
D – Loop normalization

E – Loop unrolling  
F – Mark constant variables  
G – Dismantle array instructions  
H – Eliminating copies



# Active learning to systematize and focus exploration



14 transformations, sequences of length 5, search space = **396000**

- 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*

# Universal complexity (dimension) reduction

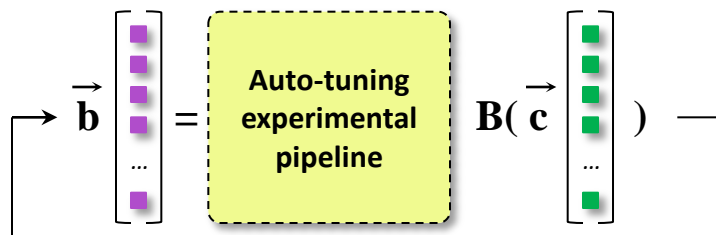
## Found solution

-O3 -fno-align-functions -fno-align-jumps -fno-align-labels -fno-align-loops -fno-asynchronous-unwind-tables -fno-branch-count-reg -fno-branch-target-load-optimize2 -fno-btr-bb-exclusive -fno-caller-saves -fno-combine-stack-adjustments -fno-common -fno-compare-elim -fno-conserve-stack -fno-cprop-registers -fno-crossjumping -fno-cse-follow-jumps -fno-cx-limited-range -fdce -fno-defer-pop -fno-delete-null-pointer-checks -fno-devirtualize -fno-dse -fno-early-inlining -fno-expensive-optimizations -fno-forward-propagate -fgcse -fno-gcse-after-reload -fno-gcse-las -fno-gcse-lm -fno-gcse-sm -fno-graphite-identity -fguess-branch-probability -fno-if-conversion -fno-if-conversion2 -fno-inline-functions -fno-inline-functions-called-once -fno-inline-small-functions -fno-ipa-cp -fno-ipa-cp-clone -fno-ipa-matrix-reorg -fno-ipa-profile -fno-ipa-pta -fno-ipa-pure-const -fno-ipa-reference -fno-ipa-sra -fno-ivopts -fno-jump-tables -fno-math-errno -fno-loop-block -fno-loop-flatten -fno-loop-interchange -fno-loop-parallelize-all -fno-loop-strip-mine -fno-merge-constants -fno-modulo-sched -fmove-loop-invariants -fomit-frame-pointer -fno-optimize-register-move -fno-optimize-sibling-calls -fno-peel-loops -fno-peephole -fno-peephole2 -fno-predictive-commoning -fno-prefetch-loop-arrays -fno-regmove -fno-rename-registers -fno-reorder-blocks -fno-reorder-blocks-and-partition -fno-reorder-functions -fno-rerun-cse-after-loop -fno-reschedule-modulo-scheduled-loops -fno-sched-critical-path-heuristic -fno-sched-dep-count-heuristic -fno-sched-group-heuristic -fno-sched-interblock -fno-sched-last-insn-heuristic -fno-sched-pressure -fno-sched-rank-heuristic -fno-sched-spec -fno-sched-spec-insn-heuristic -fno-sched-spec-load -fno-sched-spec-load-dangerous -fno-sched-stalled-insns -fno-sched-stalled-insns-dep -fno-sched2-use-superblocks -fno-schedule-insns -fno-schedule-insns2 -fno-short-enums -fno-signed-zeros -fno-sel-sched-pipelining -fno-sel-sched-pipelining-outer-loops -fno-sel-sched-reschedule-pipelined -fno-selective-scheduling -fno-selective-scheduling2 -fno-signaling-nans -fno-single-precision-constant -fno-split-ivs-in-unroller -fno-split-wide-types -fno-strict-aliasing -fno-thread-jumps -fno-trapping-math -fno-tree-bit-ccp -fno-tree-builtin-call-dce -fno-tree-ccp -fno-tree-ch -fno-tree-copy-prop -fno-tree-copyrename -fno-tree-cselim -fno-tree-dce -fno-tree-dominator-opts -fno-tree-dse -ftree-forwprop -fno-tree-fre -fno-tree-loop-distribute-patterns -fno-tree-loop-distribution -fno-tree-loop-if-convert -fno-tree-loop-if-convert-stores -fno-tree-loop-im -fno-tree-loop-ivcanon -fno-tree-loop-optimize -fno-tree-lrs -fno-tree-phi-prop -fno-tree-pre -fno-tree-pta -fno-tree-reassoc -fno-tree-scev-cprop -fno-tree-sink -fno-tree-slp-vectorize -fno-tree-sra -fno-tree-switch-conversion -ftree-ter -fno-tree-vec-loop-version -fno-tree-vectorize -fno-tree-vrp -fno-unroll-all-loops -fno-unsafe-loop-optimizations -fno-unsafe-math-optimizations -funswitch-loops -fno-variable-expansion-in-unroller -fno-vect-cost-model -fno-web



## Chain complexity reduction filter

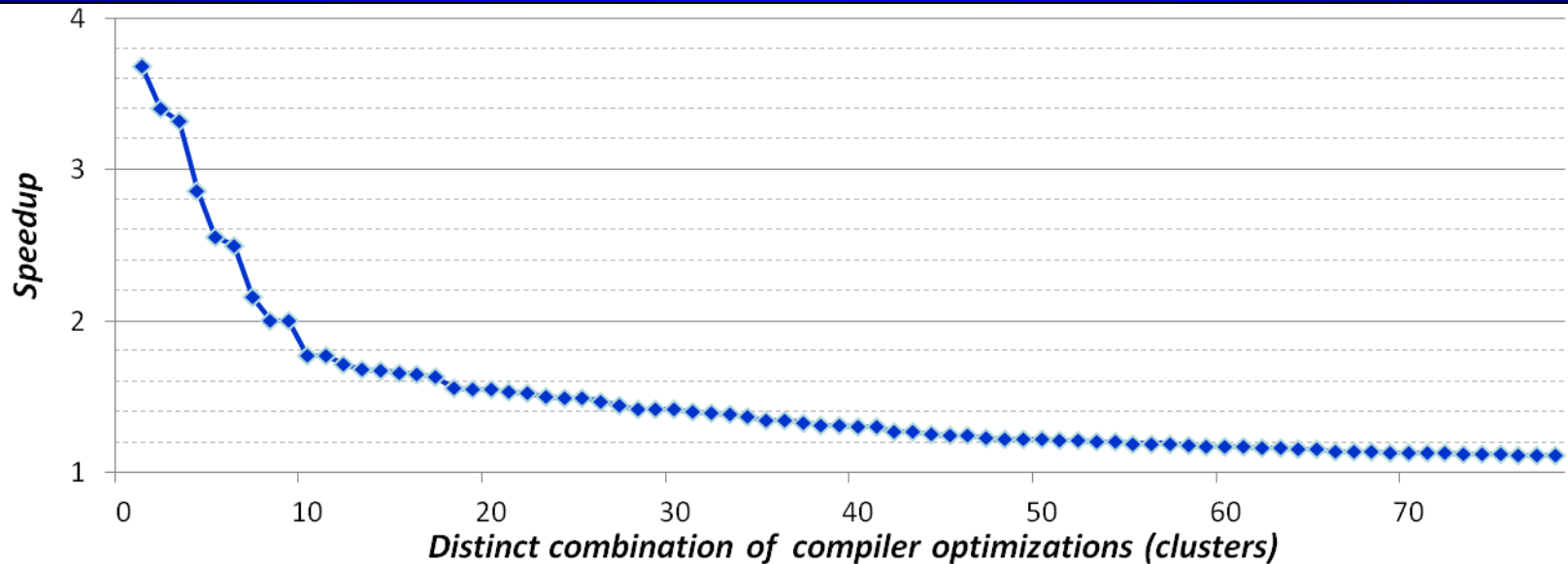
remove dimensions (or set to default)  
iteratively, ANOVA, PCA, etc...



## Pruned solution

-O3  
-fno-align-functions (15% of speedup)  
-fdce  
-fgcse  
-fguess-branch-probability (70% of speedup)  
-fmove-loop-invariants  
-fomit-frame-pointer  
-ftree-ter  
-funswitch-loops  
-fno-ALL

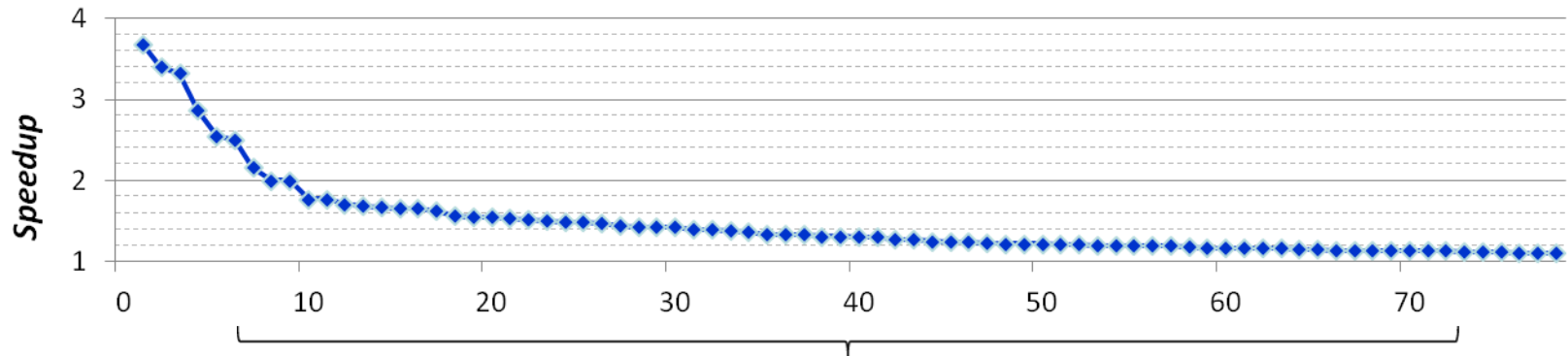
# Online optimization clustering



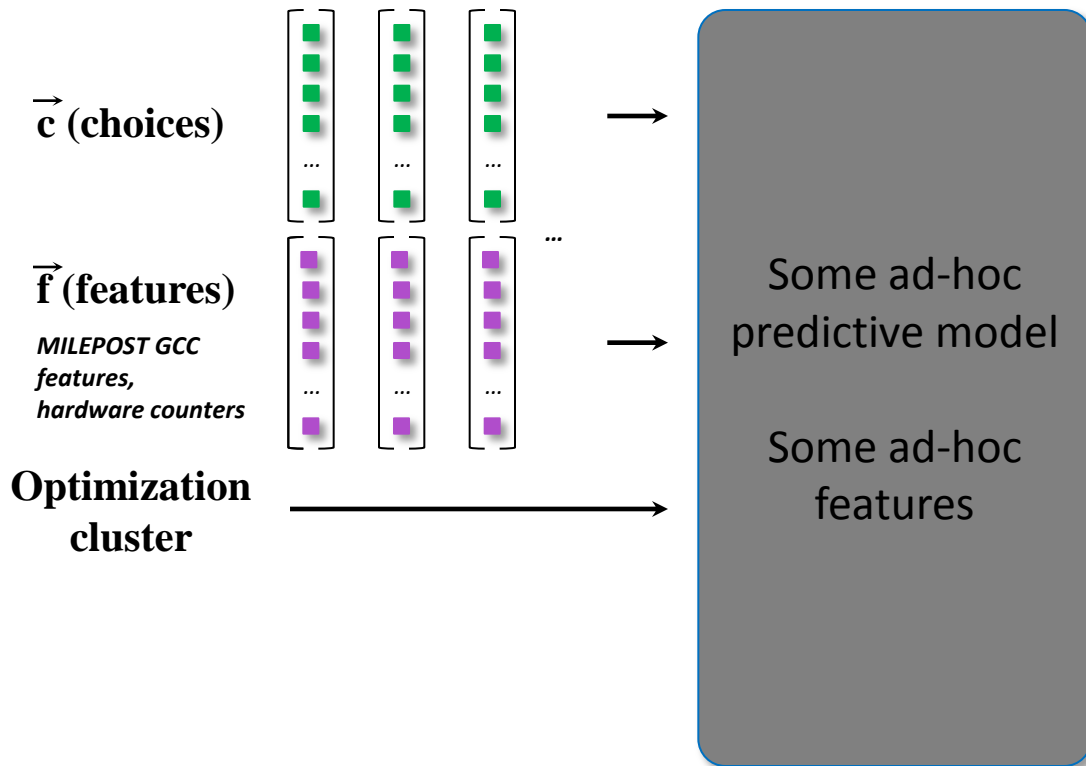
Continuously crowdtuning 285 shared code and dataset combinations from 8 benchmarks including NAS, MiBench, SPEC2000, SPEC2006, Powerstone, UTDSP and SNU-RT

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

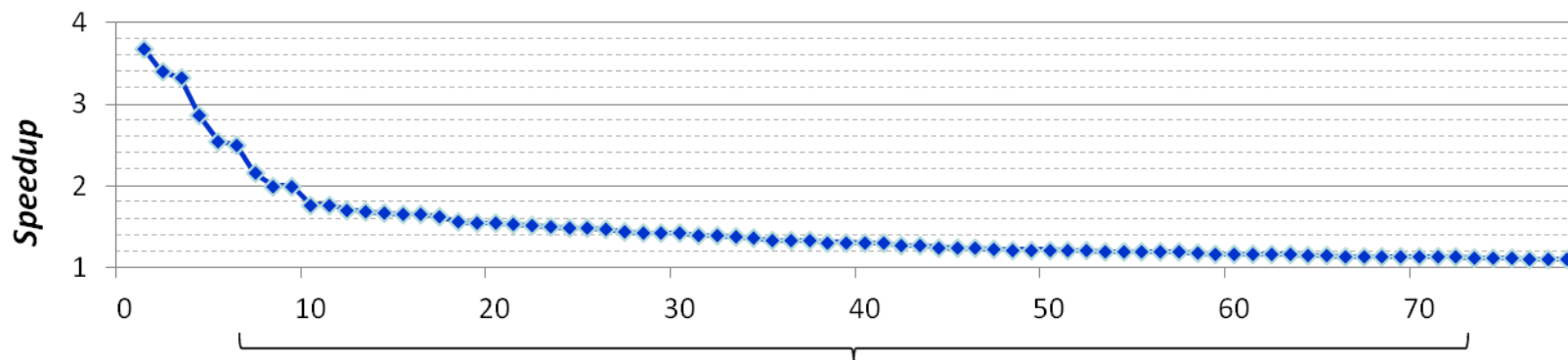
# Current machine learning usage



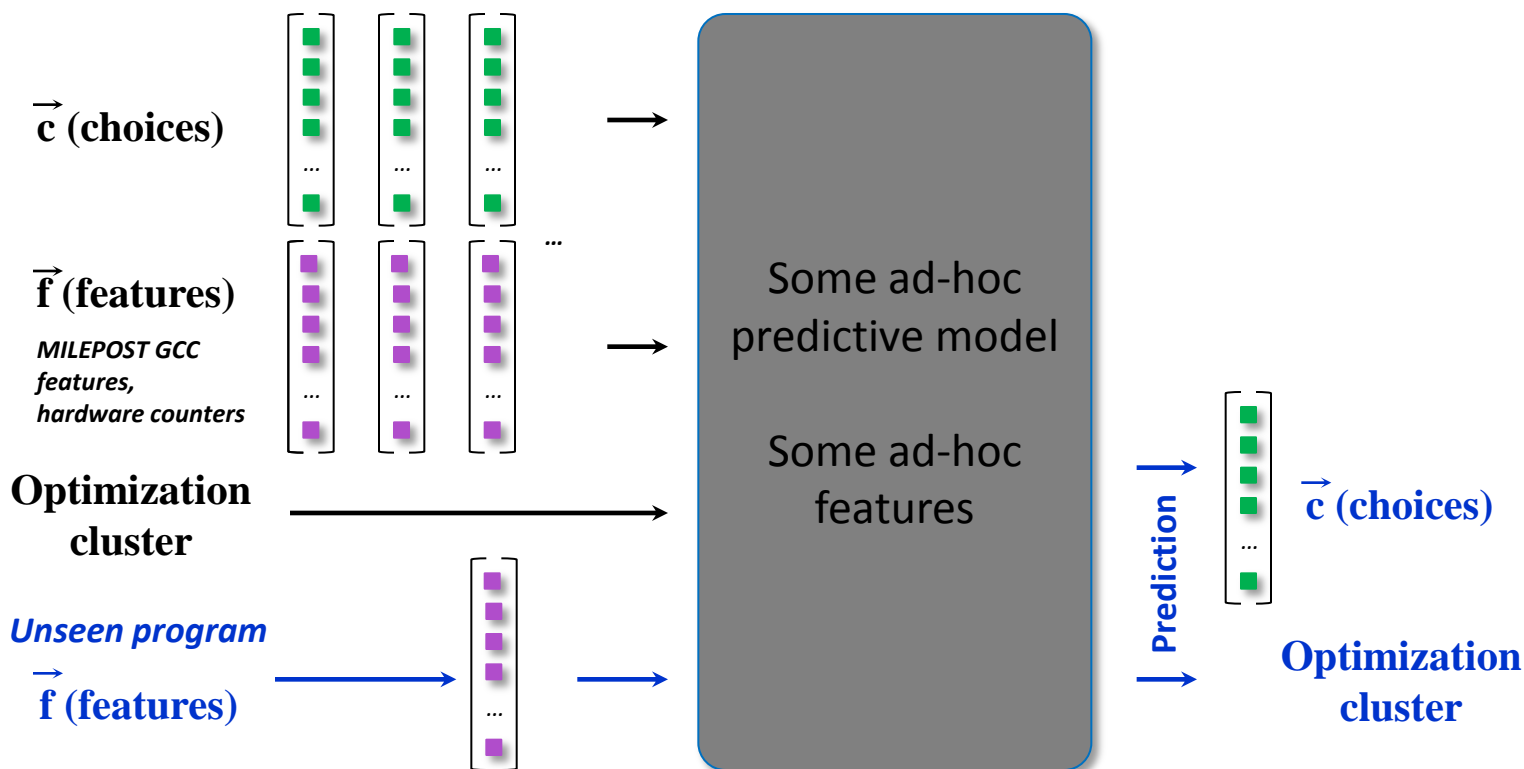
Training set: distinct combination of compiler optimizations (clusters)



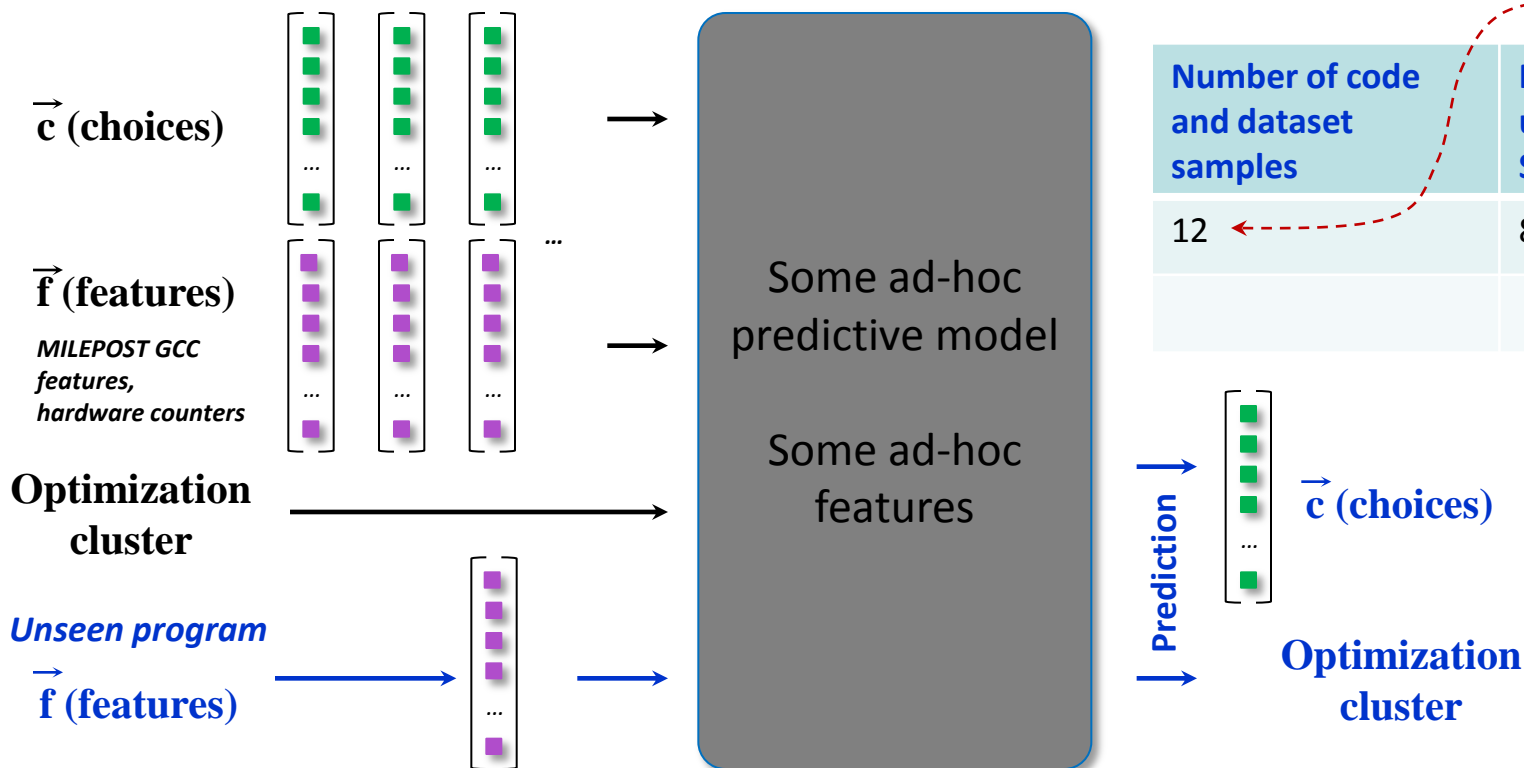
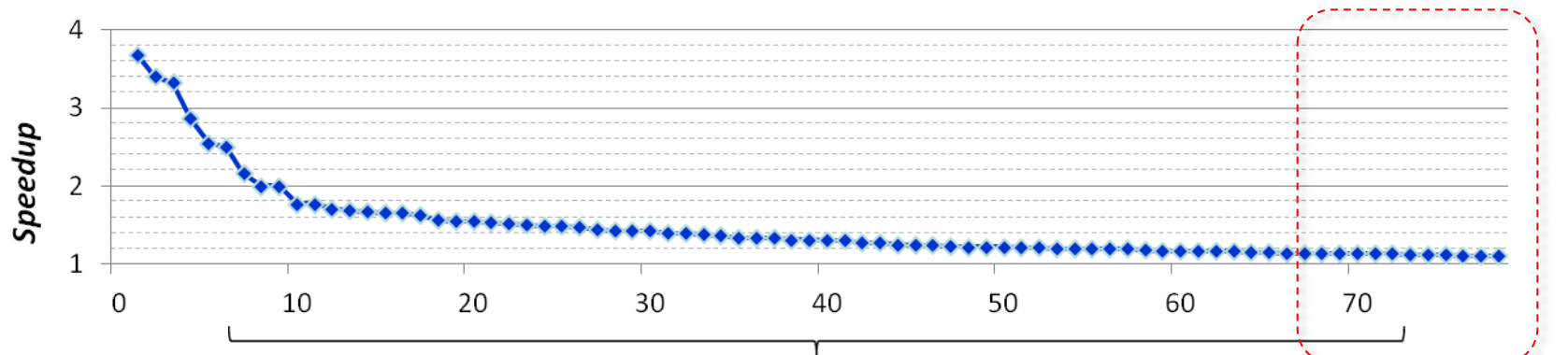
# Current machine learning usage



Training set: distinct combination of compiler optimizations (clusters)

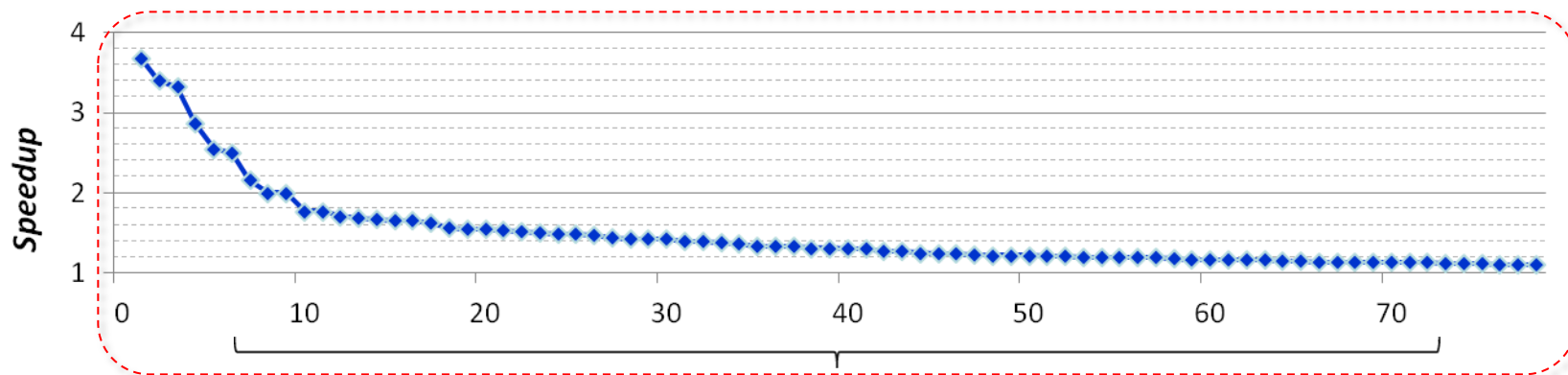


# Current machine learning usage

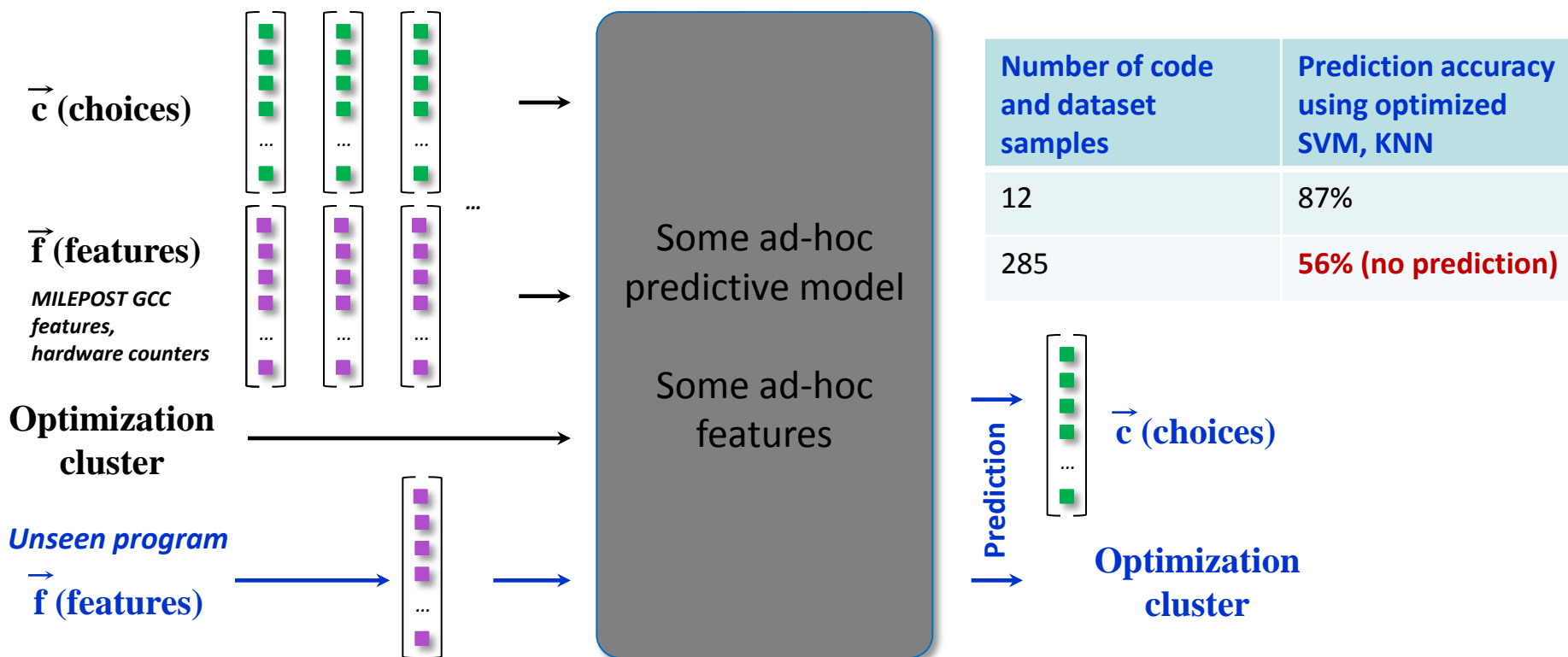


Number of code and dataset samples	Prediction accuracy using optimized SVM, KNN
12	87%

# Current machine learning usage



Training set: distinct combination of compiler optimizations (clusters)


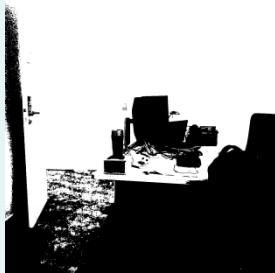

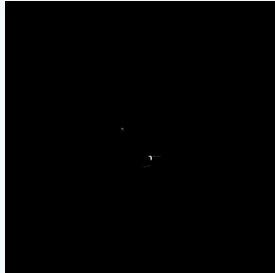




# Learning features by domain specialists

Image B&W threshold filter

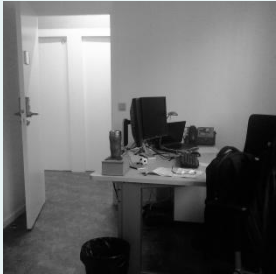
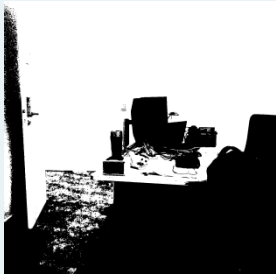

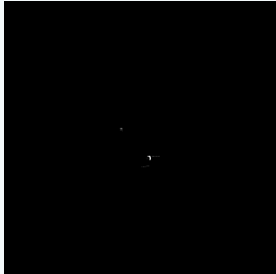
`*matrix_ptr2++ = (temp1 > T) ? 255 : 0;`

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

# Learning features by domain specialists

Image B&W threshold filter

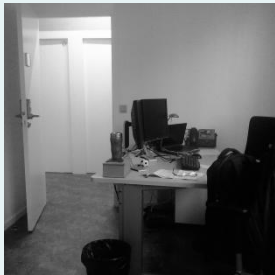
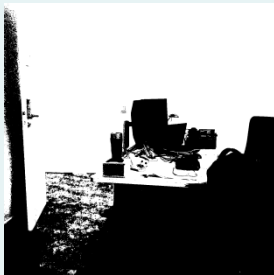

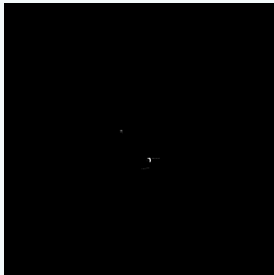
`*matrix_ptr2++ = (temp1 > T) ? 255 : 0;`

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

# Learning feature by domain specialists

Image B&W threshold filter

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

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

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!

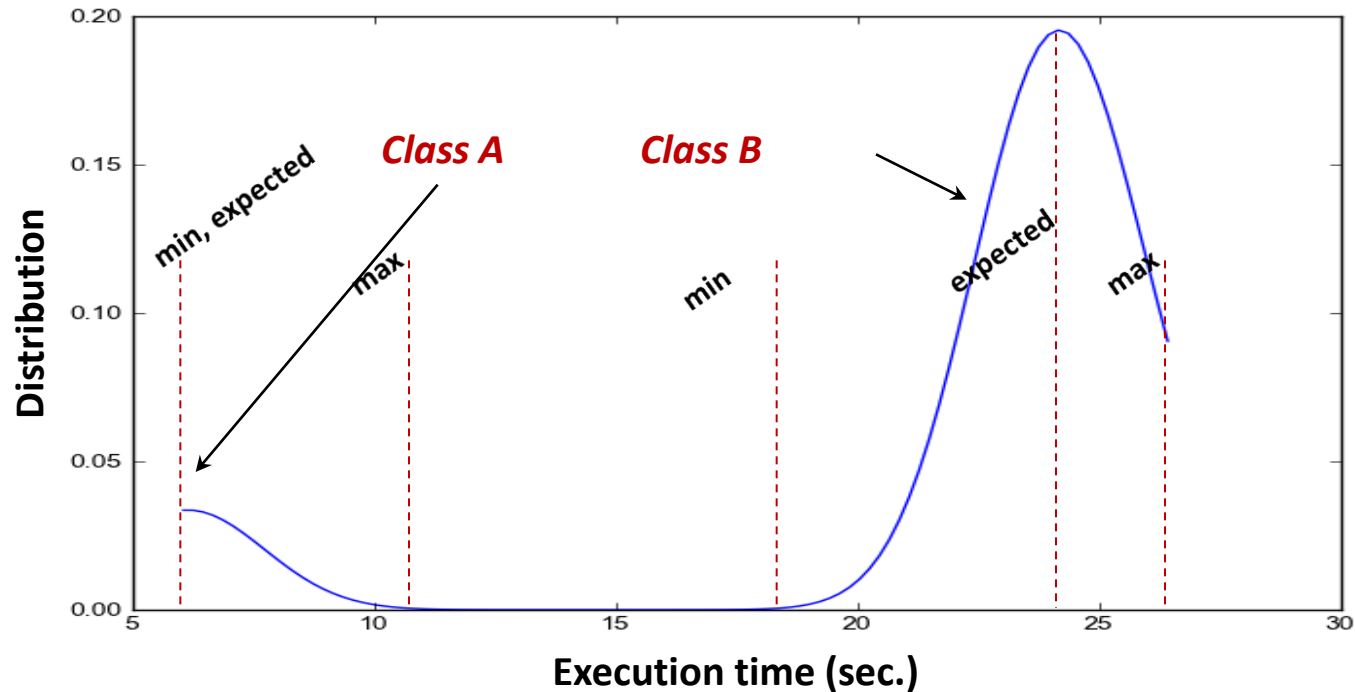
**Need split-compilation (cloning and run-time adaptation)**

if get\_feature(**TIME\_OF\_THE\_DAY**)==**NIGHT**  
else

bw\_filter\_codelet\_day(buffers);  
bw\_filter\_codelet\_night(buffers);

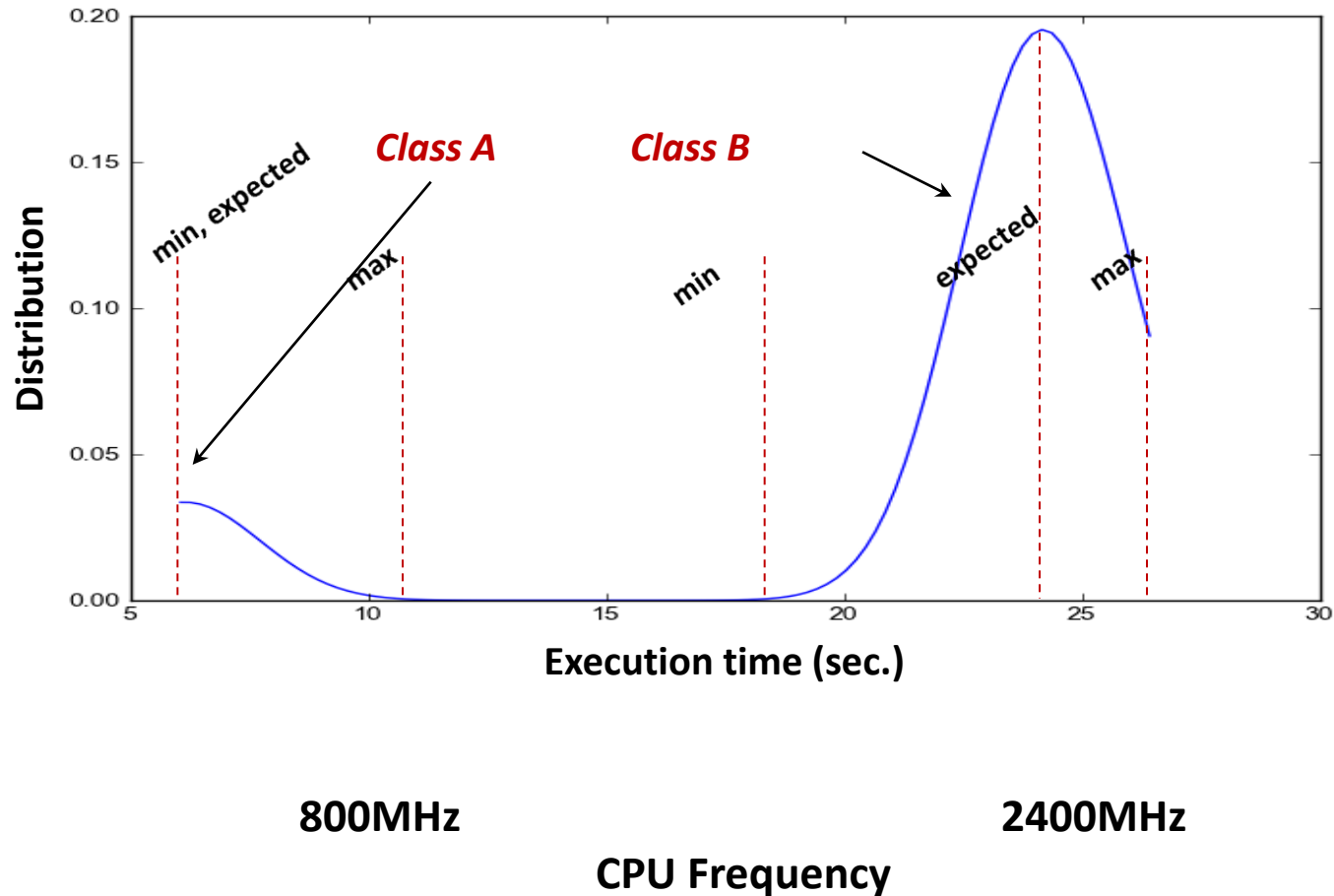
# Normality test plugin

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



# Normality test plugin

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

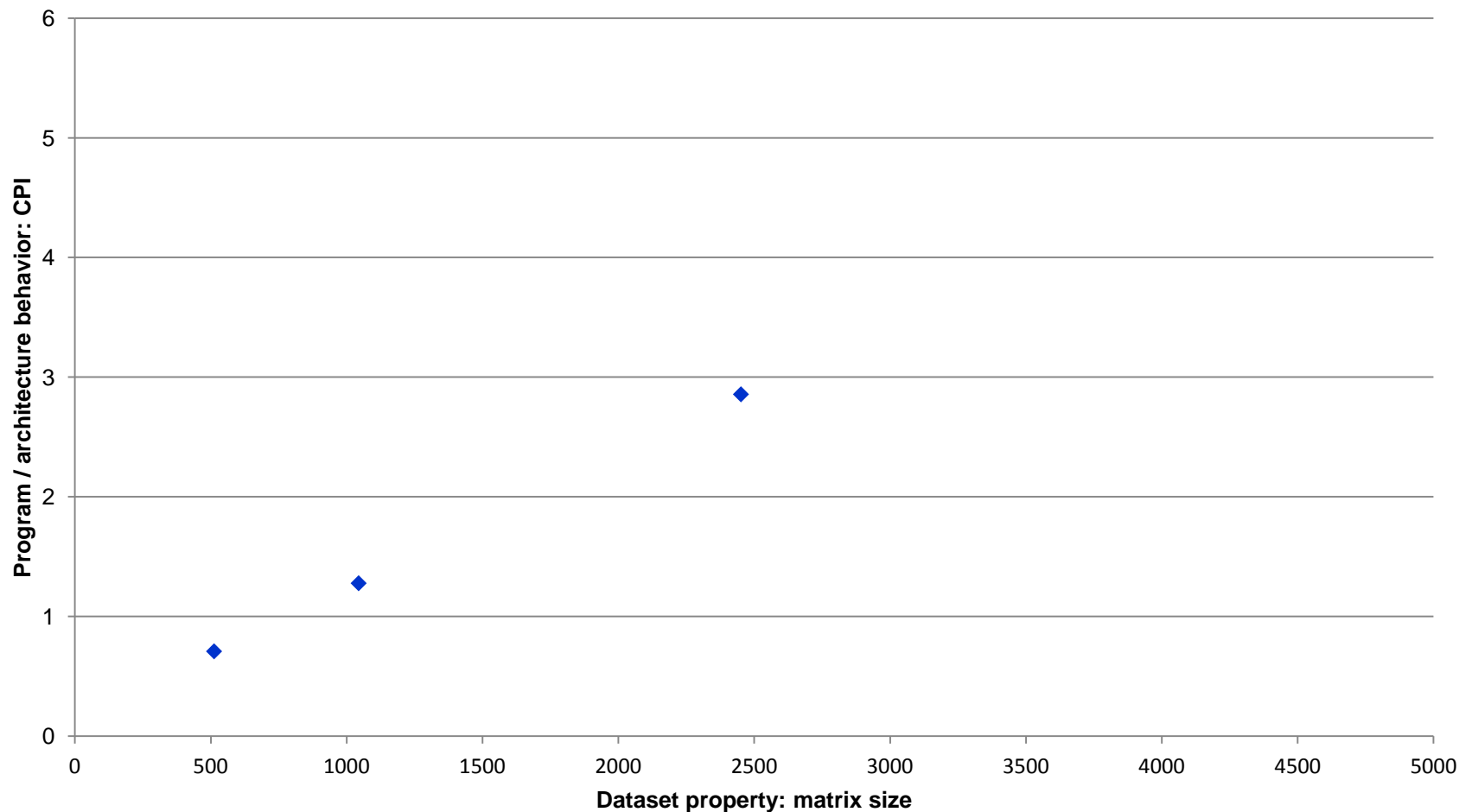


# Using Collective Mind to explore and learn behavior of computer systems

How we can explain the following observations for some piece of code (“codelet object”)?  
(LU-decomposition codelet, Intel Nehalem)



Add 1 property: **matrix size**

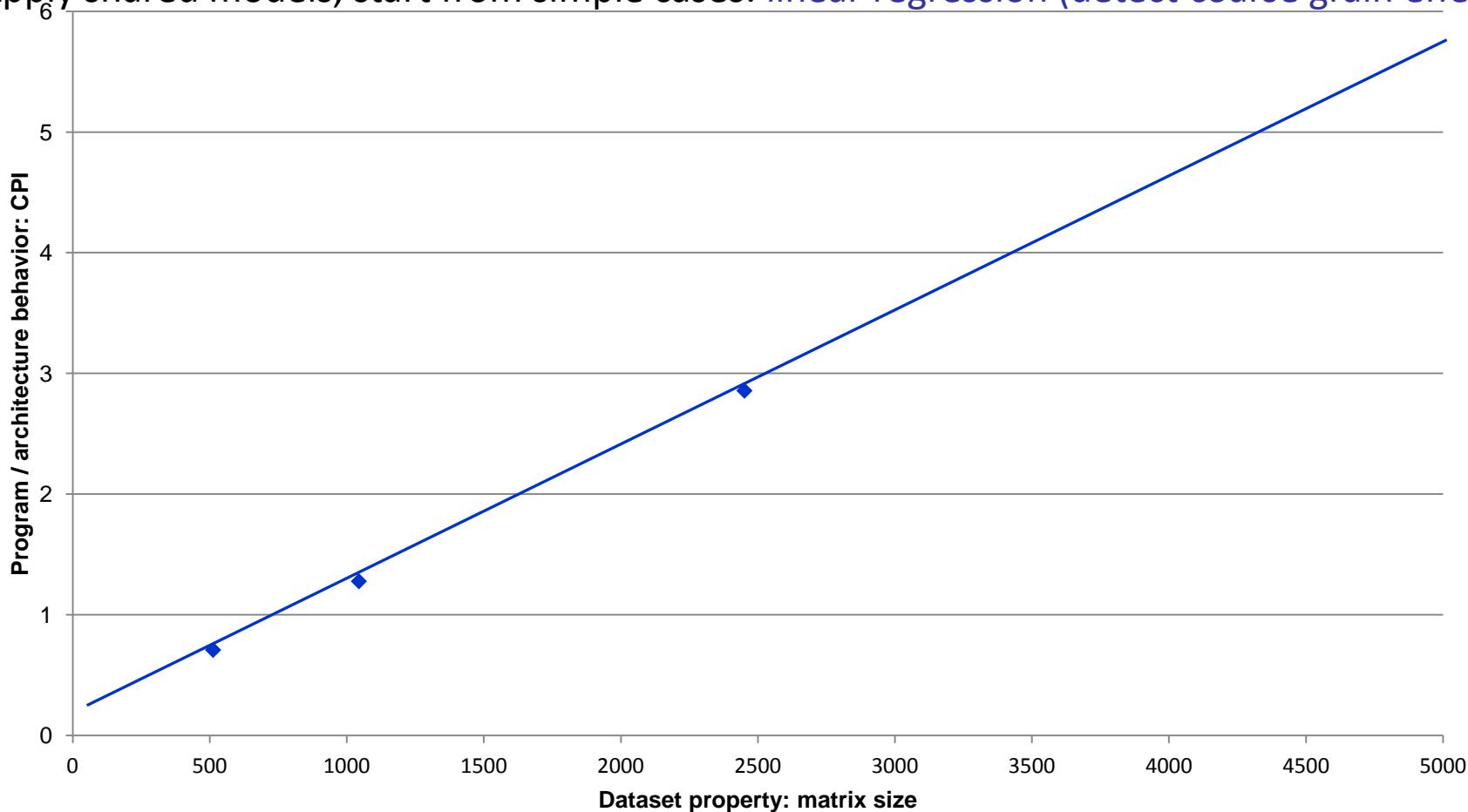




# Using Collective Mind to explore and learn behavior of computer systems

Either fit existing or build a new model to **correlate objectives** (CPI) and **features** (matrix size) while minimizing RMSE.

Apply shared models, start from simple cases: **linear regression** (detect coarse grain effects)

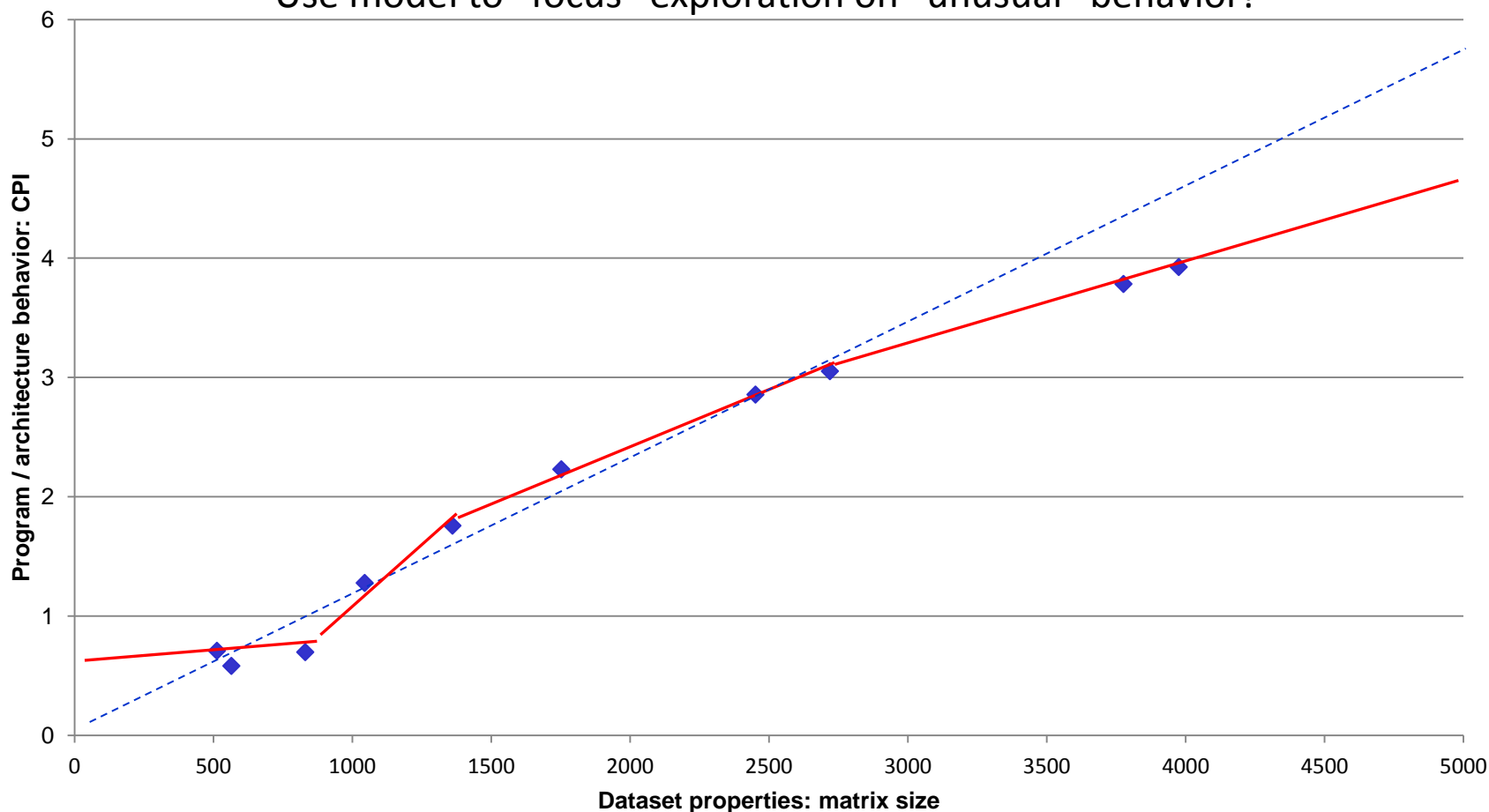


# Using Collective Mind to explore and learn behavior of computer systems

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

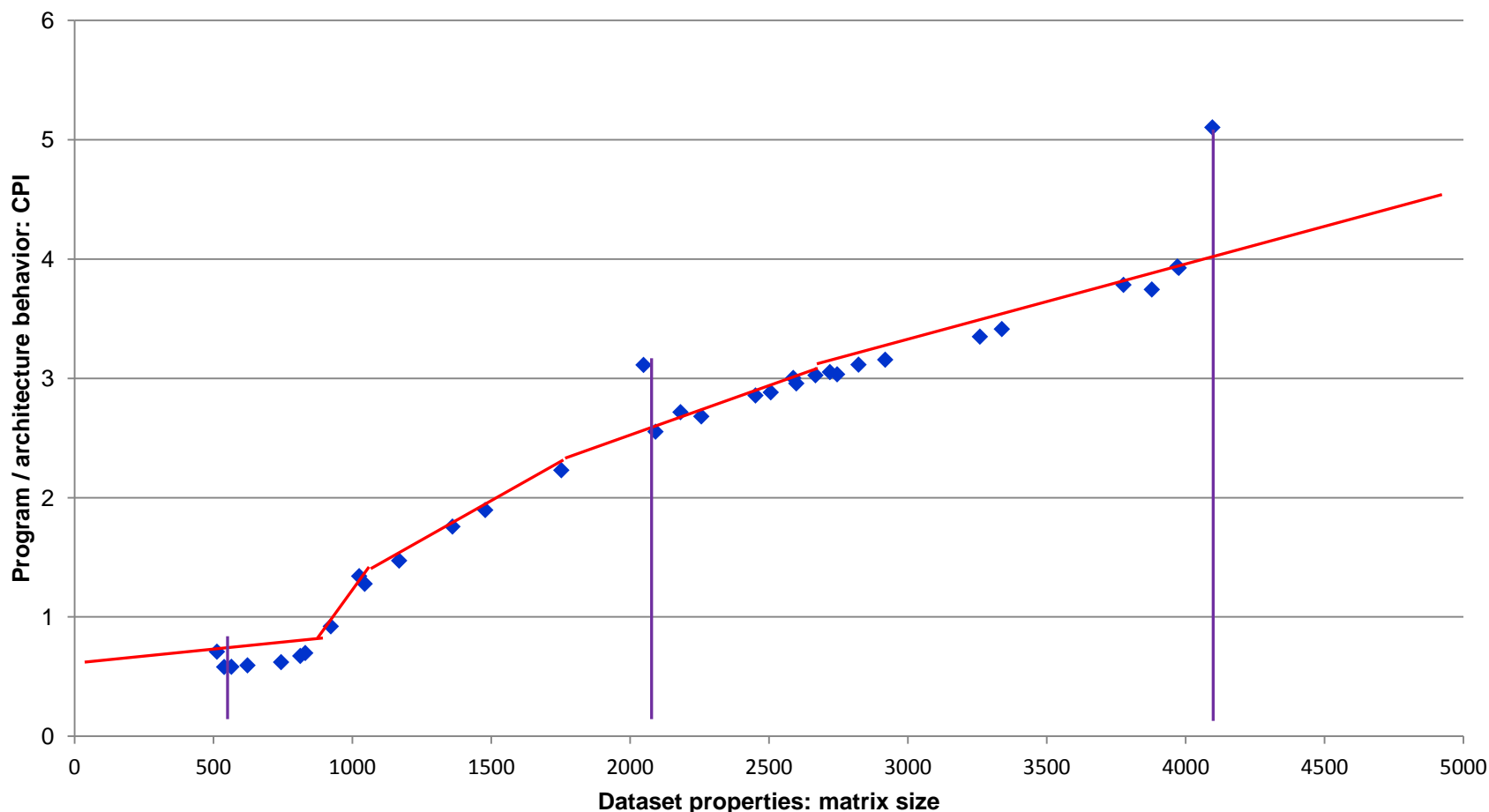
**Continuously retrain models** to fit new data!

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



# Using Collective Mind to explore and learn behavior of computer systems

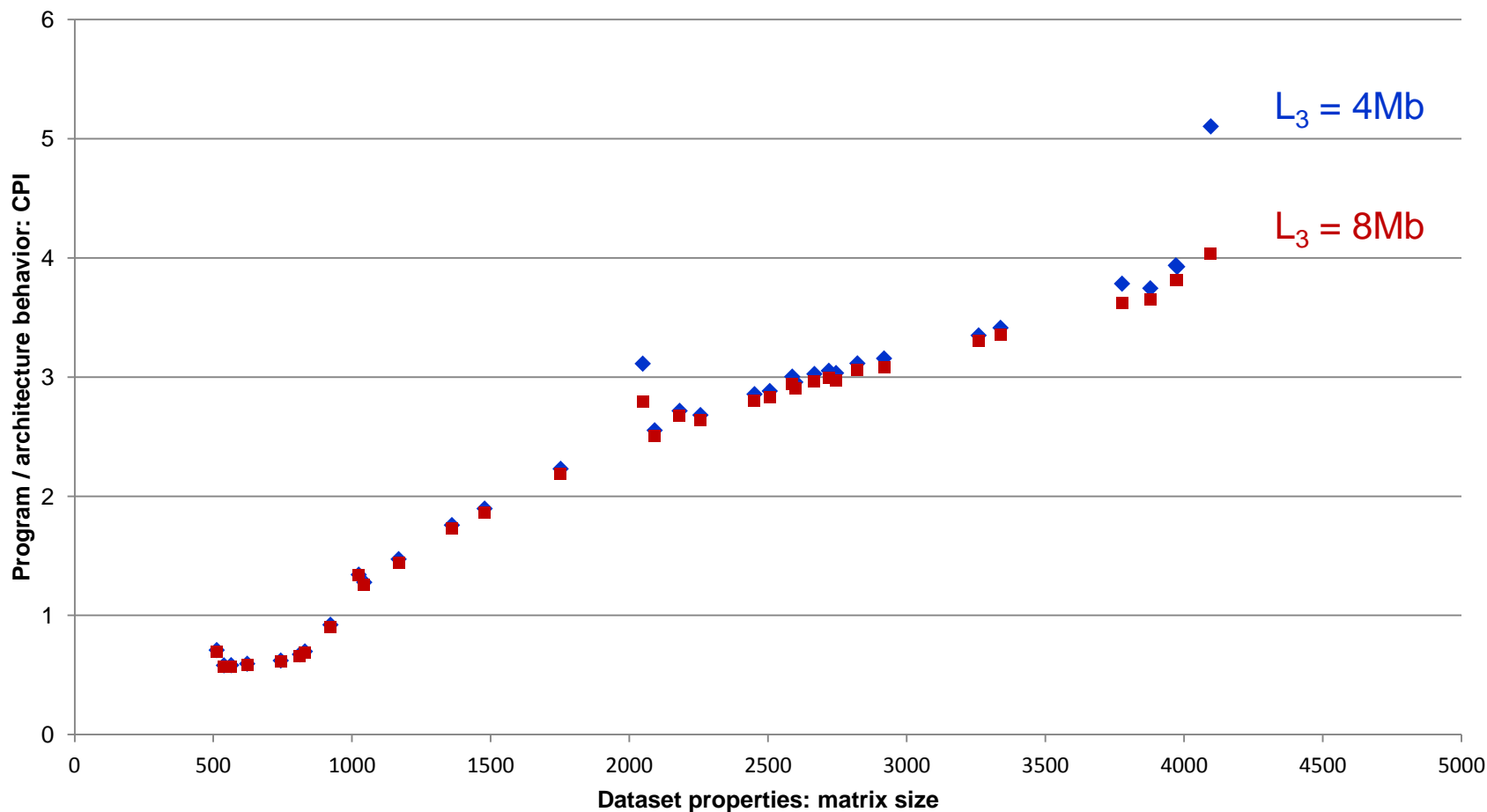
Gradually increase model complexity if needed ([hierarchical modeling](#)).  
For example, detect [fine-grain effects](#) ([singularities](#)) and [characterize](#) them.



# Using Collective Mind to explore and learn behavior of computer systems

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

Use automatic approach to correlate all objectives and features.

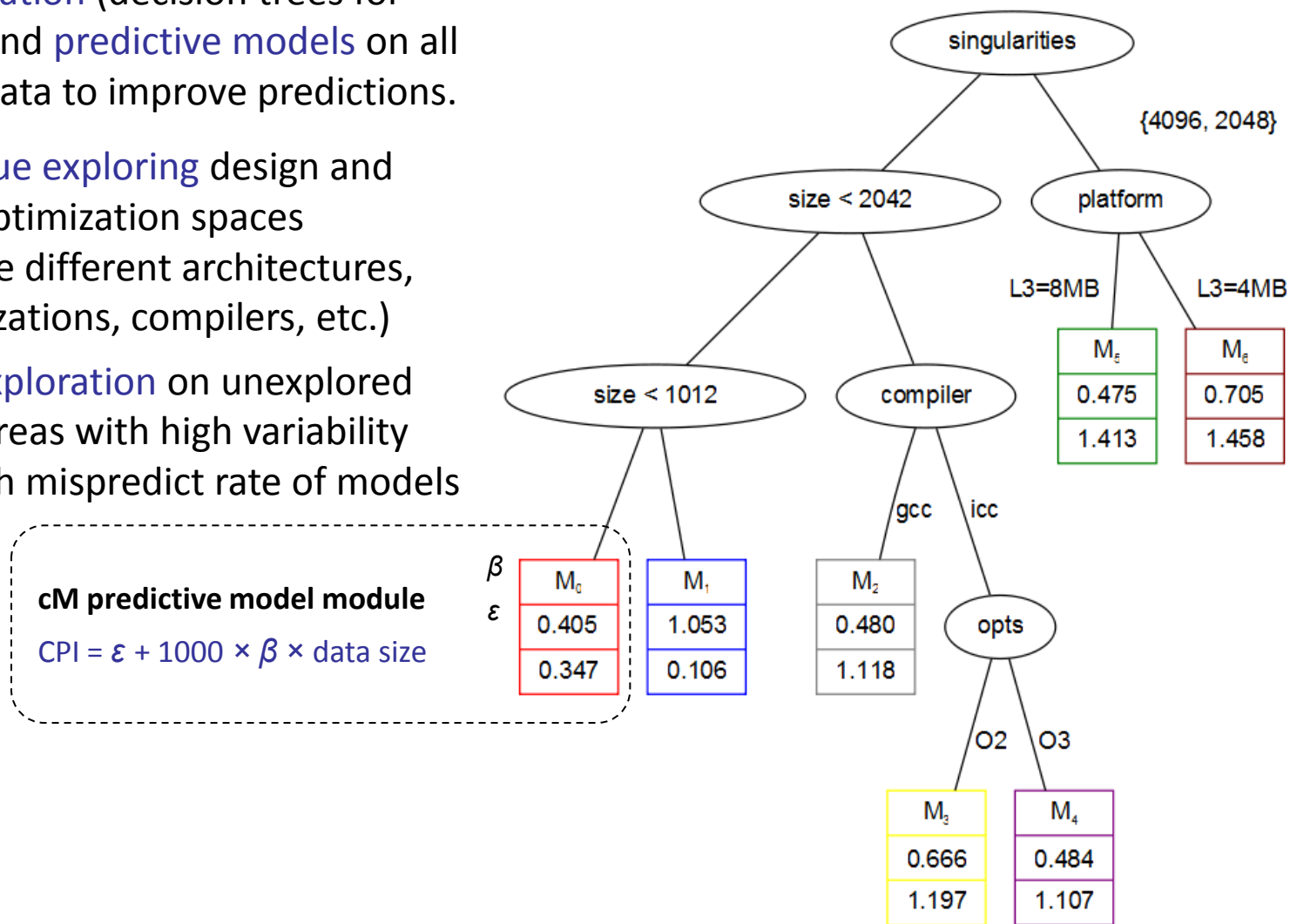


# Using Collective Mind to explore and learn behavior of computer systems

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

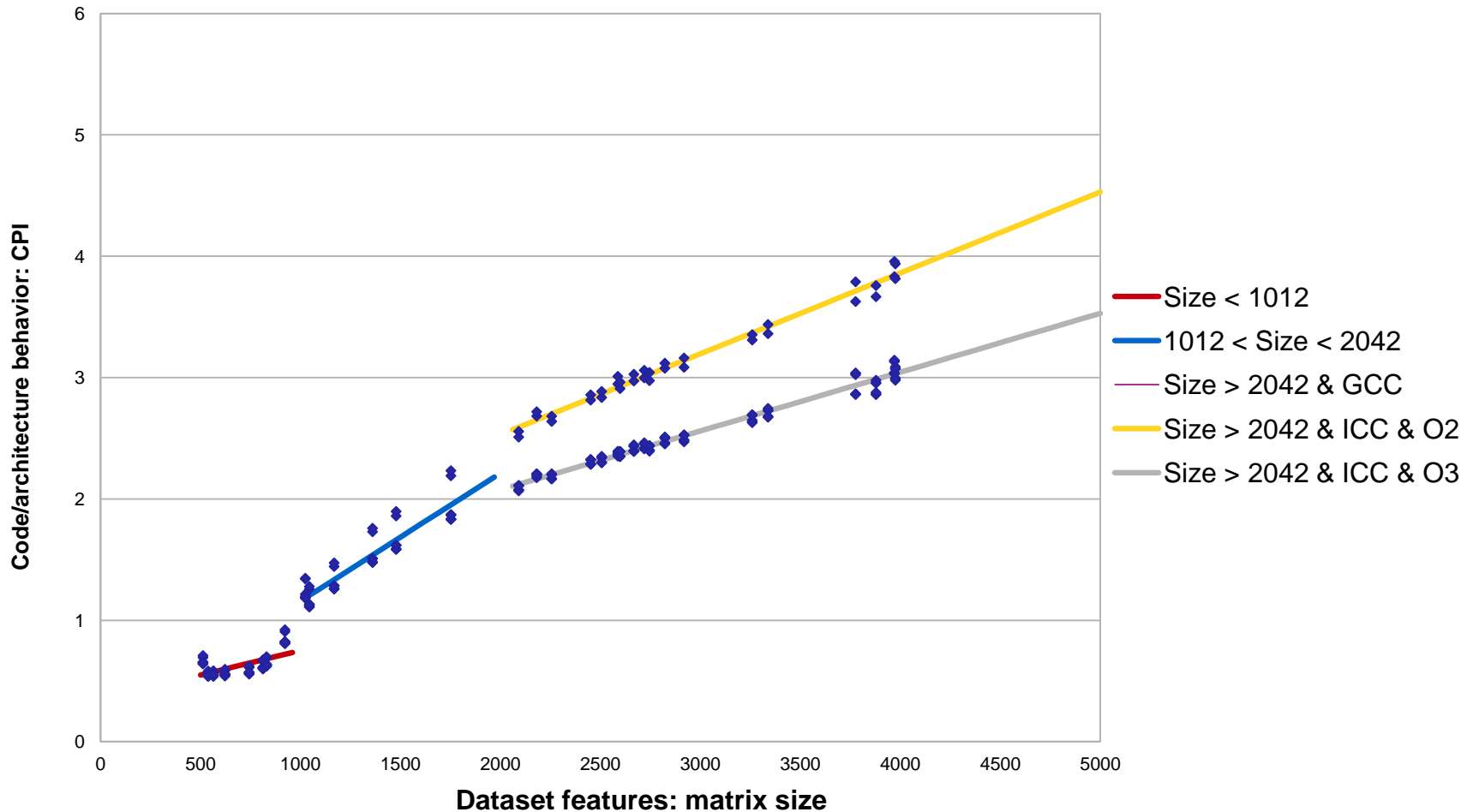


# Complexity reduction

Optimize decision tree (many different algorithms)

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

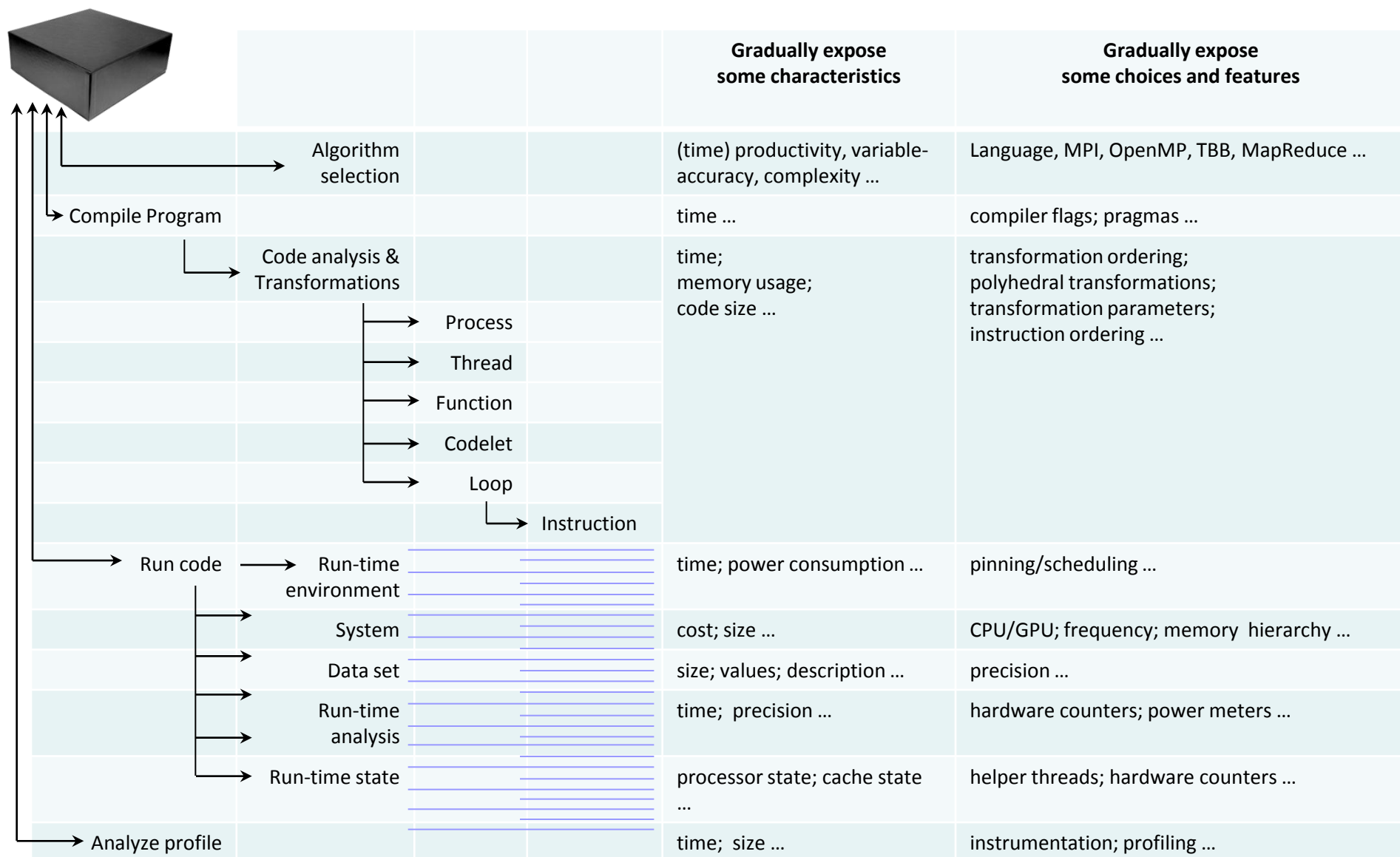
Compact data on-line before sharing with other users!



# Many new research and development opportunities

- Researchers can quickly replay, reproduce and validate existing results, and focus their effort on either feature learning and predictive models or on novel approaches combined with auto-tuning and machine learning
- 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 multi-objective program optimizations and hardware configuration for a given user scenario (requirements)*
  - *suggest run-time adaptation scenarios (co-design and co-optimization)*
  - *eventually enable self-tuning computer systems*

# Gradually and collaboratively increase granularity and complexity




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



# Current status

- Infrastructure is available under standard BSD license at <http://cTuning.org/tools/cm>
- Pilot repository is available at <http://c-mind.org/repo>  
(hundreds of kernels, thousands of datasets, tools, models, etc)
- Collective Mind concept **requires community effort** at all levels (sharing benchmarks and data sets, providing wrappers, finding features, improving models) - currently building community around this concept and infrastructure with a focus on:

Education	Academic research	Validation in industry
<div><p>Reproducible and collaborative research; new publication model where results are validated by the community.</p><ul style="list-style-type: none"><li>• Panel at ADAPT 2014 @ HiPEAC 2014 <a href="http://adapt-workshop.org">http://adapt-workshop.org</a></li><li>• REPRODUCE 2014 @ HPCA 2014 <a href="http://www.occamportal.org/reproduce">www.occamportal.org/reproduce</a></li><li>• Special journal issue on reproducible research in ACM TET</li></ul></div>	<ul style="list-style-type: none"><li>• Systematizing, validating, sharing past research knowledge and practical experience during auto-tuning and ML</li><li>• Optimal feature and model selection</li><li>• Compacting and systematizing benchmarks and data sets</li><li>• Run-time adaptation and ML</li></ul>	<ul style="list-style-type: none"><li>• Most of techniques have been validated in industry with IBM, ARM, Intel, ARC (Synopsys), CAPS, STMicroelectronics</li><li>• Continue extrapolating collected knowledge to build faster and more power efficient computer systems to continue innovation in science and technology!</li></ul>

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

- My 2 PhD students:

Abdul Memon and Yuriy Kashnikov

- Colleagues from STMicroelectronics (France):

Christophe Guillone, Antoine Moynault, Christian Bertin

- Colleagues from ARM (UK): Anton Lokhmotov

- Colleagues from NCAR (USA): *Davide Del Vento and his interns*

- Colleagues from CAPS Entreprise (France): *Francois Bodin*

- Colleagues from Intel (USA): *David Kuck and David Wong*

- cTuning community:

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



- EU FP6, FP7 program and HiPEAC network of excellence

<http://www.hipeac.net>

c-mind.org

# *Collective Mind Repository and Infrastructure*

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



**Thank you for your attention!**

**Contact: [Grigori.Fursin@cTuning.org](mailto:Grigori.Fursin@cTuning.org)**

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

*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*