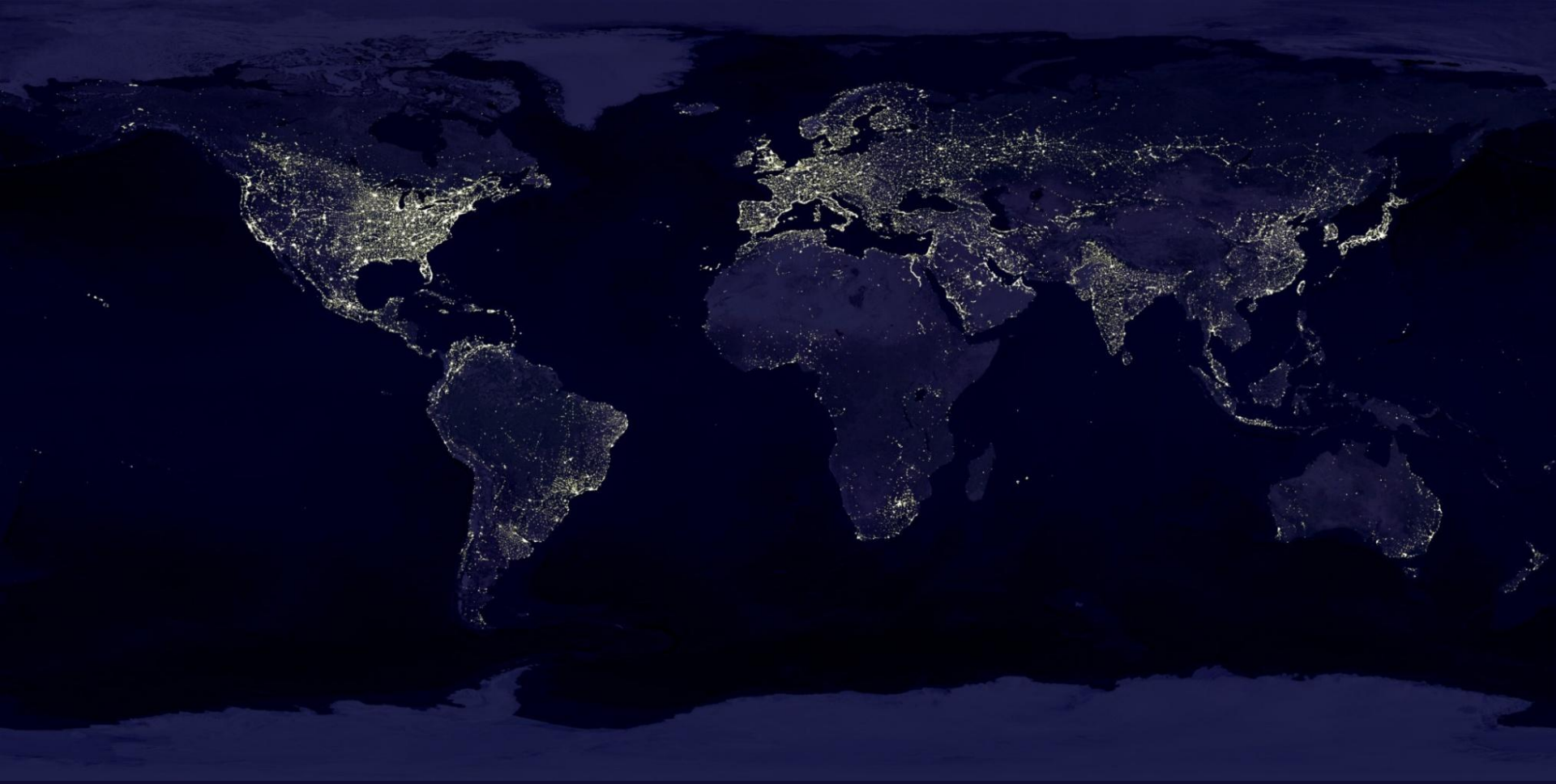


# Systematizing tuning of computer systems using crowdsourcing and statistics



**Grigori Fursin**

**INRIA, France**

**April 2013**

# Messages

## 1<sup>st</sup> talk (Wednesday)

Systematizing tuning of computer systems  
using crowdsourcing and statistics

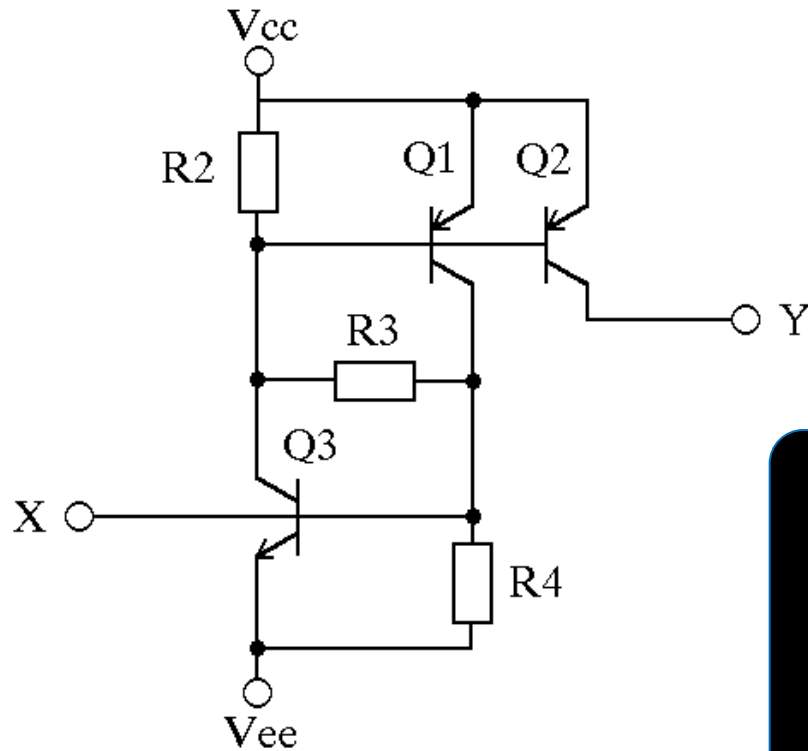
- *Revisiting current design and optimization methodology*
- *Leveraging experience and computer resources of multiple users*
- *Using predictive modelling and data mining to improve computer systems*

## 2<sup>nd</sup> talk (Friday)

Collective Mind: novel methodology, framework  
and repository to crowdsource auto-tuning

- *New plugin-based extensible infrastructure and repository for collaborative analysis and tuning of computer systems - will be released in May 2013*
- *“Big data” enables cooperation between architecture, compiler, OS and application designers and mathematicians*
- *Examples of auto-tuning and predictive modeling for numerical kernels*

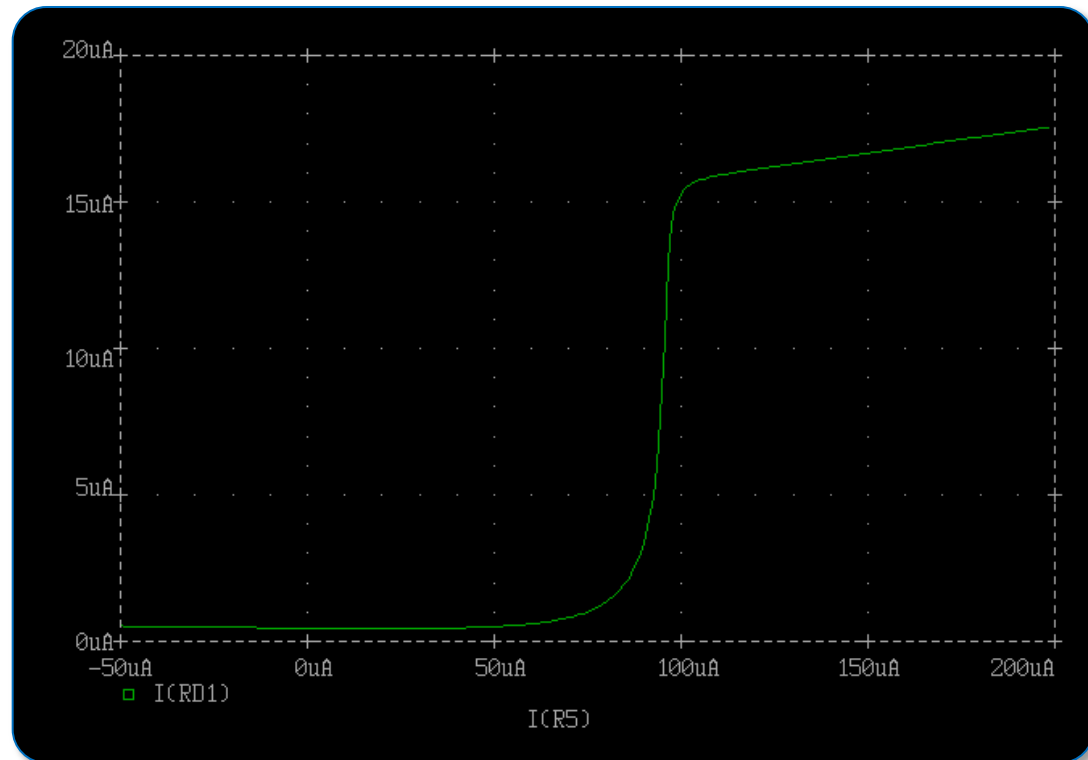
# Motivation: back to 1993



*Semiconductor neural element* -  
possible base of neural computers

Modeling and understanding  
brain functions

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

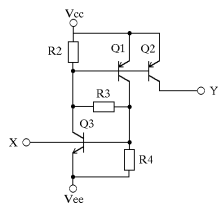


# Motivation: back to basics

**End users**



**Task**



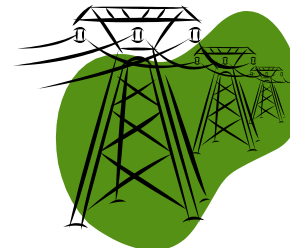
**Solution**

**User requirements:**

*most common:*

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

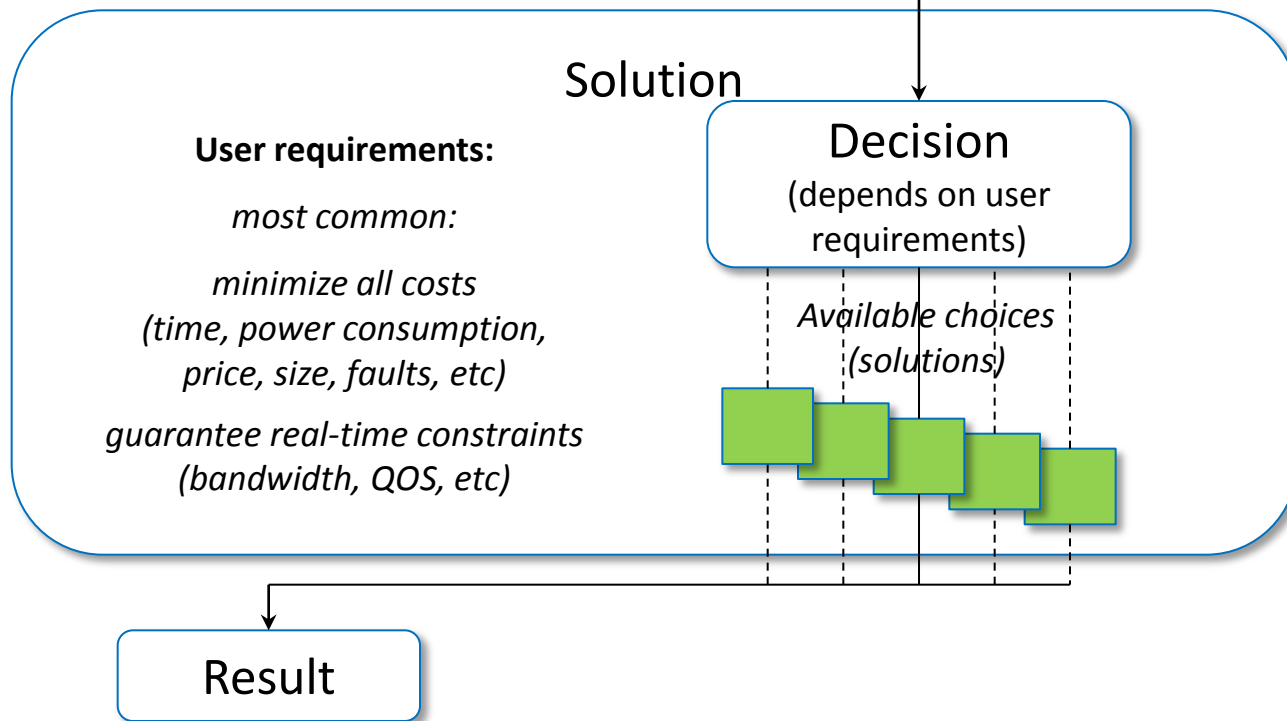
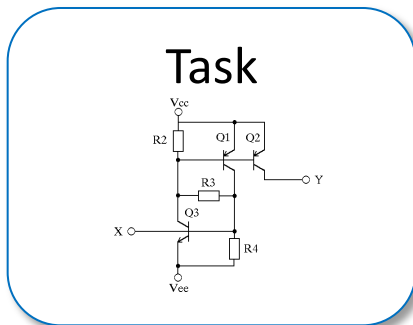
*guarantee real-time constraints  
(bandwidth, QOS, etc)*



**Result**

# Motivation: back to basics

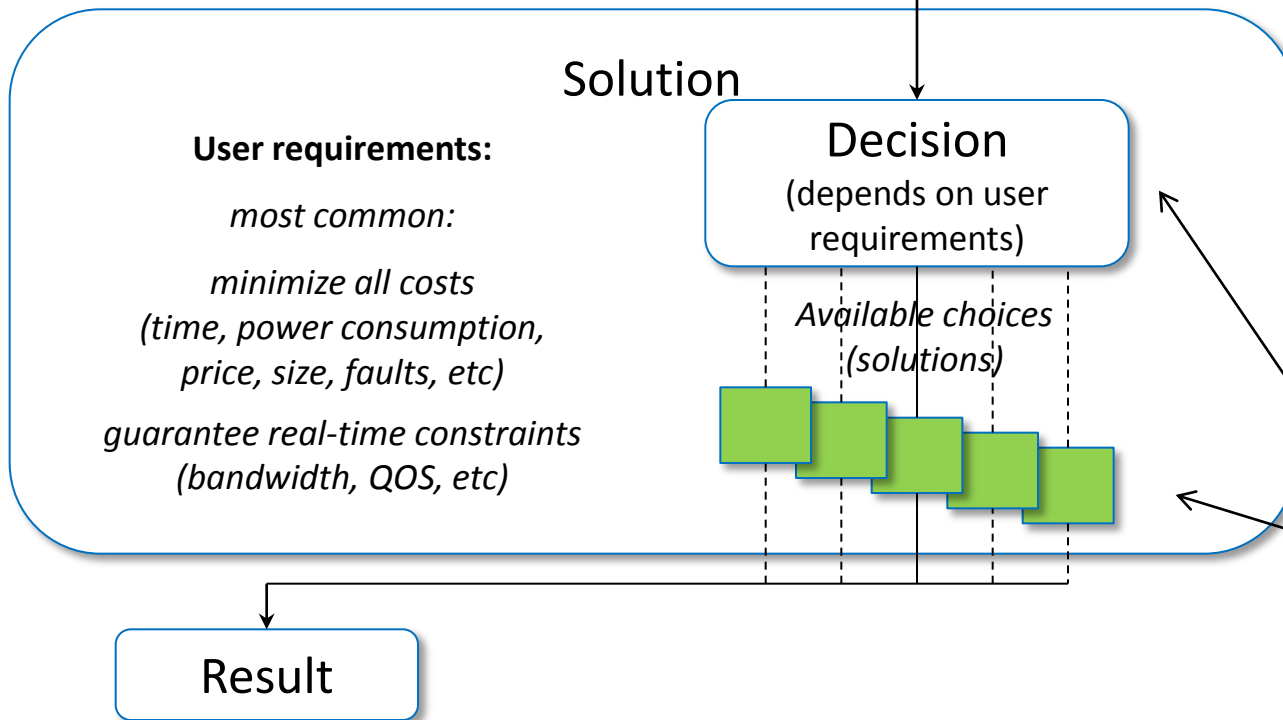
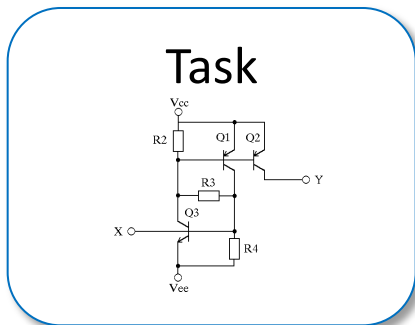
End users



Result

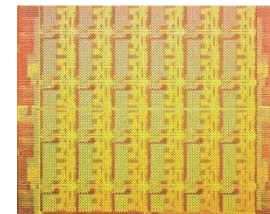
# Motivation: back to basics

End users



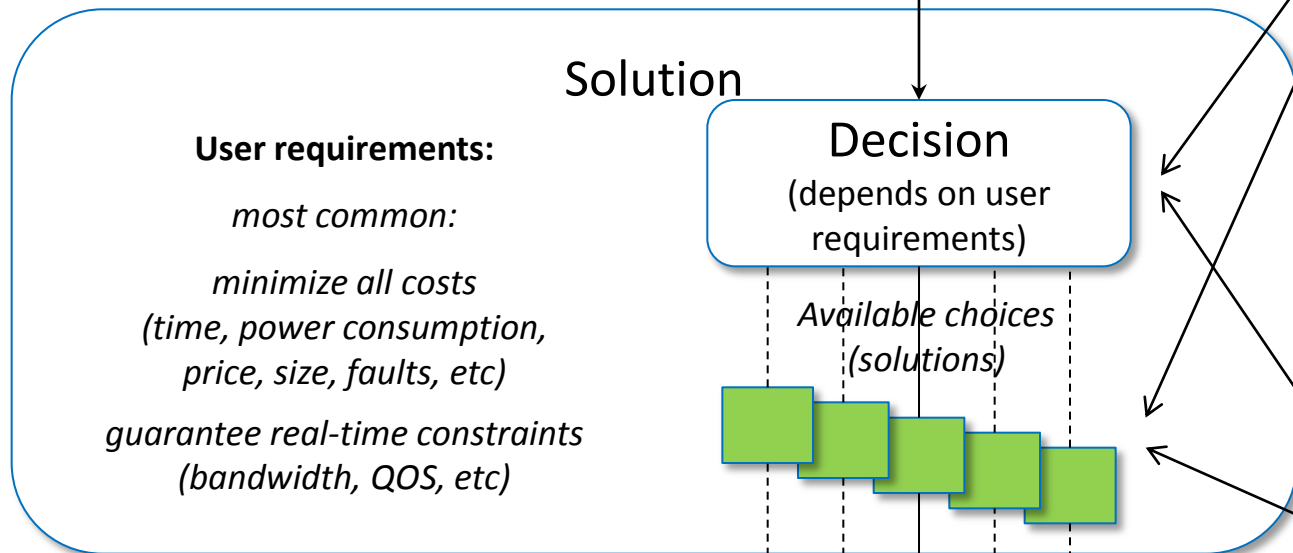
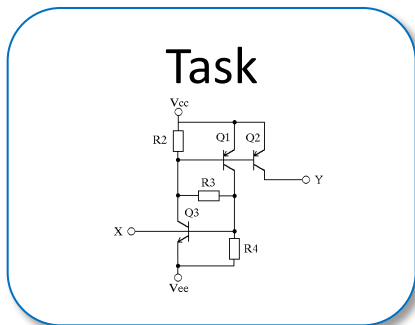
*Should provide choices and help with decisions*

**Hardware and software designers**



# Motivation: back to basics

End users



**Result**

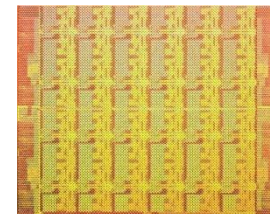


**Service/application providers**

(supercomputing, cloud computing, mobile systems)

*Should provide choices and help with decisions*

**Hardware and software designers**



# Available solutions: hardware

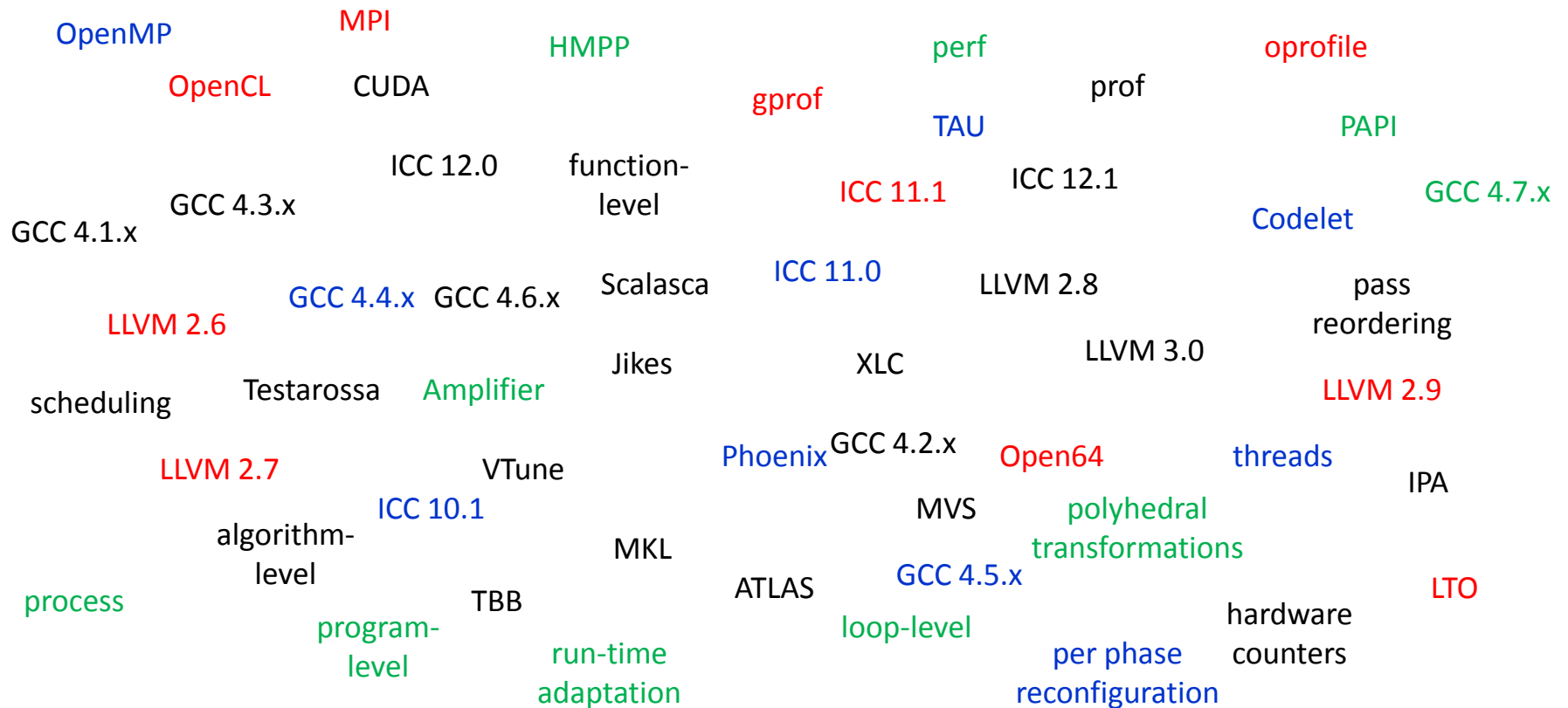
Companies compete hard to deliver many solutions with various characteristics: *performance, power consumption, size, bandwidth, response time, reliability, cost ...*



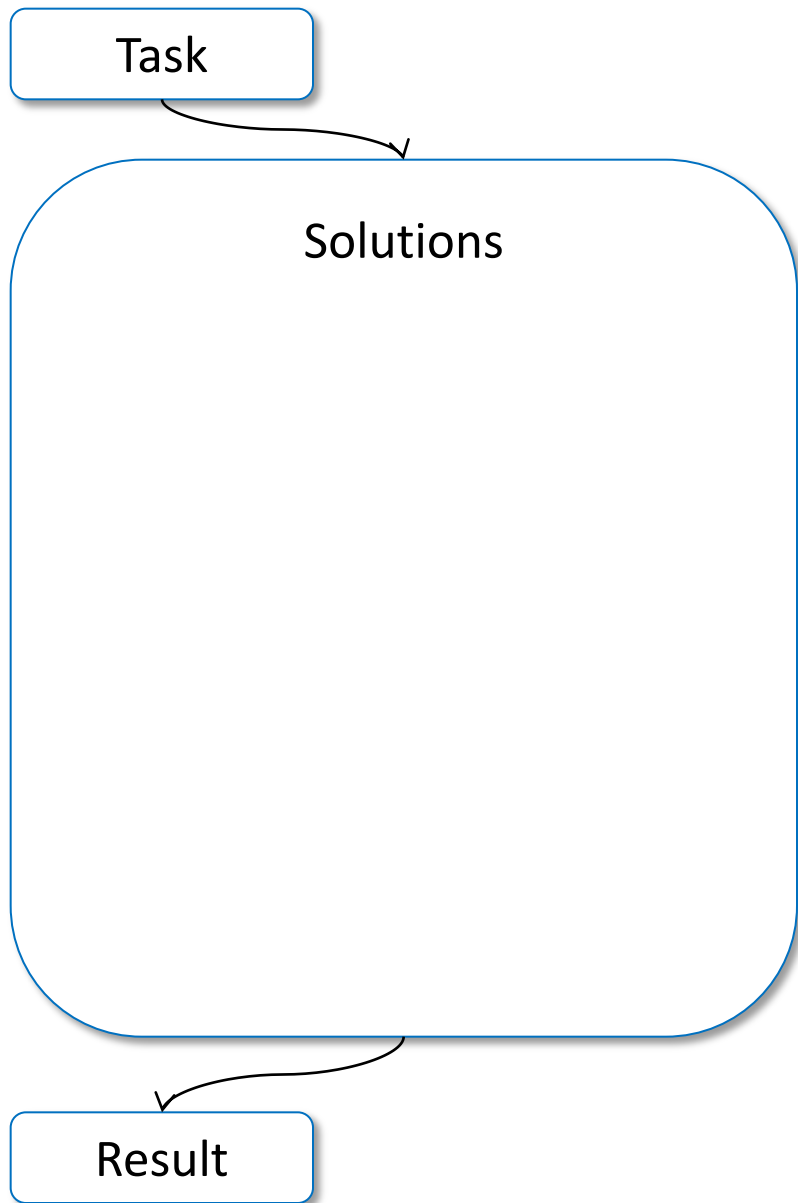


# Available solutions: software

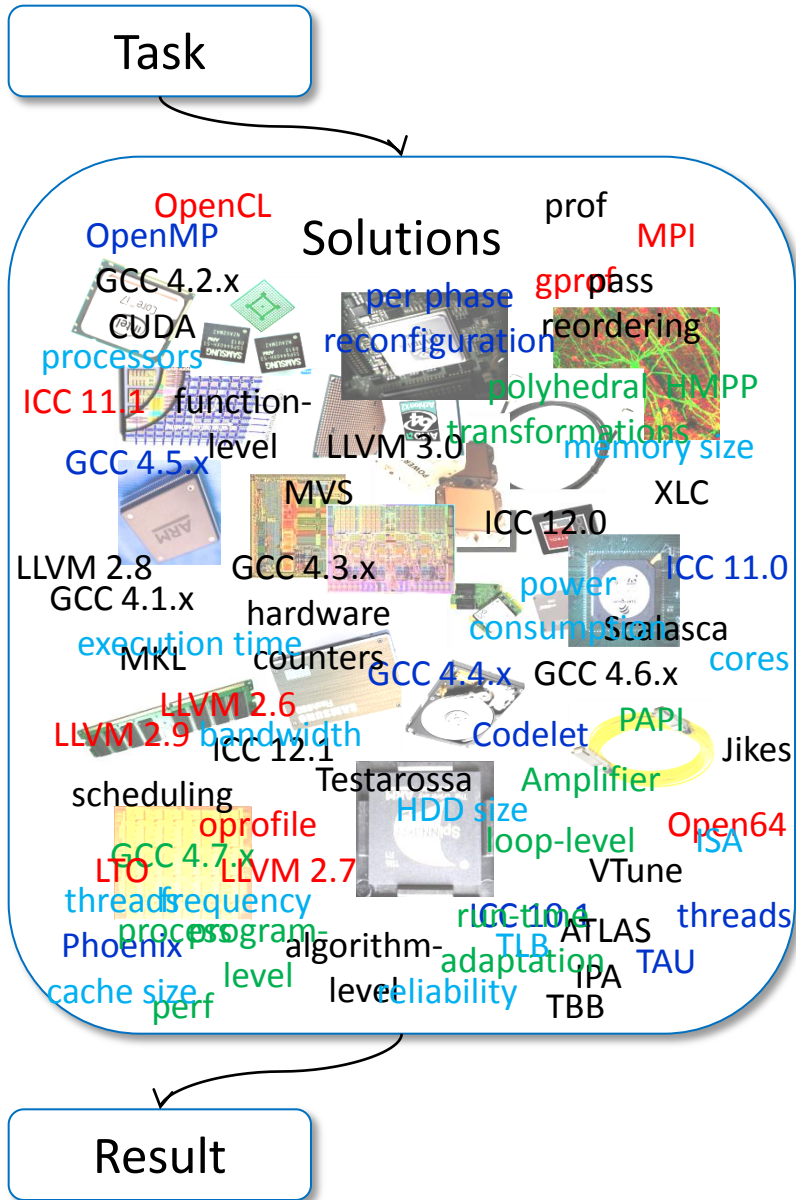
Software developers try to keep pace and produce various algorithms, programming models, languages, analysis tools, compilers, run-time systems, databases, etc.



# Challenges



# Challenges

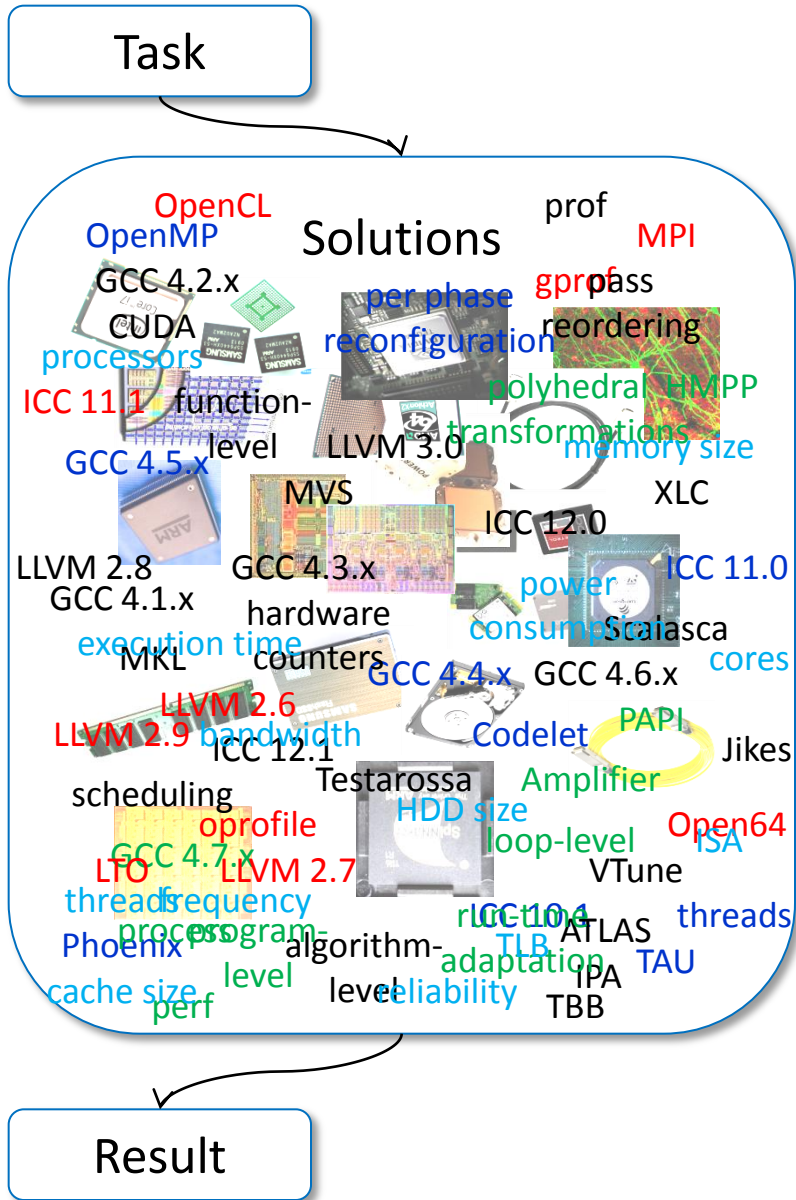


- 1) Rising complexity of computer systems:  
too many design and optimization choices
- 2) Performance is not anymore the only requirement:  
multiple user objectives vs choices  
benefit vs optimization time
- 3) Complex relationship and interactions  
between ALL software and hardware  
components.
- 4) Too many tools with non-unified interfaces  
changing from version to version:  
technological chaos

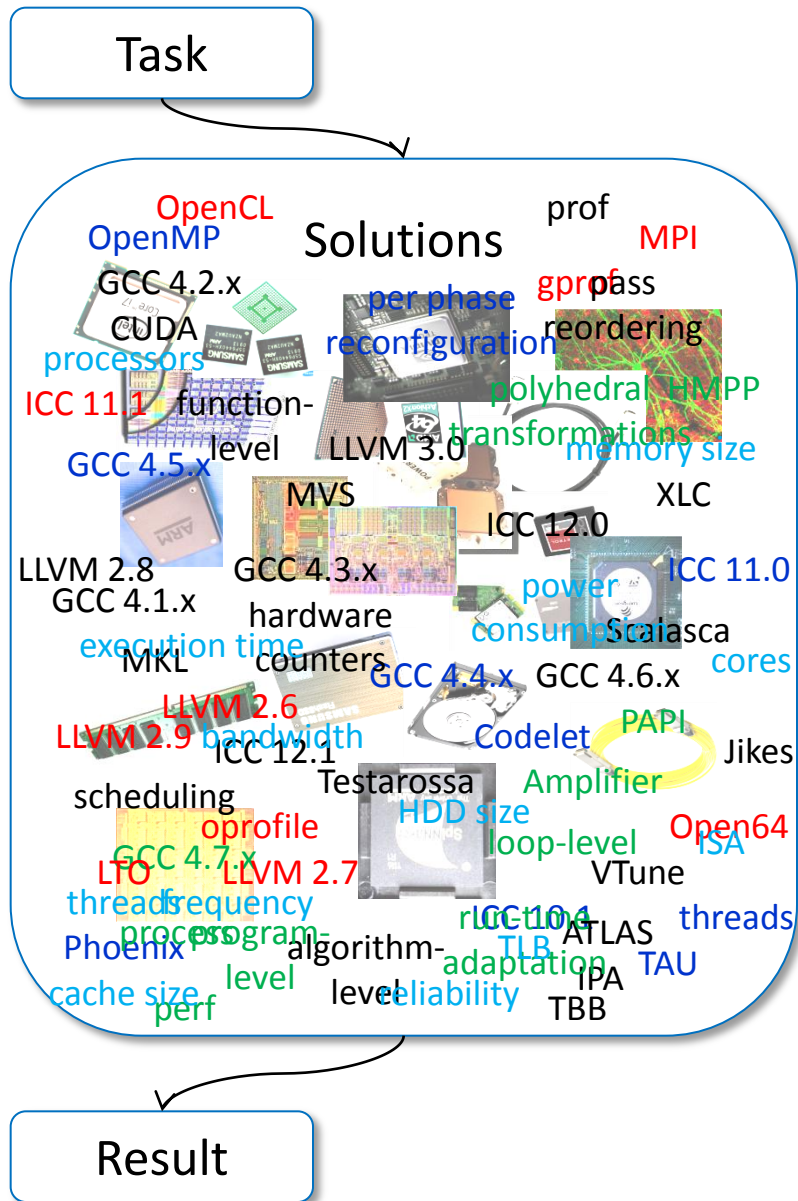
# Challenges

Result:

- finding the right solution for end-user 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

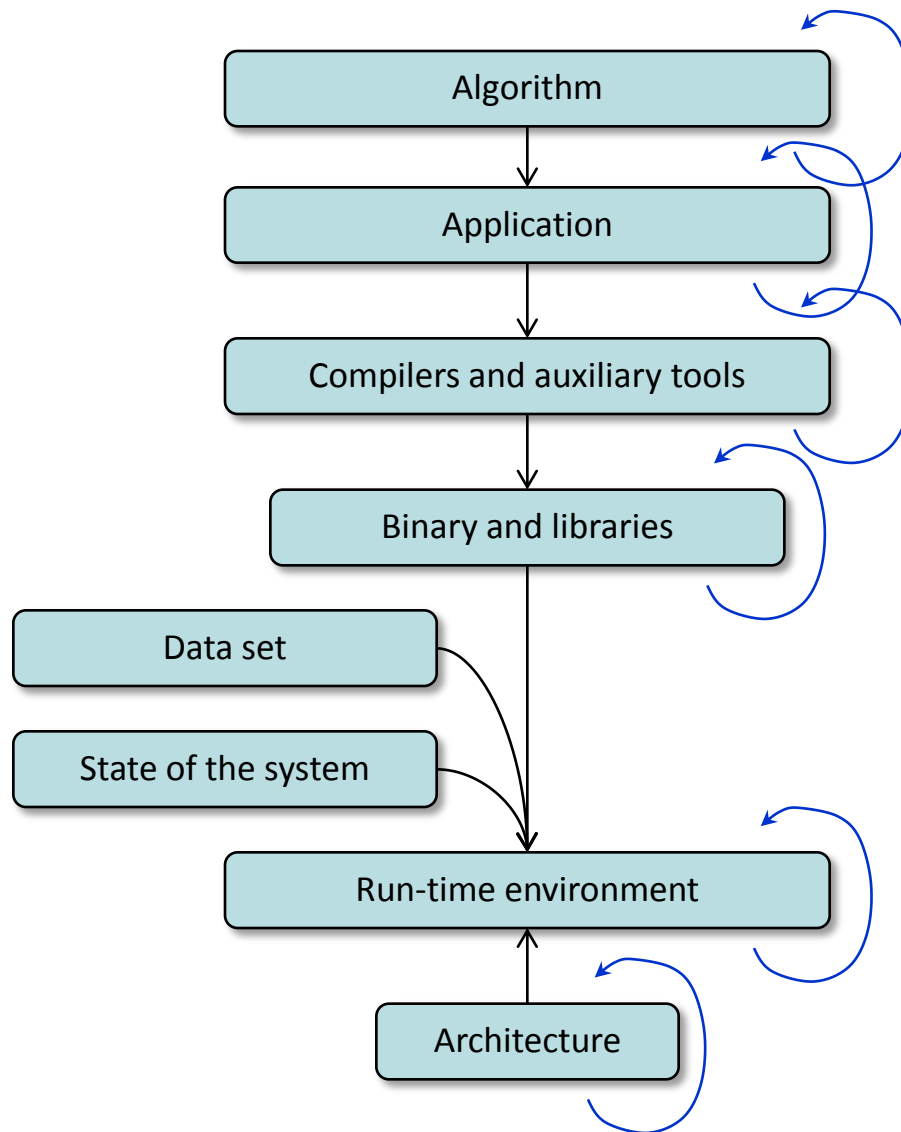


## Result:

- finding the right solution for end-user 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**

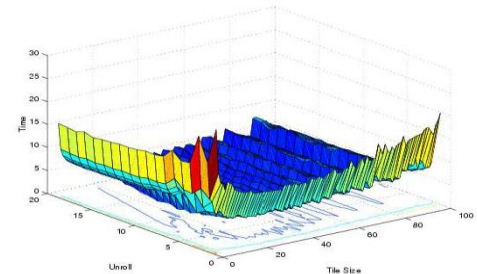
*Understanding and modeling of the overall relationship between end-user algorithms, applications, compiler optimizations, hardware designs, data sets and run-time behavior became simply infeasible!*

# Attempts to solve these problems: auto-tuning



## 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 ...

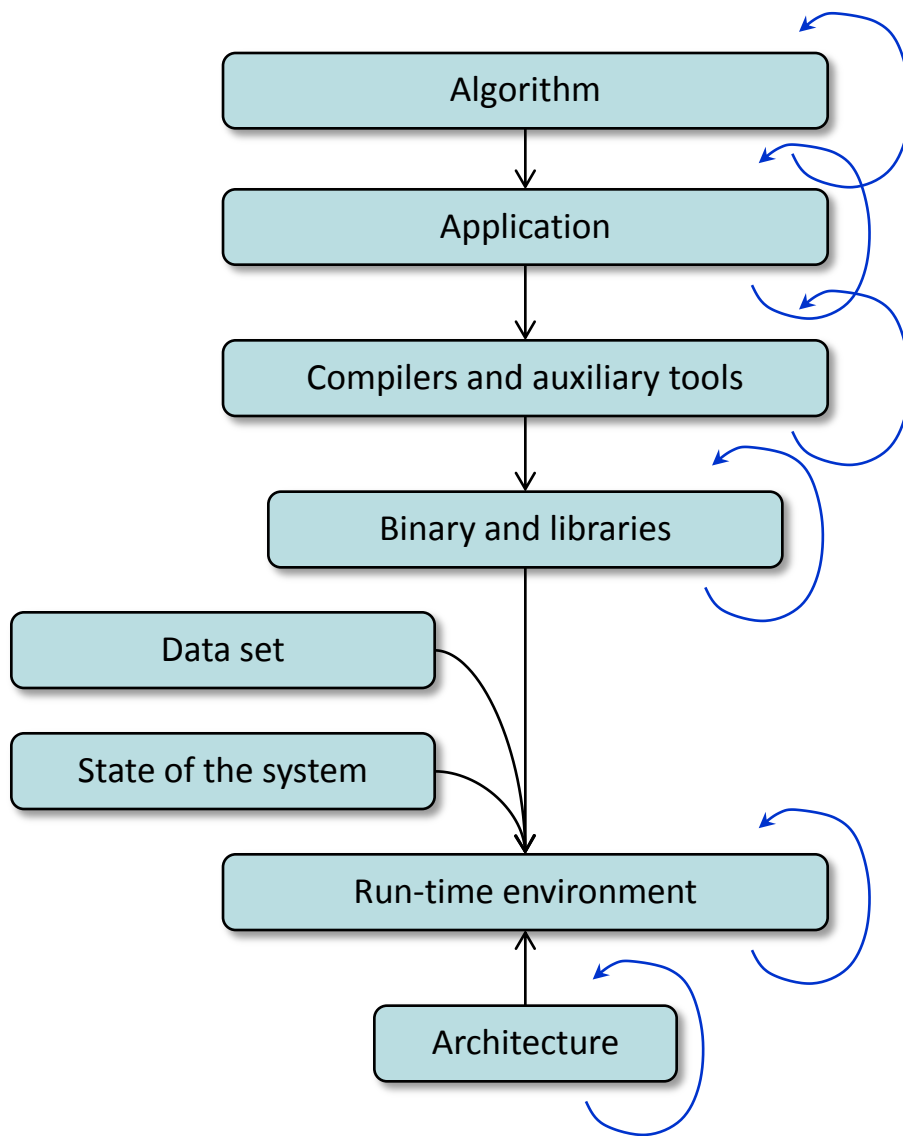
# Attempts to solve these problems: auto-tuning

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

## Why?

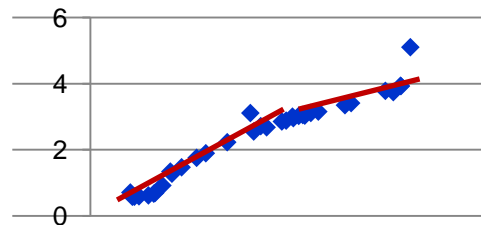
- Optimization spaces are large and non-linear with many local minima
- Exploration is slow and ad-hoc (random, genetic, some heuristics)
- Only a few benchmarks are considered
- Often the same (one) dataset is used
- Only part of the system is taken into account (rarely reflect behavior of the whole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

# Attempts to solve these problems: machine learning



**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 ...



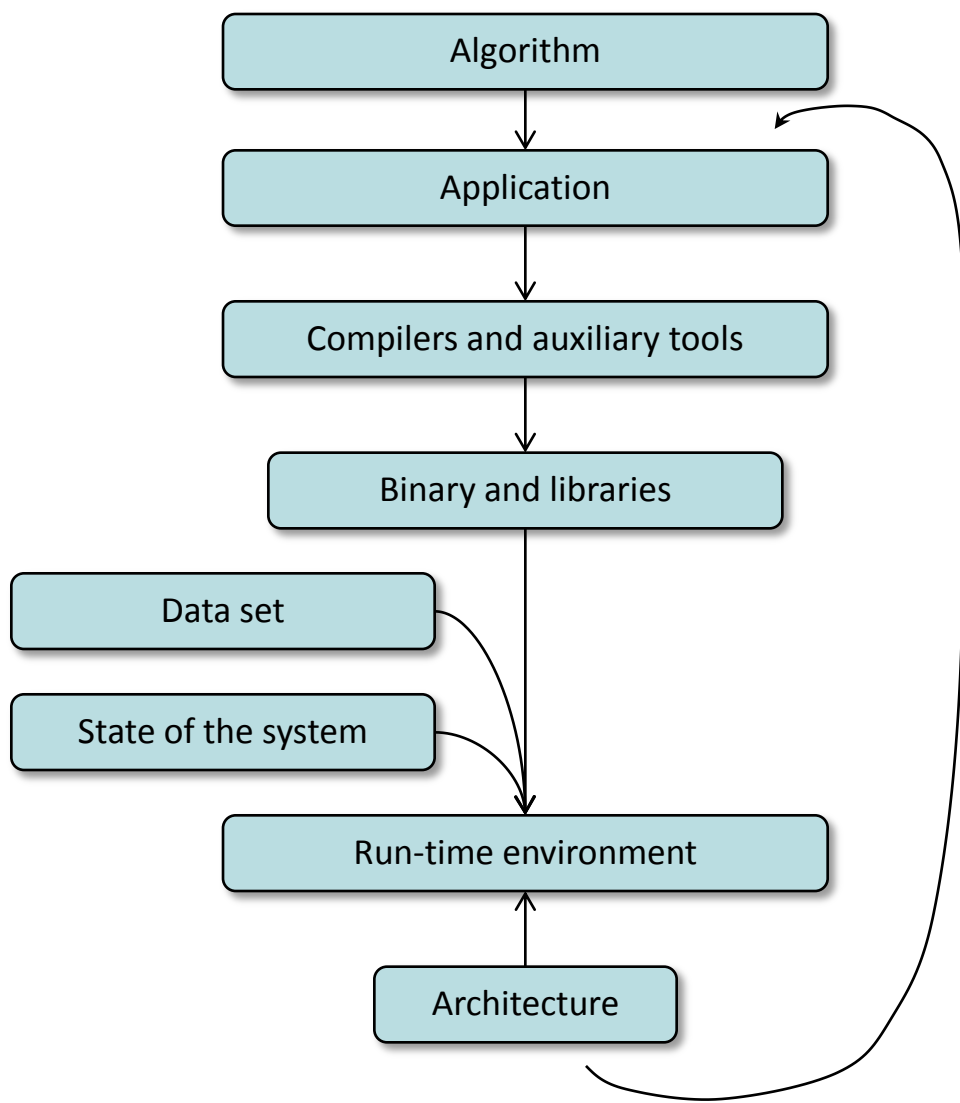
# Attempts to solve these problems: machine learning

**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



**Co-design:**  
Explore choices and behavior of the whole system.

Showed high potential in the last years but ...

# Attempts to solve these problems: co-design

**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
- Often impossible to expose tuning choices or obtain characteristics at all levels
- Limited training sets
- Still no knowledge sharing

# 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  
**- needed dramatically new approach!**

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

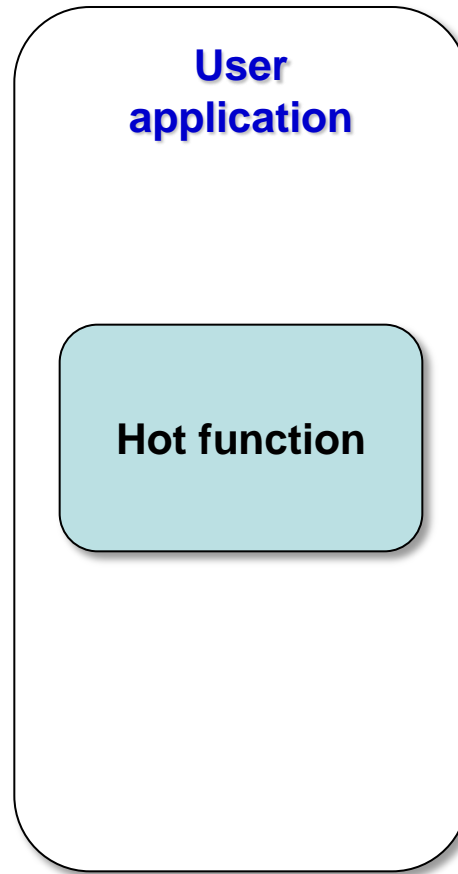


**Can we leverage their experience and computational resources?**

**Can we transparently distribute optimization  
and learning across many users?**

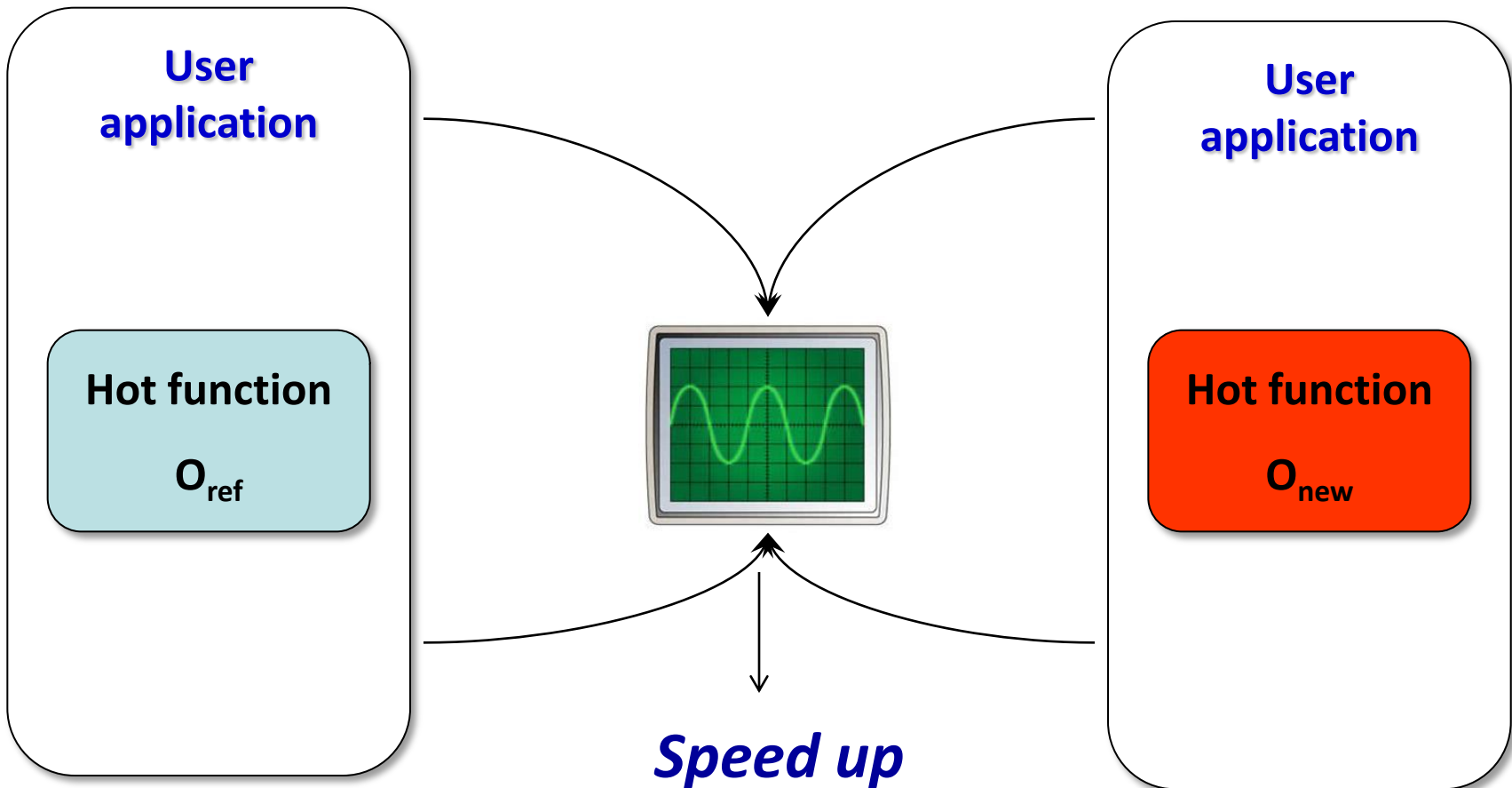
# Challenges

**How can we evaluate optimizations in a realistic environment without complex recompilation frameworks and without source code?**



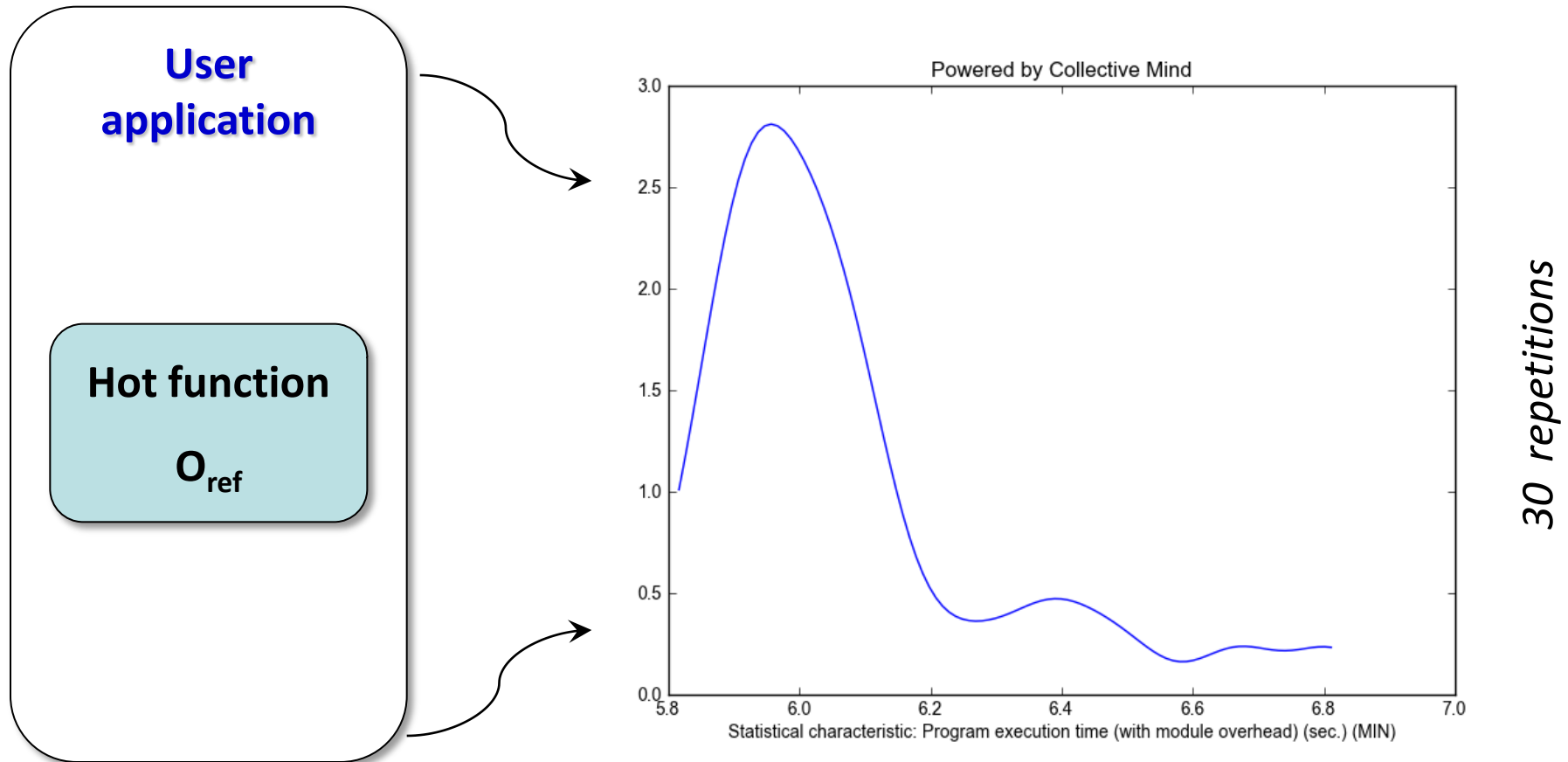
# Challenges

**First problem:  
need reference run with the same dataset**



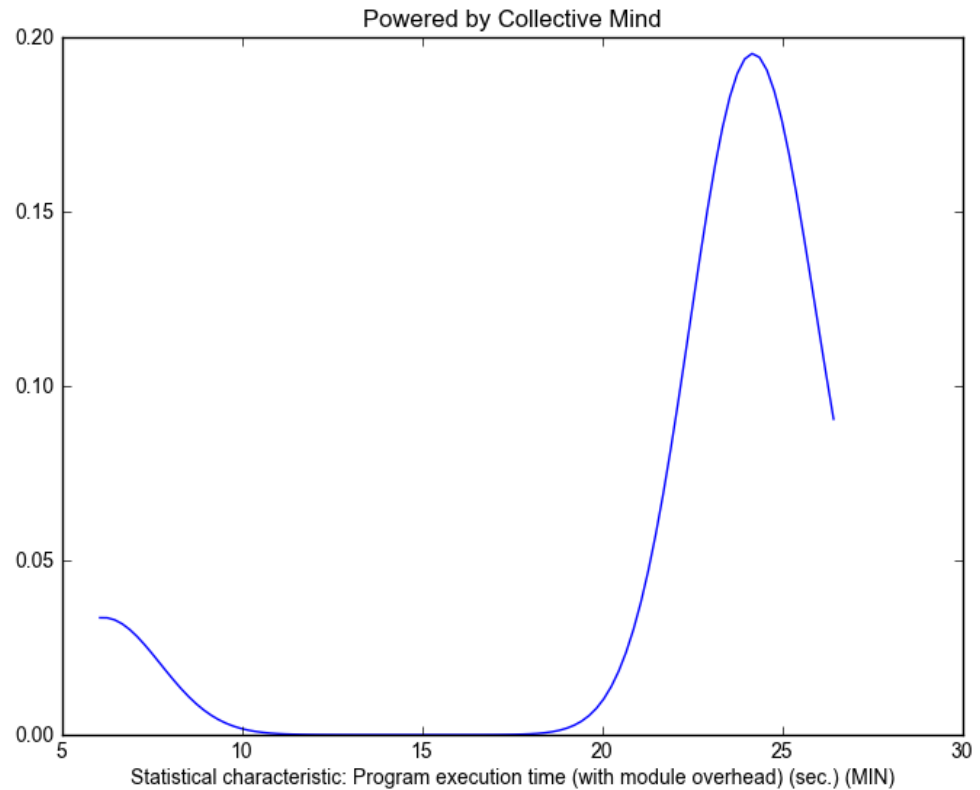
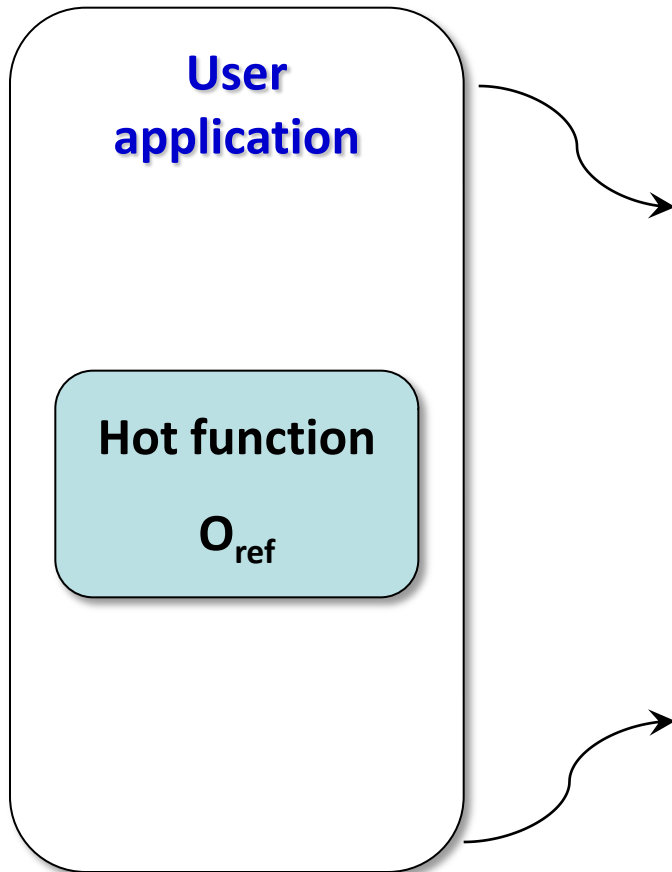
# Challenges

**Second problem: variation in execution time due to different run-time states**



# Challenges

**Second problem: variation in execution time due to different run-time states**



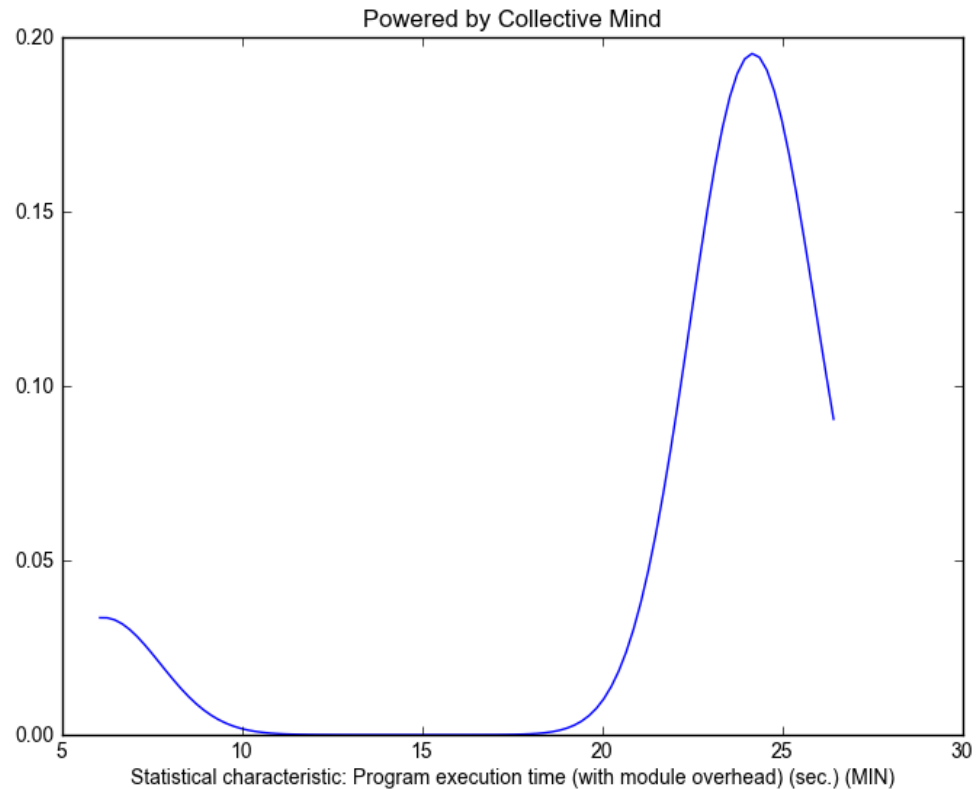
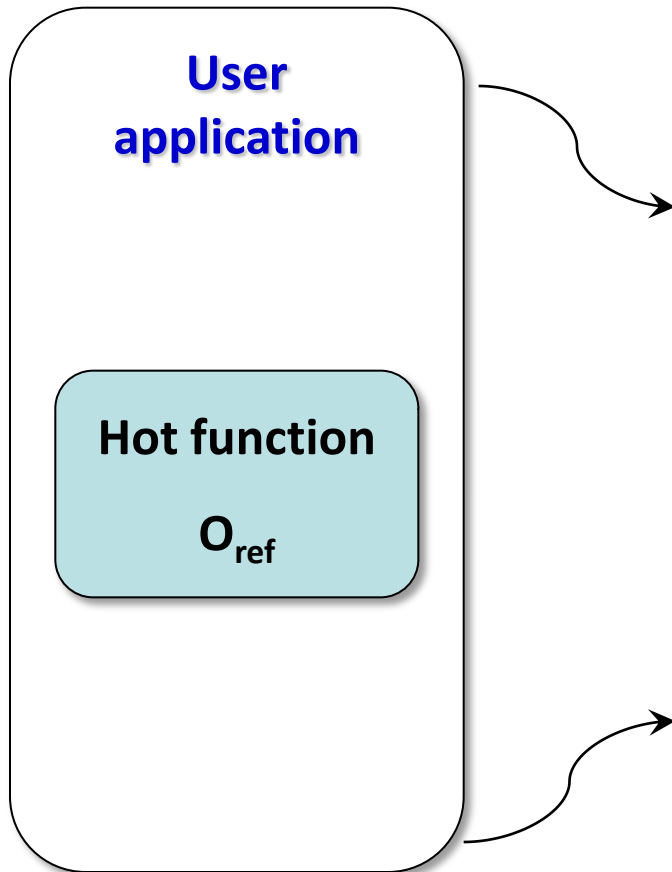
*30 repetitions*



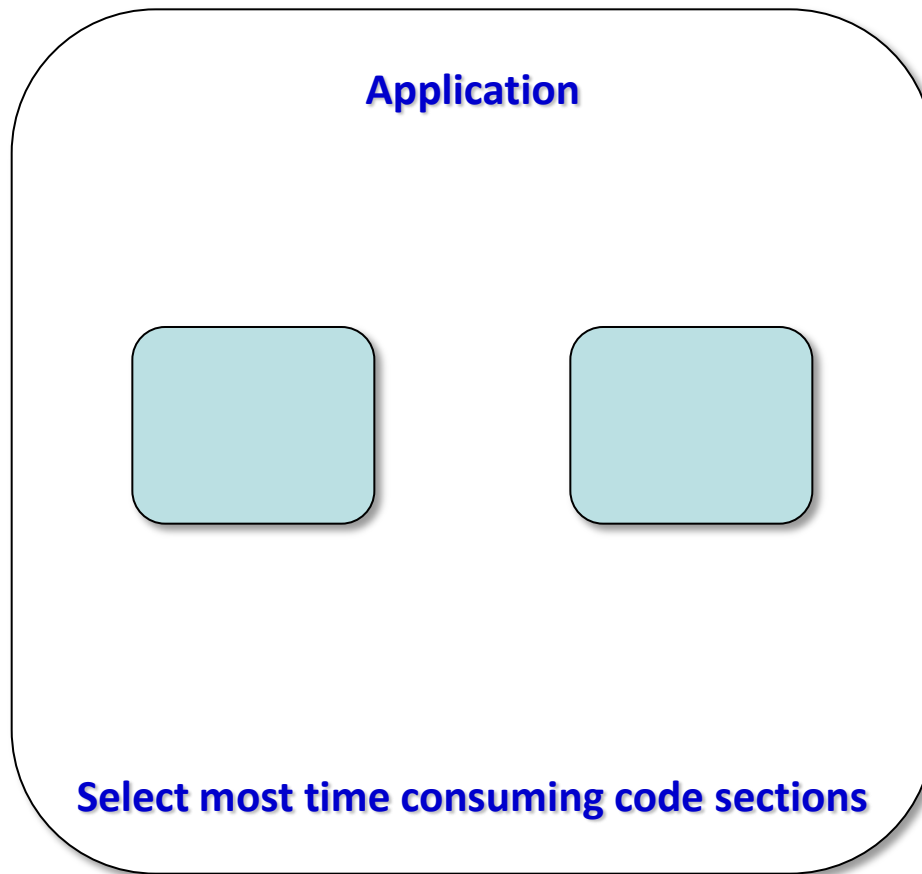
# Challenges

How can we evaluate some optimization in a realistic environment?

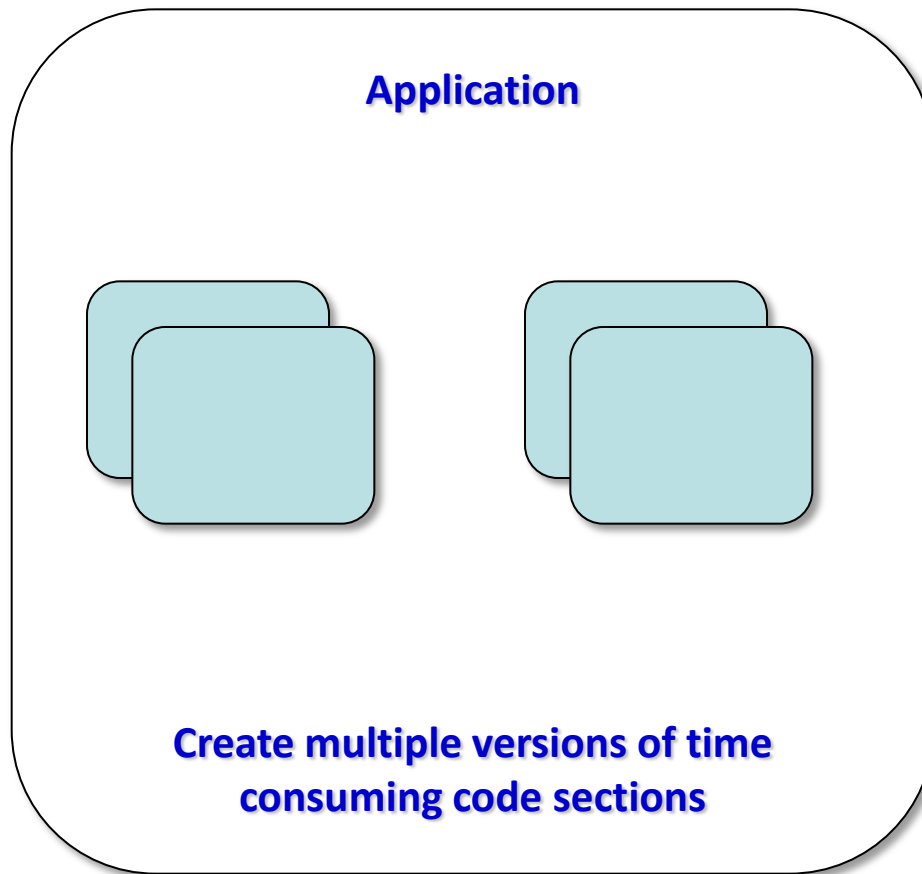
**Second problem: variation in execution time due to different run-time states (parallel processes, adaptive scheduling, pinning, cache state, bus state, frequency changes, etc)**



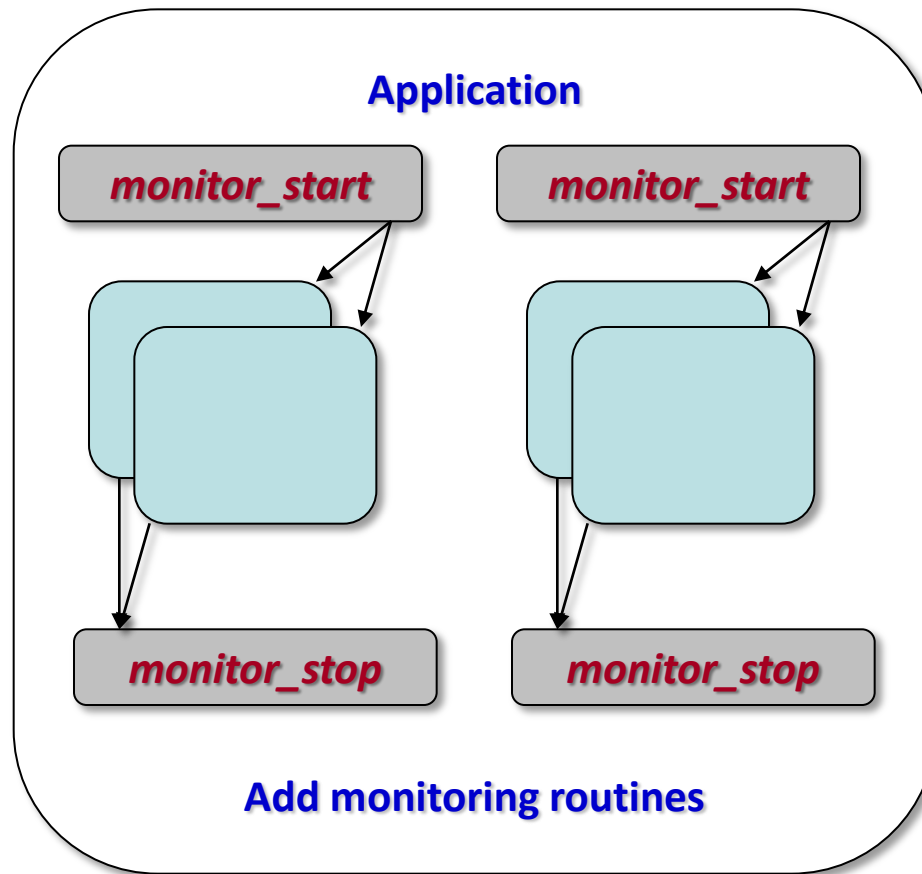
# Our approach: static multiversioning



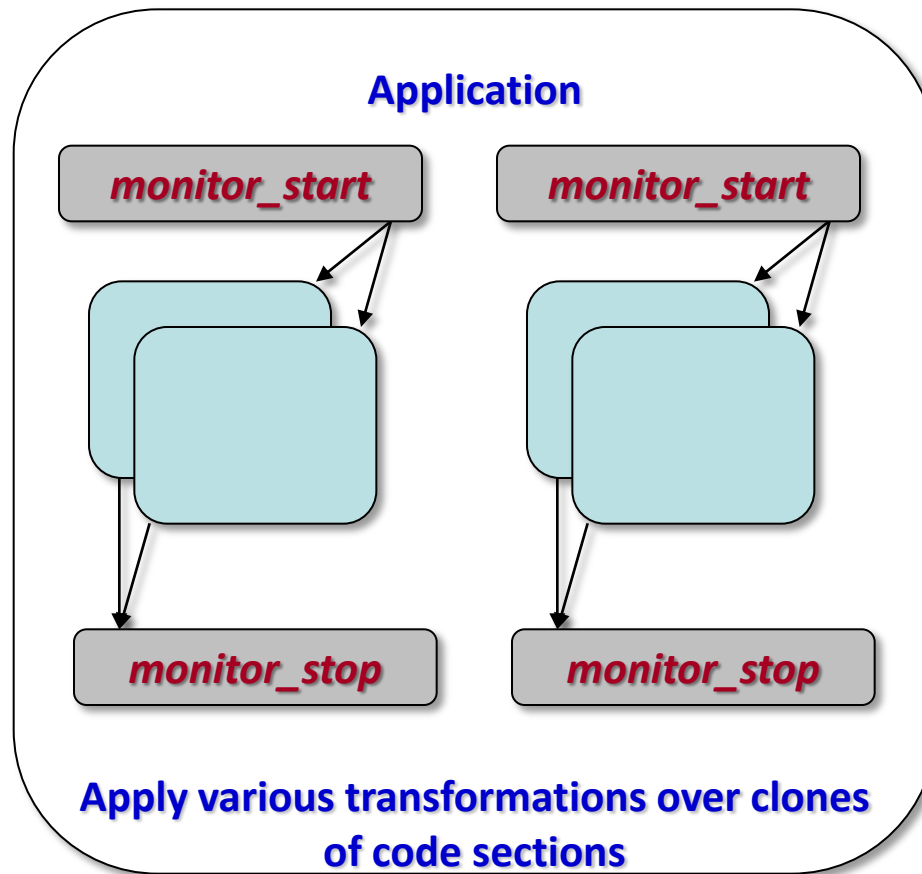
# Our approach: static multiversioning



# Our approach: static multiversioning

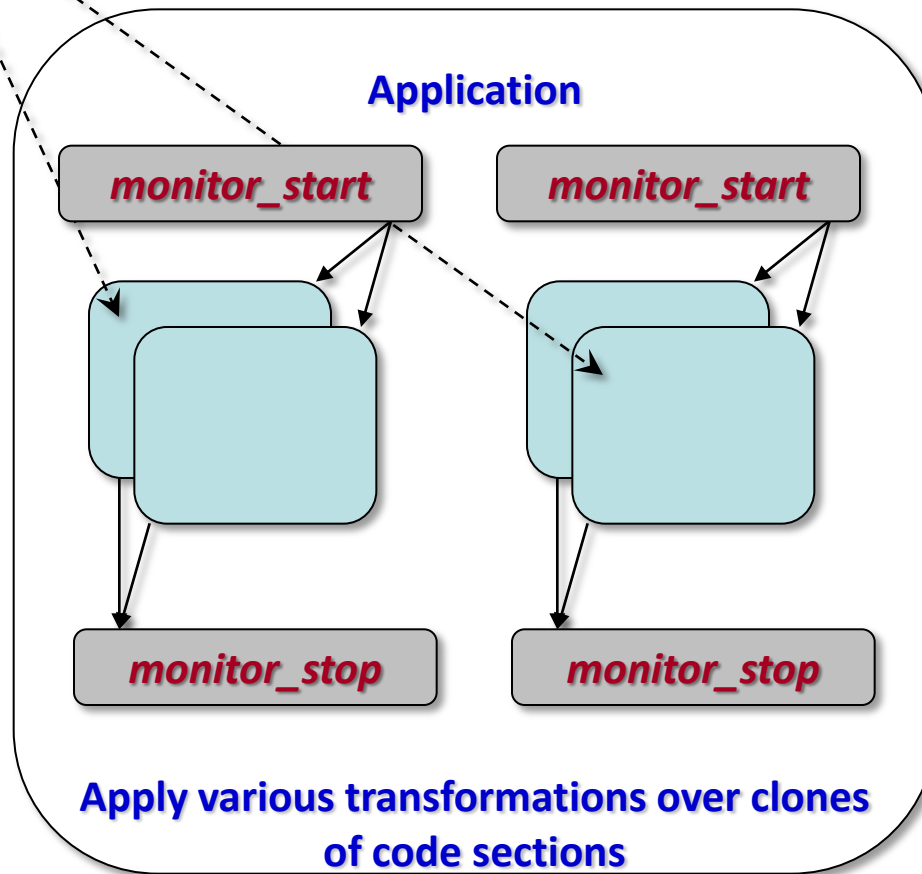


# Our approach: static multiversioning

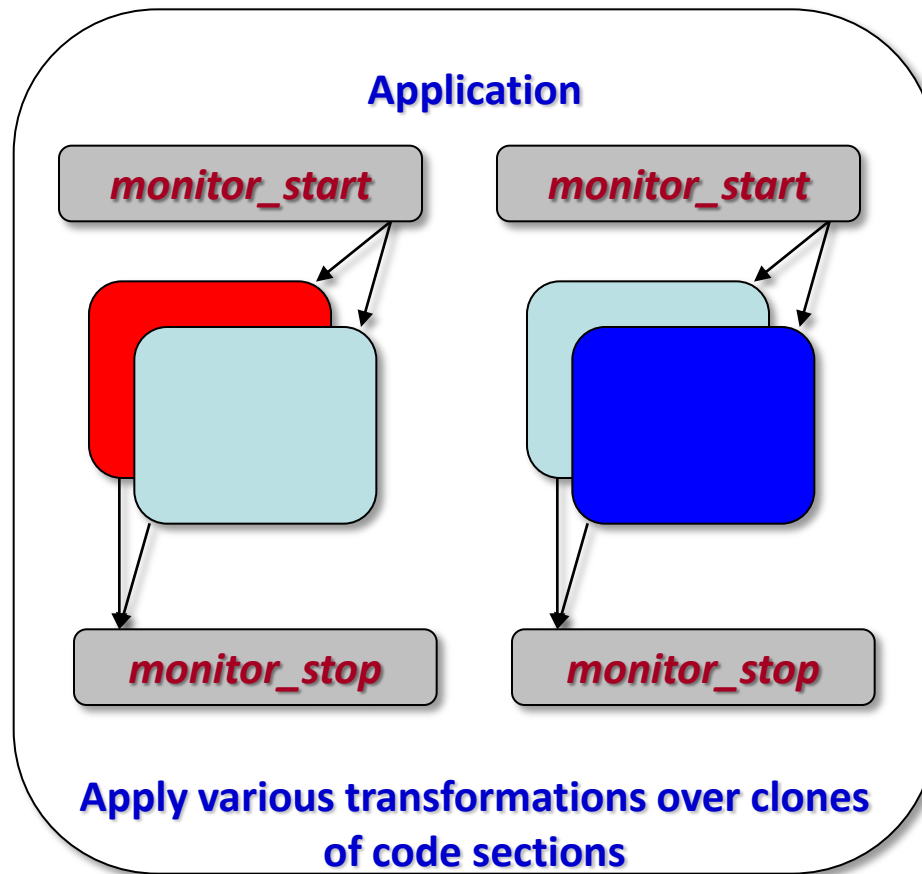


# Our approach: static multiversioning

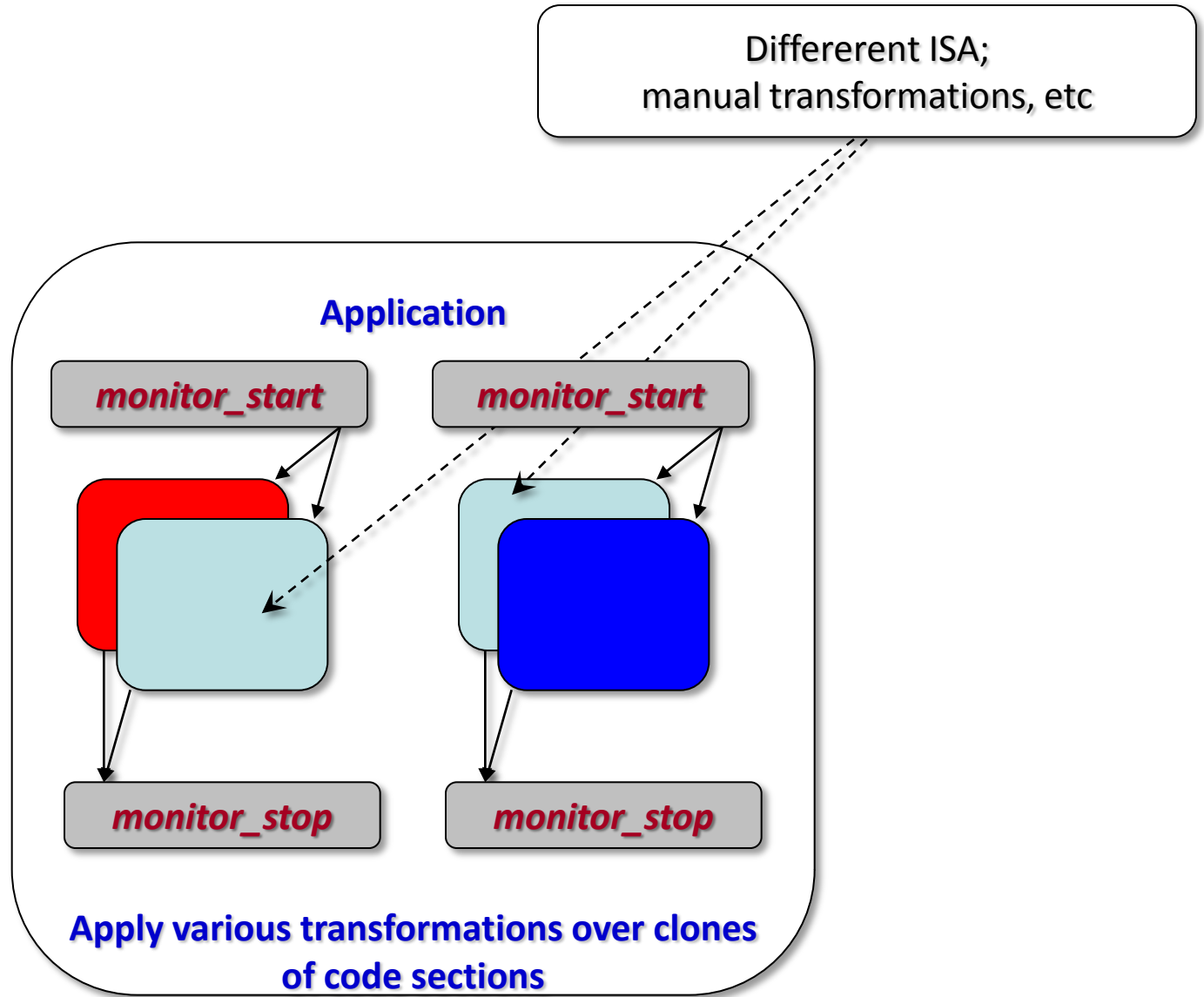
Select global or fine-grain internal compiler (or algorithm) optimizations



# Our approach: static multiversioning



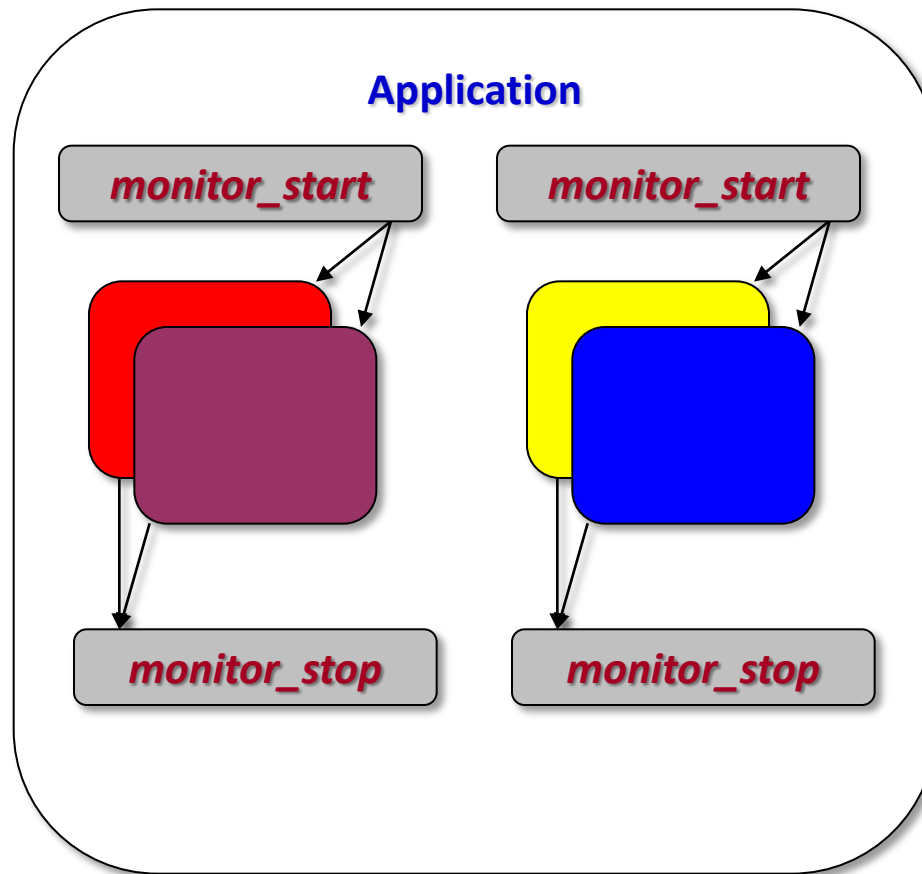
# Our approach: static multiversioning





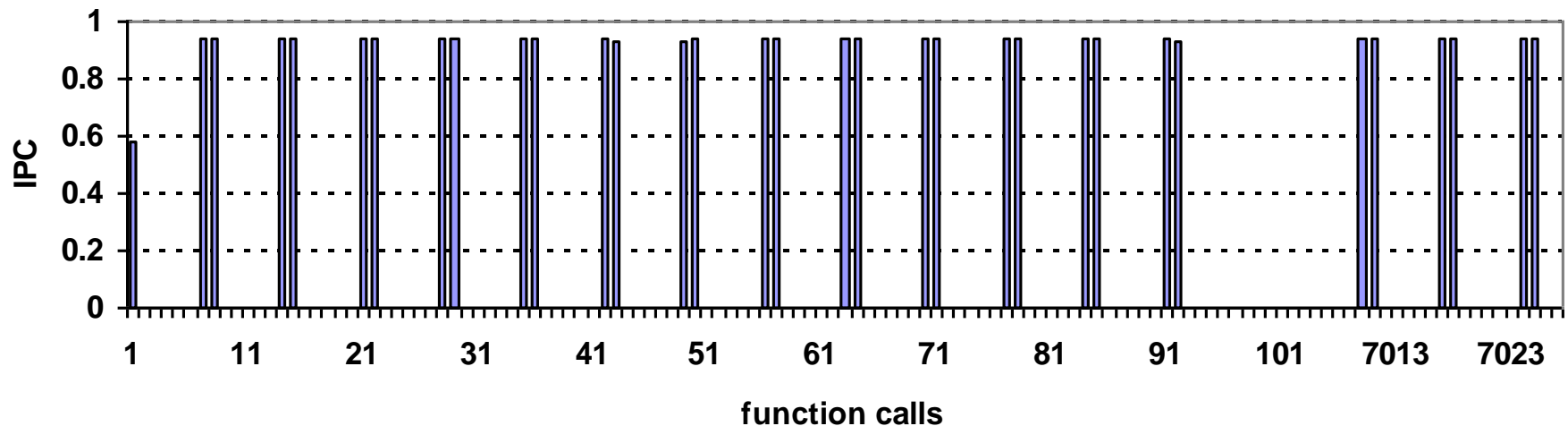
# Our approach: static multiversioning

*Final instrumented program*



# Observations: program execution phases

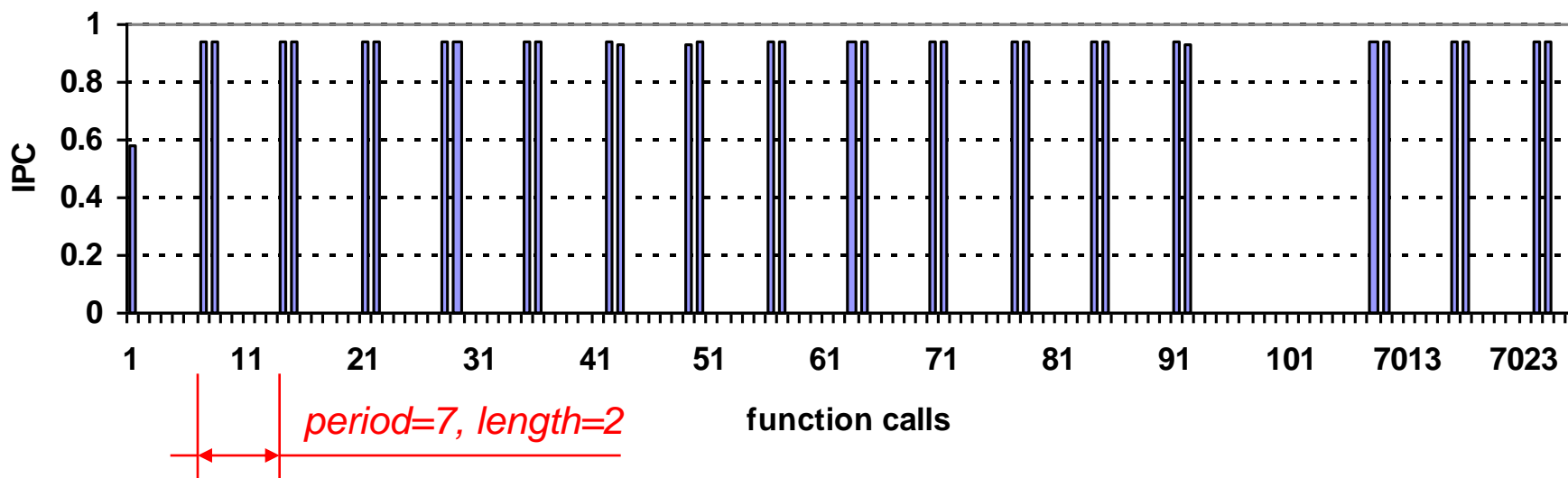
IPC for subroutine resid of benchmark mgrid across calls



- *Define stability by 3 consecutive or periodic executions with the same IPC*
- *Predict further occurrences with the same IPC (using period and length of regions with stable performance)*

# Observations: program execution phases

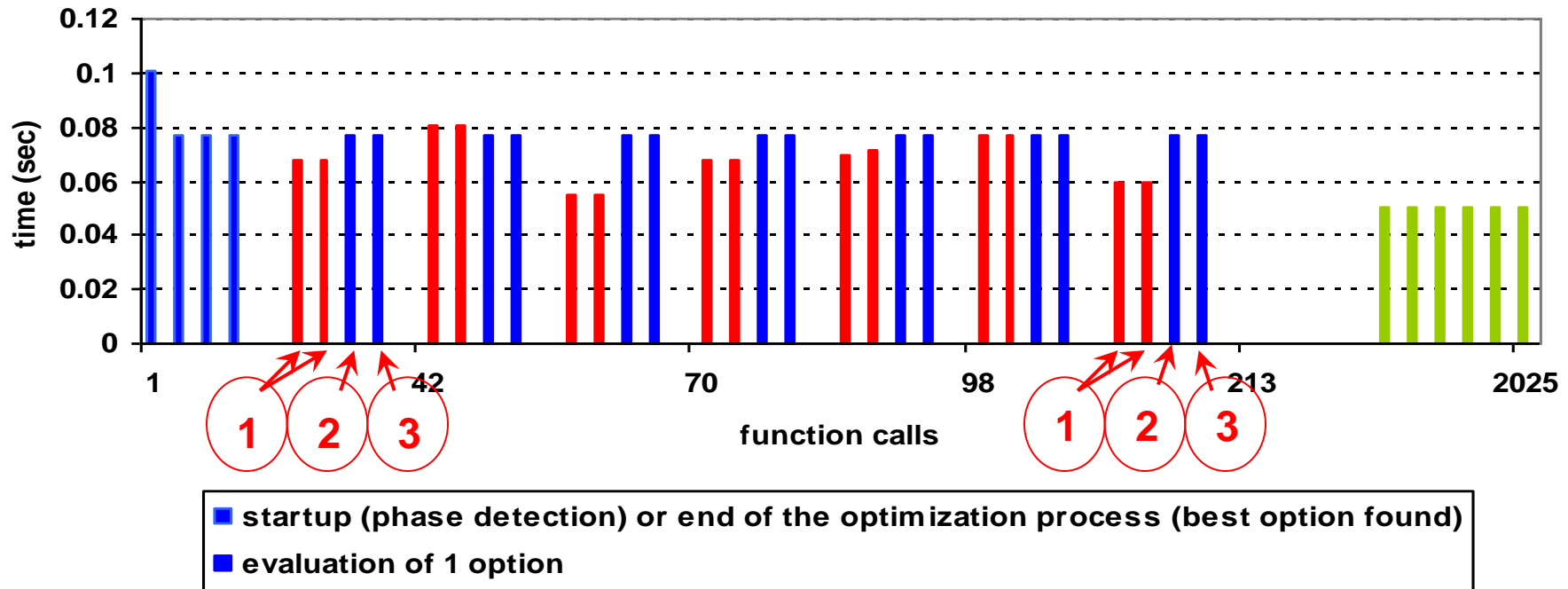
IPC for subroutine resid of benchmark mgrid across calls



- *Define stability by 3 consecutive or periodic executions with the same IPC*
- *Predict further occurrences with the same IPC (using period and length of regions with stable performance)*

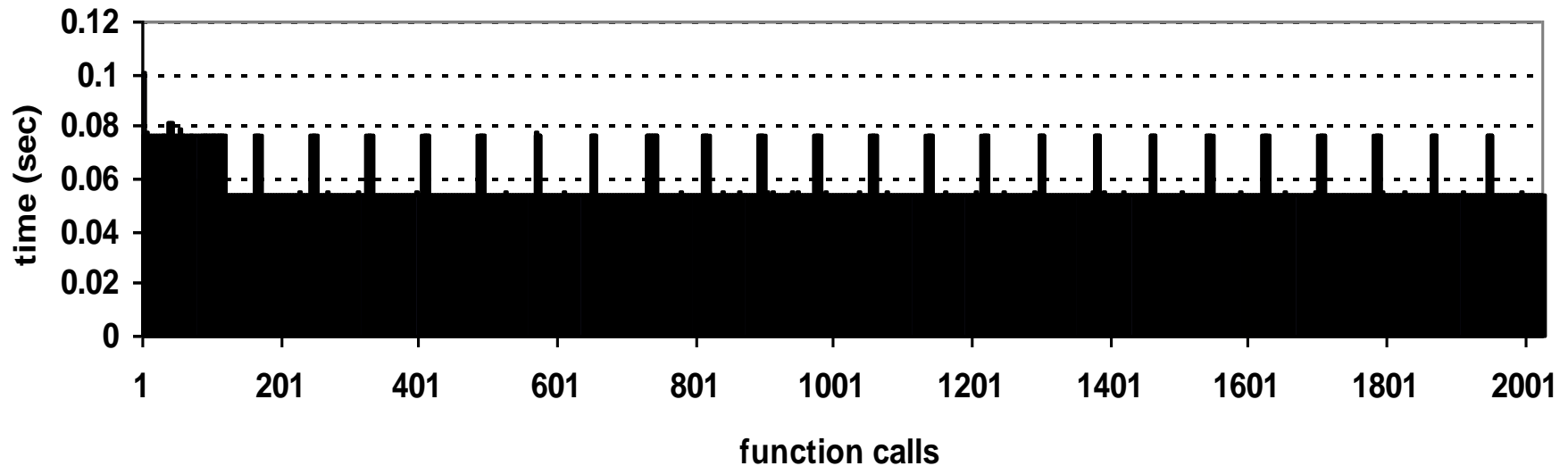
# Observations: program execution phases

Some programs exhibit stable behavior



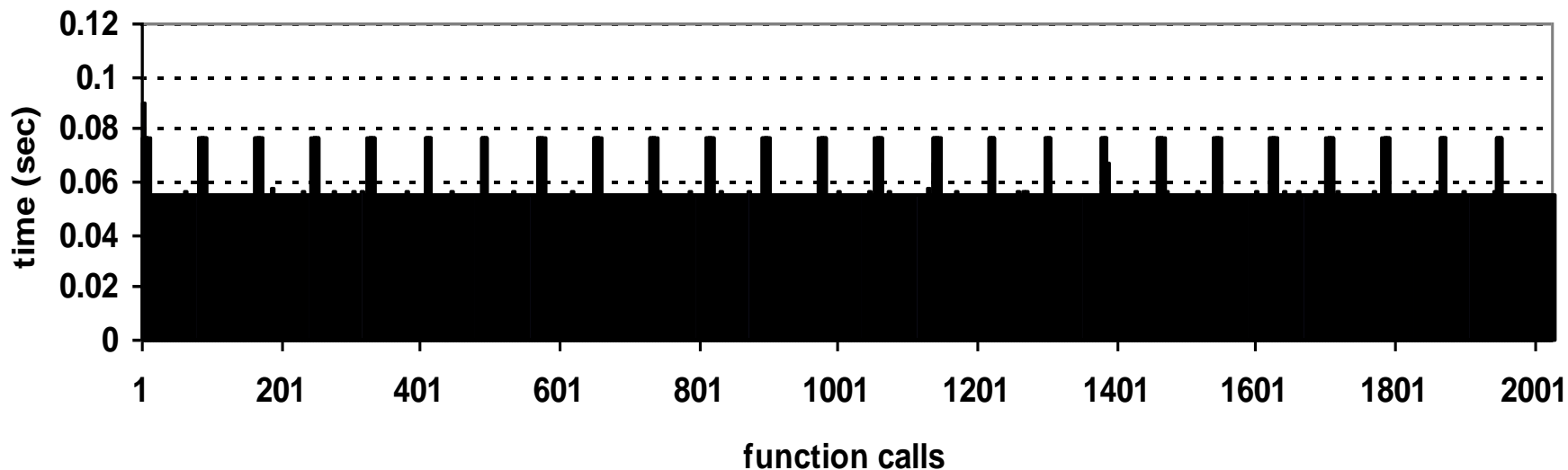
- 1) Consider clone with new optimization is evaluated after 2 consecutive executions of the code section with the same performance
- 2) Ignore one next execution to avoid transitional effects
- 3) Check baseline performance (to verify stability prediction)

# Observations: program execution phases



- Can transparently to end-user evaluate multiple optimizations
- Statically enable adaptive binaries (that can react to dataset or run-time state changes without any need for JIT or other complex frameworks)

# Transparent monitoring and adaptation of static programs

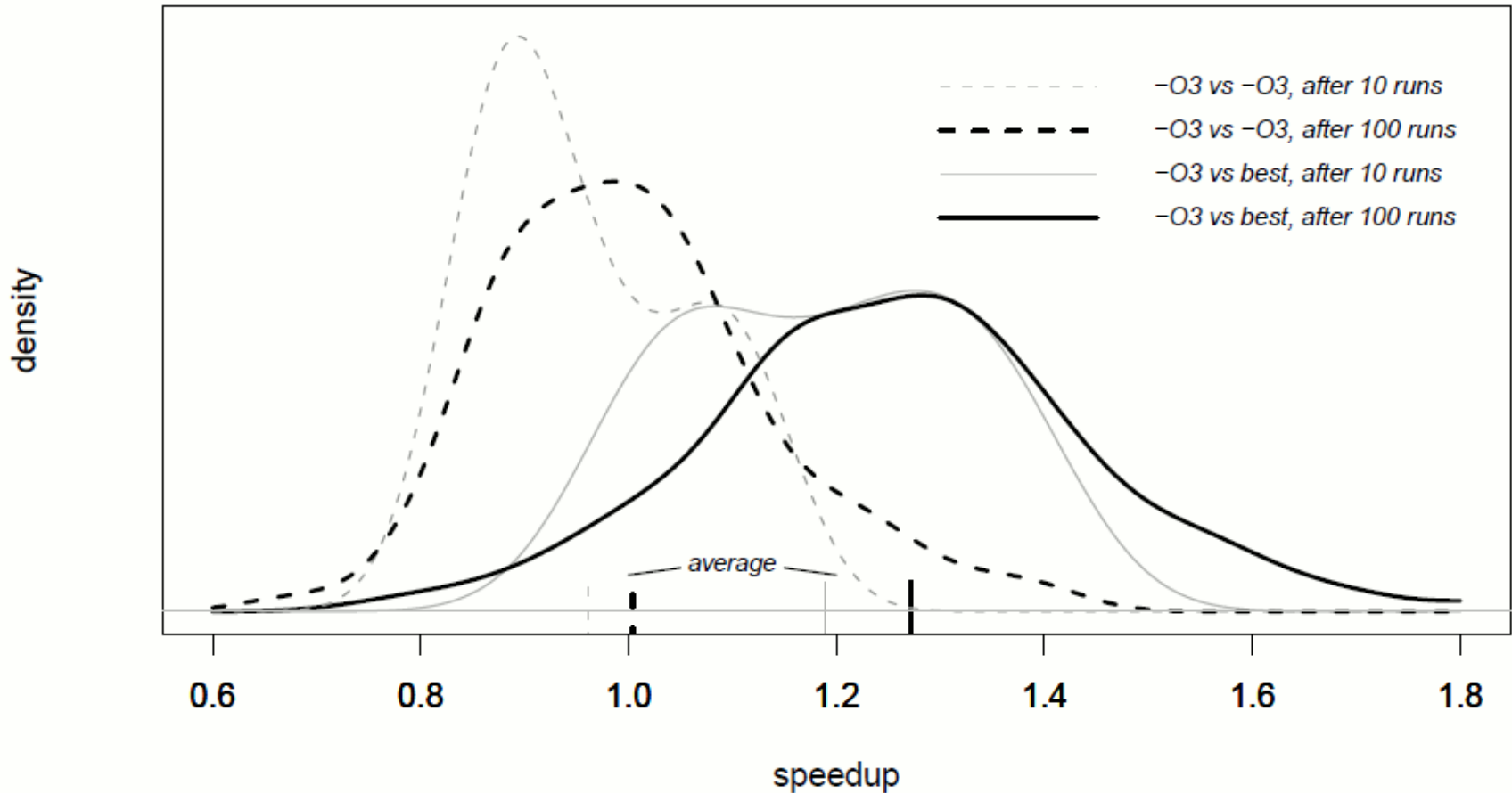


- Can transparently to end-user evaluate multiple optimizations
- Statically enable adaptive binaries (that can react to dataset or run-time state changes without any need for JIT or other complex frameworks)

• Grigori Fursin et al. *A Practical Method For Quickly Evaluating Program Optimizations*. *Proceedings of the 1<sup>st</sup> International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2005)*, number 3793 in LNCS, pages 29-46, Barcelona, Spain, November 2005 **Highest ranked paper**

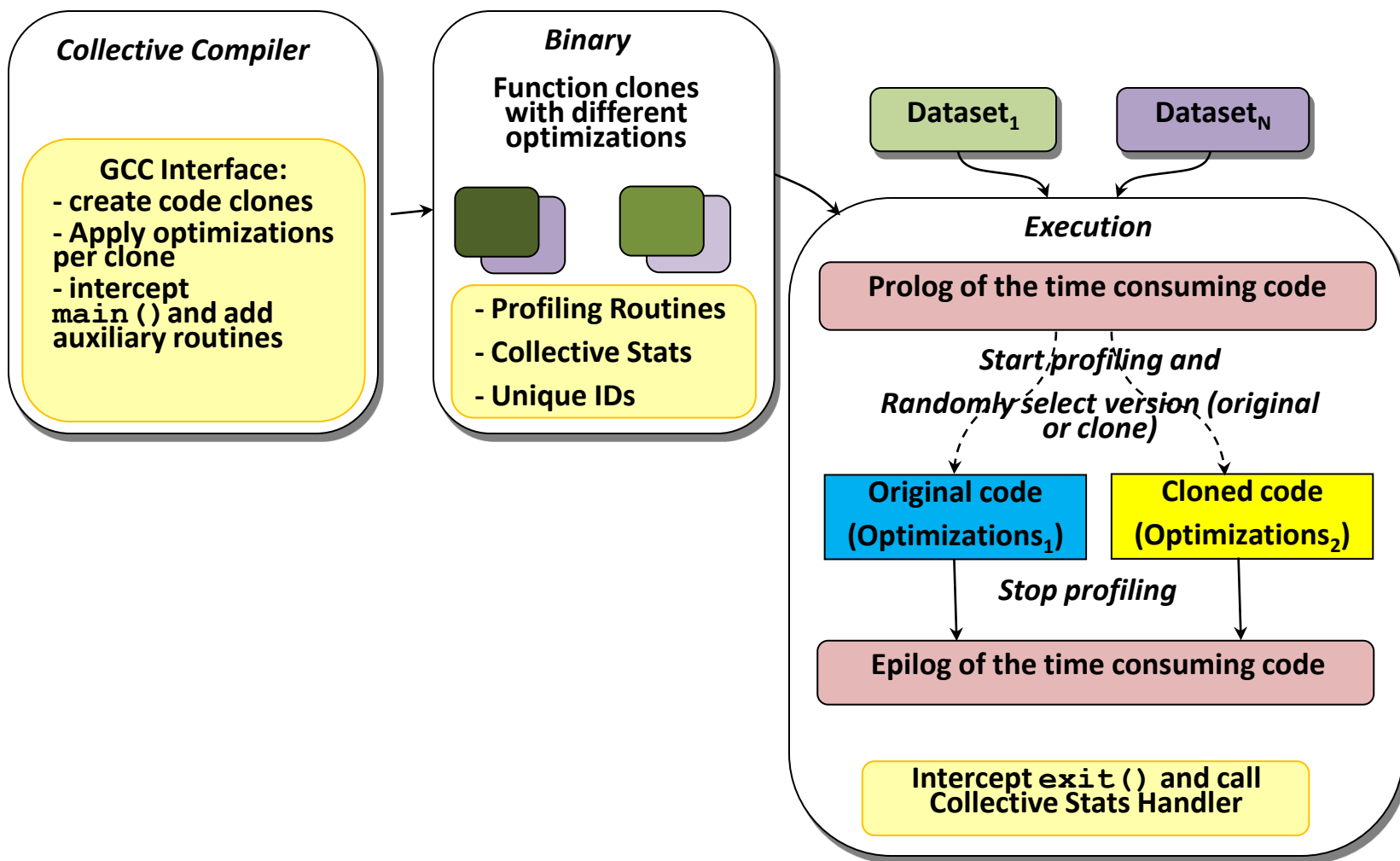
# Observations: random behavior

Randomly select versions at run-time  
Monitor speedup variation over time



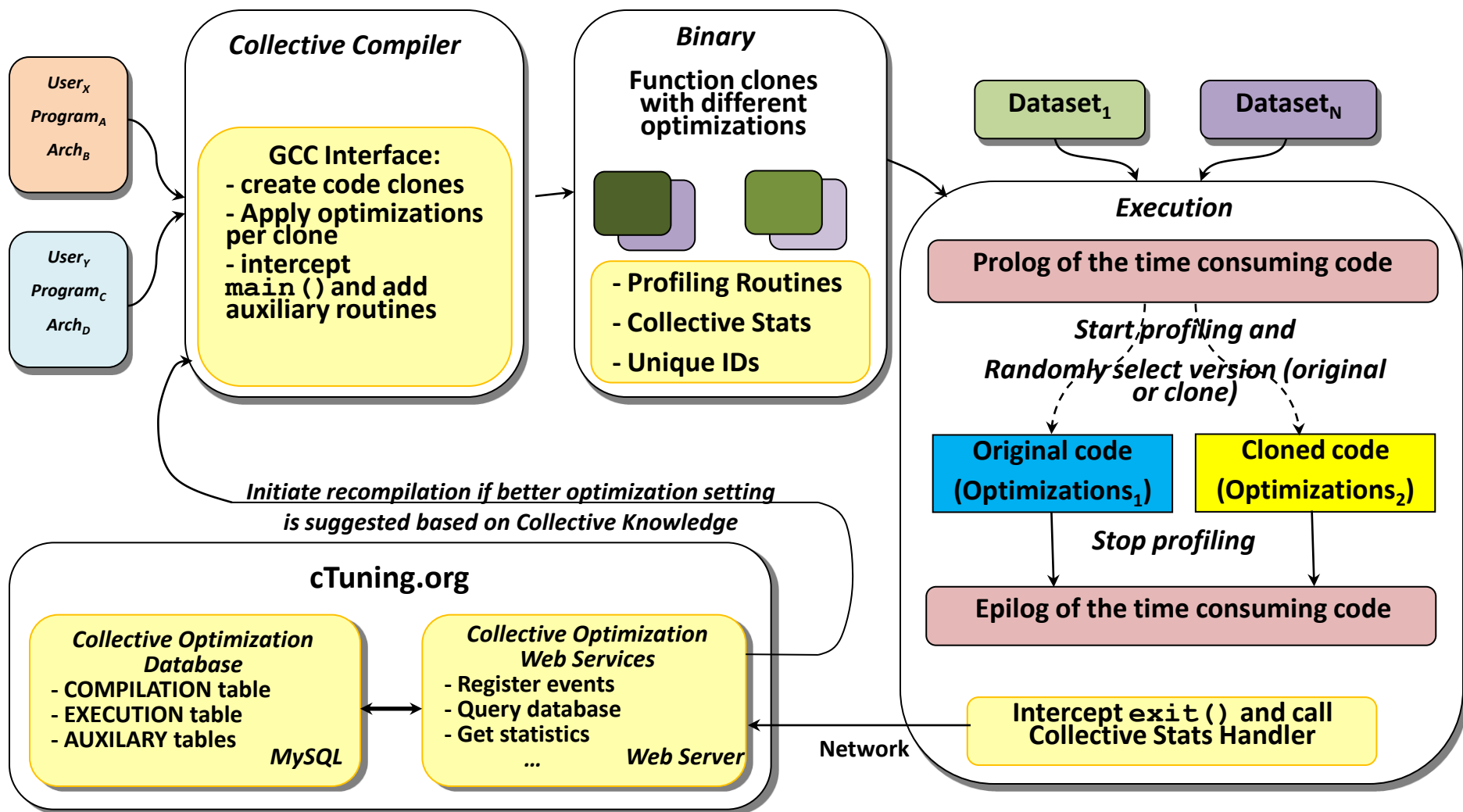
*jpeg decoder, GCC 4.5, Intel architecture*

# Transparently measuring the impact of optimizations





# Transparently measuring the impact of optimizations



# Speeding up research (2005-cur.)

- **Can observe behavior and evaluate optimizations in various GRID servers, cloud services, desktops, etc ...**
  - multiple benchmarks/datasets
  - multiple architectures
  - multiple compilers
  - multiple optimizations

**Opened up many interesting research opportunities,  
particularly for data mining and predictive modeling!**

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

**Concept is included into EU HiPEAC research vision 2012-2020**

• Grigori Fursin et al. **Collective optimization**. *Proceedings of the International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2009)*, Paphos, Cyprus, January 2009

# Collaborative exploration of large optimization spaces

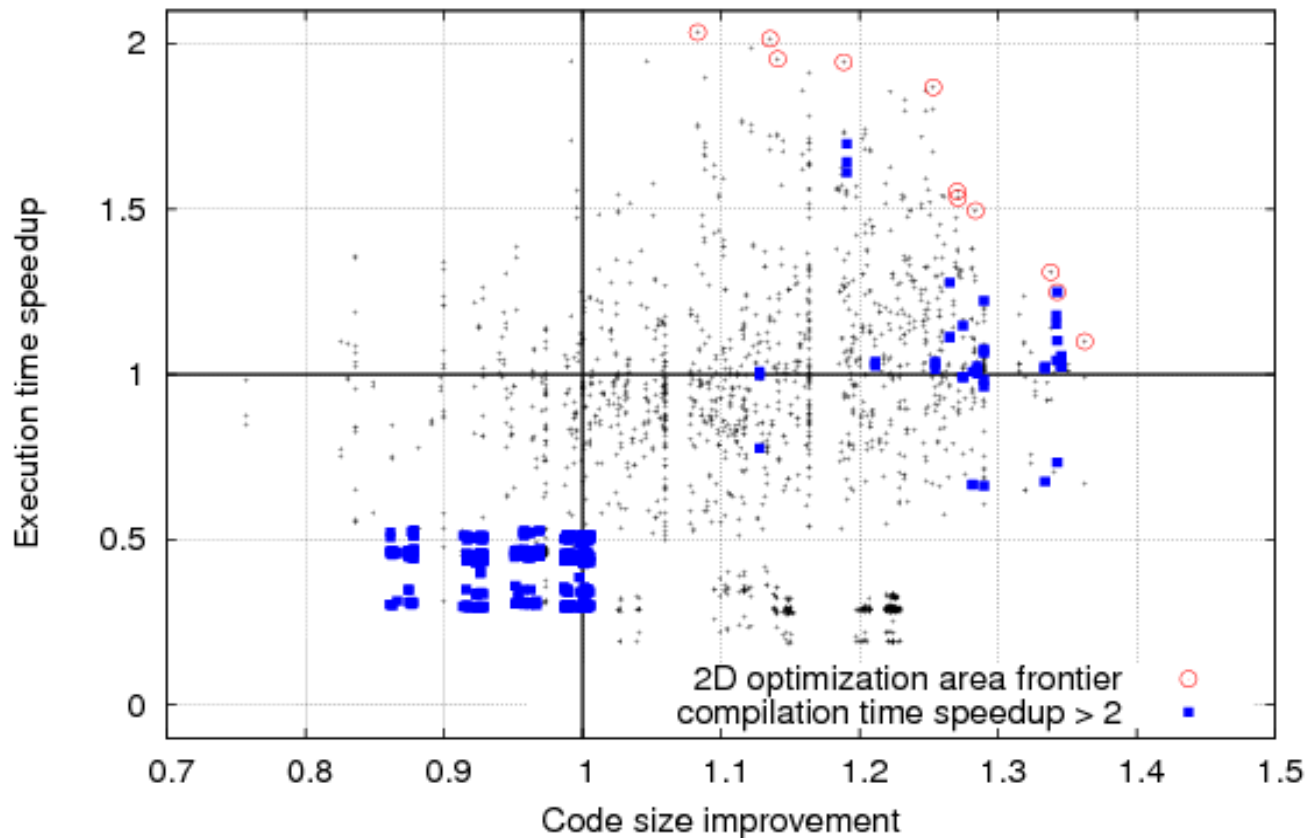
## Multi-objective optimizations (depend on user scenarios):

HPC and desktops: *improving execution time*

Data centers and real-time systems: *improving execution and compilation time*

Embedded systems: *improving execution time and code size*

Now additional requirement: *reduce power consumption*



susan corners kernel

Intel Core2

GCC 4.4.4

similar results on ICC 11.1

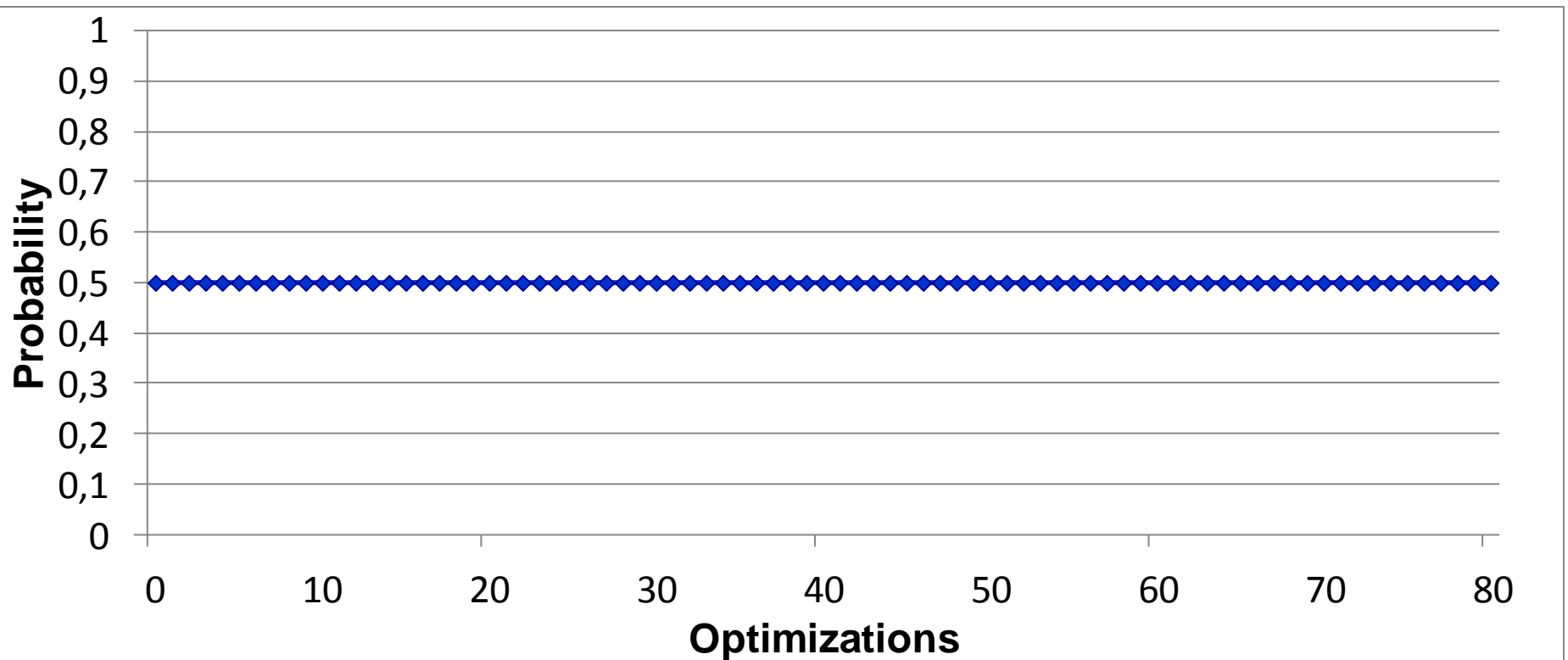
baseline opt=-O3

~100 optimizations

random combinations  
(50% probability)

Nowadays used for  
auto-parallelization,  
reduction of contentions,  
reduction of communication  
costs, etc.

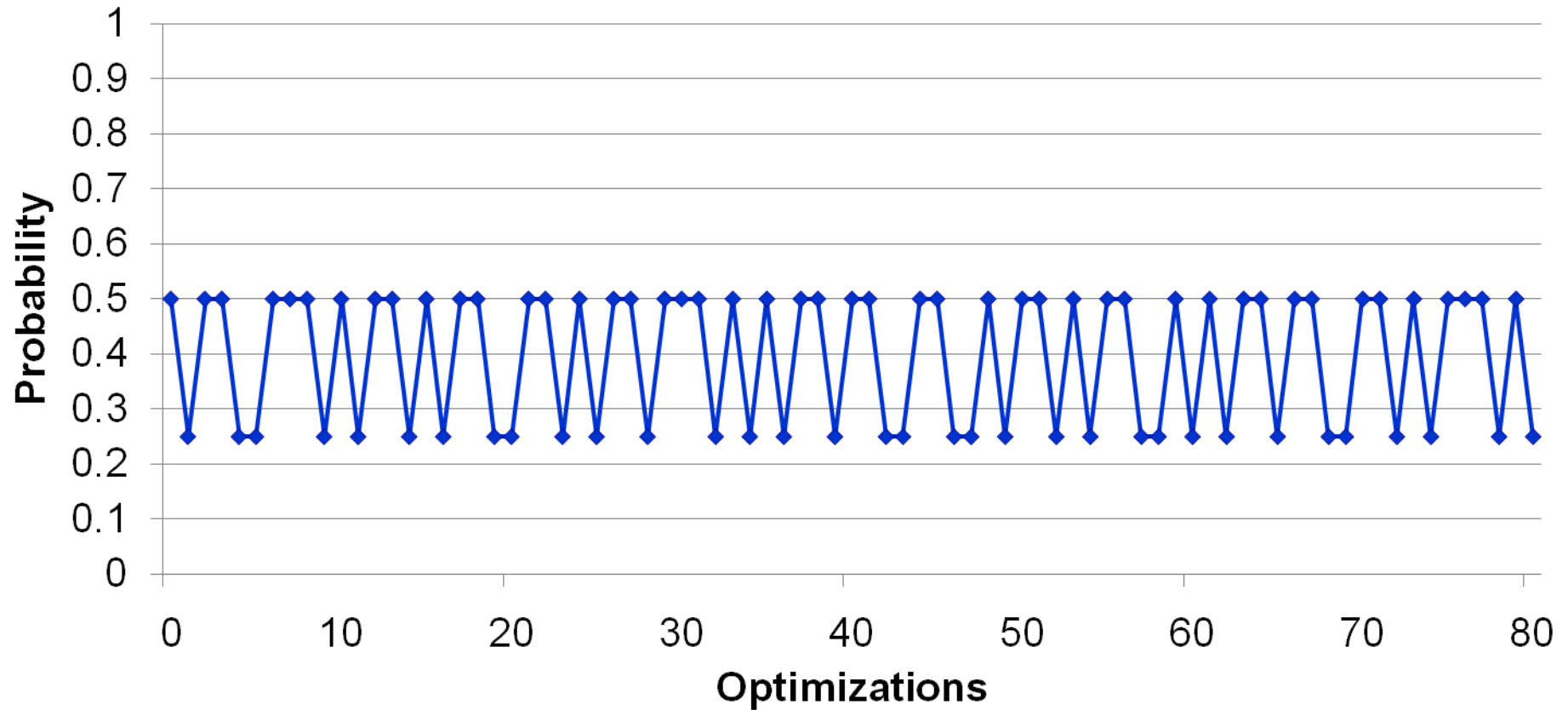
# Online focused exploration and learning



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

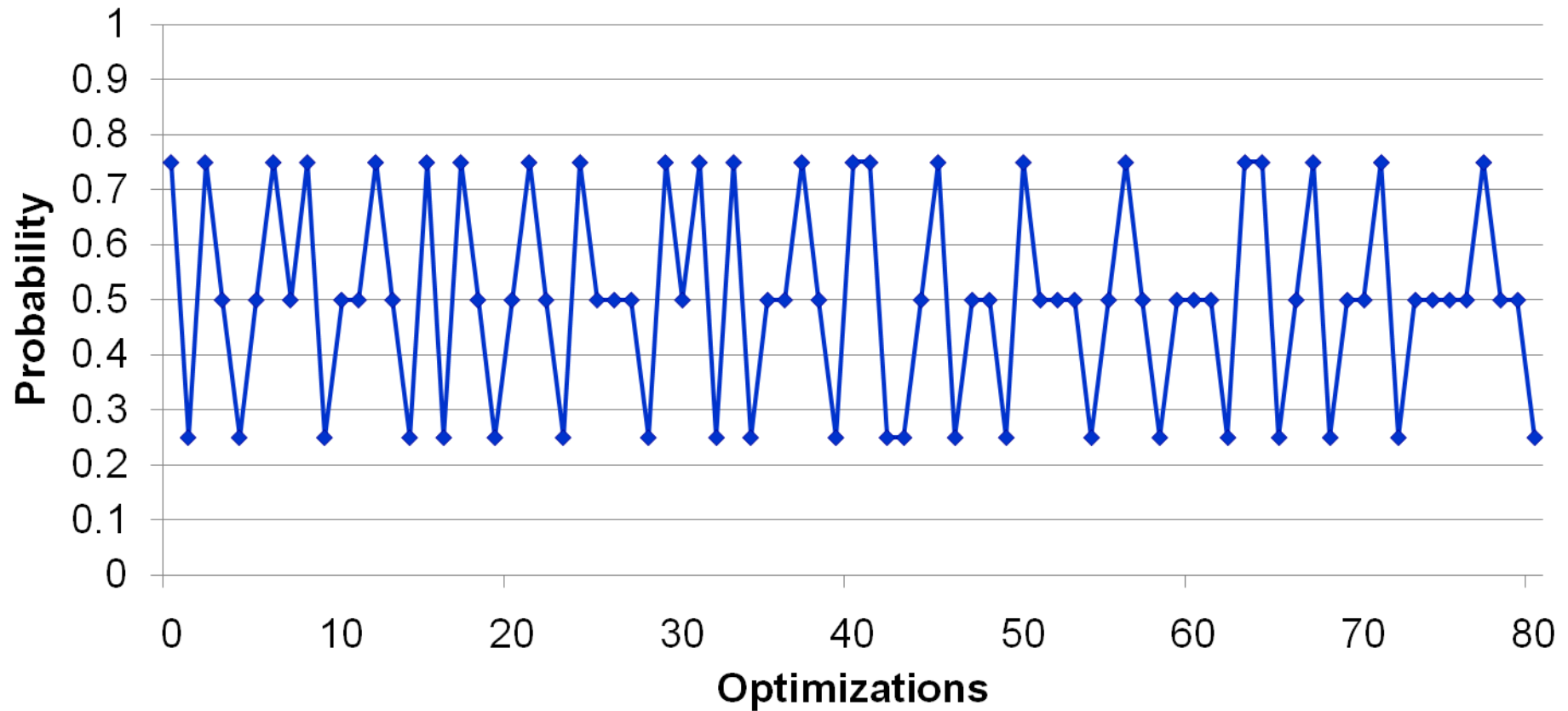
**Avoiding collection of huge amount of data -  
filtering (compacting) and learning space on the fly**

# Online focused exploration and learning



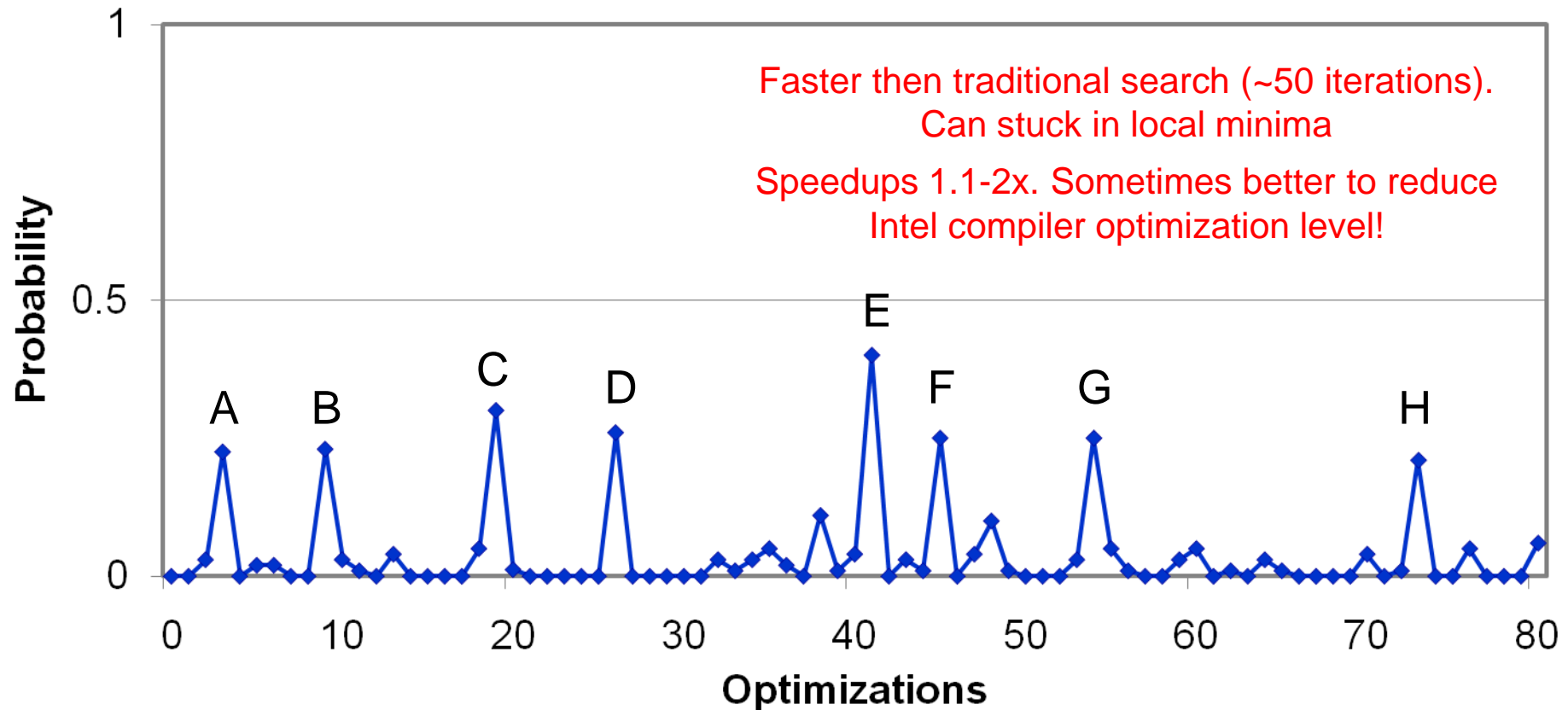
**Current random selection of optimizations reduced execution time:**  
*reduce probabilities of the selected optimizations*

# Online focused exploration and learning



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

# Online focused exploration and learning

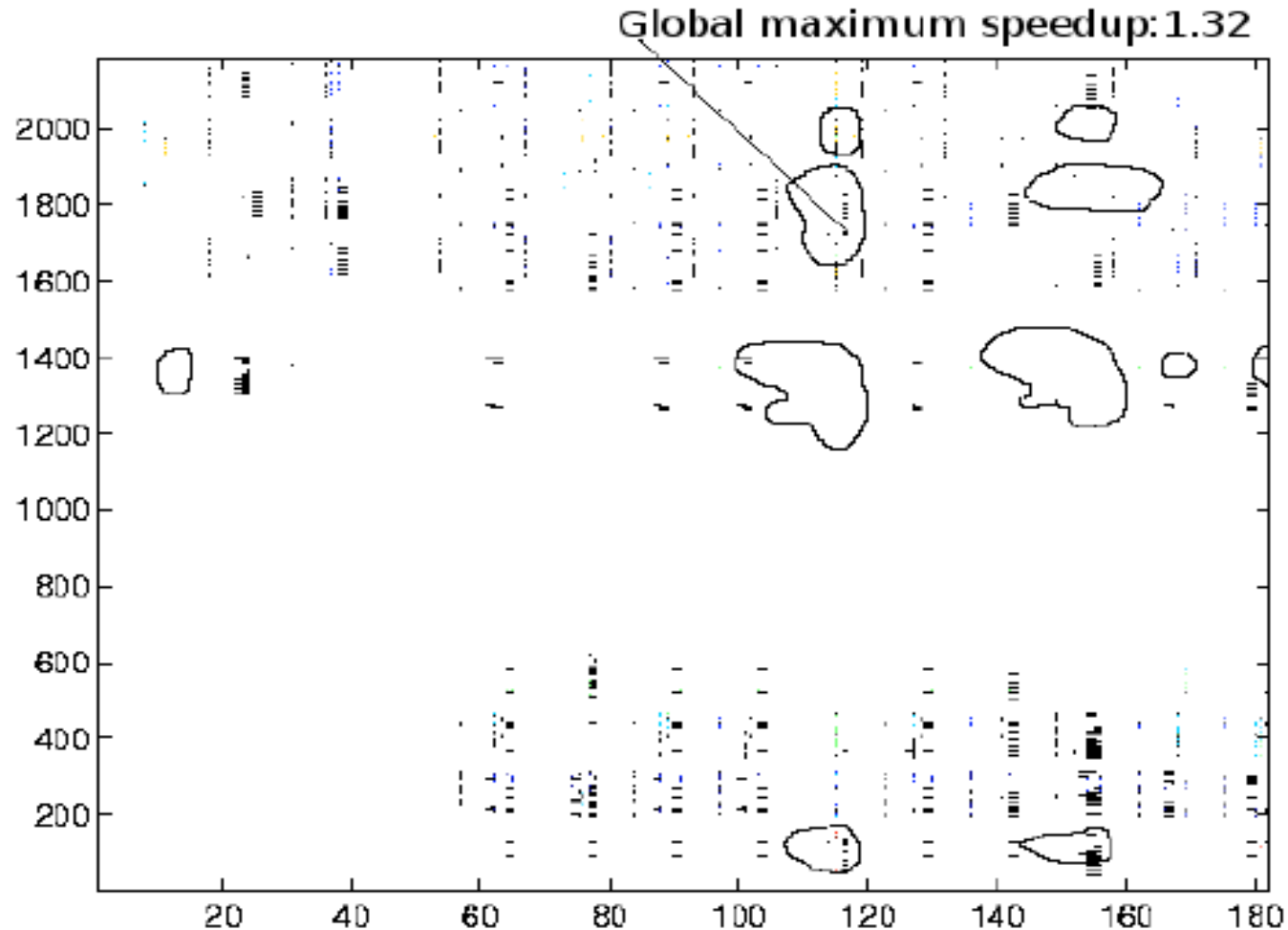


*“good optimizations” across all programs:*

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

# Online focused exploration and learning

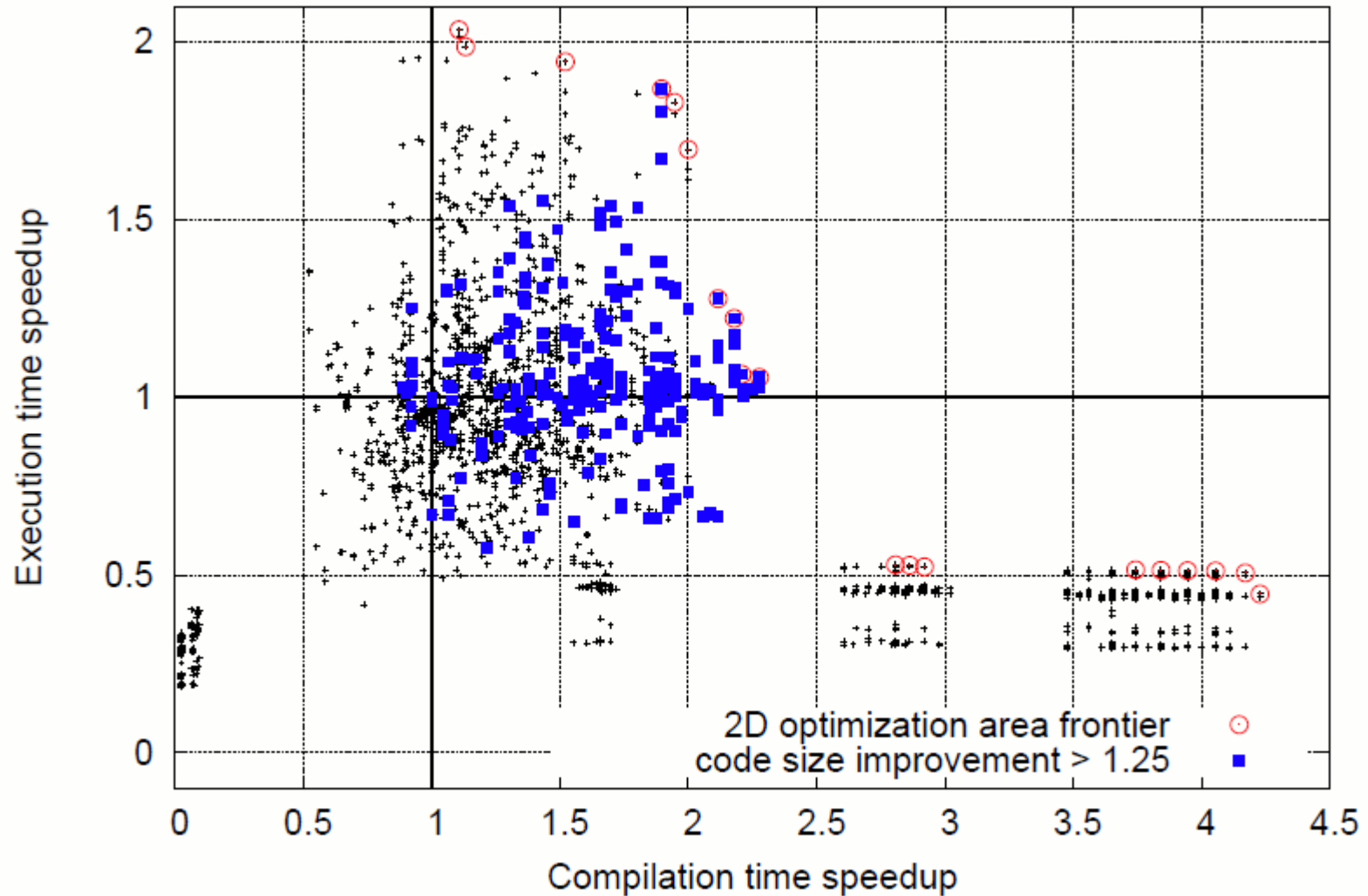


14 transformations, sequences of length 5, search space = **396000**

- 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



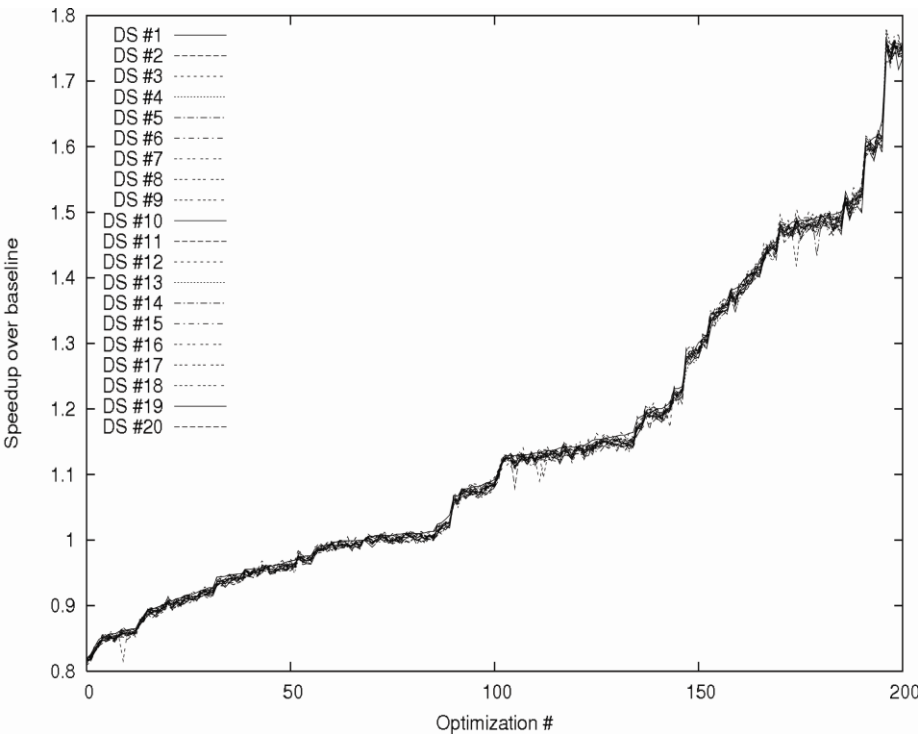
# Online probabilistic exploration



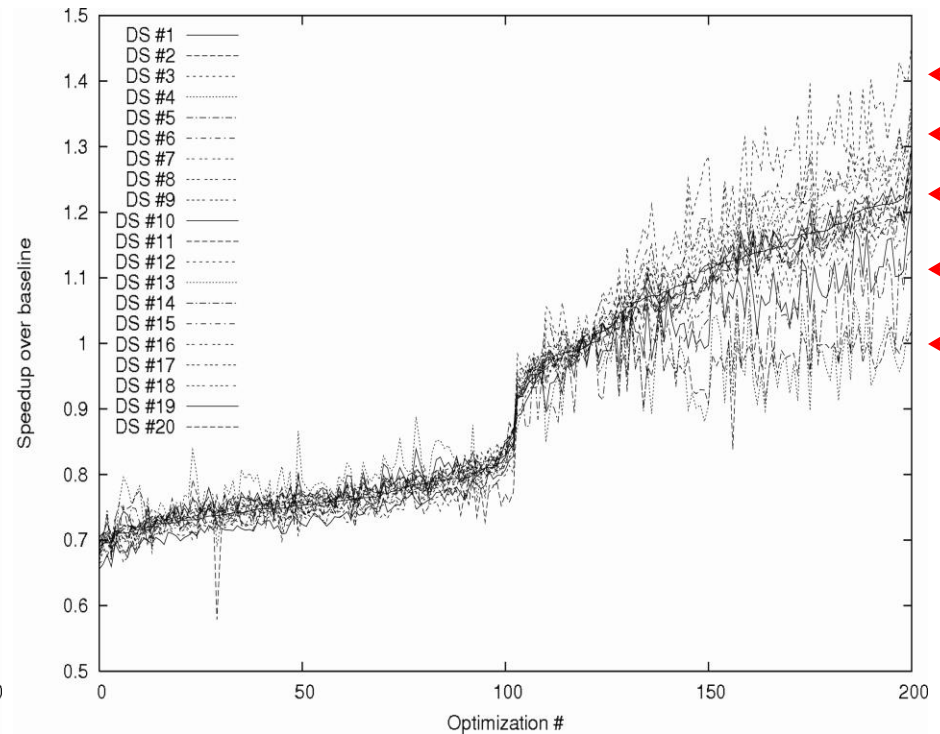
*AMD platform, GCC 4.5, image corner detection (susan\_corners)*

# Reactions to optimizations across multiple datasets

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



*dijkstra*  
(not sensitive)



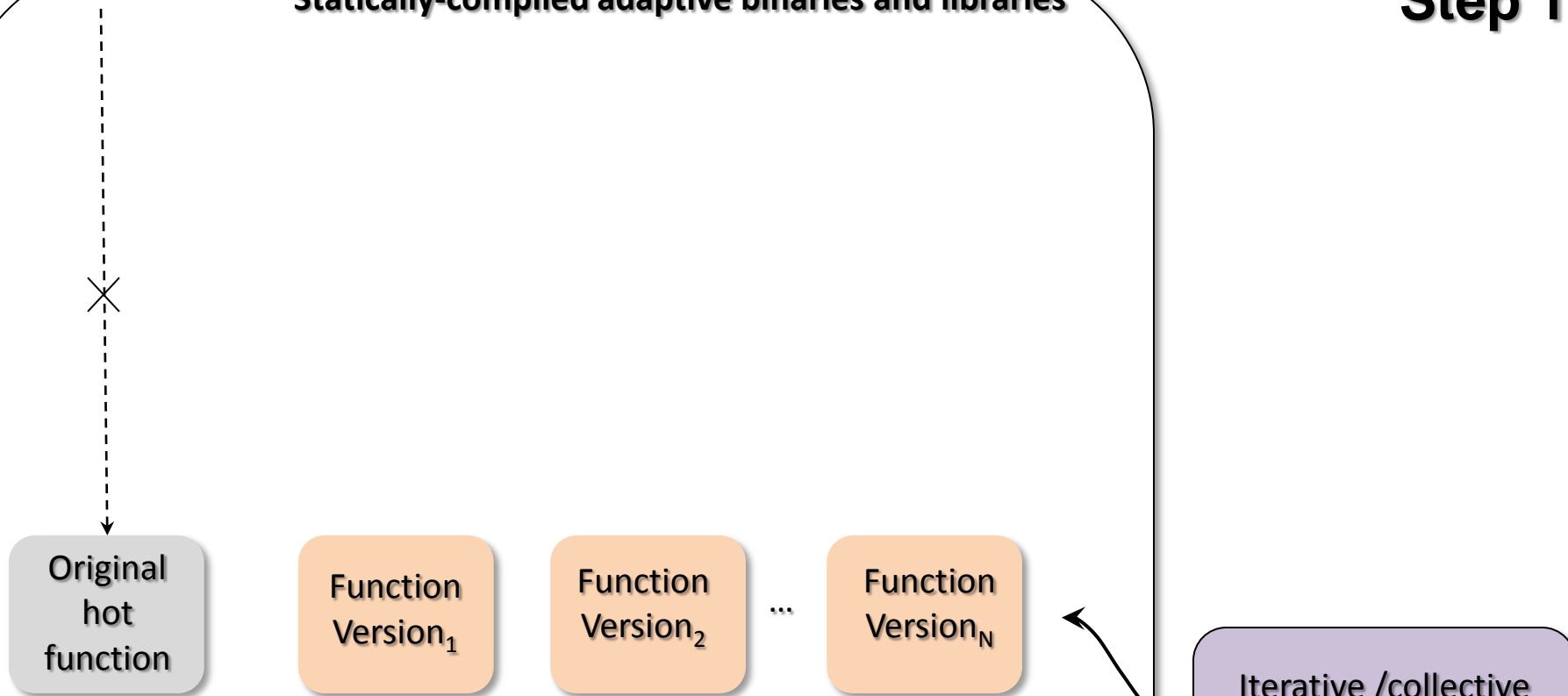
*jpeg\_d*  
(clustering)

<http://ctuning.org/cbench>

# Unifying adaptation of statically compiled programs

## Step 1

Statically-compiled adaptive binaries and libraries

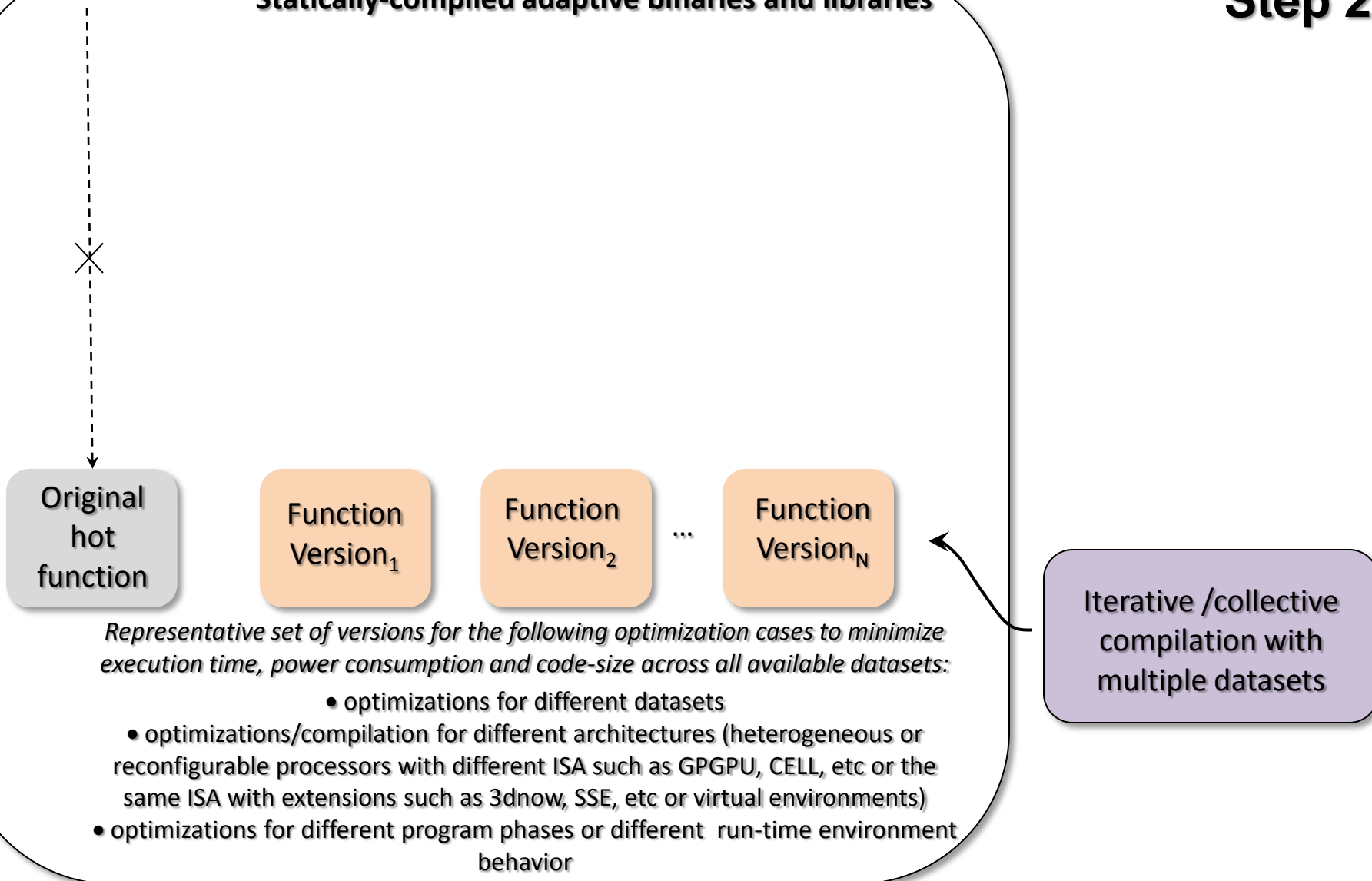


Iterative /collective  
compilation with  
multiple datasets

# Unifying adaptation of statically compiled programs

## Step 2

### Statically-compiled adaptive binaries and libraries



# Unifying adaptation of statically compiled programs

## Step 3

Statically-compiled adaptive binaries and libraries

Extract dataset features

***Selection mechanism optimized for low run-time overhead***

Original hot function

Function Version<sub>1</sub>

Function Version<sub>2</sub>

...

Function Version<sub>N</sub>

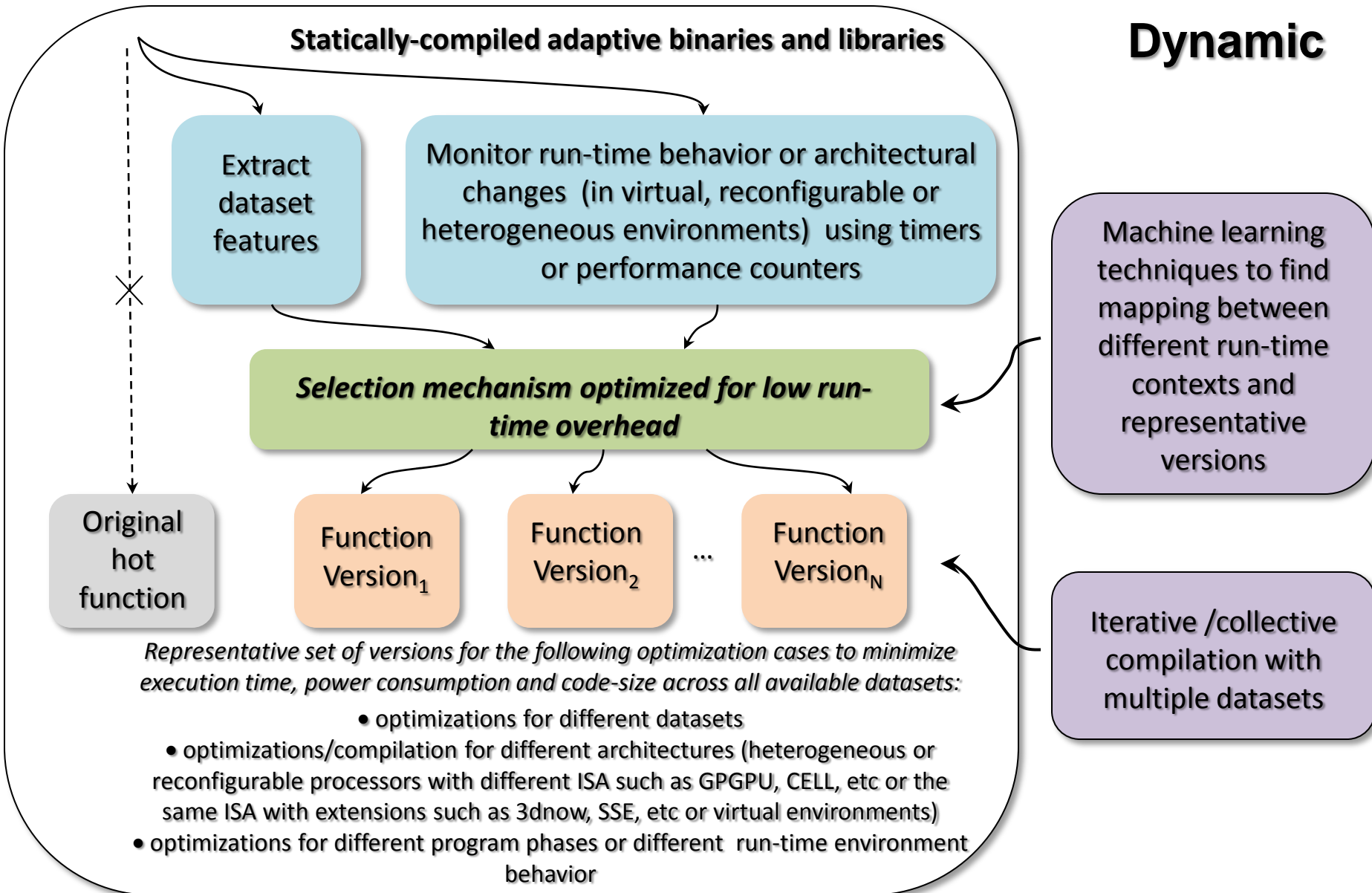
*Representative set of versions for the following optimization cases to minimize execution time, power consumption and code-size across all available datasets:*

- 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

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

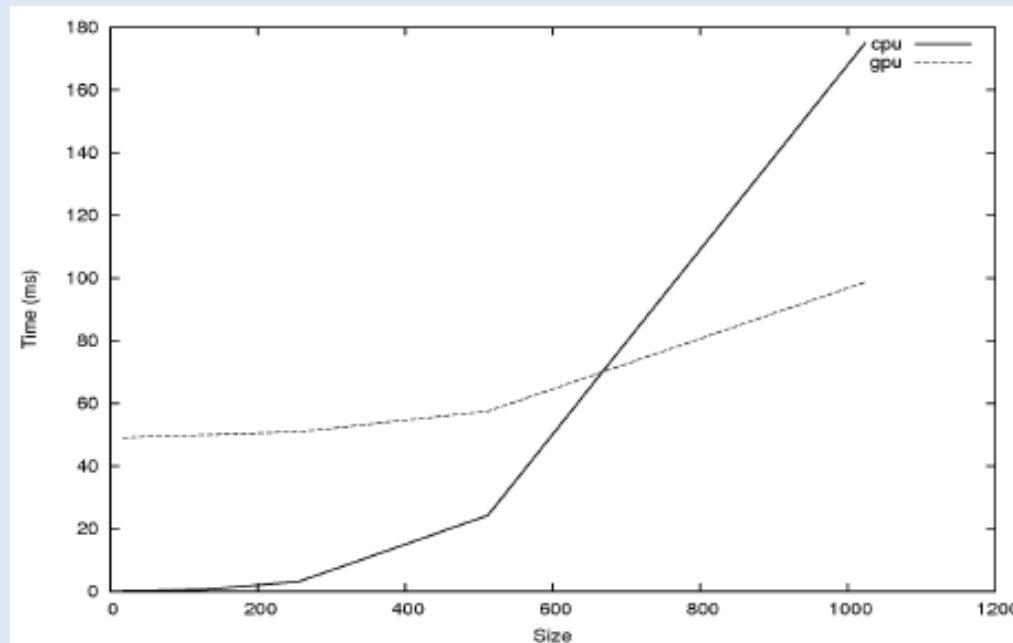
Iterative /collective compilation with multiple datasets

# Unifying adaptation of statically compiled programs



# Online tuning: adaptive scheduling

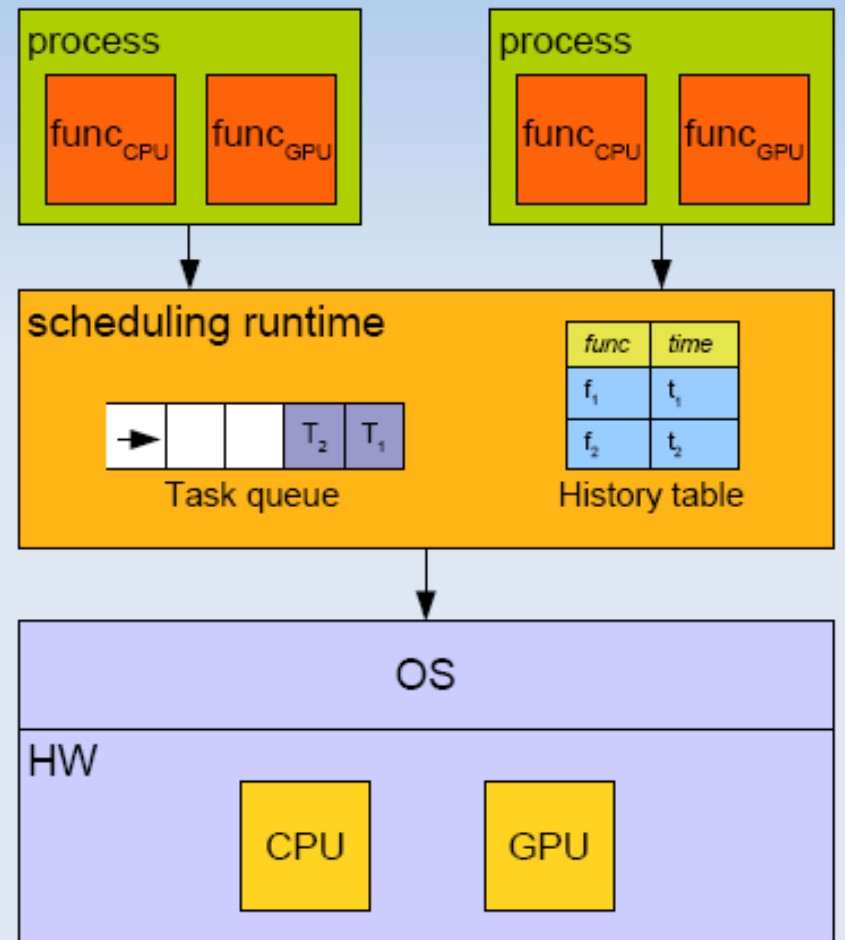
- Why do we need dynamic predictive scheduling?
  - Best fitting PE depends on the code, the input and the system current workload
- Example (matrix multiplication on single-CPU/GPU):



Matrix multiplication

# Online tuning: adaptive scheduling

- Granularity
  - Function-level
- Function versioning
  - Orthogonal to the scheduling problem
- Explicit data transfer



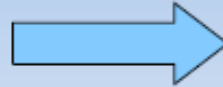
• Victor Jimenez, Isaac Gelado, Lluís Vilanova, Marisa Gil, Grigori Fursin and Nacho Navarro. Predictive runtime code scheduling for heterogeneous architectures. *Proceedings of the International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2009)*, Paphos, Cyprus, January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



# Online tuning: adaptive scheduling

1. Obtain both the CPU and the GPU versions from a current implementation

```
void  
matmul(float* A, float *B, float *C, int N);
```



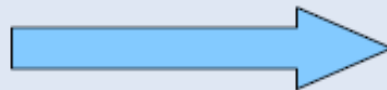
```
void matmul_cpu(float* A, float* B, float *C, int N);  
void matmul_gpu(float* A, float* B, float *C, int N);
```

versioning (CUDA, MCUDA)

2. Change calls to that function by a request to the scheduler (+ wrapper)



```
CallScheduler* cs = CallScheduler::getInstance();  
MatmulFunc* f = new MatmulFunc;  
MatmulArgs* a = new MatmulArgs(A,B,C,N);  
cs->schedule(f,a);
```

```
matmul(A,B,C,N);
```



compiler-generated

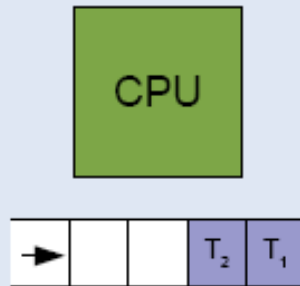
```
class MatmulFunc : public Func {  
    void run(PE::Type peType, Args* args) {  
        MatmulArgs* a = (MatmulArgs*) args;  
        switch (peType) {  
            case PE::CPU:  
                matmul_cpu(a->A,a->B,a->C,a->N);  
                break;  
            case PE::GPU:  
                matmul_gpu(a->A,a->B,a->C,a->N);  
                break;  
        }  
    }  
};
```

 original code  
 resulting code

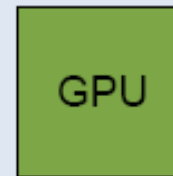
# Online tuning: adaptive scheduling

- Estimate history (*estimate-hist*)
  - Performance history table for every pair <function, PE>
  - Predict the waiting time for a new task in each PE's queue
  - Assign the task to the queue with the minimum waiting time
- Tries to reduce load unbalance

CPU history	
func	time
matmul	$time_{matmul}$
fft	$time_{fft}$



CPU task queue



GPU task queue

GPU history	
func	time
matmul	$time_{matmul}$
fft	$time_{fft}$

# Optimization knowledge reuse across programs

Started systematizing knowledge per program across datasets and architectures



# Optimization knowledge reuse across programs

Started systematizing knowledge per program across datasets and architectures

Program

Datasets

Architectures

**How to reuse knowledge among programs?**

Program

# Data mining and machine learning

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*

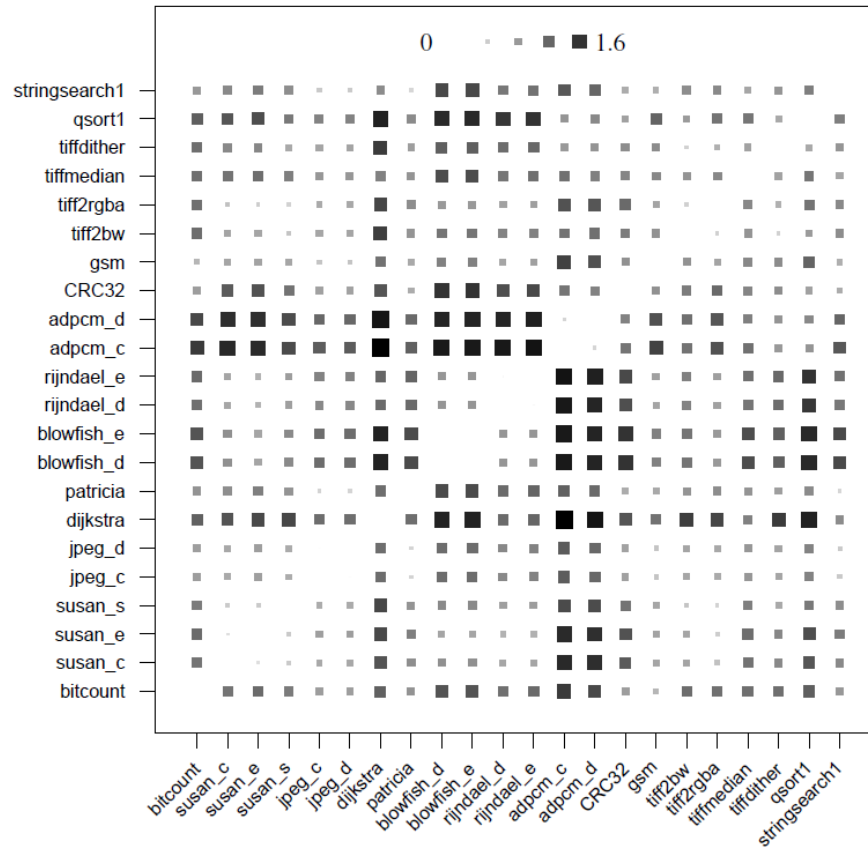
*ft55 - Number of static/extern variables that are pointers in the method*

## **Code patterns:**

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

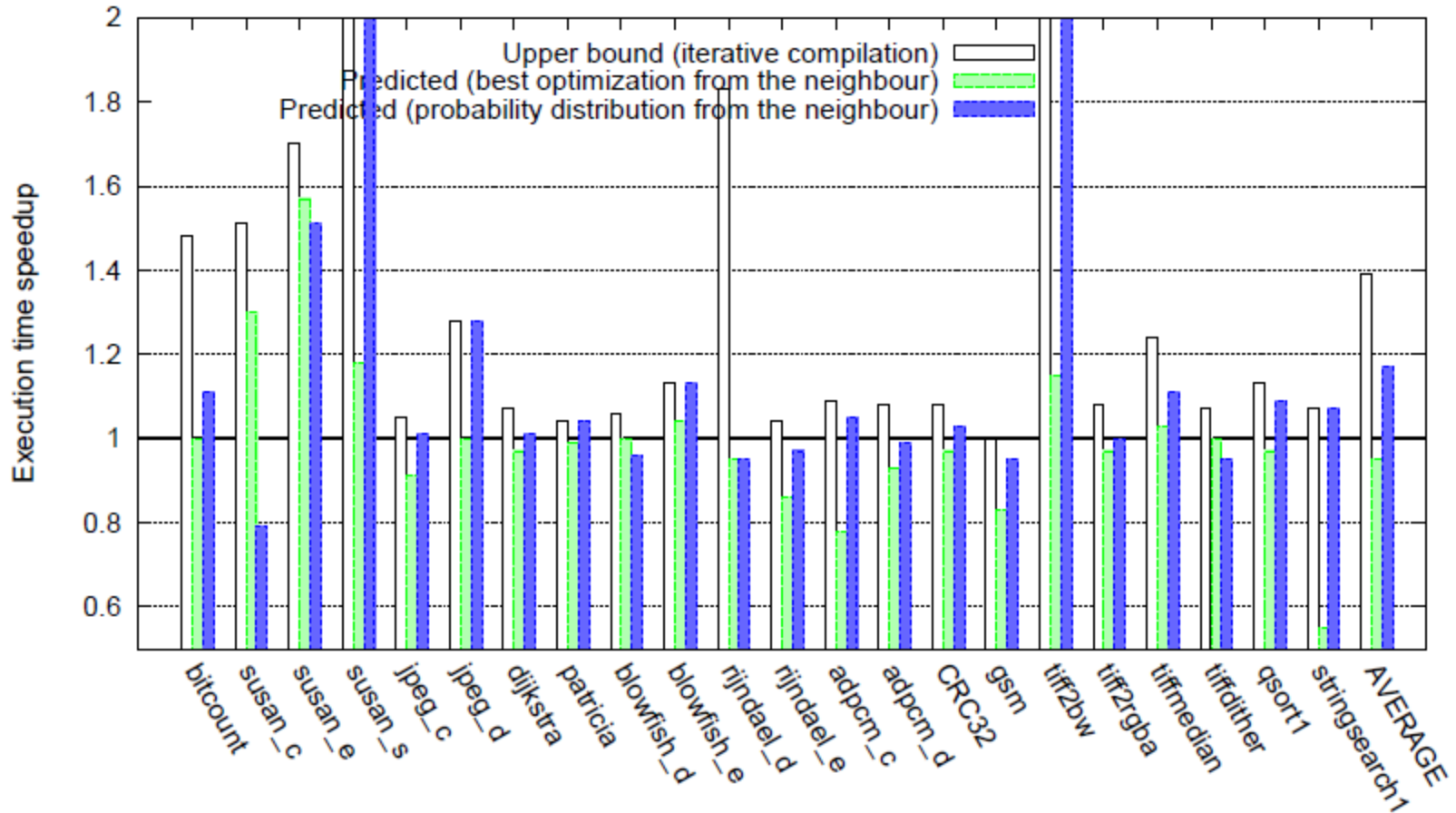
- 2) Correlate **features** and **objectives** in cTuning using **nearest neighbor classifiers, decision trees, SVM, fuzzy pattern matching, etc.**
- 3) Given **new** program, dataset, architecture, **predict behavior** based on **prior knowledge!**

# Nearest-neighbour classifier



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

# Optimization prediction



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

# Dynamic features

## *Principle Component Analysis:*

Most informative Performance Counters		
1) L1_TCA	2) L1_DCH	3) TLB_DM
4) BR_INS	5) RES_STL	6) TOT_CYC
7) L2_ICH	8) VEC_INS	9) L2_DCH
10) L2_TCA	11) L1_DCA	12) HW_INT
13) L2_TCH	14) L1_TCH	15) BR_MS

***Analysis of the importance of the performance counters.***

***The data contains one good optimization sequence per benchmark.***

***Calculating mutual information between a subset of the performance counters and good optimization sequences***

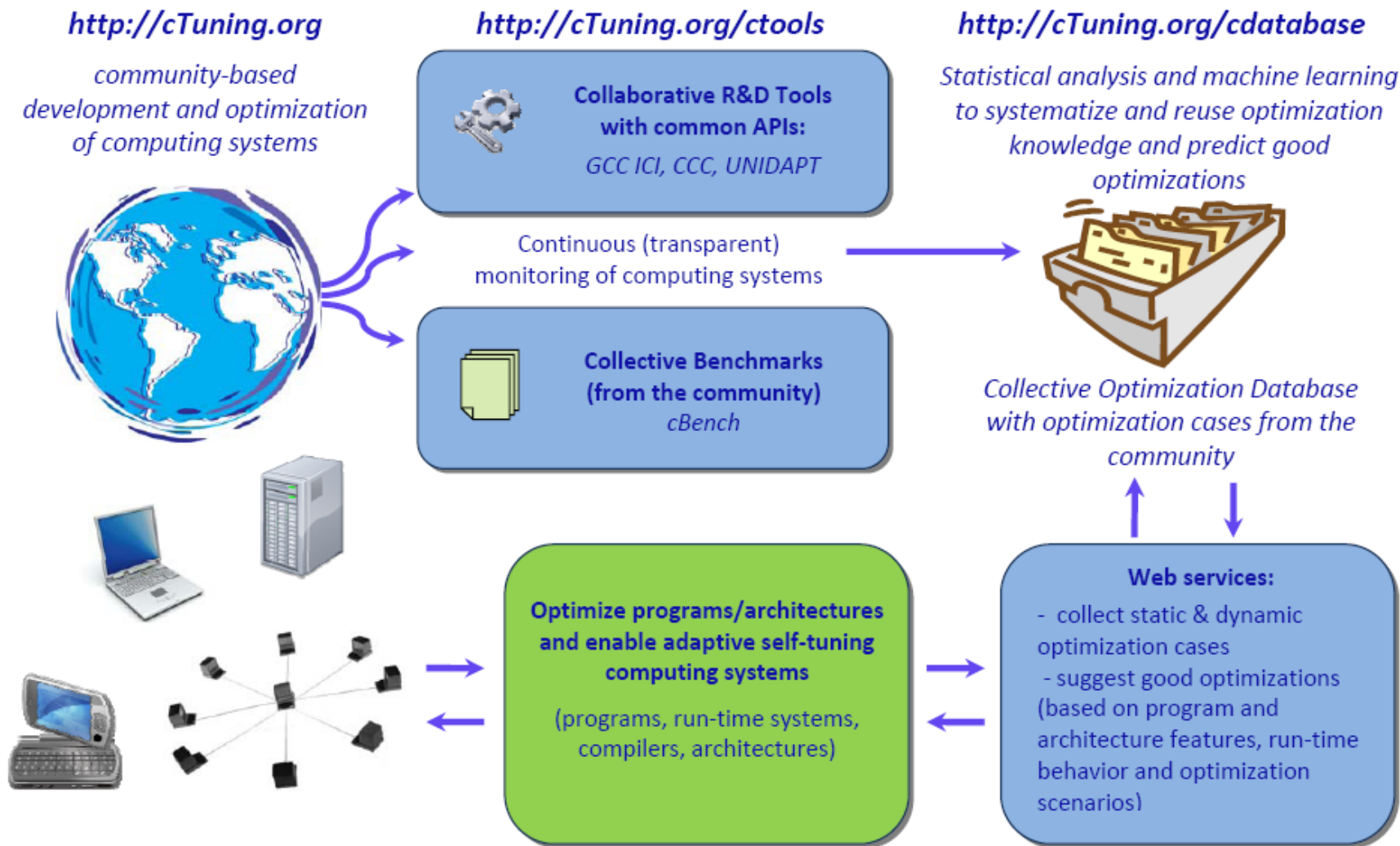
• John Cavazos, Grigori Fursin, Felix Agakov, Edwin Bonilla, Michael F.P.O'Boyle and Olivier Temam. *Rapidly Selecting Good Compiler Optimizations using Performance Counters. Proceedings of the 5th Annual International Symposium on Code Generation and Optimization (CGO), San Jose, USA, March 2007*



## And much more ...

- Analysis and detection of contentions in multi-core systems with shared cache
- Fast CPU/memory bound detection through breaking code semantics
- Software/hardware co-design (predicting better architecture designs)
- Performance/power balancing (through frequency variation)
- Decomposition of large applications into codelets for performance modeling

# Public Collective Tuning Portal (cTuning.org)



- Used in MILEPOST project (2007-2009) by IBM, CAPS, University of Edinburgh, INRIA to build first public machine-learning based compiler
- Opened for public access in 2009 to continue collaborative R&D

# Enabling reproducibility of results (new publication model)

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**

**Accepted as an EU HiPEAC theme (2012-2016)**

# What have we learnt from cTuning<sub>1</sub>

**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<sub>1</sub>

**It's fun working with the community!**

**Some comments about MILEPOST GCC from Slashdot.org:**

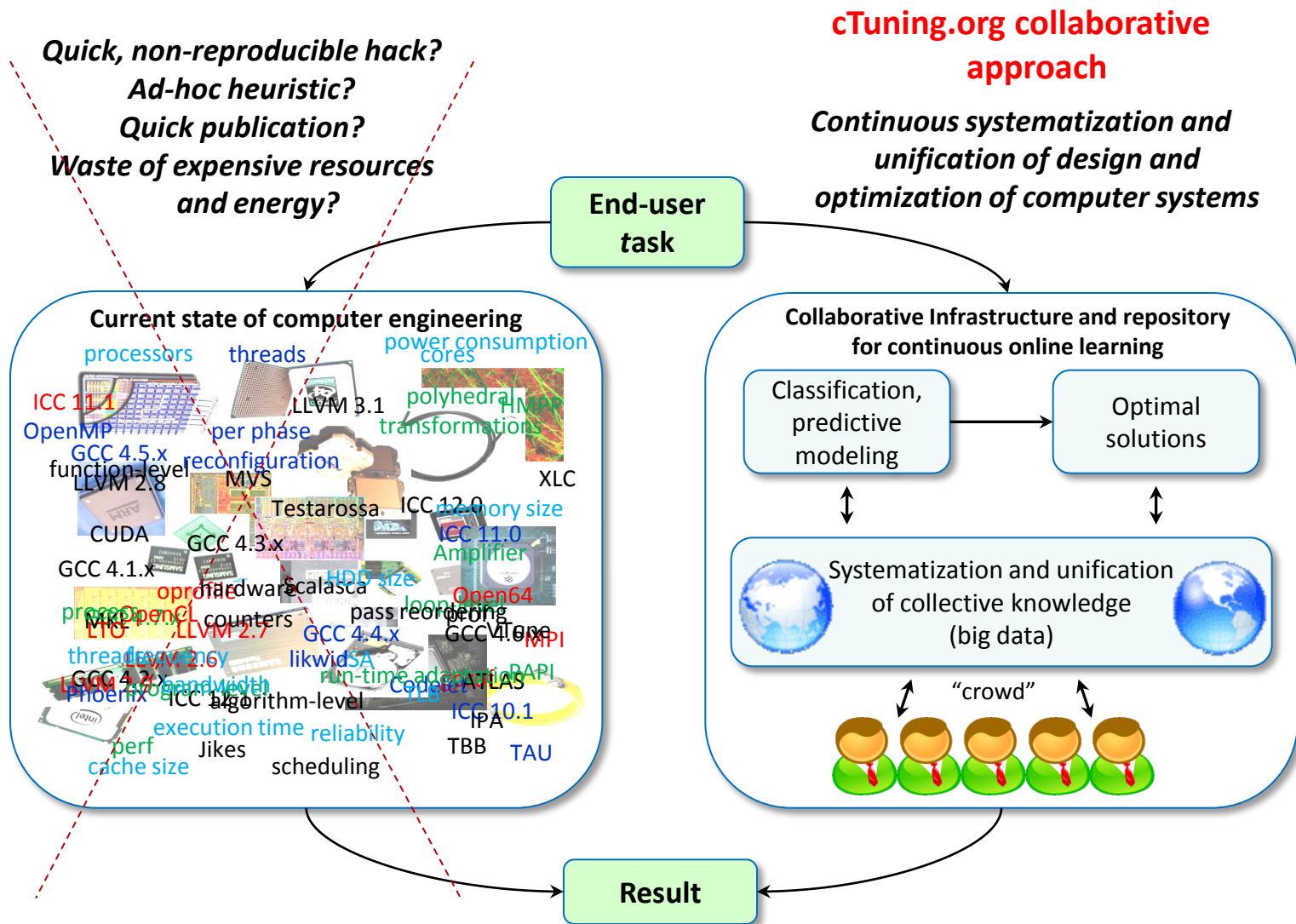
*<http://mobile.slashdot.org/story/08/07/02/1539252/using-ai-with-gcc-to-speed-up-mobile-design>*

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 was interested to validate and improve techniques!**

# Community is very interested in open “big data” for collaborative R&D



**Extrapolate collective knowledge to build faster and more power efficient computer systems to continue innovation in science and technology!**

# Conclusions - much more to be done!

**Now have public repository, tools, benchmarks, datasets and methodology that can help:**

Academia (students and researchers):

- Instead of losing time on developing tools for ever changing environments, focus on statistical, data mining and machine learning techniques to:
  - *unify program optimization, design space exploration, run-time adaptation*
  - *detect important characteristics of computer systems*
  - *detect representative benchmarks and data sets*
  - *evaluate multiple machine learning algorithms to predict optimizations or hardware designs or dynamic multi-objective adaptation (SVM, decision trees, hierarchical modeling, etc)*

Industry:

- *restore confidence in academic research due to reproducibility of results*
- *use and share collaborative tools*
- *share statistics about behavior of computer systems and optimizations*
- *expose choices and characteristics to end-users through unified interfaces*

# Conclusions - much more to be done!

## Challenges for public repositories and collaborative tools:

- Data management
  - MySQL vs schema-free databases
  - central vs distributed repository
  - performance vs portability
  - extensibility
  - online learning and data compaction
  - easy sharing
- Portability of the framework across different architectures, OSES, tools
- Interfaces to “open up” tools, architectures, applications for external tuning
  - simplicity and portability
- Reproducibility of experiments
- New publication model



# Preview of the 2<sup>nd</sup> talk

- Collective Mind: new plugin-based extensible infrastructure and schema-free repository for collaborative and holistic analysis and tuning of computer systems - will be released in May 2013 at HiPEAC computing week in Paris
- OpenME interface to “open up” compilers, run-time systems and applications for unified external tuning
- Hundreds of codelets, thousands of data sets, multiple packages prepared for various research scenarios on data mining
- Plugins for online auto-tuning and predictive modelling
- Portability across all major architectures and OS (Linux, Windows, Android)
- Collaboration with industry and academia

## Google discussion groups

*ctuning-discussions*

*collective-mind*

## Twitter

*c\_tuning*

*grigori\_fursin*

# Acknowledgements

- PhD students and postdocs (my Intel Exascale team)

*Abdul Wahid Memon, Pablo Oliveira, Yuriy Kashnikov*

- Colleague from NCAR, USA

*Daide Del Vento and his colleagues/interns*

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

- Colleagues from Intel (USA)

*David Kuck and David Wong*

- cTuning community:



- EU FP6, FP7 program and HiPEAC network of excellence

<http://www.hipeac.net>

# Main references

- Grigori Fursin. **Collective Tuning Initiative: automating and accelerating development and optimization of computing systems.** Proceedings of the GCC Summit'09, Montreal, Canada, June 2009
- Grigori Fursin and Olivier Temam. **Collective Optimization: A Practical Collaborative Approach.** ACM Transactions on Architecture and Code Optimization (TACO), December 2010, Volume 7, Number 4, pages 20-49
- Grigori Fursin, Yuriy Kashnikov, Abdul Wahid Memon, Zbigniew Chamski, Olivier Temam, Mircea Namolaru, Elad Yom-Tov, Bilha Mendelson, Ayal Zaks, Eric Courtois, Francois Bodin, Phil Barnard, Elton Ashton, Edwin Bonilla, John Thomson, Chris Williams, Michael O'Boyle. **MILEPOST GCC: machine learning enabled self-tuning compiler.** International Journal of Parallel Programming (IJPP), June 2011, Volume 39, Issue 3, pages 296-327
- Yang Chen, Shuangde Fang, Yuanjie Huang, Lieven Eeckhout, Grigori Fursin, Olivier Temam and Chengyong Wu. **Deconstructing iterative optimization.** ACM Transactions on Architecture and Code Optimization (TACO), October 2012, Volume 9, Number 3
- Yang Chen, Yuanjie Huang, Lieven Eeckhout, Grigori Fursin, Liang Peng, Olivier Temam, Chengyong Wu. **Evaluating Iterative Optimization across 1000 Data Sets.** PLDI'10
- Victor Jimenez, Isaac Gelado, Lluís Vilanova, Marisa Gil, Grigori Fursin and Nacho Navarro. **Predictive runtime code scheduling for heterogeneous architectures.** HiPEAC'09

# Main references

- Lianjie Luo, Yang Chen, Chengyong Wu, Shun Long and Grigori Fursin. **Finding representative sets of optimizations for adaptive multiversioning applications.** SMART'09 co-located with HiPEAC'09
- Grigori Fursin, John Cavazos, Michael O'Boyle and Olivier Temam. **MiDataSets: Creating The Conditions For A More Realistic Evaluation of Iterative Optimization.** HiPEAC'07
- F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M.F.P. O'Boyle, J. Thomson, M. Toussaint and C.K.I. Williams. **Using Machine Learning to Focus Iterative Optimization.** CGO'06
- Grigori Fursin, Albert Cohen, Michael O'Boyle and Oliver Temam. **A Practical Method For Quickly Evaluating Program Optimizations.** HiPEAC'05
- Grigori Fursin, Mike O'Boyle, Olivier Temam, and Gregory Watts. **Fast and Accurate Method for Determining a Lower Bound on Execution Time.** Concurrency Practice and Experience, 16(2-3), pages 271-292, 2004
- Grigori Fursin. **Iterative Compilation and Performance Prediction for Numerical Applications.** Ph.D. thesis, University of Edinburgh, Edinburgh, UK, January 2004