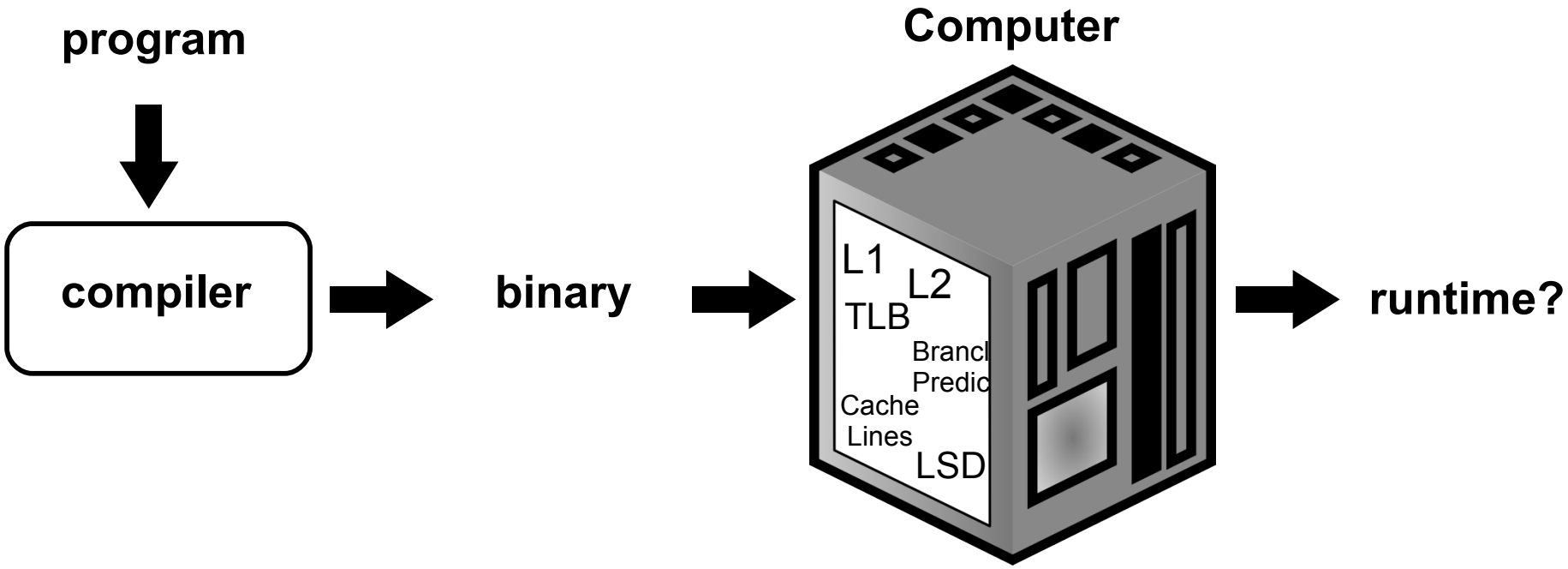
Blind Optimization
for Exploiting Hardware Features

Dan Knights, Todd Mytkowicz, Peter F. Sweeney*,
Michael C. Mozer, and Amer Diwan

University of Colorado at Boulder

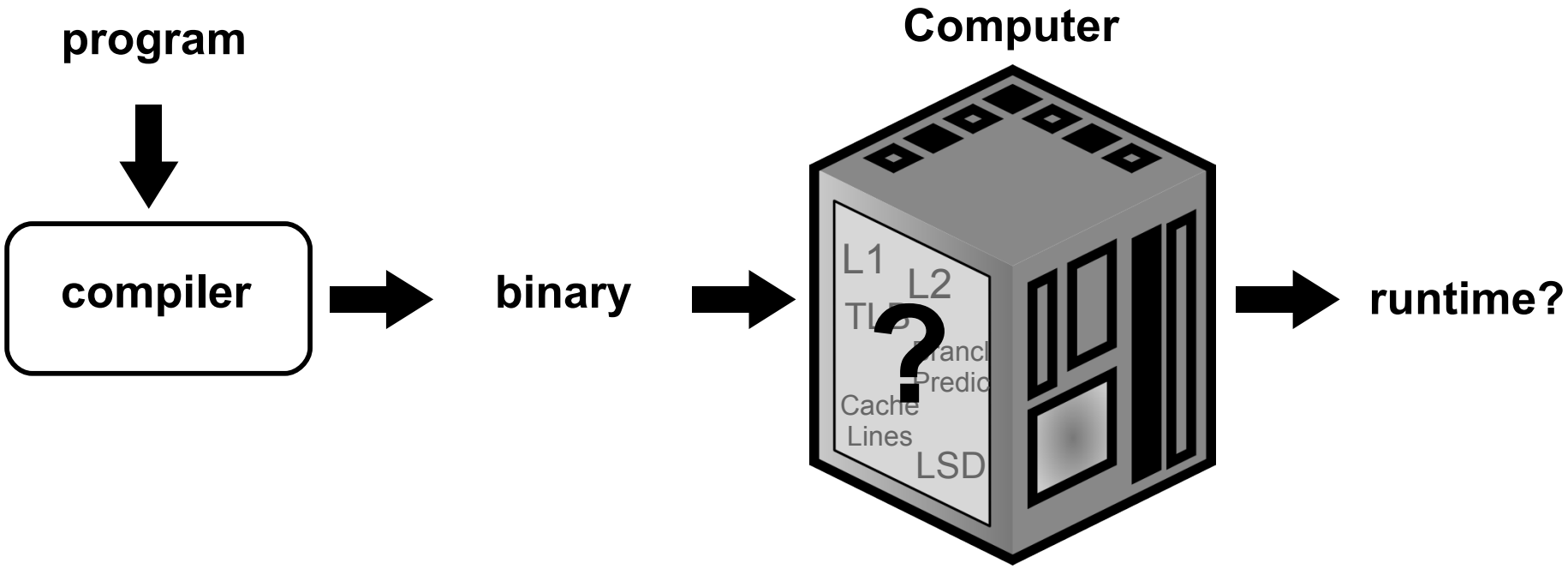
*IBM Research, Hawthorne

Motivation: Microprocessors are complex



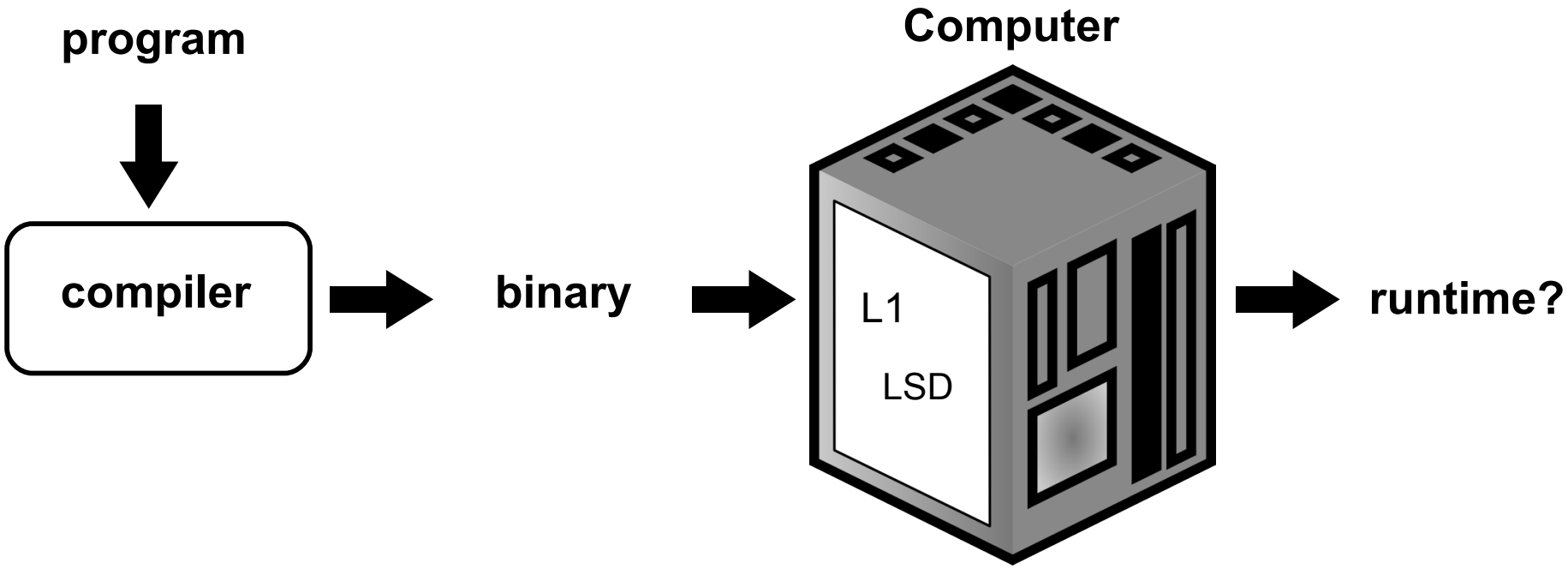
- Hard to optimize for all hardware features simultaneously

Motivation: Some features hidden



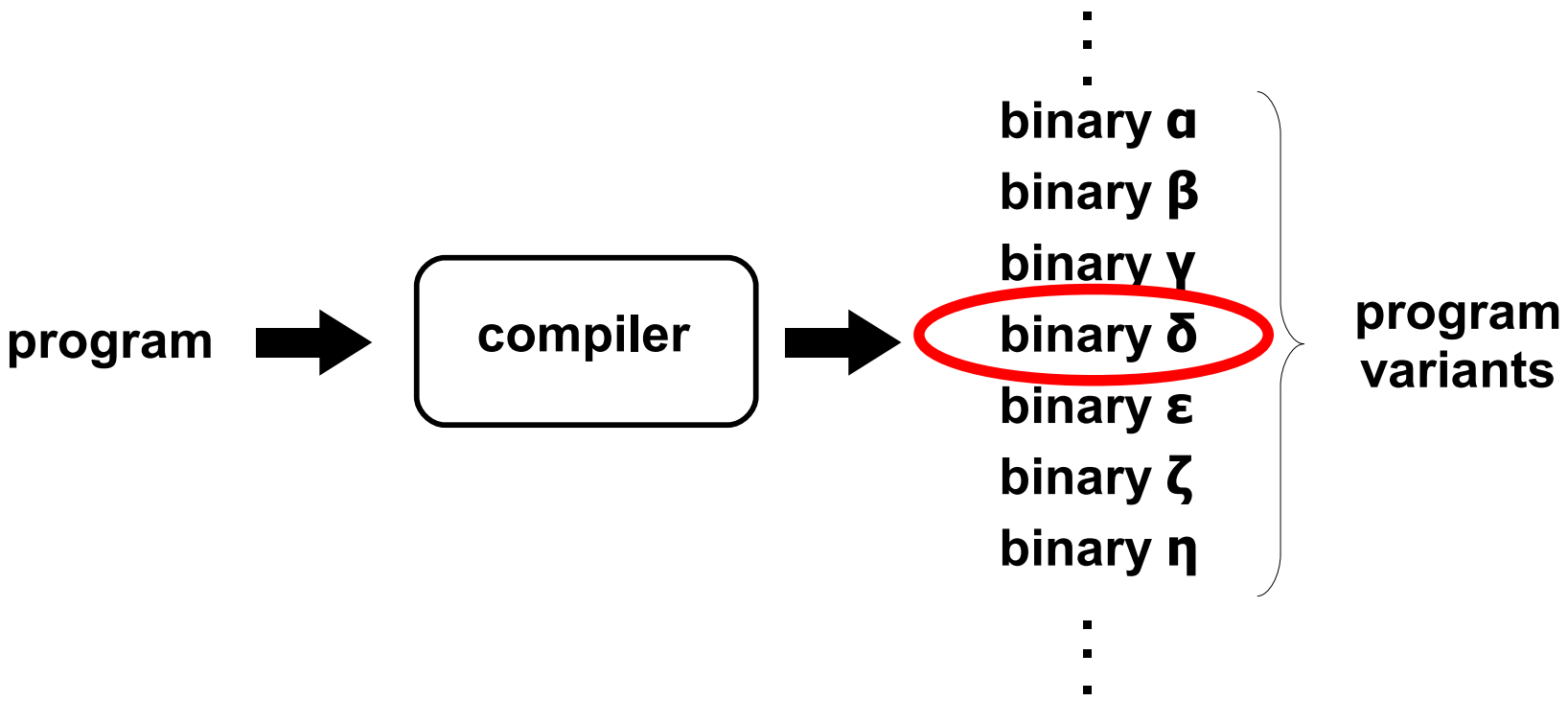
- Not all hardware details are published (e.g. trace cache)

Motivation:
Compilers use simplified models

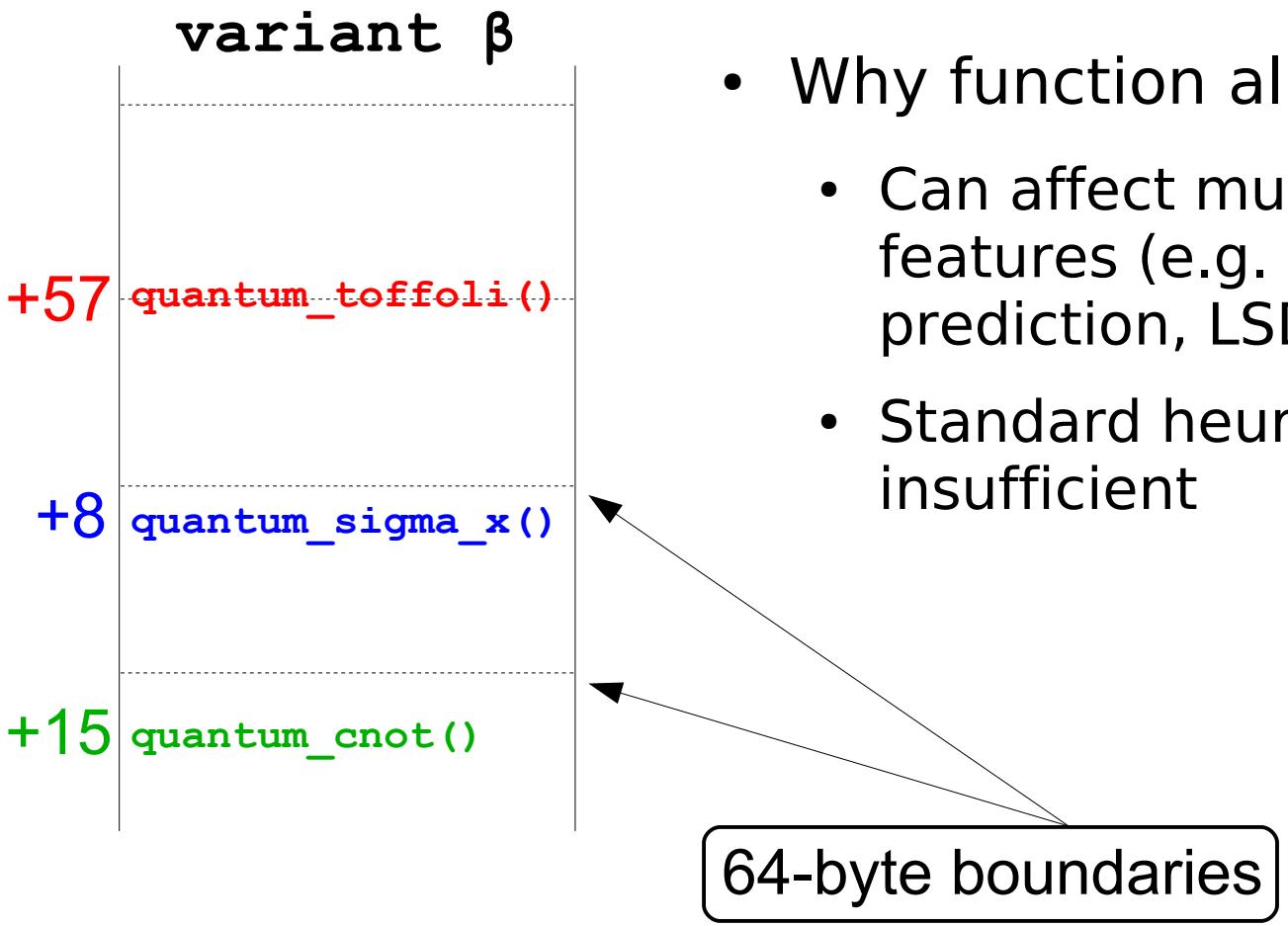


- Predictive heuristics optimize features independently
- Models may fail to capture interactions

The variant space

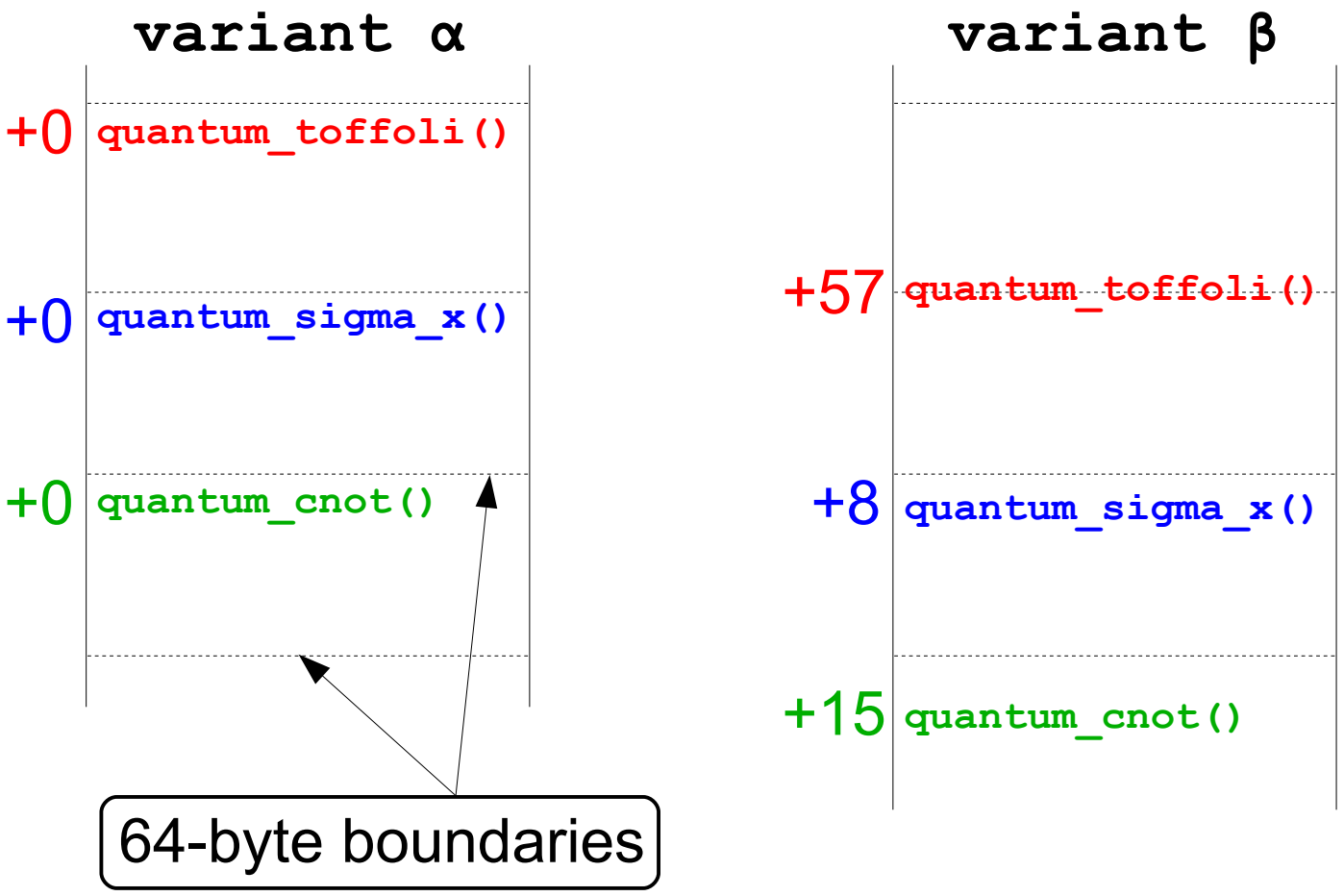


Our variant space: Function alignments

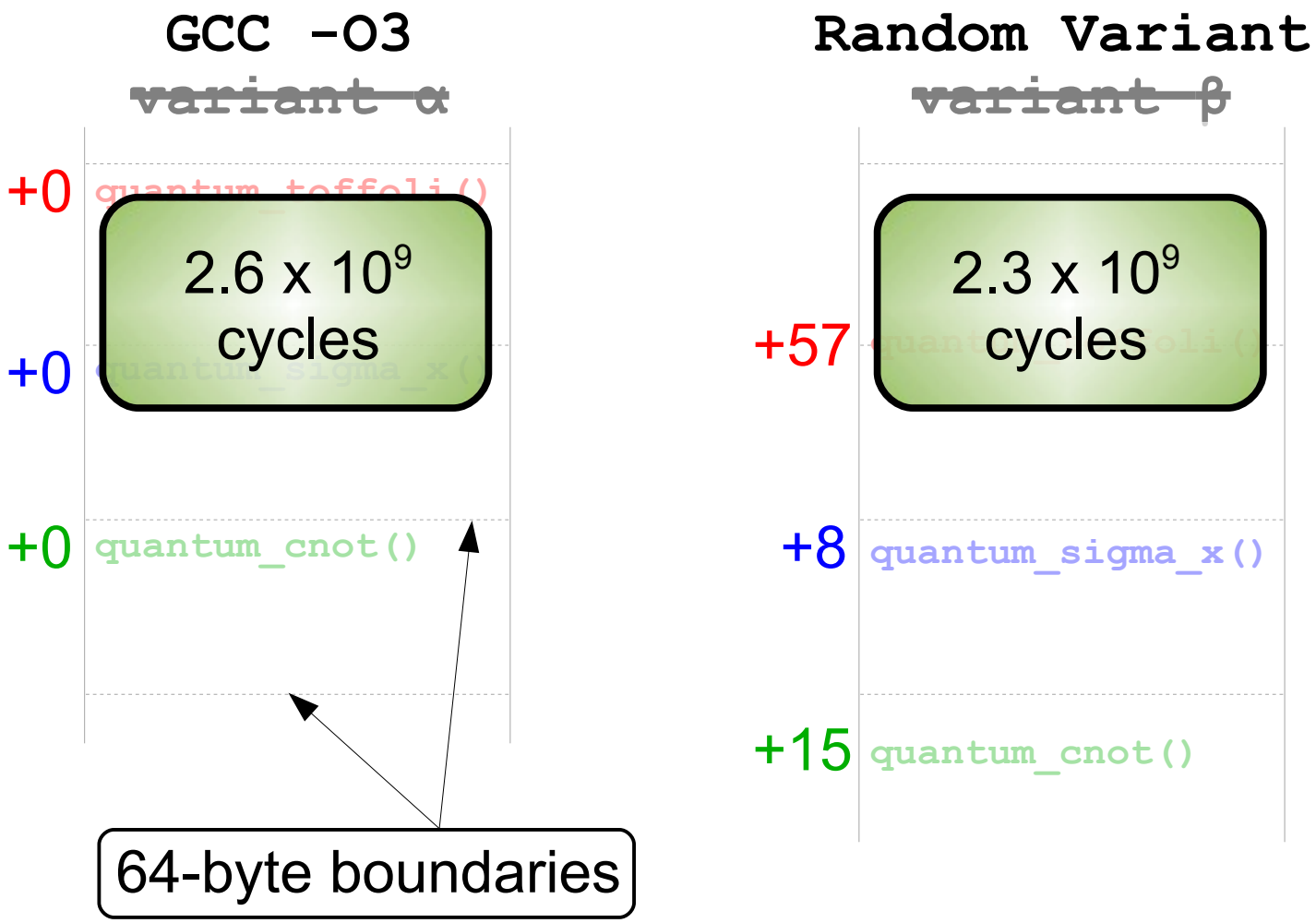


- Why function alignments?
  - Can affect multiple processor features (e.g. caches, branch prediction, LSD)
  - Standard heuristics are clearly insufficient

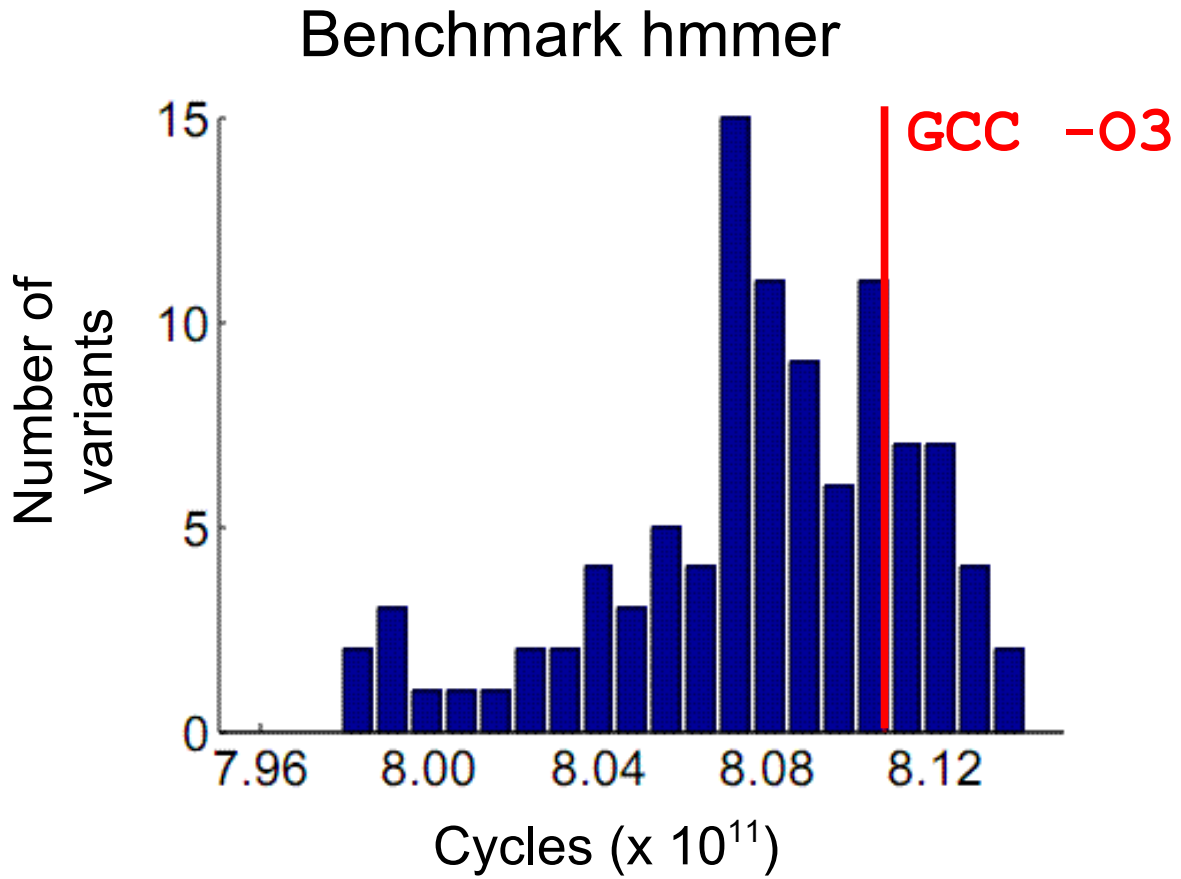
Challenge: which variant is fastest?



Challenge: which variant is fastest?

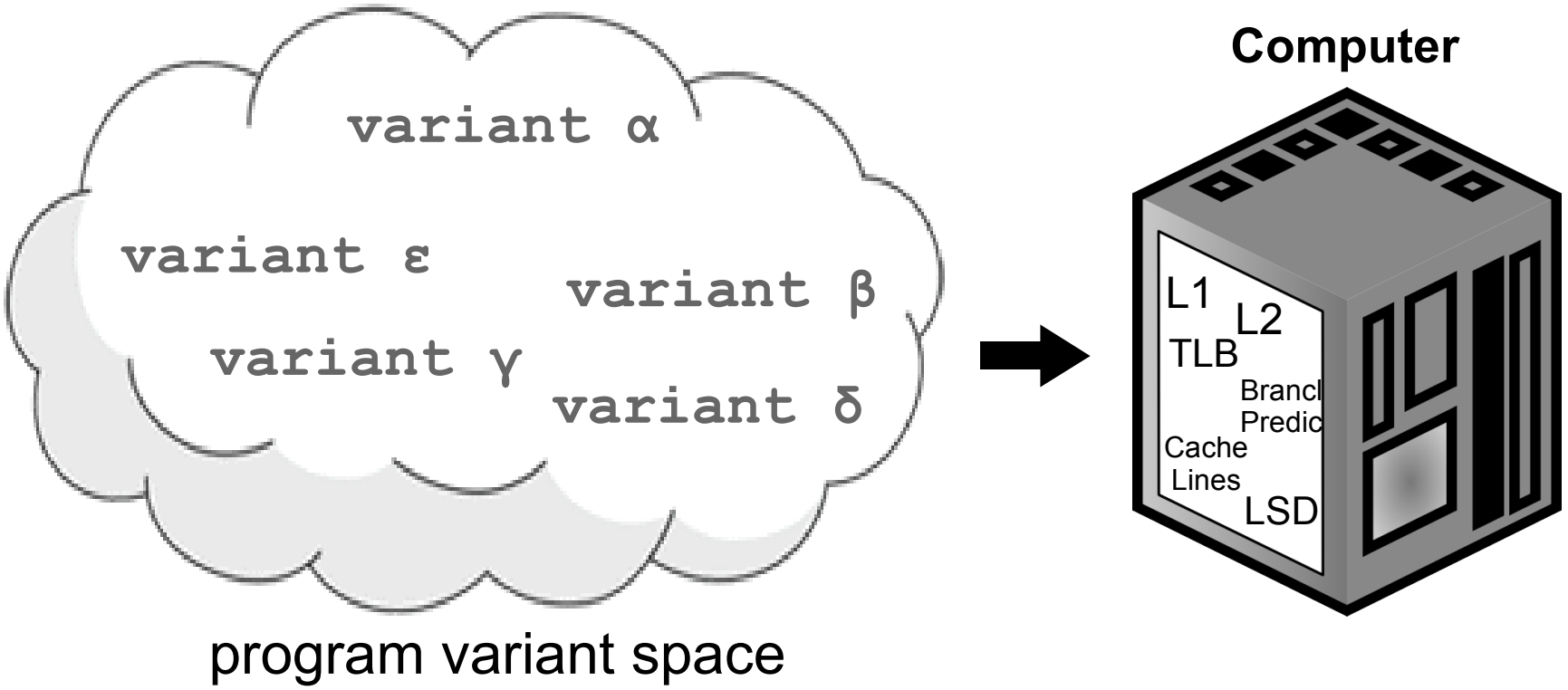


Results: distribution of run-times



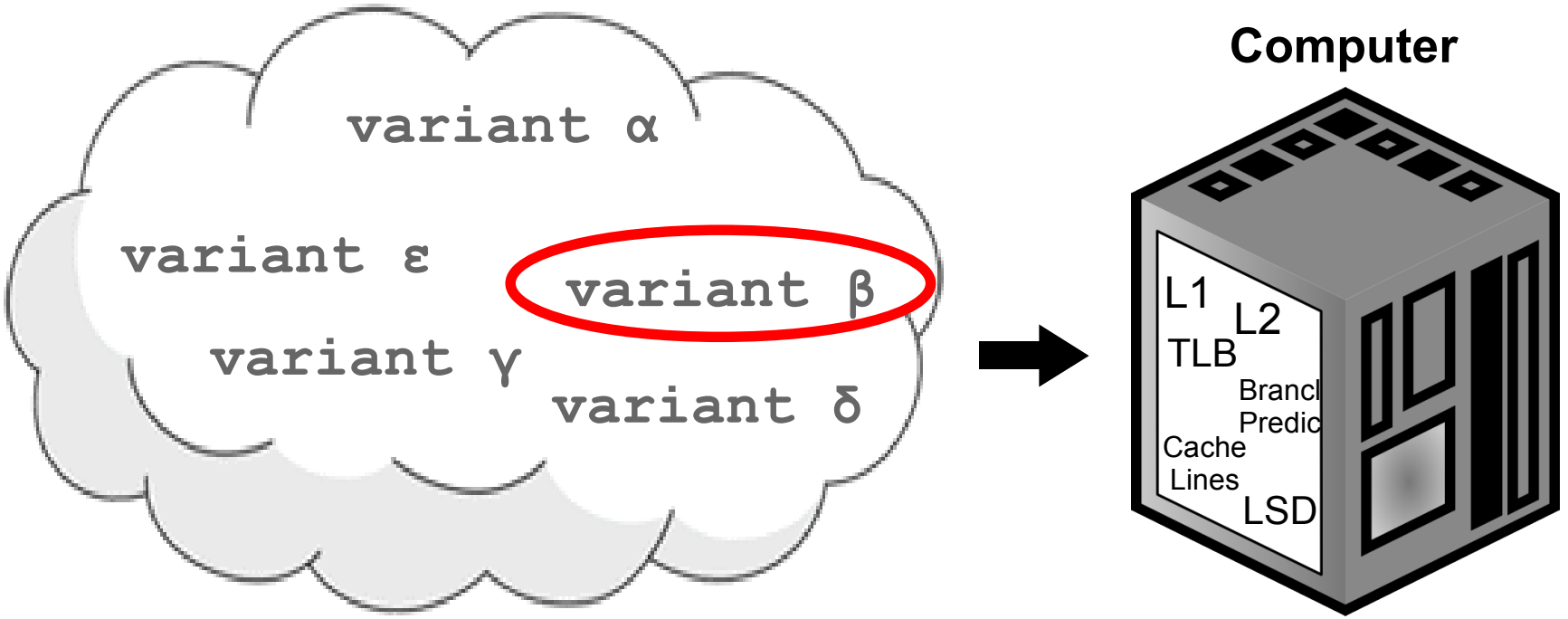
GCC on all inputs, all benchmarks: 57th percentile

The problem: a huge variant space



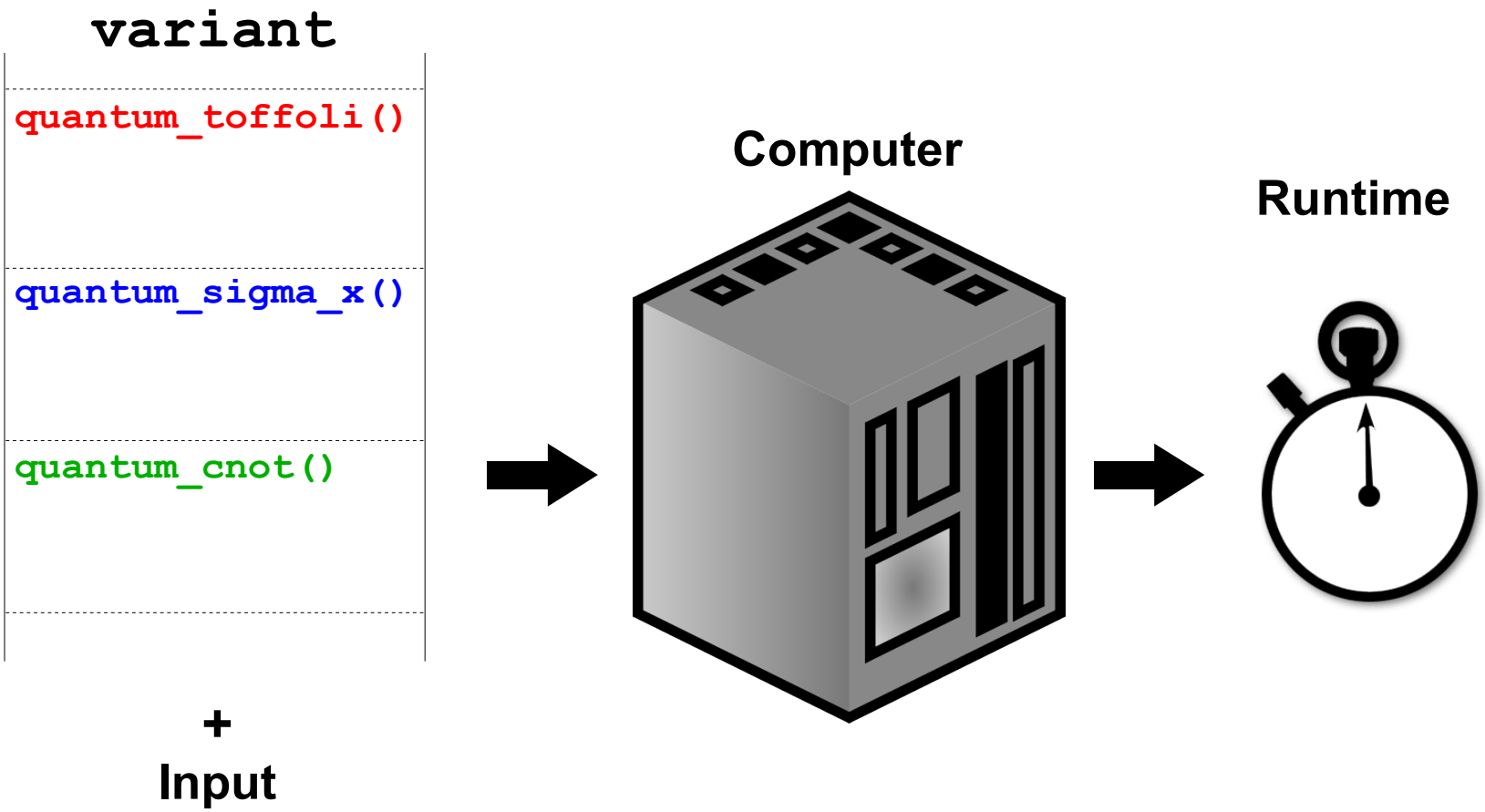
program variant space

The good news:
There is room for improvement

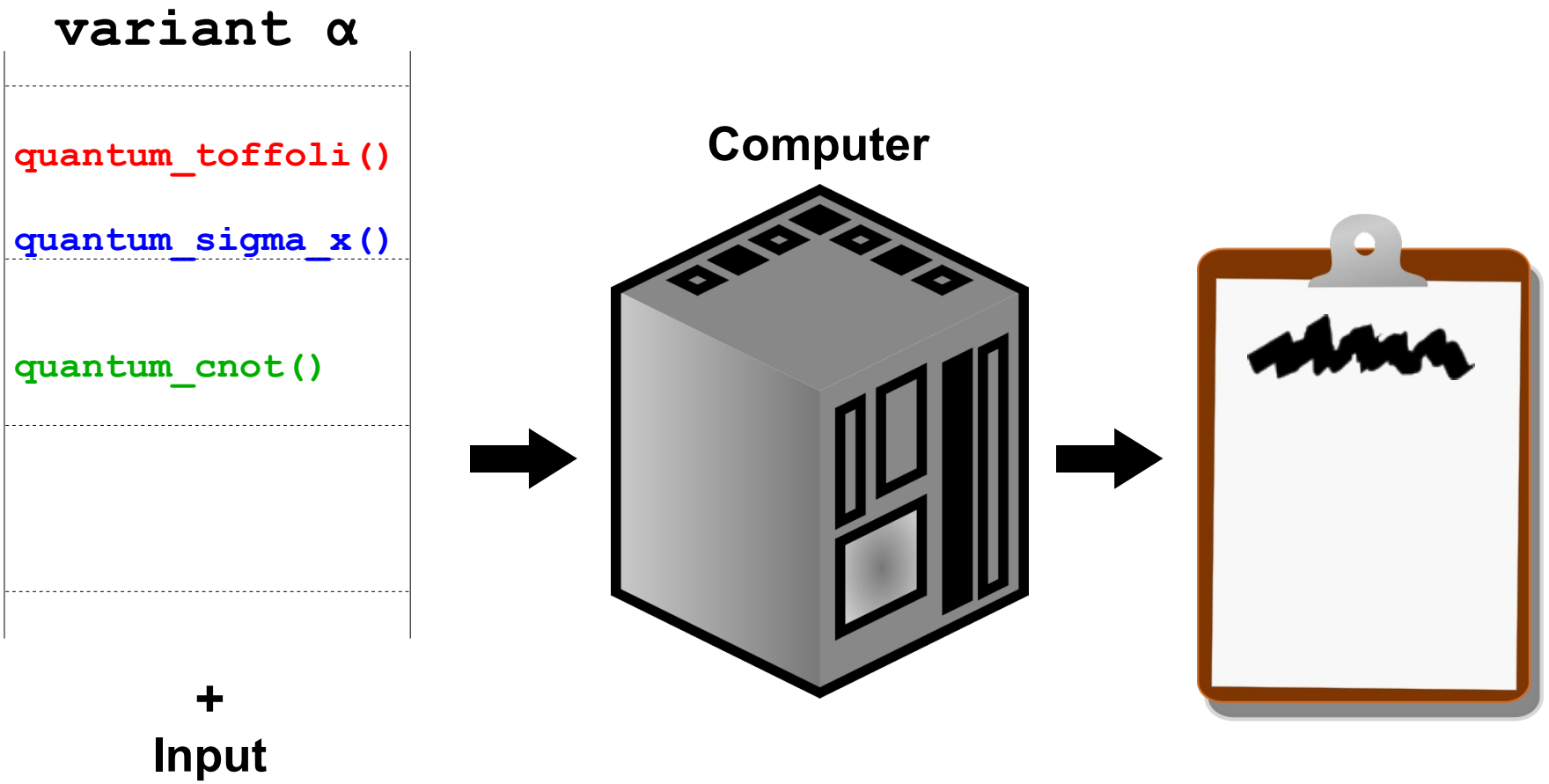


program variant space

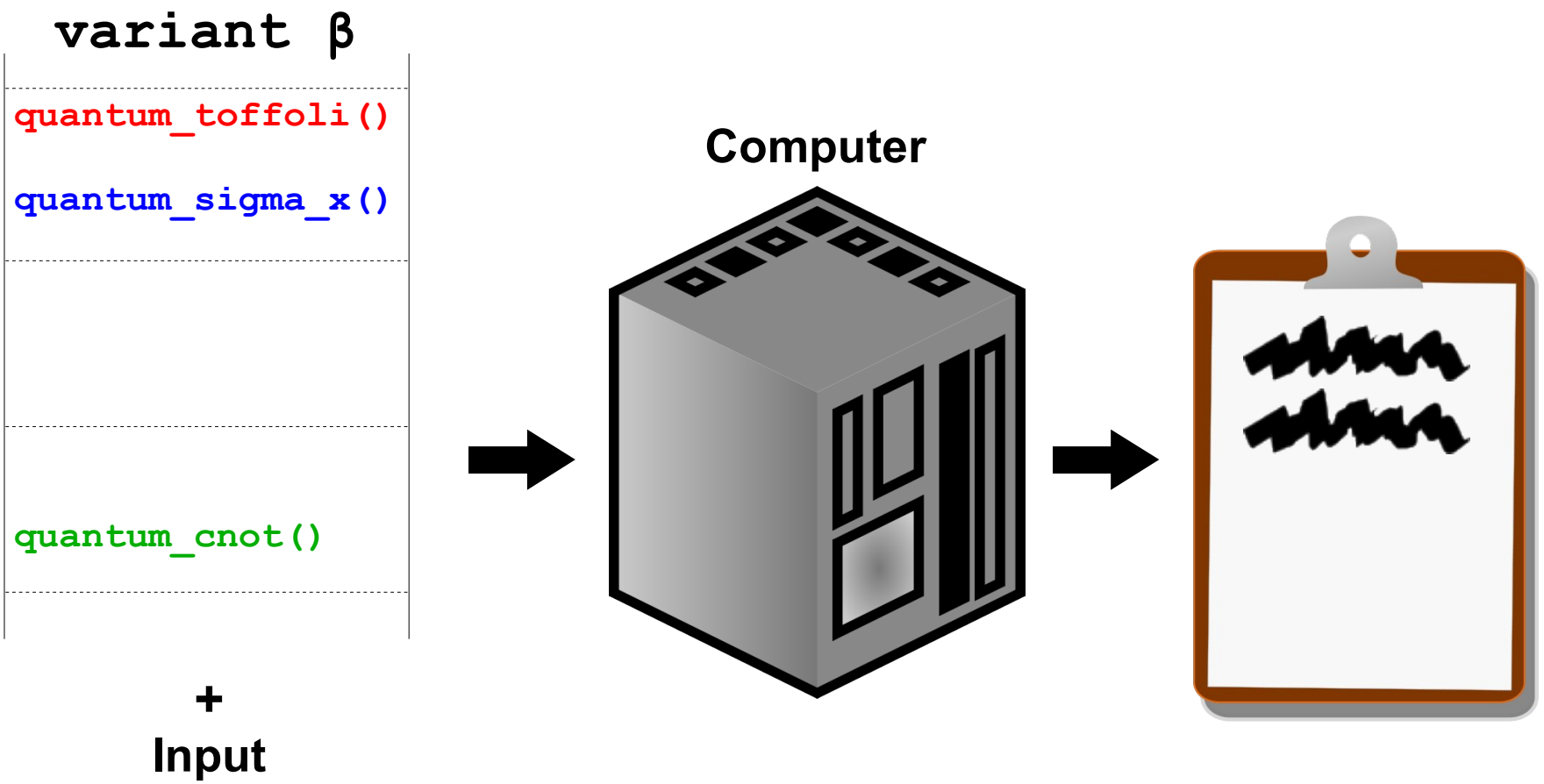
Blind Optimization:
Oblivious to hardware details



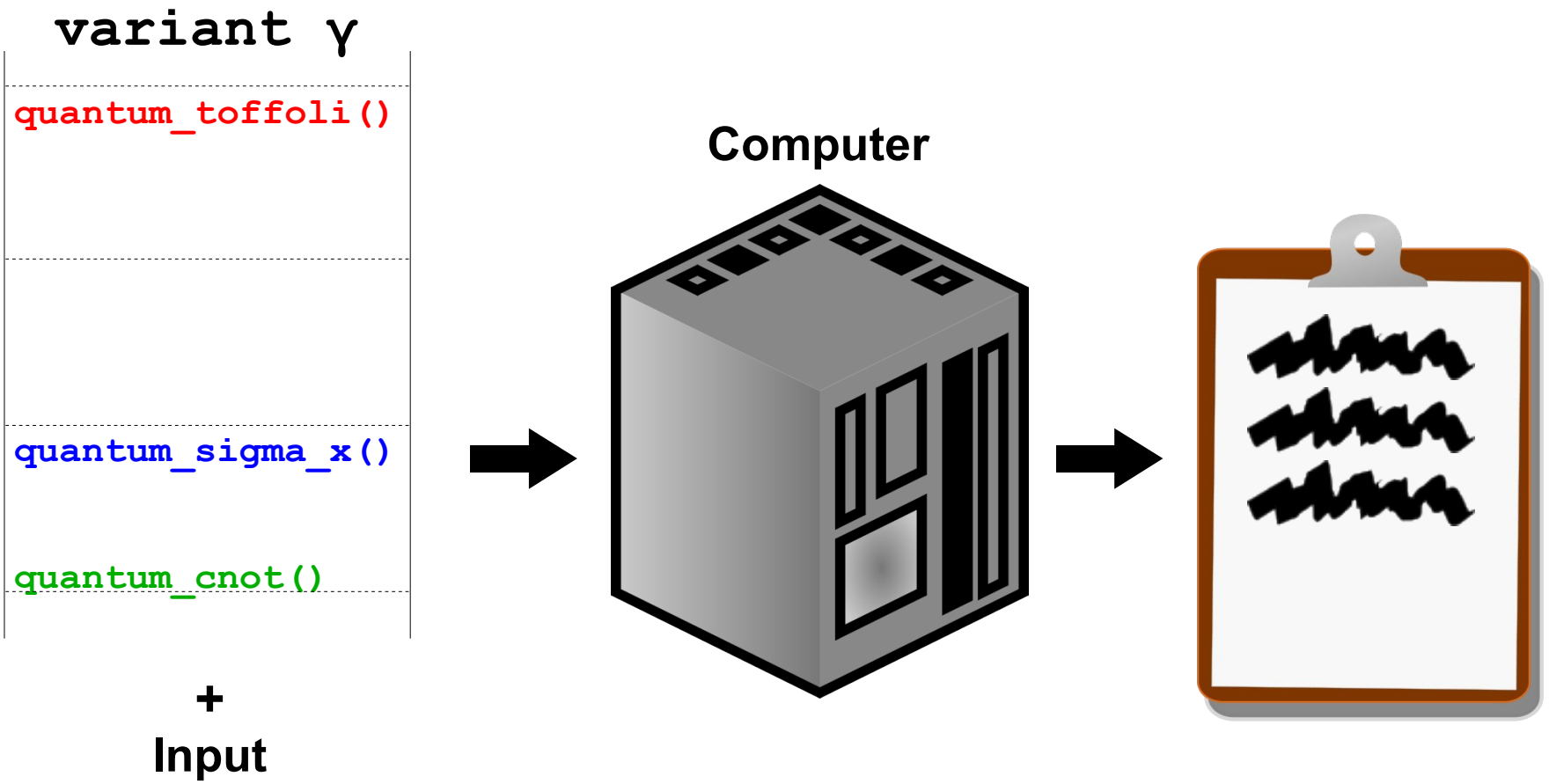
Approach:
Do a series of random experiments



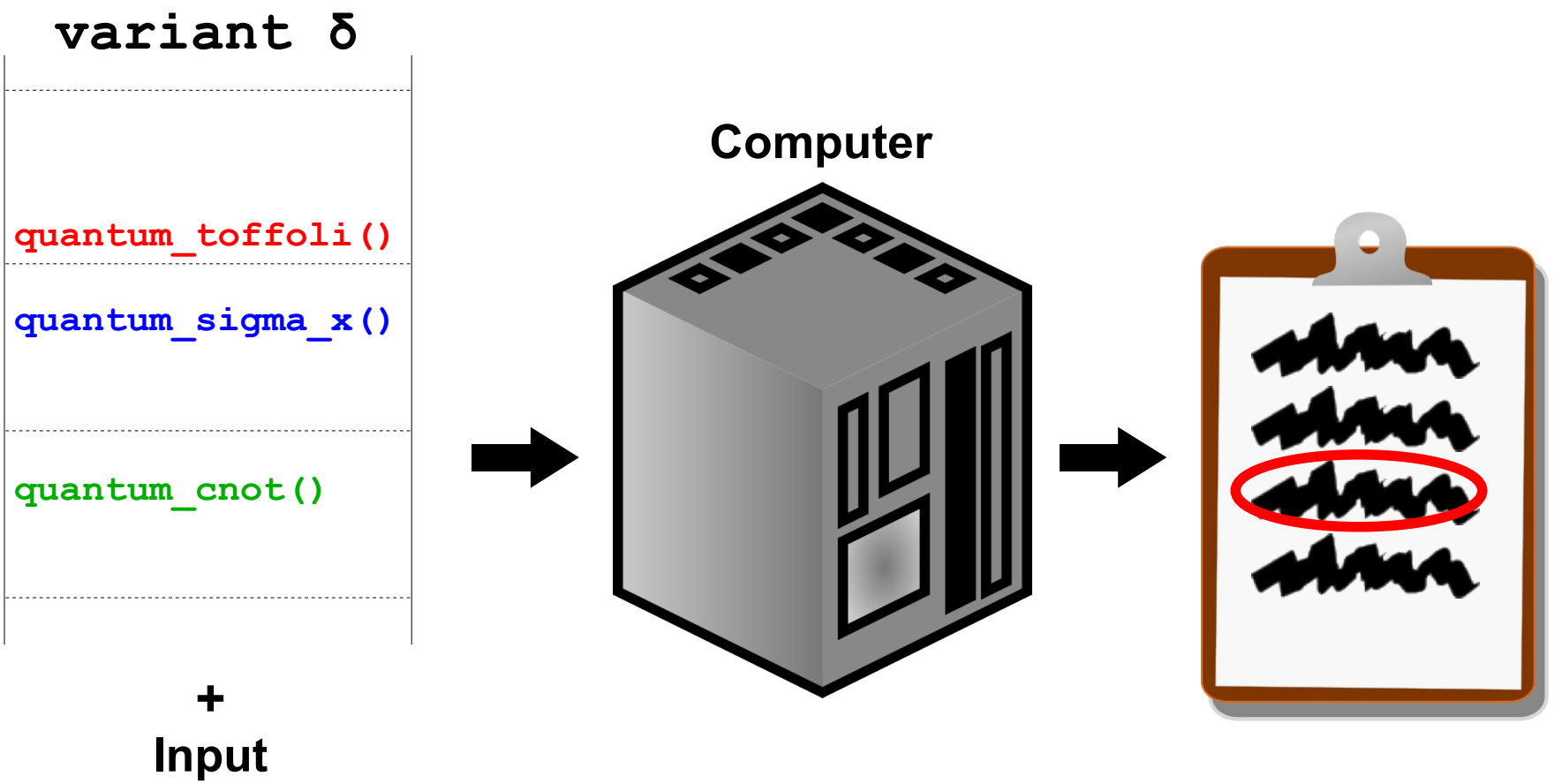
Approach:
Do a series of random experiments



Approach:
Do a series of random experiments



Approach:
Do a series of random experiments

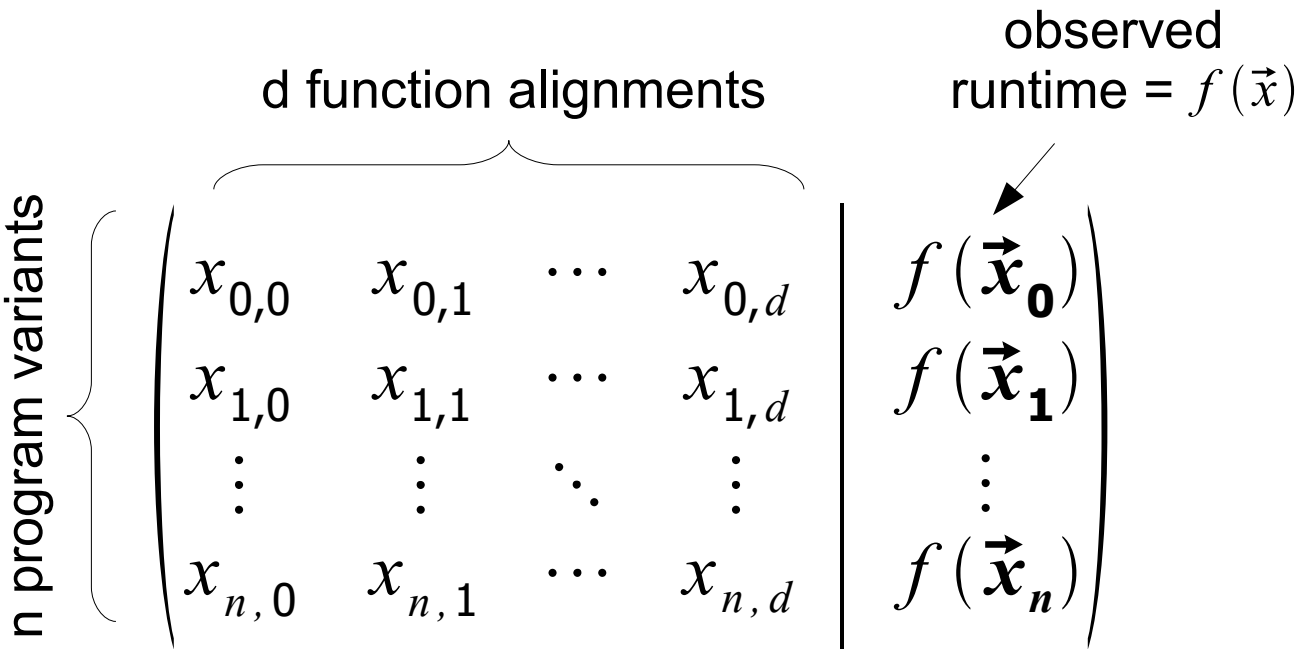


Approach: Setting up the problem

	quantum_toffoli	quantum_sigma_x	quantum_cnot	cycles x 10^9
	Alignment mod 64			
GCC -O3:	0,	0,	0,	?
variant α:	2,	31,	40,	?
variant β:	57,	8,	15,	?

Approach: Supervised Learning

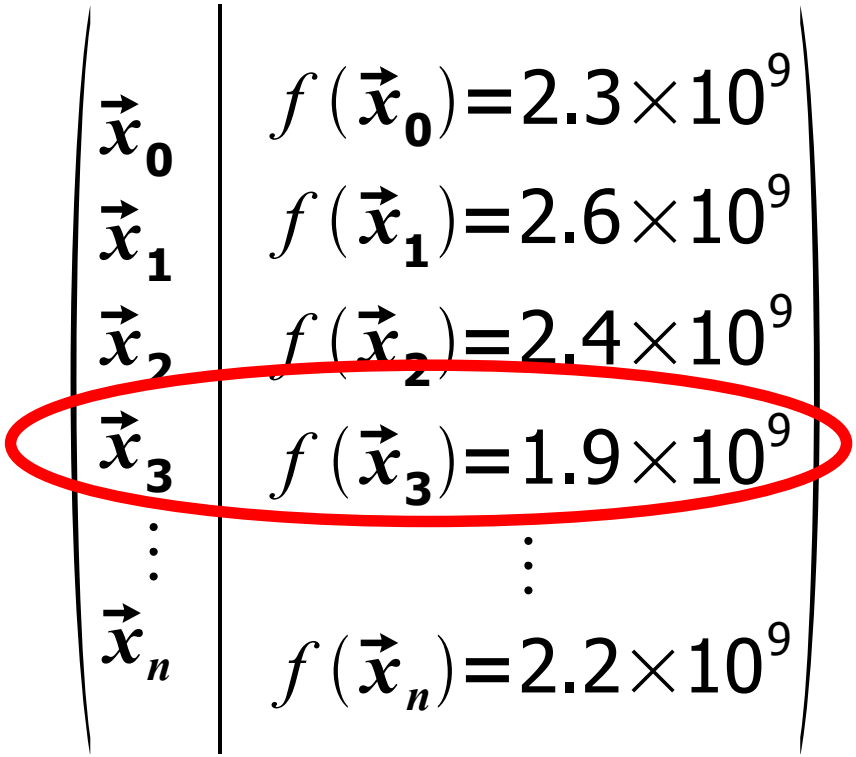
Given:



Find approximation of $\hat{f}(\vec{x})$, predict $y' = \hat{f}(\vec{x}')$ for a new $\vec{x}'$

Approach: Direct optimization

- Instead of finding $\hat{f}(\vec{x})$
- Find $\min_{\vec{x} \in X} f(\vec{x})$
  - random search
  - hill-climbing
  - genetic algorithms
  - simulated annealing
  - beam search
  - etc.



Methodology

- For each benchmark:
  1) Hold out one “test” input
  2) Time 100 random variants on the other inputs
  3) “Vote” for the best variant
  4) Time the winner on the “test” input
  5) Repeat for each input
- Report the average score (cross-validation)

Results

SPEC CPU2006 Benchmark**	Linear model (% speedup*)	Direct optimization (% speedup*)
libquantum	13.2%	13.4%
lbm	1.6	1.7
hmmer	1.5	1.4
bzip2	0.8	0.9
sjeng	-0.7	0.6
gobmk	-1.6	0.6
h264ref	0.3	0.4
mcf	-0.8	0.4
perlbench	0.0	0.3
milc	-1.6	0.1
gcc	-0.6	0.1
Average	1.1%	1.8%

*Over GCC -O3 ** Intel Xeon dual core / Core 2 Duo

Results

SPEC CPU2006 Benchmark**	Linear model (% speedup*)	Direct optimization (% speedup*)
libquantum	13.2%	13.4%
lbm	1.6	1.7
hmmer	1.5	1.4
bzip2	0.8	0.9
sjeng	-0.7	0.6
gobmk	-1.6	0.6
h264ref	0.3	0.4
mcf	-0.8	0.4
perlbench	0.0	0.3
milc	-1.6	0.1
gcc	-0.6	0.1
Average	1.1%	1.8%

← **4.6% ICC**

*Over GCC -O3 ** Intel Xeon dual core / Core 2 Duo

More good news:
We can run more experiments offline

11 PM: Go to bed.

2 AM: Defrag hard-drive.

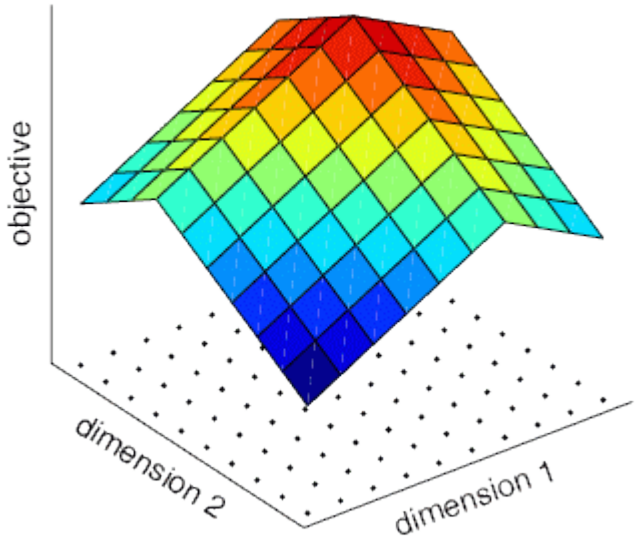
3 AM: Reoptimize software.

Checkpoint

- Blind optimizations can work
  - even a simple optimization, minimal search
  - up to 13% speedup
- Bad news: big program variant space
- Good news: potential for improvement
- What about other dimensions?
  - function alignments vs. inverting branch directions

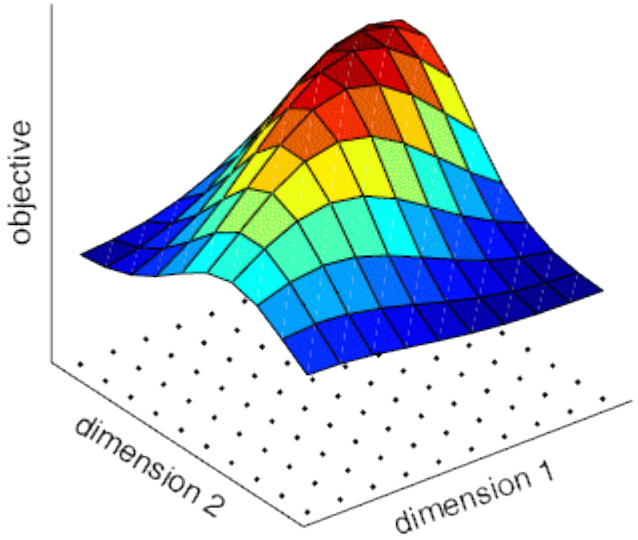
Going further: Possible objective functions

GOOD



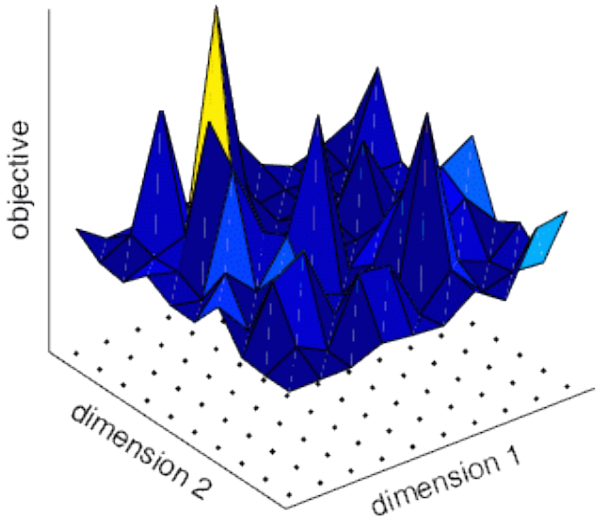
No dependencies,
Structured

BAD



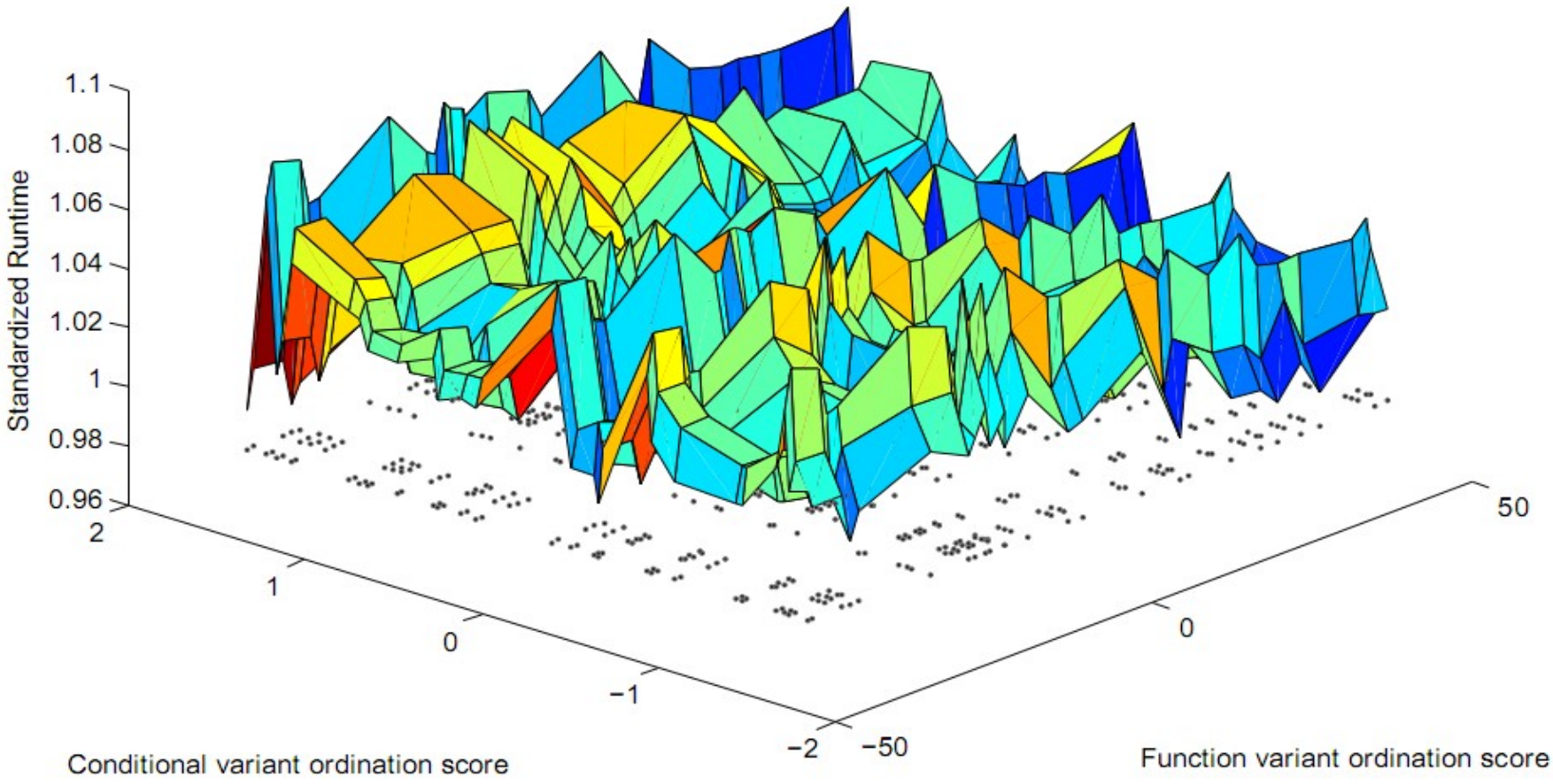
Dependencies,
Structured

WORSE

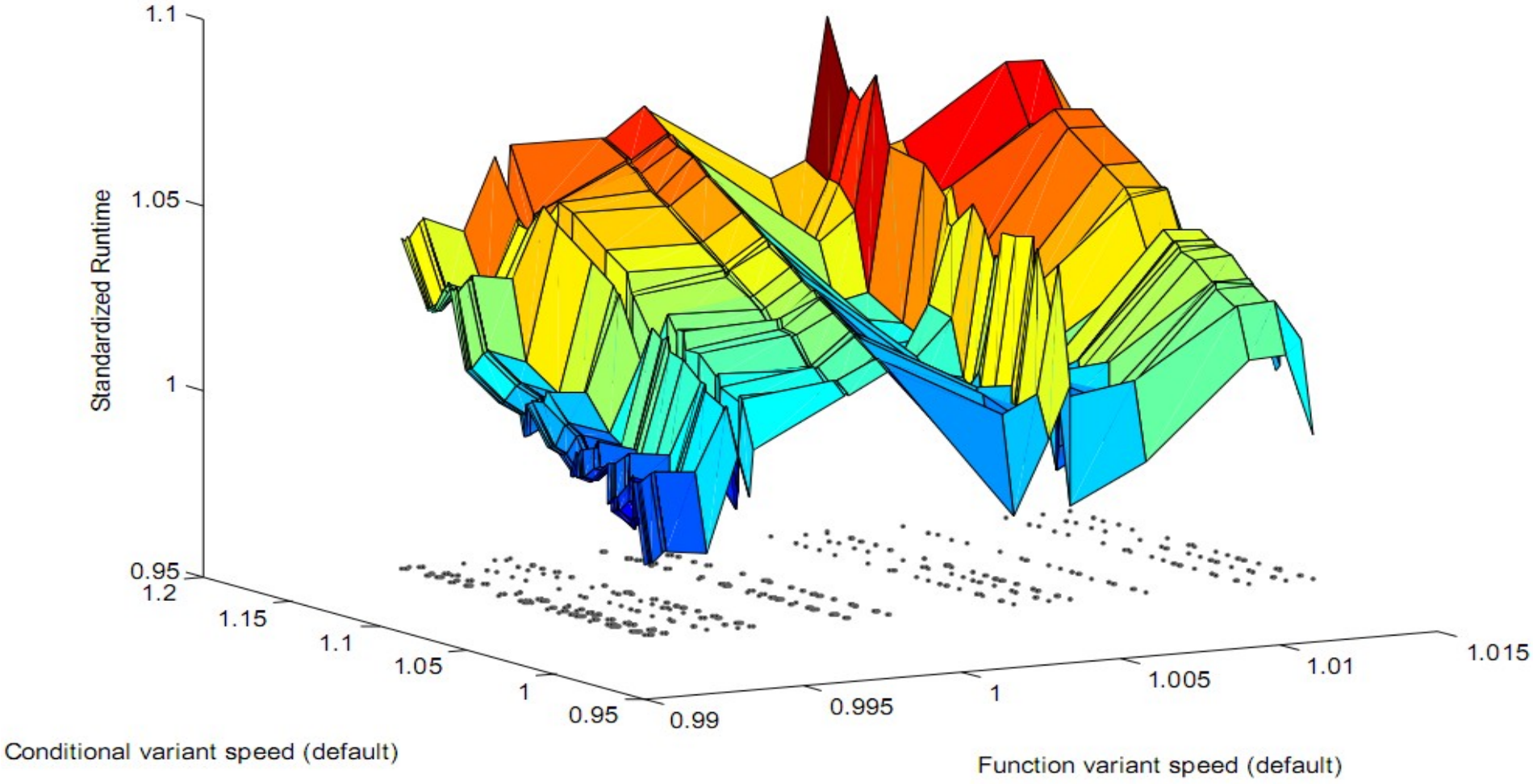


Dependencies,
No structure

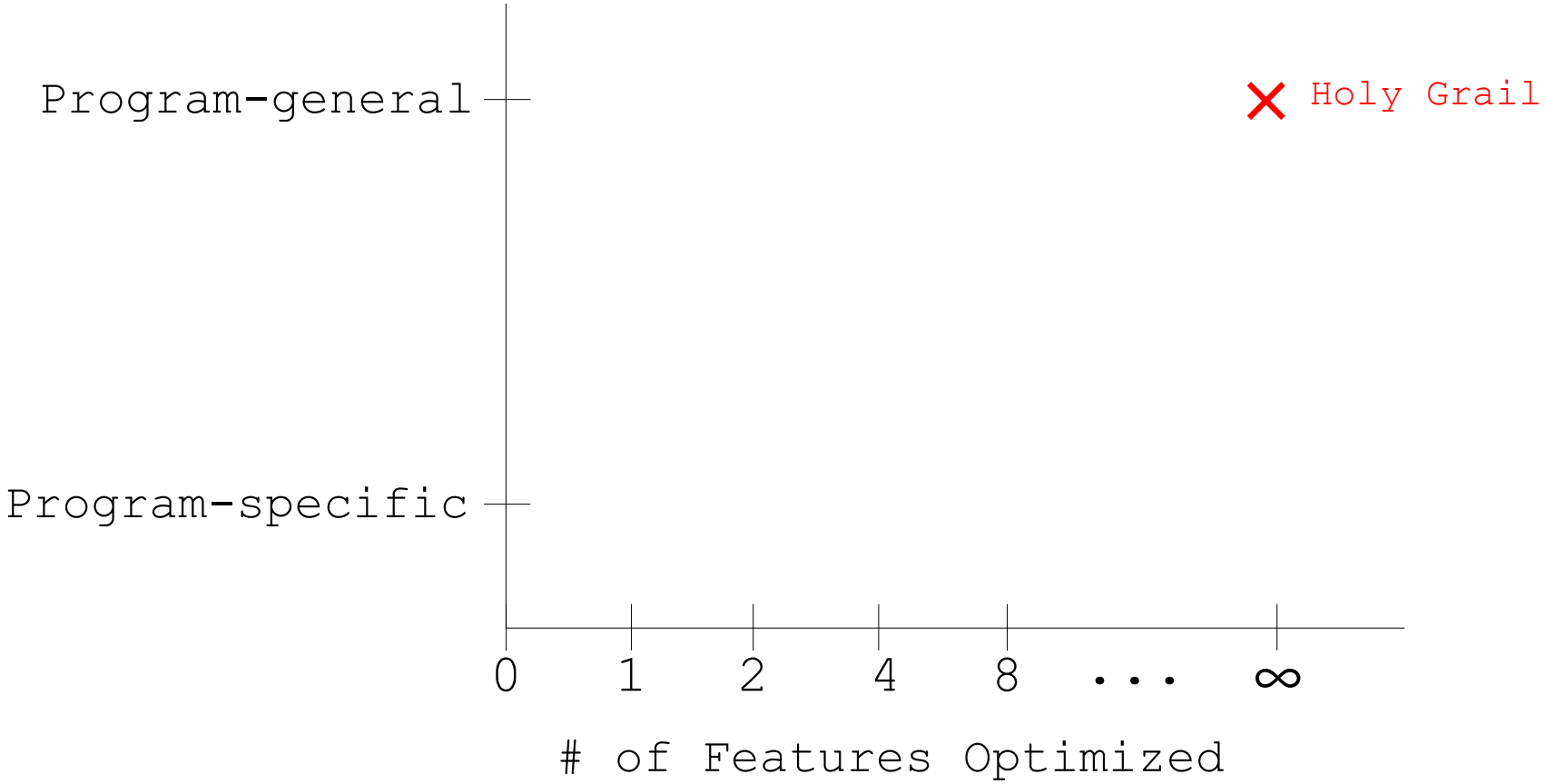
Combining optimizations: Sorted by NMDS and Manhattan distance



Combining optimizations: Sorted by “default” speedup

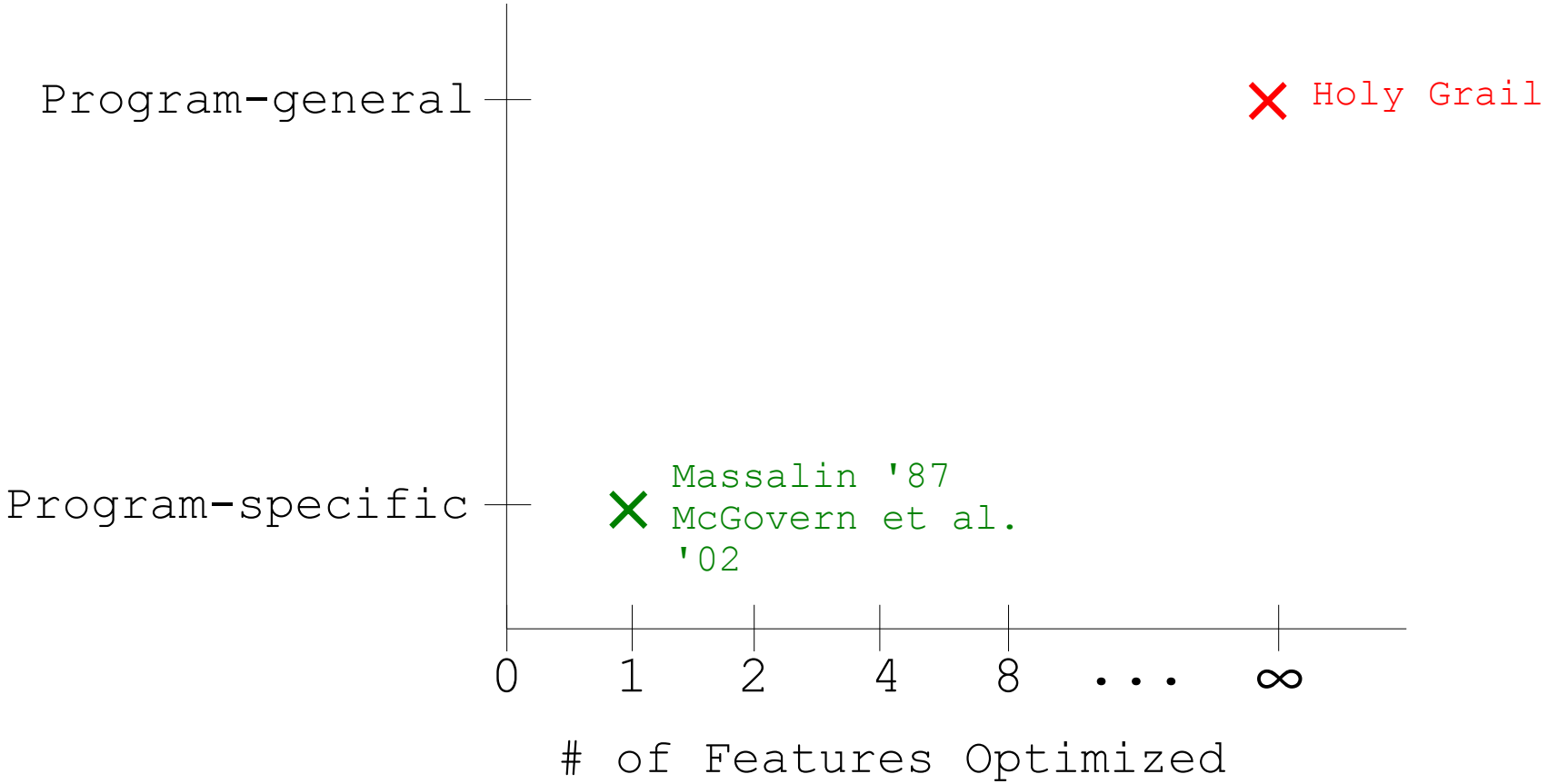


Prior work: search-based optimizations



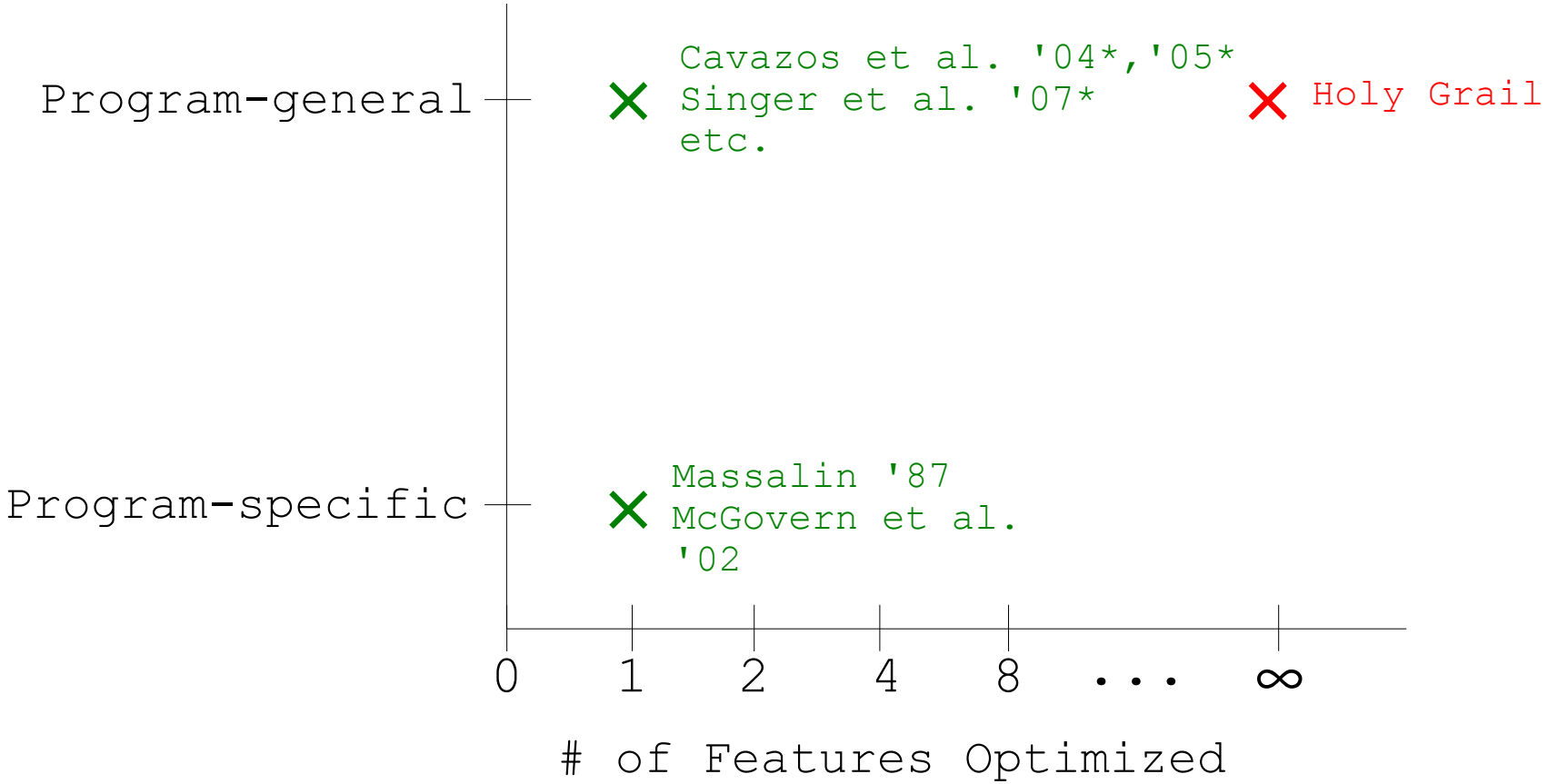
* Uses heuristics

Prior work: search-based optimizations



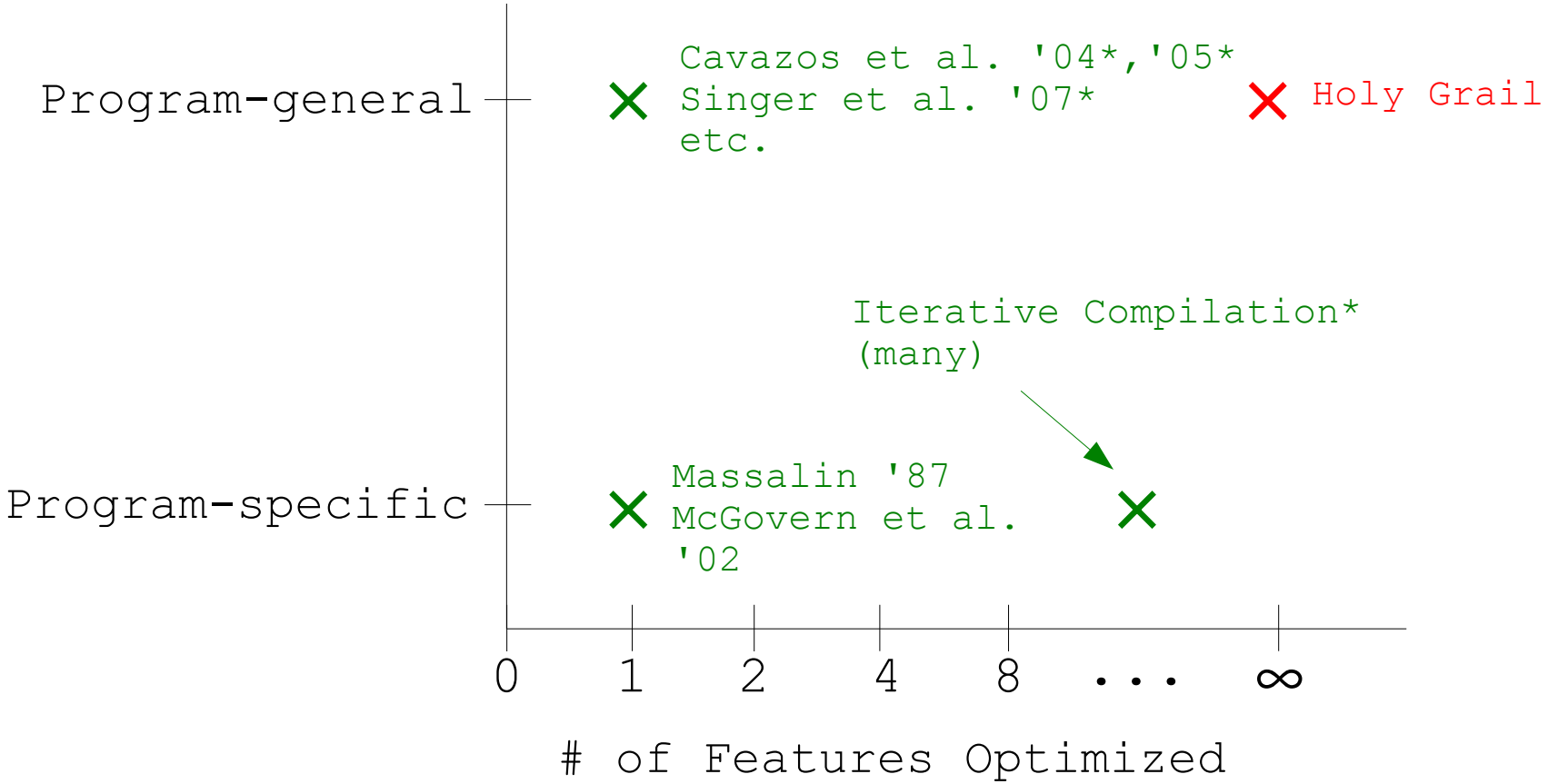
* Uses heuristics

Prior work: search-based optimizations



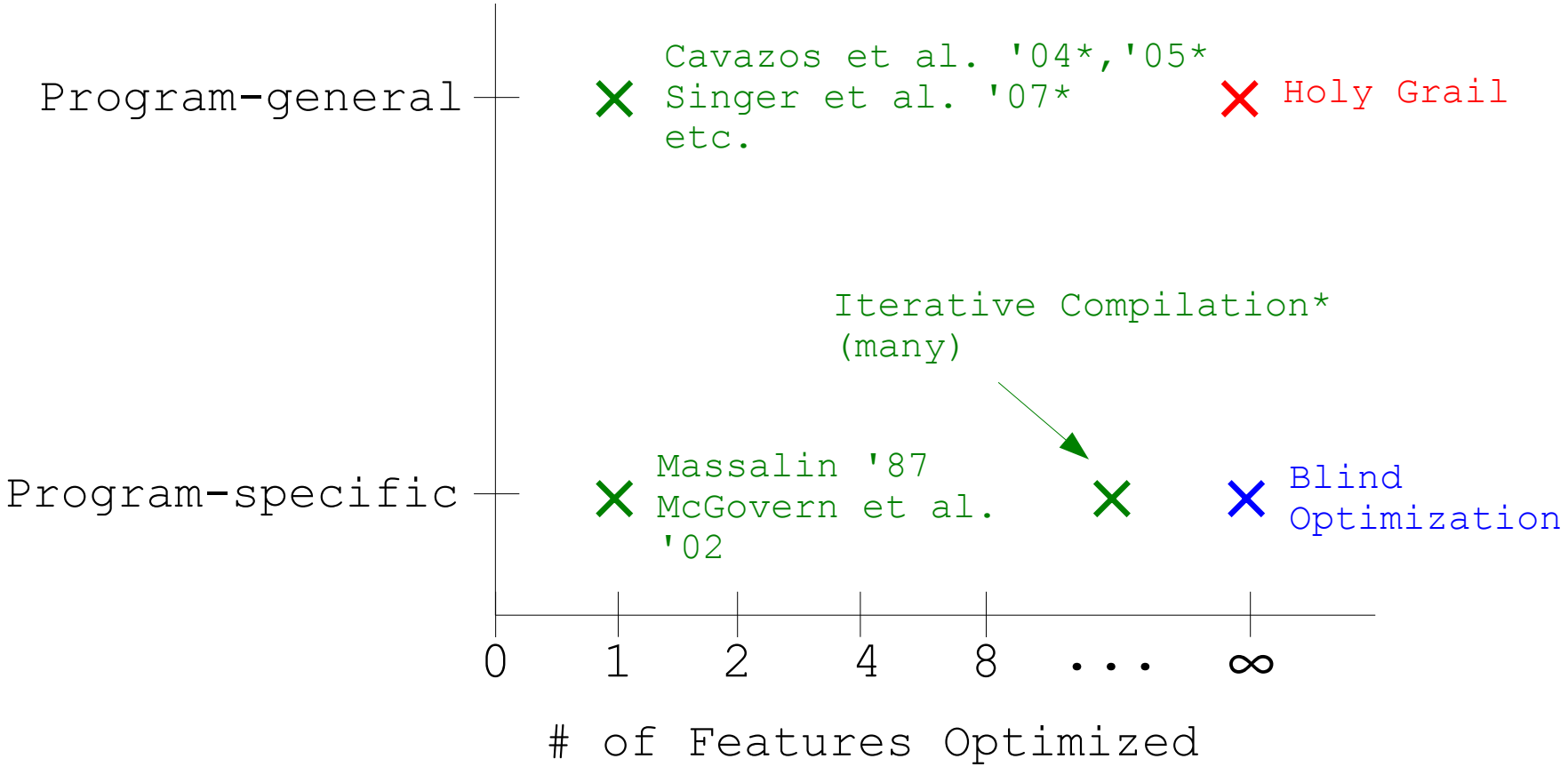
* Uses heuristics

Prior work: search-based optimizations



* Uses heuristics

Prior work: search-based optimizations



* Uses heuristics

Summary

- Blind optimization optimizes directly, without heuristics or static models
  - The search space is huge, but there is potential for improvement
  - There are highly complex interactions between optimizations
- Ongoing work:
  - Exploring other dimensions
  - Improving search