# Operator Variant Selection on Heterogeneous Hardware

Viktor Rosenfeld, Max Heimel, Christoph Viebig, Volker Markl

ADMS '15 in conjunction with VLDB '41
Kohala Coast, Hawai'i – August 31, 2015

1

Challenge

High performance or small implementations

2

Experimental demonstration

Lack of performance portability in OpenCL

3

Solution sketch

Performance-portable database operators

4

Current work

Learning fast operator implementations

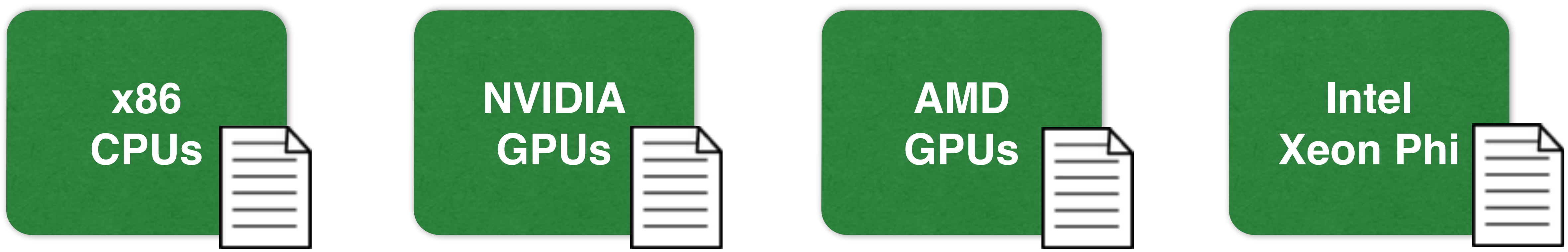
1

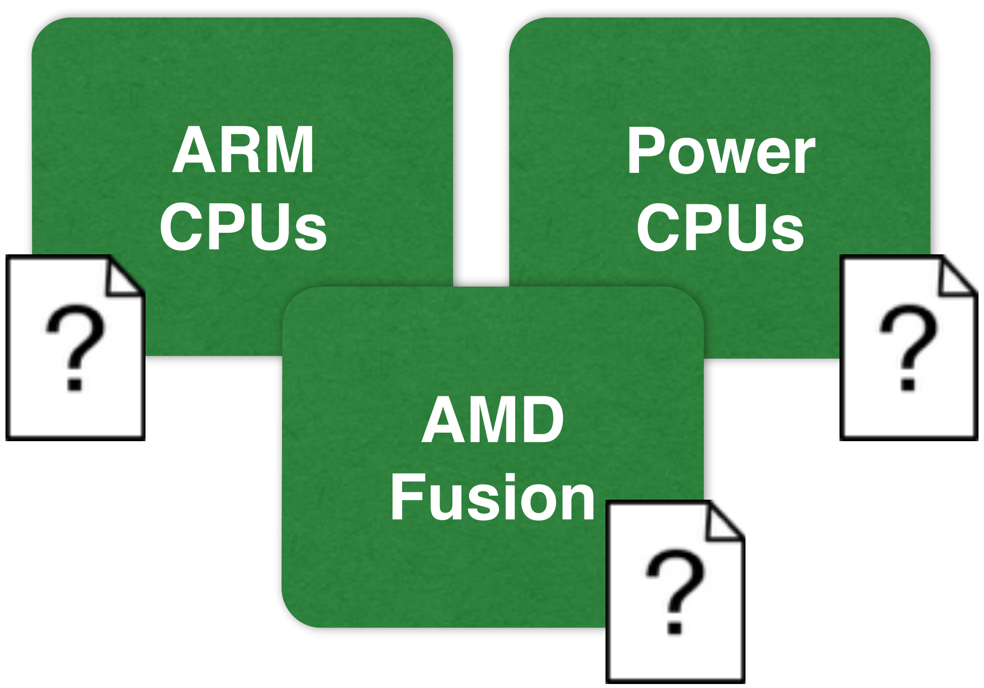
Challenge

# High performance or small implementations

# Hardware-sensitive Approach

dedicated operator implementations for each device



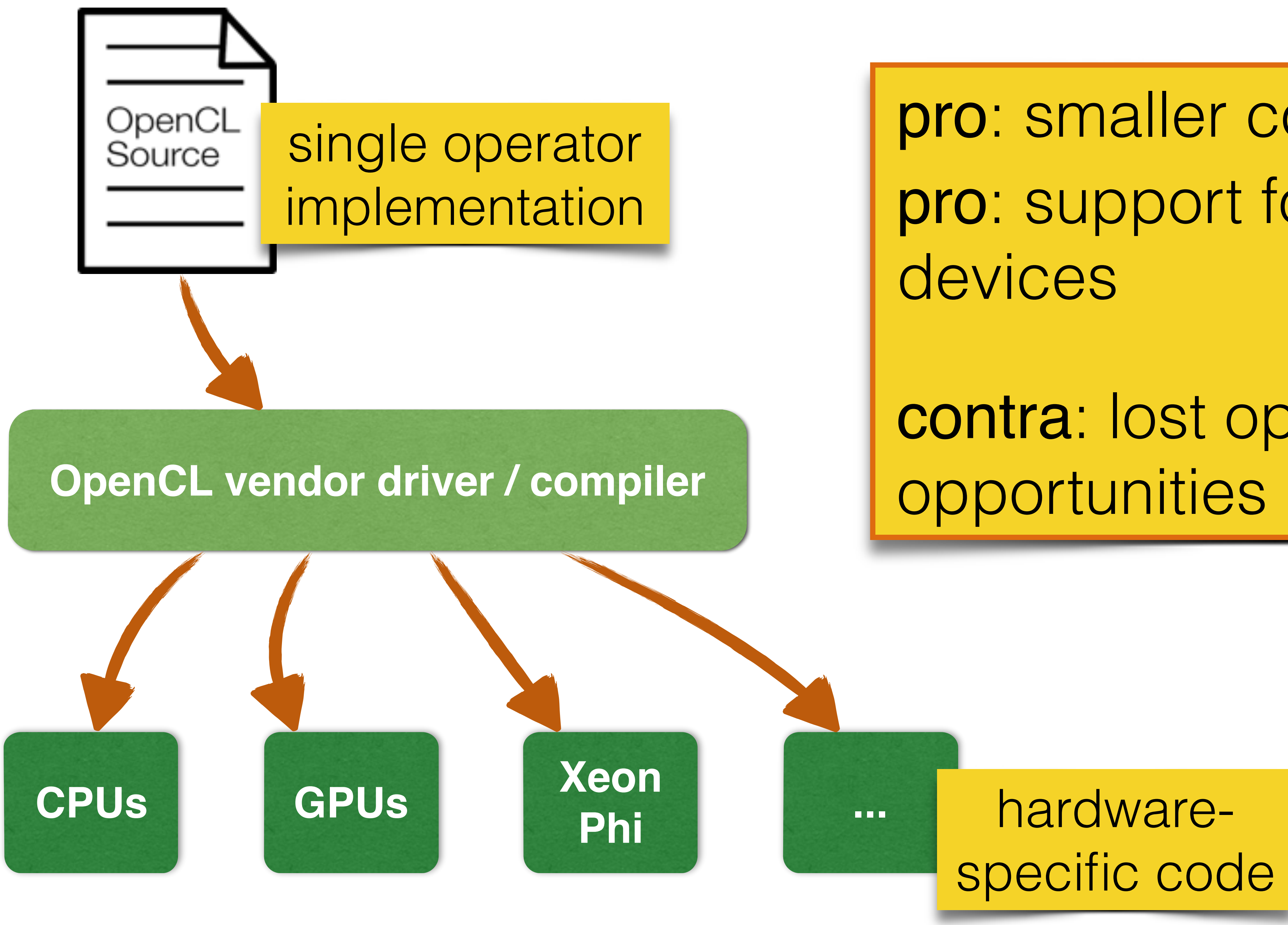




**pro**: optimal implementations for each device

**contra**: development and maintenance overhead

# Hardware-oblivious Approach

support multiple devices from a single implementation




**pro**: smaller code base

**pro**: support for unknown devices

**contra**: lost optimization opportunities

# OpenCL Portability

- OpenCL offers *functional portability*
- But not *performance portability*
- Many parameters to tweak: thread workload, memory access, special functions, ...
- Hardware-specific OpenCL implementations?

lack of performance portability
limits the value of functional portability

2

Experimental demonstration

# Lack of performance portability in OpenCL

# Selection Kernels

**Basic algorithm**

- scan over column
- evaluate predicate for each value, $x < const$
- return bitmap indicating satisfying values

# Variant Dimensions

Code modifications ~ 60 variants

- Basic algorithm (memory access & result bitmap construction): sequential, atomic-global, atomic-local, reduce, collect, transpose
- Result bitmap granularity: 8 bit, 16 bit, 32 bit, 64 bit
- Loop unrolling: yes, no
- Predication: yes, no

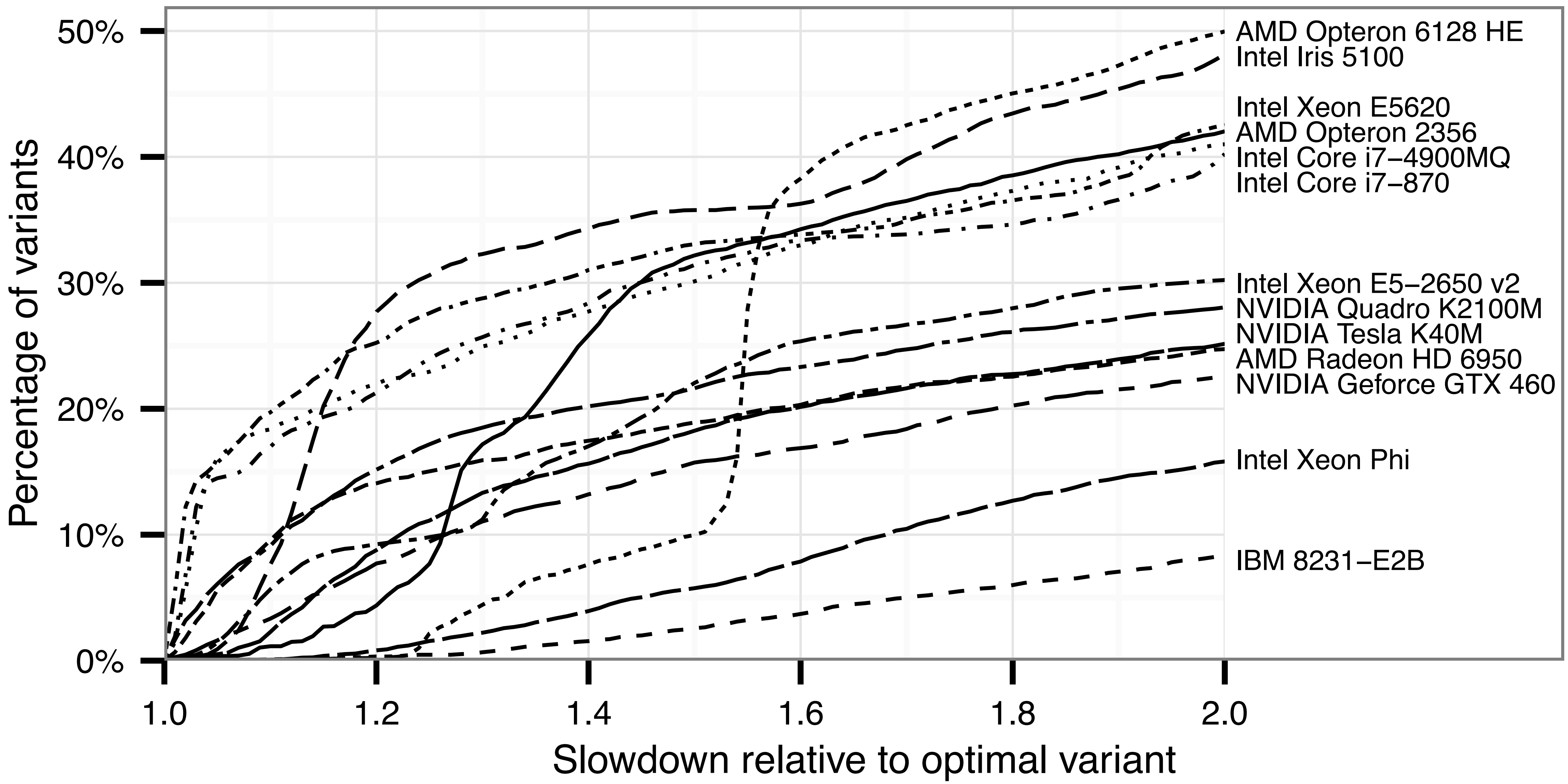
Workload parameters

- Local size: 1, 2, 4, 8, ..., max
- Elements per thread: 1, 2, 4, 8, ..., 1024

~5000 of selection kernel variants

# Competitive Variants

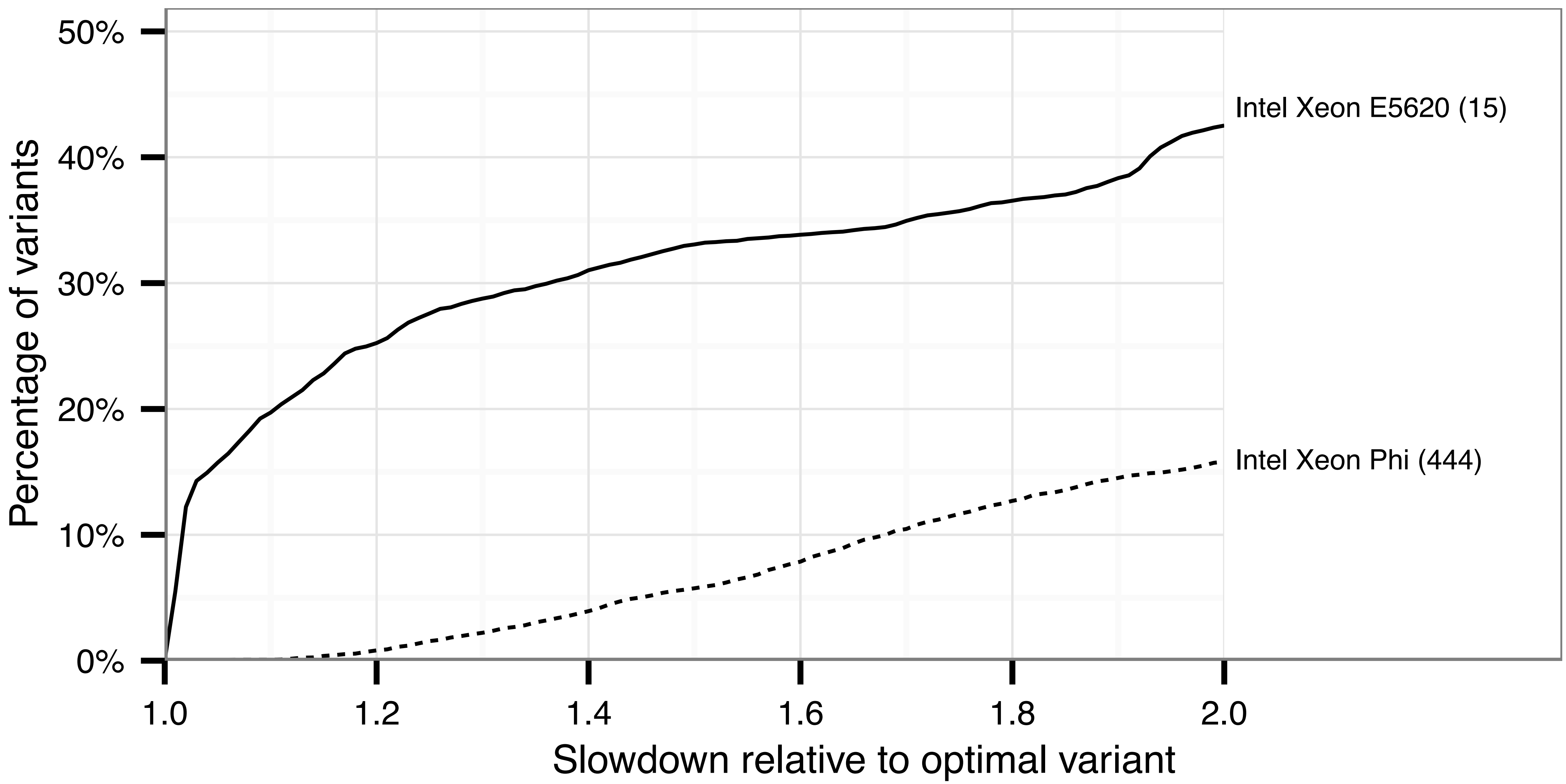
percentage of variants that are at most 2x slower than fastest variant for each device




often many variants are competitive

# Competitive Variants

percentage of variants that are at most 2x slower than fastest variant for each device




"easy" and "difficult" devices

3

Solution sketch

# Performance-portable database operators

# Automatic Variant Tuning

1. specify operators in generic fashion ?
2. derive different implementations ?
3. learn best implementation per device ✓

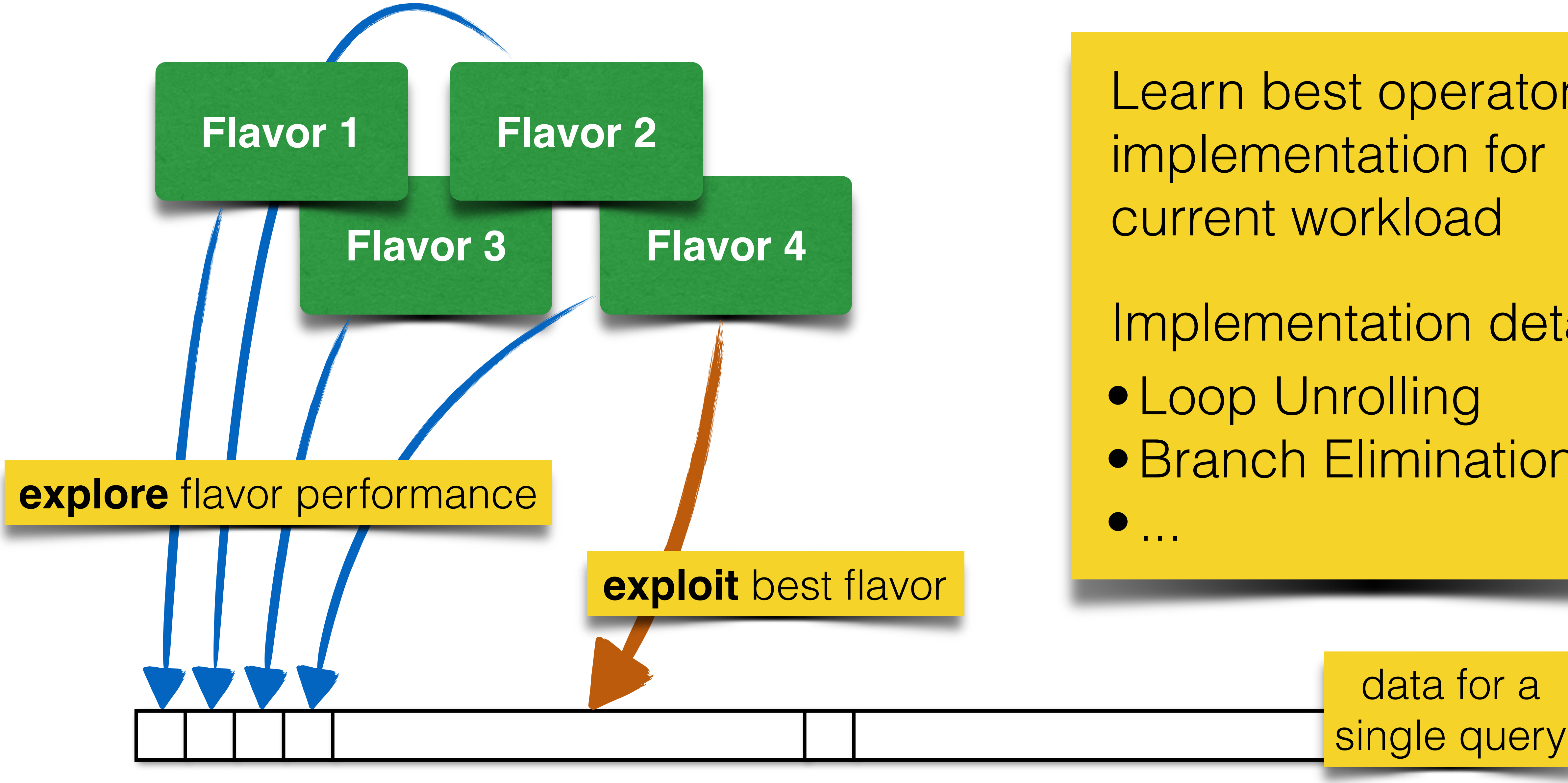
**let the system generate and find the best variant**

4

Current work

# Learning fast operator implementations

# Micro Adaptivity




Learn best operator implementation for current workload

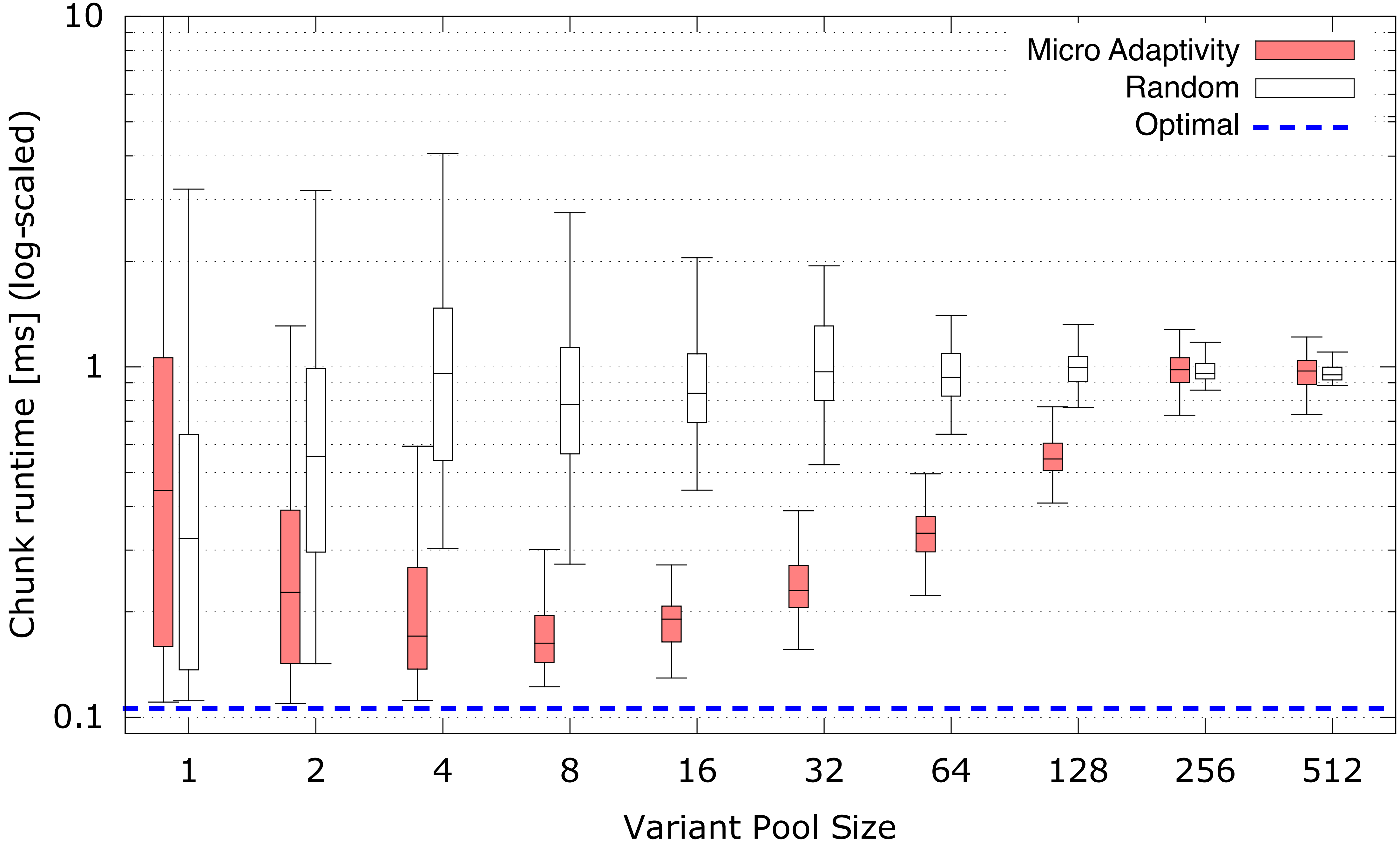
Implementation details

- Loop Unrolling
- Branch Elimination
- ...

no optimal operator implementation, even for a single query

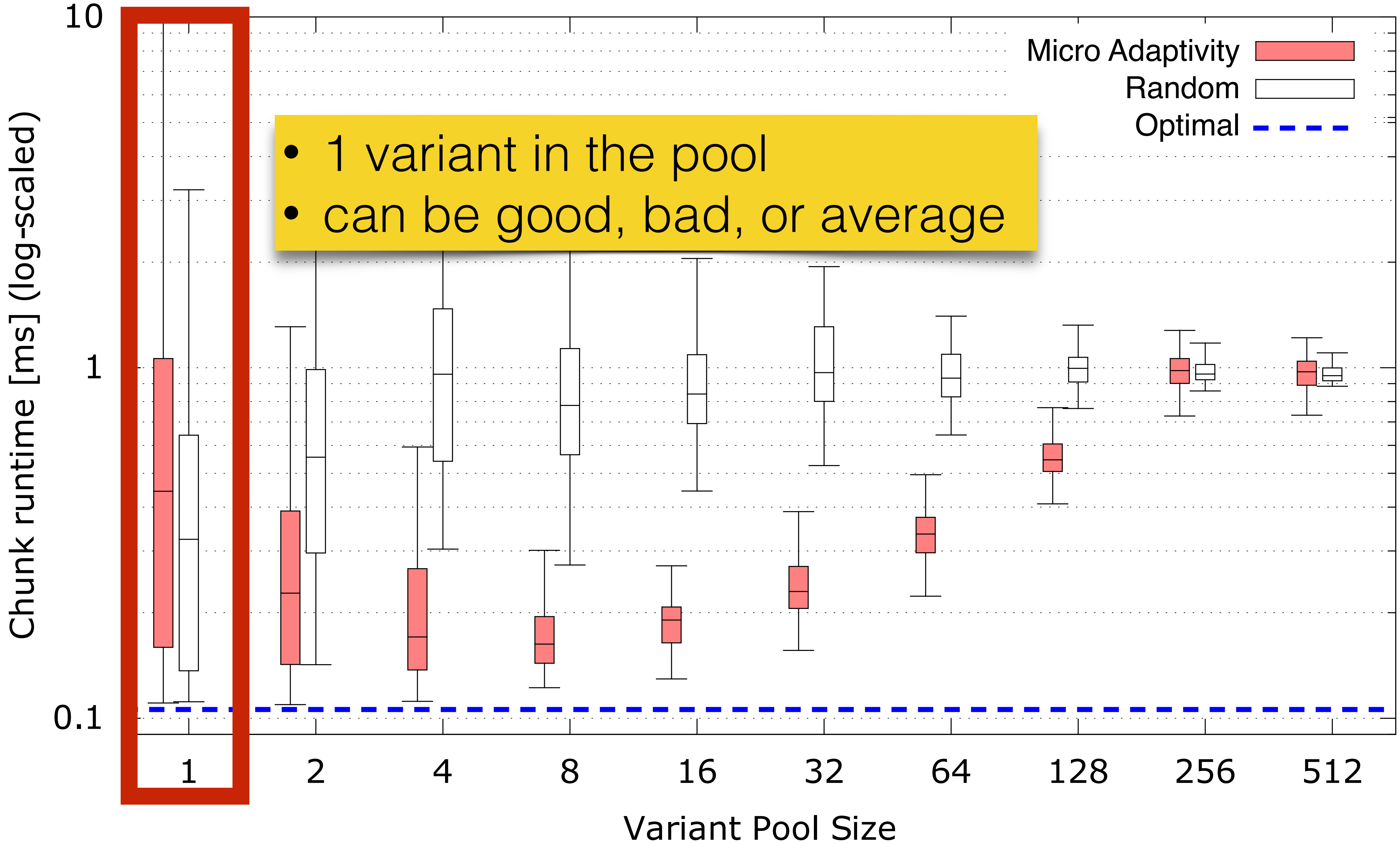
Raducanu et al., "Micro Adaptivity In Vectorwise", SIGMOD, 2013

# Micro Adaptivity With Many Variants



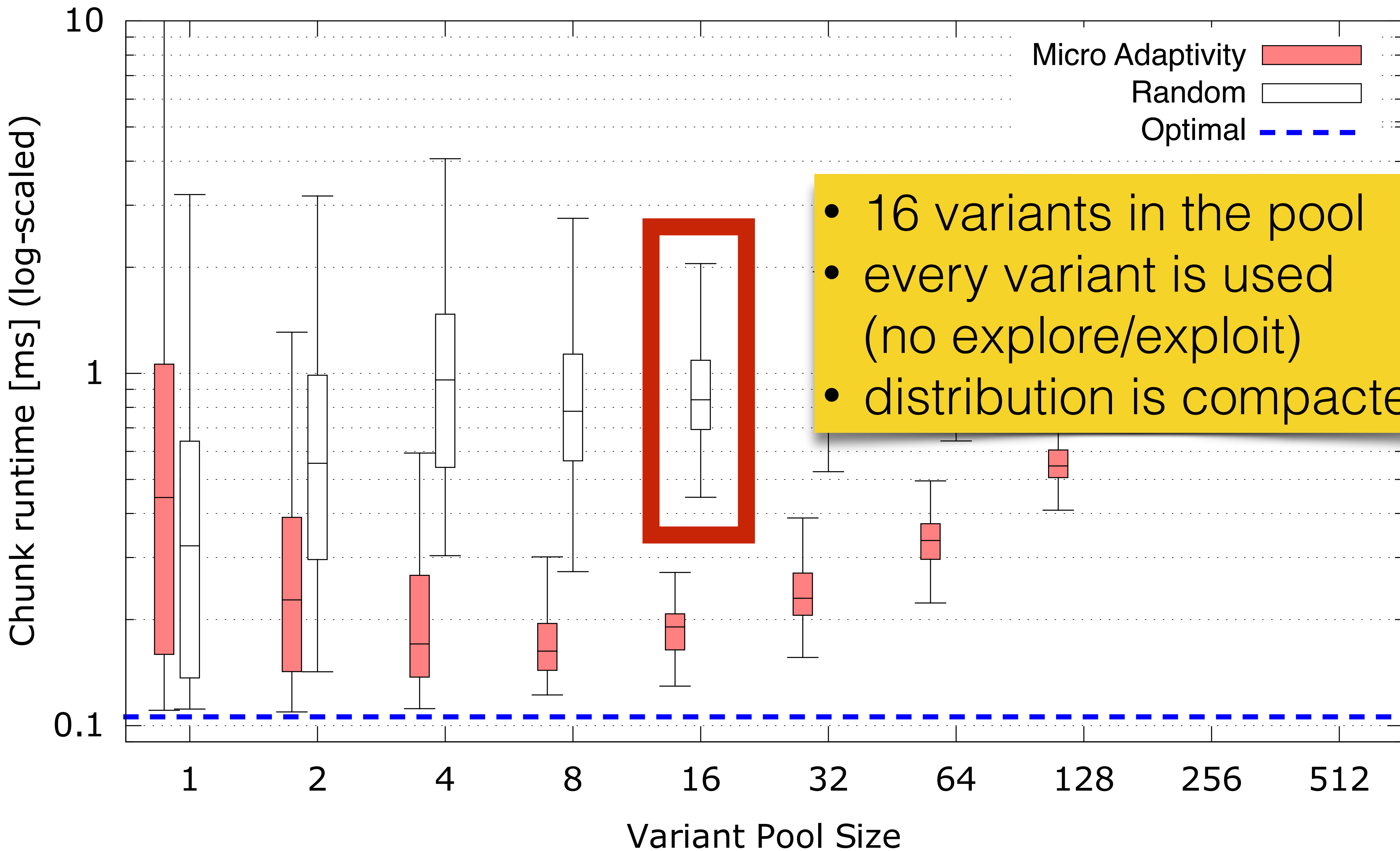

- runtime distribution of 300 queries with 1K chunks
- for each query: select pool of size X from ~5000 different variants

# Micro Adaptivity With Many Variants




- runtime distribution of 300 queries with 1K chunks
- for each query: select pool of size X from ~5000 different variants

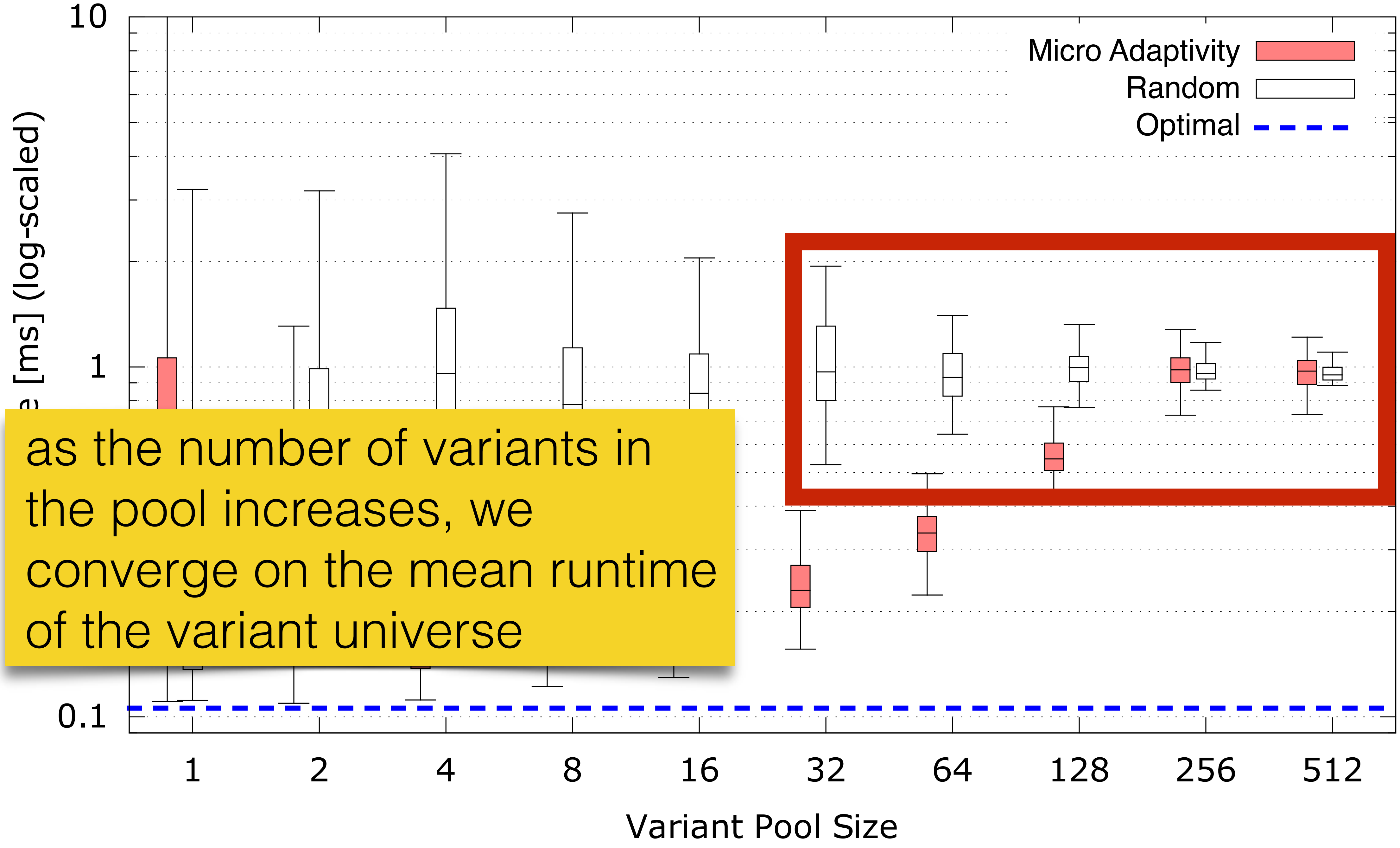
# Micro Adaptivity With Many Variants




- 16 variants in the pool
- every variant is used (no explore/exploit)
- distribution is compacted

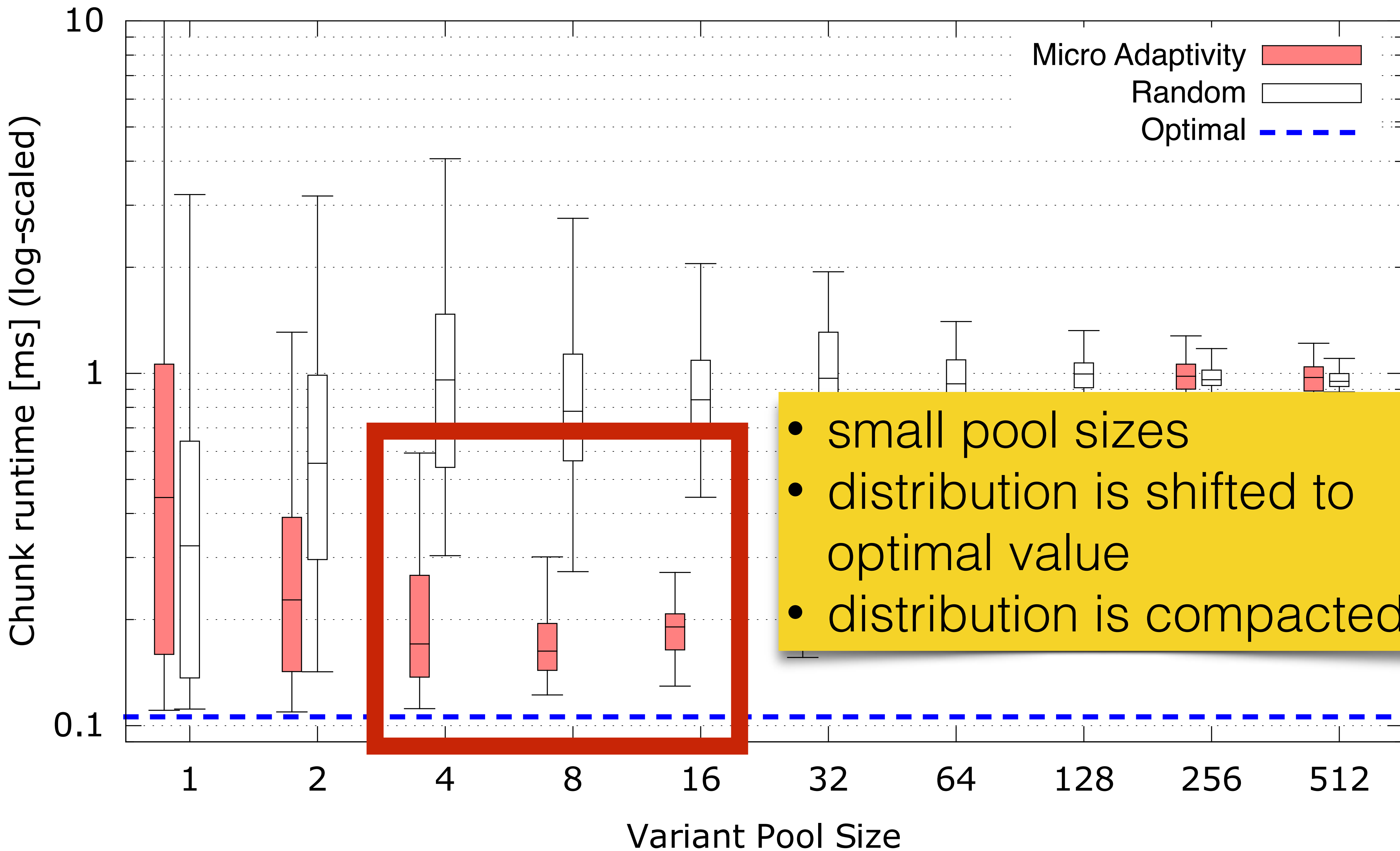
- runtime distribution of 300 queries with 1K chunks
- for each query: select pool of size X from ~5000 different variants

# Micro Adaptivity With Many Variants




- runtime distribution of 300 selection queries with 1K chunks
- for each query: select pool of size X from ~5000 different variants

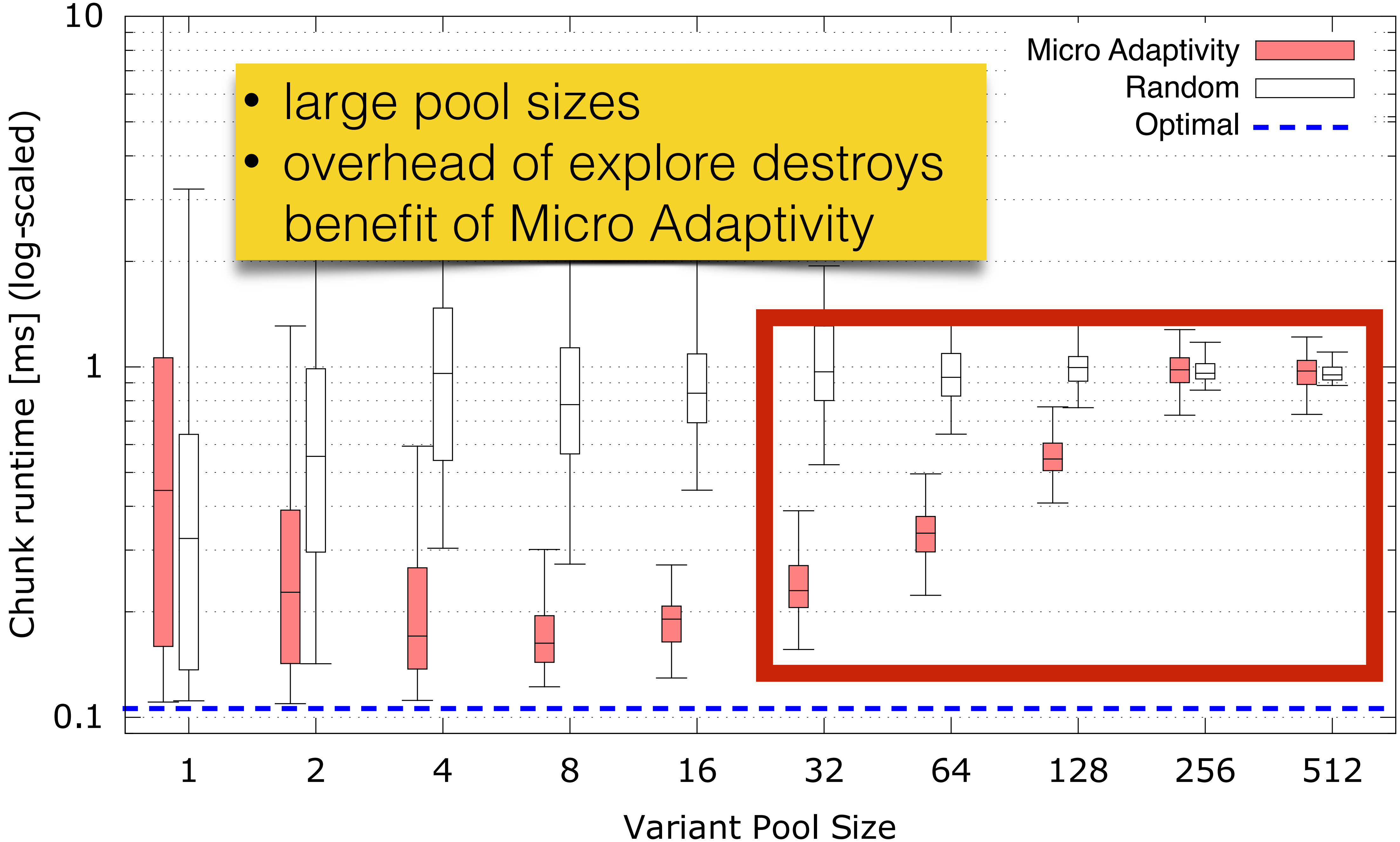
# Micro Adaptivity With Many Variants




- small pool sizes
- distribution is shifted to optimal value
- distribution is compacted

- runtime distribution of 300 queries with 1K chunks
- for each query: select pool of size X from ~5000 different variants

# Micro Adaptivity With Many Variants




- runtime distribution of 300 queries with 1K chunks
- for each query: select pool of size X from ~5000 different variants

# Search Strategies

improve the pool between queries

## Greedy

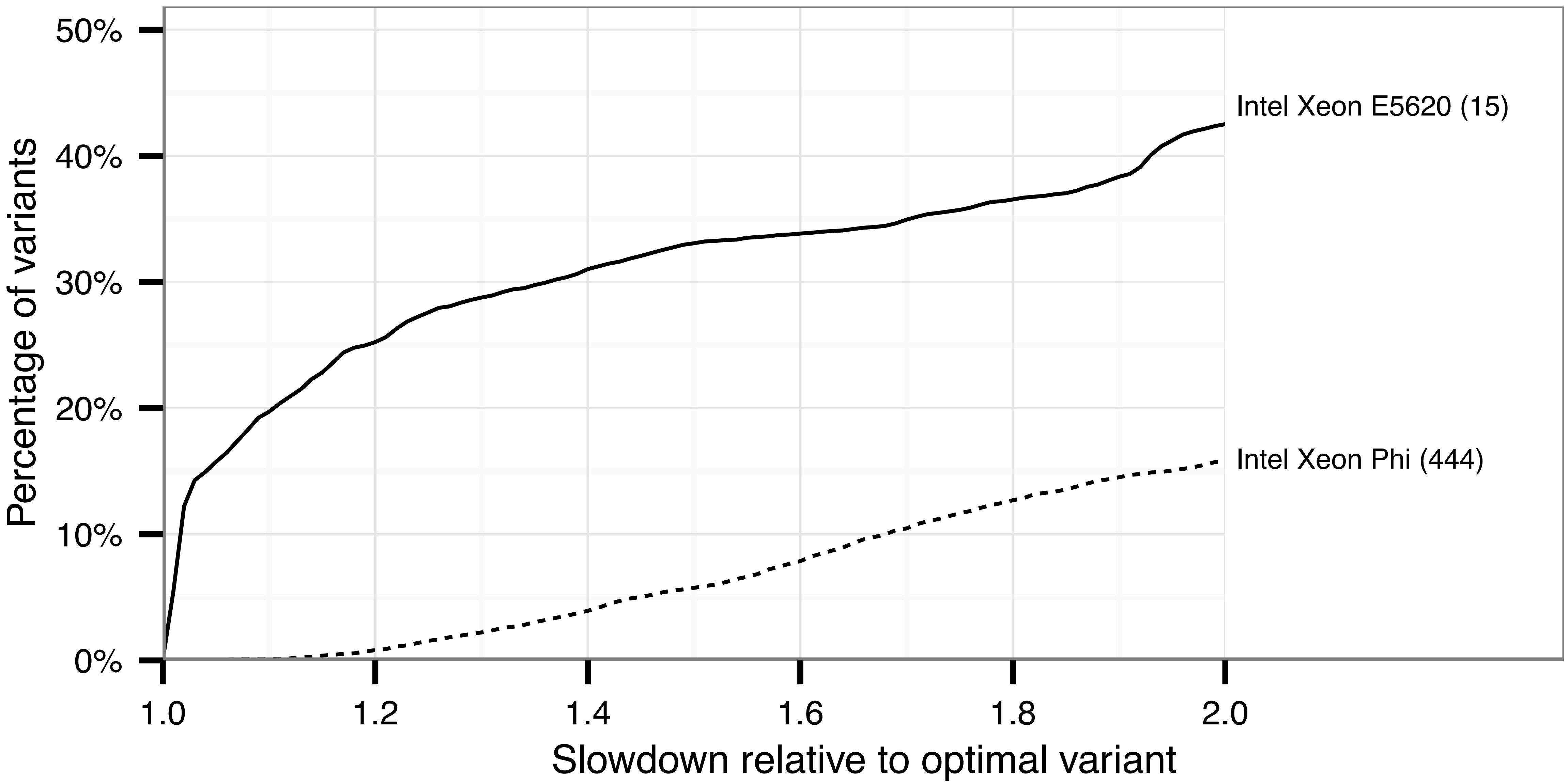
- keep 2 fastest variants
- randomly replace others

## Genetic

- keep 2 fastest variants
- replace others by combining attributes from 2 parents currently in pool
- chance of becoming a parent depends on performance
- mutate variants to get out of local minima

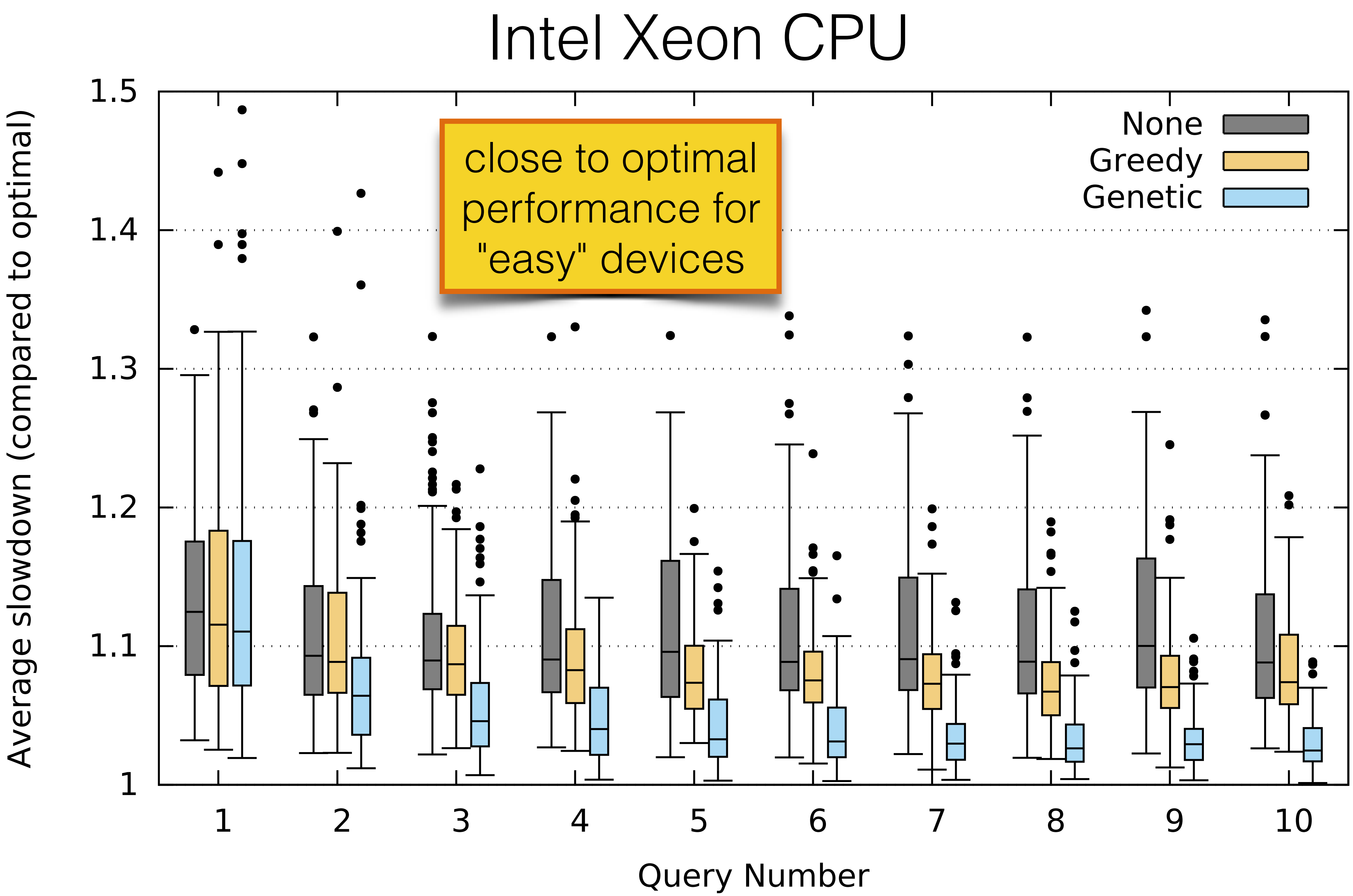
# Competitive Variants

percentage of variants that are at most 2x slower than fastest variant for each device



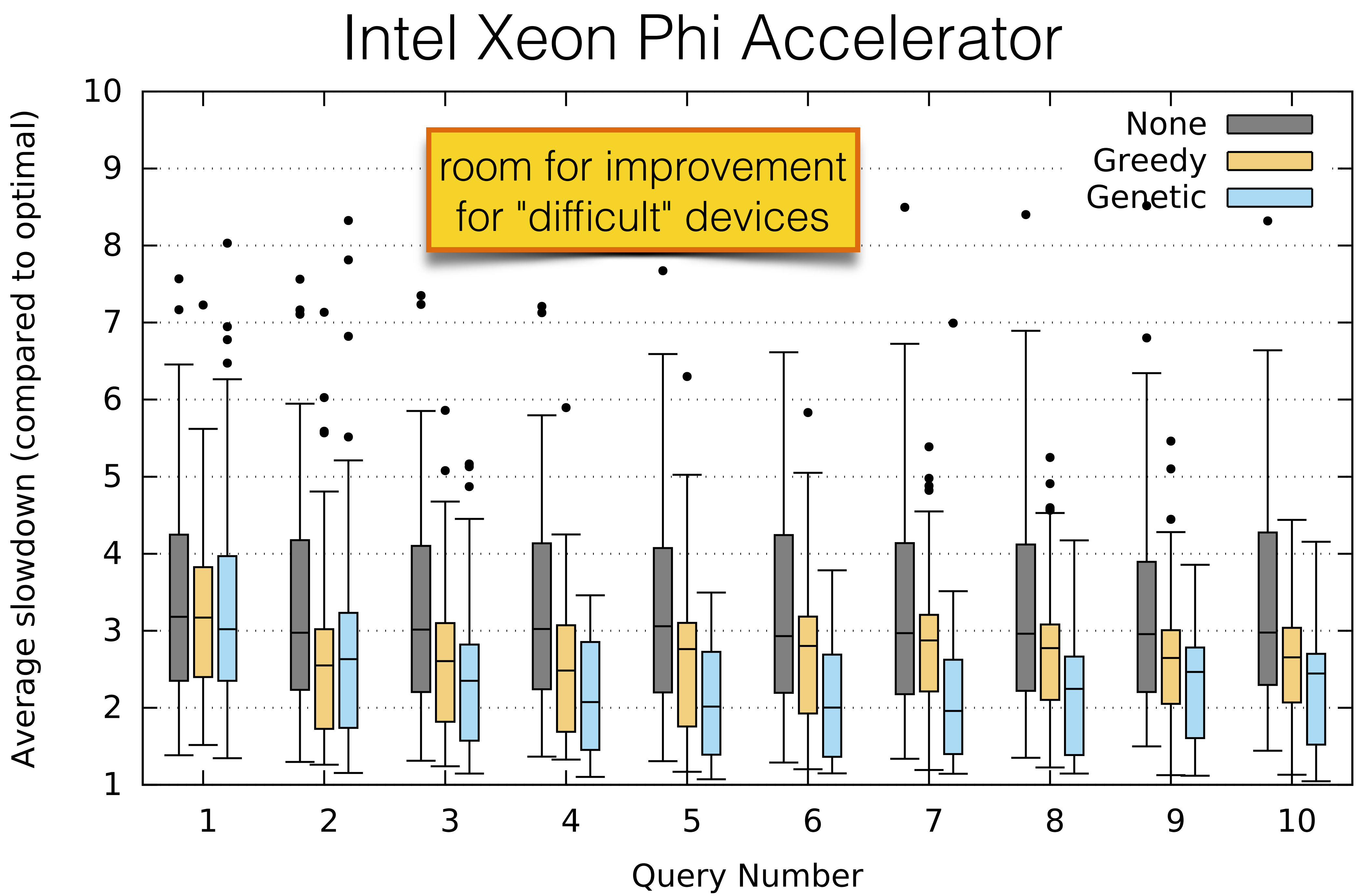

"easy" and "difficult" devices

# Influence of Search Strategies




- 100 series of 10 consecutive selection queries
- working pool: 8 variants chosen randomly at start of series
- baseline None: no updates of working pool between queries

# Influence of Search Strategies



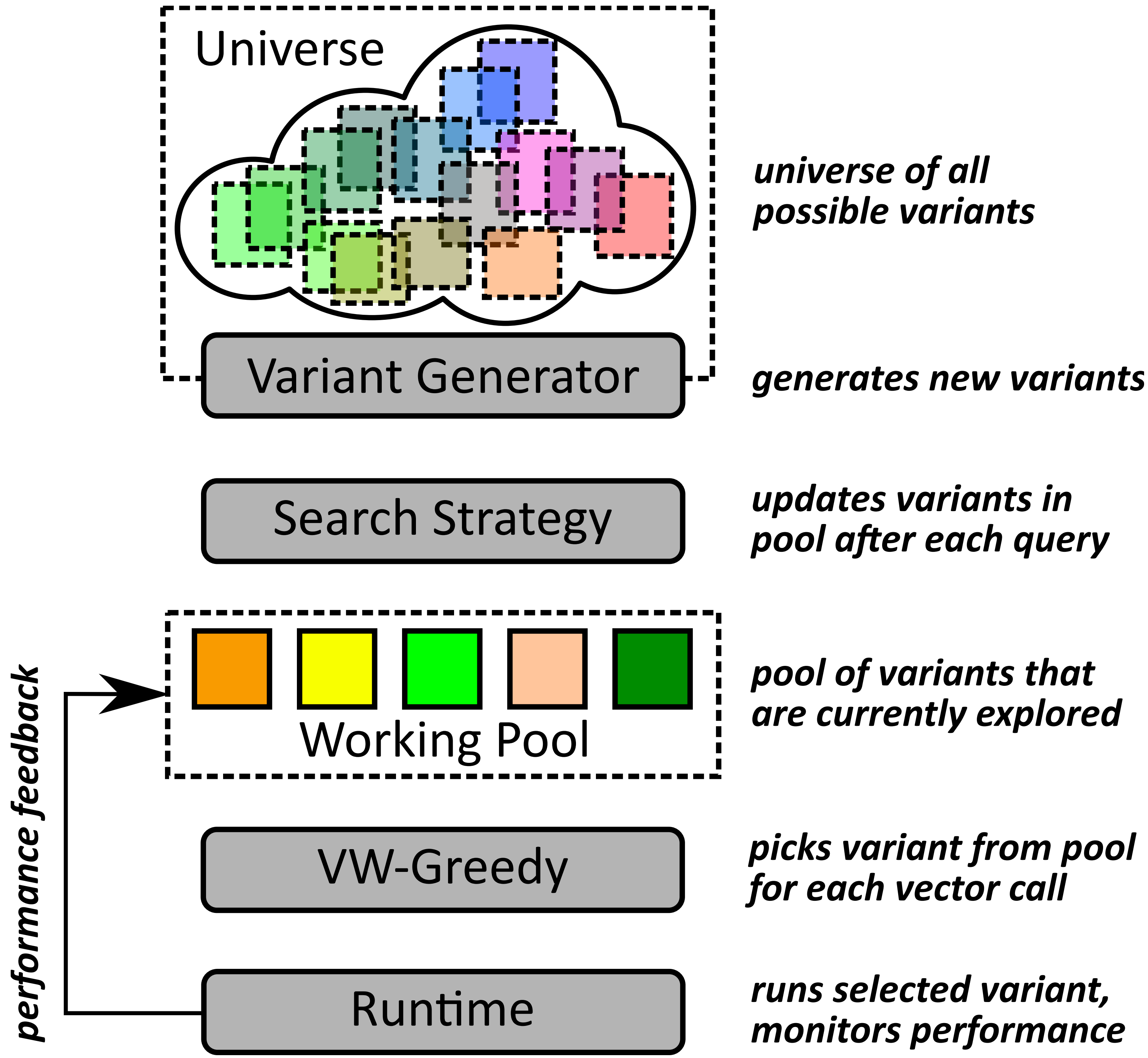

- 100 series of 10 consecutive selection queries
- working pool: 8 variants chosen randomly at start of series
- baseline None: no updates of working pool between queries

# Summary and Outlook

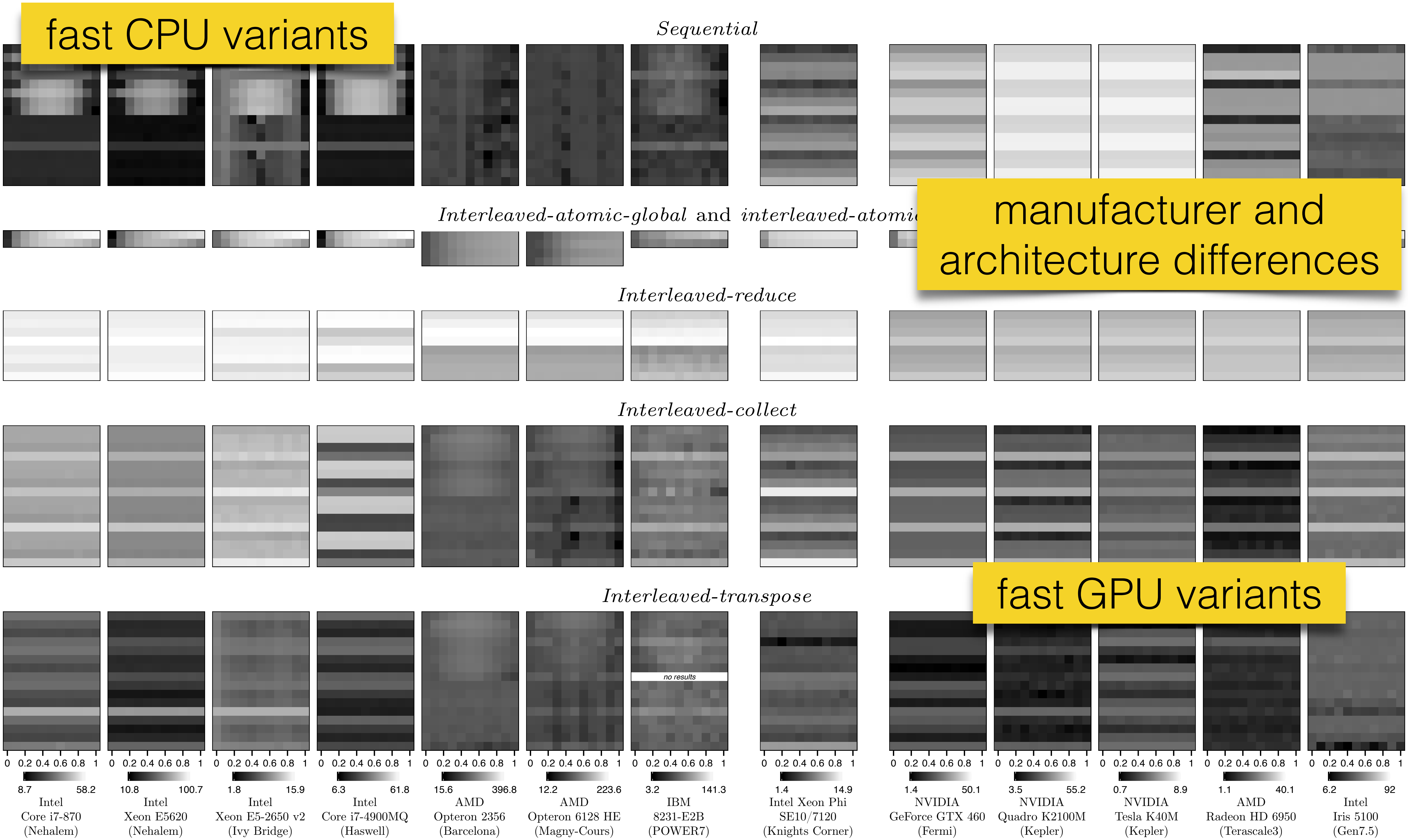
- OpenCL offers functional portability, but lack of performance portability limits usefulness
- we can use query feedback to learn fast data processing operators

- generate variants automatically (step 1 and 2)
- improve search strategies (micro benchmarks, source code metrics, ...)

# Learning Framework



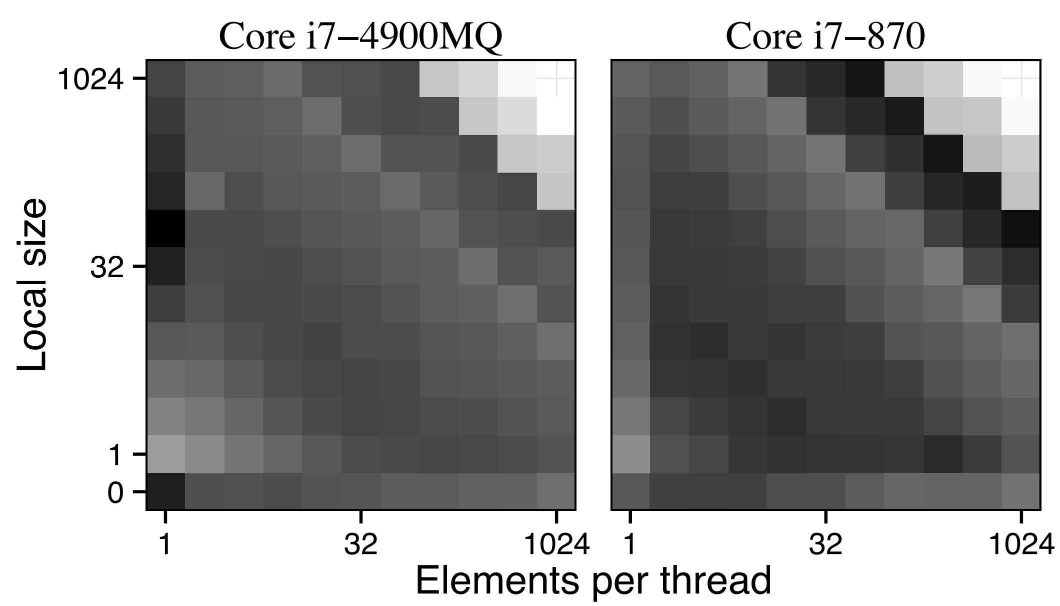
# Processor Characteristics



# Best Variants By Device

| Device | Variant | Elements per thread | Local size |
|---|---|---|---|
| Intel CPUs | 64-bit sequential PU | *device-specific* | |
| NVIDIA GeForce GTX 460 | 32-bit transpose U | 1 | 256 |
| NVIDIA Quadro K2100M | 16-bit transpose (P/PU) | 1/2/4 | 128 |
| NVIDIA Tesla K40M | 16-bit transpose (U) | 1 | 128 |
| AMD Radeon HD 6950 | 8-bit collect (U) | 1 | 128 |
| Intel Iris 5100 | 64-bit transpose P/PU | 1024 | 64/128 |

*P: predicated, U: unrolled*




best variant is hardware-specific