Guided Profiling for Auto-Tuning Array Layouts on GPUs

Nicolas Weber, Sandra C. Amend and Michael Goesele

TU Darmstadt

Motivation

- Memory access is one of the most important performance factors in CUDA applications
- CUDA Programming Guide
  - It is one of the three basic optimization strategies to “Optimize memory usage to achieve maximum memory throughput”
- Performance difference up to an order of magnitude between best and worst implementation
- Experience alone does not guarantee to find the optimal configuration

Motivation

- Tedious to optimize in big GPU applications
  - Layouts: Array of Structs, Structure of Arrays, AoSoA
  - Transpositions of multi-dimensional arrays
  - Size of L1 cache / shared memory
  - Memory placement: Global, Texture, Shared, Local and Constant memory

- Changing GPU architectures require to reoptimize
  - Memory hierarchy was changed in every architecture

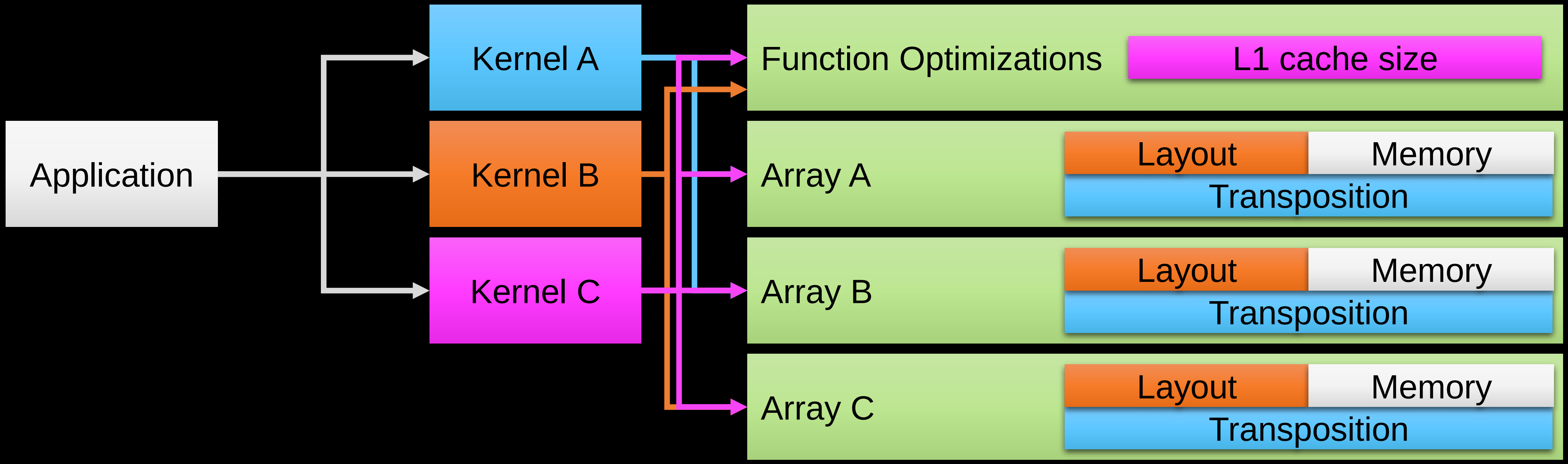
➢ Automated optimization for most GPUs and algorithm
  - We develop an open source auto-tuner to automatically optimize array access in CUDA applications (with minimal programming overhead)

What is the optimal configuration for a kernel?

- Difficult to find an analytical solution
  - Memory access can be input data sensitive
  - Different optima for varying input data
  - Many GPU architectures with different memory hierarchies

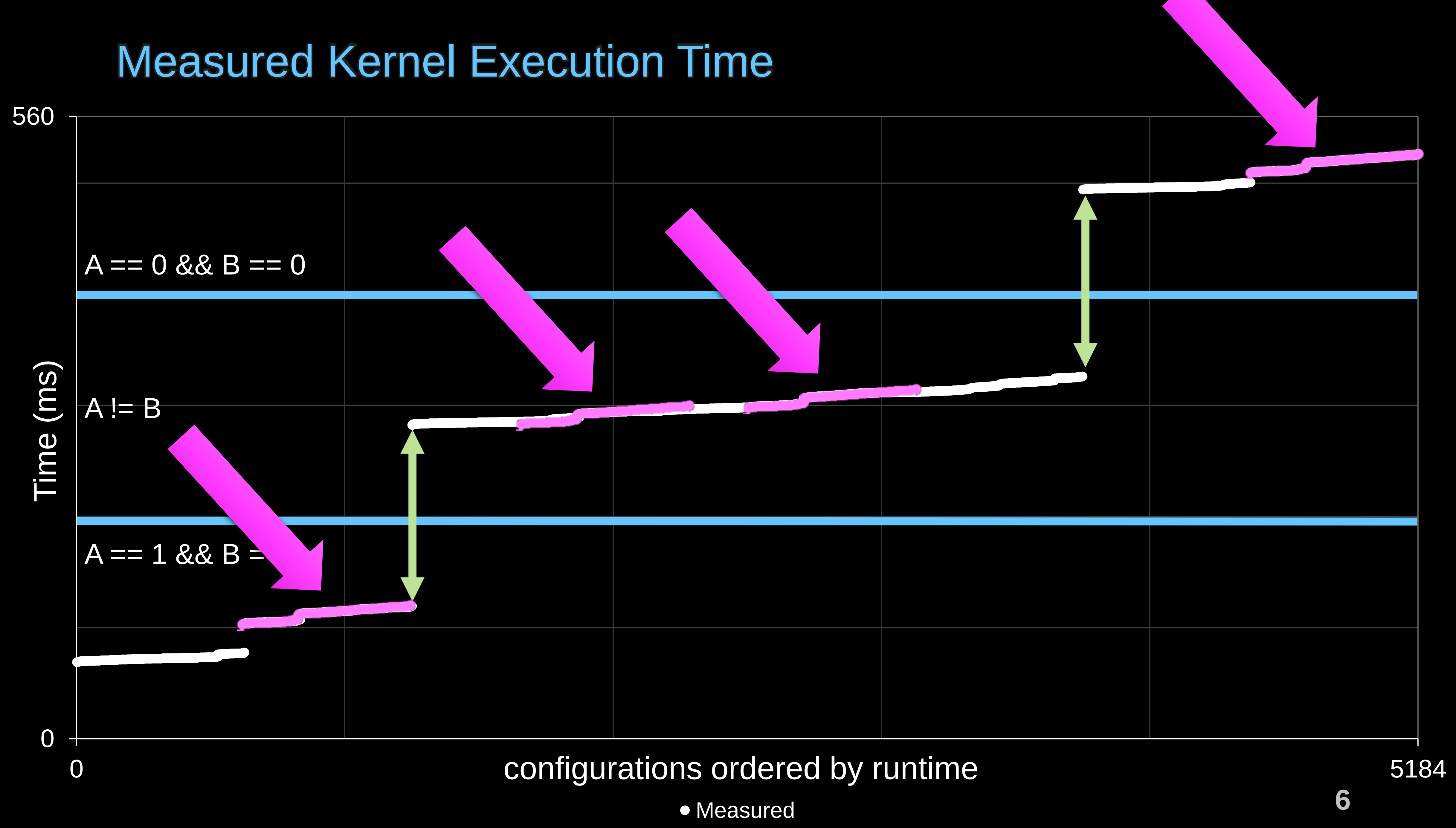
- ➢ Empirical profiling
  - Requires to compile & execute many different implementations
  - Very time intensive

High Dimensionality

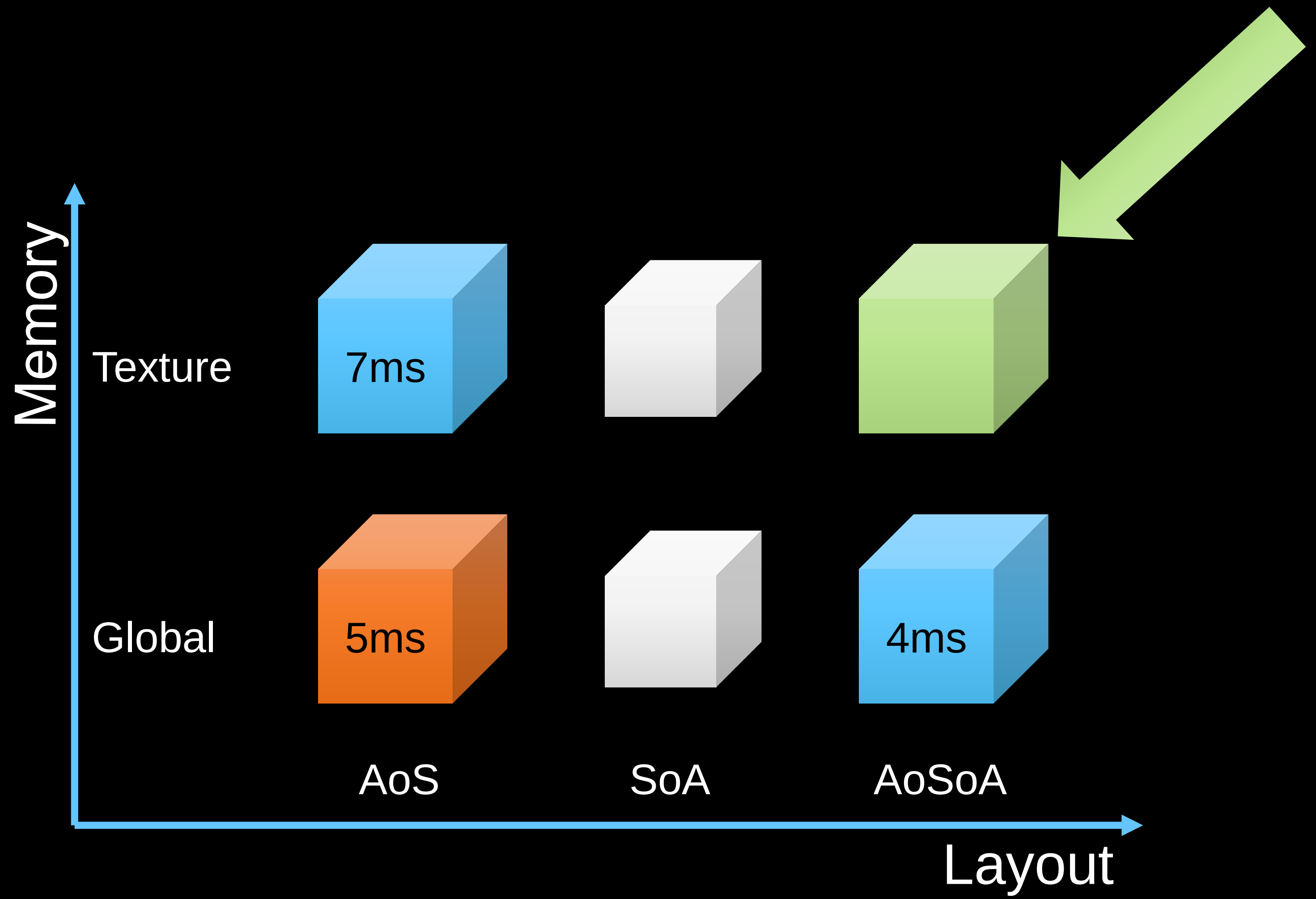


- Up to several million configurations!
  - $1000000 \cdot \left(\frac{5s\ (Compilation\ time)}{16\ (Cores)} + 0.5s\ (Execution\ time) \right) \geq 9\ days$

Measured Kernel Execution Time

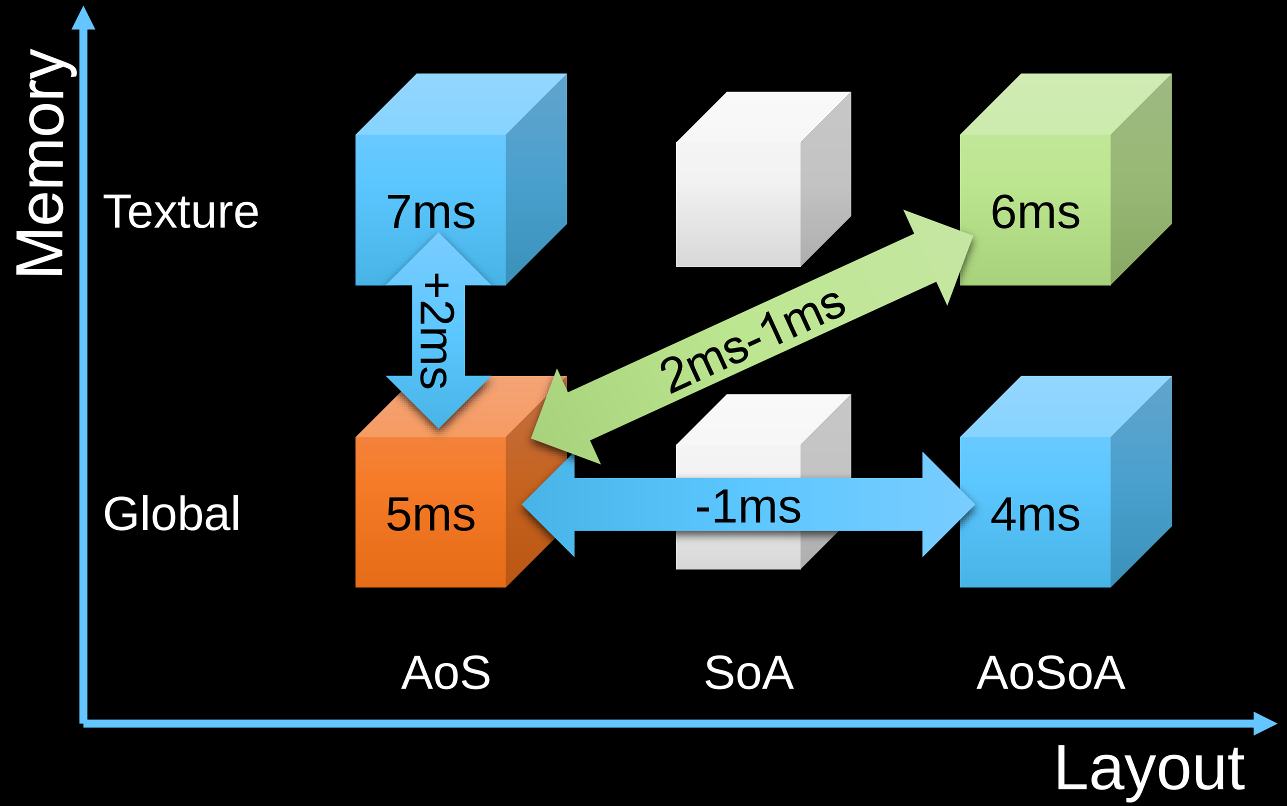


Toy Example: Performance Estimation 2D

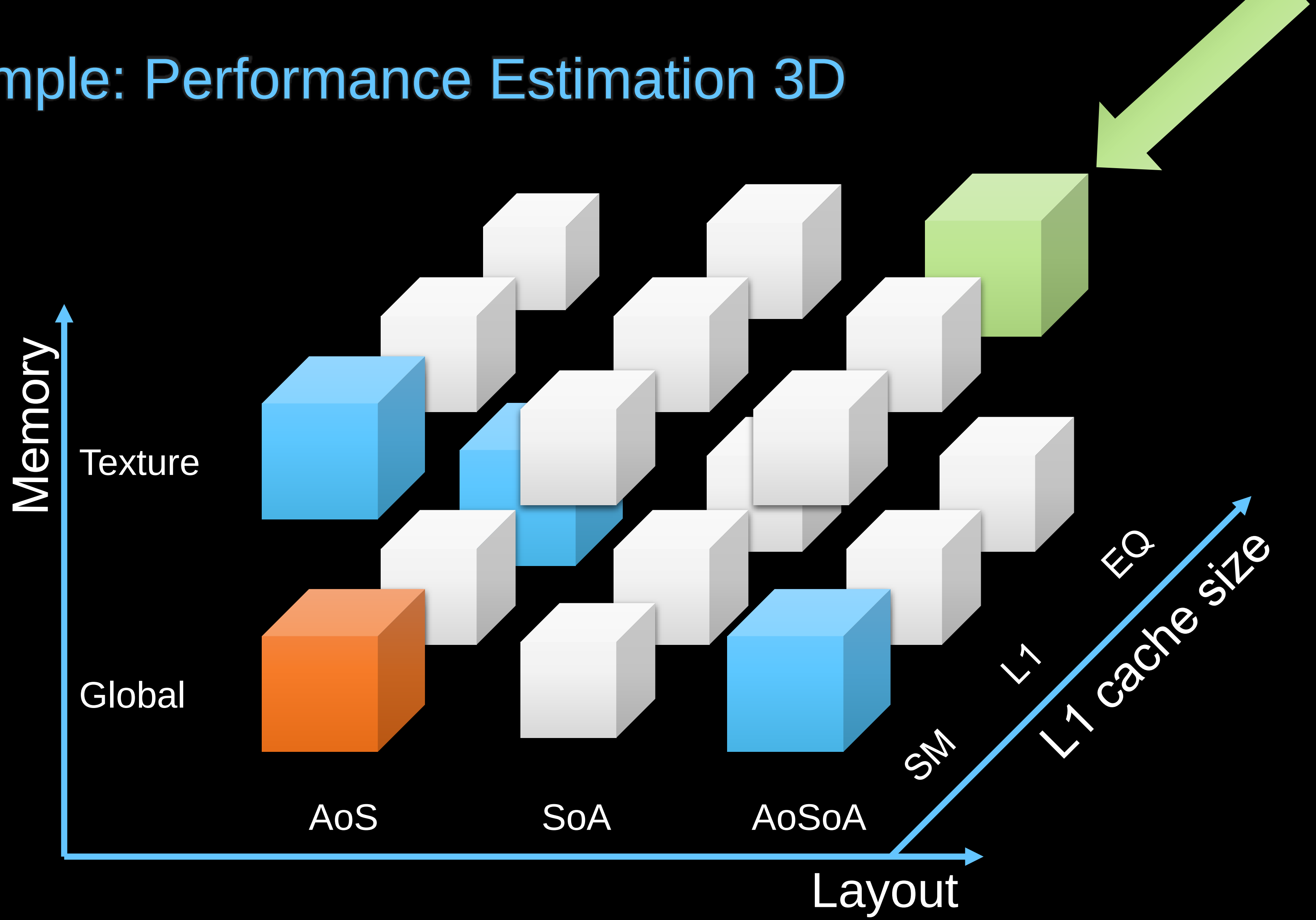


Toy Example: Performance Estimation 2D

- Predicted Execution Time
  - Execution time of Base + Sum(Δ(Base, Support Configurations))



Toy Example: Performance Estimation 3D



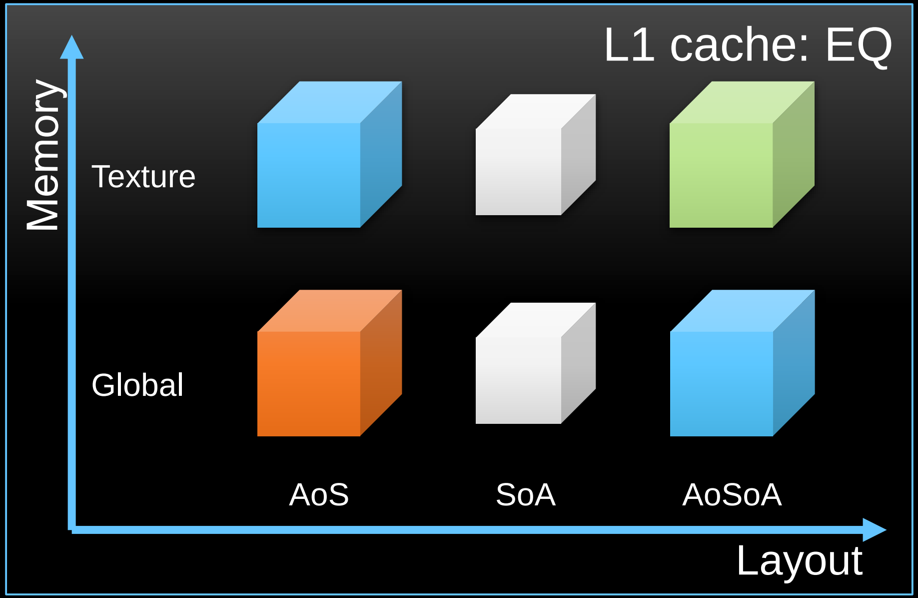
Non-Linear Relationship

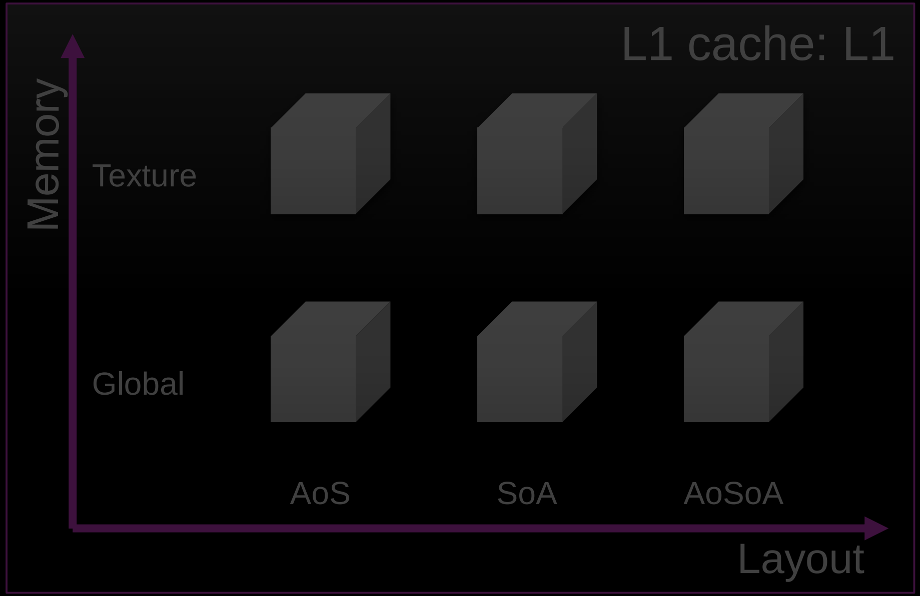
- Not all configurations are linearly related to each other

- Shared dimensions
  - Affect all arrays
  - L1 cache size

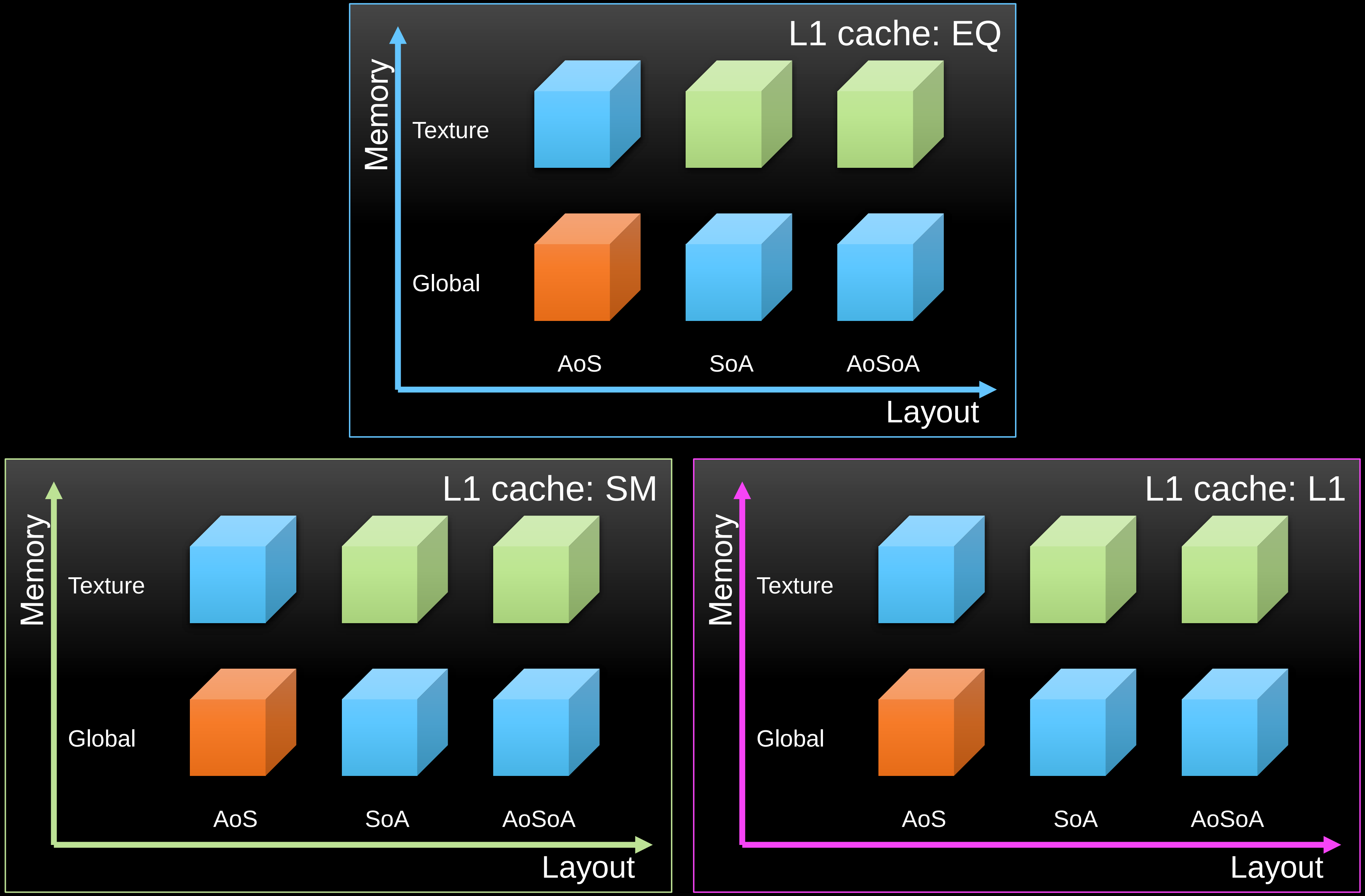
- Independent dimensions
  - Only affect one array
  - Layout, memory and transposition

Toy Example: Prediction Domains

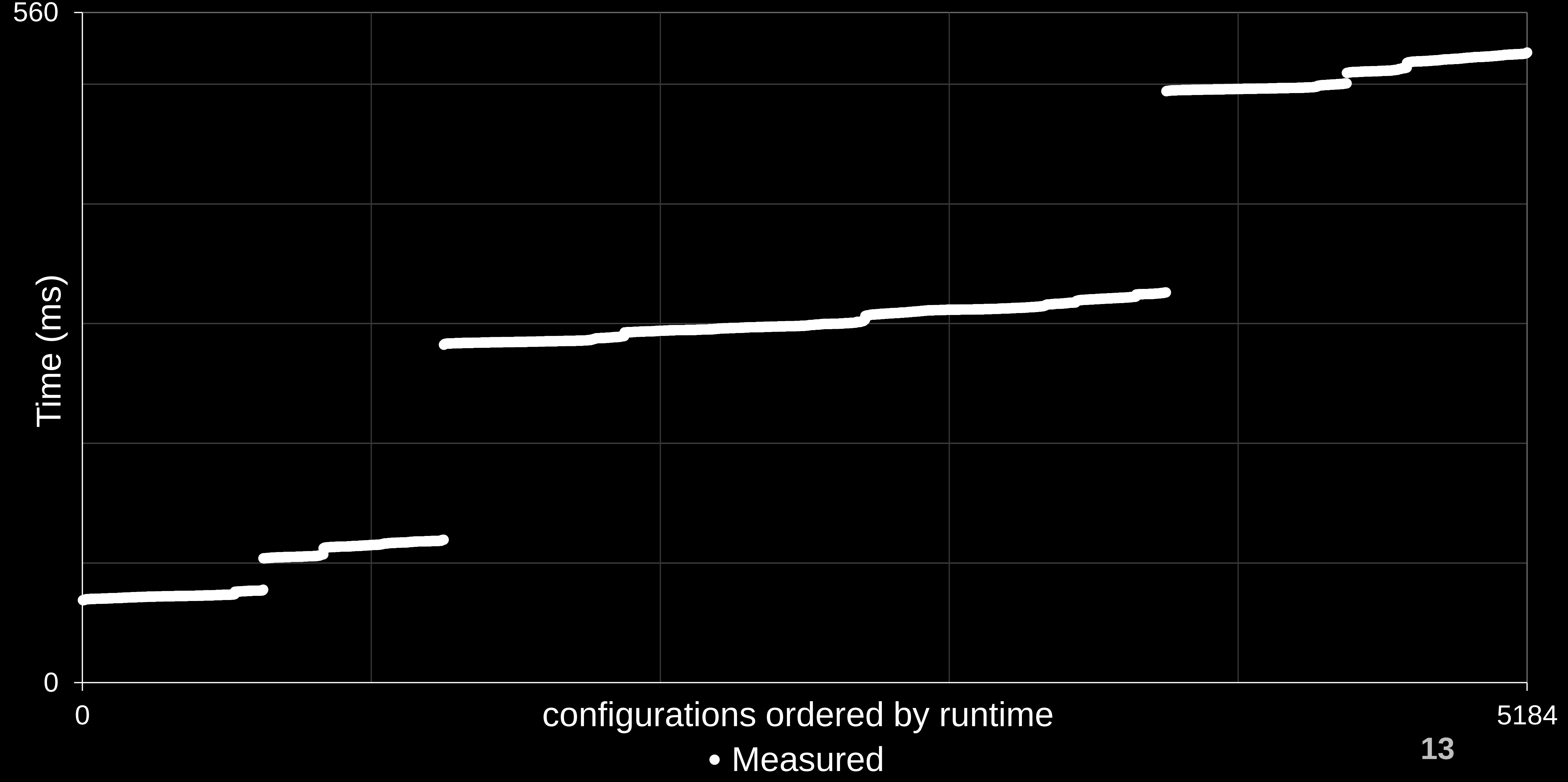




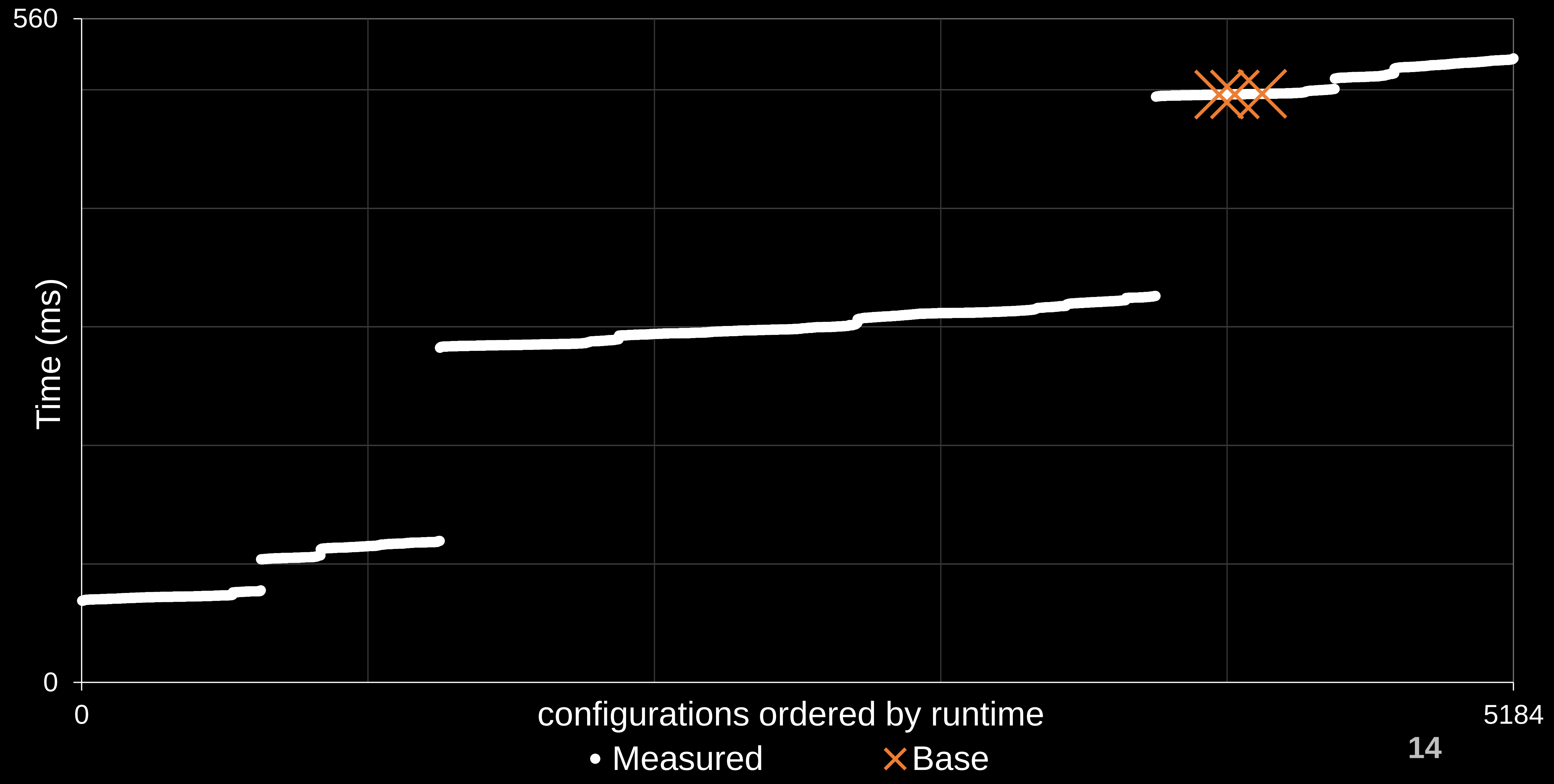
Toy Example: Prediction Domains



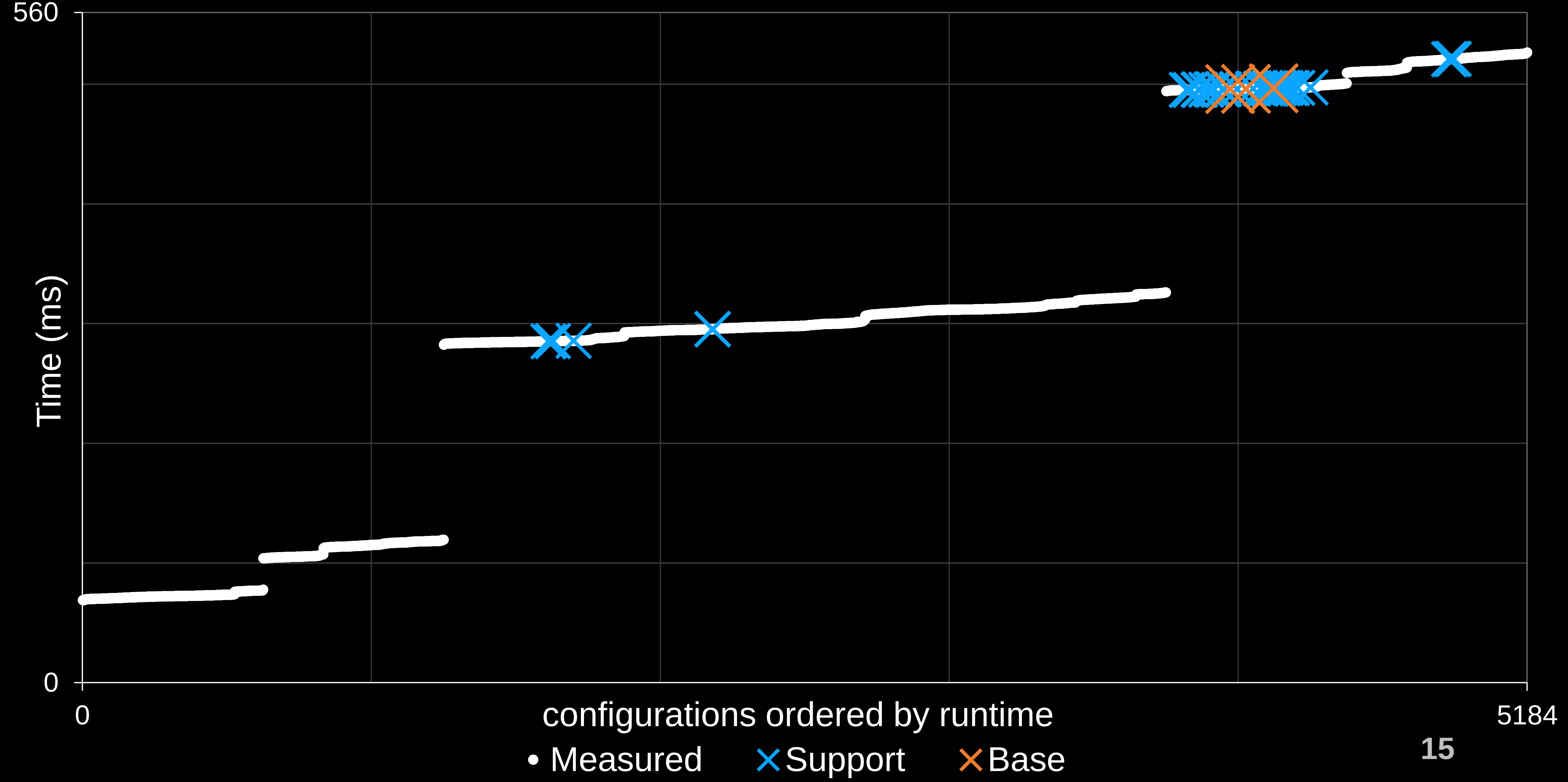
Real Example: Measured Time



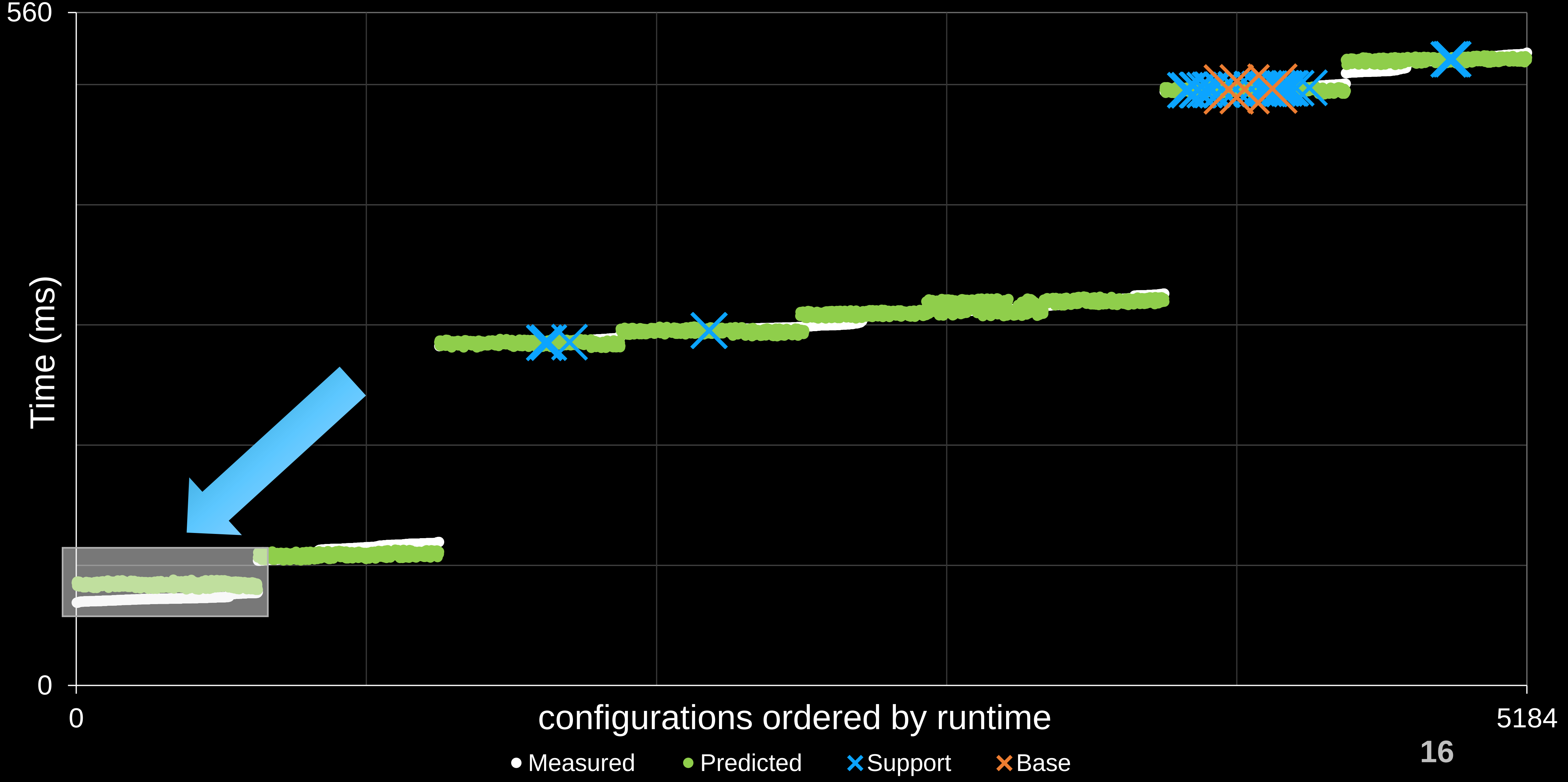
Real Example: Base Configurations



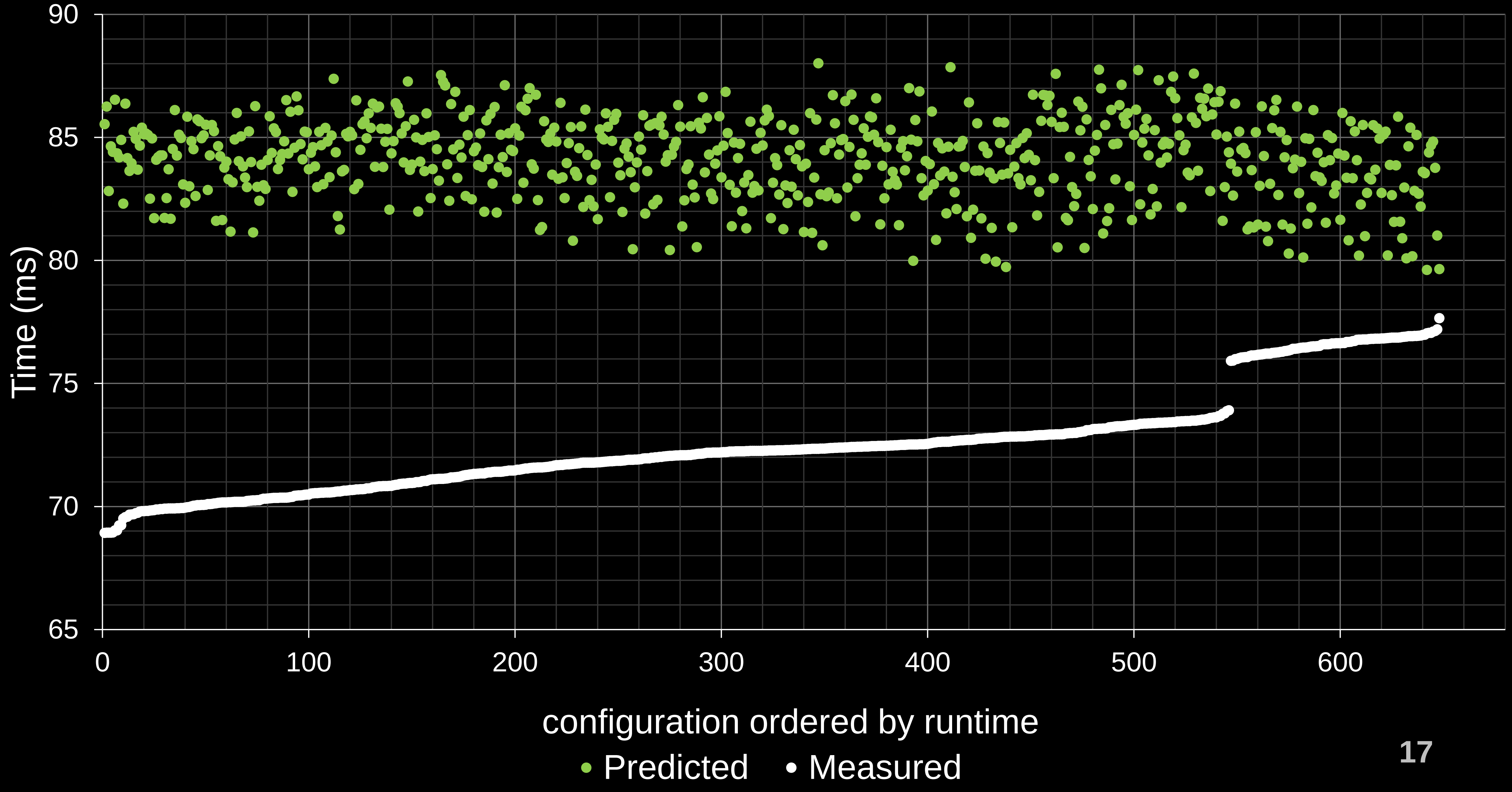
Real Example: Support Configurations



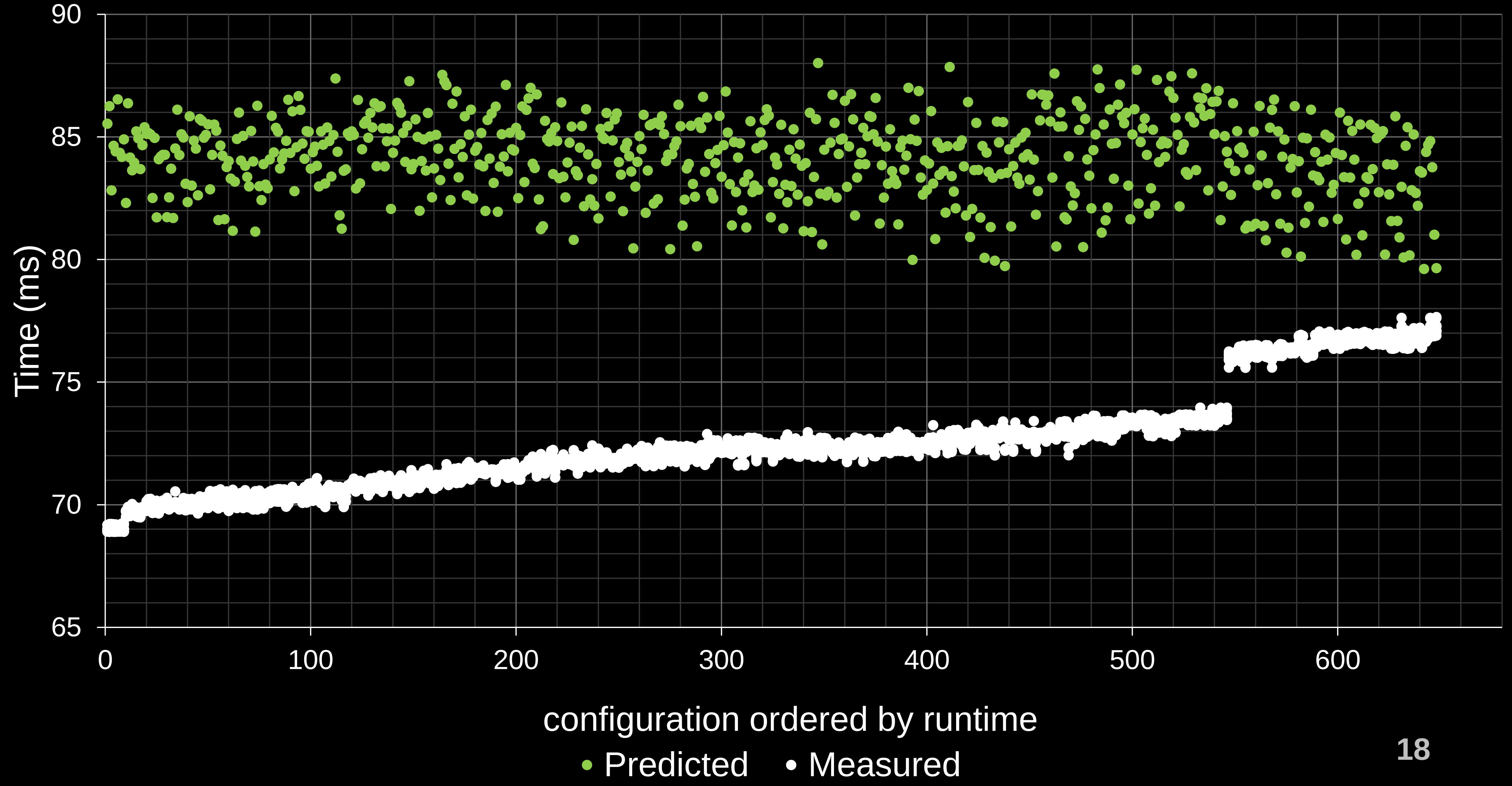
Real Example: Prediction



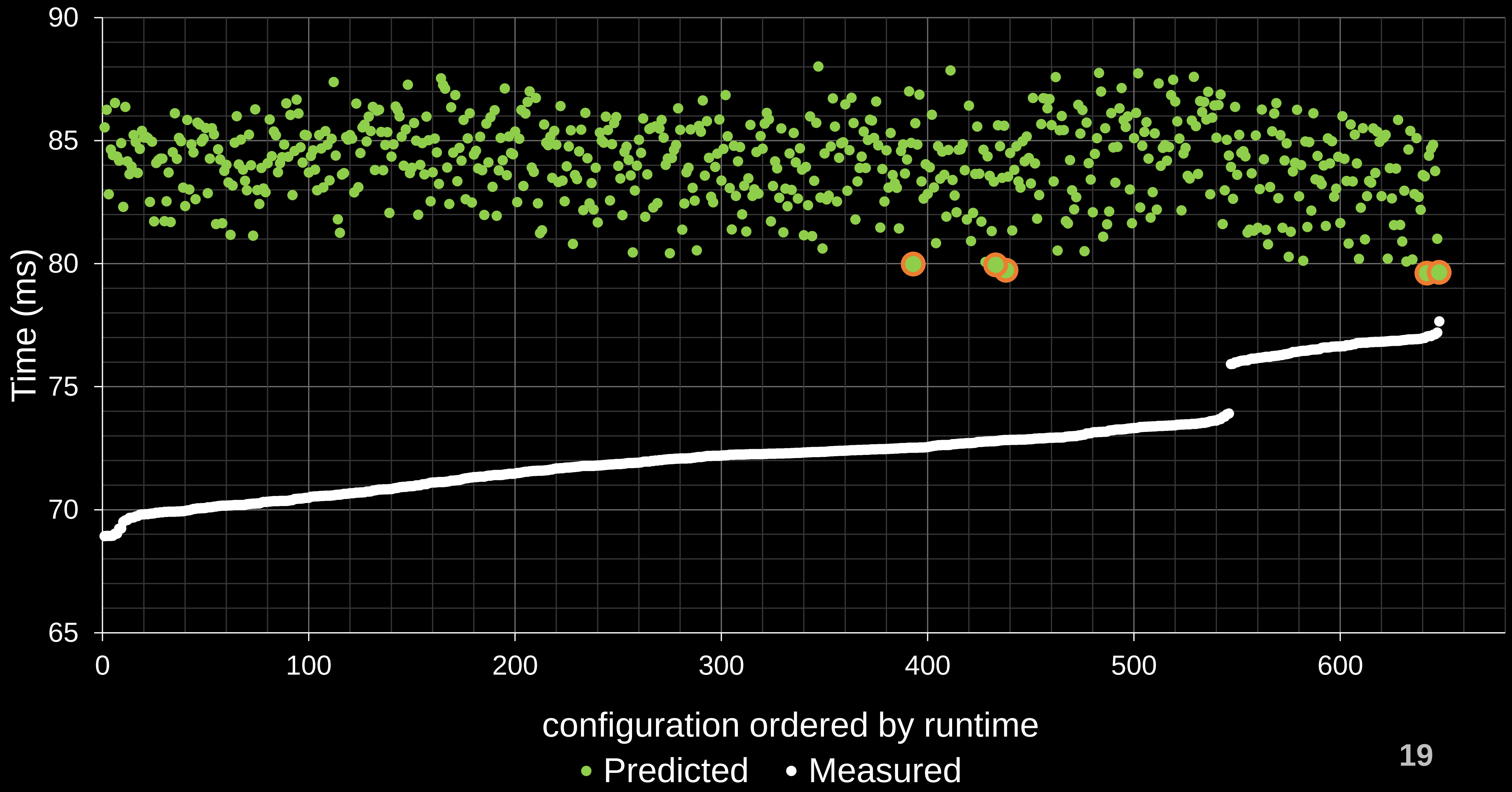
Real Example: Prediction (zoom in)



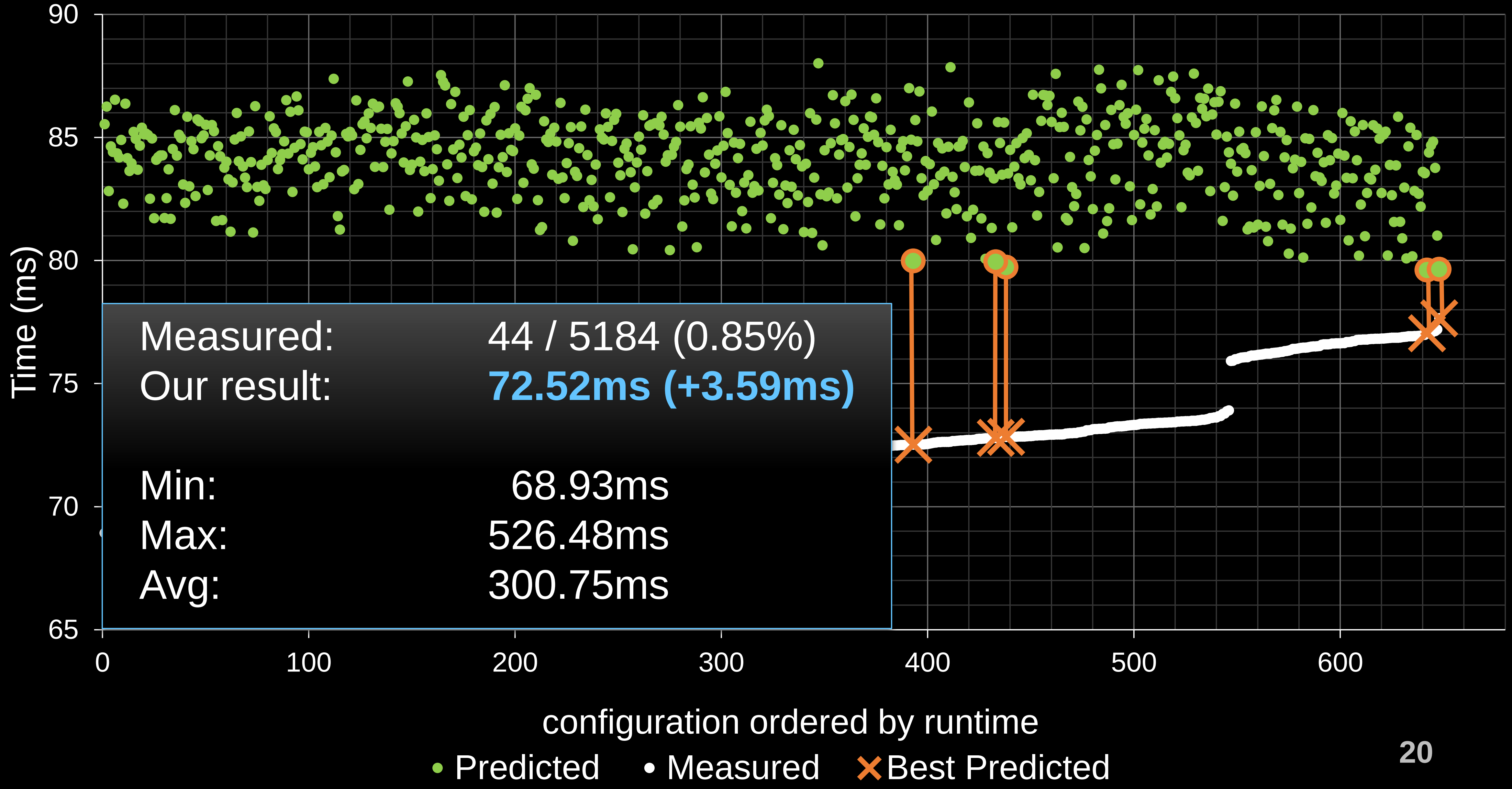
Real Example: Prediction (zoom in)



Real Example: Prediction (zoom in)



Real Example: Prediction (zoom in)



EVALUATION

1. Benchmark: BitonicSort

```
struct {
    long a;
    int b;
    short c;
    char d;
}
```

- Sorting for each field, A < B < C < D
- Values limited to 0…1023 to cause equal columns

- 2 Kernels
- 27 configurations

2. Benchmark: KD-Tree Builder

- 9 Kernels
- > 570k configurations

3. Benchmark: REYES

- 4 Kernels
- > 2.4M configurations

Profiling Algorithms

- Exhaustive Search [Muraladinharan et al. 2014]
  - Tries all possible configurations
- Greedy Profiling [Liu et al. 2008]
  - Optimize each dimension after each other
- Evolutionary Algorithm [Jordan et al. 2012]
  - Starts with a random population of configurations
  - Good configurations are stored
  - Bad configurations are mutated, combined or randomly sampled

Evaluation

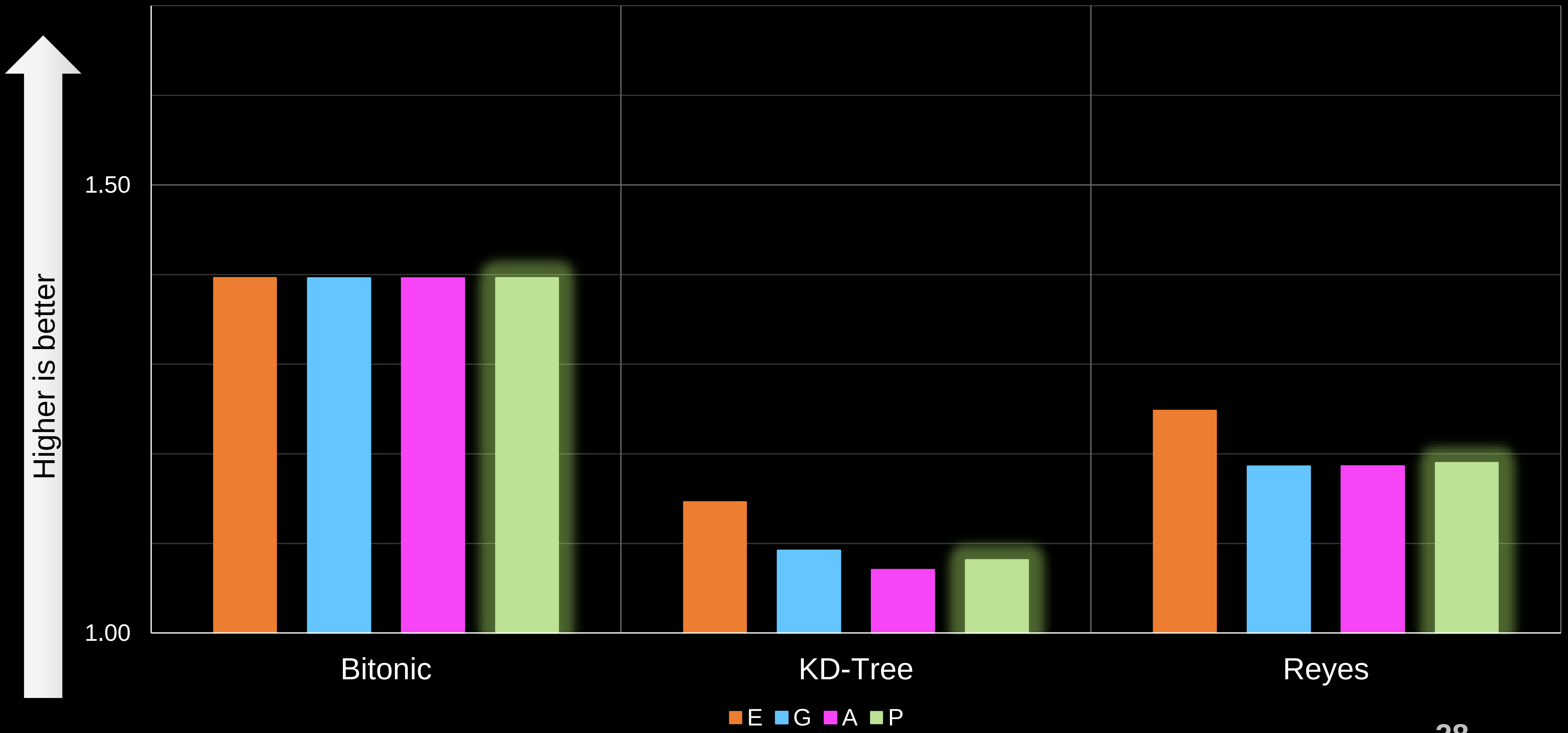
- Profiling Algorithms
  - Exhaustive Search (E)
  - Greedy Algorithm (G)
  - Evolutionary Algorithm (A)
  - Our Algorithm (P)

- GPUs
  - GeForce GTX980 (Maxwell)
  - Tesla K20 (Kepler)

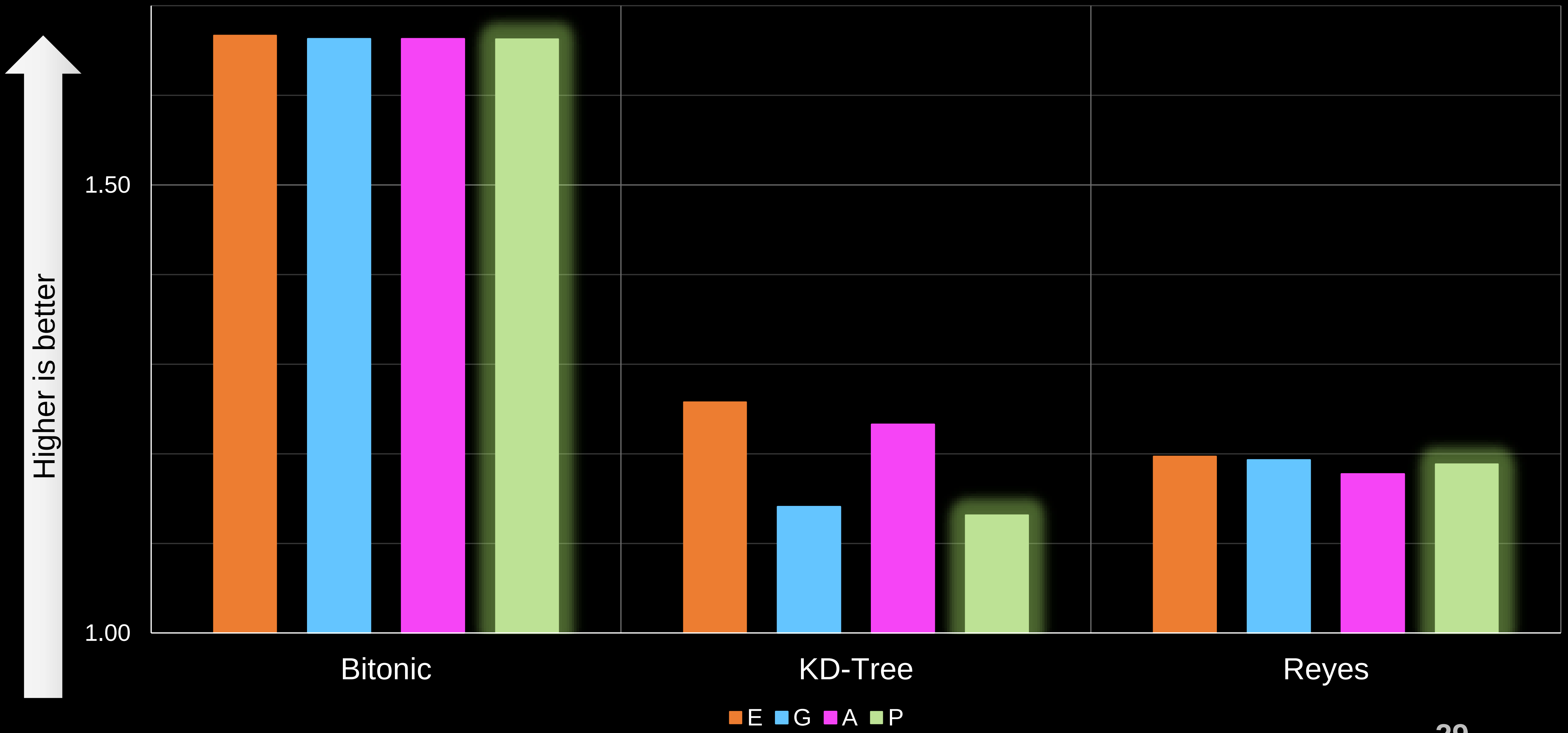
- CUDA WatchDog: kills configurations which exceed the execution time of the best found

QUALITY

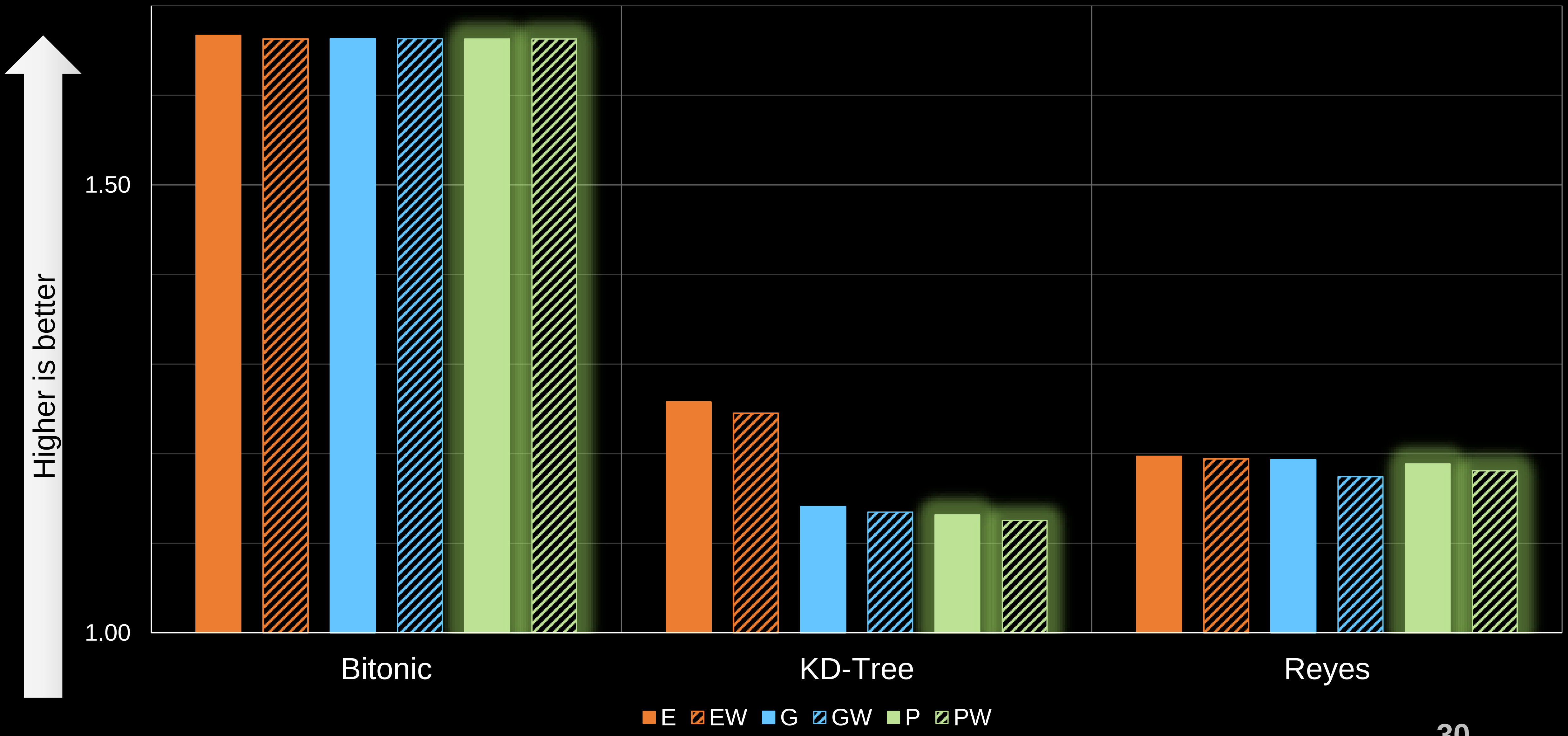
Execution Speed Up: GTX980 w/o WatchDog



Execution Speed Up: Tesla K20 w/o WatchDog

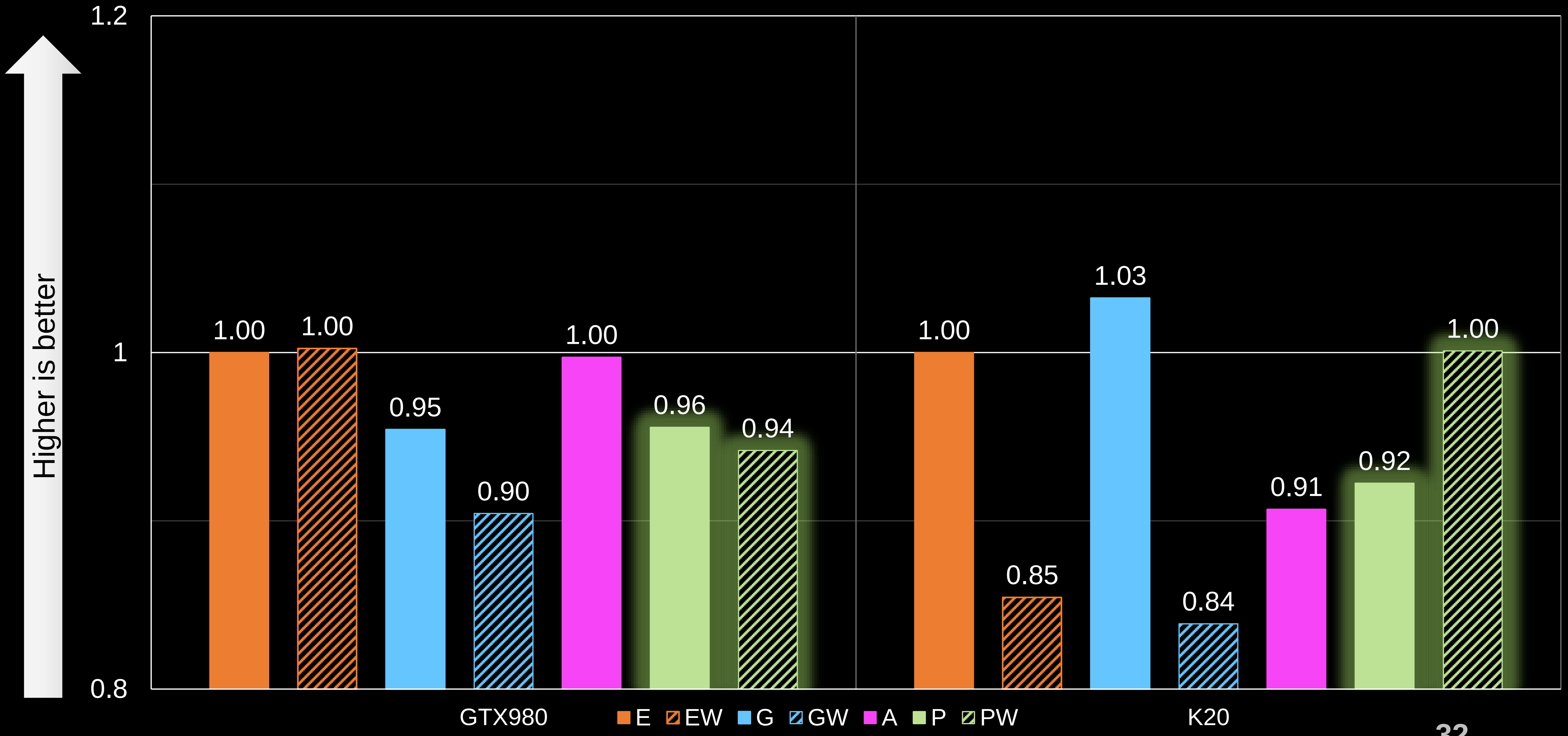


Execution Speed Up: Tesla K20 with WatchDog

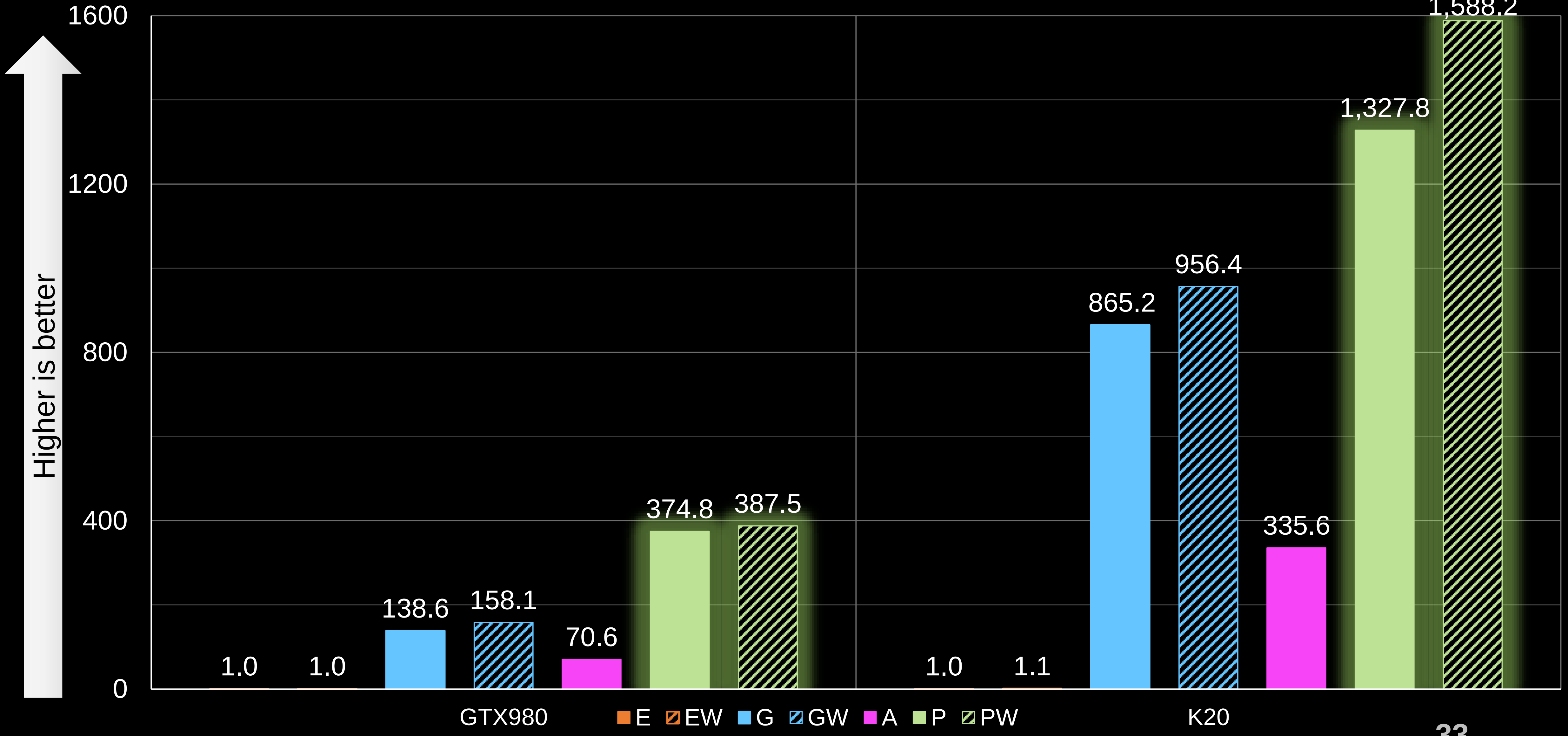


Speed Up

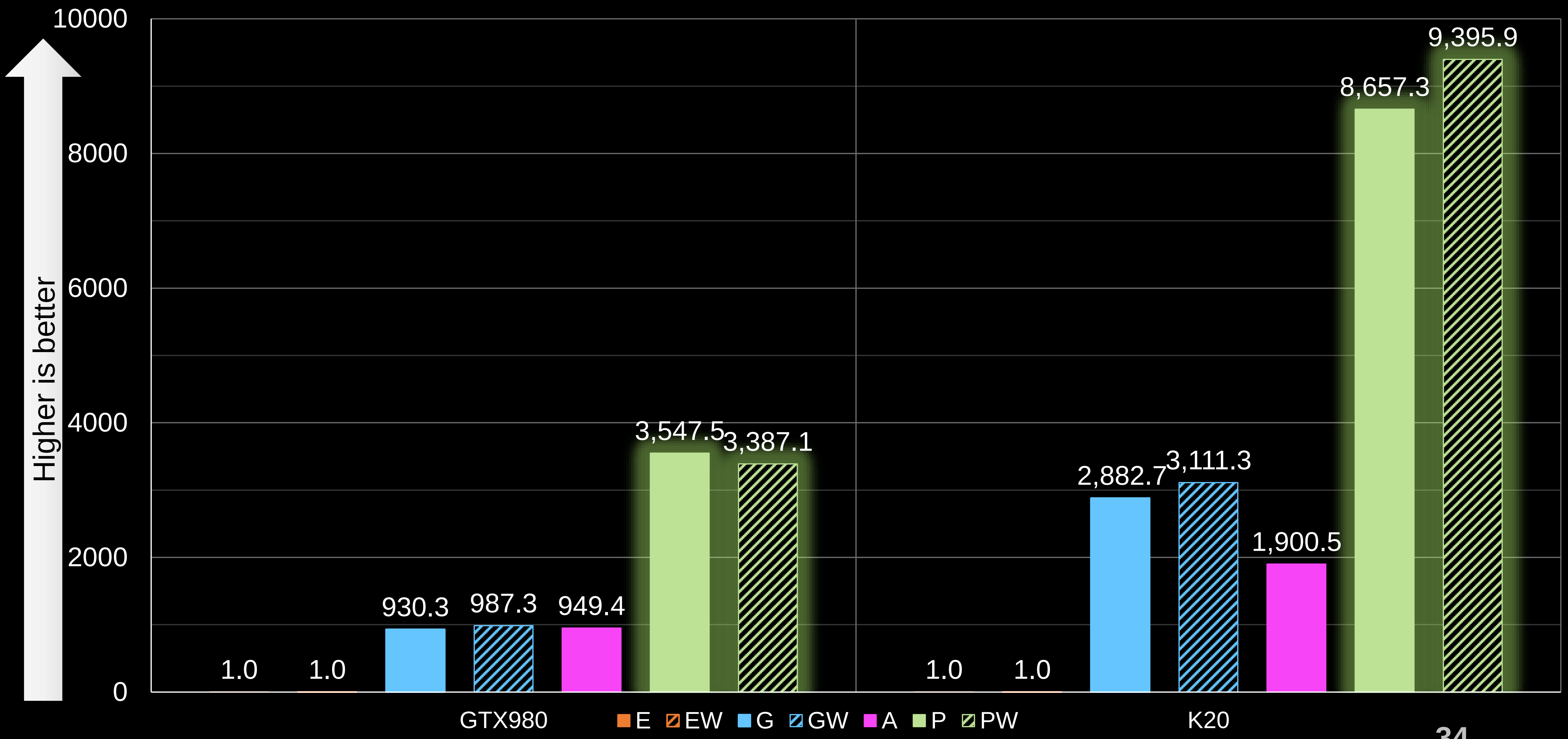
Profiling Speed Up: BitonicSort



Profiling Speed Up: KD-Tree Builder



Profiling Speed Up: REYES



Summary

- Introduced prediction guided profiling algorithm
  - up to 5.5x faster than other state of the art methods
    - while achieving comparable results
  - up to 9300x faster than exhaustive search
    - 10 days 20 hours → 1 minute 40 seconds

- Limitations
  - No global optimization → only one kernel at once is optimized

Thank you for your attention!

Source Code available @

http://tinyurl.com/matog
(BSD 3-Clause license)

Contact: matog@gris.tu-darmstadt.de