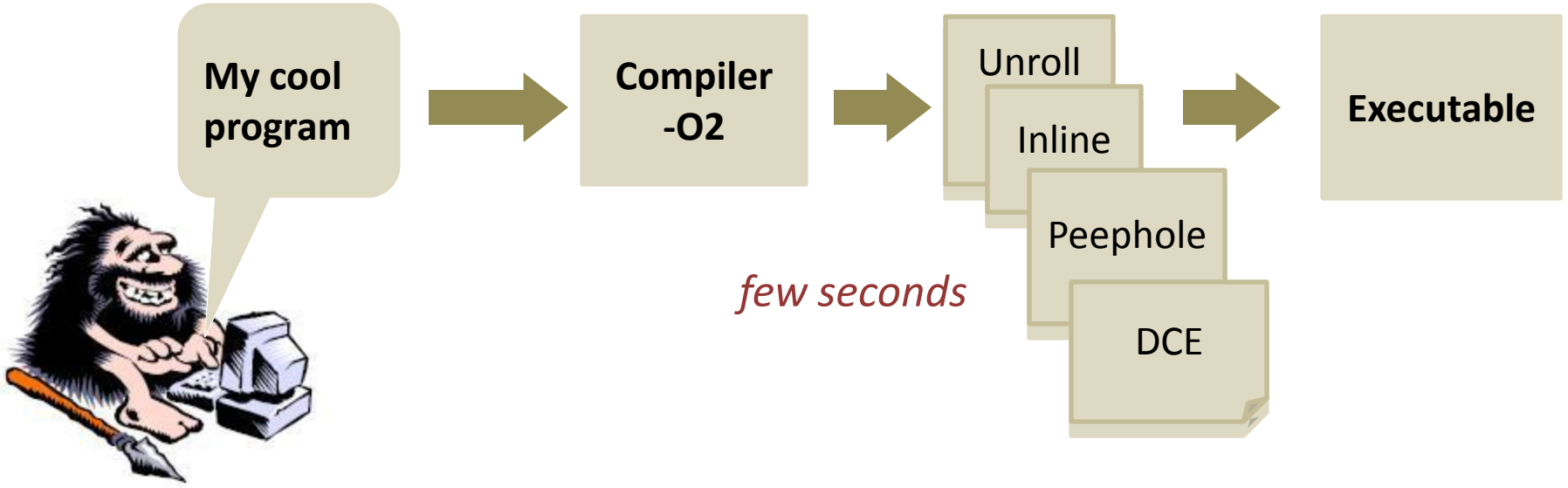
Efficient Program Compilation through Machine Learning Techniques

Gennady Pekhimenko
IBM Canada

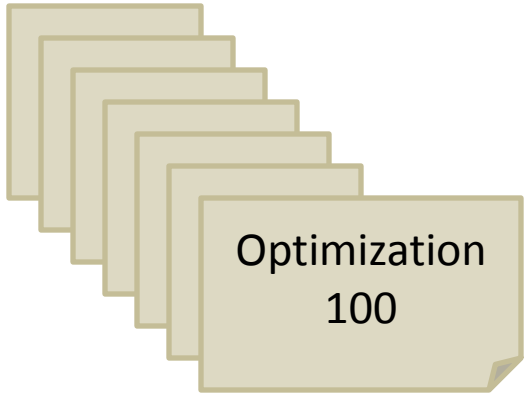
Angela Demke Brown
University of Toronto

Motivation



But what to do if executable is slow?

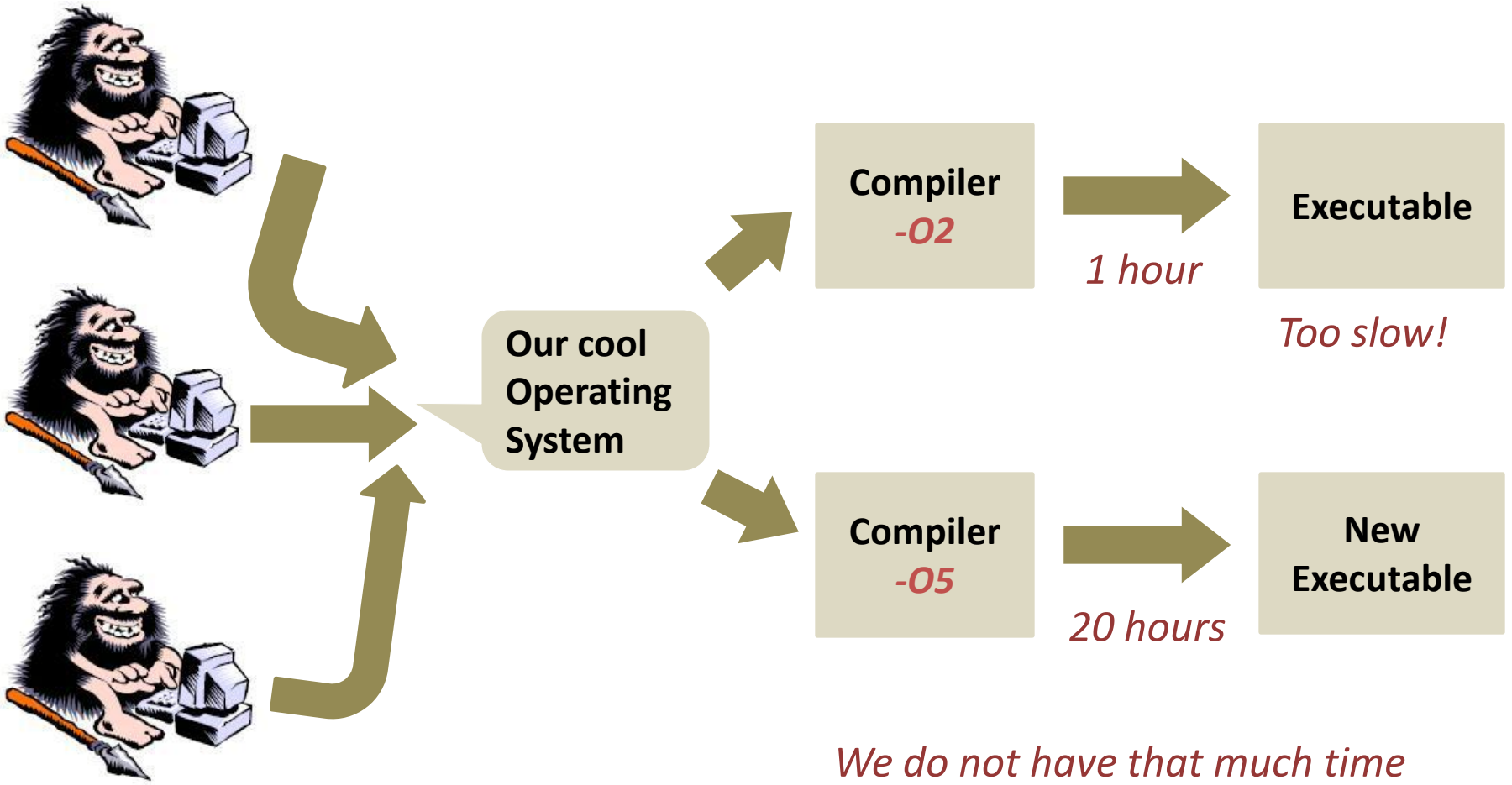
Replace *–O2* with *–O5*



New Fast Executable

1-10 minutes

Motivation (2)



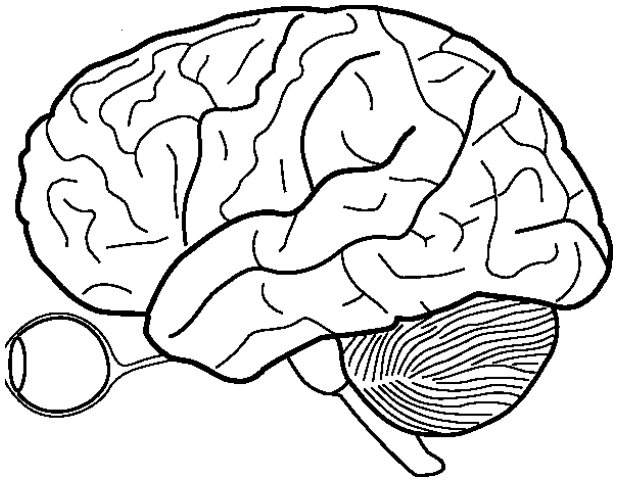
Why did it happen?

Basic Idea

Do we need all these optimizations for every function?

Probably not.

Compiler writers can typically solve this problem*, but how ?*



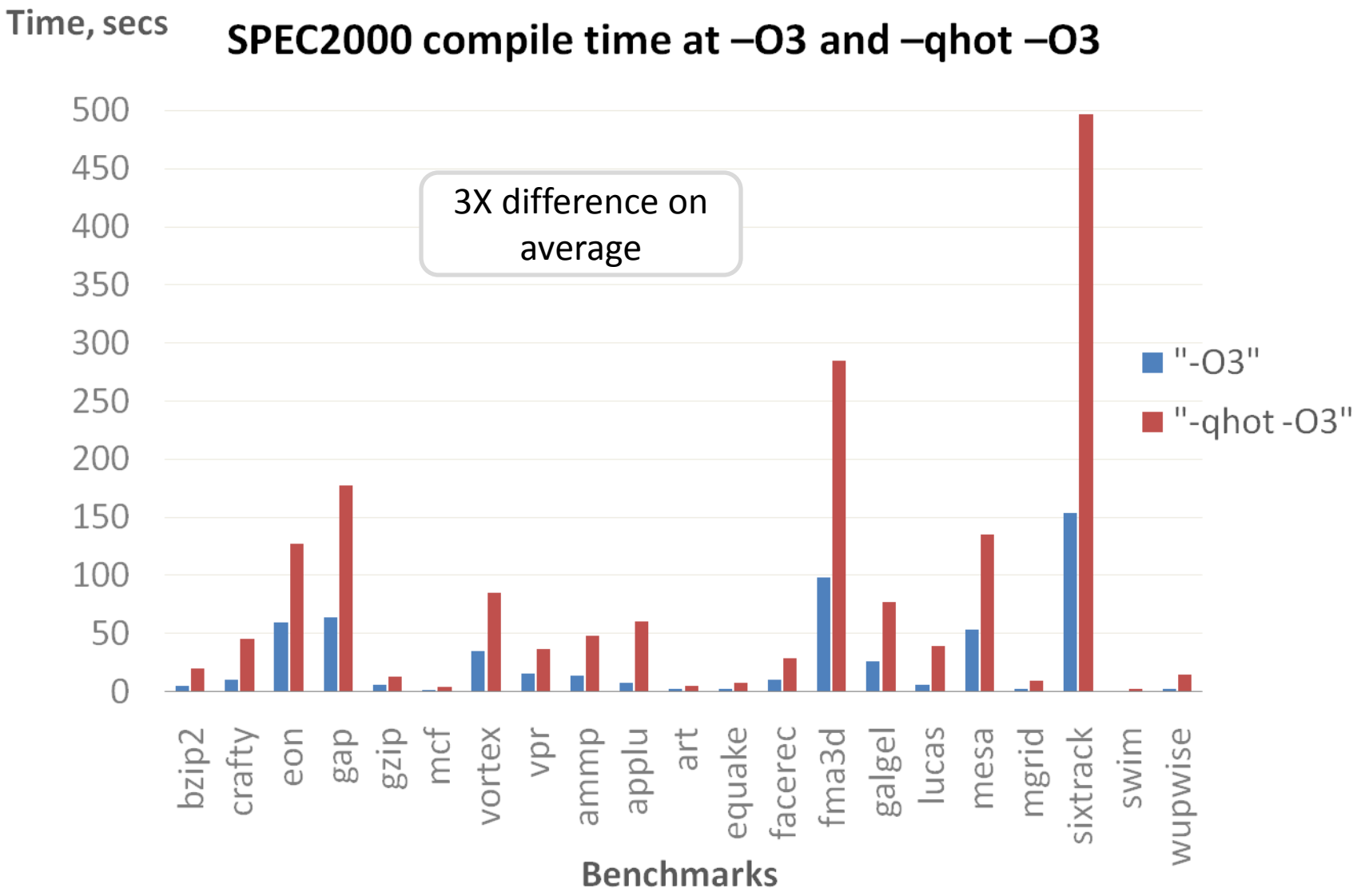
1. ***Description*** of every function
2. **Classification** based on the description
3. **Only certain** optimizations for every class

Machine Learning is good for solving this kind of problems

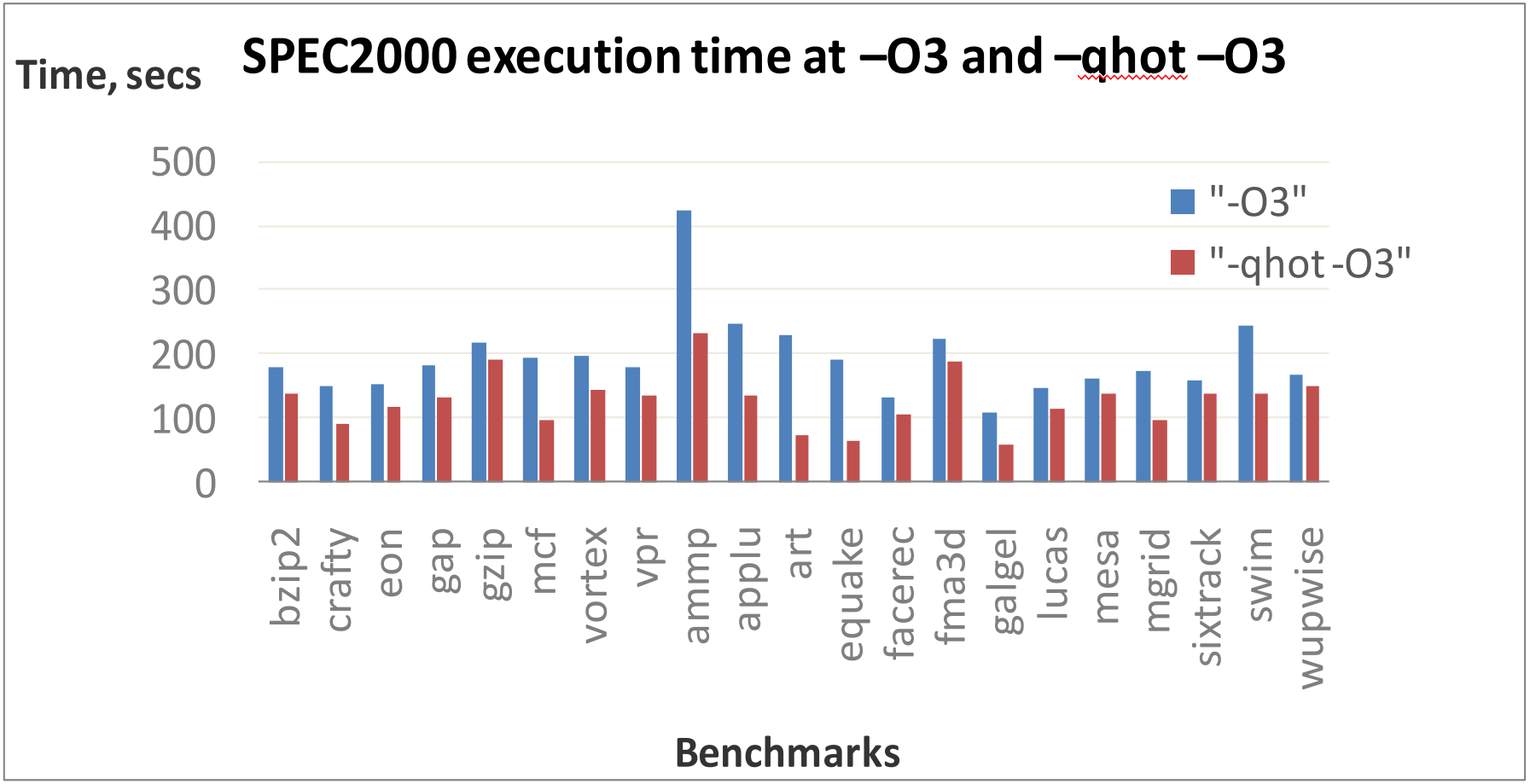
Overview

- Motivation
- System Overview
- Experiments and Results
- Related Work
- Conclusions
- Future Work

Initial Experiment



Initial Experiment (2)

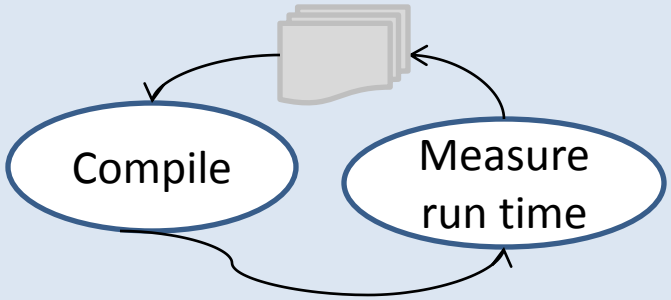


Our System

Prepare

- extract features
- modify heuristic values
- choose transformations
- find hot methods

Gather Training Data

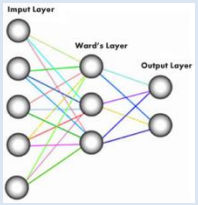


Best feature settings

Online | Offline

Learn

Logistic Regression Classifier



Classification parameters

Deploy

TPO/XL Compiler
set heuristic values

Data Preparation

Three key elements:

- Feature extraction
- Heuristic values modification
- Target set of transformations

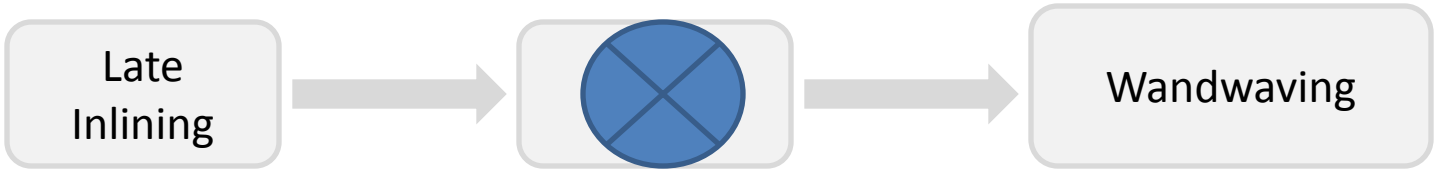
• Existing XL compiler is missing functionality
• Extension was made to the existing Heuristic Context approach

• Total # of insts
• Loop nest level
• # and % of Loads, Stores, Branches
• Loop characteristics
• Float and Integer # and %

• Unroll
• Wandwaving
• If-conversion
• Unswitching
• CSE
• Index Splitting ….

Gather Training Data

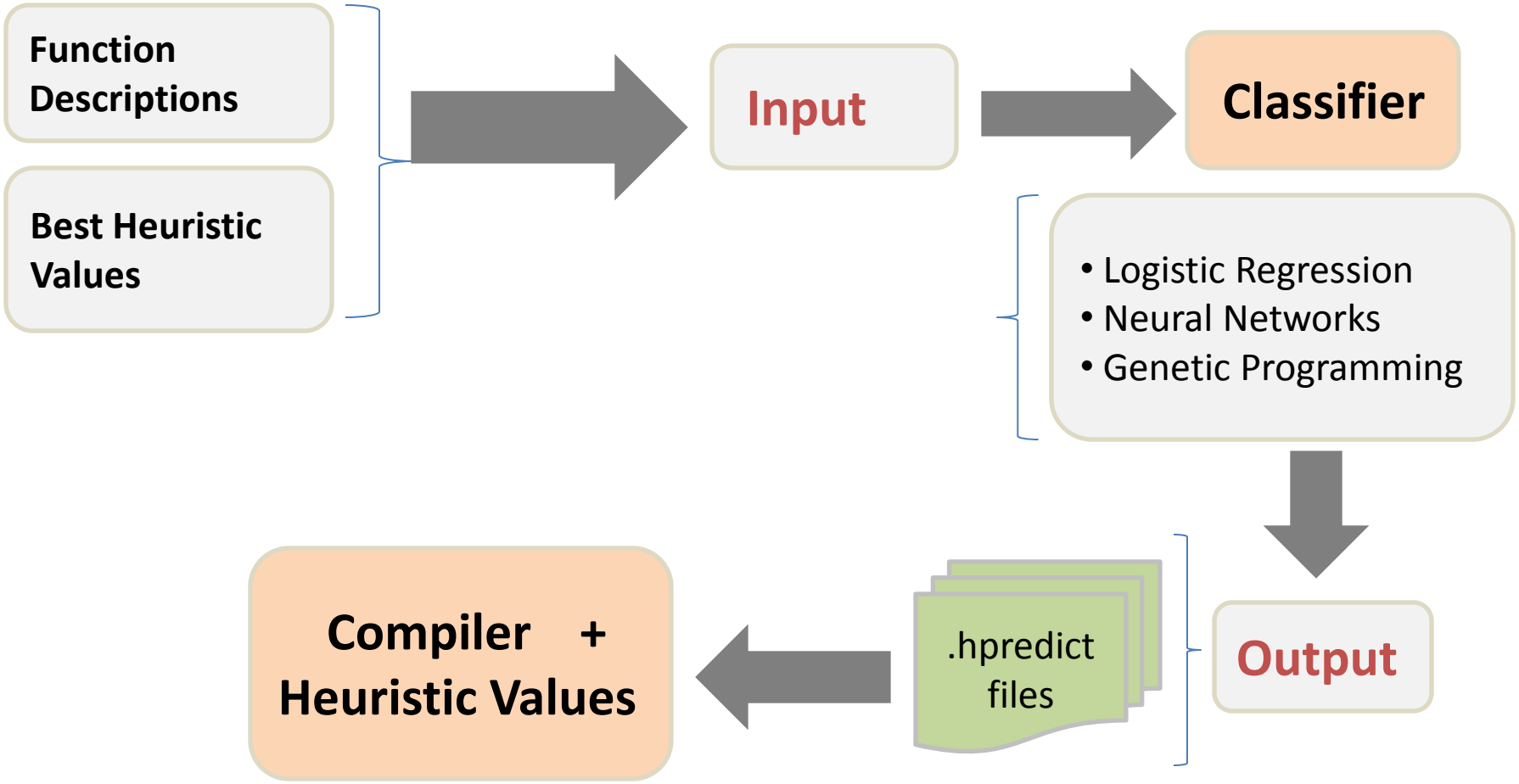
- Try to “cut” transformation backwards (from last to first)



- If run time not worse than before, transformation can be skipped
- Otherwise we keep it
- We do this for every hot function of every test

The main benefit is *linear* complexity.

Learn with Logistic Regression



Deployment

Online phase, for every function:

- Calculate the *feature vector*
- Compute the *prediction*
- Use this prediction as *heuristic context*

Overhead is negligible

Overview

- Motivation
- System Overview
- **Experiments and Results**
- **Related Work**
- **Conclusions**
- **Future Work**

Experiments

Benchmarks:

SPEC2000

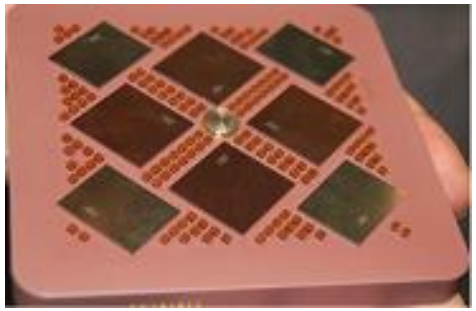
Others from IBM customers

Platform:

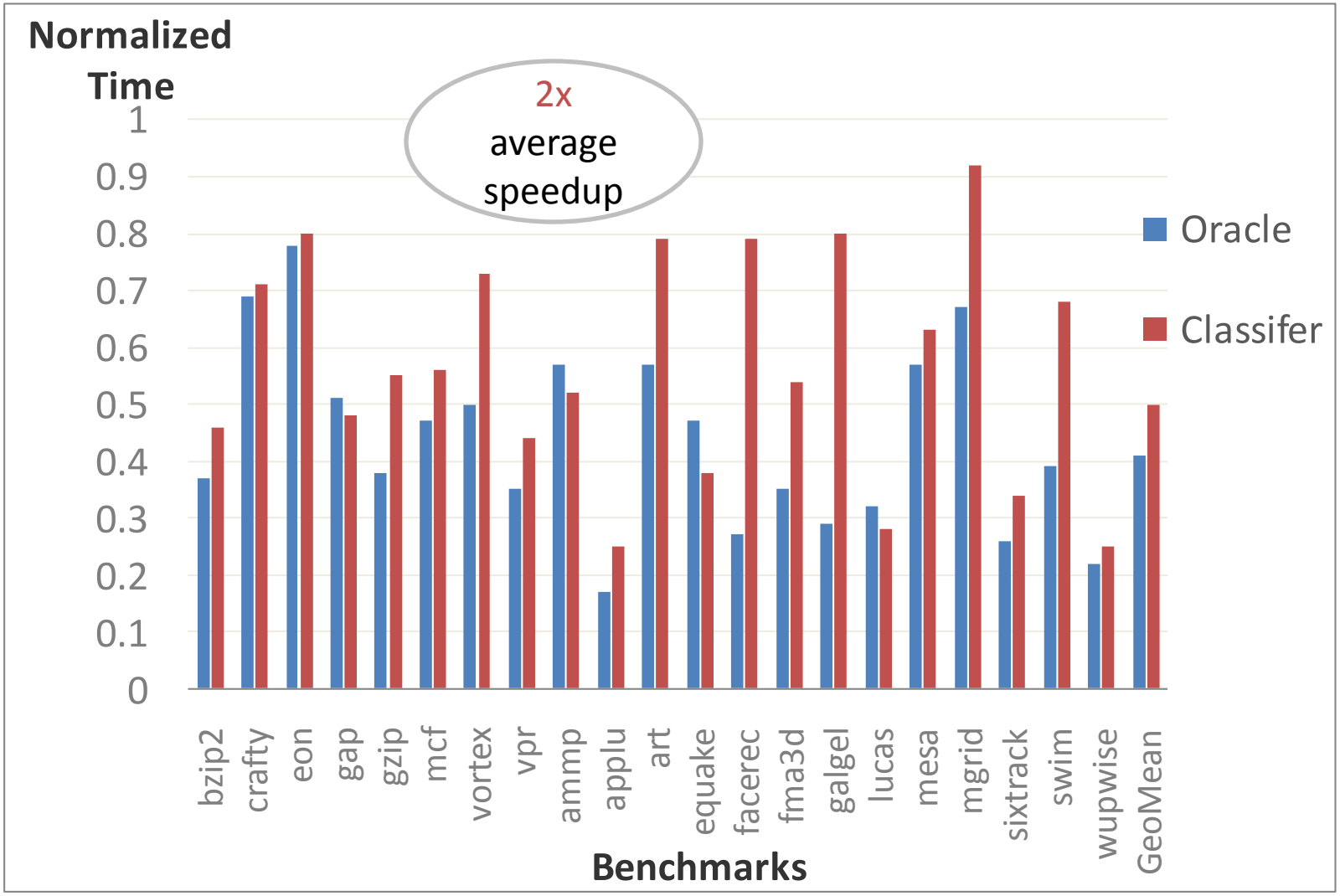
IBM server, 4 x Power5

1.9 GHz, 32GB RAM

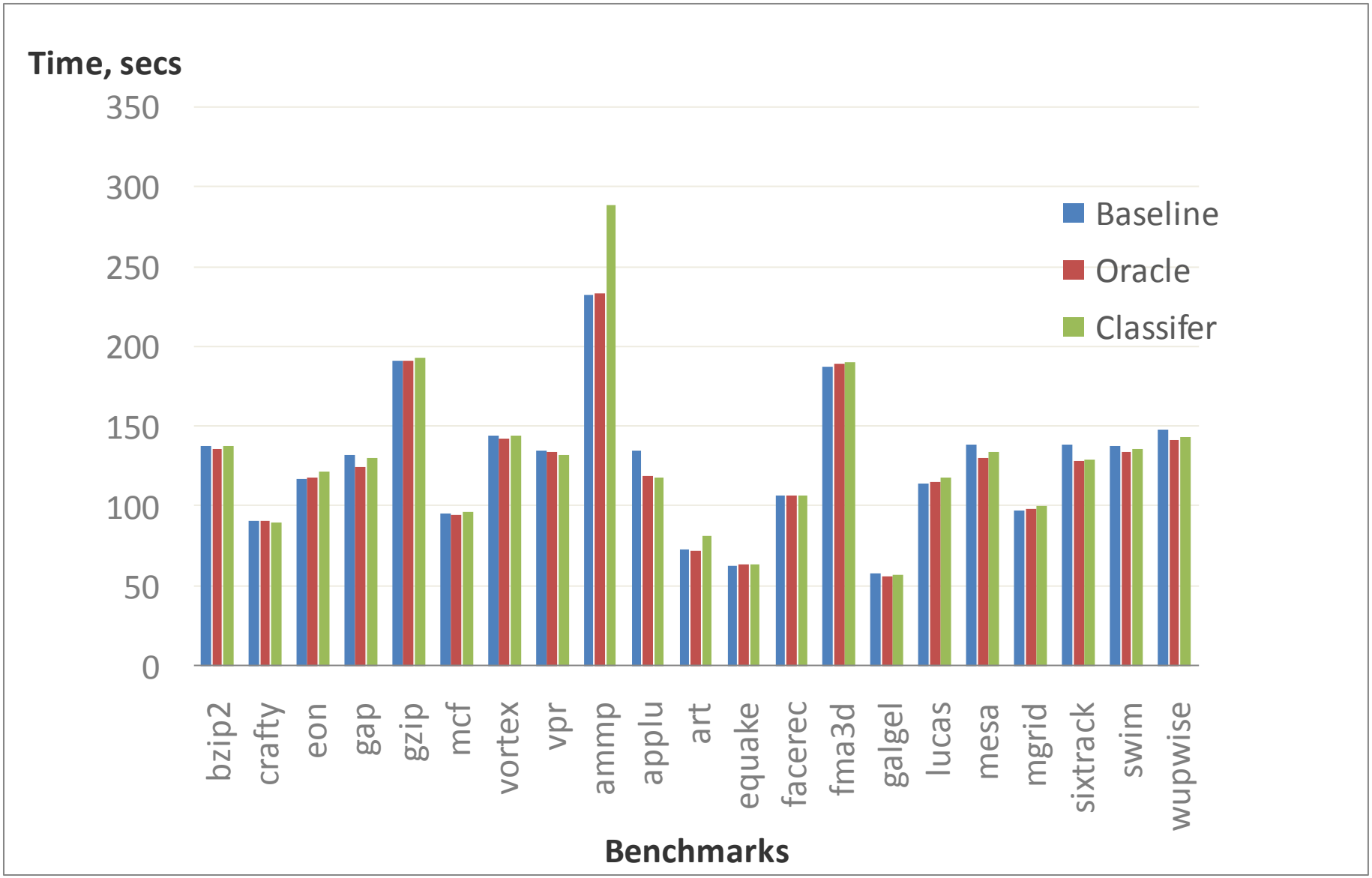
Running AIX 5.3



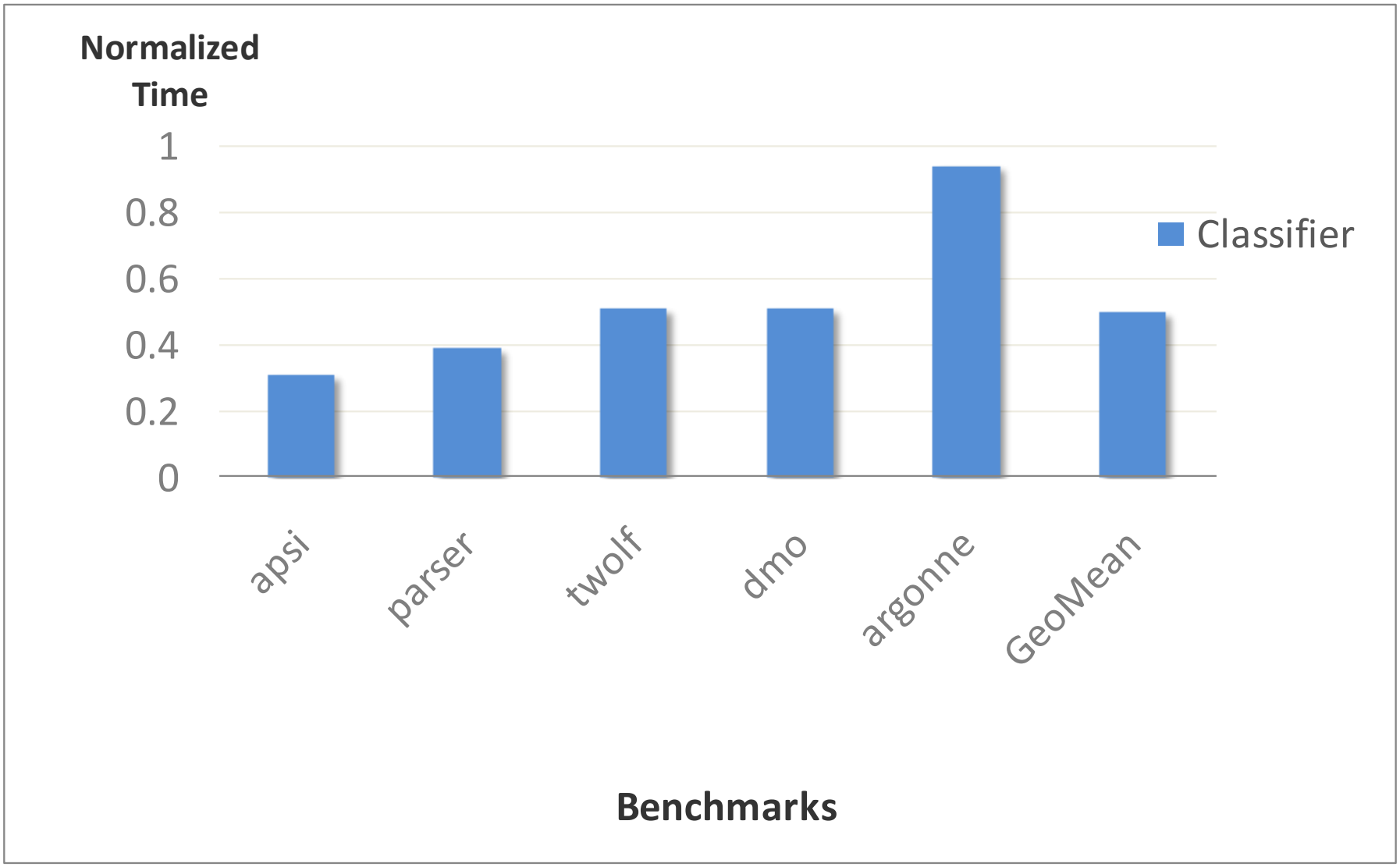
Results: compilation time



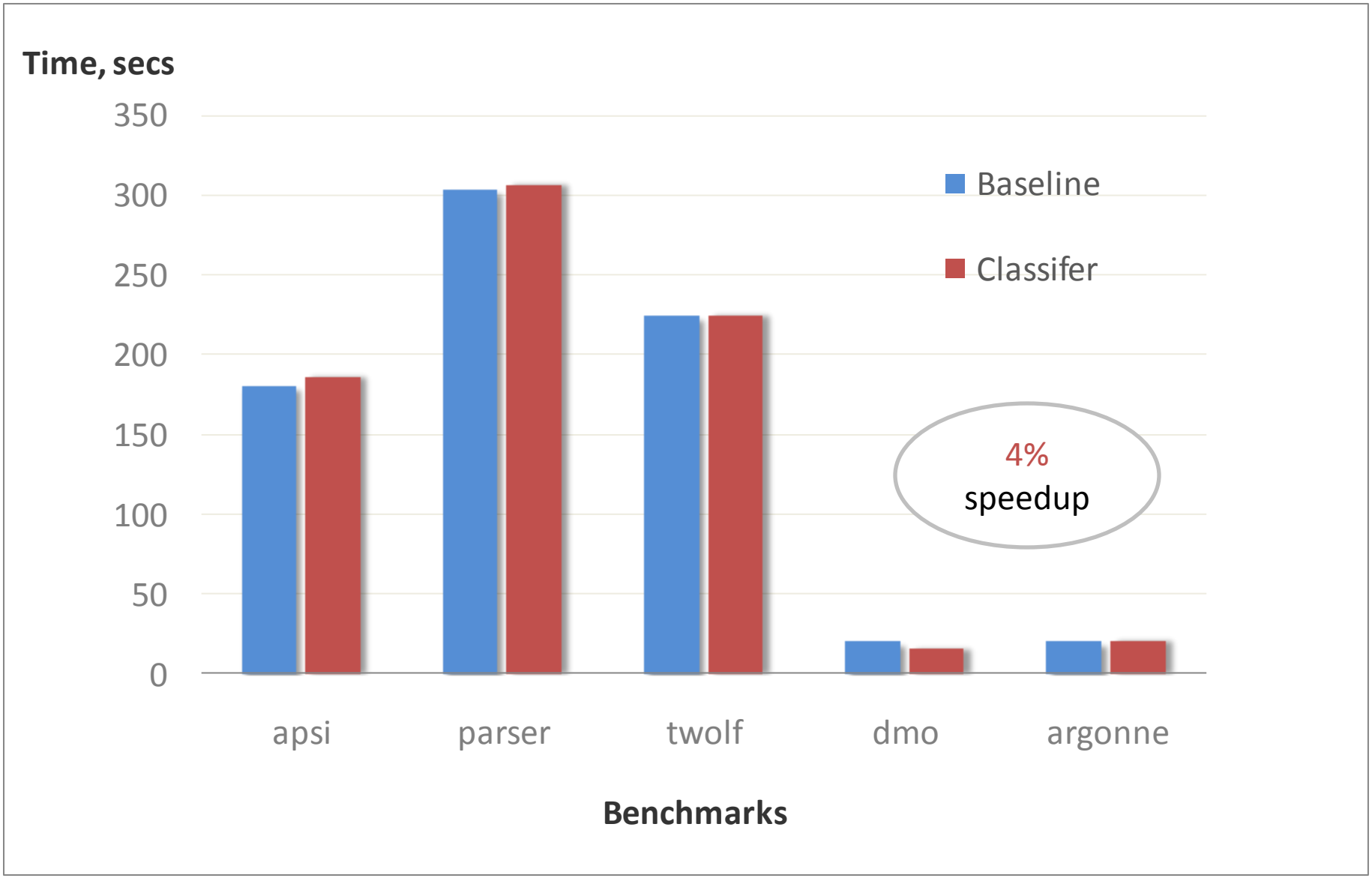
Results: execution time



New benchmarks: compilation time



New benchmarks: execution time



Overview

- Motivation
- System Overview
- Experiments and Results
- **Related Work**
- **Conclusions**
- **Future Work**

Related Work

- **Iterative Compilation**
  - Pan and Eigenmann
  - Agakov, et al.
- **Single Heuristic Tuning**
  - Calder, et al.
  - Stephenson, et al.
- **Multiple Heuristic Tuning**
  - Cavazos, et al.
  - MILEPOST GCC

Conclusions and Future Work

- *2x average* compile time decrease
- Future work
  - Execution time improvement
  - -O5 level
  - Performance Counters for better method description
- Other benefits
  - Heuristic Context Infrastructure
  - Bug Finding

Thank you

- Raul Silvera, Arie Tal, Greg Steffan, Mathew Zaleski

- Questions?