Learning Compilers for Configurable Processors

Hugh Leather
University of Edinburgh

Overview

- Members of the Milepost group
- Introduction to machine learning in compilers
- Learning compiler flags for an embedded processor
- Learning compilers over multiple achitectures
- List of open source software in the project
- Pros/cons of open source software

Milepost Members

Funded by:
Information Society Technologies (IST) of the European Union under 6th FWP

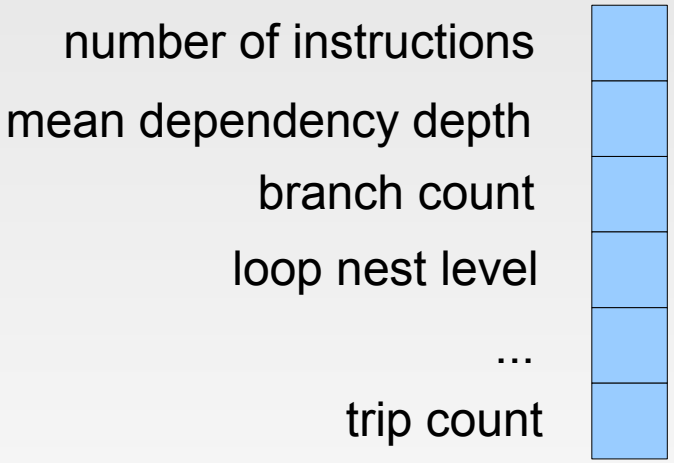
Machine Learning in Compilers

- Writing compiler heuristics is hard
  - Need deep knowledge
  - Change elsewhere needs change everywhere
  - Architectures/libraries/OS change
- Humans can't keep up
  - Heuristics remain unoptimised

- Can machine learning help?

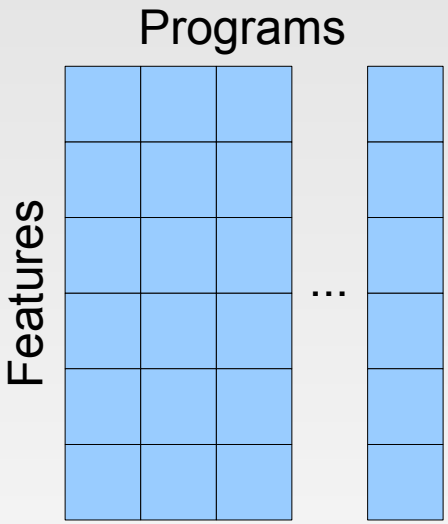
Machine Learning in Compilers

- We start by choosing some information that might be useful to the heuristic (*features*)



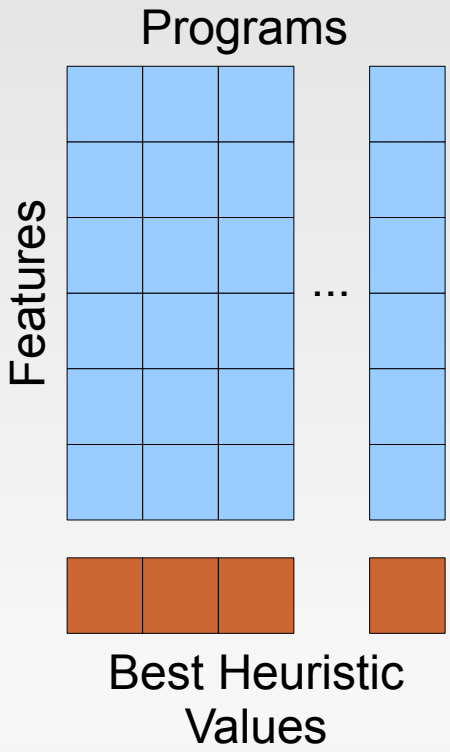
Machine Learning in Compilers

- Collect lot of example benchmarks and compute their features



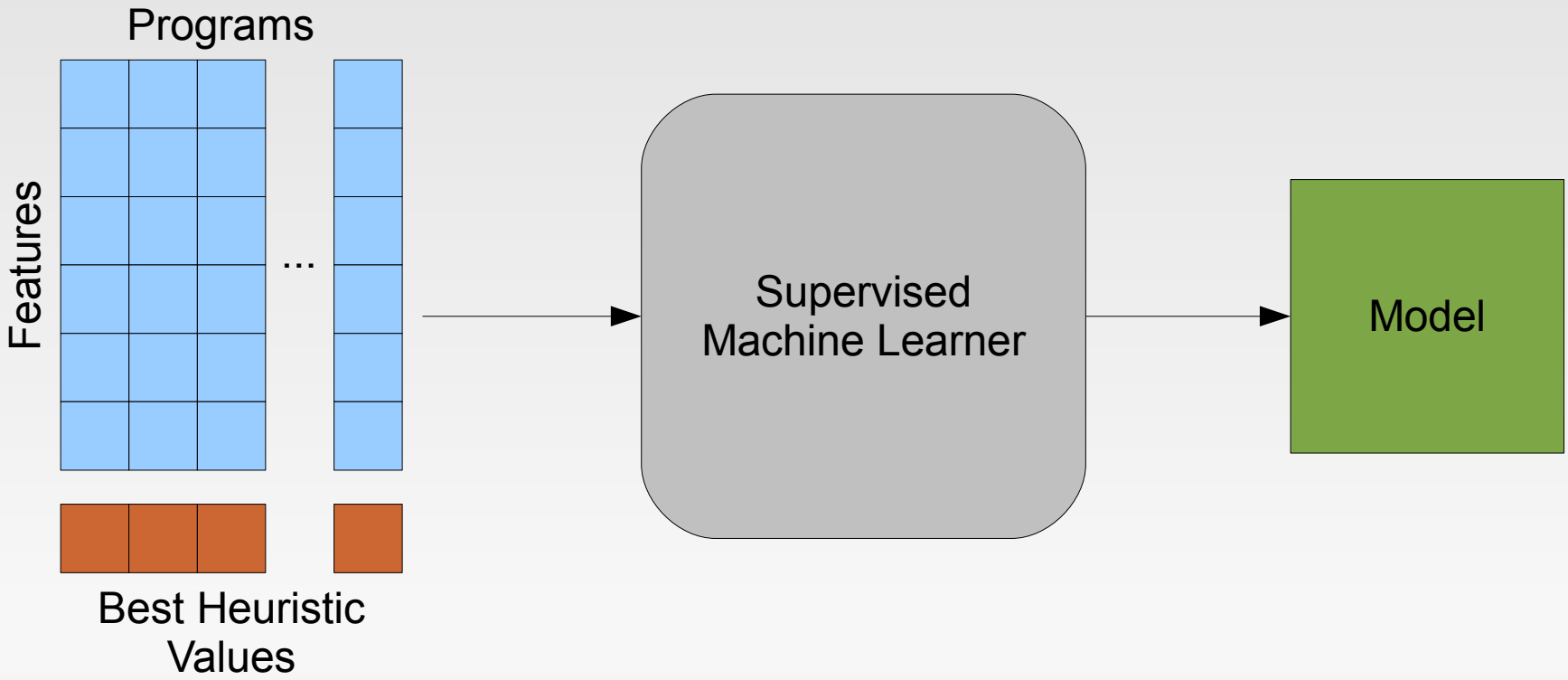
Machine Learning in Compilers

- Execute the programs with different compilation strategies and determine best for each



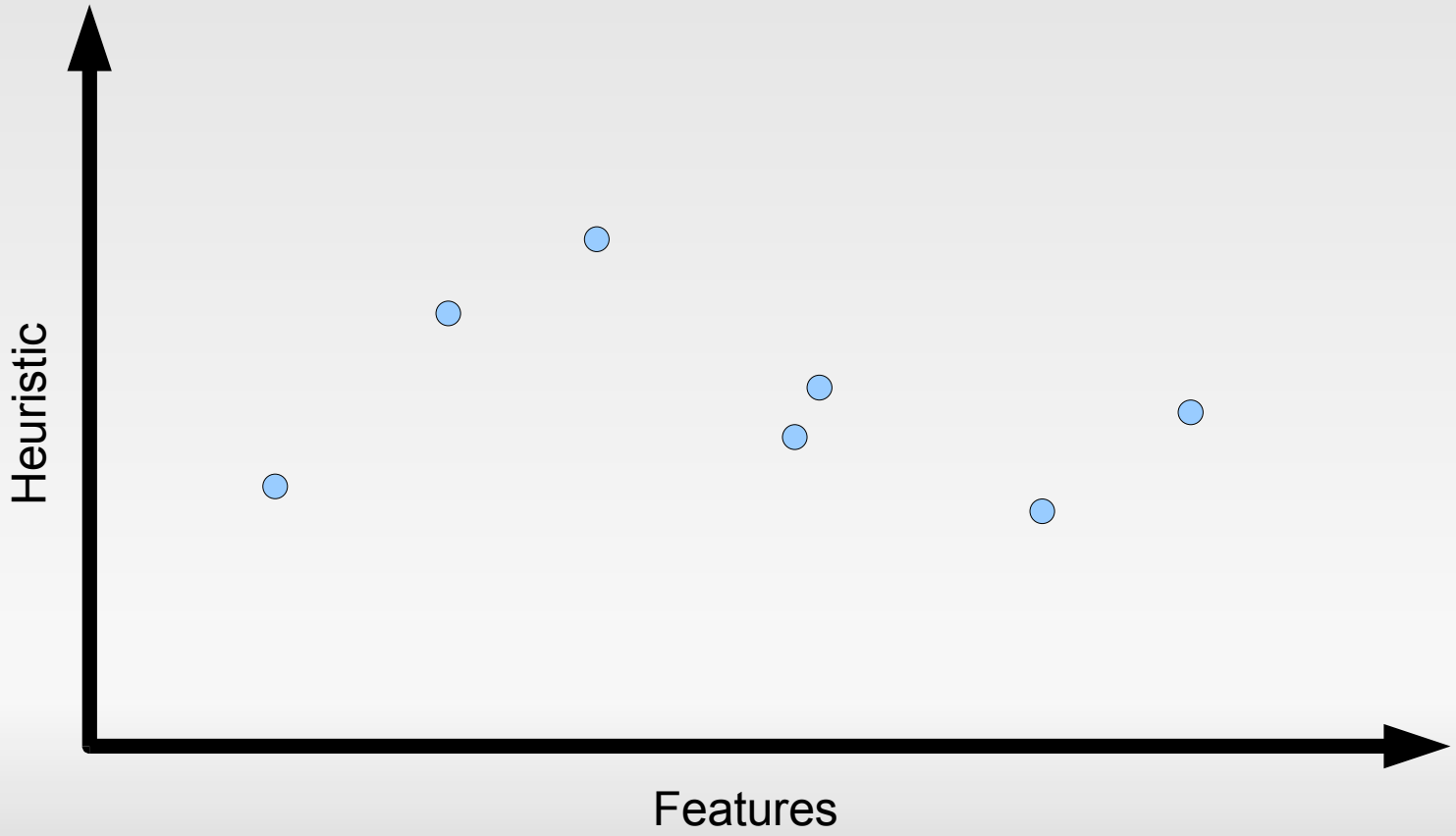
Machine Learning in Compilers

- Pass these to a machine learning tool
- It learns a model



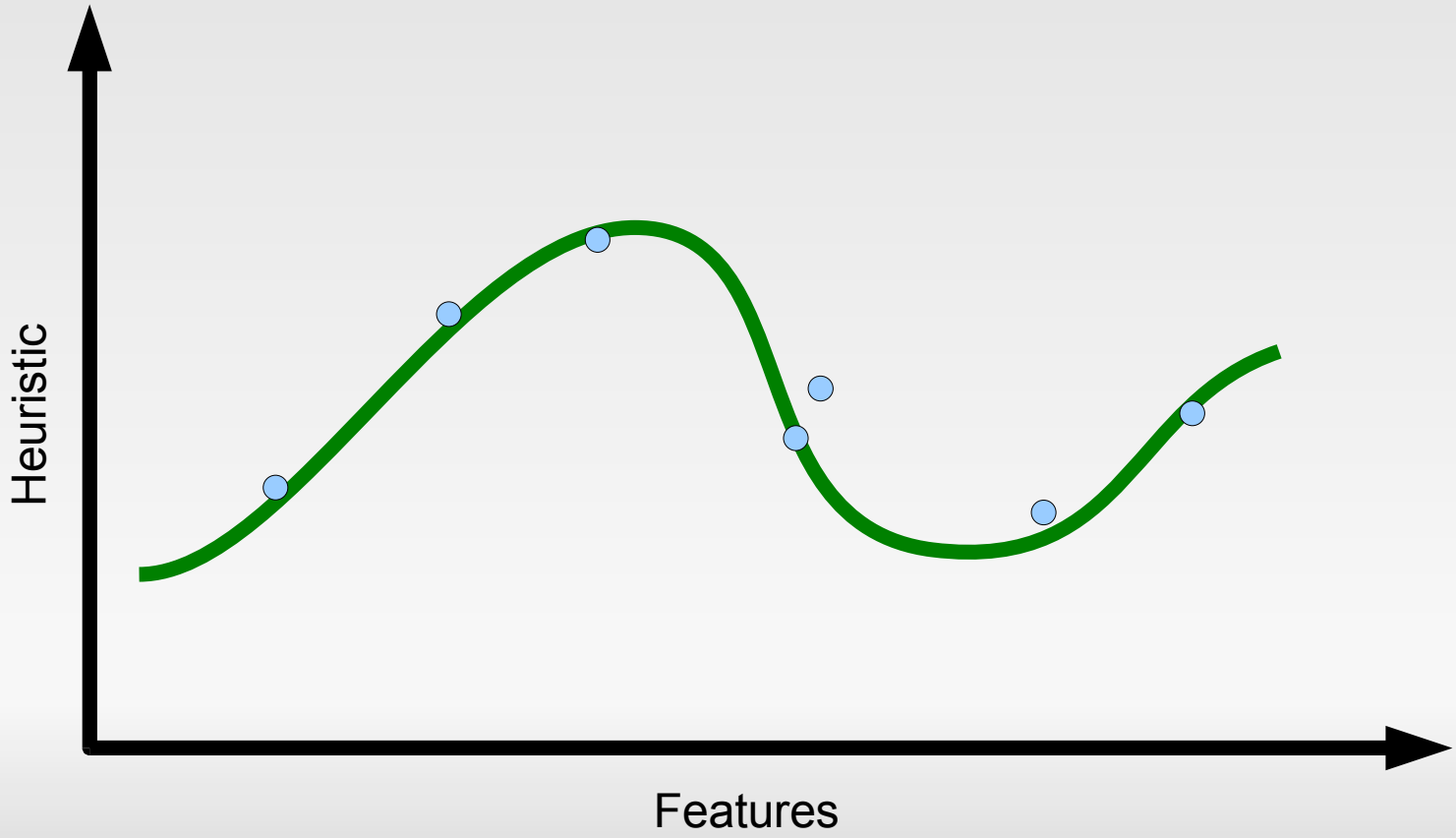
Machine Learning in Compilers

- A model is really just a way of fitting a curve to data



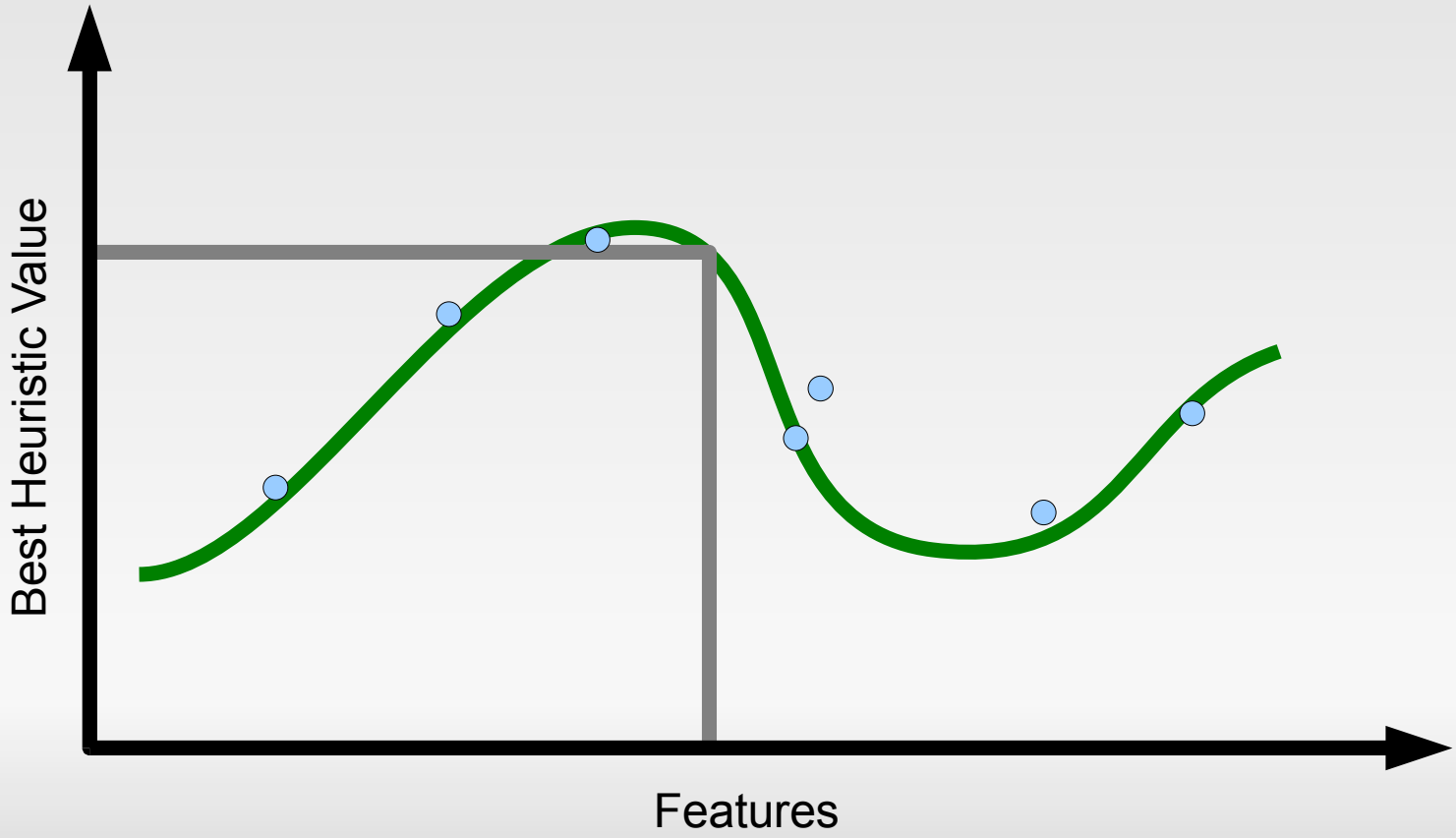
Machine Learning in Compilers

- A model is really just a way of fitting a curve to data



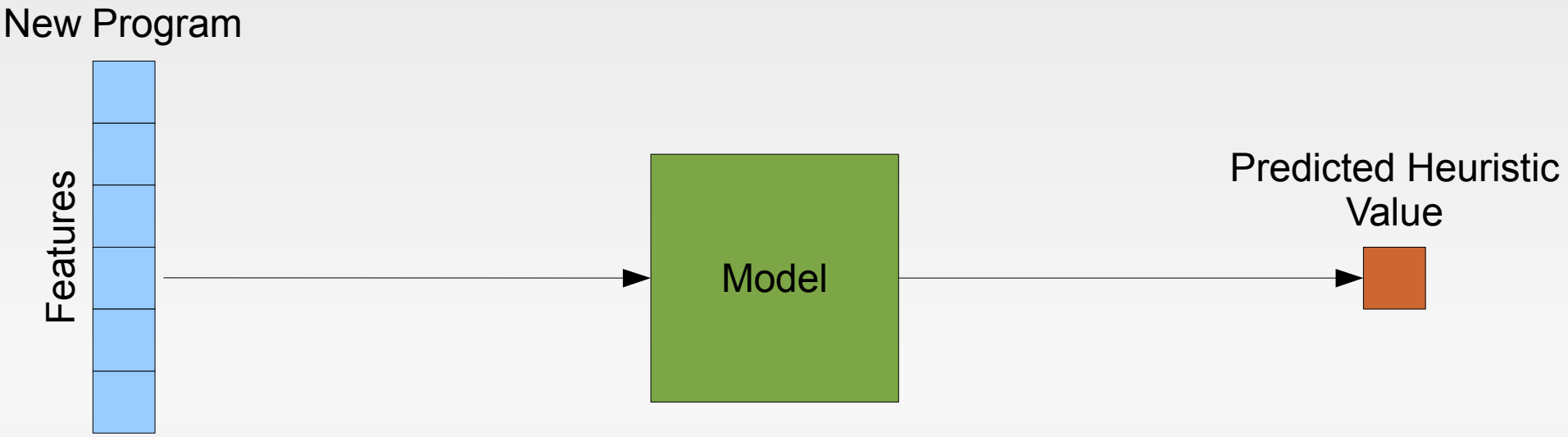
Machine Learning in Compilers

- Gives heuristic for unseen points



Machine Learning in Compilers

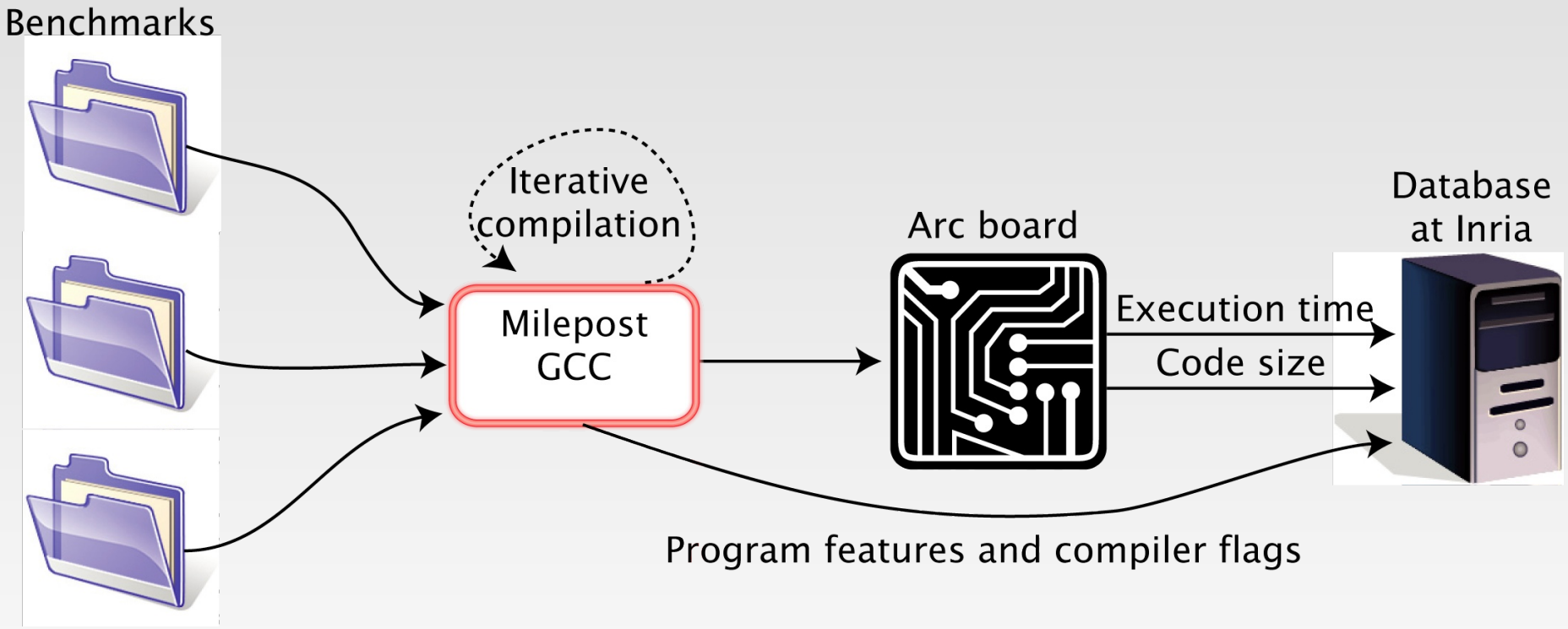
- Insert model into compiler – heuristic replaced
- Automatic
  - No human expert required
  - Can be redone after changes



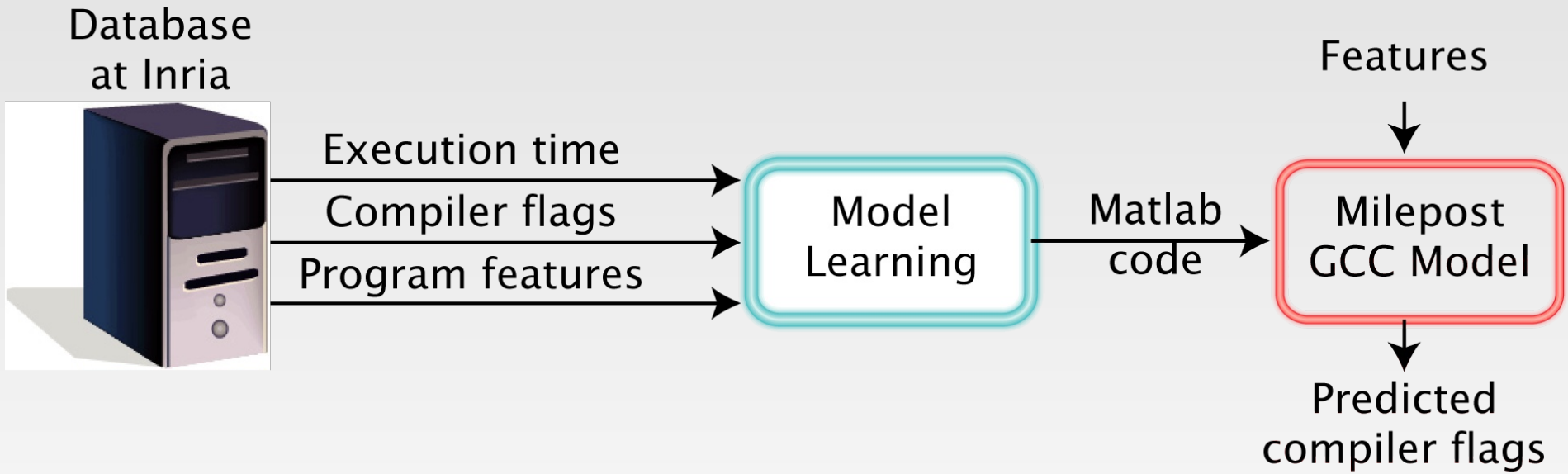
An Experiment

- Learn optimisation flags for MiBench with GCC
- Target the ARC 725 customisable processor
- Trained on
  - 500 random flag settings
  - 5 executions each
  - (2 months runtime on 2 machines)

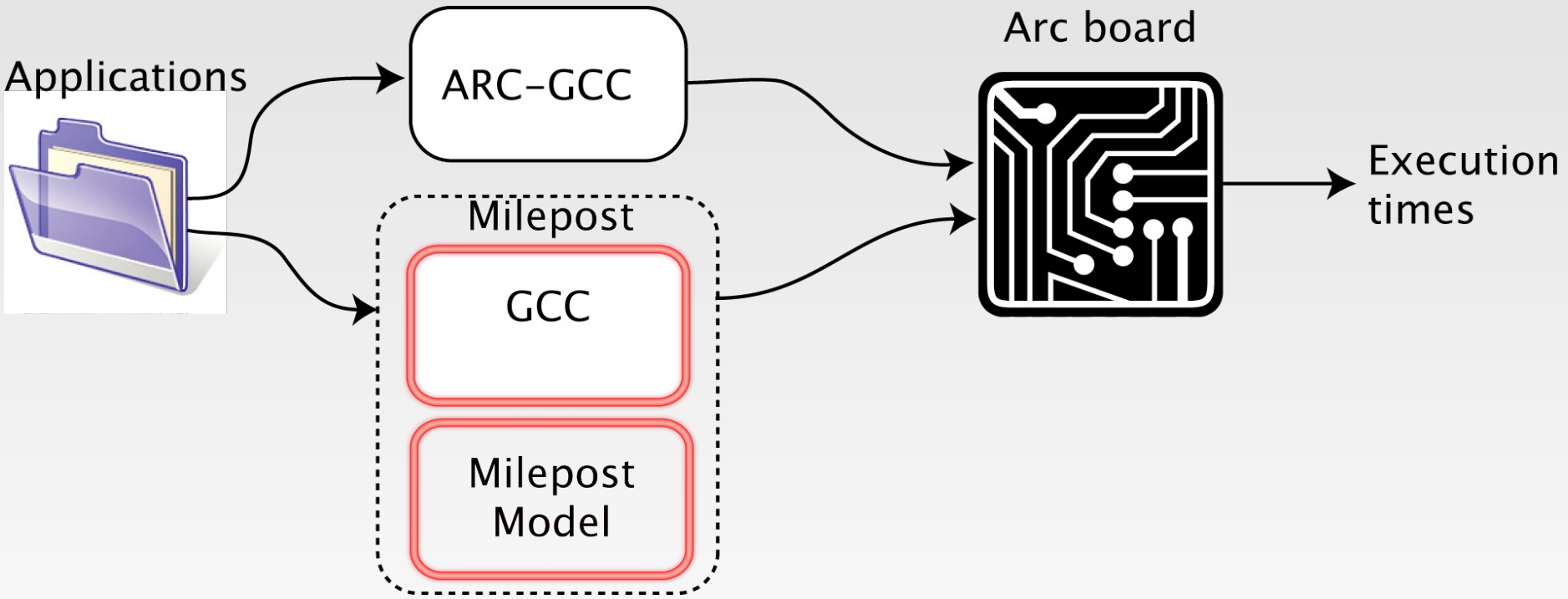
An Experiment
Filling the Database



An Experiment
Building the Model

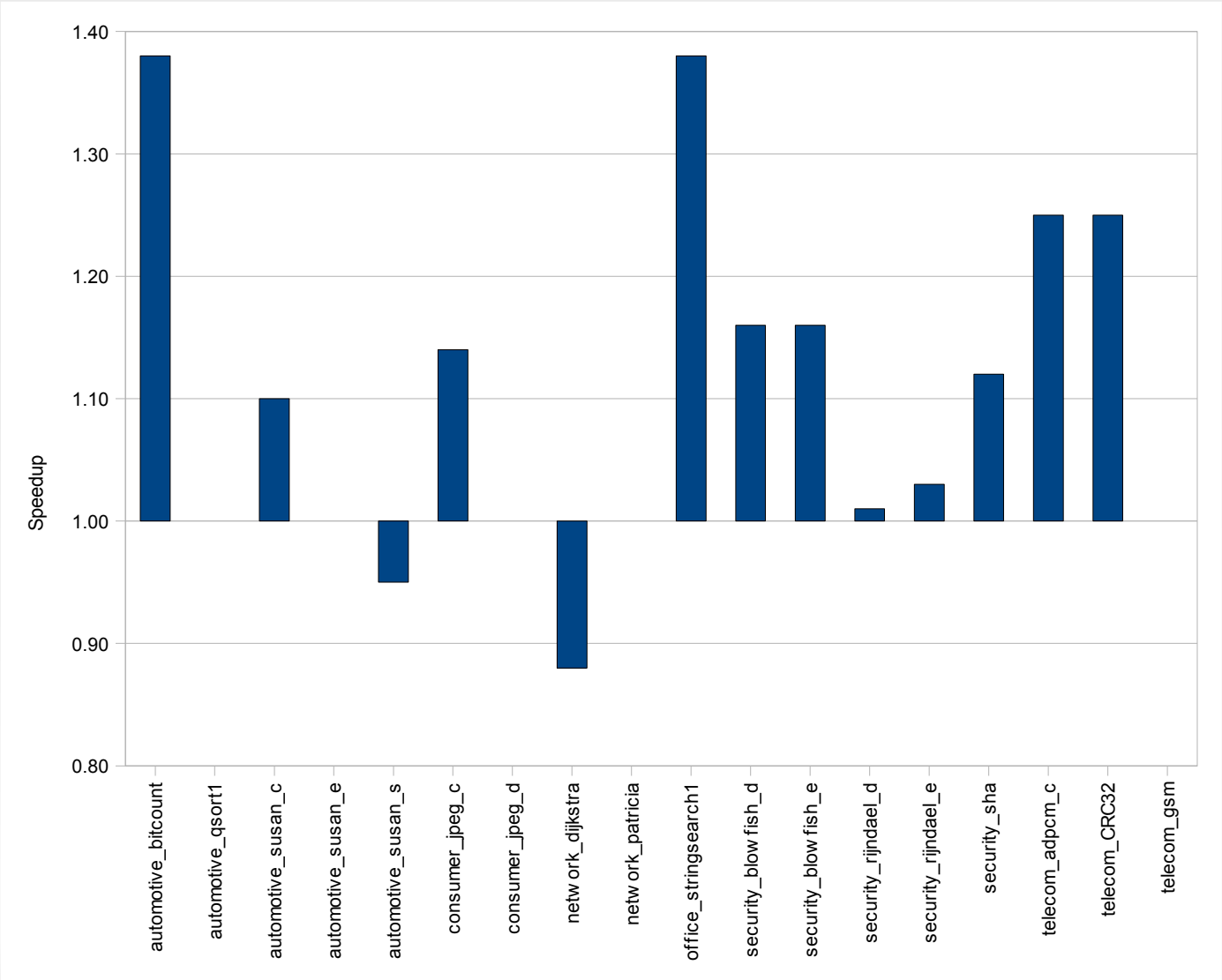


An Experiment Using Improved Compiler



An Experiment Results

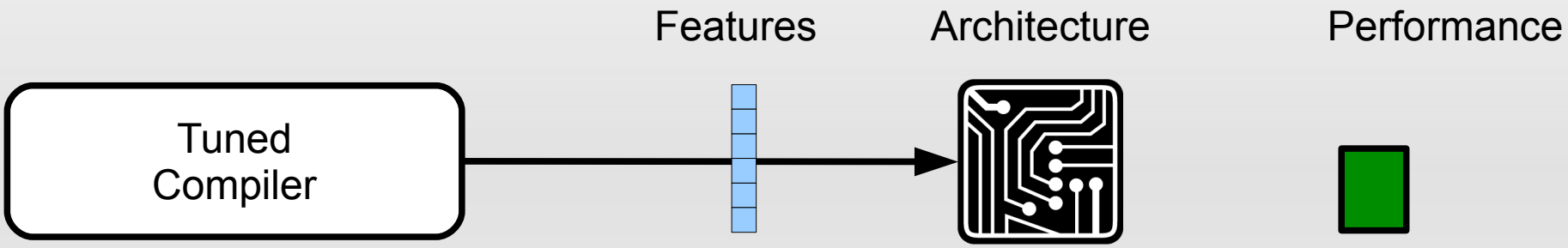
Benchmark	Speedup
automotive_bitcount	1.38
automotive_qsort1	1.00
automotive_susan_c	1.10
automotive_susan_e	1.00
automotive_susan_s	0.95
consumer_jpeg_c	1.14
consumer_jpeg_d	1.00
network_dijkstra	0.88
network_patricia	1.00
office_stringsearch1	1.38
security_blowfish_d	1.16
security_blowfish_e	1.16
security_rijndael_d	1.01
security_rijndael_e	1.03
security_sha	1.12
telecom_adpcm_c	1.25
telecom_CRC32	1.25
telecom_gsm	1.00
Average	**1.10**



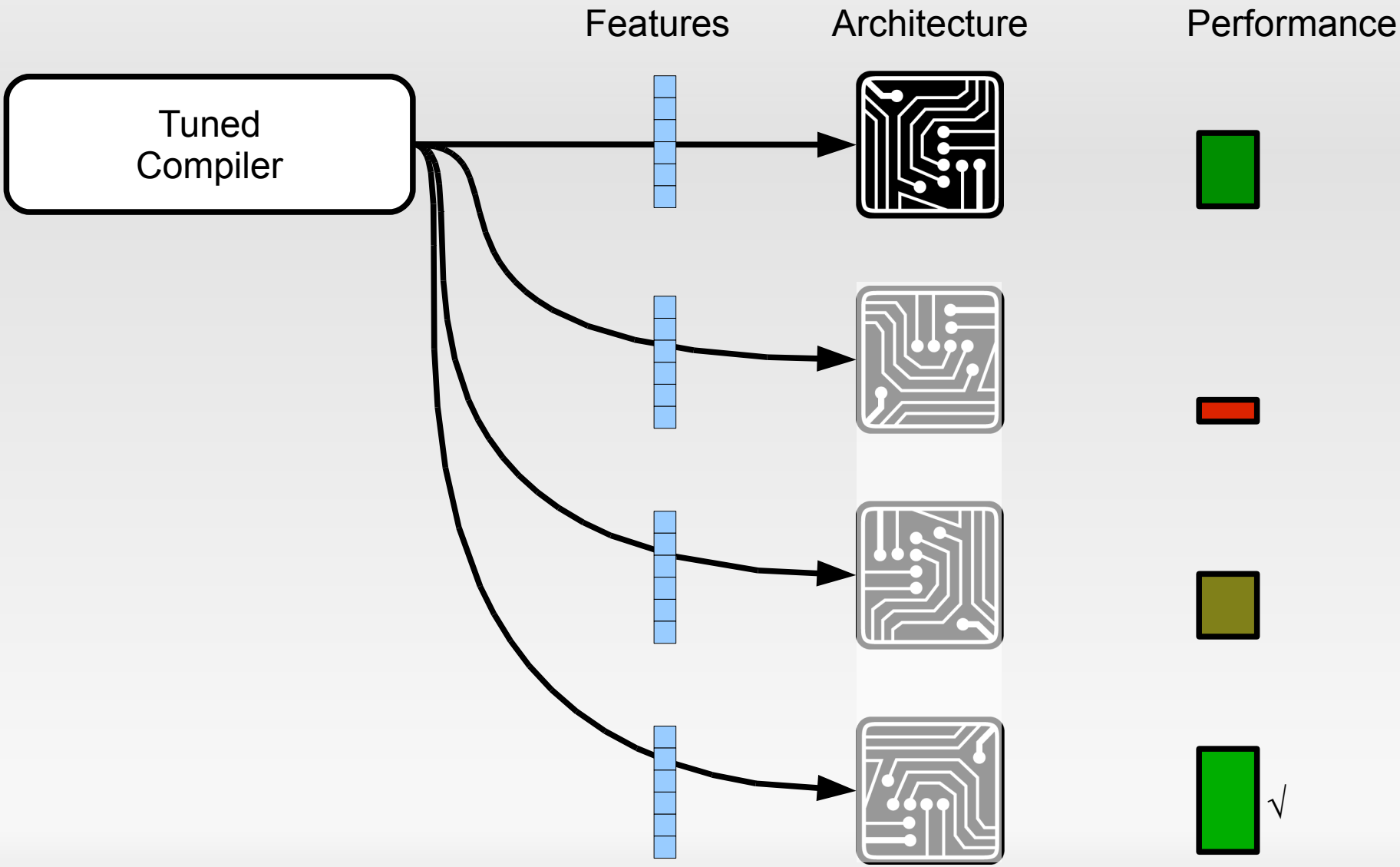
Co-Design Exploration

- Designing an embedded system for some application
- Create many architectures in a 'family'
- Choose the most efficient

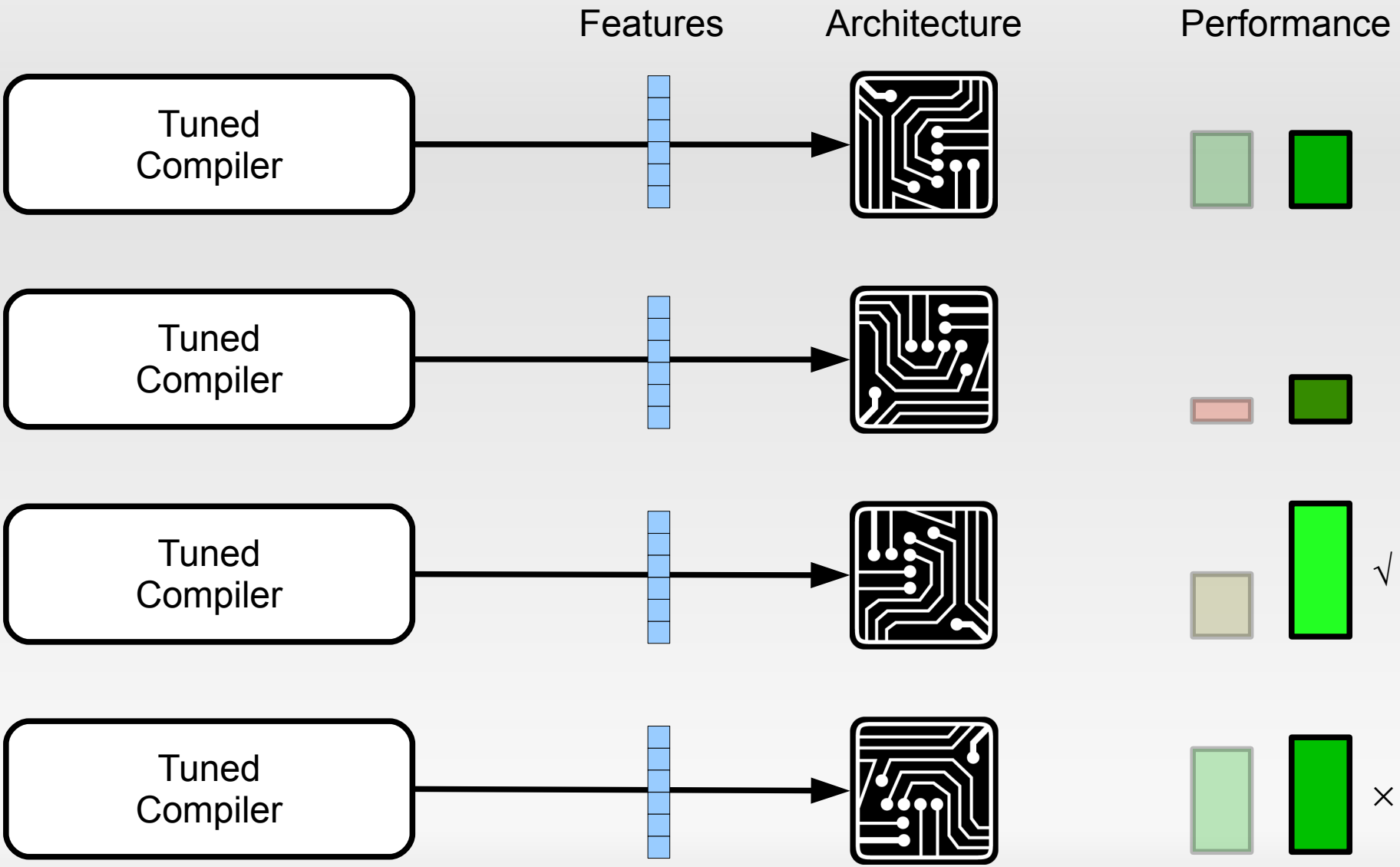
Co-Design Exploration



Co-Design Exploration



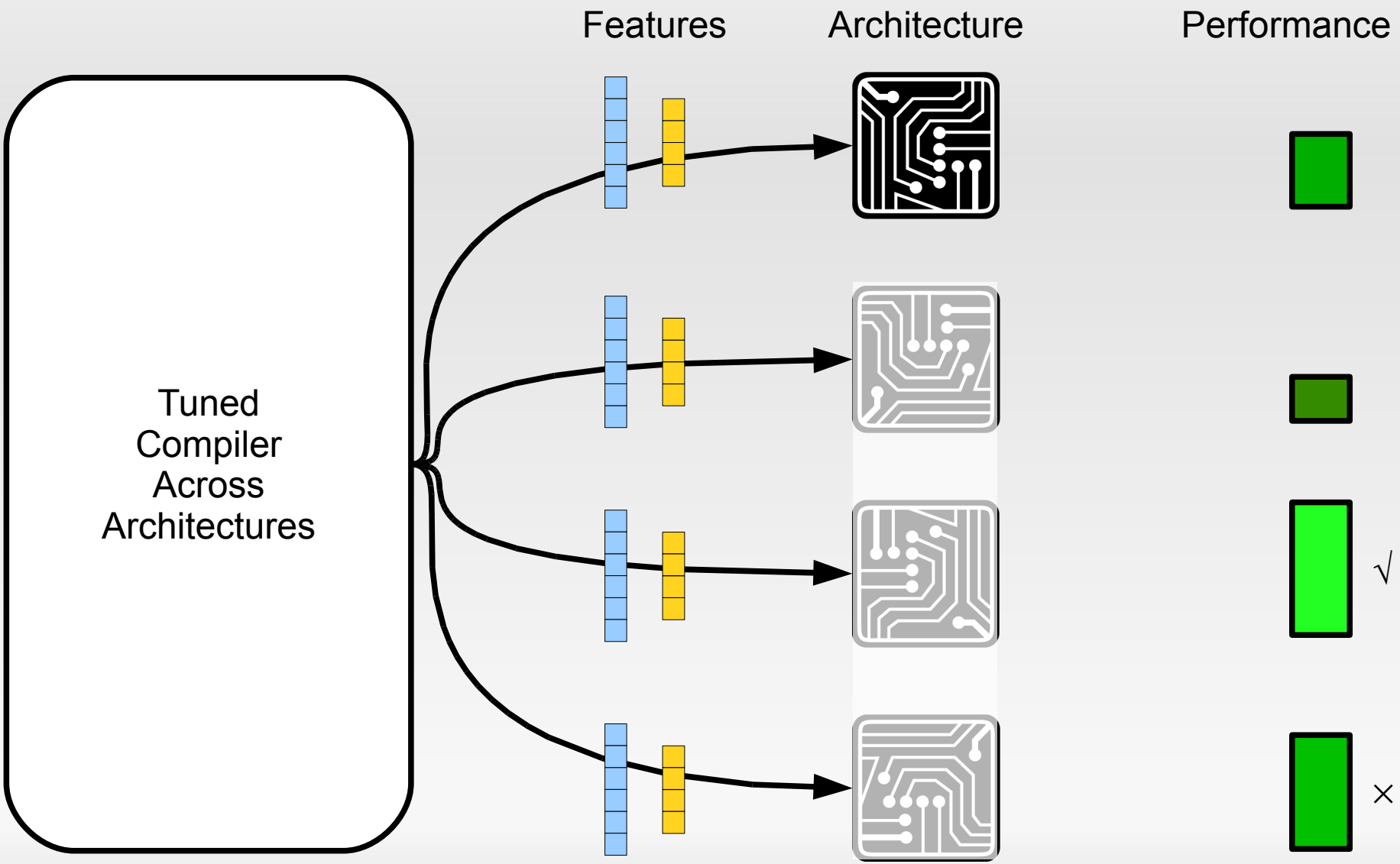
Co-Design Exploration



Co-Design Exploration

- Add some architectural features
  - Cache sizes
  - Number of registers
  - Number & type of FUs
- Incorporate into machine learning for heuristics
- Automatically learned a tuned compiler for a new architecture!

Co-Design Exploration



Open Source Work

- Must have extensible compiler
  - Alter heuristics
  - Gather features
  - Push button compiler tuning
  - Work on configurable processors

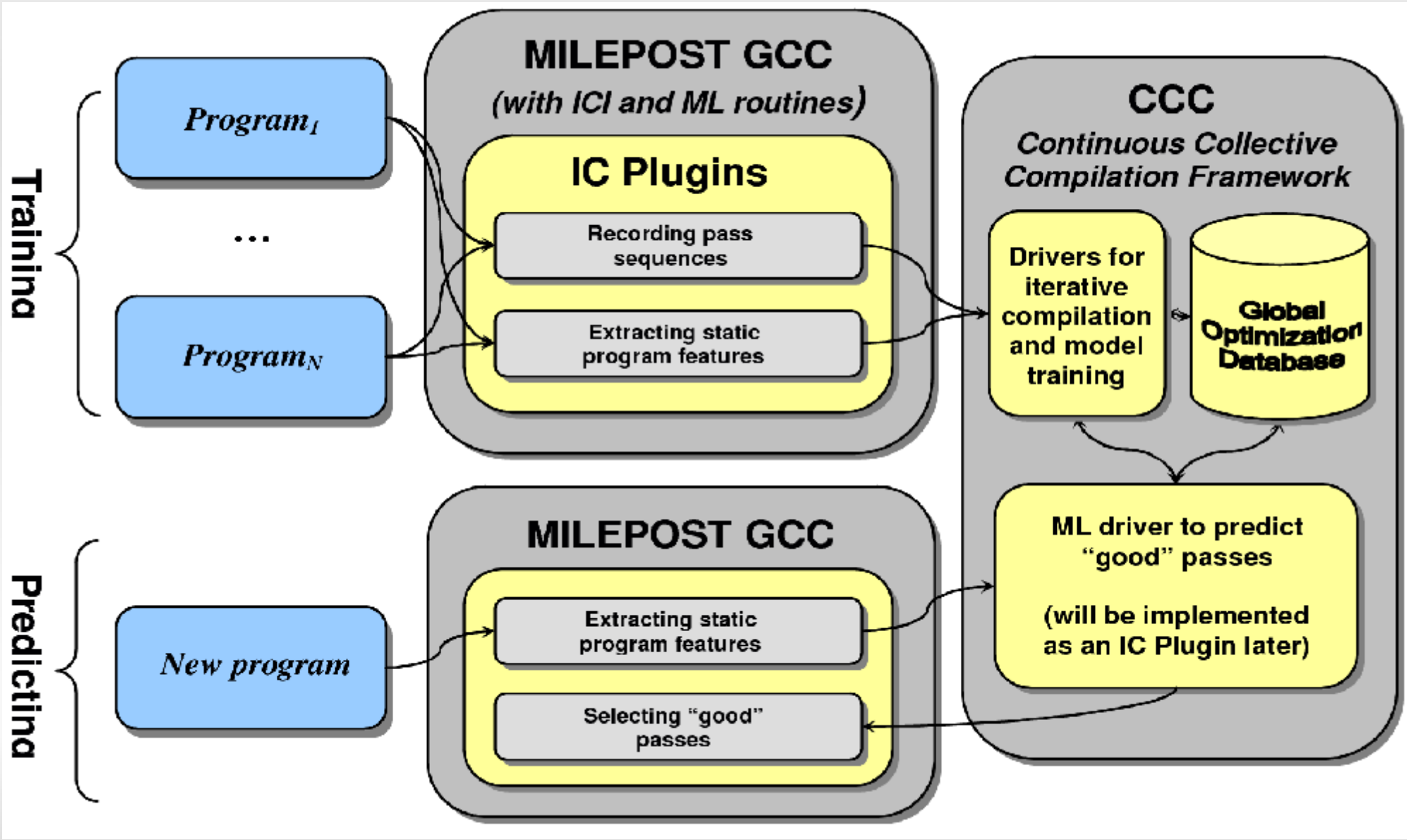
Open Source Work

- MilePost GCC (milepost branch in GCC SVN)
  - Extensibility through shared libraries
  - Extensions to control heuristics
  - Extensions to compute features
  - Provides an 'Iterative Compilation Interface'

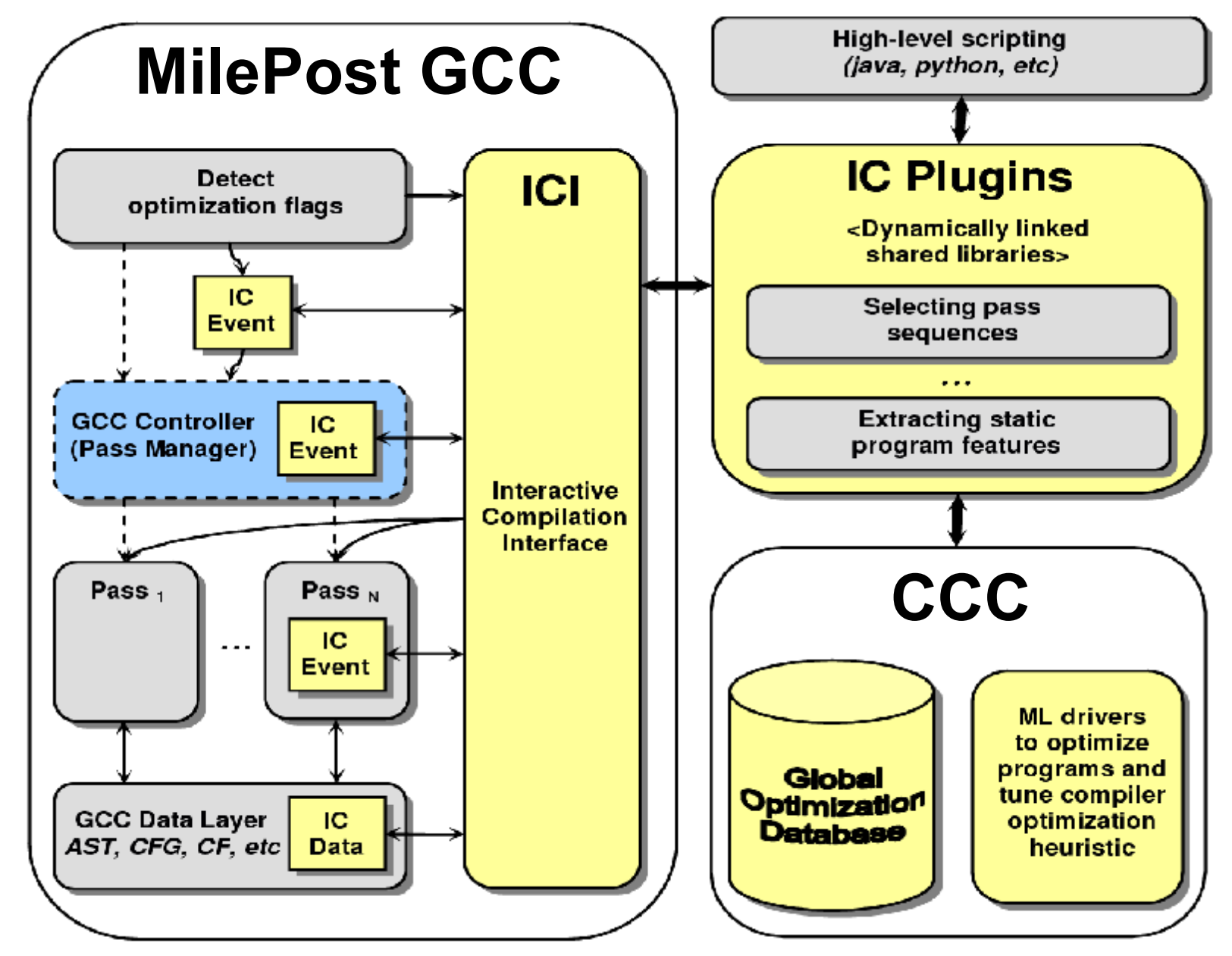
Open Source Work

- Continuous Collective Compilation Framework
  (http://sourceforge.net/projects/cccpf)
  - Driver framework – automatically tunes compiler
  - Runs iterative compilations
    - Benchmarks and settings included
  - Learns heuristics
    - Heuristics predefined
  - Uses MilePost GCC

Open Source Work



Open Source Work



Open Source Work

- ARC tool chain (http://www.arc.com/software/development/gnutools.html)
  - Provides back-end for ARC processors
  - GCC, GDB, BinUtils, uClibc
  - MilePost GCC can be layered on top

OSS Challenges

- Political
  - Extensibility in GCC seen as a threat
  - Major changes need substantial backing
- Legal
  - GPL3 makes lawyers quiver
- How to make money from OSS
- Moving target
  - Just when your code works someone changes the compiler

OSS Benefits

- Huge user base
  - Peer provided support better than most companies'
  - Much wider feedback / testing
  - Allows greater visibility
- Companies can make use of others' work
  - No NDAs needed once OSS approval in house
- Customers expect GCC availability
  - Not so good for compiler vendors

Conclusion

- A system for automatically tuning compilers
- Improves over human created heuristics
- May learn across architectures
- Heavy use of open source software
- Huge and exciting new field opening up

Questions?