THE UNIVERSITY *of* EDINBURGH
informatics

icsa
Institute for Computing
Systems Architecture

PPar Lunch 2014/12/10

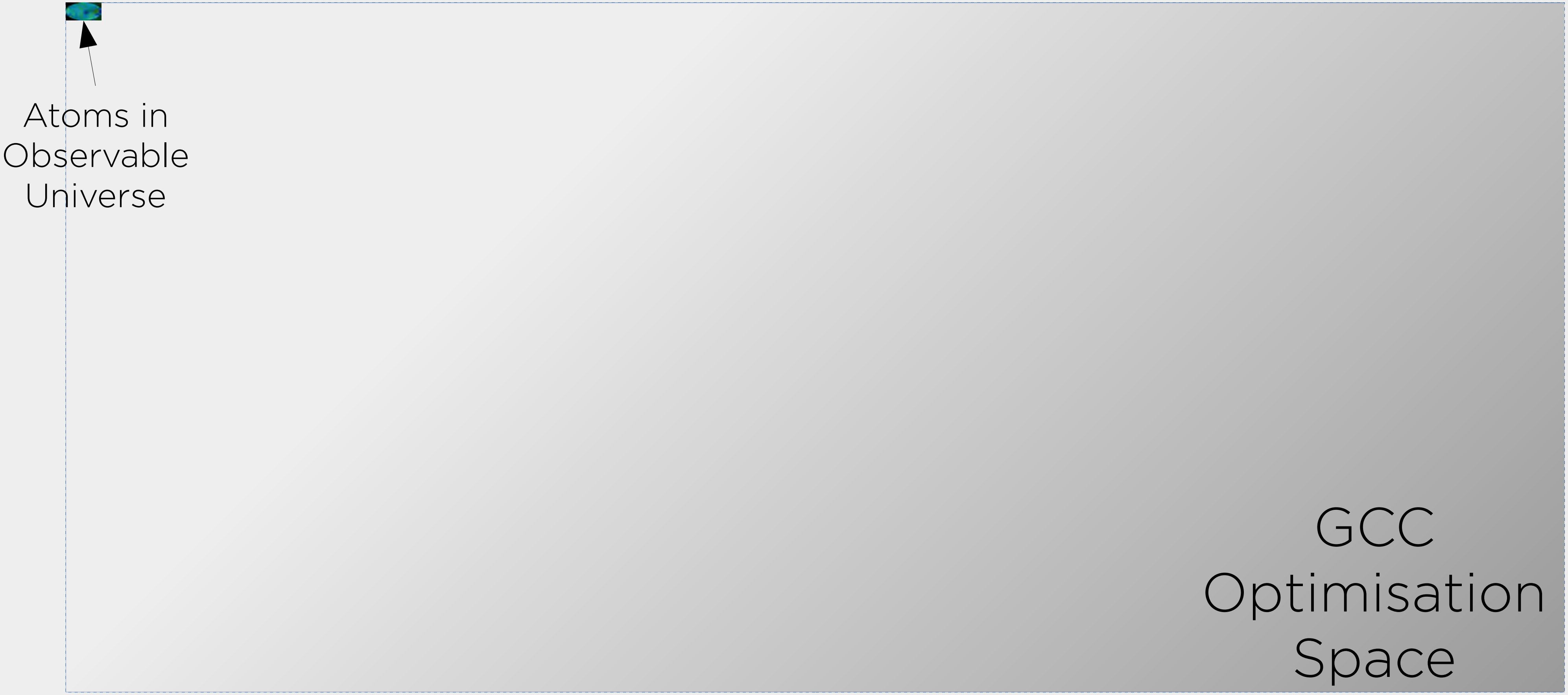
Iterative Compilation 1.01

Compiler optimisations

- Modern Compilers:
  - 100s of possible code transformations
- GCC 4.8.1:
  - 205 optimisation options
  - 138 parameters affecting optimisation

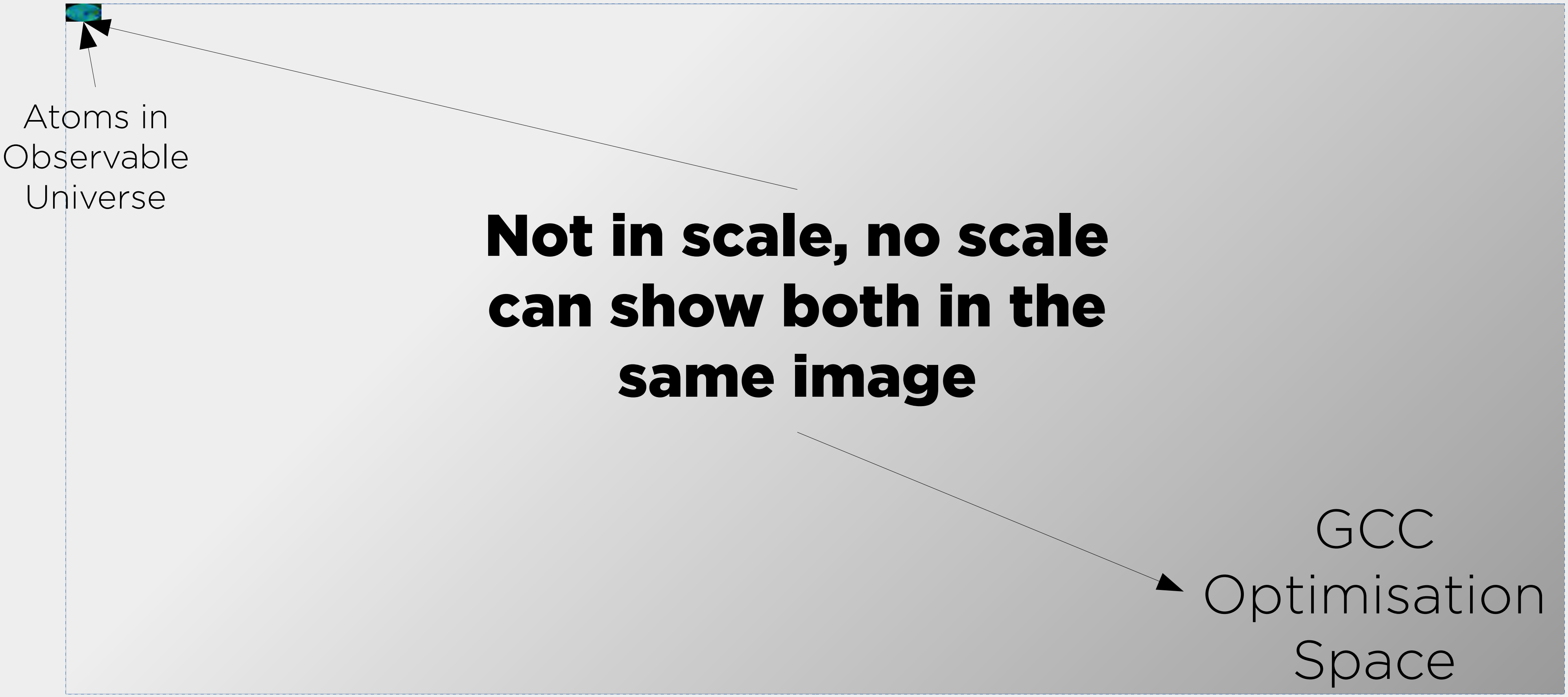
Compiler optimisations

- ~10^{340} combinations!
- ~10^{82} atoms in the universe



Compiler optimisations

- ~10^{340} combinations!
- ~10^{82} atoms in the universe



Atoms in Observable Universe

Not in scale, no scale can show both in the same image

GCC Optimisation Space

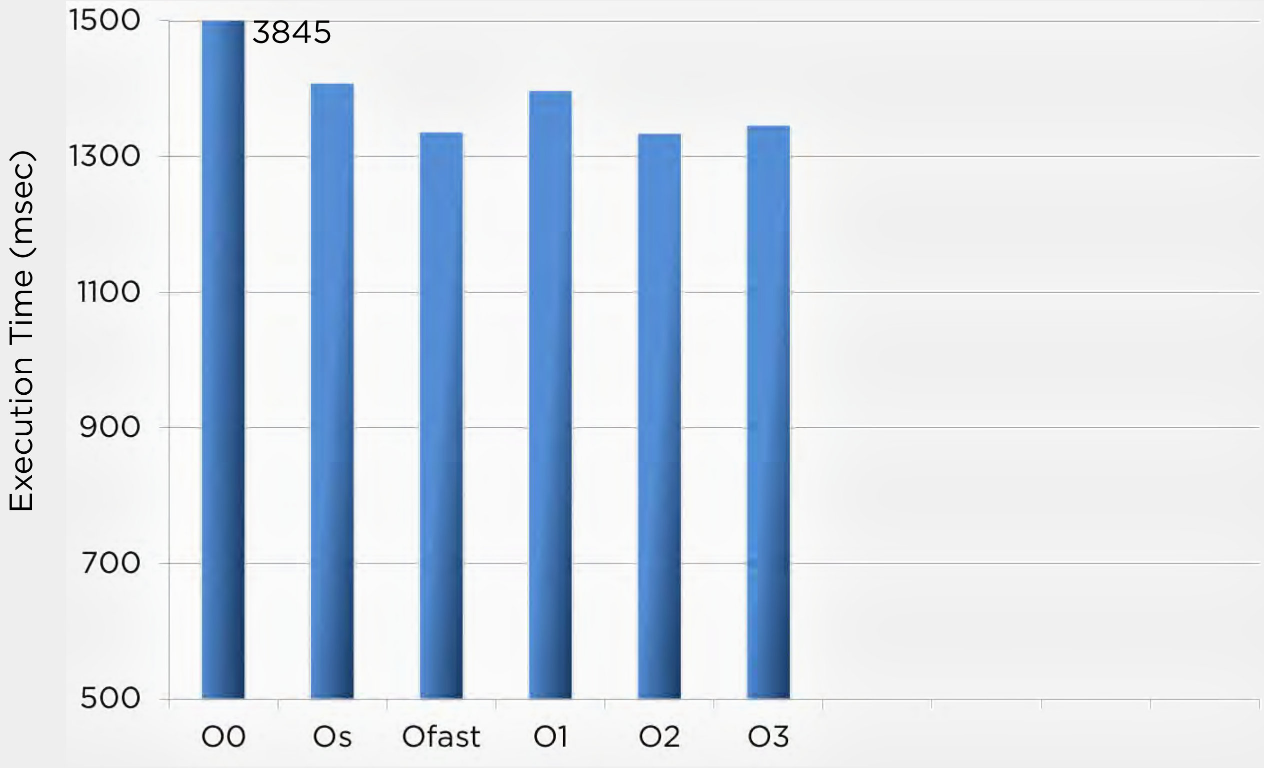
How do we choose optimisations?

How do we choose optimisations?

- Try each one on its own
  - Transformations affect each other
- Use an expert to find the best combo
  - Experts are few
  - One optimisation set doesn't fit all
- Use analytical models
  - Could help, but far from perfect
  - Architecture dependent

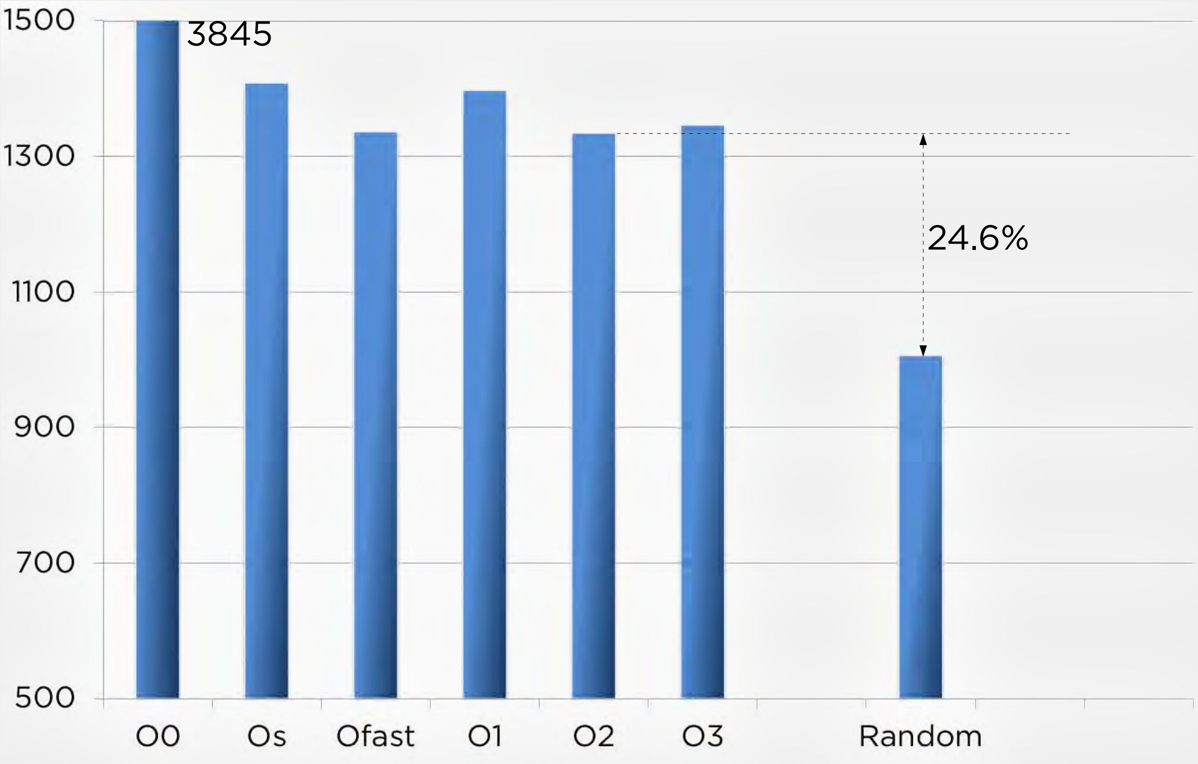
Choosing options - Example

- Usual approach: Let's pretend -O2 is the optimal choice
  - Enables transformations which usually improve performance
  - But not the optimal set of transformations for each program
- libquantum benchmark:



Choosing options - Example

- Usual approach: Let's pretend -O2 is the optimal choice
  - Enables transformations which usually improve performance
  - But not the optimal set of transformations for each program
- libquantum benchmark:

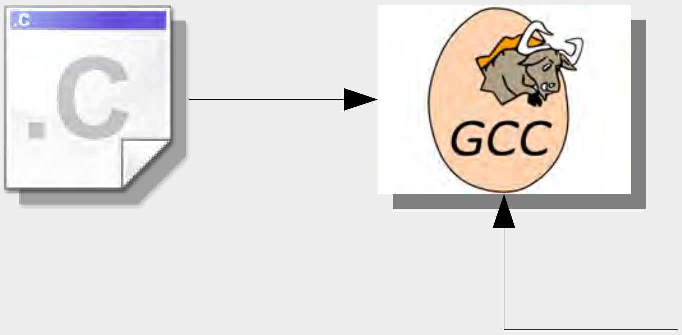


Choosing options

- Performance is lost because of inefficient optimisation strategies
- How do we harvest all the potential performance?

Iterative compilation

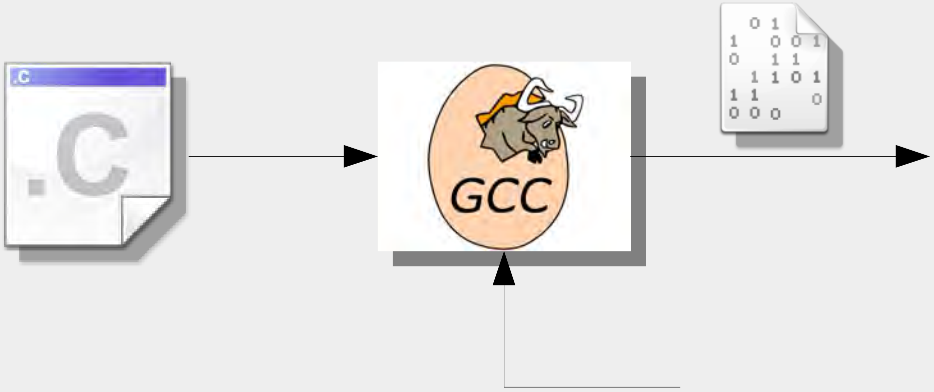
- Choose options



Options

Iterative compilation

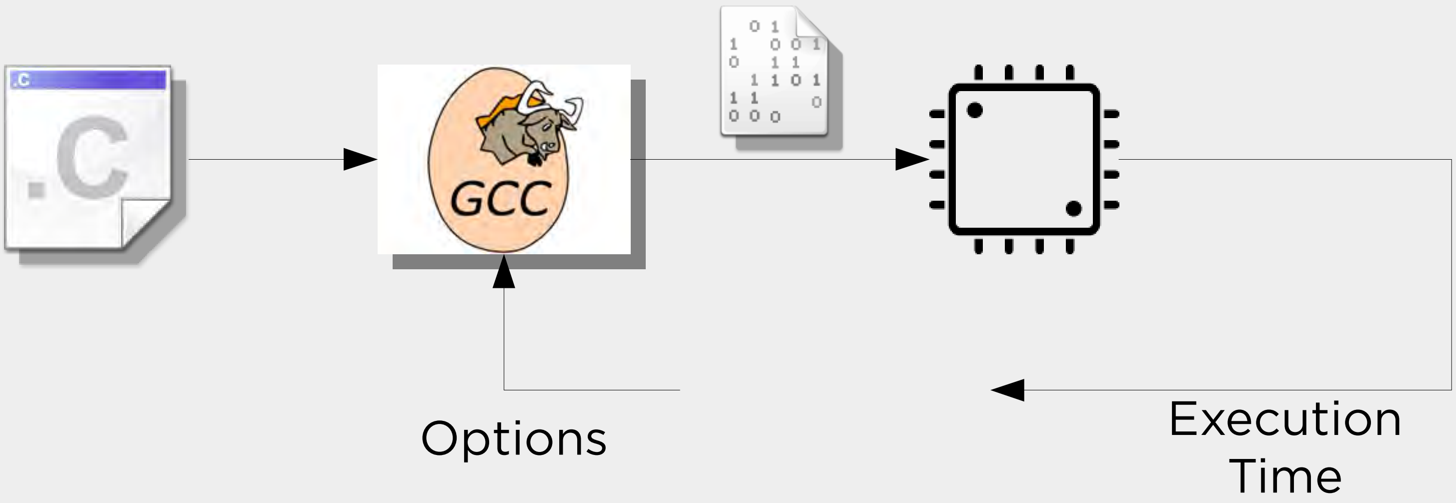
- Choose options
- Build the executables



Options

Iterative compilation

- Choose options
- Build the executables
- Get their runtime

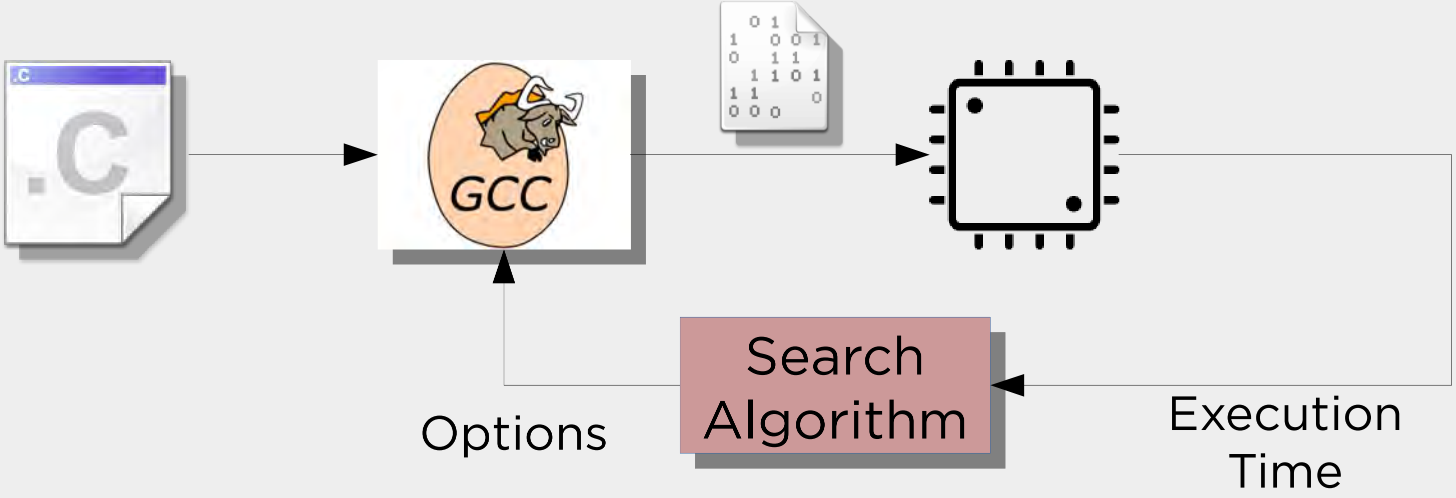


Options

Execution Time

Iterative compilation

- Choose options
- Build the executables
- Get their runtime
- Based on the results, choose the next round of optimisation options and repeat



Search Algorithm

Options

Execution Time

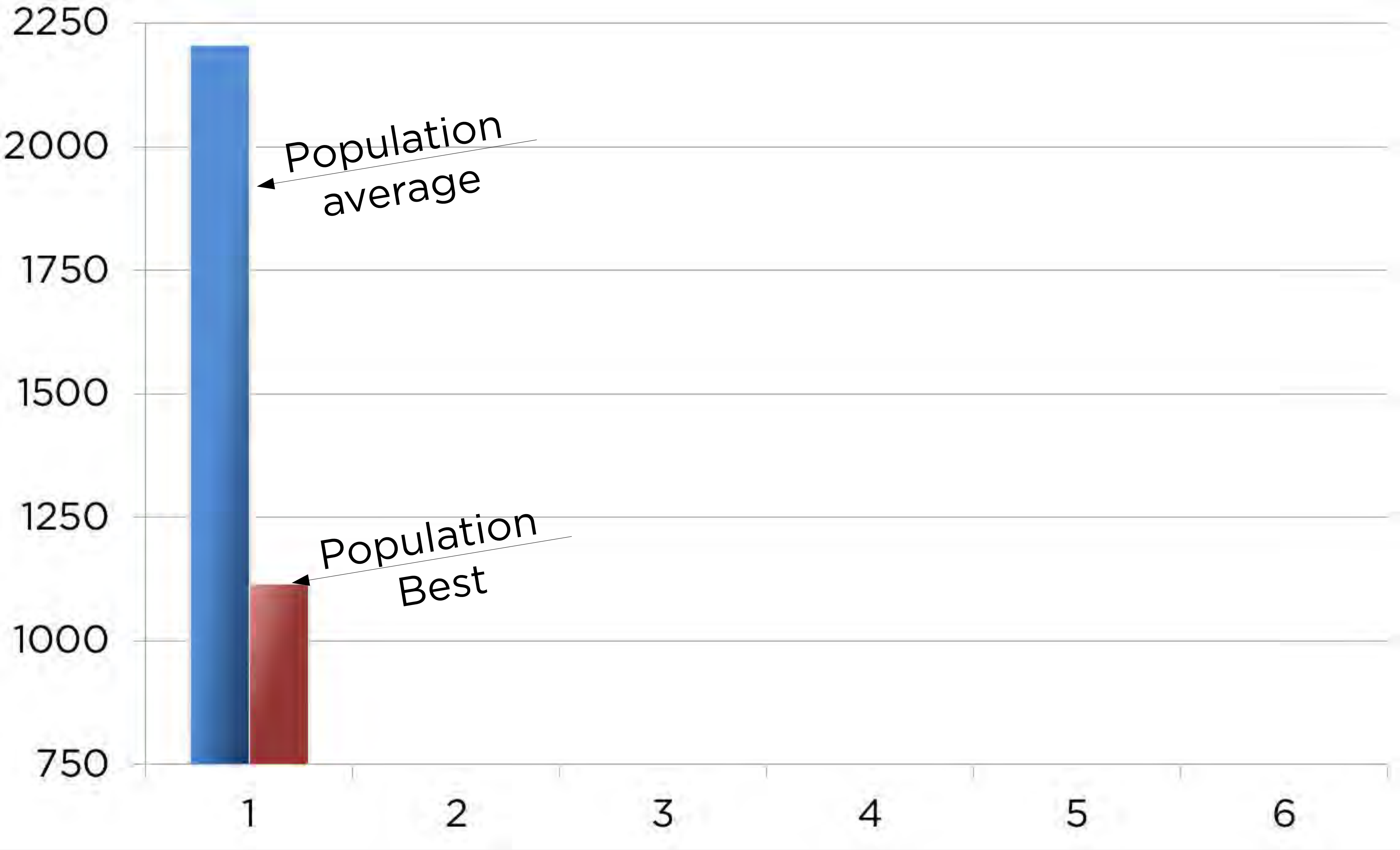
Search Algorithms

- Search algorithm choice → critical parameter
  - Genetic Algorithm, Simulated Annealing, Grid Search, Window Search, etc.
- General Idea:
  - Find high performing areas of the space
  - Improve your knowledge of these areas

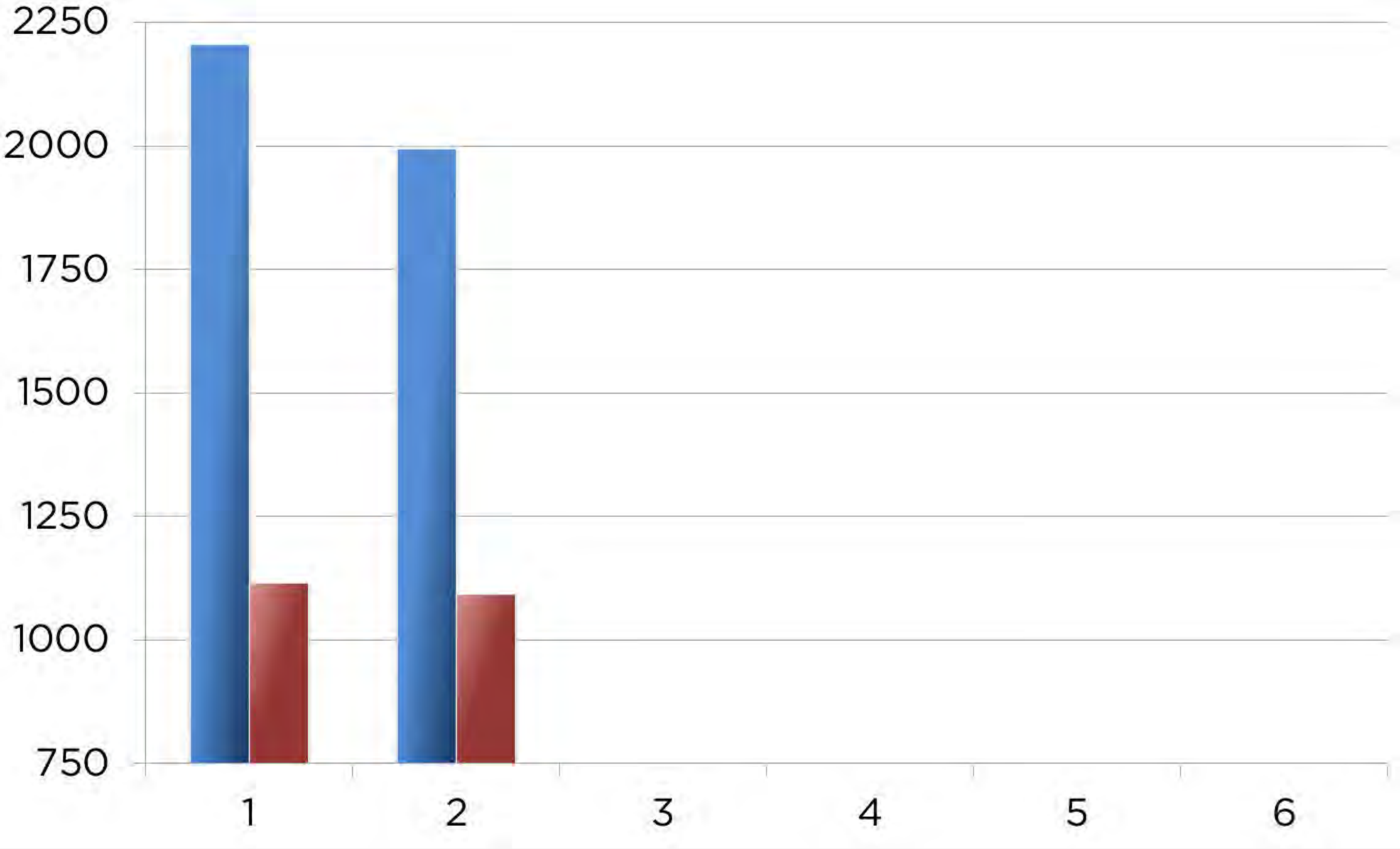
Iterative compilation - Example

- libquantum + Genetic Algorithm

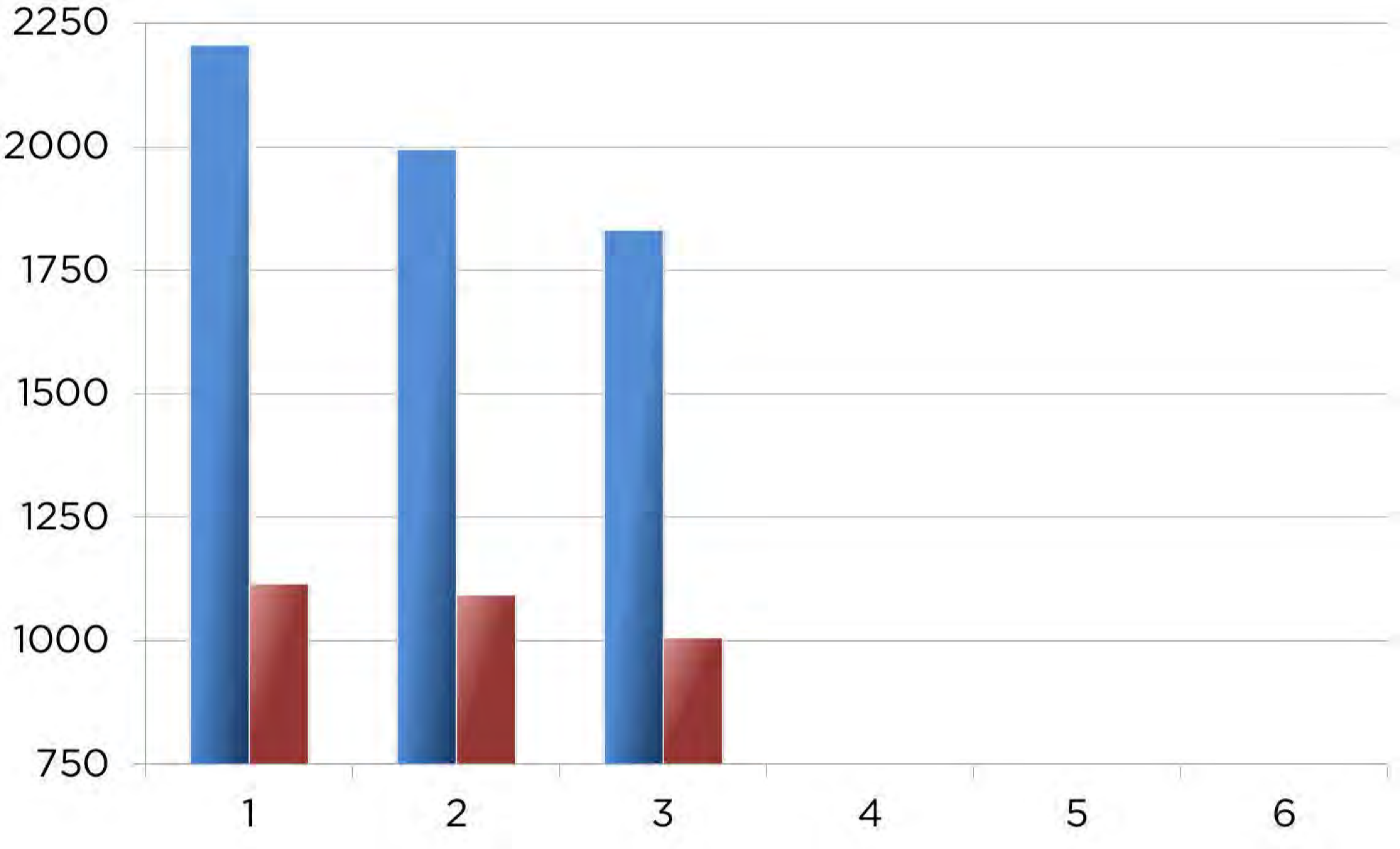
Iterative compilation - Example



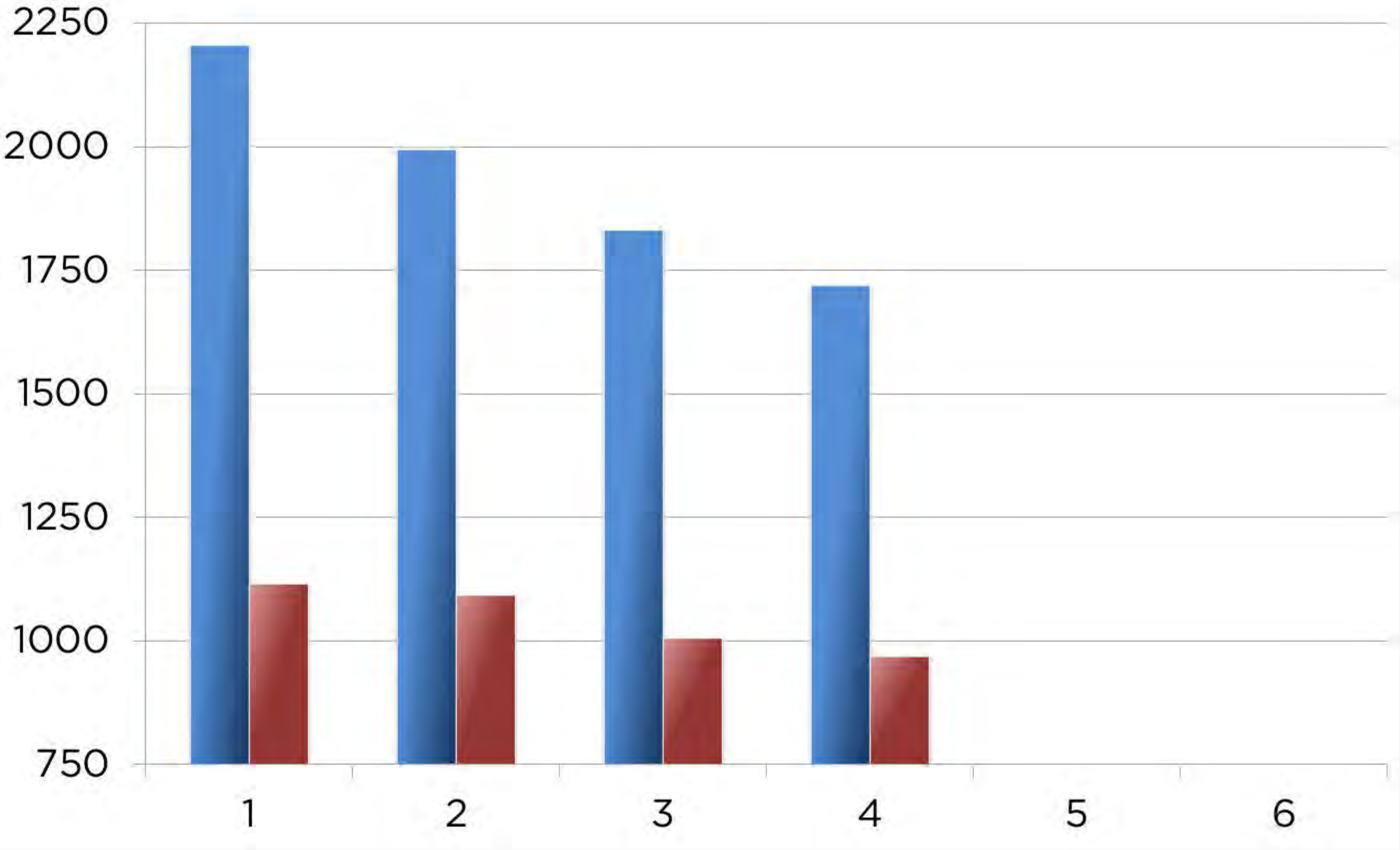
Iterative compilation - Example



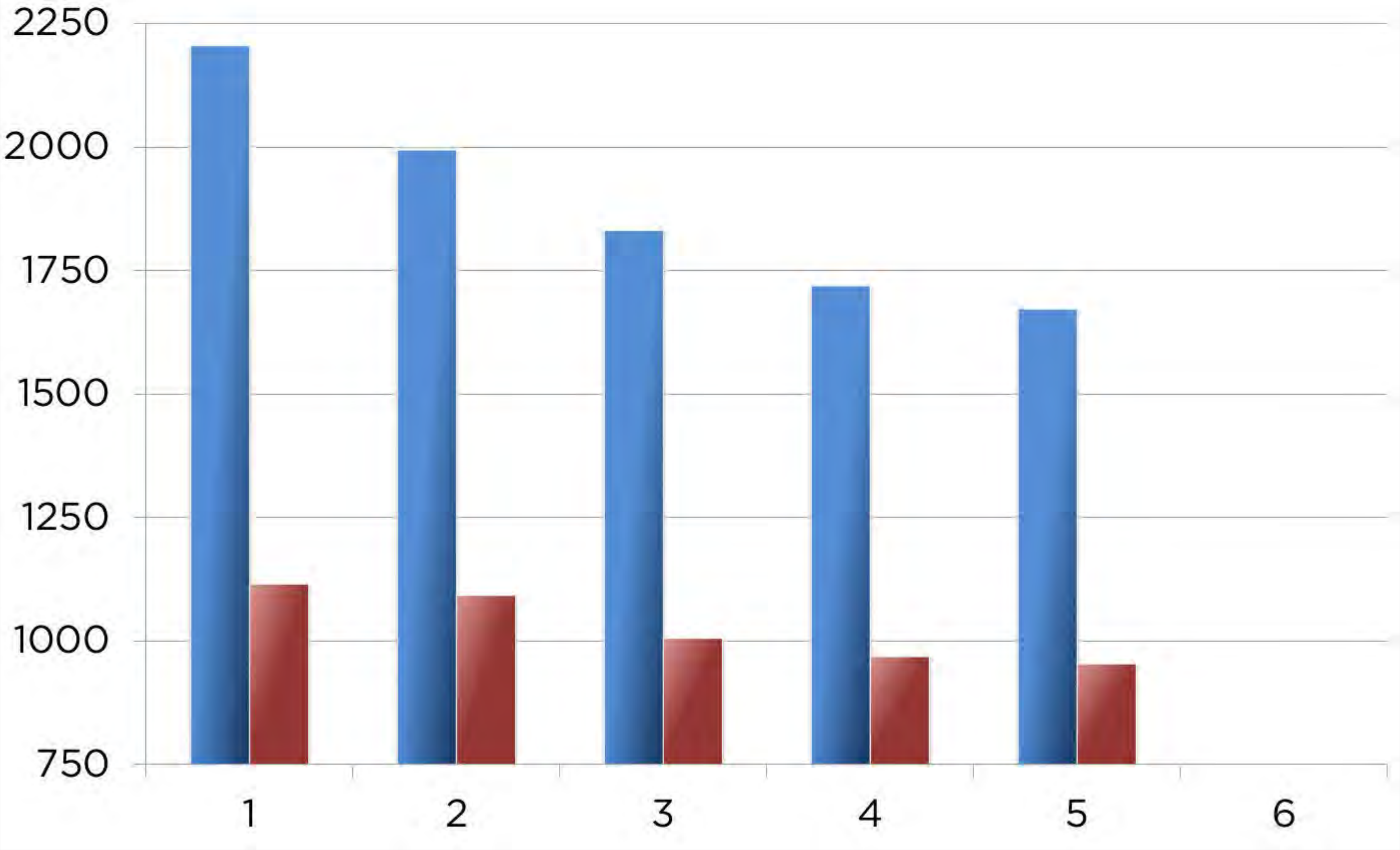
Iterative compilation - Example



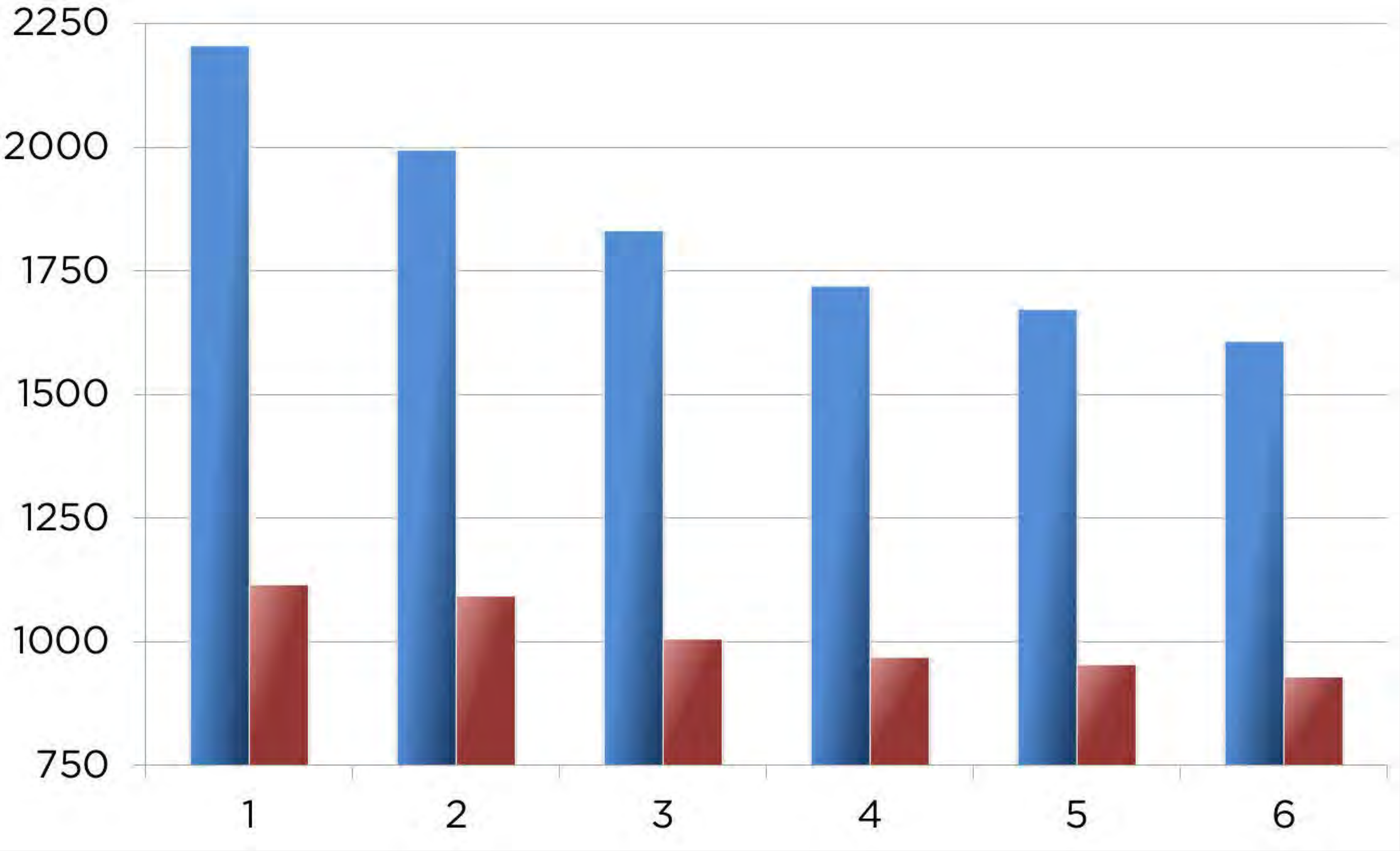
Iterative compilation - Example



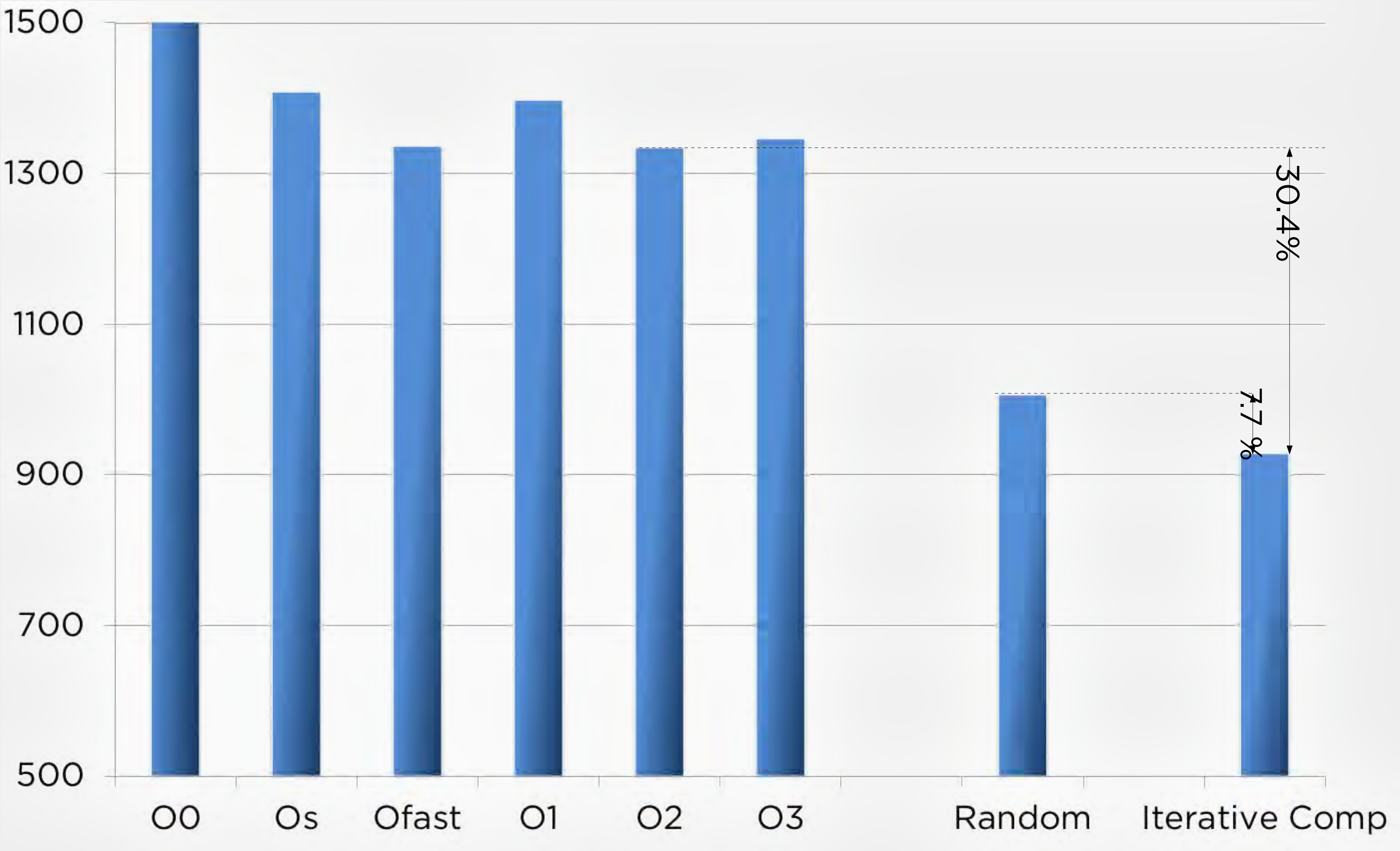
Iterative compilation - Example



Iterative compilation - Example



Iterative compilation - Example



Caveats

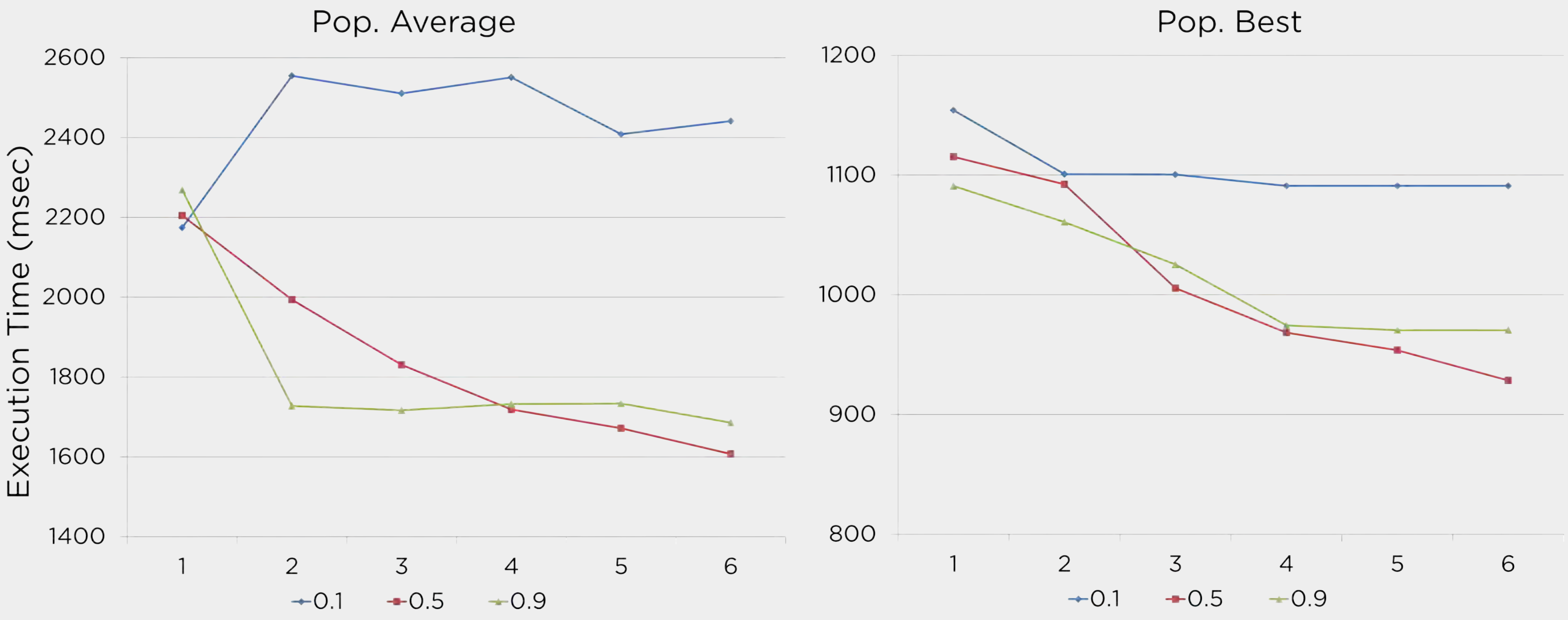
- Local maxima
- Noise
- Compiler errors

Local Maxima

- Wide & shallow vs narrow & deep search
  - Wide and shallow search not bothered by local maxima, but slow
  - Narrow and deep search focuses on local maxima, but might miss the optimal
- Algorithms and their parameters chosen to control this trade-off

Local Maxima - Example

- GA with tournament selection
  - Various tournament selection probabilities



Noise

- Noise makes iterative compilation less efficient

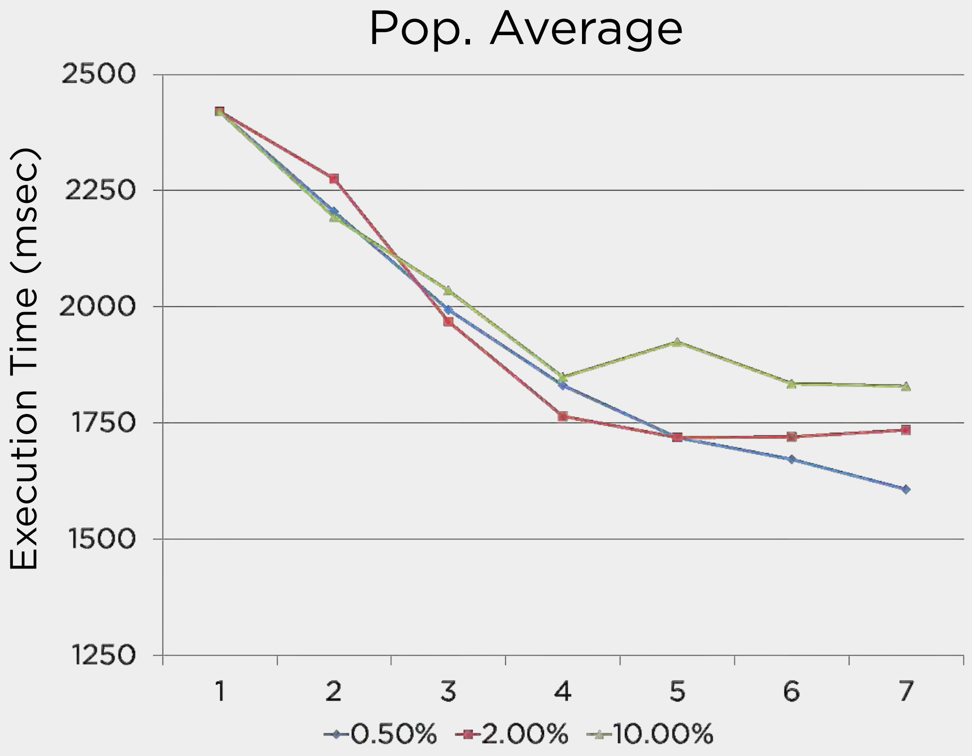
Noise

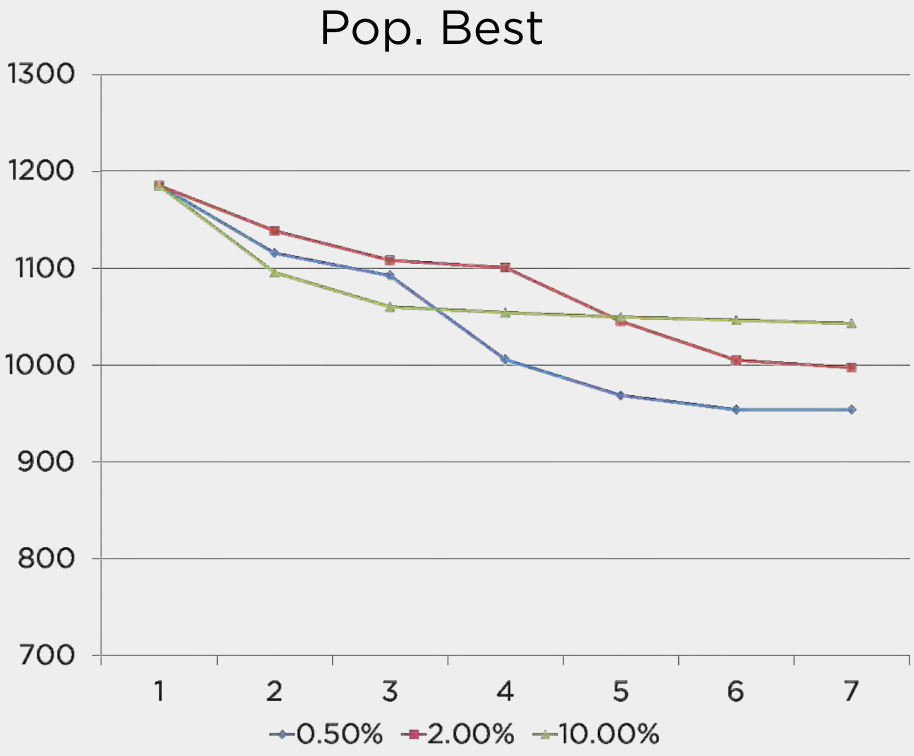
- Noise makes iterative compilation less efficient
- If noise comparable to the performance variation in the population

→ search algorithm becomes random

Noise - Example

- libquantum + gaussian noise





Noise

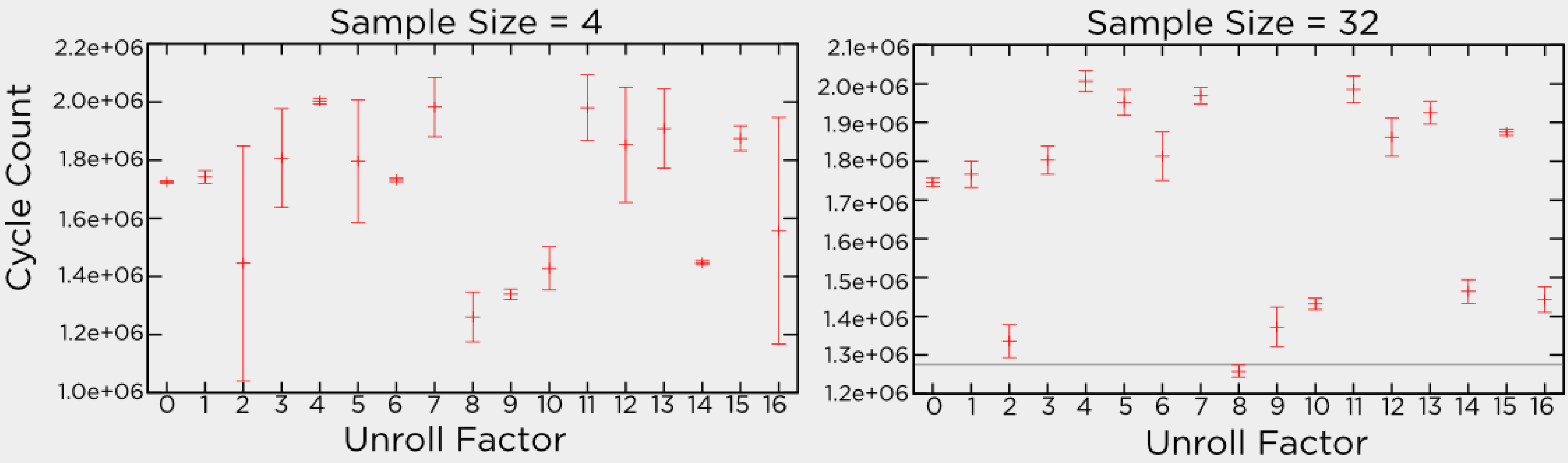
- Multiple observations to establish with some certainty the runtime of an executable

    → Higher evaluation overhead

    → slow

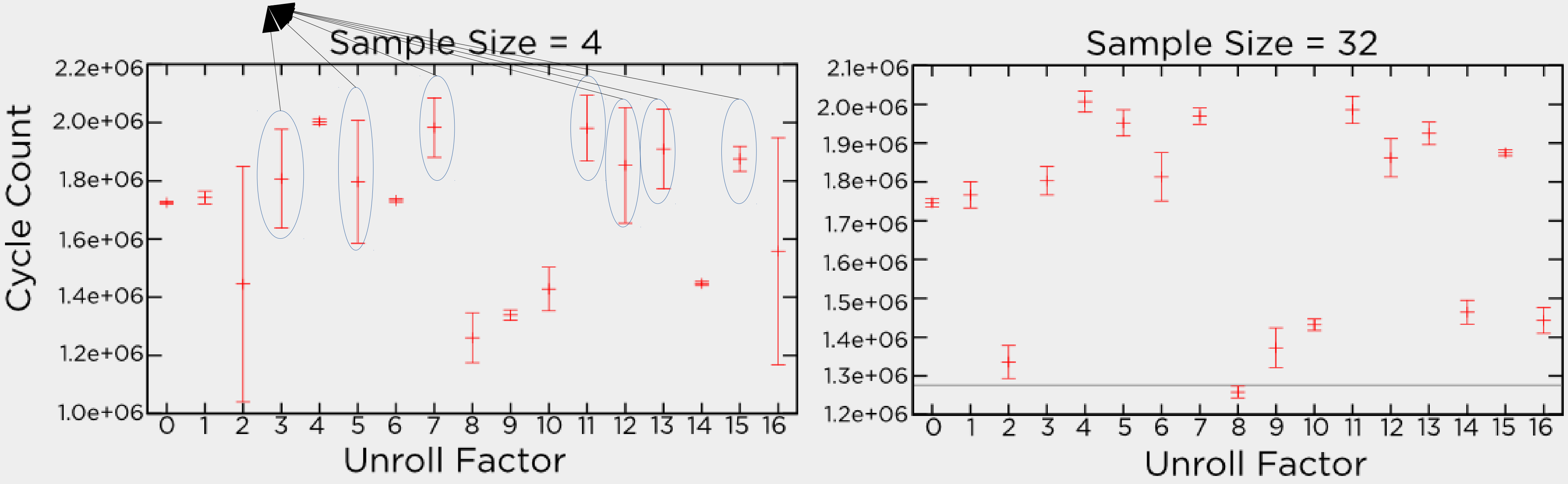
Noise

- Do we need high certainty?



Noise

- Do we need high certainty?

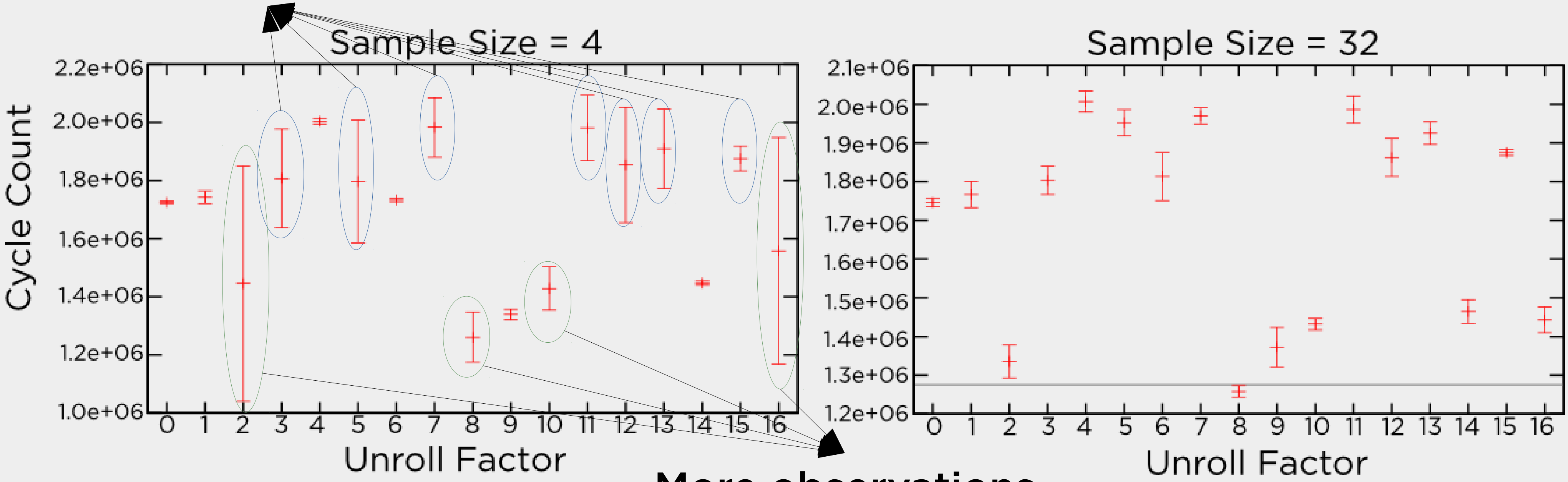


Do we care?

Noise

- Do we need high certainty?

Not really



More observations

Compiler Errors

- By randomly combining optimisations we enter an uncharted area
- Some combinations crash the compiler:
  - They don't work together and we know it
  - They don't work together and we don't know it
  - They trigger unknown bugs in the compiler code
- Crashes affect the speed of iterative compilation
  - Time wasted on unsuccessful builds
  - Benefit if we find problematic combinations early

Compiler Errors

- Others affect the program correctness
  - Due to bugs or unsafe transformations
- How do we handle them?
  - Fully deterministic behaviour
  - Output sensitive on changes in the control flow
  - Validation of the output after each execution

Improving iterative compilation

- Focus searching on areas predicted to be near optimal
  - Human expert input
  - Analytical models
- Pruning of irrelevant/harmful transformations
- Reduce evaluation overhead
  - Iterative compilation on the function call level

Iterative compilation - Drawbacks

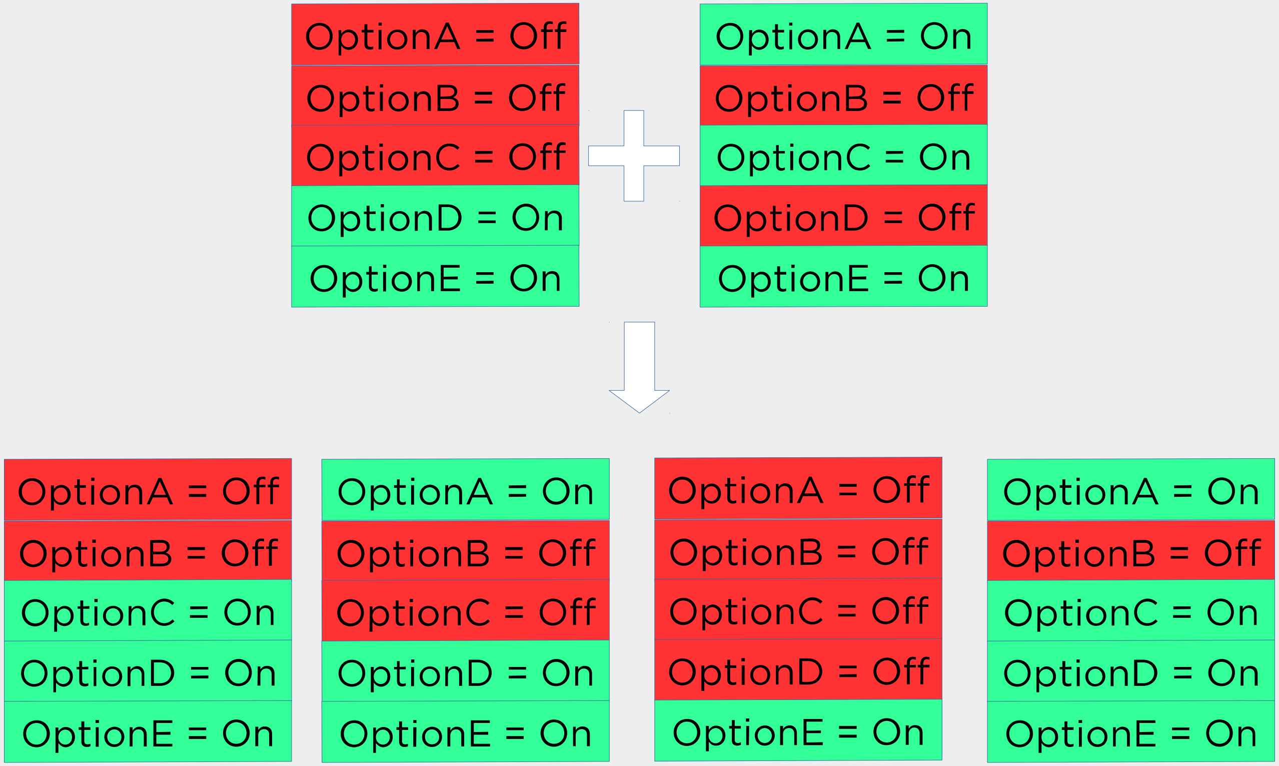
Iterative compilation - Drawbacks

- Takes too long
  - Hours to month for the search to terminate
- Search parameters might need to be hand tuned
- Still, it's worth the effort for:
  - embedded systems
  - libraries
  - and others

Beyond iterative compilation

- Most of the cost → evaluation
  - 1000s of evaluations
  - To the garbage bin when we complete the search
- Can we somehow reuse the evaluation results for different programs?

Genetic Algorithm - Example

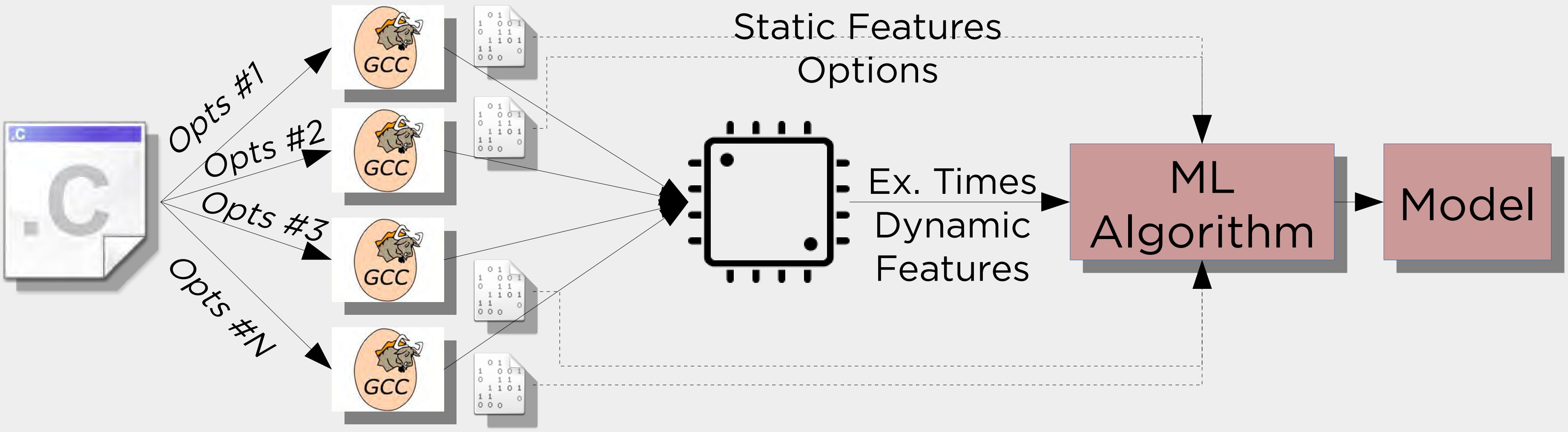


OptionA = Off	OptionA = On
OptionB = Off	OptionB = Off
OptionC = Off	OptionC = On
OptionD = On	OptionD = Off
OptionE = On	OptionE = On

OptionA = Off	OptionA = On	OptionA = Off	OptionA = On
OptionB = Off	OptionB = Off	OptionB = Off	OptionB = Off
OptionC = On	OptionC = Off	OptionC = Off	OptionC = On
OptionD = On	OptionD = On	OptionD = Off	OptionD = On
OptionE = On	OptionE = On	OptionE = On	OptionE = On

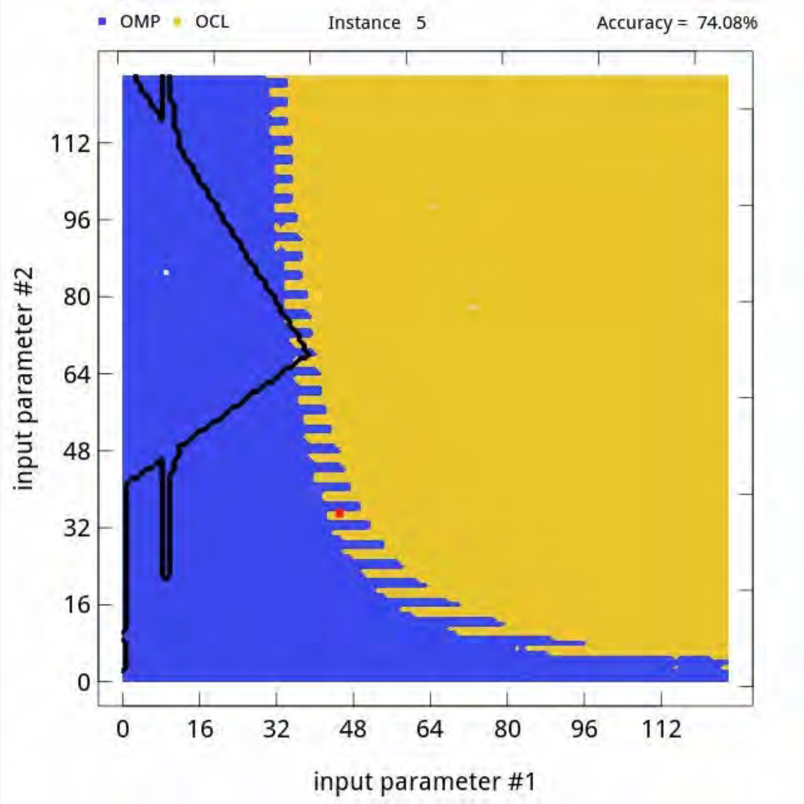
ML-based optimising compiler

- Choose options
- Build
- Evaluate
- Train



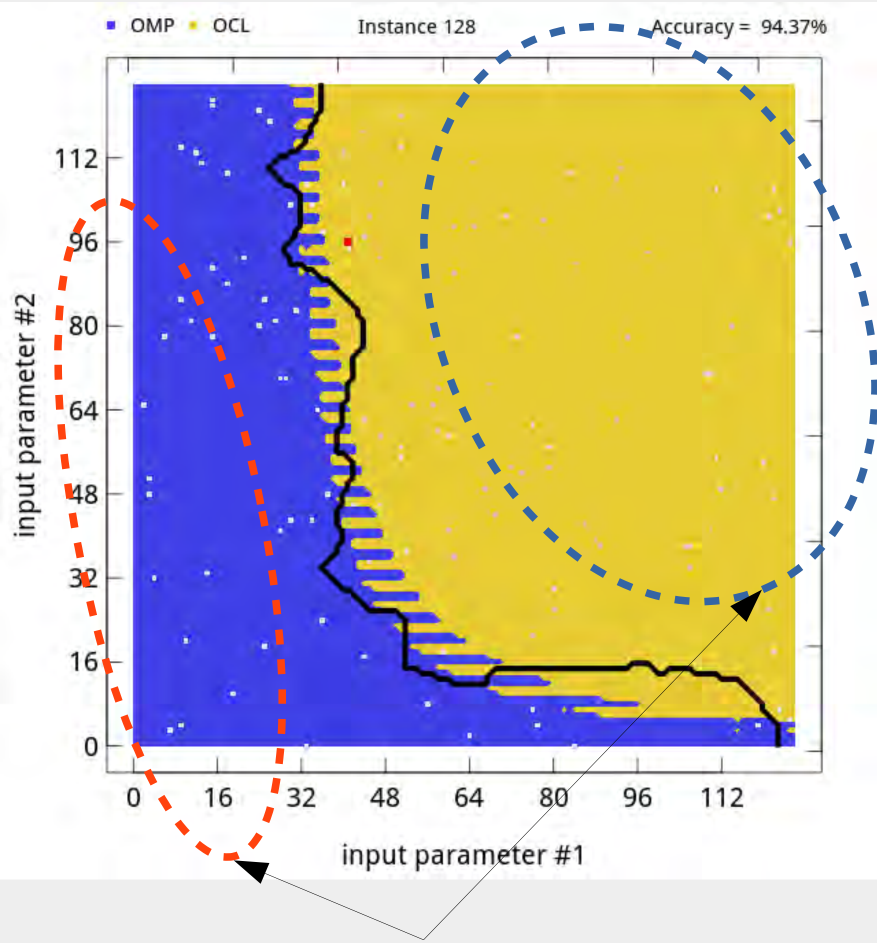
Example

- Best device (cpu or gpu) for an app based on the input size
  - Hotspot benchmark, 2d input
  - Random input sizes evaluated
  - Model built with the nearest neighbour algorithm



Example

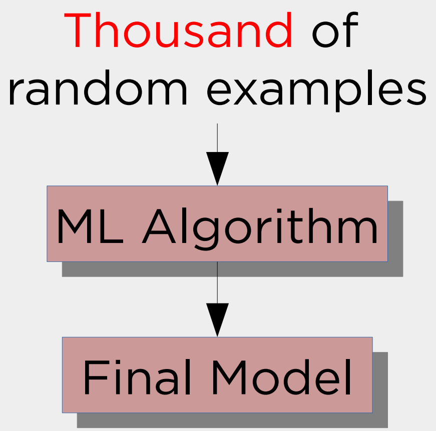
- Most points offer little to no information → don't improve the model

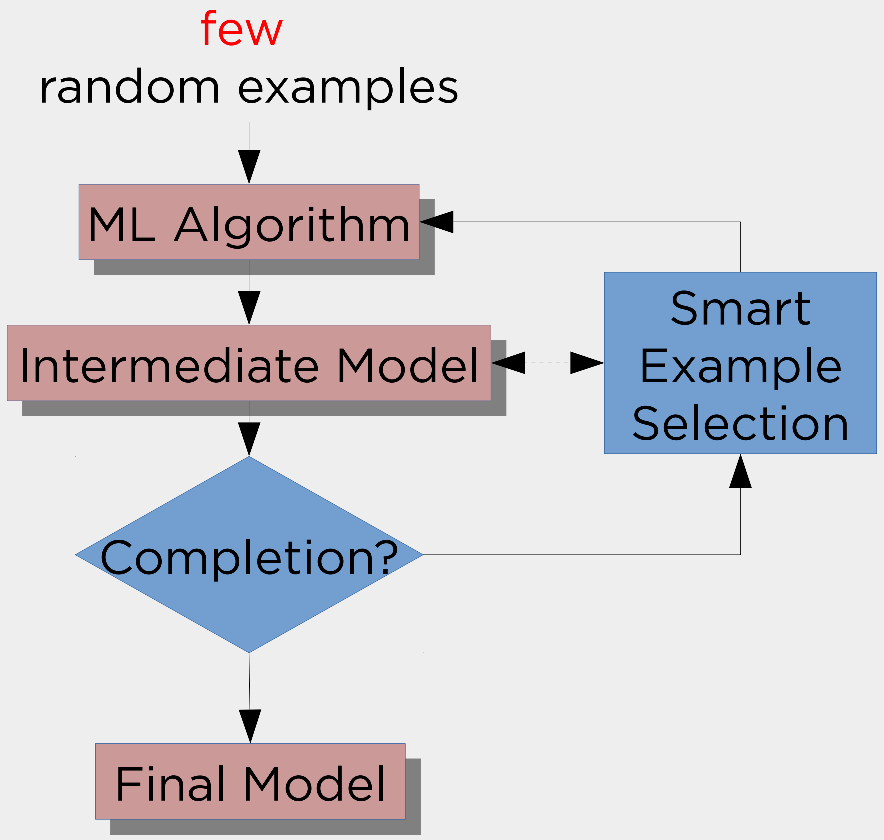


Waste of time

Active Learning

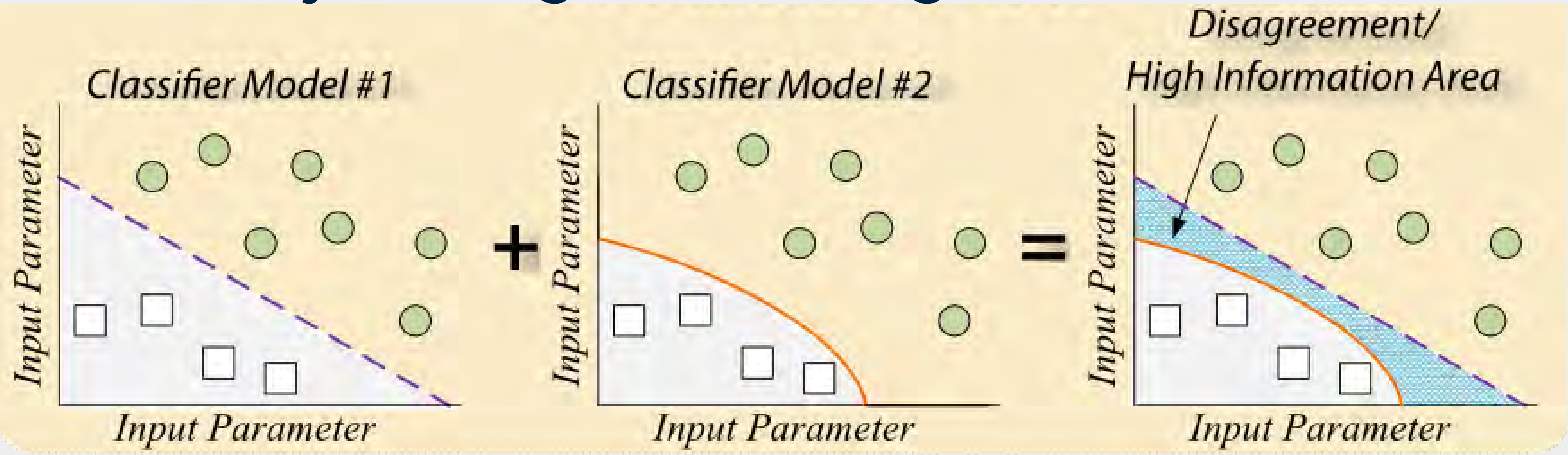
- Random training points selection wasteful
- Active Learning



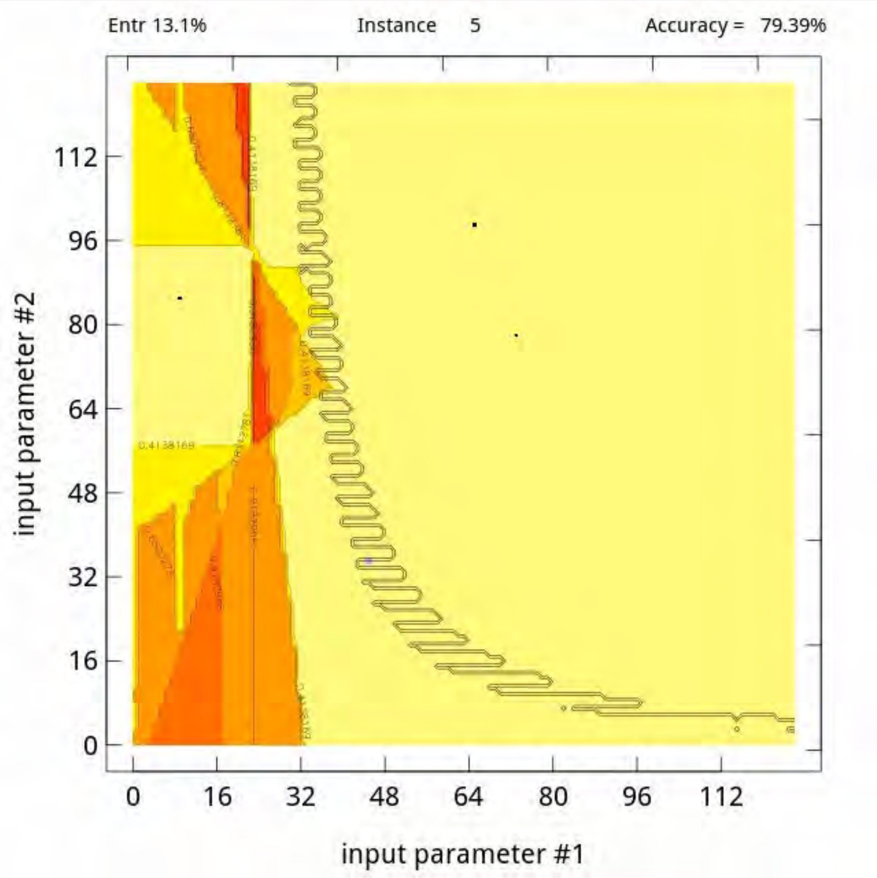


Active Learning

- Best points → High information points
  - close to the cpu-gpu border
  - max uncertainty about the best device
- Quantifying uncertainty:
  - Multiple ML algorithms trained with the same points
  - Uncertainty == degree of disagreement

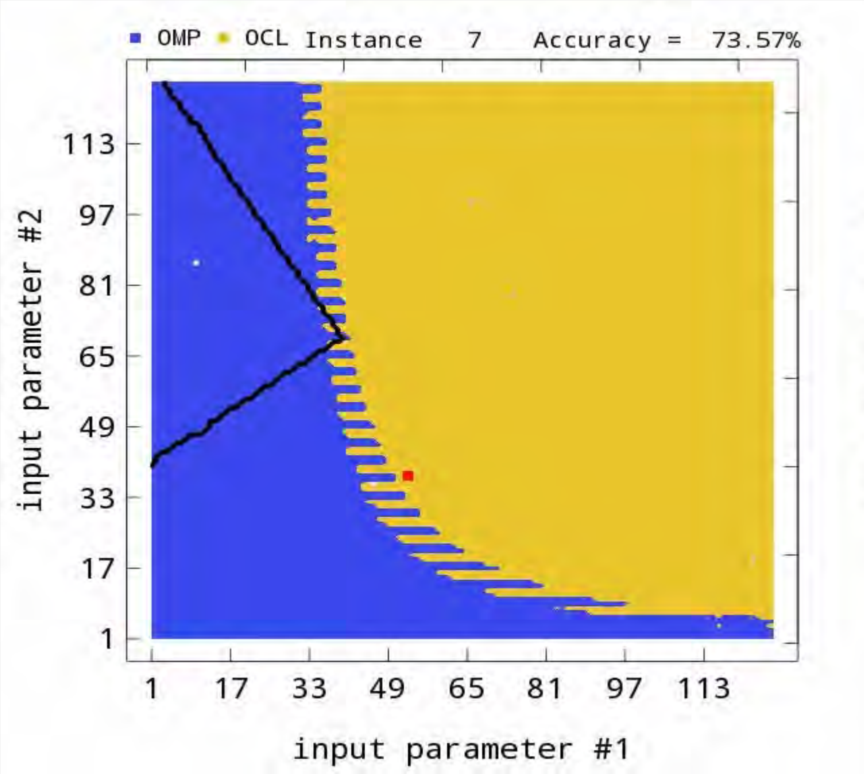


Disagreement



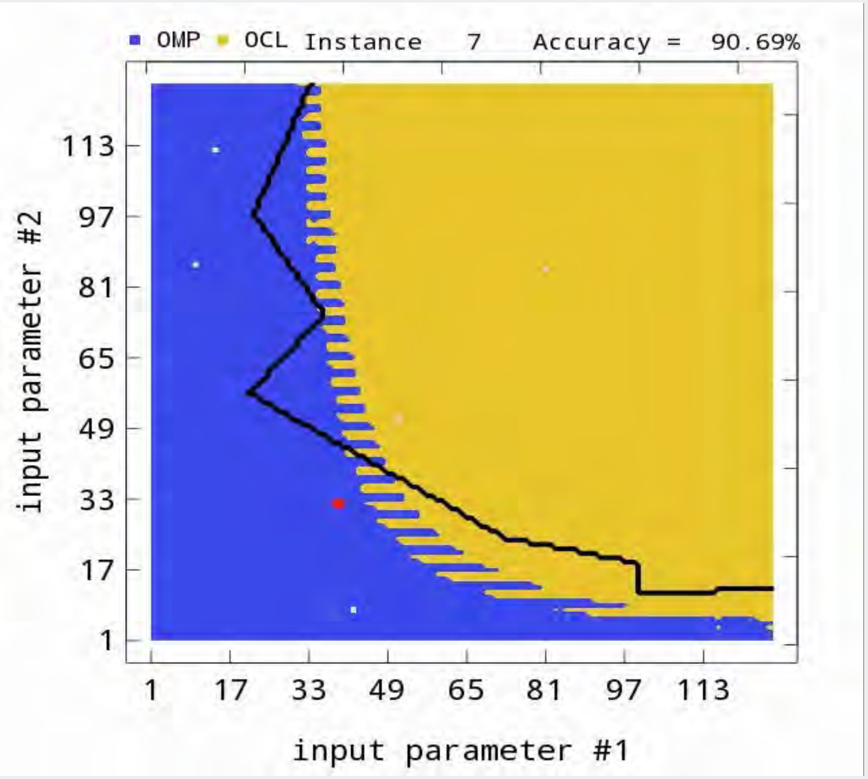
Comparison

Random



130 training points for 95% accuracy

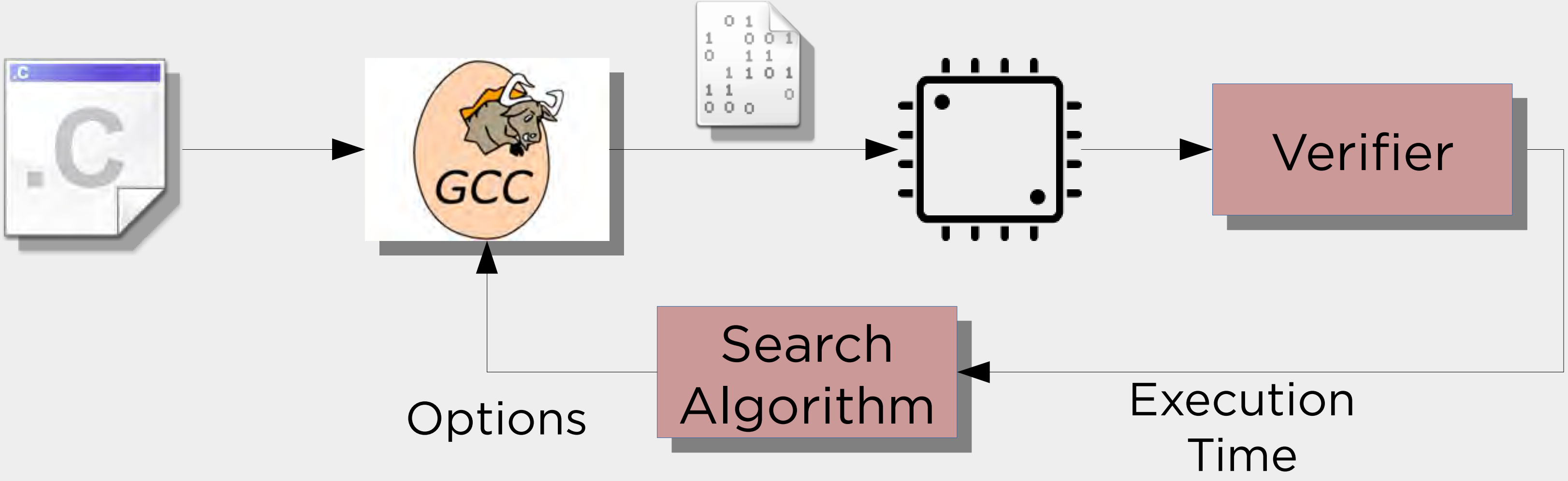
Active Learning



85 training points for 95% accuracy

Iterative compilation

- Choose options
- Build the executables
- Get their runtime
- Based on the results, choose the next round of optimisation options and repeat


.C
GCC
Verifier
Search Algorithm
Options
Execution Time