Machine Learning, Compilers and Mobile Systems

Institute for Computing Systems Architecture
University of Edinburgh, UK

Hugh Leather

Introduction

- Mobile is the next big thing
- Machine learning key to power and performance
- Prior work making machine learning in compilers practical at scale

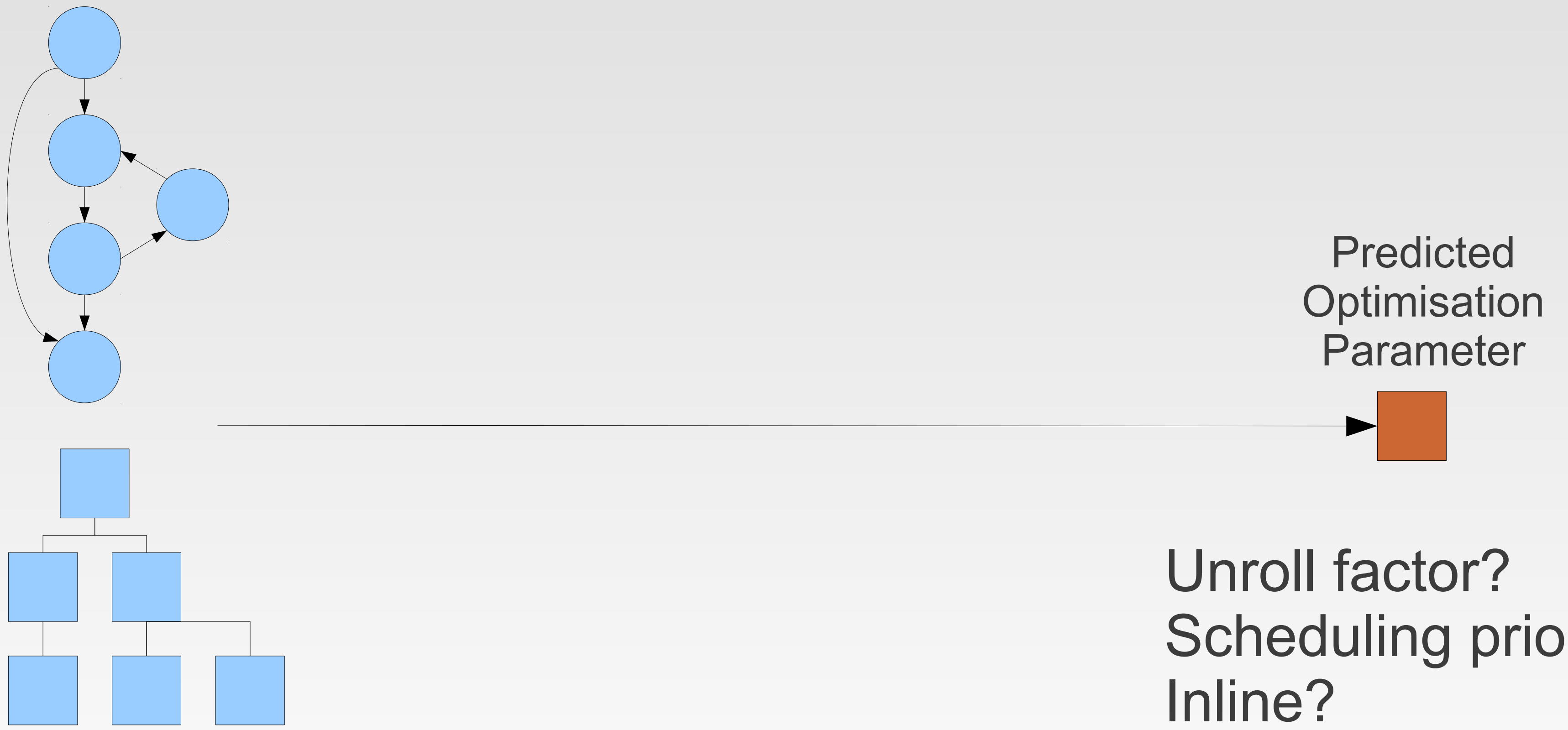
Machine Learning in Compilers

Machine learning in compilers

- Problem:
  - Tuning heuristics is hard
  - Architectures and compilers keep changing
- Goal:
  - Replace an heuristic with a Machine Learned one
  - ML performs very well

Machine learning in compilers

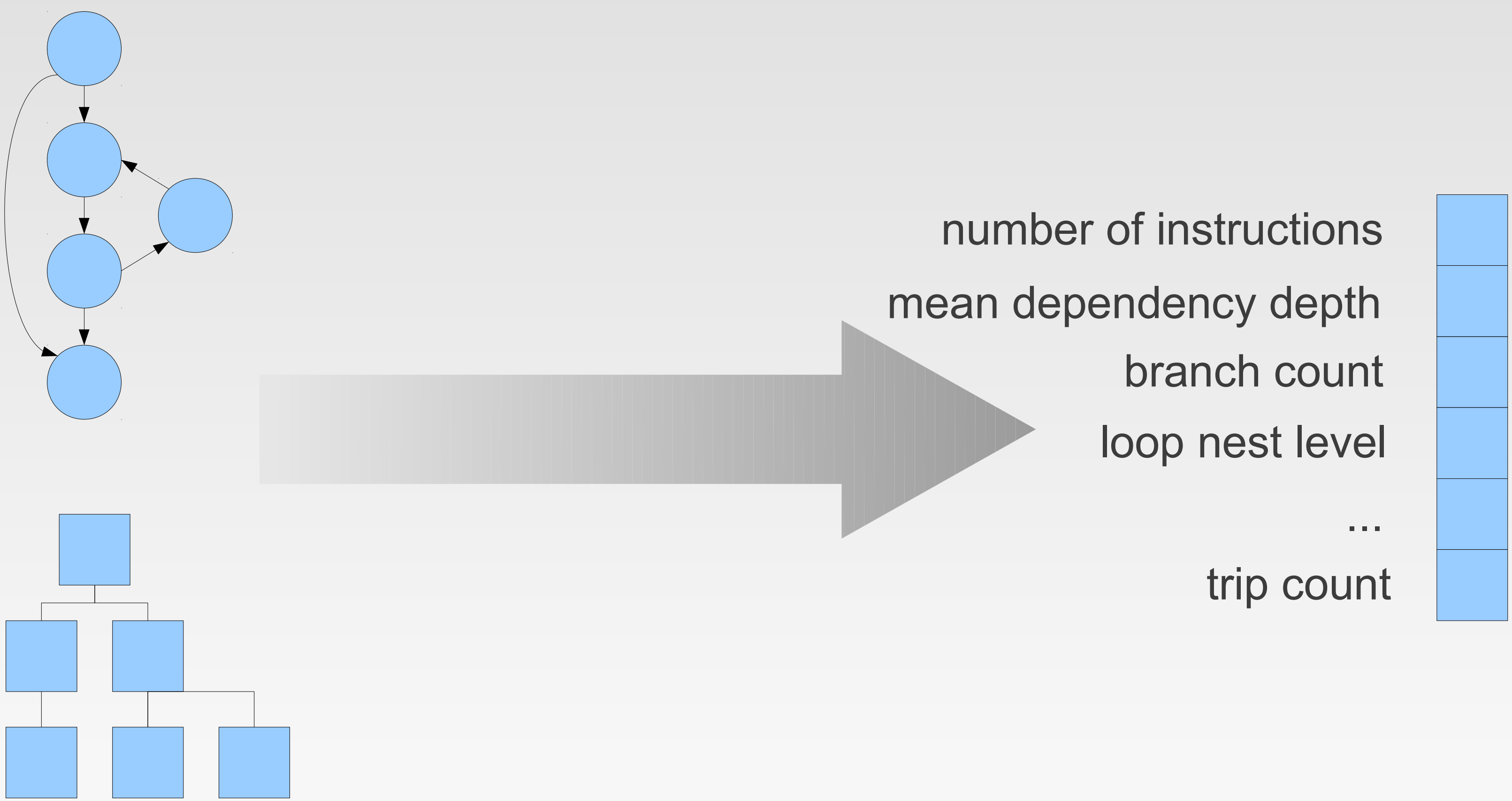
Start with compiler data structures
AST, RTL, SSA, CFG, DDG, etc.



Unroll factor?
Scheduling priority?
Inline?
Compiler flags?

Machine learning in compilers

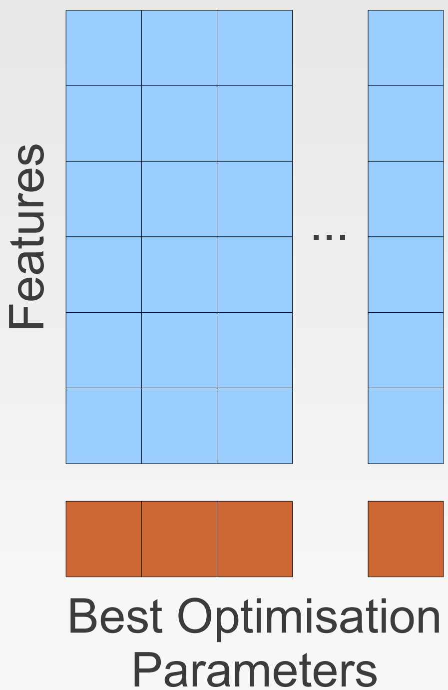
Human expert determines a mapping
to a feature vector



Machine learning in compilers

Now collect many examples of programs, determining their feature values

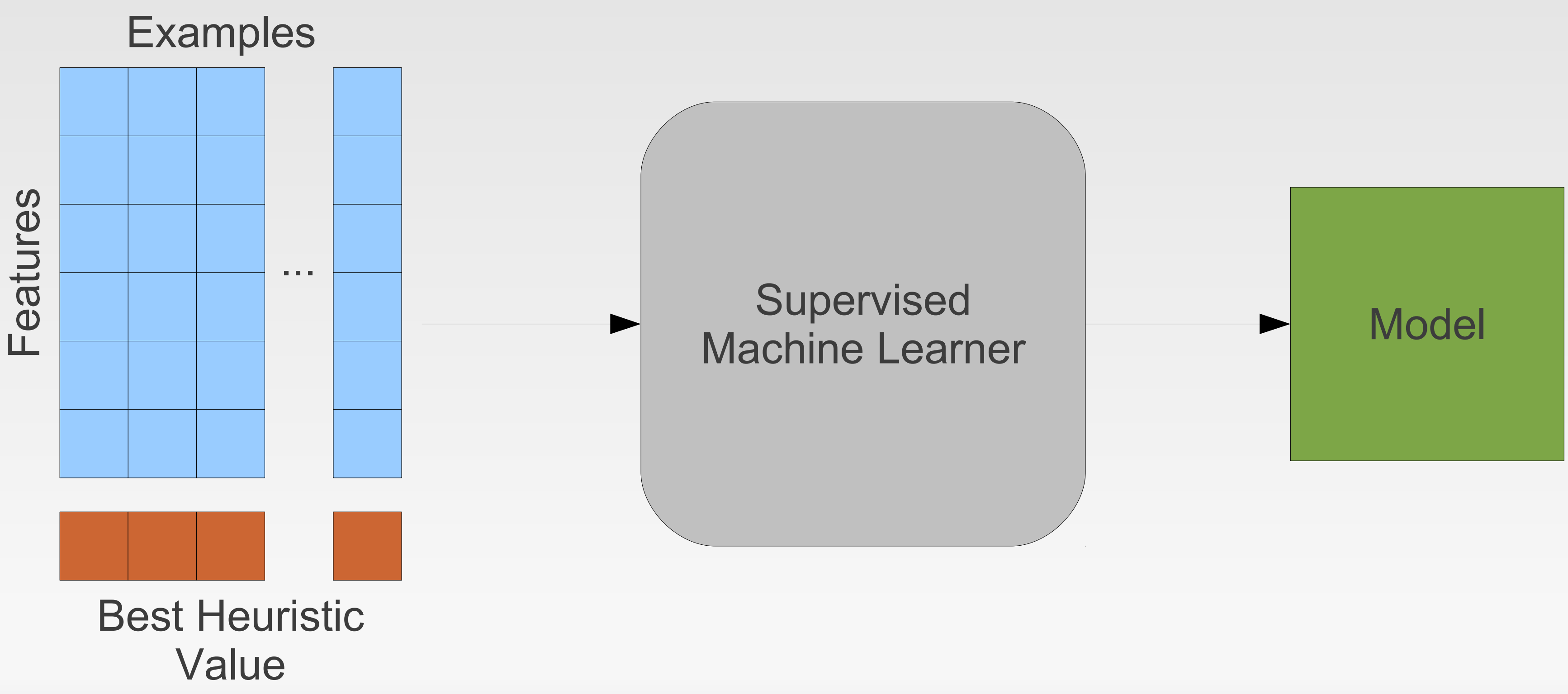
Execute the programs with different compilation strategies and find the best for each



Machine learning in compilers

Now give these examples to a machine learner

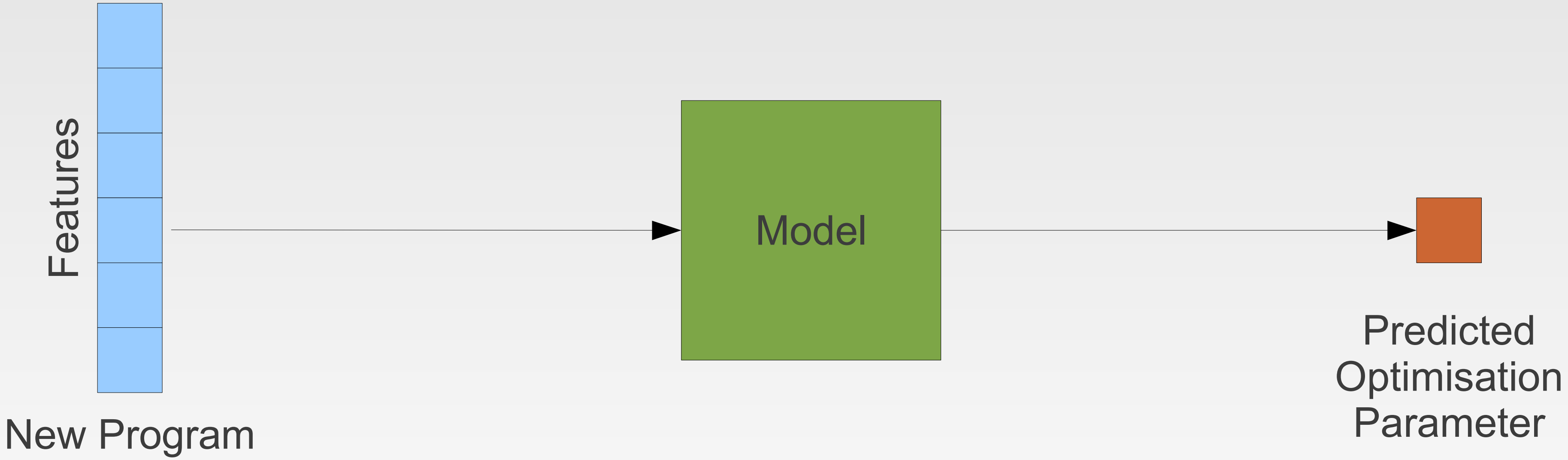
It learns a model



Machine learning in compilers

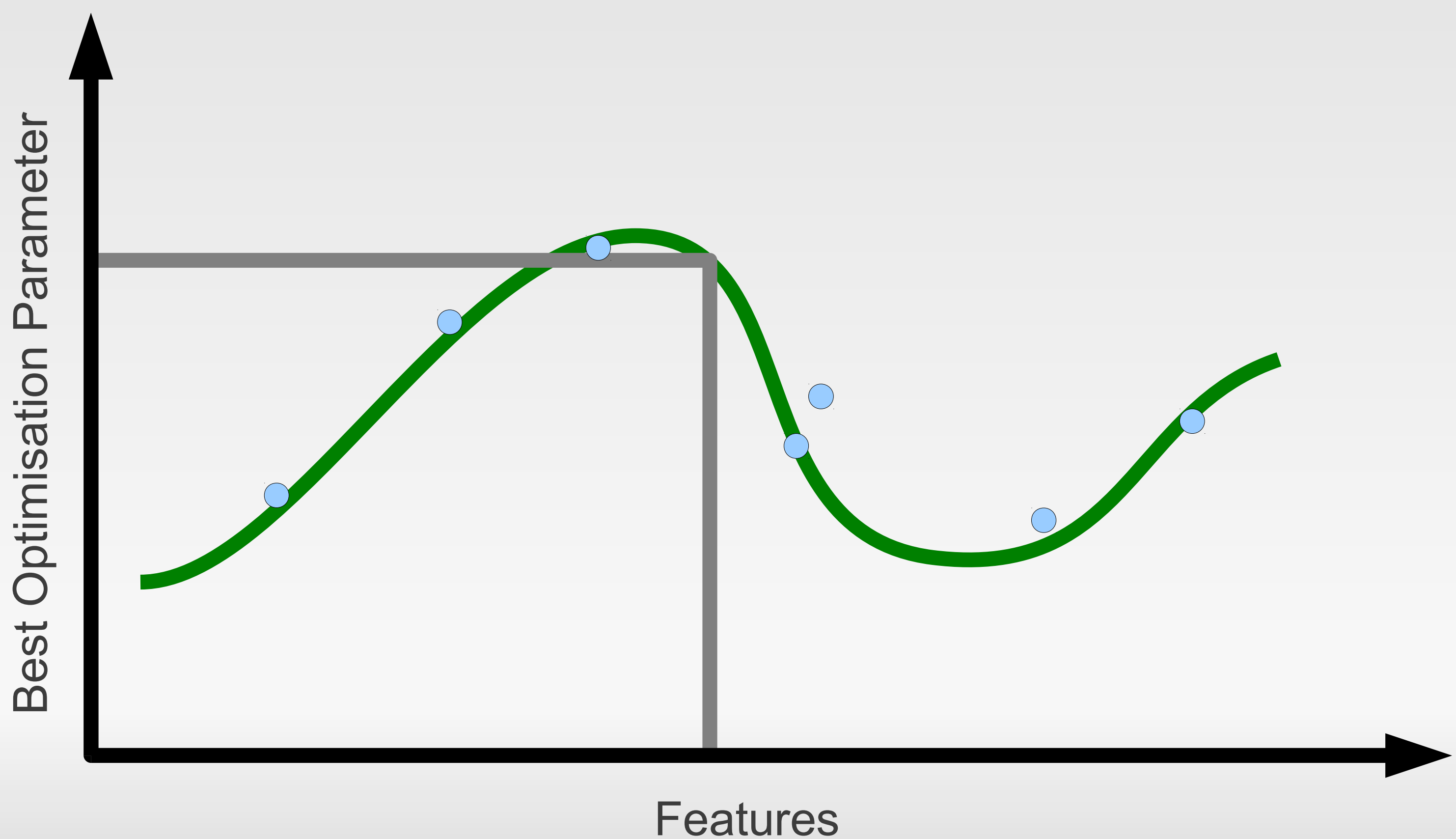
This model can then be used to predict the best compiler strategy from the features of a new program

Our heuristic is replaced



Machine learning in compilers

- Fit a curve (model) to data
- Look new point on curve for prediction

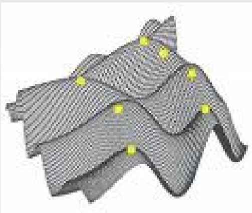


The pillars of machine learning in compilers

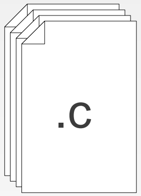
- Need to make practical at scale

Compiler Internals Access
*lib*Plugin
Sourceforge – 400+ downloads

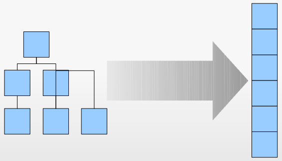
Cost of Iterative Compilation
Profile Races
LCTES 09



Enough Benchmarks
Automatic Benchmark Generation
In preparation



Choosing the Features
Automatic Feature Generation
CGO 09



Automatic Feature Generation

Choosing Features

- Problem
  - ML relies on good features
  - Subtle interaction between features and ML
  - Infinite number of features to choose from
- Solution
  - *Automatically search for good features!*

An example – Loop unrolling

- Set up
  - 57 benchmarks from MiBench, MediaBench and UTDSP
  - Found best unroll factor for each loop in [0-16]
  - Exhaustive evaluation to find oracle

An example – Loop unrolling

Original Loop

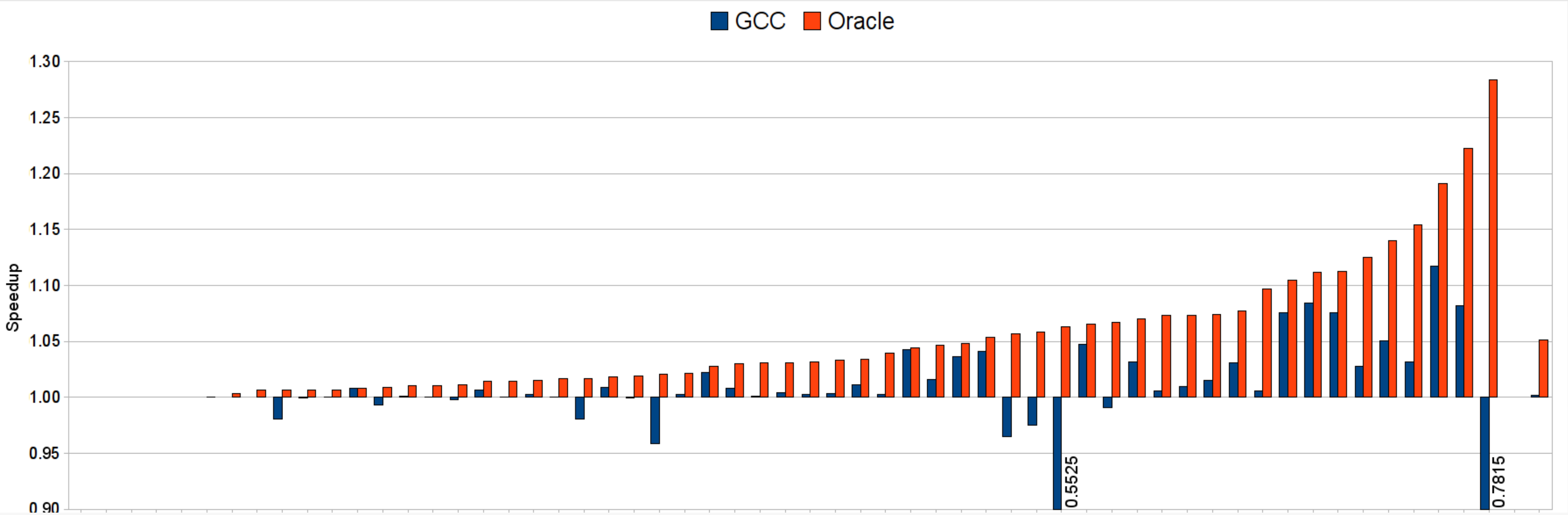
```
for( i = 0; i < n; i = i ++ ) {
    c[i] = a[i] * b[i];
}
```

Unrolled 5 times

```
for( i = 0; i < n; i = i + k ) {
    c[i+0] = a[i+0] * b[i+0];
    c[i+1] = a[i+1] * b[i+1];
    c[i+2] = a[i+2] * b[i+2];
    c[i+3] = a[i+3] * b[i+3];
    c[i+4] = a[i+4] * b[i+4];
    c[i+5] = a[i+5] * b[i+5];
}
```

GCC vs Oracle

- GCC gets 3% of maximum
- On average mostly not worth unrolling

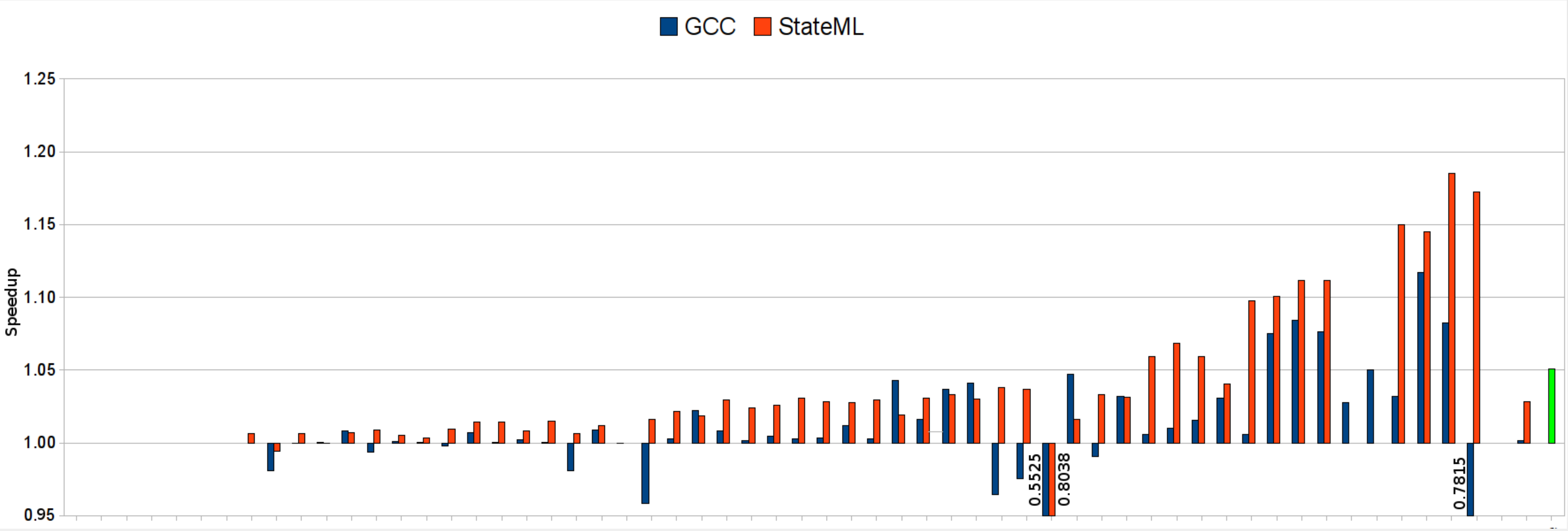


State of the art features

- Lots of good work with hand-built features
  - Dubach, Cavazos, etc
- Stephenson was state of the art
  - Tackled loop unrolling heuristic
  - Spent some months designing features
  - Multiple iterations to get right

GCC vs Stephenson

- Gets 59% of maximum!
- Machine learning does well



GCC vs Stephenson

	GCC	Stephenson
Heuristic	Months	
Features	-	Months
Training	-	Days
Learning	-	Seconds
Results	3%	59%

- To scale up, must reduce feature development time

A feature space for a motivating example

- Simple language the compiler accepts:
  - Variables, integers, '+', '*', parentheses

- Examples:
  - a = 10
  - b = 20
  - c = a * b + 12
  - d = a * ((b + c * c) * (2 + 3))

A feature space for a motivating example

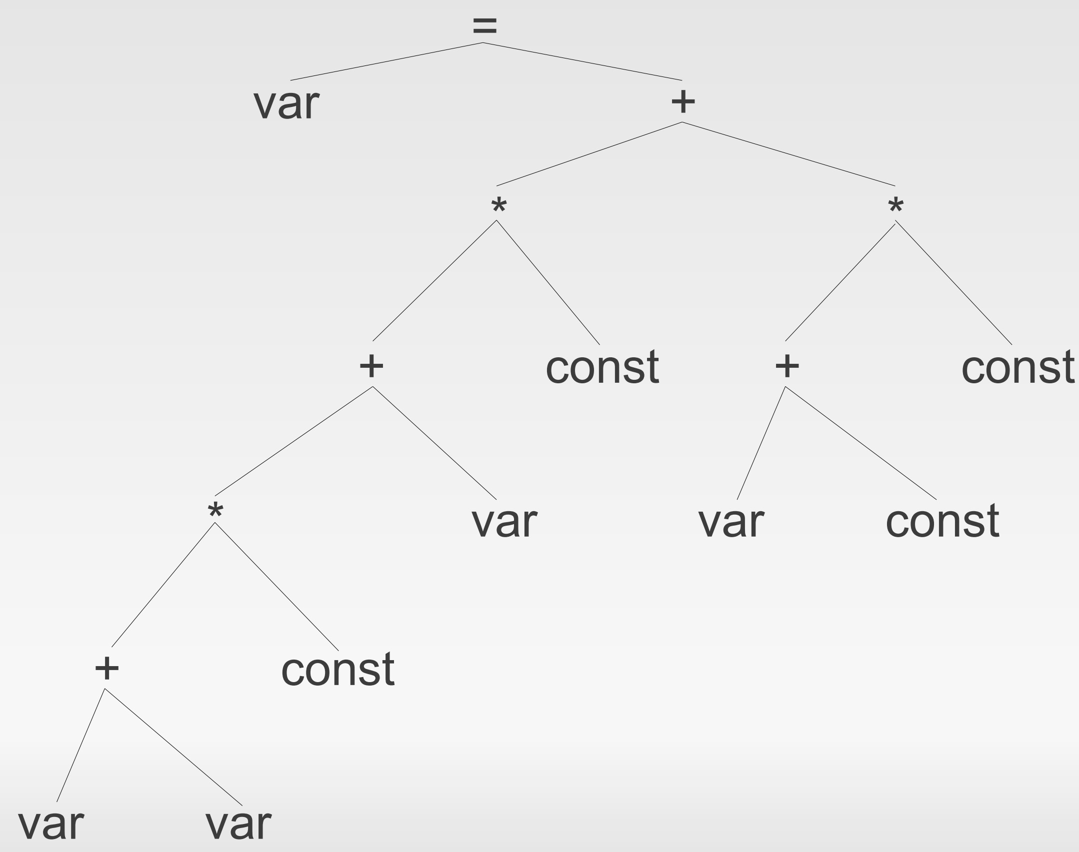
- What type of features might we want?

a = ((b+c)*2 + d) * 9 + (b+2)*4

A feature space for a motivating example

- What type of features might we want?

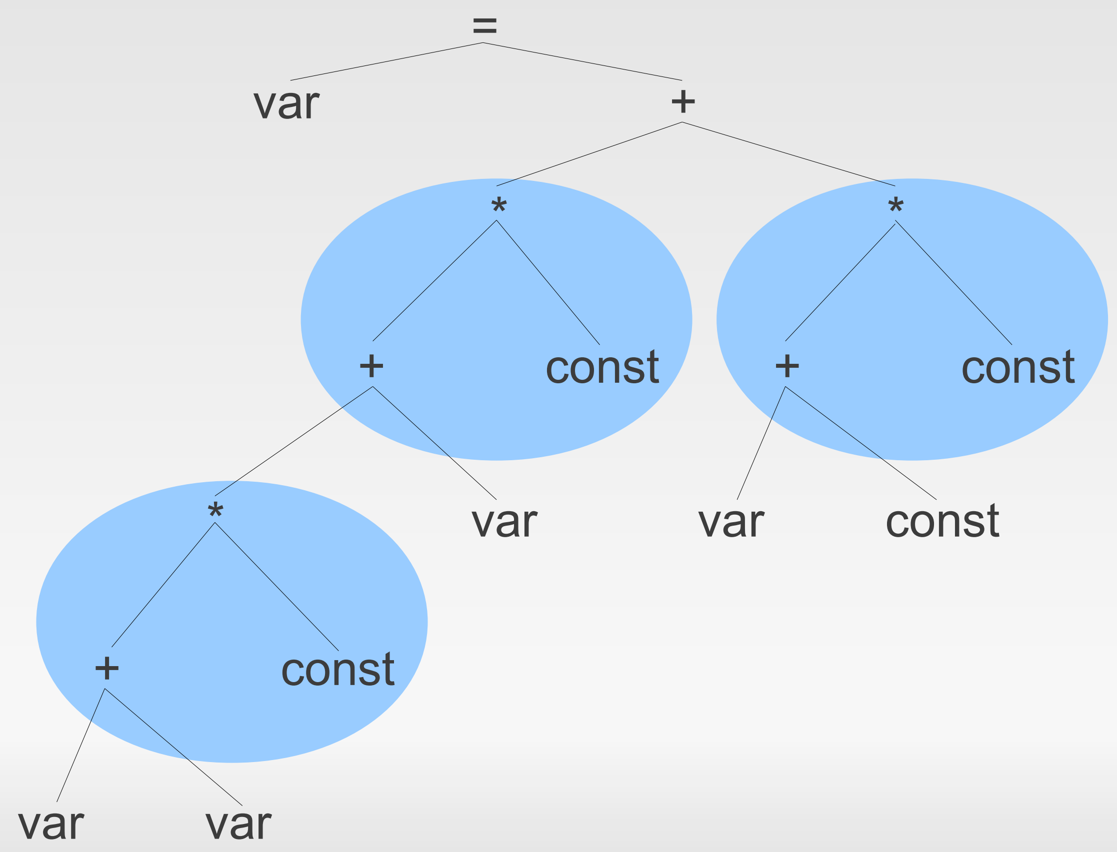
a = ((b+c)*2 + d) * 9 + (b+2)*4



A feature space for a motivating example

- What type of features might we want?

a = ((b+c)*2 + d) * 9 + (b+2)*4



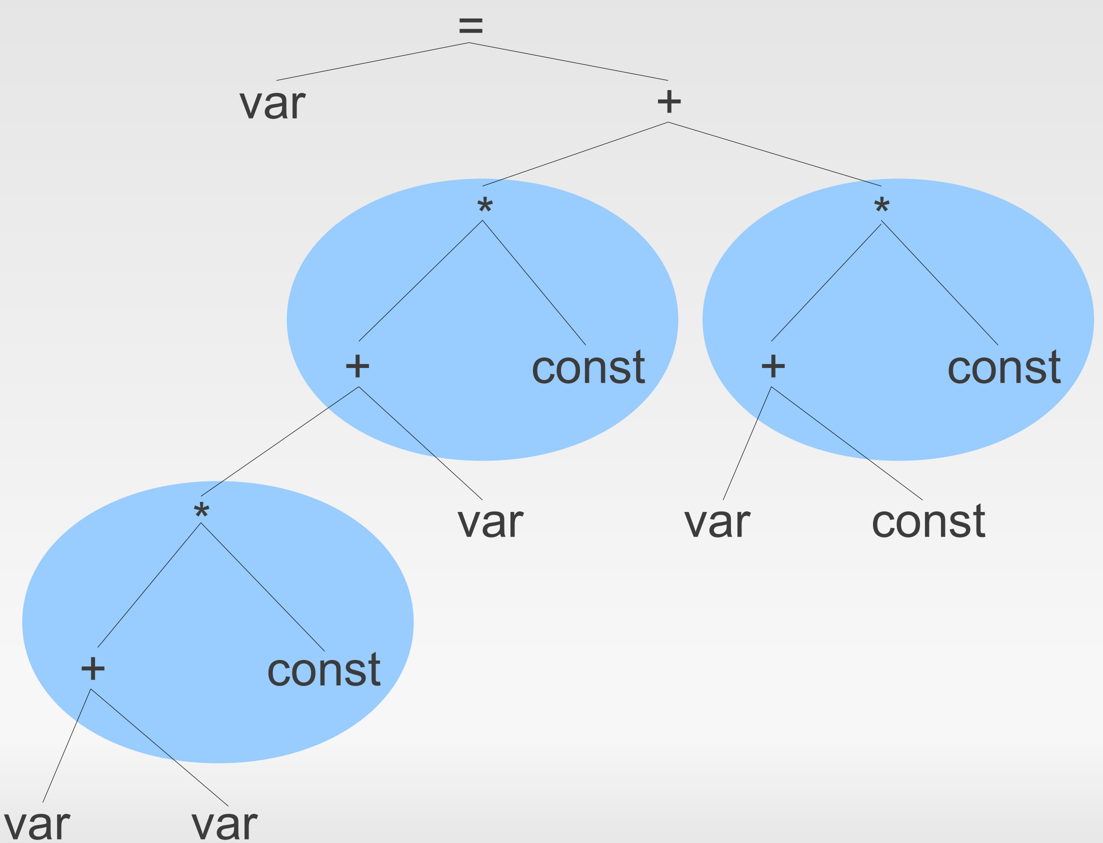
A feature space for a motivating example

- What type of features might we want?

```
count−nodes−matching(
    is−times &&
    left−child−matches(
        is−plus
    )&&
    right−child−matches(
        is−constant
    )
)
```

Value = 3

a = ((b+c)*2 + d) * 9 + (b+2)*4



A feature space for a motivating example

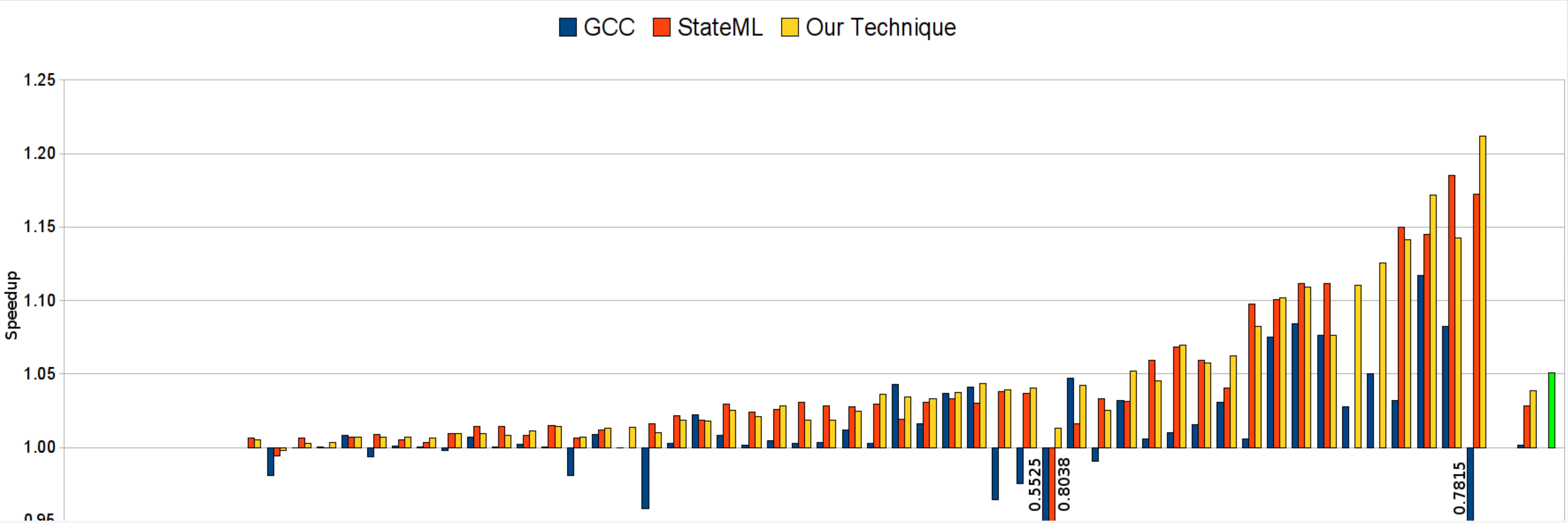
- Define a simple feature language:

```
<feature>   ::= ”count−nodes−matching(” <matches> ”)”
<matches>  ::= ”is−constant”
              |  ”is−variable”
              |  ”is−any−type”
              |  ( ”is−plus” | ”is−times” )
                 ( ”&& left−child−matches(” <matches> ”)” ) ?
                 ( ”&& right−child−matches(” <matches> ”)” ) ?
```

- GCC grammar is huge >160kb
- Genetic search for features that improve machine learning prediction

Results

- GCC 3%   Stephenson 59%   Ours 75%
- Automated features outperform human ones



Results

- Top Features Found

39% ▪ get-attr(@num-iter)

Results

- Top Features Found

39% ▪ get-attr(@num-iter)

14% ▪ count(filter(//*, !(is-type(wide-int) || (is-type(float extend) &&[(is-type(reg)]/count(filter(//*,is-type(int))))) || is-type(union type))))

Results

- Top Features Found

39% - get-attr(@num-iter)

14% - count(filter(//*, !(is-type(wide-int) || (is-type(float extend) &&[(is-type(reg)]/count(filter(//*,is-type(int))))) || is-type(union type))))

8% - count(filter(/*, (is-type(basic-block) && (

```
!@loop-depth==2 ||
(0.0 > (
    (count(filter(//*, is-type(var decl))) -
    (count(filter(//*, (is-type(xor) && @mode==HI))) +
    sum(
        filter(/*, (is-type(call insn) && has-attr(@unchanging))),
        count(filter(//*, is-type(real type)))))) /
        count(filter(/*, is-type(code label)))))))))
```

GCC vs Stephenson vs Ours

	GCC	Stephenson	Ours
Heuristic	Months	-	**-**
Features	-	Months	**-**
Training	-	Days	**Days**
Learning	-	Seconds	**Hours**
Results	3%	59%	**75%**

Machine Learning for Mobile Systems

Mobiles

- Mobile devices will become THE consumer computing platform
- Need to make mobile devices faster
  - Quad cores here already
  - Increased power demand
- Need to make mobile devices lower power
  - Battery life measured in hours
  - Battery capacity not improving

Desktop vs Mobile

- Mobile is a different beast
  - Application characteristics
  - Customers
  - Information available
- Needs different techniques

Desktop vs Mobile

Desktop

- Applications
  - No app store
  - Many languages
  - Opaque binaries

Mobile (Android)

- Applications
  - Central app store
  - Mostly Java
  - Recompilable classes

Desktop vs Mobile

Desktop

- Customers = Devs
  - Training in lab
  - Few benchmarks
  - Bad points OK

Mobile

- Customers = Users
  - Training in wild
  - All applications
  - Bad points, umm, bad

Desktop vs Mobile

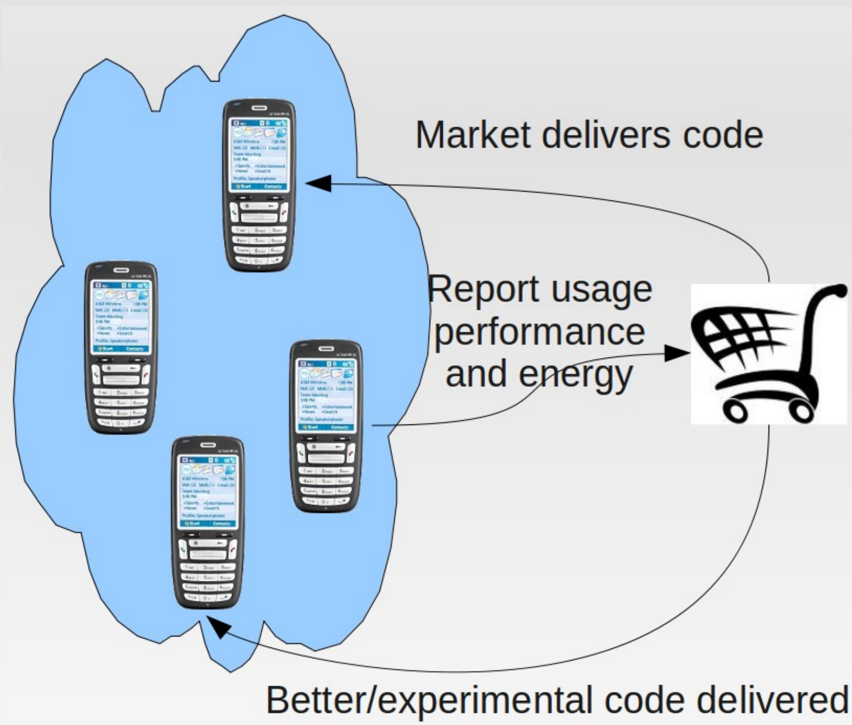
Desktop

- No user knowledge
  - Static code features

Mobile

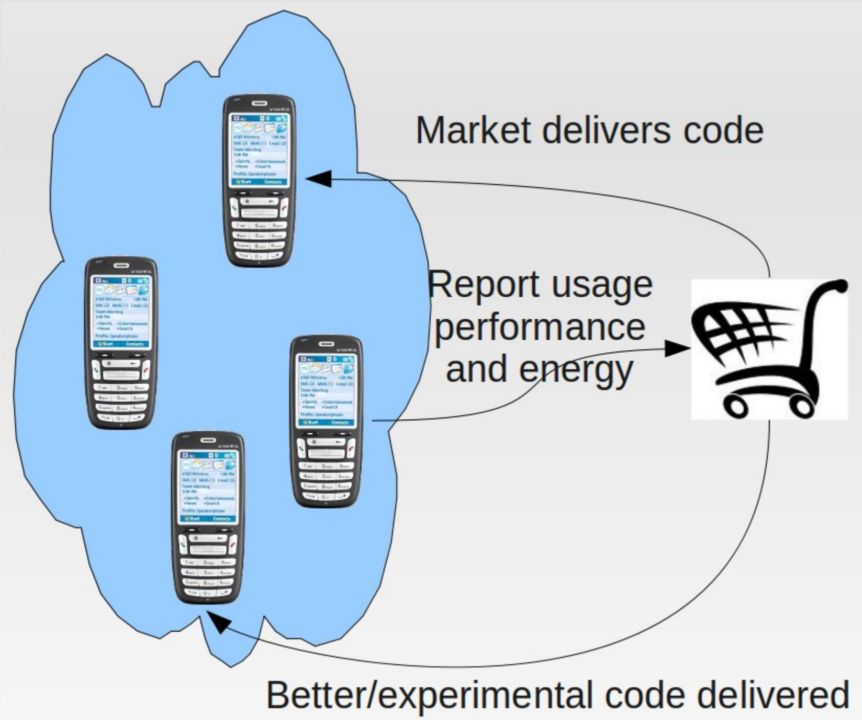
- User knowledge
  - Static code features
  - Application history
  - Geographical
  - Temporal
  - OS states
  - Usage patterns

Optimise applications



- All Android programs use Dalvik JIT - very slow
- Create a market replacement
- Light-weight profiling identifies hot methods
- Updates get experimental code
- System learns how to optimise similar apps for similar users

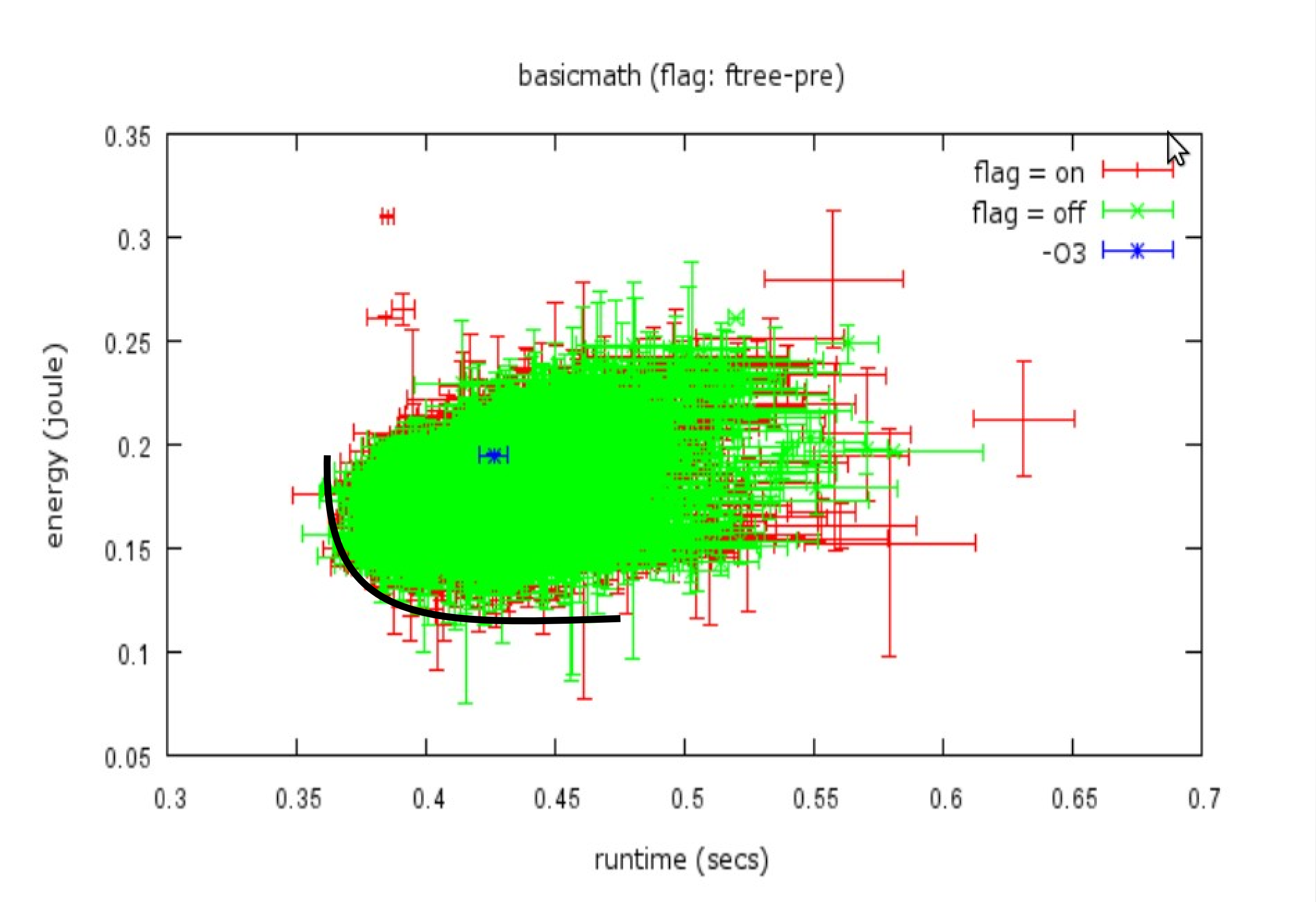
Optimise applications



- Needs zero user impact
  - ML directed profiling
  - ML guided iterative compilation
  - ML guided version selection
- Huge scope for research

Optimise power

- Recharge prediction allows power choices



Other topics

- Optimise communications
- Power modelling
- Scheduling heterogeneous multi-cores
- JIT optimisation

Conclusion

- Machine learning in compilers
  - Choosing Features
  - Enough Benchmarks
  - Cost of Iterative Compilation
  - Compiler Internals
- Machine learning the key to mobile systems
- Mobile is the next big thing
- Huge scope for research

Backup Slides

Compiler internals

- Problem
  - Compilers not built for ML
  - Must access all internals
  - Prior approach was to hack the source
- Solution
  - *lib***Plugin**
  - Opens up GCC internals
  - Modern software engineering
  - Cooperative, extensible plug-ins, with AOP
  - Plug-ins now adopted in GCC

Cost of iterative compilation

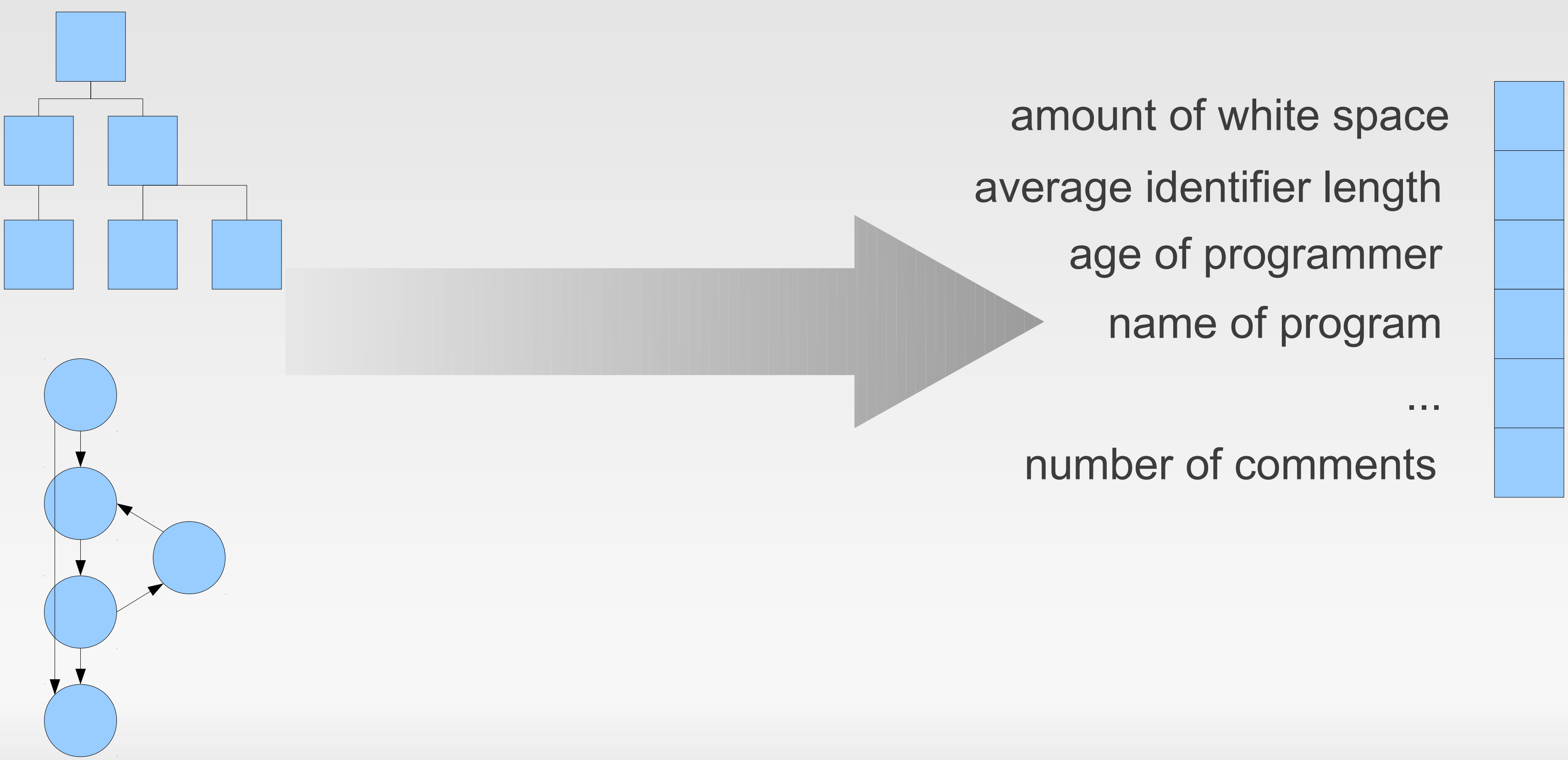
- Problem
  - Gathering training data can take months
  - Statistical soundness often overlooked
- Solution
  - *Profile Races* *(H.Leather, B.Worton, M.O'Boyle)*
  - Program version race each other, losers quit early
  - Reduces training time by order of magnitude
  - Ensures statistically sound data

Enough benchmarks

- Problem
  - ML would like 10^5 examples
  - Only got a few dozen benchmarks
- Solution
  - *Automatic Benchmark Generation*
    (H.Leather,Z.Wang,A.Magni,C.Thompson - In preparation)
  - Genetic programming + constraint satisfaction to make 'human like' programs
  - Active learning to cover the training space

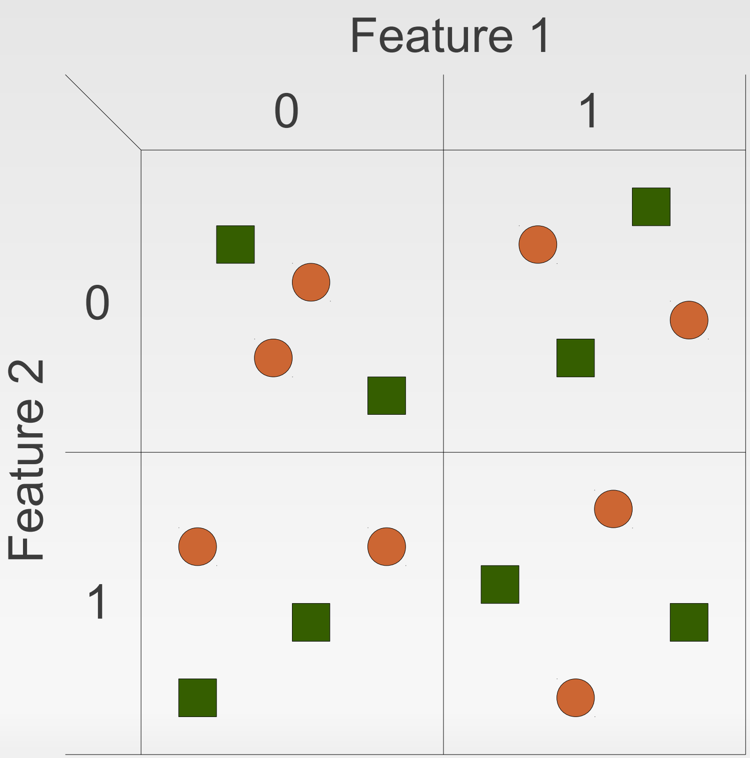
Difficulties choosing features

- The expert must do a good job of projecting down to features



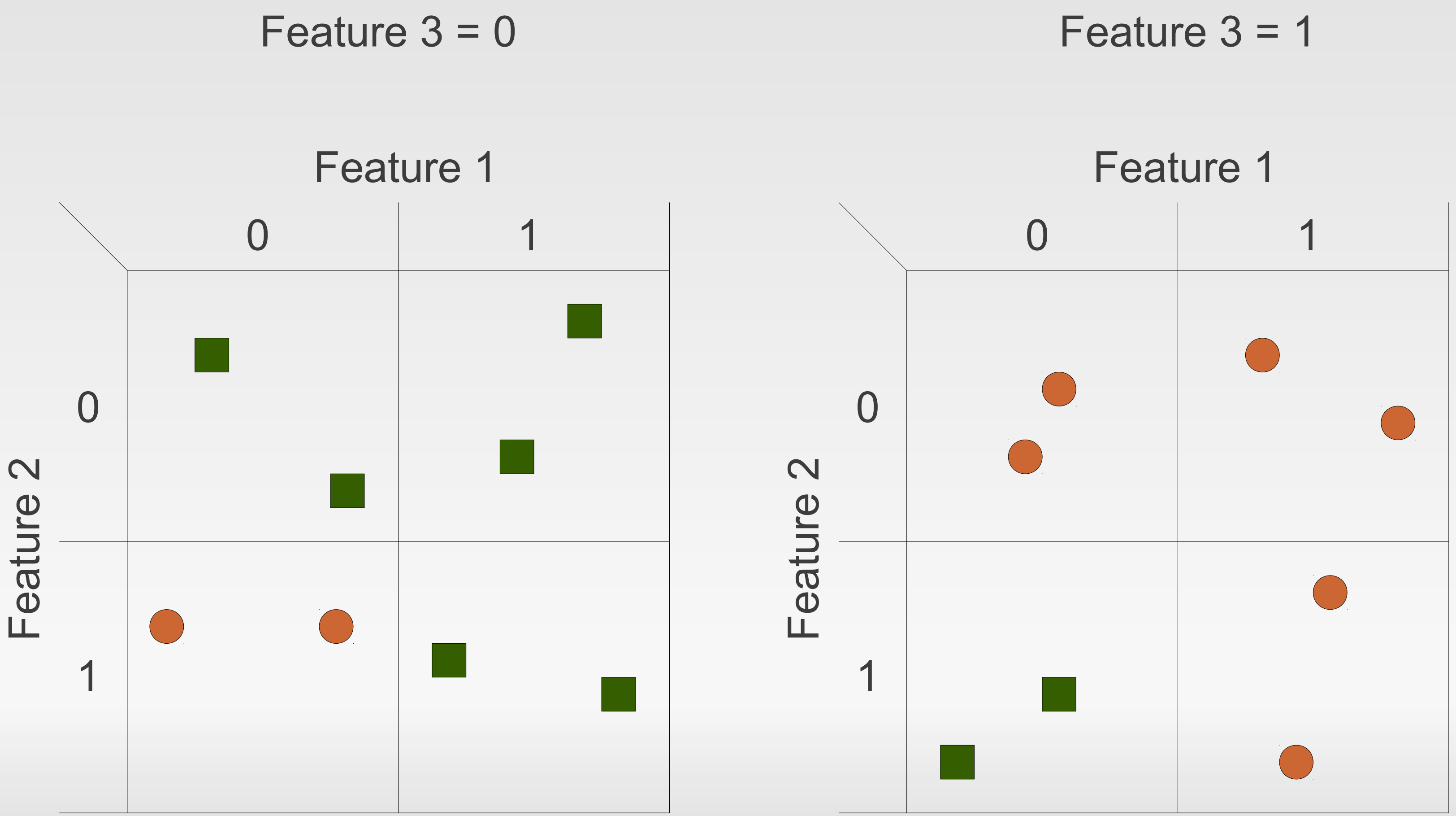
Difficulties choosing features

- Machine learning doesn't work if the features don't distinguish the examples



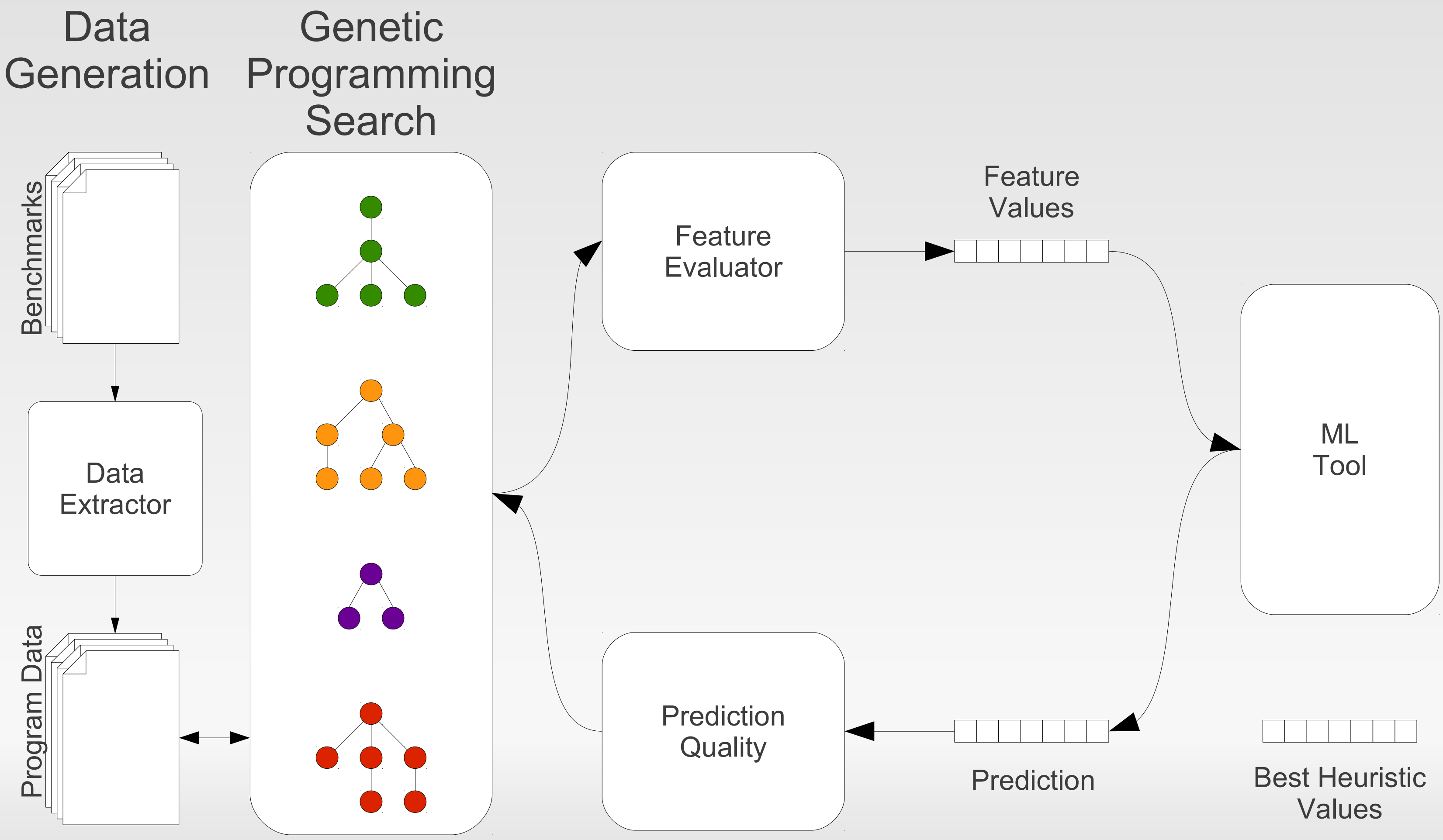
Difficulties choosing features

- Better features might allow classification



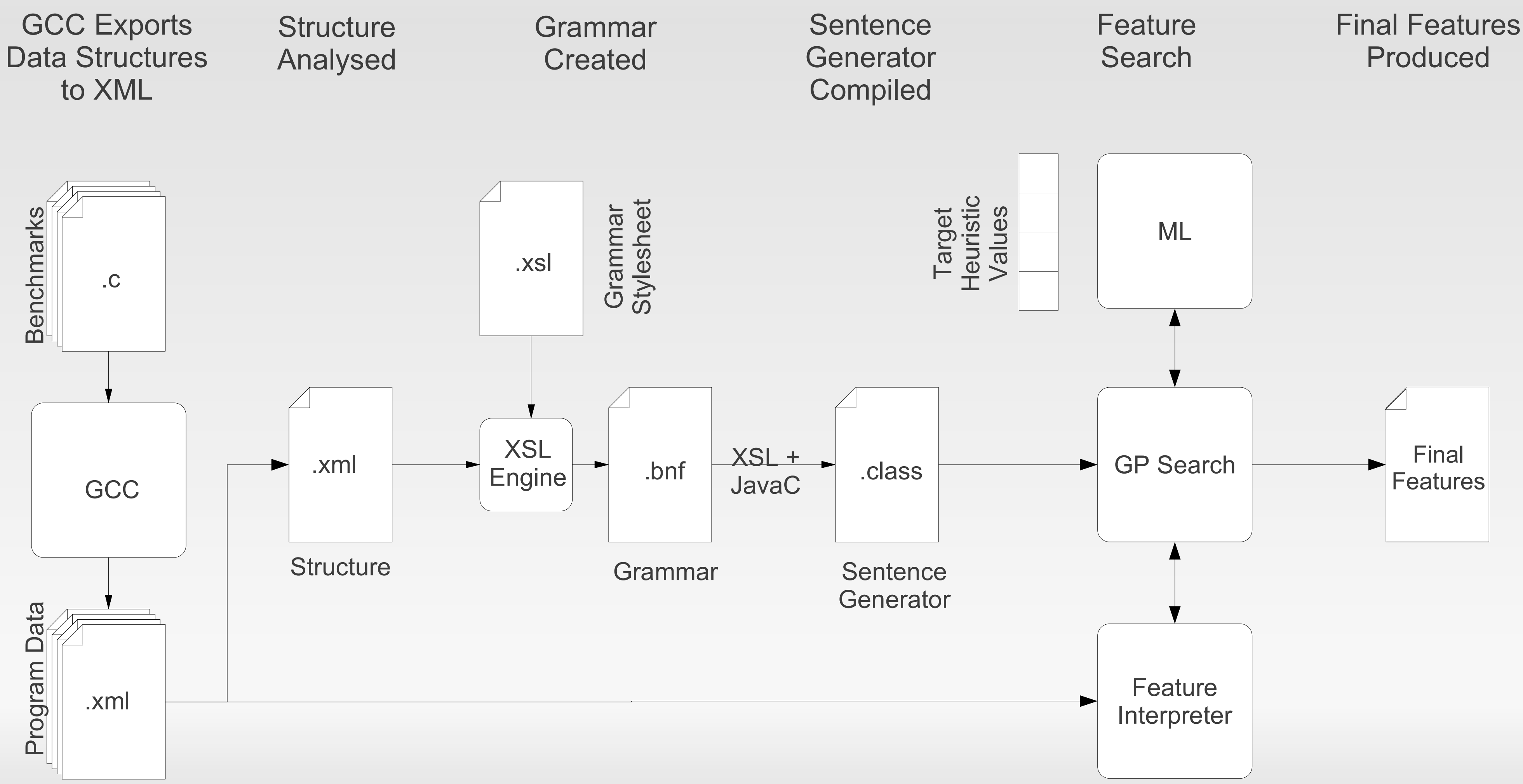
Searching the Feature Space

- Overview of searching



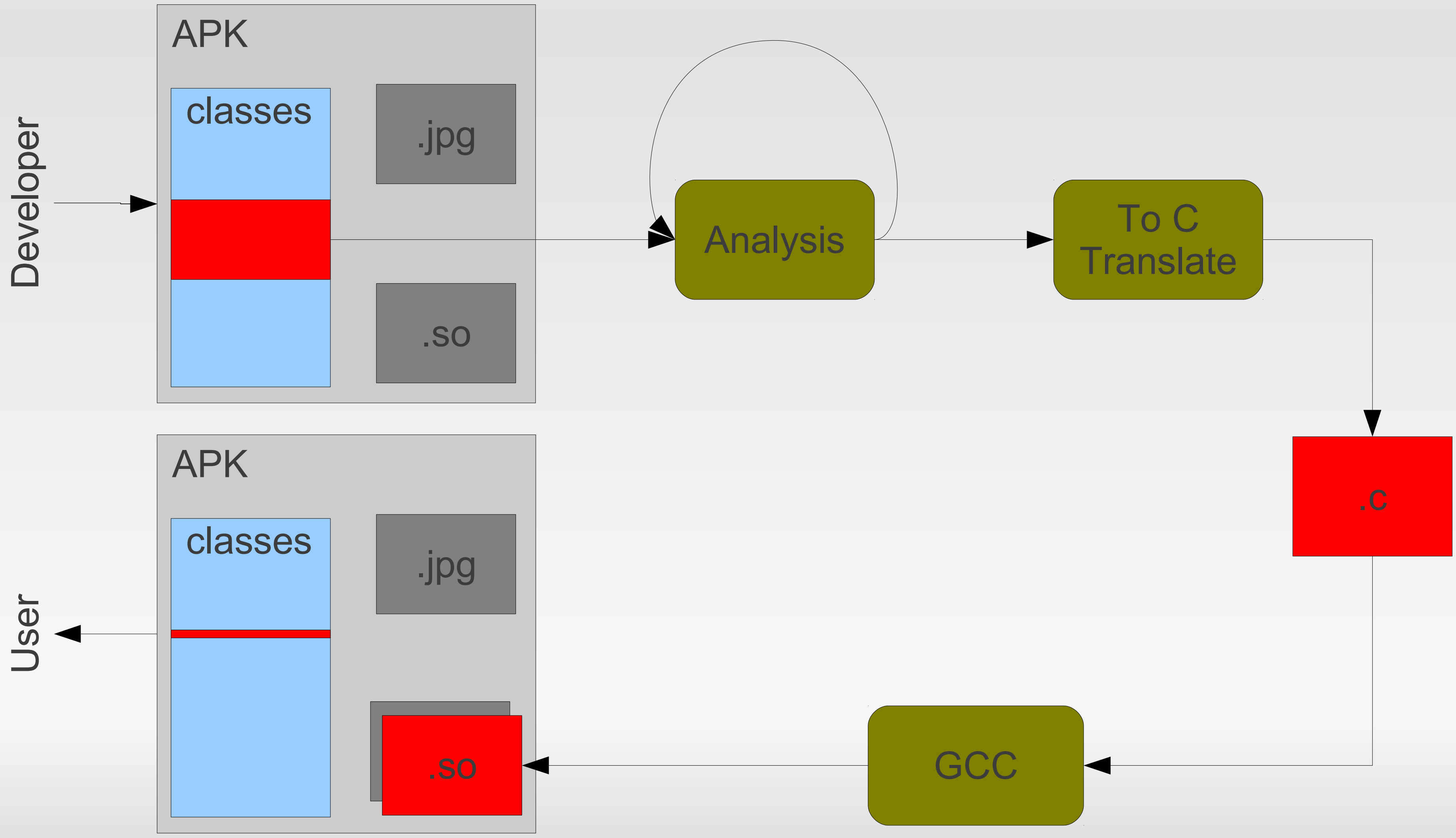
Features for GCC

- Overview of grammar production

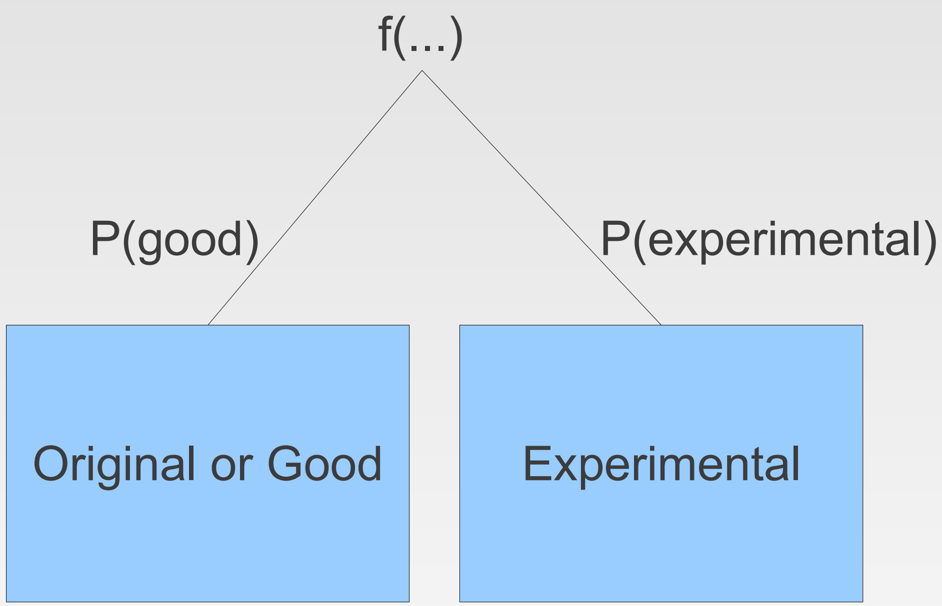


ML for Mobile

- Server side native compilation of hot methods

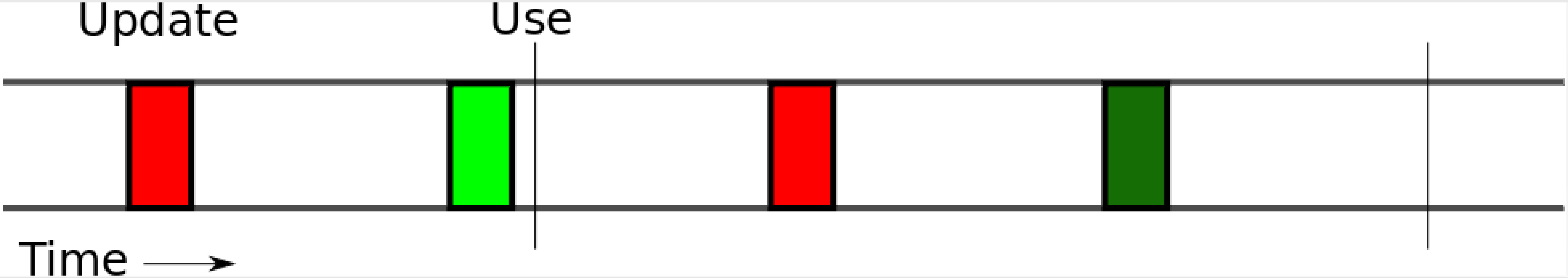


ML for Mobile - Downsizing Down Sides



- Experiments on real users' phones
- What about the bad search points?
- Multiple versions - known good and experimental
- P(experimental) ∝ confidence(experimental)

Optimise communications



- Comms updates are expensive
- Updates need to be fresh and not wasted
- Build cost models
- Predictor 'use' times
- Schedule updates for predicted lowest cost

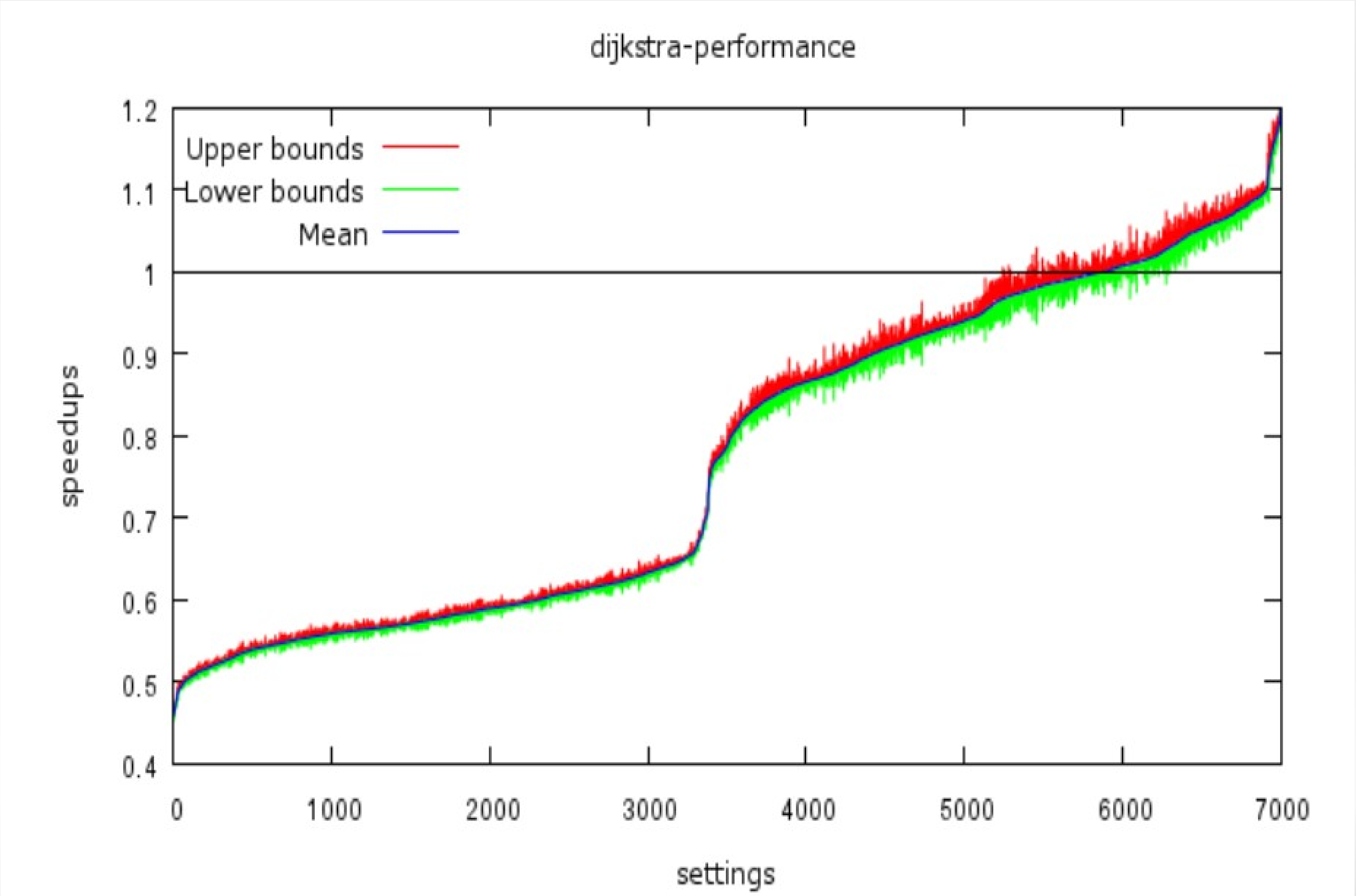
Power models

- Need power models
  - Energy sensors are low fidelity
  - Batteries non-linear
  - Allow relaxation
  - Lowest power solution may not give longest battery life
- Power aware workloads needed

Heterogeneous multi-cores

- Simple heterogeneous multi-cores here now
- Scheduling is NP-hard
  - Even when application characteristics known
- Use ML to tune scheduling heuristics for power and performance

Performance - Dijkstra



Performance

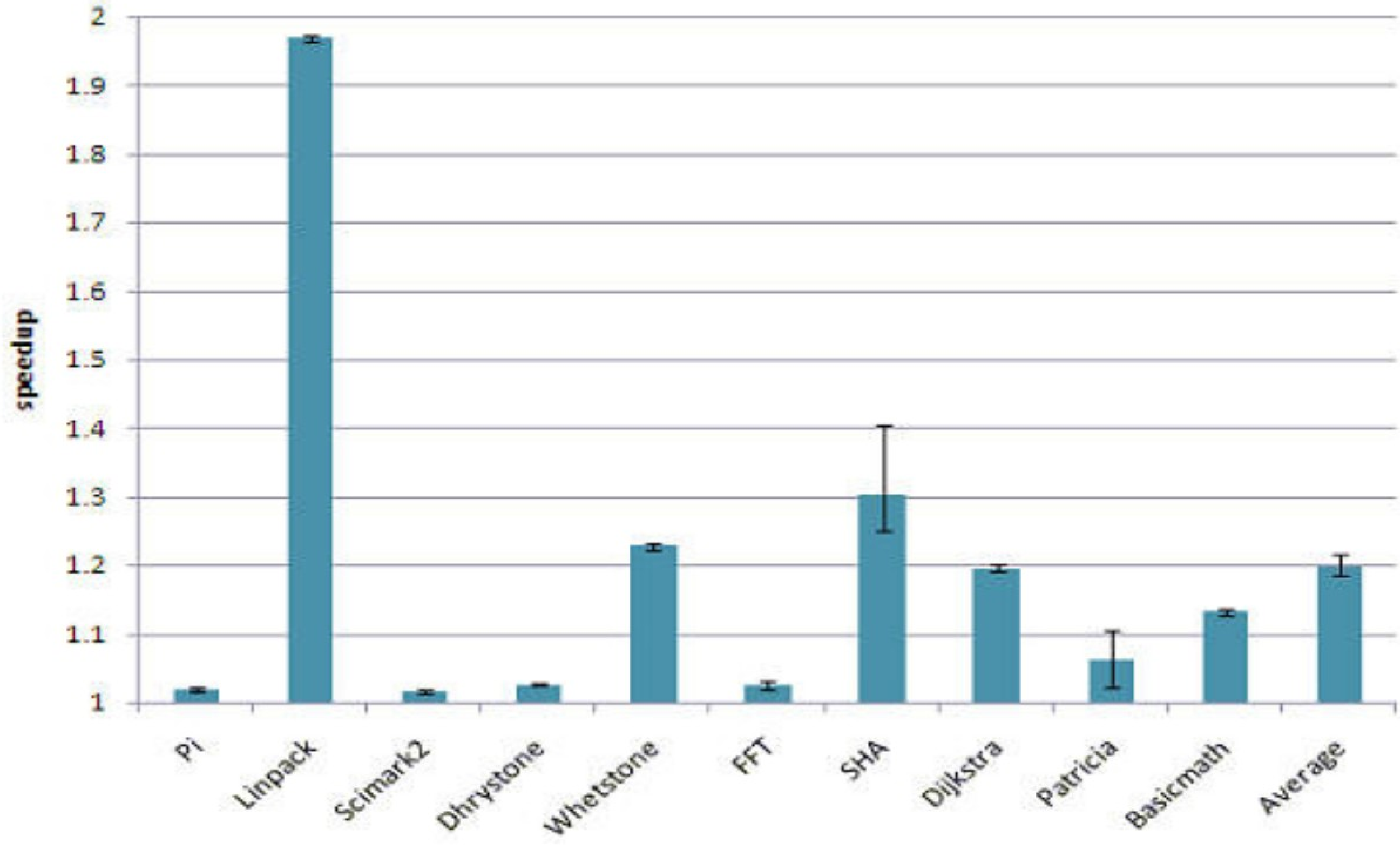


Figure 5.1: Maximum speedups of runtime

Energy

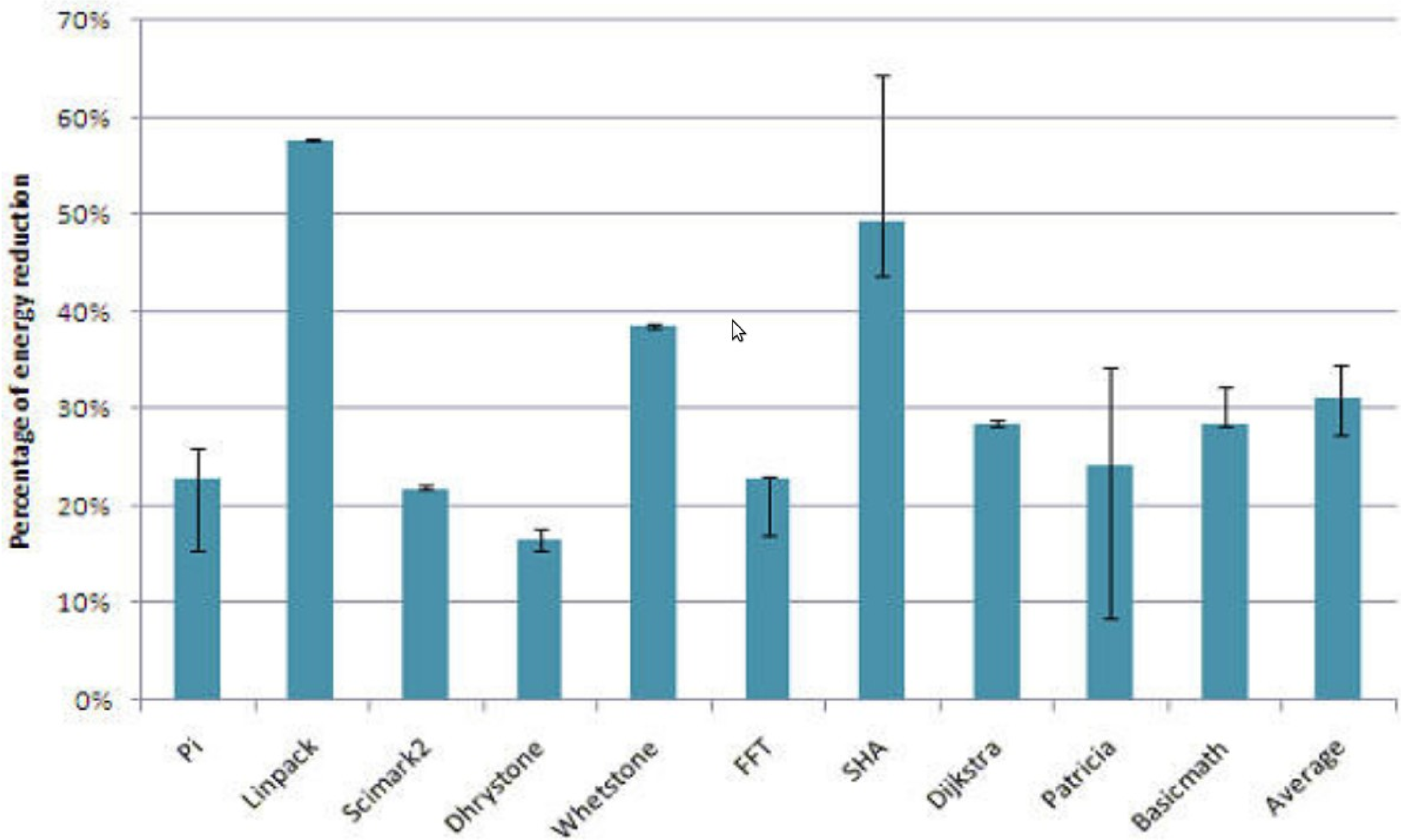
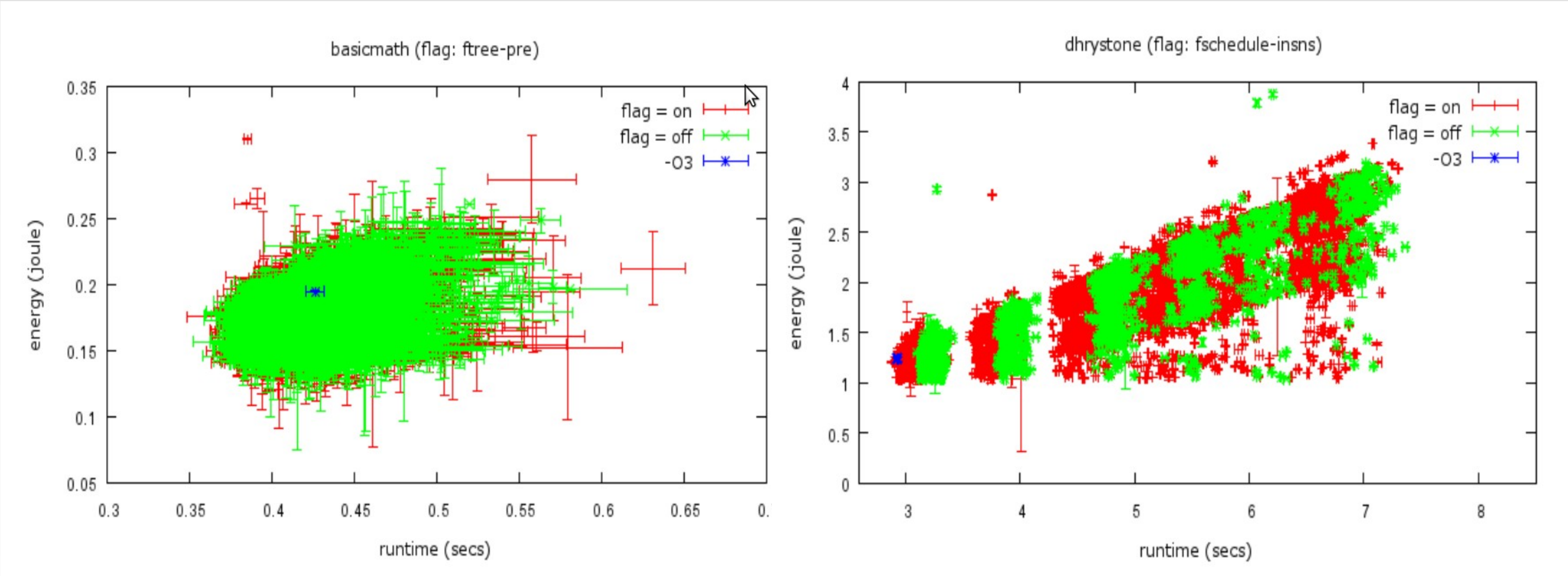


Figure 5.2: Maximum energy improvement rates

Energy vs Performance

- Are energy and performance correlated?



- Not really! Why?
- If could predict recharge time, change version