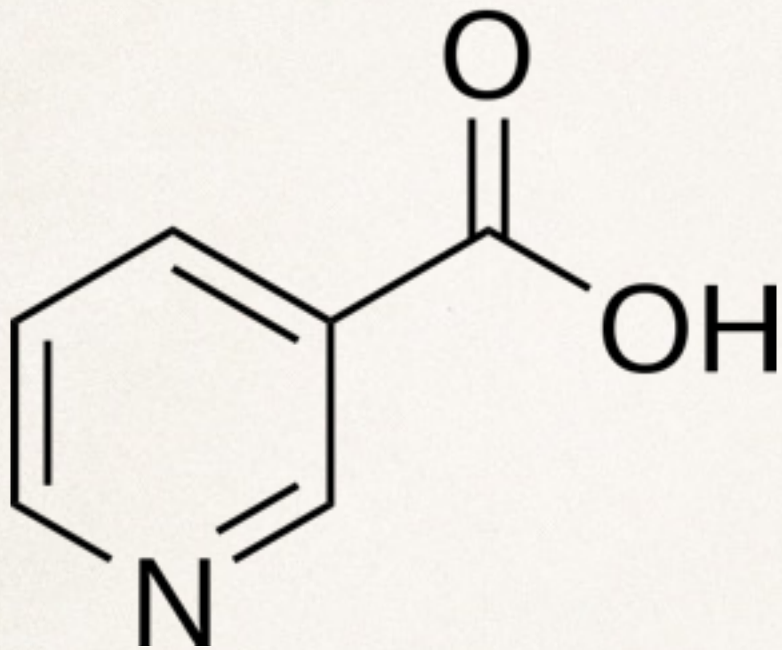


Amortised Optimisation as a Means to Achieve Genetic Improvement

Hyeongjun Cho, Sungwon Cho, Seongmin Lee, Jeongju Sohn, and **Shin Yoo**

Date 2017.01.30, The 50th CREST Open Workshop

Offline Improvement

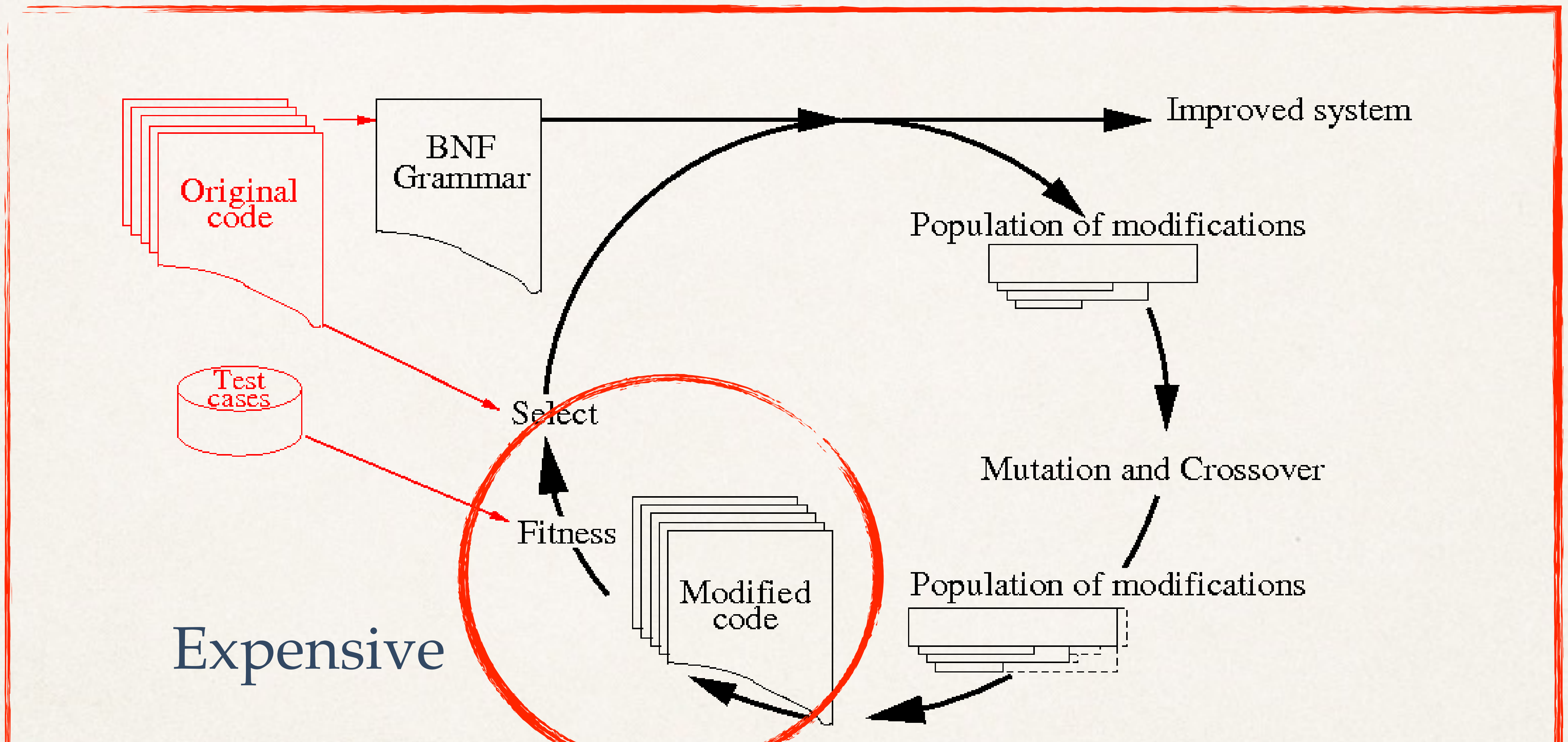
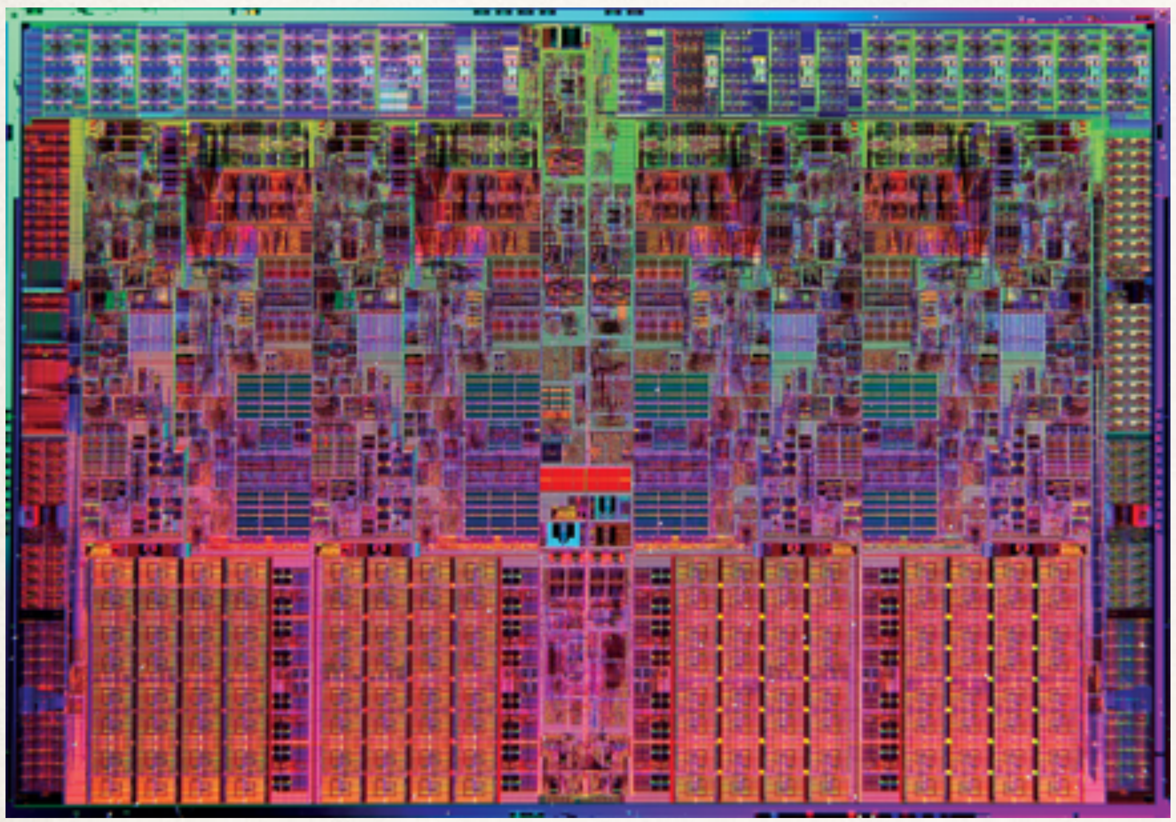


Fig. 1. GP improvement of MiniSAT.

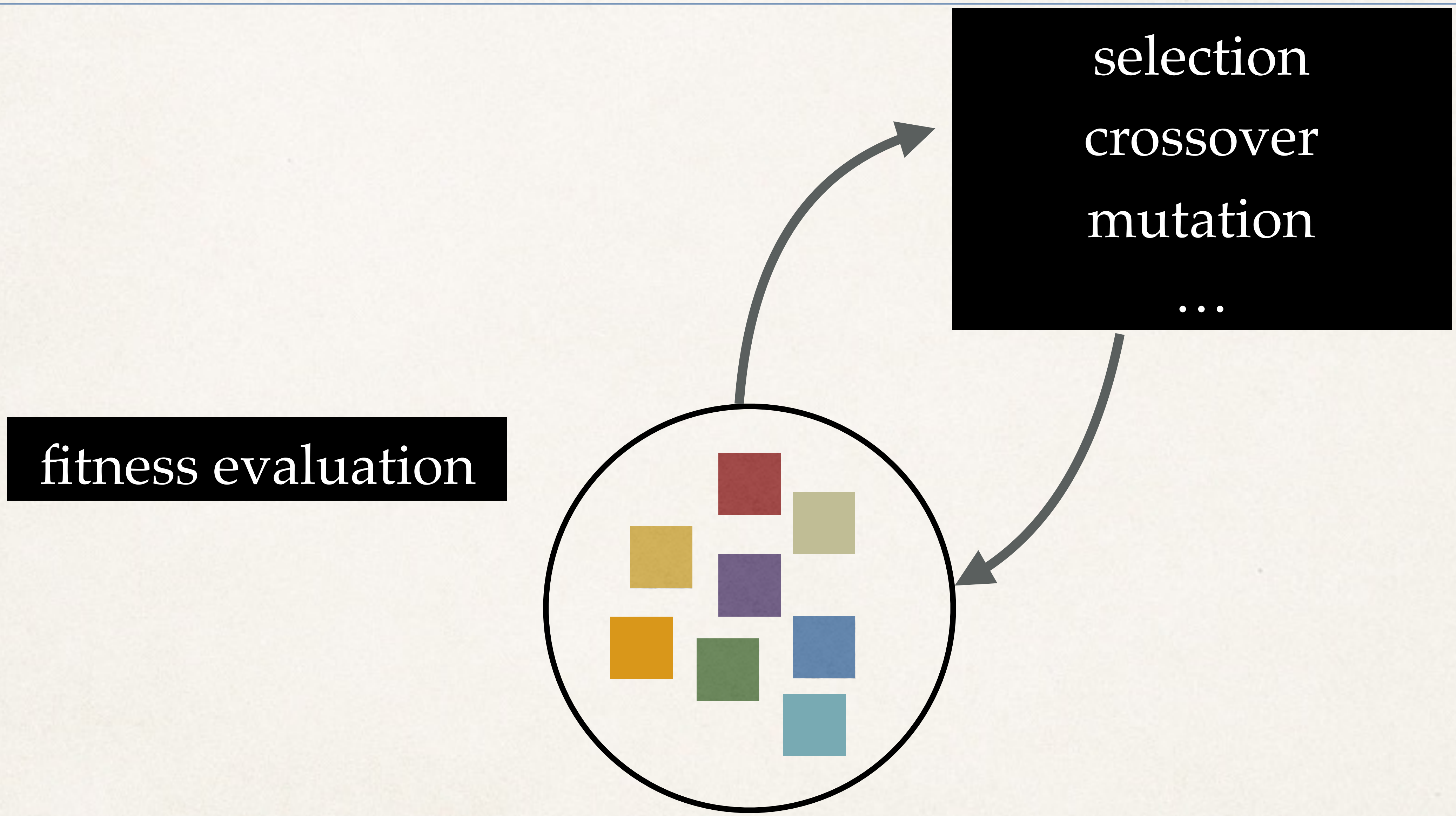
Tied to offline environment

Environmental Factors



We cannot anticipate the environment that the software will be executed; hence it is hard to optimise for it.

Offline Optimisation



One Generation

Amortised Optimisation

selection
crossover
mutation
…

Persistence Layer

Optimisation executed in micro-steps,
each in-situ execution as a single fitness evaluation

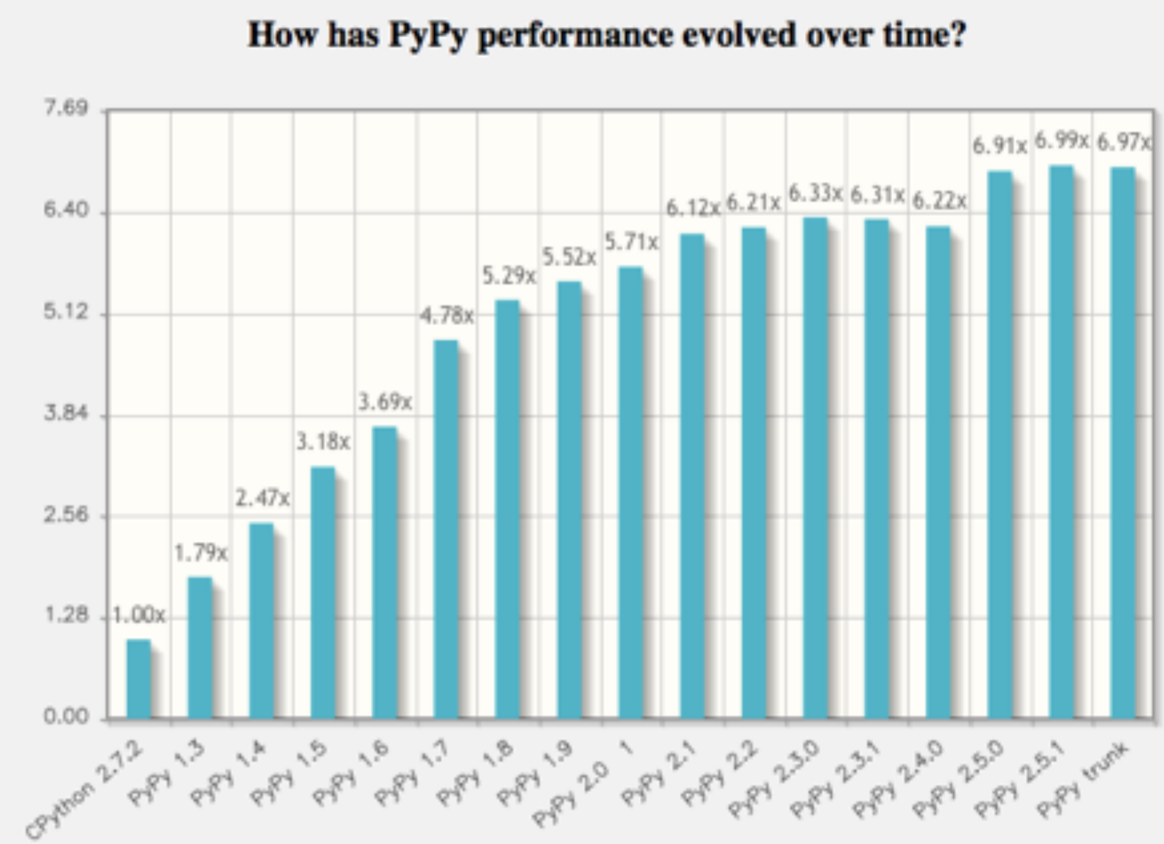
Amortised Optimisation

Genetic Improvement, out in the wild!

Budget Controlled
(will stop when run out)

Low Overhead
(only microscopes)

Does it work?



Plot 2: Speedup compared to CPython, using the inverse of the geometric average of normalized times, out of 20 benchmarks (see paper on why the geometric mean is better for normalized results).

We applied amortised optimisation to pypy,
a tracing-JIT based python implementation.

Tracing JIT Parameters

When to begin tracing?

When to mark as hot?

When to compile the bridge?

PIACIN

1. Install the package.
2. Import the package
3. There is no step 3.

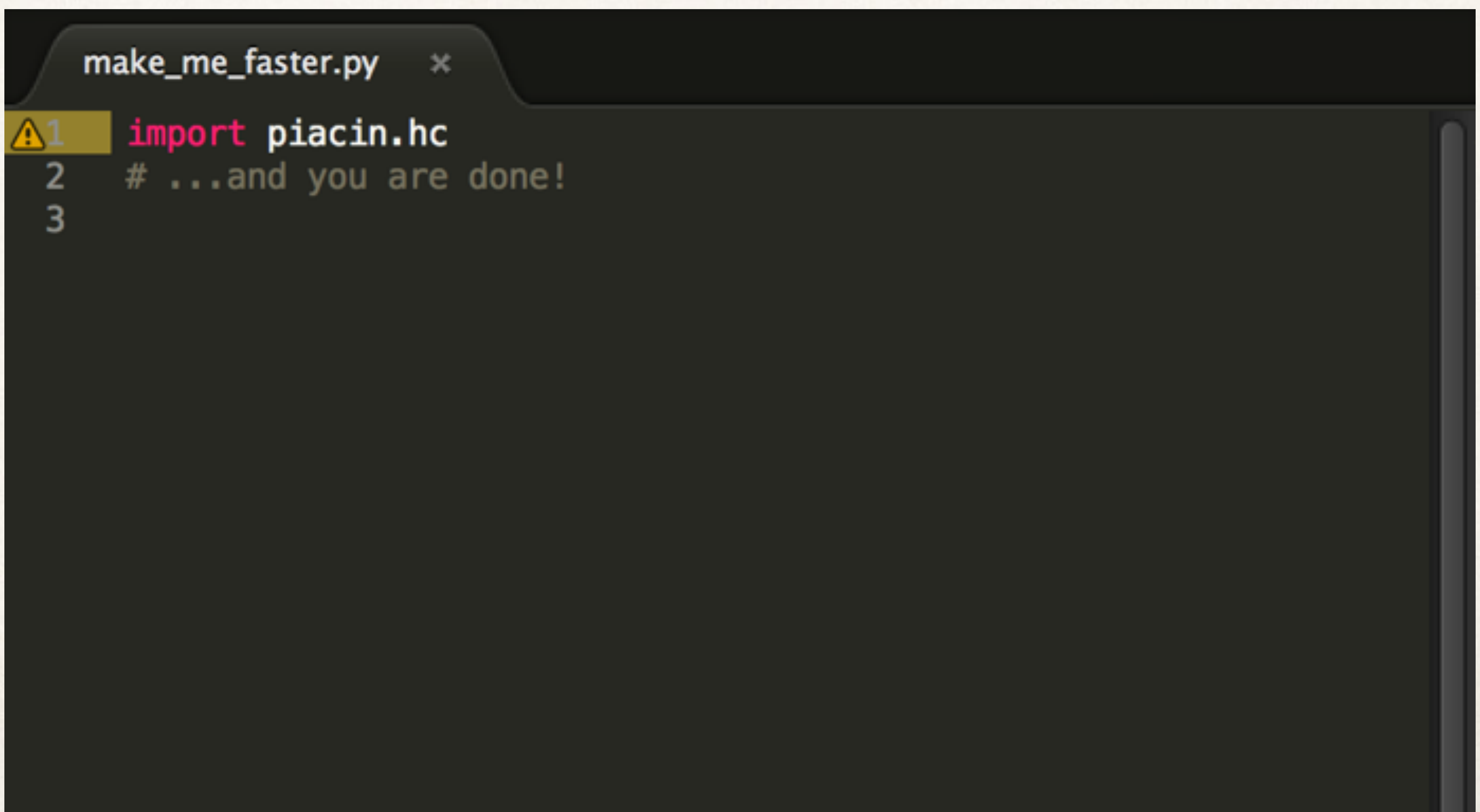


Table 1. Benchmark user scripts used for the JIT optimisation case study

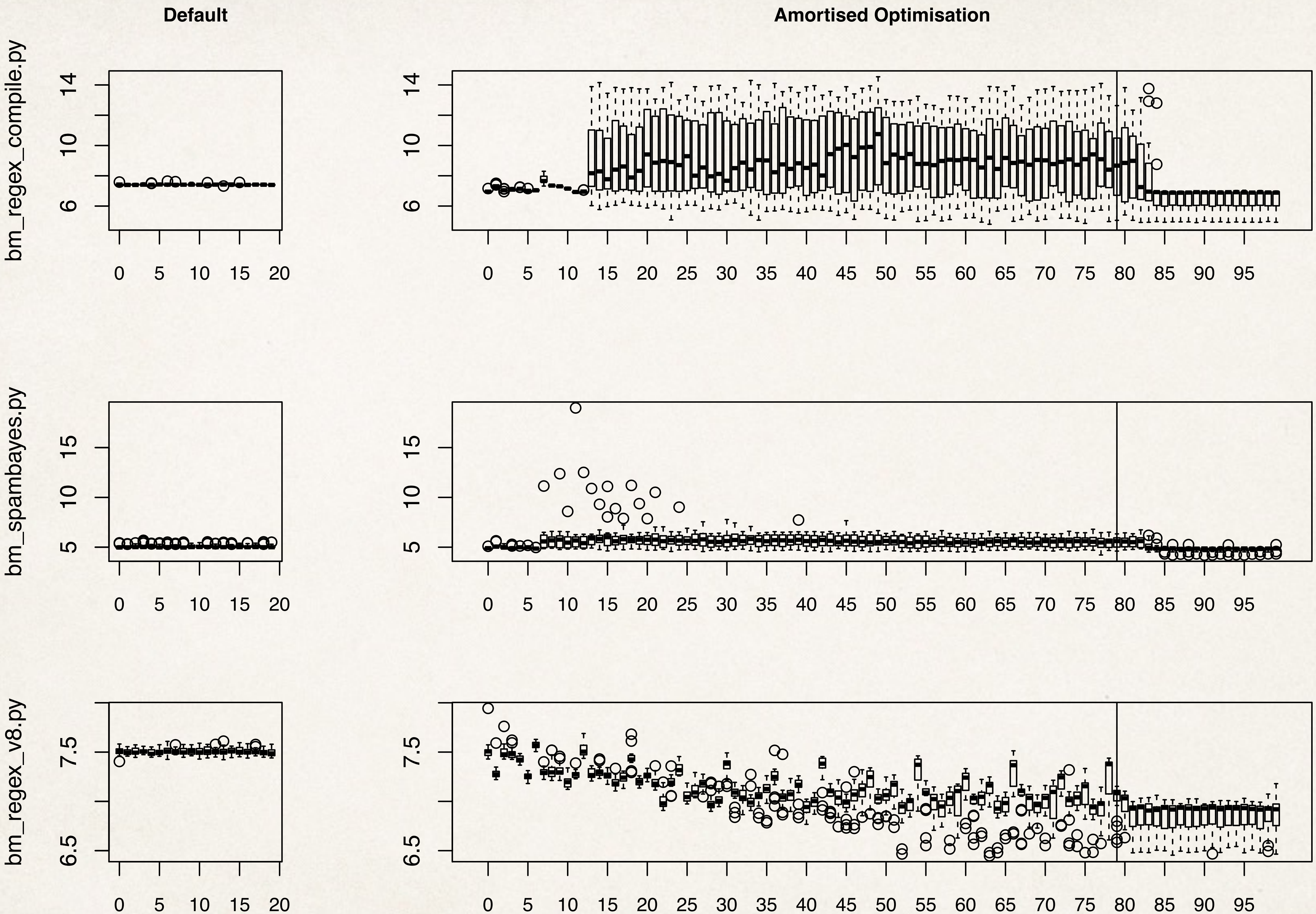
Script	Description
`bm_call_method.py`	Repeated method calls in Python
`bm_django.py`	Use `django` to generate 100 by 100 tables
`bm_nbody.py`	Predict n-body planetary movements[a]
`bm_nqueens.py`	Solve the 8 queens problem
`bm_regex_compile.py`	Forced recompliations of regular expressions
`bm_regex_v8.py`	Regular expression matching benchmark adopted from `V8`[b]
`bm_spambayes.py`	Apply a Bayesian spam filter[c] to a stored mailbox
`bm_spitfire.py`	Generate HTML tables using `spitfire`[d] library

[a] Adopted from `http://shootout.alioth.debian.org/u64q/benchmark.php?test=nbody&lang=python&id=4`.

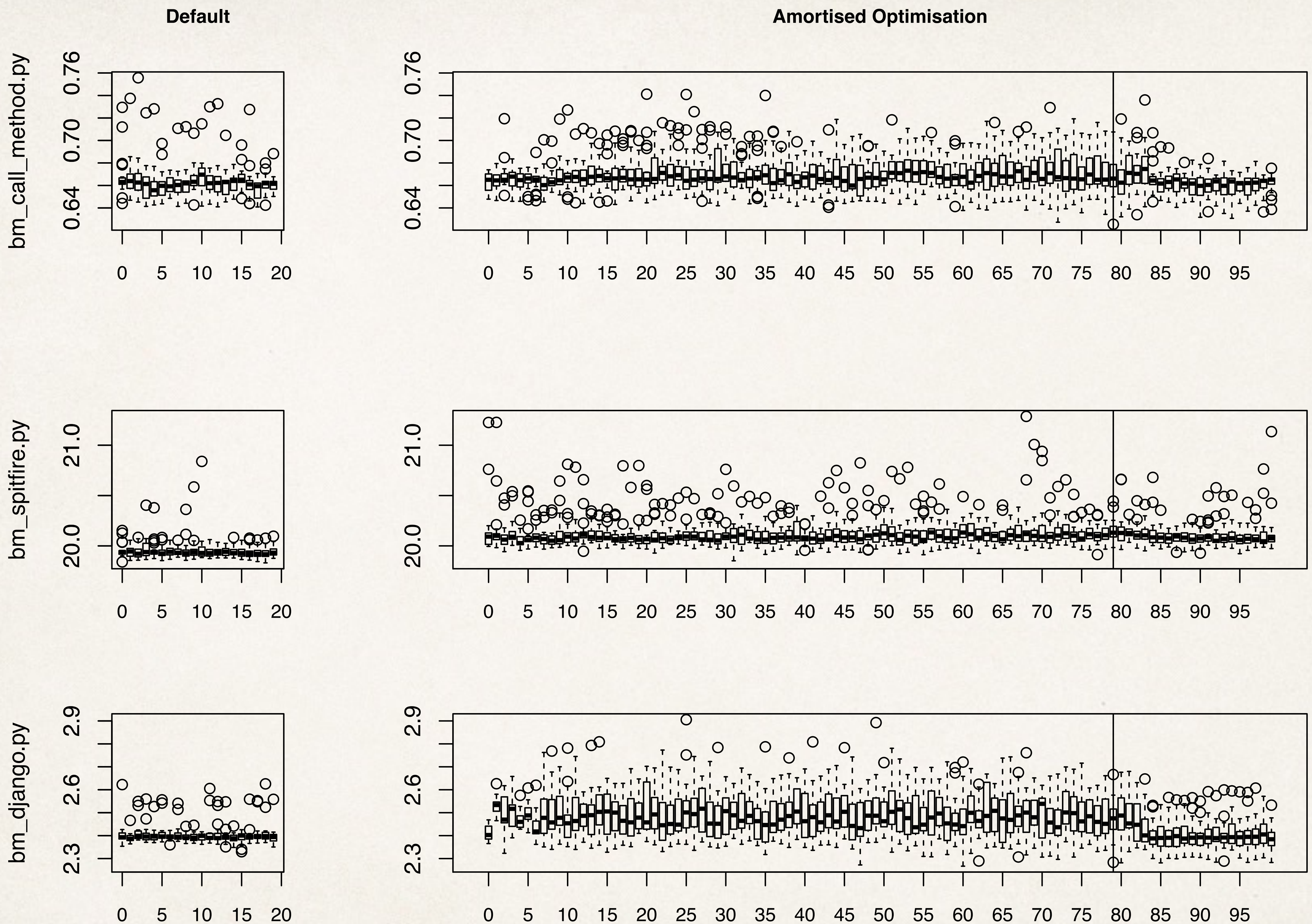
[b] Google's Javascript Runtime: `https://code.google.com/p/v8/`.

[c] `http://spambayes.sourceforge.net`

[d] A template compiler library: `https://code.google.com/p/spitfire/`



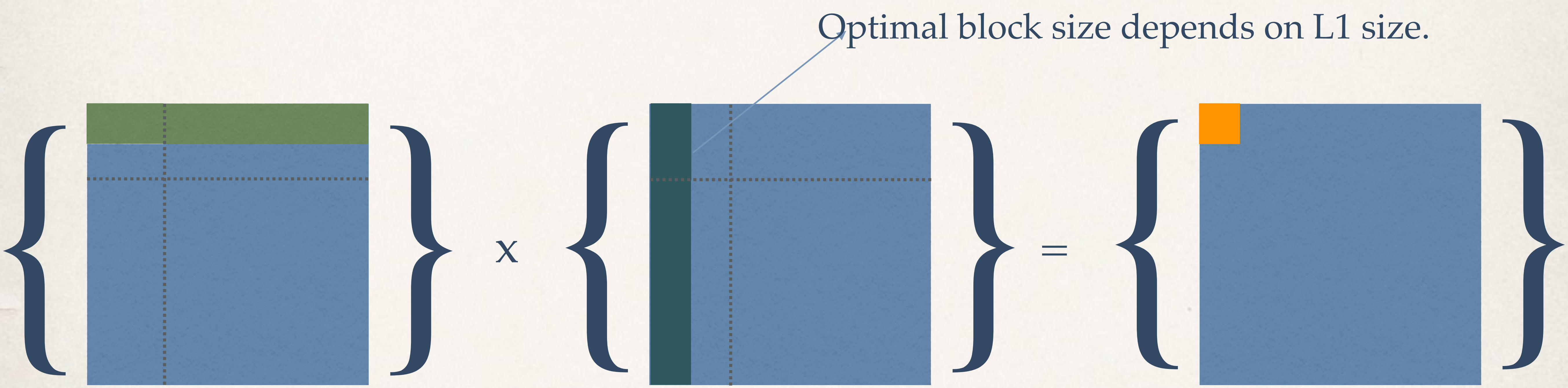
Default
Amortised Optimisation
bm_regex_compile.py
bm_spambayes.py
bm_regex_v8.py



Default
Amortised Optimisation
bm_call_method.py
bm_spitfire.py
bm_django.py

How about hardware?

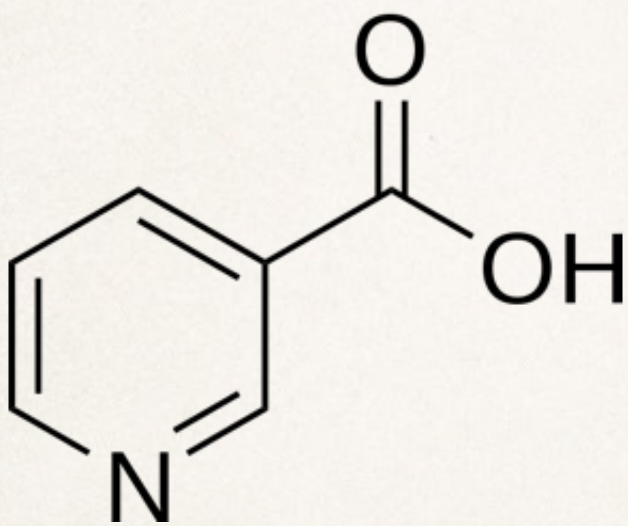
Let us consider matrix multiplication.



Blocked Matrix Multiplication: smaller inner loop
to fit everything into L1 cache.

NIA^{3}CIN

Non-Invasive, Amortised and Autonomous Code Injection



BlockedMatrixMultiplication.java

```
@Input(name = "block_size", initvalue = "2", bound = "1, 512")
public void setBlockSize(int size)
{
    this.BLOCK_SIZE = size;
}

@Optimize(name = "rate", type = Double.class, direction = Optimi
public Double getRate()
{
    return new Double(rate);
}
```

Annotation-based

```
public BlockedMatrixMultiplication()
{
    Niacin.initialise(BlockedMatrixMultiplication.class);
}
```

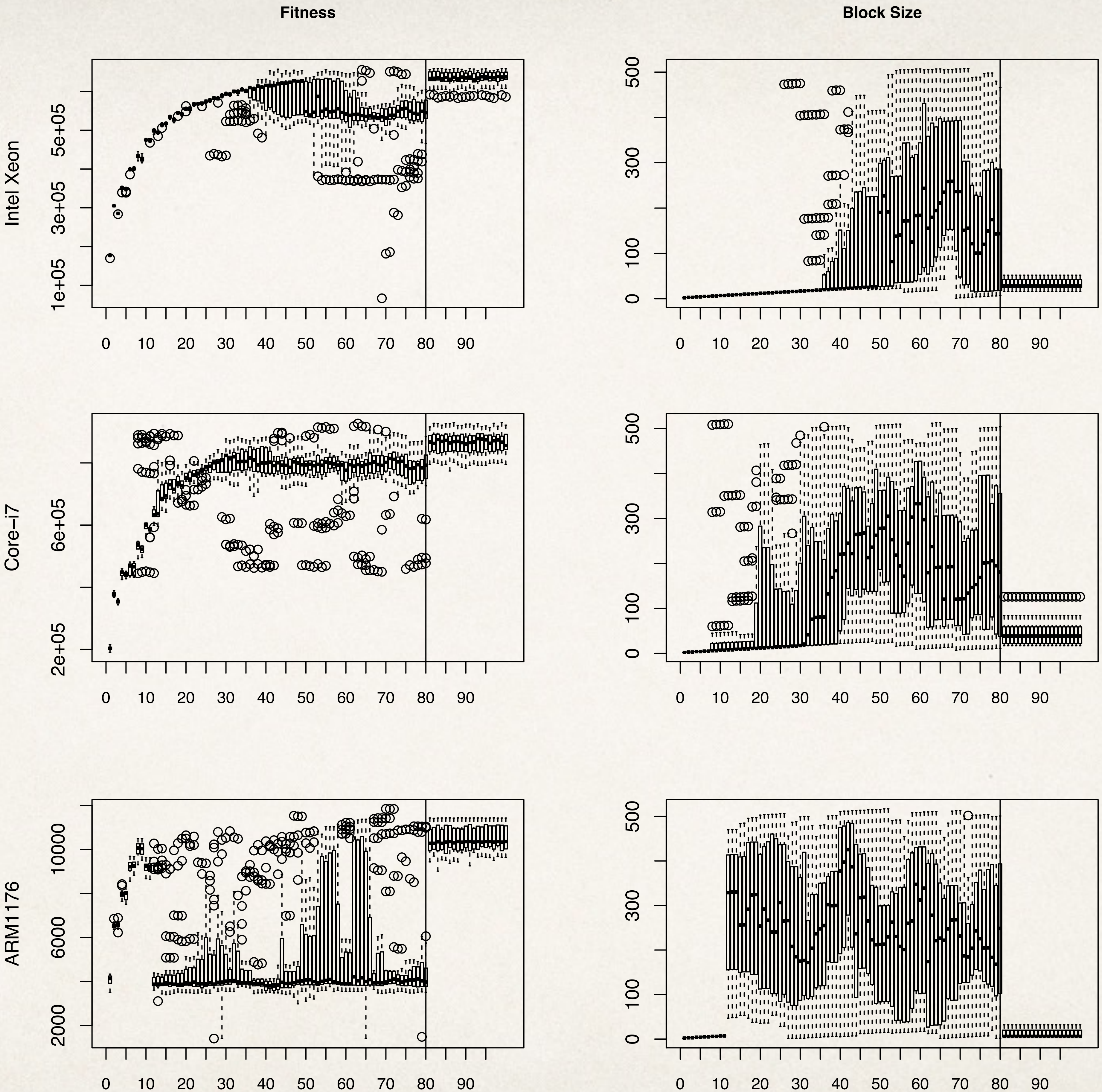
Event-driven dependency injection

Evaluation

Table 3. Information about CPUs for which BMM was optimised

CPU	Clock Frequency	L1 Instruction Cache	L1 Data Cache
Intel Xeon W3680[a]	3.33GHz	32KB	32KB
Intel Core-i7 3820QM[a]	2.7GHz	32KB	32KB
ARM1176 (BCM2835 SoC)[b]	250MHz	16KB	16KB

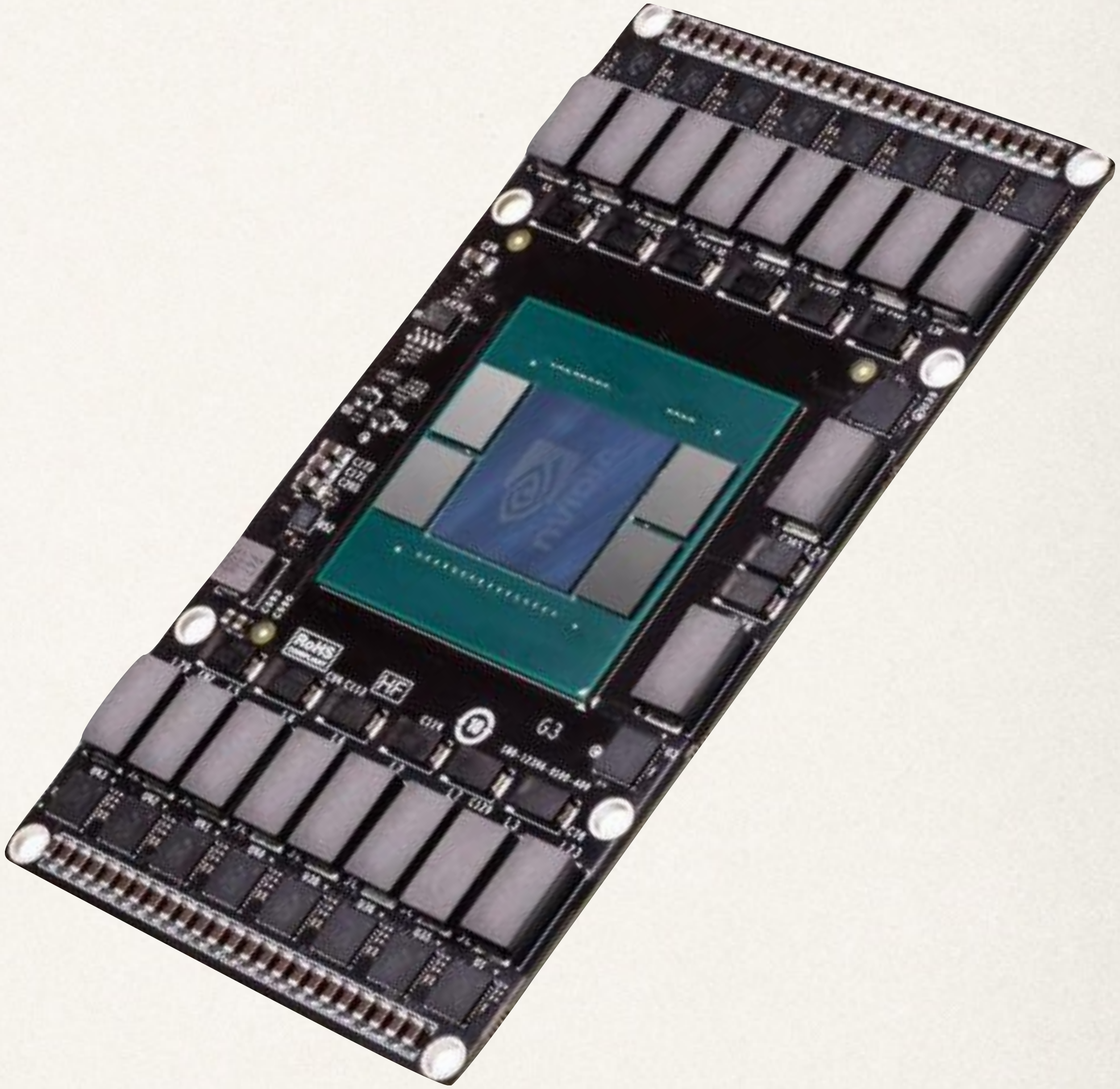
[a] These Intel CPUs share data and instruction caches between two processor threads.
[b] Raspberry Pi Model B, first edition.



Fitness
Block Size
Intel Xeon
Core−i7
ARM1176
1e+05
3e+05
5e+05
2e+05
6e+05
2000
6000
10000
0
100
300
500
0 10 20 30 40 50 60 70 80 90

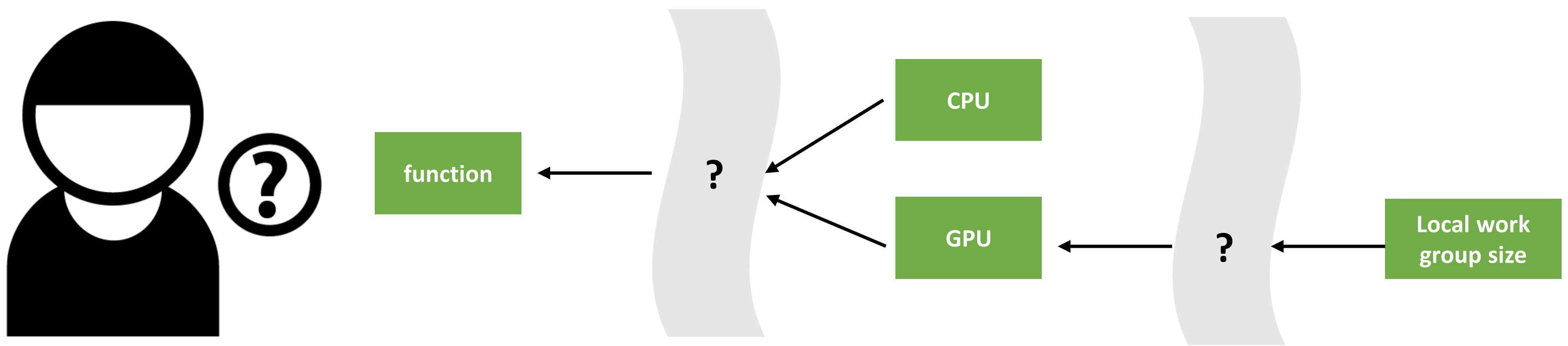
GPGPU Workgroup Size

- Local Workgroup Size: decides how many threads are executed by stream multiprocessor units
- Too small: under-utilised GPU
- Too large: local memory spill, resulting in costly I/O with RAM



Exposing hidden parameter: Deep Parameter Optimisation[2]

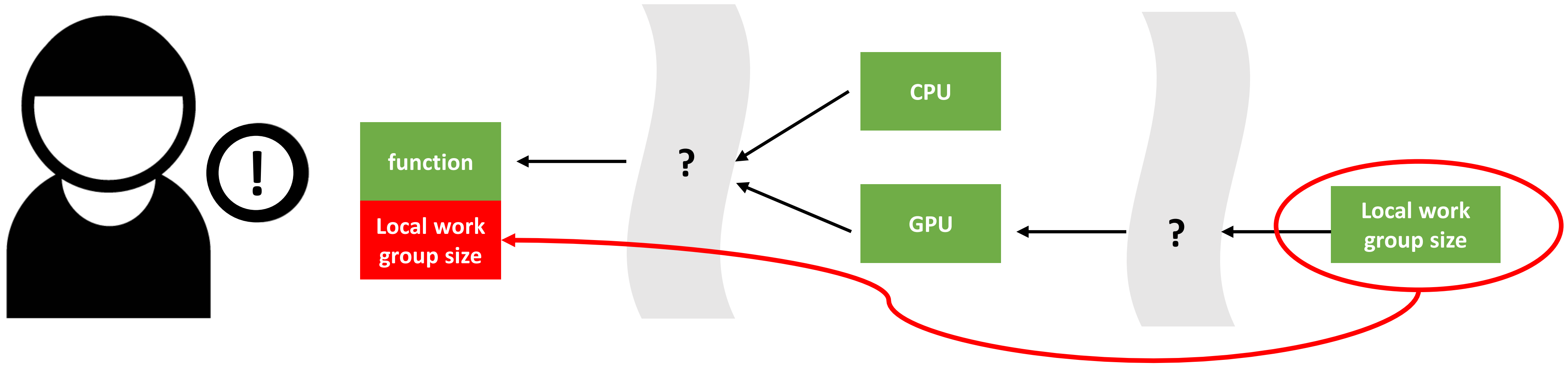
- For cases where parameter that controls the performance is **hidden**
- Expose '**deep**'(previously hidden) parameter to be explicitly controlled
- Our case,
  - Local work group size for GPGPU module of OpenCV controls the performance
  - ➔ Should be exposed to be explicitly controlled for optimisation of the performance



[2]Wu, F., Weimer, W., Harman, M., Jia, Y., Krinke, J.: Deep parameter optimisation. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 1375{1382. GECCO '15, ACM, New York, NY, USA (2015)

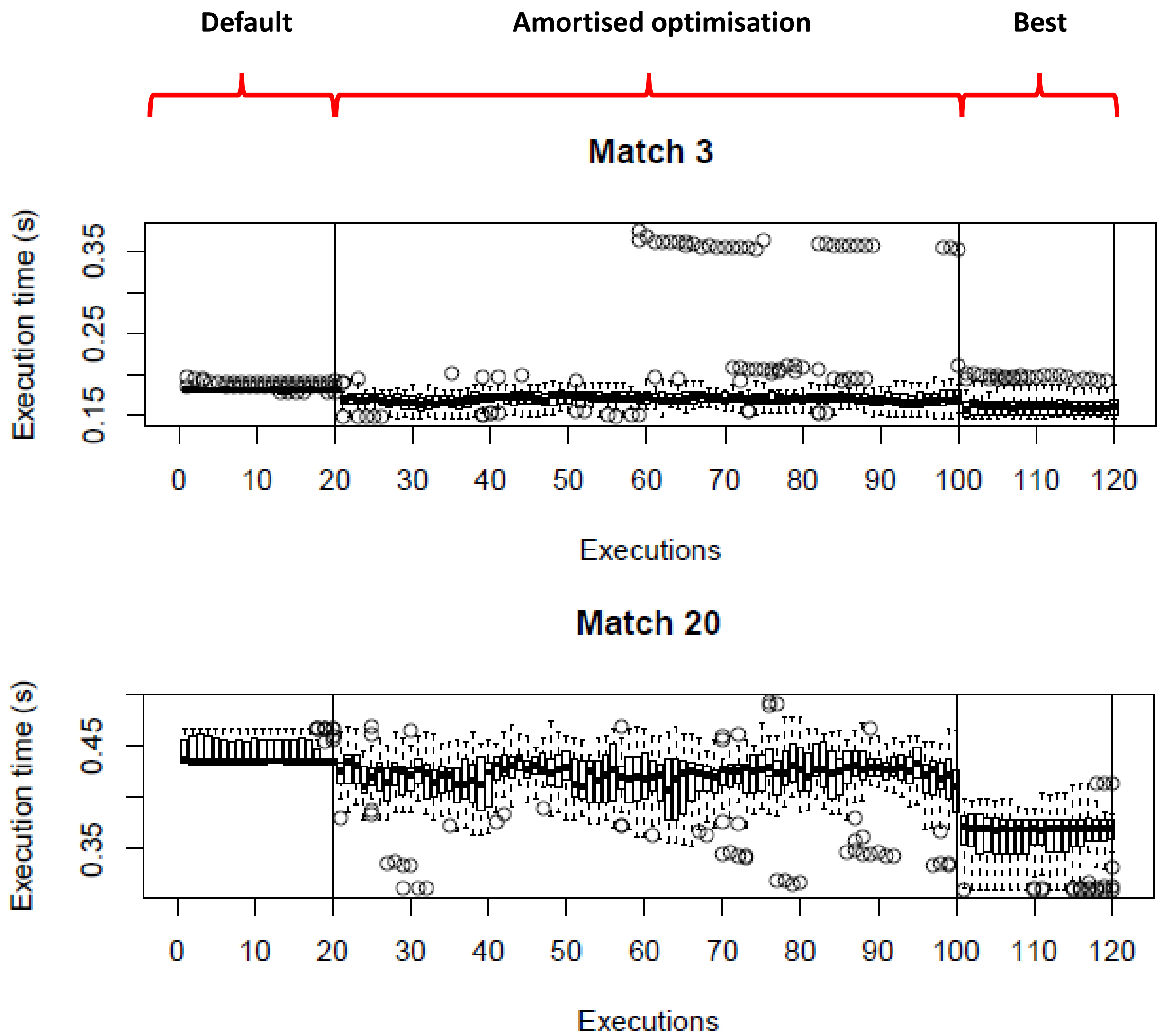
Exposing hidden parameter: Deep Parameter Optimisation[2]

- For cases where parameter that controls the performance is **hidden**
- Expose ‘**deep**’(previously hidden) parameter to be explicitly controlled
- Our case,
  - Local work group size for GPGPU module of OpenCV controls the performance
  - ➔ Should be exposed to be explicitly controlled for optimisation of the performance



[2]Wu, F., Weimer, W., Harman, M., Jia, Y., Krinke, J.: Deep parameter optimisation. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 1375{1382. GECCO '15, ACM, New York, NY, USA (2015)

Results


Default
Amortised optimisation
Best
Match 3
Execution time (s)
0.15 0.25 0.35
0 10 20 30 40 50 60 70 80 90 100 110 120
Executions
Match 20
Execution time (s)
0.35 0.45
0 10 20 30 40 50 60 70 80 90 100 110 120
Executions

Tuning MPM Modules for Apache

- Web servers run in many devices: Raspberry Pi, rack servers, desktop PCs, …
- But they have the same Apache2 parameters!

Methodology – Objective / Fitness

- Server Side (unit: %):
  - Minimize (average CPU usage) + (average memory usage)

- Client Side (unit: sec):
  - Minimize (max response time) / 10 + (average response time)

- Measured 2 times, use average value

Experiments

- Server Environments:
  - Xen Virtual Server (Hosted by SPARCS)
  - Target: a simple MediaWiki site (Apache2.4 + PHP5 + MySQL)
  - 1st Server (CPU: Xeon E5645@2.40GHz 1 core / Memory: 256MB)
  - 2nd Server (CPU: Xeon E5645@2.40GHz 2 cores / Memory: 2GB)

- Client Environments:
  - Sungwon’s Personal Computer: Ubuntu, Same Subnet
  - Microsoft Azure: Ubuntu, Different Subnet

Results

- 1st Server, 1st Scenario (around 60min):

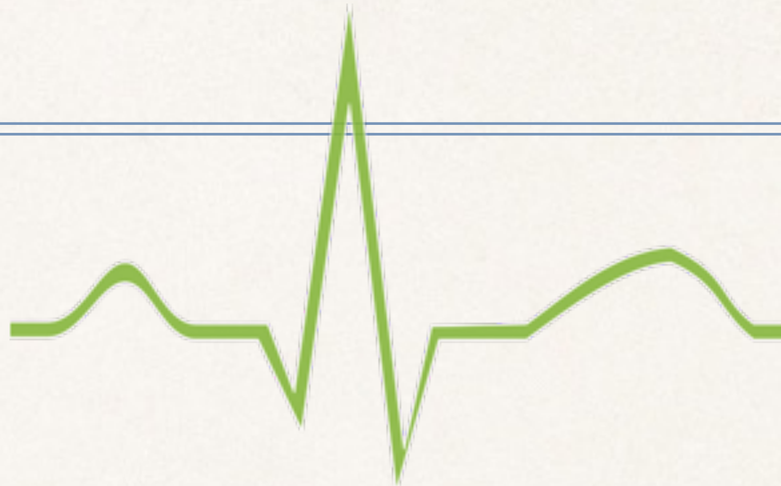
	RAW DATA				FITNESS	
	CPU AVG	MEM AVG	TIME MAX	TIME AVG	SERVER	CLIENT
DEFAULT	86.3445	61.6227	2.8777	0.8150	147.9672	1.1028
OUR SOL	82.2728	50.5601	1.7528	0.7327	132.8329	0.9080

- 1st Server, 2nd Scenario (around 70min):

	RAW DATA				FITNESS	
	CPU AVG	MEM AVG	TIME MAX	TIME AVG	SERVER	CLIENT
DEFAULT	85.3299	66.2125	2.8559	0.8190	151.5424	1.1046
OUR SOL	85.5942	47.3762	1.3903	0.7653	132.9704	0.9043

Threats

Restricted to behaviour-preserving optimisations only



User may experience performance fluctuation

We want you!

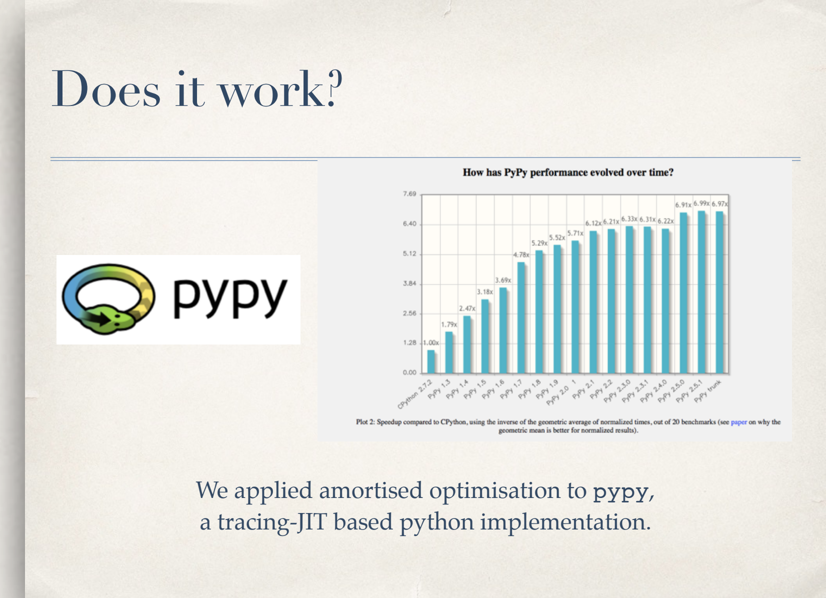
Getting precise measurements

Next Steps

- Population-based optimisation using multiplicity: for example, swarm optimisation of performance-critical parameters in a data centre.
- Shadowing: parallel instance dedicated for optimisation.
- Prepackaged GI: GI as aspects, tagging, directives

References

- S. Yoo. *Amortised optimisation of non-functional properties in production environments*. In Search-Based Software Engineering, volume 9275 of Lecture Notes in Computer Science, pages 31–46. Springer International Publishing, 2015.
- J. Sohn, S. Lee, and S. Yoo. *Amortised deep parameter optimisation of gpgpu work group size for OpenCV*. In Search Based Software Engineering, volume 9962 of Lecture Notes in Computer Science, pages 211–217. Springer International Publishing, 2016.
- Sungwon Cho, and Hyeongjun Cho, *Apache2 Parameter Optimisation*, CS492B Term Project, School of Computing, KAIST, Autumn 2016



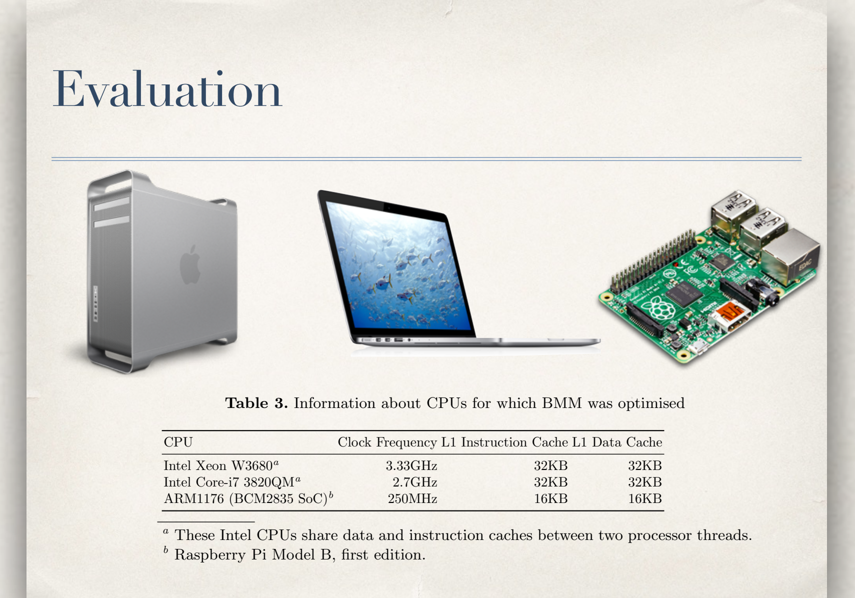
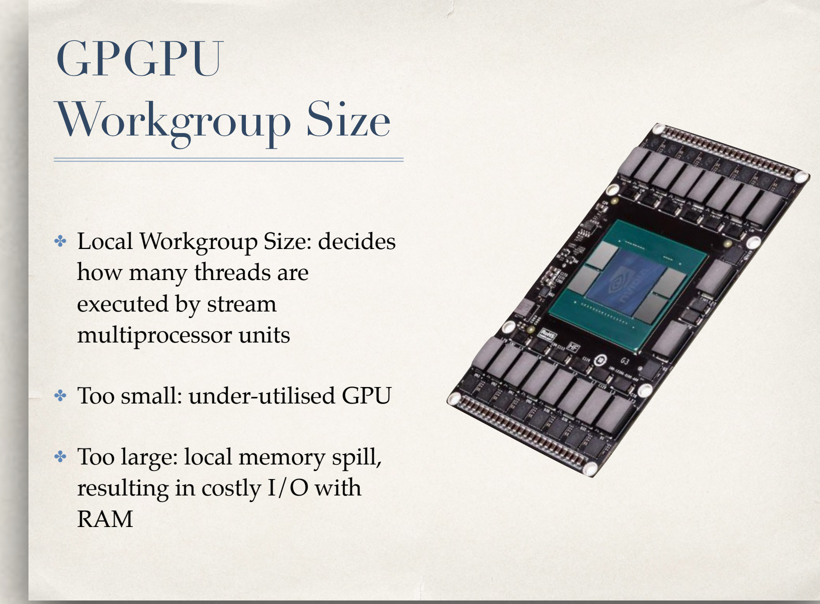
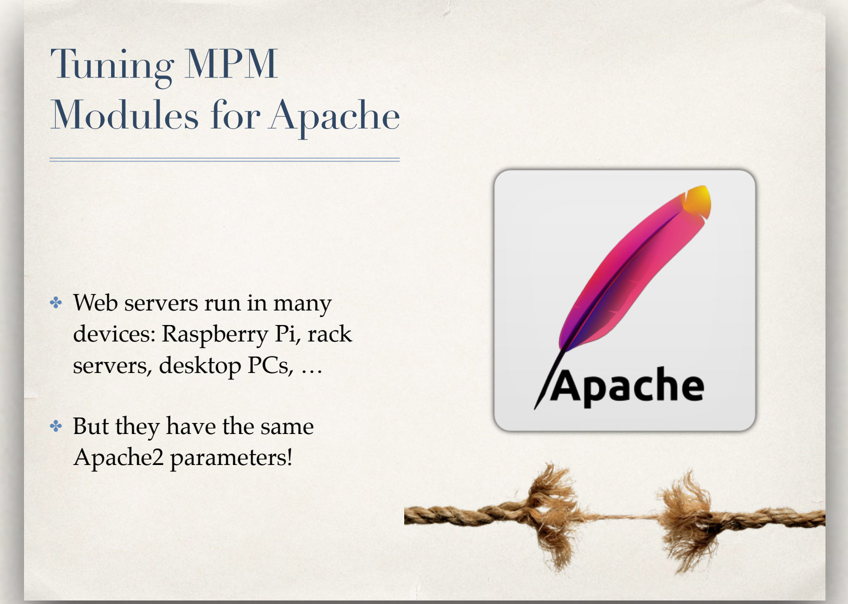


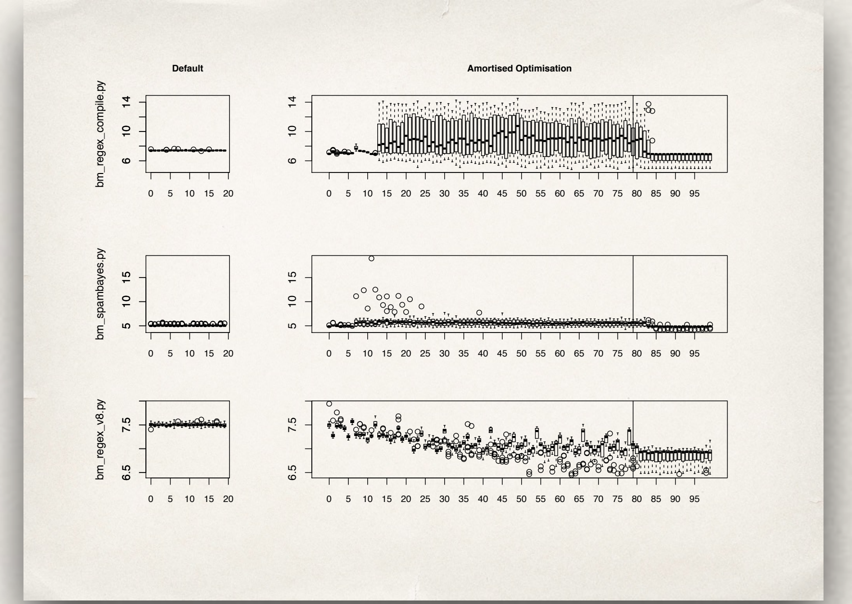
Table 3. Information about CPUs for which BMM was optimised

CPU	Clock Frequency	L1 Instruction Cache	L1 Data Cache
Intel Xeon W3680[a]	3.33GHz	32KB	32KB
Intel Core-i7 3820QM[a]	2.7GHz	32KB	32KB
ARM1176 (BCM2835 SoC)[b]	250MHz	16KB	16KB

[a] These Intel CPUs share data and instruction caches between two processor threads.
[b] Raspberry Pi Model B, first edition.







Code Available

https://bitbucket.org/ntrolls/piacin

https://bitbucket.org/ntrolls/niacin