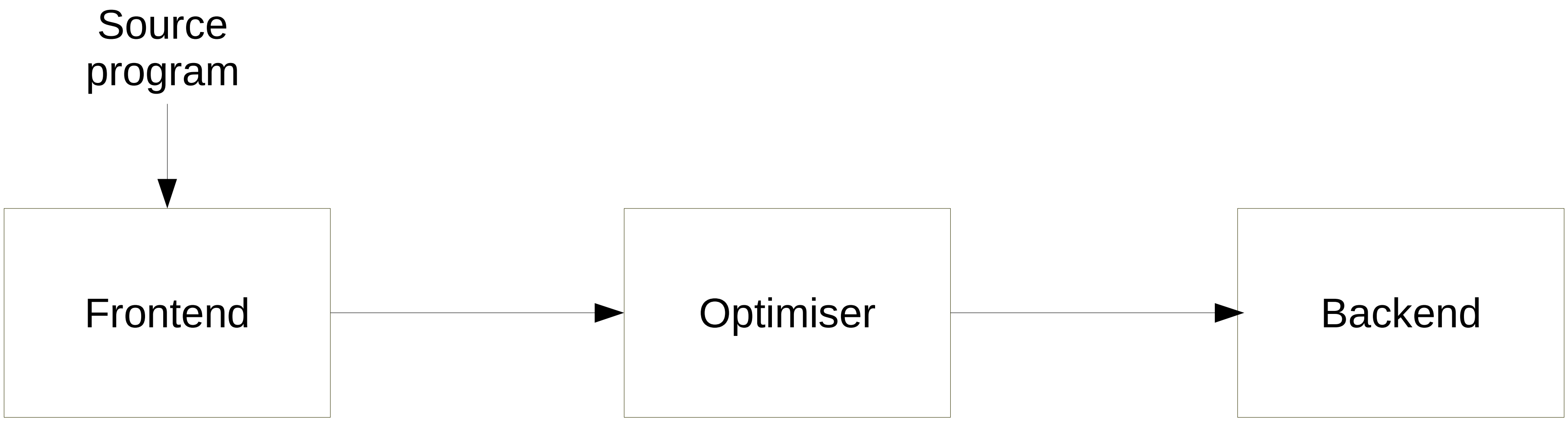
Exploring the fundamental differences between compiler optimisations for energy and for time

James Pallister and Kerstin Eder
University of Bristol

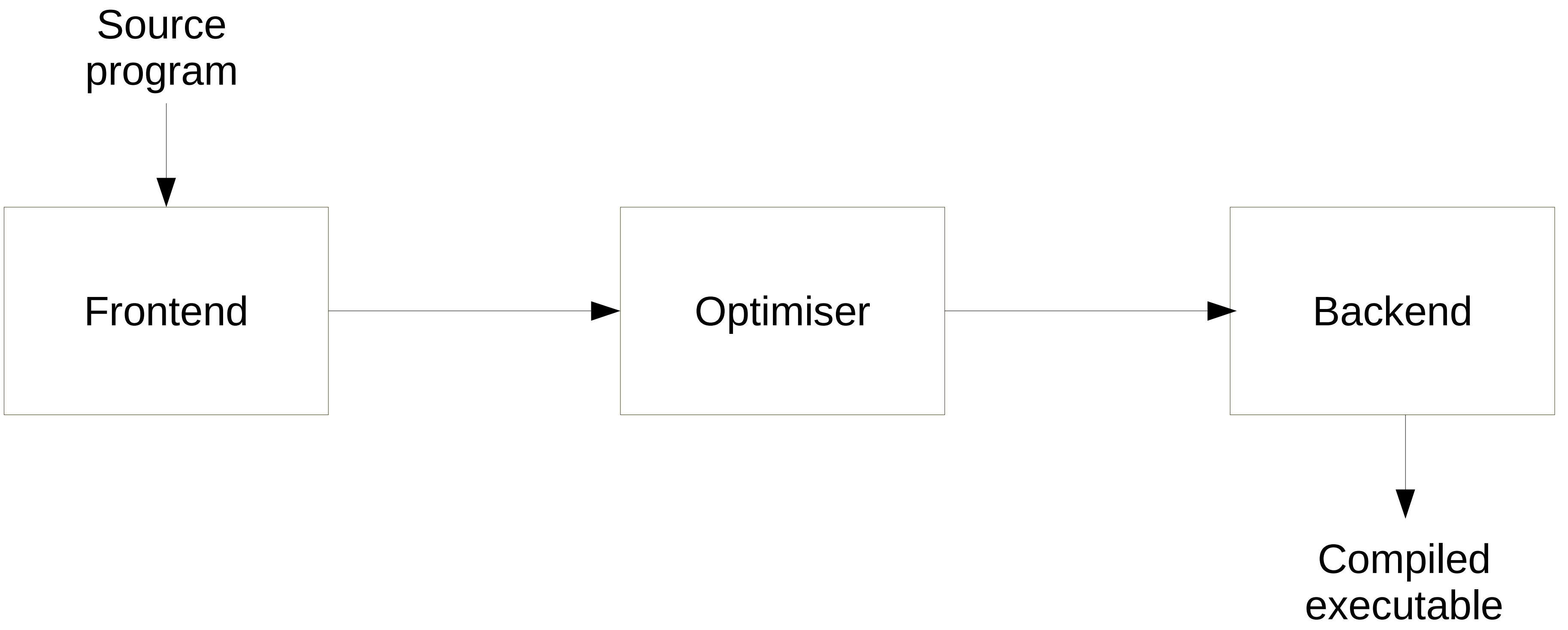
University of BRISTOL

Compilers

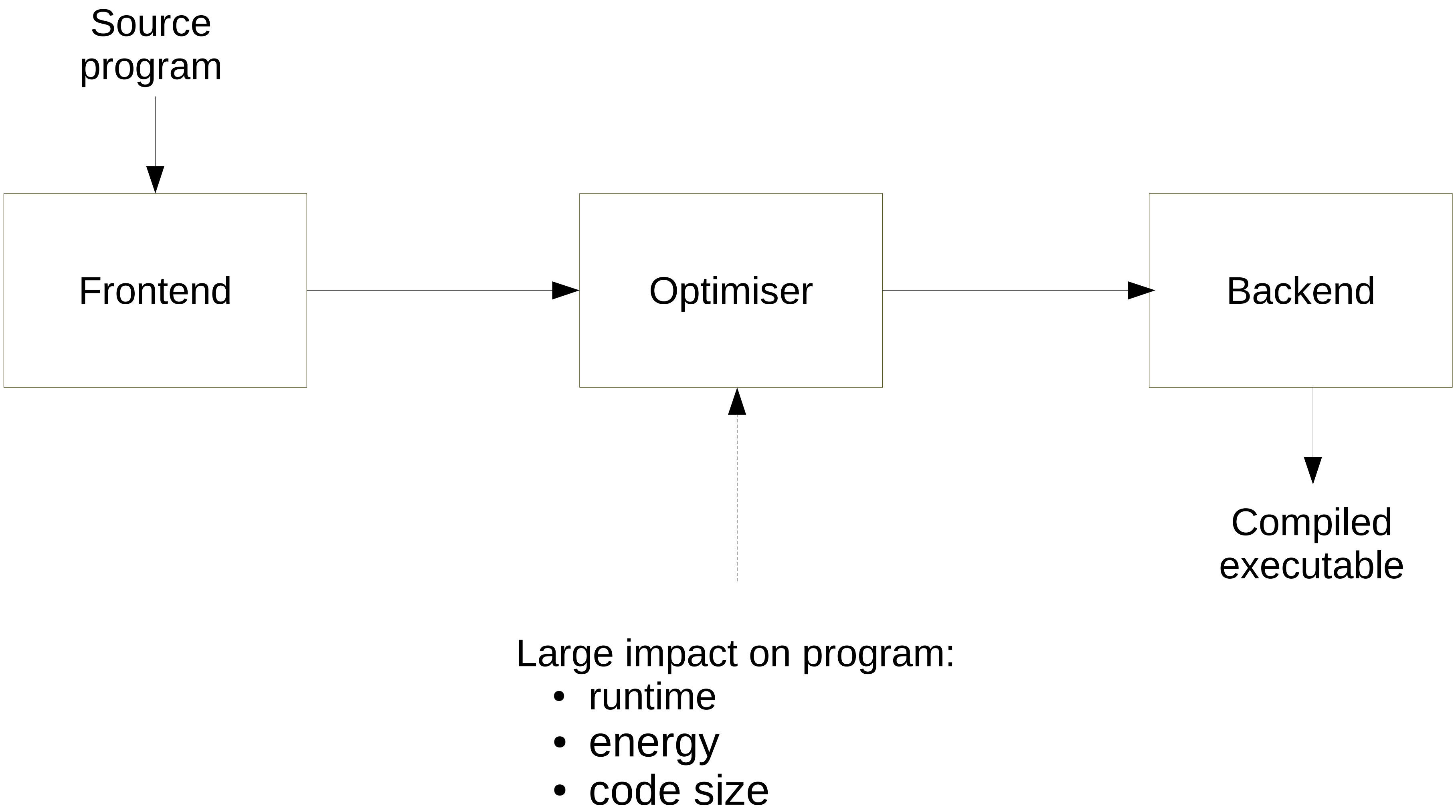
Compilers



Compilers


Source
program
Frontend
Optimiser
Backend
Compiled
executable

Compilers


Source program
Frontend
Optimiser
Backend
Compiled executable
Large impact on program:
• runtime
• energy
• code size

Topics of interest

Topics of interest

How do compiler optimisations affect energy, power and time?

Topics of interest

How do compiler optimisations affect energy, power and time?

Are there different “types” of optimisation?

Topics of interest

How do compiler optimisations affect energy, power and time?

Are there different “types” of optimisation?

How does the composition of optimisations affect these metrics?

Topics of interest

How do compiler optimisations affect energy, power and time?

Are there different “types” of optimisation?

How does the composition of optimisations affect these metrics?

Compilers → automatic

Topics of interest

How do compiler optimisations affect energy, power and time?

Are there different “types” of optimisation?

How does the composition of optimisations affect these metrics?

Compilers → automatic

Embedded systems

Overview

Overview

Introduction

Overview

Introduction

Optimisations for execution time

Overview

Introduction

Optimisations for execution time

Optimisations for energy consumption

Overview

Introduction

Optimisations for execution time

Optimisations for energy consumption

Combining optimisations for energy and time

Overview

Introduction

Optimisations for execution time

Optimisations for energy consumption

Combining optimisations for energy and time

Conclusion

Overview

- ***Introduction***
- *Optimisations for execution time*
- *Optimisations for energy consumption*
- *Combining optimisations for energy and time*
- *Conclusion*

An optimisation's effect

$$E = P \times T$$

An optimisation's effect

$$E = \underbrace{\boxed{P}}_{}\times T$$

An optimisation's effect

$$E = \overbrace{\boxed{P}}^{\text{Average power}} \times \underbrace{\boxed{T}}_{\text{Execution time}}$$

An optimisation's effect

Average power

$$\boxed{E} = \boxed{P} \times \boxed{T}$$

Energy

Execution time

An optimisation's effect

$$E = P \times T$$

An optimisation's effect

$$E = P \times T$$

$$T' = k_T \cdot T$$

An optimisation's effect

$$E = P \times T$$

Optimisation's effect on time

$$T' = \boxed{k_T} \cdot T$$

An optimisation's effect

$$E = P \times T$$

Execution time after applying the optimisation

Optimisation's effect on time

$$\boxed{T'} = \boxed{k_T} \cdot T$$

An optimisation's effect

$$E = P \times T$$

$$T' = k_T \cdot T$$

An optimisation's effect

$$E = P \times T$$

$$T' = k_T \cdot T$$

$$P' = k_P \cdot P$$

An optimisation's effect

$$E = P \times T$$

$$T' = k_T \cdot T$$

$$P' = k_P \cdot P$$

$$E' = (k_P \cdot P) \times (k_T \cdot T)$$

An optimisation's effect

$$T' = k_T \cdot T$$

$$P' = k_P \cdot P$$

$$E' = (k_P \cdot P) \times (k_T \cdot T)$$

An optimisation's effect

Energy is saved if:

$$(k_T \cdot k_P) < 1$$

$$T' = k_T \cdot T$$

$$P' = k_P \cdot P$$

$$E' = (k_P \cdot P) \times (k_T \cdot T)$$

An optimisation's effect

$$T' = k_T \cdot T$$

$$P' = k_P \cdot P$$

$$E' = (k_P \cdot P) \times (k_T \cdot T)$$

An optimisation's effect

Hypothesis:

The existing optimisations in the compiler only affect energy by reducing k_T.

$$T' = k_T \cdot T$$

$$P' = k_P \cdot P$$

$$E' = (k_P \cdot P) \times (k_T \cdot T)$$

An optimisation's effect

An optimisation's effect

Hypothesis:

The existing optimisations in the compiler only affect energy by reducing k_T.

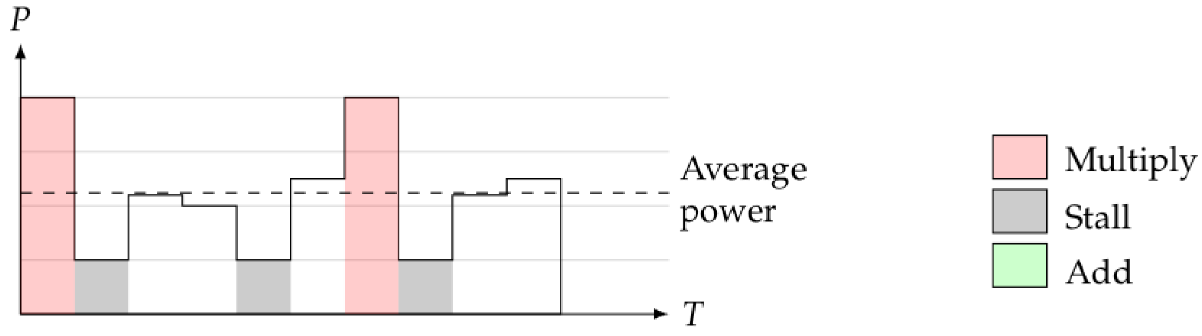
An optimisation's effect

Hypothesis:

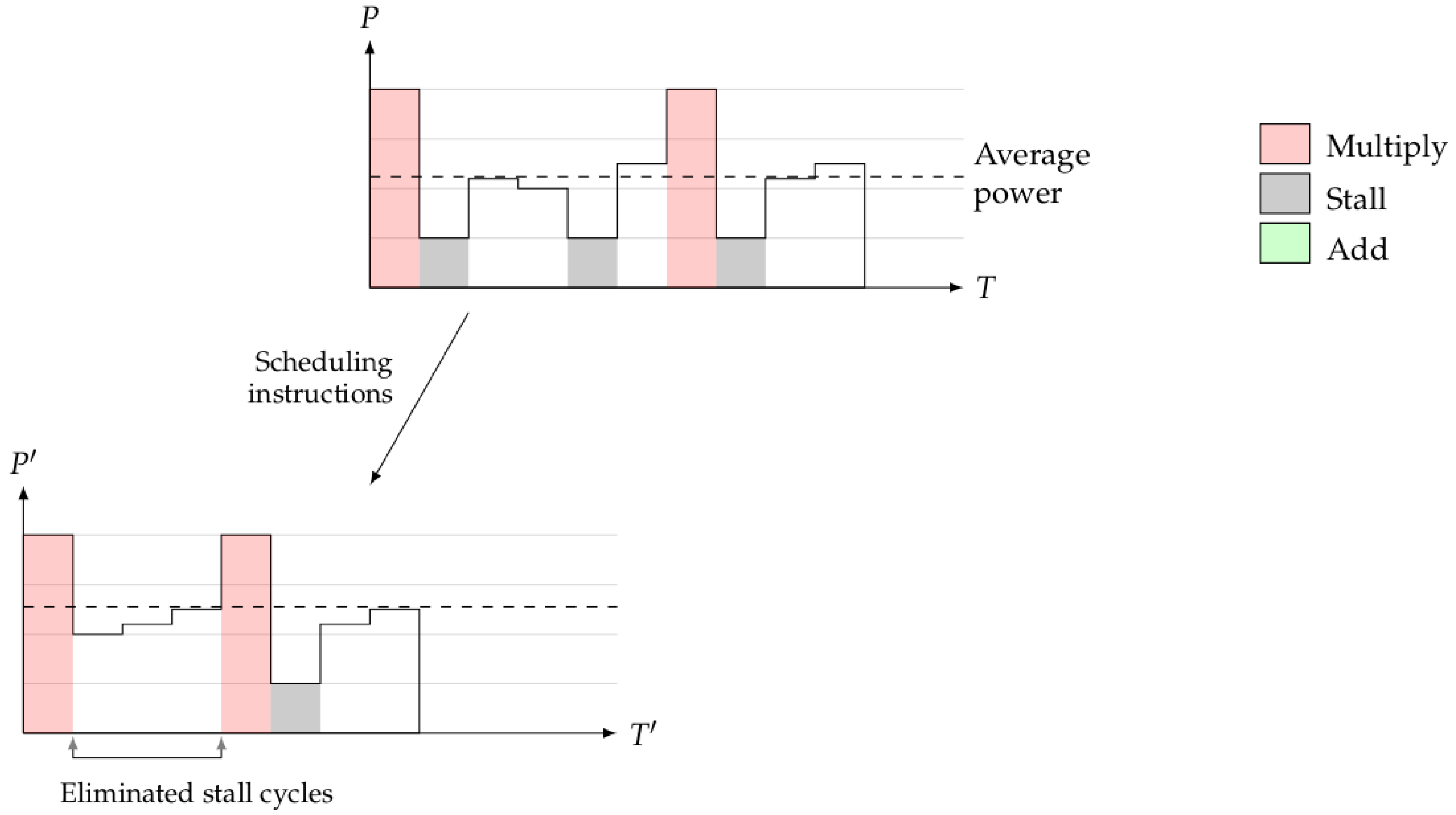
The existing optimisations in the compiler only affect energy by reducing k_T.

Optimisations in the compiler have been designed for reducing execution time.

Optimisation example



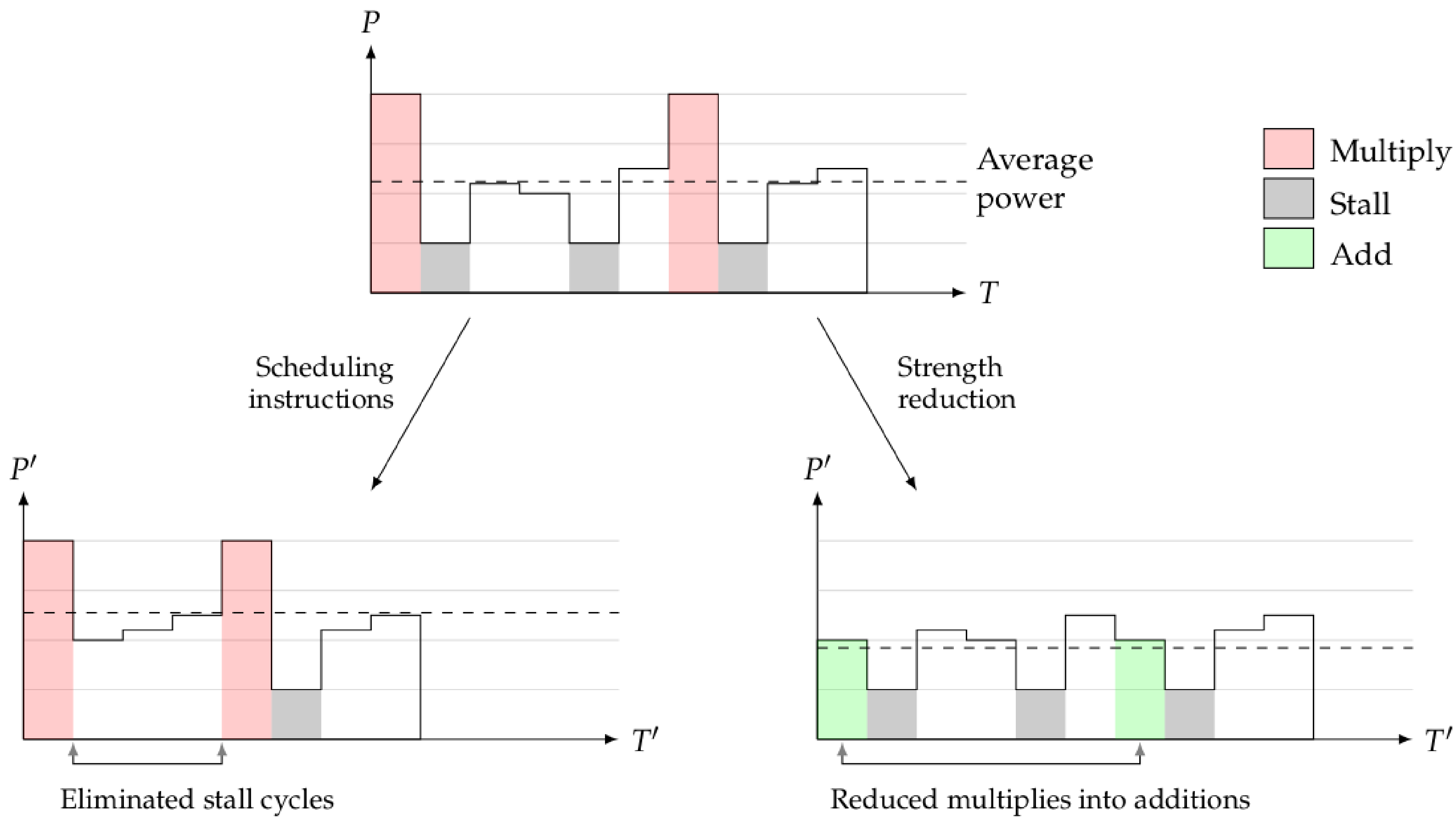
Optimisation example



$$k_T = 0.8$$
$$k_P = 1.14$$

$$k_E = k_T \cdot k_P = 0.91$$

Optimisation example



$k_T = 0.8$
$k_P = 1.14$

$k_E = k_T \cdot k_P = 0.91$

$k_T = 1.0$
$k_P = 0.82$

$k_E = k_T \cdot k_P = 0.82$

Interactions and ordering

Interactions and ordering

No transformations

```
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    return function1(b + 1)
           * function1(b + 1);
}
```

Interactions and ordering

No transformations

```
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    return function1(b + 1)
             * function1(b + 1);
}
```

CSE

```
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    int t1 = function1(b + 1);
    return t1 * t1;
}
```

Interactions and ordering

No transformations

```
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    return function1(b + 1)
             * function1(b + 1);
}
```

CSE

```
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    int t1 = function1(b + 1);
    return t1 * t1;
}
```

Inlining

```
int function2(int b)
{
    return ((b + 1) * (b + 1))
             * ((b + 1) * (b + 1));
}
```

Interactions and ordering

No transformations

```
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    return function1(b + 1)
            * function1(b + 1);
}
```

CSE

```
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    int t1 = function1(b + 1);
    return t1 * t1;
}
```

Inlining

```
int function2(int b)
{
    return ((b + 1) * (b + 1))
            * ((b + 1) * (b + 1));
}
```

CSE, then inlining

```
int function2(int b)
{
    int t1 = (b + 1) * (b + 1);
    return t1 * t1;
}
```

Interactions and ordering

No transformations

```
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    return function1(b + 1)
              * function1(b + 1);
}
```

CSE

```
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    int t1 = function1(b + 1);
    return t1 * t1;
}
```

Inlining

```
int function2(int b)
{
    return ((b + 1) * (b + 1))
              * ((b + 1) * (b + 1));
}
```

CSE, then inlining

```
int function2(int b)
{
    int t1 = (b + 1) * (b + 1);
    return t1 * t1;
}
```

Inlining, then CSE

```
int function2(int b)
{
    int t1 = b + 1;
    return (t1 * t1) * (t1 * t1);
}
```

Research questions

Research questions

Standard compiler optimisations

- Do existing compiler optimisations save energy purely by reducing the k_T coefficient?
- Are there any optimisations which affect k_P?

Research questions

Standard compiler optimisations

- Do existing compiler optimisations save energy purely by reducing the k_T coefficient?
- Are there any optimisations which affect k_P ?

Compiler optimisations for energy

- Is there a class of optimisations which can lower the energy via the k_P coefficient?
- How are they different?
- Are they effective?

Research questions

Combining optimisations

- Do optimisations designed to lower k_P interact with existing compiler optimisations?
- Are different sets of optimisations effective when including energy optimisations?

Overview

- *Introduction*
- ***Optimisations for execution time***
- *Optimisations for energy consumption*
- *Combining optimisations for energy and time*
- *Conclusion*

Optimisations for time

Optimisations for time

Optimisations already in the compiler

Optimisations for time

Optimisations already in the compiler

Typically reducing code

- Common subexpression elimination
- Dead code elimination

Optimisations for time

Optimisations already in the compiler

Typically reducing code

- Common subexpression elimination
- Dead code elimination

Typically reordering code

- Instruction scheduling
- Array access patterns
- Moving invariant code out of loops

Optimisations for time

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

Optimisations for time

Test how the compiler optimisations affect energy, by measuring both time and energy

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

Optimisations for time

Test how the compiler optimisations affect energy, by measuring both time and energy

- Individual optimisations

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

Optimisations for time

Test how the compiler optimisations affect energy, by measuring both time and energy

- Individual optimisations
- Sets of optimisations

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

Optimisations for time

Test how the compiler optimisations affect energy, by measuring both time and energy

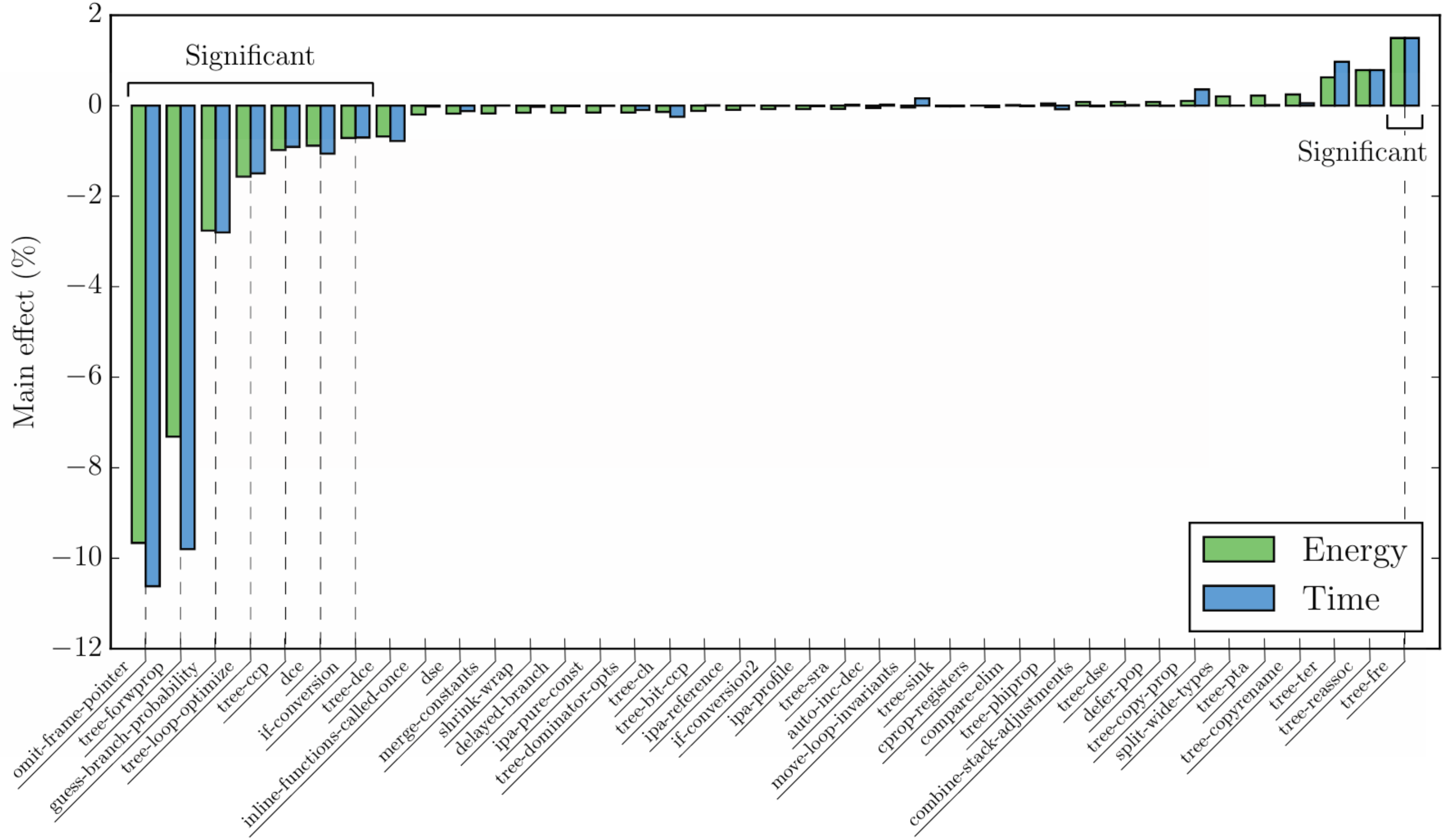
- Individual optimisations
- Sets of optimisations
- The “best” set of optimisations

“Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?”*

Individual optimisations

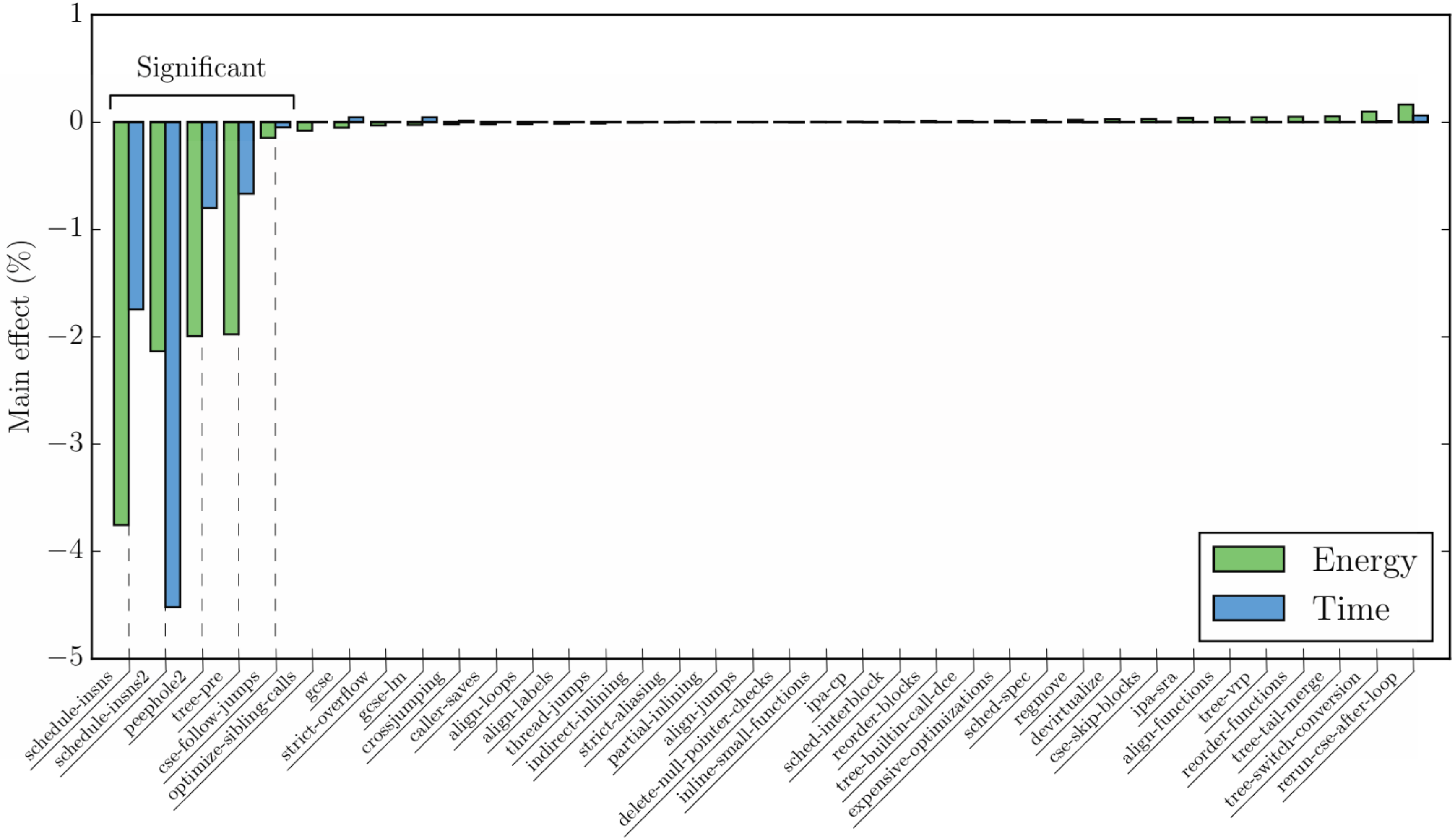
"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

Individual optimisations



"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

Individual optimisations



"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

Sets of optimisations

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

Sets of optimisations

No single set of optimisations is good for all benchmarks or all SoCs.

Sets of optimisations

No single set of optimisations is good for all benchmarks or all SoCs.

The set of effective optimisations for each benchmark is different

Sets of optimisations

No single set of optimisations is good for all benchmarks or all SoCs.

The set of effective optimisations for each benchmark is different

- But the same for time and energy

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

The best set

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

"Are there any optimisations which affect k_P *?"*

The best set

Attempt to pick good combinations of optimisations which maximise a fitness function

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

"Are there any optimisations which affect k_P *?"*

The best set

Attempt to pick good combinations of optimisations which maximise a fitness function

- Genetic algorithms

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

"Are there any optimisations which affect k_P *?"*

The best set

Attempt to pick good combinations of optimisations which maximise a fitness function

- Genetic algorithms

Compare the results of finding the best set for energy and for time.

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

"Are there any optimisations which affect k_P *?"*

The best set

Attempt to pick good combinations of optimisations which maximise a fitness function

- Genetic algorithms

Compare the results of finding the best set for energy and for time.

- If similar, most of the energy savings are coming from a reduction in time.

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

"Are there any optimisations which affect k_P *?"*

The best set

Minimise energy. $F(e,t) = \frac{1}{e}$.

Minimise time. $F(e,t) = \frac{1}{t}$.

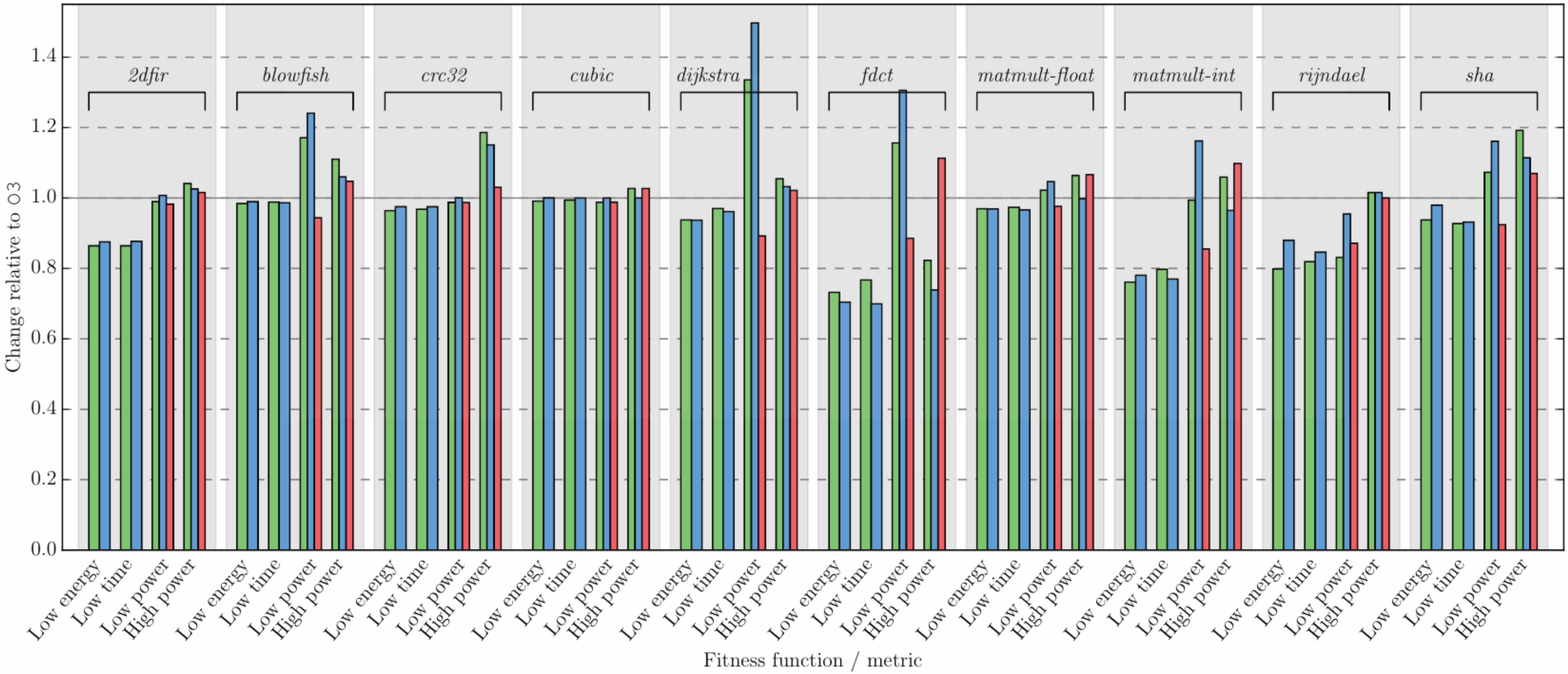
Minimise power. $F(e,t) = \frac{e}{t}$.

Maximise power. $F(e,t) = \frac{t}{e}$.

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

"Are there any optimisations which affect k_P *?"*

The best set



"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

"Are there any optimisations which affect k_P *?"*

Optimisations for time

Optimisations for time

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

Optimisations for time

“Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?”*

- Yes, but some cases where power gets lowered also

Optimisations for time

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

- Yes, but some cases where power gets lowered also
- Not significantly

Optimisations for time

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

- Yes, but some cases where power gets lowered also
- Not significantly

"Are there any optimisations which affect k_P *?"*

Optimisations for time

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

- Yes, but some cases where power gets lowered also
- Not significantly

"Are there any optimisations which affect k_P *?"*

- Not significantly

Overview

- *Introduction*
- *Optimisations for execution time*
- ► ***Optimisations for energy consumption***
- *Combining optimisations for energy and time*
- *Conclusion*

Optimisations for energy

Optimisations for energy

Try to find some optimisations which change the power coefficient

Optimisations for energy

Try to find some optimisations which change the power coefficient

Exploit some characteristics of the target processors:

- Embedded flash structure
- Flash and RAM differences

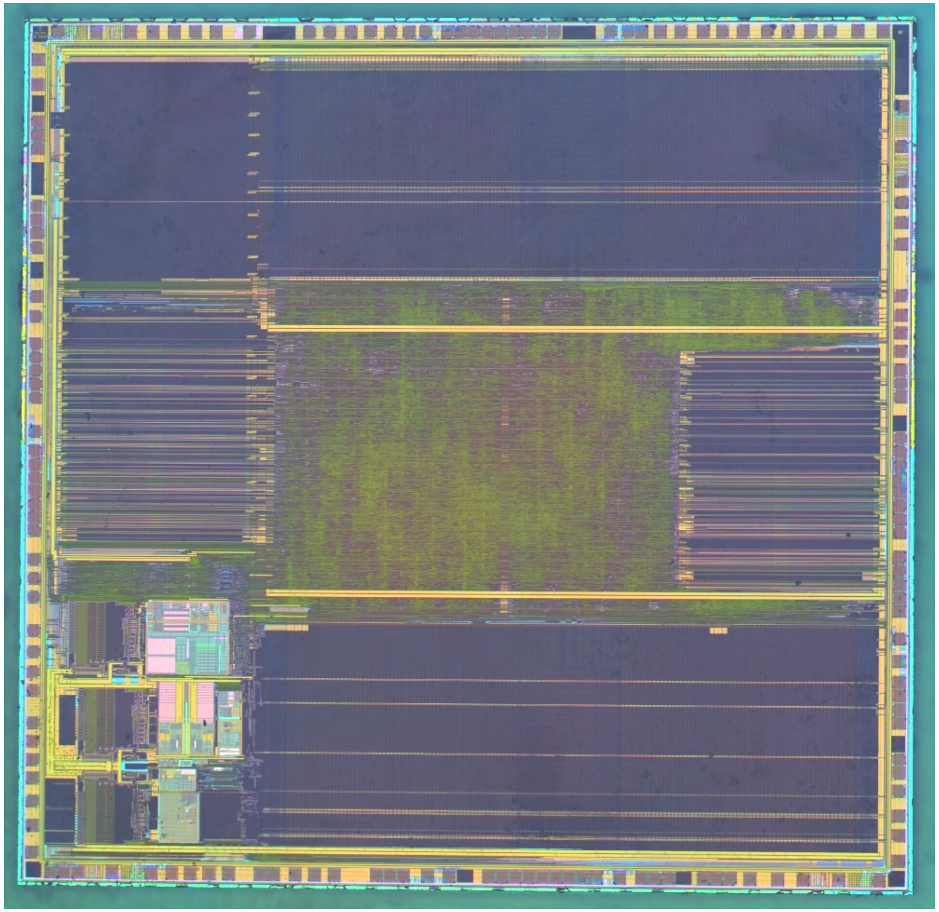
Embedded flash energy

Embedded flash energy

Flash on the same die as the rest of the SoC

Embedded flash energy

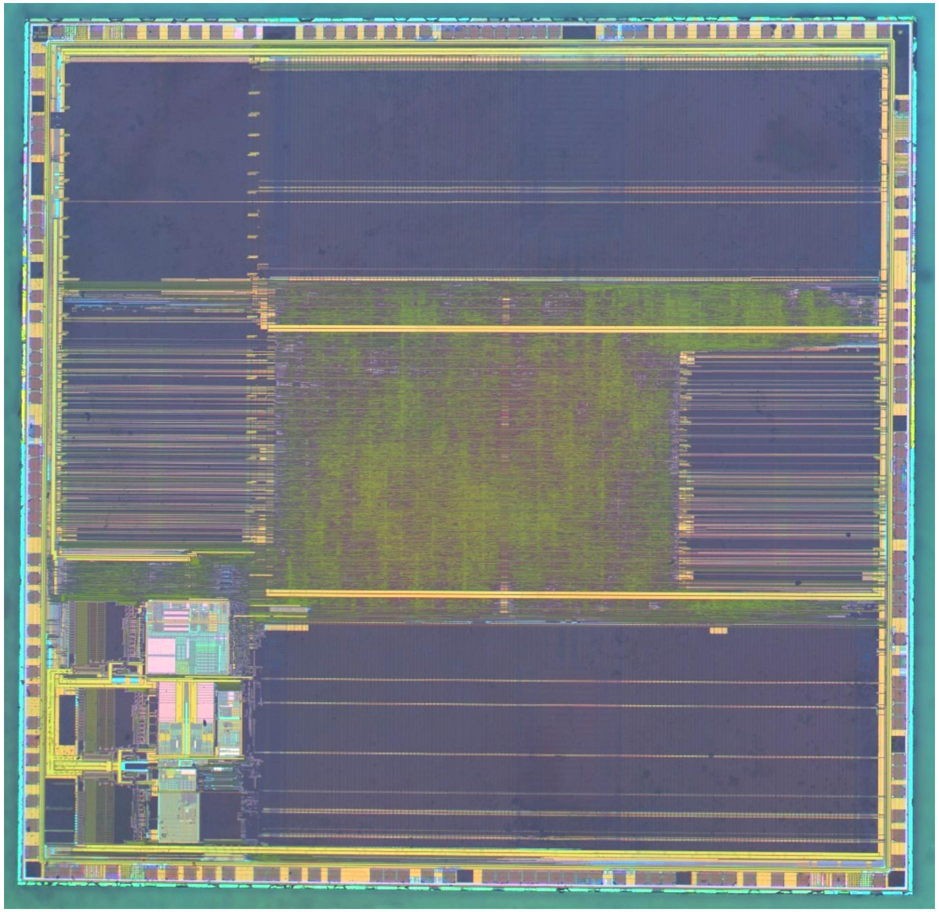
Flash on the same die as the rest of the SoC



Embedded flash energy

Flash on the same die as the rest of the SoC

Code execution directly from flash

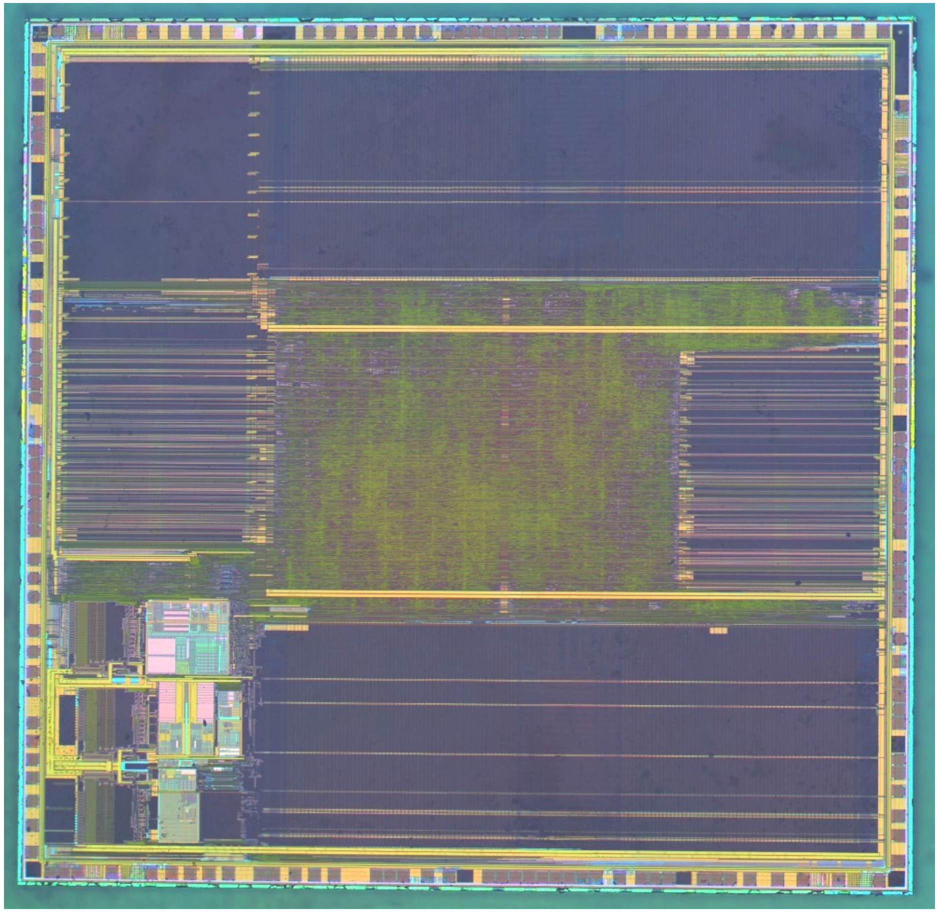


Embedded flash energy

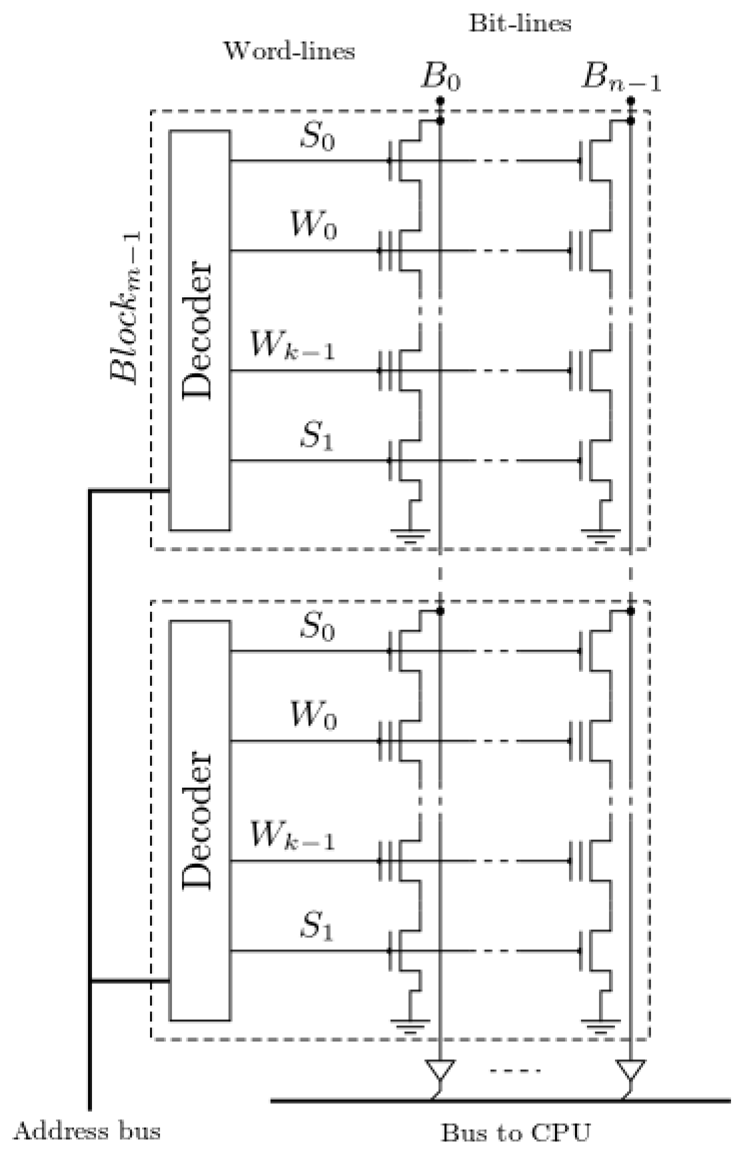
Flash on the same die as the rest of the SoC

Code execution directly from flash

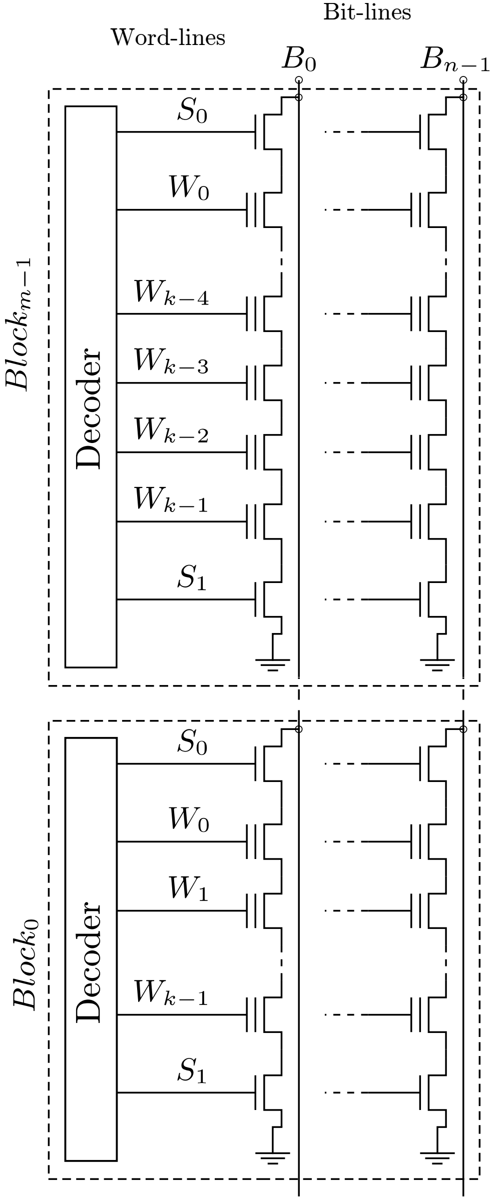
Structure of embedded flash is non-uniform



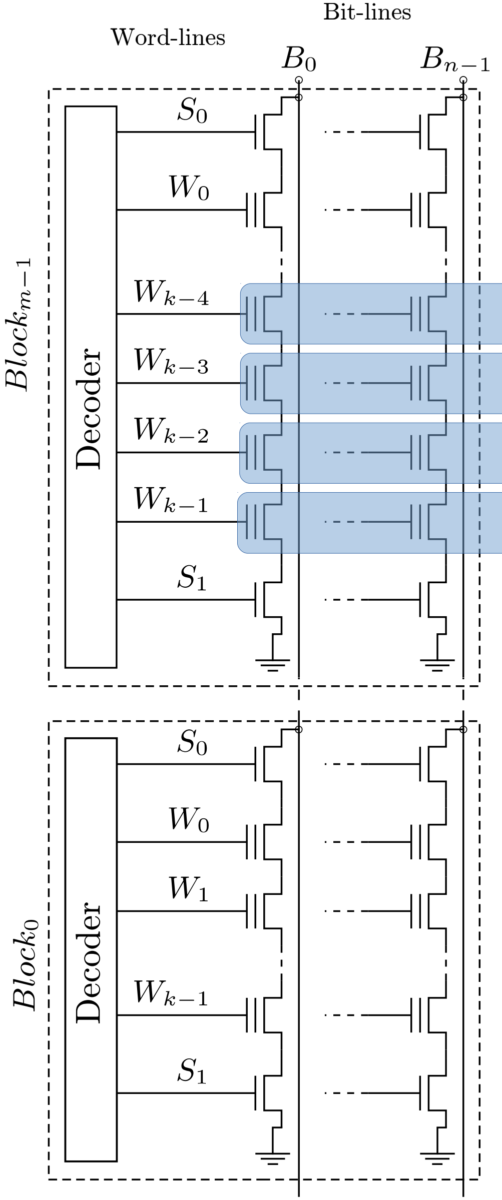
Embedded flash memory



Expected energy effect



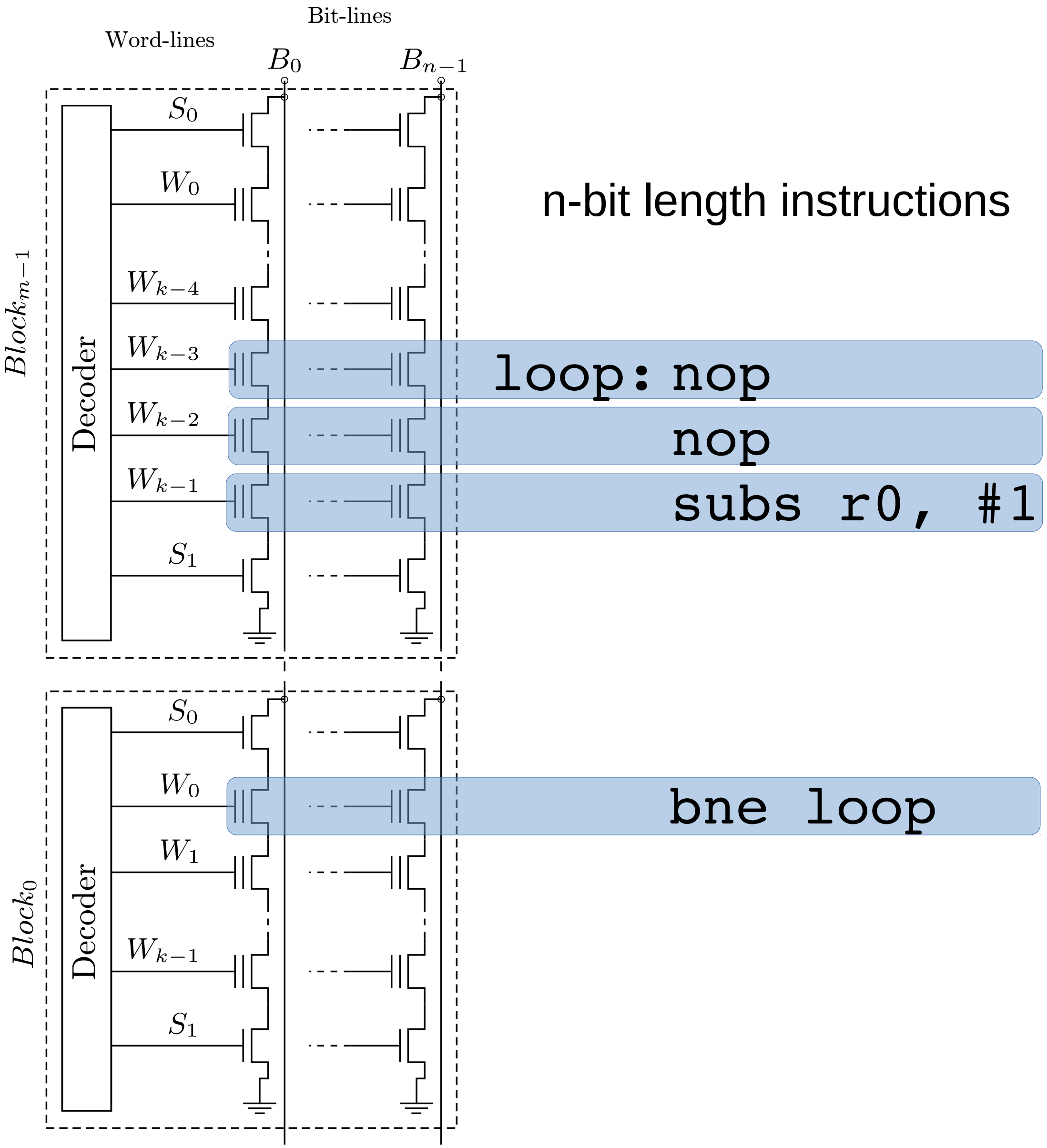
Expected energy effect



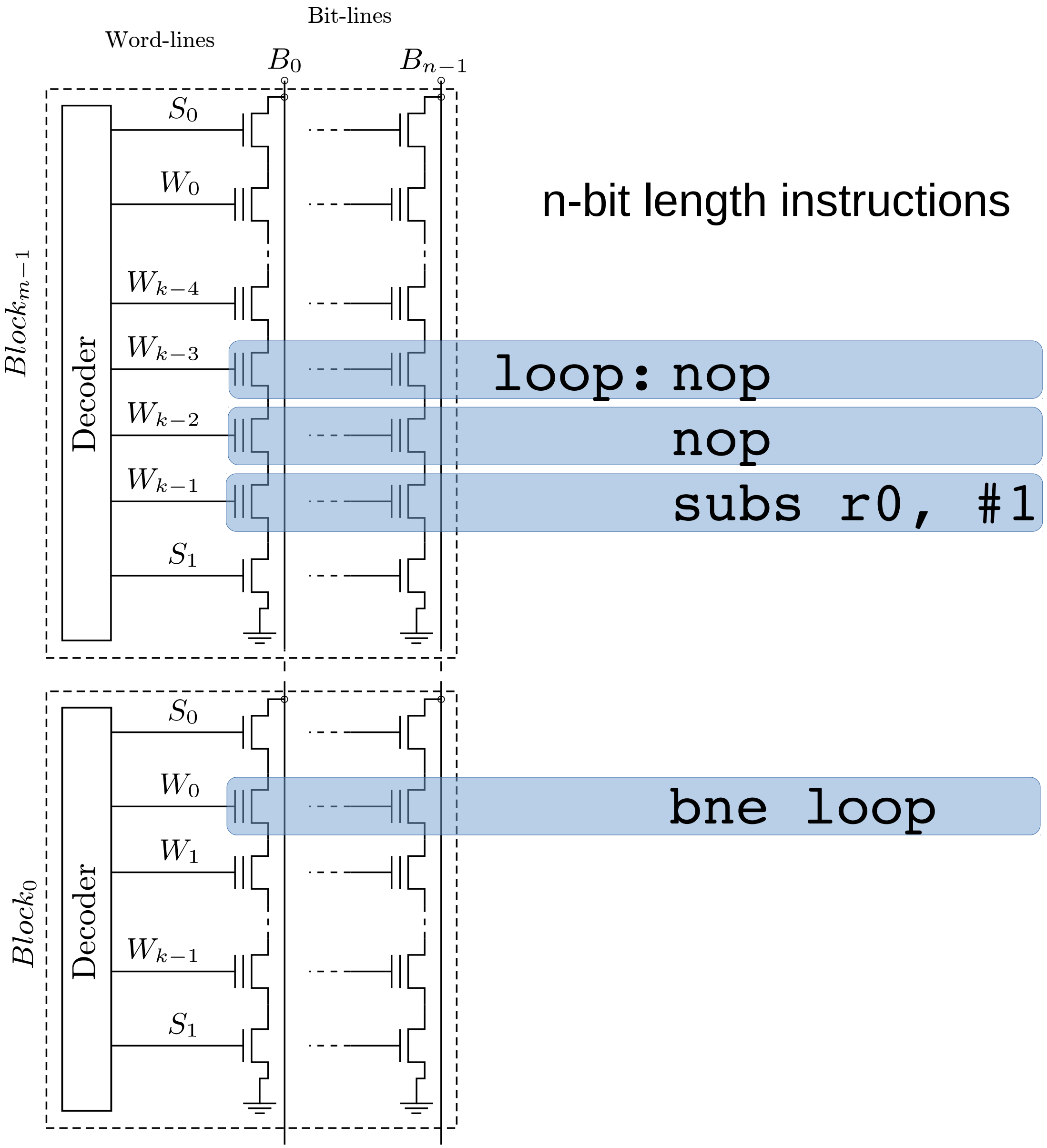
n-bit length instructions

```
loop: nop
      nop
      subs r0, #1
      bne loop
```

Expected energy effect



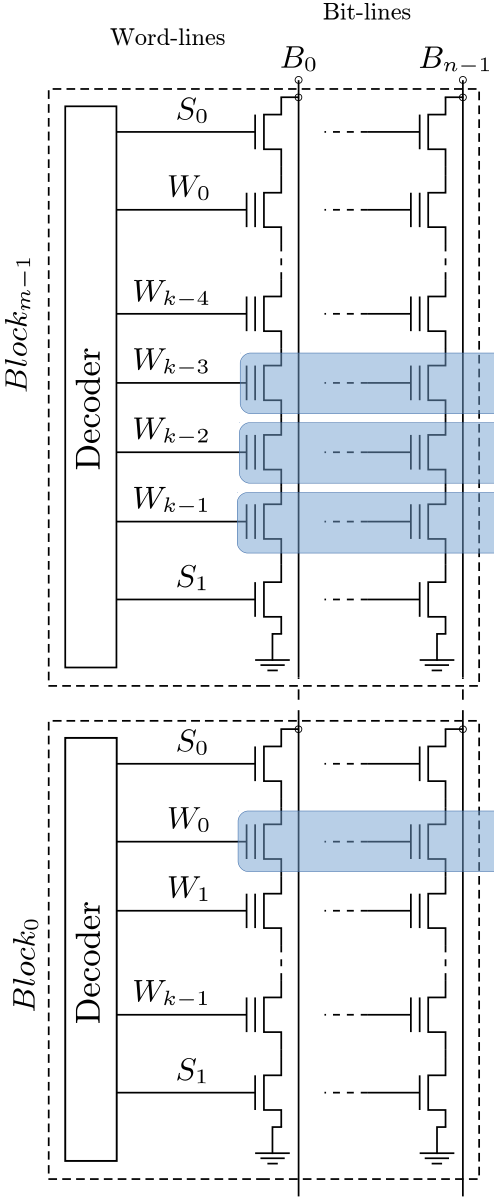
Expected energy effect



n-bit length instructions

Straddling two blocks.

Expected energy effect



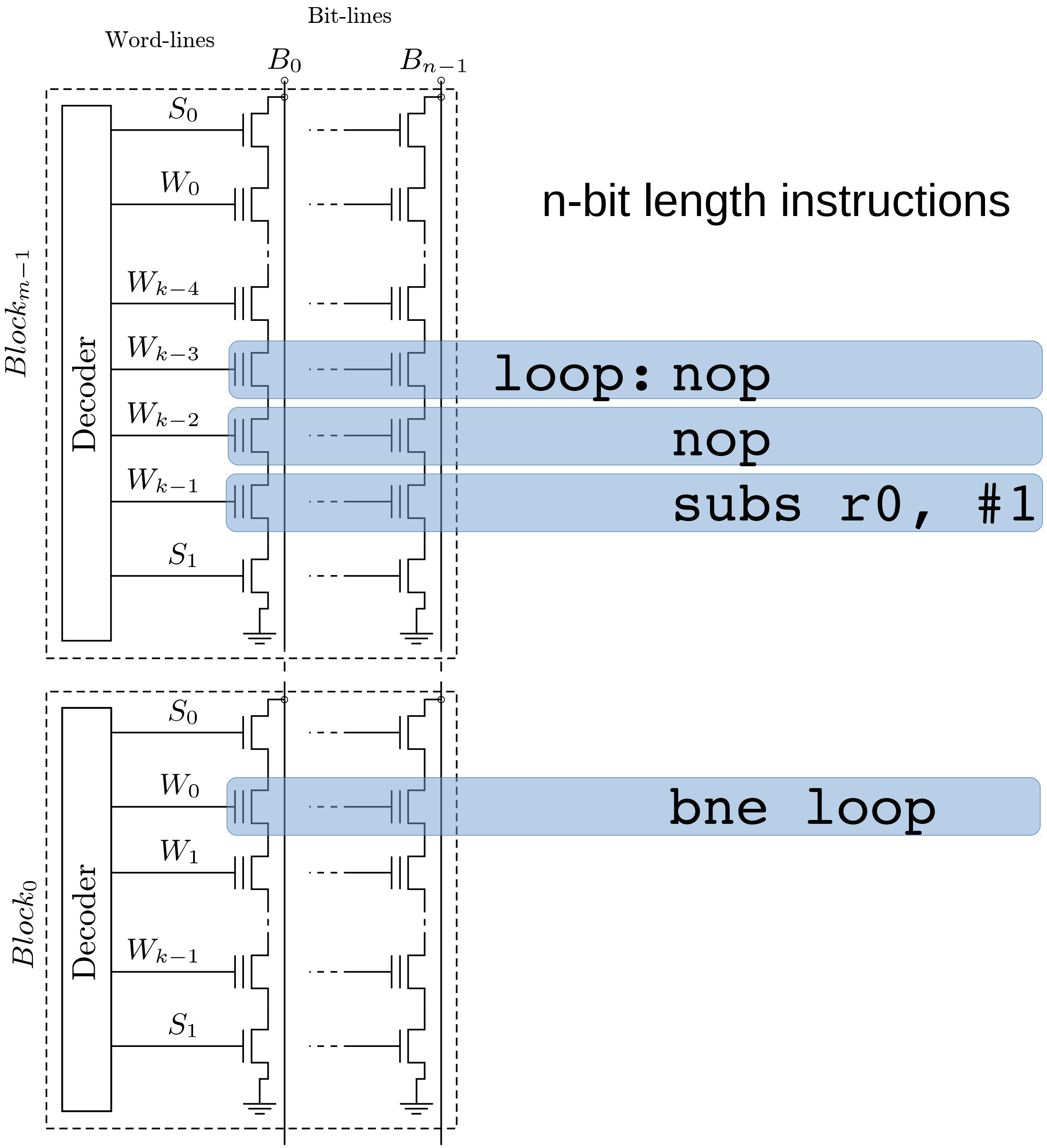
n-bit length instructions

```
loop: nop
      nop
      subs r0, #1
      bne loop
```

Straddling two blocks.

Each iteration must repeatedly power up one, then the other

Expected energy effect

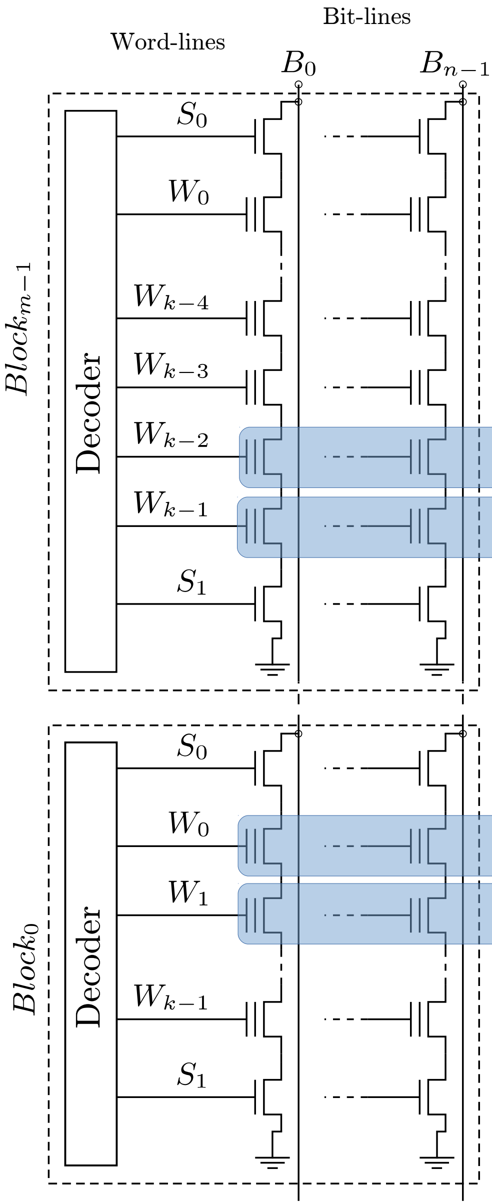


Straddling two blocks.

Each iteration must repeatedly power up one, then the other

Higher energy cost when an instruction jumps from one 'region' to another.

Expected energy effect



n-bit length instructions

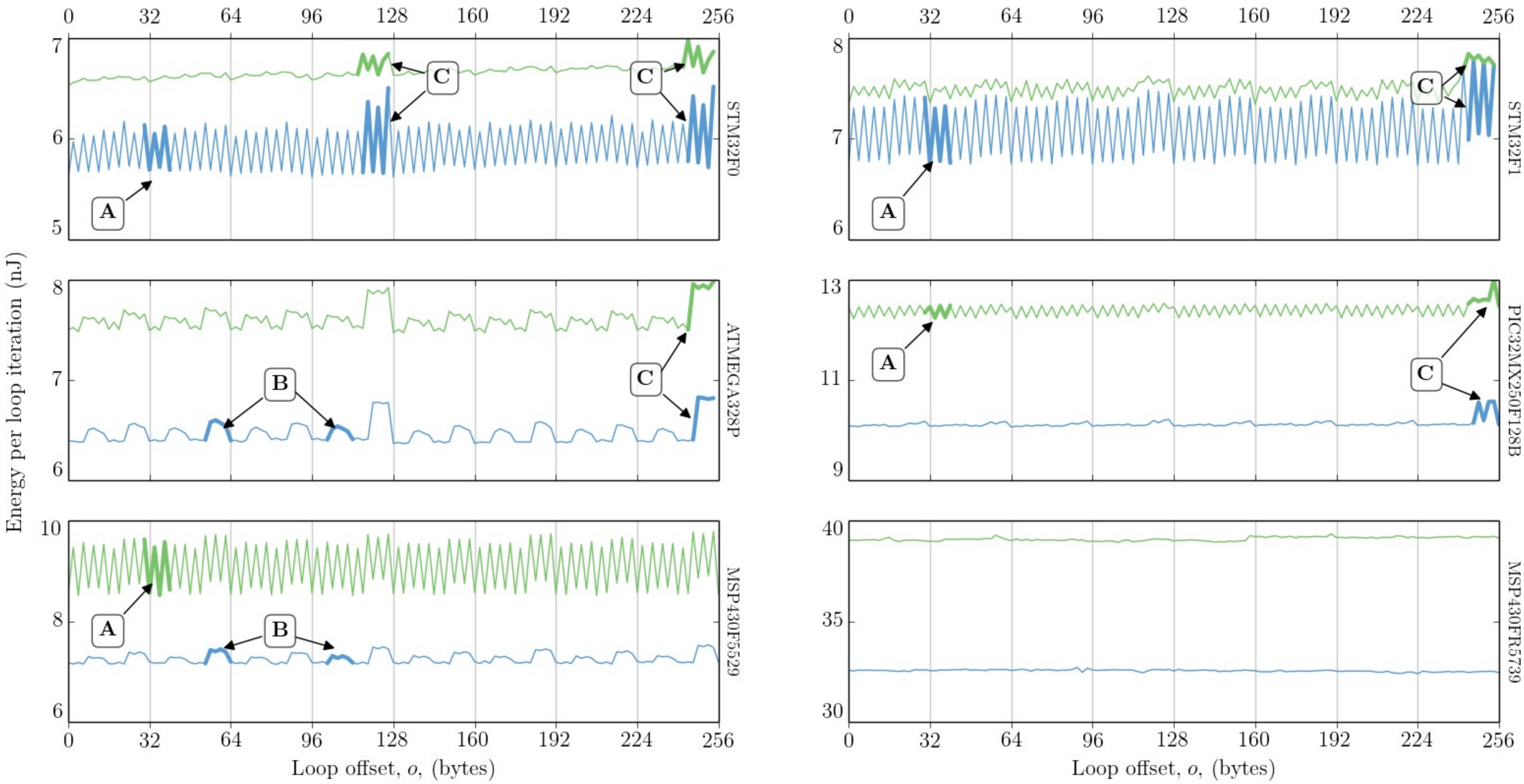
```
loop: nop
      nop
      subs r0, #1
      bne loop
```

Straddling two blocks.

Each iteration must repeatedly power up one, then the other

Higher energy cost when an instruction jumps from one 'region' to another.

Actual results



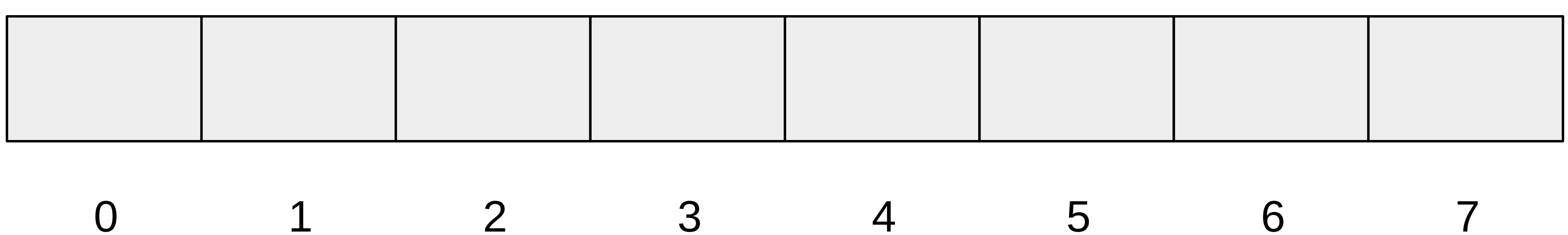
Bottom line (blue) → 8-byte loop

Top line (green) → 10-byte loop

Modelling

Each consecutive memory access has an address dependent energy consumption.

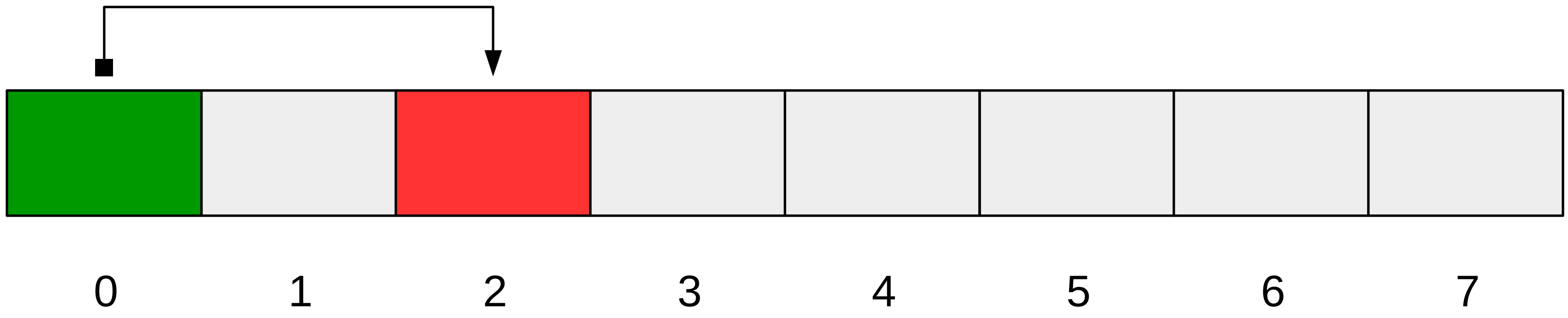
E.g. $0 \rightarrow 2$



Modelling

Each consecutive memory access has an address dependent energy consumption.

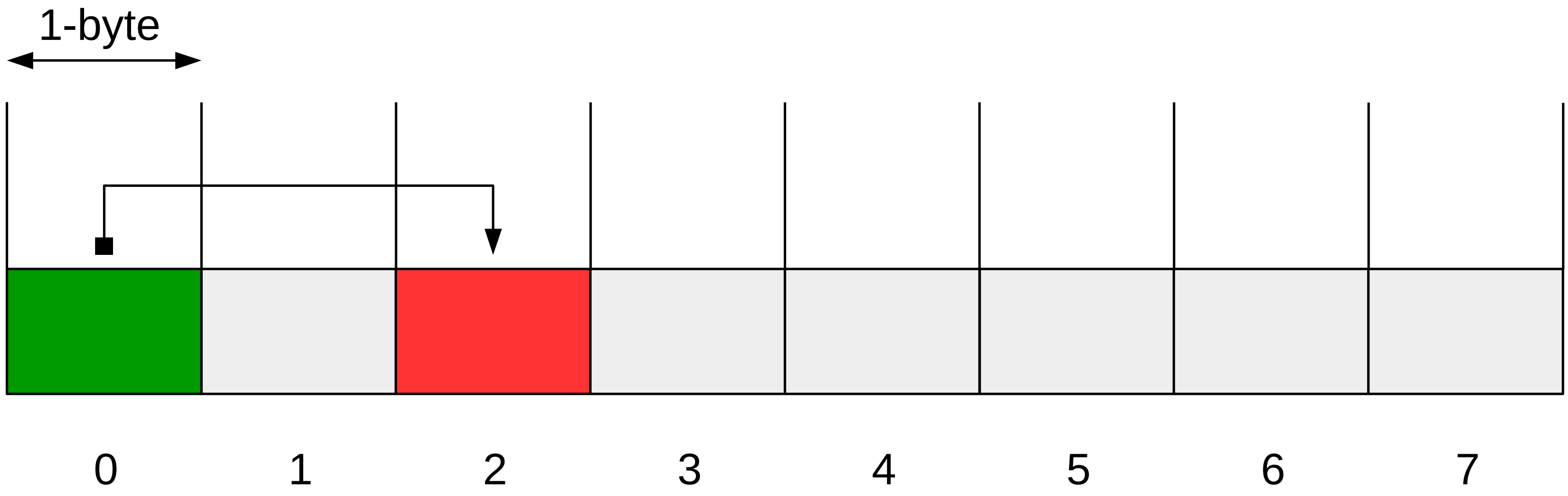
E.g. $0 \rightarrow 2$



Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $0 \rightarrow 2$

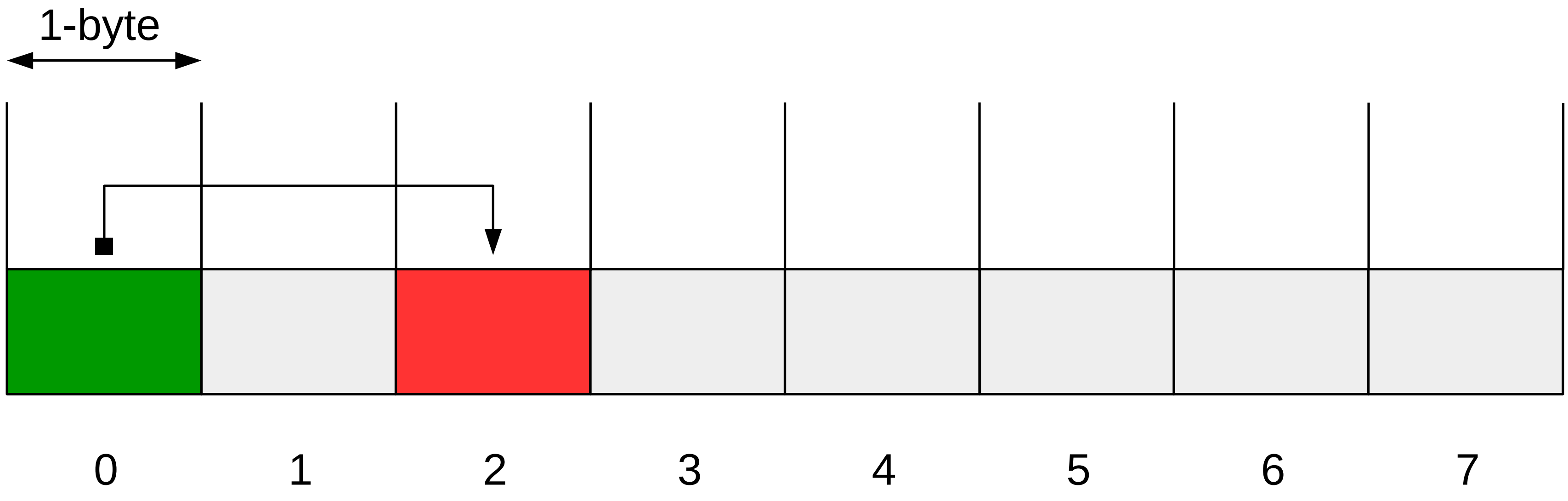


Modelling

Each consecutive memory access has an address dependent energy consumption.

E_0

E.g. $0 \rightarrow 2$

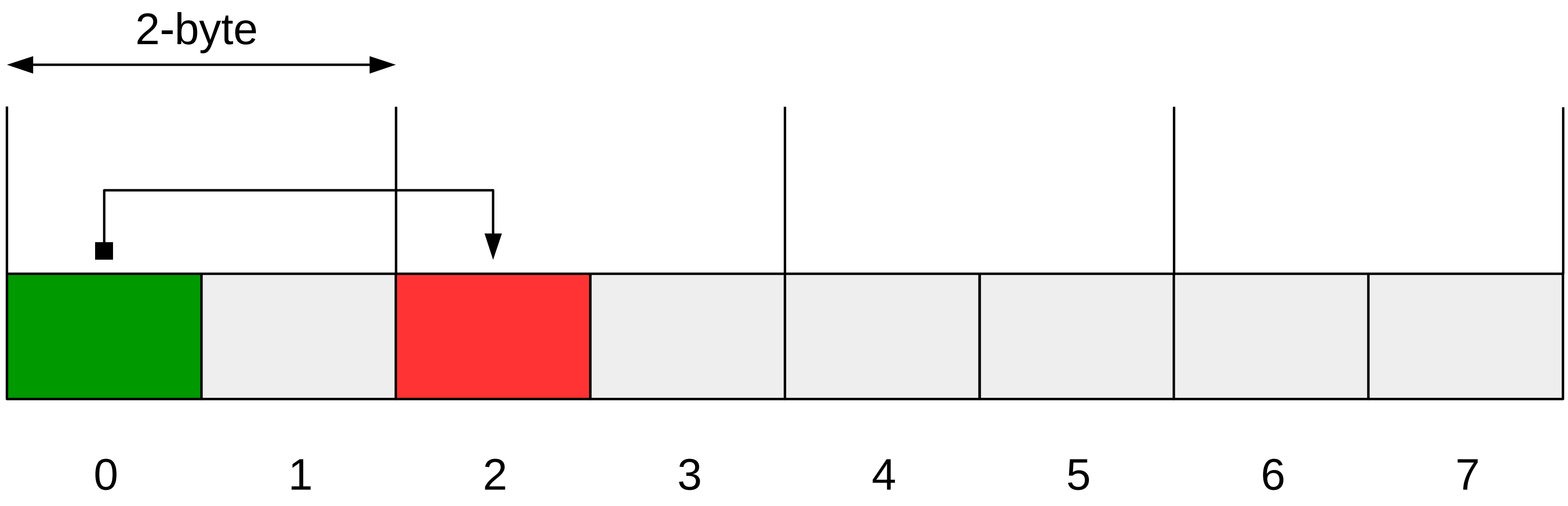


Modelling

Each consecutive memory access has an address dependent energy consumption.

E_0

E.g. $0 \rightarrow 2$

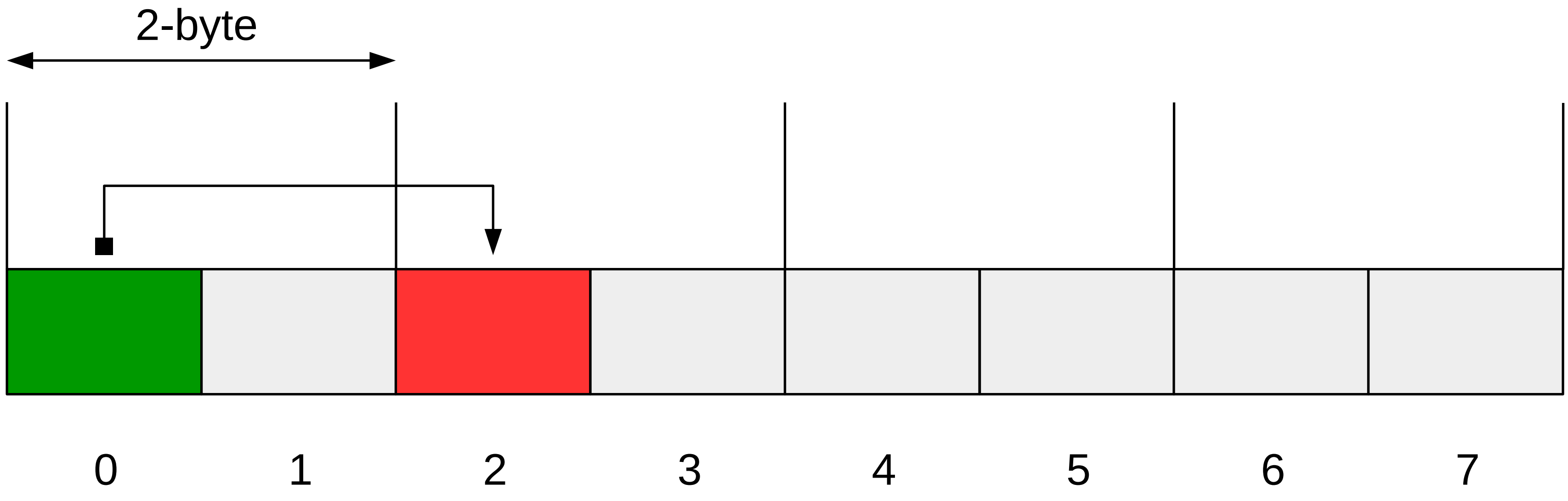


Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $0 \rightarrow 2$

$E_0 + E_1$

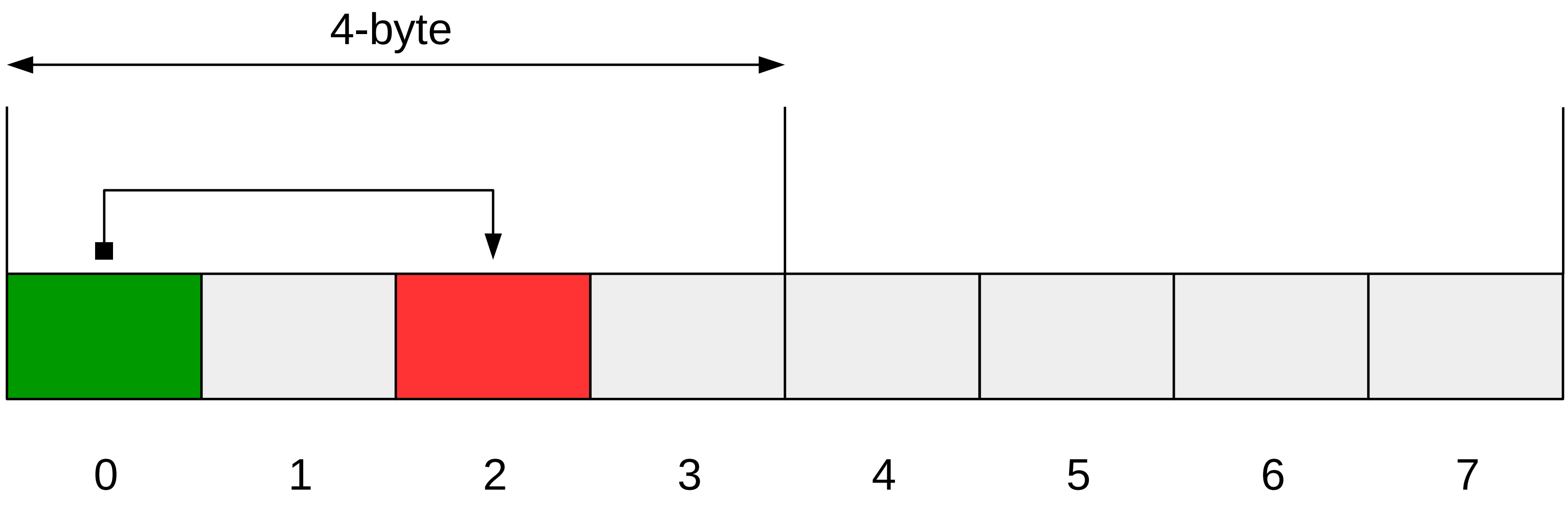


Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $0 \rightarrow 2$

$$E_0 + E_1$$

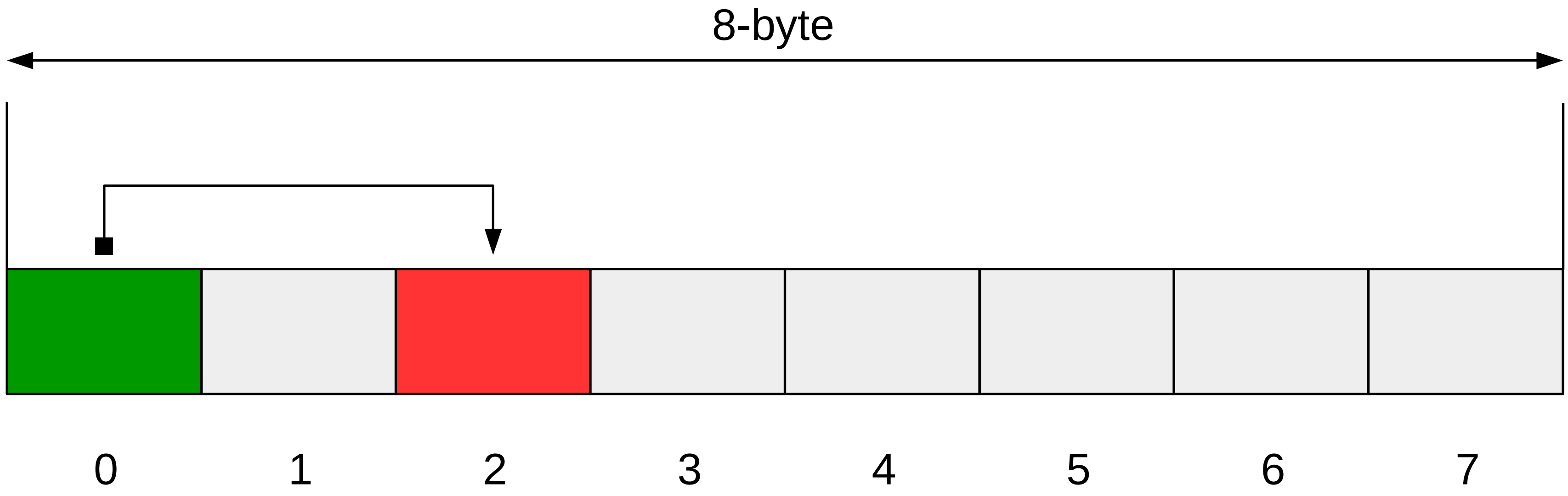


Modelling

Each consecutive memory access has an address dependent energy consumption.

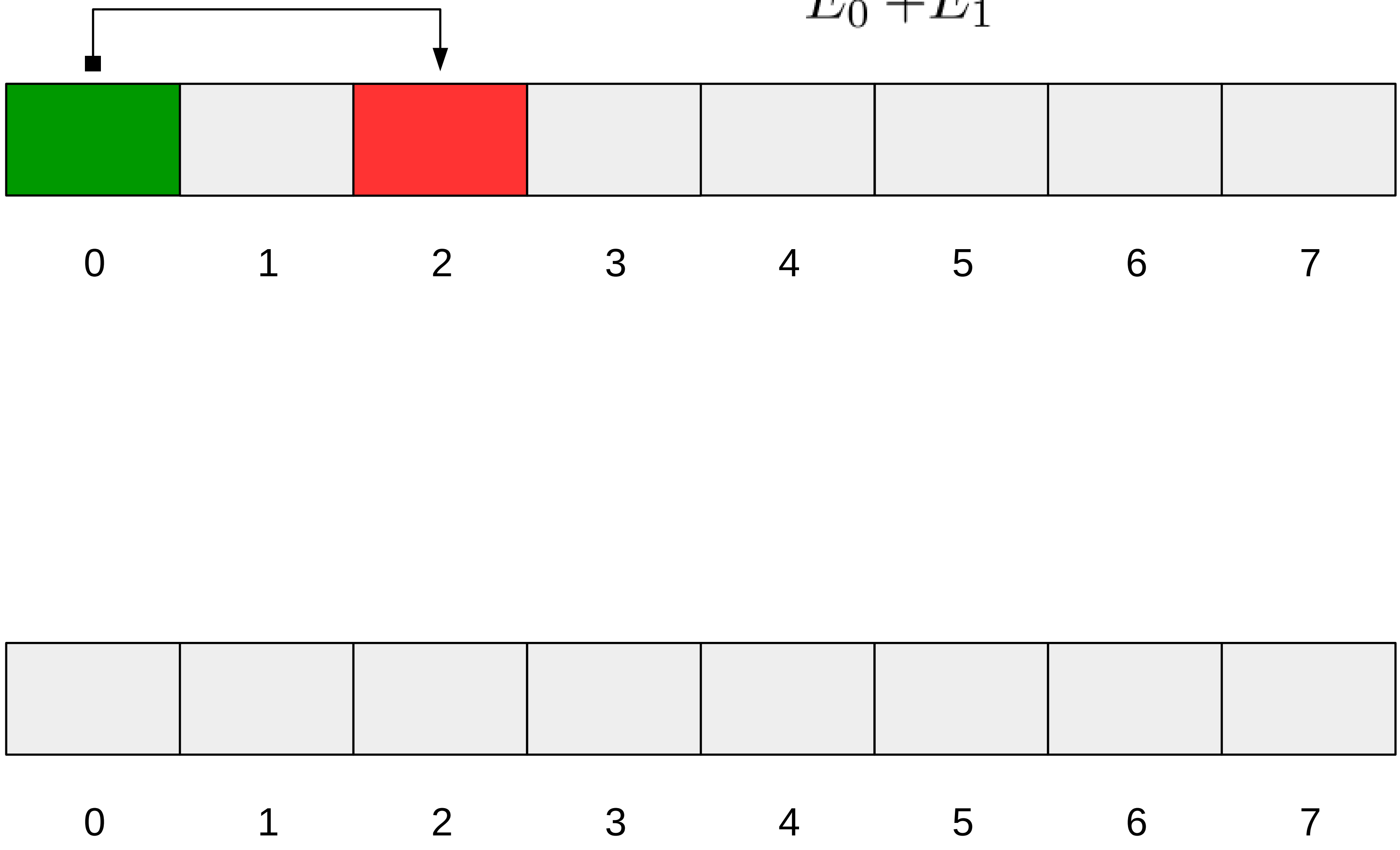
E.g. $0 \rightarrow 2$

$E_0 + E_1$



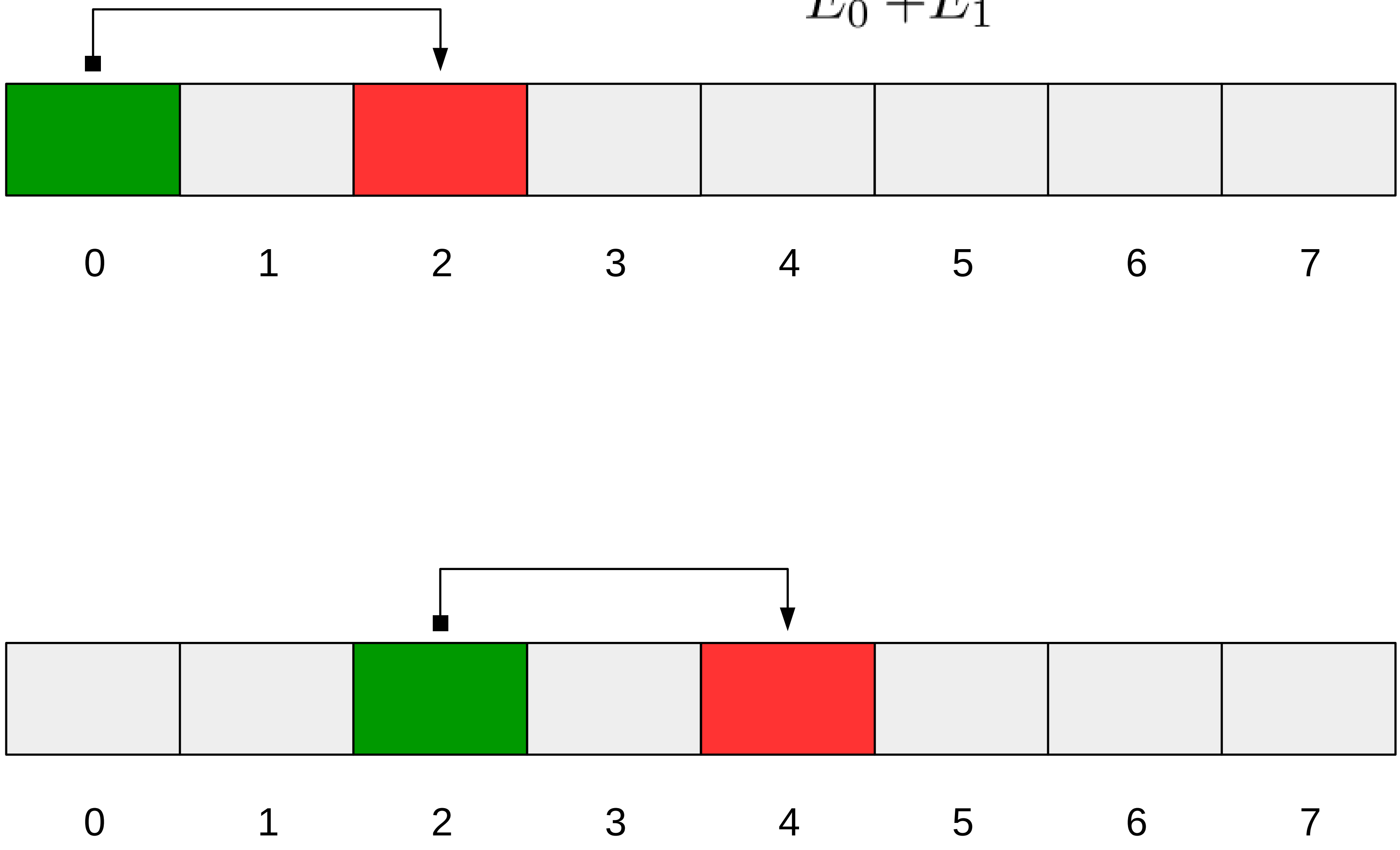
Modelling

$$E_0 + E_1$$

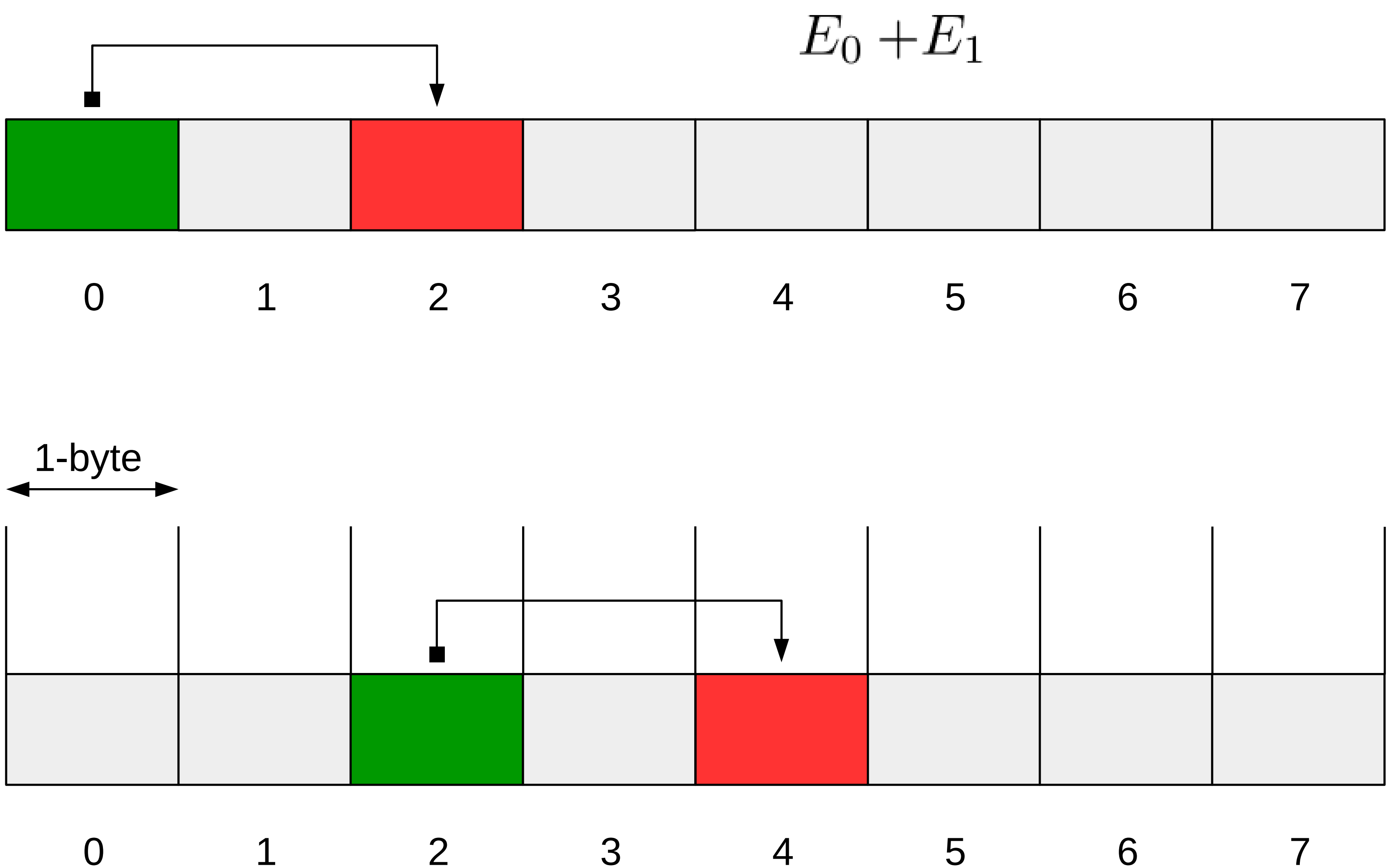


Modelling

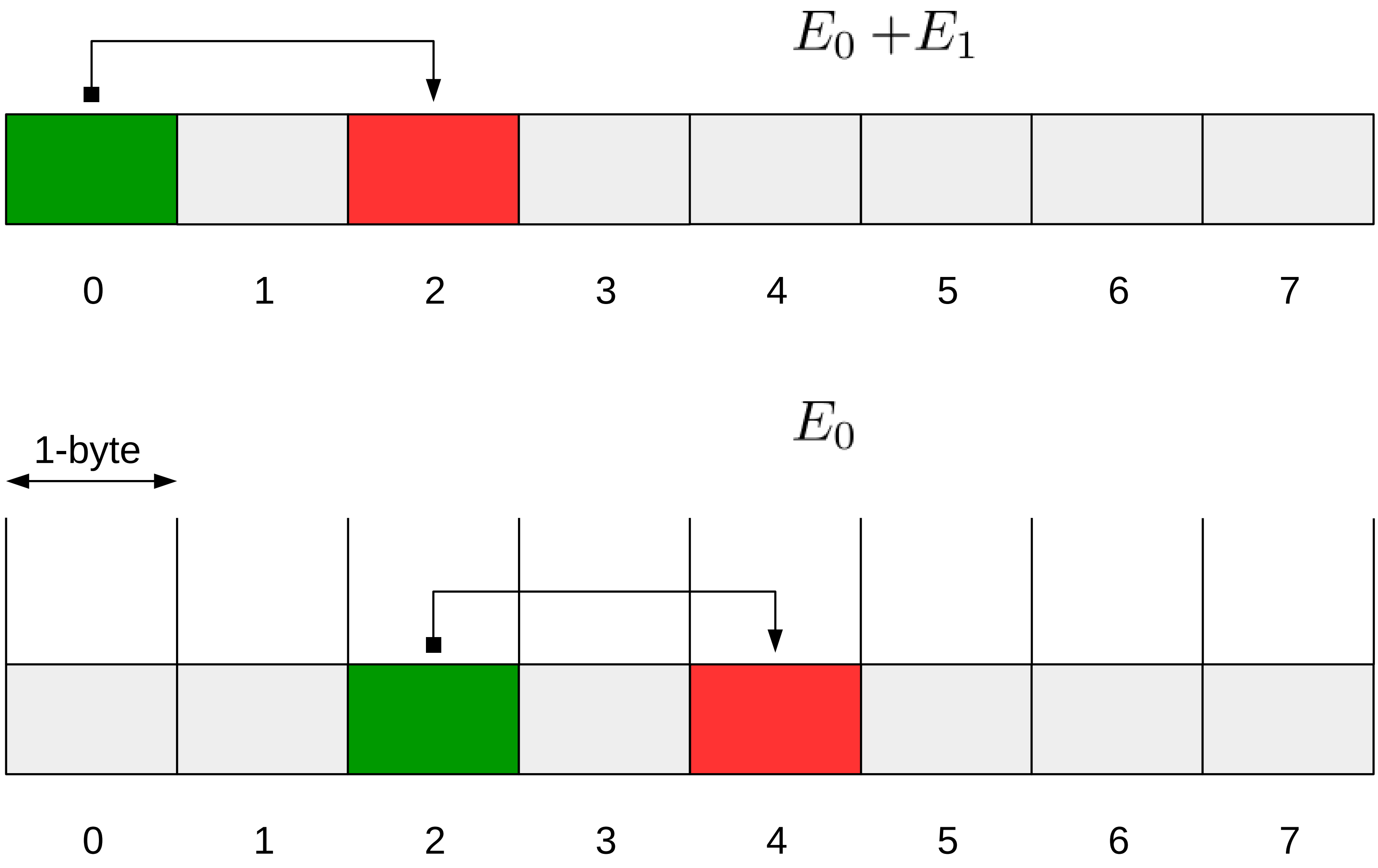
$E_0 + E_1$



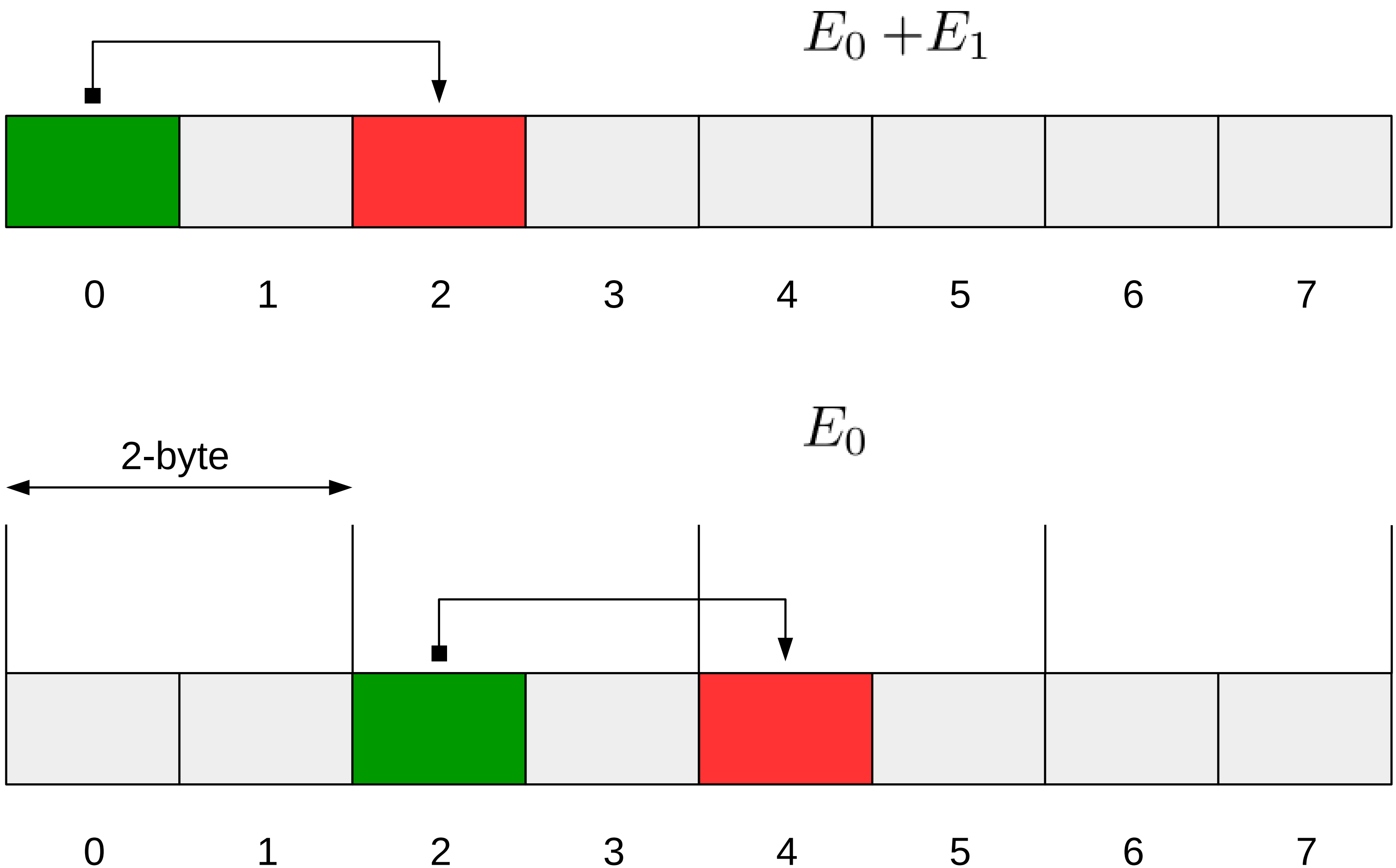
Modelling



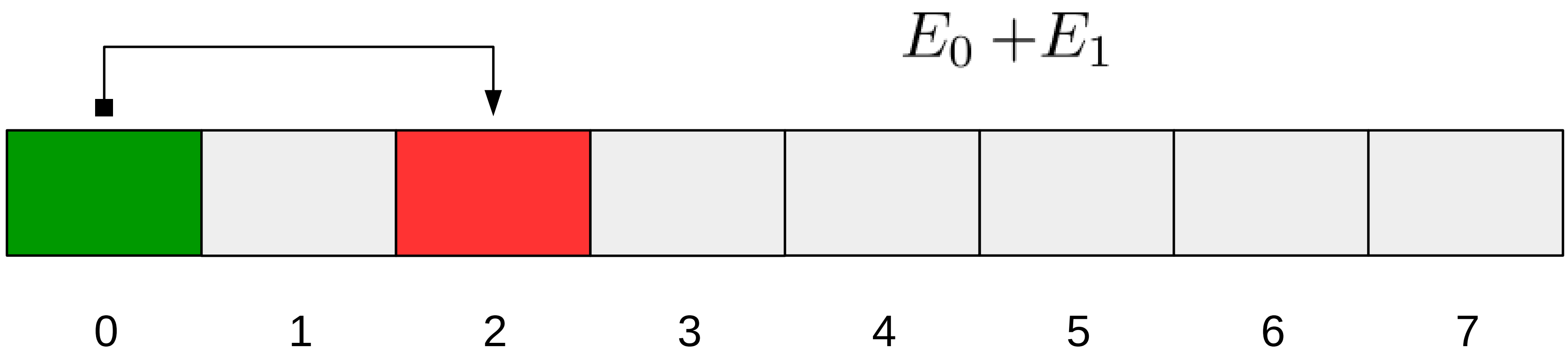
Modelling

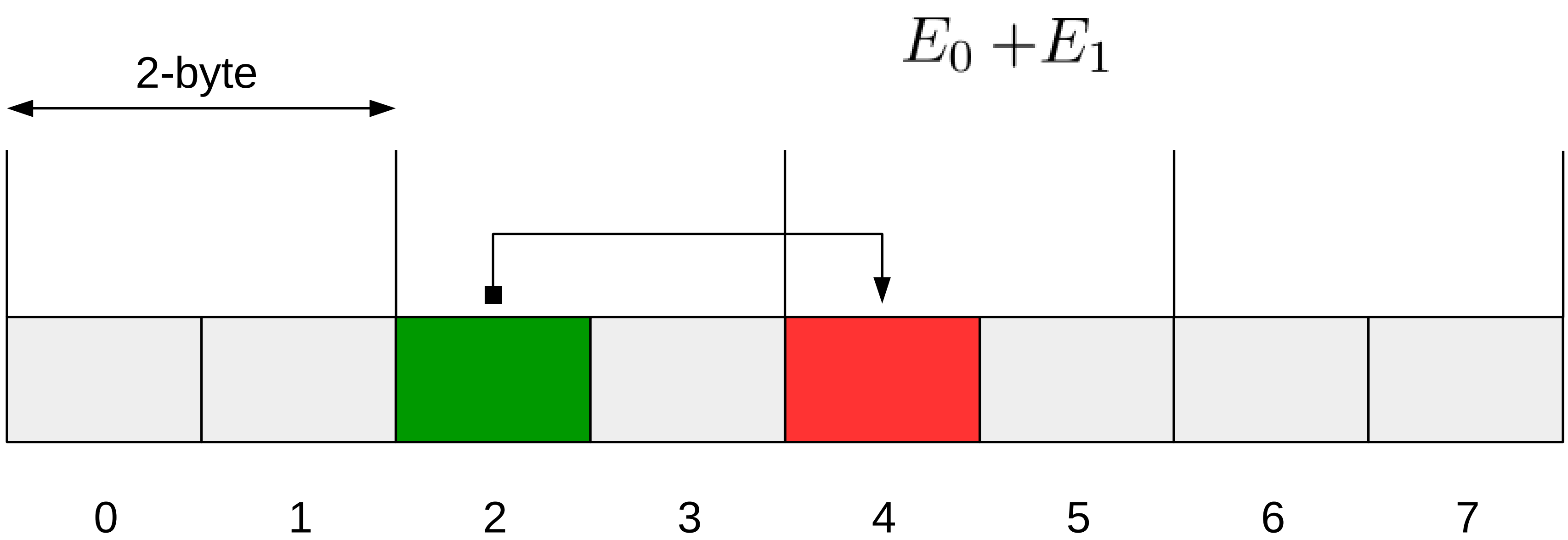


Modelling

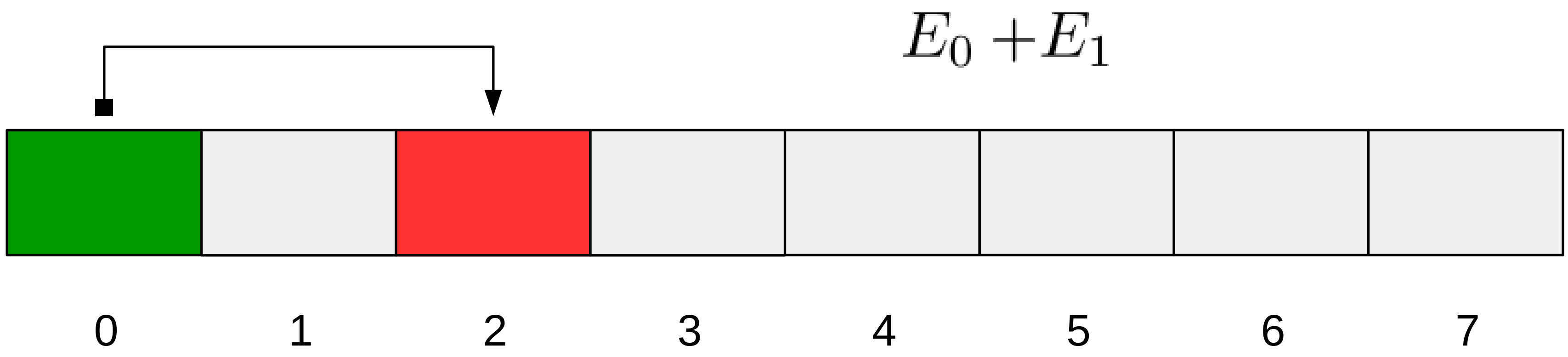


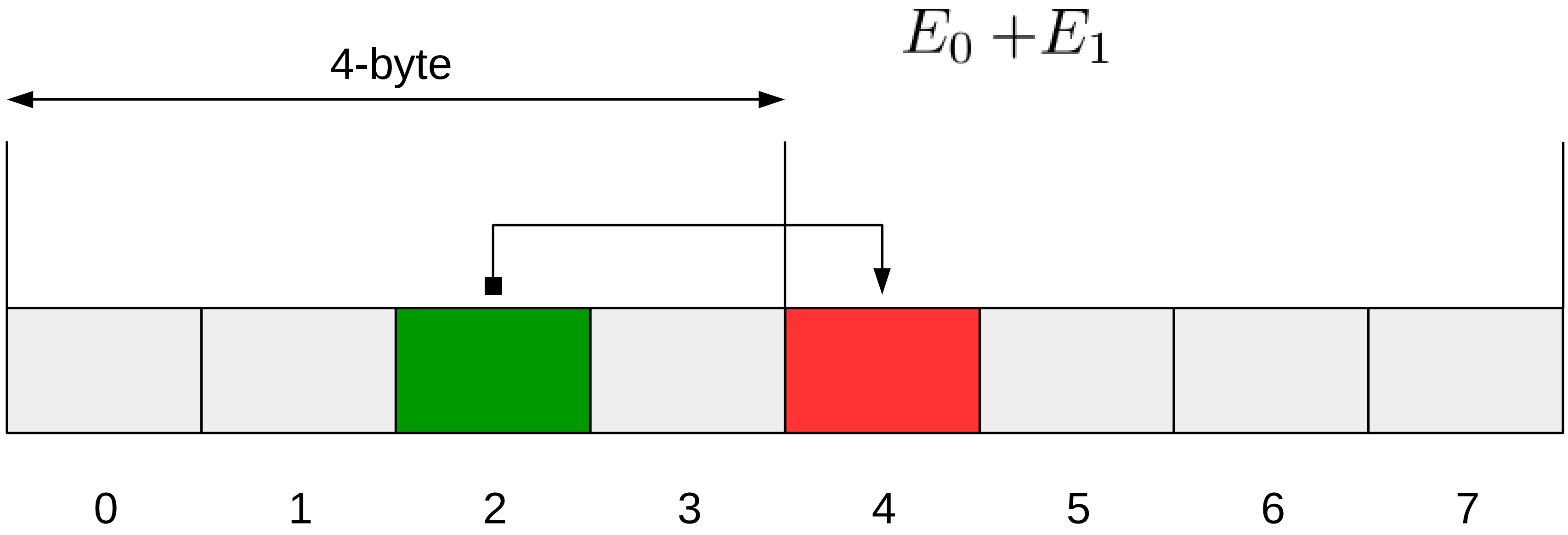
Modelling





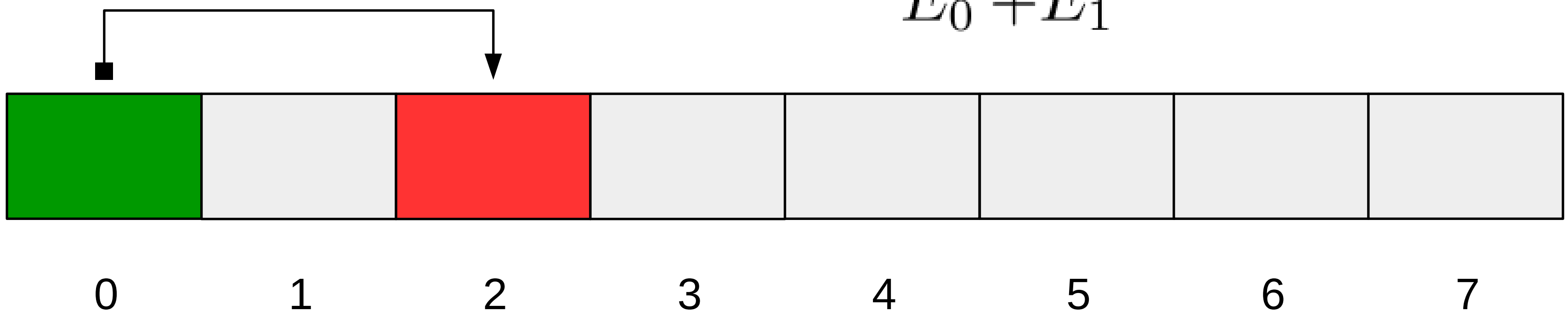
Modelling



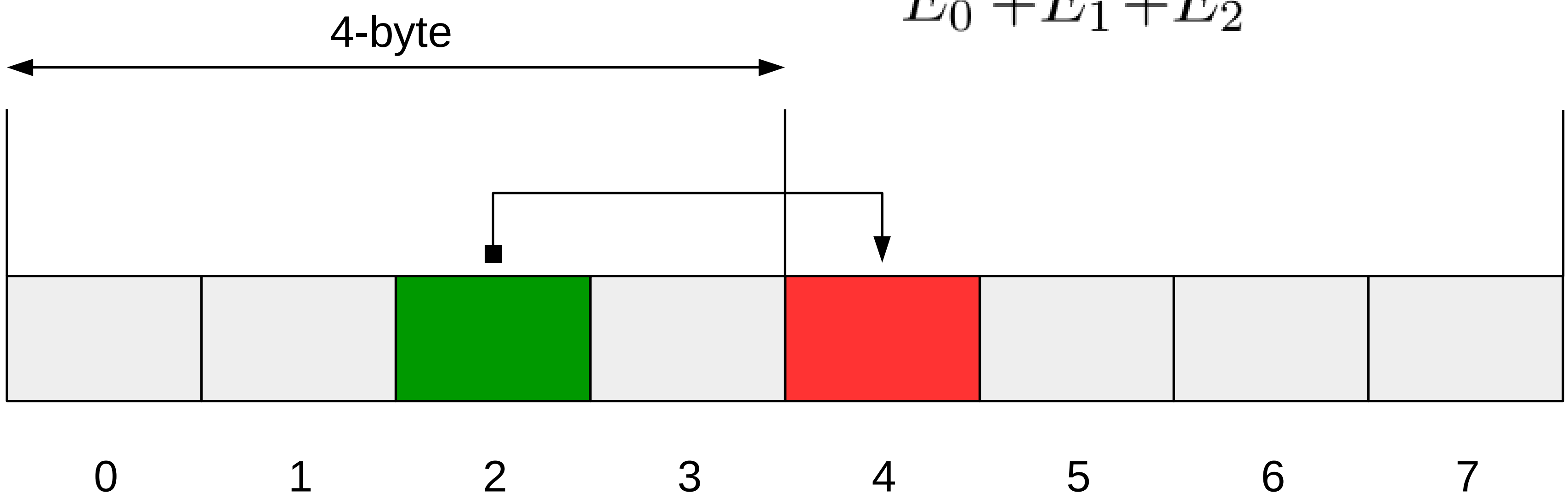


Modelling

$$E_0 + E_1$$

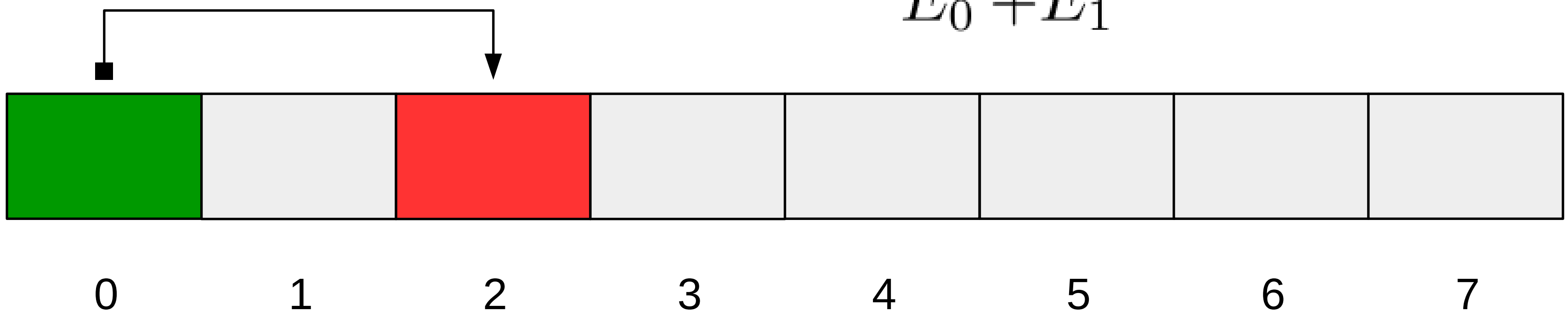


$$E_0 + E_1 + E_2$$

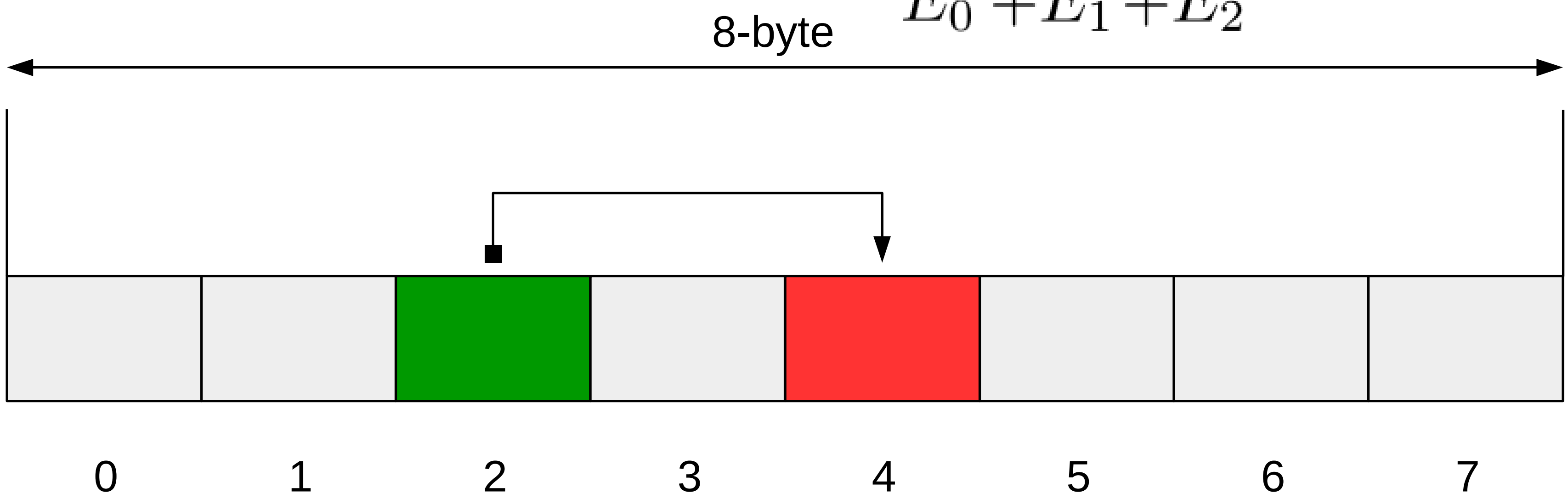


Modelling

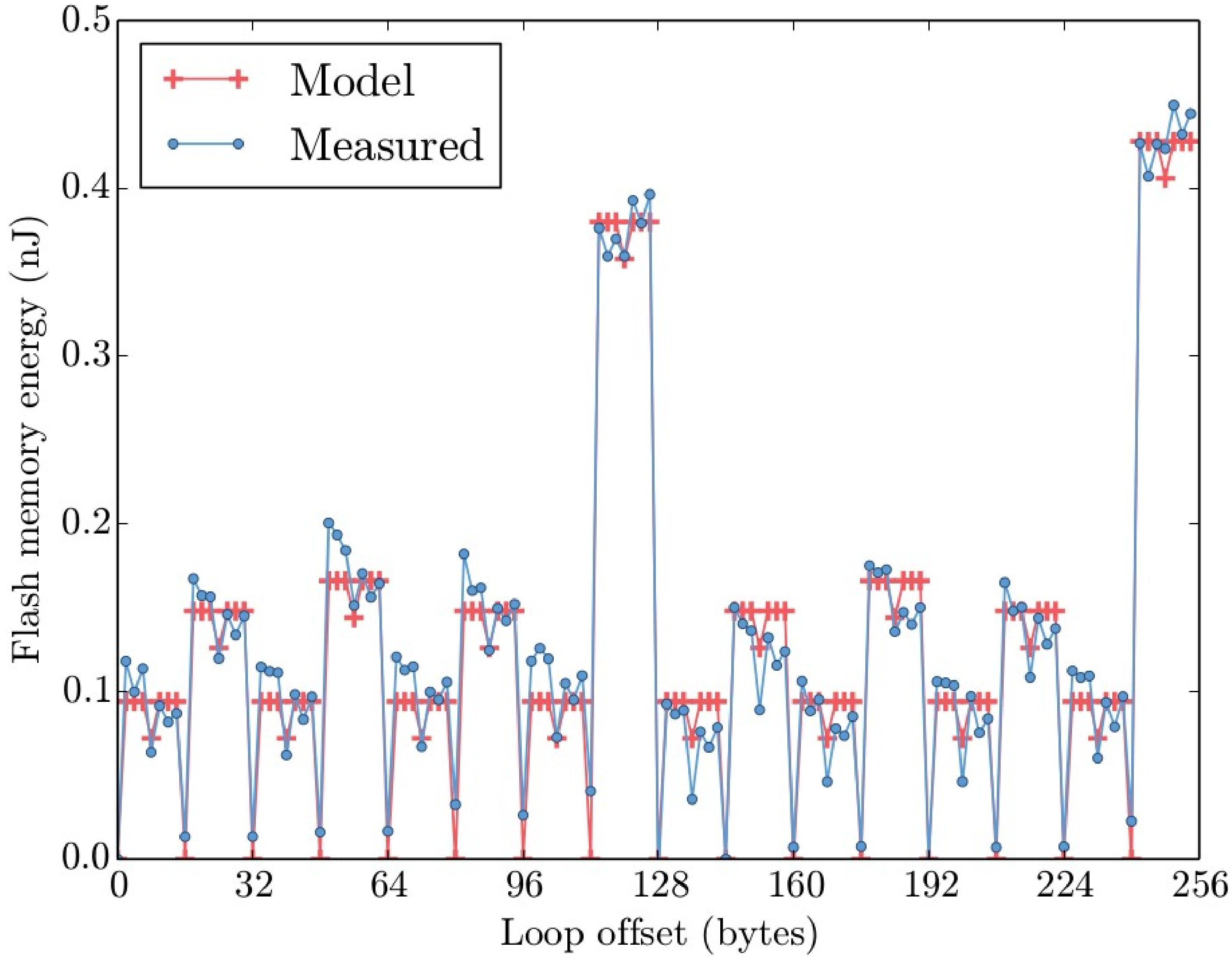
$$E_0 + E_1$$



$$E_0 + E_1 + E_2$$



Cross validation



STM32F0

A potential optimisation

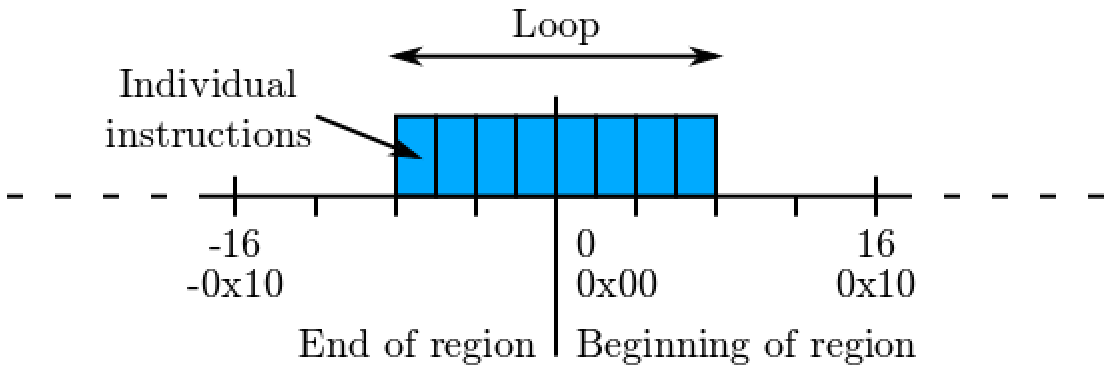
For code which is executed frequently, ensure it crosses as few costly boundaries as possible.

Use the model to make these decisions.

A potential optimisation

For code which is executed frequently, ensure it crosses as few costly boundaries as possible.

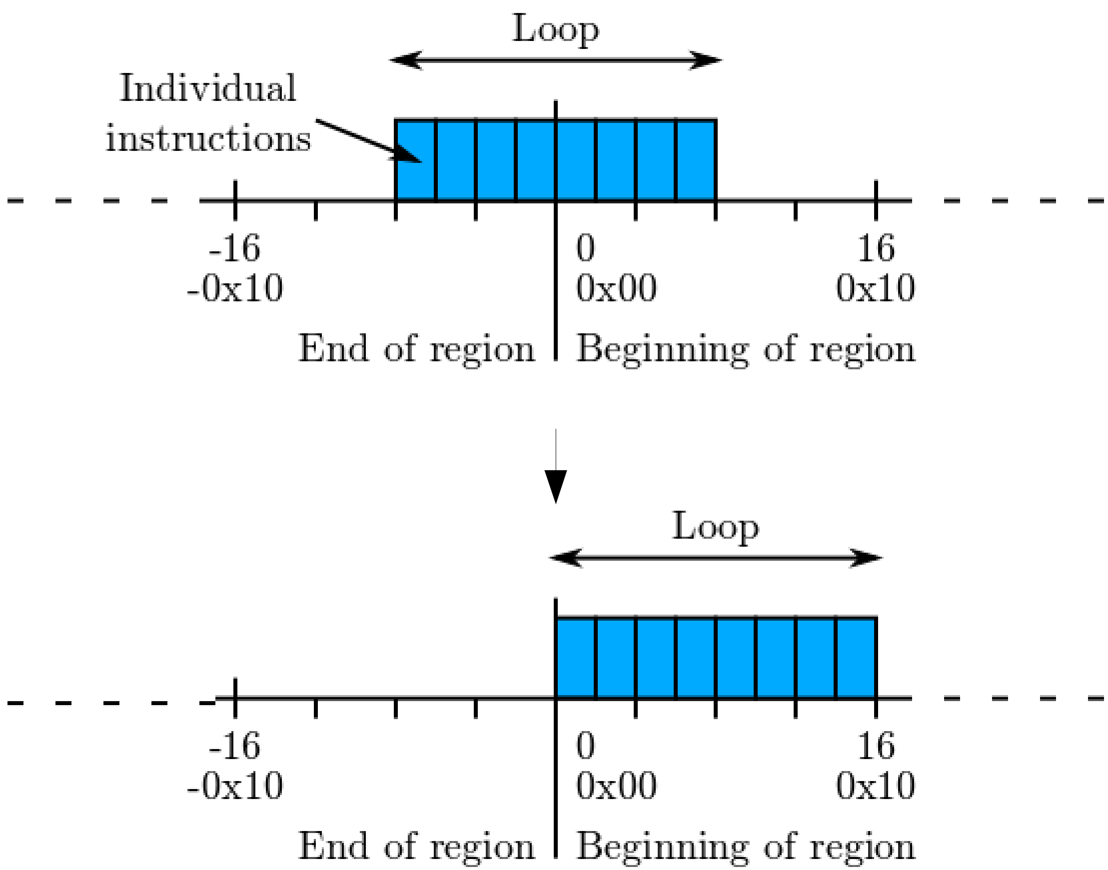
Use the model to make these decisions.



A potential optimisation

For code which is executed frequently, ensure it crosses as few costly boundaries as possible.

Use the model to make these decisions.



A potential optimisation

Unfortunately it doesn't save energy

Overhead of aligning loops outweighs the potential benefit

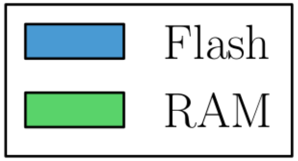
- Many boundary crossings are necessary
- Smaller boundary crossings happen more frequently

More sophistocated implementation may be able to save energy

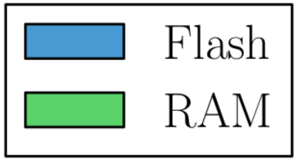
Flash or RAM

Flash or RAM

Execute instructions from

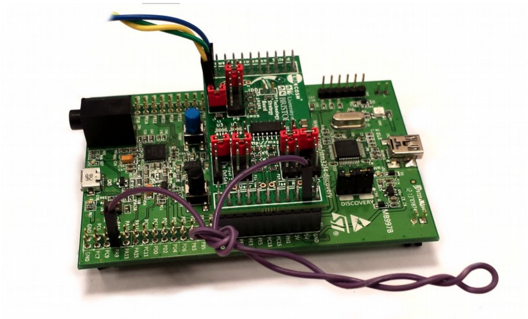


Flash or RAM



Execute instructions from

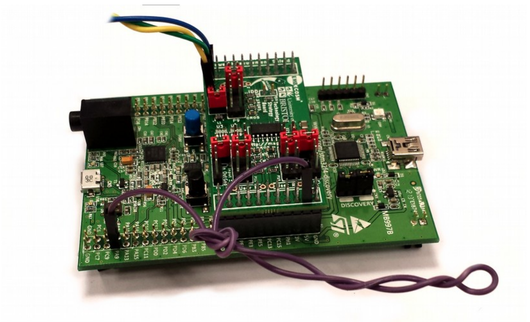
Measure the power dissipation

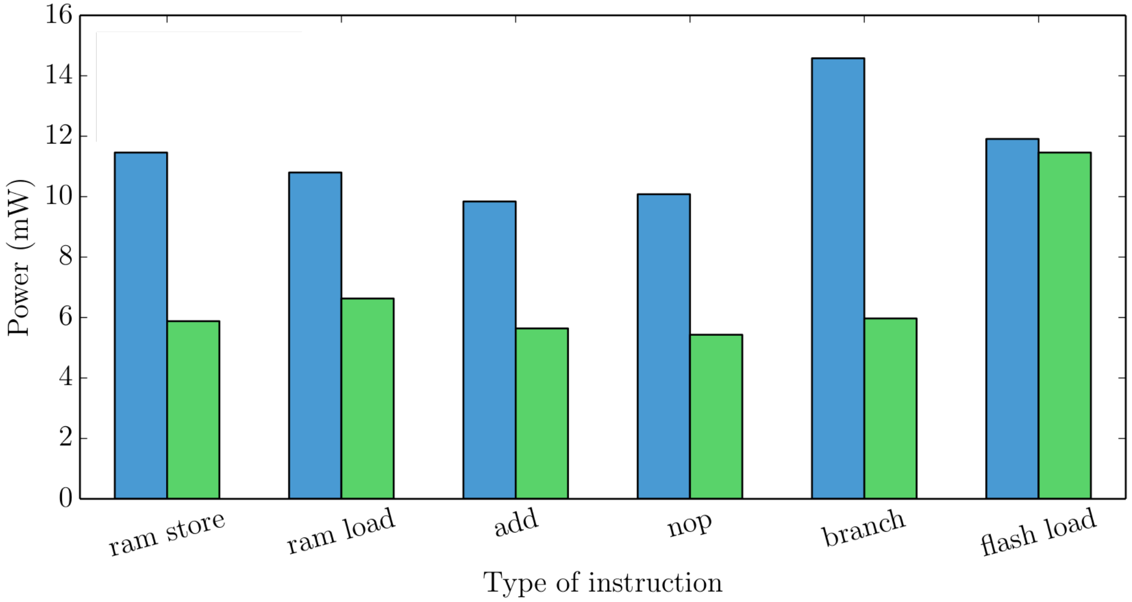


Flash or RAM

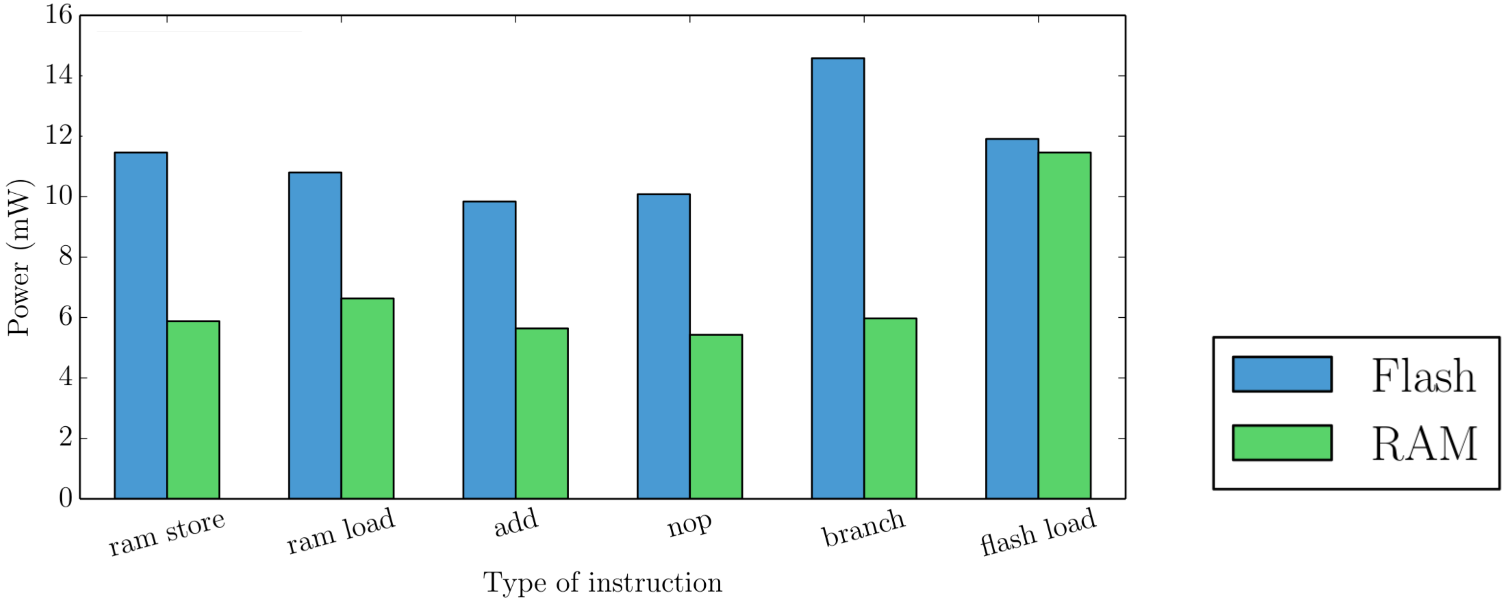
Execute instructions from

Measure the power dissipation





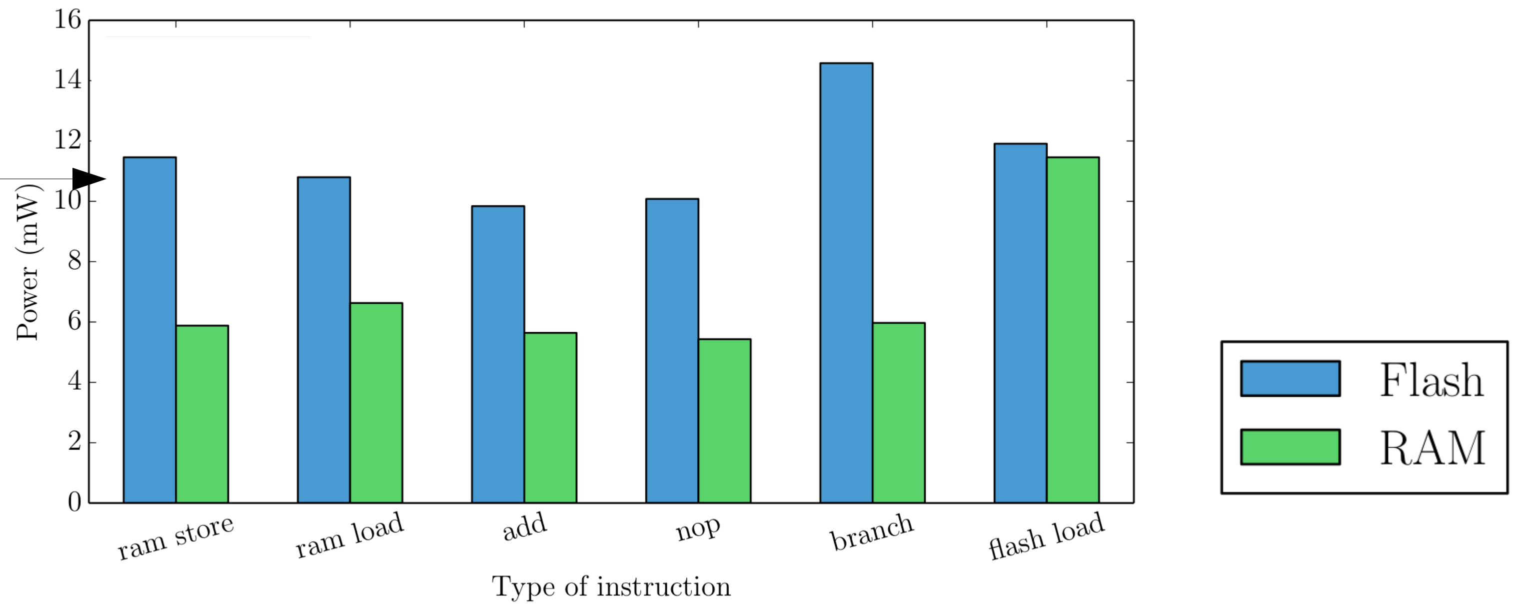
Flash or RAM



Power (mW)
0
2
4
6
8
10
12
14
16
ram store
ram load
add
nop
branch
flash load
Type of instruction
Flash
RAM

Flash or RAM

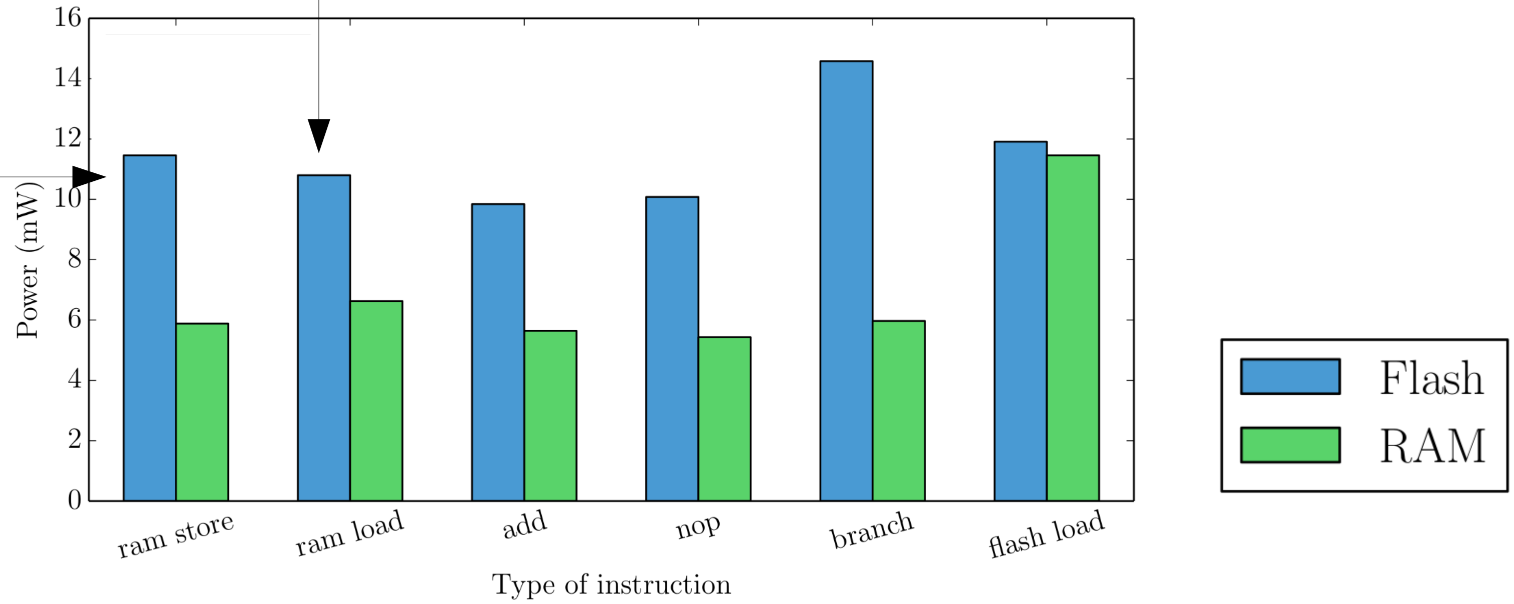
```
; r1 in RAM
str r0, [r1]
```



Flash or RAM

```
; r1 in RAM
str r0, [r1]
```

```
; r1 in RAM
ldr r0, [r1]
```

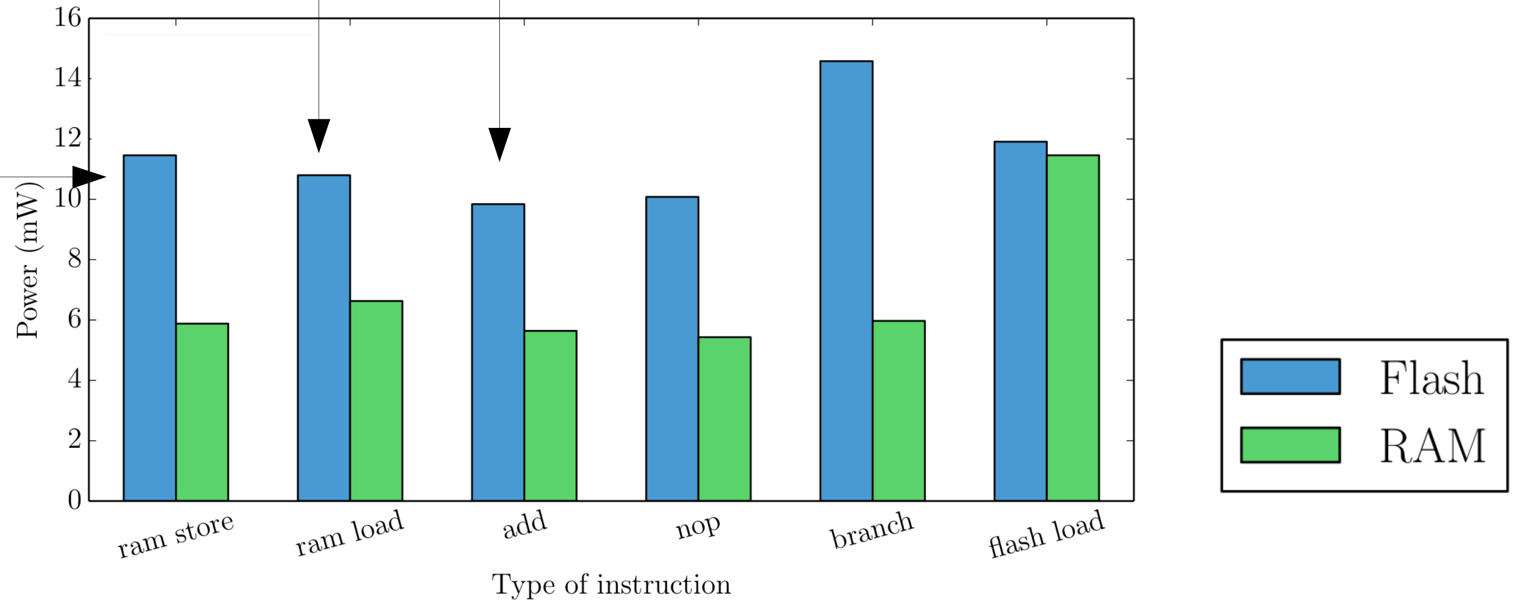


Flash or RAM

```
; r1 in RAM
str r0, [r1]
```

```
add r0, r1
```

```
; r1 in RAM
ldr r0, [r1]
```



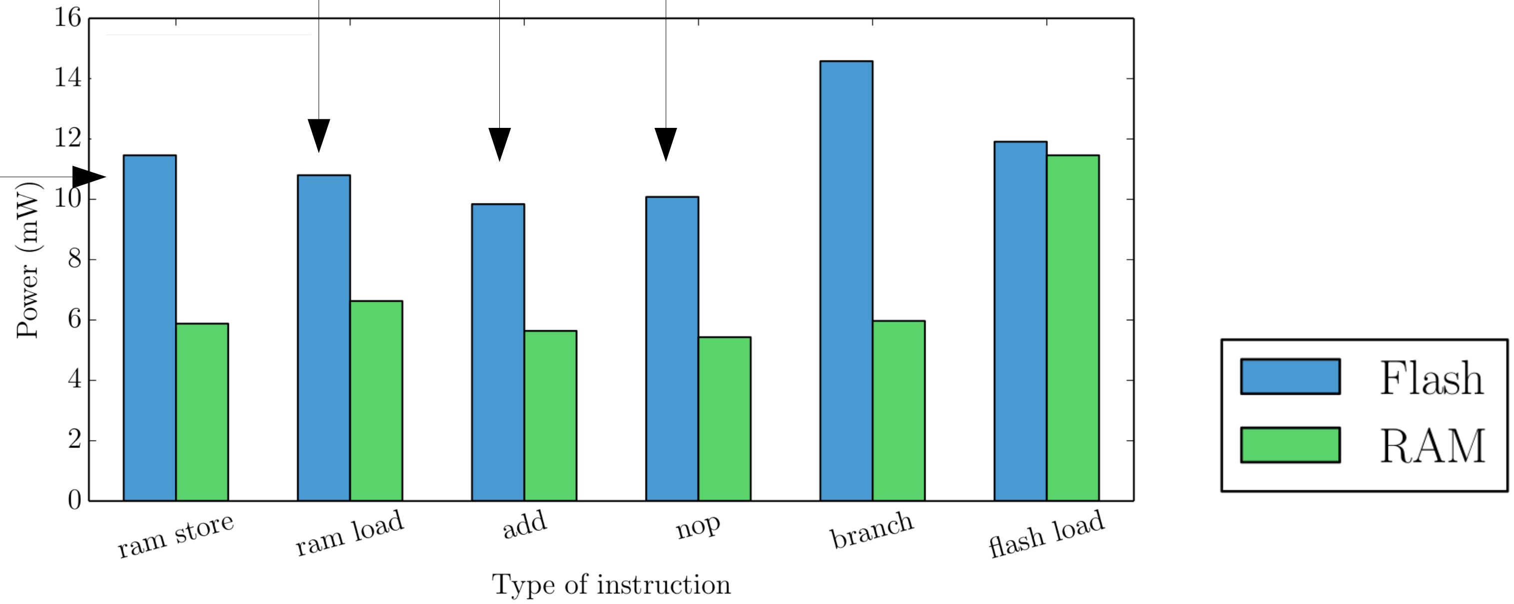
Flash or RAM

```
; r1 in RAM
str r0, [r1]
```

```
; r1 in RAM
ldr r0, [r1]
```

```
add r0, r1
```

```
nop
```



Flash or RAM

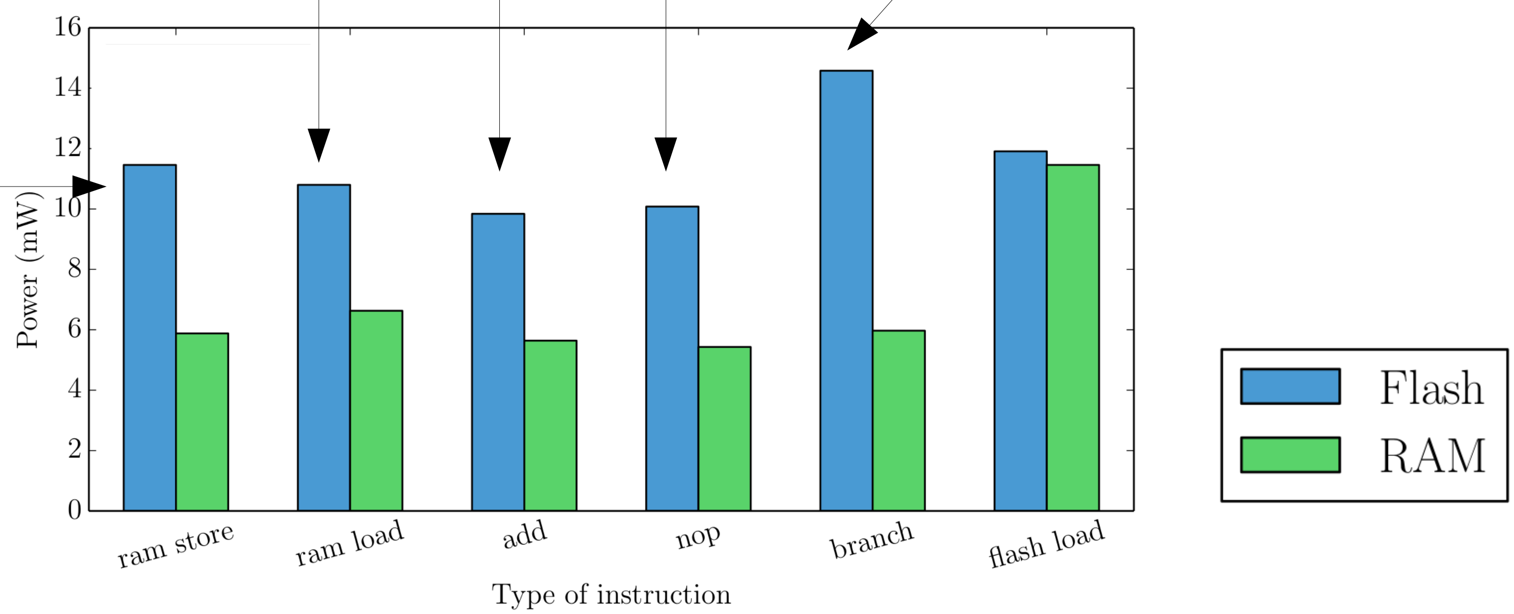
```
; r1 in RAM
str r0, [r1]
```

```
; r1 in RAM
ldr r0, [r1]
```

```
add r0, r1
```

```
nop
```

```
b label
```



Flash or RAM

```
; r1 in RAM
str r0, [r1]
```

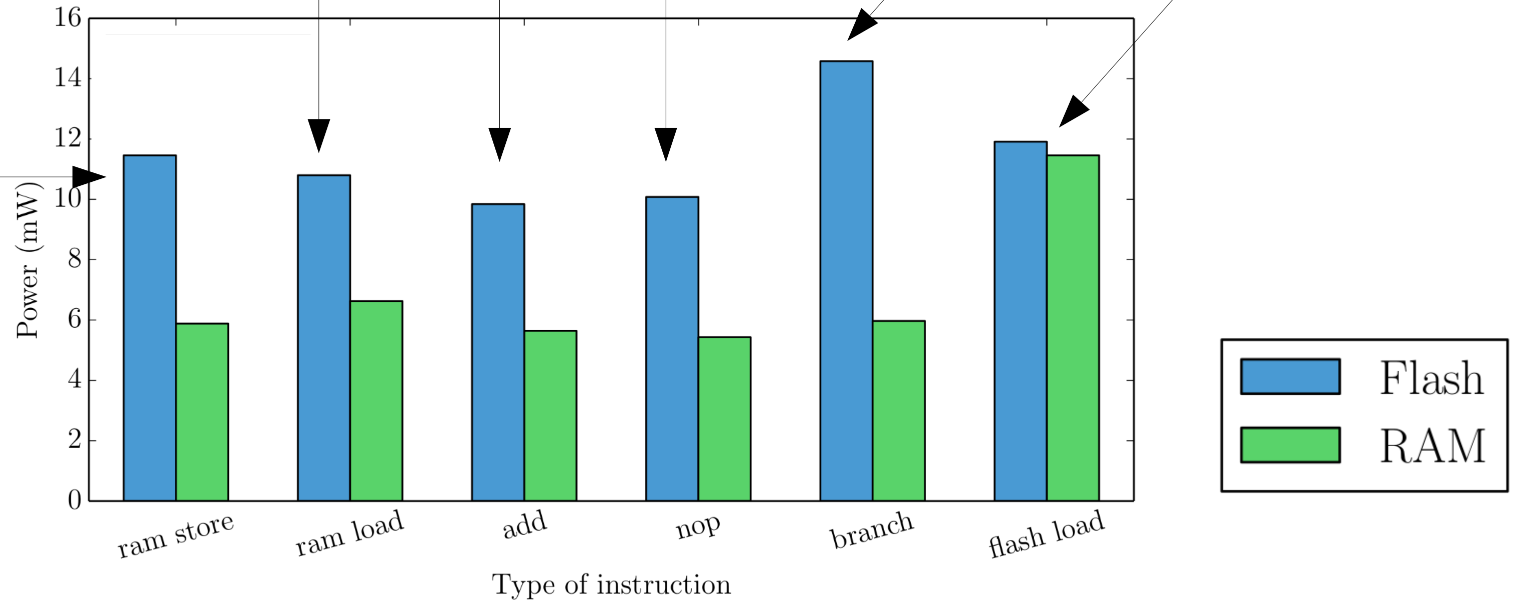
```
; r1 in RAM
ldr r0, [r1]
```

```
add r0, r1
```

```
nop
```

```
b label
```

```
; r1 in flash
ld r0, [r1]
```



Move basic blocks into RAM

Original code

```
int fn(int k)
{
  int i, x;
  x = 1;

  for(i = 0; i < 64; ++i)
  {
    x *= k;
  }

  if(x > 255)
    x = 255;

  return x;
}
```

init

loop

if

iftrue

return

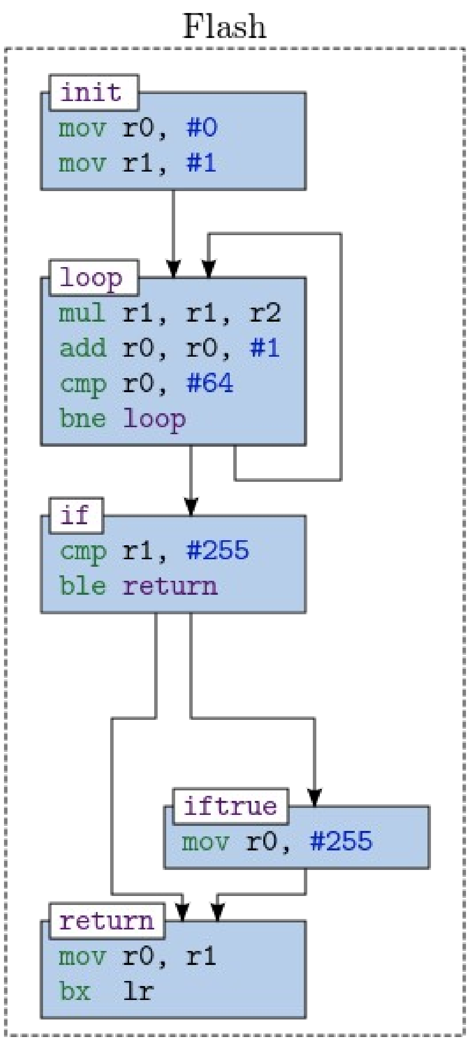
Move basic blocks into RAM

Original code

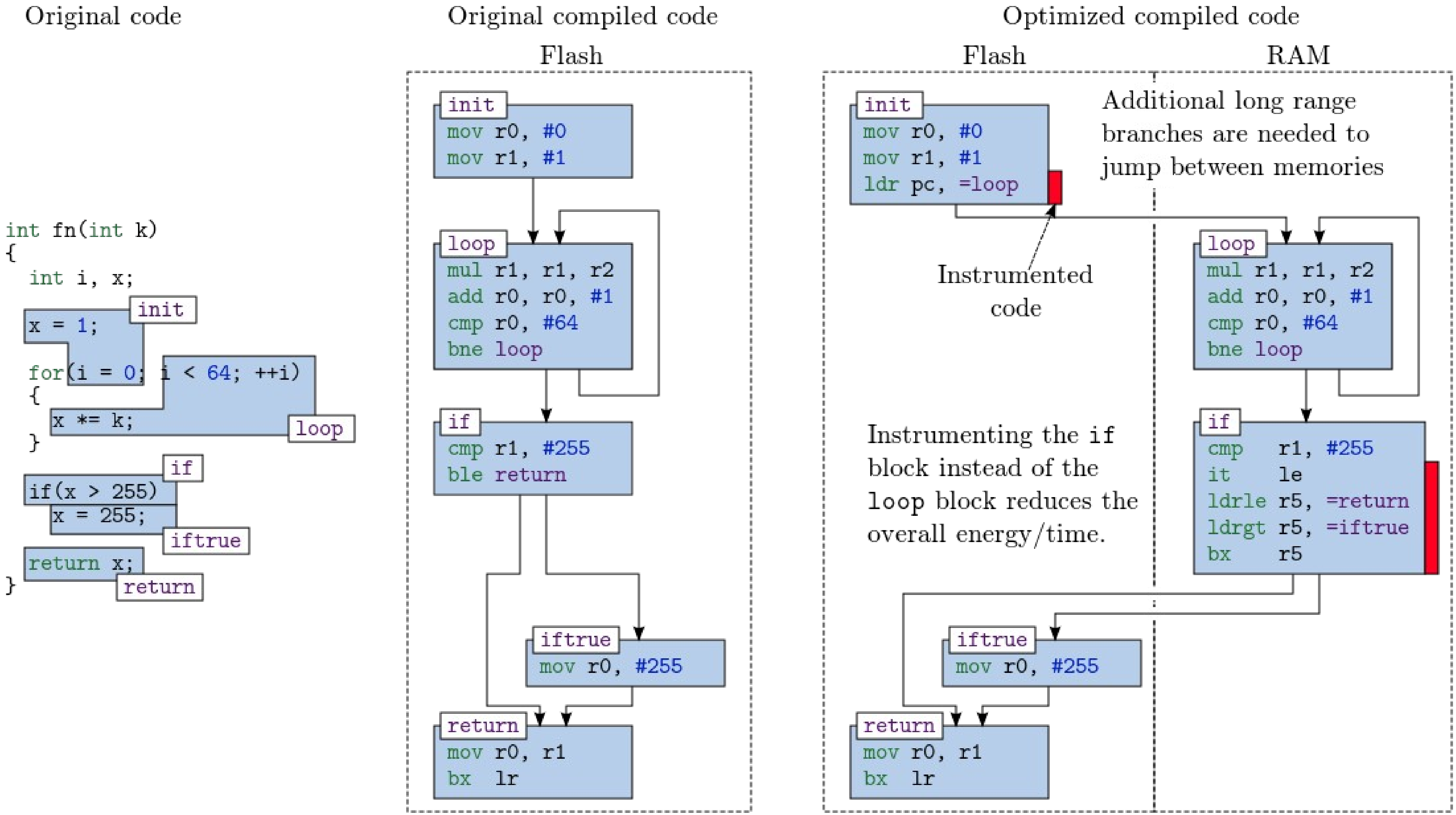
```
int fn(int k)
{
  int i, x;
                     init
  x = 1;

  for(i = 0; i < 64; ++i)
  {
    x *= k;                         loop
  }
                  if
  if(x > 255)
    x = 255;
                  iftrue
  return x;
}               return
```

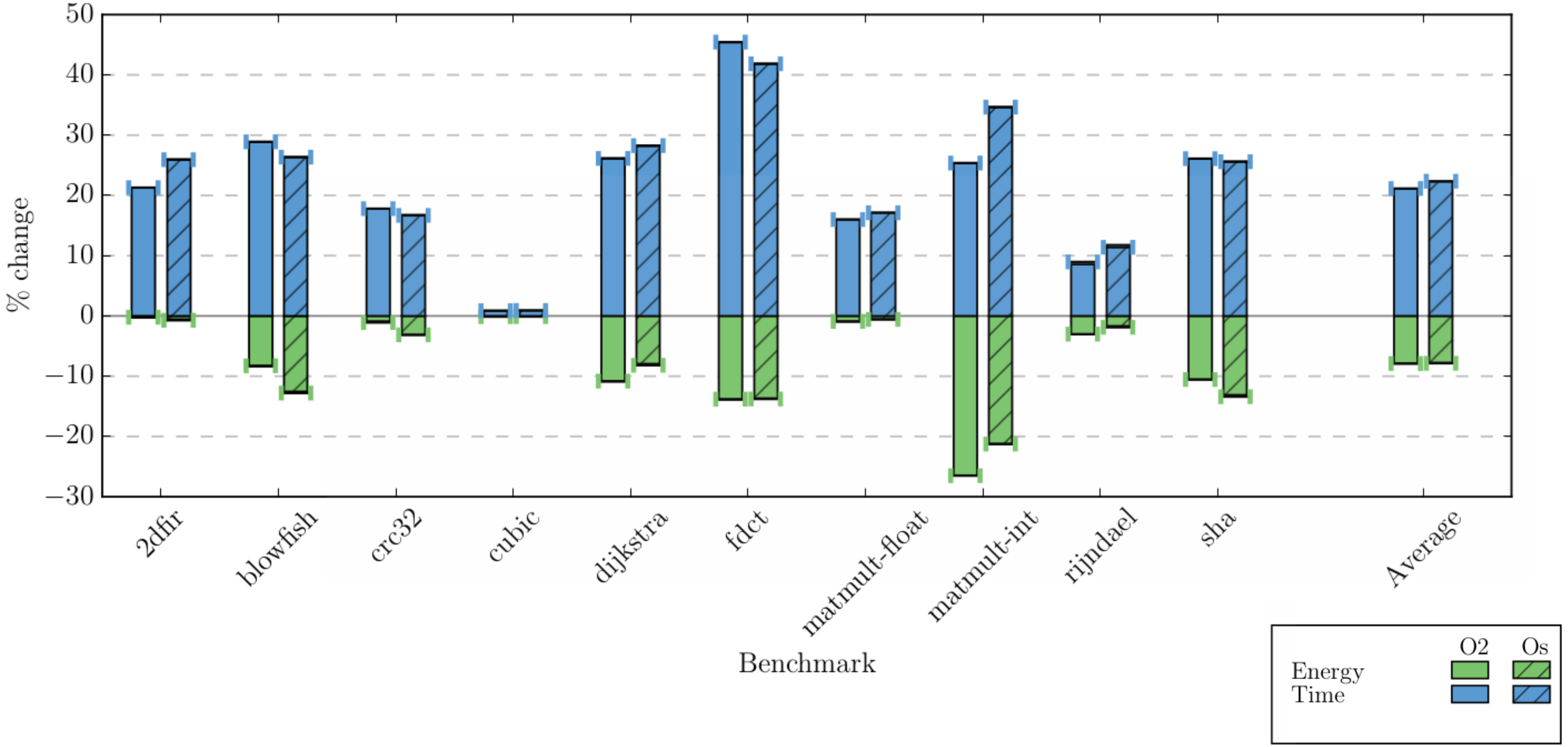
Original compiled code



Move basic blocks into RAM



Results



50
40
30
20
10
0
−10
−20
−30
% change
2dfir
blowfish
crc32
cubic
dijkstra
fdct
matmult-float
matmult-int
rijndael
sha
Average
Benchmark
O2
Os
Energy
Time

Optimisations for energy

The RAM overlay can save significant energy

- Up to 26%

This is achieved by reducing the k_P coefficient

- The execution time actually increases.

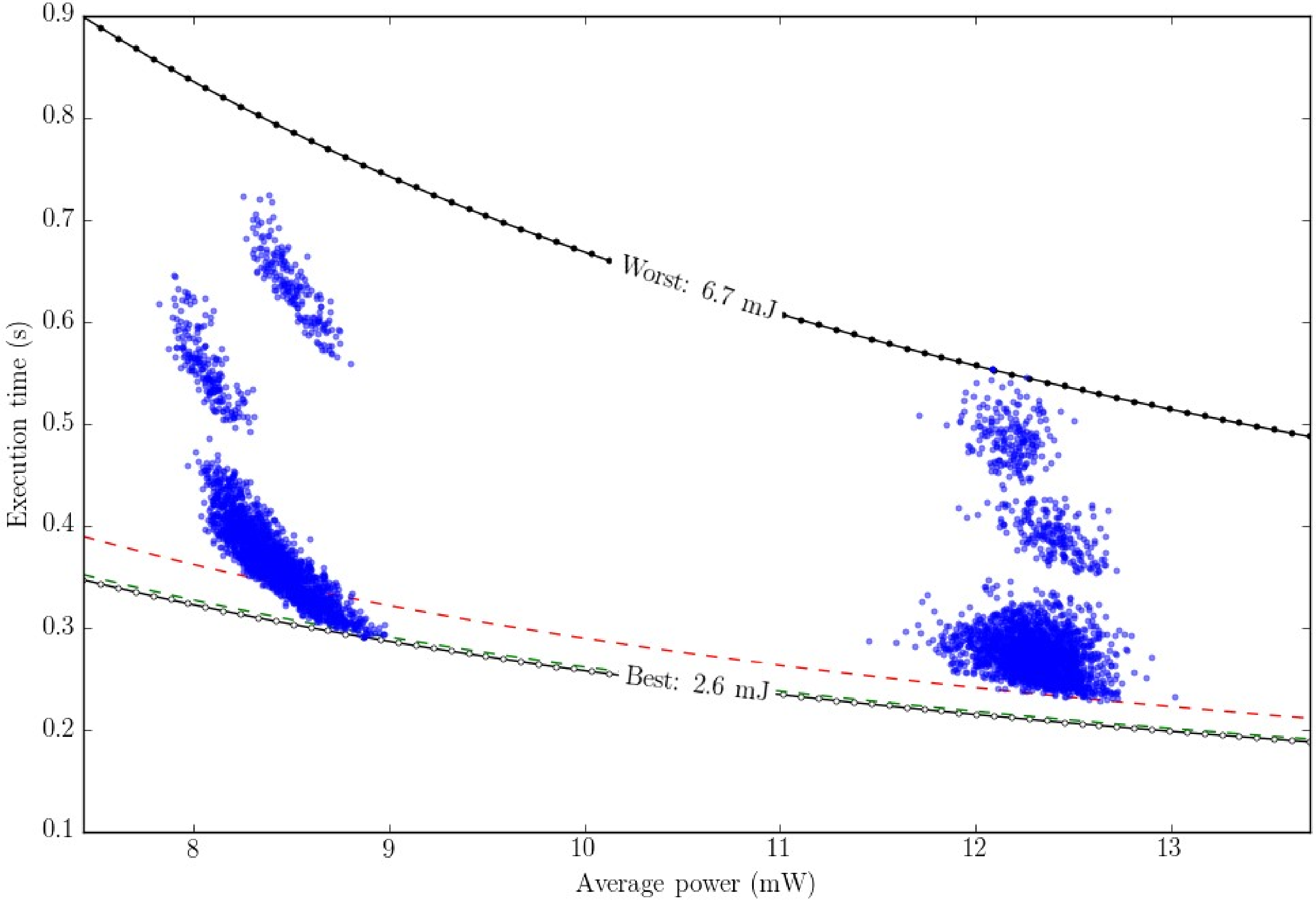
Aligning to flash boundaries is currently ineffective.

- Many of the boundary changes are necessary
- Other types of optimisations may succeed

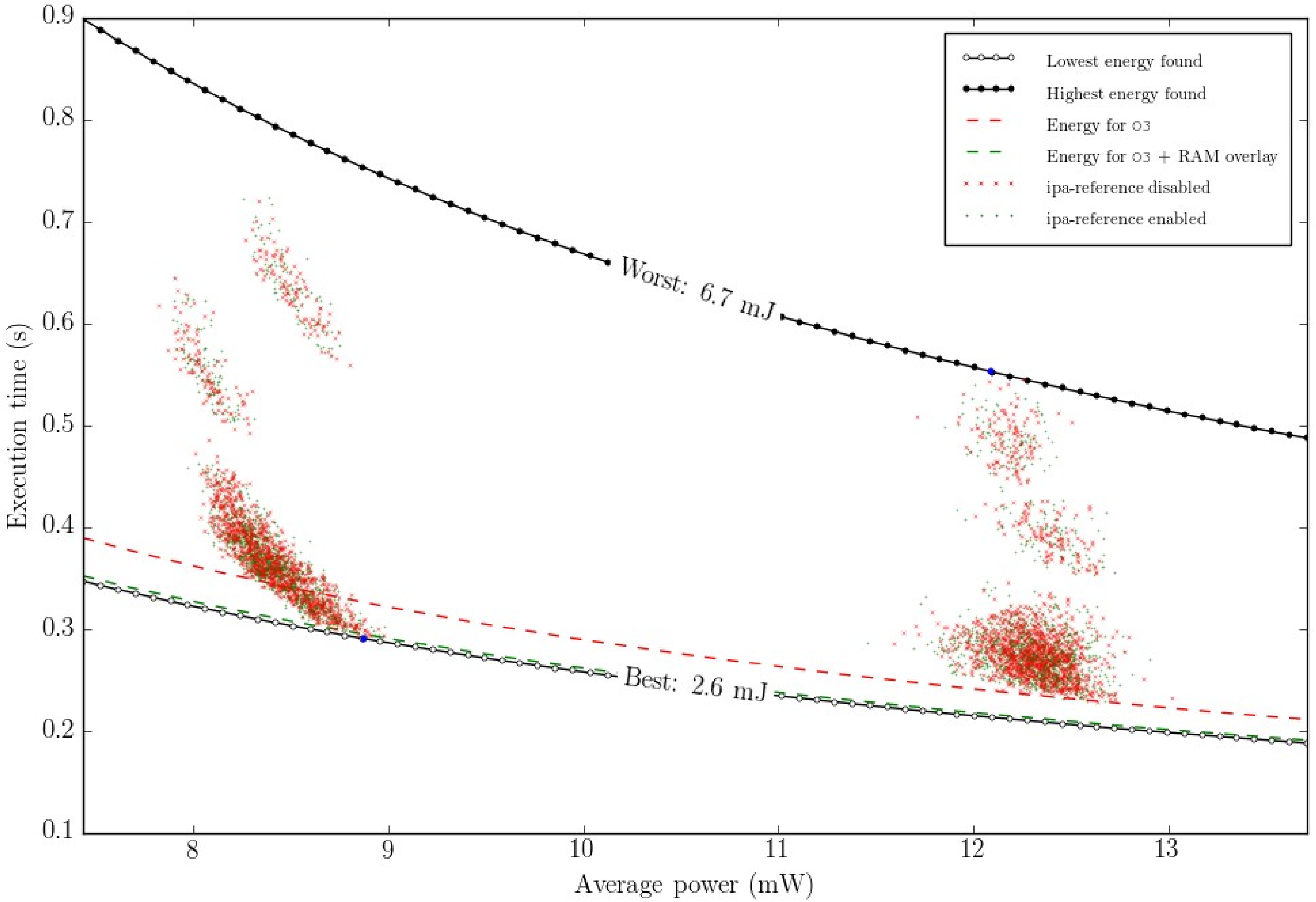
Overview

- *Introduction*
- *Optimisations for execution time*
- *Optimisations for energy consumption*
- ***Combining optimisations for energy and time***
- *Conclusion*

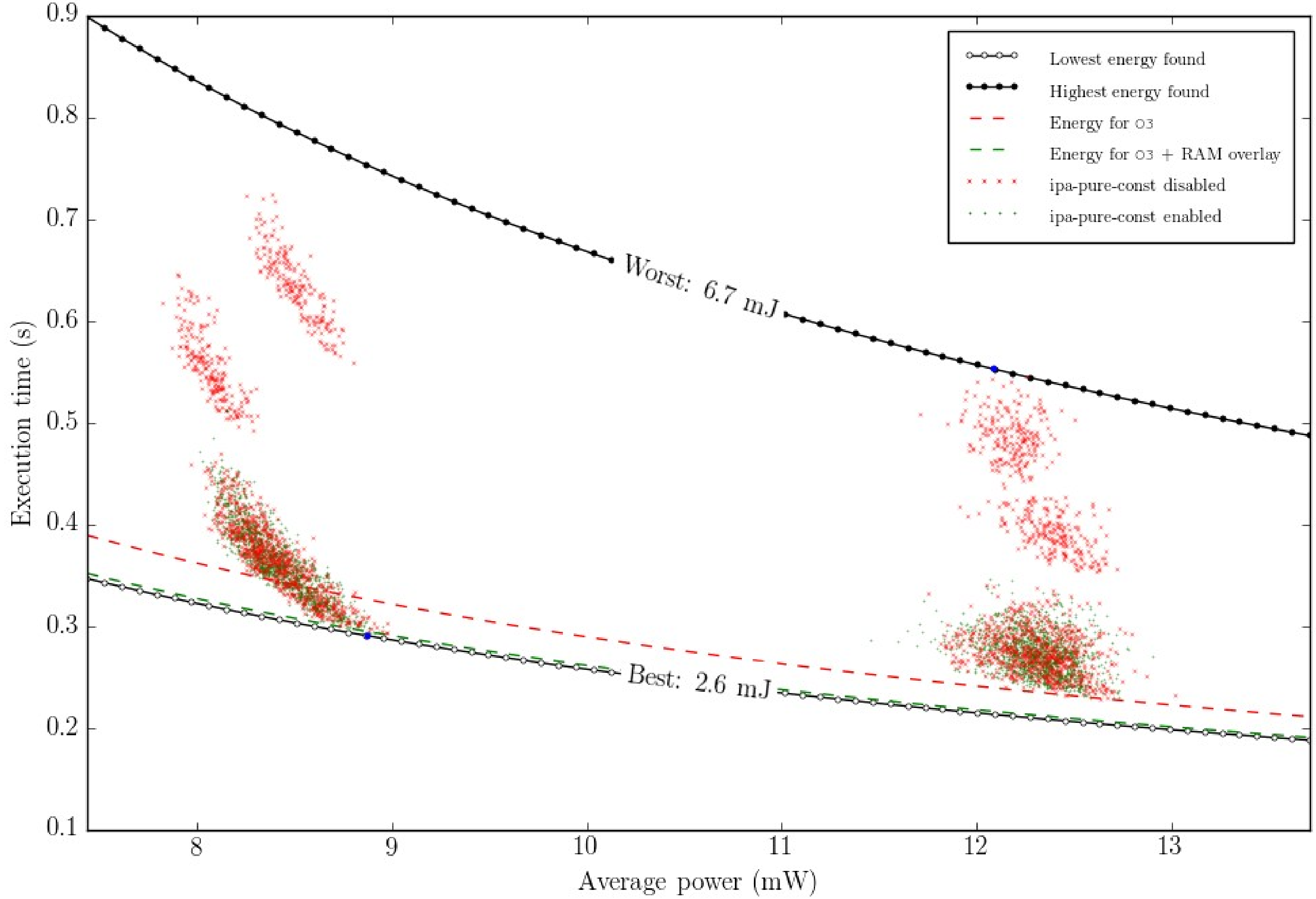
Combining optimisations



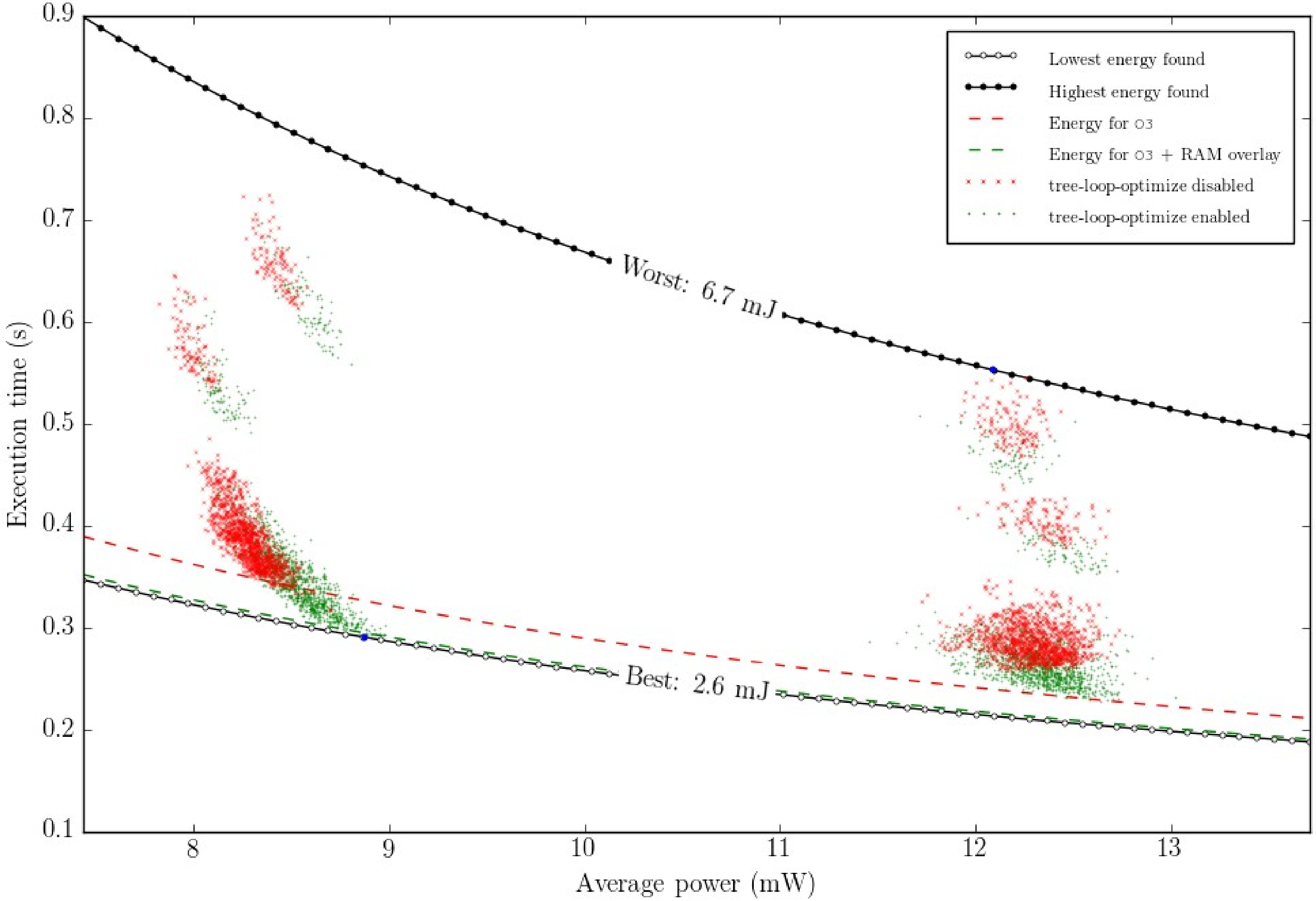
Combining optimisations



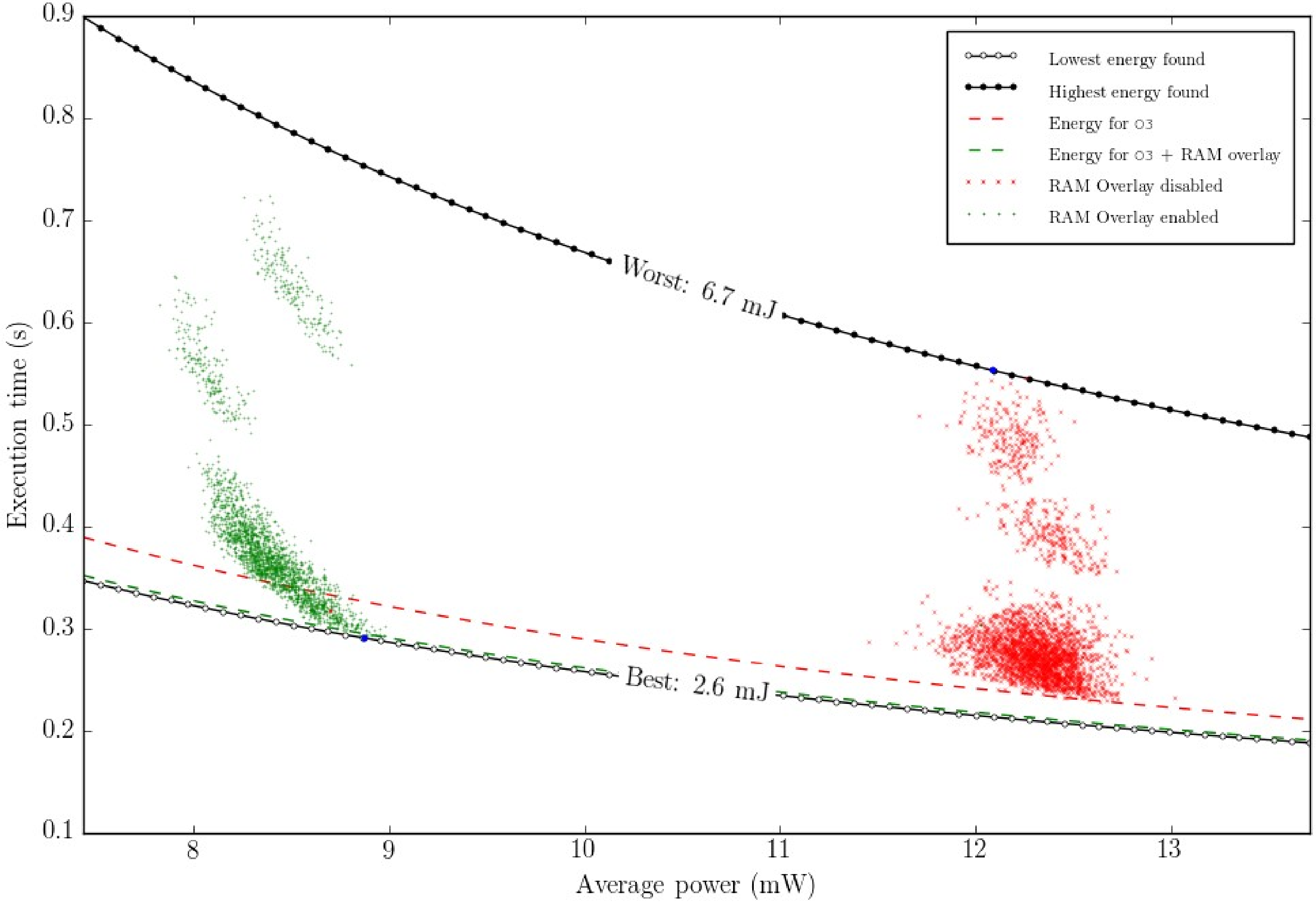
Combining optimisations



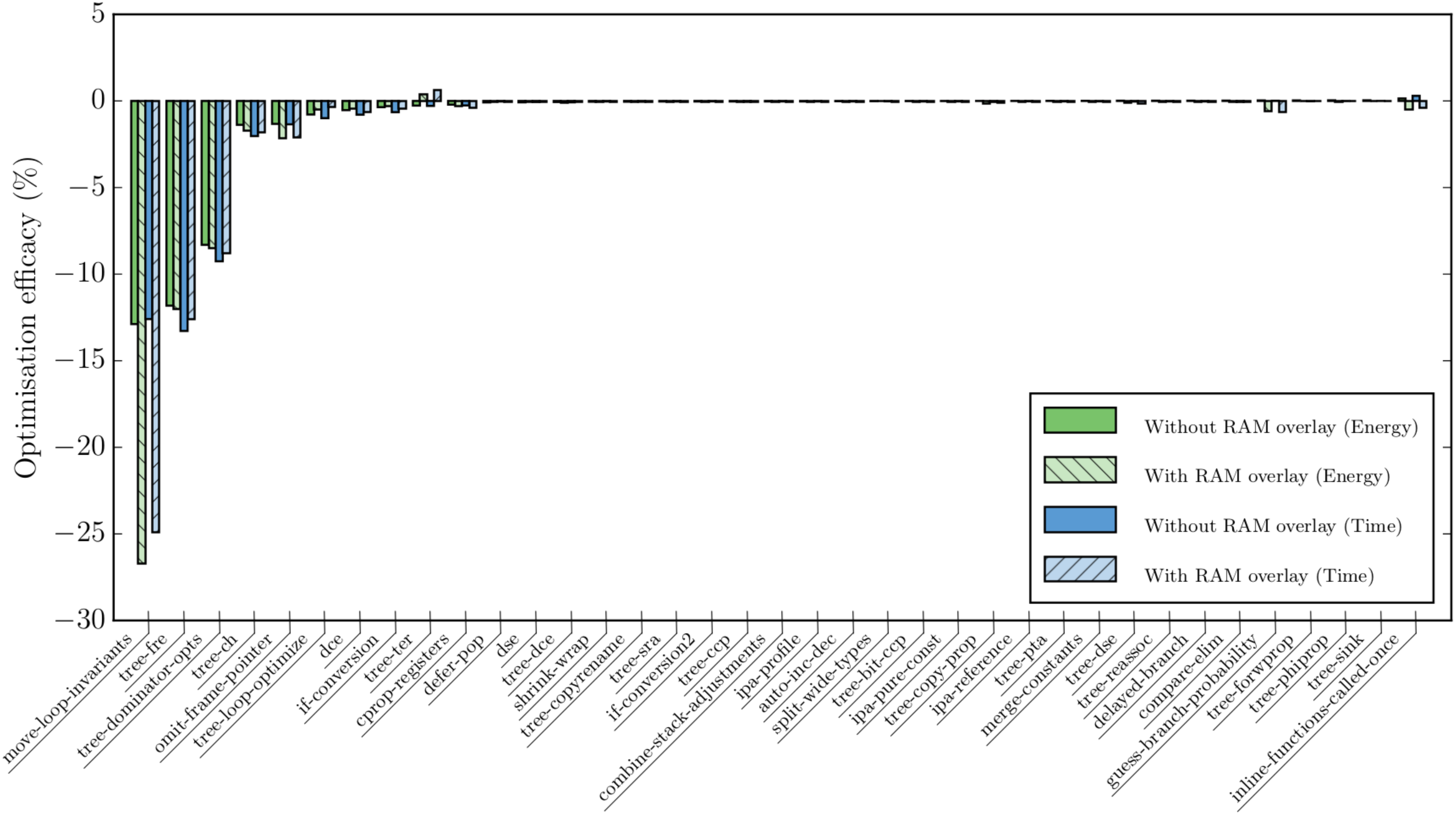
Combining optimisations



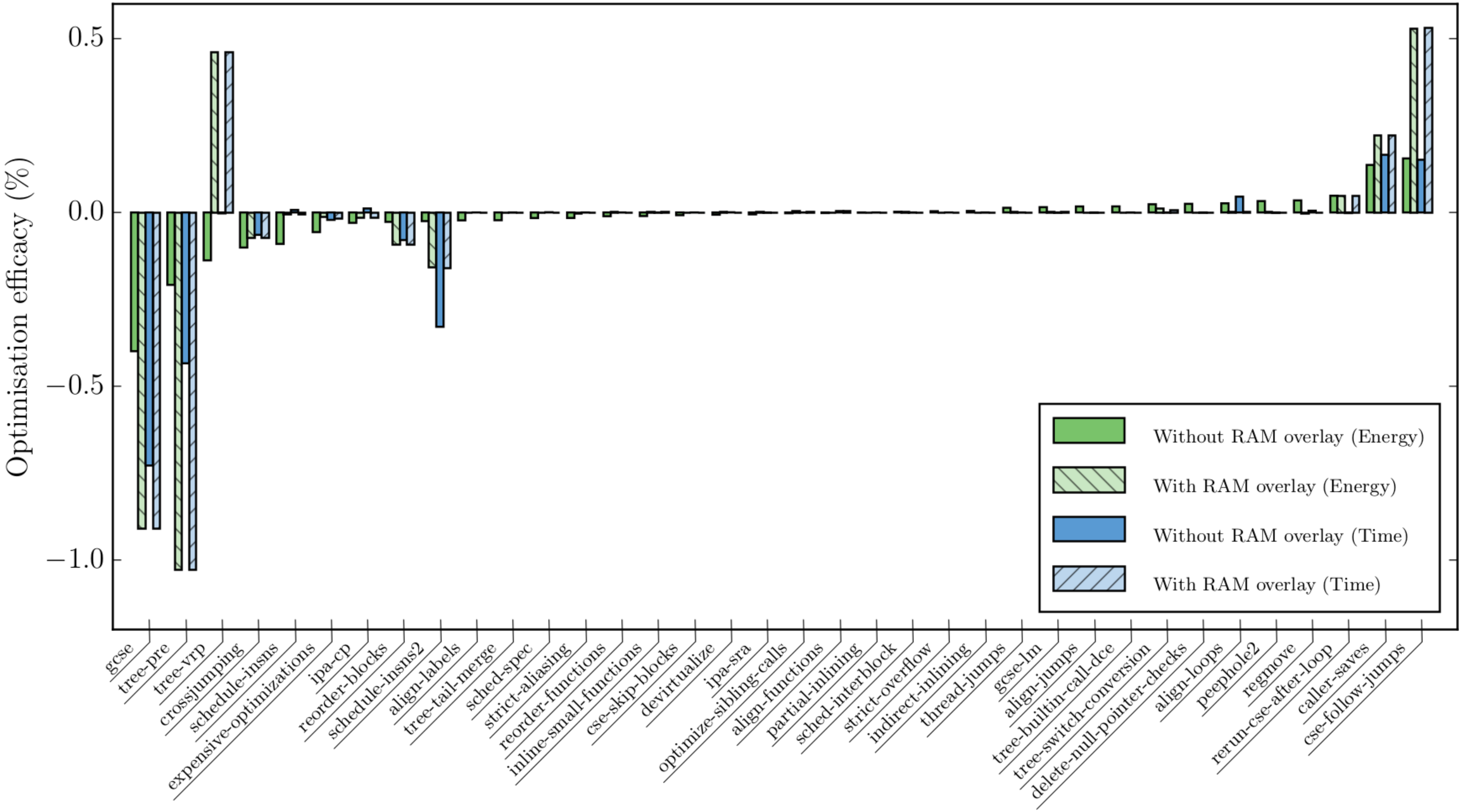
Combining optimisations



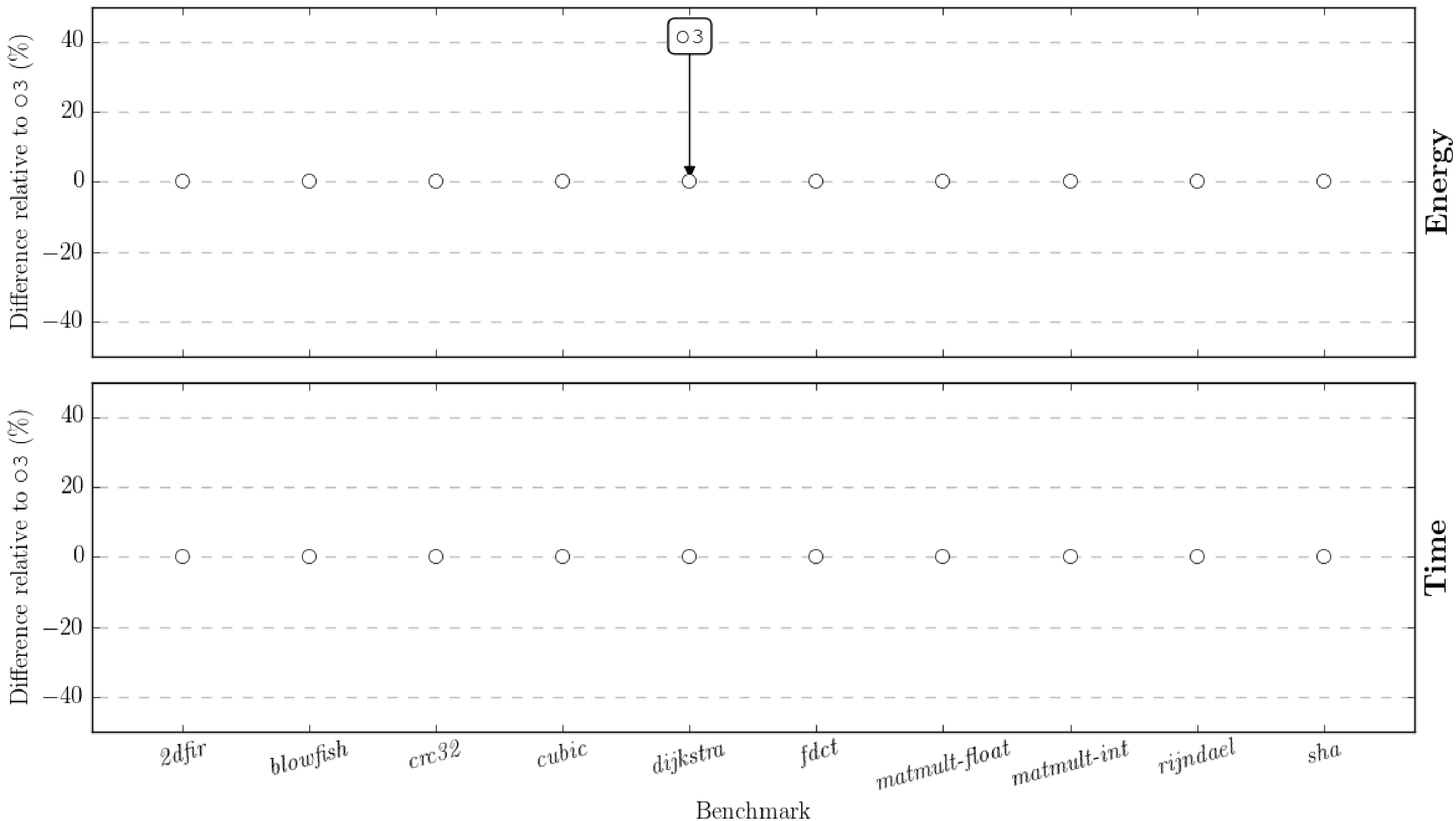
Fractional factorial design again



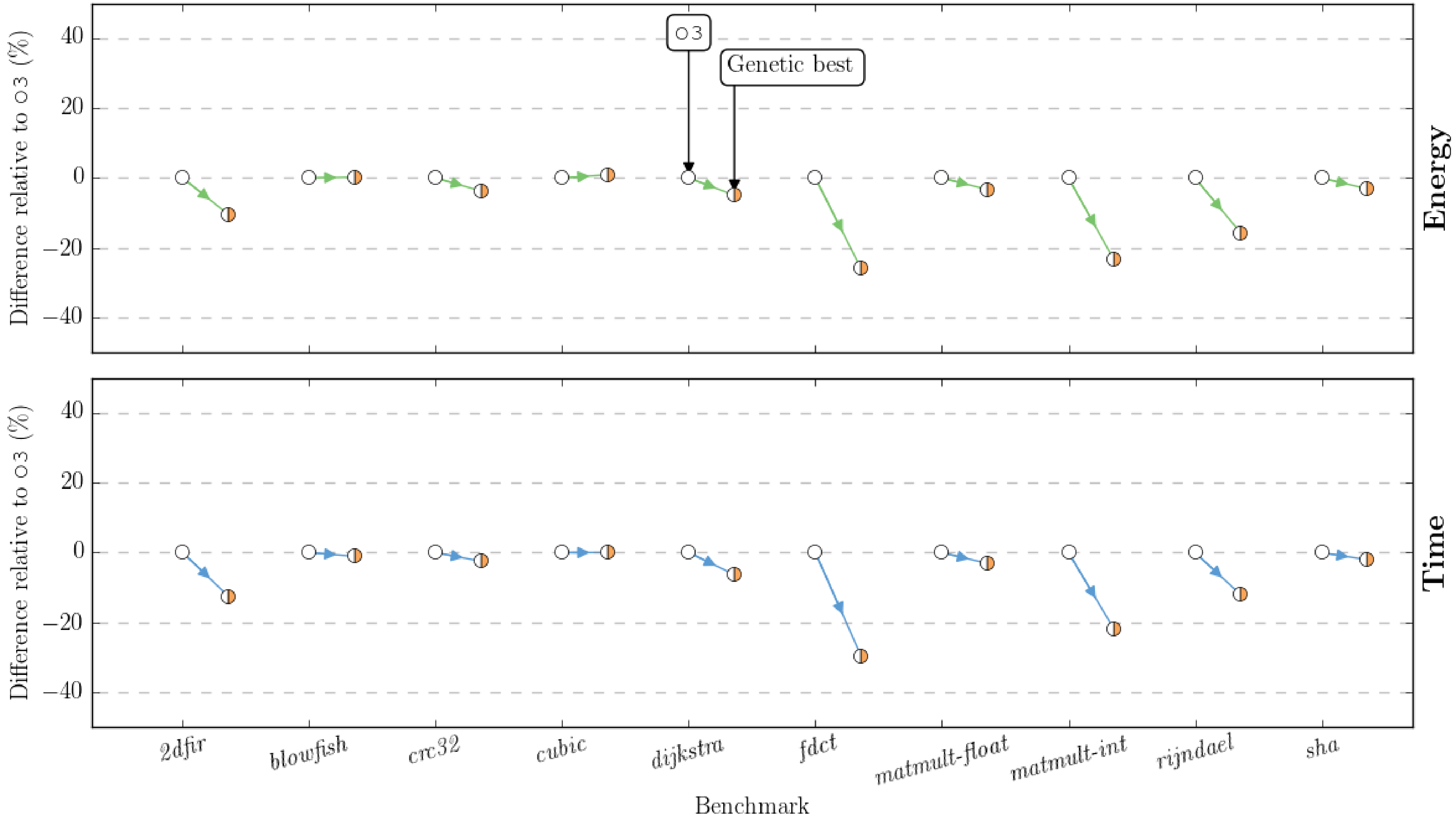
Fractional factorial design again



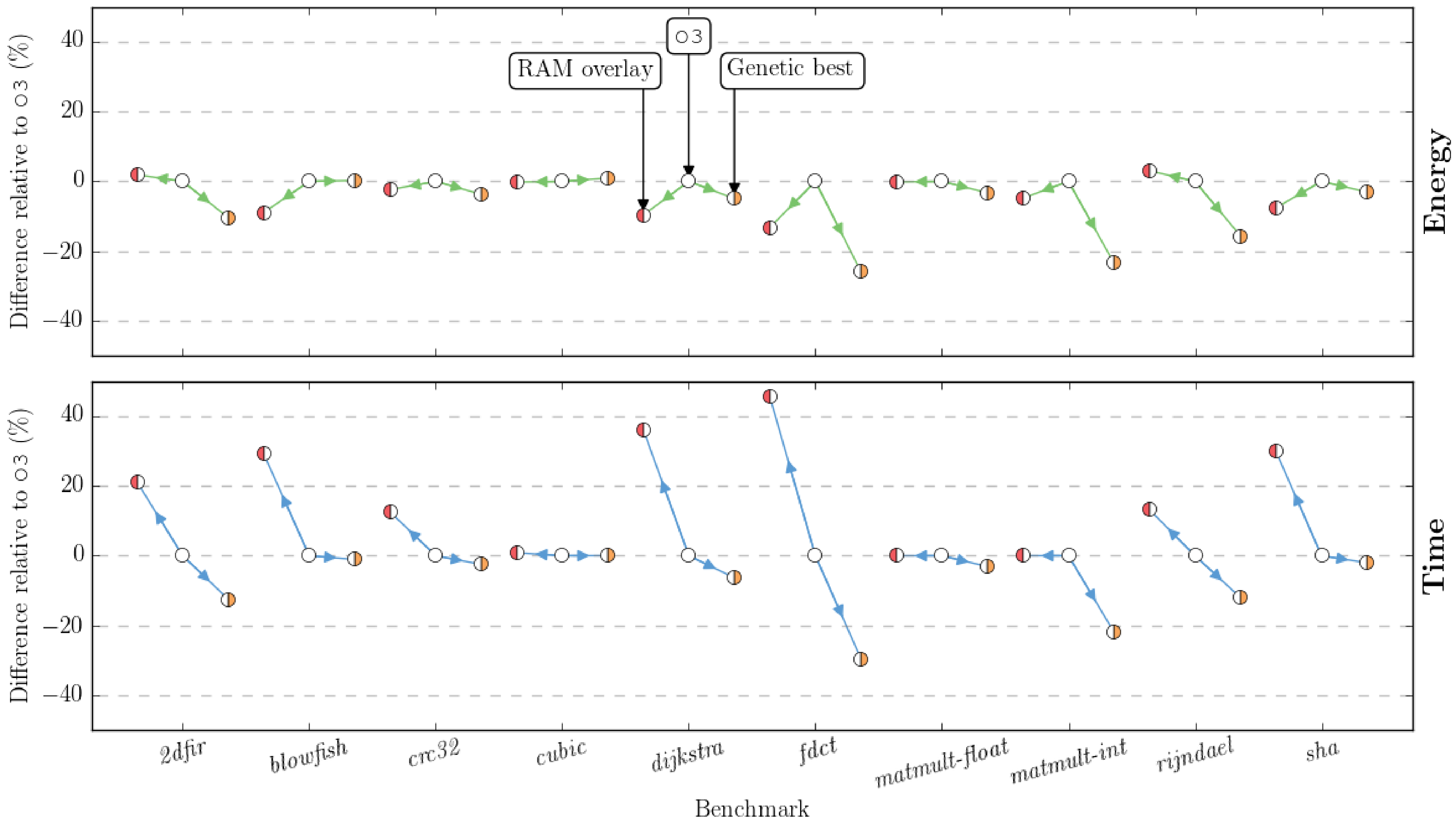
Composability



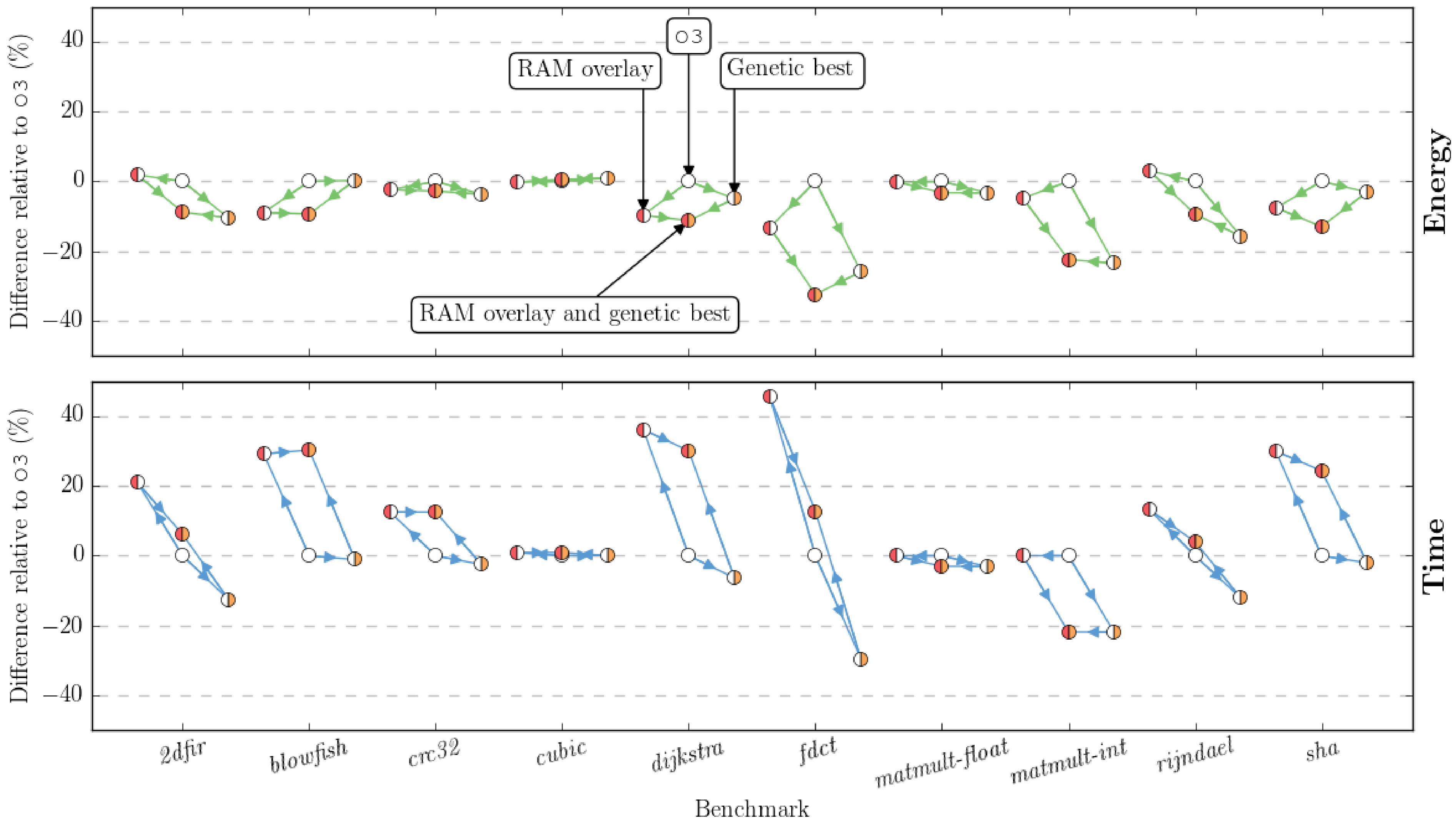
Composability



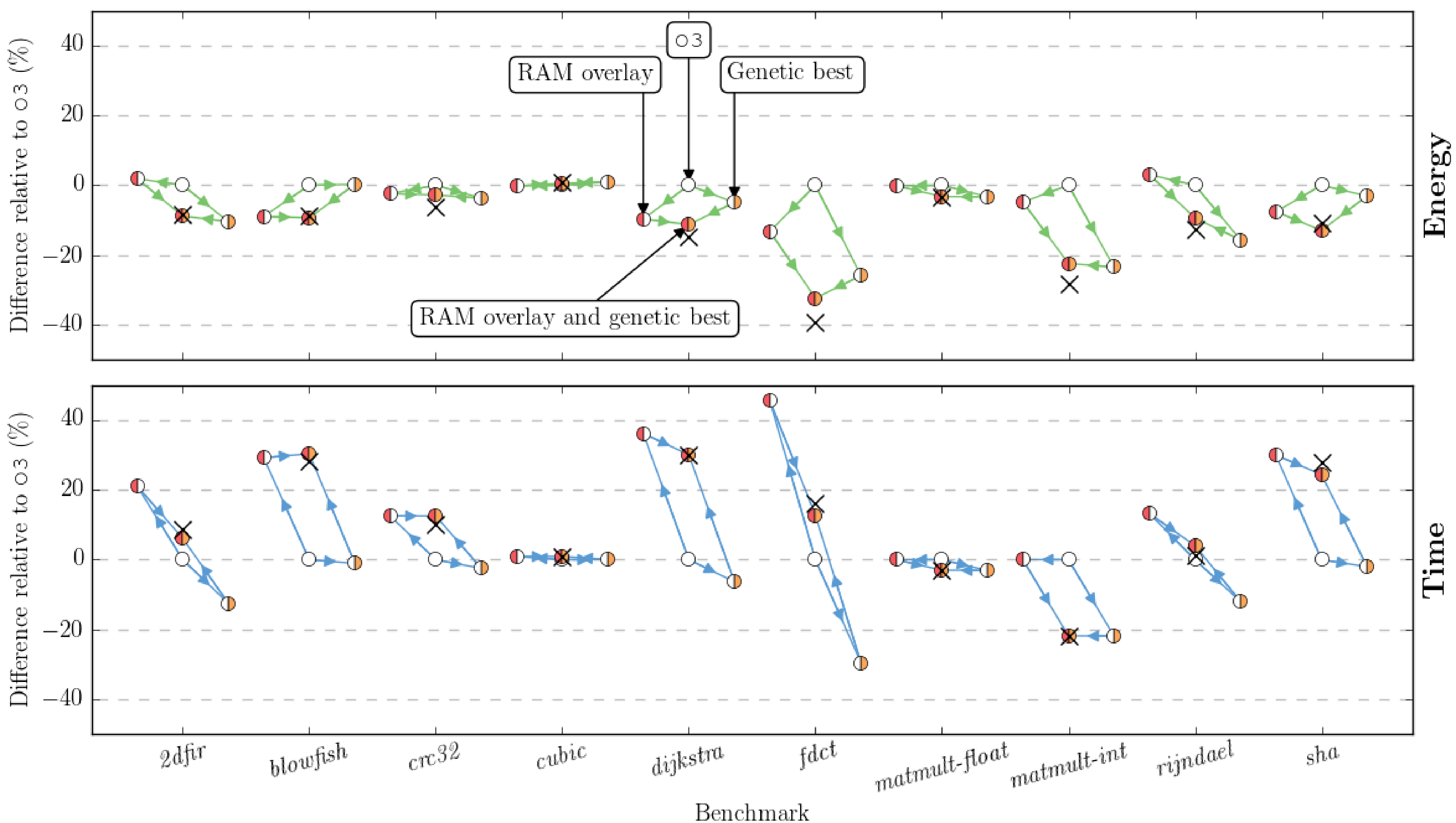
Composability



Composability



Composability



Combining optimisations

- The optimisations do not significantly interact.

- The same set of "base" optimisations can be applied
  - i.e. we do not need a new optimisation level to apply energy on top of.

Conclusion

Conclusion

Existing optimisations affect execution time by design.

Conclusion

Existing optimisations affect execution time by design.

Existing optimisations affect power by chance.

Conclusion

Existing optimisations affect execution time by design.

Existing optimisations affect power by chance.

Optimisations for energy *do* exist, they reduce energy by reducing average power.

Conclusion

Existing optimisations affect execution time by design.

Existing optimisations affect power by chance.

Optimisations for energy *do* exist, they reduce energy by reducing average power.

These optimisations combine linearly with existing optimisations.

Future work

Future work

How to find new energy optimisations?

Future work

How to find new energy optimisations?

Do all energy optimisations combine linearly?

Future work

How to find new energy optimisations?

Do all energy optimisations combine linearly?

Do the findings apply outside of embedded systems?

Future work

How to find new energy optimisations?

Do all energy optimisations combine linearly?

Do the findings apply outside of embedded systems?

Does the efficacy of the optimisations change when adding more cores/threads?

Thanks!

Questions?

More info:

James Pallister, Simon J. Hollis and Jeremy Bennett. “Identifying compiler options to minimize energy consumption for embedded platforms”. In: The Computer Journal. 2015.

James Pallister, Kerstin Eder, Simon J. Hollis and Jeremy Bennett. “A high-level model of embedded flash energy consumption”. In: *CASES’14 Proceedings of the 2014 international conference on Compilers, Architecture, and Synthesis for Embedded Systems.* New Delhi, India. ACM Press, 2014, p. 74.

James Pallister, Kerstin Eder and Simon J. Hollis. “Optimizing the flash-RAM energy trade-off in deeply embedded systems”. In: *CGO’15 Proceedings of the 2015 international symposium on Code Generation and Optimization.* San Francisco, USA. ACM Press, 2015.

Embedded systems

Embedded systems

Board Name	SoC Processor	RAM	ROM	Clock	Compiler GCC	 LLVM
STM32F0DISCOVERY	STM32F051 **Cortex-M0**	SRAM 8kB	Flash 64kB	8MHz	✓	✓
STM32VLDISCOVERY	STM32F100 **Cortex-M3**	SRAM 8kB	Flash 128kB	8MHz	✓	✓
Breadboard	ATMEGA328P **AVR8**	SRAM 2kB	Flash 32kB	8MHz	✓	✗
XMEGA-A3BU Xplained	ATXMEGA256A3BU **AVR8**	SRAM 16kB	Flash 256kB	8MHz	✓	✗
Breadboard	PIC32MX250F128B **MIPS32**	SRAM 32kB	Flash 128kB	8MHz	✓	✗
MSP-EXP430F5229LP	MSP430F5529 **MSP430**	SRAM 10kB	Flash 128kB	8MHz	✓	✗
MSP-EXP430FR5739LP	MSP430FR5739 **MSP430**	SRAM 1kB	FRAM 16kB	8MHz	✓	✗
Beagle Bone	AM335x **Cortex-A8**	DRAM 256MB	SD-Card	800MHz	✓	✓
EMEK4 Dev Board	EMEK4 **Epiphany**	SRAM 32kB†	None	800MHz	✓	✗
XK-1A	XS1-L8-64 **XMOS-XS1**	SRAM 64k	None‡	400MHz	✗	✓

Individual optimisations

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

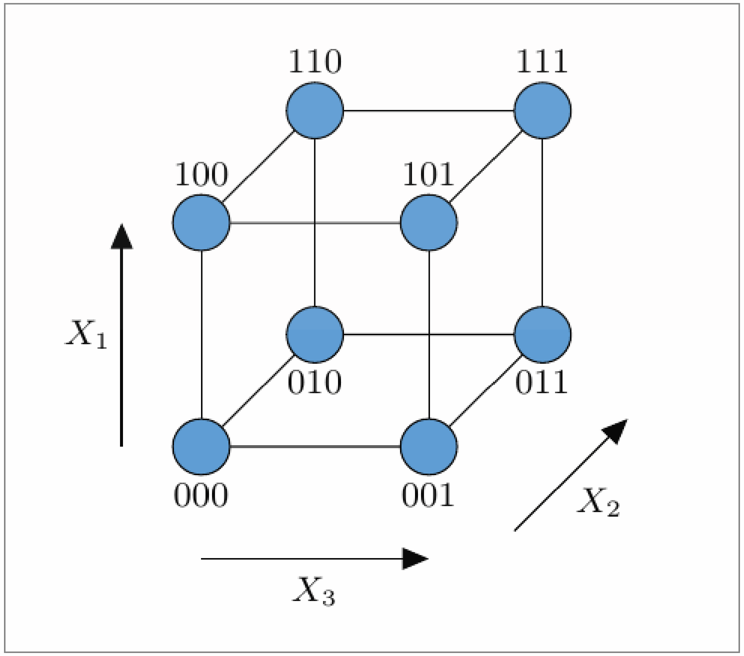
Individual optimisations

Fractional factorial design

"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

Individual optimisations

Fractional factorial design

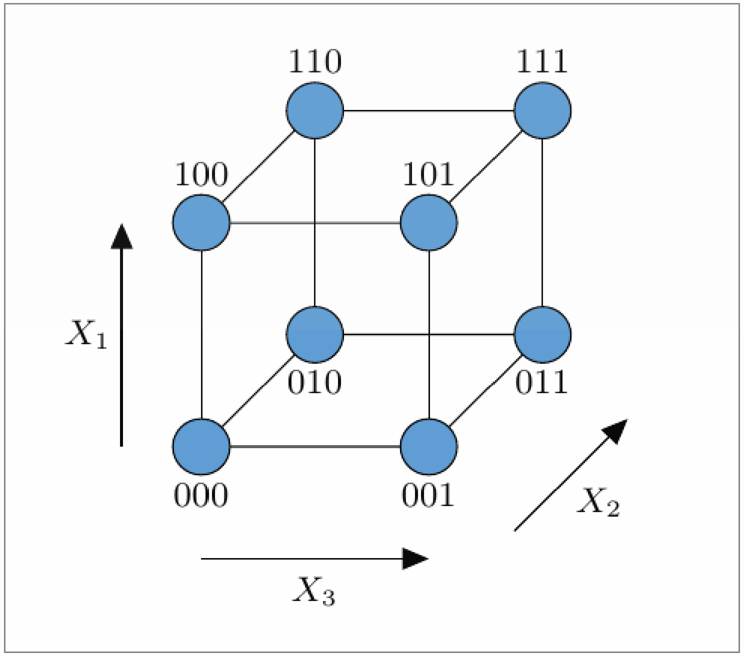


Full factorial design

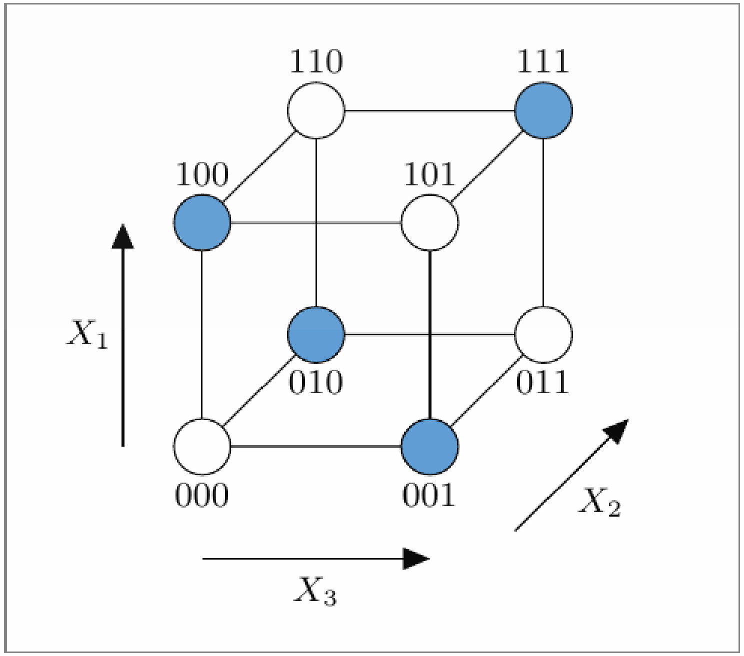
"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

Individual optimisations

Fractional factorial design



Full factorial design



Fractional factorial design

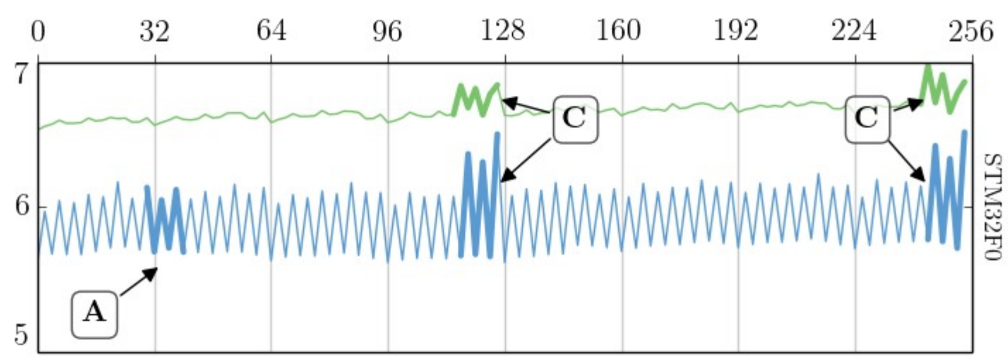
"Do existing compiler optimisations save energy purely by reducing the k_T *coefficient?"*

Parameters

SoC	Model parameters (pJ)						
	E_2 (**A**)	E_3	E_4 (**B**)	E_5	E_6	E_7 (**C**)	E_8 (**C**)
STM32F0	300	27	6	0	9	100	6
STM32F1	500	0	6	34	4	10	190
ATMEGA328P	0	22	36	27	9	107	24
PIC32MX250F128B	225	0	10	18	8	13	113
MSP430F5529	408	0	34	26	15	13	13

Parameters

SoC	Model parameters (pJ)						
	E_2 (**A**)	E_3	E_4 (**B**)	E_5	E_6	E_7 (**C**)	E_8 (**C**)
STM32F0	300	27	6	0	9	100	6
STM32F1	500	0	6	34	4	10	190
ATMEGA328P	0	22	36	27	9	107	24
PIC32MX250F128B	225	0	10	18	8	13	113
MSP430F5529	408	0	34	26	15	13	13



Parameters

SoC	Model parameters (pJ)						
	E_2 (**A**)	E_3	E_4 (**B**)	E_5	E_6	E_7 (**C**)	E_8 (**C**)
STM32F0	300	27	6	0	9	100	6
STM32F1	500	0	6	34	4	10	190
ATMEGA328P	0	22	36	27	9	107	24
PIC32MX250F128B	225	0	10	18	8	13	113
MSP430F5529	408	0	34	26	15	13	13