

Optimal Program Partitioning for Predictable Performance

Jack Whitham and Neil Audsley
Real-time Systems Group
University of York



Program Partitioning

- Programs are typically larger than local memory
 - L1 cache ~ 16kb
 - L1 scratchpad memory (SPM) ~ 16kb
 - Typical program size?
- How can a large program make good use of local memory?

Program Partitioning

- Program is divided into *regions*
 - Regions consist of methods, basic blocks, loops...
- One region is in local memory at a time
 - Regions are small enough to fit in local memory
- If execution leaves one region, another region is loaded

Program Partitioning

- Cache: implicit partitioning
 - Program elements loaded on demand
- SPM: explicit partitioning
 - Algorithm required to divide large programs into regions
- Extra complexity... why use SPM?
 - Predictability
 - Performance

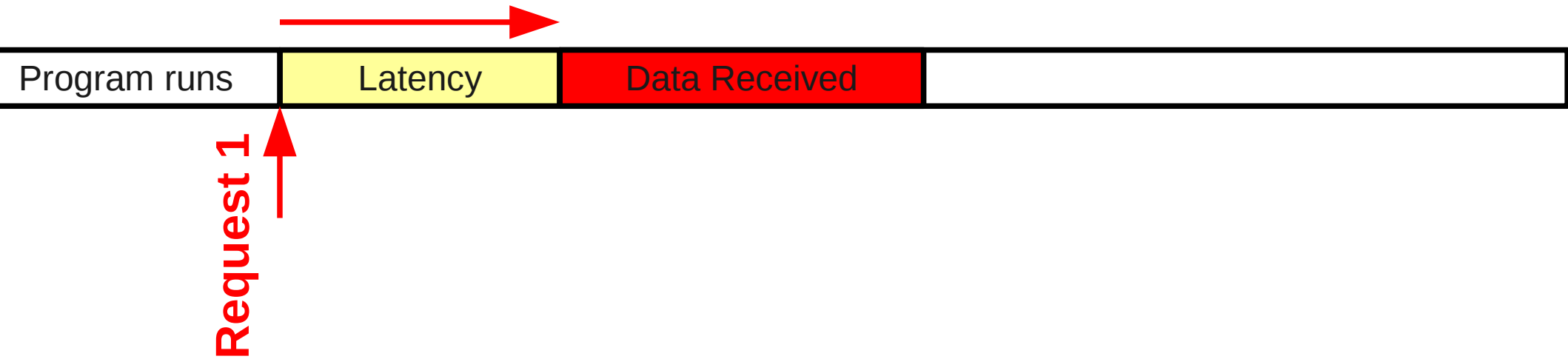
fdct

- MRTTC benchmark
- 223 words (Microblaze, mb-gcc -Os)
- 56 cache misses *OR* one SPM load

fdct

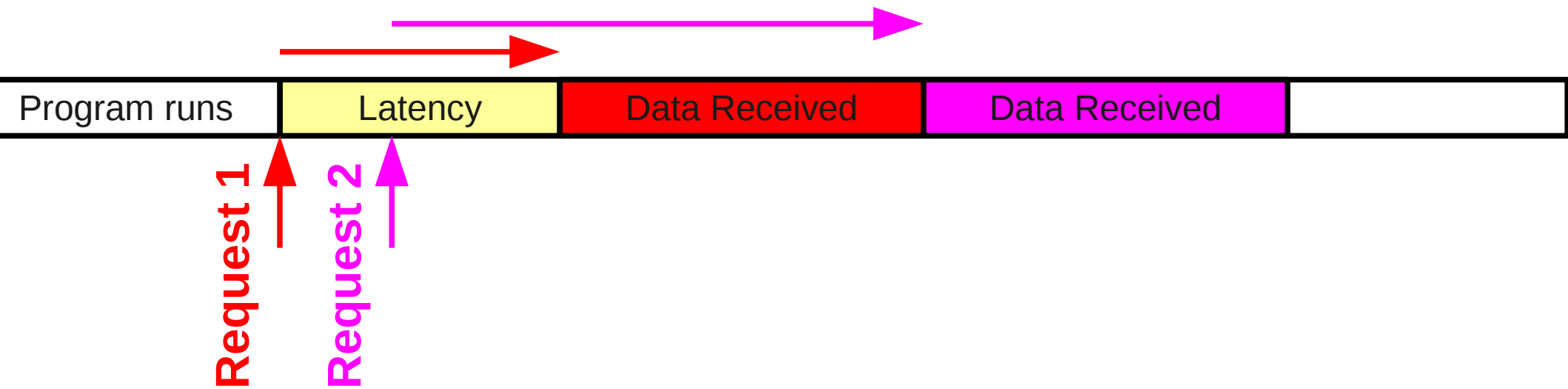
- 56 cache misses @ 29 clock cycles each
(on my FPGA)
= 1624 clock cycles waiting
OR
- One SPM load of 223 words
= 286 clock cycles waiting

One SPM load



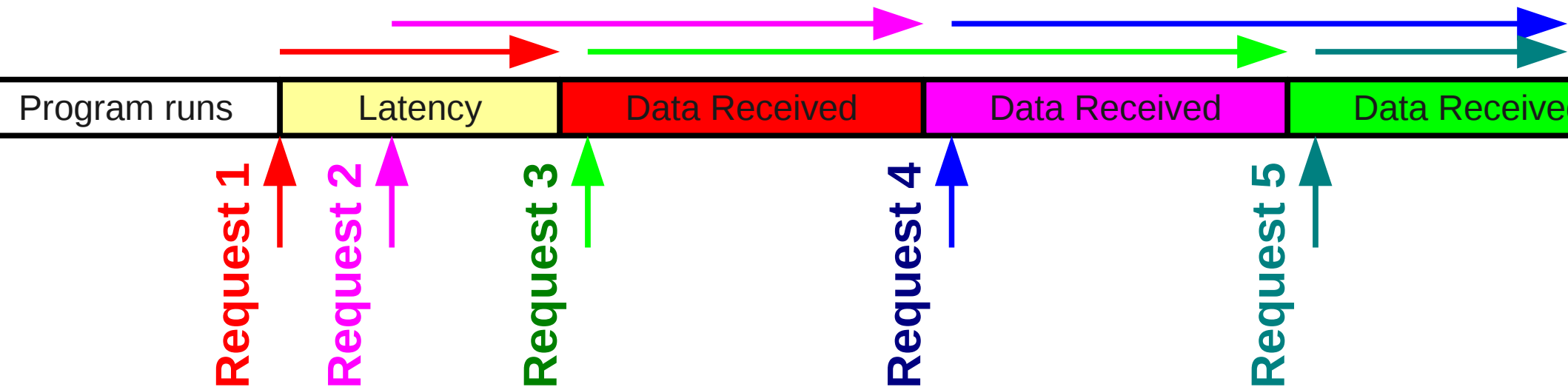
One SPM load

- SPM load is pipelined



One SPM load

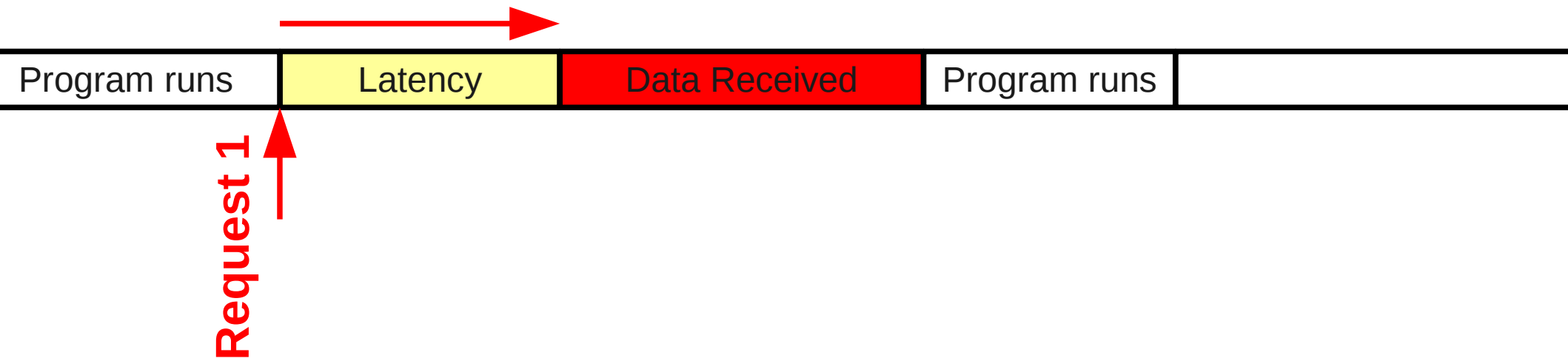
- SPM load is pipelined



- After initial latency, the bus is never idle

56 cache misses

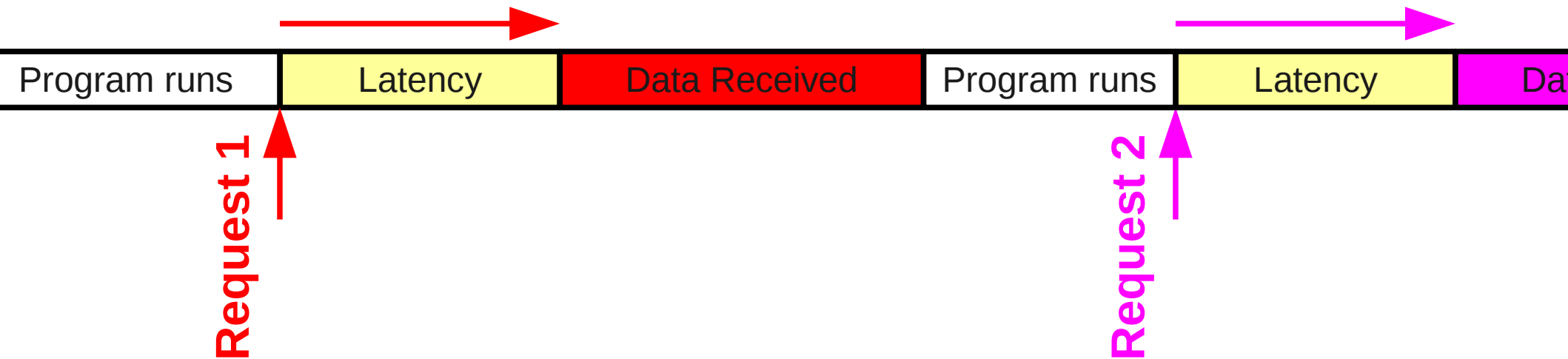
- Cache load is *not* pipelined



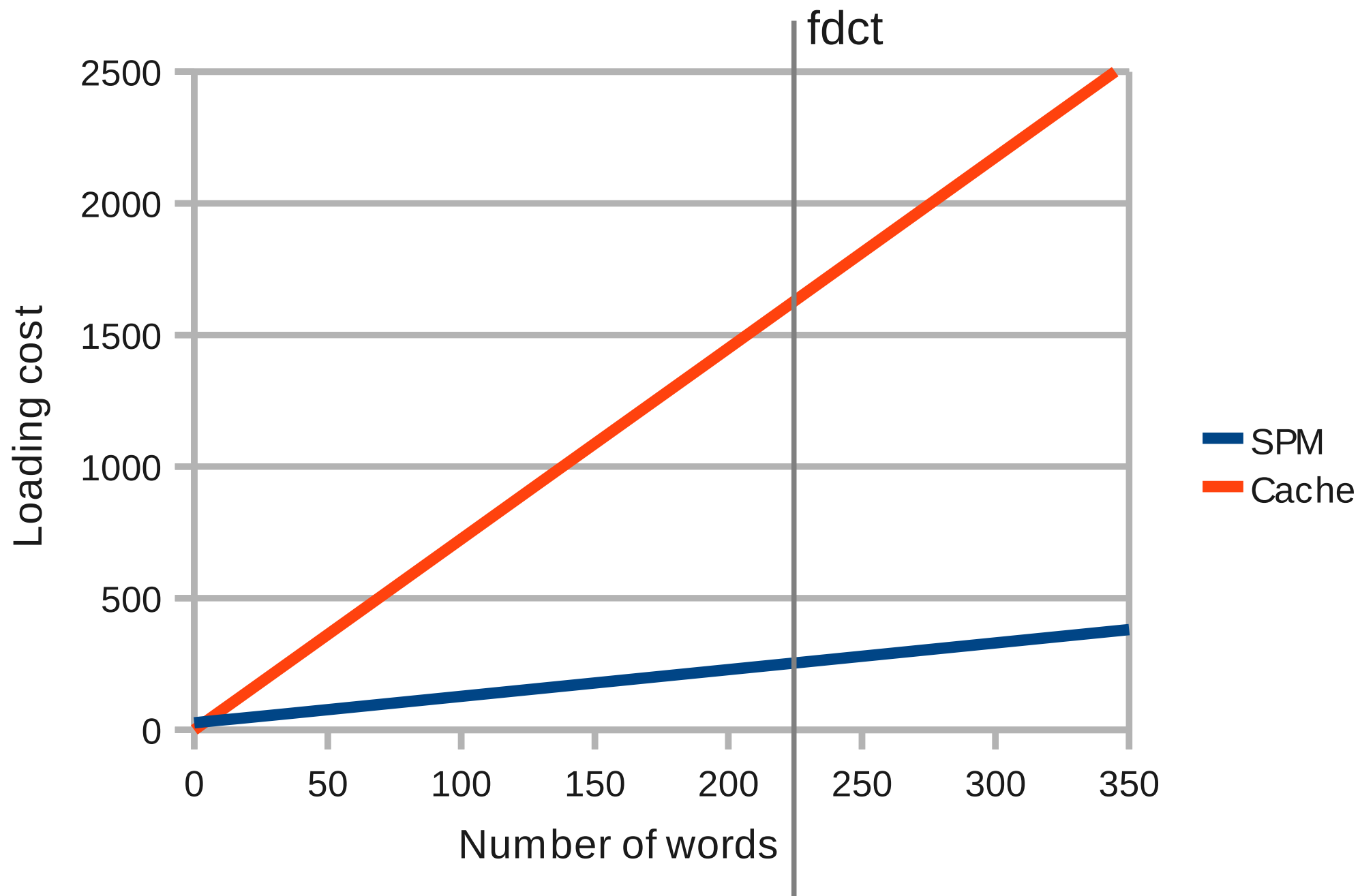
- Cache misses depend on the program control flow

56 cache misses

- Cache load is *not* pipelined



- Cache misses depend on the program control flow



fdct

- Measured WCET on FPGA platform
 - 4213 clock cycles with 256 word cache
 - 2903 clock cycles with 256 word SPM
- 45% faster (real hardware)
- But this is a small program
- Larger programs would require...

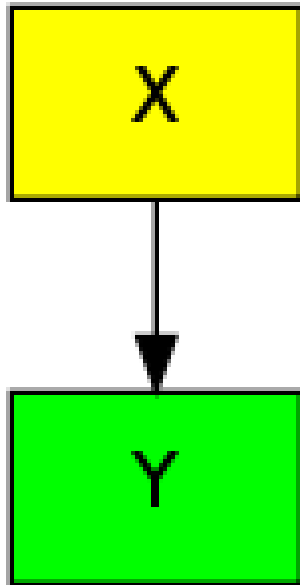
fdct

- Measured WCET on FPGA platform
 - 4213 clock cycles with 256 word cache
 - 2903 clock cycles with 256 word SPM
- 45% faster (real hardware)
- But this is a small program
- Larger programs would require...
 - *Partitioning!*

Partitioning a Call Tree

Call Tree Notation

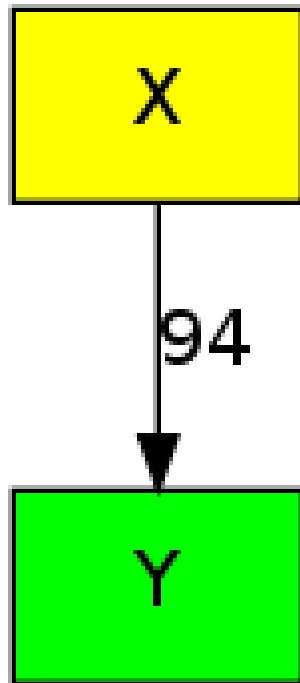
- Method X calls method Y



```
void X(void)
{
    ...
    Y();
    ...
}
```


Call Tree Notation

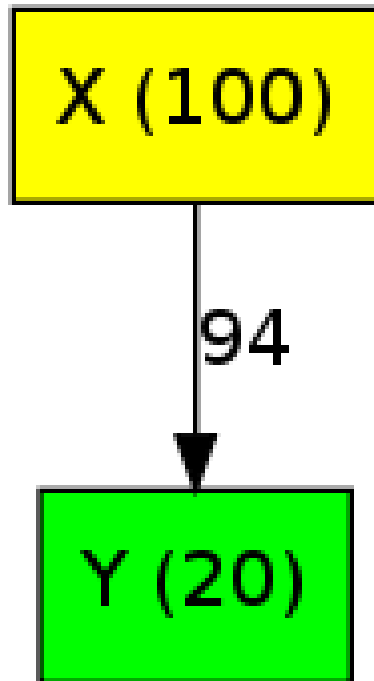
- Method X calls method Y 94 times



```
void X(void)
{
    ...
    for(i=0;i<94;i++)
        Y();
    ...
}
```

Call Tree Notation

- Method sizes

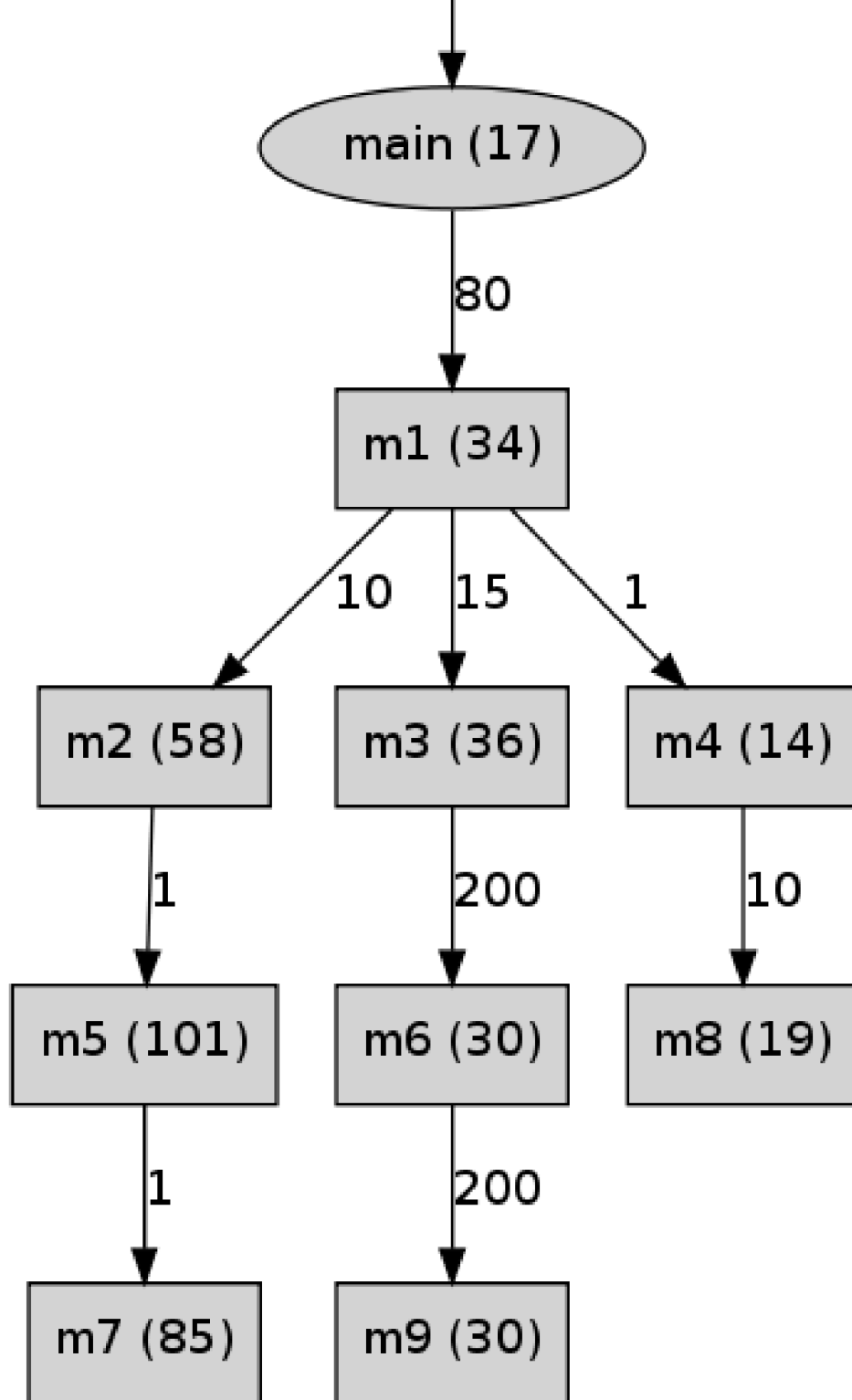


- Method X has size 100 words
- Method Y has size 20 words

Partitioning Example

- Program containing 10 methods
- Total method size 424 words
- SPM size 128 words

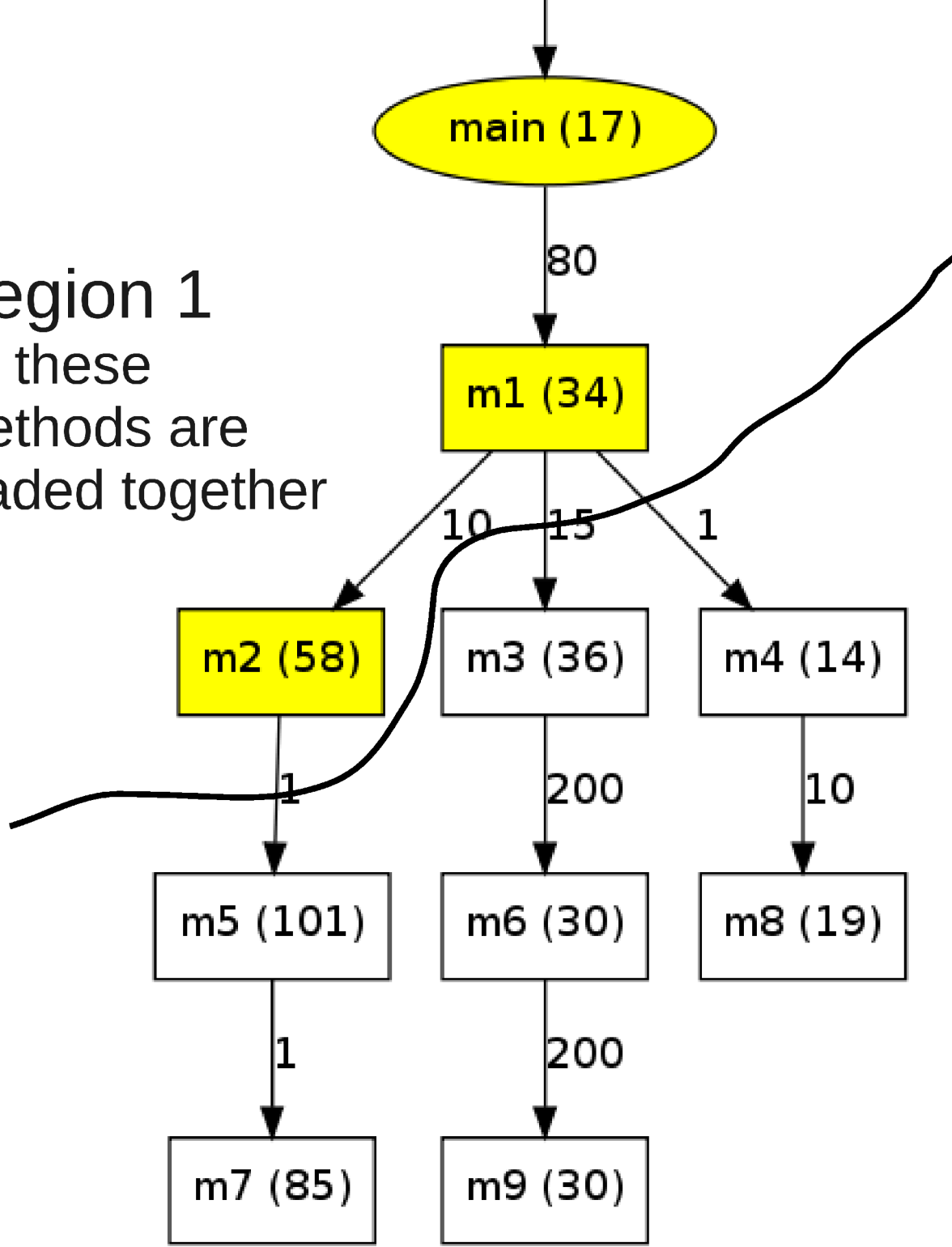
- Minimise the cost of region transitions
- Enforce upper bound on region size



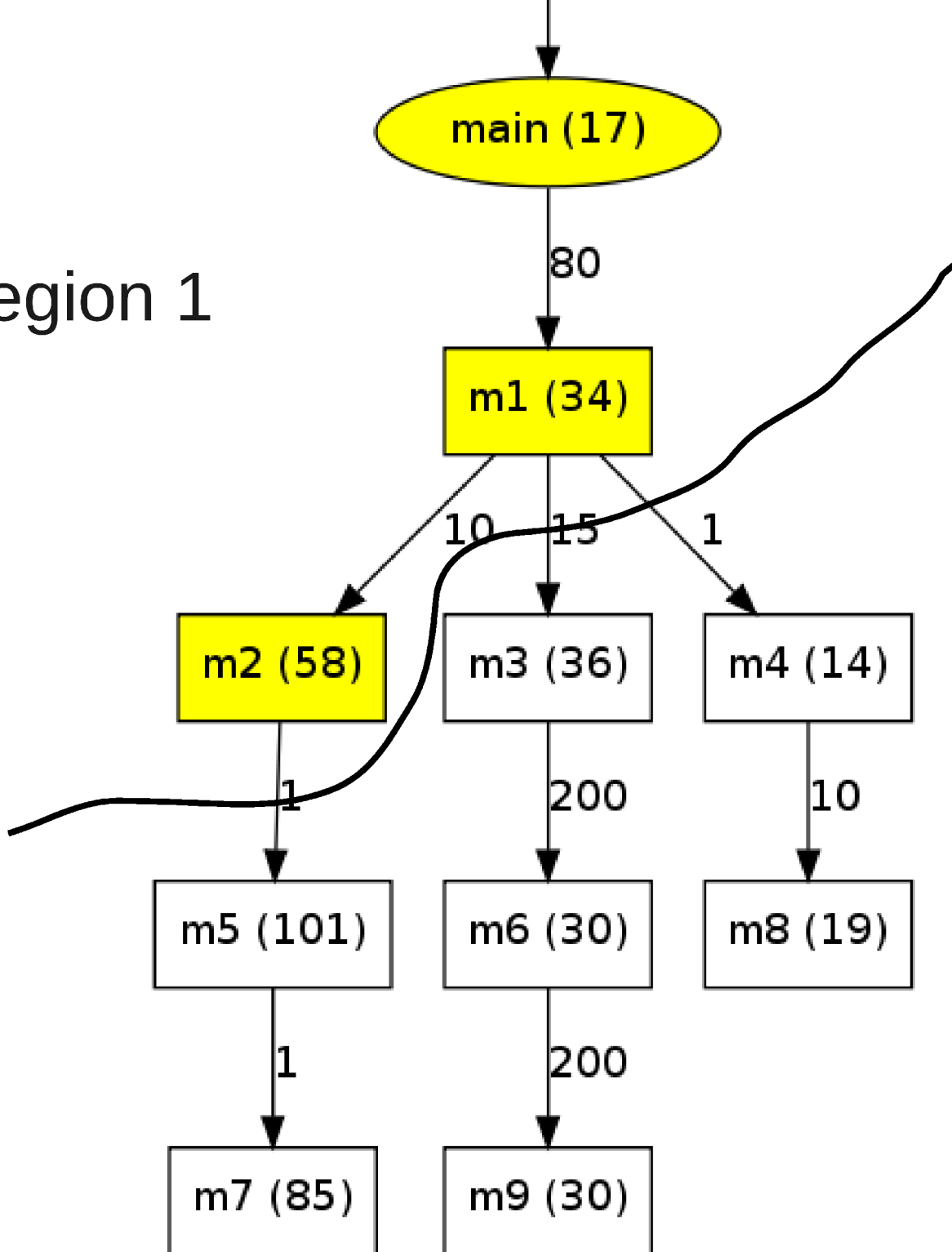
- 10 methods
- Total size 424 words
- SPM size 128 words

Region 1
All these
methods are
loaded together

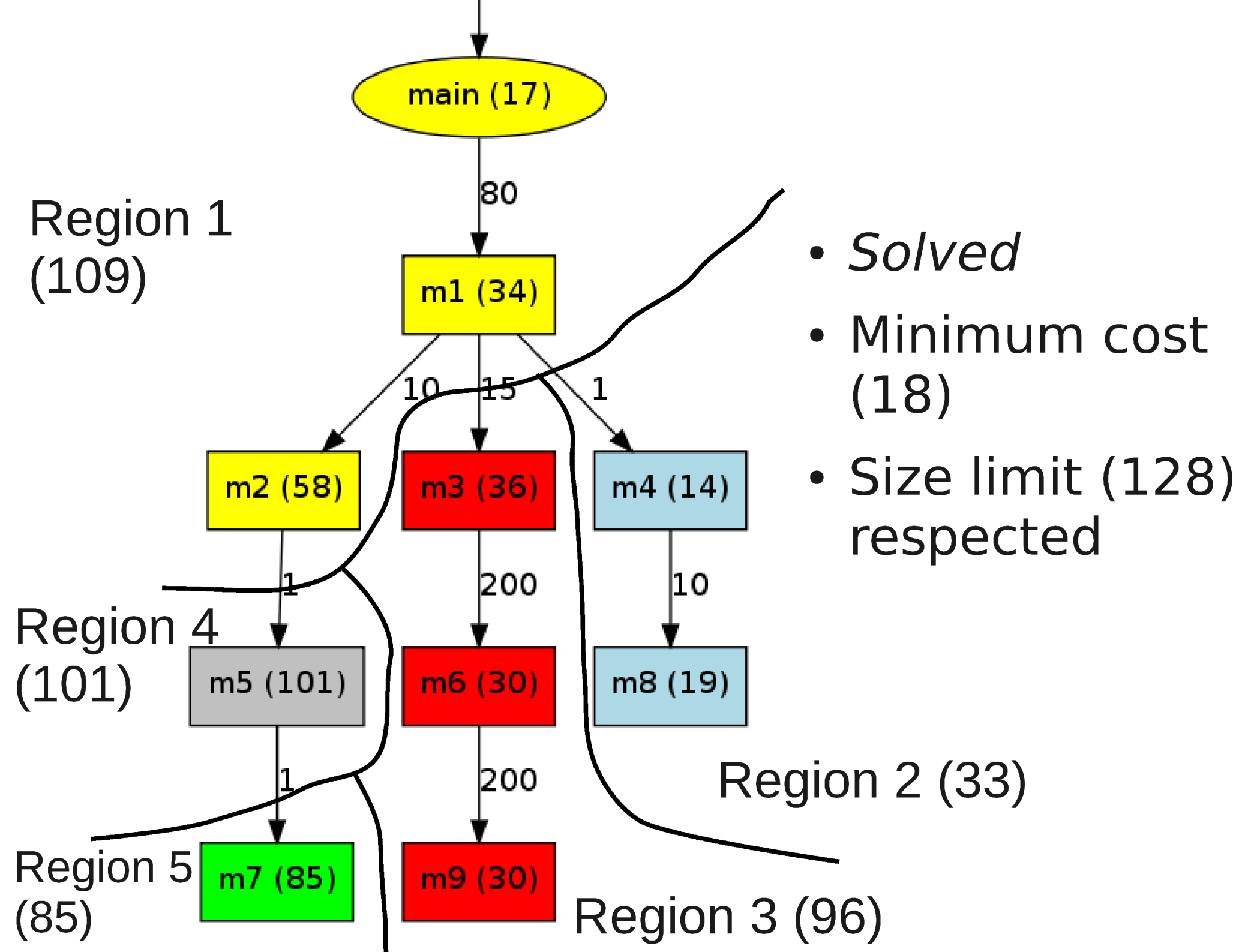
- $17 + 34 + 58 = 109$ words



Region 1



- 17 transitions
- Why not include m4? (123)



Partitioning Algorithms

- Exhaustive search
- Greedy (min-cut)
- Greedy (merging regions)
- Dynamic programming

Dynamic Programming

- Program represented as a tree
 - Typically a call tree
- Partitions created from leaves to root
- Optimal partition in polynomial time!

Dynamic Programming

- Program represented as a tree
 - Typically a call tree
- Partitions created from leaves to root
- Optimal partition in polynomial time!
 - Optimal wrt. program representation and a single “typical” execution path (not necessarily worst-case path)

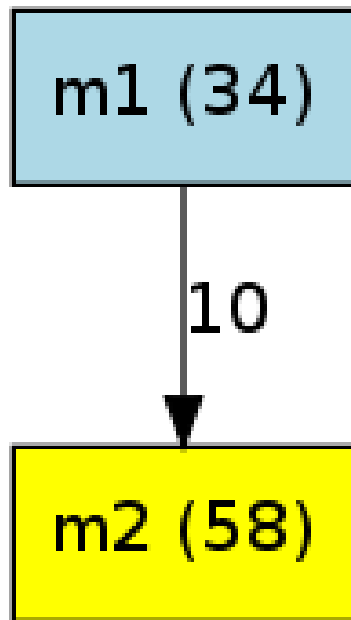
Lukes' Algorithm

- J.A. Lukes (1974) invented an $O(nk^2)$ algorithm for partitioning call trees
- For each subtree root *and* each possible root region size, *memoise* the optimal partition

Contributions of the paper

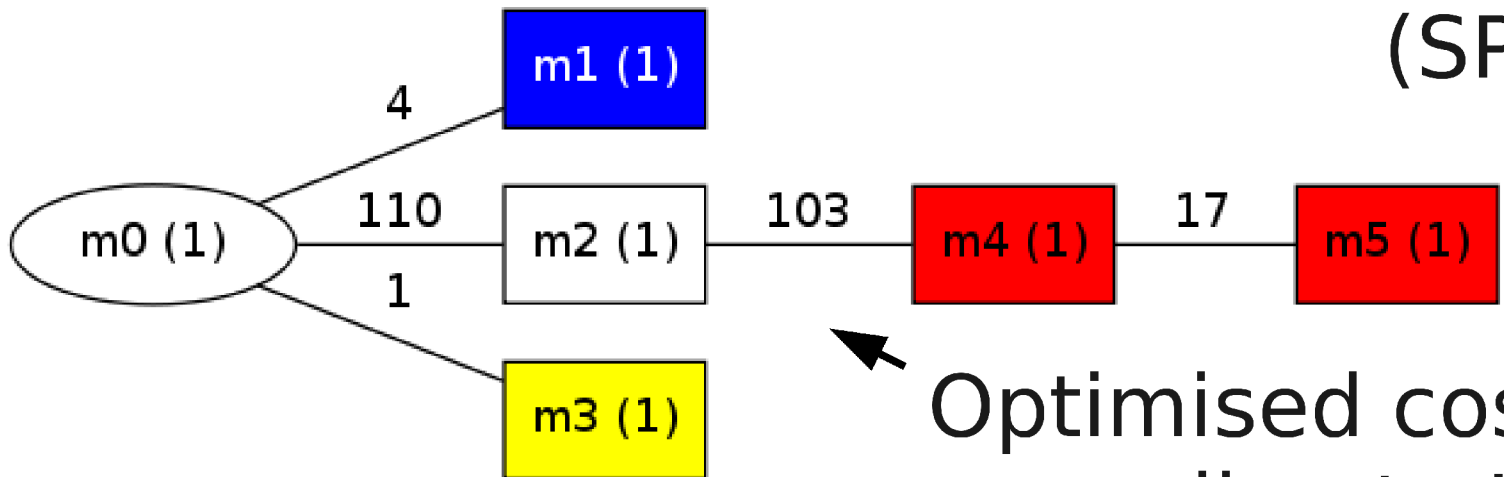
Result 1

- Lukes' algorithm does not generate optimal solutions when the cost of loading regions is taken into account



Lukes' cost:
10

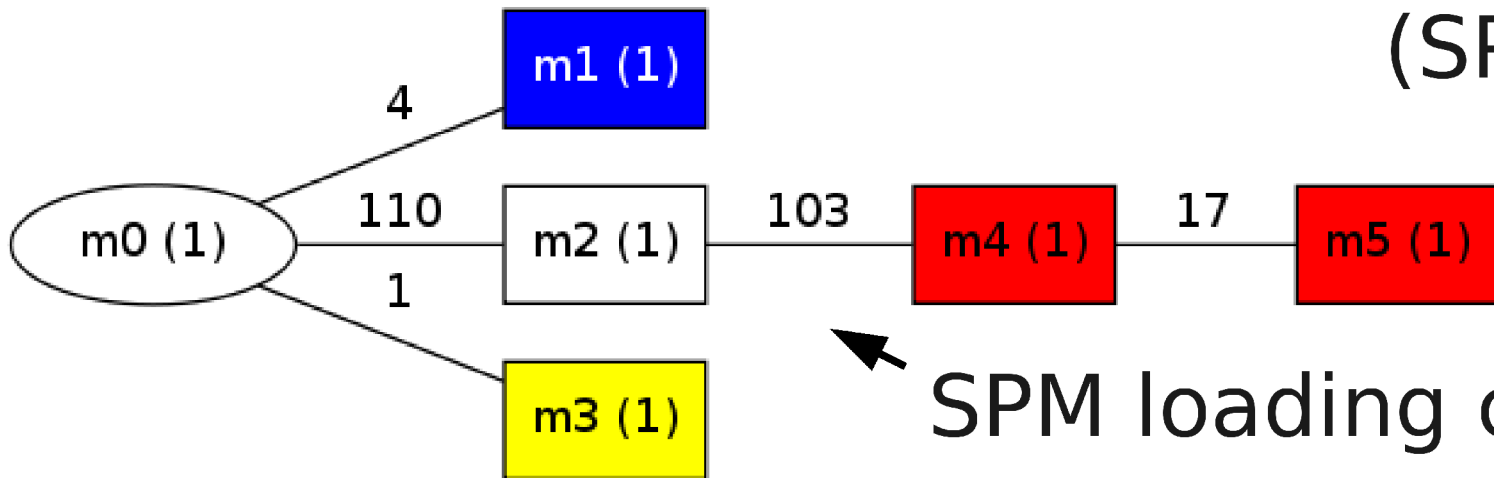
SPM loading cost:
 $10 \times \text{loading } 58 \text{ words} +$
 $10 \times \text{loading } 34 \text{ words}$



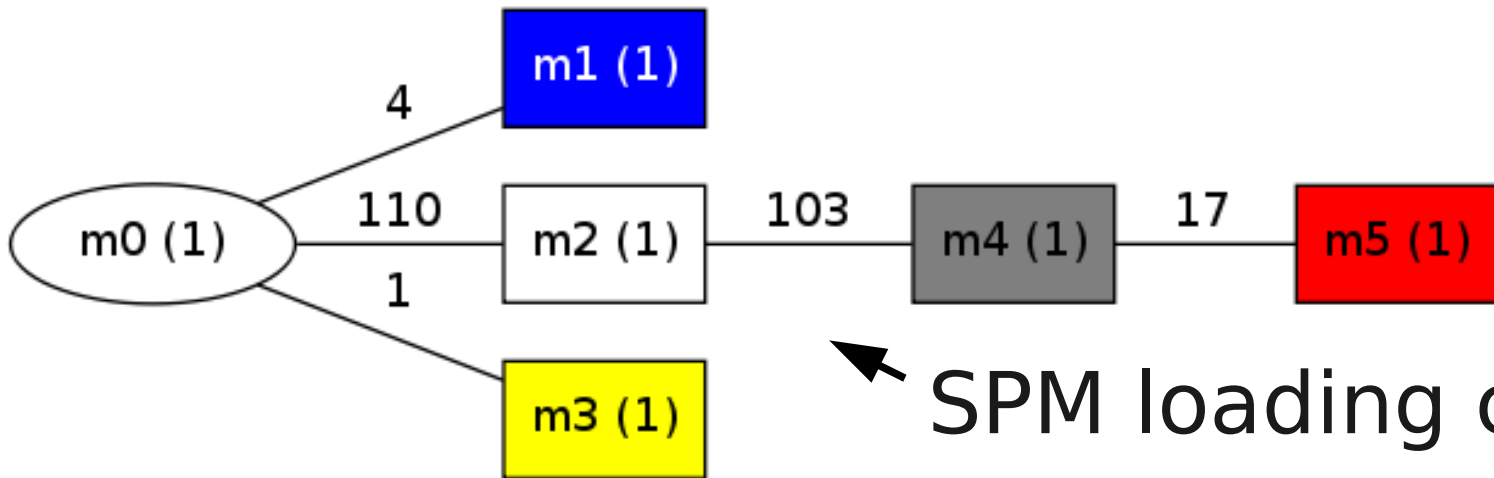
(SPM size 2)

Optimised cost
according to Lukes: 108
SPM loading cost: 429

(SPM size 2)



SPM loading cost: 429



SPM loading cost: 360

Algorithm 1

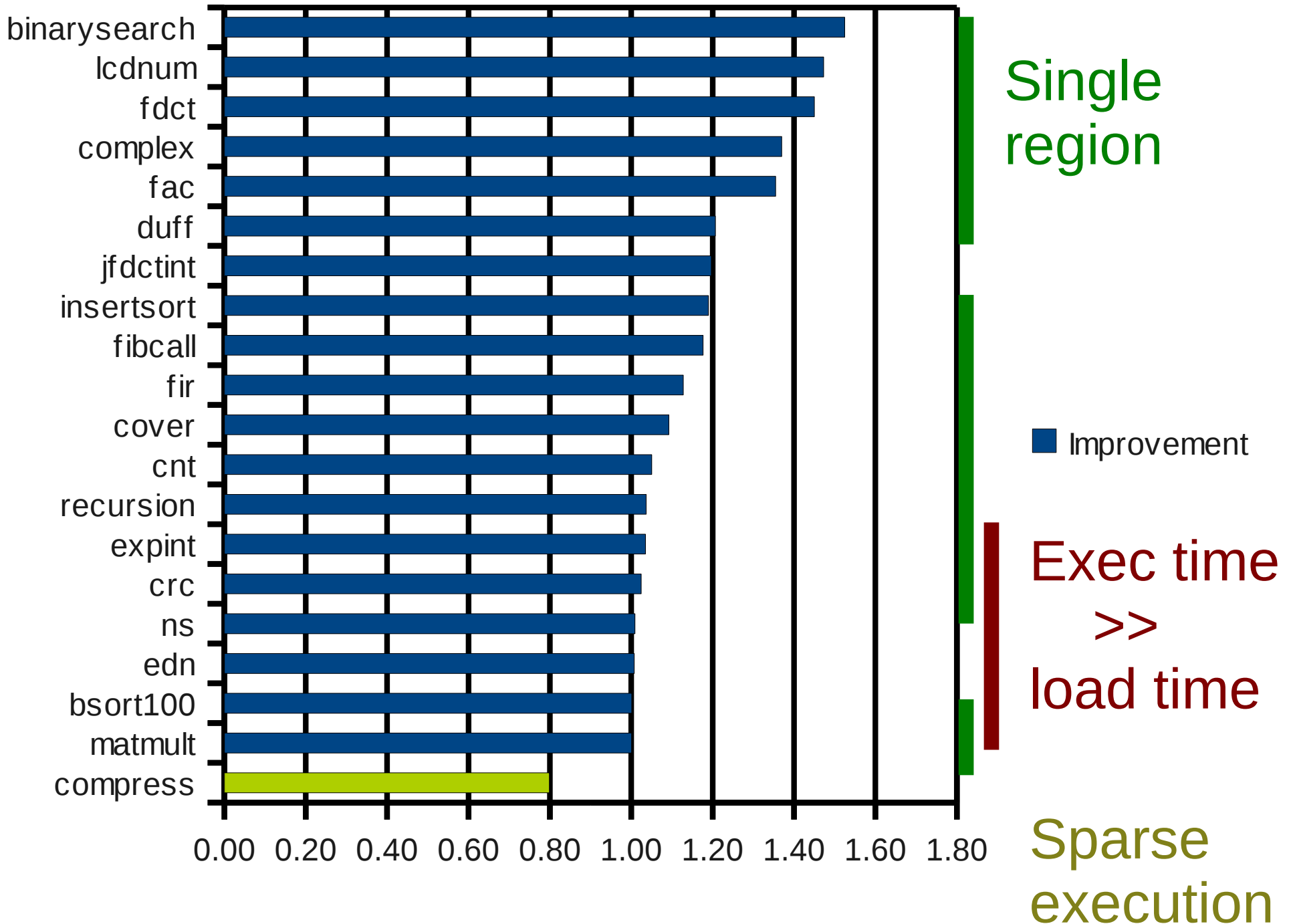
- New partitioning algorithm ELA-1 which includes region sizes in cost calculations
- Principal difficulty – cost calculations depend on the caller as well as callee
- Caller region size is unknown

ELA-1

- Unknown caller region size represented by α
- For each subtree root *and* each possible root region size *and* each possible α , store optimal partition
- Lukes: $O(nk^2)$
ELA-1: $O(nk^3)$
(up to k possible values of α)

Result 2

- Comparison of ELA-1 and cache
- Recall: fdct program
 - Single region
 - 4213 clock cycles with 256 word cache
 - 2903 clock cycles with 256 word SPM
 - 45% faster ($4213/2903 = 1.45$)
- Repeated experiment with other MRTTC programs

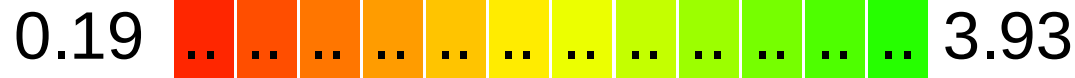


Improved evaluation

- Smaller SPM: increase pressure
 - Tried exlining loops
- Separate loading time and execution time
 - Clearer results for long-running benchmarks

ELA-1 vs Cache

	64	128	256	512	1024	2048	4096
binarysearch	-	-	2.12	2.12	2.12	2.12	2.12
bsort100	-	0.76	1.50	2.14	2.14	2.14	2.14
crc	-	-	0.94	1.13	1.74	1.74	1.74
edn	-	-	-	-	0.40	1.12	1.45
fir	-	-	0.62	1.90	1.90	1.90	1.90
insertsort	-	-	0.19	2.00	2.00	2.00	2.00
jfdctint	-	-	-	1.44	1.50	1.62	1.62
matmult	-	1.16	2.40	3.93	1.78	1.78	1.78



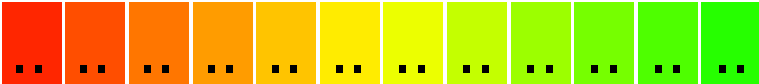
Loading times only
Loops exlined

Algorithm 2

- Problem: call tree representation
 - Sometimes, methods don't fit
 - Small SPM size
 - Whole methods are loaded even if parts are rarely/never used
 - c.f. “compress”
- Solution: ELA-2: an attempt to extend ELA-1 for general control-flow graphs

ELA-2 vs Cache

	64	128	256	512	1024	2048	4096
binarysearch	2.73	2.25	2.21	2.21	2.21	2.21	2.21
bsort100	2.55	0.94	2.09	2.22	2.22	2.22	2.22
crc	1.33	1.08	2.07	1.46	2.05	2.05	2.05
edn	1.17	1.32	1.19	1.82	2.11	1.95	1.69
fir	1.63	1.69	0.81	2.02	2.02	2.02	2.02
insertsort	2.03	1.59	1.95	2.08	2.08	2.08	2.08
jfdctint	1.50	1.48	1.33	1.94	1.91	1.81	1.81
matmult	1.94	1.47	2.75	5.79	1.85	1.85	1.85

0.19  3.93 Loading times only

ELA-2

- ELA-2 is not widely applicable
 - Loops are a problem, and poorly handled
 - $O(2^L nk^3)$ time for L loops (!)
- A better solution is required
 - Greedy heuristics may be the best-known solution so far

Conclusions

- Partitioning brings the performance and predictability benefits of SPM to larger programs
- Optimal algorithm ELA-1 specified
 - ELA-1 is very useful if a call tree can be partitioned effectively
 - Difficulties in generalising ELA-1 for control flow graphs (ELA-2)

Thankyou