

Investigating average versus worst-case timing behavior of data caches and data scratchpads

Jack Whitham and Neil Audsley
Real-time Systems Group, University of York
jack@cs.york.ac.uk

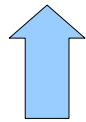


Context

- This paper is concerned with the ACET *and* WCET of the *data accesses* within a *loop kernel*.

Context

- This paper is concerned with the ACET *and* WCET of the *data accesses* within a *loop kernel*.



LOAD/STORE
operations

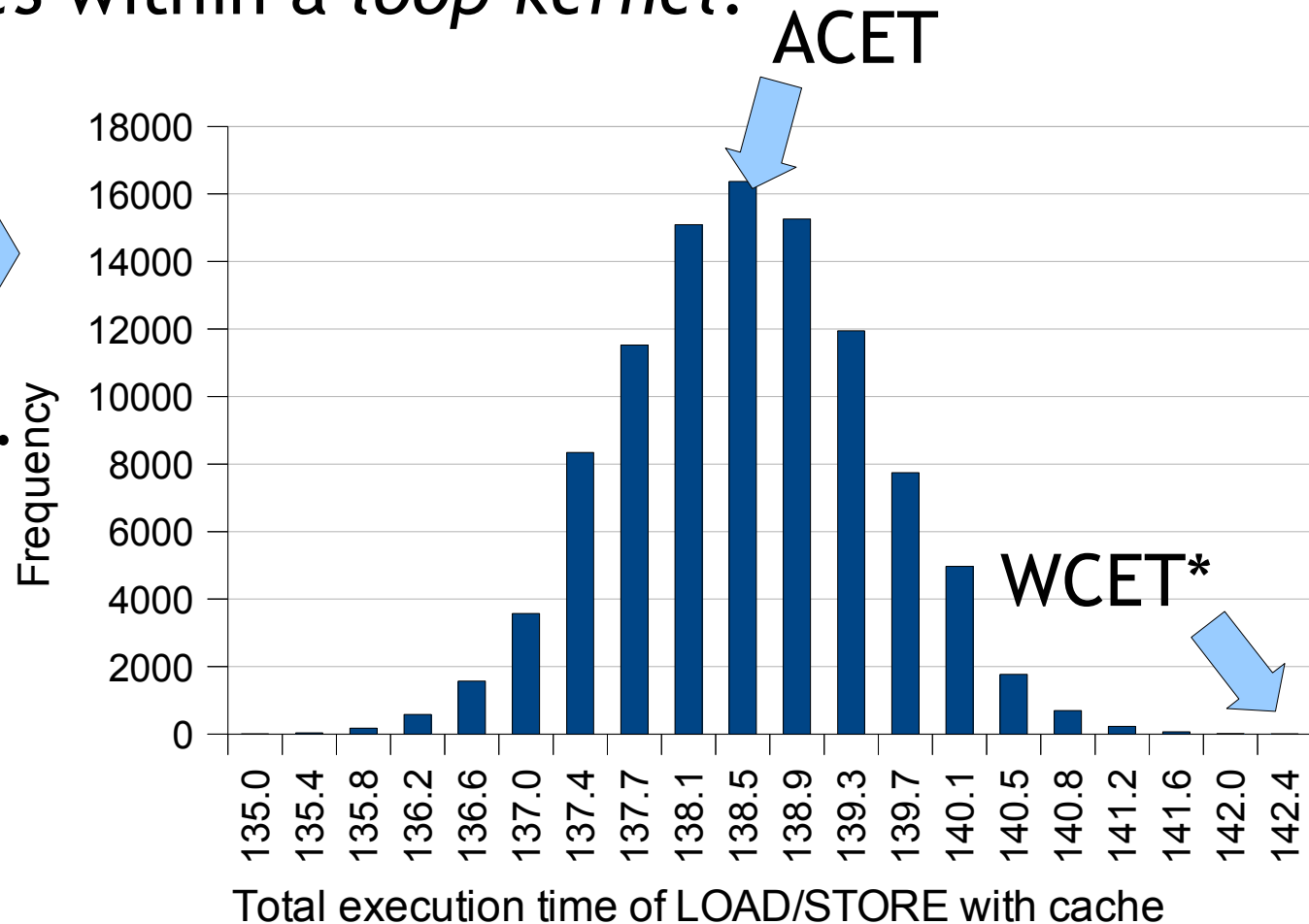
For the purposes of this paper,
all other operations are ignored!

Context

- This paper is concerned with the **ACET** and **WCET** of the *data accesses within a loop kernel*.

Example: ACET and WCET for data accesses with cache.

(susan_smoothing loop kernel)



* strictly: observed WCET...

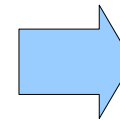
Context

- This paper is concerned with the ACET *and* WCET of the *data accesses* within a *loop kernel*.

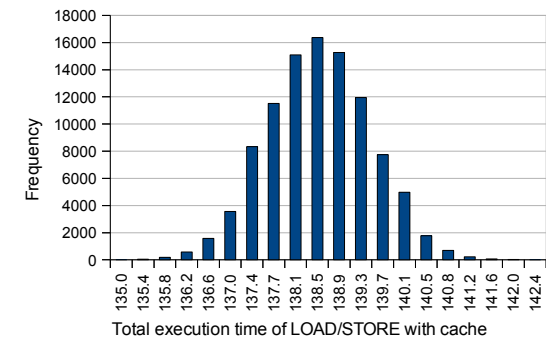


Example: any loop accessing data
and not containing other loops.

```
for(x=-mask_size; x<=mask_size; x++) {  
    brightness = *ip++;  
    tmp = *dpt++ * *(cp-brightness);  
    area += tmp;  
    total += tmp * brightness;  
}
```



generated



Which is better?



Data
cache

Data
scratchpad

- Same *location*
(physically close to
the CPU pipeline)
- Same *access latency*
(very fast)
- Same *purpose* (store
working data set)

Which is better?

Data
cache



Data
scratchpad



- Ease of use (no drivers or program modification)
- Address transparency (suitable for all data)

Which is better?

Data
cache



Data
scratchpad



- Ease of use (no drivers or program modification)
- Address transparency (suitable for all data)
- ...but...
- WCET analysis, if possible at all, comes with *preconditions* on the program and hardware.

Which is better?

Data
cache



Data
scratchpad

- Ease of use (no drivers or program modification)
- Address transparency (suitable for all data)
- ...but...
- WCET analysis, if possible at all, comes with *preconditions* on the program and hardware.
- WCET analysis is often *imprecise* (although always safe!)

Which is better?

Data
cache

Data
scratchpad



- Programs must use the scratchpad explicitly (relocating data to it: memcpy, etc.)
- No address transparency (causing *aliasing* problems with some data structures)

Which is better?

Data
cache

Data
scratchpad



- Programs must use the scratchpad explicitly (relocating data to it: memcpy, etc.)
- No address transparency (causing *aliasing* problems with some data structures)

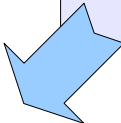
...but...

- Very predictable behavior means very precise timing analysis with fewer preconditions for WCET.

Which is better?

Data cache

Data scratchpad



- Programs must use the scratchpad explicitly (relocating data to it: memcpy, etc.)
- No address transparency (causing *aliasing* problems with some data)
...but...
- Very predictable timing analysis with fewer preconditions for WCET.

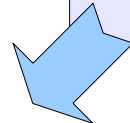


Scratchpad allocation algorithms;
WCET-aware compilers could automate this process

Which is better?

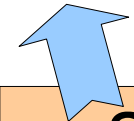
Data cache

Data scratchpad



- Programs must use the scratchpad explicitly (relocating data to it: memcpy, etc.)
- No address transparency (causing *aliasing* problems with some data structures).

...but...



Scratchpad memory management unit: solution in previous work.

- Very predictable timing analysis with fewer preconditions for WCET.

Scratchpads are a good idea

- It's well known that caches are not ideal.
- So why are hard real-time systems still using caches?
 - (1) Most *hardware* has caches - and no scratchpad.
 - (2) Most *software* (legacy, OS) assumes caches.
 - (3) Most *development tools* assume caches.
 - (4) Most *programmers* assume caches.
- Because caches are so commonly used, they are well worth studying, but we should also consider alternatives!

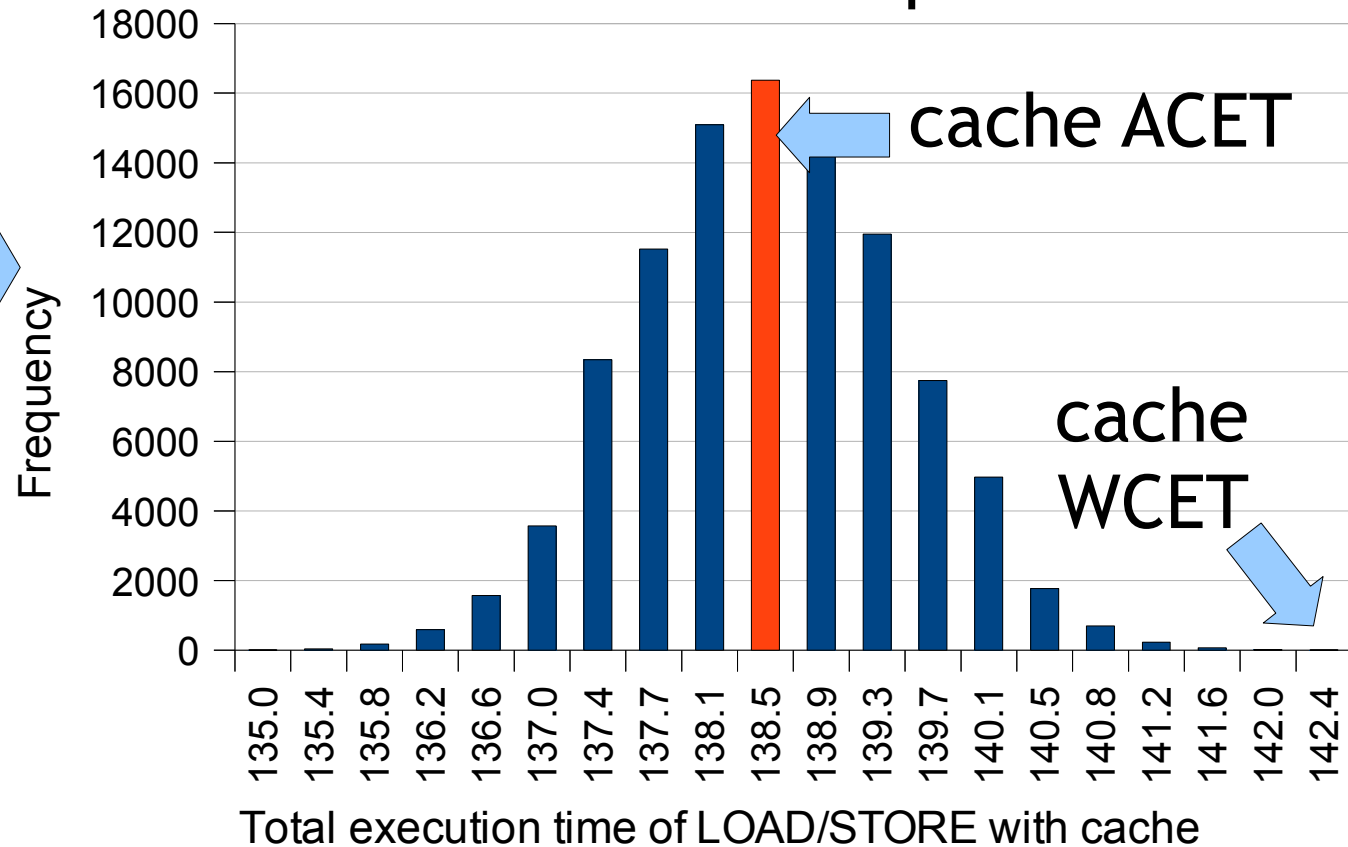
Scratchpads are a good idea

- It's hard to motivate such a change against an entrenched technology: the improvement must be significant.
- It's easy to make a qualitative case for scratchpads, but what good is that?
- Hard, quantitative evidence is needed.

Quantitative comparison

- Experiment idea: given a loop kernel and an otherwise fixed computer architecture, compare ACET and WCET with a cache and a scratchpad.

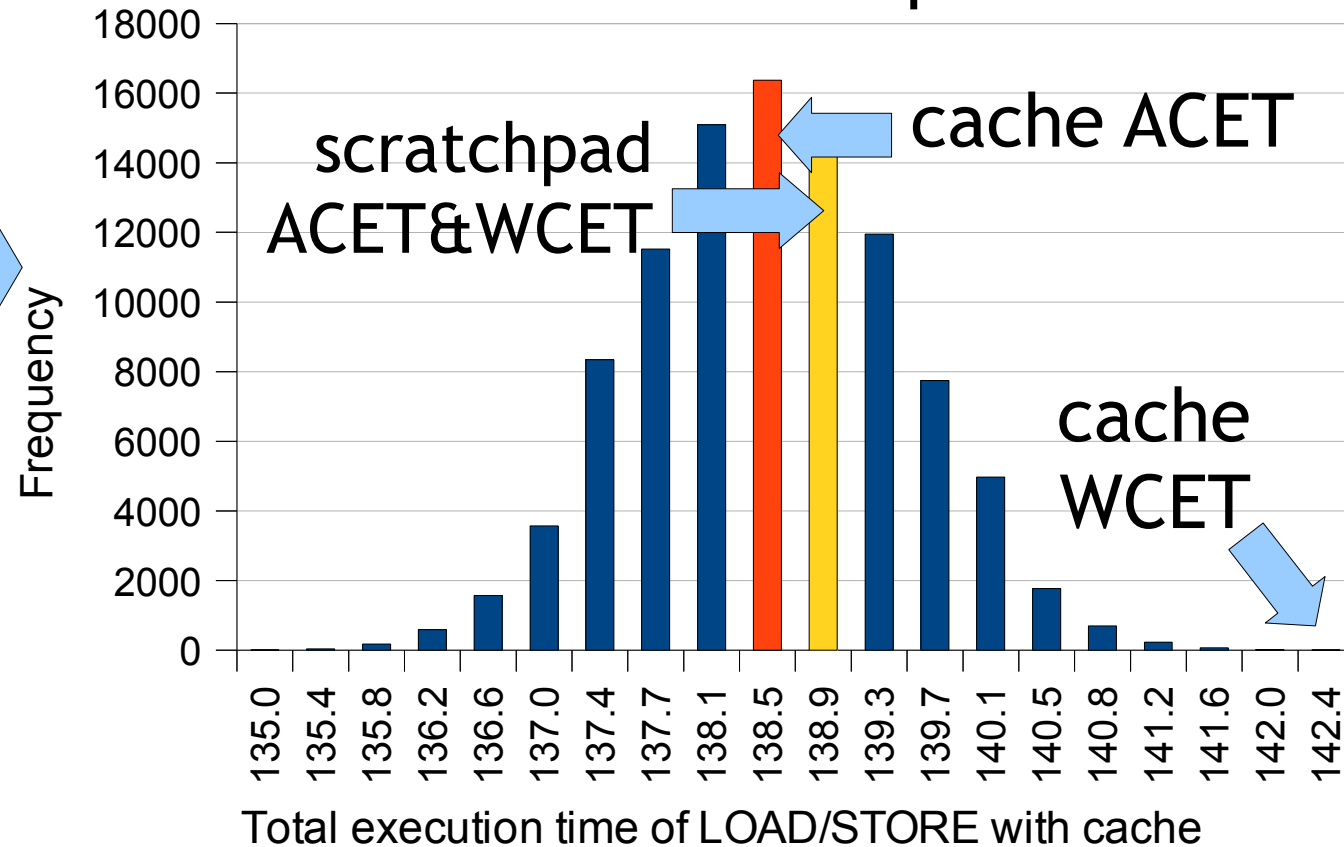
- Recall this distribution:



Quantitative comparison

- Experiment idea: given a loop kernel and an otherwise fixed computer architecture, compare ACET and WCET with a cache and a scratchpad.

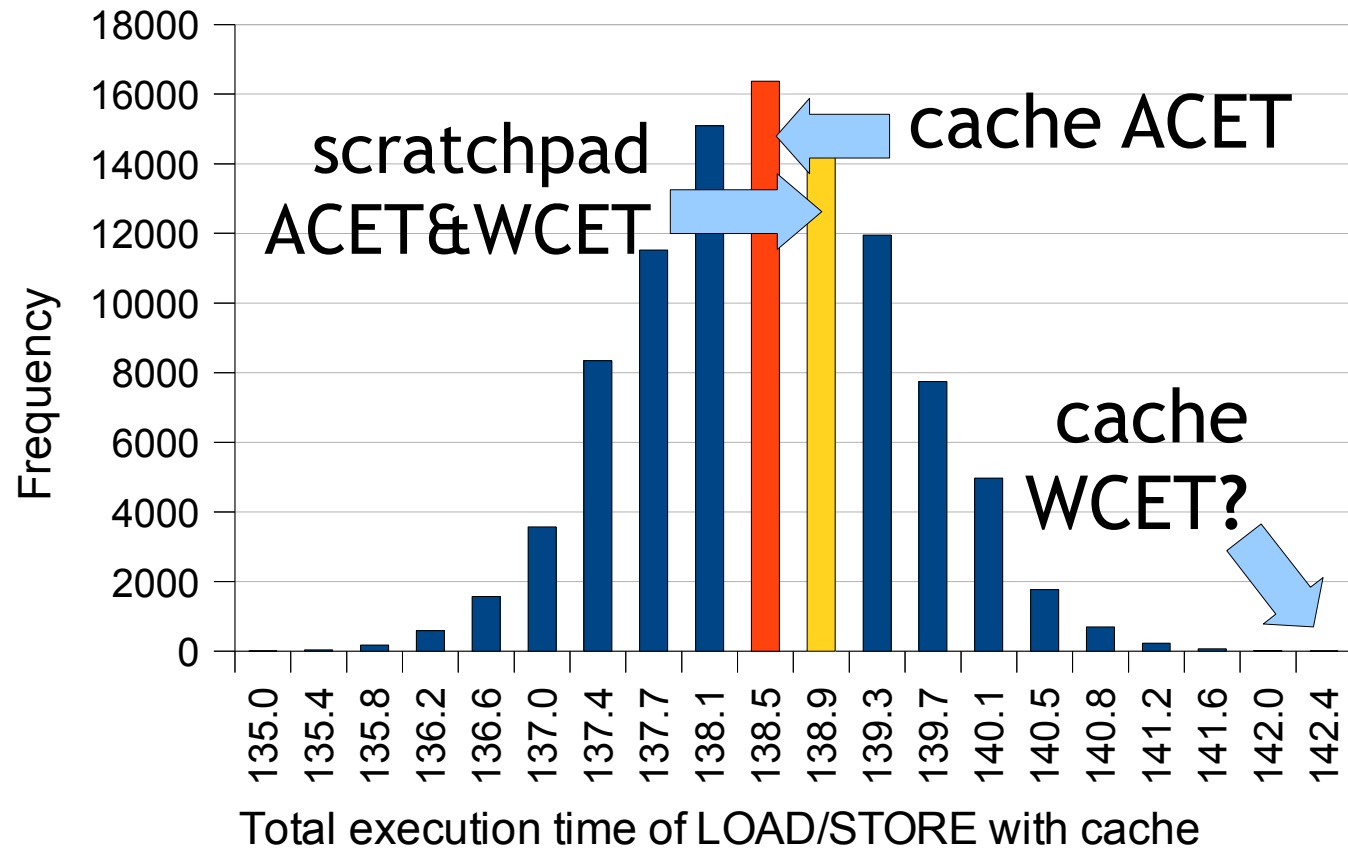
- Add scratchpad ACET/WCET



Quantitative comparison

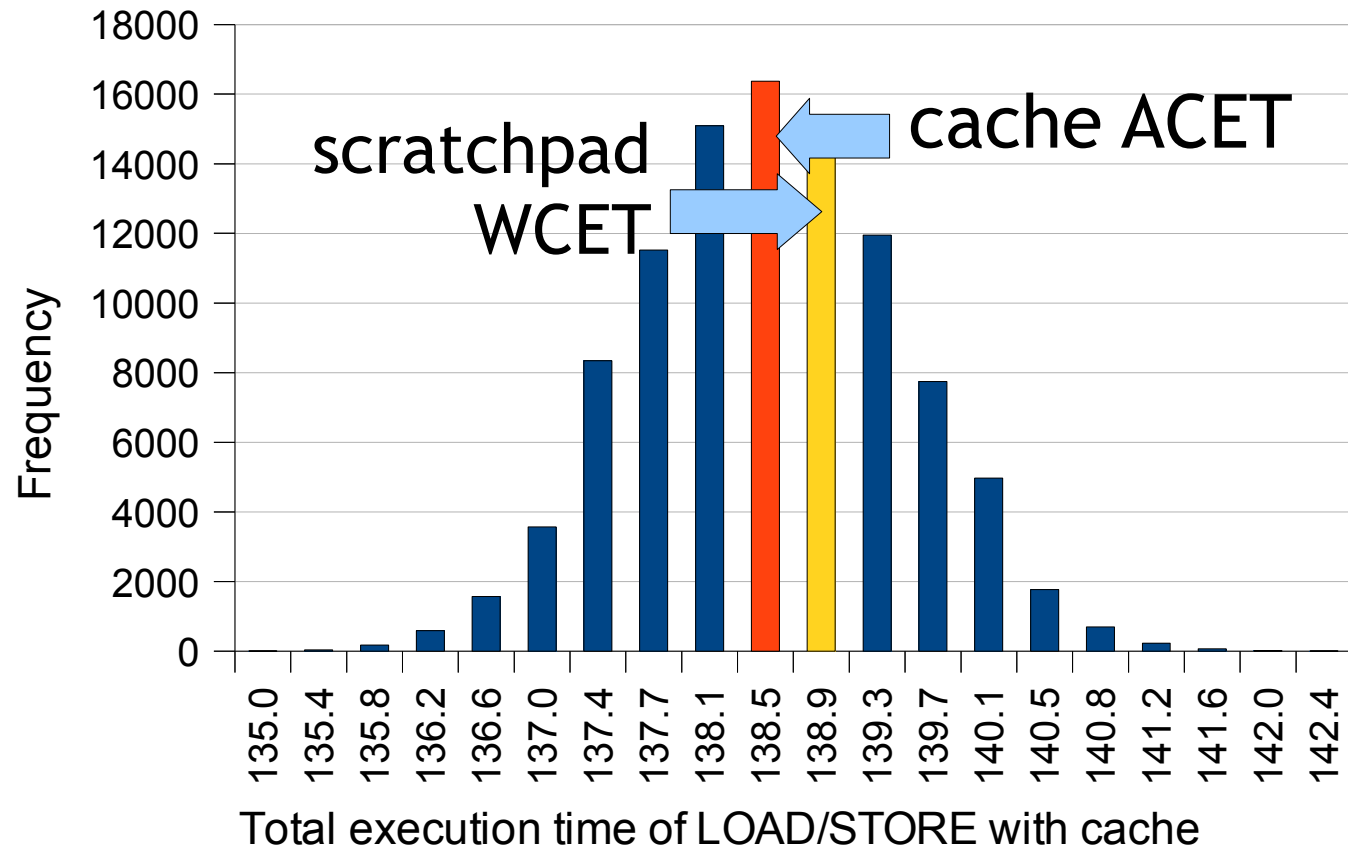
- Cache ACET, scratchpad ACET & WCET - easily found.
- Cache WCET - not easy to find!

- WCET analysis *overestimates* cache WCET.
- Measurement *underestimates* cache WCET.



Quantitative comparison

- A fair comparison is very difficult!
- Therefore, we bias the comparison *against* scratchpads!
- Compare WCET (scratchpad) with ACET (ideal cache)



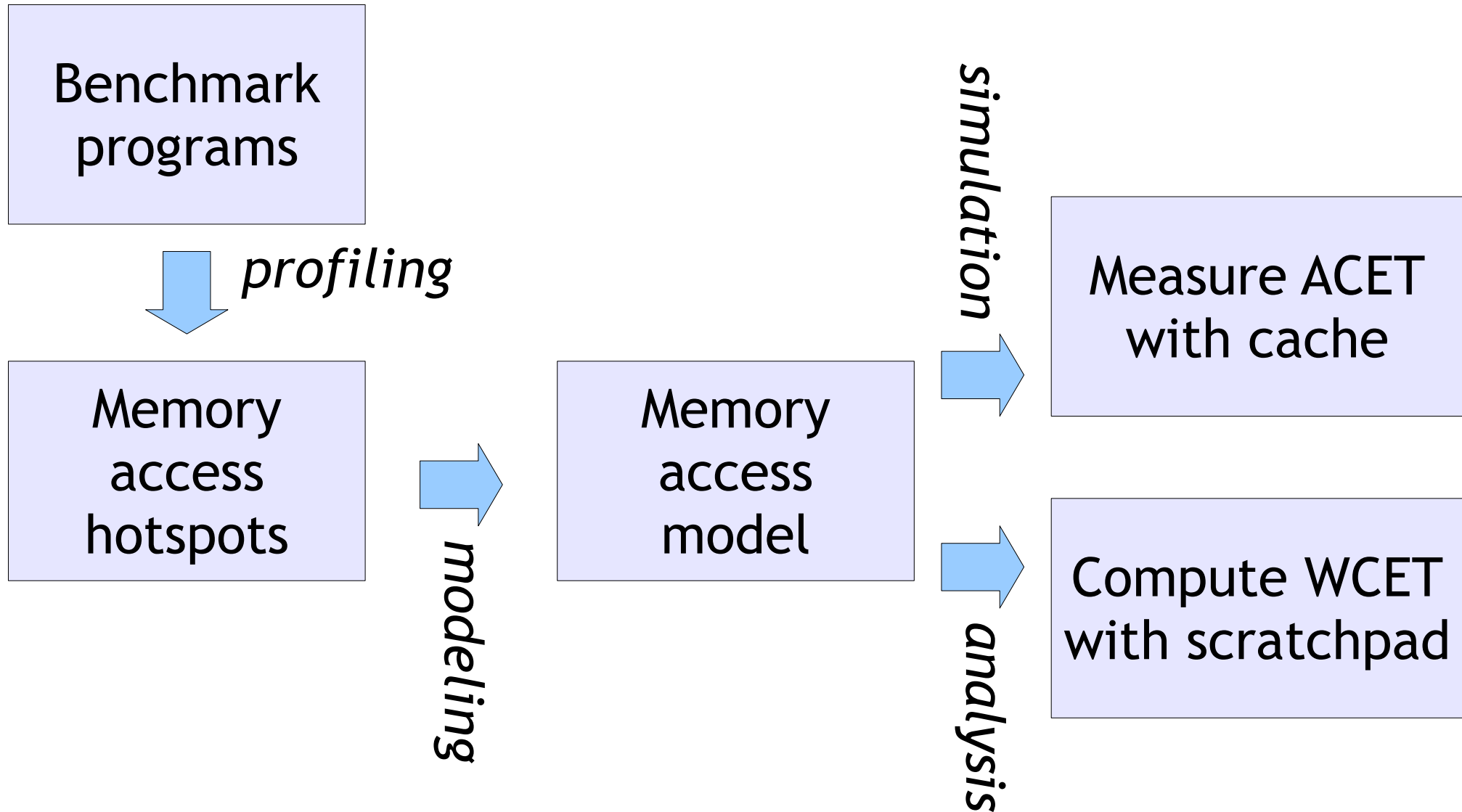
Experiment design

- Same CPU model, external memory, data bus.
- Same amount of on-chip memory; but configured as either idealised cache or realistic scratchpad.
- Same *loop kernels* executed on both.
- WCET computed for scratchpad (including transfer time for loading/unloading scratchpad).
- ACET measured for cache (including cache misses).

Input data: loop kernels

- Extracted from benchmark software:
adpcm, ammp, art, bzip2, crc32, djpeg, fft, gap, gsm, ispell, patricia, susan...
- By profiling, we identified *hotspots* with a large amount of memory accesses to heap and global data.
- These loops were then translated into *models* that reproduce similar access patterns.

Illustration



Model example

- From *susan* benchmark:

```
for(x=-mask_size; x<=mask_size; x++) {  
    brightness = *ip++;  
    tmp = *dpt++ * *(cp-brightness);  
    area += tmp;  
    total += tmp * brightness;  
}
```

Three pointers accessed on each iteration:

- *ip*: accessed once per iteration, sequentially with step 1.
- *dpt*: accessed once per iteration, sequentially with step 1.
- *cp*: accessed once per iteration, randomly, range 286 bytes.

Model example

- From *susan* benchmark:

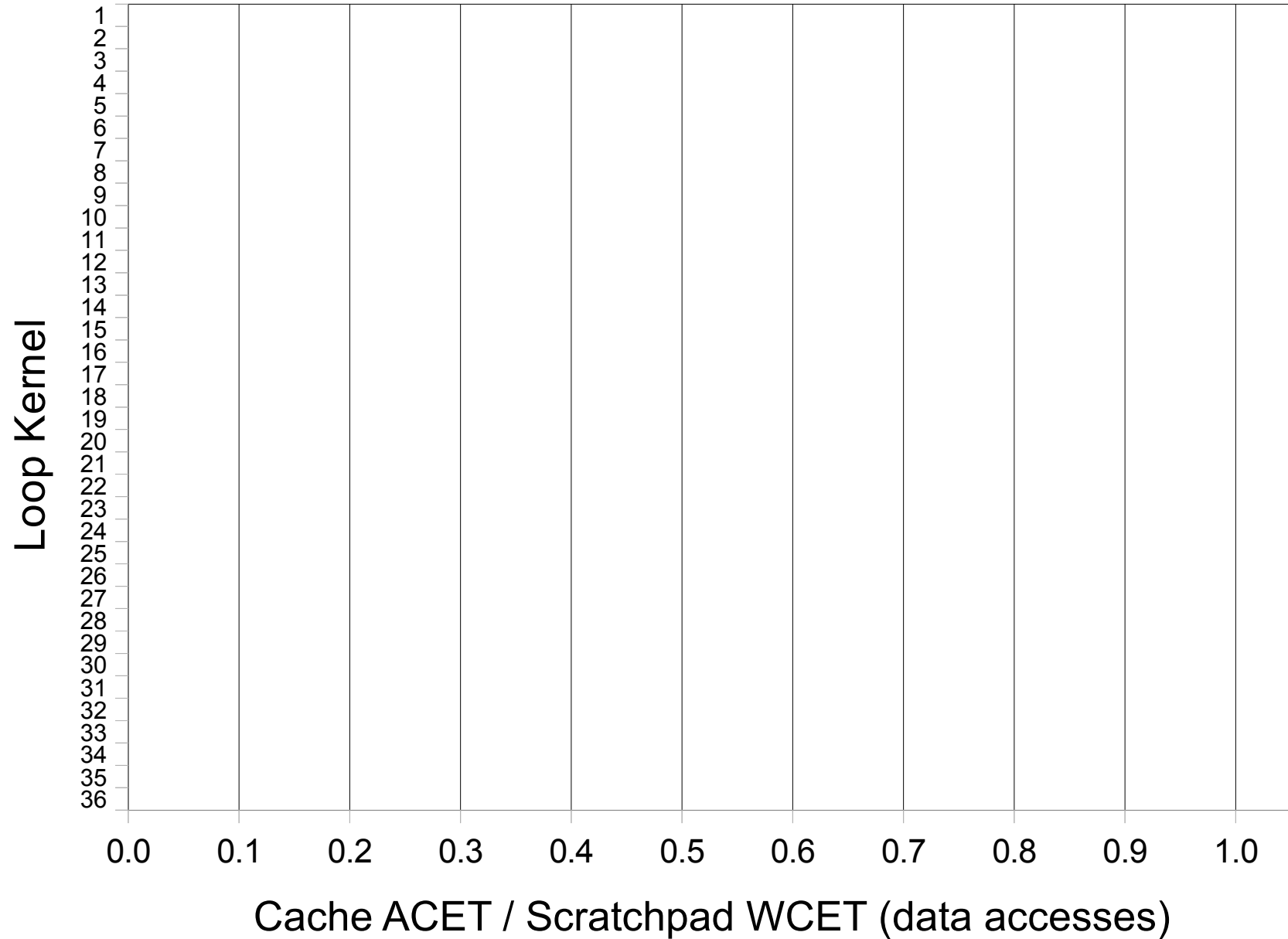
```
for(x=-mask_size; x<=mask_size; x++) {  
    brightness = *ip++;  
    tmp = *dpt++ * *(cp-brightness);  
    area += tmp;  
    total += tmp * brightness;  
}
```

- Memory access model:

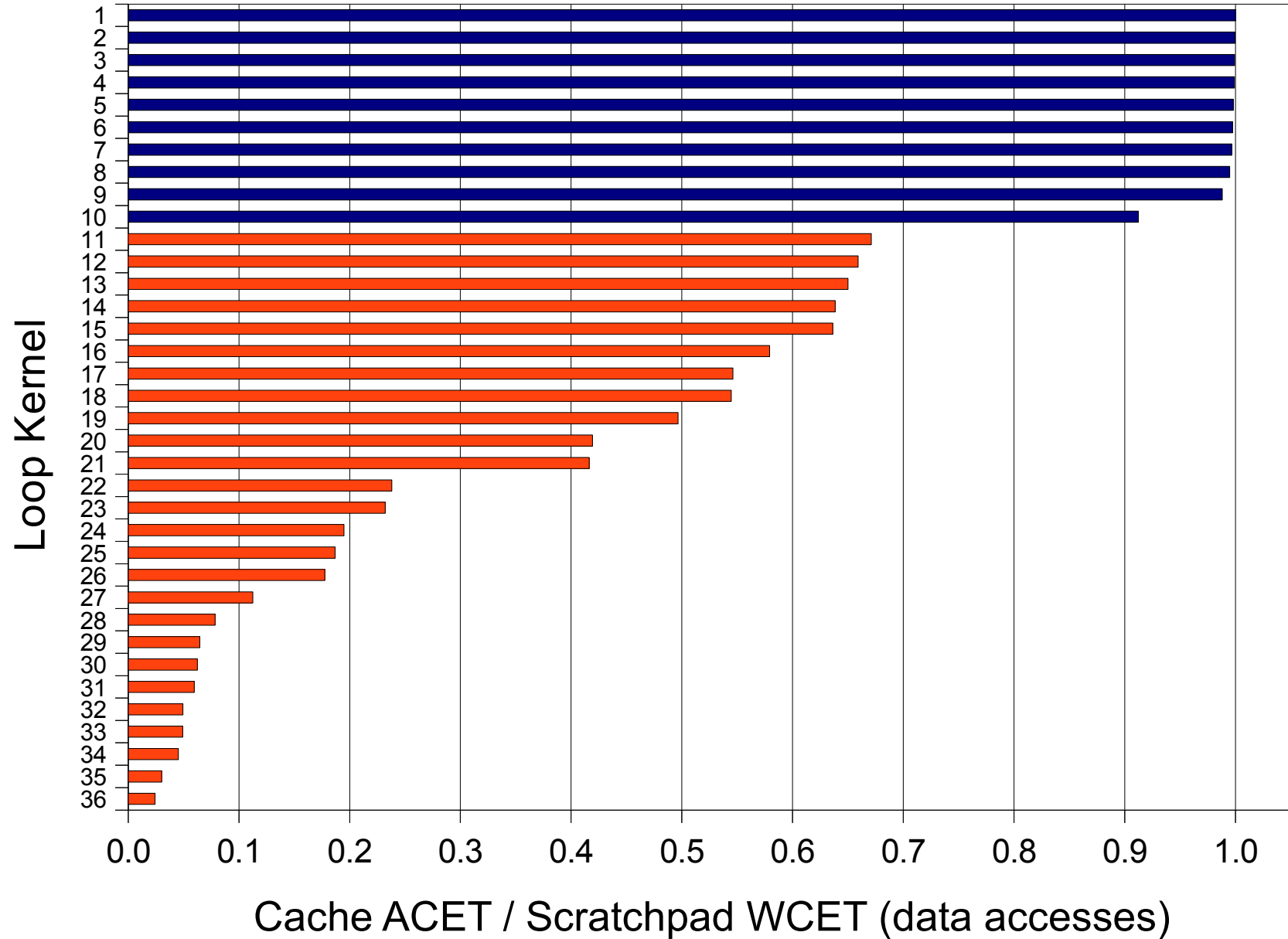
```
for(i=0; i<N; i++) {  
    LOAD(i + ip); LOAD(i + dpt);  
    LOAD(RANDOM(286) + cp);  
}
```


- Models were generated for 36 loop kernels.
- ACET of memory access operations with cache was estimated by measurement:
 - Generate random pointer values as inputs.
 - Assume fully associative write-back cache with 64-byte lines and true LRU replacement. Write-back time counts towards ET.
- WCET of memory access with scratchpad was determined by analysis.
 - Objects are allocated to scratchpad to minimise WCET; loaded before/after the loop kernel runs, load time counts toward WCET.

Initial comparison



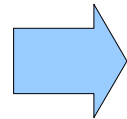
Initial comparison



What causes the shortfall?

- Objects that can't be completely stored in scratchpad because they're too large.

- Example: `ip` is too large for scratchpad if



```
for(x=-mask_size; x<=mask_size; x++) {  
    brightness = *ip++;  
    tmp = *dpt++ * *(cp-brightness);  
    area += tmp;  
    total += tmp * brightness; }  
}
```

$2 \times \text{mask_size} > \text{scratchpad_size}$

- What can be done about these?

Loop tiling

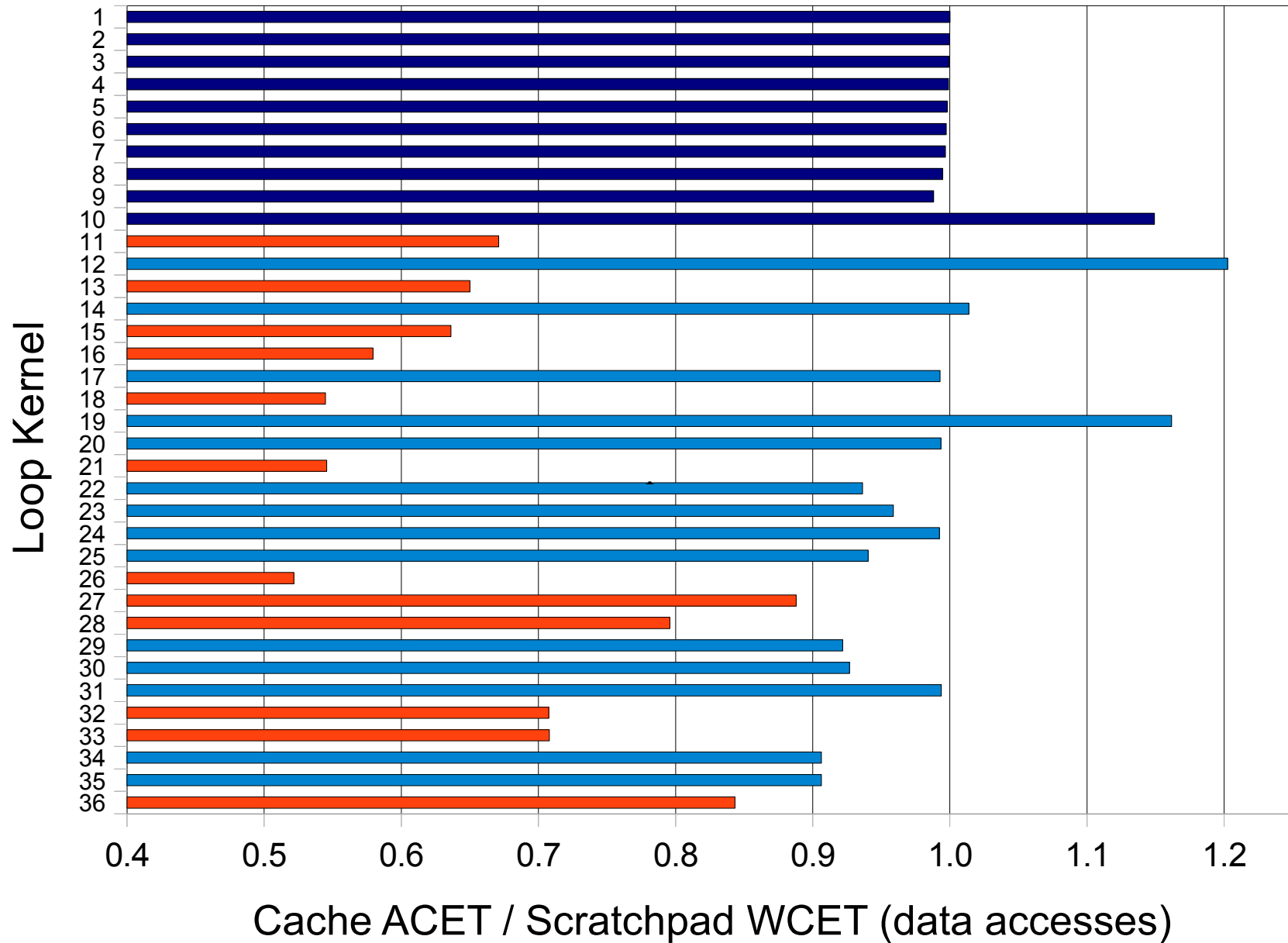
- For iterative accesses, update scratchpad contents during the loop by synthesizing a second outer loop:

```
for(x=-mask_size; x<=mask_size; x++) {  
    brightness = *ip++;  
    tmp = *dpt++ * *(cp-brightness);  
    area += tmp;  
    total += tmp * brightness;  
}
```

Loop tiling for **ip** 

```
for(x=-mask_size; x<=mask_size; ) {  
    load_scratchpad(ip, M);  
  
    for(i=M; (i > 0) && (x <= mask_size); i--, x++) {  
        brightness = *ip++;  
        tmp = *dpt++ * *(cp-brightness);  
        area += tmp;  
        total += tmp * brightness;  
    }  
    unload_scratchpad(ip, M);  
}
```

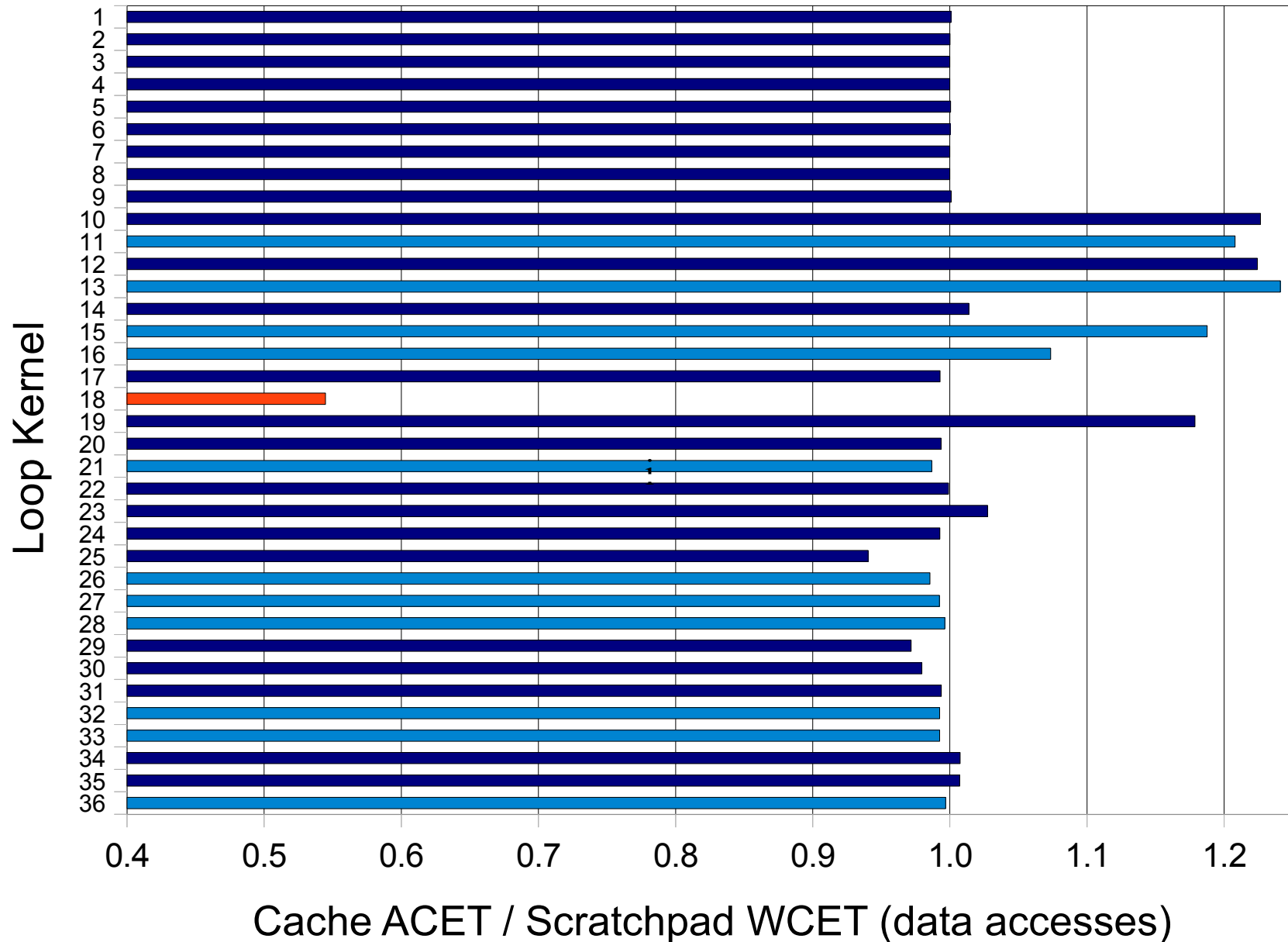
With loop tiling



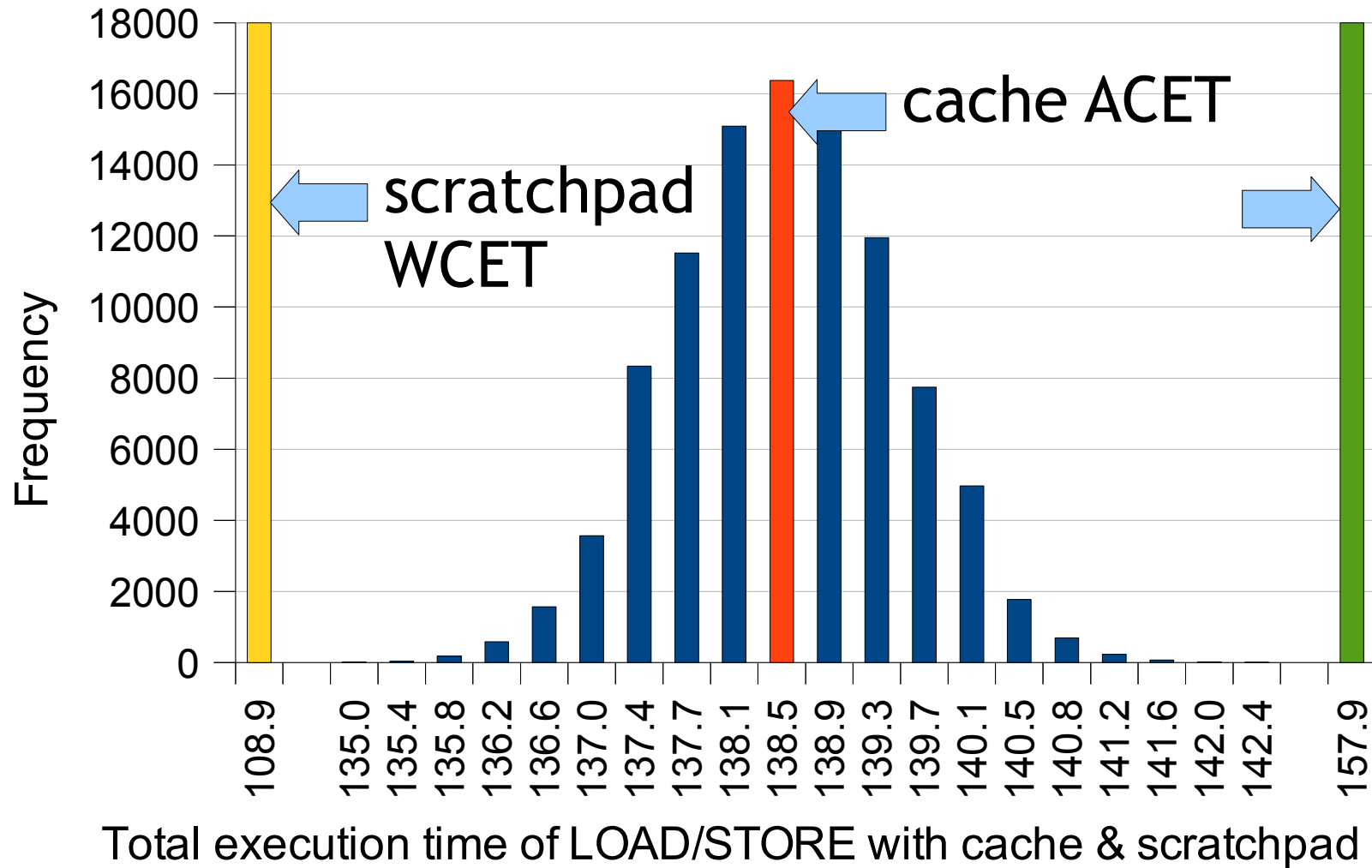
A great improvement

- But there's still a discrepancy.
- This is because the cache can *discard* data that hasn't changed; but our scratchpad model insists it is always written back to avoid aliasing issues.
- Solution: add explicit hardware support for discarding read-only data in scratchpad to the SMMU.

With loop tiling & discarding



Back to the example



Success..

- WCET with scratchpad is now within 5% of the cache ACET in 34 of 36 cases.
- WCET with scratchpad is actually smaller than the cache ACET in 16 cases!
 - Random accesses within large areas of memory don't disrupt the scratchpad state.
- The only significant discrepancy is loop kernel 18.

Loop kernel 18

- Random access to array of size 20.5kb.
- Up to 78% chance that any particular byte will be found within a 16kb cache.
- However, 0% chance that any byte will be found in scratchpad, as the entire array cannot be stored.
- Hence, cache ACET is less than scratchpad WCET.

Conclusion

- A comparison of a highly idealised data cache to a realistic scratchpad using loop kernels.
- Comparison is biased strongly in favour of caches by comparing cache ACET to scratchpad WCET, no consideration of cache WCET.
- Simple optimisations bring the scratchpad WCET close to the cache ACET, and make it better in some cases!

Strong evidence that a move away from caches will be worthwhile. “Time-predictable isn't slow”.

Thankyou

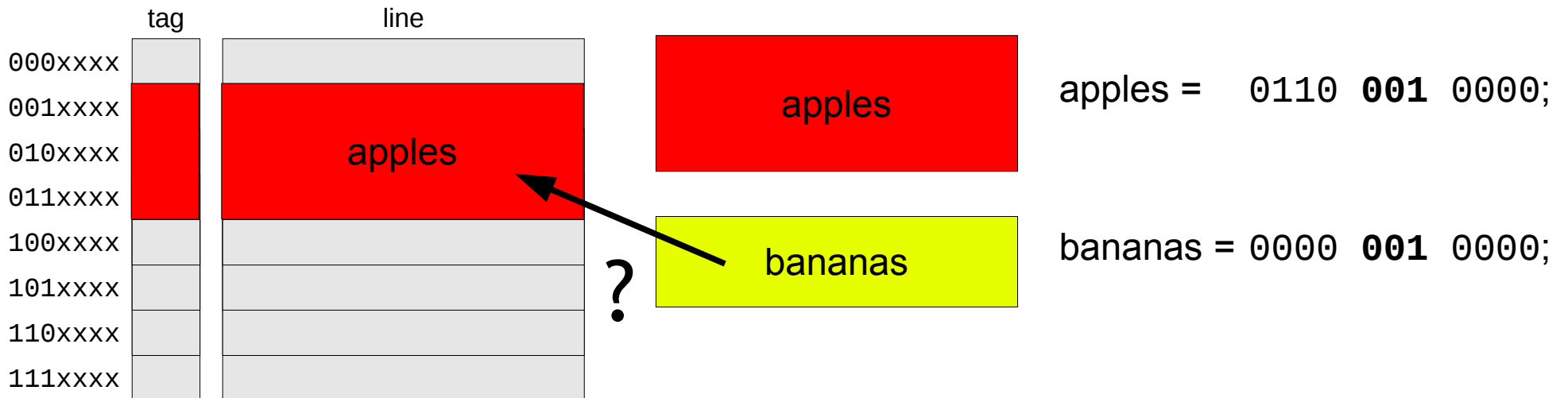


The Real-time Systems Group
at the University of York.
<http://www.cs.york.ac.uk/rts/>

Additional material

Lockable Data Cache Restriction

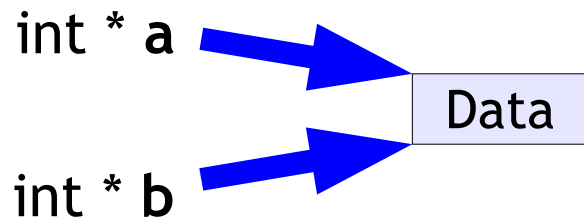
- In general, can't load & lock $N+1$ objects together in an N -way associative cache



- Major restriction on the working set!

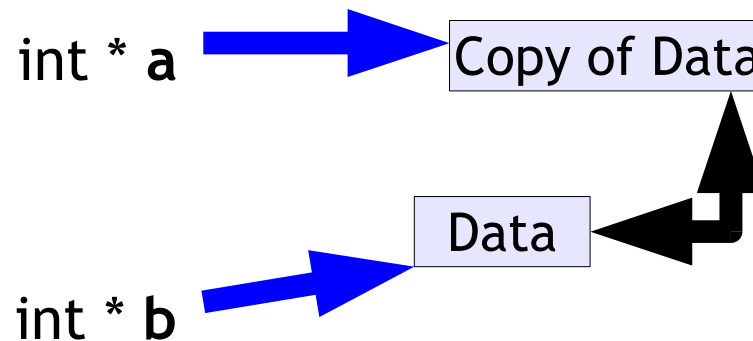
Scratchpad Memory Restriction

- If a program moves an object between external memory and scratchpad, its *address* changes.



before

```
a[1]=2;  
b[1]=3;  
(a[1] == 3) is TRUE
```

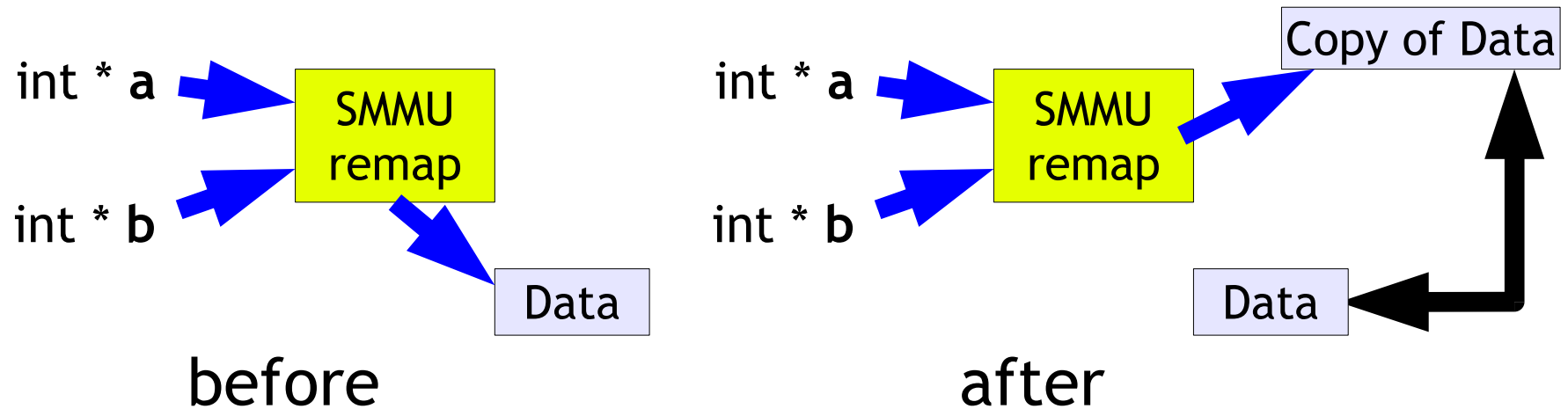


after

```
a[1]=2;  
b[1]=3;  
(a[1] == 3) is FALSE
```


Address Transparency Example

- If a program moves an object between external memory and scratchpad with SMMU, its *address does not change*.



```
a[1]=2;  
b[1]=3;  
(a[1] == 3) is TRUE
```

```
a[1]=2;  
b[1]=3;  
(a[1] == 3) is TRUE
```