Implementing Time-Predictable Load and Store Operations

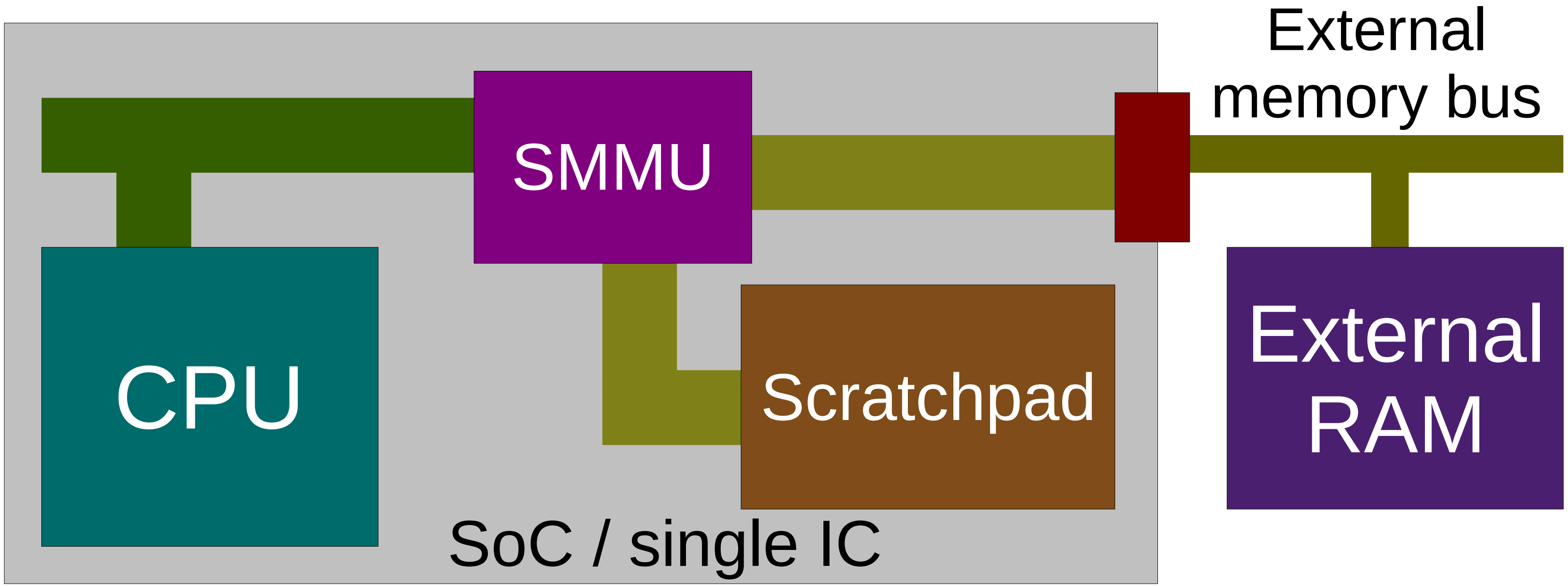
Jack Whitham
jack@cs.york.ac.uk

Neil Audsley
neil@cs.york.ac.uk

SMMU

Scratchpad Memory Management Unit.

The combination, SMMU + Scratchpad, is a *data cache alternative*.



Why replace the cache?

Within e*mbedded hard real-time systems...*

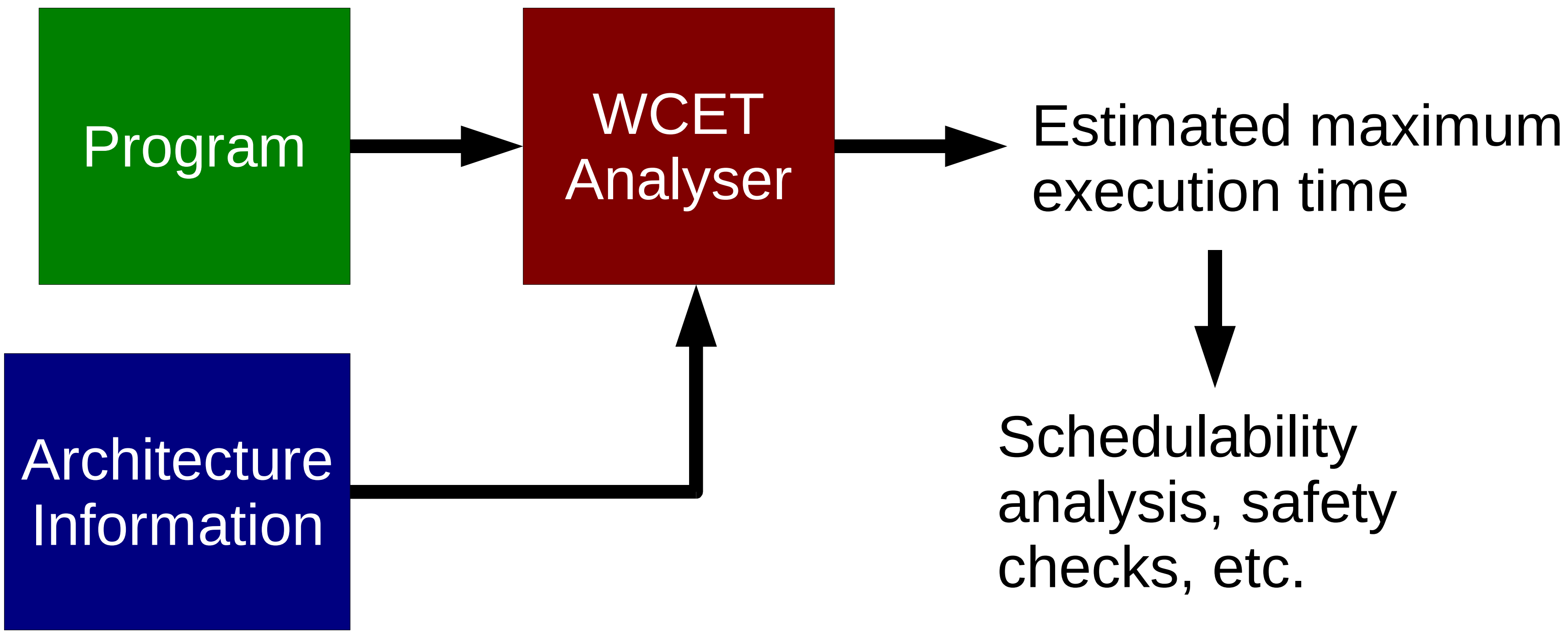
Time-predictable behavior is required.

- There are hard deadlines.

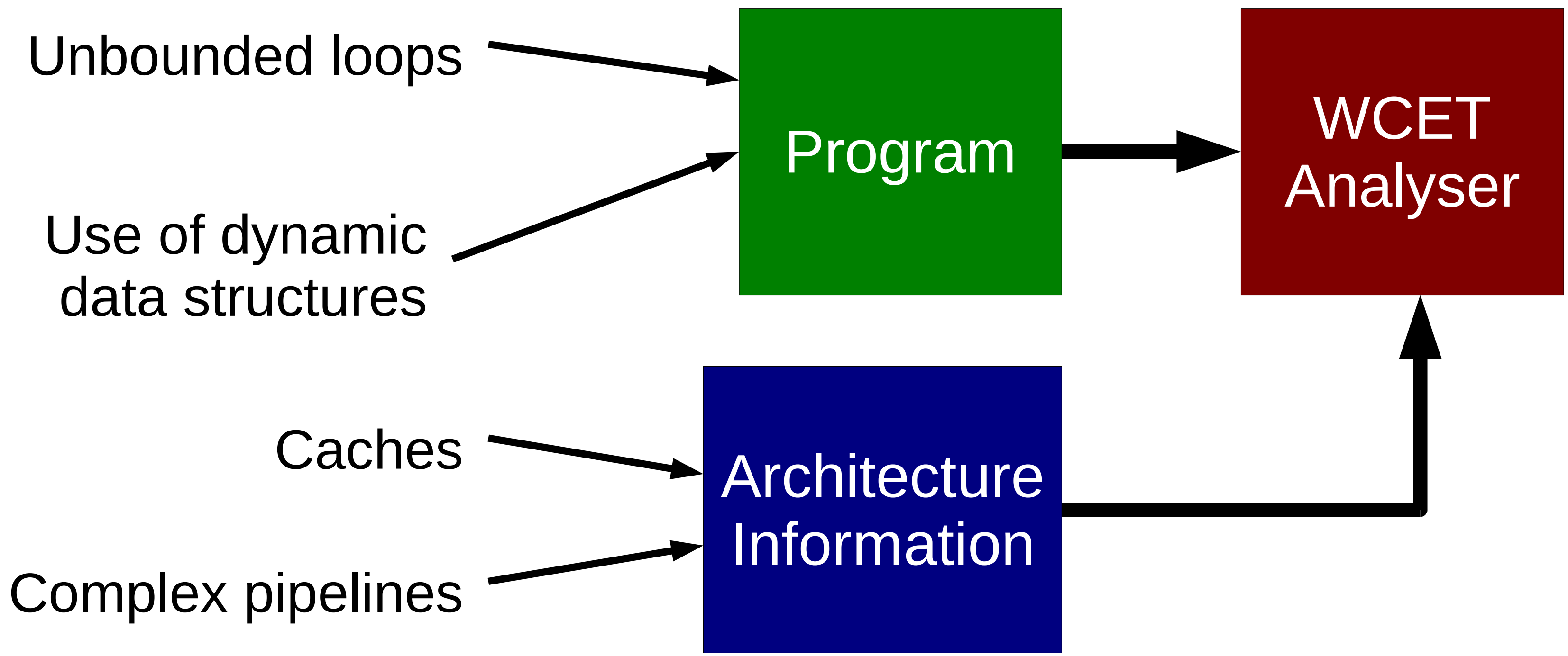
Systems need to be composed of time-predictable components.

- Caches are not very predictable for *worst-case execution time* (WCET).

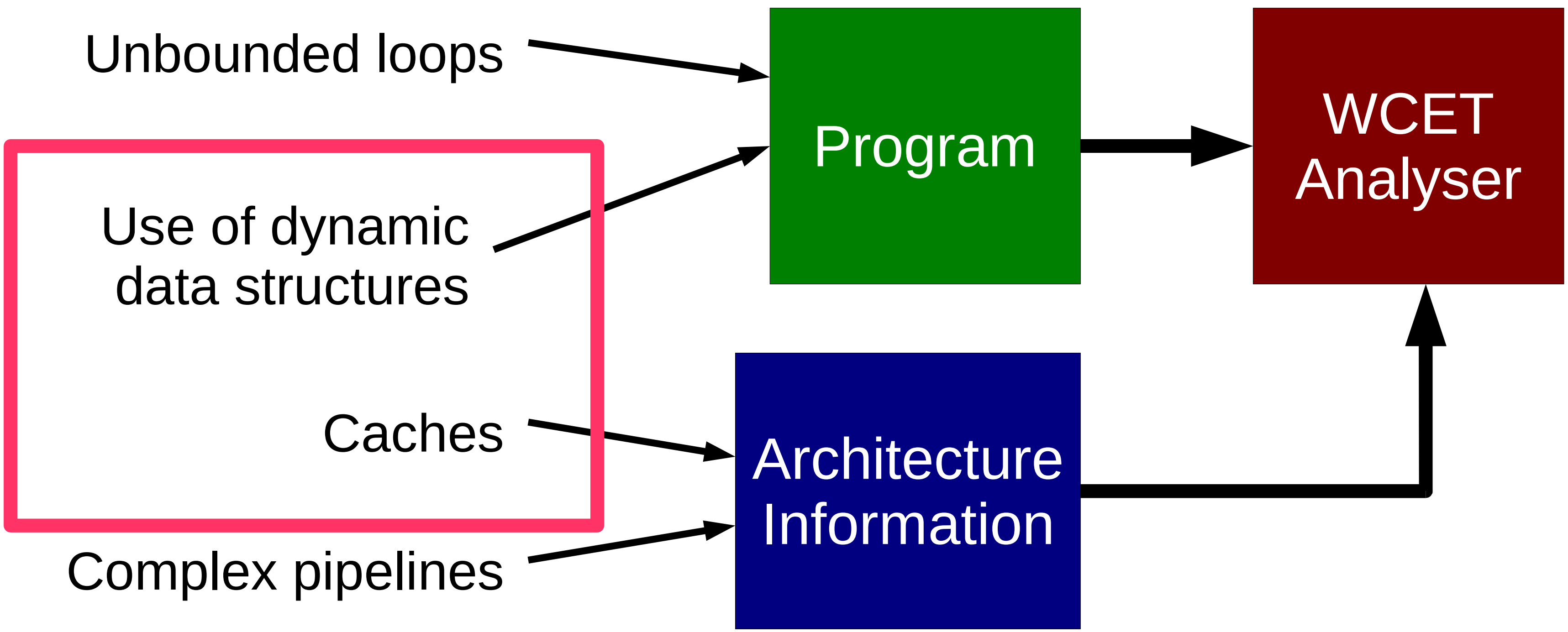
Are deadlines met? [25]



WCET Difficulties (1)



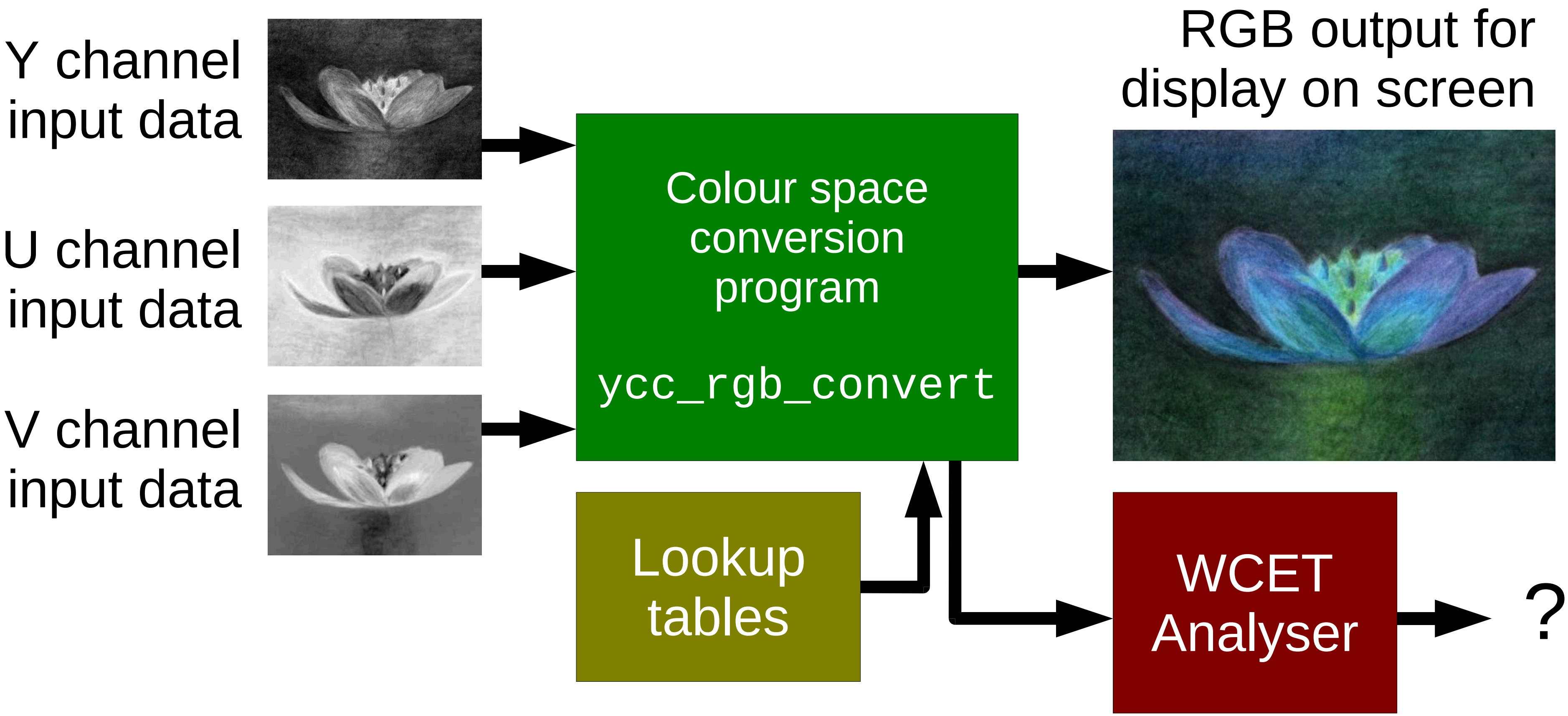
WCET Difficulties (2)



The focus of this paper is on *replacing* data caches with something *time-predictable* that still allows the use of dynamic data structures.

Example (1)

Colour space conversion in libjpeg [12].



Example (2)

`ycc_rgb_convert` reads from eight objects in memory, and writes to one.

3 buffers of input data

4 conversion tables

1 range limit table

1 output buffer (RGB)

```
FOR row FROM 0 TO num_rows – 1 DO
    FOR col FROM 0 TO num_cols - 1 DO
        y  := inptr0[col];
        cb := inptr1[col];
        cr := inptr2[col];
        outptr[0] := range_limit[y + Crrtab[cr]];
        outptr[1] := range_limit[y +
            ((Cbgtab[cb] + Crgtab[cr]) / 65536) ];
        outptr[2] := range_limit[y + Cbbtab[cb]];
        outptr := outptr + 3;
    END FOR;
END FOR;
```

If a Cache is Used... (1)

As the *base addresses* of the nine objects are unknown during analysis,

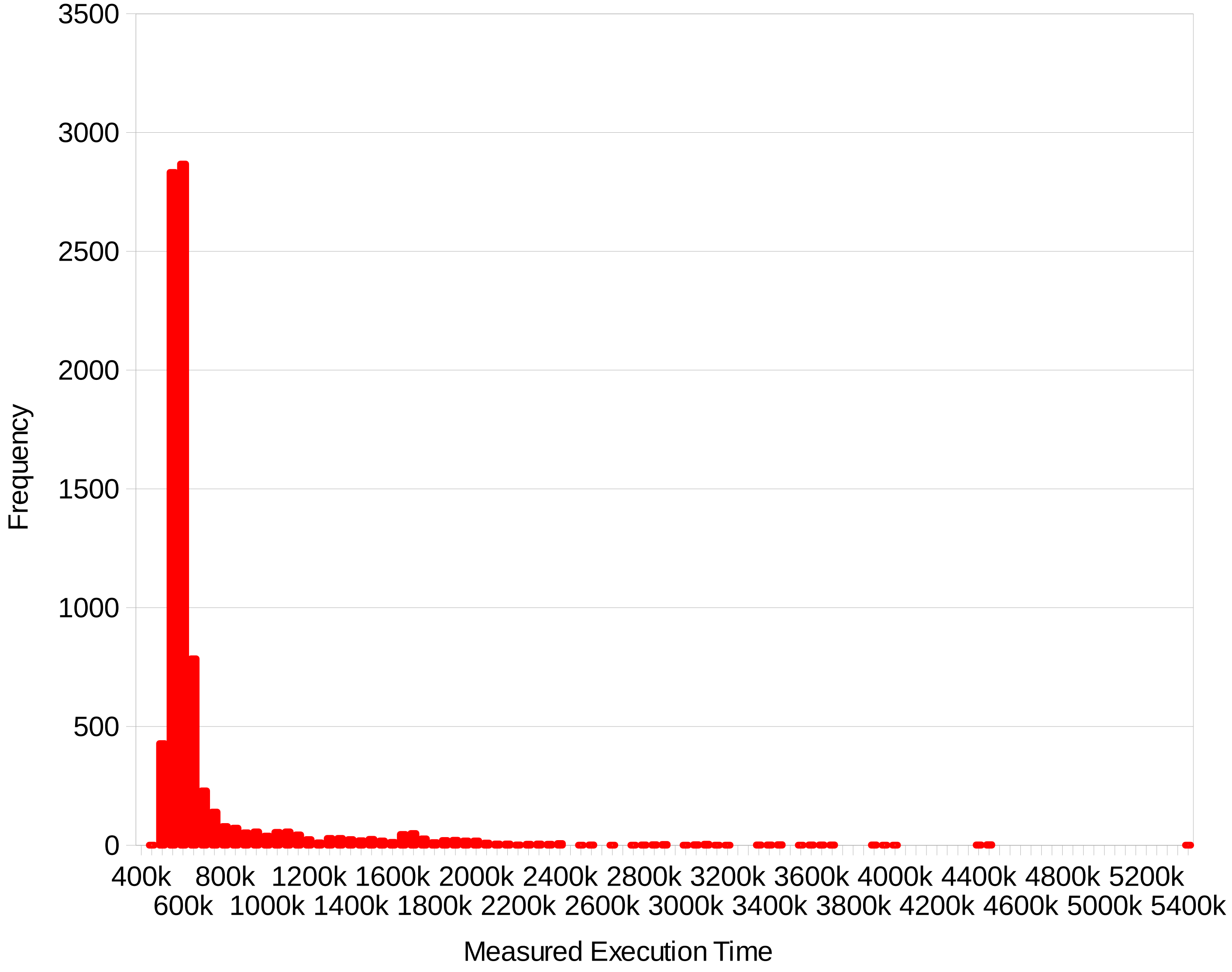
and the *input data* in the Y, U and V buffers is unknown...

⇒ Any pair of memory accesses may *conflict*!

⇒ Number of cache misses affected by *reference string*.

⇒ What is the WCET?

If a Cache is Used... (2)



Summary

Unknown base address / unknown input data is a problem for WCET analysis of data caches.

Caches are a poor solution here.

Caches should be replaced

System	Latency (CPU clock cycles)	CPU frequency (MHz)	Bus frequency (MHz)
ARM MPcore [1]	**79**	210	70
StrongARM-110 [30]	**17**	50	50
PPC 405 [36]	**33**	100	125
Microblaze [40]	**31**	125	125

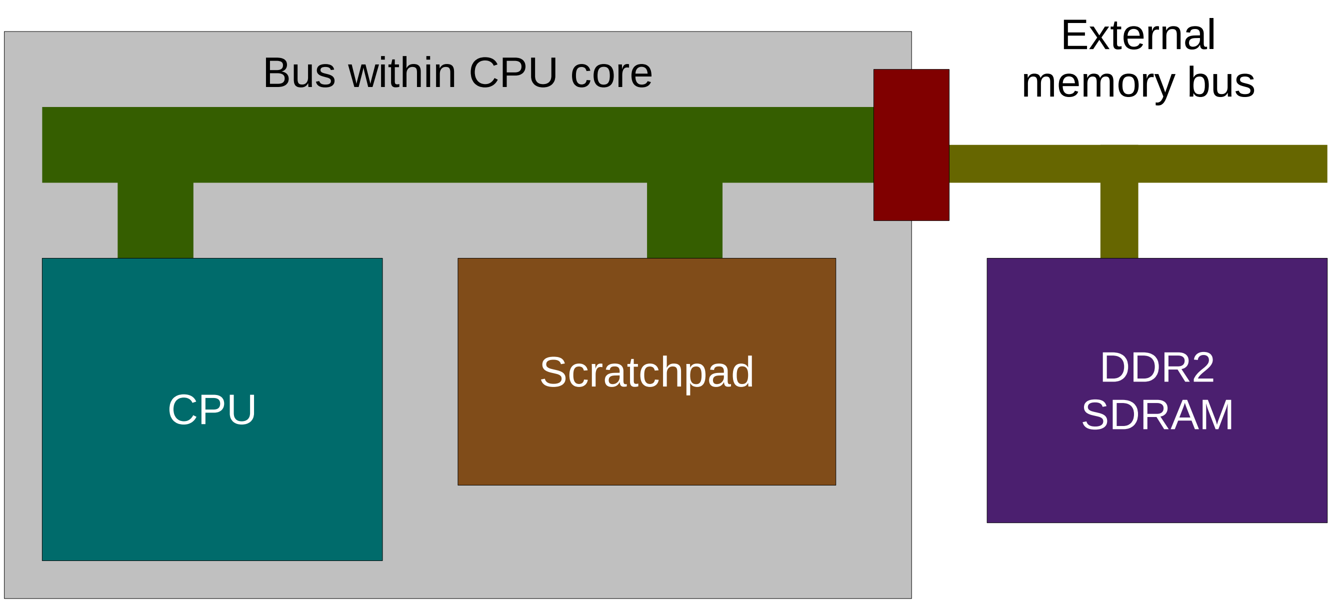
Embedded hard real-time systems need a *replacement* for a cache.

Must have time-predictable behaviour that is independent of *base address* and *input data*.

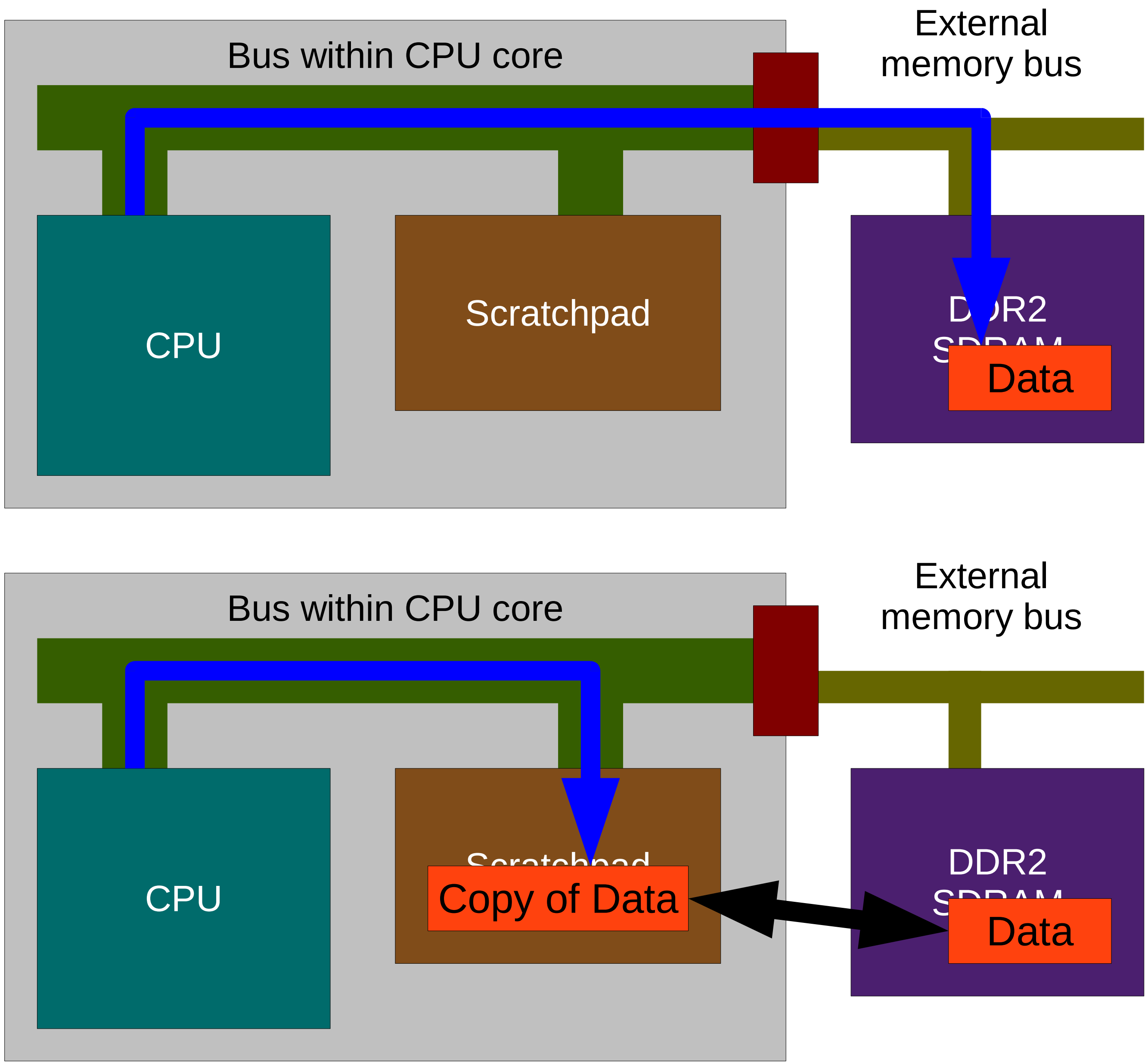
Scratchpad (1)

A small, fast and energy-efficient RAM that is physically located close to the CPU core [31].

Accesses to scratchpad are always time-predictable regardless of input data.



Scratchpad (2)



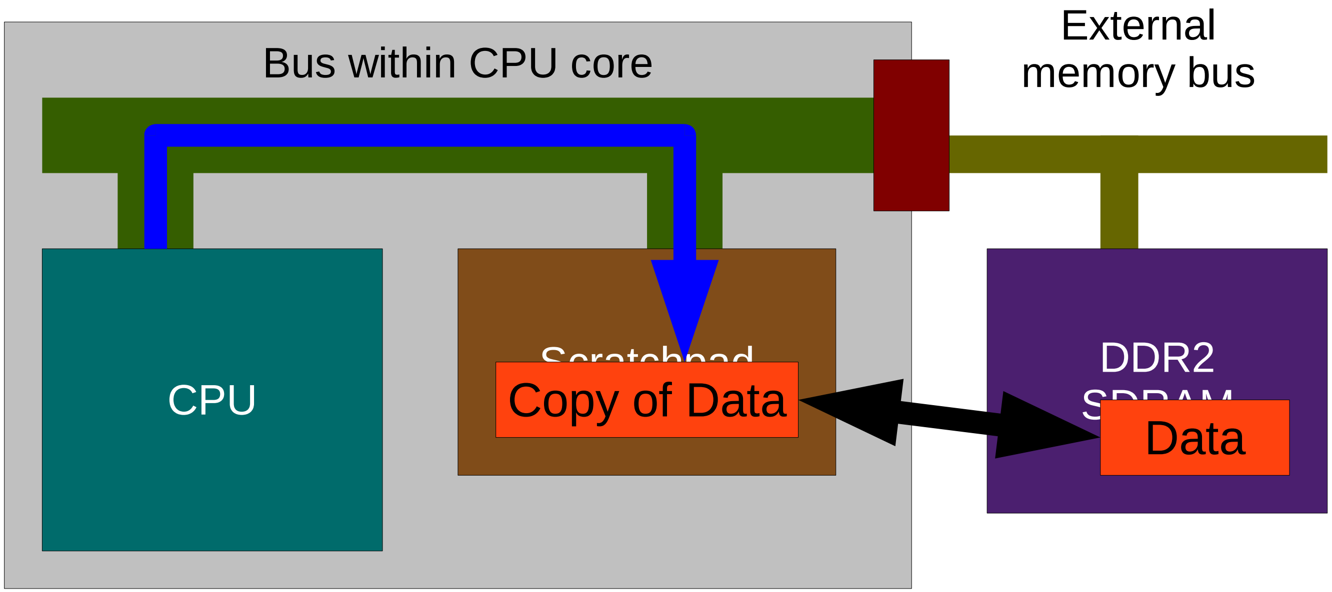
Scratchpad (3)

Problem solved?

No.

⇒ The *physical* location of data changes.

⇒ The *logical* address of data also changes.

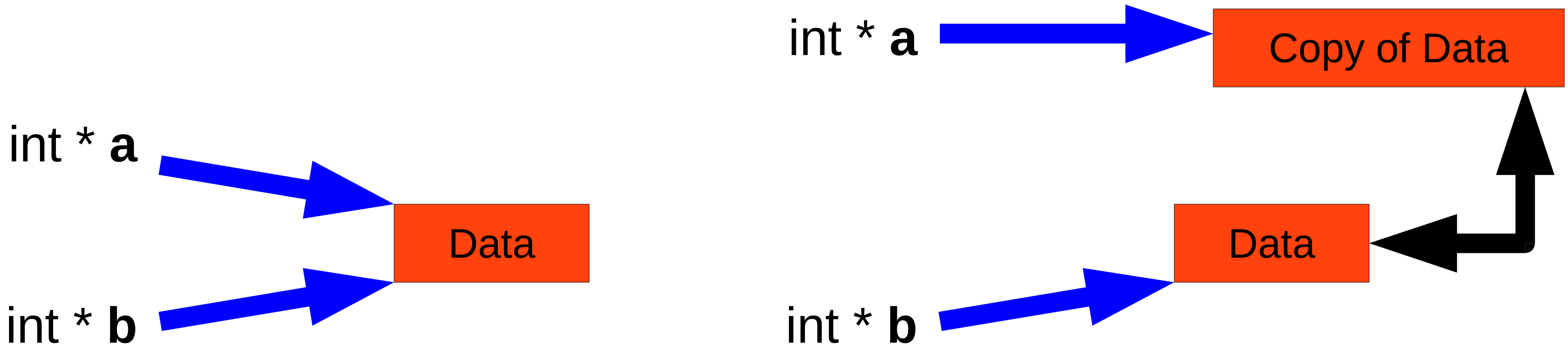


Scratchpad (4)

Relocating data:

- *Invalidates* pointers to that data;

- Changes the behaviour of *aliased* pointers.



⇒ A major problem for dynamic data structures.

Previous Work

Udayakumaran, Dominguez and Barua used whole-program pointer analysis to safely manage scratchpad space [33].

This solves problems caused by *pointer aliasing* and *invalidation*.

It doesn't help with WCET analysis.

⇒ Location of data is determined at runtime and is unknown during *analysis*.

New solution required

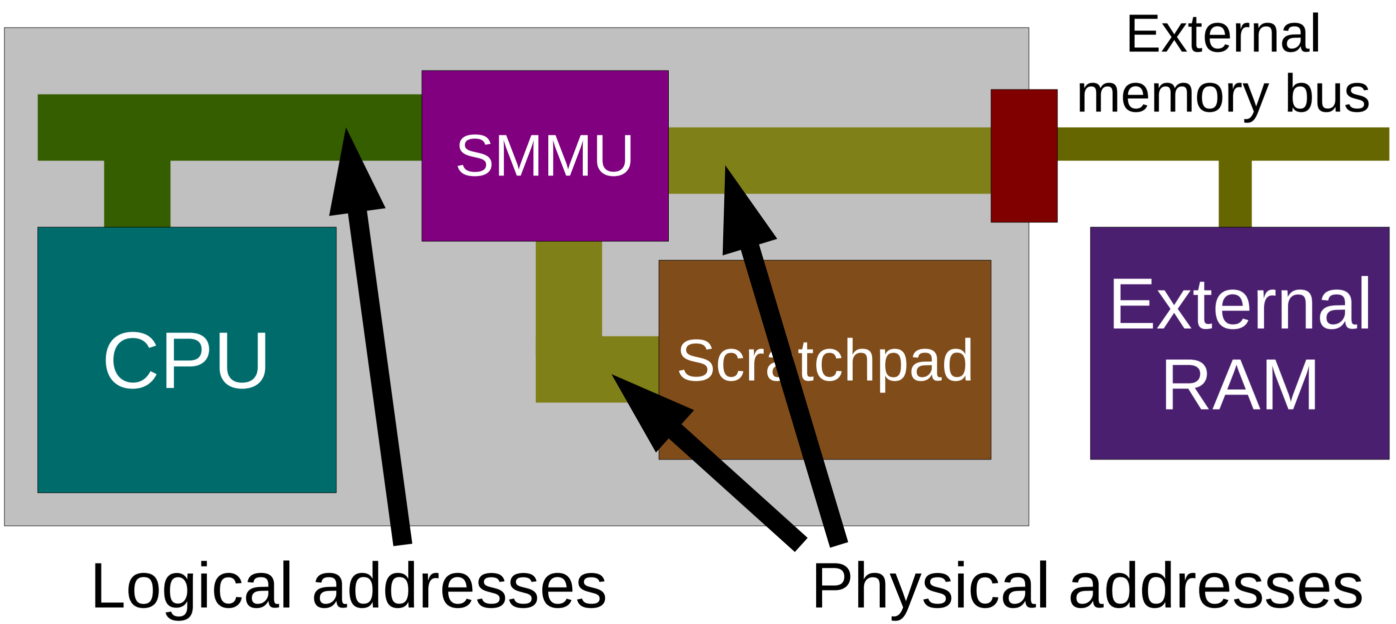
A *replacement* for a cache with time-predictable behaviour that is independent of *base address* and *input data*.

And... the replacement must guarantee that data is in scratchpad.

And... logical addresses must not change.

SMMU

Scratchpad Memory Management Unit.



Data can be *relocated* from external RAM to scratchpad without changing its logical address.

SMMU versus Scratchpad

Programs explicitly copy data from external memory to scratchpad and vice versa.

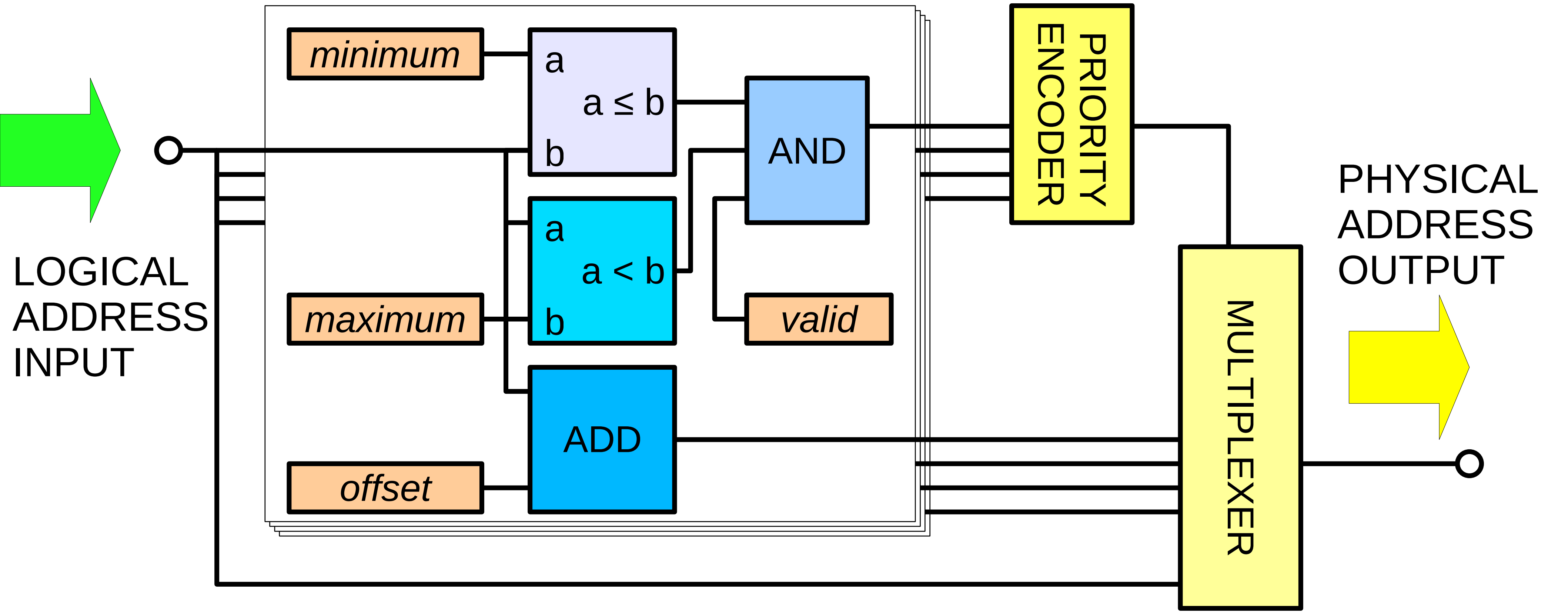
The logical address *does not change,* i.e.:

⇒ Pointers are never invalidated.

⇒ Pointer aliasing is handled correctly.

⇒ Scratchpad allocation algorithms can consider pointers rather than the objects they reference.

Inside the SMMU (1)

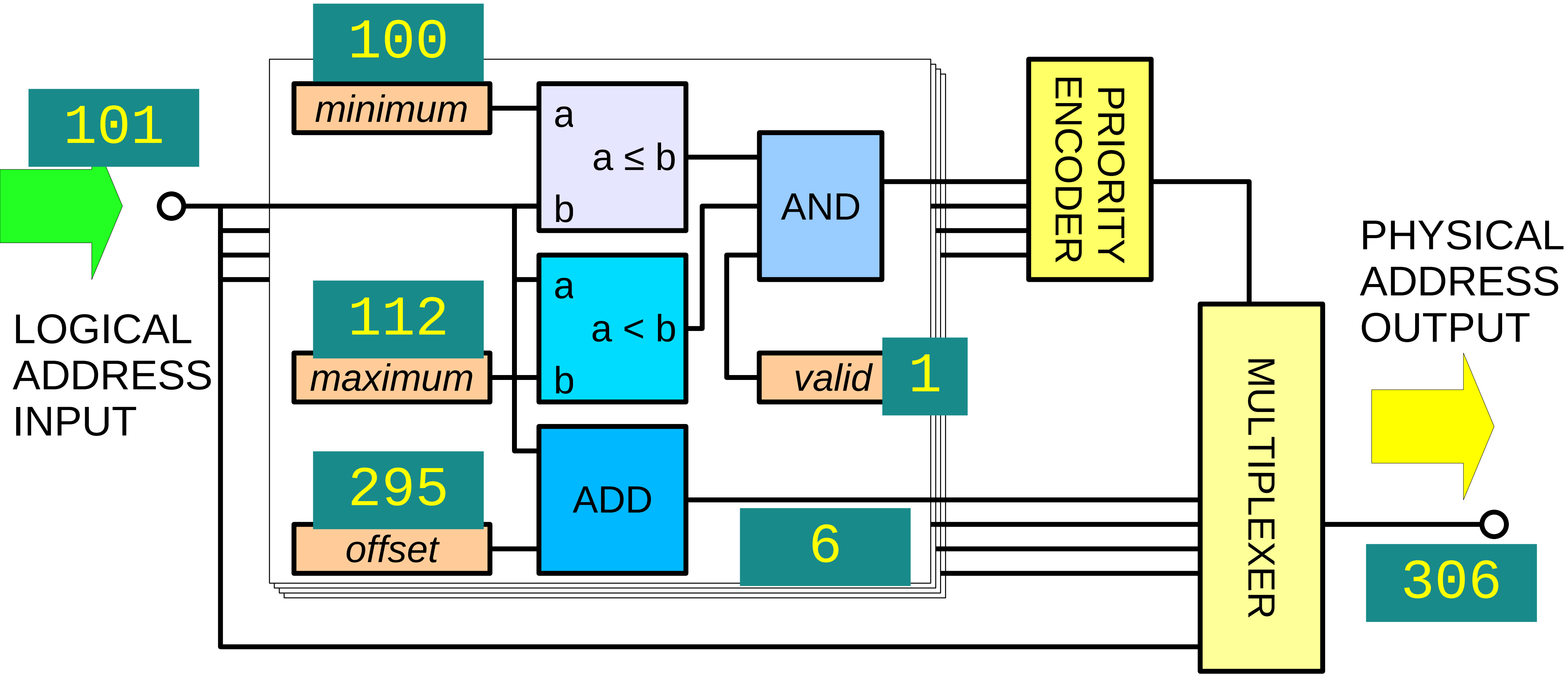


External memory @ [0:199]

Scratchpad memory @ [300:349]

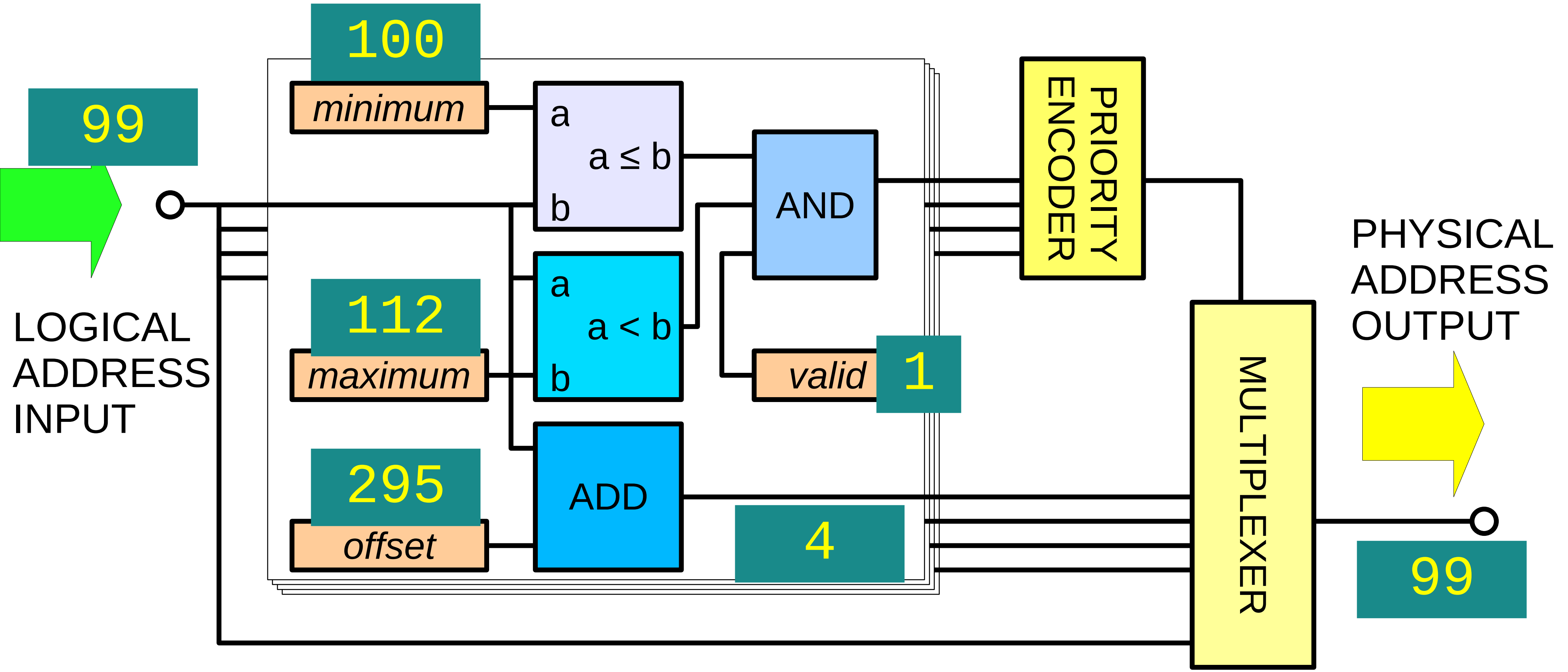
Copy the object in external memory
@ [100:112] to scratchpad @ [305, 317]

Inside the SMMU (2)



Logical address 101 matches in SMMU.

Inside the SMMU (3)



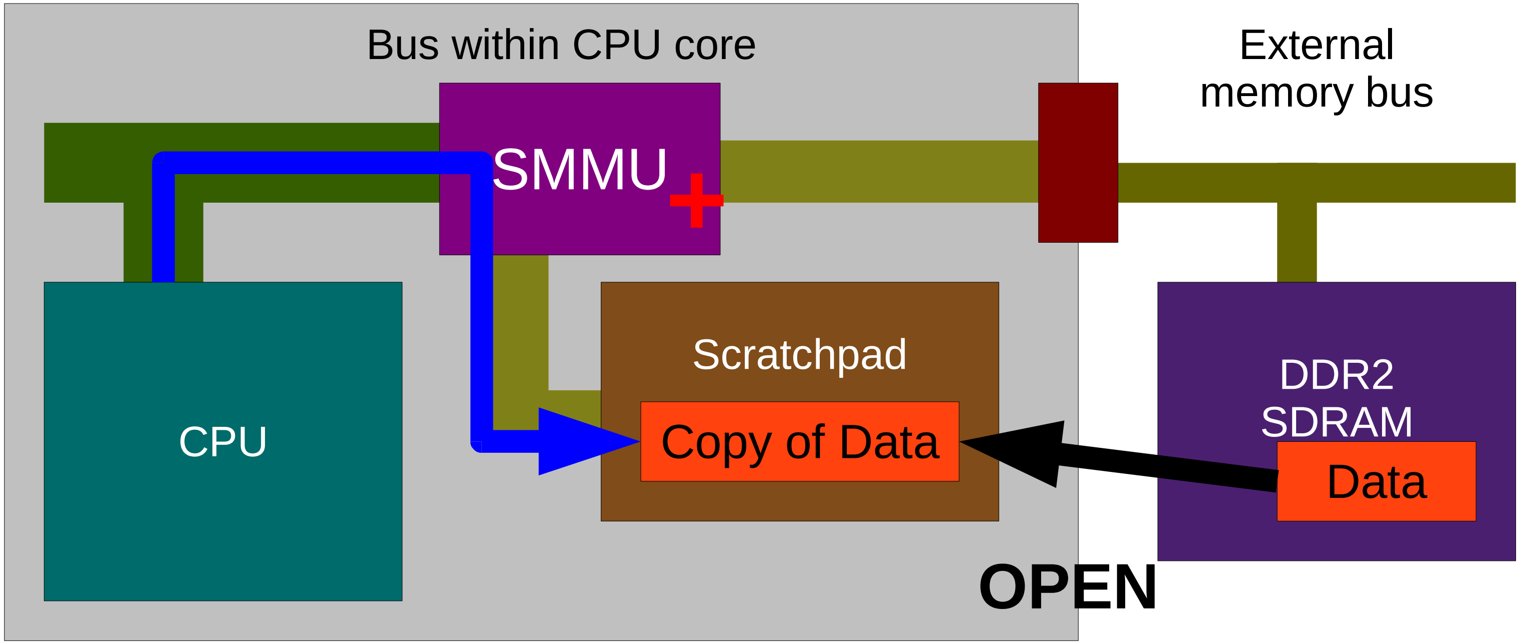
Logical address 99 does *not* match in SMMU.

Inside the SMMU (4)

Two further operations are implemented.

OPEN

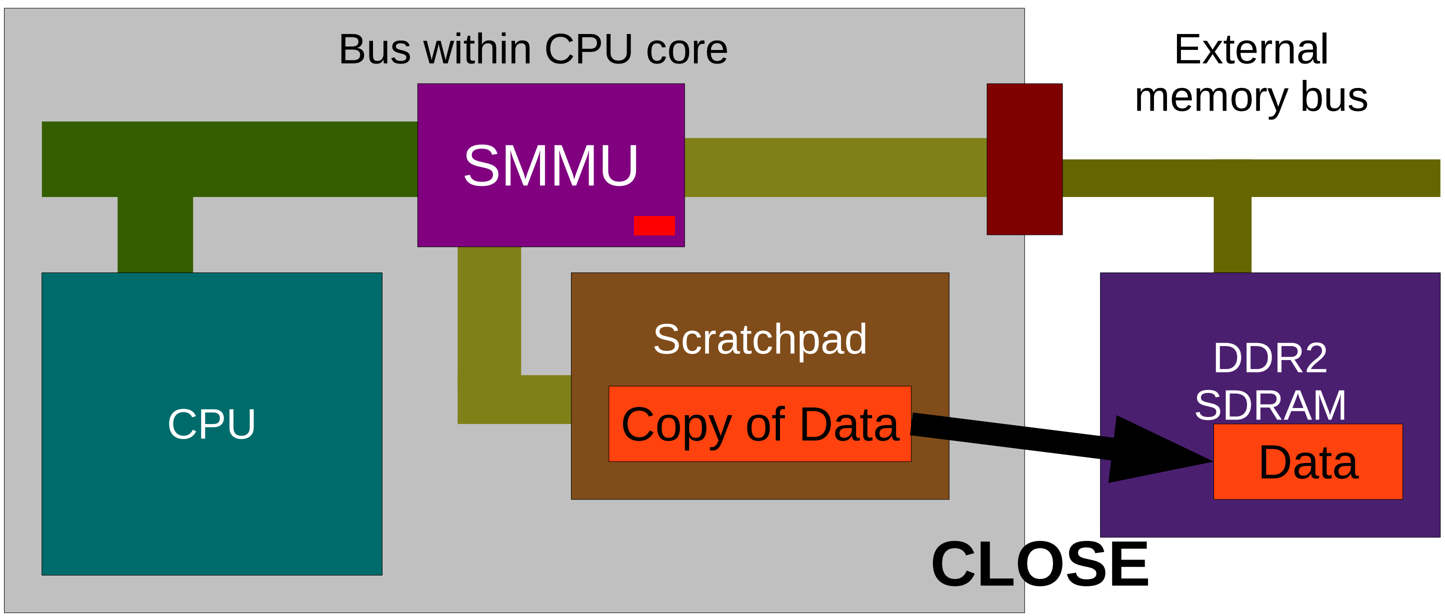
Copy data from external memory to scratchpad *and* add logical to physical address mapping.



Inside the SMMU (5)

CLOSE

Copy data from scratchpad to external memory *and* delete logical to physical address mapping.



Inside the SMMU (6)

What if data areas overlap?

$\Rightarrow$ Do OPEN and CLOSE still work correctly?

$\Rightarrow$ Always?

How to use the SMMU

Ideally:

⇒ An algorithm modifies your program to add OPEN and CLOSE operations as appropriate.

⇒ The algorithm allocates scratchpad space.

Result: time-predictable memory operations using the scratchpad whenever possible.

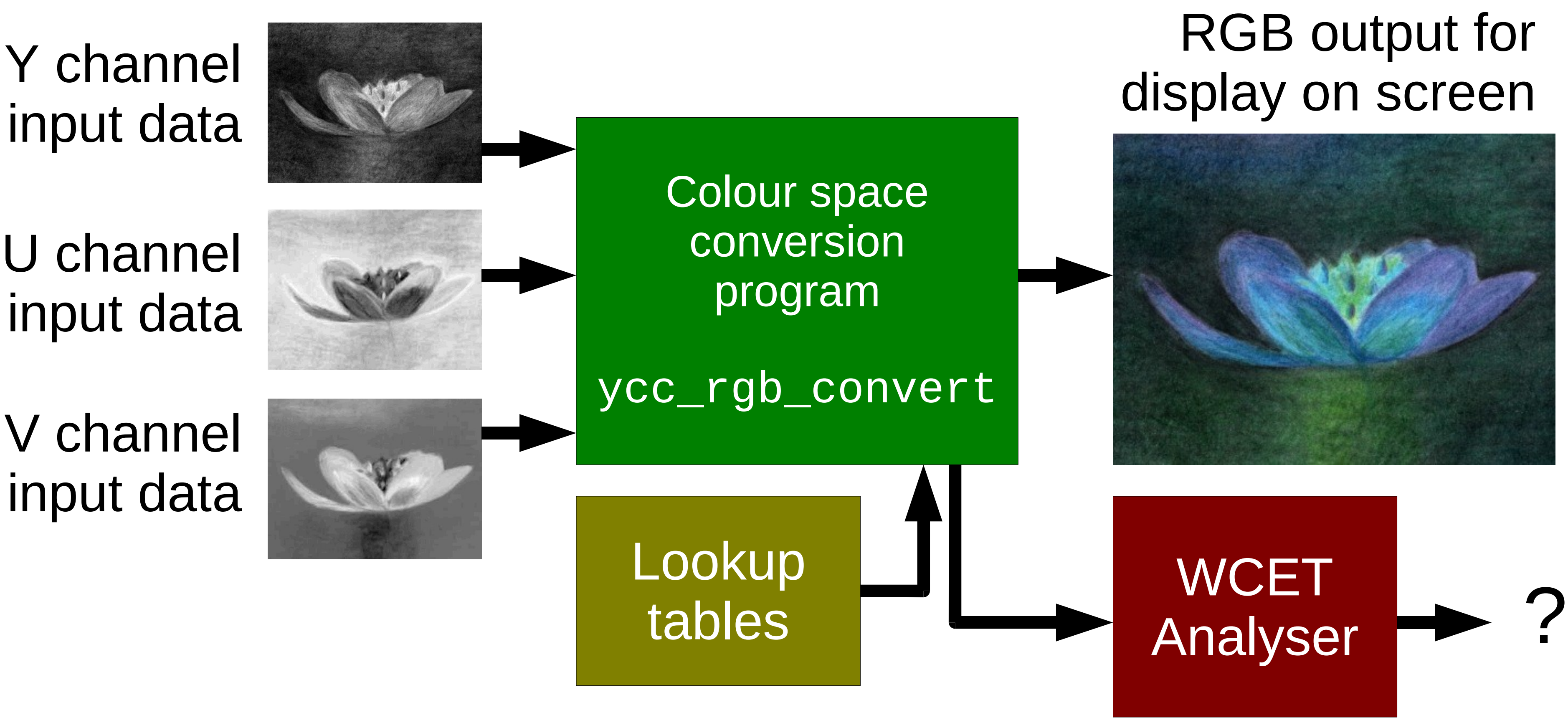
For the experiments in this paper:

⇒ The program was modified by hand.

⇒ Scratchpad space was allocated like a stack.

Back to the example

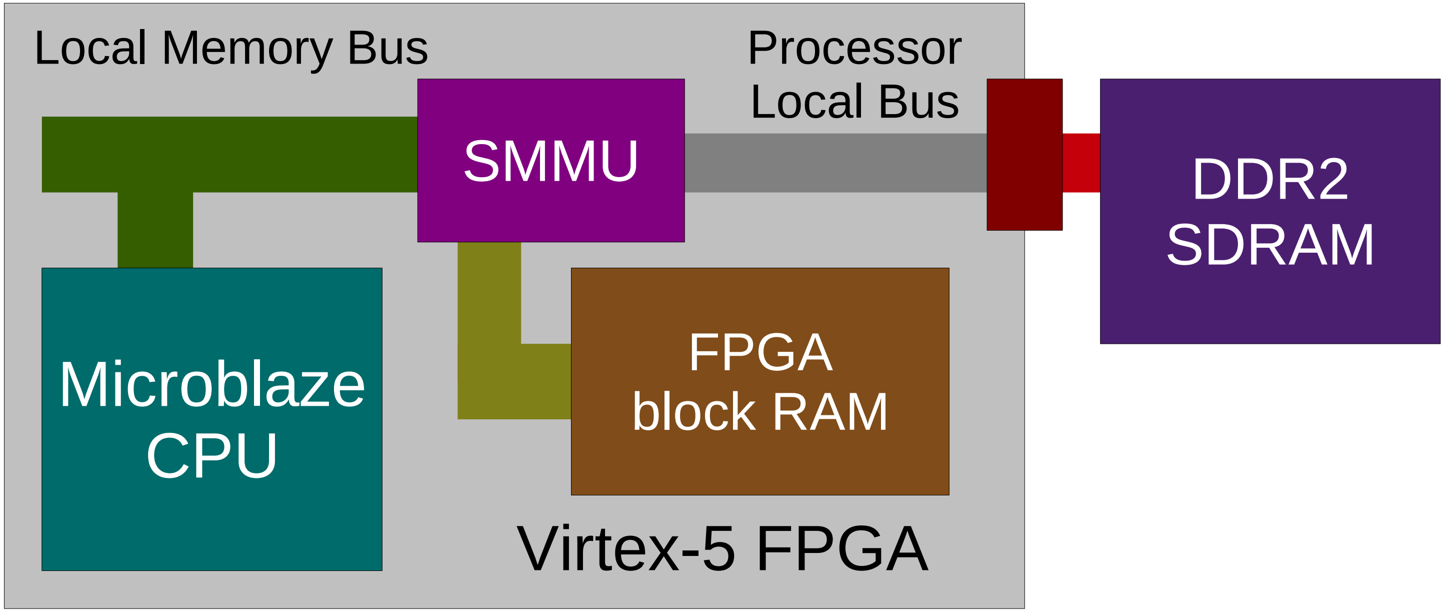
Nine dynamic pointers could be OPENed during `ycc_rgb_convert`.



```
range_limit_ref := OPEN(range_limit, SIZE(range_limit));

FOR row FROM 0 TO num_rows – 1 DO
    inptr0_ref := OPEN(inptr0, num_cols);
    inptr1_ref := OPEN(inptr1, num_cols);
    inptr2_ref := OPEN(inptr2, num_cols);

    FOR col FROM 0 TO num_cols - 1 DO
        y  := inptr0[col];
        cb := inptr1[col];
        cr := inptr2[col];
        outptr[0] := range_limit[y + Crrtab[cr]];
        outptr[1] := range_limit[y +
            ((Cbgtab[cb] + Crgtab[cr]) / 65536) ];
        outptr[2] := range_limit[y + Cbbtab[cb]];
        outptr := outptr + 3;
    END FOR;

    CLOSE(inptr0_ref);
    CLOSE(inptr1_ref);
    CLOSE(inptr2_ref);
END FOR;

CLOSE(range_limit_ref);
```

SMMU for Microblaze (1)

Microblaze: soft CPU core for Xilinx FPGAs.



SMMU implemented using VHDL [38].

SMMU for Microblaze (2)

On this platform:

⟹ Accessing *n* words of external memory takes

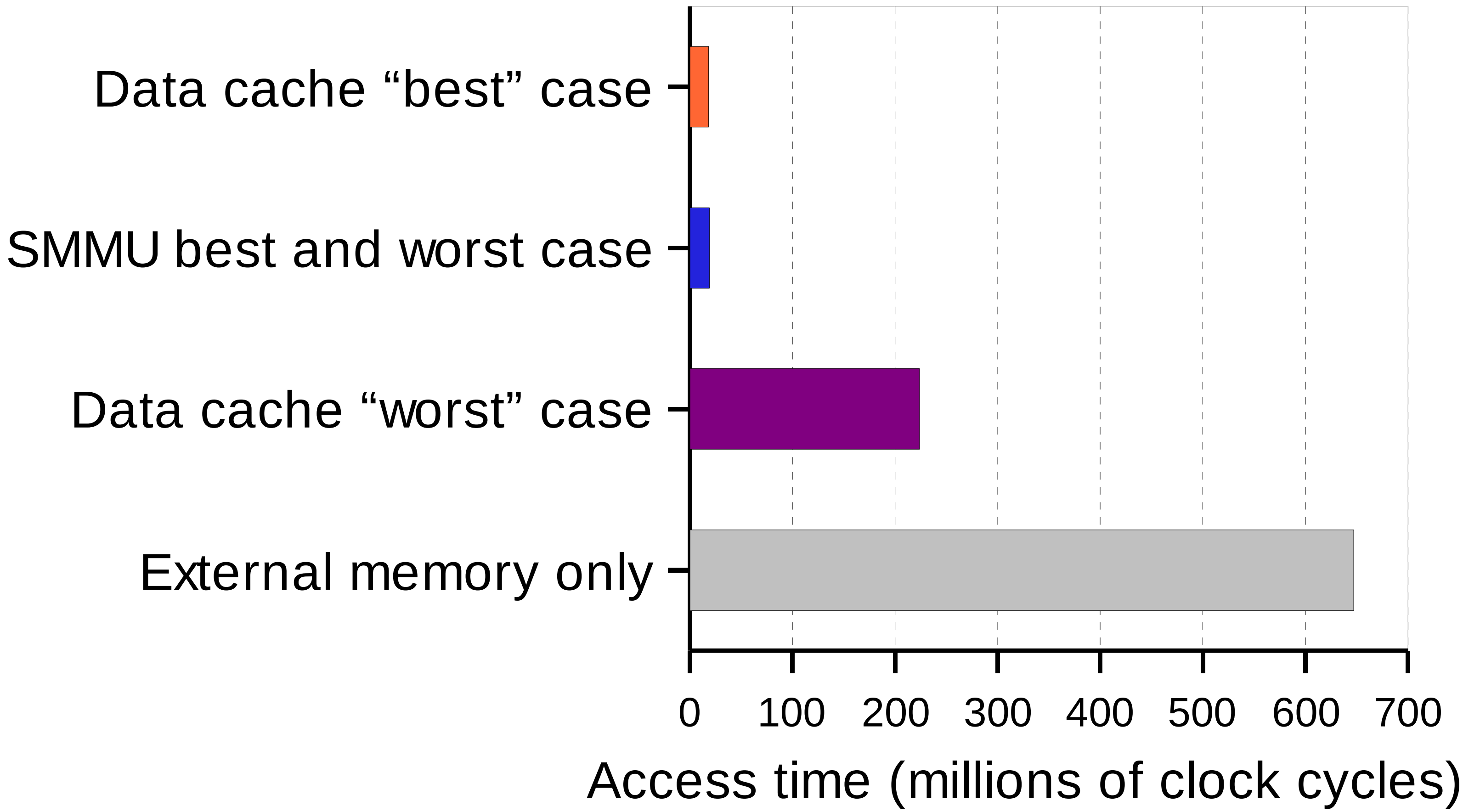
$$C(n) = 31 + n/4$$

clock cycles.

⟹ OPEN and CLOSE are implemented using memory mapped registers.

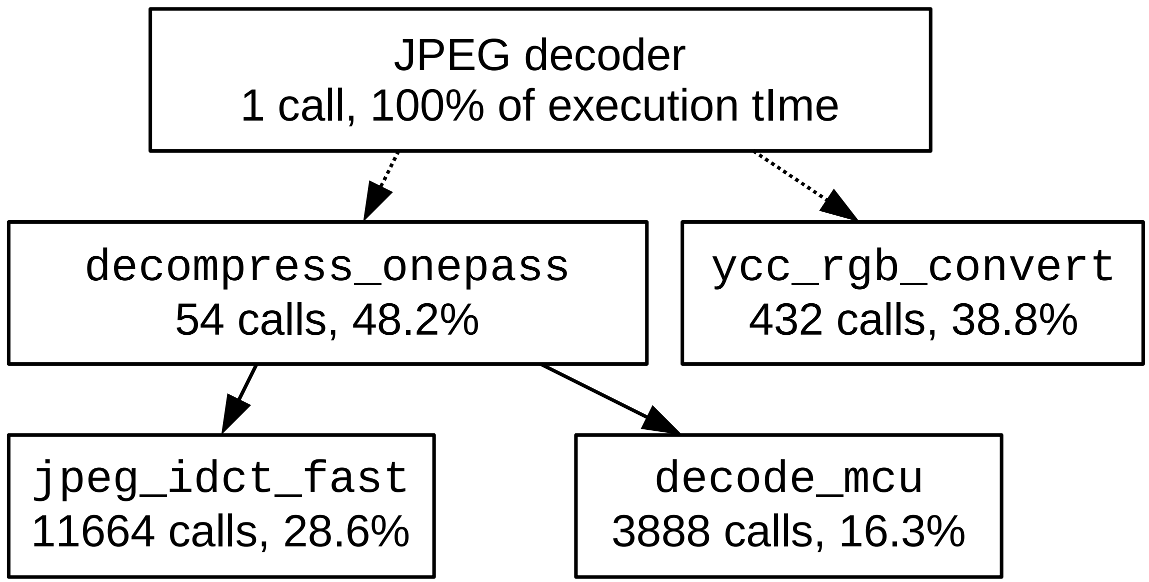
Results (1)

Using the `ycc_rgb_convert` function on a simulated platform:



Results (2)

Throughout the JPEG decoding process...



⇒ 90% of memory accesses are routed to scratchpad via the SMMU...

⇒ even though 75% of all memory accesses use dynamic data structures.

Results (3)

The remaining 10% of memory accesses consume 61% of the execution time.

Consequently, the program's execution time is 2.7 times slower than a perfect data cache.

However:

$\Rightarrow$ This is *much* better than external memory alone.

$\Rightarrow$ This is both the best and worst case!

Conclusion

The reasons for replacing data caches have been explained, along with the limitations of scratchpads.

The SMMU has been proposed as a solution.

It has been applied to a case study.

It has been implemented in hardware.

Thankyou

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