Predictable Programming on a Precision Timed Architecture

Ben Lickly, Isaac Liu, Hiren Patel, Edward Lee, University of California, Berkeley

Sungjun Kim, Stephen Edwards, Columbia University, New York

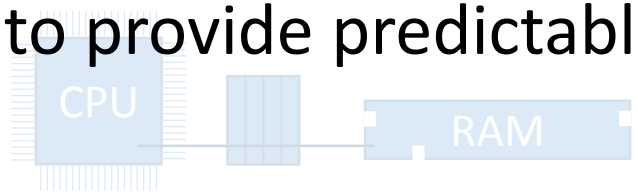
Presented By

Ashutosh Dhekne

PhD Student, University of Illinios at Urbana Champaign

Goal of the Paper

- Rethink processor architecture to provide predictable timing



Caching

Pipelined Execution

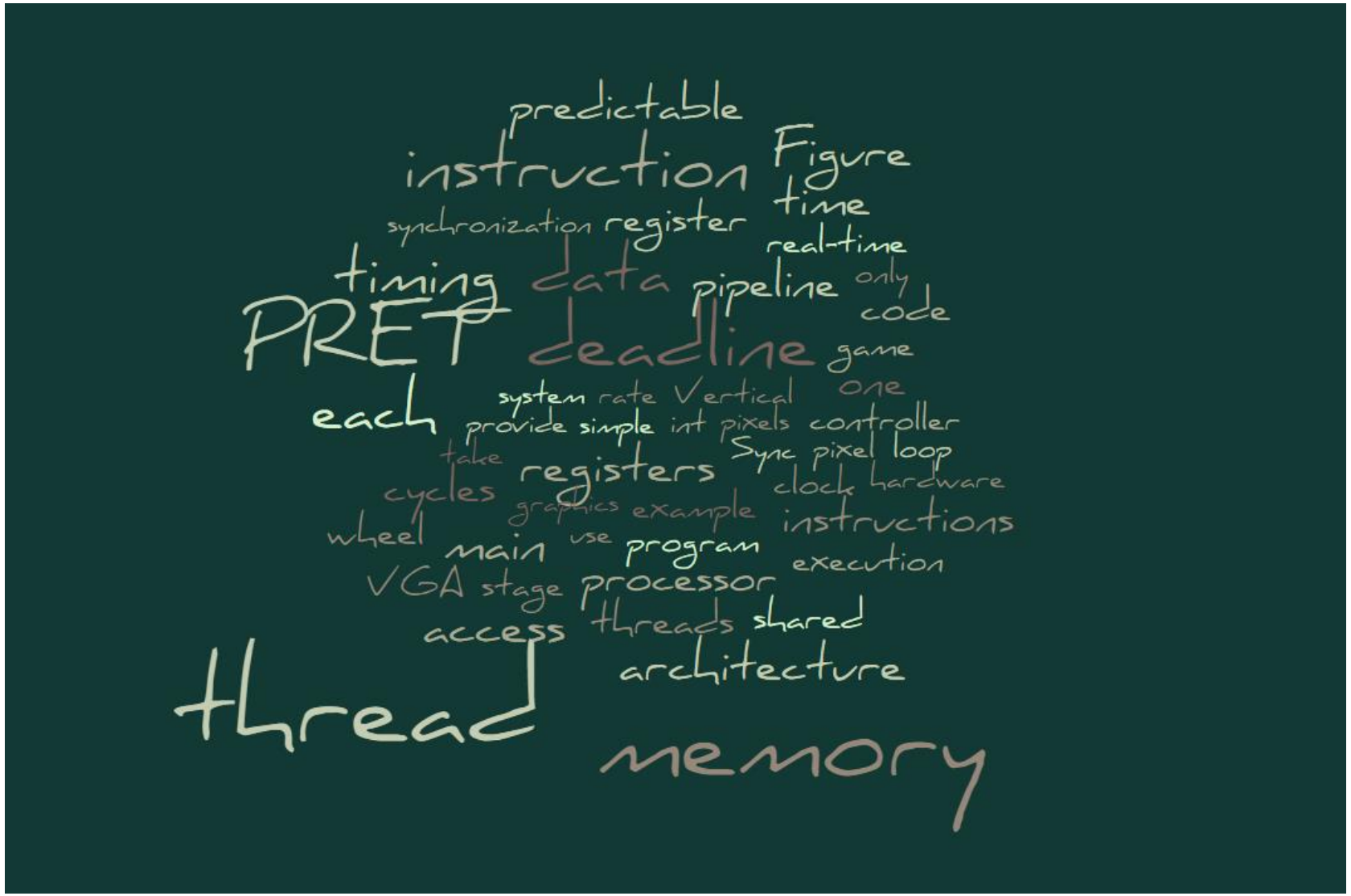
- Why such a stance?
  - Current computers optimized for average performance
  - Too many time saving tricks that complicate WCET analysis

- How to achieve it?
  - Exposed memory hierarchies
  - Thread interleaved pipelining
  - Deadline instructions

Virtual Memory

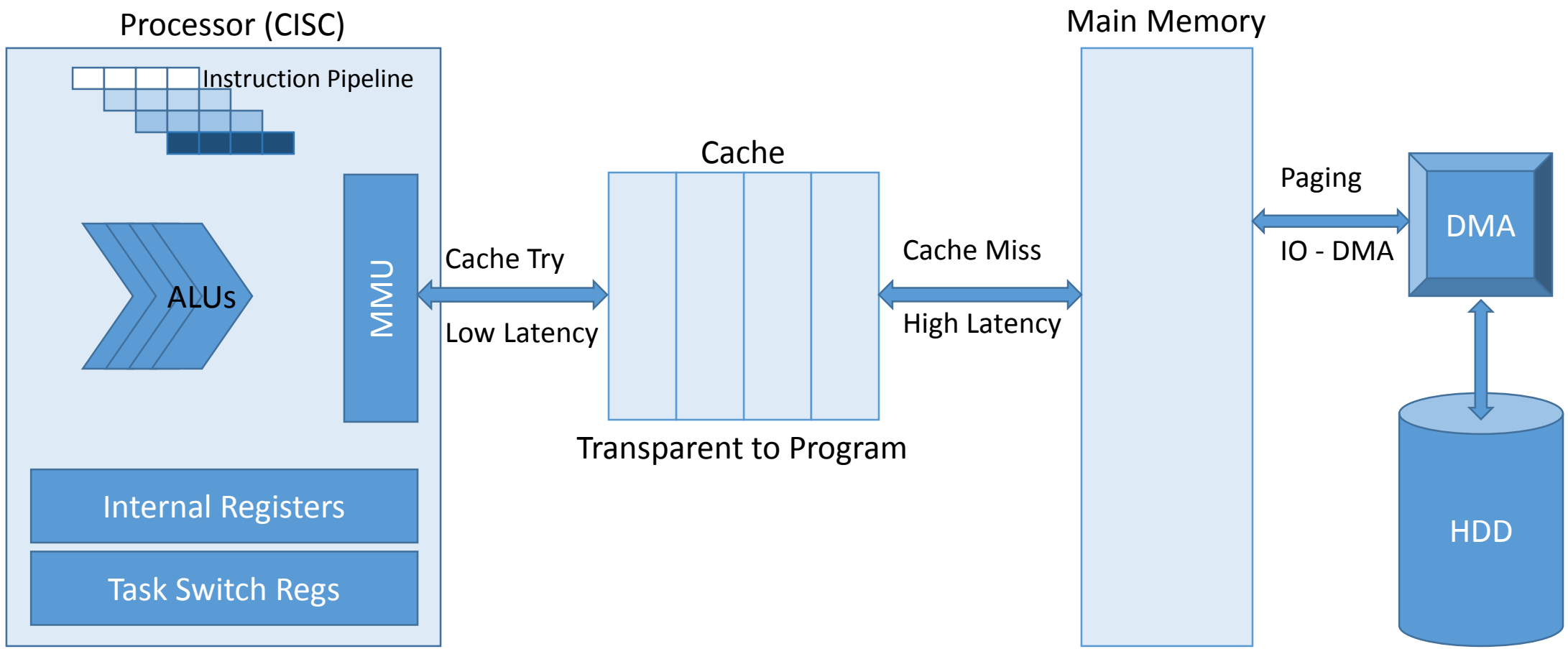
Frequency Scaling

Words that Stick



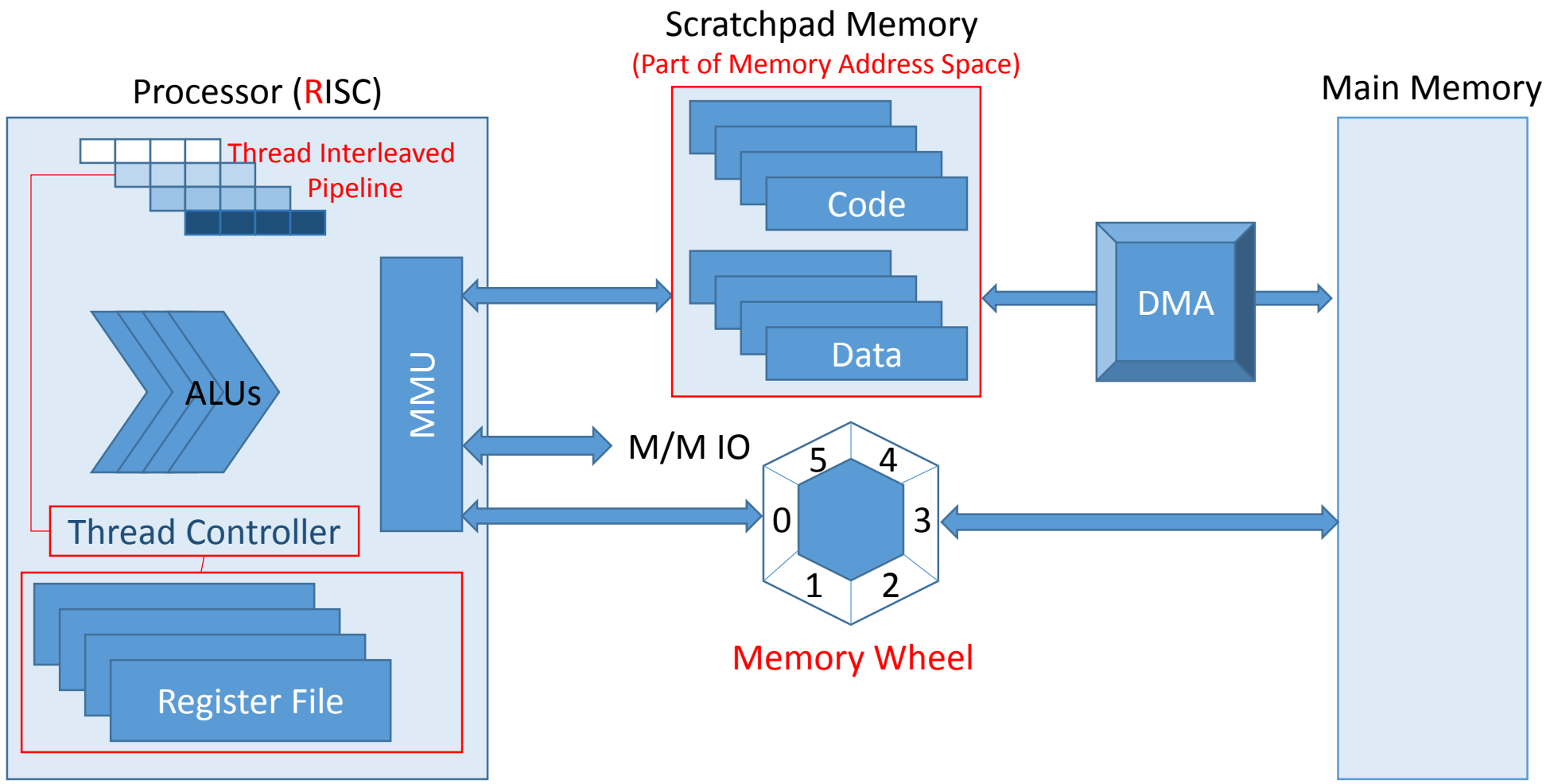
[link]

The Familiar Architecture (x86)

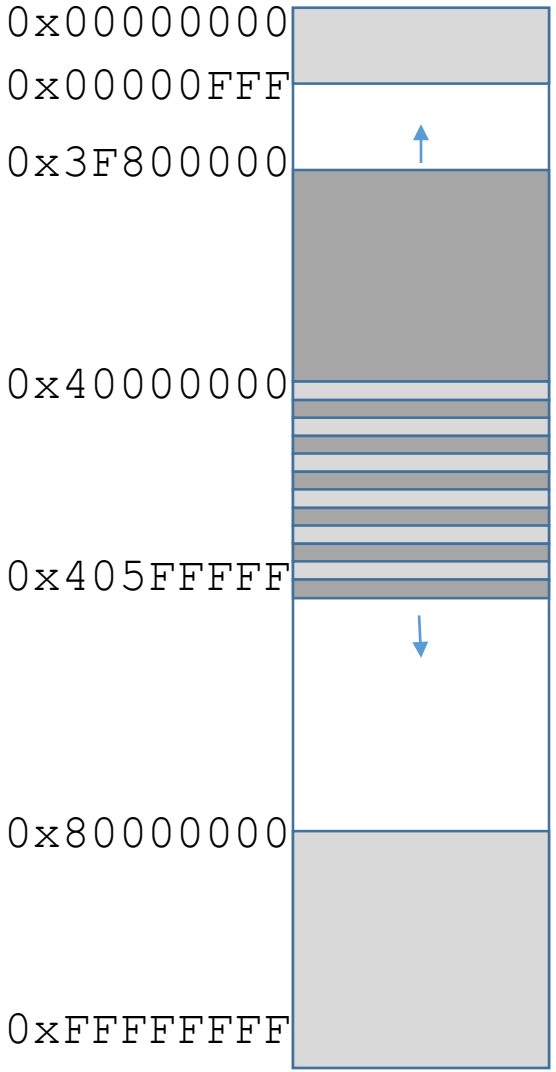


External Material – Drawn from memory

The PRET Architecture



Main Memory

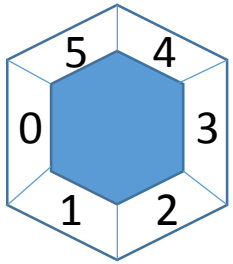


Boot code used by each thread on startup. Initializes all the registers

Shared Data 8MB between multiple threads

Thread local instructions and data (1MB per thread) 512KB for instruction, 512KB for data

Memory Mapped IO



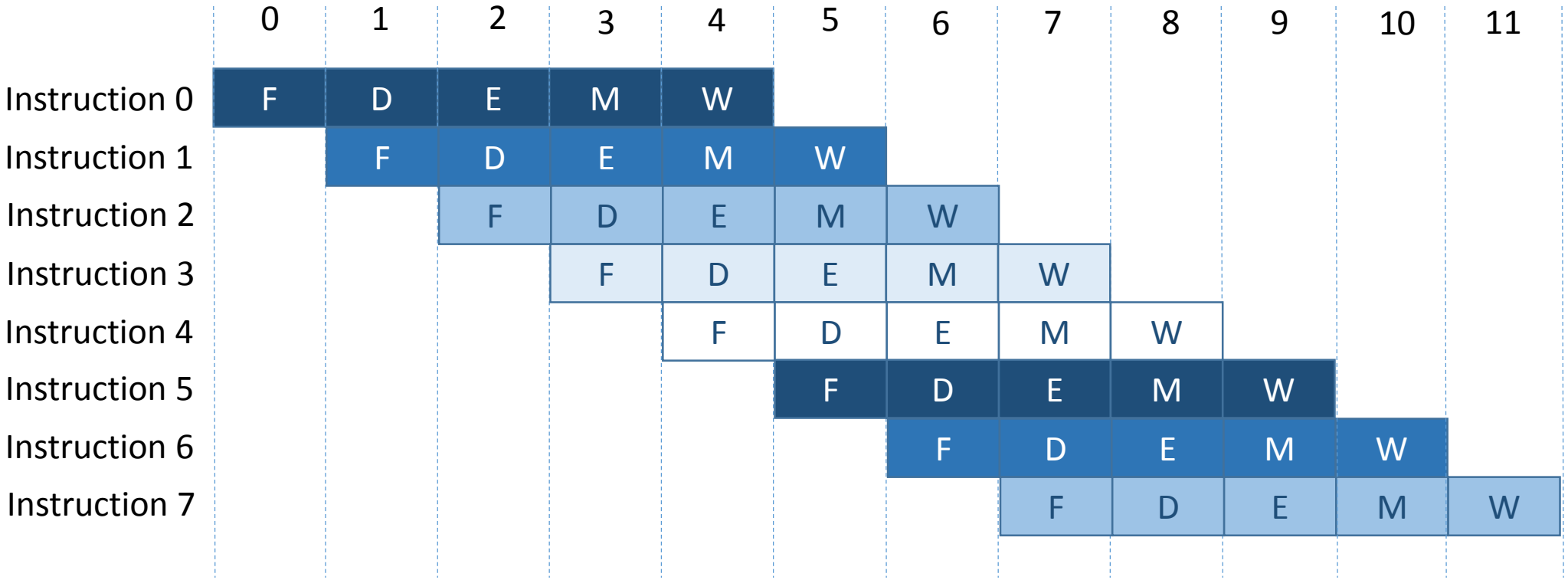
The Memory Wheel

I am feeling lucky!

- Access the Main Memory only through the Memory Wheel
- 13 cycle slotted time to access the Main Memory
- TDMA access creates false busy resource impression
- In the worst case, 90 cycles are required to access memory – bounded worst case

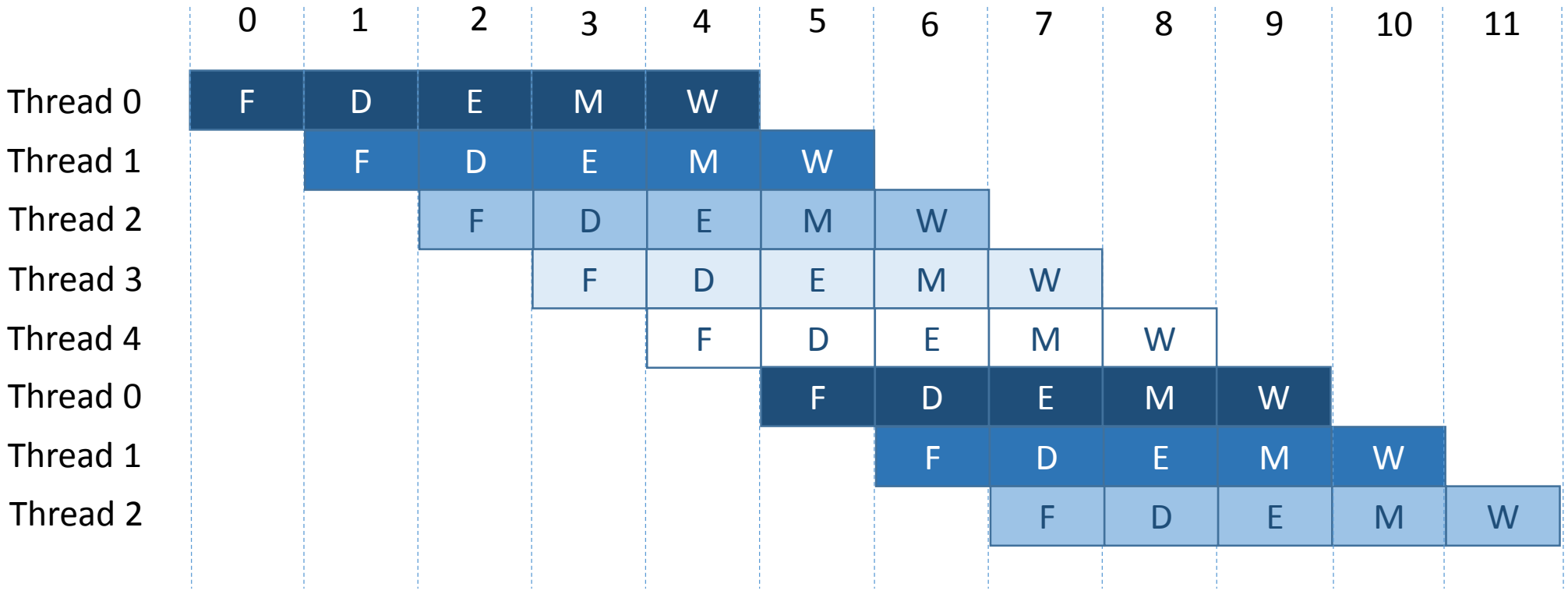
Instruction Pipelines

- Can we keep the pipeline always running?
- What about Data Hazards, Control Hazards, Structural Hazards?



Thread-Interleaved Pipelines

- What if we thread interleave pipelines, instead?
- Can we avoid all pipeline hazards?



Derived from: Precision Timed Machines, *Isaac Liu*

Hazardless Pipeline – Not Quite

- Can we ensure no hazards in thread interleaved pipelines?
- Always fill the pipelines with instructions from distinct threads
  - No explicit control dependencies between threads – No Control Hazard
  - Long latency instructions; prevent two from same thread – No Data Hazard
  - Very few concurrent threads; push in NOPs – No Data Hazard
  - Access to multi-cycle shared resources (eg. Memory) – Structural Hazard
  - TDMA access to the shared resources – removes timing dependencies
- Nonetheless, removing interdependence between pipeline units eases timing analysis

Derived from: Precision Timed Machines, *Isaac Liu*

Deadline Handling

Deadline miss

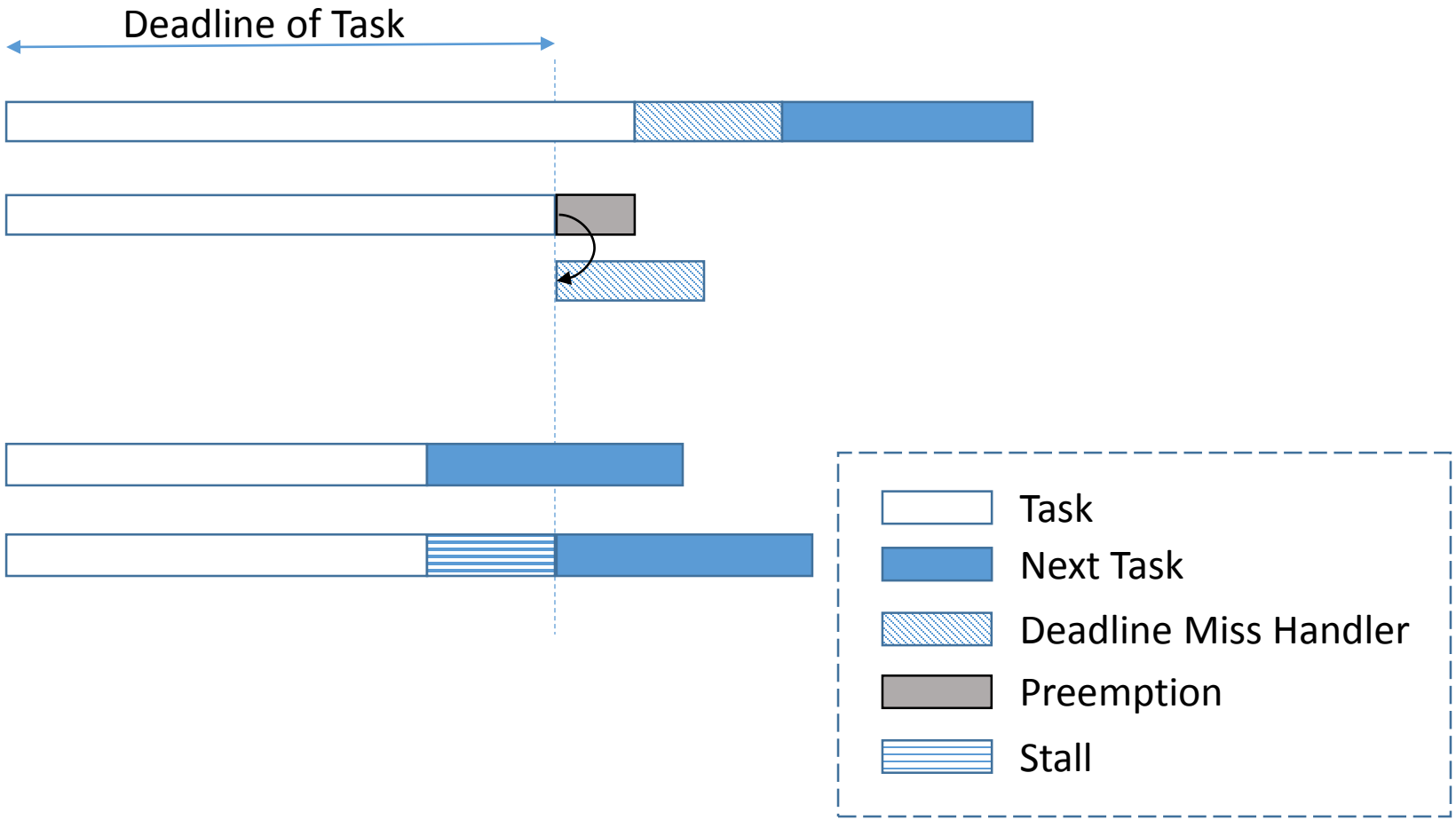
1A) Finish the task and detect at the end, if the deadline was missed

1B) Immediately handle a missed deadline

Deadline hit

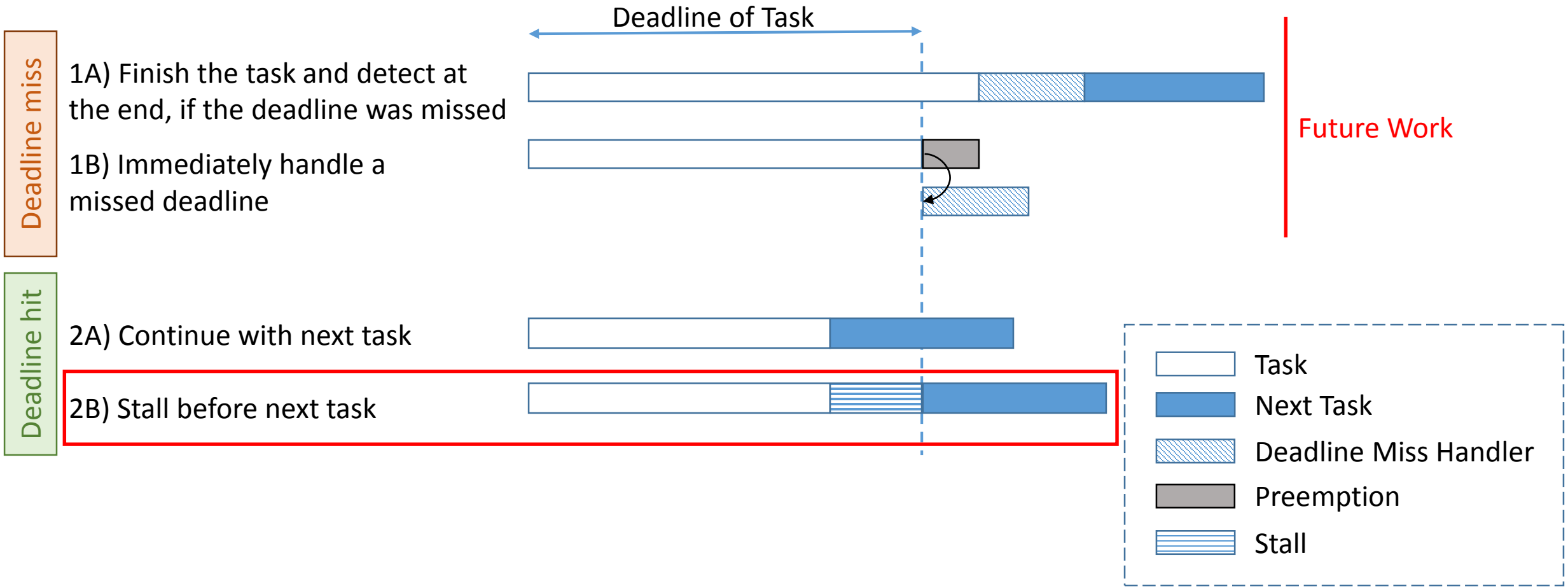
2A) Continue with next task

2B) Stall before next task



Derived from: Precision Timed Machines, *Isaac Liu*

Deadline Handling



Derived from: Precision Timed Machines, *Isaac Liu*

The Deadline Instruction

- A per-thread Deadline Register t_i
- DEAD(x) blocks until t_i reaches zero
- It then loads the value x in the register and executes next instruction
- The paper does not handle missing deadlines

Producer

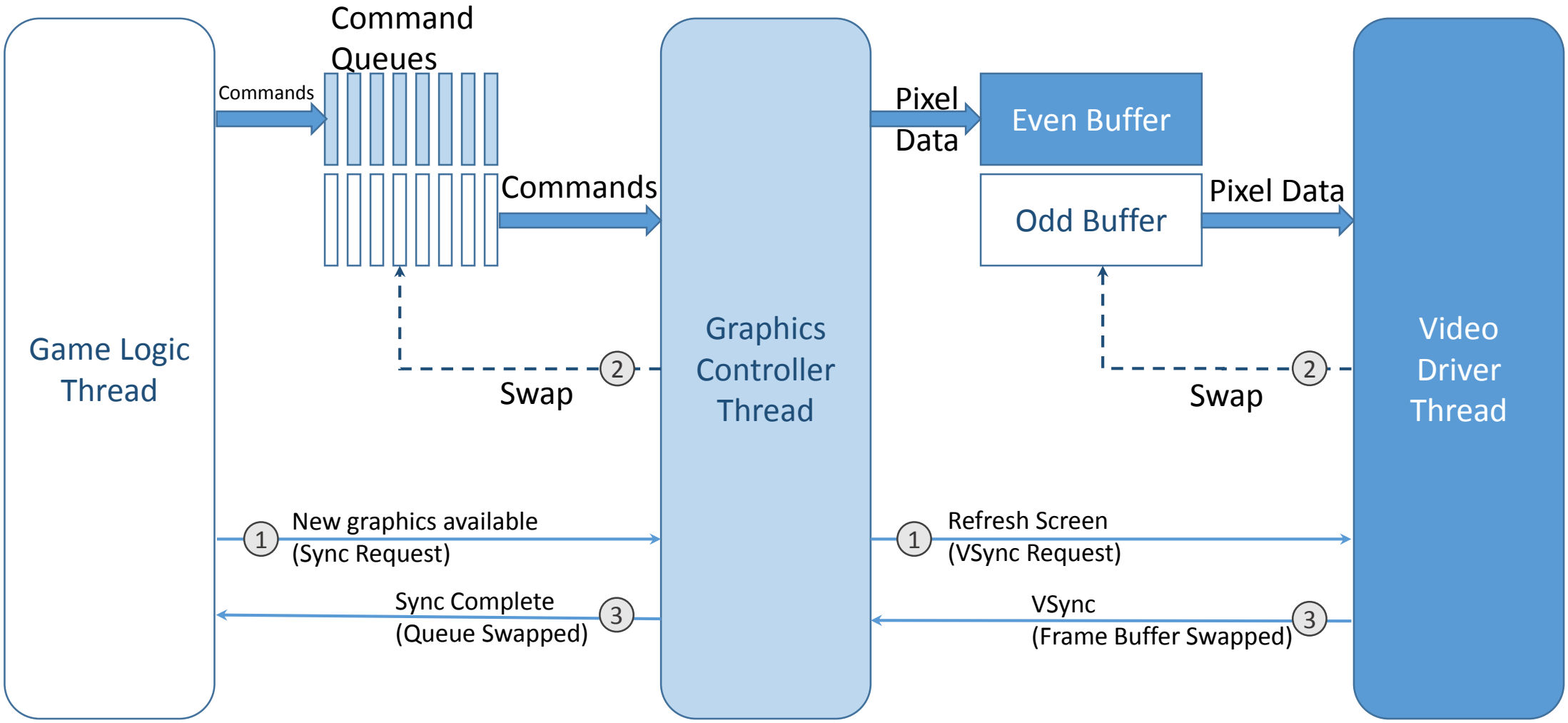
```
int main() {
    DEAD(28);
    volatile unsigned int *buf =
        (unsigned int*)
(0x3F800200);
    unsigned int i = 0;
    for (i=0; ; i++) {
        DEAD(26);
        *buf = i;
    }
    return 0;
}
```

Register t_i is loaded with value 28

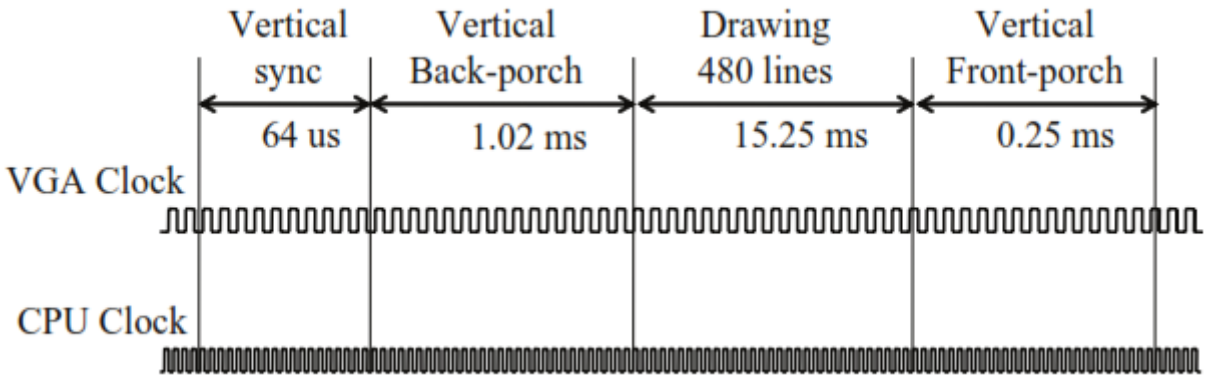
Program waits here until t_i becomes zero, then loads 26. If program returns here due to the loop, it might wait again.

- The deadline register is checked in the register access stage and replayed until it becomes zero

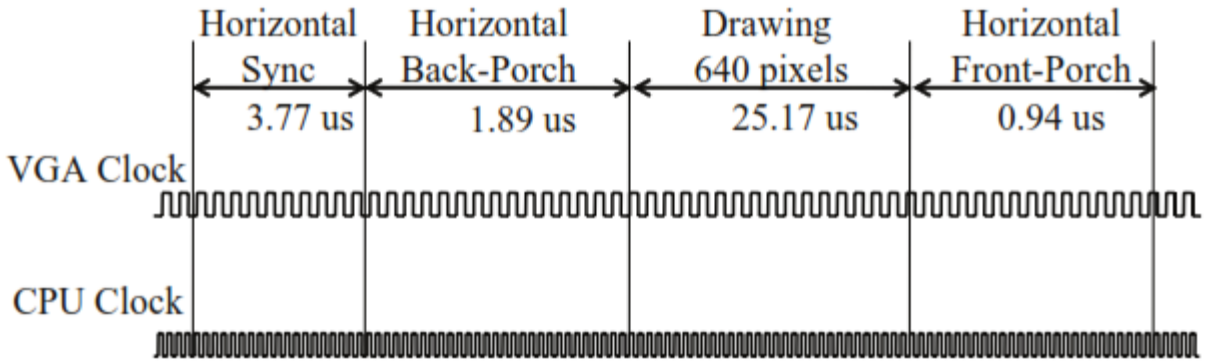
Example Game



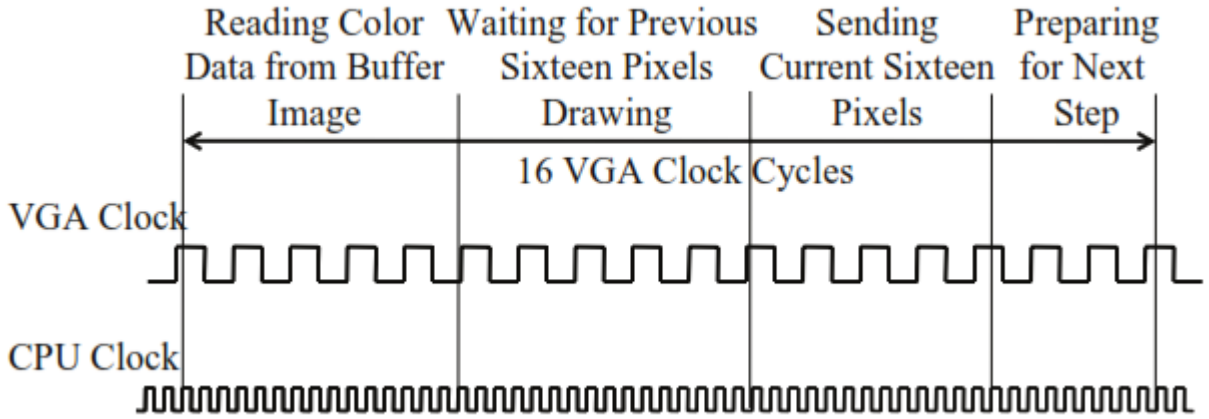
VGA Real-time Constraints



VGA Vsync Time



VGA Hsync Time



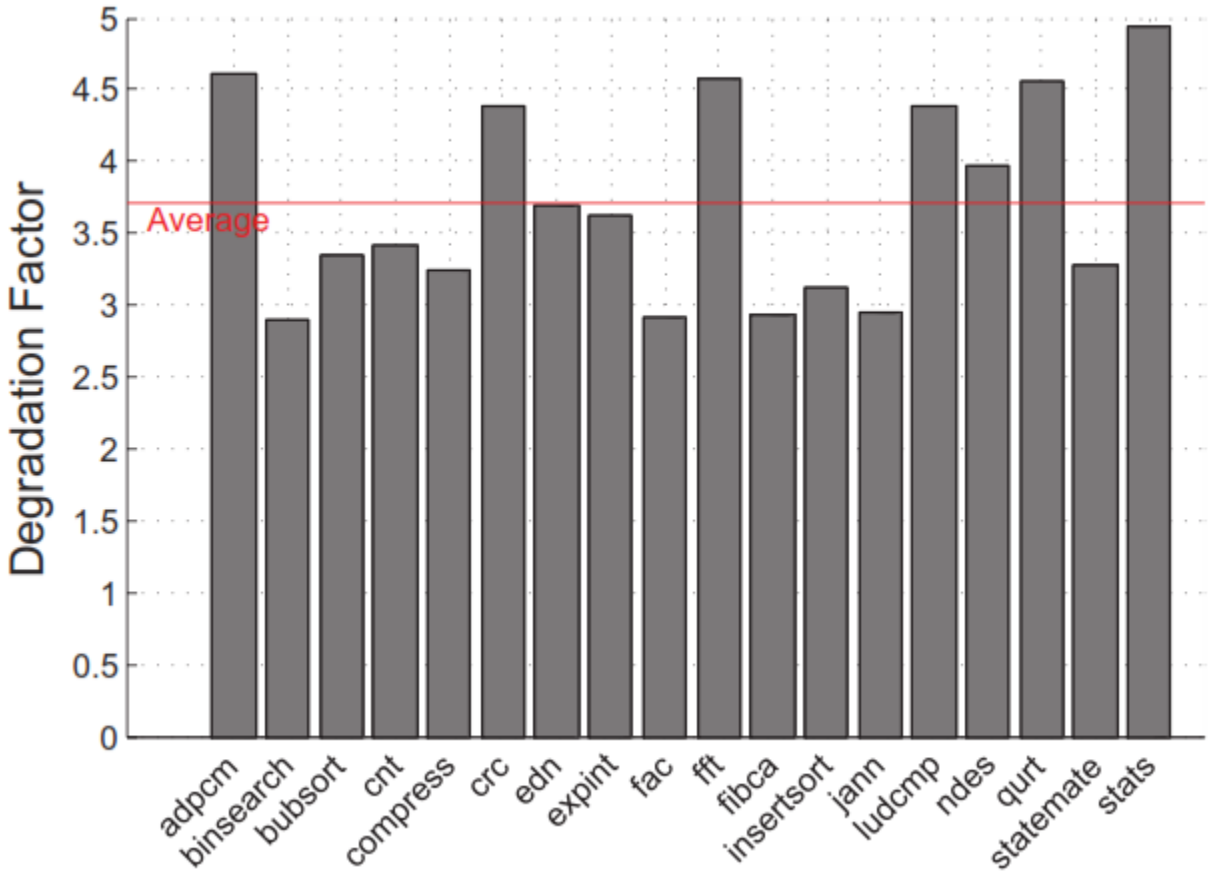
Sixteen Pixels at a time

Experiences from the Two Samples

- It is possible to provide timing guarantees using the PRET architecture
- But, timing calculations by hand are error-prone
- Automated tools will be provided in the future
- The underlying architecture lacks synchronization primitives
- Simple synchronization can be achieved using the deadline instructions

Comparison with the LEON3

- Average case time degradation is studied
- PRET shows significant degradation due to lack of parallel threads
- None of the special PRET features are used
- Degradation factor < 6; no pipeline hazard advantage?



Conclusions

- The paper builds a remarkable architecture using SystemC model
- It introduces new instruction for one type of deadlines
- PRET keeps memory hierarchy and time differences exposed to user
- The model runs actual C programs and a small game
- Somewhat unfair comparison between LEON3 and PRET at the end

It is possible to modify a RISC processor to have predictable timing

Some Observations

- With a project of this scale, it is difficult to fit all details in a paper
- I had to refer to one of the author's thesis work to gain insights
- The memory wheel assumes all threads will use memory equally
- I would suggest reduce the LEON3 comparison; include more fundamental insights instead
- Overall the work is commendable
- Provides some thoughts not discussed in any previous paper
- A true systems level work

Can off the shelf architectures provide a strict WCET mode?

Thanks!