A Hybrid Operating System Design Approach using Hardware Acceleration

Insop Song
Ericsson

Real Time Linux Workshop 2012
UNC Chapel Hill, NC

Hardware OS

- less overhead
- more deterministic

Hardware OS

- less overhead
- more deterministic

Why Hardware OS NOT seen much?

Hardware OS

- less overhead
- more deterministic

Why Hardware OS NOT seen much?

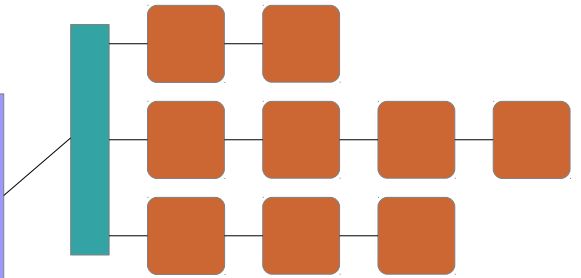
Hybrid OS

OS: Process/Timer/Event handling

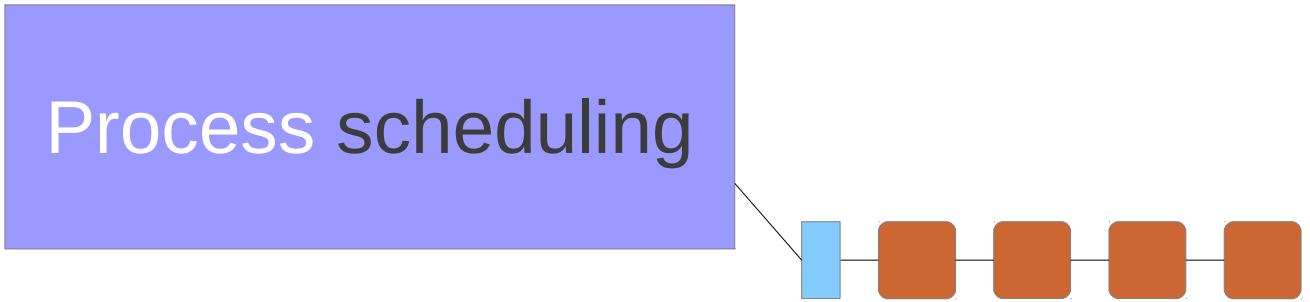
Process scheduling

OS: Process/Timer/Event handling

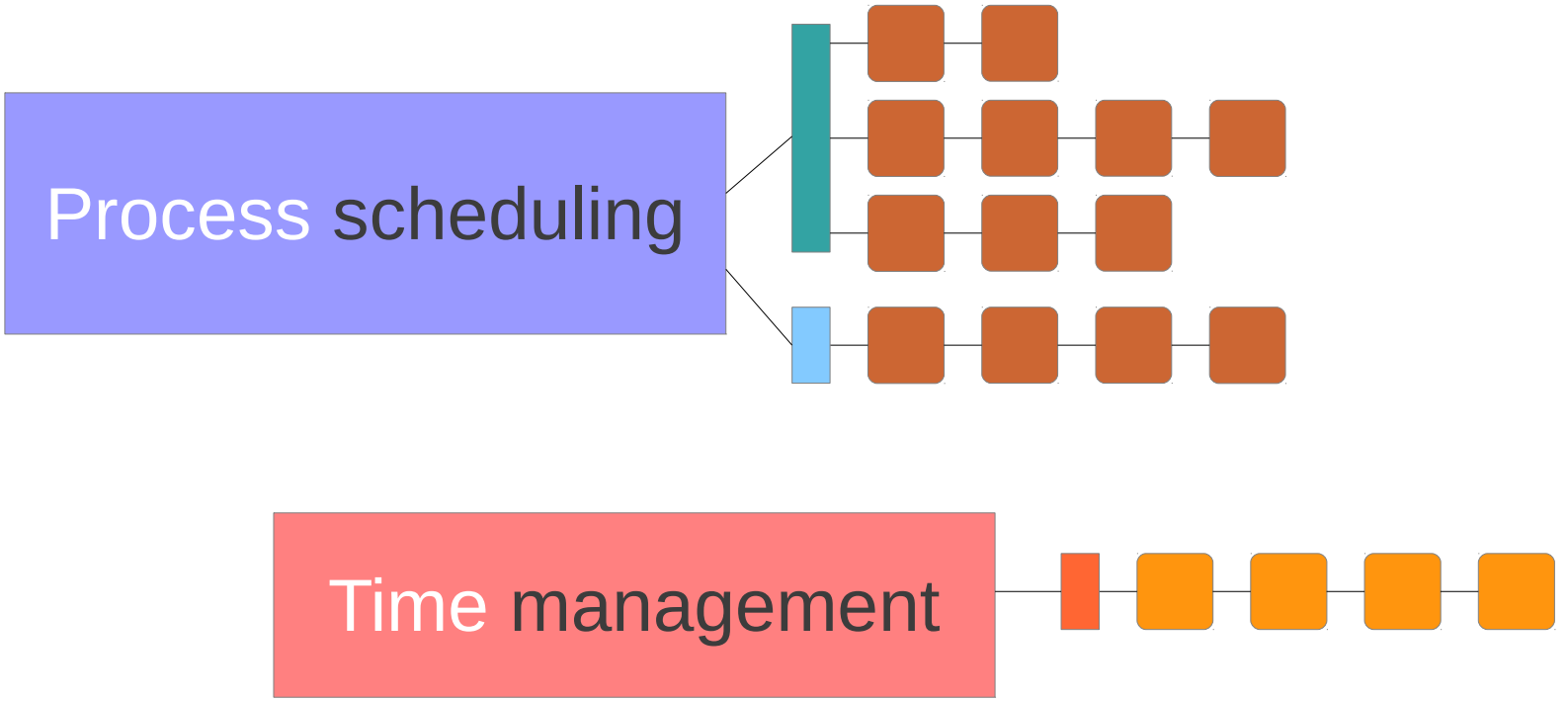
Process scheduling



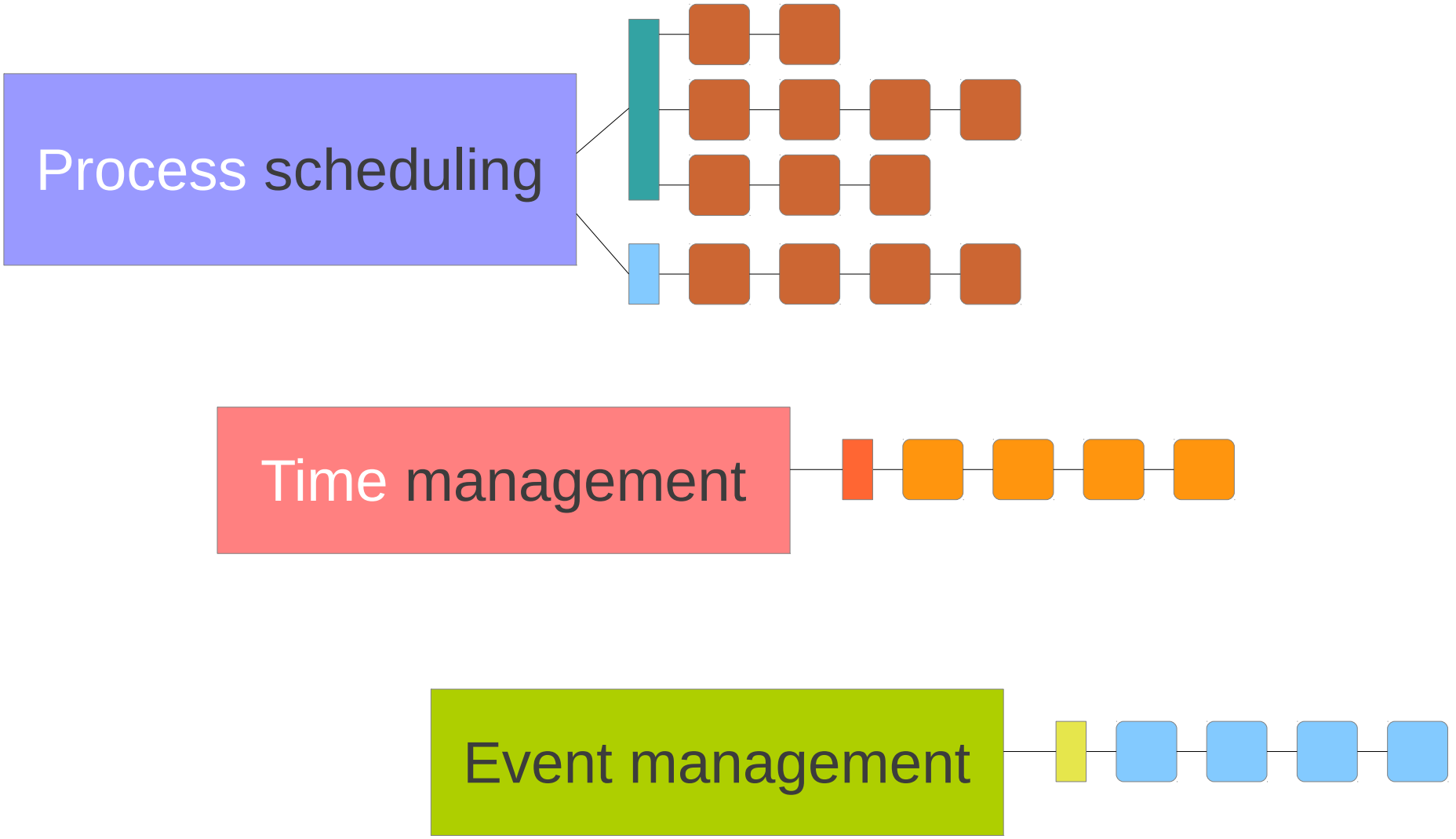
OS: Process/Timer/Event handling



OS: Process/Timer/Event handling



OS: Process/Timer/Event handling

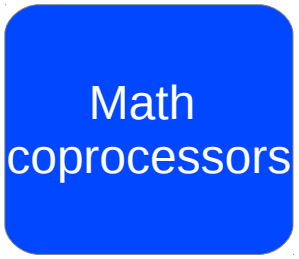


H/W accelerators & co-processors

- CPU bound
- Process intensive
- Real h/w concurrency

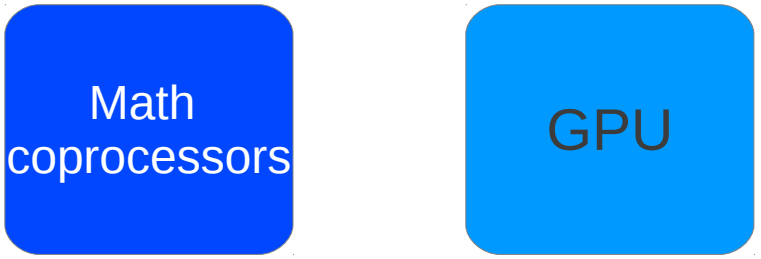
H/W accelerators & co-processors

- CPU bound
- Process intensive
- Real h/w concurrency



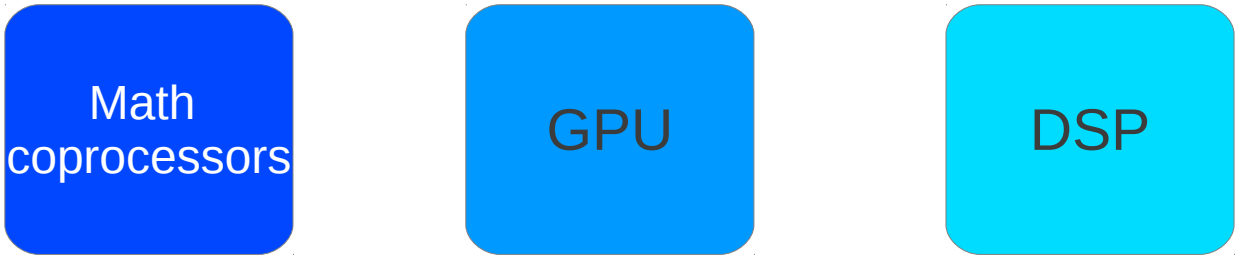
H/W accelerators & co-processors

- CPU bound
- Process intensive
- Real h/w concurrency



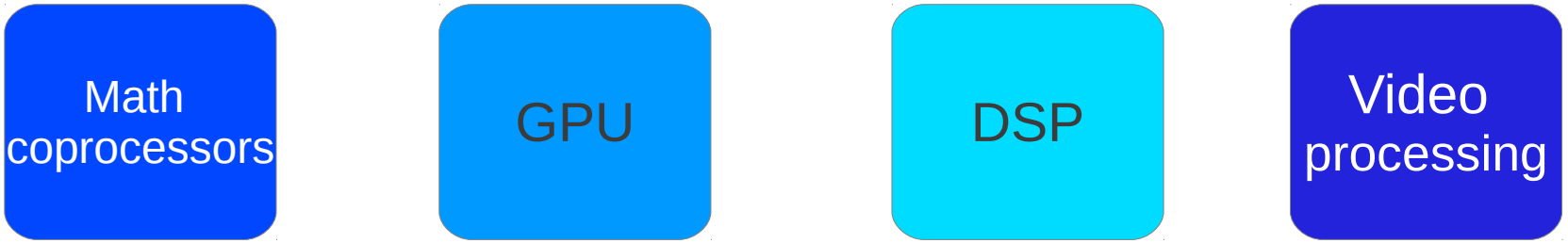
H/W accelerators & co-processors

- CPU bound
- Process intensive
- Real h/w concurrency



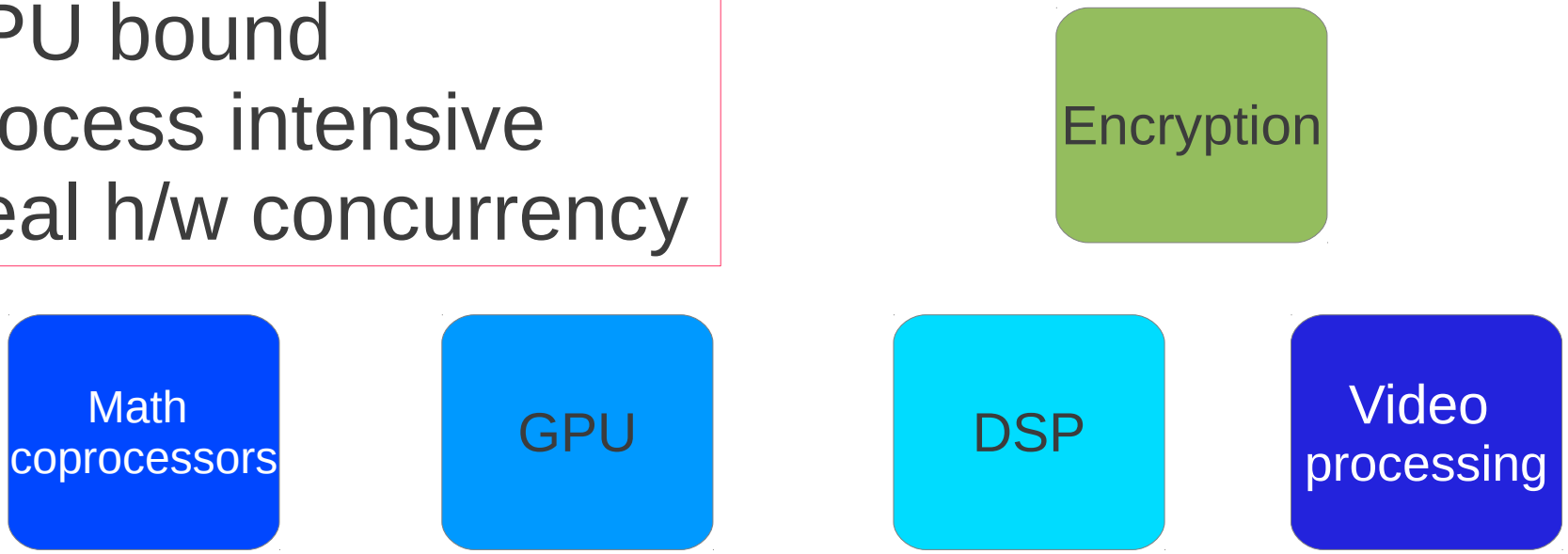
H/W accelerators & co-processors

- CPU bound
- Process intensive
- Real h/w concurrency



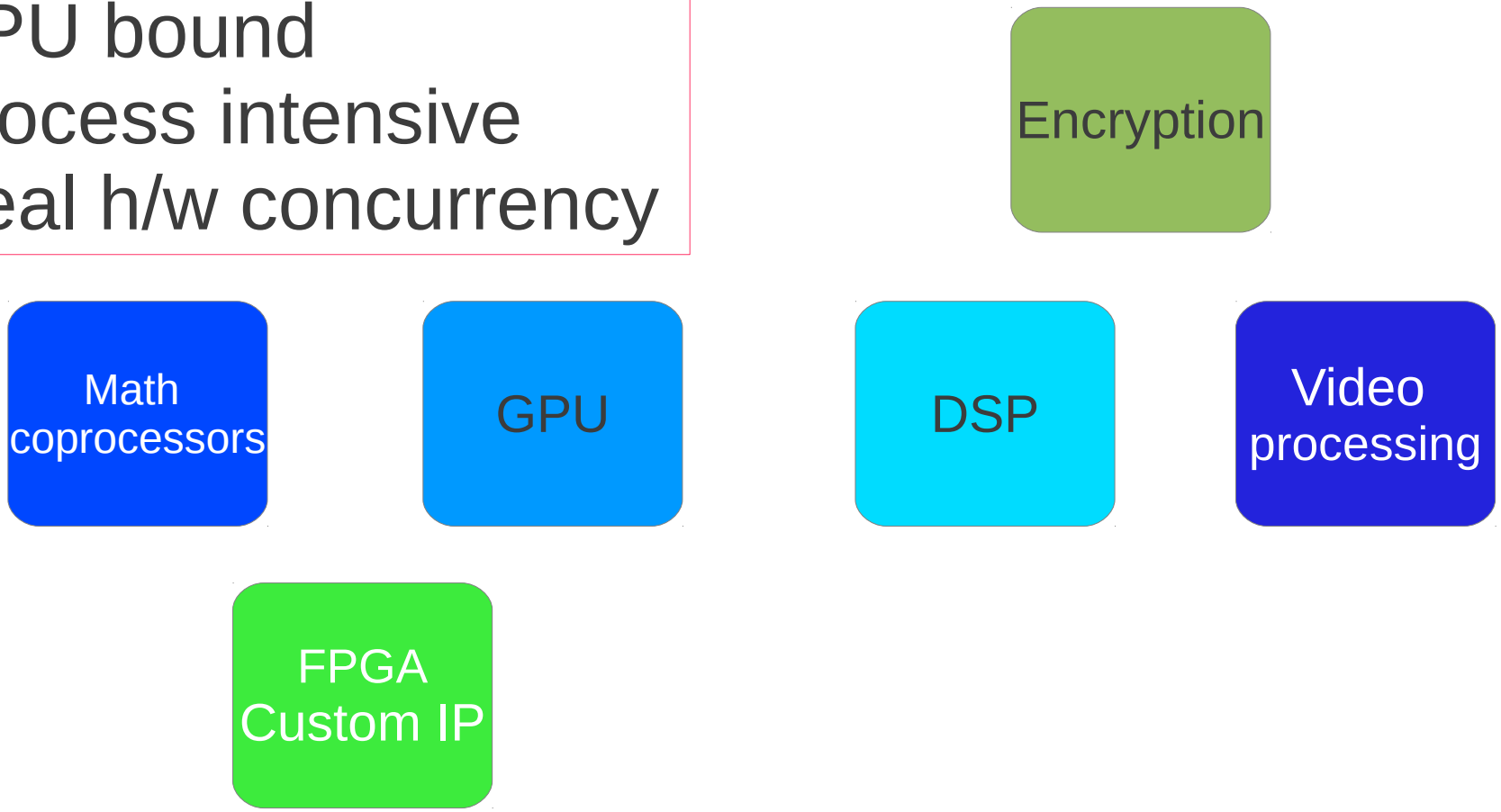
H/W accelerators & co-processors

- CPU bound
- Process intensive
- Real h/w concurrency



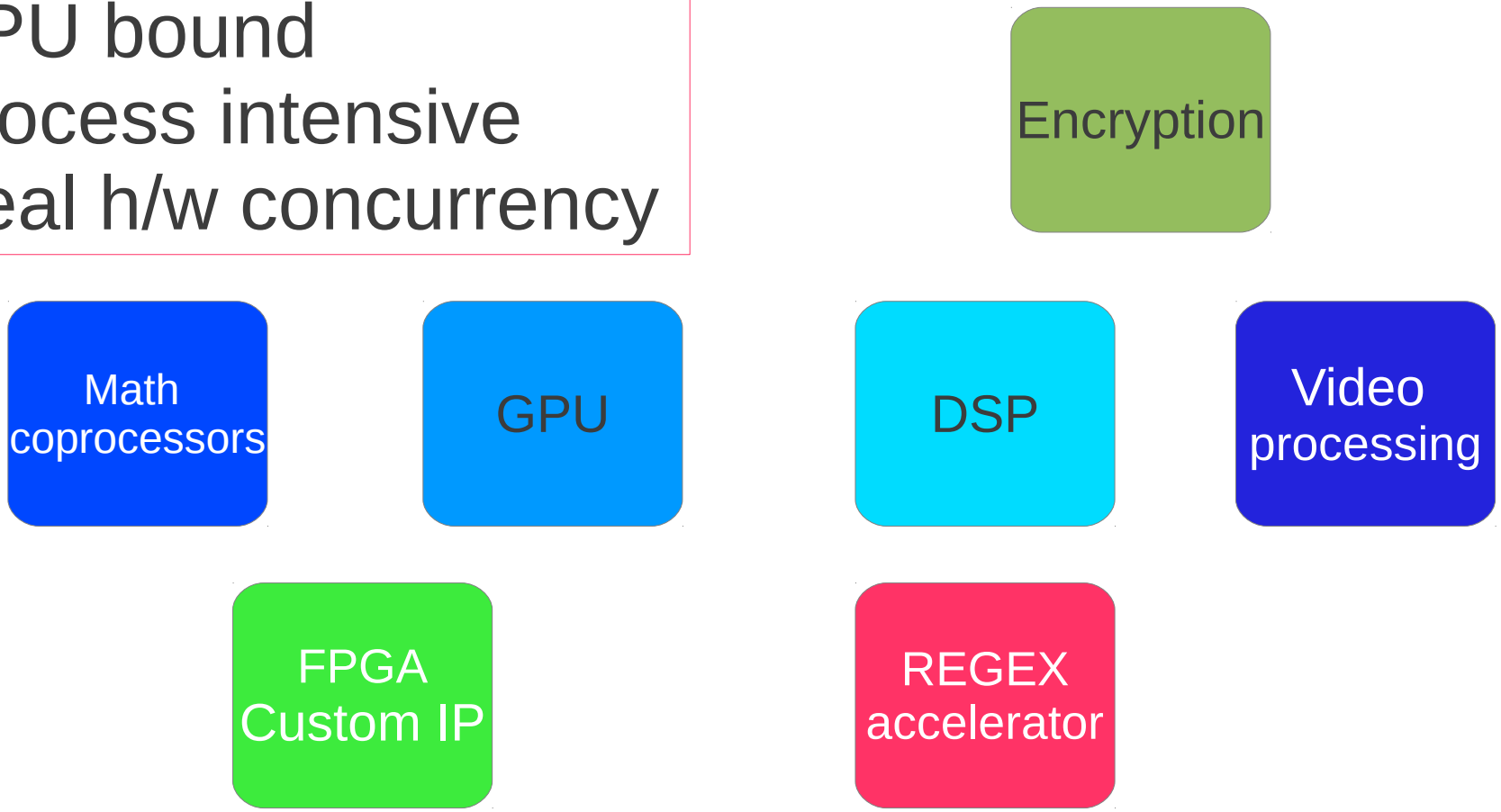
H/W accelerators & co-processors

- CPU bound
- Process intensive
- Real h/w concurrency



H/W accelerators & co-processors

- CPU bound
- Process intensive
- Real h/w concurrency

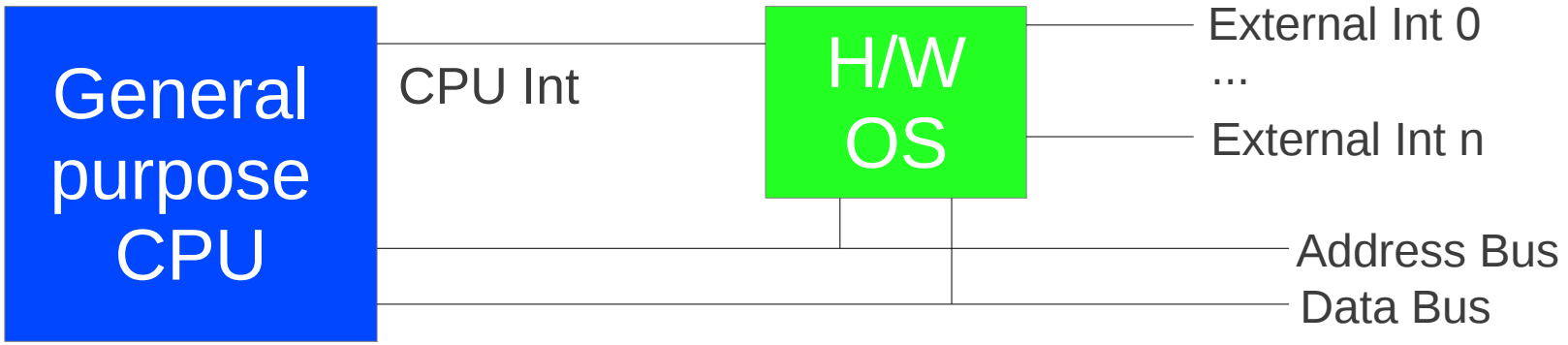


What is Hardware OS?

- Implement Process/Timer/Event handling in H/W
- Not a new idea
  - papers can be found from 90s
- Reduce scheduling overhead
- Improve determinism

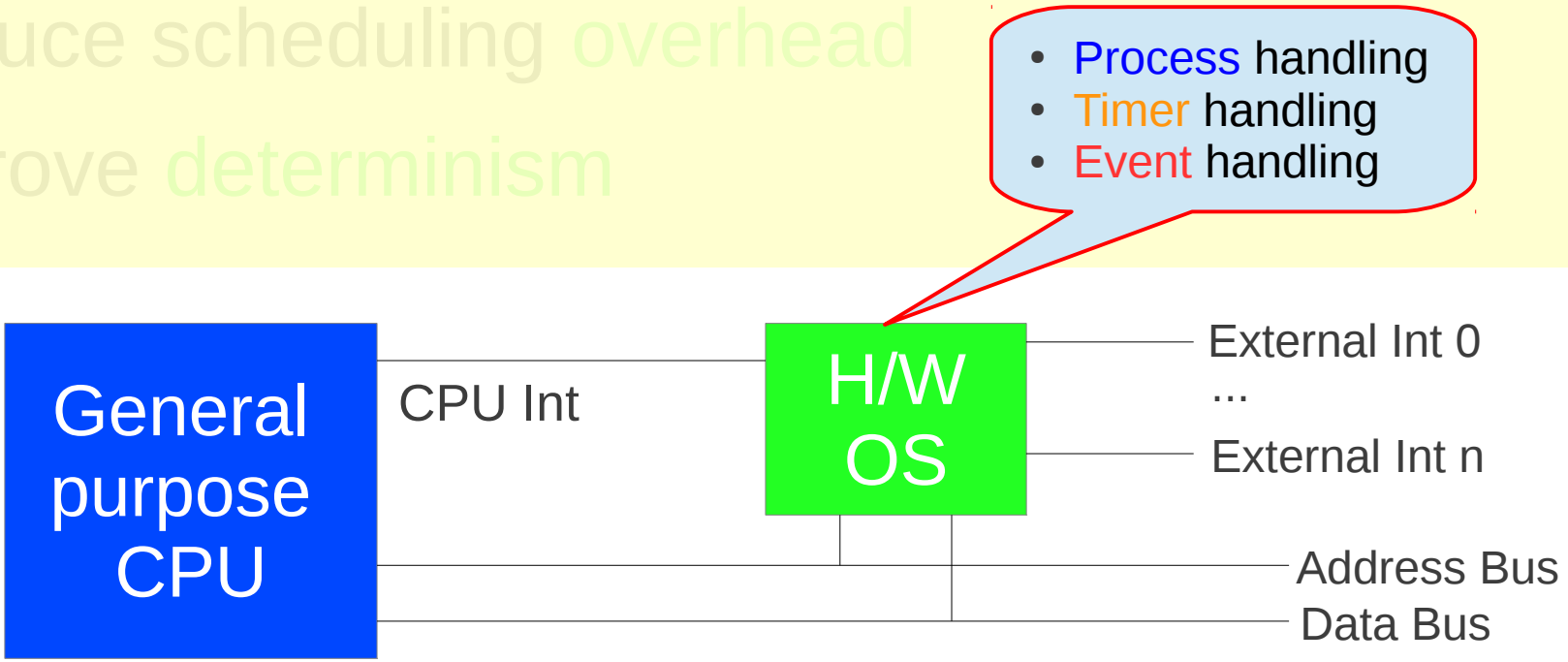
What is Hardware OS?

- Implement task/event/timer handling in H/W
- Not a new idea
  - papers can be found from 90s
- Reduce scheduling overhead
- Improve determinism

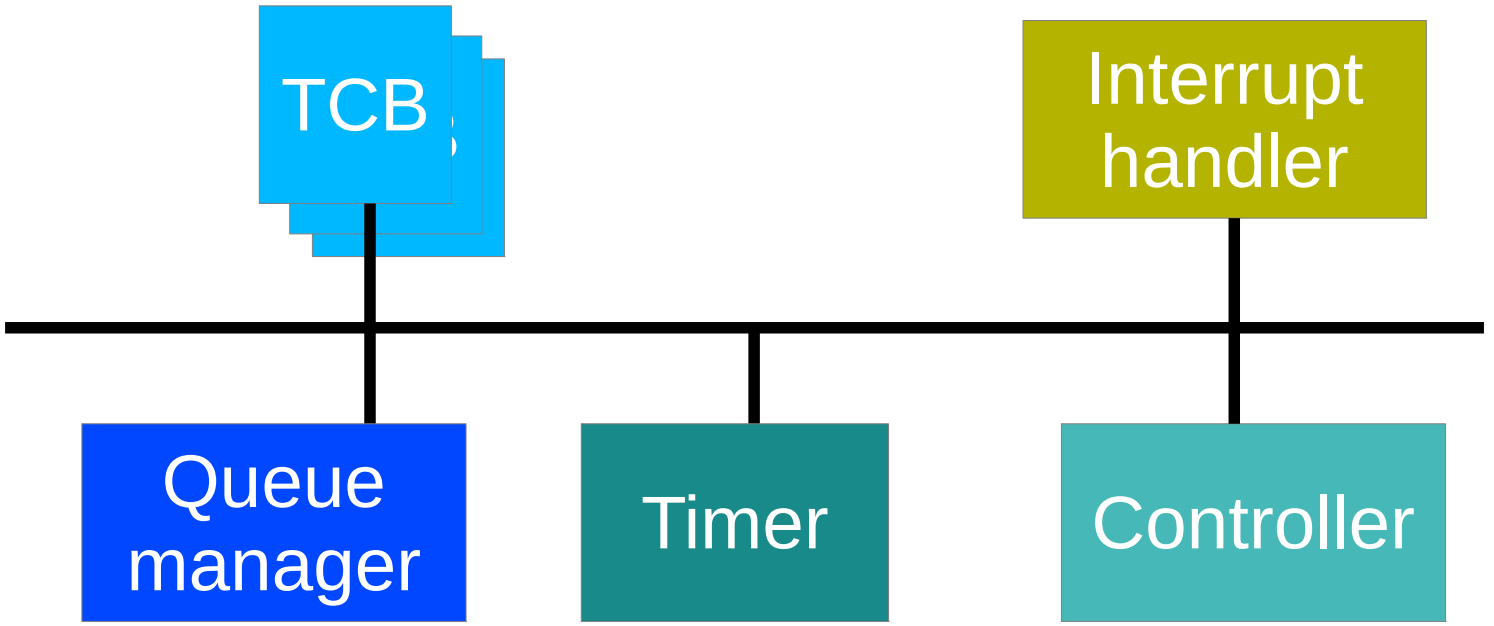


What is Hardware OS?

- Implement task/event/timer handling in H/W
- Not a new idea
  - papers can be found from 90s
- Reduce scheduling overhead
- Improve determinism



H/W blocks inside Hardware OS



Selected List of H/W OS

- Silicon TRON
- F-Timer
- Cooling's scheduler co-processor board
- FASTCHART/RTM (real time unit)
- A. Morton's scheduler coprocessor
- Sierra OS (FPGA IP)

Selected List of H/W OS

- Silicon TRON
- F-Timer
- Cooling's scheduler co-processor board
- FASTCHART/RTM (real time unit)
- A. Morton's scheduler coprocessor
- Sierra OS (FPGA IP)

Selected List of H/W OS

- Silicon TRON
- F-Timer
- Cooling's scheduler co-processor board
- FASTCHART/RTM (real time unit)
- A. Morton's scheduler coprocessor
- Sierra OS (FPGA IP)

Selected List of H/W OS

- Silicon TRON
- F-Timer
- Cooling's scheduler co-processor board
- FASTCHART/RTM (real time unit)
- A. Morton's scheduler coprocessor
- Sierra OS (FPGA IP)

Selected List of H/W OS

- Silicon TRON
- F-Timer
- Cooling's scheduler co-processor board
- FASTCHART/RTM (real time unit)
- A. Morton's scheduler coprocessor
- Sierra OS (FPGA IP)

Selected List of H/W OS

- Silicon TRON
- F-Timer
- Cooling's scheduler co-processor board
- FASTCHART/RTM (real time unit)
- A. Morton's scheduler coprocessor
- Sierra OS (FPGA IP)

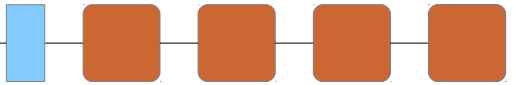
Great! H/W OS is available?

- Not quite
- Before we discuss why not
  - Let's talk briefly Linux and Open source software

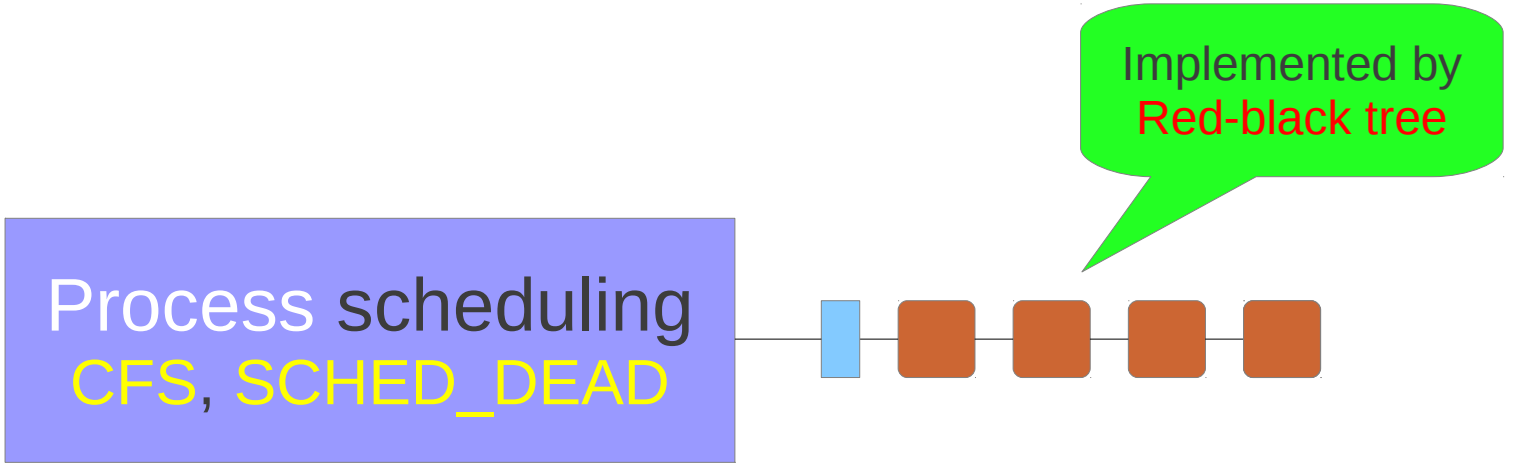
- Why not h/w OS is available?
  - Need software OS change
  - OS scheduler is quite complex and changing

Linux scheduler and hrtimer

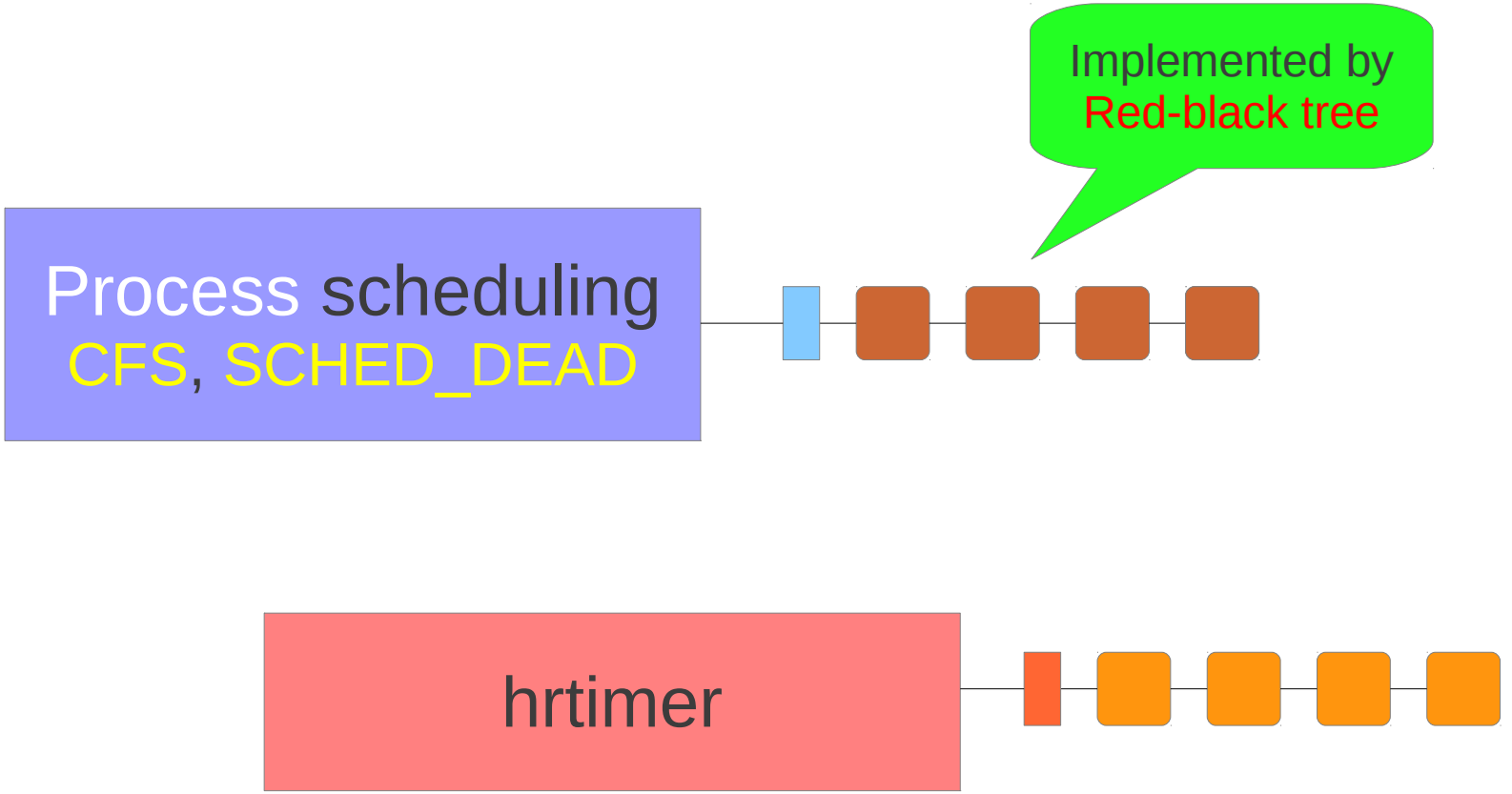
Process scheduling
CFS, SCHED_DEAD



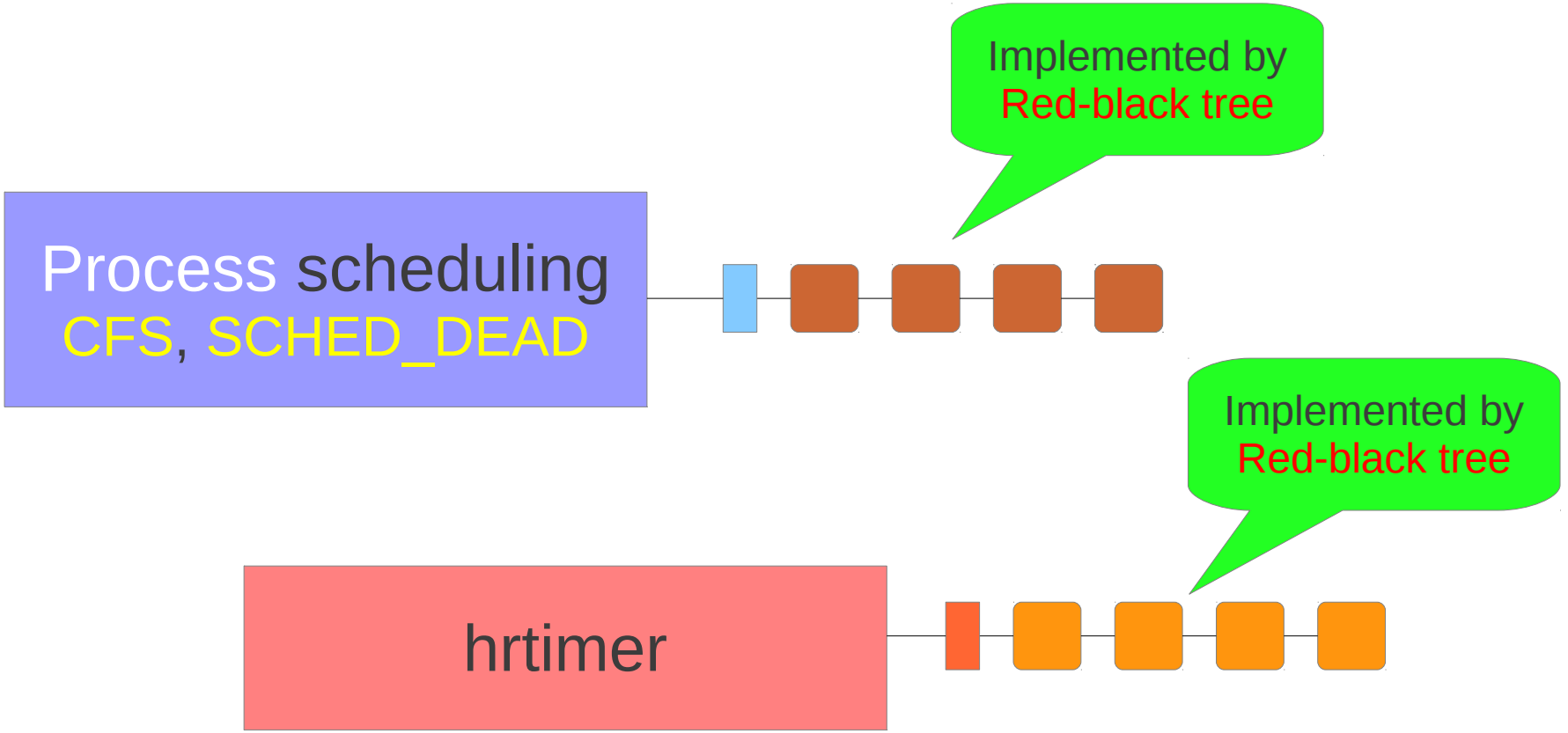
Linux scheduler and hrtimer



Linux scheduler and hrtimer



Linux scheduler and hrtimer

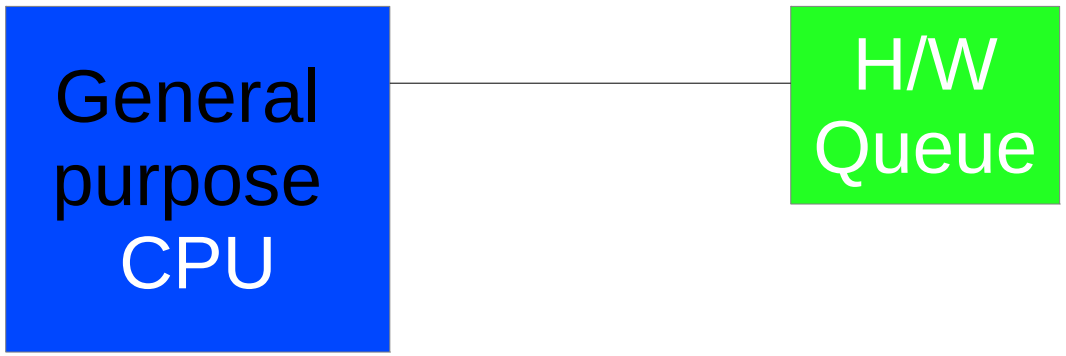


Hybrid approach

- Instead of trying to implement full/semi-full scheduler in h/w
  - Only implement a priority queue in h/w
  - Scheduler use h/w queue instead of RB-tree

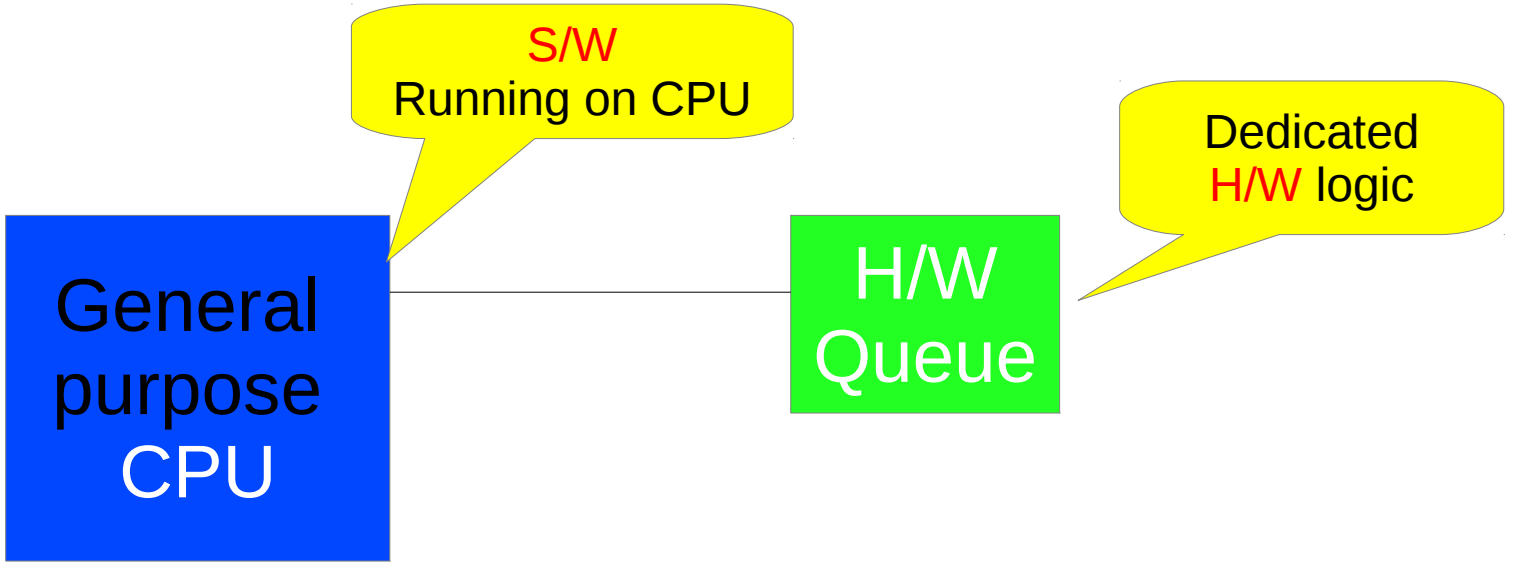
Hybrid approach

- Implement priority queue in h/w
- Scheduler use h/w queue instead of RB-tree



Hybrid approach

- Implement priority queue in h/w
- Scheduler use h/w queue instead of RB-tree



H/W Priority Queue

- Requirements:
  - Enqueue/dequeue in one cycle
  - Maintain FIFO order
- Example application of h/w priority queue
  - Network switch/router traffic management
    - for maintaining QoS during congestion

Types of h/w priority queues

- N: number entry
- P: number priorities

Types of h/w priority queues

- N: number entry
- P: number priorities

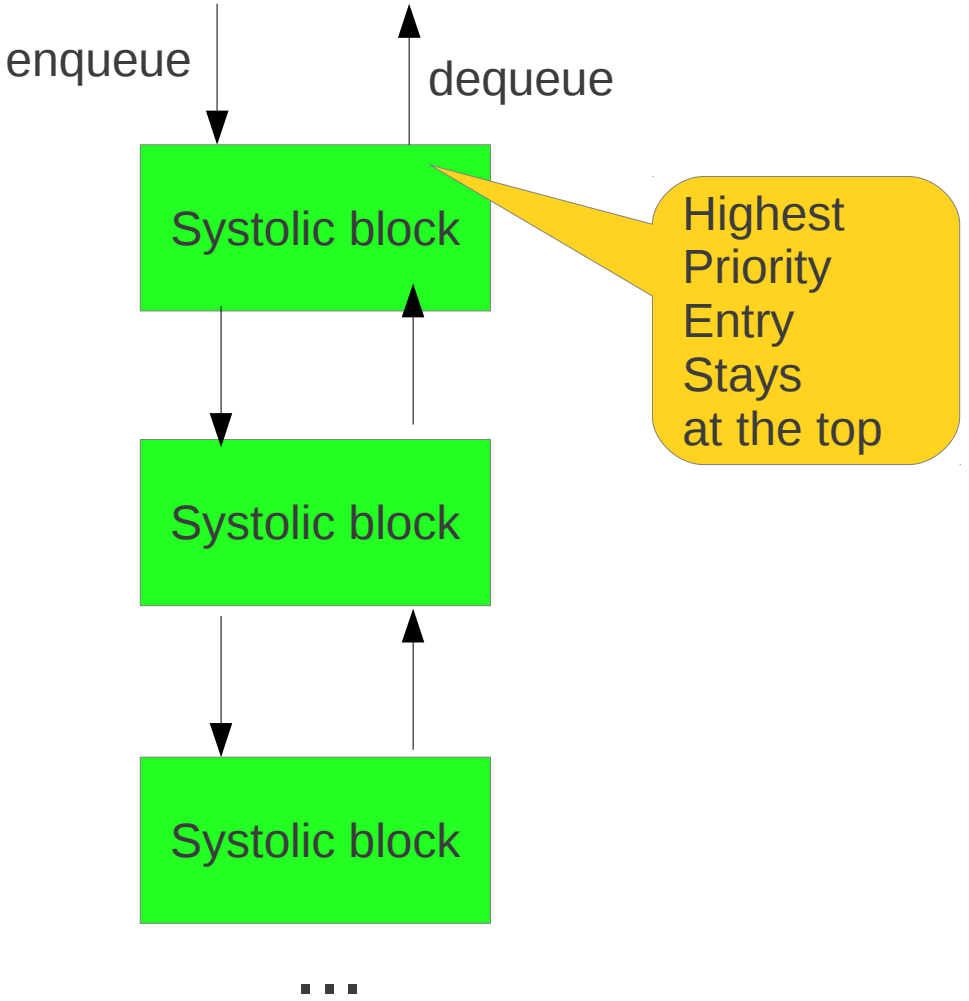
- Binary tree of comparators
- Priority encoder with multiple queues
- Tagged up/down sorter
  - Scales well with large N and P

Types of h/w priority queues

- N: number entry
- P: number priorities

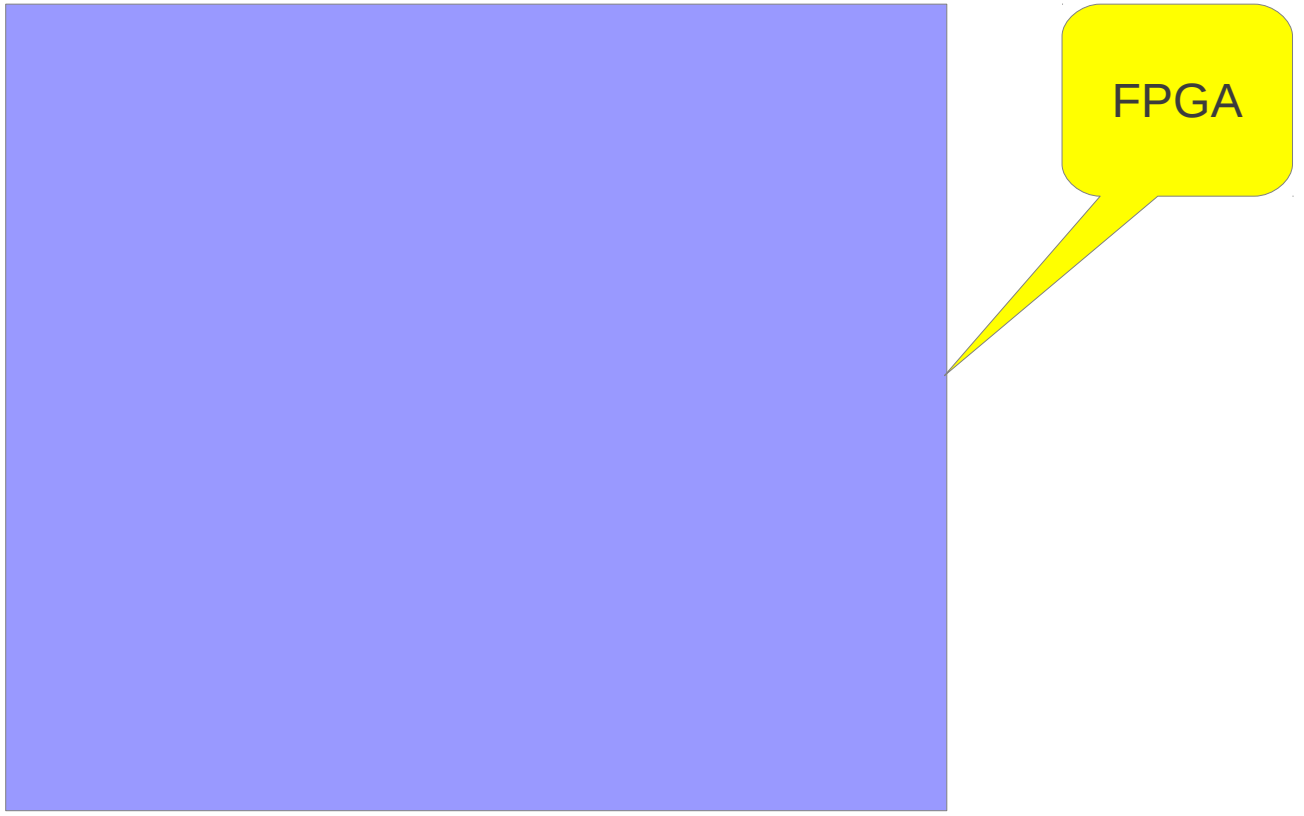
- Binary tree of comparators
- Priority encoder with multiple queues
- Tagged up/down sorter
  - Scales well with large N and P

Tagged Up/Down H/W Sorter



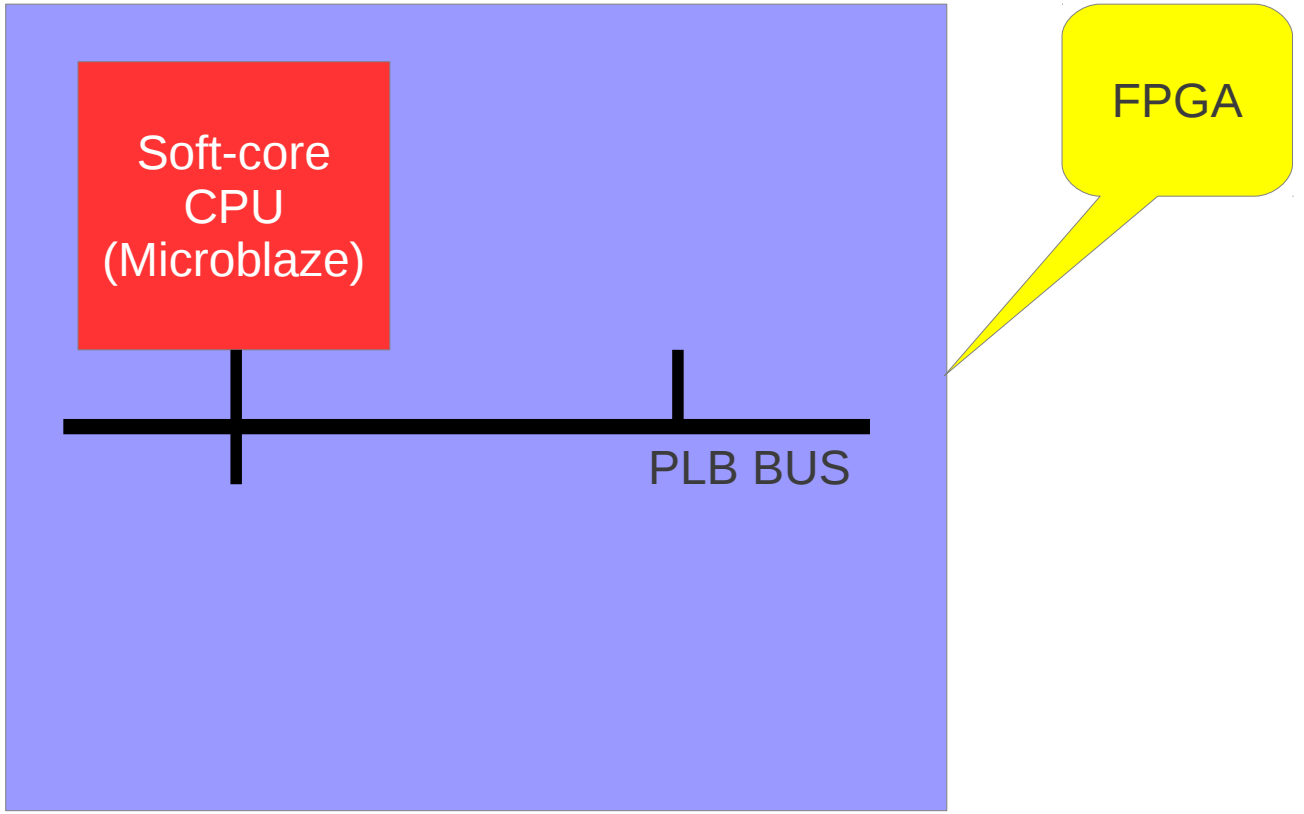
Implementation

- FPGA



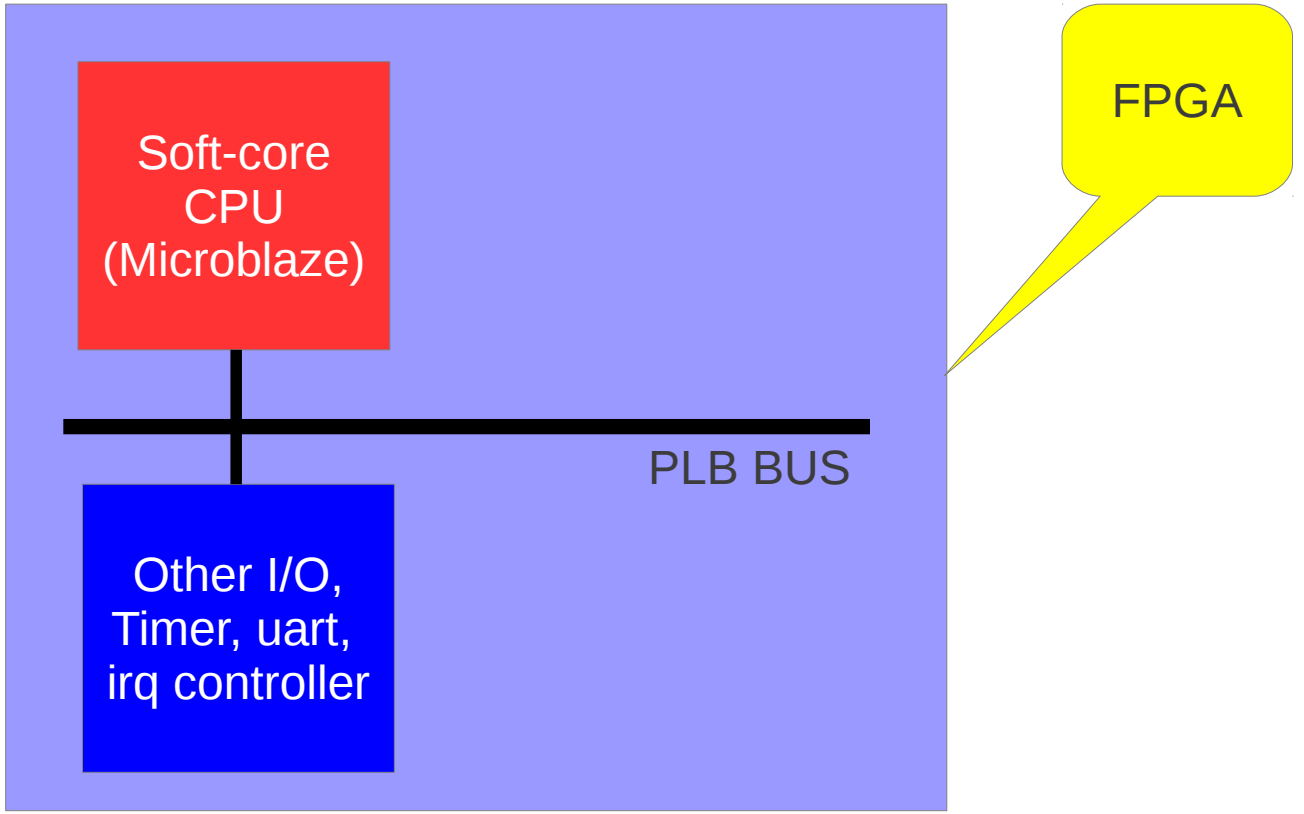
Implementation

- FPGA
- Soft-core processor



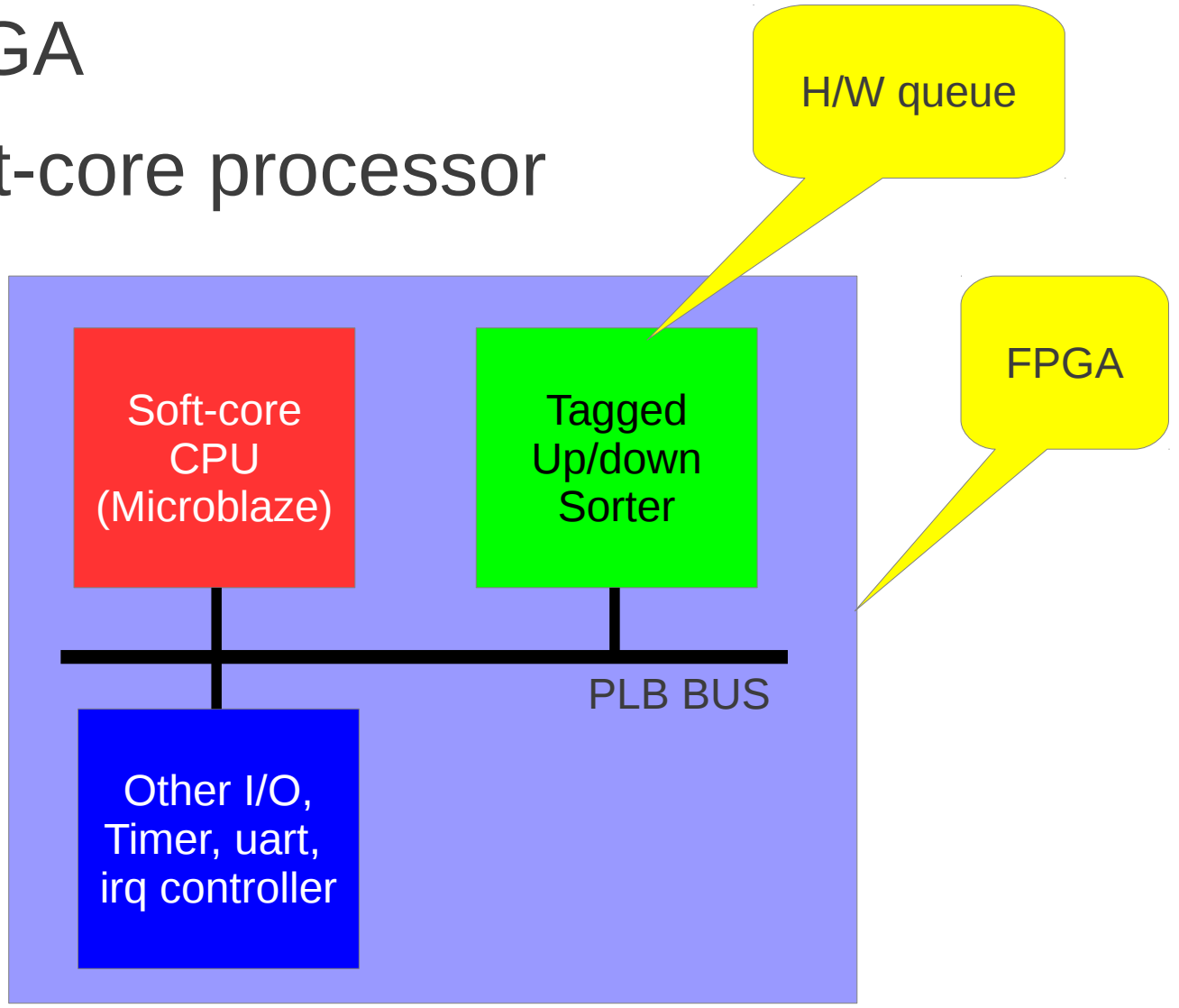
Implementation

- FPGA
- Soft-core processor



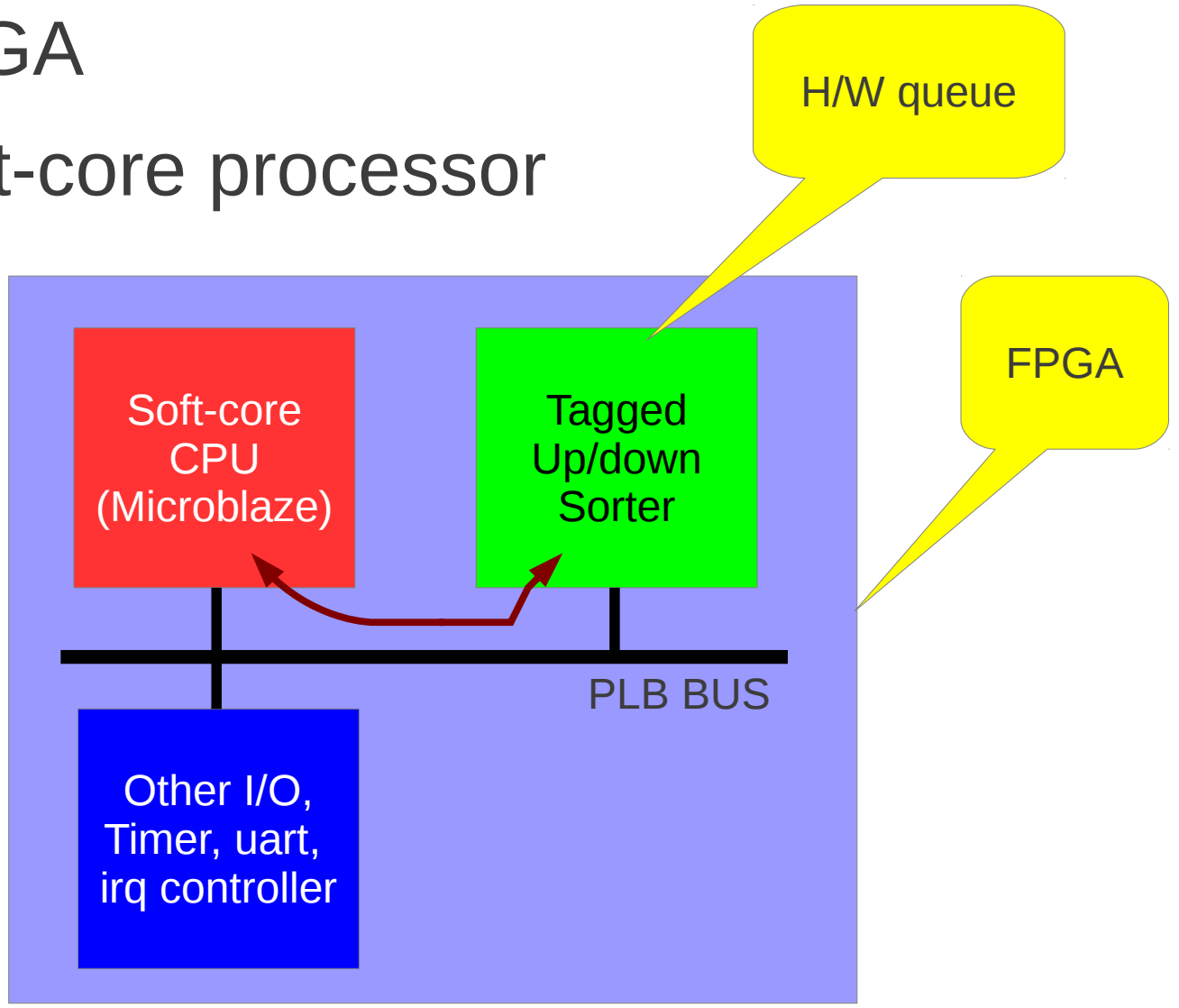
Implementation

- FPGA
- Soft-core processor



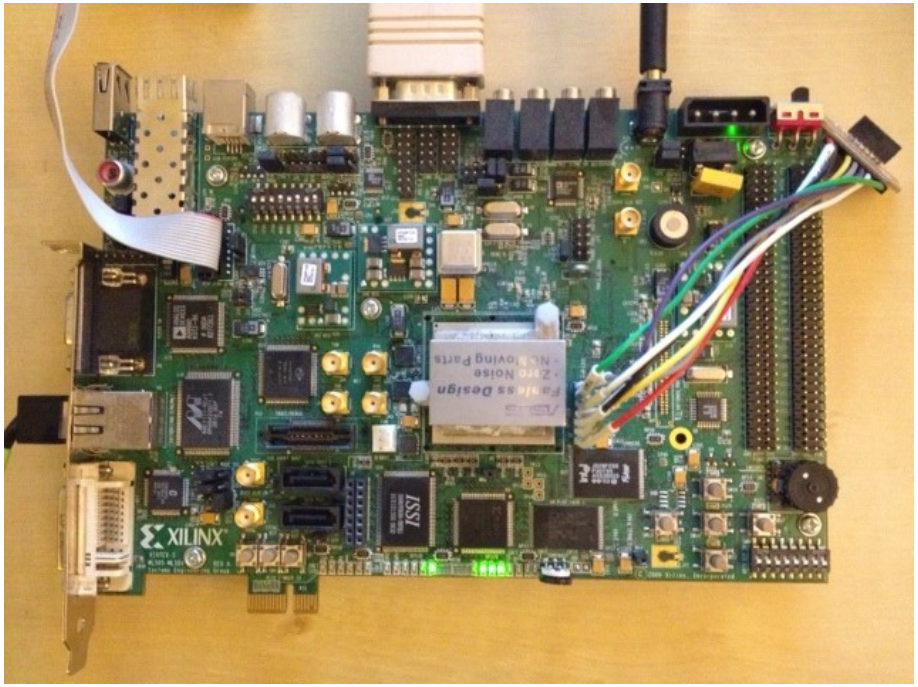
Implementation

- FPGA
- Soft-core processor

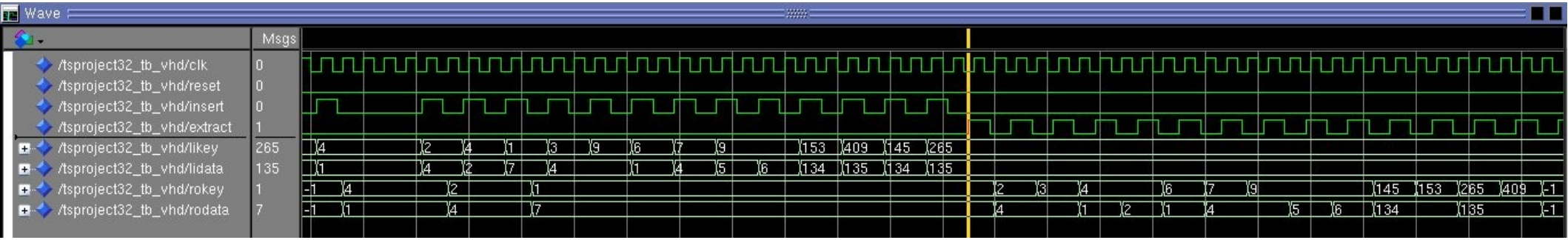


Experimental setup

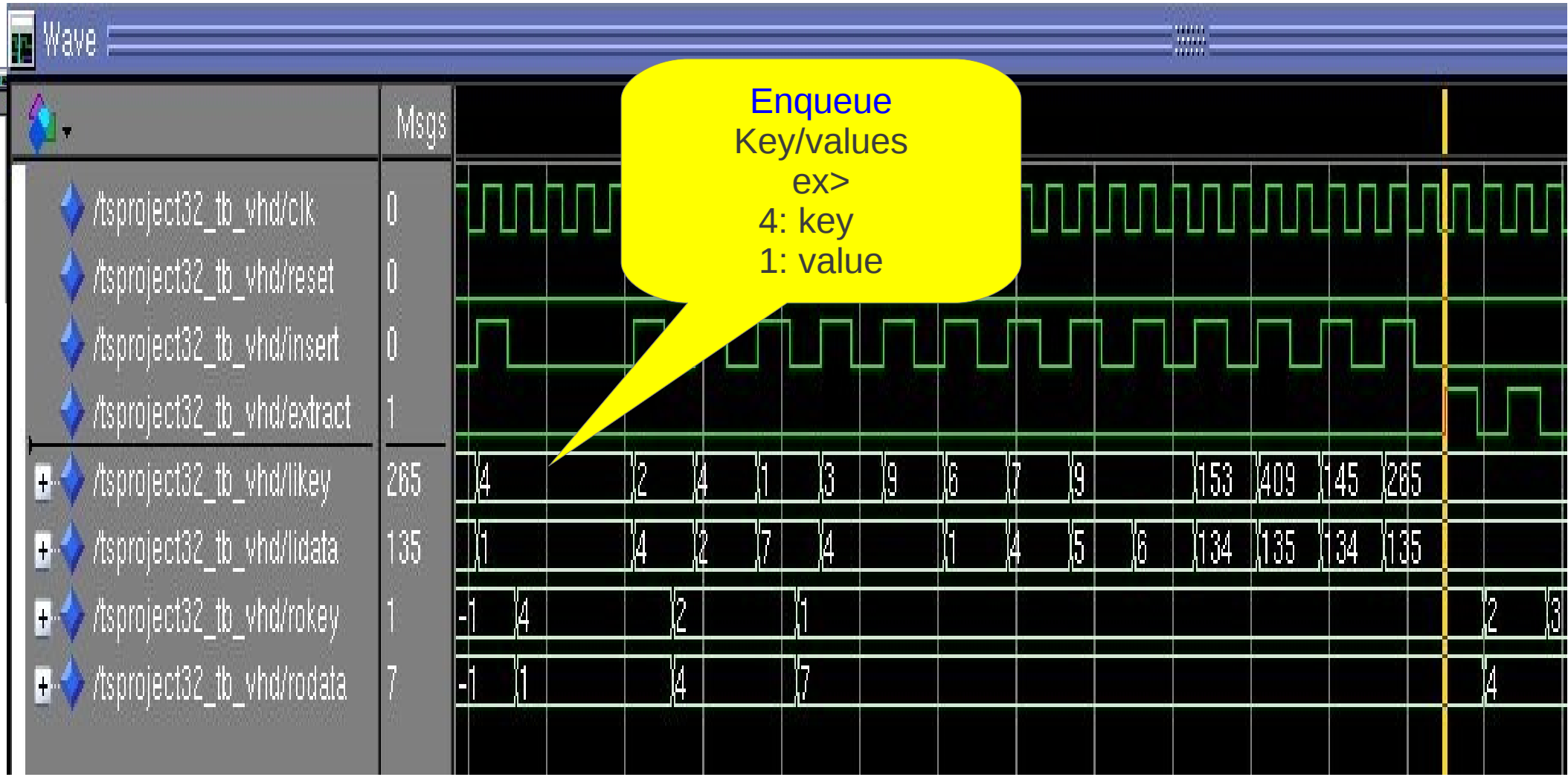
- FPGA (Field Programmable Gate Array)
  - Xilinx Virtex5
- Soft-core CPU
  - Microblaze
- ML505 evaluation board



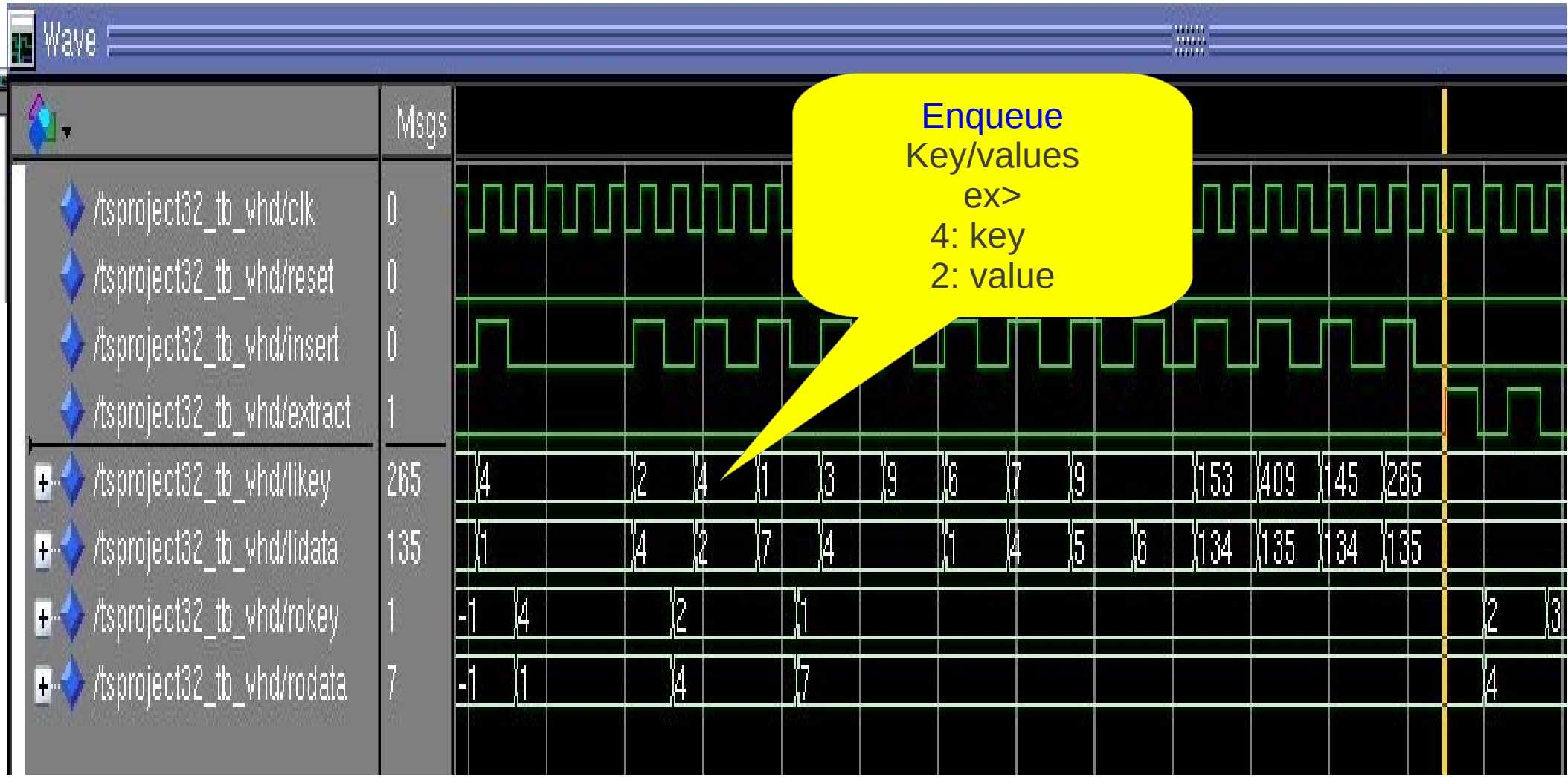
H/W Priority Queue Simulation



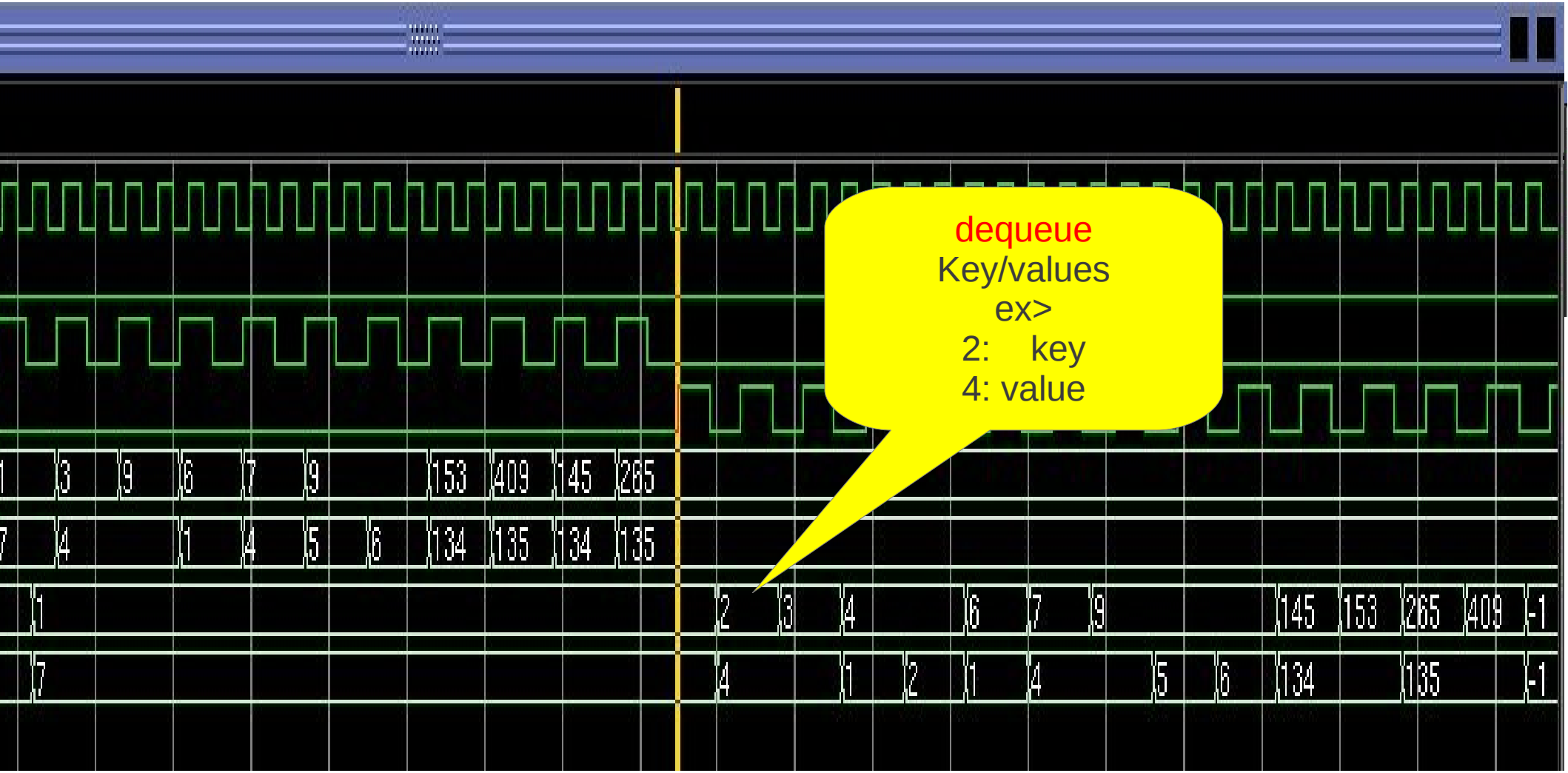
H/W Priority Queue Simulation



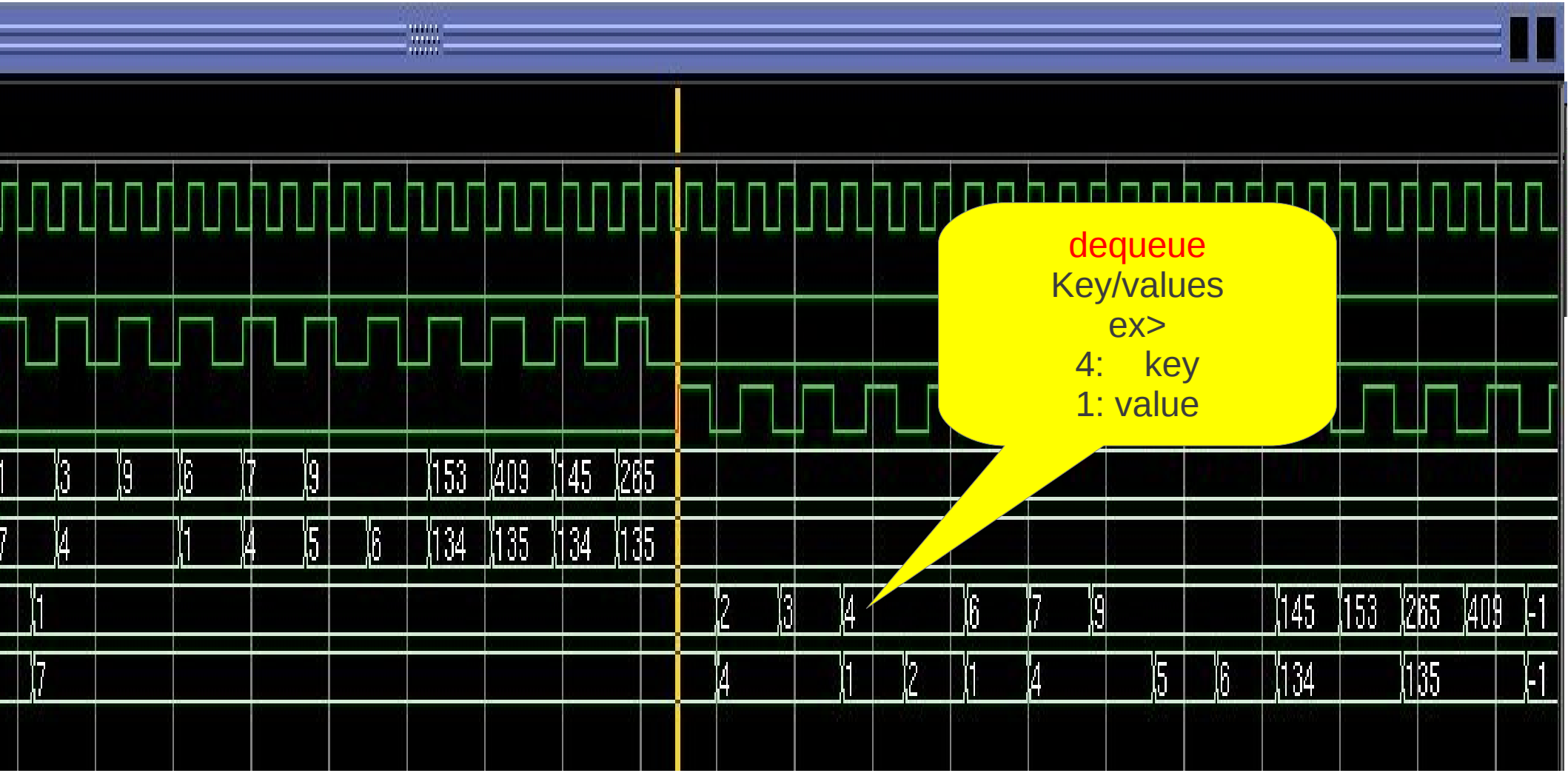
H/W Priority Queue Simulation



H/W Priority Queue Simulation



H/W Priority Queue Simulation



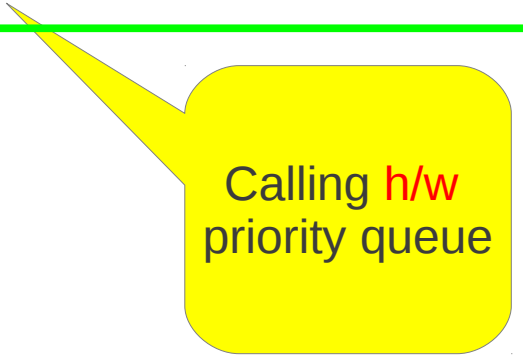
H/W Priority Queue Simulation

Scheduler change, sched_fair.c

```
/*
 * modified __enqueue_entity() to use Tagged Sorter
 * instead of rb-tree
 */
static void __enqueue_entity(struct cfs_rq *cfs_rq, struct sched_entity *se)
{
  struct rb_node **link = &cfs_rq->tasks_timeline.rb_node;
  struct rb_node *parent = NULL;
  struct sched_entity *entry;

  s64 key = entity_key(cfs_rq, se);
  __enqueue_tagged_sorter(key, se);  // enqueue to Tagged Sorter

}
```

Scheduler change, sched_fair.c

```
/*
 * modified __enqueue_entity() to use Tagged Sorter
 * instead of rb-tree
 */
static void __enqueue_entity(struct cfs_rq *cfs_rq, struct sched_entity *se)
{
  struct rb_node **link = &cfs_rq->tasks_timeline.rb_node;
  struct rb_node *parent = NULL;
  struct sched_entity *entry;

  s64 key = entity_key(cfs_rq, se);
  __enqueue_tagged_sorter(key, se);  // enqueue to Tagged Sorter

}
```

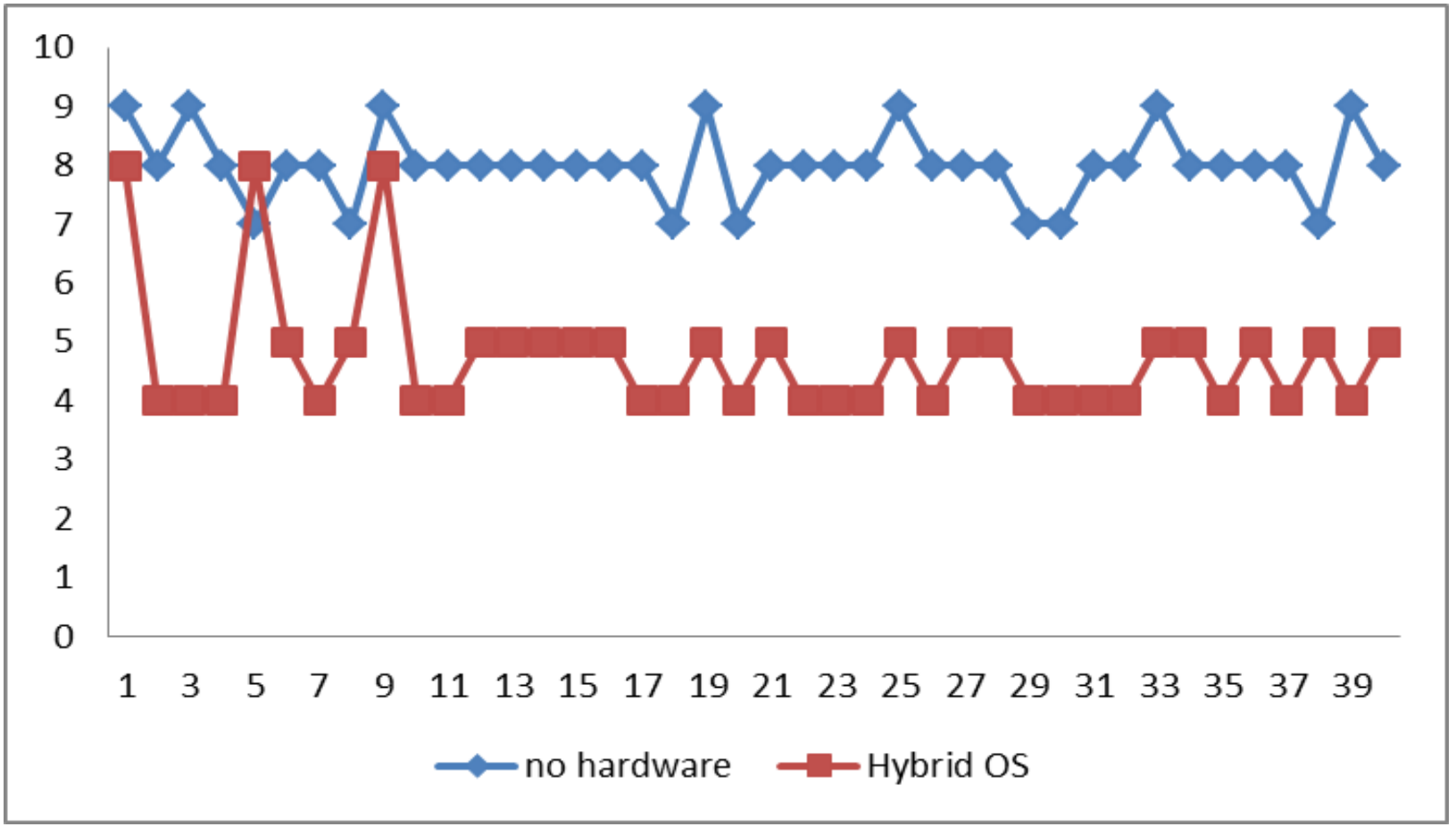


H/W interface

```
/*
 * enqueue entity to Tagged Sorter (HyOS)
 */
static void __enqueue_tagged_sorter(unsigned int key, unsigned int  data)
{
  if (init == 0) {
    hyos = (char*)ioremap(XPAR_HYOS_PLB_0_BASEADDR, 0x10000);
    init = 1;
  }

  if (hyos != NULL) {
    iowrite32(key,hyos+4*4);    // key
    iowrite32(data,hyos+5*4);   // val
    iowrite32(0xffffffff,hyos); // enqueue trigger
    iowrite32(0x0,hyos);
  }
}
```

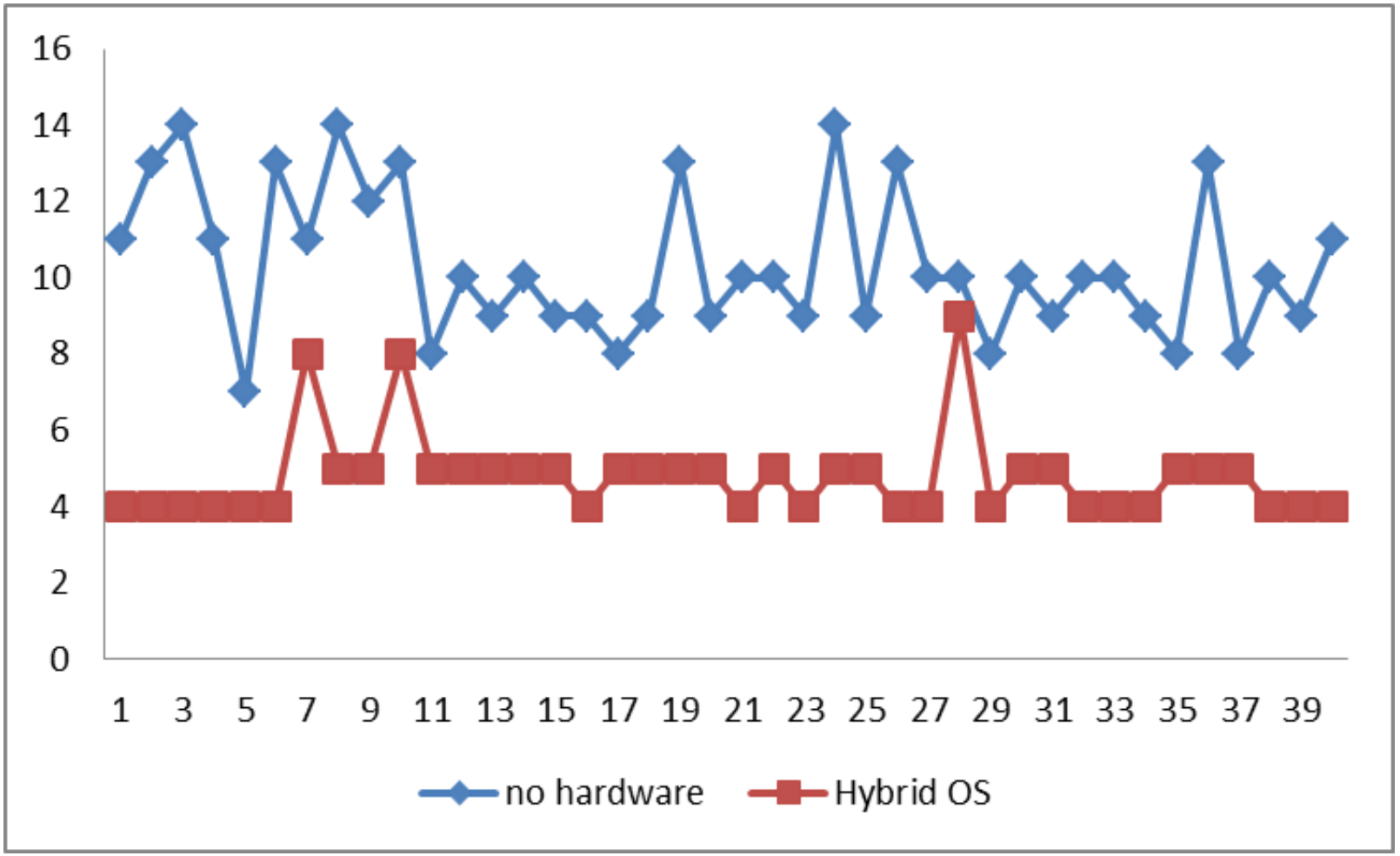
Experimental results 1

- Enqueue time for no load



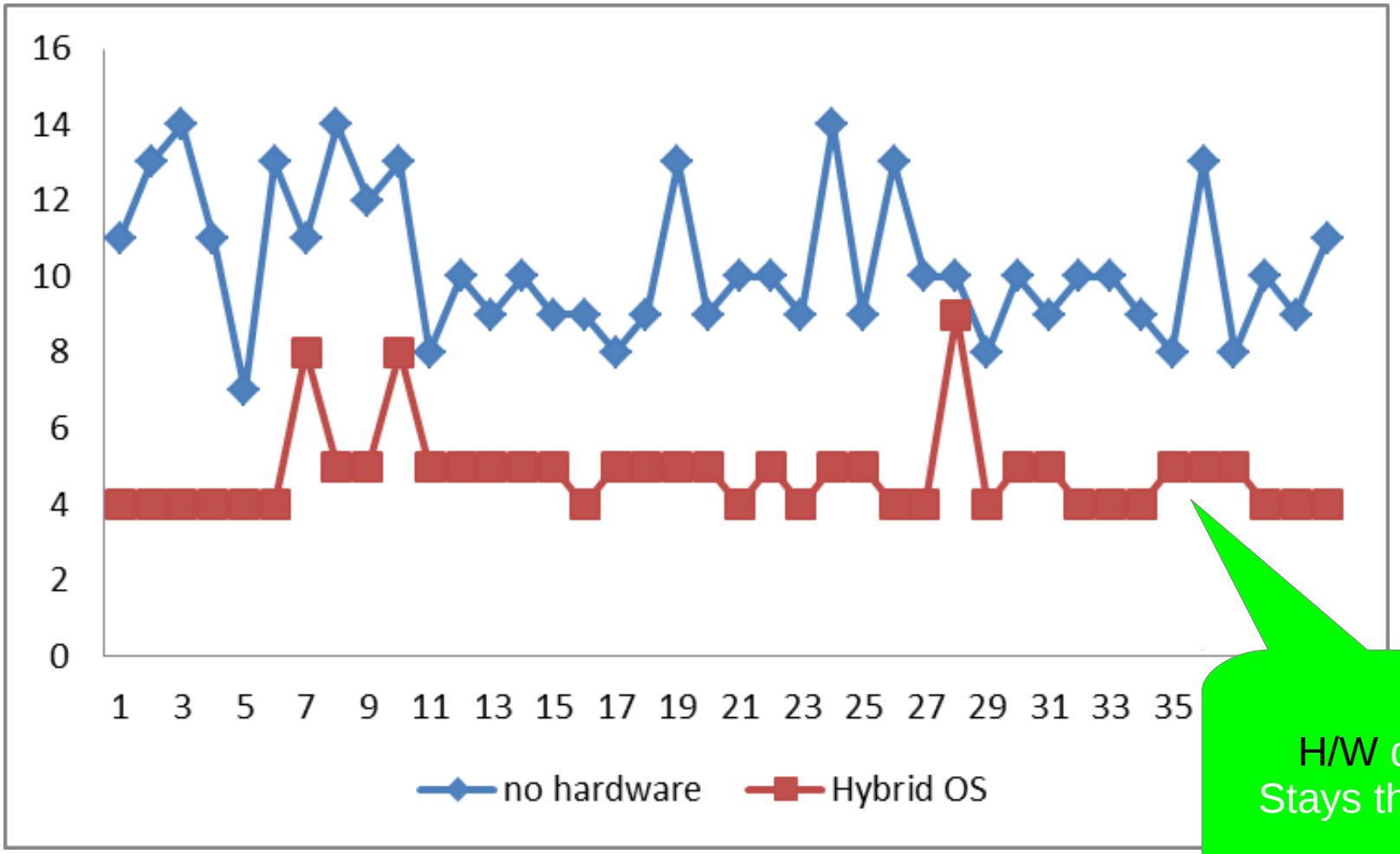
Experimental results 2

- Enqueue time for 4 yes



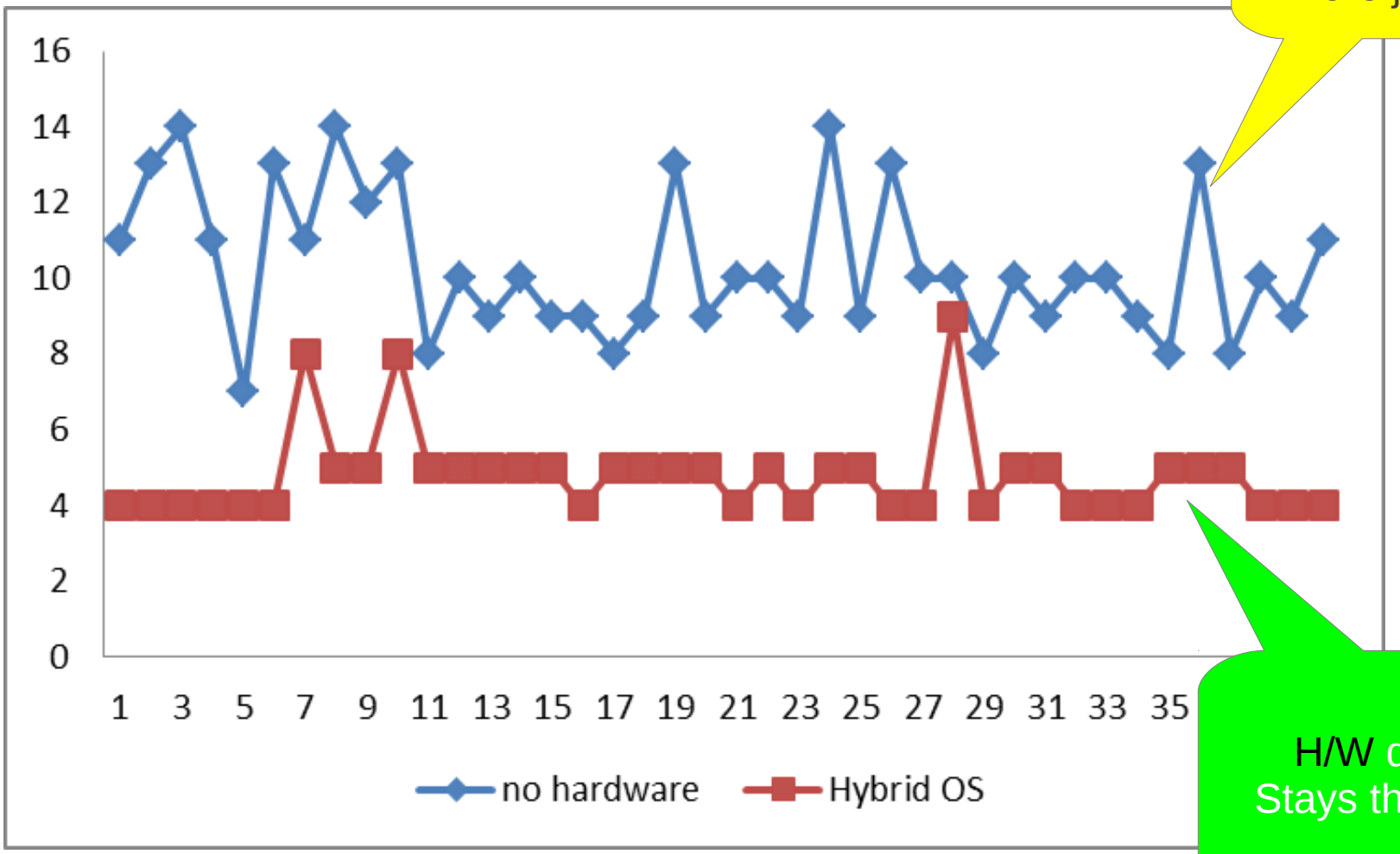
Experimental results 2

- Enqueue time for 4 yes



Experimental results 2

- Enqueue time for 4 yes



S/W queue increased & more jitters

H/W queue Stays the same

Conclusion

- H/W OS and H/W accelerator can
  - Improve performance
  - Reduce overhead
- Hybrid approach
  - Instead of implement full scheduler
  - Off-load a commonly used s/w component to h/w

- Could look for more s/w components could be off-loaded

Thank you!

Questions?