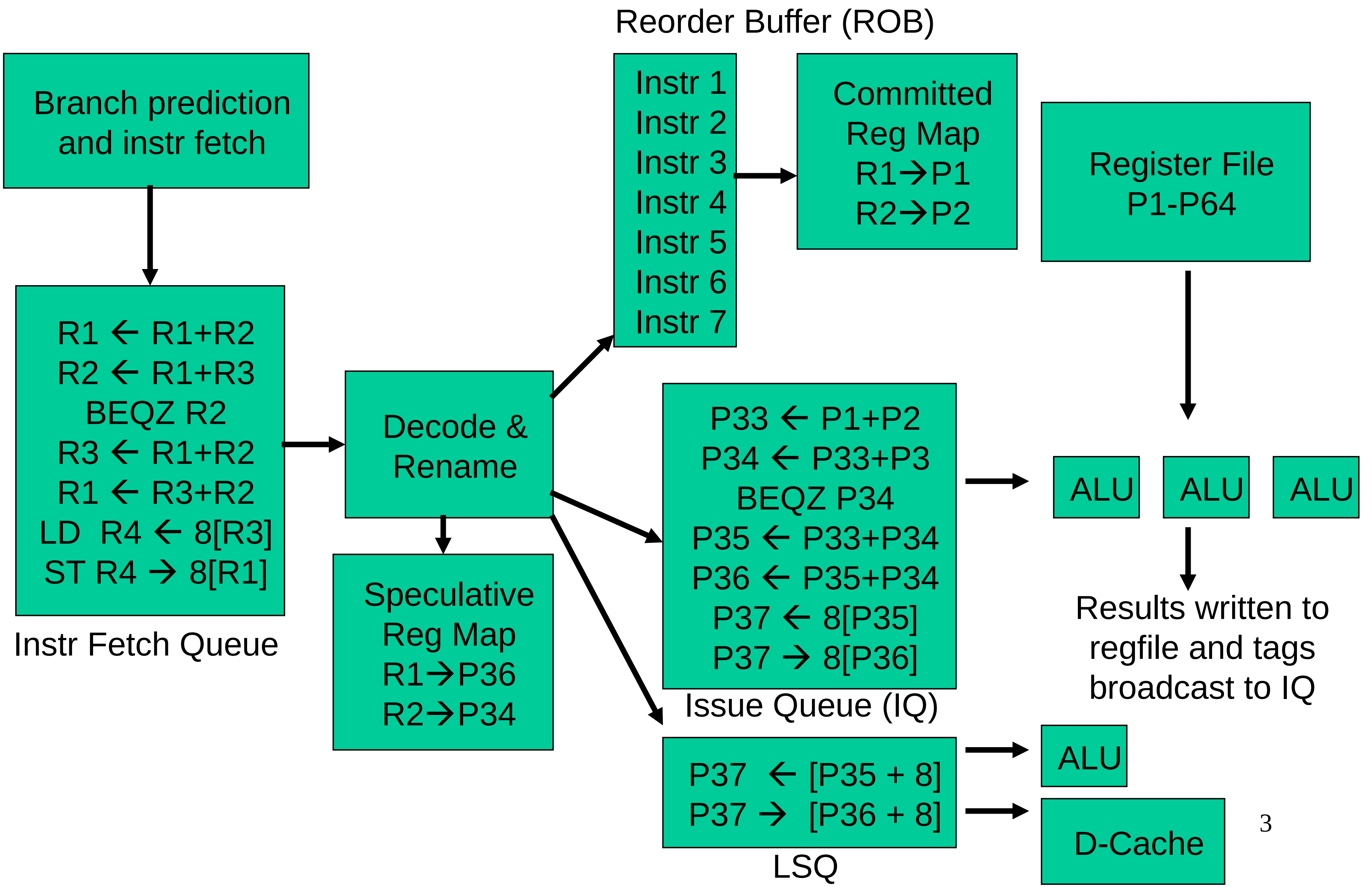
CS/EE3810: Computer Organization

Lecture 18: Hyperthreading, SIMD, GPUs

Anton Burtsev
November, 2022

Hyperthreading

The Alpha 21264 Out-of-Order Implementation



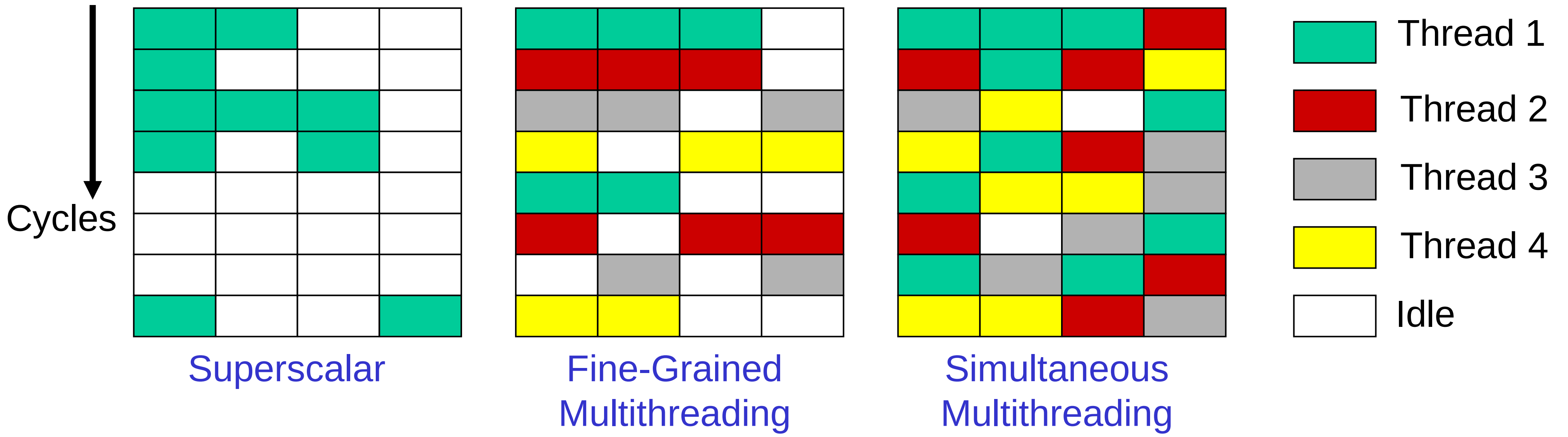
Thread-Level Parallelism

- Motivation:
  - a single thread leaves a processor under-utilized for most of the time
  - by doubling processor area, single thread performance barely improves

- Strategies for thread-level parallelism:
  - multiple threads share the same large processor → reduces under-utilization, efficient resource allocation
    Simultaneous Multi-Threading (SMT)
  - each thread executes on its own mini processor → simple design, low interference between threads
    Chip Multi-Processing (CMP) or multi-core

How are Resources Shared?

Each box represents an issue slot for a functional unit. Peak thruput is 4 IPC.



- Superscalar processor has high under-utilization – not enough work every cycle, especially when there is a cache miss
- Fine-grained multithreading can only issue instructions from a single thread in a cycle – can not find max work every cycle, but cache misses can be tolerated
- Simultaneous multithreading can issue instructions from any thread every cycle – has the highest probability of finding work for every issue slot

What Resources are Shared?

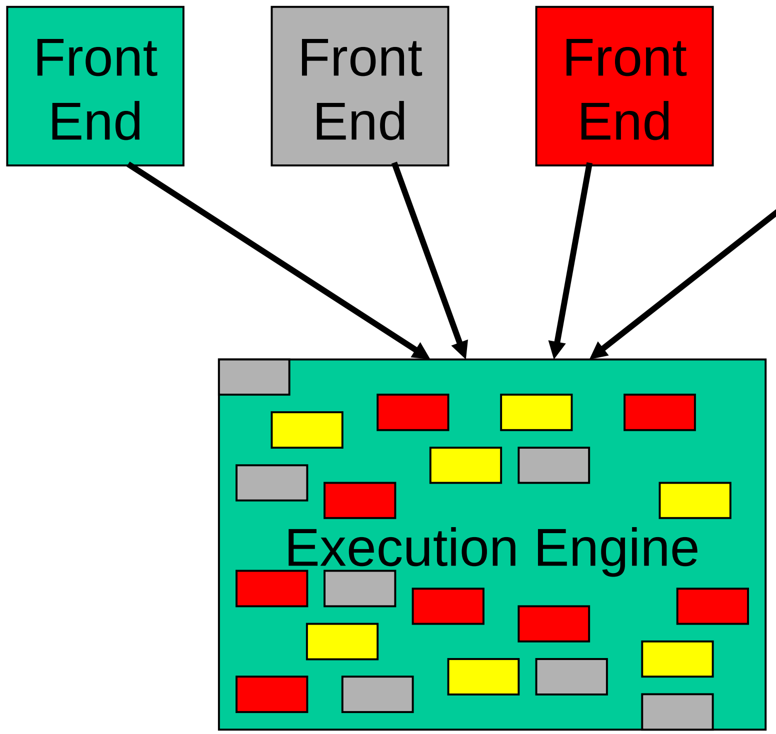
- Multiple threads are simultaneously active (in other words, a new thread can start without a context switch)

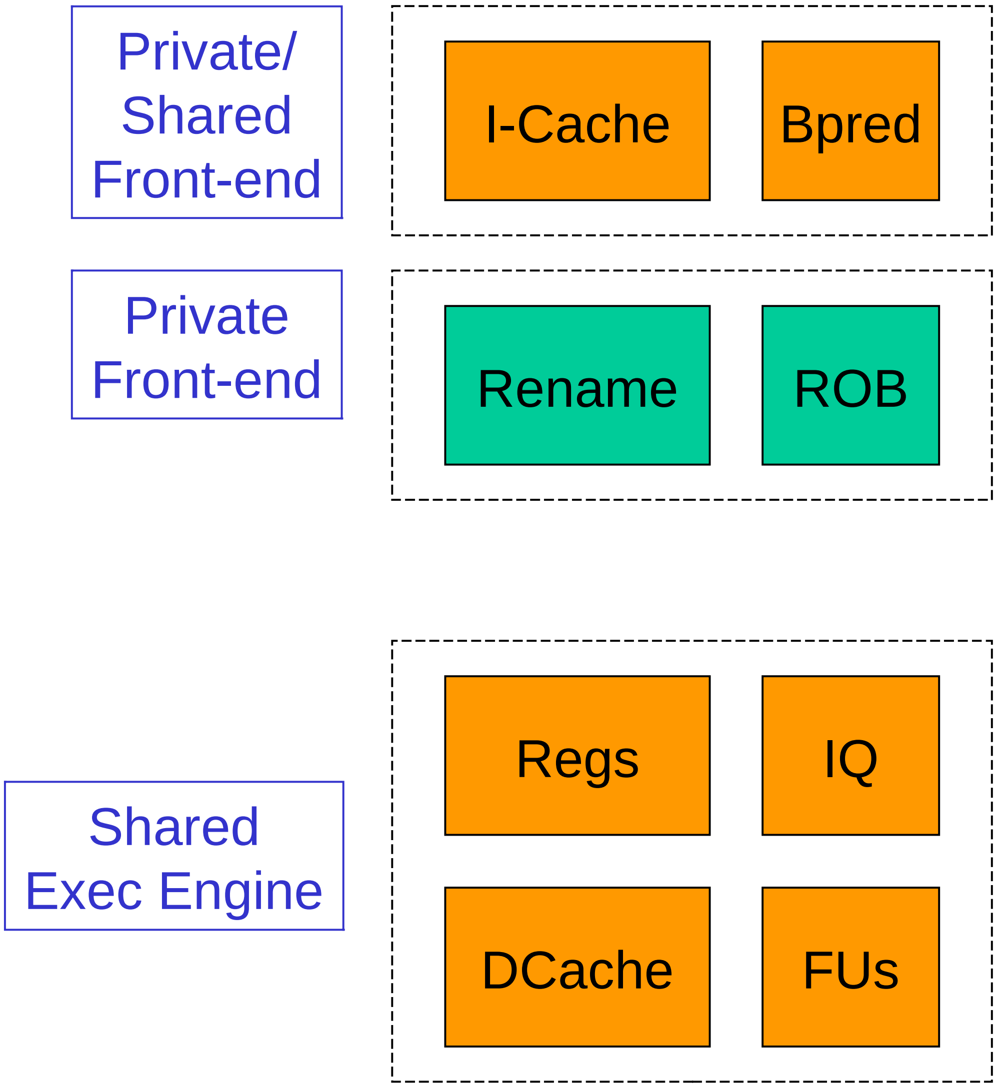
- For correctness, each thread needs its own PC, IFQ, logical regs (and its own mappings from logical to phys regs)

- For performance, each thread could have its own ROB/LSQ (so that a stall in one thread does not stall commit in other threads), I-cache, branch predictor, D-cache, etc. (for low interference), although note that more sharing → better utilization of resources

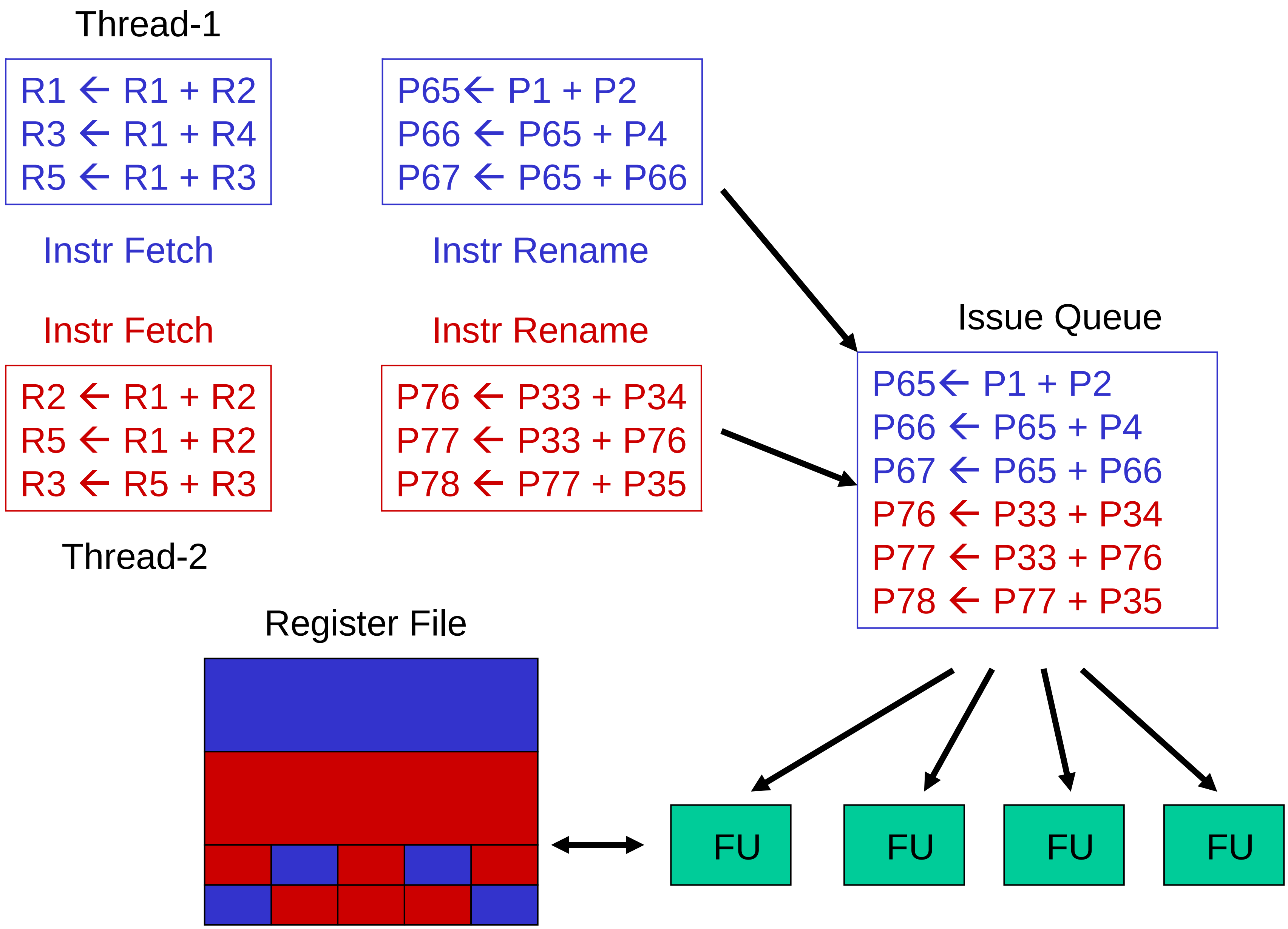
- Each additional thread costs a PC, IFQ, rename tables, and ROB – cheap!

Pipeline Structure





Resource Sharing



Performance Implications of SMT

- Single thread performance is likely to go down (caches, branch predictors, registers, etc. are shared) – this effect can be mitigated by trying to prioritize one thread

- With eight threads in a processor with many resources, SMT yields throughput improvements of roughly 2-4

Single instruction multiple data (SIMD)

SIMD Processors

- Single instruction, multiple data
- Such processors offer energy efficiency because a single instruction fetch can trigger many data operations
- Such data parallelism may be useful for many image/sound and numerical applications

Example

$$Y = a \times X + Y$$

```
      l.d      $f0,a($sp)       :load scalar a
      addiu    $t0,$s0,#512     :upper bound of what to load
loop: l.d      $f2,0($s0)       :load x(i)
      mul.d    $f2,$f2,$f0      :a x x(i)
      l.d      $f4,0($s1)       :load y(i)
      add.d    $f4,$f4,$f2      :a x x(i) + y(i)
      s.d      $f4,0($s1)       :store into y(i)
      addiu    $s0,$s0,#8       :increment index to x
      addiu    $s1,$s1,#8       :increment index to y
      subu     $t1,$t0,$s0      :compute bound
      bne      $t1,$zero,loop   :check if done
```

```
      l.d      $f0,a($sp)       :load scalar a
      lv       $v1,0($s0)       :load vector x
      mulvs.d  $v2,$v1,$f0      :vector-scalar multiply
      lv       $v3,0($s1)       :load vector y
      addv.d   $v4,$v2,$v3      :add y to product
      sv       $v4,0($s1)       :store the result
```

Example #2

- Hash table
  - Insertion

```
constexpr std::array<__mmask8, KV_PER_CACHE_LINE>
key_cmp_masks = {
        KEY3 | KEY2 | KEY1 | KEY0, // cidx: 0; all key comparisons valid
        KEY3 | KEY2 | KEY1, // cidx: 1; only last three comparisons valid
        KEY3 | KEY2, // cidx: 2; only last two comparisons valid
        KEY3, // cidx: 3; only last comparison valid
};
auto key_cmp = [&key_cmp_masks](__m512i cacheline,
    __m512i key_mask, size_t cidx) {
    __mmask8 cmp = _mm512_cmpeq_epu64_mask(cacheline, key_mask);
    // zmm registers are compared as 8 uint64_t
    // mask irrelevant results before returning
    return cmp & key_cmp_masks[cidx];
};

const size_t cidx = idx & (KV_PER_CACHE_LINE−1);
__m512i cacheline = load_cacheline(cidx);
// load a vector of the key in all 4 positions
__m512i key_mask = load_key_mask();
__mmask8 eq_cmp = key_cmp(cacheline, key_mask, cidx);
// compute a mask for copying the key into an empty slot
// will be 0 if eq_cmp != 0 (key already exists in the cacheline)
__mmask16 copy_mask = key_copy_mask(cacheline, eq_cmp, cidx);
copy_key(cacheline, key_mask, static_cast<__mmask8>(copy_mask));
// write the cacheline back; just the KV pair that was modified
__mmask8 kv_mask = key_mask | val_mask;
store_cacheline(cacheline, kv_mask);
// prepare for possible reprobe
...
```

GPUs

- Initially developed as graphics accelerators; now viewed as one of the densest compute engines available

- Many on-going efforts to run non-graphics workloads on GPUs, i.e., use them as general-purpose GPUs or GPGPUs

- C/C++ based programming platforms enable wider use of GPGPUs – CUDA from NVidia and OpenCL from an industry consortium

- A heterogeneous system has a regular host CPU and a GPU that handles (say) CUDA code (they can both be on the same chip)

GPUs

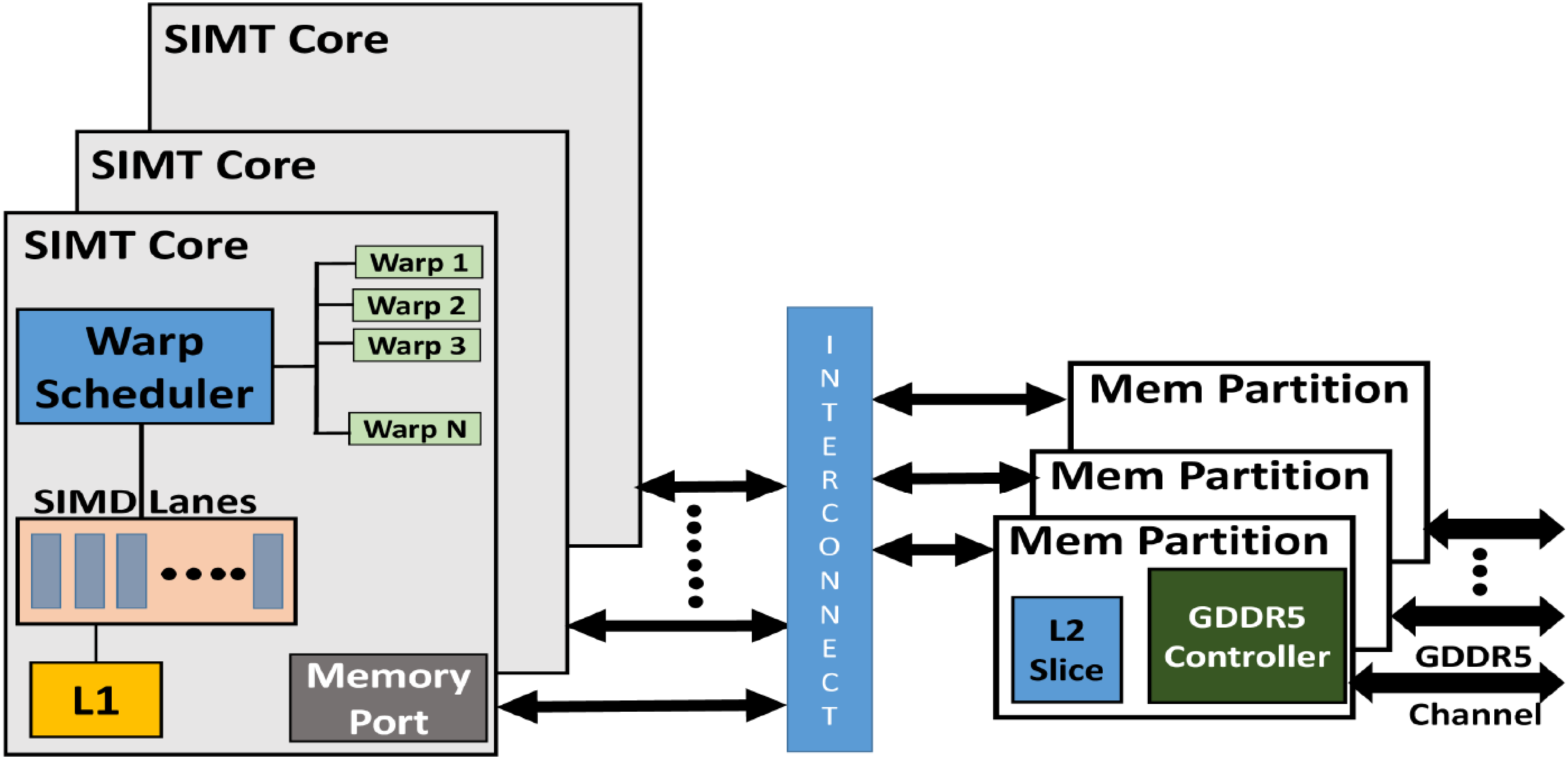
The GPU Architecture

- SIMT – single instruction, multiple thread; a GPU has many SIMT cores

- A large data-parallel operation is partitioned into many thread blocks (one per SIMT core); a thread block is partitioned into many warps (one warp running at a time in the SIMT core); a warp is partitioned across many in-order pipelines (each is called a SIMD lane)

- A SIMT core can have multiple active warps at a time, i.e., the SIMT core stores the registers for each warp; warps can be context-switched at low cost; a warp scheduler keeps track of runnable warps and schedules a new warp if the currently running warp stalls

The GPU Architecture



Architecture Features

- Simple in-order pipelines that rely on thread-level parallelism to hide long latencies

- Many registers (~1K) per in-order pipeline (lane) to support many active warps

- When a branch is encountered, some of the lanes proceed along the “then” case depending on their data values; later, the other lanes evaluate the “else” case; a branch cuts the data-level parallelism by half (branch divergence)

- When a load/store is encountered, the requests from all lanes are coalesced into a few 128B cache line requests; each request may return at a different time (mem divergence)

GPU Memory Hierarchy

- Each SIMT core has a private L1 cache (shared by the warps on that core)
- A large L2 is shared by all SIMT cores; each L2 bank services a subset of all addresses
- Each L2 partition is connected to its own memory controller and memory channel
- The GDDR5 memory system runs at higher frequencies, and uses chips with more banks, wide IO, and better power delivery networks
- A portion of GDDR5 memory is private to the GPU and the rest is accessible to the host CPU (the GPU performs copies)

Thank you!