LECTURE 7

Core Multithreading

Miquel Pericàs

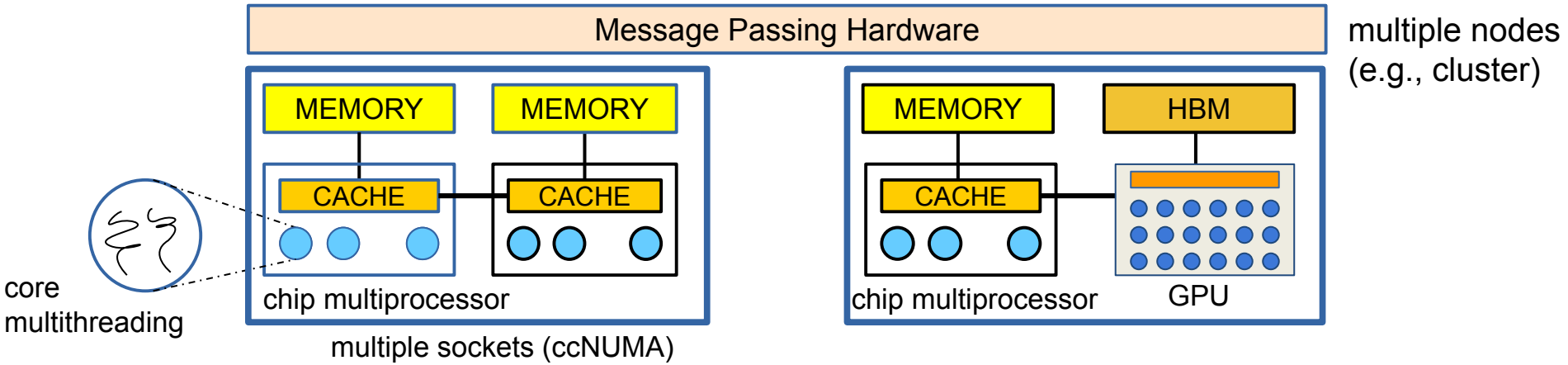
EDA284/DIT361 - 2019/2020

What's cooking

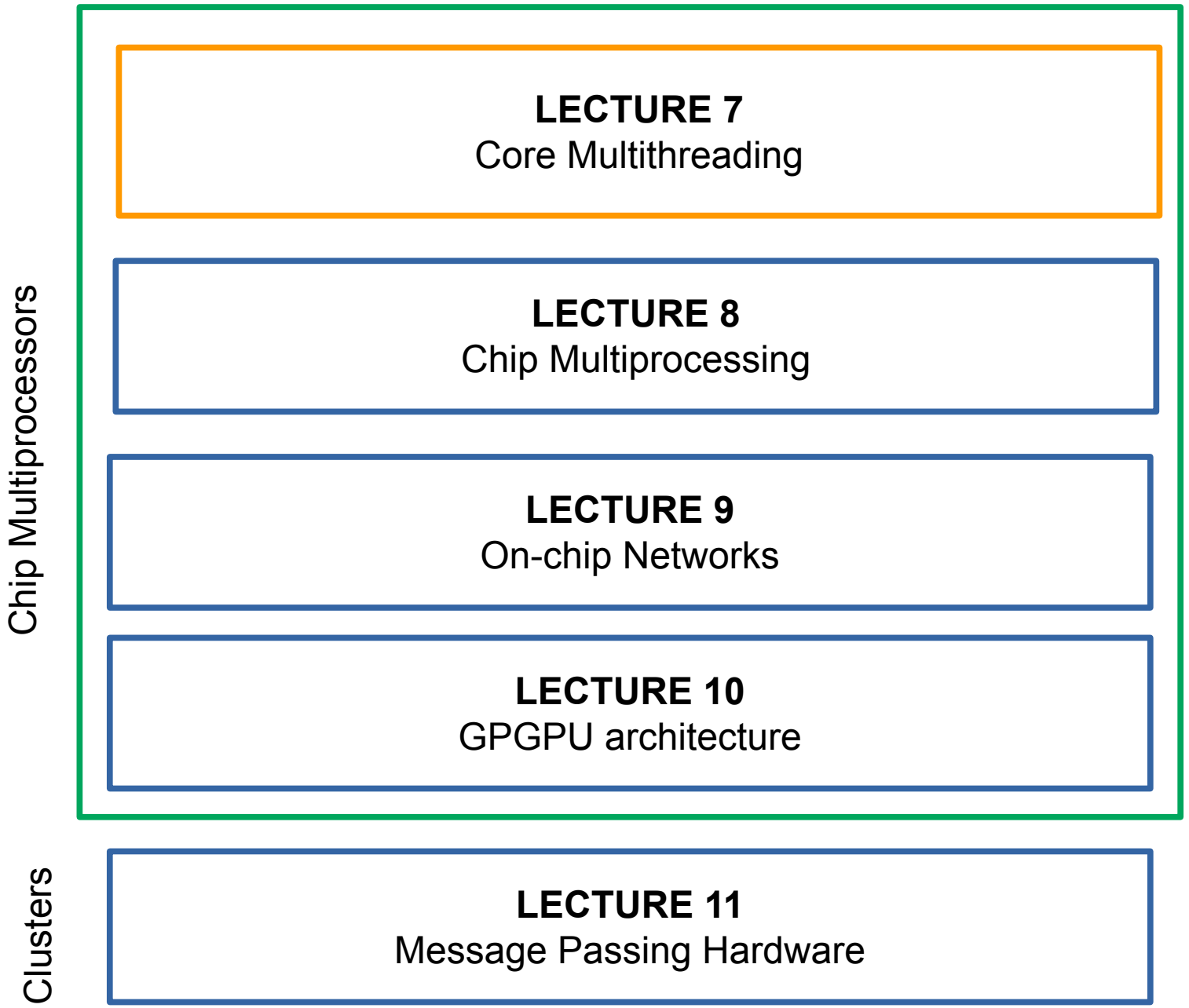
1. **Lectures**
   - Today: **Core Multithreading**
   - Tomorrow: **Chip Multiprocessors** (8h-10h)
   - Friday: **GEM5** Lab session (8h-12h @ ED3507)
2. **EDA284 project**
   - Submission instructions have been published
   - 1st deadline February 19th - Design teams submit first version to reviewers
     - TBD via Canvas
3. **EDA284 Lab #1**
   - Instructions have been published
   - To be completed in pairs (choose your partner)
   - **Submission Deadline** for Lab #1 is Feb 27th 23:59h

CMPs: new opportunities

- **Compared to traditional shared-memory:** more on-chip components + drastically reduced communication overheads
  - This enables new architectures and new programming paradigms
- **Systems with huge amount of threads can be built by exploiting parallelism at all levels:** system, processors, and cores



Lectures 7 - 10 Overview

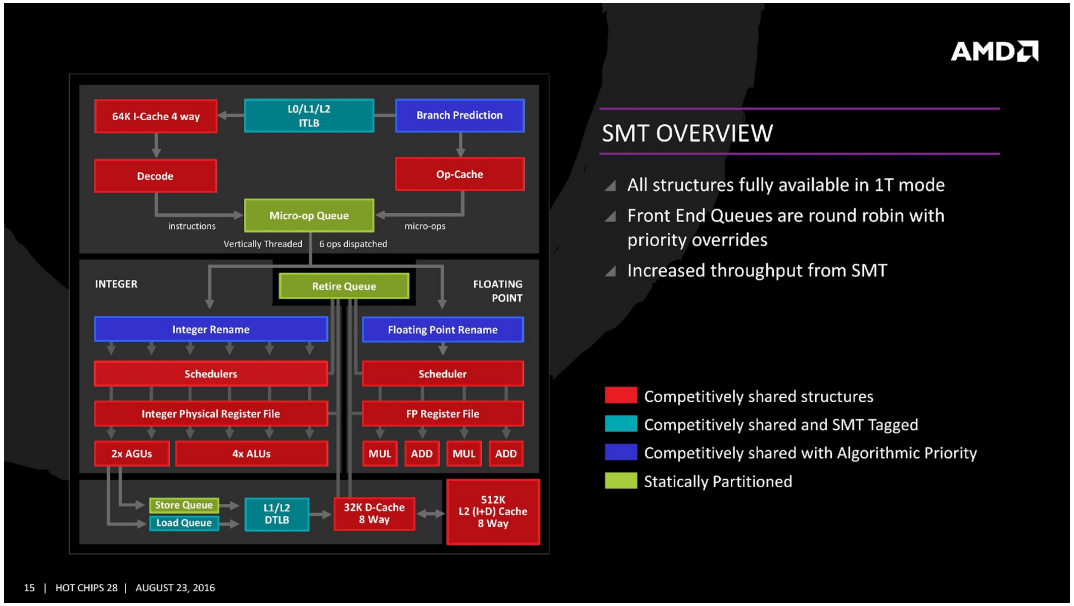


OUTLINE (Lecture 7)

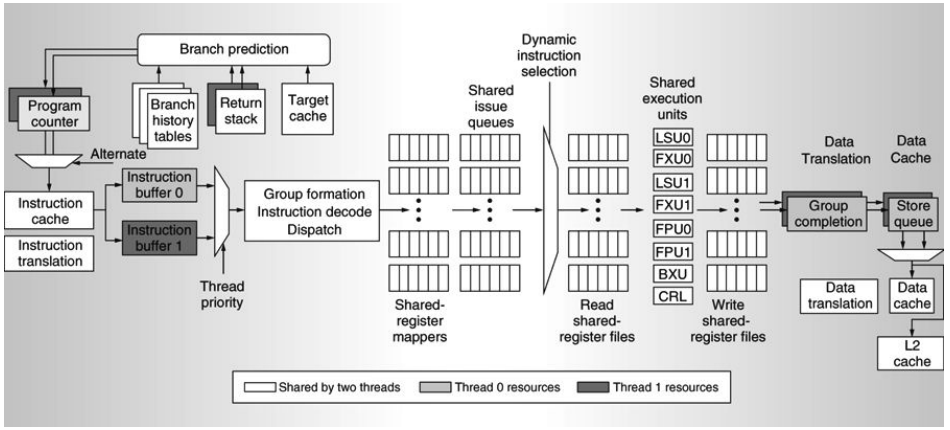
- Why Multithreading
- Block Multithreading
- Interleaved Multithreading
- Simultaneous Multithreading

Multithreading

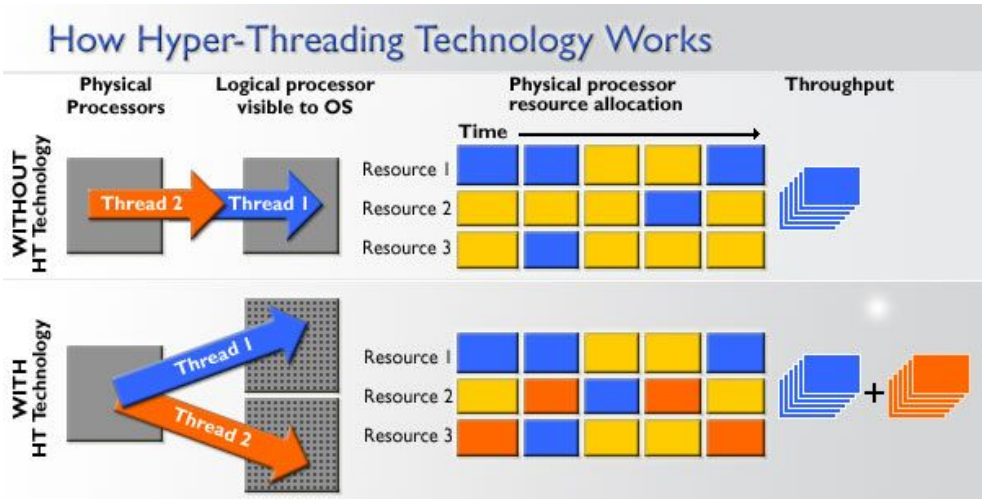
AMD Zen SMT



IBM POWER 5 SMT



Intel Hyperthreading



SUN NIAGARA Multithreading

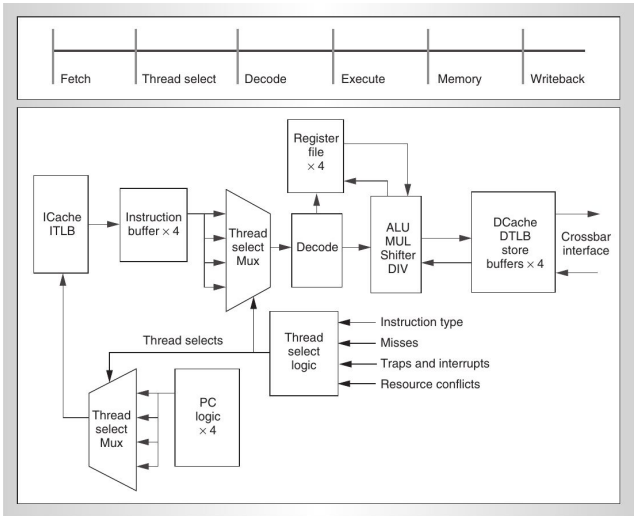


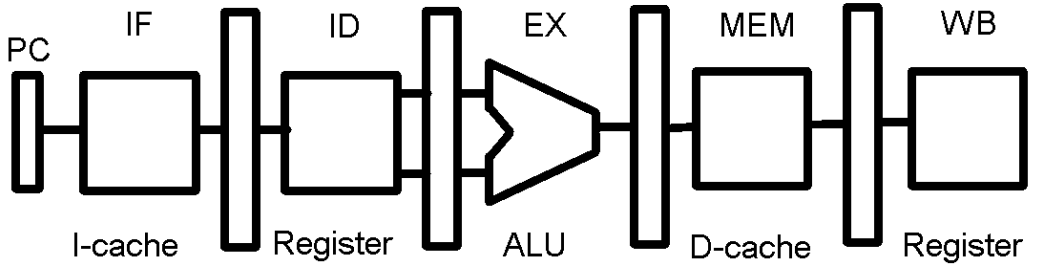
Figure 3. Sparc pipeline block diagram. Four threads share a six-stage single-issue pipeline with local instruction and data caches. Communication with the rest of the machine occurs through the crossbar interface.

and GPGPUs!

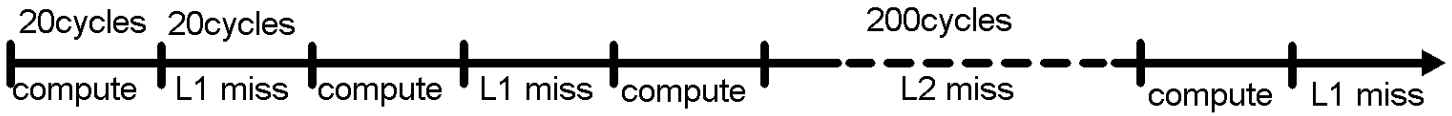
Why Core Multithreading??

Cache miss in the 5-stage pipeline

- **On a miss, the processor clock is stopped, the miss is handled and then the clock is restarted**



IDEA: *make use of idle resources via* **multithreading**

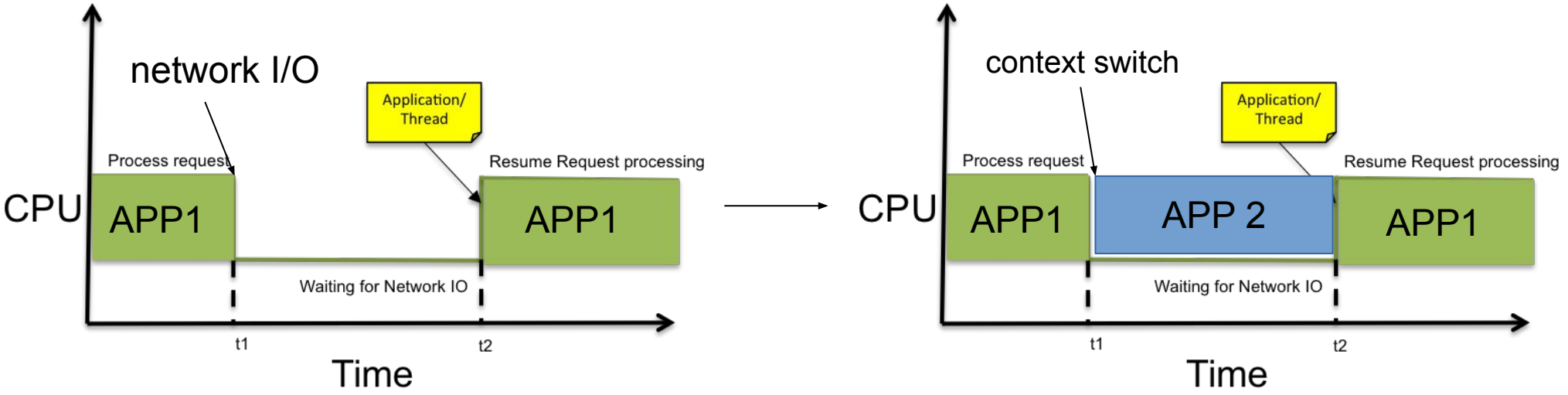


- **EXAMPLE: 20 cycles L1-miss latency/200 cycles L2 miss latency**
  - CPI without cache misses (CPI_0) is 1
  - L1- MPI (miss per instruction) is .05; L2-MPI is .005
- Q: What is the time to execute 200 instructions, with and without cache misses?
- **Without misses**: 200 instructions → 200 cycles
- **With misses**: 200(inst)*1 (CPI) + 200*0.05*20 + 200*0.005*200 = 200 + 200 + 200 = 600 cycles

CPI degrades 3x -> microprocessor resources severely underutilized!

Origins: Software Multithreading

- **Used since the 1960's to hide the latency of I/O operations**
  - Multiple processes or threads are active
    - Virtual memory space allocated
    - Process control block allocated
  - On an I/O operation
    - Process is preempted and removed from ready list
    - I/O operation is started
    - In the meantime, another active process is picked from the ready list and run
  - When I/O completes, put the preempted process back in the ready list



http://venkateshcm.com/2014/04/Reactor-Pattern-Part-1-Non-blocking-I-O/

How software multithreading works

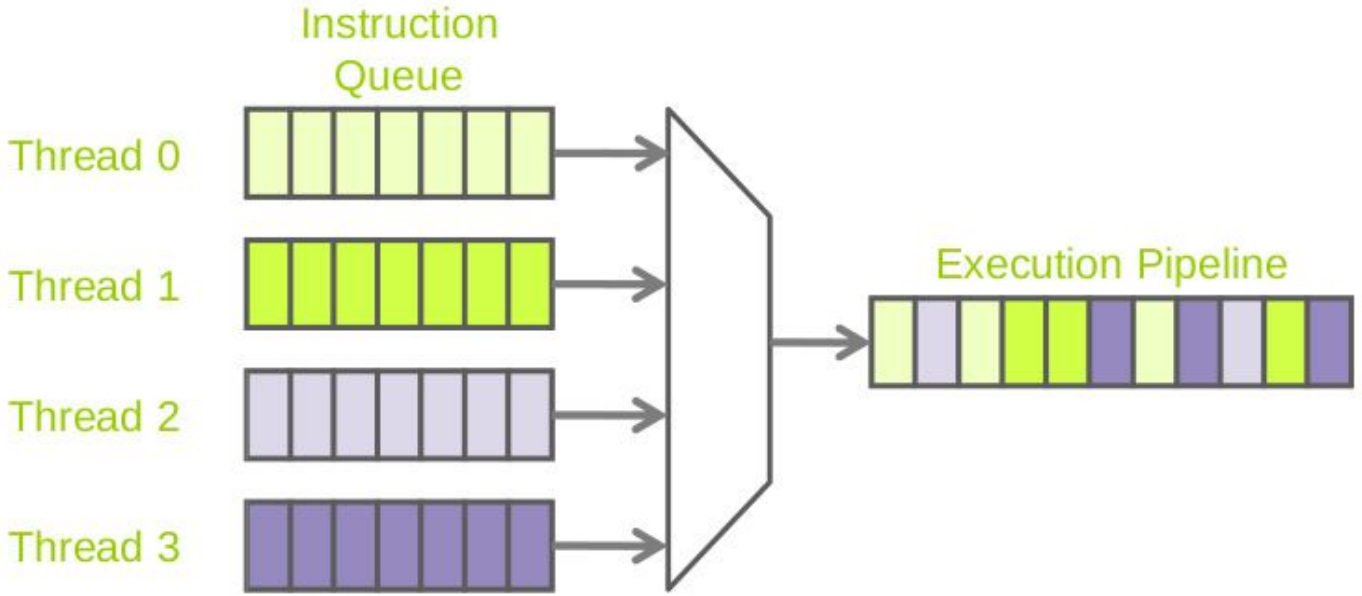
- **Context switch**
  - Trap processor--flush pipeline
  - Save process state in process control block
    - Includes register file(s), PC, interrupt vector, Page Table Base Register, etc
  - Restore process state of a different process
  - Start execution--fill pipeline
  - Order of microseconds on modern systems
- **Also triggered on**
  - Shared resource conflict (e.g., Semaphores)
  - Timer interrupts (fairness)

Very high switching overhead. Ok, since wait is very long.

Can we apply this to handle cache misses? How?

Hardware Multithreading

- **Run multiple threads on the same core at the same time**
- **Fetch instructions from another thread when a thread is blocked on:**
  - Unsuccessful synchronization
  - L1 or L2 cache misses
  - TLB misses
  - Exceptions
  - Even while waiting for operands (latency of operation)



http://key-to-programming.blogspot.com/2016/06/multi-threading-in-java-source-code.html

Hardware Multithreading

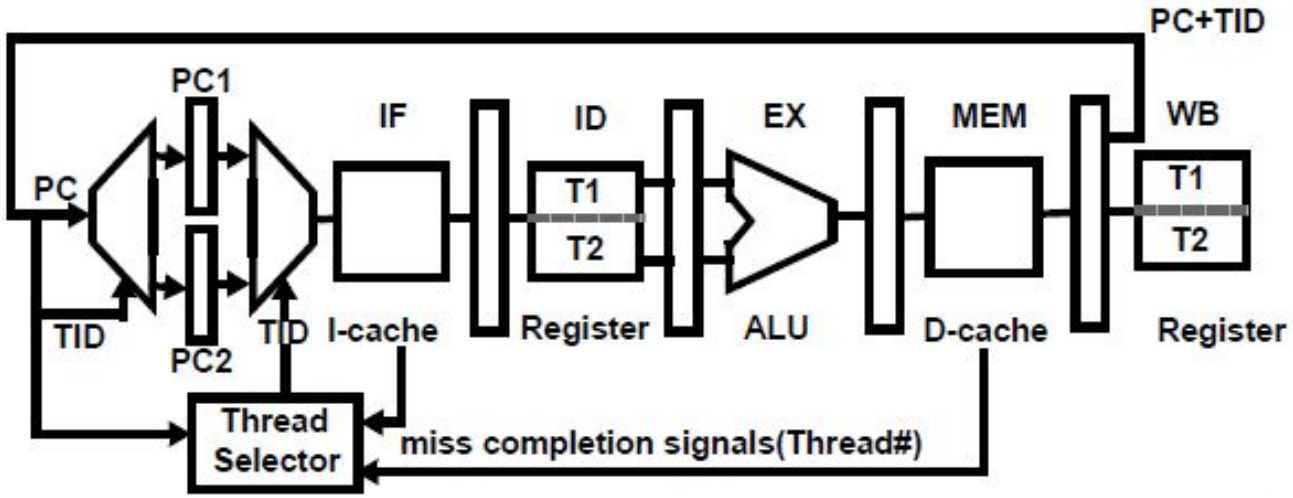
- **Minimum required hardware support: replicate architectural state**
  - All running threads must have their own thread context
    - Multiple register sets in the processor
    - Multiple state registers (PC, PTBR, IV, CF, etc..)*
  - They must not be allowed to observe each others data

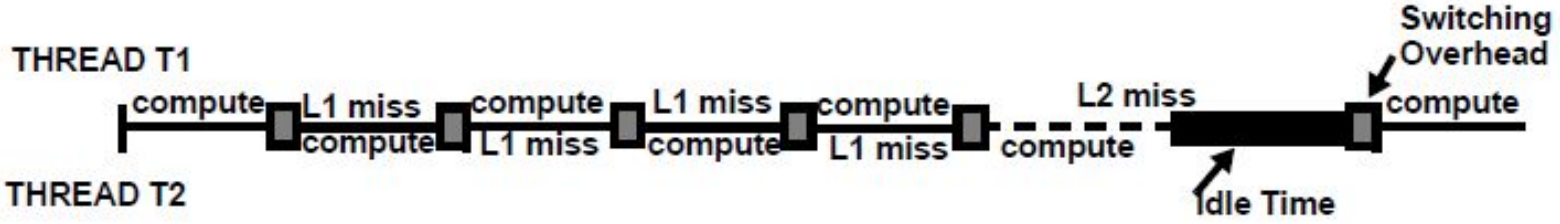
- **Three types of hardware multithreading:**
  - **Block multithreading** aka coarse-grain multithreading
  - **Interleaved multithreading** aka fine-grain multithreading
  - **Simultaneous multithreading**

(*) PC = Program Counter, PTBR = Page Table Base Register, IV = Interrupt Vector, CF = Condition Flags

Block (Coarse-grain) Multithreading

- **Each running thread executes in turn until a long latency event (cache miss)**
  - Similar to software multithreading but at a different scale
  - Only one thread in the pipeline at the same time. *Where is the parallelism?*
- **Five stage pipeline with two threads:**





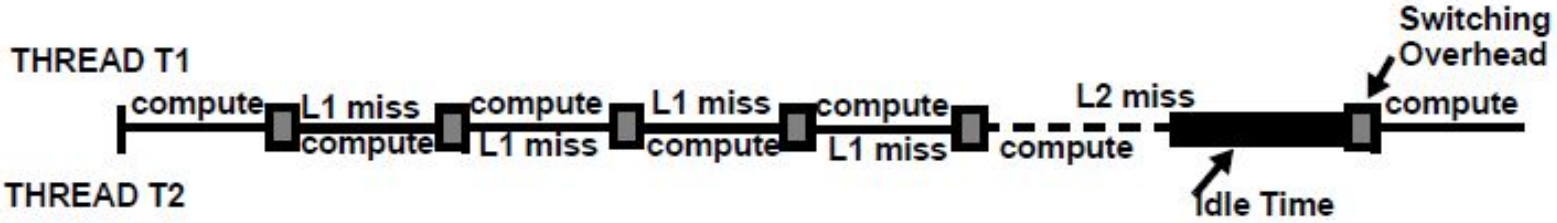
- TID := thread ID, PC := program counter. Seen as two independent CPUs by the OS
- Each context switch due to L1 miss causes overhead to **flush and refill pipeline**
  - 15 cycles compute, 5 cycles to refill pipeline, overlapped with 20 cycles L1 miss
  - cache miss is handled in writeback stage

Block multithreading (Five stage pipeline)

- **Both L1 and L2 must be lockup-free**
  - Must handle two cache accesses (one hit and one miss, or two misses)

- **Use more threads to cover more idle times**
  - more state replication
  - more complex thread selection
  - scale up TLB and cache sizes to accommodate working sets
  - diminishing returns, limited number of runnable threads

- **Fictive timeline in previous example**
  - cache misses happen at highly variable times
  - latencies are variable
  - overlap is never as perfect as in the example

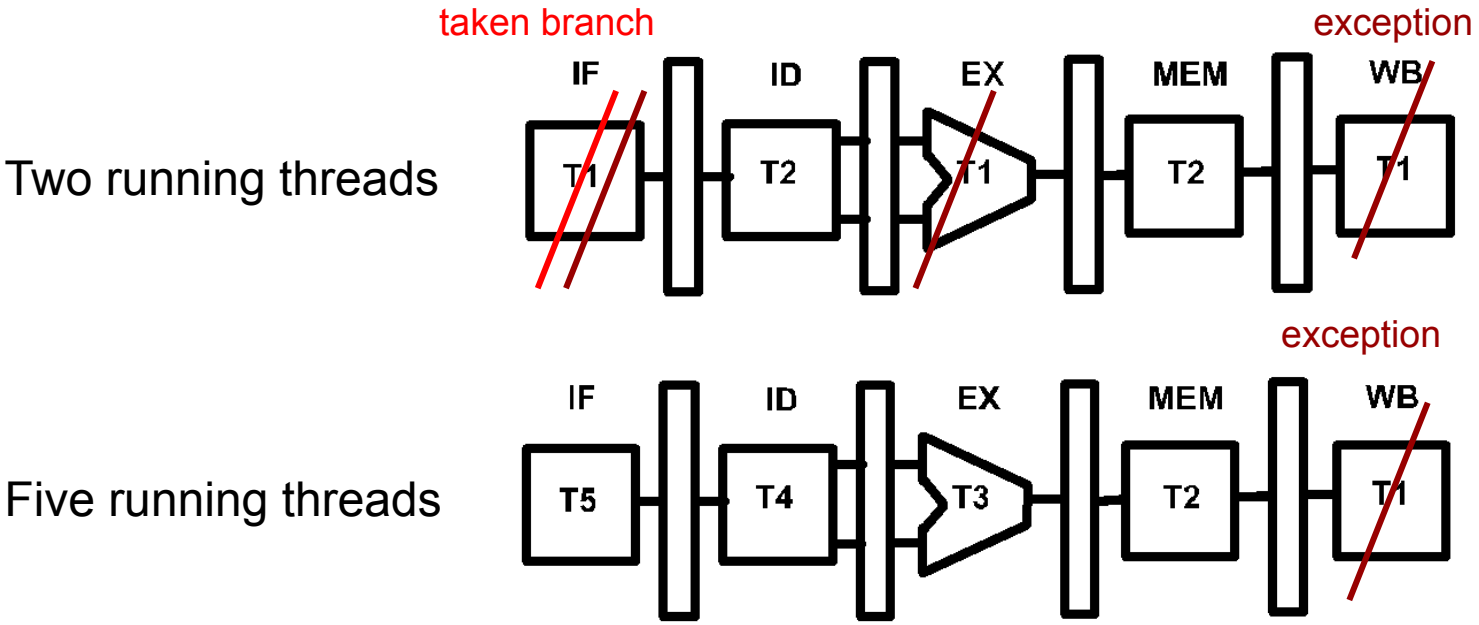


Block multithreading examples

- **IBM RS64-II (1998)**
  - called HMT (hardware MT)
  - 4-way superscalar IO (in-order) processor with a 5-stage pipeline
  - designed for commercial workloads
  - two threads: foreground and background
  - switch threads on cache misses + time-out mechanism
- **INTEL MONTECITO (2006)**
  - called TMT (temporal MT)
  - two EPIC (in-order) cores with two threads per core
  - Events: L3 cache misses/data return, expiration of quantum, thread switch hint provided by software (instruction that forces the thread to yield the core)
  - thread urgency level based on occurrence of events
  - thread switching occurs when the urgency level of suspended thread is higher than that of the running thread
- **Think:** what are the problems of block multithreading?
- How can we reduce the costly pipeline flushes?

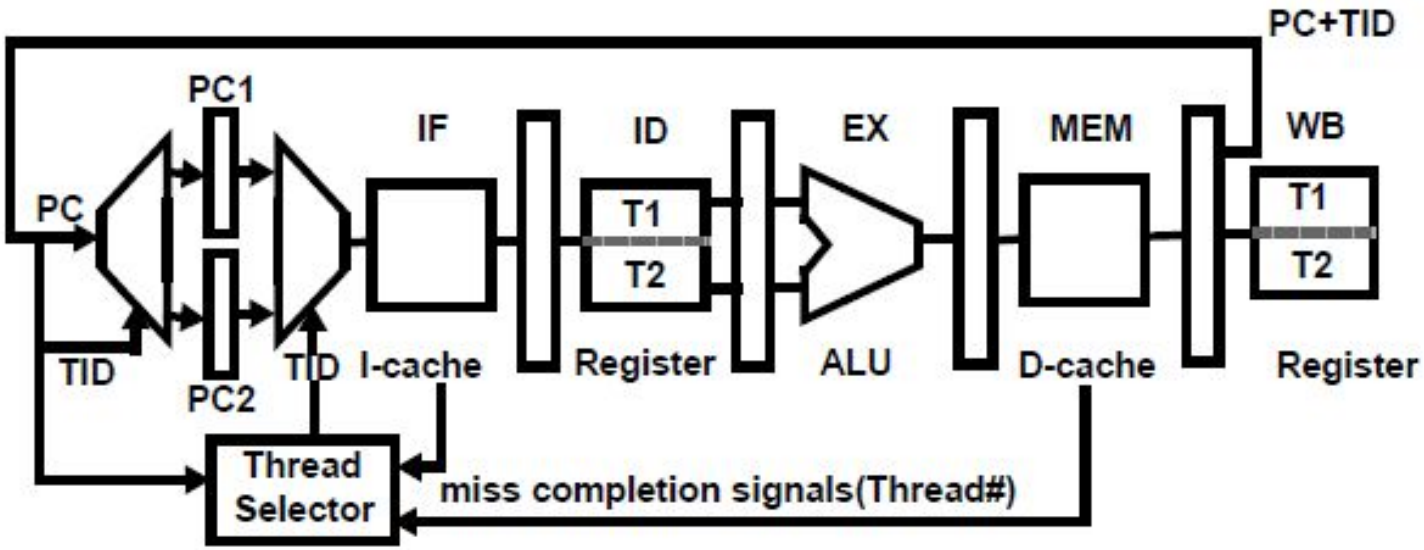
Interleaved (fine-grained) multithreading

- **Dispatch instructions from different threads in each cycle**
  - different ready threads dispatch in turn in every cycle
  - hides small latencies, such as instruction latencies causing pipeline bubbles (in addition to long latencies)



- **Two threads:** penalty of a taken branch is one clock and penalty of an exception is three clocks.
- **Five threads:** penalty of a taken branch is zero and penalty of an exception is one clock.

Interleaved multithreading

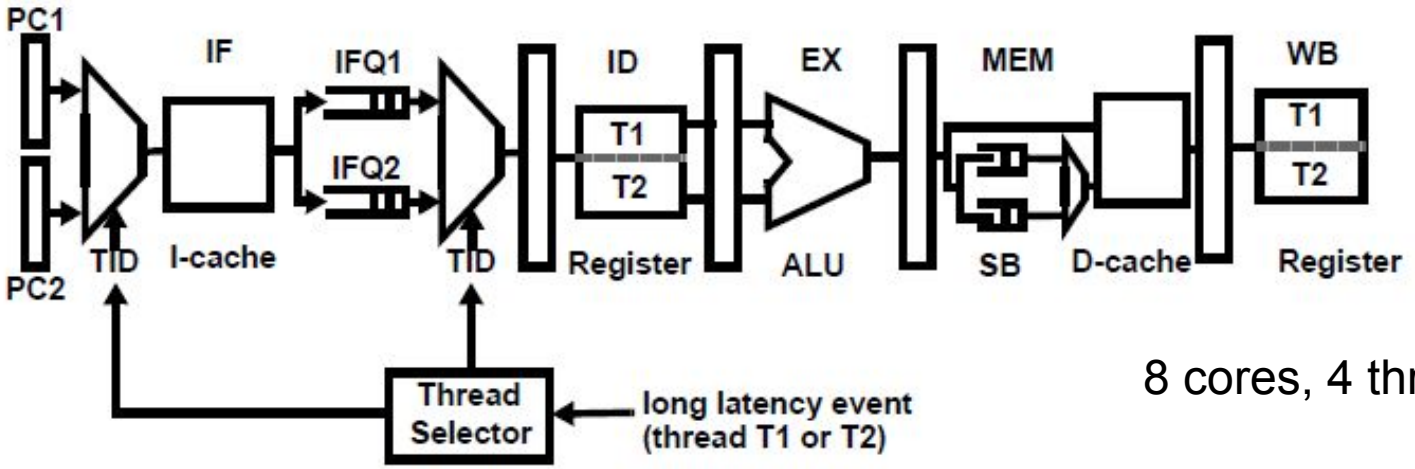


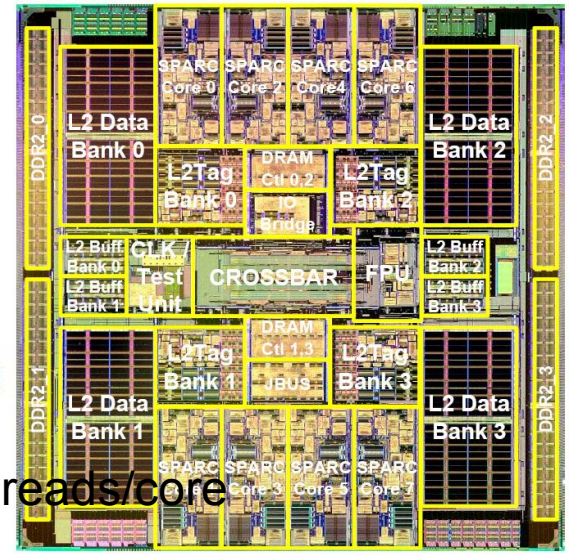
- **<u>Multiple threads can be active in the pipeline</u>**

- **Same architecture as for block multithreading except:**
  a. Data forwarding must be thread aware, context ID is carried by forwarded values
  b. Stage flushing must be thread aware. On an miss exception, taken branch or software exception IF, ID, EX and MEM cannot be flushed indiscriminately
  c. Thread selection algorithm is different
     - a different thread is selected in each cycle (round-robin)
     - on a long latency event the selector puts the thread aside and removes it from selection

Interleaved multithreading - Example

- **SPARC T1 AND T2 ("NIAGARA", 2005)**



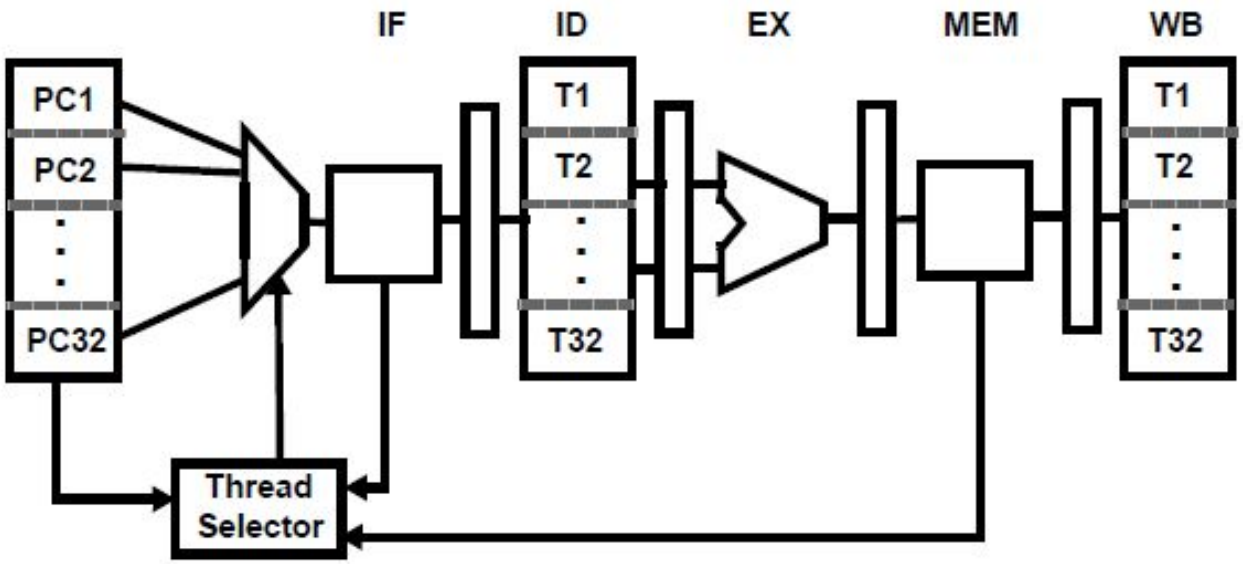


8 cores, 4 threads/core

- **In-order pipeline with add-ons:** thread selection stage; store buffers; IFQ & architectural state replicated to manage four threads
- **IFQ to smooth out instruction delivery when I-cache misses**
- **Thread selector selects the thread to fetch/decode in every cycle**
  - Typically round-robin
  - If long latency event, the selection of the thread is suspended
- **Flushing and forwarding are thread aware**
- **Q: Can we simplify this hardware?**

Barrel Processors

- **Enough threads so that the pipeline is filled with instructions from different threads**
  - **no need to forward or to detect hazards!**
  - there can be so many ready threads that there is no need for a cache
    - or cache can be very large with high hit latency
  - control hazards are also solved by multithreading
  - high throughput but very low single thread performance



Examples of Barrel processors

- **DENELCOR HEP (EARLY 1980s)**
  - up to 16 processors
  - 8-stage pipeline
  - different threads in the pipeline (needs at least 8 threads)
  - no forwarding, no hazard detection unit, no stalling and no flushing
  - no cache
  - throughput for one thread 1.25 MIPS. Eight threads: 10 MIPS

Examples of barrel processors

- **TERA MTA (1999)**
  - MP with up to 256 processors
  - 128 I-streams per processor, 128 PCs and 4096 registers
  - No hardware support for data hazards:
    - an instruction in an I-stream can issue if it has no dependencies with previous instructions
    - a lookahead field is added to every instruction. It indicates the number of following instructions that have no dependency with it
    - multiple (independent) instructions from the same thread can be in flight (unlike HEP)

How to provide both high single-thread performance and high resource utilization?

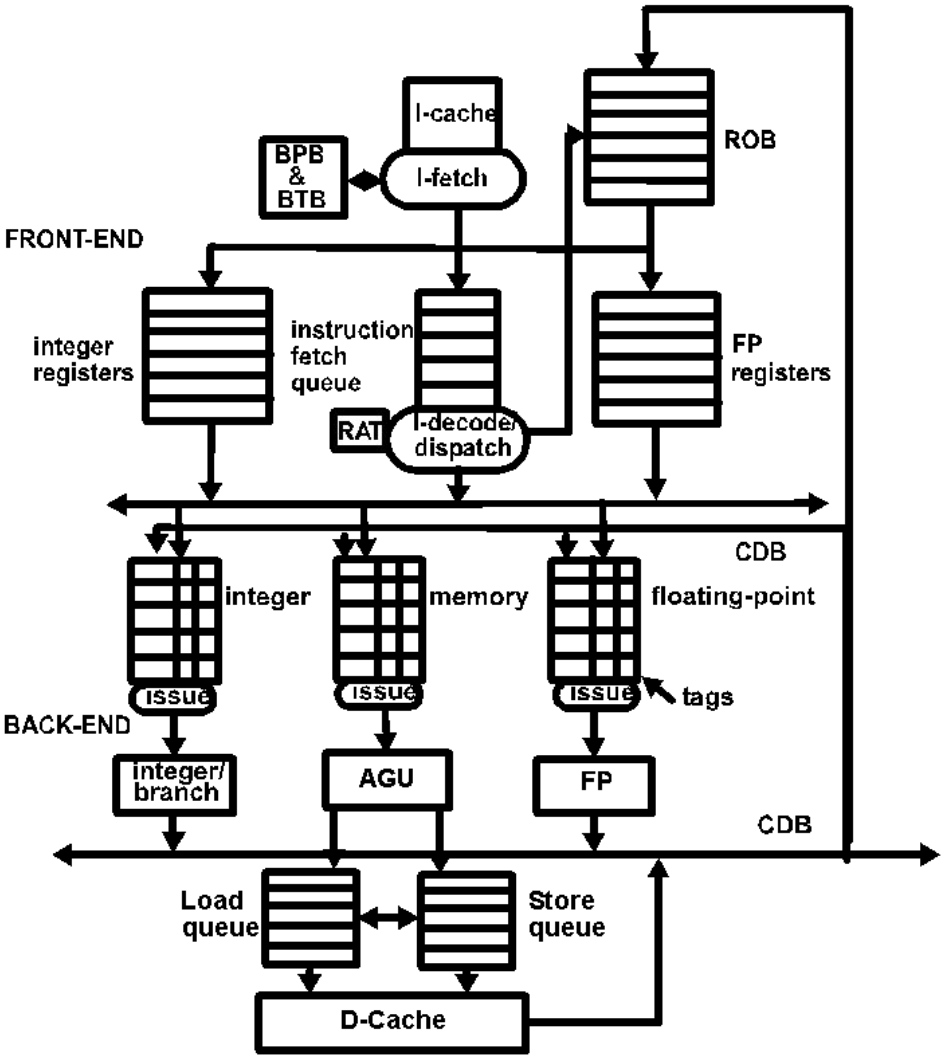
1. Use Out-of-Order execution (OoO) for single-thread processing
2. Use interleaved multithreading for improved efficiency

Simultaneous Multithreading (SMT)

OoO Pipeline with Speculative Execution

Instruction lifetime in Tomasulo algorithm with speculative execution:

1. **In-order** Fetch and Dispatch
2. **Out-of-Order** Execution and WriteBack
3. **In-order** Commit via ReOrder Buffer (ROB)



NEW STRUCTURES:

- REORDER BUFFER (ROB)
- BRANCH PREDICTION BUFFER (BPB)
- BRANCH TARGET BUFFER (BTB)

ROB:

- KEEPS TRACK OF PROCESS ORDER (FIFO)
- HOLDS SPECULATIVE RESULTS
- NO MORE SNOOPING BY REGISTERS

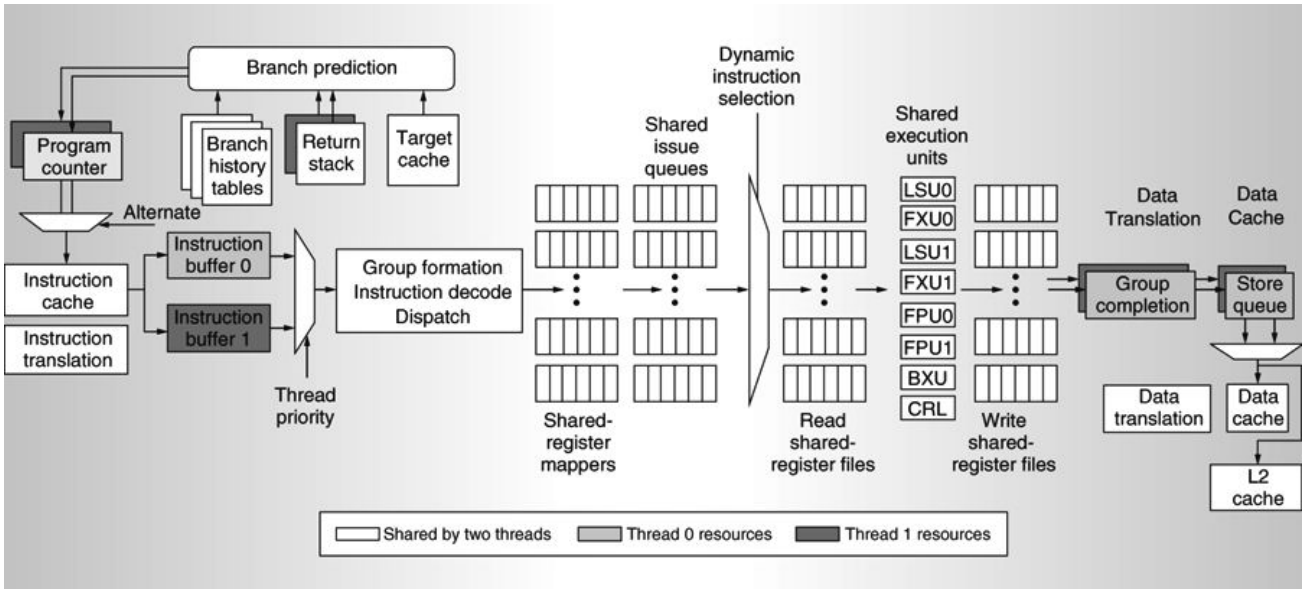
REGISTER VALUES

- PENDING IN BACK-END
- SPECULATIVE IN ROB
- COMMITTED IN THE REGISTER FILE

USE ROB ENTRY # AS TAG TO RENAME REGISTER

Simultaneous Multithreading (SMT)

- **Dispatch instructions from different threads in consecutive cycles**
  - If superscalar, may dispatch from different threads in the same cycle
  - Long latency events redirect dispatch to other threads



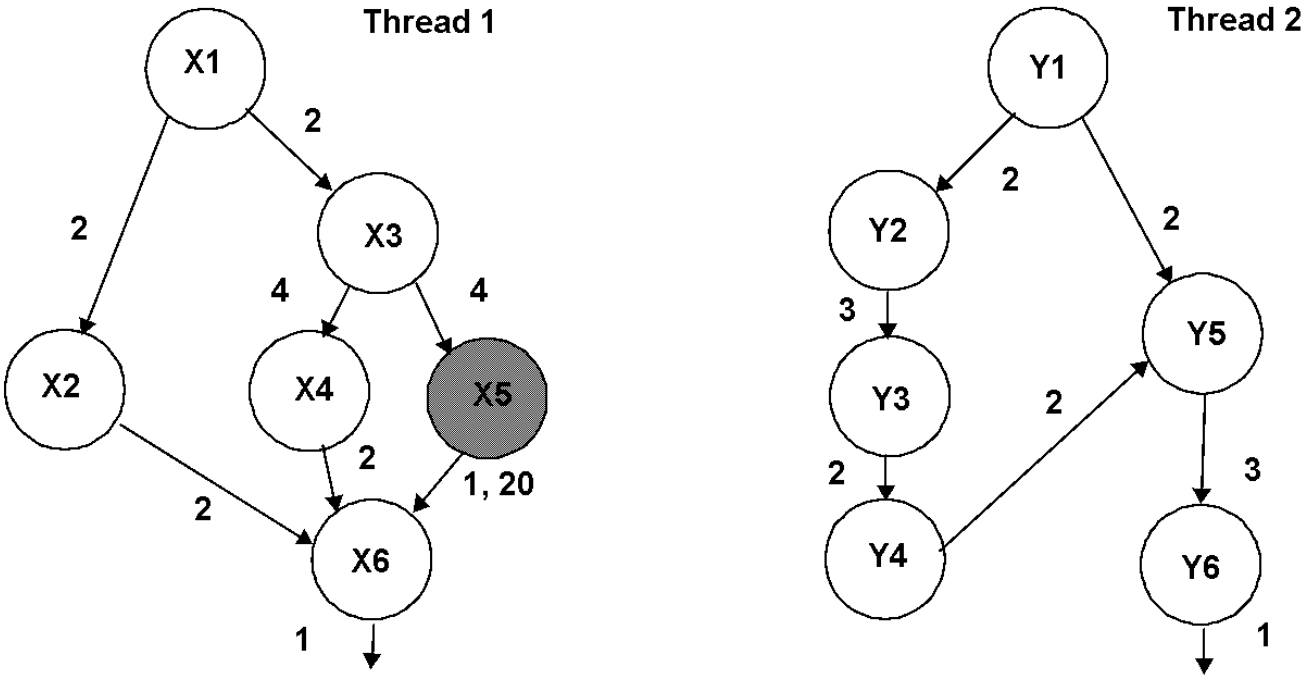
IBM POWER5 SMT pipeline

- Dispatching, scheduling, flushing and forwarding must be thread aware
- In addition to architectural state, must typically replicate: **ROB** ("*Group completion",* IBM POWER5), **IFQ** ("Instruction Buffer"), **SB** ("Store Queue")
- **A miss does not trigger an exception**
  - It simply waits in the LSQ & ROB
  - No flushing of instructions in the back-end

Block multithreading vs SMT in OoO cores

- **COST OF SWITCHING THREADS WITH BLOCK MULTITHREADING**
  1. complete all instruction prior to event in thread order
  2. flush all instructions following the event

- **EXAMPLE**



- **THREAD 1 EXPERIENCES A LONG LATENCY EVENT AT INST X5**
  - thread switch is triggered when X5 reaches the top of ROB
  - must flush large amount of instructions

Block multithreading

- **OoO PROCESSORS**
  - 2-way dispatch, 2-way issue (1 issue Q of size 4), 2 CDBs, 2-way commit

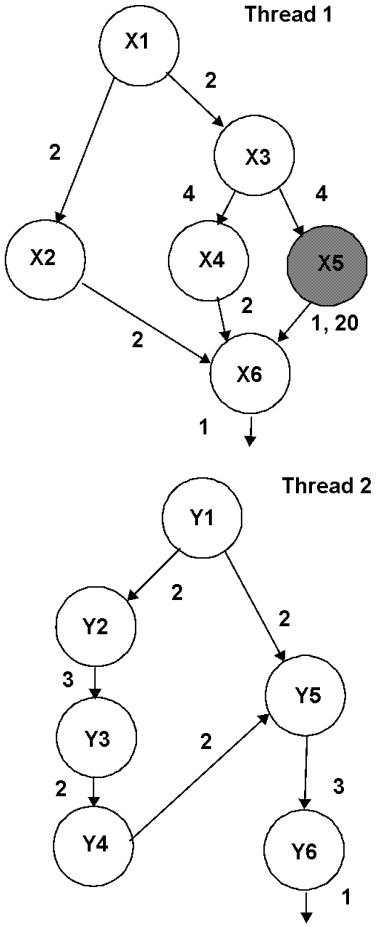
Instruction (latency)	Dispatch (issue Q)	Issue	Register fetch	Exec start	Exec complete	CDB	Commit (T1)	Commit (T2)
X1(2)	1(1)	2	3	4	5	6	7	
X2(2)	1(2)	4	5	6	7	8	9	
X3(4)	2(3)	4	5	6	9	10	11	
X4(2)	2(4)	6	7	10	11	12	13	
~~X5(1,20)~~	~~9(3)~~	~~10~~	~~11~~	~~12*~~	~~12~~	~~13~~		
~~X6(1)~~	~~9(4)~~	~~11~~	~~12~~	~~13~~	~~13~~	~~14~~	Flush	
Y1(2)	15(1)	16	17	18	19	20		21
Y2(3)	15(2)	18	19	20	22	23		24
Y3(2)	16(3)	21	22	23	24	25		26
Y4(2)	16(4)	23	24	25	26	27		28
Y5(3)	24(3)	25	26	27	29	30		31
Y6(1)	24(4)	28	29	30	30	31		32
X5(1,1)	32(3)	33	34	35	35	36	37	Replay
X6(1)	32(3)	34	35	36	36	37	38	

CPI = 2.7 (= 32 cycles / 12 instructions)

Execution in two-way OoO processor with SMT

Instruction (latency)	Dispatch	Issue	Register fetch	Exec start	Exec complete	CDB	Commit (T1)	Commit (T2)
X1(2)	1(1)	2	3	4	5	6	7	
Y1(2)	1(2)	2	3	4	5	6		7
X2(2)	2(3)	4	5	6	7	8	9	
Y2(3)	2(4)	4	5	6	8	9		10
X3(4)	7(3)	8	9	10	13	14	15	
Y3(2)	7(4)	8	9	10	11	12		13
X4(2)	10(3)	12	13	14	15	16	17	
Y4(2)	10(4)	11	12	13	14	15		16
X5(1,20)	15(2)	16	17	18*	37	38	39	
Y5(3)	15(3)	16	17	18	20	21		22
X6(1)	17(2)	36	37	38	38	39	40	
Y6(1)	17(3)	19	20	21	21	22		23

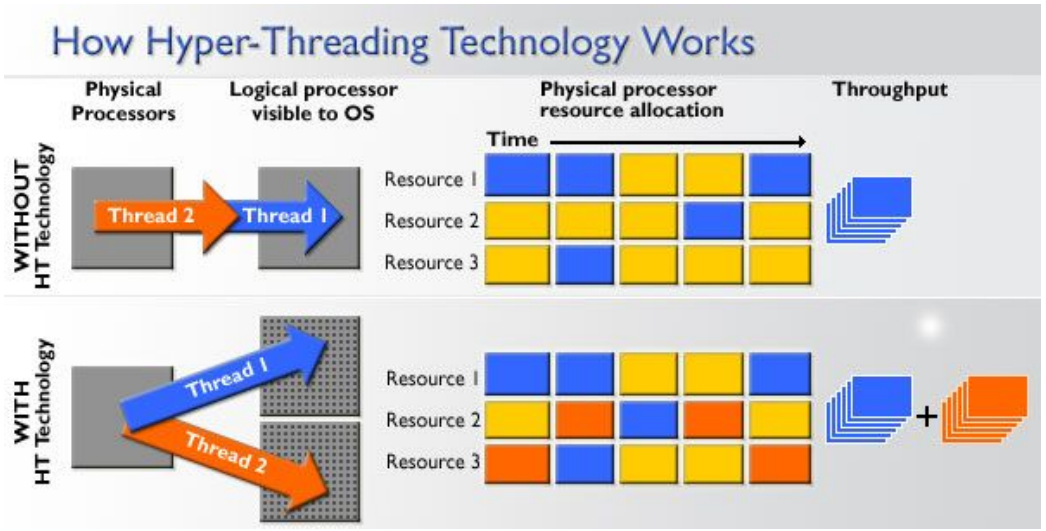
No Flush!



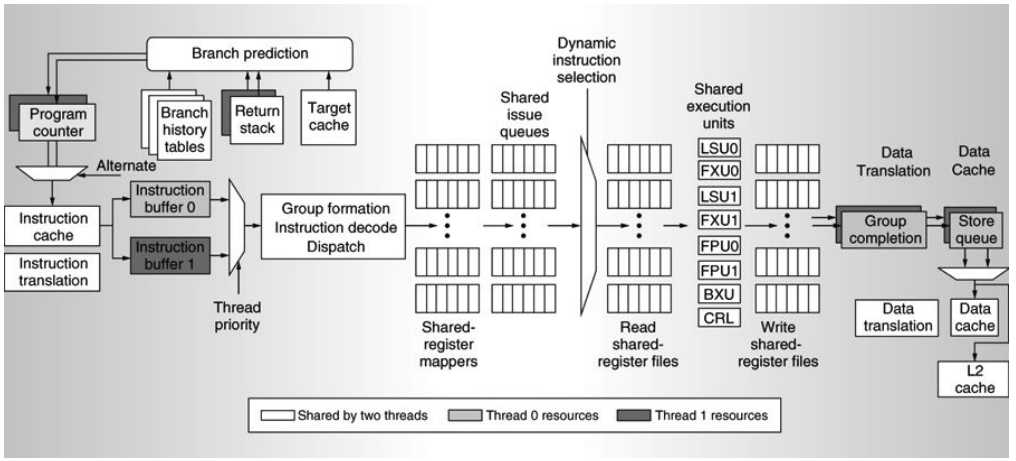
CPI = 17/11 = 1.55, compared to CPI = 2.7 with Block Multithreading

MODERN SMT EXAMPLES

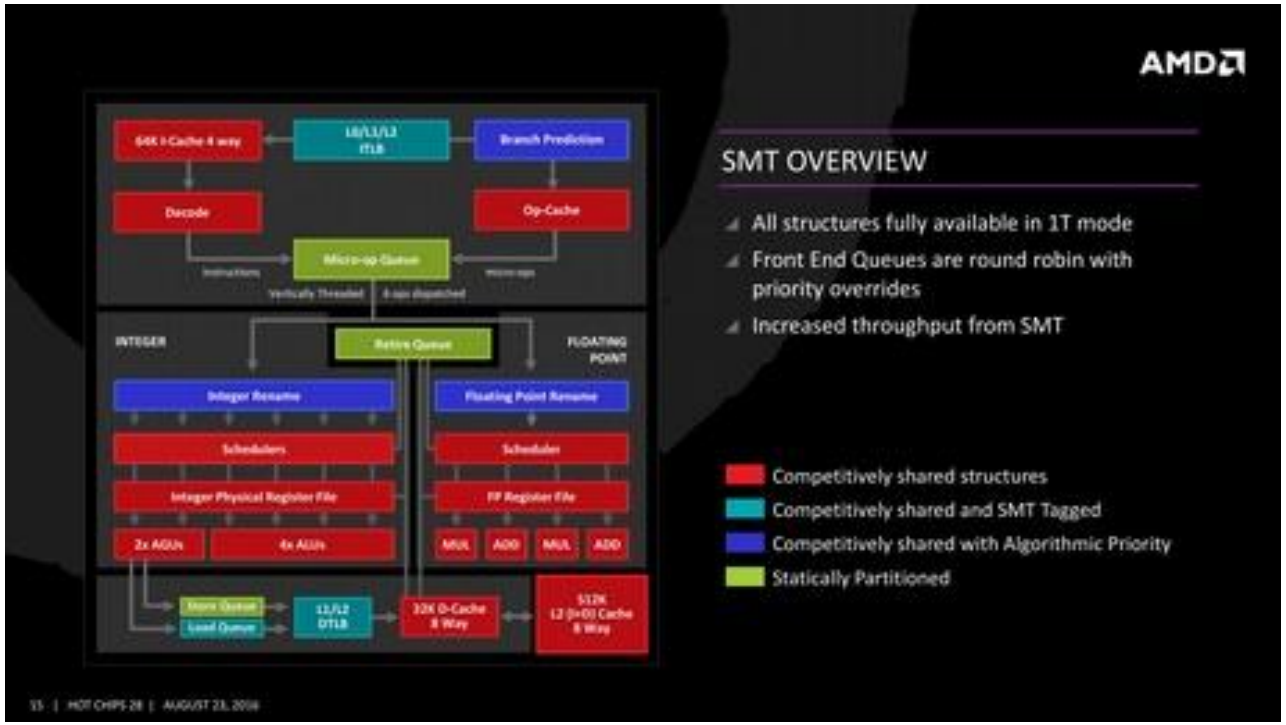
Intel Hyperthreading



IBM POWER 5



AMD Zen / Ryzen



None of these implements "true" SMT:

- No fetch, decode and dispatch instructions from multiple threads in the same cycle
- Guarantee access to all core resources in single-thread mode

Summary: multithreading

- **Block Multithreading**
- **Interleaved Multithreading**
- **Barrel Processors**
- **Simultaneous Multithreading**