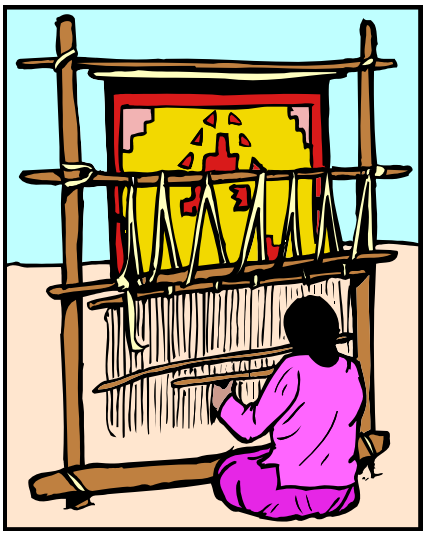
Multithreaded Processors

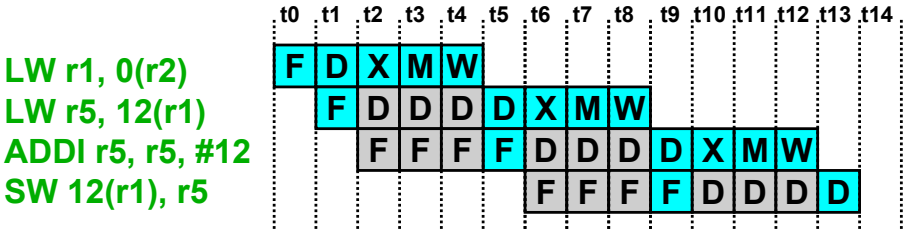


1

Pipeline Hazards

LW r1, 0(r2)
LW r5, 12(r1)
ADDI r5, r5, #12
SW 12(r1), r5

- Each instruction may depend on the next
  - Without bypassing, need interlocks



- Bypassing cannot completely eliminate interlocks or delay slots

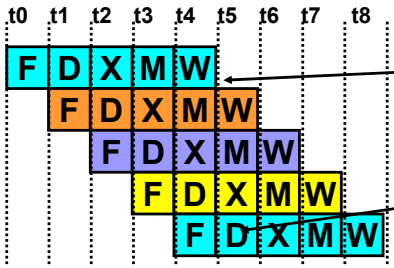
2

Multithreading

- How can we guarantee no dependencies between instructions in a pipeline?
  - One way is to interleave execution of instructions from different program threads on same pipeline

Interleave 4 threads, T1-T4, on non-bypassed 5-stage pipe

T1: LW r1, 0(r2)
T2: ADD r7, r1, r4
T3: XORI r5, r4, #12
T4: SW 0(r7), r5
T1: LW r5, 12(r1)



Last instruction in a thread always completes writeback before next instruction in same thread reads regfile

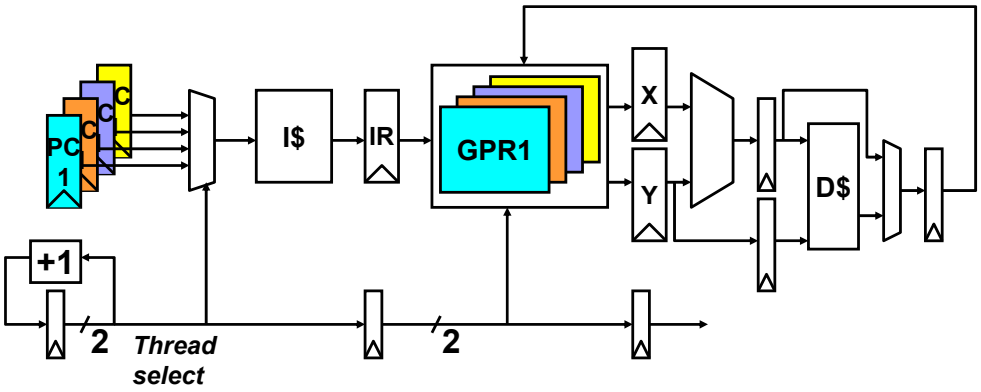
3

CDC 6600 Peripheral Processors
(Cray, 1965)

- First multithreaded hardware
- 10 “virtual” I/O processors
- fixed interleave on simple pipeline
- pipeline has 100ns cycle time
- each processor executes one instruction every 1000ns
- accumulator-based instruction set to reduce processor state

4

Simple Multithreaded Pipeline



- **Have to carry thread select down pipeline to ensure correct state bits read/written at each pipe stage**

5

Multithreading Costs

- **Appears to software (including OS) as multiple slower CPUs**
- **Each thread requires its own user state**
  - **GPRs**
  - **PC**
- **Also, needs own OS control state**
  - **virtual memory page table base register**
  - **exception handling registers**
- ***Other costs?***

6

Thread Scheduling Policies

- **Fixed interleave** ***(CDC 6600 PPUs, 1965)***
  - each of N threads executes one instruction every N cycles
  - if thread not ready to go in its slot, insert pipeline bubble
- **Software-controlled interleave** ***(TI ASC PPUs, 1971)***
  - OS allocates S pipeline slots amongst N threads
  - hardware performs fixed interleave over S slots, executing whichever thread is in that slot
- **Hardware-controlled thread scheduling** ***(HEP, 1982)***
  - hardware keeps track of which threads are ready to go
  - picks next thread to execute based on hardware priority scheme

7

Software-controlled interleave, S > N. Wheel with the threads in each slot. If thread is ready to go

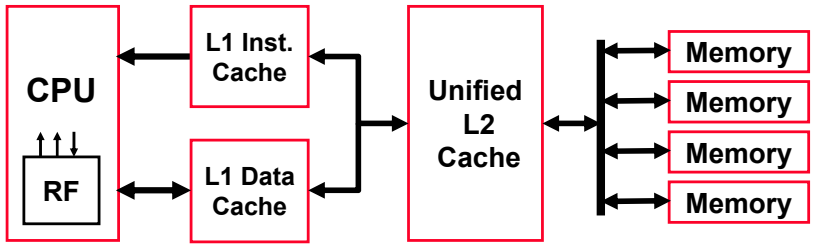
It goes, else NOP, I.e., pipeline bubble.

What "Grain" Multithreading?

- **So far assumed fine-grained multithreading**
  - **CPU switches every cycle to a different thread**
  - ***When does this make sense?***
- **Coarse-grained multithreading**
  - **CPU switches every few cycles to a different thread**
  - ***When does this make sense?***

8

Multithreading Design Choices



- Context switch to another thread every cycle, or on hazard or L1 miss or L2 miss or network request
- Per-thread state and context-switch overhead
- Interactions between threads in memory hierarchy

9

Managing interactions between threads.

Denelcor HEP
(Burton Smith, 1982)

First commercial machine to use hardware threading in main CPU

- 120 threads per processor
- 10 MHz clock rate
- Up to 8 processors
- precursor to Tera MTA (Multithreaded Architecture)

10

Tera MTA Overview

- **Up to 256 processors**
- **Up to 128 active threads per processor**
- **Processors and memory modules populate a sparse 3D torus interconnection fabric**
- **Flat, shared main memory**
  - ➢ **No data cache**
  - – **Sustains one main memory access per cycle per processor**
- **50W/processor @ 260MHz**

11

SGI bought Cray, and Tera was a spin-off.

1997. Integer sort press release.

MTA Instruction Format

- Three operations packed into 64-bit instruction word (short VLIW)
- One memory operation, one arithmetic operation, plus one arithmetic or branch operation
- Memory operations incur ~150 cycles of latency
- Explicit 3-bit "lookahead" field in instruction gives number of subsequent instructions (0-7) that are independent of this one
  - c.f. Instruction grouping in VLIW
  - allows fewer threads to fill machine pipeline
  - used for variable sized branch delay slots
- Thread creation and termination instructions

12

MTA Multithreading

- **Each processor supports 128 active hardware threads**
  - **128 SSWs, 1024 target registers, 4096 general-purpose registers**
- **Every cycle, one instruction from one active thread is launched into pipeline**
- **Instruction pipeline is 21 cycles long**
- **At best, a single thread can issue one instruction every 21 cycles**
  - **Clock rate is 260MHz, effective single thread issue rate is 260/21 = 12.4MHz**

13

32 64-bit general-purpose registers (R0-R31)

unified integer/floating-point register set

R0 hard-wired to zero

8 64-bit branch target registers (T0-T7)

load branch target address before branch instruction

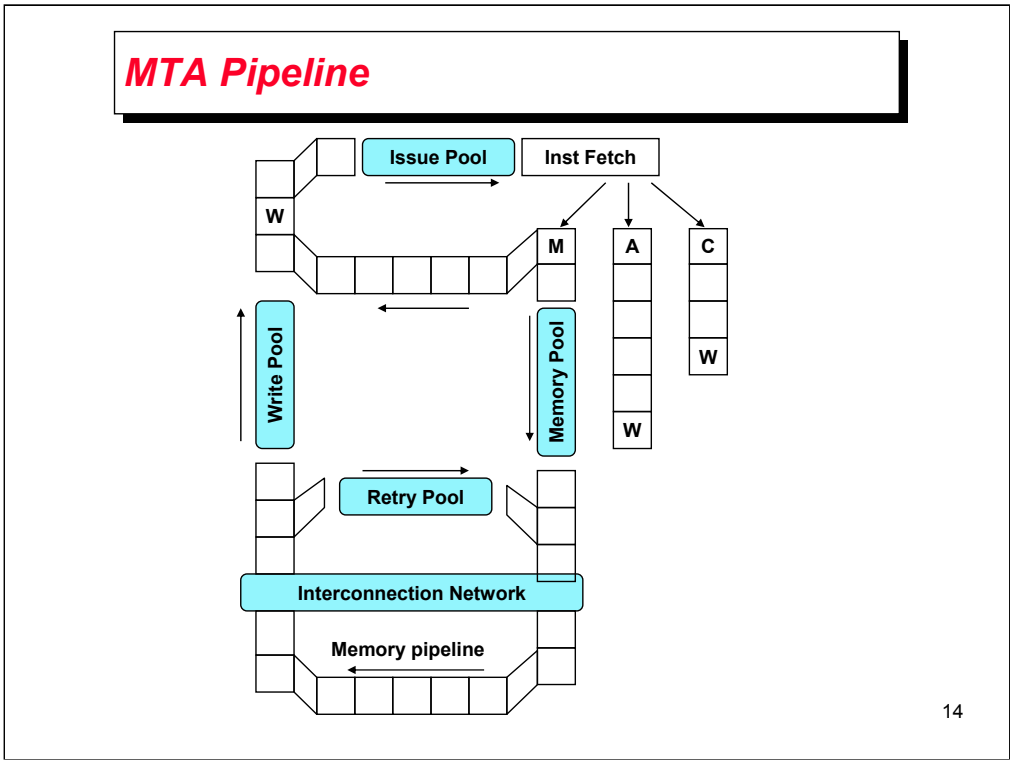
T0 contains address of user exception handler

1 64-bit stream status word (SSW)

includes 32-bit program counter

four condition code registers

floating-point rounding mode



Memory unit is busy, or sync operation failed retry.

Just goes around the memory pipeline.

Coarse-Grain Multithreading

- **Tera MTA designed for supercomputing applications with large data sets and low locality**
  - **No data cache**
  - **Many parallel threads needed to hide large memory latency**
- **Other applications are more cache friendly**
  - **Few pipeline bubbles when cache getting hits**
  - **Just add a few threads to hide occasional cache miss latencies**
  - **Swap threads on cache misses**

15

Tera not very successful, 2 machines sold.

Changed their name back to Cray!

MIT Alewife

- **Modified SPARC chips**
  - **register windows hold different thread contexts**
- **Up to four threads per node**
- **Thread switch on local cache miss**

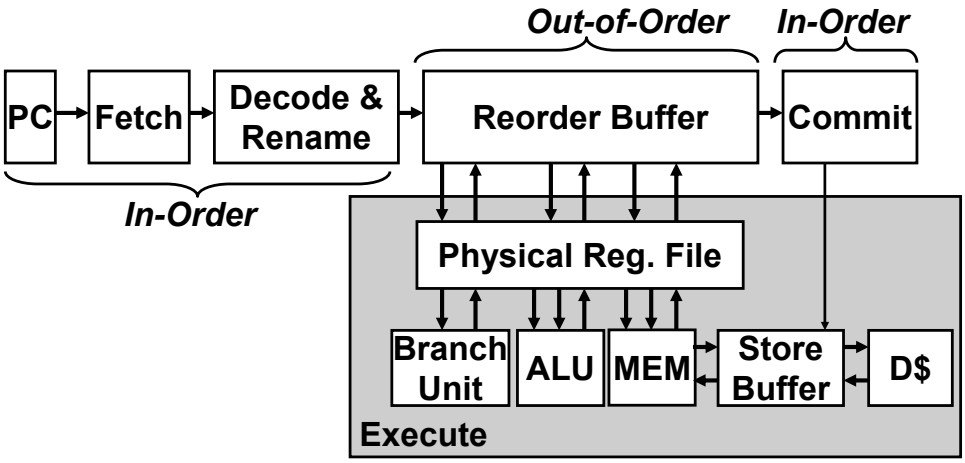
16

IBM PowerPC RS64-III (Pulsar)

- **Commercial coarse-grain multithreading CPU**
- **Based on PowerPC with quad-issue in-order five-stage pipeline**
- **Each physical CPU supports two virtual CPUs**
- **On L2 cache miss, pipeline is flushed and execution switches to second thread**
  - **short pipeline minimizes flush penalty (4 cycles), small compared to memory access latency**
  - **flush pipeline to simplify exception handling**

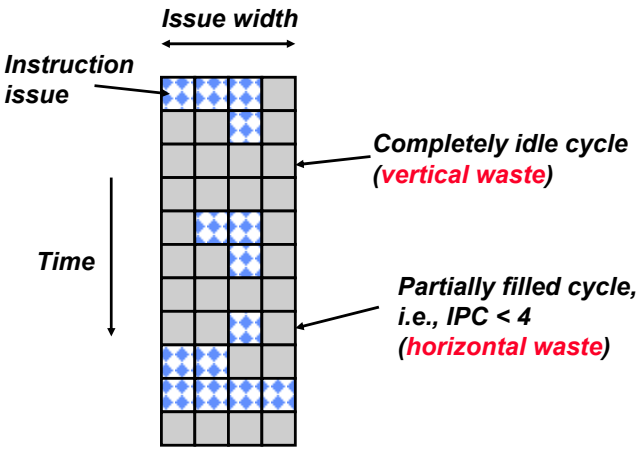
17

Speculative, Out-of-Order Superscalar Processor



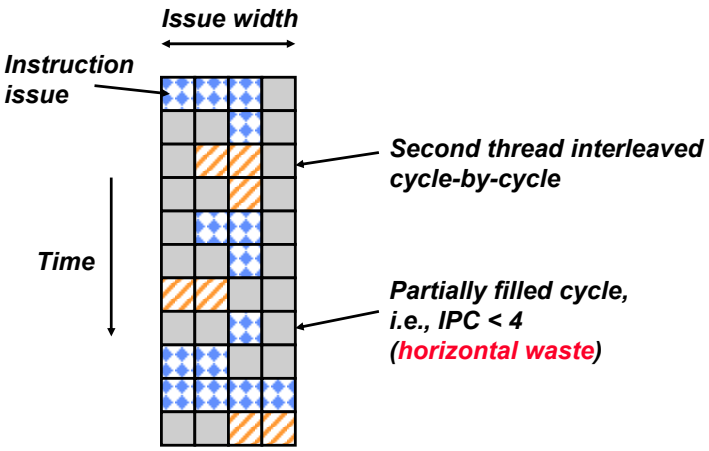
18

Superscalar Machine Efficiency



- *Why horizontal waste?*
- *Why vertical waste?*

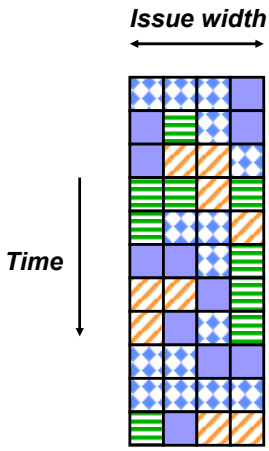
Vertical Multithreading



- **Cycle-by-cycle interleaving of second thread removes vertical waste**

20

Ideal Multithreading for Superscalar



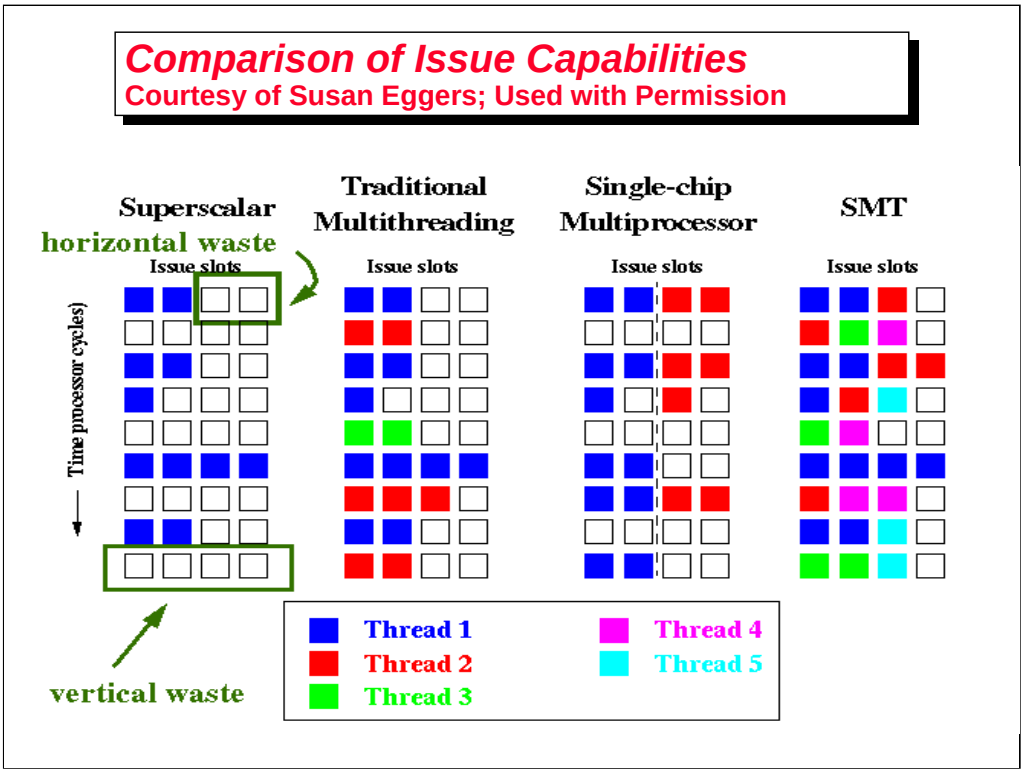
- **Interleave multiple threads to multiple issue slots with no restrictions**

21

Simultaneous Multithreading

- **Add multiple contexts and fetch engines to wide out-of-order superscalar processor**
  - **[Tullsen, Eggers, Levy, UW, 1995]**
- **OOO instruction window already has most of the circuitry required to schedule from multiple threads**
- **Any single thread can utilize whole machine**

22



Illustrates SMT thread issue & execution & how differs

SS: only single thread; long latency instructions w. lots of instructions dependent

FGMT: limited by amount of ILP in each thread, just as on the SS

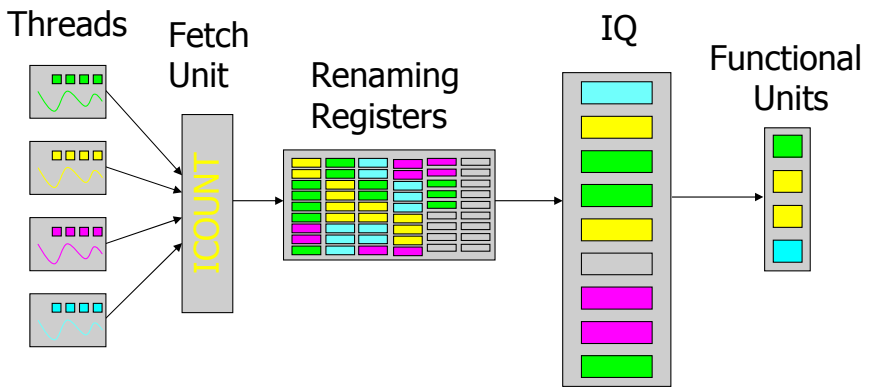
MP: each processor issues instructions from its own thread

Example of one thread stalls or has little ILP

Performance

From Superscalar to SMT

- **SMT is an out-of-order superscalar extended with hardware to support multiple executing threads**



24

no special HW for scheduling instructions from different threads onto FUs

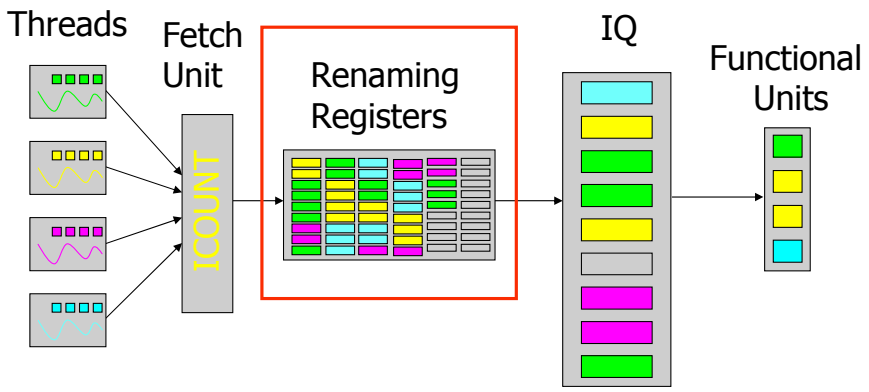
can use same ooo mechanism as superscalar for instruction issue:

RENAMING HW eliminates false dependences both within a thread (just like a conventional SS) & between threads

> **MAP thread-specific architectural registers in all threads onto a pool of physical registers**
>
> **instructions are issued when operands available without regard to thread**
> **(scheduler not look at thread IDs)**
>
> **thereafter called by their physical name**

From Superscalar to SMT

- **Extra pipeline stages for accessing thread-shared register files**

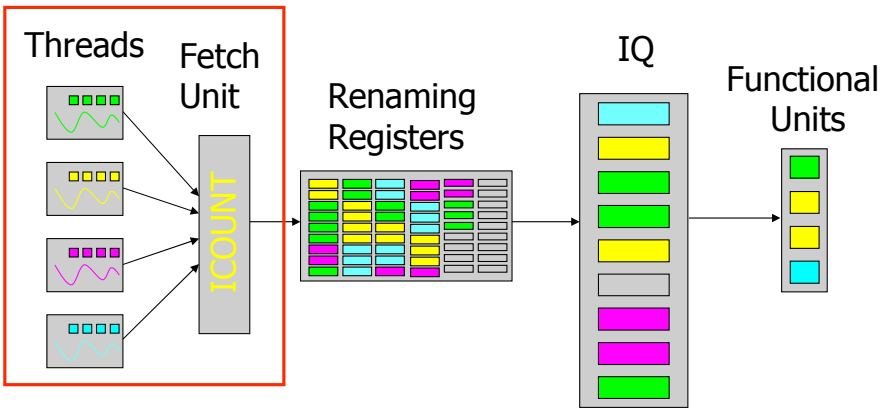


25

8*32 for the architecture state + 96 additional registers for register renaming

From Superscalar to SMT

- **Fetch from the two highest throughput threads.** ***Why?***



26

Fetch unit that can keep up with the simultaneous multithreaded execution engine

have the fewest instructions waiting to be executed

making the best progress through the machine

40% increase in IPC over RR

From Superscalar to SMT

- **Small items**
  - per-thread program counters
  - per-thread return stacks
  - per-thread bookkeeping for instruction retirement, trap & instruction dispatch queue flush
  - thread identifiers, e.g., with BTB & TLB entries

27

none of small stuff endangers critical path

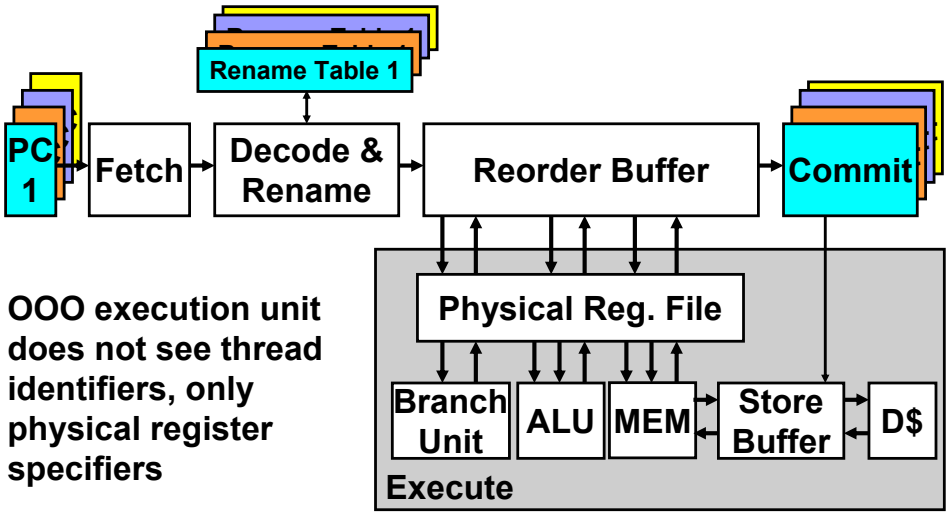
most mechanisms already exist; now duplicated for each thread or implemented to apply only to 1 thread at a time

carry thread ID for retirement, trap, queue flush, not used for scheduling

HW structure that points to all Is for each thread
need flush mechanism for branch misprediction

Fairly straightforward extension to OOO SS; this + n-fold performance boost was responsible for the technology transfer to chip manufacturers

Simultaneous Multithreaded Processor



OOO execution unit does not see thread identifiers, only physical register specifiers

28

SMT Design Issues

- **Which thread to fetch from next?**
  - Don't want to clog instruction window with thread with many stalls ➔ try to fetch from thread that has fewest insts in window
- **Locks**
  - Virtual CPU spinning on lock executes many instructions but gets nowhere ➔ add ISA support to lower priority of thread spinning on lock

29

Intel Pentium-4 Xeon Processor

- **Hyperthreading == SMT**
- **Dual physical processors, each 2-way SMT**
- **Logical processors share nearly all resources of the physical processor**
  - **Caches, execution units, branch predictors**
- **Die area overhead of hyperthreading ~ 5%**
- **When one logical processor is stalled, the other can make progress**
  - **No logical processor can use all entries in queues when two threads are active**
- **A processor running only one active software thread to run at the same speed with or without hyperthreading**

30

Load-store buffer in L1 cache doesn't behave like that, and hence 15% slowdown.