Multithreading Architectures

Joel Emer
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

Based on the material prepared by
Krste Asanovic and Arvind

Pipeline Hazards

	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14
LW r1, 0(r2)	F	D	X	M	W										
LW r5, 12(r1)		F	D	D	D	D	X	M	W						
ADDI r5, r5, #12			F	F	F	F	D	D	D	D	X	M	W		
SW 12(r1), r5							F	F	F	F	D	D	D	D	

- Each instruction may depend on the next

What can be done to cope with this?

- Even bypassing does not eliminate all delays

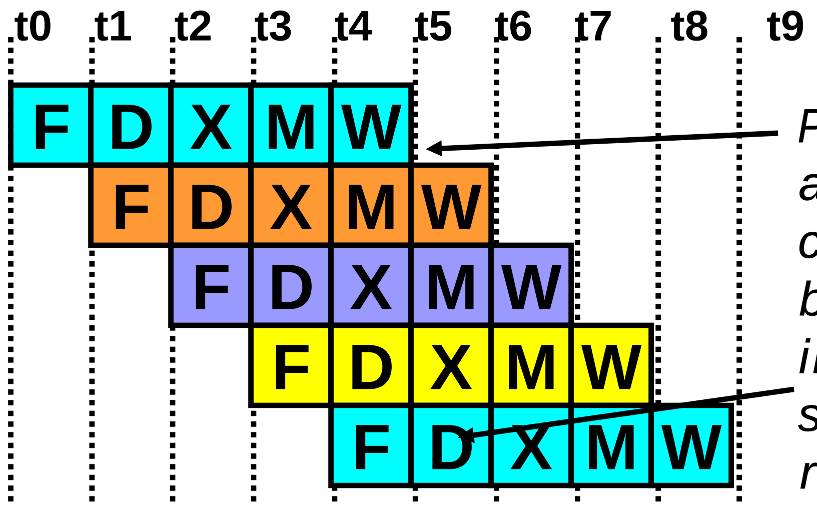
Multithreading

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)

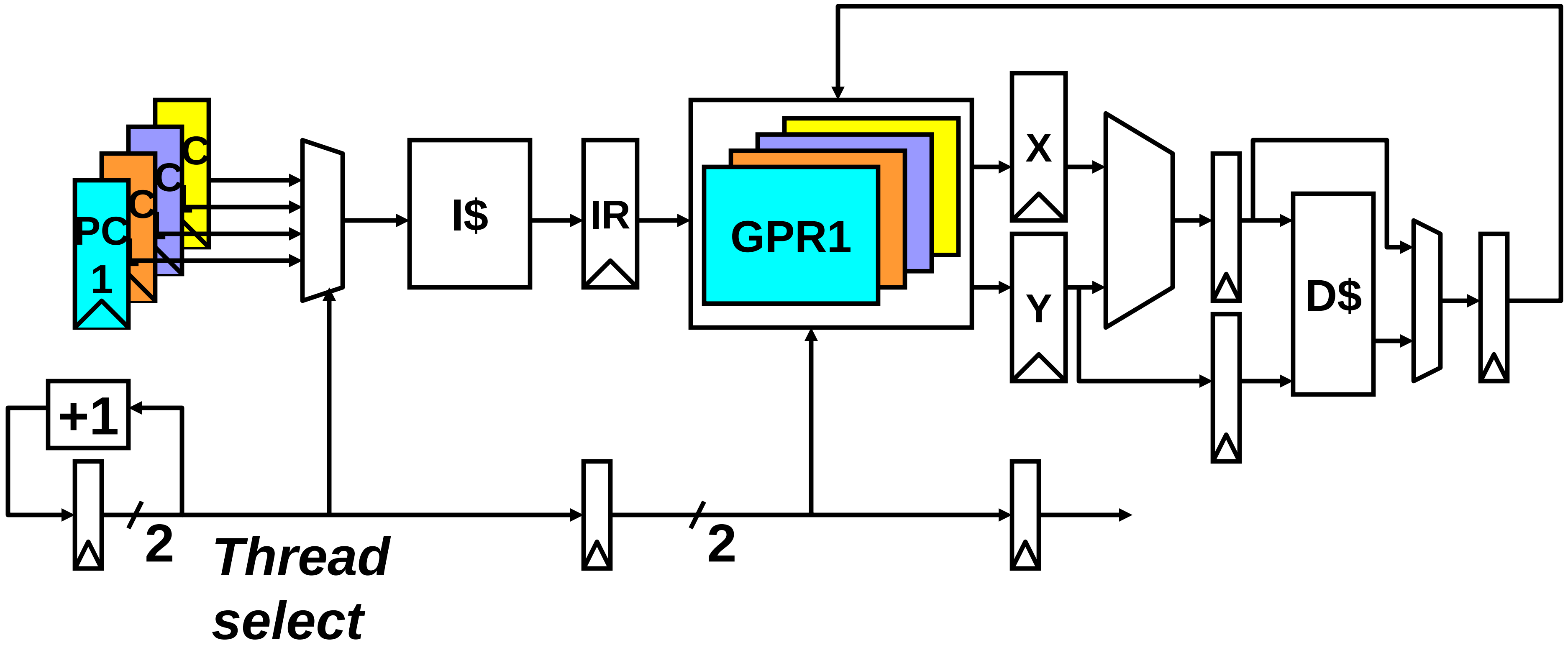


Prior instruction in a thread always completes write-back before next instruction in same thread reads register file

CDC 6600 Peripheral Processors
(Cray, 1964)

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

Simple Multithreaded Pipeline



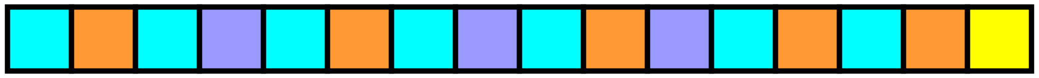
Have to carry thread select down pipeline to ensure correct state bits read/written at each pipe stage

Multithreading Costs

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

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

Denelcor HEP
(Burton Smith, 1982)

Image removed due to copyright restrictions.

To view image, visit http://ftp.arl.mil/ftp/historic-computers/png/hep2.png

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)

Tera MTA (1990-97)

Image removed due to copyright restrictions.

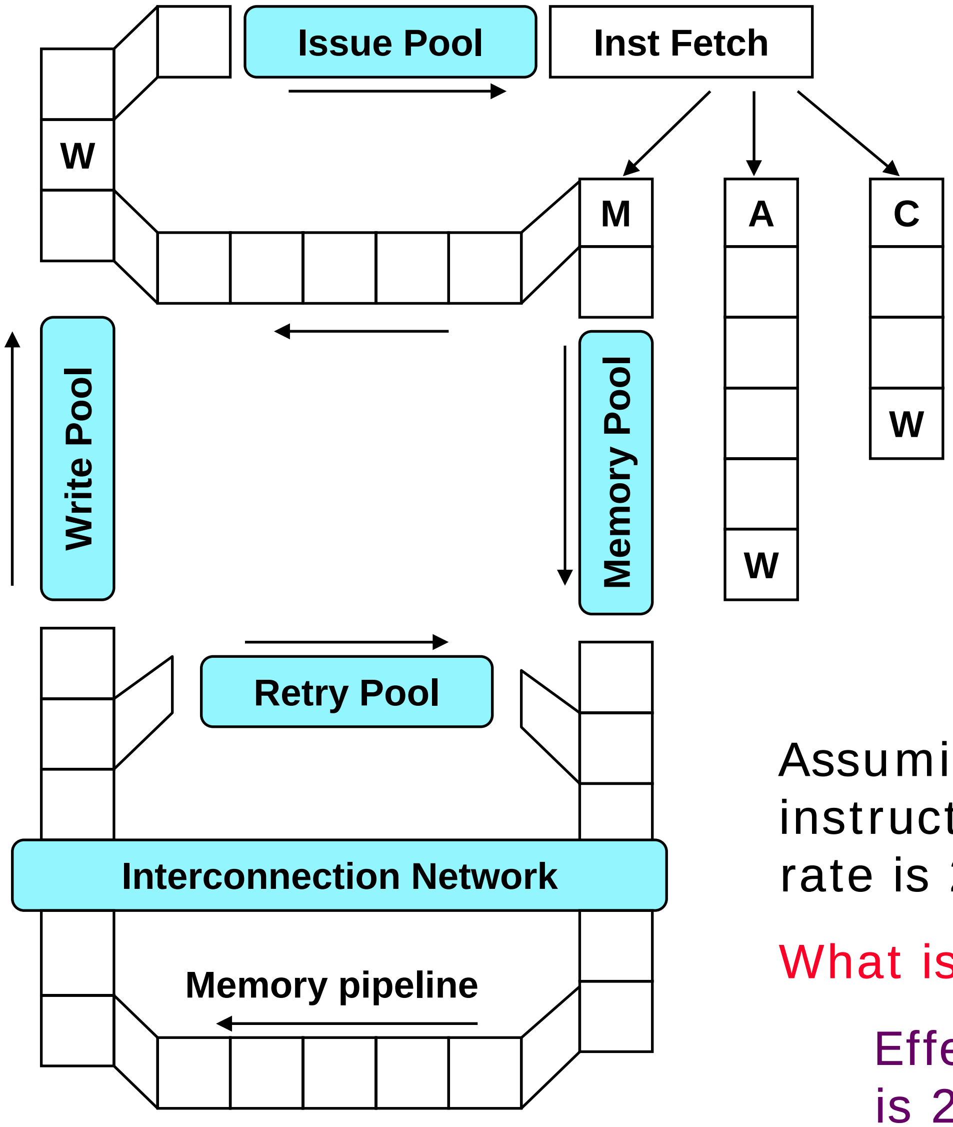
To view image, visit
http://www.npaci.edu/online/v2.1/mta.html

- 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
- GaAs logic in prototype, 1KW/processor @ 260MHz
  - CMOS version, MTA-2, 50W/processor

MTA Architecture

- Each processor supports 128 active hardware threads
  - 1 x 128 = 128 stream status word (SSW) registers,
  - 8 x 128 = 1024 branch-target registers,
  - 32 x 128 = 4096 general-purpose registers
- Three operations packed into 64-bit instruction (short VLIW)
  - One memory operation,
  - One arithmetic operation, plus
  - One arithmetic or branch operation
- Thread creation and termination instructions
- 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

MTA Pipeline



- Every cycle, one instruction from one active thread is launched into pipeline
- Instruction pipeline is 21 cycles long
- Memory operations incur ~150 cycles of latency

Assuming a single thread issues one instruction every 21 cycles, and clock rate is 260 MHz...

What is performance?

Effective single thread issue rate is 260/21 = 12.4 MIPS

Multithreading Design Choices

- Fine-grained multithreading
  - Context switch among threads every cycle
- Coarse-grained multithreading
  - Context switch among threads every few cycles, e.g., on:
    - » Function unit data hazard,
    - » L1 miss,
    - » L2 miss…
- Why choose one style over another?
- Choice depends on
  - Context-switch overhead
  - Number of threads supported (due to per-thread state)
  - Expected application-level parallelism…

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

MIT Alewife (1990)

Image removed due to copyright restrictions.

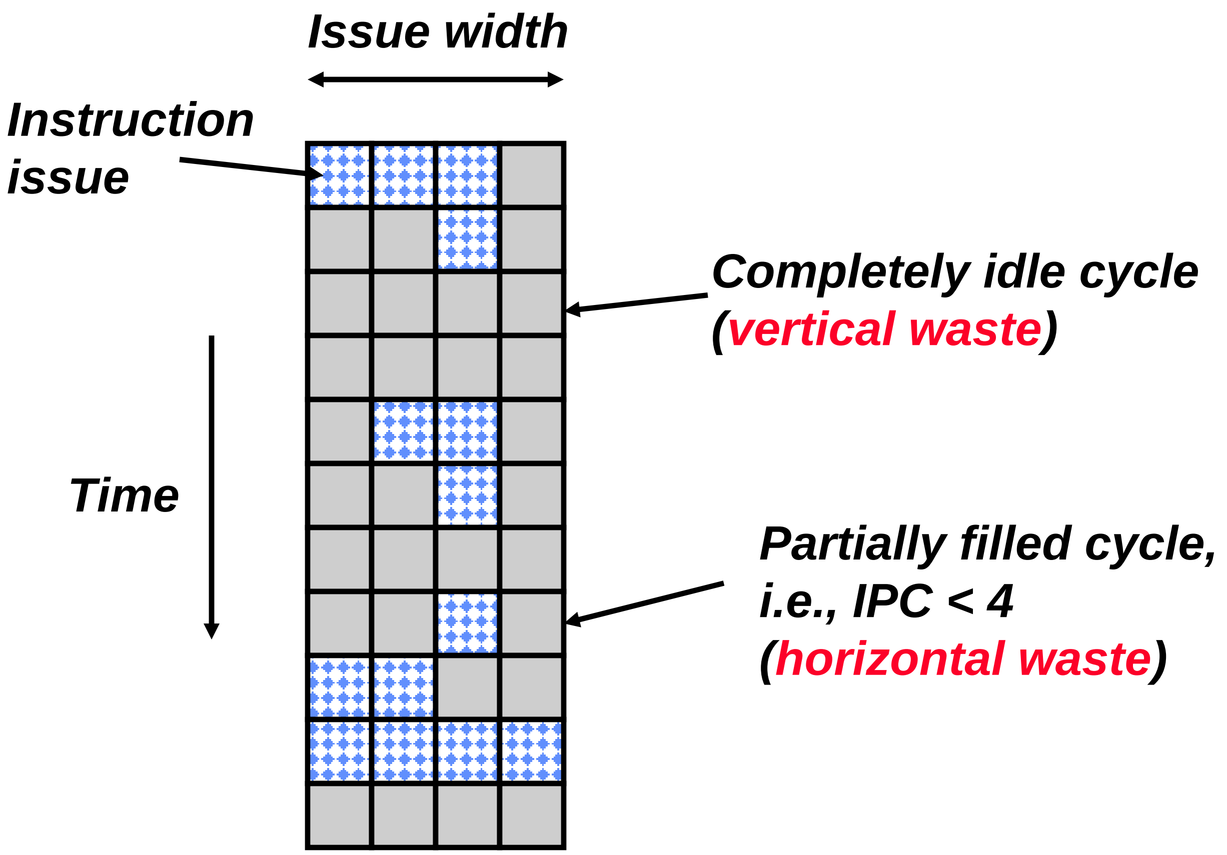
To view image, visit http://www.cag.lcs.mit.edu/alewife/pictures/jpg/16-extender.jpg

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

IBM Power RS64-IV (2000)

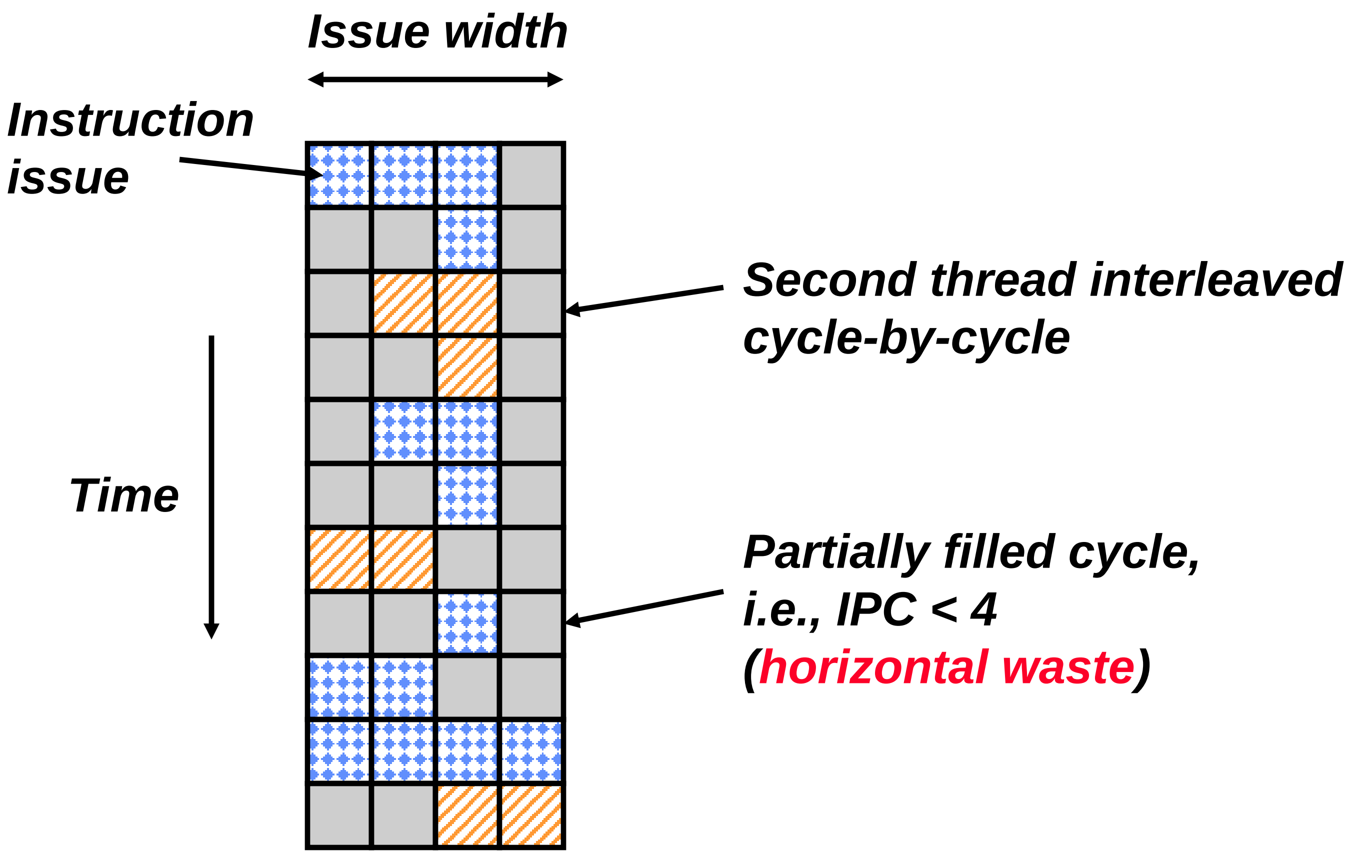
- Commercial coarse-grain multithreading CPU
- Based on PowerPC with quad-issue in-order five-stage pipeline
- Each physical CPU supports two virtual CPUs
- On L2 cache miss, pipeline is flushed and execution switches to second thread
  - short pipeline minimizes flush penalty (4 cycles), small compared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

Superscalar Machine Efficiency


Issue width
Instruction issue
Time
Completely idle cycle (vertical waste)
Partially filled cycle, i.e., IPC < 4 (horizontal waste)

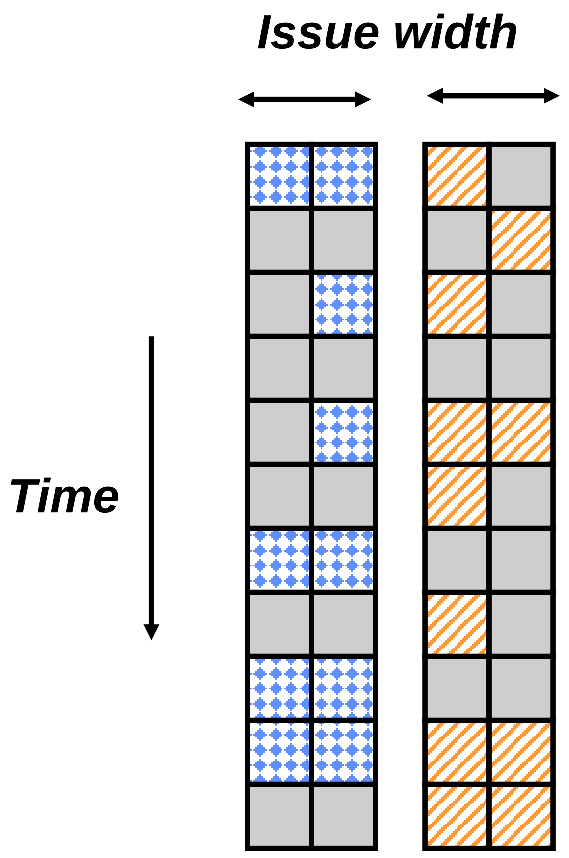
- *Why horizontal waste?*
- *Why vertical waste?*

Vertical Multithreading


Issue width
Instruction issue
Time
Second thread interleaved cycle-by-cycle
Partially filled cycle, i.e., IPC < 4 (horizontal waste)

- What is the effect of cycle-by-cycle interleaving?
  - removes vertical waste, but leaves some horizontal waste

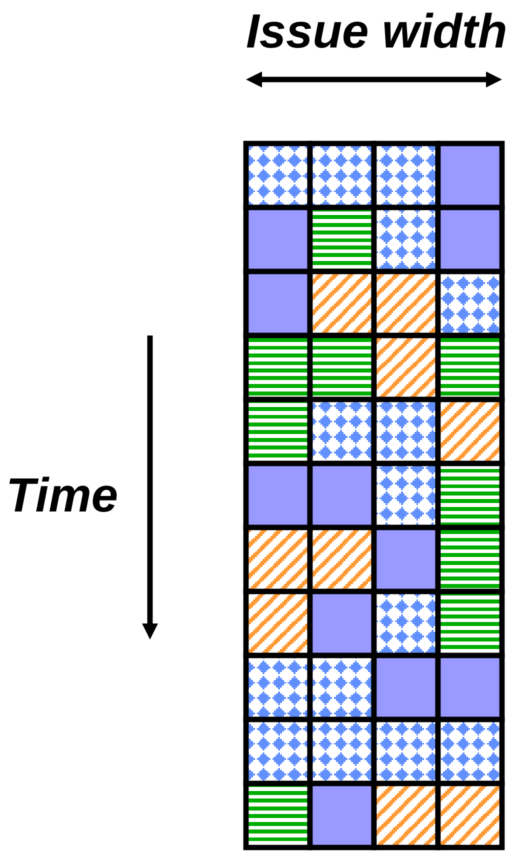
Chip Multiprocessing



- What is the effect of splitting into multiple processors?
  - eliminates horizontal waste,
  - leaves some vertical waste, and
  - caps peak throughput of each thread.

Ideal Superscalar Multithreading

[Tullsen, Eggers, Levy, UW, 1995]

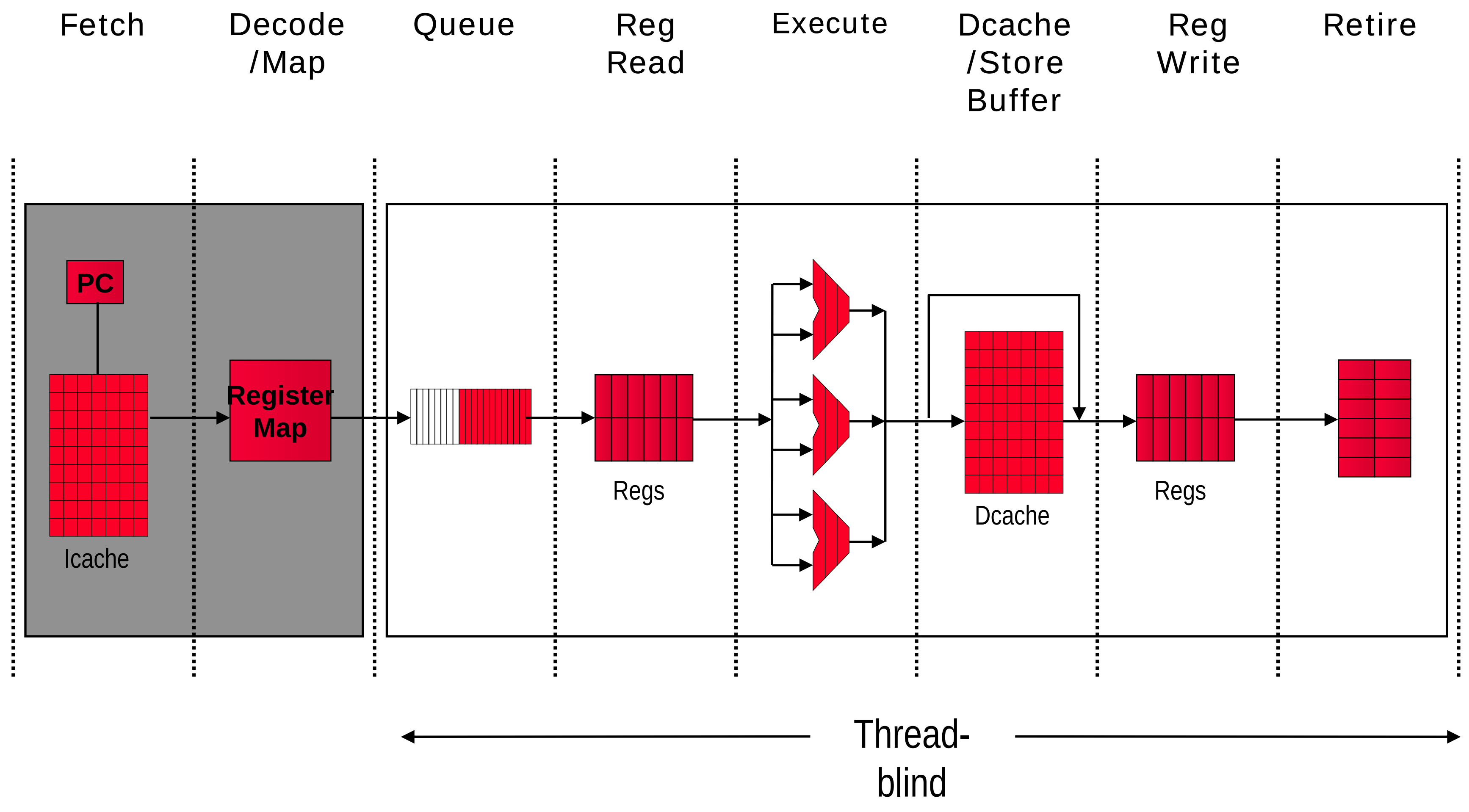


- Interleave multiple threads to multiple issue slots with no restrictions

O-o-O Simultaneous Multithreading
[Tullsen, Eggers, Emer, Levy, Stamm, Lo, DEC/UW, 1996]

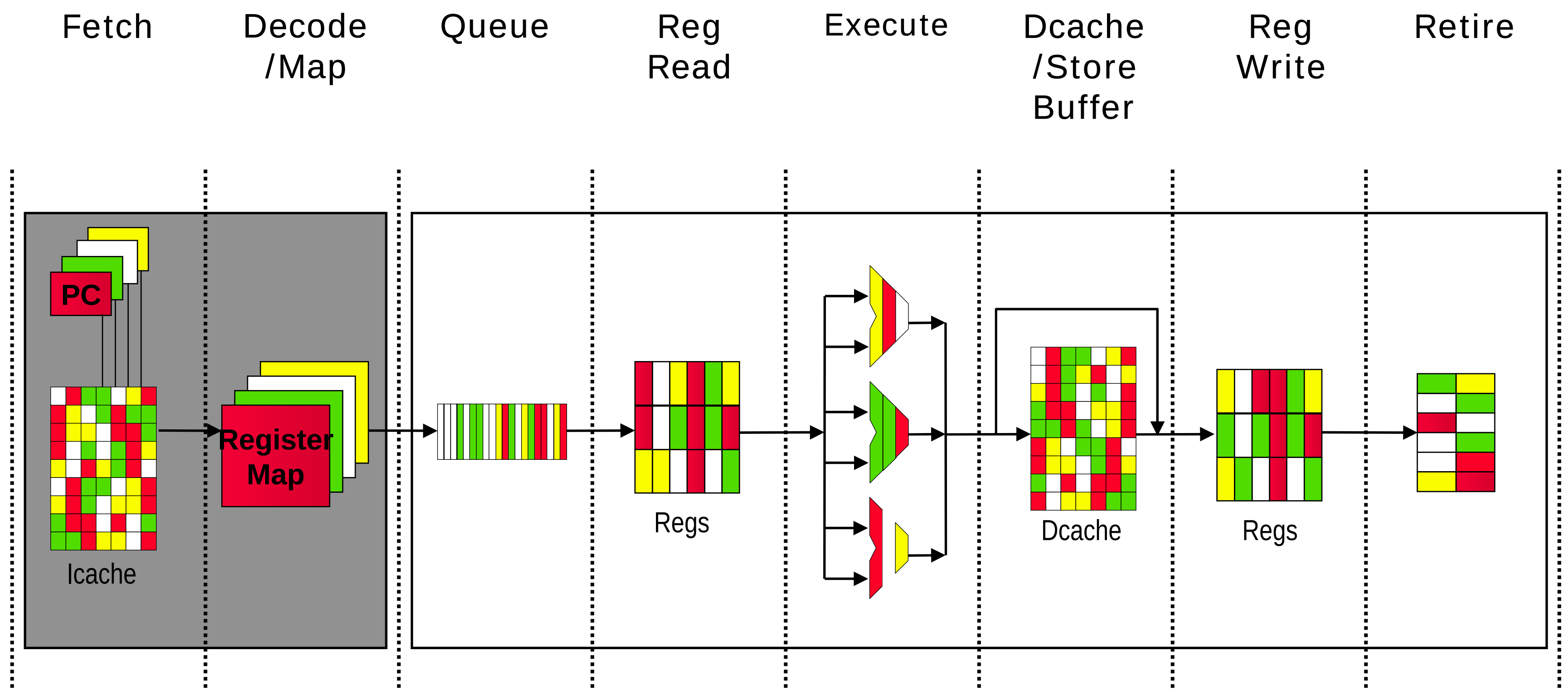
- Add multiple contexts and fetch engines and allow instructions fetched from different threads to issue simultaneously
- Utilize wide out-of-order superscalar processor issue queue to find instructions to issue from multiple threads
- OOO instruction window already has most of the circuitry required to schedule from multiple threads
- Any single thread can utilize whole machine

Basic Out-of-order Pipeline



[EV8 – Microprocessor Forum, Oct 1999]

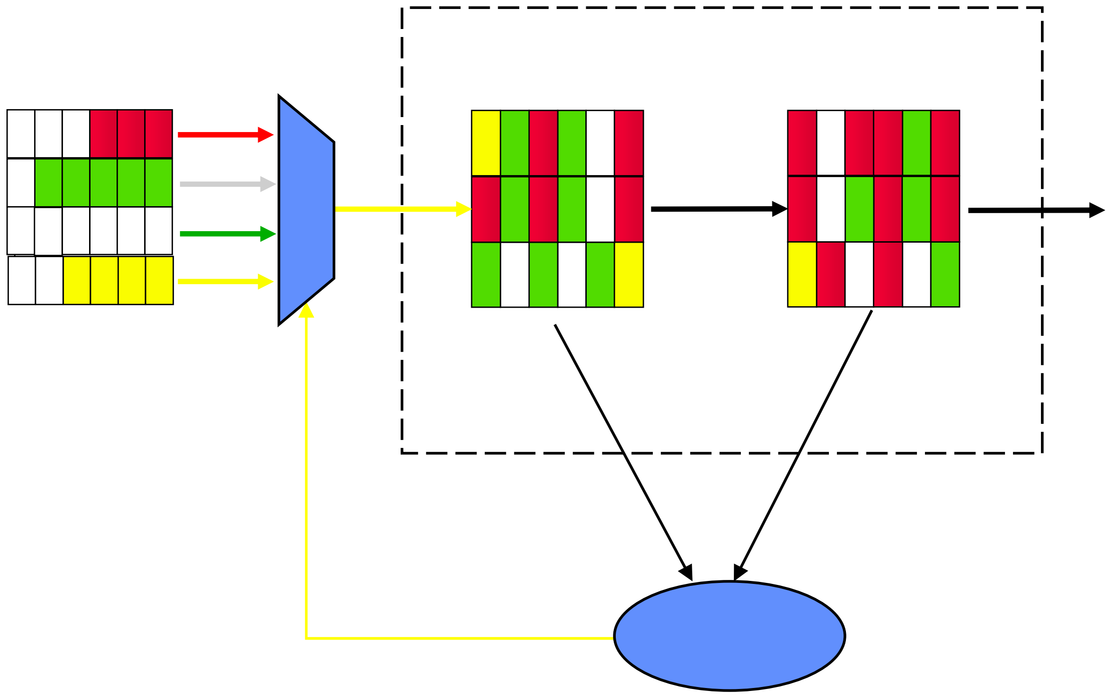
SMT Pipeline



[EV8 – Microprocessor Forum, Oct 1999]

Icount Choosing Policy

Fetch from thread with the least instructions in flight.



Why does this enhance throughput?

Why Does Icount Make Sense?

$$T = \frac{N}{L}$$

Assuming latency (L) is unchanged with the addition of threading.
For each thread i with original throughput T_i:

$$T_i/4 = \frac{N/4}{L}$$

SMT Fetch Policies (Locks)

- Problem:
  Spin looping thread consumes resources

- Solution:
  Provide quiescing operation that allows a thread to sleep until a memory location changes

```
loop:
      ARM r1, 0(r2)
      BEQ r1, got_it
      QUIESCE
      BR  loop
got_it:
```

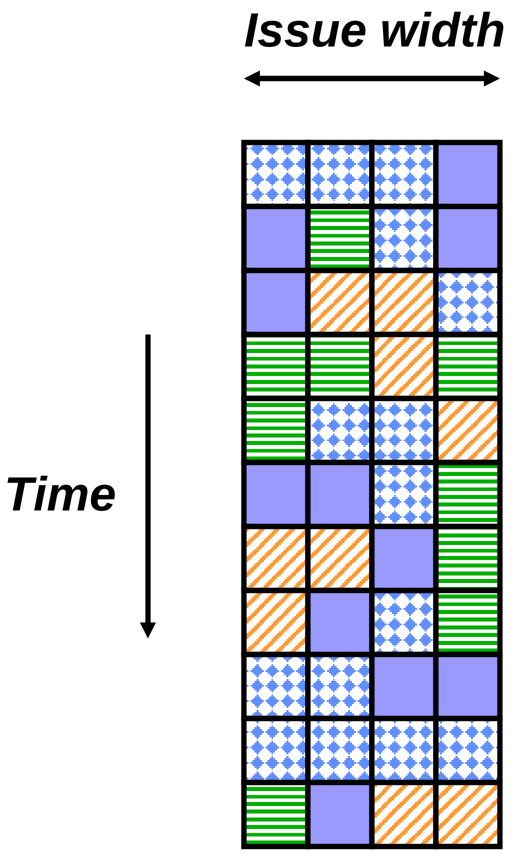
Load and start watching 0(r2)

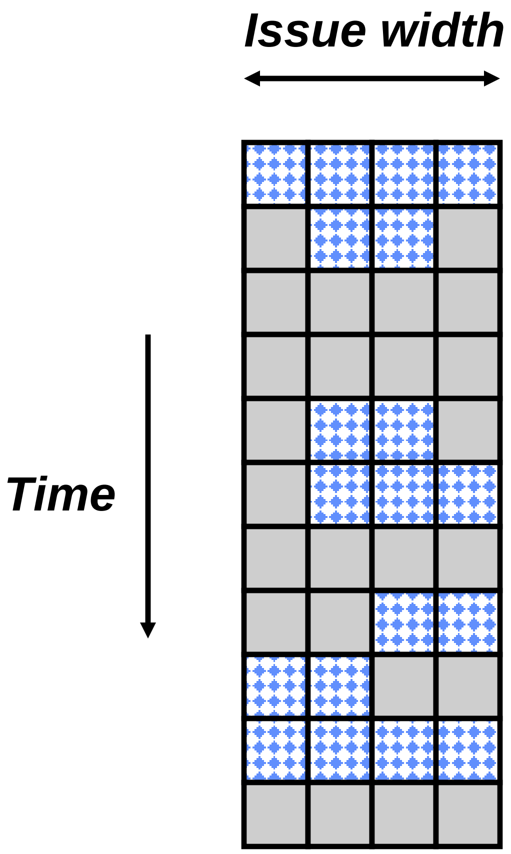
Inhibit scheduling of thread until activity observed on 0(r2)

Adaptation to parallelism type

For regions with high thread level parallelism (TLP) entire machine width is shared by all threads

For regions with low thread level parallelism (TLP) entire machine width is available for instruction level parallelism (ILP)





Pentium-4 Hyperthreading (2002)

- First commercial SMT design (2-way SMT)
  - Hyperthreading == 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
- Processor running only one active software thread runs at approximately same speed with or without hyperthreading

Pentium-4 Hyperthreading
Front End

Figure removed due to copyright restrictions.

Refer to Figure 5a in Marr, D., et al. "Hyper-threading Technology Architecture and Microarchitecture." *Intel Technology Journal* 6, no. 1 (2002): 8.
http://www.intel.com/technology/itj/2002/volume06issue01/vol6iss1_hyper_threading_technology.pdf

Pentium-4 Branch Predictor

- Separate return address stacks per thread

  Why?

- Separate first-level global branch history table

  Why?

- Shared second-level branch history table, tagged with logical processor IDs

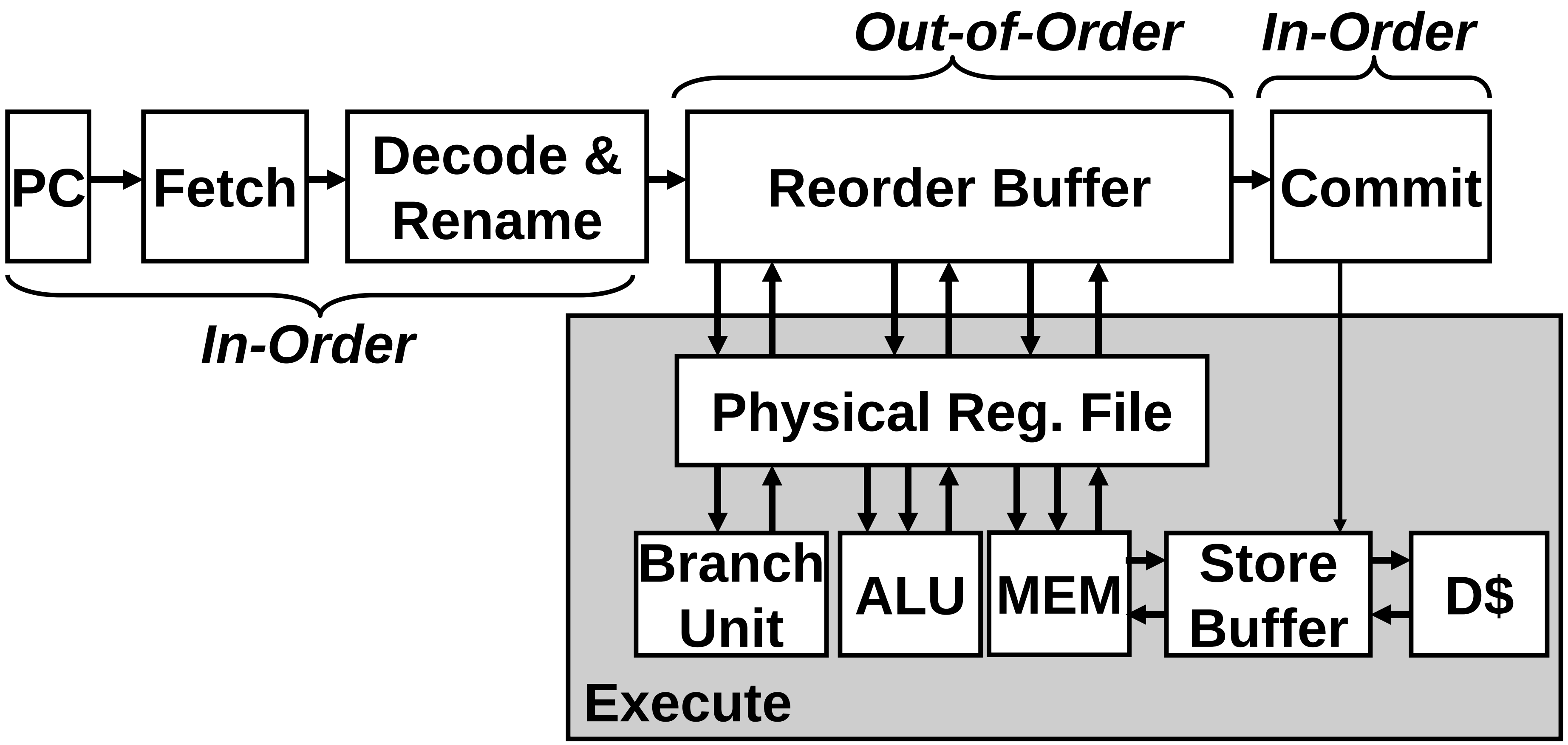
Pentium-4 Hyperthreading
Execution Pipeline

Figure removed due to copyright restrictions.

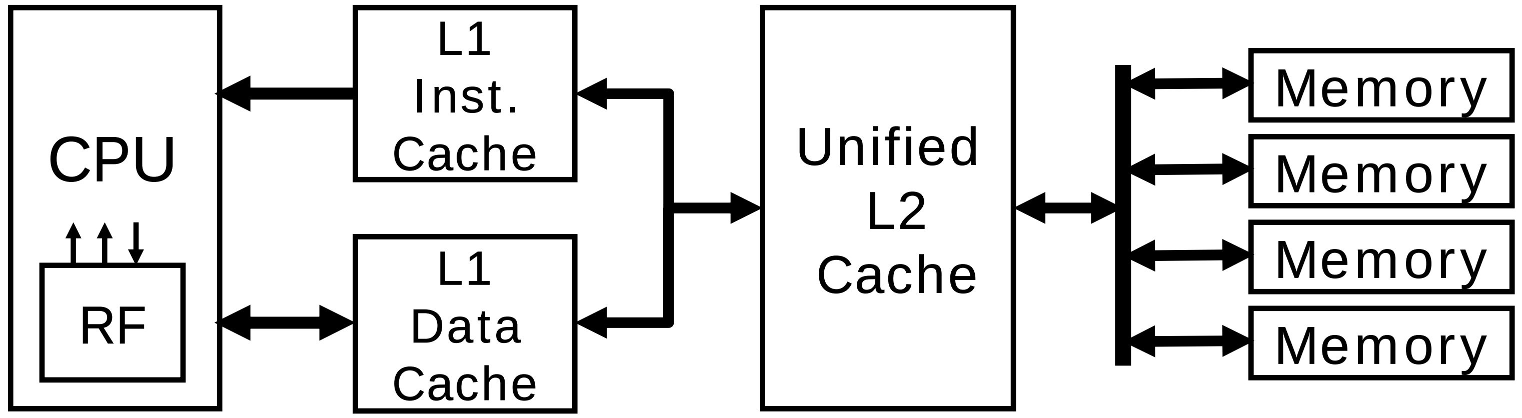
Refer to Figure 6 in Marr, D., et al. "Hyper-threading Technology Architecture and Microarchitecture." *Intel Technology Journal* 6, no. 1 (2002): 10.
http://www.intel.com/technology/itj/2002/volume06issue01/vol6iss1_hyper_threading_technology.pdf

Extras

Speculative, Out-of-Order Superscalar Processor



Granularity of 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?*