ECE 552 / CPS 550
Advanced Computer Architecture I

Lecture 16
Multi-threading

Benjamin Lee
Electrical and Computer Engineering
Duke University

www.duke.edu/~bcl15
www.duke.edu/~bcl15/class/class_ece252fall12.html

ECE552 Administrivia

13 November – Homework #4 Due

Project Status

- Plan on having preliminary data or infrastructure

8 November – Class Discussion

Roughly one reading per class. Do not wait until the day before!

1. Mudge, “Power: A first-class architectural design constraint”
2. Lamport, “How to make a multiprocessor computer that correctly executes multiprocess programs”
3. Lenoski et al. “The Stanford DASH Multiprocessor”
4. Tullsen et al. “Simultaneous multithreading: Maximizing on-chip parallelism”

Last Time

Out-of-order Superscalar

- Hardware complexity increases super-linearly with issue width

Very Long Instruction Word (VLIW)

- Compiler explicitly schedules parallel instructions
- Simple hardware, complex compiler
- Later VLIWs added more dynamic interlocks

Compiler Analysis

- Use loop unrolling and software pipelining for loops, trace scheduling for more irregular code
- Static compiler scheduling is difficult in presence of unpredictable branches and variable memory latency

Multi-threading

Instruction-level Parallelism

- Objective: Extract instruction-level parallelism (ILP)
- Difficulty: Limited ILP from sequential thread of control

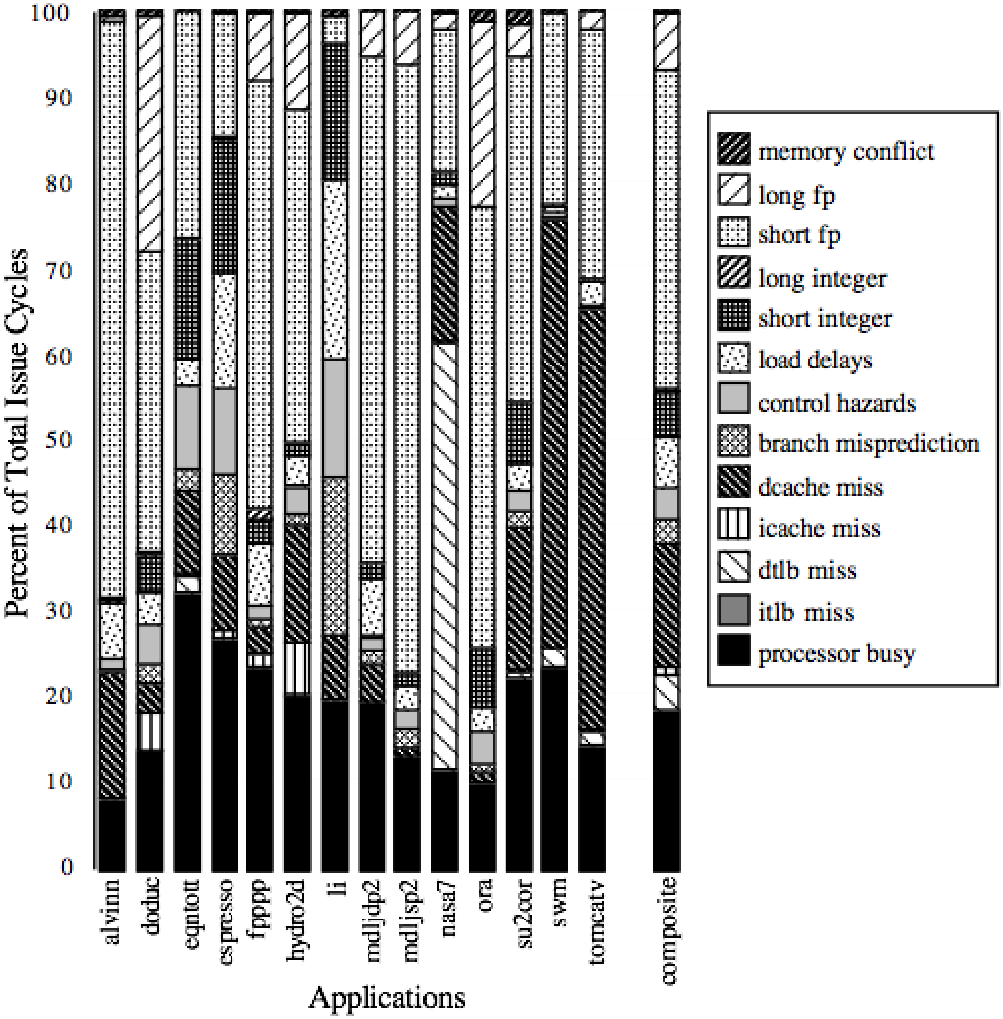
Multi-threaded Processor

- Processor fetches instructions from multiple threads of control
- If instructions from one thread stalls, issue instructions from other threads

Thread-level Parallelism

- Thread-level parallelism (TLP) provides independent threads
- Multi-programming – run multiple, independent, sequential jobs
- Multi-threading – run single job faster using multiple, parallel threads

Increasing Pipeline Utilization

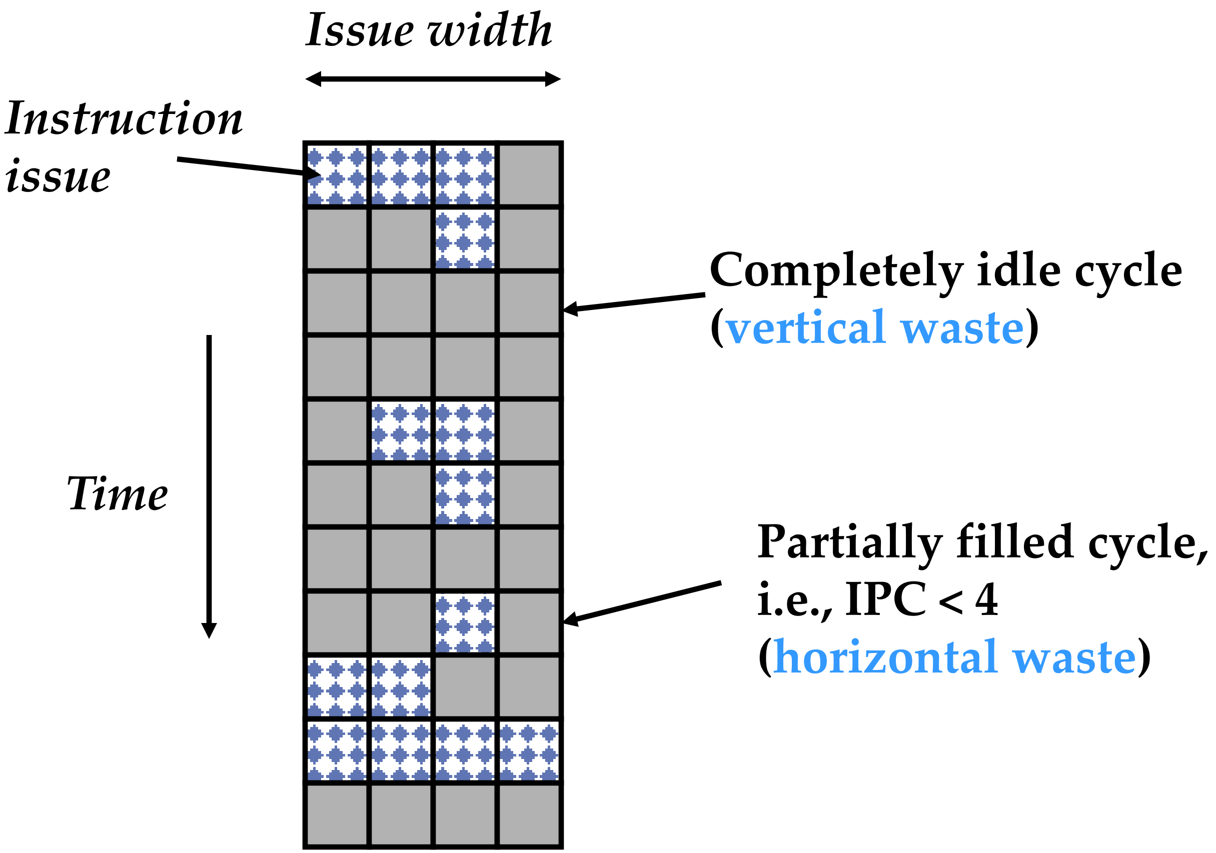


In an out-of-order superscalar processor, many apps cannot fully use execution units

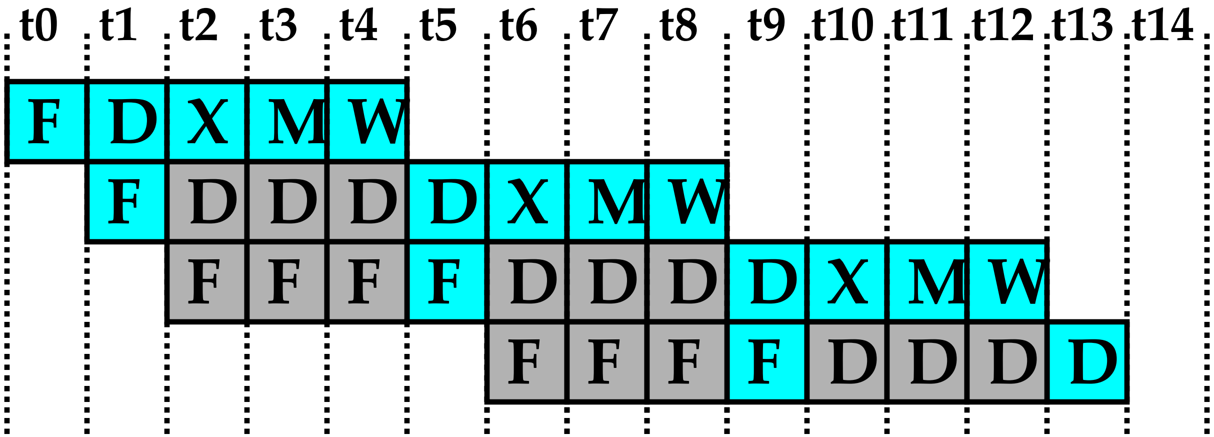
Consider percentage of issue cycles in which processor is busy.

Tullsen, Eggers and Levy. "Simultaneous multi-threading," ISCA 1995.

Superscalar (In)Efficiency


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

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	

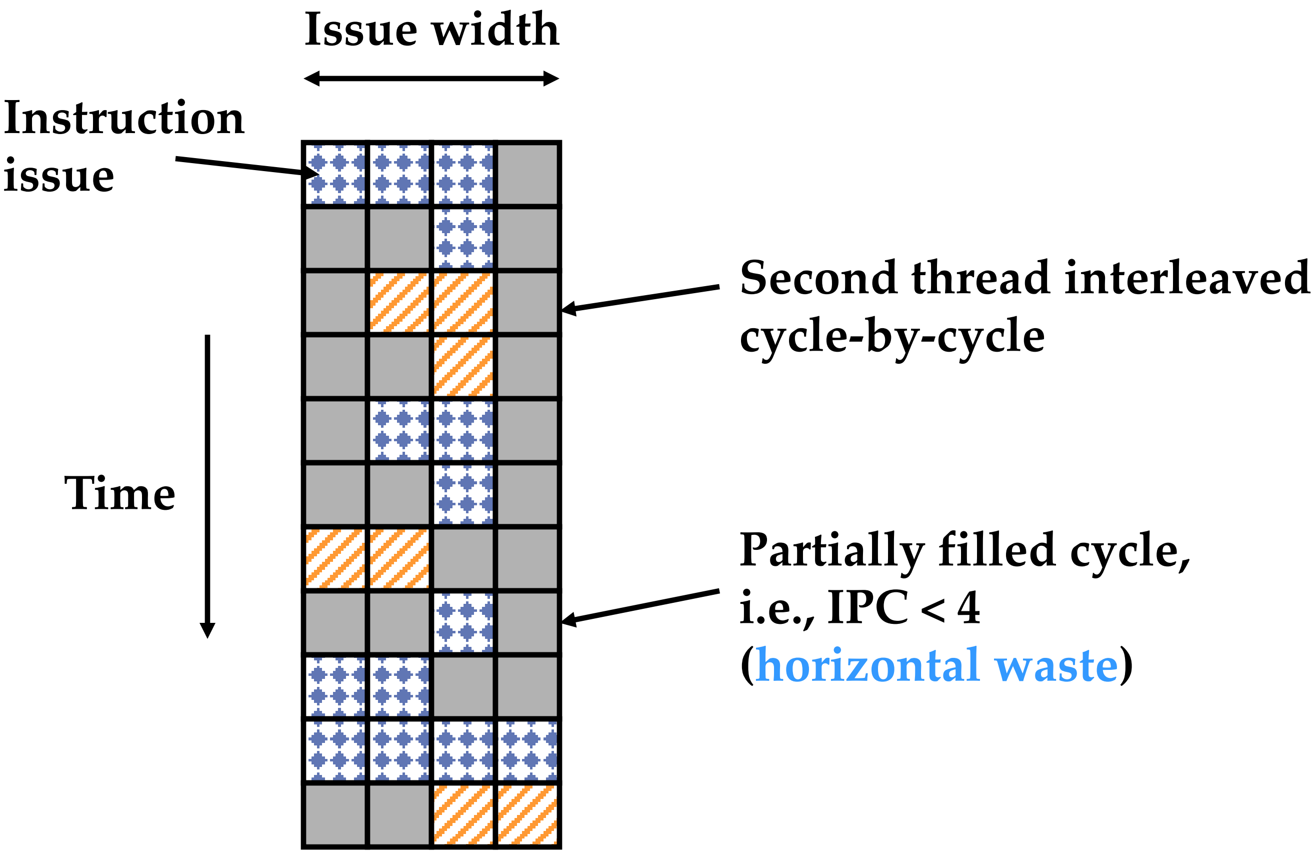
Data Dependencies

- Dependencies exist between instructions
- Example: LW-LW (via r1), LW-ADDI (via r5), ADDI-SW (via r5)

Solutions

- Interlocks stall the pipeline (slow)
- Forwarding paths (requires hardware, limited applicability)

Vertical Multithreading



Issue width

Instruction issue

Time

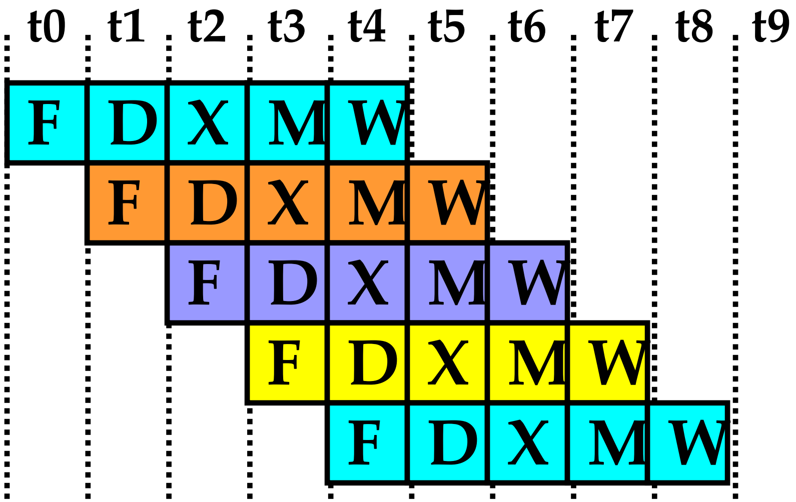
Second thread interleaved cycle-by-cycle

Partially filled cycle, i.e., IPC < 4 (horizontal waste)

With cycle-by-cycle interleaving, remove vertical waste

Multithreading

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)



t0	t1	t2	t3	t4	t5	t6	t7	t8	t9
F	D	X	M	W					
	F	D	X	M	W				
		F	D	X	M	W			
			F	D	X	M	W		
				F	D	X	M	W	

Interleave instructions from multiple threads in pipeline

- Example: Interleave threads (T1, T2, T3, T4) in 5-stage pipeline

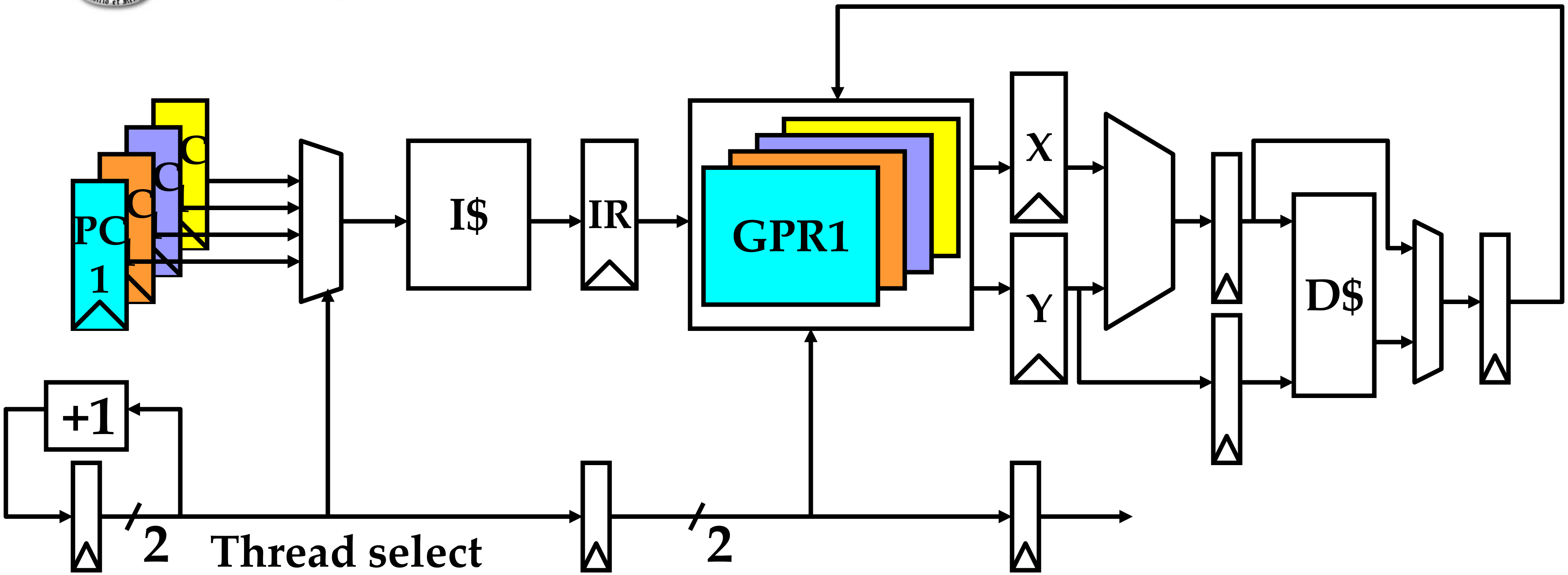
- For a given thread, earlier instruction writes-back (W) before later instruction reads register file (D).
- Example: [T1: LW r1, 0(r2)] writes back before [T1: LW r5, 12(r1)] decodes

CDC 6600 Peripheral Processors

- Cray (1964) built first multithreaded hardware architecture

- Pipeline with100ns clock period
- 10 “virtual” I/O processors → provides thread-level parallelism
- Each virtual processor executes one instruction every 1000ns (= 1ms)
- Minimize processor state with accumulator-based instruction set

Simple Multithreaded Pipeline



- Additional State: One copy of architected state per thread (e.g., PC, GPR)
- Thread Select: Round-robin logic; Propagate thread ID down pipeline to access correct state (e.g., GPR1 versus GPR2)

- Software (e.g., OS) perceives multiple, slower CPUs

Multithreading Costs

User State (per thread)

- Program counters
- Physical register files

System State (per thread)

- Page table base register
- Exception handling registers

Overheads and Contention

- Threads contend for shared caches, TLB
- Alternatively, threads require additional cache, TLB capacity
- Scheduler (OS or HW) manages threads

Fine-Grain Multithreading

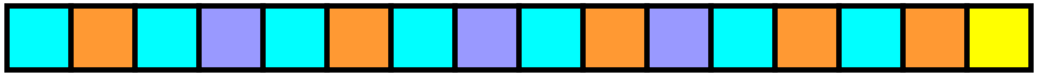
Switch threads at instruction-granularity

Fixed Interleave (CDC 6600 PPUs, 1964)

- PPU – peripheral processing unit
- Given N threads, each thread executes one instruction every N cycles
- Insert pipeline bubble (e.g., NOP) if thread not ready for its slot

Software-controlled Interleave (TI ASC PPUs, 1971)

- PPU – peripheral processing unit
- OS explicitly controls thread interleaving
- Example: blue thread scheduled 2x as often as orange, purple thread



Why was thread scheduling introduced for peripheral processing units first?

Coarse-Grain Multithreading

Switch threads on long-latency operation

Tera MTA designed for supercomputing applications with large data sets and little locality

- Little locality → no data cache
- Many parallel threads needed to hide long memory latency
- If one thread accesses memory, schedule another thread in its place

Other applications may be more cache friendly

- Good locality → data cache hits
- Provide small number of threads to hide cache miss penalty
- If one thread misses cache, schedule another thread in its place

Denelcor HEP (1982)

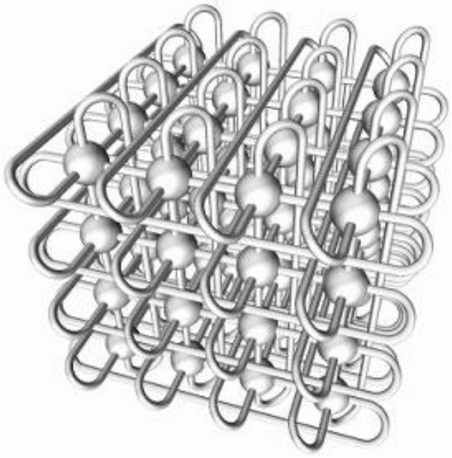
First commercial hardware-threading for main CPU

- Architected by Burton Smith
- Multithreading previously used to hide long I/O, memory latencies in PPUs

- Up to 8 processors, 10 MHz Clock
- 120 threads per processor
- Precursor to Tera MTA

Tera MTA (1990)

- Up to 256 processors
- Up to 128 threads per processor



- Processors and memories communicate via a 3D torus interconnect

  Nodes linked to nearest 6 neighbors

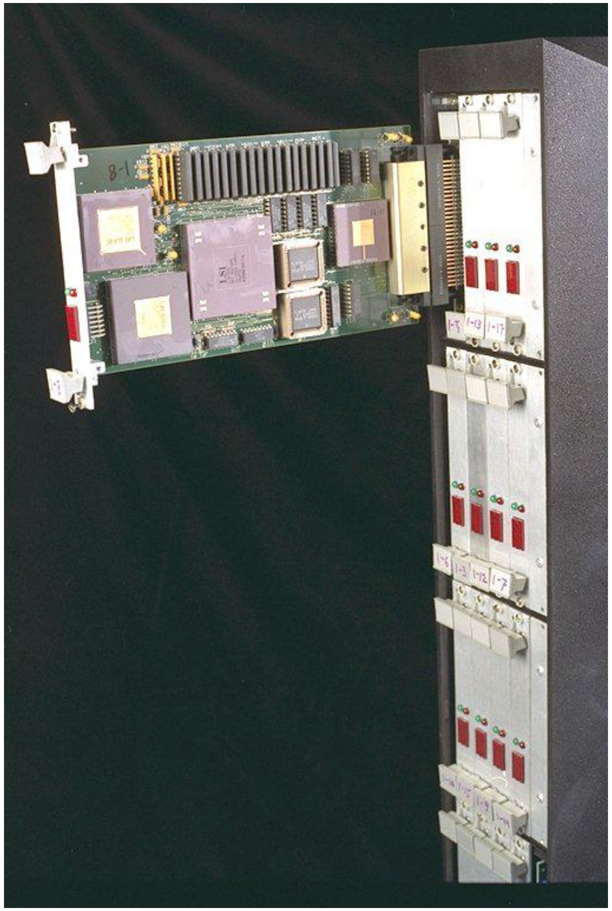
- Main memory is flat and shared

  Flat memory → no data cache

  Memory supports one memory access per cycle per processor

  Why does this make sense for a multi-threaded machine?

MIT Alewife (1990)



Anant Agarwal at MIT

SPARC

- RISC instruction set architecture from Sun Microsystems
- Alewife modifies SPARC processor

Multithreading

- Up to 4 threads per processor
- Threads switch on cache miss

Multithreading & "Simple" Cores

IBM PowerPC RS64-IV (2000)

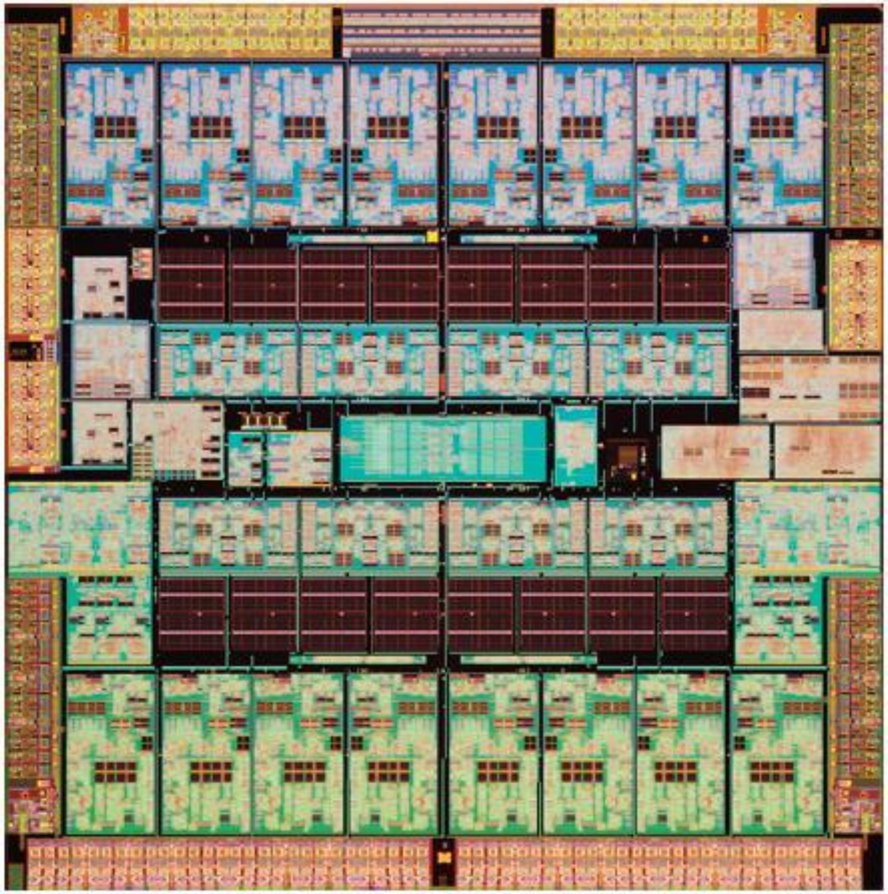
- RISC instruction set architecture from IBM
- Implements in-order, quad-issue, 5-stage pipeline
- Up to 2 threads per processor

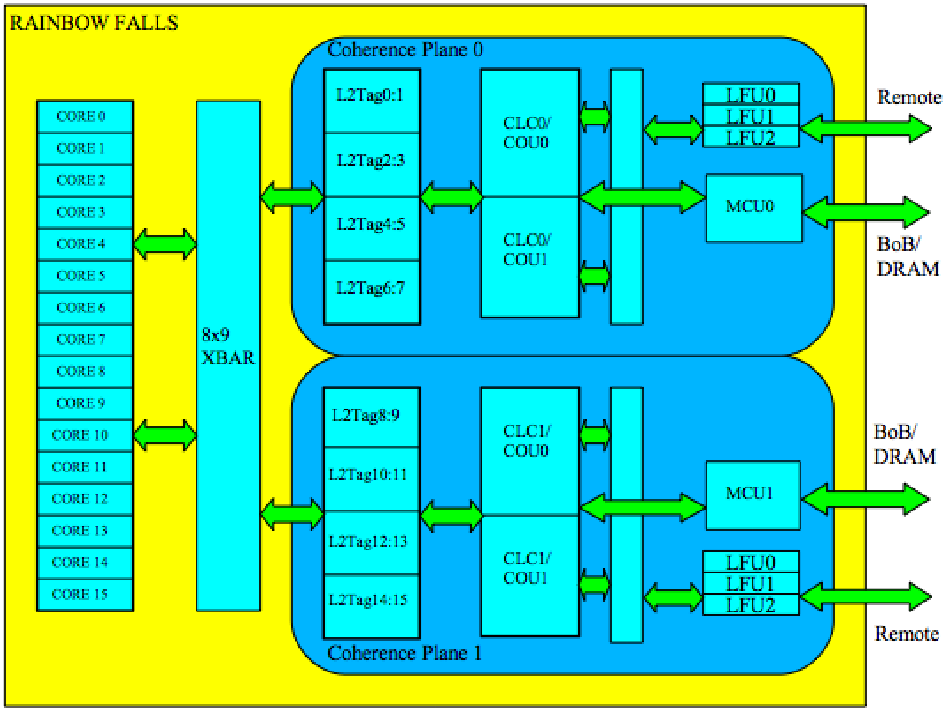
Oracle/Sun Niagara Processors (2004-2009)

- Targets datacenter web and database servers.
- SPARC instruction set architecture from Sun
- Implements simple, in-order core

- Niagara-1 (2004) – 8 cores, 4 threads/core
- Niagara-2 (2007) – 8 cores, 8 threads/core
- Niagara-3 (2009) – 16 cores, 8 threads/core

Oracle/Sun Niagara-3 (2009)





Why Simple Cores?

Switch threads on cache miss

- If a thread accesses memory, flush pipeline switch to another thread

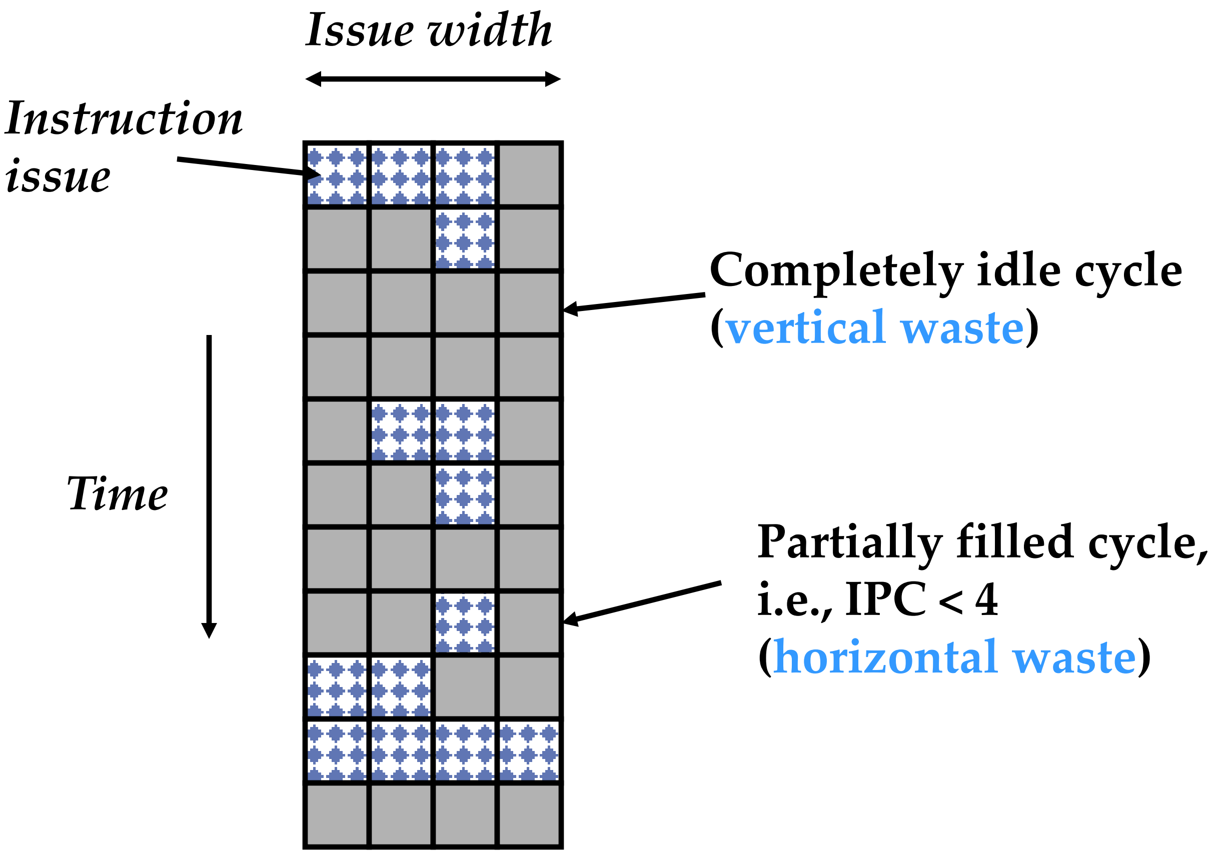
Simple Core Advantages

- Minimize flush penalty with short pipeline (4 cycles in 5-stage pipeline)
- Reduce energy cost per op (out-of-order execution consumes energy)

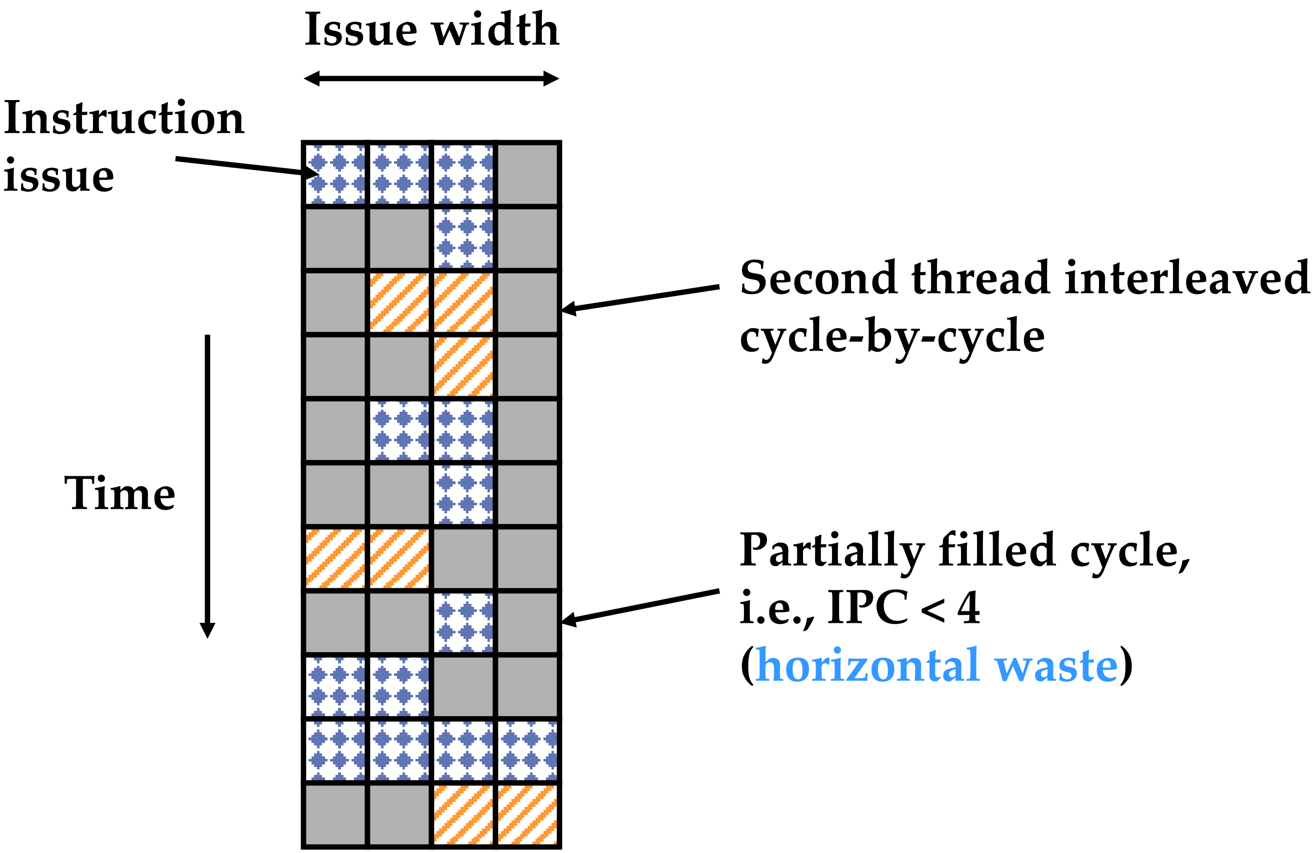
Simple Core Trade-off

- Lower single-thread performance
- Higher multi-thread performance and energy efficiency

Superscalar (In)Efficiency

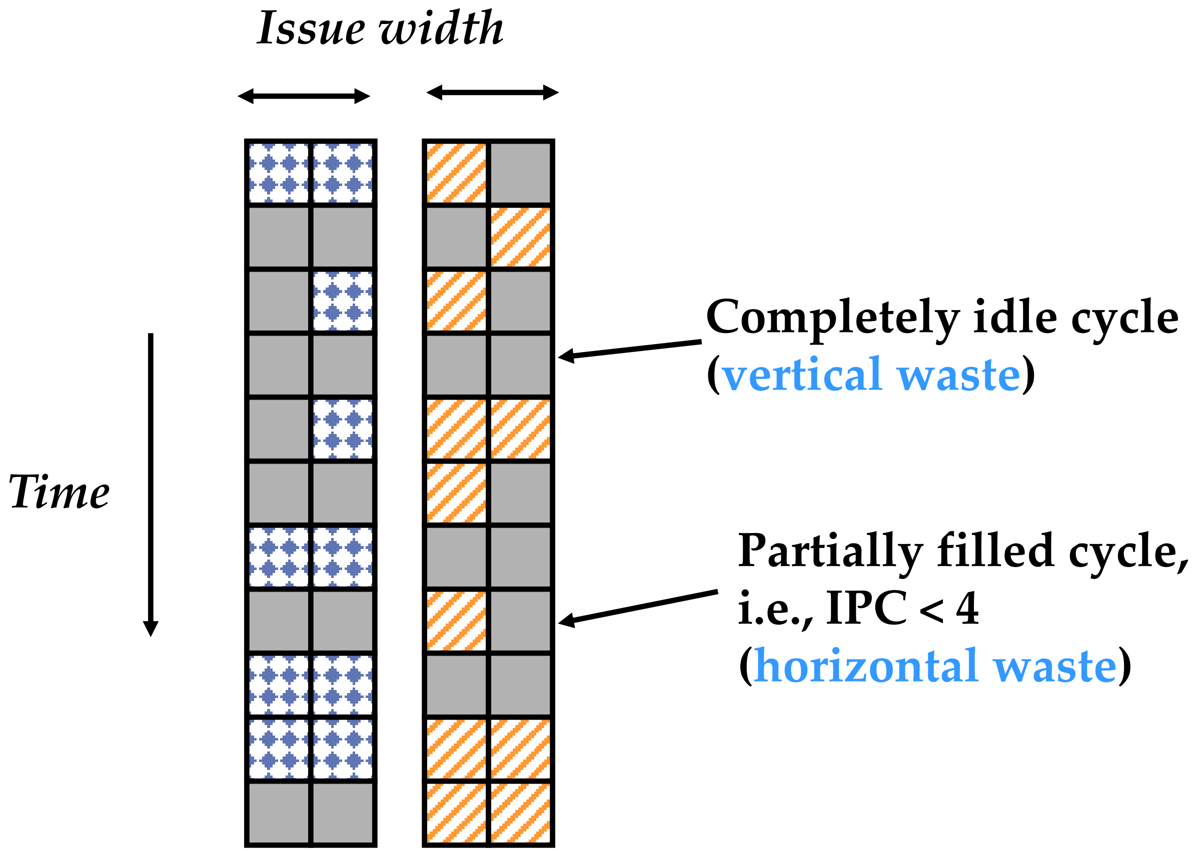

Issue width
Instruction issue
Time
Completely idle cycle (vertical waste)
Partially filled cycle, i.e., IPC < 4 (horizontal waste)

Vertical Multithreading


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

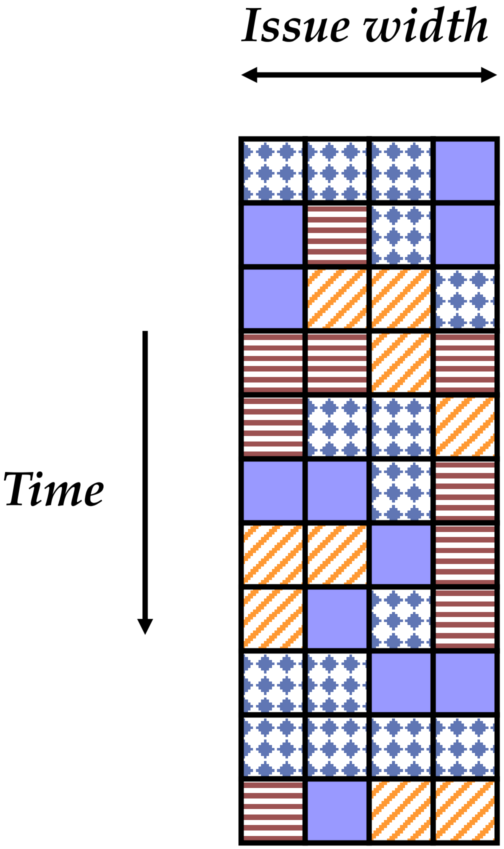
Vertical multithreading reduces vertical waste with cycle-by-cycle interleaving. However, horizontal waste remains.

Chip Multiprocessing (CMP)



Chip multiprocessing reduces horizontal waste with simple (narrower) cores. However, (1) vertical waste remains and (2) ILP is bounded.

Simultaneous Multithreading (SMT)



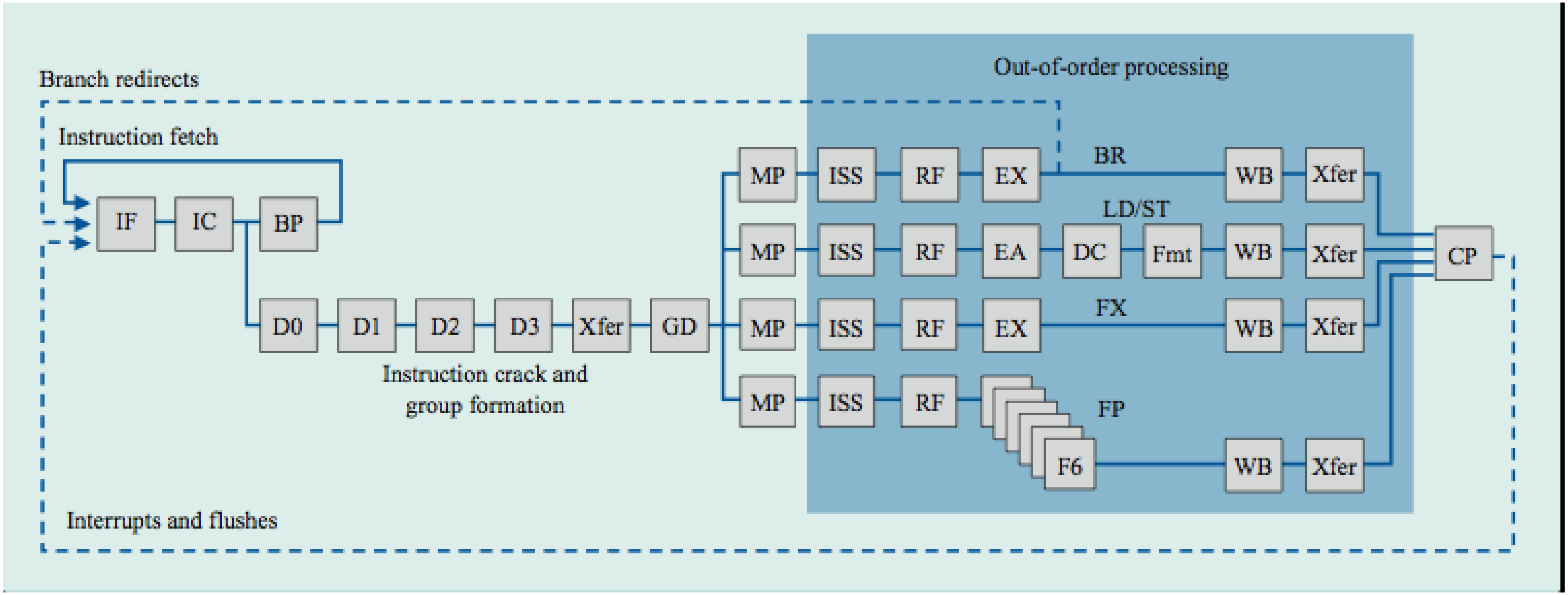
Interleave multi-threaded instructions with no restrictions.

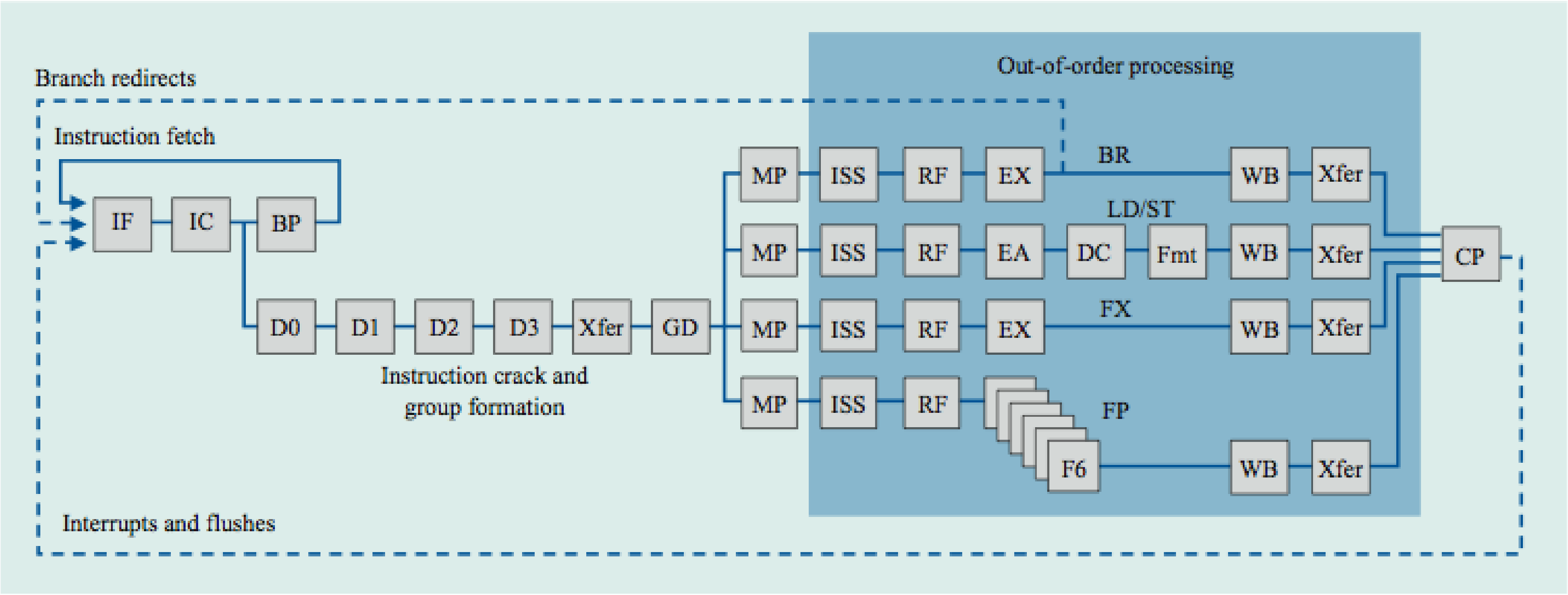
Tullsen, Eggers, Levy. University of Washington, 1995.

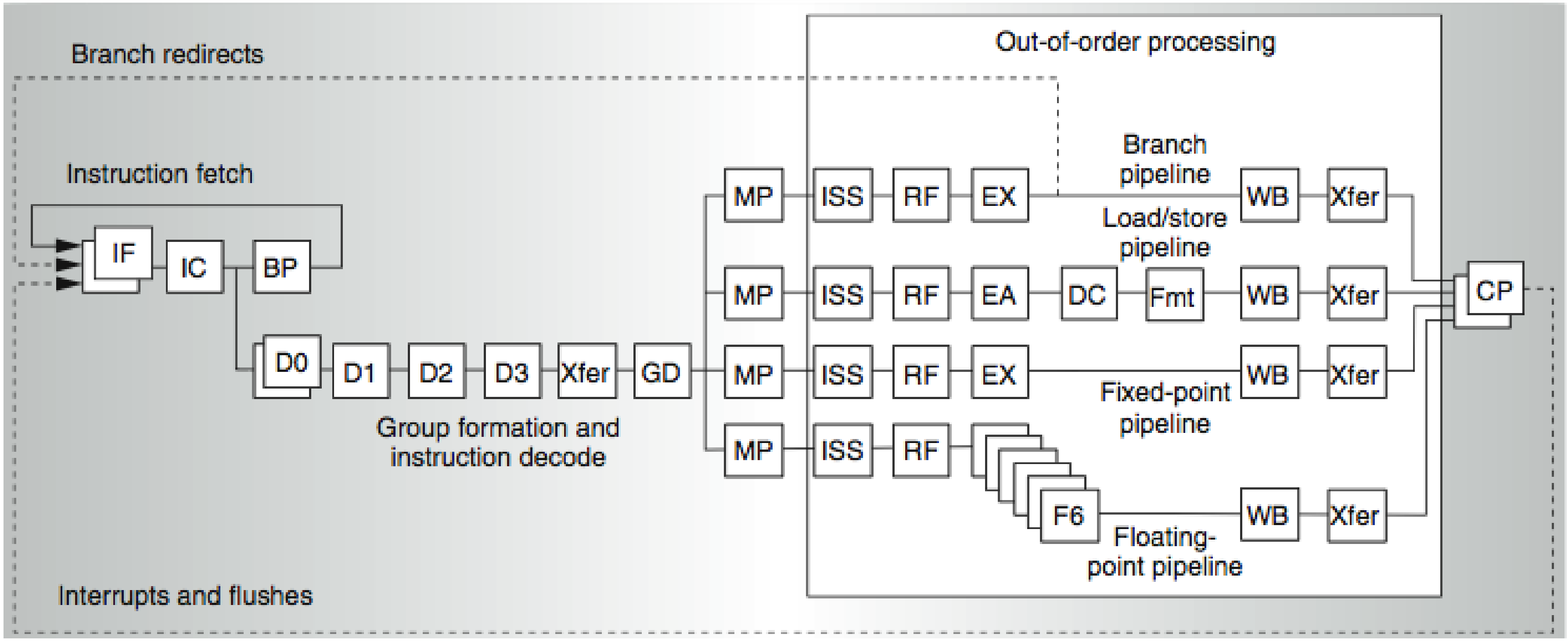
IBM Power4 → IBM Power5

Power4

- Single-threaded processor
- Out-of-order execution, superscalar (8 execution units)

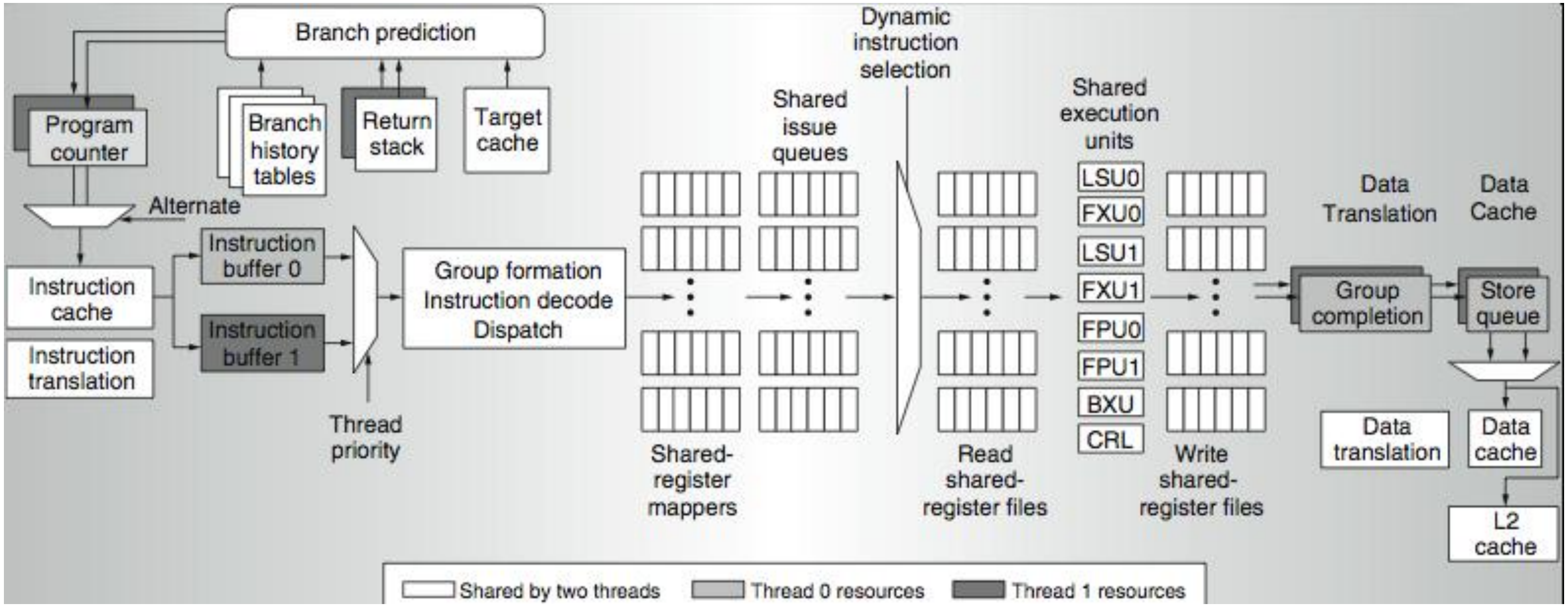



Branch redirects
Out-of-order processing
Instruction fetch
IF
IC
BP
D0
D1
D2
D3
Xfer
GD
Instruction crack and group formation
MP
ISS
RF
EX
BR
WB
Xfer
LD/ST
MP
ISS
RF
EA
DC
Fmt
WB
Xfer
CP
FX
MP
ISS
RF
EX
WB
Xfer
MP
ISS
RF
FP
F6
WB
Xfer
Interrupts and flushes


Branch redirects
Out-of-order processing
Instruction fetch
IF
IC
BP
D0
D1
D2
D3
Xfer
GD
Group formation and instruction decode
MP
ISS
RF
EX
Branch pipeline
WB
Xfer
Load/store pipeline
MP
ISS
RF
EA
DC
Fmt
WB
Xfer
CP
MP
ISS
RF
EX
WB
Xfer
Fixed-point pipeline
MP
ISS
RF
F6
WB
Xfer
Floating-point pipeline
Interrupts and flushes

IBM Power5 Data Flow

- Program Counter: duplicate and alternate fetch between two threads
- Return Stack: duplicate since different threads call different sub-routines
- Instruction Buffer: queue instructions separately
- Decode: de-queue instructions depending on thread priority



IBM Power4 → IBM Power5

Support multiple instruction streams

- Increase L1 instruction cache associativity
- Increase instruction TLB associativity
- Mitigate cache contention from instruction streams
- Separate instruction prefetch and buffering per thread
- Allow thread prioritization

Support more instructions in-flight

- Increase the number of physical registers (e.g., 152 to 240)
- Mitigate register renaming bottleneck

Support larger cache footprints

- Increase L2 cache size (e.g., 1.44MB to 1.92MB)
- Increase L3 cache size
- Mitigate cache contention from data footprints

Power5 core is 24% larger than Power4 core

Instruction Scheduling

ICOUNT: Schedule thread with fewest instructions in flight

(1) prevents one thread from filling issue queue

(2) prioritizes threads that efficiently move instructions through datapath

(3) provides an even mix of threads, maximizing parallelism

SMT Performance

Intel Pentium4 Extreme SMT

- Single program
- 1.01x speedup for SPECint and 1.07x speedup for SPECfp

- Multiprogram (pairs of SPEC workloads)
- 0.9-1.6x for various pairs

IBM Power 5

- Single program
- 1.23x speedup for SPECint and 1.16x speedup for SPECfp

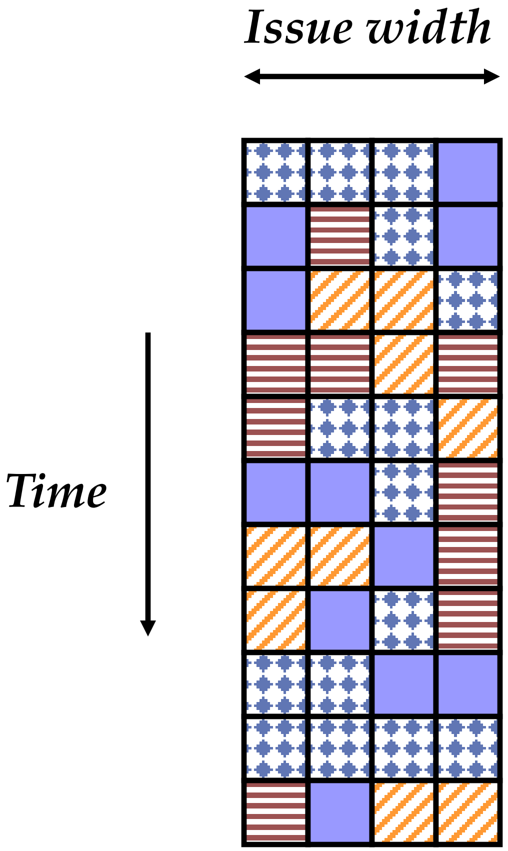
- Multiprogram (pairs of SPEC workloads)
- 0.89-1.41x for various pairs

Intuition

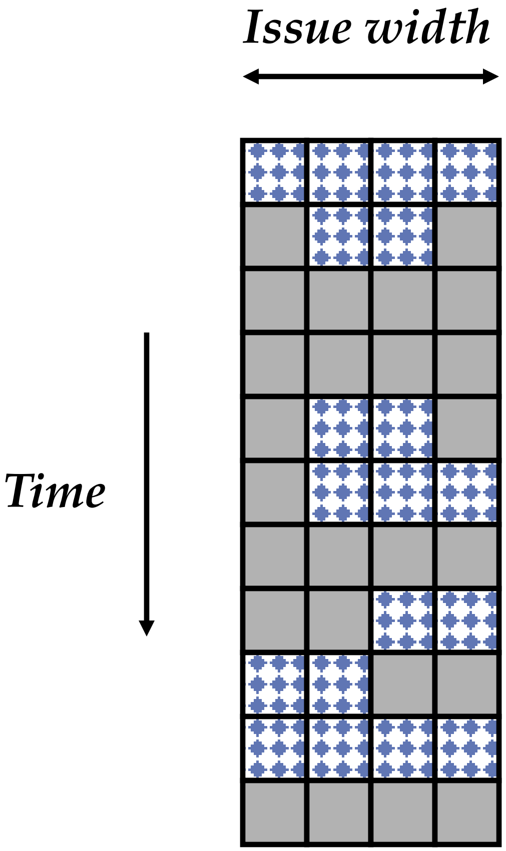
- SPECint has complex control flow, idles processor, benefits from SMT
- SPECfp has large data sets, cache conflicts, fewer benefits from SMT

SMT Flexibility

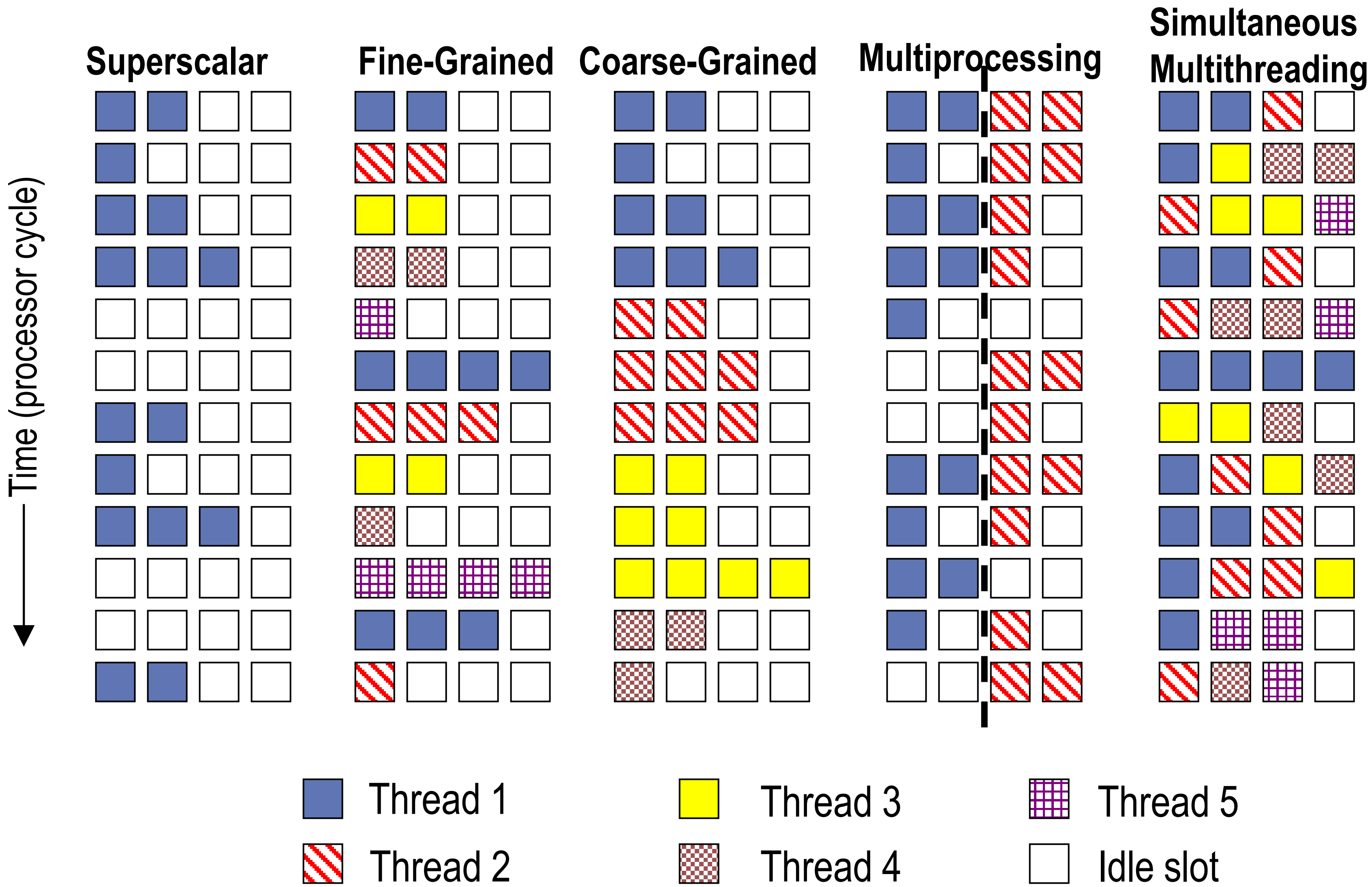
For SW regions with high thread level parallelism (TLP), share machine width across all threads



For SW regions with low thread level parallelism (TLP), reserve machine width for single thread instruction level parallelism (ILP)



Types of Multithreading



Superscalar

Fine-Grained

Coarse-Grained

Multiprocessing

Simultaneous Multithreading

Time (processor cycle)

🟦 Thread 1

▨ Thread 2

🟨 Thread 3

▩ Thread 4

▦ Thread 5

□ Idle slot

Summary

Out-of-order Superscalar

- Processor pipeline is under-utilized due to data dependencies

Thread-level Parallelism

- Independent threads more fully use processor resources

Multithreading

- Reduce vertical waste by scheduling threads to hide long latency operations (e.g., cache misses)
- Reduce horizontal waste by scheduling threads to more fully use superscalar issue width

Acknowledgements

These slides contain material developed and copyright by

- Arvind (MIT)
- Krste Asanovic (MIT/UCB)
- Joel Emer (Intel/MIT)
- James Hoe (CMU)
- Arvind Krishnamurthy (U. Washington)
- John Kubiatowicz (UCB)
- Alvin Lebeck (Duke)
- David Patterson (UCB)
- Daniel Sorin (Duke)