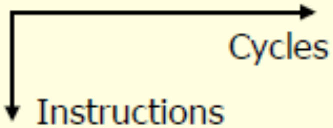
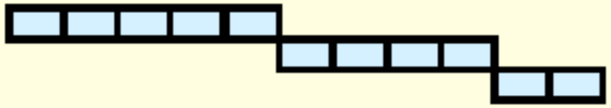
Multithreading: Exploiting Thread-Level Parallelism within a Processor

- Instruction-Level Parallelism (ILP): What we've seen so far
- Wrap-up on multiple issue machines
- ***Beyond ILP***

  → ***Multithreading***

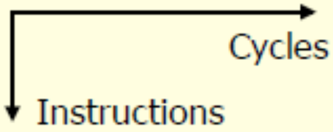
Instruction Level Parallelism (ILP): So Far…

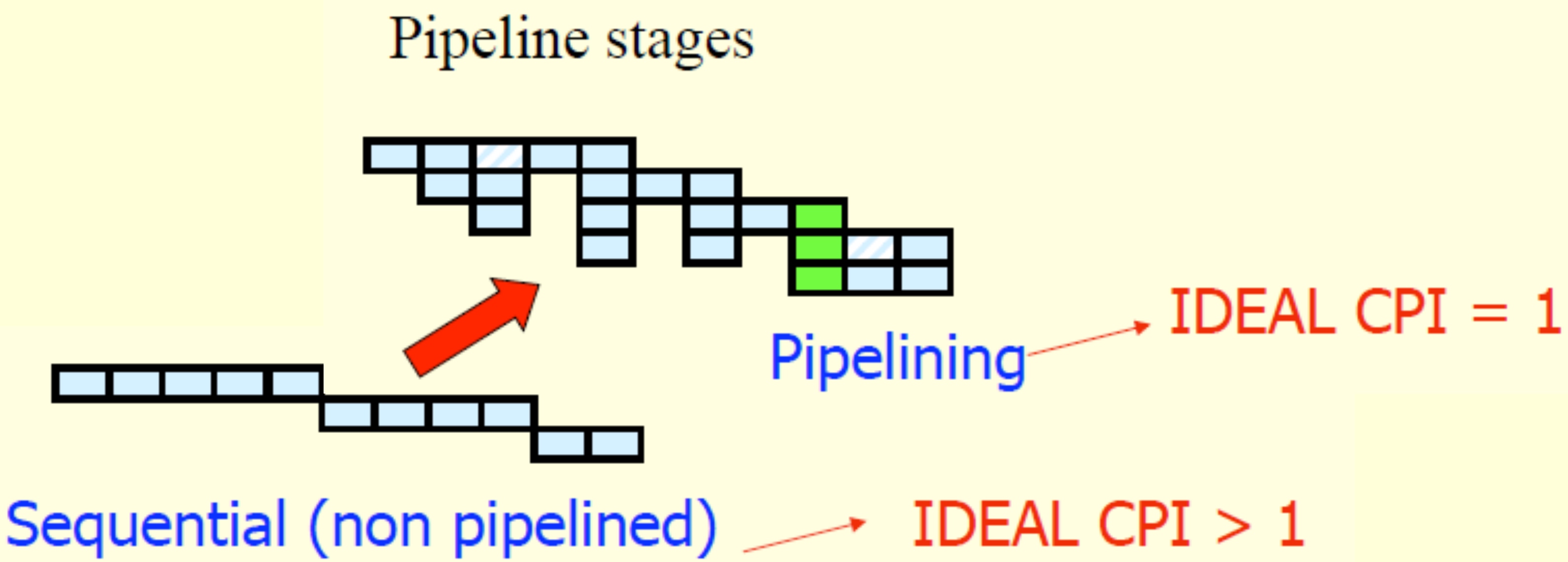




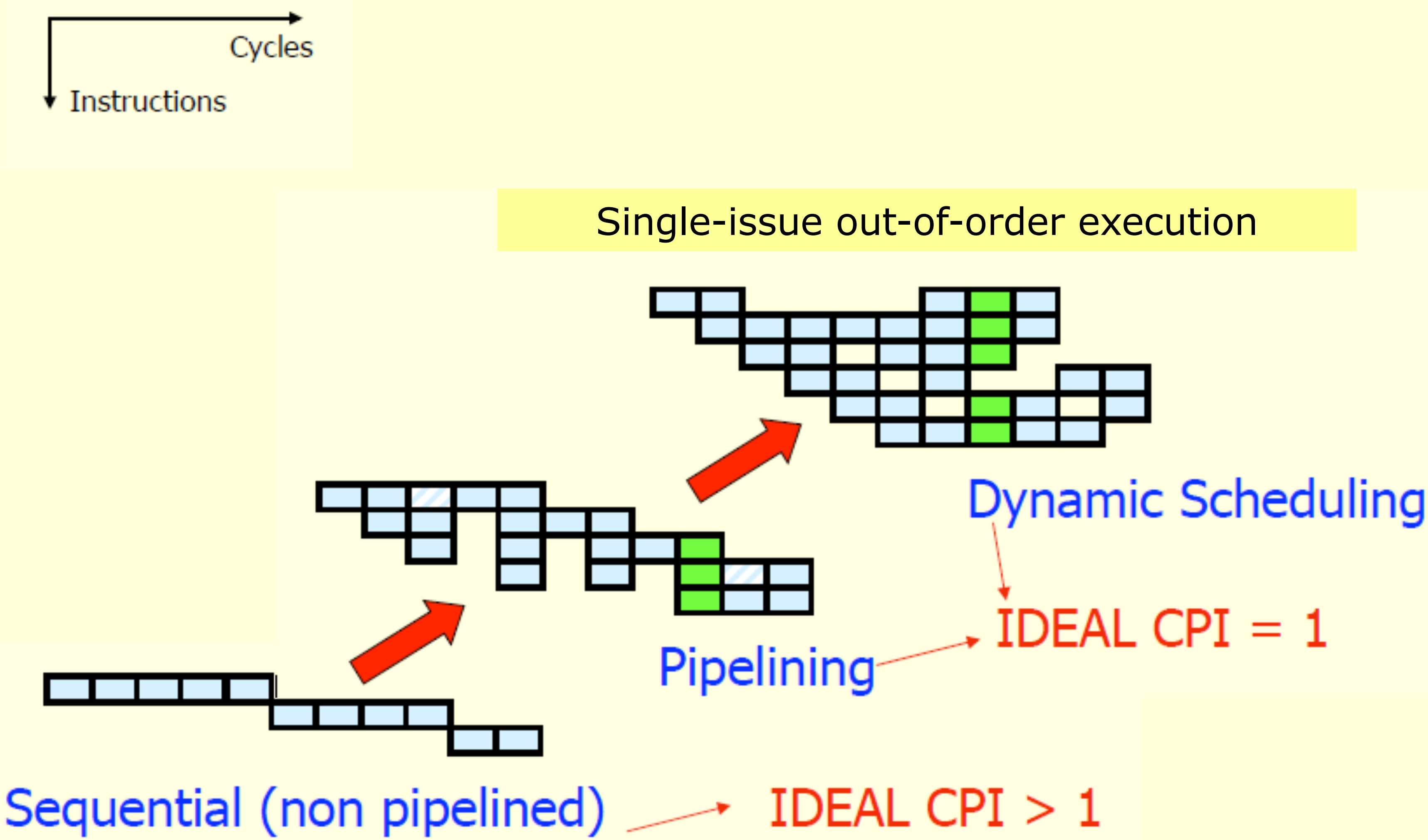
Sequential (non pipelined) → IDEAL CPI > 1

Instruction Level Parallelism (ILP): So Far…

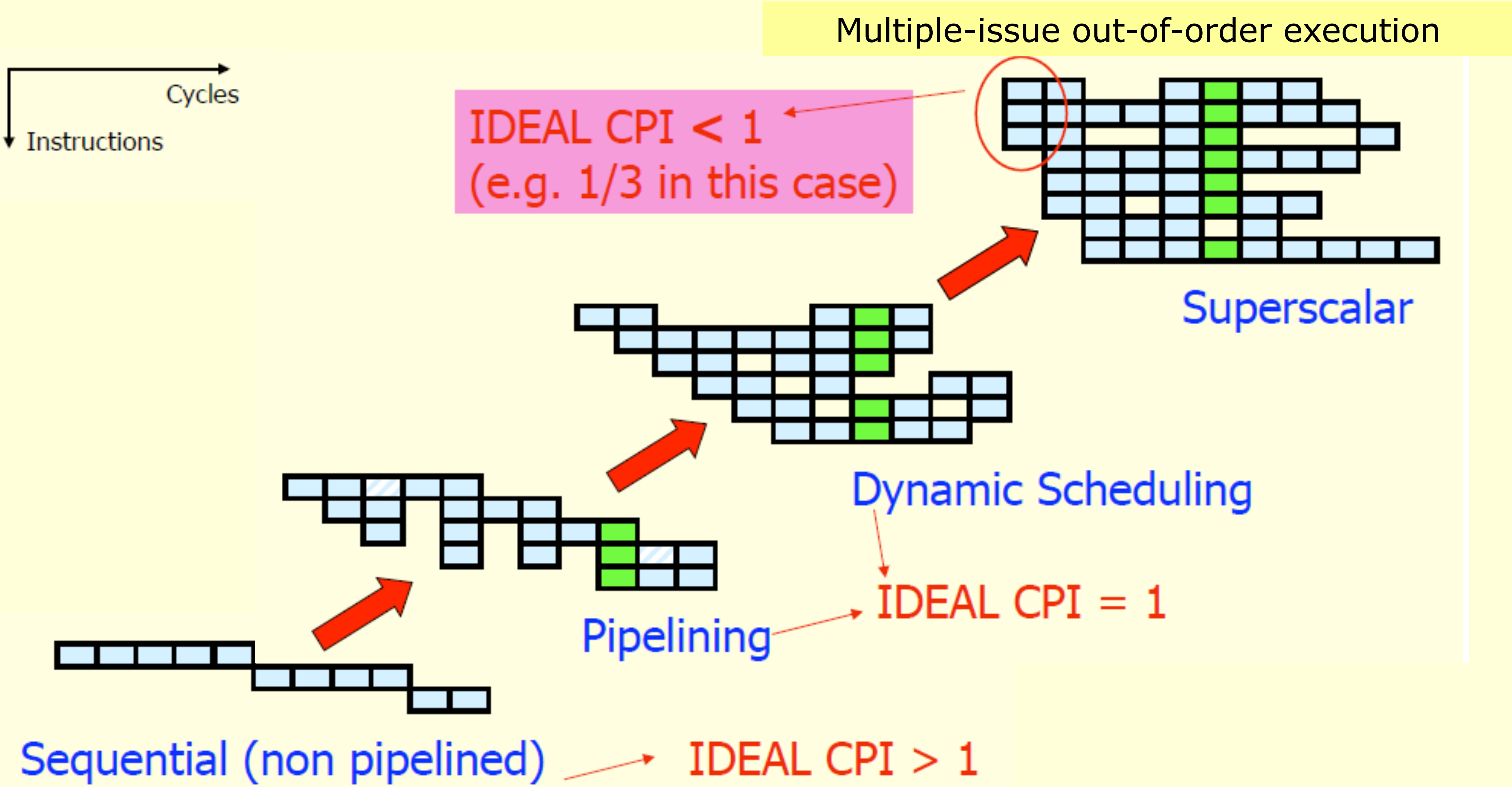




Instruction Level Parallelism (ILP): So Far…



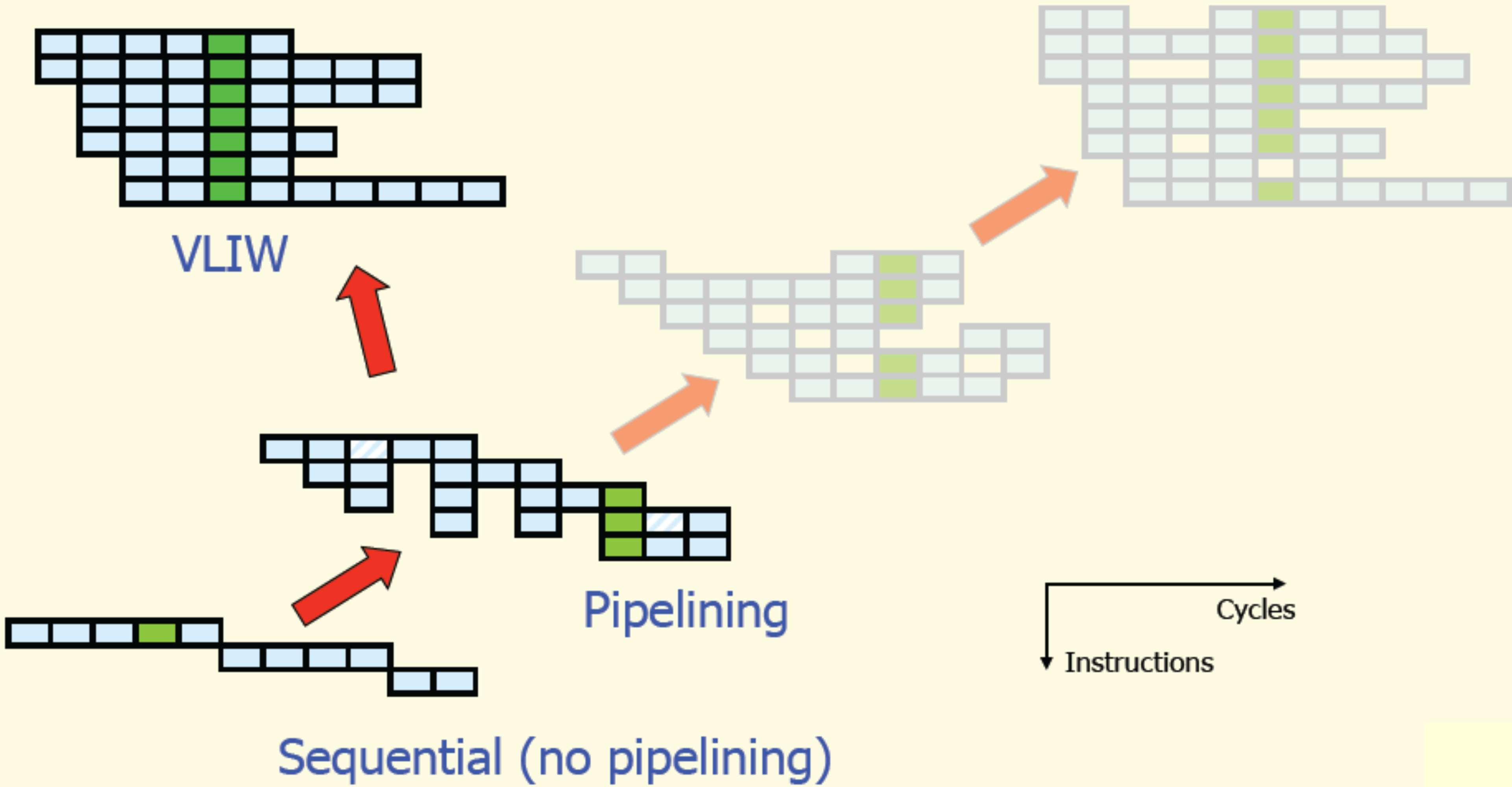
Instruction Level Parallelism (ILP): So Far…



What to do to avoid dynamic scheduling costs

- But to keep high performance?
- Go towards ***compile-time scheduling (static scheduling):*** *Why not let the compiler decide which instructions can execute in parallel at every cycle?*
- Instead of run-time (dynamic) scheduling….

Very Long Instruction Word: An Alternative Way of Extracting ILP



Superscalar vs VLIW Scheduling

Deciding ***when*** and ***where*** to execute an instruction – i.e. in which cycle and in which functional unit

1. For a superscalar processor it is decided at ***run-time,*** by custom logic in HW
2. For a VLIW processor it is decided at ***compile-time,*** by the compiler, and therefore by a SW program
   - Good for embedded processors: Simpler HW design (no dynamic scheduler), smaller area and cheap

Challenges for VLIW

- Compiler technology
  - The compiler needs to find a lot of parallelism in order to keep the multiple functional units of the processors busy
- Binary incompatibility
  - Consequence of the larger exposure of the microarchitecture (= implementation choices) at the compiler in the generated code

Current Superscalar & VLIW processors

- Dynamically-scheduled superscalar processors are the commercial state-of-the-art for general purpose: current implementations of Intel Core i, PowerPC, Alpha, MIPS, SPARC, etc. are all superscalar
- VLIW processors are primarily successful as embedded media processors for consumer electronic devices (embedded):
  - TriMedia media processors by NXP
  - The C6000 DSP family by Texas Instruments
  - The ST200 family by STMicroelectronics
  - The SHARC DSP by Analog Devices
  - Itanium 2 is the only general purpose VLIW, a ‘hybrid’ VLIW (EPIC, Explicitly Parallel Instructions Computing)

Issue-width limited in practice

- The issue width is the number of instructions that can be issued in a single cycle by a multiple issue (also called ILP) processor
  – And fetched, and decoded, etc.
- When superscalar was invented, 2- and rapidly 4-issue width processors were created (i.e. 4 instructions executed in a single cycle, ideal CPI = 1/4)

IF	ID	EX	MEM	WB				
IF	ID	EX	MEM	WB				
	IF	ID	EX	MEM	WB			
	IF	ID	EX	MEM	WB			
		IF	ID	EX	MEM	WB		
		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB
				IF	ID	EX	MEM	WB

Issue-width limited in practice

- Now, the maximum (rare) is 6, but no more exists.
  – The widths of current processors range from **single-issue** (ARM11, UltraSPARC-T1) through **2-issue** (UltraSPARC-T2/T3, Cortex-A8 & A9, Atom, Bobcat) to **3-issue** (Pentium-Pro/II/III/M, Athlon, Pentium-4, Athlon 64/Phenom, Cortex-A15) or **4-issue** (UltraSPARC-III/IV, PowerPC G4e, Core 2, Core i, Core i*2, Bulldozer) or **5-issue** (PowerPC G5), or even **6-issue** (Itanium, but it's a VLIW).
- Because it is too hard to decide which 8, or 16, instructions can execute every cycle (too many!)
  – It takes too long to compute
  – So the frequency of the processor would have to be decreased

12

Taxonomy of Multiple Issue Machines

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)					Mostly in the embedded space: MIPS and ARM, including the ARM Coretex A8
Superscalar (dynamic)					None at the present
Superscalar (speculative)					Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW					Most examples are in signal processing, such as the TI C6x
EPIC					Itanium

Taxonomy of Multiple Issue Machines

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic	Hardware	Static	In-order execution	Mostly in the embedded space: MIPS and ARM, including the ARM Coretex A8
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None at the present
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW	Static	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)	Most examples are in signal processing, such as the TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by the compiler	Itanium

What we have learnt so far

- What is static scheduling as opposed to dynamic scheduling
- What are their advantages and disadvantages
- What can HW do better than SW in scheduling, and what can SW do better than HW
- Which processors implement which strategy and why
- What compiler techniques can be used to increase ILP

- References:
  - HP-QA 4^{th} edition Chapter 2 (especially sec 2.7)
  - HP-QA 4^{th} edition Appendix G

BEYOND ILP

- **Multithreading   (Thread-Level Parallelism, TLP)**
- **Multiprocessing:** Multiple processors
- **Data Level Parallelism:** Perform identical operations on data, and lots of data

Yet another way to push performance

- So far, we've studied two types of ILP:
  - Out-of-order multiple-issue dynamically scheduled processors: Aggressive superscalars.
  - Simpler core, statically scheduled, push parallelism detection towards the compiler: VLIW and EPIC processors.
- **Main limitation of ILP:** degree of intrinsic parallelism in a single instruction stream, i.e. limited amount of parallelism to be exploited at the instruction-level.

→ Simpler core, to exploit Thread-Level Parallelism instead of ILP

  - Simple multithreading such as the Sun Niagara T1

Performance beyond single thread ILP

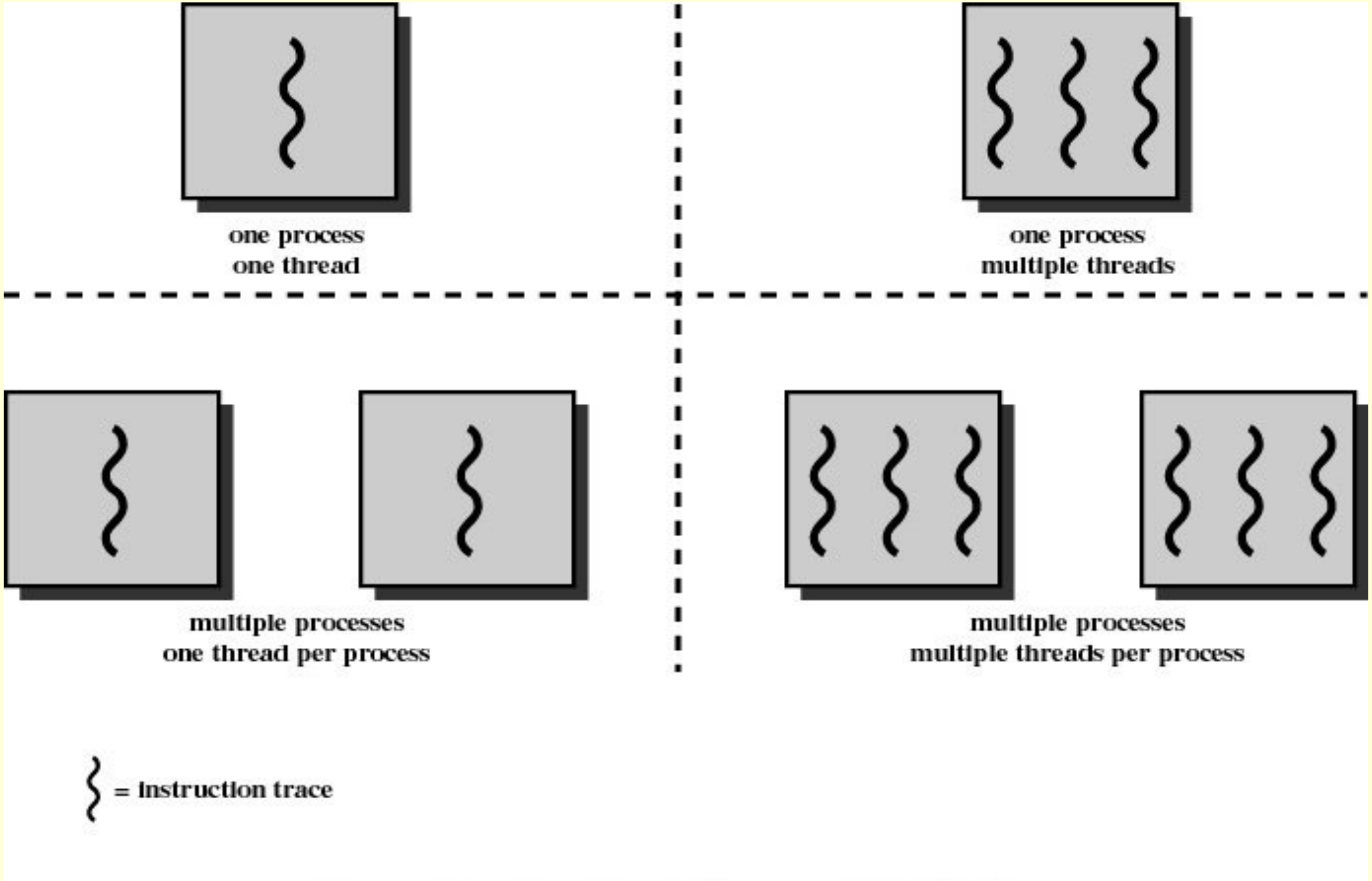
- There can be more intrinsic parallelism in some applications (e.g., database or scientific codes).

- Explicit **Thread Level Parallelism** or **Data Level Parallelism:**

1. **Thread:** lightweight process with own instructions and data
   - Thread may be a process part of a parallel program of multiple processes, or it may be an independent program
   - Each thread has all the state (instructions, data, PC, register state, and so on) necessary to allow it to execute

2. **Data Level Parallelism:** Perform identical operations on data, and lots of data (e.g., managing vectors and matrices).

Multiple process/thread

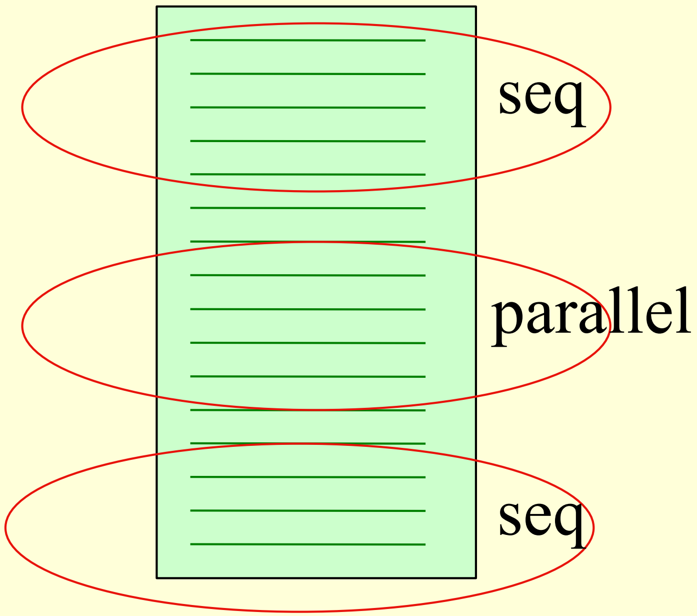


What is a thread?

- A lightweight process with own instructions and data.
- Threads can be created either *explicitly* by the programmer or *implicitly* by the OS.
- Amount of computation assigned to each thread is the *grain size:*
  - Can be from a few instructions (more suited to uniprocessor multithreading: for *1* processor, need *n* threads)
  - To hundreds, or thousands of instructions (multiprocessor multithreading: for *n* processors, need *n* threads)

Threads

Independent sequences of instructions

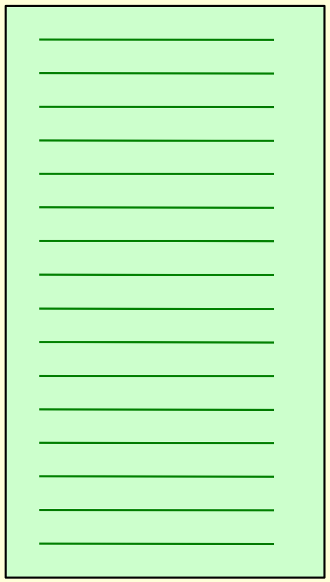


Single-threaded program

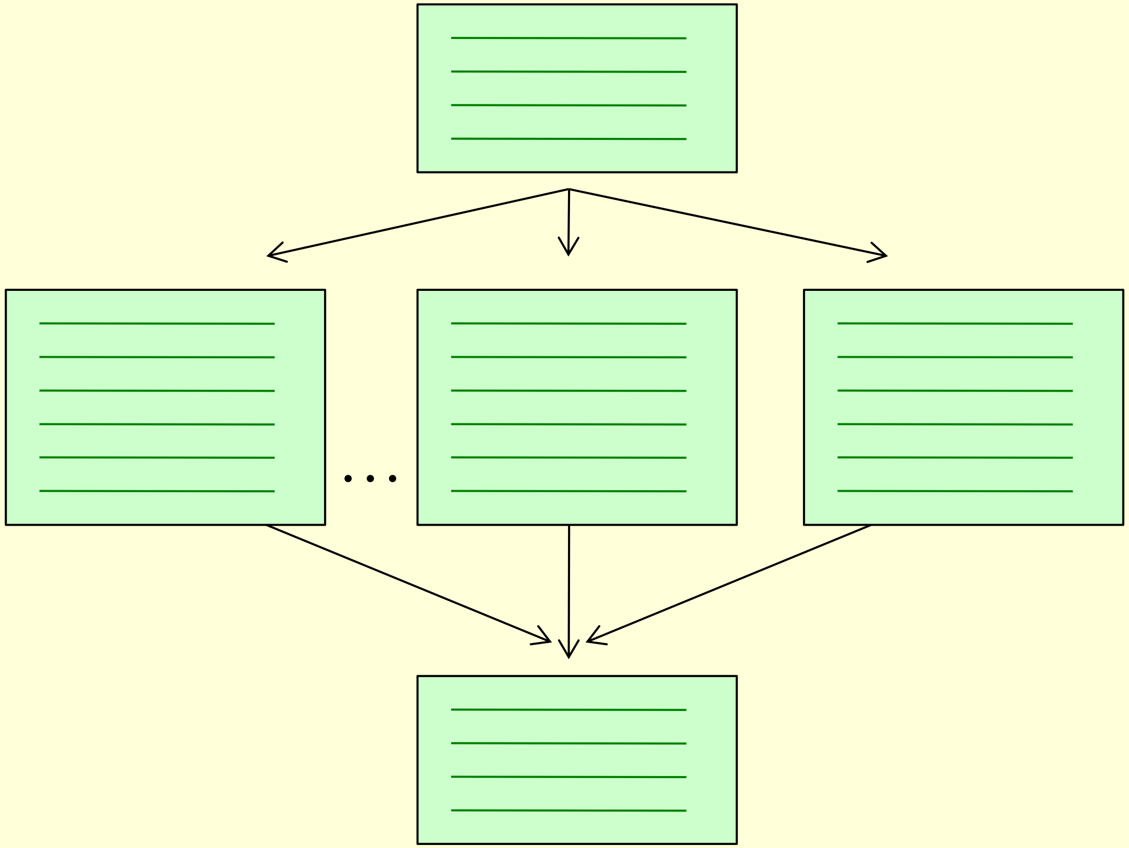
```
e.g.:   for (i=0; i<100; i++){
                A[i]=A[i]+B[i]*4;
                D[i]=B[i]+A[i];
                }
```

Threads

Independent sequences of instructions



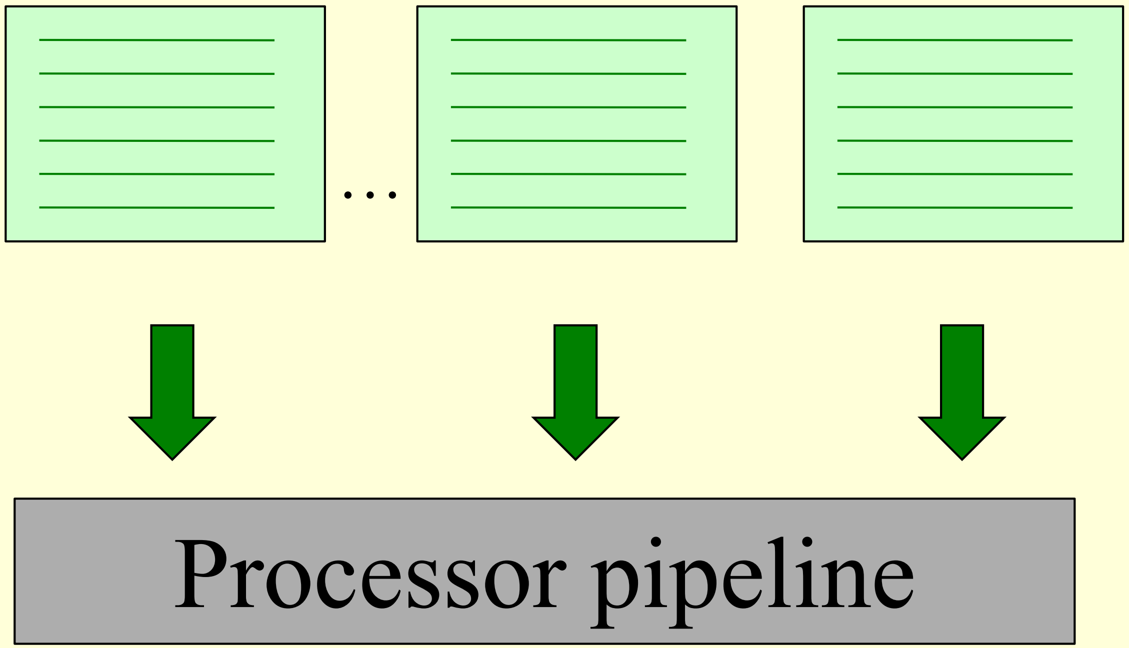
Single-threaded program



Multi-threaded program

Multithreading

Running more than one thread in parallel



Fetch and execute instructions from multiple threads

To keep the functional units busy beyond ILP

Hardware Multithreading

- No need for context switch with intervention of the Operating System.
- Multithreading is managed directly by the hardware processor architecture by additional resources.

What additional HW is needed in a multithreaded processor?

- The processor switches between different threads: when one thread stalls, another goes into execution.

→ The ***state*** of each thread must be preserved while the processor switches

→ **Need multiple Register Files and multiple Program Counters.**

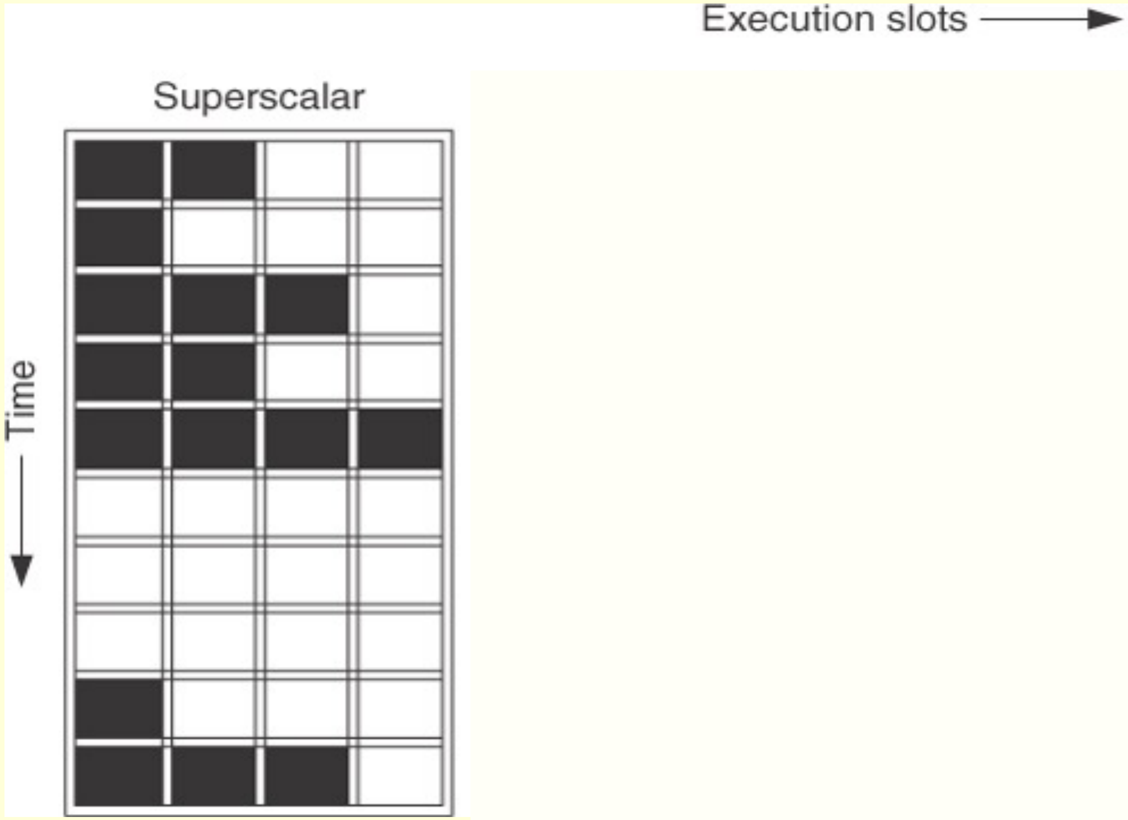
- Other parts, such as the functional units, can be ***shared*** among the threads.

HW support for multithreading

- The processor must ***duplicate*** the independent state of each thread: separate copy of the register set and separate PC for each thread.
- Multithreading allows multiple threads ***to share*** the functional units of a single processor.
- The memory address space can be ***shared*** through the virtual memory mechanism.
- The HW must support the ability to change to a different thread relatively quickly (more efficiently than a process context switch).

Types of Multithreading

- Several different types of multithreading for a superscalar processor:
  - **Coarse-grained Multithreading:** when a thread is stalled, perhaps for a cache miss, another thread can be executed;
  - **Fine-grained Multithreading:** switching from one thread to another thread on each instruction, such as Sun Niagara T1 and T2;
  - **Simultaneous Multithreading:** multiple thread are using the multiple issue slots in a single clock cycle → ***exploiting TLP and ILP,*** such as IBM Power and Intel Core i7.



Let's consider a 4-isse superscalar processor.

Horizontal dimension represents the instruction issue slots at each clock cycle.

Vertical dimension represents the time in clock cycles.

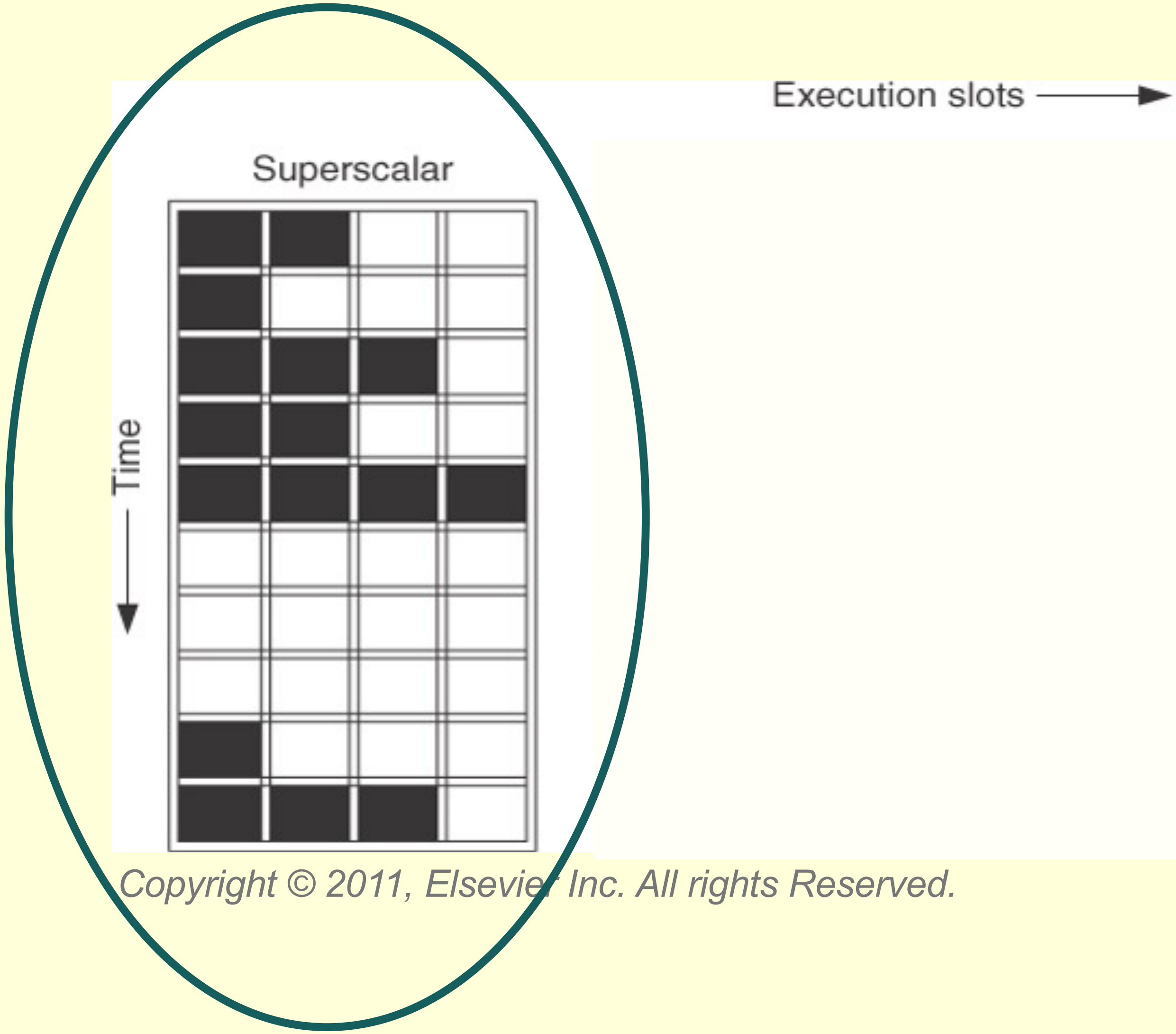
Empty (white) box indicates that the execution slot is unused in that clock cycle.

The shades of gray and black correspond to four different threads in the multithreading processors.

Black is also used to indicate the occupied issue slots in the case of the superscalar without multithreading

Hennessy Patterson A Quantitative Approach ed 4th and 5th.

Superscalar with no multithreading

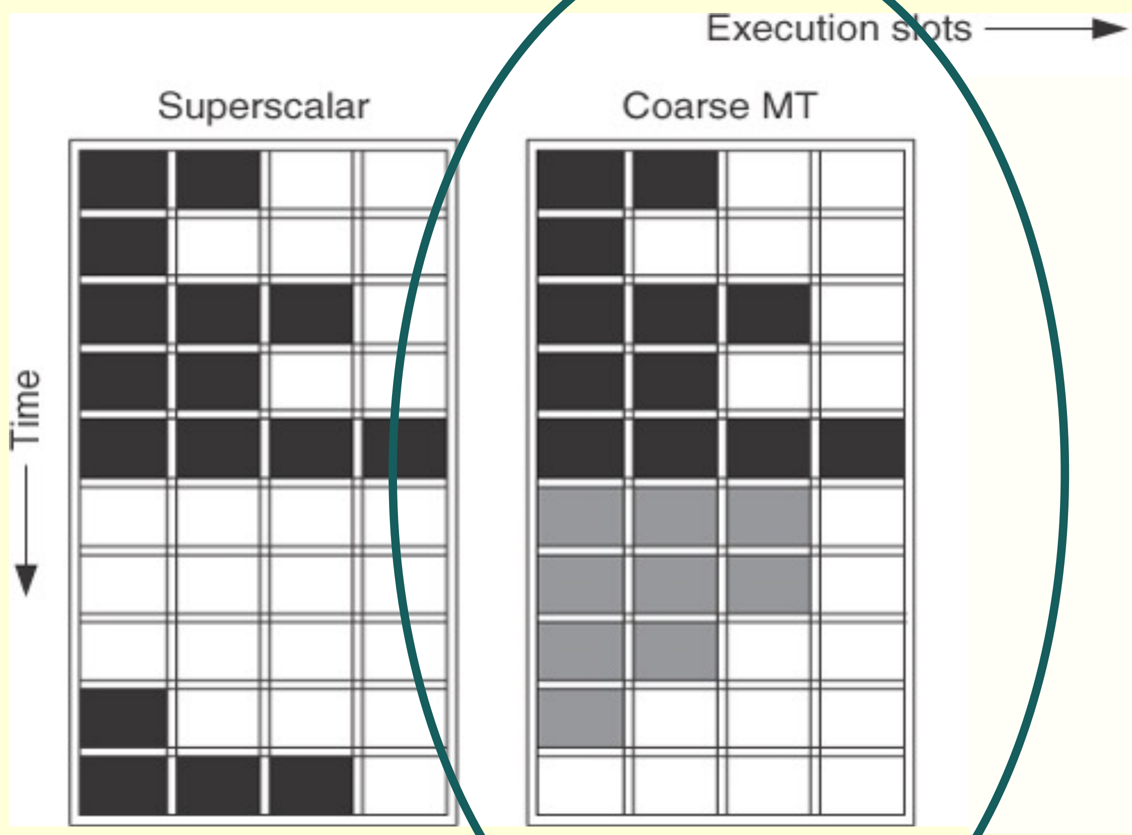


Superscalar with no multithreading

- The use of issue slots is limited by a lack of ILP.
- Multiple-issue processors often have more functional units parallelism available than a single thread can effectively use by ILP.
- A long stall, such as an instruction cache miss, can leave the entire processor idle for some clock cycles

→ ***Basic idea: in the empty slots due to a long stall of the single black thread, put another thread!***

Coarse-grained Multithreading



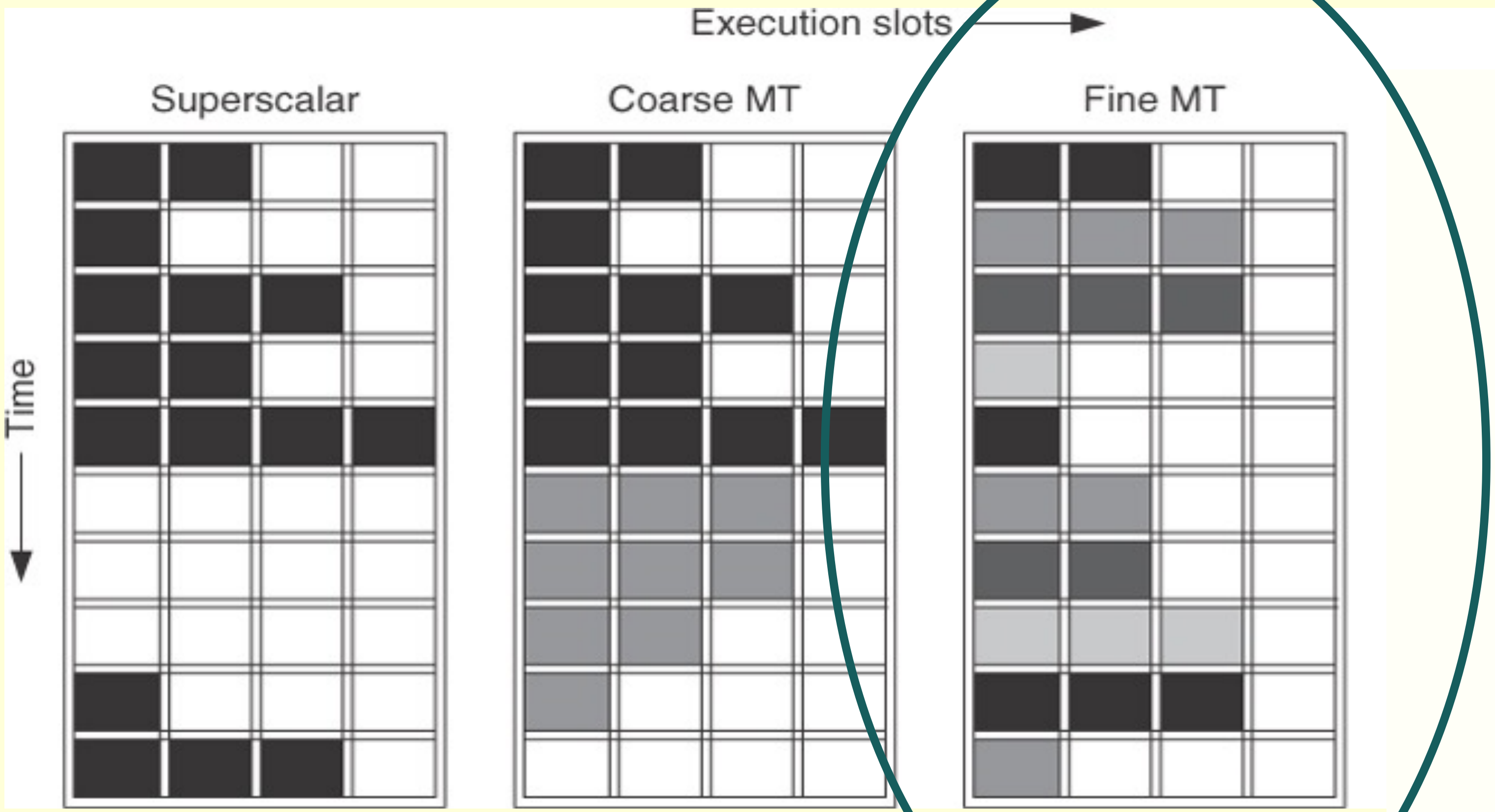
Coarse-grained Multithreading

- Long stalls (such as L2 cache misses) are hidden by switching to another thread that uses the resources of the processor.
- This reduces the number of idle cycles, but:
  - Wihin each clock, ILP limitations still lead to empty issue slots;
  - When there is one stall, it is necessary to empty the pipeline before starting the new thread;
  - The new thread has a pipeline start-up period with some idle cycles remaining and loss of throughput
  - Because of this start-up overhead, coarse-grained MT is better for reducing penalty of high-cost stalls, where pipeline refill << stall time

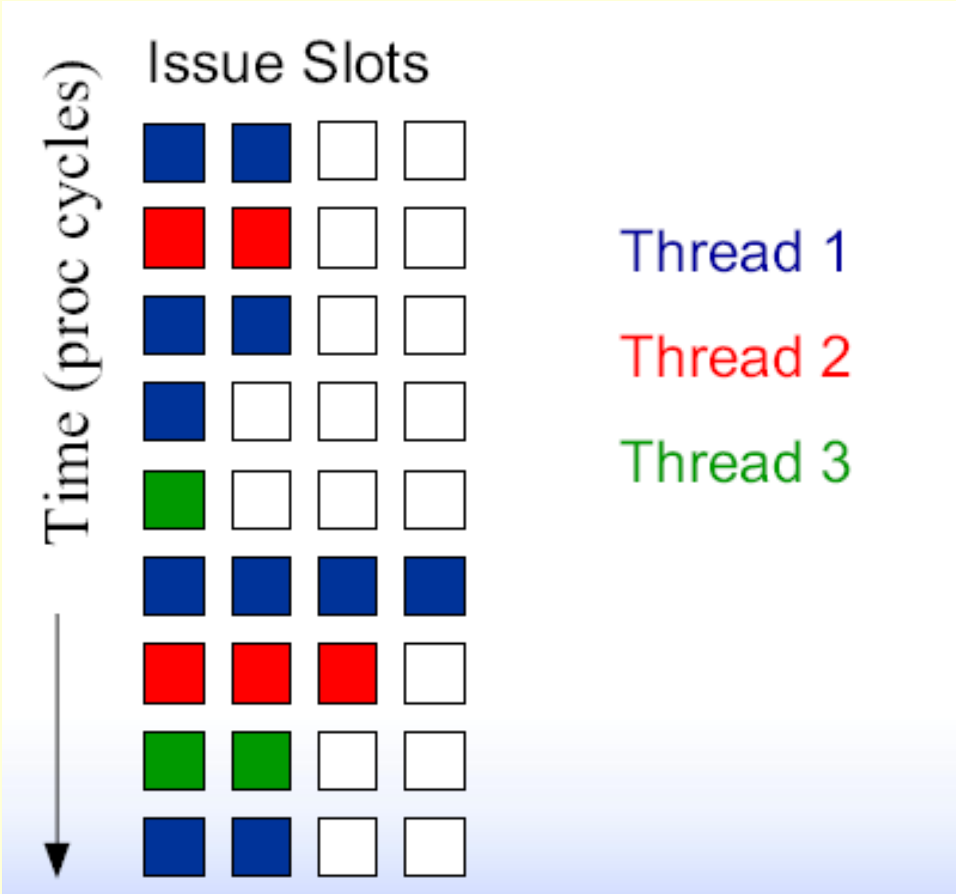
Fine-grained Multithreading

→ ***Basic idea: At each clock cycle we must switch to another thread, in a sort of round-robin among active threads!***

Fine-grained Multithreading



Fine-grained Multithreading



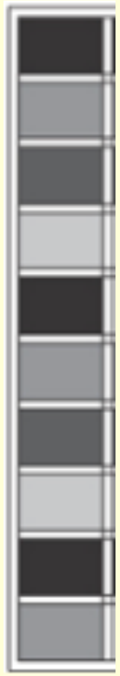
Fine-grained Multithreading

- Fine-grained MT switches between threads on each instruction → execution of multiple thread is interleaved in a round-robin fashion, skipping any thread that is stalled at that time eliminating fully empty slots.
  - The processor must be able to switch threads on every cycle.
  - It can hide both short and long stalls, since instructions from other threads are executed when one thread stalls.
  - It slows down the execution of individual threads, since a thread that is ready to execute without stalls will be delayed by another threads.
  - Within each clock, ILP limitations still lead to empty issue slots

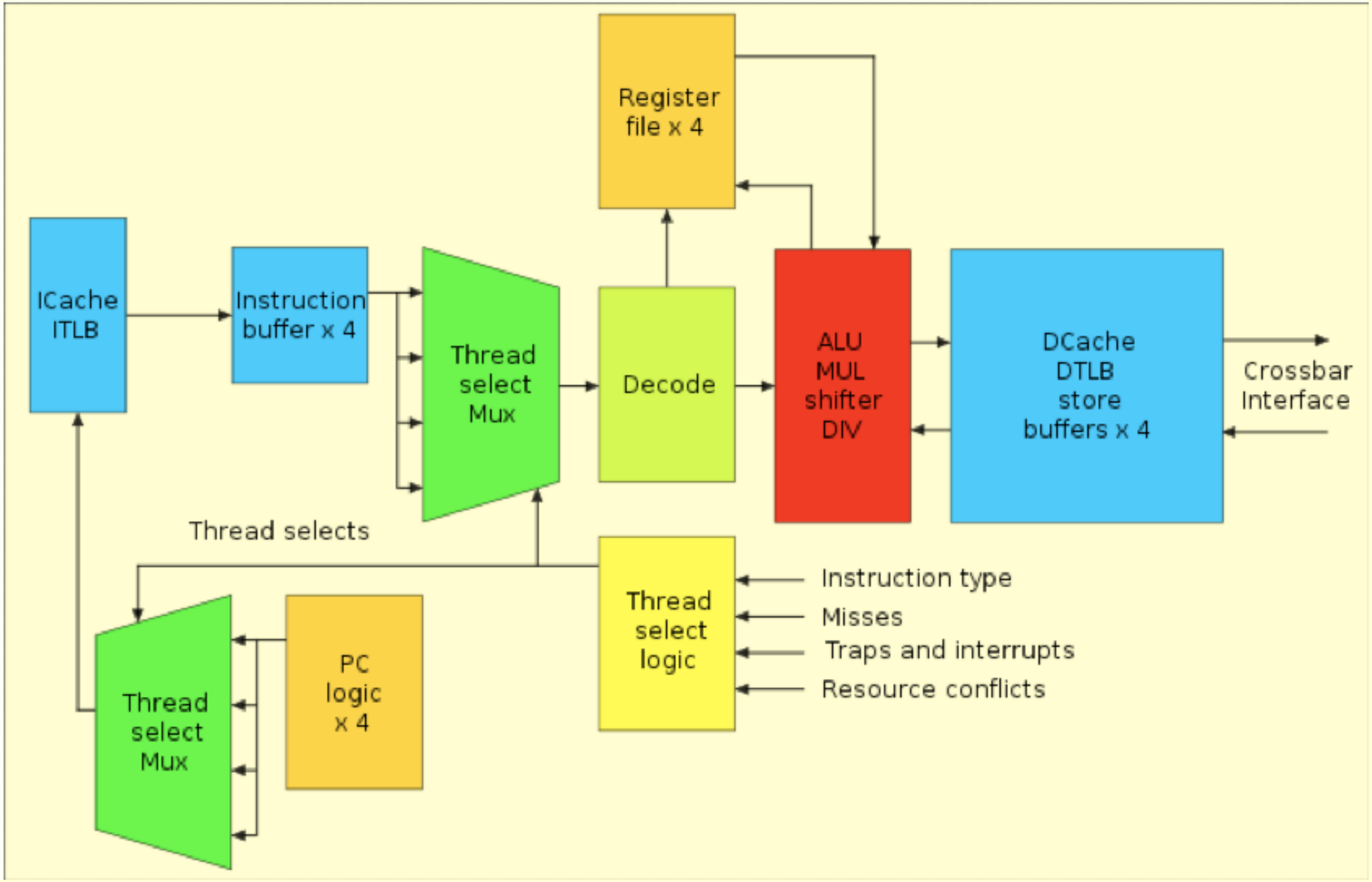
Example of Fine-Grained MT: Sun Niagara T1

- Very simple single-issue processor core that can handle by hardware up to 4 threads.

- Fine-Grain Multithreading: each cycle each processor core can switch between 4 threads.



Example: Architecture of SUN Niagara T1

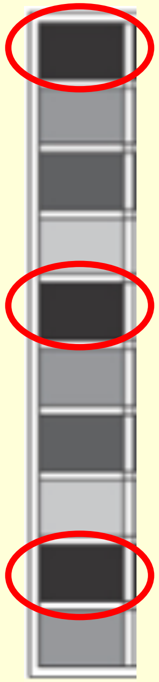


Example: Sun Niagara T1

- Whenever a thread is stalled (for dependence, for a cache miss etc), the core executes another thread
- Only when all four threads are stalled, the core is stalled; otherwise, it's always busy
- Simple, single-issue pipeline, but multithreaded

Multithreading to hide latency

- In practice, MT is an effective way to hide long-latency events in a processor and keep the execution units busy
  - → If each thread is visited every (e.g.) 4 cycles, then the dependent instruction have 4 ‘free’ cycles before the dependence is resolved



Sun Niagara T1 Performance

Benchmark	Per-thread CPI	Per-core CPI
TPC-C	7.2	1.80
SPECJBB	5.6	1.40
SPECWeb99	6.6	1.65

Ideal per-thread CPI = 4
(each thread ideally produces a new results every cycle, and there are 4 threads being served by a single-issue core)

Ideal per-core CPI = 1
(it's a single-issue core)

Multicore Sun Niagara T1

- Since the single-issue core is really simple → T1 can pack 8 cores on-chip
- There are 8 cores on the die and each core can handle 4 threads → **up to 32 threads**

Discussion on Performance

Benchmark	Per-thread CPI	Per-core CPI	Effective CPI for 8 cores	Effective IPC for 8 cores
TPC-C	7.2	1.80	0.225	4.4
SPECJBB	5.6	1.40	0.175	5.7
SPECWeb99	6.6	1.65	0.206	4.8

1.8 CPI does not sound so high compared to an aggressive dynamically scheduled superscalar ILP core (ideal CPI = ¼ …)

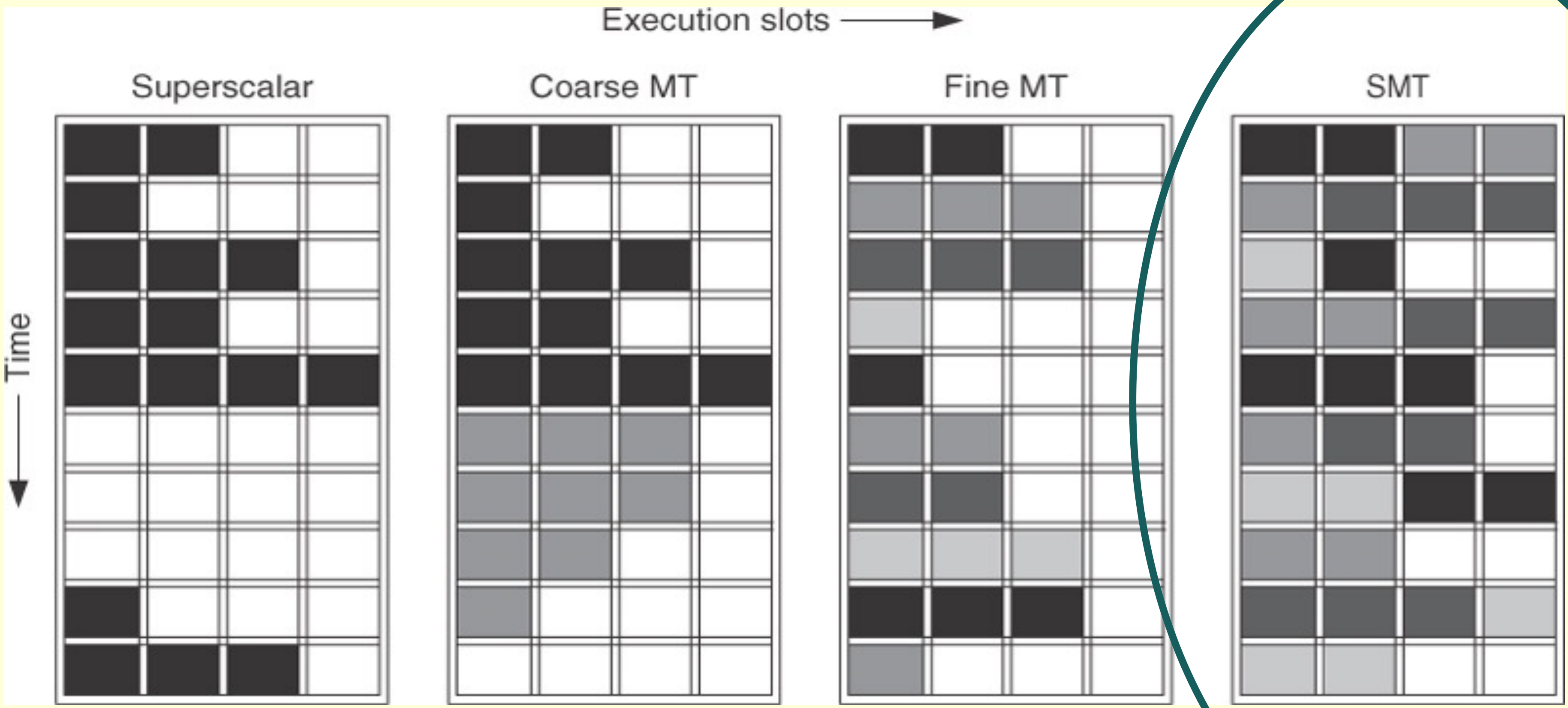
BUT:

Since the core is really simple (single-issue!), T1 can pack 8 cores in a chip, while the more aggressive ILP ones could have only 2 to 4 cores in a chip

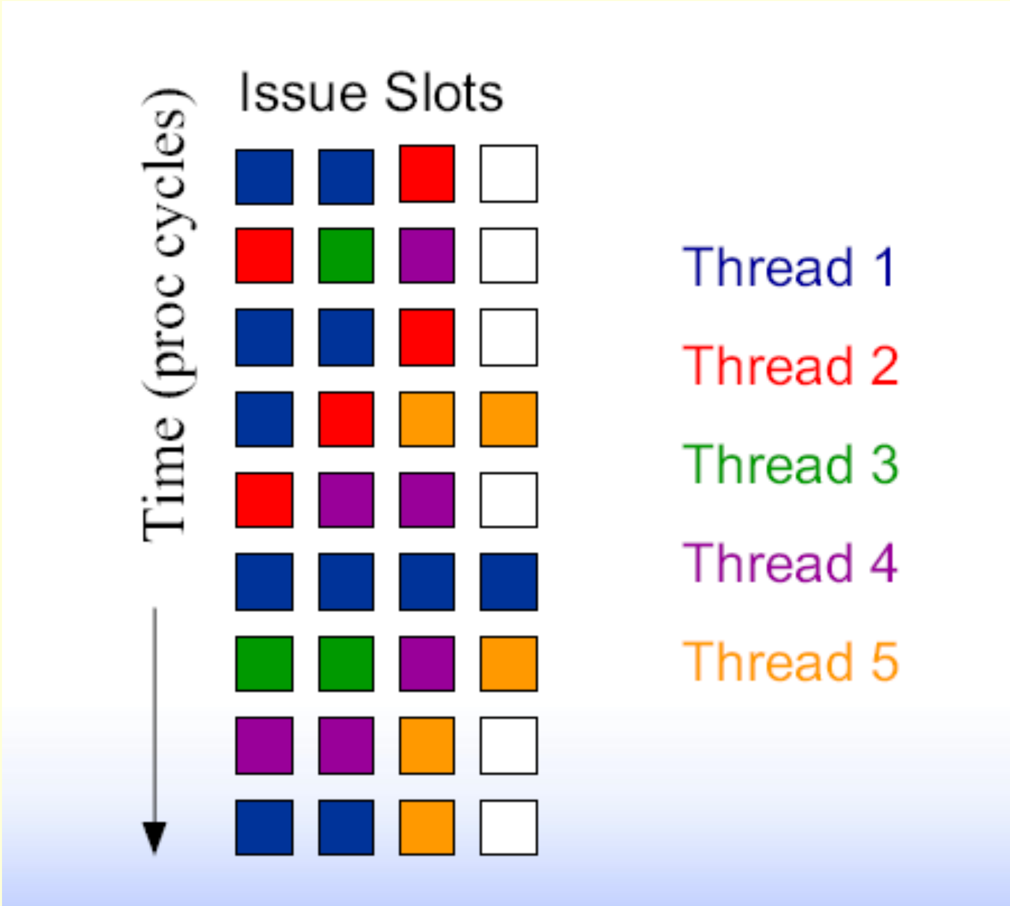
As a result, performance is comparable, if not better

Simultaneous Multithreding (SMT)

Why not to do both TLP and ILP?



Simultaneous Multithreding (SMT)

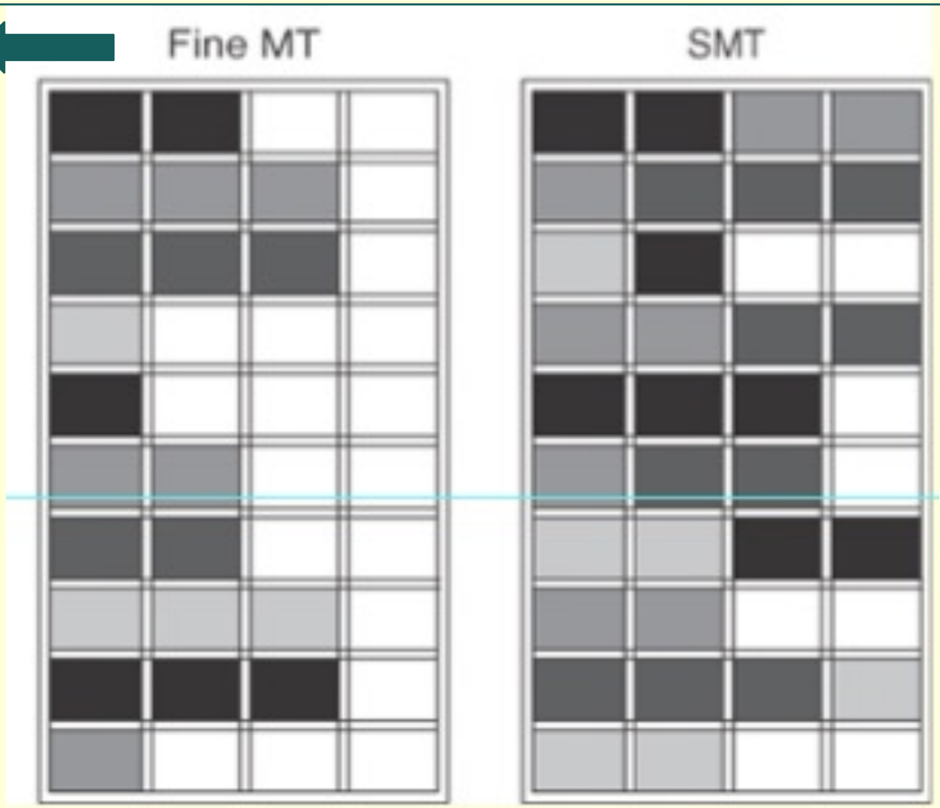


SMT

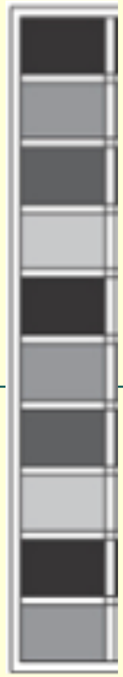
- Key motivation: a CPU today has more functional resources that what one thread can actually use
- Simultaneously schedule instructions for execution from all threads to maximize the use of resources.
- It's the most common implementation of multithreading today: Intel Core i7, IBM Power7
- It arises naturally when Fine MT is implemented on top of a multiple-issue, dynamically-scheduled processor

Exploit more parallelism than fine MT

Fine MT makes sense in small-issue machines (1 or 2-issue)

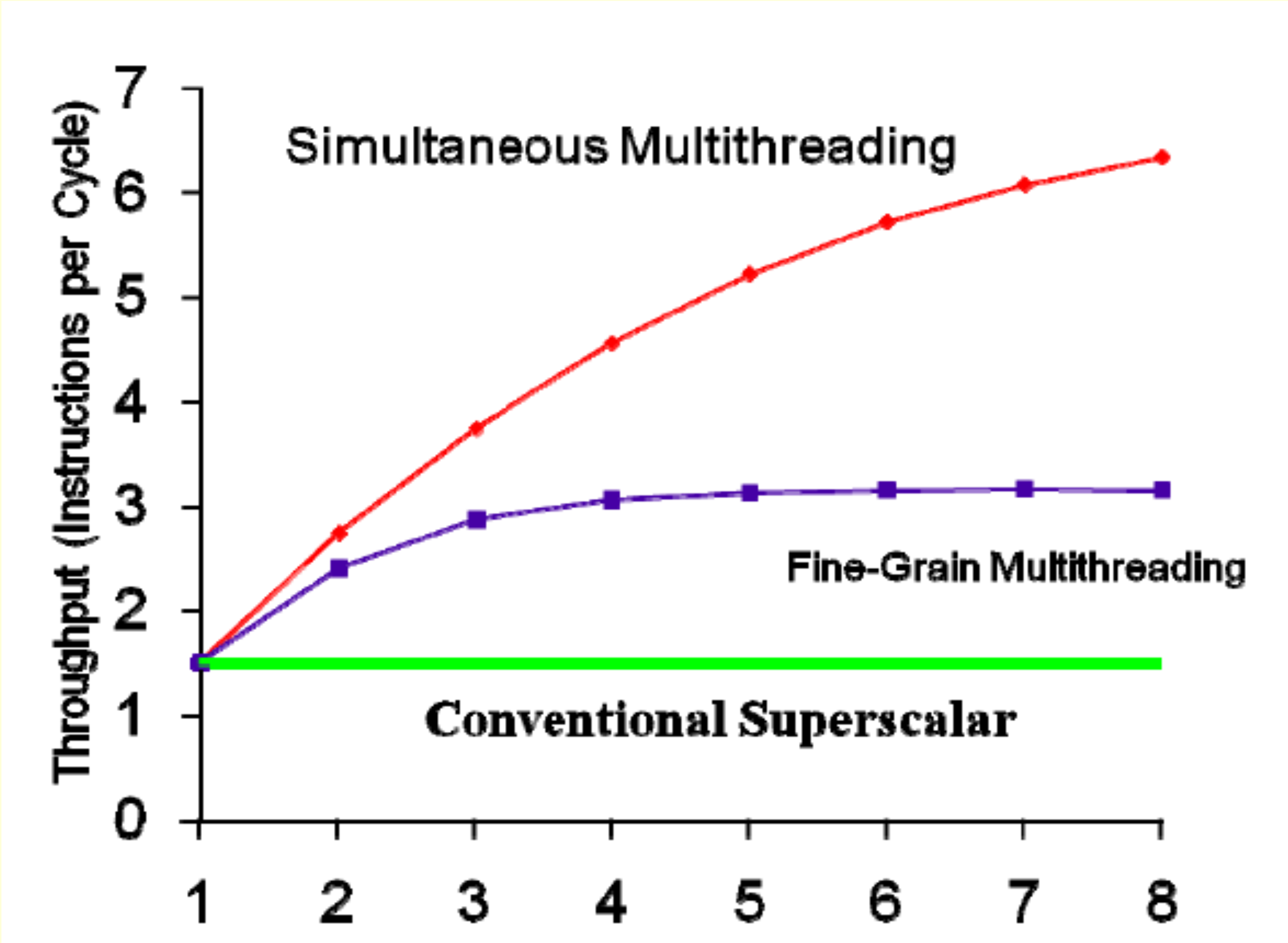


In fact, T1 Niagara is a fine-MT single-issue:



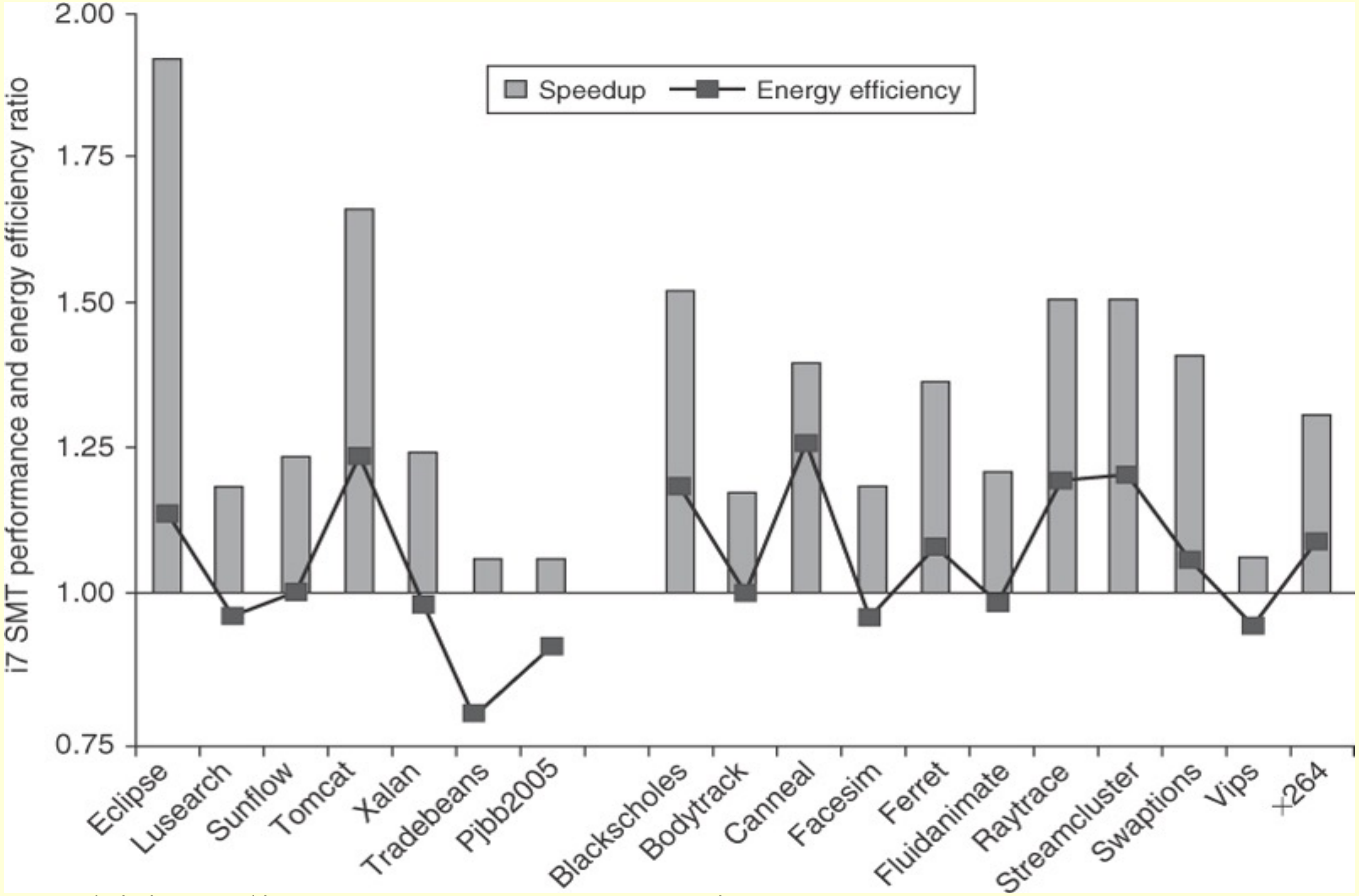
But in a large-issue machine (4-issue), where there are many slots to keep busy, SMT is the most obvious choice: let instructions from different threads be executed in the same cycle

Performance comparison



Number of threads

SMT performance



- The speedup from using multithreading on one core on an i7 processor
- The speedup averages 1.28 for the Java benchmarks and 1.31 for the PARSEC benchmarks.
- The energy efficiency averages 0.99 and 1.07, respectively.
- A value of energy efficiency **above 1.0 for energy efficiency indicates that the feature reduces execution time by more than it increases average power**
- These data were collected and analyzed by Esmaeilzadeh et al. [2011]

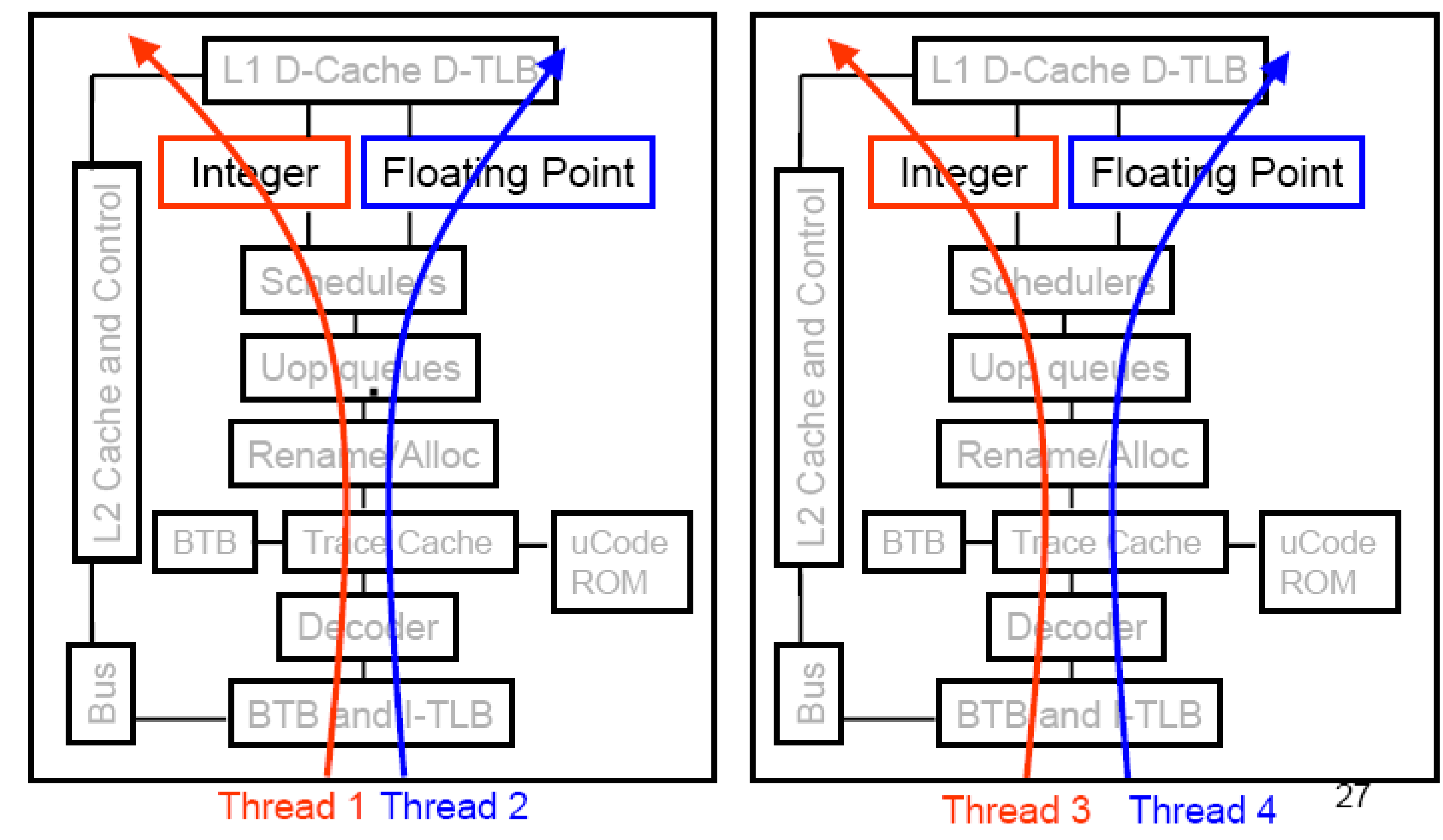
SMT

- The threads in an SMT design are all sharing just one processor core, and just one set of caches, has major performance downsides compared to a true multiprocessor (or multi-core).
- On the other hand, applications which are limited primarily by memory latency (but not memory bandwidth), such as database systems and 3D graphics rendering, benefit dramatically from SMT, since it offers an effective way of using the otherwise idle time during cache misses
- Thus, SMT presents a very complex and application-specific performance scenario.

Multicore and SMT

- Multiple cores where each processor can use SMT
- Number of threads: 2, 4 or sometime 8
    - (called ***"hyperthreading"*** by Intel)
- Memory hierarchy:
    - If only multithreading: all caches are shared
    - Multicore:
        - Cache L1 private
        - Cache L2 private in some architectures and shared in others
        - Memory always shared

SMT Dual core: 4 concurrent threads



Ideal per-core CPI=0.5 Ideal per-core CPI=0.5

Ideal dual-core CPI = 0.25

Ideal per-thread CPI = 1 (4 thread-per-clock)

SMT in commercial processors

- **Intel Pentium-4** was the first commercial processor to use SMT, which Intel calls *"hyper-threading“,* supporting 2 simultaneous threads. **Intel's Core i** and **Core i*2** are also 2-thread SMT, as is the low-power **Atom x86** processor. A typical quad-core **Intel Core i7** processor (max freq. 3.7 GHz) is thus an 8 thread chip.
- **IBM POWER 7** superscalar symmetric multiprocessor (45nm, up to 4.25 GHz) has up to 8 cores, and 4 threads per core, for a total capacity of 32 simultaneous threads (SMT).
- Sun was the most aggressive of all on the TLP front, with **UltraSPARC-T1** (aka: "Niagara") providing 8 simple in-order cores each with 4-thread, for a total of 32 threads on a single chip. This was subsequently increased to 8 threads per core in **UltraSPARC-T2,** and then 16 cores in **UltraSPARC-T3**, up to 128 threads!

QUIZ 1

Let's consider a single-issue processor that can manage up to 4 simultaneous threads.

*What are the values of the **ideal CPI** and the **ideal per-thread CPI?***

(SINGLE ANSWER)

1 point

- **Answer 1**: Ideal CPI = 1 & Ideal per-thread CPI = 0.25
- **Answer 2**: Ideal CPI = 0.25 & Ideal per-thread CPI =0.25
- **Answer 3**: Ideal CPI = 0.5 & Ideal per-thread CPI = 2

  Answer 4: Ideal CPI = 0.5 & Ideal per-thread CPI = 1
- **Answer 5**: Ideal CPI = 1 & Ideal per-thread CPI = 4

QUIZ 1: solution

Let's consider a single-issue processor that can manage up to 4 simultaneous threads.

*What are the values of the **ideal CPI** and **the ideal per-thread CPI?***

(SINGLE ANSWER)

1 point

- **Answer 1**: Ideal CPI = 1 & Ideal per-thread CPI = 0.25
- **Answer 2**: Ideal CPI = 0.25 & Ideal per-thread CPI =0.25
- **Answer 3**: Ideal CPI = 0.5 & Ideal per-thread CPI = 2

  Answer 4: Ideal CPI = 0.5 & Ideal per-thread CPI = 1
- **Answer 5**: Ideal CPI = 1 & Ideal per-thread CPI = 4 **(TRUE)**

QUIZ 2

Let's consider a dual-issue SMT processor that can manage up to 4 simultaneous threads.

*What are the values of the **ideal CPI** and **the ideal per-thread CPI?***

(SINGLE ANSWER)

1 point

- **Answer 1**: Ideal CPI = 1 & Ideal per-thread CPI = 0.25
- **Answer 2**: Ideal CPI = 0.25 & Ideal per-thread CPI =0.25
- **Answer 3**: Ideal CPI = 0.5 & Ideal per-thread CPI = 2
- **Answer 4**: Ideal CPI = 0.5 & Ideal per-thread CPI = 1
- **Answer 5**: Ideal CPI = 0.25 & Ideal per-thread CPI = 4

QUIZ 2: solution

Let’s consider a dual-issue SMT processor that can manage up to 4 simultaneous threads.

*What are the values of the **ideal CPI** and **the ideal per-thread CPI?***

(SINGLE ANSWER)

1 point

- **Answer 1**: Ideal CPI = 1 & Ideal per-thread CPI = 0.25
- **Answer 2**: Ideal CPI = 0.25 & Ideal per-thread CPI =0.25
- **Answer 3**: Ideal CPI = 0.5 & Ideal per-thread CPI = 2 **(TRUE)**
- **Answer 4**: Ideal CPI = 0.5 & Ideal per-thread CPI = 1
- **Answer 5**: Ideal CPI = 0.25 & Ideal per-thread CPI = 4

QUIZ 3

Let's consider a 4-issue SMT processor that can manage up to 4 simultaneous threads.

*What are the values of the **ideal CPI** and **the ideal per-thread CPI?***

(SINGLE ANSWER)

1 point

- **Answer 1**: Ideal CPI = 1 & Ideal per-thread CPI = 0.25
- **Answer 2**: Ideal CPI = 0.25 & Ideal per-thread CPI =0.25
- **Answer 3**: Ideal CPI = 0.25 & Ideal per-thread CPI = 1
- **Answer 4**: Ideal CPI = 0.25 & Ideal per-thread CPI = 4
- **Answer 5**: Ideal CPI = 4 & Ideal per-thread CPI = 0.25

QUIZ 3: solution

Let’s consider a 4-issue SMT processor that can manage up to 4 simultaneous threads.

What are the values of the ***ideal CPI*** *and* ***the ideal per-thread CPI?***

(SINGLE ANSWER)

1 point

- **Answer 1**: Ideal CPI = 1 & Ideal per-thread CPI = 0.25
- **Answer 2**: Ideal CPI = 0.25 & Ideal per-thread CPI =0.25
- **Answer 3**: Ideal CPI = 0.25 & Ideal per-thread CPI = 1 **(TRUE)**
- **Answer 4**: Ideal CPI = 0.25 & Ideal per-thread CPI = 4
- **Answer 5**: Ideal CPI = 4 & Ideal per-thread CPI = 0.25