Multiprocessors and Multithreading

Jason Mars

Parallel Architectures for Executing Multiple Threads

Parallel Architectures for Executing Multiple Threads

- Multiprocessor – multiple CPUs tightly coupled enough to cooperate on a single problem.

Parallel Architectures for Executing Multiple Threads

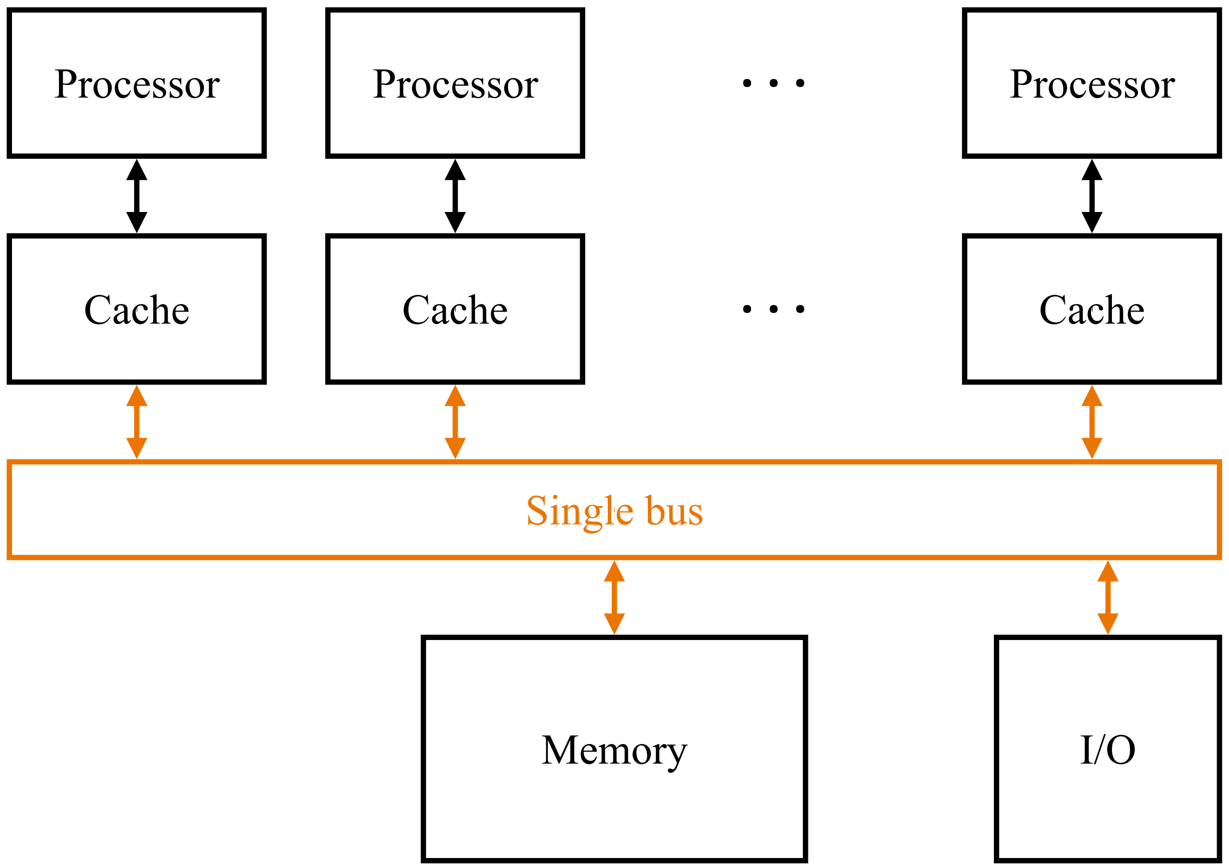
- Multiprocessor – multiple CPUs tightly coupled enough to cooperate on a single problem.
- Multithreaded processors (e.g., simultaneous multithreading) – single CPU core that can execute multiple threads simultaneously.

Parallel Architectures for Executing Multiple Threads

- Multiprocessor – multiple CPUs tightly coupled enough to cooperate on a single problem.
- Multithreaded processors (e.g., simultaneous multithreading) – single CPU core that can execute multiple threads simultaneously.
- Multicore processors – multiprocessor where the CPU cores coexist on a single processor chip.

Multiprocessors

- Not that long ago, multiprocessors were expensive, exotic machines – special-purpose engines to solve hard problems.
- Now they are pervasive.



Classifying Multiprocessors

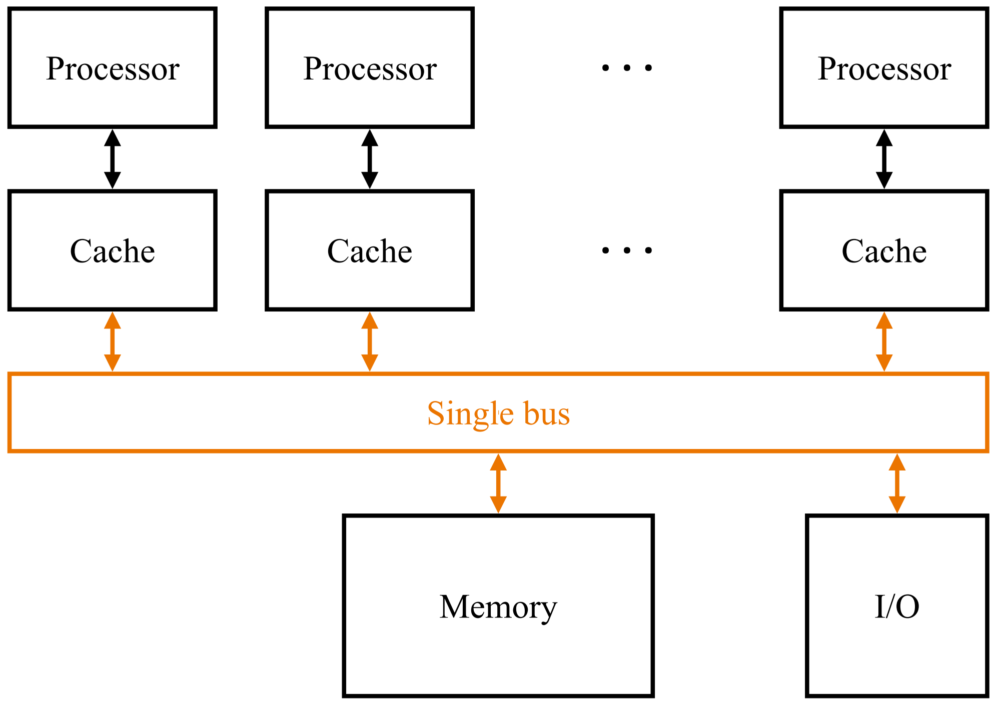
- Flynn Taxonomy
- Interconnection Network
- Memory Topology
- Programming Model

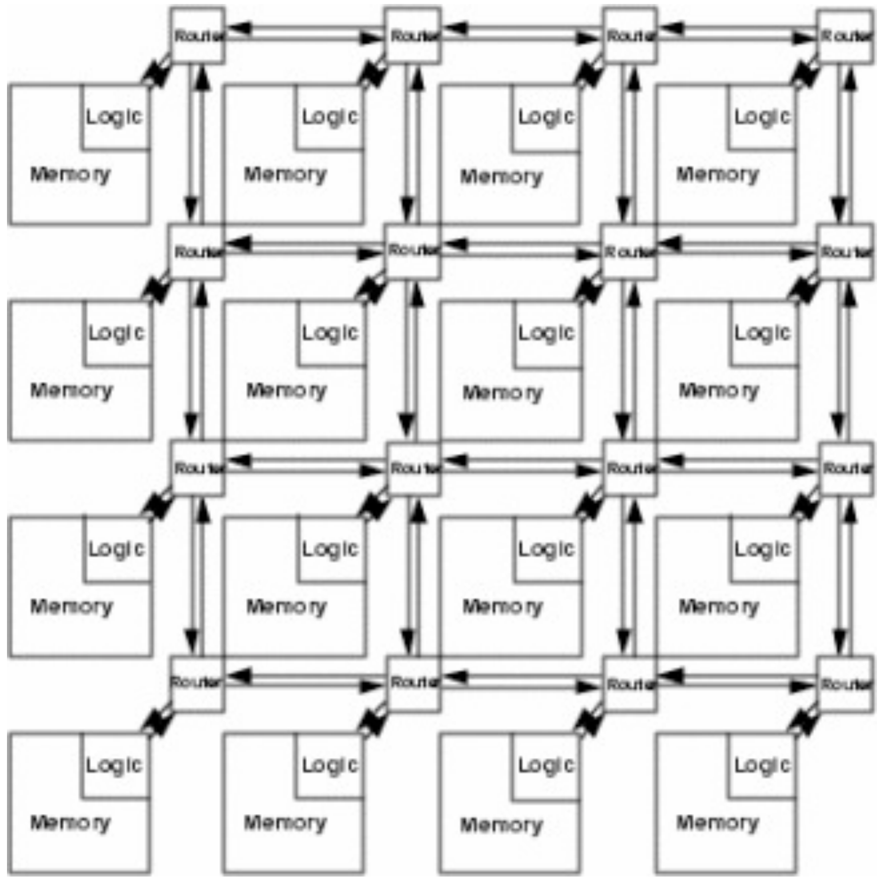
Flynn Taxonomy

- SISD (Single Instruction Single Data)
  - Uniprocessors
- SIMD (Single Instruction Multiple Data)
  - Examples: Illiac-IV, CM-2, Nvidia GPUs, etc.
    - Simple programming model
    - Low overhead
- MIMD (Multiple Instruction Multiple Data)
  - Examples: many, nearly all modern multiprocessors or multicores
    - Flexible
    - Use off-the-shelf microprocessors or microprocessor cores
- MISD (Multiple Instruction Single Data)
  - ???

Interconnection Networks

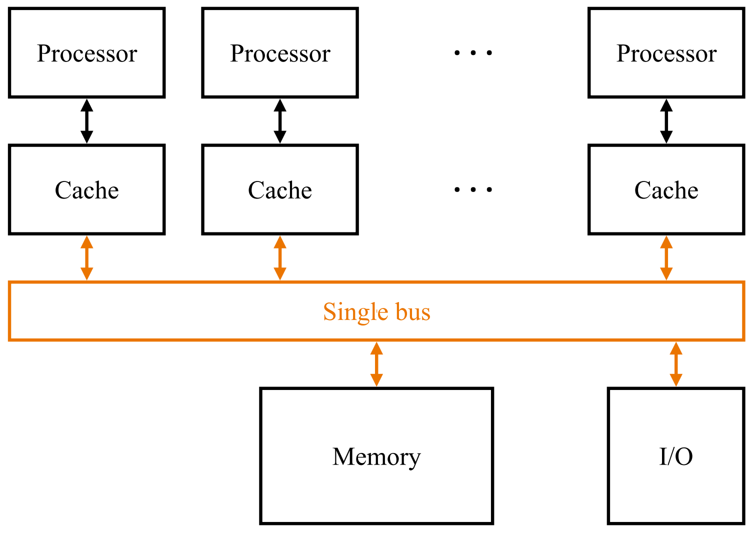
- Bus
- Network
- pros/cons?

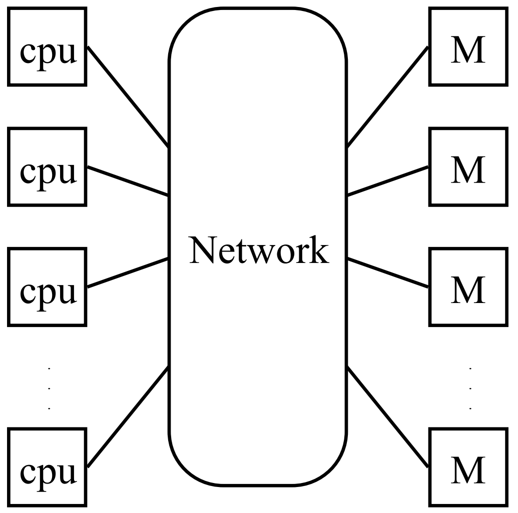


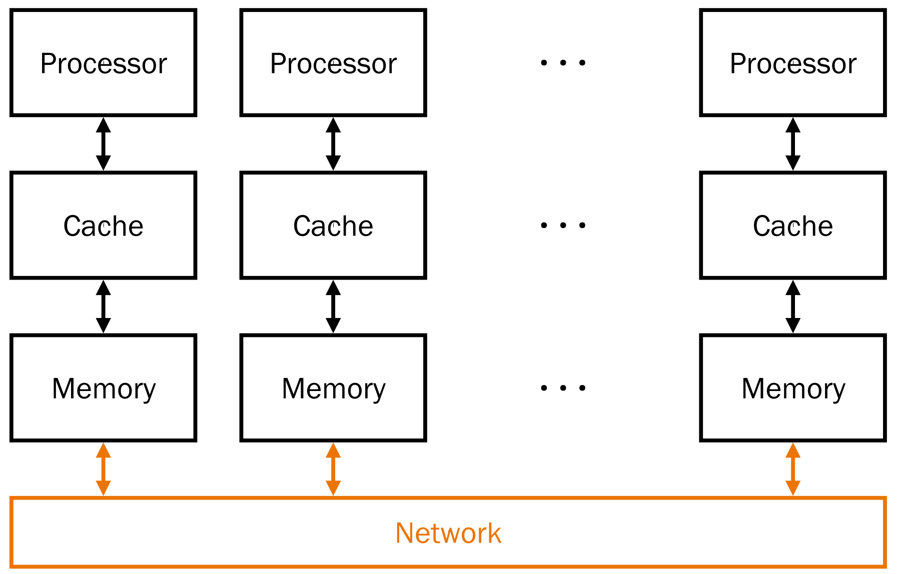


Memory Topology

- UMA (Uniform Memory Access)
- NUMA (Non-uniform Memory Access)
- pros/cons?

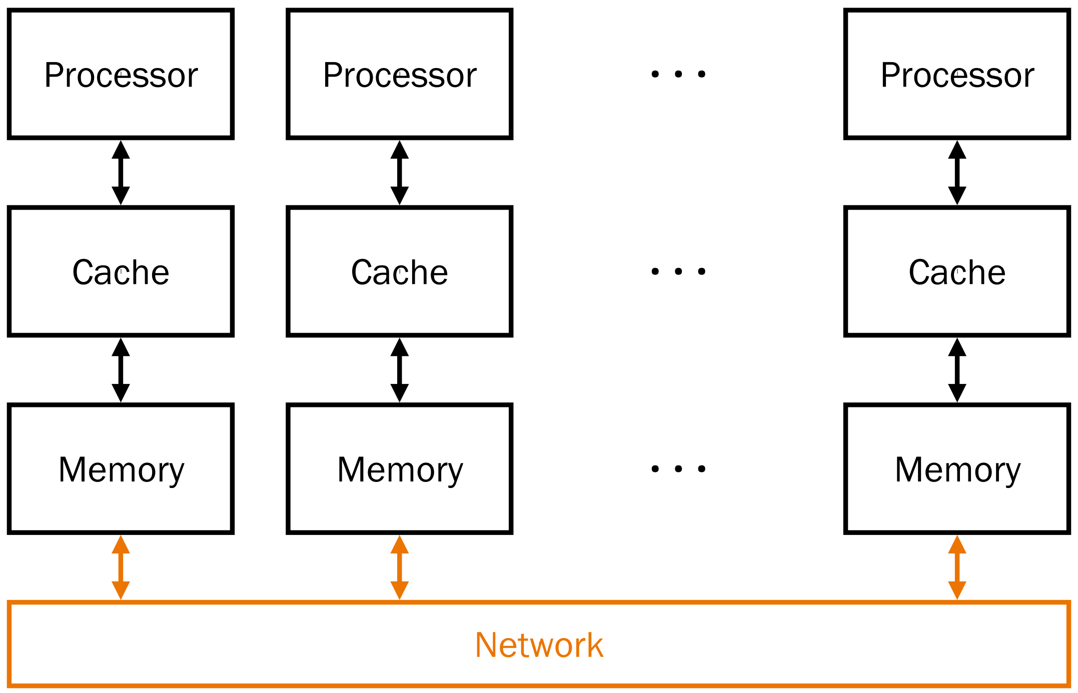






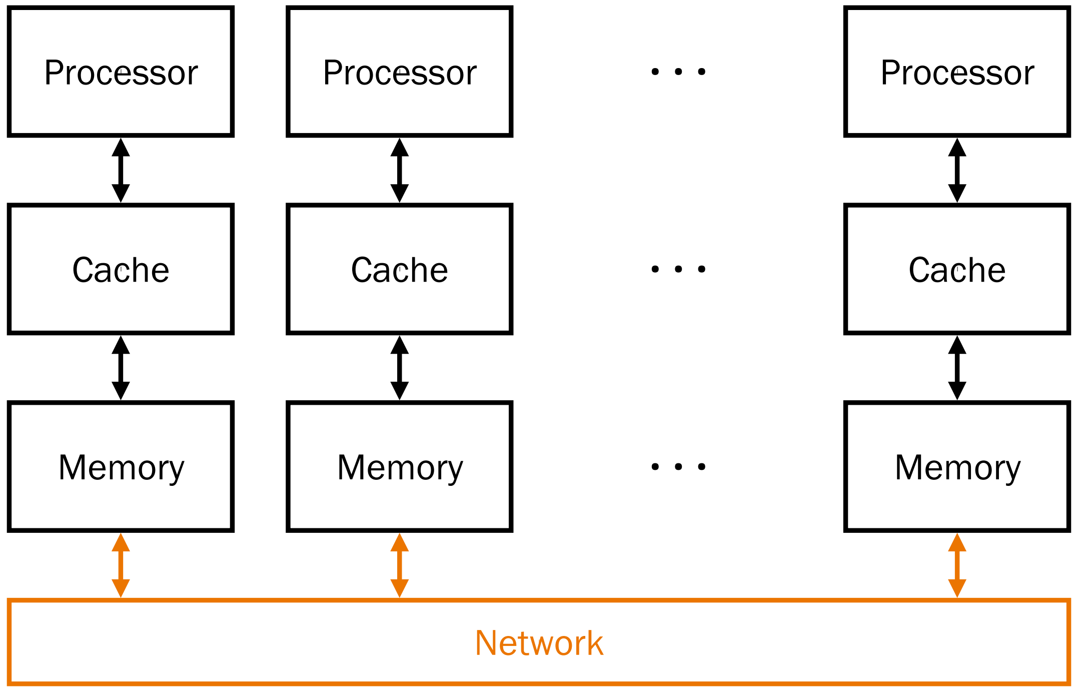
Programming Model

- Shared Memory -- every processor can name every address location
- Message Passing -- each processor can name only it’s local memory. Communication is through explicit messages.
- pros/cons?



Programming Model

- Shared Memory -- every processor can name every address location
- Message Passing -- each processor can name only it’s local memory. Communication is through explicit messages.
- pros/cons? *find the max of 100,000 integers on 10 processors.*



Parallel Programming

i = 47

Processor A

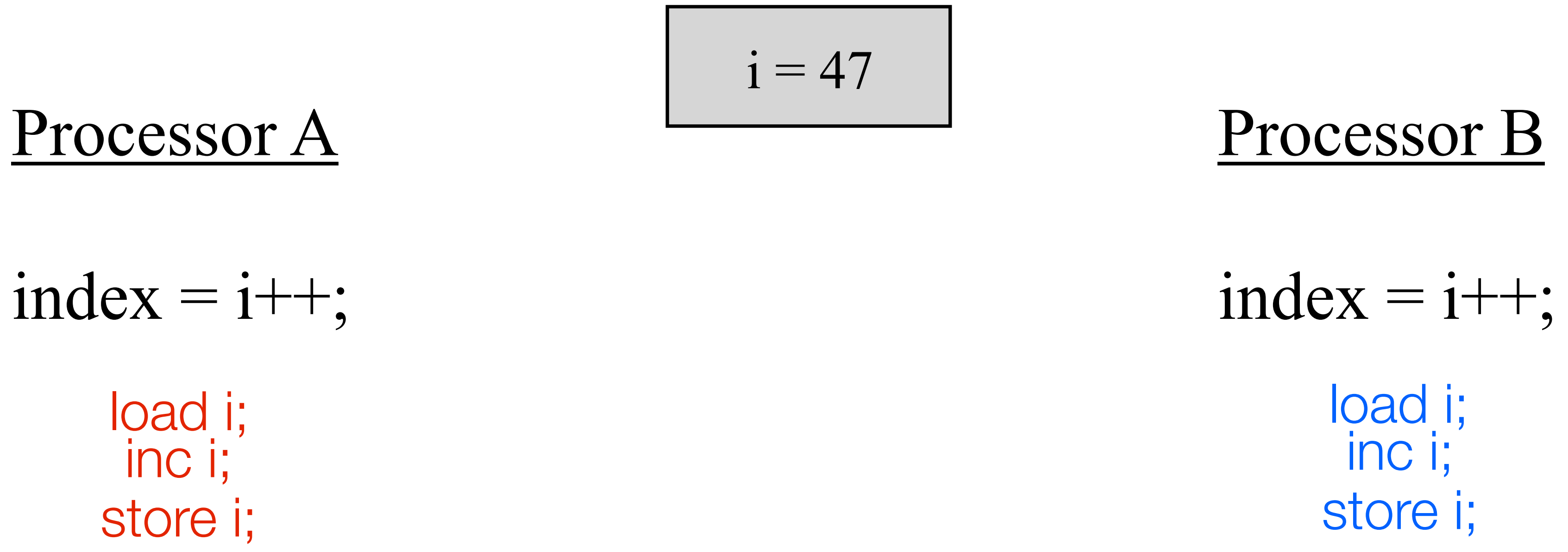
```
index = i++;
```

Processor B

```
index = i++;
```

- Shared-memory programming requires synchronization to provide mutual exclusion and prevent race conditions
  - locks (semaphores)
  - barriers

Parallel Programming



- Shared-memory programming requires synchronization to provide mutual exclusion and prevent race conditions
  - locks (semaphores)
  - barriers

Parallel Programming

i = 47

<u>Processor A</u>

index = i++;

<u>Processor B</u>

index = i++;

```
load i;
inc i;
store i;
load i;
inc i;
store i;
```

- Shared-memory programming requires synchronization to provide mutual exclusion and prevent race conditions
    - locks (semaphores)
    - barriers

Parallel Programming

i = 47

Processor A

```
index = i++;
```

Processor B

```
index = i++;
```

- Shared-memory programming requires synchronization to provide mutual exclusion and prevent race conditions
  - locks (semaphores)
  - barriers

Parallel Programming

$\boxed{i = 47}$

Processor A

```
index = i++;
```

Processor B

```
index = i++;
```

```
load i;
load i;
inc i;
inc i;
store i;
store i;
```

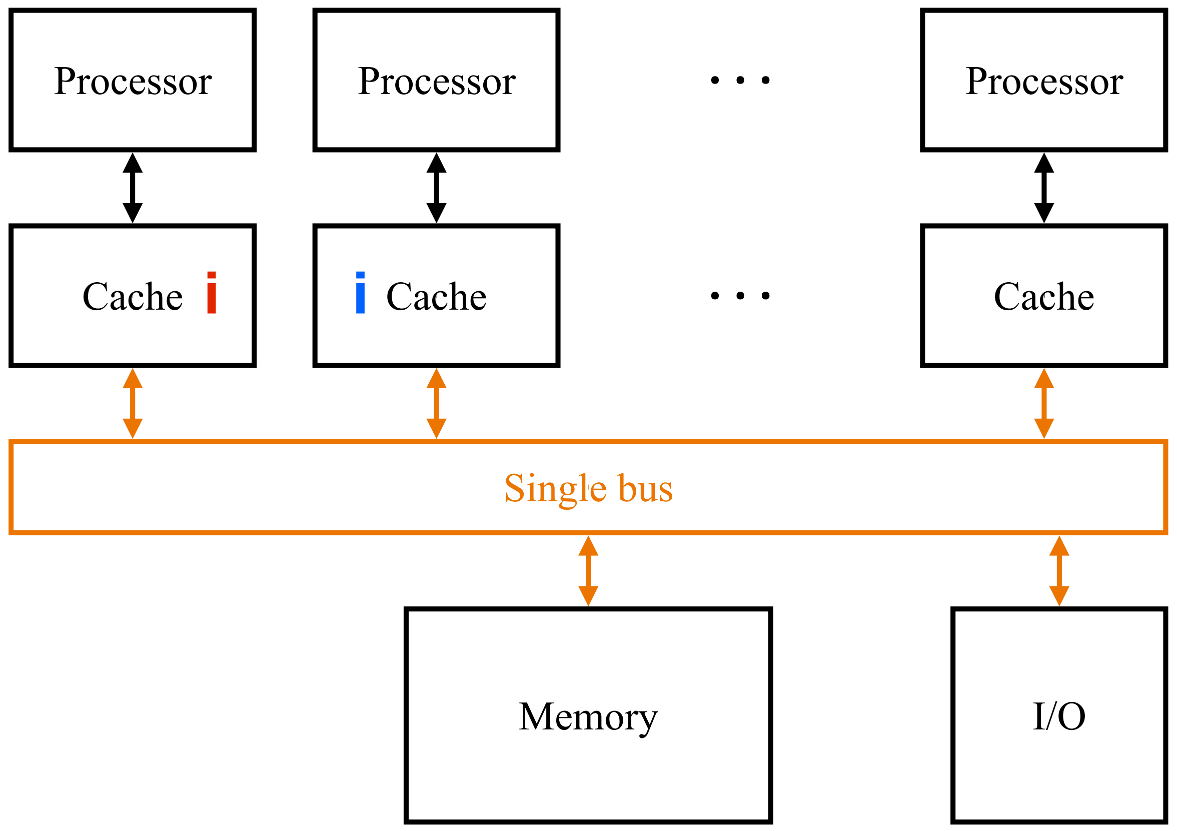
- Shared-memory programming requires synchronization to provide mutual exclusion and prevent race conditions
  - locks (semaphores)
  - barriers

But...

- That ignores the existence of caches
- How do caches complicate the problem of keeping data consistent between processors?

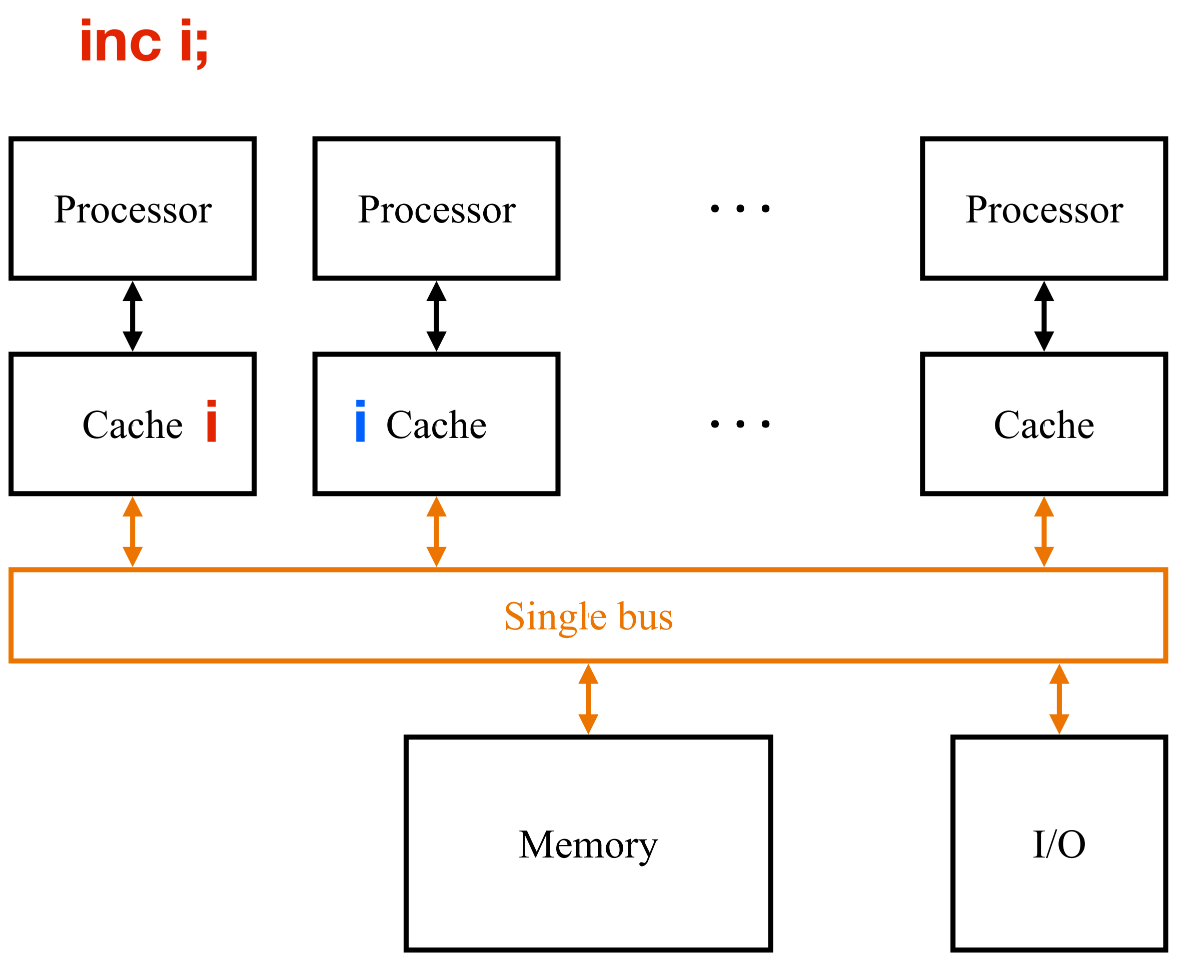
Multiprocessor Caches (Shared Memory)

- the problem -- cache coherency
- the solution?



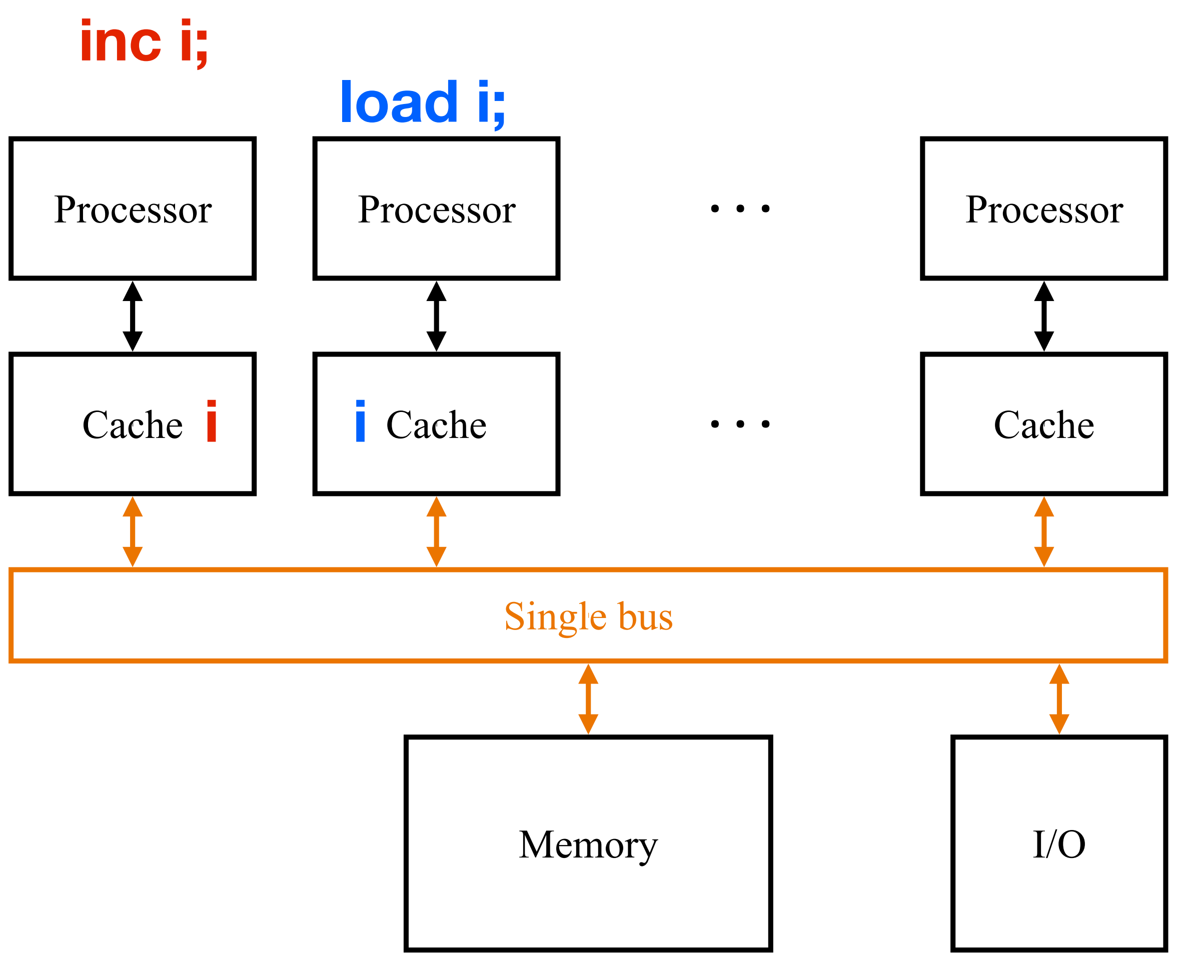
Multiprocessor Caches (Shared Memory)

- the problem -- cache coherency
- the solution?



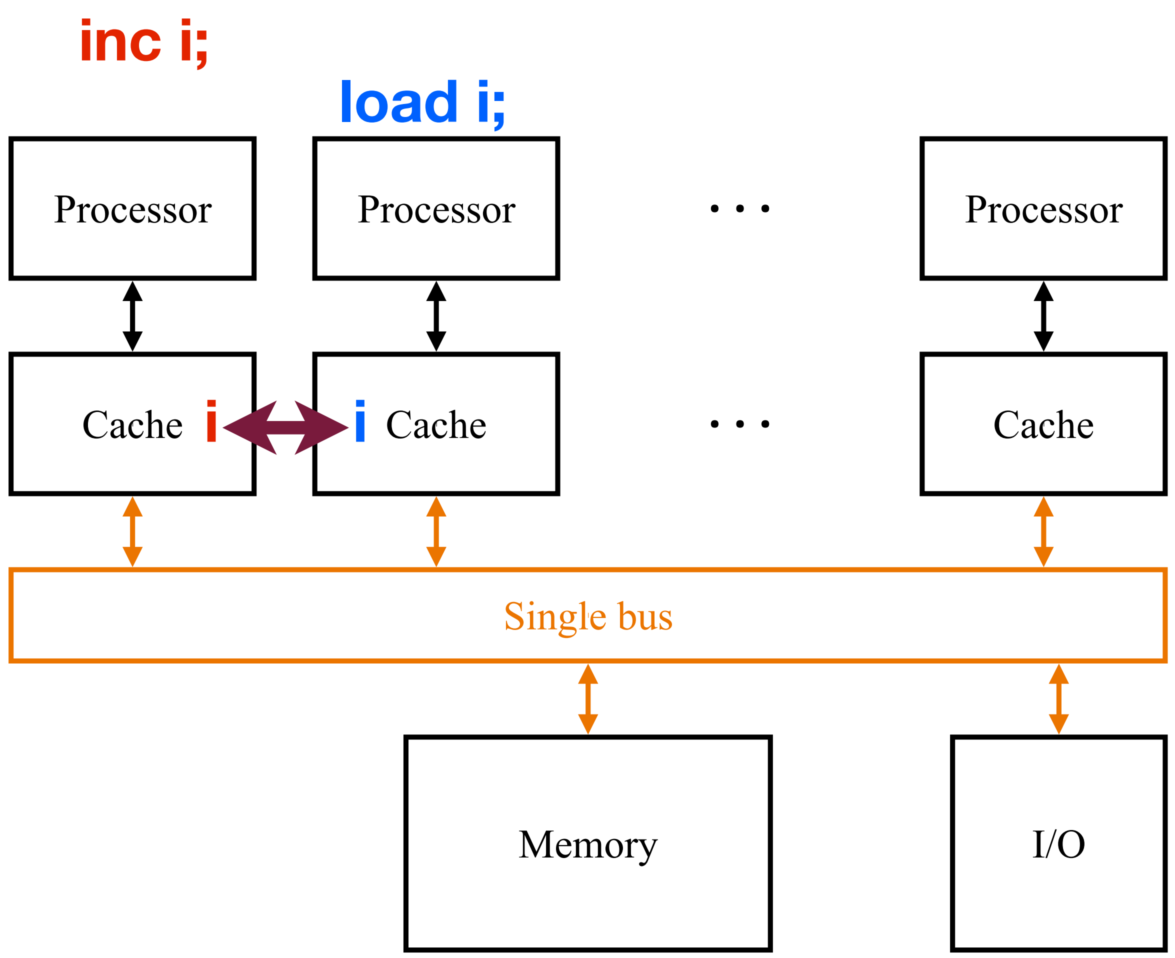
Multiprocessor Caches (Shared Memory)

- the problem -- cache coherency
- the solution?



Multiprocessor Caches (Shared Memory)

- the problem -- cache coherency
- the solution?



What Does Coherence Mean?

- Informally:
  - Any read must return the most recent write
  - Too strict and very difficult to implement
- Better:
  - A processor sees its own writes to a location in the correct order.
  - Any write must eventually be seen by a read
  - All writes are seen in order (“serialization”). Writes to the same location are seen in the same order by all processors.
- Without these guarantees, synchronization doesn’t work

Solutions

Solutions

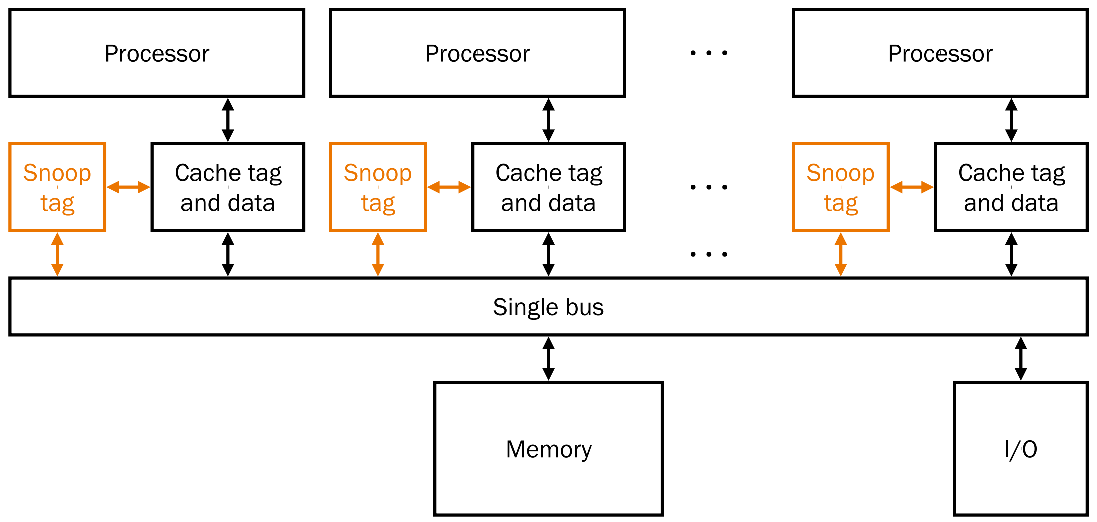
- Snooping Solution (Snoopy Bus):
    - Send all requests for unknown data to all processors
    - Processors snoop to see if they have a copy and respond accordingly
    - Requires “broadcast”, since caching information is at processors
    - Works well with bus (natural broadcast medium)
    - Dominates for small scale machines (most of the market)

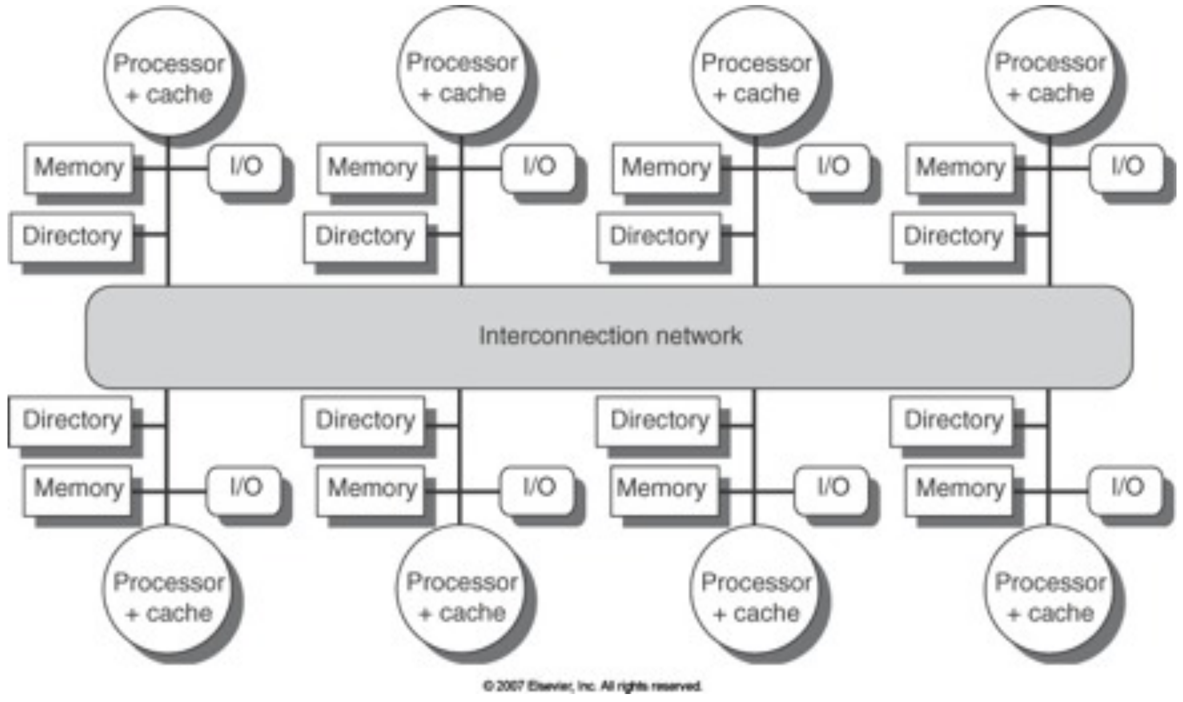
Solutions

- Snooping Solution (Snoopy Bus):
  - Send all requests for unknown data to all processors
  - Processors snoop to see if they have a copy and respond accordingly
  - Requires “broadcast”, since caching information is at processors
  - Works well with bus (natural broadcast medium)
  - Dominates for small scale machines (most of the market)
- Directory-Based Schemes
  - Keep track of what is being shared in one centralized place (for each address) => the directory
  - Distributed memory => distributed directory (avoids bottlenecks)
  - Send point-to-point requests to processors (to invalidate, etc.)
  - Scales better than Snooping for large multiprocessors

Implementing Coherence Protocols

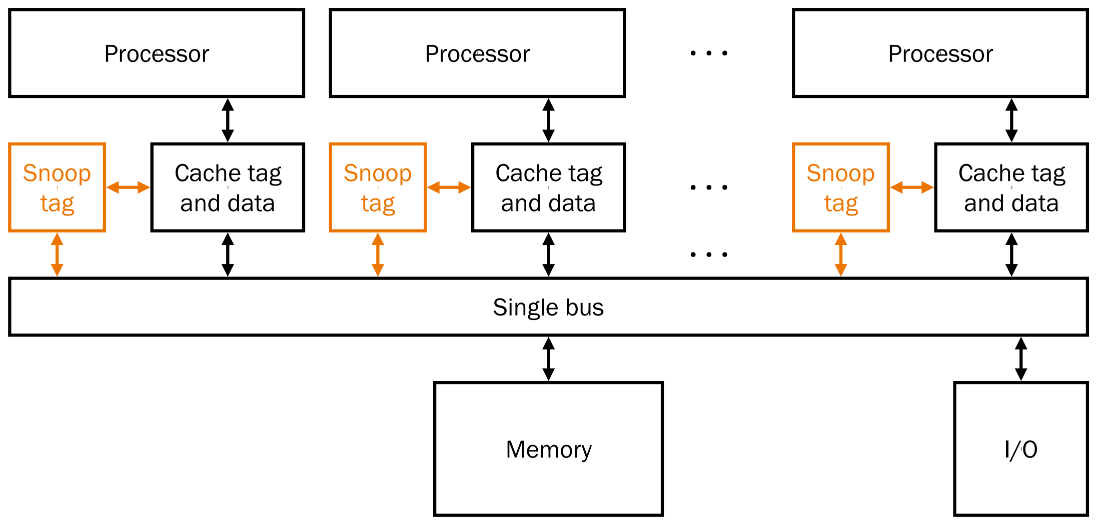
- How do you find the most up-to-date copy of the desired data?
- Snooping protocols
- Directory protocols

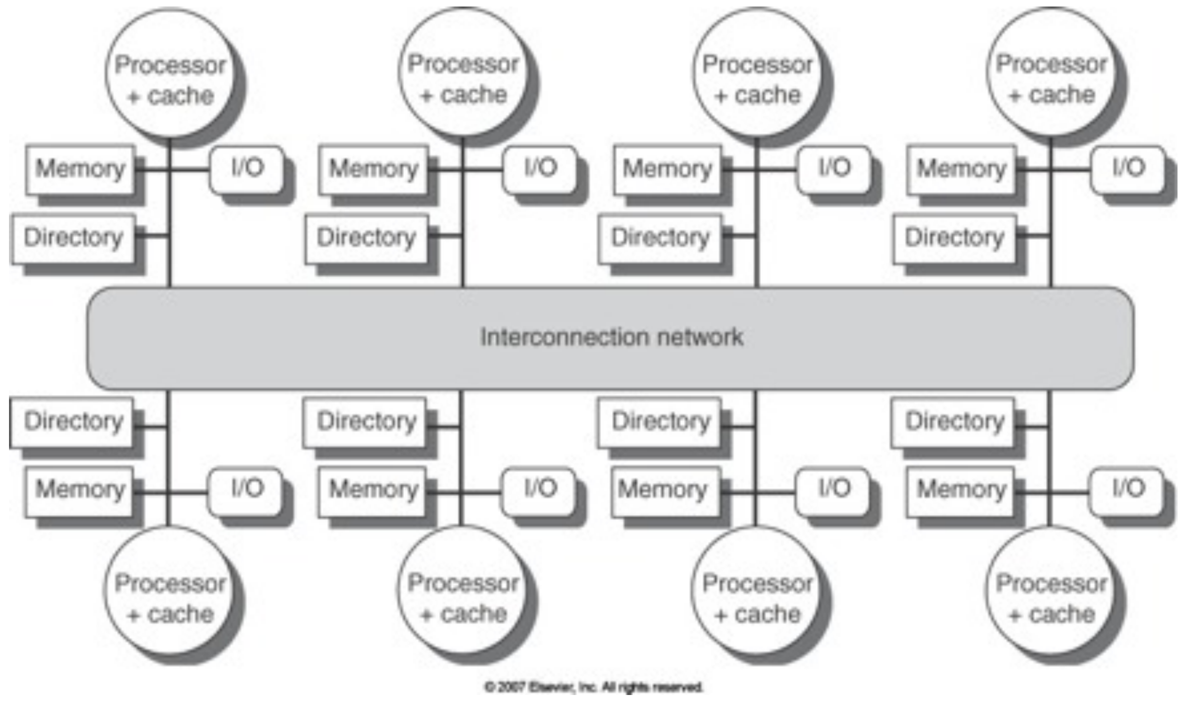




Implementing Coherence Protocols

- How do you find the most up-to-date copy of the desired data?
- Snooping protocols
- Directory protocols





Write-Update vs Write-Invalidate

Parallel Architectures for Executing Multiple Threads

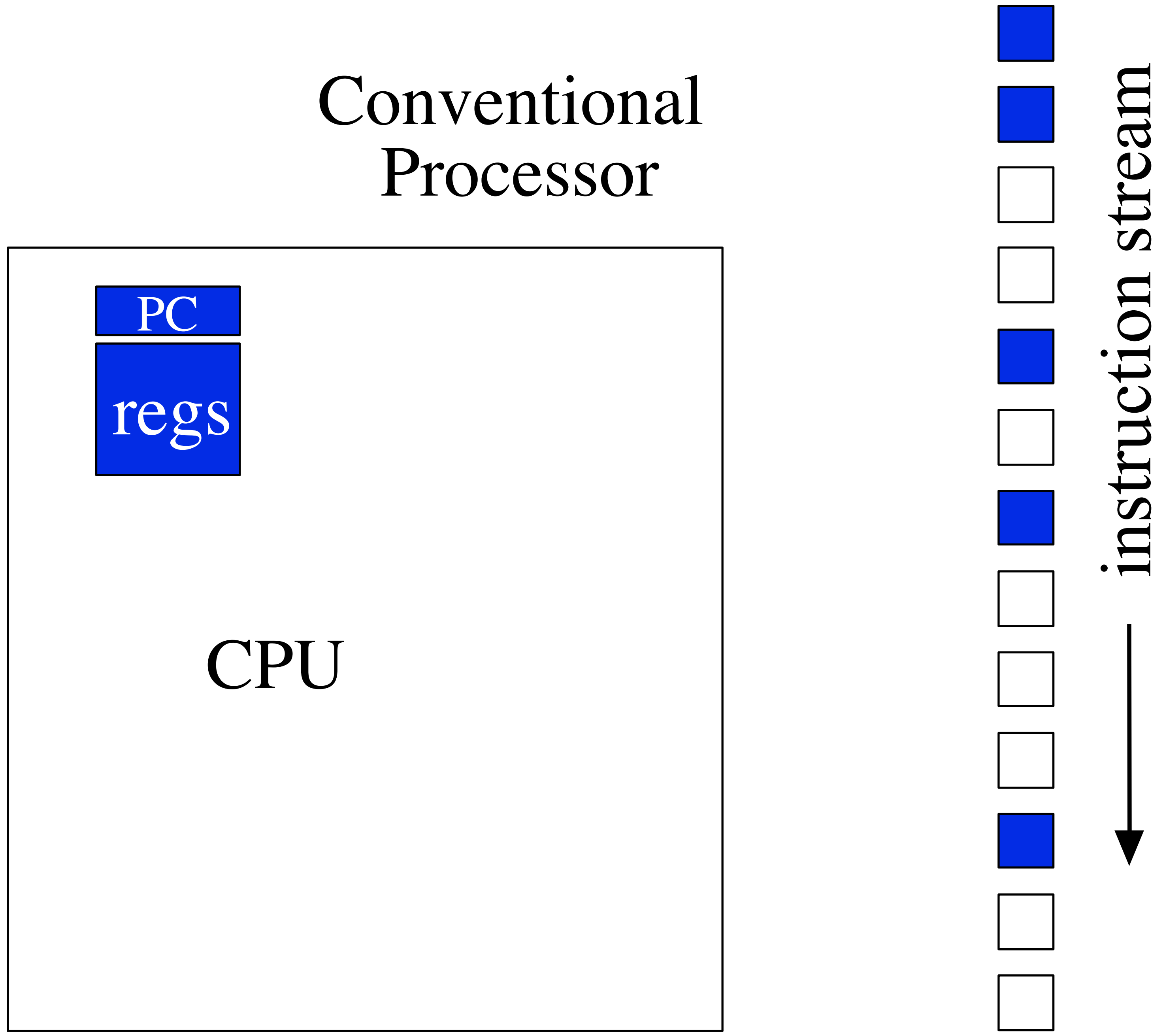
- Multiprocessor – multiple CPUs tightly coupled enough to cooperate on a single problem.
- Multithreaded processors (e.g., simultaneous multithreading) – single CPU core that can execute multiple threads simultaneously.
- Multicore processors – multiprocessor where the CPU cores coexist on a single processor chip.

Simultaneous Multithreading

(A Few of Dean Tullsen's 1996 Thesis Slides)

Dean Tullsen

Hardware Multithreading



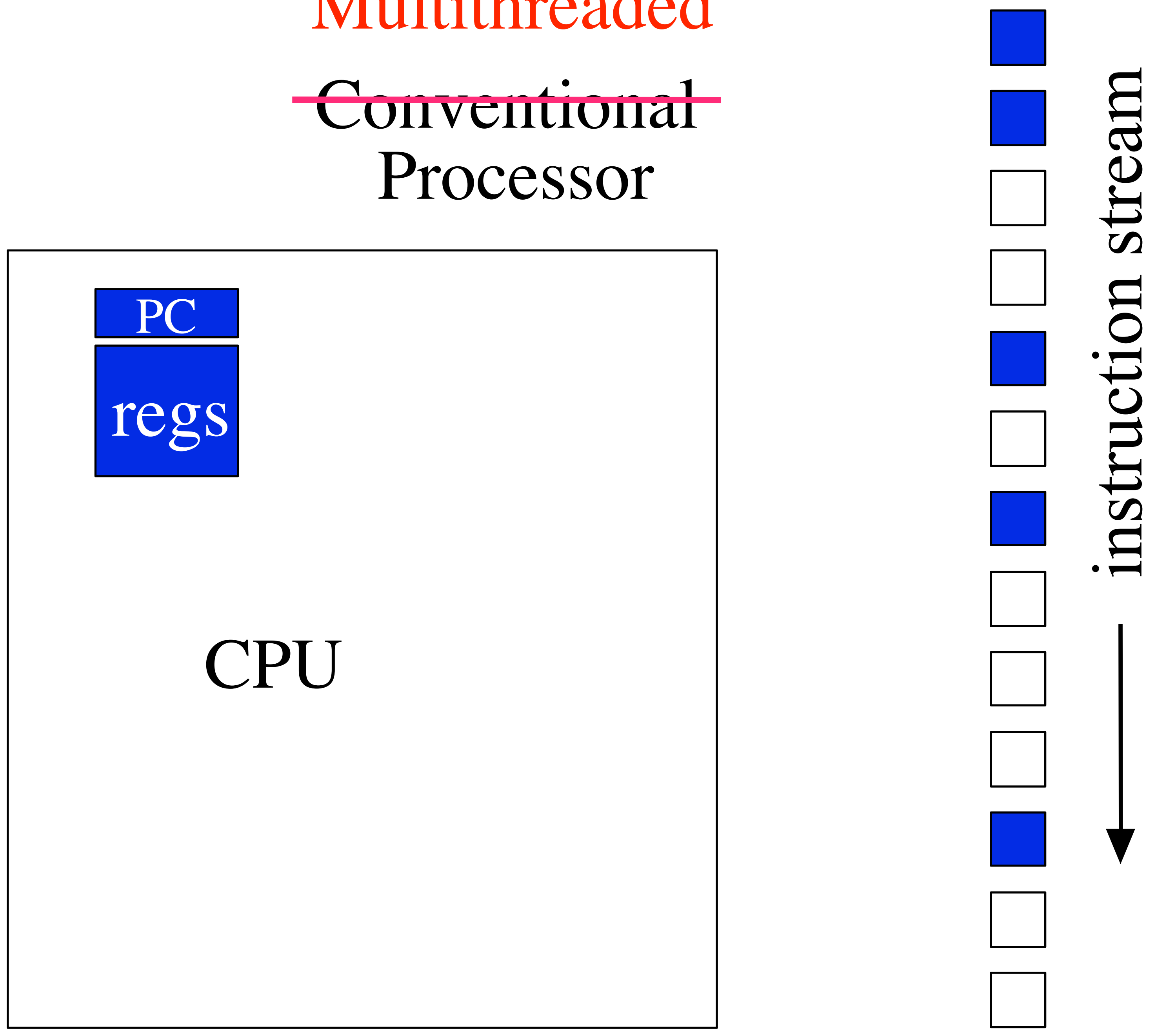
Dean Tullsen

Hardware Multithreading

Multithreaded

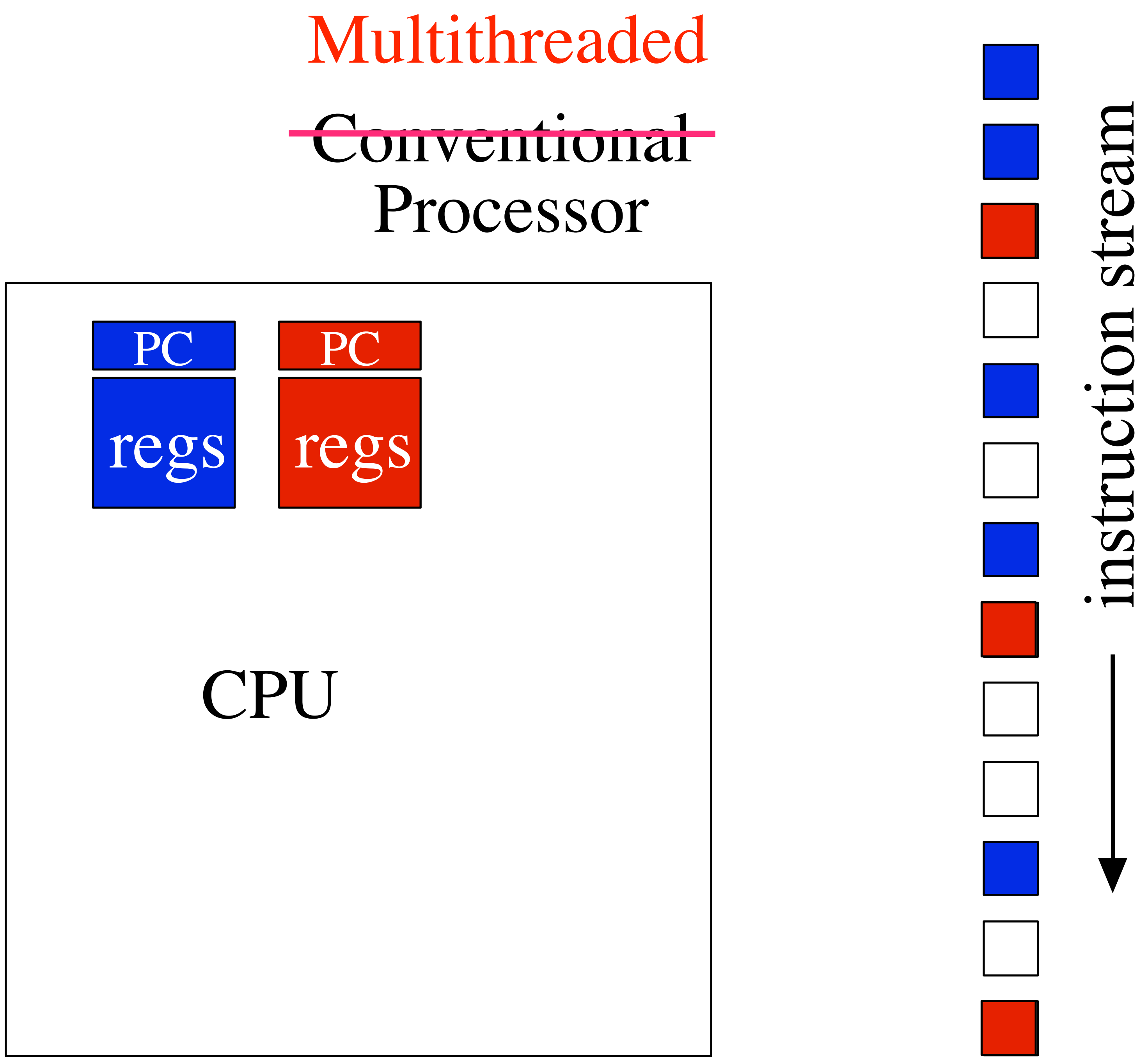
~~Conventional~~

Processor



Dean Tullsen

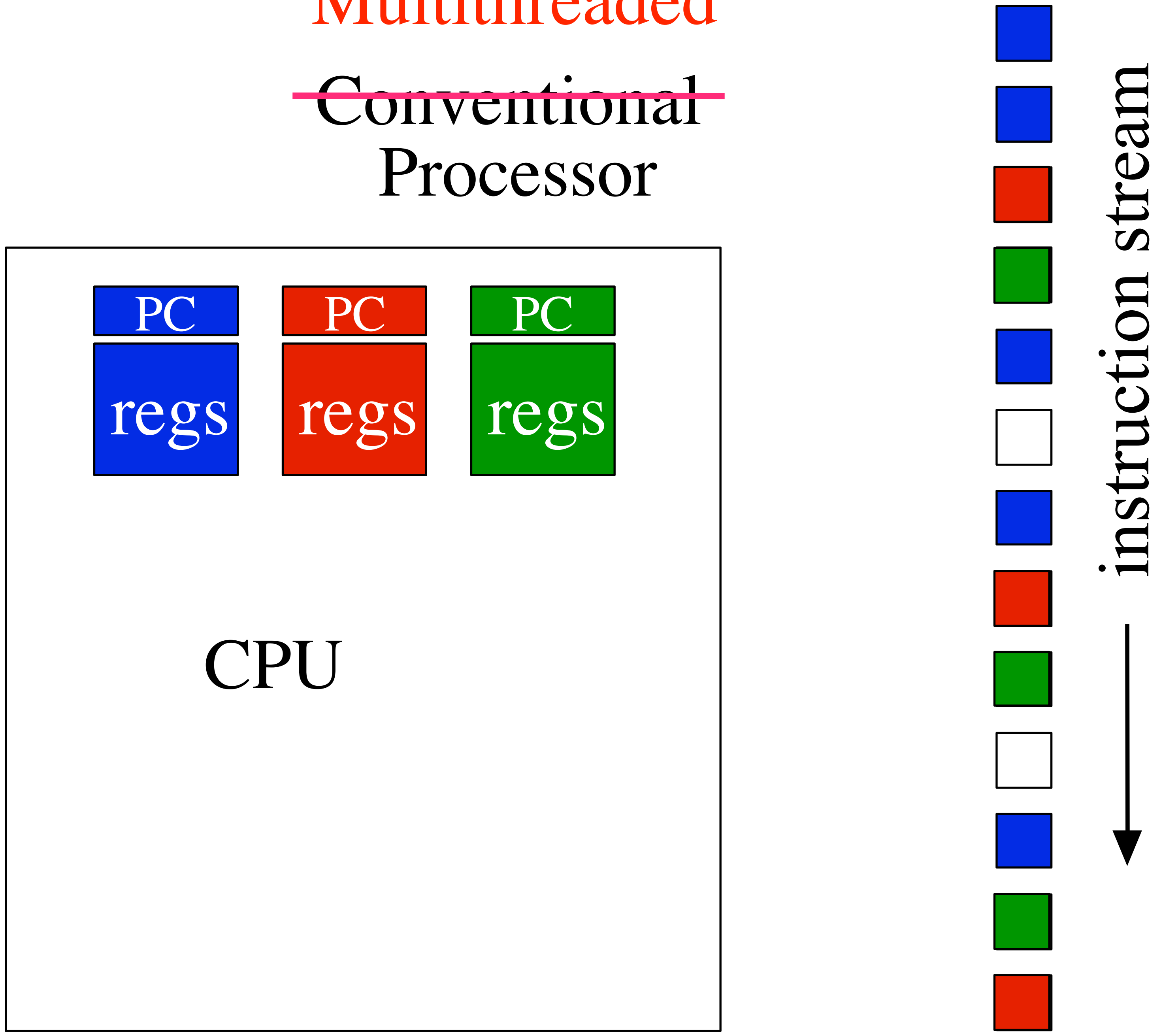
Hardware Multithreading



Dean Tullsen

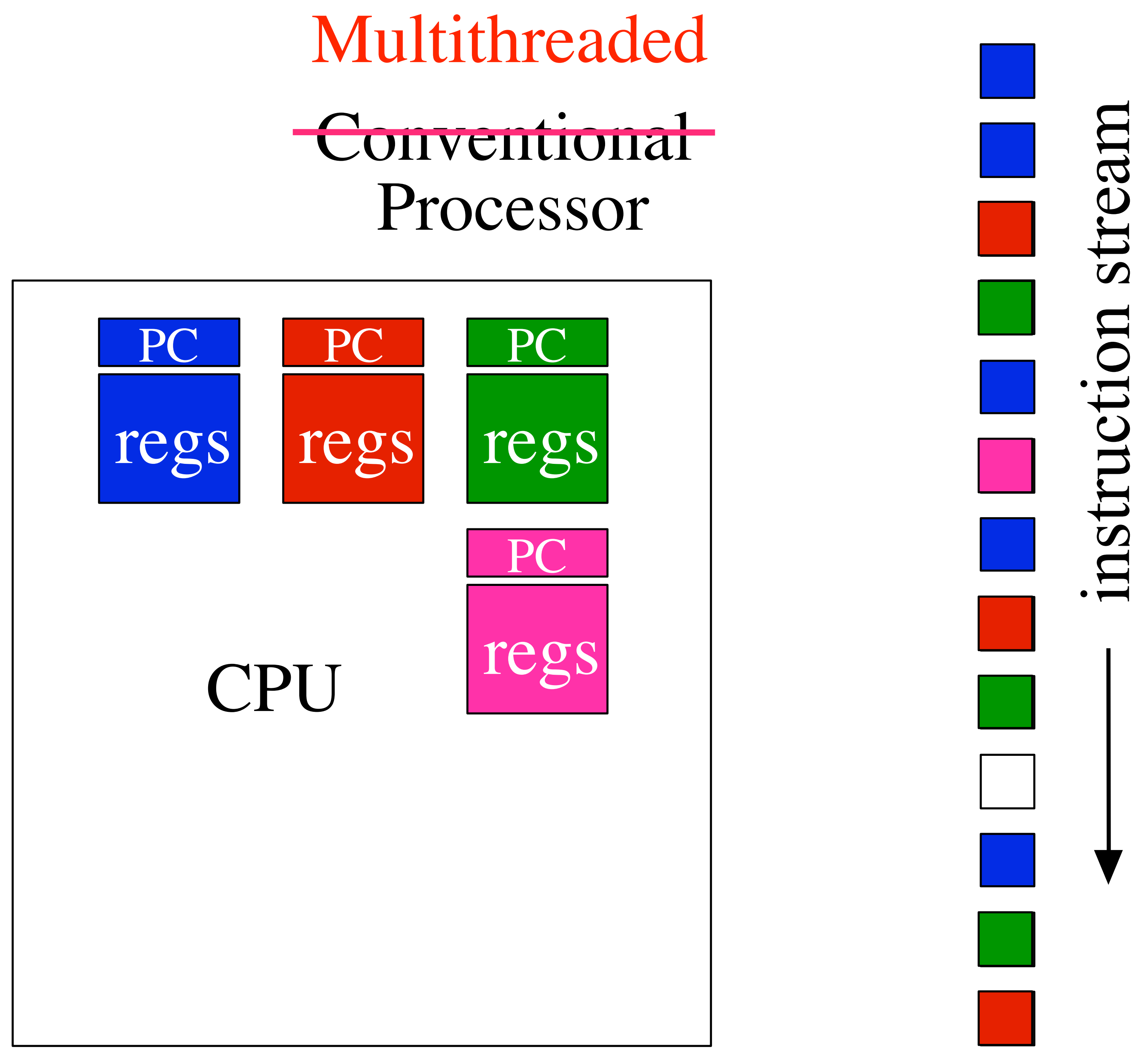
Hardware Multithreading

Multithreaded
~~Conventional~~
Processor



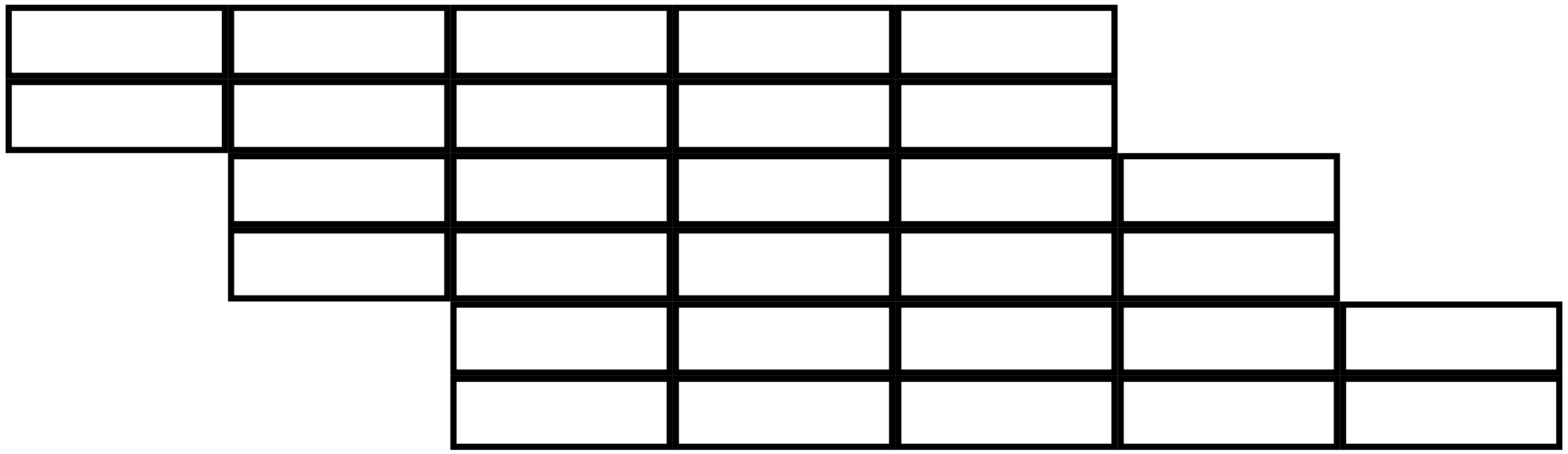
Dean Tullsen

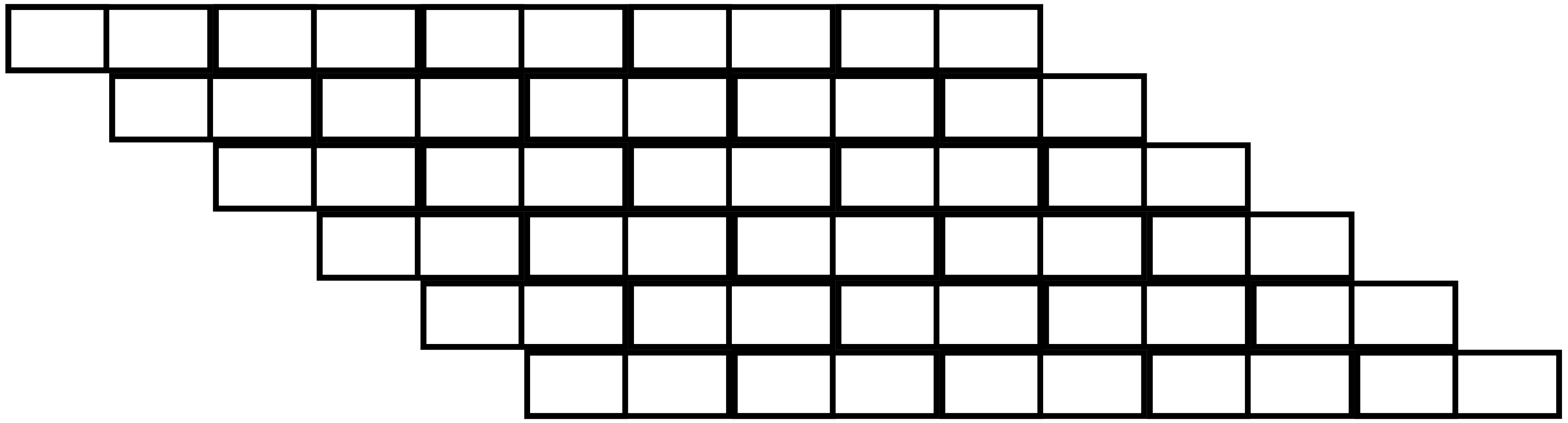
Hardware Multithreading



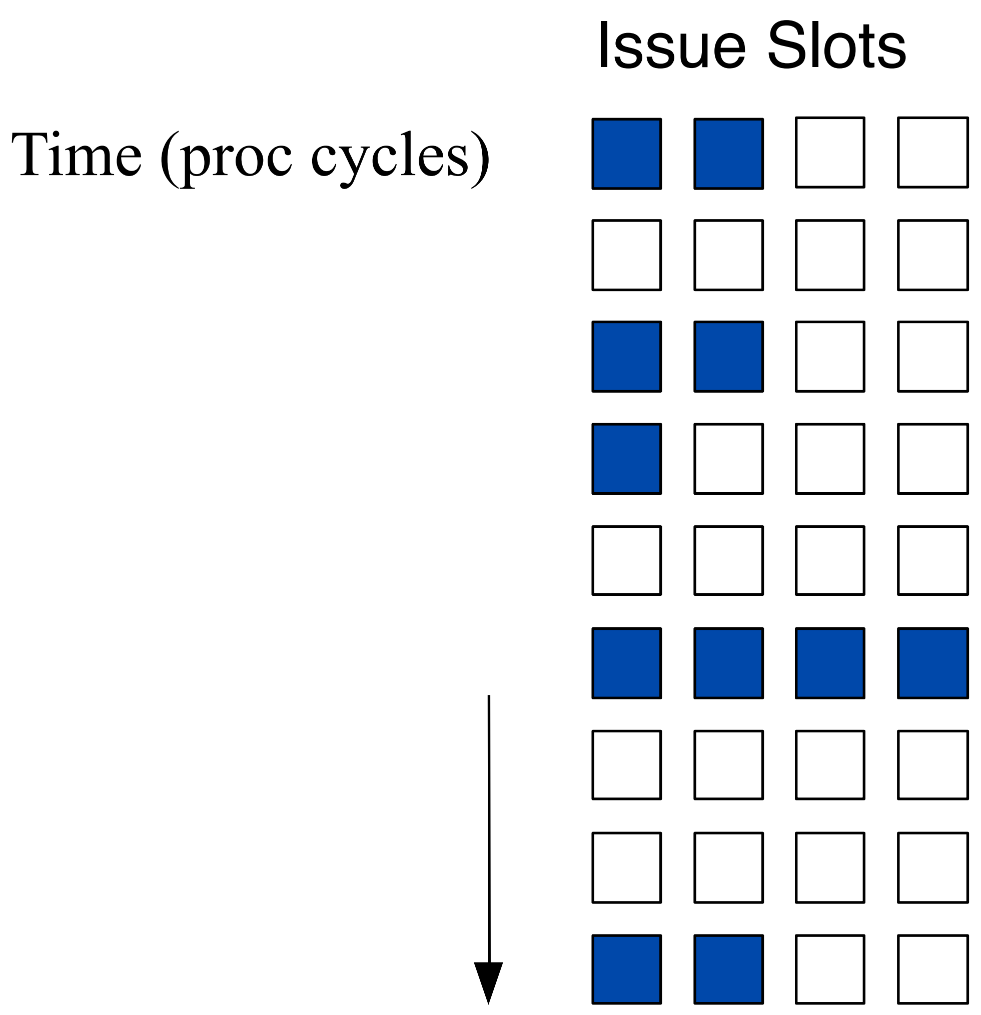
Dean Tullsen

Superscalar (vs Superpipelined)


(multiple instructions in the same stage, same CR as scalar)

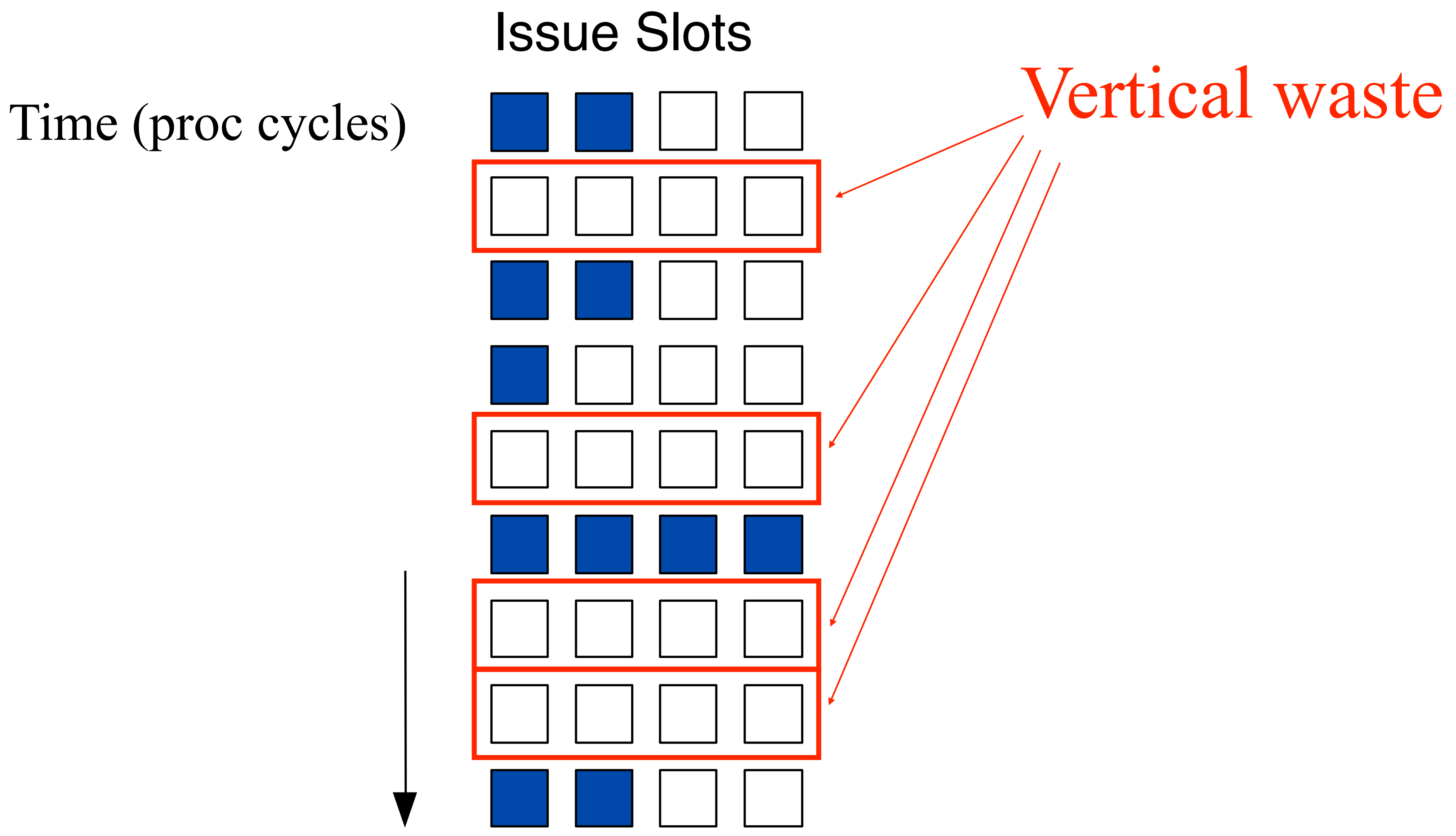

(more total stages, faster clock rate)

Superscalar Execution



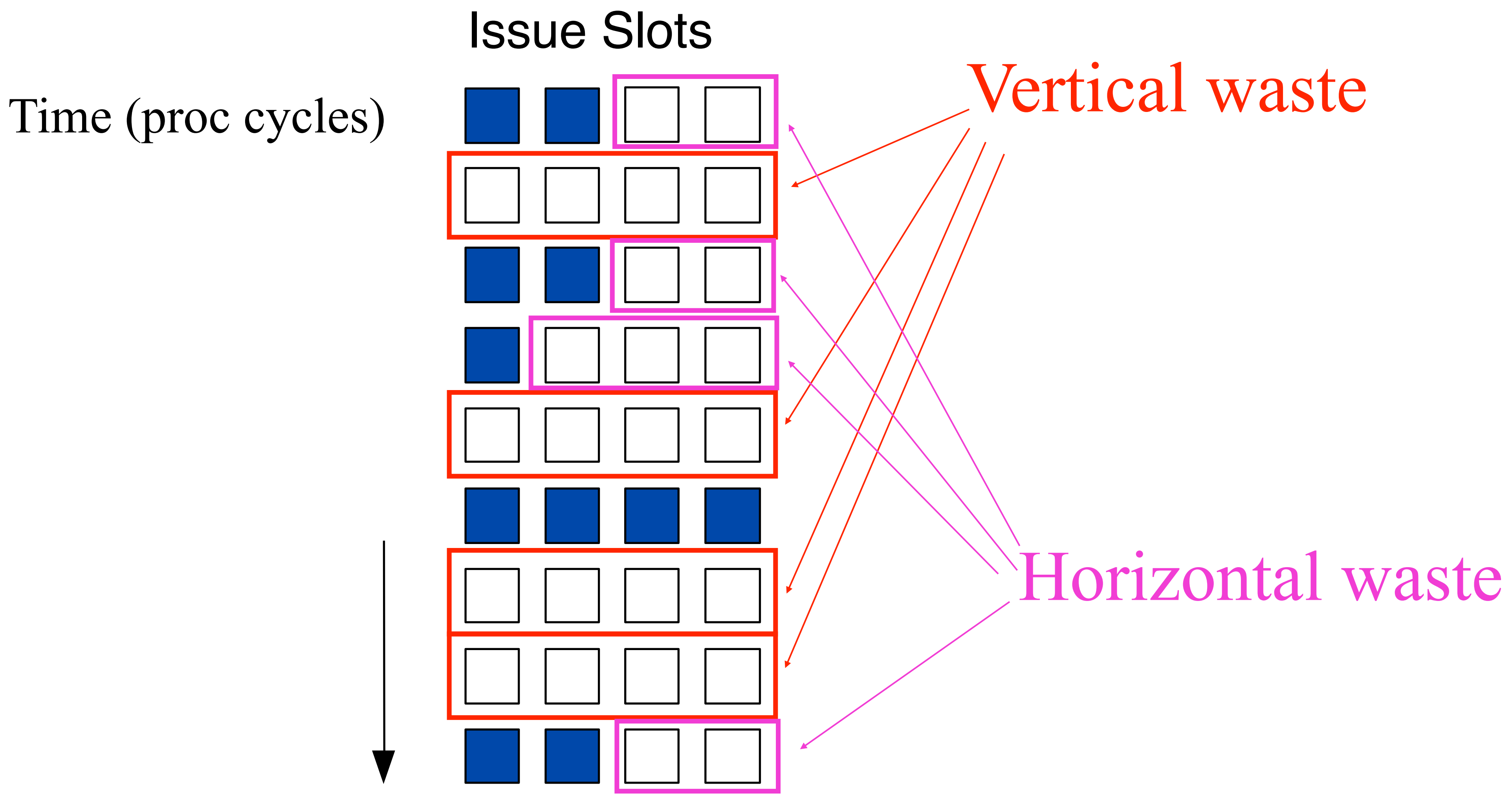
Dean Tullsen

Superscalar Execution



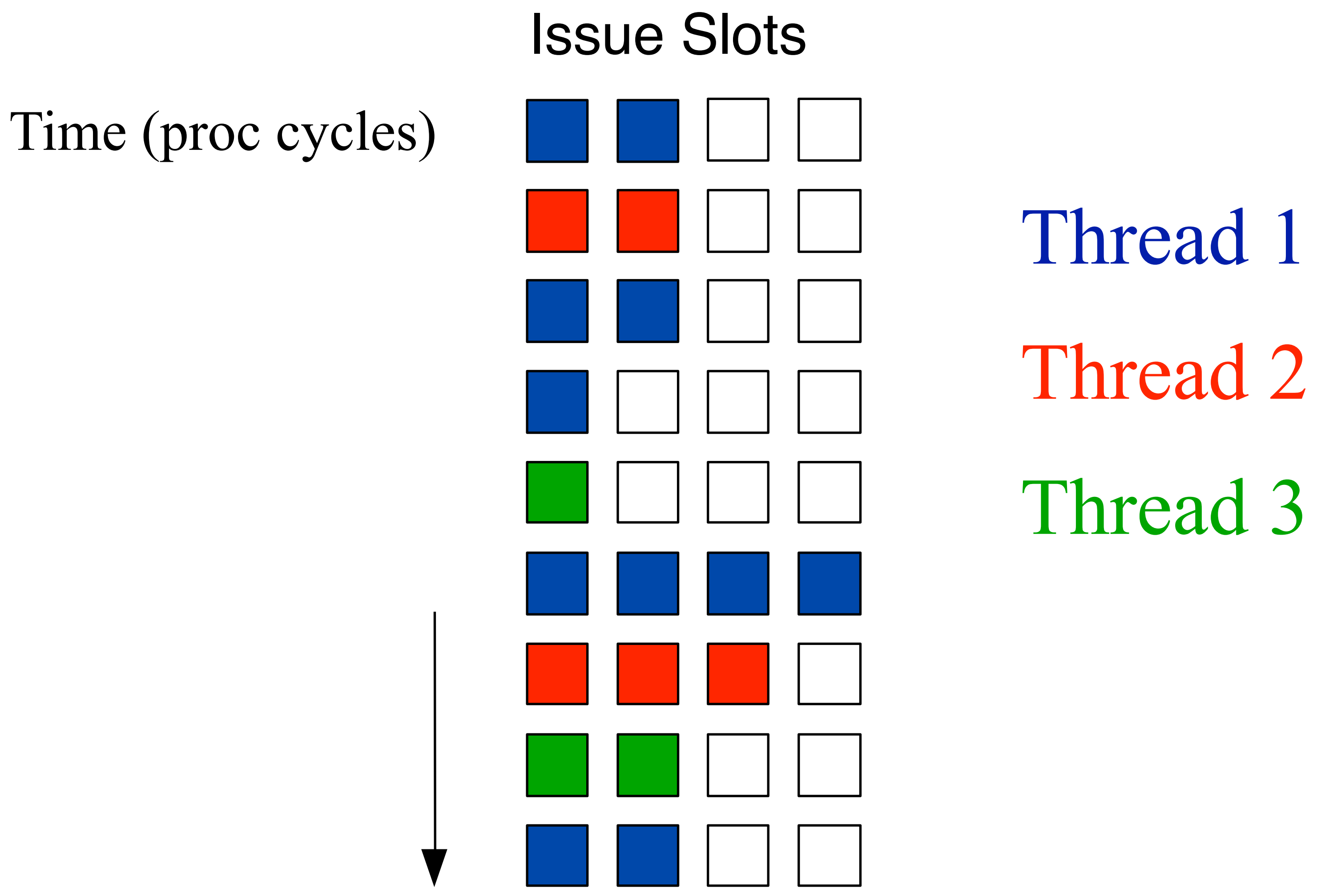
Dean Tullsen

Superscalar Execution



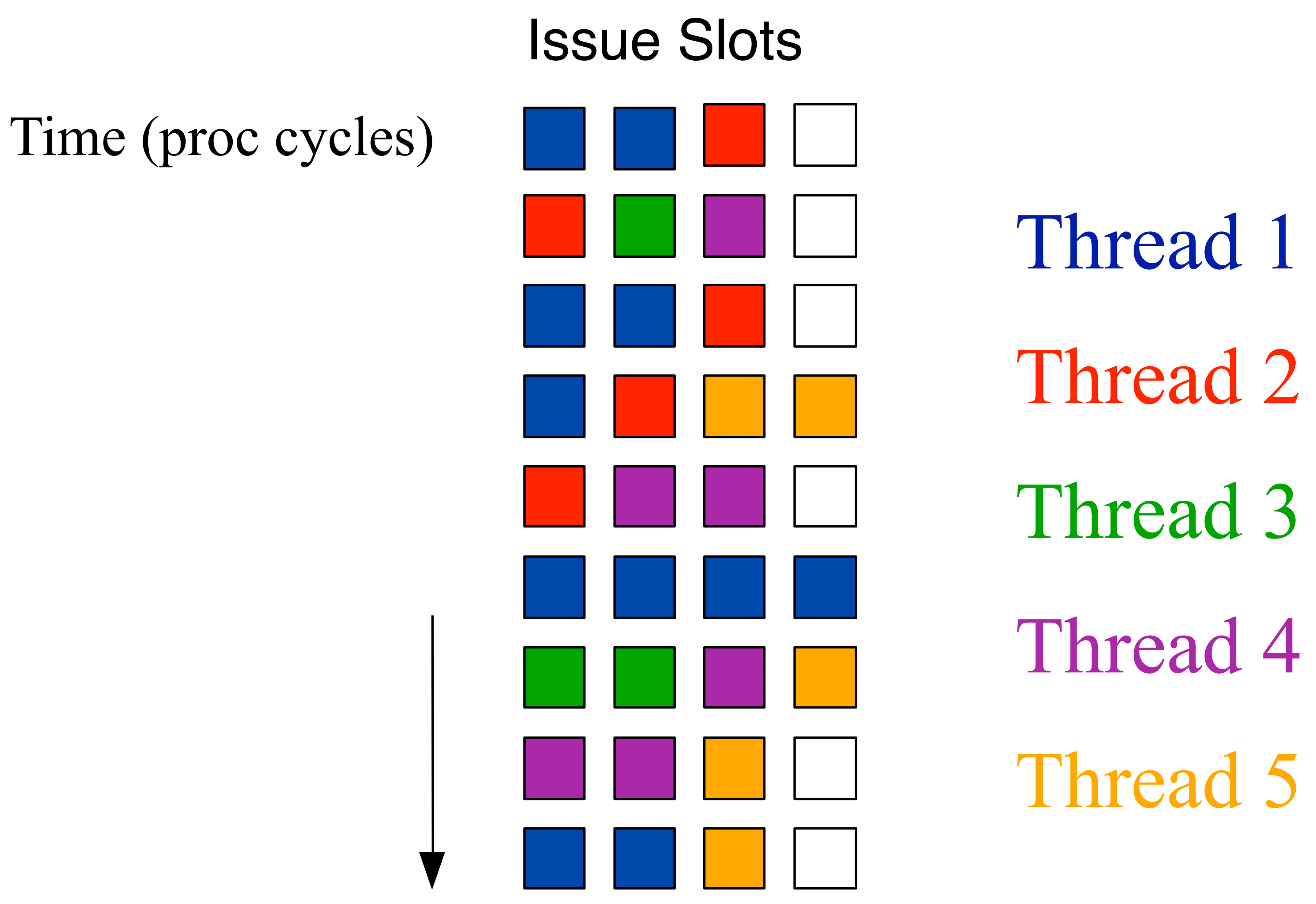
Dean Tullsen

Superscalar Execution with Fine-Grain Multithreading



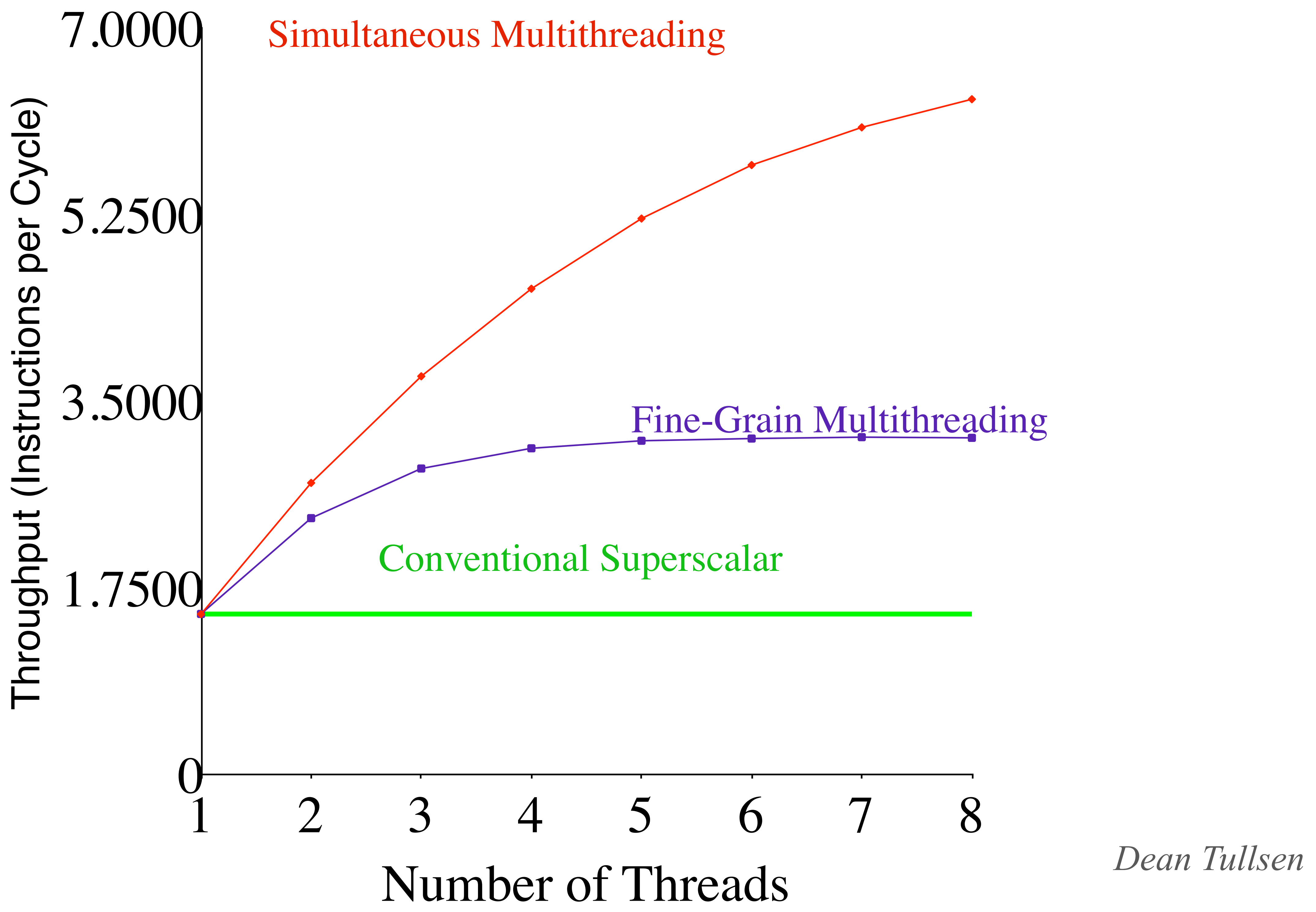
Dean Tullsen

Simultaneous Multithreading



Dean Tullsen

SMT Performance



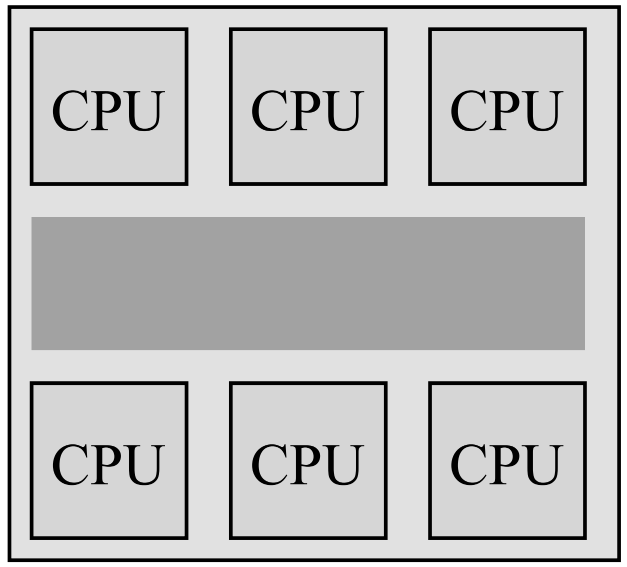
Parallel Architectures for Executing Multiple Threads

- Multiprocessor – multiple CPUs tightly coupled enough to cooperate on a single problem.

- Multithreaded processors (e.g., simultaneous multithreading) – single CPU core that can execute multiple threads simultaneously.

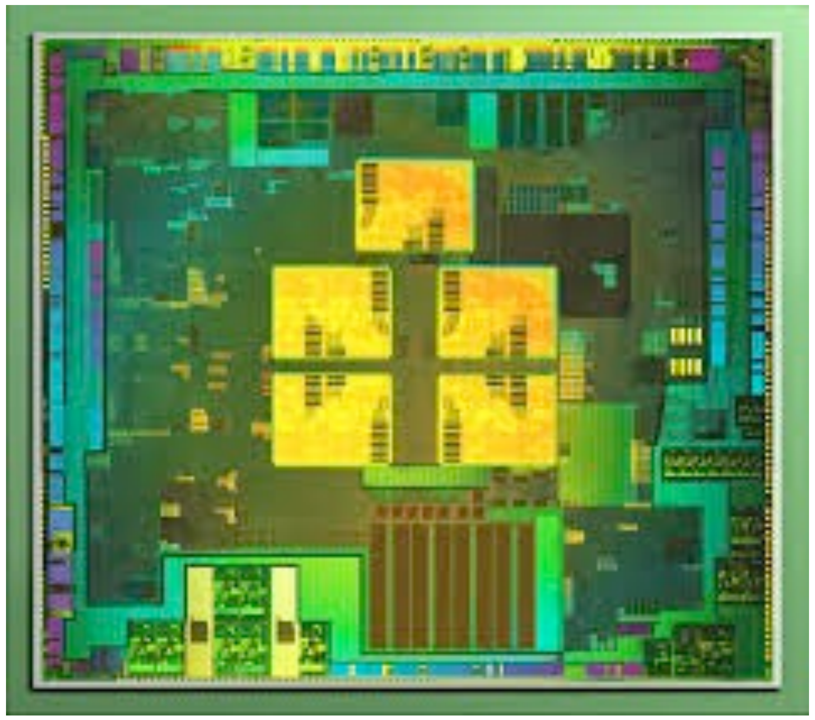
- Multicore processors – multiprocessor where the CPU cores coexist on a single processor chip.

Multicore Processors (aka Chip Multiprocessors)

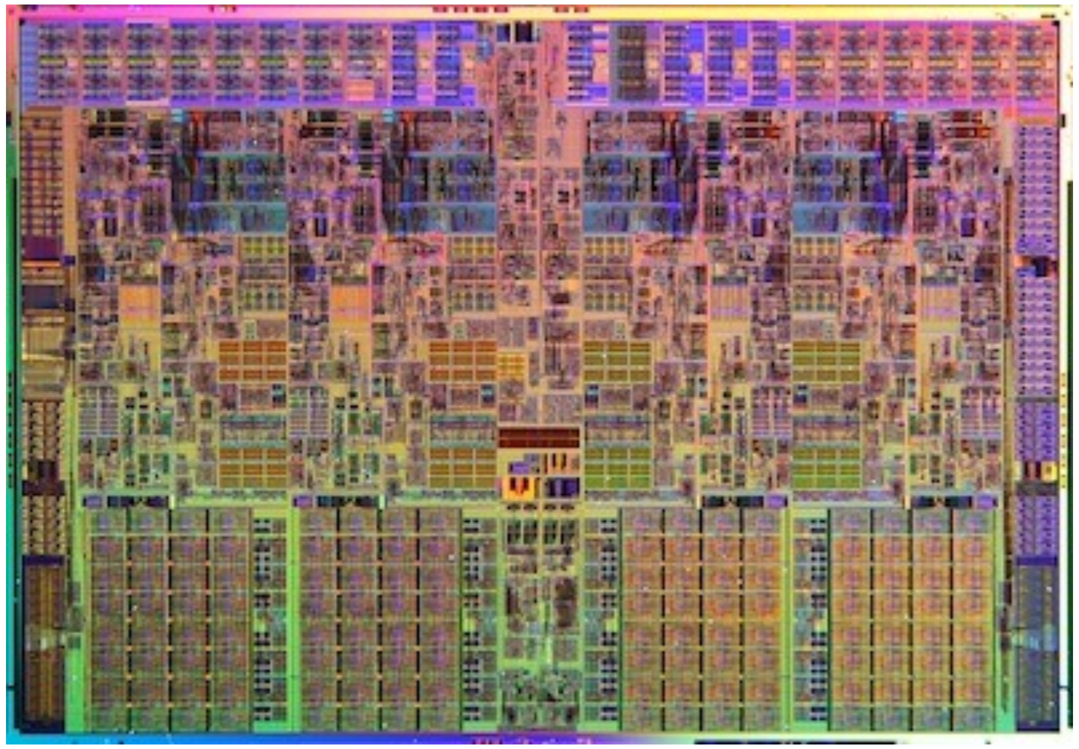


- Multiple cores on the same die, may or may not share L2 or L3 cache.
- Intel, AMD both have quad core processors. Sun Niagara T2 is 8 cores x 8 threads (64 contexts!)
- Everyone’s roadmap seems to be increasingly multi-core.

The Latest Processors


Tegra 3 (5 Cores)

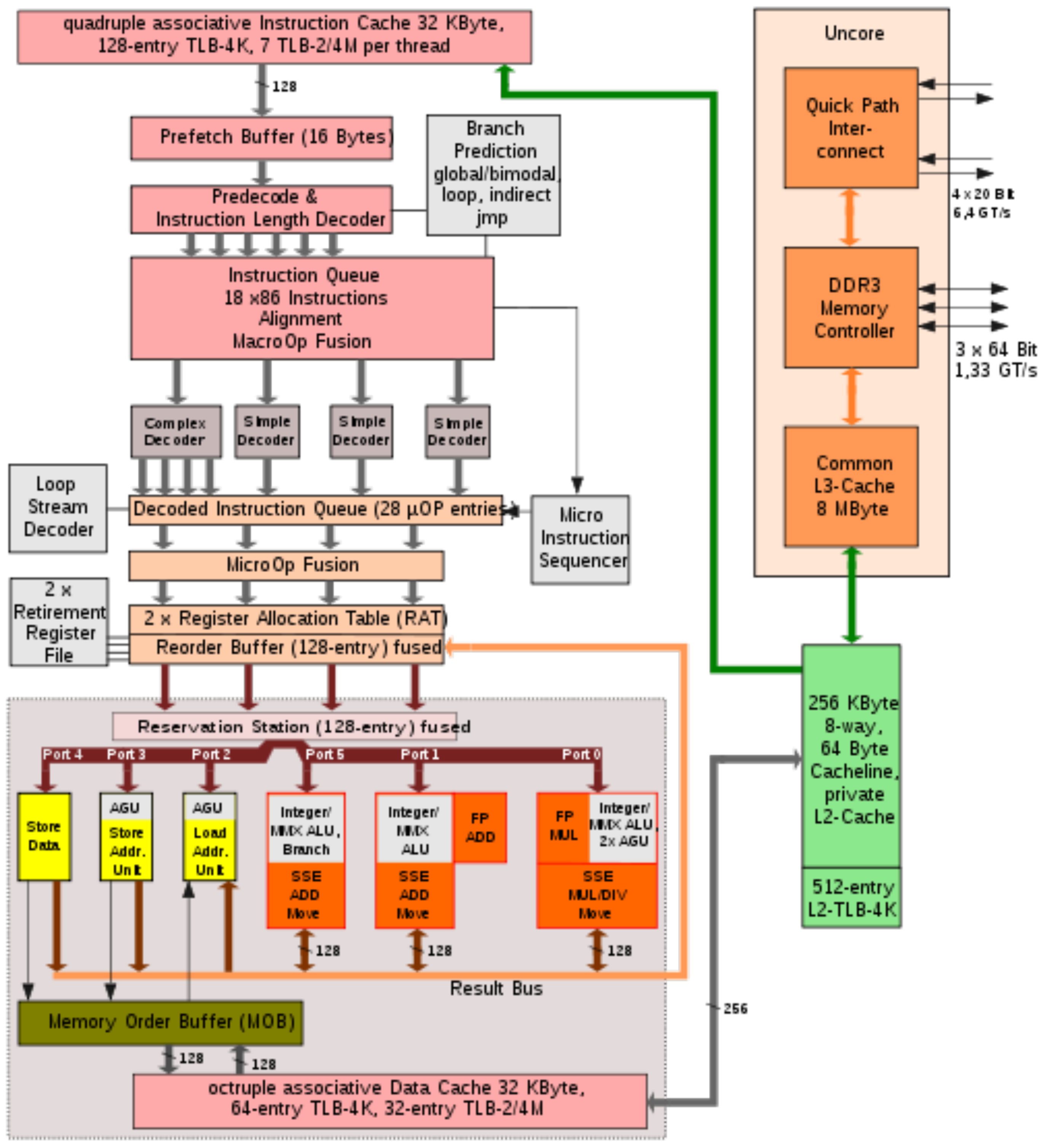
Multicore


Intel Nehalem (4 Cores)

Multicore + SMT

Nehalem

Intel Nehalem microarchitecture



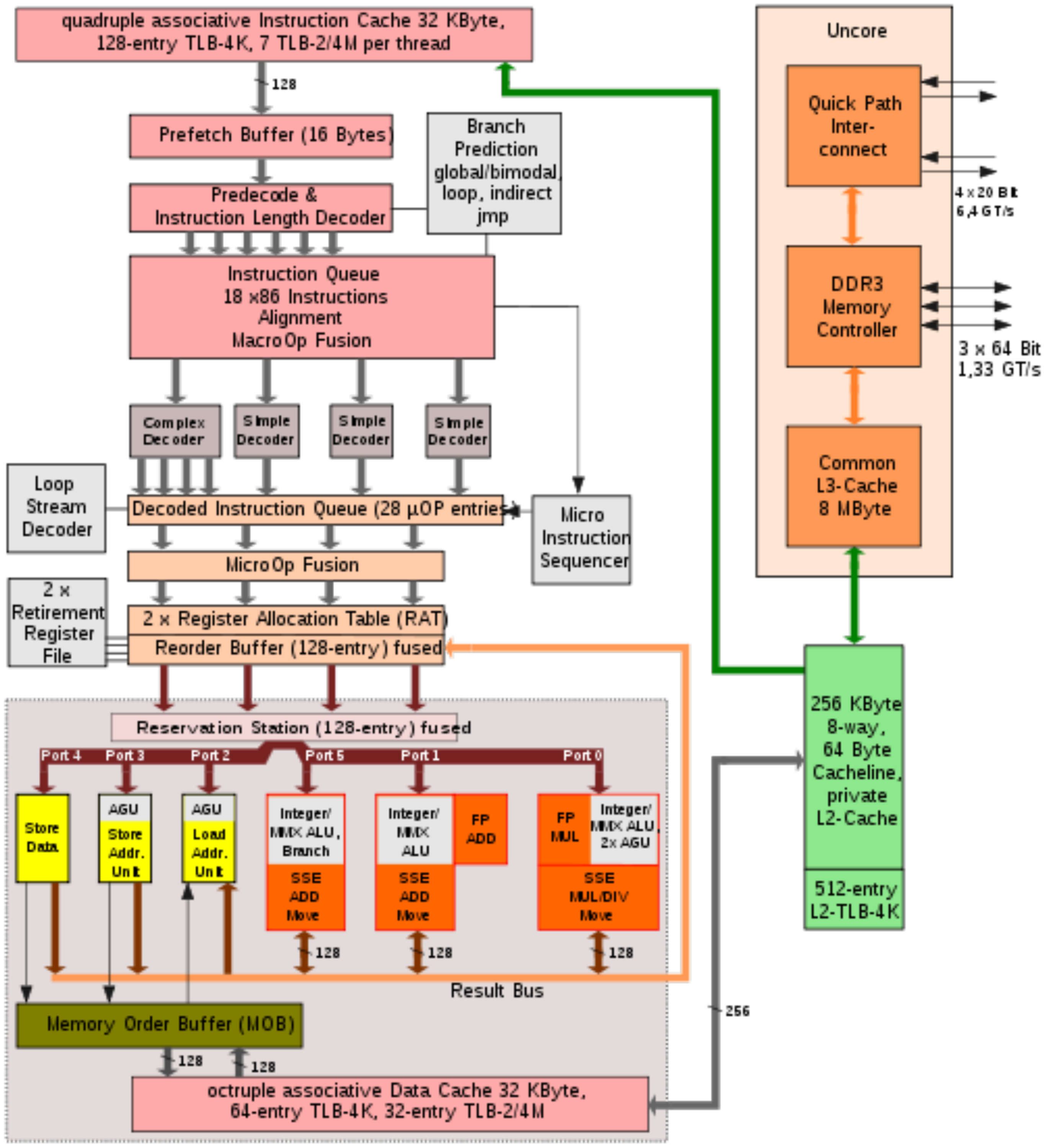
quadruple associative Instruction Cache 32 KByte, 128-entry TLB-4K, 7 TLB-2/4M per thread
128
Prefetch Buffer (16 Bytes)
Branch Prediction global/bimodal, loop, indirect jmp
Predecode & Instruction Length Decoder
Instruction Queue 18 x86 Instructions Alignment MacroOp Fusion
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
Loop Stream Decoder
Decoded Instruction Queue (28 µOP entries)
Micro Instruction Sequencer
MicroOp Fusion
2 x Retirement Register File
2 x Register Allocation Table (RAT)
Reorder Buffer (128-entry) fused
Reservation Station (128-entry) fused
Port 4
Port 3
Port 2
Port 5
Port 1
Port 0
Store Data
AGU Store Addr. Unit
AGU Load Addr. Unit
Integer/ MMX ALU, Branch
SSE ADD Move
Integer/ MMX ALU
SSE ADD Move
FP ADD
FP MUL
Integer/ MMX ALU, 2x AGU
SSE MUL/DIV Move
128
128
128
Result Bus
Memory Order Buffer (MOB)
128
128
octruple associative Data Cache 32 KByte, 64-entry TLB-4K, 32-entry TLB-2/4M
256
Uncore
Quick Path Inter-connect
4 x 20 Bit 6,4 GT/s
DDR3 Memory Controller
3 x 64 Bit 1,33 GT/s
Common L3-Cache 8 MByte
256 KByte 8-way, 64 Byte Cacheline, private L2-Cache
512-entry L2-TLB-4K

GT/s: gigatransfers per second

Nehalem

Fetch

Intel Nehalem microarchitecture

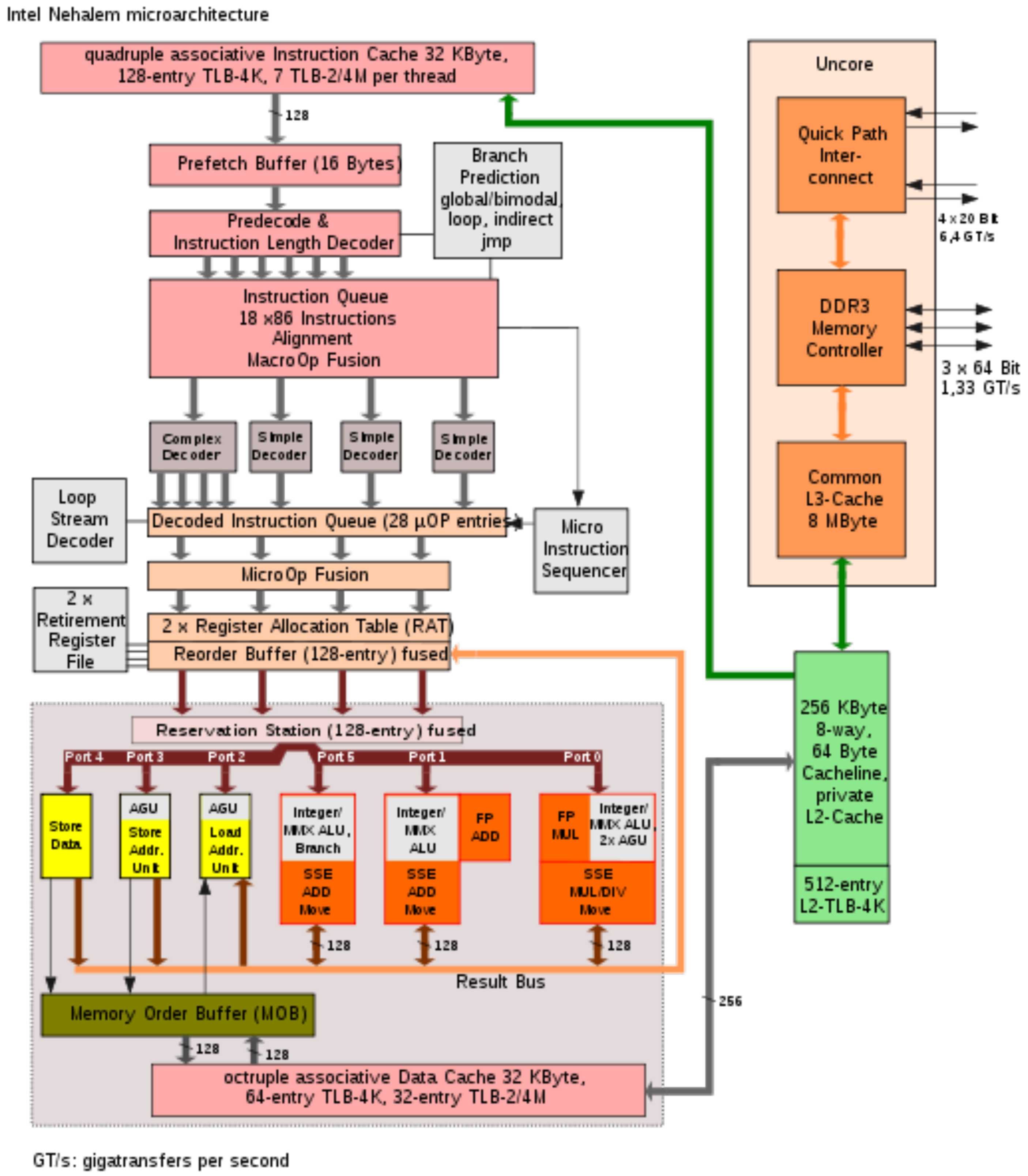

quadruple associative Instruction Cache 32 KByte, 128-entry TLB-4K, 7 TLB-2/4M per thread
128
Prefetch Buffer (16 Bytes)
Branch Prediction global/bimodal, loop, indirect jmp
Predecode & Instruction Length Decoder
Instruction Queue 18 x86 Instructions Alignment MacroOp Fusion
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
Loop Stream Decoder
Decoded Instruction Queue (28 µOP entries)
Micro Instruction Sequencer
MicroOp Fusion
2 x Retirement Register File
2 x Register Allocation Table (RAT)
Reorder Buffer (128-entry) fused
Reservation Station (128-entry) fused
Port 4
Port 3
Port 2
Port 5
Port 1
Port 0
Store Data
AGU Store Addr. Unit
AGU Load Addr. Unit
Integer/ MMX ALU, Branch
SSE ADD Move
Integer/ MMX ALU
SSE ADD Move
FP ADD
FP MUL
Integer/ MMX ALU, 2x AGU
SSE MUL/DIV Move
128
128
128
Result Bus
Memory Order Buffer (MOB)
128
128
octruple associative Data Cache 32 KByte, 64-entry TLB-4K, 32-entry TLB-2/4M
256
Uncore
Quick Path Inter-connect
4 x 20 Bit 6,4 GT/s
DDR3 Memory Controller
3 x 64 Bit 1,33 GT/s
Common L3-Cache 8 MByte
256 KByte 8-way, 64 Byte Cacheline, private L2-Cache
512-entry L2-TLB-4K

GT/s: gigatransfers per second

Nehalem

Fetch

Decode

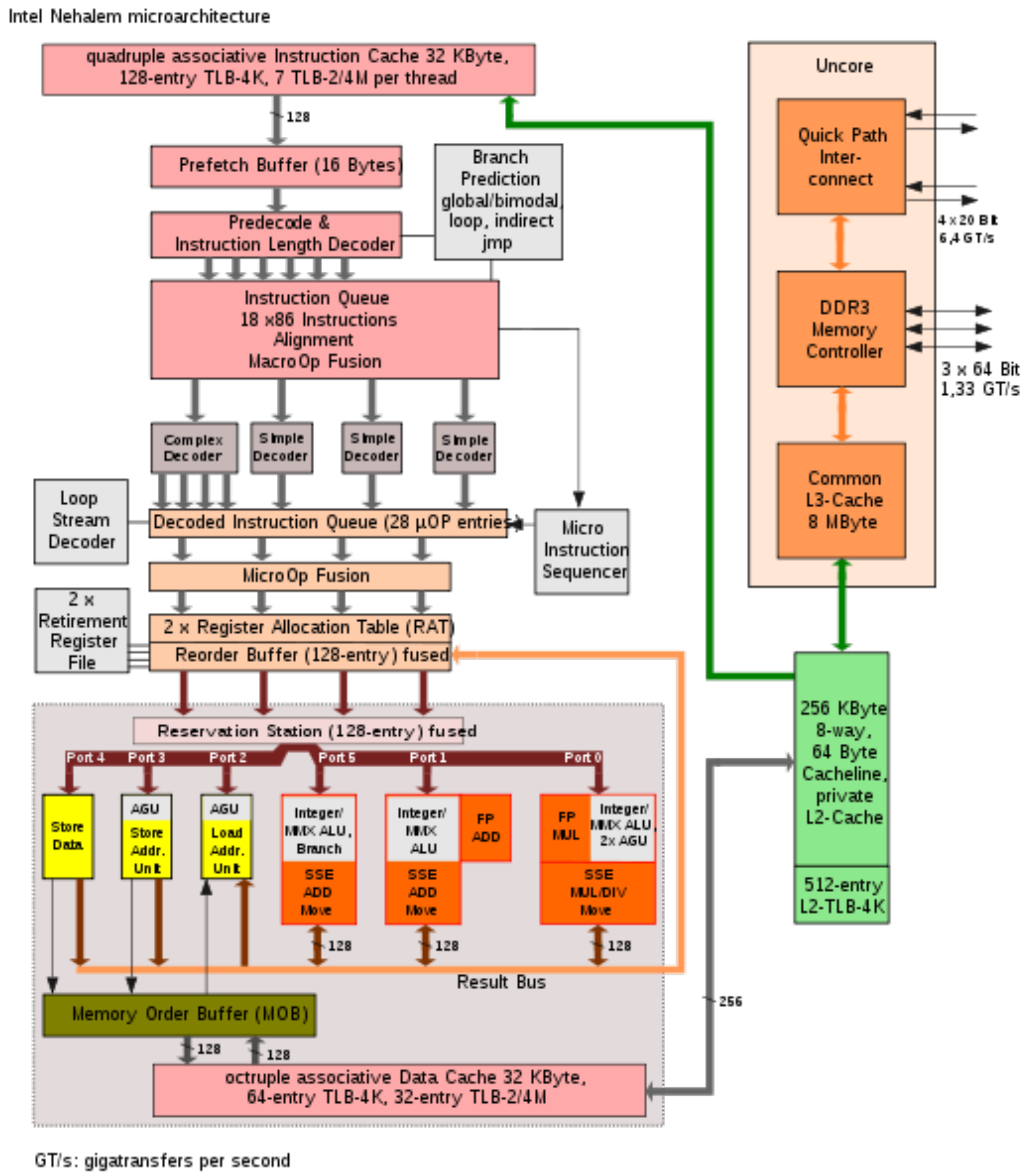

Intel Nehalem microarchitecture
quadruple associative Instruction Cache 32 KByte, 128-entry TLB-4K, 7 TLB-2/4M per thread
128
Prefetch Buffer (16 Bytes)
Branch Prediction global/bimodal, loop, indirect jmp
Predecode & Instruction Length Decoder
Instruction Queue 18 x86 Instructions Alignment MacroOp Fusion
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
Loop Stream Decoder
Decoded Instruction Queue (28 µOP entries)
Micro Instruction Sequencer
MicroOp Fusion
2 x Retirement Register File
2 x Register Allocation Table (RAT)
Reorder Buffer (128-entry) fused
Reservation Station (128-entry) fused
Port 4
Port 3
Port 2
Port 5
Port 1
Port 0
Store Data
AGU Store Addr. Unit
AGU Load Addr. Unit
Integer/ MMX ALU, Branch
SSE ADD Move
Integer/ MMX ALU
SSE ADD Move
FP ADD
FP MUL
Integer/ MMX ALU, 2x AGU
SSE MUL/DIV Move
128
128
128
Result Bus
Memory Order Buffer (MOB)
128
128
octruple associative Data Cache 32 KByte, 64-entry TLB-4K, 32-entry TLB-2/4M
256
Uncore
Quick Path Inter-connect
4 x 20 Bit 6,4 GT/s
DDR3 Memory Controller
3 x 64 Bit 1,33 GT/s
Common L3-Cache 8 MByte
256 KByte 8-way, 64 Byte Cacheline, private L2-Cache
512-entry L2-TLB-4K
GT/s: gigatransfers per second

Nehalem

Fetch

Decode

Execute



Intel Nehalem microarchitecture
quadruple associative Instruction Cache 32 KByte, 128-entry TLB-4K, 7 TLB-2/4M per thread
128
Prefetch Buffer (16 Bytes)
Branch Prediction global/bimodal, loop, indirect jmp
Predecode & Instruction Length Decoder
Instruction Queue 18 x86 Instructions Alignment MacroOp Fusion
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
Loop Stream Decoder
Decoded Instruction Queue (28 µOP entries)
Micro Instruction Sequencer
MicroOp Fusion
2 x Retirement Register File
2 x Register Allocation Table (RAT)
Reorder Buffer (128-entry) fused
Reservation Station (128-entry) fused
Port 4
Port 3
Port 2
Port 5
Port 1
Port 0
Store Data
AGU Store Addr. Unit
AGU Load Addr. Unit
Integer/ MMX ALU, Branch
SSE ADD Move
Integer/ MMX ALU
SSE ADD Move
FP ADD
FP MUL
Integer/ MMX ALU, 2x AGU
SSE MUL/DIV Move
128
128
128
Result Bus
Memory Order Buffer (MOB)
128
128
octruple associative Data Cache 32 KByte, 64-entry TLB-4K, 32-entry TLB-2/4M
256
Uncore
Quick Path Inter-connect
4 x 20 Bit 6,4 GT/s
DDR3 Memory Controller
3 x 64 Bit 1,33 GT/s
Common L3-Cache 8 MByte
256 KByte 8-way, 64 Byte Cacheline, private L2-Cache
512-entry L2-TLB-4K
GT/s: gigatransfers per second

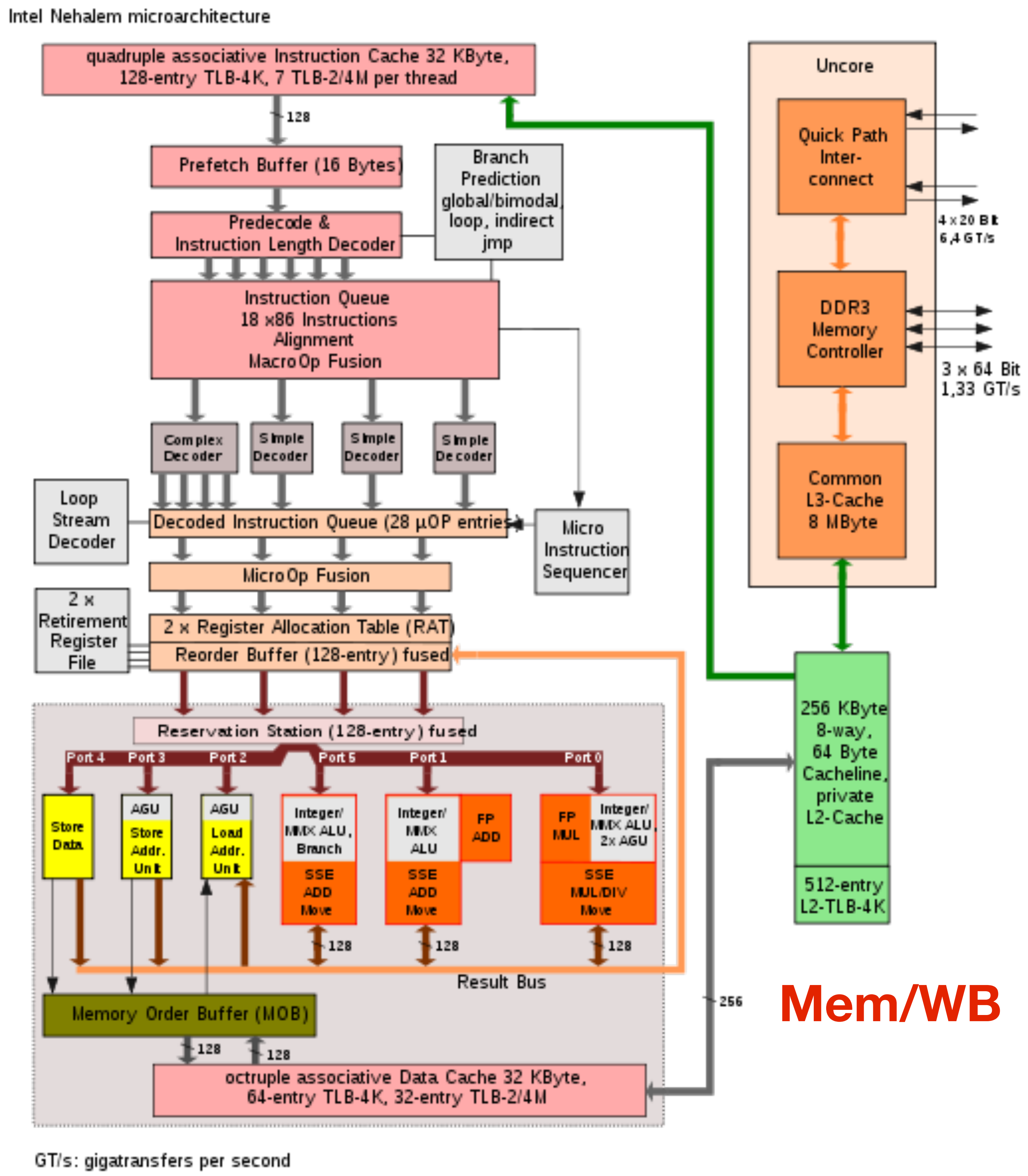
Nehalem

Fetch

Decode

Execute

Mem/WB


Intel Nehalem microarchitecture
quadruple associative Instruction Cache 32 KByte, 128-entry TLB-4K, 7 TLB-2/4M per thread
128
Prefetch Buffer (16 Bytes)
Branch Prediction global/bimodal, loop, indirect jmp
Predecode & Instruction Length Decoder
Instruction Queue 18 x86 Instructions Alignment MacroOp Fusion
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
Loop Stream Decoder
Decoded Instruction Queue (28 µOP entries)
Micro Instruction Sequencer
MicroOp Fusion
2 x Retirement Register File
2 x Register Allocation Table (RAT)
Reorder Buffer (128-entry) fused
Reservation Station (128-entry) fused
Port 4
Port 3
Port 2
Port 5
Port 1
Port 0
Store Data
AGU Store Addr. Unit
AGU Load Addr. Unit
Integer/ MMX ALU, Branch
SSE ADD Move
Integer/ MMX ALU
SSE ADD Move
FP ADD
FP MUL
Integer/ MMX ALU, 2x AGU
SSE MUL/DIV Move
128
128
128
Result Bus
Memory Order Buffer (MOB)
128
128
octruple associative Data Cache 32 KByte, 64-entry TLB-4K, 32-entry TLB-2/4M
256
Uncore
Quick Path Inter-connect
4 x 20 Bit 6,4 GT/s
DDR3 Memory Controller
3 x 64 Bit 1,33 GT/s
Common L3-Cache 8 MByte
256 KByte 8-way, 64 Byte Cacheline, private L2-Cache
512-entry L2-TLB-4K
GT/s: gigatransfers per second

Nehalem Micro-architecture: Dynamically Scalable and Innovative New Design

Scalable from 2 to 8 cores

Micro-architecture enhancements (4 –wide)

2-way simultaneous multi-threading

Integrated memory controller

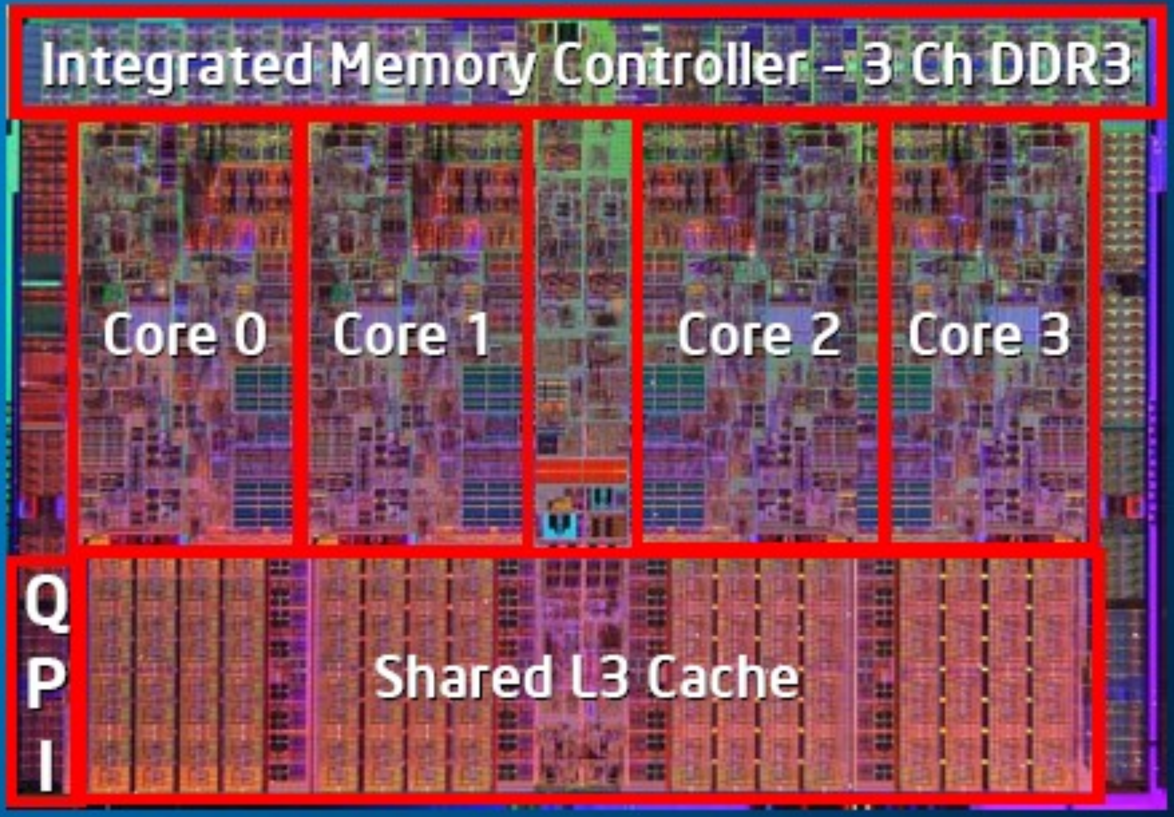
QuickPath interconnect

Shared and Inclusive Level-3 cache

Dynamic power management

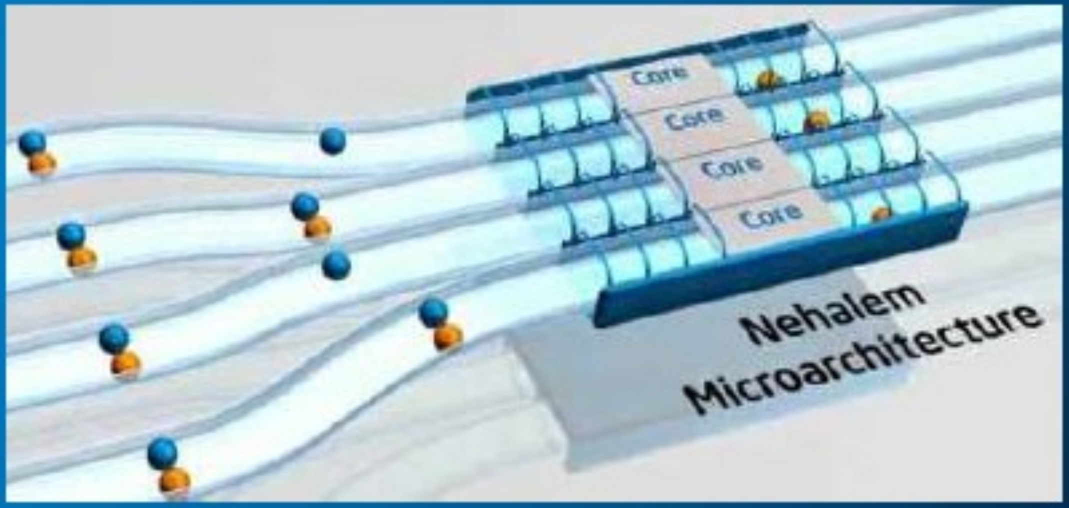
SSE 4.2

Production: Q4'08



Simultaneous Multi-Threading (SMT)

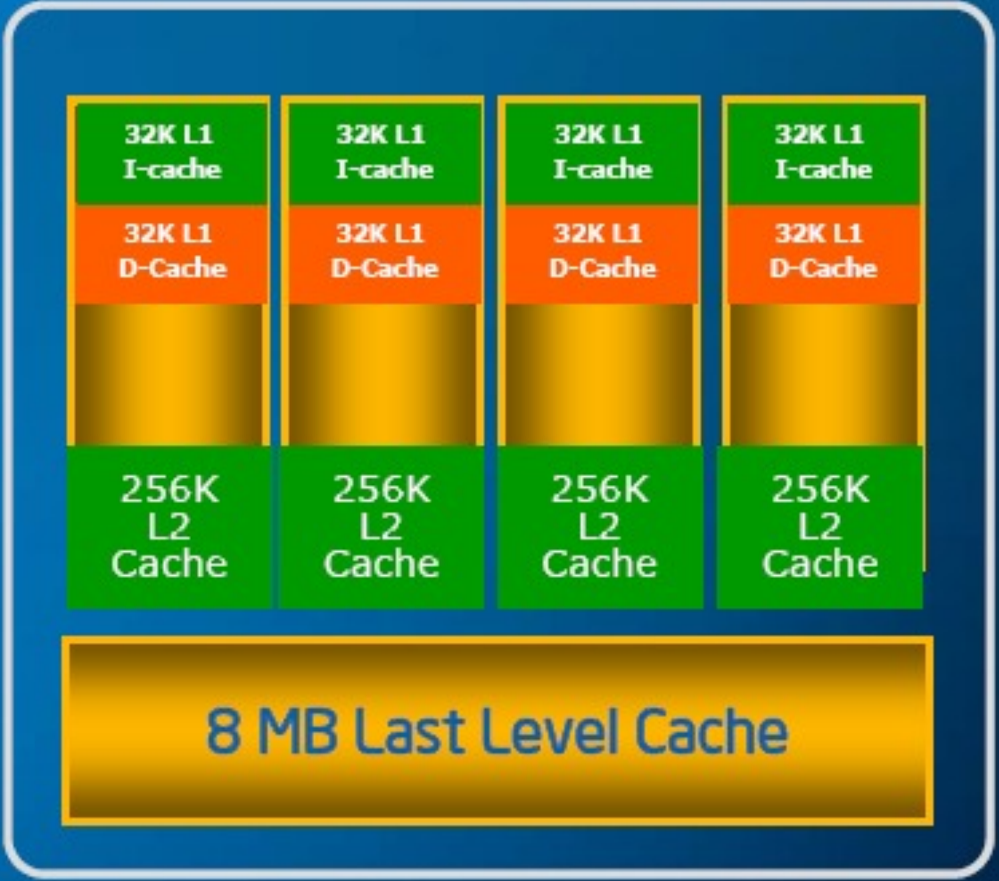
- Each core able to execute two software threads simultaneously
- Extremely power efficient
- Enhanced with larger caches and more memory bandwidth
- Benefits
  - Highly threaded workloads (eg, multi-media apps, databases, search engines)
  - Multi-Tasking scenarios



Simultaneous Multi-threading Enhances Performance and Energy Efficiency

Enhanced Cache Subsystem

- New 3-level Cache Hierarchy
  - ➢ L1 cache same as Intel Core™ uArch
    - 32 KB Instruction/32 KB Data
  - ➢ New 256 KB/core, low latency L2 cache
  - ➢ New Large 8MB fully-shared L3 cache
    - Inclusive Cache Policy - minimize snoop traffic
- New 2-level TLB hierarchy
  - ➢ Adds 2nd level 512 entry Translation Look-aside Buffer



Superior multi-level shared cache extends Intel® Smart Cache technology

Nehalem in a Nutshell

- Up to 8 cores (i7, 4 cores)
- 2 SMT threads per core
- 20+ stage pipeline
- x86 instructions translated to RISC-like uops
- Superscalar, 4 “instructions” (uops) per cycle (more with fusing)
- Caches (i7)
  - 32KB 4-way set-associative I cache per core
  - 32KB, 8-way set-associative D cache per core
  - 256 KB unified 8-way set-associative L2 cache per core
  - 8 MB shared 16-way set-associative L3 cache

Key Points

Key Points

- Network vs. Bus

Key Points

- Network vs. Bus
- Message-passing vs. Shared Memory

Key Points

- Network vs. Bus
- Message-passing vs. Shared Memory
- Shared Memory is more intuitive, but creates problems for both the programmer (memory consistency, requiring synchronization) and the architect (cache coherency).

Key Points

- Network vs. Bus
- Message-passing vs. Shared Memory
- Shared Memory is more intuitive, but creates problems for both the programmer (memory consistency, requiring synchronization) and the architect (cache coherency).
- Multithreading gives the illusion of multiprocessing (including, in many cases, the performance) with very little additional hardware.

Key Points

- Network vs. Bus
- Message-passing vs. Shared Memory
- Shared Memory is more intuitive, but creates problems for both the programmer (memory consistency, requiring synchronization) and the architect (cache coherency).
- Multithreading gives the illusion of multiprocessing (including, in many cases, the performance) with very little additional hardware.
- When multiprocessing happens within a single die/processor, we call that a chip multiprocessor, or a multi-core architecture.