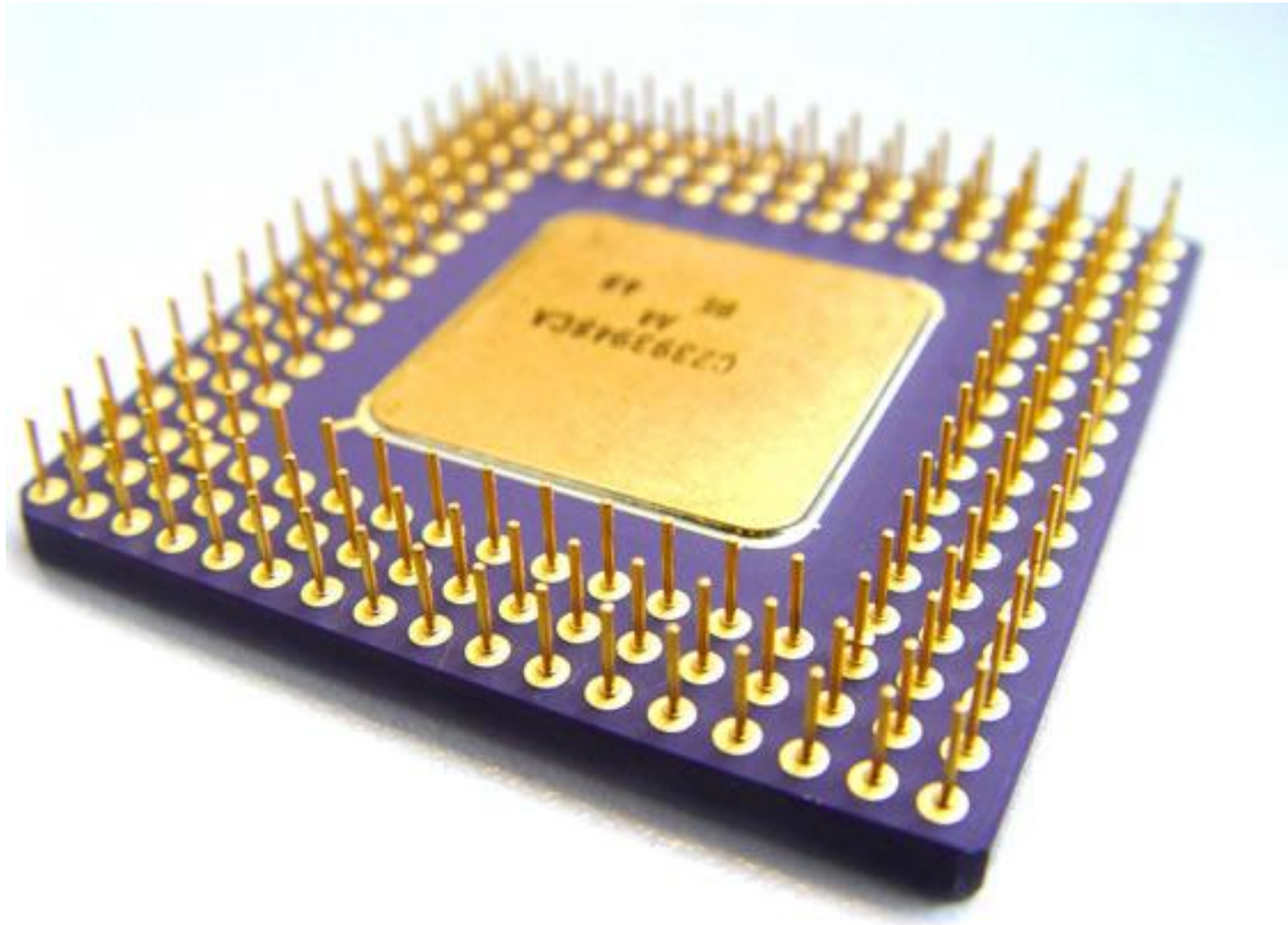
CPU架构浅析

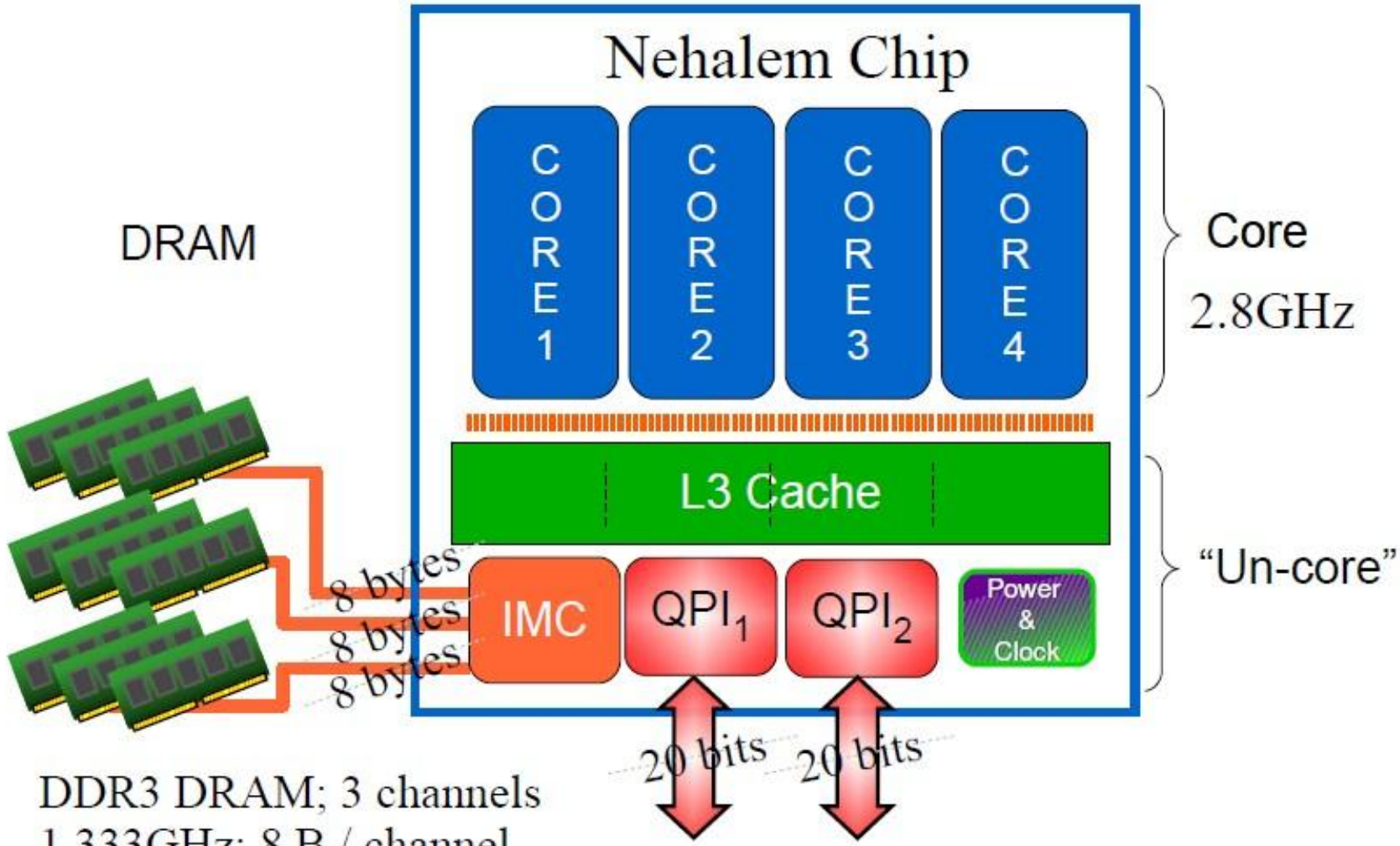
阿里数据库技术 —— 何登成

微博：何_登成

分享大纲

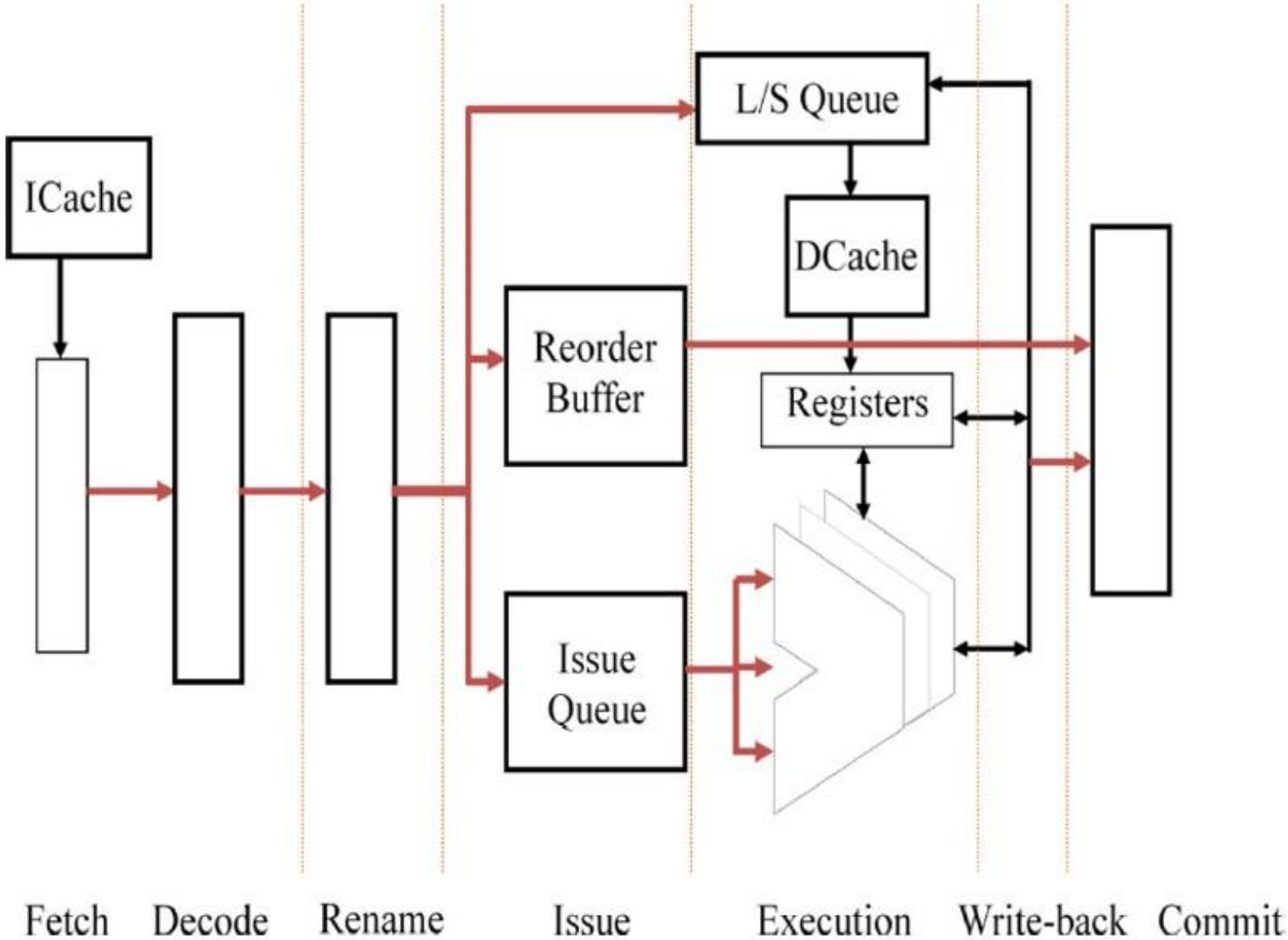
- 先介绍一个CPU的架构
  - Fetch to retire
- 通过CPU架构，理解一些技术点
  - Super pipelining
  - Superscalar
  - Simultaneous Multi-Threading（SMT）
  - Out-of-Order Execution
  - Performance vs Clock Cycle
- 再详细分析两个例子
  - Cache Coherence（缓存一致性）
  - Memory Consistency（Memory Model）
    - Intel，ARM



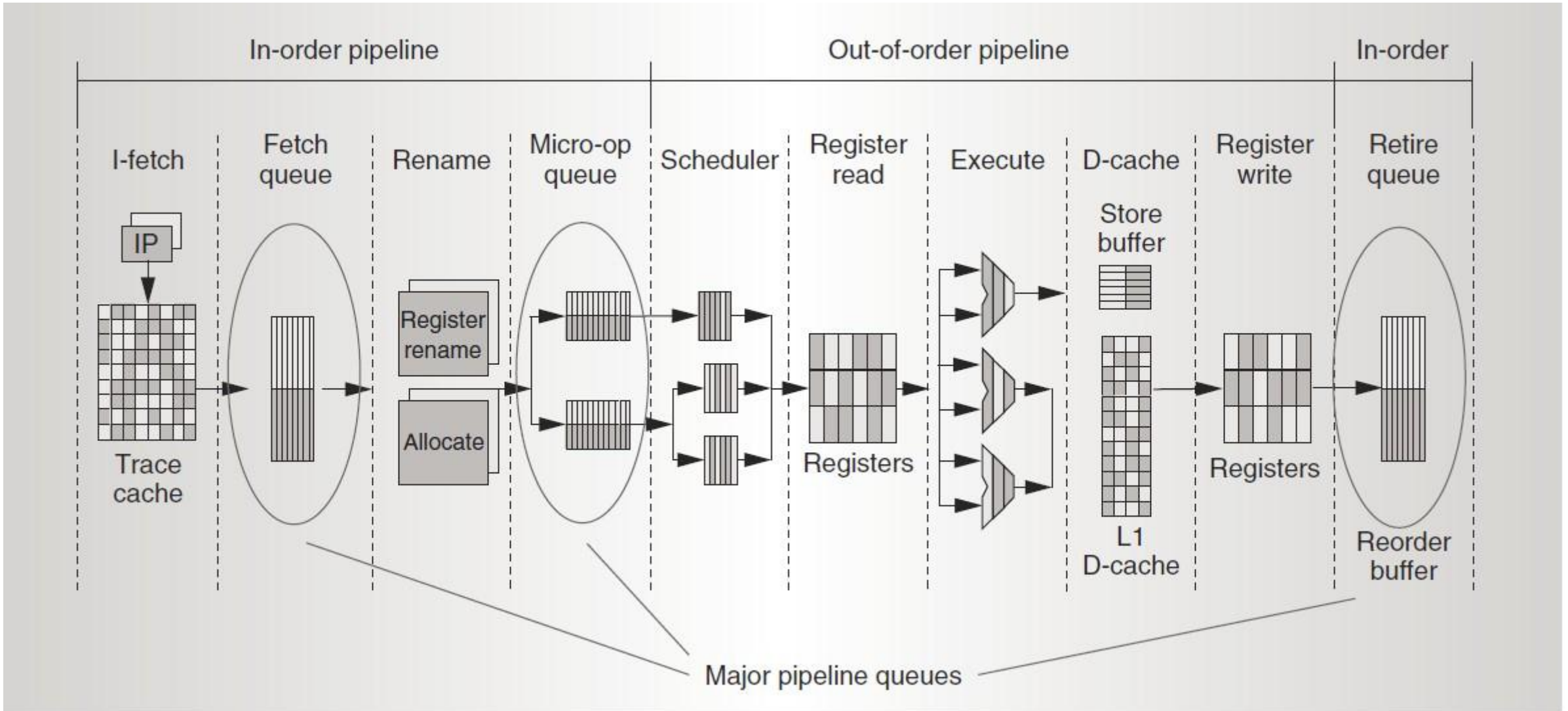


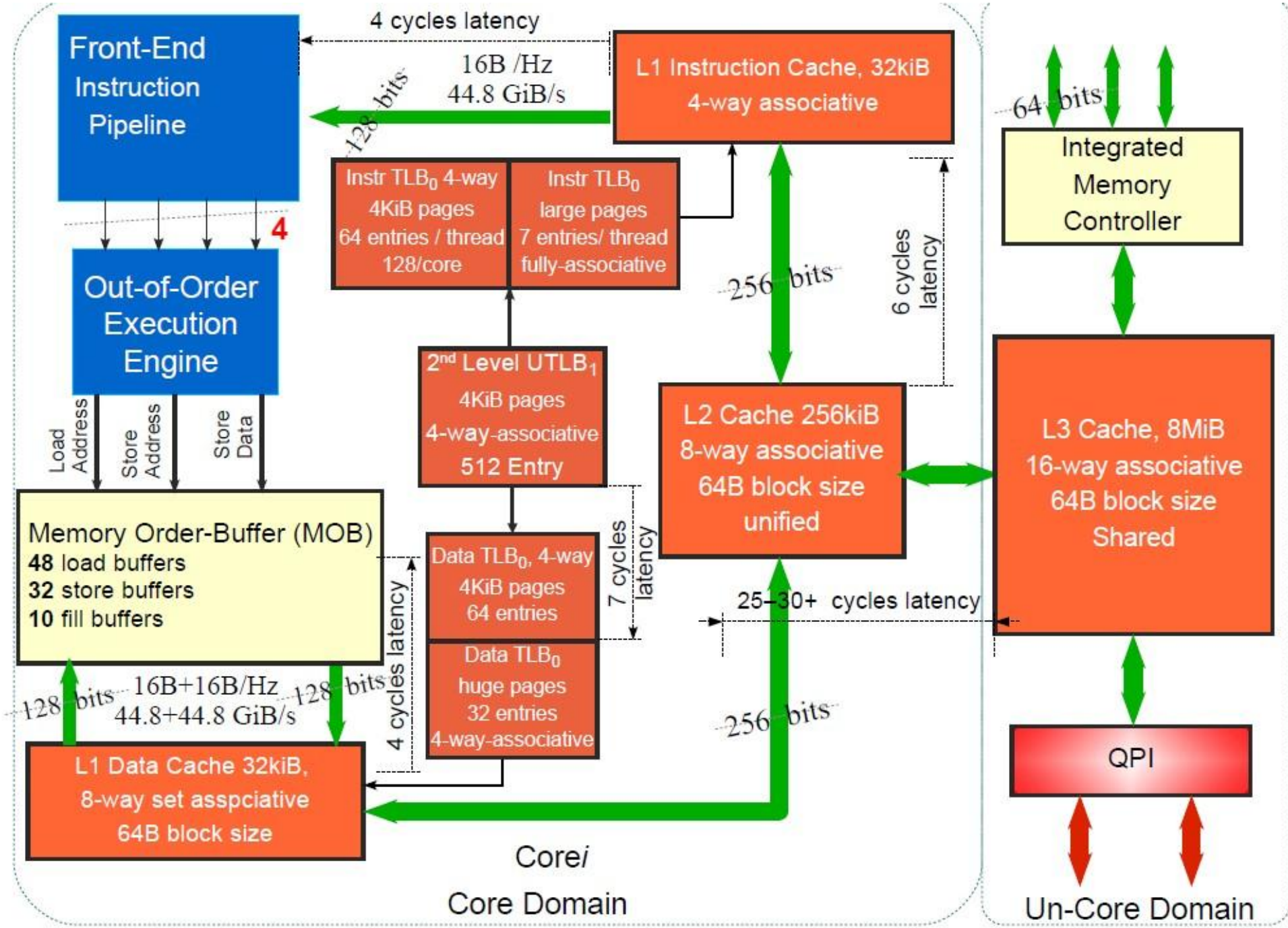
DDR3 DRAM; 3 channels
1.333GHz; 8 B / channel
31.992 GB/s aggregate
7.998 GB/s / core

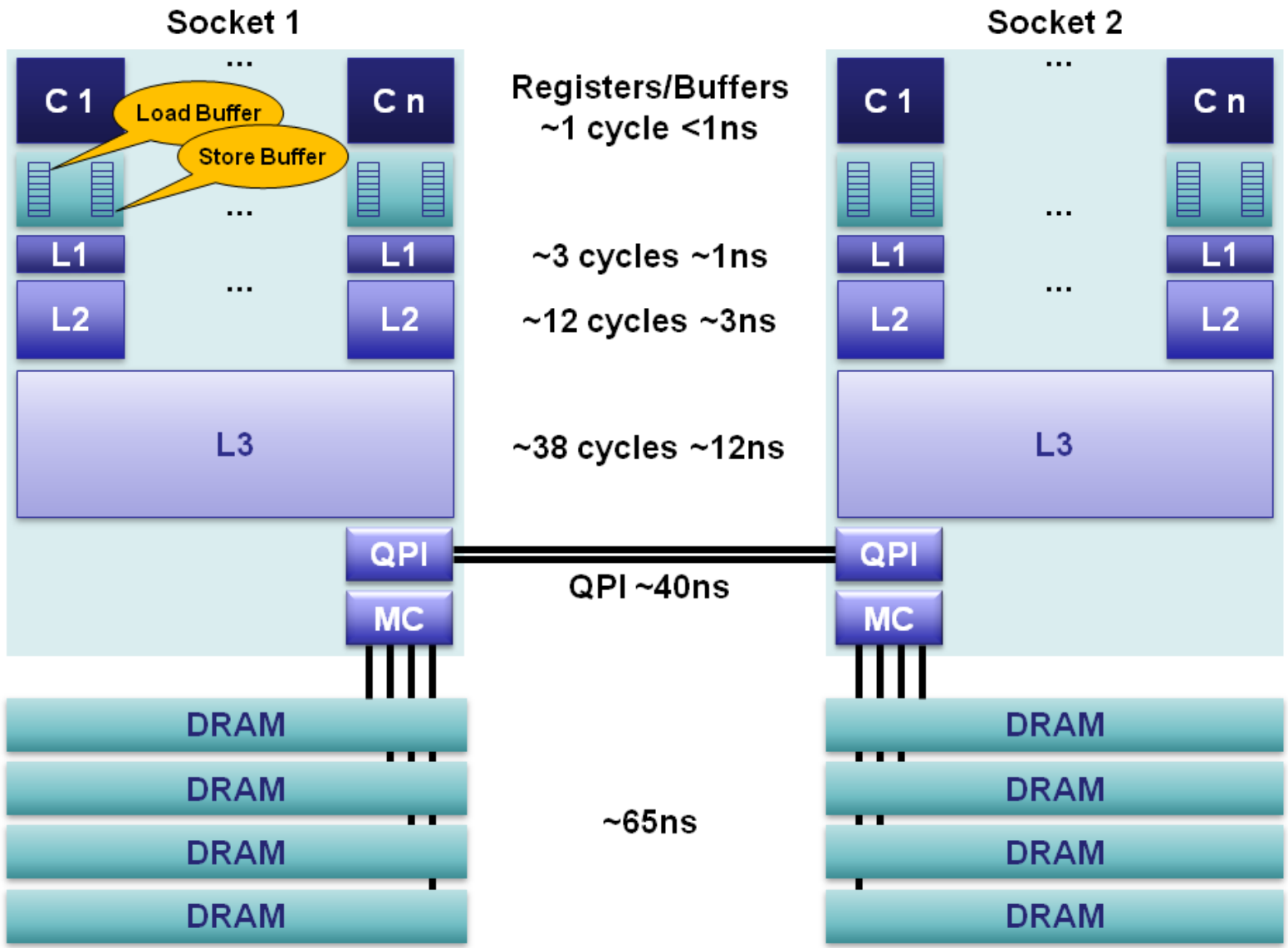
Intel® QPI point-to-point link
6.4 GT/s; full-duplex;
12.8 GiB/s + **12.8**GiB/s


L/S Queue
ICache
DCache
Reorder
Buffer
Registers
Issue
Queue
Fetch
Decode
Rename
Issue
Execution
Write-back
Commit

- The early stages of the processor **fetch-in** several **macro-instructions** at a time (say in a cache block) and
- **decode** them (break them down) into sequences of **micro-ops**.
- The micro-ops are **buffered** at various places where they can be picked up and scheduled to use the **FUs** in parallel if data dependencies are not violated. In Nehalem, micro-ops are issued to stations were they reserve their position for subsequent,
- **dispatching** as soon as **their input operands become available**.
- Finally, completed micro-ops **retire** and post their results to permanent storage.


In-order pipeline
Out-of-order pipeline
In-order
I-fetch
Fetch queue
Rename
Micro-op queue
Scheduler
Register read
Execute
D-cache
Register write
Retire queue
IP
Trace cache
Register rename
Allocate
Registers
Store buffer
L1 D-cache
Registers
Reorder buffer
Major pipeline queues


Front-End
Instruction
Pipeline
4 cycles latency
16B /Hz
44.8 GiB/s
128 bits
L1 Instruction Cache, 32kiB
4-way associative
64 bits
Integrated
Memory
Controller
Instr TLB0 4-way
4KiB pages
64 entries / thread
128/core
Instr TLB0
large pages
7 entries/ thread
fully-associative
256 bits
6 cycles
latency
4
Out-of-Order
Execution
Engine
2nd Level UTLB1
4KiB pages
4-way-associative
512 Entry
L2 Cache 256kiB
8-way associative
64B block size
unified
L3 Cache, 8MiB
16-way associative
64B block size
Shared
Load
Address
Store
Address
Store
Data
Memory Order-Buffer (MOB)
48 load buffers
32 store buffers
10 fill buffers
Data TLB0, 4-way
4KiB pages
64 entries
7 cycles
latency
25–30+ cycles latency
Data TLB0
huge pages
32 entries
4-way-associative
4 cycles latency
128 bits
16B+16B/Hz
44.8+44.8 GiB/s
128 bits
256 bits
L1 Data Cache 32kiB,
8-way set asspciative
64B block size
QPI
Corei
Core Domain
Un-Core Domain


Socket 1
Socket 2
C 1
C n
Load Buffer
Store Buffer
L1
L2
L3
QPI
MC
DRAM
Registers/Buffers
~1 cycle <1ns
~3 cycles ~1ns
~12 cycles ~3ns
~38 cycles ~12ns
QPI ~40ns
~65ns

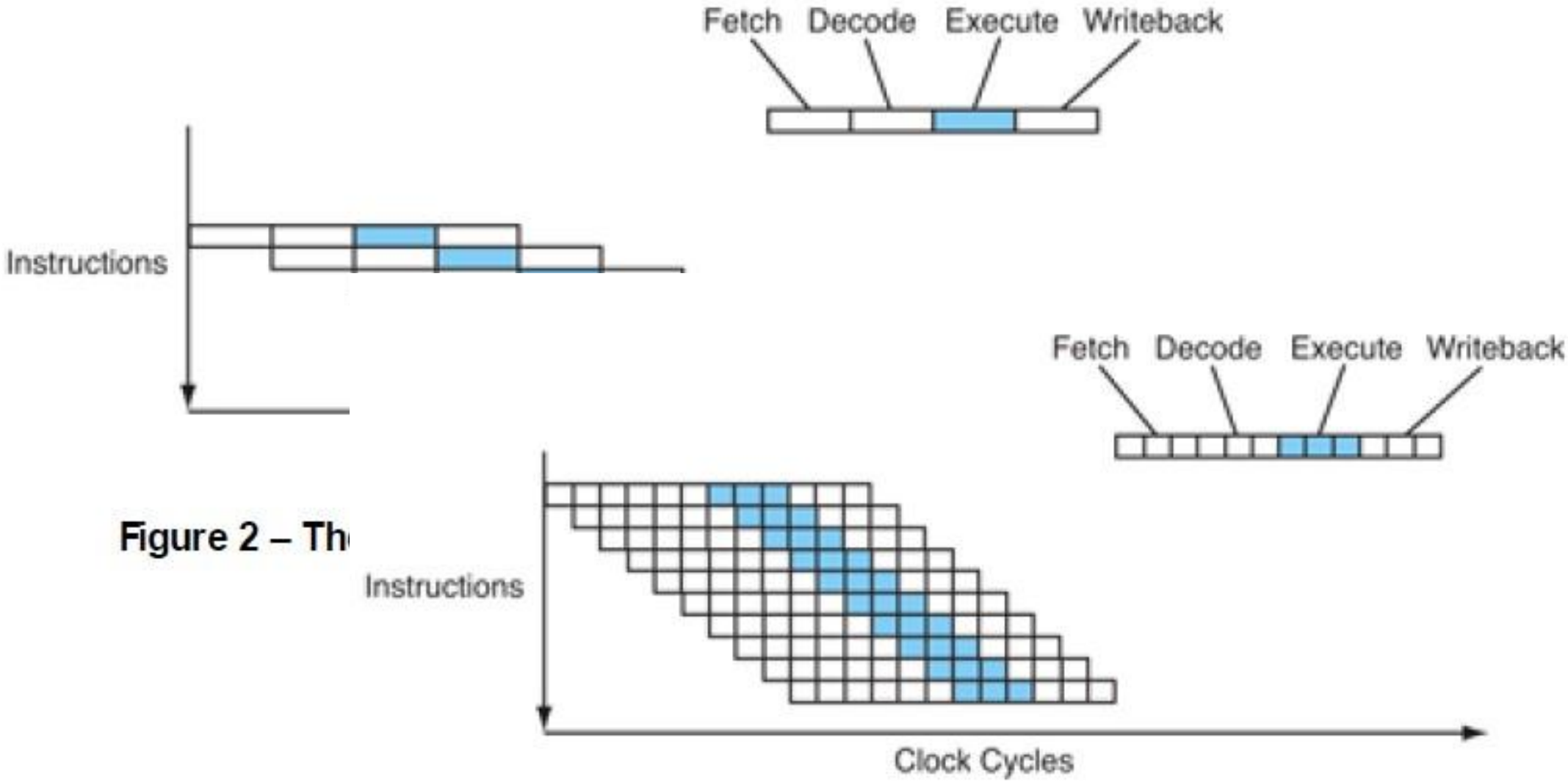


Figure 2 – Th

Figure 6 – The instruction flow of a superpipelined processor.

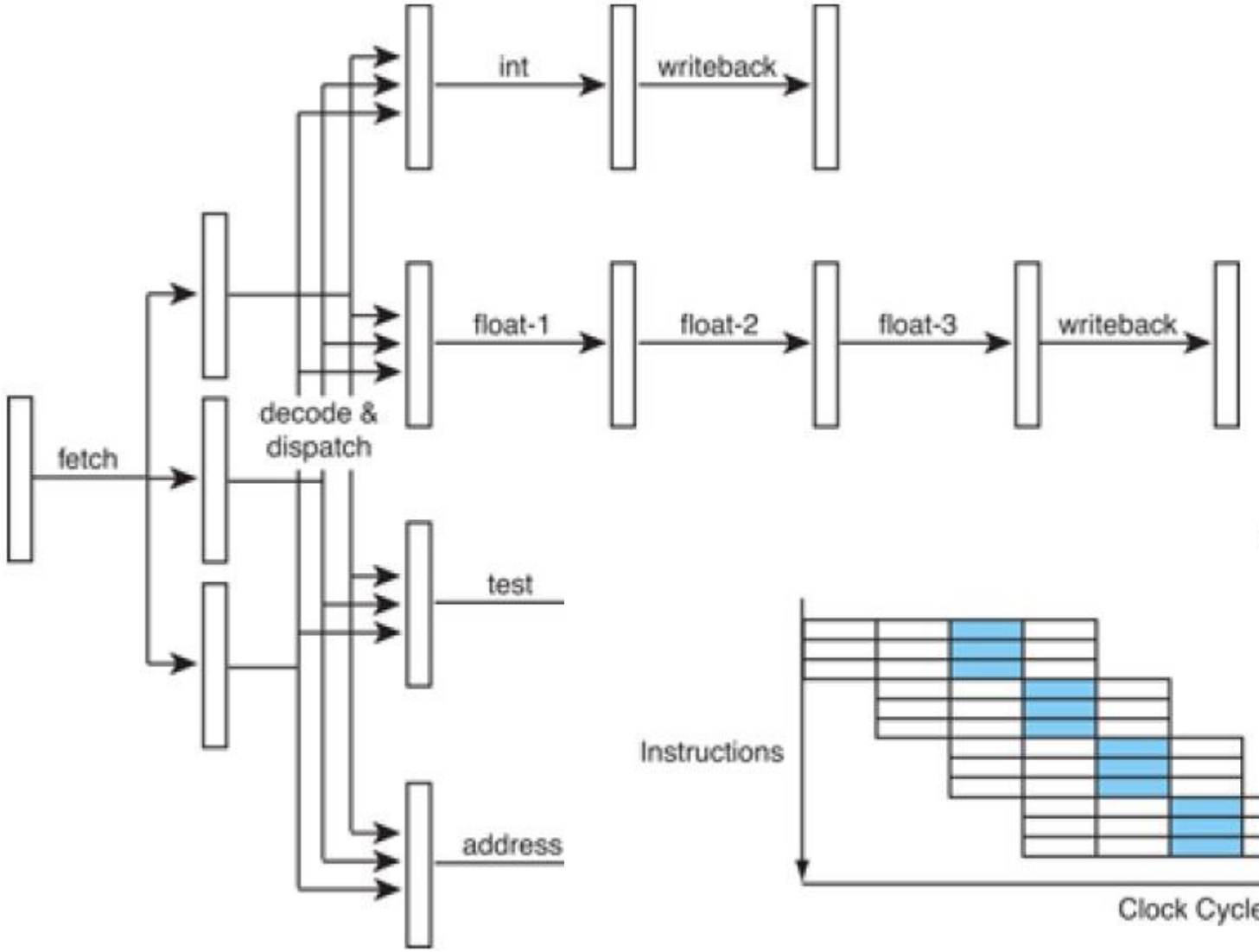


Figure 7 – A super

Figure 8 – The instruction flow of a superscalar processor.

- ILP
  - Instruction Level Parallelism
- CPI/IPC
  - Cycles per Instruction
  - Instructions per Cycle

```
# perf stat gzip file1

 Performance counter stats for 'gzip file1':

      1920.159821 task-clock                #    0.991 CPUs utilized
               13 context-switches          #    0.007 K/sec
                0 CPU-migrations            #    0.000 K/sec
              258 page-faults               #    0.134 K/sec
    5,649,595,479 cycles                    #    2.942 GHz                      [83.43%]
    1,808,339,931 stalled-cycles-frontend   #   32.01% frontend cycles idle     [83.54%]
    1,171,884,577 stalled-cycles-backend    #   20.74% backend  cycles idle     [66.77%]
    8,625,207,199 instructions              #    1.53  insns per cycle
                                            #    0.21  stalled cycles per insn [83.51%]
    1,488,797,176 branches                  #  775.351 M/sec                    [82.58%]
       53,395,139 branch-misses             #    3.59% of all branches          [83.78%]

      1.936842598 seconds time elapsed
```

```
$perf stat -p 14727
^C
 Performance counter stats for process id '14727':

        37.366764 task-clock                #    0.006 CPUs utilized
              471 context-switches          #    0.013 M/sec
                4 CPU-migrations            #    0.000 M/sec
                0 page-faults               #    0.000 M/sec
       57,734,472 cycles                    #    1.545 GHz                      [78.03%]
       44,125,966 stalled-cycles-frontend   #   76.43% frontend cycles idle     [81.73%]
       30,301,007 stalled-cycles-backend    #   52.48% backend  cycles idle     [78.16%]
       33,446,945 instructions              #    0.58  insns per cycle
                                            #    1.32  stalled cycles per insn [90.50%]
        6,291,083 branches                  #  168.360 M/sec                    [90.70%]
           82,323 branch-misses             #    1.31% of all branches          [74.49%]

      6.380130669 seconds time elapsed
```

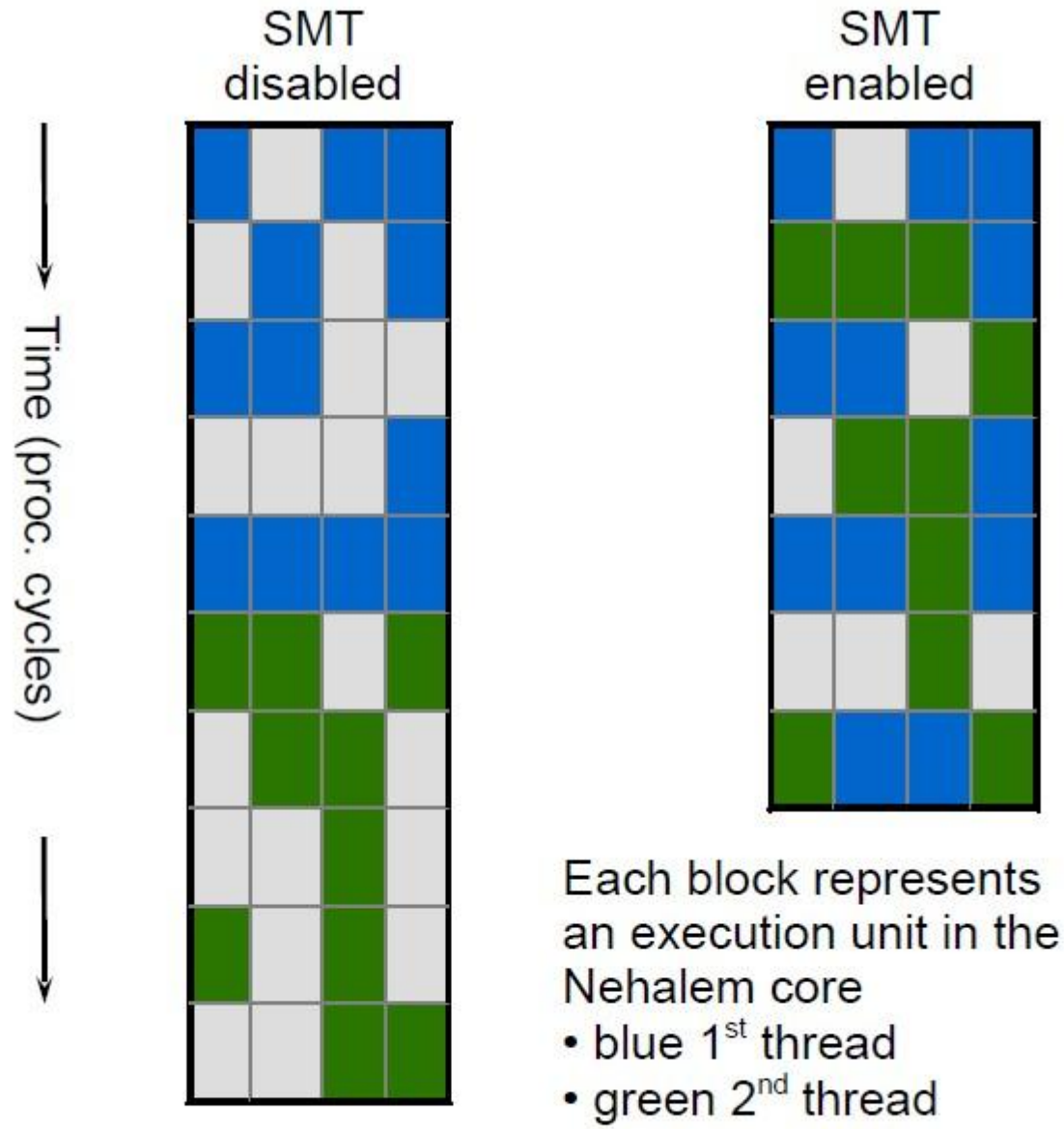
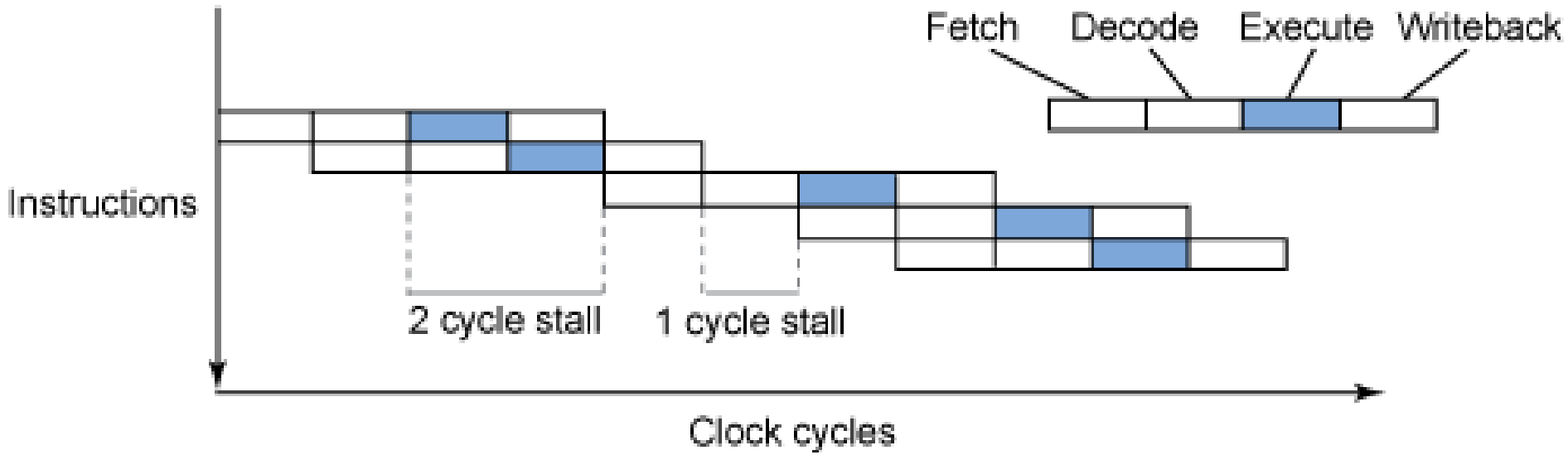


Figure 9: Simultaneous Multi-Treading (SMT) concept on Nehalem cores.

```
a = b * c;
d = a + 1;
```


Fetch
Decode
Execute
Writeback
Instructions
2 cycle stall
1 cycle stall
Clock cycles

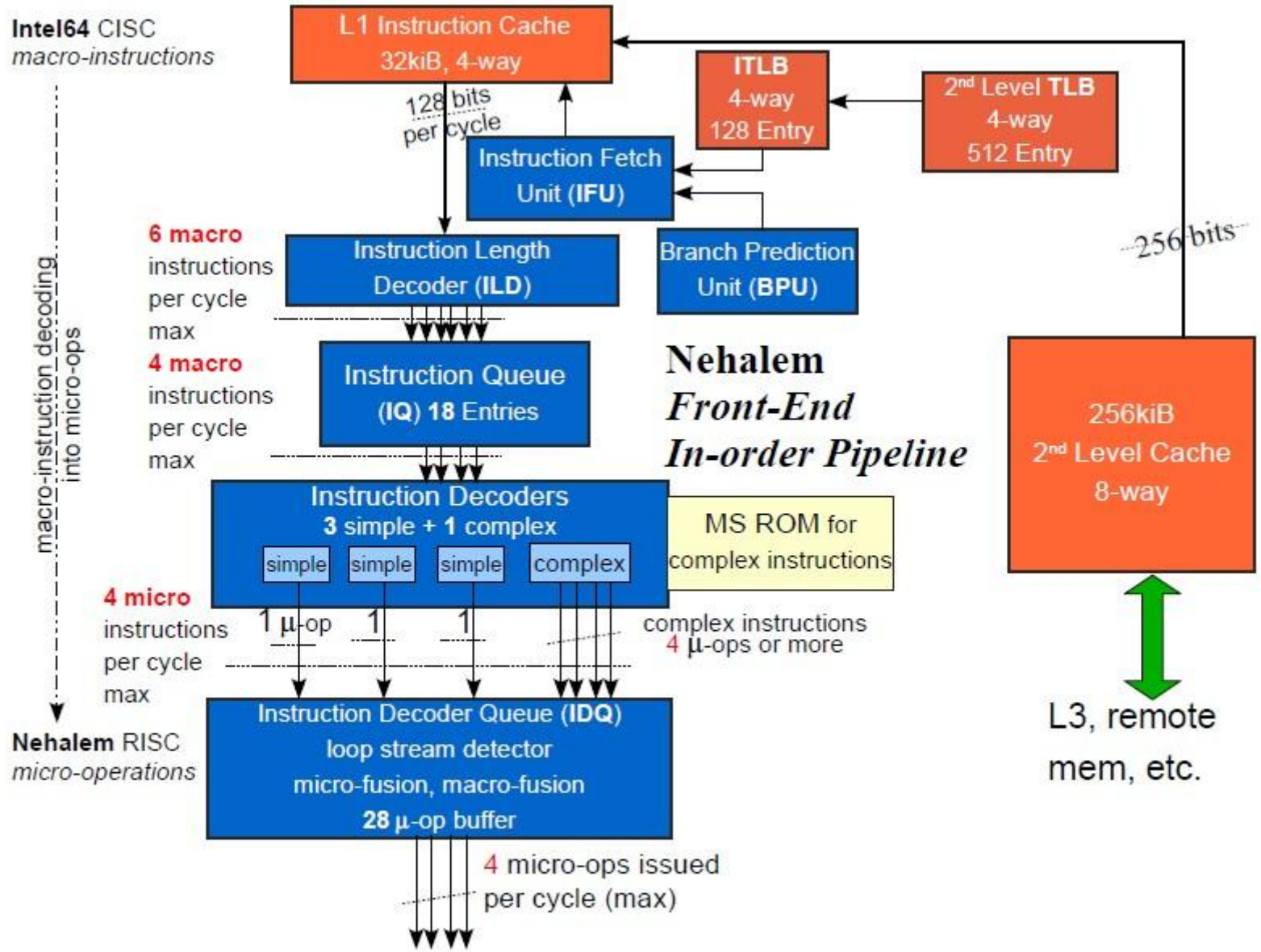


Figure 4: High-level diagram of the In-Order Front-End Nehalem Pipeline (FEP).

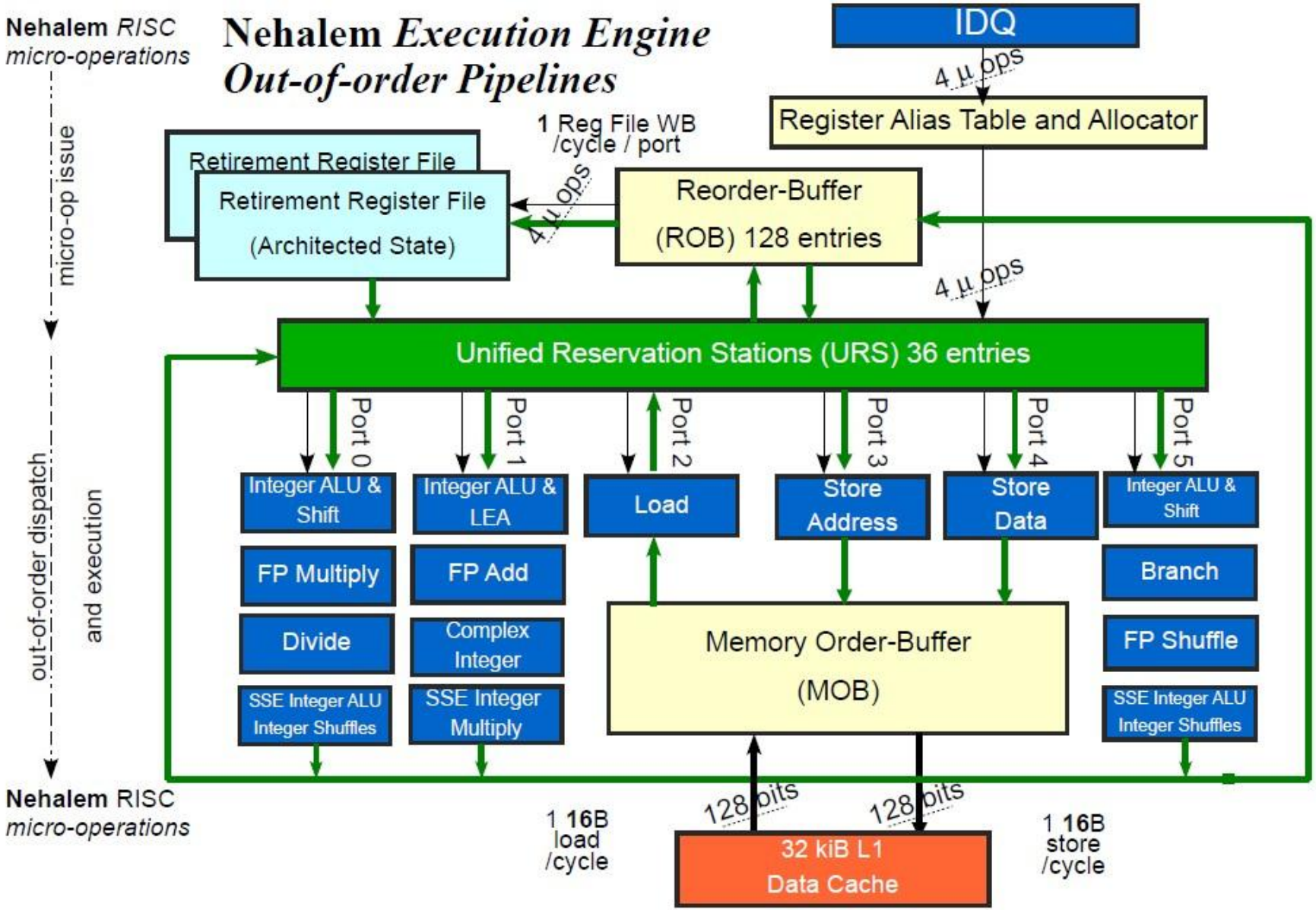


Figure 5: High-level diagram of a the out-of-order execution engine in the Nehalem core. All units are fully pipelined and can operate independently.

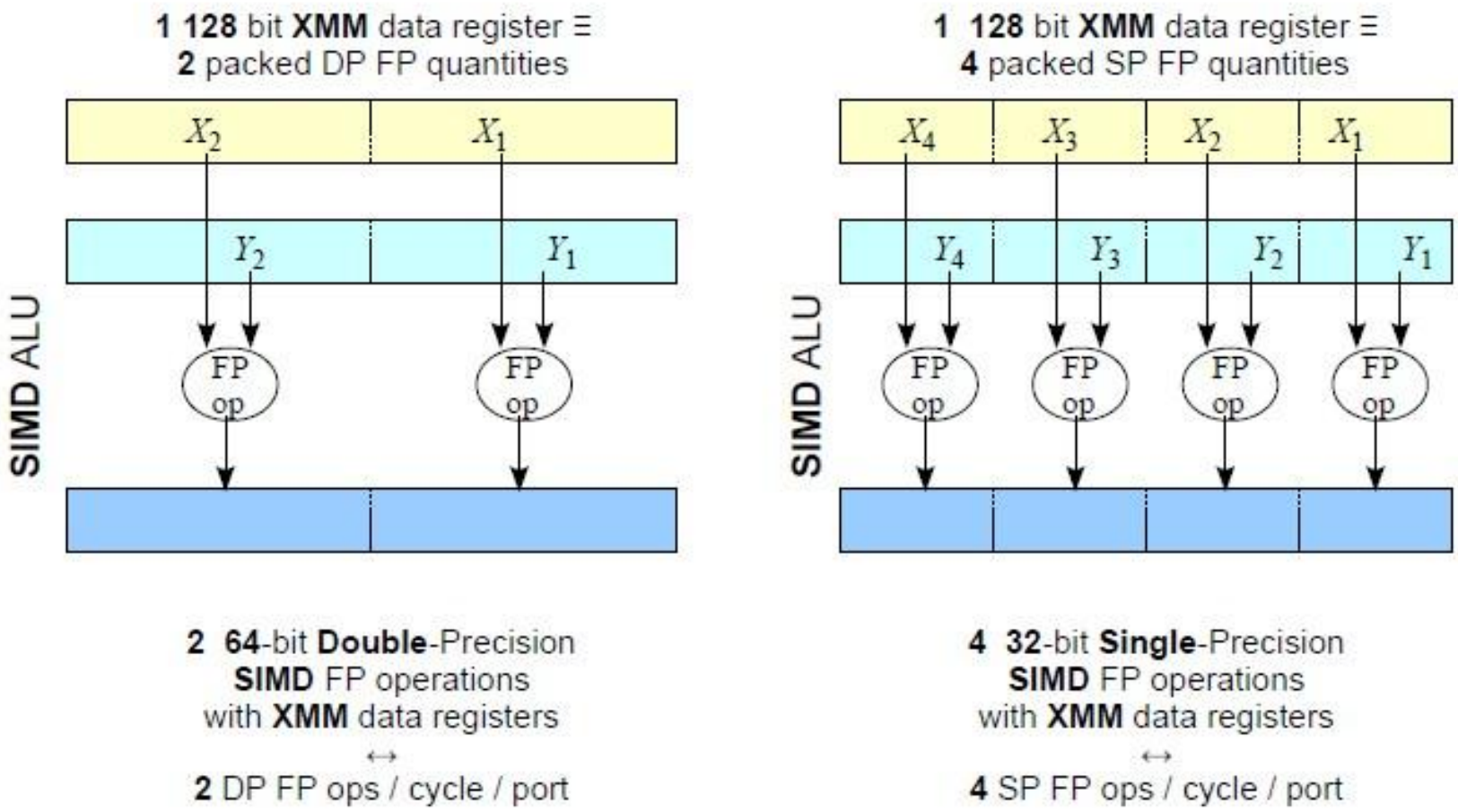


Figure 7: Floating-Point SIMD Operations in Nehalem.

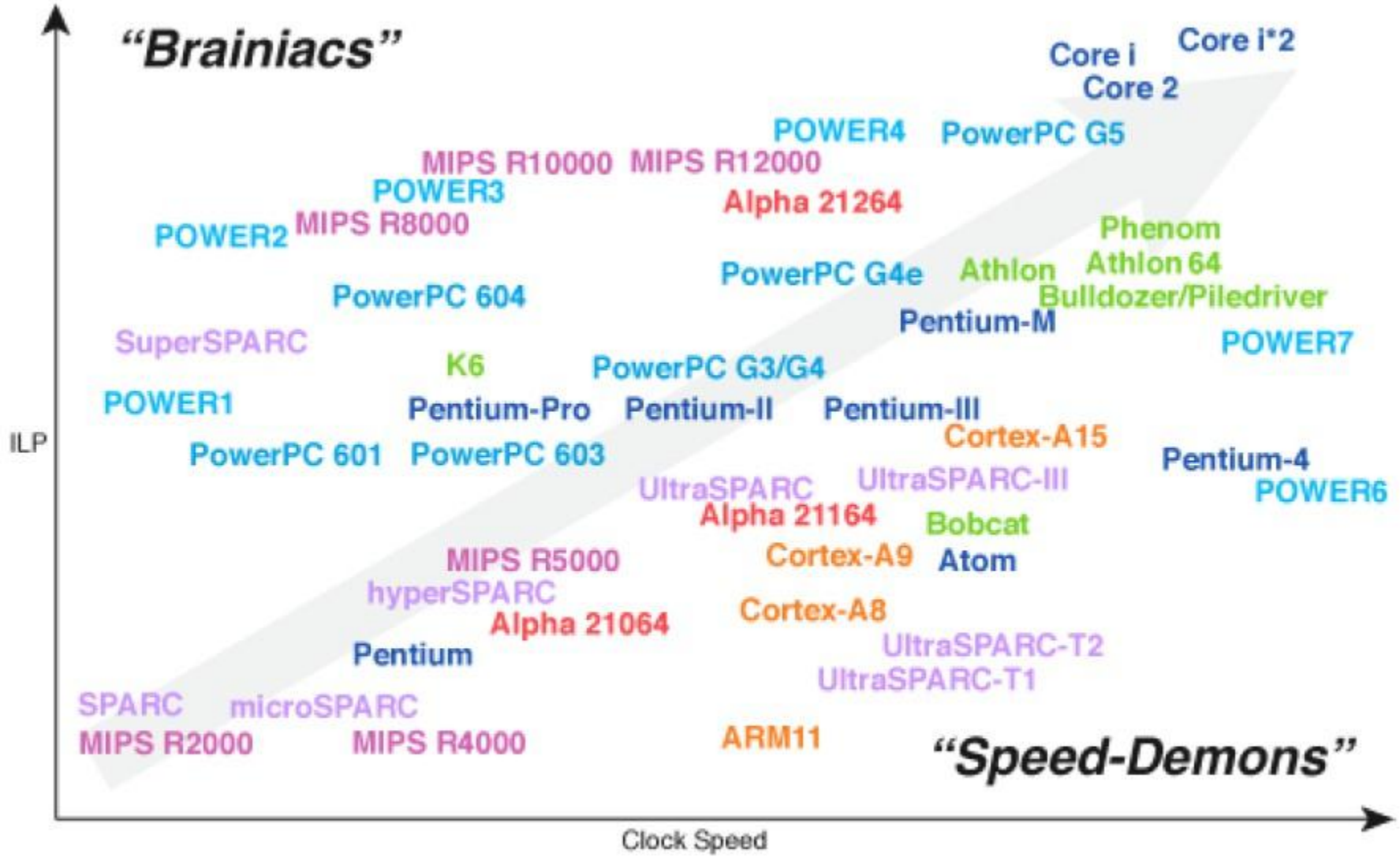


Figure 11 – Brainiacs vs speed-demons.

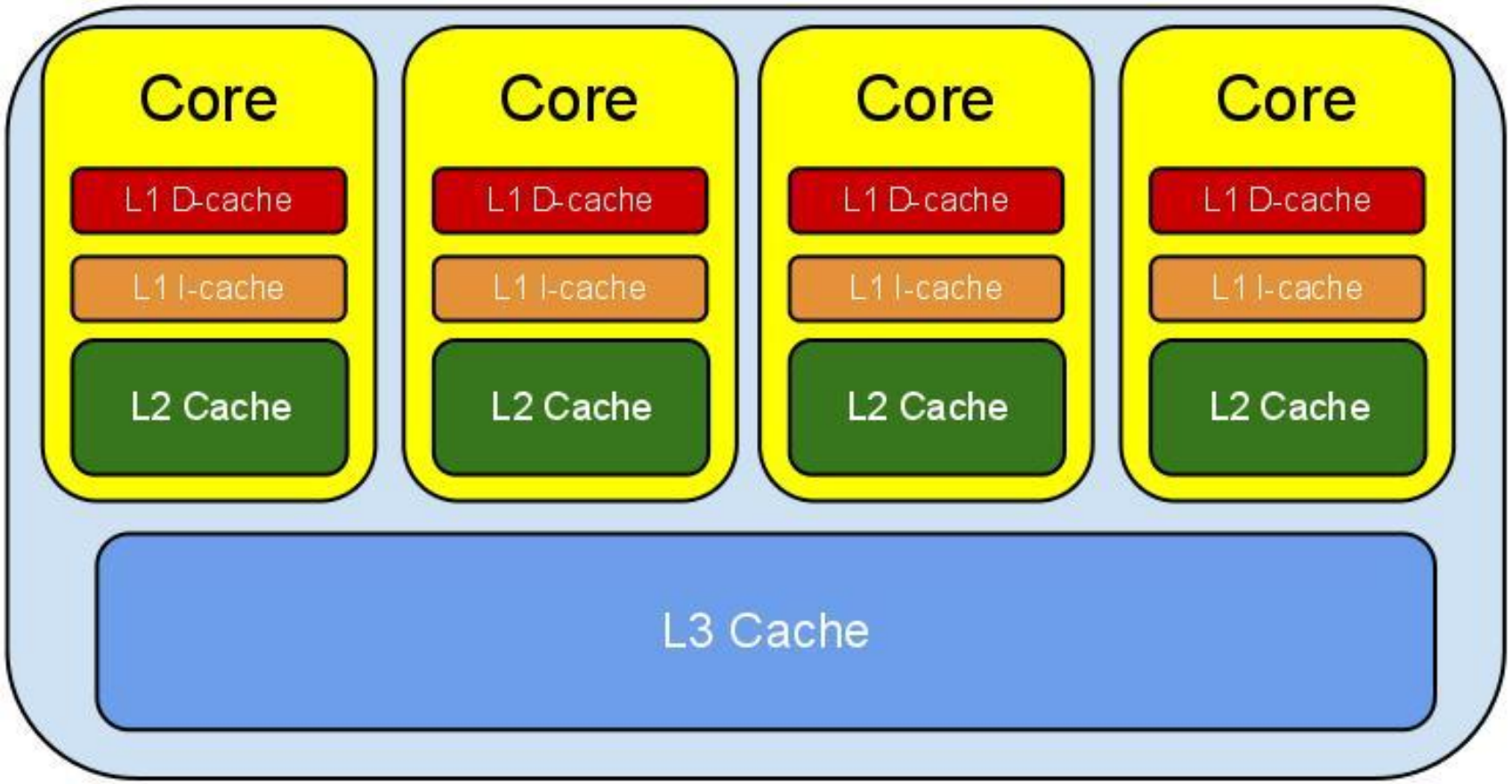
		SPECint95	*SPECfp95*
195 MHz	MIPS R10000	11.0	17.0
400 MHz	Alpha 21164	12.3	17.2
300 MHz	UltraSPARC	12.1	15.5
300 MHz	Pentium-II	11.6	8.8
300 MHz	PowerPC G3	14.8	11.4
135 MHz	POWER2	6.2	17.6

Consistency vs Coherence

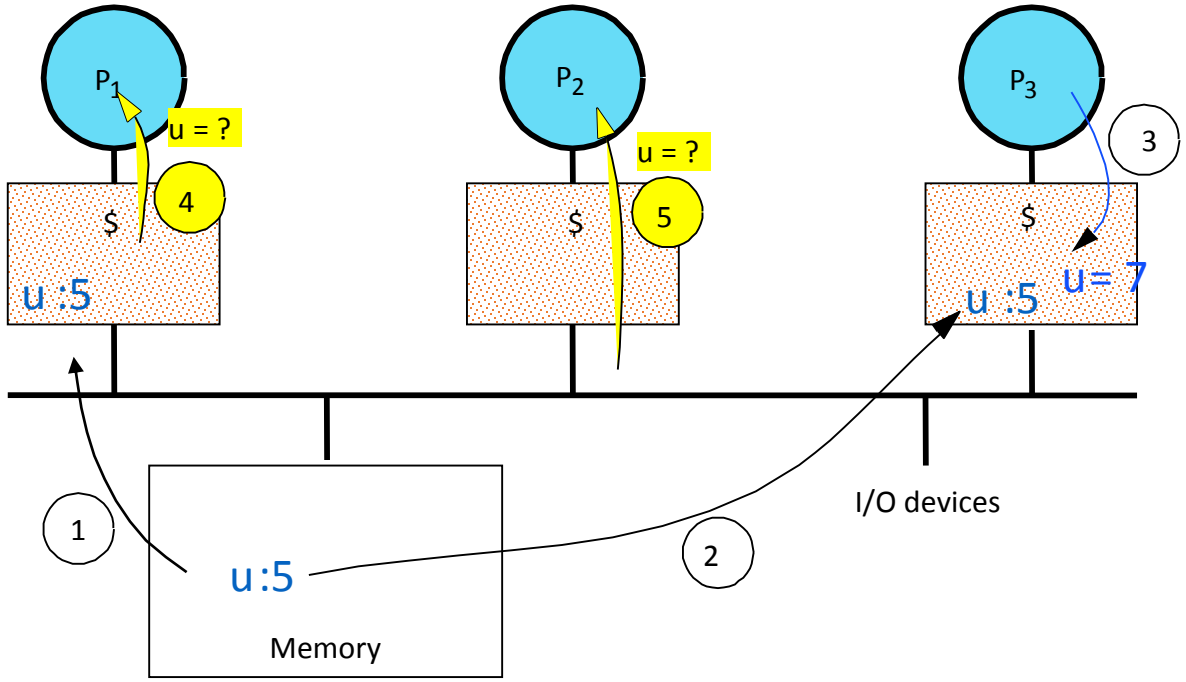
- **Consistency**（Memory Consistency，Memory Model：内存模型）
  - Consistency definitions provide rules about **loads and stores** (or memory reads and writes) and how they act upon memory.
  - 可感知，对程序正确性有影响

- **Coherence**（Cache Coherence：缓存一致性）
  - Coherence seeks to **make the caches of a shared-memory system as functionally invisible** as the caches in a single-core system. Correct coherence ensures that a programmer cannot determine whether and where a system has caches by analyzing the results of loads and stores.
  - 不可感知，CPU内部搞定

CPU Cache


Core
L1 D-cache
L1 I-cache
L2 Cache
Core
L1 D-cache
L1 I-cache
L2 Cache
Core
L1 D-cache
L1 I-cache
L2 Cache
Core
L1 D-cache
L1 I-cache
L2 Cache
L3 Cache

Cache Coherence Problem



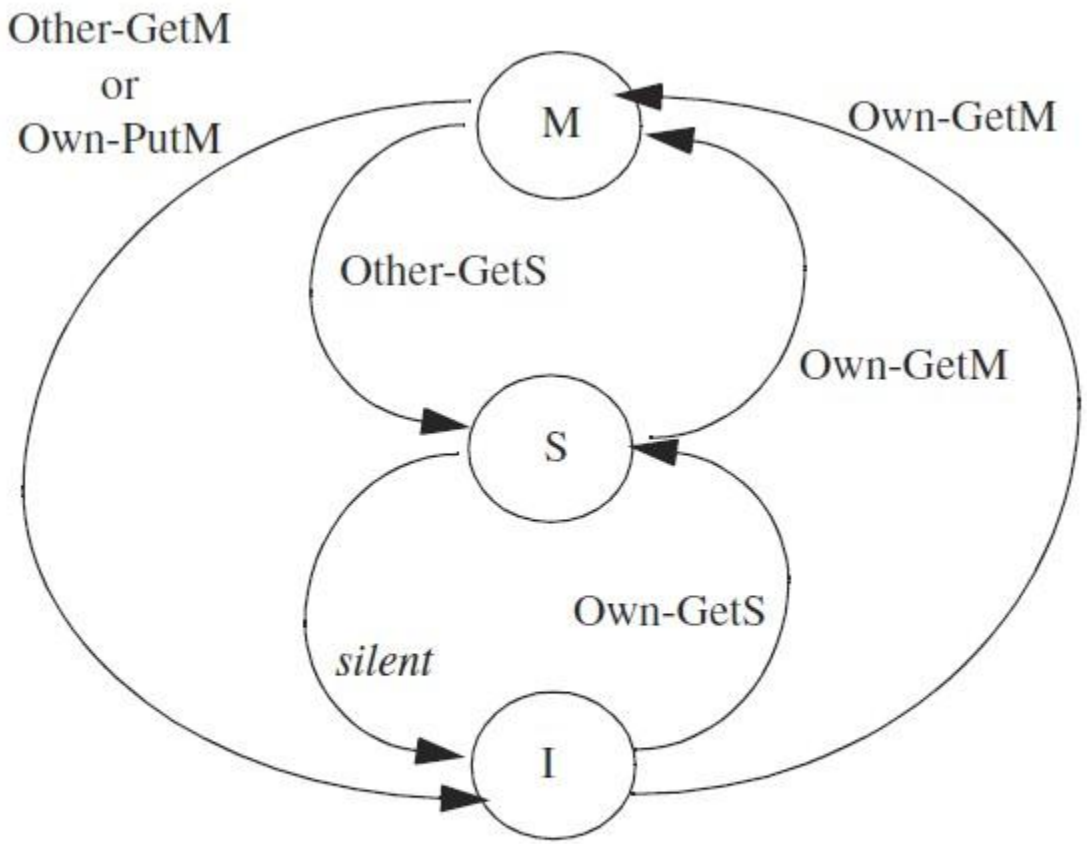


FIGURE 7.1: MSI: Transitions between stable states at cache controller.

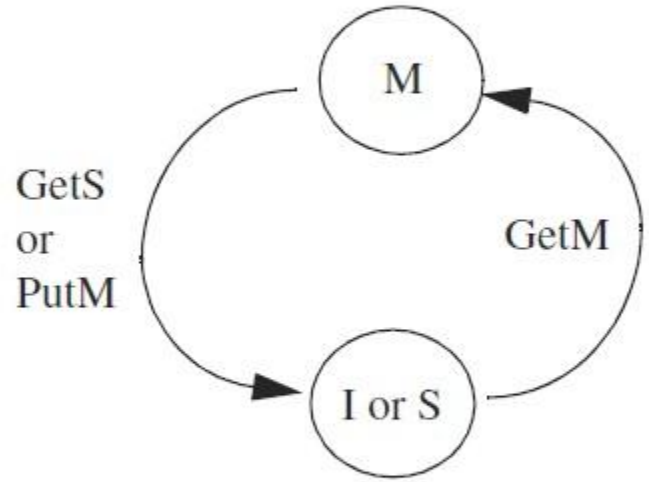


FIGURE 7.2: MSI: Transitions between stable states at memory controller.

TABLE 7.5: Simple Snooping (Atomic Requests, Atomic Transactions): Cache Controller

States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D								
IS^D	stall Load	stall Store	stall Evict				copy data into cache, load hit /S	(A)	(A)	(A)
IM^D	stall Load	stall Store	stall Evict				copy data into cache, store hit /M	(A)	(A)	(A)
S	load hit	issue GetM /SM^D	-/I						-/I	
SM^D	load hit	stall Store	stall Evict				copy data into cache, store hit /M	(A)	(A)	(A)
M	load hit	store hit	issue PutM, send Data to memory /I					send Data to req and memory /S	send Data to req /I	

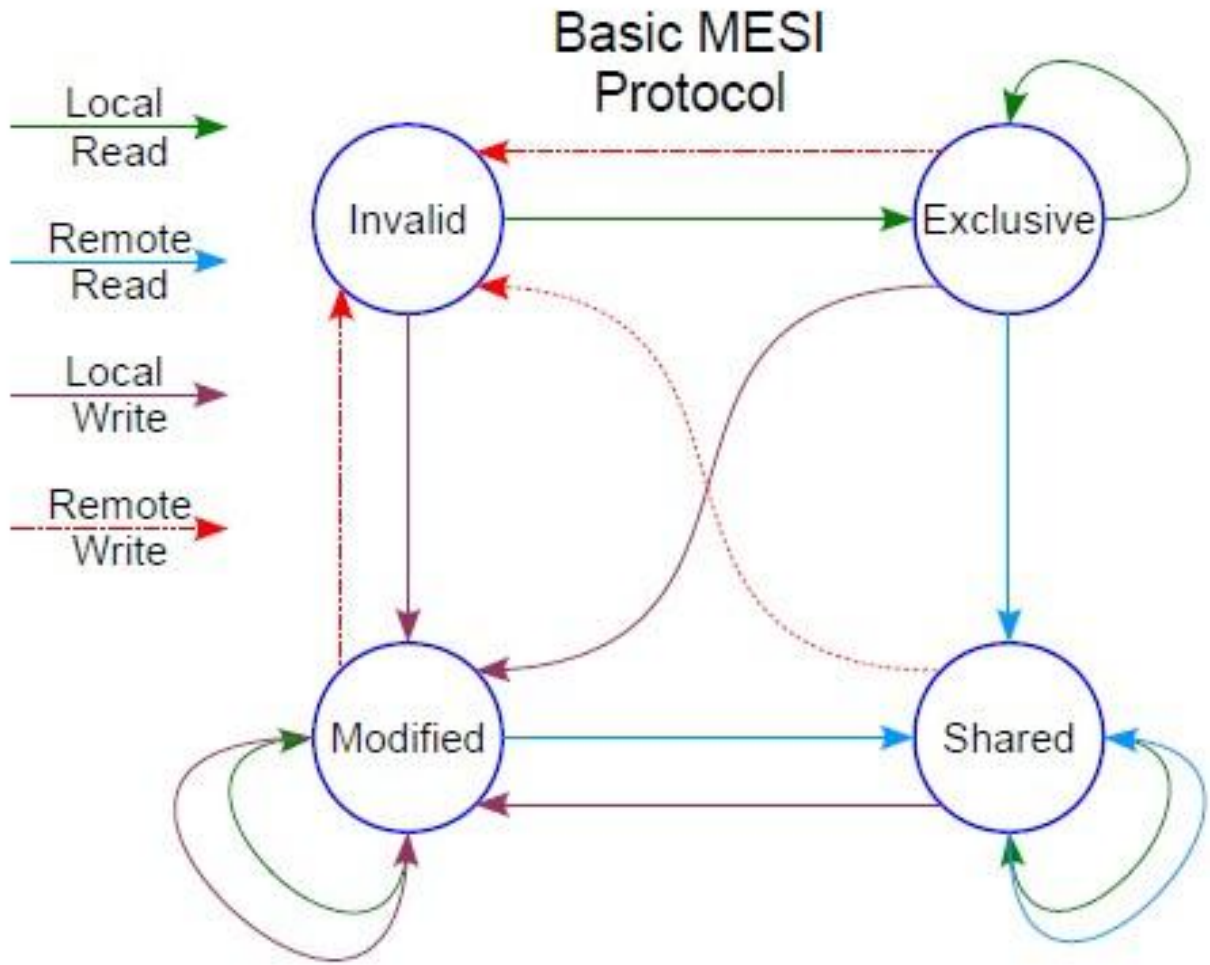


Figure 12: The State Transitions of Cache Blocks in the Basic MESI Cache-Coherence Protocol.

TestCase

- [ccBench]

```
Supported events:
      0 - STORE_ON_MODIFIED
      1 - STORE_ON_MODIFIED_NO_SYNC
      2 - STORE_ON_EXCLUSIVE
      3 - STORE_ON_SHARED
      4 - STORE_ON_OWNED_MINE
      5 - STORE_ON_OWNED
      6 - STORE_ON_INVALID
      7 - LOAD_FROM_MODIFIED
      8 - LOAD_FROM_EXCLUSIVE
      9 - LOAD_FROM_SHARED
     10 - LOAD_FROM_OWNED
     11 - LOAD_FROM_INVALID
     12 - CAS
     13 - FAI
     14 - TAS
     15 - SWAP
     16 - CAS_ON_MODIFIED
     17 - FAI_ON_MODIFIED
     18 - TAS_ON_MODIFIED
     19 - SWAP_ON_MODIFIED
     20 - CAS_ON_SHARED
     21 - FAI_ON_SHARED
     22 - TAS_ON_SHARED
     23 - SWAP_ON_SHARED
     24 - CAS_CONCURRENT
     25 - FAI_ON_INVALID
     26 - LOAD_FROM_L1
     27 - LFENCE
     28 - SFENCE
     29 - MFENCE
     30 - PROFILER
     31 - PAUSE
     32 - NOP
```

```
$./ccbench -e 8 -x 0 -y 8 -t 0
test:    STORE_ON_MODIFIED  / #cores: 2 / #repetitions: 10000 / stride: 2048 (128 kiB)  / fence:  load/full
core1:   0 / core2:   8
[00]  *** Core  0 ********************************************************************************
 ---- statistics:
[00]     avg : 36.3       abs dev : 5.8        std dev : 9.5        num     : 10000
[00]     min : 24.0       (element:     10)    max     : 256.0      (element:   5618)
[00]   0-10% : 3517       ( 35.2%  |  avg:    36.0  |  abs dev:    0.0  |  std dev:    0.0 =   0.0%)
[00]  10-25% : 5471       ( 54.7%  |  avg:    33.0  |  abs dev:    5.0  |  std dev:    5.4 =  16.5%)
[00]  25-50% : 257        (  2.6%  |  avg:    28.8  |  abs dev:    7.7  |  std dev:    9.8 =  34.2%)
[00]  50-75% : 159        (  1.6%  |  avg:    59.8  |  abs dev:    0.3  |  std dev:    0.8 =   1.4%)
[00] 75-100% : 596        (  6.0%  |  avg:    64.9  |  abs dev:    1.7  |  std dev:    9.8 =  15.2%)

[01]  *** Core  1 ********************************************************************************
 ---- statistics:
[01]     avg : 38.1       abs dev : 6.4        std dev : 9.7        num     : 10000
```

```
$./ccbench -e 8 -x 0 -y 1 -t 0
test:    STORE_ON_MODIFIED  / #cores: 2 / #repetitions: 10000 / stride: 2048 (128 kiB)  / fence:  load/full
[00]  *** Core  0 ********************************************************************************
 ---- statistics:
[00]     avg : 224.5      abs dev : 15.3       std dev : 39.3       num     : 10000
[00]     min : 0.0        (element:   4787)    max     : 1004.0     (element:   8309)
[00]   0-10% : 9466       ( 94.7%  |  avg:   232.1  |  abs dev:    2.0  |  std dev:    3.4 =   1.5%)
[00]  10-25% : 131        (  1.3%  |  avg:   236.0  |  abs dev:   22.5  |  std dev:   25.4 =  10.8%)
[00]  25-50% : 0          (  0.0%  |  avg:    -nan  |  abs dev:   -nan  |  std dev:   -nan =  -nan%)
[00]  50-75% : 4          (  0.0%  |  avg:   139.0  |  abs dev:  110.5  |  std dev:  127.6 =  91.8%)
[00] 75-100% : 399        (  4.0%  |  avg:    40.2  |  abs dev:    6.7  |  std dev:   51.6 = 128.3%)

[01]  *** Core  1 ********************************************************************************
 ---- statistics:
[01]     avg : 222.8      abs dev : 15.9       std dev : 39.3       num     : 10000
[01]     min : 0.0        (element:   5903)    max     : 392.0      (element:   5346)
[01]   0-10% : 9434       ( 94.3%  |  avg:   230.9  |  abs dev:    2.9  |  std dev:    3.7 =   1.6%)
[01]  10-25% : 143        (  1.4%  |  avg:   232.9  |  abs dev:   26.3  |  std dev:   28.1 =  12.1%)
[01]  25-50% : 0          (  0.0%  |  avg:    -nan  |  abs dev:   -nan  |  std dev:   -nan =  -nan%)
[01]  50-75% : 2          (  0.0%  |  avg:   210.0  |  abs dev:  146.0  |  std dev:  146.0 =  69.5%)
[01] 75-100% : 421        (  4.2%  |  avg:    37.8  |  abs dev:    2.8  |  std dev:   17.5 =  46.2%)
```

4.8%)
16.4%)
18.5%)
1.3%)
15.5%)

Sequential Consistency

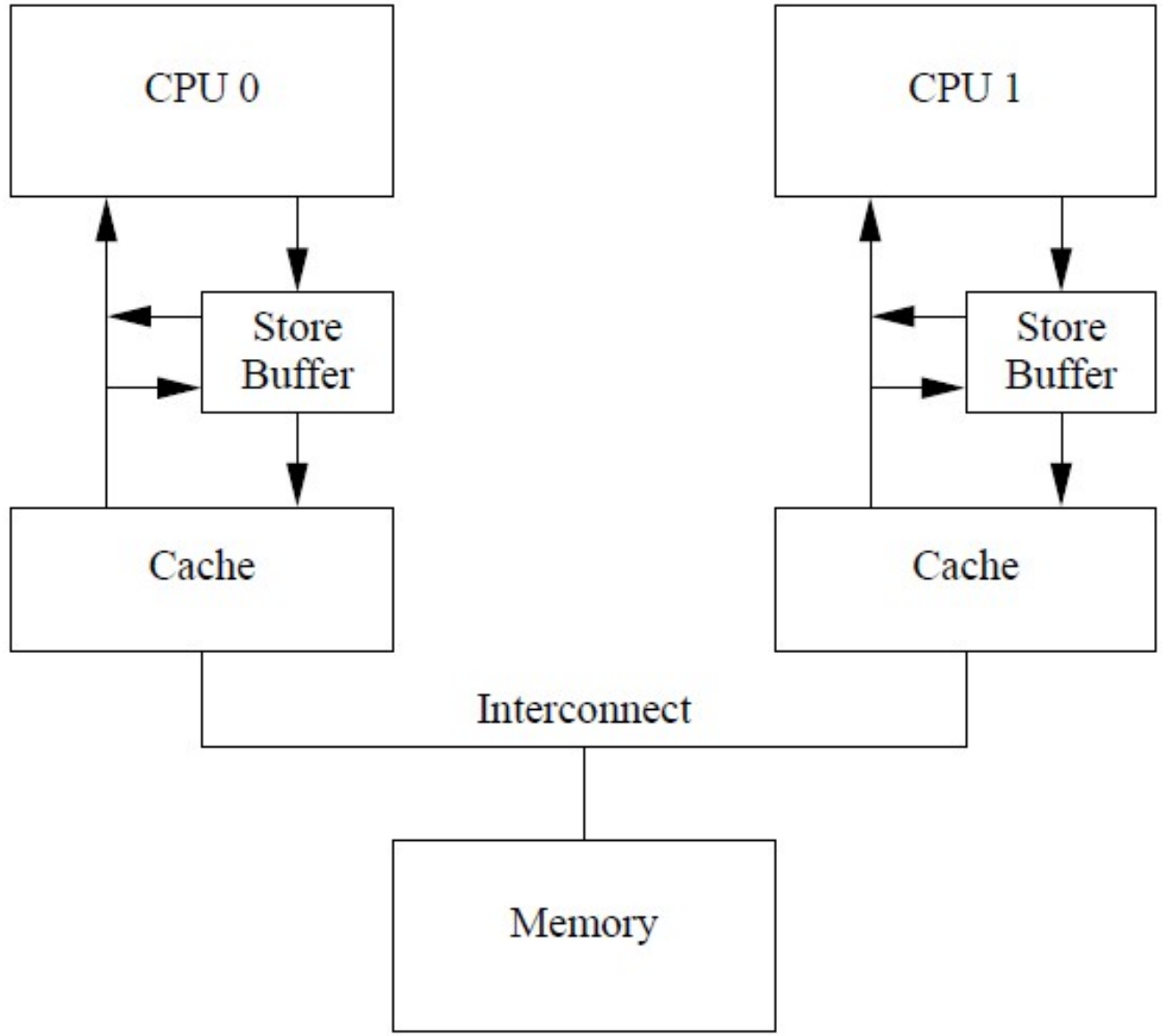
- 大家做一

- SC的缺点？（为什么会有乱序出现）
  - 性能，性能，性能
  - 乱序执行 → 消除Stall

- 有哪些常见的乱序情况？
  - 两类内存操作：Load，Store
  - 4种组合：
    - LoadLoad
    - LoadStore
    - StoreLoad
    - StoreStore

	Loads Reordered After Loads?	Loads Reordered After Stores?	Stores Reordered After Stores?	Stores Reordered After Loads?	Atomic Instructions Reordered With Loads?	Atomic Instructions Reordered With Stores?	Dependent Loads Reordered?	Incoherent Instruction Cache/Pipeline?
Alpha	Y	Y	Y	Y	Y	Y	Y	Y
AMD64	Y			Y				
IA64	Y	Y	Y	Y	Y	Y		Y
(PA-RISC)	Y	Y	Y	Y				
PA-RISC CPUs								
POWER	Y	Y	Y	Y	Y	Y		Y
SPARC RMO	Y	Y	Y	Y	Y	Y		Y
(SPARC PSO)			Y	Y		Y		Y
SPARC TSO				Y				Y
x86	Y	Y		Y				Y
(x86 OOStore)	Y	Y	Y	Y				Y
zSeries				Y				Y

Table 5: Summary of Memory Ordering


CPU 0
CPU 1
Store Buffer
Store Buffer
Cache
Cache
Interconnect
Memory

WEAK

STRONG

Really weak

<

Weak with data dependency ordering

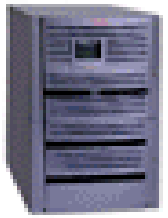
<

Usually strong
(implicit acquire/ release & TSO, usually)

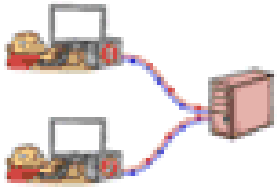
<

Sequentially consistent

DEC Alpha



C/C++11 low-level atomics



Source control analogy

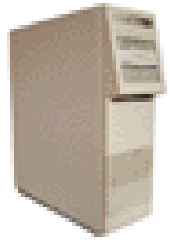
ARM

PowerPC

x86/64

SPARC TSO

dual 386 (circa 1989)



Java `volatile`

C/C++11 default atomics

Or, run on a single core without optimization

Test Case

- ordering.zip
  - LoadLoad
  - LoadStore
  - StoreLoad
  - StoreStore

```
$./ordering 1

 StoreLoad Reorder Tests.
312 storeload reorders detected after 10000 iterations
1203 storeload reorders detected after 20000 iterations
2049 storeload reorders detected after 30000 iterations
2822 storeload reorders detected after 40000 iterations
3787 storeload reorders detected after 50000 iterations
4491 storeload reorders detected after 60000 iterations
4492 storeload reorders detected after 70000 iterations
4495 storeload reorders detected after 80000 iterations
4496 storeload reorders detected after 90000 iterations
4497 storel
```

```
$./ordering 2
0 loadload & storestore reo
0 loadload & storestore reo
0 loadload & storestore reo
0 loadload & storestore reo
0 loadload & storestore reorders detected after 50000 iterations
0 loadload & storestore reorders detected after 60000 iterations
0 loadload & storestore reorders detected after 70000 iterations
0 loadload & storestore reorders detected after 80000 iterations
0 loadload & storestore reorders detected after 90000 iterations
```

```
$./ordering 3
0 loadstore reorders detected after 10000 iterations
0 loadstore reorders detected after 20000 iterations
0 loadstore reorders detected after 30000 iterations
0 loadstore reorders detected after 40000 iterations
0 loadstore reorders detected after 50000 iterations
0 loadstore reorders detected after 60000 iterations
0 loadstore reorders detected after 70000 iterations
0 loadstore reorders detected after 80000 iterations
0 loadstore reorders detected after 90000 iterations
```

参考资料

- **Gallery of Processor Cache Effects**
- HyperThreading Technology in the NetBurst MicroArchitecture
- Intel's Sandy Bridge Microarchitecture
- **Memory Barriers a Hardware View for Software Hackers**
- **Modern Microprocessors - A 90 Minute Guide!**
- SMP Primer for Android
- **The Architecture of the Nehalem Processor and Nehalem-EP SMP Platforms**
- **Processor Microarchitecture an Implementation Perspective**
- **A Primer on Memory Consistency and Cache Coherence**
- **Everything You Always Wanted to Know About Synchronization but Were Afraid to Ask**
- **Jeff Preshing’s Blog**
- **Martin Thompson’s Blog**
- **CPU Cache and Memory Ordering**
- **… …**

Questions?