# Intel® Processor Micro-architecture – Core®

Intel® Software College

# Intel® Core™ Microarchitecture

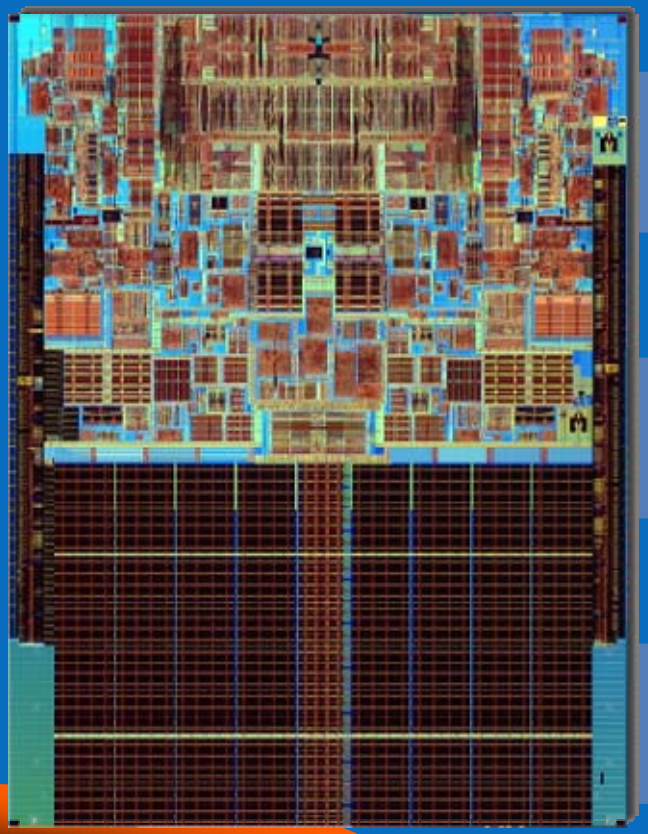
Intel® NetBurst®

Xeon® inside™ Pentium® D inside™

+ New Innovations

Pentium® M inside™ Core™ Duo inside™

Mobile Microarchitecture

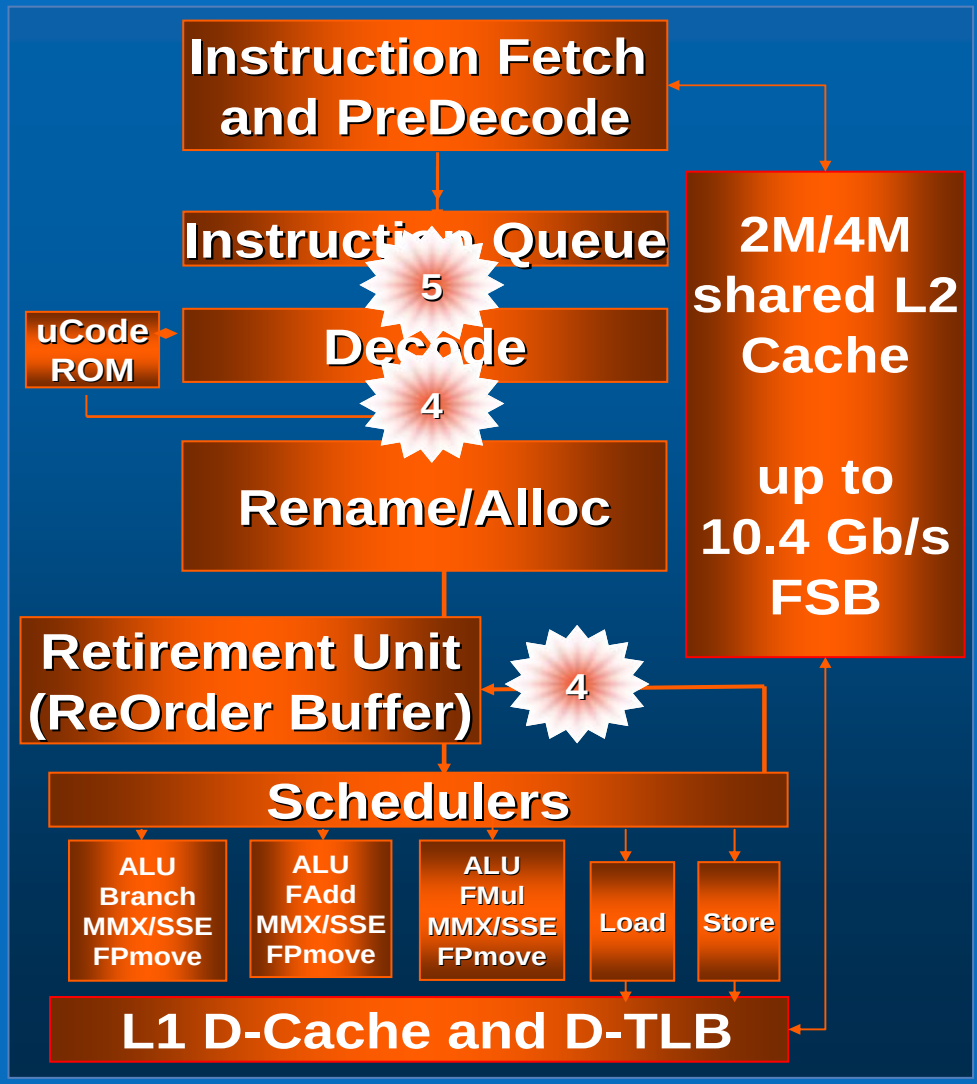




# Intel® Core® Micro-architecture Notable Features

## Intel® Wide Dynamic Execution

- 14-stage efficient pipeline
  - Wider execution path
  - Advanced branch prediction
  - Macro-fusion
    - Roughly ~15% of all instructions are conditional branches
    - Macro-fusion fuses a comparison and jump to reduce micro-ops running down the pipeline
  - Micro-fusion
    - Merges the load and operation micro-ops into one macro-op
  - Stack pointer tracker
    - ESP tracks the stack
    - This pointer allows push/pops to work returning the correct values
- 64-Bit Support
  - Merom, Conroe, and Woodcrest support EM64T



# Intel® Core® Micro-architecture Notable Features (cont.)

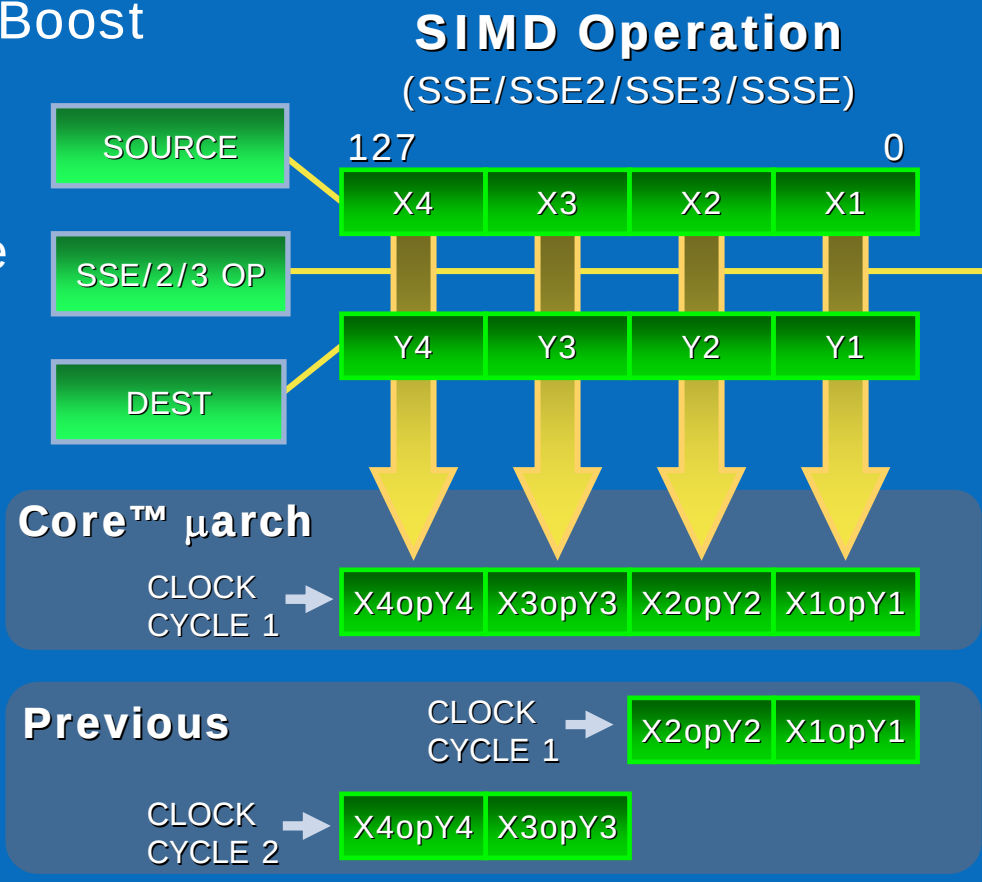
## Intel® Advanced Smart Cache

- Multi-core optimization
  - Shared between the two cores
  - Advanced Transfer Cache architecture
  - Reduced bus traffic
  - Both cores have full access to the entire cache
  - Dynamic Cache sizing
- Shared second level (L2) 2MB 8-way or 4MB 16-way instruction and data cache
- Higher bandwidth from the L2 cache to the core
  - ~14 clock latency and 2 clock throughput

# Intel® Core® Micro-architecture Notable Features (cont.)

## Intel® Advanced Digital Media Boost

- Single Cycle SIMD Operation
  - 8 Single Precision Flops/cycle
  - 4 Double Precision Flops/cycle
- Wide Operations
  - 128-bit packed Add
  - 128-bit packed Multiply
  - 128-bit packed Load
  - 128-bit packed Store



# Key Terms

CISC vs. RISC

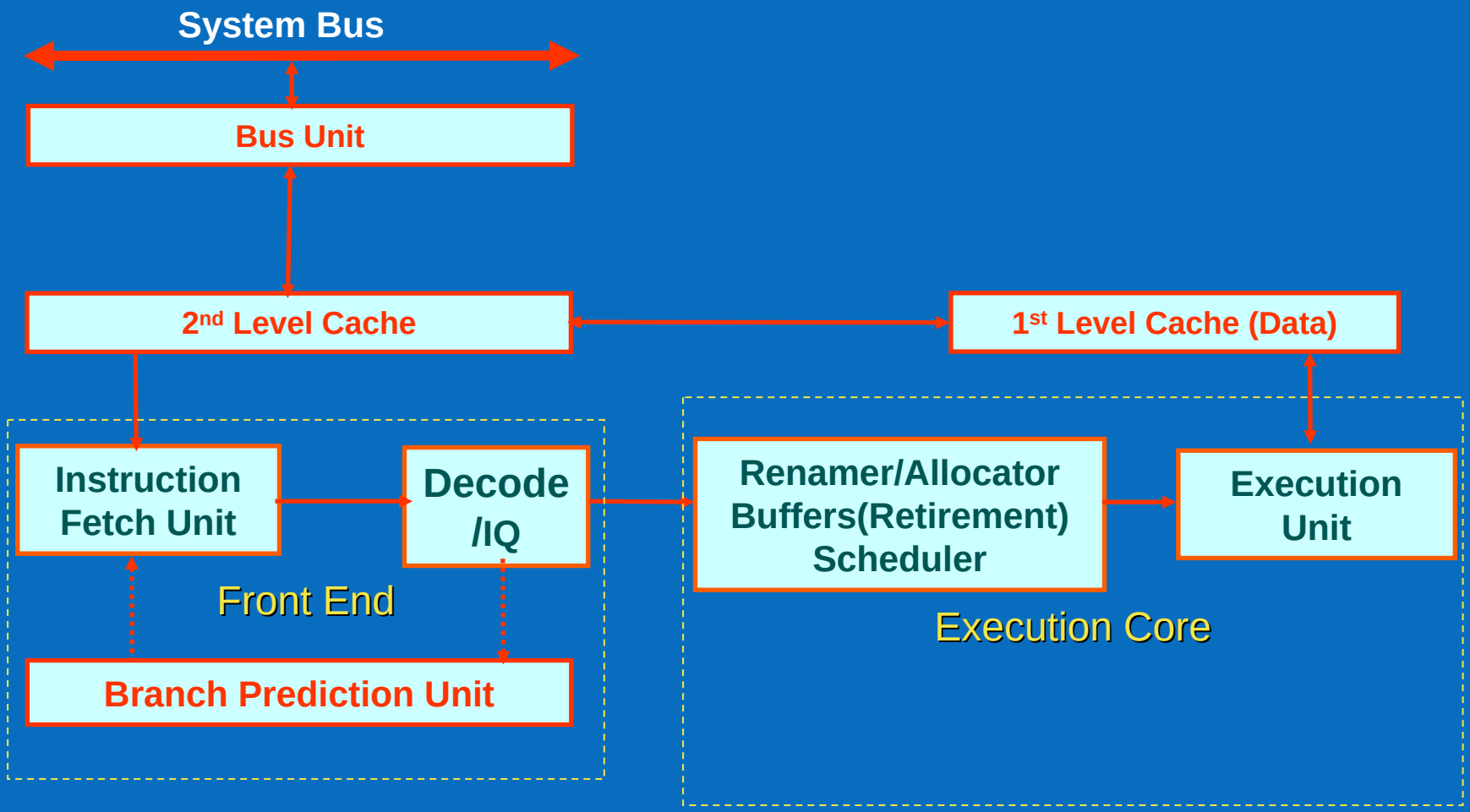
Super-scalar

Out Of Order vs. In Order

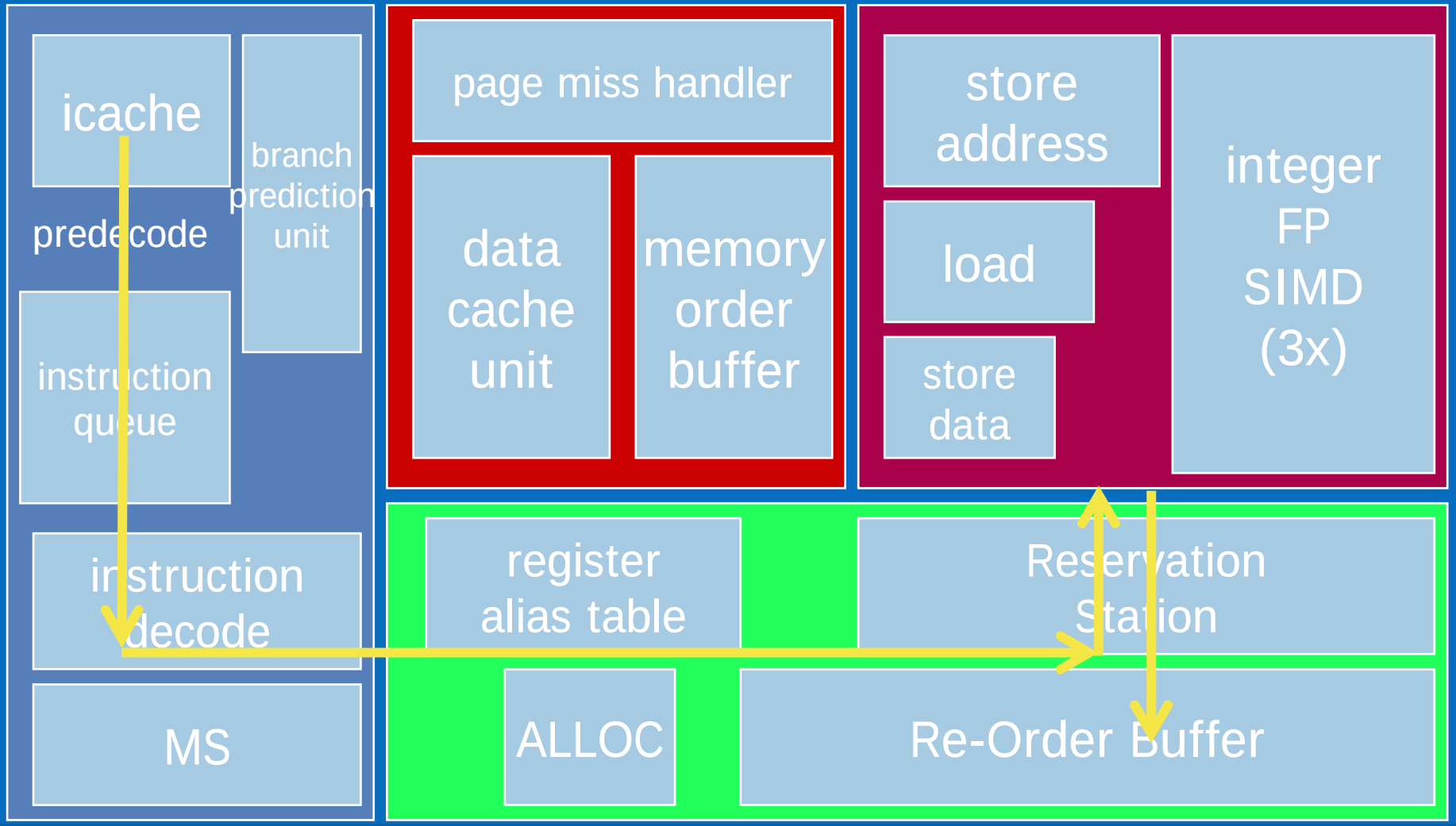
Architecture vs. Micro-architecture

- Intel Architectures
  - IA32/X86
  - Intel® 64
  - IA64
- Historical Micro-architectures
  - P6 (Pentium Pro, Pentium II, Pentium III)
  - NetBurst (Pentium 4)
  - Mobile (Centrino platforms)

# Intel® Core® Micro-architecture Overview



# Intel® Core® Micro-architecture Drill-down



# Core® Micro-architecture Front End

Instruction preparation before executed

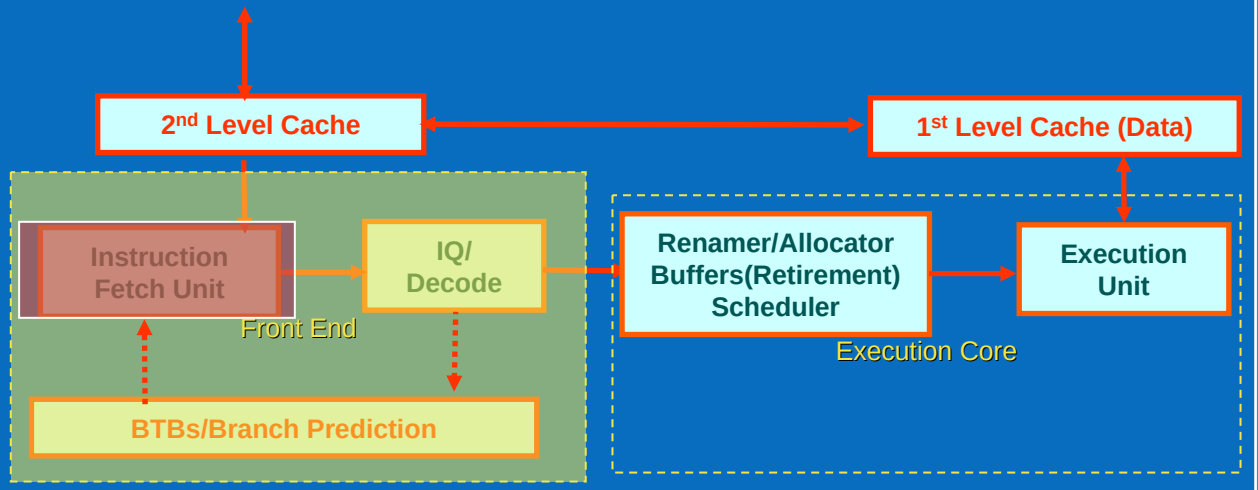
- Instruction Fetch Unit
- Instruction Queue
- Instruction Decode Unit
- Branch Prediction Unit

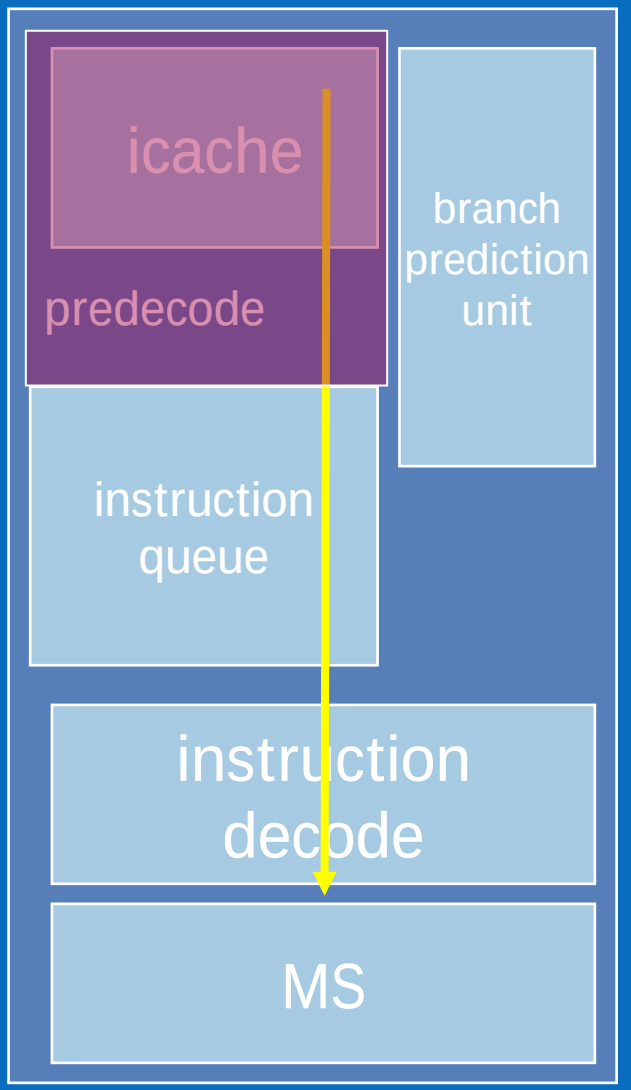
# Instruction Fetch Unit

Prefetches instructions that are likely to be executed

Caches frequently-used instructions

Predecodes and Buffers instructions



# Instruction Fetch Unit (cont.)

I-Cache (Instruction Cache)

- 32 KBytes / 8-way / 64-byte line
- 16 aligned bytes fetched per cycle

ITLB (Instruction Translation Lookaside Buffer)

- 128 4k pages, 8 2M pages

Instruction Prefetcher

- 16-byte aligned lookup through the ITLB into the instruction cache and instruction prefetch buffers

Instruction Pre-decoder

- Instruction Length Decode (predecode)
  - Avoid Length Changing Prefix, for example
    - The REX (EM64T) prefix (4xH) is not an LCP

Avoid in loop:

MOV dx, 1234h

# Instruction Queue

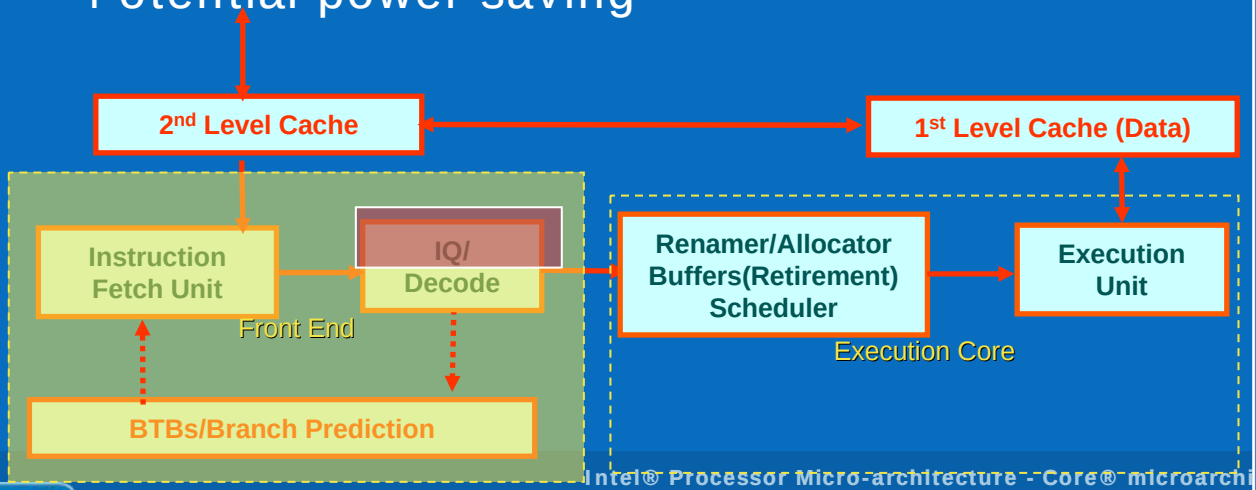
Buffer between instruction pre-decode unit and decoder

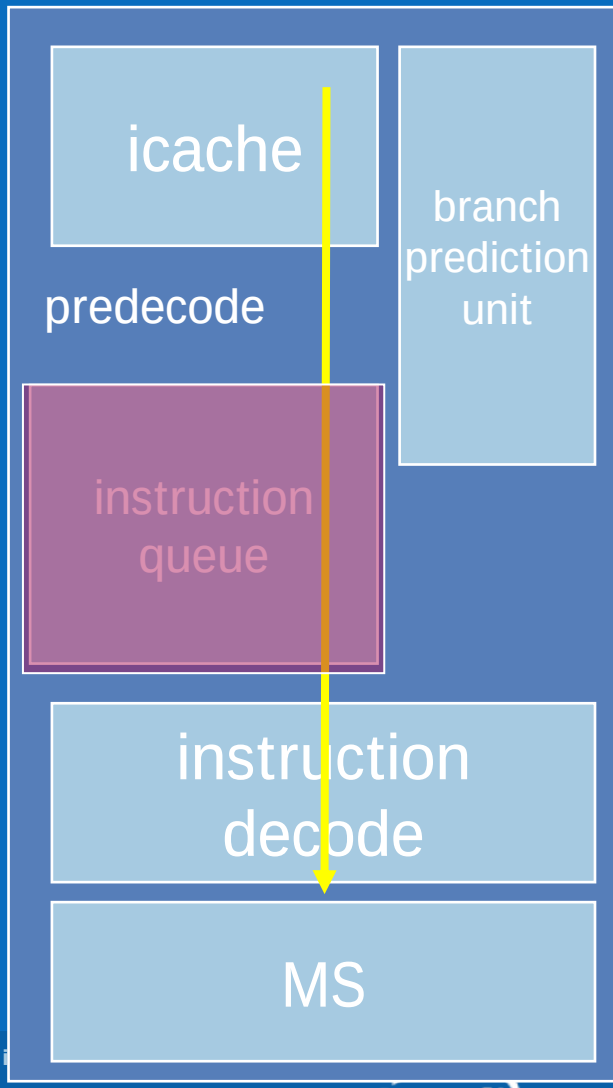
- up to six predecoded instructions written per cycle
- 18 Instructions contained in IQ
- up to 5 Instructions read from IQ

Potential Loop cache

Loop Stream Detector (LSD) support

- Re-use of decoded instruction
- Potential power saving

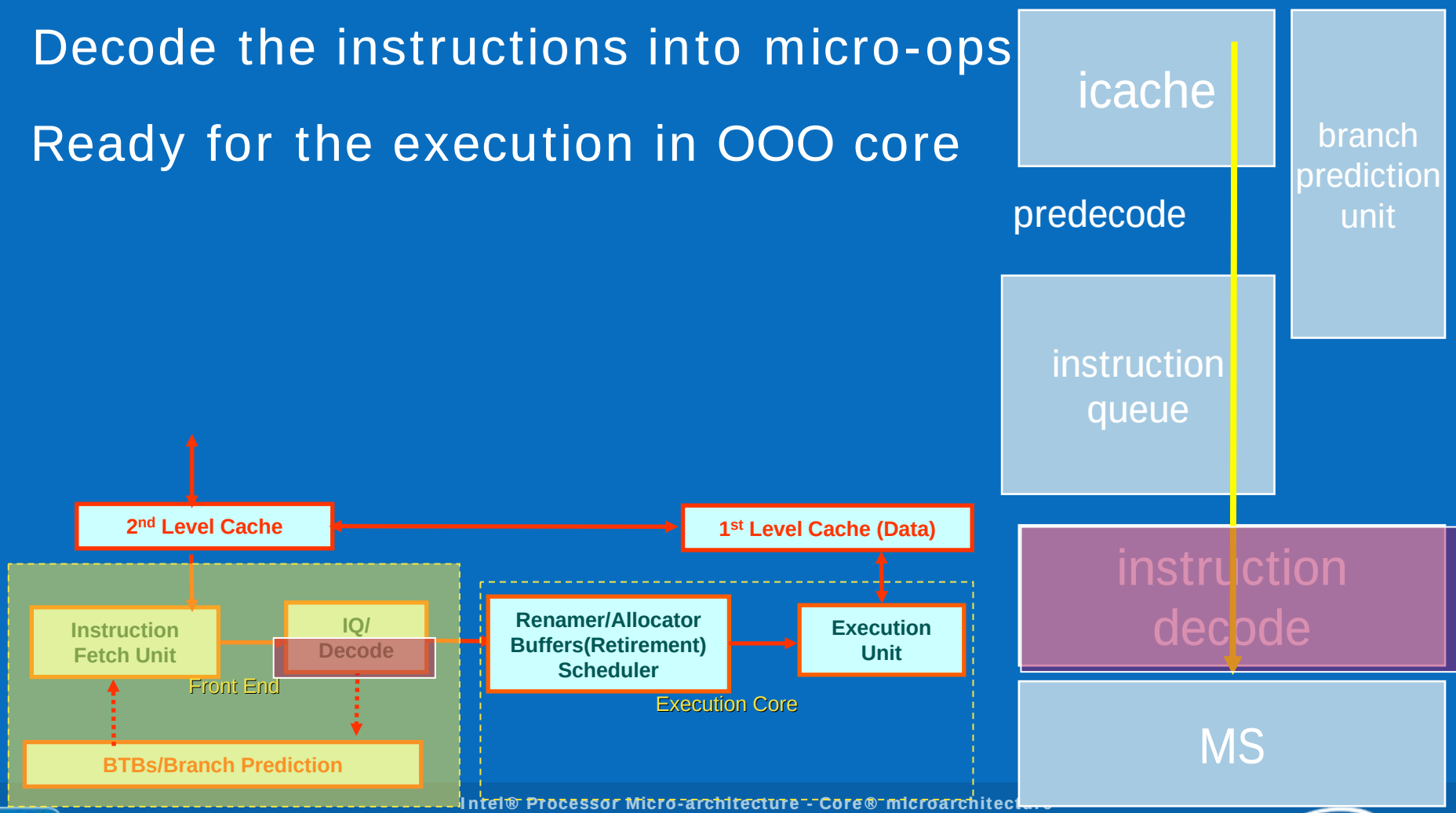


# Instruction Decode

Decode the instructions into micro-ops

Ready for the execution in OOO core



# Instruction Decode / Decoders

Instructions converted to micro-ops (uops)

- 1-uop includes load+op, stores, indirect jump, RET…

4 decoders:1 “large” and 3 “small”

- All decoders handle “simple” 1-uop instructions
- One large decoder handles instructions up to 4 uops

All decoder working in parallel

- Four(+) instructions / cycle

Micro-Sequencer takes over for long flows (handling instruction contains 2~4 uops, uCodeRom handles more complex)

# Code Sequence in Front End

these instructions took more than one fetch as they are 22 bytes

IQ buffers them together

all instructions are decodable by all decoders

CMP and adjacent JCC are “fused” into a single uop. up to 5 instructions decoded per cycle

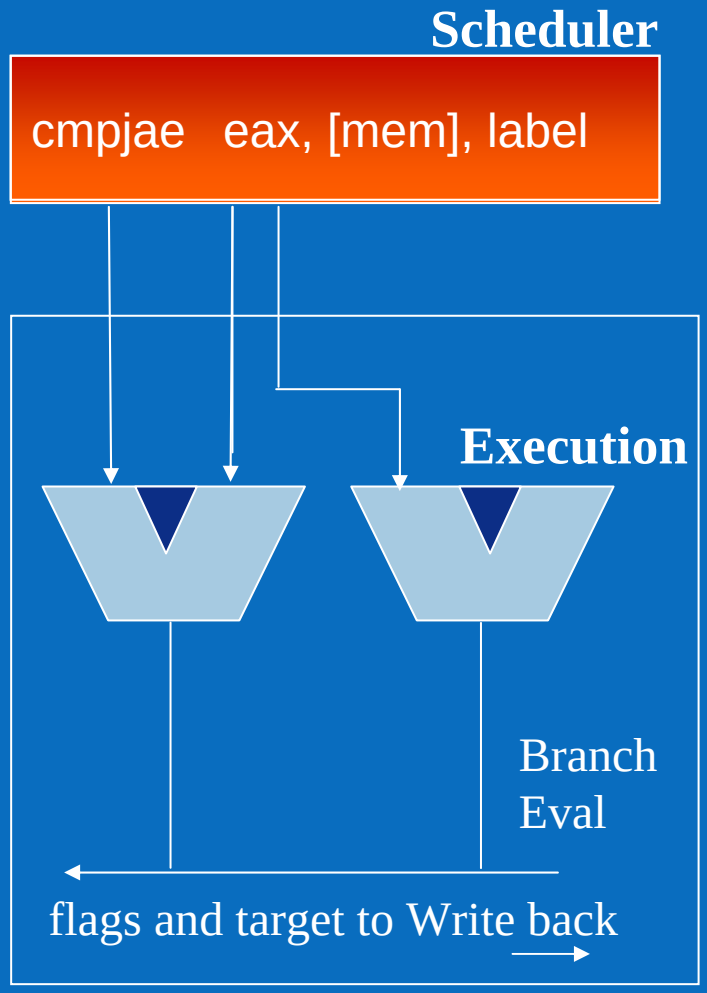
# Instruction Decode / Macro - Fusion

Roughly ~15% of all instructions are conditional branches.

Macro-fusion merges two instructions into a single micro-op, as if the two instructions were a single long instruction.

Enhanced Arithmetic Logic Unit (ALU) for macro-fusion. Each macro-fused instruction executes with a single dispatch.

Not supported in EM64T long mode



# Instruction Decode / Macro-Fusion Absent

Read four instructions from Instruction Queue

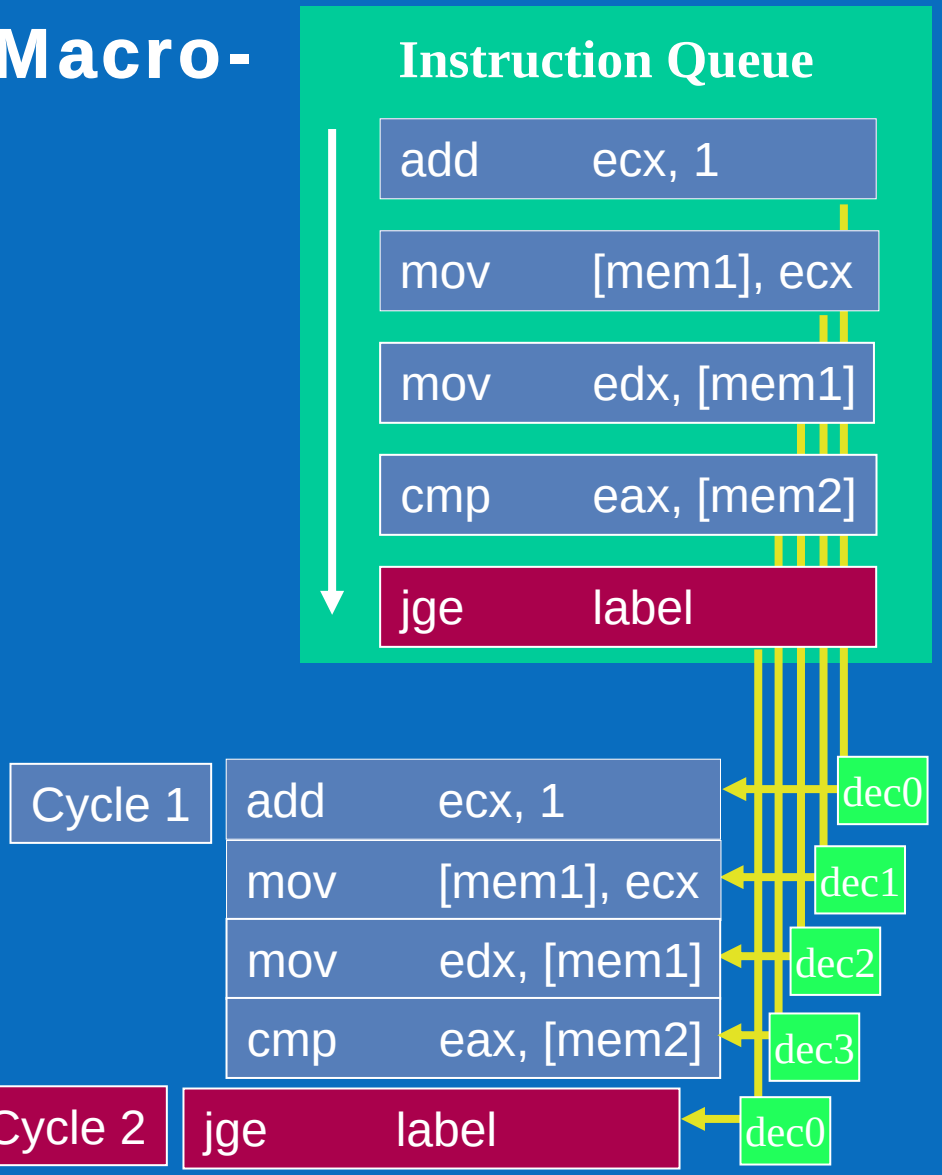
Each instruction gets decoded into separate uops

Enabling Example

```
for (int i=0; i<100000; i++) {

        ...

}
```



# Instruction Decode / Macro-Fusion Presented

Read five Instructions from Instruction Queue

Send fusable pair to single decoder

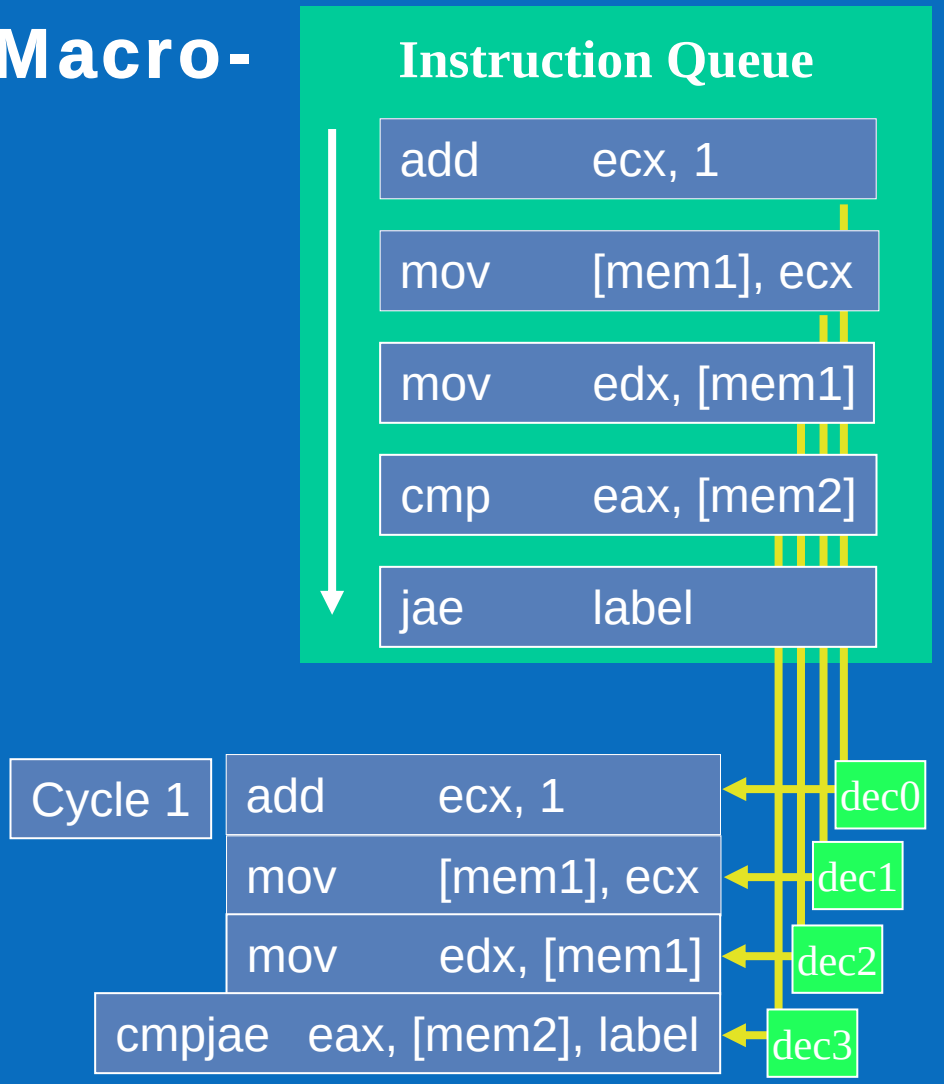
Single uop represents two instructions

Enabling Example

```
for (unsigned int i=0;
i<100000; i++) {

        ...

}
```



# Instruction Decode / Macro – Fusion (cont.)

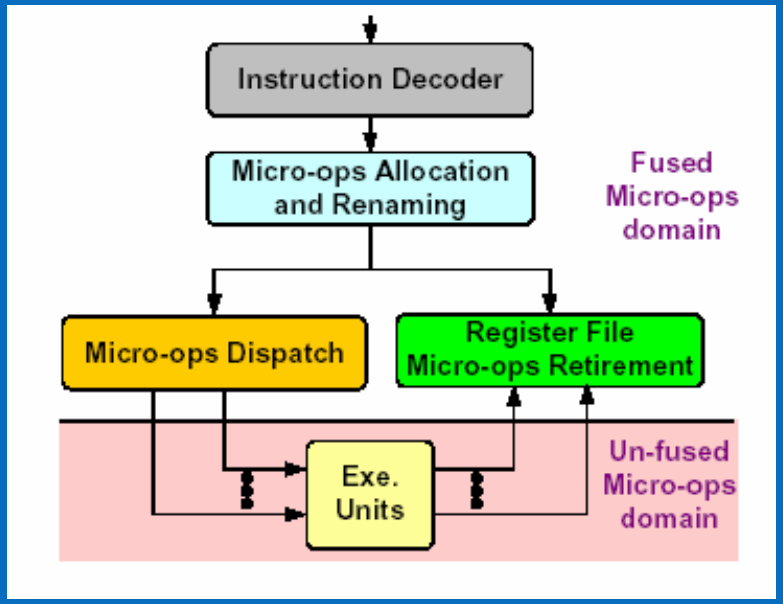
Benefits

- Reduces latency
- Increased renaming
- Increased retire bandwidth
- Increased virtual storage
- Power savings

**Enabling Greater Performance & Efficiency**

# Instruction Decode / Micro-Op Fusion

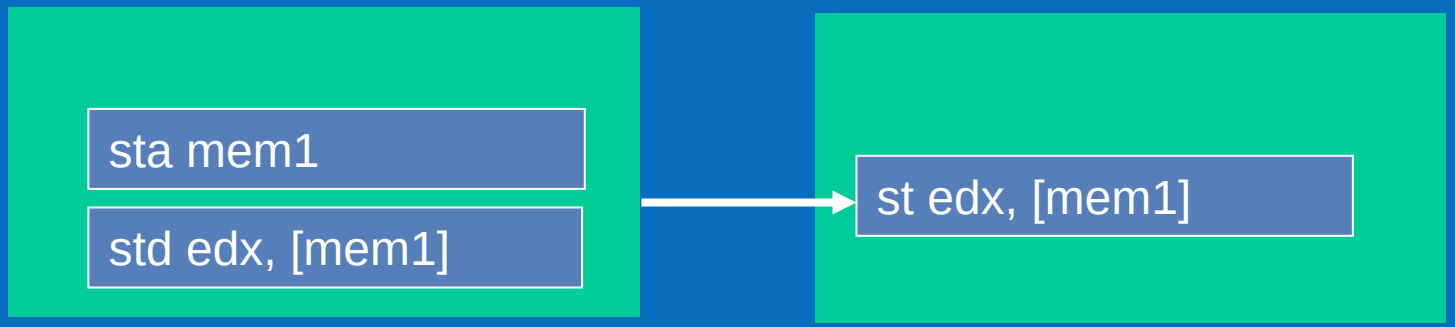
Frequent pairs of micro-operations derived from the same Macro Instruction can be fused into a single micro-operation




**Micro-op fusion effectively widens the pipeline**

# Instruction Decode / Micro-Fusion (cont.)

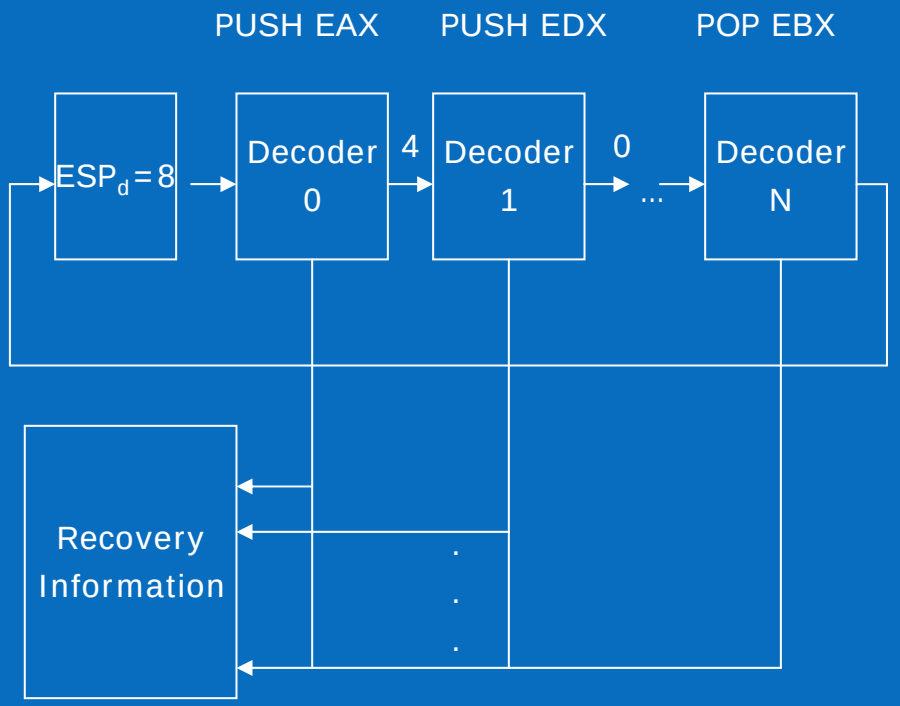
u-ops of a Store “mov edx, [mem1]”



# Instruction Decode / Stack Pointer Tracker (Extended Stack Pointer folding)

ESP is calculated by dedicate logic

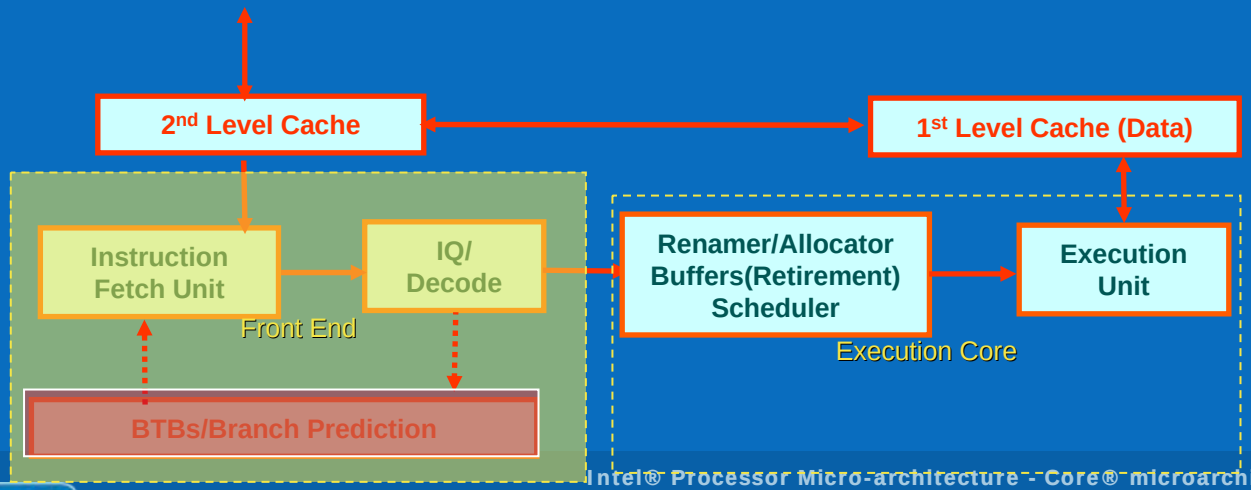
- No explicit Micro-Ops updating ESP
- Micro-Ops saving
- Power saving

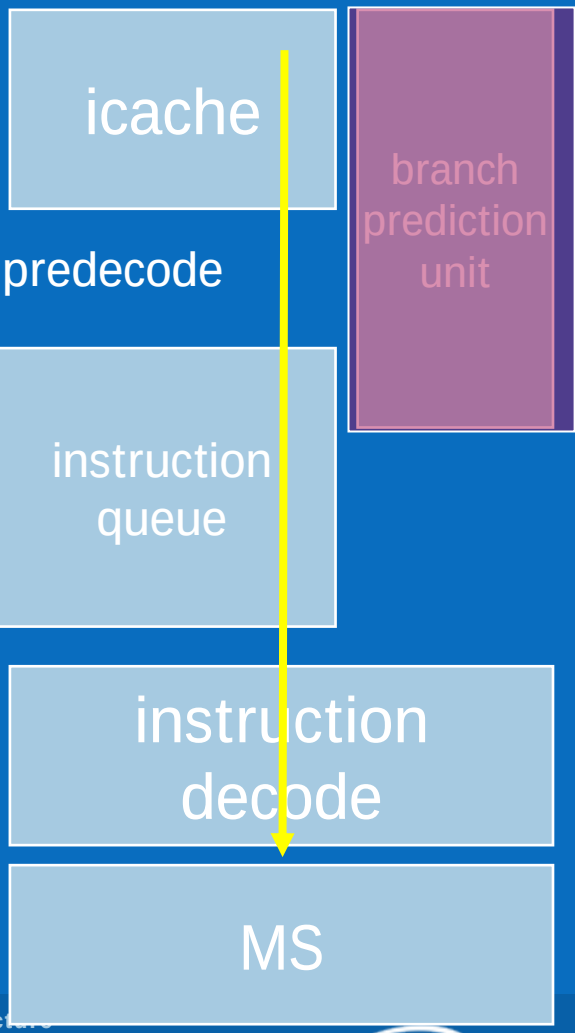


# Branch Prediction Unit

Allow executing instructions long before the branch outcome is decided

- Superset of Prescott / Pentium-M features
- One taken branch every other clock
- Branch predictions for 32 bytes at a time, twice the width of the fetch engine



# Branch Prediction Unit (cont.)

16-entry Return Stack Buffer (RSB)

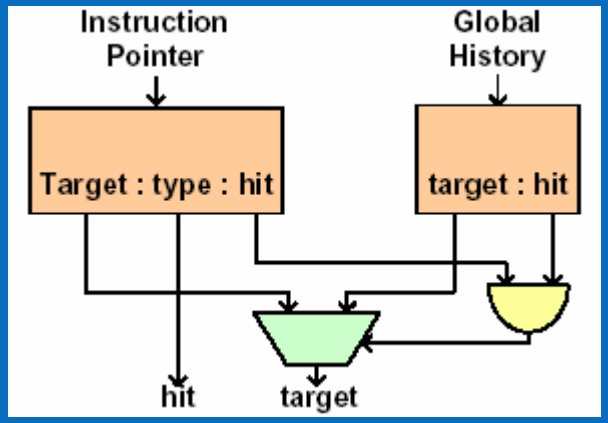
Front end queuing of BPU lookups

Type of predictions

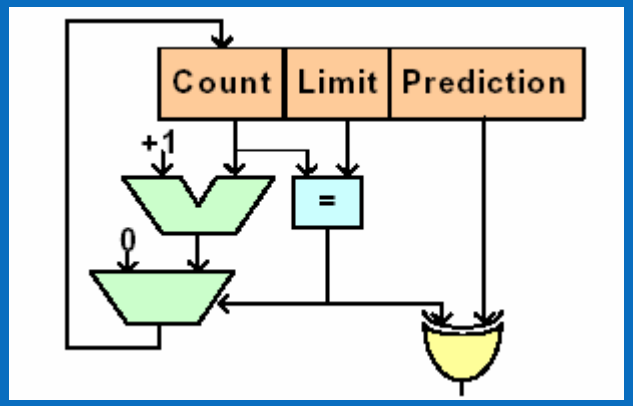
- Direct Calls and Jumps
- Indirect Calls and Jumps
- Conditional branches

# Branch Prediction Improvements

Intel® Pentium® 4 Processor branch prediction PLUS the following two improvements:




**Indirect Branch Predictor**



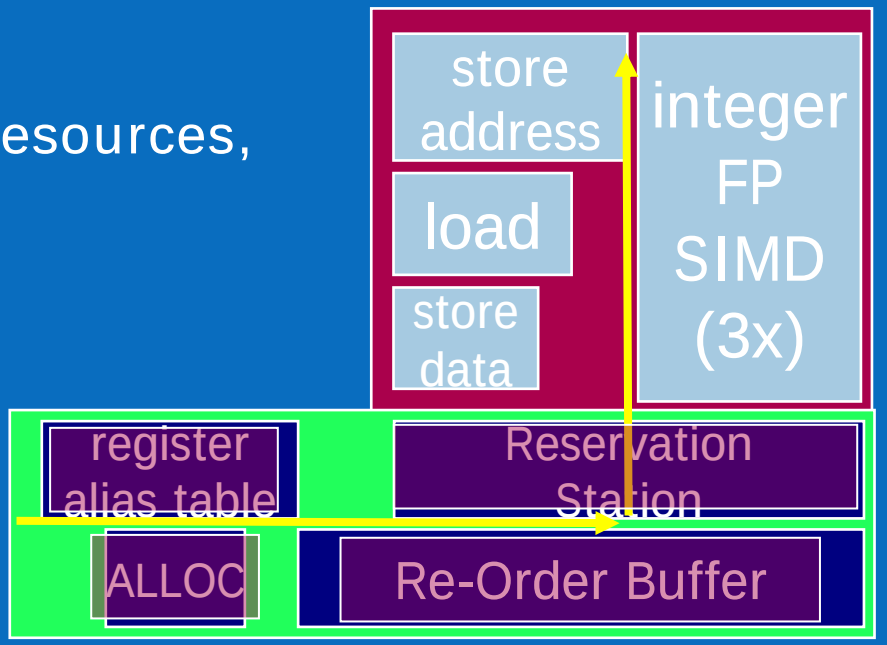

**Loop Detector**

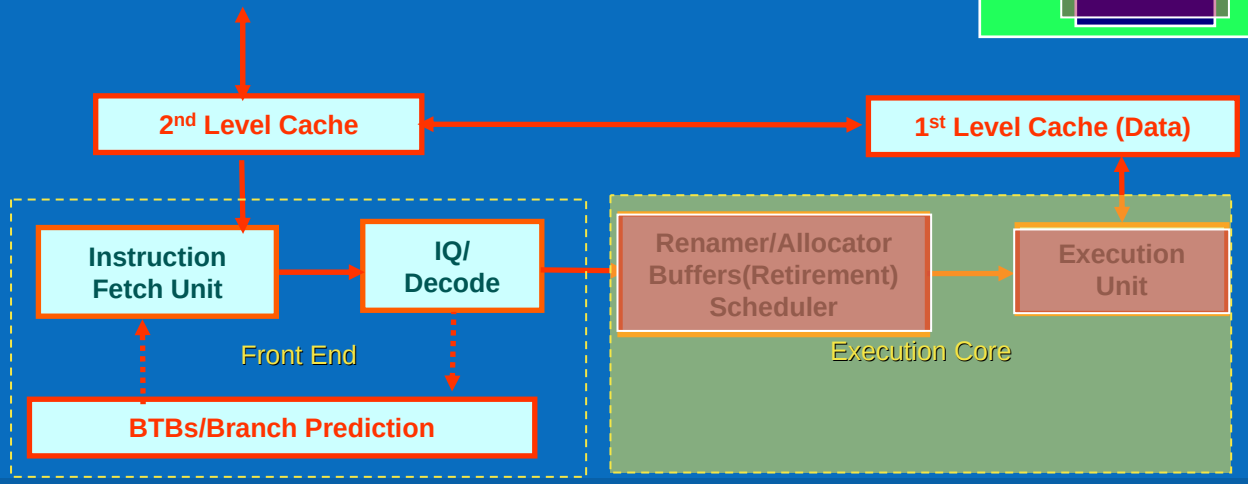
**Branch miss-predictions reduced by >20%**

# Core® Micro-architecture Execution Core

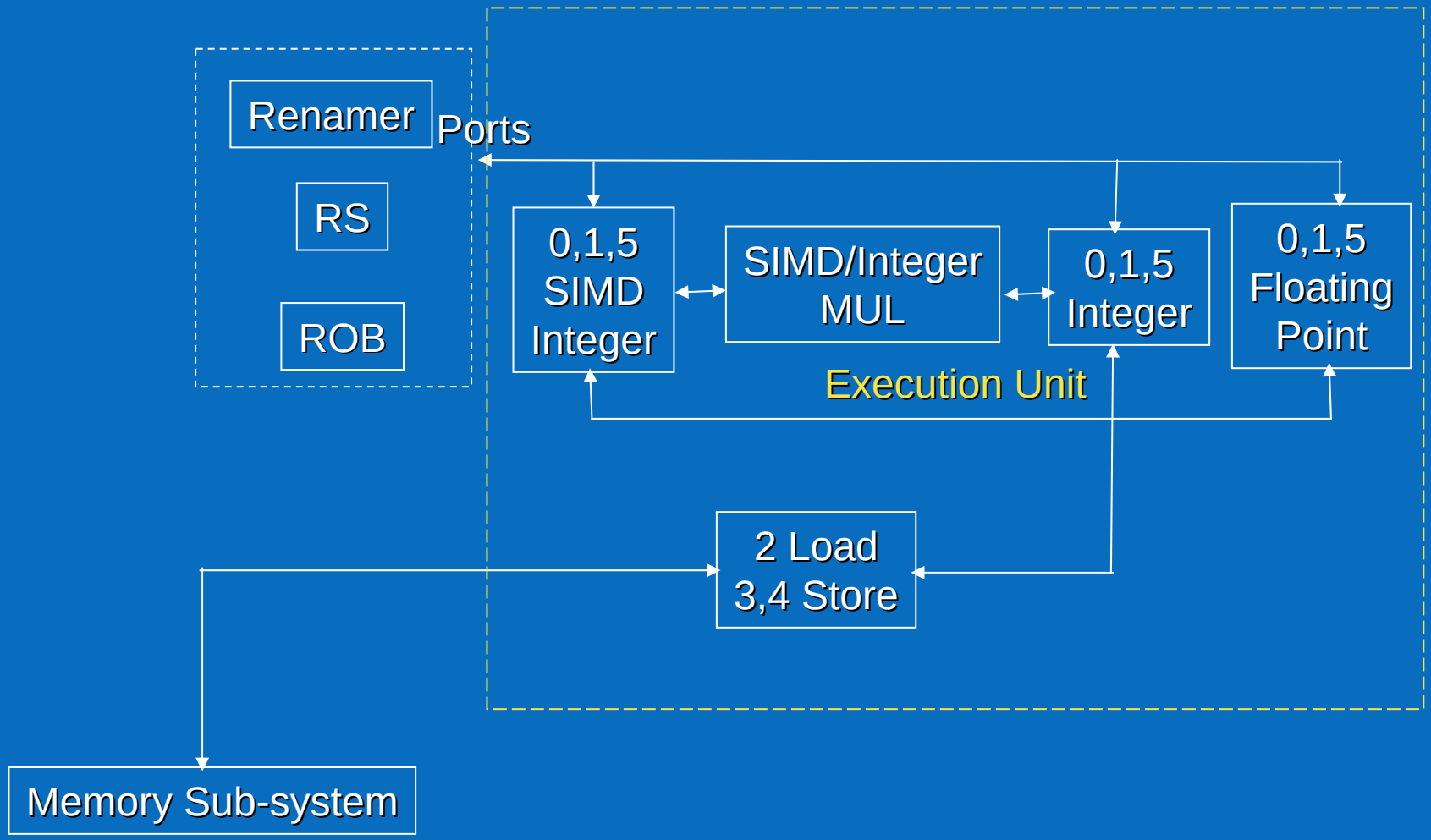
Accepted decoded u-ops, assign resources, execute and retire u-ops

- Renamer
- Reservation station (RS)
- Issue ports
- Execution Unit



# Execution Core Building Blocks



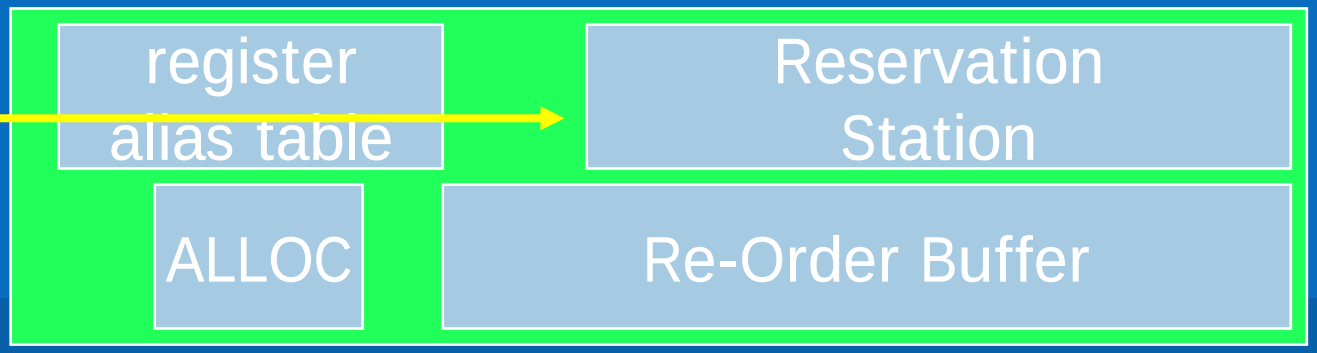
# Rename and Resources

4 uops renamed / retired per clock

- one taken branch, any # of untaken
- one fxchg per cycle

Uops written to RS and ROB

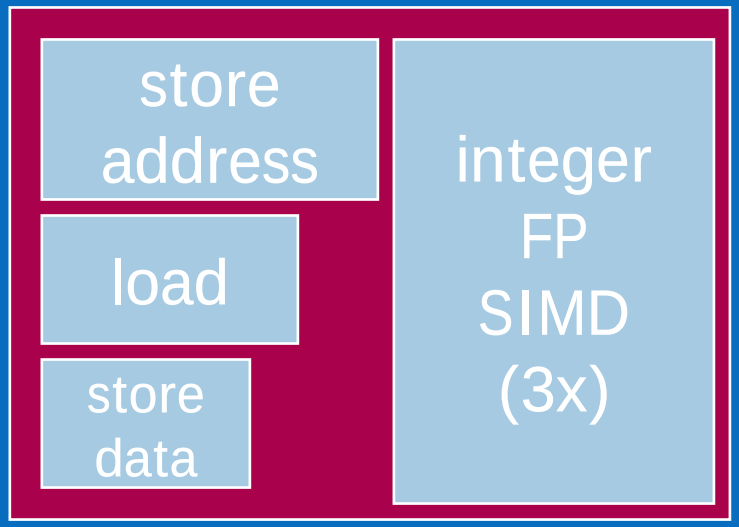
- Decoded uops were renamed and allocated with resource by RAT and sent to ROB read and RS
- RS waits for sources to arrive allowing OOO execution
- Registers not “in flight” read from ROB during RS write



# Issue Ports and Execution Units

6 dispatch ports from RS

- 3 execution ports
  - (shared for integer / fp / simd)
- load
- store (address)
- store (data)




128-bit SSE implementation

- Port 0 has packed multiply (4 cycles SP 5 DP pipelined)
- Port 1 has packed add (3 cycles all precisions)

FP data has one additional cycle bypass latency

- Do not mix SSE FP and SSE integer ops on same register

| Avoid: | Addps XMM0,XMM1<br>Pand xmm0,xmm3<br>Addps xmm2,xmm0 |
|---|---|

| Better: | Addps XMM0,XMM1<br>Addps xmm2,xmm0<br>Pand xmm0,xmm3 |
|---|---|

# The Out Of Order

each uop only takes a single RS entry

load + add dispatches twice (load, then add)

mulps dispatches once when load + add to write back

sta + std dispatches twice

- sta (address) can fire as early as possible
- std must wait for mulps to write back

cmpjne dispatches only once (functionality is truly fused)

- no dependency, can fire as early as it wants

```
cmpjne    EAX, #2000, TOP                    RS
sta_std   [EAX+240], xmm0
mulps     xmm0, xmm0, xmm0
load_add  xmm0, xmm0, [EAX+16]
```

# Dispatching to OOO EXE

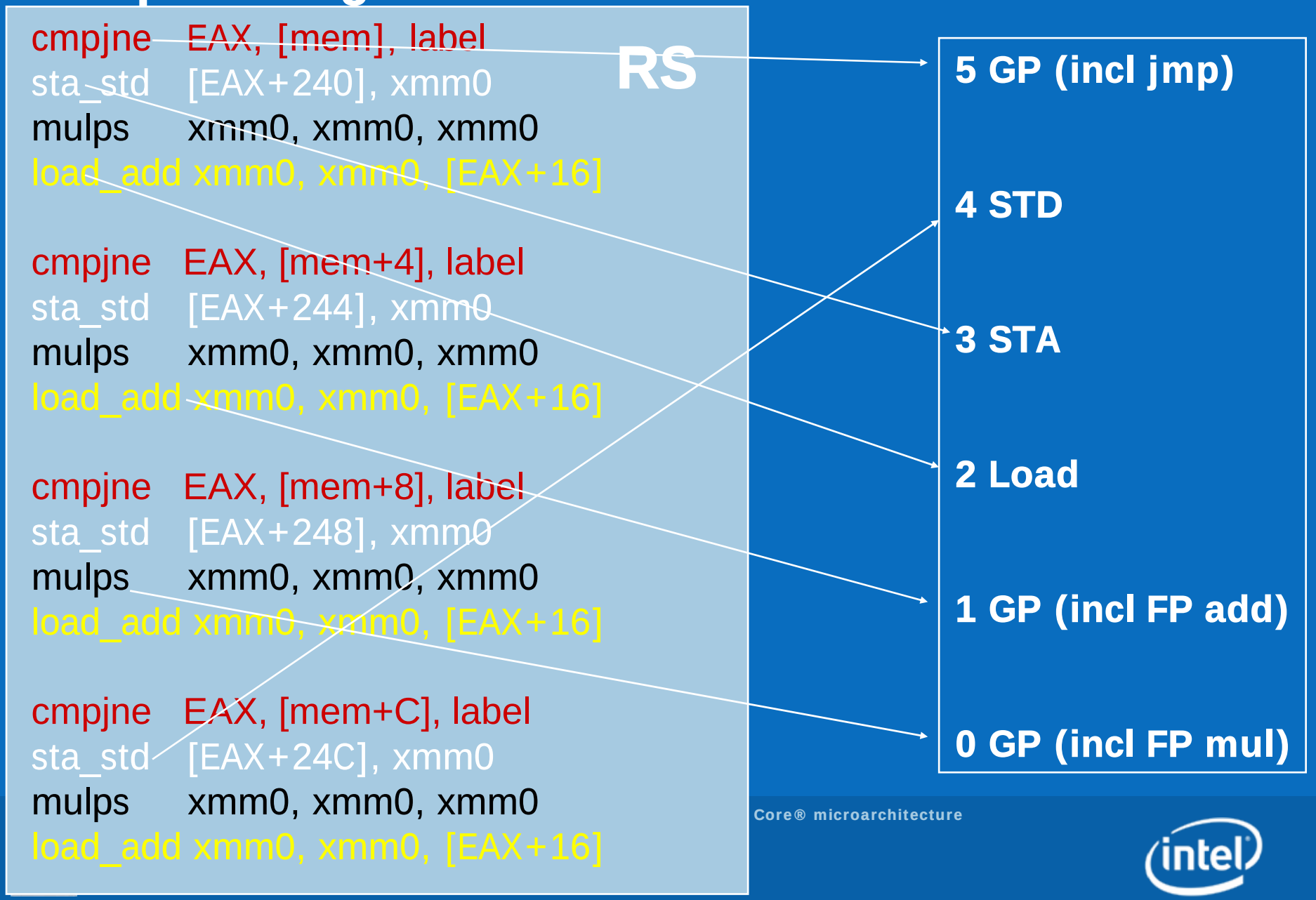
```
cmpjne   EAX, [mem], label
sta_std  [EAX+240], xmm0
mulps    xmm0, xmm0, xmm0
load_add xmm0, xmm0, [EAX+16]

cmpjne   EAX, [mem+4], label
sta_std  [EAX+244], xmm0
mulps    xmm0, xmm0, xmm0
load_add xmm0, xmm0, [EAX+16]

cmpjne   EAX, [mem+8], label
sta_std  [EAX+248], xmm0
mulps    xmm0, xmm0, xmm0
load_add xmm0, xmm0, [EAX+16]

cmpjne   EAX, [mem+C], label
sta_std  [EAX+24C], xmm0
mulps    xmm0, xmm0, xmm0
load_add xmm0, xmm0, [EAX+16]
```

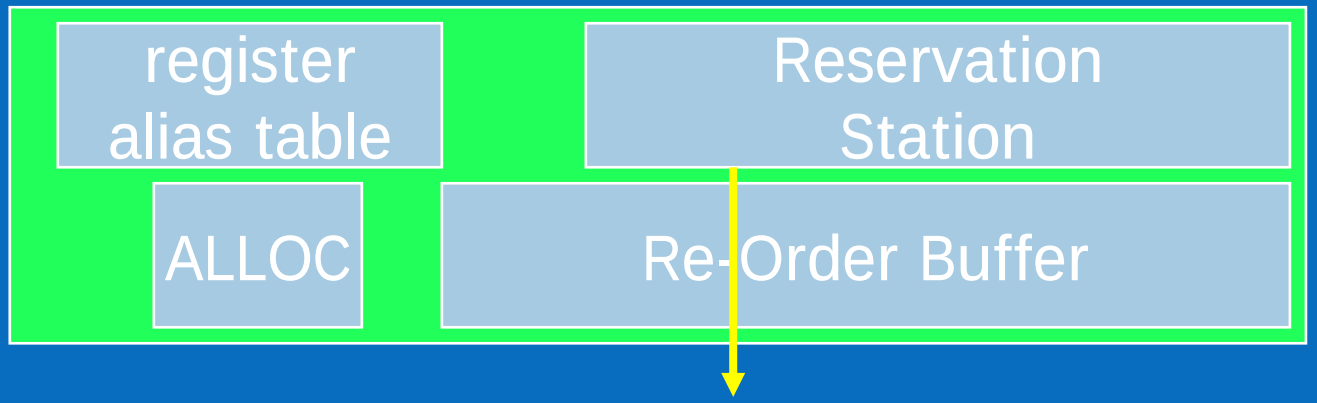
RS



# Retirement Unit

ReOrder Buffer (ROB)

- Holds micro-ops in various stages of completion
- Buffers completed micro-ops
- updates the architectural state in order
- manages ordering of exceptions



# Core® Micro-architecture Memory Sub-System

32k D-Cache (8-way, 64 byte line size)

Loads & Stores

- One 128-bit load and one 128-bit store per cycle to different memory locations
- Out of order Memory operations

Data Prefetching

Memory Disambiguation

Store Forwarding

# Advanced Memory Access

3 clk latency and 1 clk thrput of L1D; 14 and 2 for L2

Miss Latencies

- L1 miss hits L2 ~ 10 cycles
- L2 miss, access to memory ~300 cycles (server/FBD)
- L2 miss, access to memory ~165 cycles (Desk/DDR2)
  - C step broadwater is reported to have ~50ns latency

Cache Bandwidth

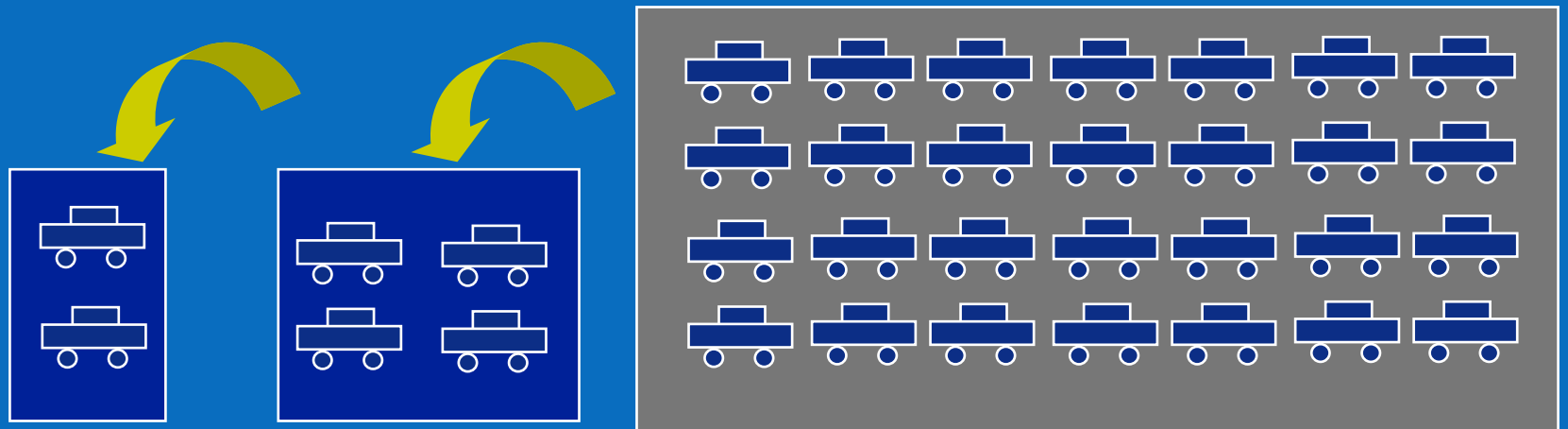
- Bandwidth to cache ~ 8.5 bytes/cycle

Memory Bandwidth

- Desktop ~ 6 GB/sec/socket (linux)
- Server ~3.5 GB/sec/socket

# Advanced Memory Access / Enhanced Data Pre-fetch Logic

Speculates the next needed data and loads it into cache by HW and/or SW



**Door (L1 Cache)**

**Valet Parking Area (L2 Cache)**

**Main Parking Lot (External Memory)**

# Advanced Memory Access / Enhanced Data Pre-fetch Logic (cont.)

- L1D cache prefetching
  - Data Cache Unit Prefetcher
    - Known as the streaming prefetcher
    - Recognizes ascending access patterns in recently loaded data
    - Prefetches the next line into the processors cache
  - Instruction Based Stride Prefetcher
    - Prefetches based upon a load having a regular stride
    - Can prefetch forward or backward 2 Kbytes
      - 1/2 default page size
- L2 cache prefetching: Data Prefetch Logic (DPL)
  - Prefetches data to the 2nd level cache before the DCU requests the data
  - Maintains 2 tables for tracking loads
    - Upstream – 16 entries
    - Downstream – 4 entries
  - Every load is either found in the DPL or generates a new entry
  - Upon recognition of the 2nd load of a “stream” the DPL will prefetch the next load

# Advanced Memory Access / Memory Disambiguation

Memory Disambiguation predictor

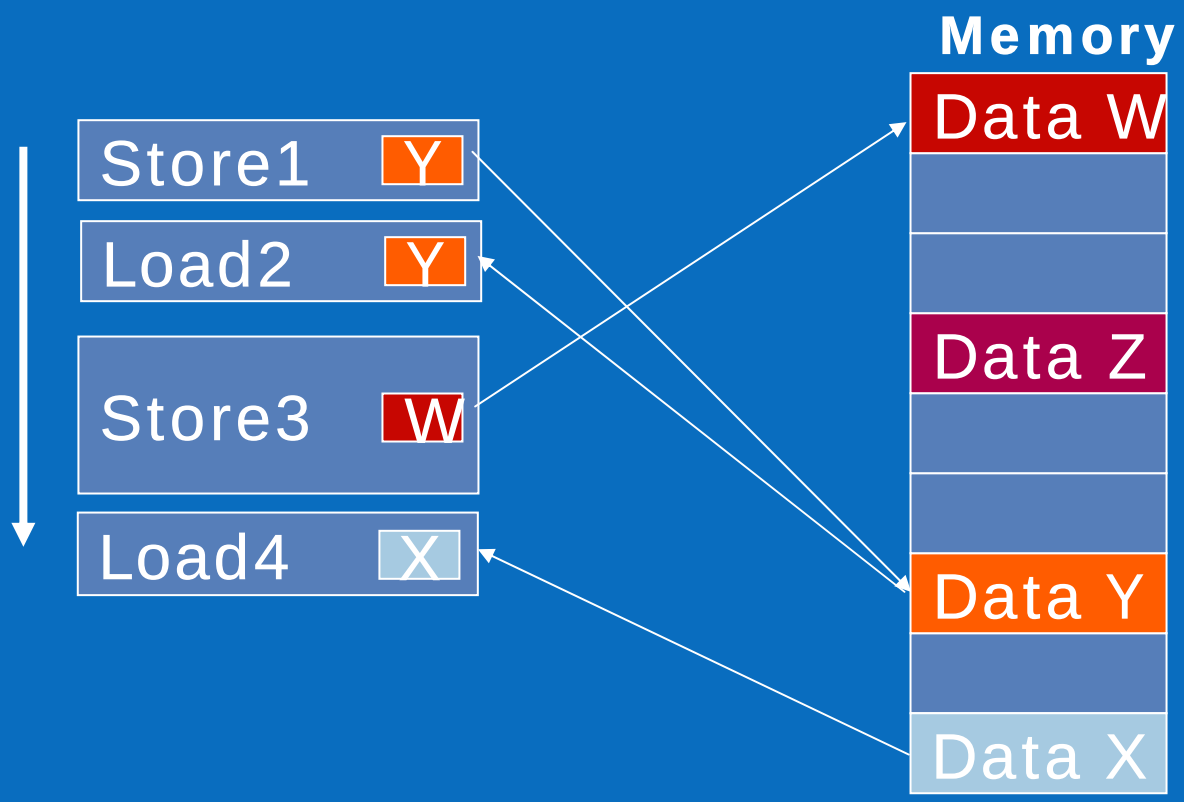
- Loads that are predicted NOT to forward from preceding store are allowed to schedule as early as possible
  - increasing the performance of OOO memory pipelines

Disambiguated loads checked at retirement

- Extension to existing coherency mechanism
- Invisible to software and system

# Advanced Memory Access / Memory Disambiguation Absent

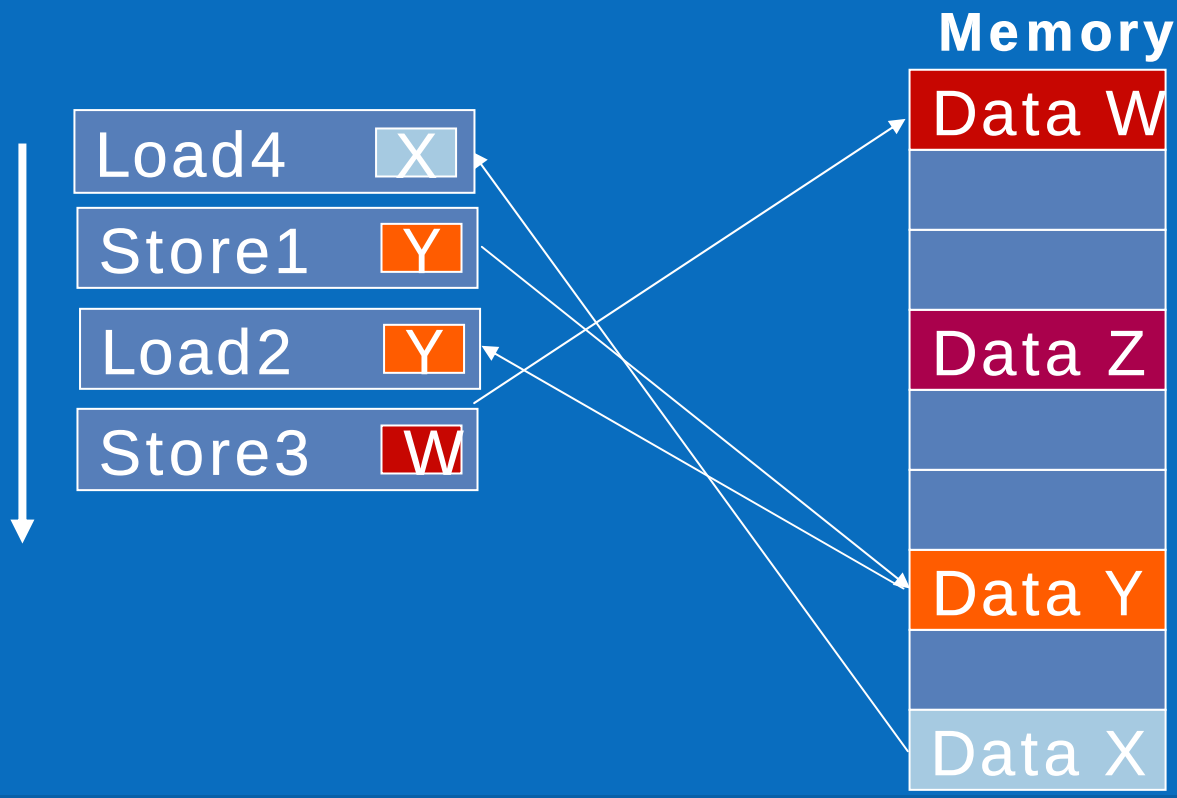
Load4 must WAIT until previous stores complete



# Advanced Memory Access / Memory Disambiguation Presented

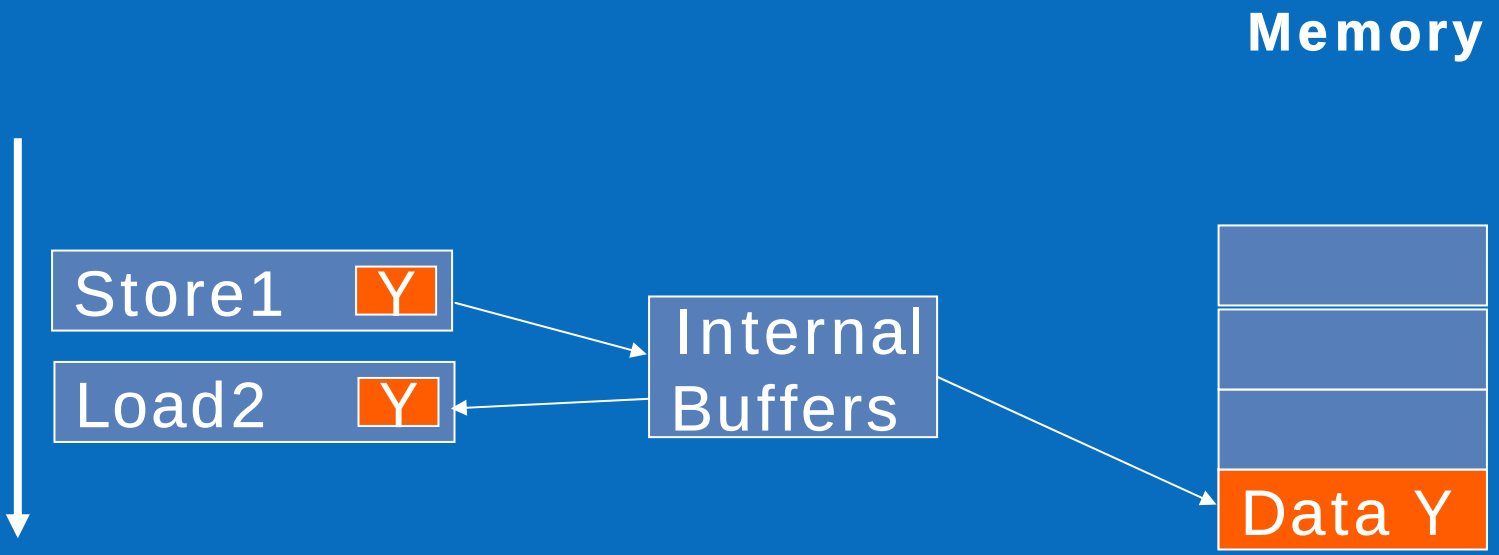
Loads can decouple from stores

Load4 can get its data WITHOUT waiting for stores

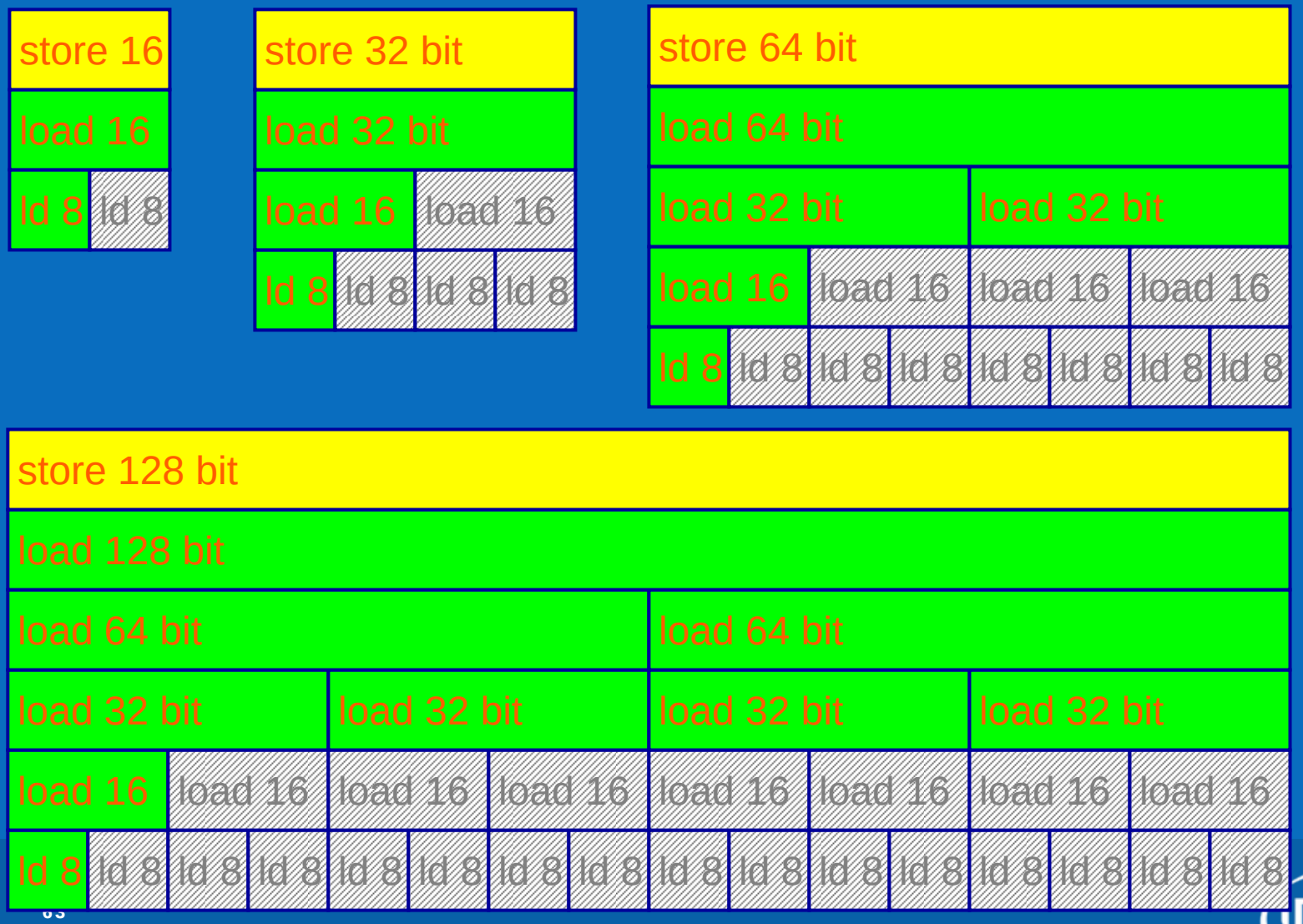


# Advanced Memory Access / Stores Forwarding

If a load follows a store and reloads the data that the store writes to memory, the micro-architecture can forward the data directly from the store to the load

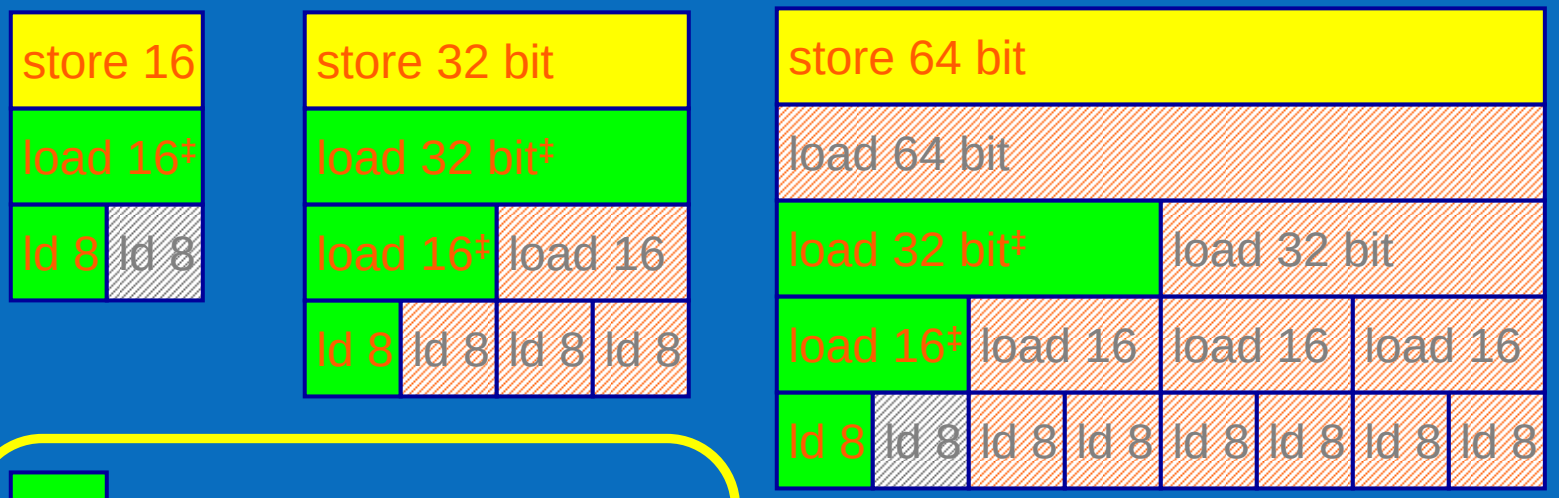


# Advanced Memory Access / Stores Forwarding: Aligned Store Cases



# Advanced Memory Access / Stores Forwarding: Unaligned Cases

Note that unaligned store forward does not occur when the *load* crosses a cache line boundary



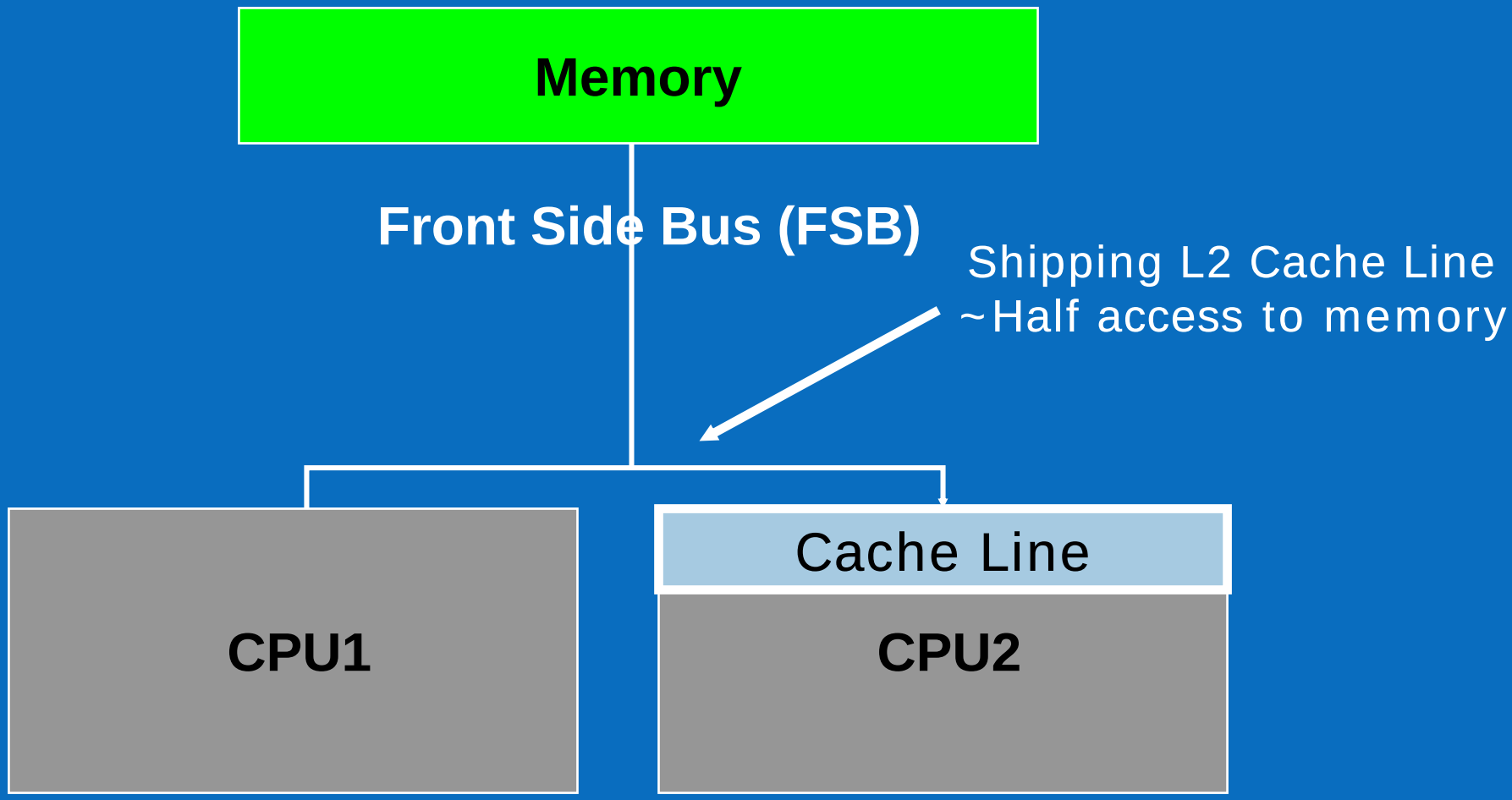

ld 8 Store forwarded to load

ld 8 No forwarding

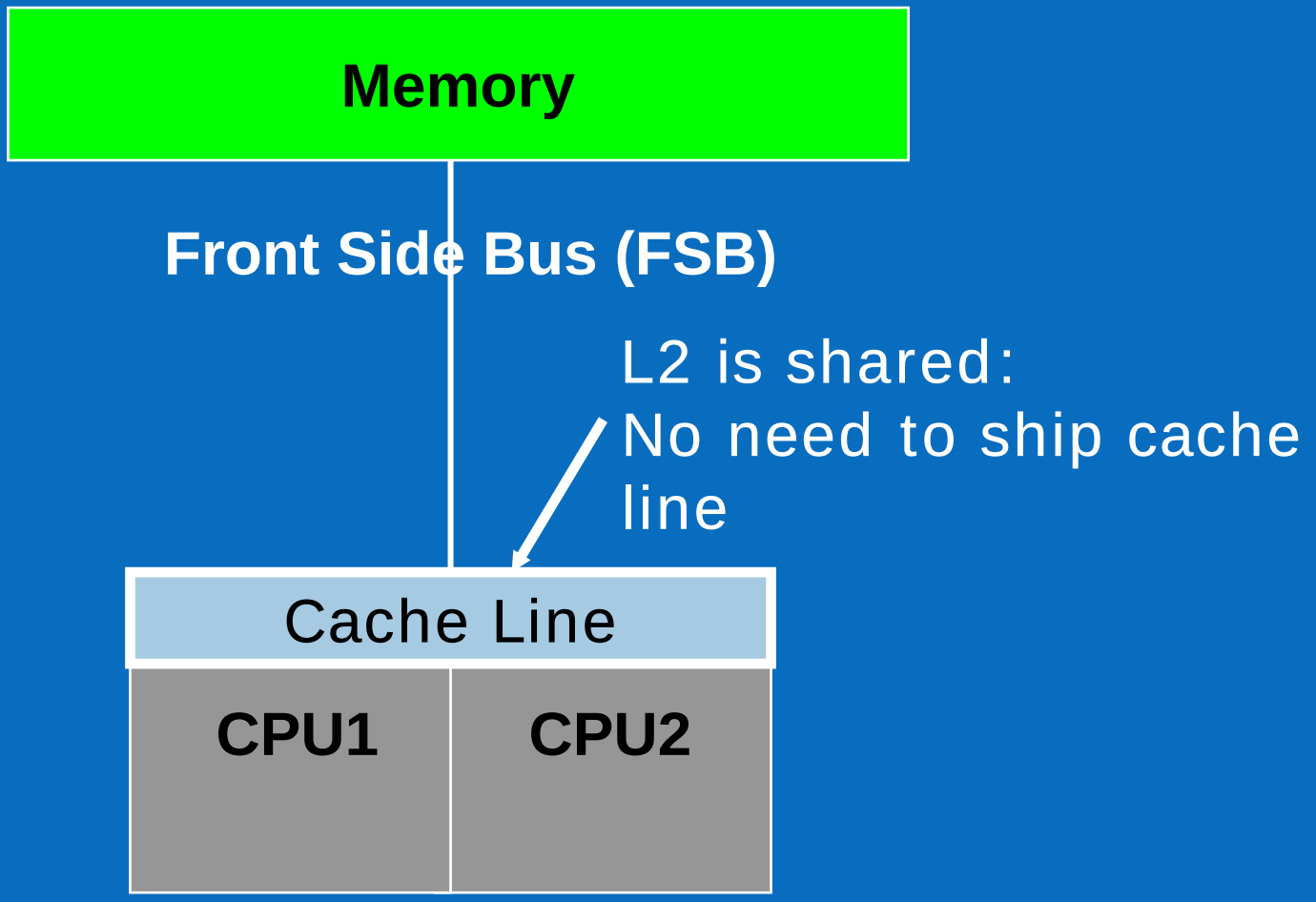
‡: No forwarding if the load crosses a cache line boundary

Note: Unaligned 128-bit stores are issued as two 64-bit stores. This provides *two* alignments for store forwarding

# Advanced Smart Cache® Technology: Advantages of Shared Cache



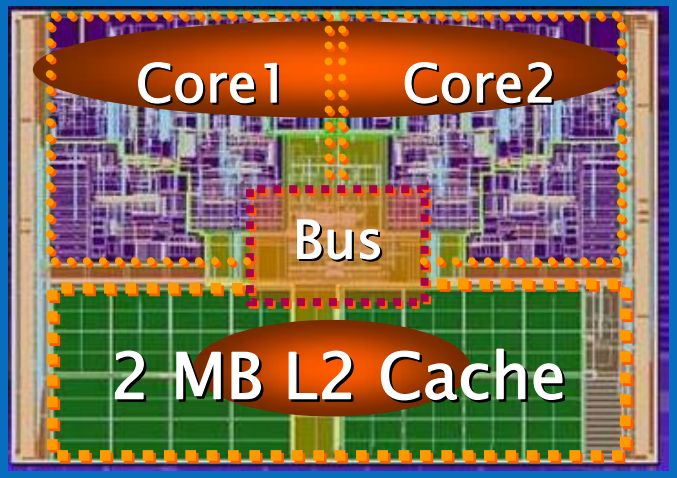
# Advanced Smart Cache® Technology: Advantages of Shared Cache (cont.)



# Advanced Smart Cache® Technology (cont.)

Load & Store Access order

1. L1 cache of immediate core
2. L1 cache of the other core
3. L2 cache
4. Memory



# Advanced Smart Cache® Technology (cont.)

Shared second level (L2) 2MB 8-way or 4MB 16-way instruction and data cache

Cache 2 cache transfer

- improves producer / consumer style MP

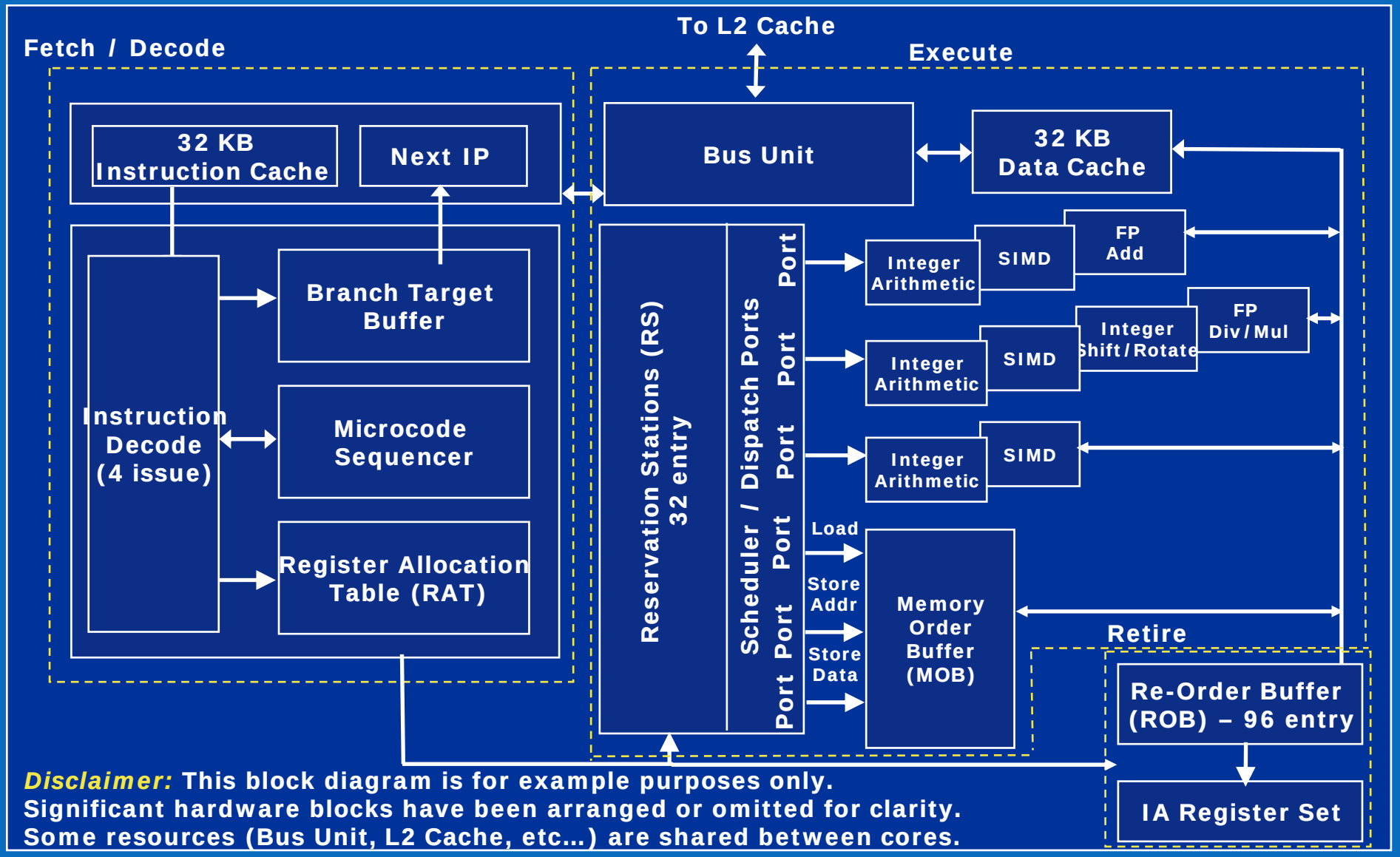
Wider interface to L2

- reduced interference
  - processor line fill is 2 cycles

Higher bandwidth from the L2 cache to the core

- ~14 clock latency and 2 clock throughput

# Intel® Core® Micro-architecture Blocks



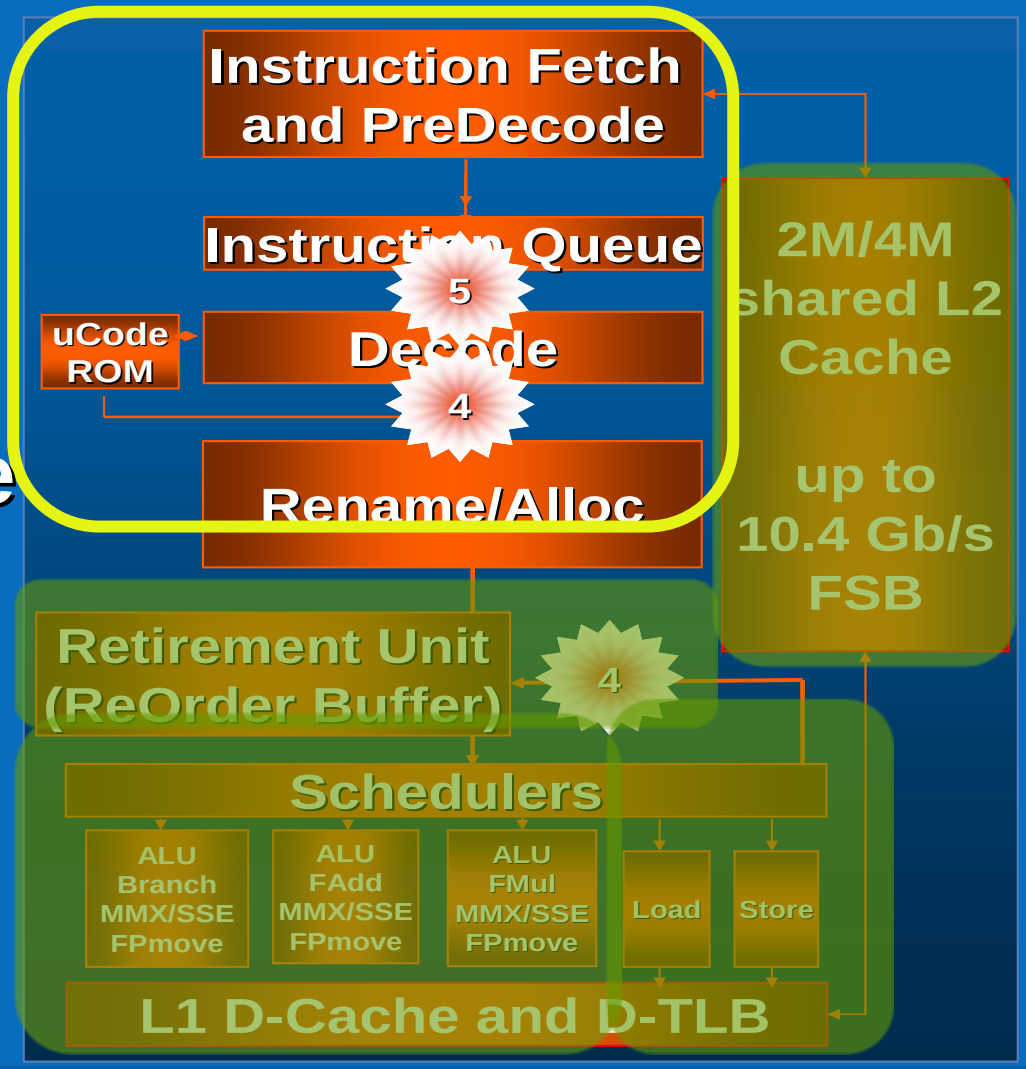

***Disclaimer:*** **This block diagram is for example purposes only.**
**Significant hardware blocks have been arranged or omitted for clarity.**
**Some resources (Bus Unit, L2 Cache, etc…) are shared between cores.**

# Intel® Core® Micro-architecture Notable Features

## Enhanced Pipeline

***in order***

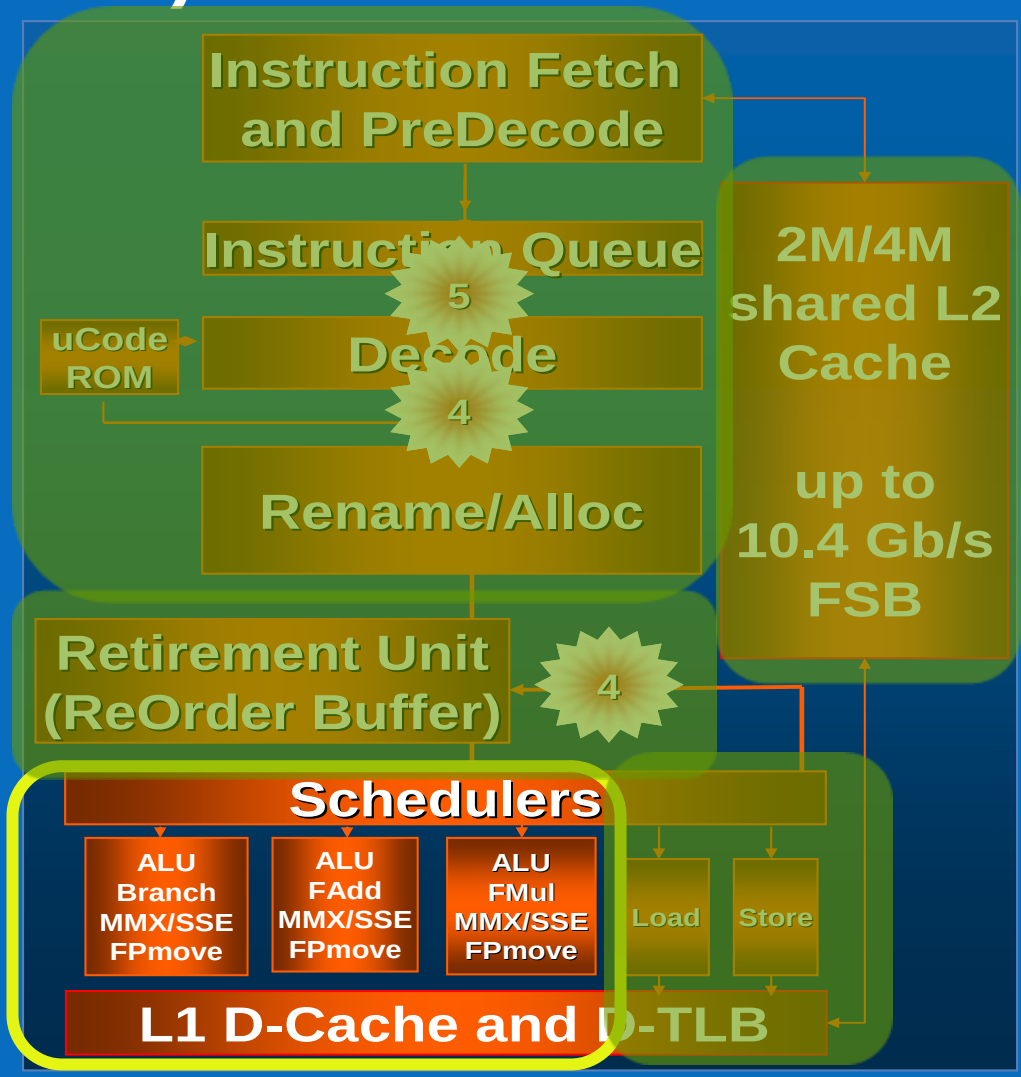
**instruction fetch**
**instruction decode**
**micro-op rename**
**micro-op allocate**



# Intel® Core® Micro-architecture Notable Features
## Enhanced Pipeline (cont.)

***out of order***

**micro-op schedule**
**micro-op execute**



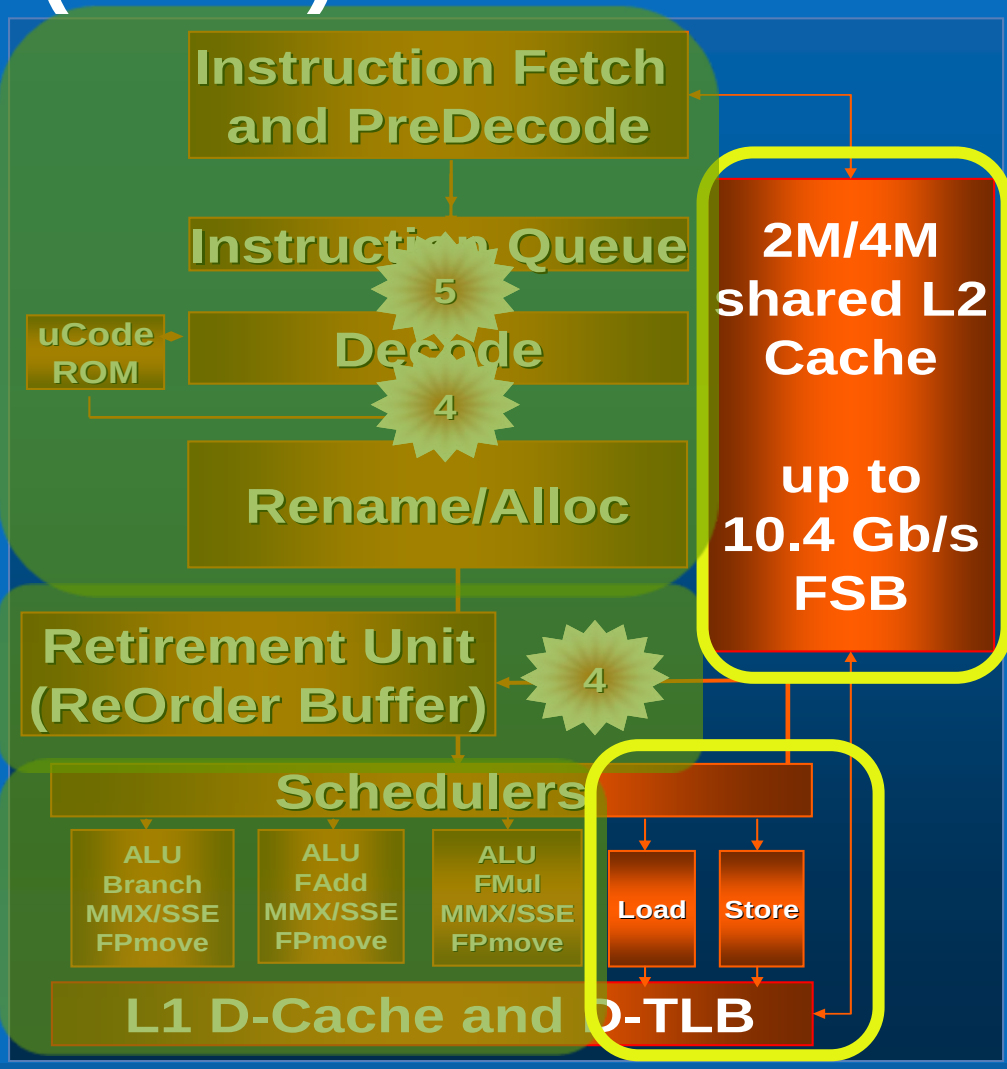
# Intel® Core® Micro-architecture Notable Features

## Enhanced Pipeline (cont.)

***out of order***

**memory pipelines**

**memory order unit maintains architectural ordering requirements**



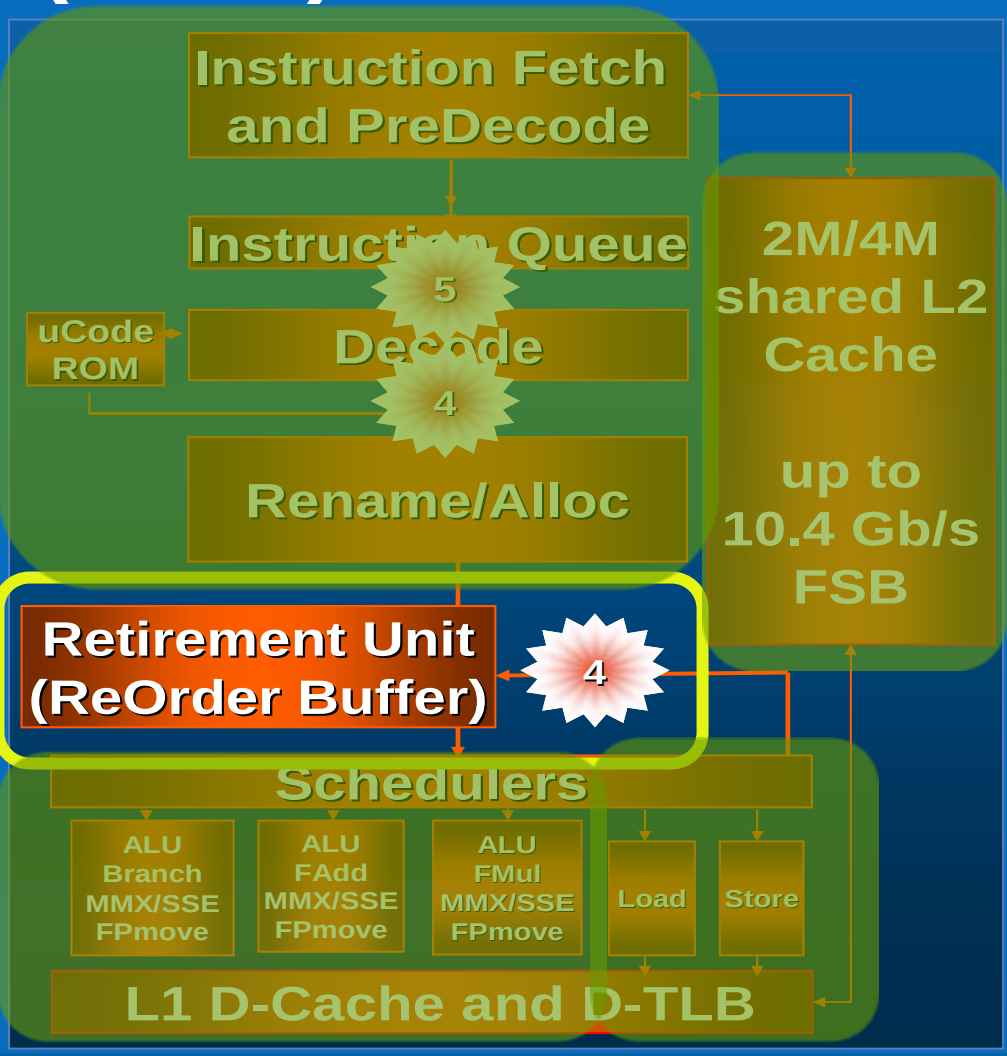
# Intel® Core® Micro-architecture Notable Features

## Enhanced Pipeline (cont.)

***in order***

**micro-op retirement**
**fault handling**

**Retirement Unit**
**maintains illusion**
**of in order**
**instruction retirement**



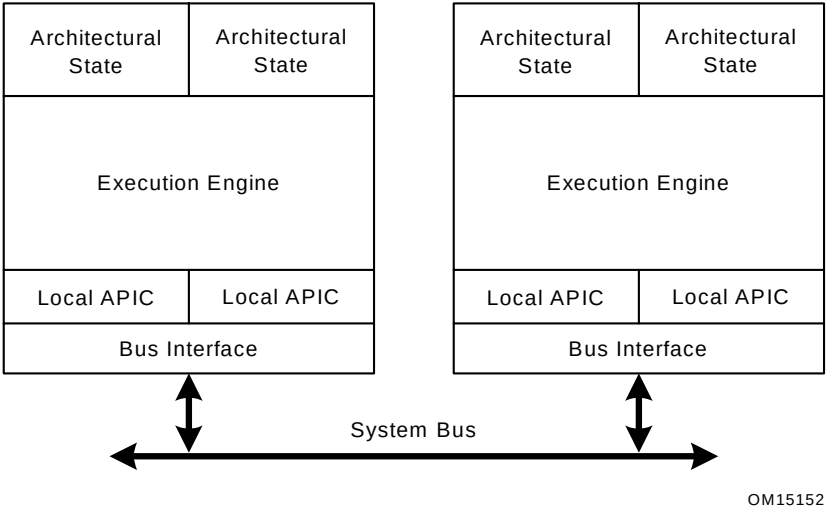



Figure 2-7. Hyper-Threading Technology on an SMP

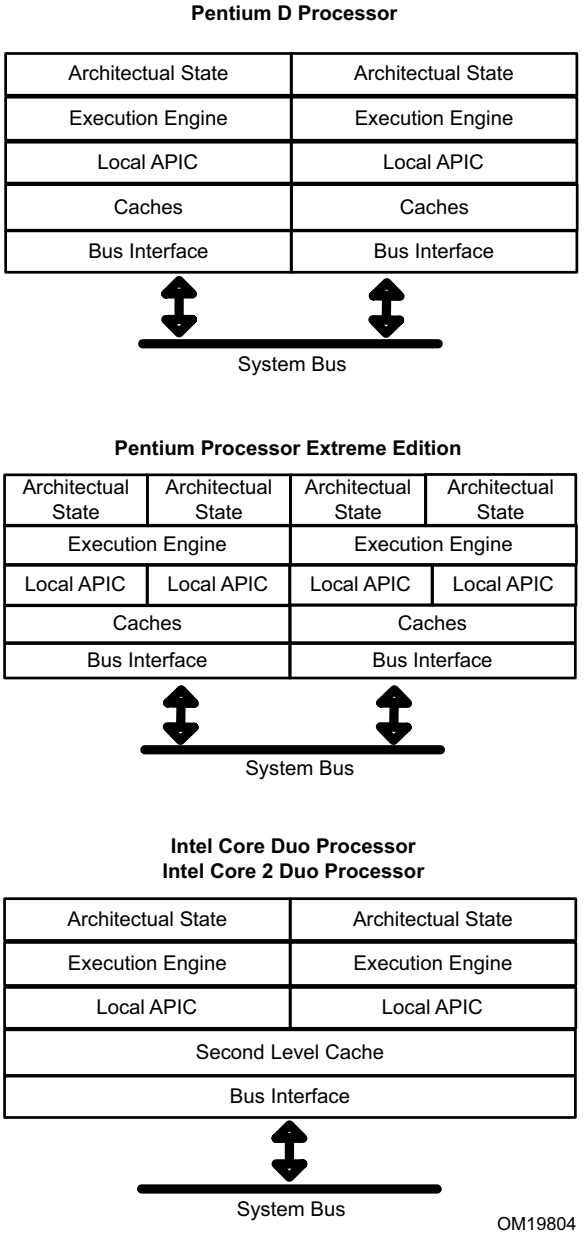



Figure 2-8. Pentium D Processor, Pentium Processor Extreme Edition, Intel Core Duo Processor, and Intel Core 2 Duo Processor

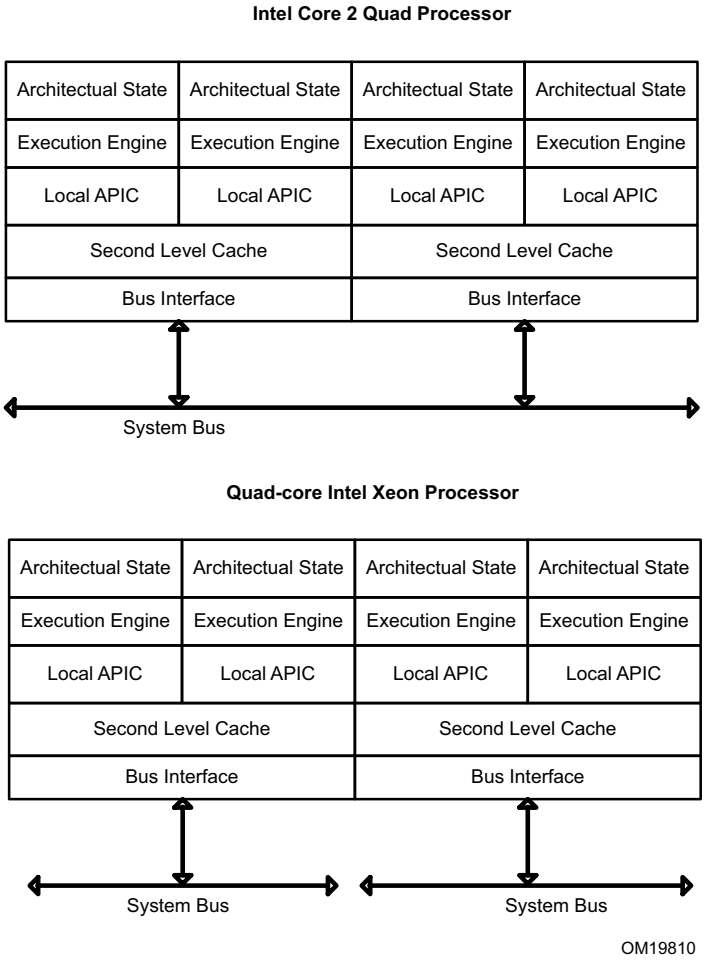



Figure 2-7. Intel 64 Processors that Support Quad-Core