Intel® Processor Micro-architecture – Core®

Intel® Software College

Objectives

After completion of this module you will be able to describe

- Components of an IA processor
- Working flow of the instruction pipeline
- Notable features of the architecture

Agenda

Introduction

Notable features

Micro-architecture drill-down

Advanced cache technology

Coding considerations

Agenda

Introduction

Notable features

Micro-architecture drill-down

Advanced cache technology

Coding considerations

Industrial Recognition

PC Format May 2006
"*Intel Strikes Back!* Conroe is the name. Pistol-whipping Athlon 64s into burger meat is the game.."

Intel's Next Generation Microarchitecture Unveiled
Real World Tech
"Just as important as the technical innovations in Core MPUs, this microarchitecture will have a profound impact on the industry. "

Intel Dishes the Knockout Punch to AMD with Conroe, GD Hardware.com
"…the results were far more than we could hope for and it'll be amusing to see AMD's response to this beat-down session

Intel Regains Performance Crown, Anandtech
"… At 2.8 or 3.0GHz, a Conroe EE would offer even stronger performance than what we've seen here."

Intel Reveals Conroe Architecture, Extremetech
"… And not only was the Intel system running at 2.66GHz— a slower clock rate than the top Pentium 4—it was outpacing an overclocked Athlon 64 FX-60. Wrap your brain around that idea for a bit…"

Conroe Benchmarks - Intel Showing Big Strength Hot Hardware
"… Intel is poised to change the *face* of the desktop computing landscape…"

Performance Summary

Intel® Core™ Microarchitecture dramatically boosts Intel platform performance

- Conroe & Woodcrest drive clear Desktop/Server performance leadership
- Merom extends Intel Mobile performance leadership

Intel® Core™ Microarchitecture-based platforms set the bar in Performance and Energy Efficiency for the Multi-Core era

- Intel's 3rd generation dual-core (while competition stuck on 1st generation)
- New Intel high-performance 'engine': Wider, Smarter, Faster, More Efficient

Best Processor on the Planet: Energy-Efficient Performance [1]

20% (Merom), 40% (Conroe), 80% (Woodcrest) Performance Boosts[1] !

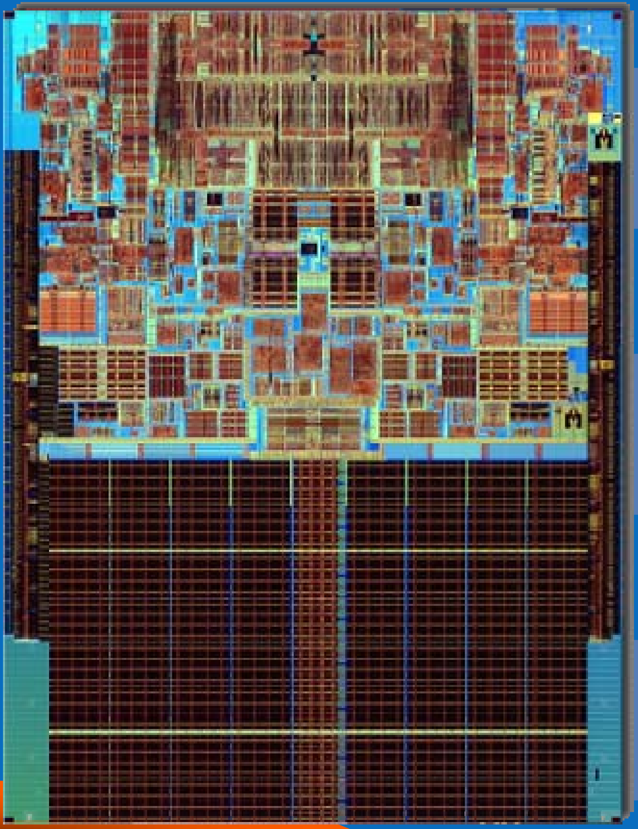
[1] *Based on SPECint*_rate_base2000*

Intel® Core™ Microarchitecture

Intel® NetBurst®

+ New Innovations

Mobile Microarchitecture



Agenda

Introduction

Notable features

- Wide Dynamic Execution
- Smart Memory Access
- Advanced Smart Cache
- Advanced Digital Media Boost
- Intelligent Power Capability

Micro-architecture drill-down

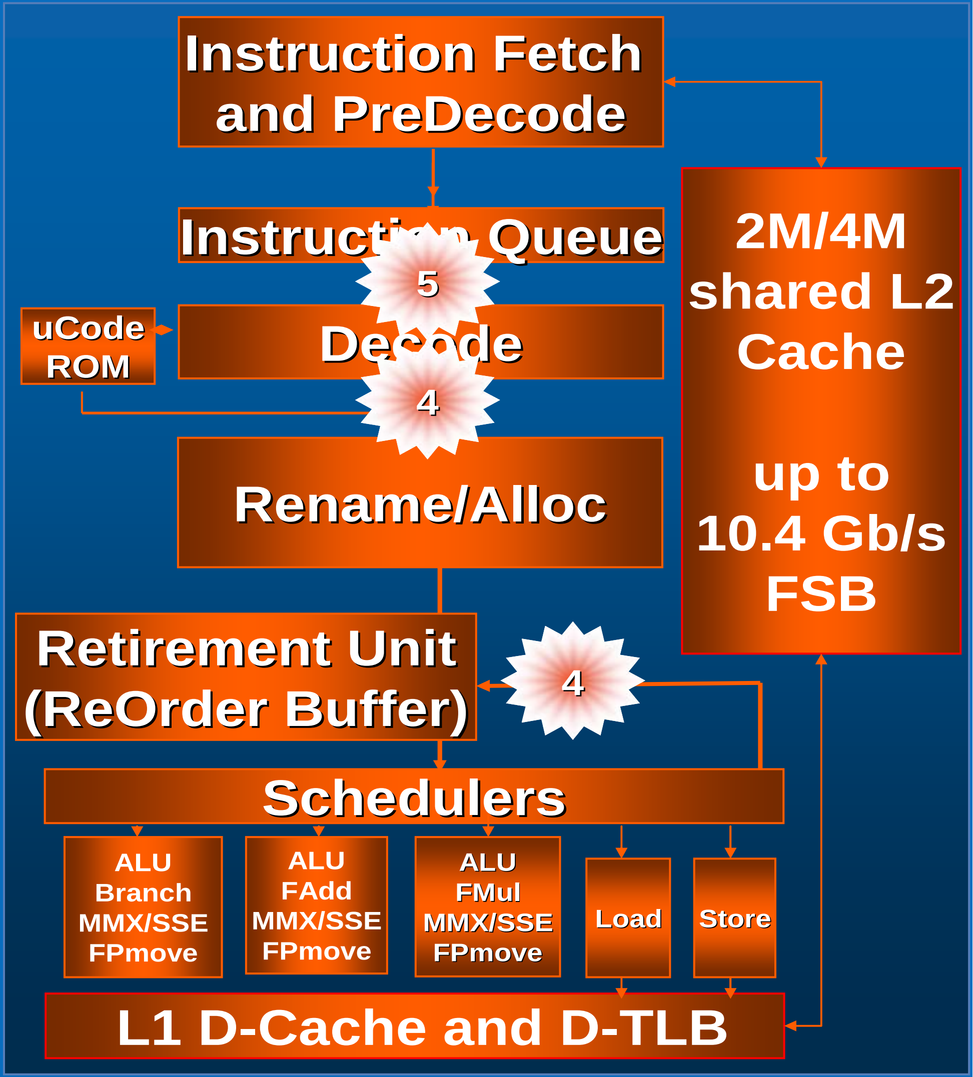
Advanced cache technology

Coding considerations

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 Memory Access

- Improved prefetching
- Memory disambiguation
  - Advance load before a possible data dependency (pointer conflict)
    - Earlier loads hide memory latencies

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

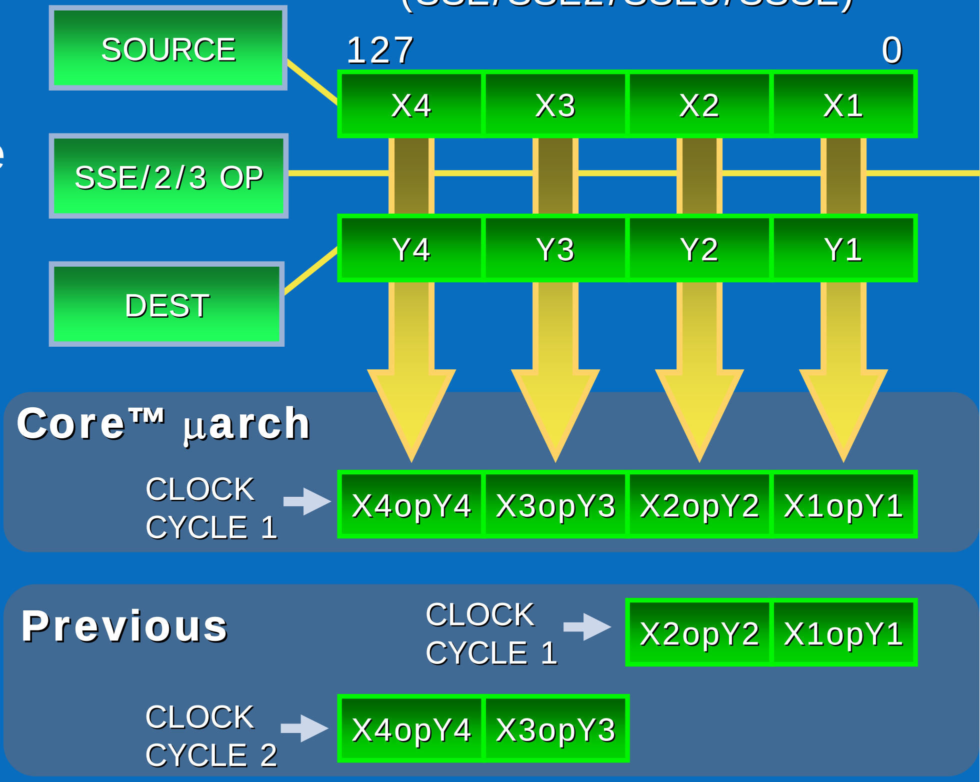
Intel® Core® Micro-architecture Notable Features (cont.)

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

SIMD Operation

(SSE/SSE2/SSE3/SSSE)



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

Intelligent Power Capability

- Advanced power gating & Dynamic power coordination
  - Multi-point demand-based switching
  - Voltage-Frequency switching separation
  - Supports transitions to deeper sleep modes
  - Event blocking
  - Clock partitioning and recovery
  - Dynamic Bus Parking
  - During periods of high performance execution, many parts of the chip core can be shut off

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

Intelligent Power Capability - Split Busses (core power feature)

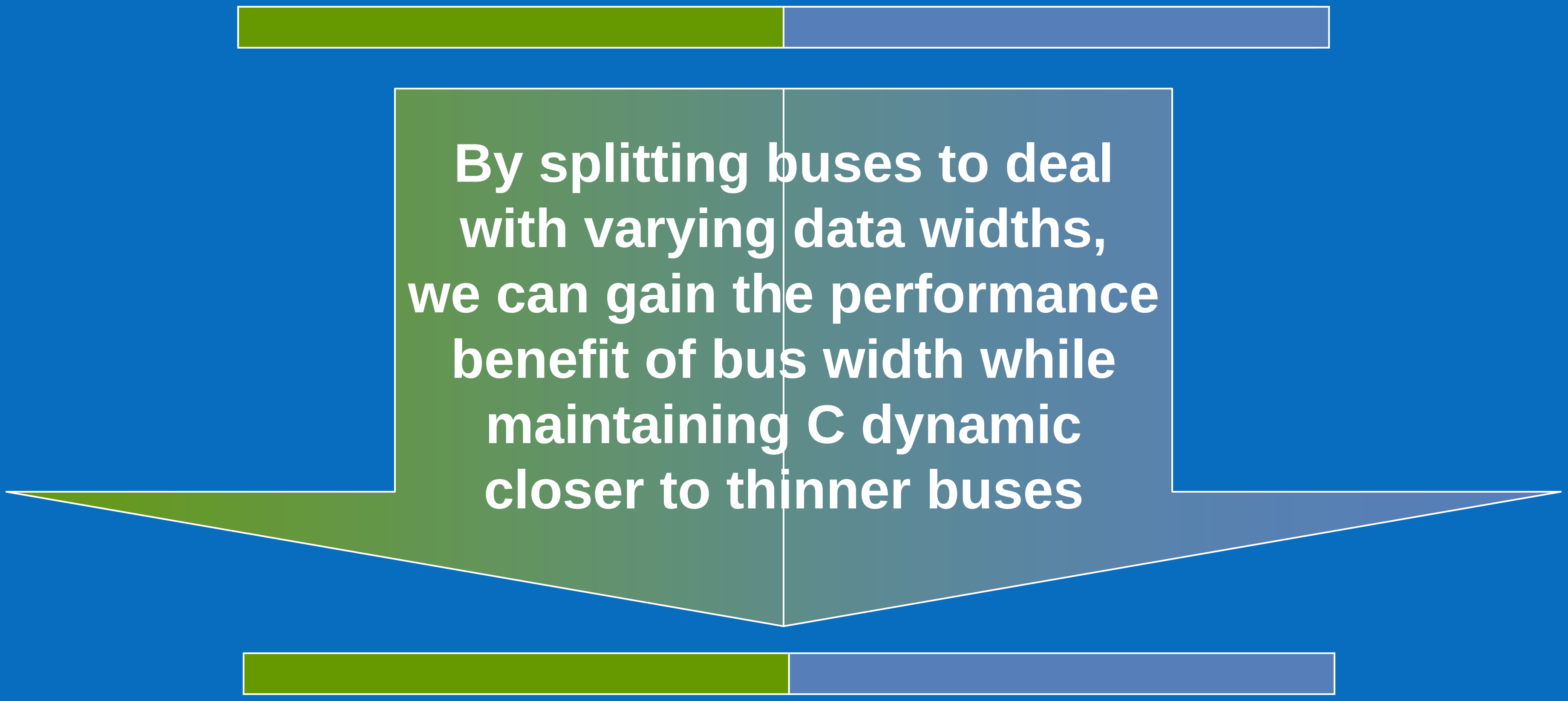
Many buses are sized
for worst case data

(x86 instruction of 15 bytes)
(ALU can write-back 128 bits)

Improved Energy Efficiency

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

Intelligent Power Capability - Split Busses (core power feature)



By splitting buses to deal with varying data widths, we can gain the performance benefit of bus width while maintaining C dynamic closer to thinner buses

Improved Energy Efficiency

Agenda

Introduction

Notable features

Micro-architecture drill-down

- Front End
- Out-Of-Order Execution Core
- Memory Sub-system

Advanced cache technology

Coding considerations

Key Terms

CISC vs. RISC

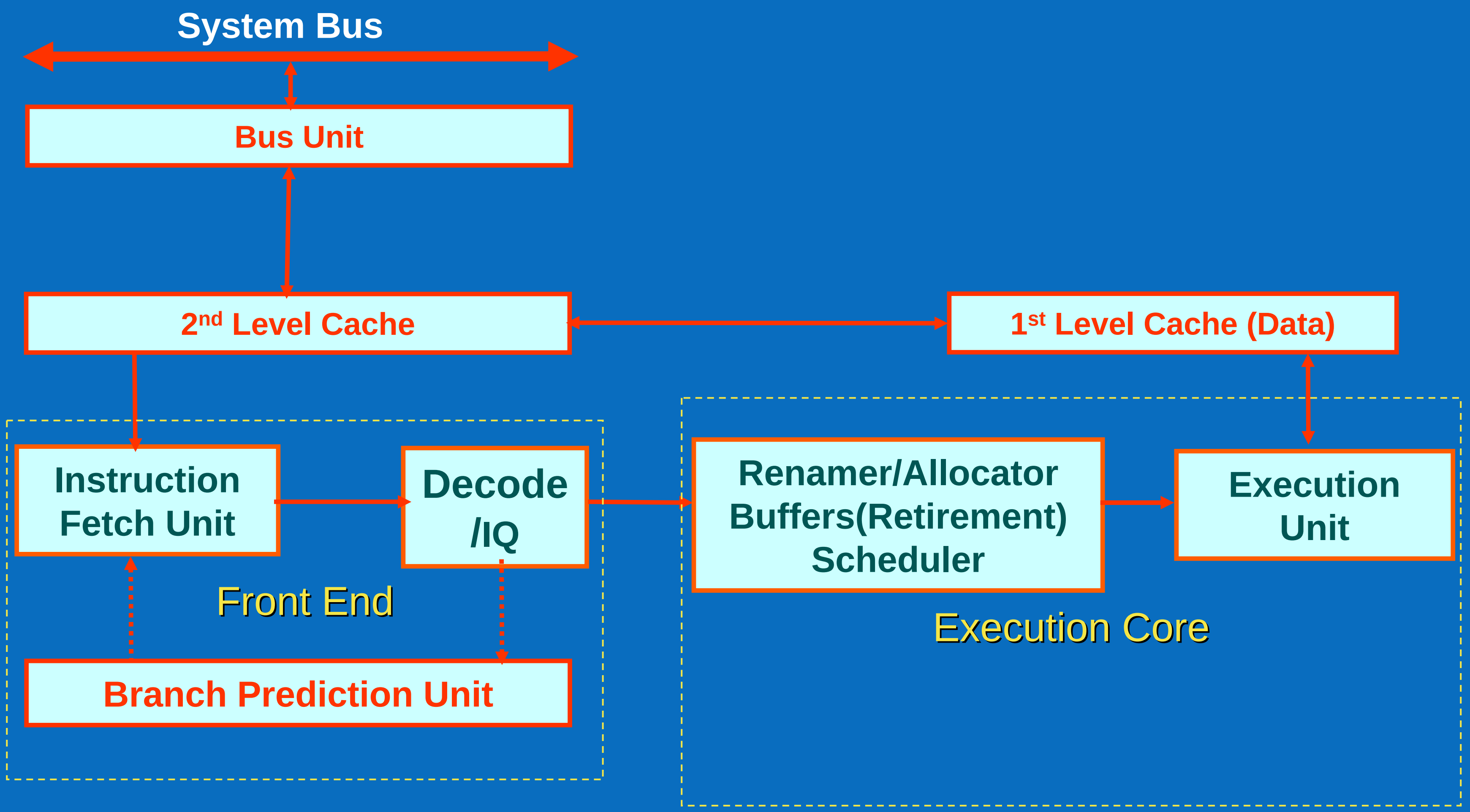
Super-scalar

Out Of Order vs. In Order

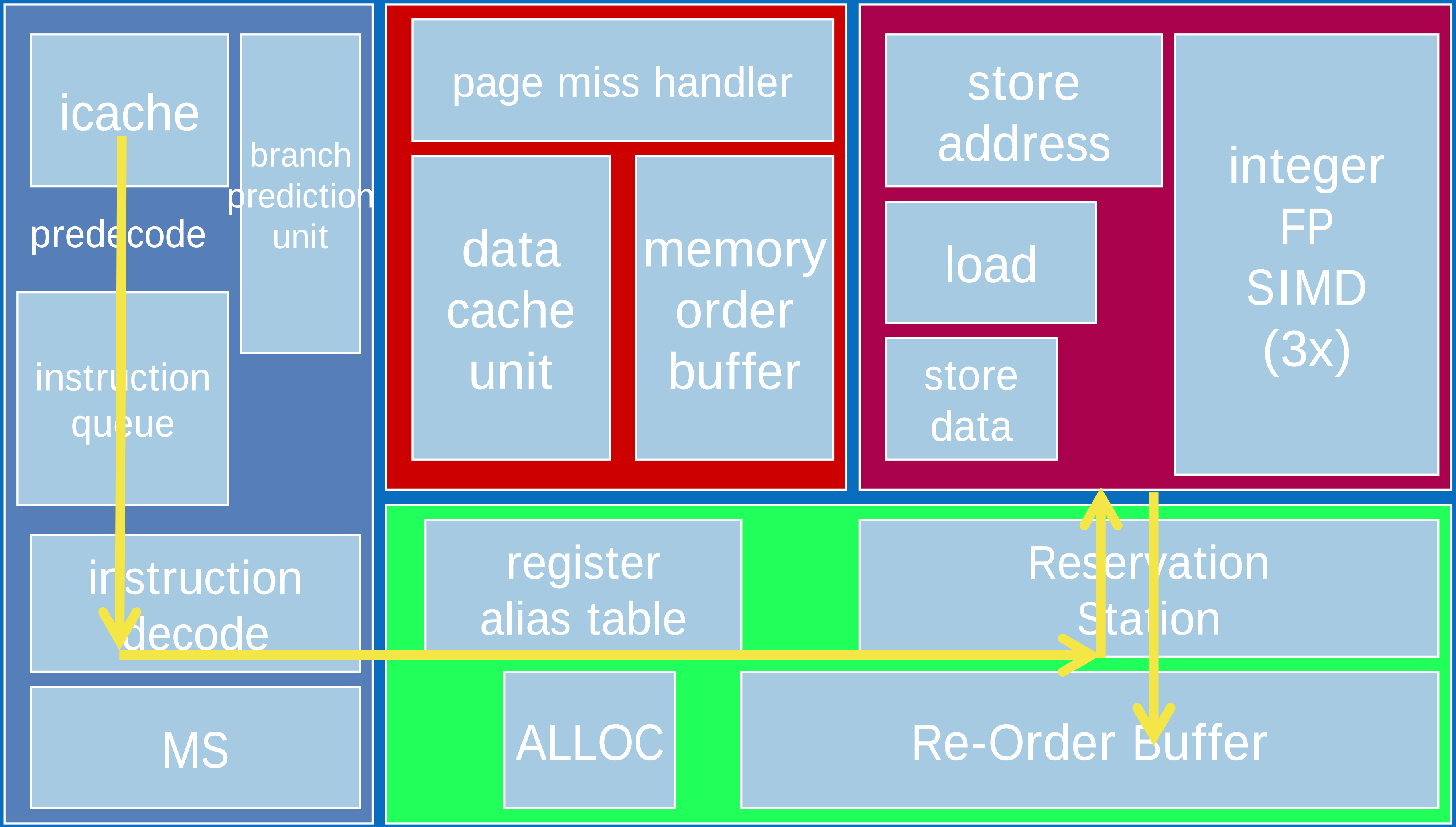
Architecture vs. Micro-architecture

- 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



Agenda

Introduction

Notable features

Micro-architecture drill-down

- Front End
- Out-Of-Order Execution Core
- Memory Sub-system

Advanced cache technology

Coding considerations

Core® Micro-architecture Front End

Instruction preparation before executed

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

Core® Micro-architecture Front End

Instruction Fetch Unit

Instruction Queue

Instruction Decode Unit

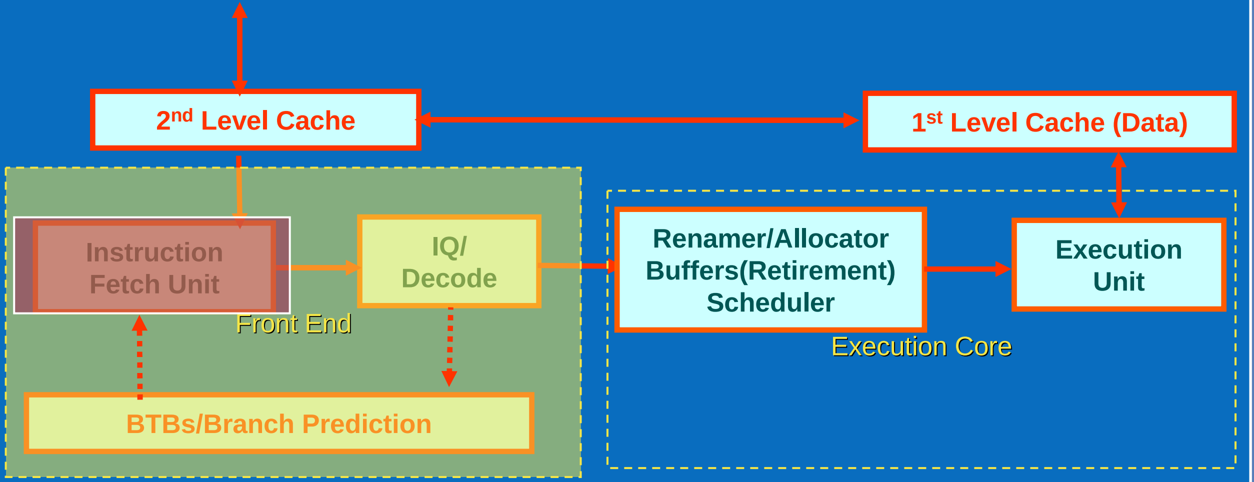
Branch Prediction Unit

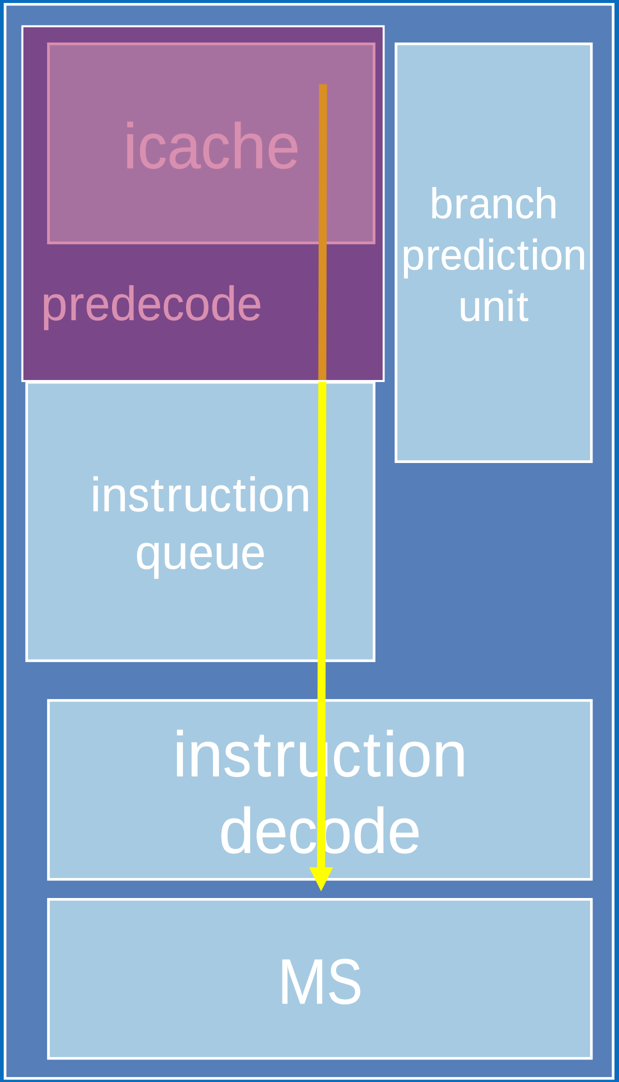
Instruction Fetch Unit

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 Prefixes (66H/67H)	Opcode	ModR/M	SIB	Displacement	Immediate

Core® Micro-architecture Front End

Instruction Fetch Unit

Instruction Queue

Instruction Decode Unit

Branch Prediction Unit

Instruction Queue

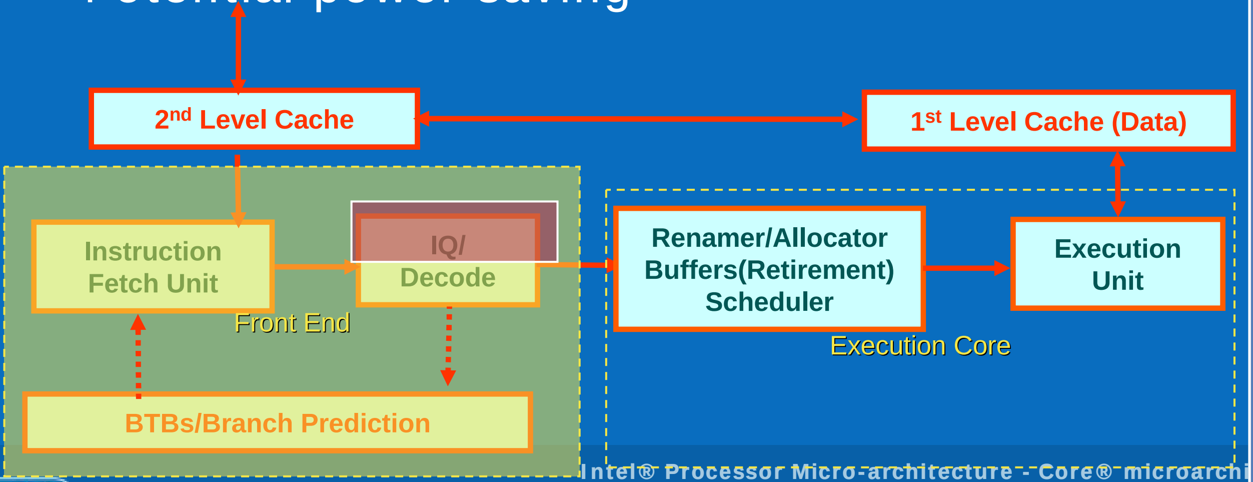
Buffer between instruction pre-decode unit and decoder

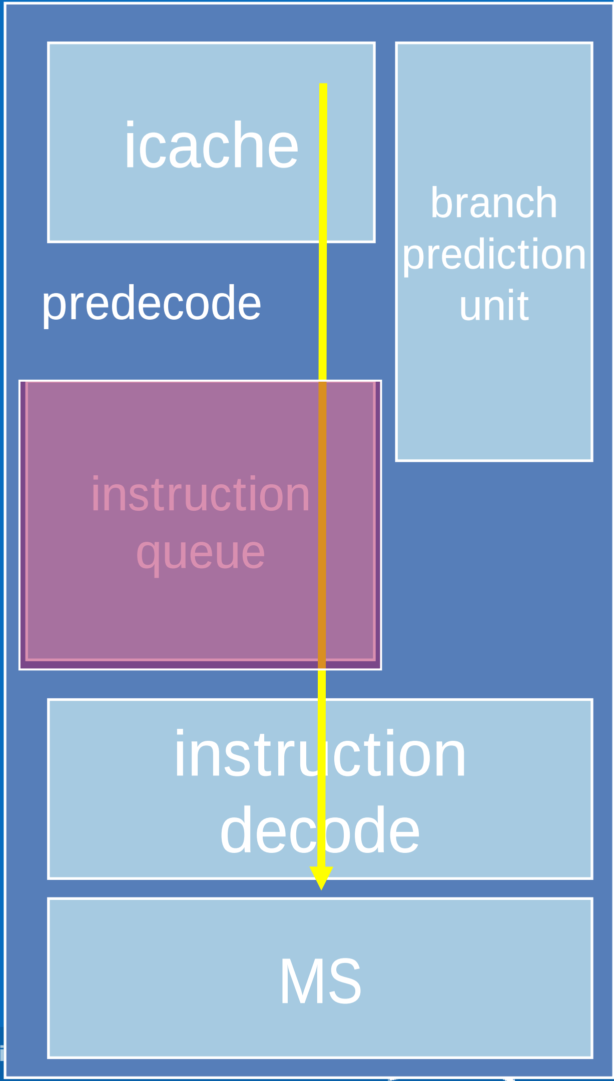
- up to six predecoded instructions written per cycle
- 18 Instructions contained in IQ
- up to 5 Instructions read from IQ

Potential Loop cache

Loop Stream Detector (LSD) support

- Re-use of decoded instruction
- Potential power saving





Core® Micro-architecture Front End

Instruction Fetch Unit

Instruction Queue

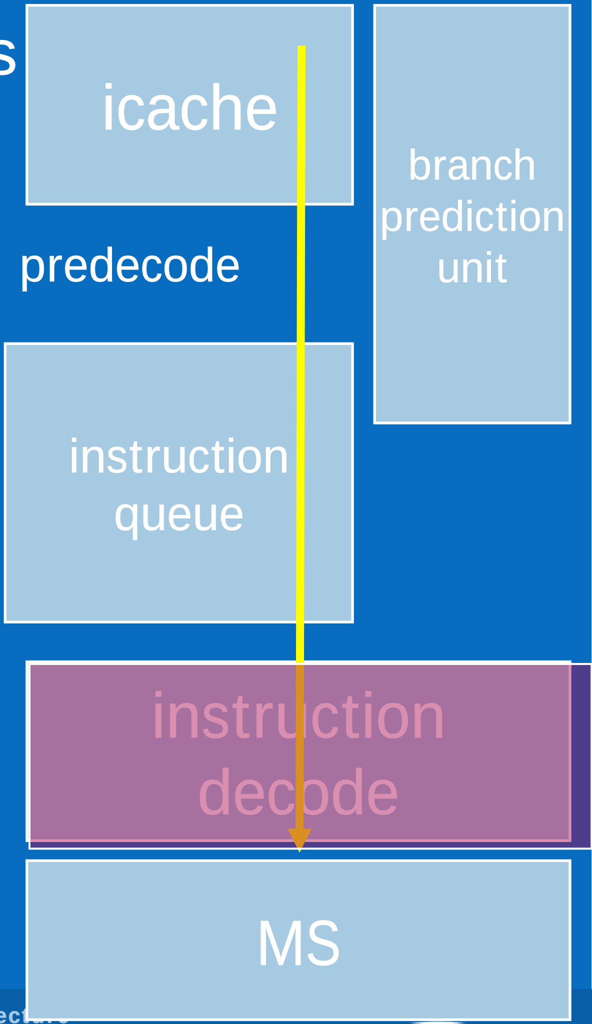
Instruction Decode Unit

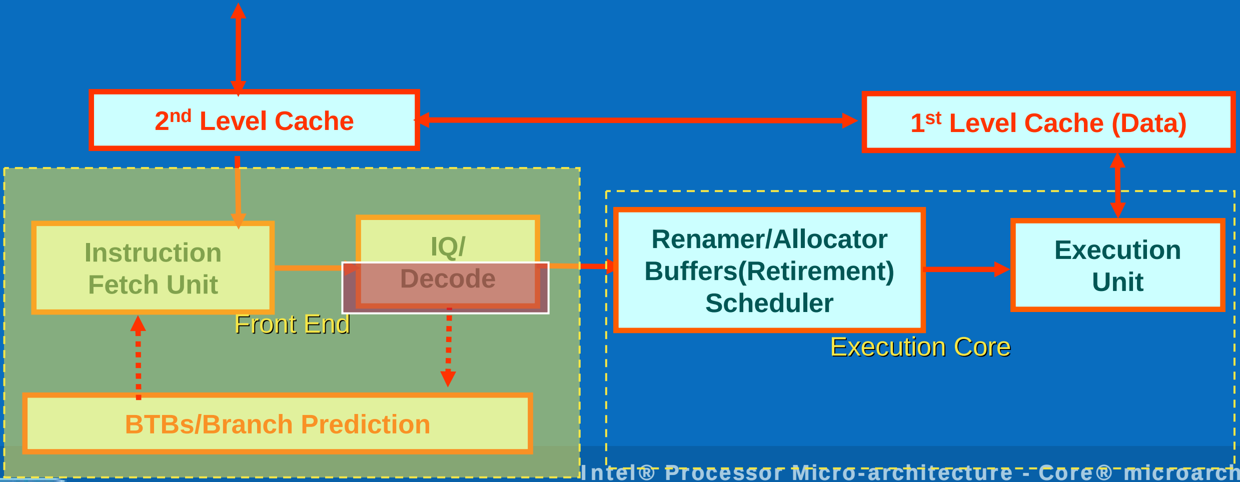
Branch Prediction Unit

Instruction Decode

Decode the instructions into micro-ops

Ready for the execution in OOO core





Instruction Decode

Decoders

Features

- Macro-fusion
- Micro-fusion
- Stack Pointer Tracking

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)

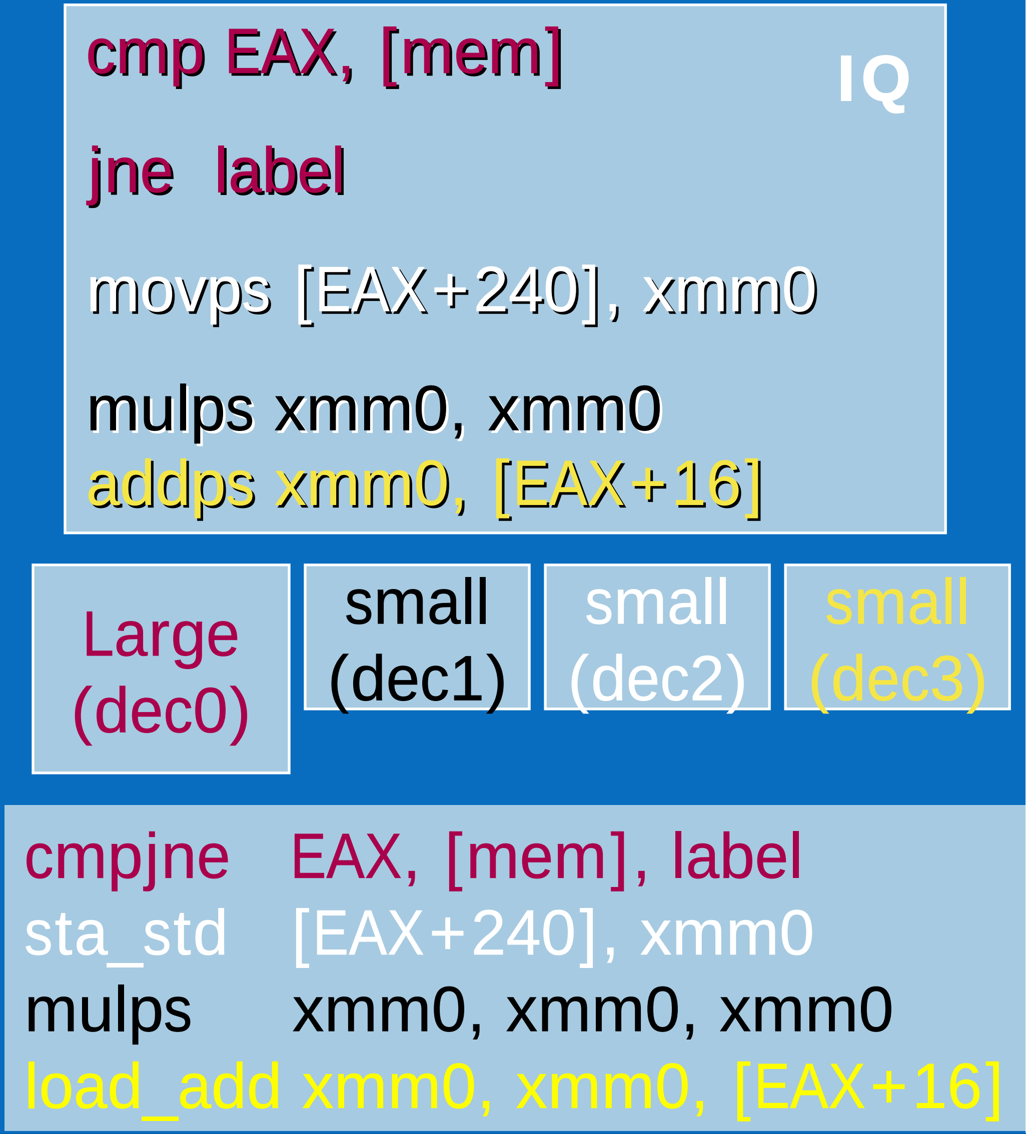
Code Sequence in Front End

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



```
cmp EAX, [mem]
jne  label
movps [EAX+240], xmm0
mulps xmm0, xmm0
addps xmm0, [EAX+16]
```

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

Instruction Decode

Decoders

Features

- Macro-fusion
- Micro-fusion
- Stack Pointer Tracking

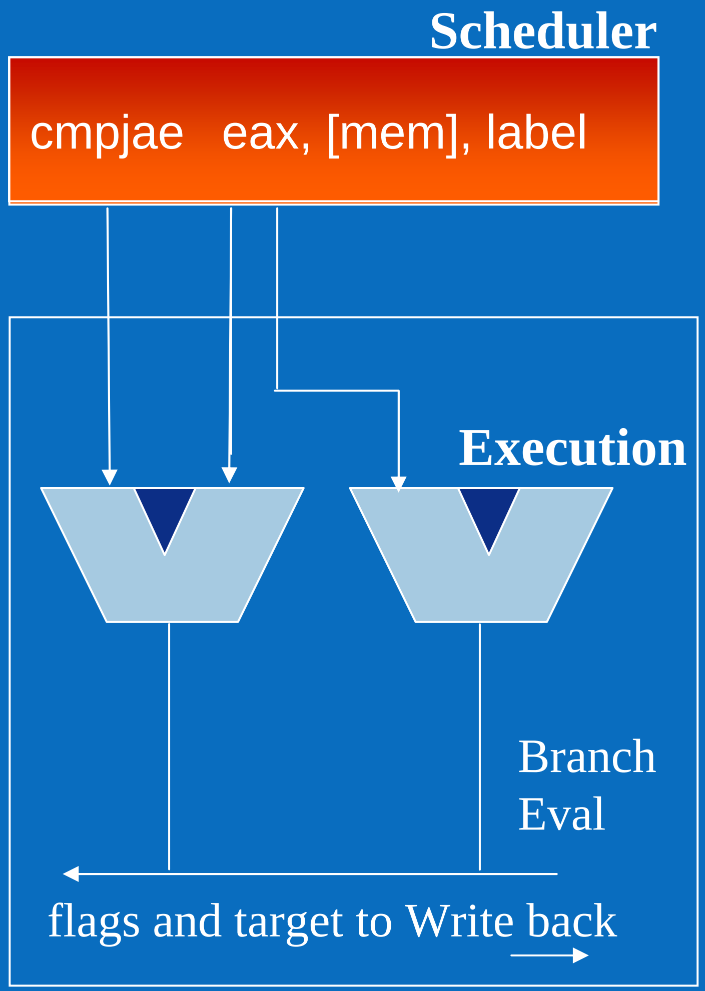
Instruction Decode / Macro - Fusion

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



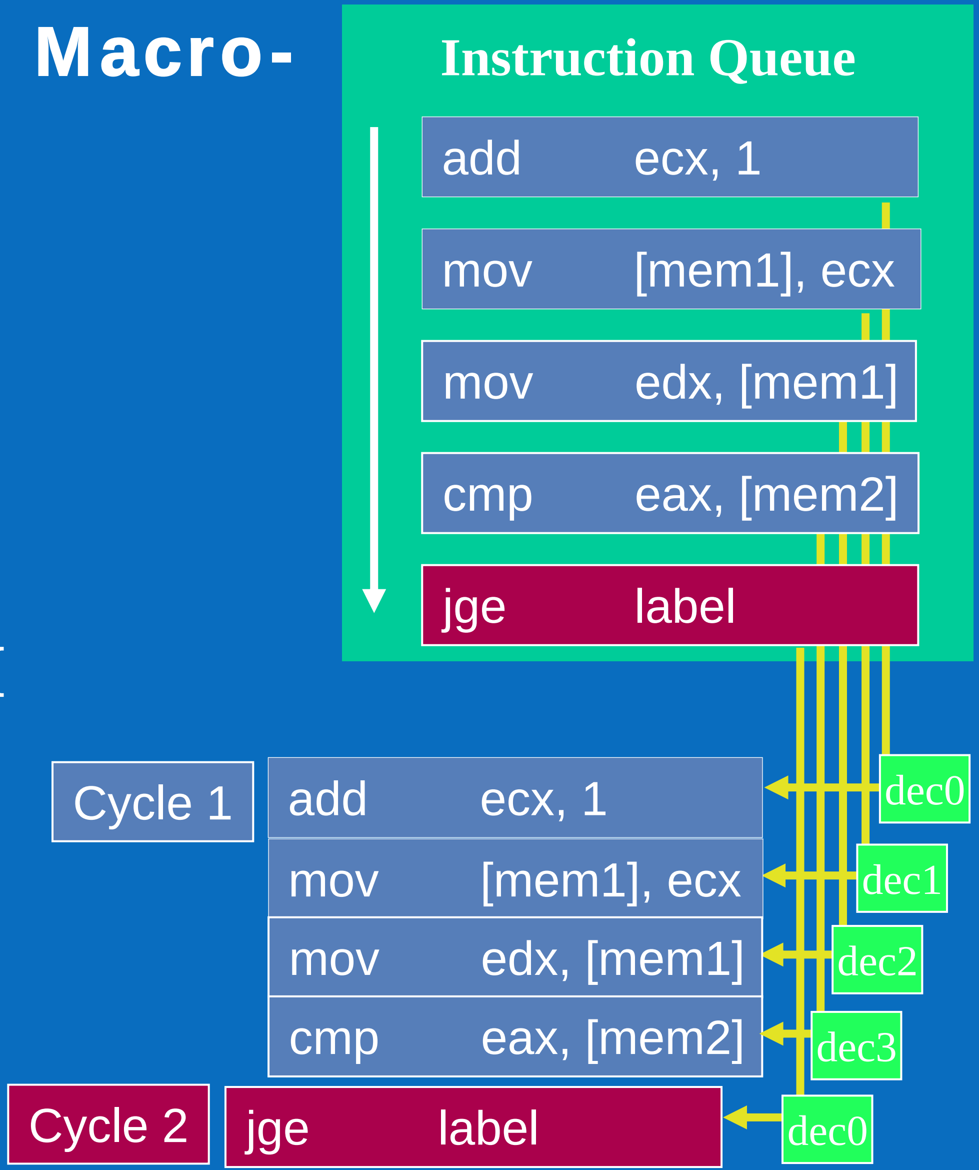
Instruction Decode / Macro-Fusion Absent

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

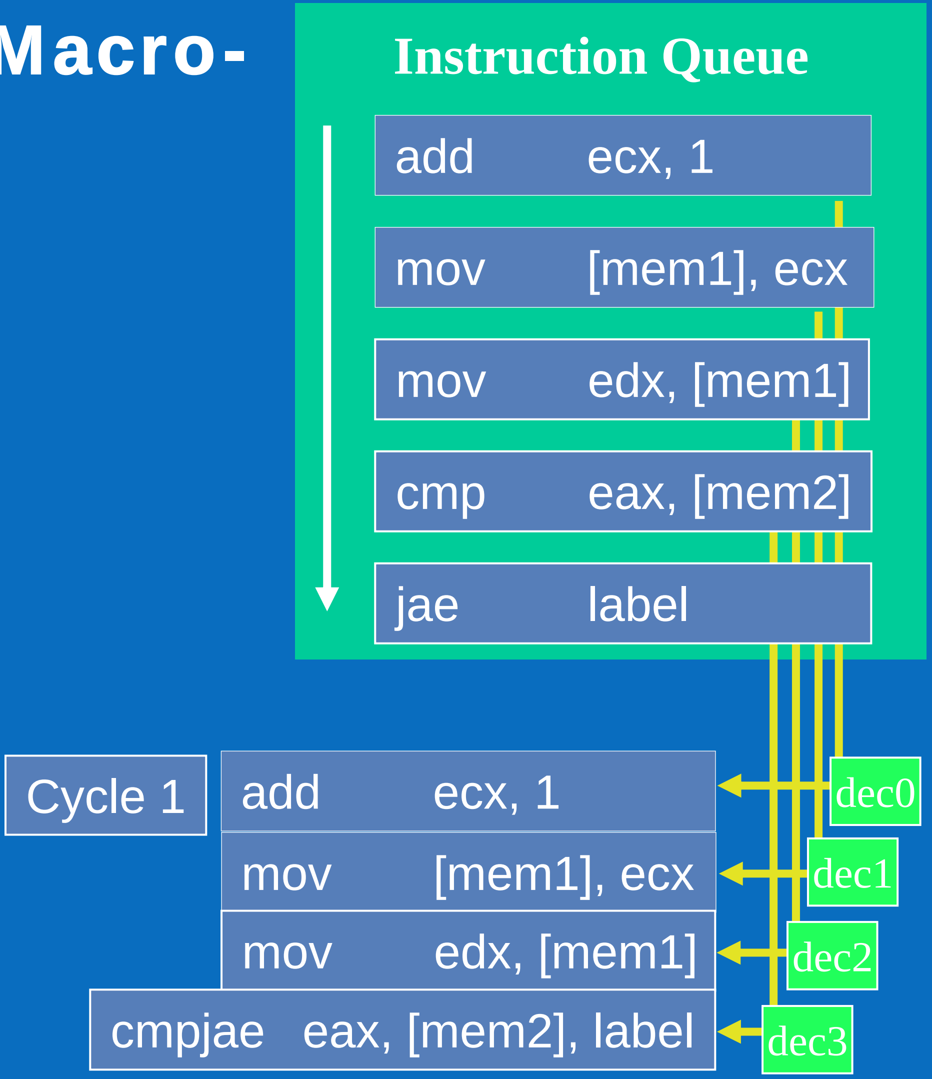
Read five Instructions from Instruction Queue

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

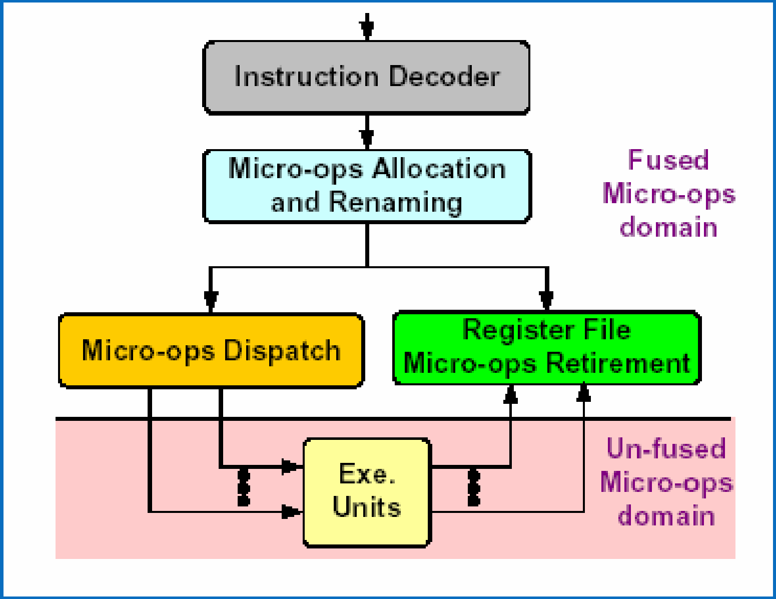
Decoders

Features

- Macro-fusion
- Micro-fusion
- Stack Pointer Tracking

Instruction Decode / Micro-Op Fusion

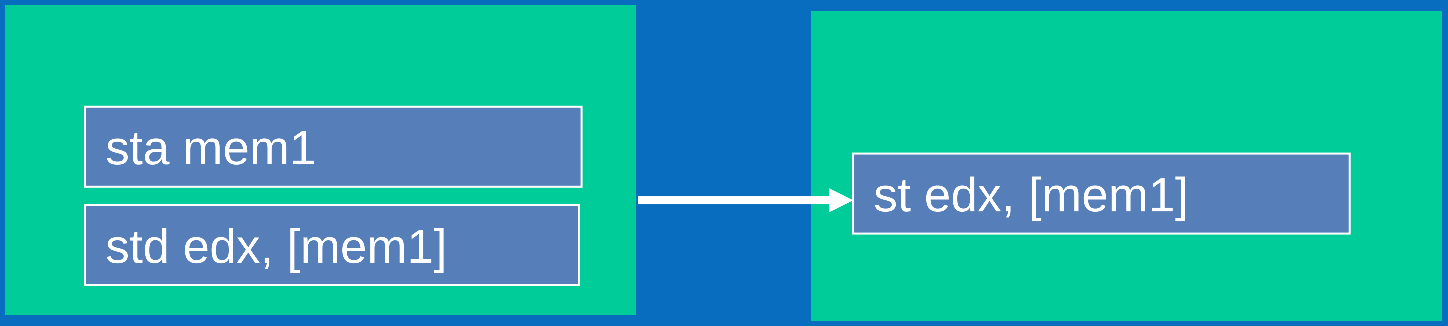
Frequent pairs of micro-operations derived from the same Macro Instruction can be fused into a single micro-operation


Instruction Decoder
Micro-ops Allocation and Renaming
Fused Micro-ops domain
Micro-ops Dispatch
Register File Micro-ops Retirement
Exe. Units
Un-fused Micro-ops domain

Micro-op fusion effectively widens the pipeline

Instruction Decode / Micro-Fusion (cont.)

u-ops of a Store “mov edx, [mem1]”



Instruction Decode

Decoders

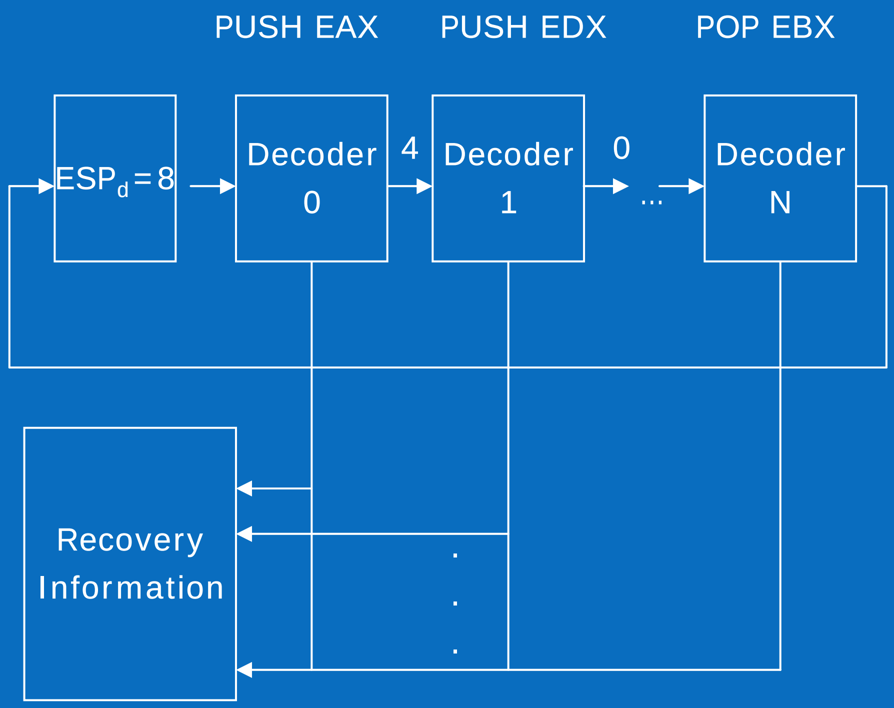
Features

- Macro-fusion
- Micro-fusion
- Stack Pointer Tracking

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



Core® Micro-architecture Front End

Instruction Fetch Unit

Instruction Queue

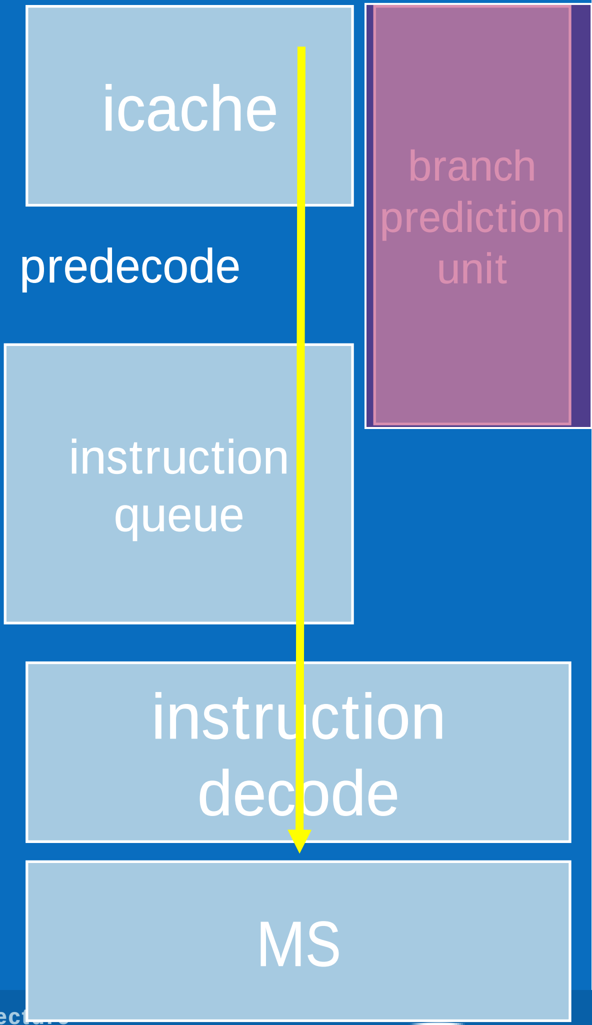
Instruction Decode Unit

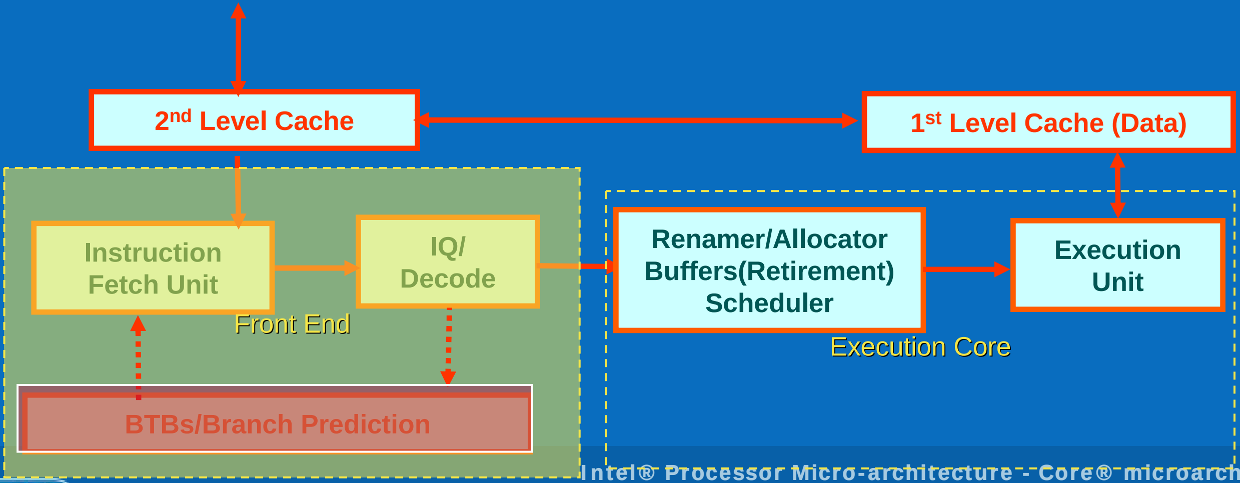
Branch Prediction Unit

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)

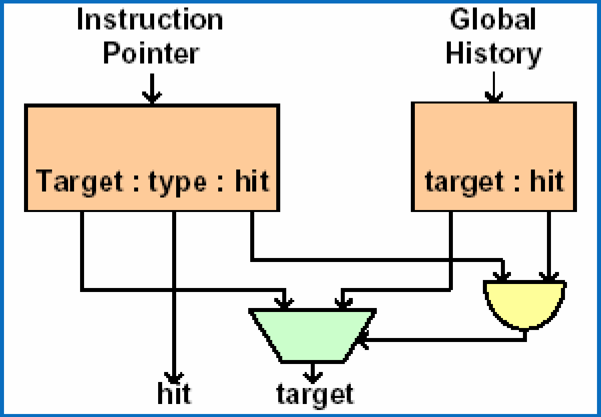
Front end queuing of BPU lookups

Type of predictions

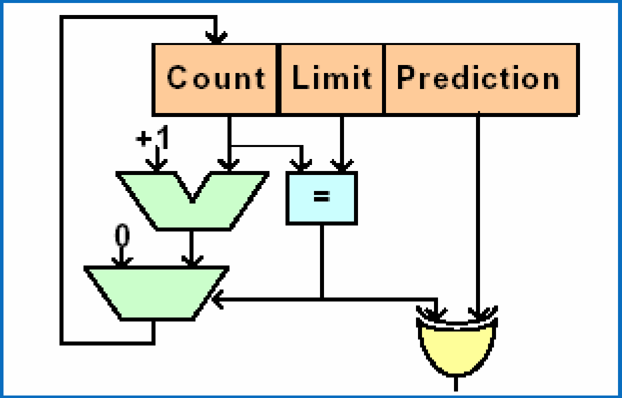
- Direct Calls and Jumps
- Indirect Calls and Jumps
- Conditional branches

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%

Agenda

Introduction

Notable features

Micro-architecture drill-down

- Front End
- Out-Of-Order Execution Core
- Memory Sub-system

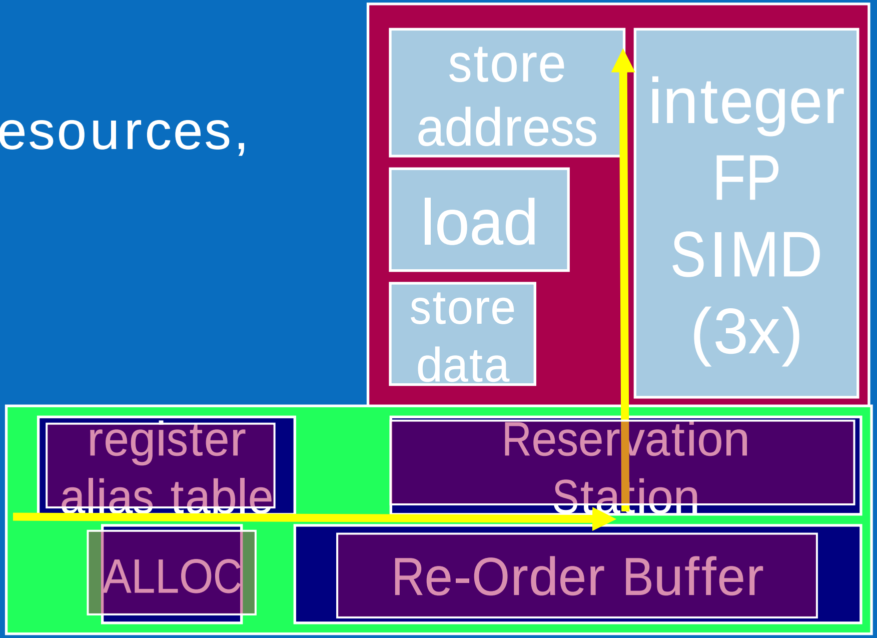
Advanced cache technology

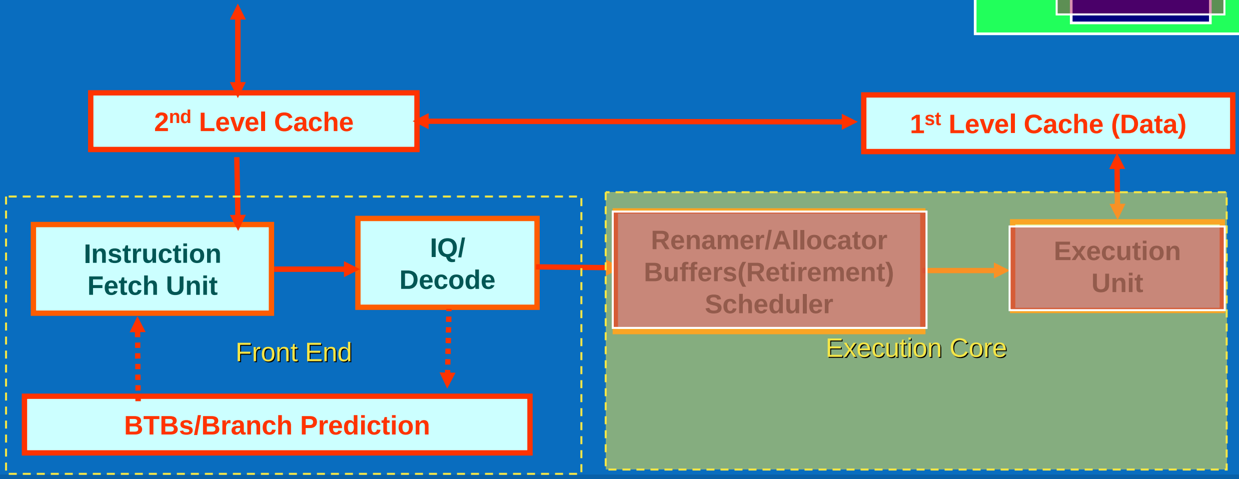
Coding considerations

Core® Micro-architecture Execution Core

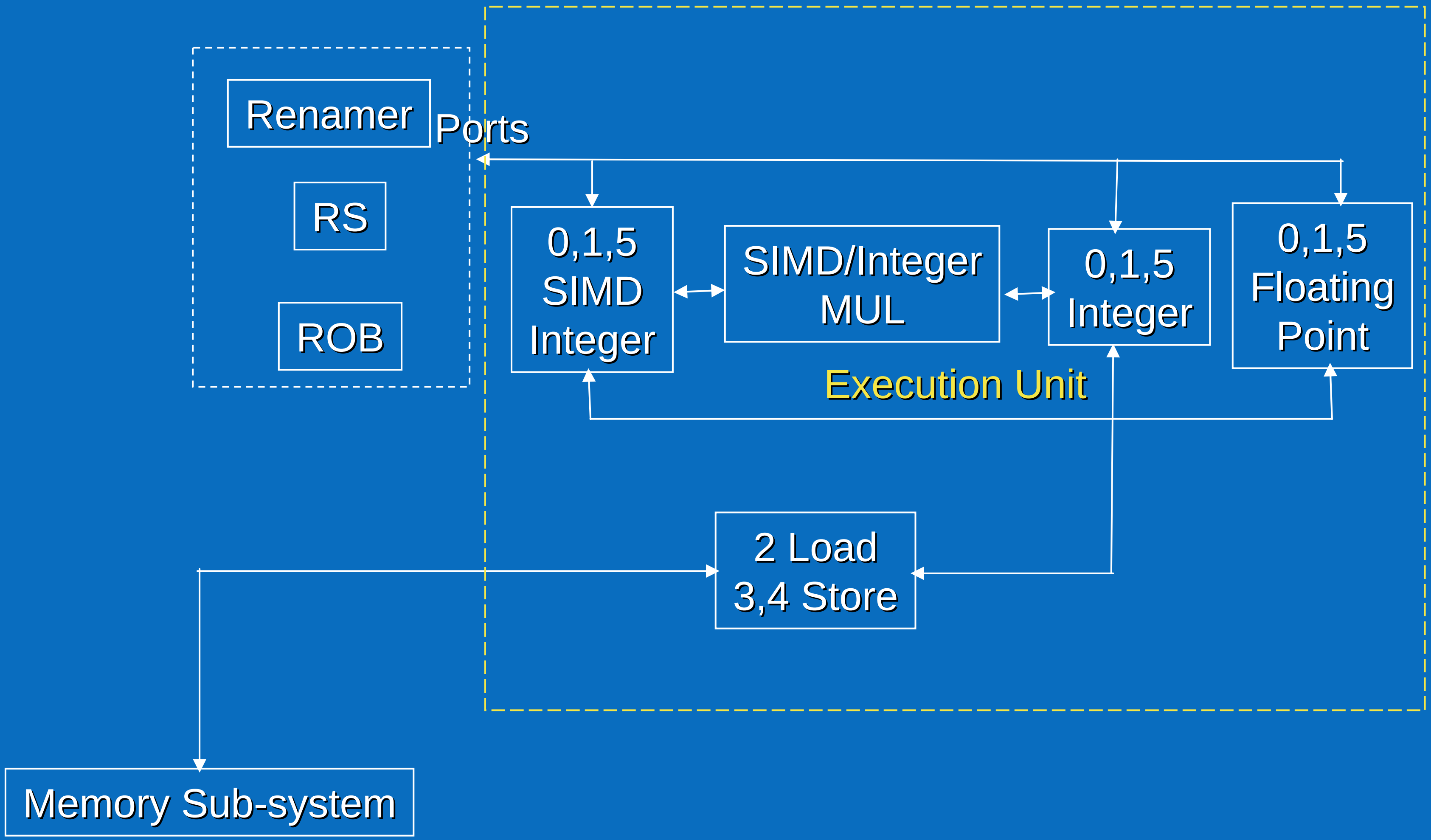
Accepted decoded u-ops, assign resources, execute and retire u-ops

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





Execution Core Building Blocks



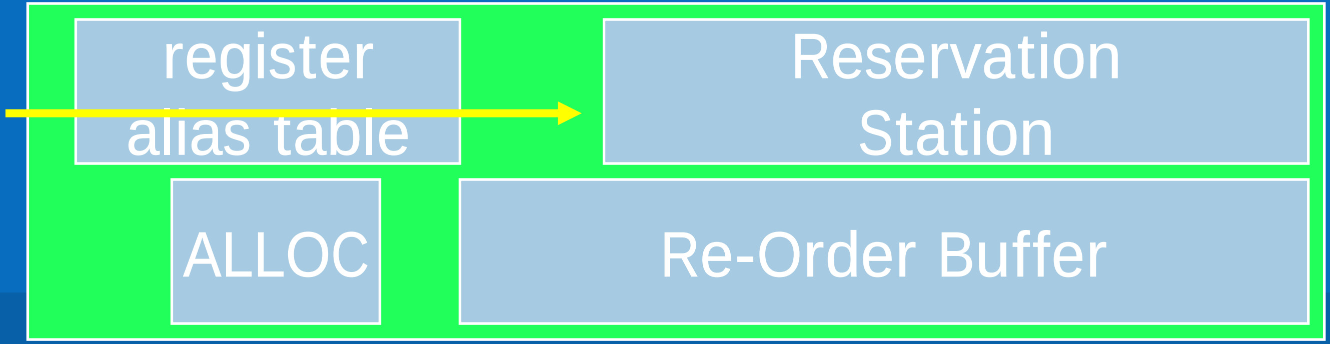
Rename and Resources

4 uops renamed / retired per clock

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

Uops written to RS and ROB

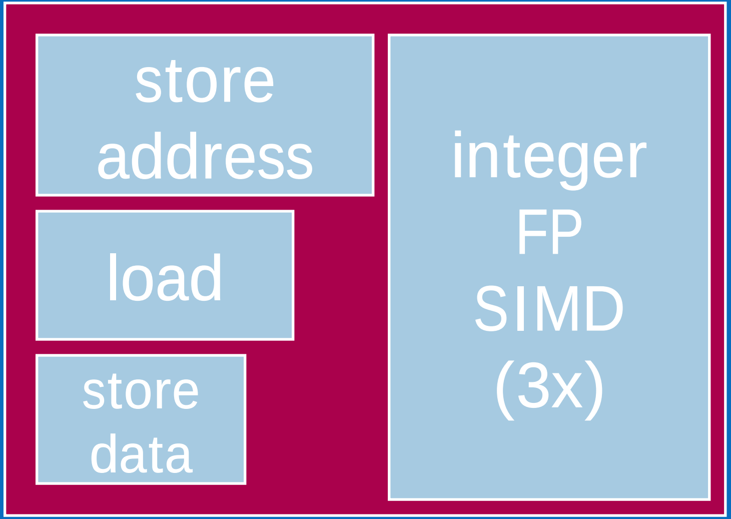
- Decoded uops were renamed and allocated with resource by RAT and sent to ROB read and RS
- RS waits for sources to arrive allowing OOO execution
- Registers not “in flight” read from ROB during RS write



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
         Pand xmm0,xmm3
         Addps xmm2,xmm0
```

```
Better:  Addps XMM0,XMM1
         Addps xmm2,xmm0
         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

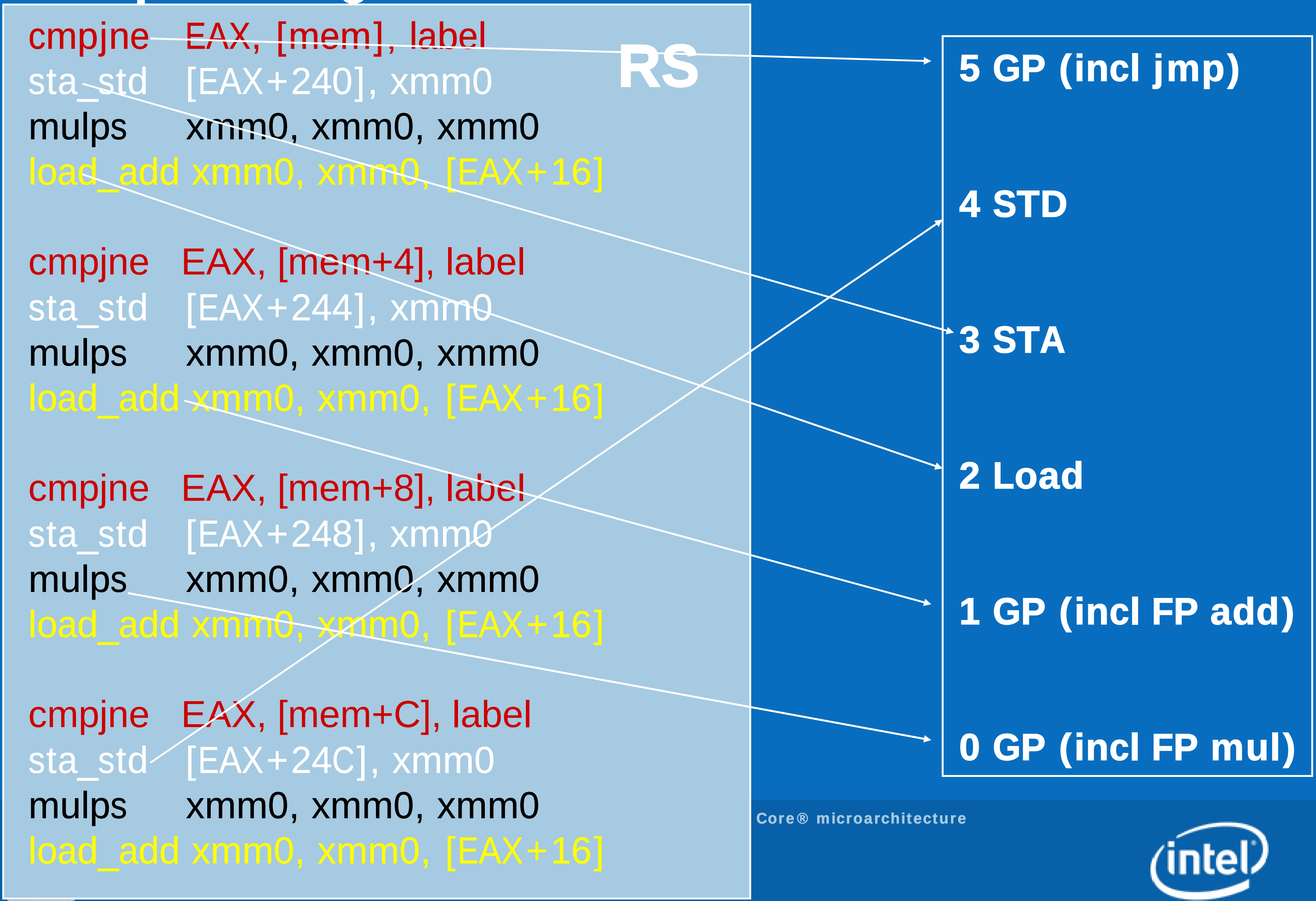
- sta (address) can fire as early as possible
- std must wait for mulps to write back

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

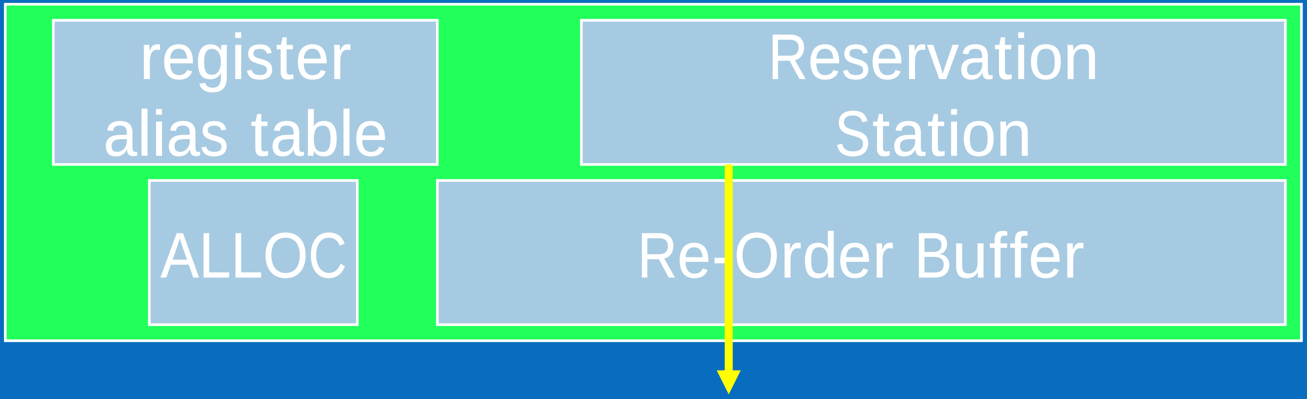


```
cmpjne   EAX, [mem], label
sta_std  [EAX+240], xmm0
mulps    xmm0, xmm0, xmm0
load_add xmm0, xmm0, [EAX+16]

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]
```

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



Agenda

Introduction

Notable features

Micro-architecture drill-down

- Front End
- Out-Of-Order Execution Core
- Memory Sub-system

Advanced cache technology

Coding considerations

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

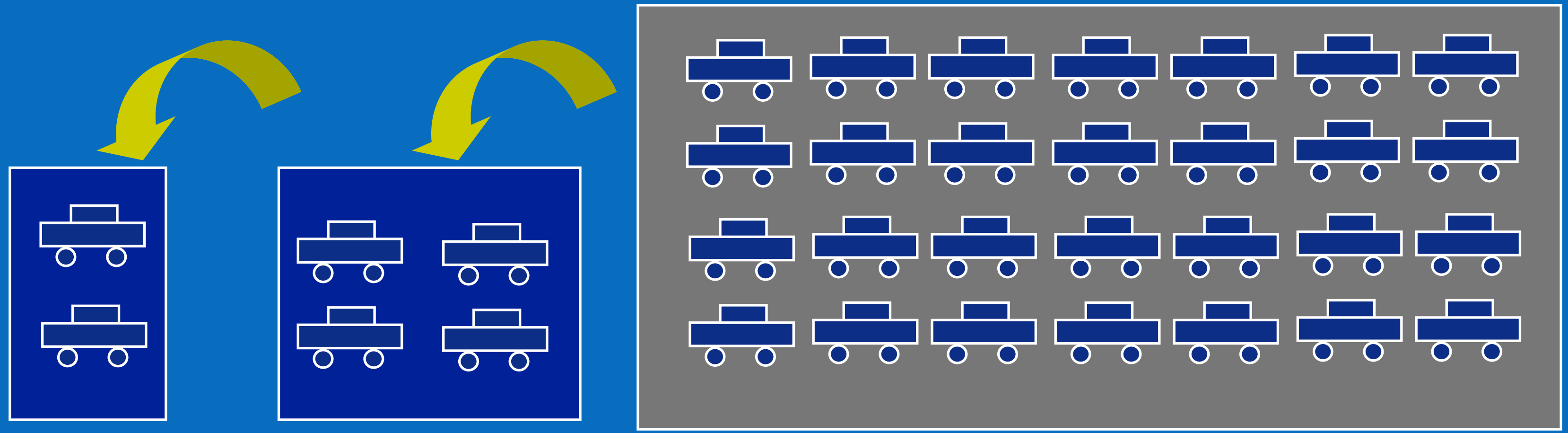
- Bandwidth to cache ~ 8.5 bytes/cycle

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

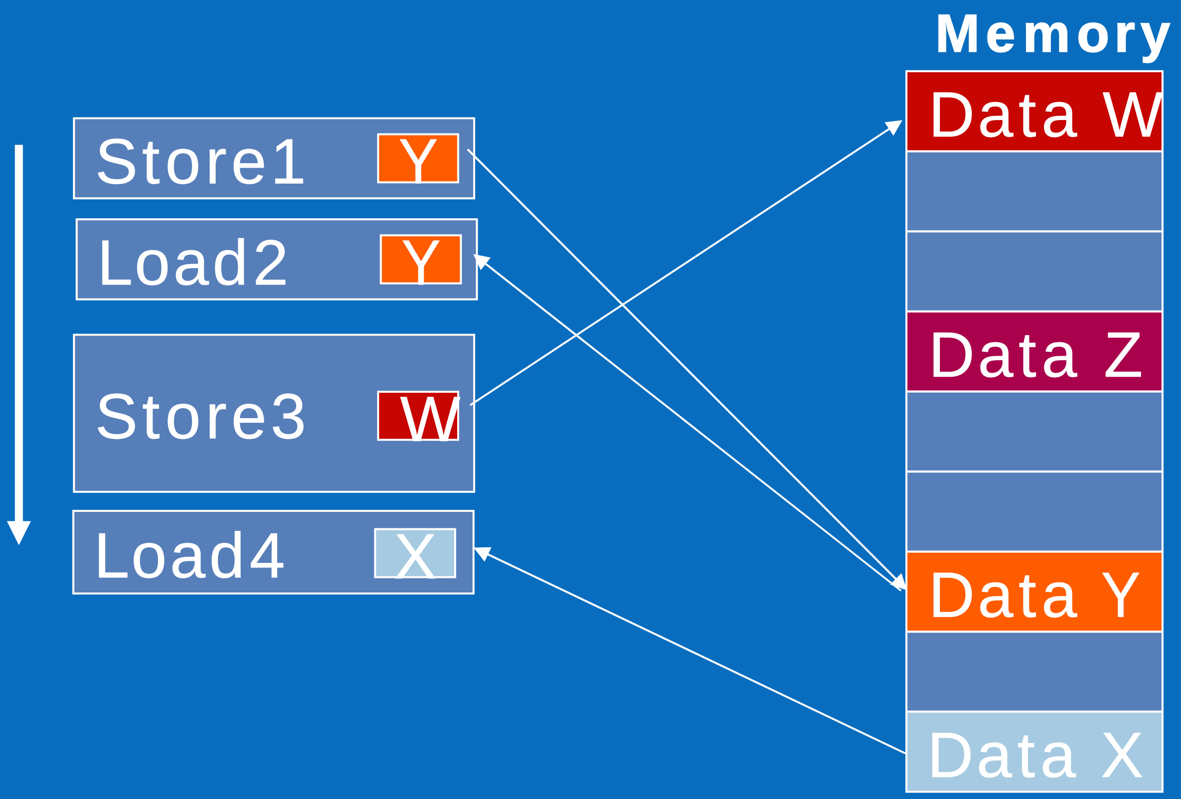
- Loads that are predicted NOT to forward from preceding store are allowed to schedule as early as possible
    - increasing the performance of OOO memory pipelines

Disambiguated loads checked at retirement

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

Advanced Memory Access / Memory Disambiguation Absent

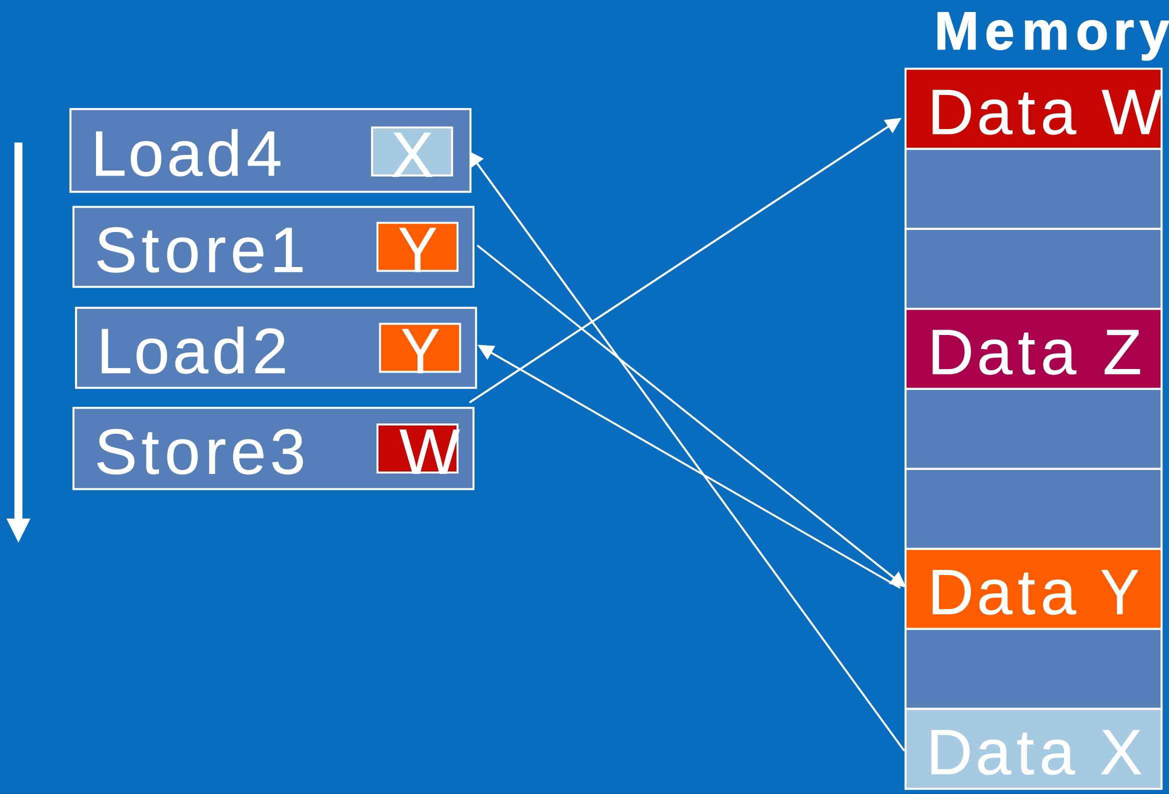
Load4 must WAIT until previous stores complete



Advanced Memory Access / Memory Disambiguation Presented

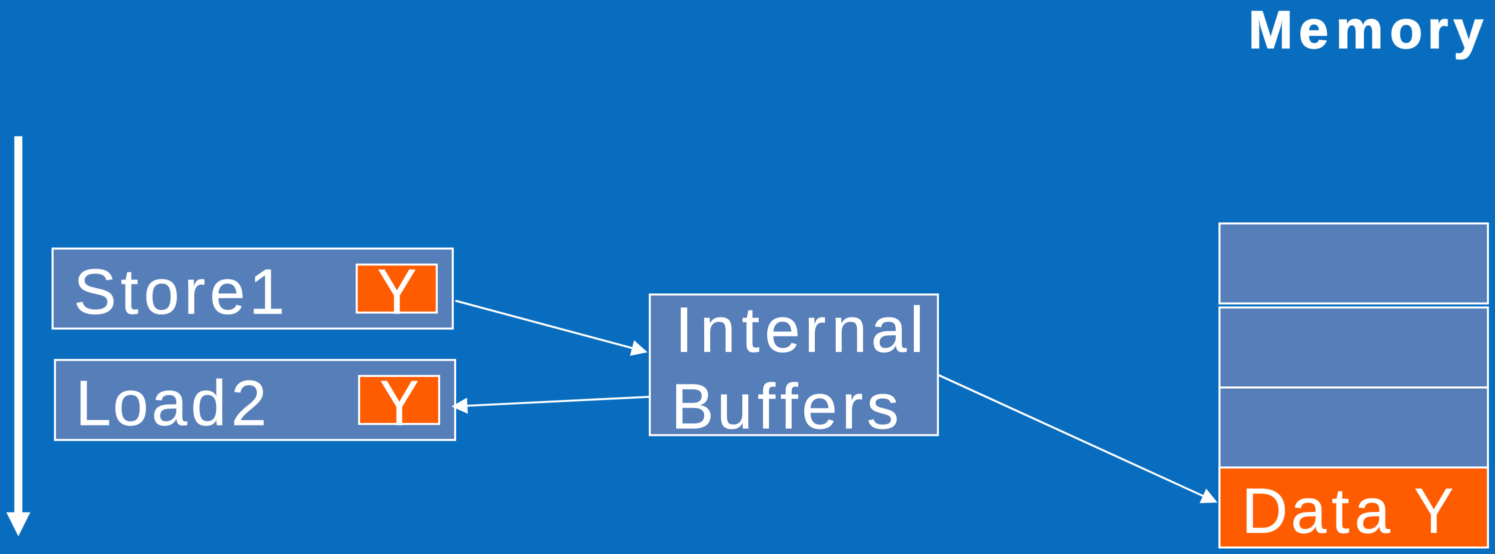
Loads can decouple from stores

Load4 can get its data WITHOUT waiting for stores

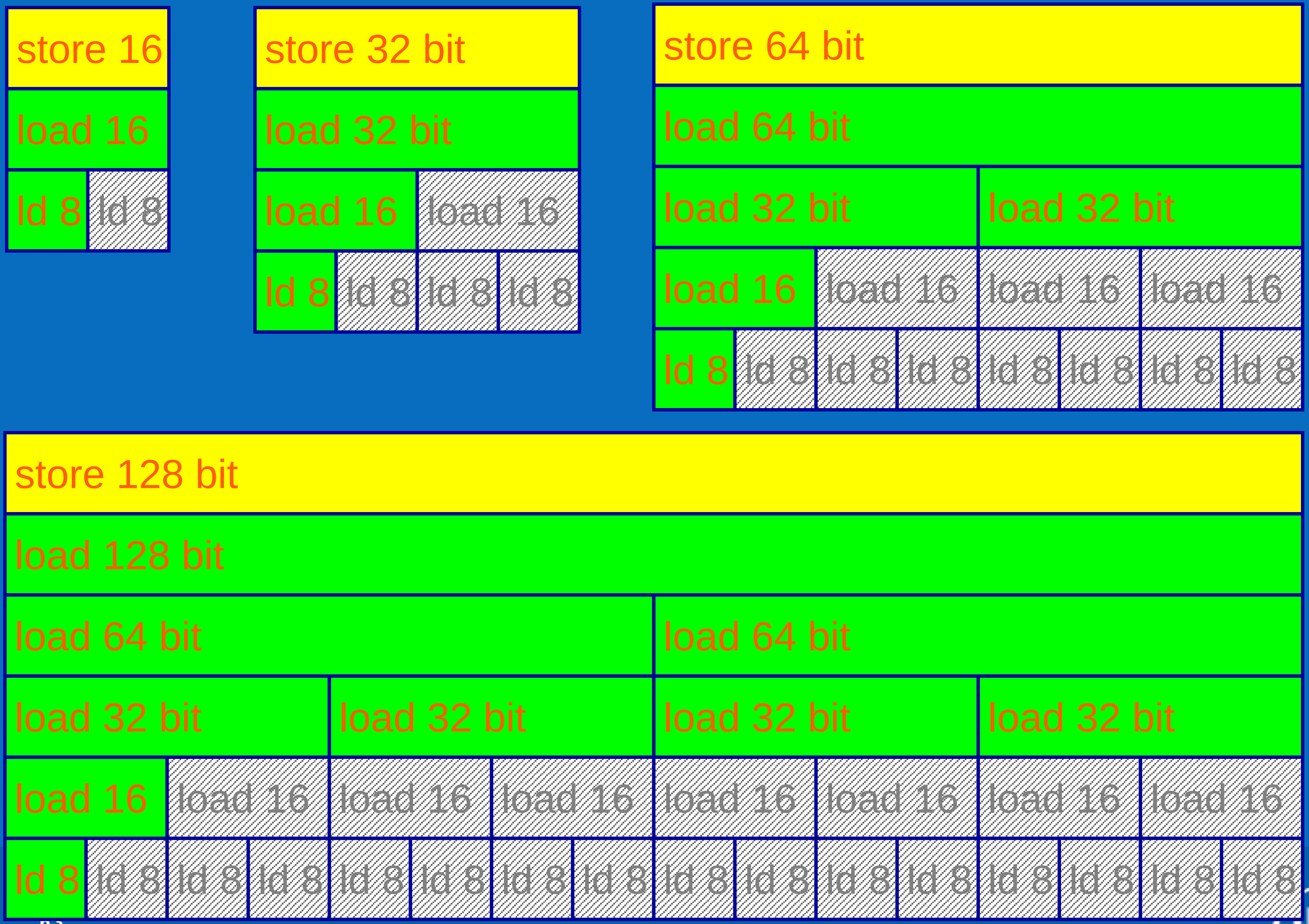


Advanced Memory Access / Stores Forwarding

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

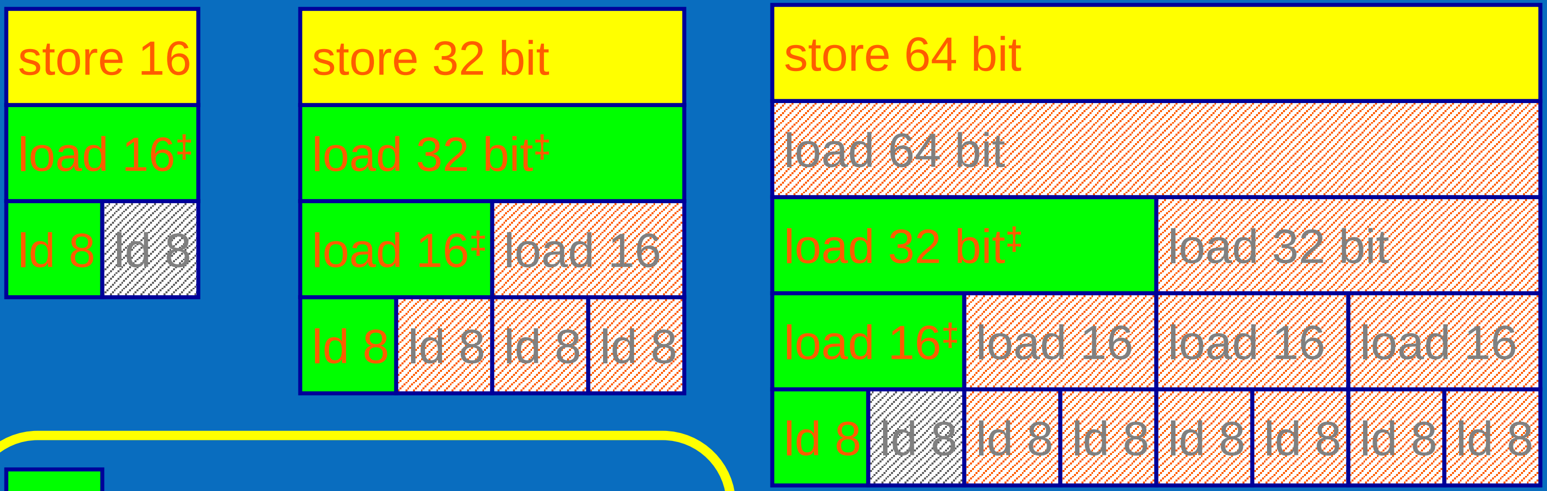


Advanced Memory Access / Stores Forwarding: Aligned Store Cases


store 16
load 16
ld 8 ld 8
store 32 bit
load 32 bit
load 16 load 16
ld 8 ld 8 ld 8 ld 8
store 64 bit
load 64 bit
load 32 bit load 32 bit
load 16 load 16 load 16 load 16
ld 8 ld 8 ld 8 ld 8 ld 8 ld 8 ld 8 ld 8
store 128 bit
load 128 bit
load 64 bit load 64 bit
load 32 bit load 32 bit load 32 bit load 32 bit
load 16 load 16 load 16 load 16 load 16 load 16 load 16 load 16
ld 8 ld 8 ld 8 ld 8 ld 8 ld 8 ld 8 ld 8 ld 8 ld 8 ld 8 ld 8 ld 8 ld 8 ld 8 ld 8

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

Agenda

Introduction

Notable features

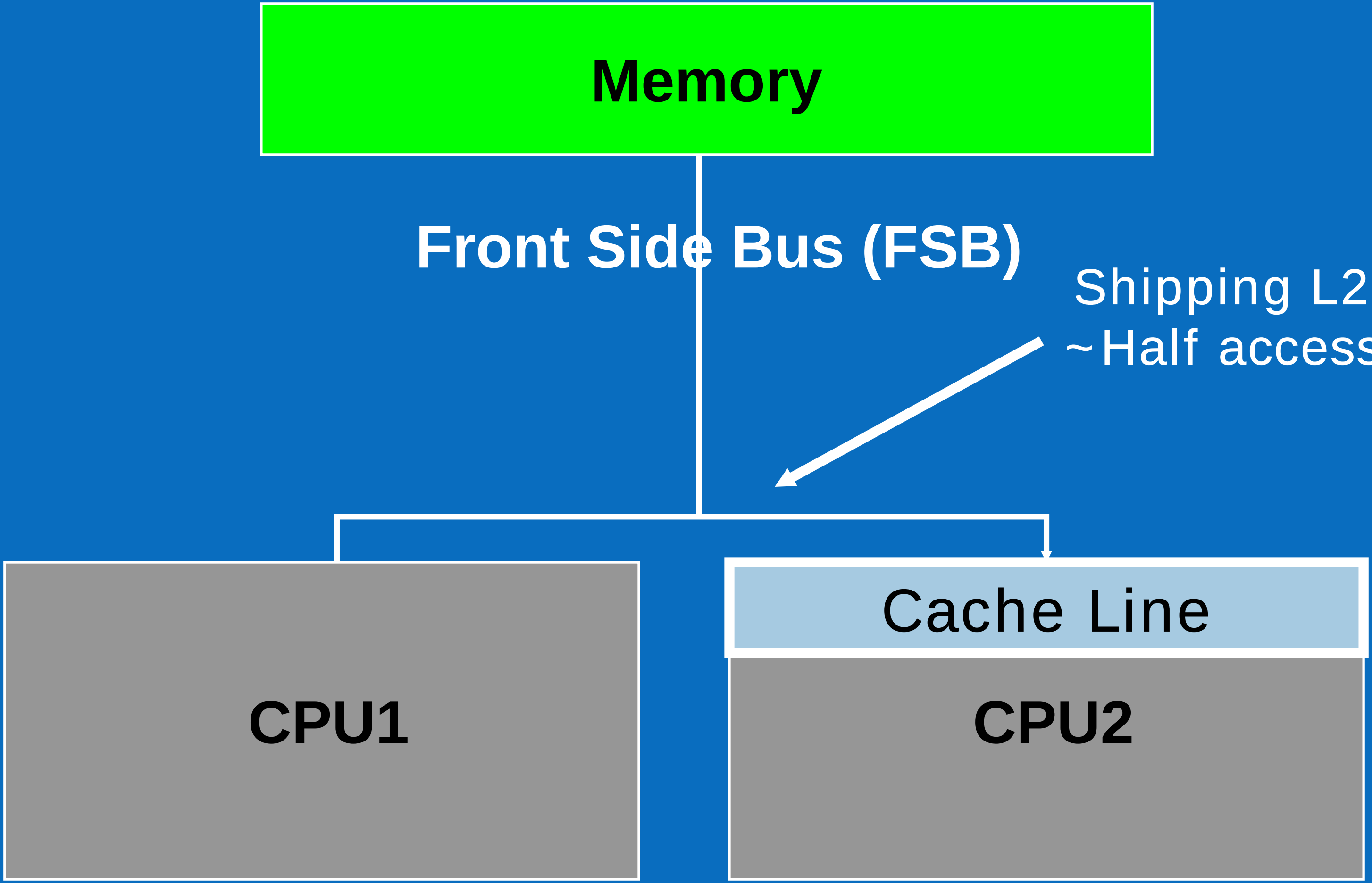
Micro-architecture drill-down

- Front End
- Out-Of-Order Execution Core
- Memory Sub-system

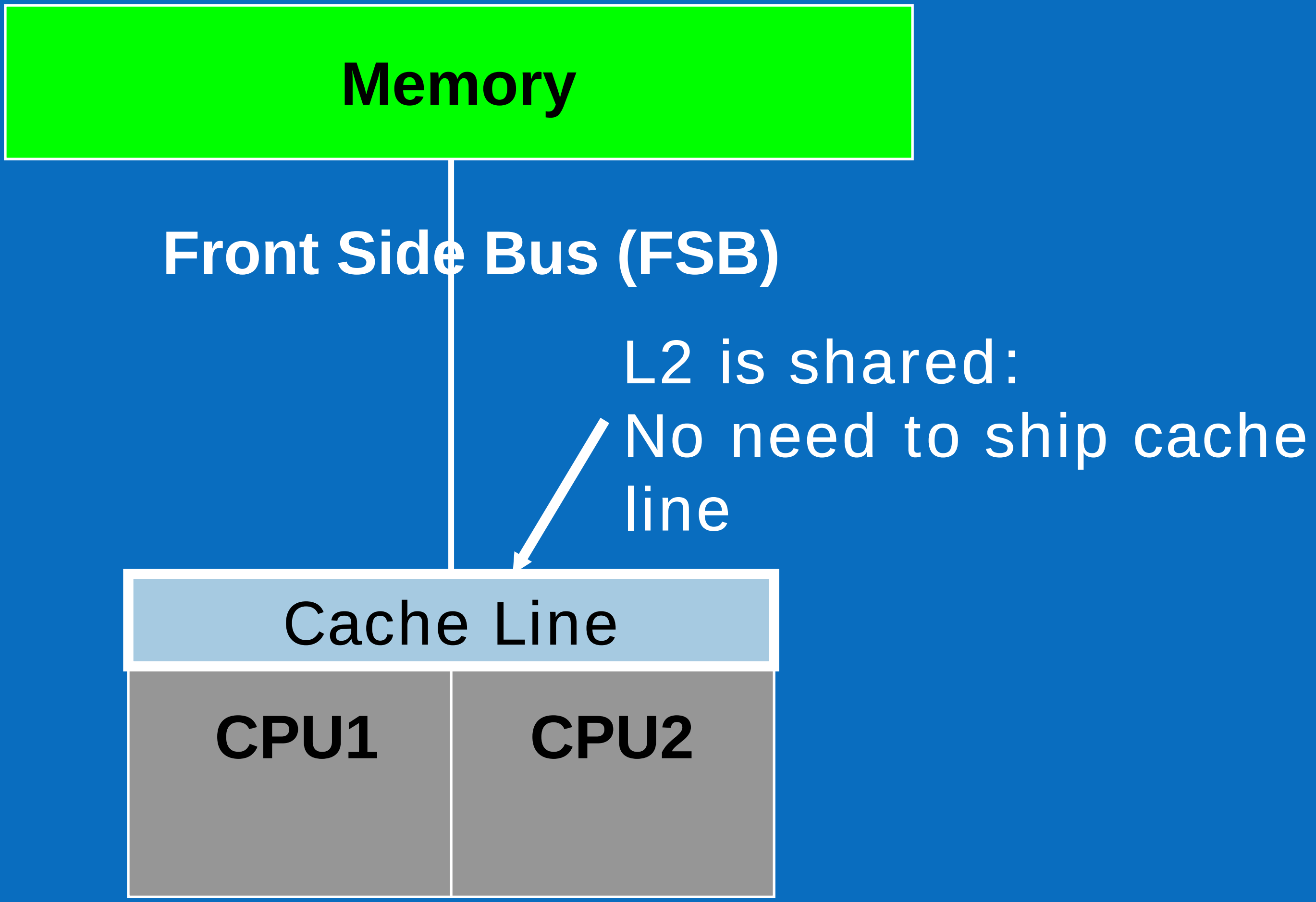
Advanced cache technology

Coding considerations

Advanced Smart Cache® Technology: Advantages of Shared Cache



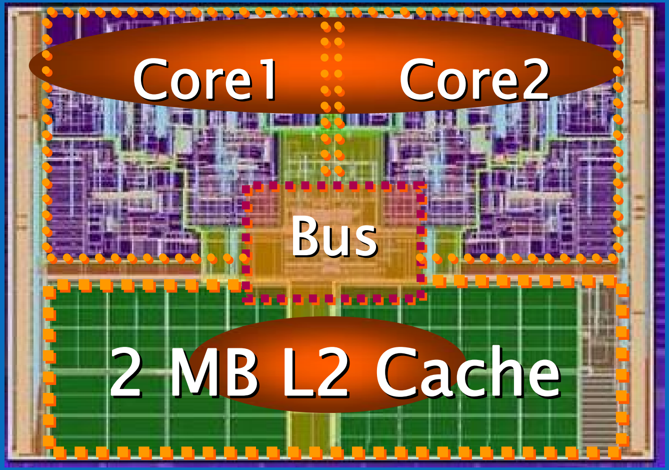
Advanced Smart Cache® Technology: Advantages of Shared Cache (cont.)



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

Agenda

Introduction

Notable features

Micro-architecture drilldown

Advanced cache technology

Coding considerations

Optimizing for Instruction Fetch and PreDecode

Avoid “Length Changing Prefixes” (LCPs)

- Affects instructions with immediate data *or offset*
- Operand Size Override (66H)
- *Address Size Override (67H) [obsolete]*
- LCPs change the length decoding algorithm – increasing the processing time from one cycle to six cycles (or eleven cycles when the instruction spans a 16-byte boundary)
- The REX (EM64T) prefix (4xH) is not an LCP
    - The REX prefix does lengthen the instruction by one byte, so use of the first eight general registers in EM64T is preferred

Optimizing for Instruction Queue

Includes a "Loop Stream Detector" (LSD)

- Potentially very high bandwidth instruction streaming
- A number of requirements to make use of the LSD
  - Maximum of 18 instructions in up to four 16-byte packets
  - No RET instructions (hence, little *practical* use for CALLs)
  - Up to four taken branches allowed
  - Most effective at 70+ iterations
- LSD is after PreDecode so there is no added cost for LCPs
- Trade-off LSD with conventional loop unrolling

Optimizing for Decode

Decoder issues up to 4 uOps for renaming/ allocation per clock

- This creates a trade off between more complex instruction uOps versus multiple simple instruction uOps
- For example, a single four uOp instruction is all that can be renamed/allocated in a single clock
- In some cases, multiple simple instructions may be a better choice than a single complex instruction
- Single uOp instructions allow more decoder flexibility
    - For example, 4-1-1-1 can be *decoded* in one clock
    - However, 2-2-2-1 takes three clocks to *decode*

Optimizing for Execution

Up to six uOps can be dispatched per clock

- “Store Data” and “Store Address” dispatch ports are combined on the block diagram

Up to four results can be written back per clock

Single clock latency operations are best

- Differing latency operations can create writeback conflicts
- Separate multiple-clock uOps with several single uOp instructions
  - Typical instructions here: ADC/SBB, RWM, CMOVcc
- In some cases, separating a RMW instruction into its piece might be faster (decode and scheduling flexibility)

When equivalent, PS preferred to PD (LCP)

- For example, MOVAPS over MOVAPD, XORPS over XORPD

Optimizing for Execution (cont.)

Bypass register "access" preferred to register reads

Partial register accesses often lead to stalls

- Register size access that 'conflicts' with recent previous register write
- Partial XMM updates subject to dependency delays
- Partial flag stall can occur, too → much higher cost
  - Use TEST instruction between shift and conditional to prevent
- Common zeroing instructions (e.g., XOR reg,reg) don't stall

Avoid bypass *between* execution domains

- For example: FP (ADDPS) and logical ops (PAND) on XMM*n*

Vectorization: careful packing/unpacking sequence

- Use MXCSR's FZ and DAZ controls as appropriate

Optimizing for Memory

Software prefetch instructions

- Can reach beyond a page boundary (including page walk)
- Prefetches only when it completes without an exception

General techniques to help these prefetchers

- Organize data in consecutive lines
- In general, increasing addresses are more easily prefetched

Summary

What has been covered

- Notable features of Core® Micro-architecture
  - Wide Dynamic Execution
  - Advanced Memory Access
  - Advanced Smart Cache
  - Advanced Digital Media Boost
  - Power Efficient Support
- Core® Micro-architecture components
  - Front End
  - OOO execution core
  - Memory sub-system
- Advanced cache technology

intel®

Leap ahead™

Reasons Why Software Prefetches Can Negatively Impact Performance on Core2 Duo Architecture

Software prefetches are rarely ignored on Merom Architecture

- On P4 if you had a DTLB miss the prefetch could be ignored
- On Merom architecture they are not ignored and the prefetch can hurt performance since it cannot retire until after the page walk

Critical chunk is not utilized on a software prefetch

- A prefetch can hurt performance if it is too close to the load
- When the data comes in due to a prefetch you need the entire cache line instead of just the critical chunk before the data can be used by the actual load

Software prefetches can trigger hardware prefetching mechanisms

- Trigger the hardware prefetchers just like a regular load
- False patterns can be found if you prefetch on the wrong data

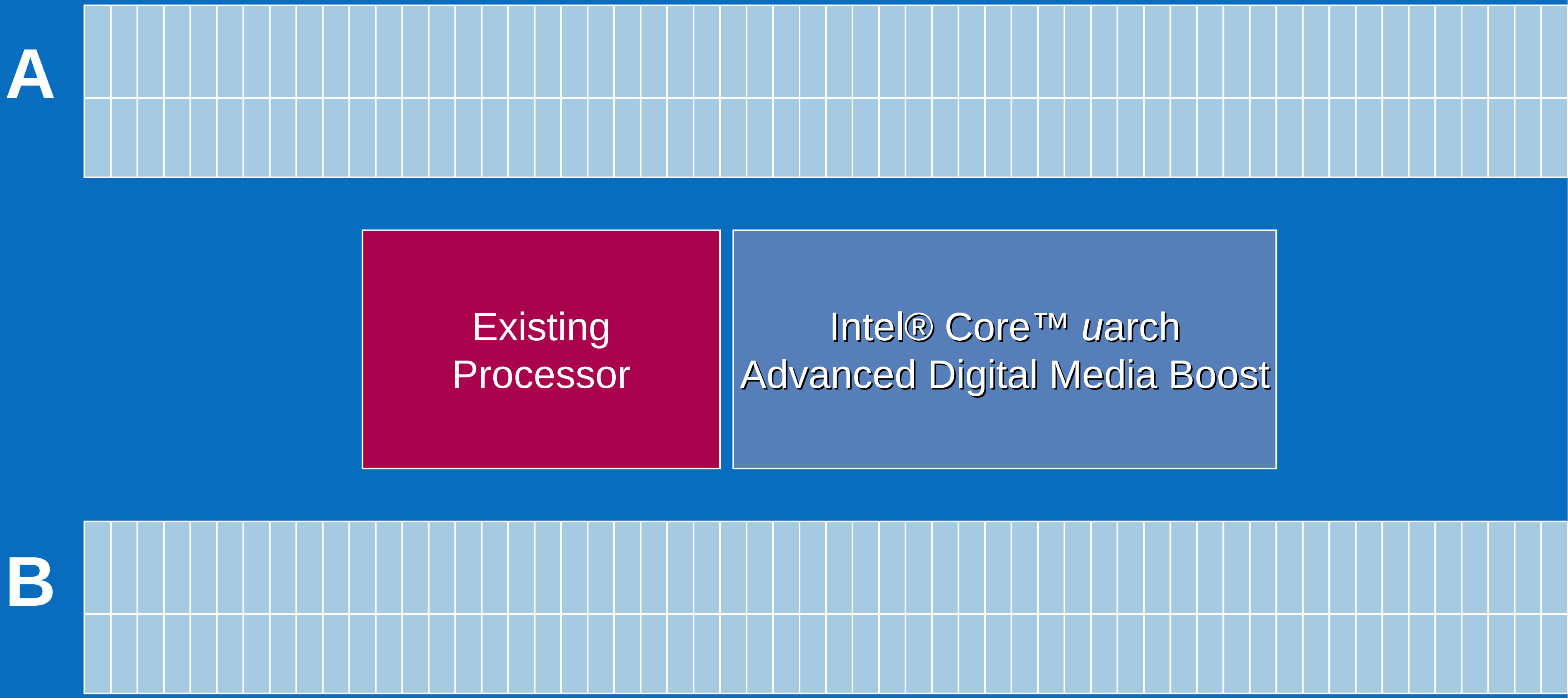
- Software prefetches can saturate the bus

Not Guaranteed to Behave Cross Architecture

Performance can vary between architectures

Advanced Digital Media Boost

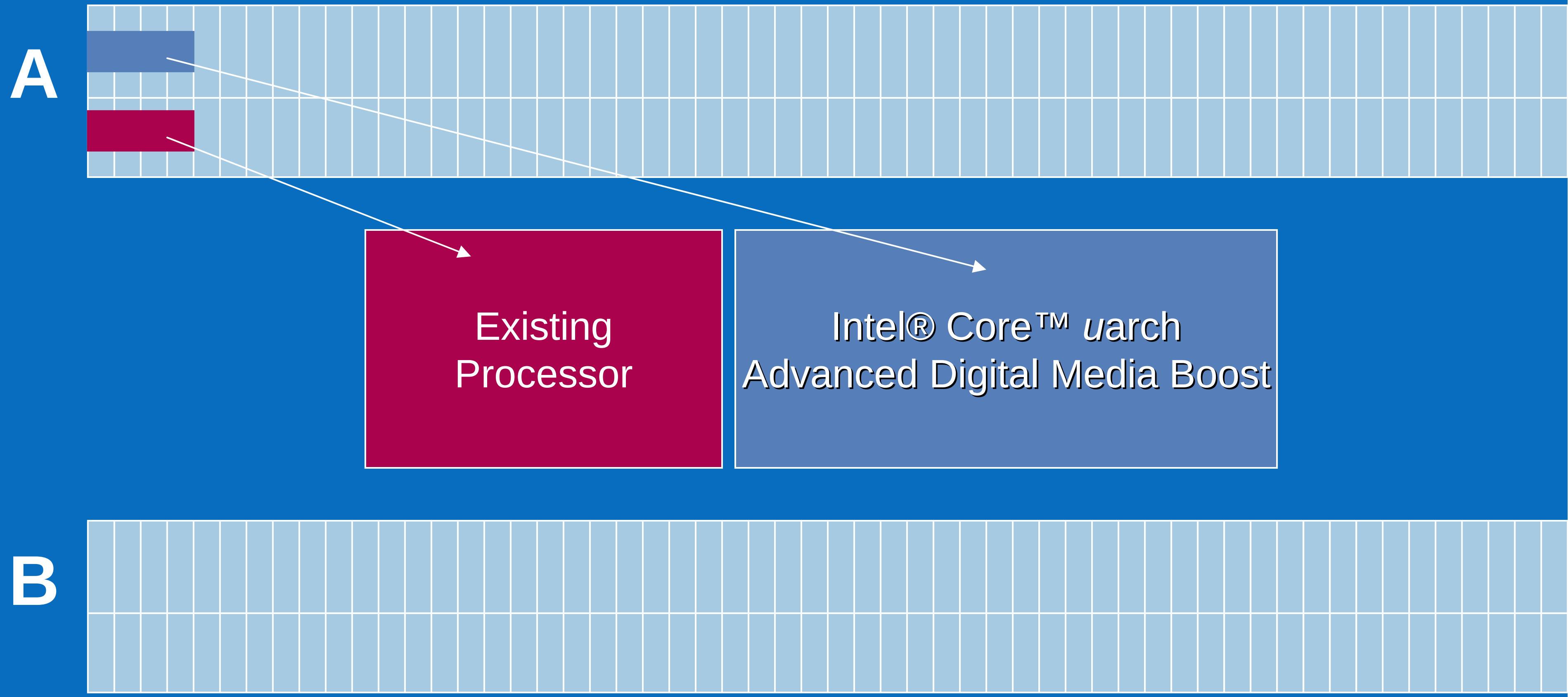
Lets scale a vector: B[i] := A[i] * C



2x Compute Throughput / Clock

Advanced Digital Media Boost

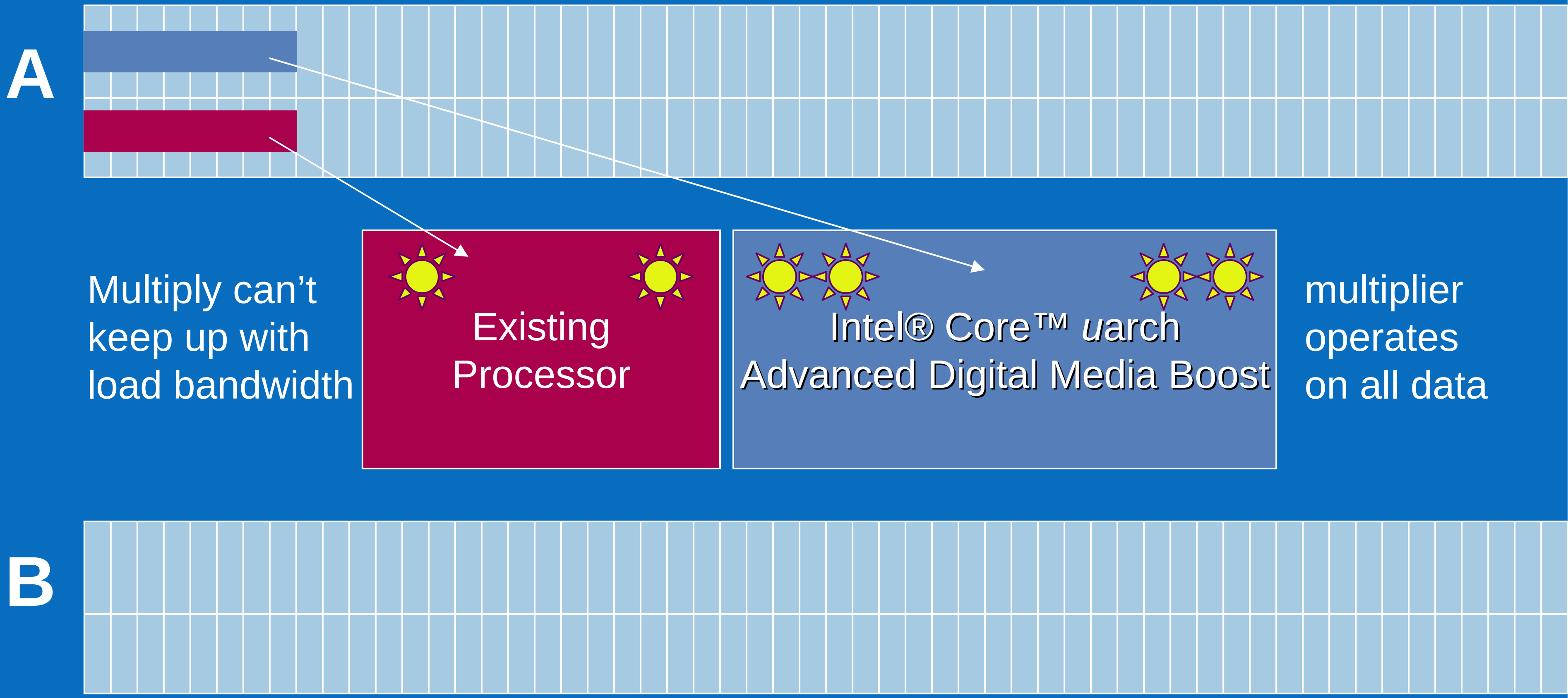
Assume both microarchs have 128-bit path from L1 to Processor



2x Compute Throughput / Clock

Advanced Digital Media Boost

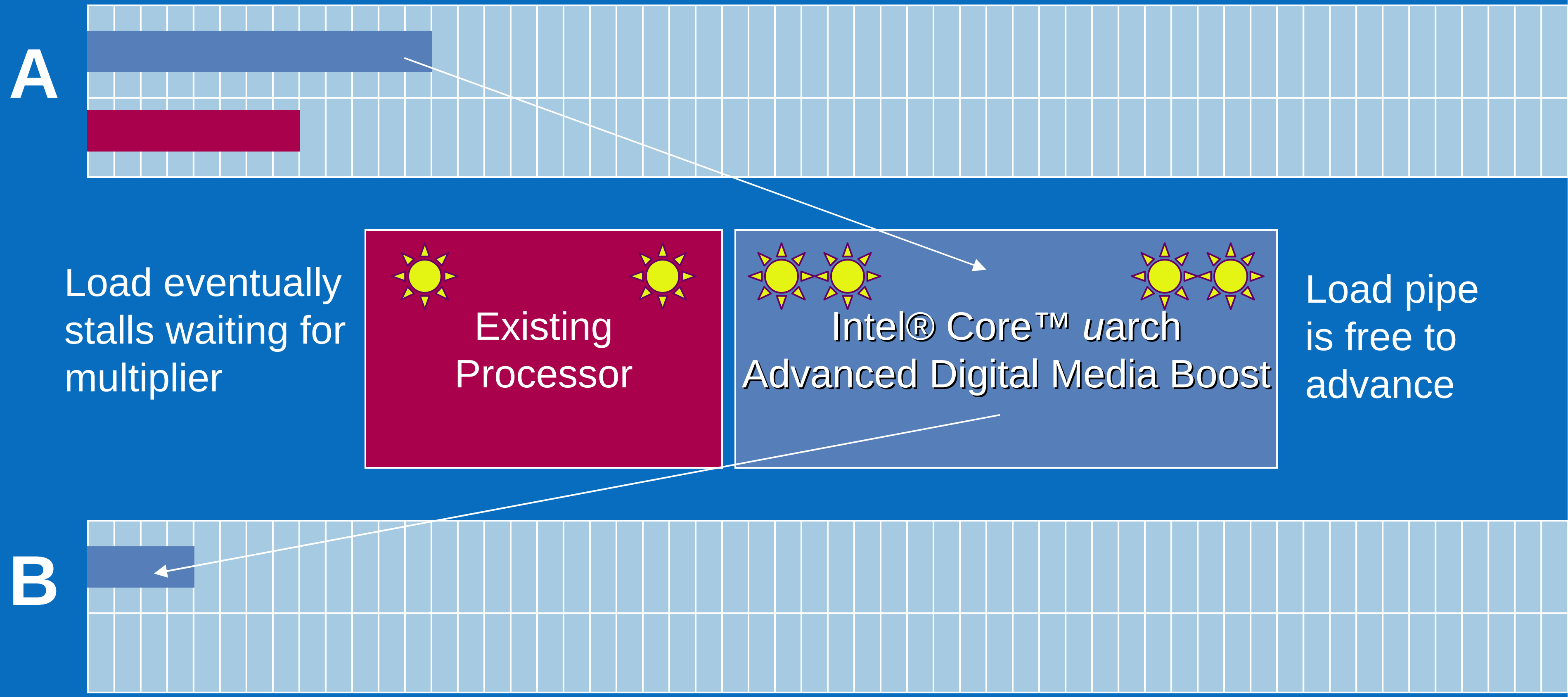
...handles all the memory data


A
Multiply can't keep up with load bandwidth
Existing Processor
Intel® Core™ *uarch* Advanced Digital Media Boost
multiplier operates on all data
B

2x Compute Throughput / Clock

Advanced Digital Media Boost

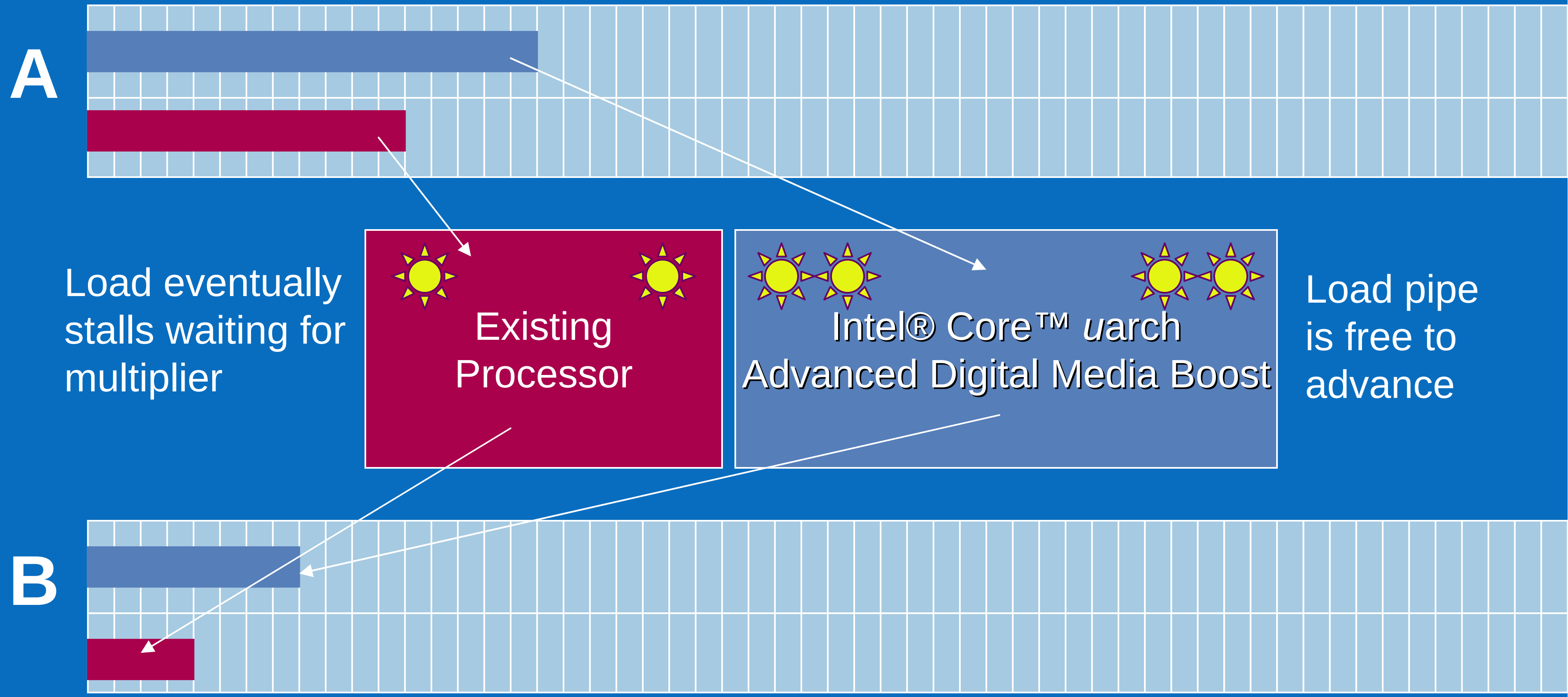
Existing implementations eventually stall the load pipe waiting for multiplier


A
B
Load eventually stalls waiting for multiplier
Existing Processor
Intel® Core™ uarch Advanced Digital Media Boost
Load pipe is free to advance

2x Compute Throughput / Clock

Advanced Digital Media Boost

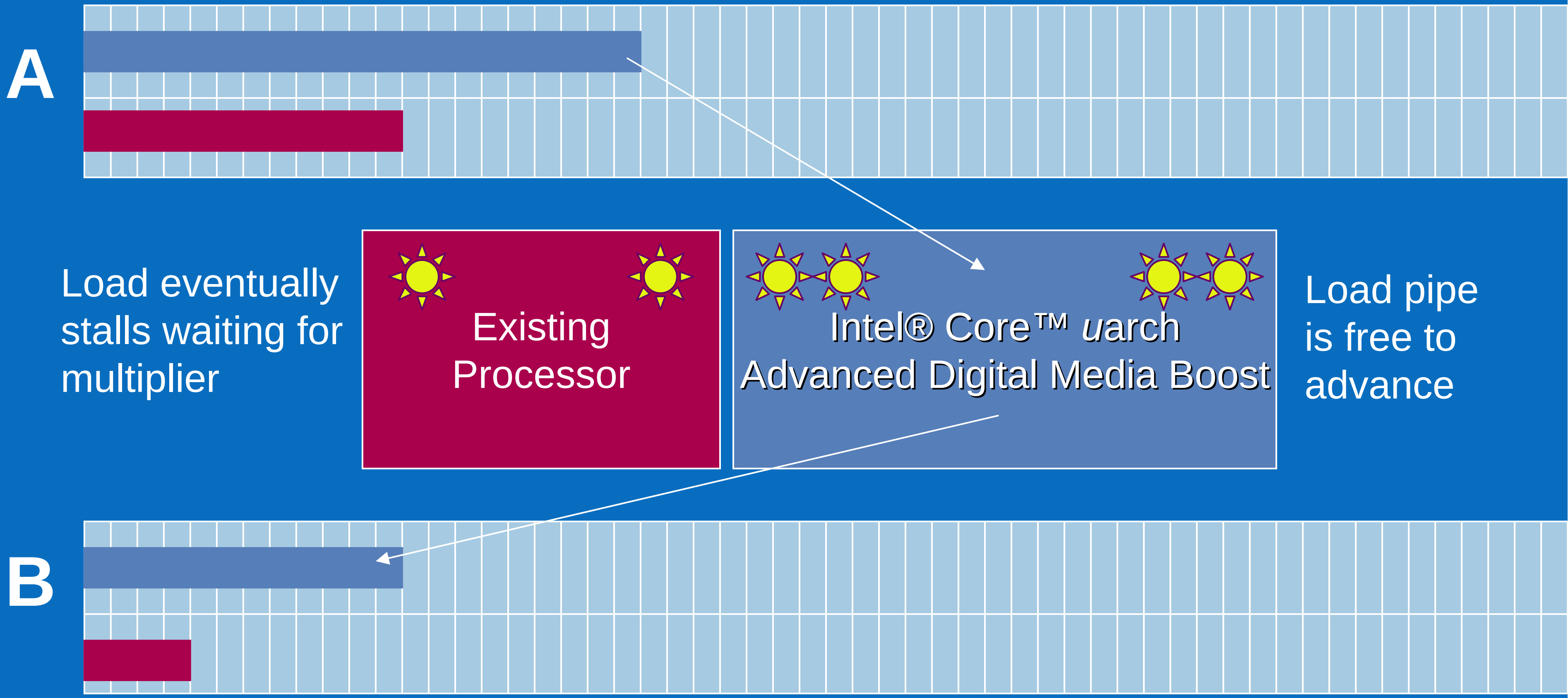
...keeps pipeline free for computations


A
Load eventually stalls waiting for multiplier
Existing Processor
Intel® Core™ *u*arch Advanced Digital Media Boost
Load pipe is free to advance
B

2x Compute Throughput / Clock

Advanced Digital Media Boost

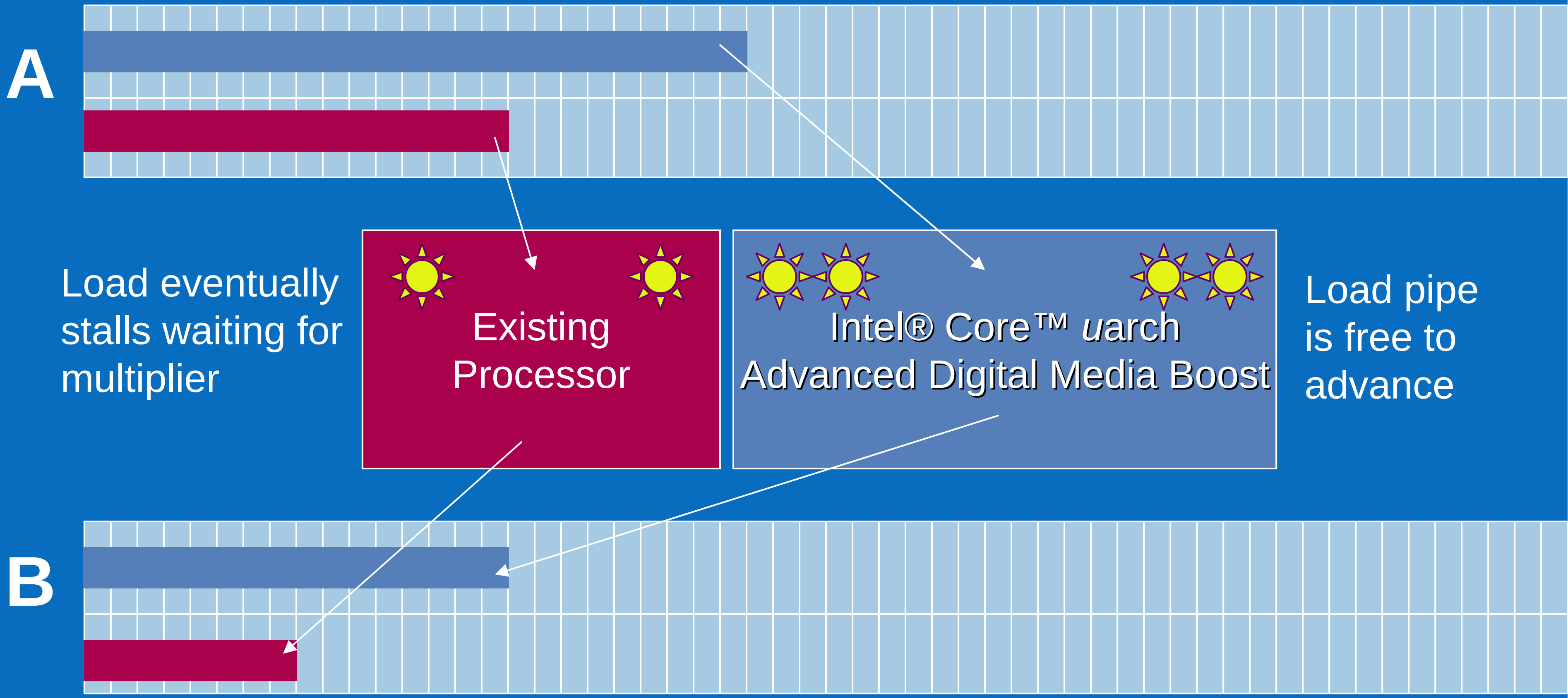
…maintains 2X throughput compared to prior implementations


A
Load eventually stalls waiting for multiplier
Existing Processor
Intel® Core™ uarch Advanced Digital Media Boost
Load pipe is free to advance
B

2x Compute Throughput / Clock

Advanced Digital Media Boost

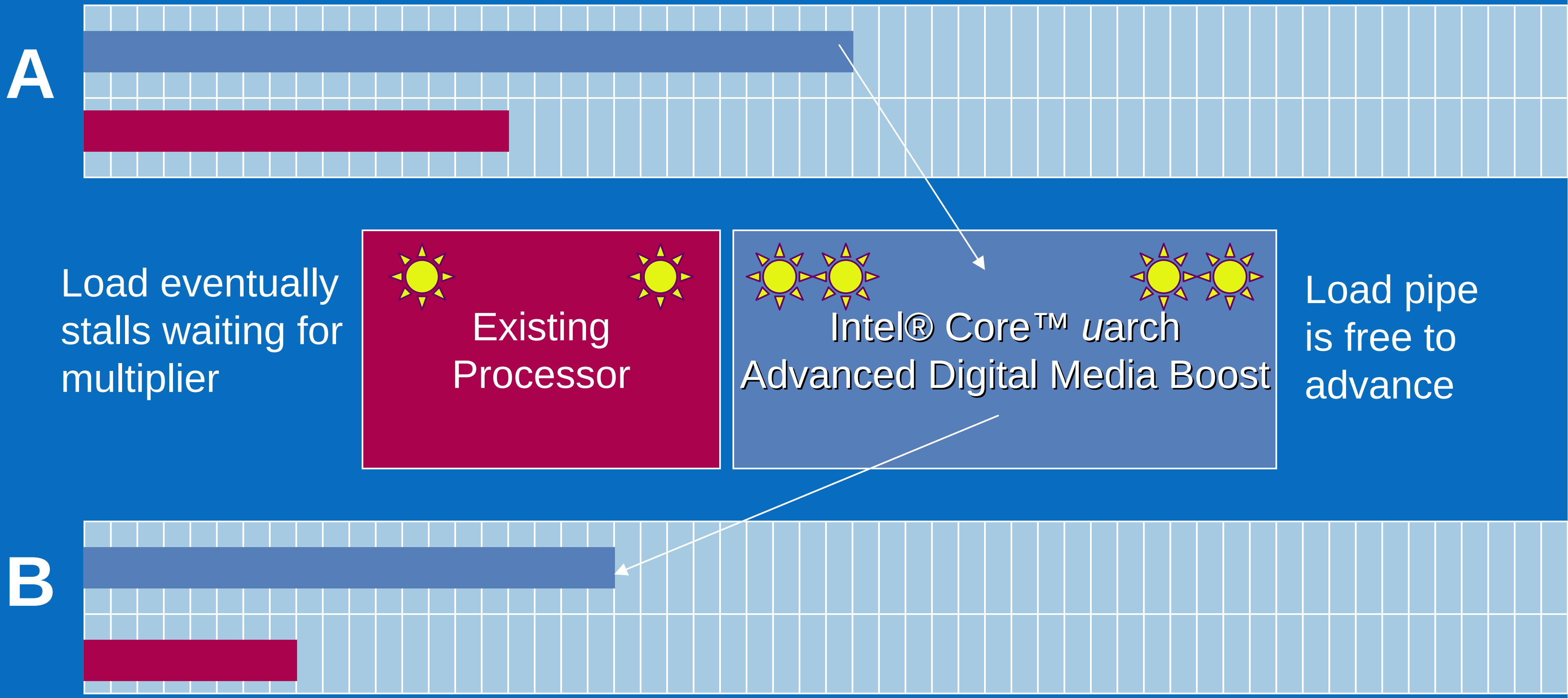
8 Single Precision Flops/cycle


A
Load eventually stalls waiting for multiplier
Existing Processor
Intel® Core™ uarch Advanced Digital Media Boost
Load pipe is free to advance
B

2x Compute Throughput / Clock

Advanced Digital Media Boost

4 Double Precision Flops/cycle

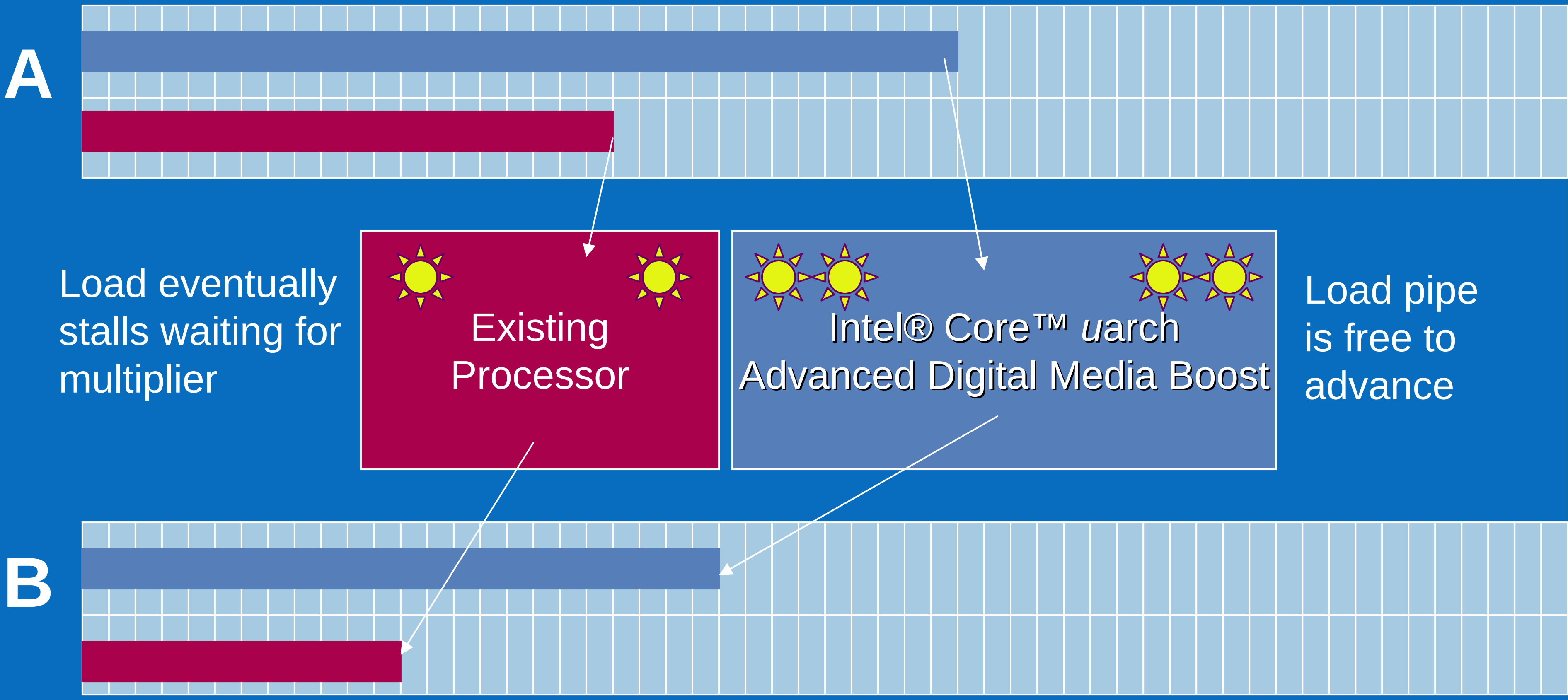


Load eventually stalls waiting for multiplier

Load pipe is free to advance

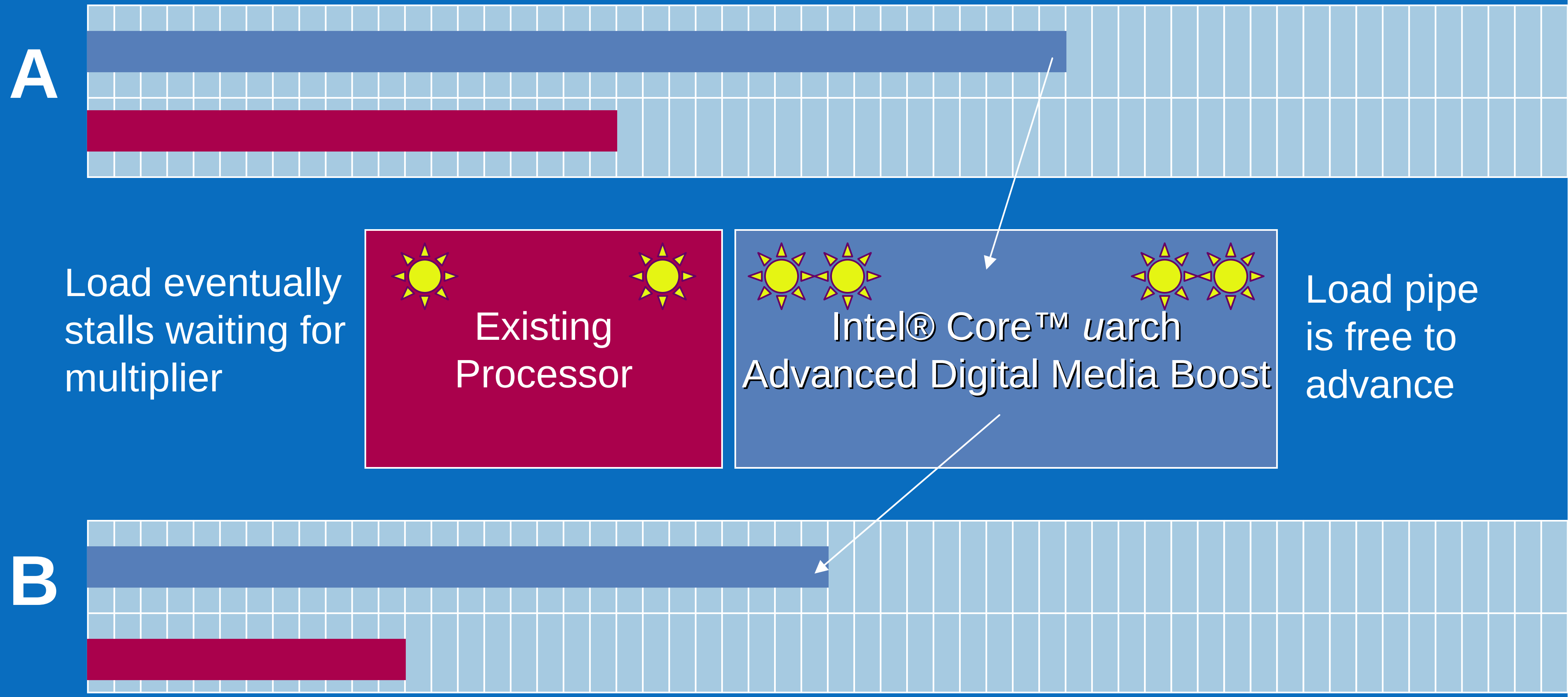
2x Compute Throughput / Clock

Advanced Digital Media Boost


A
Load eventually stalls waiting for multiplier
Existing Processor
Intel® Core™ uarch Advanced Digital Media Boost
Load pipe is free to advance
B

2x Compute Throughput / Clock

Advanced Digital Media Boost

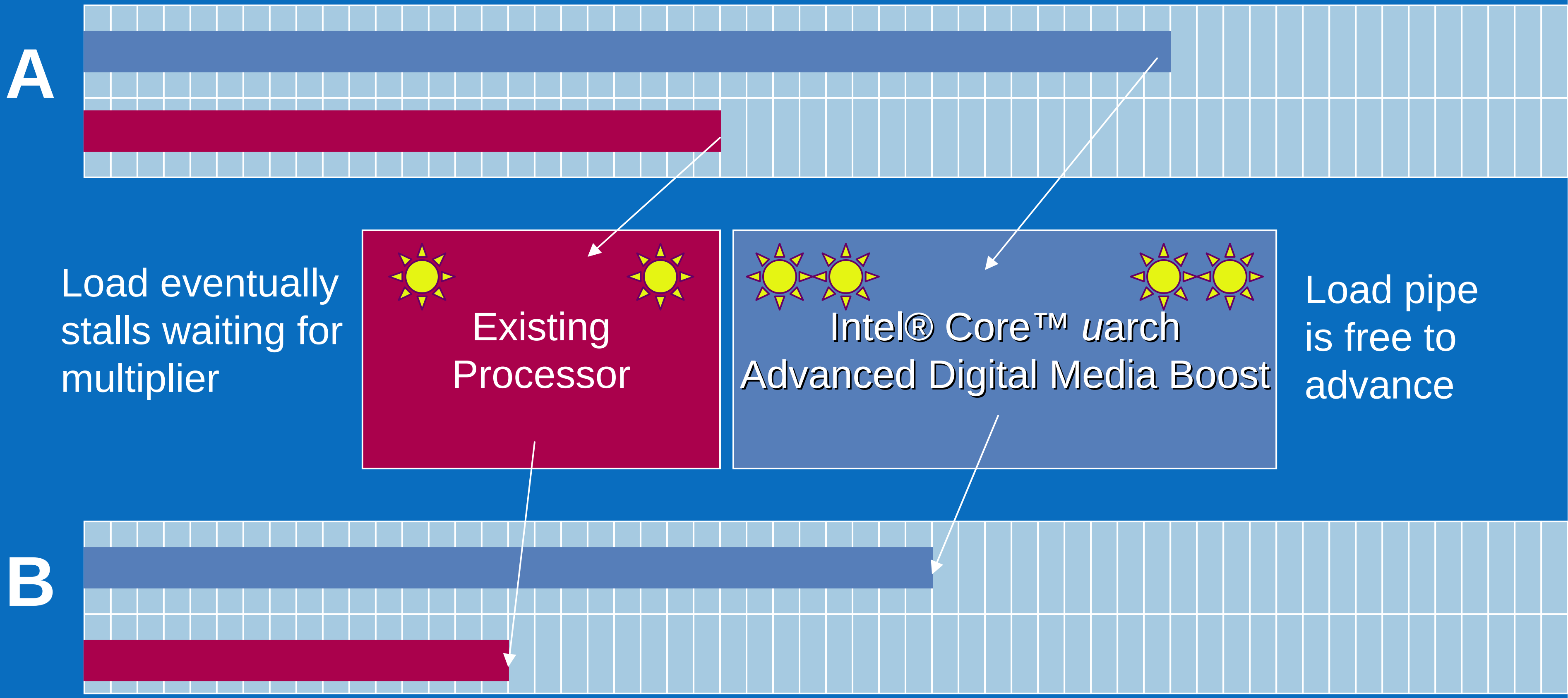


Load eventually stalls waiting for multiplier

Load pipe is free to advance

2x Compute Throughput / Clock

Advanced Digital Media Boost

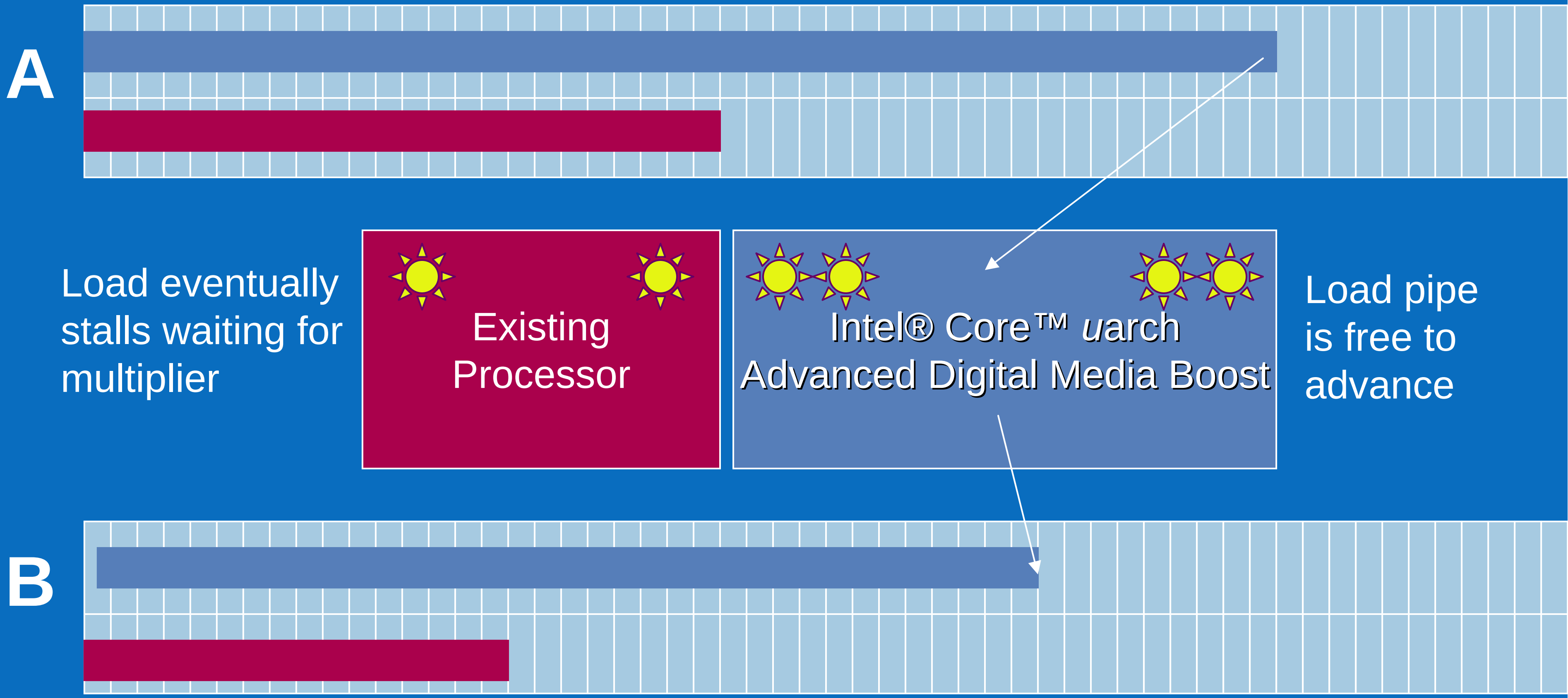


Load eventually stalls waiting for multiplier

Load pipe is free to advance

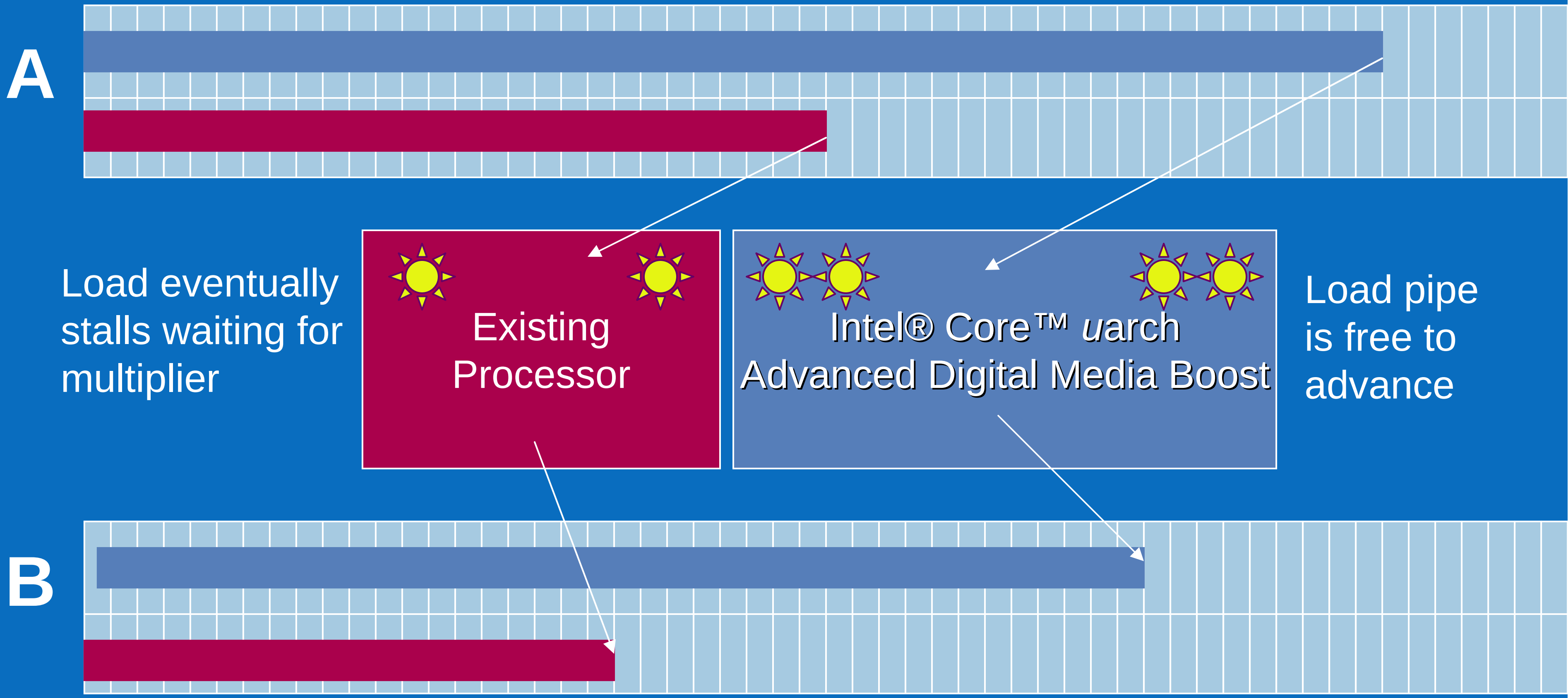
2x Compute Throughput / Clock

Advanced Digital Media Boost


A
Load eventually stalls waiting for multiplier
Existing Processor
Intel® Core™ uarch Advanced Digital Media Boost
Load pipe is free to advance
B

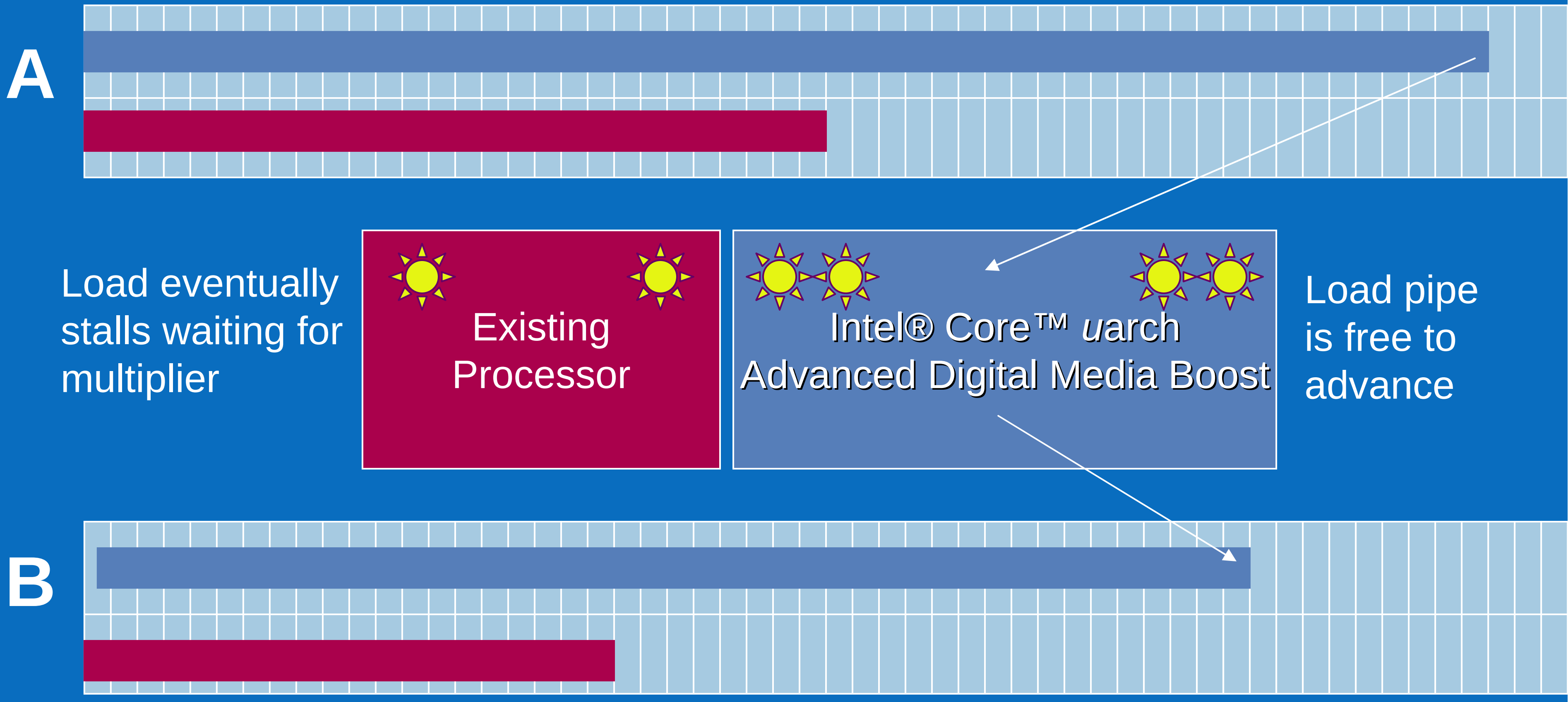
2x Compute Throughput / Clock

Advanced Digital Media Boost


A
Load eventually stalls waiting for multiplier
Existing Processor
Intel® Core™ uarch Advanced Digital Media Boost
Load pipe is free to advance
B

2x Compute Throughput / Clock

Advanced Digital Media Boost


A
B
Load eventually stalls waiting for multiplier
Existing Processor
Intel® Core™ uarch Advanced Digital Media Boost
Load pipe is free to advance

Leading Compute Density
2x Compute Throughput / Clock

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

Instruction name	Description
psignb/w/d mm, mm/m64 psignb/w/d xmm, xmm/m128	Per element, if the source operand is negative, multiply the destination operand by -1.
pabsb/w/d mm, mm/m64 pabsb/w/d xmm, xmm/m128	Per element, overwrite destination with absolute value of source.
phaddw/d/sw mm, mm/m64 phaddw/d/sw xmm, xmm/m128	Pairwise integer horizontal addition + pack.
phsubw/d/sw mm, mm/m64 phsubw/d/sw xmm, xmm/m128	Pairwise integer horizontal subtract + pack.
PMADDUBSW mm, mm/m64 PMADDUBSW xmm, xmm/m128	Multiply signed & unsigned bytes. Accumulate result to signed-words. (Multiply Accumulate)
PMULHRSW mm, mm/m64 PMULHRSW xmm, xmm/m128	Signed 16 bits multiply, return high bits.
PSHUFB mm, mm/m64 PSHUFB xmm, xmm/m128	A complete byte-granularity permutation, including force-to-zero flag.
PALIGNR mm, mm/m64, imm8 PALIGNR xmm, xmm/m128, imm8	Extract any continuous 16 (8 in the 64 bit case) bytes from the pair [dst, src] and store them to the dst register.

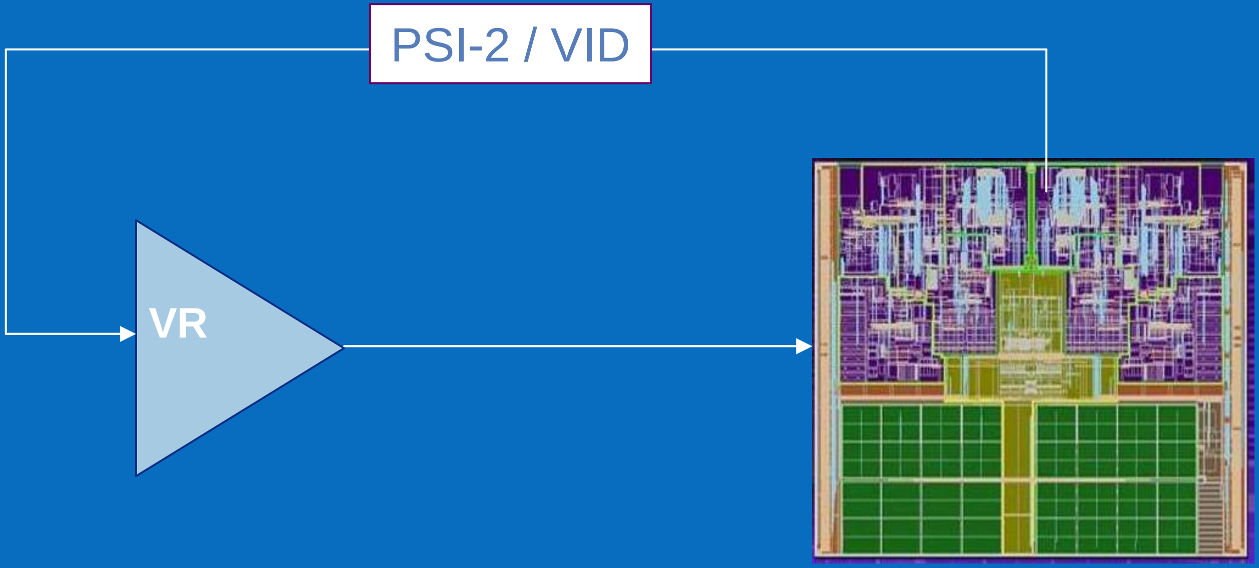
Using New Instructions with Intel Compiler

Architecture-tuning compiler switch –QxT.

Power Status Indicator (Mobile)

Processor communicates power consumption to external platform components

- Optimization of voltage regulator efficiency
- Load line and power delivery efficiency



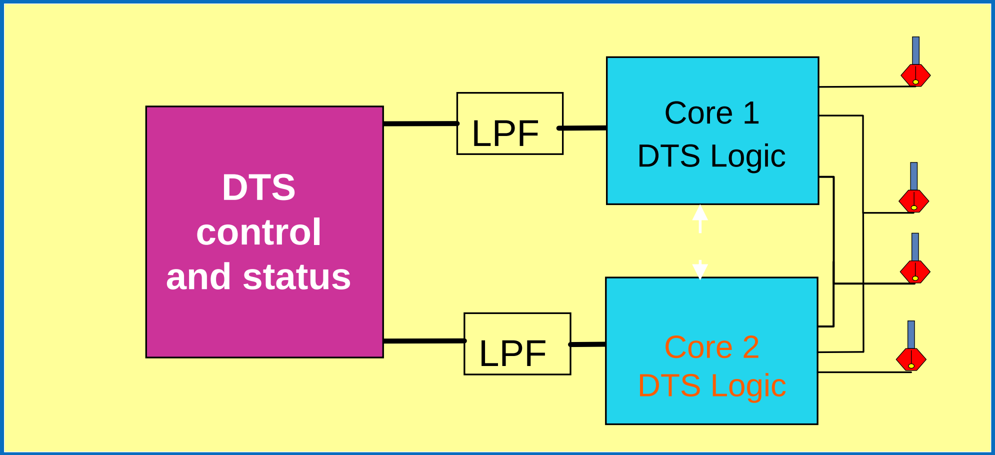
Enabling Efficient Processor and Platform Thermal Control…

DTS – Digital Thermal Sensor

Several thermal sensors are located within the Processor to cover all possible hot spots

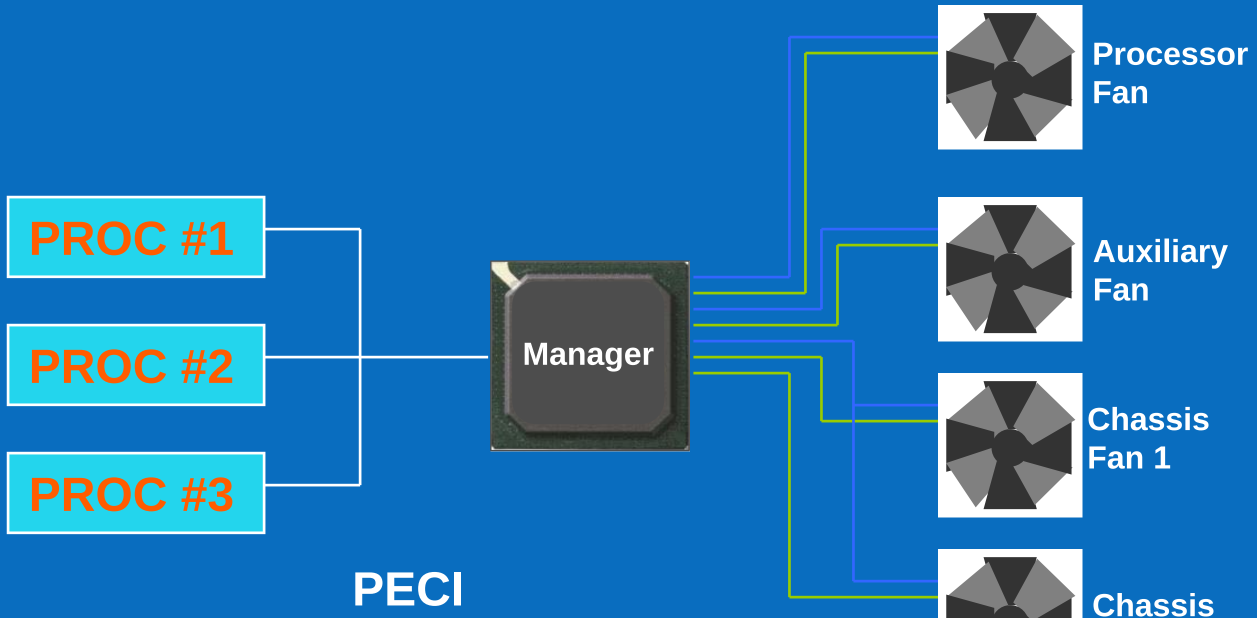
Dedicated logic scans the thermal sensors and measures the maximum temperature on the die at any given time

Accurately reporting Processor temperature enables advanced thermal control schemes



Platform Environment Control Interface (PECI)

Processor provides its temperature reading over a **multi drop single wire bus** allowing efficient platform thermal control



Other Features – Platform Power Management (cont.)

Front side bus with the following low power improvements

- Lower voltage
- DPWR# and BPRI# signals
  - Must have FSB traffic to enable data and address bus input sense amplifiers and control signals (~120 pins)
  - Eliminated higher address and dual processor capable pins

Other Features - Enhanced SpeedStep® Technology

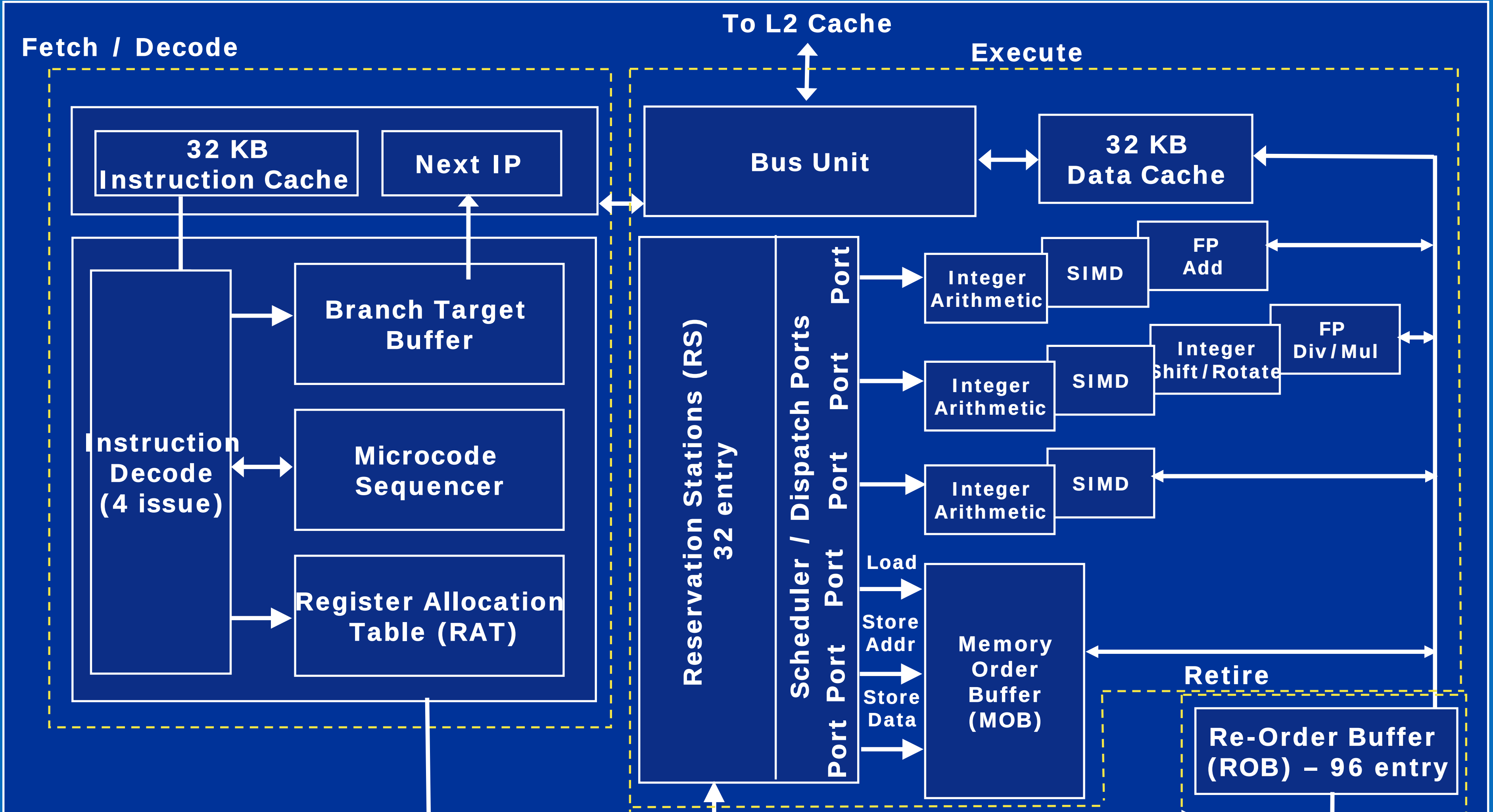
Voltage-Frequency switching separation

Clock partitioning and recovery

Event blocking

Even during periods of high performance execution, many parts of the chip core can be shut off

Intel® Core® Micro-architecture Blocks



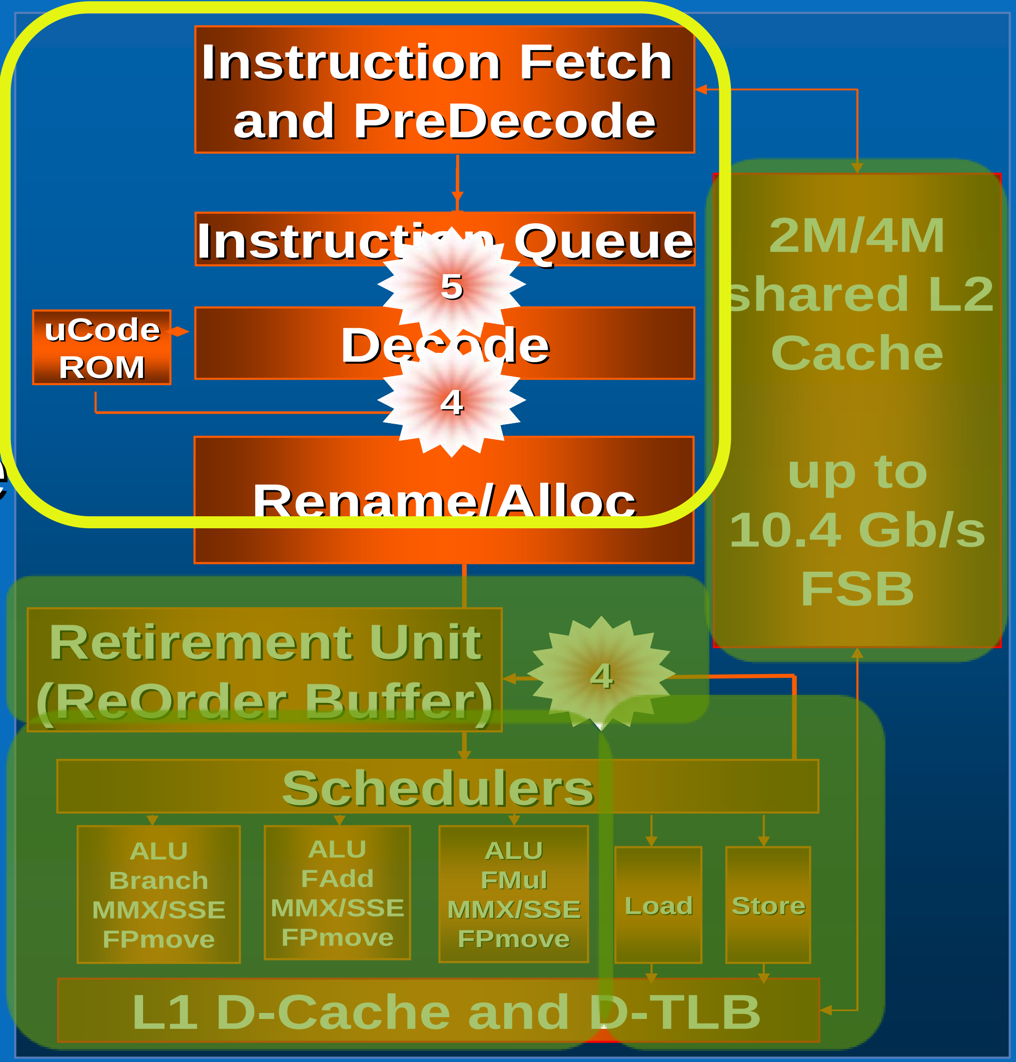
Disclaimer: This block diagram is for example purposes only.
Significant hardware blocks have been arranged or omitted for clarity.
Some resources (Bus Unit, L2 Cache, etc...) are shared between cores.

Intel® Core® Micro-architecture Notable Features

Enhanced Pipeline

in order

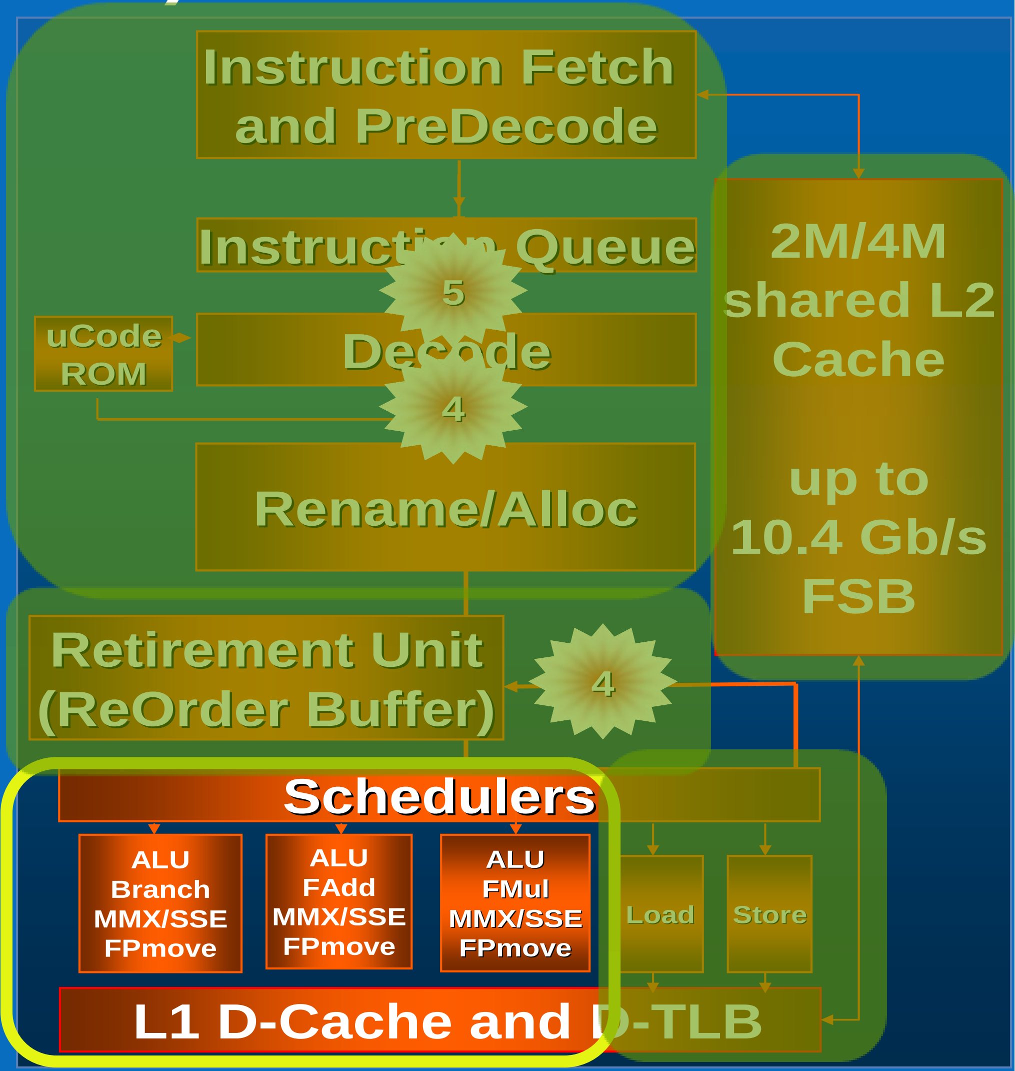
instruction fetch
instruction decode
micro-op rename
micro-op allocate



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

out of order

micro-op schedule
micro-op execute

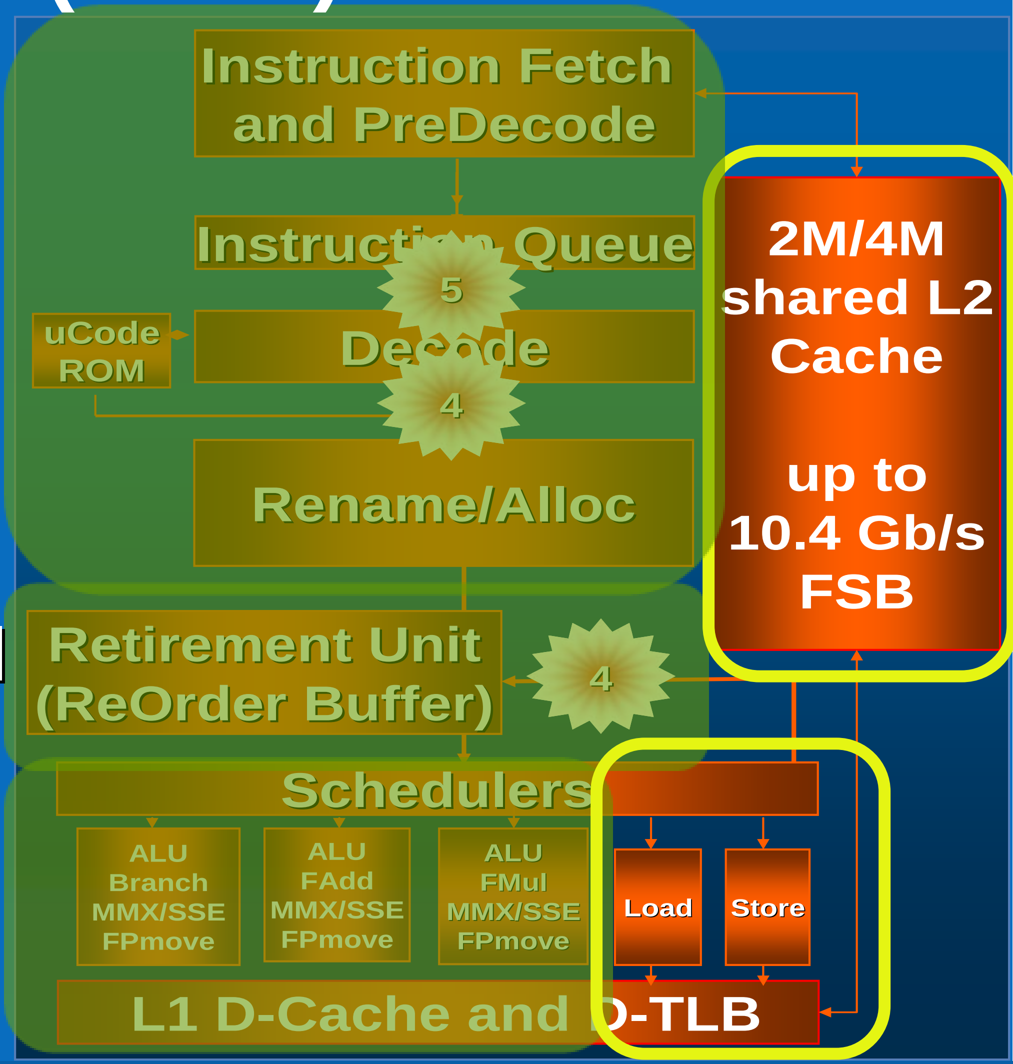


Intel® Core® Micro-architecture Notable Features
Enhanced Pipeline (cont.)

out of order

memory pipelines

memory order unit maintains architectural ordering requirements



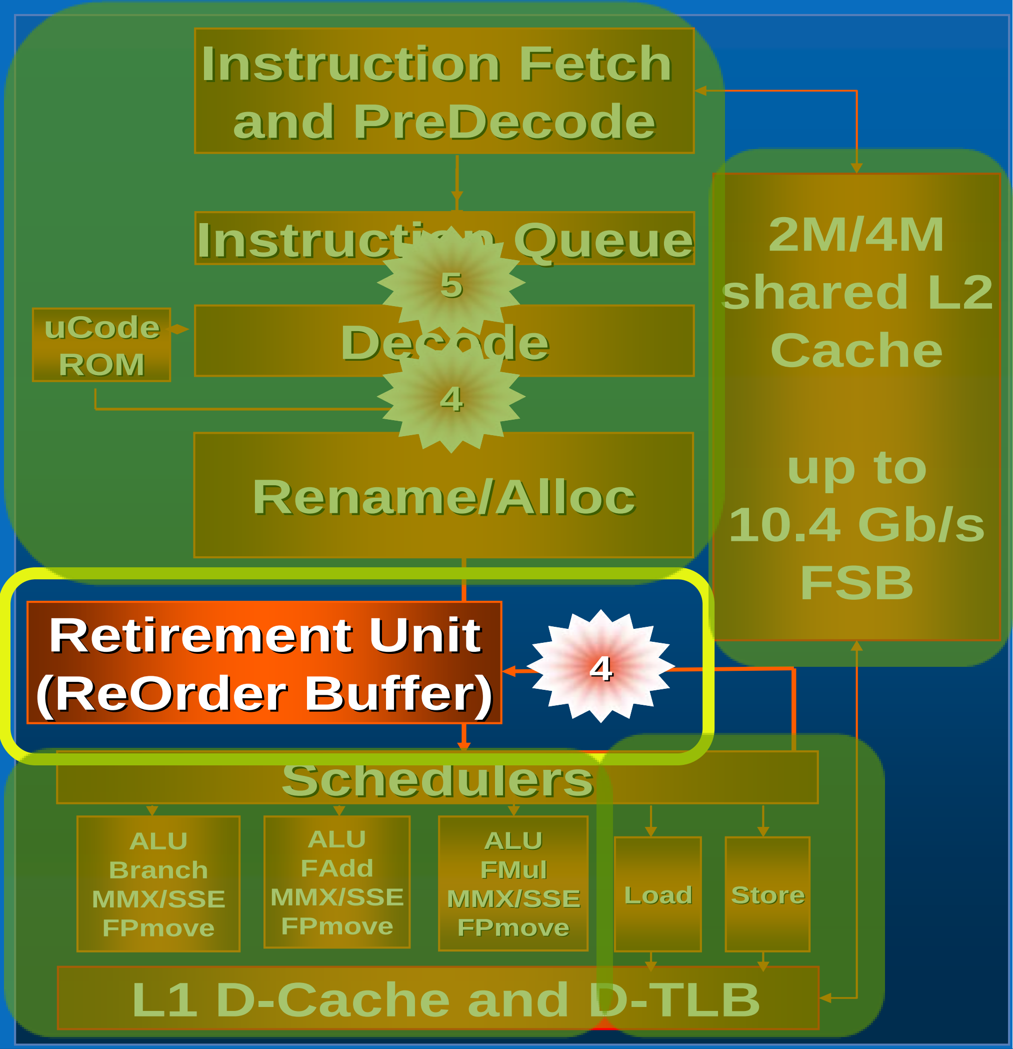
Intel® Core® Micro-architecture Notable Features

Enhanced Pipeline (cont.)

in order

micro-op retirement
fault handling

Retirement Unit maintains illusion of in order instruction retirement


Instruction Fetch and PreDecode
Instruction Queue
5
uCode ROM
Decode
4
Rename/Alloc
2M/4M shared L2 Cache
up to 10.4 Gb/s FSB
Retirement Unit (ReOrder Buffer)
4
Schedulers
ALU Branch MMX/SSE FPmove
ALU FAdd MMX/SSE FPmove
ALU FMul MMX/SSE FPmove
Load
Store
L1 D-Cache and D-TLB