Processing & Processors: Past, Present & Future

Mohammad-Reza Movahedin
Faculty of Engineering
University of Zanjan
1393/02/29

Updated 1399

Intel Processors History

- 1971: 4-bit 4004
  - 2,300 transistors, 740 KHz, 10 µm, $200
- 2012: 3rd generation Intel® Core™ processor
  - 1.4B Transistors, 2.9 GHz, 22 nm
- 2020: 3rd Generation Intel® Xeon® Scalable Processor
  - 28 Core, 56 Threads, ca. 10B Transistors
  - 2.90 GHz, 4.30 GHz Turbo
  - 38.5 MB Cache
  - TDP: 250W
  - Price: $13,000

What is a Processor?

A dumb piece of hardware that does nothing, but executing a stored program object code.

- In compare to pure hardware:
  - Slower Speed
  - Higher Flexibility
  - Higher Power Consumption
  - Better in Maintenance & Debug

HW/SW Partitioning Question

Processors Applications

- General Purpose Personal Computers (PCs)
- Hand-Held Devices (Tablet, Smart Phones)
- Servers & Data Centers
- Embedded Systems
- Cars & Automotive
- Game Boxes
- Wearables

Processor Design Start Point

Definition of Instruction Set Architecture (ISA):

- Memory Space: address and word size
- Internal Registers: number, usage and size
- Instruction Words: fields and functions
  - arithmetic and logical operations
  - jump and conditional branch
  - subroutine call support

CISC ISAs

A Complex Instruction Set Computing ISA is an ISA where single instructions can execute several low-level operations (such as a load from memory, an arithmetic operation, and a memory store) and/or are capable of multi-step operations or addressing modes within single instructions.

Examples: IBM System/360 to z/Architecture, DEC PDP11 & VAX, Motorola 68k and Intel x86.

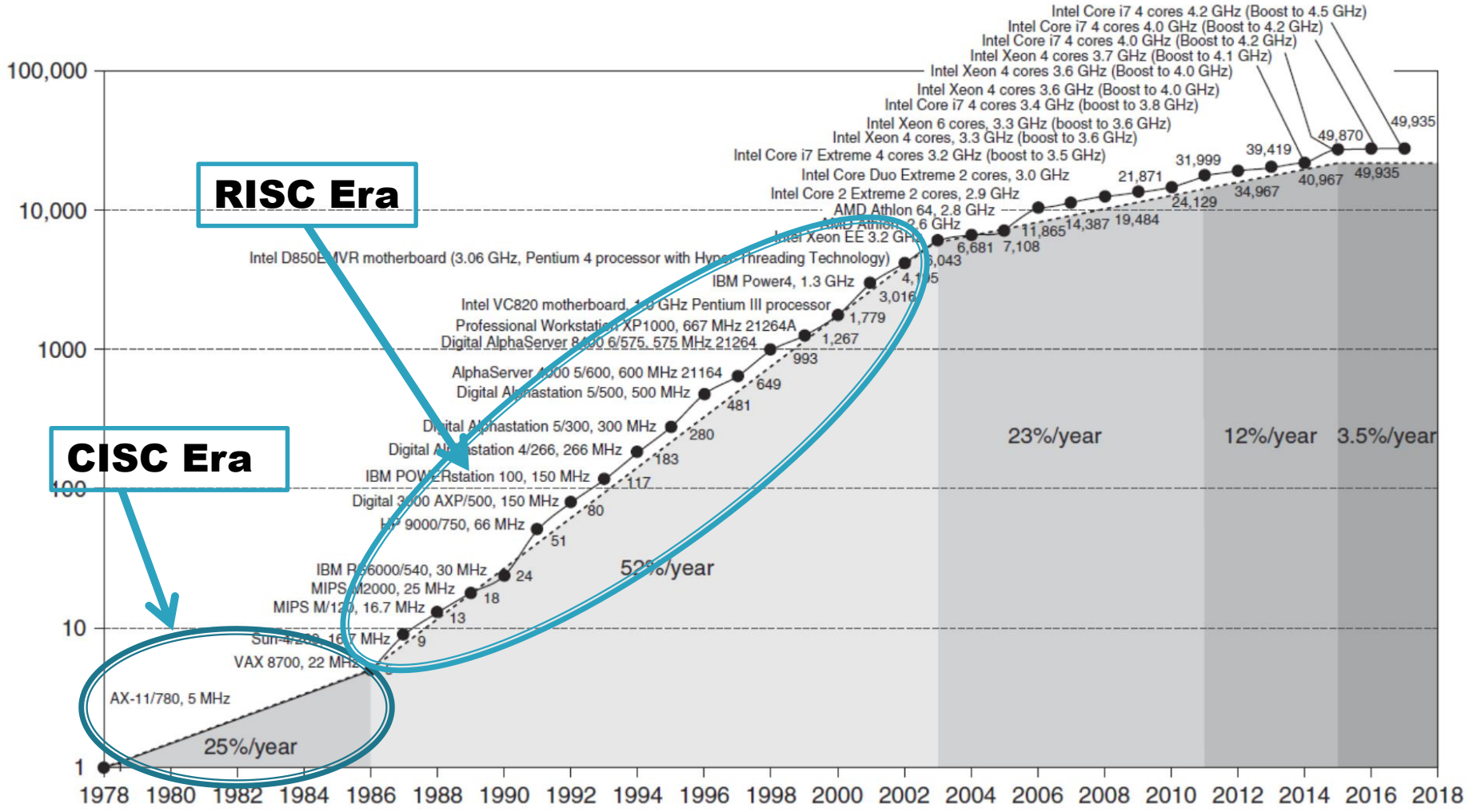
RISC ISAs

Reduced **I**nstruction **S**et **C**omputing (RISC) is a ISA design strategy based on the insight that simplified instructions can provide higher performance if this simplicity enables much faster execution of each instruction.

Furthermore, RISC ISA is a memory load/store architecture, where memory is always accessed only through specific instructions, rather than as part of other instructions.

Examples: MIPS, ARM, SPARC, PowerPC, Alpha

Uniprocessors Performance Growth



An Intricate ISA named x86 !

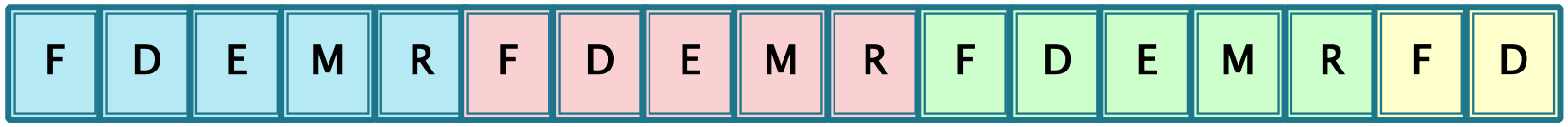
- Intel x86 is originally a CISC ISA.
- x86 *was* and *is* the dominant ISA in industry.
- All Intel efforts to enter RISC marketplace failed.
- Solution: an extra hardware added in front-end to translate CISC instructions to 1, 2, 3, 4 or even several RISC-like µops.
- Execution path of recent x86 processors (Intel or AMD) are designed for µops.
- Who pays for that extra silicon? and extra power consumed?

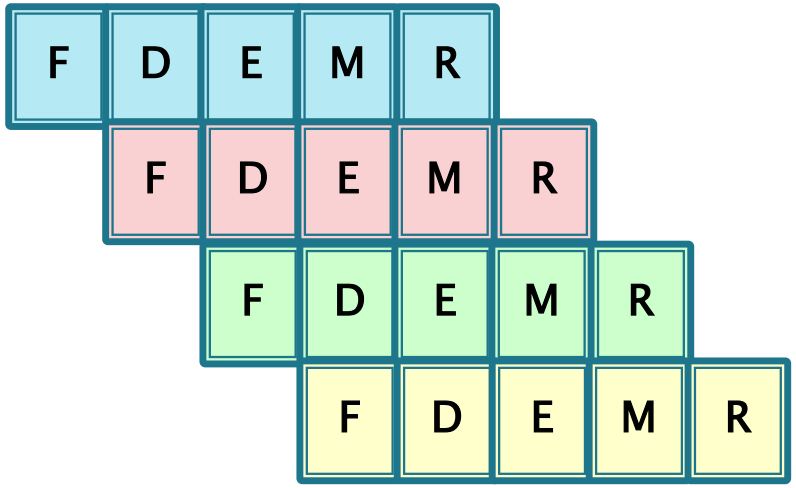
Poor users, who want SW backward compatibility

Instruction Level Parallelism (ILP)

- Pipelining
- Data Forwarding
- Branch Prediction
- Multiple Issue
- Superscalars
  - Out Of Order Execution
  - Register Renaming
  - Hyper Threading
- Real World Examples

Pipelining





- F: Instruction Fetch
- D: Instr. Decode
- E: Execute
- M: Memory Access
- R: Register File Write

Theoretical Maximum Speed-Up = Number of Pipeline Stages

Latency (time for each instruction) is not decreased

Pipelining, Examples

Microprocessor	Year	Clock Rate	Pipeline Stages
Intel 486	1989	25 MHz	5
Intel Pentium	1993	66 MHz	5
Intel Pentium Pro	1997	200 MHz	10
Intel Pentium 4 Willamette	2001	2000 MHz	22
Intel Pentium 4 Prescott	2004	3600 MHz	31
Intel Core	2006	2930 MHz	14
Intel Core i5 Nehalem	2010	3300 MHz	14
Intel Core i5 Ivy Bridge	2012	3400 MHz	14

Pipelining Antagonists: Hazards

Situations that prevent starting immediately next instruction in the next cycle, must **STALL**

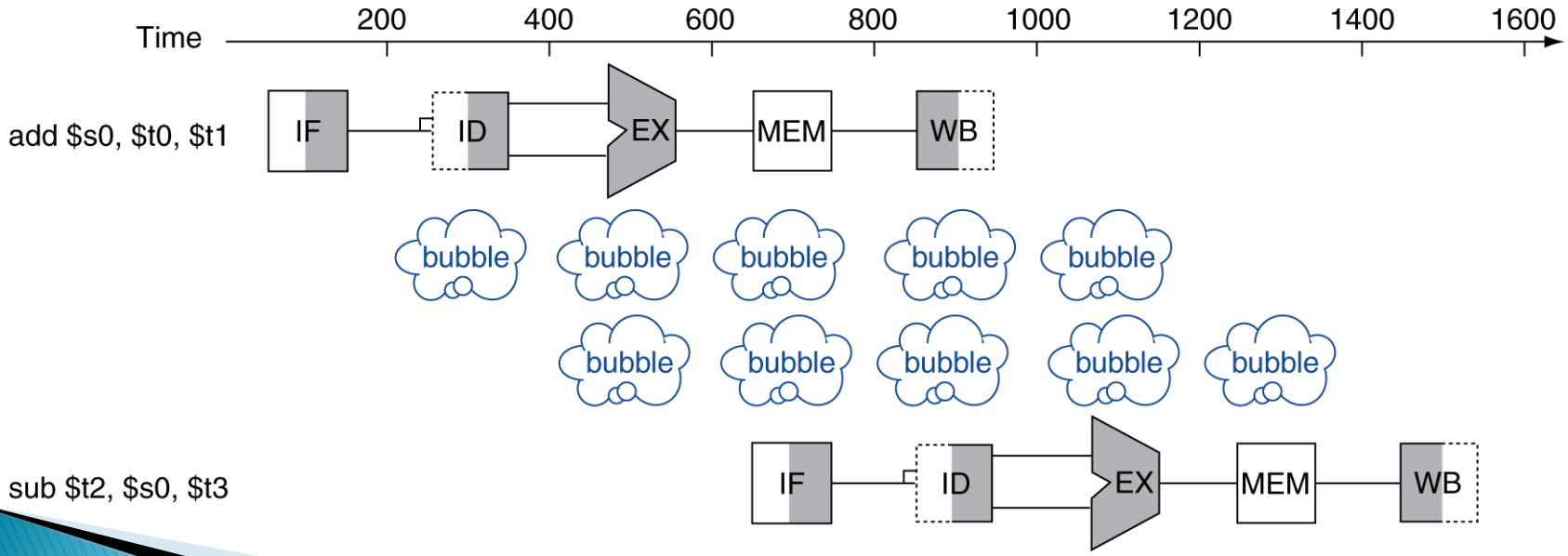
- Structure Hazards
  - A required resource is busy
- Data Hazard
  - Need to wait for previous instruction to complete its data read/write
- Control Hazard
  - Deciding on control action depends on previous instruction

Instruction Level Parallelism (ILP)

- Pipelining
- Data Forwarding
- Branch Prediction
- Multiple Issue
- Superscalars
  - Out Of Order Execution
  - Register Renaming
  - Hyper Threading
- Real World Examples

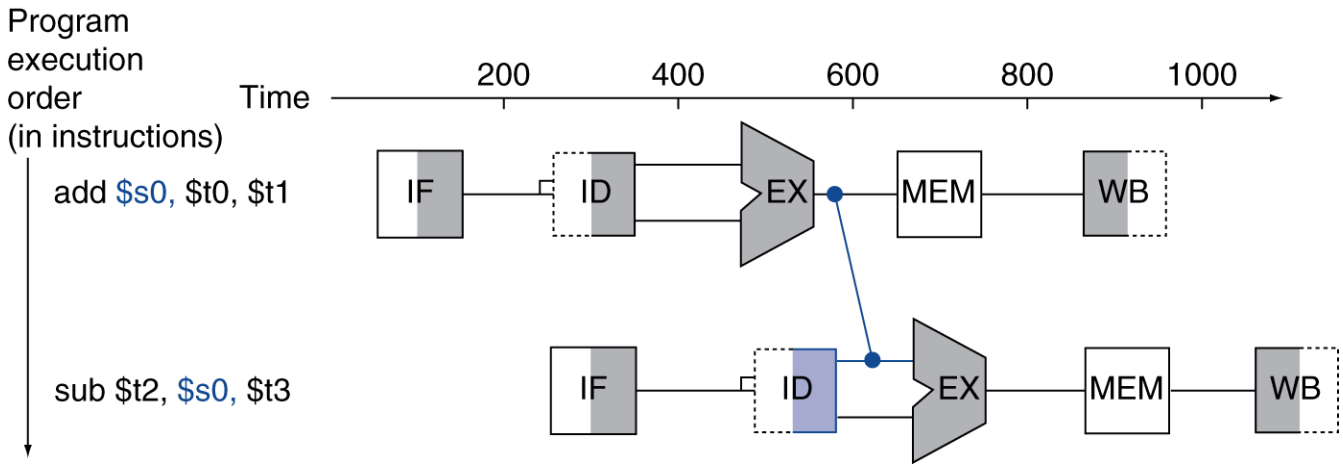
Data Hazards

- An instruction depends on completion of data access by a previous instruction
  - `add  $s0, $t0, $t1`
    `sub  $t2, $s0, $t3`



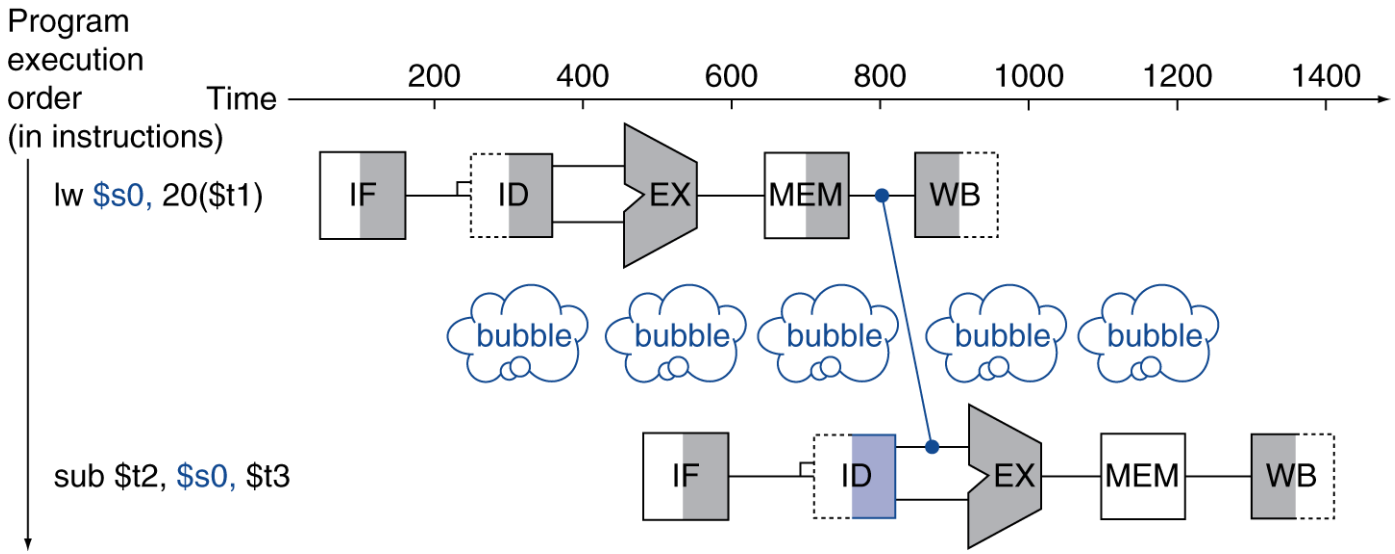
Data Forwarding

- Use result when it is computed
  - Don't wait for it to be stored in a register
  - Requires extra connections in the datapath



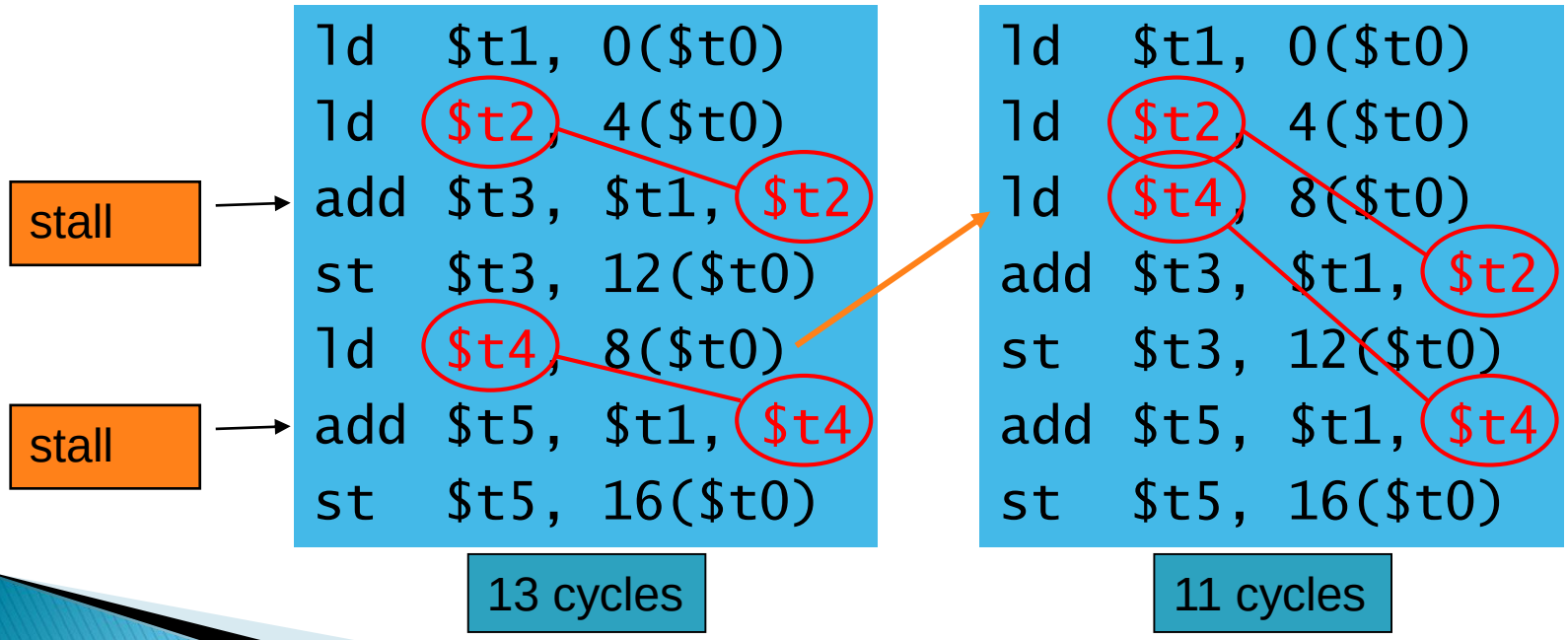
Data Forwarding, cont.

- Cannot always avoid stalls by forwarding
  - If value not computed when needed
  - Cannot forward backward in time!



Code Scheduling to Avoid Stalls

- Reorder code to avoid use of non-forwardable result in the next instruction
- C code for `A = B + E; C = B + F;`

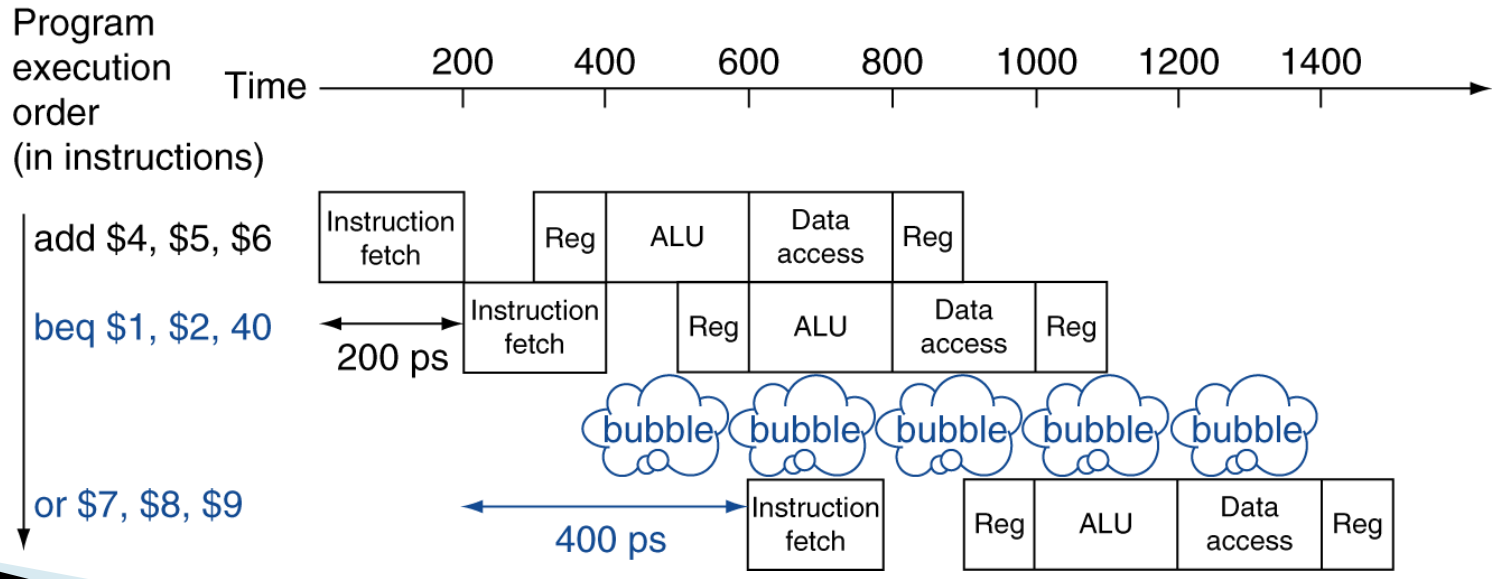


Instruction Level Parallelism (ILP)

- Pipelining
- Data Forwarding
- Branch Prediction
- Multiple Issue
- Superscalars
  - Out Of Order Execution
  - Register Renaming
  - Hyper Threading
- Real World Examples

Control Hazards, Stall on Branch

- Branch (if-then-else) determines flow of control
  - Fetching next instruction depends on branch outcome
- Should wait until branch outcome determined before fetching next instruction



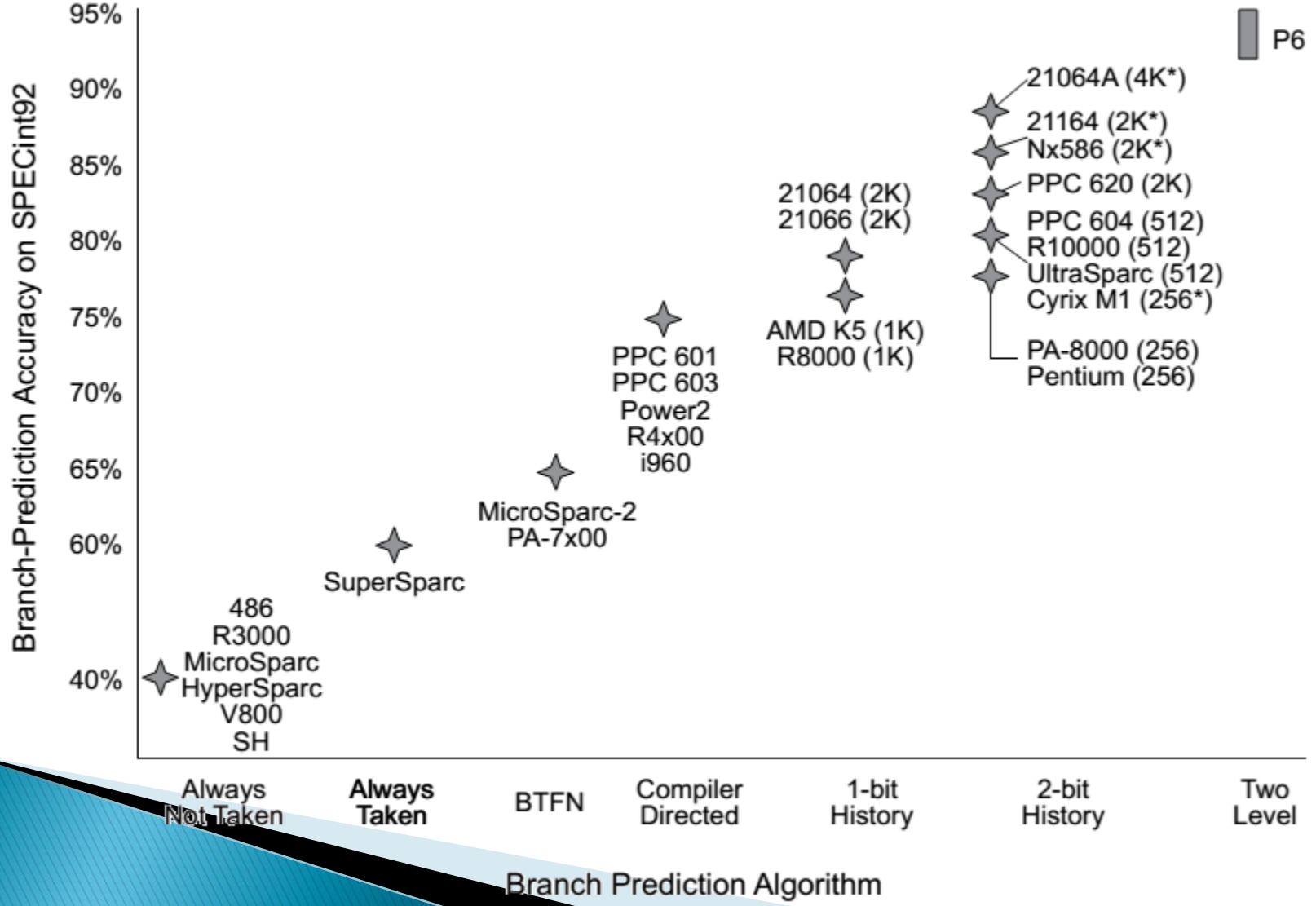
Control Hazards, Branch Prediction

- Longer pipelines cannot readily determine branch outcome early.
- Stall penalty becomes huge and unacceptable.
- It grows exponentially in Super Scalars.
- Solution:
  - **Predict** outcome of branch in early pipeline stages.
  - Continue execution based on **prediction**.
  - Prevent results to be committed (i.e. flushed out) if **prediction** determined to be wrong.

Branch Prediction

- Static Branch Prediction, based on usual branch behavior.
  - Always not-taken (or taken),
  - Separated instructions for likely taken and not-taken cases,
  - Predict backward branches taken, forward not taken.
- Dynamic Branch Prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch
  - Assume future behavior will continue the trend
    - When wrong, flush while re-fetching, and update history

Branch Prediction Performance

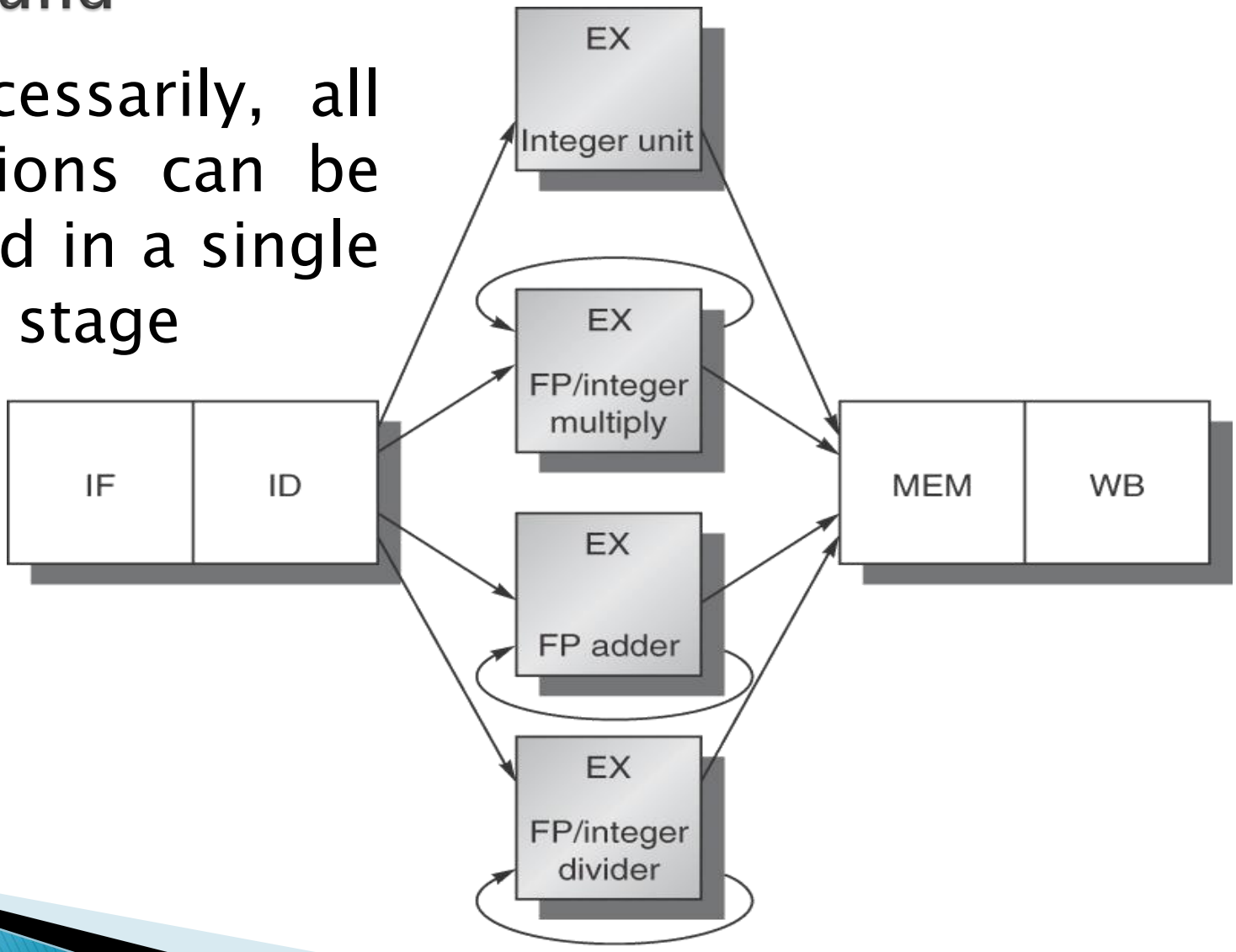


Instruction Level Parallelism (ILP)

- Pipelining
- Data Forwarding
- Branch Prediction
- Multiple Issue
- Superscalars
  - Out Of Order Execution
  - Register Renaming
  - Hyper Threading
- Real World Examples

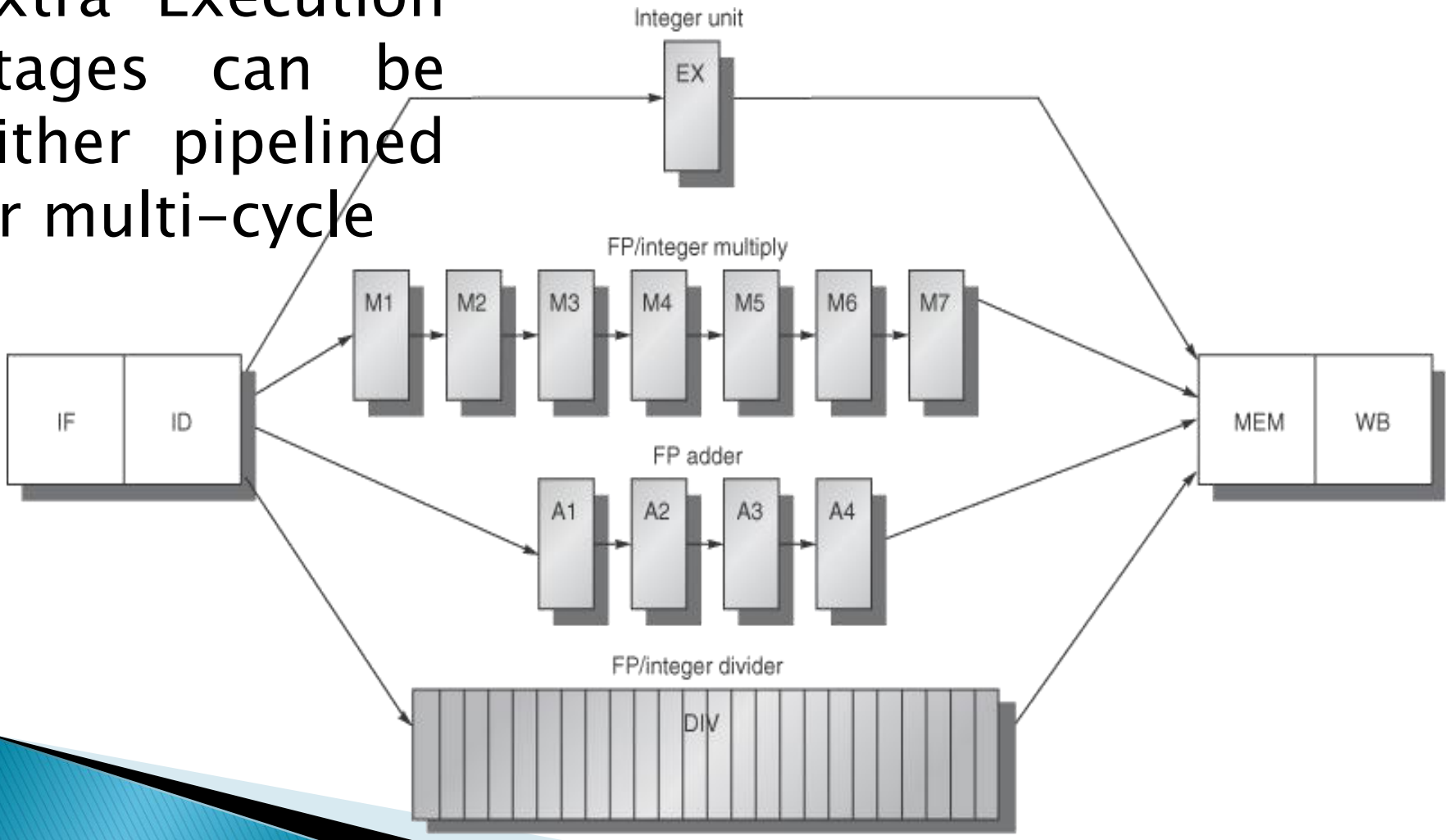
Multiple Issue,
Background

Not necessarily, all instructions can be executed in a single Execute stage



Multiple Issue,
Background

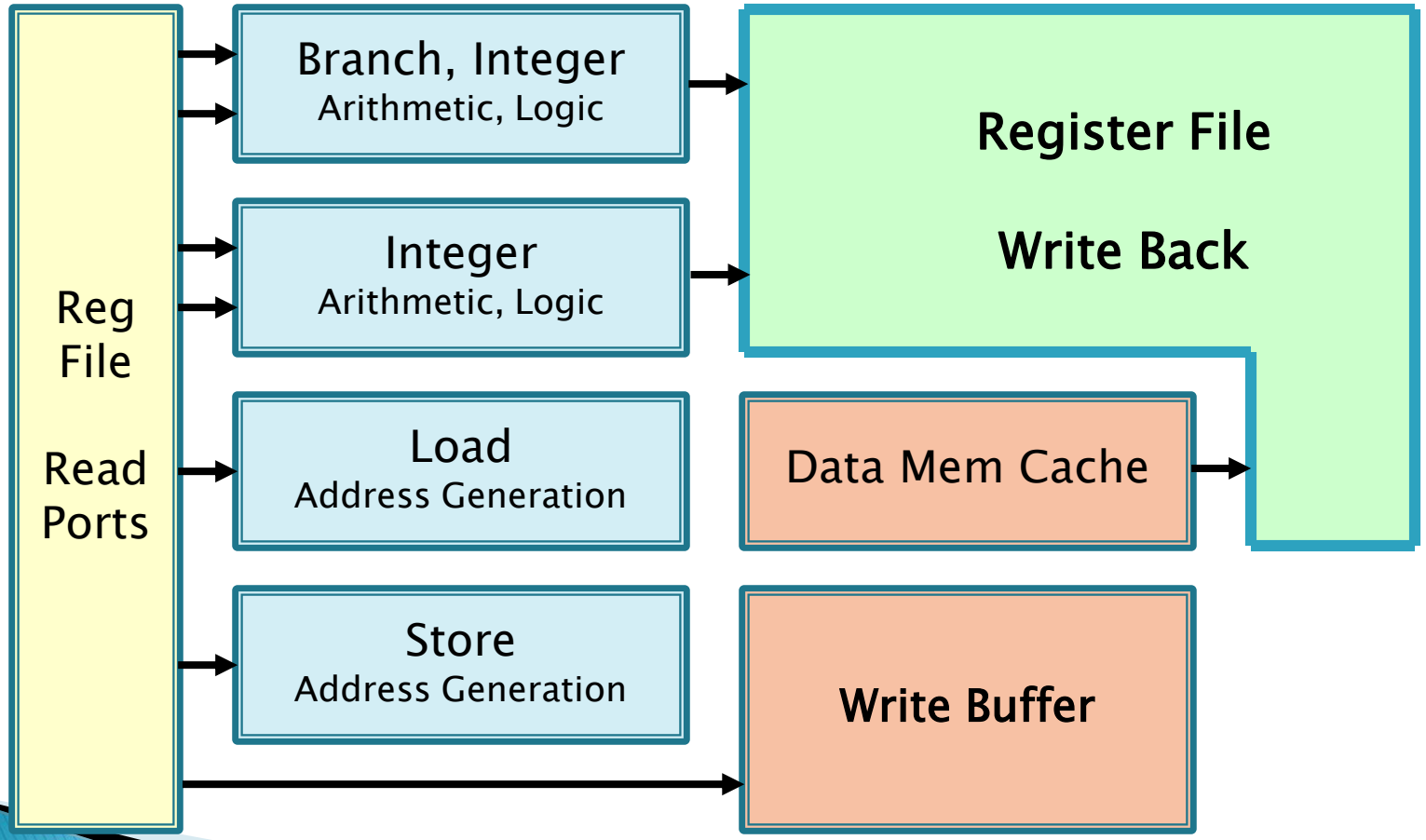
Extra Execution stages can be either pipelined or multi-cycle



Multiple Issue

- Concurrent Execution (Functional) units avails **dispatching more than one instruction per clock**.
  - Popular instructions, such as integer operations, memory load and store, etc. may have more than one Exec. unit.
- This means Instruction Per Clock (IPC) > 1.
  - Clock Per Instruction (CPI) $= 1 \ / \ \text{IPC}$
  - Non pipelined multi-cycle processor: CPI $\gg 1$
  - Pipelined processor: IPC $\simeq 1$ (max. is 1)

ACA–MIPS Multiple Issue



ACA-MIPS VLIW (Very Large Instruction Word)

32-bit	32-bit	32-bit	32-bit
Branch/Integer	Integer Instr.	Load Instr.	Store Inst.

- A very long instruction defines operation of each issue slot and may be NOP
- Each sub-Instruction is very similar to original ACA-MIPS instruction, though minor savings can be made in their coding
- This is user/compiler responsibility to fill each slot as much as possible. This is done by means of code re-ordering, loop unrolling, etc.
- A NOP of a sub-instruction means loss of performance
- All four VLIW sub-instructions are initiated and completed together. This means in case of cache miss stall, ALL instructions are stalled.

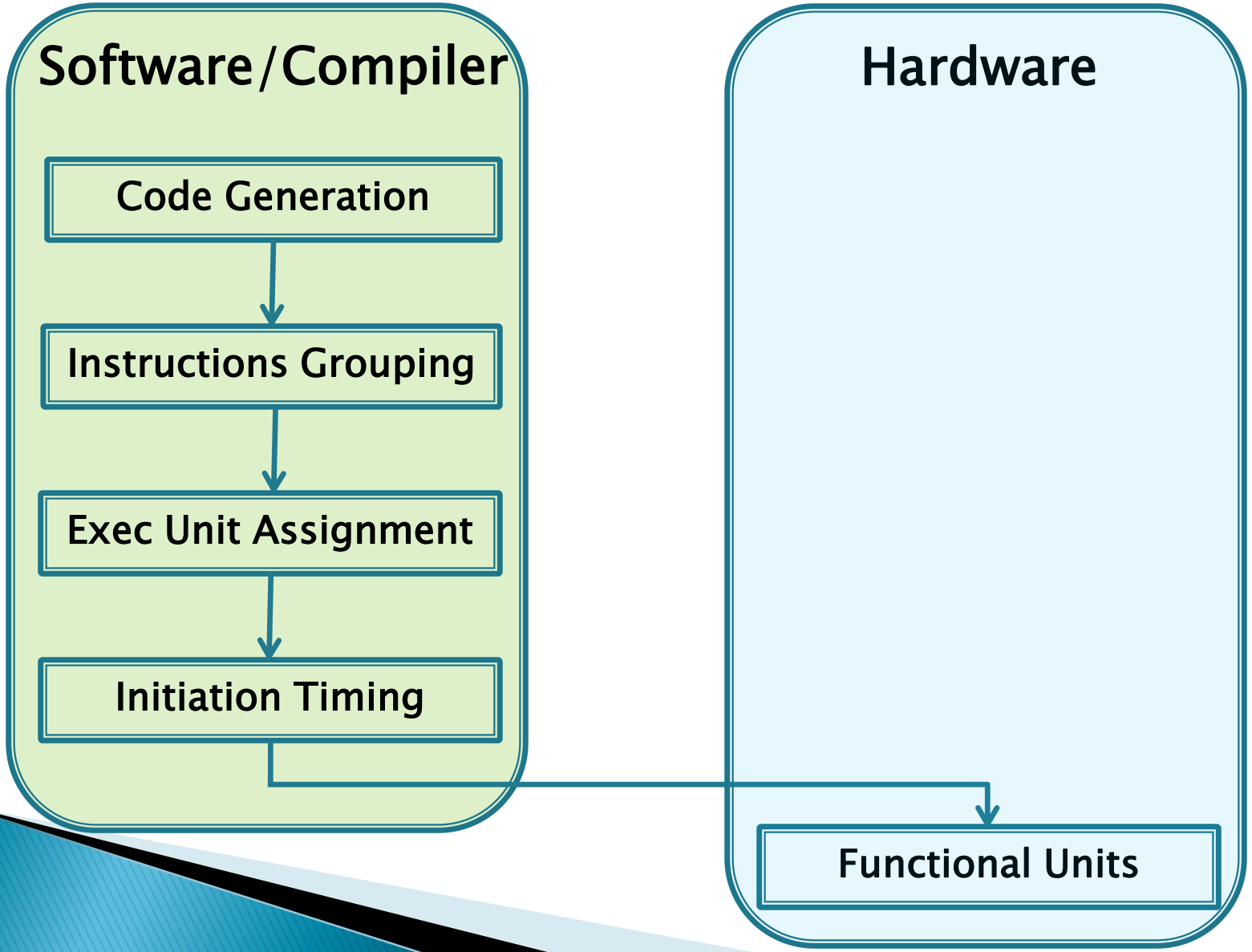
Major Tasks for Multiple-Issue Execution

Processing instructions in parallel requires three major tasks:

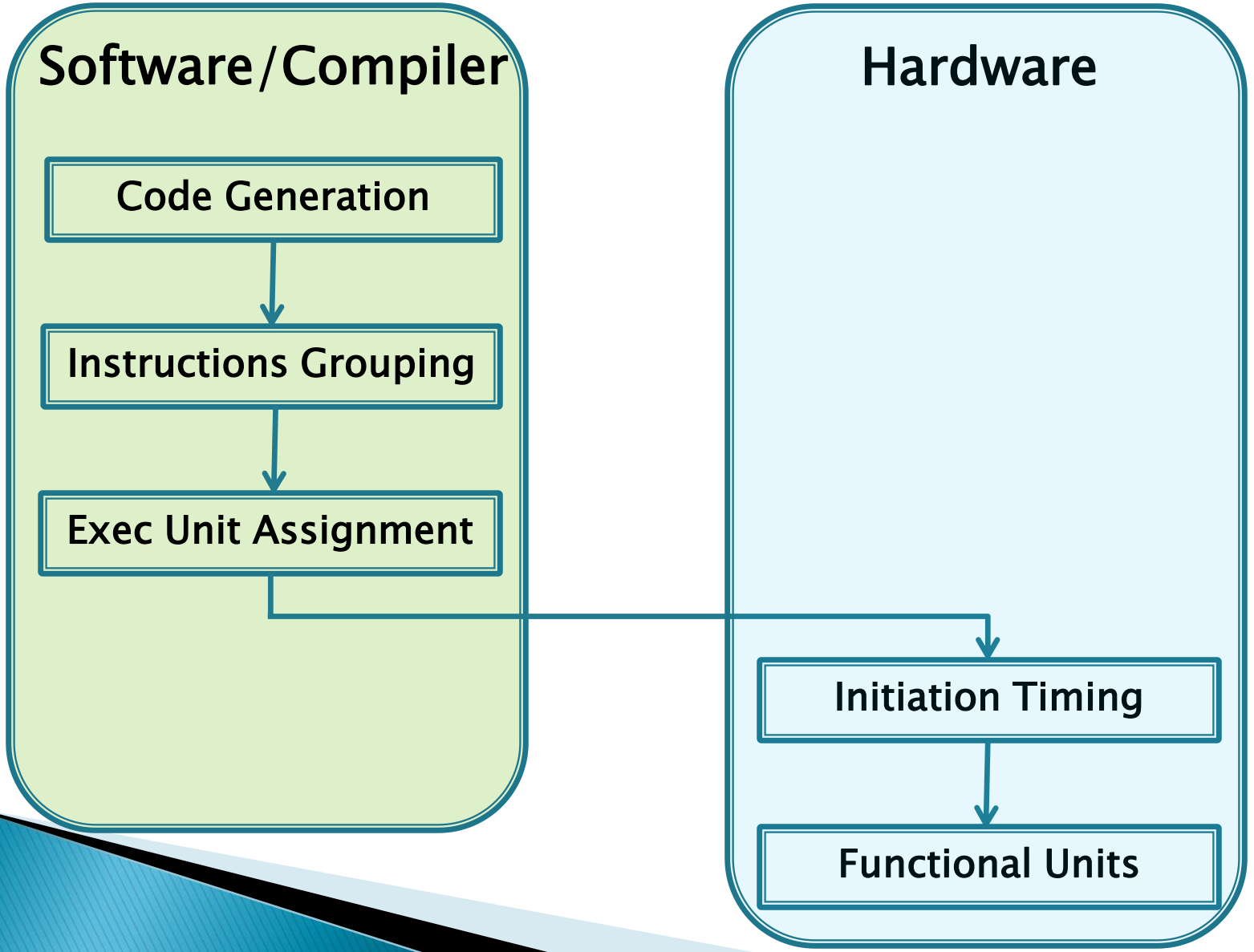
1. Checking dependencies between instructions to determine which instructions can be grouped together for parallel execution.
2. Assigning instructions to the functional units on the hardware.
3. Determining when instructions are initiated (i.e., start execution)

Ref.: Mark Smotherman, “Understanding EPIC Architectures and Implementations ”

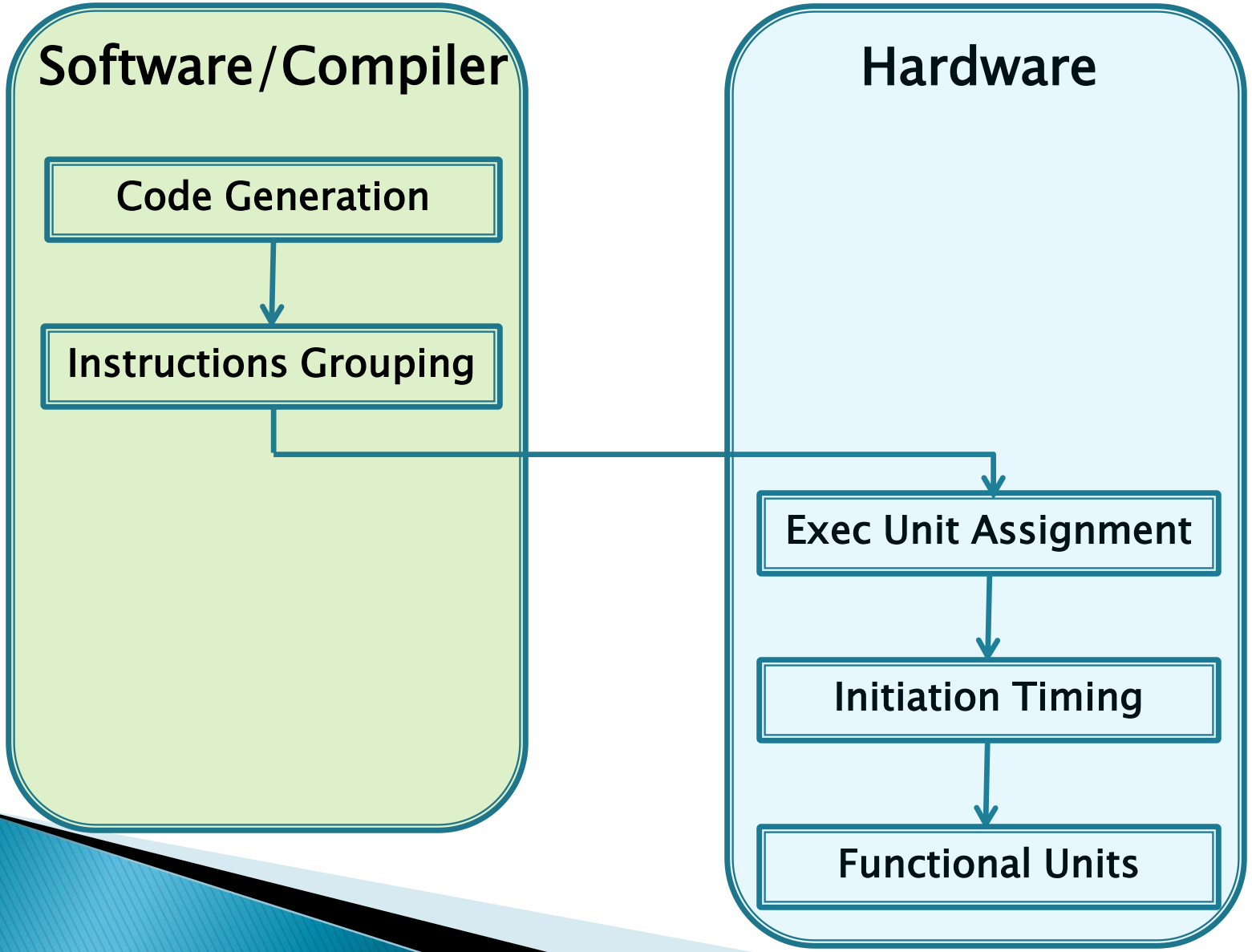
Multiple Issue, Static Scheduling: Very Long Instruction Word (VLIW)



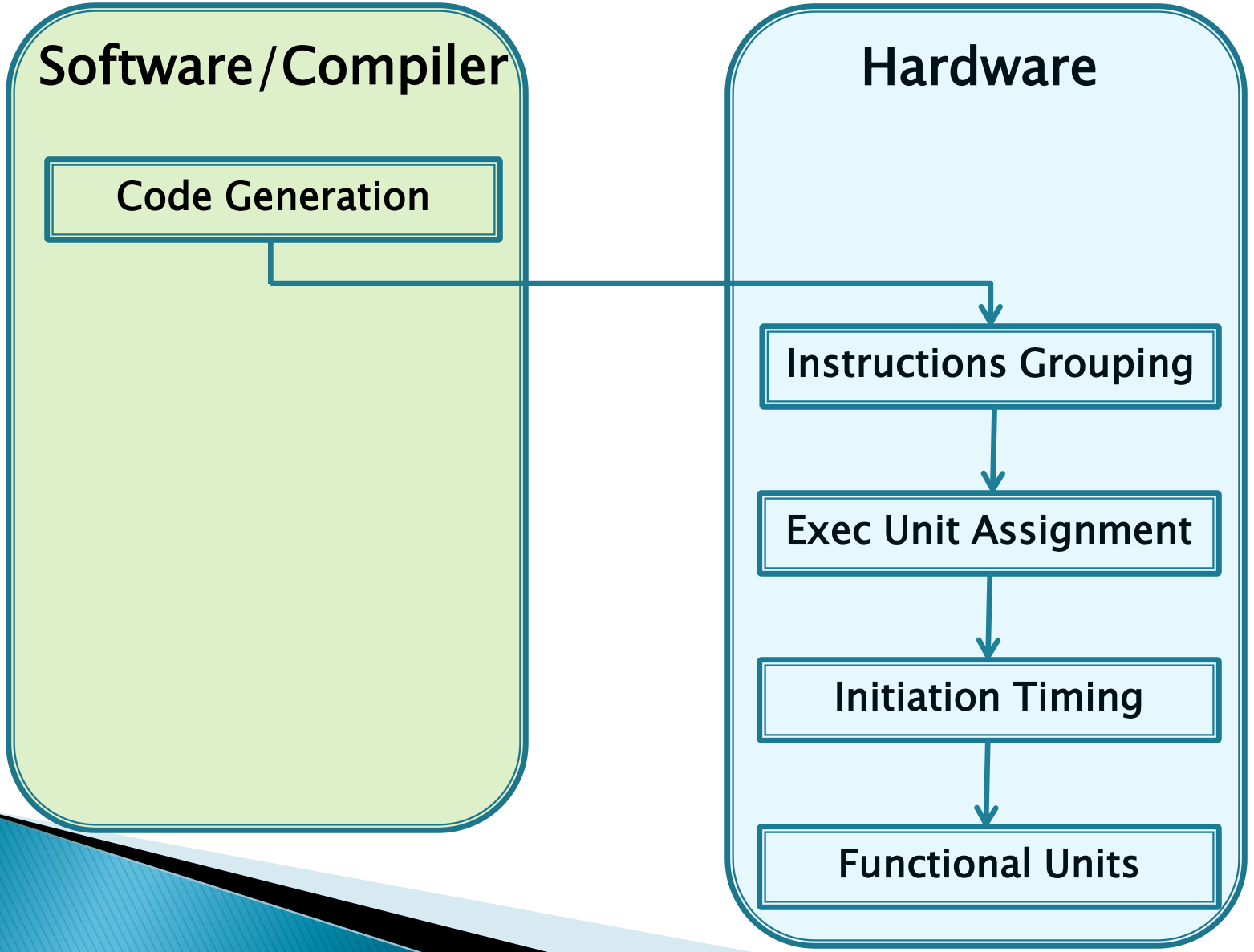
Multiple Issue, Static Scheduling: Dynamic Timing VLIW



Multiple Issue, Semi Static Scheduling: EPIC, Explicitly Parallel Instructions Computing

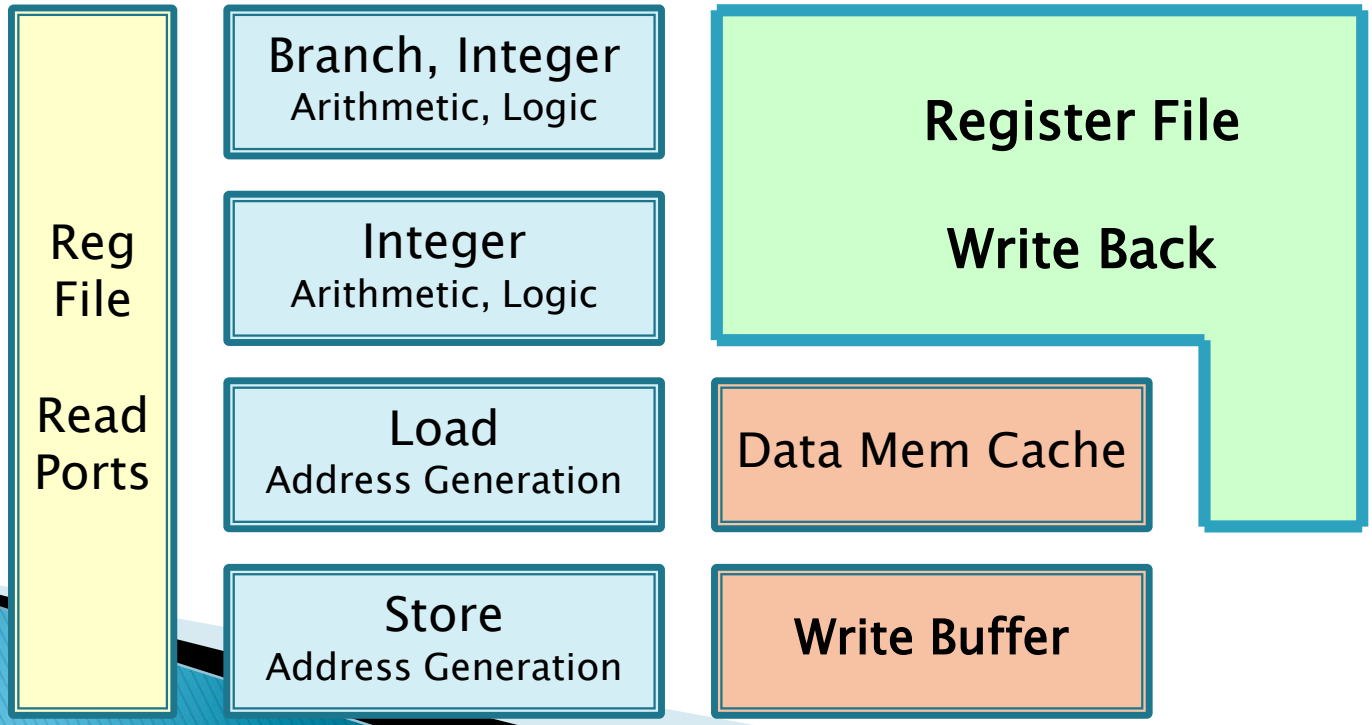


Multiple Issue, Dynamic Scheduling: Superscalar



ACA-MIPS Multiple Issue
In Order Execution (Static Scheduling)

- Multiple instruction are fetched and sent to decode stage.
- Up to four Instructions that match the slots functionality are dispatched to each issue slot.
- This is user/compiler responsibility to reorder codes in a way that maximum concurrency occurs.

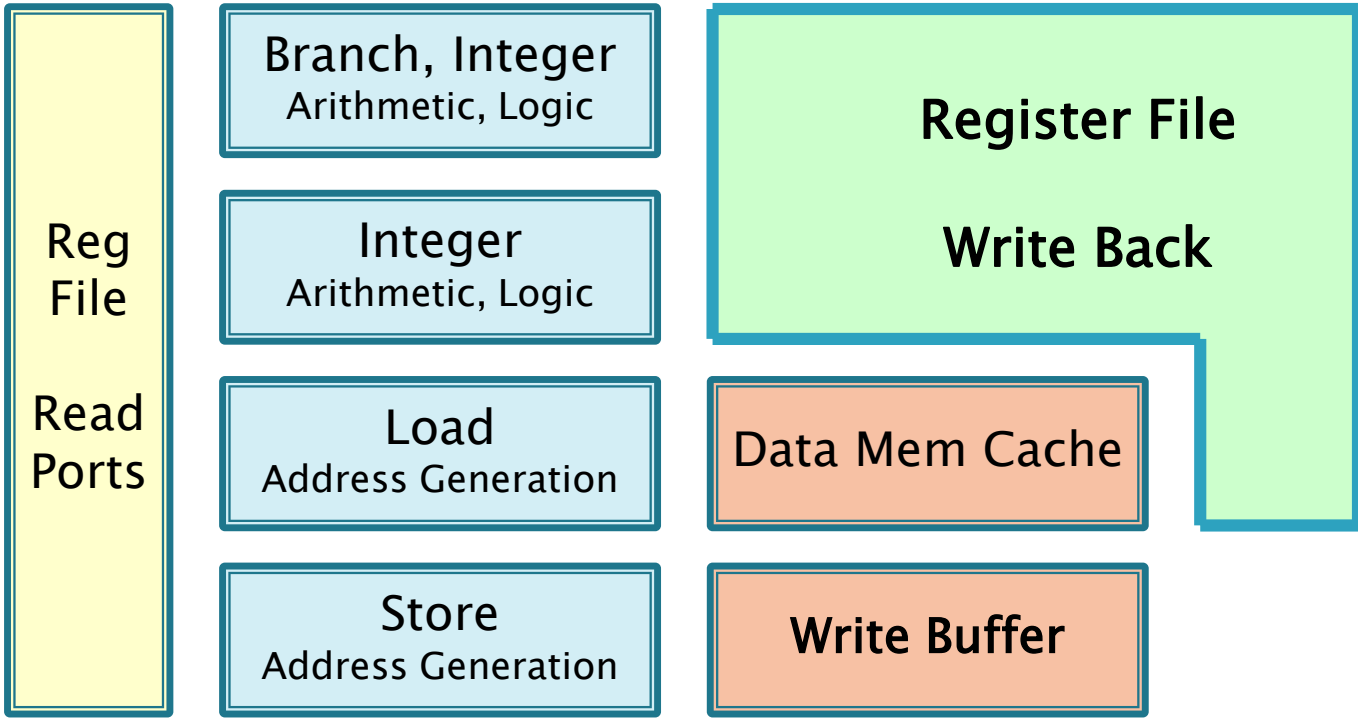


ACA–MIPS Multiple Issue
In Order Execution (Static Scheduling)

- add $1, $2, $3
- and $4, $5, $2
- lw $2, 100($0)
- sw $7, 200($12)

- lw $2, 100($0)
- add $1, $2, $3
- and $4, $5, $2
- sw $7, 200($12)

- add $1, $2, $3
- lw $2, 100($0)
- and $4, $5, $2
- sw $7, 200($12)

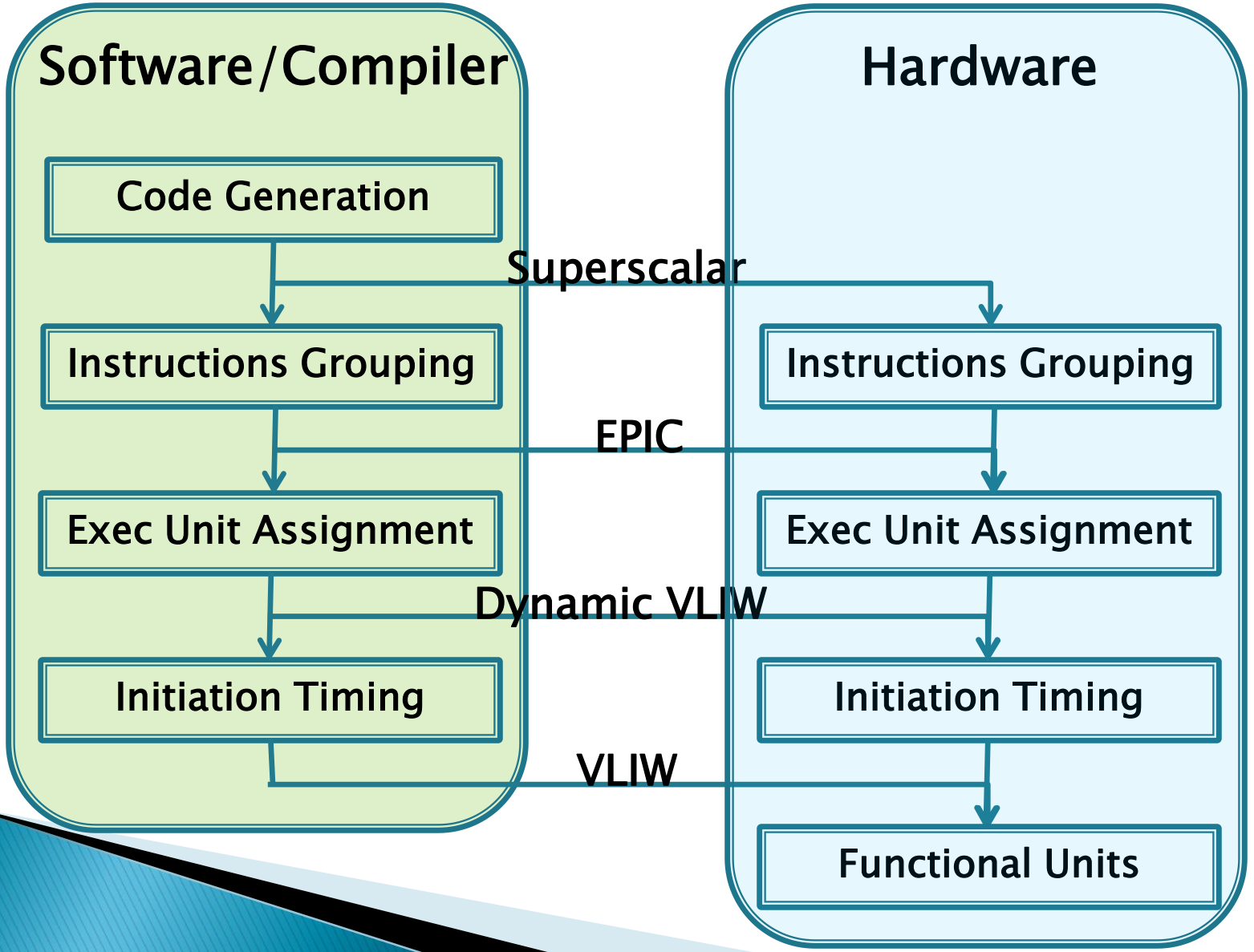


Single Issue Dynamic Scheduling and Out of Order Execution

- Instead of decoding only one instruction at each clock cycle, scan few instructions in a window.
- Dispatch earliest instruction in the window, not necessarily the head one, that has its arguments ready, instead of stalling for the head instruction
  - In example below, at T3, issue 5th instruction because arguments of 3rd and 4th instructions are not ready.
- Instruction results are committed (written in register file and/or memory) in order to respect SW behavior
- **Be aware of (time) Precise Exceptions**

```
1:ld    $t1, 0($t0)
2:ld    $t2, 4($t0)
3:add   $t3, $t1, $t2
4:st    $t3, 12($t0)
5:ld    $t4, 8($t0)
6:add   $t5, $t1, $t4
7:st    $t5, 16($t0)
```

Multiple Issue, Scheduling



Exec Units, No. of Issues, Scheduling

# Exec. Units	# Issues	Scheduling	Description, Example
1	1	Static	Regular 5-Stage MIPS Pipeline
1	1	Dynamic	See Single Issue Dynamic Scheduling and Out of Order Execution
N	1	Static/Dynamic	
N	N	Static	By ISA: **VLIW** At Run Time: See ACA-MIPS Multiple Issue In Order Execution (Static Schedu…
N	N	Dynamic	**Super Scalar**

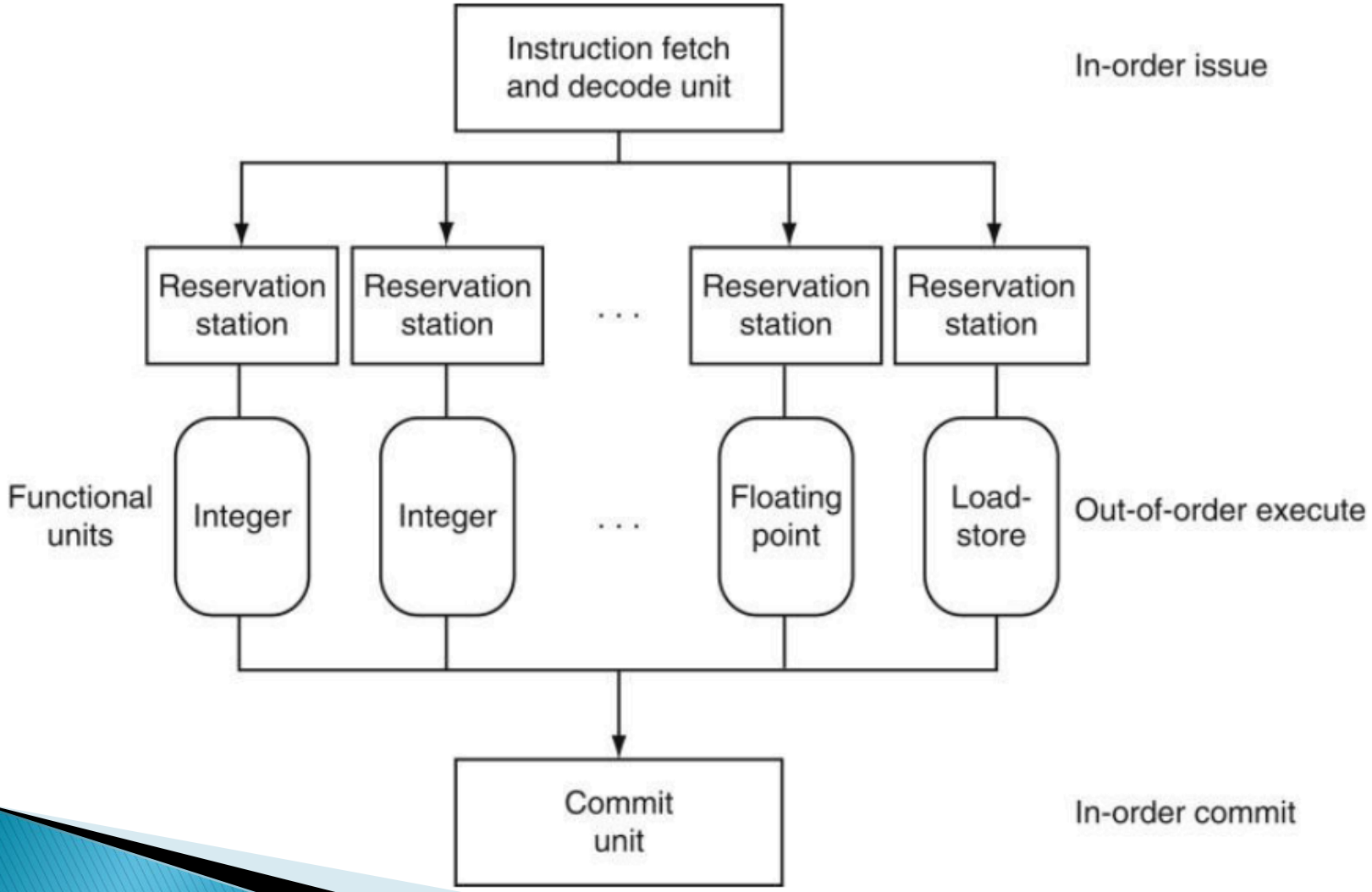
Instruction Level Parallelism (ILP)

- Pipelining
- Data Forwarding
- Branch Prediction
- Multiple Issue
- Superscalars
  - Out Of Order Execution
  - Register Renaming
  - Hyper Threading
- Real World Examples

Superscalars

- Processor datapath and controller should be divided into three major stages:
  - Instructions Fetch & Dispatch,
    - Branch Prediction unit
  - Concurrent Functional Units,
    - Pipelined or Sequential
  - Register and Memory Commit Unit.
- Reservation Stations are responsible to avoid and control data dependencies.

Superscalars, Major Stages



Superscalars, Examples

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue Width	Out-of-Order/ Speculation
Intel 486	1989	25 MHz	5	1	No
Intel Pentium	1993	66 MHz	5	2	No
Intel Pentium Pro	1997	200 MHz	10	3	Yes
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	Yes
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	Yes
Intel Core	2006	2930 MHz	14	4	Yes
Intel Core i5 Nehalem	2010	3300 MHz	14	4	Yes
Intel Core i5 Ivy Bridge	2012	3400 MHz	14	4	Yes

Register Renaming

```
ld    AX, M1
add   AX, 4
st    AX, M1
ld    AX, M2
add   AX, 8
st    AX, M2
```

```
ld    AX, M1
add   AX, 4
st    AX, M1
ld    BX, M2
add   BX, 8
st    BX, M2
```

```
ld    AX, M1; ld    BX, M2
add   AX, 4   ; add BX, 8
st    AX, M1; st    BX, M2
```

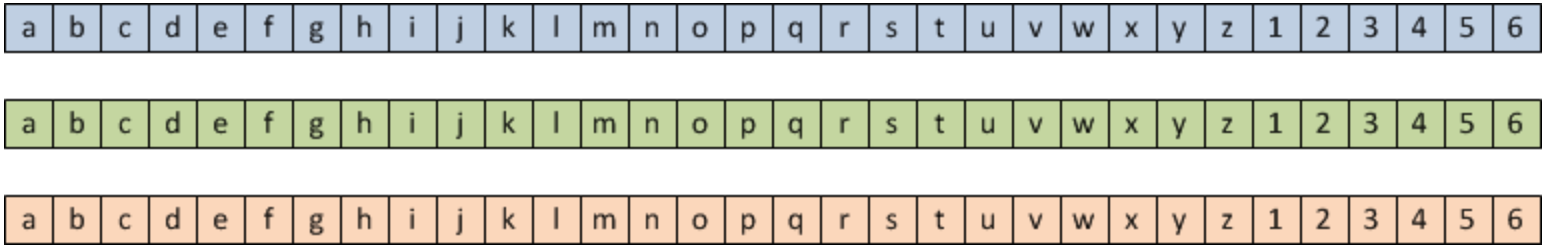
Proper use of register names by software/compiler can enable simultaneous issue of instructions.

Why not do the same in hardware?

Register Renaming

- Each ISA register is actually born when loaded by a register, memory location or a constant value.
- The register is reborn when a new value is loaded into it, as above.
- This means that each register has a limited lifetime: from each new loading to just prior to next reloading.
  - However, exact end of lifetime (death time) is unknown.
- Each ISA register is assigned to a real physical registers at born time, and detached from it at the death time.

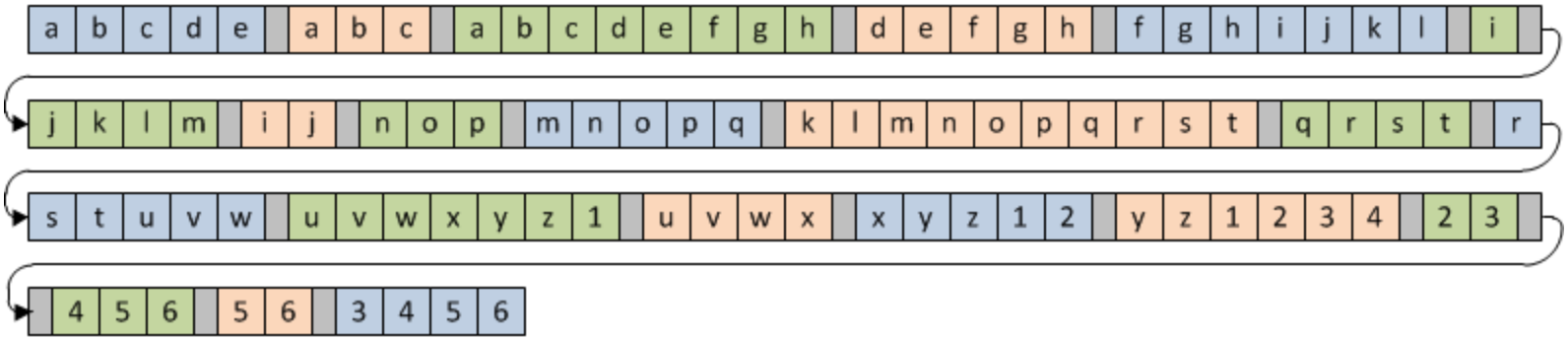
Multi Threading, Single Issue CPU



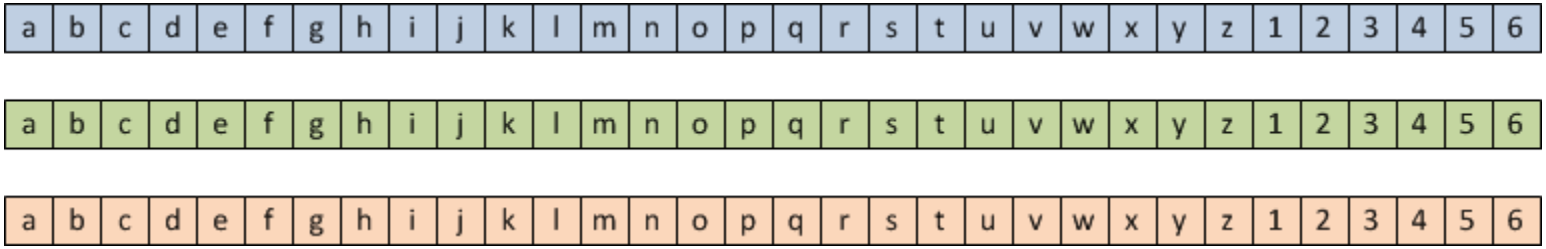
Interleave execution of several threads on a single processor.

User feels concurrent operation of threads.

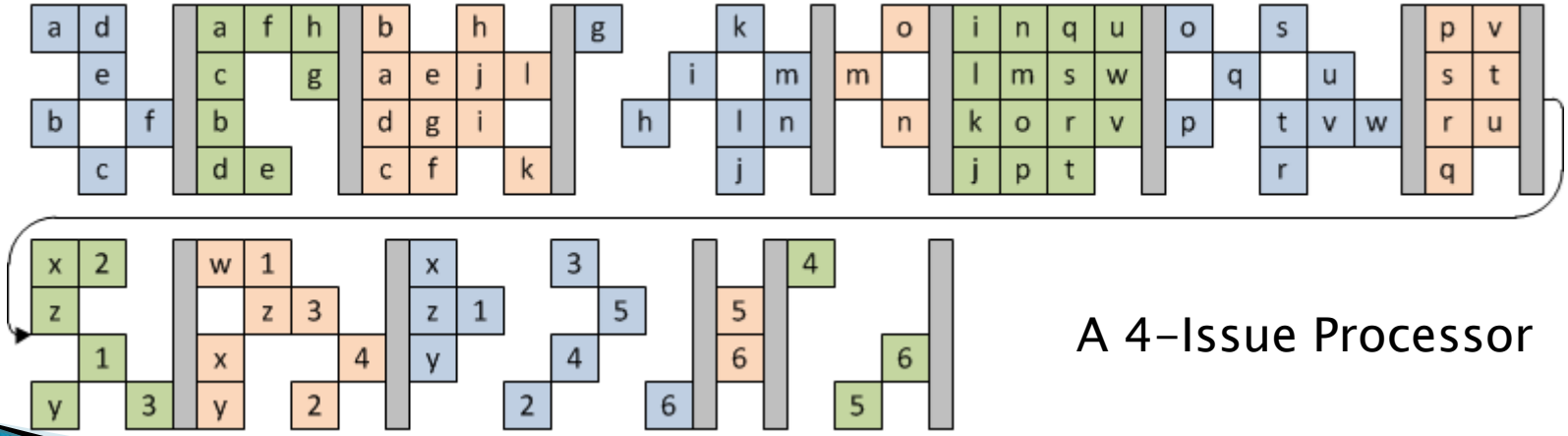
Thread **Context Switching** is the overhead to pay for.



Multi Threading, Multi Issue CPU

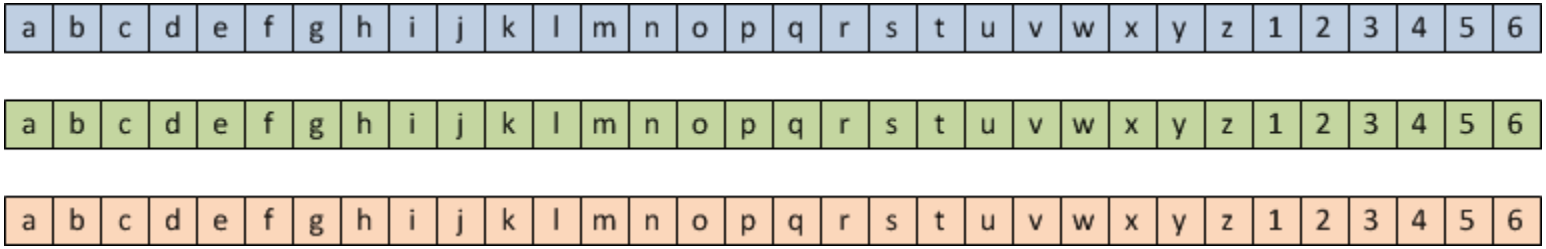


In a multi issue processor, whole processor, consisting of all issue slots are allocated to a thread, and context switched to another one when necessary. Issue slots occupancy depend on code behavior



A 4-Issue Processor

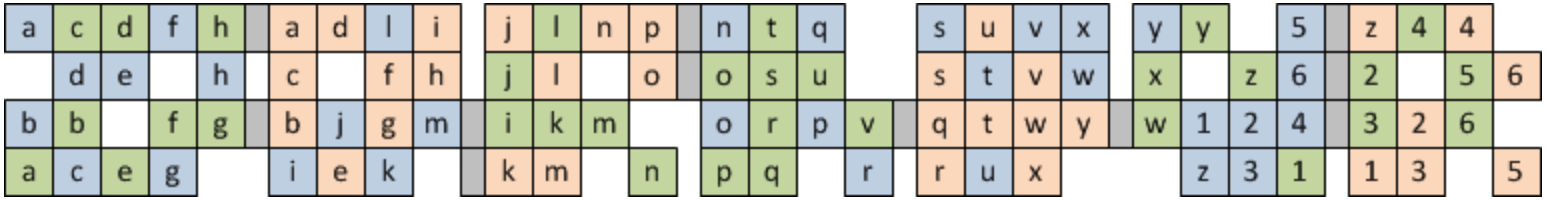
Simultaneous Multi Threading (SMT), aka Hyper Threading



Issue slots are fed by two or more threads. Intel HT handles two threads.

Context switching of running threads is done internally with zero overhead.

Operating System sees two underlying virtual logical processor.



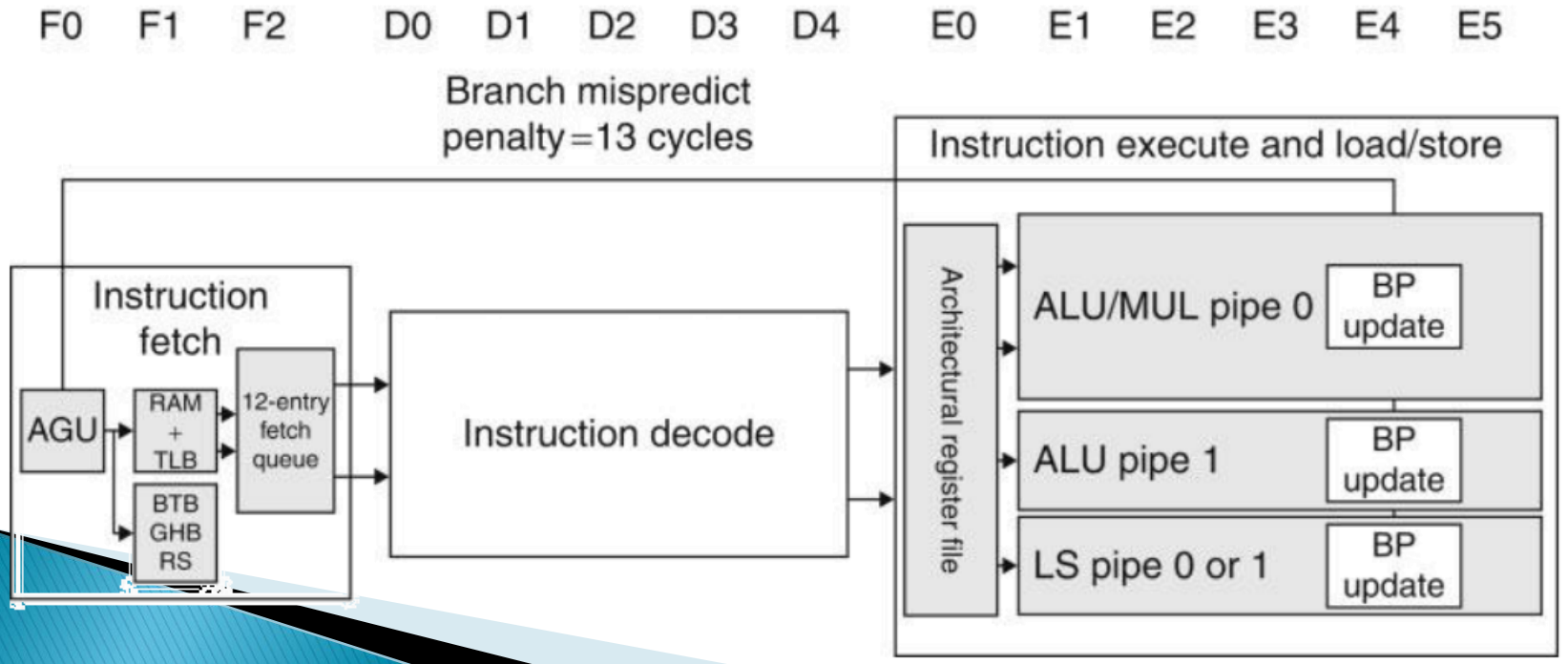
A 4-Issue Processor w/ SMT

Instruction Level Parallelism (ILP)

- Pipelining
- Data Forwarding
- Branch Prediction
- Multiple Issue
- Superscalars
  - Out Of Order Execution
  - Register Renaming
  - Hyper Threading
- Real World Examples

ARM Cortex A8

- 14 Pipeline Stage,
- 32 KB Data, 32 KB Instr. L1 Cache
- 2-Issue Static In-Order Scheduling,
- 2-Watts Power Consumption @ 1000 MHz



ARM Cortex A8 Branch Prediction

- 4K-entry global history buffer indexed by the branch history and current PC
- 512-entry 2-way set associative branch target buffer
  - If branch target buffer miss occurs, branch target is indeed computed
- Eight-entry return stack to track return addresses
- A misprediction has a penalty of 13 clock cycles

ARM Cortex A8 Instr. Decode & Dispatch

- Up to two instructions are dispatched in an in-order issue mechanism
- Scoreboarding is used to assign issue time

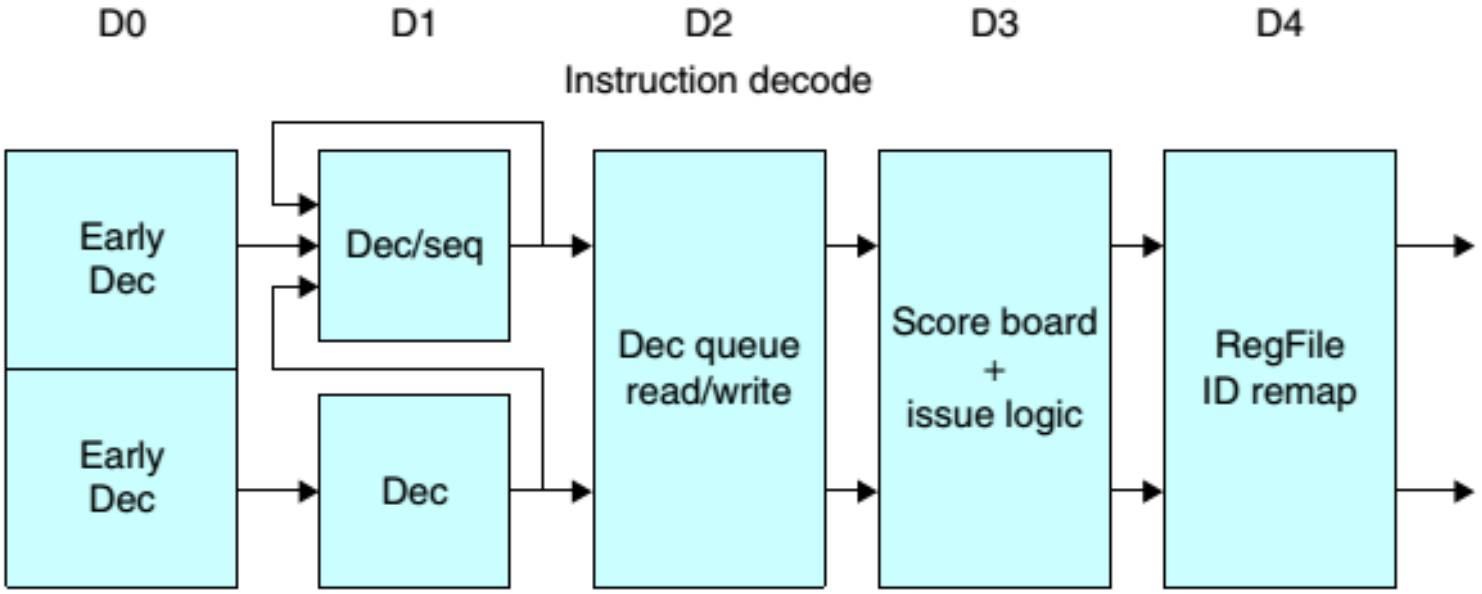
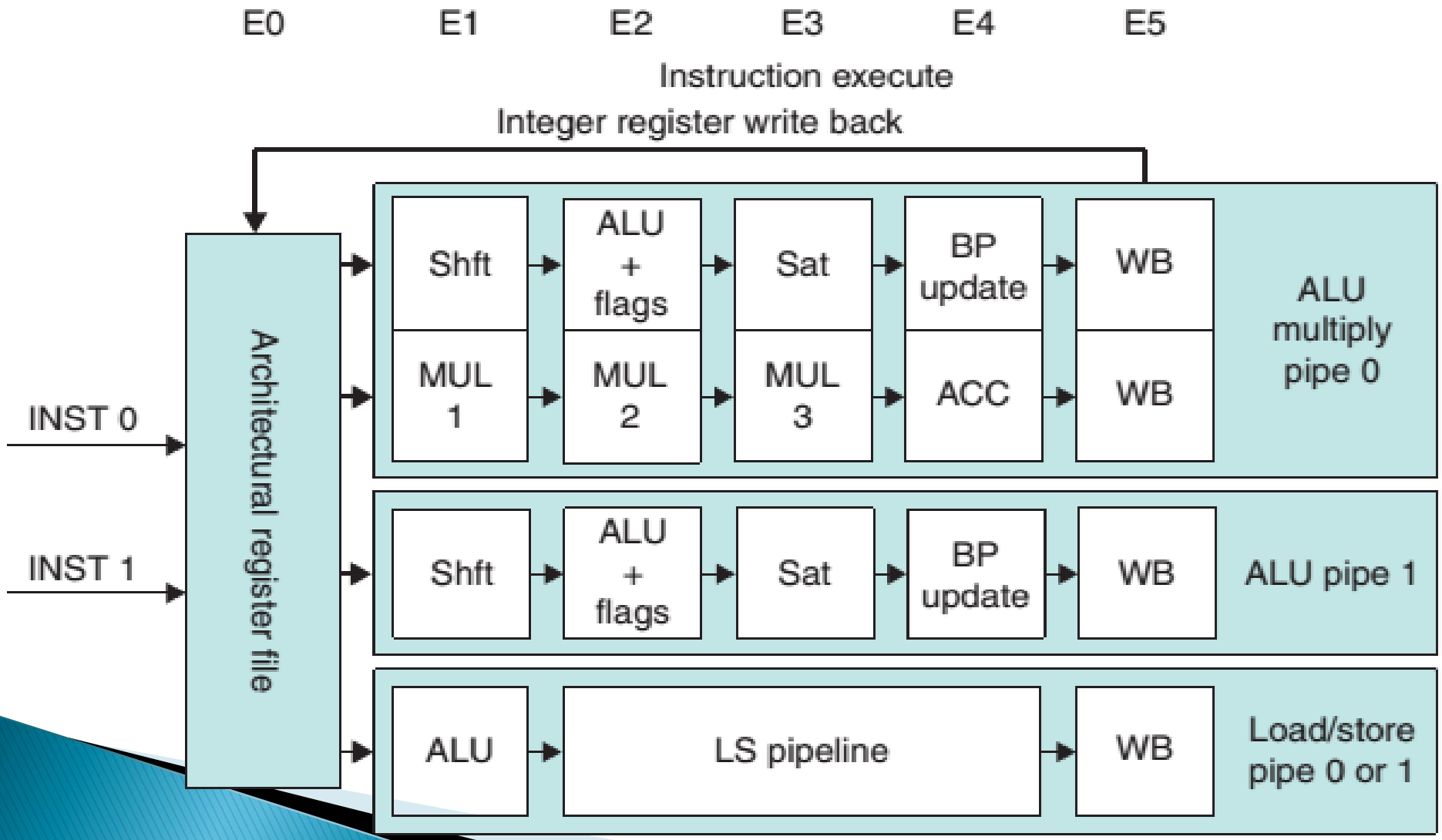


Figure 3.37 The five-stage instruction decode of the A8.

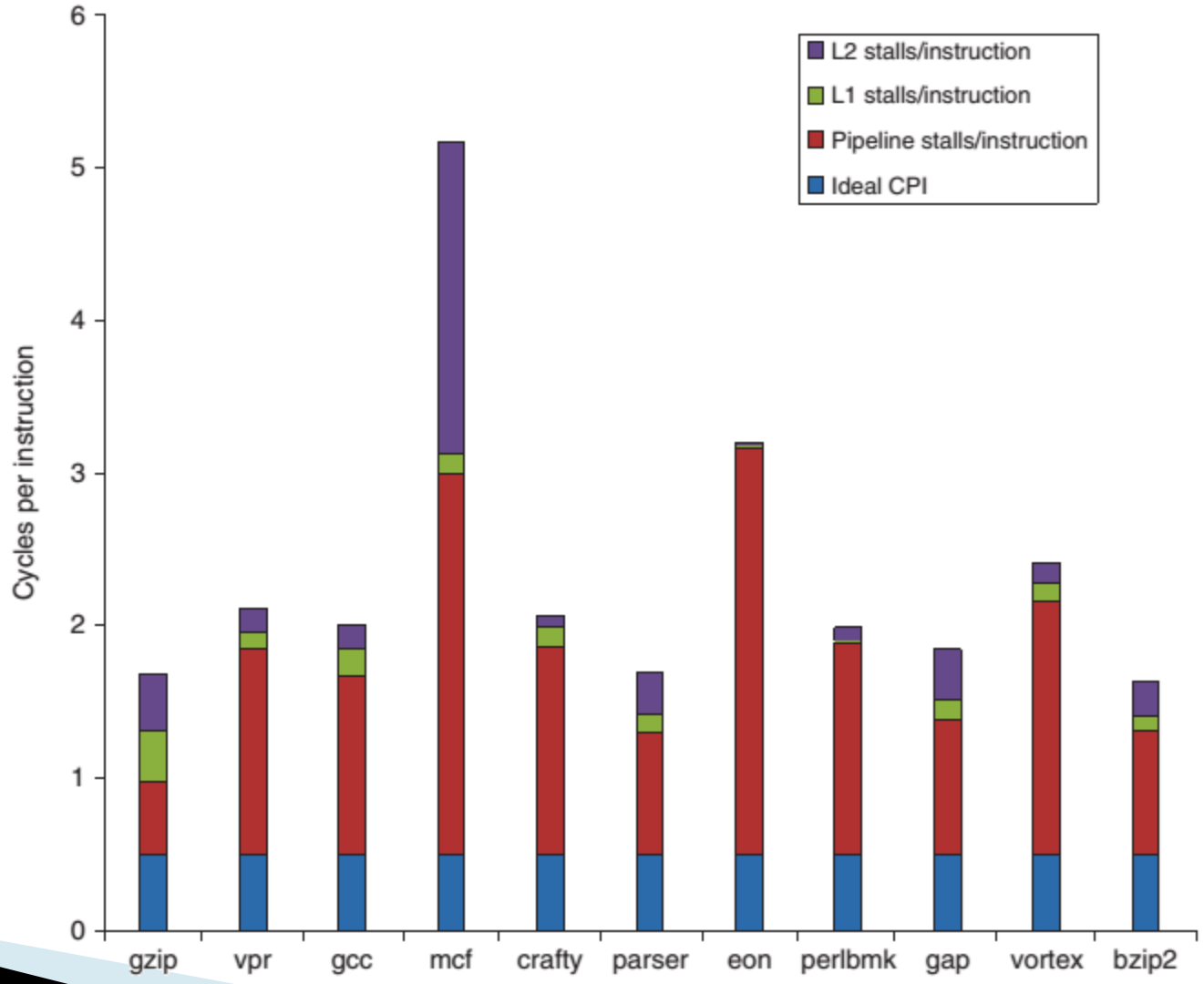
ARM Cortex A8 Execution Paths



ARM Cortex A8 Performance

Reminder:

- 2 Watts
- 1 GHz
- <$100

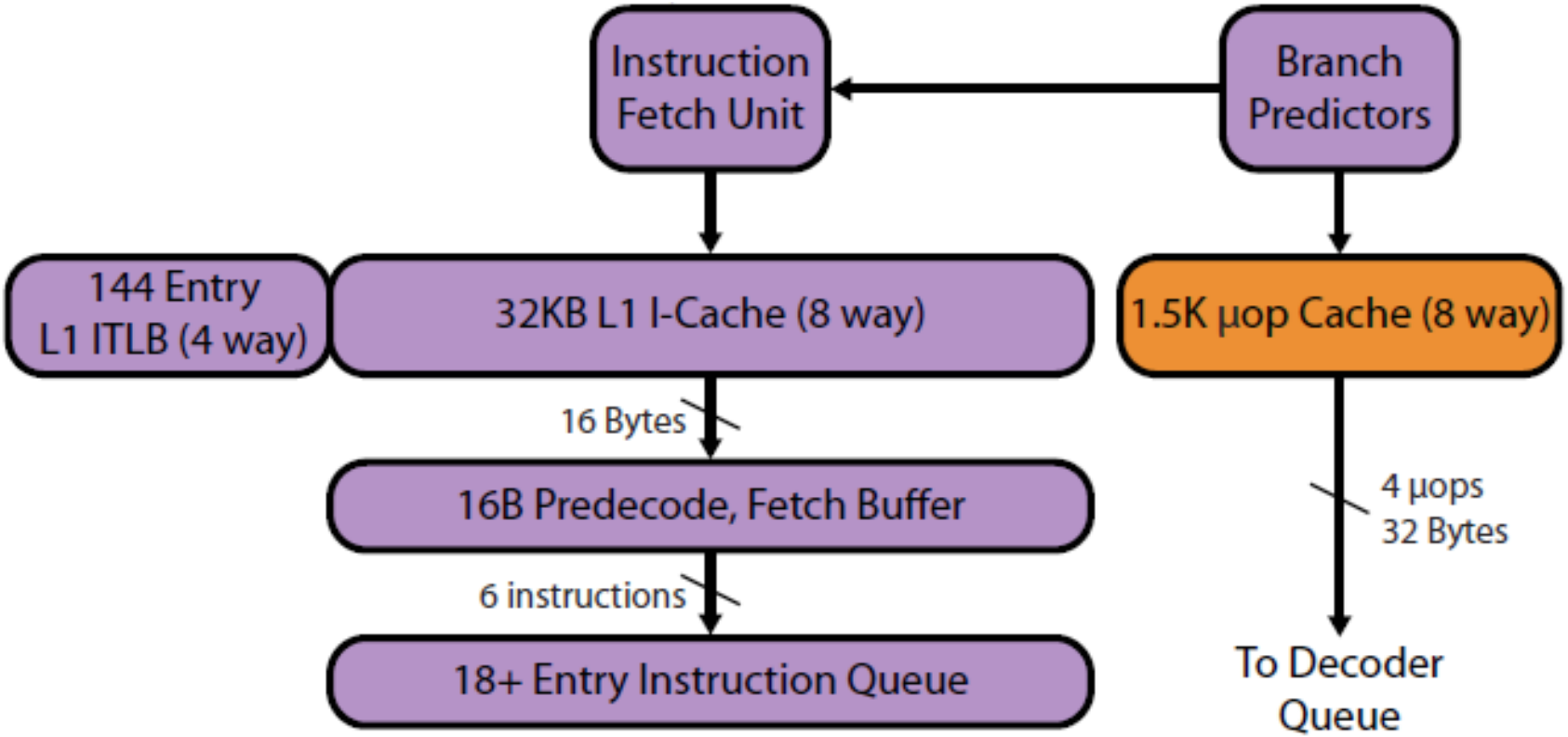


Intel Tick-Tock Strategy

Tick, 65nm:	P6, NetBurst	
Tock, 65nm:	Core	Merom
Tick, 45nm:	Core	Penryn
Tock, 45nm:	Nehalem	
Tick, 32nm:	Nehalem	Westmere
Tock, 32nm:	Sandy Bridge	
Tick, 22nm:	Sandy Bridge	Ivy Bridge
Tock, 22nm:	Haswell	
Tick, 14nm:	Haswell	Broadwell
Tock, 14nm:	Skylake	
Tock, 14nm:	Skylake	Kaby Lake
Tick, 10nm:	Skylake	Cannonlake

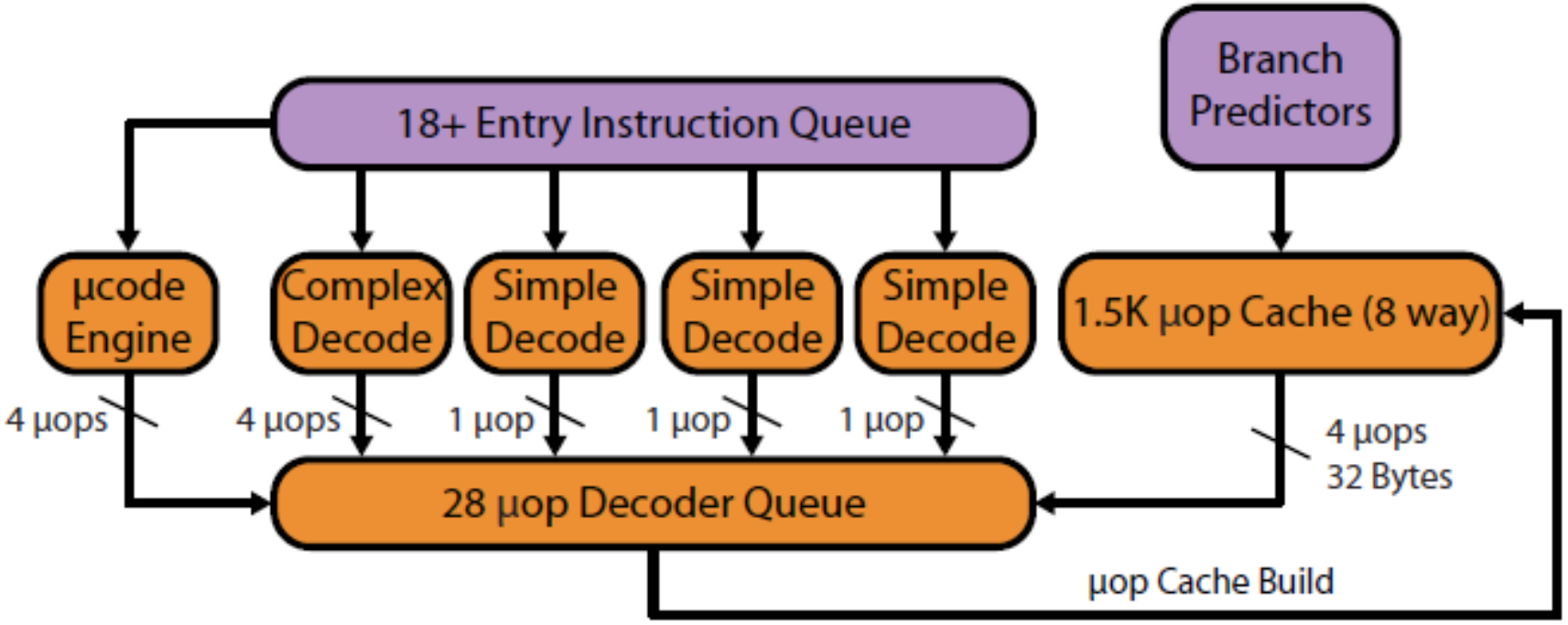
Intel Sandy Bridge Microarchitecture

Instruction Fetch



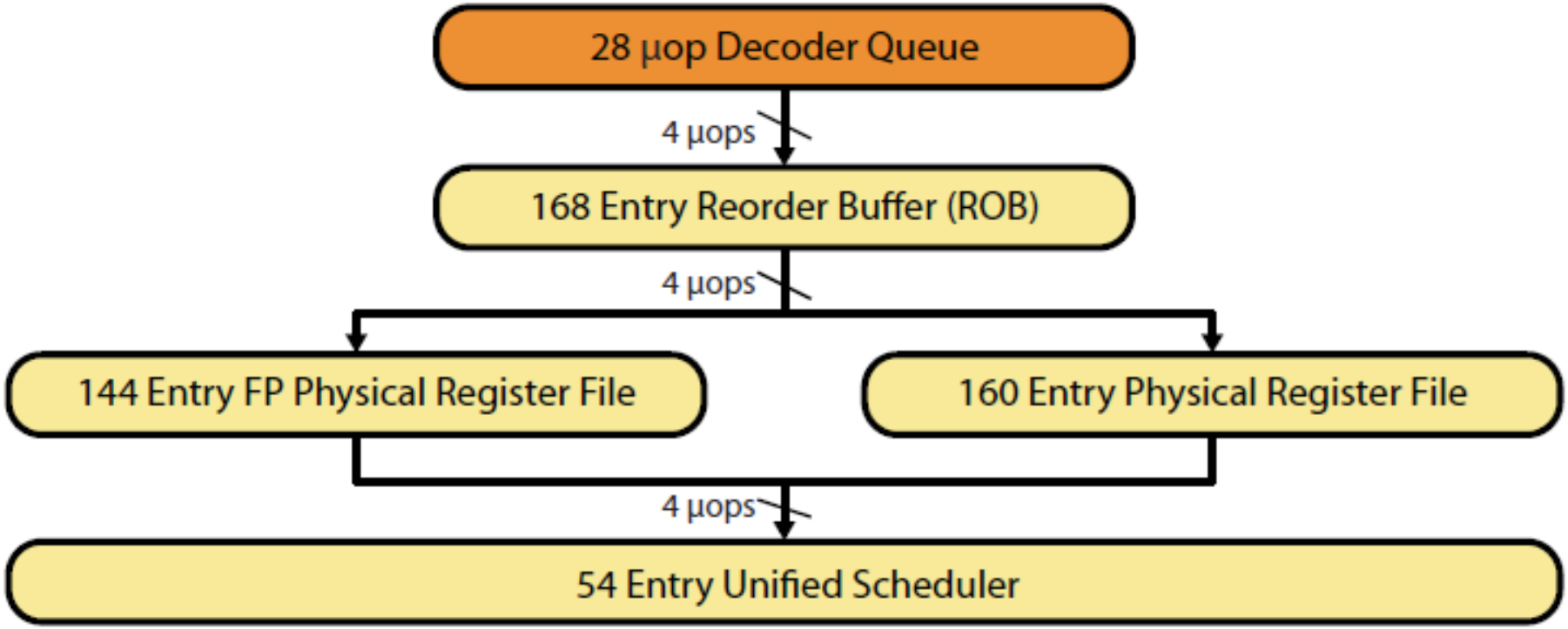
Intel Sandy Bridge Microarchitecture

Instruction Decode and µop Cache



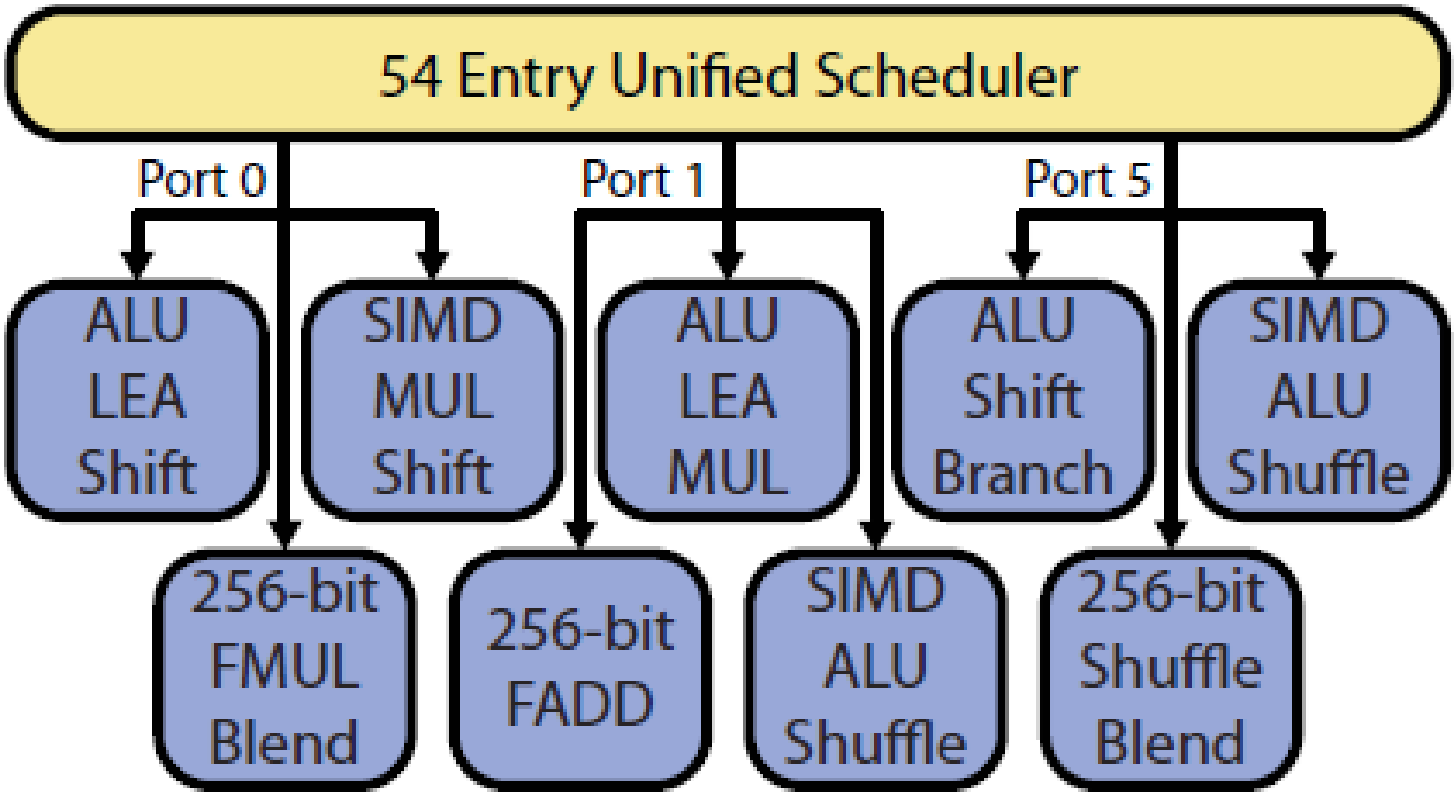
Intel Sandy Bridge Microarchitecture

Out Of Order (OOO) Execution



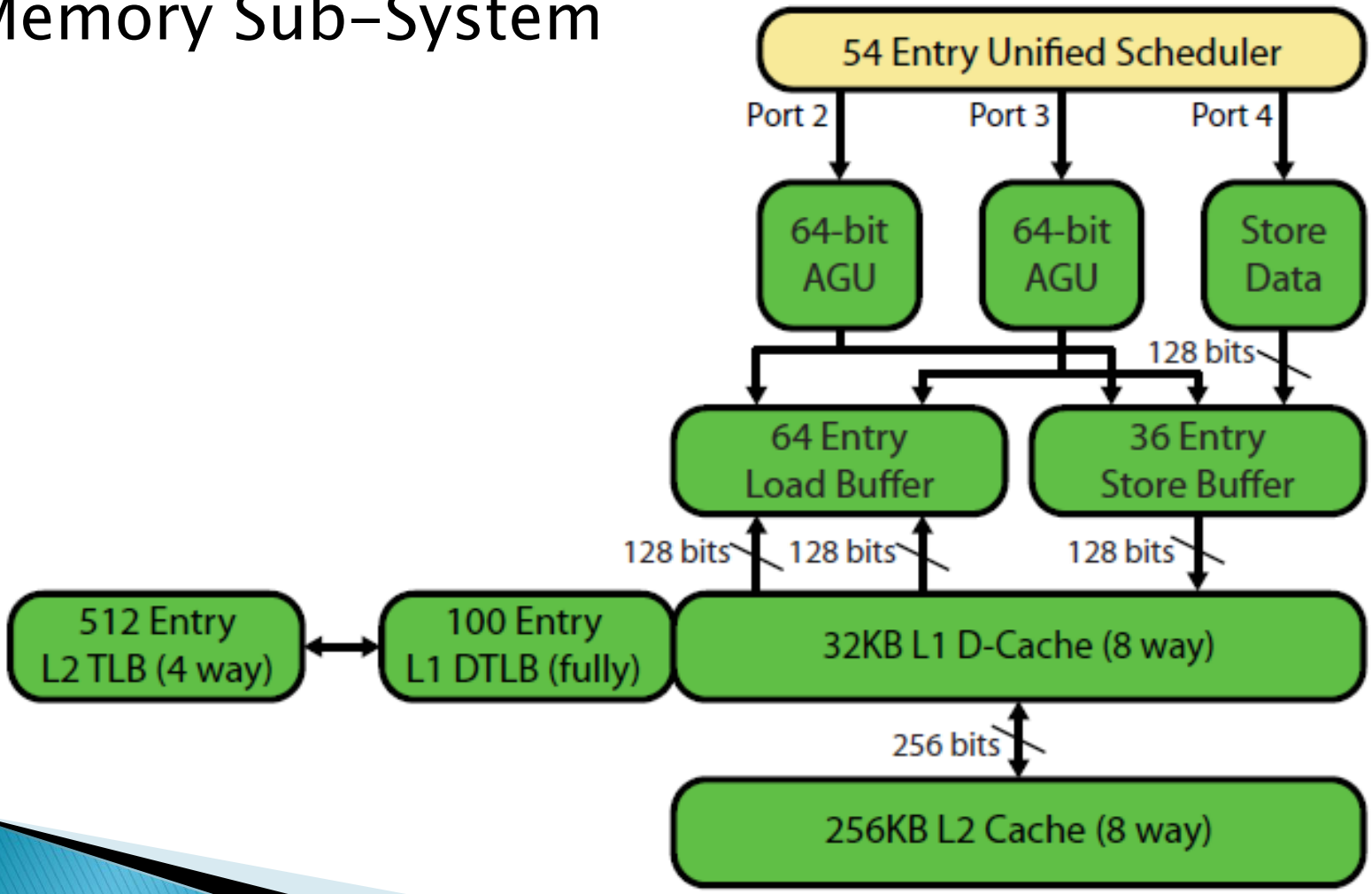
Intel Sandy Bridge Microarchitecture

Arithmetic & Logic Execution Units



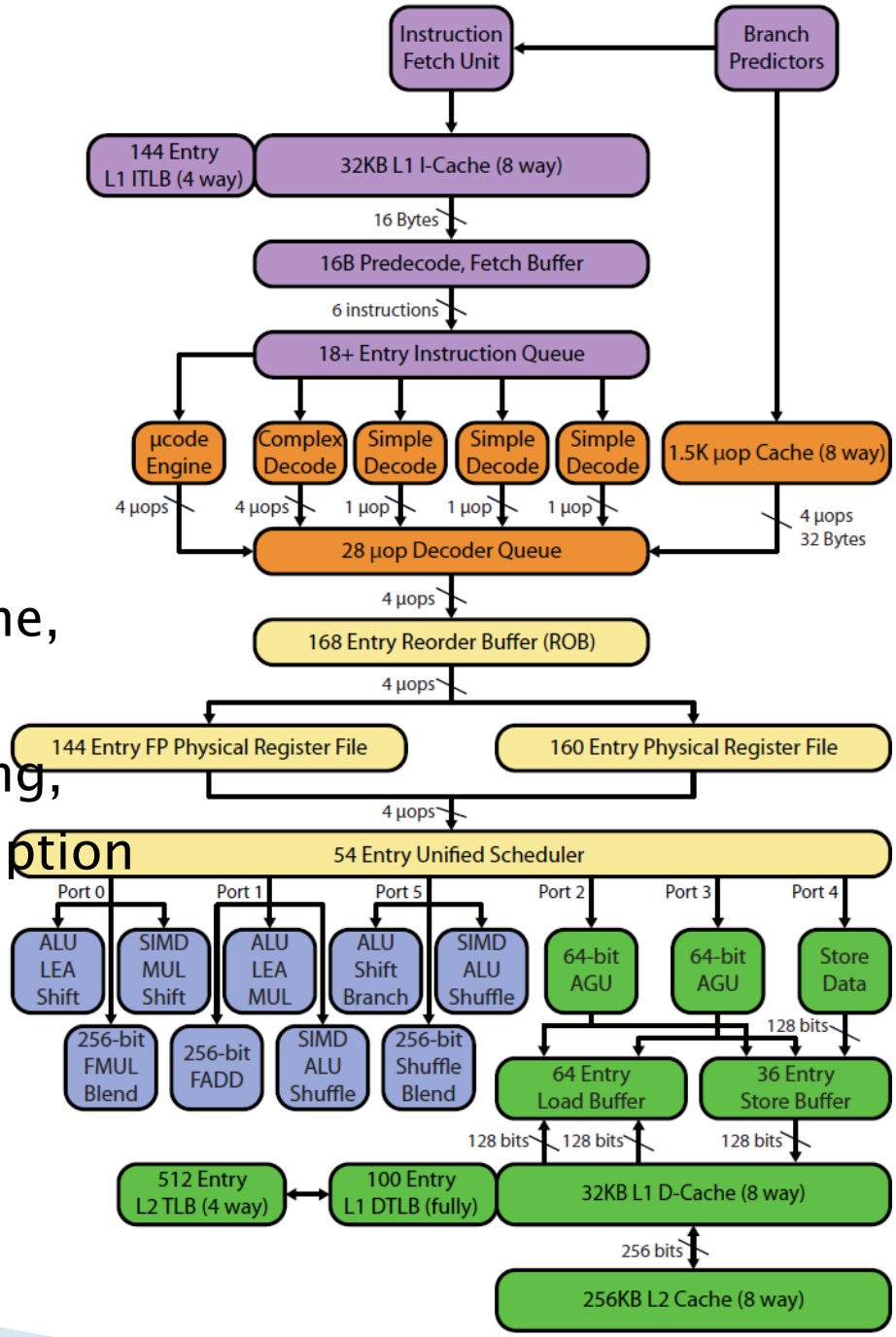
Intel Sandy Bridge Microarchitecture

Memory Sub-System

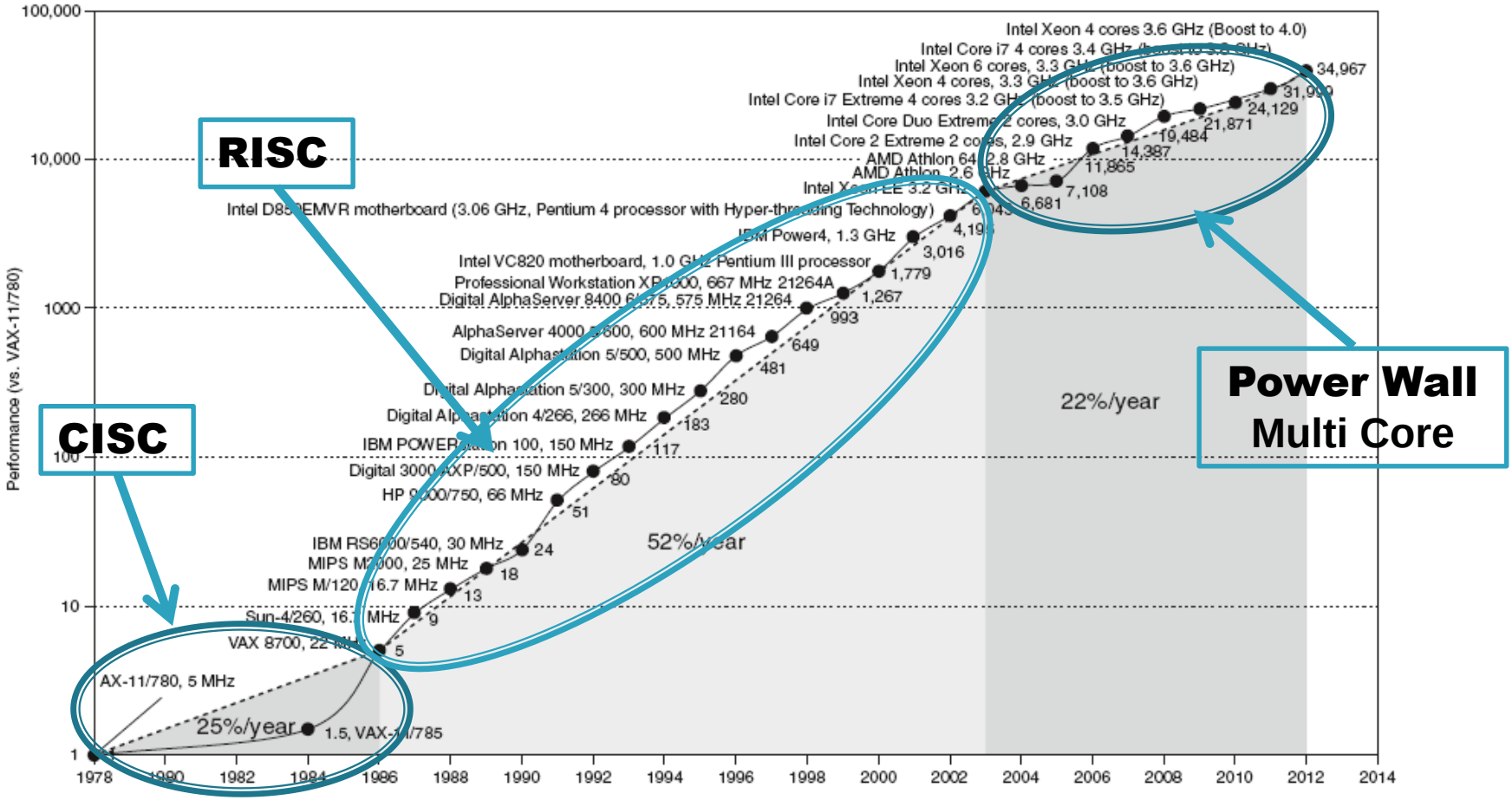


Intel Sandy Bridge Microarchitecture

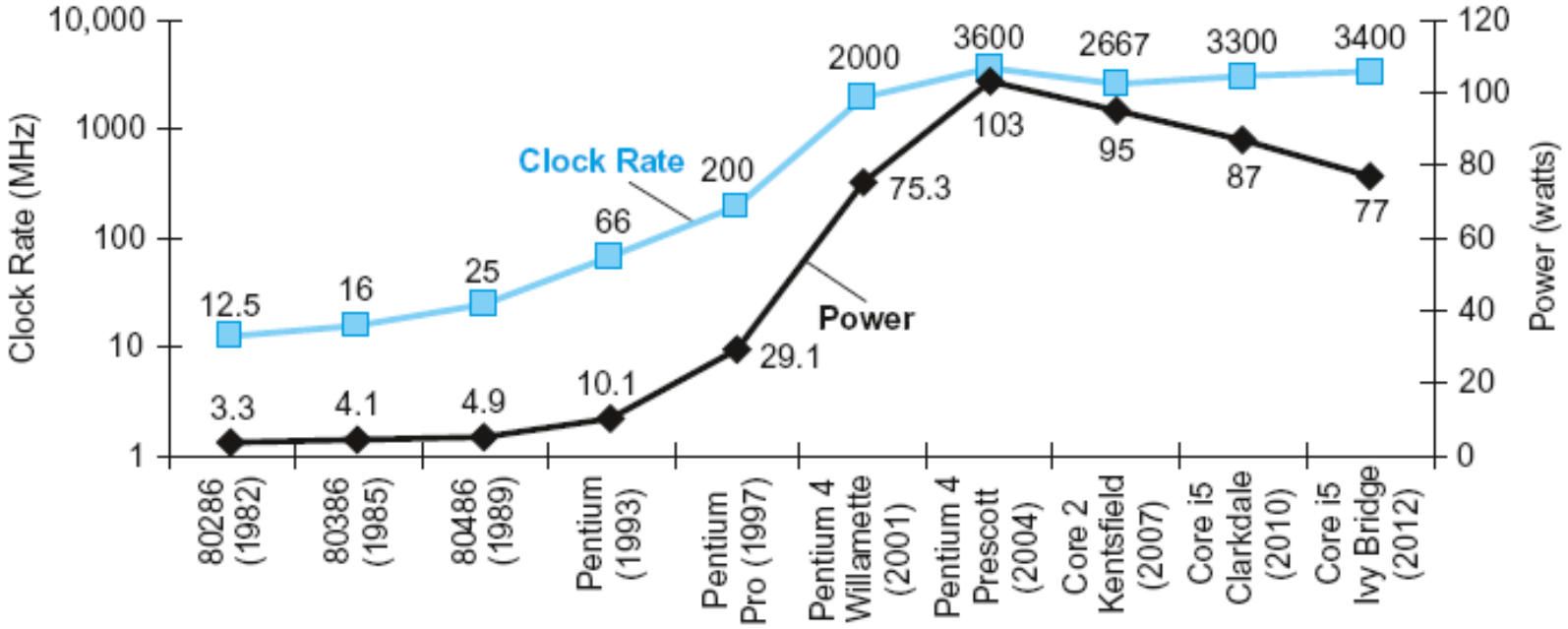
- 1-8 Physical Cores per Chip,
- 1-16 Logical Cores (Threads)
- 14-19 Pipeline Stage,
- Floating Point Instr. & Unit,
- 32 KB Data, 32 KB Instr. L1 Cache,
- 256 KB L2, 2-20 MB L3 Cache,
- 6-Issue Dynamic OOO Scheduling,
- Up to 150-Watts Power Consumption
- Up to 3.6 GHz, Turbo 4.0 GHz
- 130-400 sq. mm Die Area
- 500M to 2,270M Transistors
- Price up to ~ $4,000



Uniprocessors Performance Growth



Power Trend, Power Wall



- In CMOS IC Technology

  Static Power $\propto$ 1 / Gate Length

  Dynamic Power $\propto$ Load Cap. $\times$ Freq. $\times$ V_{DD}^{2}

Behind Power Wall

- Power wall does not allow more performance from single-threaded uniprocessors.
- But, real applications are of different types:
  1. Fully sequential, e.g. hashing, iterative equations,
  2. Parallel at algorithm level, e.g. matrix multiply,
  3. Pool of thousands almost independent threads.
- Only #2 and #3 cases can be processed faster by parallel processing:
  - SIMD: Single Instruction Multiple Data
  - MIMD: Multiple Instruction Multiple Data

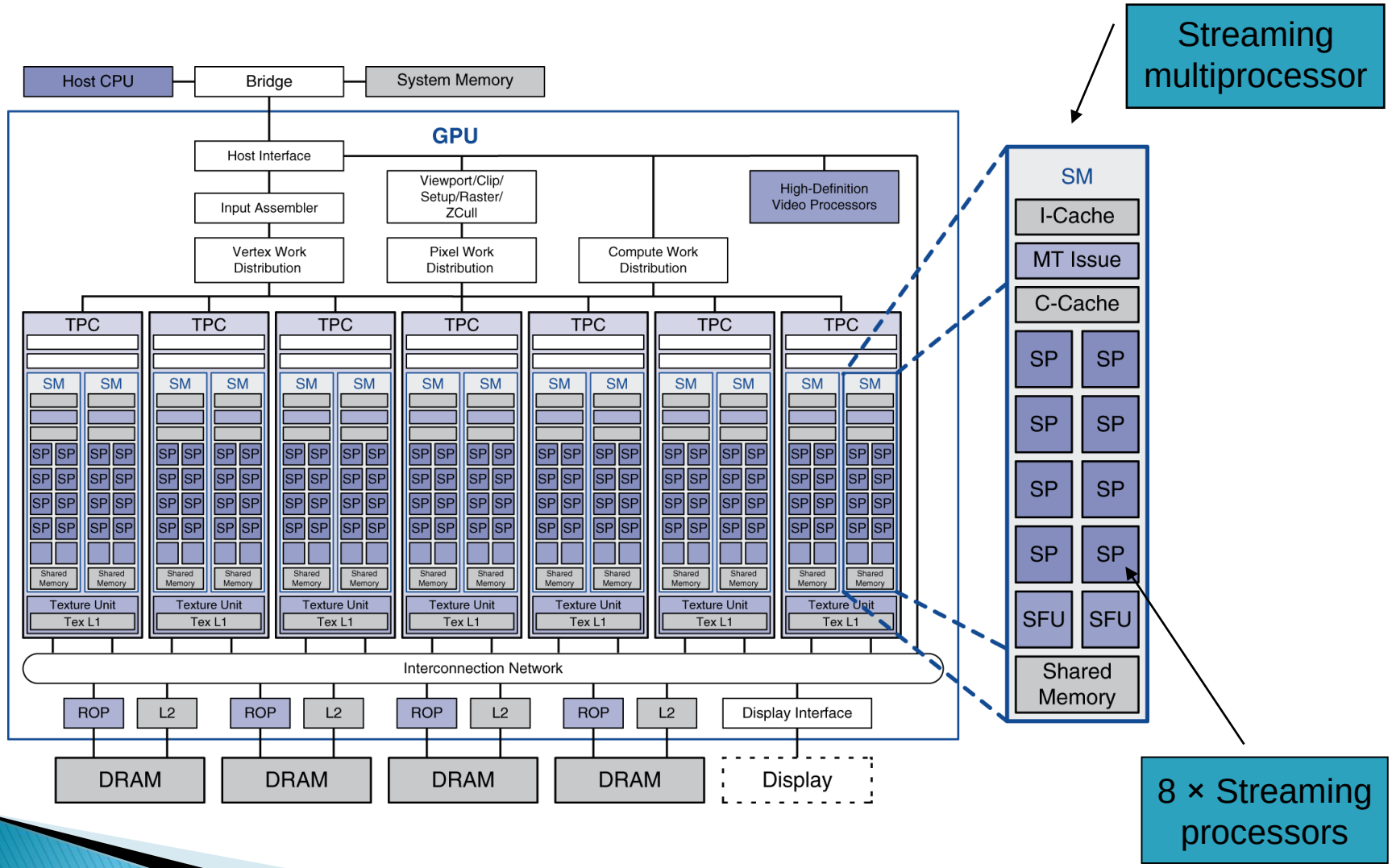
SIMD: Single Instruction Multiple Data

- Operate element-wise on vectors of data
- All processors execute the same instruction at the same time
  - Each with different data address, etc.
- Simplifies synchronization
- Reduced instruction control hardware
- Works best for highly data-parallel applications
  - Matrix & Vector operations
  - Graphic Processing (GPUs)

GPU Architectures

- Processing is highly data-parallel
  - GPUs are highly multithreaded
  - Graphics memory is wide and high-bandwidth
- Trend toward general purpose GPUs
  - Heterogeneous CPU/GPU systems
  - CPU for sequential code, GPU for parallel code
- Programming languages/APIs
  - DirectX, OpenGL
  - C for Graphics (Cg), High Level Shader Language (HLSL)
  - Compute Unified Device Architecture (CUDA)

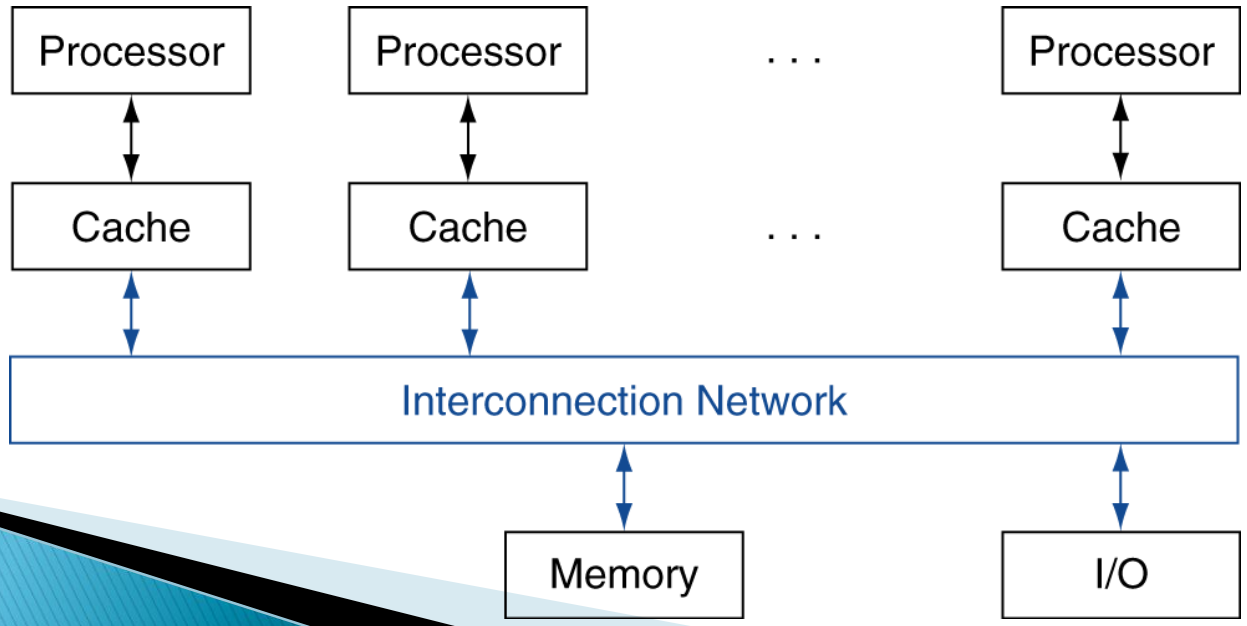
Example: NVIDIA Tesla



MIMD, Multi Core

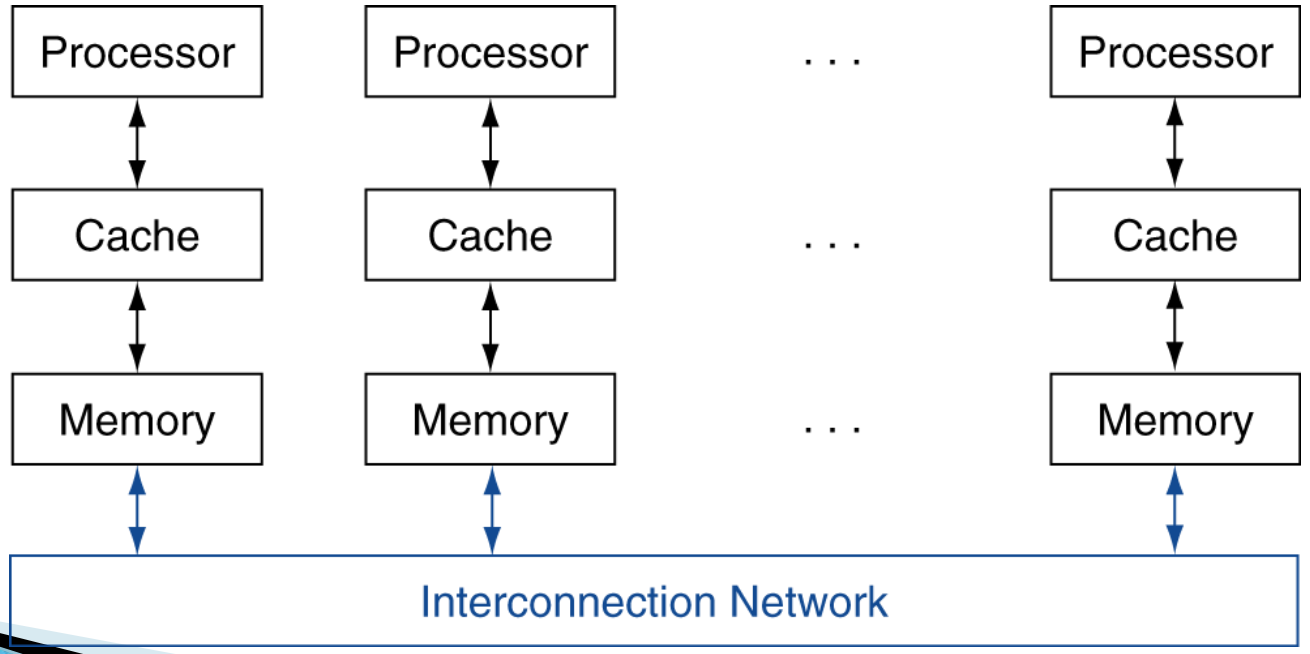
Shared **M**emory Multi Core/Processor

- Hardware provides single physical address space for all processors
- Synchronize shared variables using locks
- Memory access time
  - UMA (uniform) vs. NUMA (nonuniform)



Message Passing

- Each processor has private physical address space
- Hardware sends/receives messages between processors



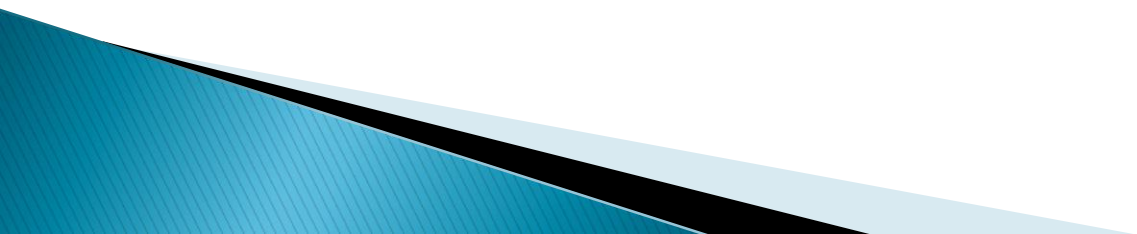
Loosely Coupled Clusters

- Network of independent computers
  - Each has private memory and OS
  - Connected using I/O system
    - e.g., Ethernet/switch, Internet
- Suitable for applications with independent tasks
  - Web servers, databases, simulations, …
- High availability, scalable, affordable
- Problems
  - Administration cost
  - Low interconnect bandwidth
    - c.f. processor/memory bandwidth on an SMP

Grid Computing

- Separate computers interconnected by long-haul networks
  - e.g. Internet connections
  - Work units farmed out, results sent back
- Can make use of idle time on PCs
  - e.g. SETI@home, World Community Grid

Cloud Computing



Micro Servers