COMP1001

Computer Systems

Dr. Vasilios Kelefouras

Email: v.kelefouras@plymouth.ac.uk

Website:
https://www.plymouth.ac.uk/staff/vasilios-kelefouras

Modern CPU Hardware Architectures

Outline

- Superscalar processors
- Superpipelining processors
- In order and out of order processors
- RISC, CISC processors
- Moore’s Law and hardware architecture trends

Superscalar Processors

- You have been taught about CPU pipelining but performance is never enough
- Any other options?
- **Why not make the pipeline deeper and deeper?**
  - By adding more pipeline stages the clock cycle is reduced
  - But beyond some point, adding more pipe stages doesn't help, because control/data hazards increase, and become costlier

Superscalar Processors

- A superscalar processor can execute more than one instruction during a clock cycle by simultaneously dispatching multiple instructions to different execution units on the processor
- Duplicates the pipeline to accommodate **Instruction Level Parallelism** (ILP)
  - Note that duplicating HW in just one pipe stage doesn't help, e.g., when having 2 ALUs, the bottleneck moves to other stages
- It therefore achieves more throughput (the number of instructions that can be executed in a unit of time) that would otherwise be impossible at a given clock rate
- Superscalar machines issue a variable number of instructions each clock cycle, up to some maximum
  - **instructions must be independent – no data or control dependencies**

Pipeline vs 2 way Superscalar (pipelined)

	1	2	3	4	5	6	7	8
Instruction1	IF	ID	EX	MEM	WB			
Instruction2		IF	ID	EX	MEM	WB		
Instruction3			IF	ID	EX	MEM	WB	
Instruction4				IF	ID	EX	MEM	WB

Instruction1	IF	ID	EX	MEM	WB	
Instruction2	IF	ID	EX	MEM	WB	
Instruction3		IF	ID	EX	MEM	WB
Instruction4		IF	ID	EX	MEM	WB

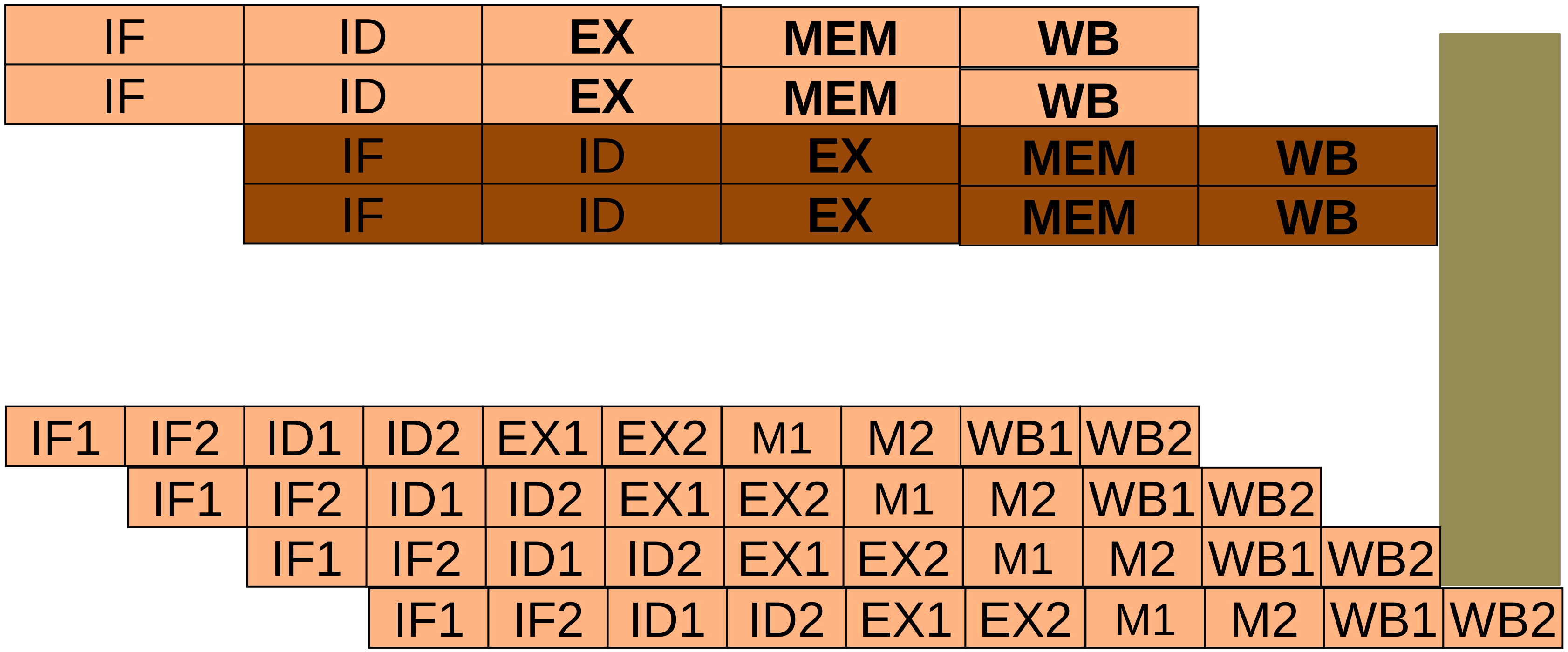
Careful: If the instructions are interdependent, then superscalar does not improve performance at all

Superscalar Hardware Support

- Extra hardware to simultaneously fetch multiple instructions is needed
- Hardware logic to determine data dependencies is needed, involving the registers’ values
- Extra hardware (functional units) to execute multiple instructions in parallel
- Extra hardware (functional units) to issue multiple instructions in parallel
- More extra hardware for more complicated notions (out of the scope of this module)

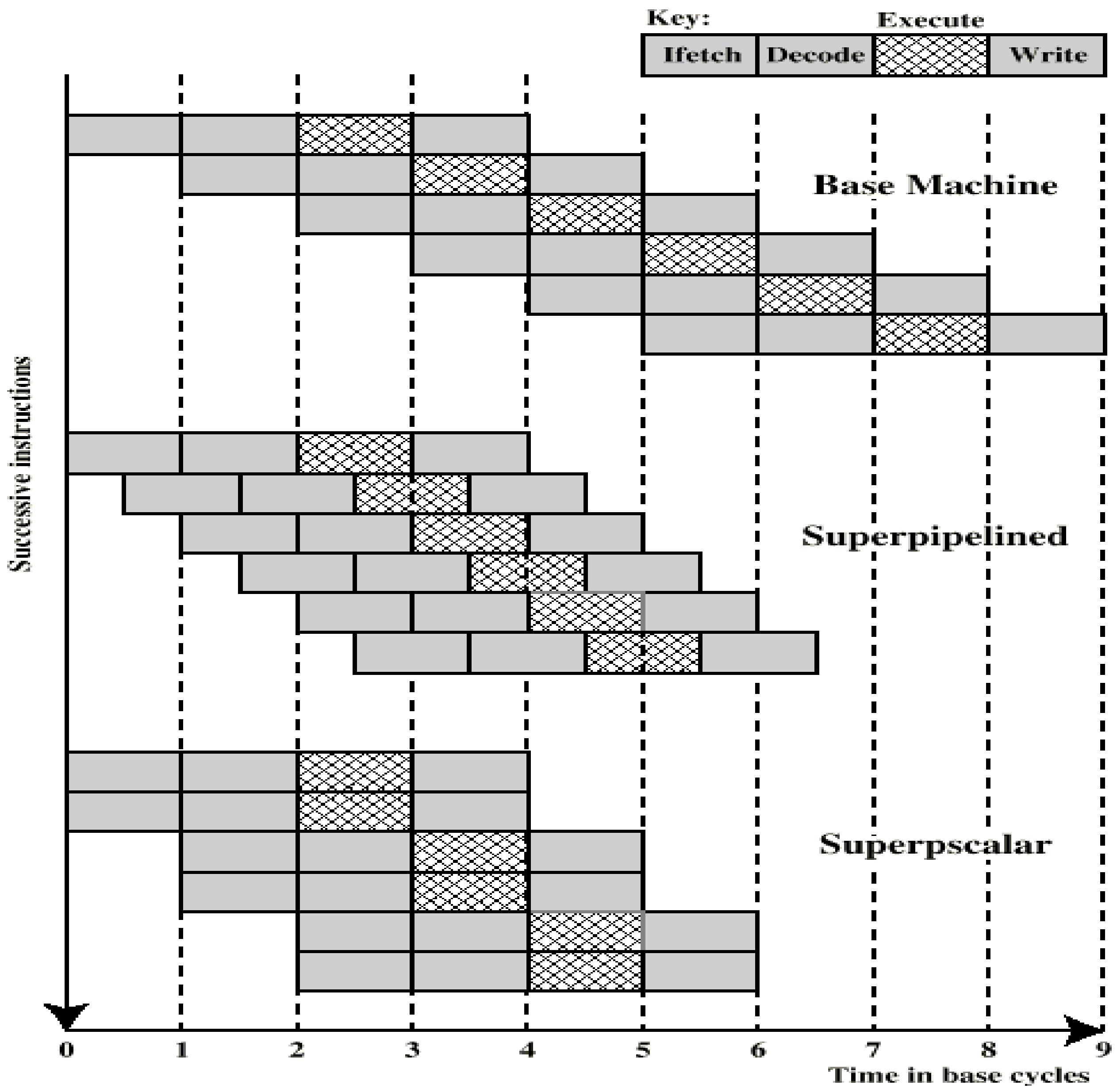
Superscalar vs Superpipelining (1)

- **Superpipelining:** *Vaguely defined as deep pipelining, i.e., lots of stages*
- **Superscalar issue complexity:** limits super-pipelining
- How to compare them?
  - e.g., 2-way Superscalar vs. two stages



IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB
IF1 IF2 ID1 ID2 EX1 EX2 M1 M2 WB1 WB2
IF1 IF2 ID1 ID2 EX1 EX2 M1 M2 WB1 WB2
IF1 IF2 ID1 ID2 EX1 EX2 M1 M2 WB1 WB2
IF1 IF2 ID1 ID2 EX1 EX2 M1 M2 WB1 WB2

Superscalar vs Superpipelining (2)



Superscalar Problem

- what if two successive instructions can’t be executed in parallel?
  - **Superscalar operation is double impacted by a stall**

- Superscalar depends upon:
  - Instruction level parallelism (ILP)
  - Compiler based optimization

- Limited by
  - Data dependencies
  - Control dependencies

Is superscalar good enough?

- **A superscalar processor can fetch, decode, execute more than one instructions in parallel**
- But…
  - Can execute only independent instructions in parallel
    - Whereas adjacent instructions are often dependent
  - **So the utilization of the second pipe is often low**
- **Solution: out-of-order execution**
  - Execute independent instructions in a different, more efficient order
  - A specific HW mechanism examines a sliding window of consecutive instructions (**instruction window** – it is a small memory)
  - Ready instructions get picked up from window and executed out of order
  - Instructions enter (**dispatched**) and leave (**committed**) the instruction window in program order, and an instruction can only leave the window when it is the oldest instruction in the window and it has been completed

Out of Order Processors (1)

- **The pipelines we have studied so far are statically scheduled and inorder**, i.e., instructions are executed in program's order
- If a hazard causes stall cycles, then all instructions up to the offending instruction are stalled
  - Forwarding, branch prediction, and other techniques can reduce the number of stall cycles, but sometimes a stall is unavoidable

```
IMUL  R3 ← R1, R2
ADD   R3 ← R3, R1
IMUL  R5 ← R6, R8
ADD   R7 ← R6, R8
```

- **Consider the following assembly code in a superscalar processor**
  - The first instruction stalls the second pipe

- **What about changing the order of the instructions?**
  - No stall exists now
  - Careful: data dependencies must be preserved

```
IMUL  R3 ← R1, R2
IMUL  R5 ← R6, R8
ADD   R7 ← R3, R5
ADD   R3 ← R3, R1
```

Data flow analysis

S1: r1=r0/7 *//division needs many cycles*

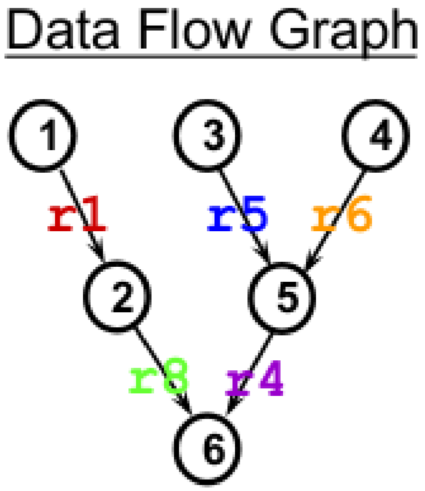
S2: r8=r1+r2

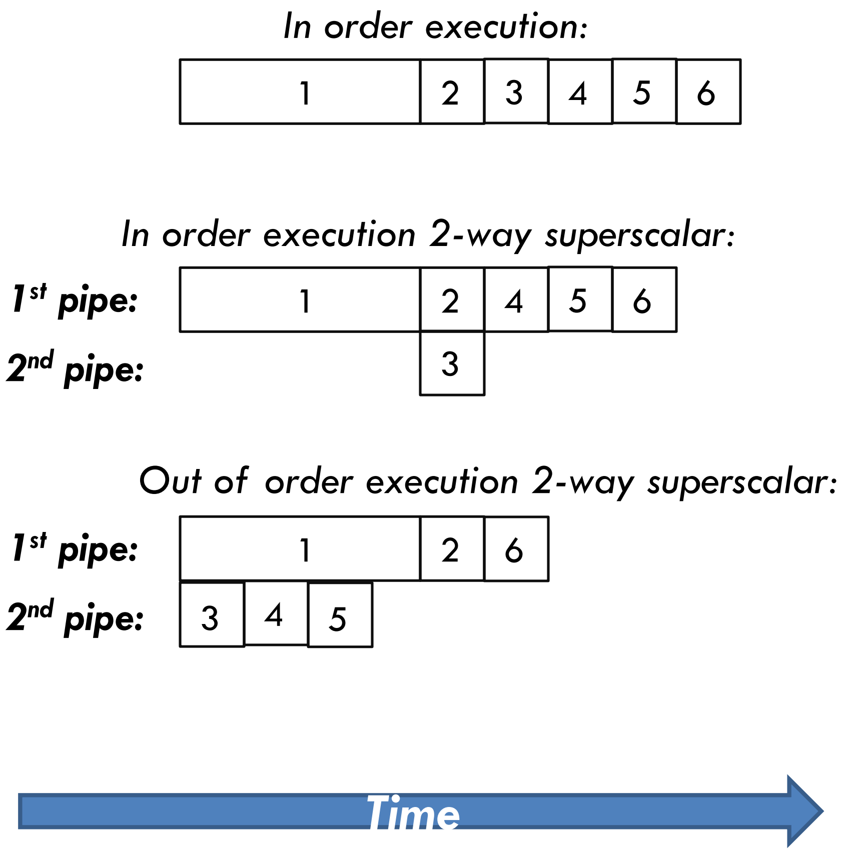
S3: r5=r5+1

S4: r6=r6-r3

S5: r4=r5+r6

S6: r7=r8*r4




In order execution:
1
2
3
4
5
6
In order execution 2-way superscalar:
1st pipe:
1
2
4
5
6
2nd pipe:
3
Out of order execution 2-way superscalar:
1st pipe:
1
2
6
2nd pipe:
3
4
5
Time

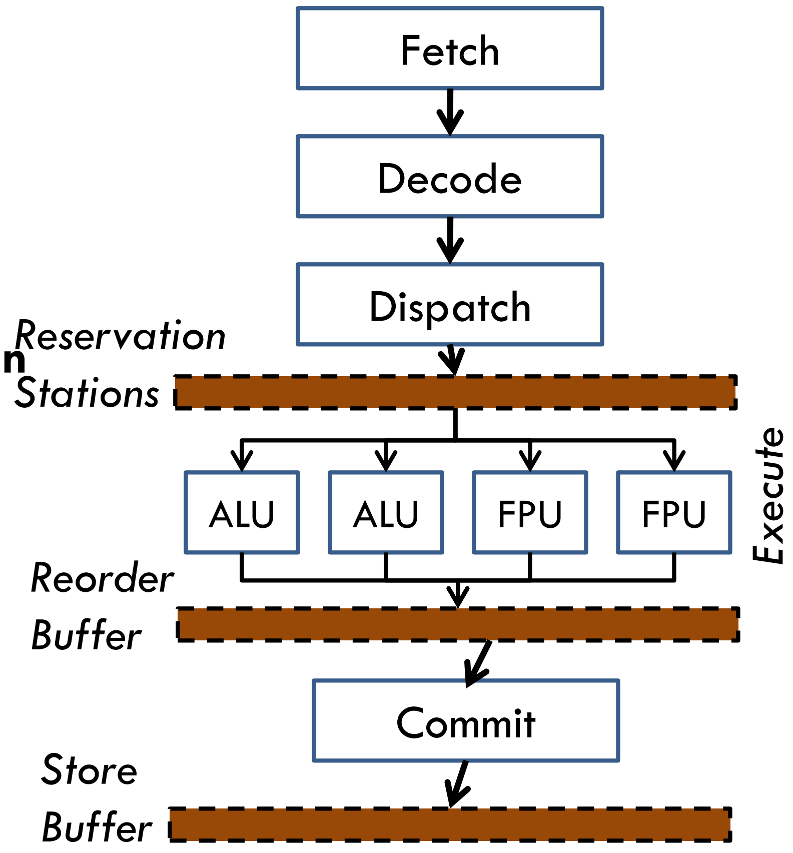
Out of Order Execution (1)

- **Idea: Move the dependent instructions out of the way of independent ones**
  - Rest areas for dependent instructions: **Reservation station**

- **Instruction window:** It is a memory that holds the instructions that have been fetched and decoded and are waiting to be executed
  - Note: Often, the instruction window doesn't actually exist as a single buffer, but is distributed among reservation stations and reorder buffer

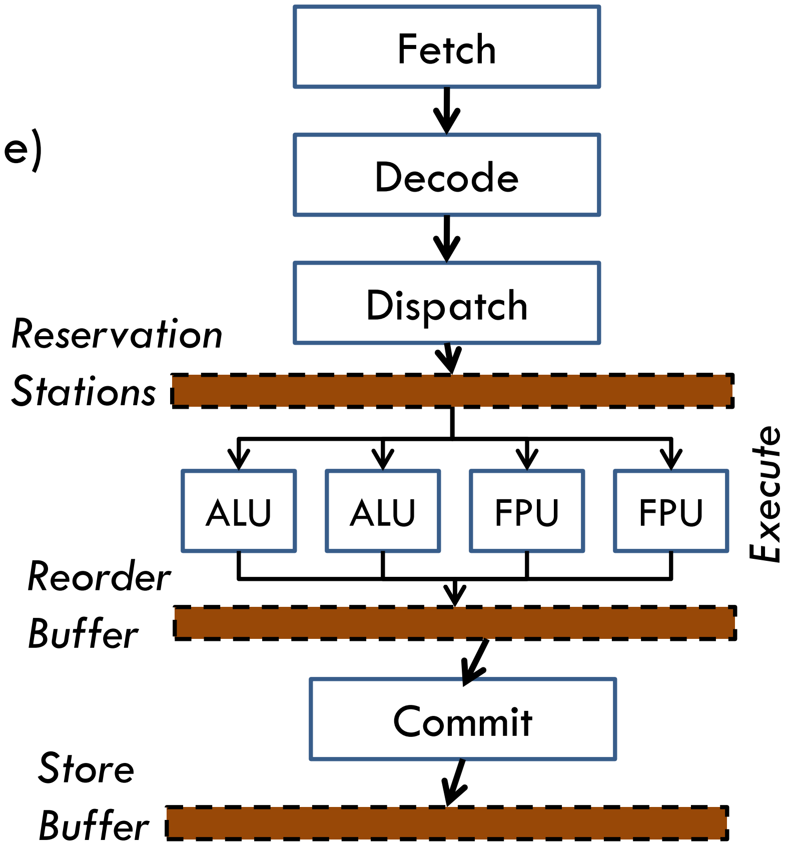
Out of Order Execution (2)

- **Fetch, decode & dispatch instructions in order**
  - Multiple instructions are fetched/decoded/dispatched in parallel
  - Instructions are put in the instruction window - reservation stations (RS)
- **Execute instructions (out of order) that are ready in the reservation stations – speculative execution**
  - Instruction operands must be ready
  - Available execution resources
- After execution:
  - Broadcast result on bypass network
  - Signal all dependent instructions that their data are ready
- Commit instructions (leave the instr. window) **in-order**
  - All execution within the instruction window is speculative (i.e., side-effects are not applied outside the CPU) until it is committed



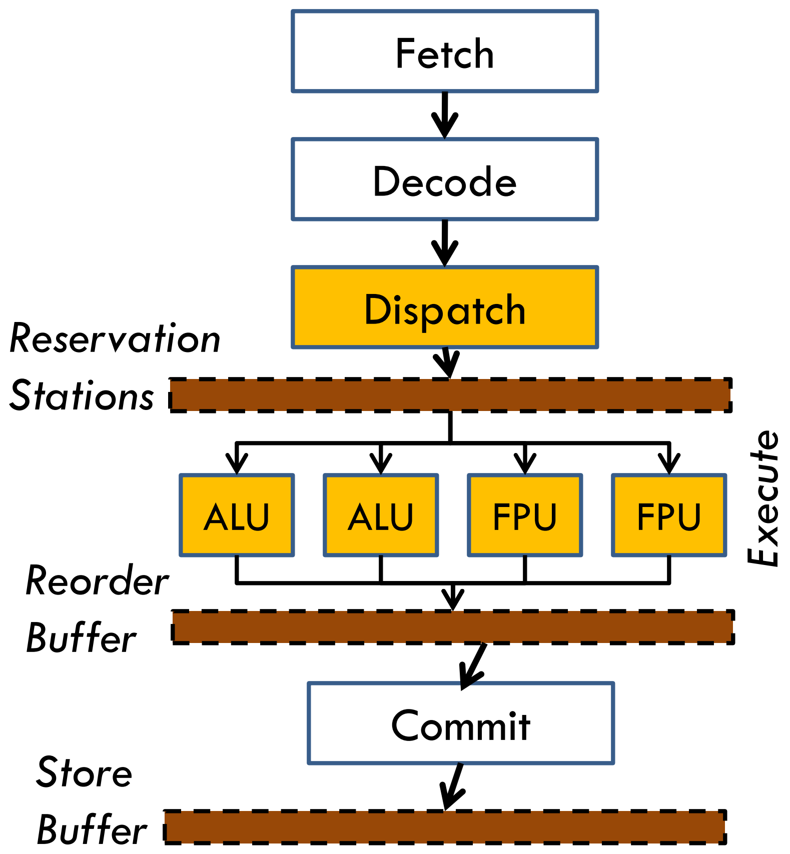
Out of Order Execution (3)

- **Advantages: Better performance!**
  - Exploit Instruction Level Parallelism (ILP)
  - Hide latencies (e.g., L1 data cache miss, divide)
- **Disadvantages:** HW is much more complex than that of in-order processors
  - **More expensive, larger chip area, higher power consumption**
- Can compilers do this work instead?
  - In a very limited way
  - Compilers lack runtime information
    - Conditional branch direction
    - Data values, which may affect calculation time and control
    - Cache miss / hit



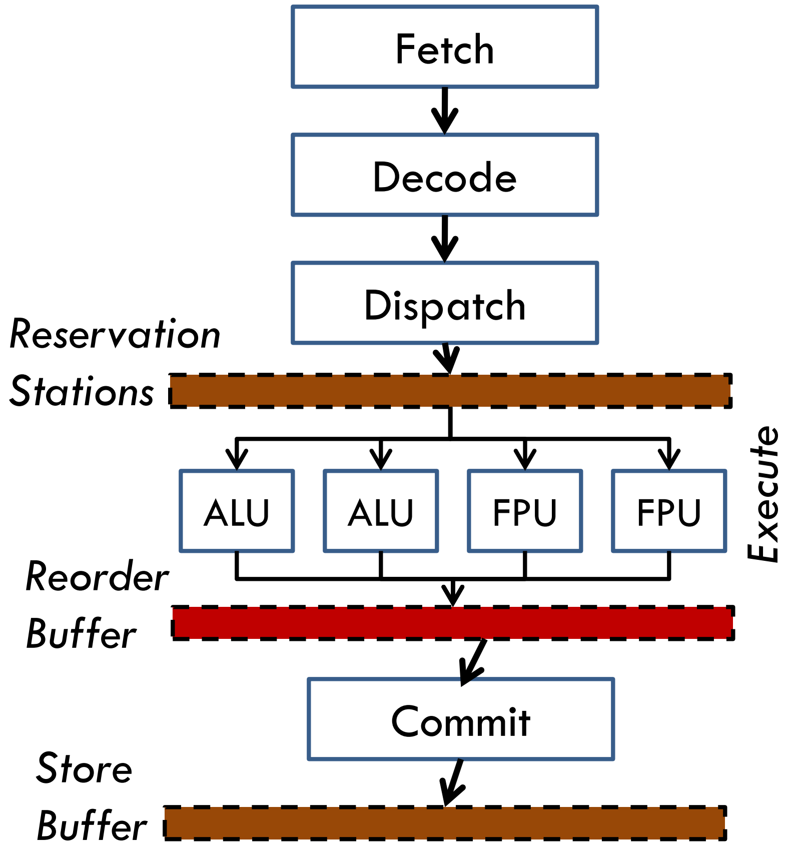
Out of Order Processors – the Pipeline (1)

- **Dispatch** : new instructions are added to the instruction window - reservation stations (RS)
- **Reservation stations (RS)**
  - Instructions wait for their inputs
  - Instructions wait for their functional units
  - If instruction operands are ready they are sent to the FU
  - Otherwise, check on bypass network and wait for operands
- Functional units (FU)
  - ALUs, AGUs, FPUs



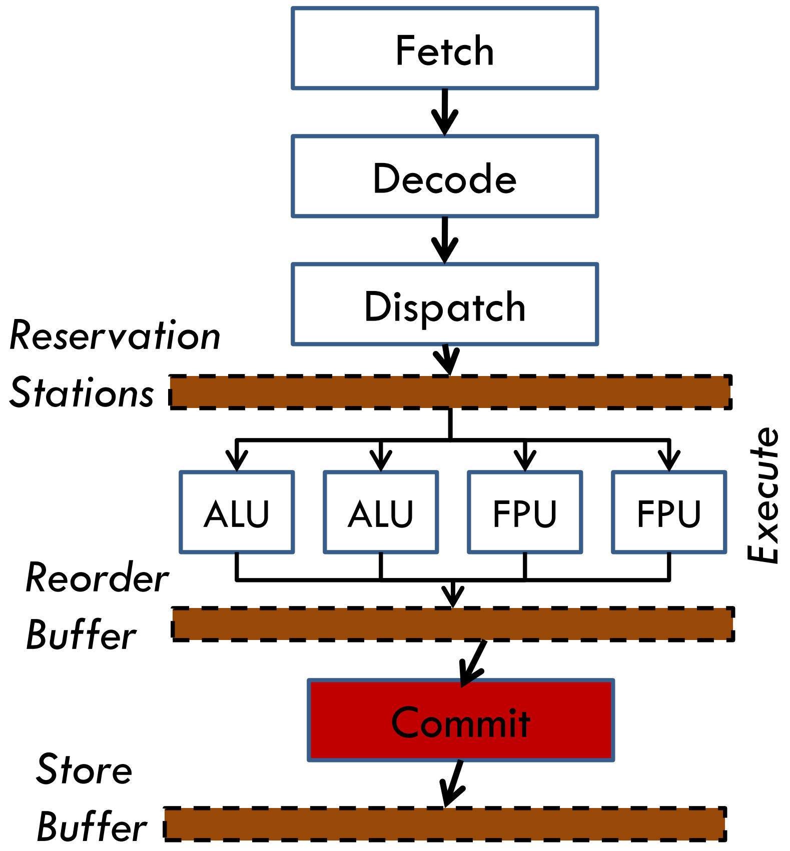
Out of Order Processors – the Pipeline (2)

- **Bypass network**
  - Broadcast computed values back to reservation stations
- **ReOrder buffer (ROB)**
  - It allows instructions to be committed in-order
  - De-speculates execution, mostly by Committing instructions in-order
  - flushes the speculative instructions when a misprediction is discovered



Out of Order Processors – the Pipeline (3)

- **Commit**
  - All execution within the instruction window is speculative (i.e., side-effects are not applied outside the CPU) until it is committed
  - Instructions can write to memory only once it is certain they should have been executed
  - Instructions must not affect machine state while they are speculative
  - **Instructions enter and leave the instruction window in program order,** and an instruction can only leave the window when it is the oldest instruction in the window and it has been completed

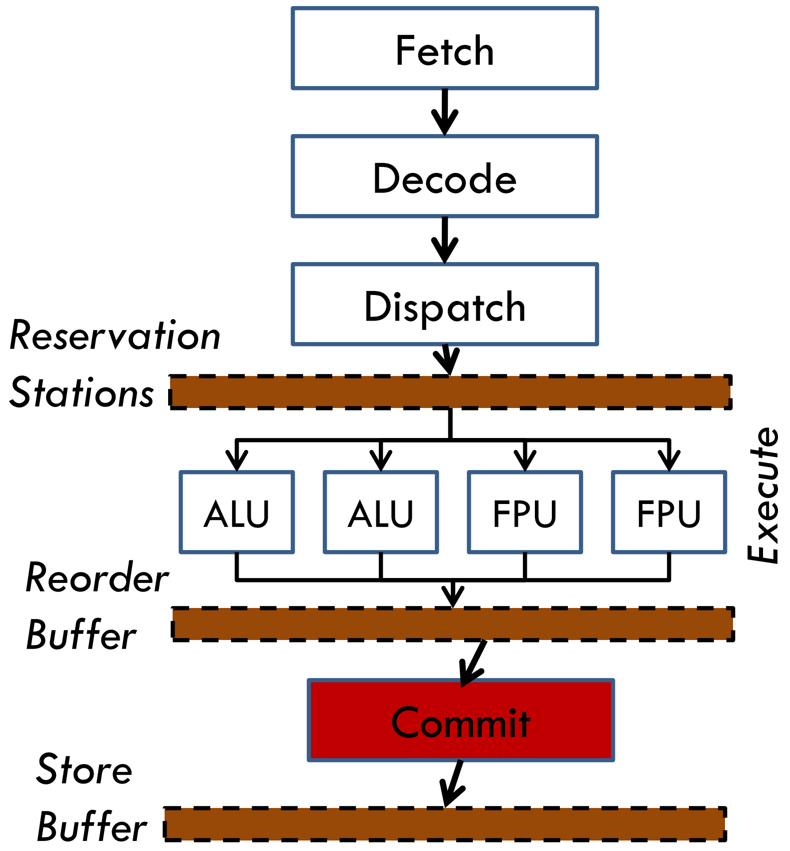


- **The instruction window is instantiated as RS & ROB**

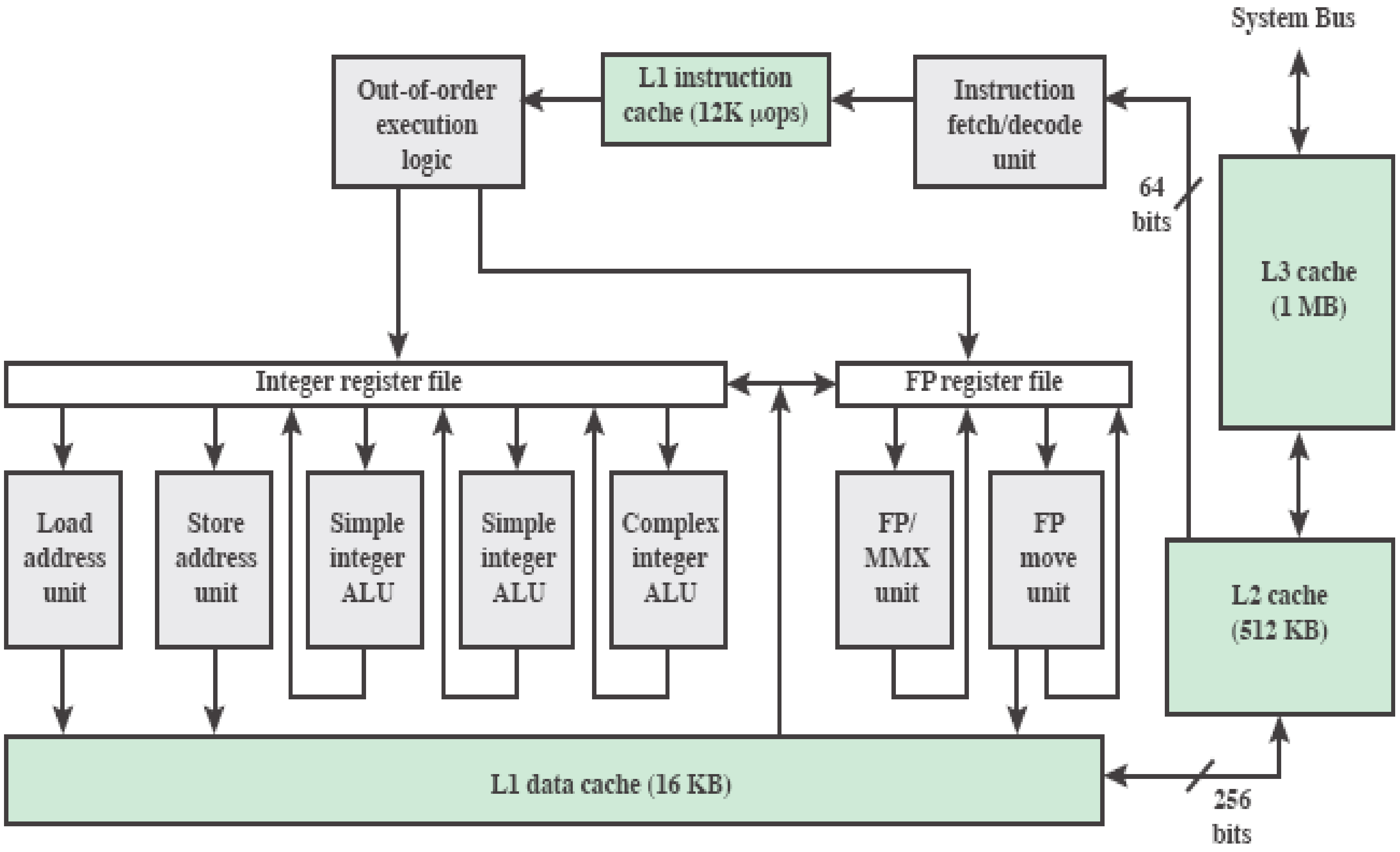
Out of Order Processors – the Pipeline (4)

Store Buffer:

- **Stores are not executed Out of Order (OoO)**
  - Stores are never performed speculatively
  - There is no transparent way to undo them
- Stores are also never re-ordered among themselves
  - The Store Buffer dispatches a store only when
    - The store has both its address and its data ready and
    - There are no older stores awaiting dispatch



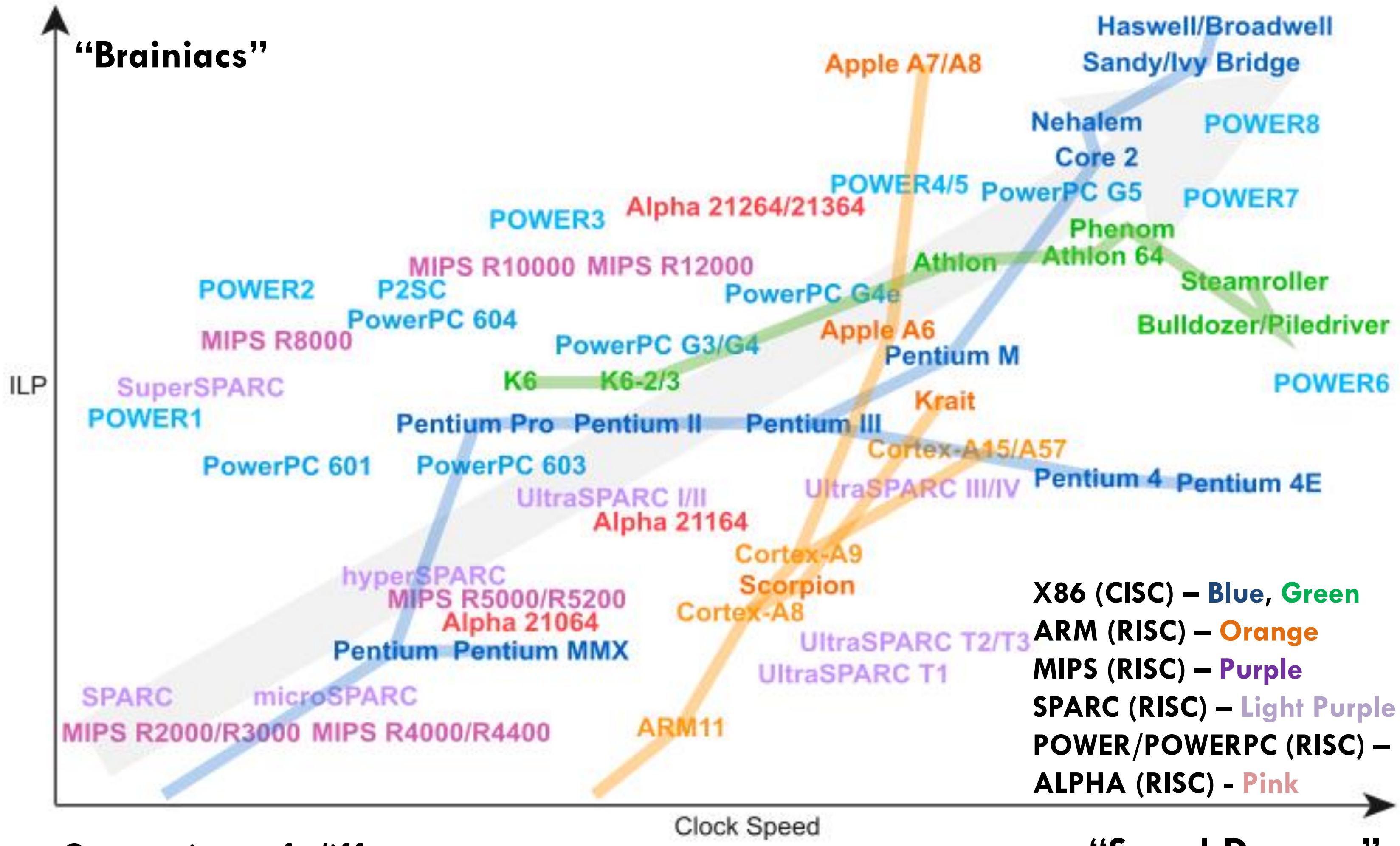
A Superscalar Processor



Out of Order Processors - Summary

- Advantages
  - Help exploit Instruction Level Parallelism (ILP)
  - Help hide latencies (e.g., cache miss, divide)
  - Superior/complementary to instuction Scheduler in the compiler
    - Dynamic instruction window
- Complex micro-architecture - hardware
  - Complex instruction logic
  - Requires reordering mechanism (retire/commit)
  - Misprediction/speculation recovery
- Speculative Execution
  - Advantage: larger scheduling window ⇒ reveals more ILP
  - Issues:
    - Complex logic needed to recover from mis-prediction
    - Runtime cost incurred when recovering from a mis-prediction

Comparison of different processors (Brainiacs vs Speed-Demons) (1)



X86 (CISC) – Blue, Green
ARM (RISC) – Orange
MIPS (RISC) – Purple
SPARC (RISC) – Light Purple
POWER/POWERPC (RISC) – Light Blue
ALPHA (RISC) - Pink

“Speed-Demons”

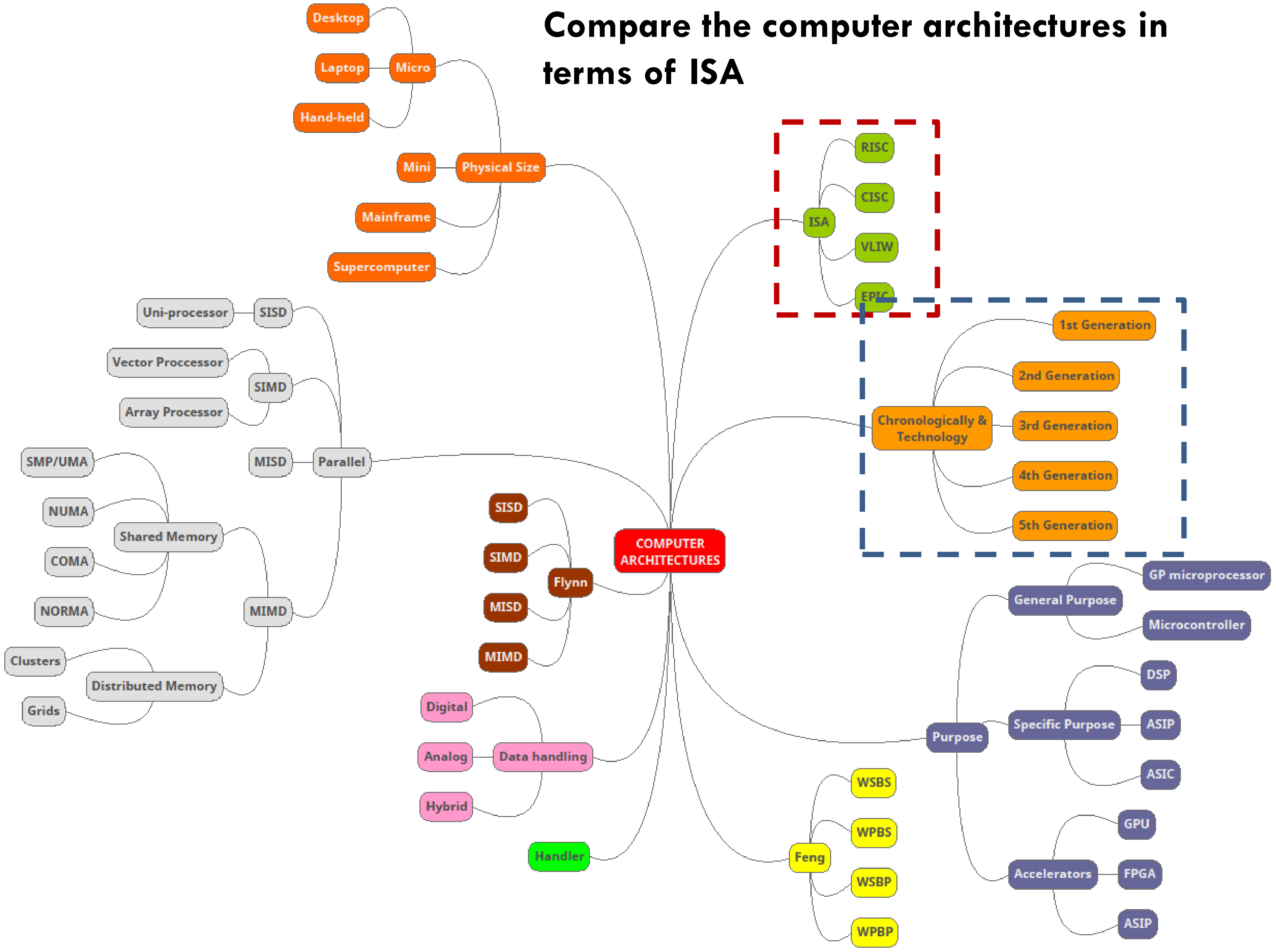
Comparison of different processors (http://www.lighterra.com/papers/modernmicroprocessors/)

Comparison of different processors (Brainiacs vs Speed-Demons) (2)

- "Brainiac designs"
  - Extra HW in order to achieve more Instruction Level Parallelism (ILP) out of the code
  - Millions of extra transistors
  - More design effort
  - Consume more power
- "Speed-Demons"
  - Run at higher clock speeds because they are simpler
  - Simple HW design
  - Less Chip area
  - Less power consumption

Which would you rather have: 4 powerful brainiac cores, or 8 simpler in-order cores? they use the same chip area

Compare the computer architectures in terms of ISA



What is ISA (Instruction Set Architecture)

- It provides commands to the processor, to tell it what to do, e.g., add, load, store
- ISA is analogous to a human language
- Allows communication
  - Human language: person to person
  - ISA: software to hardware
- Need to speak the same language/ISA
- Many common aspects
  - Part of speech: verbs, nouns, adjectives, adverbs, etc.
  - Common operations: calculation, control/branch, memory
- Different computer processors may use almost the same instruction set

RISC vs CISC (1)

- Complex instruction set computer (**CISC**): complex instructions can execute several low-level operations (such as a load from memory, an arithmetic operation, and a memory store)
  - CISC puts emphasis on HW
  - CISC was developed to make compiler development simpler
  - CISC is typically used for **general purpose computers**
- Reduced instruction set computer (**RISC**): simple and 1 cycle instructions
  - RISC puts emphasis on SW
  - RISC was developed to make HW simpler
  - RISC is typically used for **smart phones, tablets and other embedded devices**

RISC vs CISC architecture comparison

CISC

1. Instructions take varying amount of cycle time (multiple cycles)
2. Instructions provide complex operations
3. Many different instructions
4. Less registers
5. Pipeline is difficult
6. Different length instructions
7. The Opcode has no fixed position and size

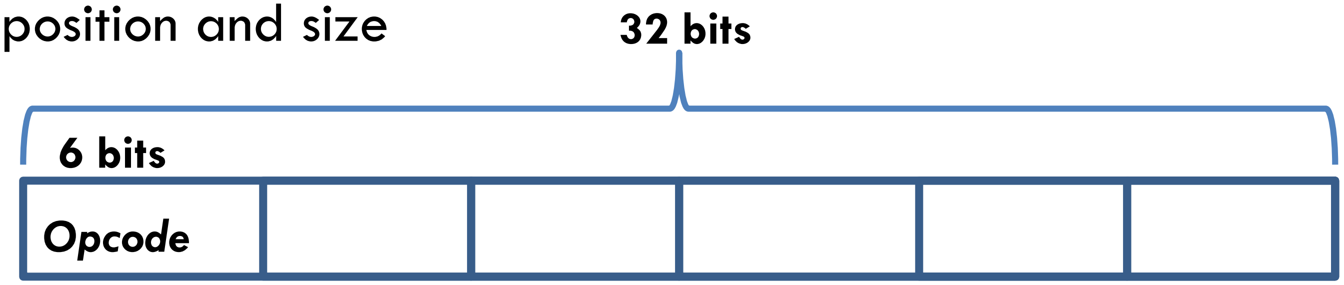
RISC

1. 1 cycle instruction
2. Simple operations
3. Few different instructions
4. More registers
5. Pipeline is easy
6. All instructions have the same length(4 bytes)
7. The Opcode has a fixed position and size within the instruction

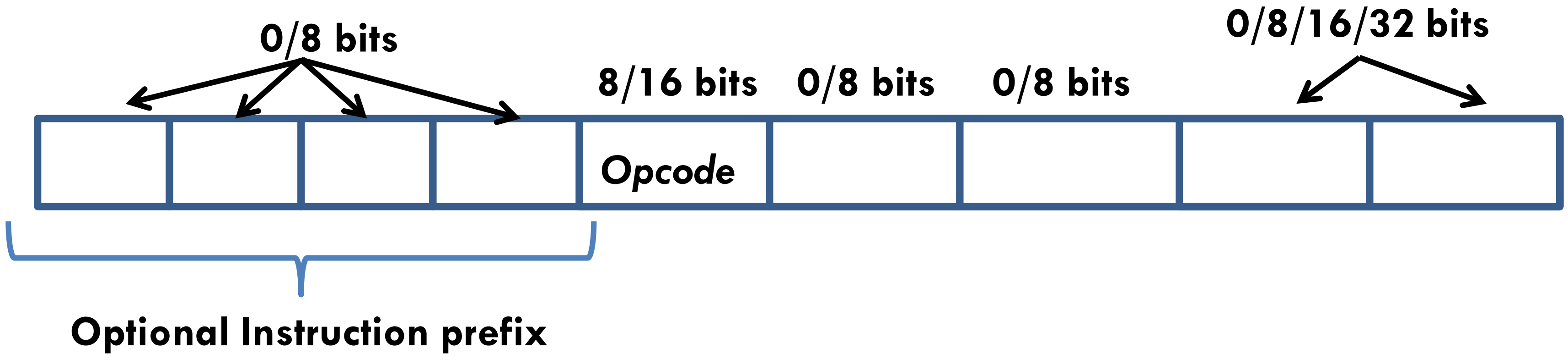
RISC vs CISC (2)

- Opcode specifies the operation to be performed
  - RISC – fixed position and size
  - CISC – no fixed position and size

RISC instruction format:



CISC instruction format:



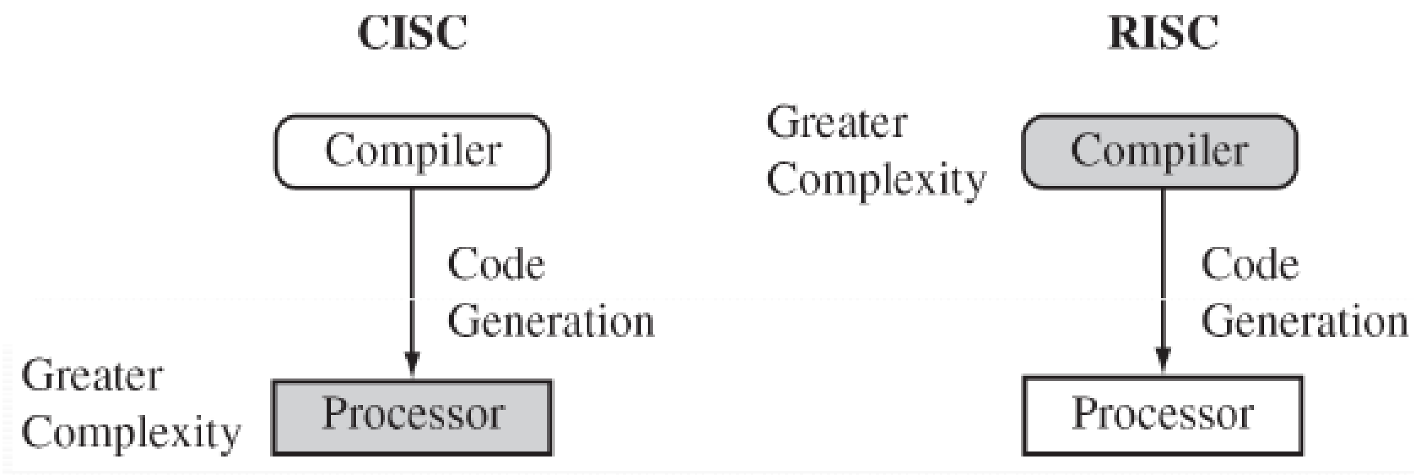
RISC vs CISC - Pros & Cons

CISC:

- Emphasis on HW
- Multi-clock complex instructions
- Less "instructions/program" with "complex" instructions
- Easy for assembly-level programmers, good code density
- Easy for compiler writers support more complex higher-level languages

RISC:

- Emphasis on SW
- Single-clock simple instructions
- Less "cycles/instruction" with single-cycle instructions
- Faster design and implementation - RISC takes about 1/5 the design time
- The performance of the RISC processors highly depends mostly on the compiler or programmer



RISC vs CISC - Conclusions

- **Nowadays, the two architectures almost seem to have adopted the strategies of the other**
- **CISC and RISC implementations are becoming more and more alike**
  - Today's RISC chips
    - support as many instructions as yesterday's CISC chips
    - incorporate more complicated, CISC-like commands
    - make use of more complicated hardware, making use of extra function units for superscalar execution
  - Today's CISC chips
    - use many techniques formerly associated with RISC chips
    - are now able to execute more than one instructions within a single clock

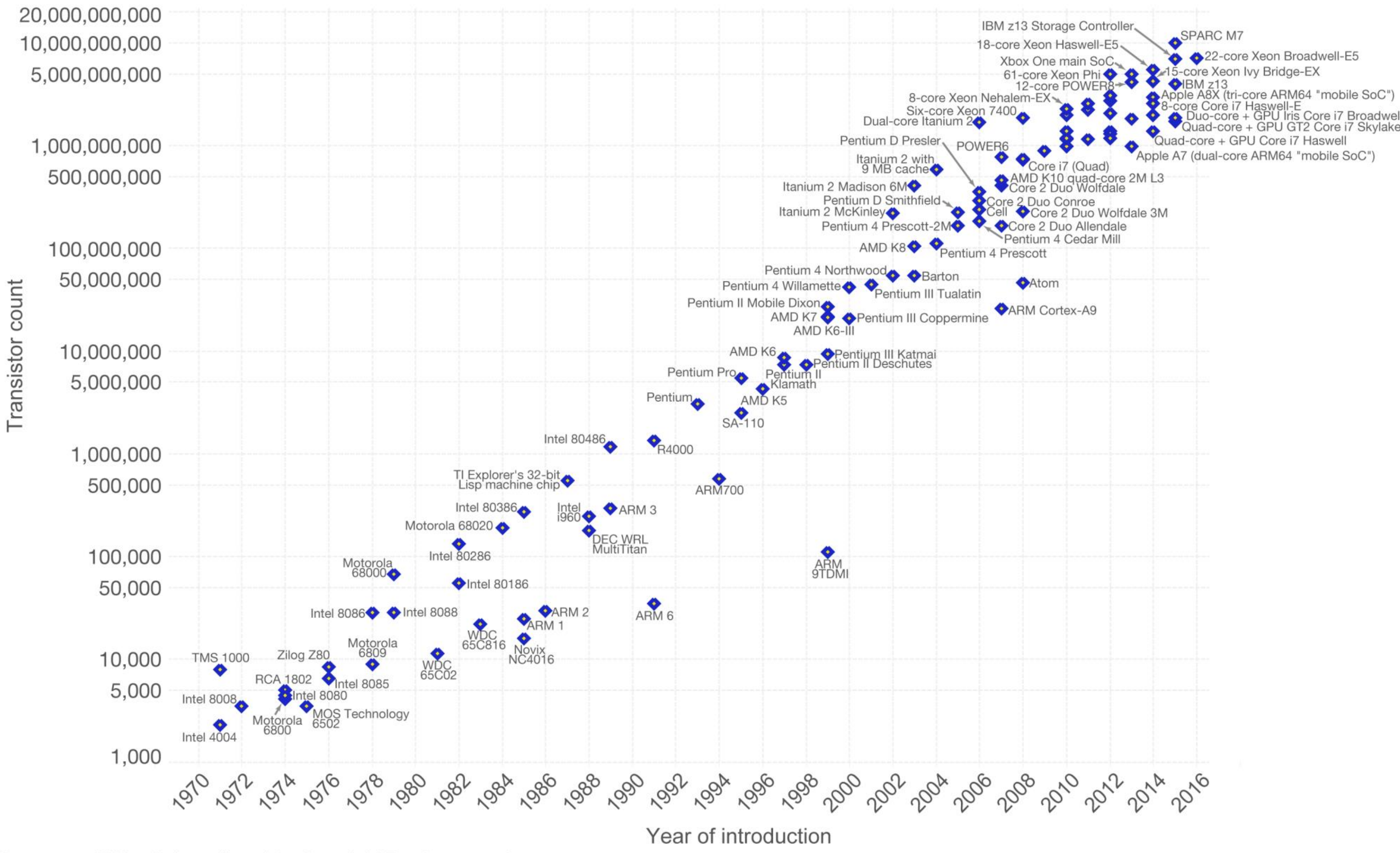
Think-Pair-Share 2nd Exercise

Question: What is in your opinion the best Instruction Set Architecture (ISA) and why?

Answer: There is no good and bad ISA, but appropriate and not appropriate. The appropriate ISA depends on the target goals and on the target application

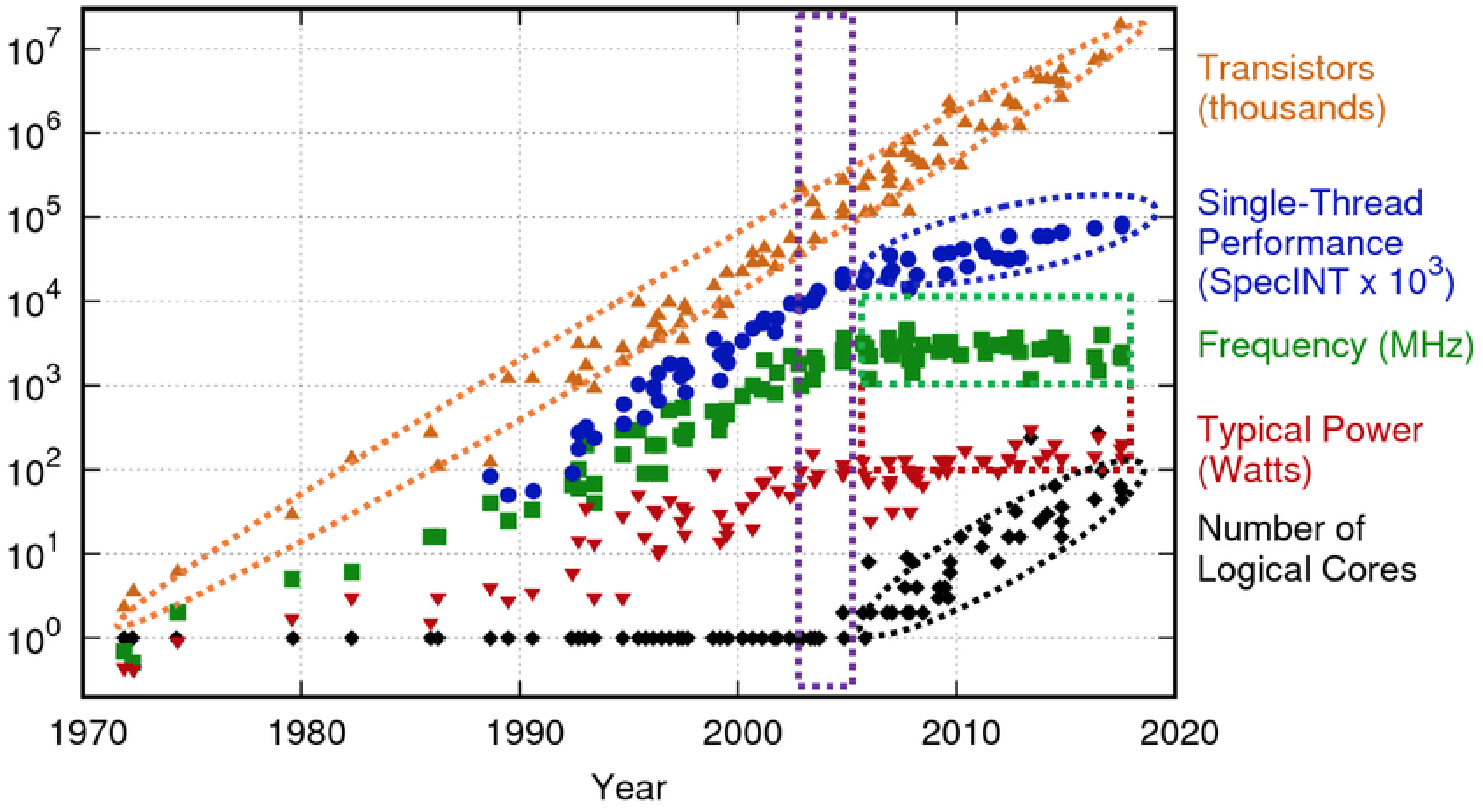
Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Hardware Architecture Trends



Taken from https://www.researchgate.net/publication/336577121_BACKUS_Comprehensive_High-Performance_Research_Software_Engineering_Approach_for_Simulations_in_Supercomputing_Systems

Conclusions

- Computer architectures are complex and diverse
- There is a large number of different computer architecture classifications
- Classification helps us to select the most appropriate architecture
- Computer architectures are evolving year by year
- The computer architecture requirements are continually increasing
- There is no good or bad computer architecture. It depends on the
  - ✓ target goals, e.g., performance, flexibility
  - ✓ target application

Further Reading

Structured Computer Organization. Sixth Edition, Andrew S. Tanenbaum, Todd Austin, PEARSON, 2012, https://universalflowuniversity.com/Books/Computer%20Programming/Computers%2C%20Architecture%20and%20Design/Structured%20Computer%20Organization%206th%20Edition.pdf

Computer Organization & Architecture. Designing for Performance. William Stallings, Seventh Edition, available at http://home.ustc.edu.cn/~louwenqi/reference_books_tools/Computer%20Organization%20and%20Architecture%2010th%20-louwenqi/reference_books_tools/Computer%20Organization%20and%20Architecture%2010th%20-%20William%20Stallings.pdf

Nicholas FitzRoy-Dale, The VLIW and EPIC processor architectures, available at https://www.cse.unsw.edu.au/~cs9244/06/seminars/02-nfd.pdf

Thank you