



# Systems Thinking

Seamless Transition-2:

Amdahl's Law, Computing Systems Landscape

阿姆达尔定律，形形色色的计算系统

zxu@ict.ac.cn

zhangjialin@ict.ac.cn

# Outline

- What is systems thinking?
- Three objectives of systems thinking
- Abstraction
- Modularization
- Seamless transition
  - The symphony of four principles
  - Yang's cycle principle
  - Postel's robustness principle
  - von Neumann's exhaustiveness principle
  - Amdahl's law
    - Pipeline
    - Cache
    - Parallelism
  - Landscape of computing systems

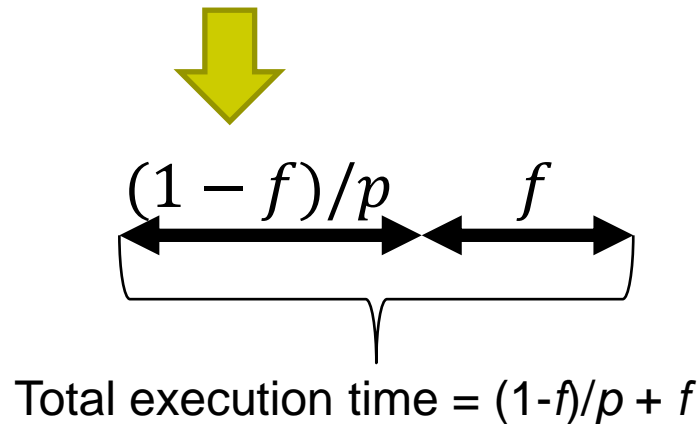
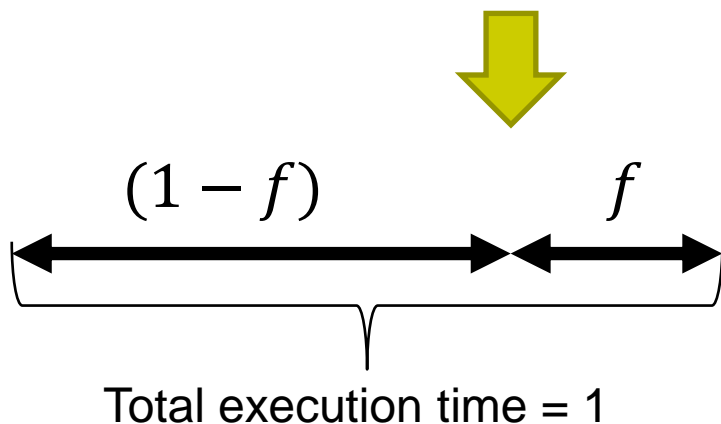
*These slides acknowledge sources for additional data not cited in the textbook*

## 5.5 Amdahl's law

- Previous 3 principles regard correctness (seamless transition)
- Amdahl's law focuses on performance (smooth transition)
- 1-minute quiz
  - A student writes a program `sort.go` to read an 8-GB file of 64-bit integers, sort the integers, and write the sorted result into another file on his laptop computer. Suppose the total execution time is  $T$ , where  $0.8 \cdot T$  is spent on I/O operations.
    - Q1: What is the problem size  $N$ , i.e., the number of integers?
    - A1: (a) 8 (b) 64 (c) 1024 (d)  $64 \cdot 1024 \cdot 1024$  (e)  $1024 \cdot 1024 \cdot 1024$
    - Q2: The student upgrades his laptop with a new wonder processor that is 100 times as fast as the old processor. How much time (approximately) is needed to execute `sort.go` on the same 8-GB file?
    - A1: (a)  $T/100$  (b)  $T/5$  (c)  $0.1 \cdot T$  (d)  $0.8 \cdot T$

# Amdahl's law 阿姆达尔定律

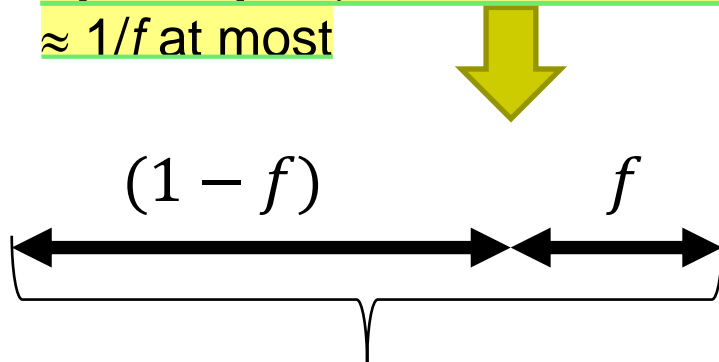
- After enhancing a portion of a system, the speedup obtained is upper bounded by the reciprocal of the other portion's time.  
改进系统的一部分后，系统加速比不会超过未改进部分执行时间的倒数
- Suppose a system's execution time is broken into two portions  $(1 - f)$  and  $f$ , such that  $(1 - f) > f$  系统执行时间1划分成两部分 $(1 - f)$ ,  $f$
- Amdahl's law: Enhancement on the  $(1 - f)$  portion can lead to a speedup no more than  $1/f$ 
  - Speedup approaches but never exceeds  $1/f$
  - Speedup** = (time before enhancement) / (time after enhancement)  
加速比 = 改进前的执行时间 / 改进后的执行时间



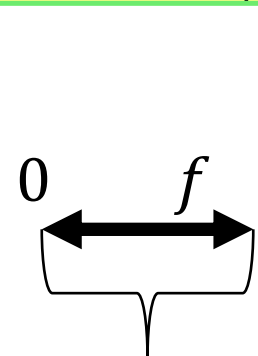
# Amdahl's law

- After enhancing a portion of a system, the speedup obtained is upper bounded by the reciprocal of the other portion's time.
- Suppose a system's execution time is broken into two portions  $(1 - f)$  and  $f$ , such that  $(1 - f) > f$
- Amdahl's law: Enhancement on the  $(1 - f)$  portion can lead to a speedup no more than  $1/f$ 
  - Speedup approaches but never exceeds  $1/f$

- **Speedup** = (time before enhancement) / (time after enhancement)  
 $\approx 1/f$  at most



Total execution time = 1



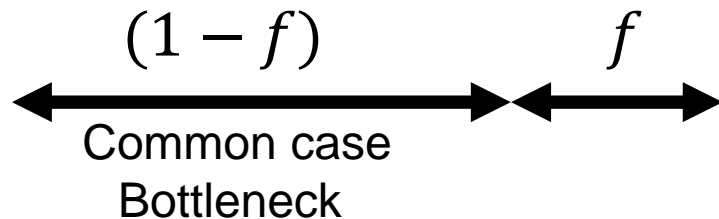
Total execution time  $\rightarrow 0 + f$ , as  $p \rightarrow \infty$

## 5.5 Amdahl's law

- 1-minute quiz
  - A student writes a program sort.go to read an 8-GB file of 64-bit integers, sort the integers, and write the sorted result into another file on his laptop computer. Suppose the total execution time is  $T$ , where  $0.8 \cdot T$  is spent on file I/O operations.
  - Q1: What is the problem size  $N$ , i.e., the number of integers?
  - A1: (a) 8 (b) 64 (c) 1024 (d)  $64 \cdot 1024 \cdot 1024$  (e)  $1024 \cdot 1024 \cdot 1024$
  - Q2: The student upgrades his laptop with a new wonder processor that is 100 times as fast as the old processor. How much time (approximately) is needed to execute sort.go on the same 8-GB file?
  - A1: (a)  $T/100$  (b)  $T/5$  (c)  $0.1 \cdot T$  (d)  $0.8 \cdot T$ 
    - The processing time is about  $0.2 \cdot T/100$ , but the I/O time stays the same. Thus, the total time is about  $0.2 \cdot T/100 + 0.8 \cdot T = 0.802 \cdot T$
    - The speedup is  $T / 0.802T \approx 1.247$ , only 24.7% faster

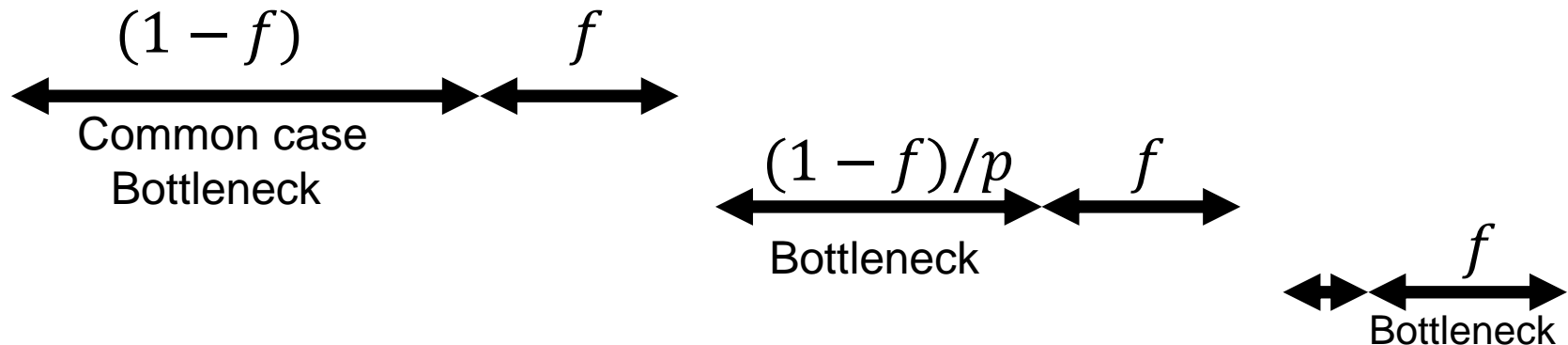
# Implications of Amdahl's law

- Amdahl's law offers two advices for system design
  - *Optimize the common case.* System enhancement, or system optimization, should focus on the common case, also known as the bottleneck, i.e., the most time consuming portion



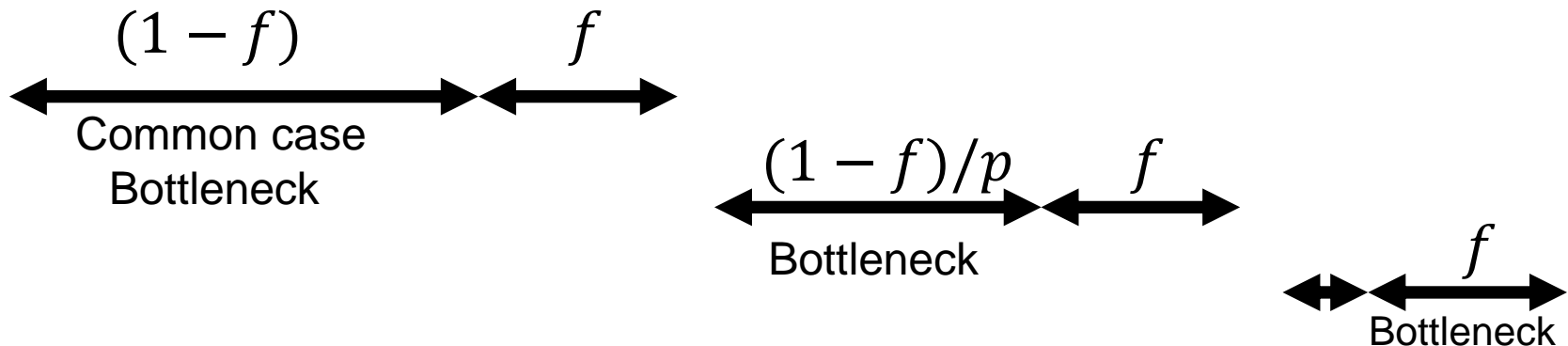
# Implications of Amdahl's law

- Amdahl's law offers two advices for system design
  - *Optimize the common case.* System enhancement, or system optimization, should focus on the common case, also known as the bottleneck, i.e., the most time consuming portion
  - *Chase the bottleneck.* When the system bottleneck changes, so does our optimization target



# Implications of Amdahl's law

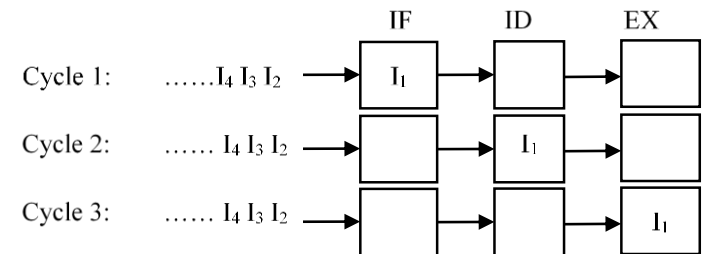
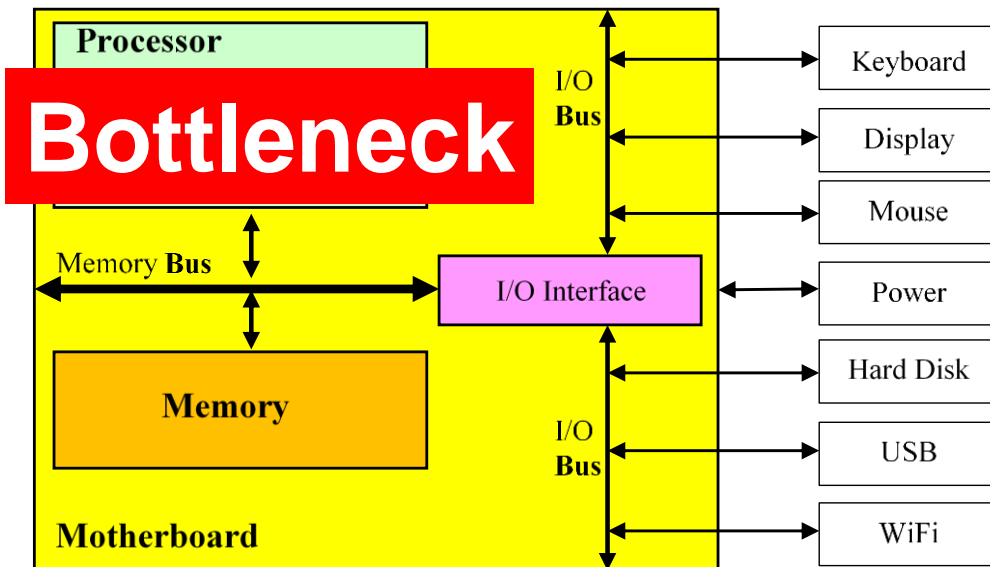
- Amdahl's law offers two advices for system design
  - *Optimize the common case*. System enhancement, or system optimization, should focus on the common case, also known as the bottleneck, i.e., the most time-consuming portion 改进瓶颈
  - *Chase the bottleneck*. When the system bottleneck changes, so does our optimization target 追逐瓶颈



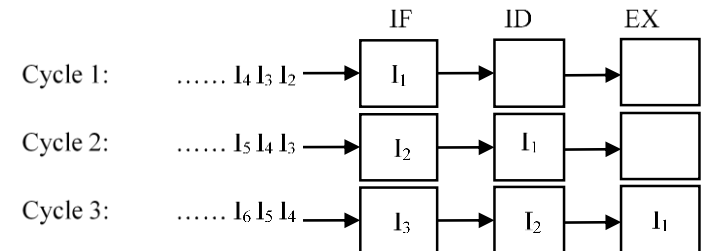
- Three techniques to utilize Amdahl's law
  - Pipelining, caching, parallel computing

## 5.5.1 Pipelining: multiple instructions overlap

- Optimize the bottleneck of the CPU's instruction pipeline
  - Key technique: overlapping pipeline stages with multiple instructions
    - Using about the same amount of resource, plus some overhead
- Assuming 1-GHz clock cycle, the average performances are
  - Without overlapping: executing an instruction needs 3 cycles and 3 ns
  - With overlapping: executing an instruction needs 1 cycle and 1 ns



(a) No overlapping of pipeline stages



(b) Overlapping of pipeline stages

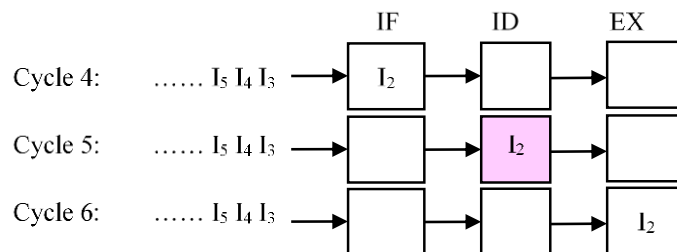
# 1-minute quiz

- Q: Continue to draw the instruction pipeline configurations for clock cycles 4, 5, 6. Which of the following configurations is correct? (Assume the ideal case)

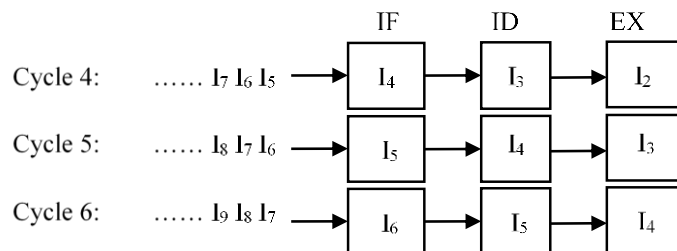
- The colored box indicates that instruction  $I_2$  is executing the Instruction Decode (ID) stage at clock cycle 5

- A:

## Configuration 1

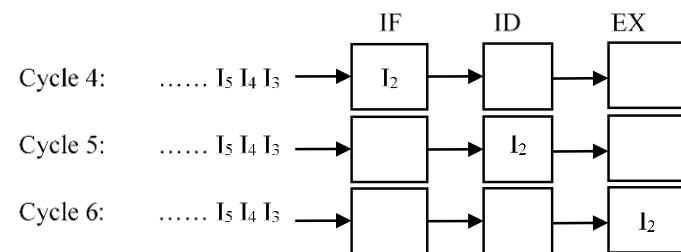


(a) No overlapping of pipeline stages

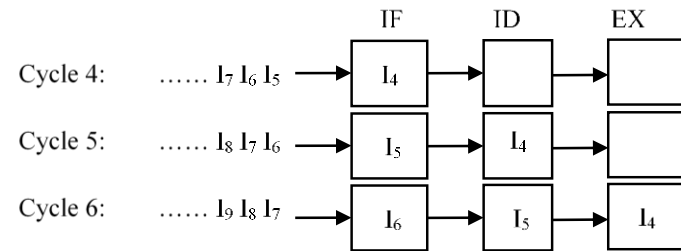


(b) Overlapping of pipeline stages

## Configuration 2



(a) No overlapping of pipeline stages



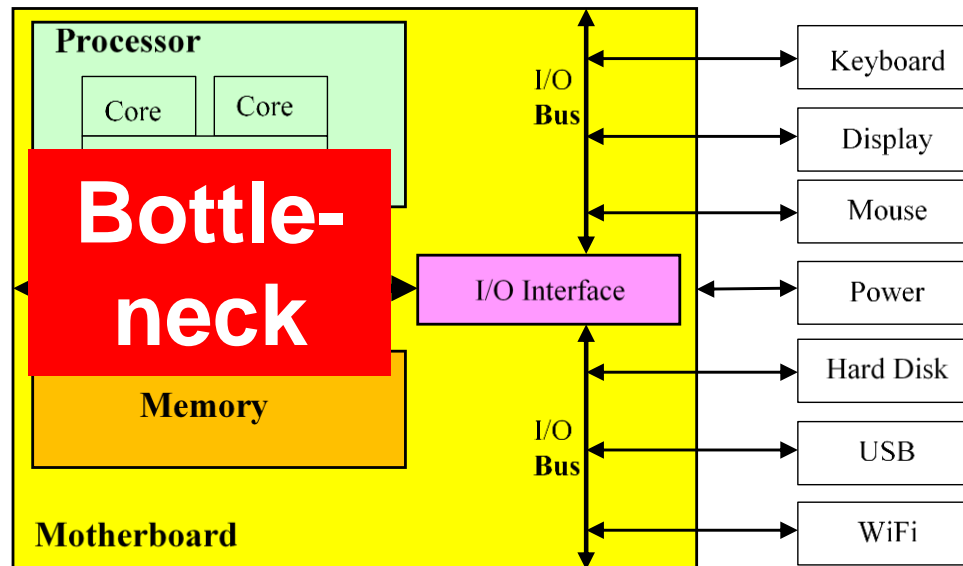
(b) Overlapping of pipeline stages

# 1-minute quiz

- Two computers X and Y have the same pipelined CPU with the same clock frequency of 3.07 GHz.
  - Q1: Do the two computers show the same speed when executing the same code (of Fibonacci loop)?
  - Q2: If computer X is 30 times faster than computer Y, what may be a reason?

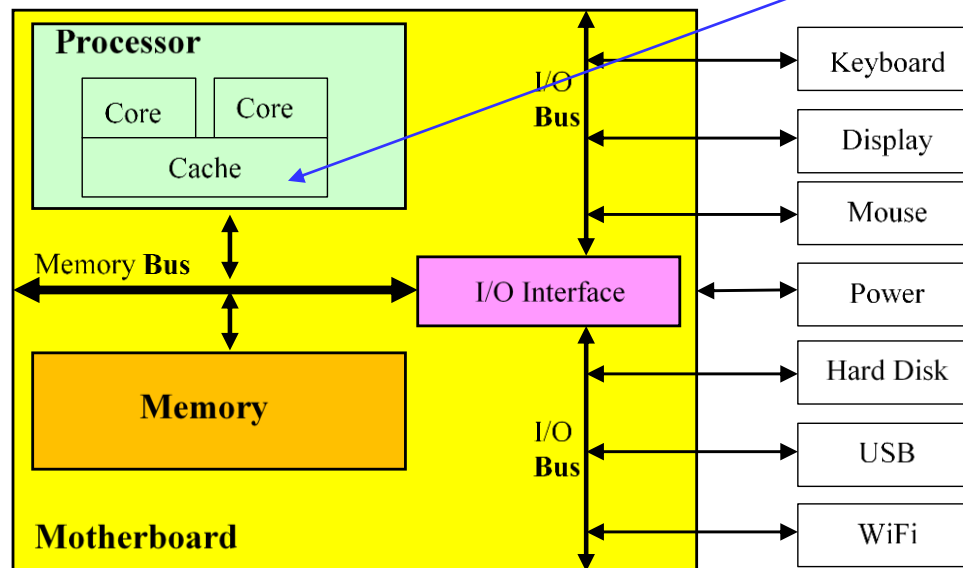
## 5.5.2 Caching: use a small but faster buffer

- Optimize the von Neumann bottleneck
  - Memory is too slow to feed data to CPU
  - On the other hand, programs have spatial and temporal **localities**
  - Key technique: caching, i.e., using a small but faster buffer



## 5.5.2 Caching: use a small but faster buffer

- Optimize the von Neumann bottleneck
  - Memory is too slow to feed data to CPU
  - On the other hand, programs have spatial and temporal localities
  - Key technique: caching, i.e., using a small but faster **buffer**
    - Without having to change the instruction set



- Design and back-of-envelop performance analysis

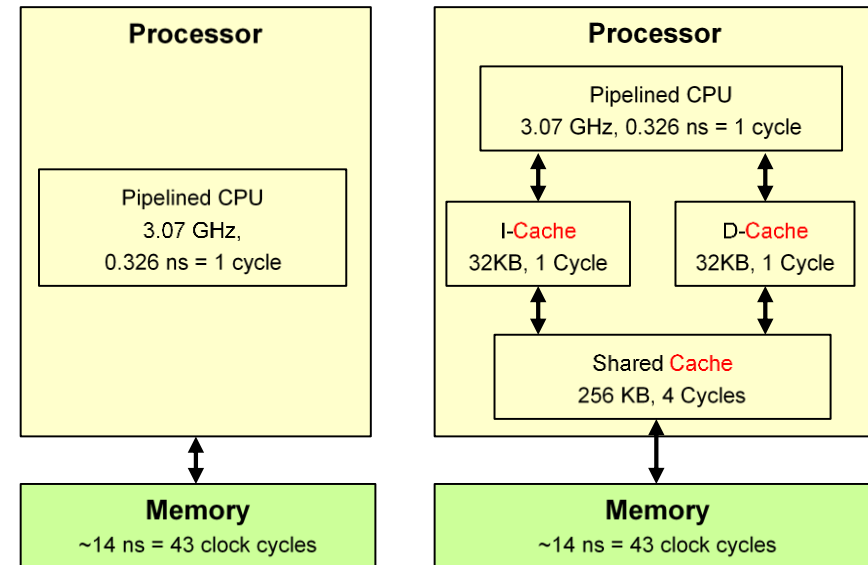
# \*\*\*Caching using the Fibonacci Computer

- Two-layer caching
  - Layer 1: separate instruction and data caches
  - Layer 2: shared cache for instruction and data
- Assume 92 iterations of the loop code in red are executed

Clock cycles of instructions without caching

| Instruction type | Times of memory accesses | Clock cycles needed             |
|------------------|--------------------------|---------------------------------|
| MOV 0, R1        | 1                        | $1 \cdot 43 + 1 \rightarrow 43$ |
| MOV R1, M[...]   | 2                        | $2 \cdot 43 + 1 \rightarrow 86$ |
| ADD M[...], R1   | 2                        | $2 \cdot 43 + 1 \rightarrow 86$ |
| INC R2           | 1                        | $1 \cdot 43 + 1 \rightarrow 43$ |
| CMP 51, R2       | 1                        | $1 \cdot 43 + 1 \rightarrow 43$ |
| JL Loop          | 1                        | $1 \cdot 43 + 1 \rightarrow 43$ |

Numbers in **blue** are canceled out due to pipeline overlapping. A simpler analysis does not consider such optimizations, and calculate the cycles for “MOV 0, R1” to be  $1 \cdot 43 + 1 = 44$ . This simplification leads to similar performance analysis results.

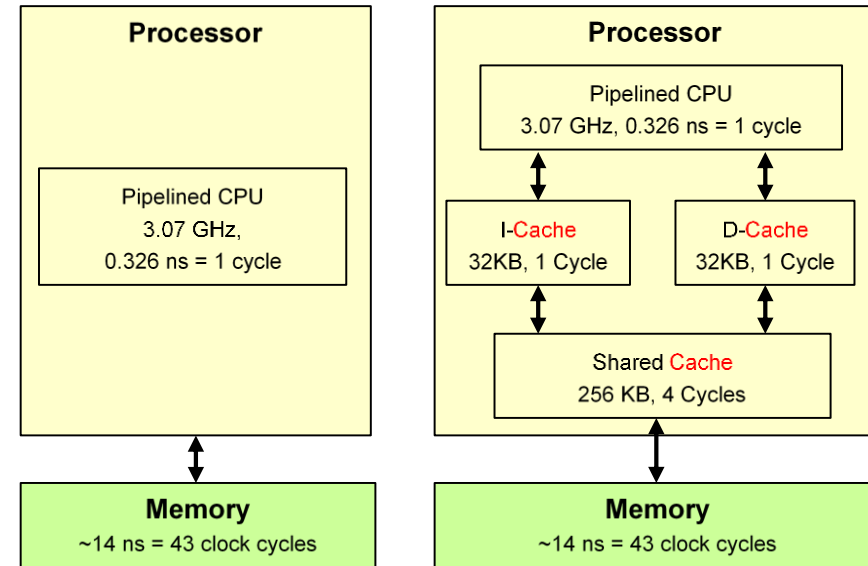


|                       | No Caching | Caching |
|-----------------------|------------|---------|
| MOV 0, R1             | 43         |         |
| MOV R1, M[R0]         | 86         |         |
| MOV 1, R1             | 43         |         |
| MOV R1, M[R0+8]       | 86         |         |
| MOV 2, R2             | 43         |         |
| Loop: MOV 0, R1       | 43         | 1       |
| ADD M[R0+R2*8-16], R1 | 86         | 1       |
| ADD M[R0+R2*8-8], R1  | 86         | 2       |
| MOV R1, M[R0+R2*8-0]  | 86         | 2       |
| INC R2                | 43         | 1       |
| CMP 51, R2            | 43         | 2       |
| JL Loop               | 43         | 3       |

Assume all internal operations together take 1 cycle. Every instruction execution needs to access memory at least once to fetch instruction. Some instructions need two accesses to also fetch data.

# \*\*\*When code and data are stored in level-1 caches

- Instruction Fetch needs only 1 cycle
  - This is why 1<sup>st</sup> MOV needs 1 cycle
- Instruction Fetch and Operand Fetch can be done simultaneously, thanks to Harvard architecture
  - This is why 1<sup>st</sup> ADD needs 1 cycle
- Data and control **dependencies** may cause pipeline to stall (wait)
  - This is why 2<sup>nd</sup> ADD needs 2 cycles and JL needs 3 cycles



## Clock cycles of instructions **with caching**

| Instruction type | Times of memory accesses | Clock cycles needed |
|------------------|--------------------------|---------------------|
| MOV 0, R1        | 1                        | 1*1 +1 → 1          |
| MOV R1, M[...]   | 2                        | 1*1 +1 → 1          |
| ADD M[...], R1   | 2                        | 1*1 +1 → 1          |
| INC R2           | 1                        | 1*1 +1 → 1          |
| CMP 51, R2       | 1                        | 1*1 +1 +1 → 2       |
| JL Loop          | 1                        | 1*1 +1 +2 → 3       |

|                       | No Caching | Caching |
|-----------------------|------------|---------|
| MOV 0, R1             | 43         |         |
| MOV R1, M[R0]         | 86         |         |
| MOV 1, R1             | 43         |         |
| MOV R1, M[R0+8]       | 86         |         |
| MOV 2, R2             | 43         |         |
| Loop: MOV 0, R1       | 43         | 1       |
| ADD M[R0+R2*8-16], R1 | 86         | 1       |
| ADD M[R0+R2*8-8], R1  | 86         | 2       |
| MOV R1, M[R0+R2*8-0]  | 86         | 2       |
| INC R2                | 43         | 1       |
| CMP 51, R2            | 43         | 2       |
| JL Loop               | 43         | 3       |

Assume all internal operations together take 1 cycle.  
Every instruction execution needs to access memory at least once to fetch instruction. Some instructions need two accesses to also fetch data.

# Simplified performance analysis

- Simplify by consider a single iteration of loop

- Without caching

- Total time =  $4 \times 43 + 3 \times 86$   
= 430 clock cycles  
=  $430 \times 0.326 = 140$  ns

|                       | No Caching | Caching |
|-----------------------|------------|---------|
| Loop: MOV 0, R1       | 43         | 1       |
| ADD M[R0+R2*8-16], R1 | 86         | 1       |
| ADD M[R0+R2*8-8], R1  | 86         | 2       |
| MOV R1, M[R0+R2*8-0]  | 86         | 2       |
| INC R2                | 43         | 1       |
| CMP 51, R2            | 43         | 2       |
| JL Loop               | 43         | 3       |

- With caching

- Total time =  $1 \times 3 + 2 \times 3 + 3 \times 1$   
= 12 clock cycles  
=  $12 \times 0.326 = 3.91$  ns

- Caching brings about significant speedup

- Speedup =  $140 \text{ ns} / 3.91 \text{ ns} = 35.8$ , or 34.8 times faster

# Performance metrics of the 3.07-GHz FC

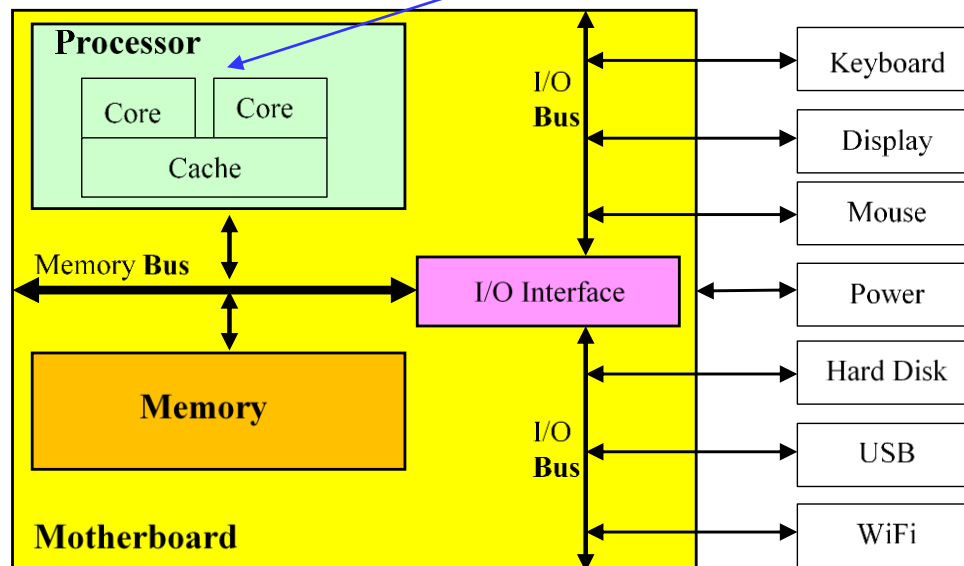
- The **peak speed** of the Fibonacci Computer is the maximal speed possible
  - Peak speed = 1 instruction per cycle = 3.07 GIPS
- The **sustained speed** of the Fibonacci Computer is the real speed achieved when executing the loop application code
  - Sustained speed =  $7/140 = 0.05$  GIPS
    - Each iteration takes 140 ns to execute 7 instructions
  - **Efficiency** = sustained speed / peak speed =  $0.05 / 3.07 = 1.63\%$

# Performance metrics of the 3.07-GHz FC

- The **peak speed** of the Fibonacci Computer is the maximal speed possible
  - Peak speed = 1 instruction per cycle = 3.07 GIPS
- The **sustained speed** of the Fibonacci Computer is the achieved speed when executing the loop application code
  - Sustained speed =  $7/140 = 0.05$  GIPS
  - **Efficiency** = sustained speed / peak speed =  $0.05 / 3.07 = 1.63\%$
- For the Fibonacci Computer with caching
  - Sustained speed =  $7/12 = 1.79$  GIPS
  - **Speedup** =  $1.79 \text{ GIPS} / 0.05 \text{ GIPS} = 35.8$
  - Efficiency = sustained speed / peak speed =  $1.79 / 3.07 = 58.3\%$
- Both have the same peak speed, but caching significantly improves real speed (sustained speed)

## 5.5.3 Parallel computing

- Also known as parallel processing
- What if one CPU is not enough?
  - What if we need 10 GIPS, 1 million GIPS, 1 trillion GIPS?
- Use multiple CPUs/processors/computers in one system
  - A processor having multiple CPUs is called a **multicore** processor
    - Each CPU is called a **core**



# Supercomputer examples

- Top500.org is a list ranking the world's fastest supercomputers
  - Maintained since 1993 by scientists in Europe and USA
- Rank the 500 fastest supercomputers by their speeds in executing the Linpack benchmark
  - Speed = executed 64-bit floating-point operations per second (FLOPS)
- The main contributing factor of progress is parallel computing

**The Linpack benchmark** program for solving a system of linear equations using Gaussian elimination. It finds  $\mathbf{x}$  in  $\mathbf{Ax} = \mathbf{b}$ , where  $\mathbf{A}$  is an  $N \times N$  matrix, and  $\mathbf{x}, \mathbf{b}$  are two  $N$ -dimensional vectors. For  $N=3$ , we have

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

| Time of Test    | 1993                  | 2020            | 1993-2020 Growth Factor |
|-----------------|-----------------------|-----------------|-------------------------|
| Top-1 Name      | Thinking Machine CM-5 | Fujitsu Fugaku  | N/A                     |
| Problem Size    | N = 52,224            | N = 20,459,520  | 392                     |
| Speed           | 59.7 GFlop/s          | 415,530 TFlop/s | 6,960,302               |
| Clock Frequency | 32 MHz                | 2.2 GHz         | 69                      |
| Parallelism     | 1,024 cores           | 7,299,072 cores | 7,128                   |
| Memory          | 32 GB                 | 4,866,048 GB    | 152,064                 |
| Power           | 96.5 KW               | 28,334.5 KW     | 294                     |
| Cost            | US \$30 million       | US \$1 billion  | 33                      |

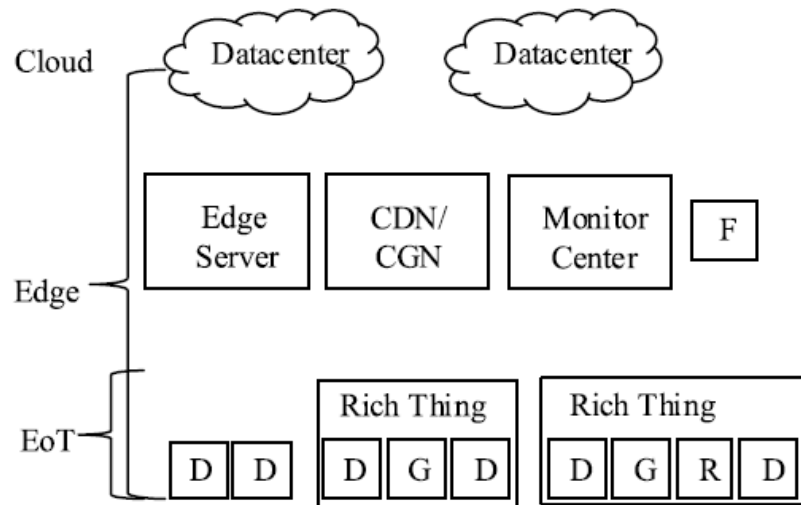
## 5.5.4\*\*\* The landscape of computing systems

- Focus on growing systems and future trends
  - Students may see when they graduate
- Computers
  - **Datacenters** for cloud computing
    - Computer **cluster**, i.e., a system of multiple computers
  - Computers for cyber-human-physical systems
  - Domain-specific computers
  - Open-source computers
- Application systems
  - AI + traditional applications
  - The network is the computer
    - The Internet is a computer

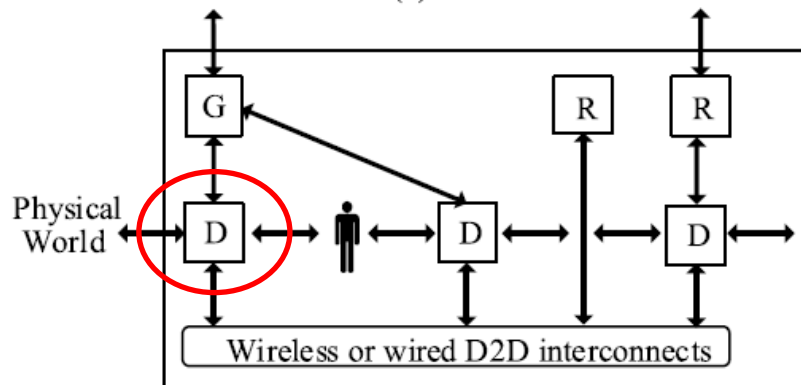
<https://www.gartner.com/en/newsroom/press-releases/2020-01-25-gartner-forecasts-worldwide-it-spending-to-grow-6-point-2-percent-in-2021>

| <b>Gartner Worldwide IT Prediction</b> | 2020 Spending (US\$B) | 2020 Growth (%) | 2021 Spending (US\$B) | 2021 Growth (%) |
|--|-----------------------|-----------------|-----------------------|-----------------|
| Data Center Systems                    | 215                   | 0.0             | 228                   | 6.2             |
| Enterprise Software                    | 465                   | -2.4            | 506                   | 8.8             |
| Devices                                | 653                   | -8.2            | 705                   | 8.0             |
| IT Services                            | 1012                  | -2.7            | 1073                  | 6.0             |
| Communications Services                | 1350                  | -1.7            | 1411                  | 4.5             |
| Overall IT                             | 3695                  | -3.2            | 3923                  | 6.2             |

# Computing devices of cloud, edge, and things



(a)



(b)

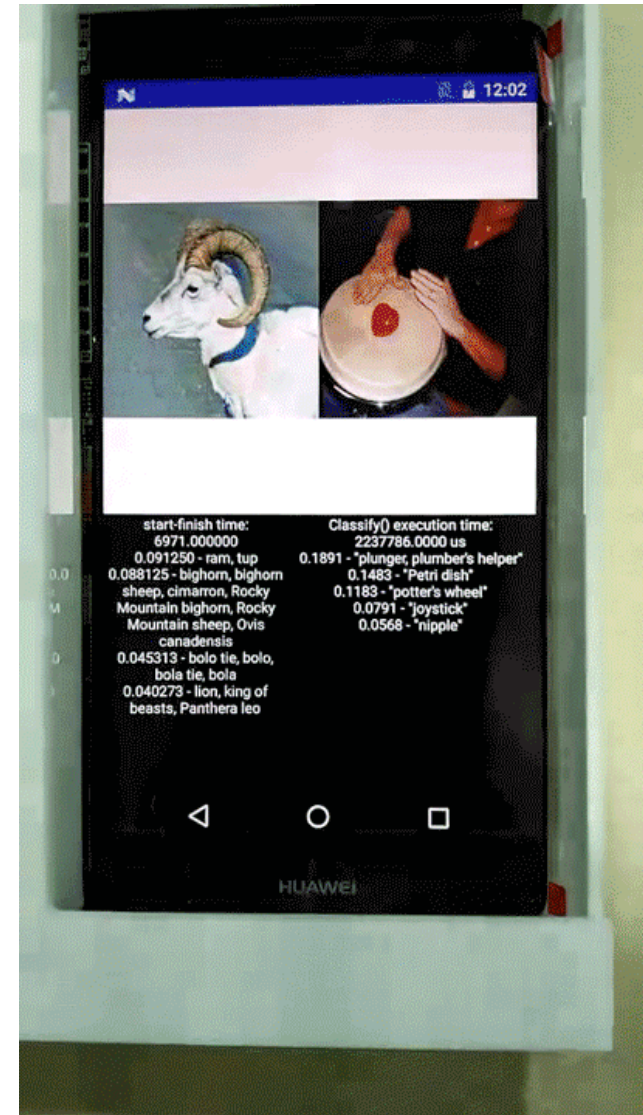
| Term (Label)            | Description   |
|-------------------------|---|
| Cloud devices           | Datacenters (cloud servers).  |
| Edge devices            | Any device out of the cloud, including fog device and things device.  |
| Fog device (F)          | Edge device that is not a things device, such as an edge server or a CDN.   |
| Simple device (D)       | Device for a simple thing, <i>e.g.</i> , a smart lamp, a microwave oven.  |
| Rich thing, Rich device | An integrated set of multiple simple devices with a natural physical enclosure, such as a smart vehicle or a smart home. A rich thing can be viewed as a rich device. |
| Gateway (G)             | Device used to connect the rich thing to other edge devices or the cloud, such as a WiFi router.  |
| Remote (R)              | Device used by users to interact with other devices in a rich thing, such as a smartphone.  |

## A device of thing has four interfaces

- West: bridging the physical world
- East: bridging the human society
- South: bridging other devices of things
- North: bridging edge and cloud devices

# Domain-specific computers

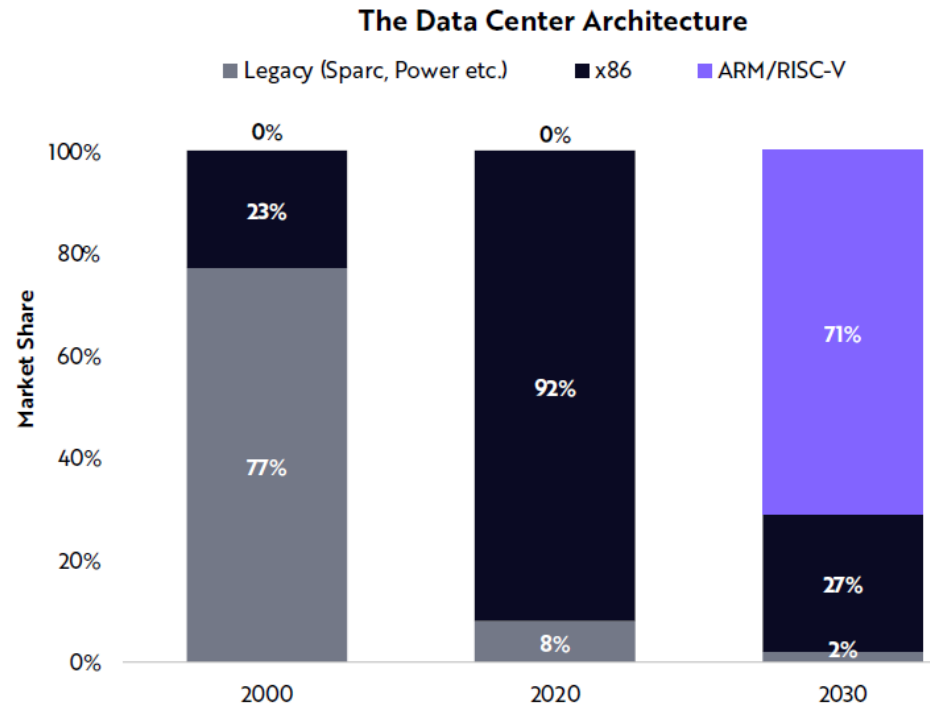
- A computer or a part of a computing system
  - Specifically designed for an application domain,
  - such that its functionality, performance, interface, efficiency, etc., are particularly effective for the target domain
- Processor examples
  - DSP: digital signal processing
  - GPU: graphic processing unit
    - A domain-specific processor may also be good for other domains
    - GPUs have been used for scientific computing and AI
  - DPU: for deep learning
- An inspirational perspective
  - Hennessy, J. L., & Patterson, D. A. (2019). A new golden age for computer architecture. *Communications of the ACM*, 62(2), 48-60.



# Open-source computer example: RISC-V

- Form the official site: <https://riscv.org/about/>
  - RISC-V is a free and open ISA enabling a new era of processor innovation through open standard collaboration.
  - The RISC-V ISA delivers a new level of free, extensible software and hardware freedom on architecture, paving the way for the next 50 years of computing design and innovation.
    - ISA = instruction set architecture
- Originated at UC-Berkeley
  - RISC-V = Berkeley RISC Five
- RISC-V has become an international effort
  - RISC-V International
  - > 1000 members in 50 countries

Asanović K, Patterson D A. Instruction sets should be free: The case for risc-v. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146, 2014.

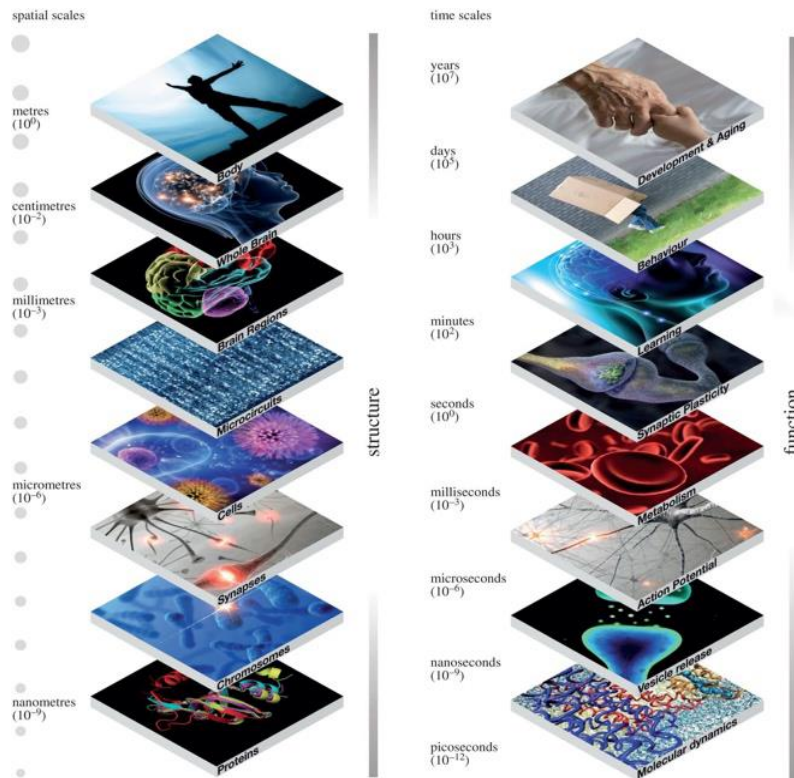


ARK Invest, Big Ideas 2021 Report,  
<https://ark-invest.com/big-ideas-2021/>

# The trend of intelligent application systems

## For sciences

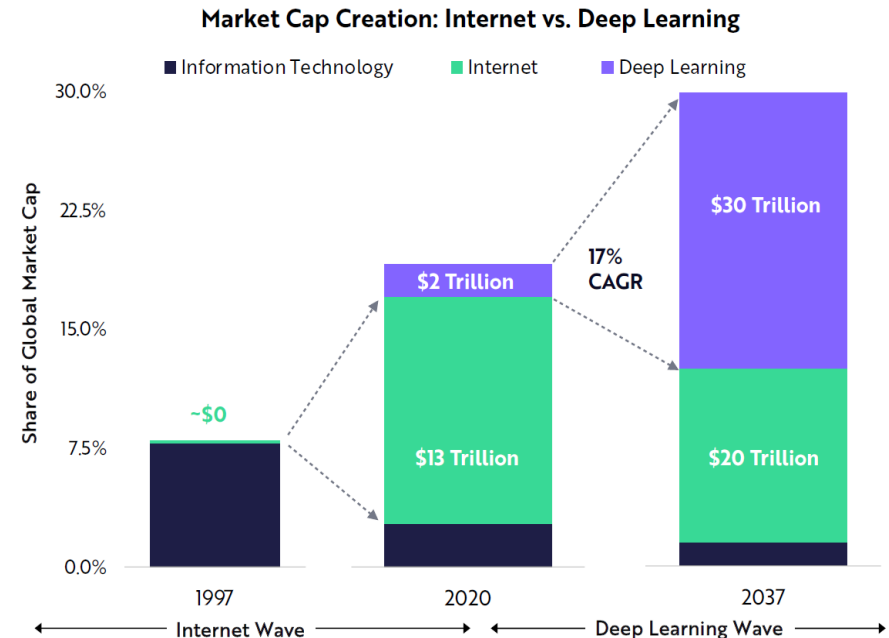
Enable inquiry of all structures & functions



Stevens, R., Taylor, V., Nichols, J., Maccabe, A. B., Yelick, K., & Brown, D. (2020). AI for Science (No. ANL-20/17). Argonne National Lab.(ANL), Argonne, IL (United States).

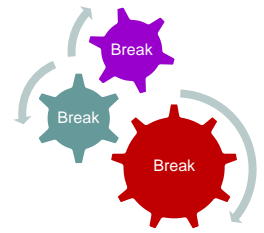
## For economy

Value of AI companies > IT + Internet



ARK Invest, Big Ideas 2021 Report,  
<https://ark-invest.com/big-ideas-2021/>

# Take-Home Messages



- Amdahl's law regards performance enhancement
  - After enhancing a portion of a system, the speedup is upper bounded by the reciprocal of the other portion's time
    - Total time = 1, enhanced portion's time =  $1-f$ , other portion's time =  $f$
    - The targeted portion is enhanced  $p$  times
    - Speedup =  $(1 / ((1-f)/p + f)) \rightarrow 1/f$ , when the enhancement factor  $p \rightarrow \infty$
- Three examples to utilize Amdahl's law
  - Pipelining: steps overlap pipeline stages
  - Caching: insert a small but fast buffer to utilize localities
  - Parallel computing: use multiple units (e.g., CPU cores) to perform a computational job (e.g., Linpack)
- The computer landscape is growing rapidly
  - Five types of examples, e.g., open-source hardware