CUDA Programming - Introduction

Dr. Nileshchandra Pikle
Assistant Professor,
VITAP
&
"A certified CUDA Programming instructor "
By

NVIDIA DLI

About me

PhD in computer science and engineering

Area of Interest: Parallel Computing, Machine Learning, Image Processing etc.

Certified CUDA programming instructor by NVIDIA

NVIDIA DEEP LEARNING INSTITUTE

Certificate of Completion

This certificate is awarded to:

Nileshchandra Pikle

For the successful completion of:

Fundamentals of Accelerated Computing with CUDA C/C++

A Course of Study Offered by The Deep Learning Institute.

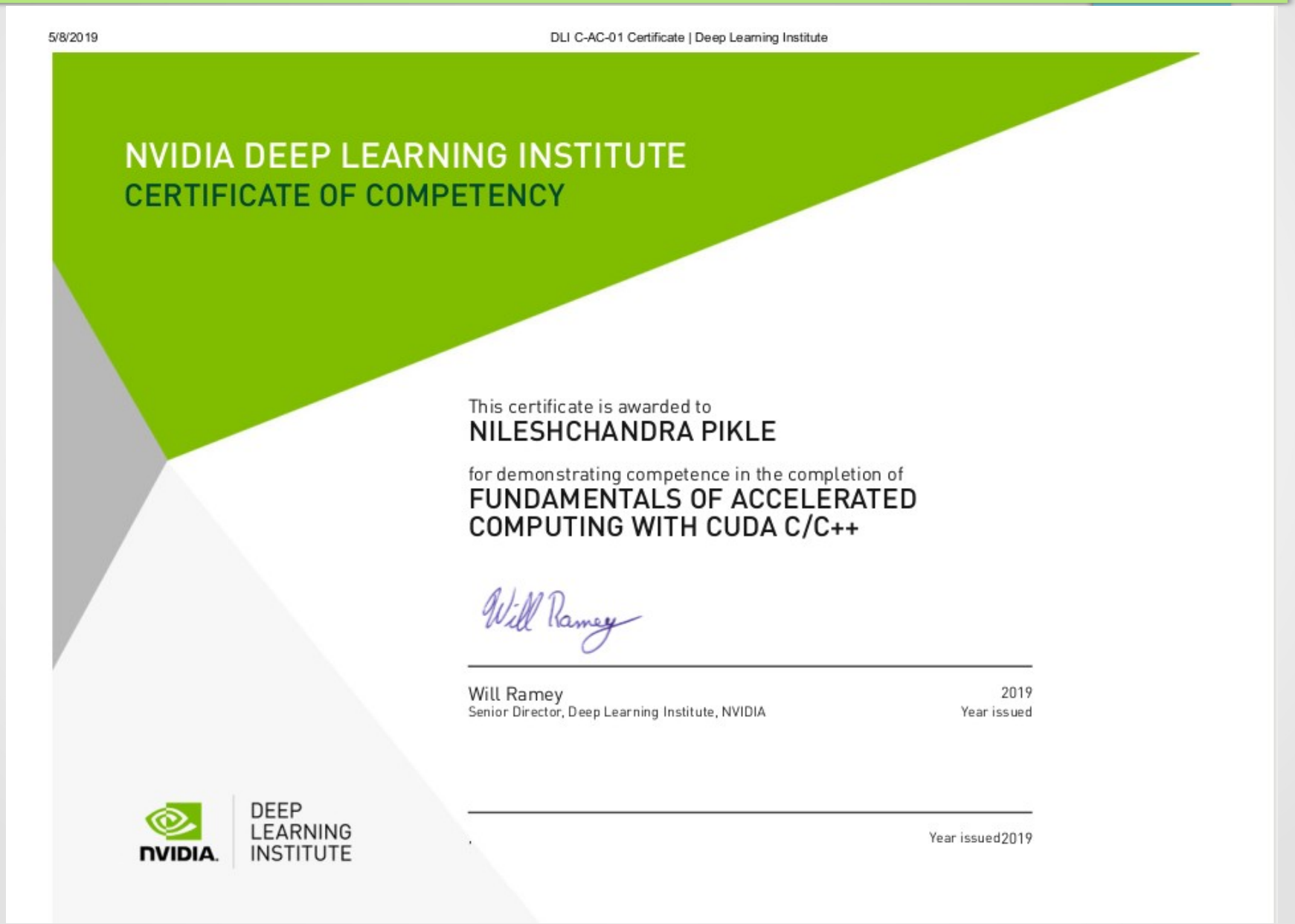
Will Ramey

Will Ramey
Director of Deep Learning Institute
NVIDIA

Certificate ID Number: 7b0c253de4a24edfa725221eebd7310e

November 19, 2018

Certificate



5/8/2019 DLI C-AC-01 Certificate | Deep Learning Institute

NVIDIA DEEP LEARNING INSTITUTE
CERTIFICATE OF COMPETENCY

This certificate is awarded to
NILESHCHANDRA PIKLE

for demonstrating competence in the completion of
FUNDAMENTALS OF ACCELERATED COMPUTING WITH CUDA C/C++

Will Ramey
Senior Director, Deep Learning Institute, NVIDIA

2019
Year issued

NVIDIA DEEP LEARNING INSTITUTE

Year issued2019

Contents

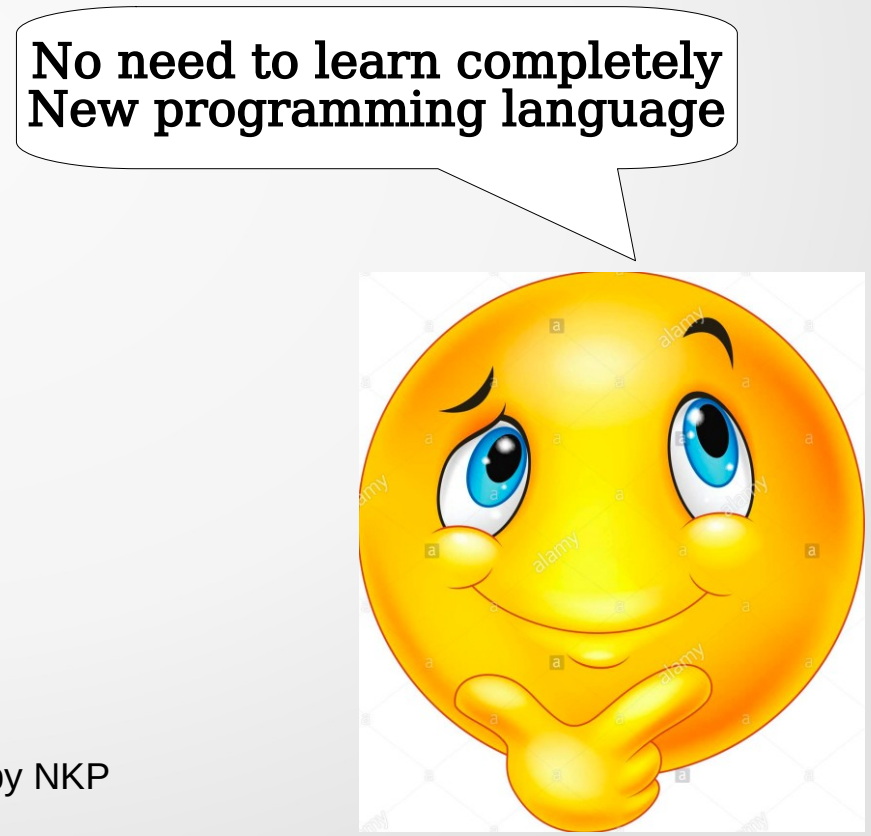
- Prerequisites
- What is parallel computing?
- Sequential vs parallel?
- What is CPU - advancements and limitations
- Moore‘s law
- Multi-core architectures
- Performance metrics

Prerequisites

- C/C++ Programming
- Algorithms
- Data Structures
- Computer Organization
- Operating System

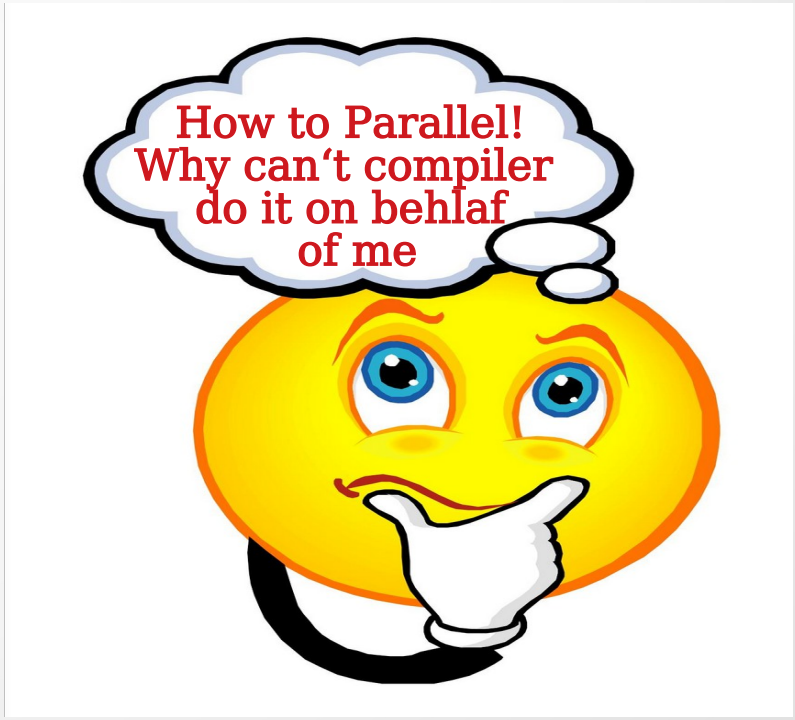
Prerequisites

- C/C++ programming **CUDA is an extension of such languages**
- Algorithms
- Data structures
- Computer organization
- Operating System

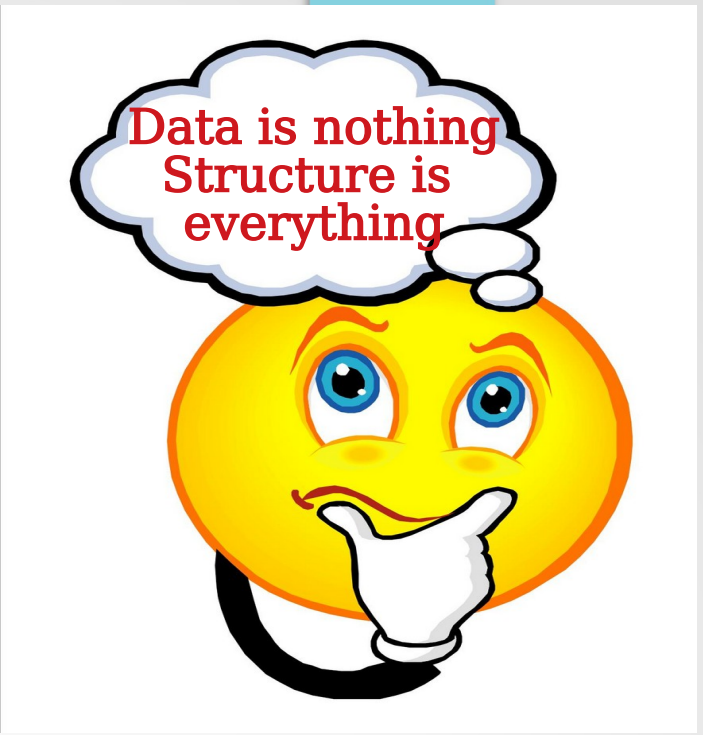


Prerequisites

- C/C++ programming
- Algorithms **Thinking in parallel needs sequential thinking first**
- Data structures
- Computer organization
- Operating System



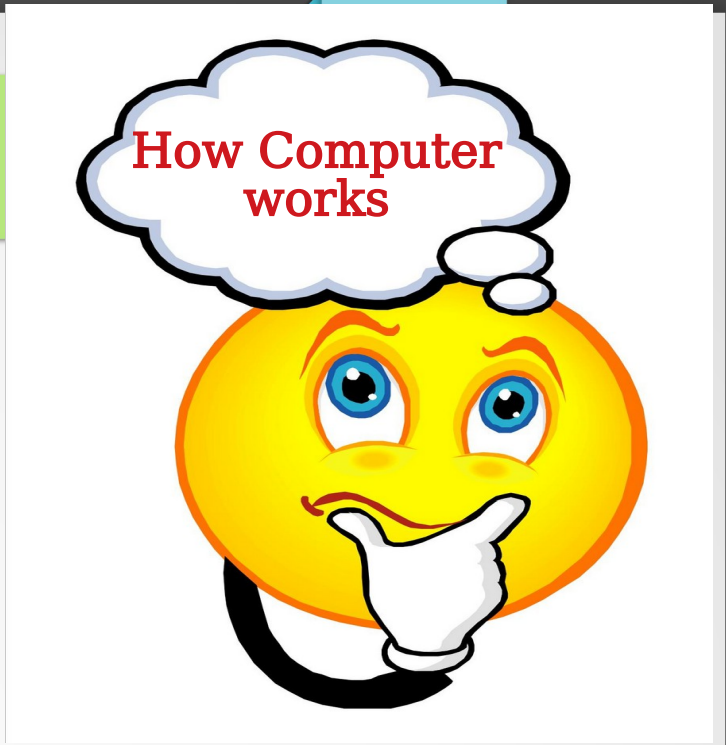
Prerequisites



- C/C++ programming
- Algorithms
- Data structures

  Store Access Modify data structures such as arrays, structures etc.
- Computer organization
- Operating System

Prerequisites



- C/C++ programming
- Algorithms
- Data structures
- Computer organization **Clocks, frequency, data and address bus, etc.**
- Operating System

Prerequisites

- C/C++ programming
- Algorithms
- Data structures
- Computer organization
- Operating System: **Memory management**

Sequential vs Parallel Work

Scenario 1

Task:	Digging a 20 meter big hole for preserving water
Resources:	1 Worker, 1 shovel with 1 tooth
Efficiency:	1 meter per hour
Time:	20 Hrs.

Sequential vs Parallel Work

Scenario 1

CPU	Single Core
Clock Frequency	2 GHz
Pipelining	NO
Branch Prediction	NO
Cache	Small

Time: 10 min.

Sequential vs Parallel Work

Scenario 2

Task: Digging a 20 meter big hole for preserving water
Resources: 1 Worker
Efficiency: 1.5 meter per hour
Time: 13.33 Hrs.

Sequential vs Parallel Work

Scenario 2

CPU	Single Core
Clock Frequency	4 GHz
Pipelining	YES
Branch Prediction	YES
Cache	Large

Time: 5 min.

Sequential vs Parallel Work

Scenario 3

Task: Digging a 20 meter big hole for preserving water
Resources: 4 Workers
Efficiency: 6 meter per hour
Time: 3.33 Hrs.

Sequential vs Parallel Work

Scenario 3

CPU	Quad Core
Clock Frequency	4 GHz
Pipelining	YES
Branch Prediction	YES
Cache	Large

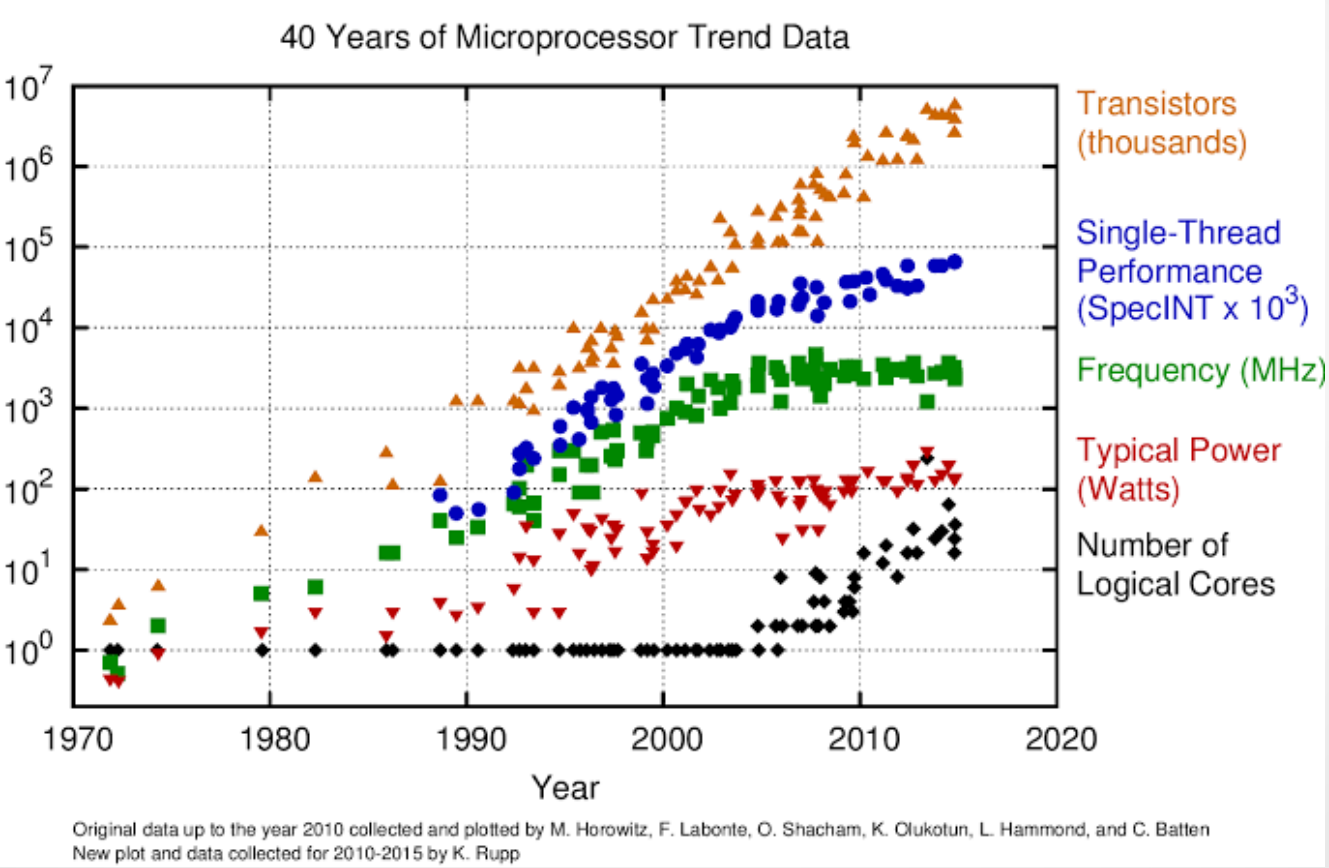
Time: 1.2 min.

Central Processing Unit (Recap)

- **How to increase the performance of a single core CPU**
  1. Clock frequency/ clock cycles
  2. Instructions per cycle (IPC)
  3. Pipeline
  4. Number of transistors
  5. Word length
  6. Data bus and address bus
  7. Multi-core processors
  8. RAM- frequency, synchronous, DDR3 etc.
  9. Memory latency
  10. Memory bandwidth

Clock Frequency

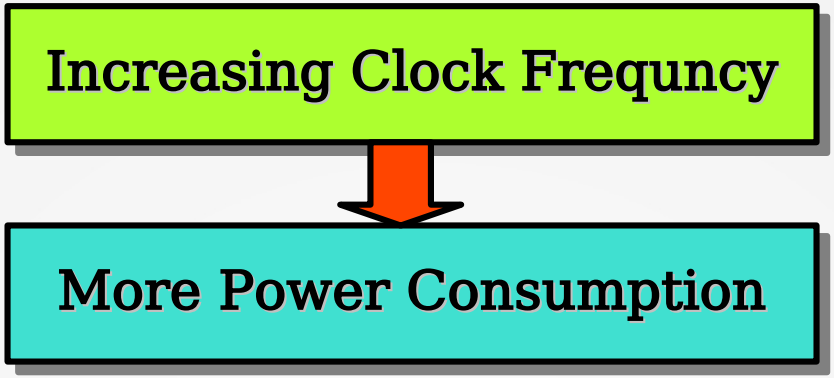
- **Clock frequency:** Rate at which processor executes instructions



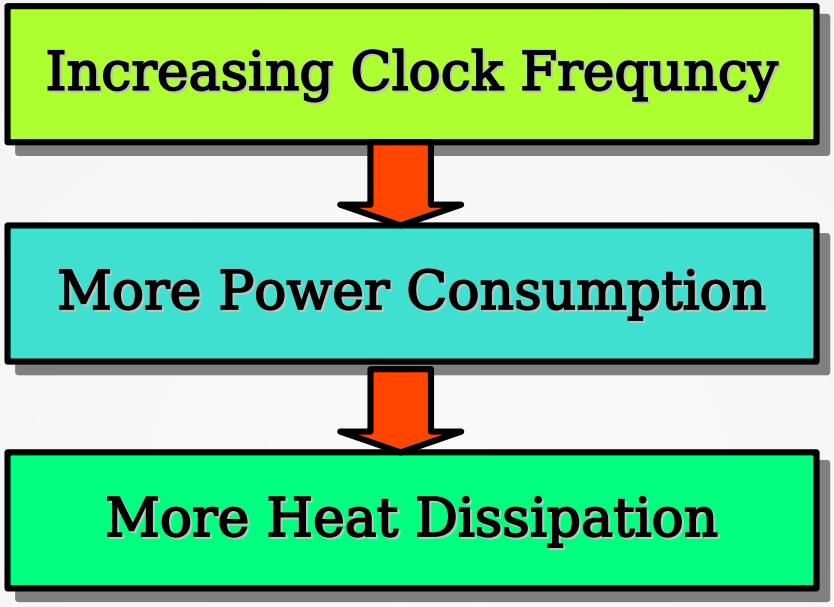
Clock Frequency Limitations

Increasing Clock Frequncy

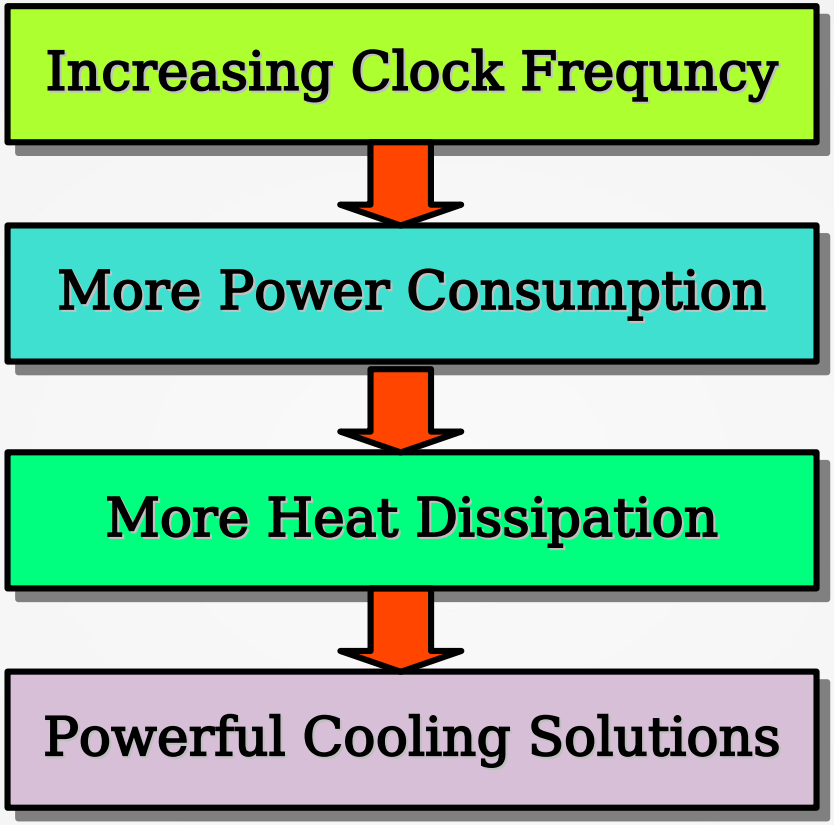
Clock Frequency Limitations



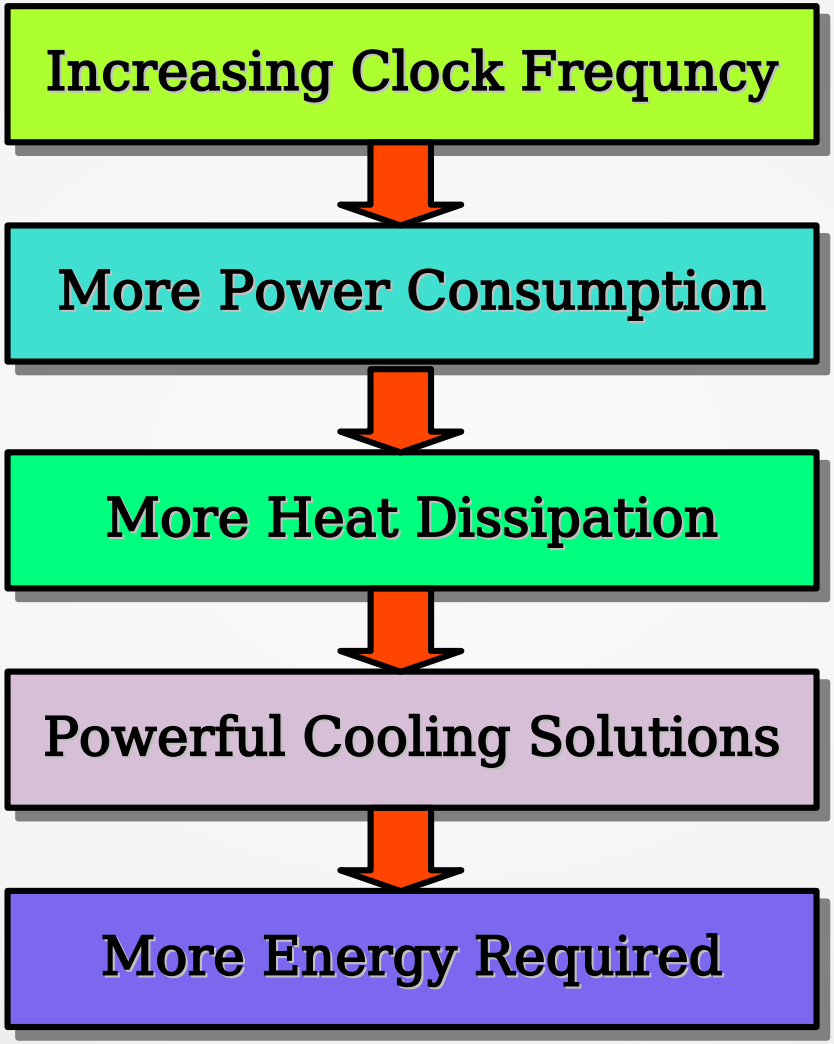
Clock Frequency Limitations



Clock Frequency Limitations

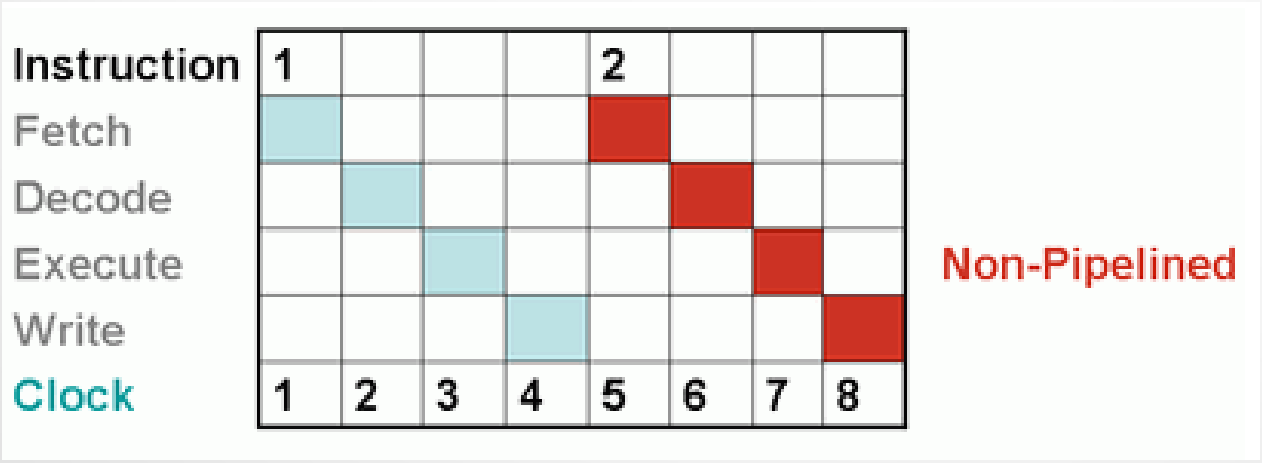


Clock Frequency Limitations



Instruction Per Cycle (IPC)

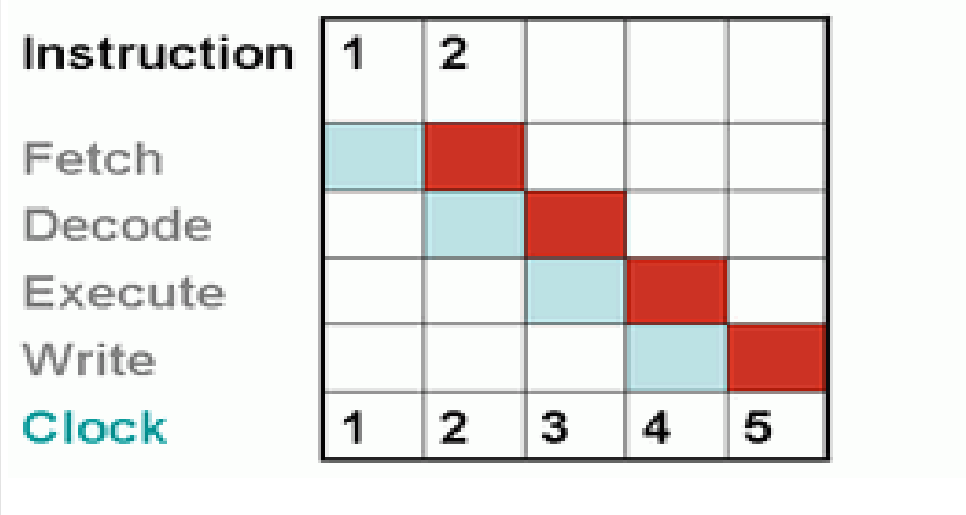
- **Instruction execution on older CPU**
  1. Bring data from main memory to CPU – **FETCH**
  2. Data is decoded at CPU – **DECODE**
  3. Execute instructions on the data – **EXECUTE**
  4. Write data from CPU to main memory - **WRITE**



IPC = 1/4

Pipelining

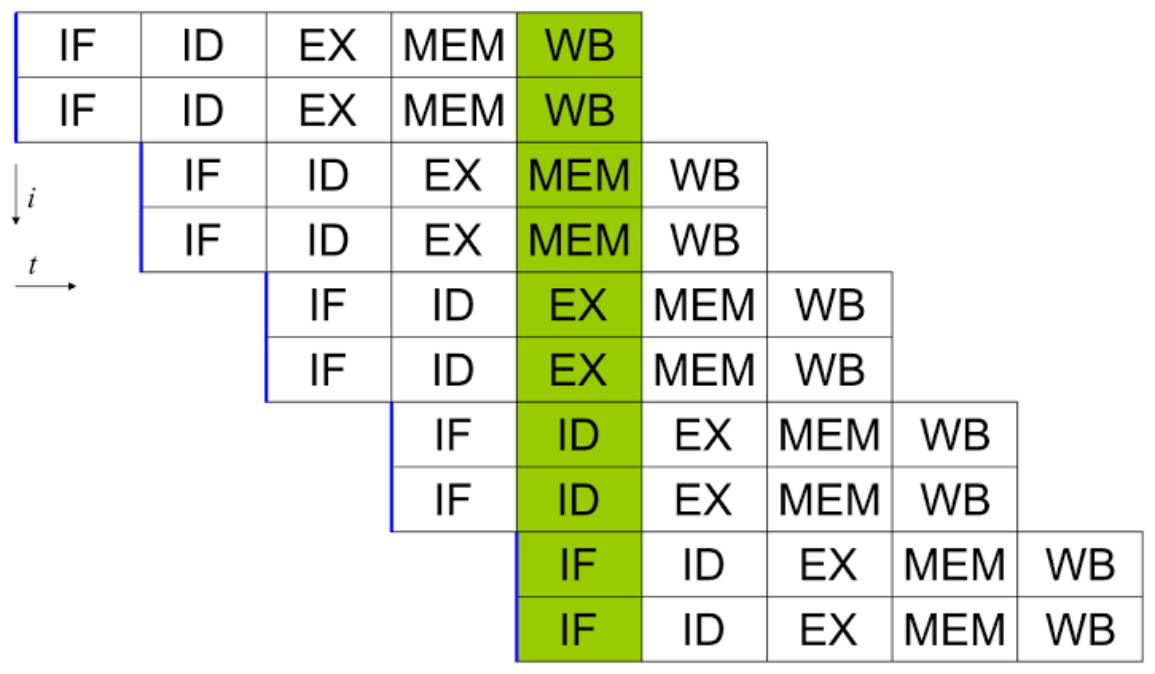
- **Pipelining to utilize every clock cycle efficiently**
  1. Bring data from main memory to CPU – **FETCH**
  2. Data is decoded at CPU – **DECODE**
  3. Execute instructions on the data – **EXECUTE**
  4. Write data from CPU to main memory - **WRITE**



IPC = 1

Pipelining

- **Advancements**
  1. Better branch prediction logic – **Avoid stalls**
  2. Superscalar pipeline–**Multiple execution sub units**
  3. Instruction Level Parallelism – **Same thing in parallel**



Pipelining

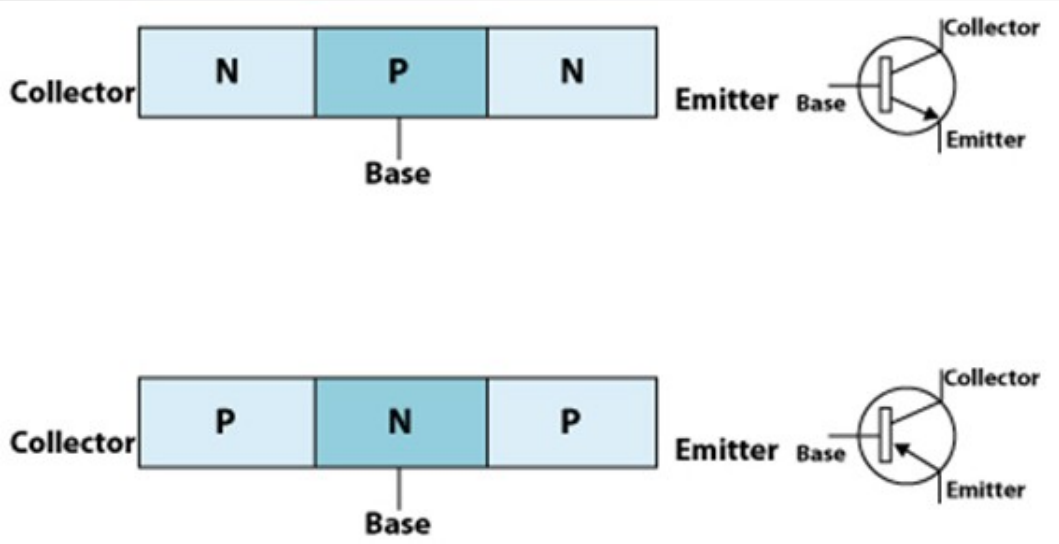
- **Limitations!!!**

  Adding more stages results in

  1. Circuitary complexity
  2. Bigger size occupation
  3. More power consumption

Transistors

- **Increase the number of transistors**
  1. Put more transistor in small area
  2. As transistors are smaller they **require less energy** to ON/OFF
  3. **Less time** required to ON/OFF



- Current size is 14 nm
- 14 times wider than DNA molecule
- Nowadays 70 silicon atoms wide transistors
- Silicon atomic size is 0.2 nm

Transistors

- **Increase the number of transistors**

  Dennard Scaling

  Power needed to run transistors in a particular unit volume stays constant even as the number of transistors increases

- However, transistors become so small that Dennard scaling no longer holds!!!

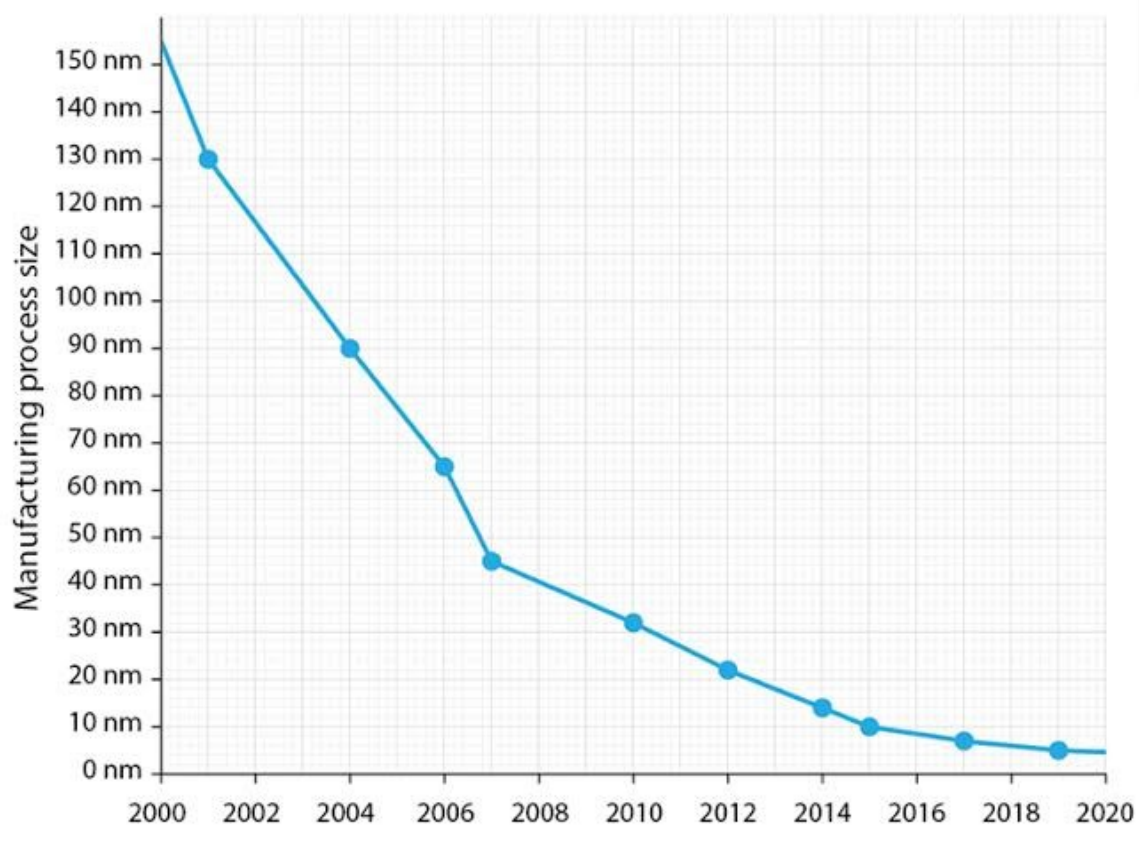
  Because Leakage current was not considered

- **Physics limit:-** How further we can go?

  Final is electron size!

Transistors

- Reduction in transistor size over years



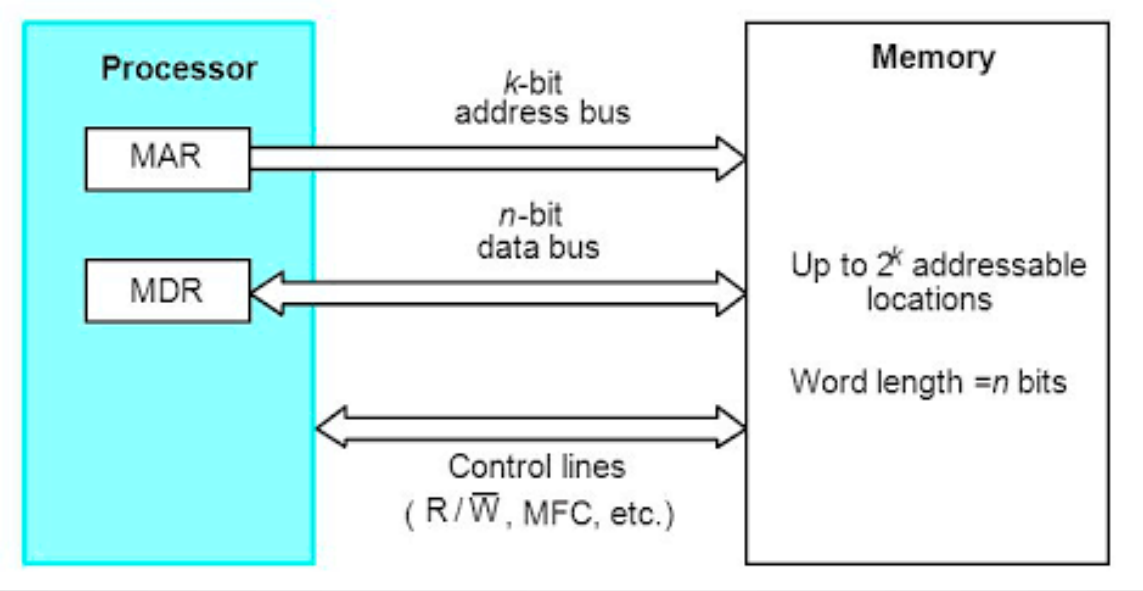
Intel working on 10nm size

But
What
Is
Next!!!

Physics limit
MAY end here!

Word length

- **Word length:** Number of bits processed by CPU in one go
- What do yo mean by 8, 16, 32 or 64-bit processor?
- When a processor needs a data from memory it is done with wordsize quantities



Word length

- **Simple addition of two 64-bit numbers on**

1. 64-bit processor

```
        ADDA [Num1], [Num2]
        STA [RESULT]
```

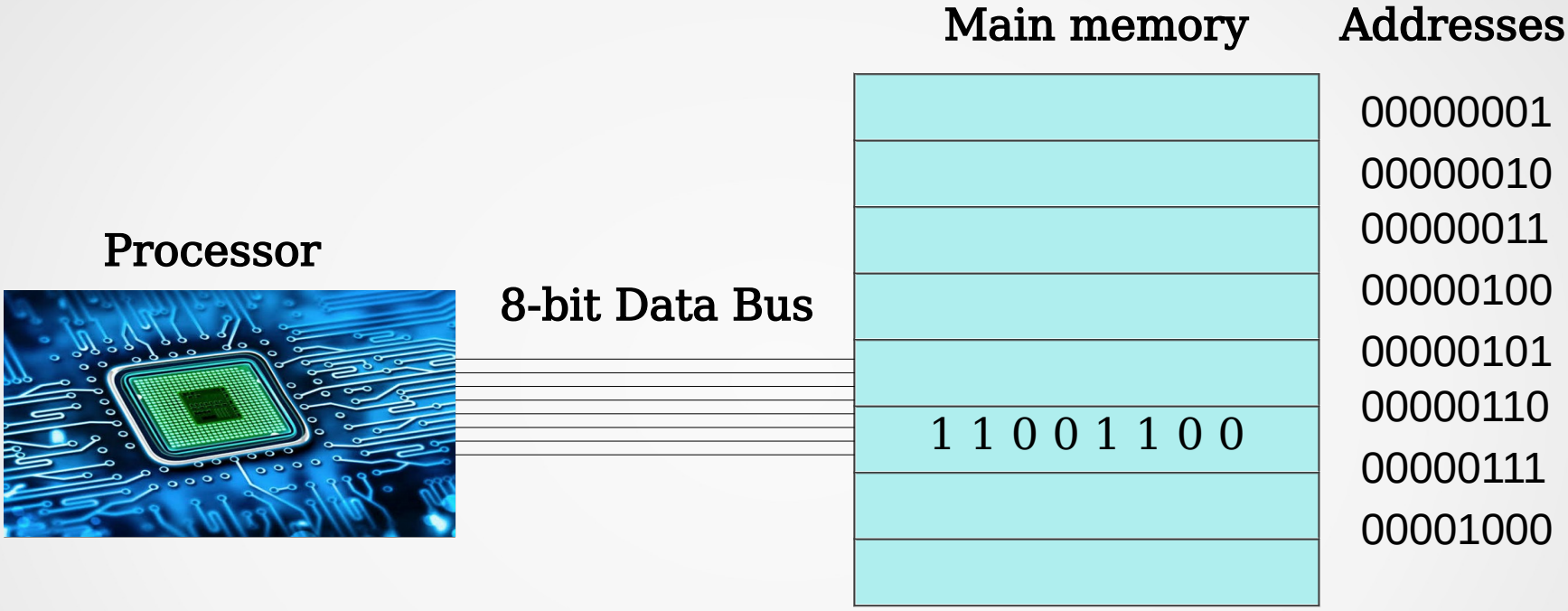
Require less cycles

2. 32-bit processor

```
        ADDA [Num1_lower], [Num2_lower]
        STAA [RESULT_lower]
        BRNO CMPS
        ADDA #1
CMPS:   ADDA [Num1_upper], [Num2_upper]
        STAA [RESULT_upper]
```

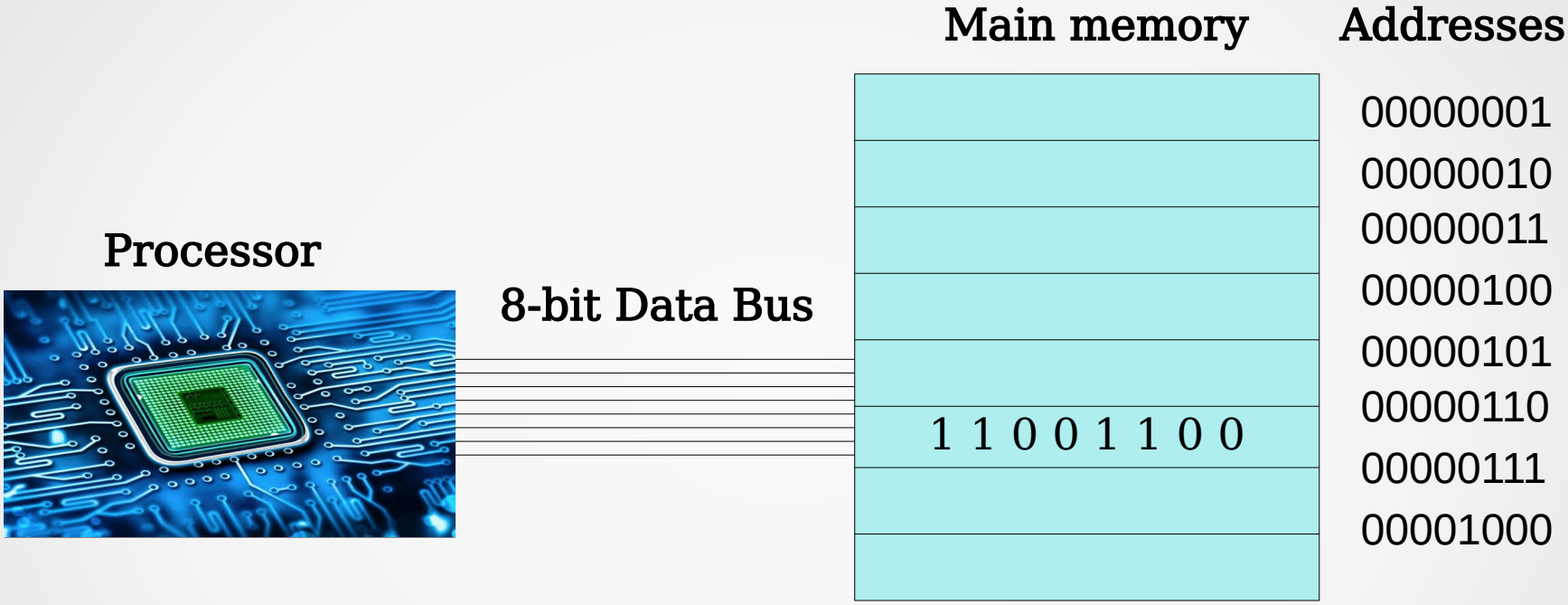
Require more cycles

Data bus and Address bus



Eg. For 200 Mhz DDR Memory what is the theoretical bandwidth?

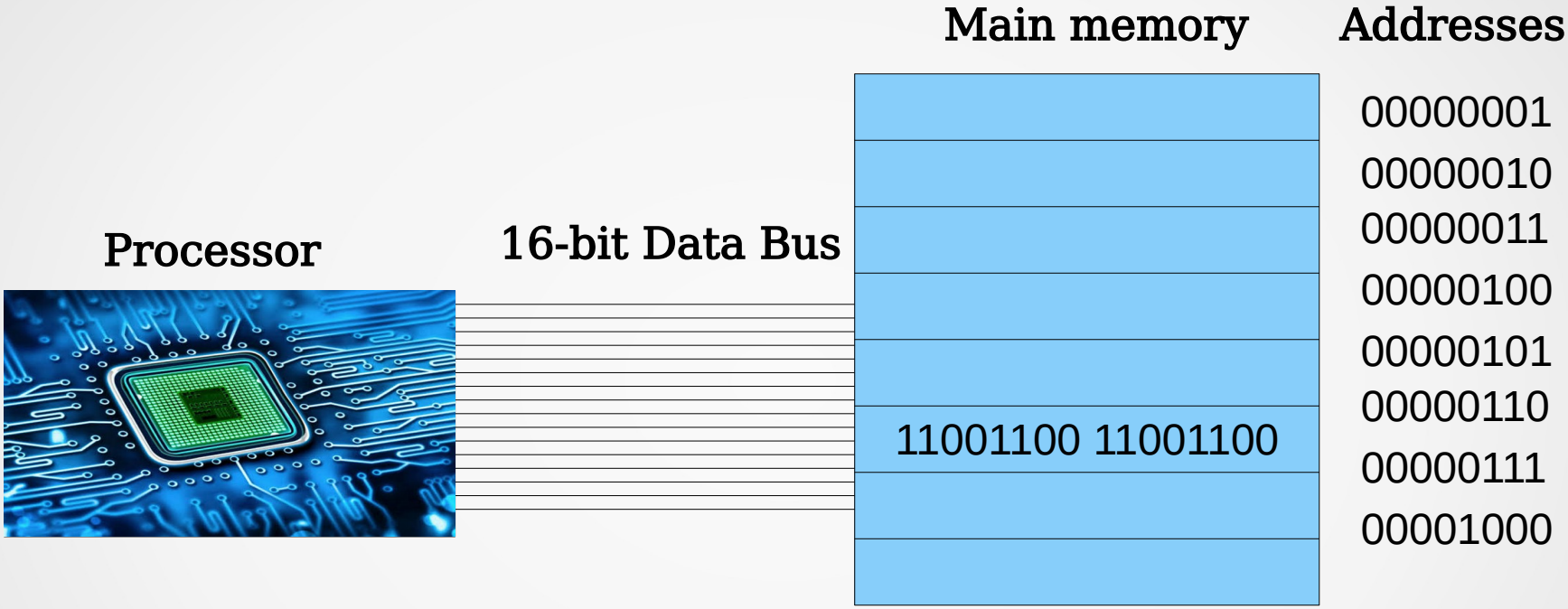
Data bus and Address bus



Eg. For 200 Mhz DDR Memory what is the theoretical bandwidth?

$$B = 200 \times 2 \times 8 \times 10^6 = 3200\, bits/sec = 400\, MB/s$$

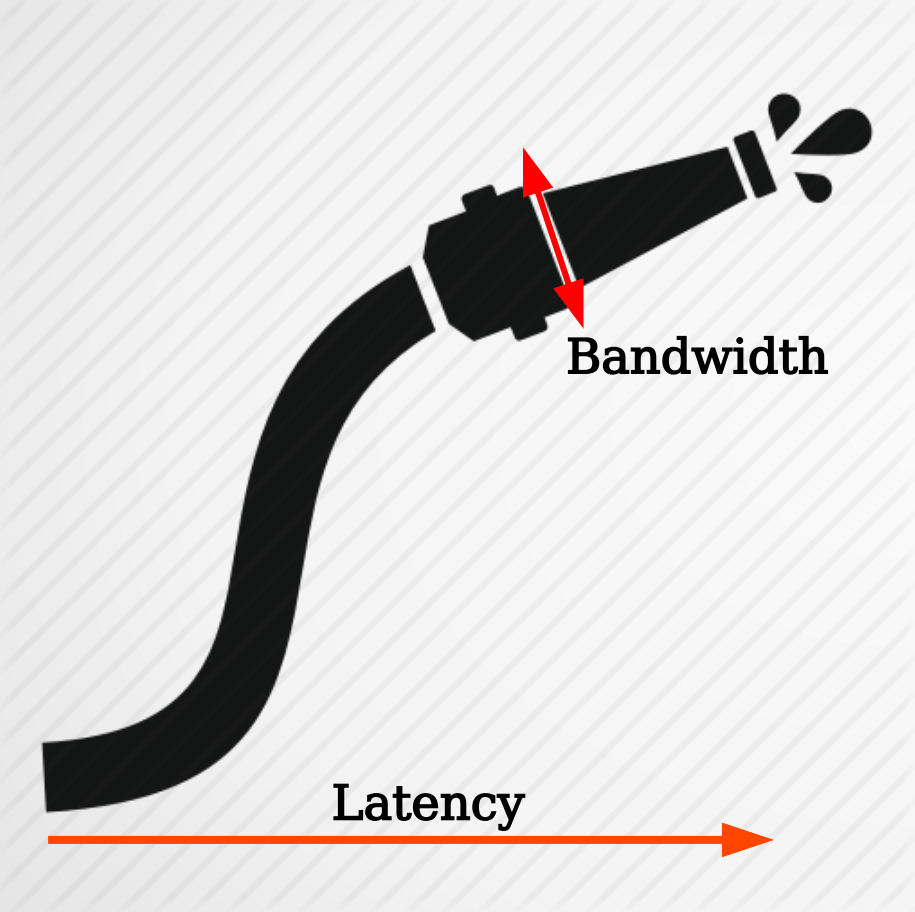
Data bus and Address bus



Eg. For 200 Mhz DDR Memory what is the theoretical bandwidth?

$$B = 200 \times 2 \times 16 \times 10^6 = 6400\, bits/sec = 800\, MB/s$$

Memory latency and bandwidth



To reduce latency
1. Increase the water force

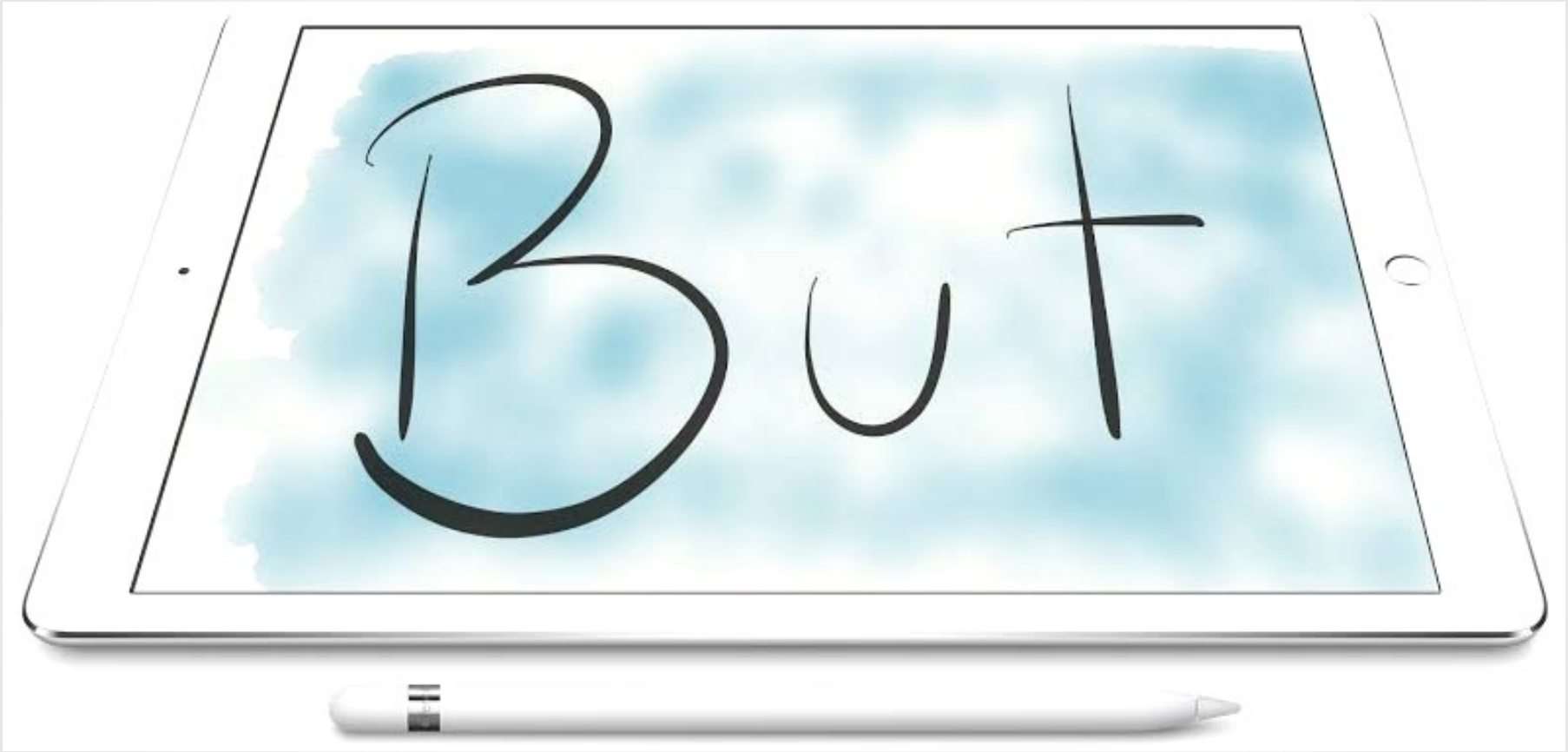
To increase bandwidth
2. Increase nosal diameter

Quick Recap

- **Which of the following makes faster CPU?**
  1. Longer clock period
  2. Increase clock frequency
  3. More work per cycle
  4. Increase hard disk
  5. Increase RAM
  6. Increase transistor count
  7. Increase data bus width
  8. Increase address bus width
  9. Pipeline

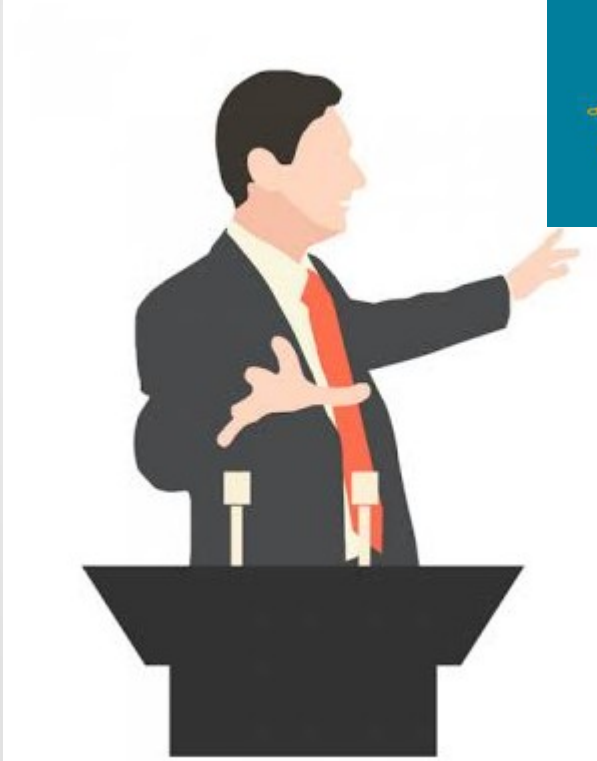
Quick Recap

- **Which of the following makes faster CPU?**
  1. Longer clock period
  2. Increase clock frequency
  3. More work per cycle
  4. Increase hard disk
  5. Increase RAM
  6. Increase transistor count
  7. Increase data bus width
  8. Increase address bus width
  9. Pipeline



Limits reached?

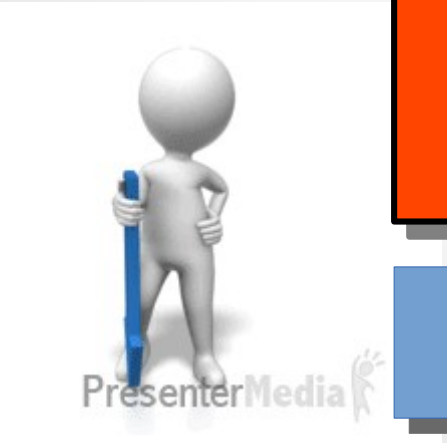
Is Moore‘s law dead?



Debate

Is Moore‘s law dead?

- Human brain have had **100 billion** neurons for
- at least the last **35,000 years.**
- Yet we learned to do a lot more with **same**
- **computing power**



Its a whole new game!

New ways to utilize those 10 billion transistors

Multi-core and many-core solutions

Think Parallel!

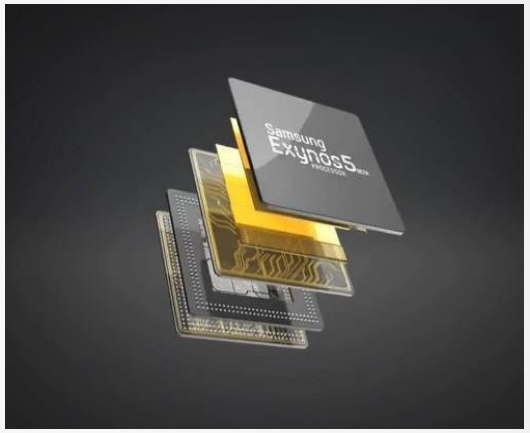
Why to make a single core CPU so powerful ?

Think Parallel!

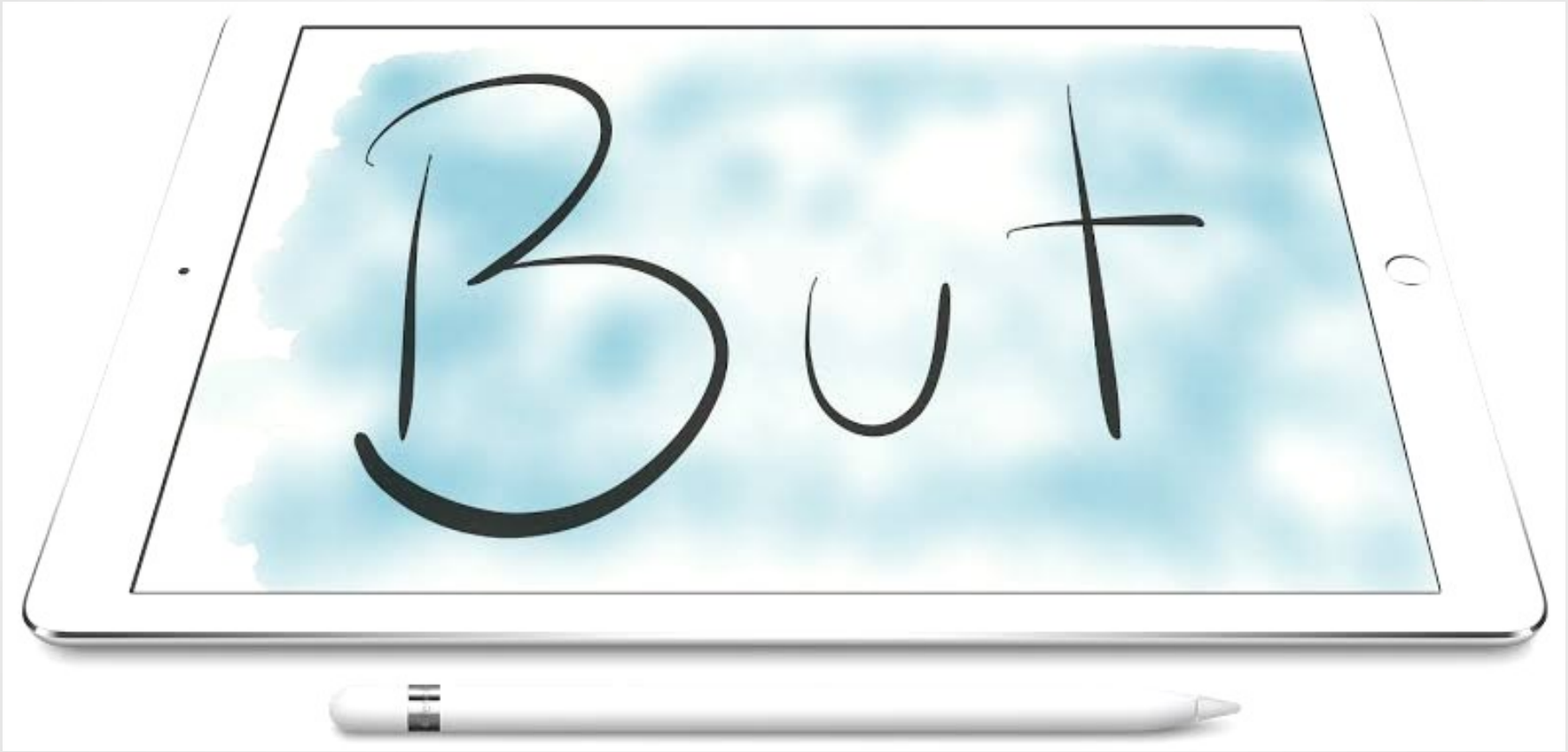
Put more number of cores in a CPU

Multi-core architectures

- What is Dual-core, quad core, octa-core....?



- Quad-core Laptop vs Quad-core Mobile?



How to use these cores?

Suppose matrix-matrix multiplication program executed on

1. Single core processor
2. Intel dual core CPU
3. Quad-core CPU
4. Octa-core CPU

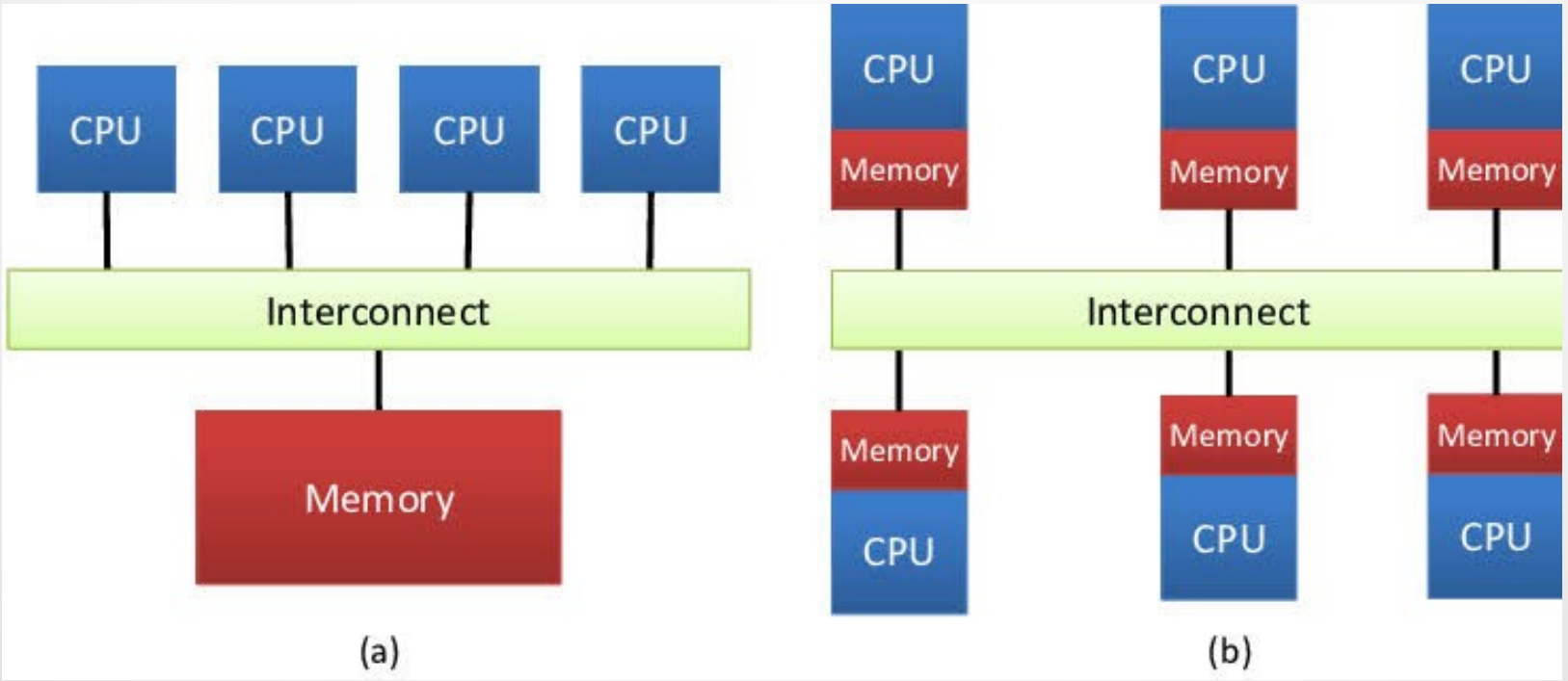
Assume each processor have same frequency

Which one is faster?

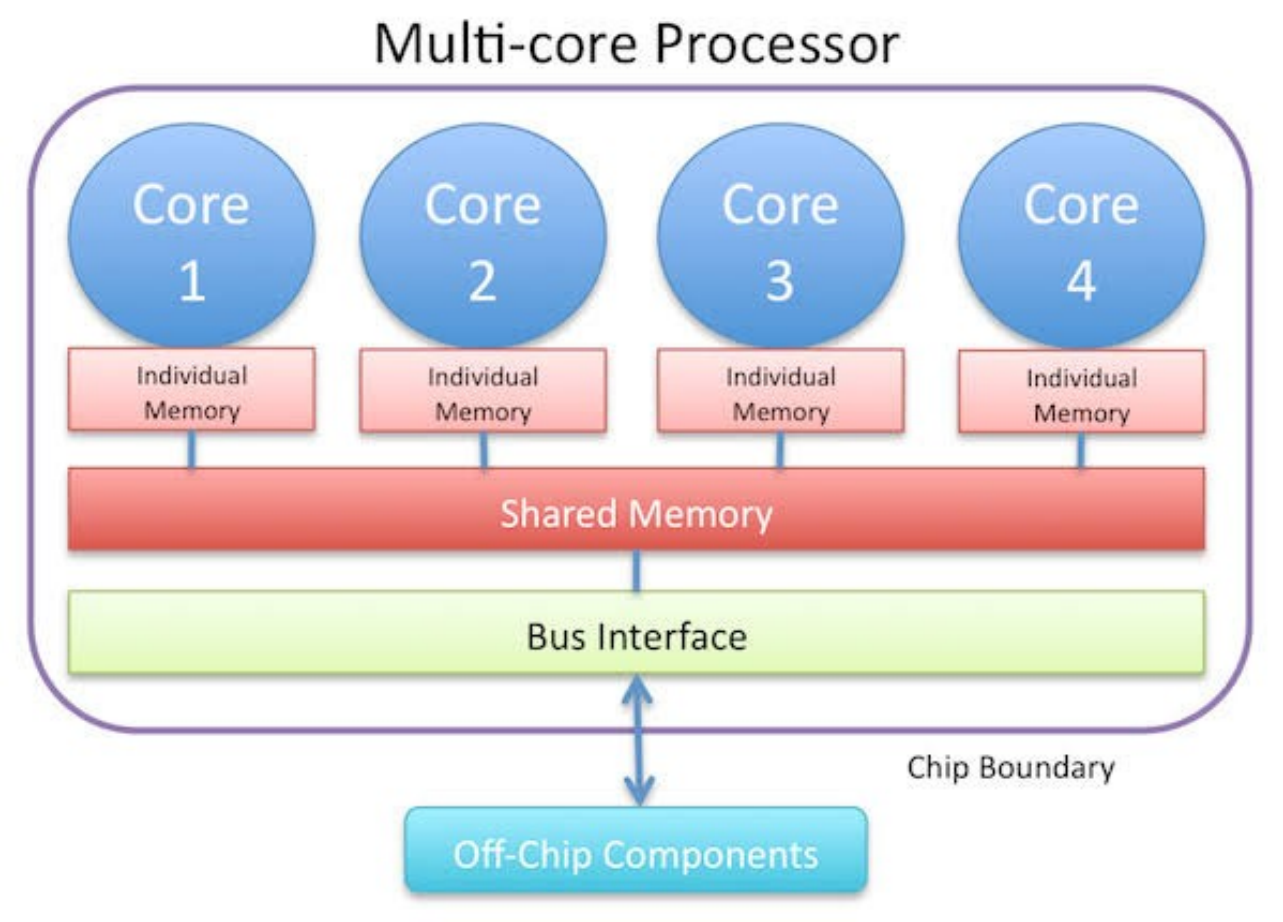
Paralle Architectures

(a) **Shared memory architecture:** Communication by R/W

(b) **Distributed memory architecture:** Communication by Message Passing



Multi-core architectures



Performance Matrics of Parallel Computing

Performance metrics

- Two main goals to be achieved with the design of parallel applications are:

  - **Performance:**

    The capacity to reduce the time to solve the problem when the computing resources increase;

  - **Scalability:**

    The capacity to increase performance when the complexity, or size of the problem, increases.

Performance metrics

Factors limiting the performance and the scalability

- **Architectural Limitations**
  1. Latency
  2. Bandwidth
  3. Data Coherency
  4. Memory Capacity

- **Algorithmic Limitations**
  1. Missing Parallelism
  2. Communication Frequency
  3. Synchronization Frequency
  4. Poor Scheduling (task granularity/load balancing)

Performance metrics

Performance Metrics for Parallel Applications

- Speedup
- Efficiency
- Redundancy
- Utilization
- Quality

Performance metrics

Performance Metrics for Parallel Applications

- **Speedup (S)**

  Speedup is a measure of performance. It measures the ration between the sequential execution time and the parallel execution time.

$$S(p) = \frac{T(1)}{T(p)}$$

T(s) = Sequential run time T(p) = Parallel run time

Performance metrics

Performance Metrics for Parallel Applications

- **Efficiency (E)**

  Efficiency is a measure of the usage of the computational resources. It measures the ration between performance and the resources used to achieve that performance.

$$E = \frac{S(p)}{p} = \frac{T(1)}{p \cdot T(p)}$$

Amdahl's Law

- The computations performed by a parallel application are of 3 types:
- **C(seq):** Computations that can only be realized sequencially.
- **C(par):** Computations that can be realized in parallel.
- **C(com):** Computations related to communication/ synchronization/ initialization.

$$S(p) = \frac{T(1)}{T(p)} = \frac{C(seq) + C(par)}{C(seq) + \frac{C(par)}{p} + C(com)}$$

Amdahl's Law

$$S(p) = \frac{T(1)}{T(p)} = \frac{C(seq) + C(par)}{C(seq) + \frac{C(par)}{p} + C(com)}$$

If **f** is the fraction of the computation that can only be realized sequentially

$$f = \frac{C(seq)}{C(seq) + C(par)}$$

Amdahl's Law

- Let $0 \leq f \leq 1$ be the computation fraction that can only be realized sequentially.

The Amdahl law tells us that the maximum speedup that a parallel application can attain with p processors is:

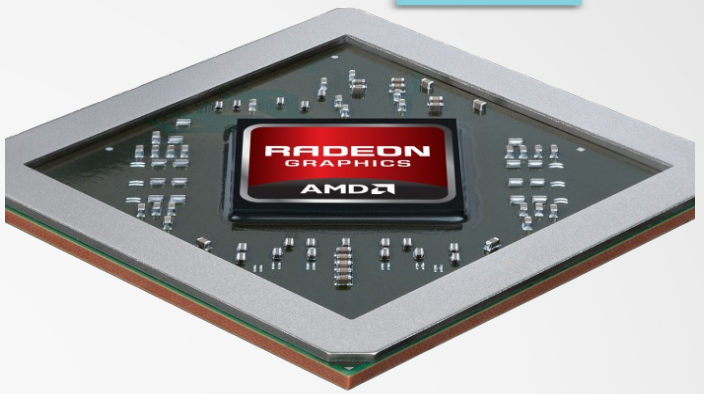
$$S(p) \leqslant \frac{1}{f + \frac{(1-f)}{p}}$$

Summary

- Pre-requisites
- Sequential vs Parallel work
- How to increase the performance of CPU
- Moore‘s law existance
- Multi-core architecture
- Performance metrics

Graphics Processing Unit

AMD



Guru3D.com

NVIDIA