EE5201- Computer Architecture

Designing for Performance

Dr. Subodha Gunawardena
(Original Lecture Material from Mr. T. D. Gamage)
Department of Electrical and Information Engineering
Faculty of Engineering, University of Ruhuna

Learning Outcomes

LO-1 : Describe the evolution of the design of computing devices and the purpose of related technologies with respect to performance, costs and capacity.

LO-2 : Describe the design and functional aspects of John von Neumann Architecture of computers based on the major constituencies

LO-3 : Describe architectural and organizational aspects of modern computer components

LO-4 : Describe design aspects and operating principles of currently prevailing computer technologies that correspond to basic components of computer and Instruction Set.

LO-5 : Demonstrate the ability to write simple assembly programs using instruction set of Intel x86 via assembly using ISA aspects.

Designing for performance

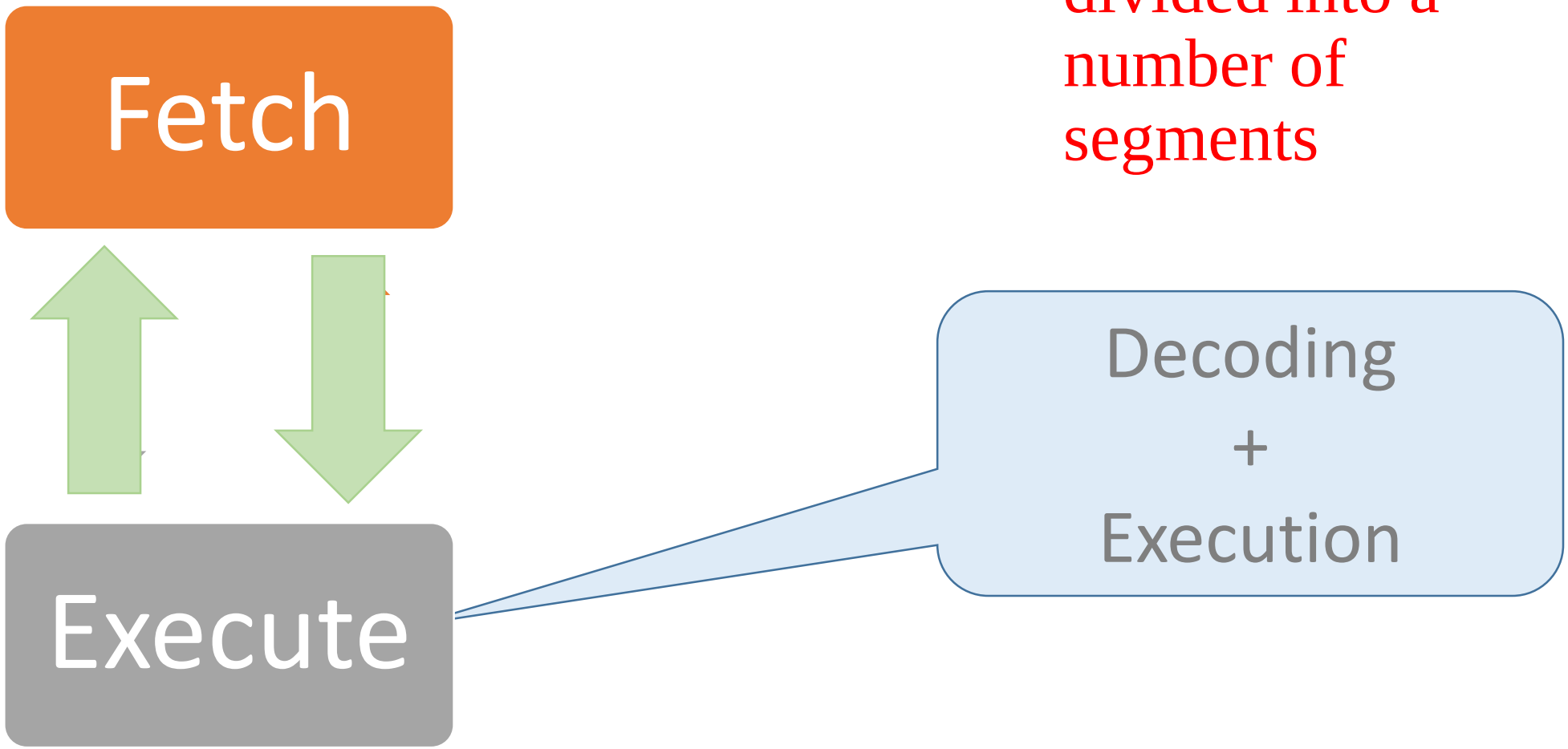
Techniques built into processors

1. Pipelining
2. Branch prediction
3. Data flow analysis
4. Speculative execution

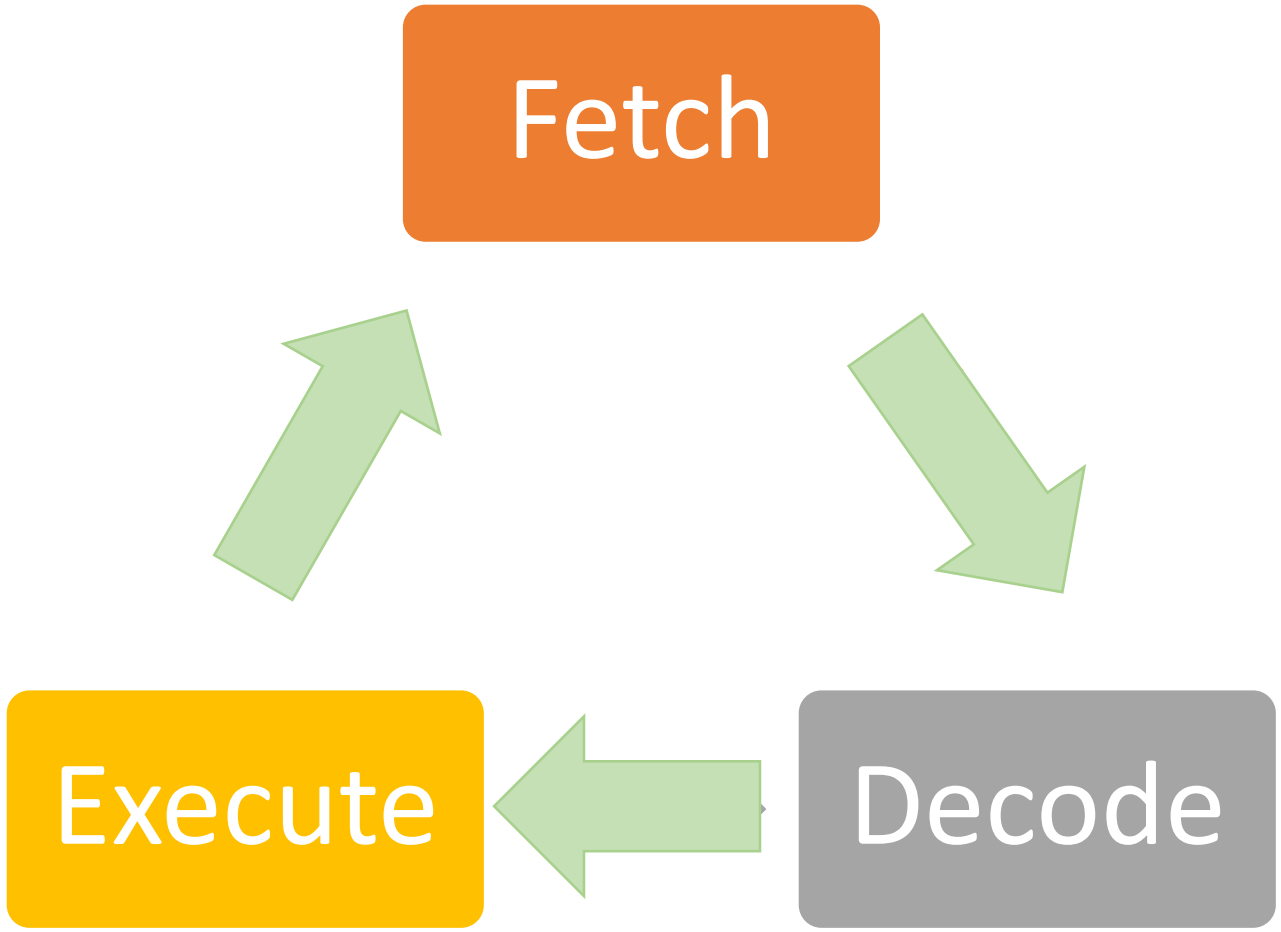
What is instruction cycle?

What is instruction cycle?

Instruction's execution can be divided into a number of segments

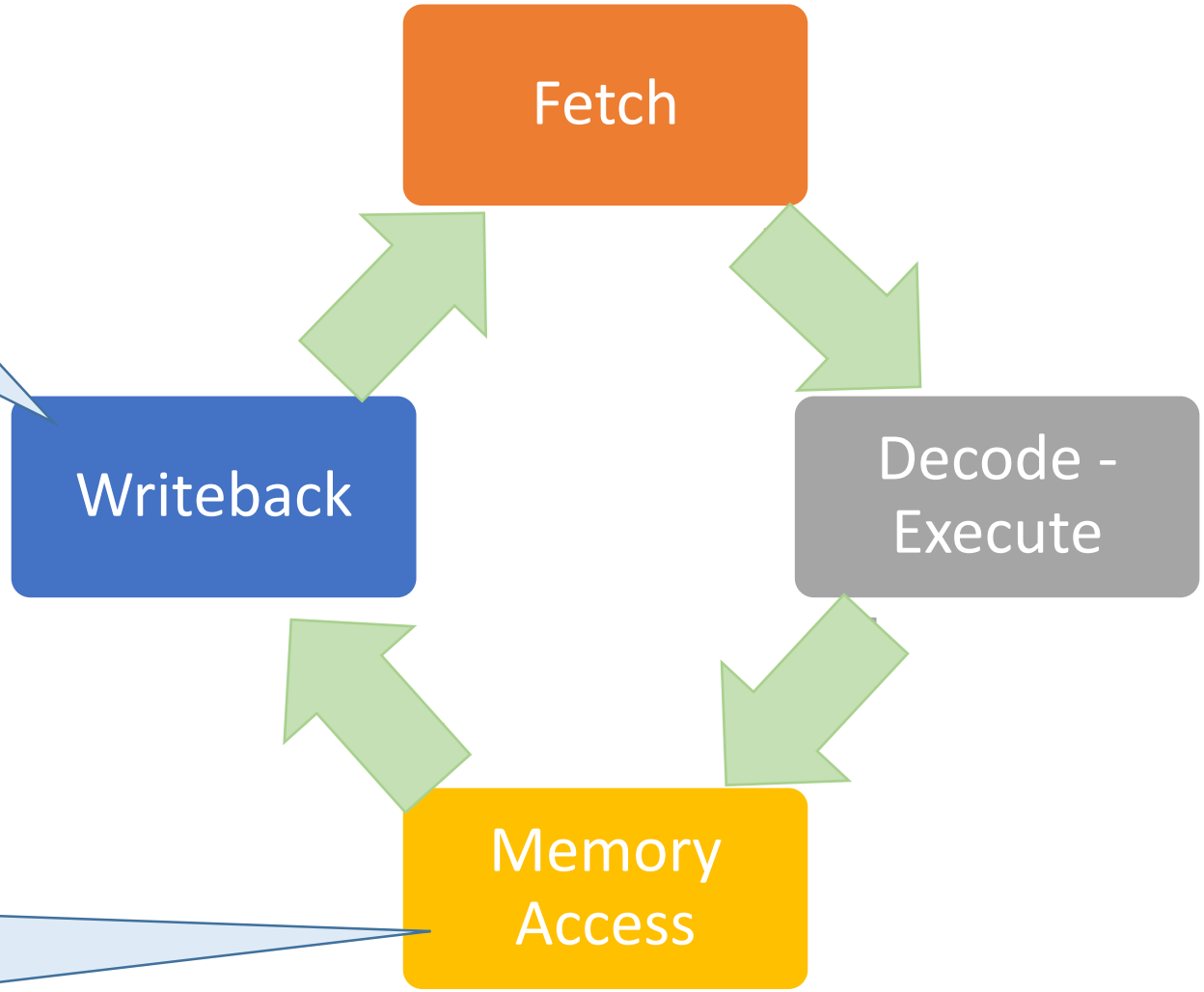


What is instruction cycle?



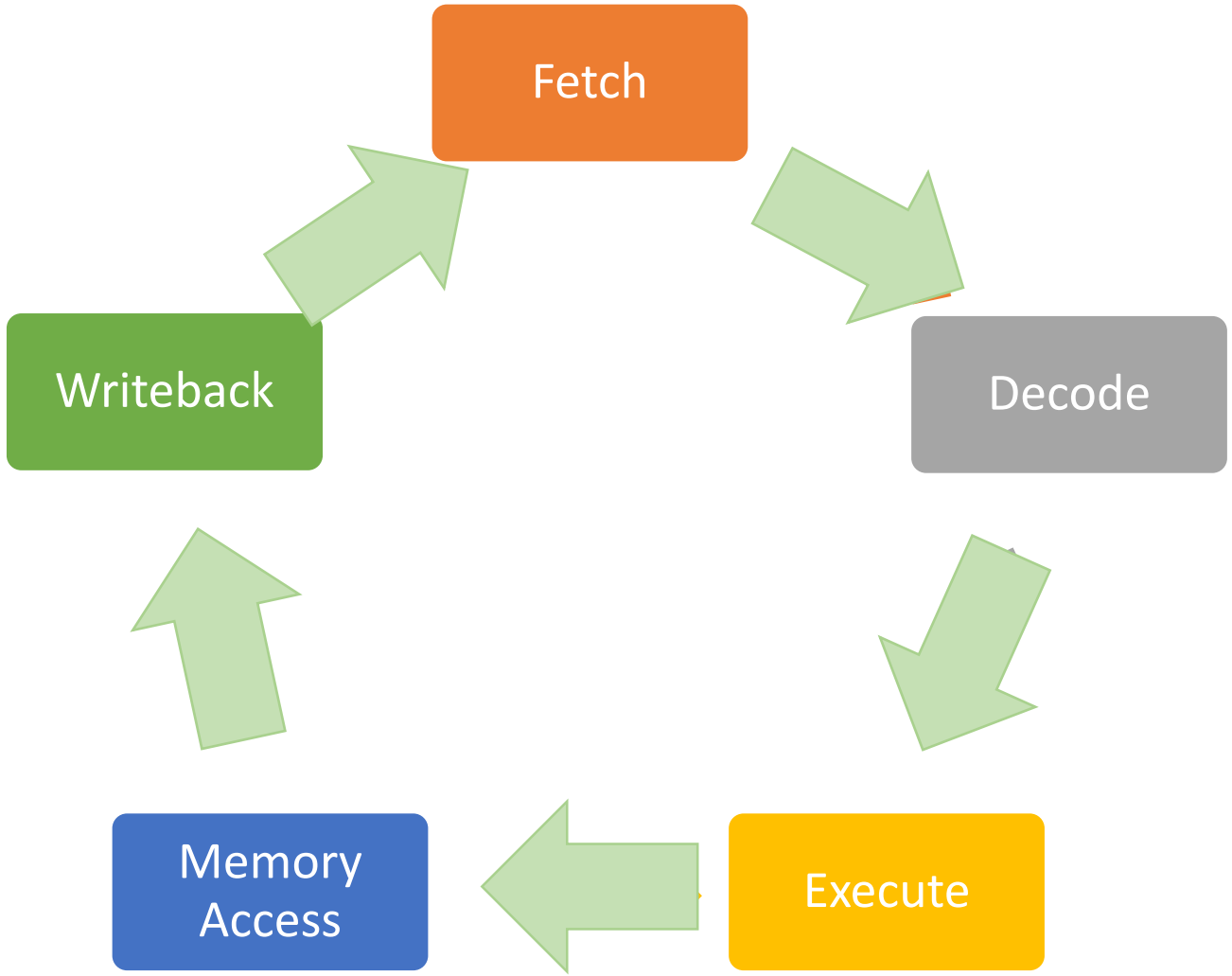
What is instruction cycle?

Results of the instruction written back to registers



If the instruction involves memory operations

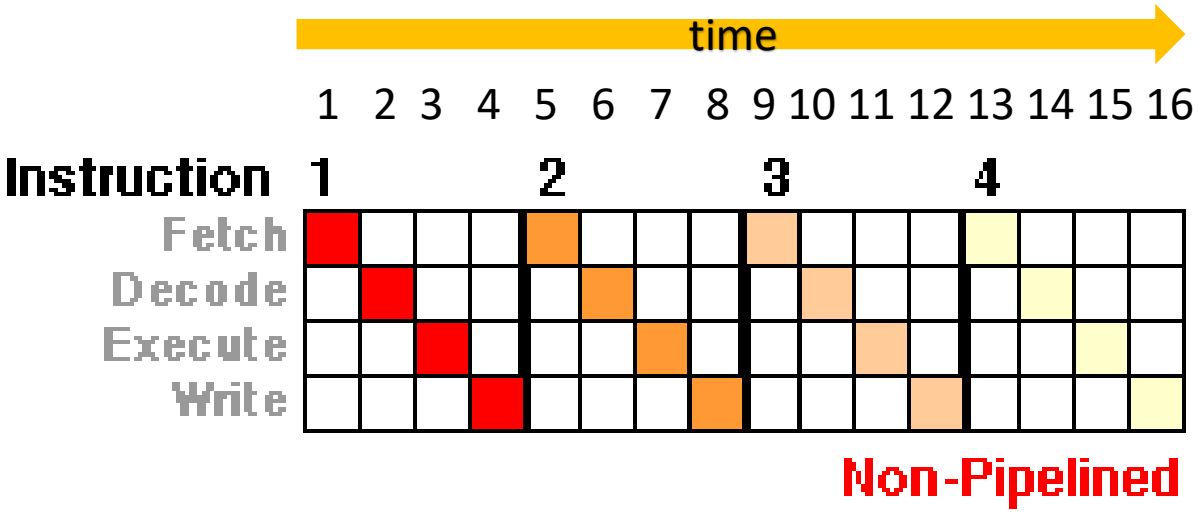
What is instruction cycle?



What is pipelining?

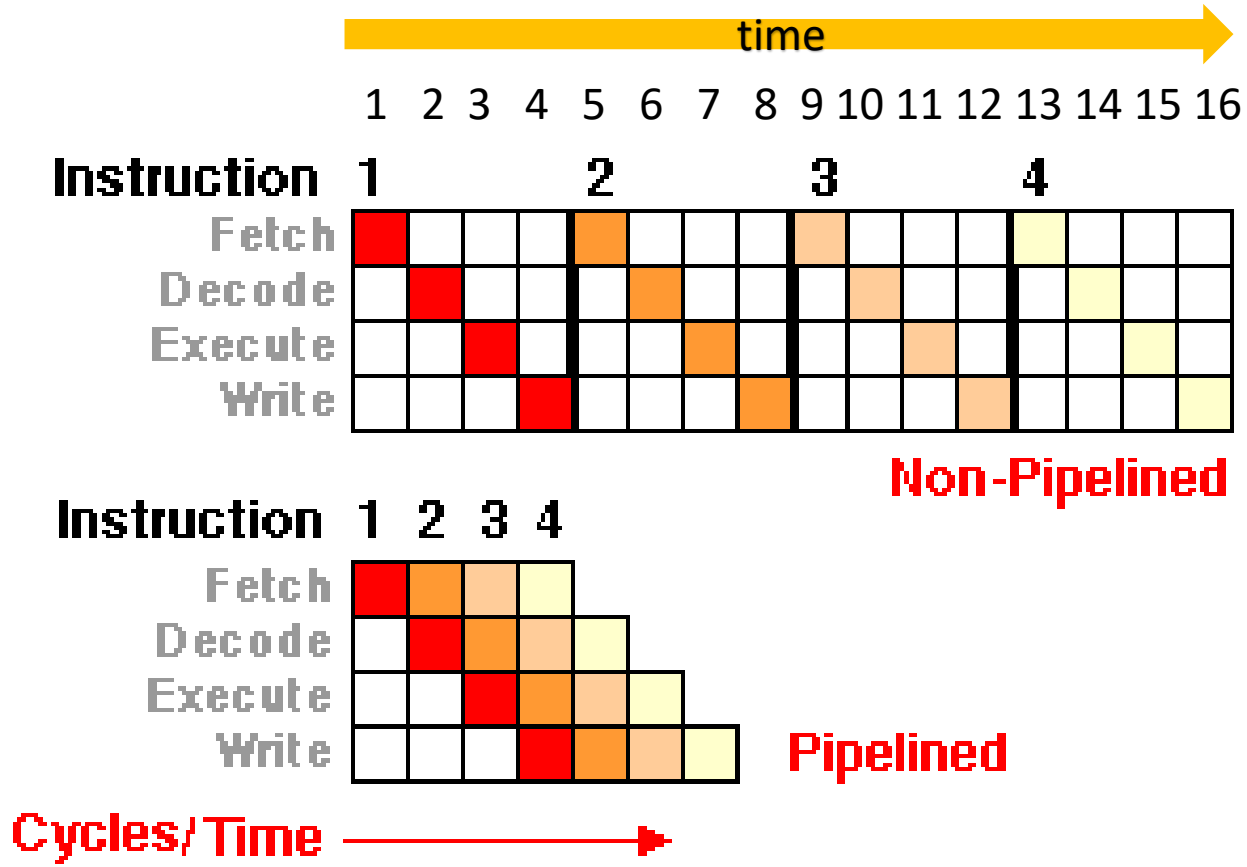
Designing for performance : Pipelining

- Traditional sequential instruction execution



Designing for performance : Pipelining

- A processor can simultaneously work on multiple instructions
- While one instruction is being executed, the computer is decoding the next instruction

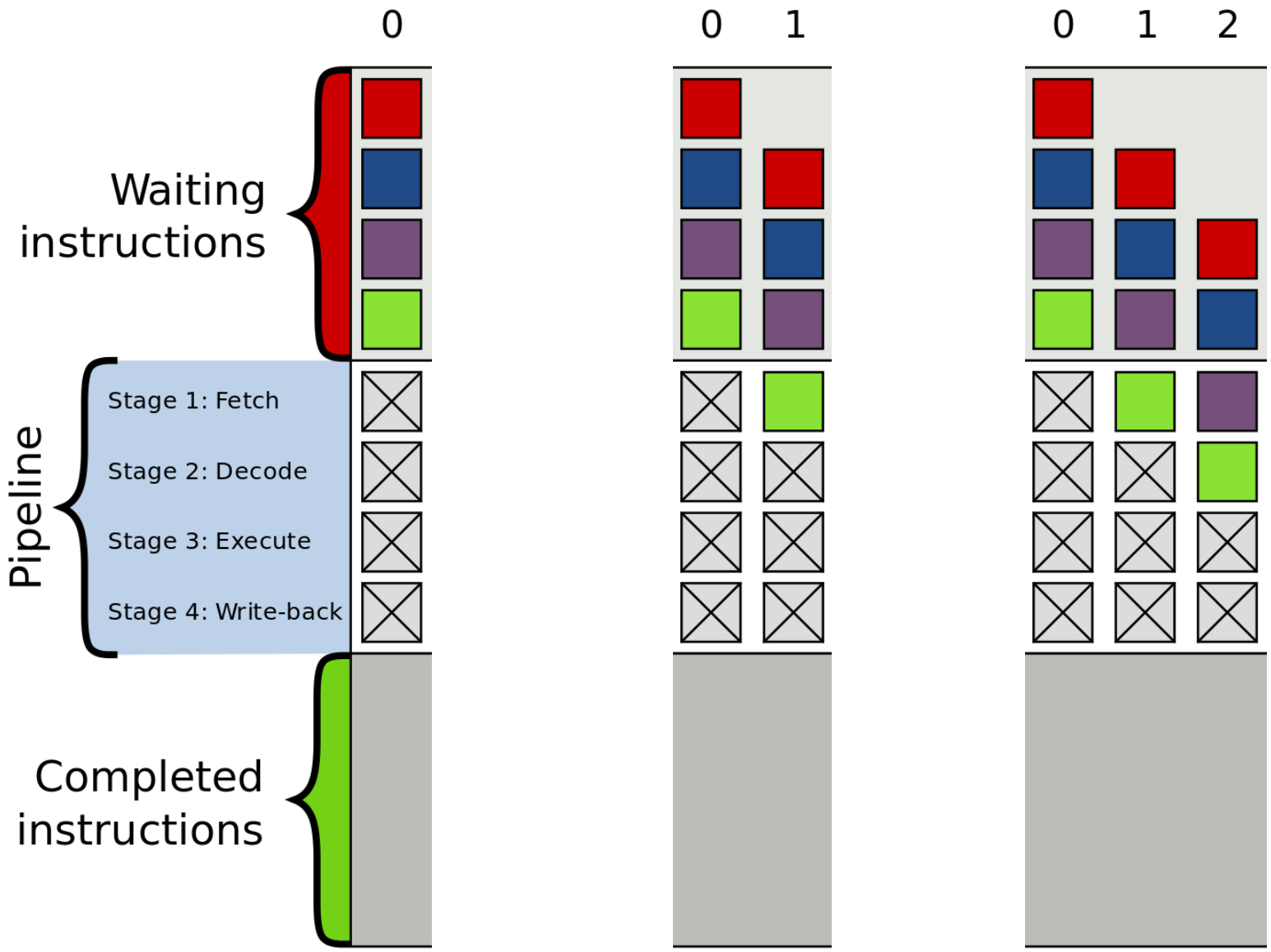


Designing for performance :Pipelining

Consider four instructions

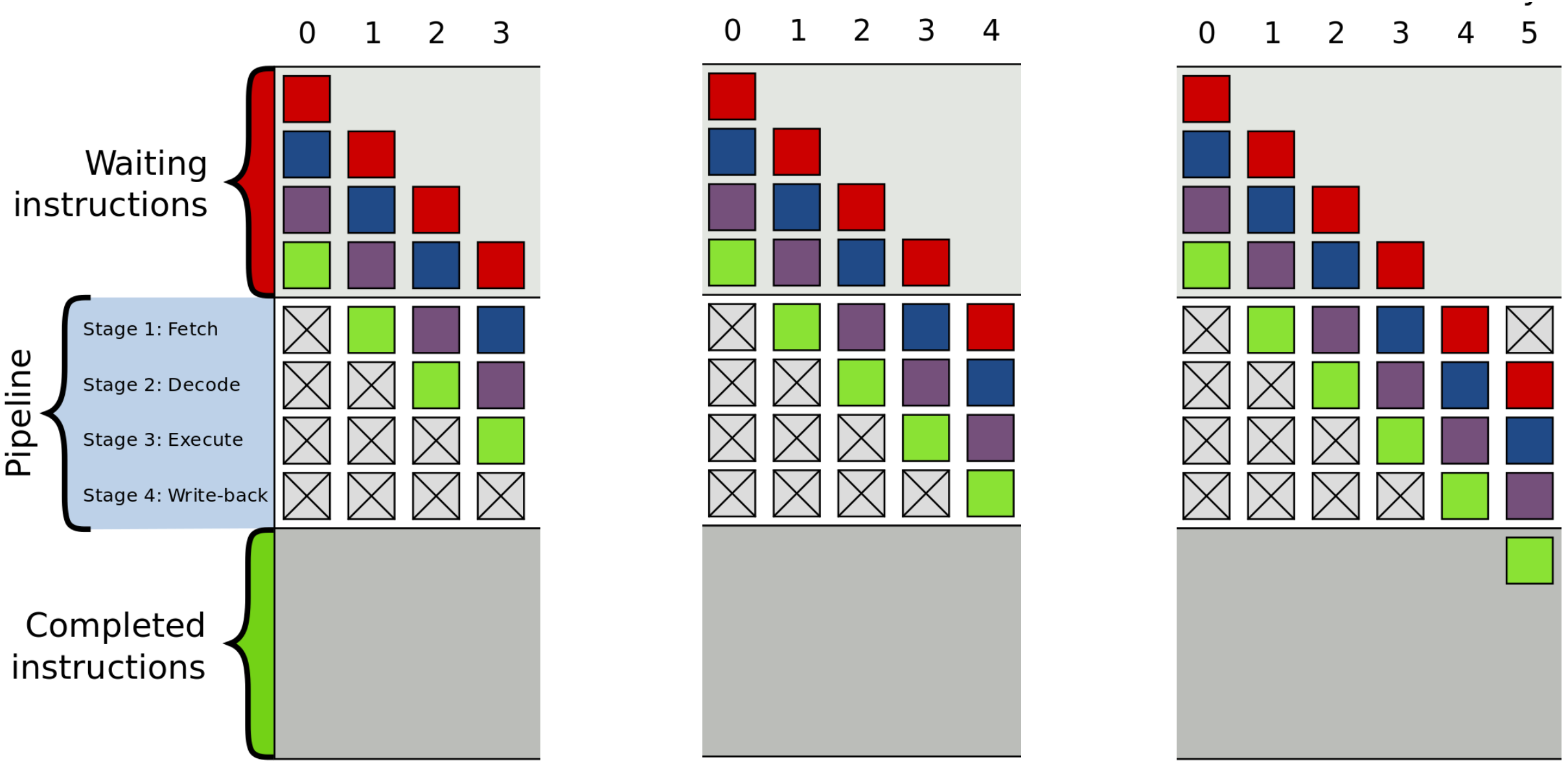
- Green,
- Magenta,
- Blue,
- Red

Four instructions can be executed in pipeline



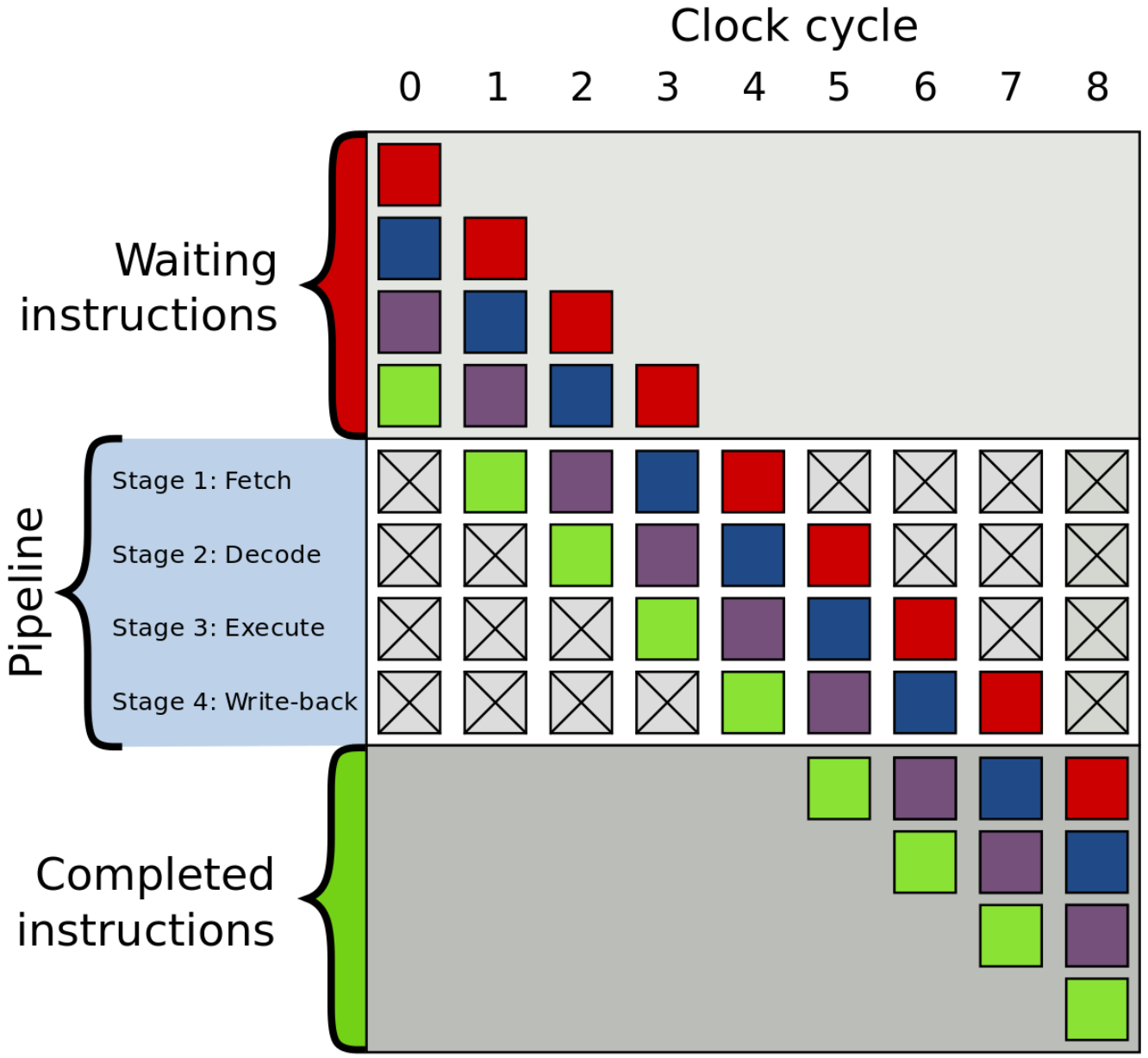
Dr. Subodha Gunawardena

Designing for performance :Pipelining



Designing for performance :Pipelining

Total time to execute is
7 cycles.

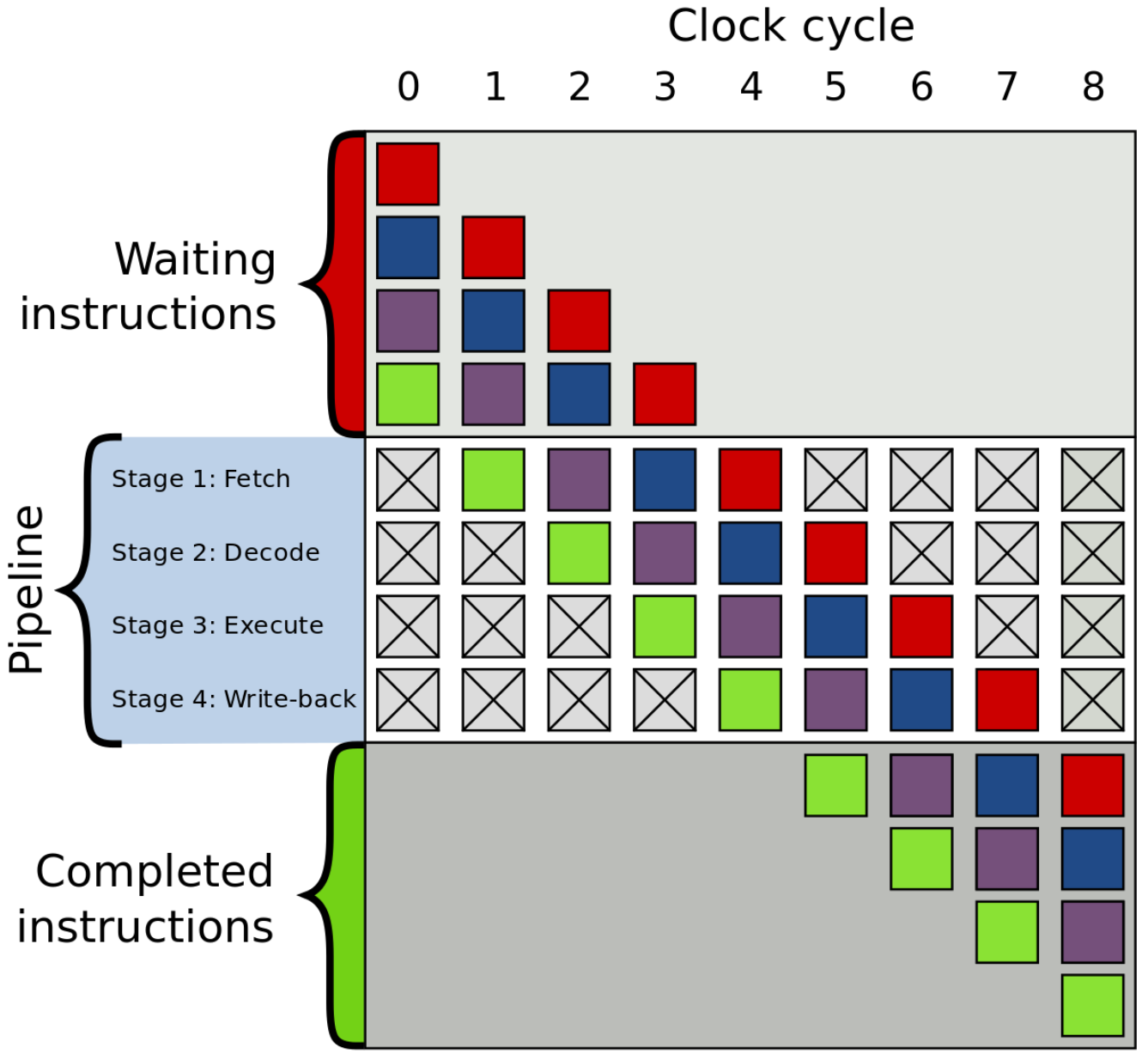


Designing for performance : Branch Prediction

Dr. Subodha Gunawardena

Suppose that the **Green** instruction is a branch instruction and it can either branch to the **Purple** instruction or to **Red** instruction.

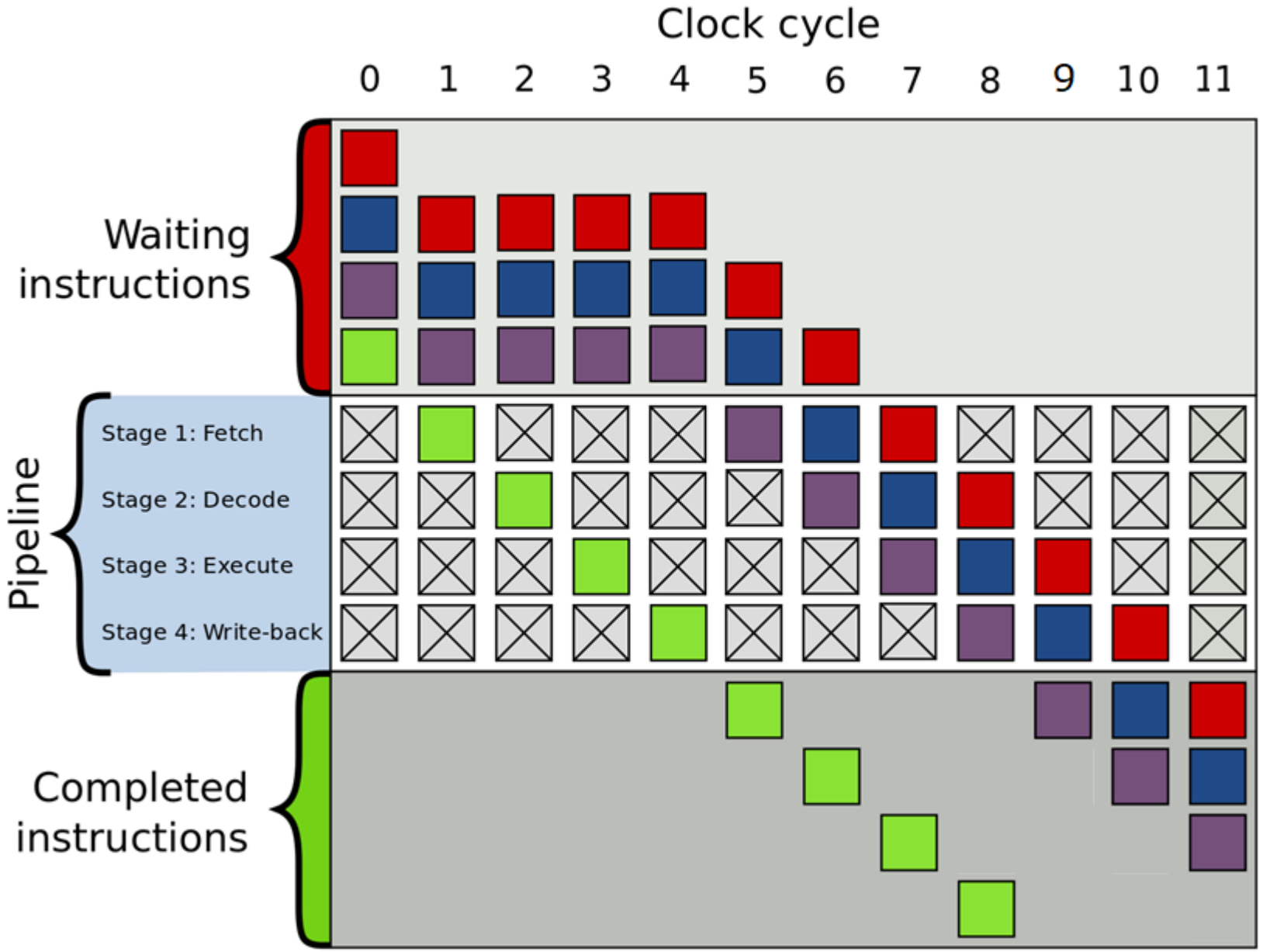
What should the processor do?



Designing for performance : Branch Prediction

If execution of **Green** Instruction result in

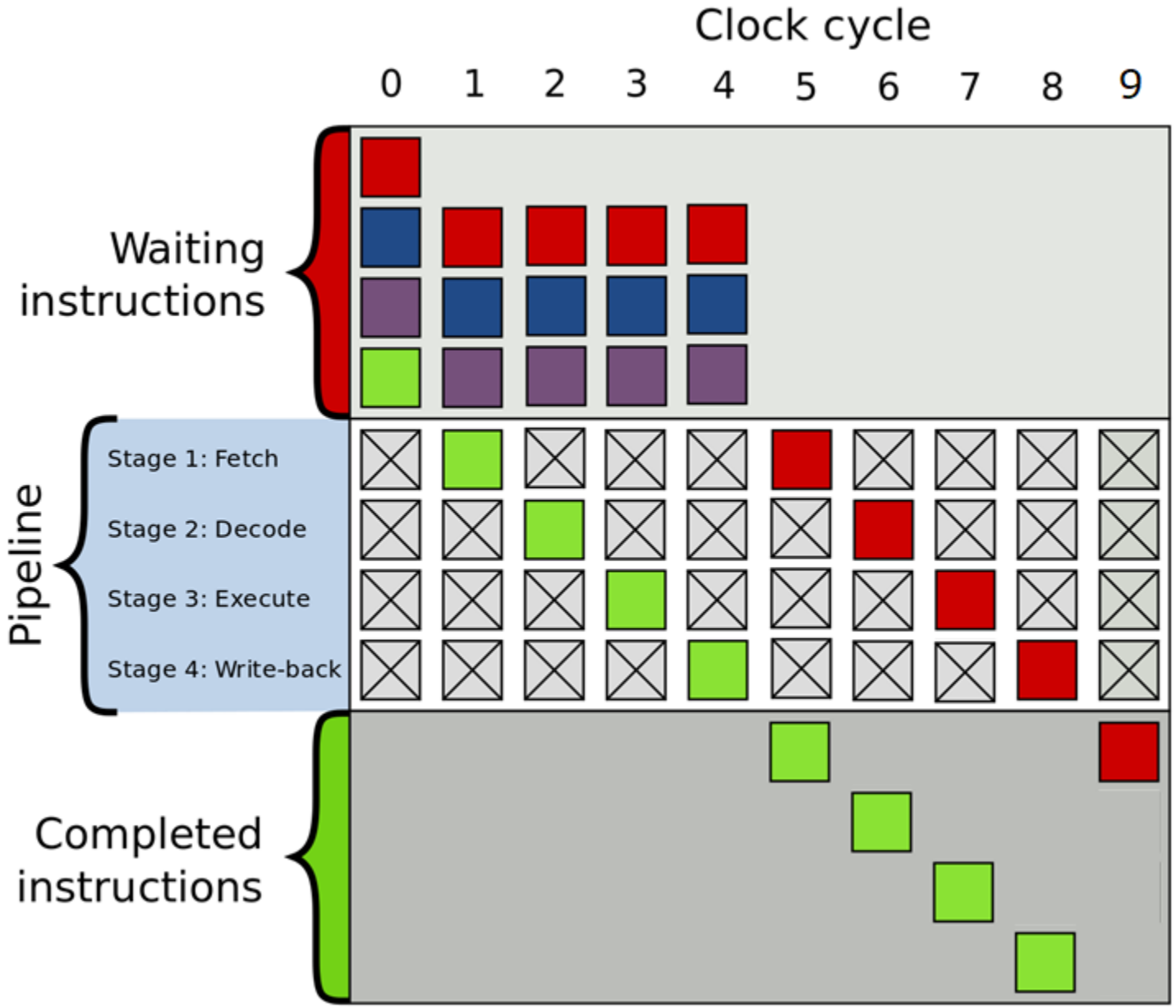
Branch to **Magenta** …



Designing for performance : Branch Prediction

If execution of **Green** Instruction result in

Branch to **Red** branch …



What would be the smart thing to do?

What would be the smart thing to do?

```
X = 0

While ( x <  100)
{
    print (x)
    x = x + 1
}
```


What is branch prediction?

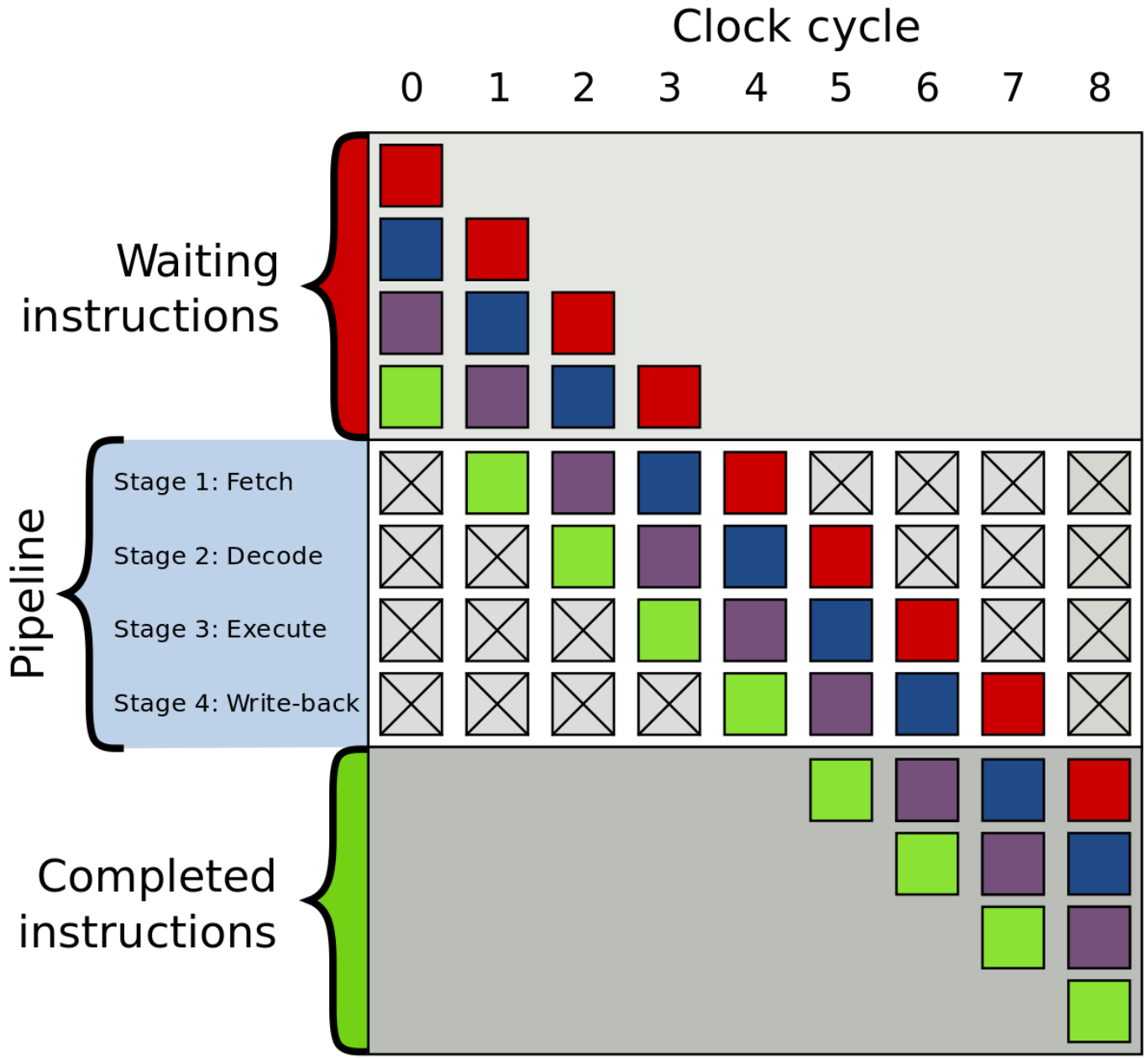
What is branch prediction?

The processor looks ahead in the fetched instructions and predicts which branches, or groups of instructions, are likely to be processed next

With branch prediction, processor predicts the next instruction and keep fetching instructions until the end of the execution of branch instruction

This is **Out-of-order execution**

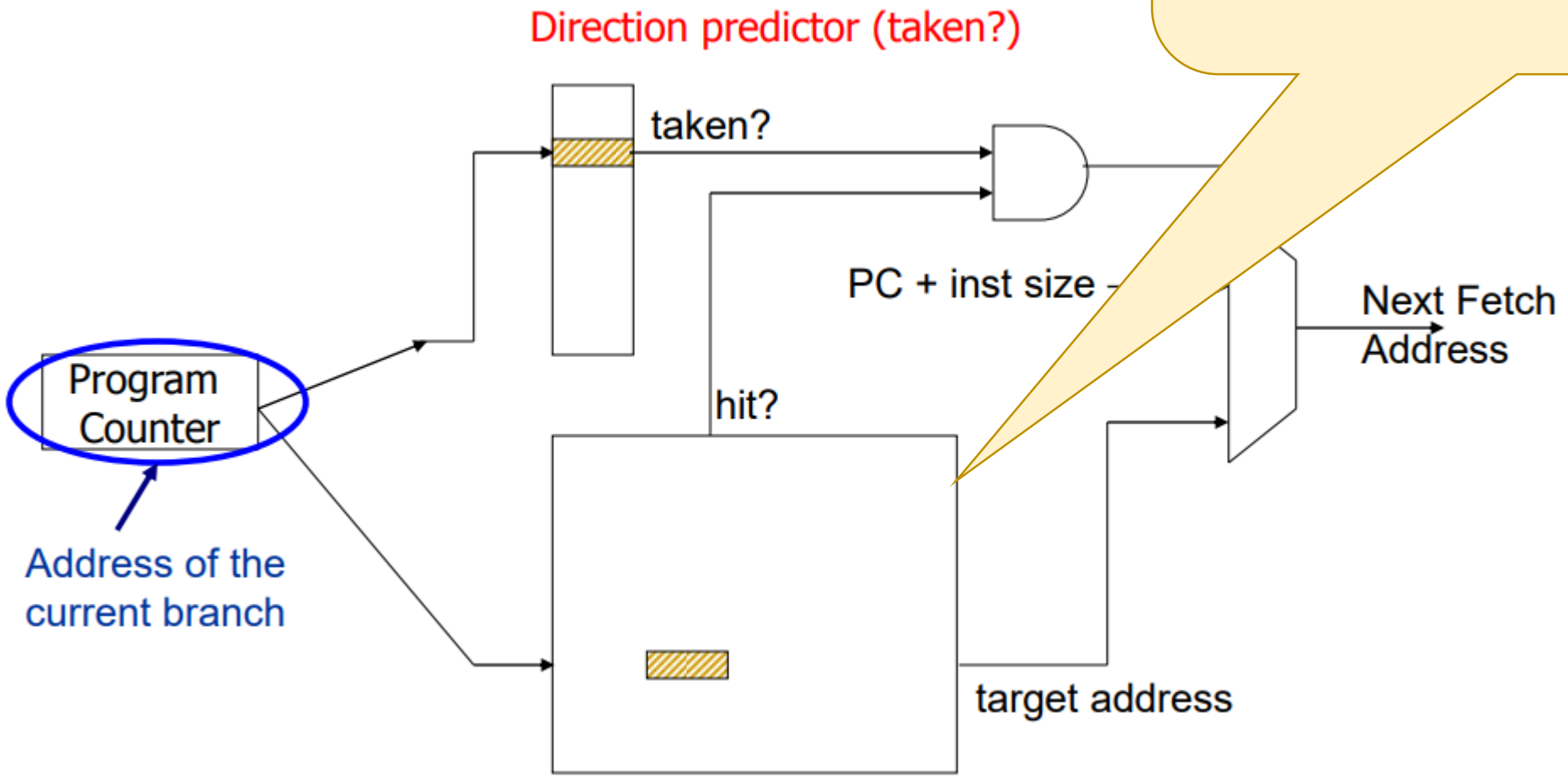
Designing for performance : Branch Prediction



What is branch prediction buffer?

Branch Prediction Using BTB

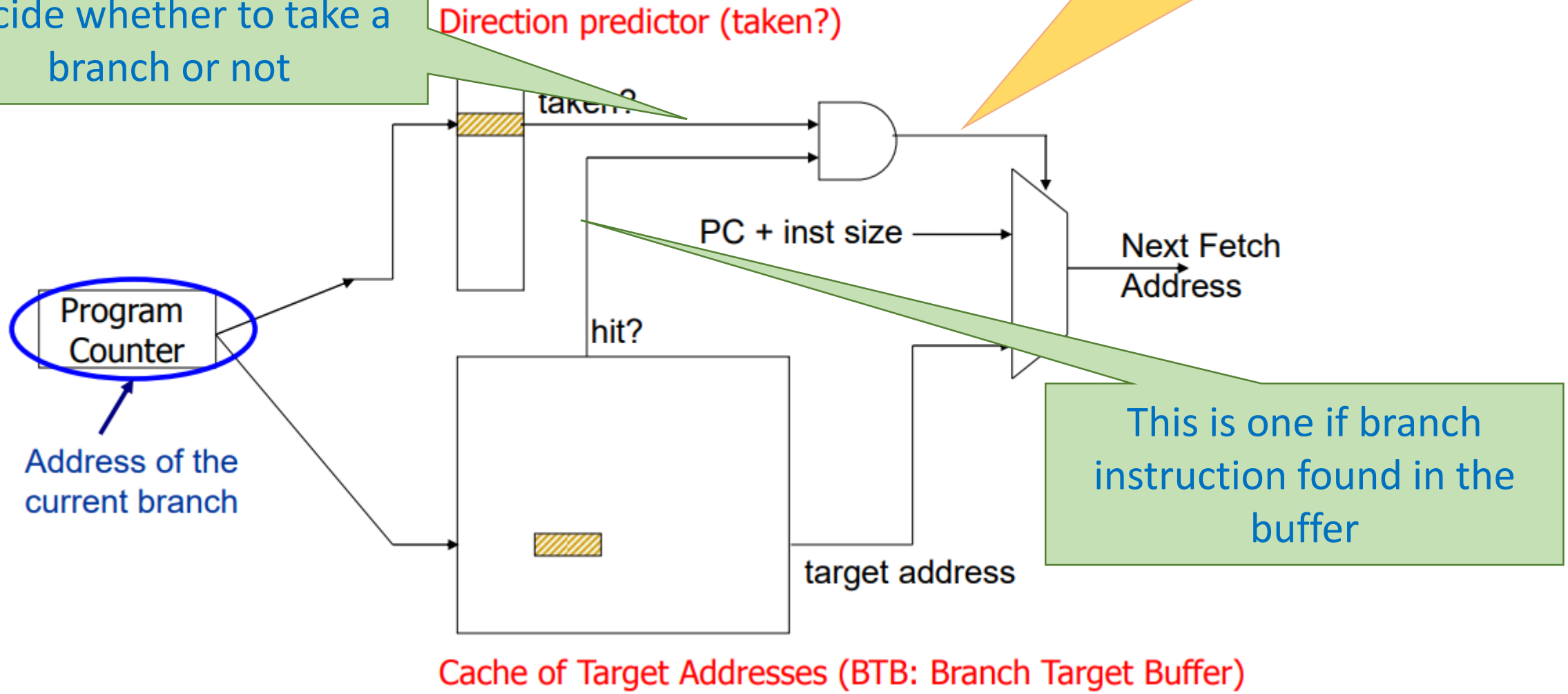
BTB records the results of the branch instructions taken



Branch Prediction Using BTB

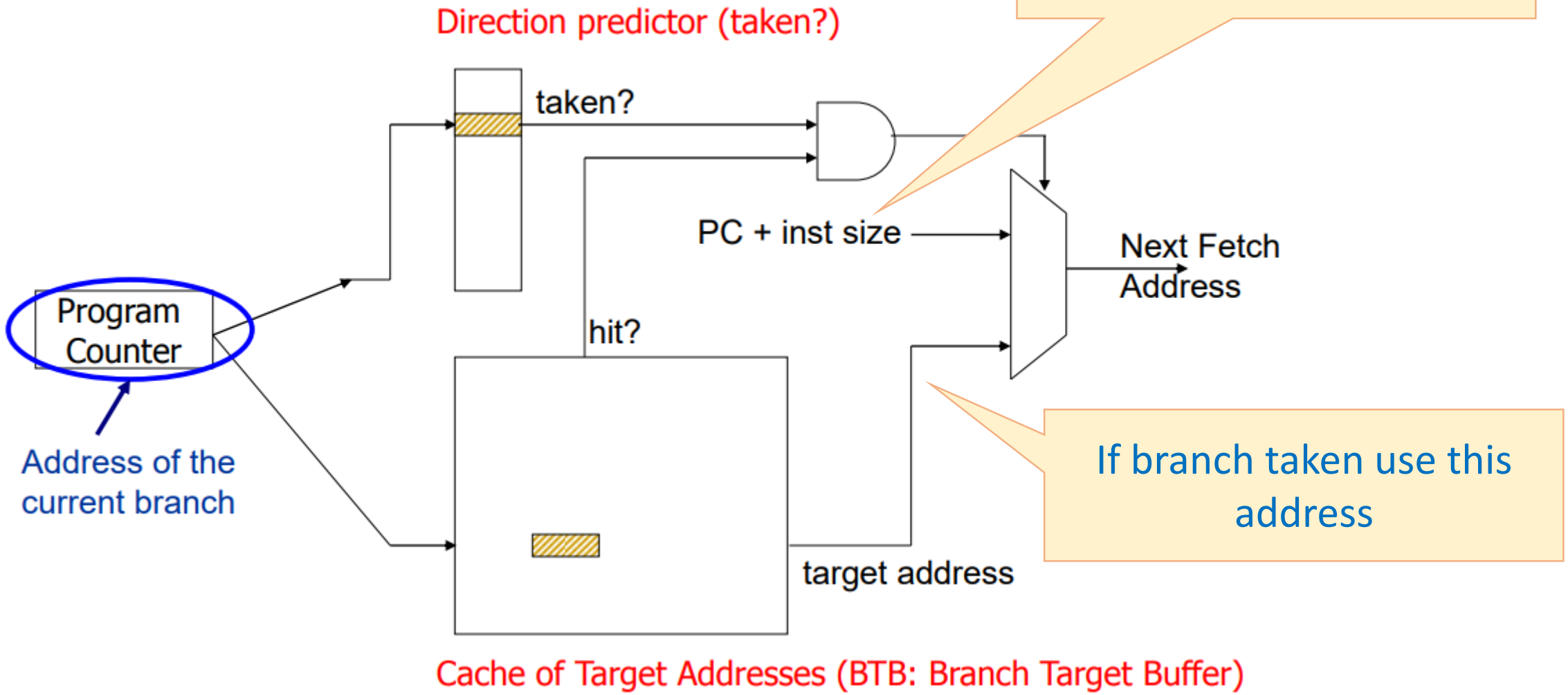
Different algorithms to decide whether to take a branch or not

This result decides whether to take the branch or not



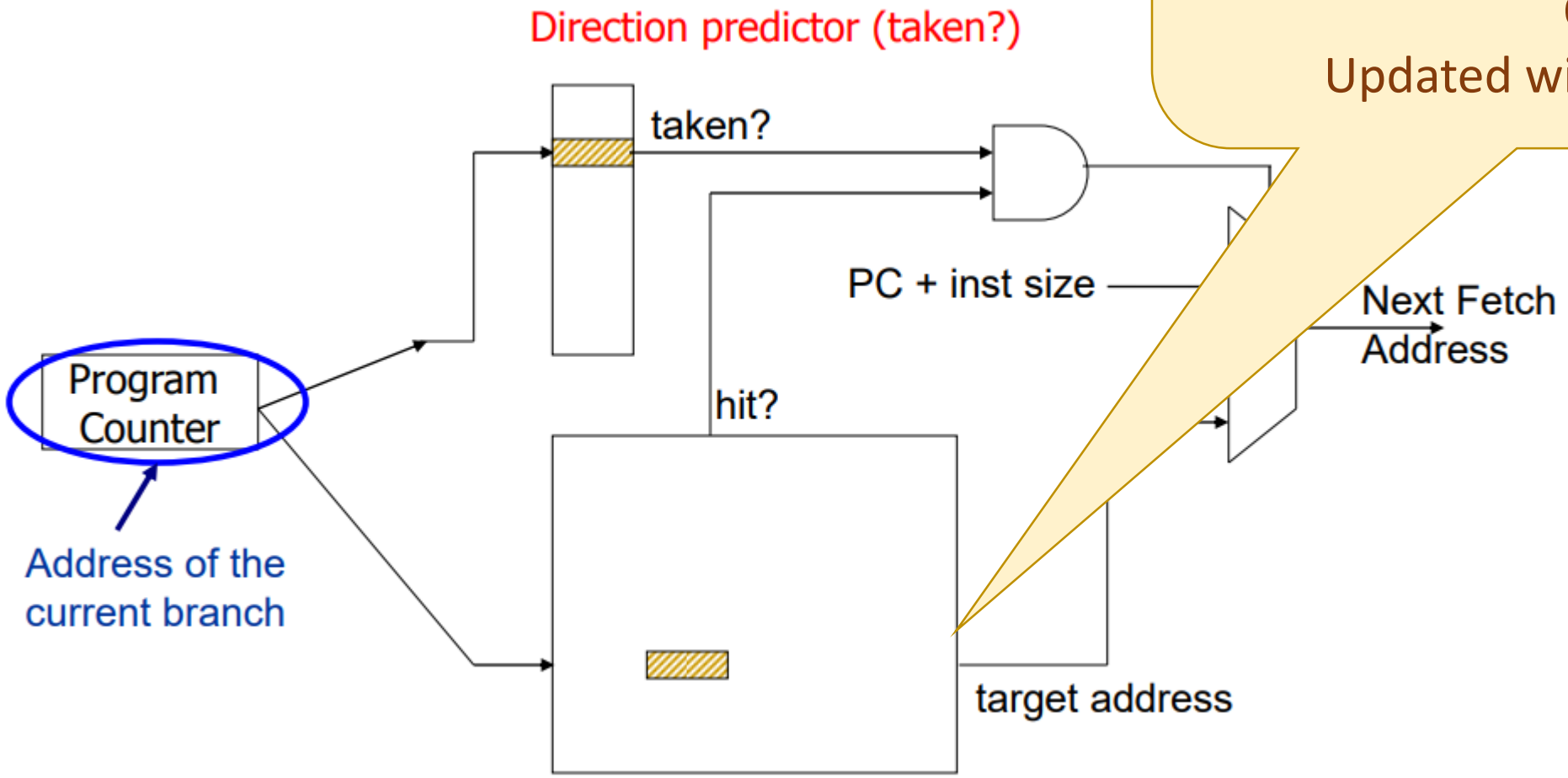
This is one if branch instruction found in the buffer

Branch Prediction Using BTB



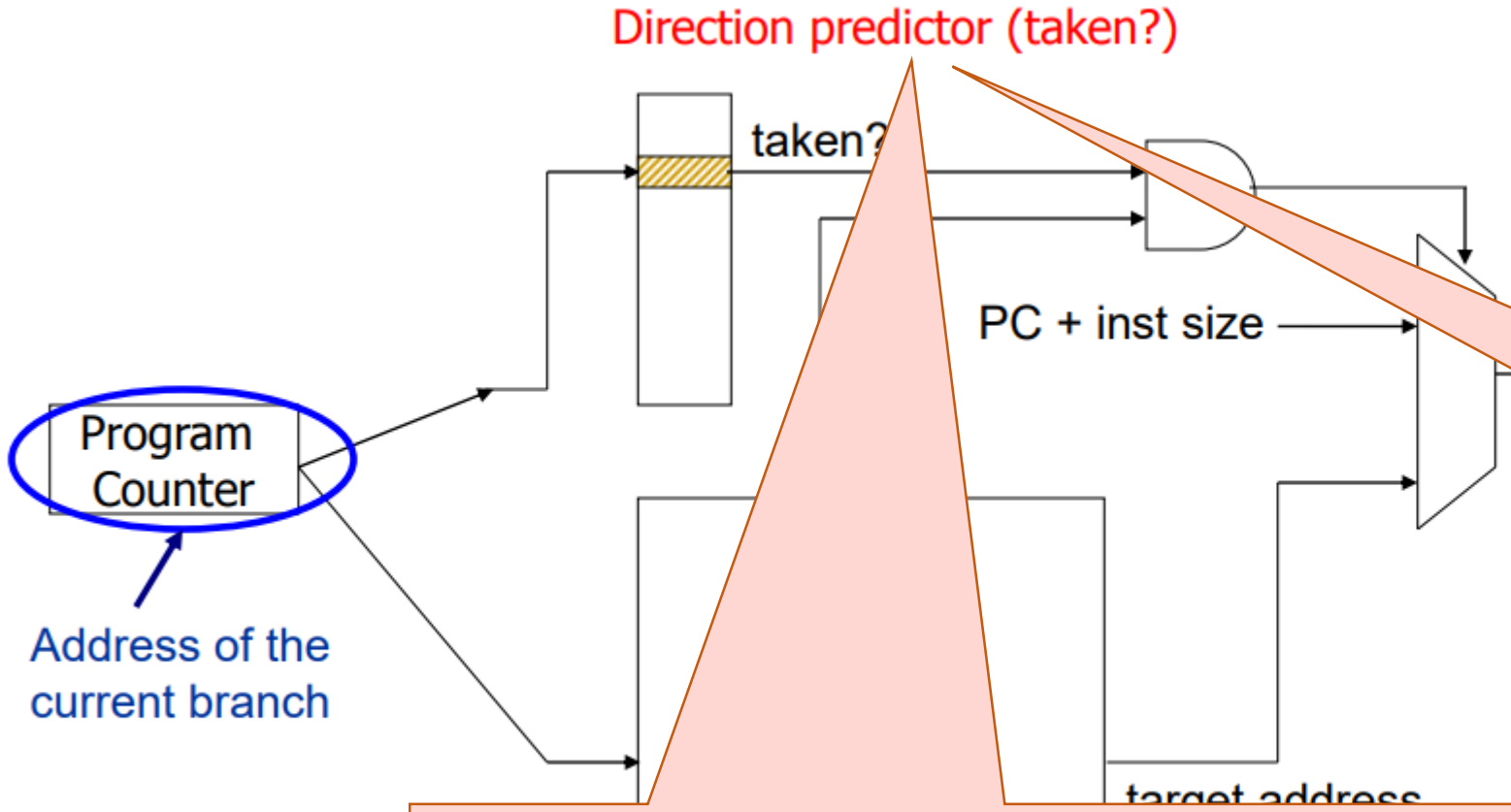
Branch Prediction Using BTB

If the predicted branch is unsuccessful,
Branch instruction record is removed from the BTB
Or
Updated with new value



What are the branch prediction algorithms ?

Branch Prediction Using BTB



Compile time (static)

1. Always not taken
2. Always taken
3. BTFN (Backward taken, forward not taken)
4. Profile based (likely direction)

Run time (dynamic)

Last time prediction (single-bit)

There are more sophisticated prediction algorithms

https://course.ece.cmu.edu/~ece740/f15/lib/exe/fetch.php?media=18-740-fall15-lecture05-branch-prediction-afterlecture.pdf

What is speculative execution ?

Designing for performance : Speculative execution

- Closely related to branch prediction

- Executed the instructions along the predicted branch

What is data flow analysis?

Designing for performance : Speculative execution

- processor analyses which instructions are dependent on each other's results
- Create an optimized schedule for execution
- Instructions are executed in the prepared order irrespective of the original order in the program,

Designing for performance : Exercise

Suppose that,

A pipeline has 20 Stages (N = 20)

The Processor fetches 5 instructions at a time (W = 5, also called 5 wide fetch)

In addition, 1 out of 5 instructions is a branch instruction (Assume that Each 5 instruction-block ends with a branch. That is the 5^{th} instruction is always a branch instruction)

How long does it take to fetch 500 instructions, if the branch prediction is 100% ?

As each fetch cycle fetches 5 instructions, in order to correctly fetch 500 instructions, it would take 500/5 = 100 cycles.

As there are no errors in prediction, cost due to errors = 0

Therefore, it takes 100 cycles to fetch 500 instructions.

Designing for performance : Exercise

Suppose that,

A pipeline has 20 Stages (N = 20)

The Processor fetches 5 instructions at a time (W = 5, also called 5 wide fetch)

In addition, 1 out of 5 instructions is a branch instruction (Assume that Each 5 instruction-block ends with a branch. That is the 5^{th} instruction is always a branch instruction)

How long does it take to **fetch** 500 instructions, if the branch prediction is 90% ?

As each fetch cycle fetches 5 instructions, in order to correctly **fetch** 500 instructions, it would take 500/5 = 100 cycles.

As there are 10% errors in prediction of branch instructions,

Number of **fetch** cycles in error = 10

Duration it takes to wait until branch instruction finish the pipeline = (20 – 1) cycles

Therefore, it takes 100 + 10 x (20-1) = 290 cycles to fetch 500 instructions.

Designing for performance

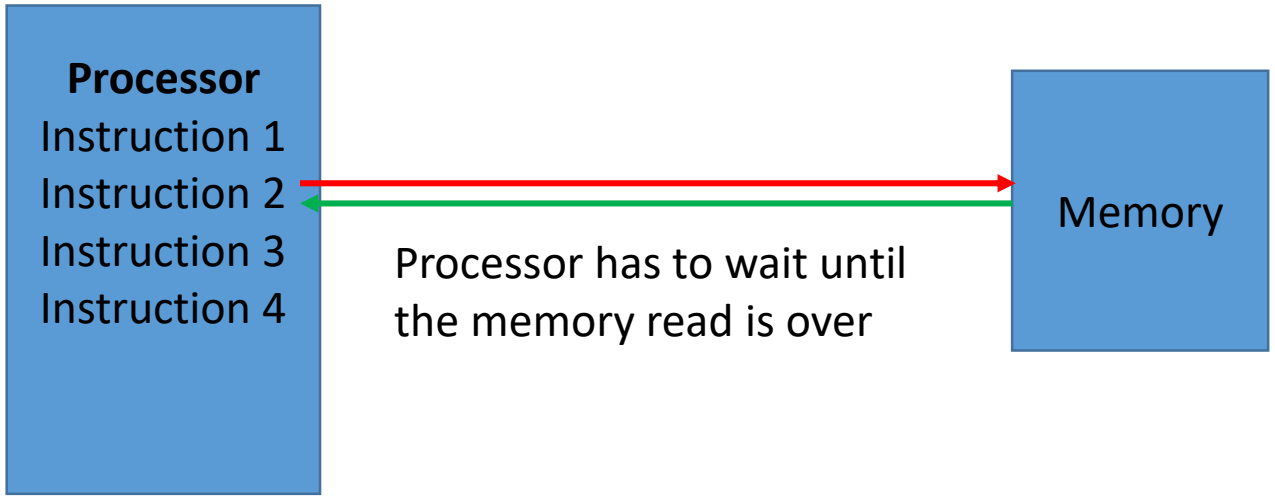
Performance Balance

- The processor speed has grown rapidly,

  However, **data transfer speed** between main memory and the processor has lagged

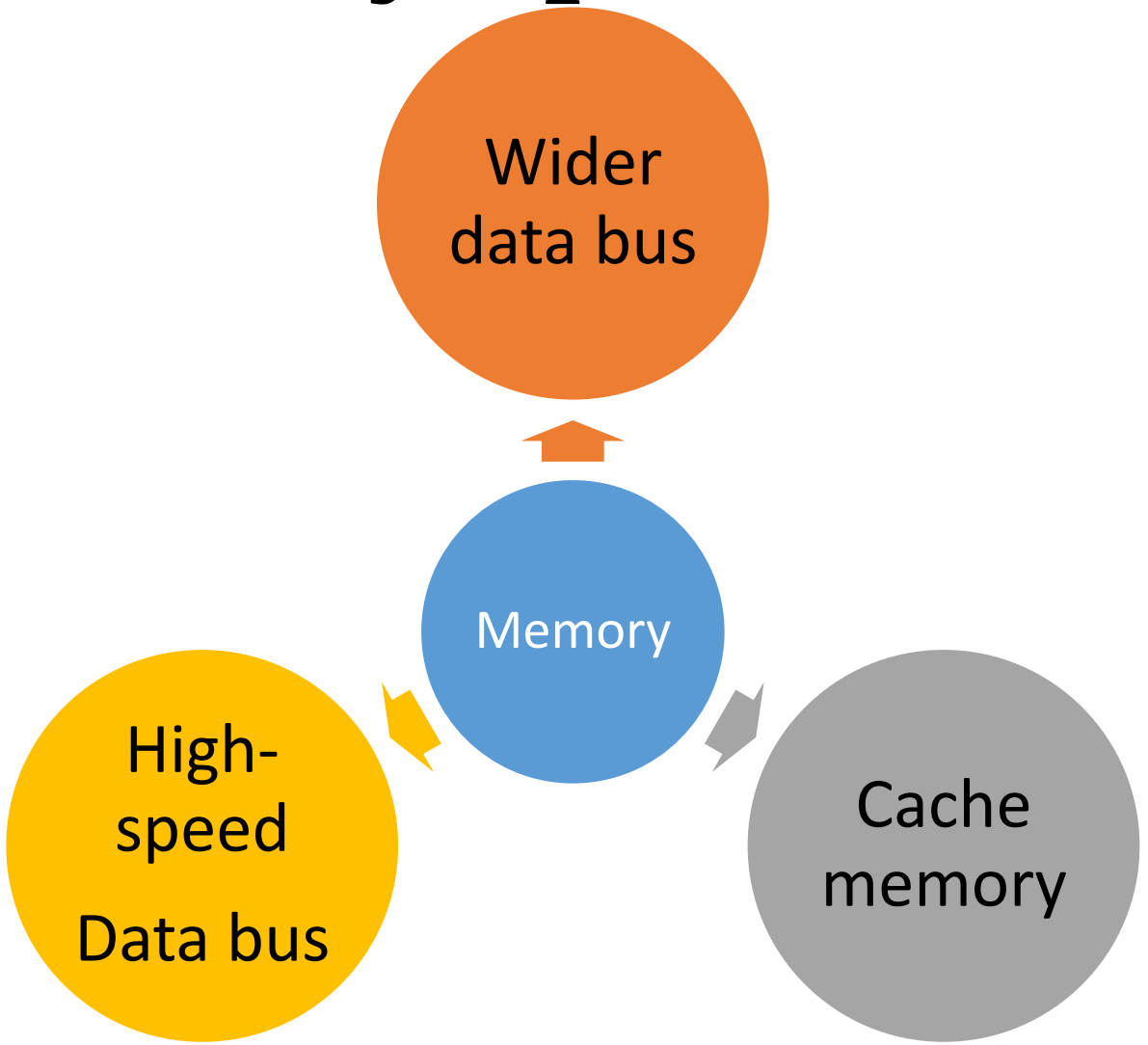
- Interface between processor and main memory is the most crucial pathway

- If memory or the pathway fails to keep pace with the processor's insistent demands, the processor stalls in a wait state



What are the solutions for processor speed and memory speed imbalance?

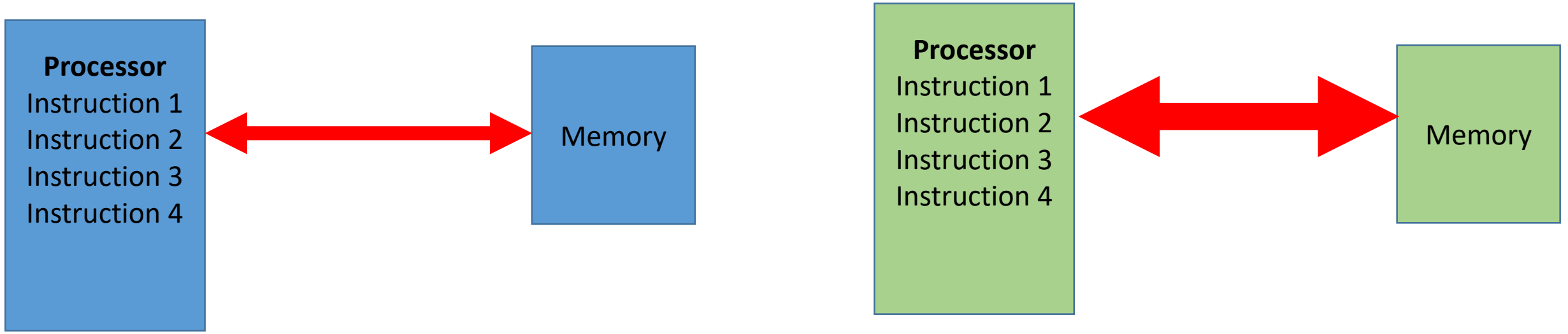
What are the solutions for processor speed and memory speed imbalance?



Designing for performance : Memory Access Speed

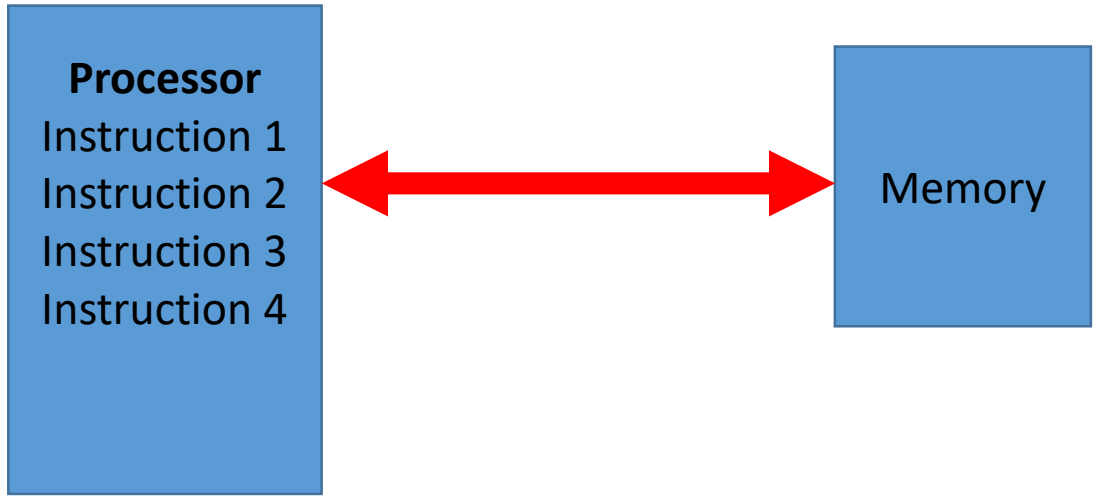
- **Wider RAMs**:

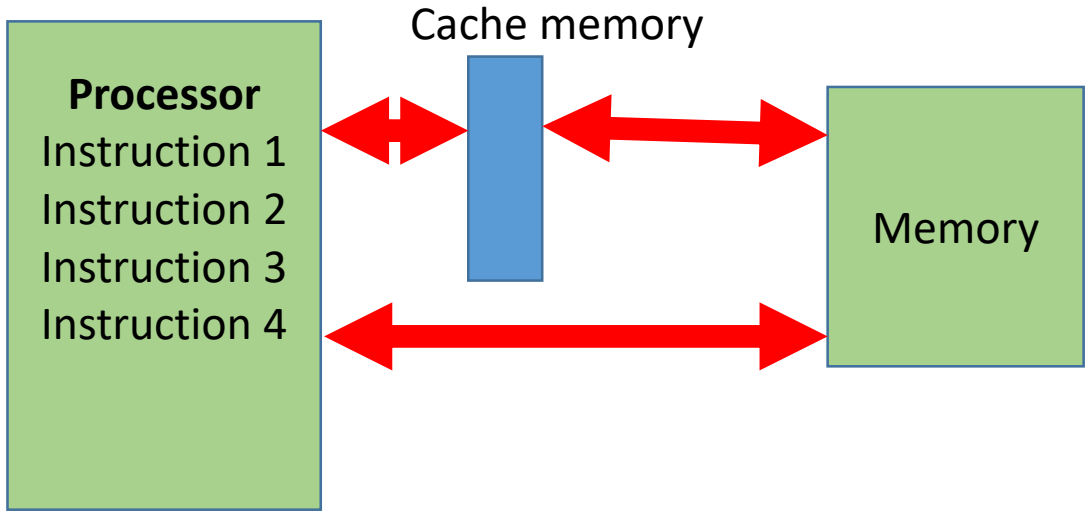
Increase the number of bits that are retrieved at one time by making DRAMs "wider" rather than "deeper" and by using wide bus data paths



Designing for performance : Memory Access Speed

- **Cache Memory**: Reduce the frequency of memory access by incorporating increasingly complex and efficient cache structures between the processor and main memory





Designing for performance : Memory Access Speed

A system architect can address this problem in a number of ways

- **Wider RAMs**: Increase the number of bits that are retrieved at one time by making DRAMs “wider” rather than “deeper” and by using wide bus data paths
- **Cache Memory**: Reduce the frequency of memory access by incorporating increasingly complex and efficient cache structures between the processor and main memory
- **High-speed data busses**: Increase the interconnect bandwidth between processors and memory by using higher-speed buses

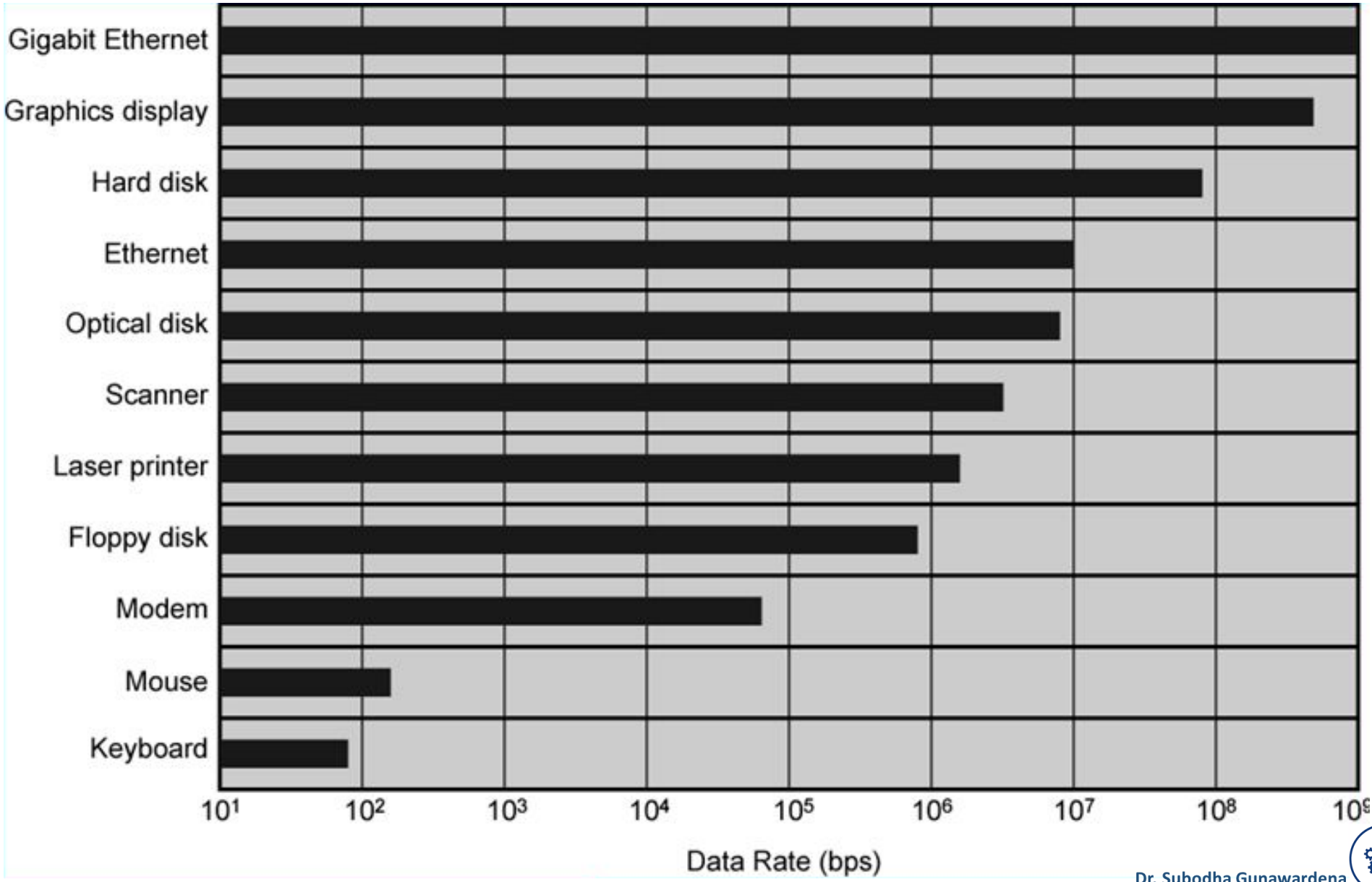
Designing for performance : I/O Speeds

Another area of design focus is the handling of I/O devices

- The problem is getting that data moved between processor and peripheral
- Strategies here include caching and buffering schemes plus the use of higher-speed interconnection buses

Designing for performance : I/O Speeds

Typical I/O device data rates



Designing for performance

Designers constantly strive to balance.

- Throughput and processing demands of the processor components
- Main memory
- I/O devices
- Interconnection structures

Designing for performance : Throughput

Improvements in Chip Organization and Architecture for processor speed

- Increase the **hardware speed** of the processor,
  more gates can be packed together more tightly and to increasing the clock rate
- Increase the **size and speed of caches** between the processor and main memory
- Make changes to the processor organization and architecture that increase the effective speed of instruction execution **using parallelism**

What are challenges to increase the processor speed?

Designing for performance : Clock Speed

Obstacles to increase the clock speed

Power

The difficulty of dissipating the heat generated on high-density, high-speed chips is becoming a serious design issue

RC delay

Components on the chip decrease in size, the wire interconnects become thinner, increasing resistance.

When the wires are closer together, increasing capacitance.

Increase of R and C delays the electron flow between transistors

Memory latency

Speed which data can be transferred between main memory and the processor has lagged

Designing for performance : Processor Trends

To prevent a further rise in power

Designers developed multicore computer chip

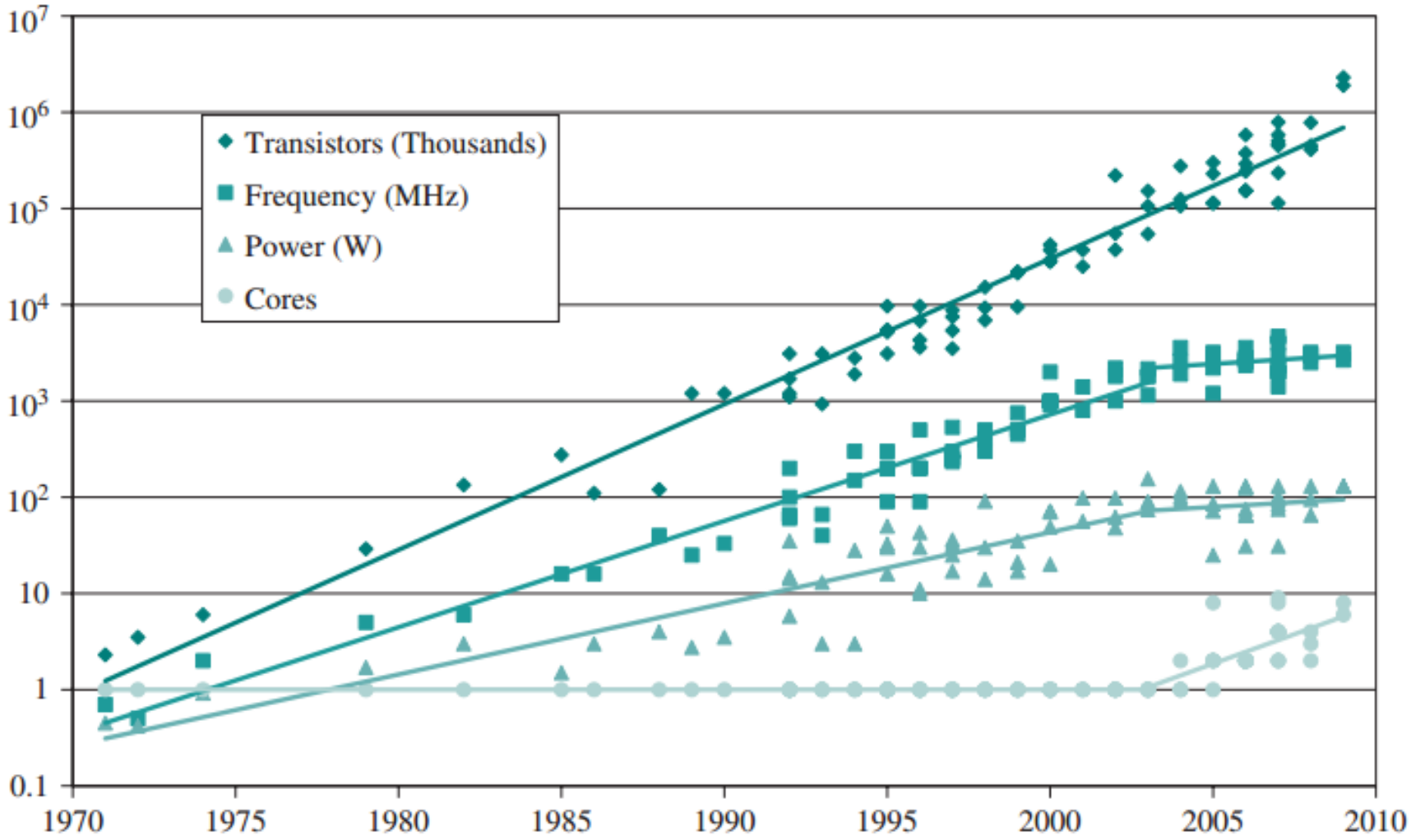
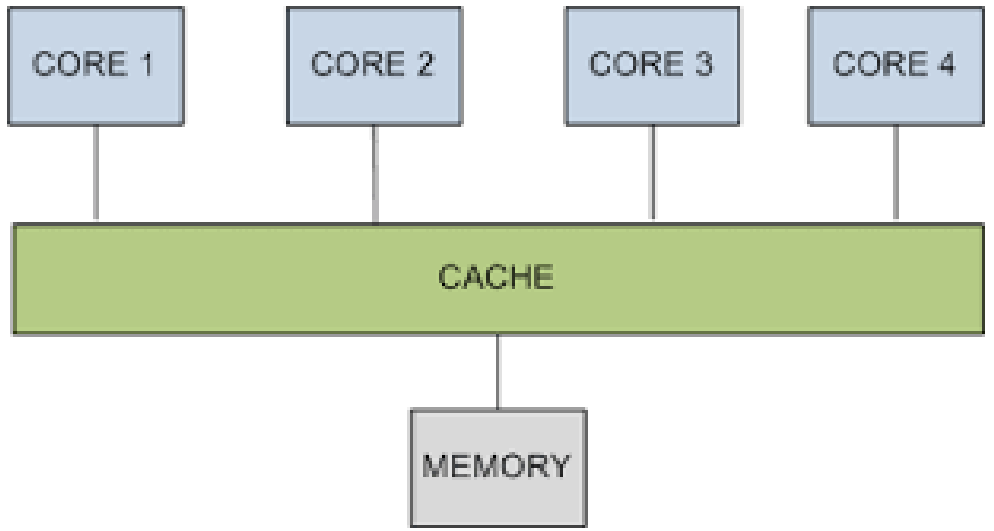


Figure 2.11 Processor Trends

Multicore

Multiple cores or multicore

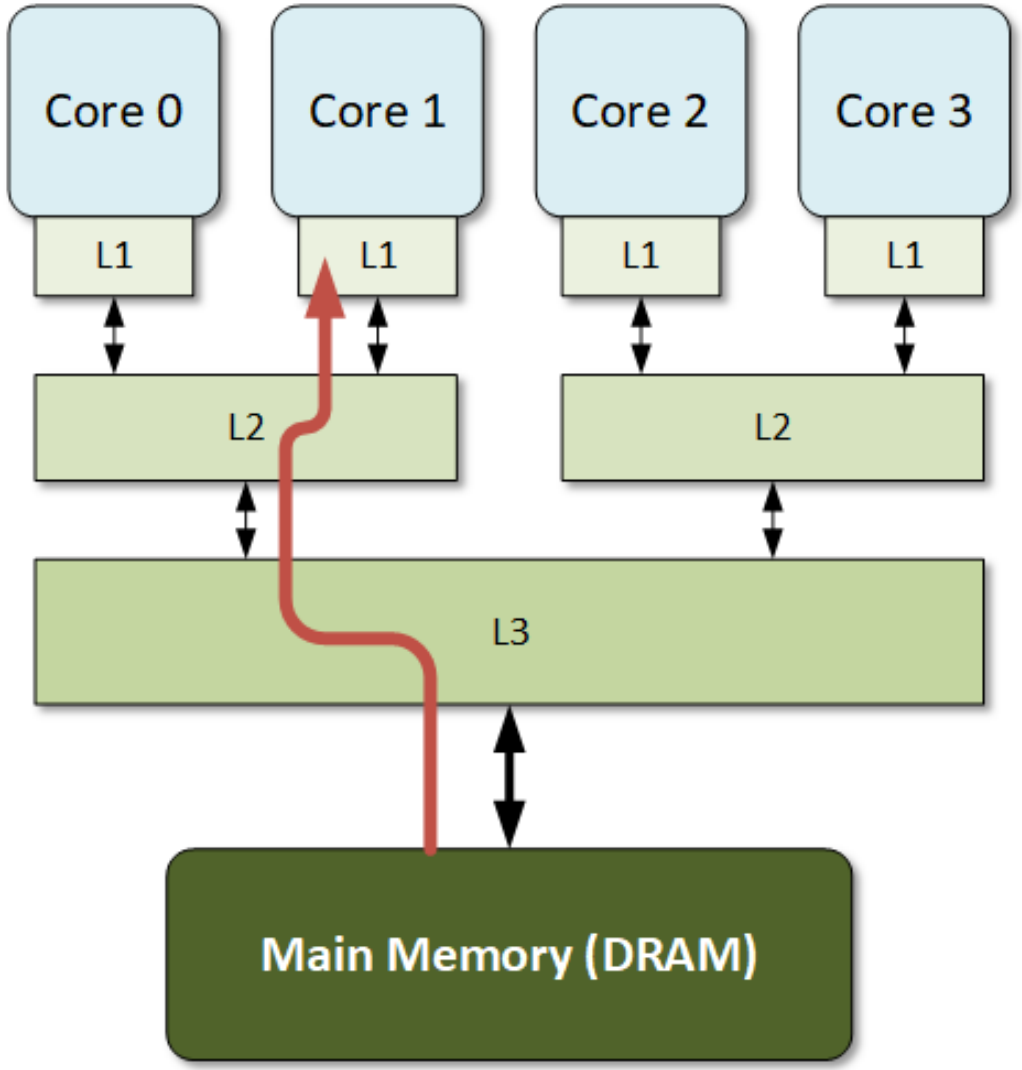
- The use of **multiple processors** on the same chip with a large **shared cache**
- If the software can support the effective use of multiple processors, then doubling the number of processors almost doubles performance
- Strategy is to use two simpler processors on the chip rather than one more complex processor



Multicores

Multiple cores or multicore

- Two-core chips were quickly followed by four-core chips, then 8, then 16, and so on
- First-level cache dedicated to an individual processor and levels two and three being shared by all the processors

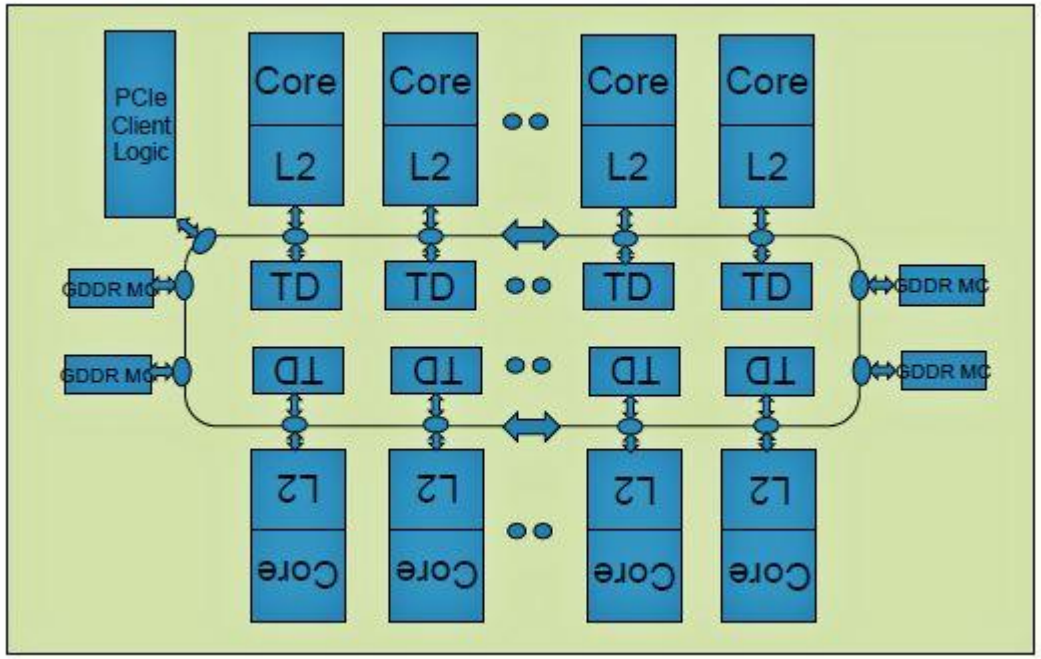


Mics

Many Integrated Core (MIC)

- More than 50 cores per chip
- Difficult to develop software to exploit such a large number of cores

http://www.intel.com/content/www/us/en/products/processors/xeon-phi/xeon-phiprocessors/7290f.html

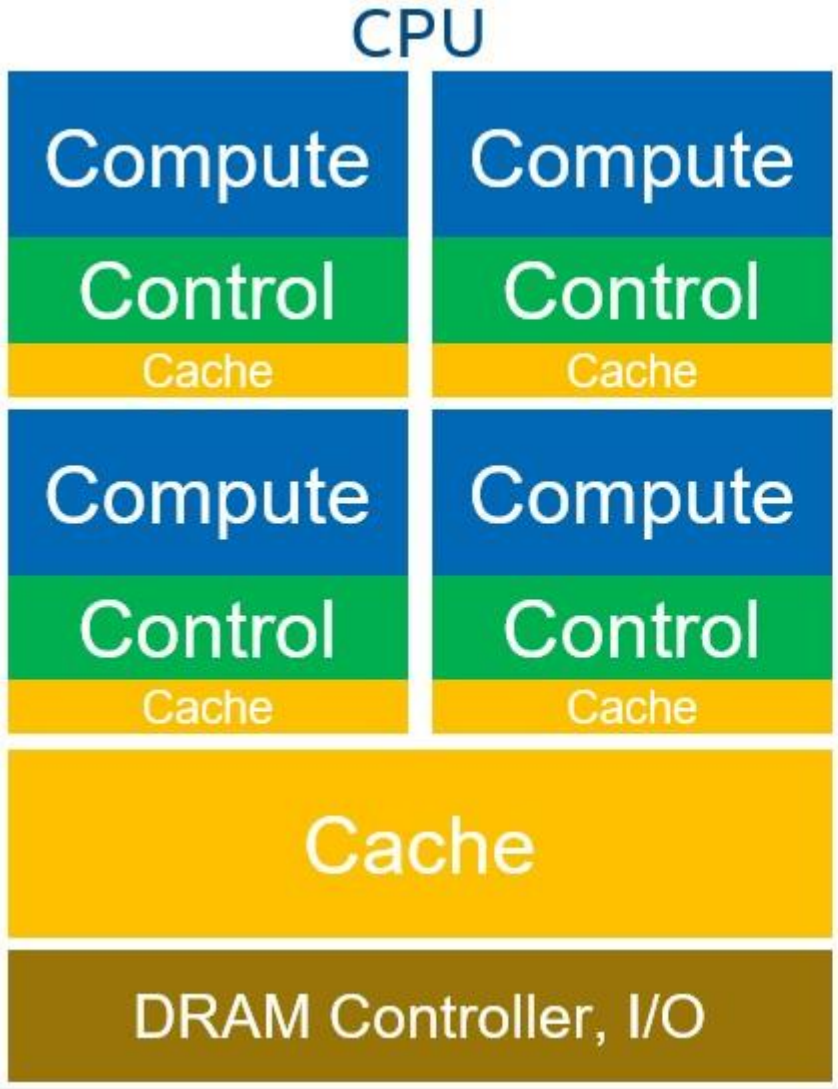


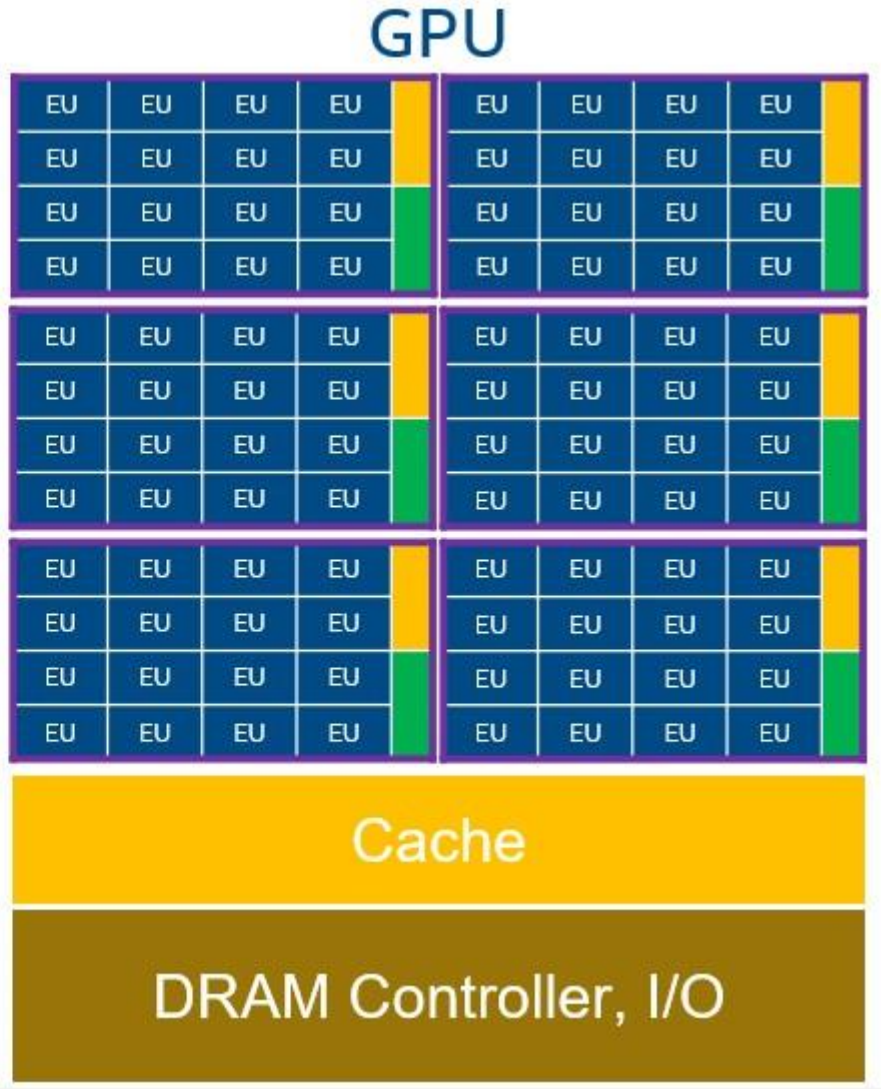
50 x86 cores on the Intel Knights Corner Manycore Coprocessor
EDA360 Insider

GPUs

GPUs and CPUs are both processors,

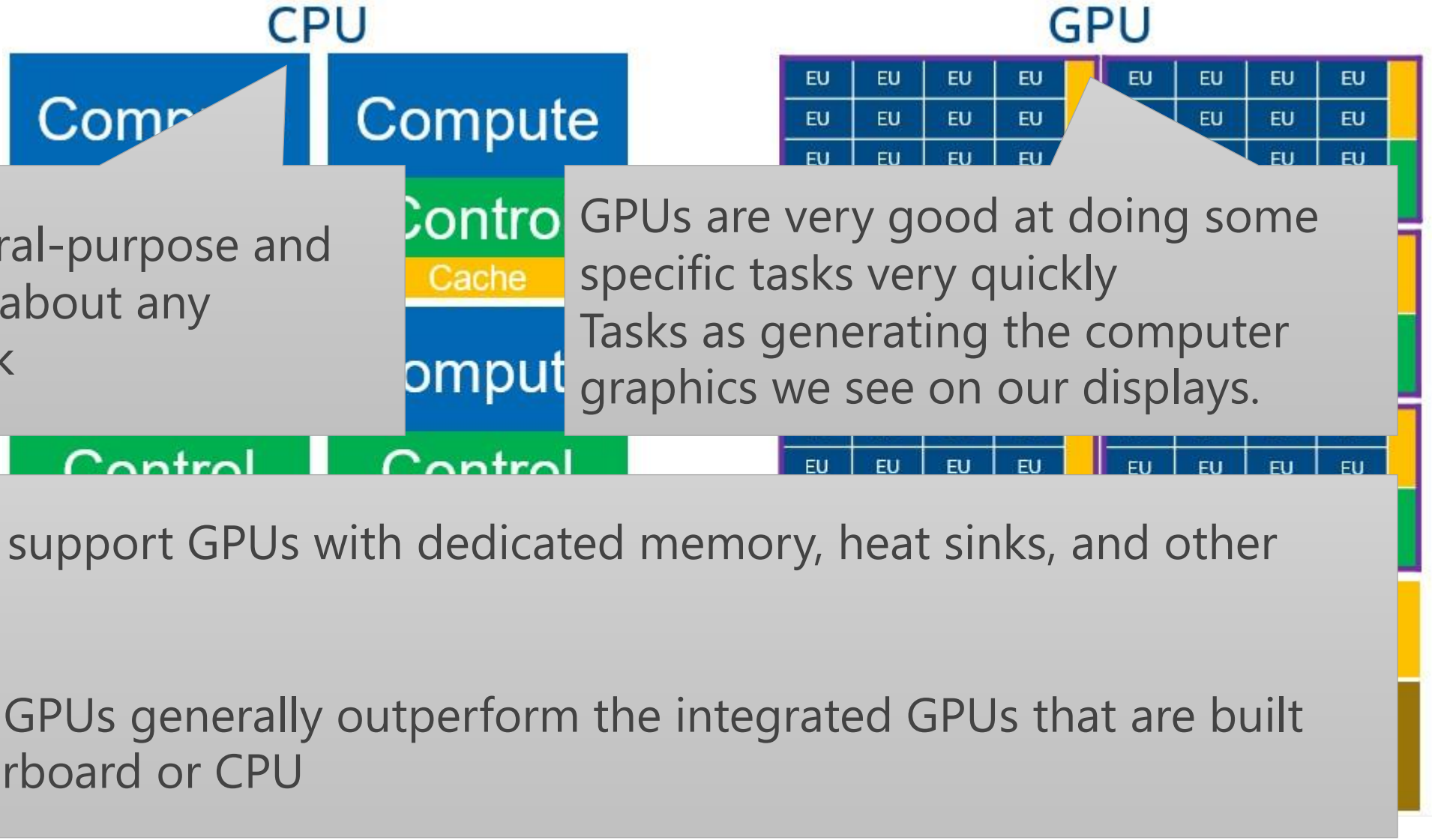
Different in many fundamental ways.





- GPU is a core designed to perform parallel operations on graphics data
- Uses a higher number of simpler cores than the more complex CPUs

GPUs



GP
are
pro

Dif
fun
wa

CPUs are general-purpose and can tackle just about any computing task

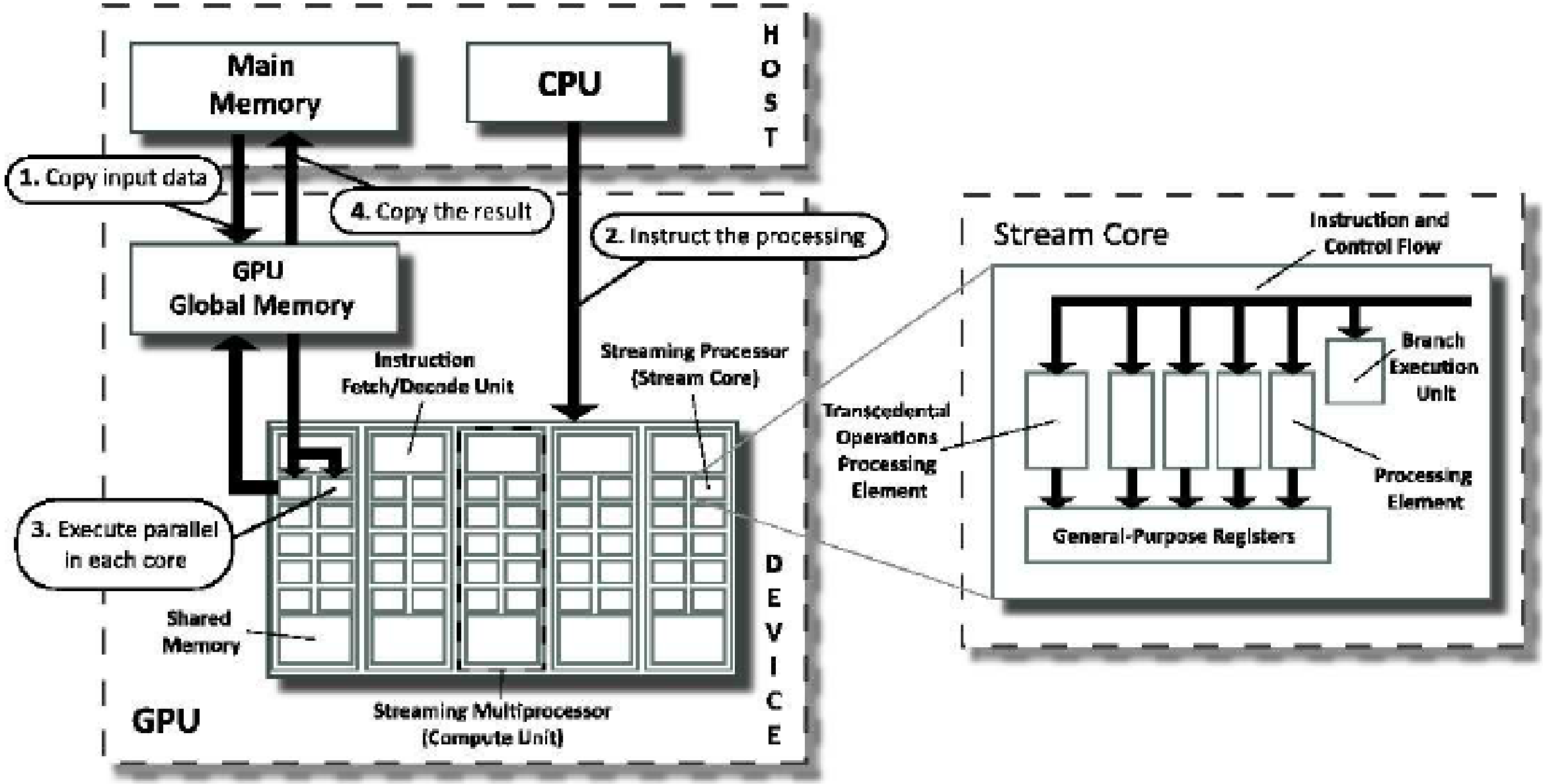
GPUs are very good at doing some specific tasks very quickly
Tasks as generating the computer graphics we see on our displays.

Graphics cards support GPUs with dedicated memory, heat sinks, and other components,

These types of GPUs generally outperform the integrated GPUs that are built into the motherboard or CPU

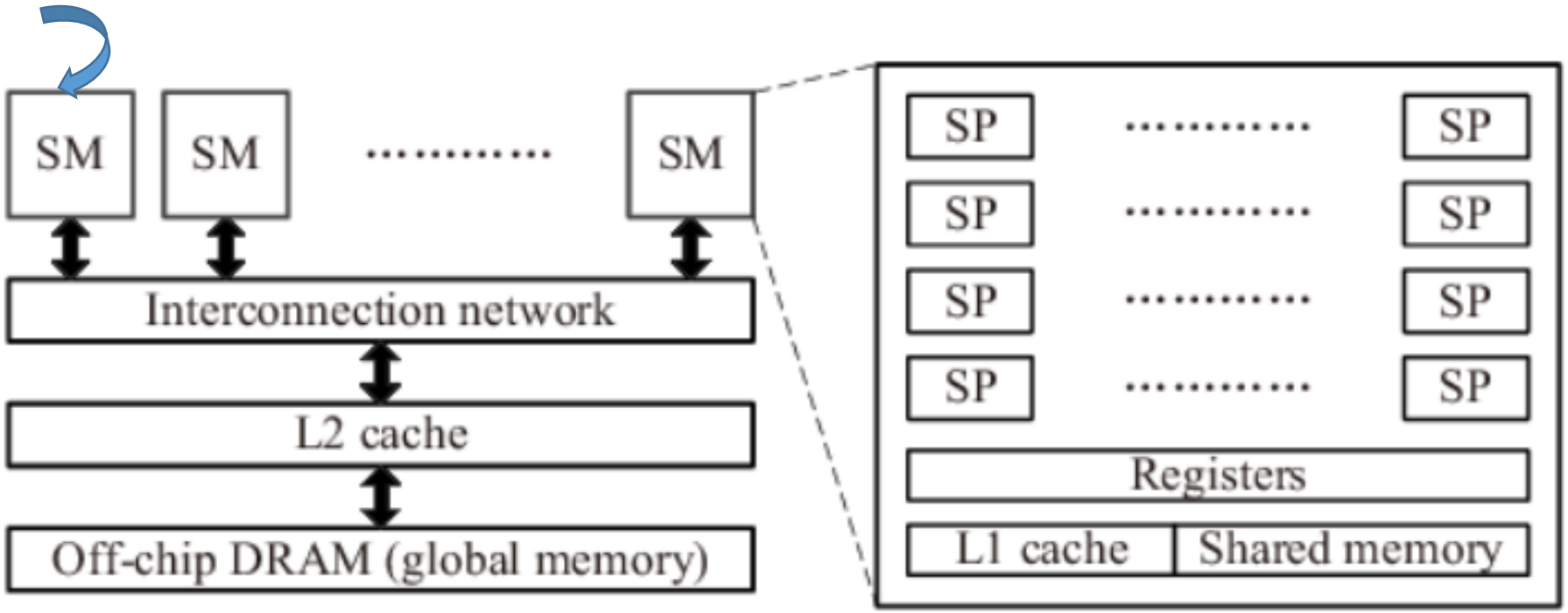
- GPU is a core designed to perform parallel operations on graphics data
- Uses a higher number of simpler cores than the more complex CPUs

GPUs


Main Memory
CPU
HOST
1. Copy input data
4. Copy the result
2. Instruct the processing
GPU Global Memory
Instruction Fetch/Decode Unit
Streaming Processor (Stream Core)
3. Execute parallel in each core
Shared Memory
GPU
Streaming Multiprocessor (Compute Unit)
DEVICE
Stream Core
Instruction and Control Flow
Branch Execution Unit
Transcedental Operations Processing Element
Processing Element
General-Purpose Registers

GPUs

Streaming Multiprocessor



GPGPUs

- It is discovered that many other computing tasks can be performed by GPUs, so long as the data is in graphical form.

- GPUs used for something other than generating graphics are sometimes called GPGPUs.

- High performance computing (HPC), AI development, and many other astounding breakthroughs have been made possible by using servers with a large number of GPGPUs.

Evolution of the Intel x86 Architecture

8080

- World's first general-purpose microprocessor
- 8-bit machine, with an 8-bit data path to memory

8086

- 16-bit machine
- larger registers
- Pre-fetch few instructions before they are executed

80286

- Extension of the 8086 enabled addressing a 16-MByte memory (instead of just 1-Mbyte)

Evolution of the Intel x86 Architecture

80386

- Intel's first 32-bit machine
- First Intel processor to support multitasking, meaning it could run multiple programs at the same time

80486

- Much more sophisticated and **powerful cache technology**
- Sophisticated **instruction pipelining**

Pentium

- Multiple instructions to execute in parallel
- L2 Cache

Pentium Pro

- Branch prediction
- Data flow analysis

Evolution of the Intel x86 Architecture

Pentium II

- Intel MMX technology, which is designed specifically to process video, audio, and graphics data

Pentium III

- Incorporates additional floating-point instructions to support 3D graphics software

Pentium 4

- Includes additional floating-point and other enhancements for multimedia

Evolution of the Intel x86 Architecture

Core

- This is the first Intel x86 microprocessor with a dual core
- Implementation of two processors on a single chip

Core 2

- Core 2 extends the architecture to 64 bits
- Core 2 Quad provides four processors on a single chip

Exercise: Identify the specialities in **Core i3, Core i5, Core i7**

Performance Assessment : Processor Speed

Traditional measures of processor speed

Clock Speed

Instructions per Second

Cycles per instruction

- Clock rate or clock speed - 1-GHz processor receives 1 billion pulses per second
- Clock cycle - One increment, or pulse, of the clock
- Cycle time - Time between pulses

Performance Assessment : Cycles Per Instruction

Cycles per instruction (CPI)

- Number of clock cycles required varies for different types of instructions, such as load, store, branch
- CPI_i - number of cycles required for instruction type i
- N_i - number of executed instructions of type i for a given program
- N_c - number of machine instructions executed for that program
- n is the number of instruction types

$$CPI = \frac{\sum_{i=1}^{n}(CPI_i \times N_i)}{N_c}$$

Performance Assessment : Amdahl's Law

Potential speedup of a program using multiple processors compared to a single processor

Let, ***T*** be the total execution time of the program using a single processor

Fraction $(1 - f)$ of the execution time involves code that is serial

Fraction ***f*** that involves code that is parallelizable

$$\text{Speedup} = \frac{\text{Time to execute program on a single processor}}{\text{Time to execute program on } N \text{ parallel processors}}$$

$$= \frac{T(1-f) + Tf}{T(1-f) + \frac{Tf}{N}} = \frac{1}{(1-f) + \frac{f}{N}}$$

Performance Assessment

Application performance depends not just on the raw speed of the processor

- on the instruction set
- choice of implementation language
- efficiency of the compiler
- skill of the programmer

Reference

William Stallings - Computer Organization and Architecture Designing for Performance (9th Edition)

- 2.2 Designing For Performance
- 2.3 Multicore, Mics, And GPGPUs
- 2.4 The Evolution of the Intel x86 Architecture
- 2.6 Performance Assessment

Questions

- What is Pipelining in processors?
- How a system architect can address the data transfer speed lag between main memory and the processor?
- What is the main reason to design multicore computer chips?
- What is Intel MMX technology?
- What is Amdahl’s Law?

Cache Memory

Designed using **SRAM** usually on the same chip as the processor

Cache challenges

- The challenge in cache design is to ensure that the desired data and instructions are in the cache. It should achieve a high hit ratio
- The cache system must be quickly searchable as it is checked every memory reference

There may be 1000 times more RAM than cache

The cache algorithms have to carefully select the 0.1% of the memory that is likely to be most accessed

Cache Memory : Organization

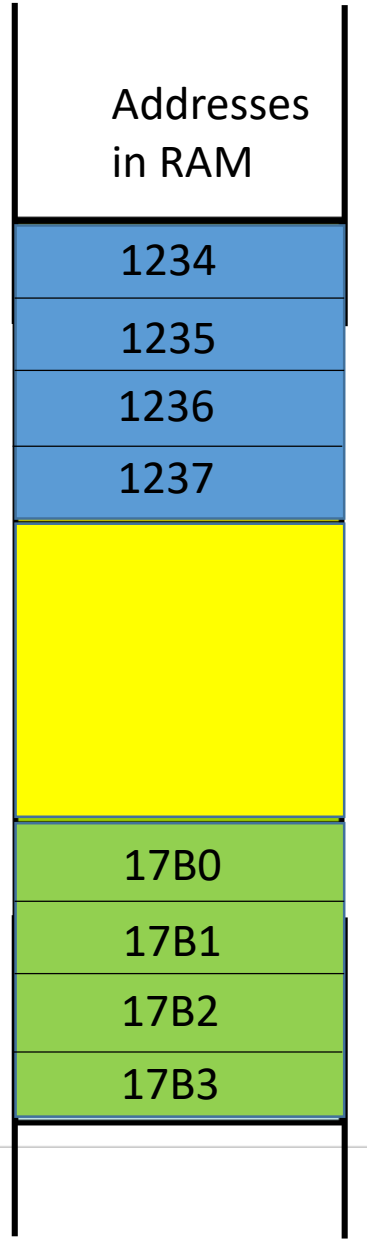
Cache is organized into a set of lines

Each line consists of a fixed number of bytes

A whole line of data copied from RAM and placed at a line when required

Address in the main memory

Tag	Data	Data	Data	Data
1234	from 1234	from 1235	from 1236	from 1237
2458	from 2458	form 2459	from 245A	from 245B
17B0	from 17B0	from 17B1	from 17B2	from 17B3
5244	from 5244	from 5245	from 5246	from 5247



Given the address of the RAM, identification of a memory location in cache is called **mapping**

Cache Memory : Mapping

The memory management unit has to quickly determine if a given memory address is in the cache

Direct mapping	:	Each memory address is mapped exactly into a specific location in cache
Fully associative mapping	:	A memory address can be anywhere in the cache. Have to search the entire cache
Set associative mapping	:	Each memory address can be in one of the sets in cache memory

Cache Memory : Direct Mapping

Assume if we have a Cache with 8 lines,

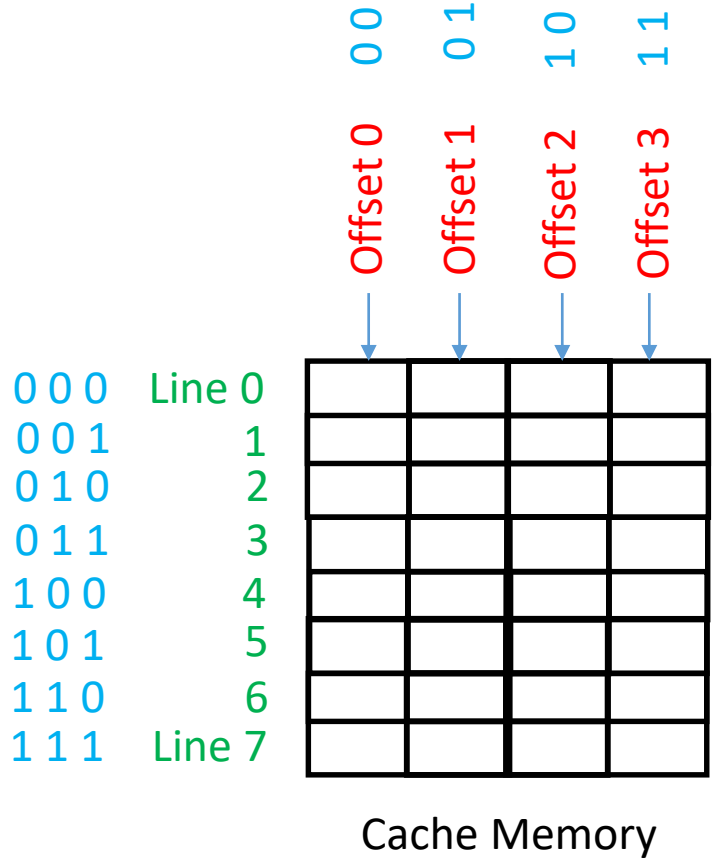
each line contains 4 bytes

In order to access 4 bytes in each line

$Log_2(4) = 2$ bits required (Offset)

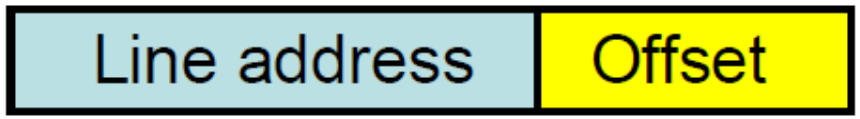
In order to access 8 lines

$Log_2(8) = 3$ bits required (Line address)



Cache Memory

Addressing a byte in cache require 5 bit addressing



Dr. Subodha Gunawardena

Cache Memory : Direct Mapping

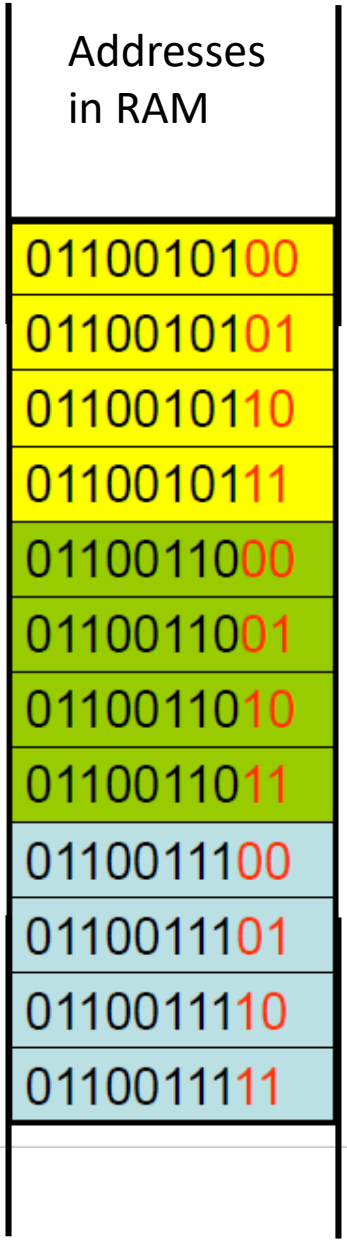
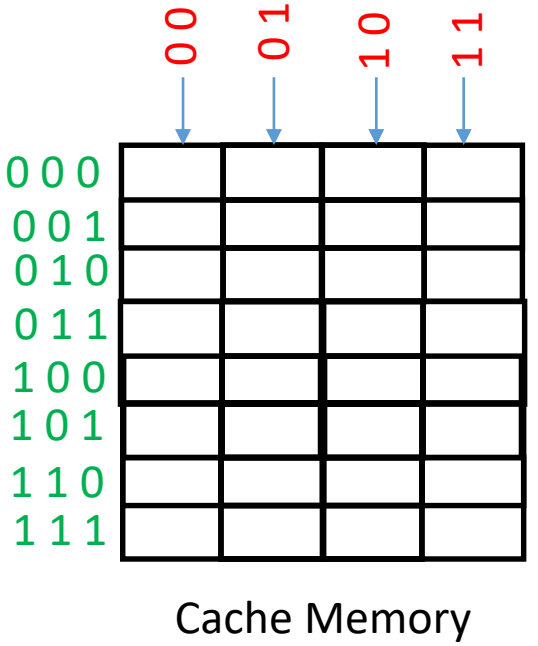


Figure represent a RAM with 10 bit addressing.

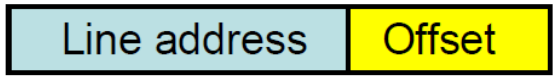
That is the RAM size is 2^{10} = 1 kB

Whenever we map an address from RAM to Cache, least significant (left most) 5 bits are used to address the location in Cache.

Remaining 5 most significant bits are called the Tag

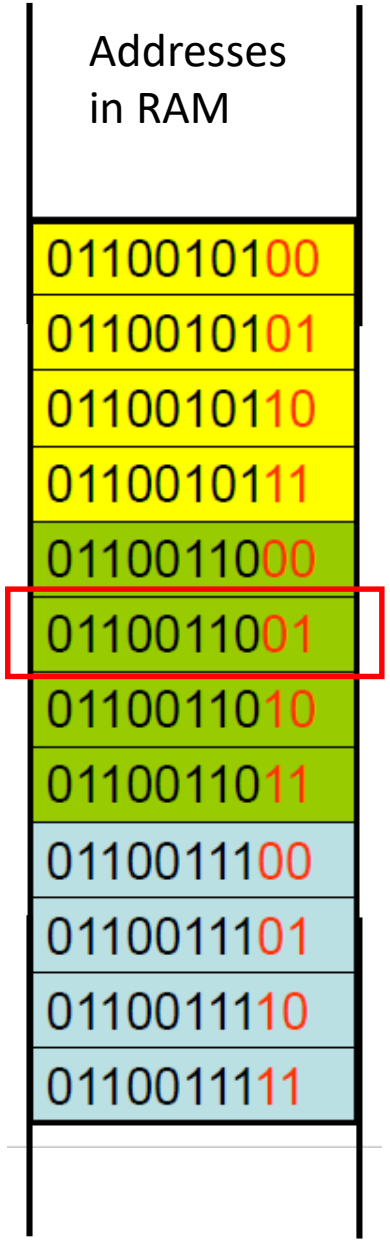


Cache Memory



Dr. Subodha Gunawardena

Cache Memory : Direct Mapping



If we consider the address 0110011001

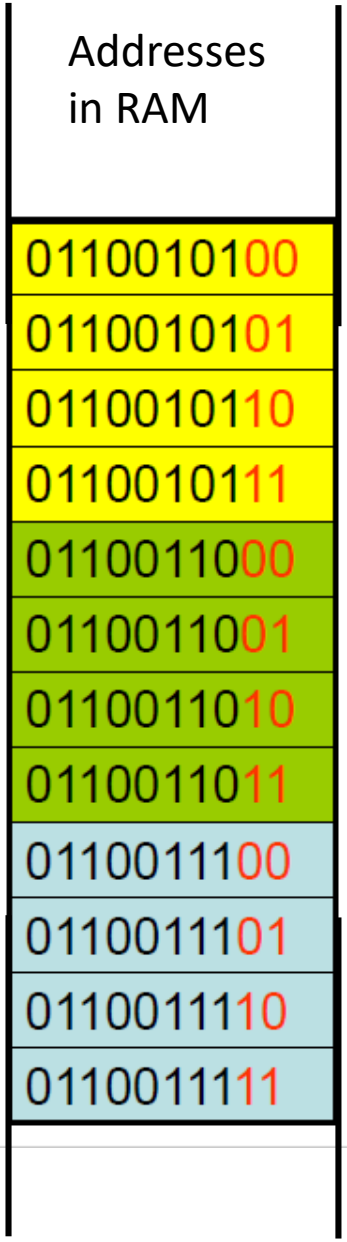
Tag: 01100 Line: 110 Offset: 01

Tag	Data offset 0	Data offset 1	Data offset 2	Data offset 3
xxxxx				

Compare the Tag at line 110 with tag Portion of the Memory address to check if the data is in the cache

Dr. Subodha Gunawardena

Cache Memory Direct Mapping : Exercise



A memory system has a 32 bit bus, 64 byte lines, 32 KB of cache answer the following questions

Size of RAM?

Number of bits to represent Offset?

Number of Lines? Number of bits to represent Line?

Tag length? Number of bits to represent tag?

What is the Tag for memory address

01111101011101110001101100111000 ?

Dr. Subodha Gunawardena

Cache Memory : Associative Mapping

In associative cache mapping, the data from any location in RAM can be stored in any location in cache

When the processor wants an address, all tag fields in the cache are checked to determine if the data is already in the cache

Each tag line requires circuitry to compare the desired address with the tag field

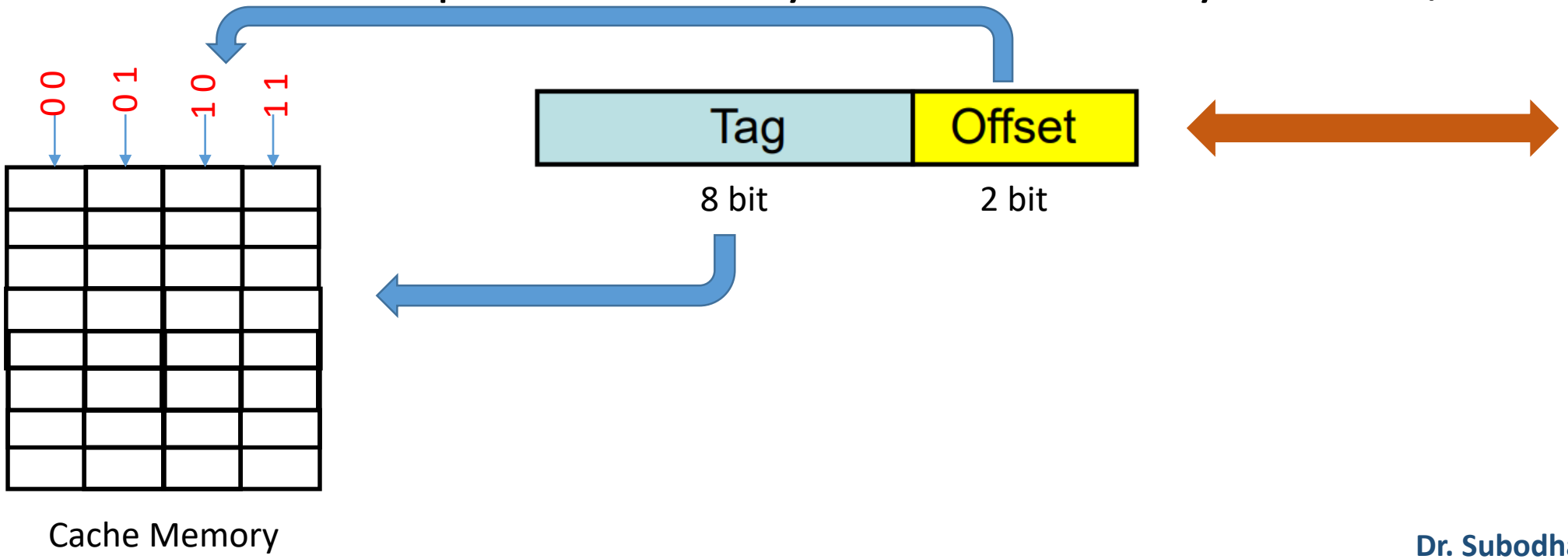
All tag fields are checked in parallel (simultaneously)

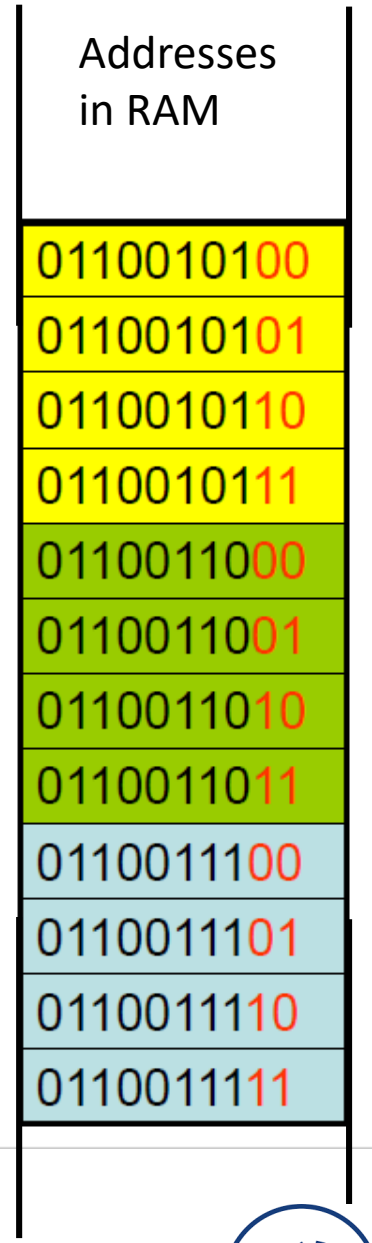
Cache Memory : Associative Mapping

The lower $\log_2$(line size) bits define which byte in the block/line

The remaining upper bits are the tag field.

For a 1 kB address space with 32 Byte cache and 4 byte blocks/lines:



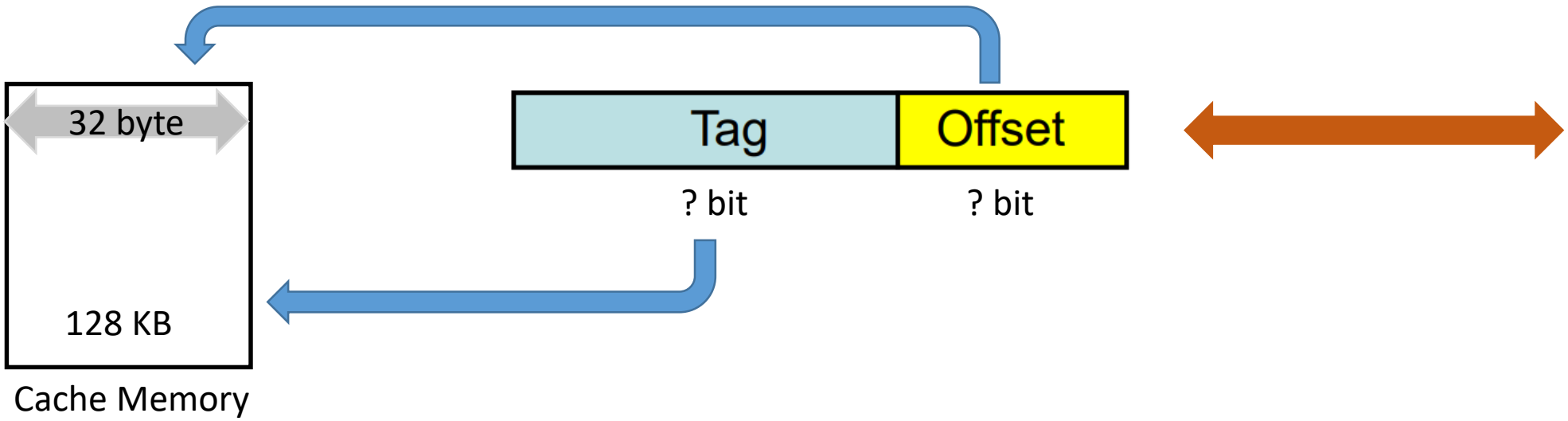


Cache Memory Associative Mapping : Exercise

A memory system has a 4GB RAM, 128 kB Cache, and 32 byte Blocks answer the following questions

What is the Tag for Memory address

01111101011101110001101100111000 ?



Cache Memory : Set Associative Mapping

Set associative mapping is a compromising between the fully associative mapping and direct mapping

Lines of cache are grouped into **'sets'.**

A block of main memory can be mapped into any of the lines in a specified set that block is assigned to

2 lines per set is called 2-way set associative mapping

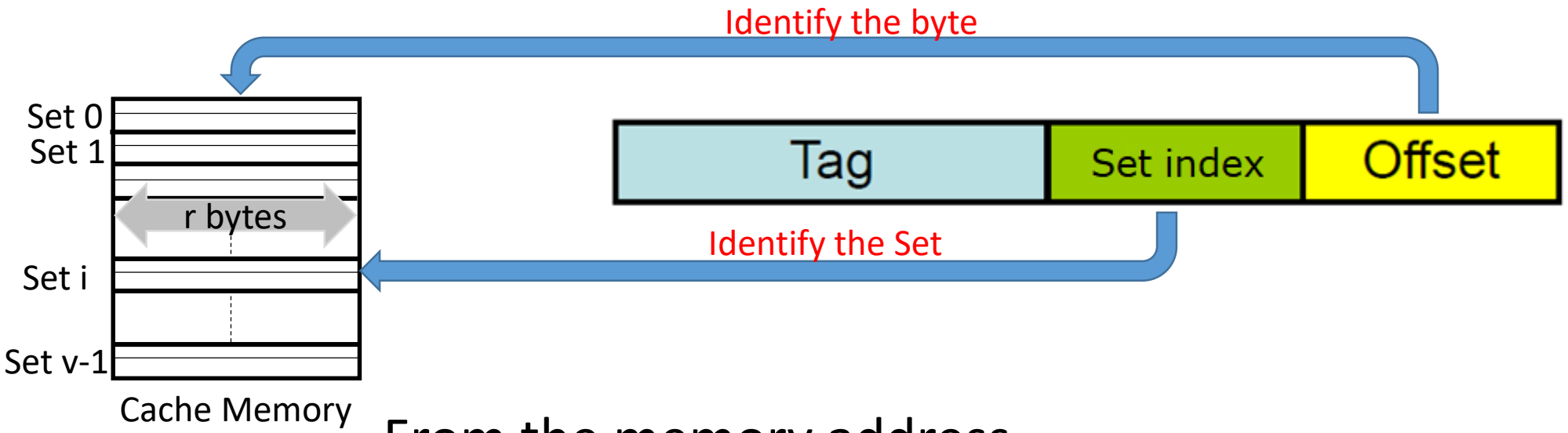
4 lines per set is called 4-way set associative mapping

Number of lines per set increases in powers of 2s

Cache Memory : Set Associative Mapping

In 2-way set associative mapping,

2 lines per set, v sets in the cache, and r bytes in a line. Cash size is 2*v*r bytes



From the memory address,

Identify the **set** and check if **lines** in the set has matching **tag**.

If the **tag** is matching, identify the byte using **offset**.

Dr. Subodha Gunawardena

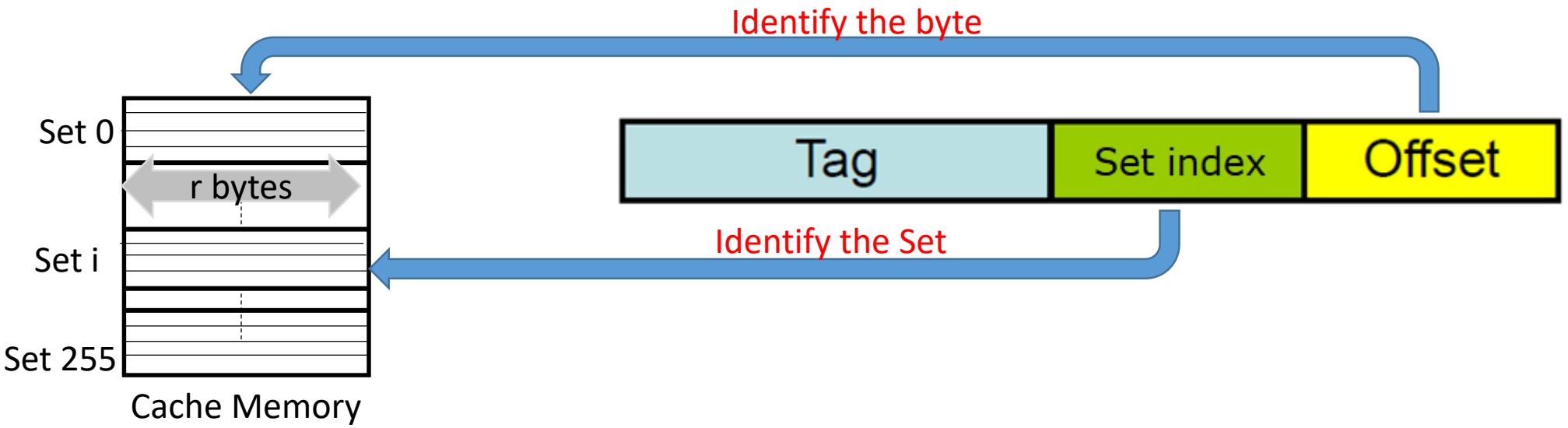
Cache Memory Set Associative Mapping : Exercise

A memory system has a 24 bit bus, 4-way set associative mapping, 256 sets, and a 2 bit offset.

Size of RAM?

Size of the Cache?

What is the Tag for memory address 011111010111011100011011001110000 ?



Dr. Subodha Gunawardena

Cache Memory : Performance Matrices

Miss Rate Miss Rate (p_miss**)**

Fraction of memory references not found in cache (misses/references)

Typically 3-10% for L1 cache and can be quite small (< 1%) for L2 cache, depending on size, etc.

Hit Time (t_hit**)**

Time to deliver a line in the cache to the processor (includes time to determine whether the line is in the cache)

Typically 1-3 clock cycle for L1 cache and 5-12 clock cycles for L2 cache

Miss Penalty (t_penalty**)**

Additional time required because of a miss

Typically 100-300 cycles for main memory access

Cache Memory : Average Memory Access Time

$T_{access} = (1 - p_{miss})\, t_{hit} + p_{miss} * t_{miss}$ --- (1)

$t_{miss} = t_{hit} + t_{penalty}$ ---(2)

From (1) and (2)

$T_{access} = t_{hit} + p_{miss} . t_{penalty}$

Assume Level 1 cache shows a 90% hit rate, 1 cycle hit time, and a 200 cycle miss penalty

Average memory access time = ?

Cache Memory : Average Memory Access Time

T_access = (1 – p_miss) t_hit + p_miss) t_miss --- (1)

t_miss = t_hit + t_penalty ---(2)

From (1) and (2)

T_access = t_hit + p_miss . t_penalty

Assume Level 1 cache shows a 90% hit rate, 1 cycle hit time, and a 200 cycle miss penalty

Average memory access time = 1 + 0.1 * 200 = 21 cycles

Cache Memory : Average Memory Access Time

How to reduce average memory access time?

- Reduce miss rate
- Reduce miss penalty
- Reduce hit time

Miss penalty and **Hit time** are mostly hardware dependent and are expensive to improve.

Hit rate (1 - Miss rate) can be improved by proper memory management and program code.

Can write code to improve hit rate

Cache Memory : Example

```
int sumArrayRows(int a[M][N]) {
        int i, j, sum = 0;
        for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
                sum += a[i][j];
        return sum;
}
```

Accessing memory

in order → high hit-ratio

```
int sumArrayRows(int a[M][N]) {
        int i, j, sum = 0;
        for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
                sum += a[i][j];
        return sum;
}
```

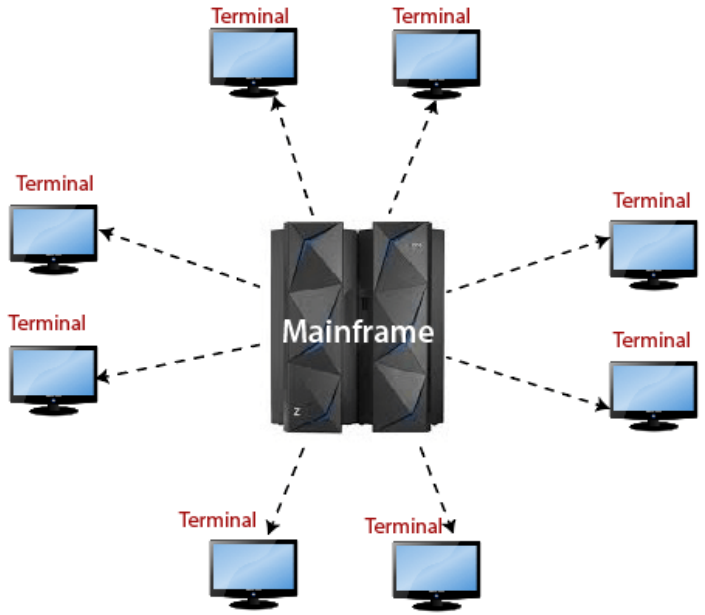
Memory access

not in order → low hit-ratio

Dr. Subodha Gunawardena

Computer Architecture and Speed

Today's laptops have the computing power of an **IBM mainframe** from 10 or 15 years ago



But, the basic building blocks for today's computers are virtually the same as those of the IAS computer from over 50 years ago

Microprocessor Speed

- While the chipmakers have been busy learning how to fabricate chips of greater density
- Processor designers came up with more elaborate techniques

Computer Architecture and Speed : Mainframe

Mainframe computers

- Large amount of memory and processing power
- Extremely complex processes run Entirely in memory.
- Used in systems such as Banking, Government, Security, Education etc.