HELP US ALL STAY HEALTHY

SIX SIMPLE TIPS

Maintain 1.5 metres distance between yourself and others where possible

Cough and sneeze into your elbow or a tissue (not your hands)

Avoid shaking hands

Put used tissues in the bin

Wash hands with soap and warm water or use an alcohol-based hand sanitiser after you cough or sneeze

Do not touch your face

IF YOU ARE UNWELL AND WORRIED ABOUT COVID-19:

- Call the National Coronavirus Helpline: 1800 020 080
- Call your usual GP for advice
- Call the UWA Medical Centre for advice: 6488 2118

UWA FAQs:
uwa.edu.au/coronavirus

Report COVID-19 hazards and suspected/confirmed cases via RiskWare:
uwa.edu.au/riskware

THE UNIVERSITY OF WESTERN AUSTRALIA

DCS:507757521

High-Performance Comput

Lecture 1 Introduction to Computer Architecture

Acknowledgement: The lecture slides are adapted from many online sources.

Teaching Staff Information

- Lecturer and Unit Coordinator
  - **Amitava Datta**
    - Email: amitava.datta@uwa.edu.au
    - Office: CSSE Room 1.07
    - Phone: 6488 3449
  - **Consultation**
    - 10:00 am to 11:00 am Tuesdays
  - **Discussion forum for not personal questions in nature**
- Lab Facilitators
  - **Amitava Datta, TBA**
    - Monday 4-6 pm
    - Wednesday 12-2 pm
- More information
  - ✓ https://research-repository.uwa.edu.au/en/persons/amitava-datta

Assessments, Lectures and Labs

→ Multicore programming
→ Distributed Memory

- **Two programming projects: 50%**
  - ✓ Each project contributes 25%.
  - ✓ Project specifications will be available on the LMS page.
- **Final examination: 50%**
- **Lectures**
  - Face to face + recorded
- **Laboratories**
  - Lab sessions start from **Week 3**.
- C/C++ will be used in this unit

Textbooks and Recommended Readings

THE UNIVERSITY OF WESTERN AUSTRALIA

- **No textbook needed for this unit.**
- **Recommended Readings**
  - **Using OpenMP: Portable Shared Memory Parallel Programming**, by B. Chapman, G. Jost and R. van der Paas, MIT Press, 2008.
  - **Parallel Programming in OpenMP**, by Rohit Chandra et al., Morgan Kaufmann, 2000.
  - **MPI – The Complete Reference**, by Snir et al., MIT Press, 1996.
  - MPI Tutorial: https://mpitutorial.com/tutorials/

Should I Study CITS5507?

THE UNIVERSITY OF WESTERN AUSTRALIA

- This is a “programming” unit.
- You should study this unit, if
  - you enjoy programming,
  - it is a core unit for your degree
- It opens a new door for programming
  - parallel and distributed computing

Outline of Today's Lecture

THE UNIVERSITY OF WESTERN AUSTRALIA

- **What and why we are studying**

- **Computer architecture**
  - ✓ Processor
  - ✓ Process and thread
  - ✓ Memory hierarchy
    - An example on cache line

Note: we will sometime call a computer as a machine.

Why High-Performance Computing

THE UNIVERSITY OF WESTERN AUSTRALIA

- Most of the programs that you write, run almost instantaneously.
- There are programs that take a long time to run.
- People are solving larger and larger problems as computers are becoming faster and faster.
- Some programs never complete execution, e.g. weather simulation (and forecasting) programs.
- The speed of hardware can never keep up with the demands of faster execution of programs.

What is High-Performance Computing

THE UNIVERSITY OF WESTERN AUSTRALIA

- **What is meant by high-performance computing?**
  - ✓ The practice of aggregating computing power to deliver much higher performance than one could get out of a typical desktop computer or workstation.
  - ✓ To solve large problems in science, engineering or business.

- **High-performance computing**
  - ✓ Computations done on multiple machines, or a single machine with multiple/many cores.
  - ✓ About using all the compute resources when possible

Current Trend in HPC

- **Single machine is getting faster and cheaper**
  - ✓ Graphics Processing Units (GPUs)
  - ✓ Multi/Many core CPUs
  - ✓ AI Accelerators (e.g. Google TPUs)

- **HPC cluster with multiple machines**
  - ✓ Enormous data sizes
  - ✓ On-demand HPC infrastructure
  - ✓ Much faster networking capabilities

CPU vs TPU vs GPU

Why Should You Care

- Electronic computers were invented to perform computation, but we still need to design programs.

- Learning to write scalable code is something difficult to learn by yourself in a short time.

- Writing fast, reliable code will never go out of style.
  - ✓ Someone has to build the efficient tools
  - ✓ Large, numerical computations always exist
  - ✓ Most of the tools used in data-science, AI, etc. (e.g. Sklearn) are all high-performance codes.

Outline of Today's Lecture

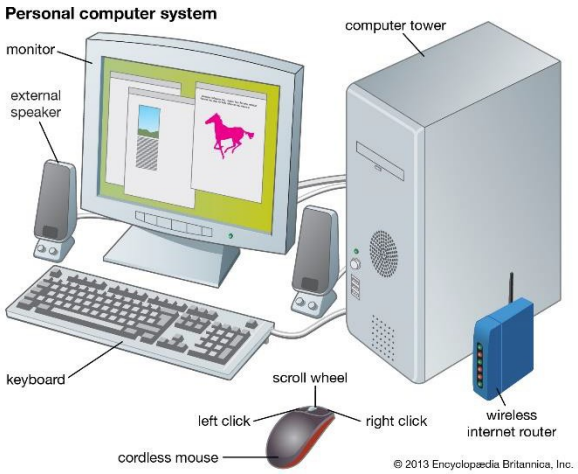
THE UNIVERSITY OF WESTERN AUSTRALIA

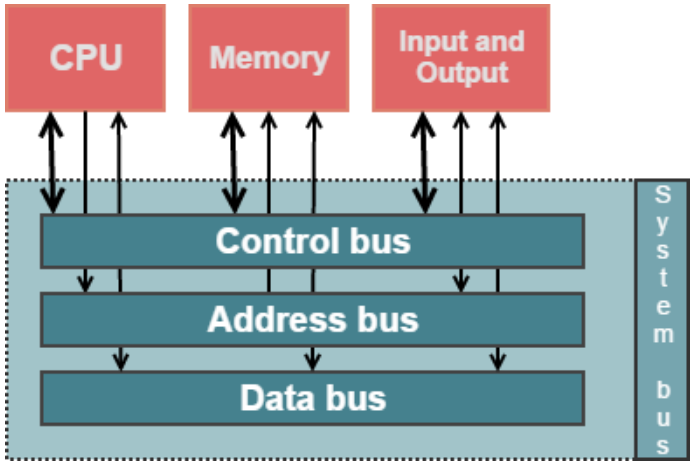
- **What and why we are studying**

- **Computer architecture**
  - ✓ Processor
  - ✓ Process and thread
  - ✓ Memory hierarchy
    - An example on cache line

What is a Computer

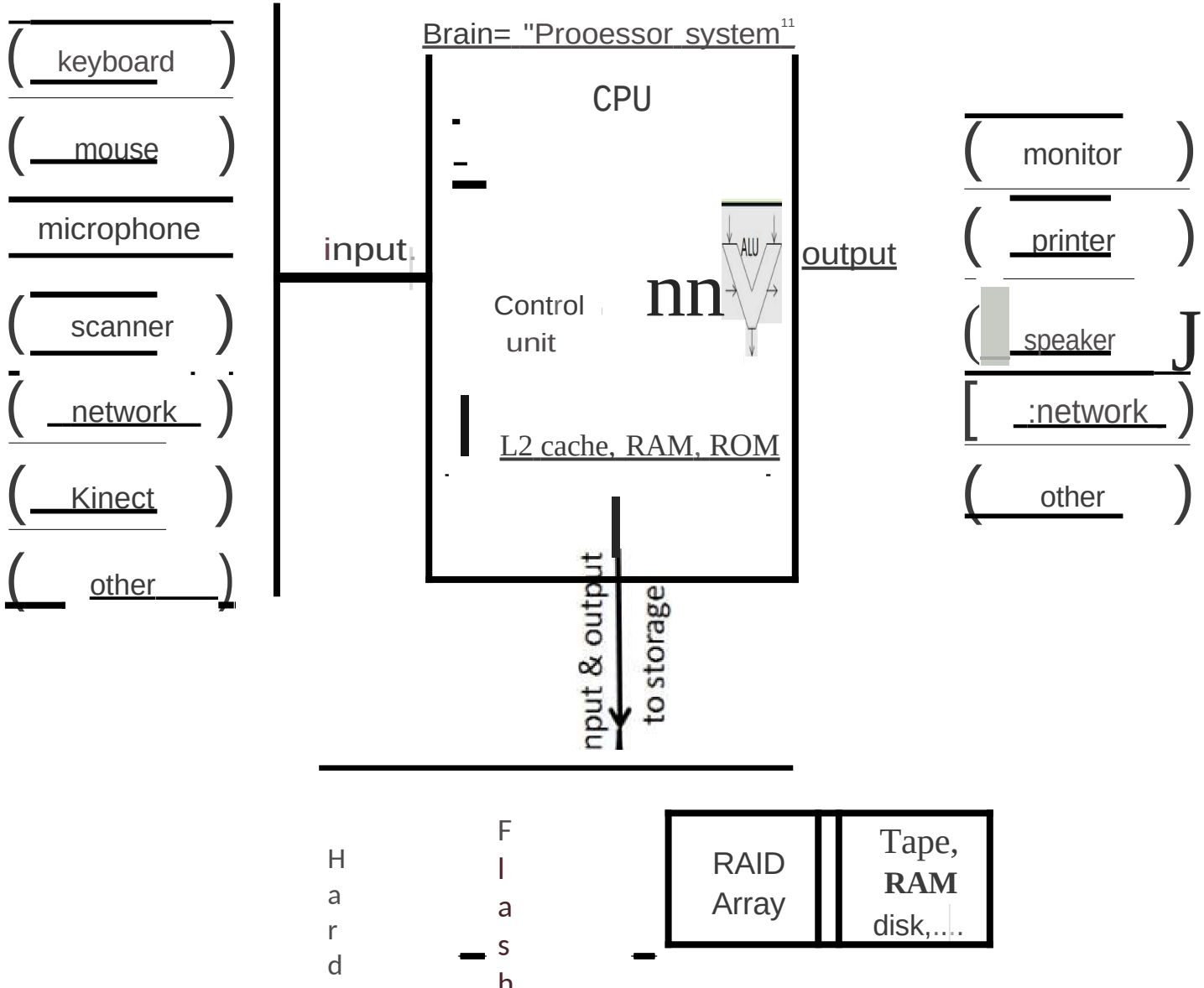
- **A computer has three parts:**
  - ✓ the Central Processing Unit (CPU) or processor,
  - ✓ the memory, and some input/output (I/O) devices.
- **The parts are connected by system bus.**





- For executing a program, it has to be first stored in the memory and then executed on the CPU.
- This is the 'von Neumann' architecture used in all forms of computers today. It was first proposed John von Neumann.

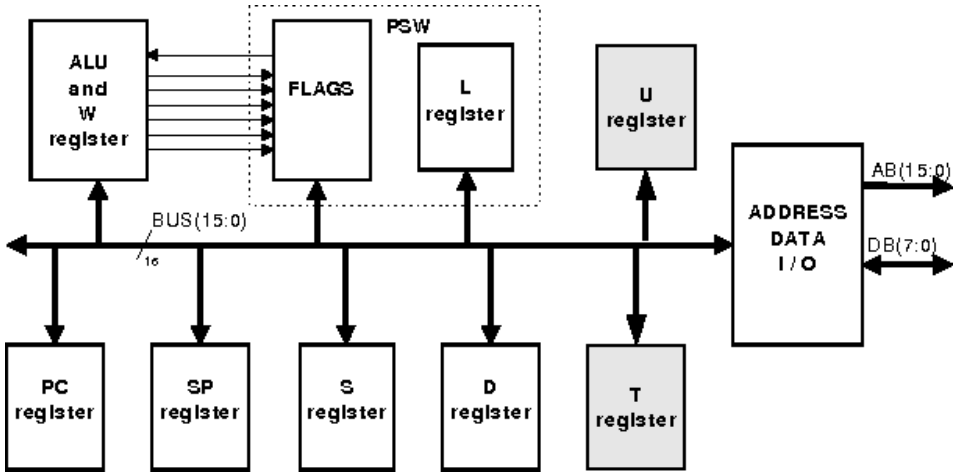
What is a Computer (continued)

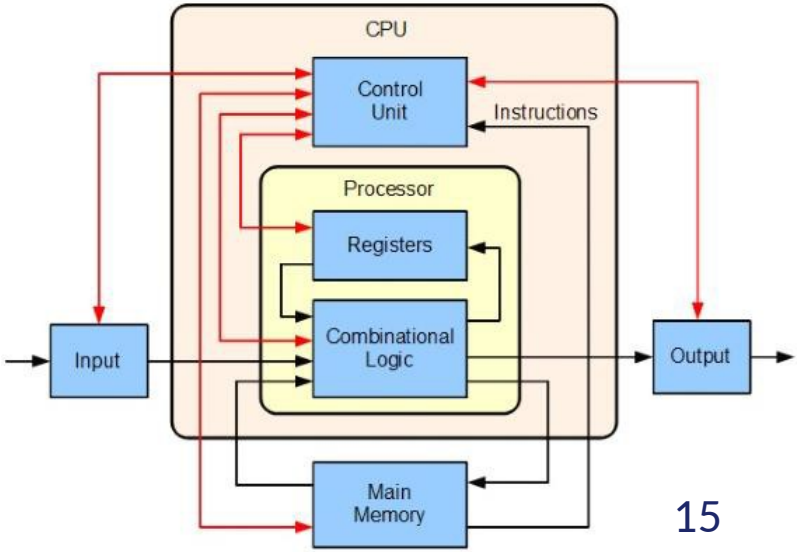


A CPU

is
k

d
ri
v
e

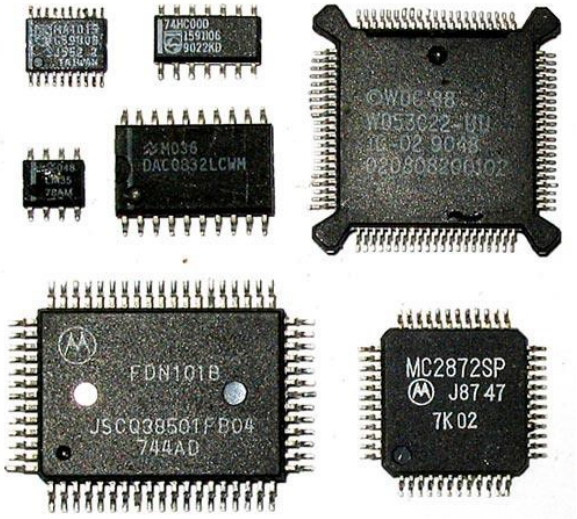


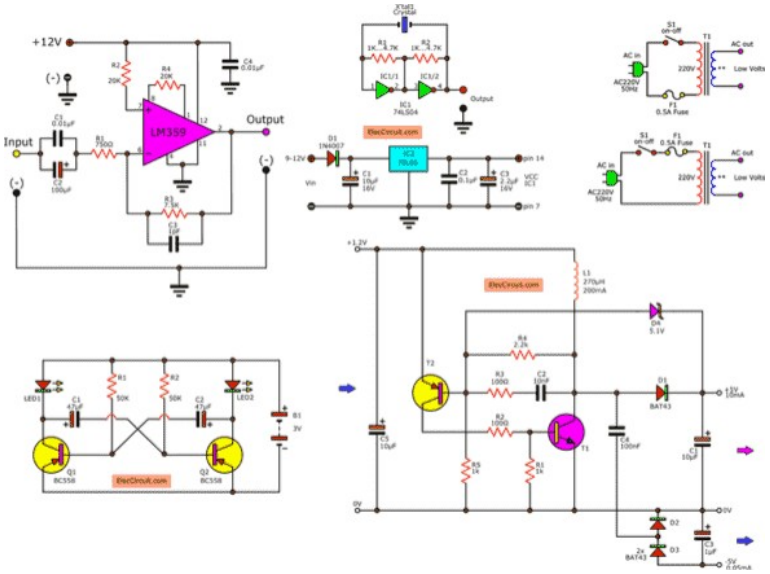


Evolution of Processors

- Every computer (e.g. desktop/laptop and mobile) has a processor (usually an electronic chip).

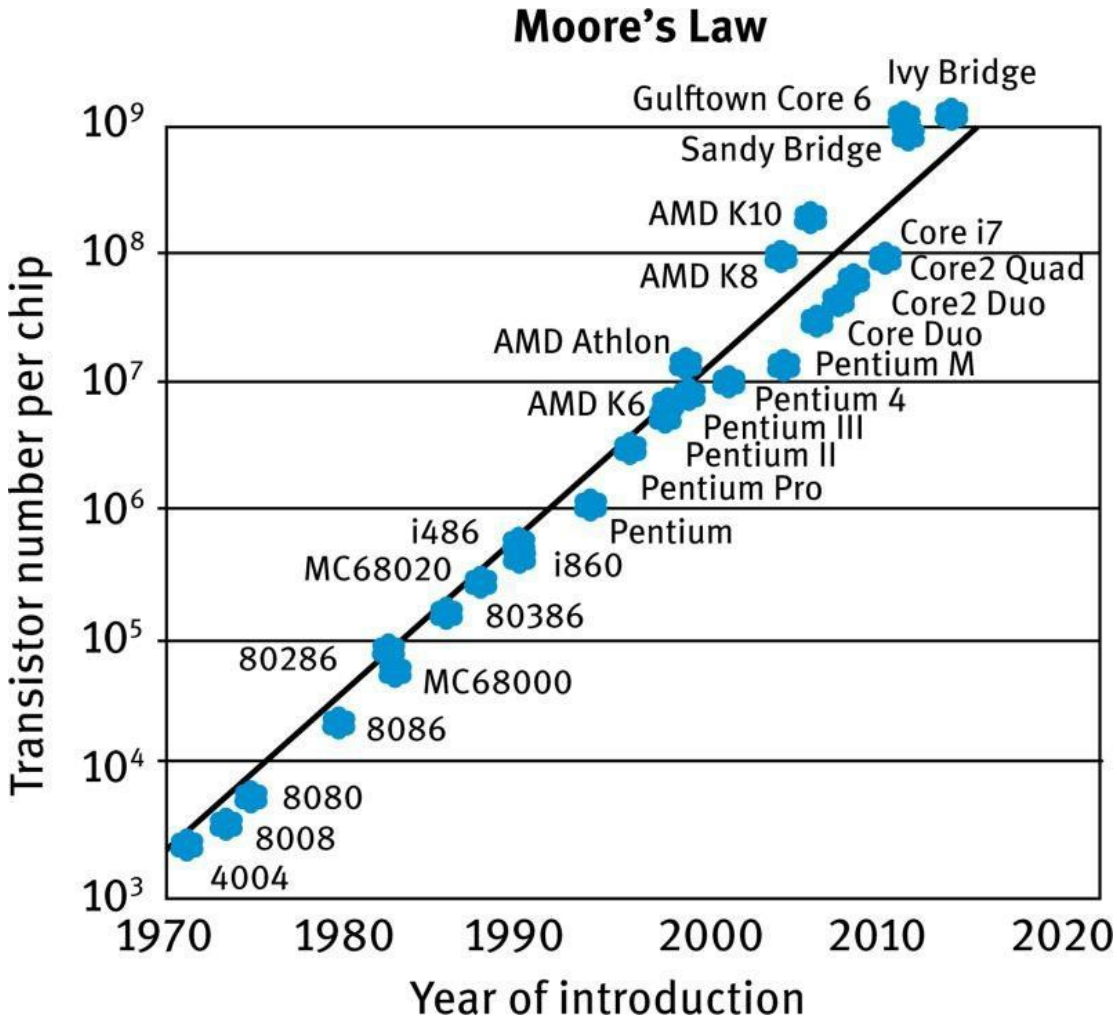
- Processors become more and more powerful.
  - By packing more and more electronic circuits within a small chip area.





Evolution of Processors—Moore's Law

- In 1965, Intel co-founder Gordon Moore predicted that semiconductor density would double every 18 months.
- Since the 2000s, there has been an ongoing debate about whether Moore's Law is still alive or dead.



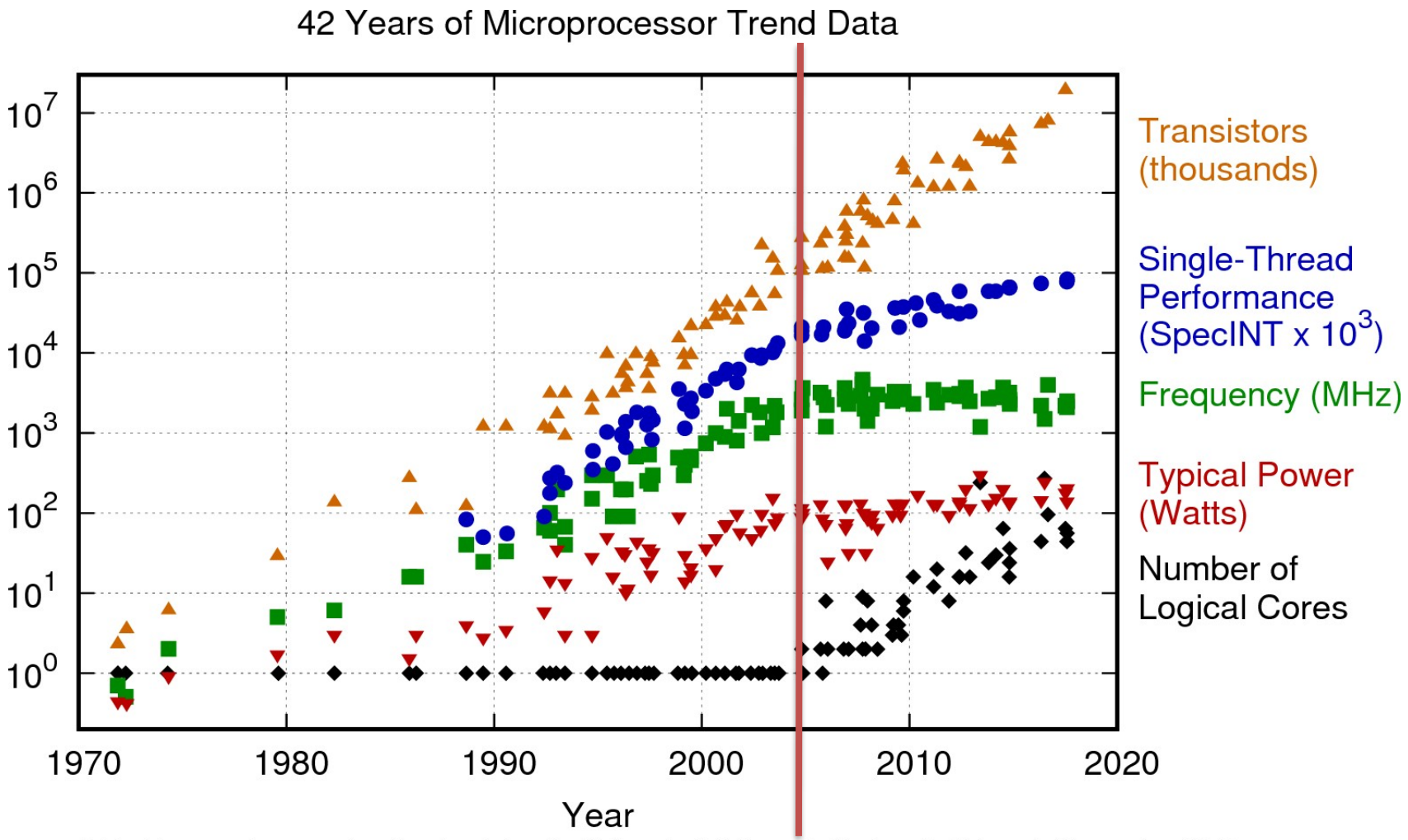
Evolution of Processors

- Every computer (e.g. desktop/laptop and mobile) has a processor (usually an electronic chip).

- Processors become more and more powerful.
  - By packing more and more electronic circuits within a small chip area.

- Processors have almost reached the limits.
  - Physical limit: how small the electronic circuit elements can be.
  - Thermal limit: electronic circuits generate heat during their operations. Processors must be cooled.

Evolution of Processors (continued)



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Multi-Core Processors

THE UNIVERSITY OF WESTERN AUSTRALIA

- The new trend has been
  - ✓ not to have a single core powerful processor,
  - ✓ but to have several less powerful cores in a processor (i.e. multi-core processor)

- The challenge is to make use of multi-core processors for executing programs faster.
  - ✓ We will learn techniques to make use of them.

- There are two related, but slightly different issues regarding this, concurrency and parallelism.

Concurrency

- **Concurrency:**
  - ✓ simultaneous execution of multiple tasks.
  - ✓ E.g. at the same time on a laptop, you could be
    - browsing the internet,
    - writing a program and
    - listening to music.

- **Concurrency could be logical or physical.**
  - ✓ Individual tasks can be executed on the same processor giving a logical concurrency,
  - ✓ or each task may run on a different processor, giving physical concurrency.

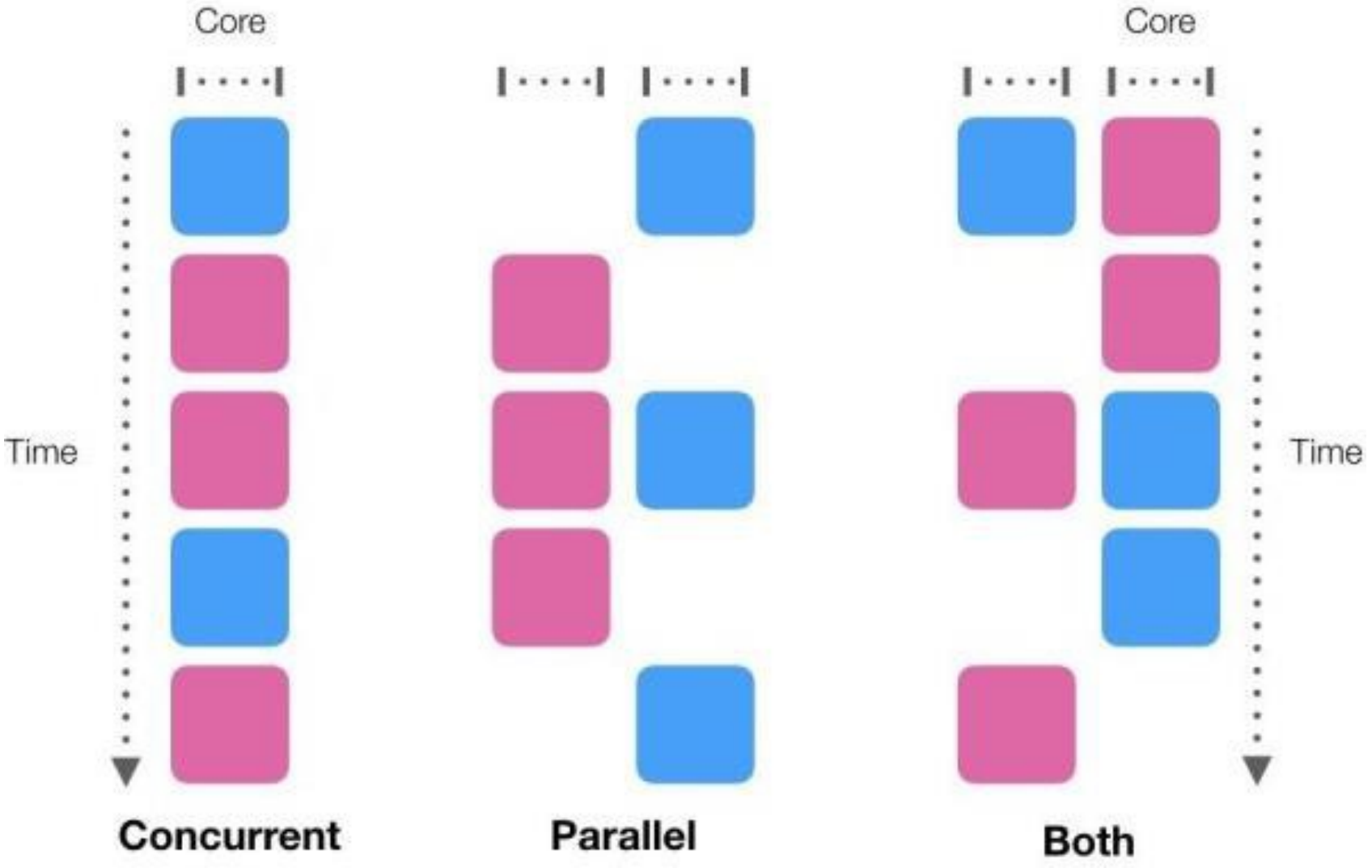
Parallelism

- Parallelism is
  - ✓ breaking up a task into smaller tasks and
  - ✓ solving these smaller tasks simultaneously.

- The smaller tasks will give only partial results and we have to combine these partial results to get the final result.

- Suppose you want to add 1000 numbers.
  - ✓ You can ask **10** friends each of whom adds **100** numbers at the same time.
  - ✓ You add these partial sums to get the final sum.
  - ✓ You can complete the task ~10 times faster than adding the 1000 numbers by yourself.

Concurrency v.s. Parallelism Example



HPC is about Parallelism

THE UNIVERSITY OF WESTERN AUSTRALIA

- Our aim in this unit is to study parallelism on machines that have multicore processors.

- There are many such machines including
  - ✓ laptop, desktop, mobile phone, supercomputers.

- We will study two kinds of machines in this unit,
  - ✓ multicore processors
    - available in your desktop/laptop computer.
  - ✓ cluster of workstations
    - The cluster is called a 'distributed memory' architecture
    - We will have access to a cluster in UWA.

Programs that Computers Can Execute

- Though you write programs in many languages, e.g. *C* and *Python*, a particular computer can execute programs that is written in one particular language.
  - ✓ called the machine language.

- A special program called a compiler translates a program written in a higher level language like *C* or *Python* into a machine language program.
  - ✓ The machine language program is stored in RAM and executed by CPUs, instruction by instruction.
- An example machine language program A=A+B:
  - ✓ MOV A, R1
  - ✓ MOV B, R2
  - ✓ ADD R1,R2
  - ␣ MOV R1,

MPI uses processes; OpenMP uses threads

Outline of Today's Lecture

- **What and why we are studying**
- **Computer architecture**
  - ✓ Processor
  - ✓ Process and thread
  - ✓ Memory hierarchy
    - An example on cache line

What is a Process?

- A program is the code, or the machine code after it has been translated into machine language.
- A process is a program in execution.
- A process requires resources for its execution.
  - ✓ The operating system (OS) allocates memory, and allows the process to make system calls for performing input/output (I/O) operations.
  - ✓ A process can access memory that is allocated to it by the operating system (OS).
    - The OS usually terminates a process if the process tries to access memory that is not allocated to it (e.g. Segmentation Fault).

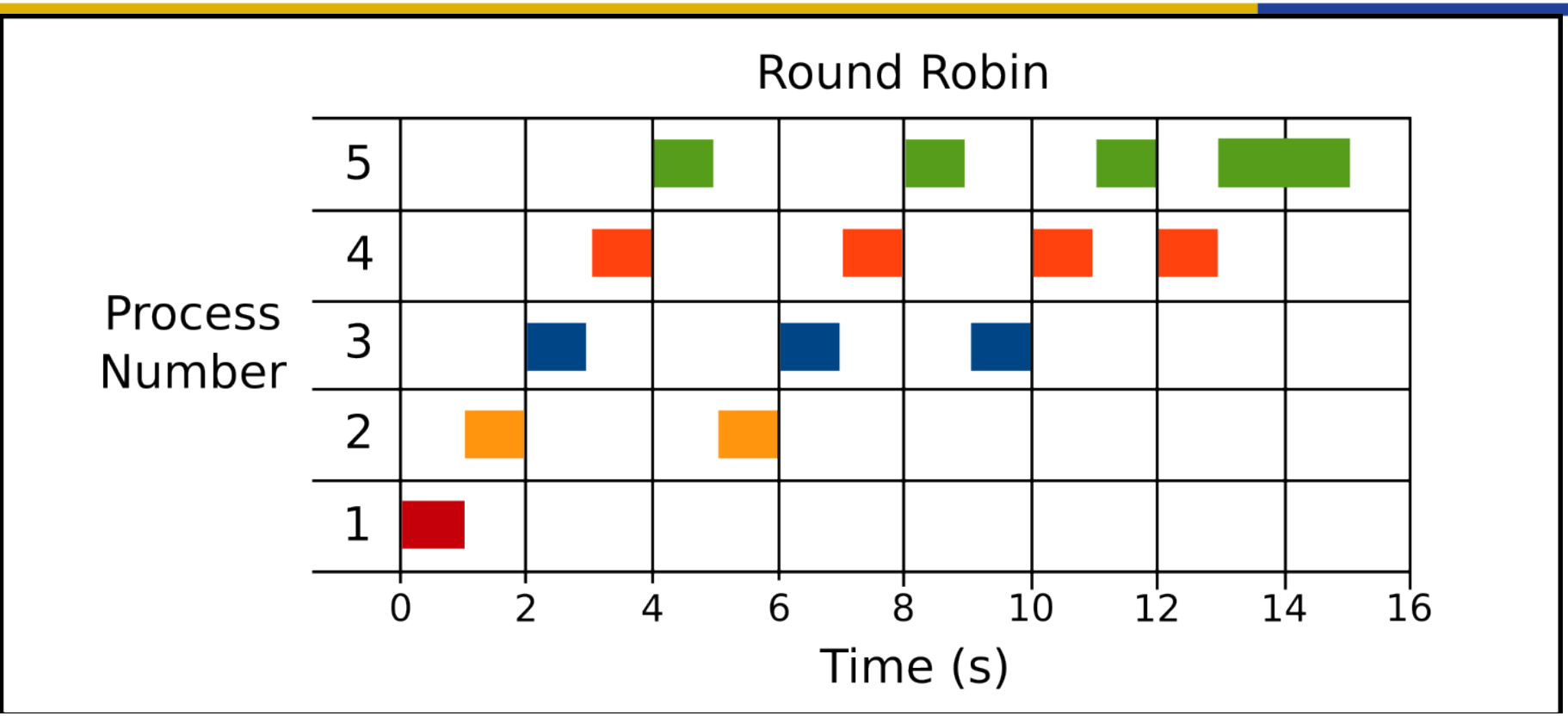
How a Process is Executed?

- A processor usually executes processes in a round-robin fashion.
  - ✓ Many processes wait for execution in a queue.

- The operating system gives a fixed time for the process at the head of the queue.
  - ✓ If the process does not complete execution, it is suspended and the next process is given time.
  - ✓ The suspended process goes to the back of the queue and get scheduled to execute later.

Round-Robin Process Execution

How a Process is Executed?



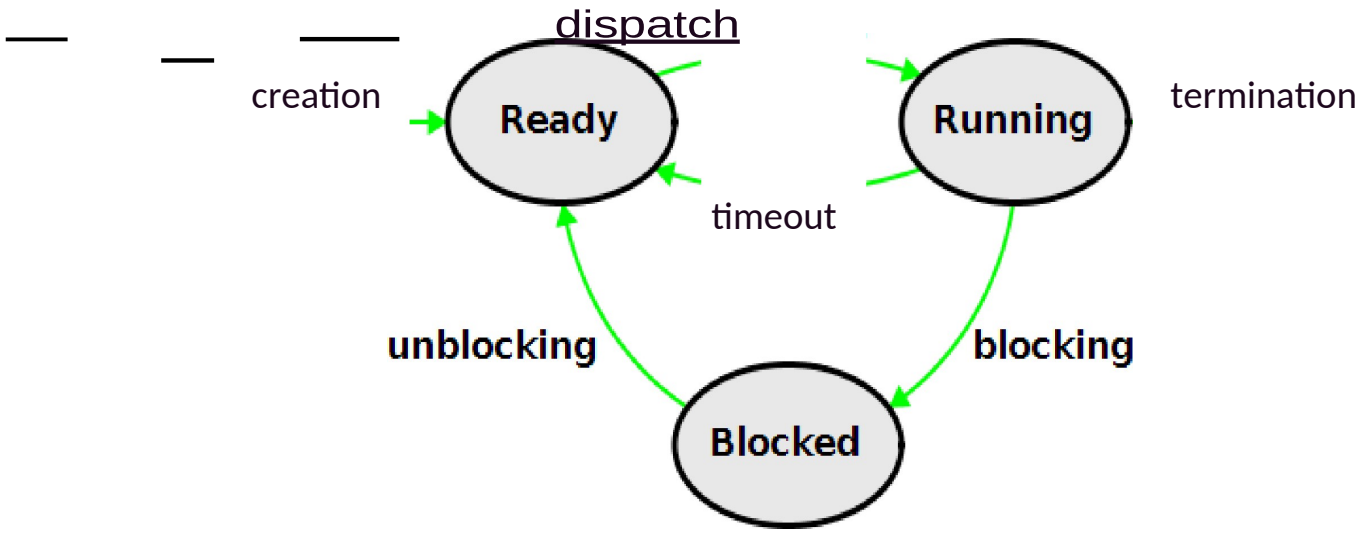
- A processor usually executes processes in a round-robin fashion.
  - ✓ Many processes wait for execution in a queue.

How a Process is Executed?

THE UNIVERSITY OF WESTERN AUSTRALIA

- The operating system gives a fixed time for the process at the head of the queue.
  - ✓ If the process does not complete execution, it is suspended and the next process is given time.
  - ✓ The suspended process goes to the back of the queue and get scheduled to execute later.
  - ✓ The operating system stores the execution status of a process for resuming it later.

Process states



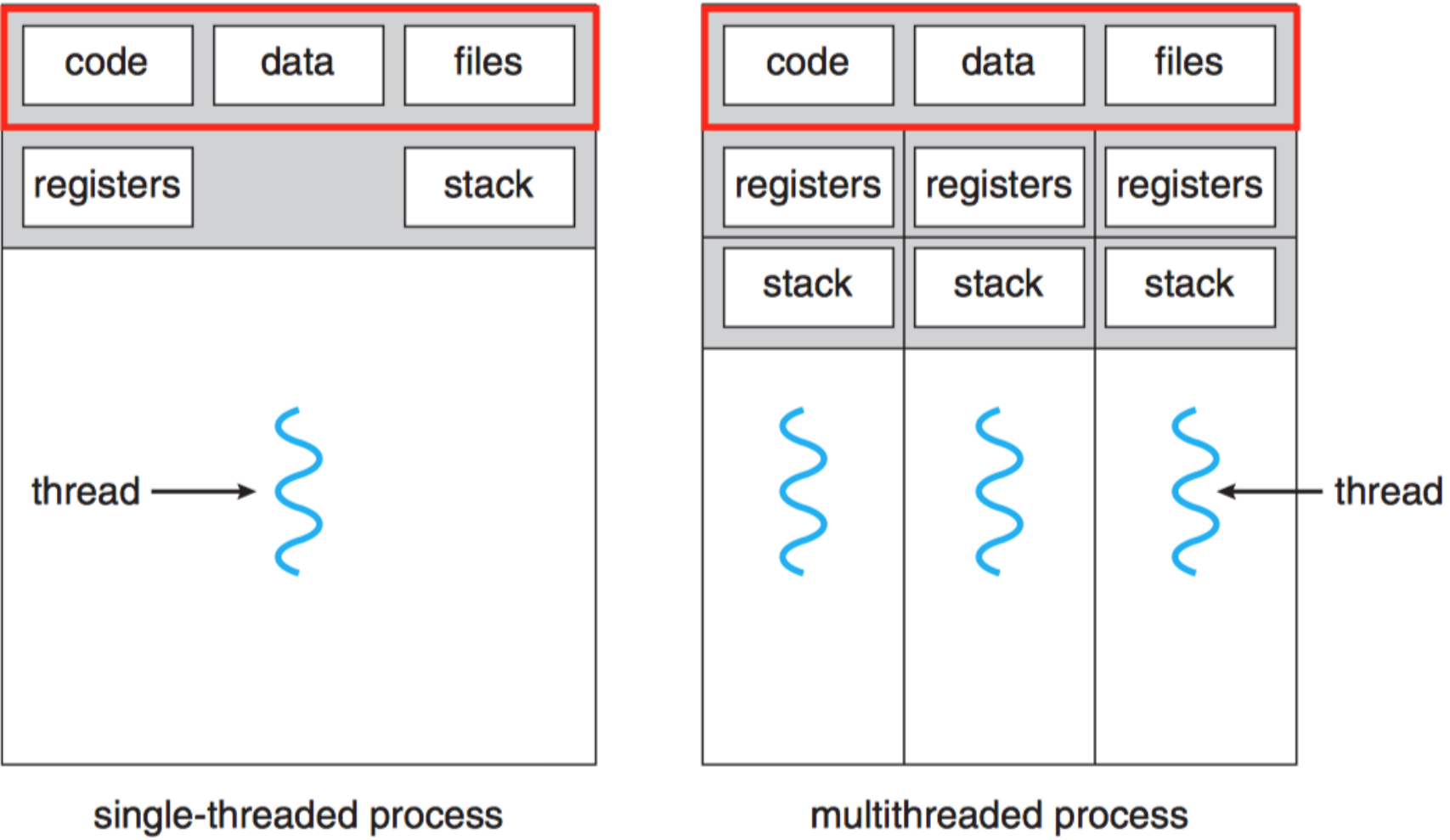
Process and Thread

- A process can be considered as an independent execution environment in a computer system.
- There are usually many processes in a system at any time, each with its own memory space.
- Each process executes a sequence of instructions (the machine language program).
- Threads are also independent execution environments, but with a shared memory space (or address space).

What is a Thread

- Threads are also independent execution environments, but with a shared memory space (or address space).
- Each thread executes its own sequence of instructions (or machine language code), but they can access a set of common memory locations.
- Usually a process creates multiple threads and these threads share the address space of the parent process.

Single and Multiple Threads in a Process



single-threaded process

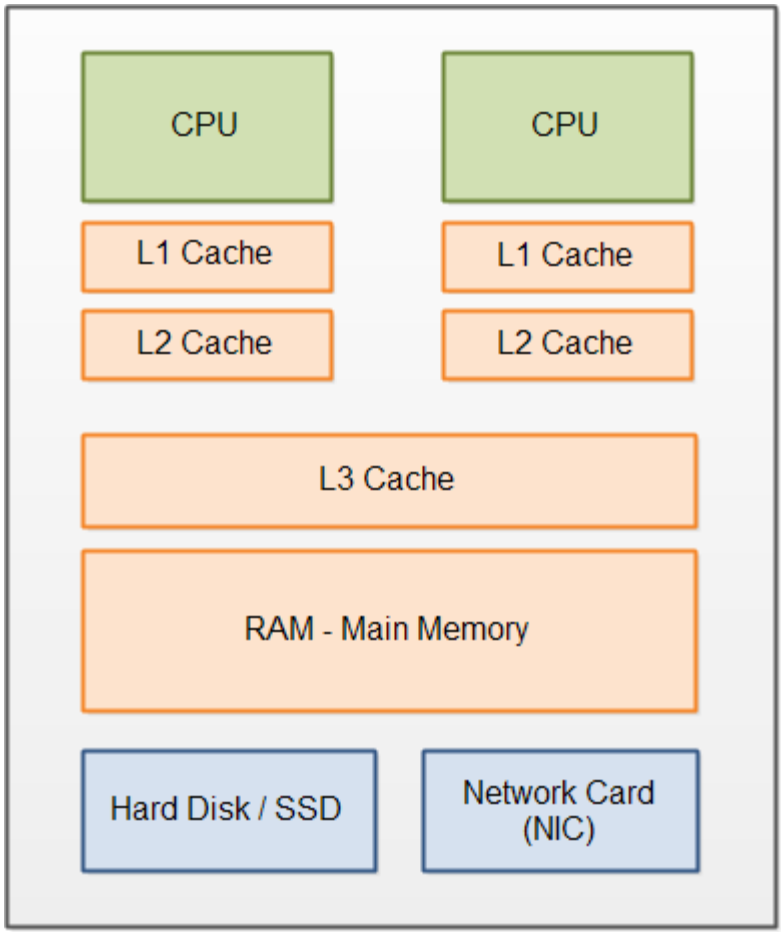
multithreaded process

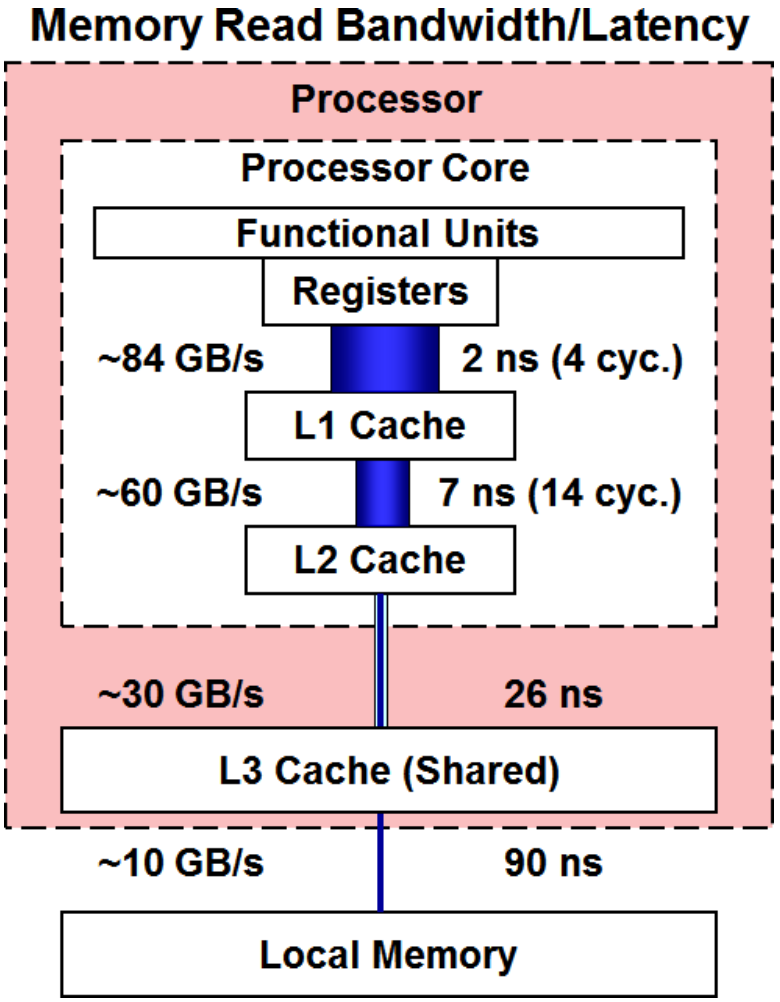
Outline of Today's Lecture

- **What and why we are studying**

- **Computer architecture**
  - ✓ Processor
  - ✓ Process and thread
  - ✓ Memory hierarchy
    - An example on cache line

Memory Hierarchy





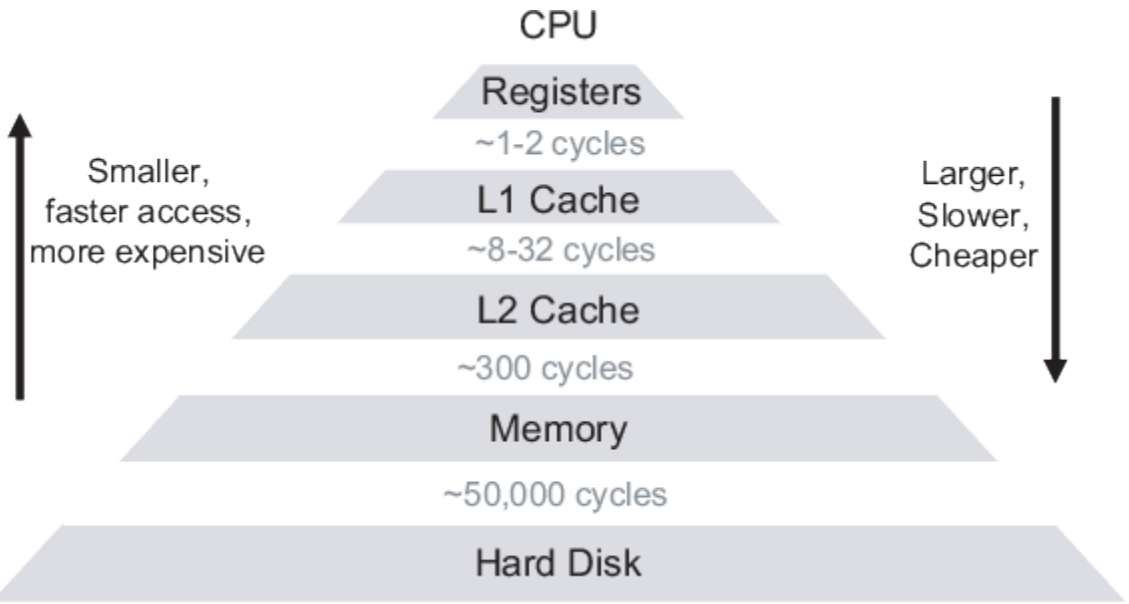
The Memory Hierarchy

THE UNIVERSITY OF WESTERN AUSTRALIA

- Most computers have caches.
  - ✓ The access time for the cache memory is orders of magnitude faster than RAM or hard disks.
  - ✓ Registers < caches < RAM < hard disks
- When the CPU needs an instruction or data from the RAM, the cache is looked up first.
  - ✓ If the item is present in cache, the memory access is very fast.
- How fast a process will run depends on how much of the memory access is in the cache.

The Memory Hierarchy

- Why not build computers entirely with fast memories like the cache?
  - ✓ The cost will be high, consumer hardware will not be competitively priced.
  - ✓ The cost is in manufacturing hardware, caches are usually "on-chip" with the processor.



Multi-Level Cache

- **Multi-core processors usually have multiple levels of cache, e.g. L1, L2 and L3.**
  - ✓ Generally, the L1 cache is private for each core, each pair of cores share an L2 cache and all cores share the L3 cache.
  - ✓ L1 cache is the smallest, the L2 cache is larger than the L1 cache but smaller than L3.

- **The memory access pattern is,**
  i. first look-up the L1 cache,
  ii. then the L2 cache if L1 doesn't have the data,
  iii. then the L3 cache if L2 doesn't have the data,
  iv. and finally RAM if the caches don't have the data.

Multi-Core and Multi-Threading

THE UNIVERSITY OF WESTERN AUSTRALIA

- The purpose of multi-core, multi-threaded programming is to make processes run faster.

- We partition the work so that multiple threads can share the work.
  - ✓ More on this later, the entire unit is about this.

- **The performance improvement:**
  - X if multiple threads run on the same CPU core, they will just get round-robin access to the CPU;
  - ✓ if multiple threads run on multiple CPU cores, they will run simultaneously in parallel.

Cache Hit and Cache Miss

THE UNIVERSITY OF WESTERN AUSTRALIA

- Suppose a machine only has a cache and RAM.
- It is a cache hit if the required information is found in the cache, and a cache miss if it is not found in the cache.
- When cache miss, perform the following steps.
  - ✓ The memory access request goes to RAM.
  - ✓ Then, the cache is filled with a block of instructions or data (called a cache line) from RAM. This is a contiguous block of instructions or data adjacent to the request that the CPU made.
  - ✓ The previous block of instructions or data in the cache is over-written (and lost).

Cache Hit and Cache Miss (Analogy)

- A customer orders meal in KFC—require for information
  - The ordered boxed meal is already in a tray—cache hit
  - The customer gets the boxed meal fast
- A customer orders meal in KFC—require for information
  - The ordered boxed meal isn't in a tray—cache miss
  - The order goes to the kitchen—goes to RAM
  - The ordered boxed meal is put in a tray
- When frying chips, they fry 1kg at a time
  - cache line

Principle of Locality

- Ideally, the process will need instructions and data adjacent to the instruction or data that caused the cache miss.
  - This is based on the principle of locality.

- Process execution is mostly localised in code—machine instructions are mostly executed one after another.
  - The exceptions are conditional statements like *if* and *function or procedure calls*, where jumps in process code occur.
  - But these jumps are infrequent.

- Data used by a process is often sequential.

Principle of Locality (continued)

THE UNIVERSITY OF WESTERN AUSTRALIA

- Instructions and data are not accessed randomly.
- **Temporal locality**
  - ✓ Recently accessed items will be accessed in the near future (e.g. code in loops, top of stack)
- **Spatial locality**
  - ✓ Items at addresses close to the addresses of recently accessed items will be accessed in the near future (sequential code, elements of arrays)

Multi-Threading Performance

- When cache access results in a cache-hit, the higher level caches will be filled up.
  - E.g. if there is a cache-hit in the L3 cache, the L2 and L1 caches will filled up with blocks from the L3 cache, depending on their capacities.

- The performance of multi-threaded programs notably impacts by how they utilise the caches.

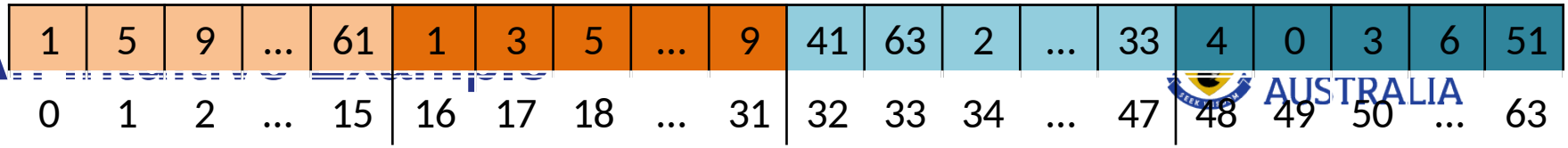
- Multi-threaded programs may run slower than single-threaded (or sequential) programs if the programs are not carefully written or data structures are not carefully designed.

Outline of Today's Lecture

THE UNIVERSITY OF WESTERN AUSTRALIA

- **What and why we are studying**

- **Computer architecture**
  - ✓ Processor
  - ✓ Process and thread
  - ✓ Memory hierarchy
    - An example on cache line



- Consider a simple example of adding the numbers stored in an array.

- Suppose the array is: int a[64]
  - ✓ Change **64** to a large number (e.g. 100,000) in your code to observe more significant differences.

- A natural way of adding the elements in this array is to use two threads, t_1 and t_2. Thread t_1 will add the elements a[0…31] and t_2 the elements a[32…63]

Thread $\mathbf{t_1}$ Thread $\mathbf{t_2}$

Value

Index

An Intuitive Example

- Consider a simple example of adding the numbers stored in an array.

- Suppose the array is: int a[64]
  - ✓ Change **64** to a large number (e.g. 100,000) in your code to observe more significant differences.

- A natural way of adding the elements in this array is to use two threads, t_1 and t_2. Thread t_1 will add the elements a[0…31] and t_2 the elements a[32…63]

- The program should run nearly twice faster, as the work is halved between the two threads.
- So we write a program where the two threads run on two cores that share an L2 cache.

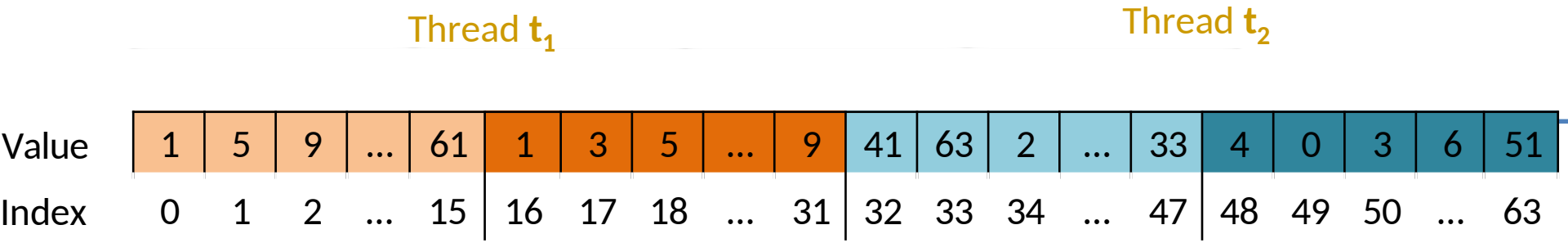
An Intuitive Example—Poor Performance

THE UNIVERSITY OF WESTERN AUSTRALIA

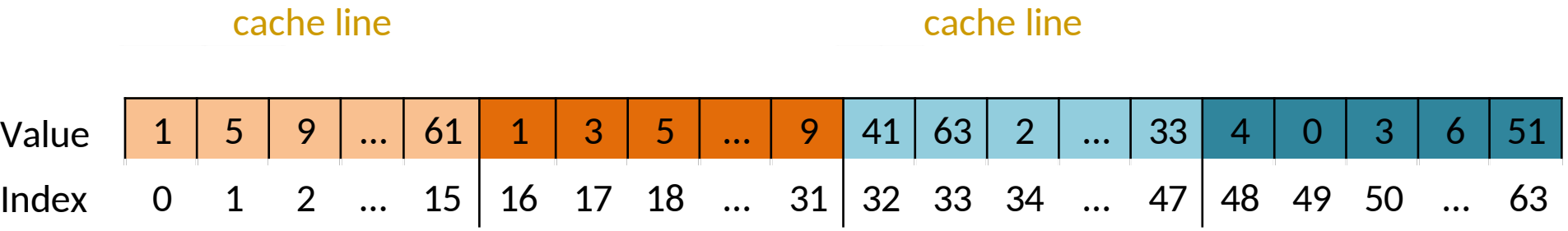
- Threads t_1 and t_2 execute different loops:
  - ✓ t_1: for(i=0;i<32;i++)
    localSum1 = localSum1+a[i];
  - ✓ t_2: for(i=32;i<64;i++)
    localSum2=localSum2+a[i];
- Finally, one of t_1 or t_2 computes the sum:
  sum=localSum1+localSum2;

- Though it seems a beautiful partition of work between the two threads, this will result in **very poor performance**.
  - ✓ The program will run much slower compared to the sequential program (just one thread).

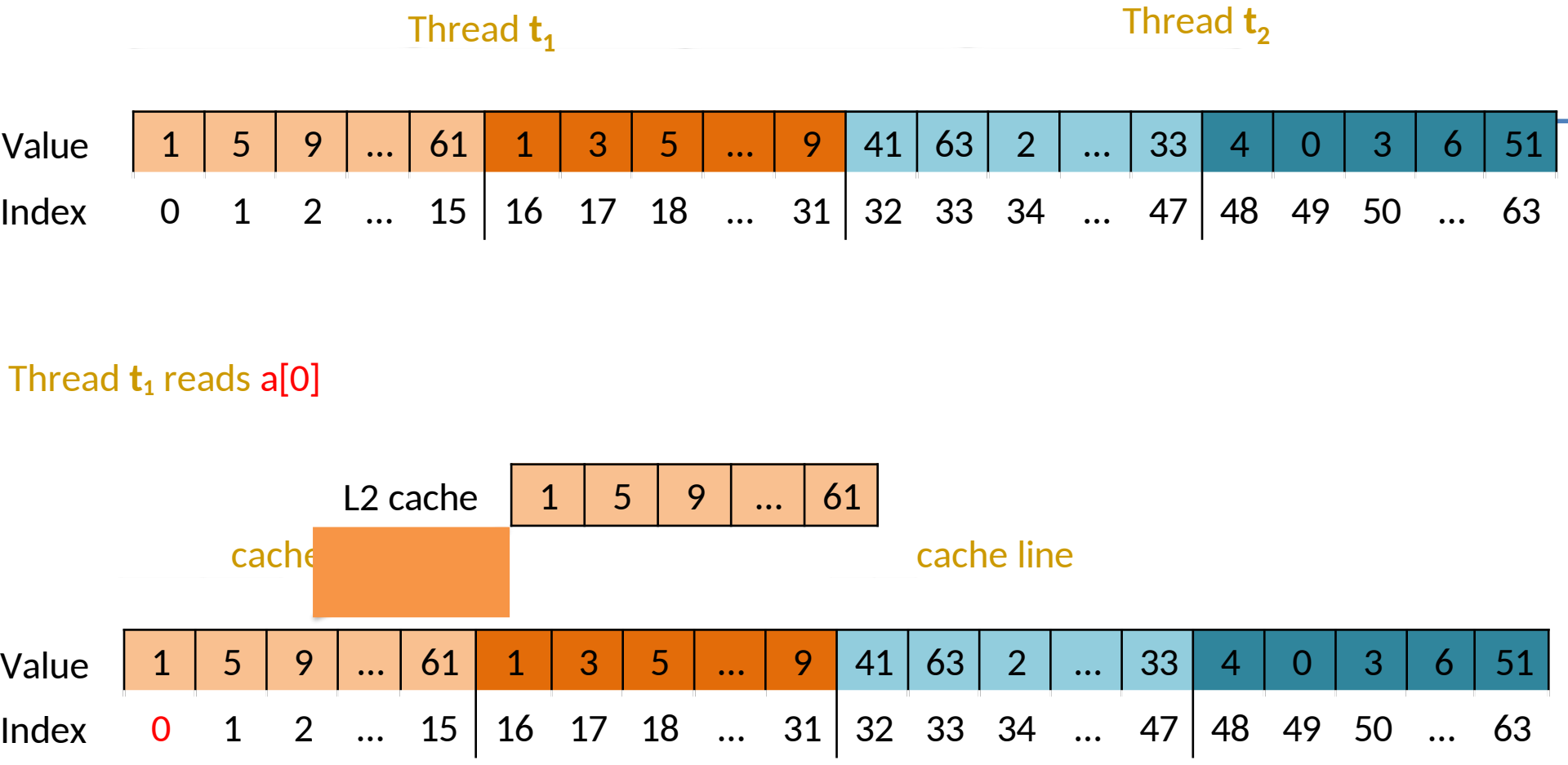
An Intuitive Example—Cache Line



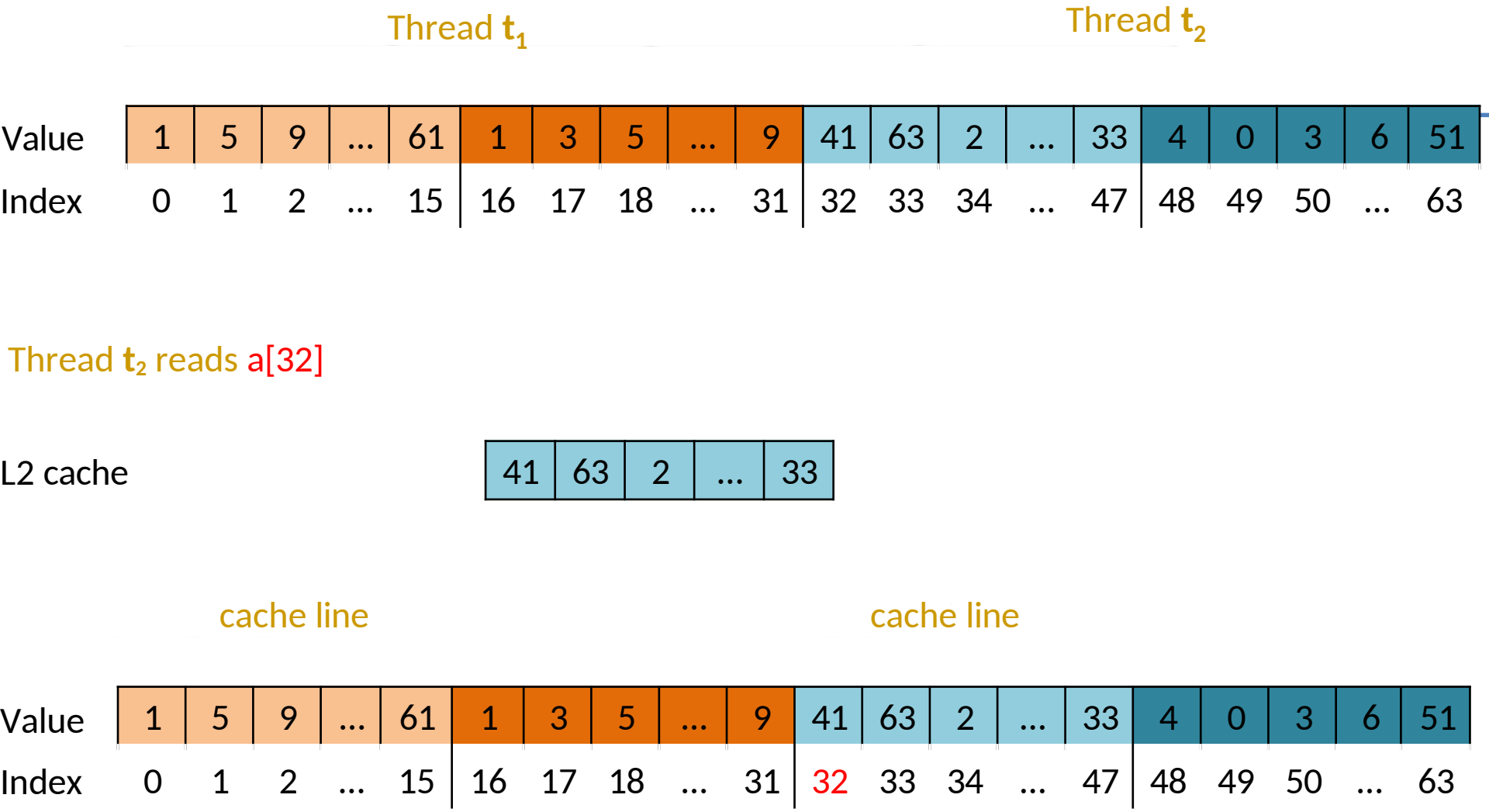
L2 cache is empty at the beginning.



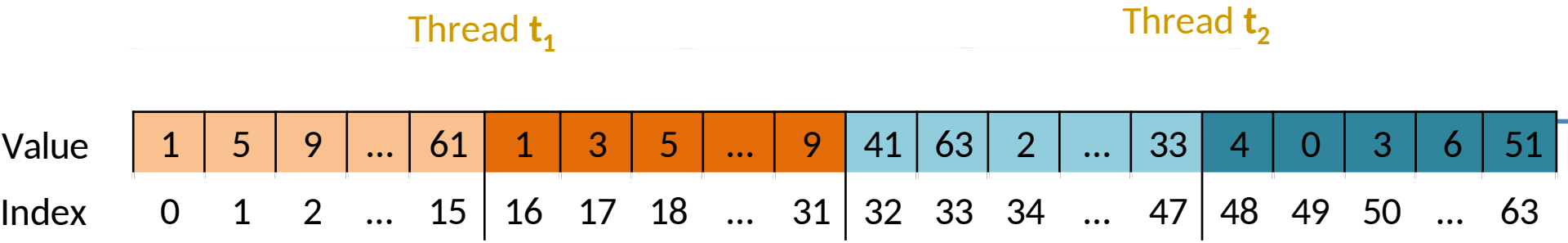
An Intuitive Example—Cache Line



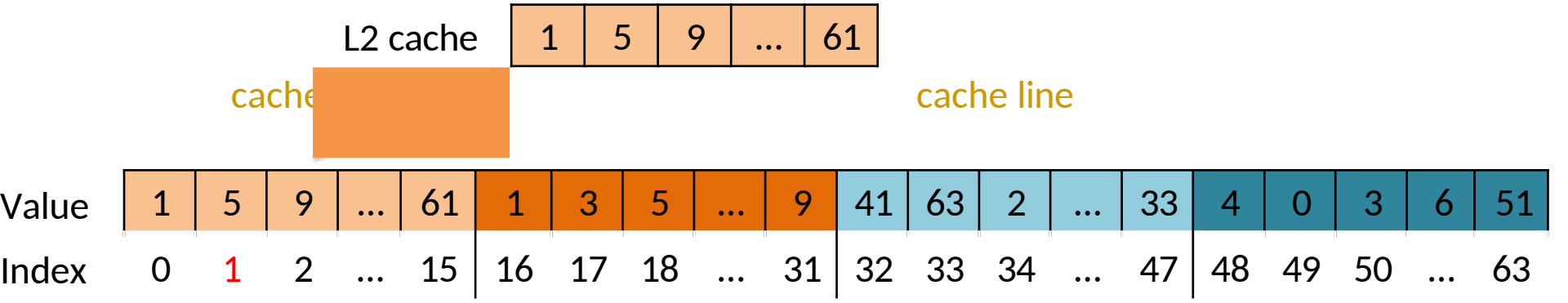
An Intuitive Example—Cache Line



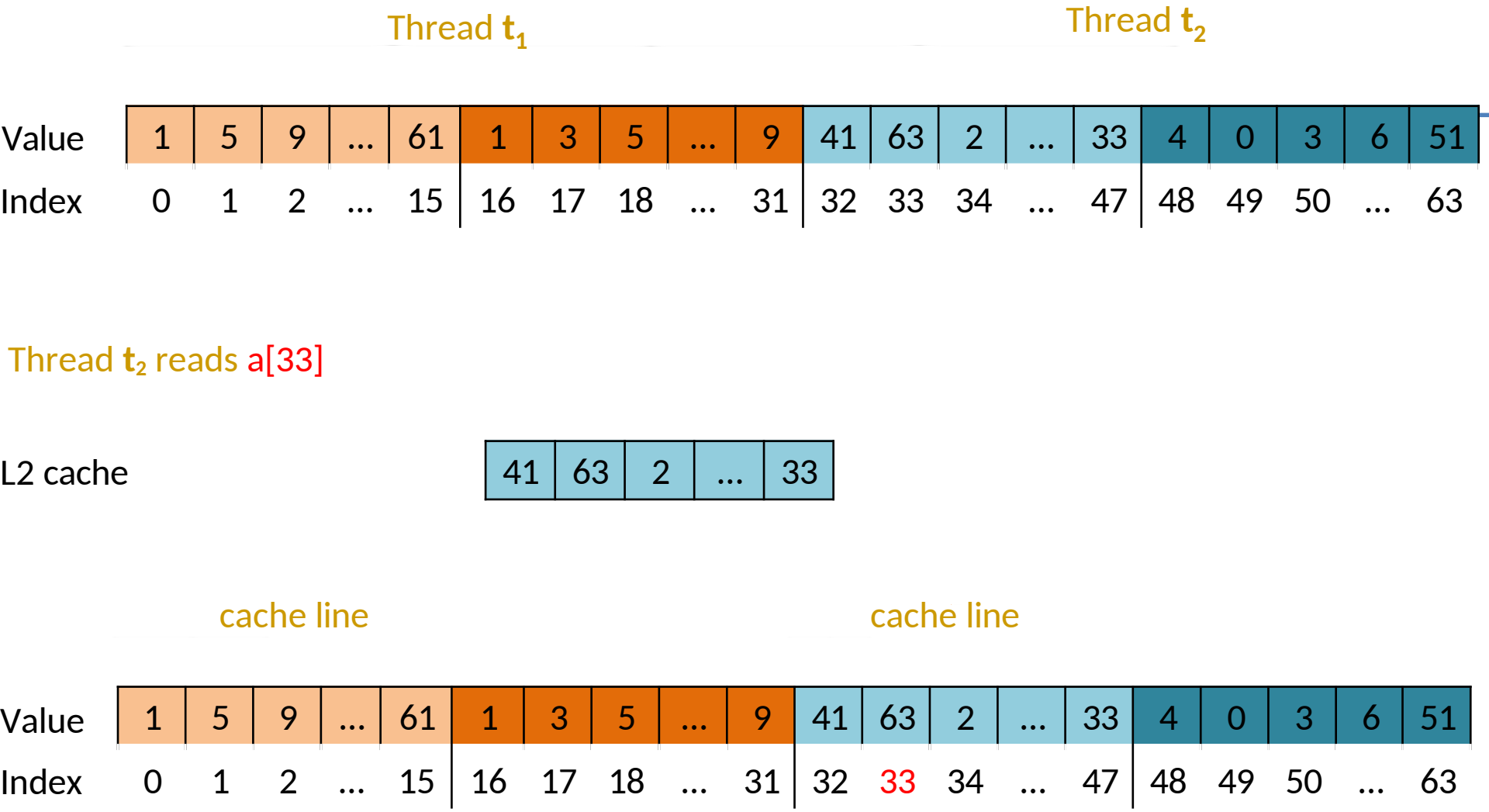
An Intuitive Example—Cache Line



Thread t_1 reads a[1]



An Intuitive Example—Cache Line



An Intuitive Example

THE UNIVERSITY OF WESTERN AUSTRALIA

- When t_1 executes the first iteration of its loop, elements a[0]…a[15] will be brought to the L2 cache.
- When t_2 executes the first iteration of its loop, elements a[32]…a[47] will brought to the L2 cache.
- This ping-pong effect continues and the cache is filled again and again by the elements each thread requires.
- In effect, almost every iteration of each thread's loop results in a cache-miss and the overall execution time is much longer compared to the sequential code.

1	5	9	…	61	1	3	5	…	9	41	63	2	…	33	4	0	3	6	51
0	1	2	…	15	16	17	18	…	31	32	33	34	…	47	48	49	50	…	63

- In a sequential program (a single thread):
  - ✓ Element a[0] will be accessed in the first iteration of the loop.
  - ✓ However, this will be a cache-miss (as the cache is empty in the beginning) and hence elements a[0]..a[15] will be brought to the L2 cache.
  - ✓ The next 15 elements can now be accessed from the L2 cache, i.e. there will be 15 consecutive cache-hits after this.
  - ✓ This will result in very fast execution of the sequential algorithm.

Thread t_1

Value

Index

References

- Readings
  - Moore’s Law Visualisation (1971 to 2019)
  - Memory Hierarchy
  - Concurrency v.s. Parallelism

Copyright Notice