Memory Hierarchy:

Memory Technology and Principal of Locality

by: Tassadaq Hussain

Director Centre for AI and BigData

Professor Department of Electrical Engineering

Namal University Mianwali

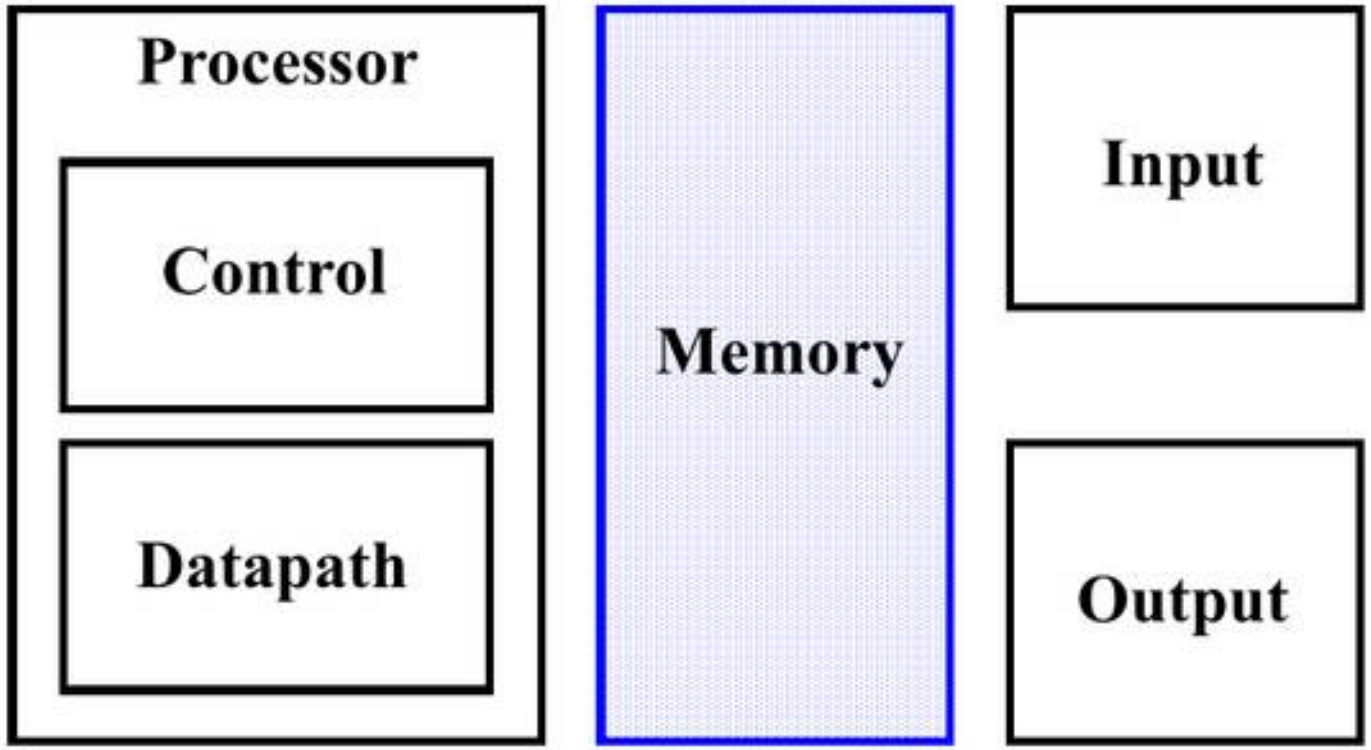
Collaborations:

Barcelona Supercomputing Center, Spain

European Network on High Performance and Embedded Architecture and Compilation

Pakistan Supercomputing Center

Contents

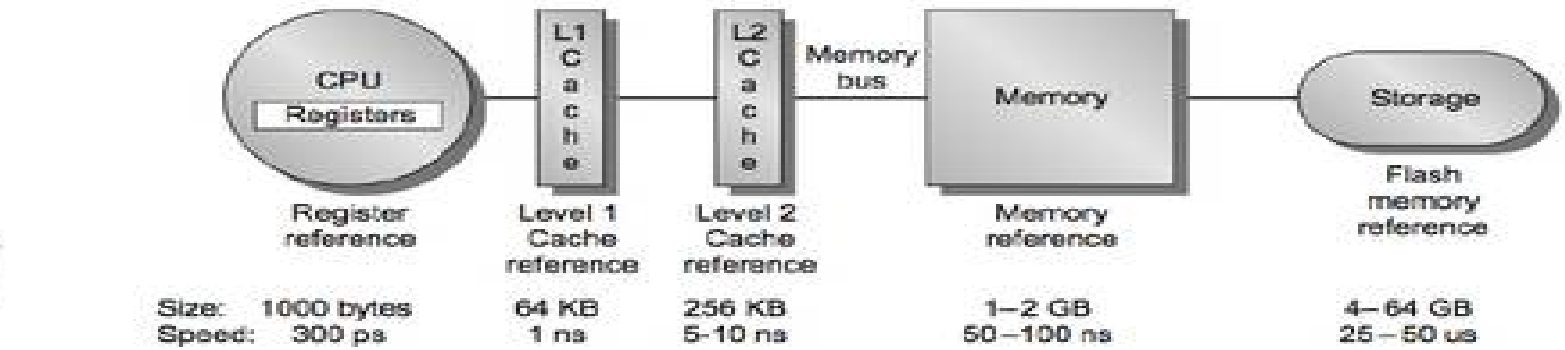


Memory system

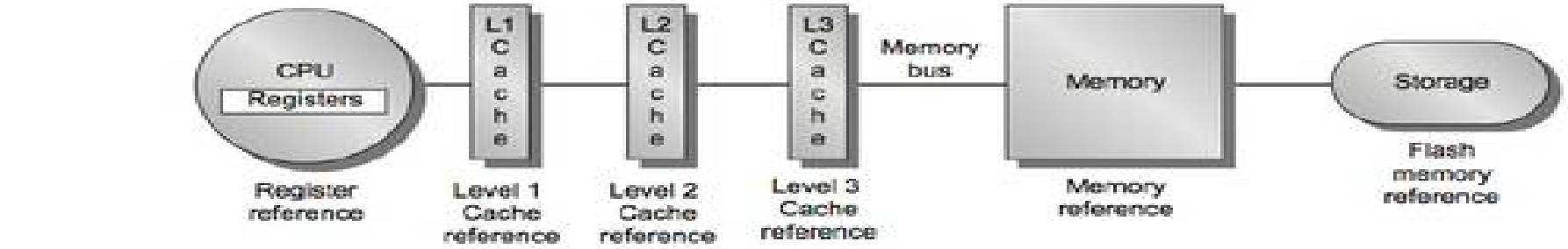
– Supplying data on time for computation (speed)

– Large enough to hold everything needed (capacity)

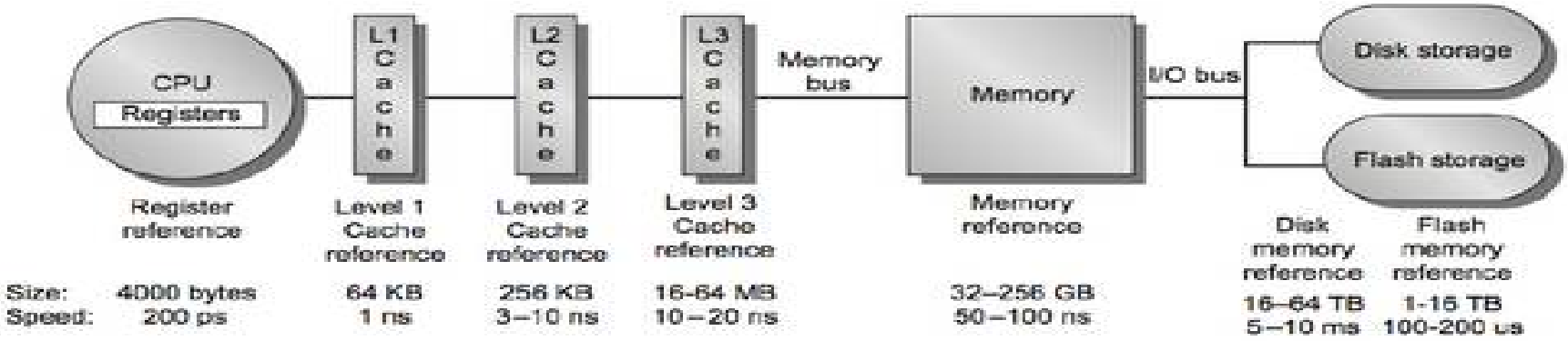
Memory Hierarchy



(A) Memory hierarchy for a personal mobile device



(B) Memory hierarchy for a laptop or a desktop



(C) Memory hierarchy for server

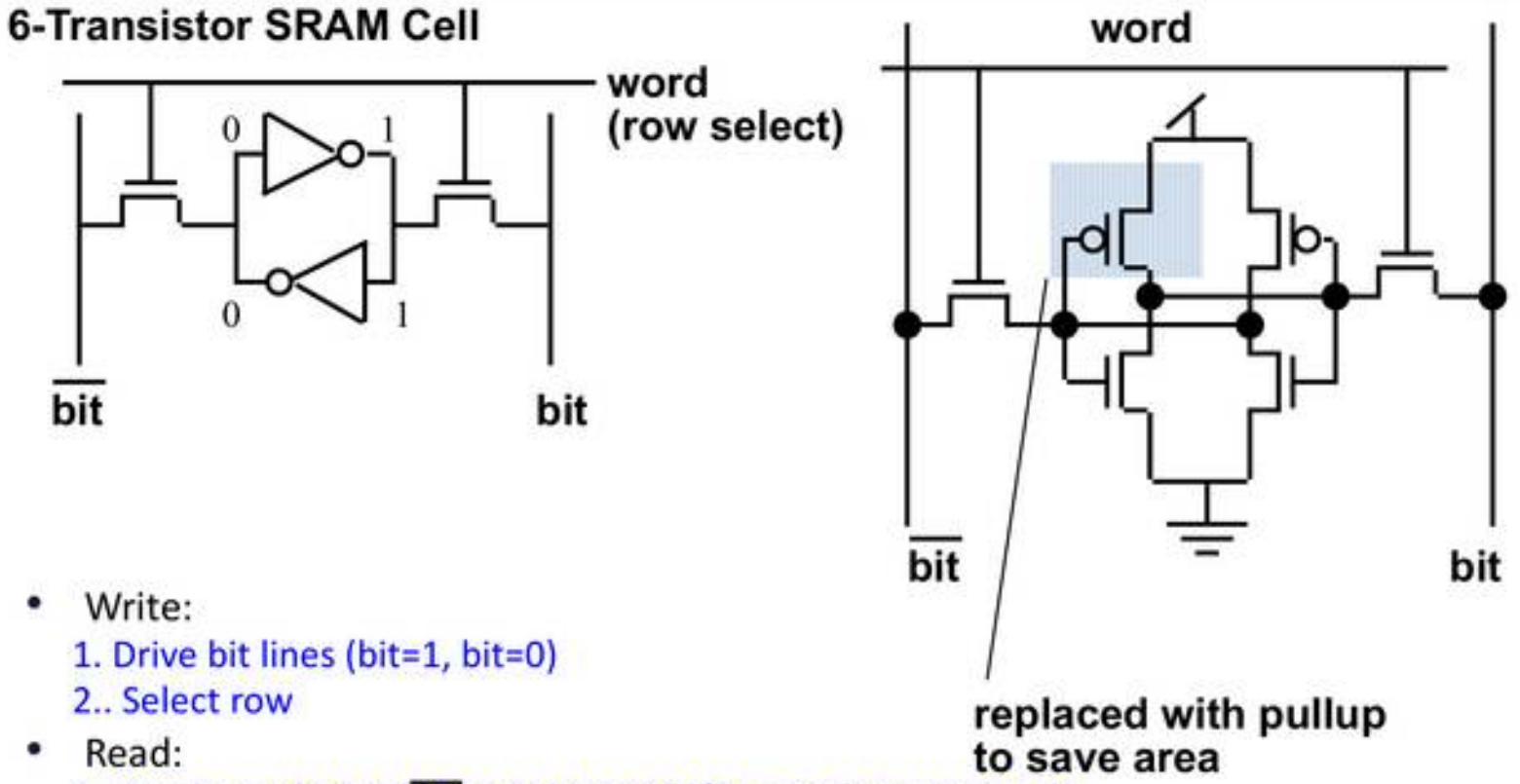
Memory Technology

- **Random Access: access time is the same for all locations**
- **DRAM: Dynamic Random Access Memory**
  - **High density, low power, cheap, slow**
  - **Dynamic: need to be "refreshed" regularly**
  - **50ns – 70ns, $20 – $75 per GB**
- **SRAM: Static Random Access Memory**
  - **Low density, high power, expensive, fast**
  - **Static: content will last "forever"(until lose power)**
  - **0.5ns – 2.5ns, $2000 – $5000 per GB**

- Magnetic disk
  - 5ms – 20ms, $0.20 – $2 per GB

Ideal memory:
- **Access time of SRAM**
- **Capacity and cost/GB of disk**

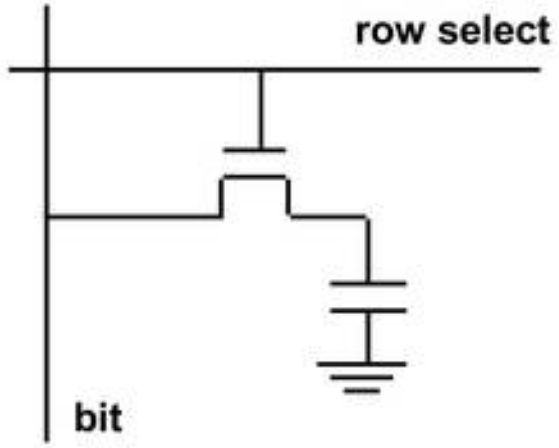
Static RAM (SRAM) 6-Transistor Cell – 1 Bit



- Write:
  1. Drive bit lines (bit=1, bit=0)
  2.. Select row
- Read:
  1. Precharge bit and $\overline{bit}$ to Vdd or Vdd/2 => make sure equal!
  2.. Select row
  3. Cell pulls one line low
  4. Sense amp on column detects difference between bit and $\overline{bit}$

DRAM Cell Architecture

- Write:
  - 1. Drive bit line
  - 2. Select row
- Read:
  - 1. Precharge bit line to Vdd
  - 2. Select row
  - 3. Cell and bit line share charges
    - Very small voltage changes on the bit line
  - 4. Sense (fancy sense amp)
    - Can detect changes of ~1 million electrons
  - 5. Write: restore the value
- Refresh
  - 1. Just do a dummy read to every cell.



Performance: Latency and Bandwidth

- Performance of Main Memory:
- – Latency: Cache Miss Penalty
  - Access Time: time between request and word arrives
  - Cycle Time: time between requests
- – Bandwidth: I/O & Large Block Miss Penalty (L2)
- Main Memory is DRAM : Dynamic Random Access Memory
- – Needs to be refreshed periodically (8 ms)
- – Addresses divided into 2 halves (Memory as a 2D matrix)
  - RAS or Row Access Strobe and CAS or Column Access Strobe
- Cache uses SRAM : Static Random Access Memory
- – No refresh (6 transistors/bit vs. 1 transistor)
  - Size: DRAM/SRAM 4-8
  - Cost/Cycle time: SRAM/DRAM 8-16

Stacked/Embedded DRAMs

- Stacked DRAMs in same package as processor
  - High Bandwidth Memory (HBM)

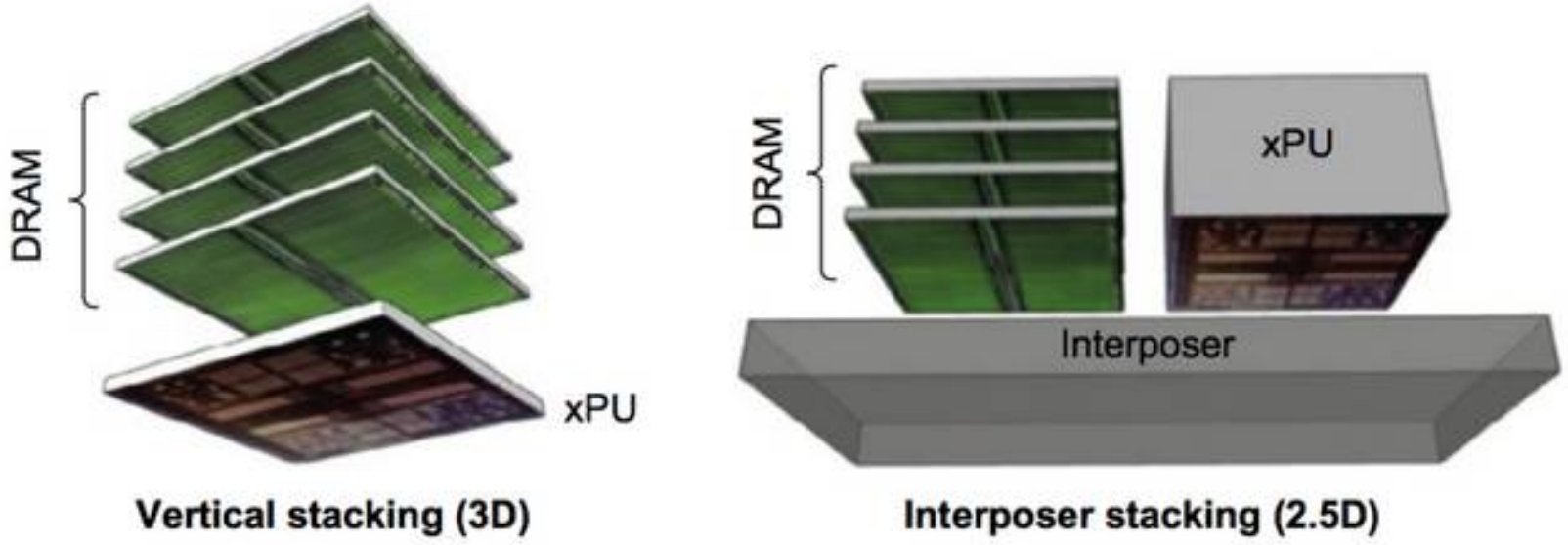
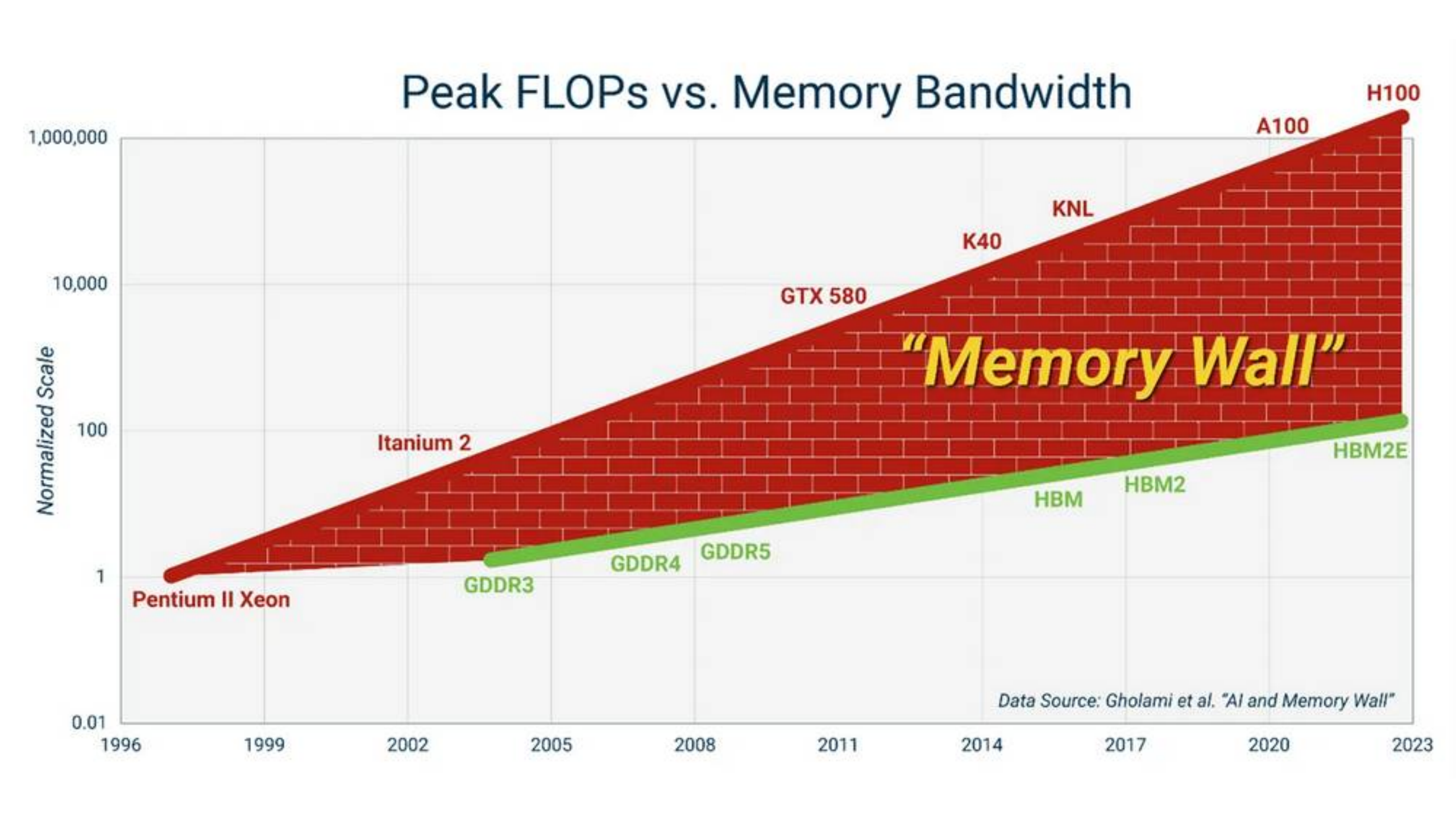


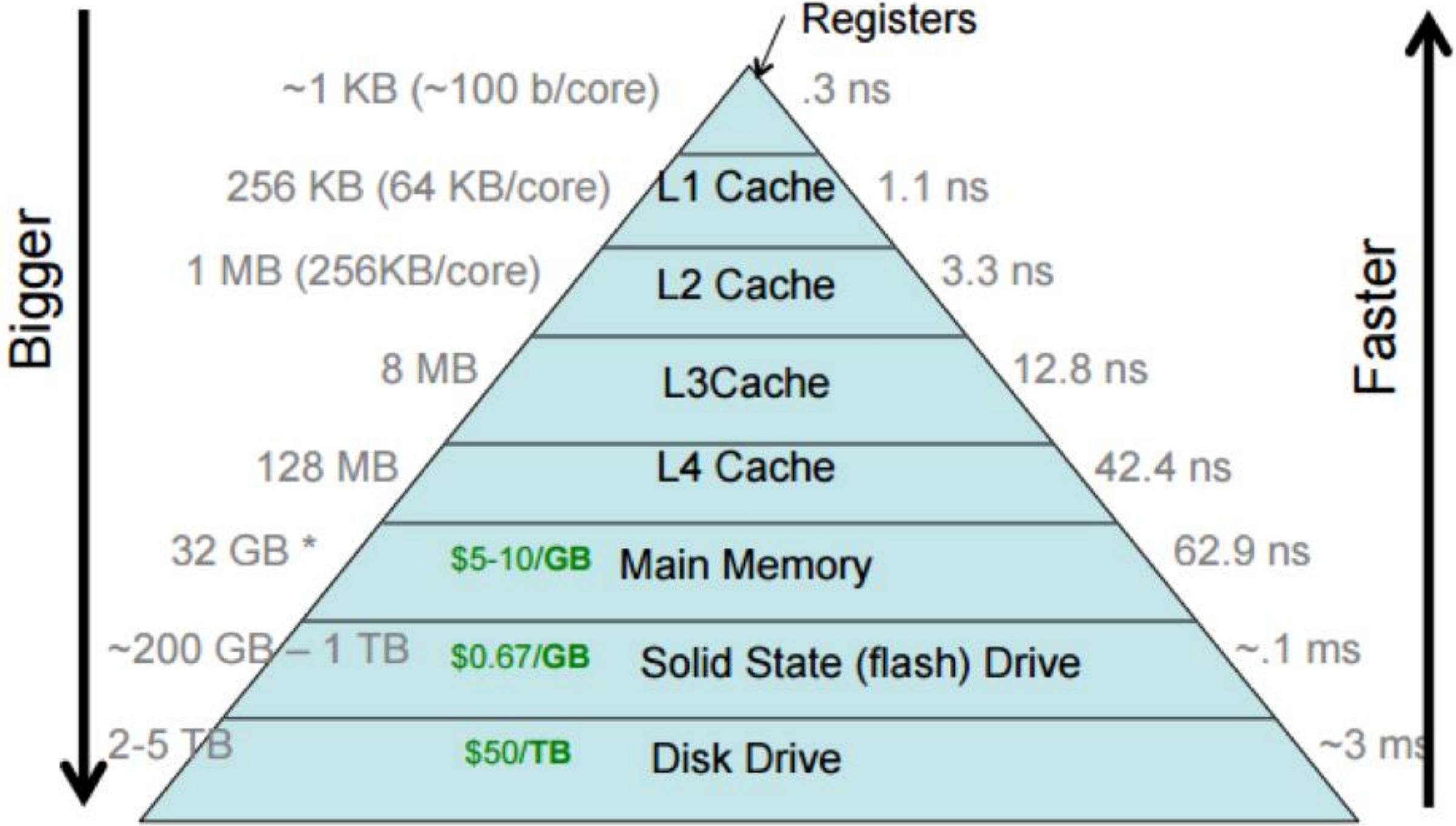
Figure Two forms of die stacking. The 2.5D form is available now. 3D stacking is under development and faces heat management challenges due to the CPU.

Flash Memory

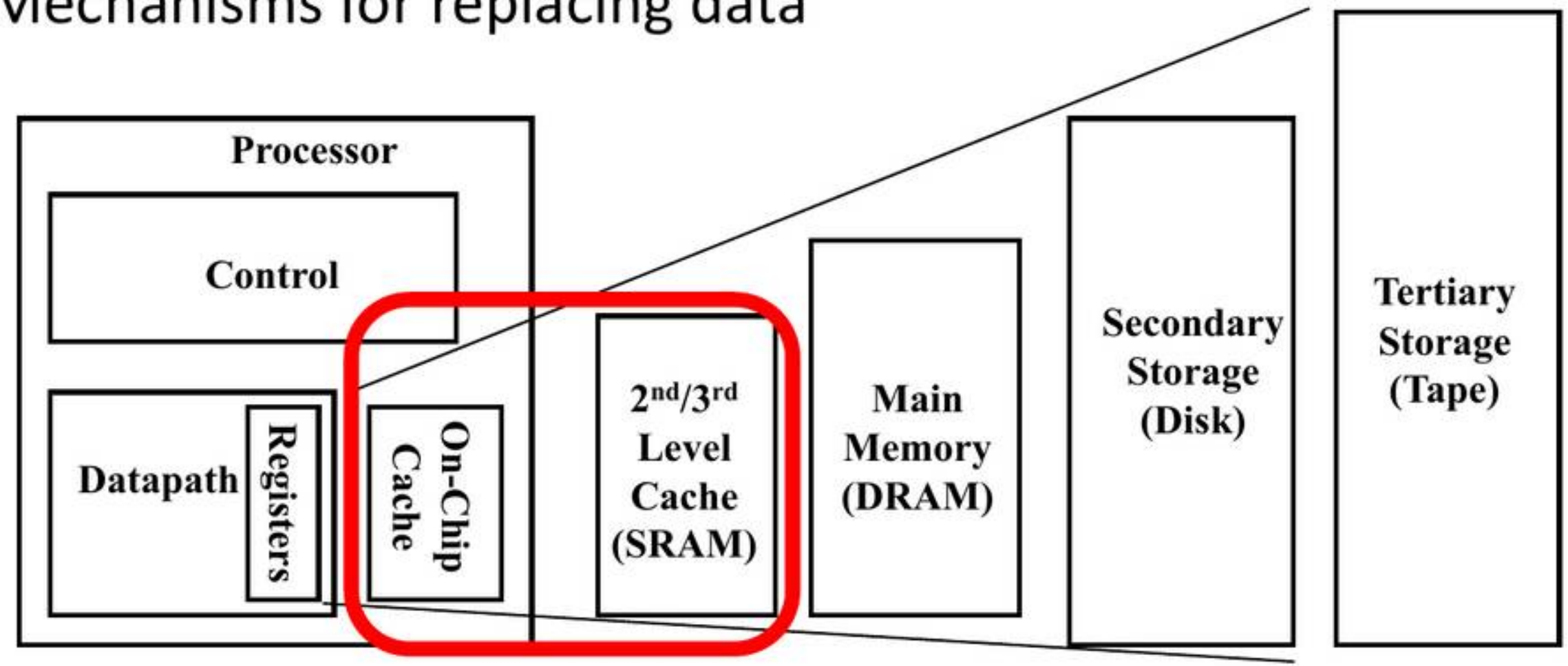
- Type of EEPROM
- Types: NAND (denser) and NOR (faster)
- NAND Flash:
  - Reads are sequential, reads entire page (.5 to 4 KiB)
  - 25 us for first byte, 40 MiB/s for subsequent bytes
  - SDRAM: 40 ns for first byte, 4.8 GB/s for subsequent bytes
  - 2 KiB transfer: 75 uS vs 500 ns for SDRAM, 150X slower
  - 300 to 500X faster than magnetic disk


Peak FLOPs vs. Memory Bandwidth
H100
A100
KNL
K40
GTX 580
"Memory Wall"
Itanium 2
HBM2E
HBM2
HBM
GDDR5
GDDR4
GDDR3
Pentium II Xeon
Data Source: Gholami et al. "AI and Memory Wall"
Normalized Scale
1,000,000
10,000
100
1
0.01
1996
1999
2002
2005
2008
2011
2014
2017
2020
2023

- Memory hierarchy design becomes **more crucial with recent multi-core processors:**
- Aggregate peak bandwidth grows with # cores:
  - Intel Core i7 can generate two references per core per clock
  - Four cores and 3.2 GHz clock
    - 25.6 billion 64-bit data references/second +
    - 12.8 billion 128-bit instruction references/second
    - = 409.6 GB/s!
  - DRAM bandwidth is only 8% of this (34.1 GB/s)
- Requires:
  - Multi-port, pipelined caches
  - Two levels of cache per core
  - Shared third-level cache on chip


Registers
~1 KB (~100 b/core)
.3 ns
256 KB (64 KB/core)
L1 Cache
1.1 ns
1 MB (256KB/core)
L2 Cache
3.3 ns
8 MB
L3Cache
12.8 ns
128 MB
L4 Cache
42.4 ns
32 GB *
$5-10/GB
Main Memory
62.9 ns
~200 GB – 1 TB
$0.67/GB
Solid State (flash) Drive
~.1 ms
2-5 TB
$50/TB
Disk Drive
~3 ms
Bigger
Faster

- **Keep most recent accessed data and its adjacent data in the smaller/faster caches that are closer to processor**
- Mechanisms for replacing data



Principle of Locality

- Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves
- **Spatial locality:** Items with nearby addresses tend to be referenced close together in time
- **Temporal locality:** Recently referenced items are likely to be referenced in the near future

```
sum = 0;
for(i=0; i<n; i++)
  sum += a[i];
return sum;
```

- **Data**
  - Reference array elements in succession (stride-1 reference pattern): Spatial Locality
  - Reference sum each iteration: Temporal Locality
- **Instructions**
  - Reference instructions in sequence: Spatial Locality
  - Cycle through loop repeatedly: Temporal Locality

Memory Characteristics

- Location
- Capacity
- Unit of transfer
- Access method
- Performance
- Physical type
- Physical characteristics
- Organisation

Location

- CPU
- Internal
- External

Unit of Transfer

- Internal
  - Usually governed by data bus width
- External
  - Usually a block which is much larger than a word
- Addressable unit
  - Smallest location which can be uniquely addressed

Access Methods (1)

- Sequential
  - Start at the beginning and read through in order
  - Access time depends on location of data and previous location
  - e.g. tape
- Direct
  - Individual blocks have unique address
  - Access is by jumping to vicinity plus sequential search
  - Access time depends on location and previous location
  - e.g. disk

Access Methods (2)

- Random
  - Individual addresses identify locations exactly
  - Access time is independent of location or previous access
  - e.g. RAM
- Associative
  - Data is located by a mechanism based on placement
  - Access time is independent of location or previous access
  - e.g. cache

Performance

- Access time
  - Time between presenting the address and getting the valid data
- Transfer Rate
  - Rate at which data can be moved

Physical Types

- Semiconductor
  - RAM
- Magnetic
  - Disk & Tape
- Optical
  - CD & DVD
- Others
  - Bubble
  - Hologram

Physical Characteristics

- Switching
- Decay
- Volatility
- Erasable
- Power consumption

Organisation

- Physical arrangement of bits into words
- Not always obvious
- e.g. interleaved

The Bottom Line

- How much?
  - Capacity
- How fast?
  - Time is money
- How expensive?

Hierarchy List

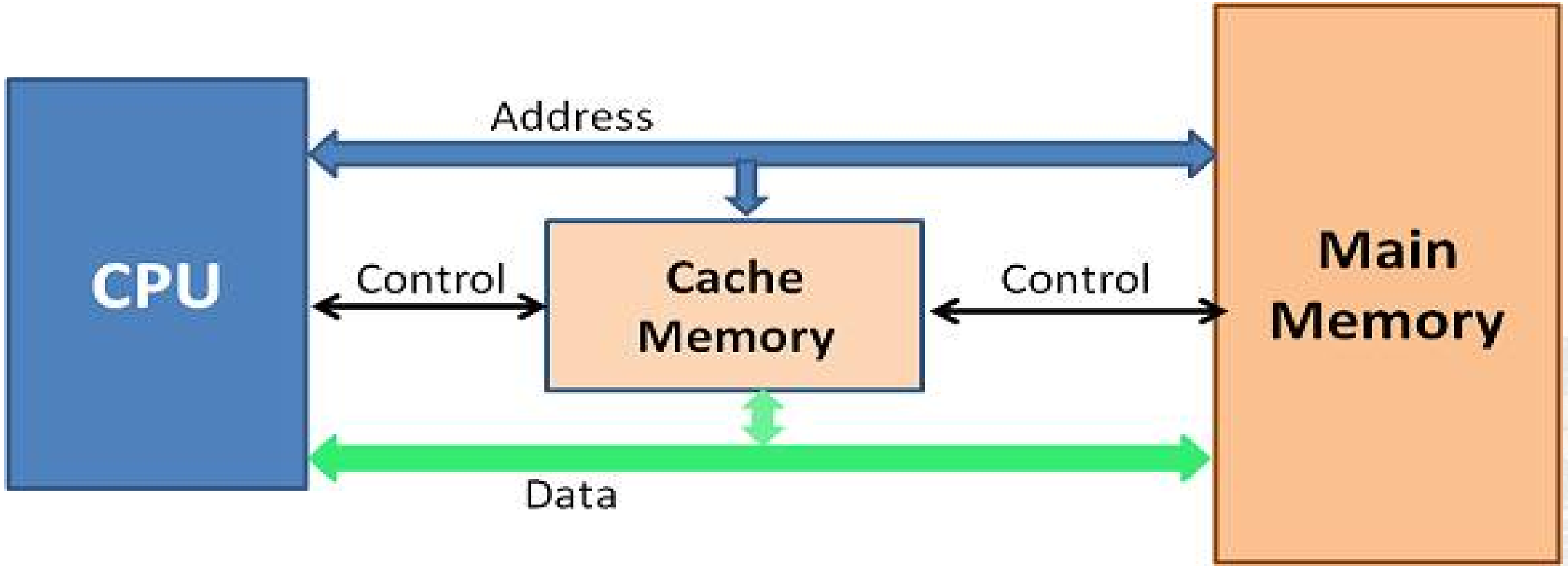
- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache

So you want fast?

- It is possible to build a computer which uses only static RAM (see later)
- This would be very fast
- This would need no cache
  - How can you cache cache?
- This would cost a very large amount

Local Memory System

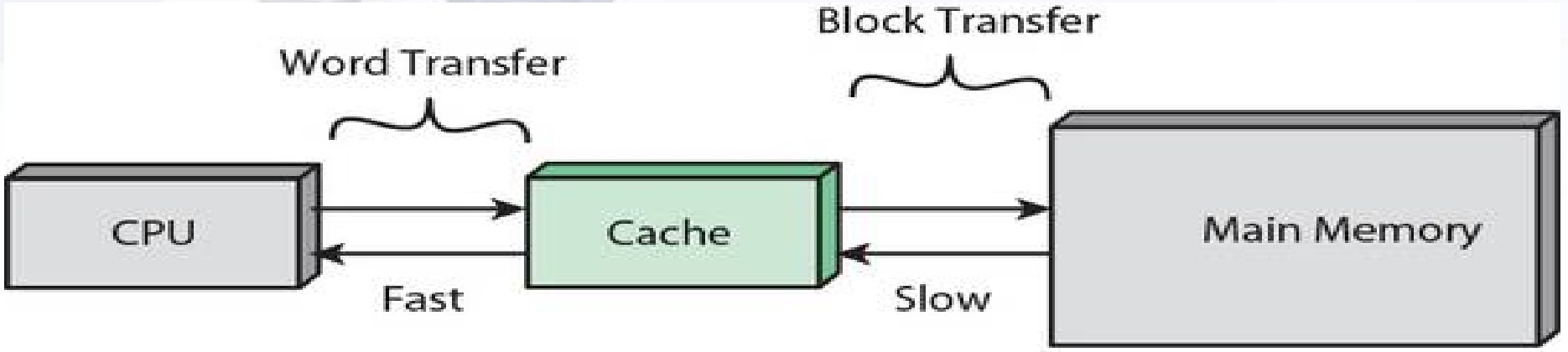
- Cache
- Scratchpad



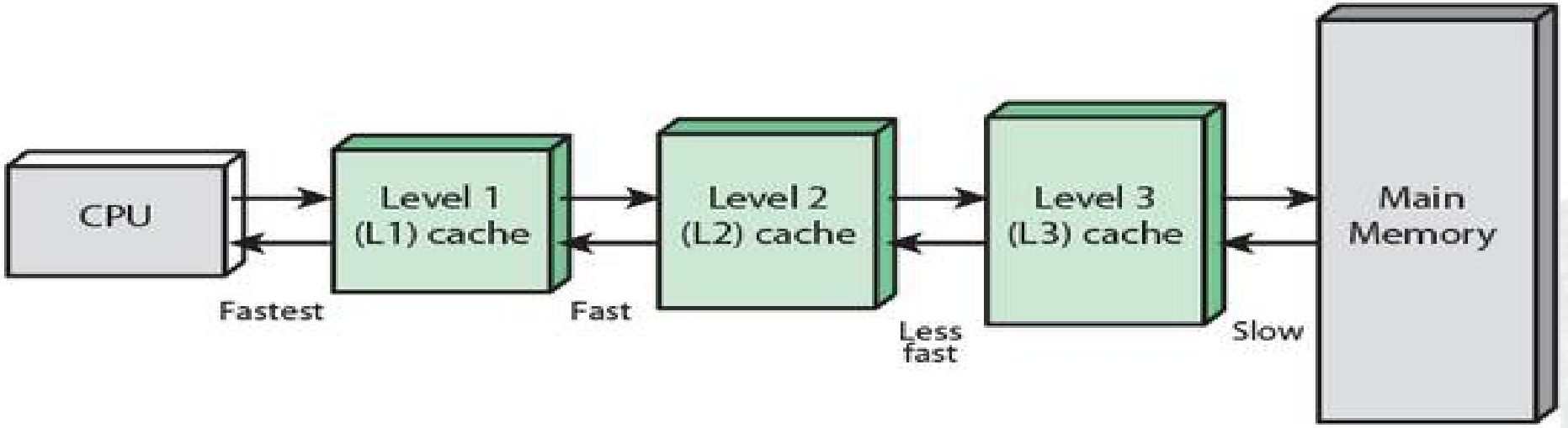
Cache

- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module

•Cache and Main Memory

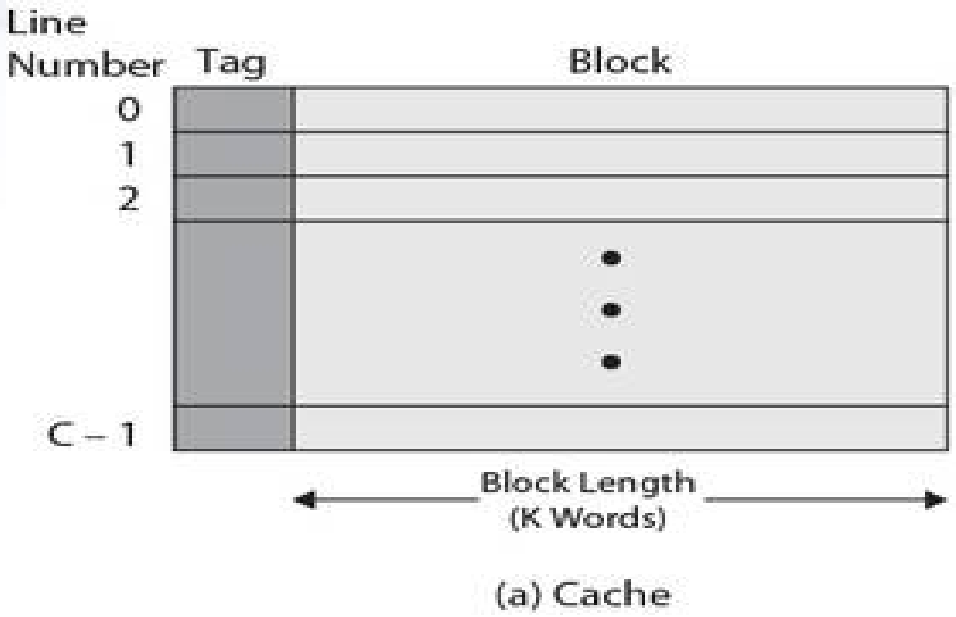


(a) Single cache

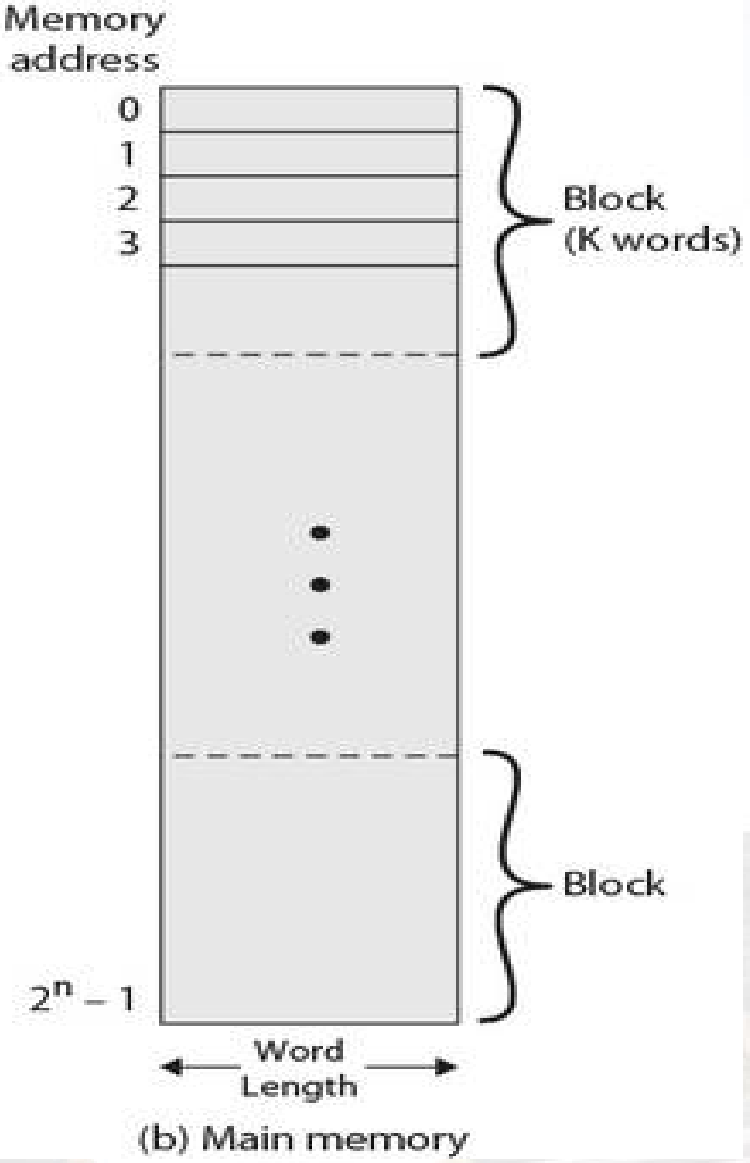


(b) Three-level cache organization

Cache/Main Memory Structure



(a) Cache

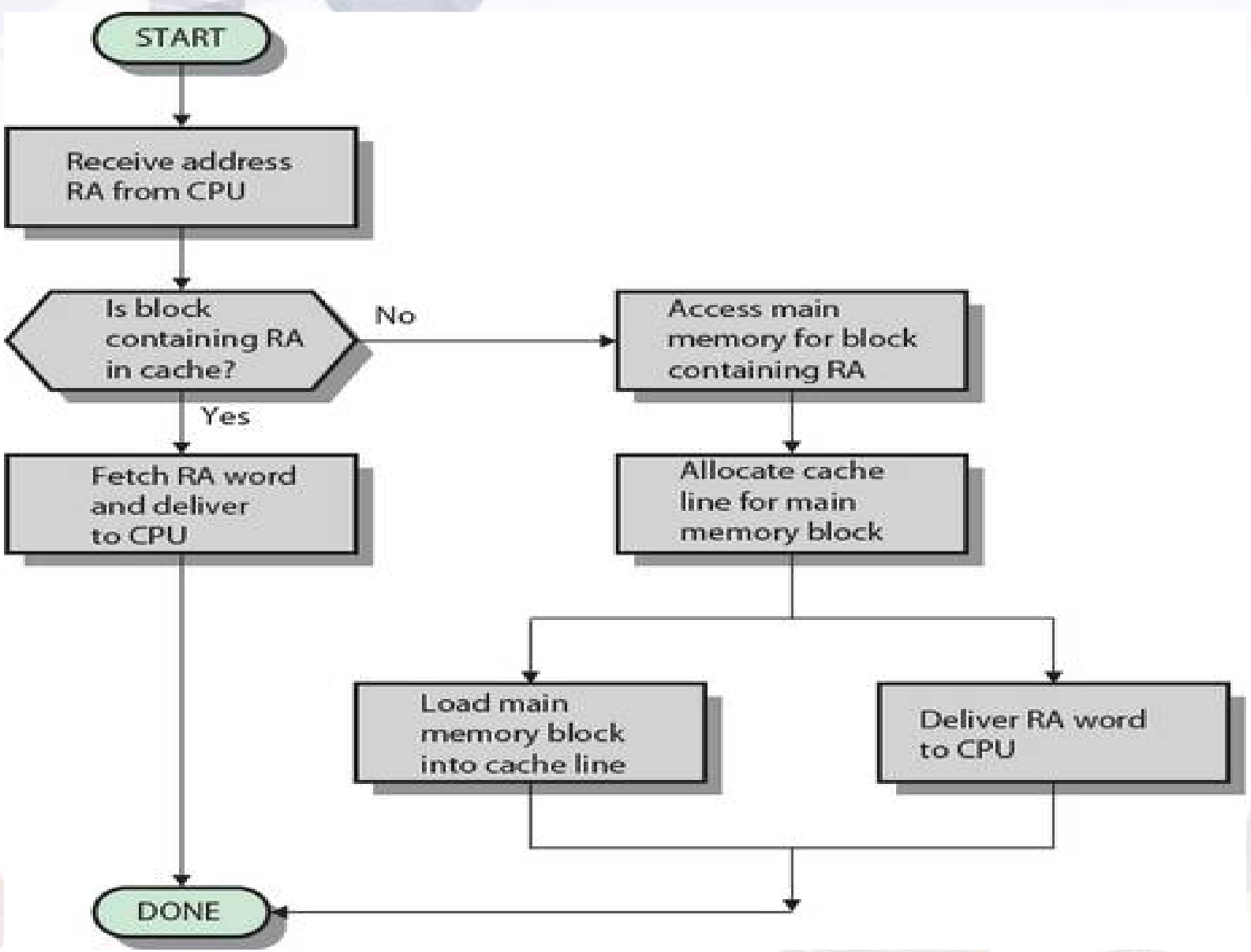


(b) Main memory

Cache operation – overview

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

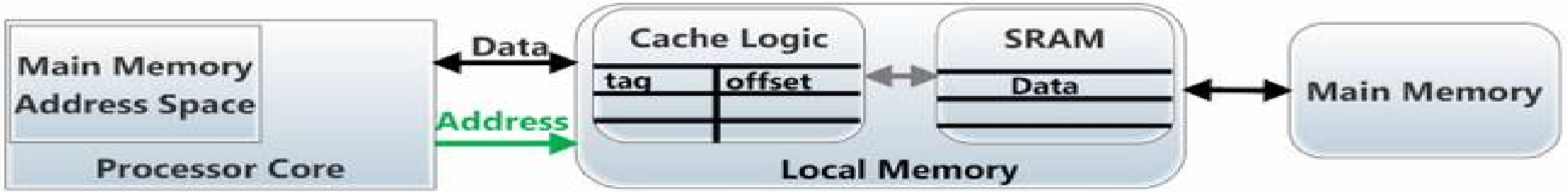
- Cache Read Operation - Flowchart



Cache Design

- Addressing
- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches

A Conventional Memory System Architecture



Cache

- Caches are present in most memory systems.
- The Cache dynamically stores a subset of the frequently used data. Thus, the timing of a load or store operation depends on the relationship between its effective address and the effective addresses of earlier operations.

- Conventional Cache used byte addressable memory.
- 2^b = size of a Cache
- 2^B = size of Main Memory
- Addr = Address of Main Memory
- CL = Data Transfer from/to the memory
- NCL = Number of Cache lines (Cls)
- CLS = Cache Line Size

CL, CS, Tag, Index, Offset

- **Determine Cache Details:**
- Number of cache lines NCL = 10
- Cache line size CLS = 1024 bytes
- **Calculate Total Cache Size:**
- Total cache size = NCL × CLS
- = 10240 bytes

- **Address Structure in Cache: When accessing data in a cache, the address is typically divided into three parts:**
- Tag: Identifies the unique block of memory that each cache line represents.
- Index: Specifies which cache line to check.
- Offset: Points to the exact byte within a cache line.

- Define Cache Addressing Parameters:
- Offset Size: Since each cache line is 1024 bytes, we need enough bits to address 1024 locations within each line.
- 1024=2 power 10, so the offset size is 10 bits.
- Index Size: With 10 cache lines, we need enough bits to uniquely identify each line.
- 10≈2410≈24, so the index size is approximately 4 bits.

- **Determine Tag Size: The tag size depends on the total address space. Let’s assume a 32-bit address space, which is common in many systems:**
- Total address bits = 32 bits
- Tag size = Total address bits - Index bits - Offset bits
- Tag size = 32−4−10=1832−4−10=18 bits

3. Read from address 1010000001100000

Direct-Mapped Cache

Line #	V	D	Tag	Data (32-byte block)
0:	0	0		
1:	1	1	1111	
2:	0	0		
3:	1	0	1010	
4:	0	0		
	...			
127:	0	0		

Result: hit, the line is valid, and the tag matches.

Direct-Mapped Cache

- Direct Mapped cache is an array of fixed size blocks.
- Each block holds consecutive bytes of main memory data.
- **Mapping Process:**
  - The index is used to select a specific cache line.
  - The tag in the cache line is compared to the tag of the memory address to confirm a match (cache hit).
  - The offset specifies the exact location of the requested data within the cache block.

Fully associative cache

- A fully associative cache.
- A fully associative cache permits data to be stored in any cache block, instead of forcing each memory address into one particular block. — When data is fetched from memory, it can be placed in any unused block of the cache.

Tag: The tag is derived from the most significant bits of the memory address. It uniquely identifies the memory block stored in the cache.

Offset: The offset specifies the exact byte within a cache block. Derived from the least significant bits of the memory address.

Index: Unlike in direct mapping, there is no index because any block can go into any cache line.

Mapping Process in Fully Associative Cache

- **Placement:**
  - A memory block can be placed in any cache line. There is no restriction based on an index.
- **Tag Matching:**
  - When accessing data, the tag of the memory address is compared to the tag stored in each cache line.
  - This requires checking all cache lines (a process called tag lookup).
- **Offset Usage:**
  - Once a tag match (cache hit) is found, the offset is used to locate the specific byte within the cache block.

Task

- You have 16 bit of address bus
- The maximum size of main memory 64K byte (byte addressable)
- Design a cache having NCL = 8 and CLS = 4K
  - Find out required memory for cache

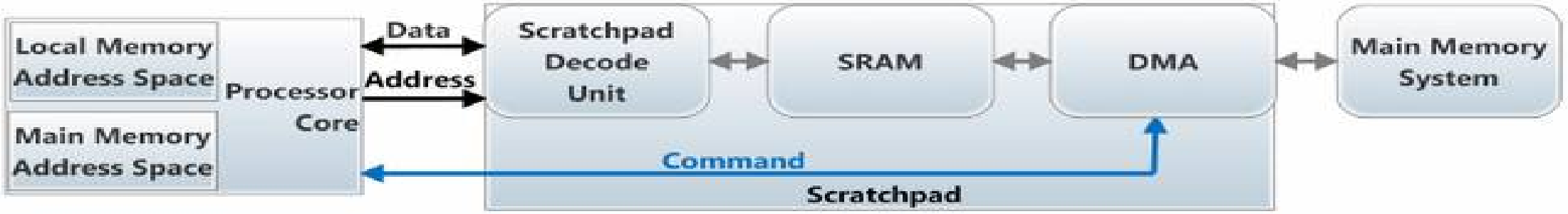
Set Associative Cache

- A set-associative scheme is a hybrid between a fully associative cache, and direct mapped cache.
- It's considered a reasonable compromise between the complex hardware needed for fully associative caches (which requires parallel searches of all slots), and the simplistic direct-mapped scheme, which may cause collisions of addresses to the same slot (similar to collisions in a hash table).

Mapping

- Index Selection:
  - The index is used to determine which set the memory block maps to.
- Tag Matching:
  - Within the selected set, the tag of the memory address is compared with the tag of each cache line in the set.
  - This process determines if the block is in the cache (cache hit).
- Offset Usage:
  - If the block is found in the set, the offset specifies the exact byte within the cache block.

Scratchpad

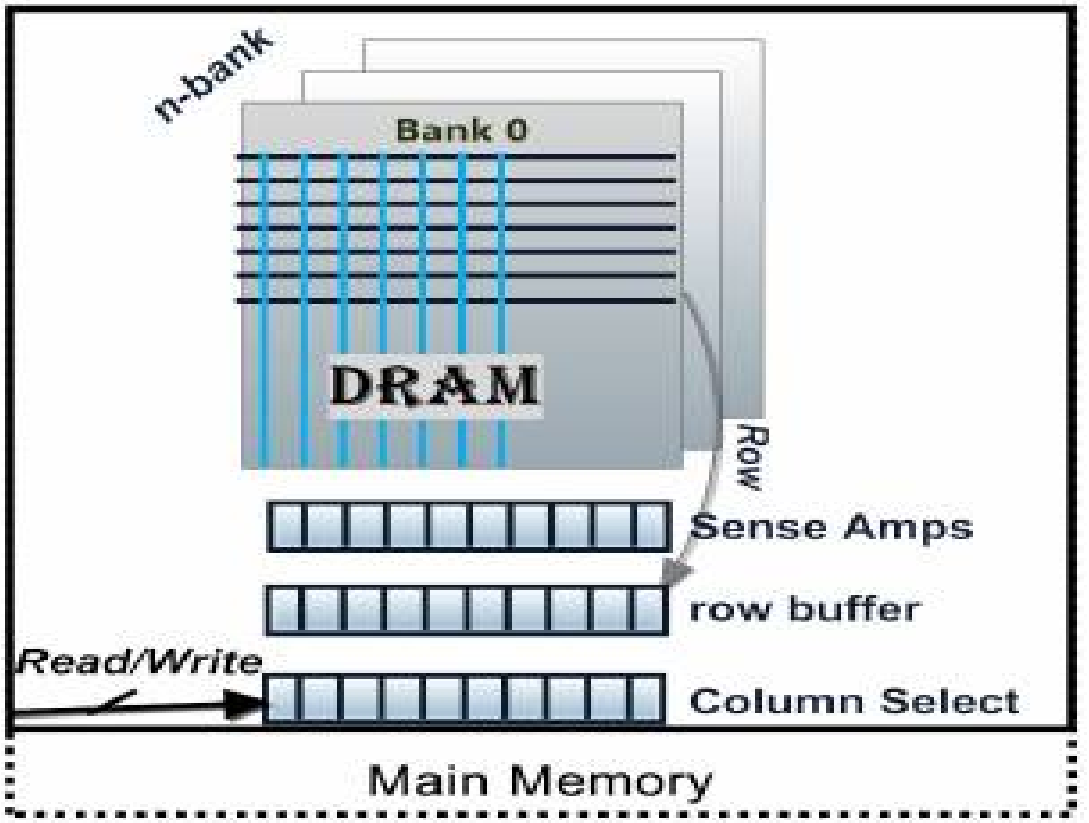

Local Memory Address Space
Main Memory Address Space
Processor Core
Data
Address
Scratchpad Decode Unit
SRAM
DMA
Main Memory System
Command
Scratchpad

Scratchpad

- The Scratchpad is a fast directly addressed software managed SRAM memory.
- The Scratchpad has better real-time guarantees than caches and by its significantly lower overheads it is better in access time, energy consumption and area.
- Instead of using traditional load/store instructions the scratchpad uses direct memory-memory operations using DMA.
- The Scratchpad memory access uses source and destination address registers, each of which holds a starting address of the memory.

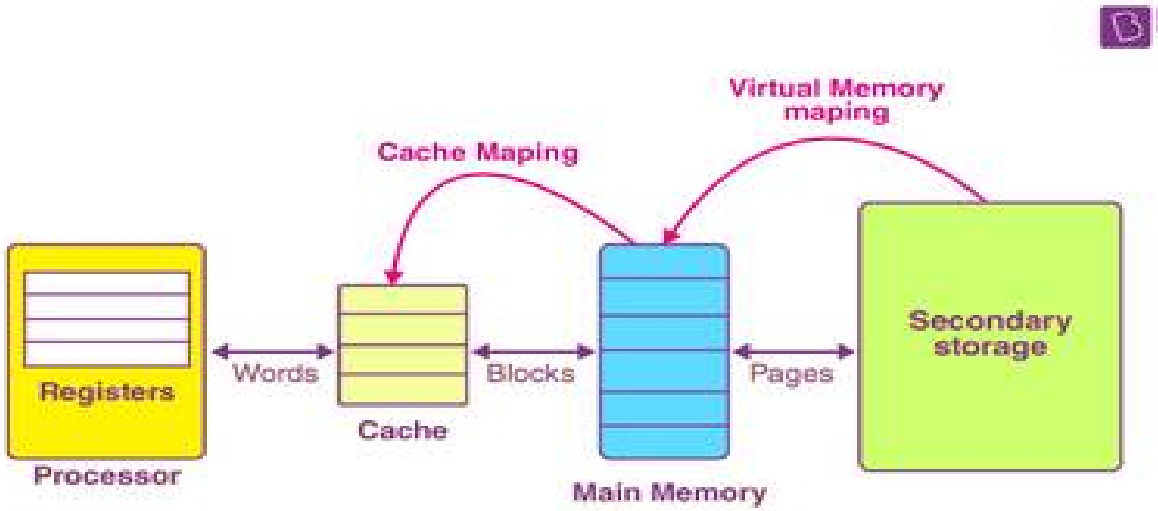
Main Memory System

- DRAM is combination Row x Column
- Row Address
- Column Address

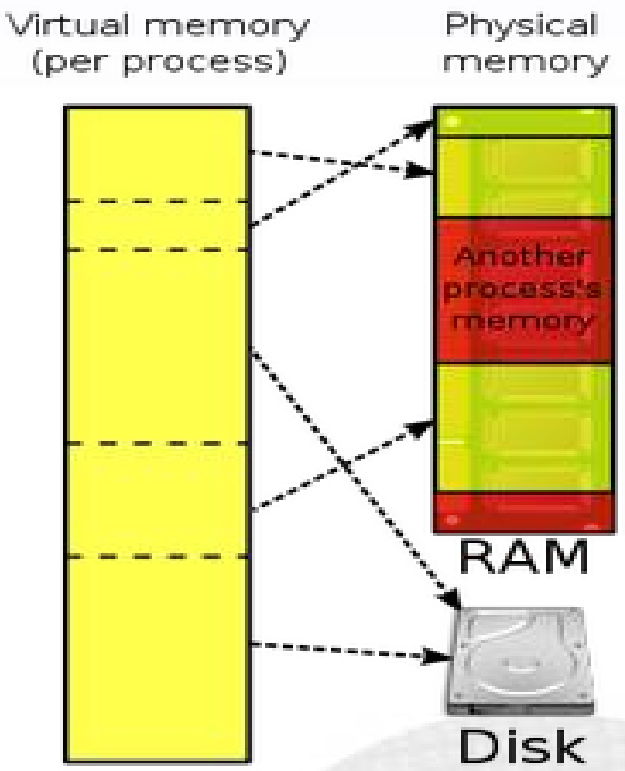


Virtual Memory

- Virtual memory is a memory management capability of an OS that uses hardware and software to allow a computer to compensate for physical memory shortages by temporarily transferring data from random access memory (RAM) to disk storage

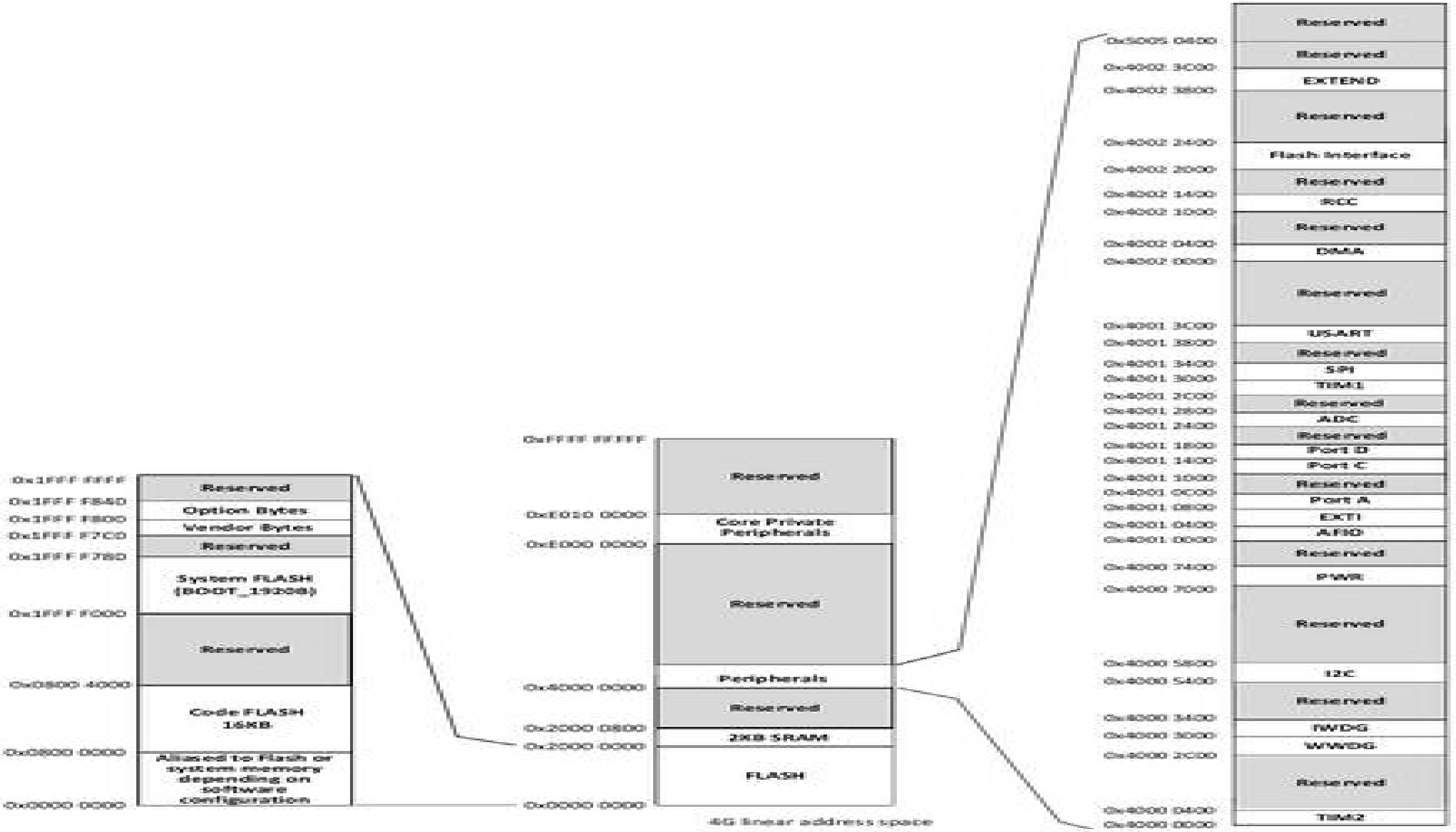


- The operating system, using a combination of hardware and software, maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.
- Main storage, as seen by a process or task, appears as a contiguous address space or collection of contiguous segments.



Memory Mapping

- Memory mapping is a crucial aspect of System on Chip (SoC) architecture. It refers to the way different components of the SoC are allocated addresses in the memory space. This mapping allows the CPU and other components to access and interact with various parts of the system's memory and peripherals.
- **Key Aspects of Memory Mapping in SoCs.**
  - Bus System
  - Address Space Allocation
  - Memory Regions
  - Memory Mapping Techniques:
  - Memory Map Tables
  - Access Mechanisms
  - Virtual Memory Mapping
  - Address Decoding:
- **Example of Memory Mapping in an SoC**
  - 0x0000_0000 - 0x1FFF_FFFF: RAM (1 GB of addressable RAM)
  - 0x2000_0000 - 0x3FFF_FFFF: ROM or Flash memory
  - 0x4000_0000 - 0x5FFF_FFFF: Peripheral registers (e.g., GPIO, UART)
  - 0x6000_0000 - 0x7FFF_FFFF: External memory or memory-mapped I/O space


0x1FFF FFFF
Reserved
0x1FFF F80F
Option Bytes
0x1FFF F800
Vendor Bytes
0x1FFF F7CD
Reserved
0x1FFF F7BD
System FLASH (BOOT_19208)
0x1FFF F000
Reserved
0x0800 4000
Code FLASH 16KB
0x0800 0000
Aliased to Flash or system memory depending on software configuration
0x0000 0000
0xFFFF FFFF
Reserved
0xE010 0000
Core Private Peripherals
0xE000 0000
Reserved
0x4000 0000
Peripherals
Reserved
0x2000 0800
2KB SRAM
0x2000 0000
FLASH
0x0000 0000
4G linear address space
0x5005 0400
Reserved
0x4002 3C00
Reserved
0x4002 3800
EXTEND
0x4002 3400
Reserved
0x4002 2000
Flash interface
0x4002 1400
Reserved
0x4002 1000
RCC
0x4002 0400
Reserved
0x4002 0000
DMA
Reserved
0x4001 3C00
USART
0x4001 3800
Reserved
0x4001 3400
SPI
0x4001 3000
TIM1
0x4001 2C00
Reserved
0x4001 2800
ADC
0x4001 2400
Reserved
0x4001 1800
Port D
0x4001 1400
Port C
0x4001 1000
Reserved
0x4001 0C00
Port A
0x4001 0800
EXTI
0x4001 0400
AFIO
0x4001 0000
Reserved
0x4000 7400
PWR
0x4000 7000
Reserved
0x4000 5800
I2C
0x4000 5400
Reserved
0x4000 3400
IWDG
0x4000 3000
WWDG
0x4000 2C00
Reserved
0x4000 0400
TIM2
0x4000 0000

Bus System Components

- Address Bus: Carries address information from the CPU to memory and peripherals. The address bus width determines the range of addresses that can be used in memory mapping.
- Data Bus: Transfers data between components based on the address specified on the address bus.
- Control Bus: Carries control signals that manage the read and write operations and other control functions.
- Functions:
  - Memory Map Configuration
  - Interconnects and Buses
  - Address Decoding
  - Memory-Mapped I/O

Example:

On-Chip Memory and Peripheral Mapping: Within the bus system, the memory map determines the layout of on-chip memory, peripheral registers, and I/O devices. The bus system ensures that the CPU and other components access the correct addresses based on this map.

Address Map/Space Allocation

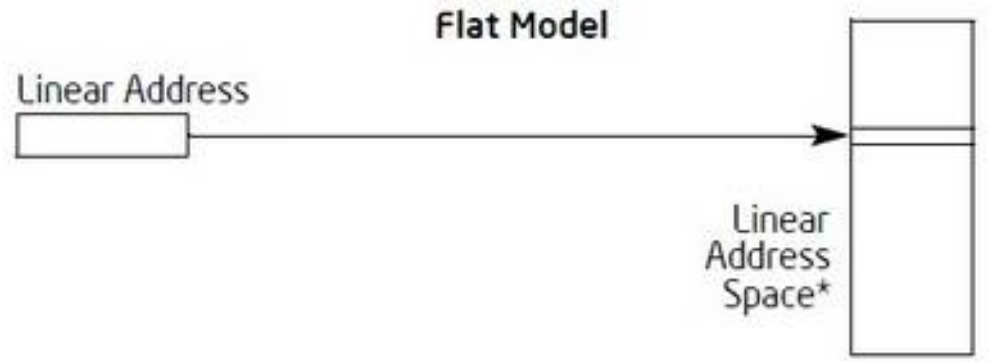
- Memory Address Space: Defines the range of addresses used to access different types of memory, including RAM, ROM, and external memory.
- Peripheral Address Space: Allocates addresses for various peripherals and I/O devices.

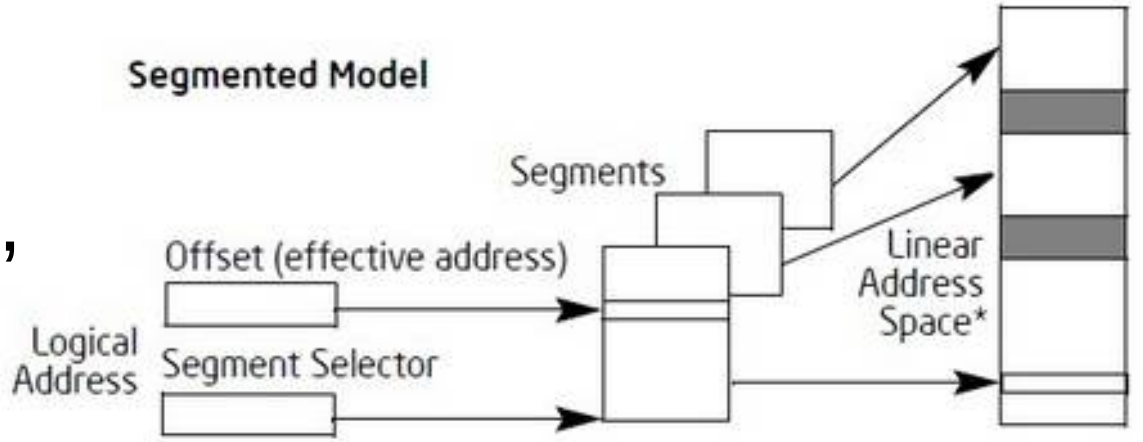
Memory Regions

- Boot Memory: Often used to store the bootloader or initial firmware.
- Code Memory: Stores executable code and program instructions.
- Data Memory: Used for storing variables, stack, and heap data.
- Peripheral Registers: Memory-mapped addresses used to control and interact with peripheral devices (e.g., timers, UARTs, GPIOs).

Memory Mapping Techniques:

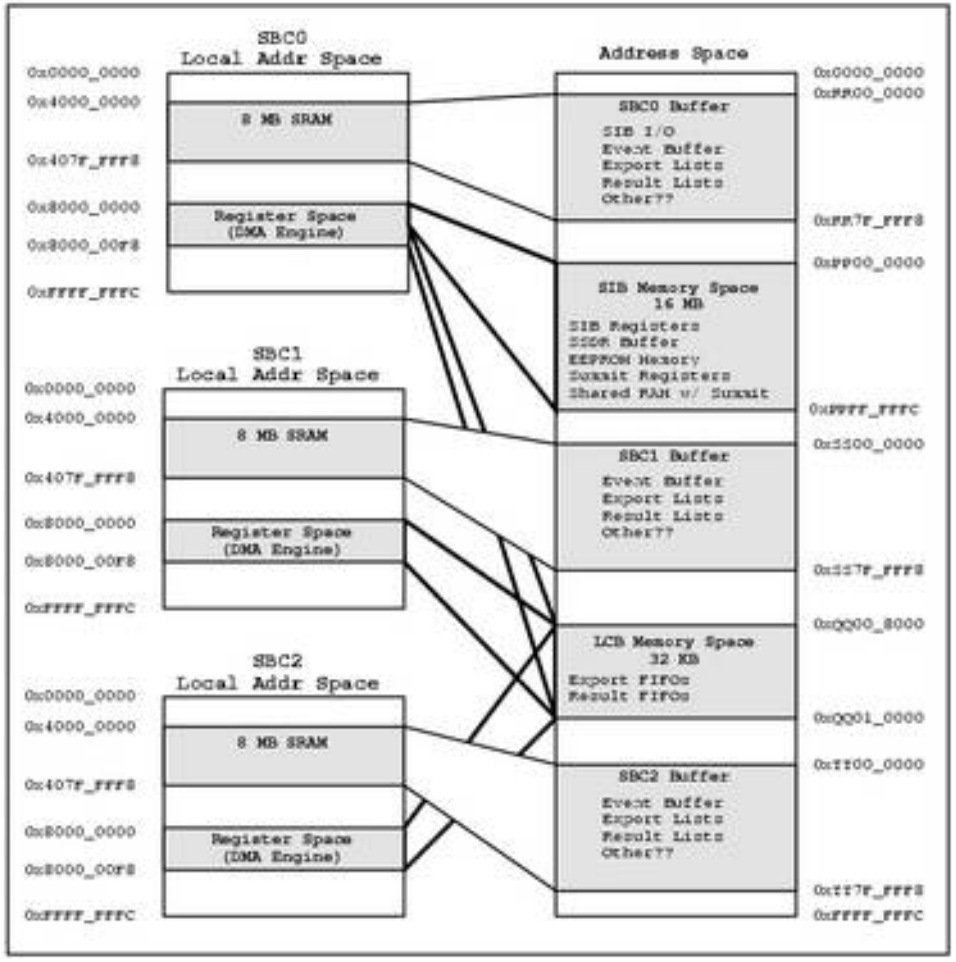
- Flat Memory Model: All memory and peripherals are mapped into a single, linear address space.
- Segmented Memory Model: Memory and peripherals are divided into segments or blocks, each with a specific address range.





Memory Map Tables and Access Mechanisms

- Memory Map Table: A detailed table that outlines the starting address, size, and type of each memory region and peripheral.
- Memory-Mapped I/O: Peripherals are accessed by reading from or writing to specific memory addresses.
- Access Mechanism:
  - Linker Script: Define how different code and data sections are placed in memory.
  - Direct Memory Access (DMA): Allows peripherals to directly access memory without CPU intervention, reducing latency and improving performance.

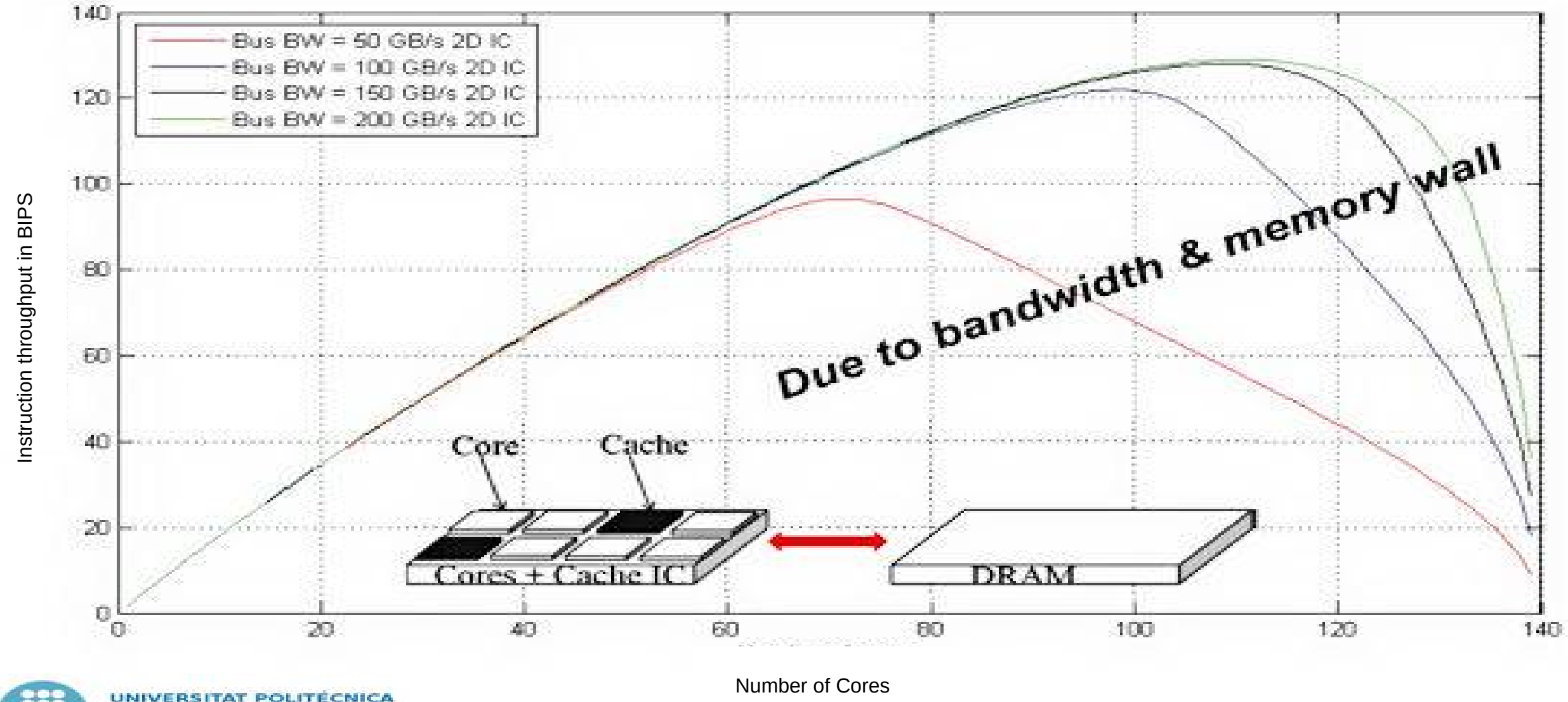


Virtual Memory Mapping

- Virtual Address Space: Some SoCs use virtual memory systems to abstract physical memory addresses, providing flexibility in memory management.
- Virtual Address: It is an address of a program's memory space.
- Page Table: The table contains mappings from virtual addresses to physical addresses. Each entry in the page table corresponds to a "page" of memory.
- Page Size: Memory is divided into fixed-size pages, typically ranging from 2 KB to 16 KB (though sizes like 4 KB or 8 KB are common). The virtual address is split into two parts:
  - Page Number: Identifies the page within the virtual address space.
  - Offset: Identifies the specific location within the page.
- Translation: When a virtual address is used, the page number is looked up in the page table to find the corresponding physical page. The offset is then added to this physical page to get the final physical address.
- Physical Address: The final physical address points to the exact location in the system's memory (RAM) where the data is stored.

Problem Statement

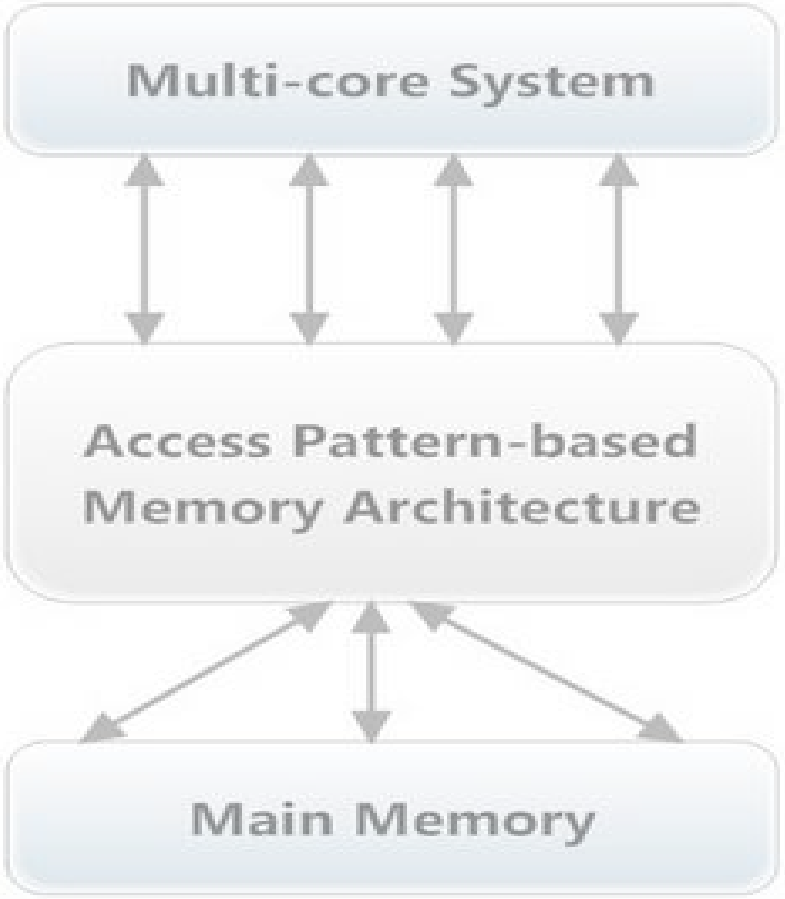
Bandwidth wall with increase in the number of cores [2]



[2] Muhammad Bakir; Georgia Tech

Focus to improve processor/memory performance

- Multi-core system
  - RISC
  - Vector & hardware accelerator cores
- Access Pattern-based Memory Architecture
  - Irregular/complex access patterns



Basic types of memory access patterns

- Regular access
  - Fixed stride
  - Predictable
  - Parallel
- Irregular access
  - Variable strides
  - Known
    - » Predictable at compile-time
  - Unknown
    - » Independent
    - » Dependent

Basic types of memory access units

- Load/store access
  - Conventional
  - Arbitrary access patterns
  - Fine granularity access
  - Low throughput

- for (inc=start_value; inc=end_value; inc++)
  - {
  - //Addresses Management by Microprocessor
  - local_buffer[inc]=mem_buffer[inc+offset];
  - }

- DMA
  - Streaming access
  - Programmed with function call
  - High latency
  - High throughput

Read_DMA(source,destination,stream);
Write_DMA(source,destination,stream);

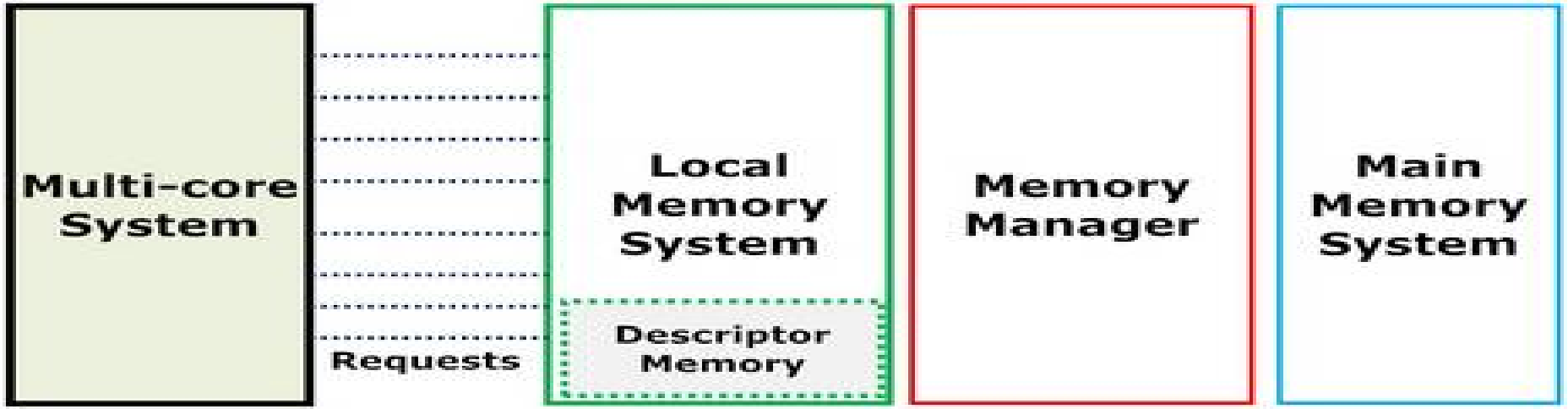
Proposal

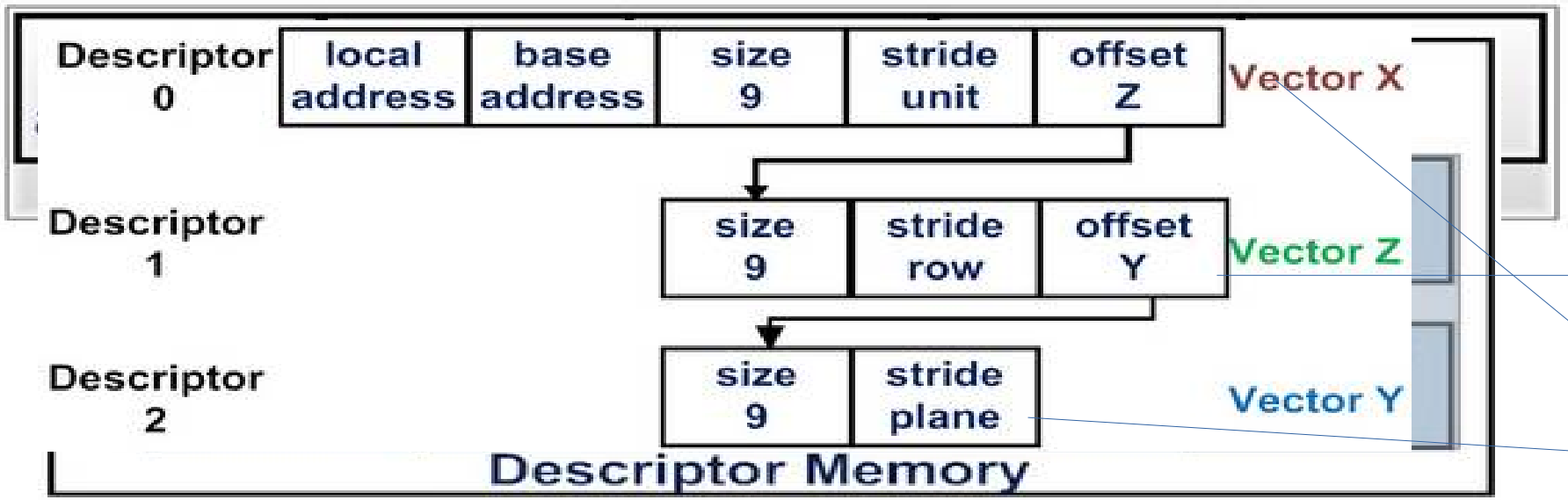
- Improve processor/memory performance gap by
  - Scheduling multiple cores
  - Managing/arranging memory access patterns
  - Efficiently utilizing hardware resources

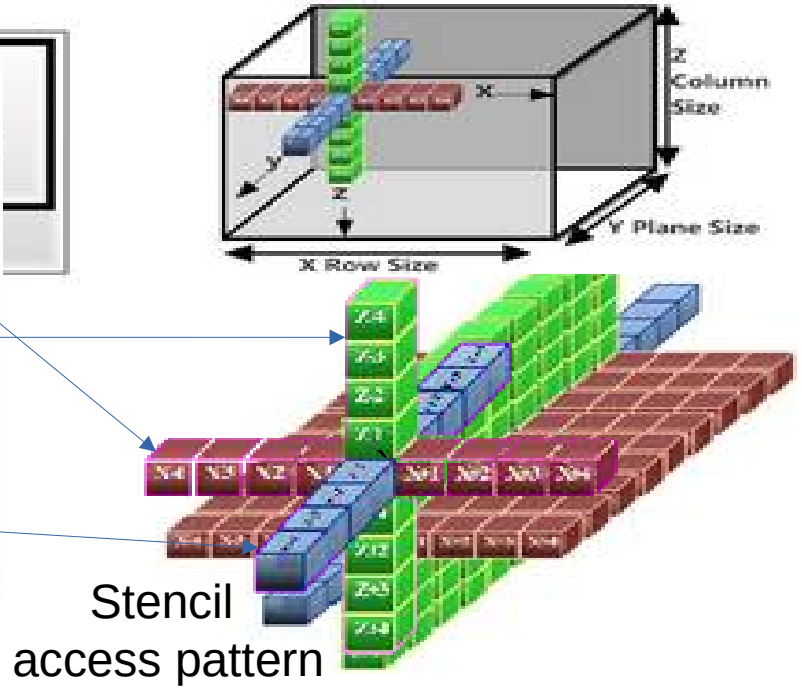
Objectives

- Operates independently (run time)
  - Manage/control data requests and access description
  - Perform memory read/write operations in parallel with processing unit
- Schedules requests
  - RISC (SSP), vector processor (VP) and application specific hardware accelerator (ASHA)
- Accesses complex / irregular patterns
  - Strided access, 1D/2D/3D tiles
  - Irregular complex patterns (e.g. linked list streams)
- Provides Programming support
  - Provide standard C/C++ language calls

Pattern based Memory Controller (PMC)







Stencil access pattern

PMC: Architecture

- Local Memory System
- Memory Manager
- Main Memory System
  - Regular pattern for single core
  - Irregular pattern for single core
  - Regular and irregular pattern for multi core