It's the Memory, Stupid!

Richard Sites, lead designer of the DEC Alpha, 1996

*I expect that over the coming decade memory subsystems design will be the **only** important design issue for microprocessors.*

Most of his colleagues designing next-generation Alpha architectures at Digital Equipment Corp. ignored his advice and instead remained focused on building ever faster microprocessors, rather than shifting their focus to the building of ever faster *systems* [8].

It's the Memory, Stupid!

Richard Sites, lead designer of the DEC Alpha, 1996

*I expect that over the coming decade memory subsystems design will be the **only** important design issue for microprocessors.*

Most of his colleagues designing next-generation Alpha architectures at Digital Equipment Corp. ignored his advice and instead remained focused on building ever faster microprocessors, rather than shifting their focus to the building of ever faster *systems* [8].

N.B.: Digital Equipment Corp. no longer exists

What memory is needed for?

What memory is needed for?

- storing data

What memory is needed for?

- storing data
- storing instructions

What memory is needed for?

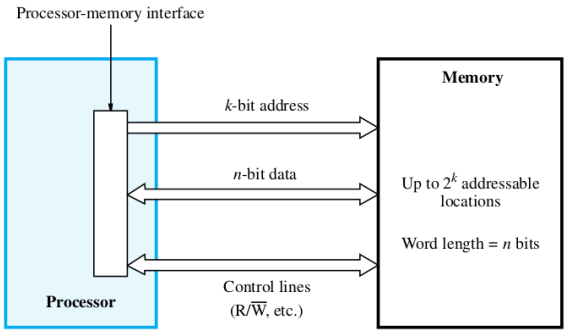
- storing data
- storing instructions
- saving temporary values

What memory is needed for?

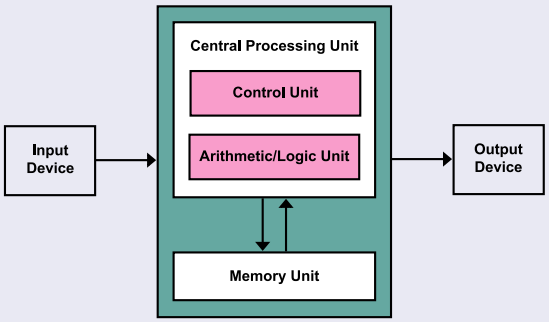
- storing data
- storing instructions
- saving temporary values
- synchronizing processes/threads

Connection of the memory to the processor

Von Neumann vs Harvard Architecture

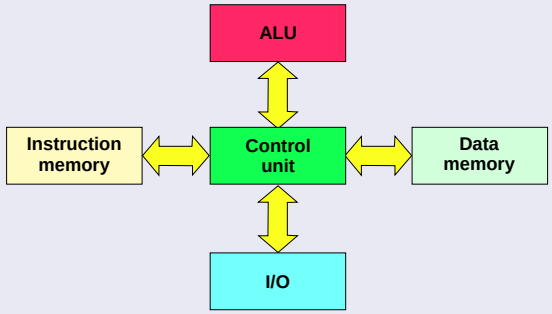


Von Neumann Architecture



- 1945
- common bus for instruction and data

Harvard Architecture



- 1944
- separate bus for instruction and data

The ideal memory is

- fast
- large
- inexpensive

The ideal memory is

- fast
- large
- inexpensive

Impossible to meet these three requirements

- physical properties of memories: area, delay, energy consumption
- economical issues
- $\nearrow$ speed + $\nearrow$ size = $\nearrow$ cost
- $\nearrow$ size $\Rightarrow$ $\searrow$ speed

The ideal memory is

- fast
- large
- inexpensive

Impossible to meet these three requirements

- physical properties of memories: area, delay, energy consumption
- economical issues
- $\nearrow$ speed + $\nearrow$ size = $\nearrow$ cost
- $\nearrow$ size $\Rightarrow$ $\searrow$ speed

Different solutions and structures exist

- different technologies
- different organisations
- for different needs (permanent store, operating store, and a fast store)

Memory hierarchy

The processor designer would choose

The user would choose

The manufacturer would choose

Memory hierarchy

The processor designer would choose

speed

The user would choose

The manufacturer would choose

Memory hierarchy

The processor designer would choose

speed

The user would choose

size

The manufacturer would choose

Memory hierarchy

The processor designer would choose

speed

The user would choose

size

The manufacturer would choose

cost

Memory hierarchy

The processor designer would choose

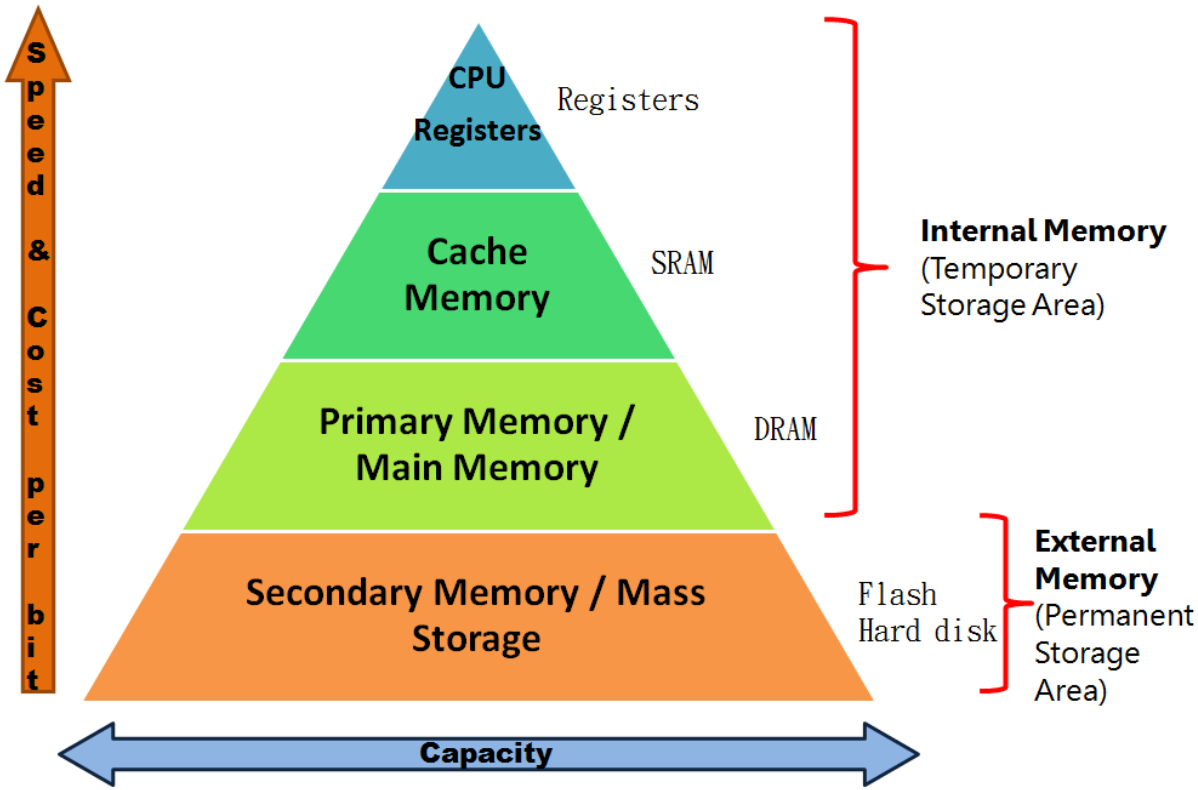
speed

The user would choose

size

The manufacturer would choose

cost



Source: not me

Memory hierarchy

The processor designer would choose

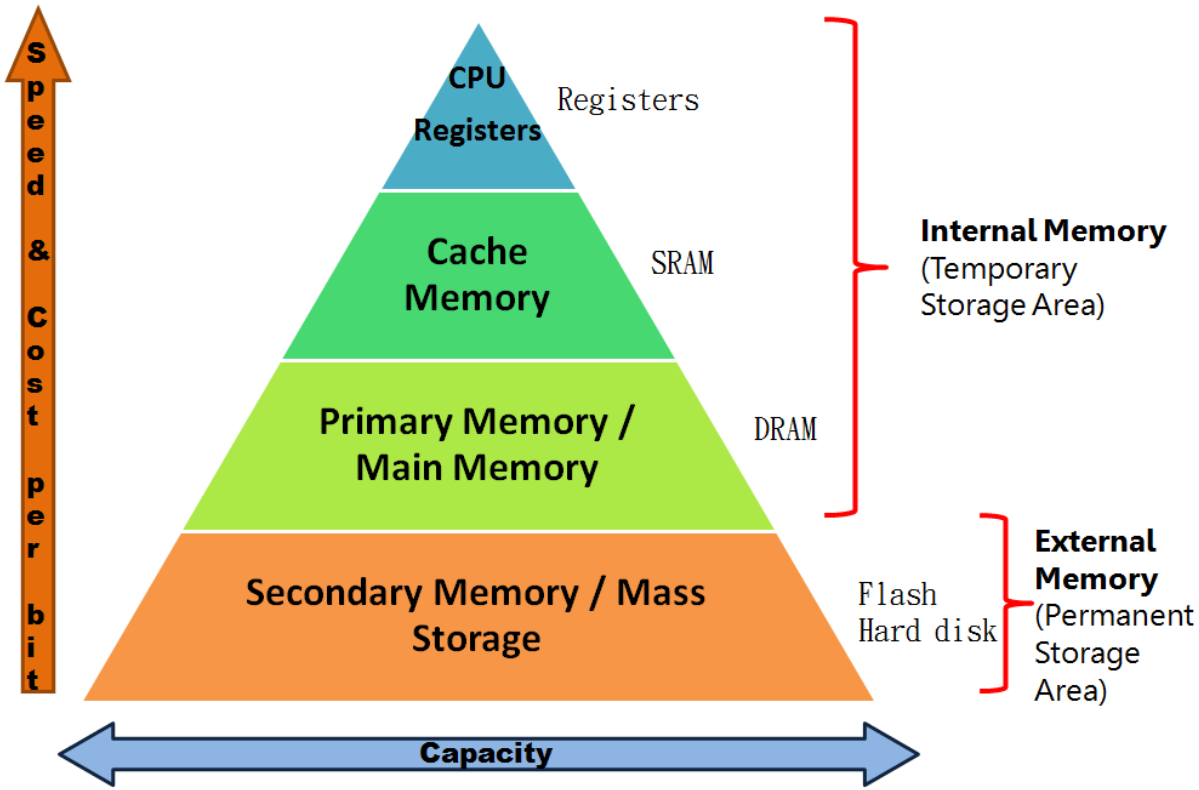
speed

The user would choose

size

The manufacturer would choose

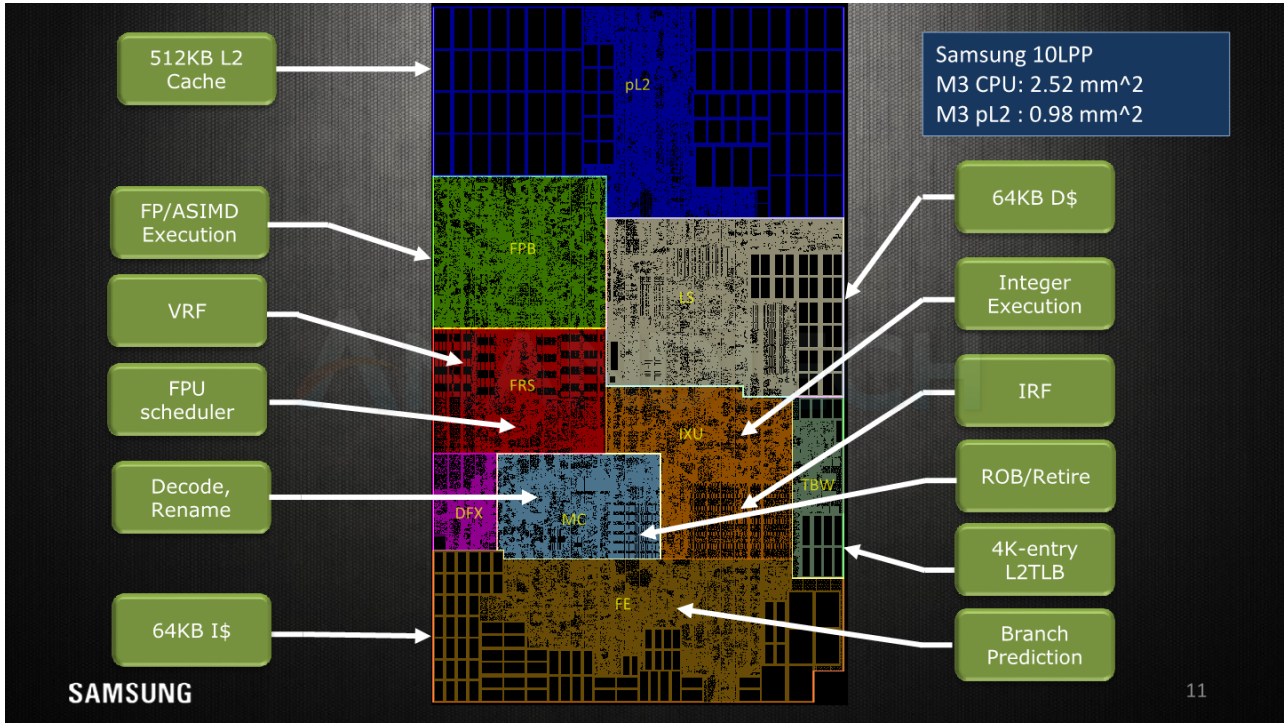
cost



Memory hierarchy as an enabler

Source: not me

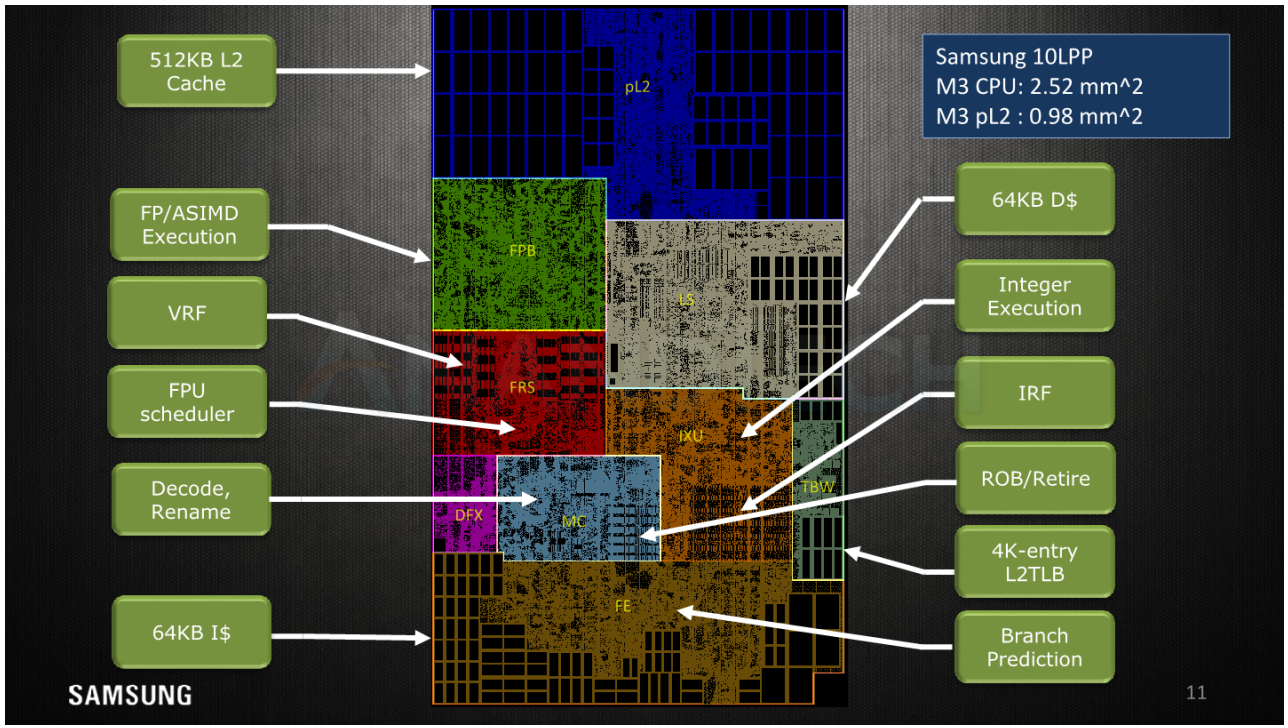
Exynos M3 Core Layout



Source: www.anandtech.com

- pL2: Private L2 cache, 512KB

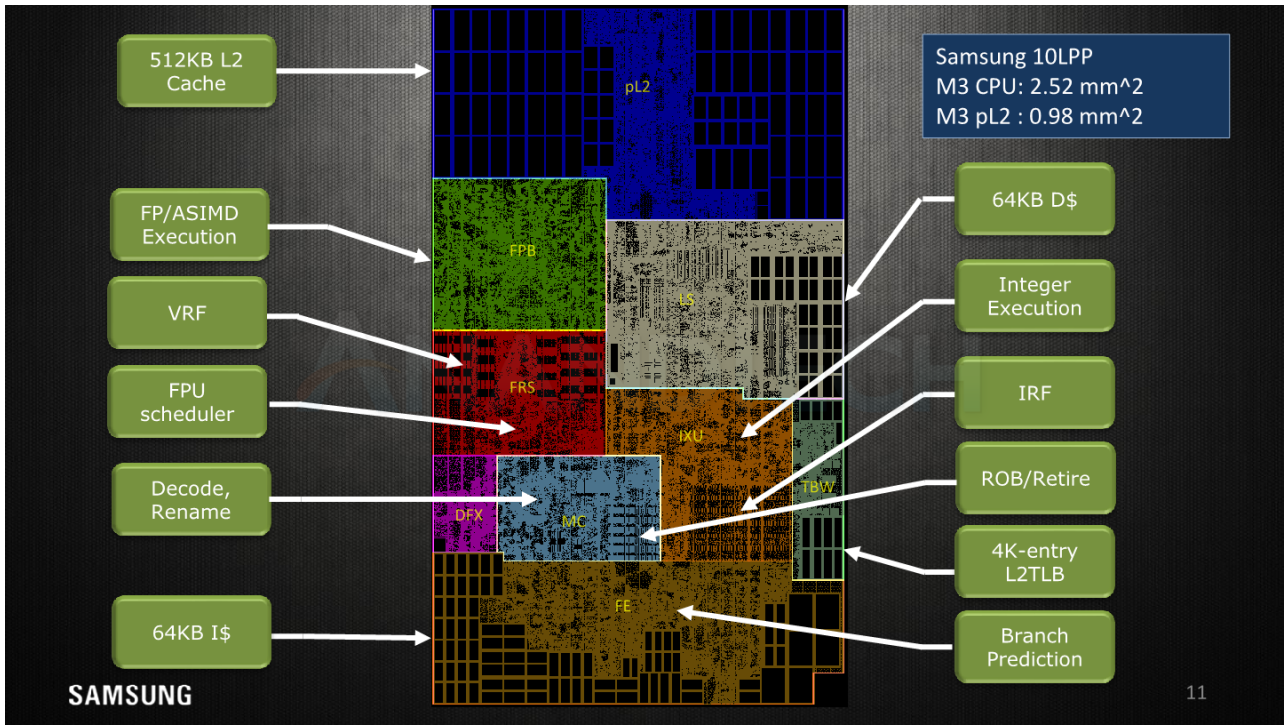
Exynos M3 Core Layout



Source: www.anandtech.com

- pL2: Private L2 cache, 512KB
- FPB: Floating point data path

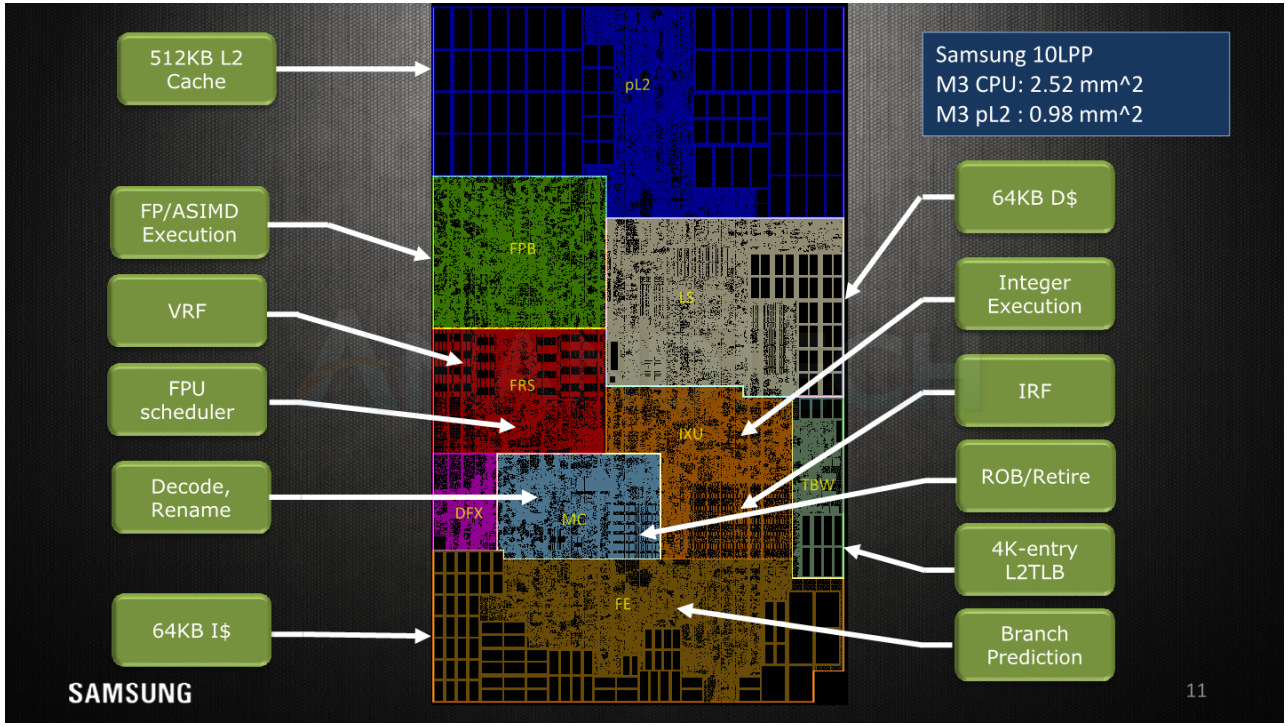
Exynos M3 Core Layout



Source: www.anandtech.com

- pL2: Private L2 cache, 512KB
- FPB: Floating point data path
- FRS: Floating point schedulers

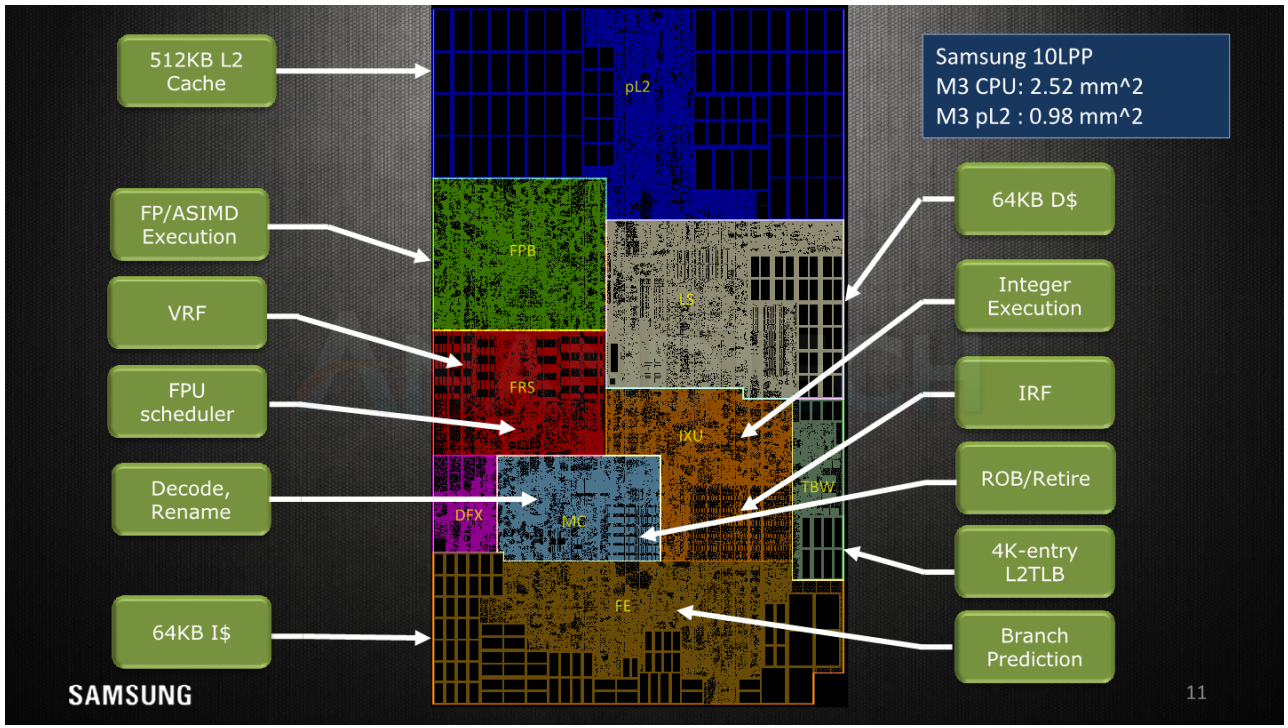
Exynos M3 Core Layout



Source: www.anandtech.com

- pL2: Private L2 cache, 512KB
- FPB: Floating point data path
- FRS: Floating point schedulers
- MC: Mid-core, the decoders and rename units.

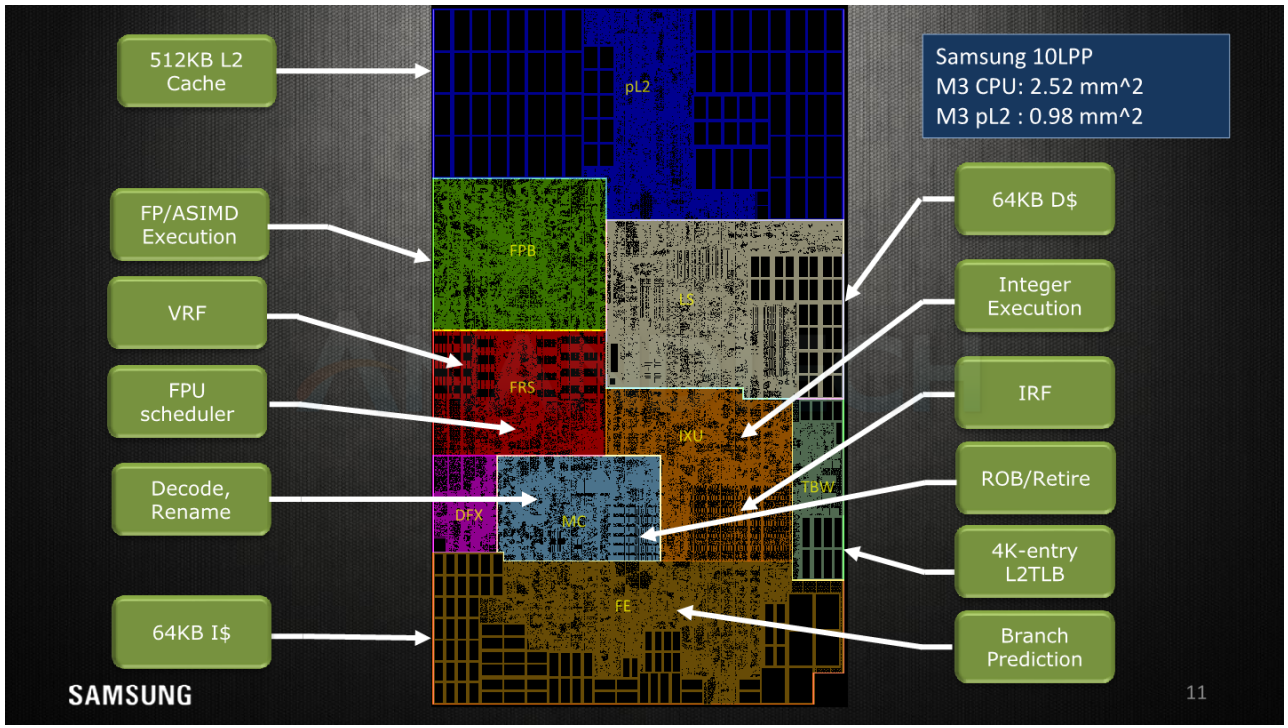
Exynos M3 Core Layout



Source: www.anandtech.com

- pL2: Private L2 cache, 512KB
- FPB: Floating point data path
- FRS: Floating point schedulers
- MC: Mid-core, the decoders and rename units.
- DFX: This is debug/test logic

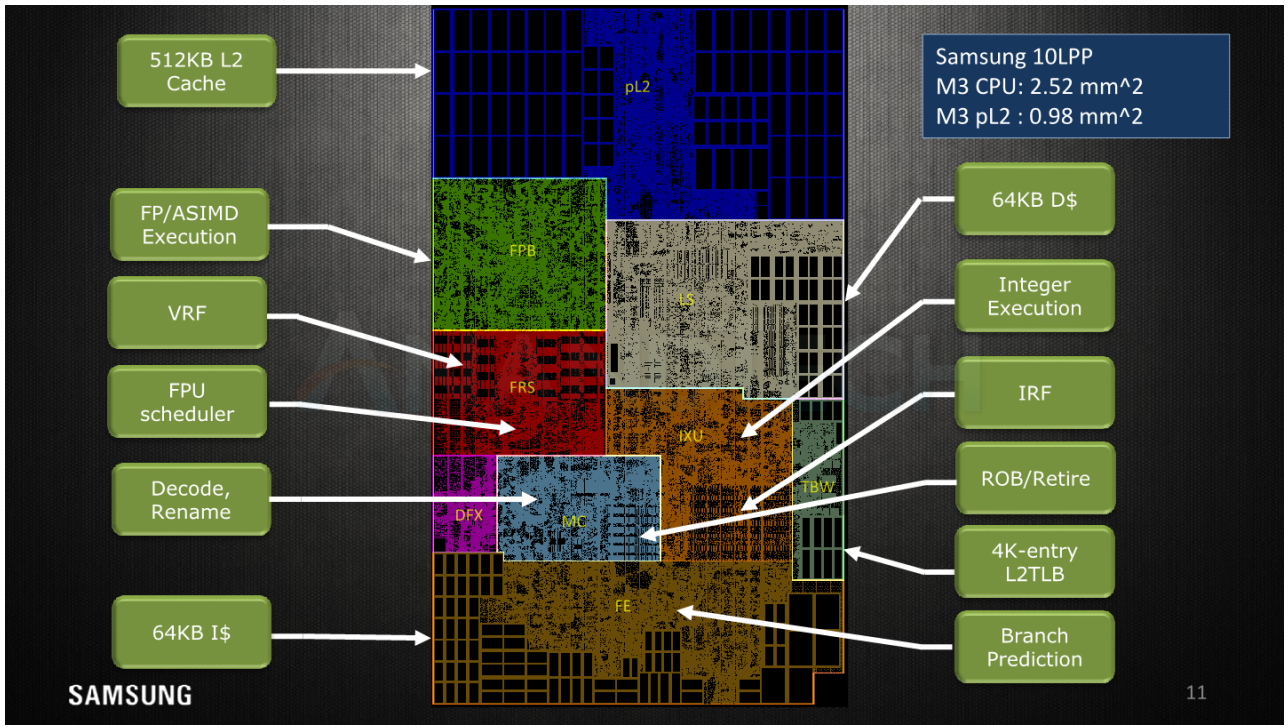
Exynos M3 Core Layout



Source: www.anandtech.com

- pL2: Private L2 cache, 512KB
- FPB: Floating point data path
- FRS: Floating point schedulers
- MC: Mid-core, the decoders and rename units.
- DFX: This is debug/test logic
- LS: Load/store unit along with the 64KB of L1 data cache memories.

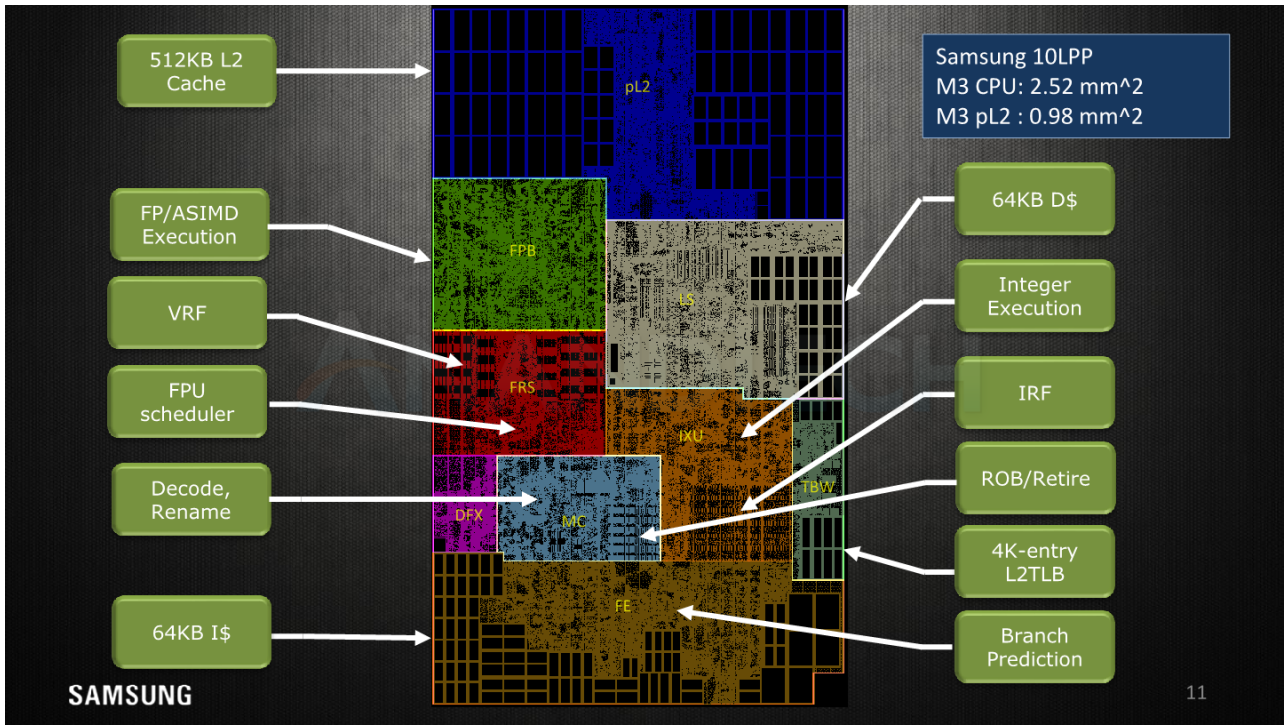
Exynos M3 Core Layout



Source: www.anandtech.com

- pL2: Private L2 cache, 512KB
- FPB: Floating point data path
- FRS: Floating point schedulers
- MC: Mid-core, the decoders and rename units.
- DFX: This is debug/test logic
- LS: Load/store unit along with the 64KB of L1 data cache memories.
- IXU: Integer execution unit; execution units, schedulers, integer physical register file memories.

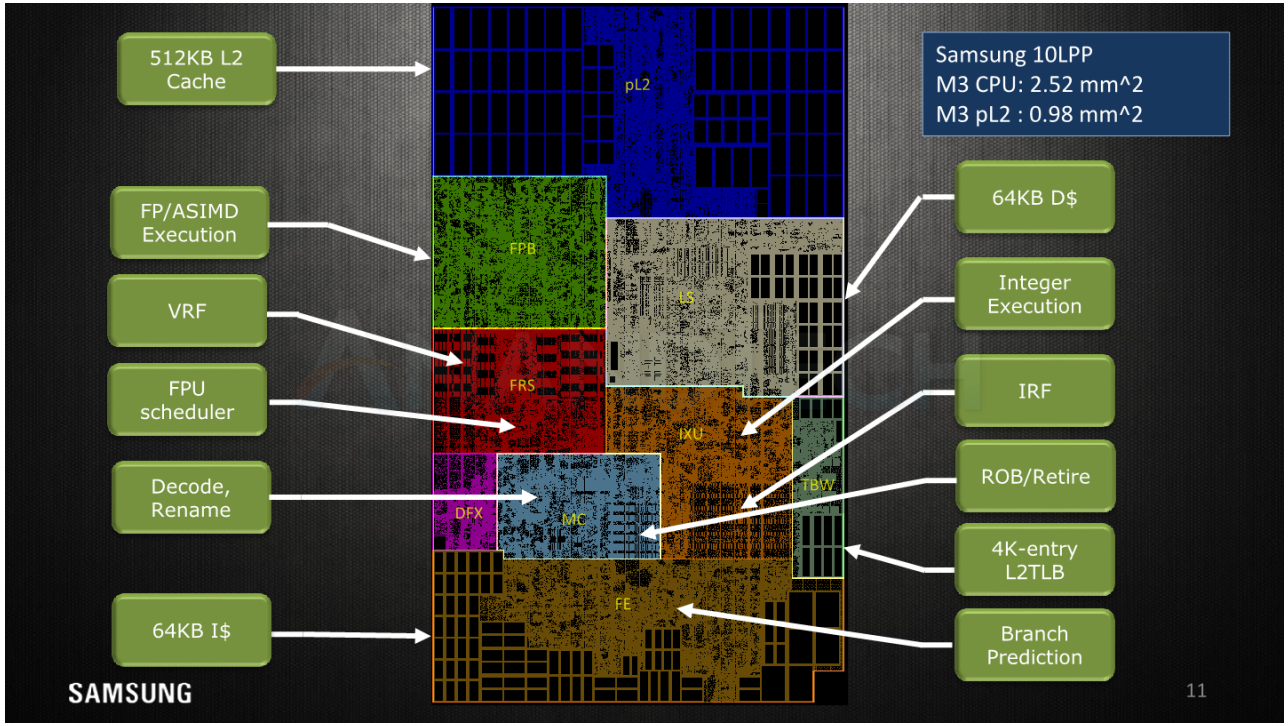
Exynos M3 Core Layout



Source: www.anandtech.com

- pL2: Private L2 cache, 512KB
- FPB: Floating point data path
- FRS: Floating point schedulers
- MC: Mid-core, the decoders and rename units.
- DFX: This is debug/test logic
- LS: Load/store unit along with the 64KB of L1 data cache memories.
- IXU: Integer execution unit; execution units, schedulers, integer physical register file memories.
- TBW: Transparent buffer writes, includes the TLB structures.

Exynos M3 Core Layout



Source: www.anandtech.com

- pL2: Private L2 cache, 512KB
- FPB: Floating point data path
- FRS: Floating point schedulers
- MC: Mid-core, the decoders and rename units.
- DFX: This is debug/test logic
- LS: Load/store unit along with the 64KB of L1 data cache memories.
- IXU: Integer execution unit; execution units, schedulers, integer physical register file memories.
- TBW: Transparent buffer writes, includes the TLB structures.
- FE: The front-end including branch predictors, fetch units and the 64KB L1 instruction cache memories.

Insight on current challenges

- more than 80% of the chip area is dedicated to caches, memories, memory controllers, interconnects and so on, whose sole purpose is to buffer data or control the buffering of data [1]
- $\Rightarrow$ workarounds which are making systems ever more complex
- more than 62% of the entire measured system energy is spent on moving data between memory and the computation units [1]

Content

This lecture will cover

- The basics (Memory elements, memory cells)
- Overview on SRAM and DRAM
- Memory system
  - Caches
  - Virtual memory
  - Virtual Machine
- The future
  - Technological improvements
  - Disruptive schemes

This lecture will NOT cover

- Massive storage (Hard Disk drives, magnetic or optical drives, etc.)
- Exhaustive DRAM features (timing, controller, protocol, system)
- I/O Topics

Part I

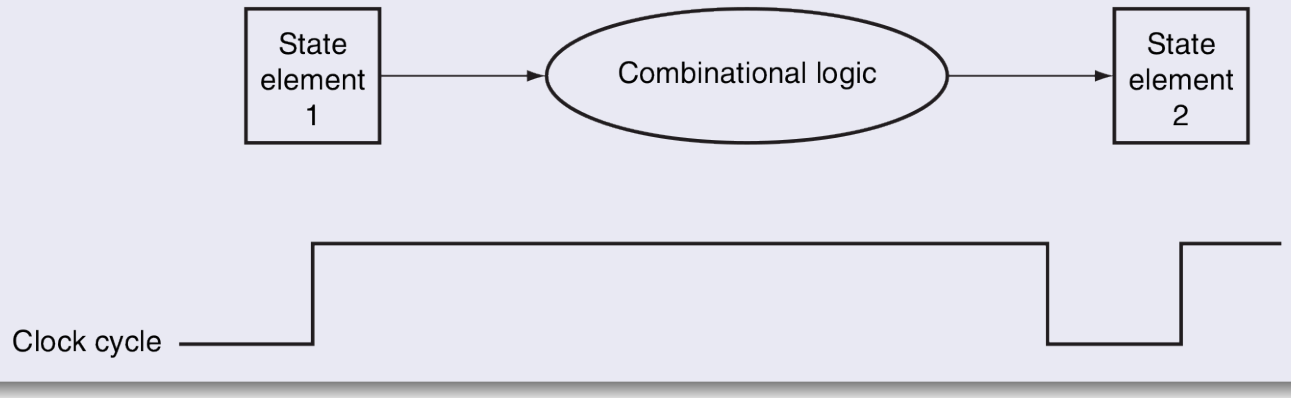
Basics

Outline of Part I

Outline

Clocks and sequential logic

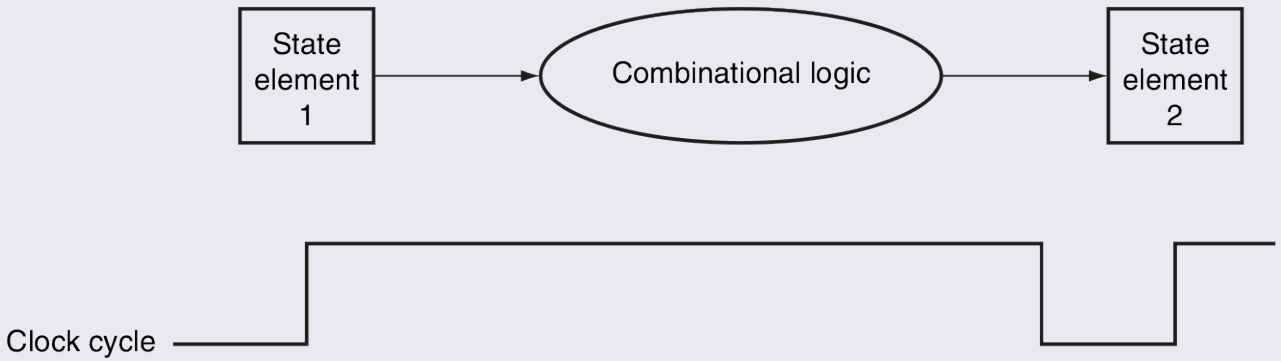
Clocking is used to update state elements



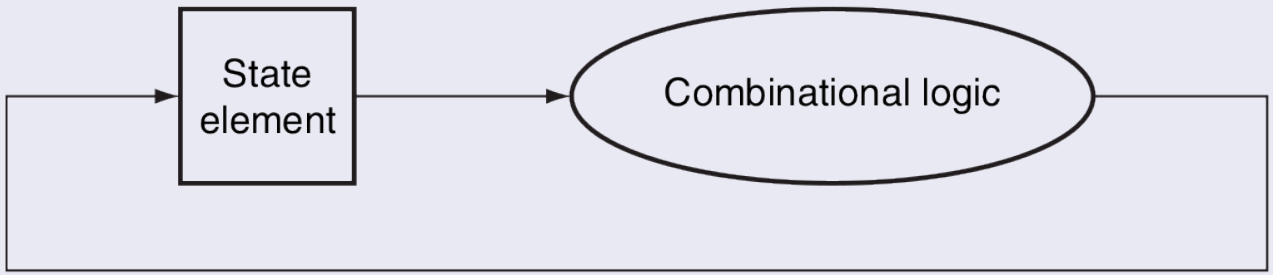
Source: [10]

Clocks and sequential logic

Clocking is used to update state elements



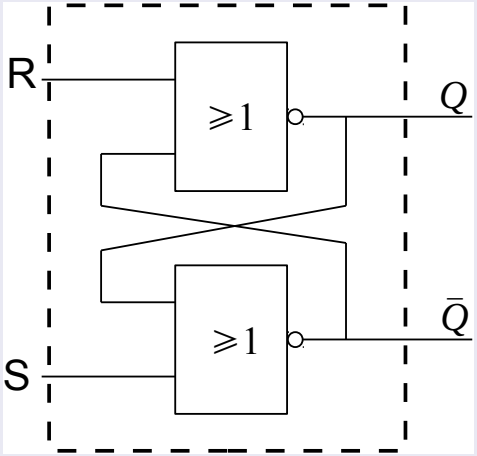
Edge-triggered methodology: a state element to be read and written in the same clock cycle



Source: [10]

S-R latch (set-reset latch)

Implementation with NOR gates



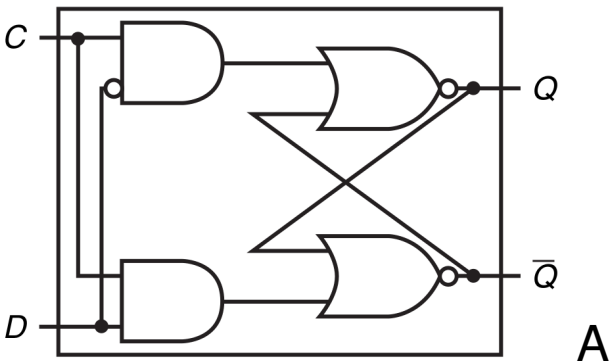
Truth table

S	R	Q_{t+1}	$\overline{Q_{t+1}}$	
0	0	Q_t	$\overline{Q_t}$	unchanged $\Rightarrow$ memory
0	1	**0**	**1**	**reset to 0**
1	0	**1**	**0**	**set to 1**
1	1	**0**	**0**	**Forbidden state**

Flip-Flops and Latches

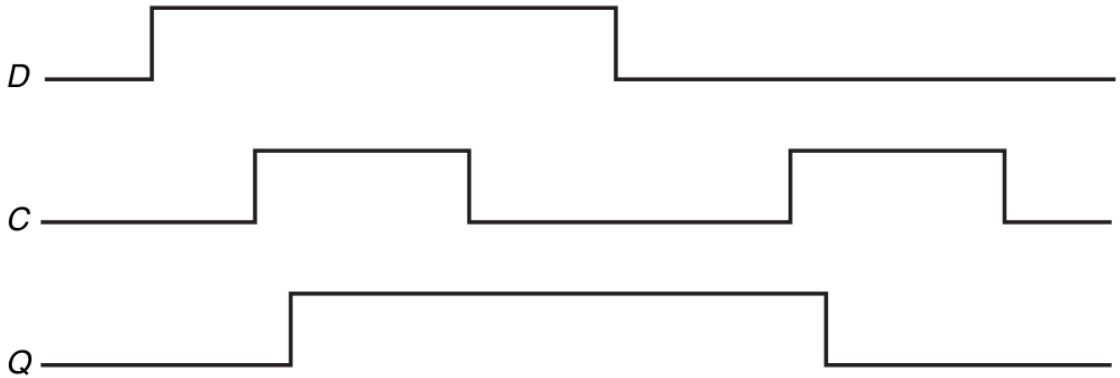
Definition

- latch: A memory element in which the output is equal to the value of the stored state inside the element and the state is changed whenever the appropriate inputs change and the **clock is asserted**
- D latch: A latch with **one data input** (called *D*) that stores the value of that input signal in the internal memory



A D latch implemented with NOR gates

Source: [10]

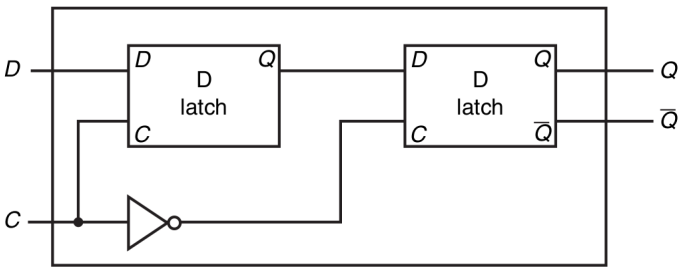


Operation of a D latch, assuming the output is initially deasserted

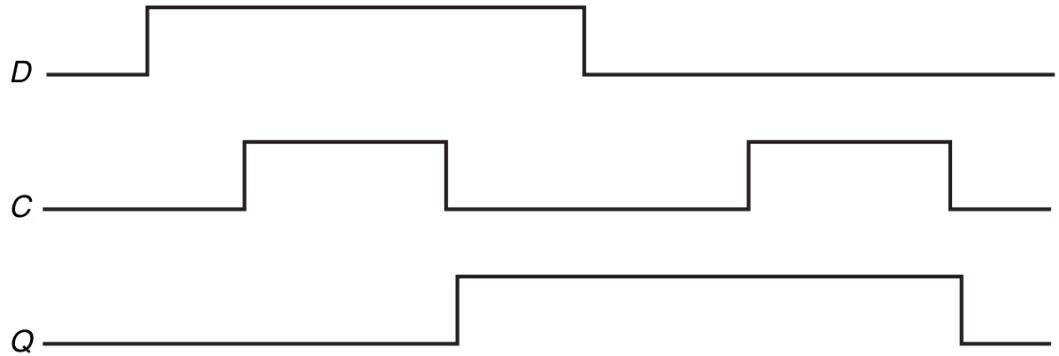
D flip-flop

Definition

- flip-flop: A memory element for which the output is equal to the value of the stored state inside the element and for which the internal state is changed **only on a clock edge**
- D flip-flop: A flip-flop with **one data input** (called *D*) that stores the value of that input signal in the internal memory when the **clock edge occurs**



A D flip-flop with a falling-edge trigger



Operation of a D flip-flop with a falling-edge trigger, assuming the output is initially deasserted

Source: [10]

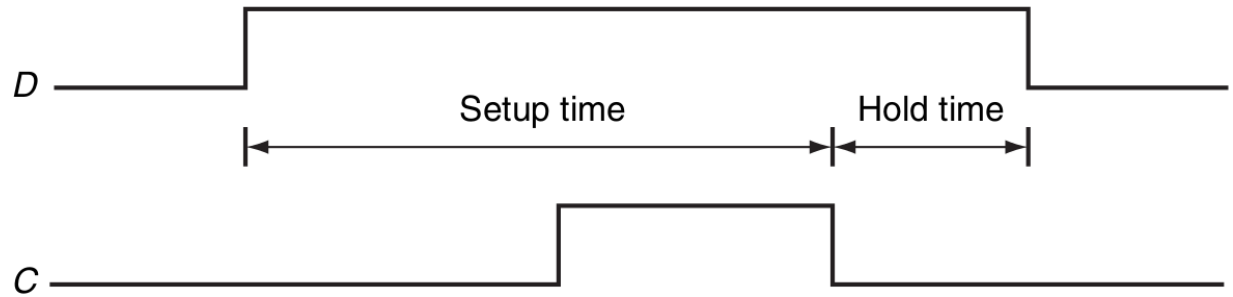
D flip-flop

Timing

The D input is sampled on the clock edge, it must be valid for a period of time immediately before and immediately after the clock edge

Definition

- setup time: The minimum time that the input to a memory device must be valid before the clock edge
- hold time: The minimum time during which the input must be valid after the clock edge



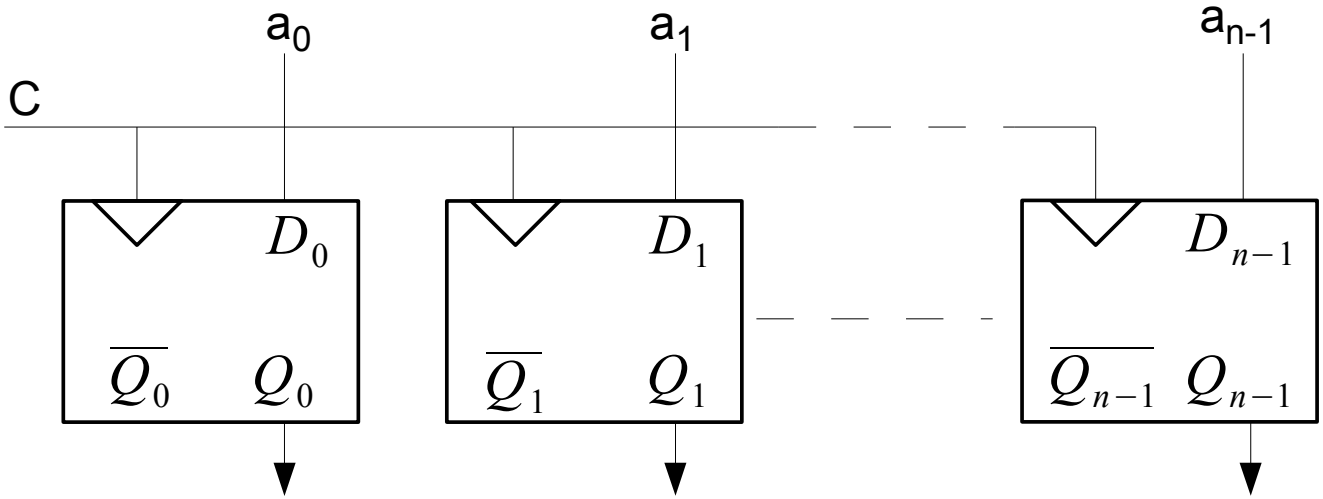
Setup and hold time requirements for a D flip-flop with a falling-edge trigger

Source: [10]

Registers

Definition

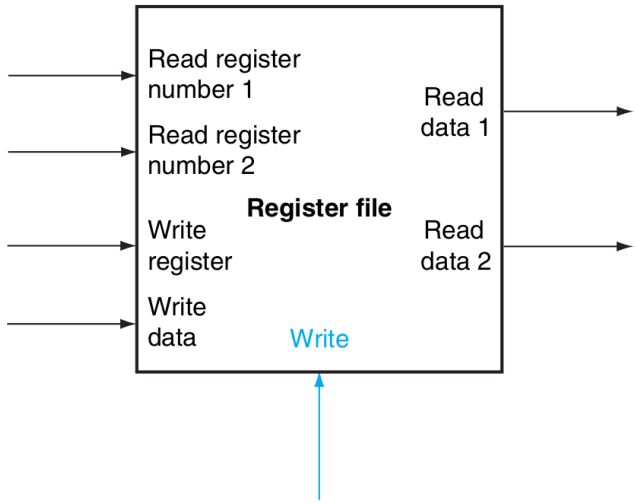
- Register: an array of D flip-flops that can hold a multibit datum, such as a byte or word



Registers and register files

Register File

- Set of registers
- Specify the register number to be accessed
- One decoder per read or write port
- **Central structure of the datapath of a processor**



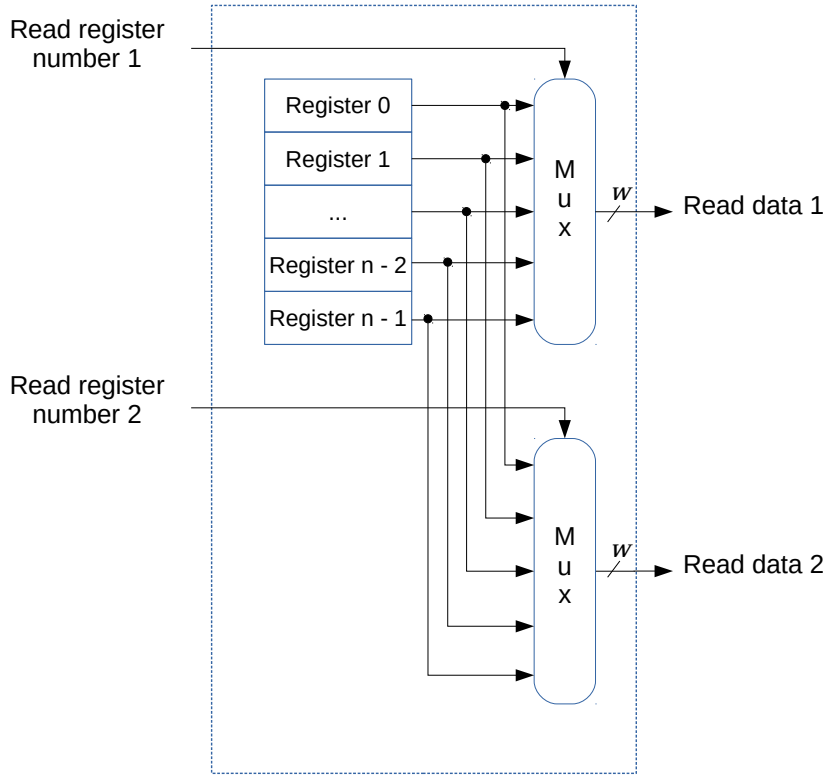
A register file with two read ports and one write port has five inputs and two outputs

Registers and register files

Reading a value

Read operation

- Input: register number
- Output: data contained in that register



Implementation of two read ports for a register file with n registers with a pair of n-to-1 multiplexors, each *w* bits wide

Registers and register files

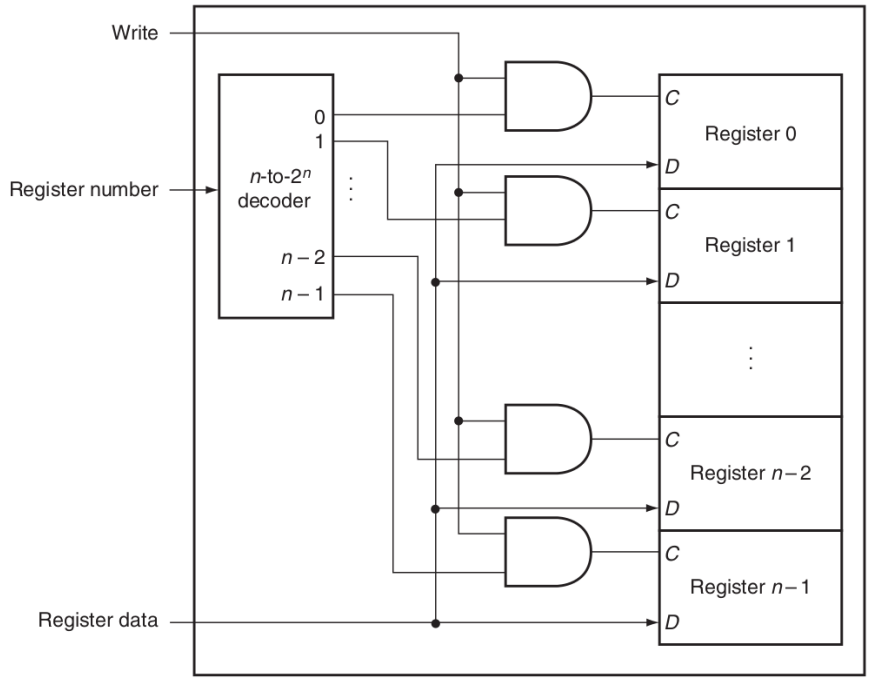
Writing a value

Write operation

- 3 inputs:
  1. register number
  2. data value
  3. clock (write signal)

Timing constraints

Setup and hold-time constraints to ensure that the correct data is written into the register file



The write port for a register file is implemented with a decoder that is used with the write signal to generate the *C* input to the registers

Registers and register files

Register file parameters

- Size (number of registers)
- Number of ports
- Width? (usually set by data width)

Registers and register files

Register file parameters

- Size (number of registers)
- Number of ports
- Width? (usually set by data width)

Size

- Too small: register *spilling*
- Too large: static energy, extra chip area

Registers and register files

Register file parameters

- Size (number of registers)
- Number of ports
- Width? (usually set by data width)

Size

- Too small: register *spilling*
- Too large: static energy, extra chip area

Number of ports

- nonlinear cost function of the number of ports
  - (partitioned register files for some VLIW processors)
- 2 read ports + 1 write port: good trade-off

Registers and register files

Register file parameters

- Size (number of registers)
- Number of ports
- Width? (usually set by data width)

Size

- Too small: register *spilling*
- Too large: static energy, extra chip area

The tricky thing

Reading the value currently being written (in the same clock cycle)

Number of ports

- nonlinear cost function of the number of ports
  - (partitioned register files for some VLIW processors)
- 2 read ports + 1 write port: good trade-off

Registers and register files

Unsuited for big memories

Bigger memories

Small memories are built using registers and register files:

- configuration registers
- pipeline registers
- processor register file (32x32 = 128 B)

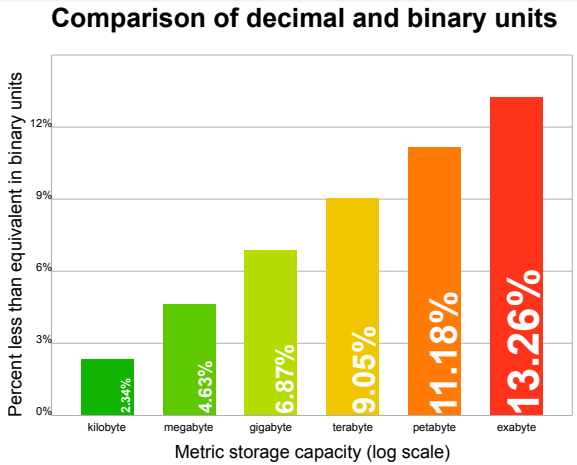
Bigger memories are built upon another organisation

RAM

Measuring memory size

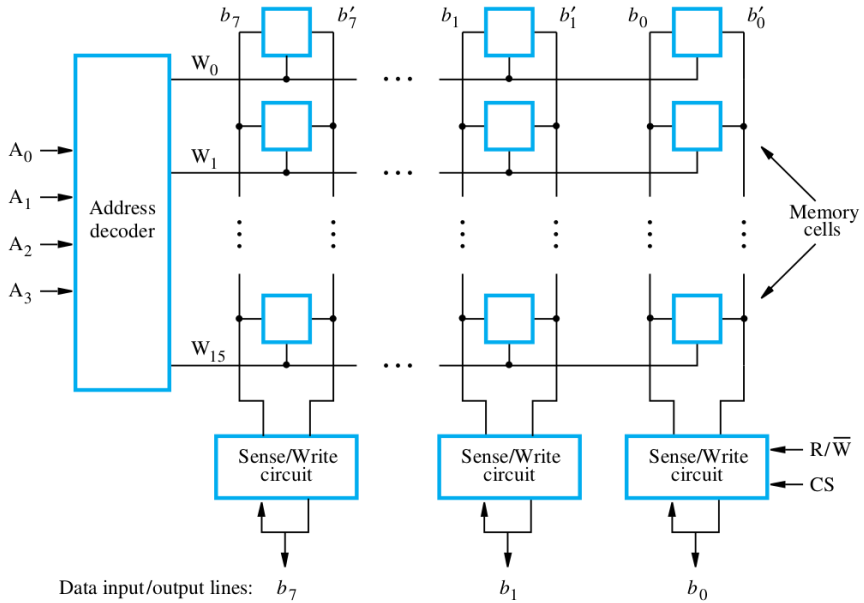
Unit multiples of the octet (byte) may be formed with SI prefixes and binary prefixes (power of 2 prefixes) as standardized in 1998 [2]

- 1 Byte = 8 bits
- 1 kilobyte (kB) = 10^3 bytes = 1 000 bytes
- 1 kibibyte (KiB)= 2^{10} bytes = 1 024 bytes
- ...



By Ryoushi19 - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=6808262

Internal Organisation of Memory Chips

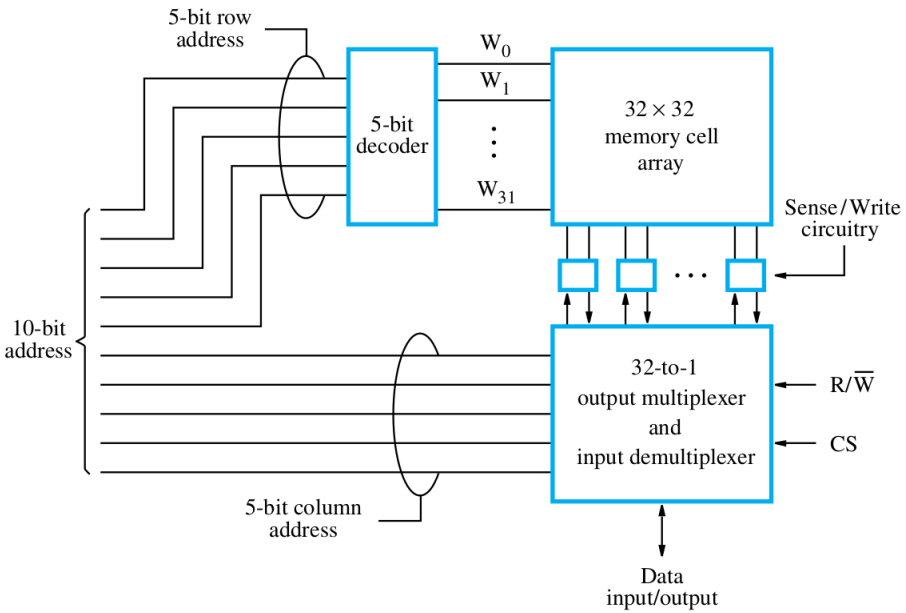


Memory cells organized in the form of an array

Definition

- height: the number of addressable locations
- width: the number of bits per unit

Internal Organisation of Memory Chips



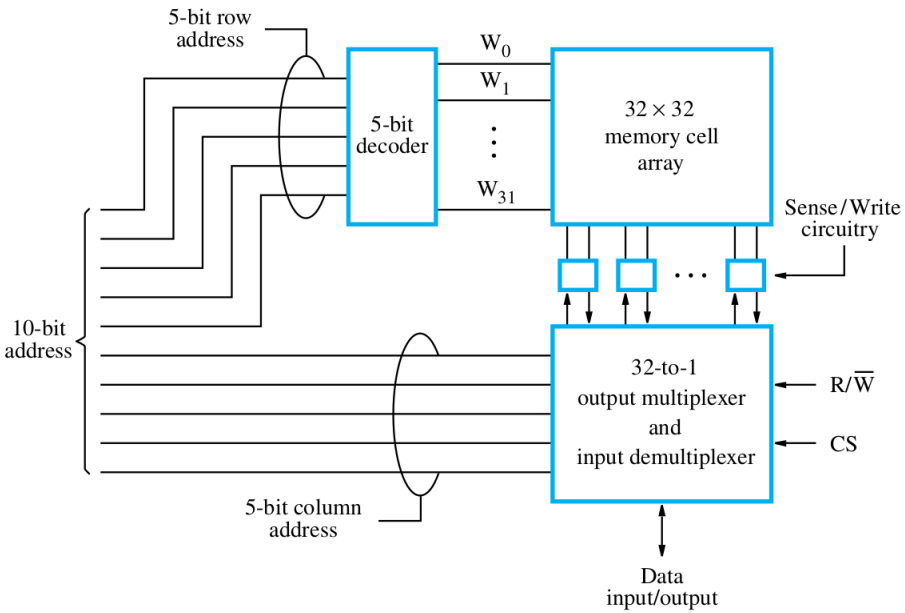
Organization of a 1K x 1 memory chip

Impact on the number of wires

Example with storing 1024 bits

	1K x 1	128 x 8	32 x 32
CS	1	1	1
$R/\overline{W}$	1	1	1
VCC	1	1	1
GND	1	1	1
Data	1	8	32
Address	10	7	5
Σ	15	19	41

Internal Organisation of Memory Chips



Organization of a 1K x 1 memory chip

Impact on the number of wires

Example with storing 1024 bits

	1K x 1	128 x 8	32 x 32
CS	1	1	1
$R/\overline{W}$	1	1	1
VCC	1	1	1
GND	1	1	1
Data	1	8	32
Address	10	7	5
Σ	15	19	41

Narrow configurations

- Fastest and newest memories use narrow configurations (x1 or x4)

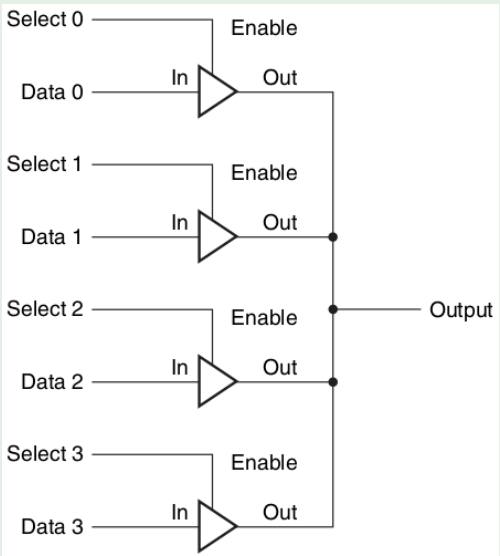
Internal Organisation of Memory Chips

The output multiplexor

Large memory

A 64K-to-1 multiplexor that would be needed for a 64K x 1 memory is totally impractical!

Tri-State Buffers



Four three-state buffers are used to form a multiplexor

Two inputs:

- data signal
- Output enable

One output with three states:

- asserted
- deasserted
- high impedance

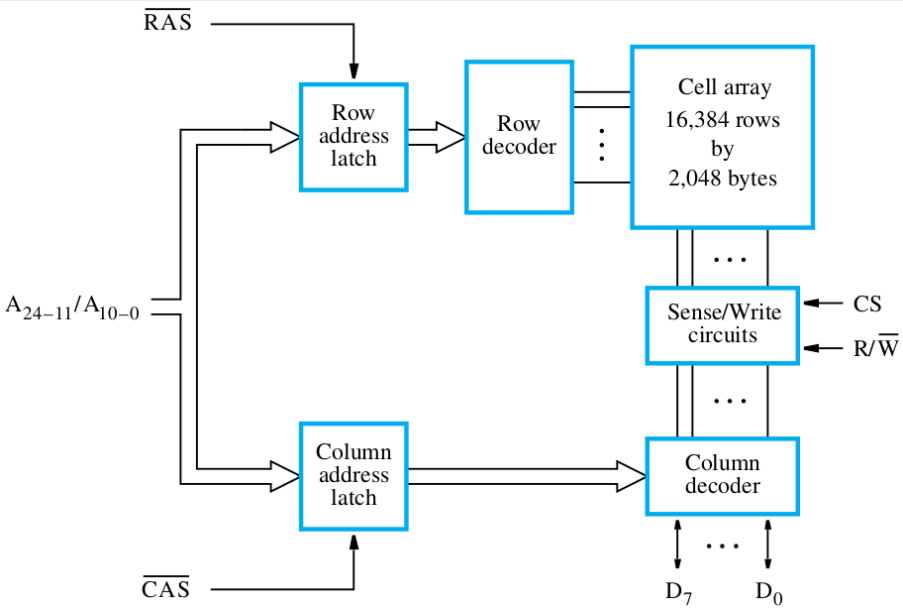
Internal Organisation of Memory Chips

The decoder

Large memory

A 4M x 8 memory, we would need a 22-to-4M decoder and 4M word lines!

Rectangular arrays and two-step decoding process



Internal organization of a 32M x 8 memory chip

- 16K x 16K array
- 16,384 cells in each row divided into 2,048 groups of 8 $\Rightarrow$ 2,048 bytes of data
- 14 address bits to select a row, 11 address bits needed to select a column
- Multiplexing the wires
  - RAS: Row Access Strobe
  - CAS: Column Access Strobe

Memory elements

RAM

The memory cell can be:

- SRAMs (static random access memories)
- DRAMs (dynamic random access memories)

SRAM

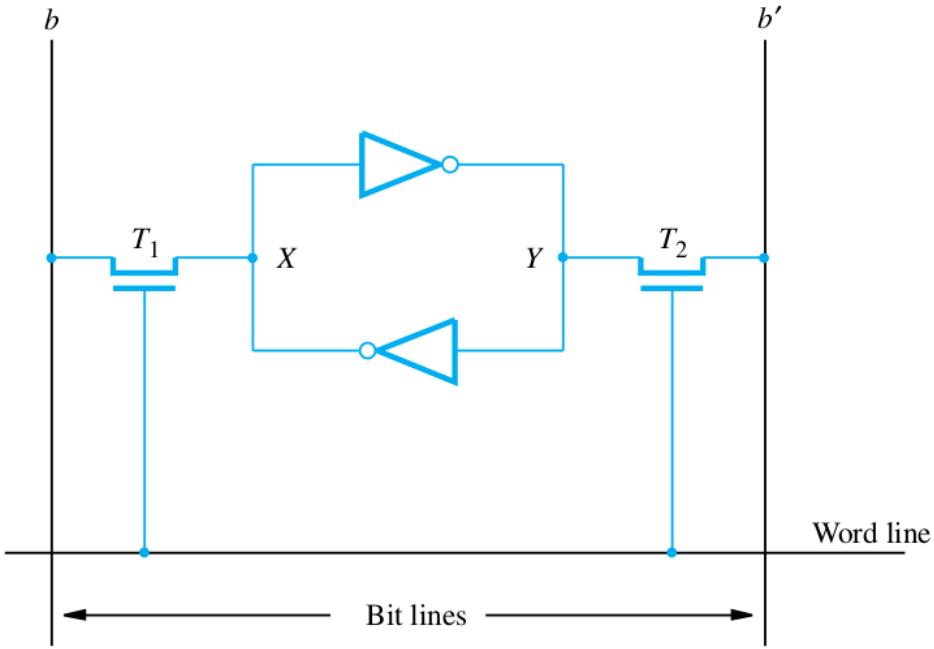
Definition

Static Memory: *Memory capable of retaining its state as long as power is applied* [5]

SRAM

Definition

Static Memory: *Memory capable of retaining its state as long as power is applied* [5]



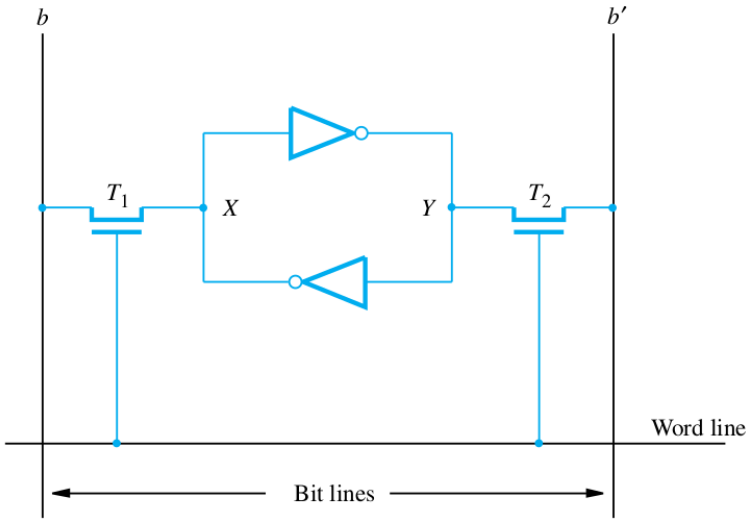
Source: [5]

SRAM: Static Random Access Memory

- Two inverters cross-connected to form a latch
- Two bit lines (T1 and T2)
- *b* and *b'* are always complements

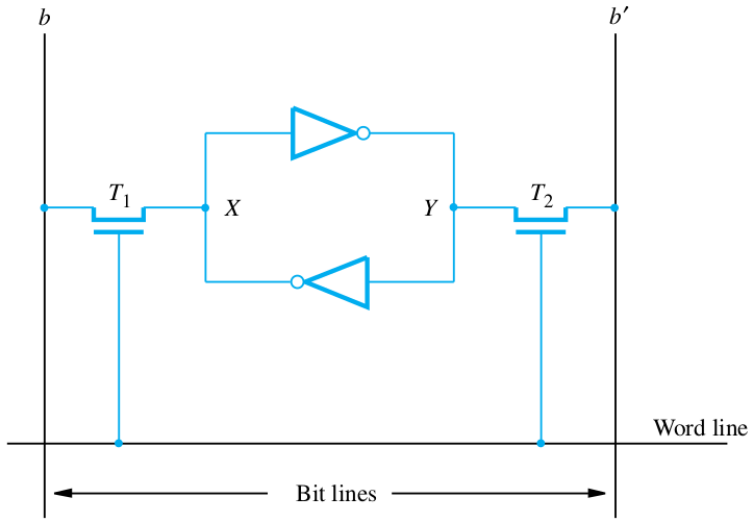
SRAM

Reading and writing an SRAM cell



SRAM

Reading and writing an SRAM cell

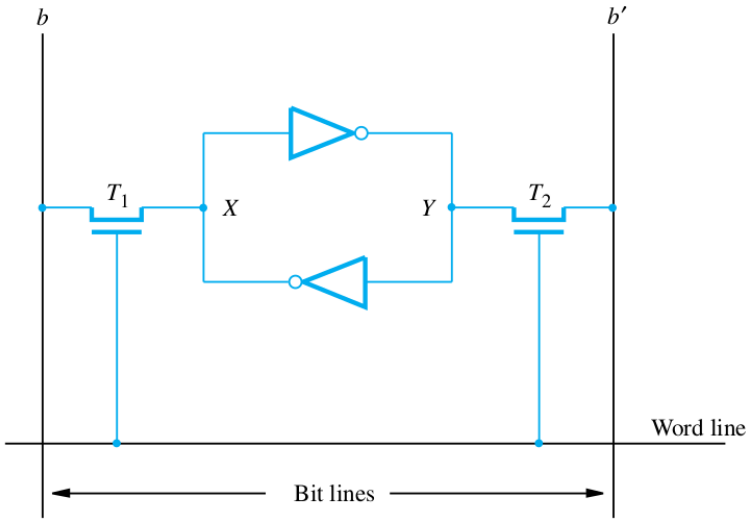


Read Operation

1. The word line is activated to close switches T_1 and T_2
2. The Sense/Write circuit at the end of the two bit lines monitors their state

SRAM

Reading and writing an SRAM cell



Read Operation

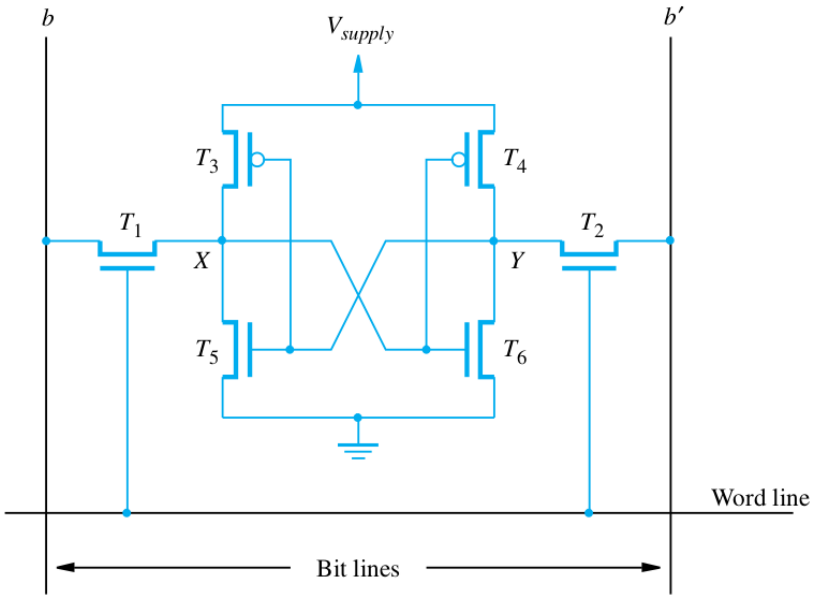
1. The word line is activated to close switches T_1 and T_2
2. The Sense/Write circuit at the end of the two bit lines monitors their state

Write Operation

1. The word line is activated to close switches T_1 and T_2
2. the Sense/Write circuit drives bit lines *b* and *b'*, instead of sensing their state

Memory elements

SRAM



CMOS implementation of SRAM memory cell [5]

CMOS SRAM cell

- Transistor pairs (T_3 , T_5) and (T_4, T_6) form the inverters in the latch
- Continuous power is needed for the cell to retain its state
- Content lost when power down
- Back in stable state when power on (but maybe not the same state)

Volatile memory

SRAMs are *volatile memory* because their loose their content when power is shut down

SRAM

Pros and cons

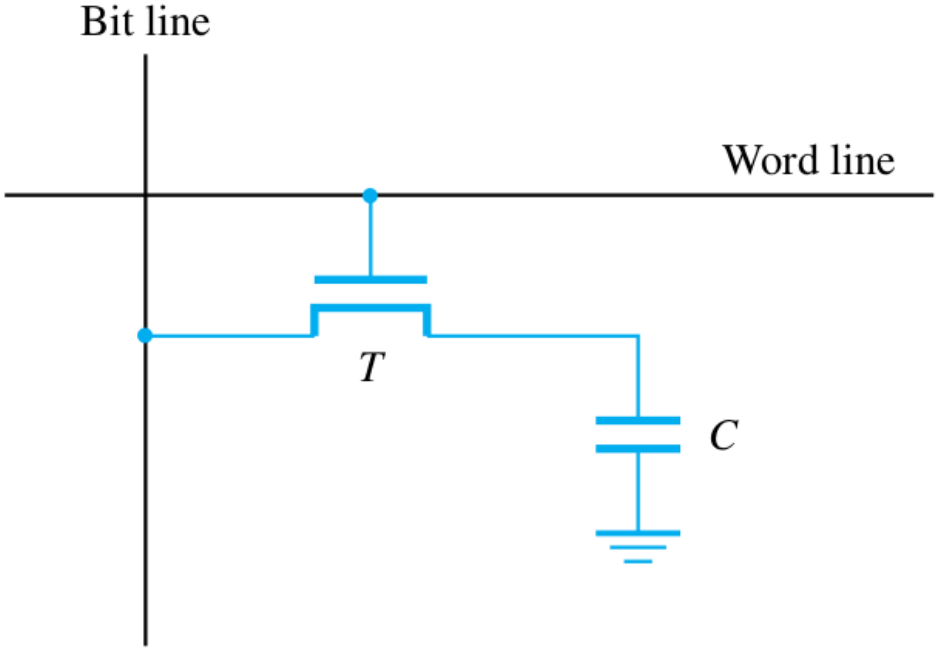
SRAM strong points

- same fabrication process as logic circuit
  - Good integration on processor die
  - Ideal candidate for cache implementation
- Low power consumption
- Fast (+fixed access time)

SRAM weak points

- Expensive
- Low density

DRAM

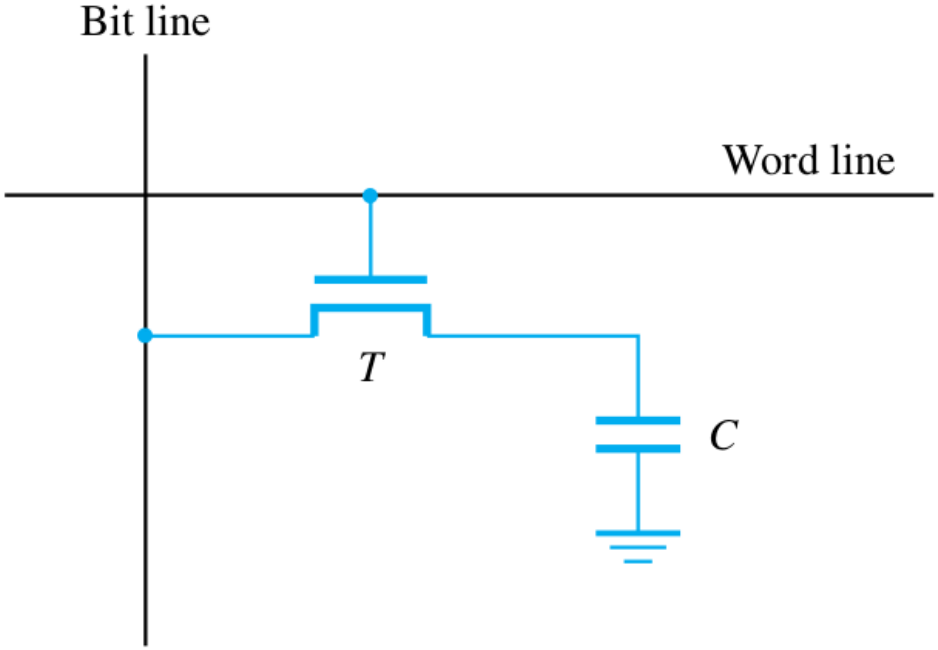


Source: [5]

DRAM: *Dynamic* Random Access Memory

- A capacitor C
- A transistor T

DRAM



Source: [5]

DRAM: *Dynamic* Random Access Memory

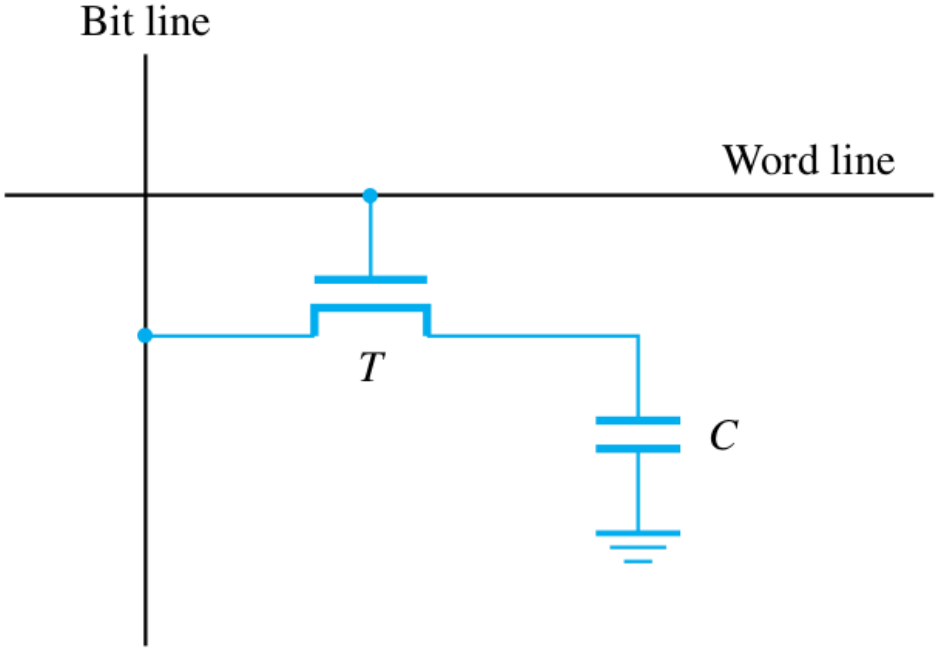
- A capacitor C
- A transistor T

Why is it called *Dynamic*?

The capacitor can retain its state for tens of milliseconds only

Need to *refresh* periodically.

DRAM



Source: [5]

DRAM: *Dynamic* Random Access Memory

- A capacitor C
- A transistor T

Why is it called *Dynamic*?

The capacitor can retain its state for tens of milliseconds only

Need to *refresh* periodically.

Volatile memory

DRAMs are *volatile memory* because their loose their content when power is shut down

DRAM

Definition

Page: a large block of data

Fast Page Mode

Transfer a *page* of data: all bytes of the selected row in sequential order

- no need to reselect the row
- successive CAS signals

DRAM

Definition

Page: a large block of data

Fast Page Mode

Transfer a *page* of data: all bytes of the selected row in sequential order

- no need to reselect the row
- successive CAS signals

SDRAM: *Synchronous* DRAM

Synchronisation with a clock signal

- built-in refresh circuitry (hides the dynamic feature of DRAM to the user)
- *burst* mode: starting address + burst length

DRAM

DDR: Double-Data-Rate SDRAM

- Transfer data on both rising and falling edge of the clock
- Open standard

DRAM

DDR: Double-Data-Rate SDRAM

- Transfer data on both rising and falling edge of the clock
- Open standard

Rambus Memory

- Differential-signaling technique to transfer data to and from the memory chips
- Proprietary scheme that must be licensed

DRAM

Pros and cons

DRAM strong points

- High density
- Low cost per bit

DRAM weak points

- Must be refreshed periodically
- Hard integration of DRAM with logic technology
- Slower than SRAM

Memory chips

Static memory chip

Alliance Memory SRAM, AS6C1008-55STIN- 1048576bit

RS Stock No.: **170-0890** | Mfr. Part No.: **AS6C1008-55STIN** | Brand: Alliance Memory

Available to back order for despatch 23/07/2019

Price Each (In a Tray of 234)

£1.314 (exc. VAT) £1.577 (inc. VAT)

Units	Per unit	Per Tray*
234 - 468	**£1.314**	**£307.476**
702 - 936	£1.297	£303.498
1170 - 2106	£1.28	£299.52

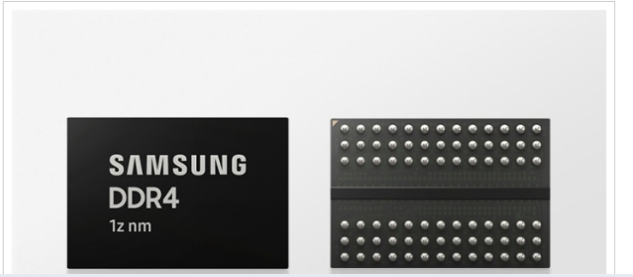
Dynamic memory chip

SEMI-CONDUCTEURS

TROISIÈME GÉNÉRATION DE DRAM DDR4 8 GBITS CHEZ SAMSUNG

RÉDIGÉ PAR FRÉDÉRIC RÉMOND - LUNDI, 25 MARS 2019 12:14

Taille de police | Imprimer | E-mail



DRAM

Memory controller

In charge of:

- 2 steps access: row + column, RAS + CAS signals
- chip select: when multiple memory modules
- refresh: periodic read cycles of asynchronous DRAM

DRAM

Memory controller

In charge of:

- 2 steps access: row + column, RAS + CAS signals
- chip select: when multiple memory modules
- refresh: periodic read cycles of asynchronous DRAM

Refresh overhead

- When internal refresh operation occurs, the memory cannot respond
- Typically few percent of the total time available for accessing the memory
- Still ongoing research activities
  - Goal: hide completely the refresh

Memory technology comparison

Technology	Bytes per Access (typ.)	Latency per Access	Cost per Megabyte[a]	Energy per Access
On-chip Cache	10	100 of picoseconds	$1–100	1 nJ
Off-chip Cache	100	Nanoseconds	$1–10	10–100 nJ
DRAM	1000 (internally fetched)	10–100 nanoseconds	$0.1	1–100 nJ (per device)
Disk	1000	Milliseconds	$0.001	100–1000 mJ

Source: [8]

Choice of technology

- SRAM: small but very fast memory
- DRAM (DDR SDRAM): main memory

ROM

Read Only Memory

Volatile memory

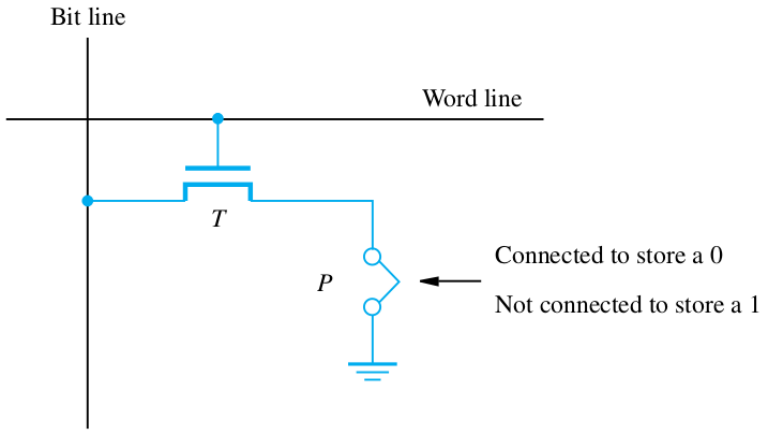
SRAMs and DRAMs are *volatile memory*

Non Volatile memory

Need to store software and data and not loose information when power is shut down

ROM

Read Only Memory



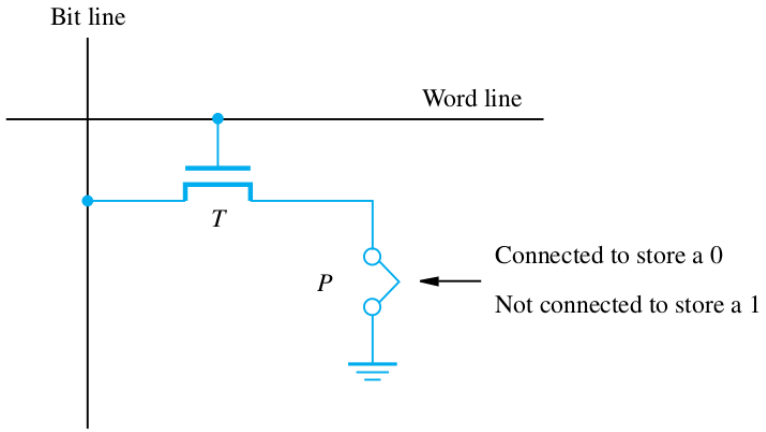
Source: [5]

Configure point P

- Ground: value 0 stored
- Not connected: value 1 stored

ROM

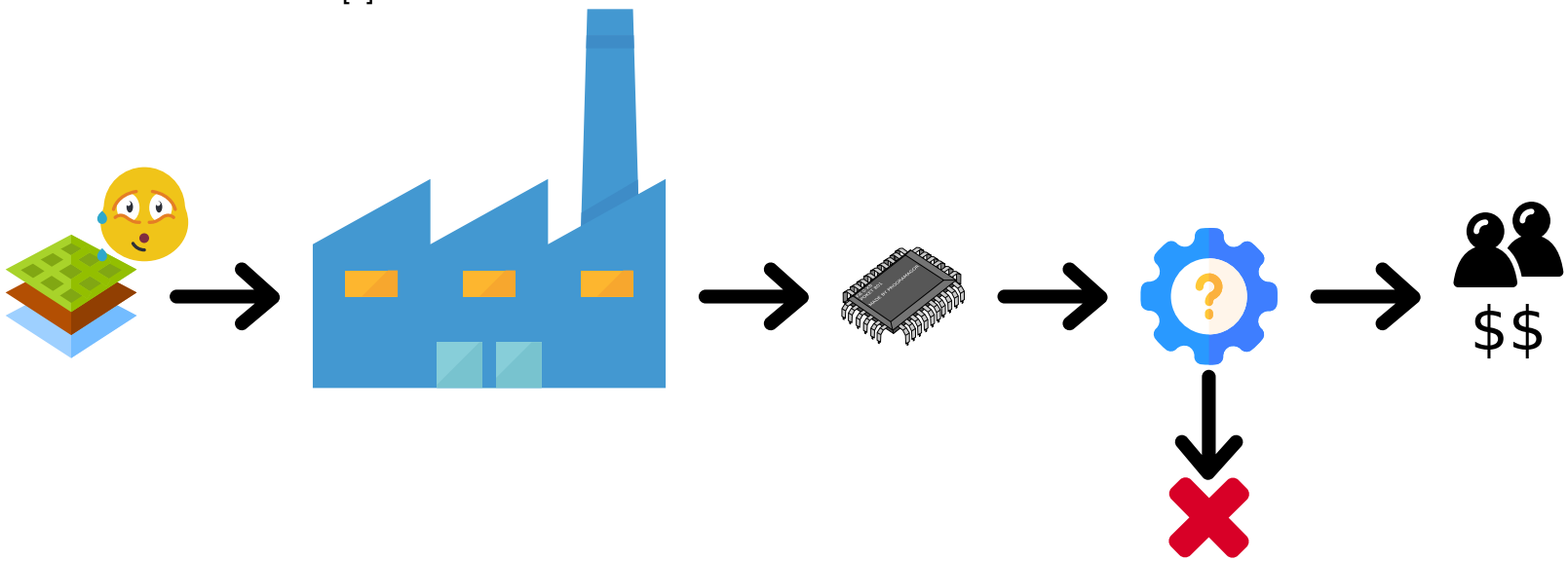
Read Only Memory



Configure point P

- Ground: value 0 stored
- Not connected: value 1 stored

Source: [5]

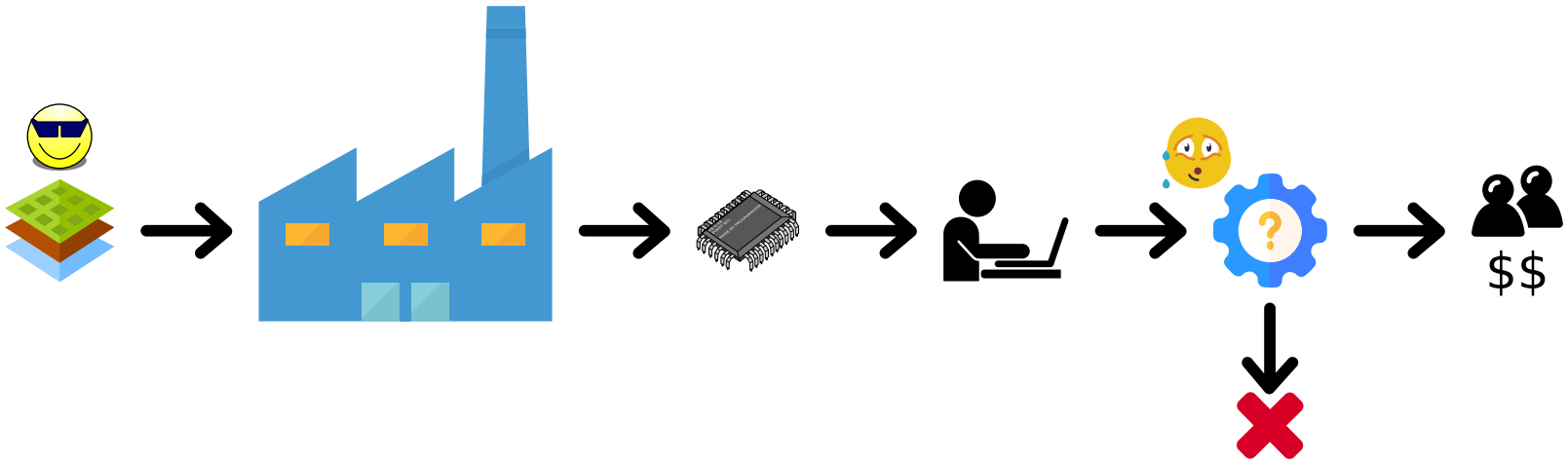


PROM

Programmable Read Only Memory

PROM

"Programmable" through a fuse



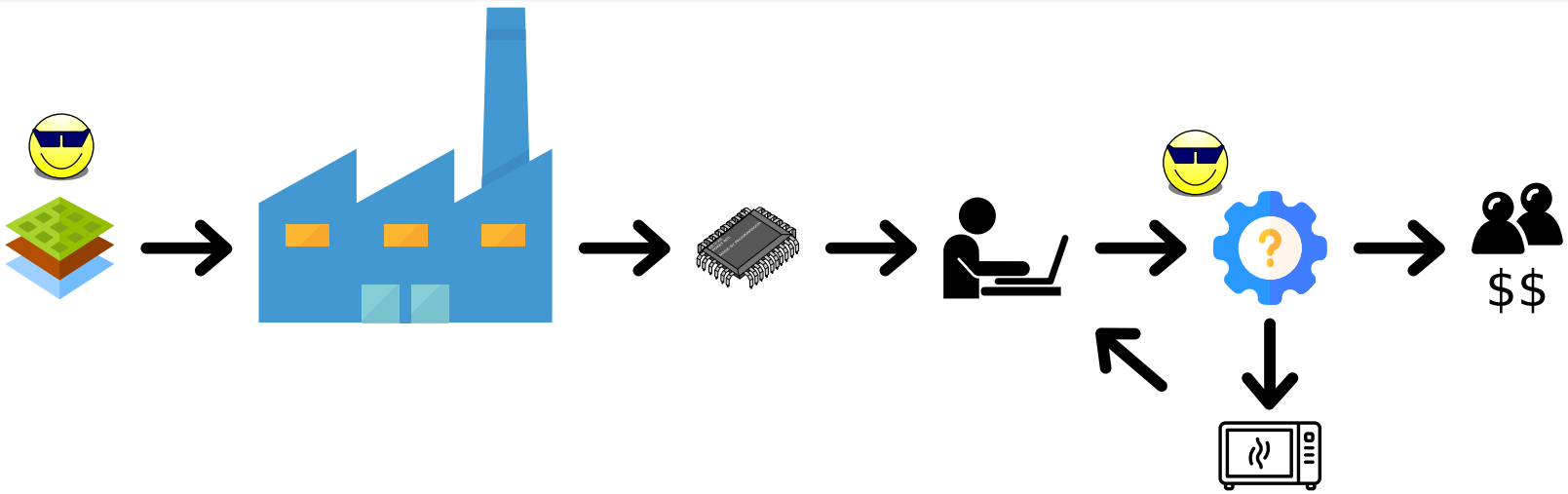
EPROM

Erasable Programmable Read Only Memory

EPROM

"Erasable and Programmable" through a *special transistor*

Expose the chip to ultraviolet light to erase



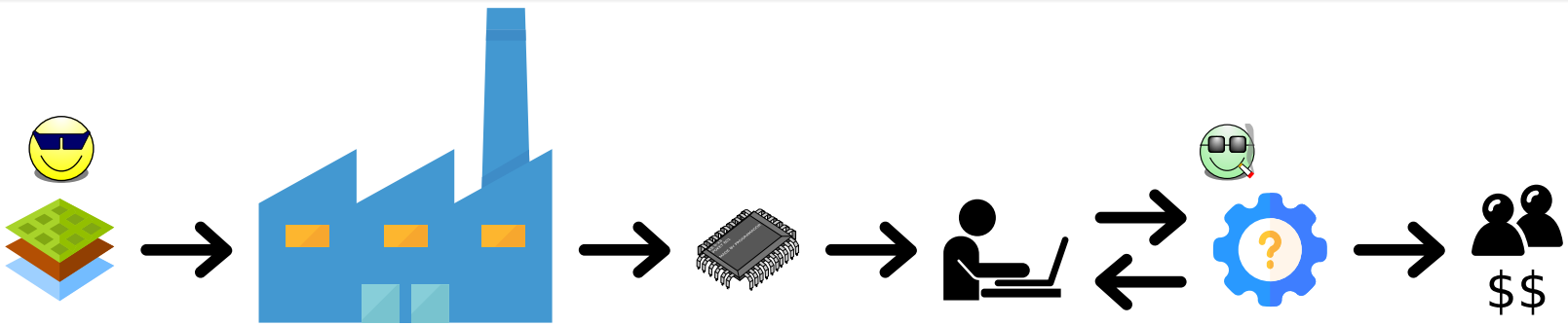
EEPROM

Electrically Erasable Programmable Read Only Memory

EEPROM

different voltages are needed for erasing, writing, and reading the stored data

$\Rightarrow$ circuit complexity



Flash Memory

Flash Memory

- read the contents of a single cell
- write an entire block of cells

Flash Cards

Flash drives (SSD)

2 To, SATA 3 (6 Gb/s), 2,5"

Types of memories

The sore point

volatile vs non-volatile

RAM vs ROM

RAM: Random Access Memory

RAM = volatile?

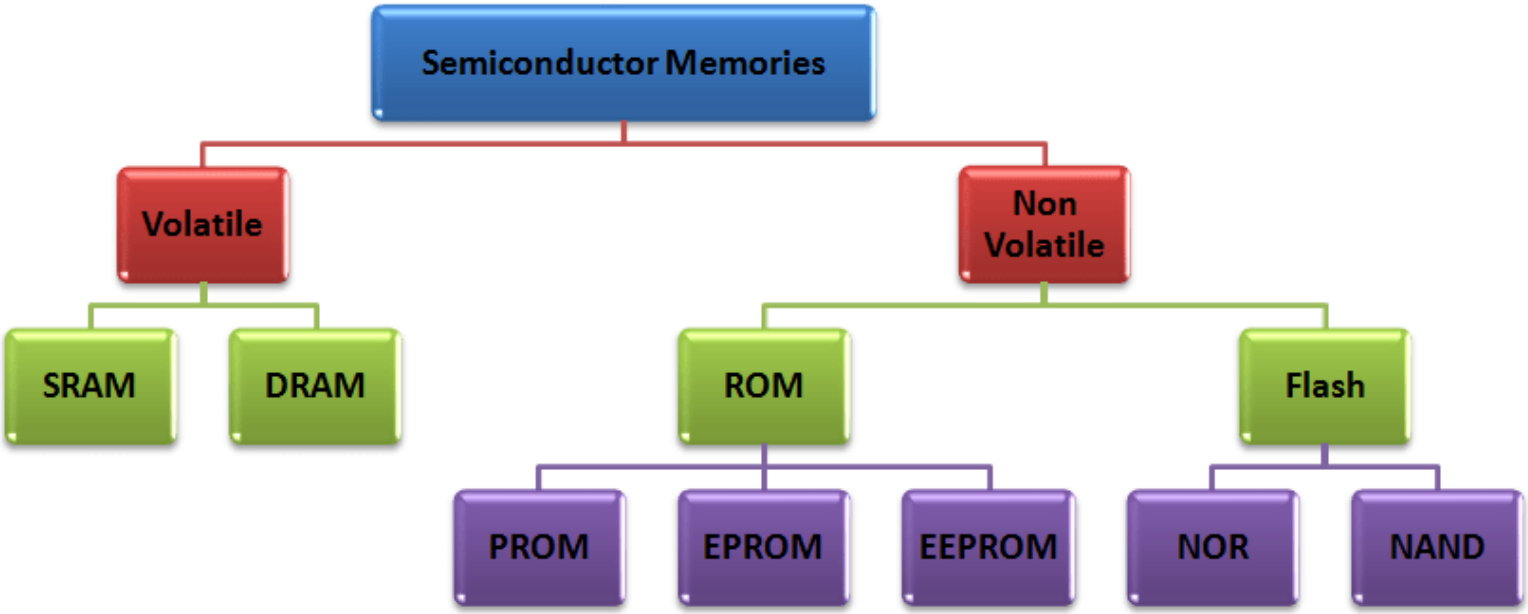
ROM: Read Only Memory

ROM = non-volatile?

What about writing data in a non-volatile memory?

e.g. Hard disk drive, USB key, ...

Memory classification

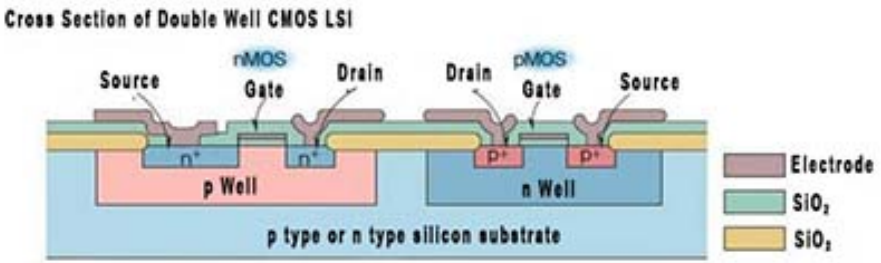


Source: [11]

Outline

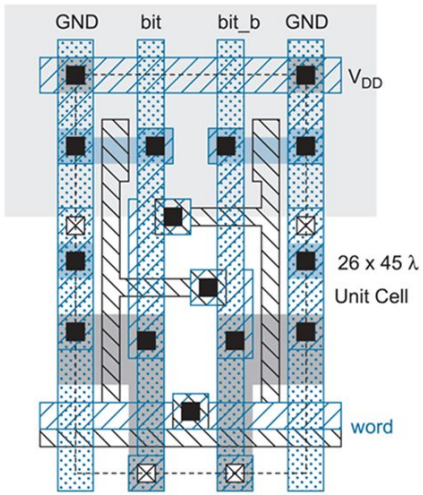
A little bit of techno

SRAM

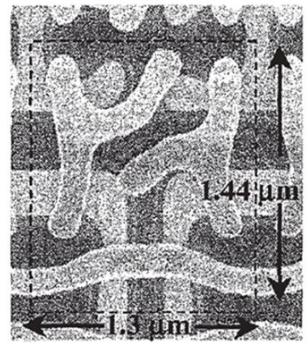


Source: http://www.shmj.or.jp/innovation50/english/detail_D05E.htm

Layout of 6T SRAM Cell



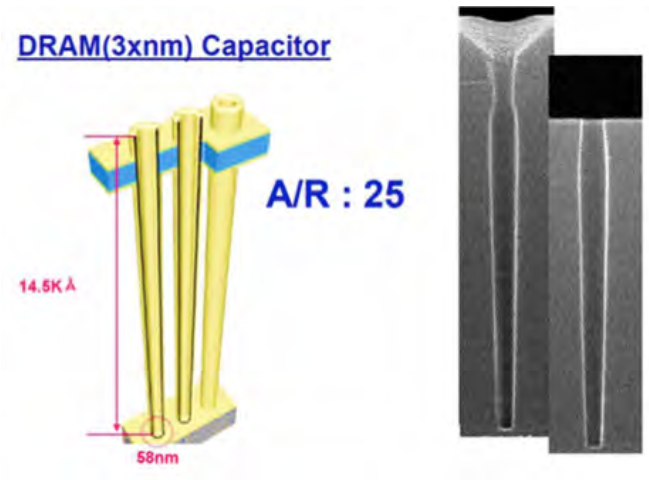
Only poly and diff layers are shown.



Source:Digital Design: Principles and Practices, Published by Jonas Wilkerson

A little bit of techno

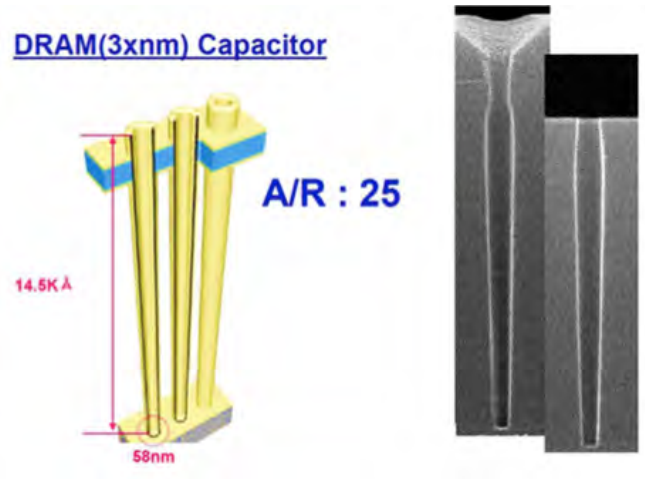
DRAM



Source: [7]

A little bit of techno

DRAM



Source: [7]

Did you know?

DRAM's capacity has been one of the more consistent incarnations of Moore's law [7]

- scaled in capacity by a factor of over 16 million
- 1 kbits on a die in 1970 to 16 Gbits today

A little bit of techno

Emerging technologies

Patience, grasshoper. . .

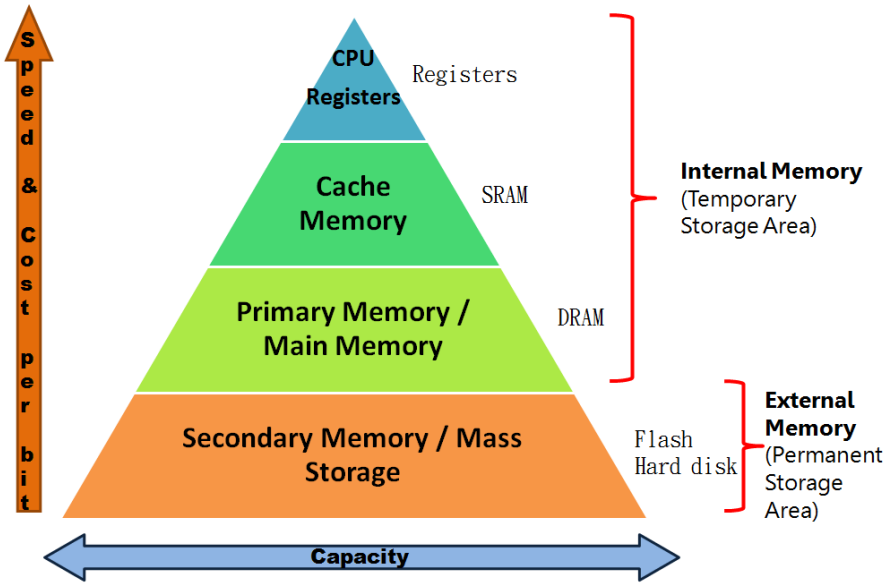
This is discussed later

Part II

Memory system

Outline of Part II

Overview of computer memory organisation



From the I/O device to the processor

The data need to move across the levels

- From the I/O to the main memory
- From the main memory to the cache
- From the cache to CPU registers

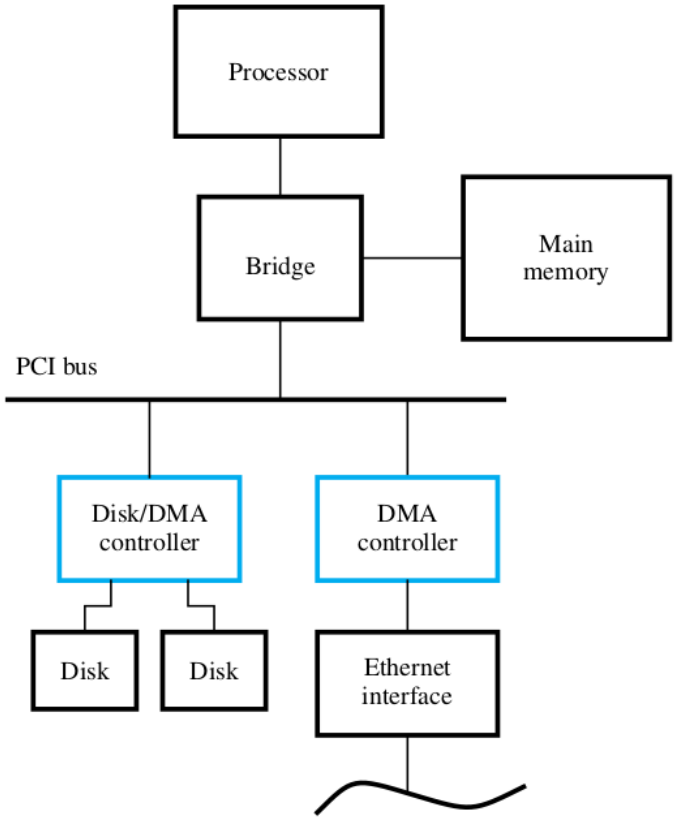
Managing data movement

Offload the processor from weighty data movement tasks

Outline

DMA

Direct Memory Access



DMA

A special control unit to manage the transfer, without continuous intervention by the processor

DMA

Direct Memory Access

Under supervision (usually operating system)

- Processor provides:
  - starting address
  - the number of words in the block
  - the direction of the transfer
- DMA raises an interrupt when finished

Outline

Caches

The illusion of a large memory

accessible as fast as a small memory [10]

Caches

The illusion of a large memory

accessible as fast as a small memory [10]

Idea

A processor does not need to access all of the program codes and data at once. Let's keep near the useful parts.

Caches

The illusion of a large memory

accessible as fast as a small memory [10]

Idea

A processor does not need to access all of the program codes and data at once. Let's keep near the useful parts.

Principle of locality

- Temporal locality (locality in time): if an item is referenced, it will tend to be referenced again soon.
- Spatial locality (locality in space): if an item is referenced, items whose addresses are close by will tend to be referenced soon.

Caches

Definition

- *block* or *line*: the minimum unit of information that can be either present or not present in a cache.
- *hit*: the data requested is present in the cache
- *miss*: the data requested is NOT present in the cache
- *hit rate* or *hit ratio*: the fraction of memory accesses found in the upper level
- *miss rate* (1 - hit rate): the fraction of memory accesses NOT found in the upper level

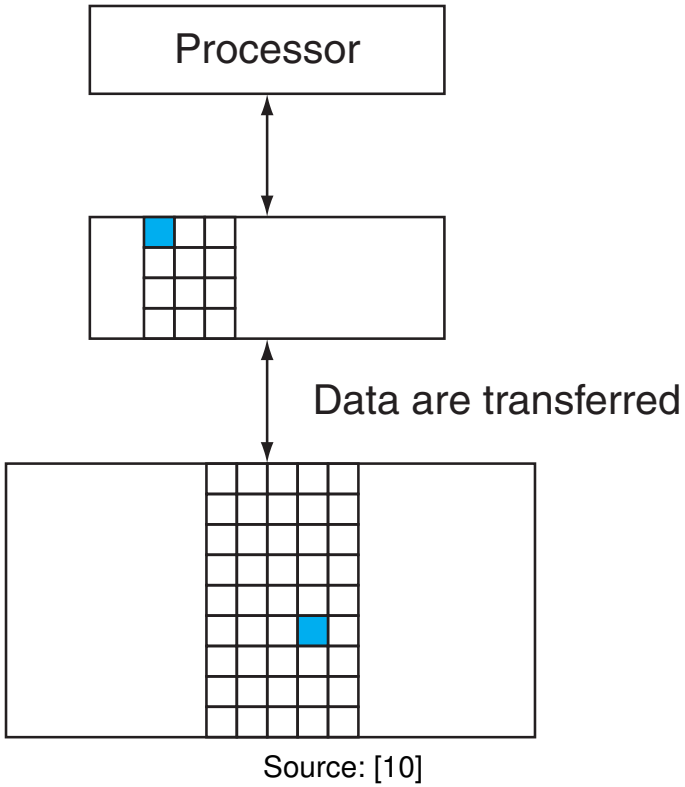
Caches

Message to programmers

No, memory is NOT a flat, random access device

You need to understand memory hierarchy to get good performance

The big picture



Source: [10]

memory hierarchy

- upper and lower level
- transfer entire block between levels

The serious game!

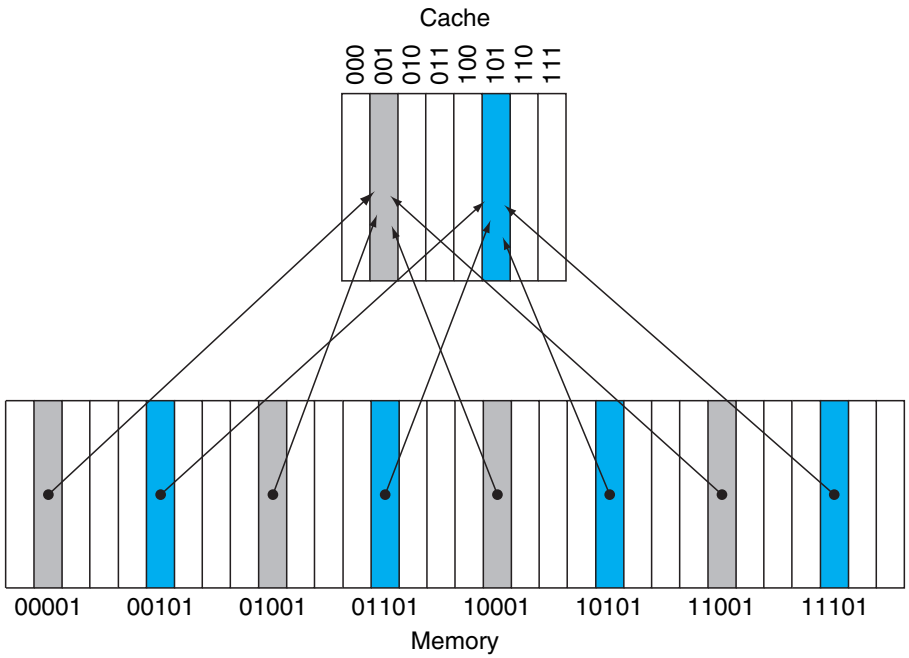
Caches

Direct-map cache

Direct-mapped cache

A cache structure in which each memory location is mapped to exactly one location in the cache

Position = (Block number) modulo (Number of *blocks* in the cache)



Source: [10] A direct-mapped cache with eight entries showing the addresses of memory words between 0 and 31 that map to the same cache locations

Simple to implement

High miss rate

Caches

Fully Associative Cache

Fully Associative Cache

A cache structure in which a block can be placed in any location in the cache

Low miss rate

Hardware cost

Caches

n-way set-associate cache

set-associative cache

- A cache that has a fixed number of locations (at least two) where each block can be placed
- Each block in the memory maps to a unique set in the cache
- A block can be placed in *any* element of that set
- n is the number of places in the set

Good trade-off between direct-map and fully associative

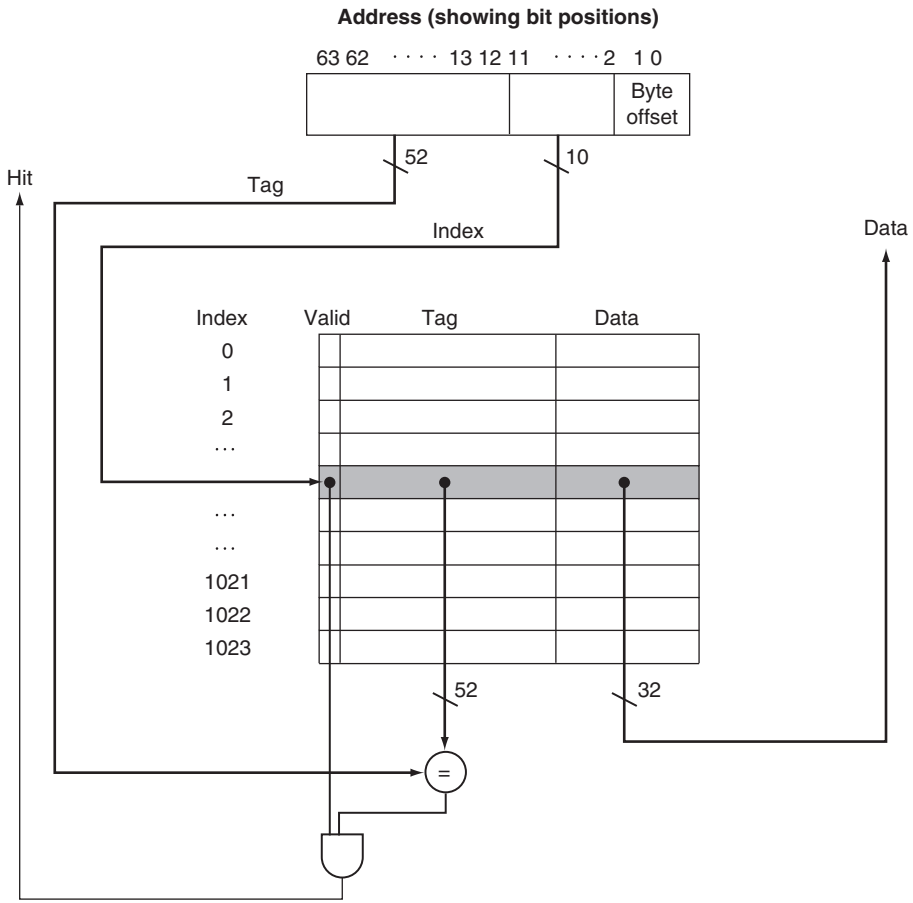
Finding the good trade-off!

Caches

Tags

Definition

- *Tag*: A field in a table used for a memory hierarchy that contains the address information required to identify whether the associated block in the hierarchy corresponds to a requested word
- *Valid bit*: A field in the tables of a memory hierarchy that indicates that the associated block in the hierarchy contains valid data



The lower portion of the address is used to select a cache entry consisting of a data word and a tag [10]

Caches

Tags

Size of tags versus associativity

Increasing associativity requires more comparators and more tag bits

Cache of 4096 blocks, four-word block size, 64-bit address

16 bytes per block $\Rightarrow$ 64-4=60 bits for index and tag

- Direct-map cache
  - number of sets = number of blocks $\Rightarrow$ 12 bits of index ($log_2(4096) = 12$)
  - $(60 - 12) \times 4096 = 192$ Ki tag bits
- Two way associative cache
  - number of sets = 2048 $\Rightarrow$ 11 bits of index ($log_2(2048) = 11$)
  - $(60 - 11) \times 2 \times 2048 = 196$ Ki tag bits
- Fully associative cache
  - number of sets = 1 $\Rightarrow$ 0 bits of index ($log_2(1) = 0$)
  - $60 \times 1 \times 4096 = 240$ Ki tag bits

Caches

Tags

Size of tags versus associativity

Increasing associativity requires more comparators and more tag bits

Size given by the manufacturer

The size of the cache given by the manufacturer does not include the size of tags + valid bit.

Caches

CAM

Content Addressable Memory

A circuit that combines comparison and storage in a single device

- RAM: supply address, return data
- CAM: supply data, return index

Caches

Handling writes

Consistency

When the cache and the main memory have different values, they are *inconsistent*

Caches

Handling writes

Consistency

When the cache and the main memory have different values, they are *inconsistent*

Write-through

- Always update cache AND next lower level of the hierarchy
- Processor is *stall* during writing to main memory ($\approx$ 100 cycles)
- Use of *write buffer* to free the processor

Caches

Handling writes

Consistency

When the cache and the main memory have different values, they are *inconsistent*

Write-through

- Always update cache AND next lower level of the hierarchy
- Processor is *stall* during writing to main memory ($\approx$ 100 cycles)
- Use of *write buffer* to free the processor

Write-back

- Update the cache only
- Update next lower level when the block is replaced
- Improve performance but more complex to implement

Which block to replace?

- Intuitively, replace the one that has gone the longest time without being referenced
  - *Least Recently Used* (LRU)
  - keep track of all references by means of counters
- The simplest algorithm: randomly choose the block to be replaced
  - quite effective in practice
- many others:
  - FIFO (*First In First Out*)
  - LFU (*Least Frequently Used*)
  - pseudo-LRU

Caches

Split cache

Two independent caches

- instruction cache
- data cache

operating in parallel

Harvard computer style

Can the split cache be considered as an implementation of Harvard architecture

Caches

Multilevel cache

Close the gap between primary cache and DRAM

- Primary cache: focus on minimizing hit time
  - Smaller block size, reduce miss penalty
- Secondary cache: focus on minimizing the miss rate
  - Larger total size, higher associativity

Caches

The three Cs: behavior of memory hierarchies

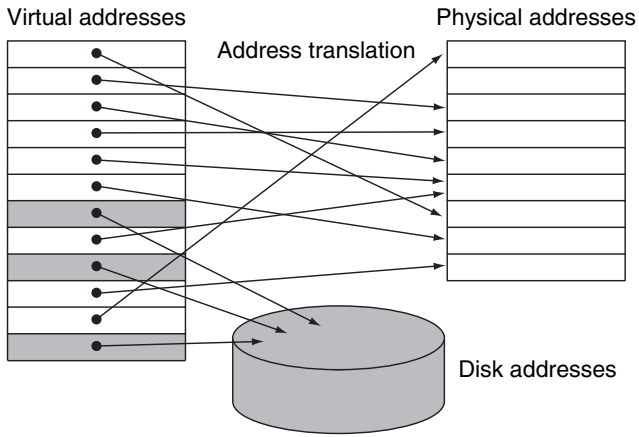
- *Compulsory miss* (or *cold-start miss*): A cache miss caused by the first access to a block that has never been in the cache
- *Capacity miss*: A cache miss that occurs because the cache, even with full associativity, cannot contain all the blocks needed to satisfy the request
- *Conflict miss*: A cache miss that occurs in a set-associative or direct mapped cache when multiple blocks compete for the same set and that are eliminated in a fully associative cache of the same size

Outline

Virtual memory

Two historical motivations

1. efficient and safe sharing of memory among several programs
2. remove the programming burden of a small limited amount of main memory

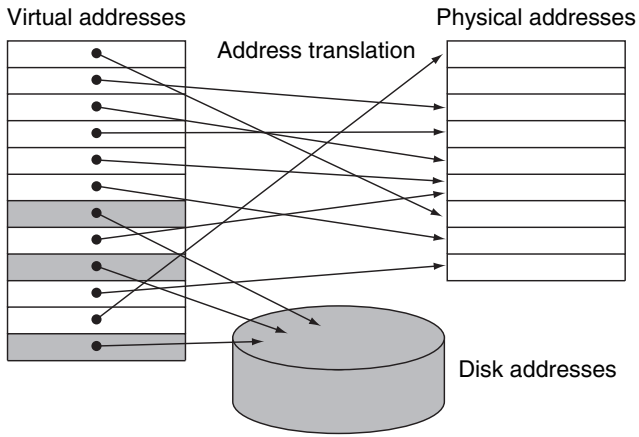


In virtual memory, blocks of memory (called pages) are mapped from one set of addresses (called virtual addresses) to another set (called physical addresses) [10]

Virtual memory

Two historical motivations

1. efficient and safe sharing of memory among several programs
2. remove the programming burden of a small limited amount of main memory



In virtual memory, blocks of memory (called pages) are mapped from one set of addresses (called virtual addresses) to another set (called physical addresses) [10]

Size of virtual address space

- A 32-bit processor can address up to $2^{32} = 4Gi$ elements
- A 64-bit processor can address up to $2^{64} = 16Ei$ (exbi) elements ($> 10^{18}$)

Virtual memory

Definition

- Virtual memory: A technique that uses main memory as a “cache” for secondary storage

Virtual memory

Definition

- Virtual memory: A technique that uses main memory as a “cache” for secondary storage
- Physical address: An address in main memory

Virtual memory

Definition

- Virtual memory: A technique that uses main memory as a “cache” for secondary storage
- Physical address: An address in main memory
- Protection: A set of mechanisms for ensuring that multiple processes sharing the processor, memory, or I/O devices cannot interfere, intentionally or unintentionally, with one another by reading or writing each other’s data. These mechanisms also isolate the operating system from a user process

Virtual memory

Definition

- Virtual memory: A technique that uses main memory as a “cache” for secondary storage
- Physical address: An address in main memory
- Protection: A set of mechanisms for ensuring that multiple processes sharing the processor, memory, or I/O devices cannot interfere, intentionally or unintentionally, with one another by reading or writing each other’s data. These mechanisms also isolate the operating system from a user process
- Page fault: An event that occurs when an accessed page is not present in main memory

Virtual memory

Definition

- Virtual memory: A technique that uses main memory as a “cache” for secondary storage
- Physical address: An address in main memory
- Protection: A set of mechanisms for ensuring that multiple processes sharing the processor, memory, or I/O devices cannot interfere, intentionally or unintentionally, with one another by reading or writing each other’s data. These mechanisms also isolate the operating system from a user process
- Page fault: An event that occurs when an accessed page is not present in main memory
- Virtual address: An address that corresponds to a location in virtual space and is translated by address mapping to a physical address when memory is accessed

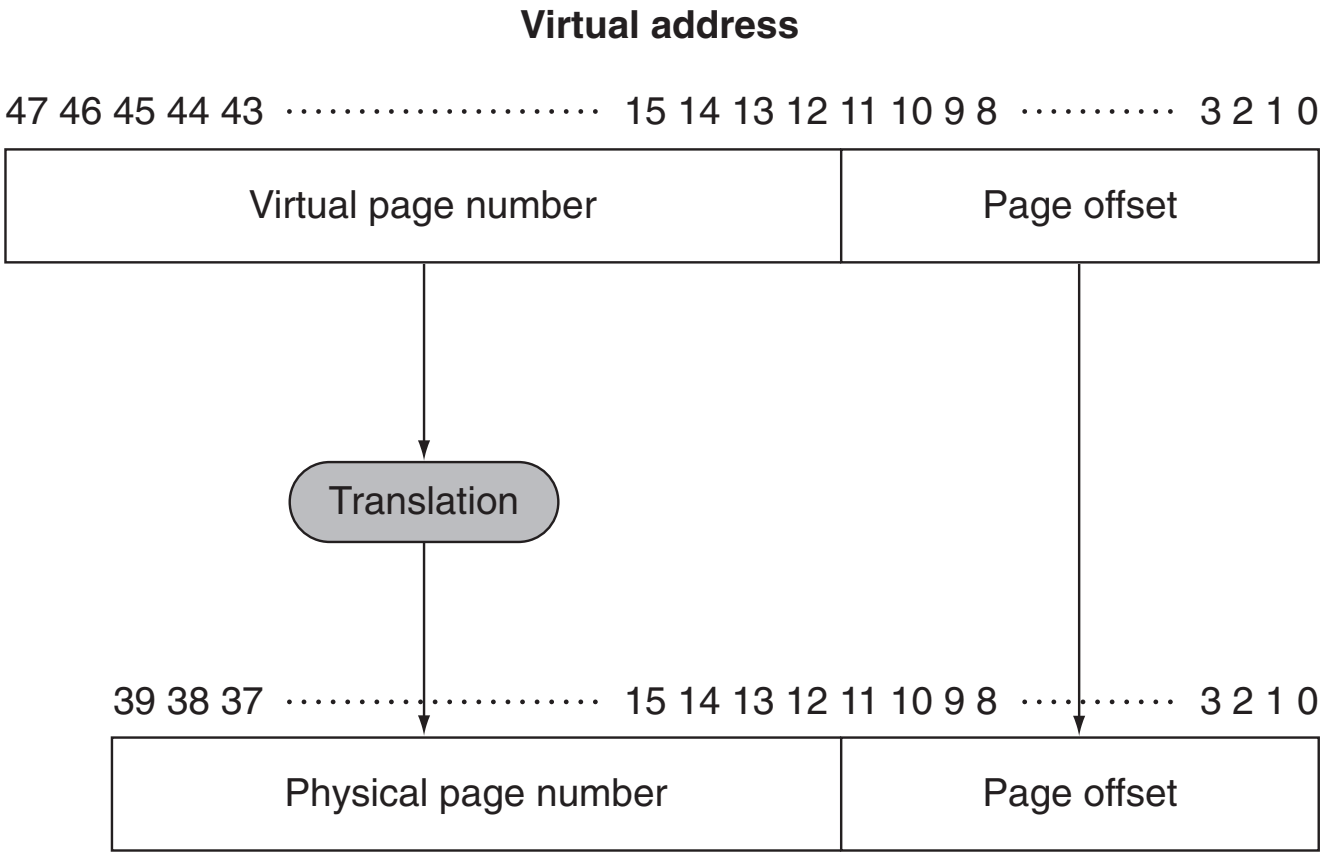
Virtual memory

Definition

- Virtual memory: A technique that uses main memory as a “cache” for secondary storage
- Physical address: An address in main memory
- Protection: A set of mechanisms for ensuring that multiple processes sharing the processor, memory, or I/O devices cannot interfere, intentionally or unintentionally, with one another by reading or writing each other’s data. These mechanisms also isolate the operating system from a user process
- Page fault: An event that occurs when an accessed page is not present in main memory
- Virtual address: An address that corresponds to a location in virtual space and is translated by address mapping to a physical address when memory is accessed
- Address translation (or address mapping): The process by which a virtual address is mapped to an address used to access memory

Virtual memory

Mapping



Mapping from a virtual to a physical address. The page size is $2^{12} = 4$ KiB. The number of physical pages allowed in memory is 2^{28}, since the physical page number has 28 bits in it. Thus, main memory can have at most 1 TiB, while the virtual address space is 256 TiB [10].

Virtual memory

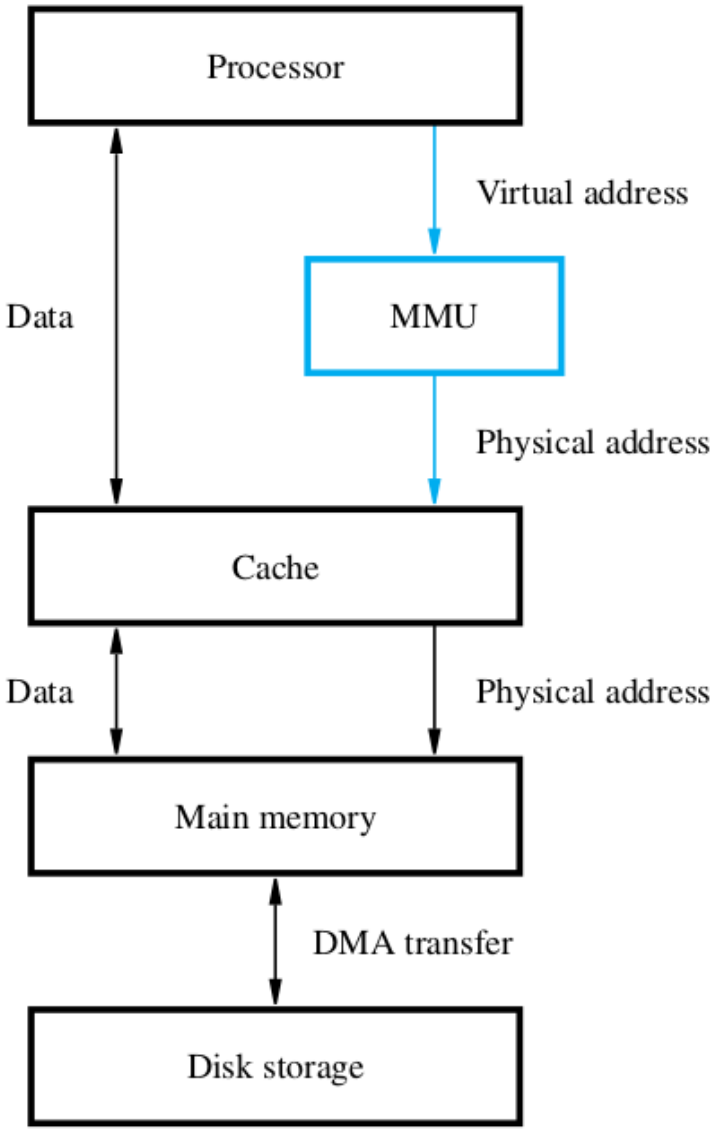
High cost of a page fault

Enormous miss penalty: 1 page fault = millions of clock cycles

Key decisions:

- Page should be large enough to amortize the high access time (4 KiB to 64 KiB)
- Allow fully associative placement
- Software page handling (faults and placement)
- Write-back (write-through takes too long)

Virtual memory



Virtual memory organization [5]

Memory Management Unit (MMU)

- keeps track of which parts of the virtual address space are in the physical memory
- translates the virtual address into the corresponding physical address

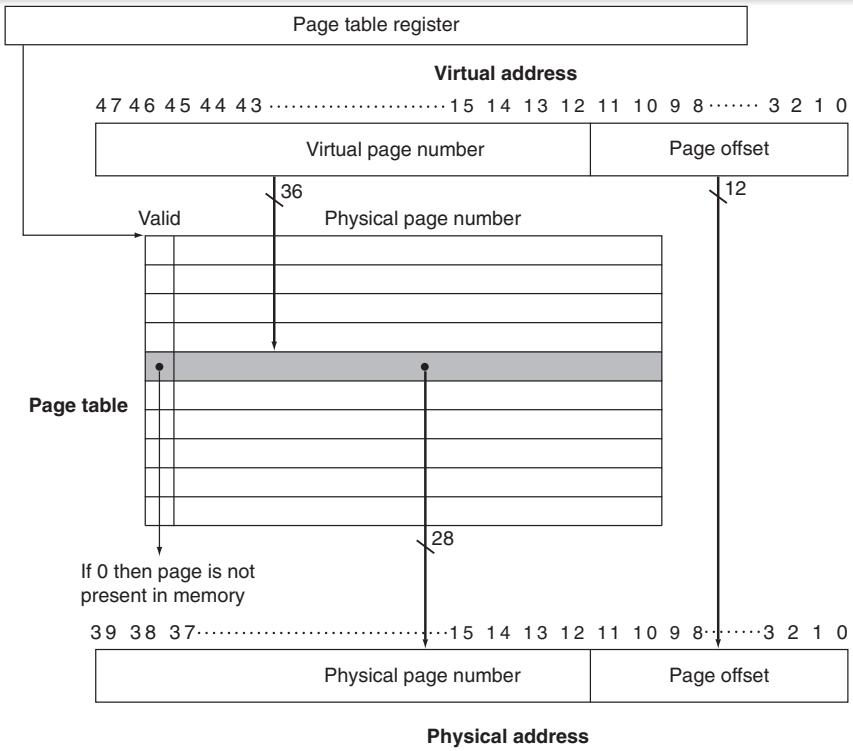
Virtual memory

Placing a page and finding it again

Definition

Page Table: The table containing the virtual to physical address translations

- stored in memory, indexed by the virtual page number
- each entry in the table contains the physical page number for that virtual page



- Each program has its own page table
- *page table register*: the start of the page table
- *valid bit*: page present or not in memory

The page table is indexed with the virtual page number to

Virtual memory

Page faults

- Page fault when the valid bit for a virtual page is off
- Software exception
- Operating system gets control
  - find the page
  - decide where to place it in main memory
  - *Least Recently Used* replacement scheme

Virtual memory

Page faults

- Page fault when the valid bit for a virtual page is off
- Software exception
- Operating system gets control
  - find the page
  - decide where to place it in main memory
  - *Least Recently Used* replacement scheme

The operating system is a process

The tables controlling the memory are in the memory.

We need to access to the memory to access to the memory!

Virtual memory

Making the translation fast

Definition

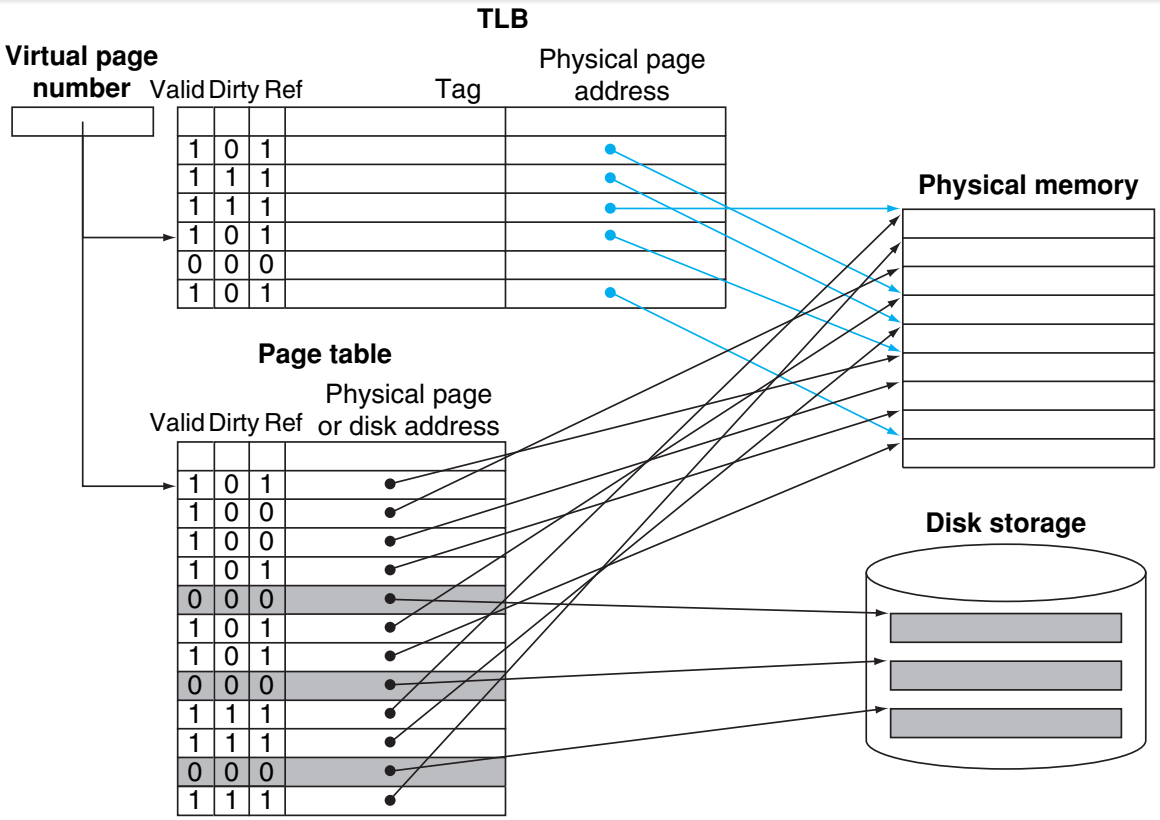
TLB: translation-lookaside buffer. A cache that keeps track of recently used address mappings to try to avoid an access to the page table

Virtual memory

Making the translation fast

Definition

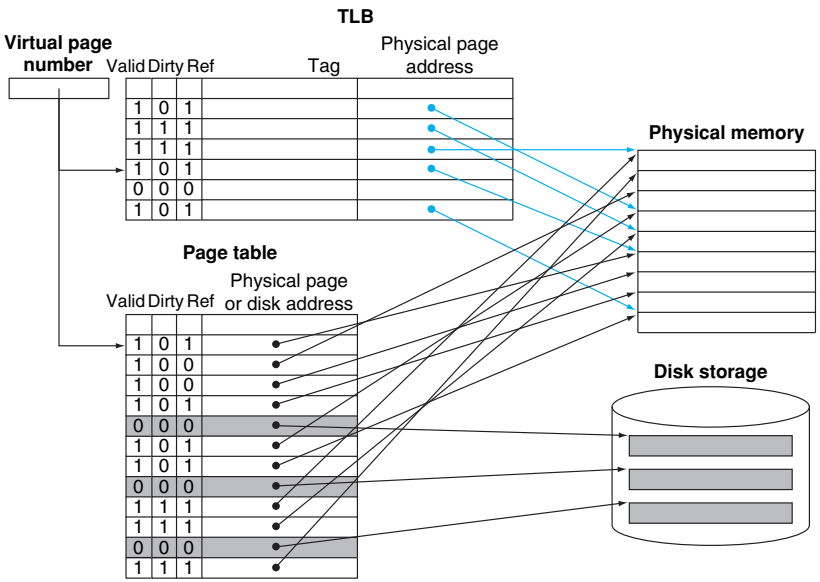
TLB: translation-lookaside buffer. A cache that keeps track of recently used address mappings to try to avoid an access to the page table



The TLB acts as a cache of the page table for the entries that map to physical pages only [10]

Virtual memory

Making the translation fast



The TLB acts as a cache of the page table for the entries that map to physical pages only [10]

TLB

- TLB is a cache, it must have a tag field
- If TLB miss, check the page table
- Typical values:
  - TLB size: 16-512 entries
  - Block size: 1-2 page table entries
  - Miss penalty: 10-100 clock cycles
  - Miss rate: 0.01%-1%

Virtual memory

Putting it all together: Virtual memory, TLBs, and Caches

Interaction

- Virtual memory and cache systems work together
- Under the supervision of the operating system

Virtual memory

Putting it all together: Virtual memory, TLBs, and Caches

Interaction

- Virtual memory and cache systems work together
- Under the supervision of the operating system

Best case

Virtual address translated by TLB, sent to cache where data is found, retrieved and sent back to the processor

Virtual memory

Putting it all together: Virtual memory, TLBs, and Caches

Interaction

- Virtual memory and cache systems work together
- Under the supervision of the operating system

Best case

Virtual address translated by TLB, sent to cache where data is found, retrieved and sent back to the processor

Worst case

Miss in all three components: TLB, page table, cache

Virtual memory

TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.

The possible combinations of events in the TLB, virtual memory system and cache [10]

Virtual memory

Tag, index, virtual and physical addresses

Four types of organisation

1. *Physically indexed, physically tagged*: time to memory = TLB access + cache access

Virtual memory

Tag, index, virtual and physical addresses

Four types of organisation

1. *Physically indexed, physically tagged*: time to memory = TLB access + cache access
2. *Virtually indexed, virtually tagged*: time to memory = cache access (TLB used when cache miss only)

Virtual memory

Tag, index, virtual and physical addresses

Four types of organisation

1. *Physically indexed, physically tagged*: time to memory = TLB access + cache access
2. *Virtually indexed, virtually tagged*: time to memory = cache access (TLB used when cache miss only)
   - *Aliasing*: when two virtual addresses target the same physical page

Virtual memory

Tag, index, virtual and physical addresses

Four types of organisation

1. *Physically indexed, physically tagged*: time to memory = TLB access + cache access
2. *Virtually indexed, virtually tagged*: time to memory = cache access (TLB used when cache miss only)
   - *Aliasing*: when two virtual addresses target the same physical page
   - The same word may be cached in two different locations

Virtual memory

Tag, index, virtual and physical addresses

Four types of organisation

1. *Physically indexed, physically tagged*: time to memory = TLB access + cache access
2. *Virtually indexed, virtually tagged*: time to memory = cache access (TLB used when cache miss only)
   - *Aliasing*: when two virtual addresses target the same physical page
   - The same word may be cached in two different locations
   - Need: specific design + operating system + user!

Virtual memory

Tag, index, virtual and physical addresses

Four types of organisation

1. *Physically indexed, physically tagged*: time to memory = TLB access + cache access
2. *Virtually indexed, virtually tagged*: time to memory = cache access (TLB used when cache miss only)
   - *Aliasing*: when two virtual addresses target the same physical page
   - The same word may be cached in two different locations
   - Need: specific design + operating system + user!
3. *Virtually indexed, physically tagged*: the common trade-off

Virtual memory

Tag, index, virtual and physical addresses

Four types of organisation

1. *Physically indexed, physically tagged*: time to memory = TLB access + cache access
2. *Virtually indexed, virtually tagged*: time to memory = cache access (TLB used when cache miss only)
   - *Aliasing*: when two virtual addresses target the same physical page
   - The same word may be cached in two different locations
   - Need: specific design + operating system + user!
3. *Virtually indexed, physically tagged*: the common trade-off
4. *Physically indexed, virtually tagged*: (double drawbacks?)

Virtual memory

Tag, index, virtual and physical addresses

Four types of organisation

1. *Physically indexed, physically tagged*: time to memory = TLB access + cache access
2. *Virtually indexed, virtually tagged*: time to memory = cache access (TLB used when cache miss only)
   - *Aliasing*: when two virtual addresses target the same physical page
   - The same word may be cached in two different locations
   - Need: specific design + operating system + user!
3. *Virtually indexed, physically tagged*: the common trade-off
4. *Physically indexed, virtually tagged*: (double drawbacks?)
   - Used in MIPS R6000 [12]

Virtual memory

Implementing protection

Context

Sharing a single main memory by multiple processes

Virtual memory

Implementing protection

Context

Sharing a single main memory by multiple processes

Three basic capabilities:

1. *Supervisor mode* (or *kernel mode*): mode indicating that a running process is an operating system process.

Virtual memory

Implementing protection

Context

Sharing a single main memory by multiple processes

Three basic capabilities:

1. *Supervisor mode* (or *kernel mode*): mode indicating that a running process is an operating system process.
2. Processor state readable (but not writable) by a user process: user/supervisor mode bit + page table pointer + TLB

Virtual memory

Implementing protection

Context

Sharing a single main memory by multiple processes

Three basic capabilities:

1. *Supervisor mode* (or *kernel mode*): mode indicating that a running process is an operating system process.
2. Processor state readable (but not writable) by a user process: user/supervisor mode bit + page table pointer + TLB
3. From user mode to supervisor mode (and vice versa):

Virtual memory

Implementing protection

Context

Sharing a single main memory by multiple processes

Three basic capabilities:

1. *Supervisor mode* (or *kernel mode*): mode indicating that a running process is an operating system process.
2. Processor state readable (but not writable) by a user process: user/supervisor mode bit + page table pointer + TLB
3. From user mode to supervisor mode (and vice versa):
   - *System call*: special instruction that transfers control from user mode to a dedicated location in supervisor code space, invoking the exception mechanism in the process

Virtual memory

Implementing protection

Context

Sharing a single main memory by multiple processes

Three basic capabilities:

1. *Supervisor mode* (or *kernel mode*): mode indicating that a running process is an operating system process.
2. Processor state readable (but not writable) by a user process: user/supervisor mode bit + page table pointer + TLB
3. From user mode to supervisor mode (and vice versa):
   - *System call*: special instruction that transfers control from user mode to a dedicated location in supervisor code space, invoking the exception mechanism in the process
   - *Supervisor exception return*: resets to user mode

Virtual memory

Implementing protection

Context

Sharing a single main memory by multiple users

Virtual memory

Implementing protection

Context

Sharing a single main memory by multiple users

Preventing reading and writing by another (user) process

- Each process has its own virtual space

Virtual memory

Implementing protection

Context

Sharing a single main memory by multiple users

Preventing reading and writing by another (user) process

- Each process has its own virtual space
- The operating system keeps the page tables

Virtual memory

Implementing protection

Context

Sharing a single main memory by multiple users

Preventing reading and writing by another (user) process

- Each process has its own virtual space
- The operating system keeps the page tables
- The user process cannot change its own page table

Virtual memory

Implementing protection

Context

Sharing a single main memory by multiple users

Preventing reading and writing by another (user) process

- Each process has its own virtual space
- The operating system keeps the page tables
- The user process cannot change its own page table
- All page tables placed in a protected address space

Virtual memory

Context switch

Definition

Context switch: changing of the internal state of the processor to allow a different process to use the processor that includes saving the state needed to return to the currently executing process

Virtual memory

Context switch

Definition

Context switch: changing of the internal state of the processor to allow a different process to use the processor that includes saving the state needed to return to the currently executing process

Overhead

- clear TLB entries
- Inefficient when high process switch rate

Virtual memory

Context switch

Definition

Context switch: changing of the internal state of the processor to allow a different process to use the processor that includes saving the state needed to return to the currently executing process

Overhead

- clear TLB entries
- Inefficient when high process switch rate

Process identifier

- Concatenated to the tag
- TLB hit when
  - Page number + process identifier match

Virtual memory

Memory hierarchy design challenges

Design change	Effect on miss rate	Possible negative performance effect
Increases cache size	Decreases capacity misses	May increase access time
Increases associativity	Decreases miss rate due to conflict misses	May increase access time
Increases block size	Decreases miss rate for a wide range of block sizes due to spatial locality	Increases miss penalty. Very large block could increase miss rate

Source: [10]

Virtual memory

Summary

Virtual memory

- Manage caching between the main memory and secondary memory
- Virtual address (beyond physical address)
- Share main memory between several processes, users with protections

Virtual memory

Summary

Virtual memory

- Manage caching between the main memory and secondary memory
- Virtual address (beyond physical address)
- Share main memory between several processes, users with protections

High cost of page fault

Miss rate reduced by

- Large page: spatial locality $\searrow$ miss rate
- Fully associative mapping between virtual and physical addresses
- LRU replacement technique (OS)
- Write-back scheme
- TLB: cache for translations

Outline

Virtual machines

Virtual Machines (VM)

- First developed in the mid-1960s
- Recent gained popularity
  - Isolation and security in modern systems
  - Failures in security and reliability of standard OS
  - Sharing single computer among unrelated users (cloud computing)
  - Increase in raw speed of processors: overhead of VM acceptable

Virtual machines

Virtual Machines (VM)

- First developed in the mid-1960s
- Recent gained popularity
  - Isolation and security in modern systems
  - Failures in security and reliability of standard OS
  - Sharing single computer among unrelated users (cloud computing)
  - Increase in raw speed of processors: overhead of VM acceptable

Definition

Broad definition of VM: all emulation methods that provide and standard software interface. E.g. JVM (Java Virtual Machine)

Virtual machines

System Virtual Machine

(Operating) System Virtual Machine

- Functionalities to emulate of full operating system
- IBM VM/370, VirtualBox, VMware, etc.
- *Virtual Machine Monitor (VMM)* or *Hypervisor*
  - *Host*: the underlying hardware platform
  - *Guest*: the virtualized system

Virtual machines

System Virtual Machine

(Operating) System Virtual Machine

- Functionalities to emulate of full operating system
- IBM VM/370, VirtualBox, VMware, etc.
- *Virtual Machine Monitor (VMM)* or *Hypervisor*
  - *Host*: the underlying hardware platform
  - *Guest*: the virtualized system

Two main benefits

1. *Managing software*: abstraction of the complete software stack (legacy OSes, current OSes, testing next OSes)
2. *Managing Hardware*: decoupling guest and host
   - servers on separate computers: migration of a running VM to a different computer

Virtual machines

Cost of virtualisation

Depend on the workload:

- User-level processor-bound: no overhead (no OS invocation)

Virtual machines

Cost of virtualisation

Depend on the workload:

- User-level processor-bound: no overhead (no OS invocation)
- I/O intensive workloads: high overhead

Virtual machines

Cost of virtualisation

Depend on the workload:

- User-level processor-bound: no overhead (no OS invocation)
- I/O intensive workloads: high overhead
  - Except if also *I/O bound*

Virtual machines

Cost of virtualisation

Depend on the workload:

- User-level processor-bound: no overhead (no OS invocation)
- I/O intensive workloads: high overhead
  - Except if also *I/O bound*

Virtual machines

Cost of virtualisation

Depend on the workload:

- User-level processor-bound: no overhead (no OS invocation)
- I/O intensive workloads: high overhead
  - Except if also *I/O bound*

Instructions to emulate:

- Number of instructions and time it takes
- Same ISA between host and guest: native instructions

Virtual machines

Requirements of a VMM

The VMM must

- isolate the state of guests from each other
- protect itself from guest software (including OS)

Virtual machines

Requirements of a VMM

The VMM must

- isolate the state of guests from each other
- protect itself from guest software (including OS)

Requirements

- Guest software should behave exactly as if it were on the native hardware
- Guest software should not be able to change the allocation of real system resources directly

Virtual machines

VMM must control everything

- Access to privilege state, I/O, exceptions, interrupts
- Higher privilege level than the guest VM (runs in user mode)
- System requirements:
  - Two processor modes: system and user
  - Subset of instructions available only in system mode to control all system resources

Virtual machines

Lack of instruction set architecture support for VM

- *Virtualizable*: architecture that allows the VM to execute directly on the hardware
  - IBM 370, RISC-V
- No virtualization: x86, ARMv7, MIPS

Protection and instruction set architecture

- Unexpected side effects when running privileged instructions in user mode on x86 architecture
- Three steps to improve performance
  1. Reduce the cost of processor virtualization
  2. Reduce the interrupt overhead
  3. Reduce interrupt cost by steering interrupts to the proper VM without invoking VMM

Virtual machines

RISC-V

RISC-V traps all privileged instructions when running in user mode, supporting *classical virtualization*.

Part III

The future

Outline of Part III

Outline

Still ever increasing technology achievements

Three technologies as the leading contenders [7](2014):

- STT-RAM (spin-transfer torque RAM)[Mos05]
- PCM (phase-change memory) [Rao08,Lee09]
- Memristor [Stru08]

Still ever increasing technology achievements

Embedded COMPUTING DESIGN

Arm Debuts eMRAM IoT test chip with Samsung, Cadence

BRANDON LEWIS MAY 15, 2019

SAMSUNG FOUNDRY FORUM. Arm, Samsung Foundry, Cadence, and Sondrel have collaborated on the Musca-S1, a 28 nm fully-depleted silicon-on-insulator (FD-SOI) embedded MagnetoResistive RAM (eMRAM) test chip based on Arm Cortex-M33 IP. The Musca-S1 test chip and an accompanying development board enable IoT SoC designers to evaluate eMRAM technology, which can easily scale below 40 nm to support a broad range of memory and power requirements.

Source: arm.com

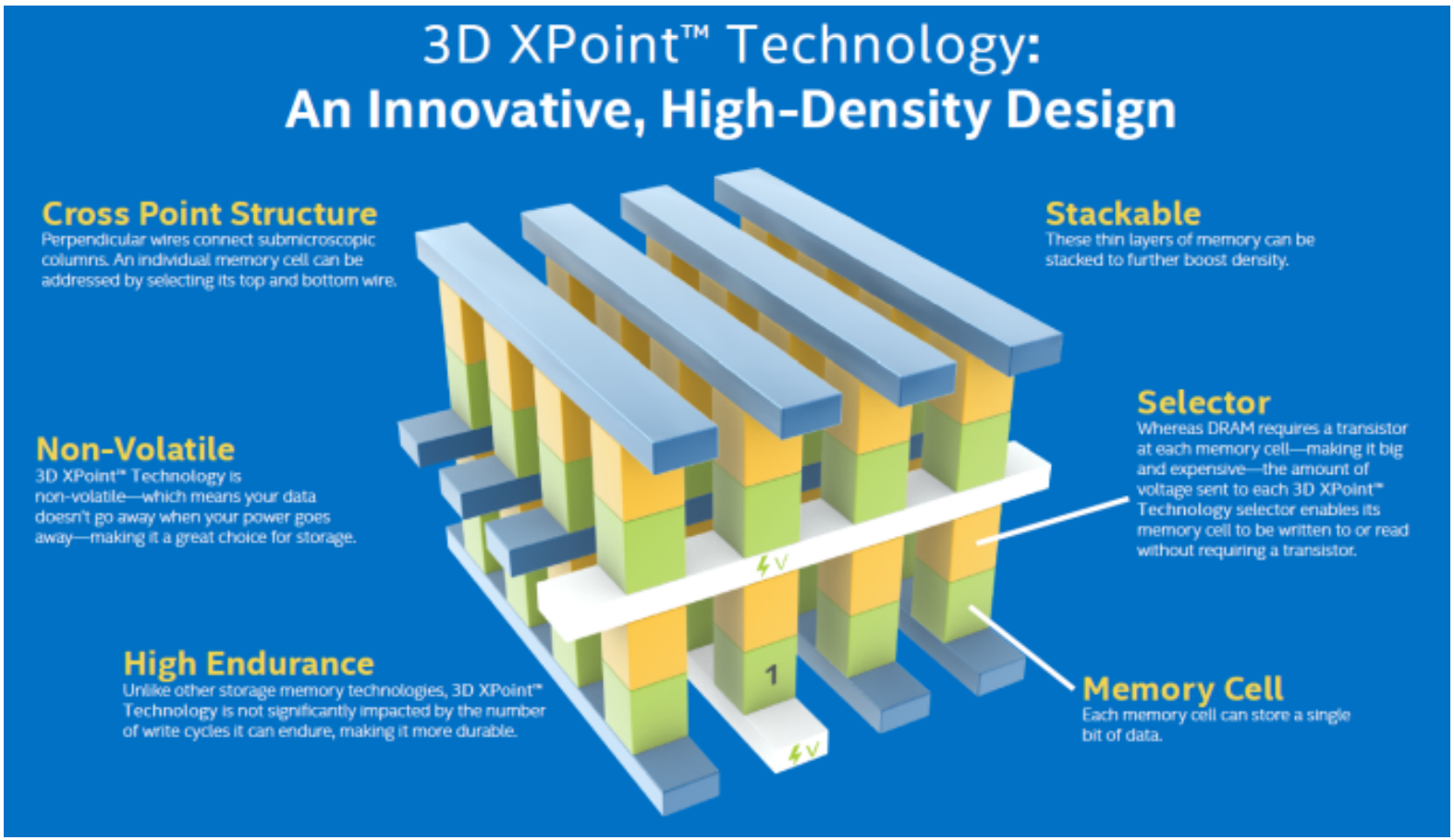
Still ever increasing technology achievements

3D XPoint

4th August 2015

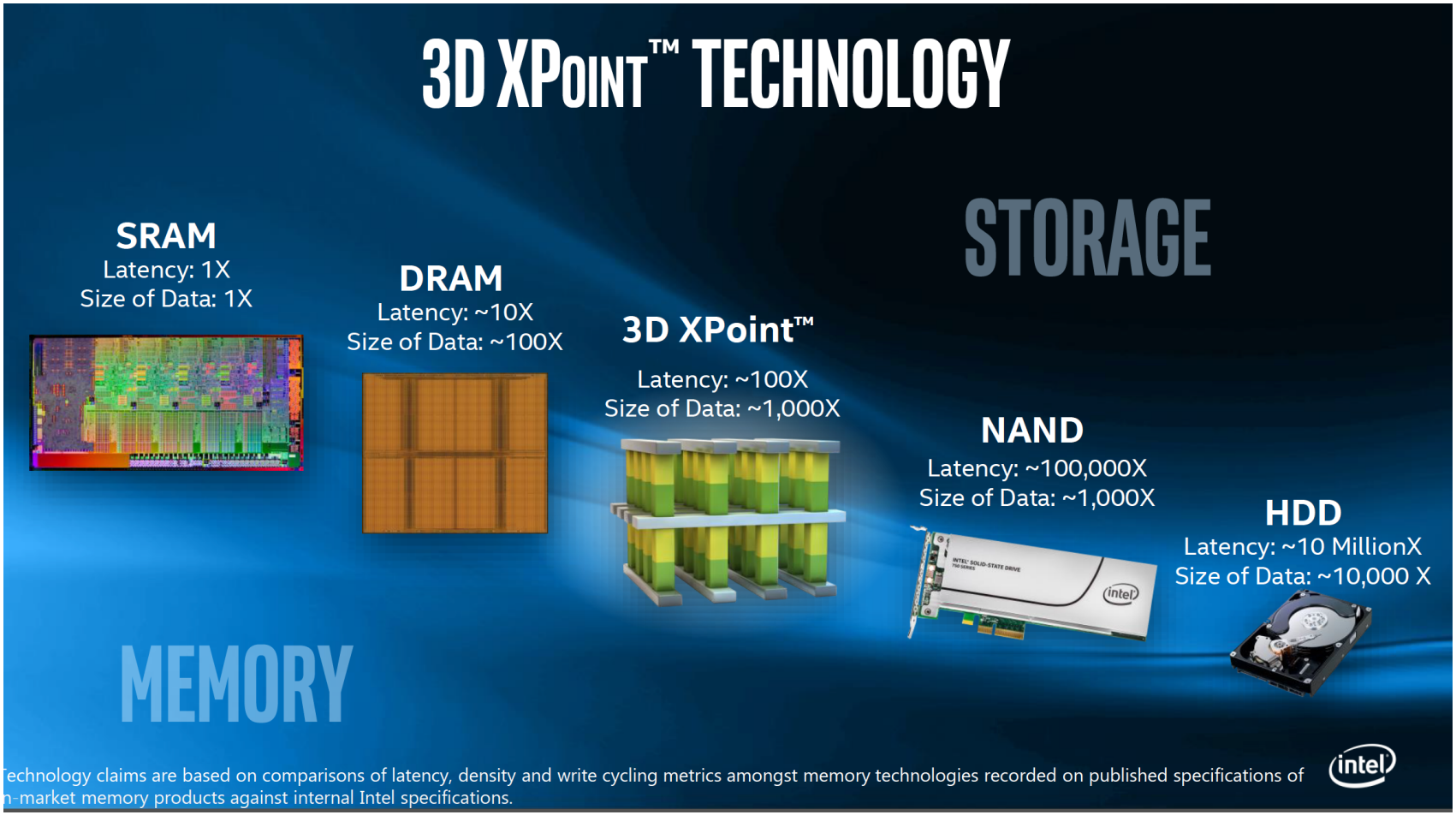
New memory technology is 1,000 times faster

Intel and Micron have unveiled "3D XPoint" – a new memory technology that is 1,000 times faster than NAND and 10 times denser than conventional DRAM.



Still ever increasing technology achievements

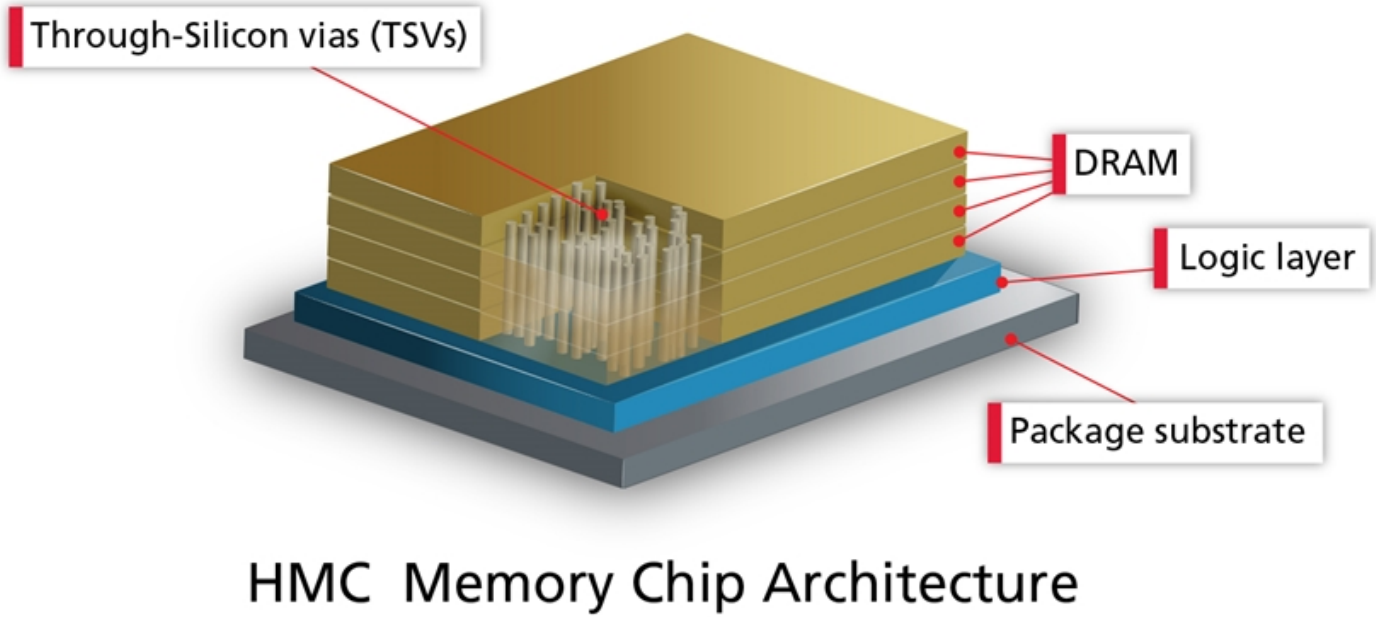
3D XPoint



Commercial product: Optane SSD PC P4800X

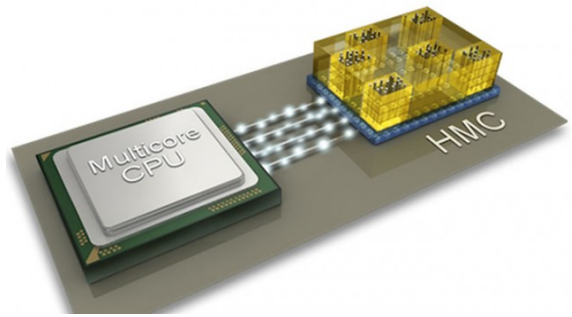
Still ever increasing technology achievements

Hybrid Memory Cube



HMC Memory Chip Architecture

https://community.cadence.com/cadence_blogs_8/b/ip/posts/what-s-new-with-hybrid-memory-cube-hmc



Still ever increasing technology achievements

Still...

- more than 80% of the chip area is dedicated to caches, memories, memory controllers, interconnects and so on, whose sole purpose is to buffer data or control the buffering of data [1]
- $\Rightarrow$ workarounds which are making systems ever more complex
- more than 62% of the entire measured system energy is spent on moving data between memory and the computation units [1]

Still ever increasing technology achievements

Still. . .

- more than 80% of the chip area is dedicated to caches, memories, memory controllers, interconnects and so on, whose sole purpose is to buffer data or control the buffering of data [1]
- $\Rightarrow$ workarounds which are making systems ever more complex
- more than 62% of the entire measured system energy is spent on moving data between memory and the computation units [1]

Enabling the continued performance scaling of smaller systems requires significant research breakthroughs in three key areas [6]

1. power efficiency
2. programmability
3. execution granularity

Down with Hierarchy!

Outline

Processing close to memory

Stop or reduce moving data

Process the data where it is: in the memory!

Processing close to memory

Stop or reduce moving data

Process the data where it is: in the memory!

Sort this out!

Computing-In-Memory, Processing In Memory, In-memory computing, Logic In Memory, Near-Memory Computing, Intelligent Memory, Smart memories, Near-memory processing, Active memory, Memory-driven computing

Processing close to memory

Back to the future!

Processing close to memory

Back to the future!

Hitting the Memory Wall: Implications of the Obvious

Wm. A. Wulf
Sally A. McKee

Department of Computer Science
University of Virginia
{wulf | mckee}@virginia.edu

December 1994

Processing close to memory

Back to the future!

Hitting the Memory Wall: Implications of the Obvious

Wm. A. Wulf
Sally A. McKee

Department of Computer Science
University of Virginia
{wulf | mckee}@virginia.edu

December 1994

Processing in Memory: The Terasys Massively Parallel PIM Array

Maya Gokhale
David Sarnoff Research Center*

Bill Holmes and Ken Iobst
Supercomputing Research Center

**The work reported here was done while the author was at the Supercomputing Research Center, Bowie, Maryland.*

SIMD processor arrays provide superior performance on fine-grained massively parallel problems in which all parallel threads do the same operations most of the time. However, this fine-grained synchrony limits the application space of SIMD (single instruction, multiple data) machines. If there are many alternative data-dependent actions among the parallel threads, the total execution time is the *sum* of the alternatives rather than the maximum single-thread execution time. Additionally, if the application is not inherently load-balanced, performance can degrade seriously: Most of the processors

Processing close to memory

Where to compute then?

In DRAM

- Processing in memory: inside DRAM (UpMem)
- In-memory computing primarily relies on keeping data in a server's RAM as a means of processing at faster speeds

Source: https://www.upmem.com/

Processing close to memory

Where to compute then?

In SRAM

- X-SRAM: Enabling In-Memory Boolean Computations in CMOS Static Random Access Memories [4] (2018)
  - 75% of memory accesses can be saved
- XNOR-SRAM: In-Memory Computing SRAM Macro for Binary/Ternary Deep Neural Networks [9] (2018)
  - 33X better energy and 300X better energy-delay product

Processing close to memory

Where to compute then?

In SRAM

- X-SRAM: Enabling In-Memory Boolean Computations in CMOS Static Random Access Memories [4] (2018)
  - 75% of memory accesses can be saved
- XNOR-SRAM: In-Memory Computing SRAM Macro for Binary/Ternary Deep Neural Networks [9] (2018)
  - 33X better energy and 300X better energy-delay product

In cache

- Compute cache [3] (2017)
  - performance by 1.9× and reduce energy by 2.4×
  - 54× throughput, 9× dynamic energy savings

Memory-Driven Computing

THE machine

HP bets it all on The Machine, a new computer architecture based on memristors and silicon photonics

By Sebastian Anthony on June 11, 2014 at 11:30 am | 39 Comments

Memory-Driven Computing

THE machine

HP bets it all on The Machine, a new computer architecture based on memristors and silicon photonics

By Sebastian Anthony on June 11, 2014 at 11:30 am | 39 Comments

What happened to the HP machine?

Anyone remember the HP announcement about 'the machine'? Billy MacInnes does and he wonders just what has happened to the grand project

Memory-Driven Computing

THE machine

HP bets it all on The Machine, a new computer architecture based on memristors and silicon photonics

By Sebastian Anthony on June 11, 2014 at 11:30 am | 39 Comments

What happened to the HP machine?

Anyone remember the HP announcement about 'the machine'? Billy MacInnes does and he wonders just what has happened to the grand project

HP Enterprise unveils The Machine, a single-memory computer capable of addressing 160 terabytes

DEAN TAKAHASHI @DEANTAK MAY 16, 2017 6:01 AM

Memory-Driven Computing

THE machine

HP bets it all on The Machine, a new computer architecture based on memristors and silicon photonics

By Sebastian Anthony on June 11, 2014 at 11:30 am | 39 Comments

What happened to the HP machine?

Anyone remember the HP announcement about 'the machine'? Billy MacInnes does and he wonders just what has happened to the grand project

HP Enterprise unveils The Machine, a single-memory computer capable of addressing 160 terabytes

DEAN TAKAHASHI @DEANTAK MAY 16, 2017 6:01 AM

WHAT'S NEW

HPE Persistent Memory, available in 128, 256, and 512 GB kits, features Intel® Optane™ DC Persistent Memory to approach the speed of traditional DRAM with the persistence of storage, ensuring high capacity, high performance, and ongoing data safety — even in the event of an interruption in power due to an unexpected power loss, system crash, or normal system shutdown.

Processing close to memory

Active Research

Displaying results **1-25** of **119** for **(("Document Title":in-memory) AND "Document Title":processing)**
Filters Applied: Conferences Journals & Magazines

Displaying results **1-25** of **136** for **(("Document Title":in-memory) AND "Document Title":computing)**
Filters Applied: Conferences Journals & Magazines

Year	In-memory AND processing	In-memory AND computing
2019	10	8
2018	30	51
2017	31	32
2016	17	18
2015	9	16
2014	3	4
2013	1	4
2012	0	0
2011	0	0
1995-2010	18	3

What about programmability?

What about execution granularity?

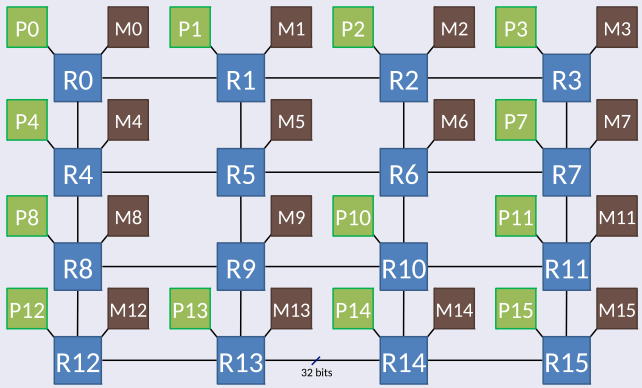
What memory is needed for?

- storing data
- storing instructions
- saving temporary values
- synchronizing processes/threads

Processing close to memory

Notifying memories

Network on Chip



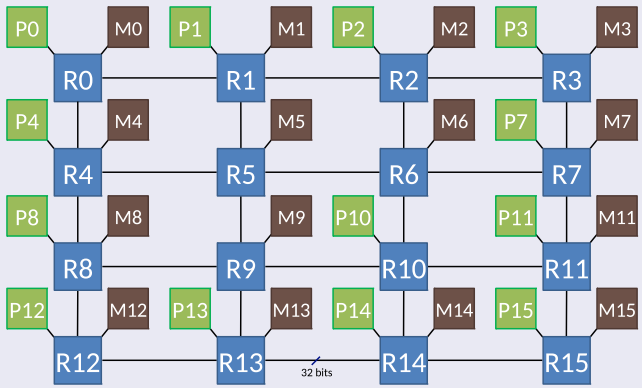
Network on Chip

- × Long latency
- × Sometimes useless for data-flow
- × High Energy consumption (up to 40%)

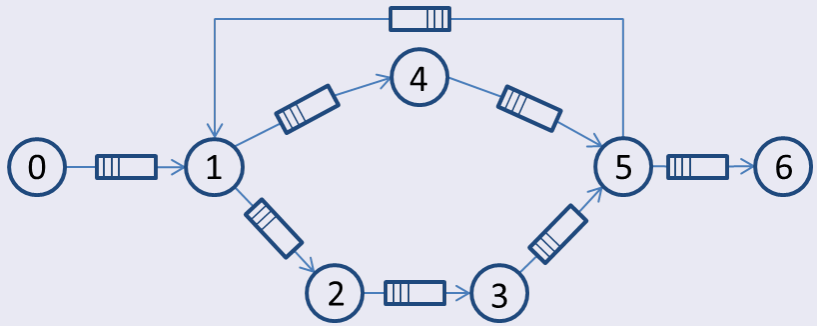
Processing close to memory

Notifying memories

Network on Chip



Data-flow application



Network on Chip

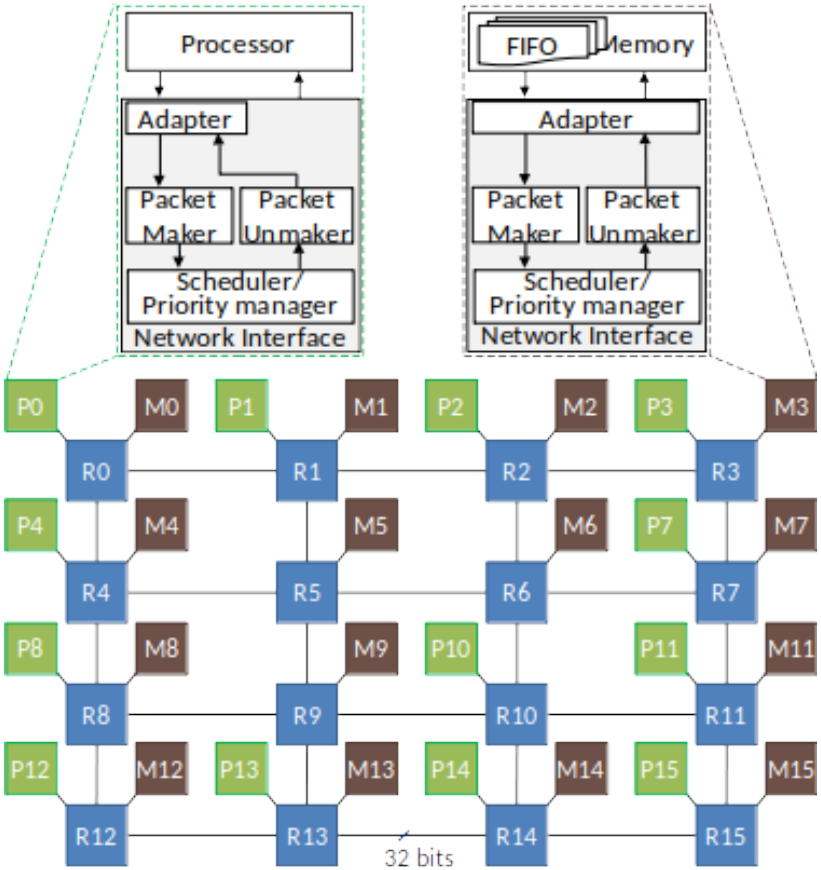
- × Long latency
- × Sometimes useless for data-flow
- × High Energy consumption (up to 40%)

Memory request

- × processor initiates transactions
- × the memory replies
- × several times the same data

Notifying memories

Network on Chip

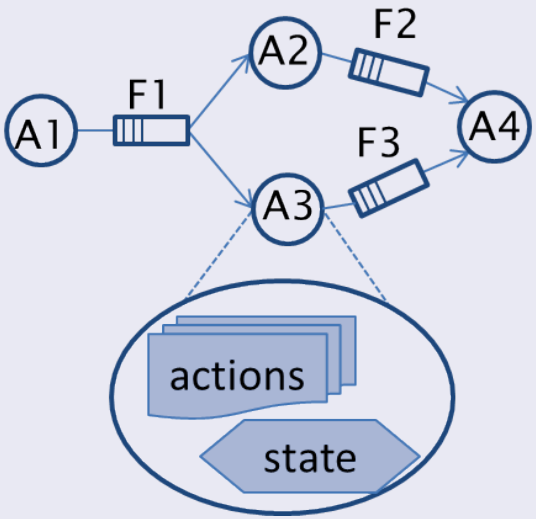


- Interconnection network
  - Routers
  - Network interface
- High bandwidth
- Long latency
- High energy consumption

Notifying memories

Dynamic Dataflow

Network of actors



Dataflow

- Formal Model Of Computation
- Explicit spatial and temporal parallelism
- Static or dynamic actors
  - Execute actions (“fire” actions)
- Firing rule
  - Enough tokens in input FIFOs
  - Enough space in output FIFOs

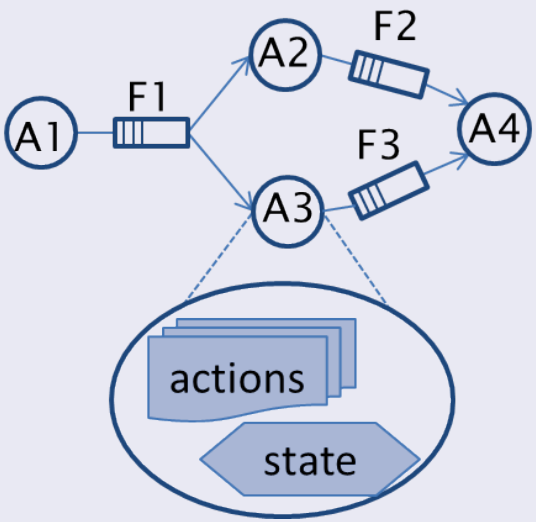
Static actors

- Fixed number of consumed and produced tokens
- Can be solved at compile time

Notifying memories

Dynamic Dataflow

Network of actors



Dataflow

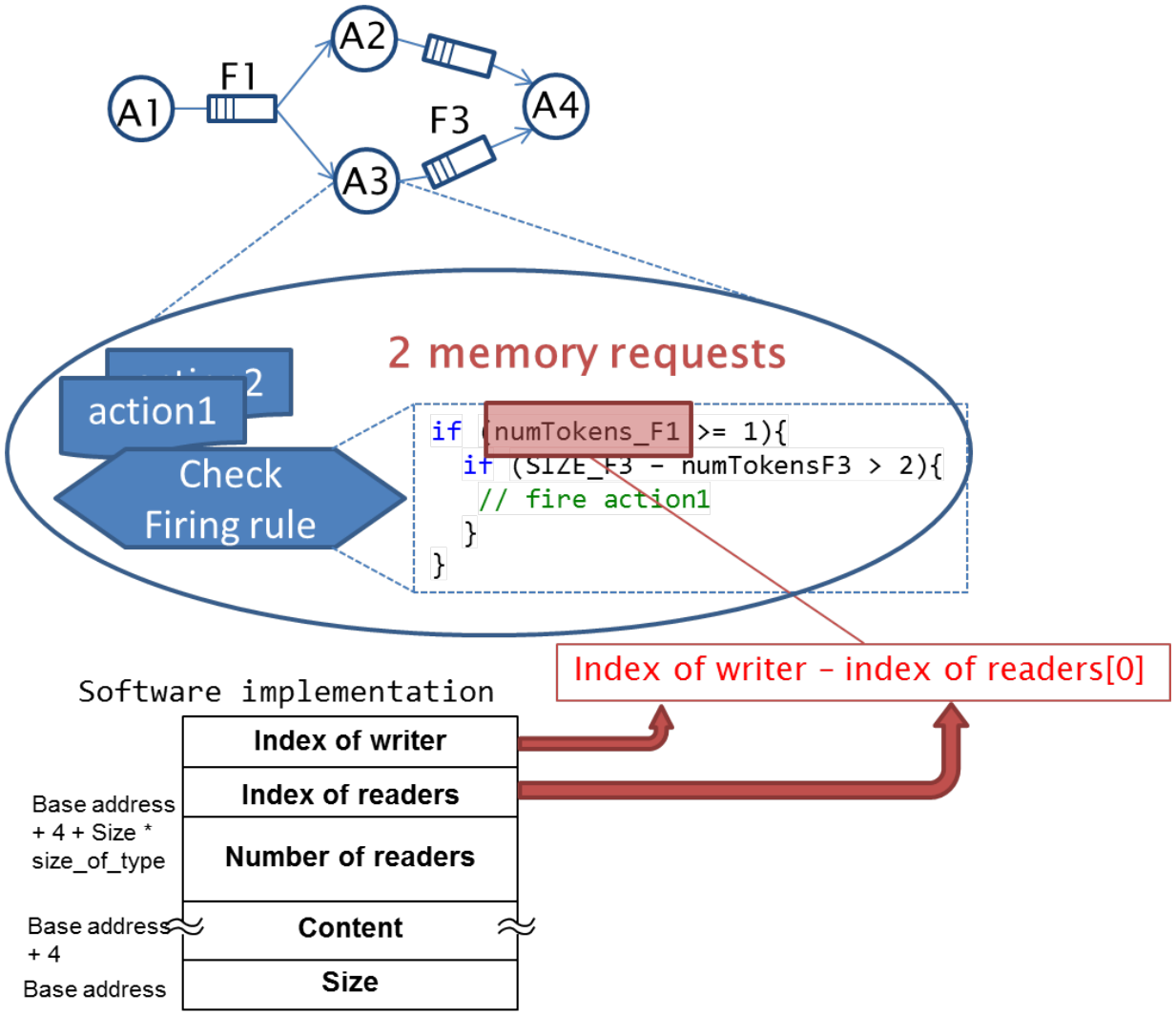
- Formal Model Of Computation
- Explicit spatial and temporal parallelism
- Static or dynamic actors
  - Execute actions (“fire” actions)
- Firing rule
  - Enough tokens in input FIFOs
  - Enough space in output FIFOs

Dynamic actors

- Variable number of consumed /produced tokens
- Must be solved at run time

Notifying memories

Dynamic Dataflow



Execution model

- 2 memory requests per FIFO
- If no action fired, the same requests are made again and again
- NoC Latency = Huge penalty for Polling

Notifying memories

Motivational example

Unsuccessful scheduling by the MPEG4-SP decoder for different video sequences and formats

Video		Useless attempt	Empty input FIFO	Full output FIFO
Sequence	Format			
Akiyo	CIF	42.7%	63.7%	36.3%
Parkjoy	720p	21.3%	90.8%	9.2%
Foreman	CIF	34.8%	90.7%	9.3%
Coastguard	CIF	27.8%	98.4%	1.6%
Stefan	CIF	25.9%	83.3%	16.7%
Bridge far	QCIF	23.8%	38.4%	61.6%
Ice	4CIF	45.6%	70.4%	29.6%

Useless Memory Accesses through + long NoC latency Penalty

Notifying memories

Motivational example

Unsuccessful scheduling by the MPEG4-SP decoder for different video sequences and formats

Video		Useless attempt	Empty input FIFO	Full output FIFO
Sequence	Format			
Akiyo	CIF	42.7%	63.7%	36.3%
Parkjoy	720p	21.3%	90.8%	9.2%
Foreman	CIF	34.8%	90.7%	9.3%
Coastguard	CIF	27.8%	98.4%	1.6%
Stefan	CIF	25.9%	83.3%	16.7%
Bridge far	QCIF	23.8%	38.4%	61.6%
Ice	4CIF	45.6%	70.4%	29.6%

Useless Memory Accesses through + long NoC latency Penalty

Monitor FIFOs and emit notifications about their status

Notifying memories

Related work

- Active memory processor [Yoo 2012]
- Smart memory [Mai 2000]: Modular reconfigurable architecture
- Processing In Memory (PIM) [Gokhale 1995]: Offload computation in the memory
- Logic In Memory [Gaillardon 2016]: Fine grained, Technology dependent
- Intelligent Memory [Kozyrakis 1997]
- Near Memory Computing [NeMeCo]

In all cases

- Memory are slaves
- No notification feature

Notifying memories

Observer design pattern (software engineering)

- Subject: sends the notifications
- Observer: reacts to notifications

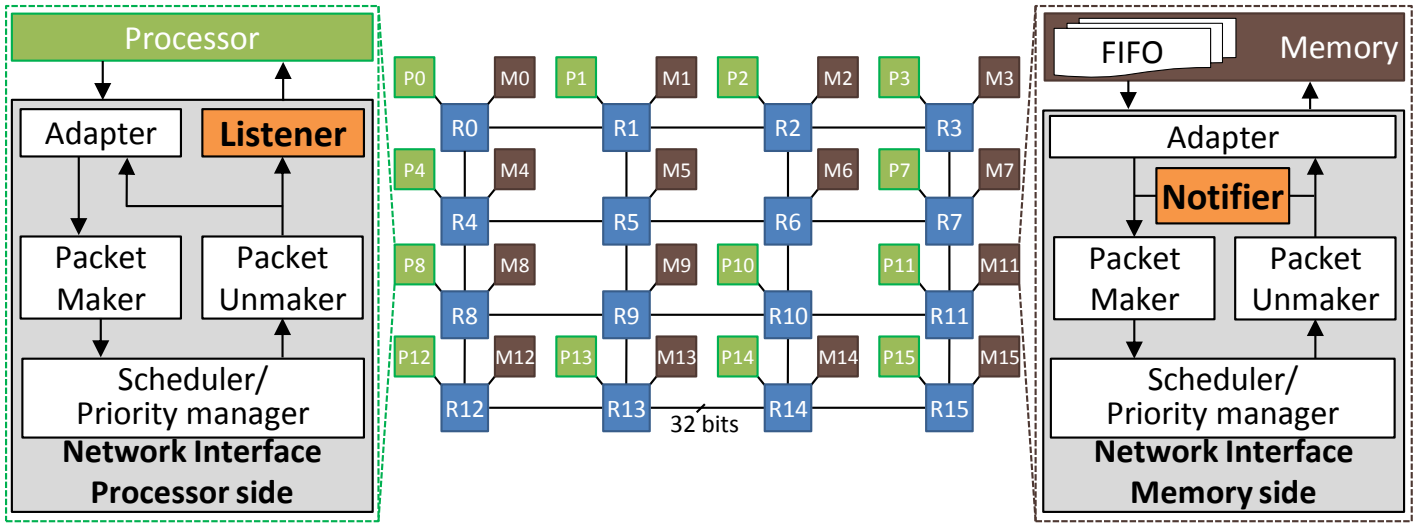
Implementation in the Network Interface (NI)

- Master component that can send packets through the network
- Component that can monitor requests
- Independent from processor, memory, NoC parameters
- The subject is the memory (becomes master)
- The listener is the processor (becomes slave)

Notifying memories

Listener and notifier: new components of Network Interface

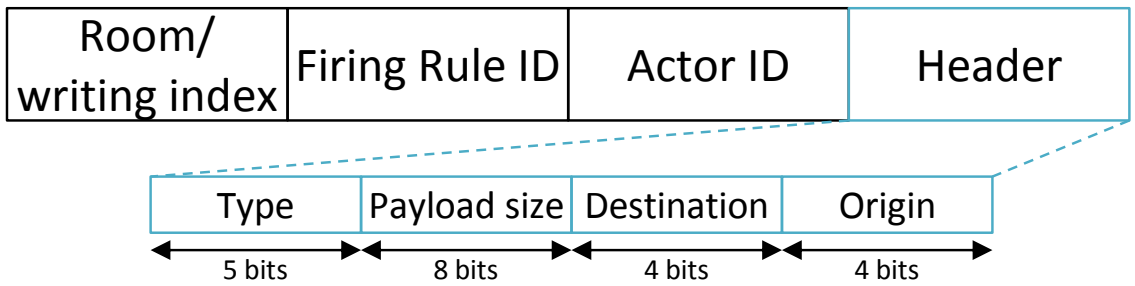
- Notifier on memory side
- Listener on processor side



Notifying memories

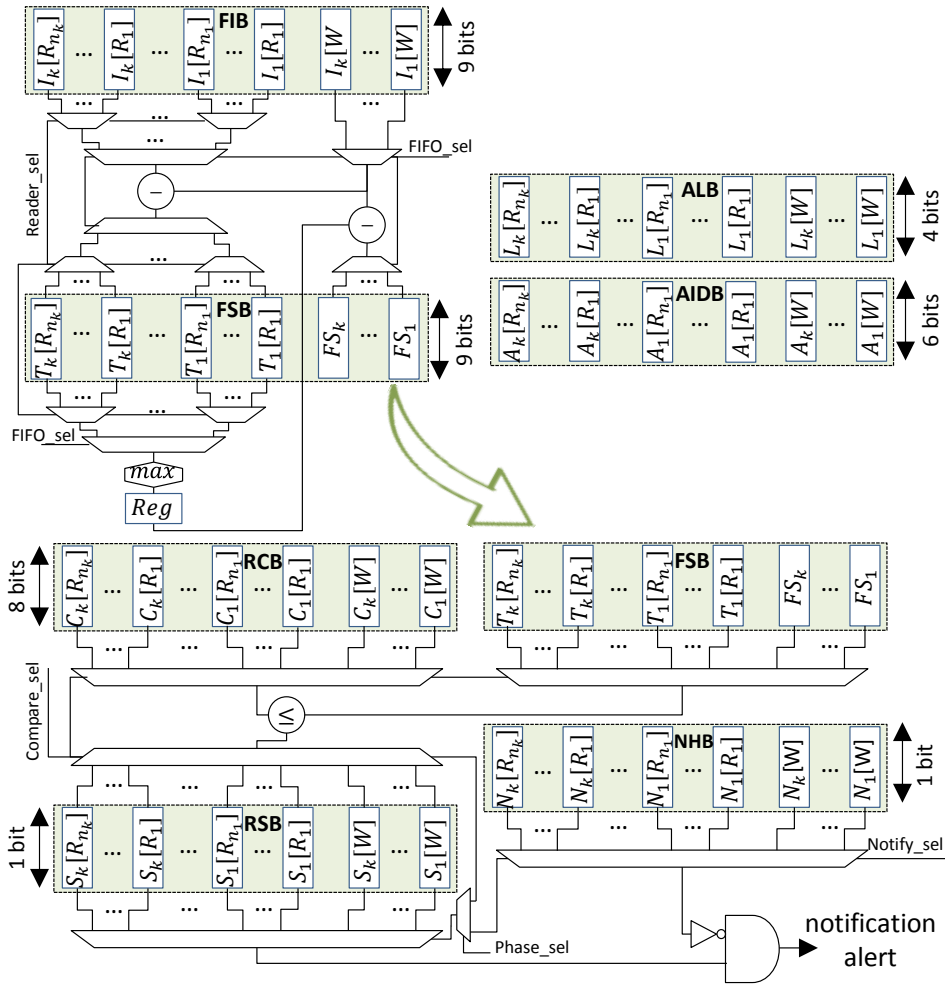
The notifier

1. Configuration phase
   - Specify what FIFOs to monitor
2. Checking phase
   - “Packet sniffer”: retrieves indexes of writers and readers Computes the number of tokens in a FIFO
3. Notification phase
   - Provides the packet maker with the target location, identity number, satisfied firing rule identity number, and the number of available tokens or free space



Notifying memories

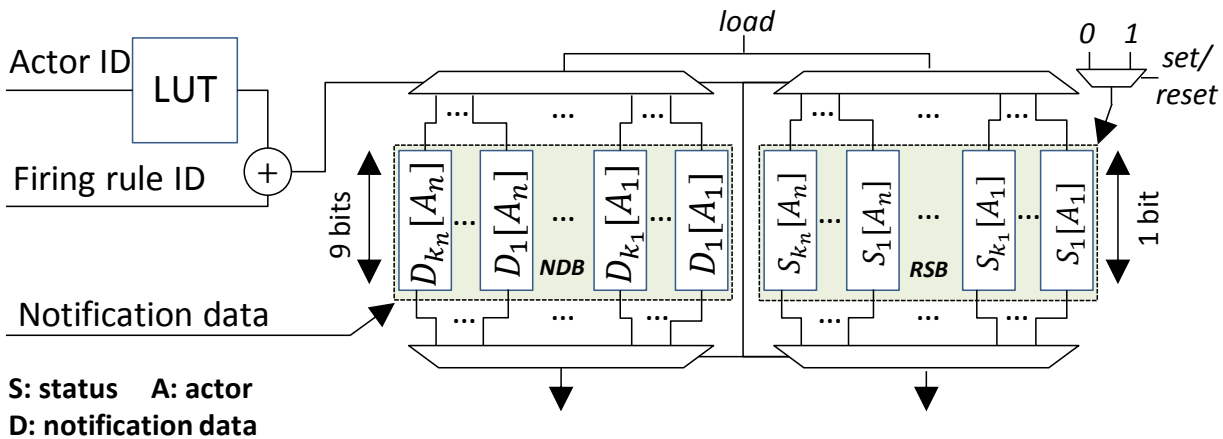
The notifier



Notifying memories

The listener

1. Configuration phase
   - Specifies what notification to listen to
2. Execution phase
   - Sets the firing rule validity bit when a notification is caught
   - Clears the validity bit when the action is performed



Notifying memories

Experimental Setup

NoC

- 4x4 mesh-based SystemC cycle-accurate model
- 13 processors, 12 memories
- Wormhole packet switching, XY routing algorithm
- Routers: one arbiter per port, one buffer per input port
- Round robin

Application

- MPEG4-SP (H264) decoder
- 41 actors, 70 FIFOs

Mapping

- Manual mapping, minimize number of hops
- FIFOs equally distributed in memories

Notifying memories

Results

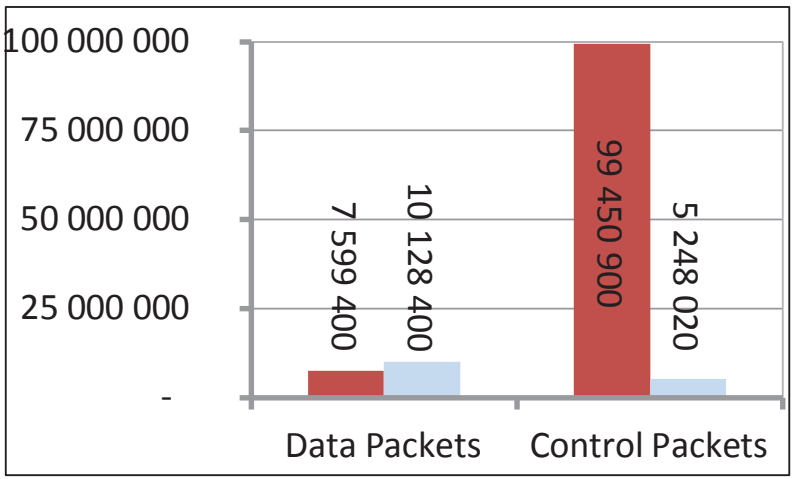
Results of decoding 10 frames of ice video sequence in 4CIF format

Parameter	Notifying memory	Ordinary memory	gain
Latency (μs)	143.42	665.06	-78.44%
Throughput (frames/s)	27.53	23.29	+15.41%
Injection rate(flits/s)	60 167 732	121 635 294	-50.53%
Switch conflicts	71 182 509	288 574 519	-75.33%
Transported flits	109 264 000	261 123 000	-58.16%
Transported packets	15 376 400	107 050 000	-85.64%

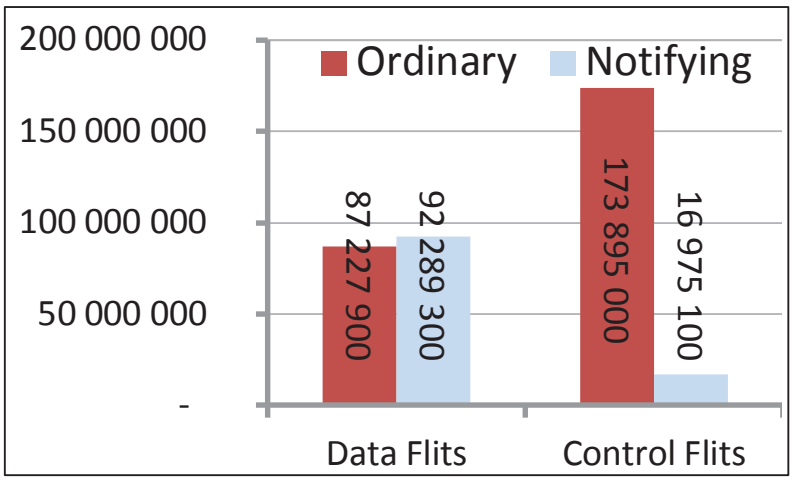
Notifying memories

Results

- Data packets: tokens
- Control packets: mapping information, memory requests, notifications



(a)

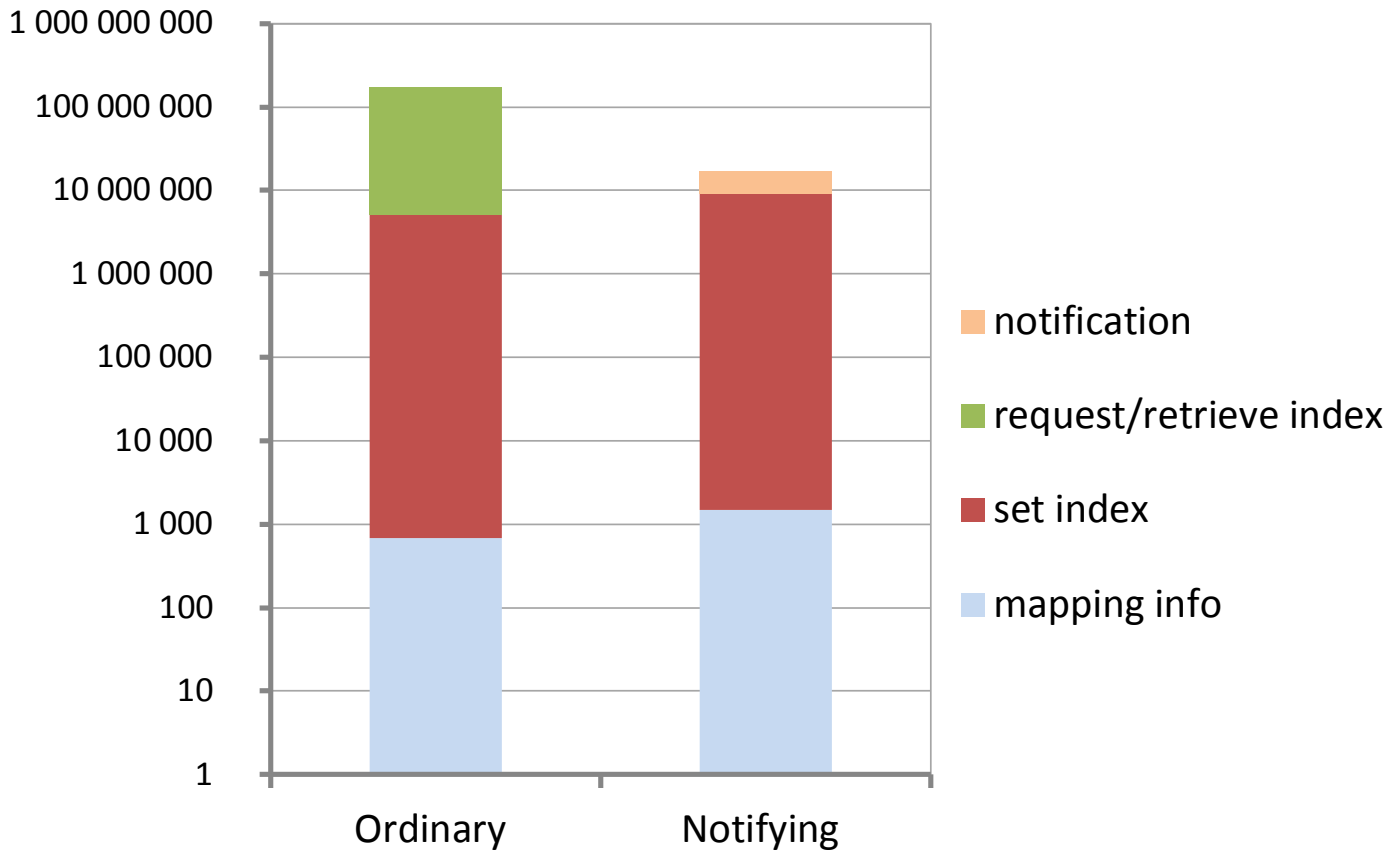


(b)

Notifying memories

Results

- Control flits classification



Notifying memories

Results

Notification memory gain for decoding 10 frames of different video sequences

Video		Throughput	Latency	Injection rate	Switch conflicts	Flits number
Sequence	**Format**					
Bridgefar	QCIF	+15.53%	-73,96%	-45,80%	-71,38%	-54,22%
bus	CIF	+2.84%	-73,79%	-53,40%	-72,90%	-54,73%
grandma	QCIF	+16.79%	-68,96%	-60,78%	-85,50%	-67,36%
foreman	CIF	+14.26%	-78,41%	-46,81%	-72,86%	-54,39%
ice	4CIF	+15.41%	-78,44%	-50,53%	-75,33%	-58,16%

Notifying memories

Preliminary synthesis results

- Worst-case implementation
  - Same notifier in all memories: able to deal with the 70 FIFOs
  - Same listener in all processors: able to deal with the 41 actors
- Cadence Encounter RTL Compiler 65nm (500MHz, 25 deg C)
- Leakage and dynamic power
- NoC adopting notifying memories saves 49.1% of energy
- Power overhead of notifying NoC is 16.3%
- Area overhead of notifying NoC is 12.4%

Notifying memories

Wrap-up

- Notifying memories concept
  - The memories send notifications to processors
  - Notifiers on memory side
  - Listeners on processor side
- SystemC model
  - Notifiers and listeners in the Network Interface of the NoC
  - New kind of packet : the notification packet
- Simulation results
  - Latency (-78%), injection rate (-60%)
- Synthesis results
  - Worst-case implementation
  - +12% area
  - -49% energy

Conclusion

Conclusion

Memory system

- Central component of any digital device
- Keep the pace with faster processors
  - Principle of locality
  - Memory hierarchy

Conclusion

Memory system

- Central component of any digital device
- Keep the pace with faster processors
  - Principle of locality
  - Memory hierarchy

Von Neumann architecture

Computer architecture heavily rely on a 70 years old scheme.

Many additional features to get higher performance

Conclusion

Memory system

- Central component of any digital device
- Keep the pace with faster processors
  - Principle of locality
  - Memory hierarchy

Von Neumann architecture

Computer architecture heavily rely on a 70 years old scheme.

Many additional features to get higher performance

The future

- Still rely on technology improvements?

Conclusion

Memory system

- Central component of any digital device
- Keep the pace with faster processors
  - Principle of locality
  - Memory hierarchy

Von Neumann architecture

Computer architecture heavily rely on a 70 years old scheme.

Many additional features to get higher performance

The future

- Still rely on technology improvements?
- Does the memory need to be subject to the processor?

Conclusion

Memory system

- Central component of any digital device
- Keep the pace with faster processors
  - Principle of locality
  - Memory hierarchy

Von Neumann architecture

Computer architecture heavily rely on a 70 years old scheme.

Many additional features to get higher performance

The future

- Still rely on technology improvements?
- Does the memory need to be subject to the processor?
- How to stop useless and (energy) wasteful memory accesses?

Conclusion

Memory system

- Central component of any digital device
- Keep the pace with faster processors
  - Principle of locality
  - Memory hierarchy

Von Neumann architecture

Computer architecture heavily rely on a 70 years old scheme.

Many additional features to get higher performance

The future

- Still rely on technology improvements?
- Does the memory need to be subject to the processor?
- How to stop useless and (energy) wasteful memory accesses?
- Any disruptive scheme to come over?

THE MEMORY REMAINS !

References I

[1] ‘It’s the memory, stupid’: A conversation with Onur Mutlu - Press.

[2] Octet (computing), Feb. 2019.
Page Version ID: 882550034.

[3] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das.
Compute Caches.
In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 481–492, Feb. 2017.

[4] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy.
X-SRAM: Enabling In-Memory Boolean Computations in CMOS Static Random Access Memories.
IEEE Transactions on Circuits and Systems I: Regular Papers, 65(12):4219–4232, Dec. 2018.

[5] H. Carl, V. Zvonko, Z. Safwat, and M. Naraig.
Computer Organization and Embedded Systems.
McGraw-Hill Education, New York, NY, 6 edition edition, Jan. 2011.

[6] B. Dally.
Power, Programmability, and Granularity: The Challenges of ExaScale Computing.
In *2011 IEEE International Parallel Distributed Processing Symposium*, pages 878–878, May 2011.

[7] A. Hasan, F. Paolo, F. Eitan, K. Hironori, L. Danny, L. Phil, M. Arif, M. Dejan, and S. Karsten.
IEEE CS 2022 Report.

[8] B. Jacob, S. Ng, and D. Wang.
Memory Systems: Cache, DRAM, Disk.
Morgan Kaufmann, Burlington, MA, 1 edition edition, Sept. 2007.

[9] Z. Jiang, S. Yin, M. Seok, and J. Seo.
XNOR-SRAM: In-Memory Computing SRAM Macro for Binary/Ternary Deep Neural Networks.
In *2018 IEEE Symposium on VLSI Technology*, pages 173–174, June 2018.

[10] D. A. Patterson and J. L. Hennessy.
Computer Organization and Design RISC-V Edition: The Hardware Software Interface.
Morgan Kaufmann, 2017.

References II

[11] Sidhartha.
Classification of Semiconductor Memories and Computer Memories, July 2015.

[12] G. Taylor, P. Davies, and M. Farmwald.
The TLB Slice—a Low-cost High-speed Address Translation Mechanism.
In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, ISCA '90, pages 355–363, New York, NY, USA, 1990. ACM.
event-place: Seattle, Washington, USA.