Computer Fundamentals
Modern Computer Components

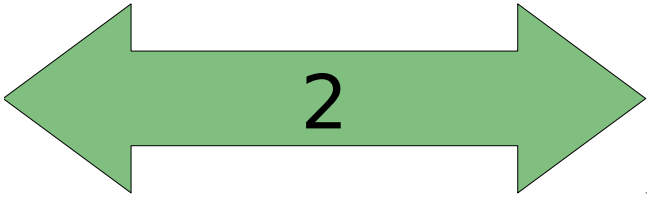
Dr Robert Harle

Today's Topics

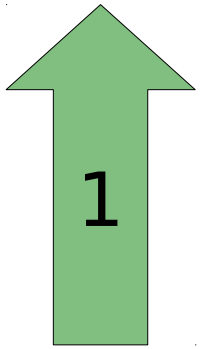
- CPUs in more detail
- Motherboards, buses, peripherals
- Memory hierarchy
- (S)RAM cells
- Spinning HDDs
- Flash and SSDs
- Graphics Cards and GPUs
- RISC and CISC architectures

Our Simple Model So Far

CPU

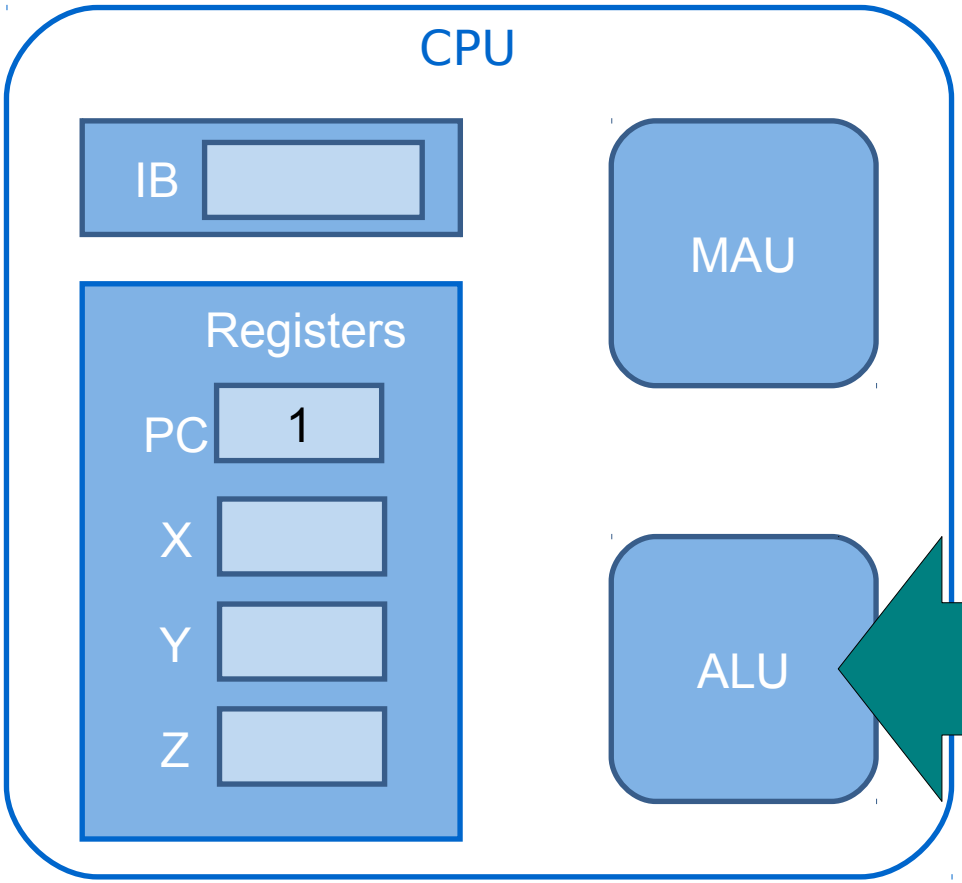


Memory



Program & Data

More on the CPU

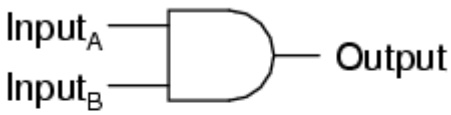


These units are the really clever bits. But how do they work?

Logic

- All these units ultimately do is to provide logic operators
- NOT, AND, NAND, OR, etc.

2-input AND gate

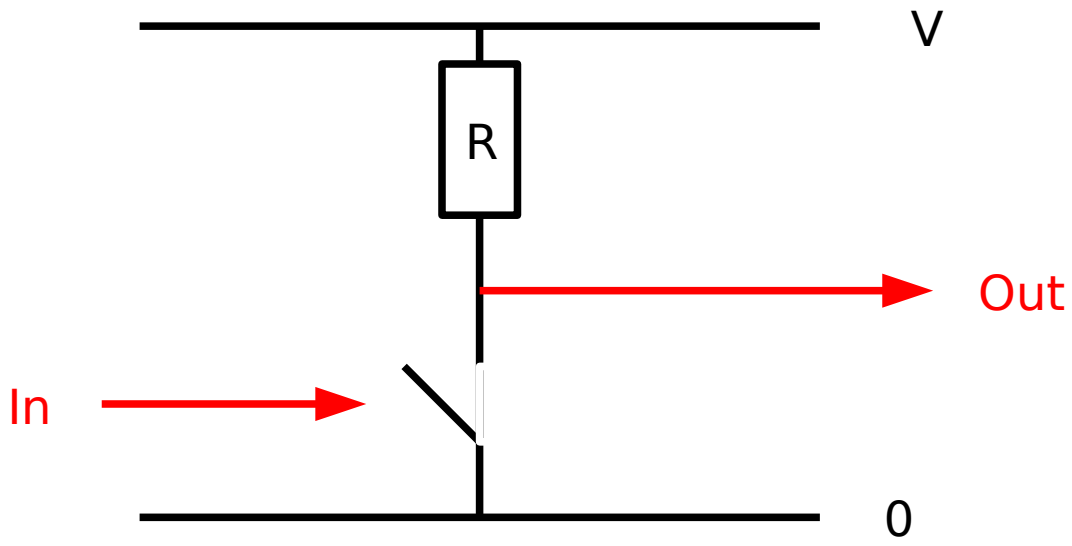


A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Input/outputs are voltages
Notionally logic '1' is some voltage
logic '0' something else

Switches!

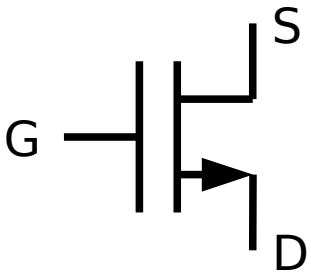
- The trick is to fashion these gates out of switches and basic electronics
- E.g. NOT gate:



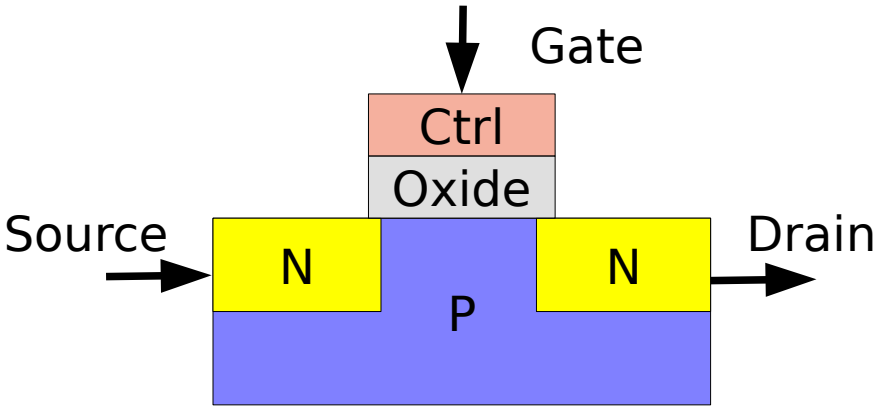
MOSFETs

- Of course, it can't be a push switch
- We use MOSFETs (Metal-Oxide-SemiConductor Field-Effect-**Transistor**)

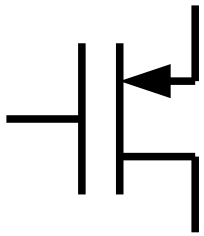
nMOS



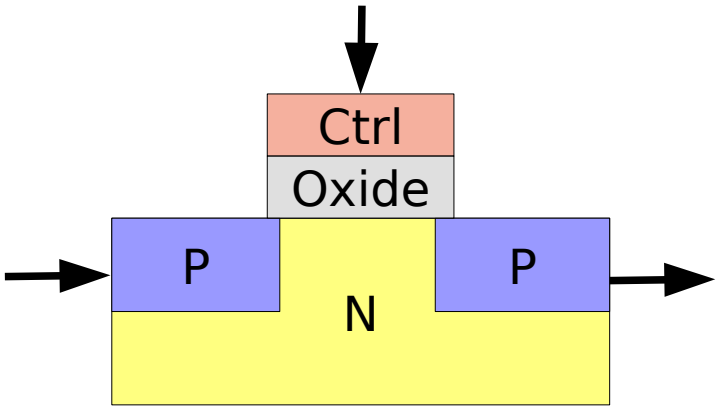
G	Switch
1	Closed
0	Open



pMOS

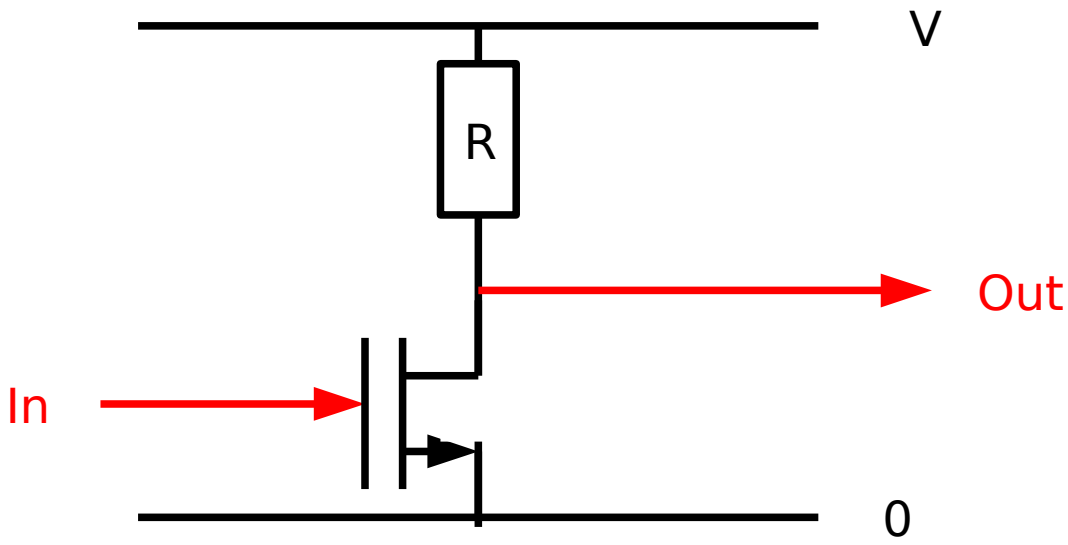


G	Switch
1	Open
0	Closed



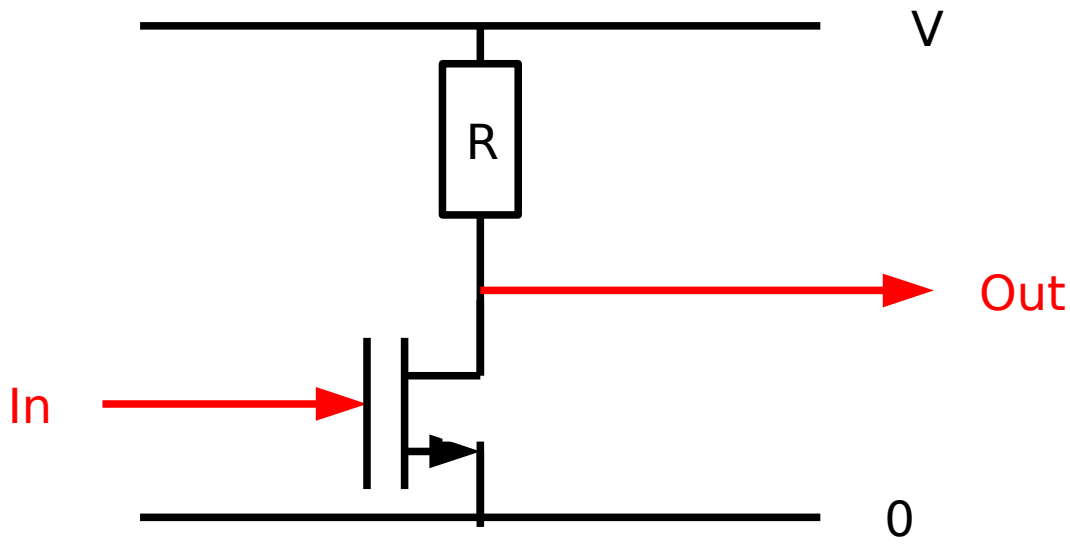
Switches!

- NOT can be done trivially with an nMOS



Switches!

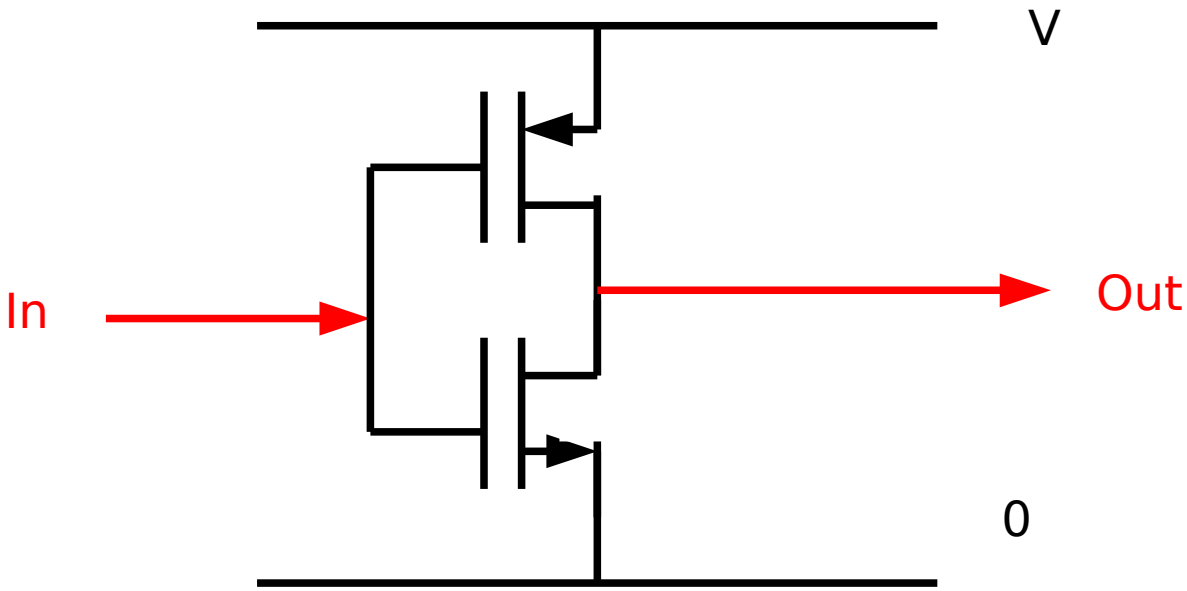
- NOT can be done trivially with an nMOS



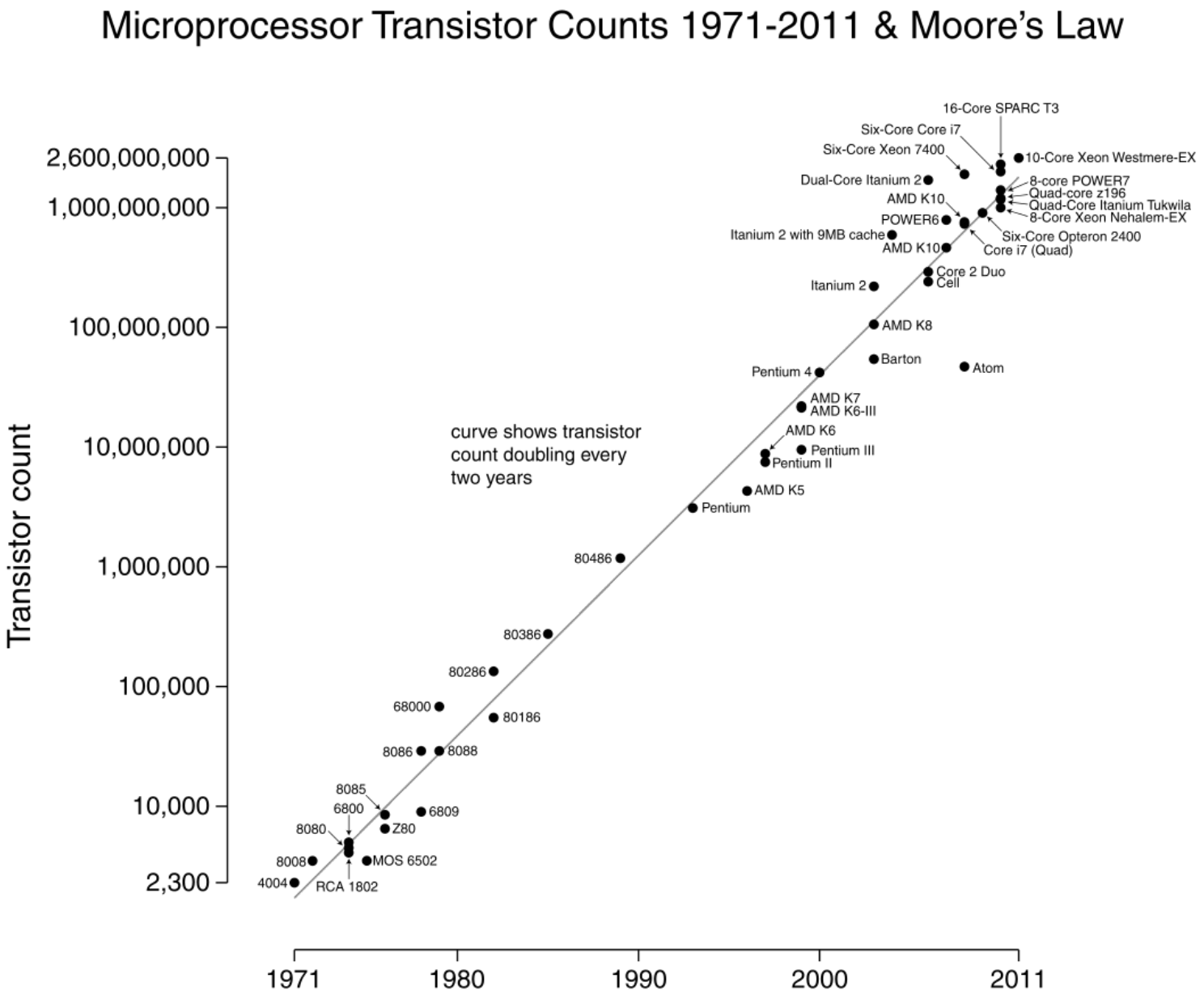
- But when the transistor is 'closed', a current flows through the resistor. So it works but is a very power hungry approach

CMOS

- Today's computers use CMOS (Complementary-MOS)
- i.e. combinations of nMOS and pMOS
- E.g.



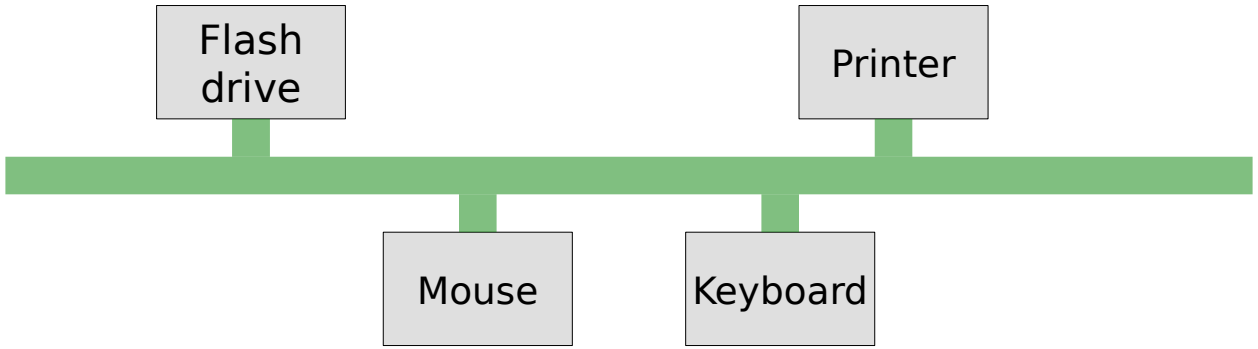
Transistor Counts



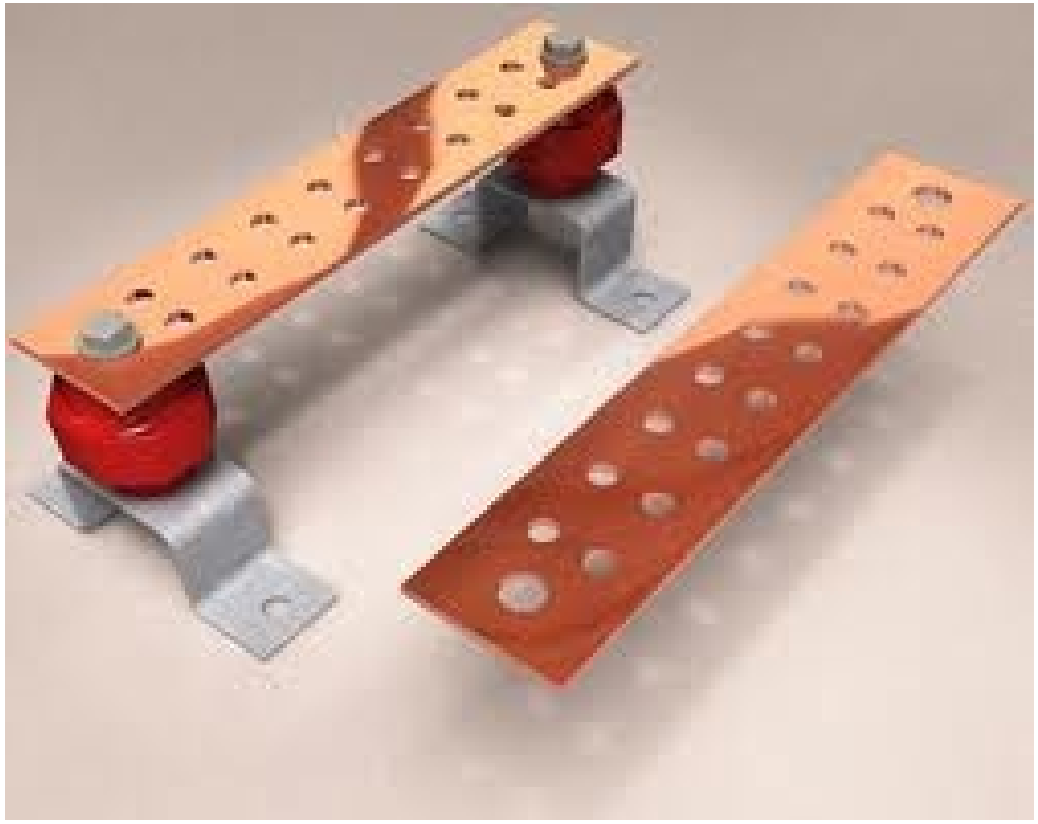
Transistor Size

Communications

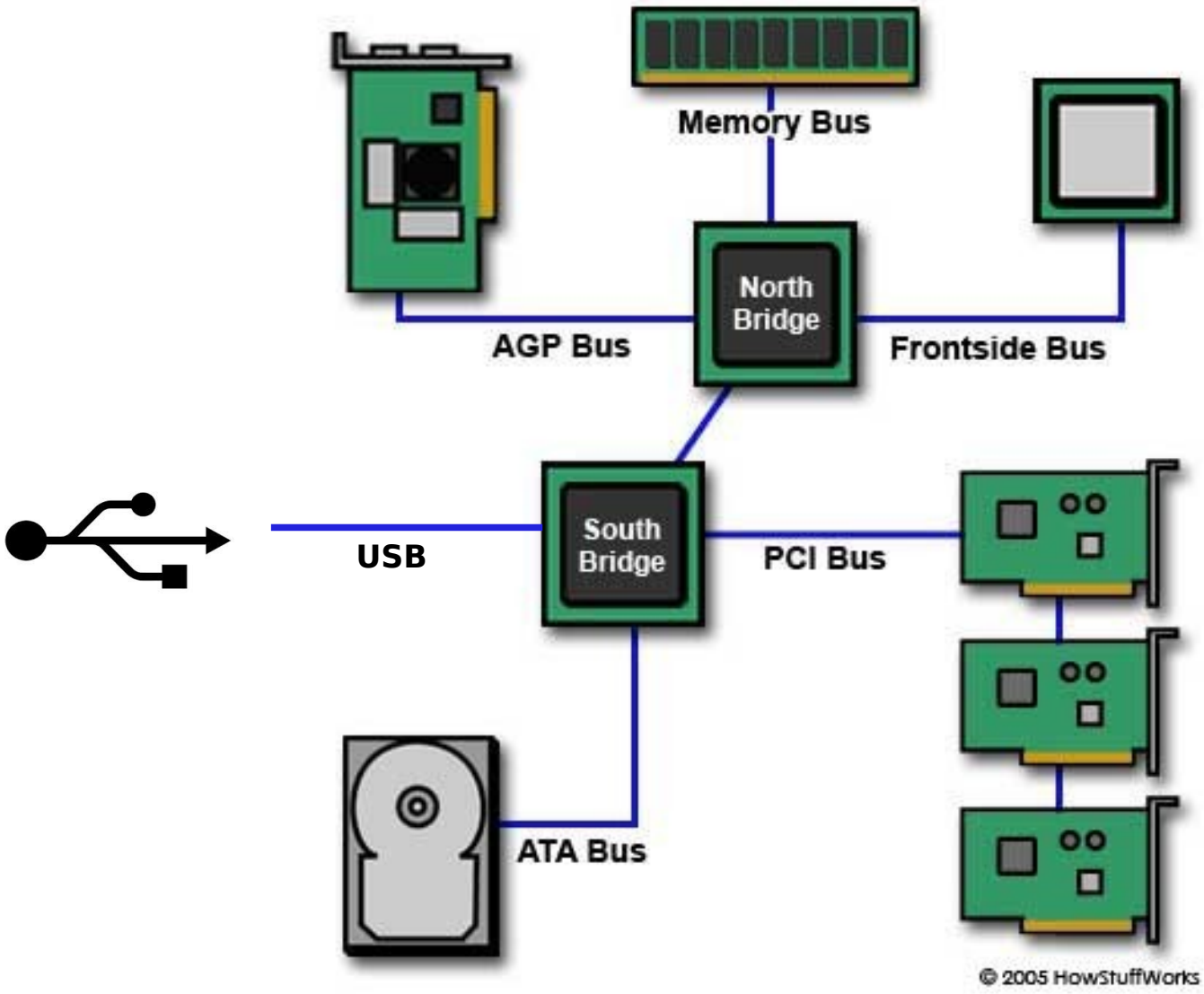
- A useful computer needs peripherals
  - Input (mouse, keyboard, etc)
  - Output (printer, display)
  - Network adapter, etc
- Peripherals connect to **buses** in order to communicate with the core system
  - A bus is just a set of wires that can be used by multiple peripherals.



“Bus” from Power Busbar



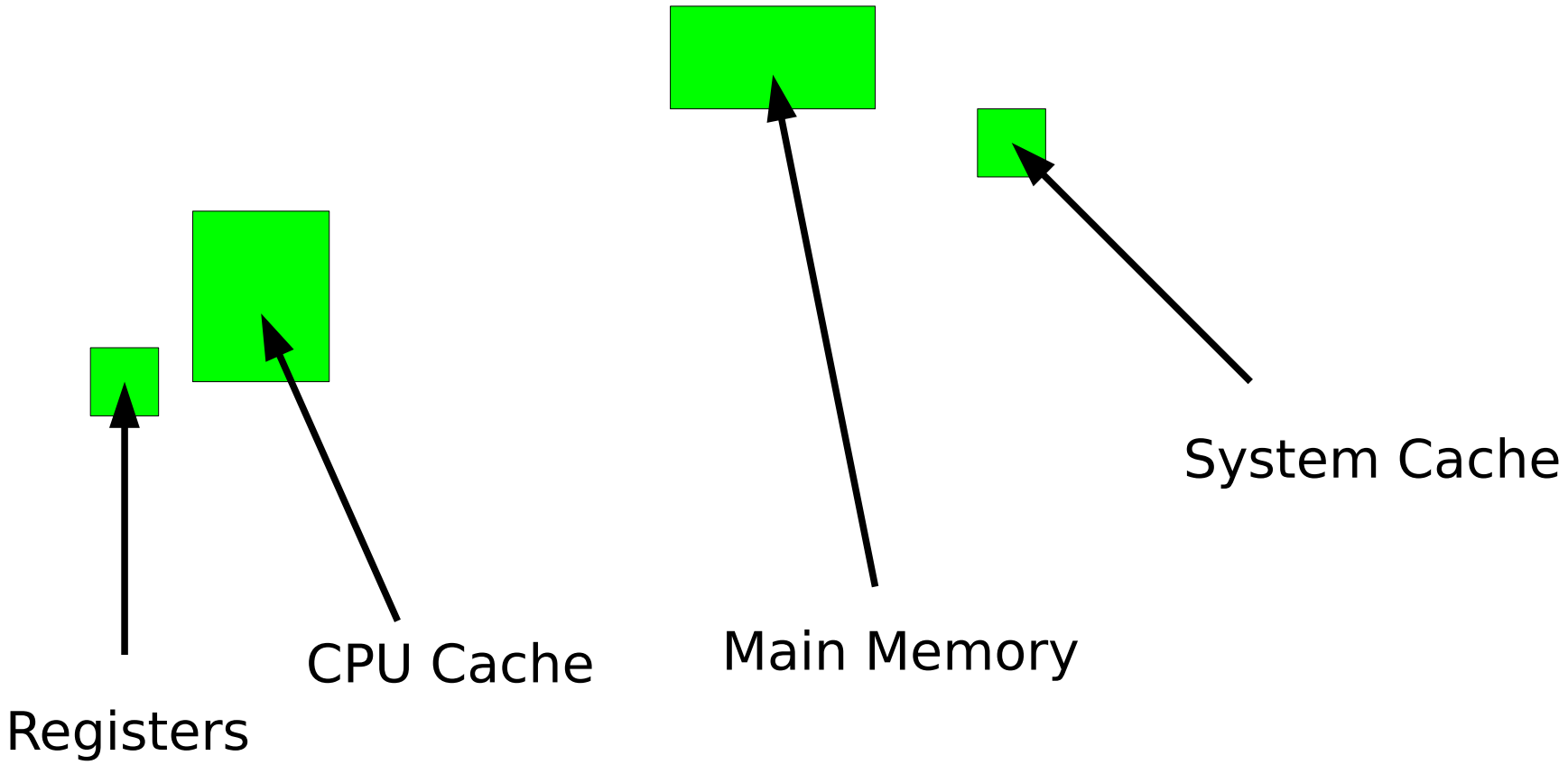
Typical Desktop Architecture



The Motherboard

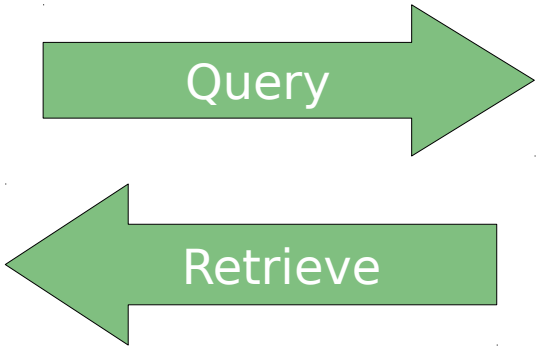
- An evolution of the circuitry between the CPU and memory to include general purpose buses (and later to integrate some peripherals directly)
- Internal Buses
  - ISA, PCI, PCIe, SATA, AGP
- External buses
  - USB,Firewire,
    eSATA, PC card

Typical Memories



Caches

- Put frequently-accessed data in a fast cache to speed things up

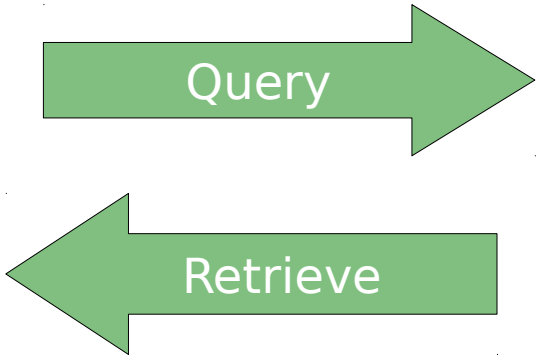


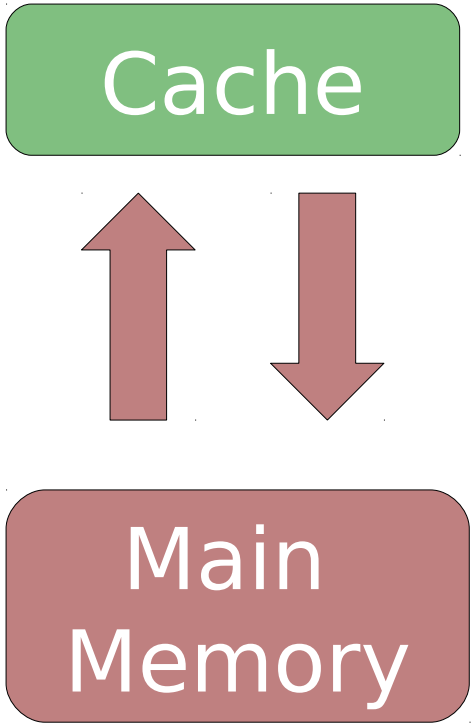
Cache

Cache hit: it's in the cache (fast)

Caches

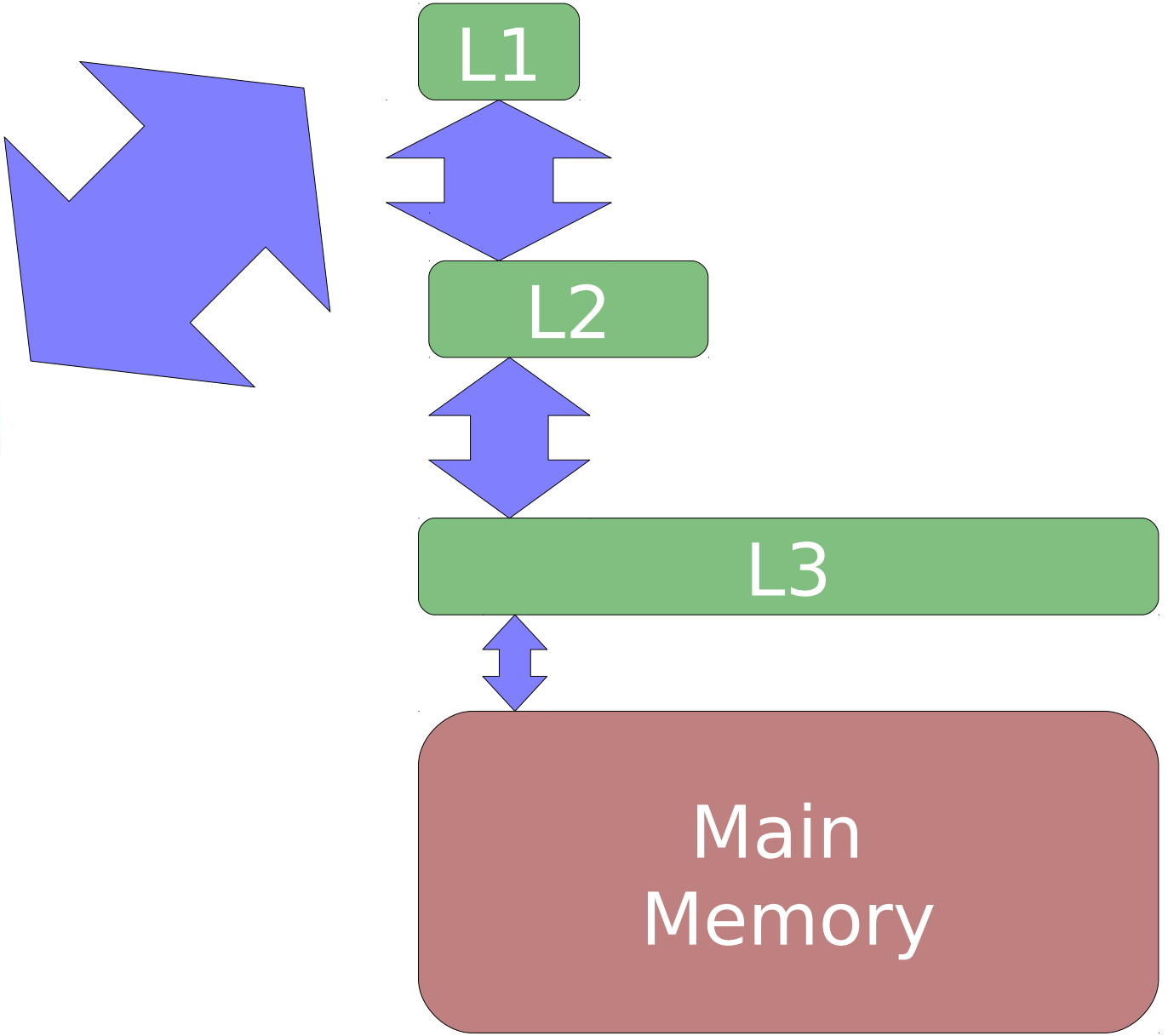
- Put frequently-accessed data in a fast cache to speed things up



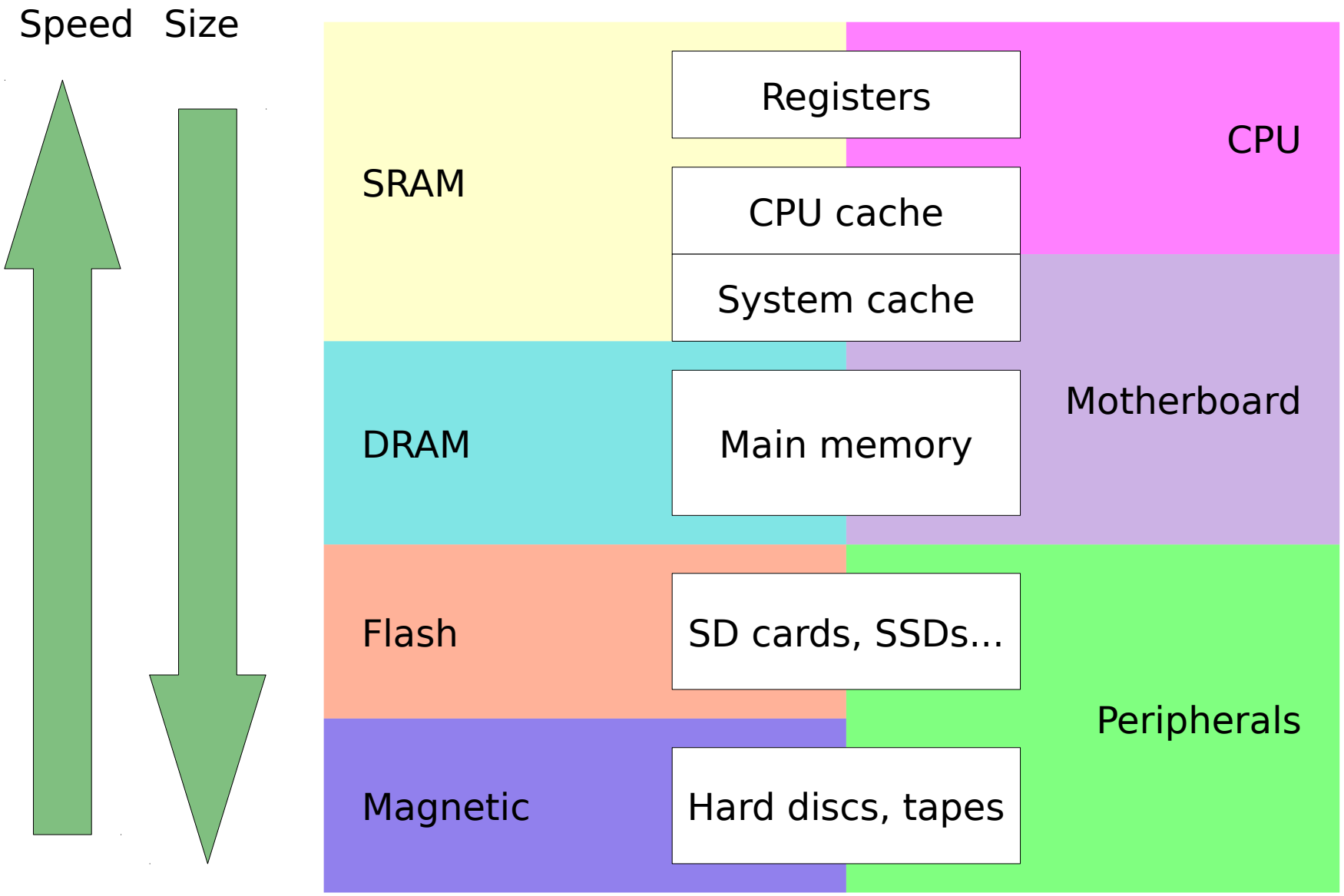


Cache miss: get from memory (slower)

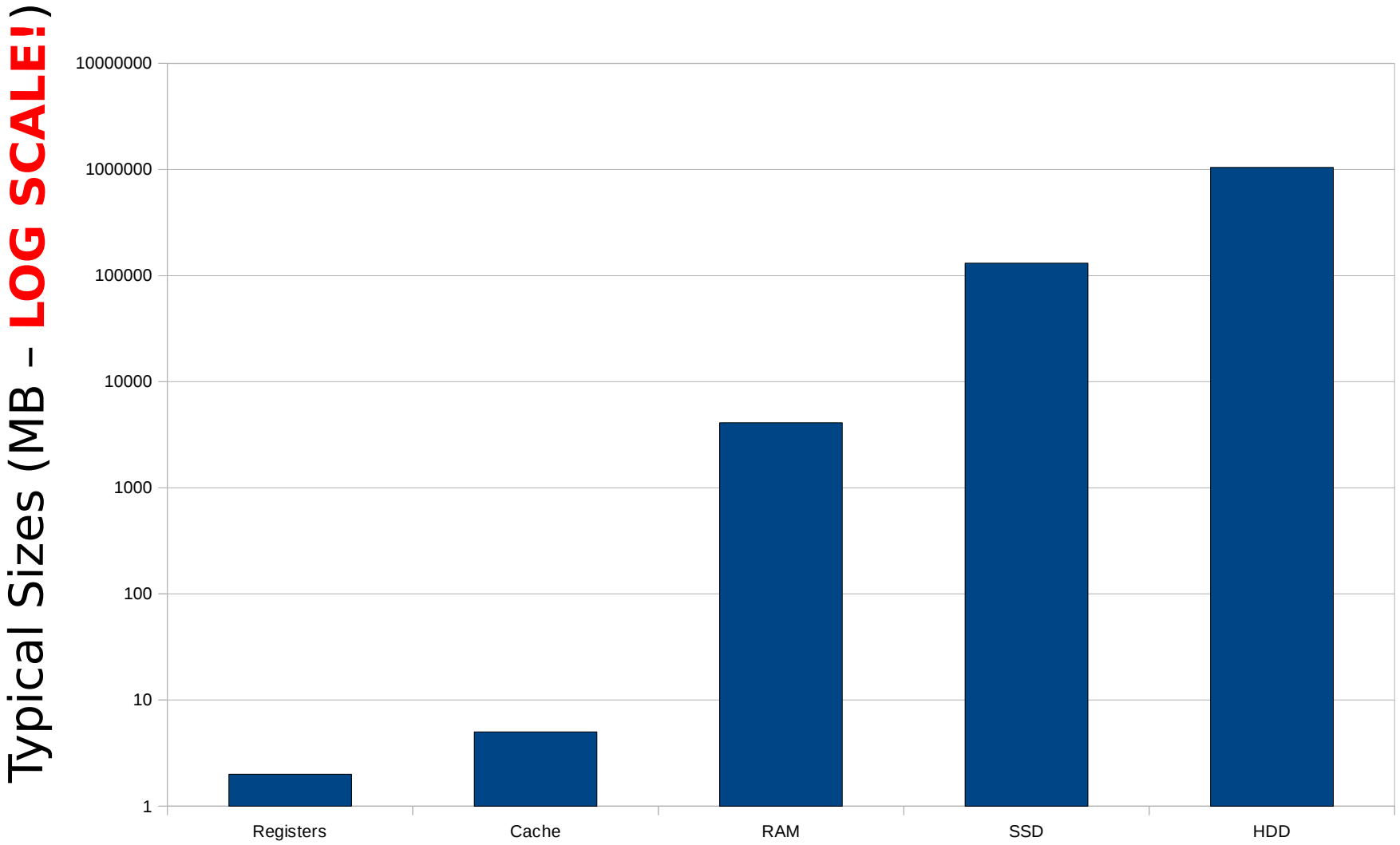
Cache the Cache!



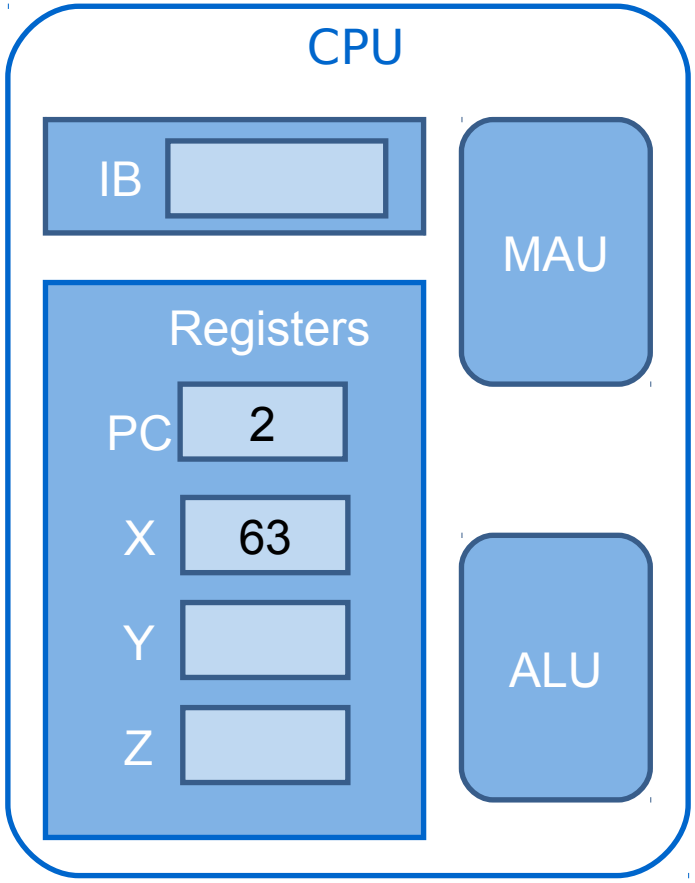
Memory Hierarchy (Typical)



Typical Memory Capacities

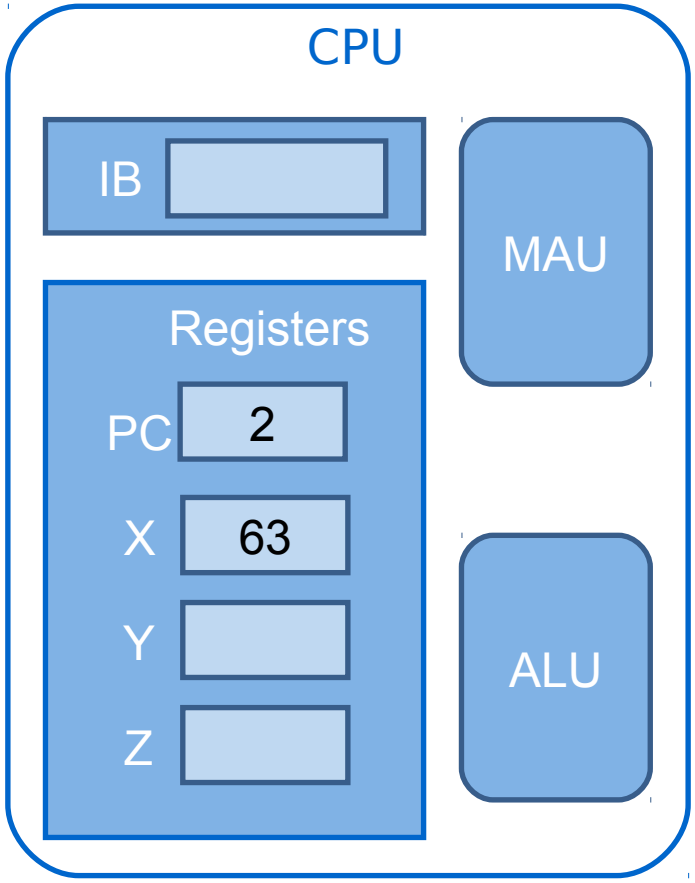


Register Size Limits Memory



- Each slot in memory has a unique address
- The address must fit inside a register
- 32 bits → 2^{32} slots
- 2^{32} bytes → 4 GB
- (64 bits → 18 quintillion bytes ~ 10^{19} bytes)

Register Size Limits Memory



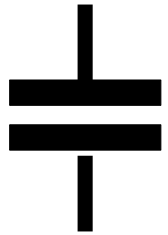
- Each slot in memory has a unique address
- The address must fit inside a register
- 32 bits → 2^{32} slots
- 2^{32} bytes → 4 GB

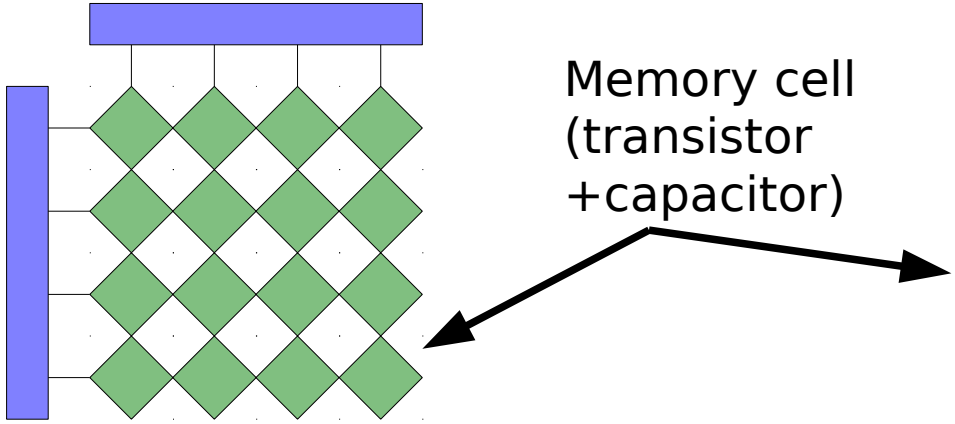
Register Sizes

- Registers are fixed size, super-fast on-chip memory usually made from SRAM.
- When we build the CPU we have to decide how big to make them
  - Bigger registers
    - Allow the CPU to do more per cycle
    - Mean we can have more main RAM (longer addresses can be supported)
    - Too big and we might never use the whole length (waste of electronics)
  - Smaller registers
    - Less electronics (smaller, cooler CPUs)
    - Too small and it takes more cycles to complete simple operations

Random Access Memory (DRAM)

- Capacitor + transistor = memory cell
- Capacitor charged → 1, discharged → 0
- Matrix of cells → transistors allow us to 'activate' cells
- Hence we can randomly jump around in the data (random access)





- This is Dynamic RAM (DRAM), cheap and simple
- BUT: capacitors leak charge over time, so a “1” becomes a “0”. Therefore we must refresh the capacitor regularly and this slows everything down plus it drains power...

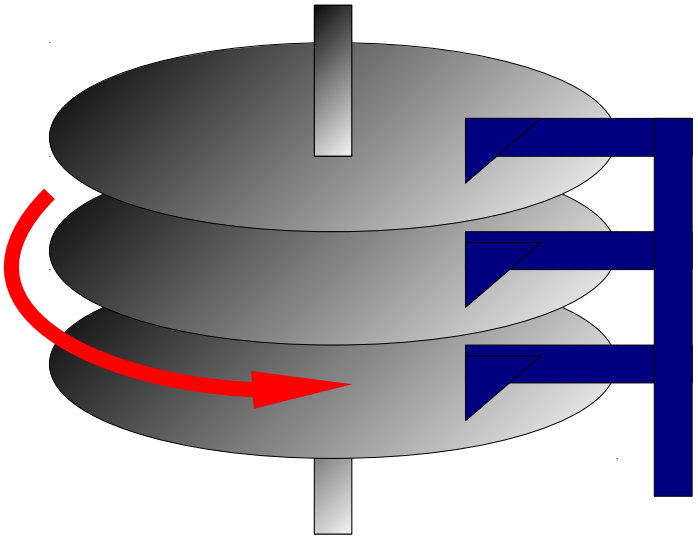
Static RAM (SRAM)

- We can avoid the need to refresh by using Static RAM (SRAM) cells. These use more electronics (typically 6 transistors per cell) to effectively self-refresh.

SRAM Memory Cell

- This is 8-16x faster than DRAM
- But each cell costs more and takes more space so it's also about 8-16x more costly!
- And both DRAM and SRAM are volatile (lose power = lose data)

Magnetic Media (Hard Discs)



- Lots of tiny magnetic patches on a series of spinning discs
- Similar to an old cassette tape only more advanced
- Read and write heads move above each disc, reading or writing data as it goes by

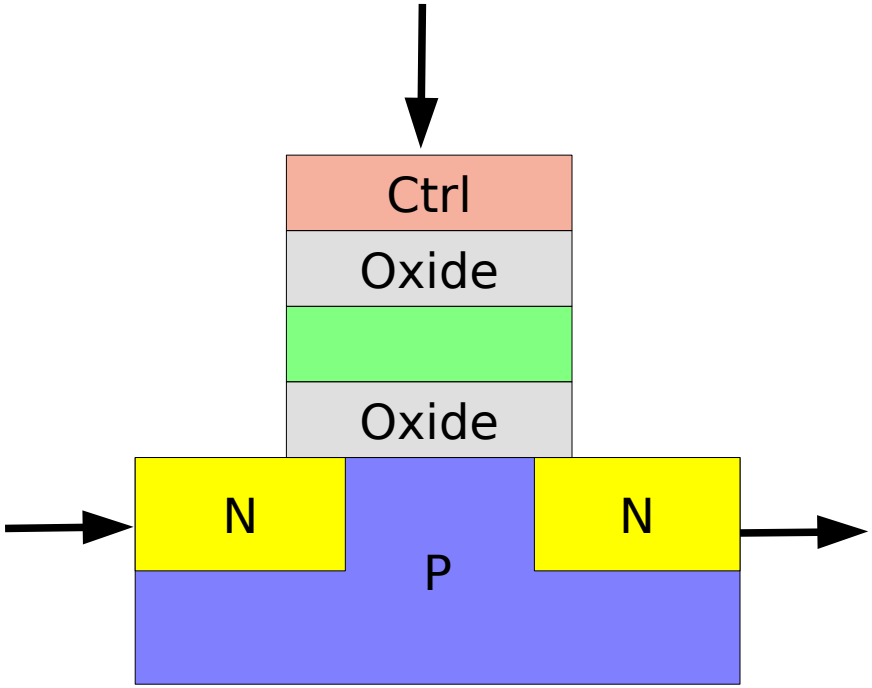
- Remarkable pieces of engineering that can store terabytes (TB, 1,000,000MB) or more.
- Cheap mass storage
- Non-volatile (the data's still there when you next turn it on)
- But much slower than RAM (→ SAM)

Flash and SSDs

- Toshiba came up with Flash memory in the 1980s as a non-volatile storage without moving parts

Flash and SSDs

- Toshiba came up with Flash memory in the 1980s as a non-volatile storage without moving parts
- Floating gate MOSFET

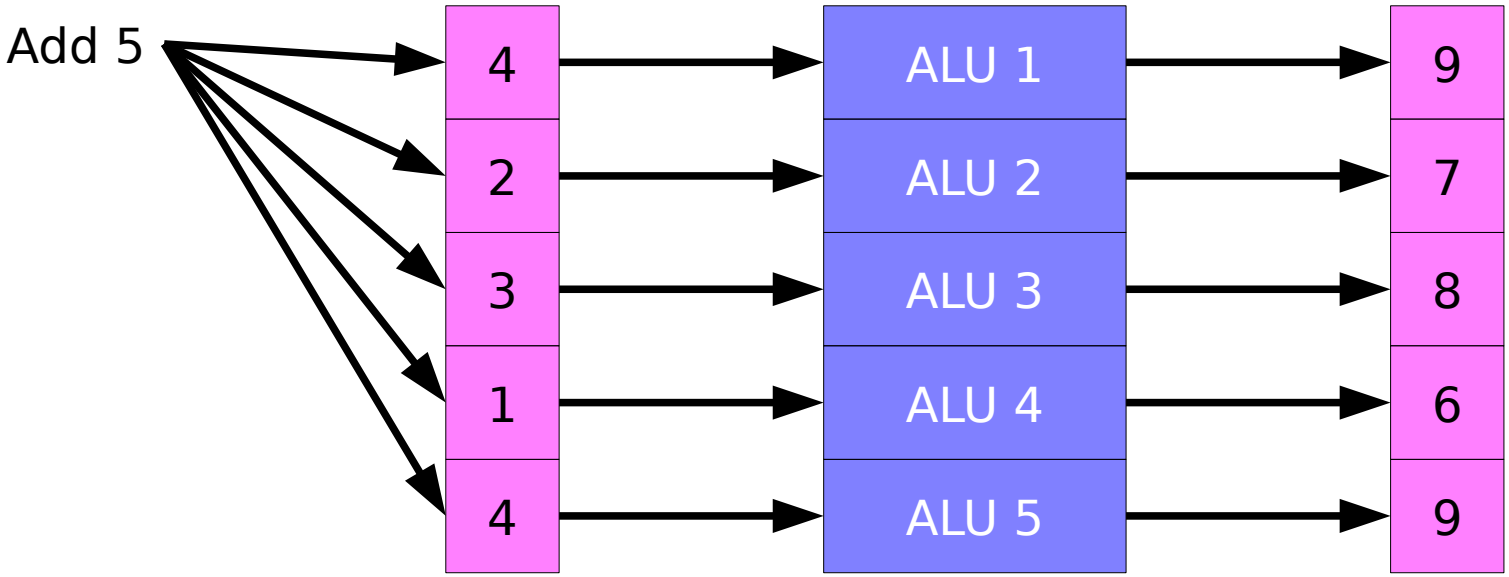


Graphics Cards

- Started life as simple Digital to Analogue Convertors (DACs) that took in a digital signal and spat out a voltage that could be used for a cathode ray screen
- Have become powerful computing devices of their own, transforming the input signal to provide fast, rich graphics.
- Todays GCs are based around GPUs with lots of tiny processors (cores) sharing some memory. The notion is one of SIMD – Single Instruction Multiple Data
  - Every instruction is copied to each core, which applies it to a different (set of) pixel(s)
  - Thus we get parallel computation → fast
  - Very useful for scientific computing
  - CPUs better for serial tasks

GPUs and SIMD

- So called vector processing: apply one instruction to a vector of data



RISC

- The simplest way to create a CPU is to have a small number of simple instructions that allow you to do very small unit tasks
  - E.g. load a value to a register, add two registers
  - If you want more complicated things to happen (e.g. multiplication) you use just use multiple instructions
  - This is a **RISC** approach (Reduced Instruction Set arChitecture) and we see it in the ARM CPUs

CISC

- Actually, two problems emerged
  - People were coding at a low level and got sick of having to repeatedly write multiple lines for common tasks
  - Programs were large with all the tiny instructions. But memory was limited...
- Obvious soln: add “composite” instructions to the CPU that carry out multiple RISC instructions for us
  - This is a **CISC** (Complex Instruction Set arChitecture) and we see it in the Intel chips
  - Instructions can even be variable length

RISC vs CISC

RISC

- Every instruction takes one cycle
- Smaller, simpler CPUs
- Lower power consumption
- Fixed length instructions

CISC

- Multiple cycles per instruction
- Smaller programs
- Hotter, complex CPUs
- Variable length instructions

RISC vs CISC

- CISC has traditionally dominated (for backwards compatibility and political reasons) e.g. Intel PCs
- RISC was the route taken by Acorn, and resulted in the ARM processors e.g. smartphones