CPS 188

Computer Programming Fundamentals

Prof. Alex Ufkes

Topic 1.1: Course intro, computing intro, C intro

Toronto Metropolitan University

Notice!

Obligatory copyright notice in the age of digital delivery and online classrooms:

The copyright to this original work is held by Alex Ufkes. Students registered in course CPS 188 can use this material for the purposes of this course but no other use is permitted, and there can be no sale or transfer or use of the work for any other purpose without explicit permission of Alex Ufkes.

Instructor

Alex Ufkes

aufkes@torontomu.ca

Class times (Sec 16-18):

Wed 11-1, DSQ 01

Fri 11-1, DSQ 13

When Contacting…

- E-mail – I check it often (**aufkes@ryerson.ca**)
- Please put CPS188 in the subject line
- Include your full name
- Ideally, use your Ryerson account

My Background

Computer Science:

- BSc in computer science
- MSc in computer science
- Teaching since 2017

My research focus has been in search and rescue robotics and related technologies

Disaster Scene Reconstruction

+

(… and thousands of lines of C code)

Computer Programming Fundamentals

https://www.cs.ryerson.ca/~cps188/

Course Leaning Outcomes:

- Understand computers, how they work, and how data is represented and processed inside them.
- Use the basic and advanced functionalities of the C programming language.
- Create algorithms to solve different engineering related problems.
- Write programs in the C programming language to solve simple and complex engineering problems.

Mark Breakdown

https://www.cs.ryerson.ca/~cps188/

Assessment	**Weight**
Labs	25%
Midterm	25%
Project	25%
Exam	25%
Total	**100%**

References

https://www.cs.ryerson.ca/~cps188/

Textbook:
Jeri R. Hanly & Elliot B. Koffman (2016). *Problem Solving and Program Design in C, 8th Edition*. Pearson. ISBN: 978-01342-4394-8

- Lecture slides/notes will be posted to D2L
- When it comes to coding, online resources are excellent and ubiquitous.
- Google is great but be careful – you're likely to find a lot of complex or irrelevant information.

How to ~~Succeed~~ EXCEL!

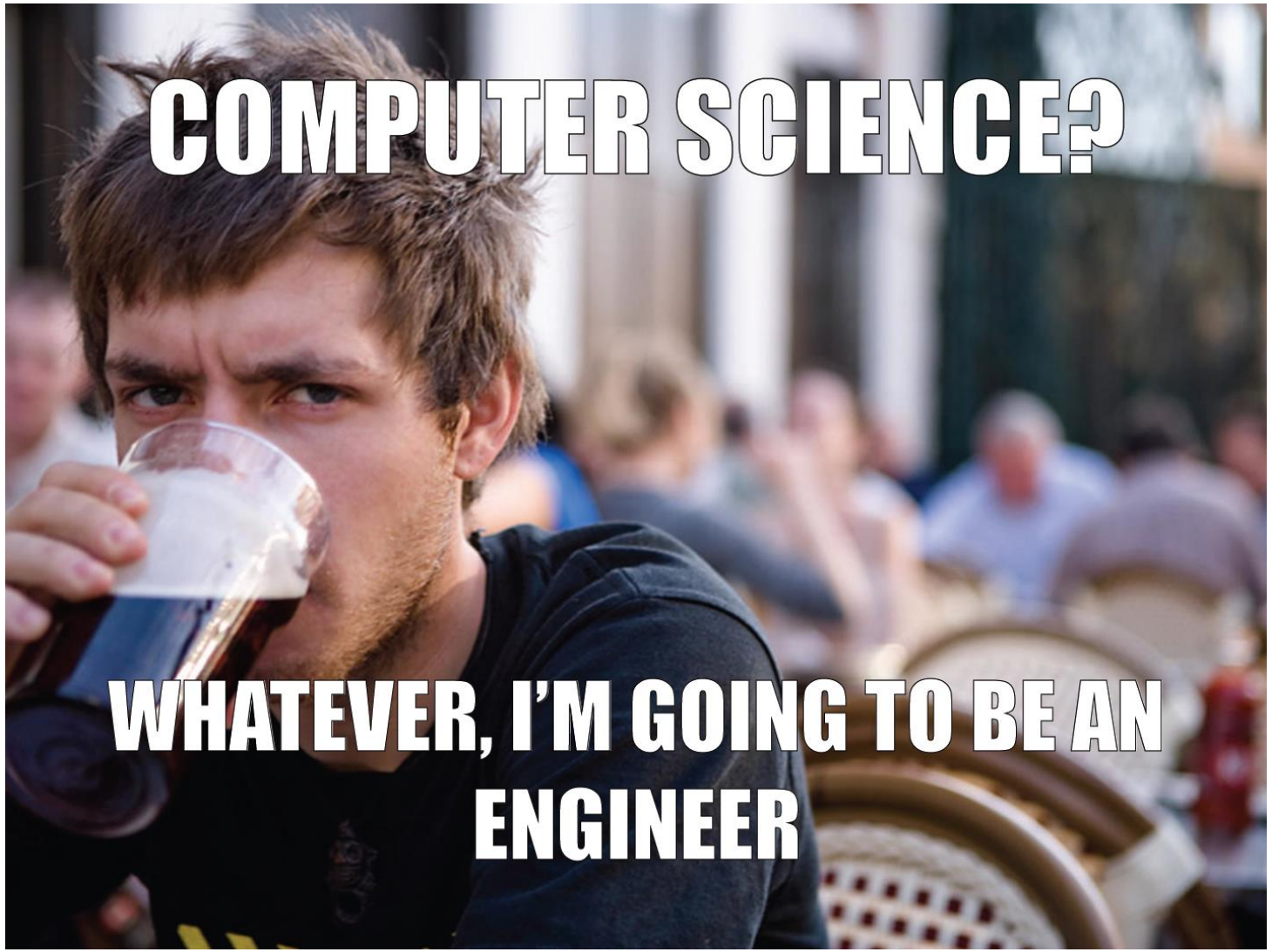
1) Write code. Write A LOT of code.

Just because you understand how a program works doesn't mean you could have written it yourself.

2) Debug your own code.

Don't ask for help right away. Compare your syntax with a working program, try to understand error messages, etc.

Why Programming?



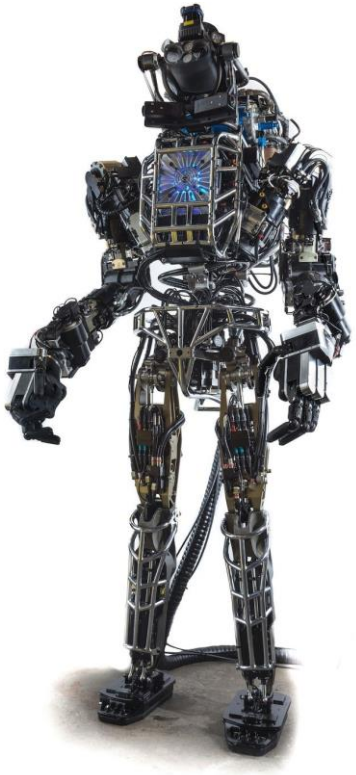
Why Programming?

Why do we need to know how to program?

- Everything has a processor or microcontroller in it these days that runs some kind of software.
- Internet of Things (IoT): Lightbulbs, wall sockets, power tools, appliances, *everything*.
- For better or worse, every electrical device is now digital.

Why C?

Most embedded software is based on C, and it doesn't write itself



Why C?

- C is efficient, portable, flexible, and is a subset of C++.
- C code can be compiled as C++, and we are free to add C++ syntax to existing C code. Two languages for the price of one!

Application to industry:

- Most embedded systems use some flavor of C/C++.
- Understanding C will lead to greater understanding of other languages. The process of logical thought is the same.
- It's not about memorizing C syntax but ***learning to think algorithmically***.

IEEE Spectrum's Top Programming Languages 2022

https://spectrum.ieee.org/top-programming-languages-2022

Questions So Far?

Today

- ~~Course intro~~
- A (brief) history of computing
- Intro to computer architecture
- Programs, algorithms, implementation
- C language intro

Wolf Radius Bone

25,000 - 30,000 B.C.E.

- 55 cuts in groups of five.
- Suggesting rudimentary form of multiplication or division
- Predates agriculture by ~20,000 years.

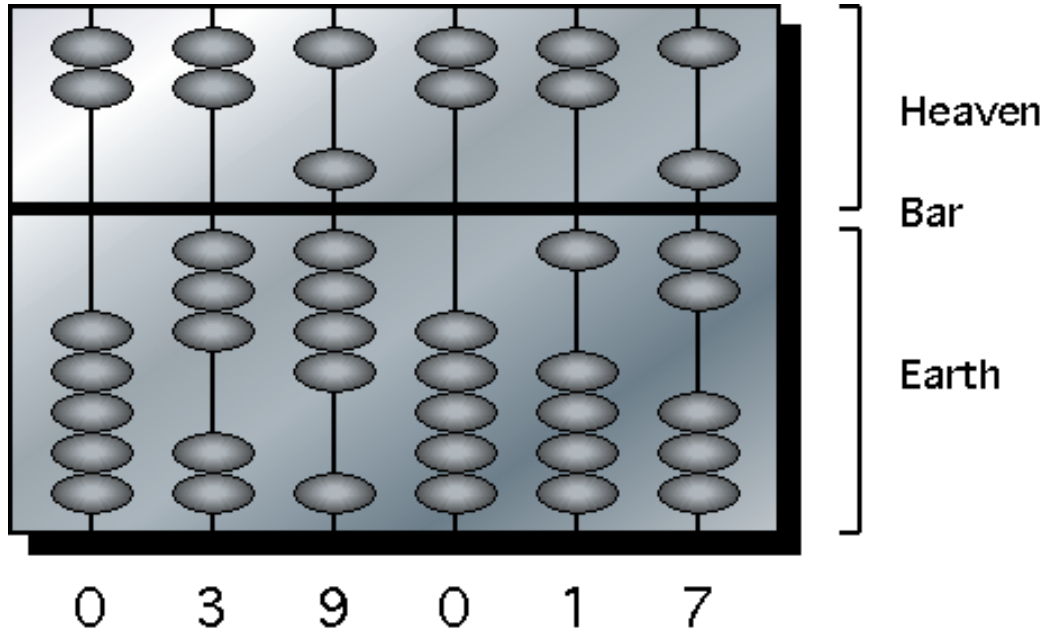
Tally Sticks

Westminster, England: 1250-1275 C.E.

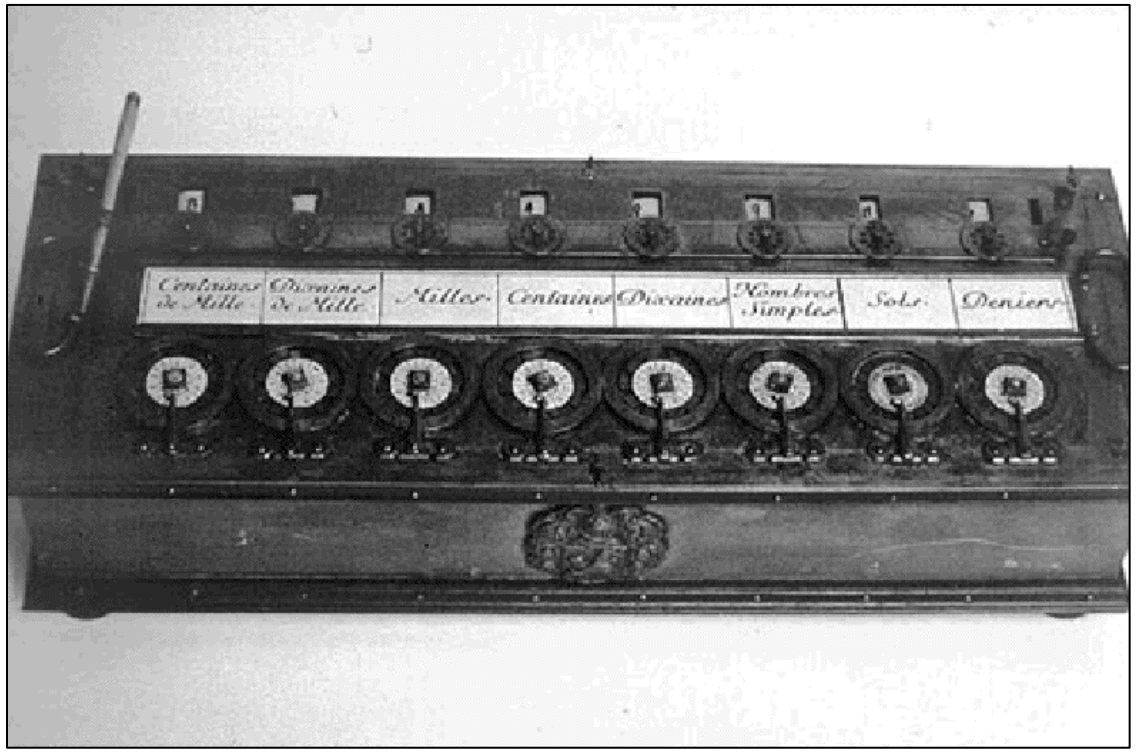
- Primitive memory aid device
- Wolf radius bone could have been a primitive tally stick
- Medieval Europe was largely illiterate and short on coin
- Tally sticks were notched with monetary values, split in half.
- One half, when matched to the other, was proof of value.

Chinese Abacus

- Abacus origins disputed
- What you think of when you hear “abacus” is from the Ming dynasty (12th century)
- Upper beads have value of 5, lower beads have a value of 1.
- Beads moved toward the bar are counted.
- Numbers represented are base-10



Pascal's Calculating Machine



Getting closer…

- "Pascaline", 1642-1645
- Performs addition, subtraction
- Multiplication and division achieved by chaining adds, subtracts
- Uniquely effective *carry* mechanism
- Many variations by others followed
- Still does not have the basic parts to be considered a ***computer***.

Jacquard Pattern Weaving Loom

Circa 1804, The Deutsches Museum

- Used a chain of punch cards to instruct the loom on how to weave intricate patterns

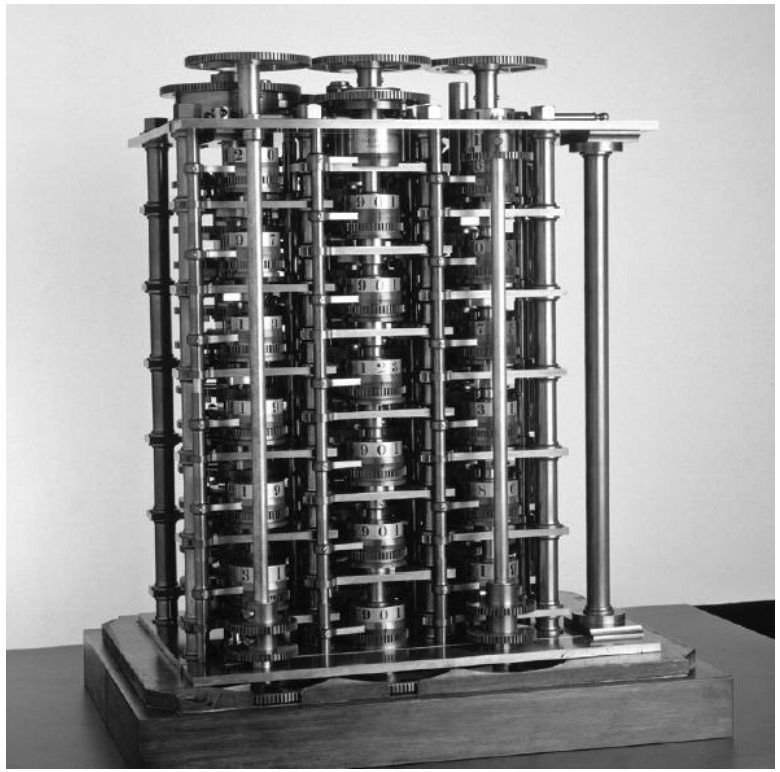
Jacquard Pattern Weaving Loom

Circa 1804, The Deutsches Museum

- Used a chain of punch cards to instruct the loom on how to weave intricate patterns
- Hundreds of cards whose holes correspond to hooks being raised or lowered
- First automated machine to use interchangeable punch cards

Babbage's Difference Engine

First known automatic calculator (1822)

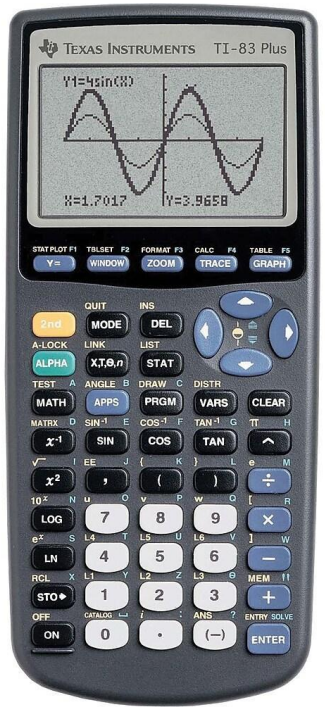


- Performed *divided differences*, a method for interpolating functions given a small set of polynomial coefficients.
- Never mind the details. It was used to approximate logarithms and Trig functions using polynomials.
- Very common in science, engineering, and navigation.
- Huge lookup tables could be computed

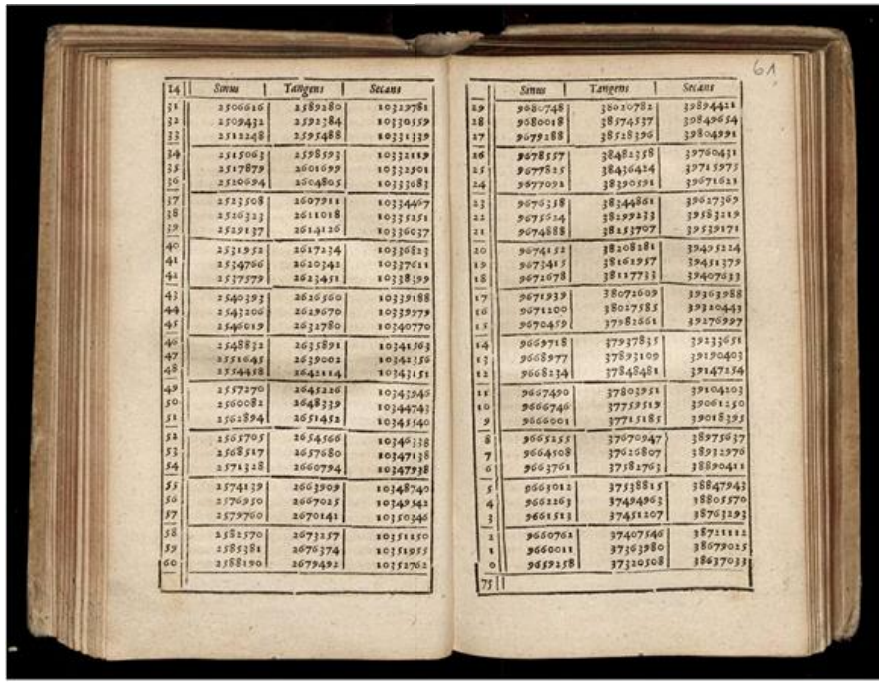
Babbage's Difference Engine

Huge lookup tables could be computed

Pre- these:



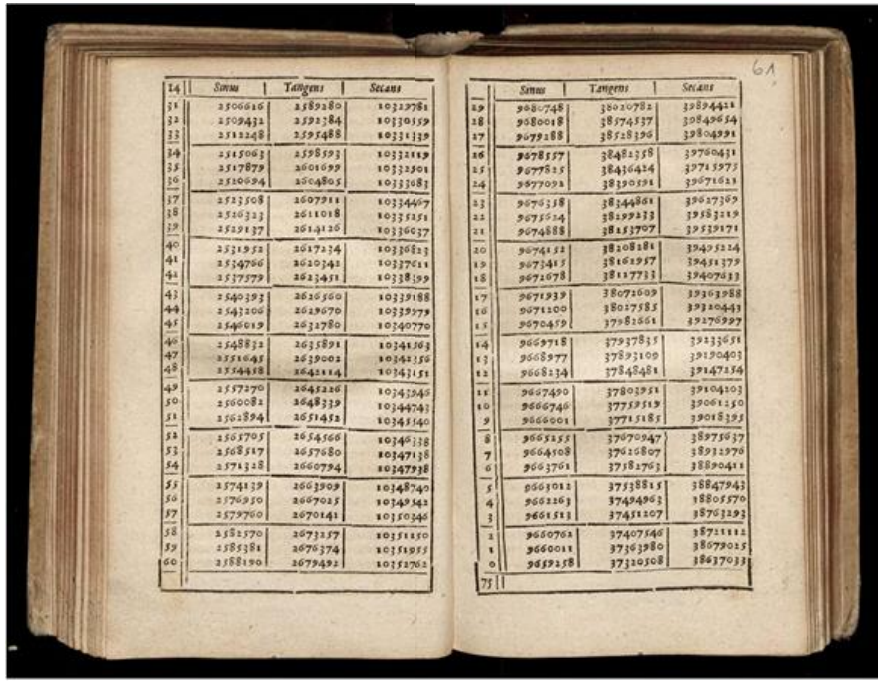
These were used:



Babbage's Difference Engine

Huge lookup tables could be computed

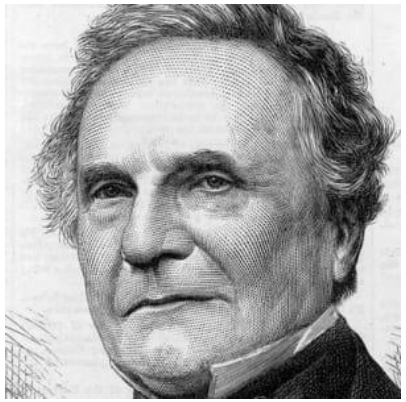
These were used:



- The difference engine was used to automatically compute such tables.
- There are many professors still teaching who used these in school.
- Such tables are still common for probability distributions.
- In fact, lookup tables are still very common in computing as well.

Babbage's Difference Engine

Clowns to the left of him, jokers to the right:



"On two occasions I have been asked [by members of Parliament]: '*Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?*' I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question."

- Charles Babbage

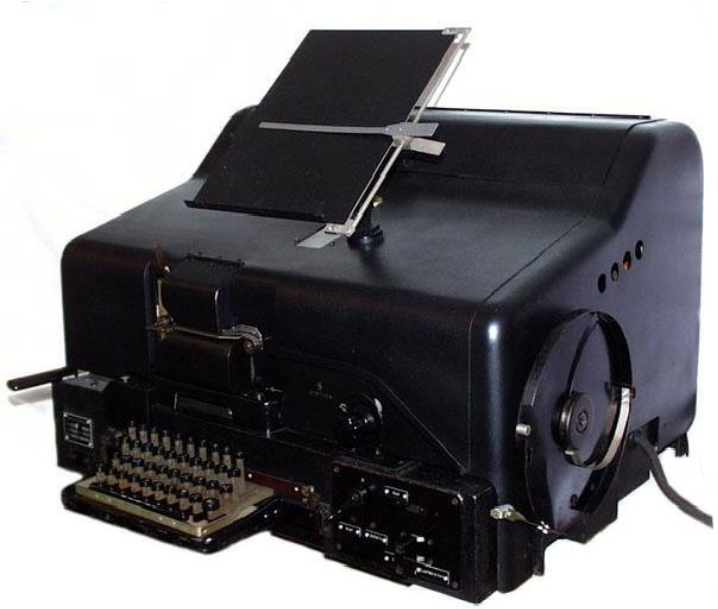
Ada Lovelace

The first “programmer” (1815-1852)

- Described the first algorithm use on Babbage’s analytical engine (Bernoulli numbers)
- Recognized the true potential of the difference engine as a general-purpose computing device.
- Not just number crunching!

Enigma

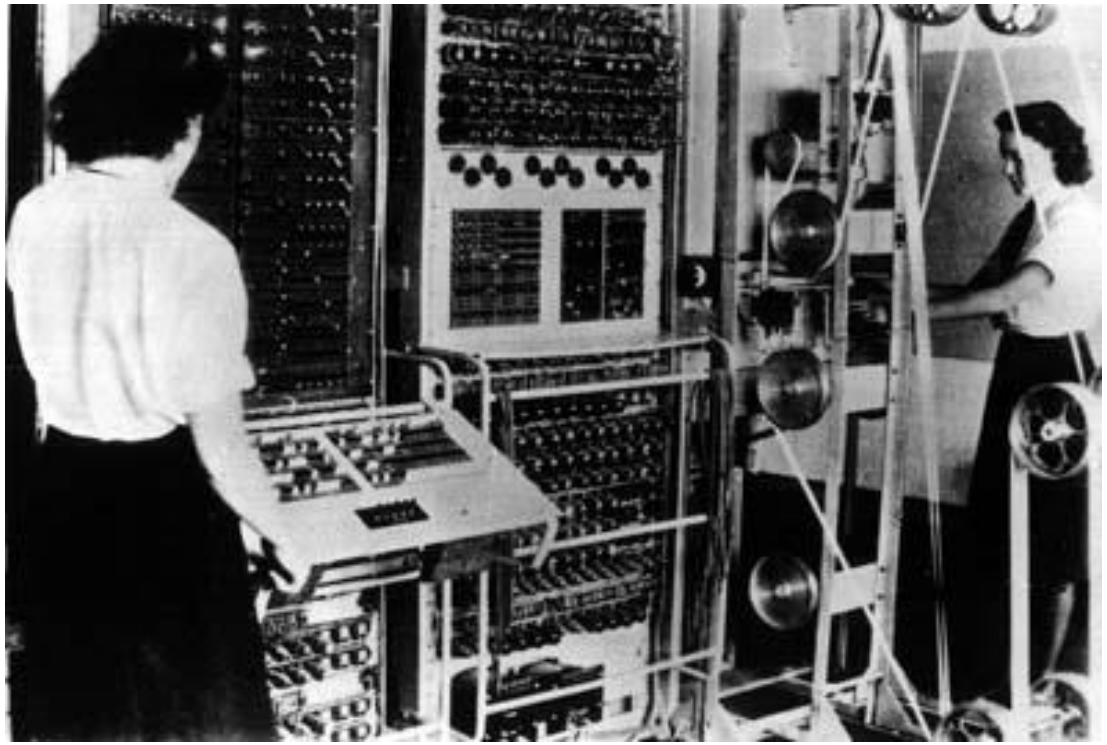
Siemens Halkse T-52 Sturgeon (Enigma) cipher machine



- Encryption device used from 1923 onward and used extensively in WWII.
- Each keystroke changes the electromechanical configuration of the internal rotors and lights
- Receiving station must be correctly initialized to match transmitting station.
- Initial configuration changed daily.
- Broken in late 30s by Polish “Bomba”
- Declassified in the 70s, well known in pop culture.

Colossus

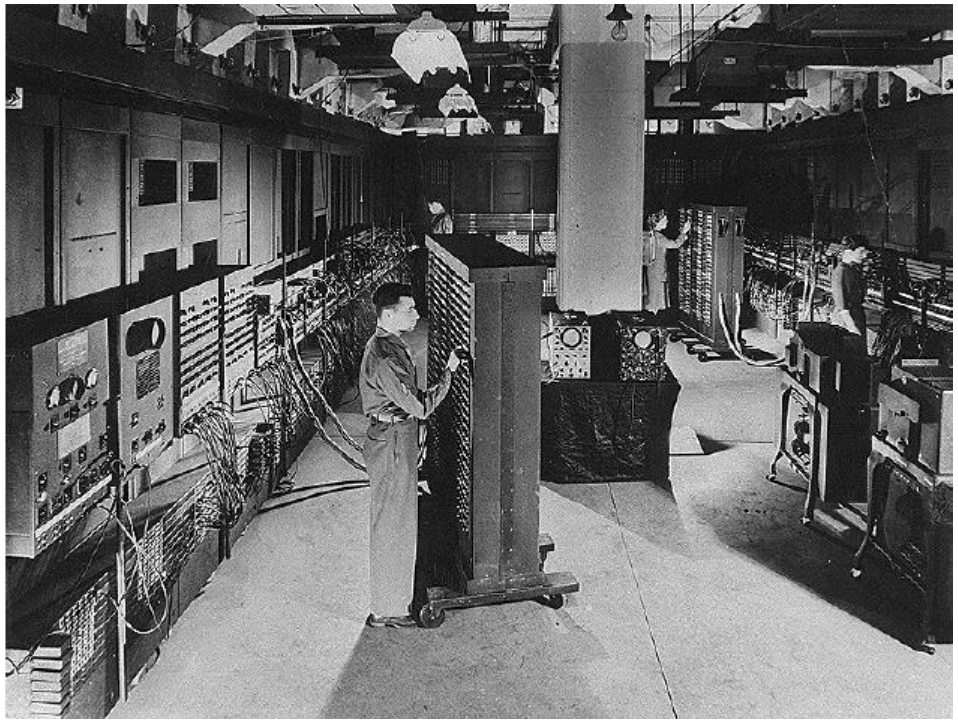
1943: World's first programmable, electronic, digital computer



- Programmed using switches and plugs, not a stored program
- Vacuum tubes used to perform Boolean and counting operations.
- Colossus Mark II improved processing speed.
- 10 Colossi were in use by the end of WWII to break *Lorenz* cipher
- Lorenz not declassified until 2002!

The ENIAC

1945 - Electronic Numerical Integrator and Computer



- First general-purpose computer
- Weighed 30 tons
- Built primarily to calculate ballistic trajectories.
- Did in 30s what took humans 20h
- By the end of 1956, ENIAC had:
  - 20,000 vacuum tubes
  - 7200 diodes
  - ~70,000 resistors
  - 5,000,000 hand-soldered joints

“Modern” Computing

“Modern” Computing

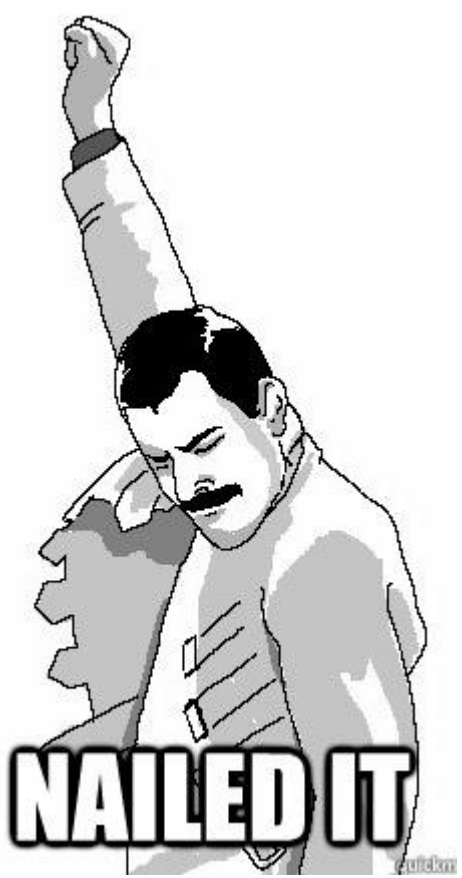
"I think there is a world market for maybe five computers."
(Thomas Watson, chairman of IBM, 1943)

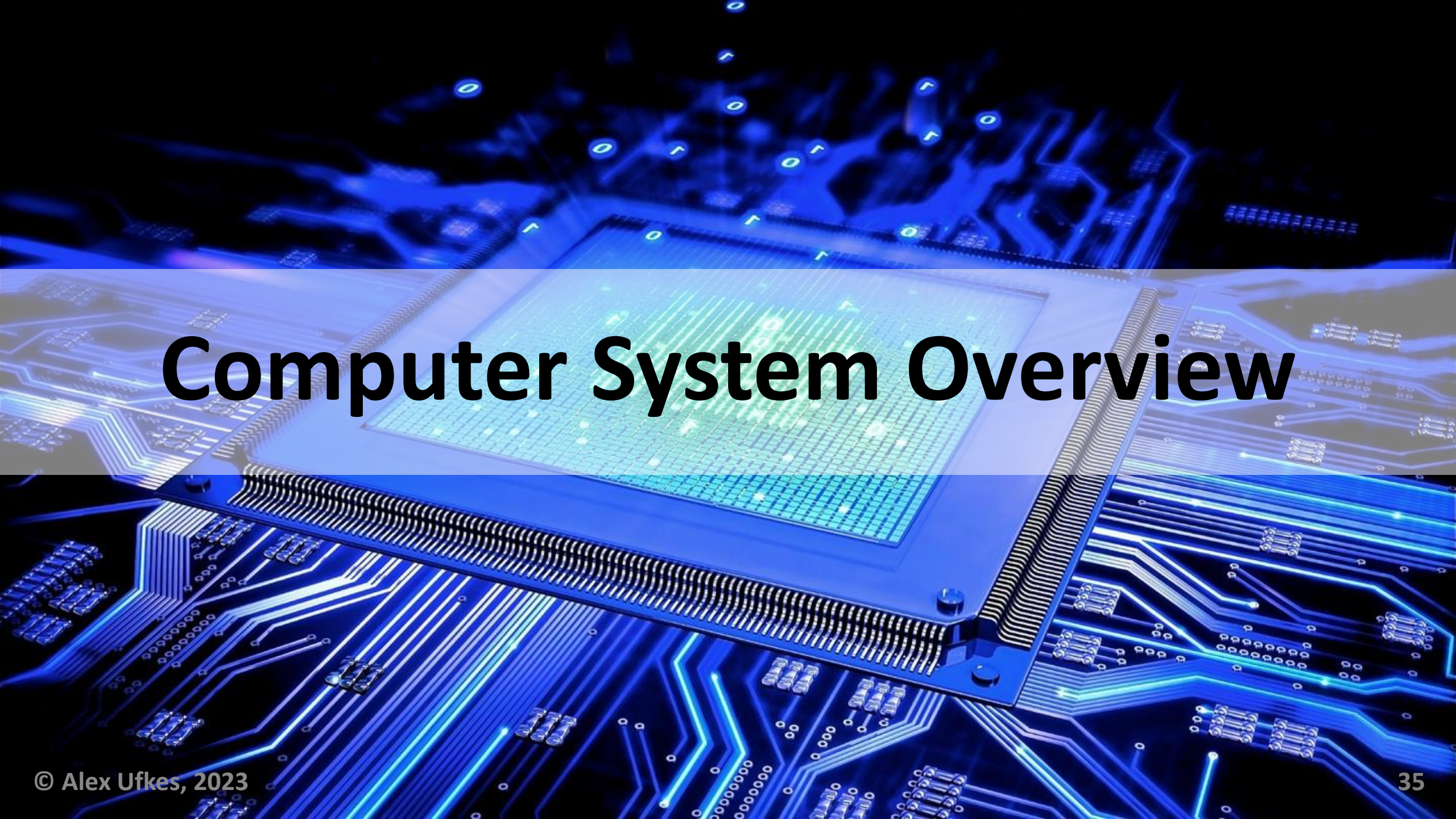
“Computers in the future may weigh no more than 1.5 tons.”
(Popular Mechanics, 1949)

“There is no reason anyone would want a computer in their home.”
(Ken Olsen, Digital Equipment Corporation, 1977)

“We will never make a 32-bit operating system.”
(Bill Gates, Microsoft, 1989)

“Spam will be a thing of the past in two years’ time.”
(Bill Gates, Microsoft, 2004)



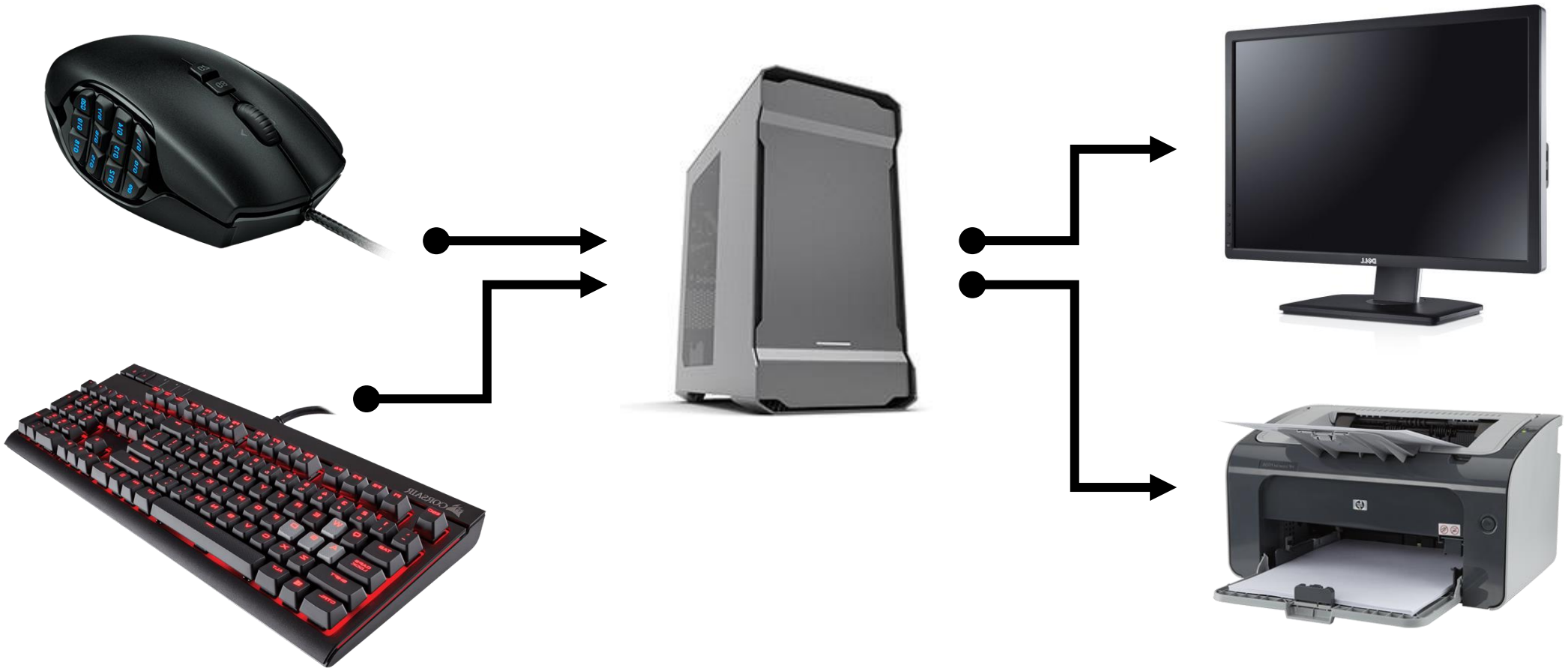


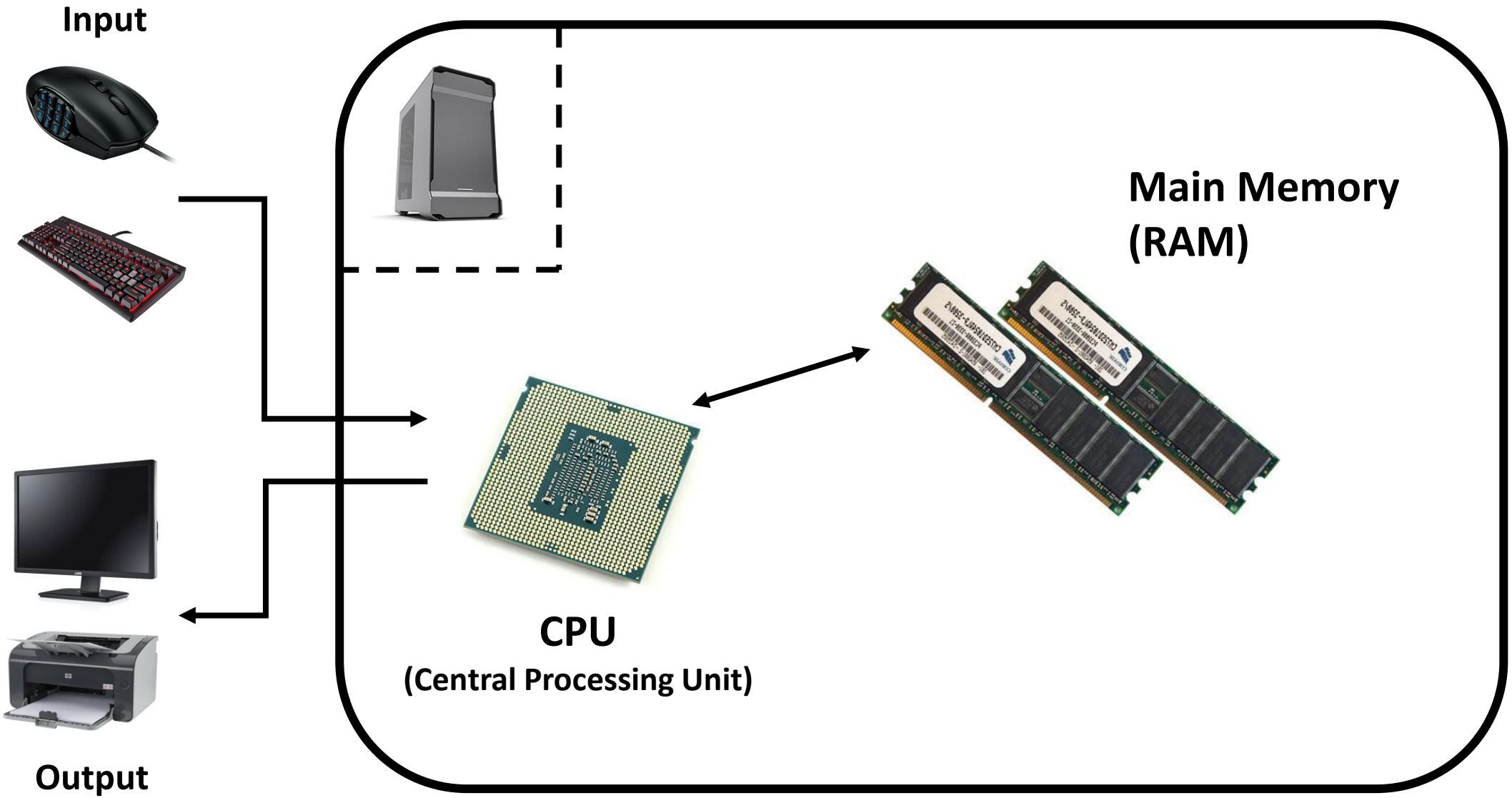
Computer System Overview

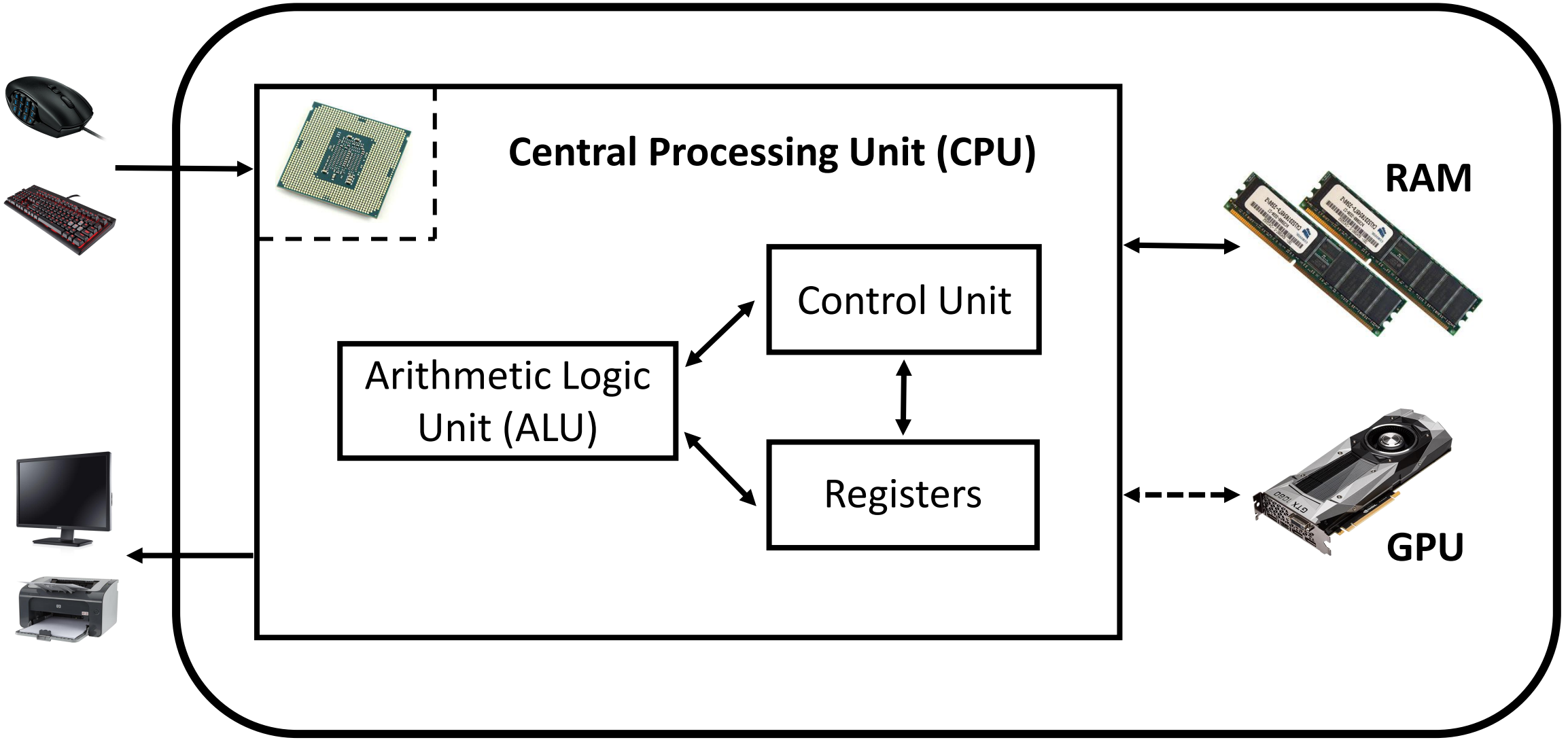
Modern Computer Systems

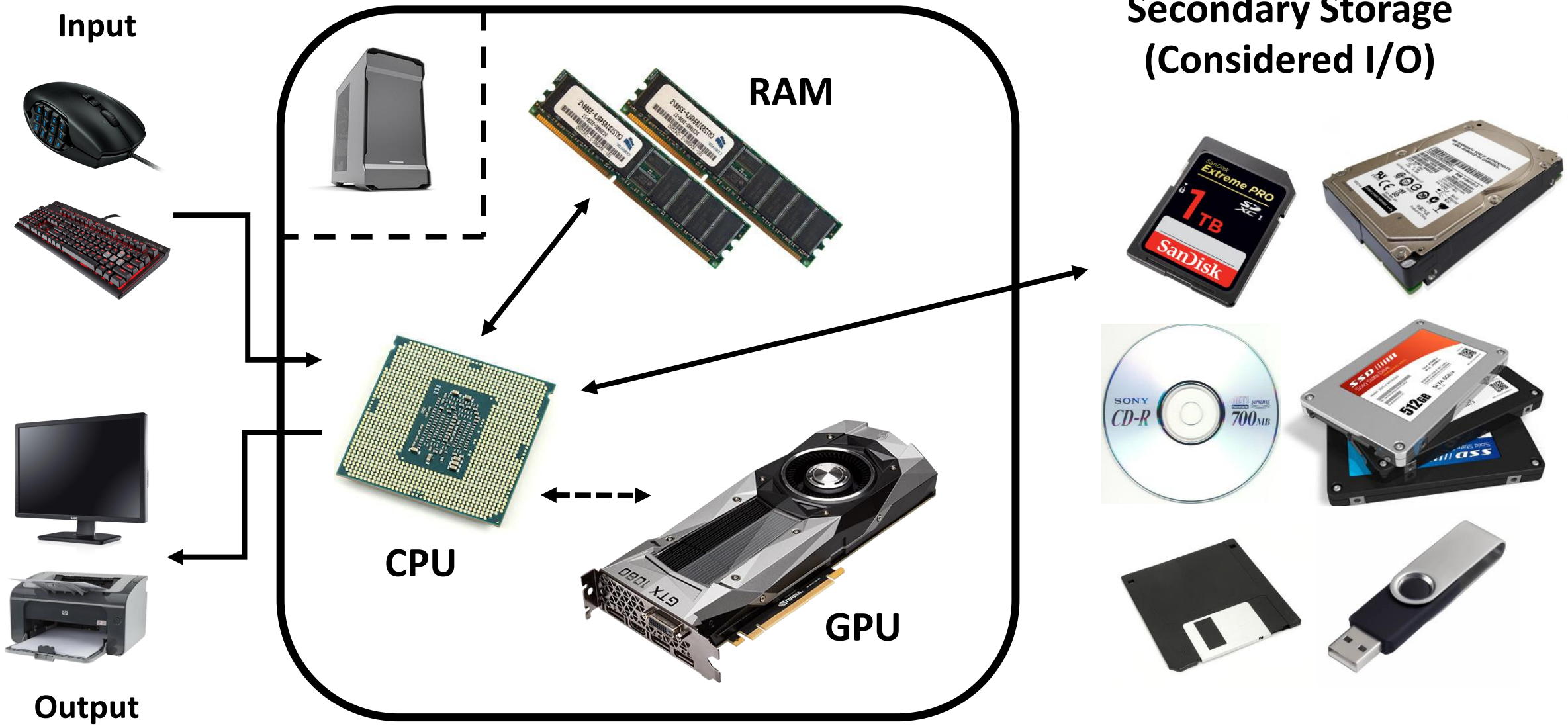
Input

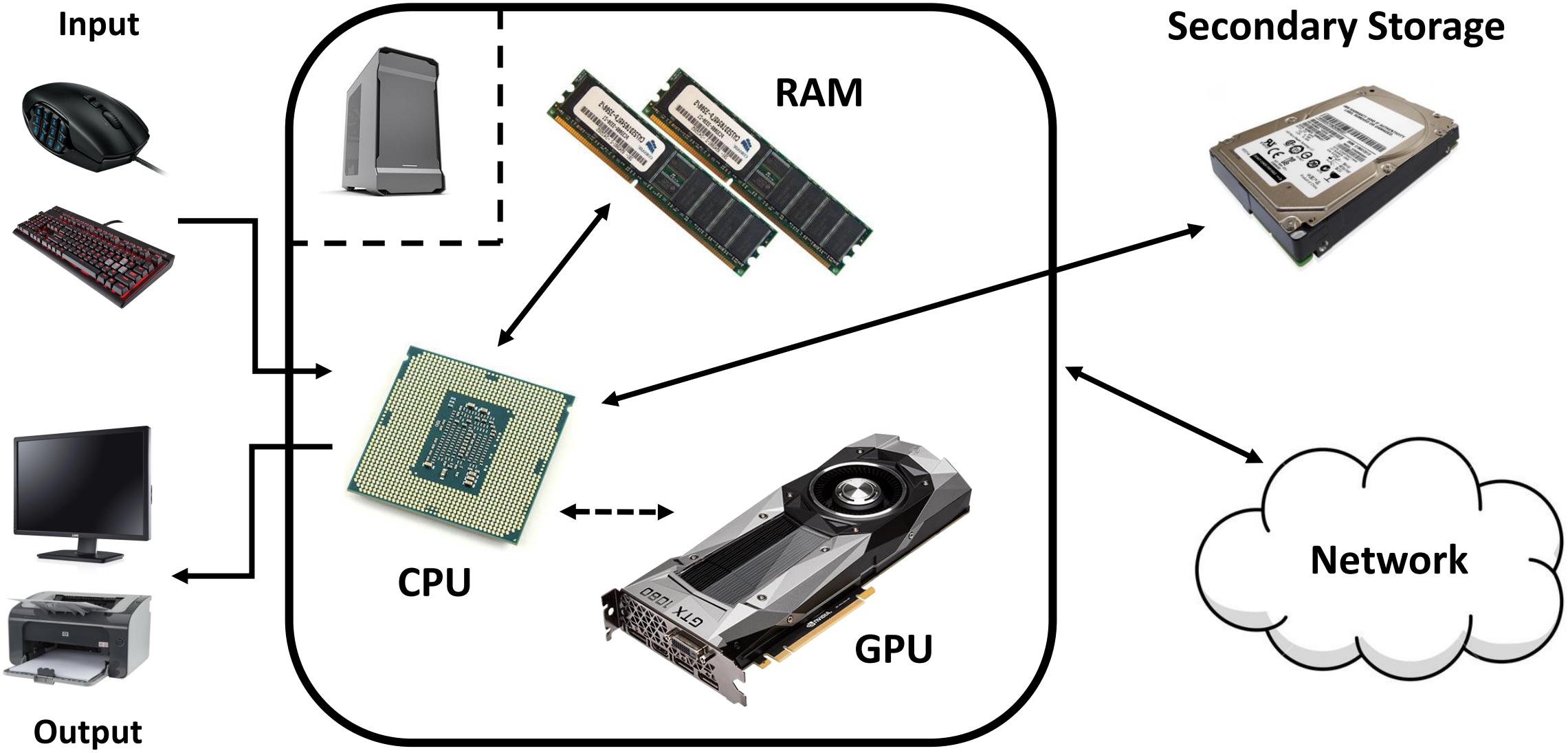
Output

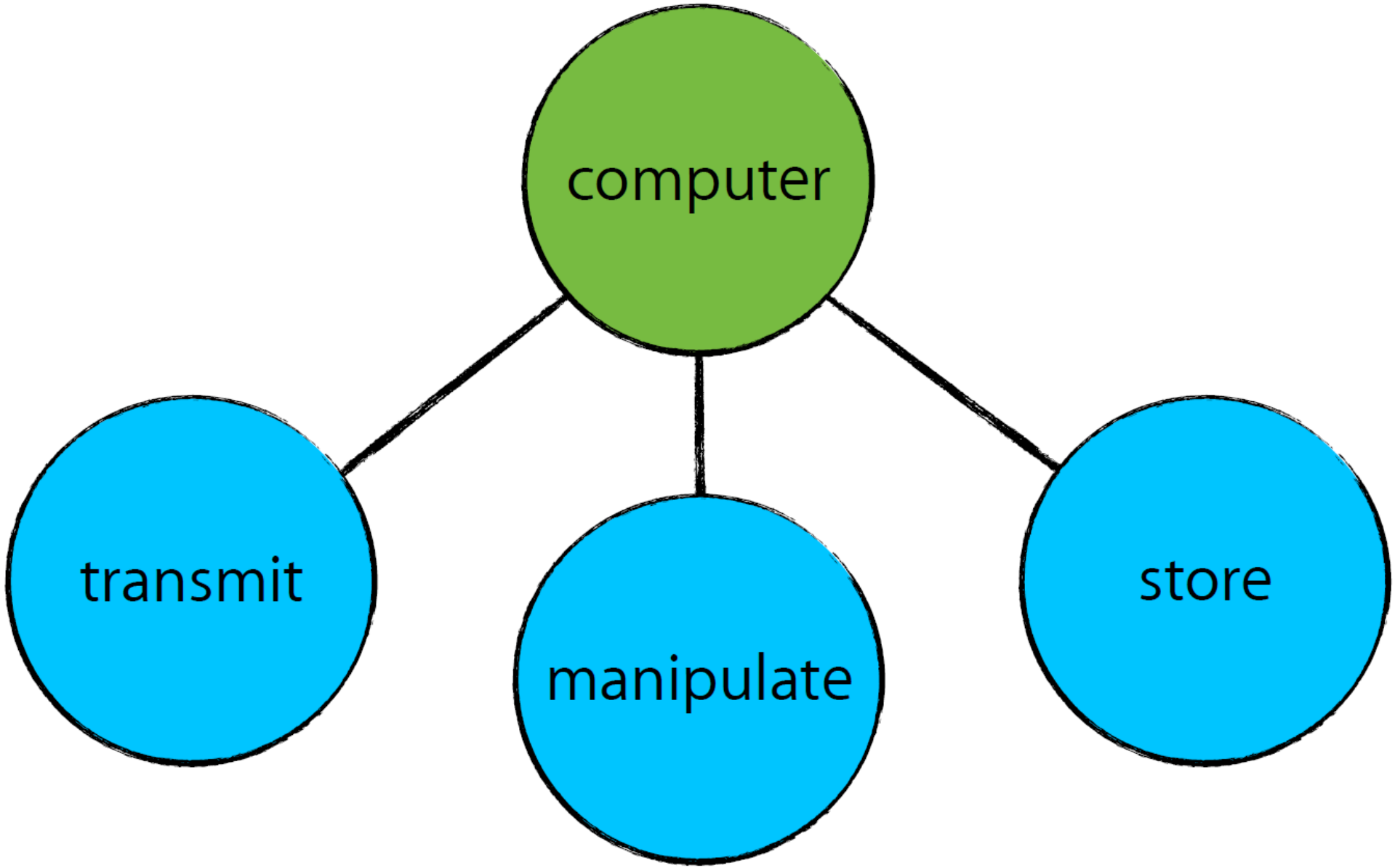




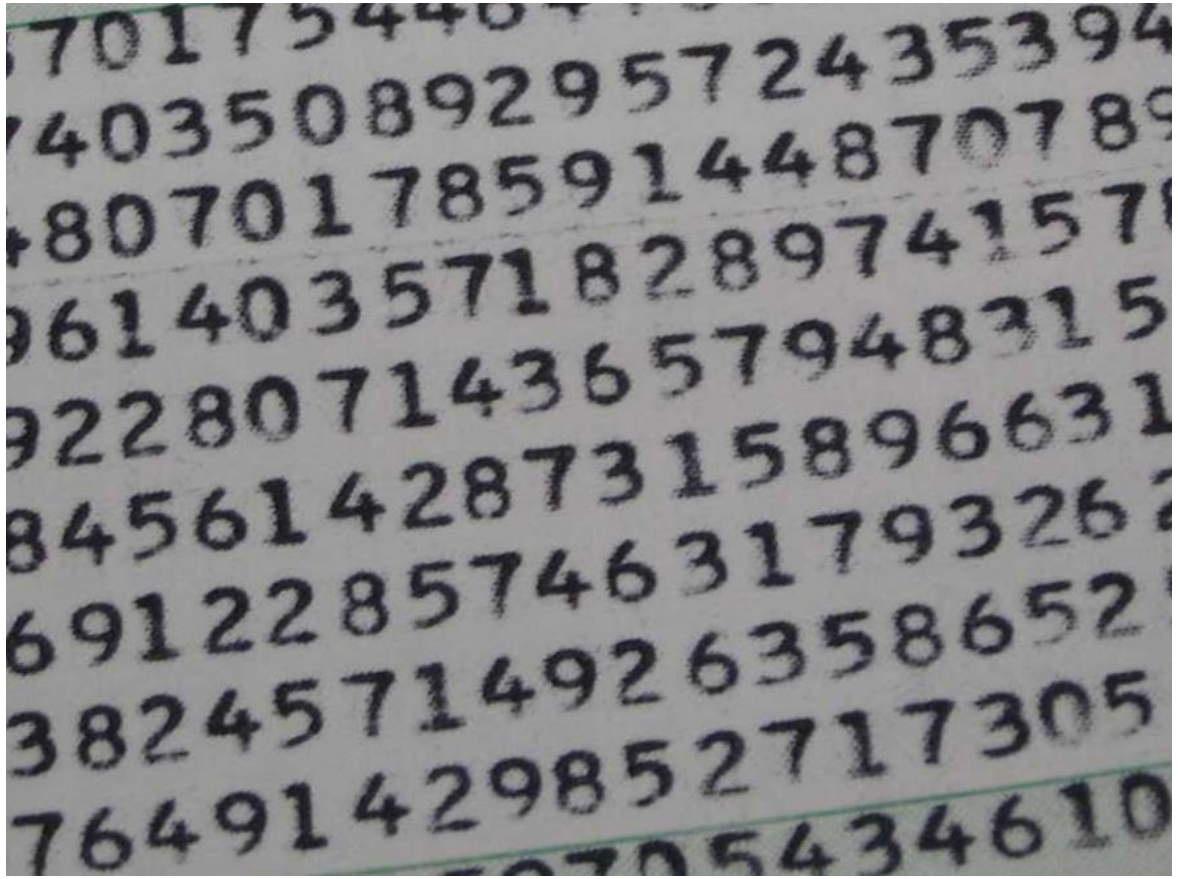




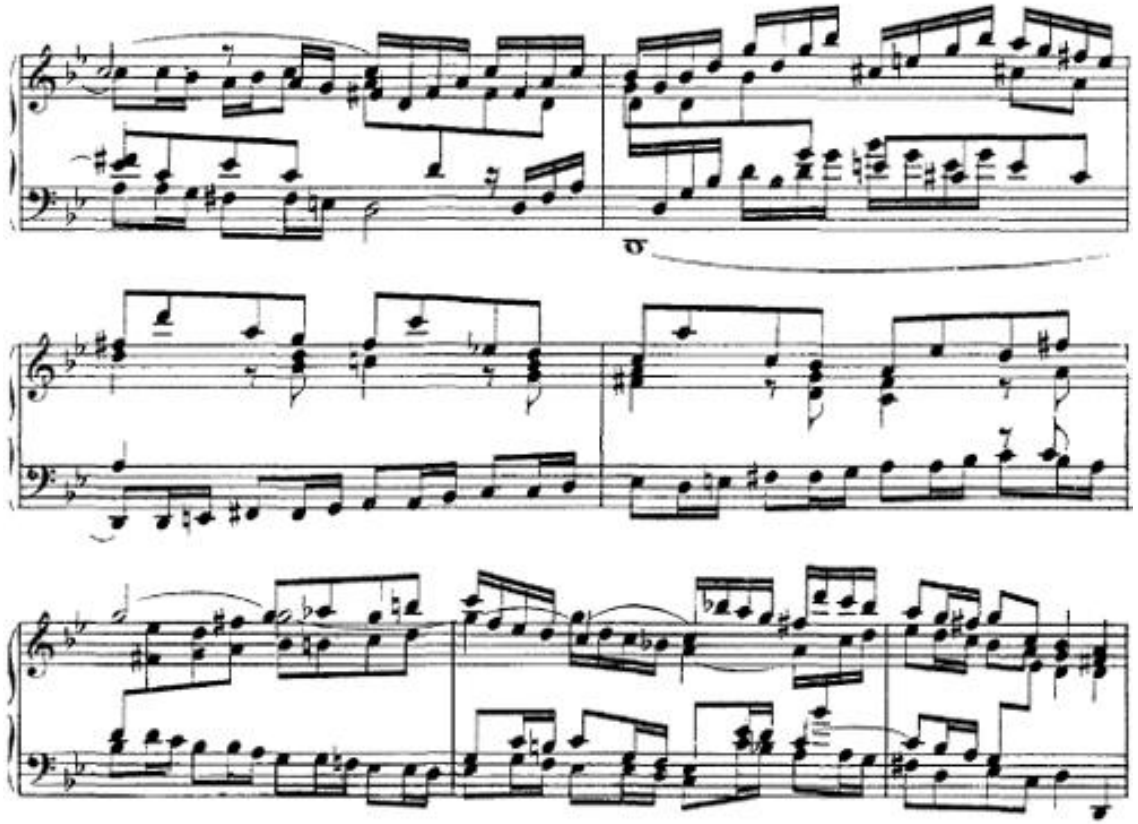




Data Sources



Data Sources



Data Sources

- As far as the physical computer system is concerned, there is no such thing as imagery, audio, text, or even numbers.
- The CPU has no idea what a JPG or MP3 is.
- At the hardware level, we deal exclusively in binary (0, 1)
- Different data types are encoded in different ways.
- A Bitmap image is encoded (in binary) differently than a JPG image, and so on.

(Even 0 and 1 are abstractions over voltage levels – 0V/GND for 0, 1.7/3.3/5.0v for 1, depending on the circuit)

Program Development

```
void applyTransformation(float *rgbToWorld, float *rotMat, float *translation)
{
    int i, j;
    float tempPoints[3];

    for (i = 0; i < WIDTH*HEIGHT; i++)
        if (rgbToWorld[i * 3 + 0] != 0 || rgbToWorld[i * 3 + 1] != 0 || rgbToWorld[i * 3 + 2] != 0) {
        matMul(rotMat, 3, 3, &rgbToWorld[i * 3], 3, 1, tempPoints);
        for (j = 0; j < 3; j++)
            rgbToWorld[i * 3 + j] = tempPoints[j] + translation[j];
        }
}

int matchORB(std::vector<cv::KeyPoint> & kp, std::vector<cv::KeyPoint> & kp_p, cv::Mat & desc, cv::Mat & desc_p, float *i2w, float *i2w_p, int *matchesUV)
{
    int totalMatches = 0;
    float relative_thresh = 0.7;
    float absolute_thresh = 5.0;

    //printf("%d %d - %d - %d %d\n", (int)kp.size(), (int)kp_p.size(

    cv::BFMatcher matcher(cv::NORM_HAMMING);
    std::vector<std::vector<cv::DMatch>> nn_matches;
    matcher.knnMatch(desc, desc_p, nn_matches, 2);

    std::vector<cv::KeyPoint> matched1, matche
    std::vector<cv::DMatch> good_matches;

    for (size_t i = 0; i < nn_matches.size();
        cv::DMatch first = nn_matches[i][0];
        float dist1 = nn_matches[i][0].distanc
        float dist2 = nn_matches[i][i].distance;

        if (dist1 < relative_thresh * dist2) {

            int u1 = kp[first.queryIdx].pt.x;
            int v1 = kp[first.queryIdx].pt.y;
            int u2 = kp_p[first.trainIdx].pt.x;
            int v2 = kp_p[first.trainIdx].pt.y;

            if (i2w[(v1*WIDTH + u1) * 3 + 2] != 0.0 || i2w[(v1*WIDTH + u1) * 3 + 1] != 0.0 || i2w[(v1*WIDTH + u1) * 3 + 0] != 0.0) {
                if (i2w_p[(v2*WIDTH + u2) * 3 + 2] != 0.0 || i2w_p[(v2*WIDTH + u2) * 3 + 1] != 0.0 || i2w_p[(v2*WIDTH + u2) * 3 + 0] != 0.0) {
                    matched1.push_back(kp[first.queryIdx]);
                    matched2.push_back(kp_p[first.trainIdx]);
                }
            }
        }
    }

    totalMatches = (int)matched1.size();
    for (int i = 0; i < totalMatches; i++)
    {
```

Program

pro·gram
/ˈprōˌgram/

noun

1. a planned series of future events, items, or performances.
   "a weekly program of films"
   synonyms: schedule, agenda, calendar, timetable; More

verb

1. provide (a computer or other machine) with coded instructions for the automatic performance of a particular task.
   "it is a simple matter to program the computer to recognize such symbols"

A series of instructions that a computer can interpret and execute.

In this course, you will write programs. **The more, the better.**

Program Development

- To transmit, manipulate, and store data, a computer needs a set of instructions called a ***program***
- These instructions are the *software* that get executed by the CPU/computer system (hardware)
- Learning to write such programs is the primary objective of this course.
- You will write programs that **transmit**, **manipulate**, and **store** data for the purpose of *solving a problem*.

Algorithm

al·go·rithm
/ˈalgəˌriT͟Həm/

noun

a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.
"a basic **algorithm for** division"

In short, a procedure to solve a problem

A program is an implementation of an algorithm

Algorithm Characteristics

- Steps must be in the correct **order**
- Operations must be **unambiguous**
- Operations must be **feasible**
- A **result** must be produced …
- … in a **finite** amount of time

Programs, Algorithms, Implementation

Don’t be confused by the jargon:

- An ***algorithm*** can be written in plain English, a ***program*** cannot.
- Thus, a program is a *translation* of an algorithm into a language that a computer can understand.
- This translation process is called ***implementation***.
- A program is an implementation of an algorithm.

Bread

Ingredients:

- 3/4 cup warm water
- 1 package active dry yeast
- 1 tsp salt
- 1-1/2 tbsp sugar
- 1 tbsp vegetable shortening
- 1/2 cup milk
- 3 cups all-purpose flour

input

Directions:

1. Add dry ingredients.
2. Mix.
3. Add water.
4. Knead.
5. Let rise.
6. Bake.

algorithm

output

Bread

Ingredients:

3/4 cup warm water
1 package active dry yeast
1 tsp salt
1-1/2 tbsp sugar
1 tbsp vegetable shortening
1/2 cup milk
3 cups all-purpose flour

Directions:

1. Add dry ingredients.
2. Mix.
3. Add water.
4. Knead.
5. Let rise.
6. Bake.

Is this a good algorithm?

- Are the steps unambiguous?
- If you've never made bread before, could you succeed, given these instructions?

Propose an Algorithm:

Find the smallest number in the following list:

6 1 13 7 11 9 2 8

How would you, a human, do this?

Are your steps unambiguous?

- ***Find the smallest number*** is not a feasible operation.
- There is no single instruction for this.
- Instead, we build this operation from more primitive instructions that ***are*** feasible (comparisons, etc.)

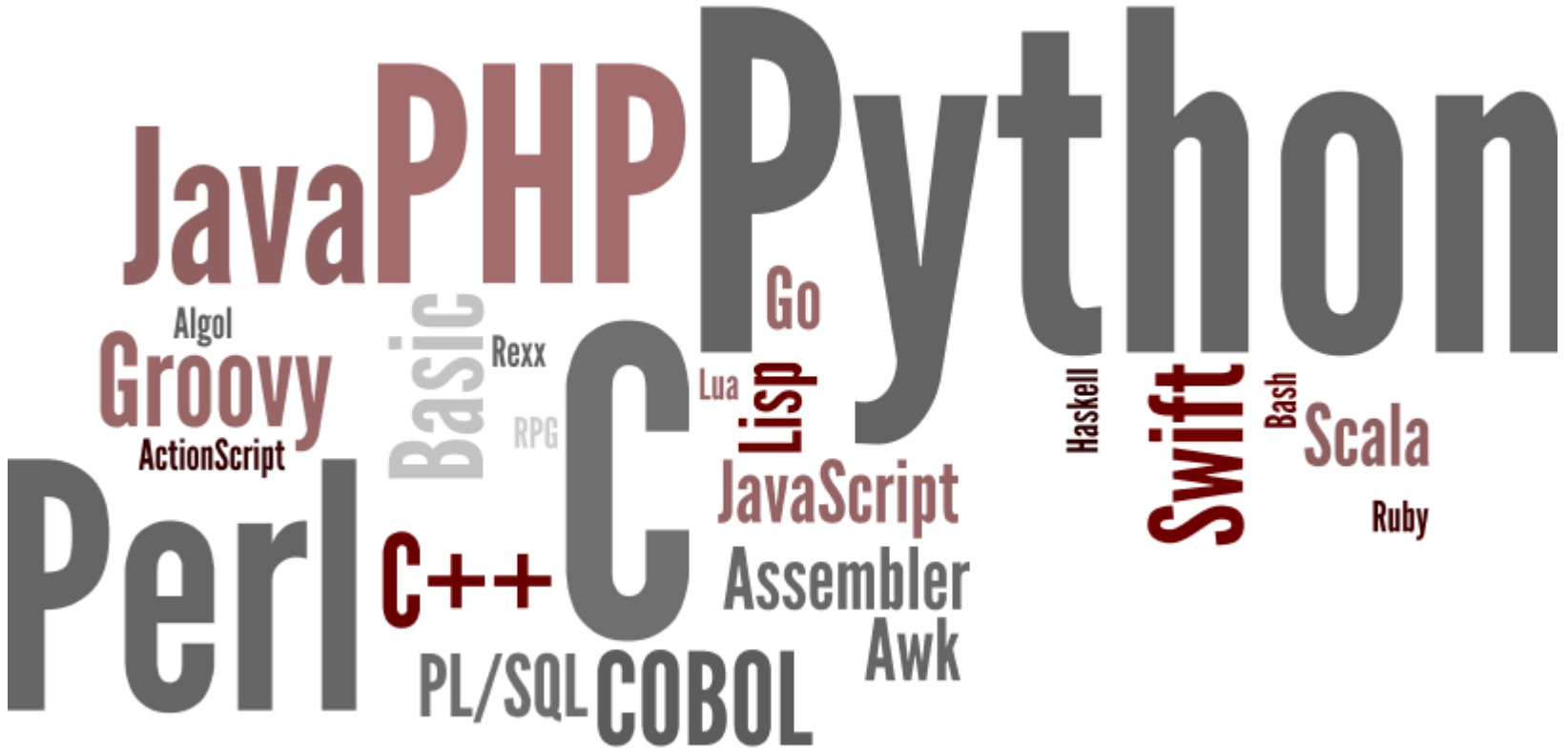
A Slight Adjustment…

Sort the number in the following list:

6 1 13 7 11 9 2 8

- A “good” algorithm makes implementation easy.
- A “bad” algorithm makes implementation impossible.

Programming Languages



Programming Languages

1) Machine code (pure binary): Tedious to program, tied to computer architecture. This is what the CPU executes.

```
01010100 01101000 01101001 01110011
00100000 01101001 01110011 00100000
01110100 01101000 01100101 00100000
01110100 01110101 01110100 01101111
01110010 01101001 01100001 01101100
00100000 01110100 01101111 00100000
```

2) Assembly Language: Less tedious, still tied to computer architecture. Assembly converts unambiguously to binary

```
movl $0xFF001122, %eax
addl %ecx, %edx
xorl %esi, %esi
pushl %ebx
```

3) High Level Languages (Java, **C**, Python, etc.): Easier to understand, writing code is far more efficient.

```
#include <stdio.h>

int main (void)
{
     printf ("Hello, world!\n");

     return (0);
}
```

Programming Languages

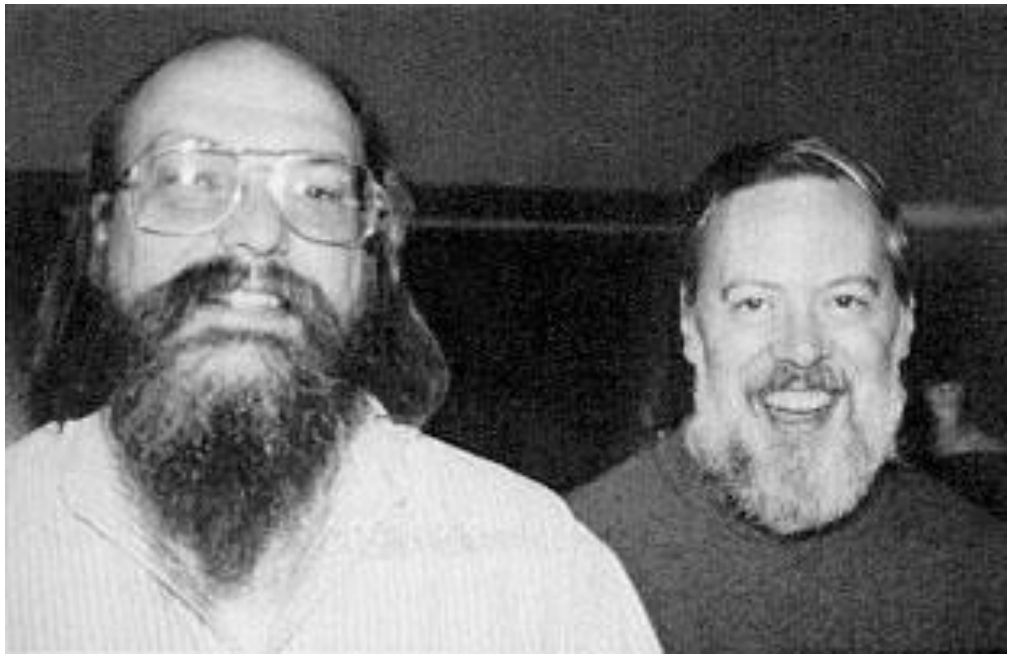
Compile

01010100 01101000 01101001 01110011
00100000 01101001 01110011 00100000
01110100 01101000 01100101 00100000
01110100 01110101 01110100 01101111
01110010 01101001 01100001 01101100
00100000 01110100 01101111 00100000

Different HLLs compile to the same architecture-specific machine language.

(*This is a simplification, but a reasonable and helpful one*)

What, Why, When, Who?



- Dennis Ritchie & Ken Thompson
- Bell Labs, 1972-1973
- Wanted to write utilities for Unix
- This was early Unix, being developed in Assembly at the time.
- Later, C was used to re-implement the entire Unix kernel.
- One of the first kernels implemented in something *other* than Assembly!

What, Why, When, Who?

C was one of the early ***general-purpose*** programming languages.

Meaning: it wasn't developed with a specific application domain in mind

FORTRAN – **FOR**mula **TRAN**slation
- Turn mathematical formulas into code

COBOL – **CO**mmon **B**usiness **O**riented **L**anguage
- Designed for business, finance use

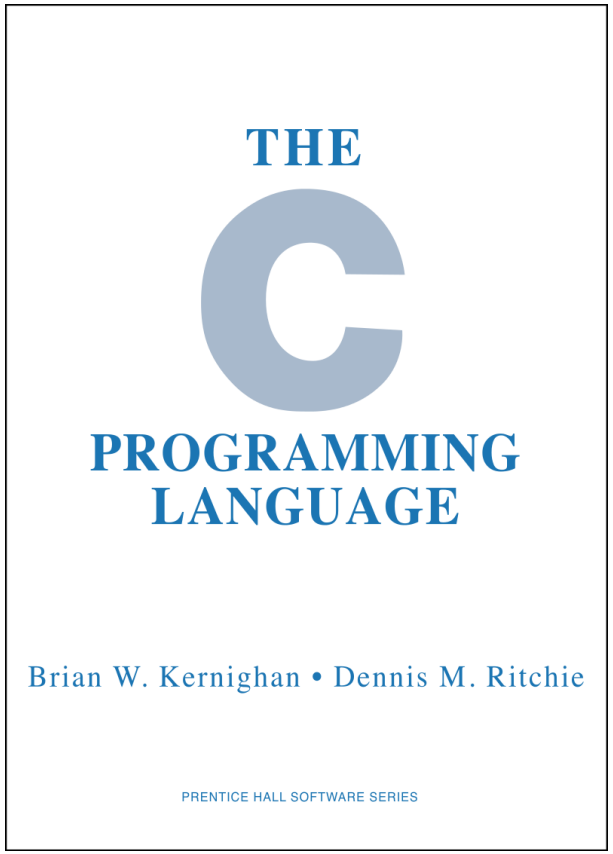
ALGOL – Meant for algorithm description

What, Why, When, Who?

Why teach C?

- C is relatively *small* (32 reserved words)
- C is *common*, it's everywhere
- C is *stable* (well established language, doesn't change much anymore)
- C is *efficient* at runtime
- C is the *basis for other languages* (e.g. C++)
- C is (relatively) *easy to learn*.

What, Why, When, Who?



The original C manual…

```
#include <stdio.h>

int main (void)
{
     printf ("Hello, world!\n");

     return (0);
}
```

…was the first to say hello to the world.

Hello, world!

```c
#include <stdio.h>
int main(void)
{
   printf("Hello, world!\n");
   return (0);
}
```

Questions?