Massachusetts Institute of Technology

Boğaziçi University

Introduction to Computing
Programming with R

Introduction

Suayb S. Arslan
BCS@MIT, Comp. Science@BU

2024

Class Policy
Syllabus

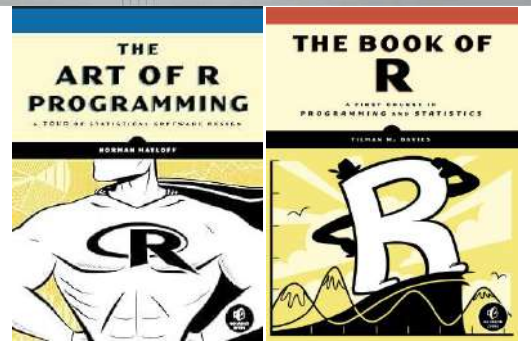
Class Intructors

- Şuayb Ş. Arslan (lectures) suayb.arslan@boun.edu.tr
- Serhat Çevikel (labs & PS) serhat.cevikel@boun.edu.tr
- Ömer Eren (labs & PS) omer.eren@boun.edu.tr
- Nevzat Ersoy (labs & PS) nevzat.ersoy@boun.edu.tr
- Lectures: WW56
- PS: F6 (BM4)
- Lab: FF78 (BM4)

Course Objectives

By the end of this course you will be able to

- apply structured programming concepts,
- write R programs using basic programming structures,
- find and correct errors in your programs,
- write R programs for descriptive analysis of data,
- visualize your data with built-in R graphics.



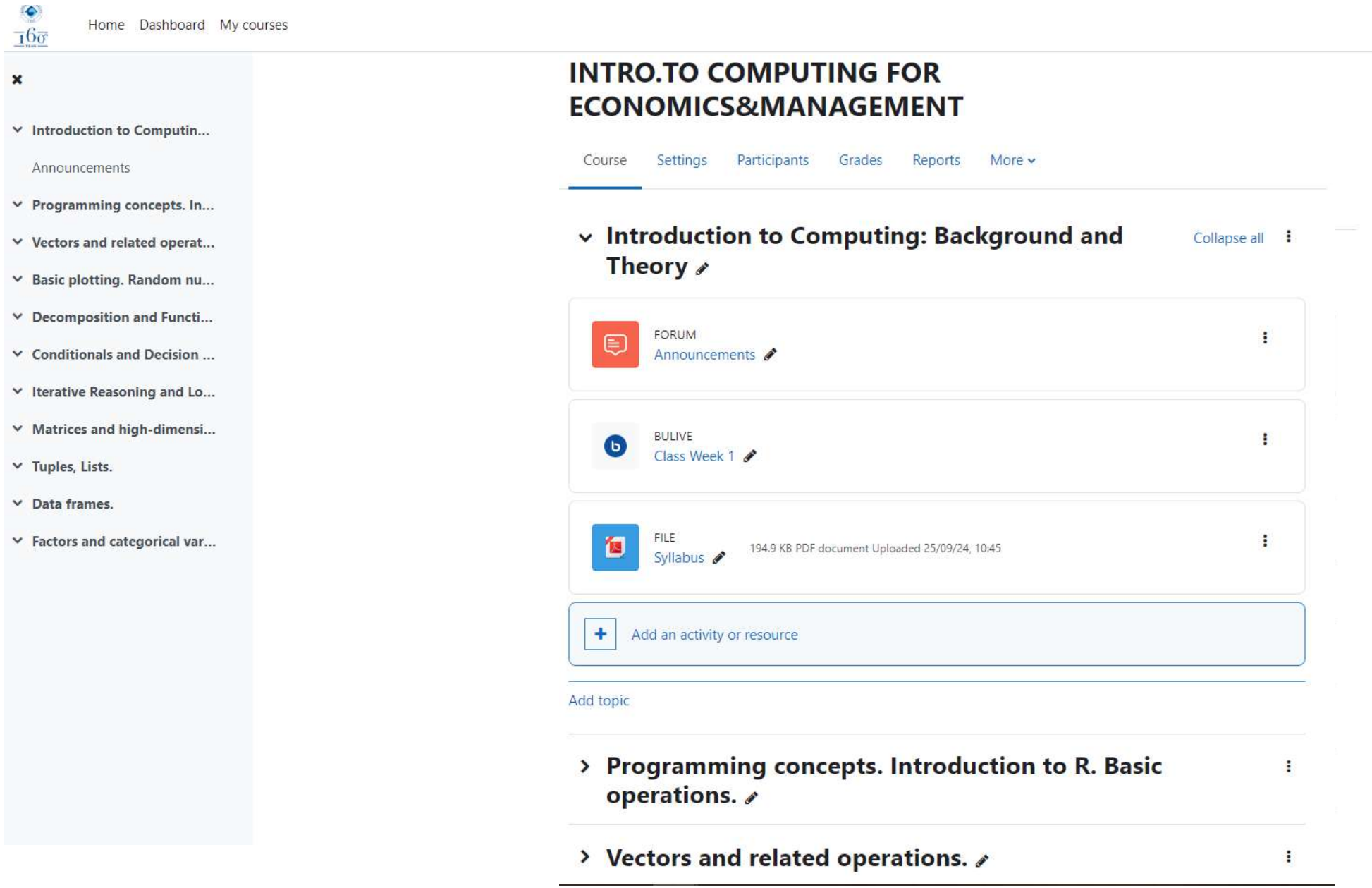
Textbooks

- Norman Matloff, *The Art of R Programming*. No Starch Press, 2011.
- Tilman M. Davies, *The Book of R*. No Starch Press, 2016.

- GitHub repository of Jupyter notebooks used in lectures and PS (can be viewed online): https://github.com/suaybarslan/CMPE140
- **First week slides, syllabus (also uploaded to Moodle), and other materials:**
  - https://www.dropbox.com/scl/fo/29h9isqgpjpm9ch0lfjz9/AGbQHjNyeXL9otRcC17PLXQ?rlkey=b8f2vgpeucxu5ymvbx9w7kr8u&st=l3uos1am&dl=0

Moodle

sporadic updates



Tentative Lecture Schedule

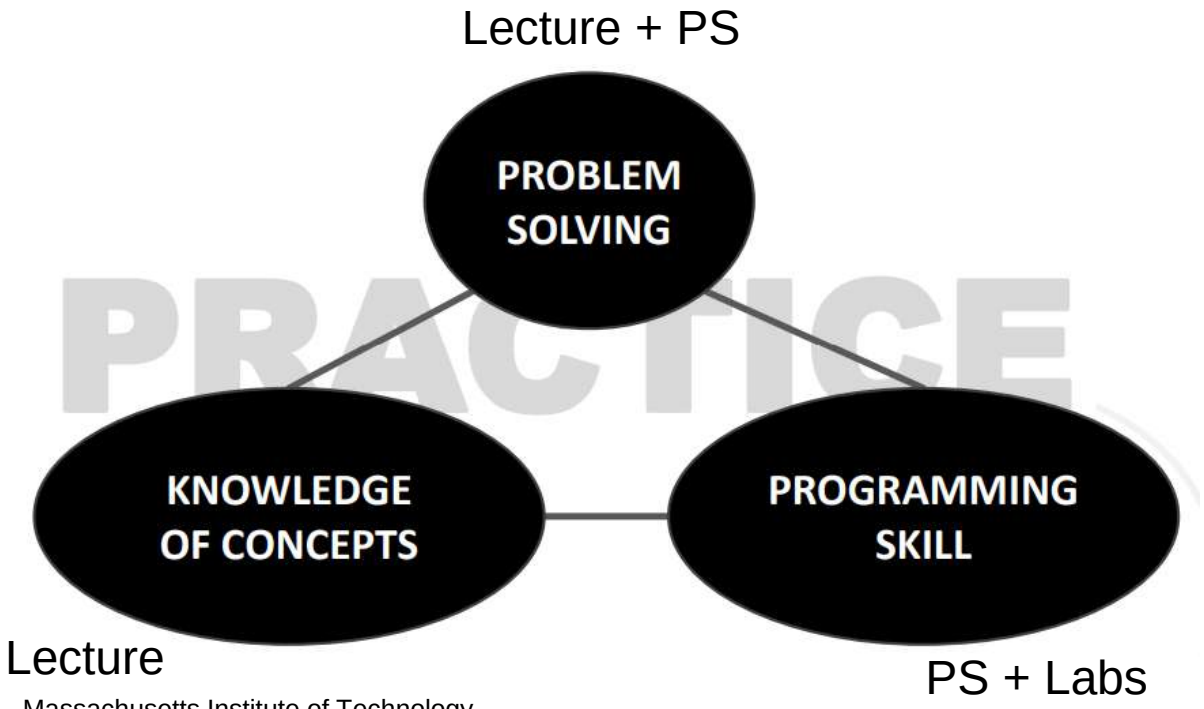
Grading

Week	Lecture date	PS+Lab date	Subject
1	**25.09.2024**	**27.09.2021**	**Introduction to Computing: Background and Theory**
2	02.10.2024	04.10.2024	Programming concepts. Introduction to R. Basic operations.
3	09.10.2024	11.10.2024	Vectors and related operations.
4	16.10.2024	18.10.2024	Basic plotting. Random number generation and simulation.
5	23.10.2024	25.10.2024	Decomposition and Functions (Testing and Debugging)
6	30.10.2024	01.11.2024	Conditionals and Decision structures
7	06.11.2024	08.11.2024	Iterative Reasoning and Loops.
8	13.11.2024	15.11.2024	Matrices and high-dimensional operations **Midterm**
9	20.11.2024	22.11.2024	Tuples, Lists.
10	27.11.2024	29.11.2024	Data frames.
11	04.12.2024	06.12.2024	Factors and categorical variables.
12	11.12.2024	13.12.2024	Data import. Built-in data sets. More plotting.
13	18.12.2024	20.12.2024	Recursions
14	25.12.2024	27.12.2024	Correlation and regression

Grading will be based on **seven** bi-weekly **quizzes** (5% each), one **Midterm** (25%), one **Final** exam (30%), and **attendance** (10%). Submitted exams will be graded semi-automatically.

New to Programming?

- **Recipe:** PRACTICE. PRACTICE? PRACTICE!
  - can't **passively absorb** programming as a skill
  - **Download & study code** before lecture and follow along
  - do finger/typing exercises (yourself)
  - don't be afraid to try out R commands on your local!



Why do we need computation?

Why do we need computation?

- **Solve Complex Problems:** Computation allows us to model, simulate, and solve problems that are too complex for manual calculations, such as climate modeling or molecular simulations. (Everyday Accounting)
- **Automate Repetitive Tasks:** It enables the automation of routine tasks, improving efficiency and freeing up time for more creative or critical thinking. (Efficiency&Productivitiy)
- **Data Processing and Analysis:** Computation is essential for analyzing large datasets, making sense of information, and deriving insights in fields like healthcare, finance, and AI.
- **Enable Technological Innovation:** Advances in computation power drive progress in artificial intelligence, robotics, space exploration, and many other cutting-edge fields. (Scientific&Tech Progress)
- **Enhance Decision Making:** By processing vast amounts of data, computation helps in making more informed, accurate, and timely decisions across industries.

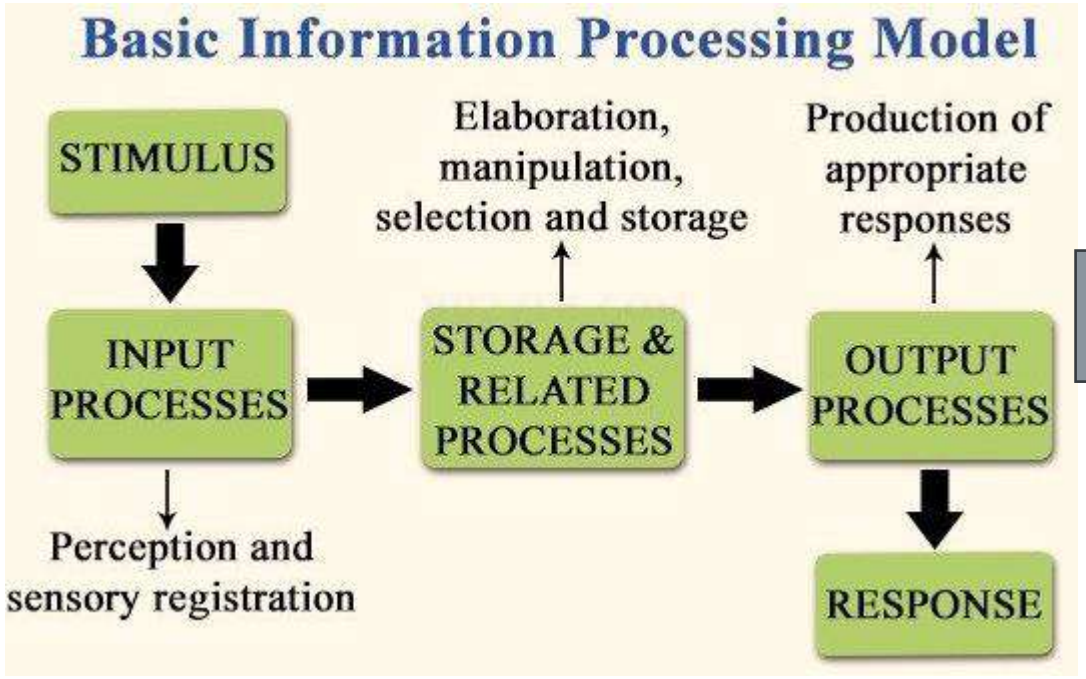
Computers

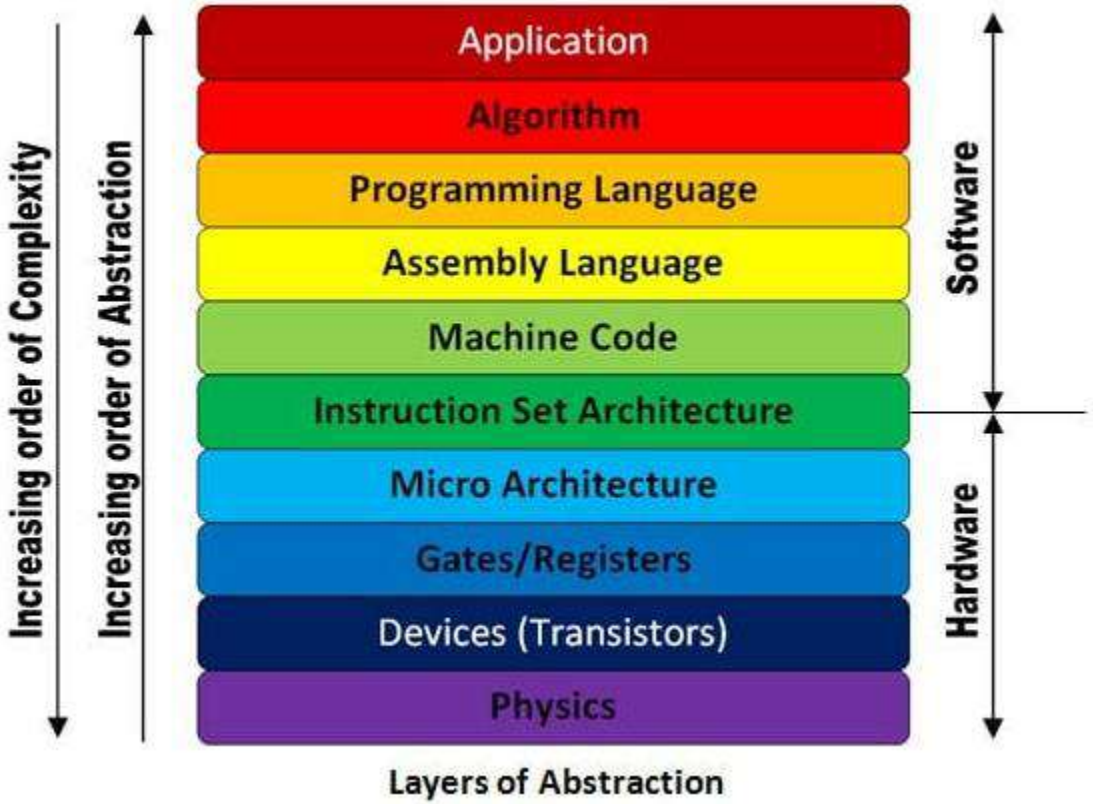
What are they?

- Fundamentally a data processing machine:
  - performs calculations (CPU, GPU): over billion calculations per second!
  - remembers results (data storage): 100s of gigabytes of storage!
- What kinds of calculations?
  - Typically built-in to the language.
  - Ones that you define as the programmer.
- Computers only know what you tell them.
- (1) Fixed program computer
  - calculator
- (2) Stored program computer
  - machine stores and executes instructions

Data Processing

Idea for computers

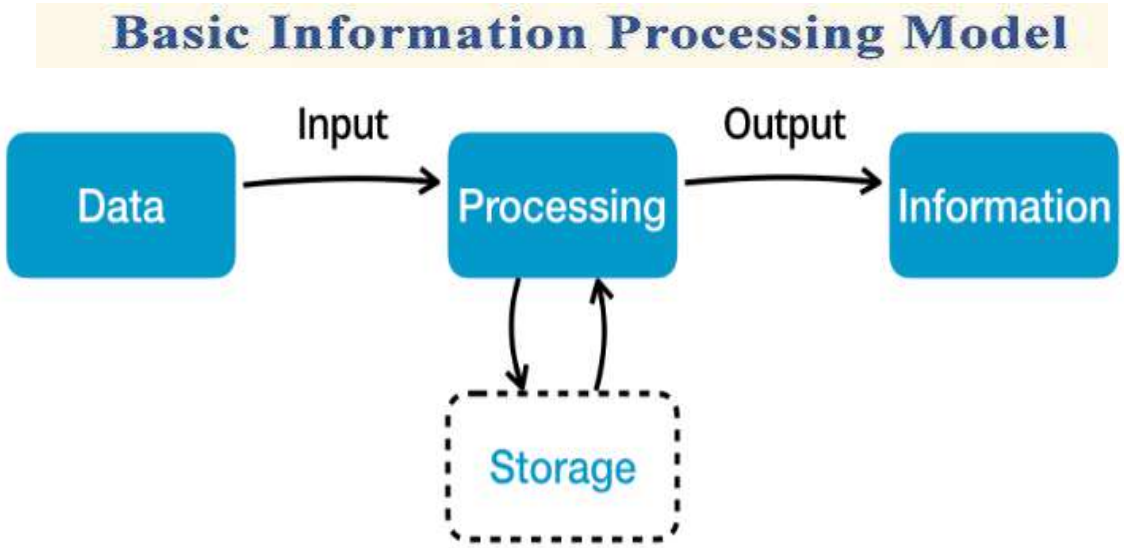




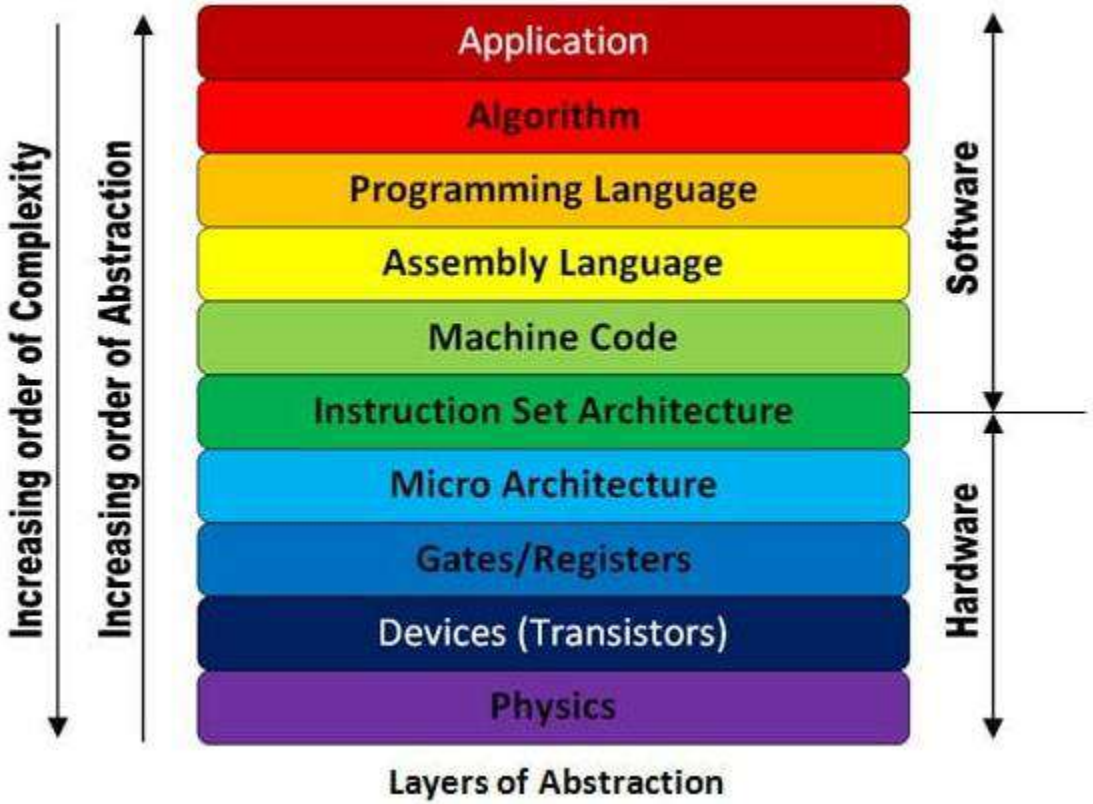
- Computers are natural consequence of human evolution.
- Humans process data for information almost constantly.

Data Processing

Idea for computers



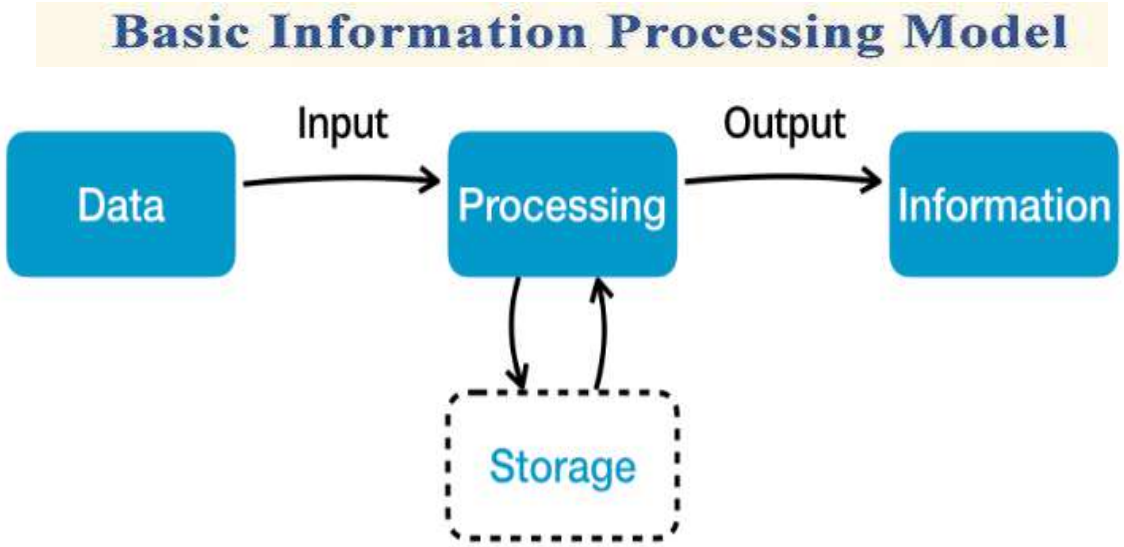
SIMPLIFIED

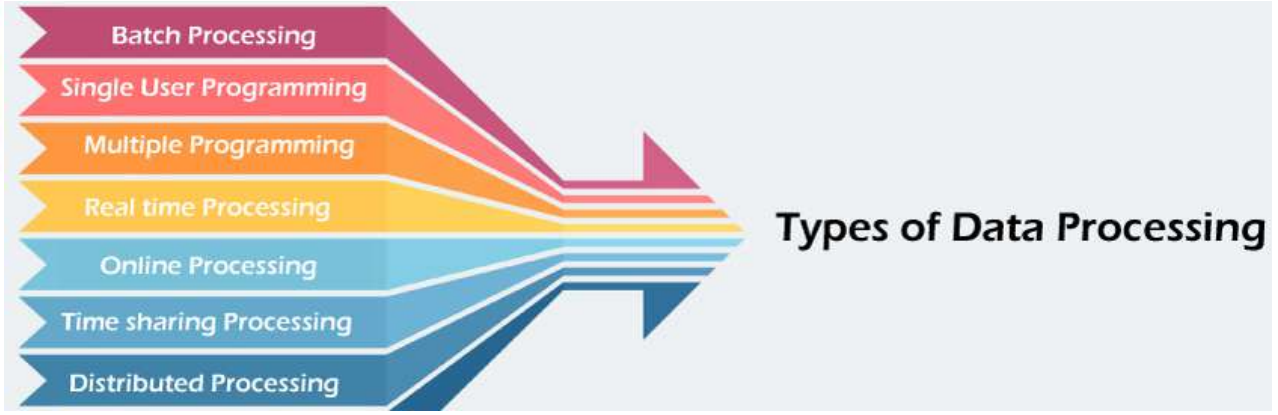


- Computers are natural consequence of human evolution.
- Humans process data for information almost constantly.

Data Processing

Idea for computers



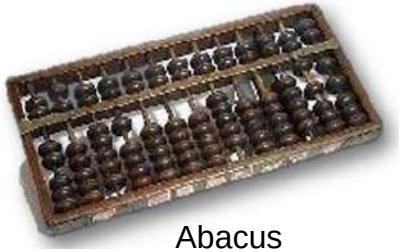


SIMPLIFIED

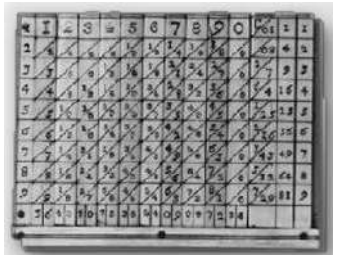
- Different kinds of processing is preferred given the resources and application requirements.

Computers
History

Vacuum tubes



Abacus

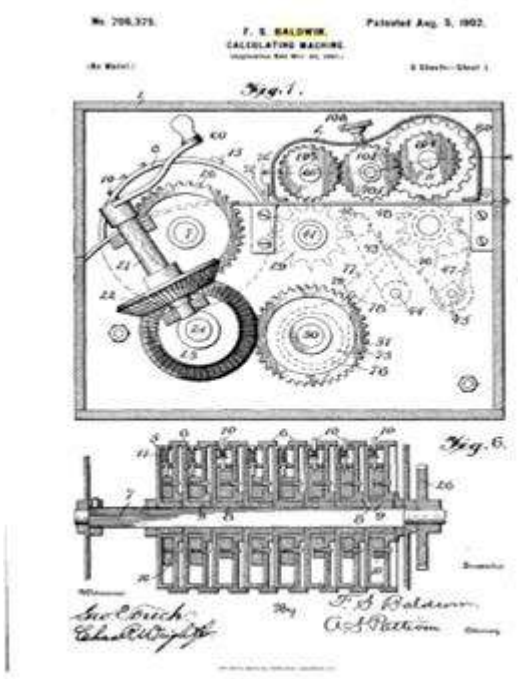


Logarithm Table

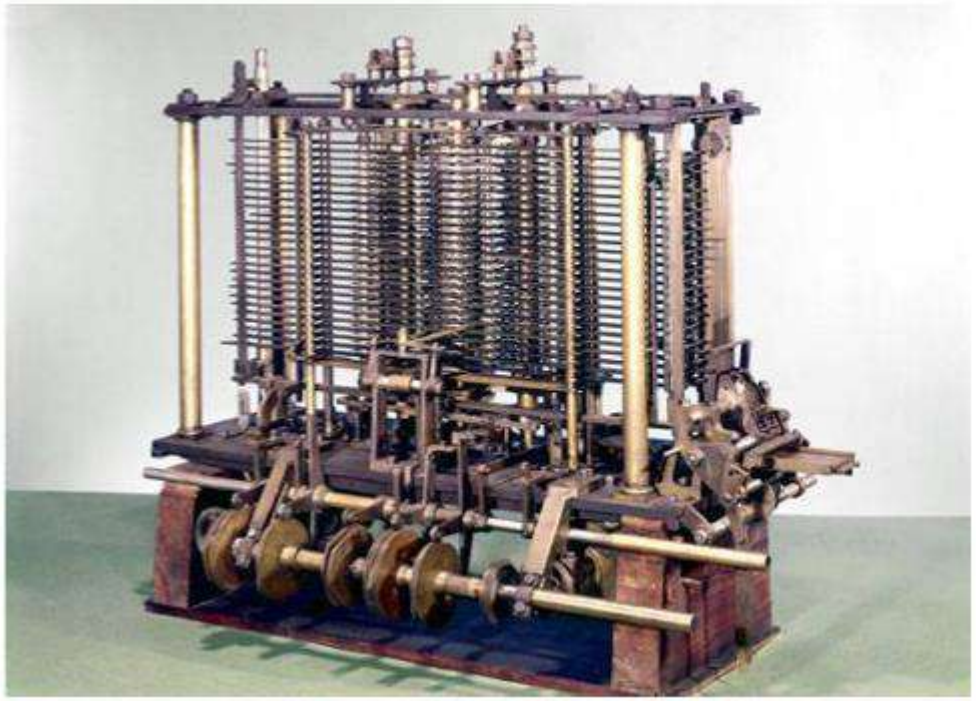
Intel 4004
Microprocessor

- **Need for calculation is as old as human mind.**
  - The **abacus** was used in Babylons 2000 years before the Greeks used it to help with calculating.
  - First **calculator** (1645) by Blaise Pascal.
  - Leibnitz invented a machine ("**Stepped Reckoner**") in 1674, around 30 years after Pascal invented his machine.
  - Joseph-Marie Jacquard (1804) adapted the use of **punched cards**, a model as input and output of data in the electromechanical and electronic computing industry (before the birth of microprocessors).
  - Charles Babbage designed the "**Difference Engine**" and "**Analytical Engine**" in the early 19th Century.
  - The worlds first programmable, electronic, digital computer, **Colossus**, was developed by the British codebreaker Tommy Flowers during the later part of the Second World War to help in the 'cryptanalysis' of the Lorenz cipher.
  - Colossus used **vacuum tubes (equivalent of transistors)** to perform logic and counting operations and was programmed using switches and plugs and not by a programmer typing at a keyboard.
  - Alan Turing deviced a machine capable of processing a stream of 1s and 0s (binary) according to programmed instructions would be capable of solving any problem (**theoretical foundations**).
  - The first **transistor** was successfully demonstrated on December 23, 1947, at Bell Laboratories.
  - First commercially available **microprocessor** was the **Intel 4004**, designed by Federico Faggin and introduced in 1971.

Difference & Analytical Engines by
Charles Babbage



- Difference Engine

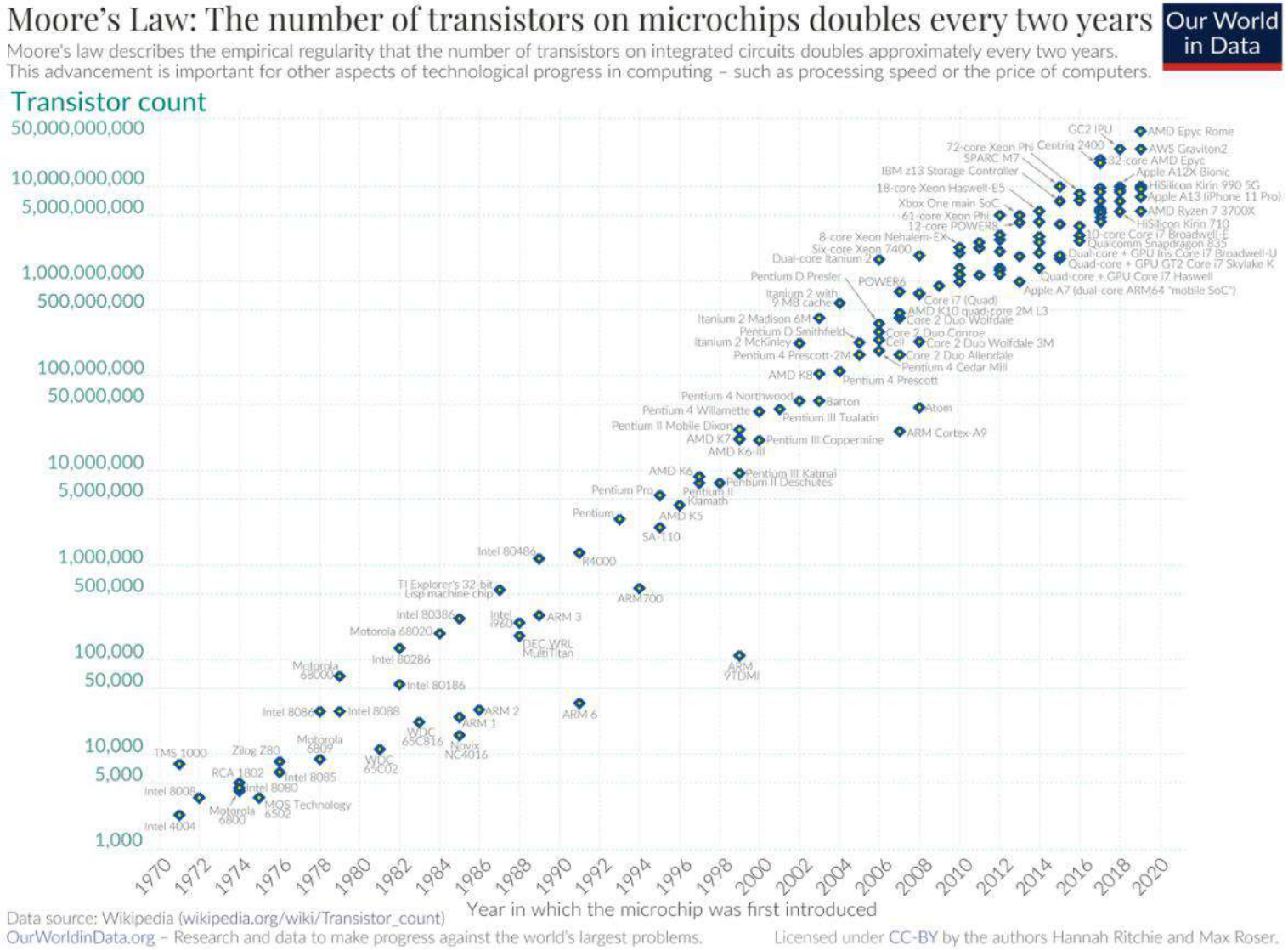


- Analytical Engine

Moore's law
Limitations

- Gordon Moore (the co-founder of Intel Corporation)
- Moore's law (in 1965) states that

  "Over the history of computing, the number of transistors on integrated circuits doubles (exponential increase) approximately every two years."

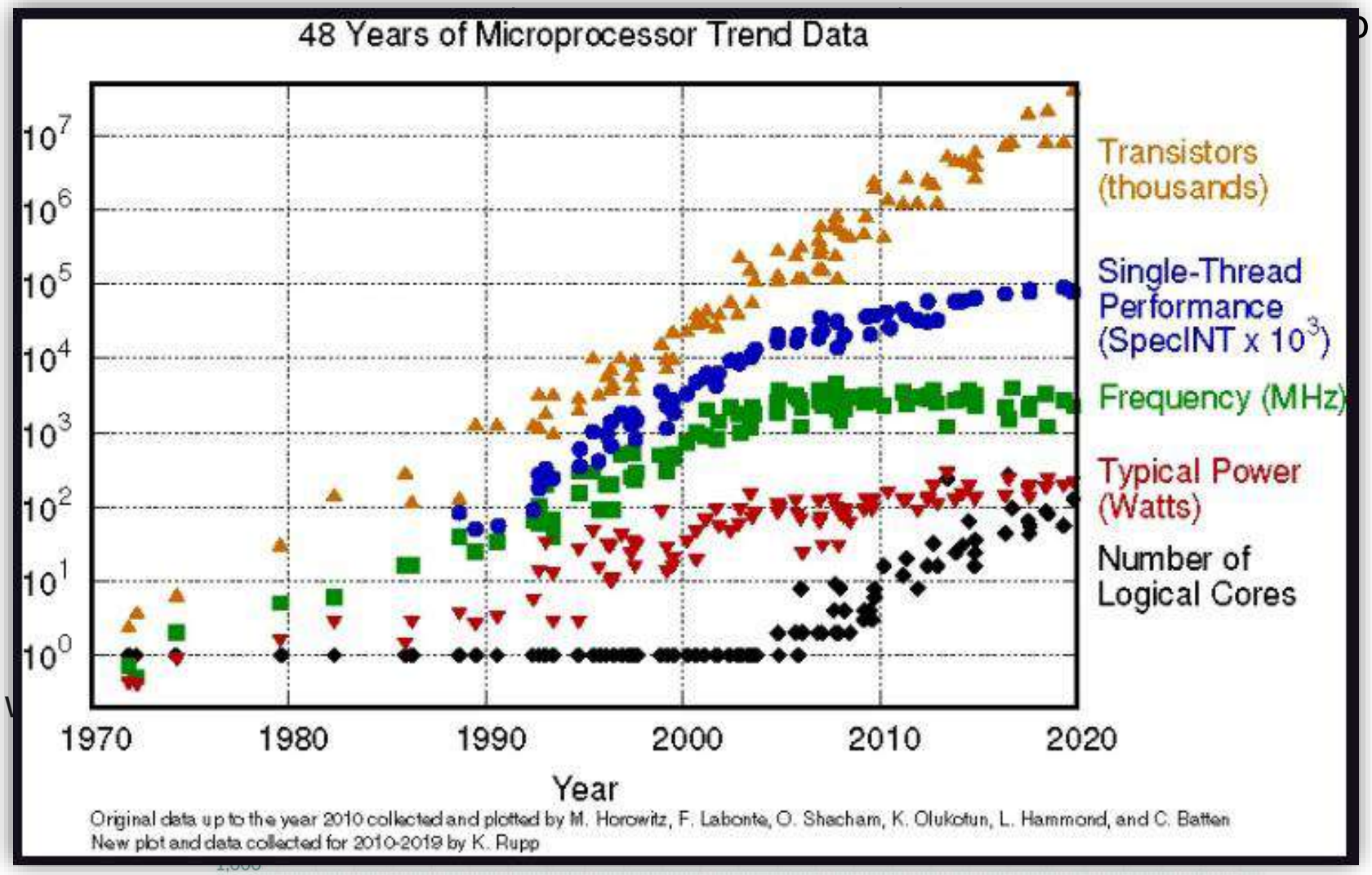


transistors eventually would reach the limits of miniaturization at atomic levels….

Moore's law
Limitations

- Gordon Moore (the co-founder of Intel Corporation)
- Moore's law (in 1965) states that

"Over the history of computing, the number of transistors on integrated
o years."



transistors eventually
reach the limits of
miniaturization
at atomic levels….

1970 1972 1974 1976 1978 1980 1982 1984 1986 1988 1990 1992 1994 1996 1998 2000 2002 2004 2006 2008 2010 2012 2014 2016 2018 2020
Year in which the microchip was first introduced

Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
OurWorldinData.org – Research and data to make progress against the world's largest problems.

Where is this going?
Evolution of Computing

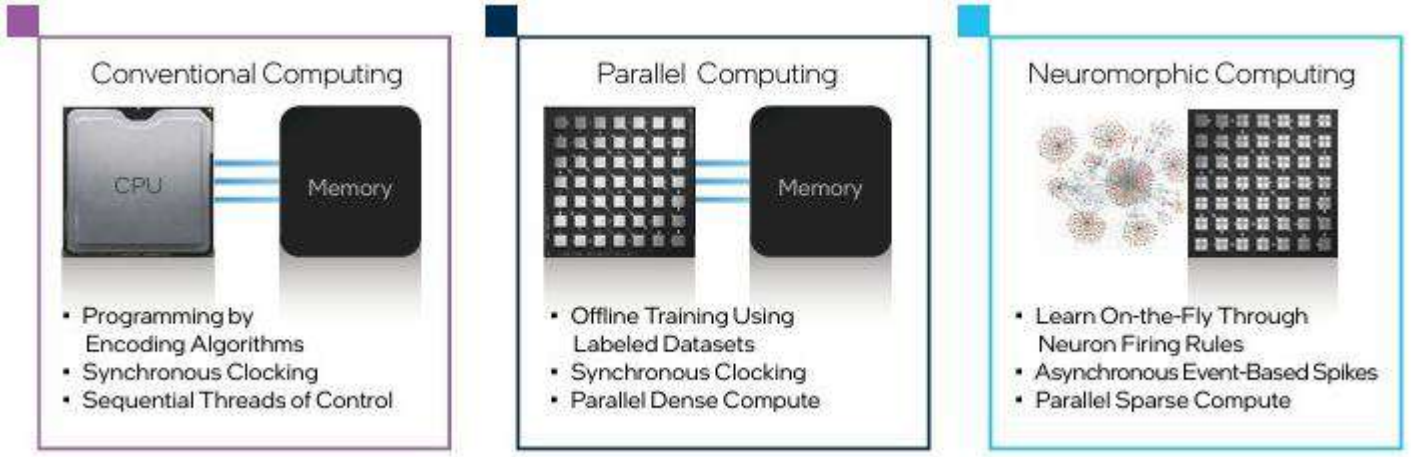
- Parallelization is inevitable.
- GPUs are the new platforms for implementation
  - Problem: Unimaginable power requirements. Likely to be powered by Nuclear power in the near future.
  - Nvidia’s top-end “Blackwell” GPU accelerators now pushing up to 1,200 watts.
- **Neuromorphic system** act more like real neurons used in real brains and also burn orders of magnitude less power.

Fun Fact

According to research, the power consumption of **human neural networks** is somewhere around 20W.

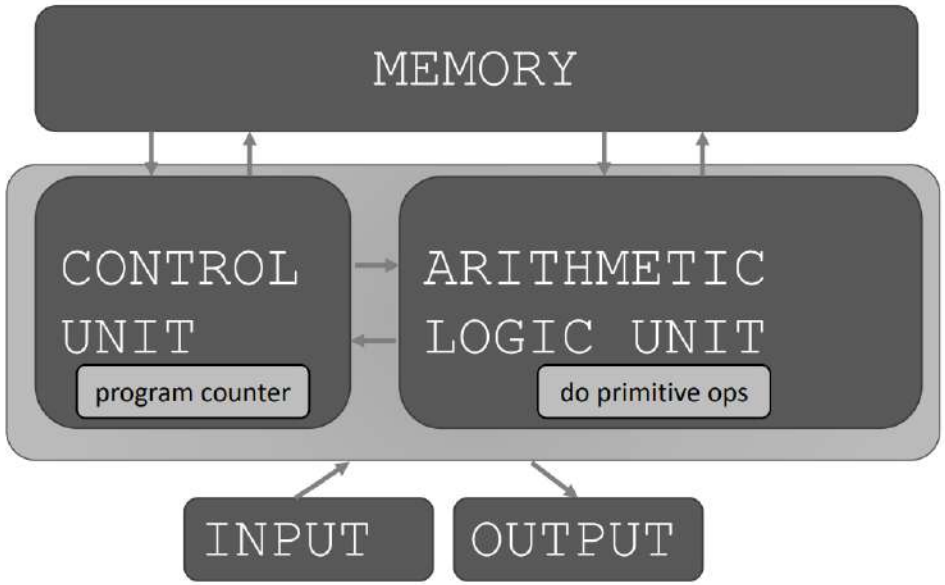
•Facebook's 65B LLaMA trained for 21 days on 2048 Nvidia A100 GPUs. At $3.93/hr on GCP, that's a total of ~$4M.

•~ 552,000 kilowatt hours total.



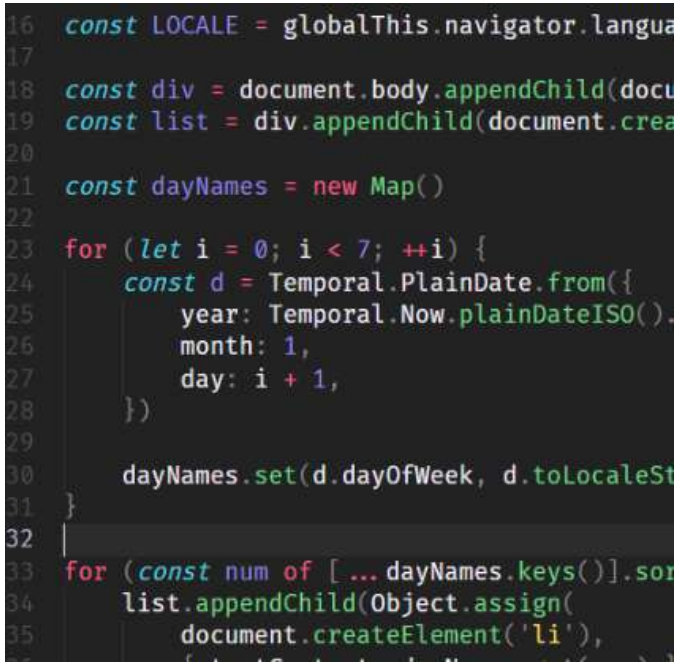
Machine Architecture

How to execute the program?



- Sequence of instructions stored inside computer
  - built from predefined set of primitive instructions
    - 1) arithmetic and logic
    - 2) simple tests
    - 3) moving data
- Special program (interpreter) executes each
- Instruction in order
  - use tests to change flow of control through sequence
  - stop when done

Primitives – Expressions
Building a recipe



```
const LOCALE = globalThis.navigator.langua

const div = document.body.appendChild(docu
const list = div.appendChild(document.crea

const dayNames = new Map()

for (let i = 0; i < 7; ++i) {
    const d = Temporal.PlainDate.from({
        year: Temporal.Now.plainDateISO().
        month: 1,
        day: i + 1,
    })

    dayNames.set(d.dayOfWeek, d.toLocaleSt
}

for (const num of [...dayNames.keys()].sor
    list.appendChild(Object.assign(
        document.createElement('li'),
```

- Turing showed that you can compute anything using 6 primitives.
  - **Move Left (L):** Move the machine's tape head one position to the left.
  - **Move Right (R):** Move the tape head one position to the right.Read Symbol:
  - Read the symbol currently under the tape head.
  - **Write Symbol:** Write a symbol on the tape at the current position.
  - **Change State:** Change the internal state of the machine.
  - **Halt:** Stop the machine when the computation is complete.
- Modern programming languages have more convenient set of primitives
- Use of abstract methods to create new primitives
- **Proposition:** Anything computable in one language is computable in any other programming language.
- A programming language (e.g. R) provides a set of primitive operations
  - expressions are complex but legal combinations of primitives in a programming language.
  - expressions and computations have values and meanings in a programming language.

Example

How to solve a problem?

- **Definition:** Square root of a number x is y such that y*y = x
- **Recipe** for deducing square root of a number x (16)
  - 1) Start with a guess, g
  - 2) If g*g is close enough to x, stop and say g is the answer
  - 3) Otherwise make a new guess by averaging g and x/g.
  - 4) Using the new guess, repeat process until close enough.
- Recipe (algorithm) is a sequence of simple steps, flow of control process that specifies when each step is executed, a means of determining when to stop.

g	g*g	x/g	(g+x/g)/2
3	9	16/3	4.17
4.17	17.36	3.837	4.0035
4.0035	16.0277	3.997	4.000002

Aspects of Languages

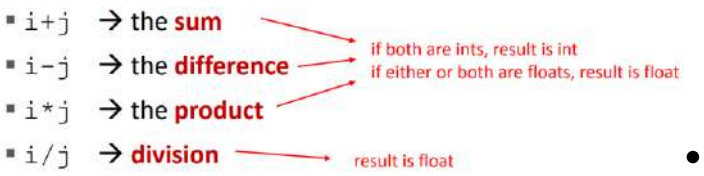
Features

- Primitive constructs. English: words. Programming language: numbers, strings, simple operators
- syntax
  - **English**: "cat dog boy" → not syntactically valid
    "cat hugs boy" → syntactically valid
  - **programming language**: "hi"5 → not syntactically valid
    3.2*5 → syntactically valid
- static semantics is which syntactically valid strings have meaning.
  - **English**: "I are hungry" → syntactically valid but there is a static semantic error.
  - **programming language**: 3.2*5 → syntactically valid
    3+"hi" → static semantic error
- semantics is the meaning associated with a syntactically correct string of symbols with no static semantic errors
  - **English**: can have many meanings: "Flying planes can be dangerous".
  - **programming languages**: have only one meaning but may not be what programmer intended.

R Programming

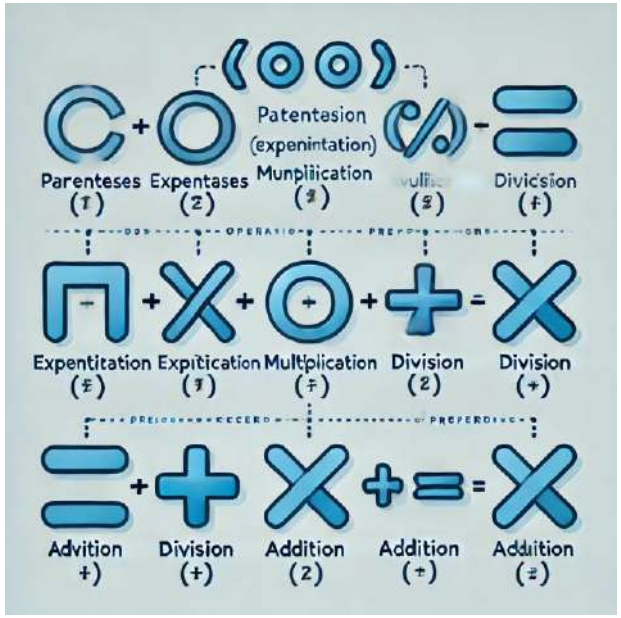
Definitions to Operations

- A **program** is a sequence of definitions and commands
  - definitions evaluated
  - commands executed by R interpreter in a shell
- **commands** (a.k.a. statements) instruct interpreter to do something
- **programs** can be typed directly in a shell or stored in a file that is read into the shell and evaluated.
  - In problem session, you will be introduced to Rstudio and R interpreter.
- **programs** manipulate data **objects.**
- **objects** have a **type** that defines the kinds of things programs can do to them
  - Ana is a human so she can walk, speak English, etc.
- **objects** are
  - scalar (cannot be subdivided – e.g. int, float, bool, etc.).
  - non-scalar (have internal structure that can be accessed)
- combine **objects** and **operators (+,-,/, etc.)** to form expressions. An **expression** has a value, which has a type
  - syntax for a simple expression: <object> <operator> <object>



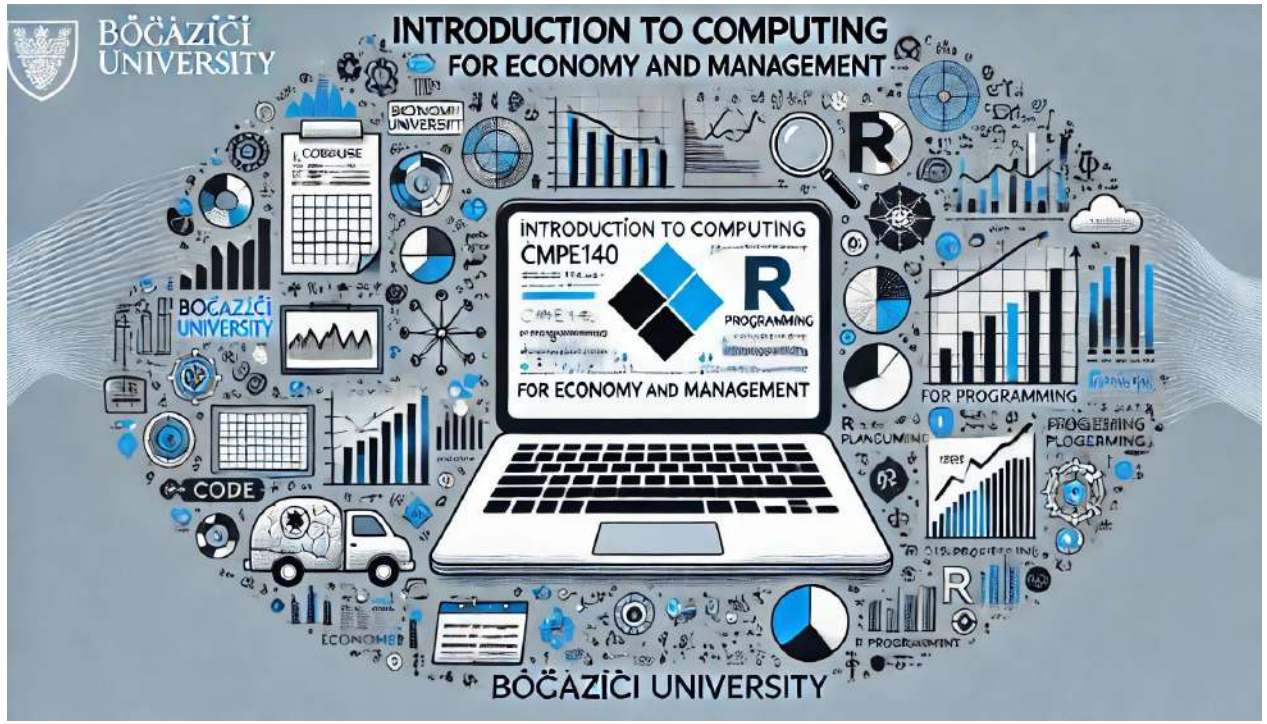
Basic Operations

Precedence



- Parentheses used to tell R to do these operations first
- **Operator precedence** without parentheses
  - **
  - *
  - /
  - + and – executed left to right, as appear in expression
- Example: 2 + 6 * 5?
- equal sign (=) is an assignment of a value to a variable name
- **why give names to values of expressions?**
  - to reuse names instead of values
  - easier to change code later
    - pi = 3.14159
    - radius = 2.2
    - area = pi*(radius**2)

References



- https://ocw.mit.edu/courses/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/pages/lecture-slides-code/