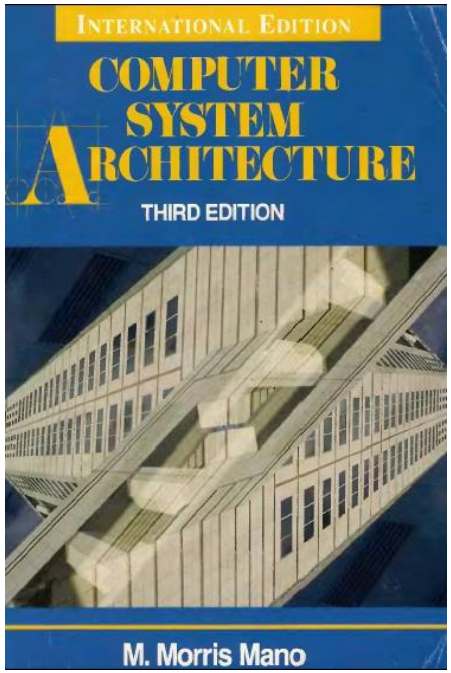
Computer Architecture

Dr. Esti Stein
(Partly taken from Dr. Alon Schclar slides)

Based on slides by:
Prof. Myung-Eui Lee
Korea University of Technology & Education
Department of Information & Communication

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Course Administration



- **Instructor**

  Carmi Merimovich (carmi@mta.ac.il)
  Esti Stein (esterst@mta.ac.il)

- **Exercise checker**

  ofirco@mta.ac.il

  talhaz@mta.ac.il

- **Books**

  Mano M.**: Digital Design,** Prentice Hall.

  Patterson and Hennessy, Computer Organization Design, The Hardware/Software Interface, *3rd Edition,* Morgan Kaufmann.

Course Administration

- Grade
  - Exam – 100% (must pass > 60)
  - Exercises
    - Every 1-2 weeks
    - 100% mandatory submission – 10 point reduction for each overdue day
- Office:
  - Zoom meetings
- Office hour
  - Email us

Course Outline

שבוע 1	מבוא כללי ושפת RTL	פרק 4
שבוע 2	ארגון המחשב – ה-BUS	פרק 5
שבוע 3	ארגון המחשב – יחידת הבקרה	פרק 5
שבוע 4	ארגון המחשב – פסיקות	פרק 5
שבוע 5	תכנות המחשב הבסיסי	פרק 6
שבוע 6	תכנות המחשב הבסיסי	פרק 6
שבוע 7	תכנות המחשב הבסיסי	פרק 6
שבוע 8	מיקרו תכנות	פרק 7
שבוע 9	מיקרו תכנות	פרק 7
שבוע 10	יחידות הקלט/פלט	פרק 11
שבוע 11	יחידות הקלט/פלט	פרק 11
שבוע 12	מבנה הזיכרון	פרק 12
שבוע 13	זיכרון CACHE	פרק 12

What is "Computer Architecture"

Architecture:

Design a building that is well suited to its purpose

Computer Architecture:

Design a computer that is well suited to its purpose

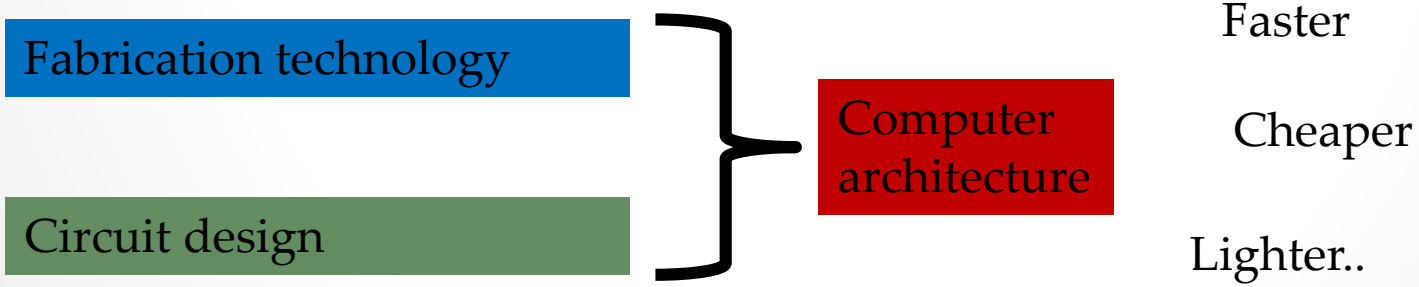
Why do we need computer architecture

Improve performance:

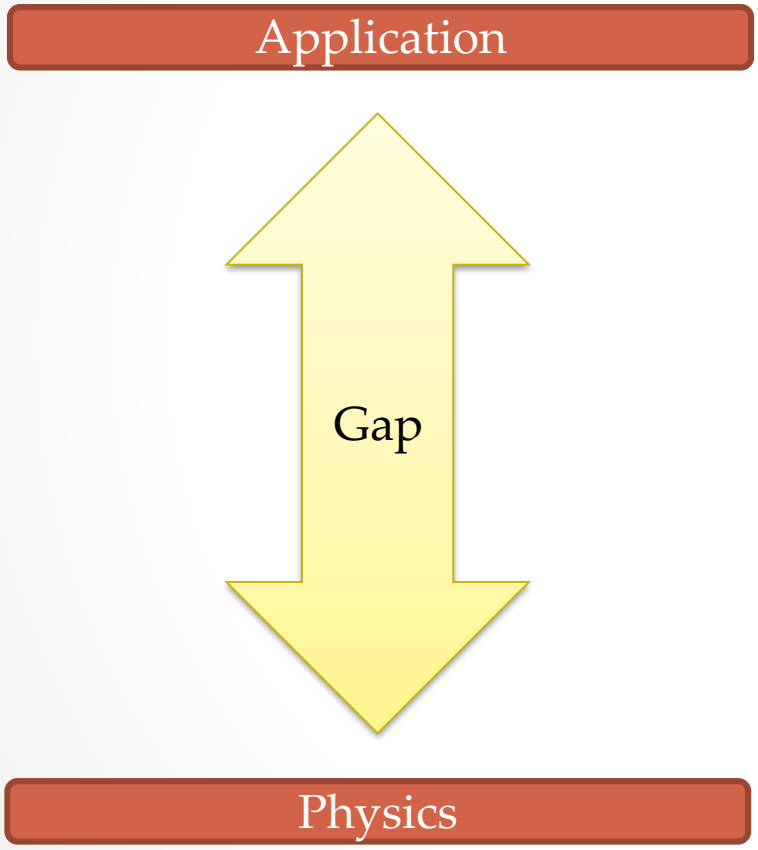
speed, battery life, size, weight, energy efficiency..

Improve abilities:

3d graphics, debugging support, security..

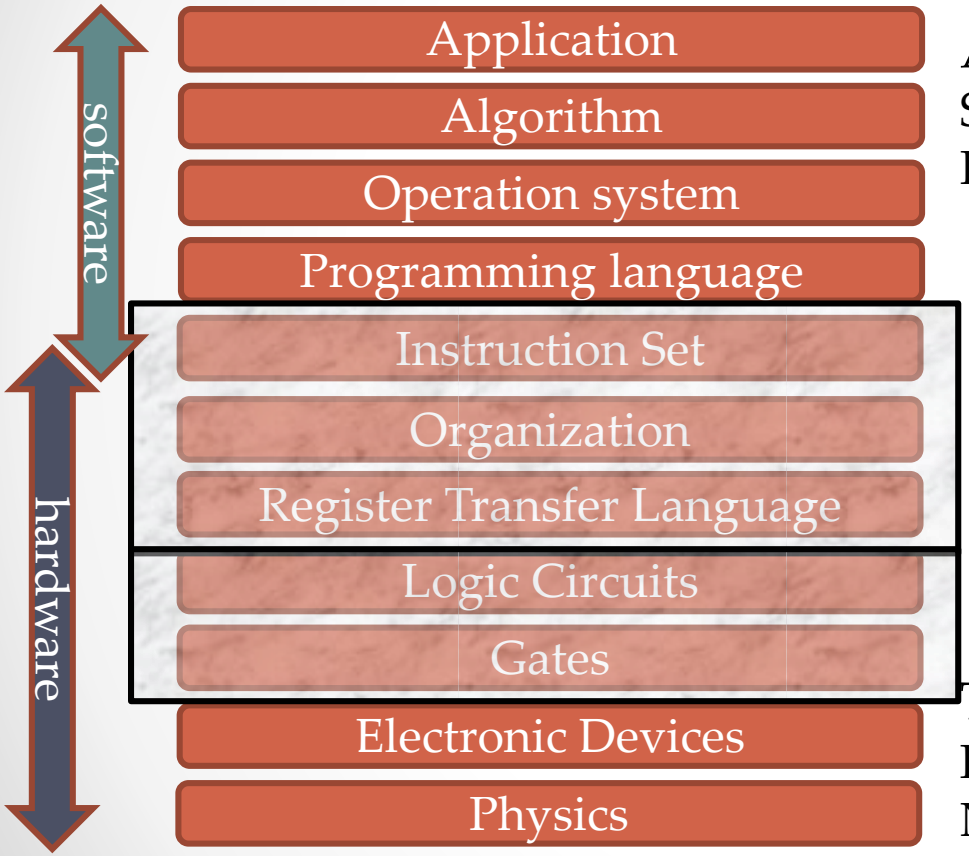


From Application to Physics



It's the abstraction of the implementation layers that needs to bridge the gap

From Application to Physics



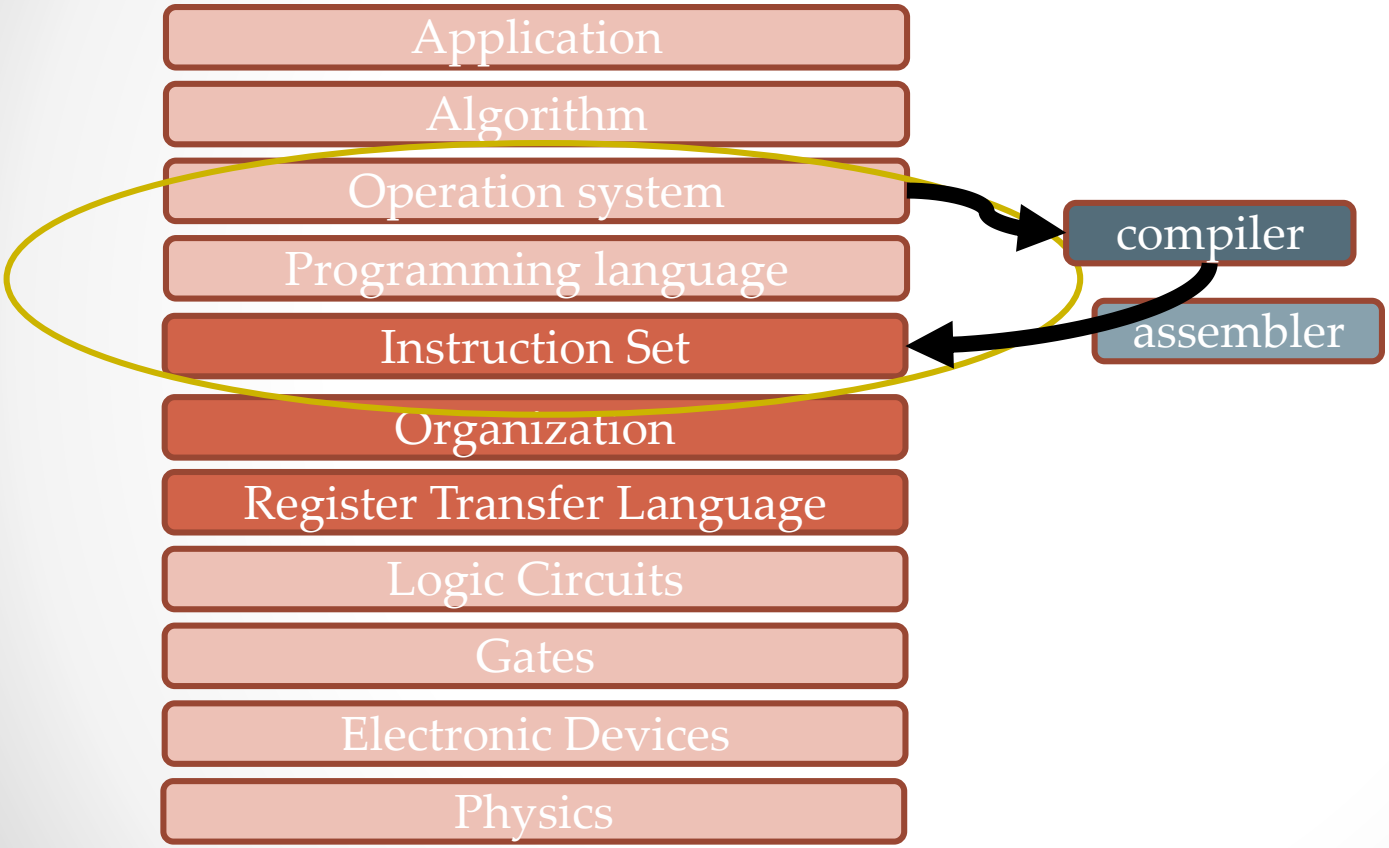
Application requirements:
Suggest how to improve architecture
Provide money to develop architecture

Our course

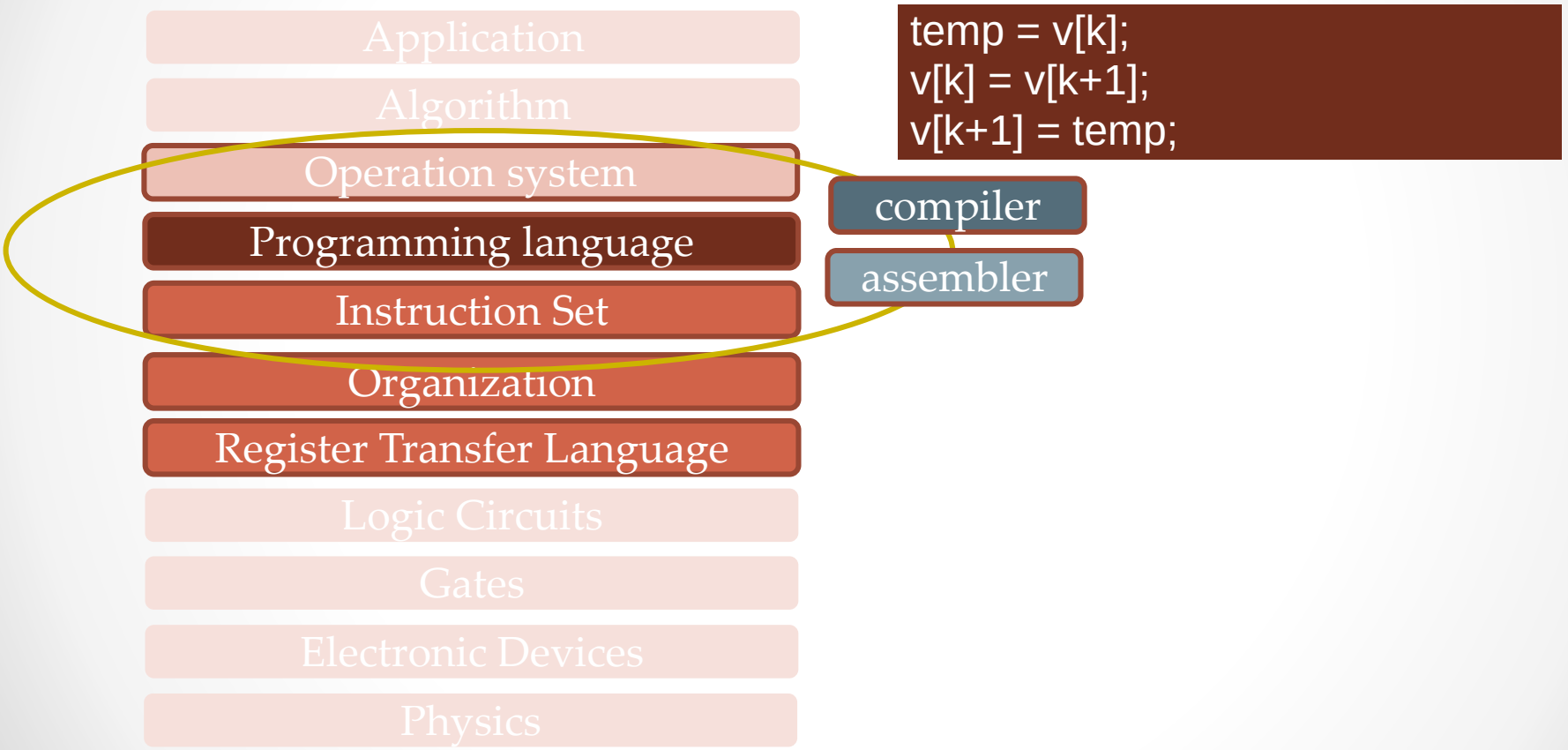
Computer Structure

Technology constraints:
Restrict the implementation
New technologies → new architecture

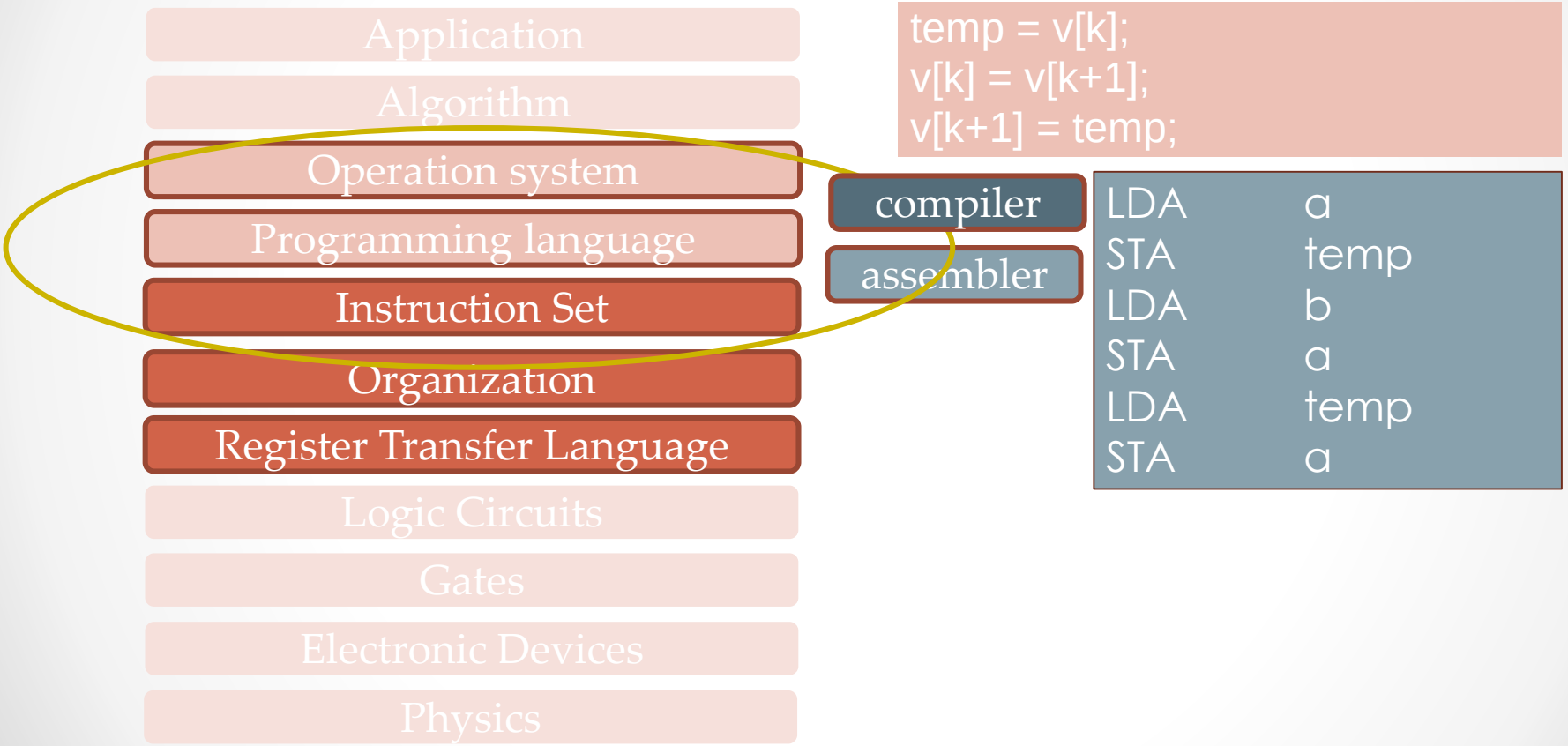
From Application to Physics



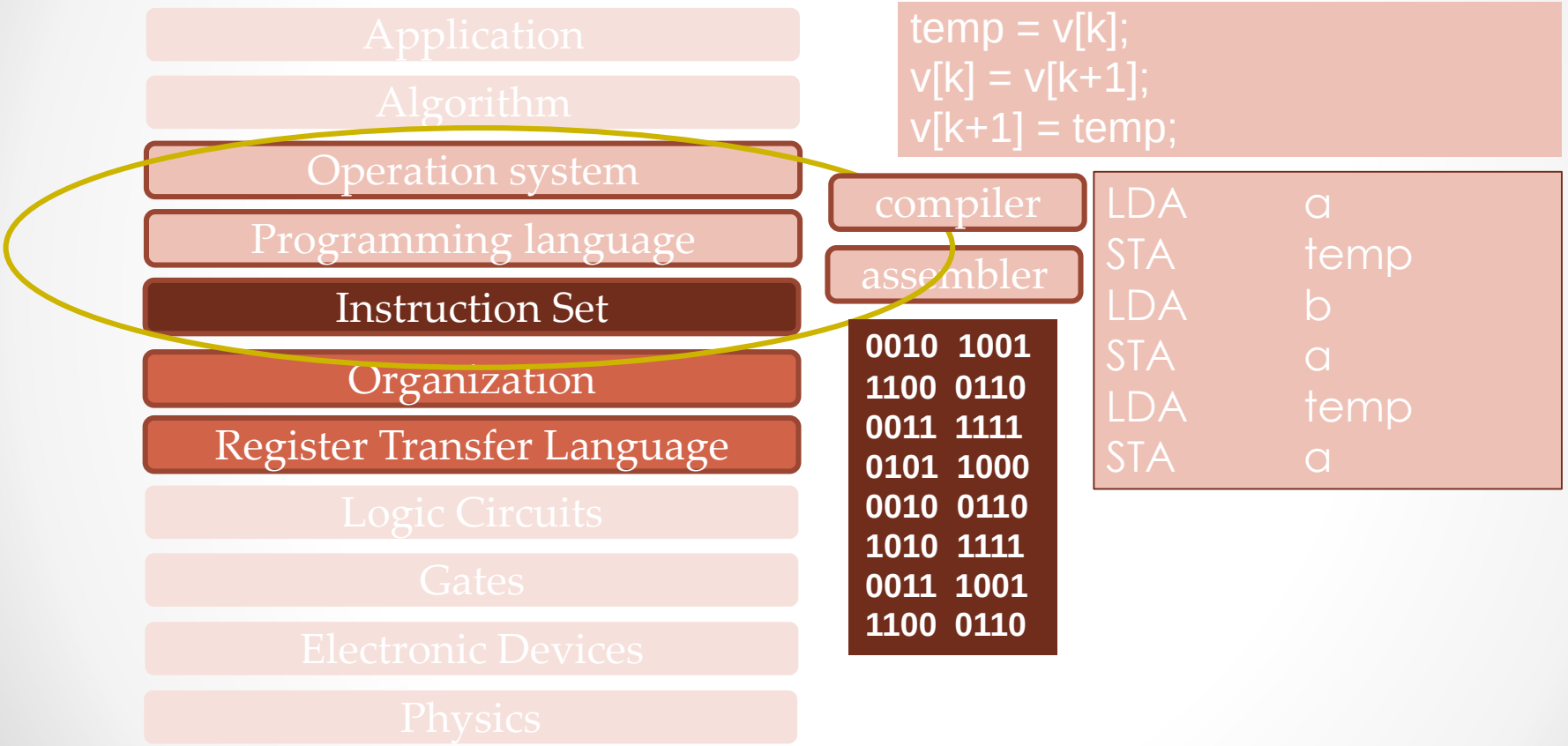
From Application to Physics



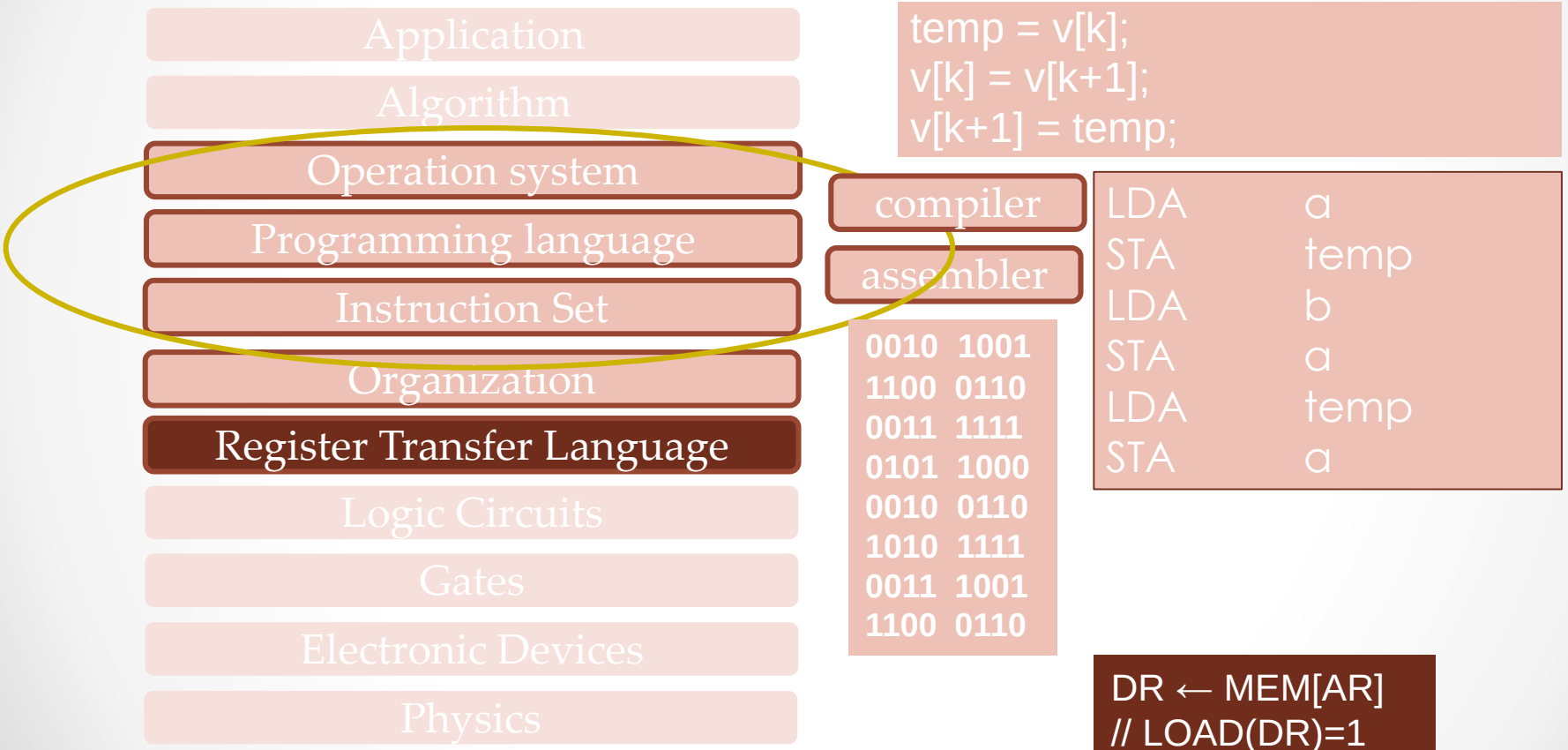
From Application to Physics



From Application to Physics



From Application to Physics



Talking to the Computer

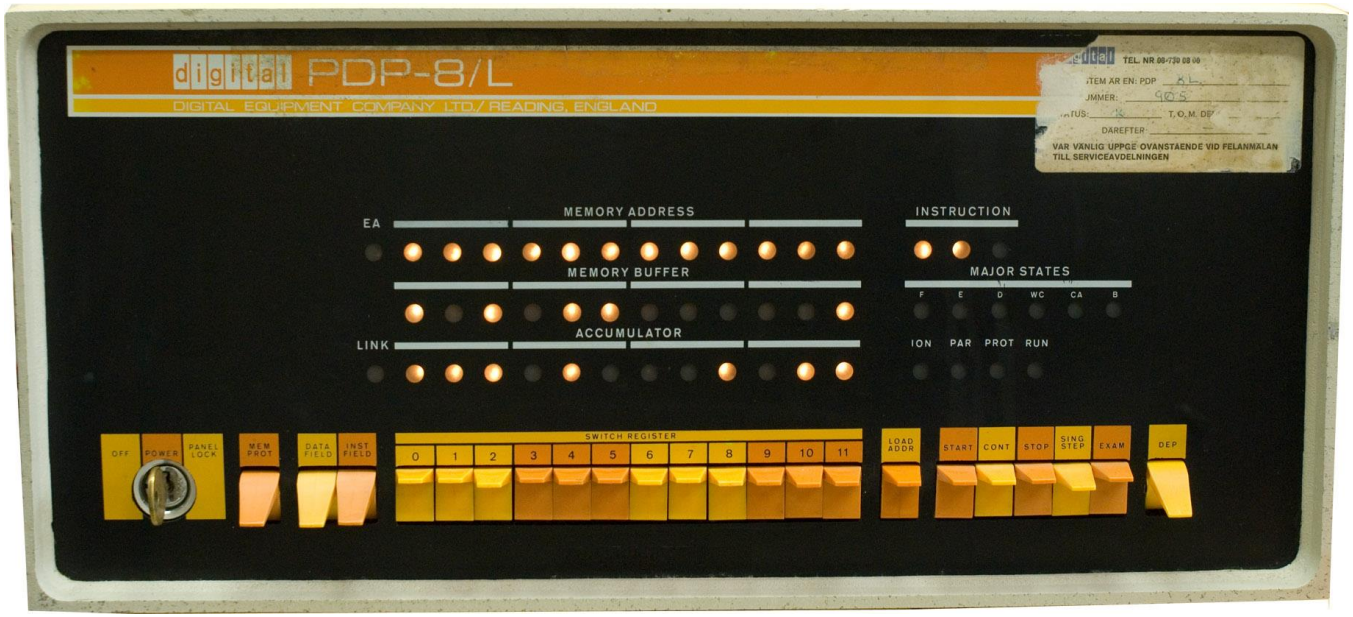
- In order to "**talk**" with the computer we must send it electronic signals.
- The easiest signals for an electronic machine to understand are ***on*** and ***off***
  - Correspond to *high* voltage and *low* **voltage**.
- Thus the computer's alphabet is composed of two symbols **0** and **1**.
- Any "words" composed of these 2 numbers are called ***binary numbers***.

Talking to the Computer

- A computer needs **our instructions** in order to function
- Instructions are formulated as **binary** numbers
- The binary number **11101100100001** can be an instruction to **subtract** two numbers
- Every computer understands a predefined set of instructions – **instruction set**
- An **instruction** has a **predefined structure** which matches the architecture of the computer

Assembly Language

- Initially (in the late 40s) computer **programs** where written as **binary numbers**
- They were input to the computer by **turning on and off** switches

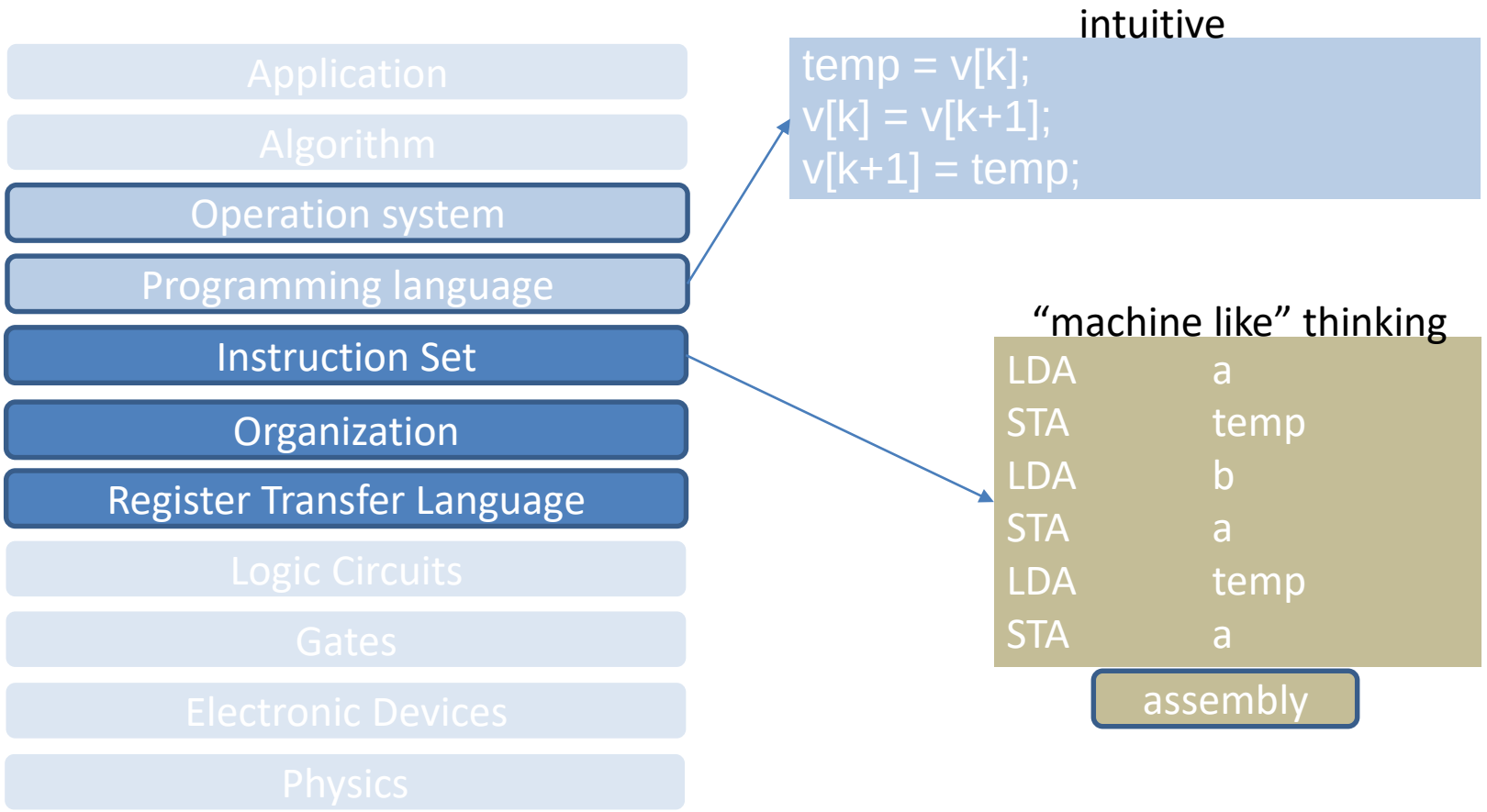


- It becomes extremely **tedious** after a short while

Assembly Language

- Next step: **automatic conversion** of the codes into binary instructions
- This program is called an **assembler**.
- The symbolic names of the instructions are called the **assembly language** :
  - Increases productivity
  - However
    - Each machine instruction must be written in a single line
    - The programmer needs to think like a machine
- A higher level of abstraction is needed

High-level & Assembly Languages



High Level Languages

- Humans think in a language that consists of **sentences** over the alphabet
- Why not **define** a **language** that is more **intuitive** for the programmer
- Next step :
  - Define a **high level language** (Fortran, JAVA, C)
  - Construct a translator that converts high level instructions into binary instructions – the **compiler**

A Simple Example

- The high level expression in C:

```
A = A + B
```

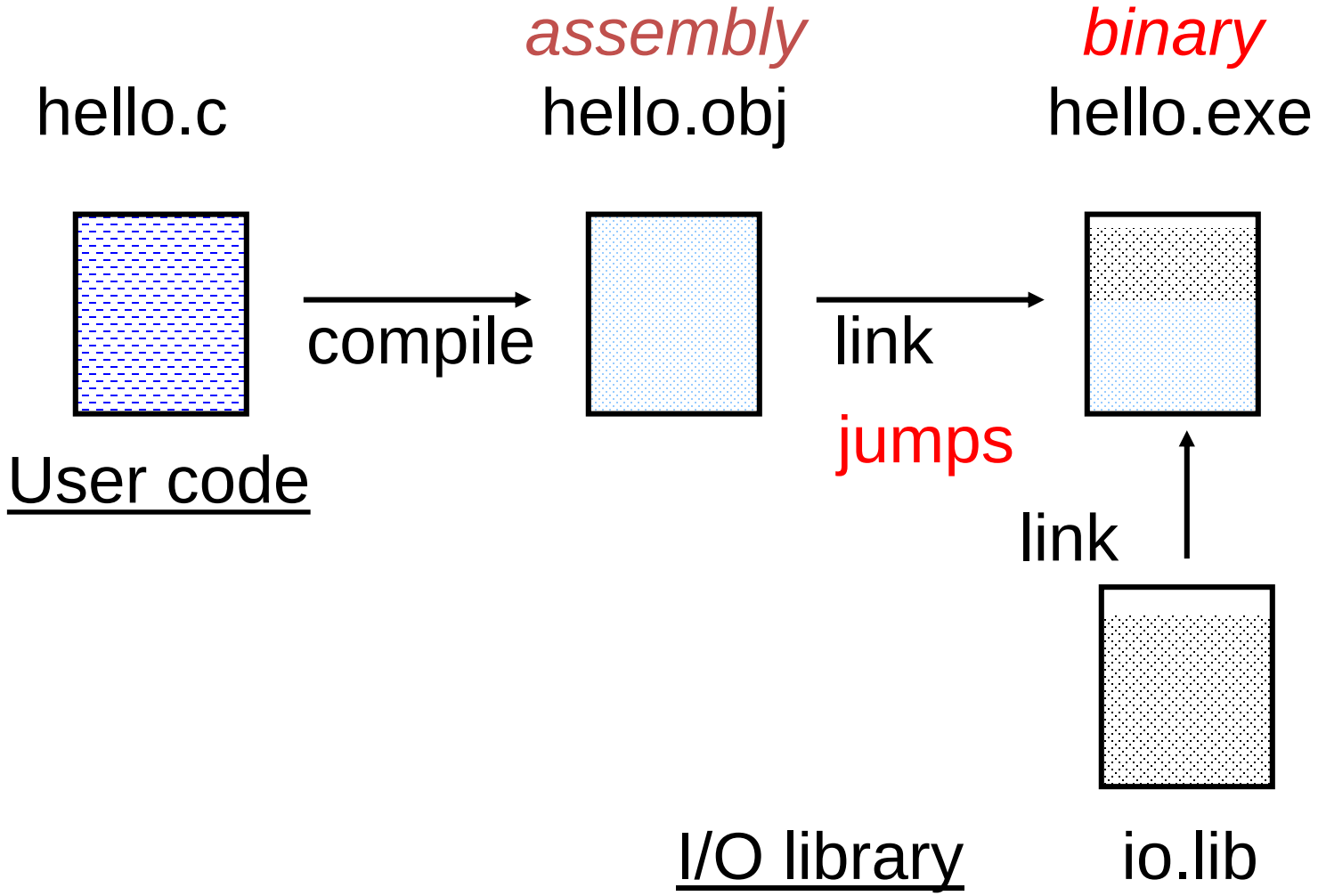
- Translates into the assembly instruction

```
LDA A
ADD B
STA A
```

- Translates into the binary instruction:

```
0010 110010100000
0001 100010100001
0011 110010100000
```

Compiling and Linking a Program



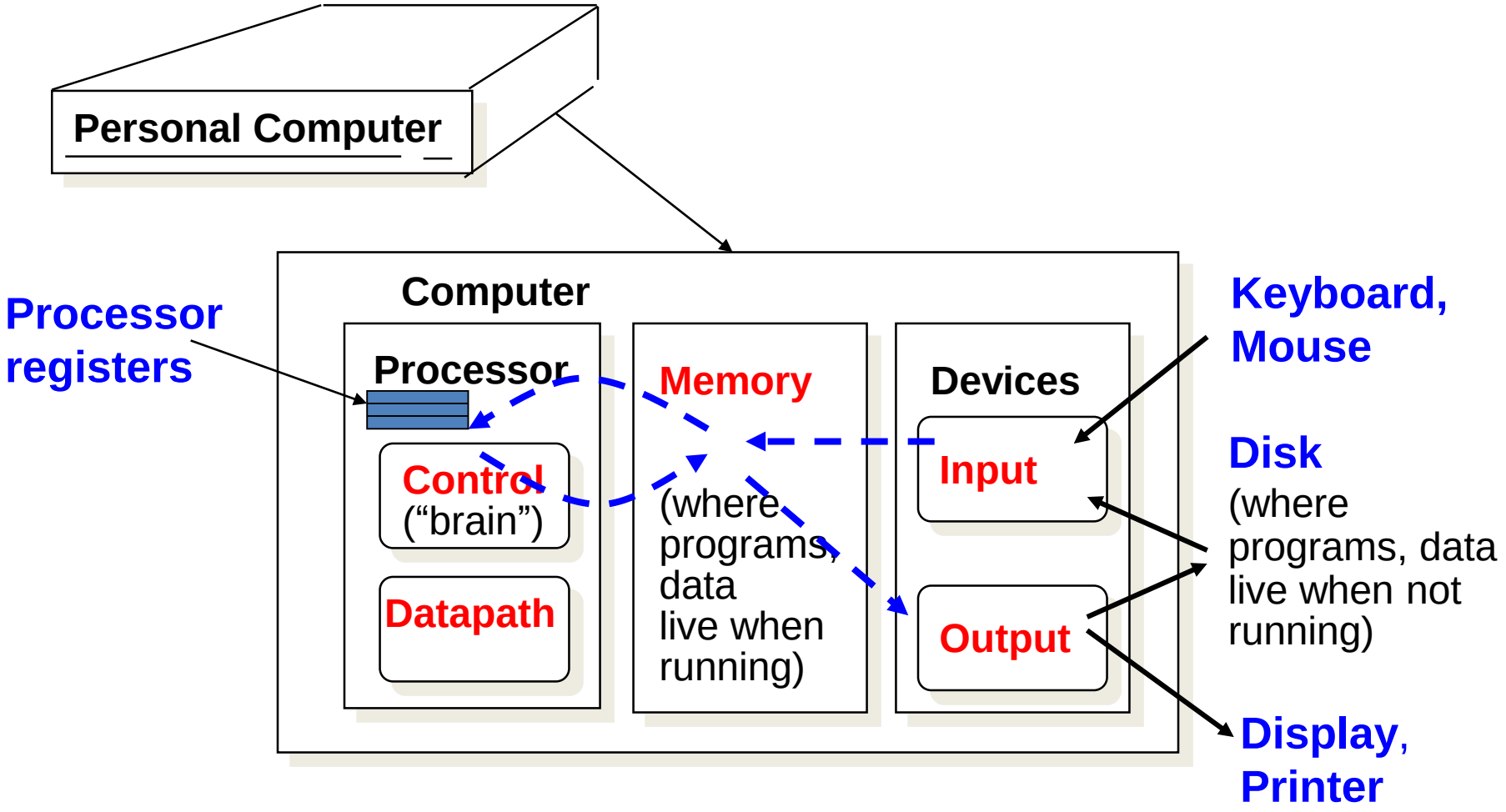
Executable program (EXE)

- A sequence of machine instructions
  - Binary coded
  - Operations, operands (values)
  - Operands may be
    - Memory addresses (= variables)
    - Values – floating point, 2's-comp
    - Register numbers

From EXE file to execution

- Given a binary executable file
  - the file is ***loaded*** from the I/O device (hard disk, DVD, Disk-on-Key etc.) into the memory
  - the processor **executes** the instructions in an **iterative** process
  - the **operating system** coordinates this process

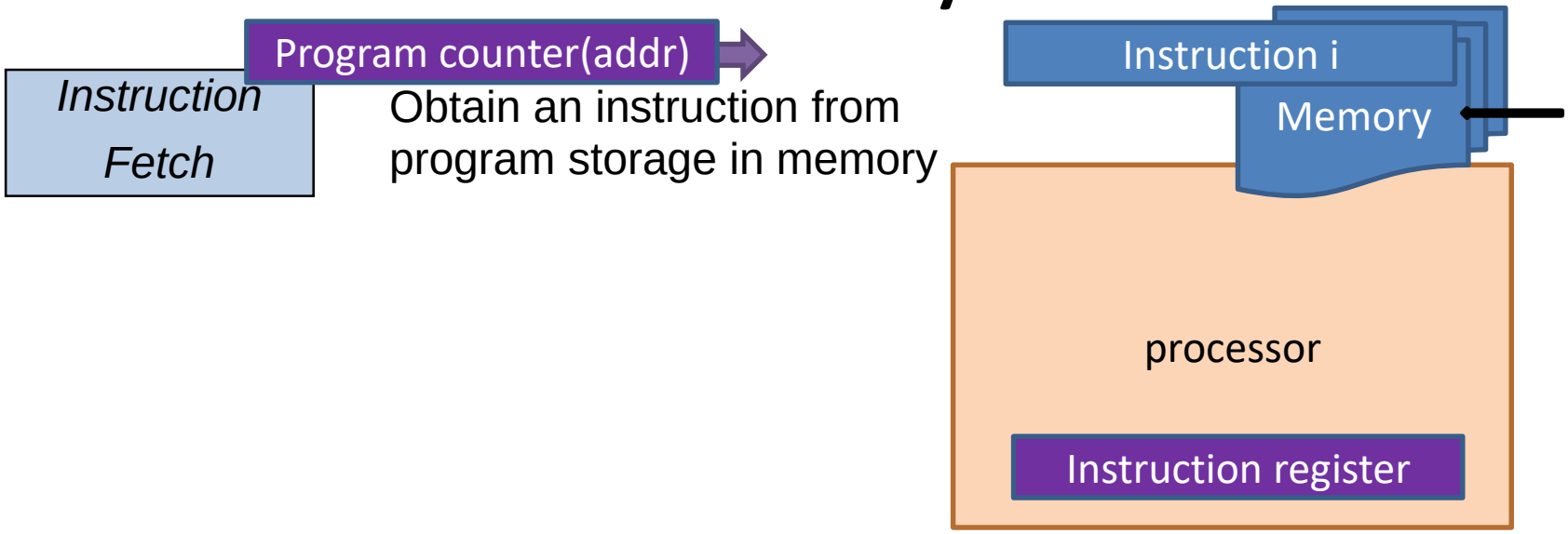
Anatomy:
5 components of any Computer



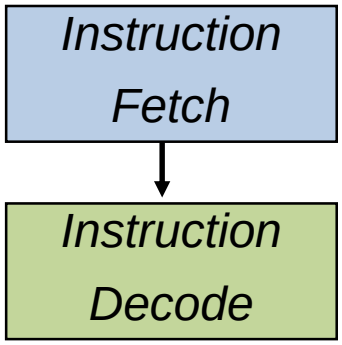
Inter-component flow

- Data flows from the input devices into memory
- From the memory into the processor
- The data is processed and written back to memory
- It is then stored or displayed in the output devices

Execution Cycle

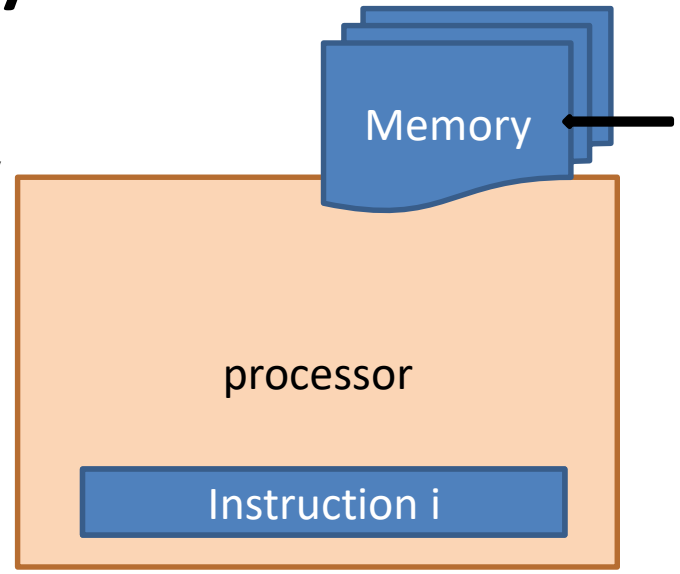


Execution Cycle

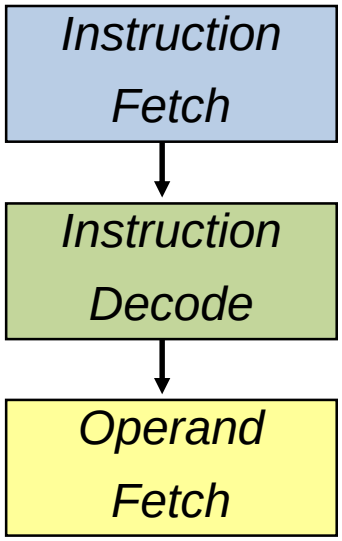


Obtain instruction from program storage in memory

Determine required actions and instruction size



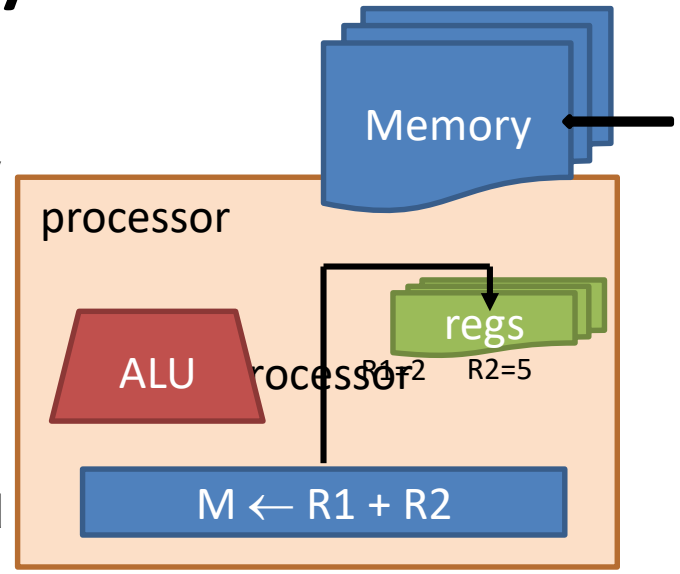
Execution Cycle



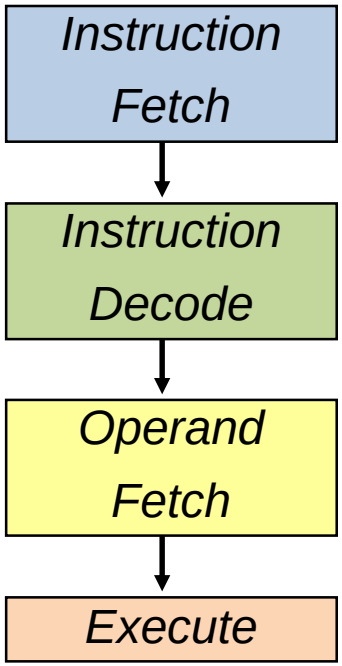
Obtain instruction from program storage in memory

Determine required actions and instruction size

Locate and obtain operand data



Execution Cycle

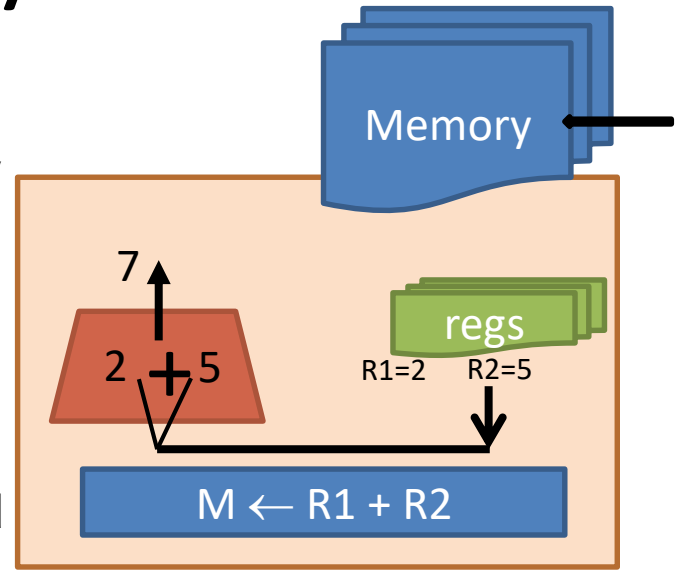


Obtain instruction from program storage in memory

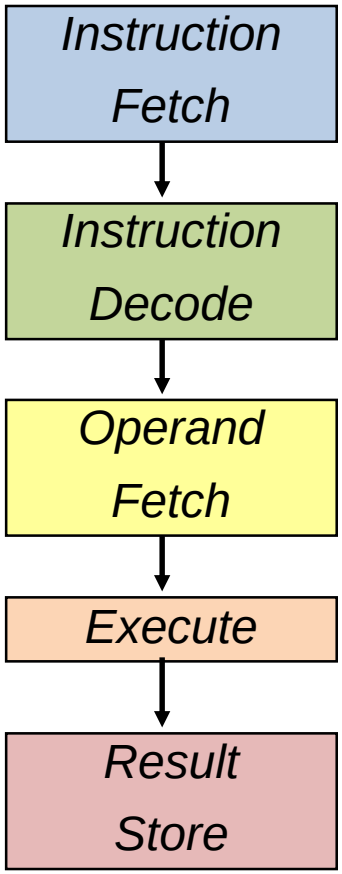
Determine required actions and instruction size

Locate and obtain operand data

Compute result value or status



Execution Cycle



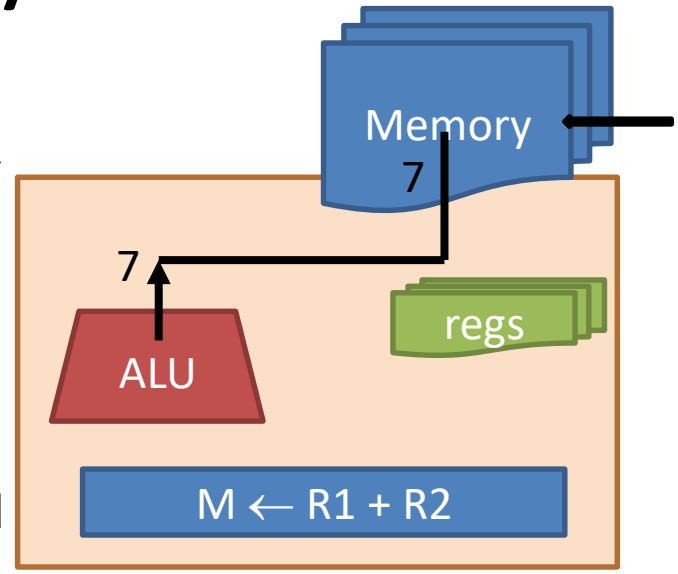
Obtain instruction from program storage in memory

Determine required actions and instruction size

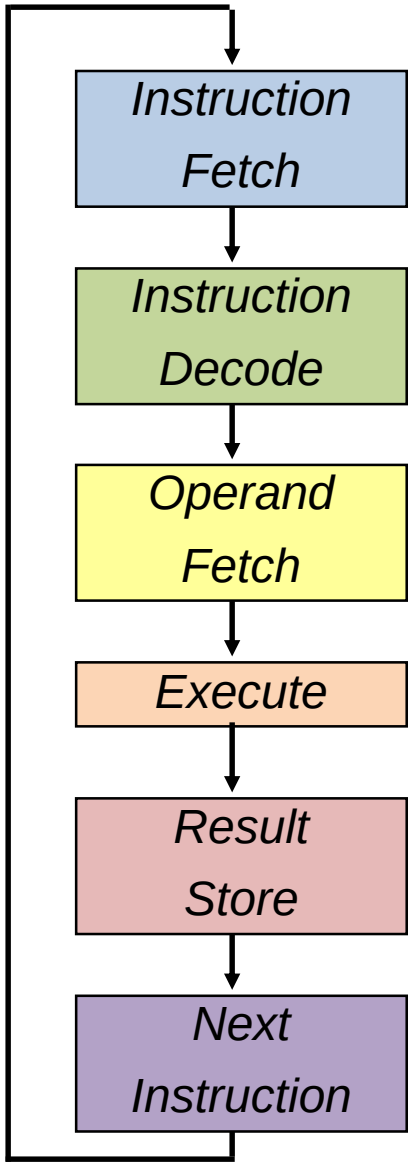
Locate and obtain operand data

Compute result value or status

Deposit results in storage



Execution Cycle



Obtain instruction from program storage in memory

Determine required actions and instruction size

Locate and obtain operand data

Compute result value or status

Deposit results in storage

Determine next instruction (not the next in case of **branch**)