Computer Organization 2022
Lab1 – Introduction & Environment Lab
(Part I : Lab Introduction)

Video link : https://youtu.be/UOyDK2rI8_E

Outline

- Part I - Lab Introduction
- Part II - Server Usage Basic Guide
- Part III - A Simple Experiment

Part I
Lab Introduction

Objective of this lab course

Knowledge

- Learn What is the CPU & How it works

Implementation

- Learn How to design & implement a Pipeline RISC-V CPU
- Learn How to add advanced technology to a Pipeline RISC-V CPU (branch prediction & cache)

What is the Computer

Automatically

1. **Accept** the data
2. **Process (verb.)** the accepted (or pre-prepared) data with preset logic (or program)
3. **(Optional) Produce** the result (or meaningful data)

E.g.
- Microwave ovens
- Smartphone
- Personal Computer
- …

Accept

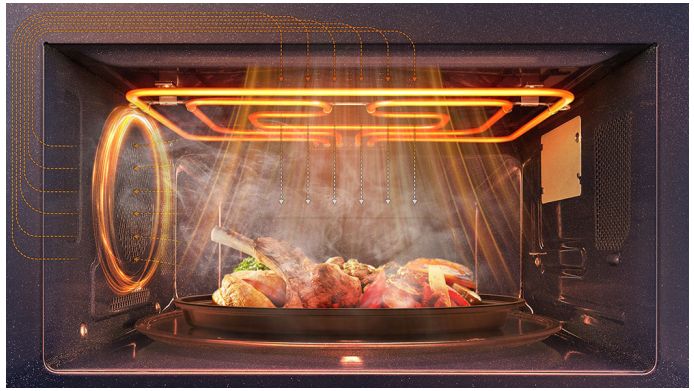
data

Computer

(**Process**)

Produce

result

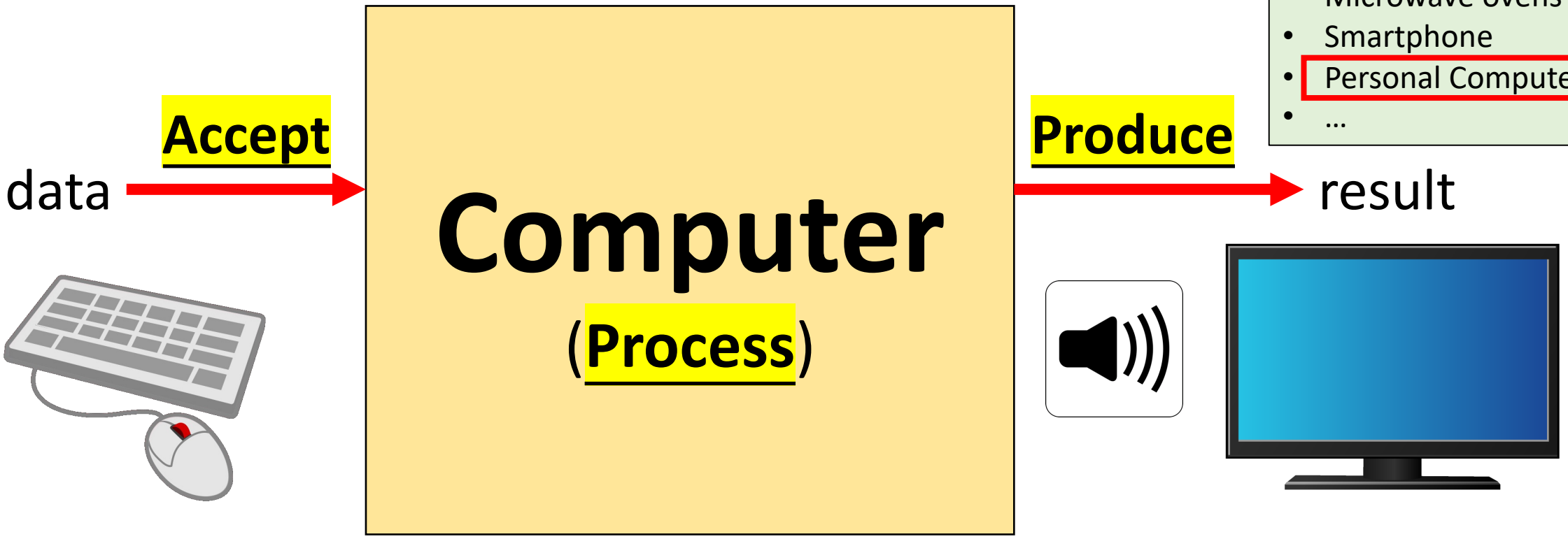


What is the Computer

Automatically

1. **Accept** the data
2. **Process (verb.)** the accepted (or pre-prepared) data with preset logic (or program)
3. **(Optional) Produce** the result (or meaningful data)

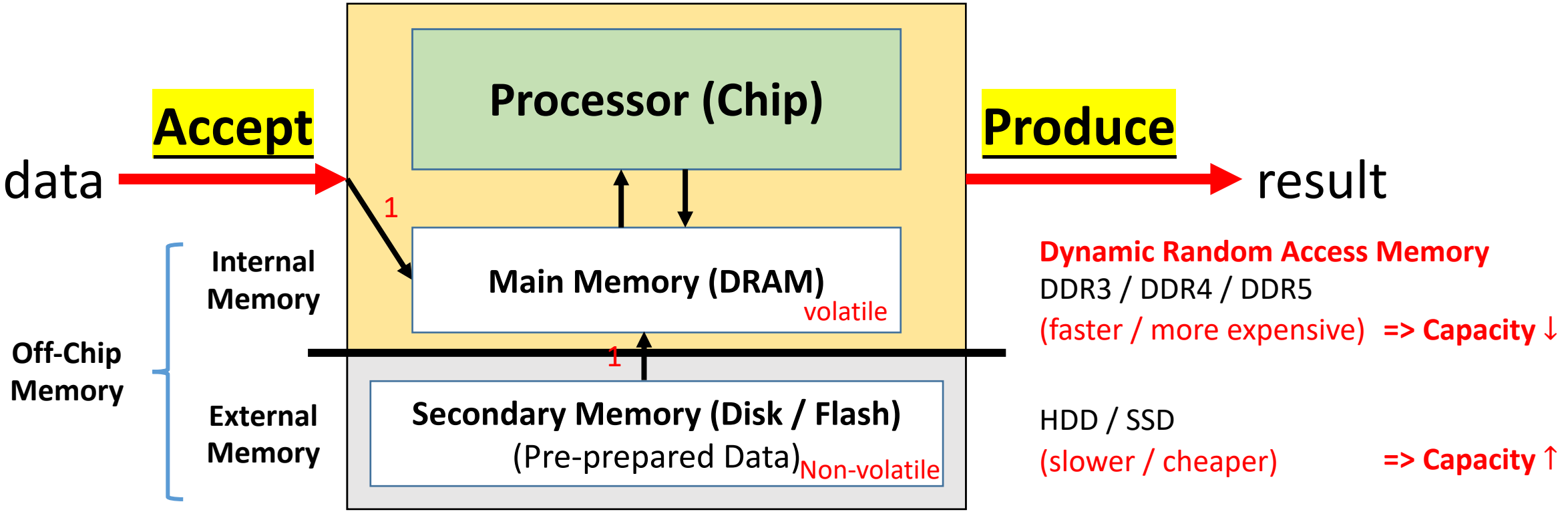
E.g.
- Microwave ovens
- Smartphone
- Personal Computer
- …



What your computer is made of & How they work

Automatically

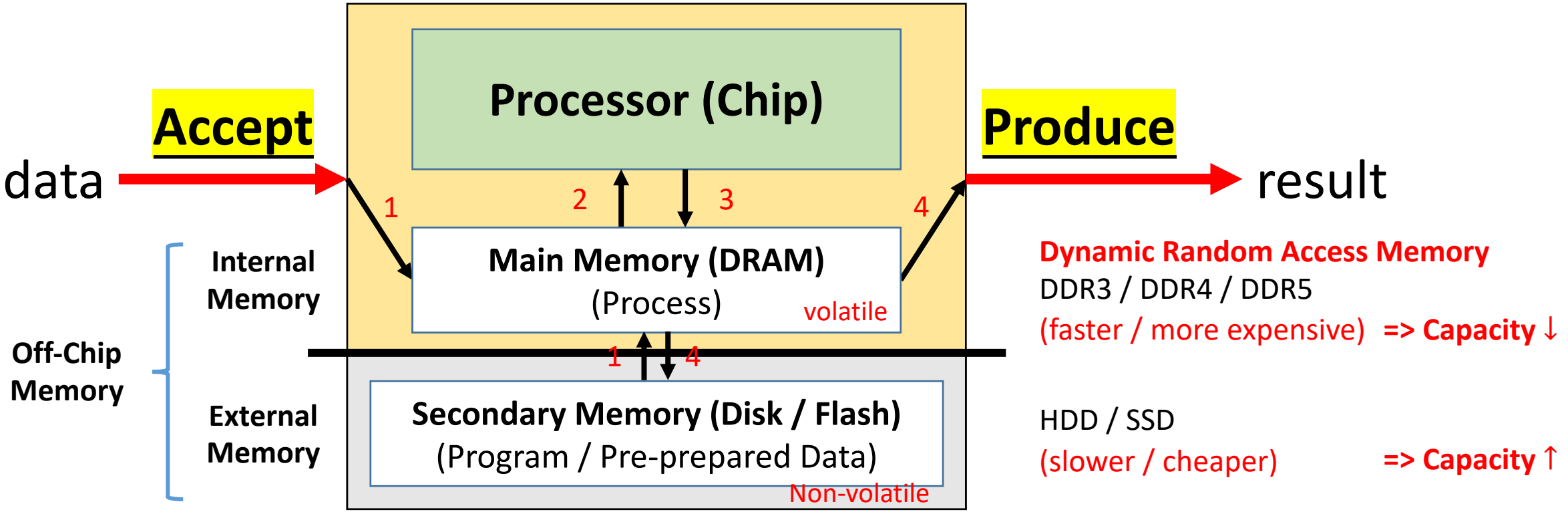
1. **Accept** the data
2. **Process (verb.)** the accepted (or pre-prepared) data with preset logic (or program)
3. **(Optional) Produce** the result (or meaningful data)



What your computer is made of & How they work

Automatically

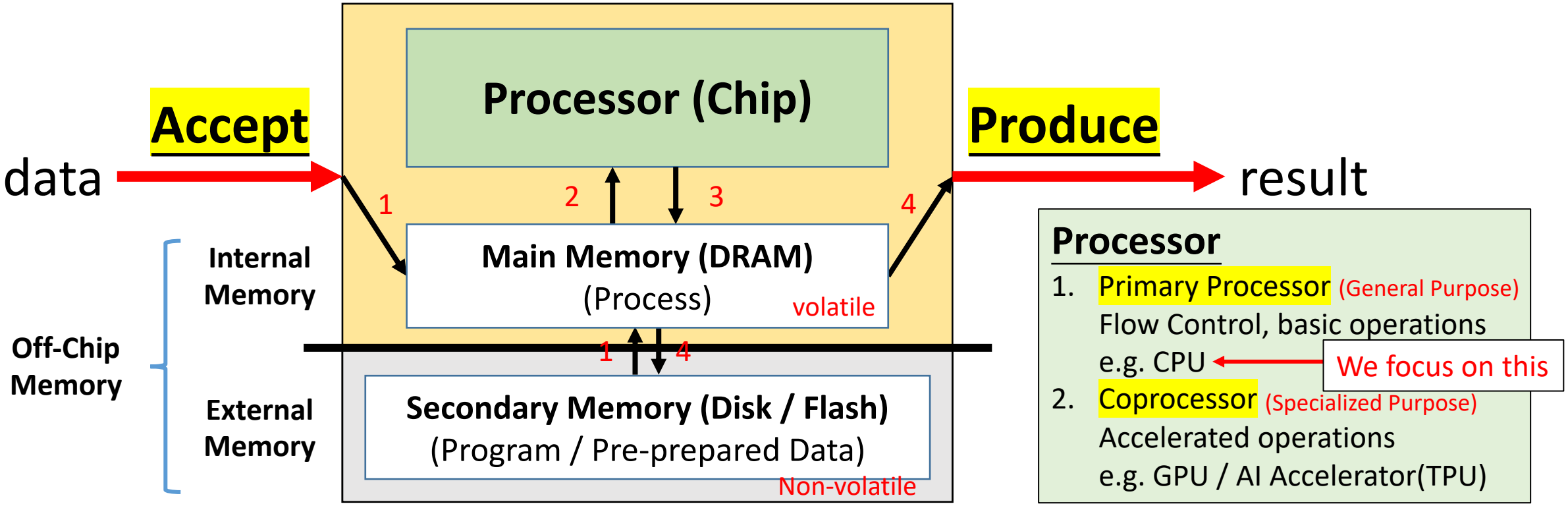
1. **Accept** the data
2. **Process (verb.)** the accepted (or pre-prepared) data with preset logic (or program)
3. **(Optional) Produce** the result (or meaningful data)



What your computer is made of & How they work

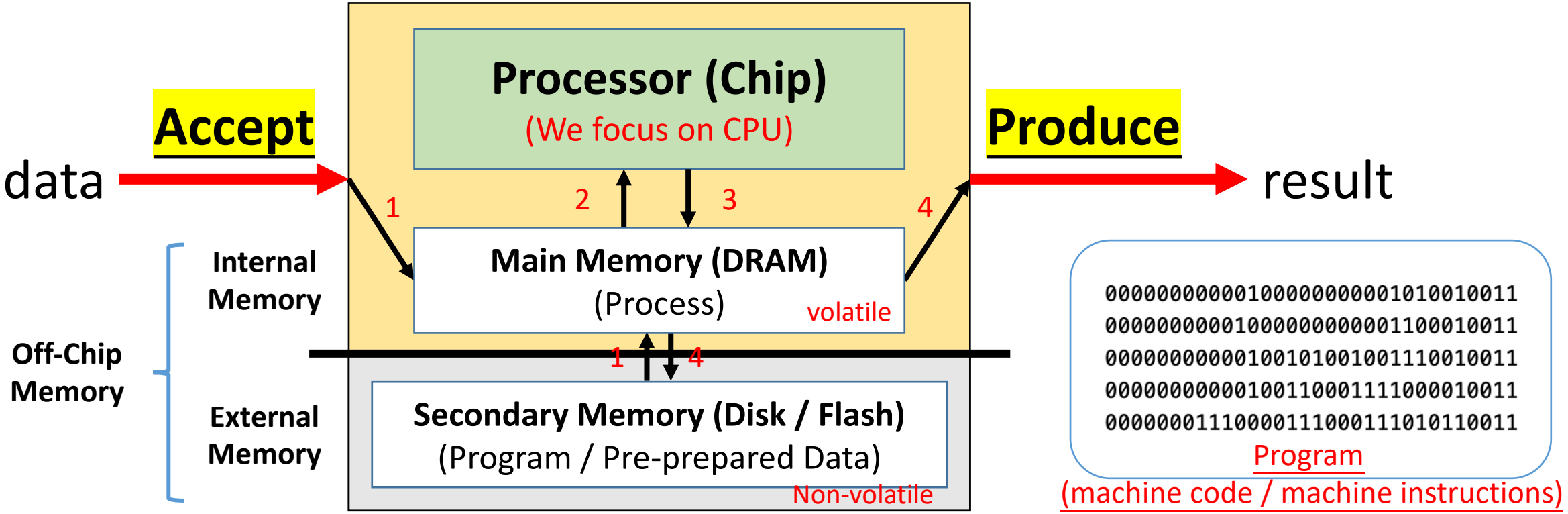
Automatically

1. **Accept** the data
2. **Process (verb.)** the accepted (or pre-prepared) data with preset logic (or program)
3. **(Optional) Produce** the result (or meaningful data)



What is the CPU (Central Processing Unit)

- Central Processing Unit (CPU) also called a central processor, main processor or just processor
- CPU is a general-purpose processor focused on flow control
- CPU is the electronic circuitry that executes program (instructions stream / instruction sequence)

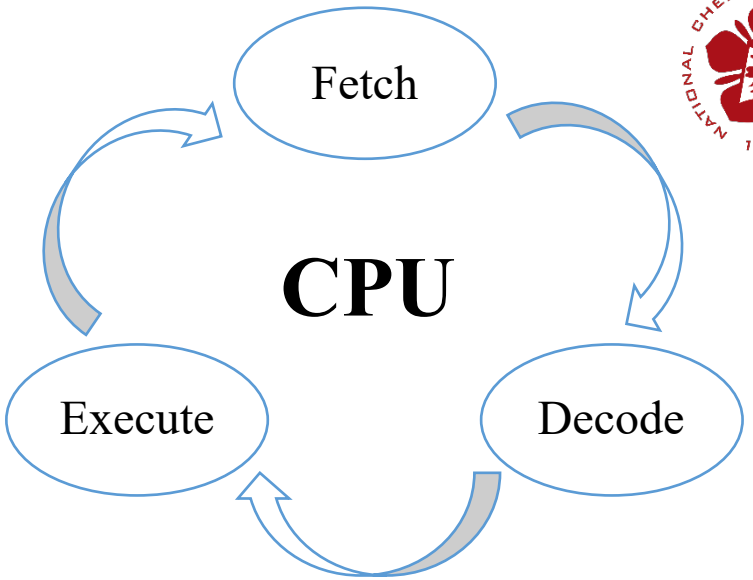


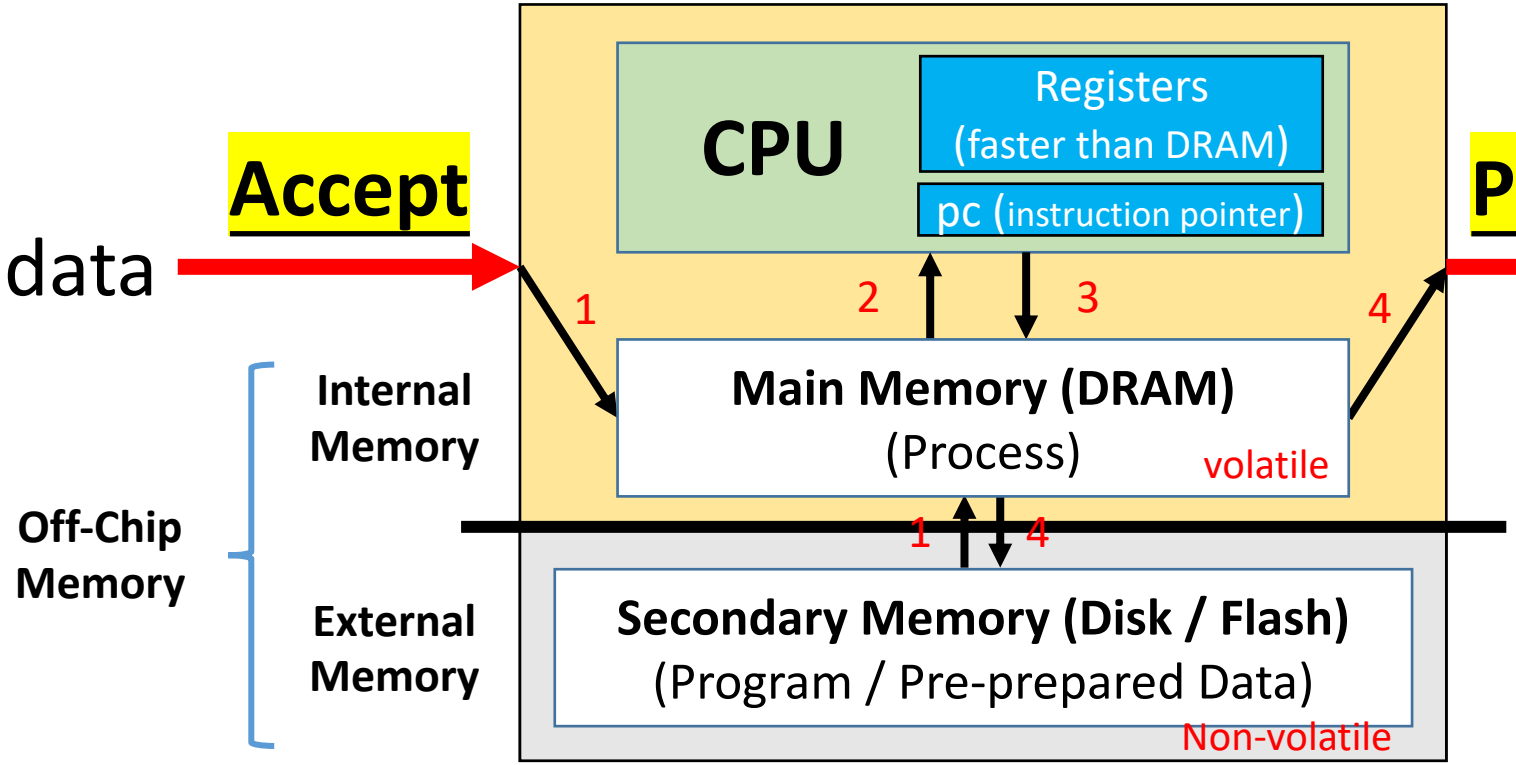
What the CPU actually does

1. **Fetch** the instruction
2. **Decode** the instruction
3. **Execute** the instruction

Take a simple Example
C = A + B

1. Load A (DRAM -> Register 1)
2. Load B (DRAM -> Register 2)
3. A + B (Reg 1 + Reg 2-> Reg 3)






```
0000000000010000000000001010010011
0000000000010000000000001100010011
0000000000010010100100111001001 1
0000000000010011000111100001001 1
0000000111000011100011101011001 1
```

Program
(machine code / machine instructions)

How the Hardware executes the C code ?

How can the hardware understand
and execute this C code (Source Code) ?

```
int main() {
    int a = 1;
    int b = 2;
    int c = a*2+b*4;
    return 0;
}
```

software

hardware

How the Hardware executes the C code ?

```c
int main() {
    int a = 1;
    int b = 2;
    int c = a*2+b*4;
    return 0;
}
```

Convert

Compiler

```
00000000000100000000001010010011
00000000001000000000001100010011
00000000000100101001001110010011
00000000000100110001111000010011
00000001110000111000111010110011
```

software

hardware

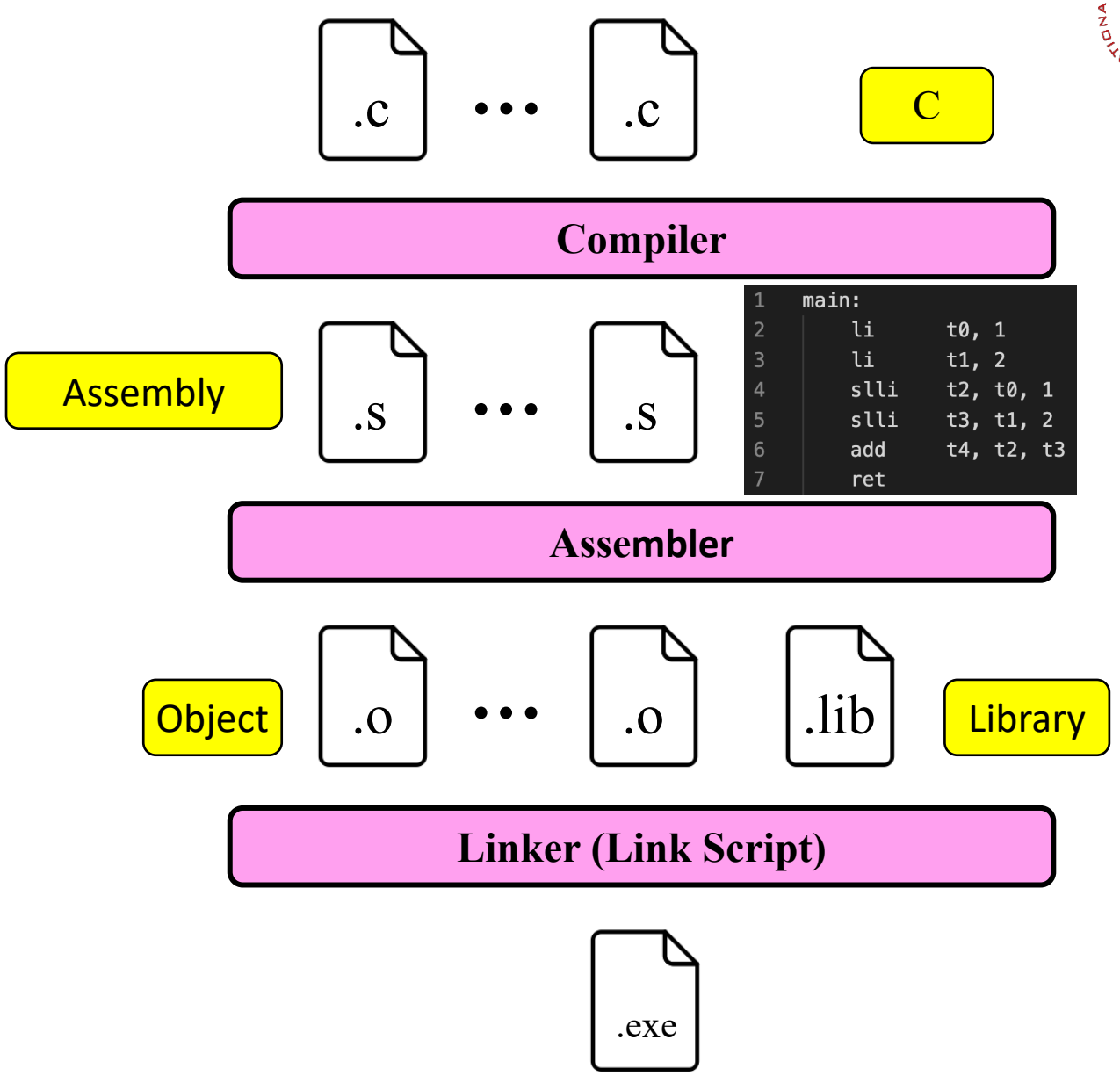
Hardware can only understand "machine instructions" which it supports

Compilation Flow

```c
int main() {
    int a = 1;
    int b = 2;
    int c = a*2+b*4;
    return 0;
}
```

Convert Compiler

```
00000000000100000000001010010011
00000000001000000000001100010011
00000000000100101001001110010011
00000000000100110001111000010011
00000001110000111000111010110011
```



How the Hardware executes the C code ?

```c
int main() {
    int a = 1;
    int b = 2;
    int c = a*2+b*4;
    return 0;
}
```

Convert

Compiler

```
00000000000100000000001010010011
00000000001000000000001100010011
00000000000100101001001110010011
00000000000100110001111000010011
00000001110000111000111010110011
```

Think…

What will happen when different hardware supports different instructions ?

software

hardware

Hardware can only understand "machine instructions" which it supports

How the Hardware executes the C code ?

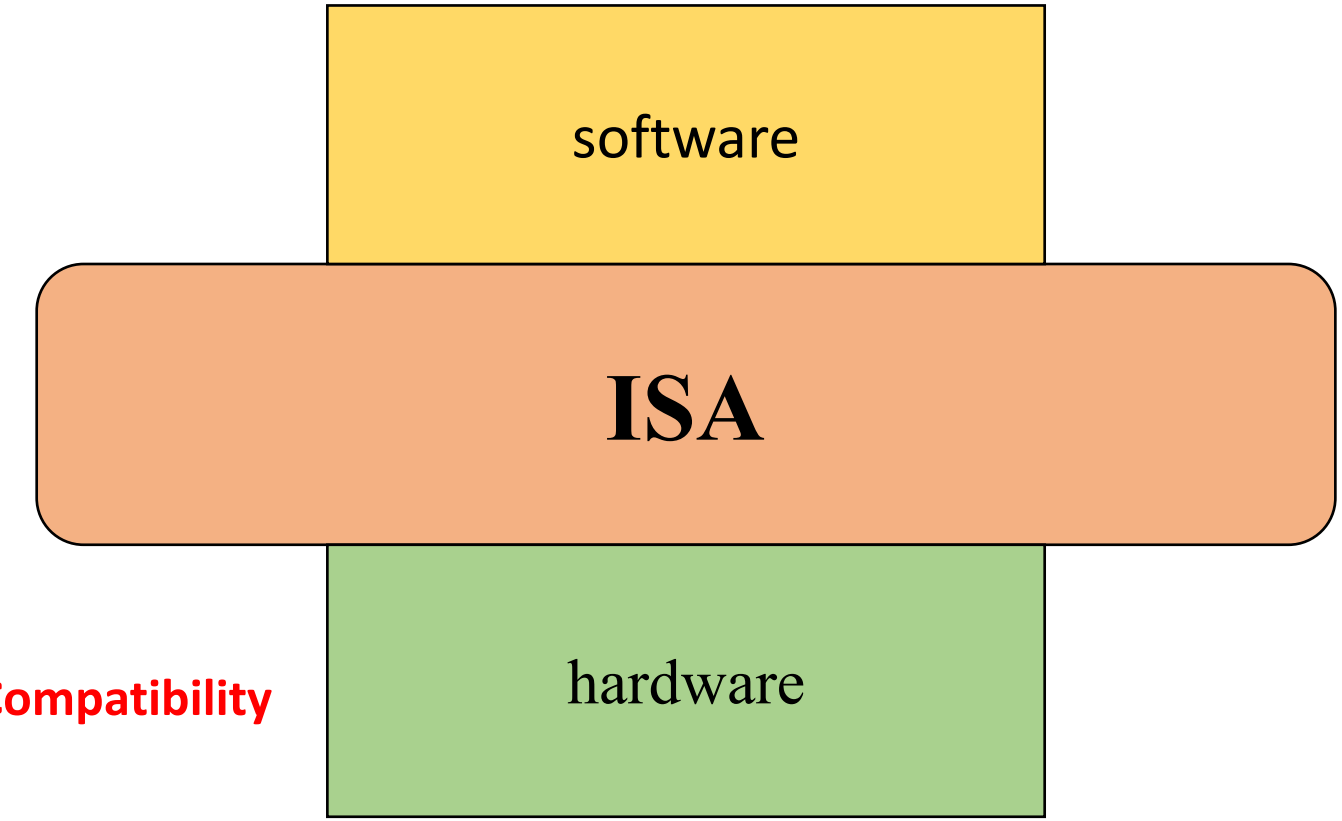
- Convert “C code” to “machine code” **according to ISA specifications**

```c
int main() {
    int a = 1;
    int b = 2;
    int c = a*2+b*4;
    return 0;
}
```

Convert Compiler

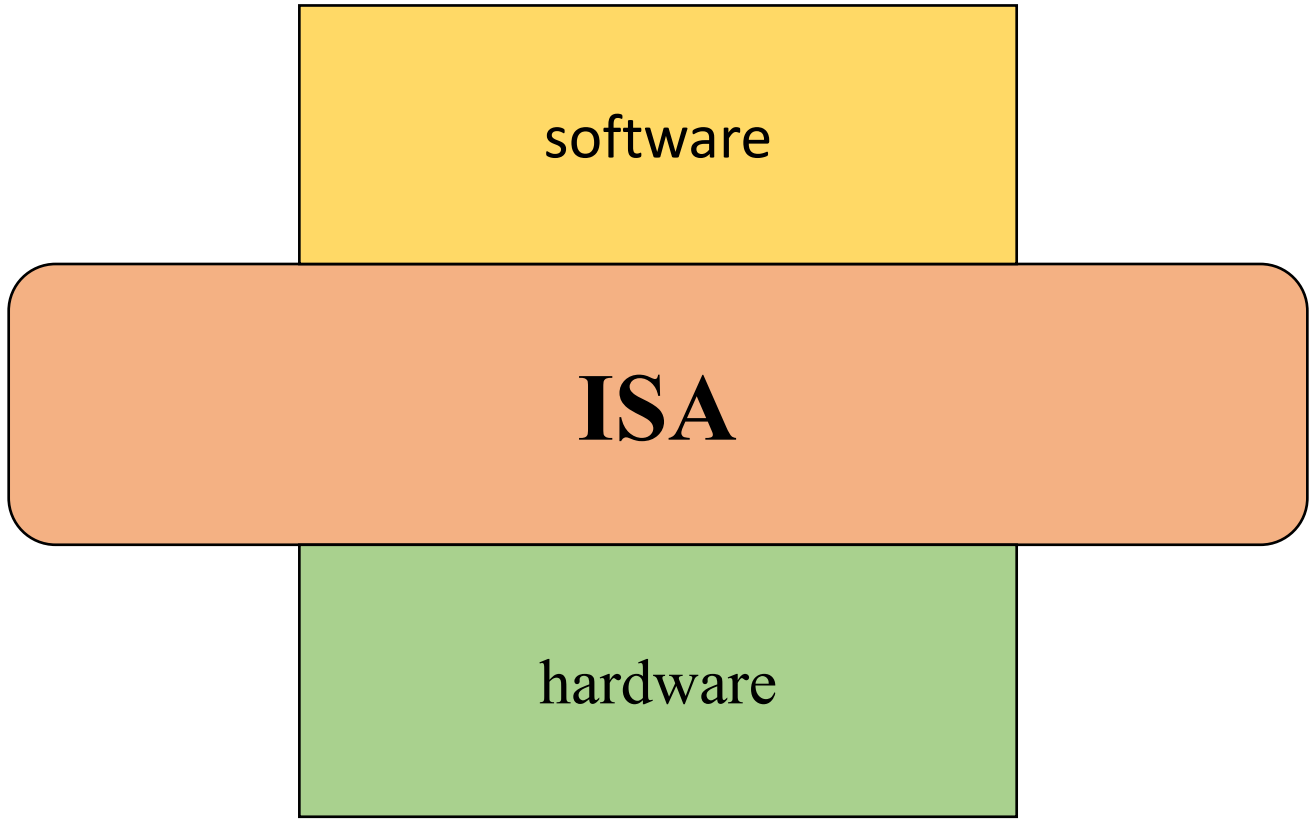
```
00000000000100000000001010010011
00000000001000000000001100010011
00000000000100101001001110010011
00000000000100110001111000010011
00000001110000111000111010110011
```

Binary Compatibility



ISA (Instruction Set Architecture)

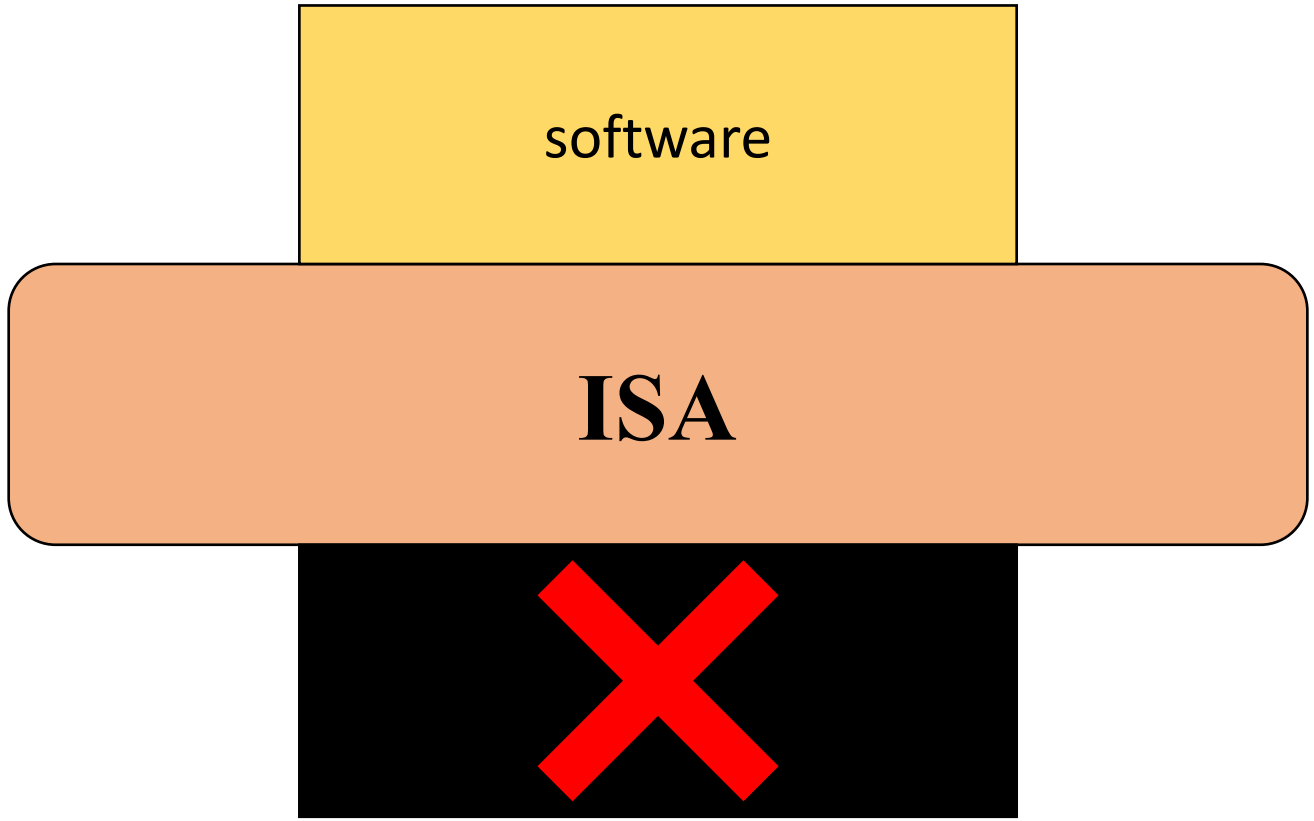
- It's an **interface** between the hardware and the software



ISA (Instruction Set Architecture)

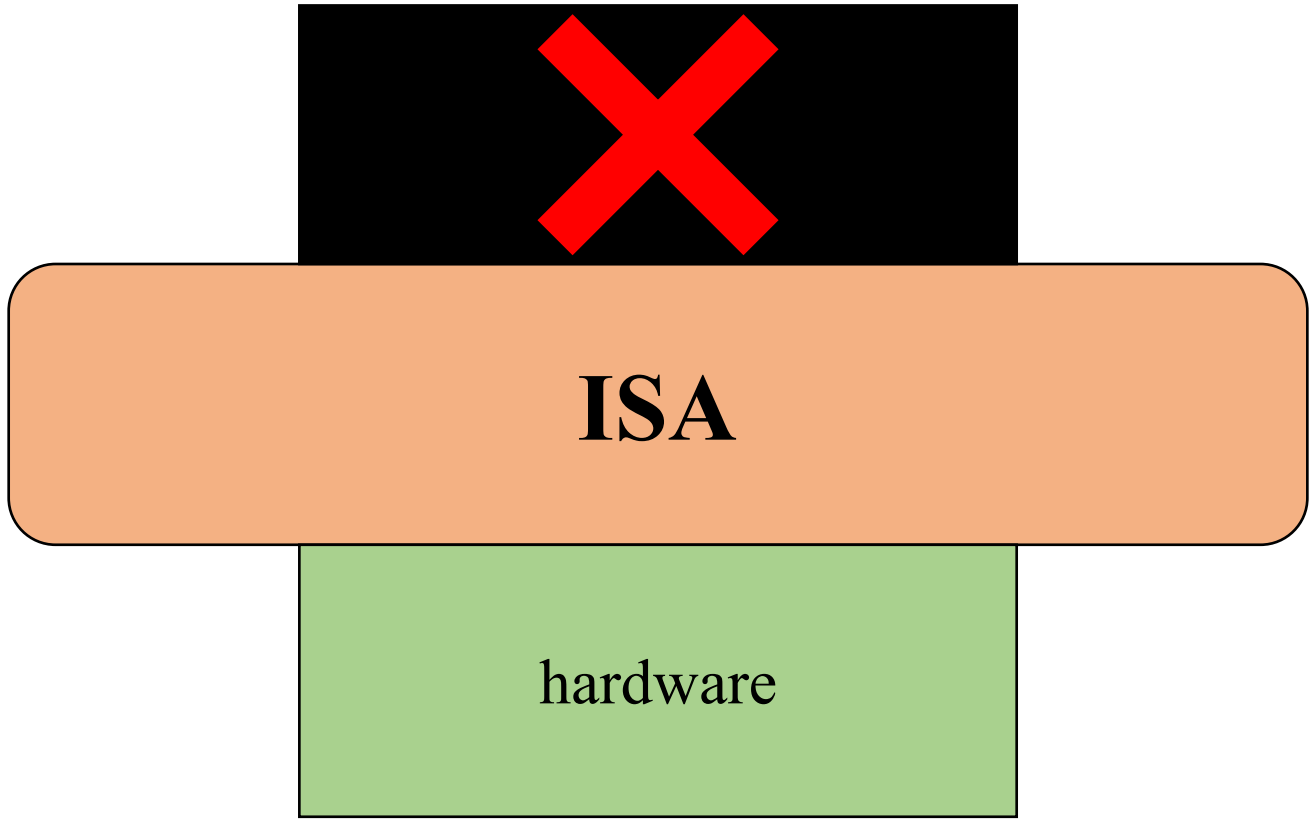
- It's an **interface** between the hardware and the software
- It's **a set of instructions** that defines how the hardware is controlled by the software
- **For Software**, it's an abstraction of the hardware

```
00000000000100000000001010010011
00000000001000000000001100010011
00000000000100101001001110010011
00000000000100110001111000010011
00000001110000111000111010110011
```



ISA (Instruction Set Architecture)

- It's an **interface** between the hardware and the software
- It's **a set of instructions** that defines how the hardware is controlled by the software
- **For Software**, it's an abstraction of the hardware
- **For Hardware**, it's an implementation target



ISA (Instruction Set Architecture)

- It's an **interface** between the hardware and the software
- It's **a set of instructions** that defines how the hardware is controlled by the software
- **For Software**, it's an abstraction of the hardware
- **For Hardware**, it's an implementation target
- ISA can also be viewed as a **programmer's manual** which is referenced by the assembly language programmer and the compiler writer.

```
int main() {
    int a = 1;
    int b = 2;
    int c = a*2+b*4;
    return 0;
}
```

Convert Compiler

```
00000000000100000000001010010011
00000000000100000000001100010011
00000000000100101001001110010011
00000000000100110001111000010011
00000001110000111000111010110011
```

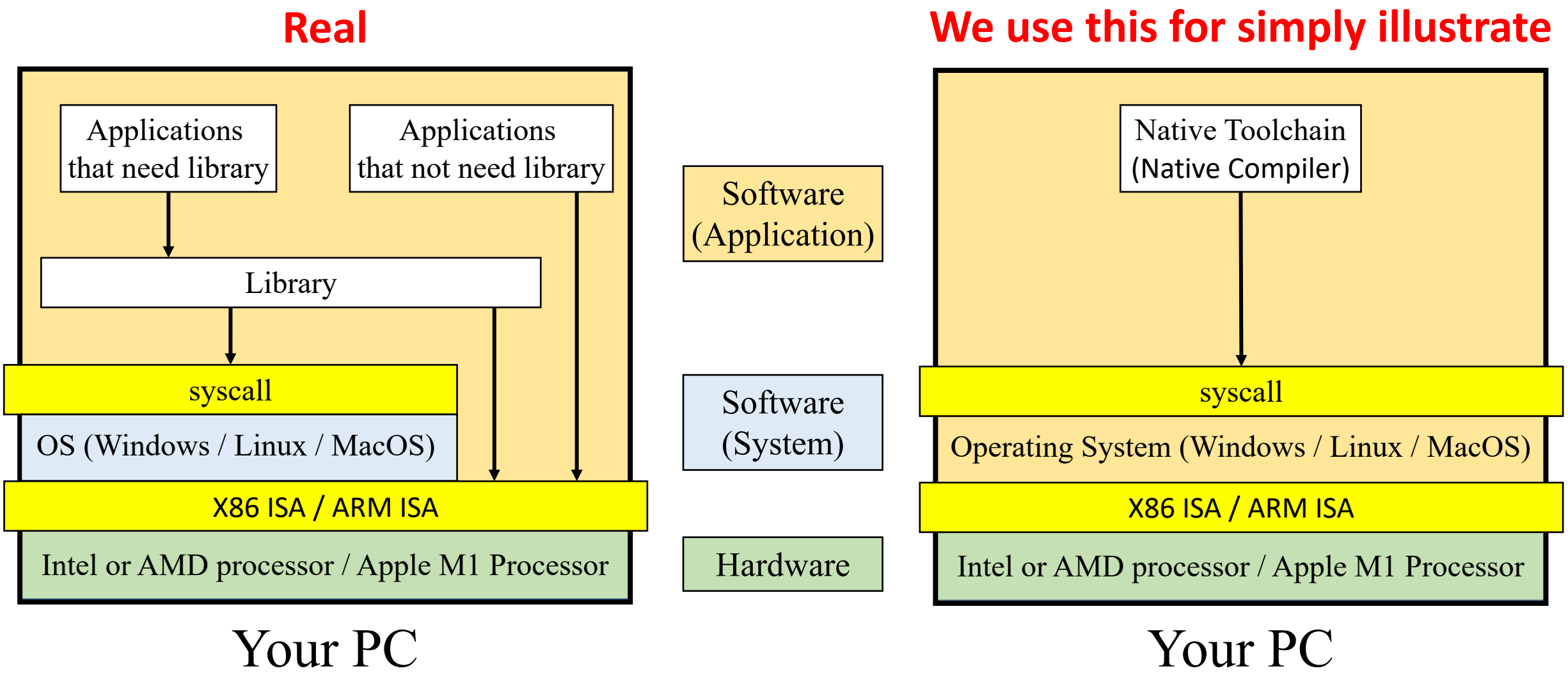
RISC-V ISA Simply Introduction

- RISC-V (pronounced “risk-five”) is started by students from UC Berkeley in May 2010.
- The stability of RISC-V is maintained by RISC-V Foundation.
- **Feature**
  - Free & Open Source
  - Simple & Elegant
  - Modulization & Stable & Extensibility
  - Customizable
  - Good Ecosystem

We focus on this

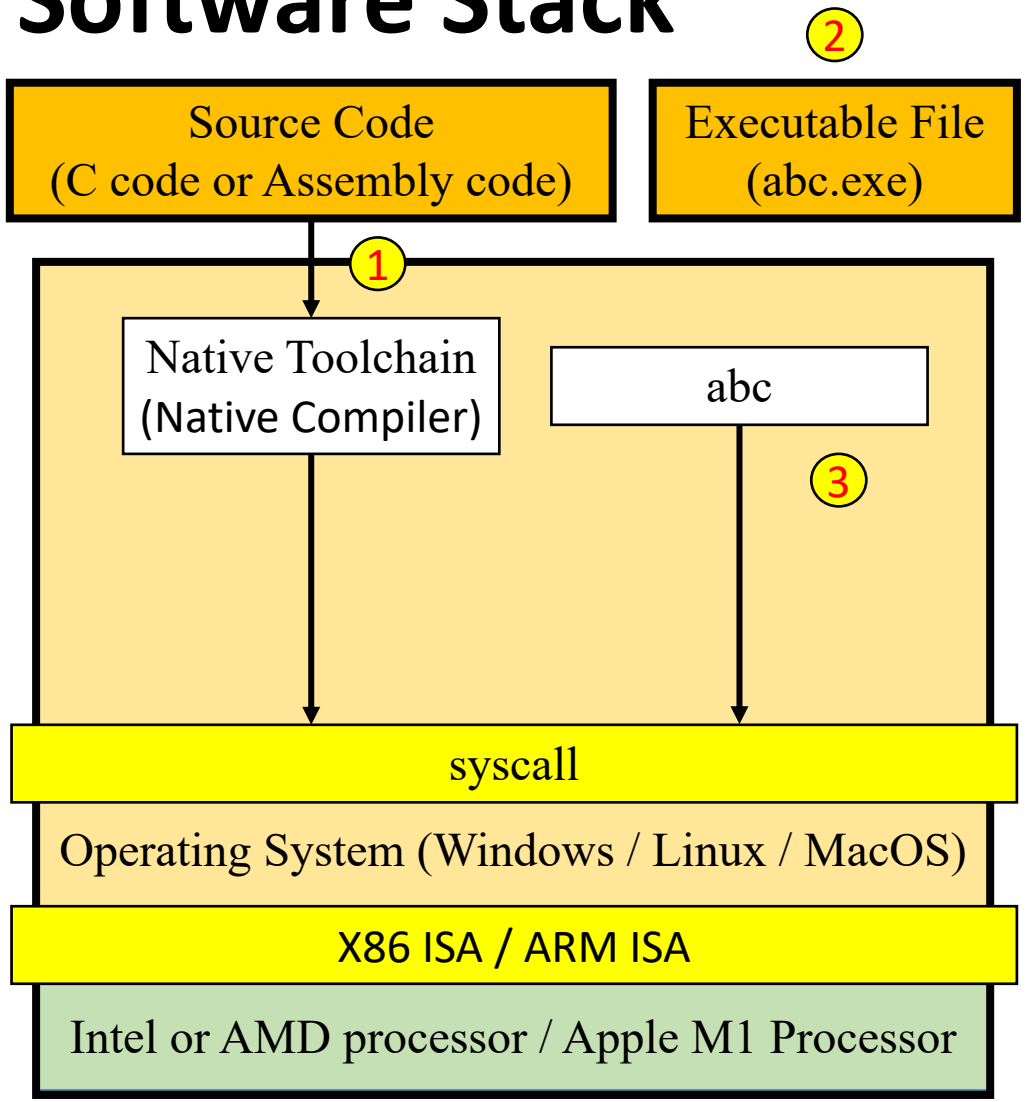
Base	Version	Status
RVWMO	2.0	**Ratified**
RV32I	**2.1**	**Ratified**
RV64I	**2.1**	**Ratified**
RV32E	*1.9*	*Draft*
RV128I	*1.7*	*Draft*
Extension	**Version**	**Status**
M	**2.0**	**Ratified**
A	**2.1**	**Ratified**
F	**2.2**	**Ratified**
D	**2.2**	**Ratified**
Q	**2.2**	**Ratified**
C	**2.0**	**Ratified**
Counters	*2.0*	*Draft*
L	*0.0*	*Draft*
B	*0.0*	*Draft*
J	*0.0*	*Draft*
T	*0.0*	*Draft*
P	*0.2*	*Draft*
V	*0.7*	*Draft*
Zicsr	**2.0**	**Ratified**
Zifencei	**2.0**	**Ratified**
Zihintpause	**2.0**	**Ratified**
Zam	*0.1*	*Draft*
Zfh	*0.1*	*Draft*
Zfhmin	*0.1*	*Draft*
Zfinx	*1.0*	*Frozen*
Zdinx	*1.0*	*Frozen*
Zhinx	*1.0*	*Frozen*
Zhinxmin	*1.0*	*Frozen*
Ztso	*0.1*	*Frozen*

Software Stack



Software Stack

1. Compile source files using native compiler
2. Generate executable file (abc.exe)
3. OS loads abc.exe into DRAM, and abc.exe becomes the application process



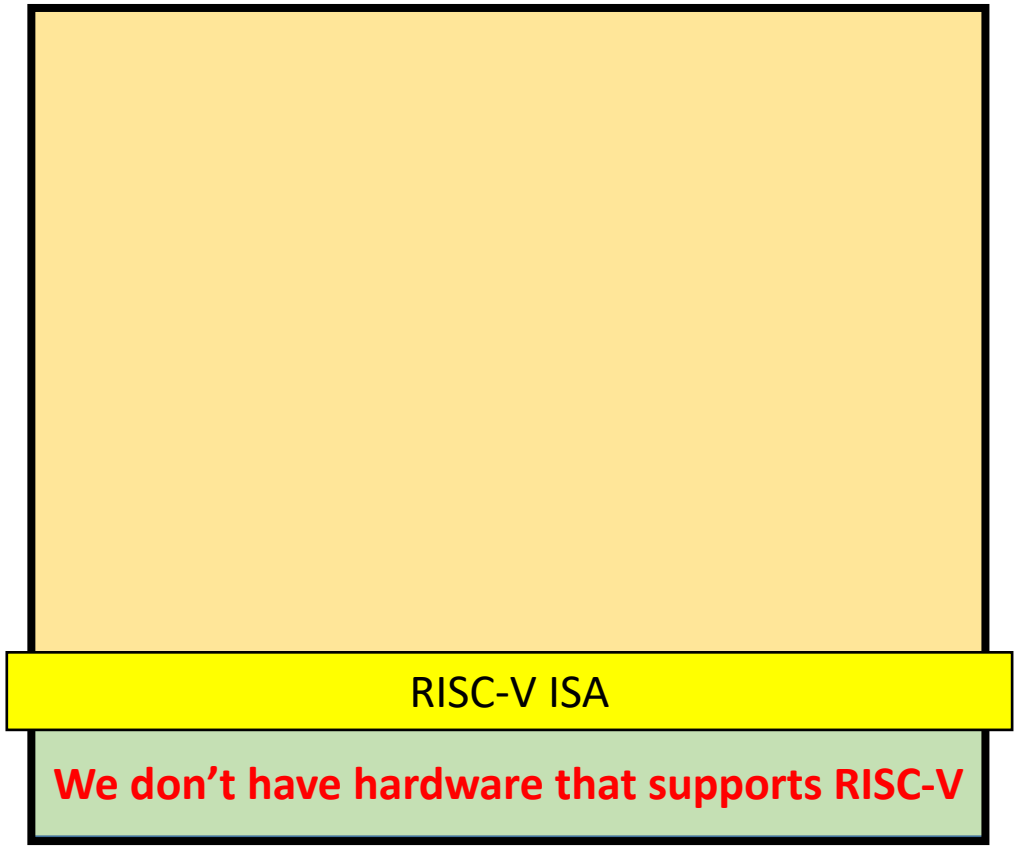
Your PC

Software (Application)

Software (System)

Hardware

In this lecture, we want to learn RISC-V ISA



Machine 2

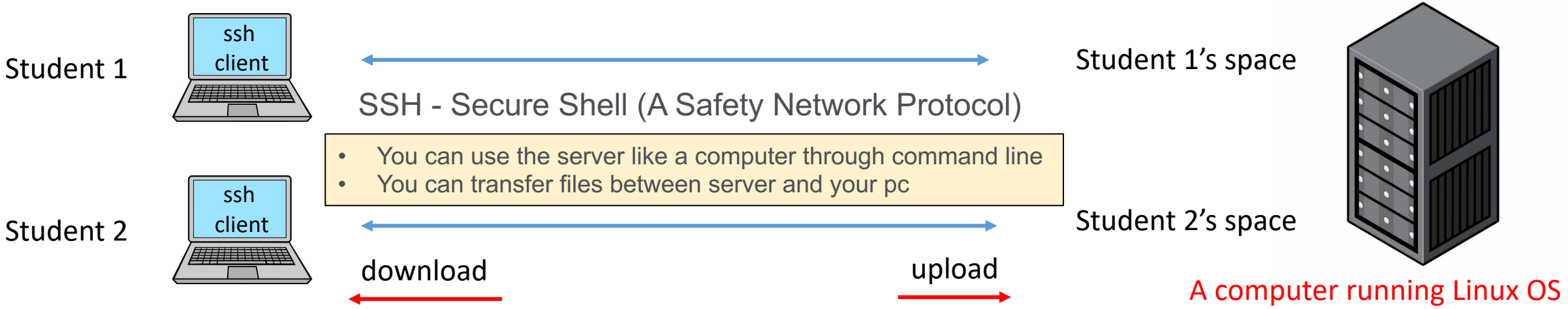
Use Software to simulate Hardware

- Because we don't have hardware that supports RISC-V ISA
  - So, we need to use the software to simulate the hardware that supports RISC-V ISA

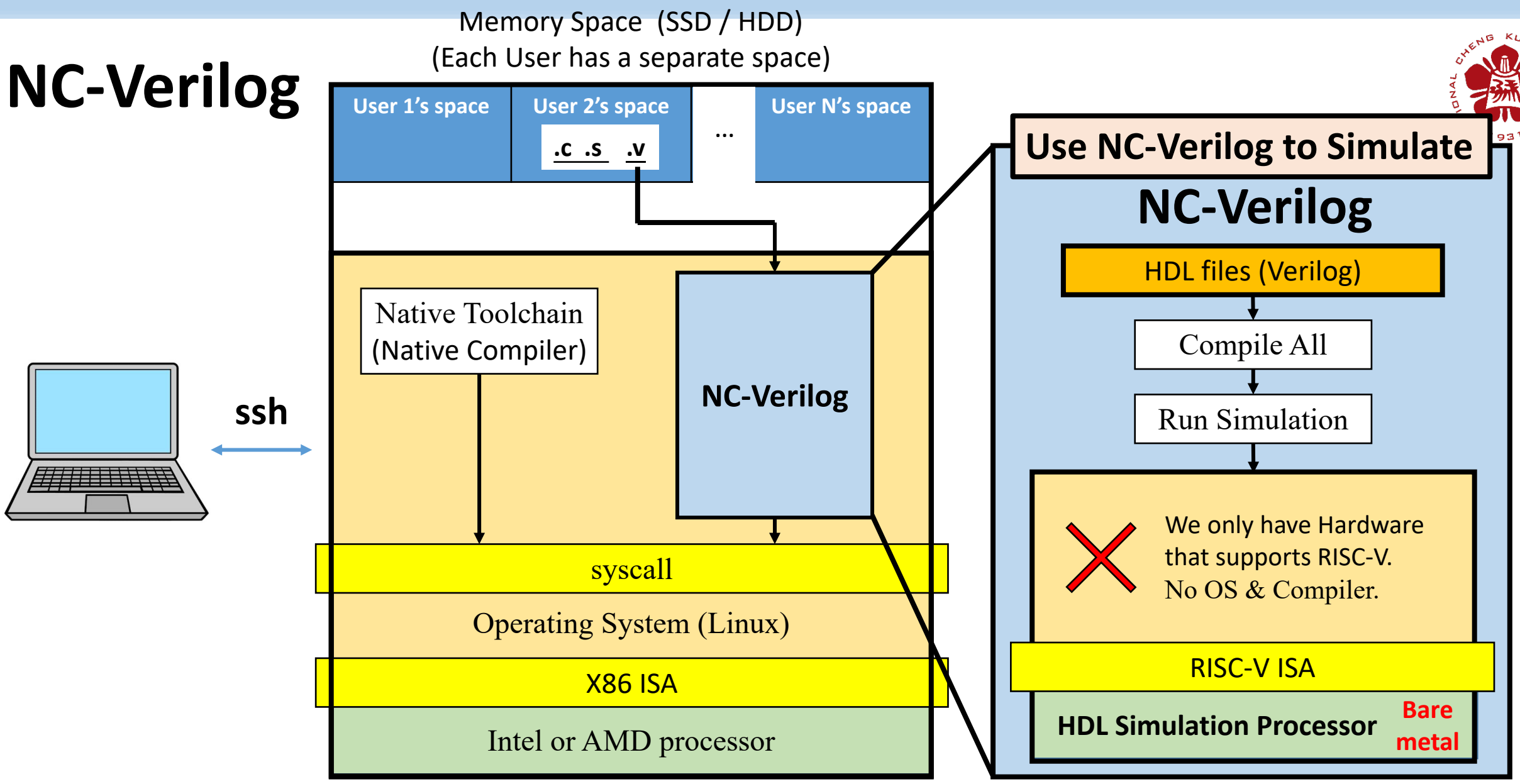
- The simulation software we used : **Cadence NC-Verilog**
  - It can simulate hardware through **Hardware Description Language(硬體描述語言, HDL)**
    - **HDL** : VHDL / Verilog / System-Verilog

We focus on this

- But NC-Verilog needs to run on Linux OS
  - So, we provide a server running Linux OS for you to connect to and use it !!!



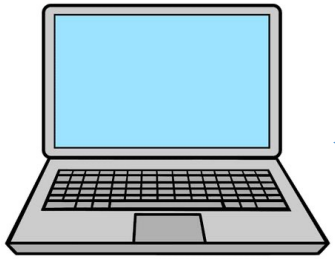
NC-Verilog


Memory Space (SSD / HDD)
(Each User has a separate space)
User 1's space
User 2's space
.c .s .v
...
User N's space
Native Toolchain
(Native Compiler)
NC-Verilog
syscall
Operating System (Linux)
X86 ISA
Intel or AMD processor
ssh
Server
Use NC-Verilog to Simulate
NC-Verilog
HDL files (Verilog)
Compile All
Run Simulation
We only have Hardware
that supports RISC-V.
No OS & Compiler.
RISC-V ISA
HDL Simulation Processor
Bare metal
Machine 2
(NC-Verilog Simulation)

How to Debug our design ?

1. Run some test program

How to produce program ?



ssh

Memory Space (SSD / HDD)
(Each User has a separate space)

User 1's space | User 2's space | … | User N's space

.c .s .v

Native Toolchain
(Native Compiler)

NC-Verilog

syscall

Operating System (Linux)

X86 ISA

Intel or AMD processor

Server

Use NC-Verilog to Simulate

NC-Verilog

HDL files (Verilog)

Compile All

Run Simulation

program

???

We only have Hardware that supports RISC-V.
No OS & Compiler.

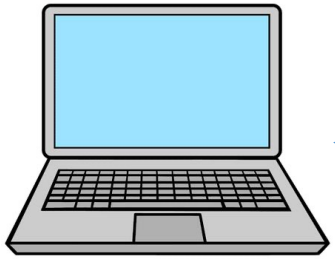
RISC-V ISA

HDL Simulation Processor

Bare metal

Machine 2
(NC-Verilog Simulation)

RISC-V Toolchain



ssh

Memory Space (SSD / HDD)
(Each User has a separate space)

User 1's space | User 2's space (.c .s .v) | … | User N's space

Native Toolchain (Native Compiler)

RISC-V Toolchain (Cross Compiler)

NC-Verilog
program ???
RISC-V ISA

syscall

Operating System (Linux)

X86 ISA

Intel or AMD processor

Server

Use NC-Verilog to Simulate

NC-Verilog

HDL files (Verilog / VHDL)

Compile All

Run Simulation

program ???

We only have Hardware that supports RISC-V.
No OS & Compiler.

RISC-V ISA

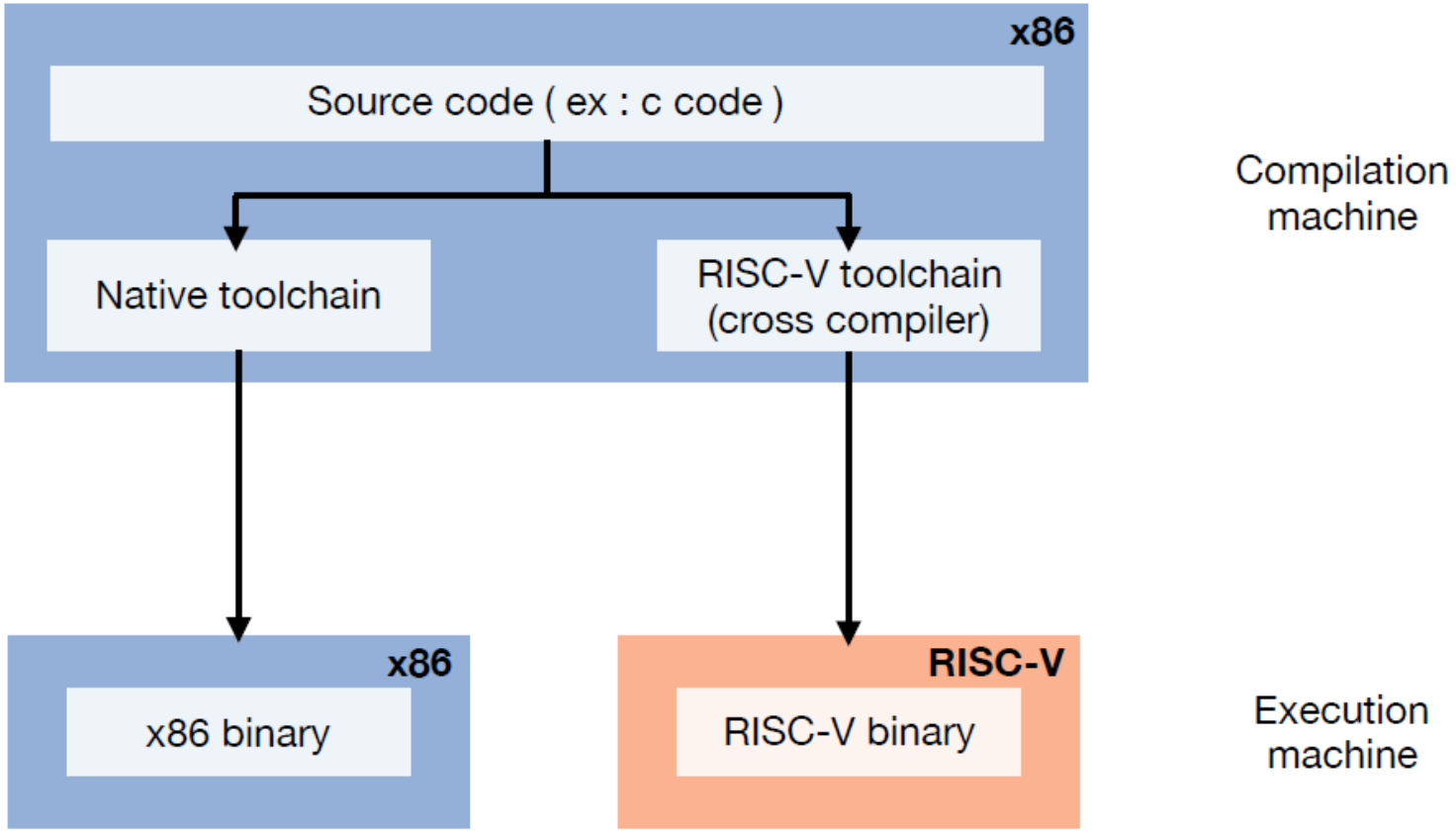
HDL Simulation Processor

Bare metal

Machine 2
(NC-Verilog Simulation)

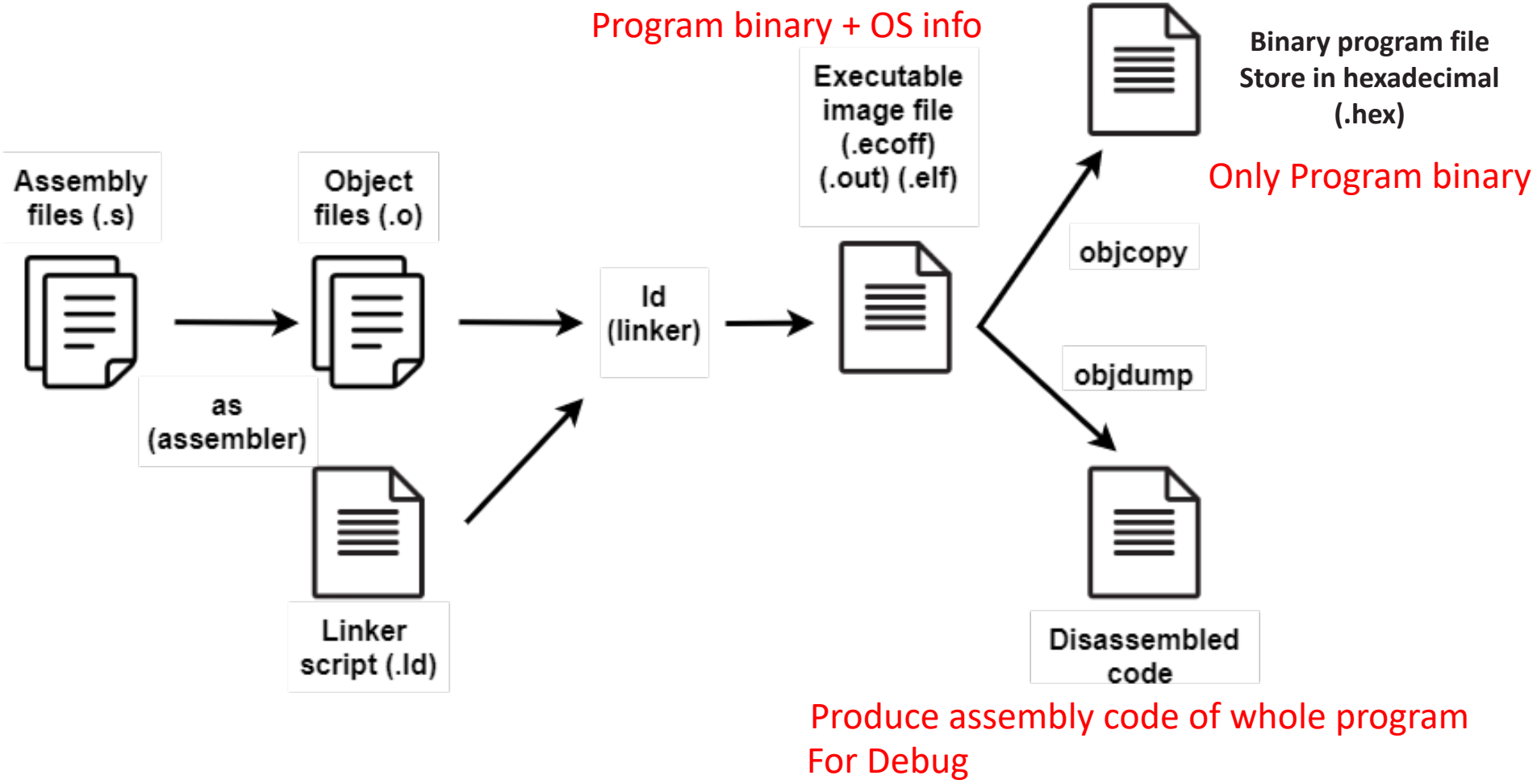
RISC-V Cross Compiler (RISC-V Toolchain)

- A native compiler can only generate executable programs that are identical to the ISA it is running on
- If you want to generate executable programs with different ISA from compiler environment
  => You needs to use cross compiler instead of native compiler

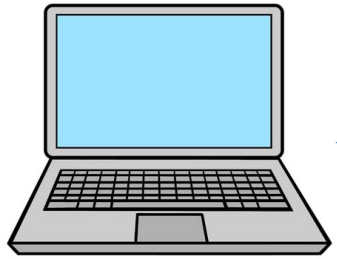


RISC-V Toolchain WorkFlow

- We use the open-source **RISC-V GNU Compiler Toolchain** maintained by **RISC-V Software Collaboration**



RISC-V Toolchain



ssh

1. Cross-compile source files using RISC-V Toolchain
2. Generate .hex
3. Simulate with NC-Verilog

Memory Space (SSD / HDD)
(Each User has a separate space)

User 1's space | User 2's space | ... | User N's space

.c .s .v .hex

1 2 3

Native Toolchain (Native Compiler)

RISC-V Toolchain (Cross Compiler)

NC-Verilog

program ???

RISC-V ISA

syscall

Operating System (Linux)

X86 ISA

Intel or AMD processor

Server

Use NC-Verilog to Simulate

NC-Verilog

HDL files (Verilog / VHDL)

Compile All

Run Simulation

program ???

We only have Hardware that supports RISC-V.
No OS & Compiler.

RISC-V ISA

HDL Simulation Processor

Bare metal

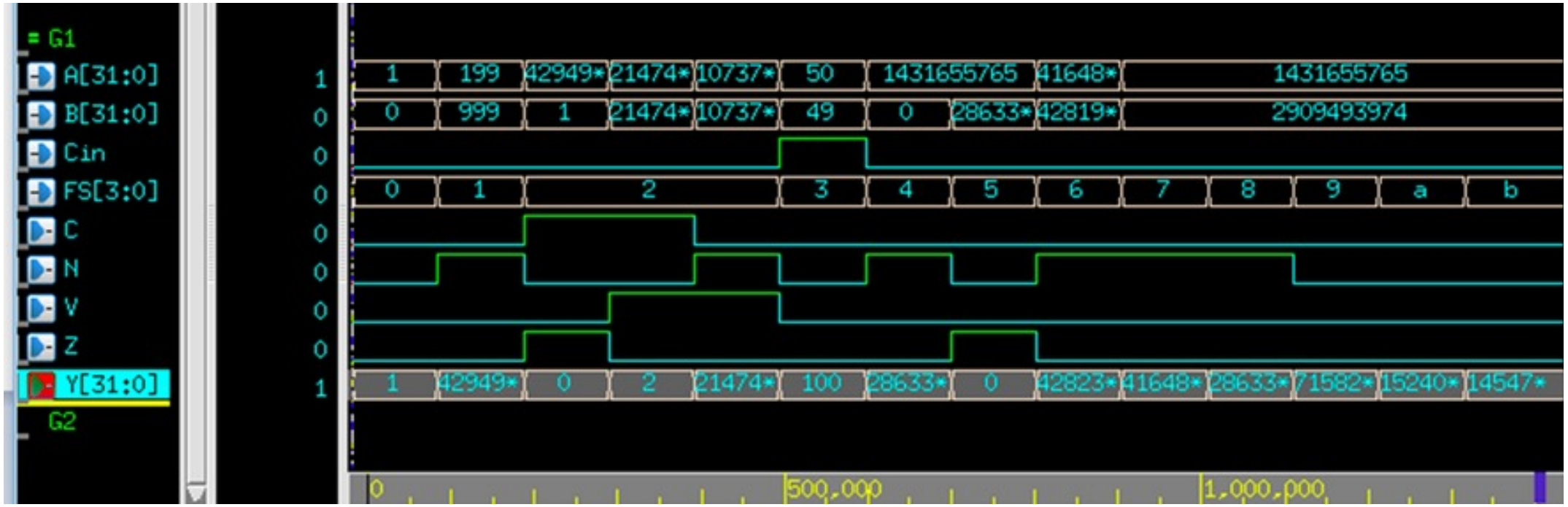
Machine 2
(NC-Verilog Simulation)

How to Debug our design ?

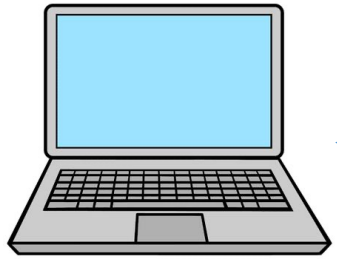
1. **Run some test program** (What to do if the result is wrong ?)

2. **Check whether all signals are always correct => Check Signal Waveform**

   - Synopsys provides a waveform viewer called “nWave” to check the signals



nWave



ssh

1. Simulate with NC-Verilog
2. Generate .fsdb
3. Debug with nWave

Memory Space (SSD / HDD)
(Each User has a separate space)

User 1's space

User 2's space (2)
.c .s .v .hex .fsdb

...

User N's space

(1) (3)

Native Toolchain (Native Compiler)

RISC-V Toolchain (Cross Compiler)

NC-Verilog
.hex
RISC-V ISA

nWave

syscall

Operating System (Linux)

X86 ISA

Intel or AMD processor

Server

- Verilog is usually just at RTL level
- It just simulates the behavior & functionality
- It doesn't include timing delay simulation
- It's not real hardware yet !

IC Design Flow

High-Level

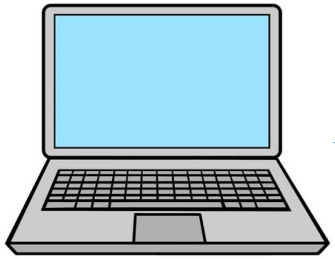
Text	Specification		
Software	Behavior Level	Input – Output Check	Function Verification
Block Diagram Split Block	Architecture Level		**We focus on this part**
HDL Abstract Behavior	RTL Level	Pre-Simulation	Function Verification
Front end Gate	Gate Level	Post-Simulation	Function + Timing Verification
Back end Transistors	Transistor Level	Pre-Simulation(HSpice)	
Physics	Layout	DRC、LVS、PEX	

Logic Systhesis (合成) : generate netlist (standard gate connectivity)

Low-Level

Logic Synthesis

- Synopsys provides a logic synthesis tool called "design compiler" to do synthesize



ssh

1. Logic synthesis for pre.v using Design Compiler
2. Generate post.v
3. Simulate with NC-Verilog
4. Generate post.fsdb
5. Debug with nWave

Memory Space (SSD / HDD)
(Each User has a separate space)

User 1's space | User 2's space | ... | User N's space

.v .hex post.v post.fsdb

Native Toolchain (Native Compiler)

RISC-V Toolchain (Cross Compiler)

NC-Verilog
.hex
RISC-V ISA

nWave

Design Compiler (Logic Synthesis)

syscall

Operating System (Linux)

X86 ISA

Intel or AMD processor

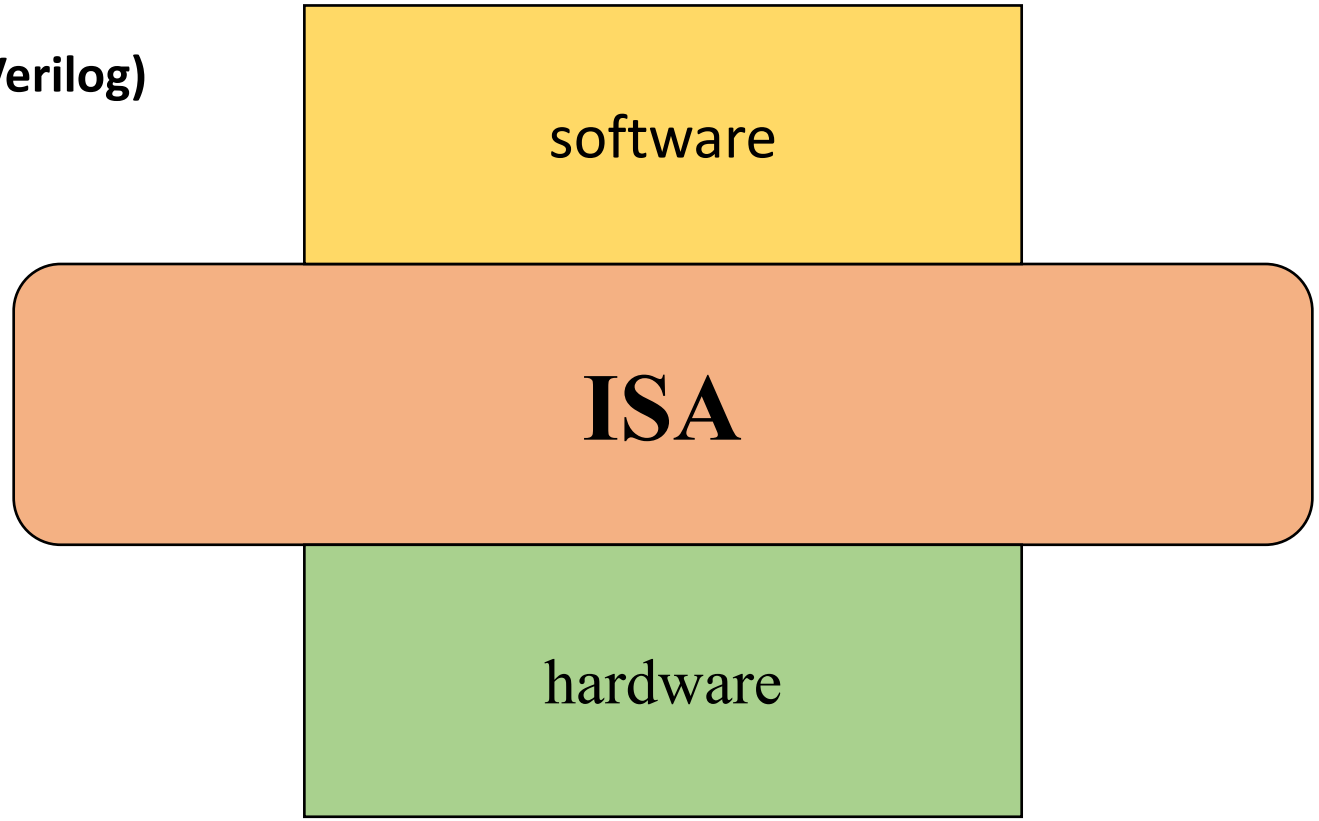
Server

Prepare for implementing a CPU

1. **Learn ISA**
   - RISC-V Base ISA - RV32I

2. **Learn Hardware Description Language (e.g. Verilog)**
   - Design
     - Combinational circuit
     - Sequential circuit
   - Test & Debug
     - Testbench
     - Waveform
   - Logic Synthesis

3. **Learn CPU Micro-Architecture**
   - Single-Cycle CPU
   - Pipeline CPU
     + Branch Prediction
     + Cache



Course Gantt

Week	Date	Lecture	Lab
1	9/7	Introduction	
2	9/14	Computer Abstractions and Technology	Lab1: Introduction and Environment Setup
3	9/21	Instructions: Language of the Computer	Lab2: Assembly 1 - RV32I
4	9/28	Instructions: Language of the Computer	Lab3: Assembly 2 - RV32I
5	10/5	Instructions: Language of the Computer	
6	10/12	Arithmetic for Computers	Lab4: Verilog 1 - Testbench + Combinational
7	10/19	Arithmetic for Computers	Lab5: Verilog 2 - Sequential
8	10/26	Arithmetic for Computers	Lab6: Verilog 3
9	11/2	Midterm Exam	
10	11/9	The Processor	Lab7: Single-Cycle CPU
11	11/16	The Processor	
12	11/23	The Processor	Lab8: Pipeline CPU
13	11/30	Memory Hierarchy	
14	12/7	Memory Hierarchy	Lab9: Branch Prediction
15	12/14	Memory Hierarchy	
16	12/21	Parallel Processors	Lab10: Cache
17	12/28	Final Exam	
18	1/4		