Hello, World

Things All Developers (and Other People) Should Know

Chapter Information

Learning Goals

1. Learn how to use GitHub
2. Learn how to install and use IntelliJ
3. Learn how to write an Hello World application in Java, run it from IntelliJ, from the command line and debug it
4. How to read from console using Scanner
5. How to do basic String operations
6. How to use variables and constants
7. Do Loop
8. Learn some of the basics of computer design and architecture
9. Comments and when to use them (and when NOT to use them)
10. Java Packages, versions

Final Project

Animation:
Show a single line animation on the console

Tools to Install

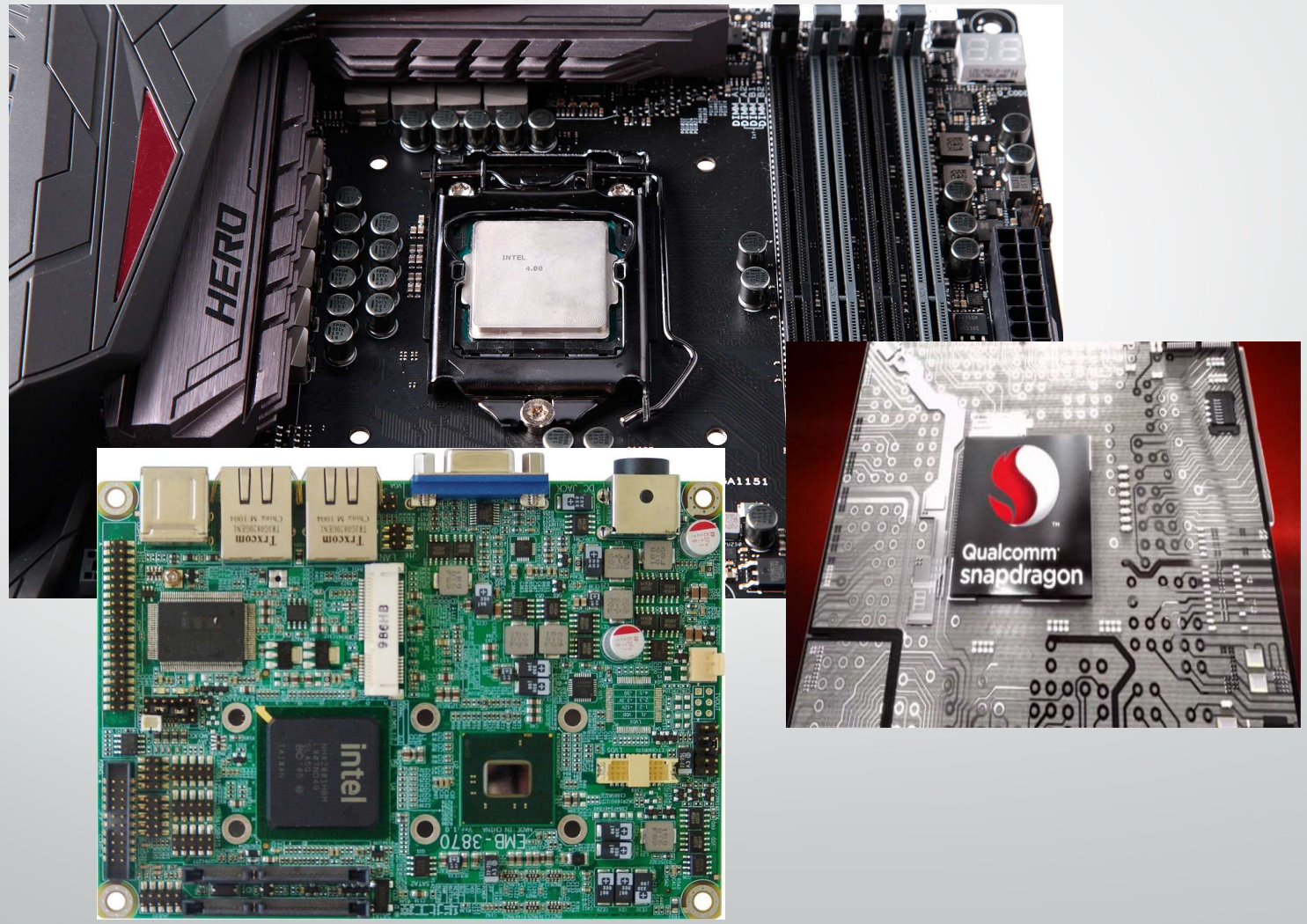
1. Java SDK version 8 unless your team works with 11
2. IntelliJ (from JetBrains or local repository)
3. Create GitHub account
4. Git (from GitHub)
5. TortoiseGit (makes it easy to view repositories in Explorer)
6. Notepad++ (quick editor with syntax highlighting)

Computer Hardware

- **CPU**
- **Assembly Language**
- **Registers & Memory**
- **Virtual Memory**
- **Von Neumann Model**
- **I/O and Interrupts**
  - Disk I/O
  - Graphics
  - Networking

CPU

- Central Processing Unit
- Sits on Motherboard and controls most everything
- Processes ***Machine Instructions***
  - Recipes with predefined steps
    - Compare
    - Read and Write to memory
    - Jump around
    - Arithmetic


HERO
INTEL
4.00
A1151
Trxcom
Trxcom
intel
EMB-3870
Qualcomm
snapdragon

Assembly Language

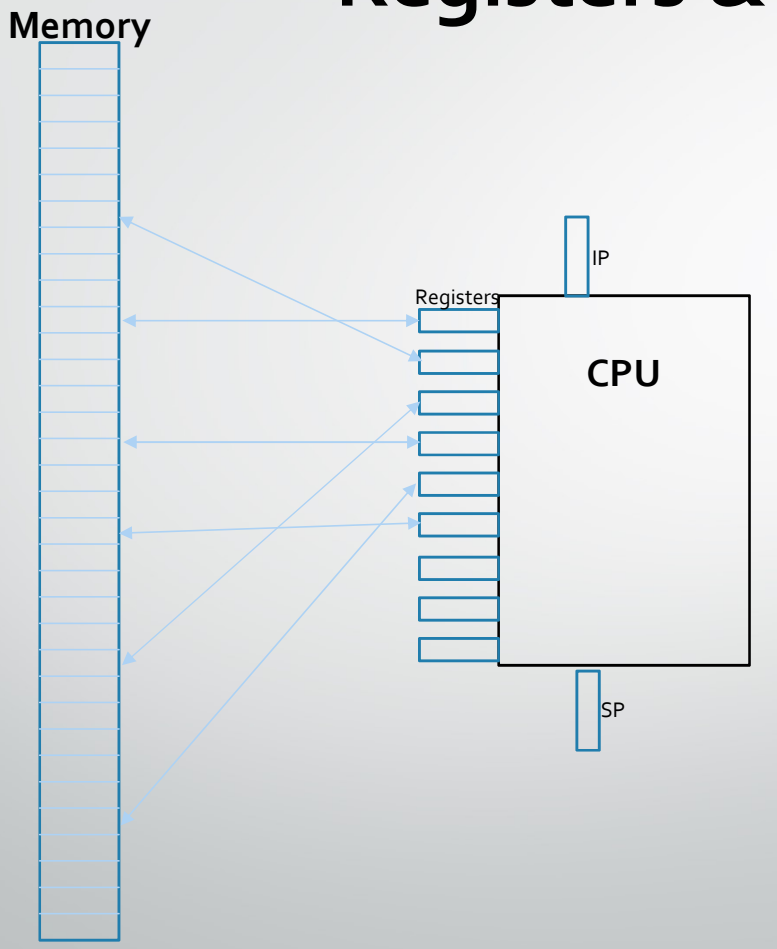
- A very primitive programming language
  - Instructions that directly translate to machine language
  - Macros
- Very efficient
- Very hard to write long meaningful programs
- CPU specific
- Assembler: Assembly -> Machine Code


```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2


C000                    ORG     ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START    LDS     #STACK

                ***************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

0013            RESETA  EQU     %00010011
0011            CTLREG  EQU     %00010001

C003 86 13      INITA   LDA A   #RESETA   RESET ACIA
C005 B7 80 04           STA A   ACIA
C008 86 11              LDA A   #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04           STA A   ACIA

C00D 7E C0 F1           JMP     SIGNON    GO TO START OF MONITOR

                ***************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04  INCH     LDA A   ACIA      GET STATUS
C013 47                 ASR A             SHIFT RDRF FLAG INTO CARRY
C014 24 FA              BCC     INCH      RECIEVE NOT READY
C016 B6 80 05           LDA A   ACIA+1    GET CHAR
C019 84 7F              AND A   #$7F      MASK PARITY
C01B 7E C0 79           JMP     OUTCH     ECHO & RTS

                ***************************************
                * FUNCTION: INHEX - INPUT HEX DIGIT
                * INPUT: none
                * OUTPUT: Digit in acc A
                * CALLS: INCH
                * DESTROYS: acc A
                * Returns to monitor if not HEX input

C01E 8D F0      INHEX   BSR     INCH      GET A CHAR
C020 81 30              CMP A   #'0       ZERO
C022 2B 11              BMI     HEXERR    NOT HEX
C024 81 39              CMP A   #'9       NINE
C026 2F 0A              BLE     HEXRTS    GOOD HEX
C028 81 41              CMP A   #'A
C02A 2B 09              BMI     HEXERR    NOT HEX
C02C 81 46              CMP A   #'F
C02E 2E 05              BGT     HEXERR
C030 80 07              SUB A   #7        FIX A-F
C032 84 0F      HEXRTS  AND A   #$0F      CONVERT ASCII TO DIGIT
C034 39                 RTS

C035 7E C0 AF  HEXERR   JMP     CTRL      RETURN TO CONTROL LOOP
```

Registers & Memory

- Memory – holds binary data
- Manipulated by CPU
  - In Registers
  - Directly
- Registers
  - How CPU manipulates data
  - Special CPUs
- CPU Cache

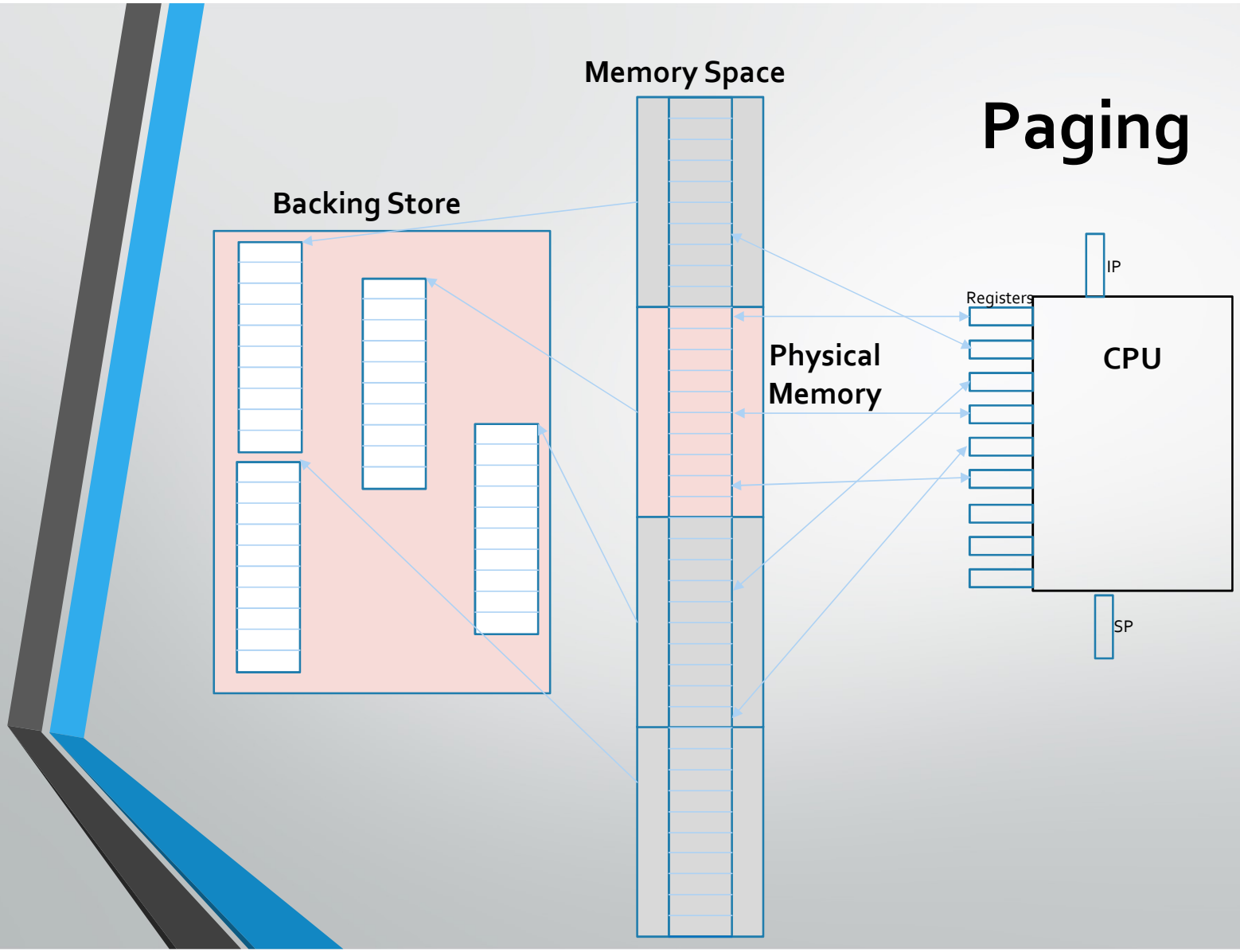
Registers & Memory



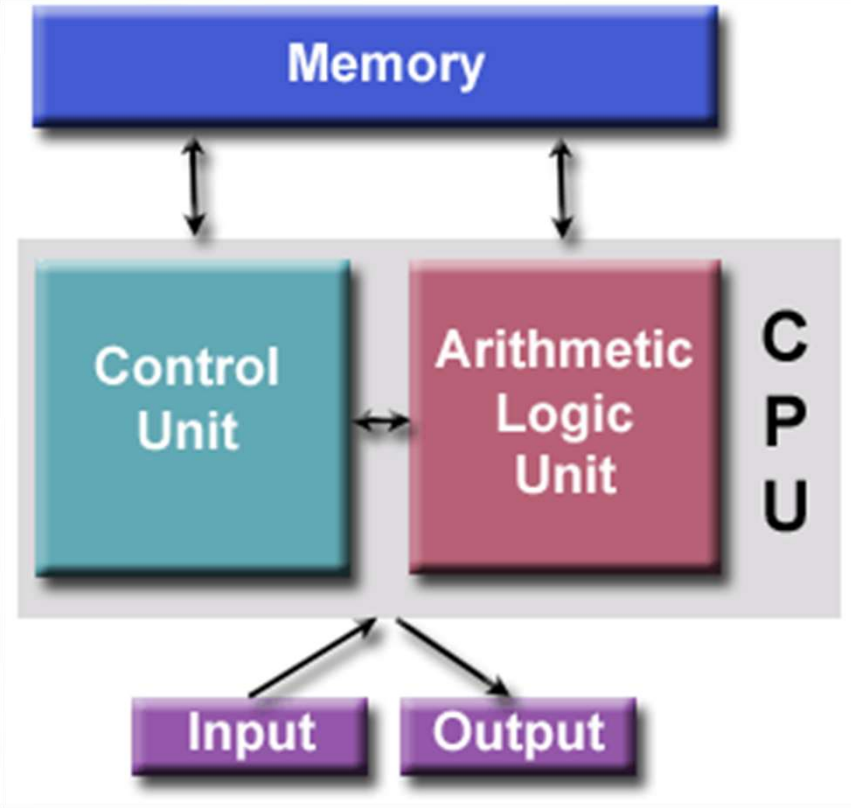
Virtual Memory

- Larger than life
- Page tables

Paging


Memory Space
Backing Store
Physical Memory
Registers
IP
CPU
SP

Von Neumann Model



I/O and Interrupts

- Interrupts
- Memory mapped I/O
- Devices
  - Graphics
  - Disks
  - Network

Procedural Programming

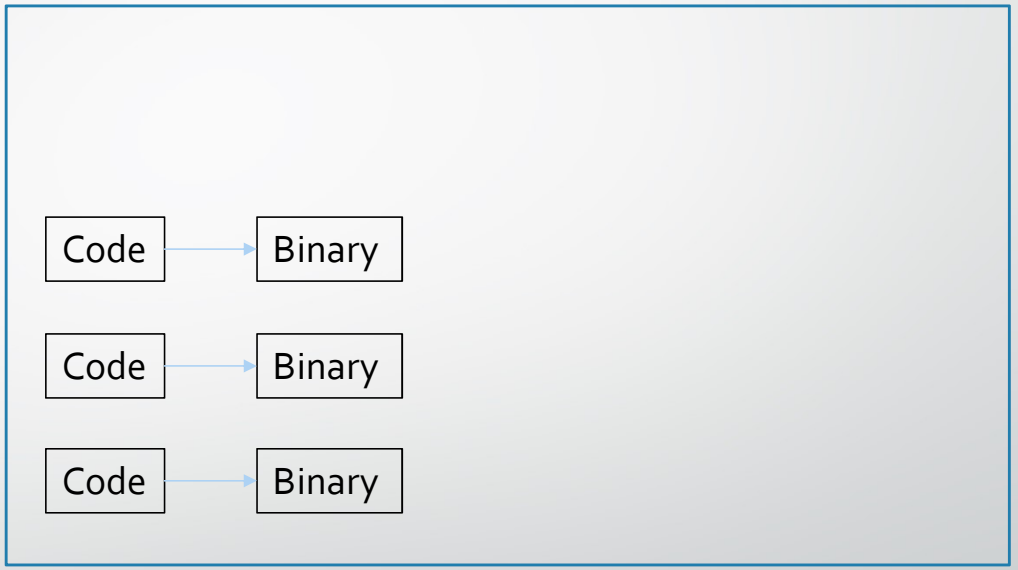
- **Some History**
- **What is Compiling**
  - Creating Binaries
- **What is Linking**
  - Putting things together
- **What is Loading**
  - Putting things where CPU can get them
- **Executing code**
  - Starting the program
- **Procedures and calling**
- **Basic Procedural Programming Constructs**

History

- 60's (Algol, ForTran, BASIC)
- 70's (Pascal, C)
- 80's (Ada)

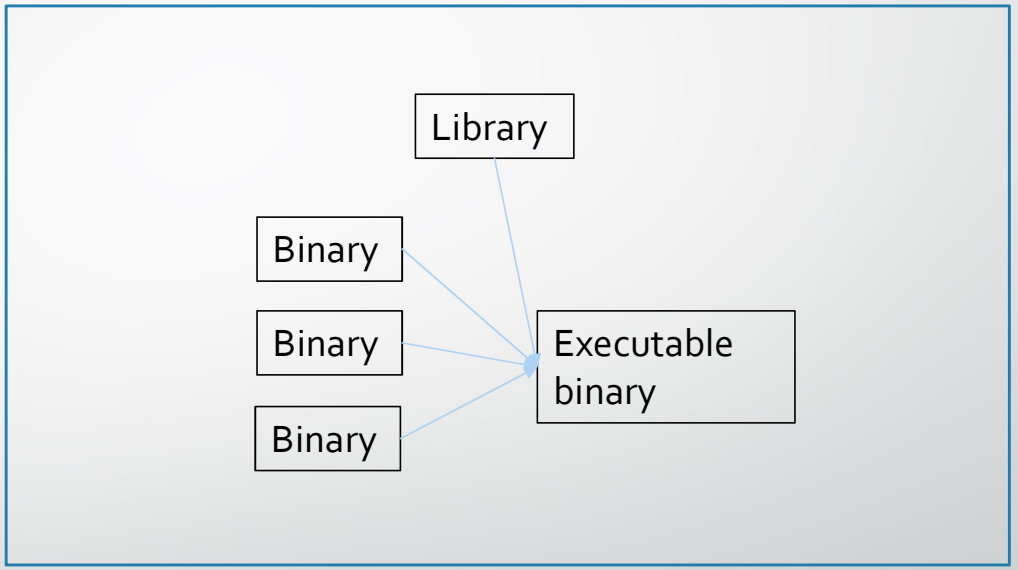
Compile

- Turn text (code) to binary



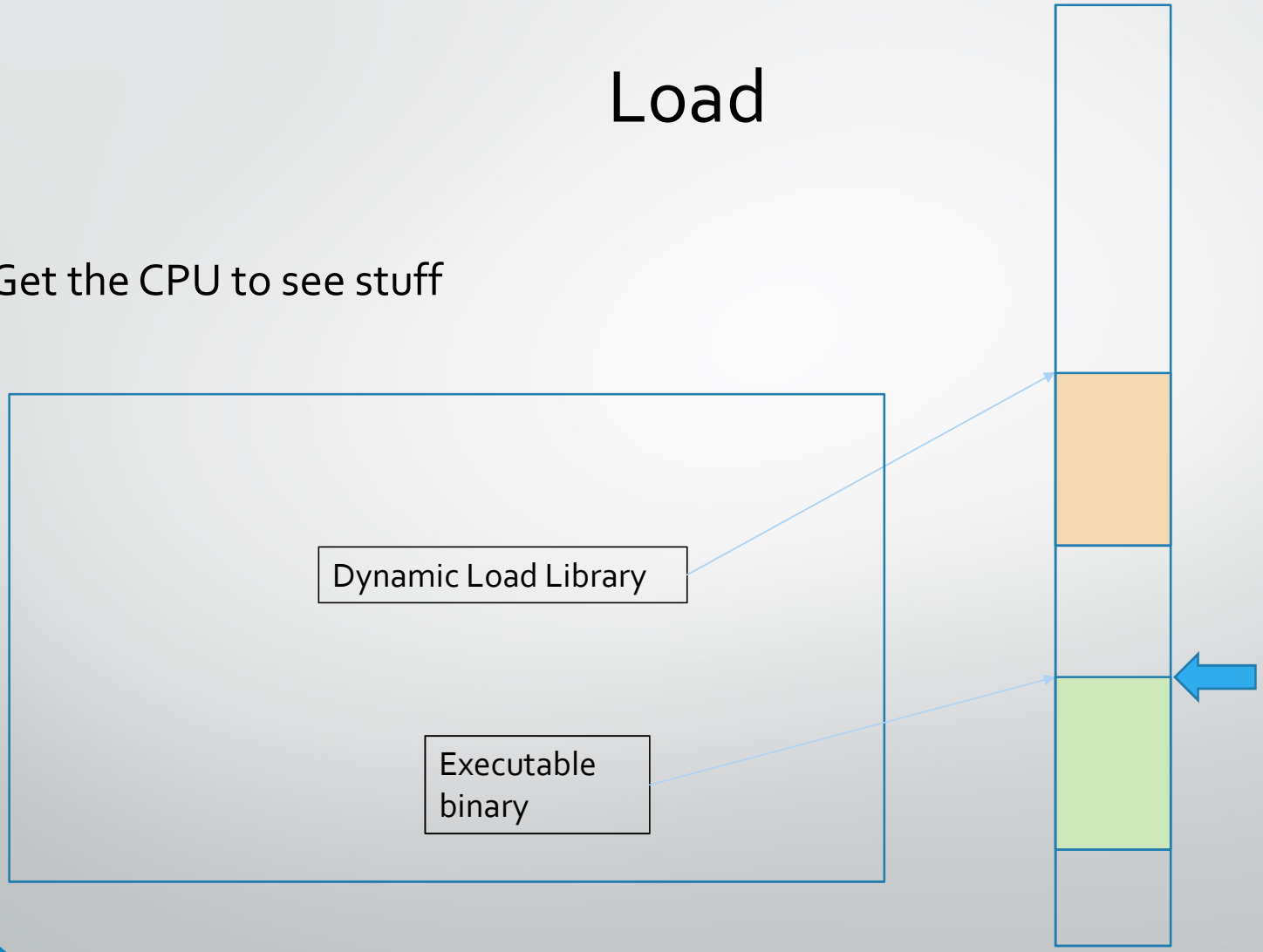
Link

- Bring things together

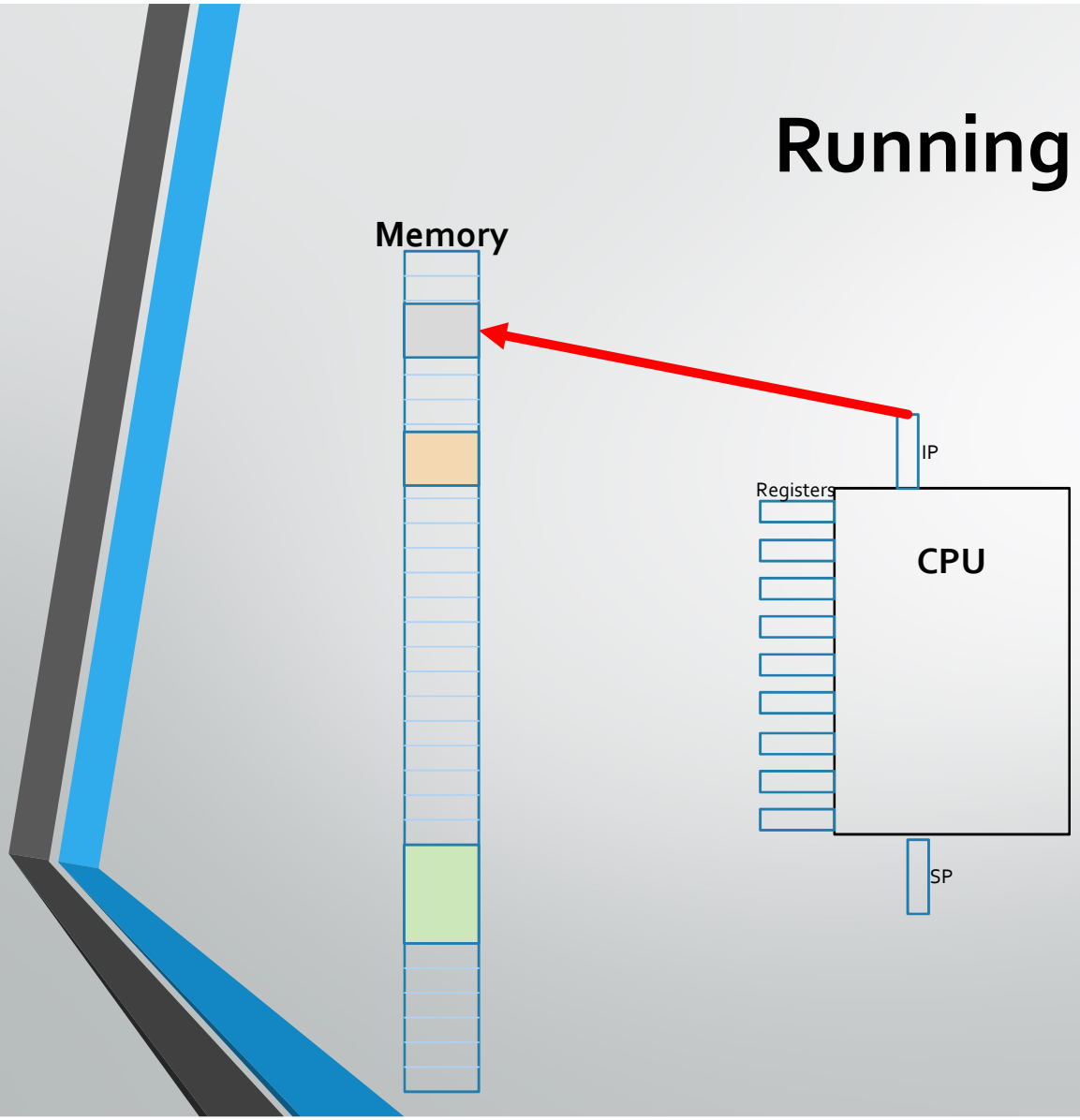


Load

- Get the CPU to see stuff



Running – The IP



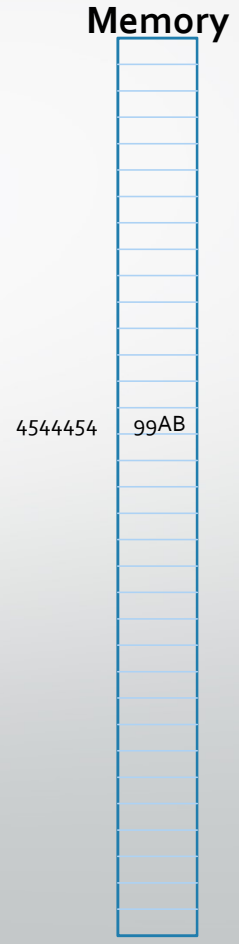
Notation

- X = reference to X
  - Address
  - Register
- (X) = content of X

E.g.

R1, 4544454

(R1), (4544454)



Fetch Process

- Fetch the next instruction

  CPU ←((IP))

  (IP) = (IP) + 8

- Process CPU

Example

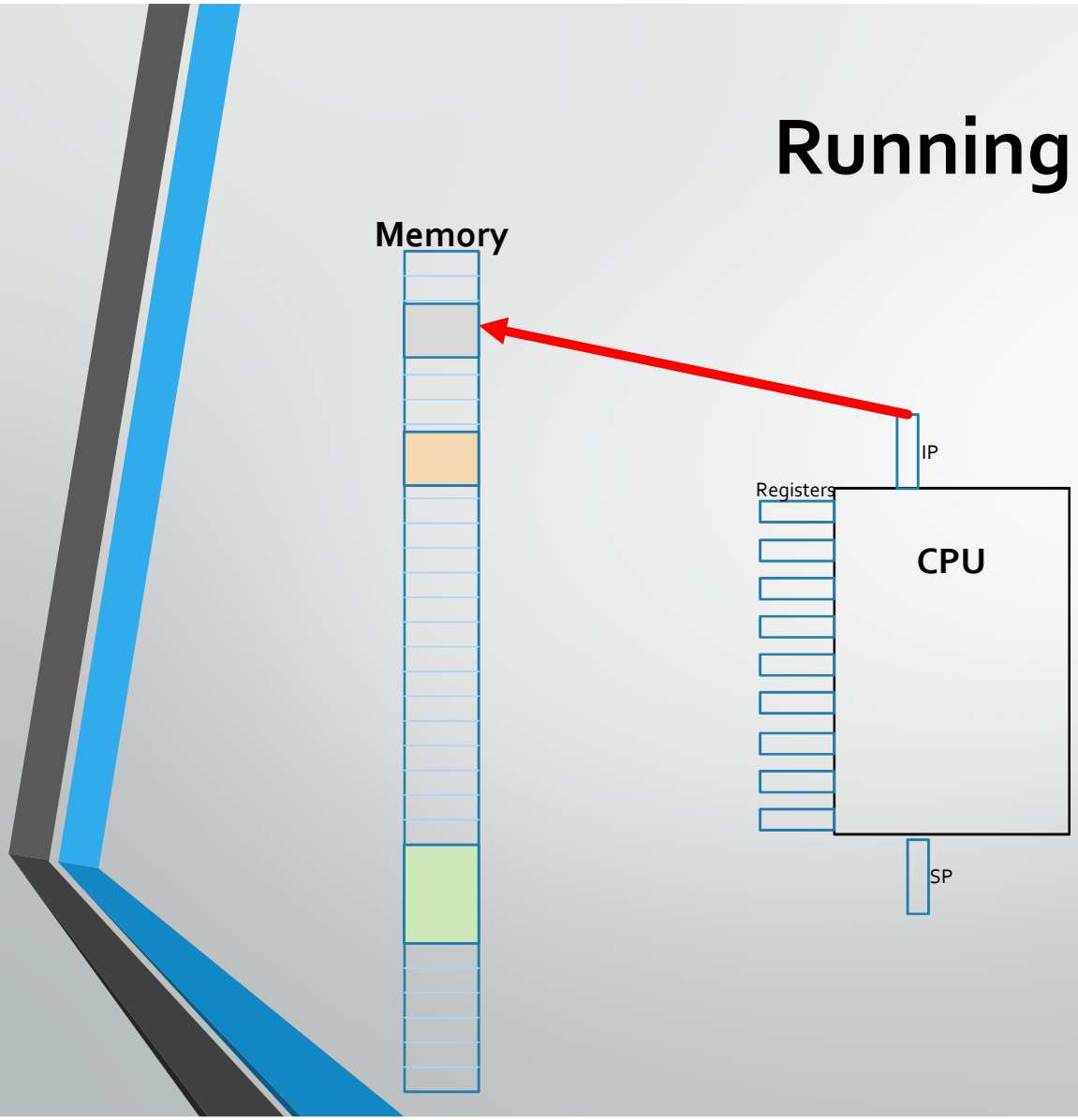
- R1 = 4000016
- IP = 4000000

4000000	Mov R1,IP
4000008	Mov, 3, R1
4000016	Mov, 2, R1

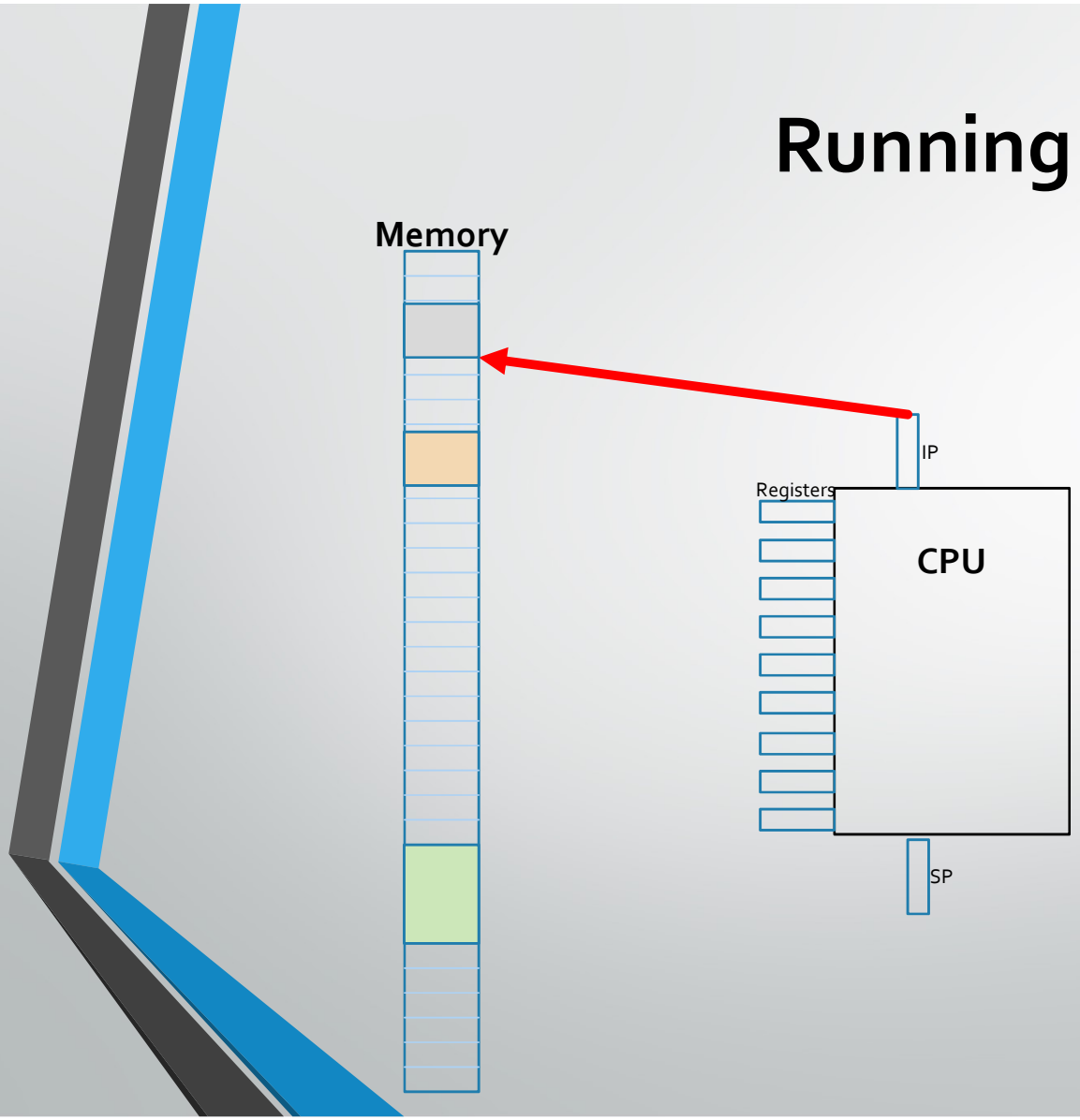
What is IP, R1 after FETCH?

- Fetch the next instruction
  CPU ←((IP))
  (IP) = (IP) + 8
- Process CPU

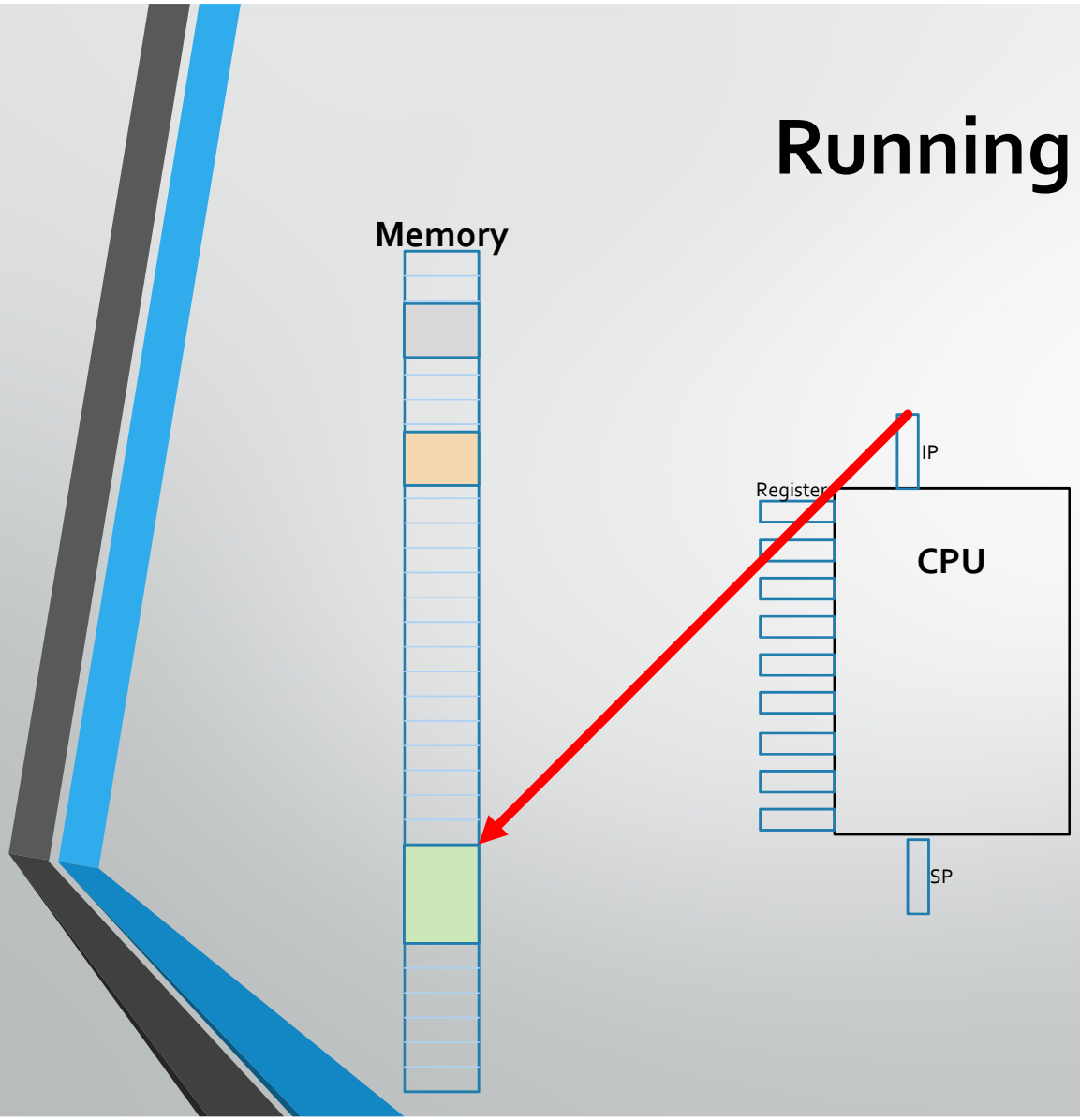
Running – The IP



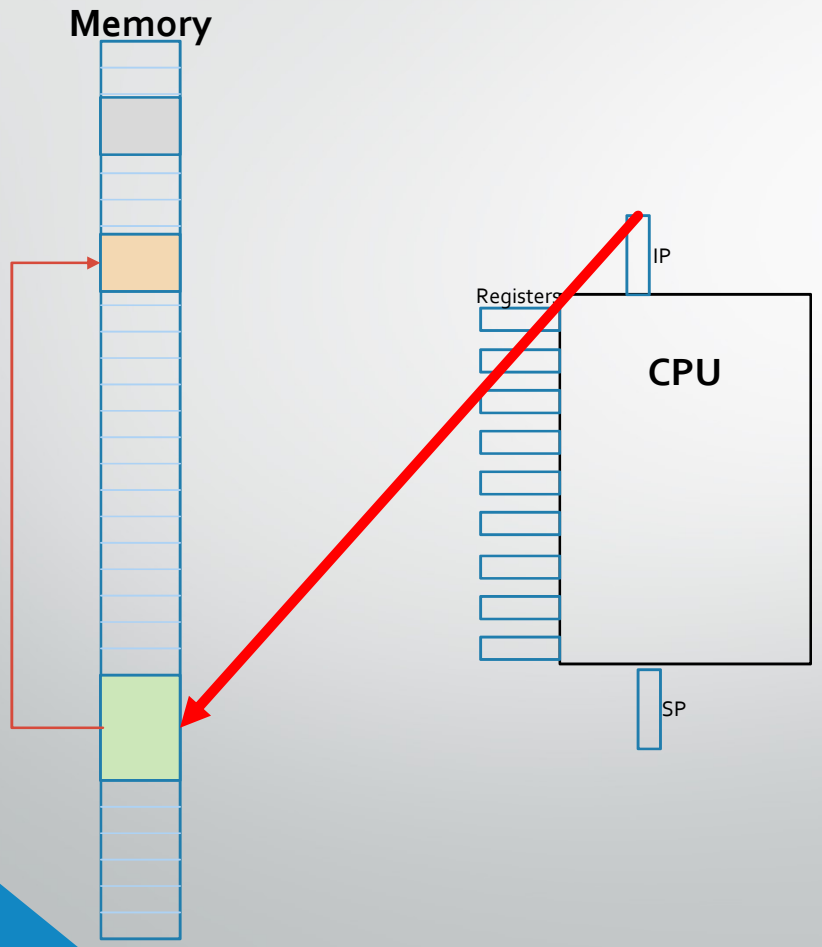
Running – The IP



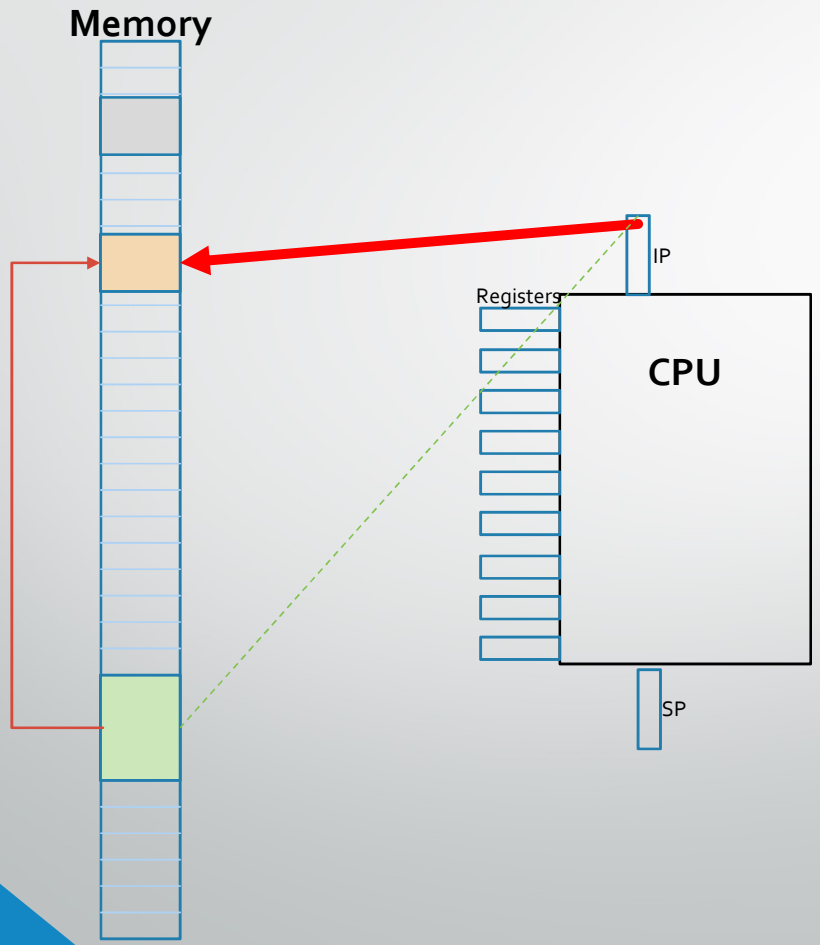
Running – The IP



Running – The IP



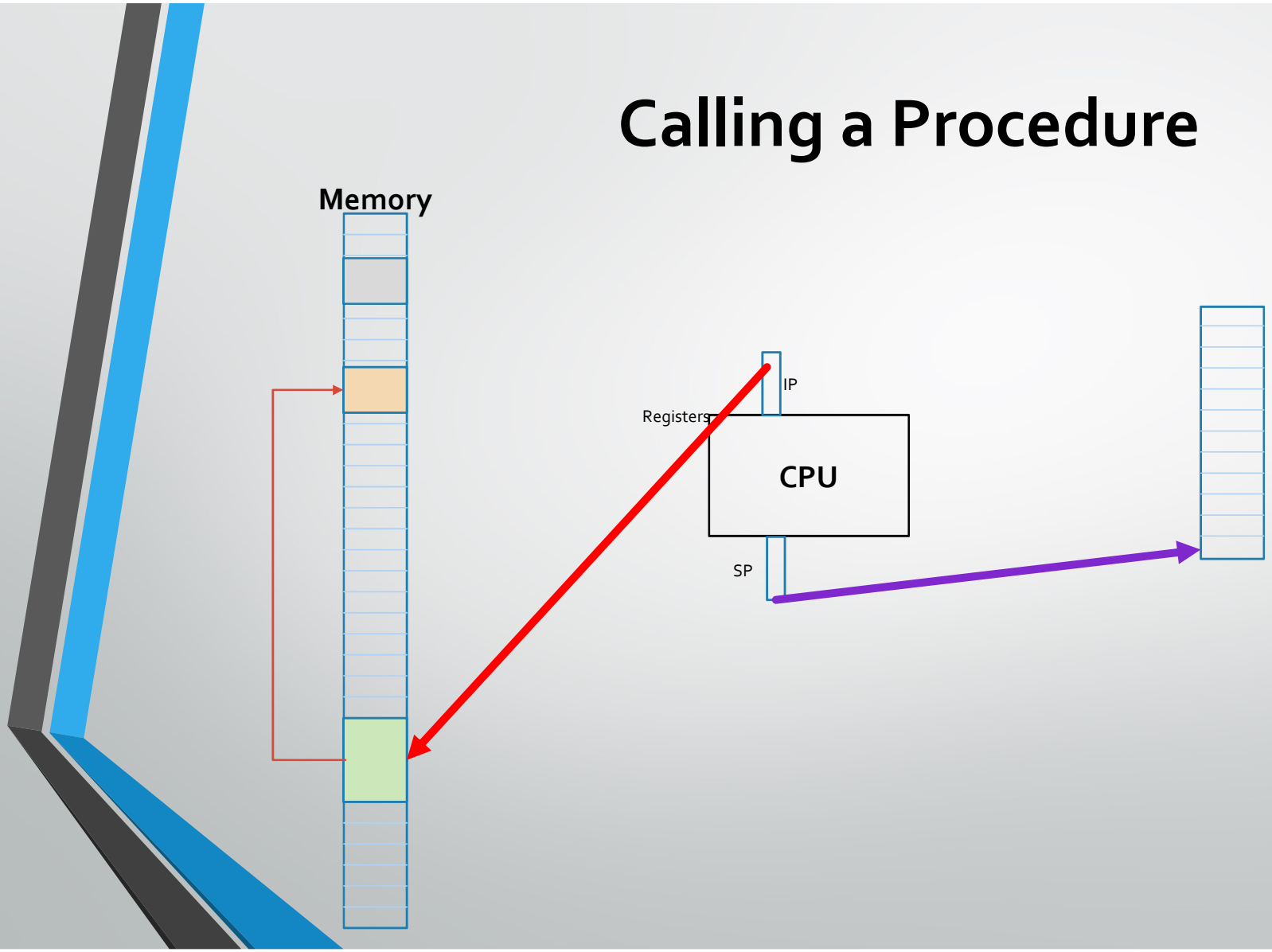
Calling a Procedure



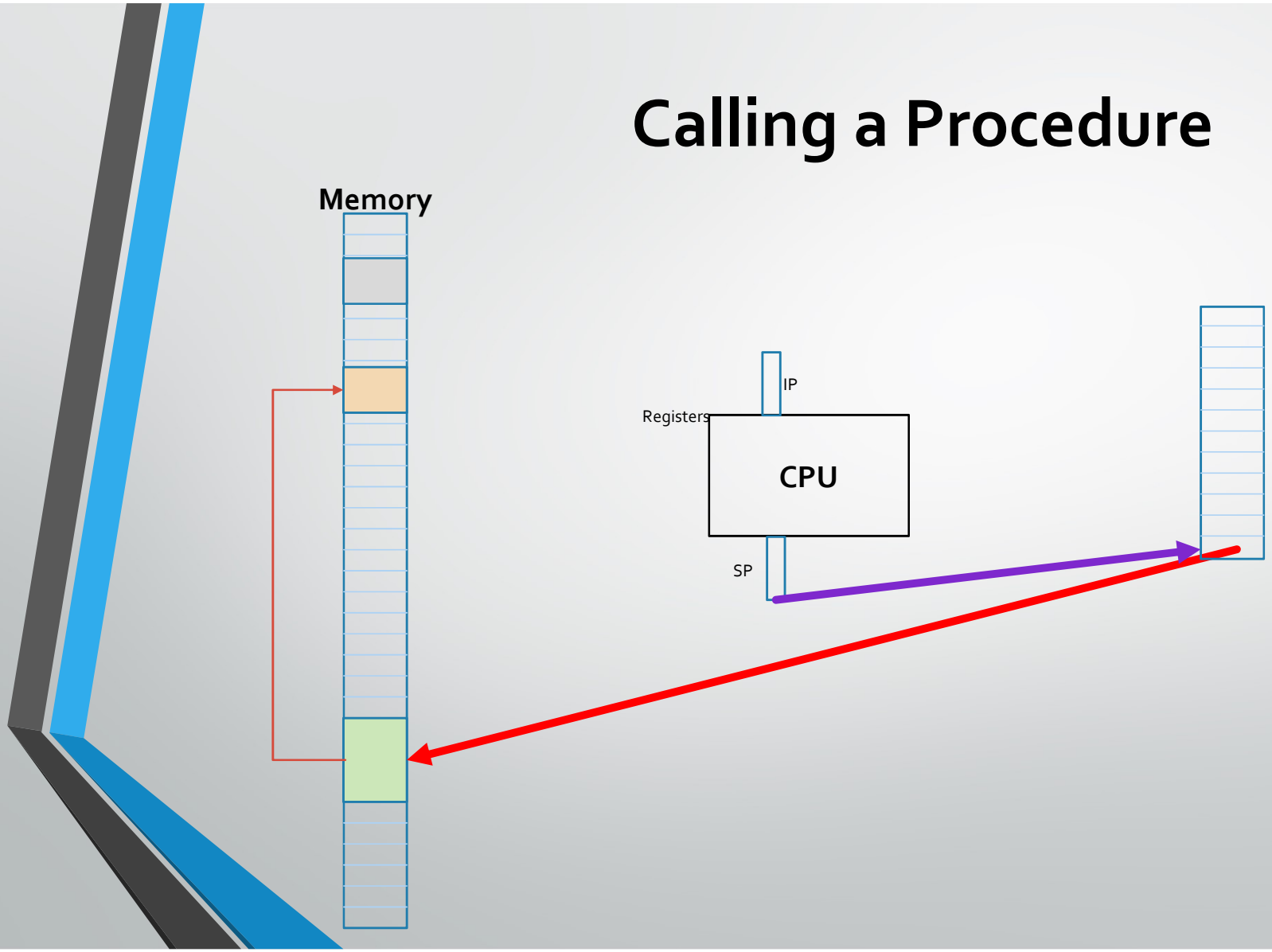
Procedures

- The Stack
- The Stack pointer

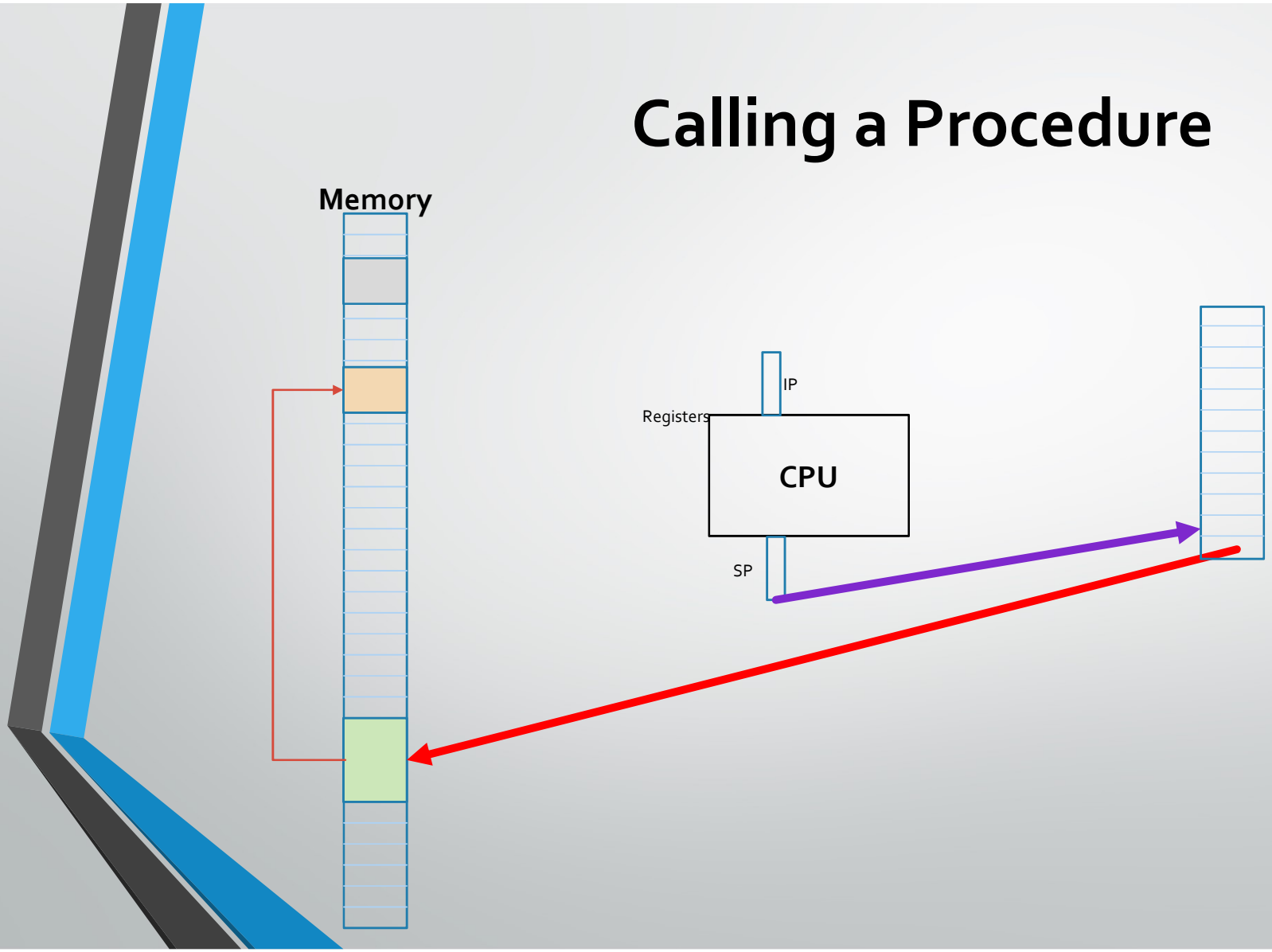
Calling a Procedure



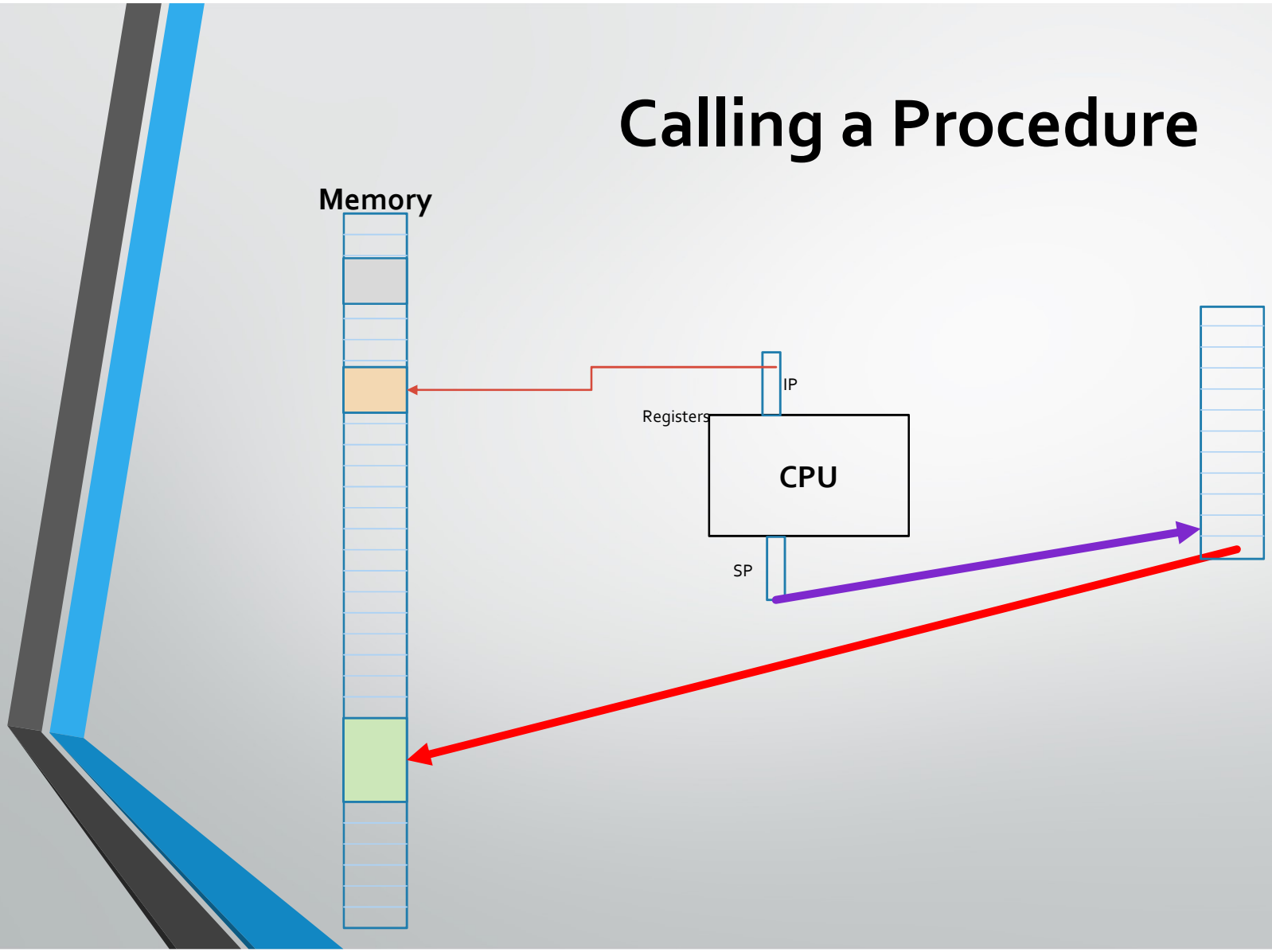
Calling a Procedure



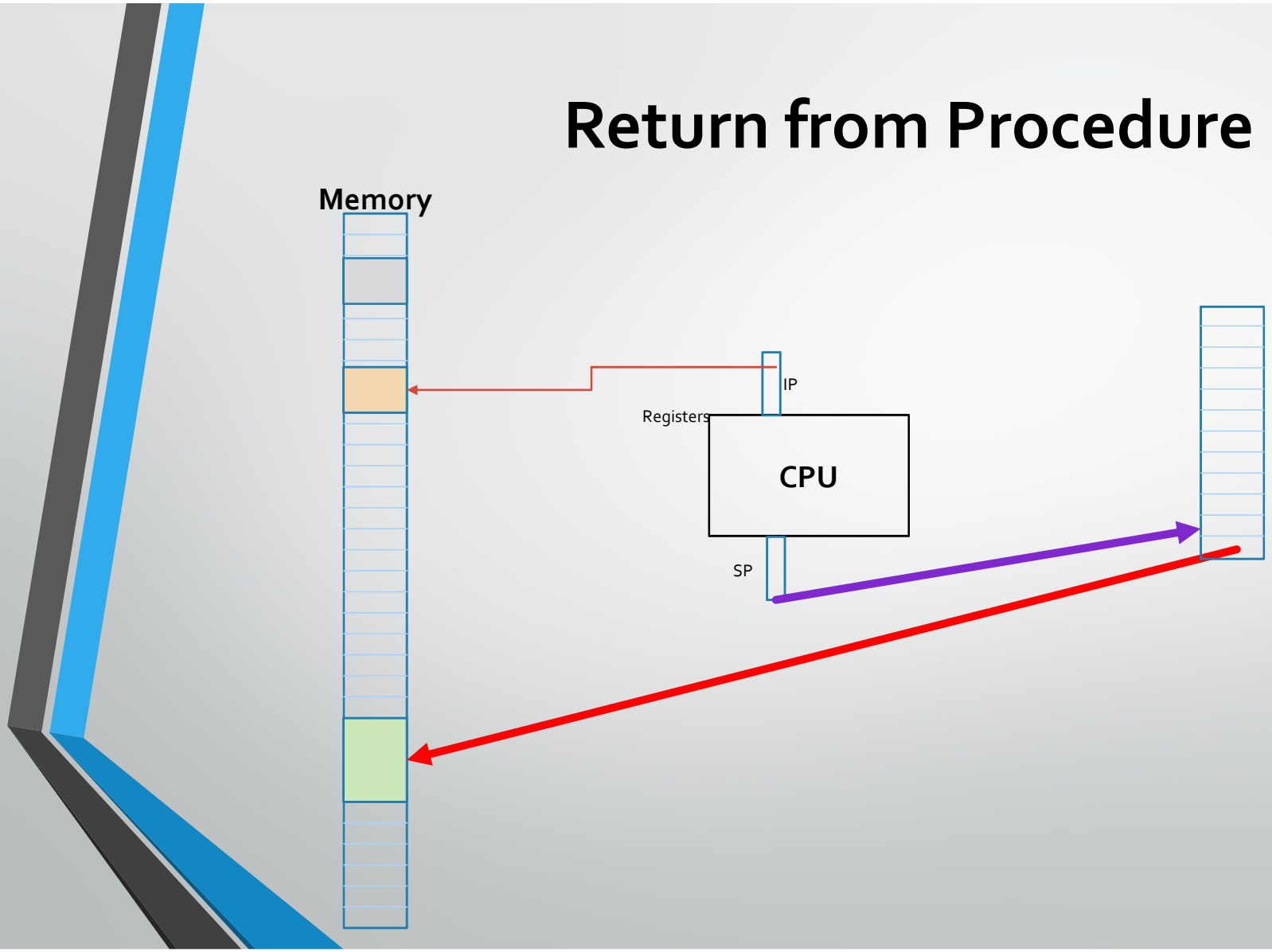
Calling a Procedure



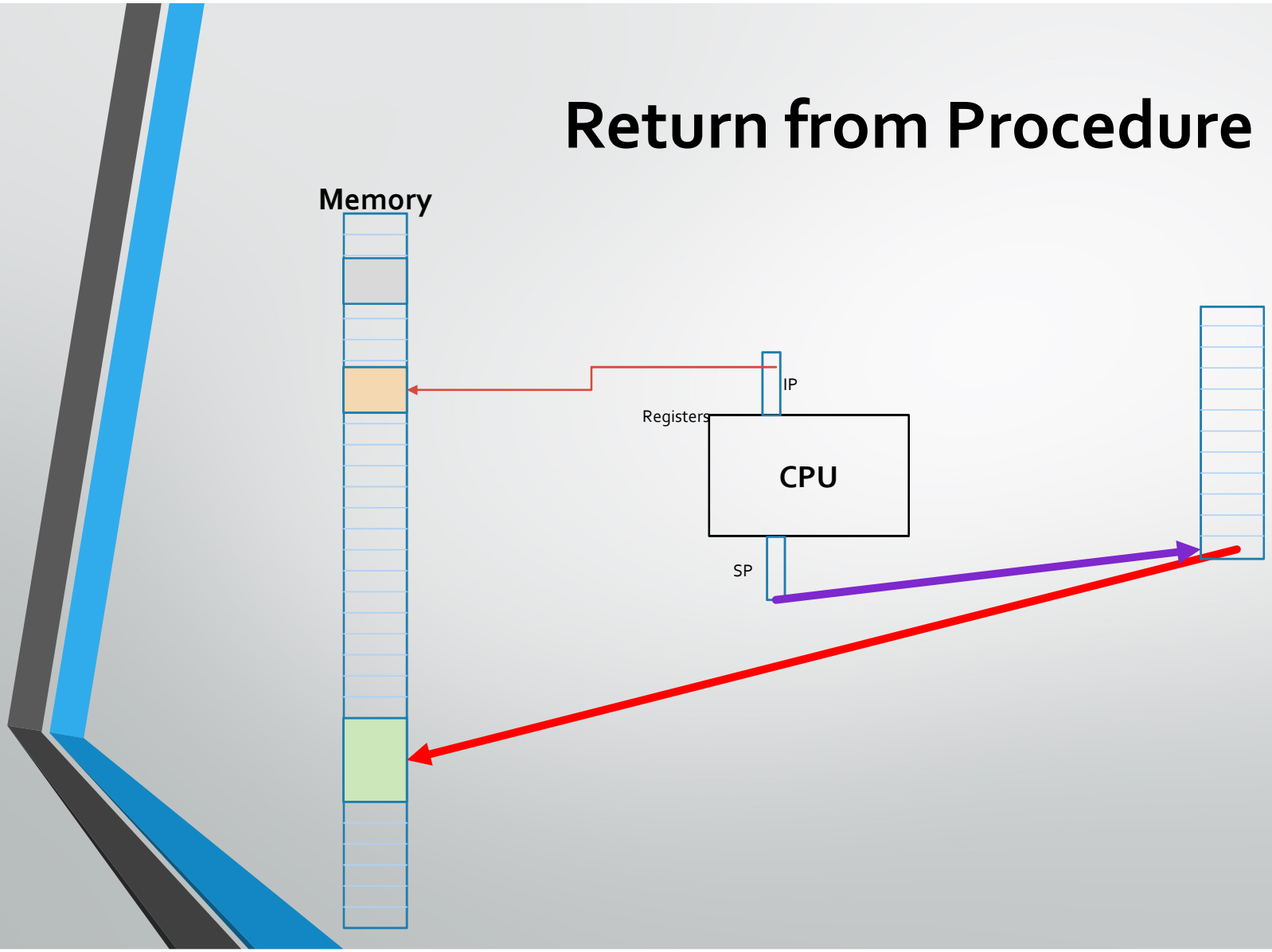
Calling a Procedure



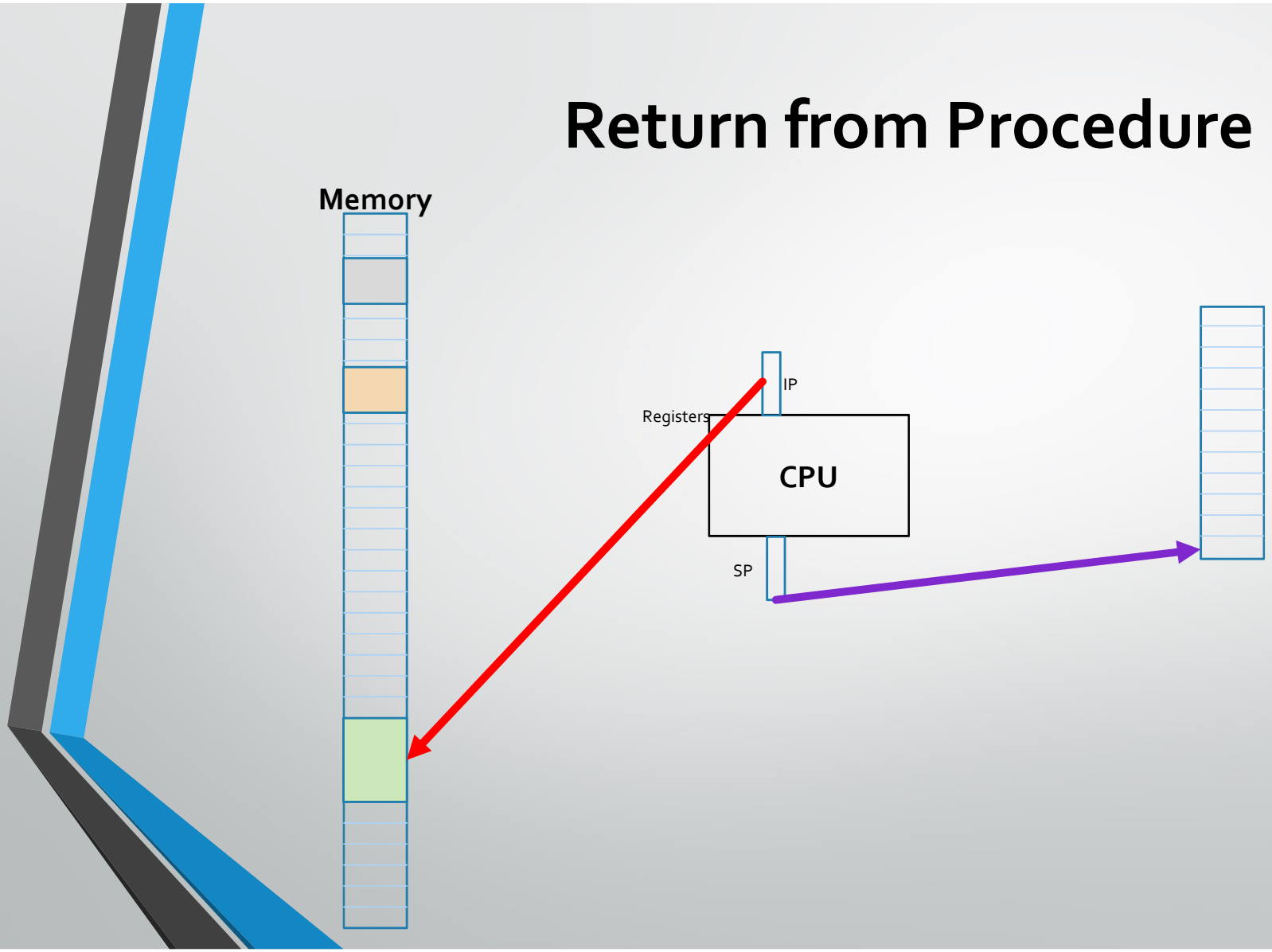
Return from Procedure



Return from Procedure



Return from Procedure



Logic / Arithmetic

- Usually executed by the CPU
- On data in registers
- Sometime on memory
- Depends on CPU

Assignment

- Move values from CPU registers into memory locations
- Sometime can move data from memory to memory
- Block transfers (BLOBs)
- Specific capabilities depend on CPU

ALGOL

```
BEGIN
FILE F (KIND=REMOTE);
EBCDIC ARRAY E [0:11];
REPLACE E BY "HELLO WORLD!";
WRITE (F, *, E);
END.
```

Fortran

```
program hello
print *, "Hello, World"
end program hello
```

BASIC

```
10 PRINT "Hello, World"
20 END
```

ADA

```
with Ada.Text_IO;

procedure HelloWorld is
   output_string : String(3..13);
begin
   output_string := "hello, world";
   Ada.Text_IO.Put (output_string);
end
```

C

```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, int argv)
{
   printf("hello, world\n“);
   return 0;
}
```

Pascal

```
Program Hello(output)
Begin
   writeln("hello, world\n“);
End;
```

Conditionals

- Conditionals
  - Simple (if)
    - Special comparison operators in the CPU
    - Work on registers or on memory
  - Complex (switch)
    - Works with jumps: IP assignments

Memory Allocation

- Memory management subsystem
- Allocate()
- Free()
- Memory leaks if something goes wrong
- Most difficult part of "regular programming"
  - "Solved" through **managed memory systems** (later)