Embedded Systems (9)

- Will start at 15:10
- PDF of this slide is available via ScombZ

Hiroki Sato <i048219@shibaura-it.ac.jp>

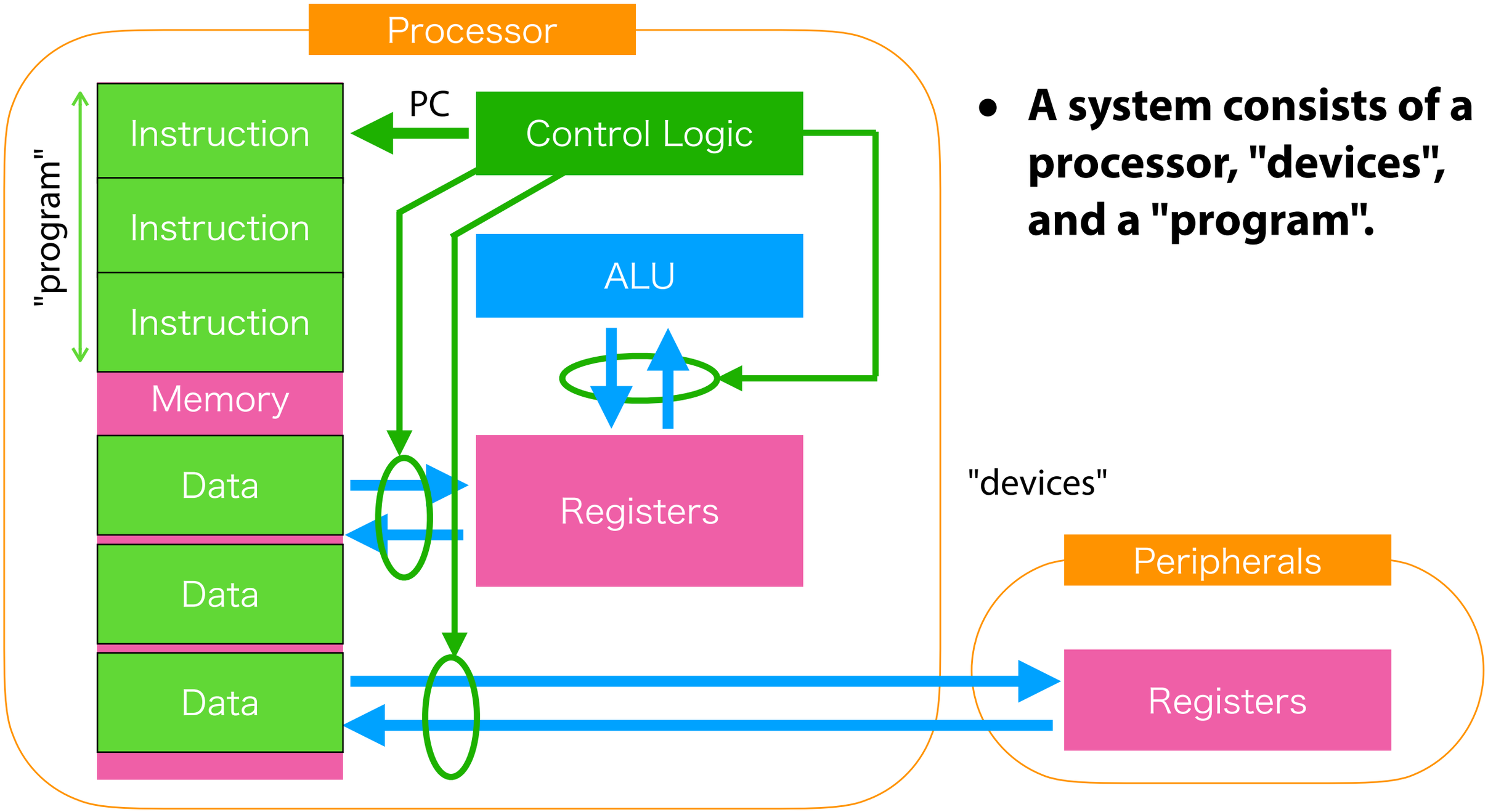
15:10-16:50 on Wednesday

Targets At a Glance

- **What you will learn today**
  - Development of an embedded system: what we learned so far
  - Complexity due to processor instructions and primitives
    - Software service: library and subroutine
  - Complexity due to multiple tasks
    - Concurrency and parallelism
  - A solution: operating system and middleware
    - Hardware abstraction and resource management

  - **Today's Project**
    - c) 4-wire communication (for evaluation)
      - **Submit your programs of (c) by December 1st.**

What we learned so far



- **A system consists of a processor, "devices", and a "program".**

Dealing with Multiple Tasks

- A practical system needs to deal with multiple tasks.
- A single processor can do one thing at a time (aka execution context)
- Execution context management is essential technique:
  - **Simple branch** by processor's branch instructions
  - **Subroutine call** by processor's call-ret instruction pair
  - **Interrupt** (event-driven branch)

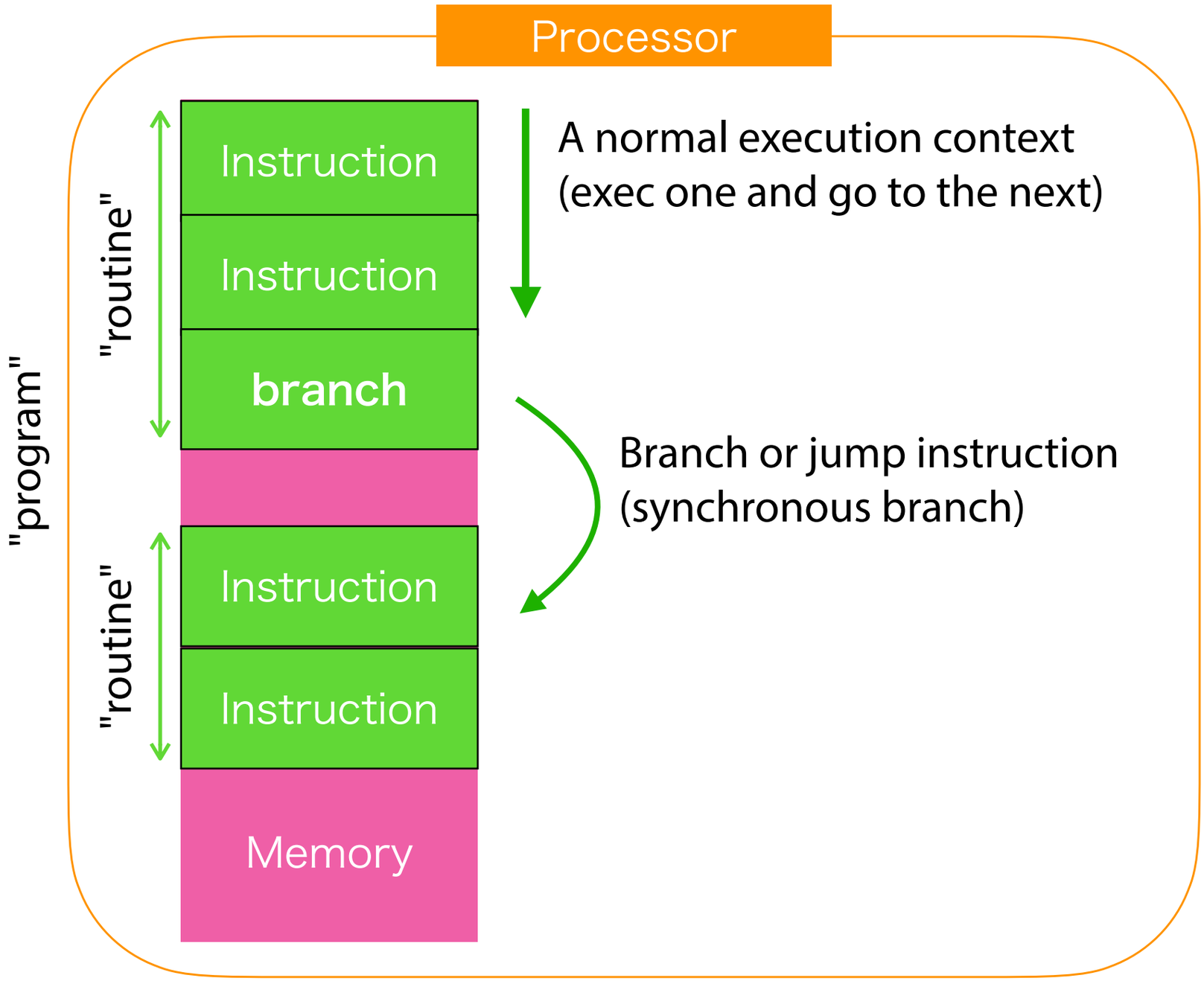
Branch

- Statements like **if**, **for**, and **while** use branch instructions.

```
void blink_led()
{
  if (digitalRead(A) == HIGH) {
    digitalWrite(B, HIGH);
  } else {
    digitalWrite(B, LOW);
  }
  delay(100);
}
```

- There are unconditional and conditional branches.
- No need to be aware of machine code-level instructions in most cases.

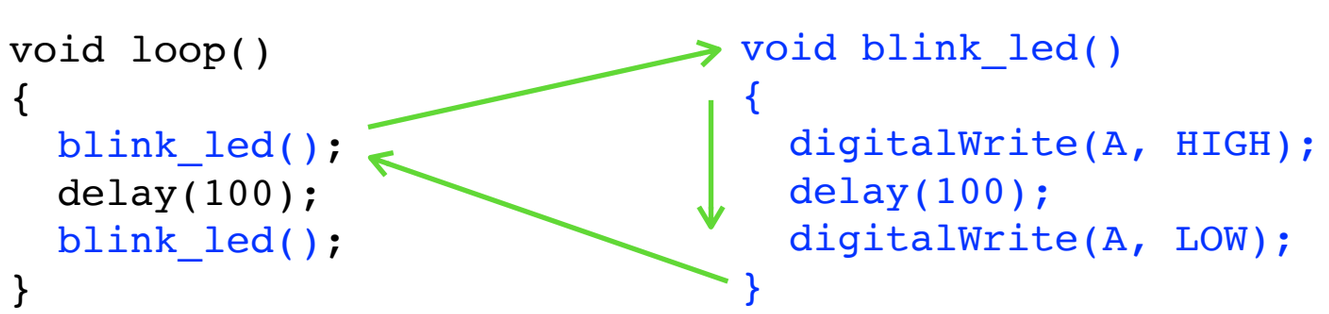
Branch



- **An execution context is controlled by PC (program counter).**
- **PC can be changed by "branch", "call", "ret" or an interrupt.**

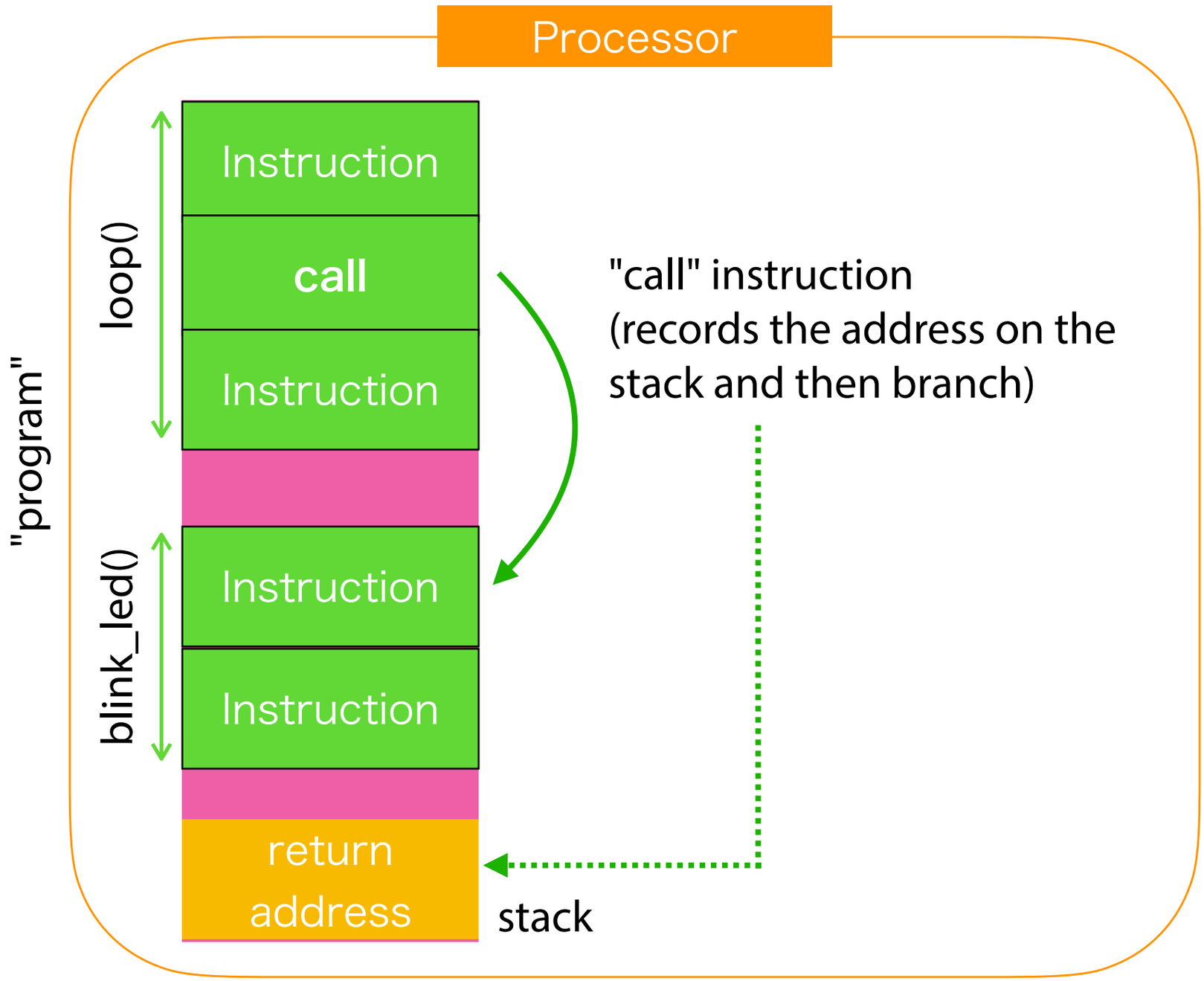
Subroutine

- A small program which is supposed to be called by another program. It does a specific task and also known as software service.
- Example: blink an LED



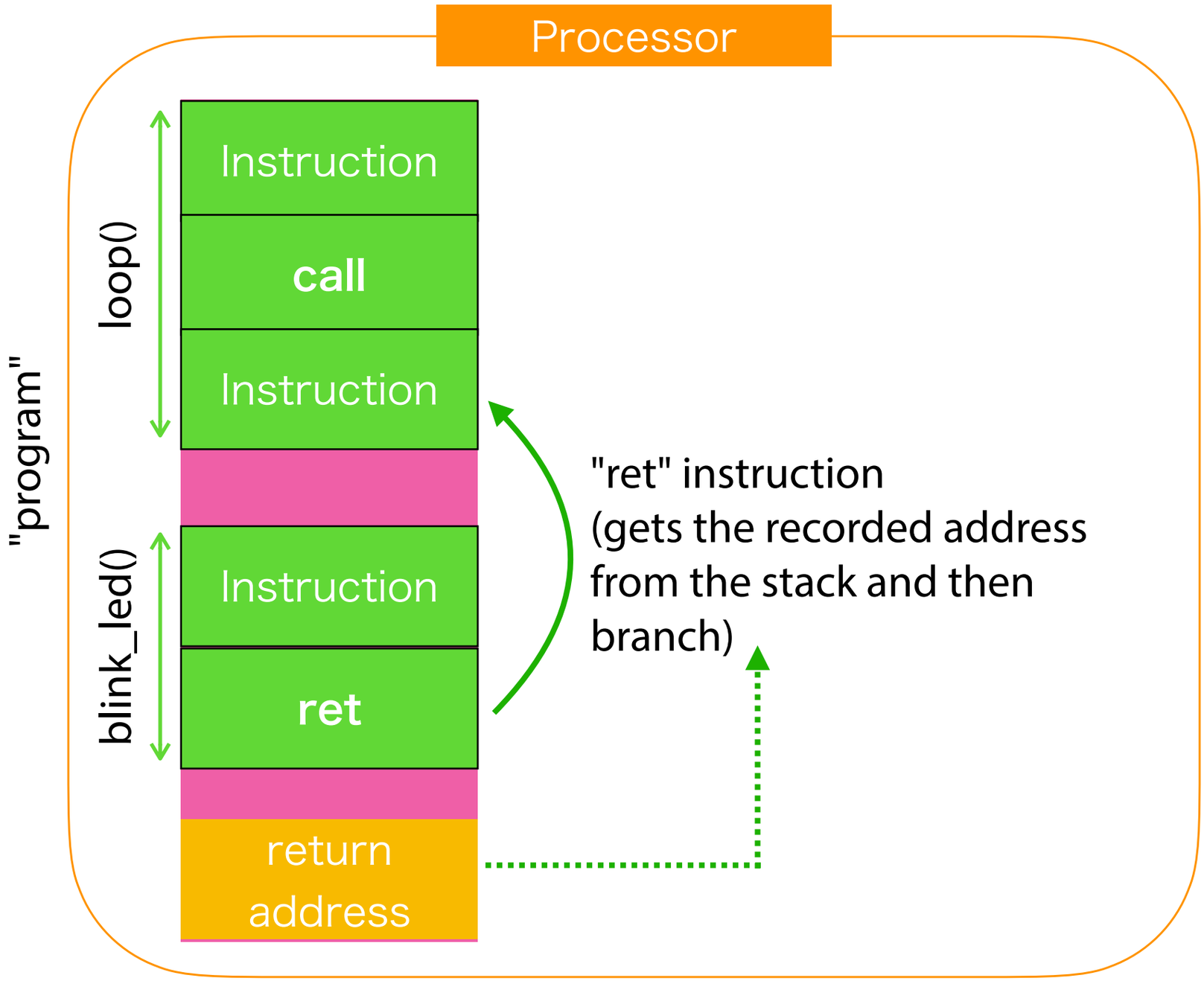
- In C language, it can be written as a function.
  - A call-ret pair is actually used in the machine-code level.
  - You can define a task as a function for re-use and forget the details.

Call



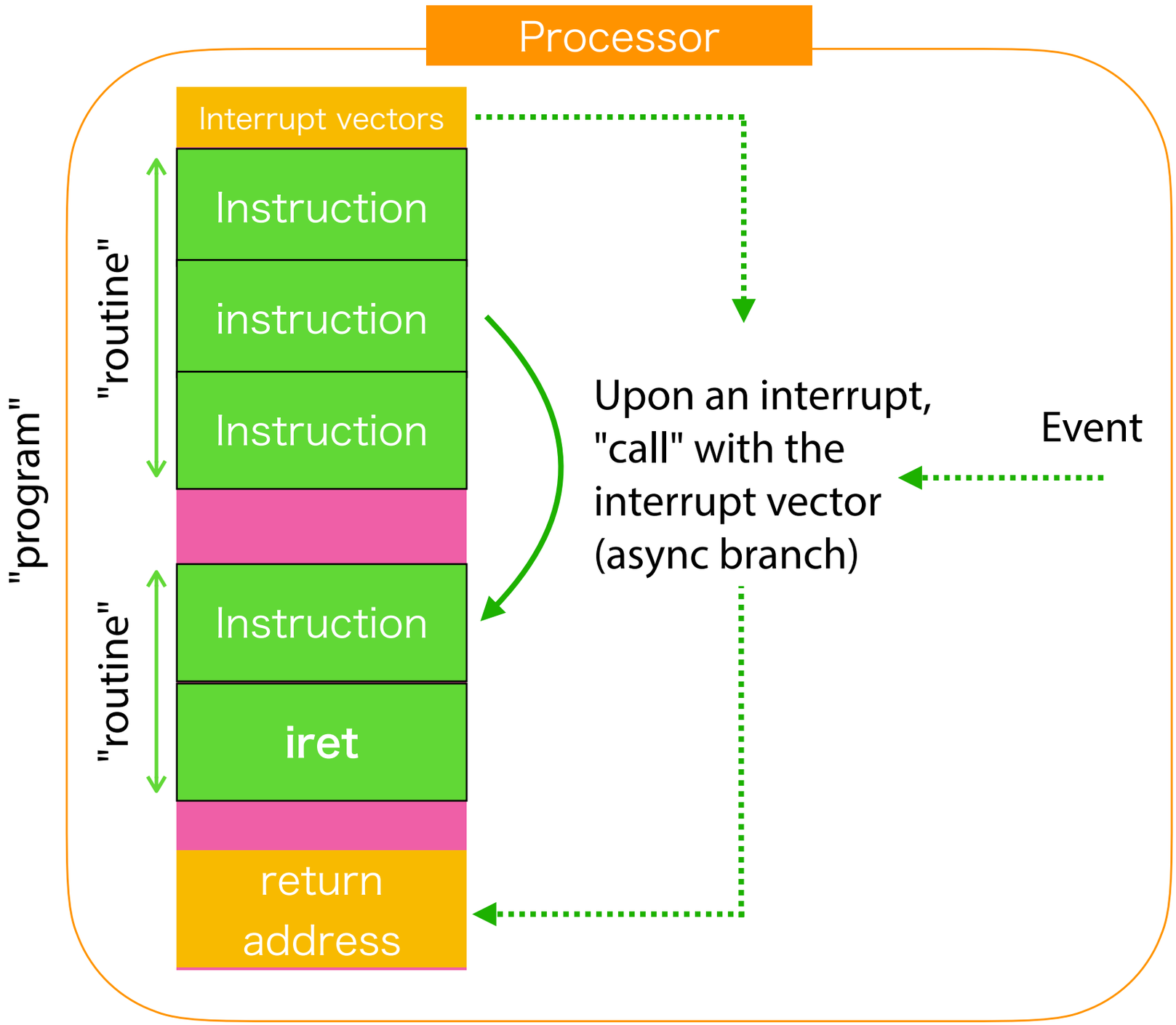
- **An execution context is controlled by PC (program counter).**
- **PC can be changed by "branch", "call", "ret" or an interrupt.**

Call and Ret



- "call" can be used recursively; the return addresses are recorded in a "stack" structure
- The depth is limited by the stack

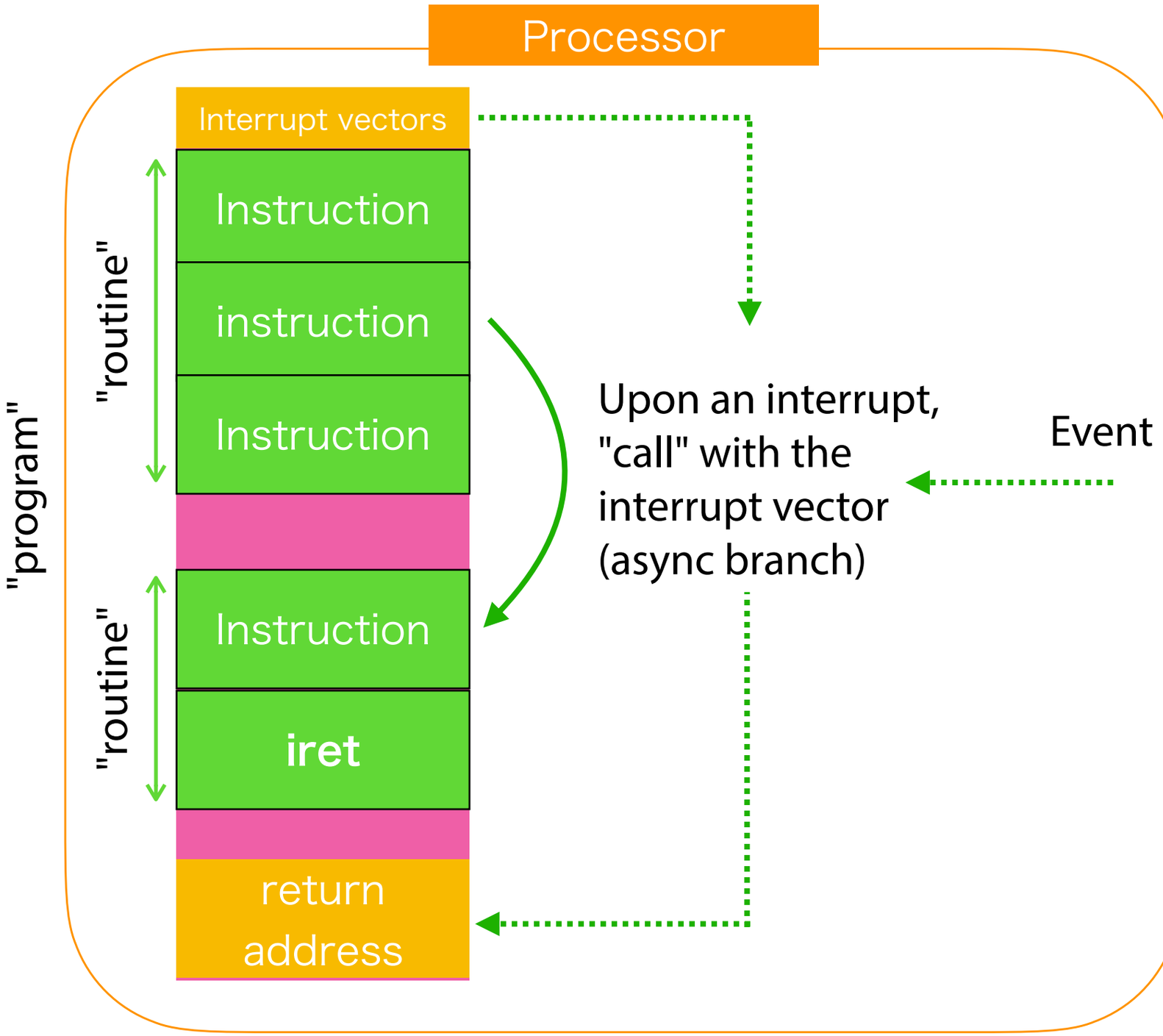
Interrupt



- You can disable interrupt using special instructions

```
void loop() {
  noInterrupts();
  //
  // no interrupt occurs
  //
  interrupts();
  //
  // interrupt occurs
  //
}
```

Interrupt



- Interrupt MUST NOT be recursive!

```
ISR (TIMER1_COMPA_vect) {

  // 1) this does not work because
  //    delay() uses interrupt
  delay(100);

  // 2) A timer interrupt handler
  //    must finish before another
  //    trap occurs.
  //
  //    Or you can disable the
  //    interrupt inside the handler.
  //    This breaks the fixed
  //    interval of the interrupts.
}
```

Complexity of Development

- Instructions are too primitive and hardware-dependent.
  - OCR1A means a register of the timer device. What is the number (62500 - 1)?

```
void setup()
{
  digitalWrite(CLK_OUT, HIGH);
  digitalWrite(DATA_OUT, led & 1);

  TCCR1A  = 0;
  TCCR1B = (1 << WGM12) | (1 << CS12);
  OCR1A   = 62500 - 1;
  TIMSK1 |= (1 << OCIE1A);
}
```

- Solution: using libraries and subroutines

Complexity of Development

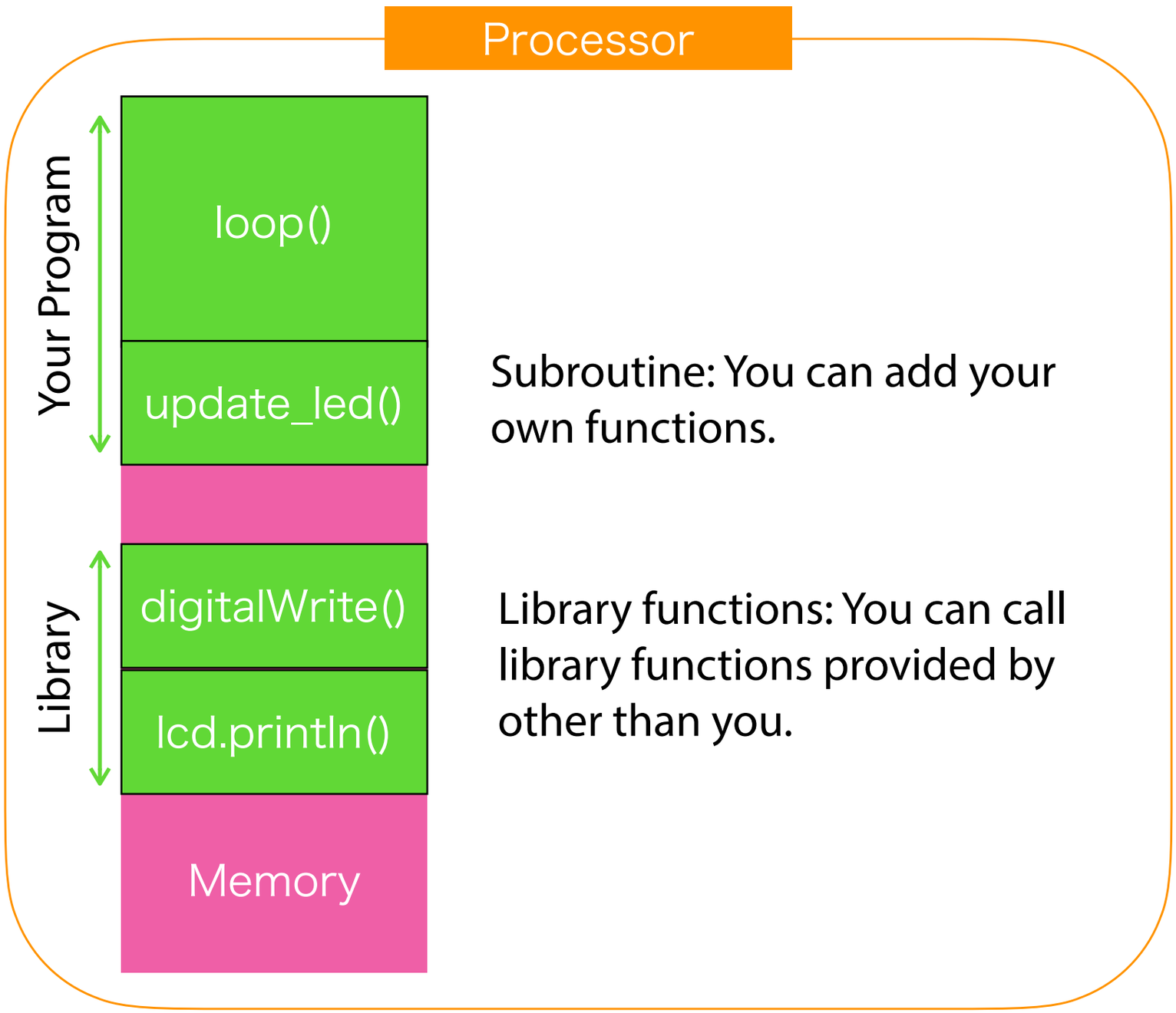
- Instructions are too primitive and hardware-dependent.
  - OCR1A means a register of the timer device. What is the number (62500 - 1)?

```
void setup()
{
  digitalWrite(CLK_OUT, HIGH);
  digitalWrite(DATA_OUT, led & 1);
  setup_timer(1000); // about 1000ms
}

void setup_timer(int time)
{
  TCCR1A  = 0;
  TCCR1B = (1 << WGM12) | (1 << CS12);
  OCR1A   = time * 62 - 1;
  TIMSK1 |= (1 << OCIE1A);
}
```

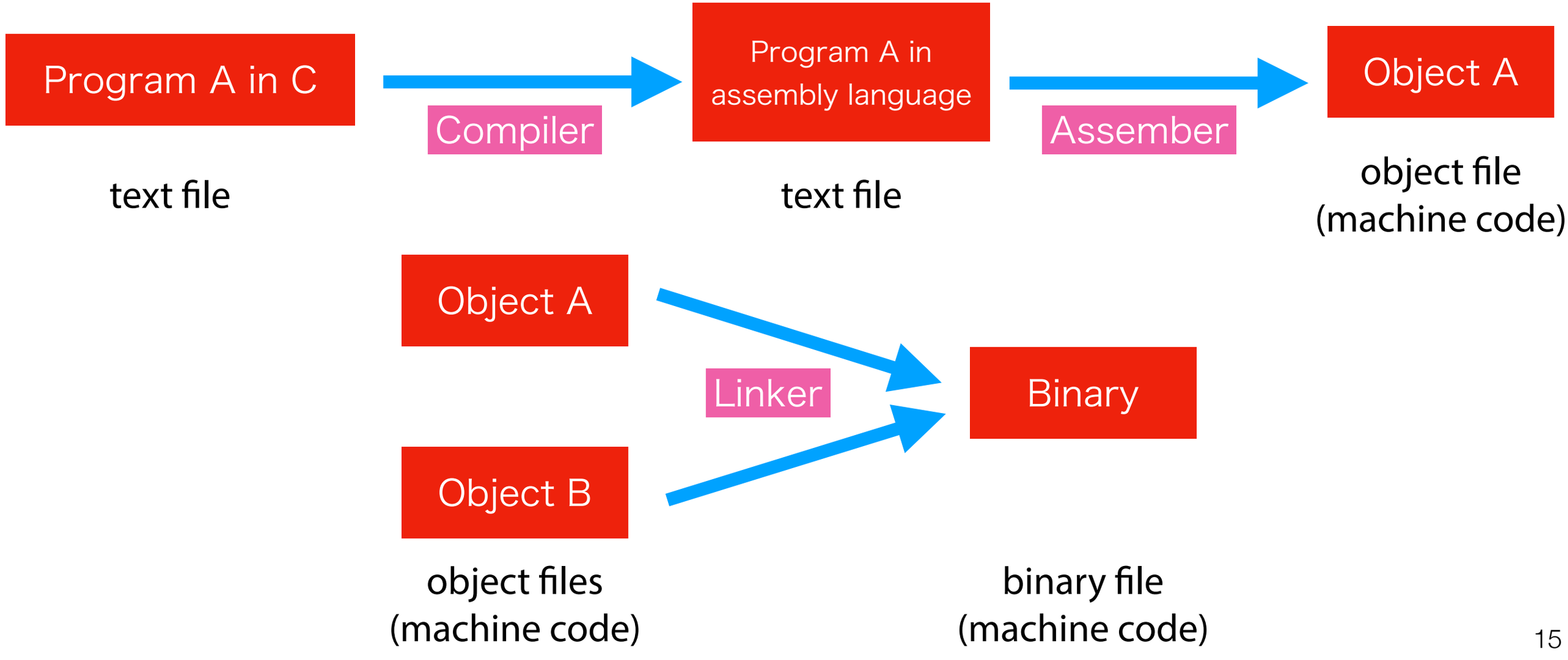
- Solution: using libraries and subroutines

Subroutine and Library



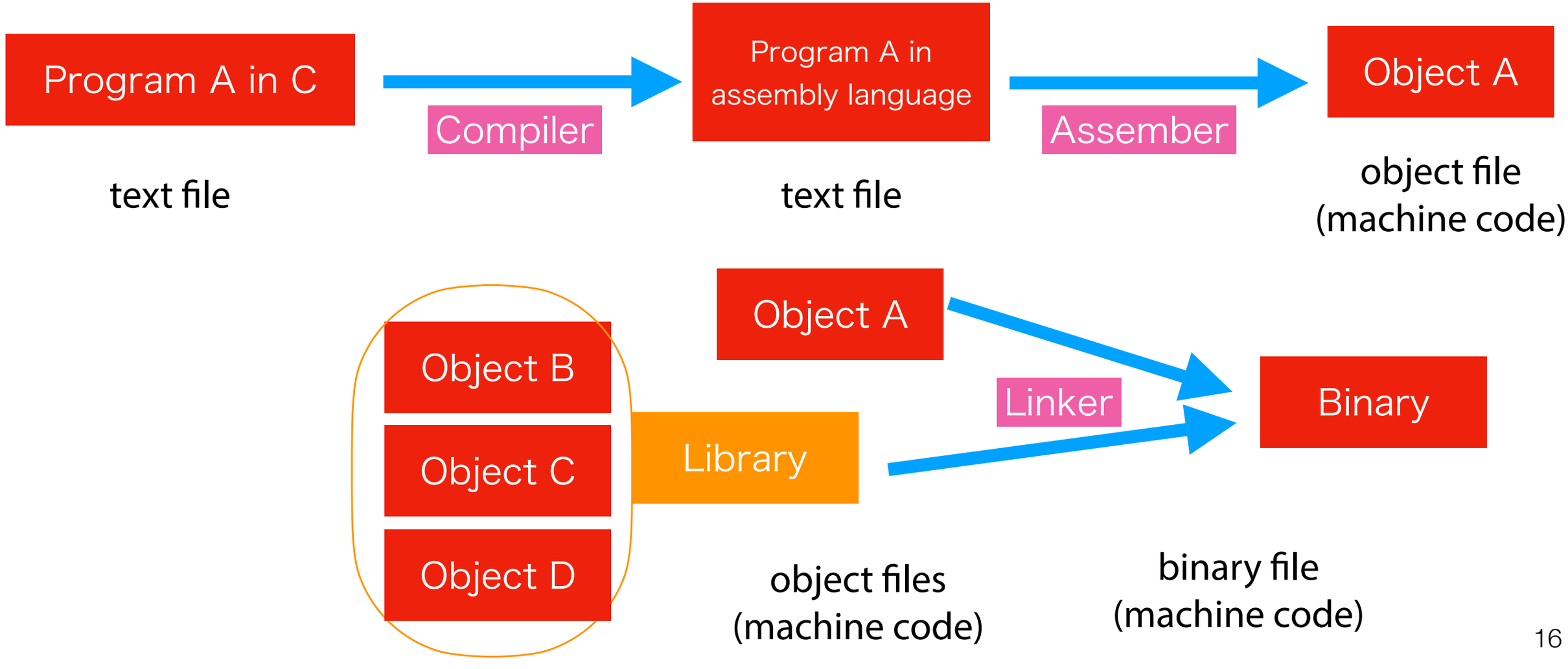
Library

- A collection of pre-compiled objects which contains subroutines.



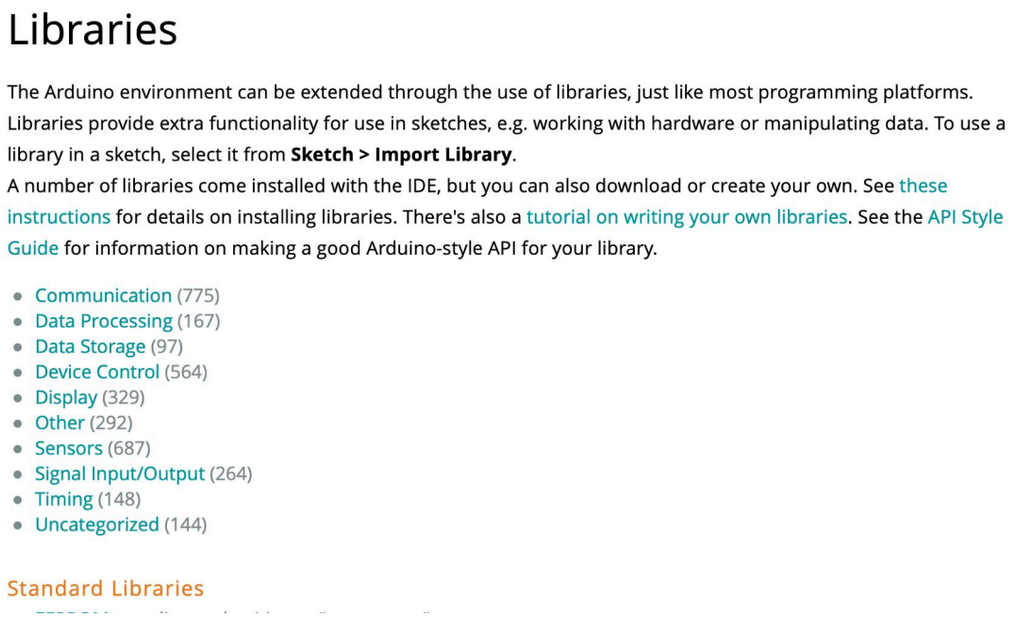
Library

- A collection of pre-compiled objects which contains subroutines.



Subroutine and Library

- Both are useful for re-using programs.
- Libraries are typically available on the development platform (toolchains).
- Arduino supports a lot of libraries.



Libraries

The Arduino environment can be extended through the use of libraries, just like most programming platforms. Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from **Sketch > Import Library**.
A number of libraries come installed with the IDE, but you can also download or create your own. See these instructions for details on installing libraries. There's also a tutorial on writing your own libraries. See the API Style Guide for information on making a good Arduino-style API for your library.

- Communication (775)
- Data Processing (167)
- Data Storage (97)
- Device Control (564)
- Display (329)
- Other (292)
- Sensors (687)
- Signal Input/Output (264)
- Timing (148)
- Uncategorized (144)

Standard Libraries

See: https://www.arduino.cc/reference/en/

Subroutine and Library

- API (Application Programming Interface)
  - Defines how to use the subroutines: parameters, data structure, calling conventions.

SoftwareSerial Library

The Arduino hardware has built-in support for serial communication on pins 0 and 1 (which also goes to the computer via the USB connection). The native serial support happens via a piece of hardware (built into the chip) called a UART. This hardware allows the Atmega chip to receive serial communication even while working on other tasks, as long as there room in the 64 byte serial buffer.

The SoftwareSerial library has been developed to allow serial communication on other digital pins of the Arduino, using software to replicate the functionality (hence the name "SoftwareSerial"). It is possible to have multiple software serial ports with speeds up to 115200 bps. A parameter enables inverted signaling for devices which require that protocol.

The version of SoftwareSerial included in 1.0 and later is based on the NewSoftSerial library by Mikal Hart.

To use this library

```
#include <SoftwareSerial.h>
```

Functions

- SoftwareSerial()
- available()
- begin()
- isListening()
- overflow()
- peek()
- read()
- print()
- println()
- listen()
- write()

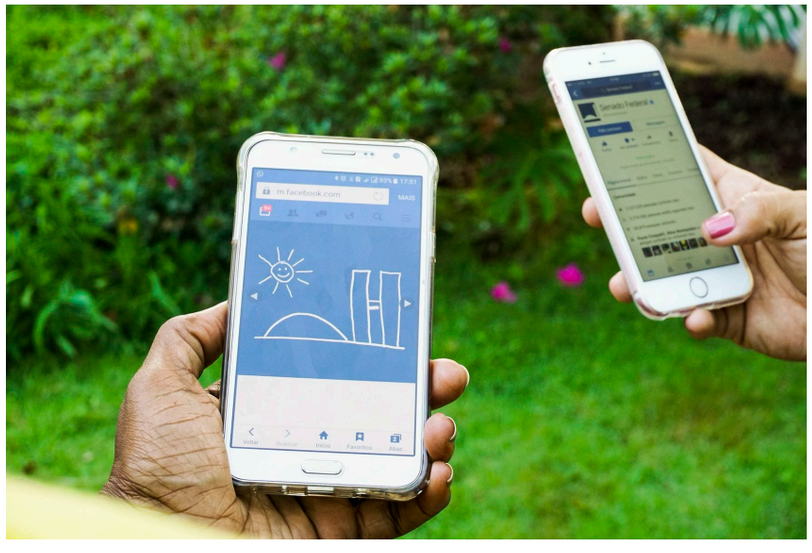
```
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  int i = 0;

  Serial.println(i++);
  delay(1000);
}
```

Complexity of Development

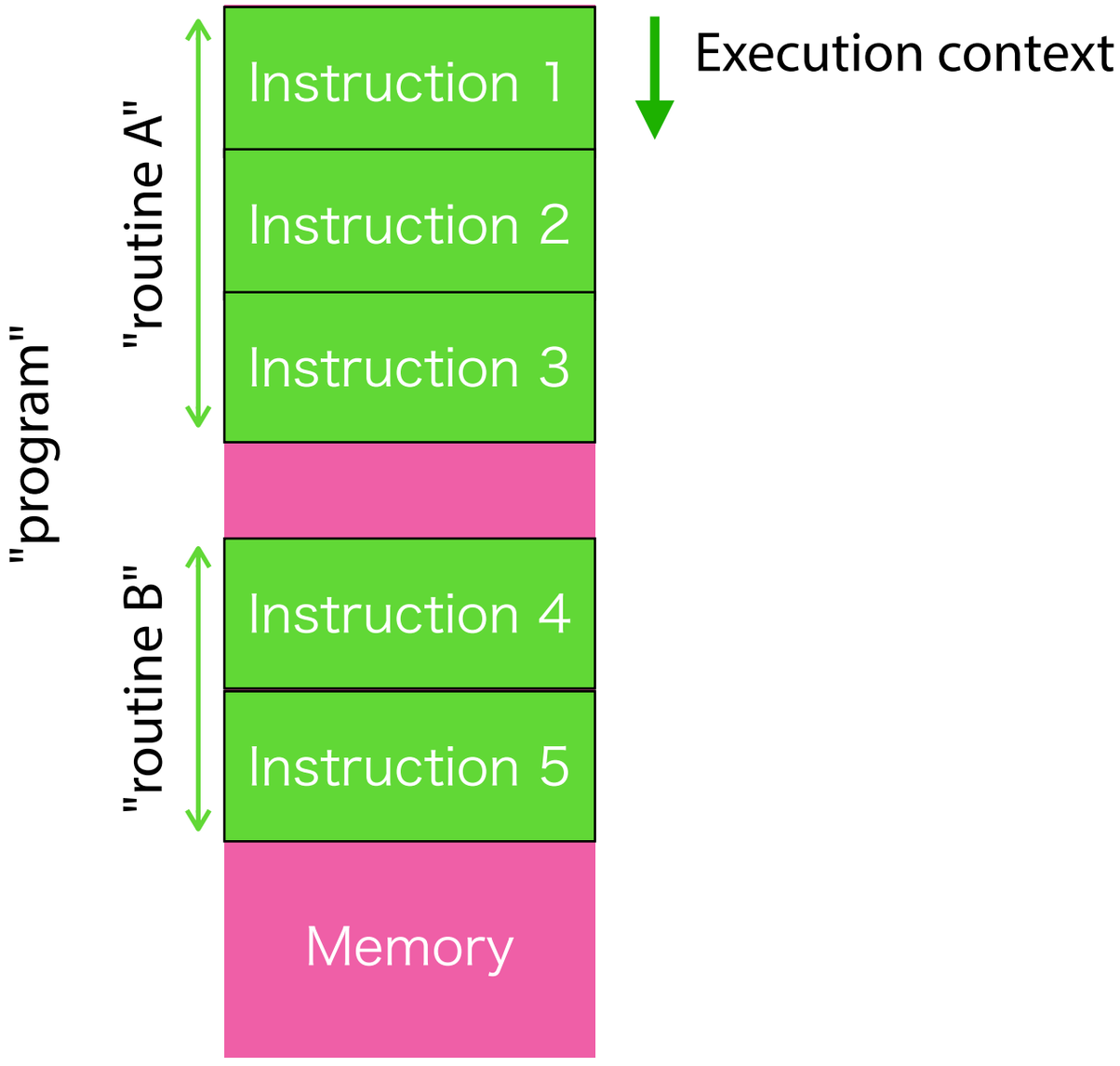
- A simple task can be handled easily. Say, a remote controller reads the status of buttons and sends a signal when a button is pressed.



- How to deal with multiple tasks? Reading buttons, doing communication, etc. at a time?

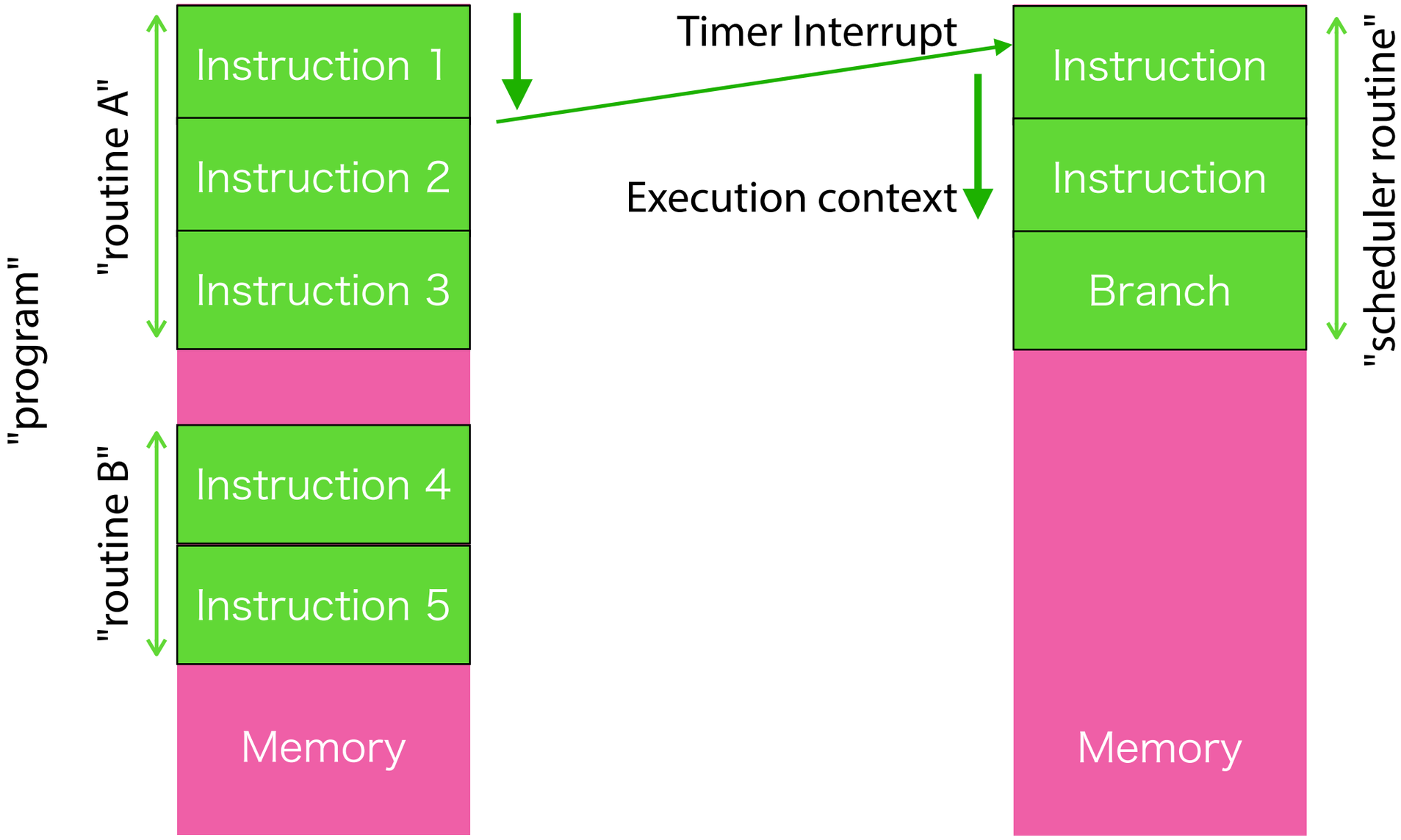
Concurrency

- A processor can have a single execution context.



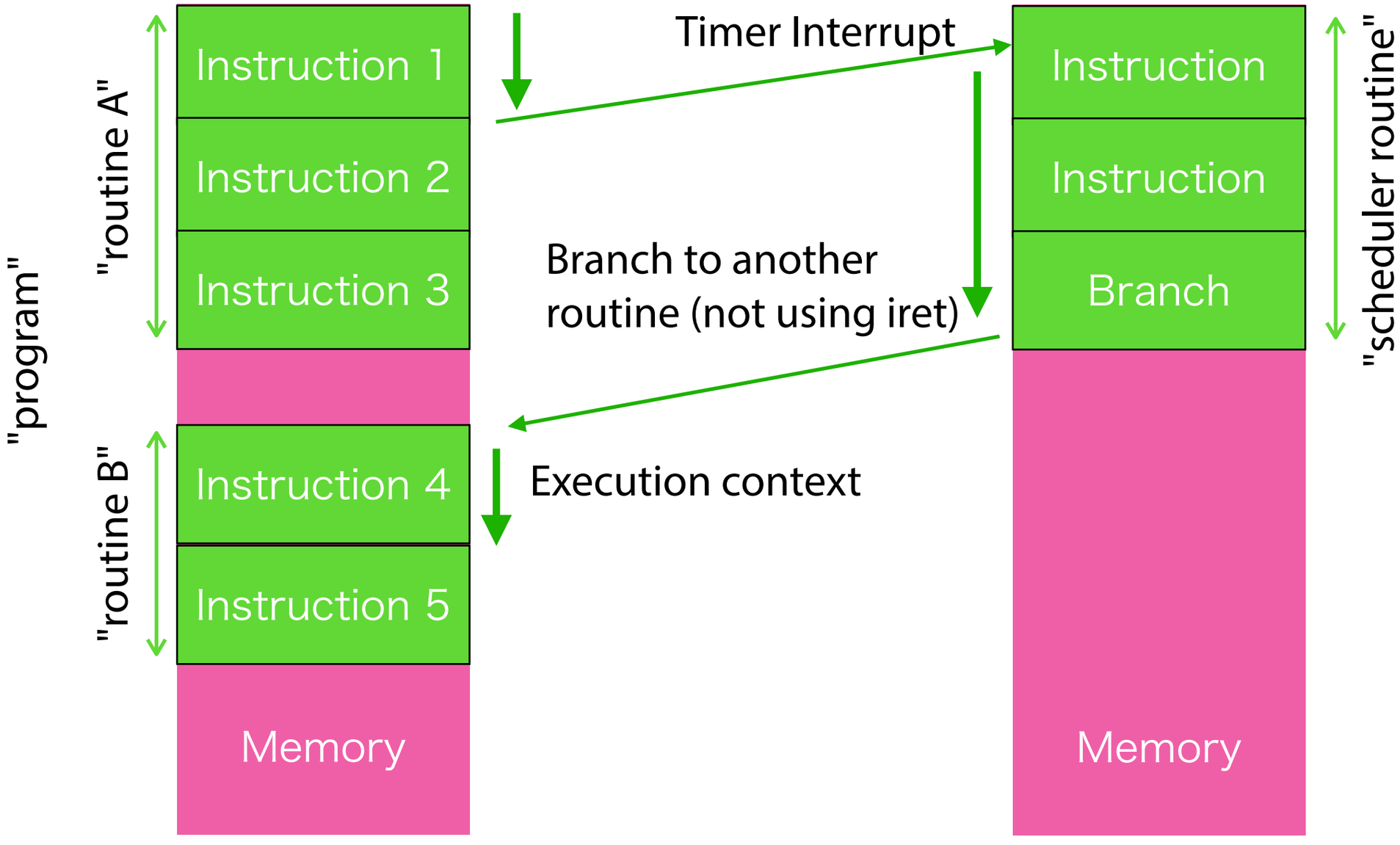
Concurrency

- A processor can have a single execution context.



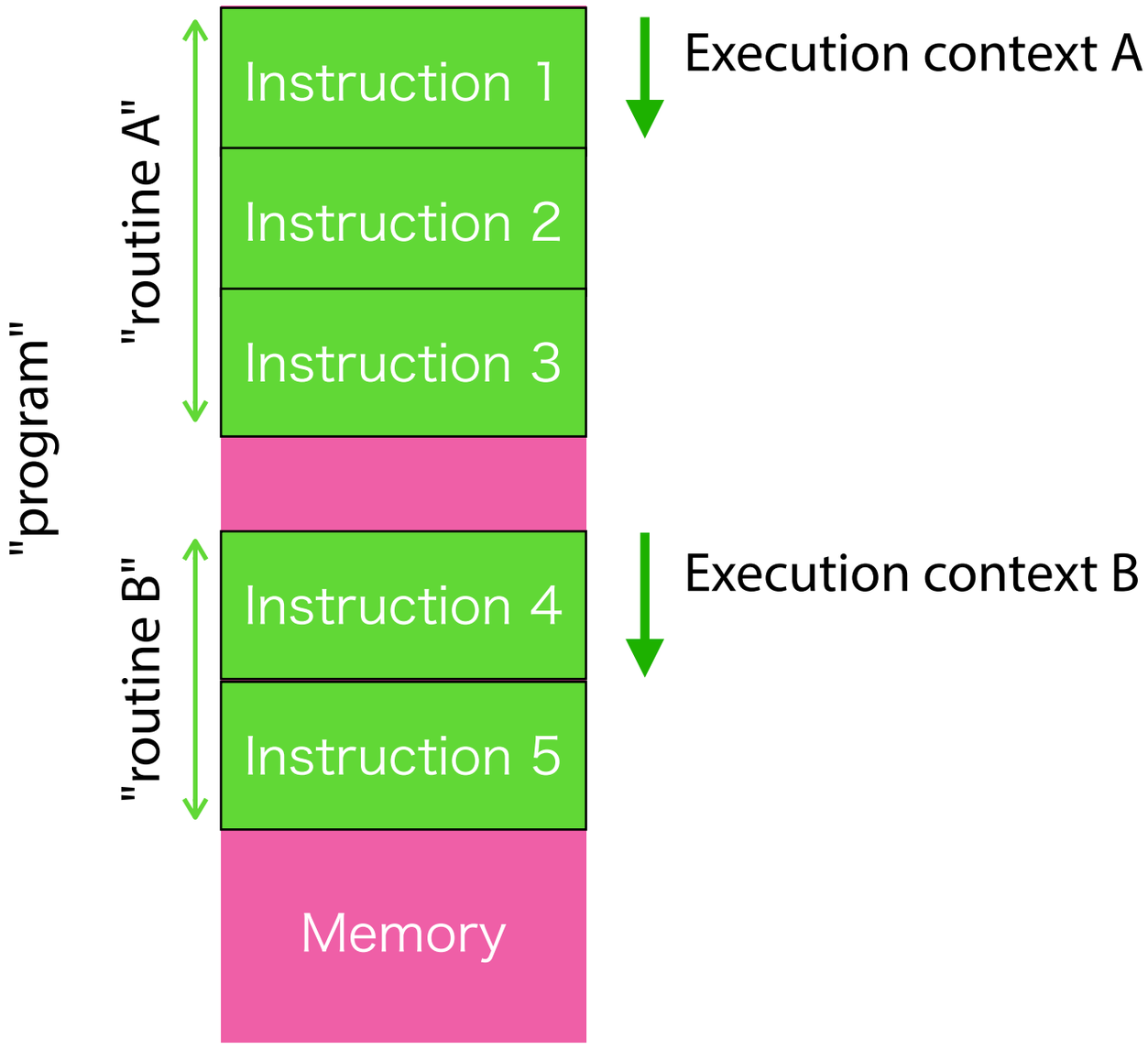
Concurrency

- A processor can have a single execution context.



Concurrency

- Two routines can run in a time-division basis.



Note that A or B may or may not finish

Concurrency

- **Concurrency is realized by scheduling the multiple execution contexts.**
  - The scheduler is a routine which controls the program counter. The timer interrupt is often used together.
  - The execution contexts handled by the scheduler are called a "thread" or a "process".
  - The scheduler saves contents of the registers and switches the context to another thread.
- You can program blinking LEDs and serial communication as two independent programs, and run them by the scheduler.
- Who prepares the scheduler? The answer is "system software".

Parallelism

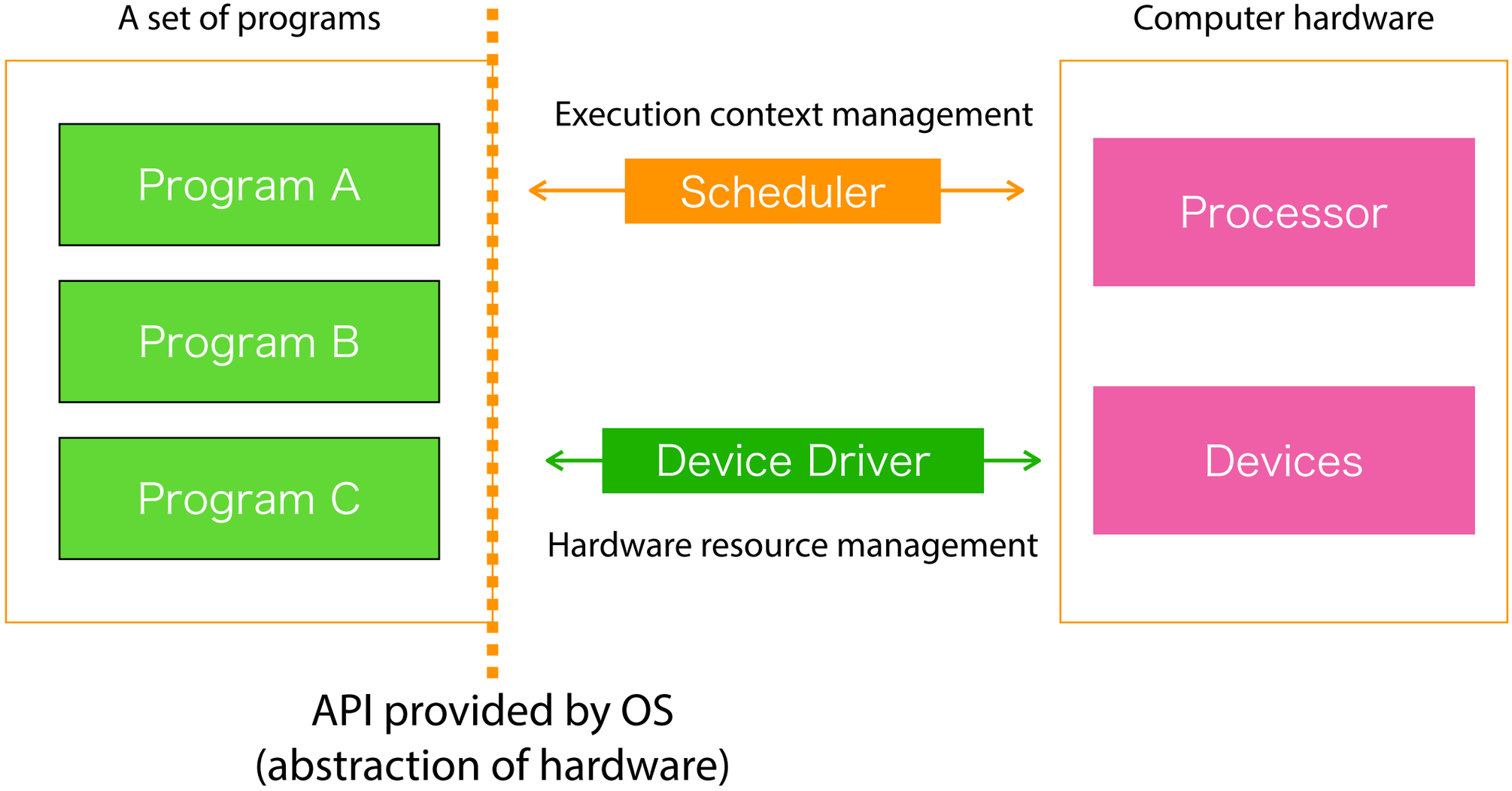
- Parallel execution is also possible when multiple processors are available on the system.
- In that case, there are two or more execution context at a time. It is a different concept from concurrency.
- Most of modern general-purpose computers are multi-processor (multi-core) systems.
  - **From the perspective of software, there are multiple program counters.**
- Synchronization of memory access is one of the problems because peripheral devices are shared.

Operating System

- A solution to mitigate the complexity issues
  - It is a software component.
  - It provides a set of programs for re-use.
  - It provides scheduler to realize concurrency.
- It provides "abstraction of resources"

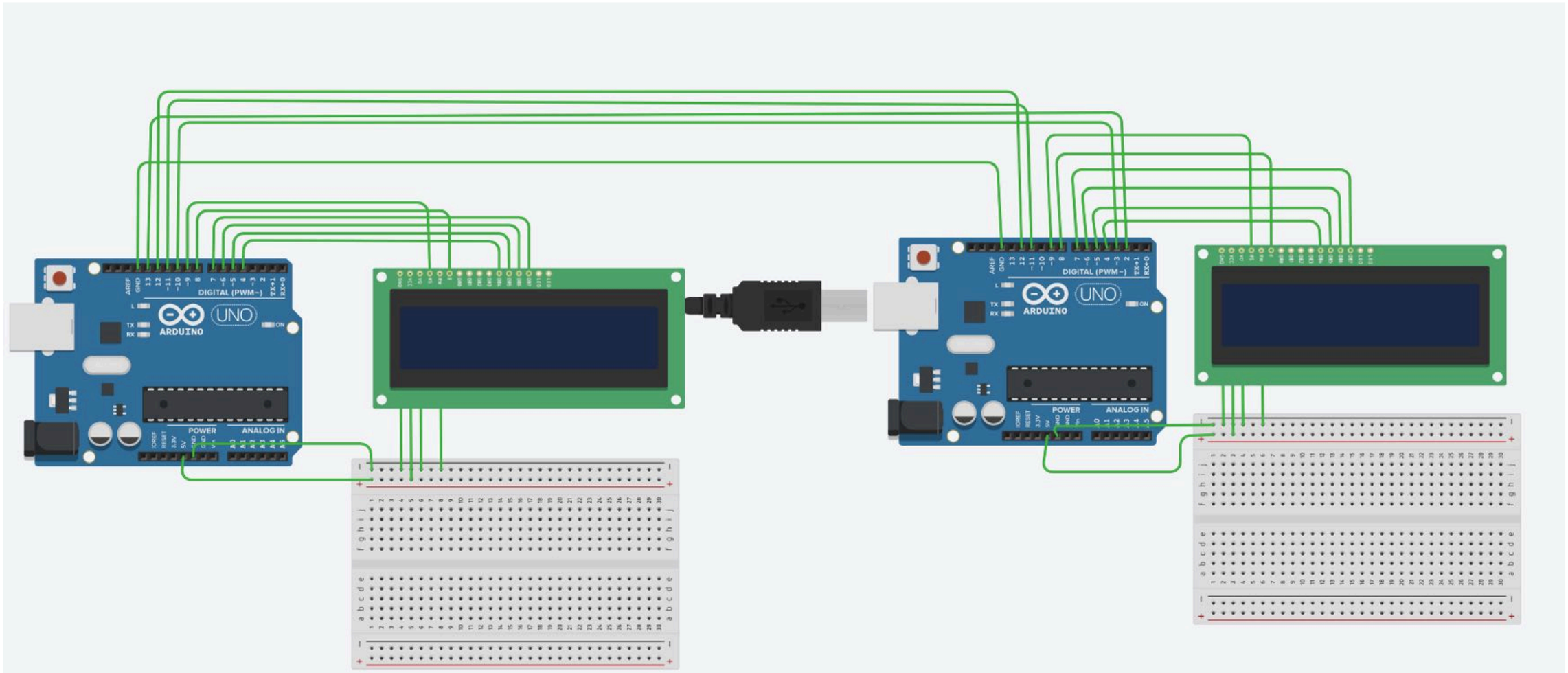
Operating System

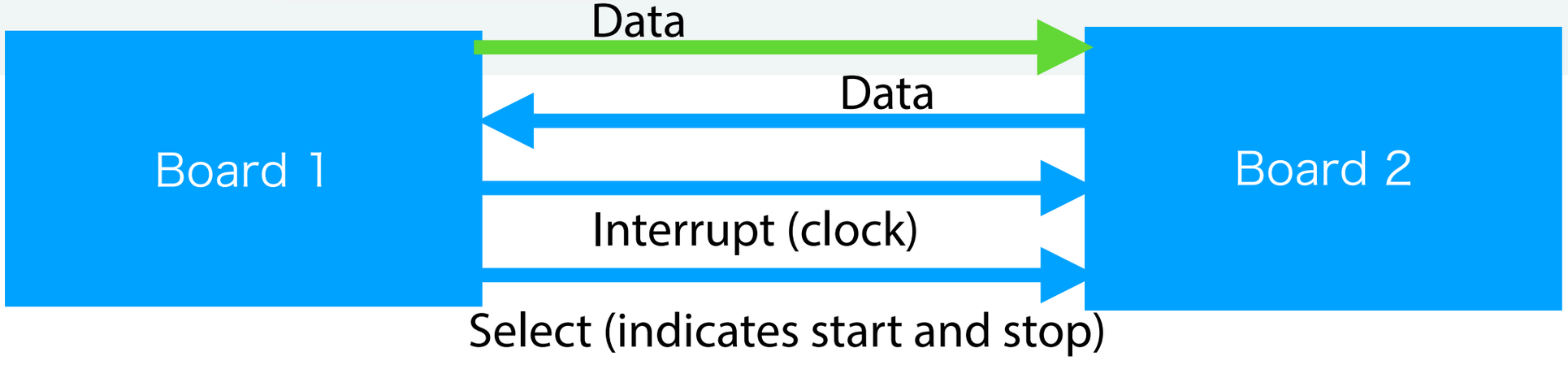
OS works as an intermediate layer between programs and computer hardware



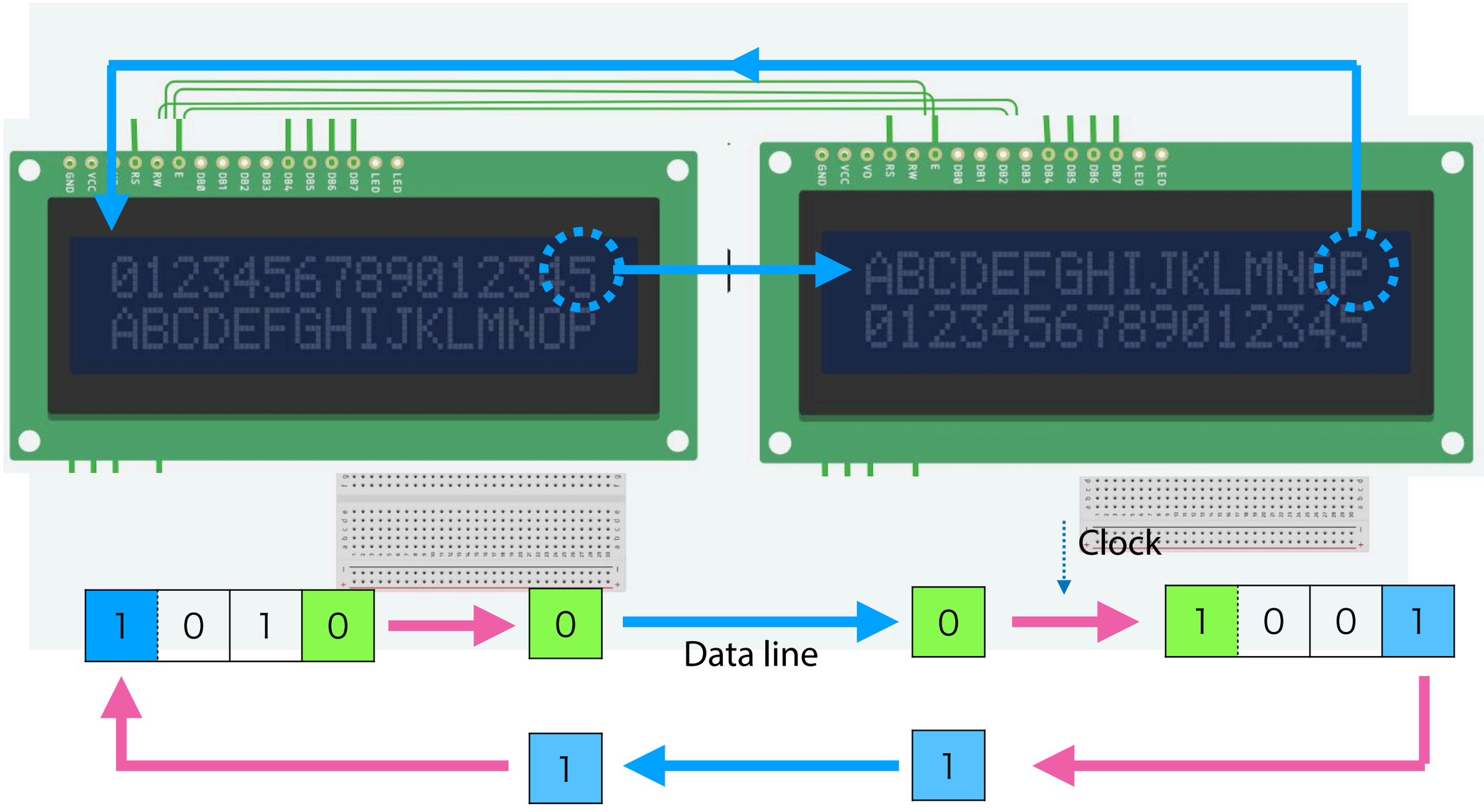
Time for Your Project

c) 4-Wire Serial Communication

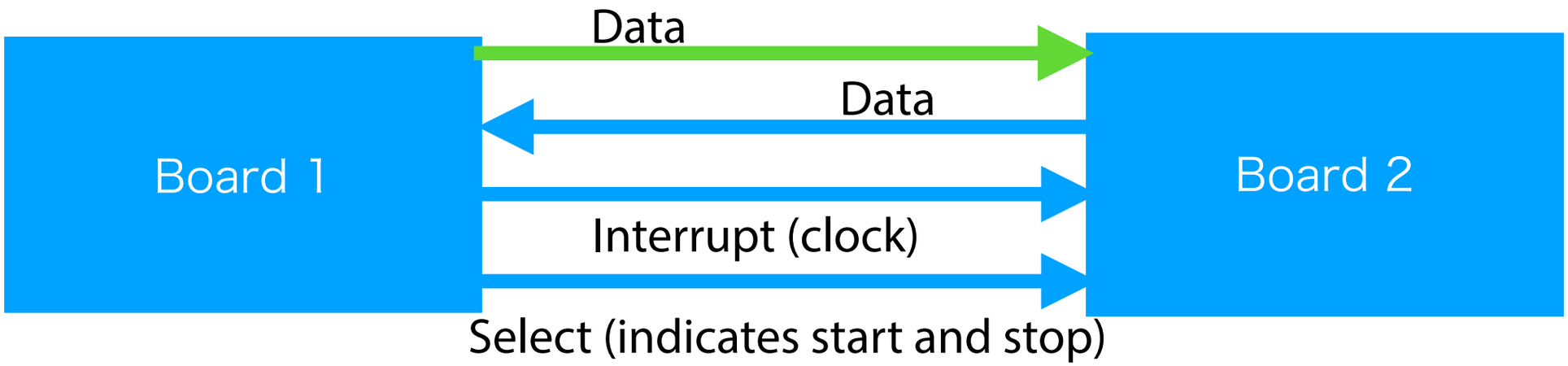




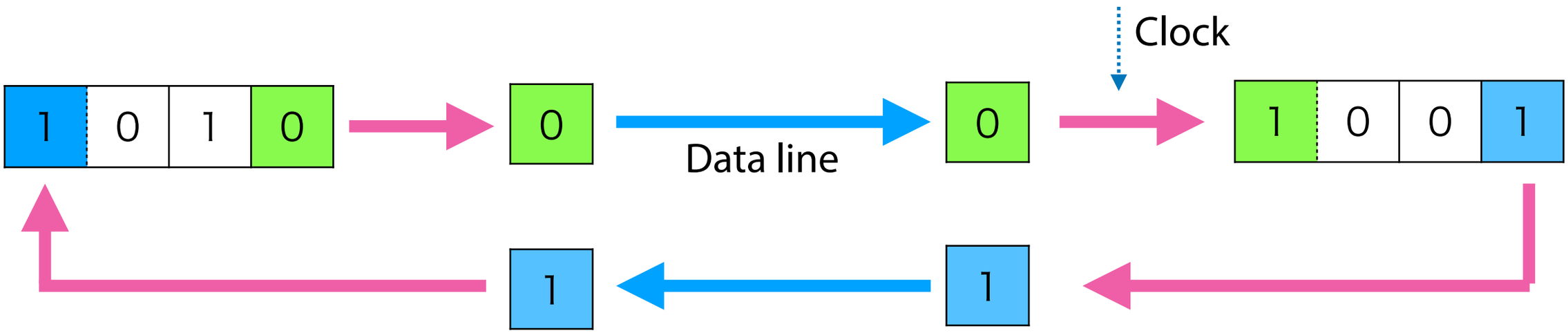
c) 4-Wire Serial Communication



c) 4-Wire Serial Communication



- Bidirectional 1-bit data transfer from each other.
- Cyclic data transfer from 1 to 2 and 2 to 1.

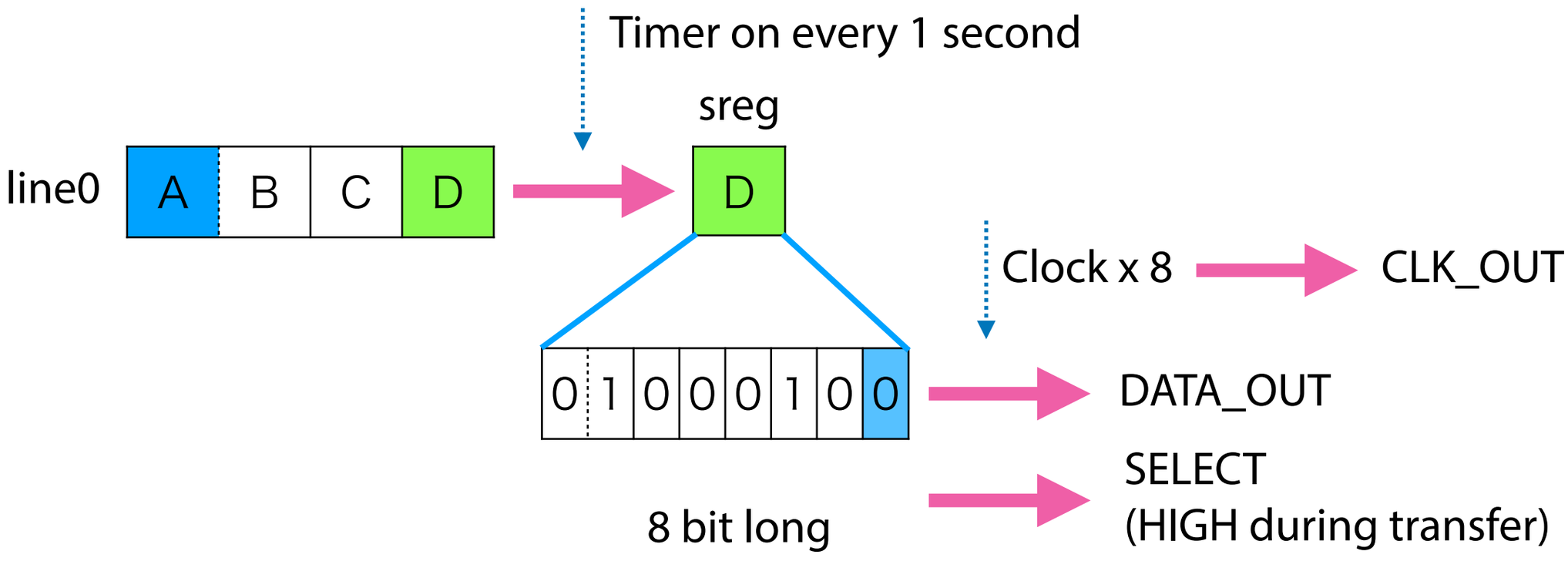


A data boundary can be defined by "Select" line

Board 1

line0

line1



Board 1 (example)

```
/* shift register, 8 bit long */
unsigned char sreg;
char line0[] = "0123456789012345";
char line1[] = "ABCDEFGHIJKLMNOP";
char linebuf[]="                ";
int count;
```

Characters for LCD

```
ISR (TIMER1_COMPA_vect) {
  int i;
```

Timer interrupt handler on every 10 ms

```
  /* send/recv in every 100 interrupts */
  if ((count++ % 100) != 0)
    return;
```

Transfer on every 1 s

```
digitalWrite(SELECT, HIGH);

/* Copy line0 from the 2nd char except
   for the right-most. */
memcpy(linebuf + 1, line0,
  sizeof(linebuf) - 2);
/* Get the right-most. */
sreg = line0[sizeof(line0) - 2];

for (i = 0; i < BITLEN(sreg); i++) {
  digitalWrite(CLK_OUT, HIGH);
/* Set LSB as data to be sent. */
 digitalWrite(DATA_OUT, sreg & 1);

 /* Generate a falling edge on CLK. */
 digitalWrite(CLK_OUT, LOW);

 /* Wait so that board 2 can update
    DATA_IN wire. */
 digitalWrite(CLK_OUT, LOW);
 digitalWrite(CLK_OUT, LOW);

 /* Read DATA_IN and update sreg */
   sreg >>= 1;
   if (digitalRead(DATA_IN) == HIGH)
     sreg |= 1 << (BITLEN(sreg) - 1);
 }
 digitalWrite(SELECT, LOW);

 /* Update the left-most char. */
 linebuf[0] = sreg;
 /* Update line0 */
 memcpy(line0, linebuf, sizeof(line0));
 update_lcd();
}
```

Board 1 (example)

```
/* shift register, 8 bit long */
unsigned char sreg;
char line0[] = "0123456789012345";
char line1[] = "ABCDEFGHIJKLMNOP";
char linebuf[]="                ";
int count;

ISR (TIMER1_COMPA_vect) {
  int i;

  /* send/recv in every 100 interrupts */
  if ((count++ % 100) != 0)
    return;

  digitalWrite(SELECT, HIGH);

  /* Copy line0 from the 2nd char except
     for the right-most. */
  memcpy(linebuf + 1, line0,
    sizeof(linebuf) - 2);
  /* Get the right-most. */
  sreg = line0[sizeof(line0) - 2];

  for (i = 0; i < BITLEN(sreg); i++) {
    digitalWrite(CLK_OUT, HIGH);

    /* Set LSB as data to be sent. */
    digitalWrite(DATA_OUT, sreg & 1);

    /* Generate a falling edge on CLK. */
    digitalWrite(CLK_OUT, LOW);

    /* Wait so that board 2 can update
       DATA_IN wire. */
    digitalWrite(CLK_OUT, LOW);
    digitalWrite(CLK_OUT, LOW);

    /* Read DATA_IN and update sreg */
    sreg >>= 1;
    if (digitalRead(DATA_IN) == HIGH)
      sreg |= 1 << (BITLEN(sreg) - 1);
  }
  digitalWrite(SELECT, LOW);

  /* Update the left-most char. */
  linebuf[0] = sreg;
  /* Update line0 */
  memcpy(line0, linebuf, sizeof(line0));
  update_lcd();
}
```

Timer interrupt handler on every 10 ms

Transfer on every 1 s

SELECT→HIGH

sreg ← right-most char

CLK → HIGH

each bit in sreg → DATA_OUT

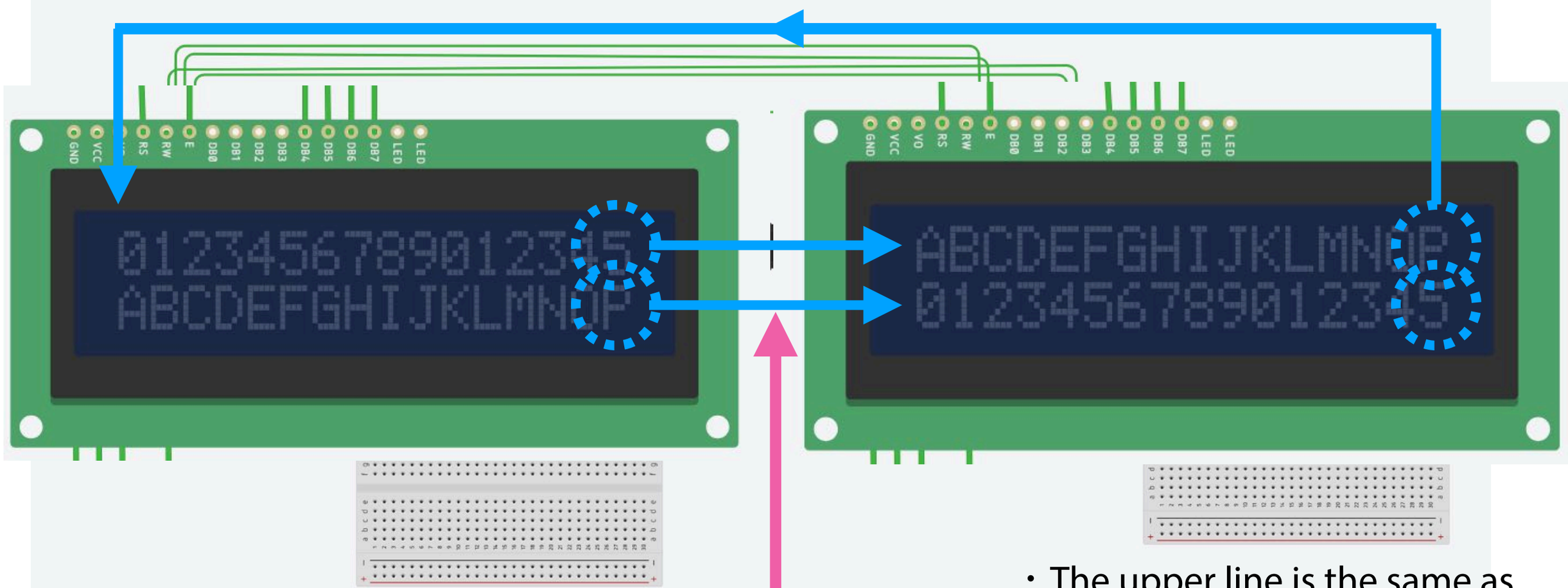
CLK → LOW

SELECT → LOW

Update chars on LCD

c+) 4-Wire Serial Communication

If you finished DESIGN-8C, try this:



- The upper line is the same as DESIGN-8C (driven by a timer).
- The lower line is switch-driven.

1) Add a switch to the Board 1.

2) Characters on the 2nd line will be transferred only when the switch is pressed

Conclusions

- For more complex systems: library and subroutine, concurrency and parallelism
- A solution: operating system for hardware abstraction and resource management

- **Next week**:
  Modern embedded systems by theory and examples

Time for Your Project

- Feel free to discuss with your friends
- If you have a question, ask the teaching assistant or just speak up.
- **Create a circuit for (c) and set the name as "DESIGN-8C".**
- **Complete your programs of (c) by December 1st.**