

# System-Level Programming

## 3 Java/Python vs. C – Some Examples

**J. Kleinöder, D. Lohmann, V. Sieh, P. Wägemann**

Lehrstuhl für Informatik 4  
Systemsoftware

Friedrich-Alexander-Universität  
Erlangen-Nürnberg

Summer Term 2024

<http://sys.cs.fau.de/lehre/ss24>



# The First C Program

- The most famous program of the world in **C**

```
#include <stdio.h>

int main(int argc, char **argv) {
    /* greet user */
    printf("Hello World!\n");
    return 0;
}
```



# The First C Program

- The most famous program of the world in **C**

```
#include <stdio.h>

int main(int argc, char **argv) {
    /* greet user */
    printf("Hello World!\n");
    return 0;
}
```

- Compilation and execution (on a UNIX system)

```
~> gcc -o hello hello.c
~> ./hello
Hello World!
~>
```

Not that complicated at all :-)



# The First C Program – a Comparison to Java

- The most famous program of the world in **C**

```
#include <stdio.h>

int main(int argc, char **argv) {
    /* greet user */
    printf("Hello World!\n");
    return 0;
}
```

- The most famous program of the world in **Java**

```
import java.lang.System;
class Hello {
    public static void main(String[] args) {
        /* greet user */
        System.out.println("Hello World!");
        return;
    }
}
```



# The First C Program – a Comparison to Java

- The most famous program of the world in **C**

```
#include <stdio.h>

int main(int argc, char **argv) {
    /* greet user */
    printf("Hello World!\n");
    return 0;
}
```

**C:** A C program starts with `main()`, a **global function** of type `int`, which is defined in exactly one **file**.

- The most famous program of the world in **Java**

```
import java.lang.System;
class Hello {
    public static void main(String[] args) {
        /* greet user */
        System.out.println("Hello World!");
        return;
    }
}
```

**Java:** Each Java program starts with the function `main()`, a **static method** of type `void`, which is defined in exactly one **class**.



# The First C Program – a Comparison to Java

- The most famous program of the world in **C**

```
#include <stdio.h>

int main(int argc, char **argv) {
    /* greet user */
    printf("Hello World!\n");
    return 0;
}
```

**C:** <no counterpart>

- The most famous program of the world in **Java**

```
import java.lang.System;
class Hello {
    public static void main(String[] args) {
        /* greet user */
        System.out.println("Hello World!");
        return;
    }
}
```

**Java:** Each Java program consists of at least one **class**.



# The First C Program – a Comparison to Java

- The most famous program of the world in **C**

```
#include <stdio.h>

int main(int argc, char **argv) {
    /* greet user */
    printf("Hello World!\n");
    return 0;
}
```

**C:** The output of the string takes place due to the **function** `printf()`. (`\n` ~> new line)

- The most famous program of the world in **Java**

```
import java.lang.System;
class Hello {
    public static void main(String[] args) {
        /* greet user */
        System.out.println("Hello World!");
        return;
    }
}
```

**Java:** The output of one string takes place in the **method** `println()` from the class `out`, which is from the package `System`.



# The First C Program – a Comparison to Java

- The most famous program of the world in **C**

```
#include <stdio.h>

int main(int argc, char **argv) {
    /* greet user */
    printf("Hello World!\n");
    return 0;
}
```

**C:** To use the function `printf()`, the **library** `stdio.h` gets included by the **preprocessor instruction** `#include`.

- The most famous program of the world in **Java**

```
import java.lang.System;
class Hello {
    public static void main(String[] args) {
        /* greet user */
        System.out.println("Hello World!");
        return;
    }
}
```

**Java:** To use the **class** `out`, the **package** `System` gets included by the `import` instruction.





# The First C Program – a Comparison to Java

- The most famous program of the world in **C**

```
#include <stdio.h>

int main(int argc, char **argv) {
    /* greet user */
    printf("Hello World!\n");
    return 0;
}
```

**C:** Return to the operating system with **return value**. 0 in this case indicates that no error has happened.

- The most famous program of the world in **Java**

```
import java.lang.System;
class Hello {
    public static void main(String[] args) {
        /* greet user */
        System.out.println("Hello World!");
        return;
    }
}
```

**Java:** Return to the operating system.

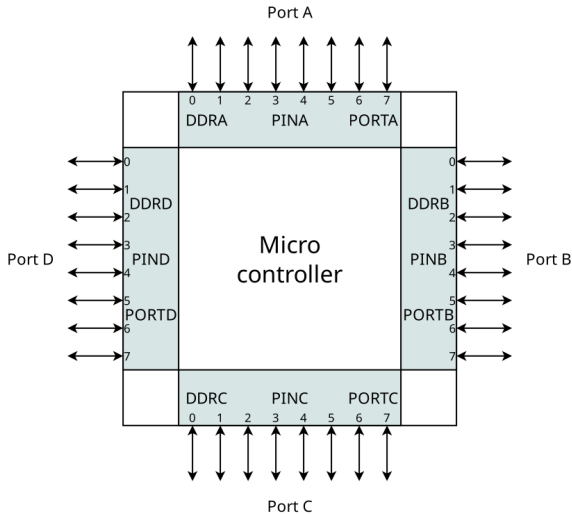


- **C** version gets explained line by line  
Return to the operating system with **return value**. 0 in this case indicates that no error has happened.
- **Java**-version gets explained line by line  
Return to the operating system.



# The First C Program for a $\mu$ Controller

Preliminary information:

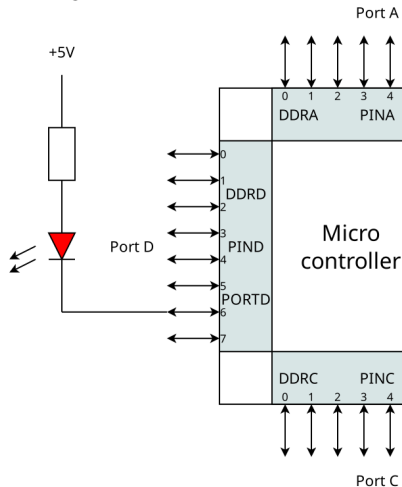


- DDRx: data direction register
- PINx: port input register
- PORTx: port output register (of size 8 bit each)



# The First C Program for a $\mu$ Controller

Background information:



- LED is not lit:
  - DDRD bit 6: '1' (output)
  - PORTD bit 6: '1' (5V)
- LED lights up:
  - DDRD bit 6: '1' (output)
  - PORTD bit 6: '0' (0V)



# The First C Program for a $\mu$ Controller

- “Hello world” for AVR ATmega (SPiCboard)

```
#include <avr/io.h>

void main(void) {
    /* initialize hardware: LED on port D pin 6, active low */
    DDRD  |= (1<<6); /* PD6 is used as output */
    PORTD |= (1<<6); /* PD6: high --> LED is off */

    /* greet user */
    PORTD &= ~(1<<6); /* PD6: low --> LED is on */

    /* wait forever */
    while (1) {
    }
}
```



# The First C Program for a $\mu$ Controller

- “Hello world” for AVR ATmega (SPiCboard)

```
#include <avr/io.h>

void main(void) {
    /* initialize hardware: LED on port D pin 6, active low */
    DDRD  |= (1<<6); /* PD6 is used as output */
    PORTD |= (1<<6); /* PD6: high --> LED is off */

    /* greet user */
    PORTD &= ~(1<<6); /* PD6: low --> LED is on */

    /* wait forever */
    while (1) {
    }
}
```

- Compilation and **flashing** (with SPiC-IDE)

↪ Exercises



# The First C Program for a $\mu$ Controller

- “Hello world” for AVR ATmega (SPiCboard)

```
#include <avr/io.h>

void main(void) {
    /* initialize hardware: LED on port D pin 6, active low */
    DDRD  |= (1<<6); /* PD6 is used as output */
    PORTD |= (1<<6); /* PD6: high --> LED is off */

    /* greet user */
    PORTD &= ~(1<<6); /* PD6: low --> LED is on */

    /* wait forever */
    while (1) {
    }
}
```

- Compilation and **flashing** (with SPiC-IDE)

↪ Exercises

- Execution (SPiCboard):  (red LED lit)



# The First C Program for a $\mu$ Controller

- “Hello world” for AVR ATmega (SPiCboard)

```
#include <avr/io.h>

void main(void) {
    /* initialize hardware: LED on port D pin 6, active low */
    DDRD  |= (1<<6); /* PD6 is used as output */
    PORTD |= (1<<6); /* PD6: high --> LED is off */

    /* greet user */
    PORTD &= ~(1<<6); /* PD6: low --> LED is on */

    /* wait forever */
    while (1) {
    }
}
```

$\mu$ Controller programming is “somewhat different”.

- Compilation and **flashing** (with SPiC-IDE) ↪ Exercises
- Execution (SPiCboard):  (red LED lit)





# The First C Program for a $\mu$ Controller

- “Hello world” for AVR ATmega (compare  $\leftrightarrow$  3-1)

```
#include <avr/io.h>

void main(void) {
    /* initialize hardware: LED on port D pin 6, active low */
    DDRD  |= (1<<6); /* PD6 is used as output */
    PORTD |= (1<<6); /* PD6: high --> LED is off */

    /* greet user */
    PORTD &= ~(1<<6); /* PD6: low --> LED is on */

    /* wait forever */
    while (1) {
    }
}
```

The `main()`-function has **no return value** (type `void`). A  $\mu$ Controller program runs **indefinitely**  $\rightsquigarrow$  `main()` does not terminate.



# The First C Program for a $\mu$ Controller

- “Hello world” for AVR ATmega (compare  $\leftrightarrow$  3-1)

```
#include <avr/io.h>

void main(void) {
    /* initialize hardware: LED on port D pin 6, active low */
    DDRD  |= (1<<6); /* PD6 is used as output */
    PORTD |= (1<<6); /* PD6: high --> LED is off */

    /* greet user */
    PORTD &= ~(1<<6); /* PD6: low --> LED is on */

    /* wait forever */
    while (1) {
    }
}
```

There will be **no return** to an operating system (which one?). The endless loop assures that `main()` does not terminate.



# The First C Program for a $\mu$ Controller

- “Hello world” for AVR ATmega (compare  $\leftrightarrow$  3-1)

```
#include <avr/io.h>

void main(void) {
    /* initialize hardware: LED on port D pin 6, active low */
    DDRD  |= (1<<6); /* PD6 is used as output */
    PORTD |= (1<<6); /* PD6: high --> LED is off */

    /* greet user */
    PORTD &= ~(1<<6); /* PD6: low --> LED is on */

    /* wait forever */
    while (1) {
    }
}
```

First, the **hardware** gets initialized (i. e., put in a pre-defined state). For this, **single bits** in certain **hardware registers** have to be changed.



# The First C Program for a $\mu$ Controller

- “Hello world” for AVR ATmega (compare  $\leftrightarrow$  3-1)

```
#include <avr/io.h>

void main(void) {
    /* initialize hardware: LED on port D pin 6, active low */
    DDRD  |= (1<<6); /* PD6 is used as output */
    PORTD |= (1<<6); /* PD6: high --> LED is off */

    /* greet user */
    PORTD &= ~(1<<6); /* PD6: low --> LED is on */

    /* wait forever */
    while (1) {
    }
}
```

The interaction with the environment (in this case: switching on the LED) takes place by **manipulating single bits** in hardware registers.



# The First C Program for a $\mu$ Controller

- “Hello world” for AVR ATmega (compare  $\leftrightarrow$  3-1)

```
#include <avr/io.h>

void main(void) {
    /* initialize hardware: LED on port D pin 6, active low */
    DDRD  |= (1<<6); /* PD6 is used as output */
    PORTD |= (1<<6); /* PD6: high --> LED is off */

    /* greet user */
    PORTD &= ~(1<<6); /* PD6: low --> LED is on */

    /* wait forever */
    while (1) {
    }
}
```

To access the hardware registers (DDRD, PORTD, provided as **global variables**), the **library** `avr/io.h` gets included with `#include`.



## The Second C Program – Input with Linux

- user interaction (reading one character) with Linux:

```
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Press key: ");
    char key = getchar();

    printf("You pressed %c\n", key);
    return 0;
}
```



# The Second C Program – Input with Linux

- user interaction (reading one character) with Linux:

```
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Press key: ");
    char key = getchar();

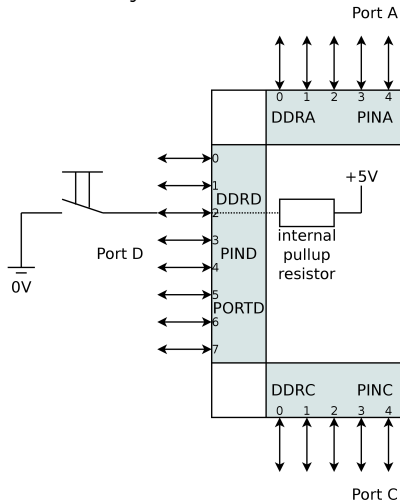
    printf("You pressed %c\n", key);
    return 0;
}
```

The `getchar()`-function reads one character from the standard input (here: keyboard). The function “waits”, if necessary, until a character is available.



# The Second C Program for a $\mu$ Controller

Preliminary information:



- Initialising:
  - DDRD bit 2: '0' (input)
  - PORTD bit 2: '1' (pull-up switched on)
- Detection:
  - PIND bit 2: '1' => button not pressed
  - PIND bit 2: '0' => button pressed





## The Second C Program – Input with $\mu$ Controller

- User interaction (waiting for a button to be pressed) on the SPiCboard:

```
#include <avr/io.h>

void main(void) {
    /* initialize hardware: button on port D pin 2 */
    DDRD  &= ~(1 << 2); /* PD2 is used as input */
    PORTD |= (1 << 2); /* activate pull-up: PD2: high */

    /* initialize hardware: LED on port D pin 6, active low */
    DDRD  |= (1 << 6); /* PD6 is used as output */
    PORTD |= (1 << 6); /* PD6: high --> LED is off */

    /* wait until PD2 -> low (button is pressed) */
    while ((PIND >> 2) & 1) {
    }

    /* greet user */
    PORTD &= ~(1 << 6); /* PD6: low --> LED is on */

    /* wait forever */
    while (1) {
    }
}
```



# The Second C Program – Input with $\mu$ Controller

- User interaction (waiting for a button to be pressed) on the SPiCboard:

```
#include <avr/io.h>

void main(void) {
    /* initialize hardware: button on port D pin 2 */
    DDRD  &= ~(1 << 2); /* PD2 is used as input */
    PORTD |= (1 << 2); /* activate pull-up: PD2: high */

    /* initialize hardware: LED on port D pin 6, active low */
    DDRD  |= (1 << 6); /* PD6 is used as output */
    PORTD |= (1 << 6); /* PD6: high --> LED is off */

    /* wait until PD2 -> low (button is pressed) */
    while ((PIND >> 2) & 1) {
    }

    /* greet user */
    PORTD &= ~(1 << 6); /* PD6: low --> LED is on */

    /* wait forever */
    while (1) {
    }
}
```

Just like the LED, the button is connected to a **digital IO pin** of the  $\mu$ Controller. We now configure pin 2 at port D as an **input** by **deleting** the corresponding bits in the register DDRD.



# The Second C Program – Input with $\mu$ Controller

- User interaction (waiting for a button to be pressed) on the SPiCboard:

```
#include <avr/io.h>

void main(void) {
    /* initialize hardware: button on port D pin 2 */
    DDRD  &= ~(1 << 2); /* PD2 is used as input */
    PORTD |= (1 << 2); /* activate pull-up: PD2: high */

    /* initialize hardware: LED on port D pin 6, active low */
    DDRD  |= (1 << 6); /* PD6 is used as output */
    PORTD |= (1 << 6); /* PD6: high --> LED is off */

    /* wait until PD2 -> low (button is pressed) */
    while ((PIND >> 2) & 1) {
    }

    /* greet user */
    PORTD &= ~(1 << 6); /* PD6: low --> LED is on */

    /* wait forever */
    while (1) {
    }
}
```

By **setting** bit 2 in the register PORTD as 1, the internal pull-up resistor (high resistance) gets activated. Which is connected to  $V_{CC} \rightsquigarrow$  PD2 = high.



# The Second C Program – Input with $\mu$ Controller

- User interaction (waiting for a button to be pressed) on the SPiCboard:

```
#include <avr/io.h>

void main(void) {
    /* initialize hardware: button on port D pin 2 */
    DDRD  &= ~(1 << 2); /* PD2 is used as input */
    PORTD |= (1 << 2); /* activate pull-up: PD2: high */

    /* initialize hardware: LED on port D pin 6, active low */
    DDRD  |= (1 << 6); /* PD6 is used as output */
    PORTD |= (1 << 6); /* PD6: high --> LED is off */

    /* wait until PD2 -> low (button is pressed) */
    while ((PIND >> 2) & 1) {

        /* greet user */
        PORTD &= ~(1 << 6); /* PD6: low --> LED is on */

        /* wait forever */
        while (1) {
        }
    }
}
```

**Active waiting:** waits for a button to be pressed, i. e., while PD2 (bit 2 in the register PIND) is *high*. When the button gets pressed, PD2 is pulled to ground  $\rightsquigarrow$  bit 2 in the register PIND is now *low* and the loop is exited.



- User interaction with the SPiCboard gets explained line by line
  - Active waiting:** waits for a button to be pressed, i. e., while PD2 (bit 2 in the register `PIND`) is *high*. When the button gets pressed, PD2 is pulled to ground  $\rightsquigarrow$  bit 2 in the register `PIND` is now *low* and the loop is exited.



## As a Reference: User Interaction as a Java-Program

```
import java.lang.System;
import javax.swing.*;
import java.awt.event.*;

public class Input implements ActionListener {
    private JFrame frame;

    public static void main(String[] args) {
        // create input, frame and button objects
        Input input = new Input();
        input.frame = new JFrame("Java Program");
        JButton button = new JButton("Press me");

        // add button to frame
        input.frame.add(button);
        input.frame.setSize(400, 400);
        input.frame.setVisible(true);

        // register input as listener of button events
        button.addActionListener(input);
    }

    public void actionPerformed(ActionEvent e) {
        System.out.println("Button pressed!");
        System.exit(0);
    }
}
```

Input as a “typical” Java program (**object-oriented, graphic**)



## As a Reference: User Interaction as a Java-Program

```
import java.lang.System;
import javax.swing.*;
import java.awt.event.*;

public class Input implements ActionListener {
    private JFrame frame;

    public static void main(String[] args) {
        // create input, frame and button objects
        Input input = new Input();
        input.frame = new JFrame("Java Program");
        JButton button = new JButton("Press me");

        // add button to frame
        input.frame.add(button);
        input.frame.setSize(400, 400);
        input.frame.setVisible(true);

        // register input as listener of button events
        button.addActionListener(input);
    }

    public void actionPerformed(ActionEvent e) {
        System.out.println("Button pressed!");
        System.exit(0);
    }
}
```

Input as a “typical” Java program (**object-oriented, graphic**)

The class `Input` implements an **interface** to receive interaction events.



## As a Reference: User Interaction as a Java-Program

```
import java.lang.System;
import javax.swing.*;
import java.awt.event.*;

public class Input implements ActionListener {
    private JFrame frame;

    public static void main(String[] args) {
        // create input, frame and button objects
        Input input = new Input();
        input.frame = new JFrame("Java Program");
        JButton button = new JButton("Press me");

        // add button to frame
        input.frame.add(button);
        input.frame.setSize(400, 400);
        input.frame.setVisible(true);

        // register input as listener of button events
        button.addActionListener(input);
    }

    public void actionPerformed(ActionEvent e) {
        System.out.println("Button pressed!");
        System.exit(0);
    }
}
```

Input as a “typical” Java program (**object-oriented, graphic**)

The program behaviour is implemented with the help of a multitude of **objects** (frame, button, input), which are created during initialization.





## As a Reference: User Interaction as a Java-Program

```
import java.lang.System;
import javax.swing.*;
import java.awt.event.*;

public class Input implements ActionListener {
    private JFrame frame;

    public static void main(String[] args) {
        // create input, frame and button objects
        Input input = new Input();
        input.frame = new JFrame("Java Program");
        JButton button = new JButton("Press me");

        // add button to frame
        input.frame.add(button);
        input.frame.setSize(400, 400);
        input.frame.setVisible(true);

        // register input as listener of button events
        button.addActionListener(input);
    }

    public void actionPerformed(ActionEvent e) {
        System.out.println("Button pressed!");
        System.exit(0);
    }
}
```

Input as a “typical” Java program (**object-oriented, graphic**)

The created button-object sends a message to the input-object.



## As a Reference: User Interaction as a Java-Program

```
import java.lang.System;
import javax.swing.*;
import java.awt.event.*;

public class Input implements ActionListener {
    private JFrame frame;

    public static void main(String[] args) {
        // create input, frame and button objects
        Input input = new Input();
        input.frame = new JFrame("Java Program");
        JButton button = new JButton("Press me");

        // add button to frame
        input.frame.add(button);
        input.frame.setSize(400, 400);
        input.frame.setVisible(true);

        // register input as listener of button events
        button.addActionListener(input);
    }

    public void actionPerformed(ActionEvent e) {
        System.out.println("Button pressed!");
        System.exit(0);
    }
}
```

Input as a “typical” Java program (**object-oriented, graphic**)

The button press gets signaled by an `actionPerformed()`-message (method call).



- The program cannot be compared to its counterpart in C directly.
  - It uses the (already known to you) **object-oriented paradigm**, which is typical for Java.
  - This difference shall be emphasised here.
- User interaction in Java explained line by line
  - The button press gets signaled by an `actionPerformed()`-message (method call).



# A First Conclusion: From Java → C (Syntax)

- Java and C have similar **syntax**  
(Syntax: “What do **valid** programs of the language look like?”)
- C syntax was used as a reference for the development of Java  
~> many language elements are similar or identical
  - blocks, loops, conditions, statements, literals
  - these elements will be looked at in detail in the following chapters
- Major elements from Java are **not** present in C
  - classes, packages, objects, exceptions, . . .



# A First Conclusion: From Java → C (Idiomatic)

- There are major **idiomatic** differences (Idiomatic: “What do programs of the language **usually** look like?”)
- **Java: object-oriented paradigm**
  - Central question: From which **things** is a problem made of?
  - Segmentation of the problem in **classes** and **objects**
  - Hierarchy by **inheritance** and **aggregation**
  - Program flow by interaction between **objects**
  - Re-usability through extensive **class libraries**
- **C: imperative paradigm**
  - Central question: From which **steps** is the problem made up?
  - Segmentation of the problem in **functions** and **variables**
  - Hierarchy by breakdown into **functions**
  - Program flow through calls between **functions**
  - Re-usability through **function libraries**



# A First Conclusion: From Java → C (Philosophy)

- There are **philosophical** differences as well (Philosophy: “Basic ideas and concepts of a language”)
- **Java:** Security and portability due to **abstracting from machine**
  - Compilation for **virtual machines** (JVM)
  - **Extensive** checks for programming errors during runtime
    - range overflow, division by 0, . . .
  - **Problem-centric** memory model
    - Only type-safe memory accesses, automatic garbage collection during runtime.
- **C:** efficiency and lightweight due to **machine orientation**
  - Compilation for **concrete hardware architecture**
  - **No** checks for programming errors during runtime
    - some error are caught by the operating system – **if present**
  - Memory model **close** to the machine
    - **pointers** provide direct memory access
    - coarse-grained access protection and automatic garbage collection (at processor level) by an OS – **if present**



# A First Conclusion: $\mu$ Controller-Programming

C  $\mapsto$  machine orientation  $\mapsto$   $\mu$ C programming

The **machine orientation** of the language C especially shows when looking at  $\mu$ Controller programming!

- Only one program is running
  - On RESET the program is loaded directly from flash memory
  - Hardware has to be initialized by the program first
  - Shall never terminate (e. g., with the help of a infinite loop in `main()`)
- The solution is implemented close to the machine
  - Direct manipulation of single bits in hardware registers
  - Therefore detailed knowledge of *electrical wiring* is needed
  - No support of an operating system (like Linux)
  - Usually a low level of abstraction  $\rightsquigarrow$  error-prone... *but fast*



# A First Conclusion: $\mu$ Controller-Programming

C  $\mapsto$  machine orientation  $\mapsto$   $\mu$ C programming

The **machine orientation** of the language C especially shows when looking at  $\mu$ Controller programming!

- Only one program is running
  - On RESET the program is loaded directly from flash memory
  - Hardware has to be initialized by the program first
  - Shall never terminate (e. g., with the help of a infinite loop in `main()`)
- The solution is implemented close to the machine
  - Direct manipulation of single bits in hardware registers
  - Therefore detailed knowledge of *electrical wiring* is needed
  - No support of an operating system (like Linux)
  - Usually a low level of abstraction  $\rightsquigarrow$  error-prone... *but fast*

**Approach:** Higher abstraction with **problem-oriented libraries**

