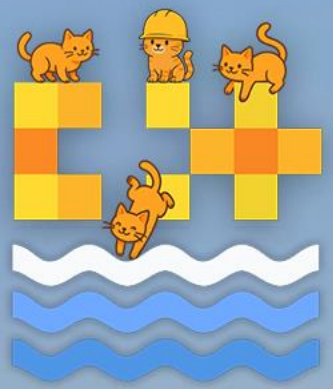


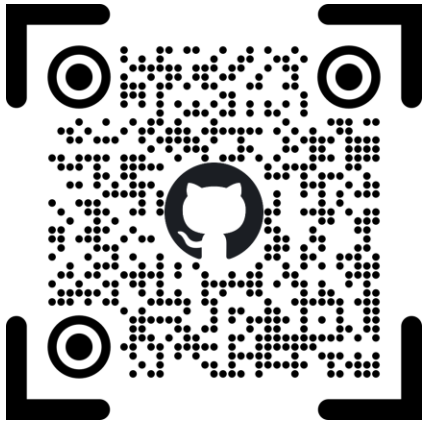
Balancing Efficiency and Flexibility

Cost of Abstractions in Embedded Systems

Marcell Juhasz

2025

whoami



Marcell Juhasz

marcelljuhasz.com
github.com/juhaszmarcell96
linkedin.com/in/juhaszmarcell
marcell.juhasz96@gmail.com

zühlke

Zühlke Engineering (Austria) GmbH

Rivergate, Handelskai 92
1200, Vienna, Austria
wien@zuehlke.com
+43 1 205 11 6800

Motivation

lambda

move semantics

direct interaction
with hardware

function overloading

large library support

exception
handling

auto

RAII

encapsulation

templates

C compatibility

polymorphism

namespace

references

inheritance

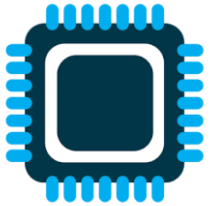
?

overhead

memory usage

runtime performance

Overview



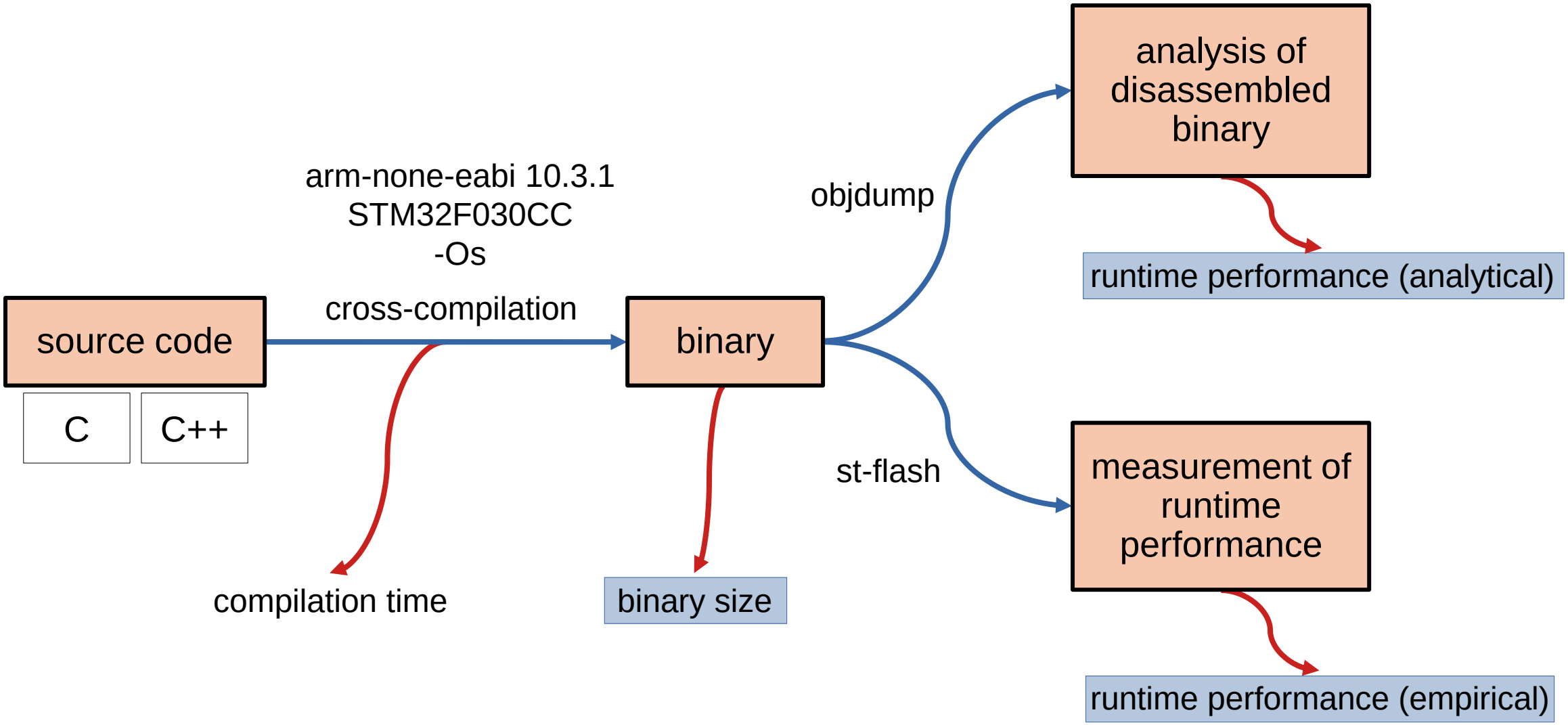
Hardware **A**bstraction **L**ayer

- Encapsulation
- Inheritance
- Polymorphism

+

- Template Metaprogramming
- Concepts
- Constant Expressions
- Immediate Functions
- Parameter Pack and Fold Expressions
- Constexpr If Statements

Methodology



Base Firmware

Absolute minimal embedded project:

- Main function:
  - Consists of a single, empty infinite loop
- Startup script:
  - Defines the vector table
  - Sets stack pointer
  - Copies data section from Flash to RAM
  - Initializes uninitialized global and static variables to zero
  - Calls static constructors
  - Calls *main()*
- Linker Script: specifies the memory

2092 bytes
- text: 524 bytes
- data: 0 bytes
- bss: 1568 bytes

Traditional HAL

```c
typedef struct {
    uint32_t pin;
    GPIO_Modes mode;
} GPIO_InitStruct;

void GPIO_Init(GPIO_InitStruct* conf) {
    uint32_t temp;

    /* check the values */
    if (!IS_GPIO_PIN(conf->pin)) { return; }
    if (!IS_GPIO_MODE(conf->mode)) { return; }

    /* configure the GPIO based on the settings */
    if (conf->mode == GPIO_MODE_OUTPUT) {
        temp = OSPEEDR;
        temp &= ~(OSPEEDR_MASK << (conf->pin * 2u));
        temp |= (GPIO_SPEED_FREQ_LOW << (conf->pin * 2u));
        OSPEEDR = temp;
        /* ... */
    }
    /* ... */
}
```

```c
int main (void) {
    GPIO_InitStruct conf = { 0 };

    conf.pin = GPIO_PIN_6;
    conf.mode = GPIO_MODE_INPUT;

    GPIO_Init(&conf) ;

    while (1) { }
}
```

Building Layers of Abstractions

Building Layers of Abstractions

generic register

```
temp = OSPEEDR;
temp &= ~(OSPEEDR_MASK << (conf->pin * 2u));
temp |= (conf->speed << (conf->pin * 2u));
OSPEEDR = temp;
```

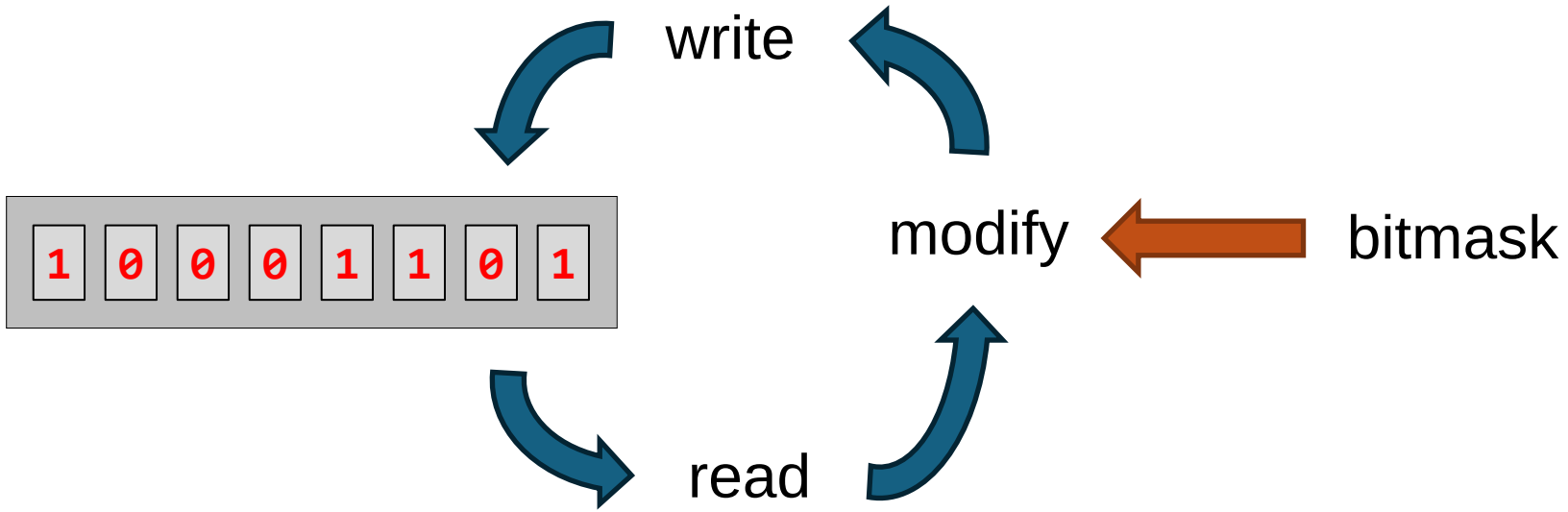
specific register

Building Layers of Abstractions

generic register

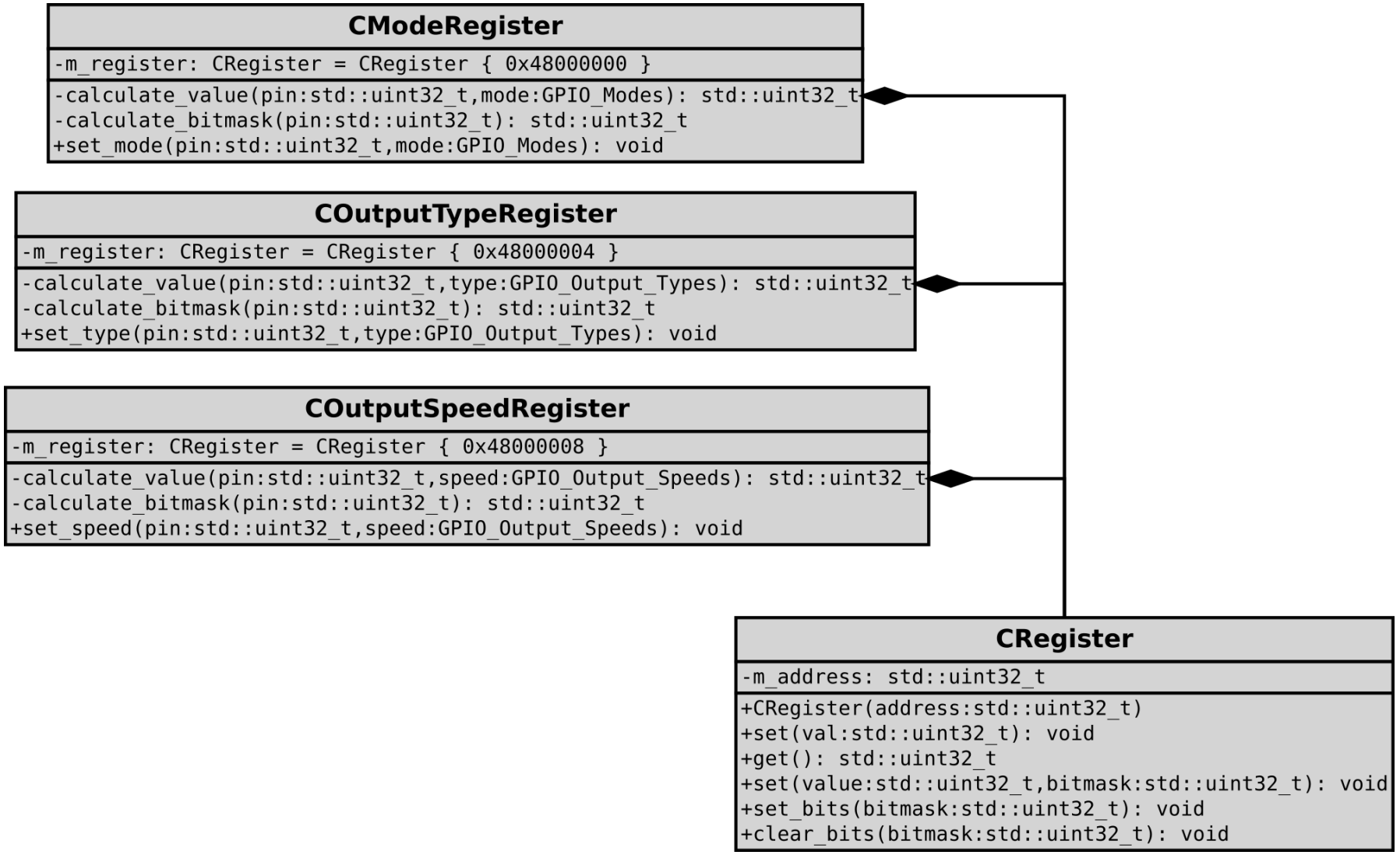
```
temp = OSPEEDR;
temp &= ~(OSPEEDR_MASK << (conf->pin * 2u));
temp |= (conf->speed << (conf->pin * 2u));
OSPEEDR = temp;
```

specific register



Encapsulation

Encapsulation

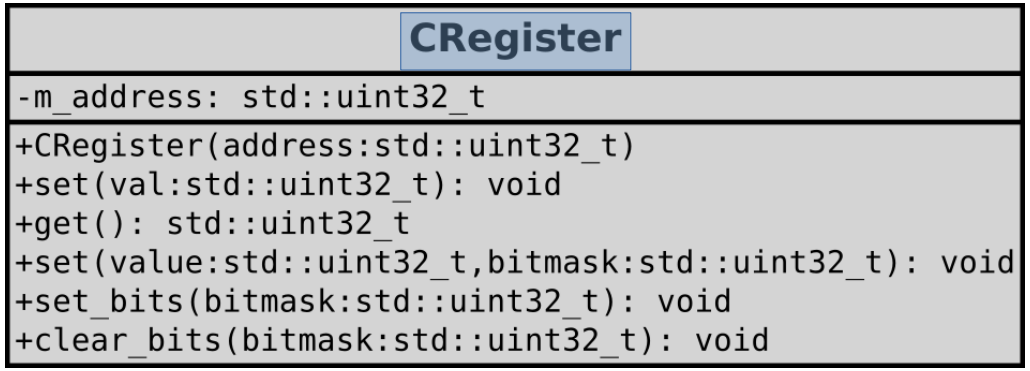


Encapsulation

(basic)

```
class CRegister {
private:
    const std::uint32_t m_address;
public:
    CRegister (std::uint32_t address) : m_address(address) { }
    void set (std::uint32_t val) const {
        *(reinterpret_cast<volatile std::uint32_t *>(m_address)) = val;
    }
    /* ... */
};
```

```
temp = OSPEEDR;
temp &= ~(OSPEEDR_MASK << (conf->pin * 2u));
temp |= (conf->speed << (conf->pin * 2u));
OSPEEDR = temp;
```

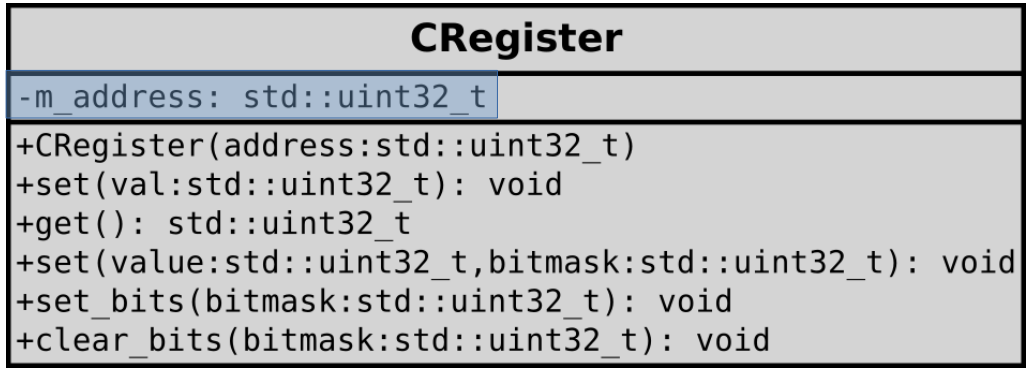


Encapsulation

(basic)

```
class CRegister {
private:
    const std::uint32_t m_address;
public:
    CRegister (std::uint32_t address) : m_address(address) { }
    void set (std::uint32_t val) const {
        *(reinterpret_cast<volatile std::uint32_t *>(m_address)) = val;
    }
    /* ... */
};
```

```
temp = OSPEEDR;
temp &= ~(OSPEEDR_MASK << (conf->pin * 2u));
temp |= (conf->speed << (conf->pin * 2u));
OSPEEDR = temp;
```

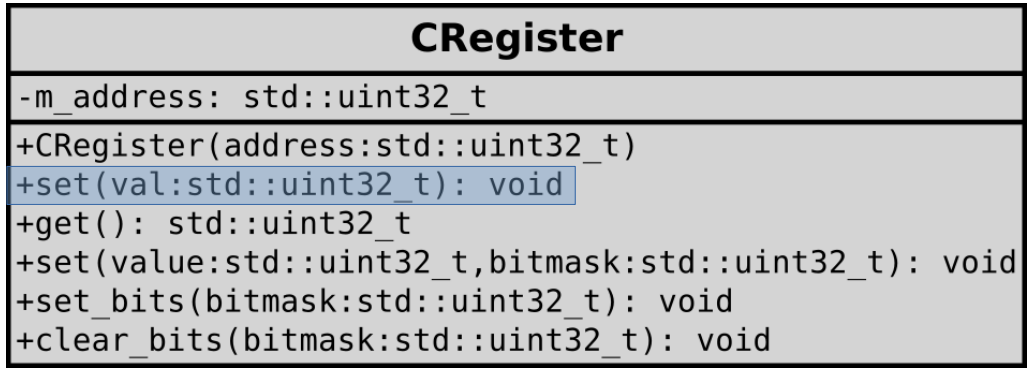


Encapsulation

(basic)

```
class CRegister {
private:
    const std::uint32_t m_address;
public:
    CRegister (std::uint32_t address) : m_address(address) { }
    void set (std::uint32_t val) const {
        *(reinterpret_cast<volatile std::uint32_t *>(m_address)) = val;
    }
    /* ... */
};
```

```
temp = OSPEEDR;
temp &= ~(OSPEEDR_MASK << (conf->pin * 2u));
temp |= (conf->speed << (conf->pin * 2u));
OSPEEDR = temp;
```

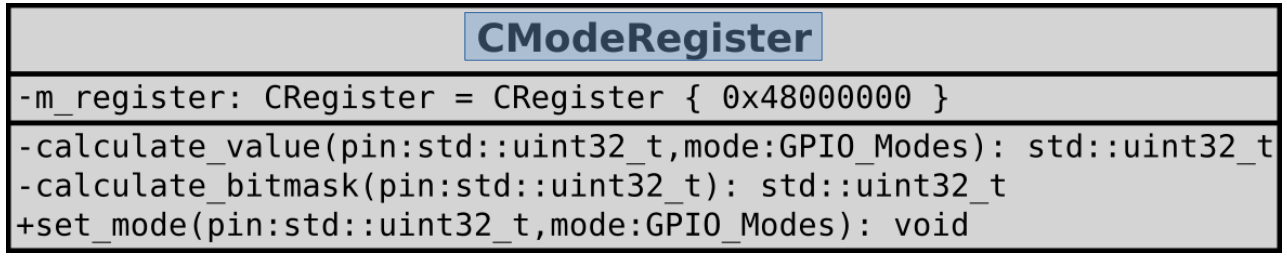


Encapsulation

(basic)

```
temp = OSPEEDR;
temp &= ~(OSPEEDR_MASK << (conf->pin * 2u));
temp |= (conf->speed << (conf->pin * 2u));
OSPEEDR = temp;
```

```
class CModeRegister {
private:
    const CRegister m_register { 0x48000000 };
    inline std::uint32_t calculate_value (std::uint32_t pin, GPIO_Modes mode) {
        return (mode & GPIO_MODE) << (pin * 2);
    }
    inline std::uint32_t calculate_bitmask (std::uint32_t pin) {
        return MODER_MASK << (pin * 2);
    }
public:
    inline void set_mode (std::uint32_t pin, GPIO_Modes mode) {
        m_register.set(calculate_value(pin, mode), calculate_bitmask(pin));
    }
};
```

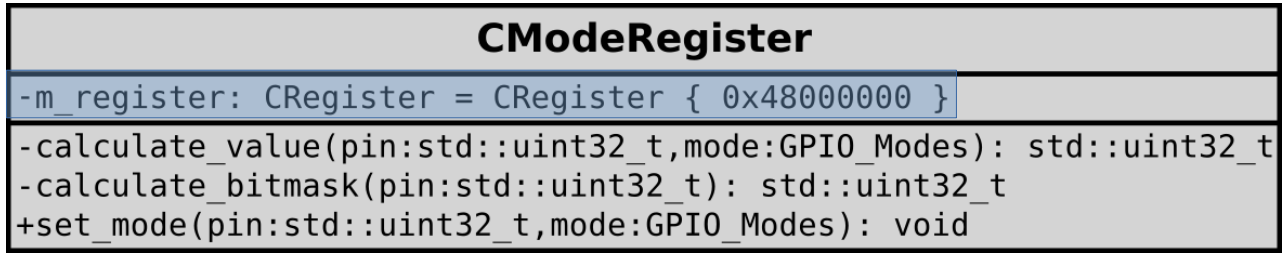


Encapsulation

(basic)

```
temp = OSPEEDR;
temp &= ~(OSPEEDR_MASK << (conf->pin * 2u));
temp |= (conf->speed << (conf->pin * 2u));
OSPEEDR = temp;
```

```
class CModeRegister {
private:
    const CRegister m_register { 0x48000000 };
    inline std::uint32_t calculate_value (std::uint32_t pin, GPIO_Modes mode) {
        return (mode & GPIO_MODE) << (pin * 2);
    }
    inline std::uint32_t calculate_bitmask (std::uint32_t pin) {
        return MODER_MASK << (pin * 2);
    }
public:
    inline void set_mode (std::uint32_t pin, GPIO_Modes mode) {
        m_register.set(calculate_value(pin, mode), calculate_bitmask(pin));
    }
};
```

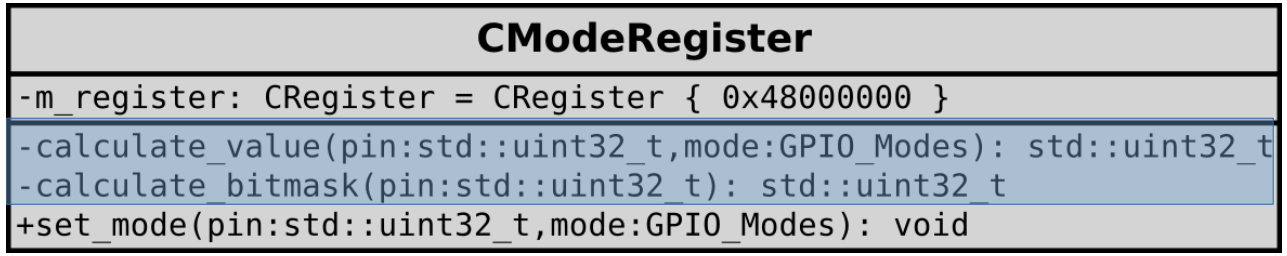


Encapsulation

(basic)

```
temp = OSPEEDR;
temp &= ~(OSPEEDR_MASK << (conf->pin * 2u));
temp |= (conf->speed << (conf->pin * 2u));
OSPEEDR = temp;
```

```
class CModeRegister {
private:
    const CRegister m_register { 0x48000000 };
    inline std::uint32_t calculate_value (std::uint32_t pin, GPIO_Modes mode) {
        return (mode & GPIO_MODE) << (pin * 2);
    }
    inline std::uint32_t calculate_bitmask (std::uint32_t pin) {
        return MODER_MASK << (pin * 2);
    }
public:
    inline void set_mode (std::uint32_t pin, GPIO_Modes mode) {
        m_register.set(calculate_value(pin, mode), calculate_bitmask(pin));
    }
};
```

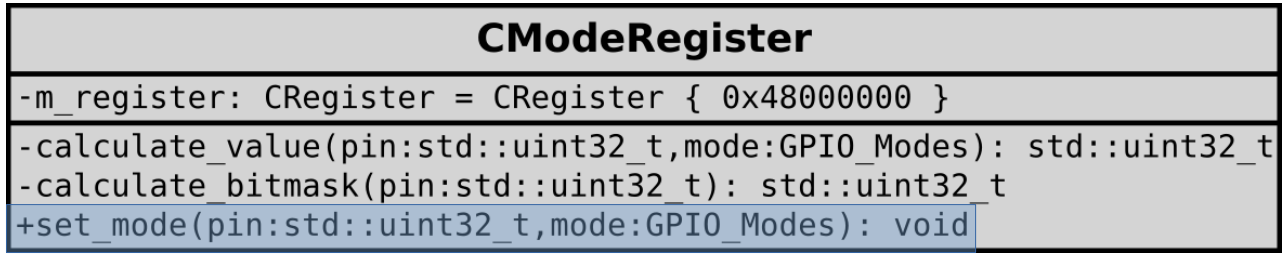


Encapsulation

(basic)

```
temp = OSPEEDR;
temp &= ~(OSPEEDR_MASK << (conf->pin * 2u));
temp |= (conf->speed << (conf->pin * 2u));
OSPEEDR = temp;
```

```
class CModeRegister {
private:
    const CRegister m_register { 0x48000000 };
    inline std::uint32_t calculate_value (std::uint32_t pin, GPIO_Modes mode) {
        return (mode & GPIO_MODE) << (pin * 2);
    }
    inline std::uint32_t calculate_bitmask (std::uint32_t pin) {
        return MODER_MASK << (pin * 2);
    }
public:
    inline void set_mode (std::uint32_t pin, GPIO_Modes mode) {
        m_register.set(calculate_value(pin, mode), calculate_bitmask(pin));
    }
};
```



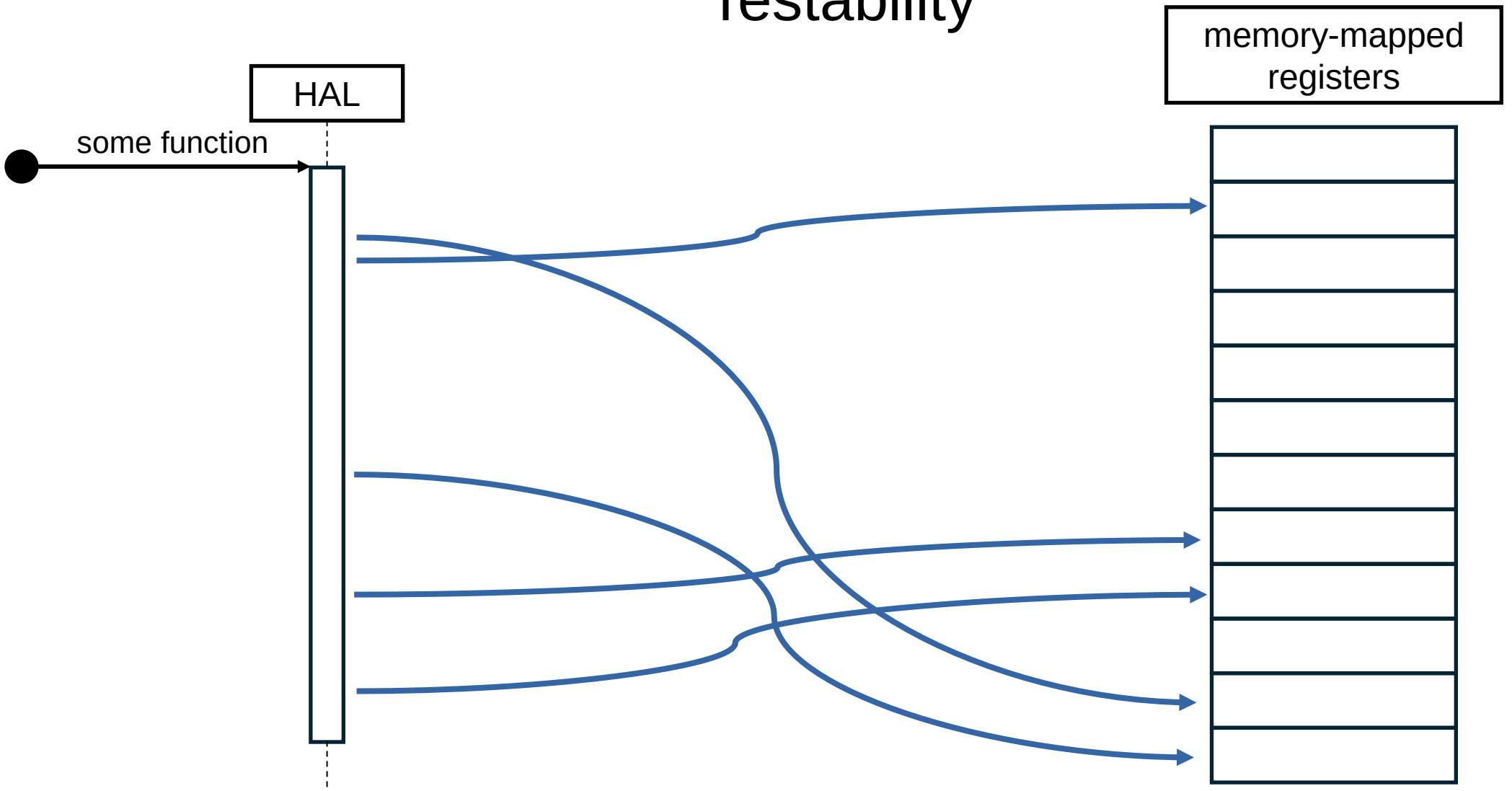
Encapsulation

(basic)

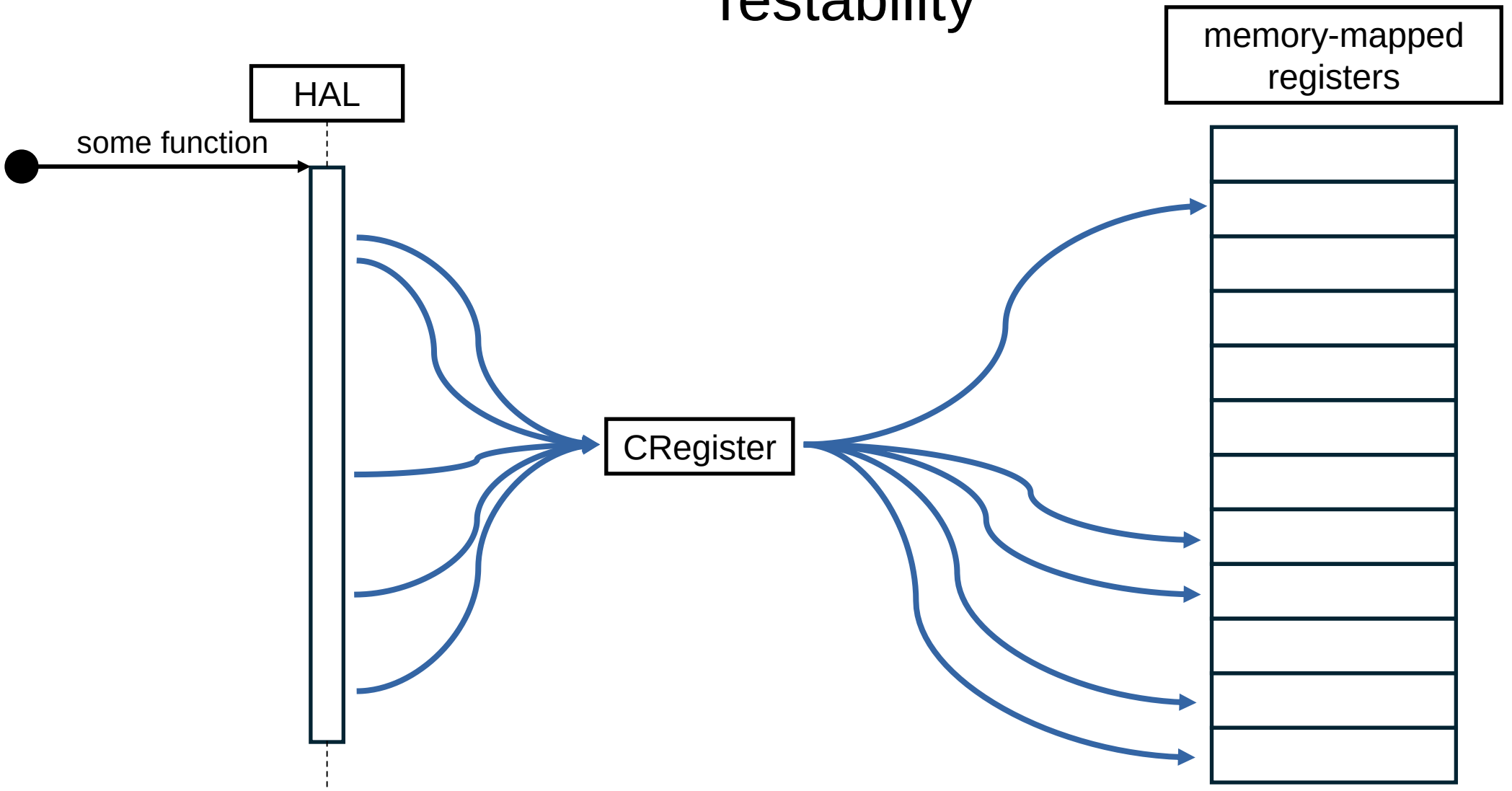
```
void GPIO_Init(GPIO_InitStruct* conf) {
    /* ... */
    if (conf->mode == GPIO_MODE_OUTPUT) {
        temp = OSPEEDR;
        temp &= ~(OSPEEDR_MASK << (conf->pin * 2u));
        temp |= (conf->speed << (conf->pin * 2u));
        OSPEEDR = temp;

        temp = OTYPER;
        temp &= ~(OTYPER_MASK << conf->pin);
        temp |= (((conf->type & GPIO_OUTPUT_TYPE) >> 4u)
                                        << conf->pin);
        OTYPER = temp;
    }
    temp = MODER;
    temp &= ~(MODER_MASK << (conf->pin * 2u));
    temp |= ((conf->mode & GPIO_MODE) << (conf->pin * 2u));
    MODER = temp;
    /* ... */
}
```

```
void GPIO_Init(GPIO_InitStruct* conf) {
    /* ... */
    if (conf->mode == GPIO_MODE_OUTPUT) {
        COutputSpeedRegister ospeedr {};
        ospeedr.set_speed(conf->pin, conf->speed);
        COutputTypeRegister otyper {};
        otyper.set_type(conf->pin, conf->type);
    }
    CModeRegister moder {};
    moder.set_mode(conf->pin, conf->mode);
    /* ... */
}
```

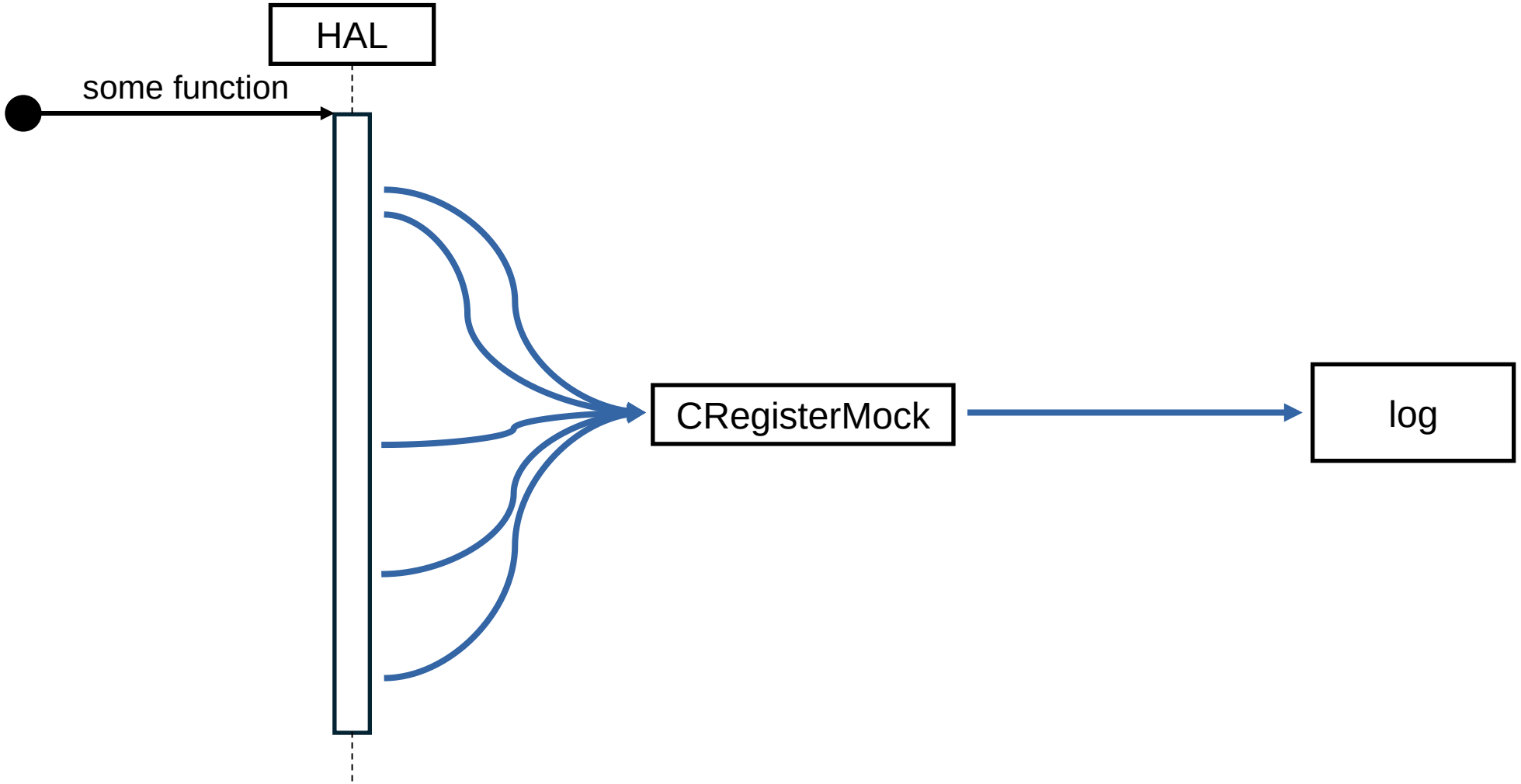
Testability



Testability



Testability



Encapsulation

(basic)

Compiles to the same binary.

Encapsulation

(static)

```
void GPIO_Init(GPIO_InitStruct* conf) {
    /* check the values */
    if (!IS_GPIO_PIN(conf->pin)) { return; }
    if (!IS_GPIO_MODE(conf->mode)) { return; }

    /* configure the GPIO based on the settings */
    if (conf->mode == GPIO_MODE_OUTPUT) {
        COutputSpeedRegister::set_speed(conf->pin, conf->speed);
        COutputTypeRegister::set_type(conf->pin, conf->type);
    }
    CModeRegister::set_mode(conf->pin, conf->mode);
    /* ... */
}
```

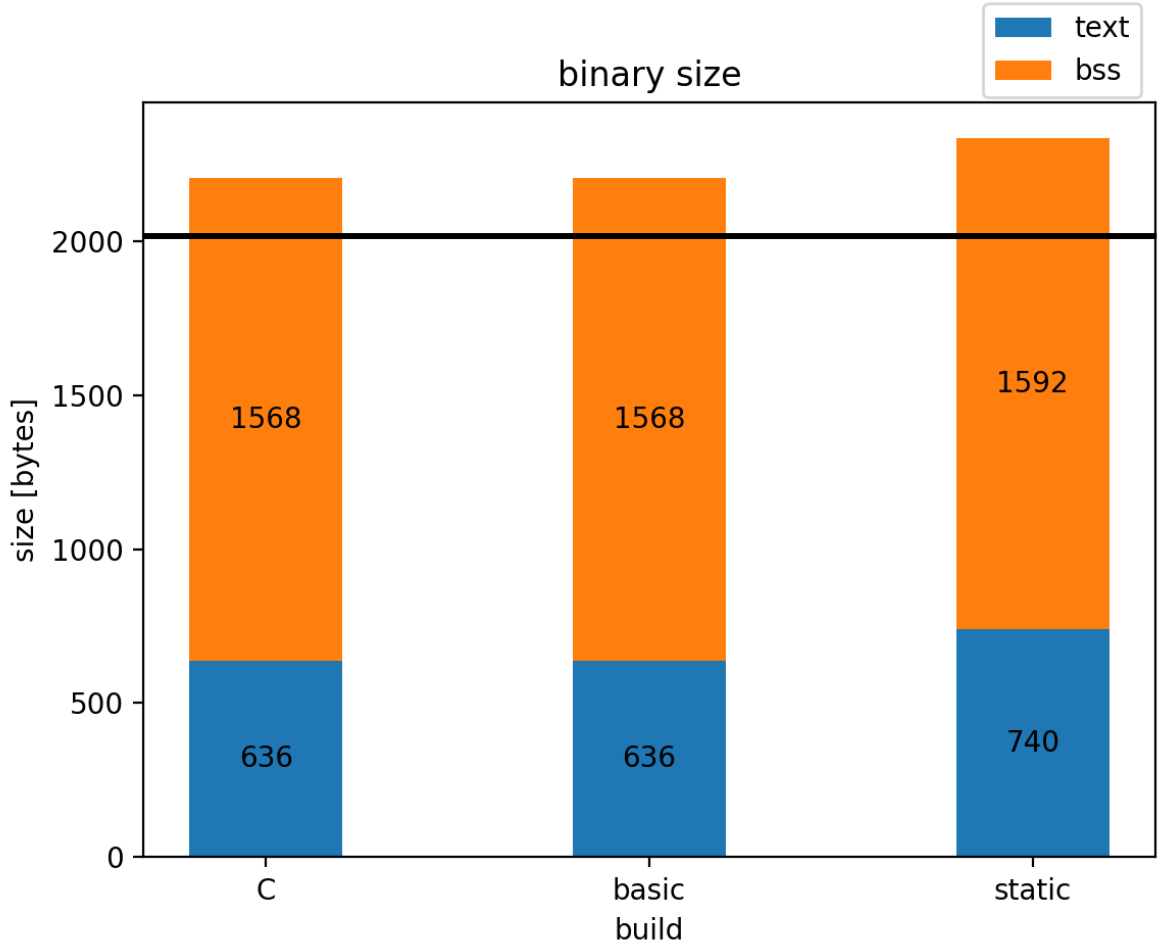
Encapsulation

(static)

```
class CModeRegister {
private:
    static inline const CRegister m_register { 0x48000000 };
    static inline std::uint32_t calculate_value (std::uint32_t pin, GPIO_Modes mode) {
        return (mode & GPIO_MODE) << (pin * 2);
    }
    static inline std::uint32_t calculate_bitmask (std::uint32_t pin) {
        return MODER_MASK << (pin * 2);
    }
public:
    static inline void set_mode (std::uint32_t pin, GPIO_Modes mode) {
        m_register.set(calculate_value(pin, mode), calculate_bitmask(pin));
    }
};
```

Encapsulation

(static)

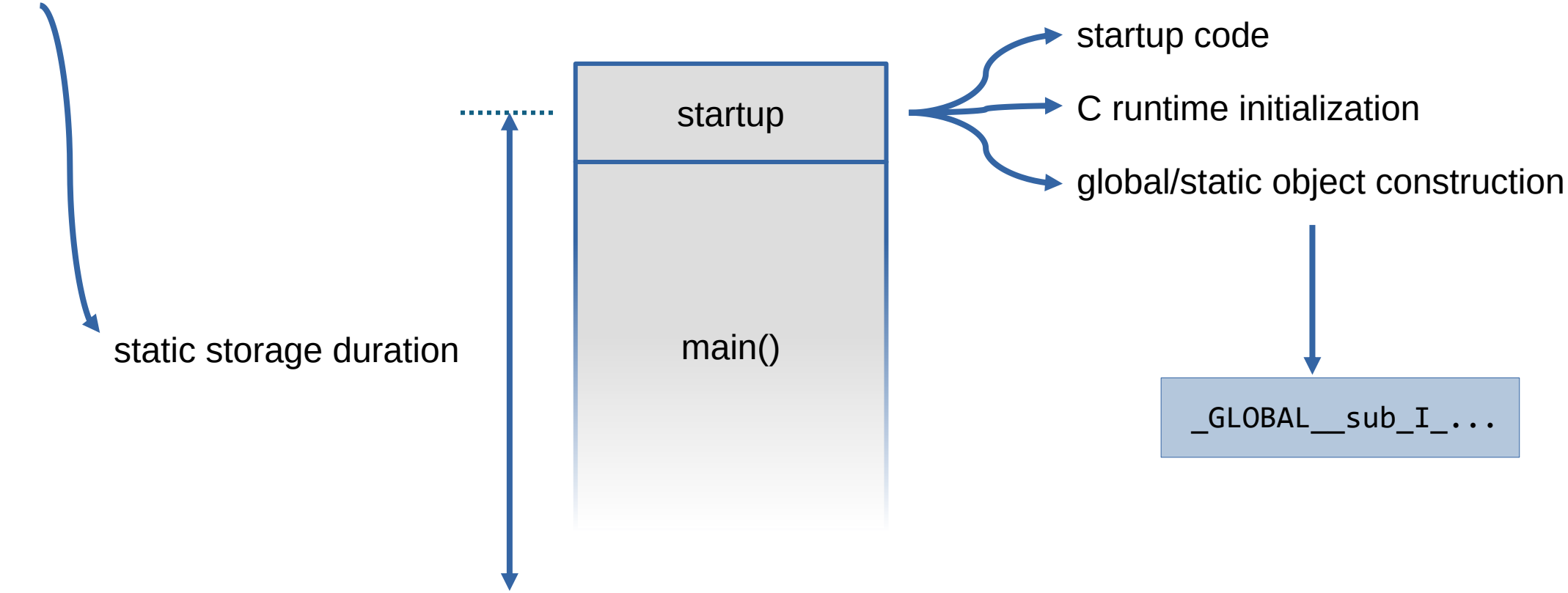


Binary size increase… why?

Encapsulation

(static)

```
static inline const CRegister m_register { 0x48000000 };
```



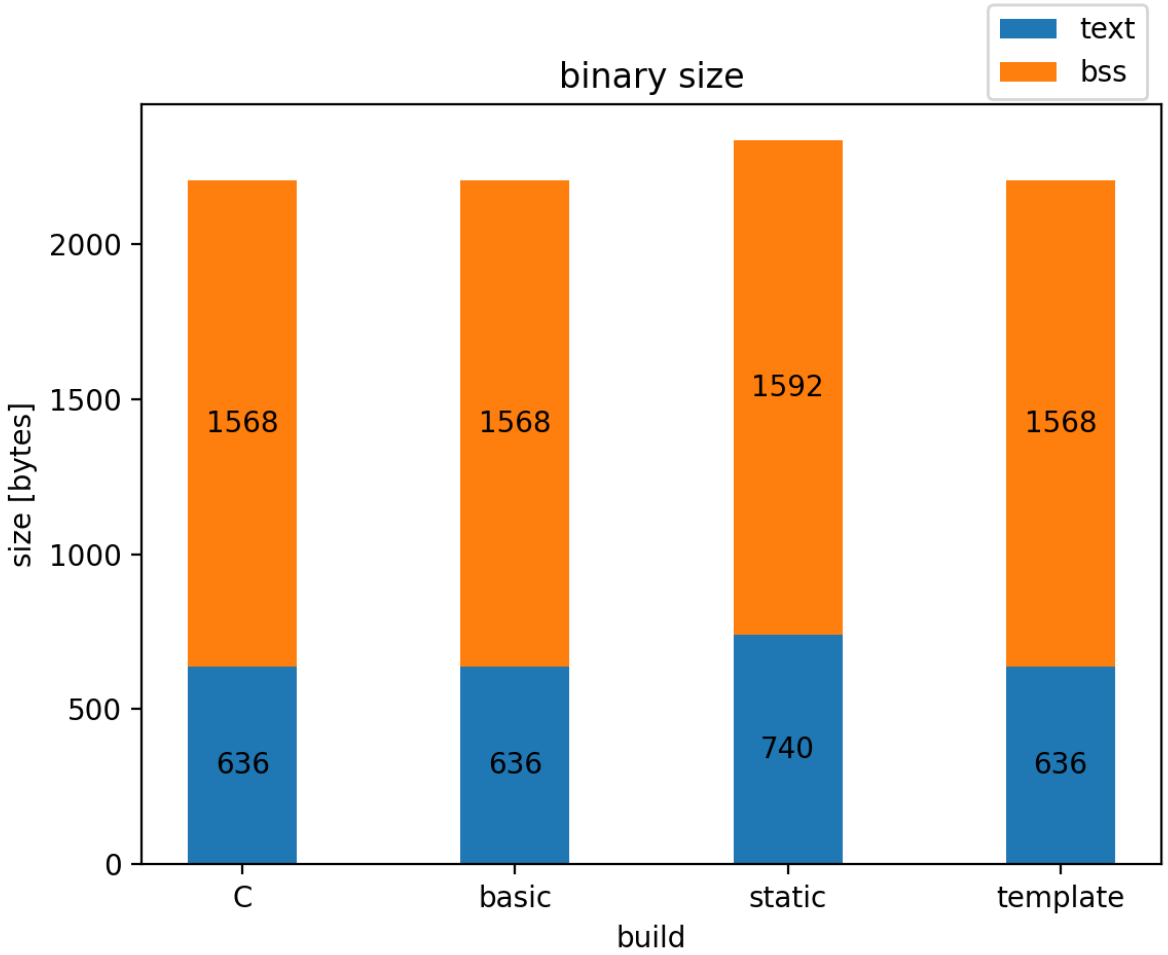
Encapsulation

(static & template)

```
template <std::uint32_t address>
class CRegister {
public:
    void set (std::uint32_t val) const {
        *(reinterpret_cast<volatile std::uint32_t *>(address)) = val;
    }
    /* ... */
};
```

Encapsulation

(static & template)



Binary size is back to the original.

Why?

- Each CRegister instance is fully defined at compile time.
- No global state that needs to be explicitly initialized before main.
- Compiler can better optimize the code at compile time.

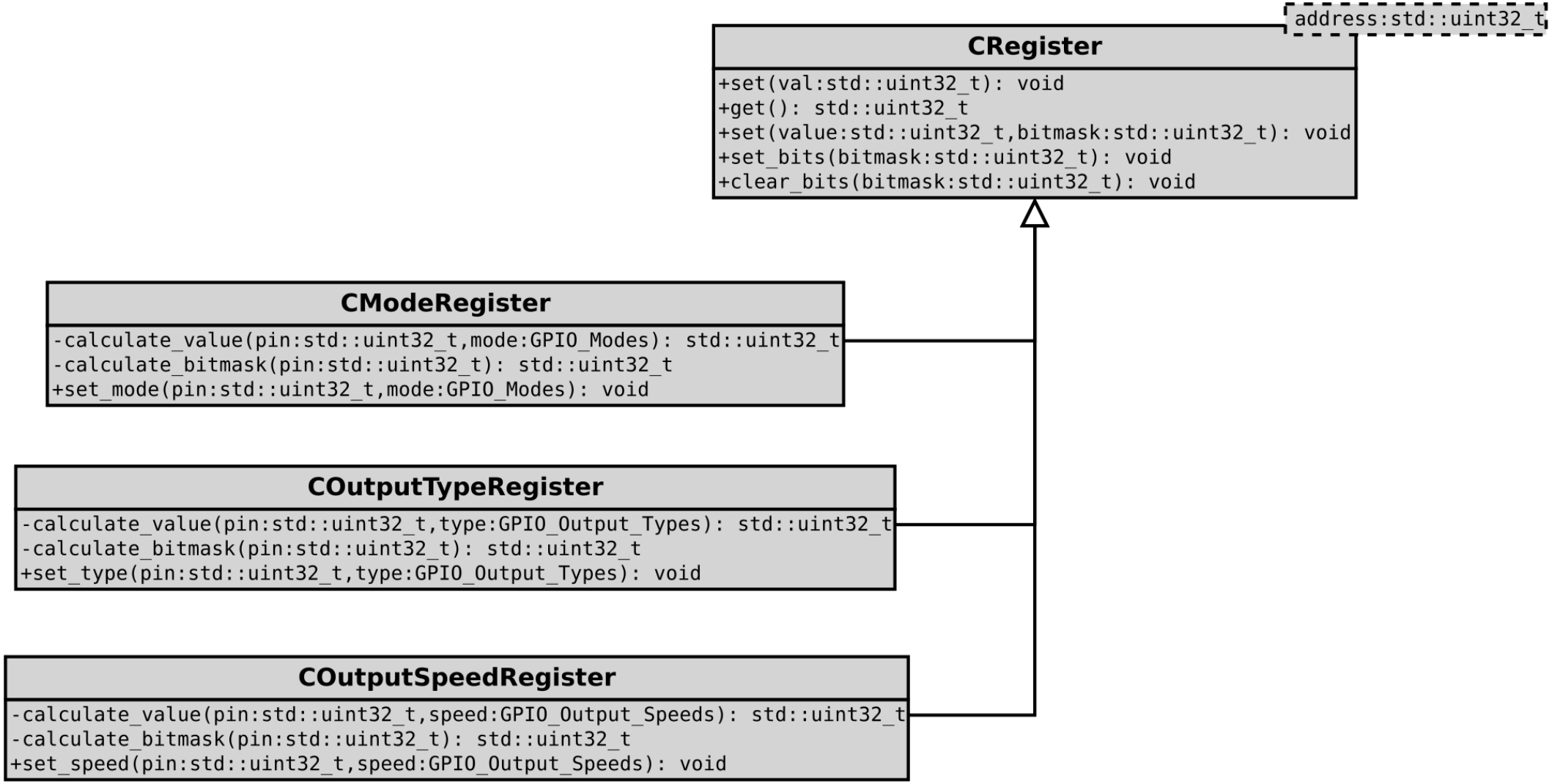
```
const std::uint32_t m_address;
```

vs

```
template <std::uint32_t address>
```

Inheritance

Inheritance



Inheritance

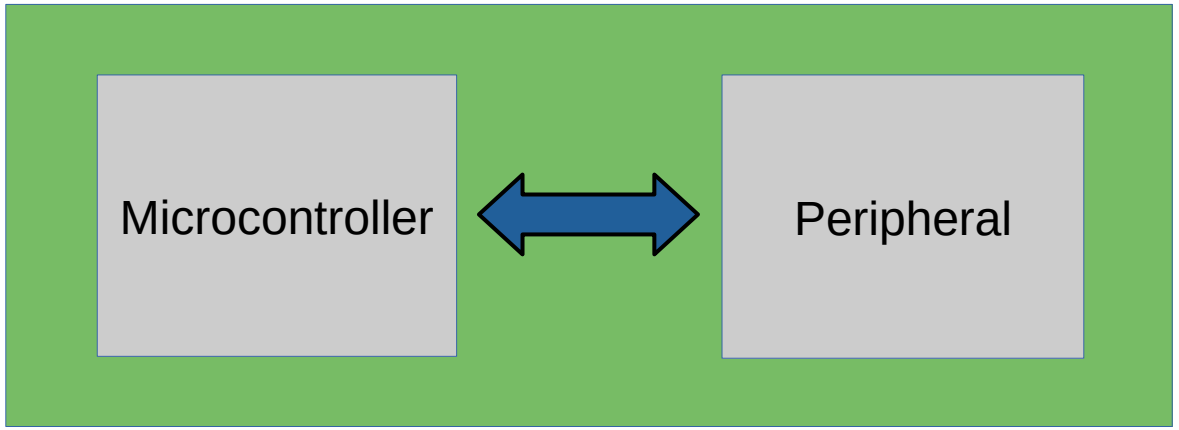
```
class COutputSpeedRegister : public CRegister<0x48000008> {
private:
    static inline std::uint32_t calculate_value (std::uint32_t pin, GPIO_Output_Speeds speed) {
        return speed << (pin * 2);
    }
    static inline std::uint32_t calculate_bitmask (std::uint32_t pin) {
        return OSPEEDR_MASK << (pin * 2);
    }
public:
    static inline void set_speed (std::uint32_t pin, GPIO_Output_Speeds speed) {
        set(calculate_value(pin, speed), calculate_bitmask(pin));
    }
};
```

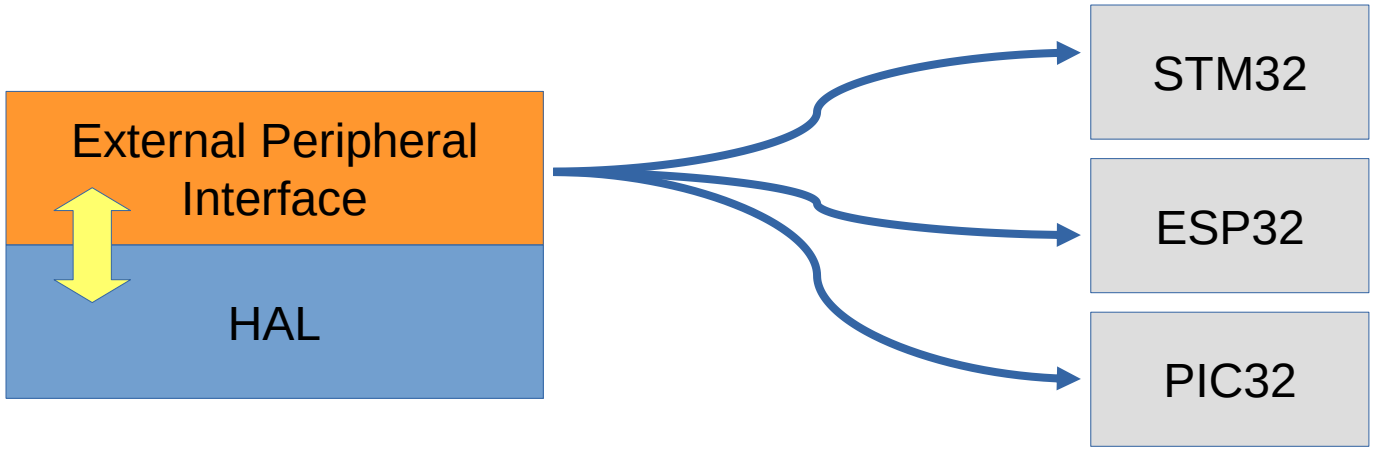
Inheritance

Compiles to the same binary.

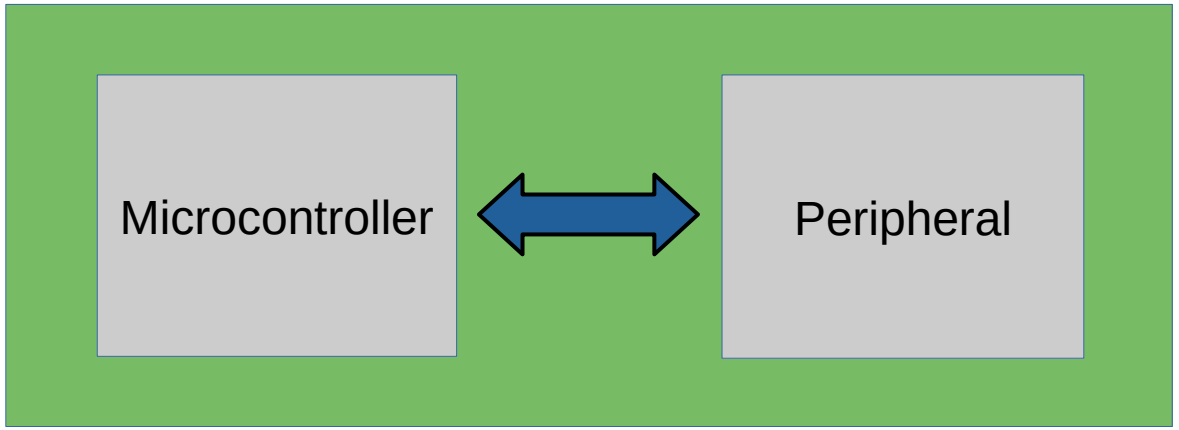
Polymorphism

Moving Higher

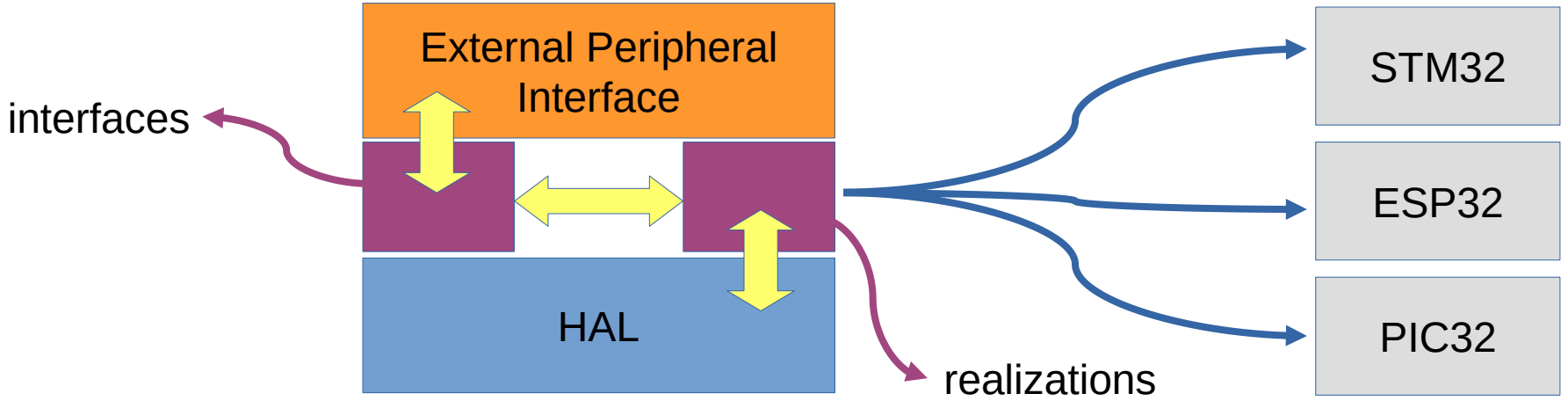




Moving Higher

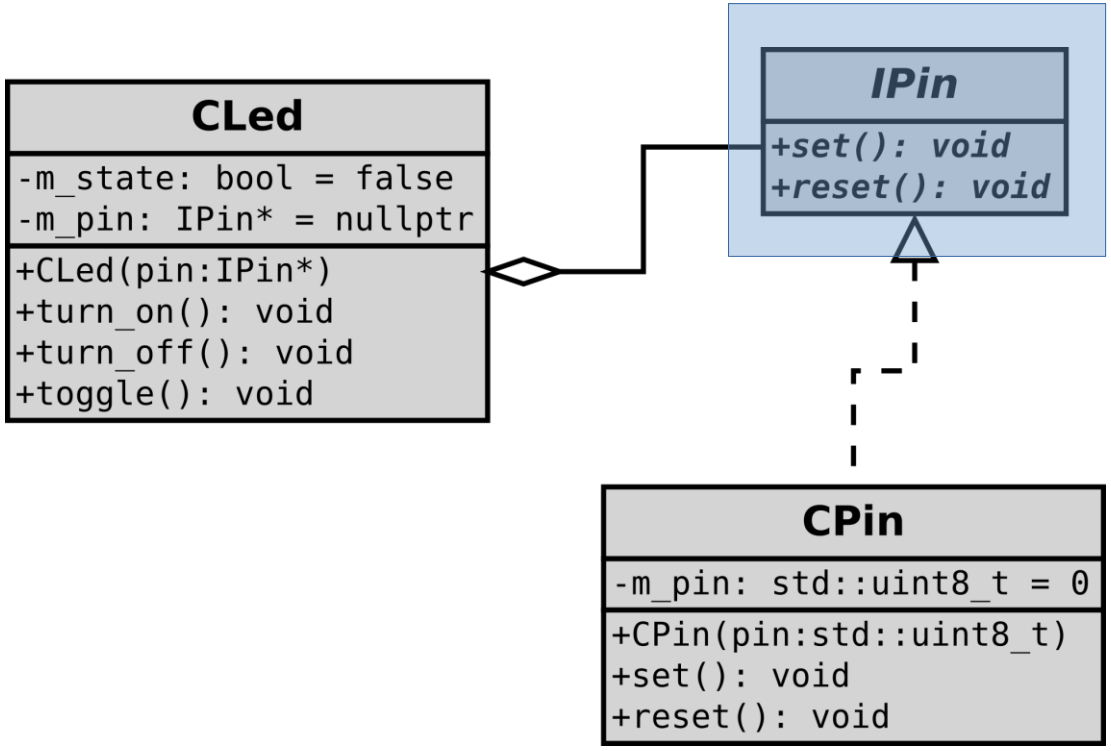


Dynamic Polymorphism



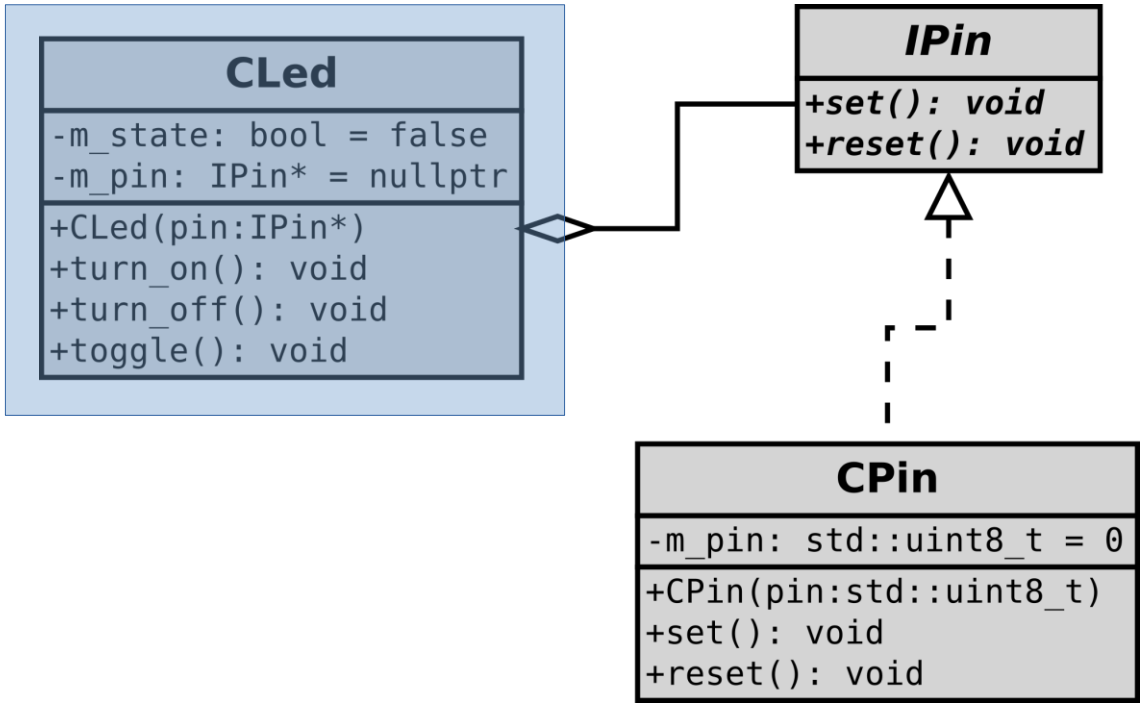
Dynamic Polymorphism

```
class IPin {
public:
    virtual void set () = 0;
    virtual void reset () = 0;
};
```



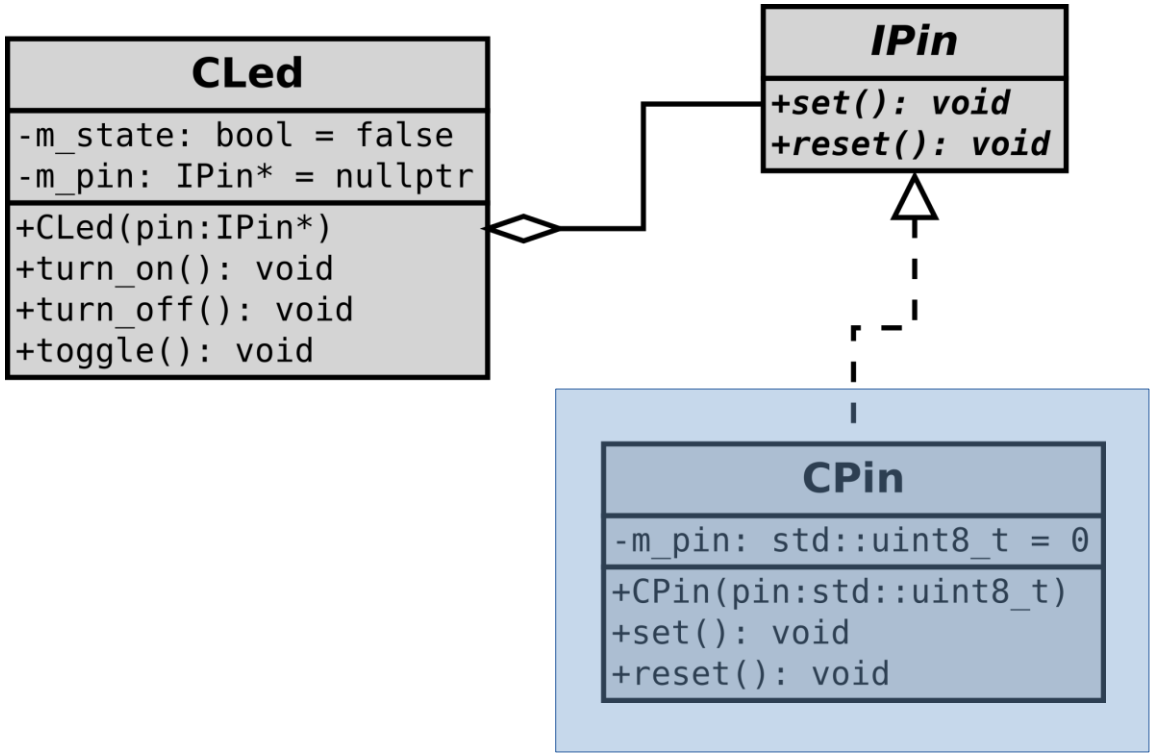
Dynamic Polymorphism

```
class CLed {
private:
    bool m_state { false };
    IPin* m_pin { nullptr };
public:
    CLed () = delete;
    CLed (IPin* pin) : m_pin(pin) {
        m_pin->reset();
    }
    /* ... */
};
```

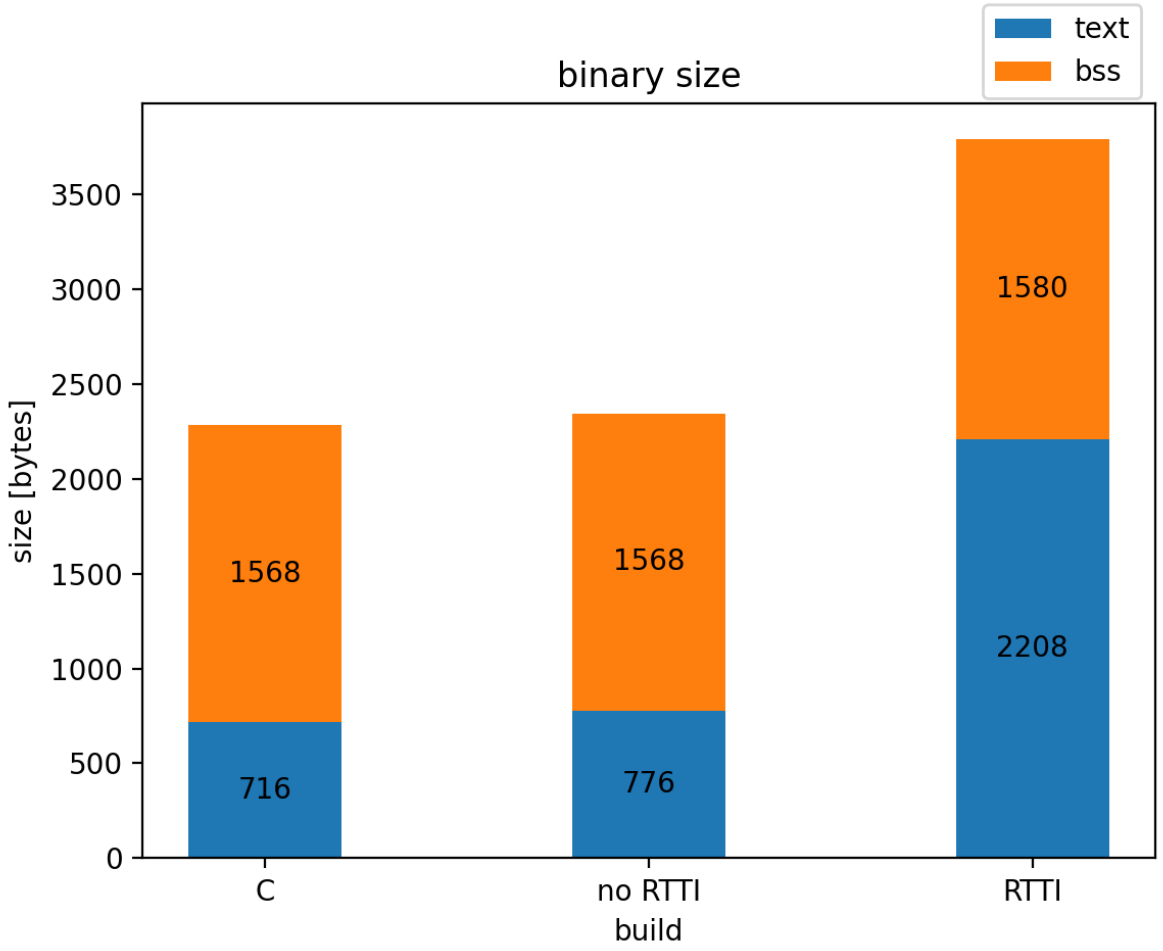


Dynamic Polymorphism

```
class CPin : public IPin {
private:
    std::uint8_t m_pin { 0 };
public:
    CPin () = delete;
    CPin (std::uint8_t pin) : m_pin(pin) {}
    void set () override {
        CBitSetResetRegister::set_pin(m_pin);
    }
    void reset () override {
        CBitSetResetRegister::reset_pin(m_pin);
    }
};
```



Dynamic Polymorphism

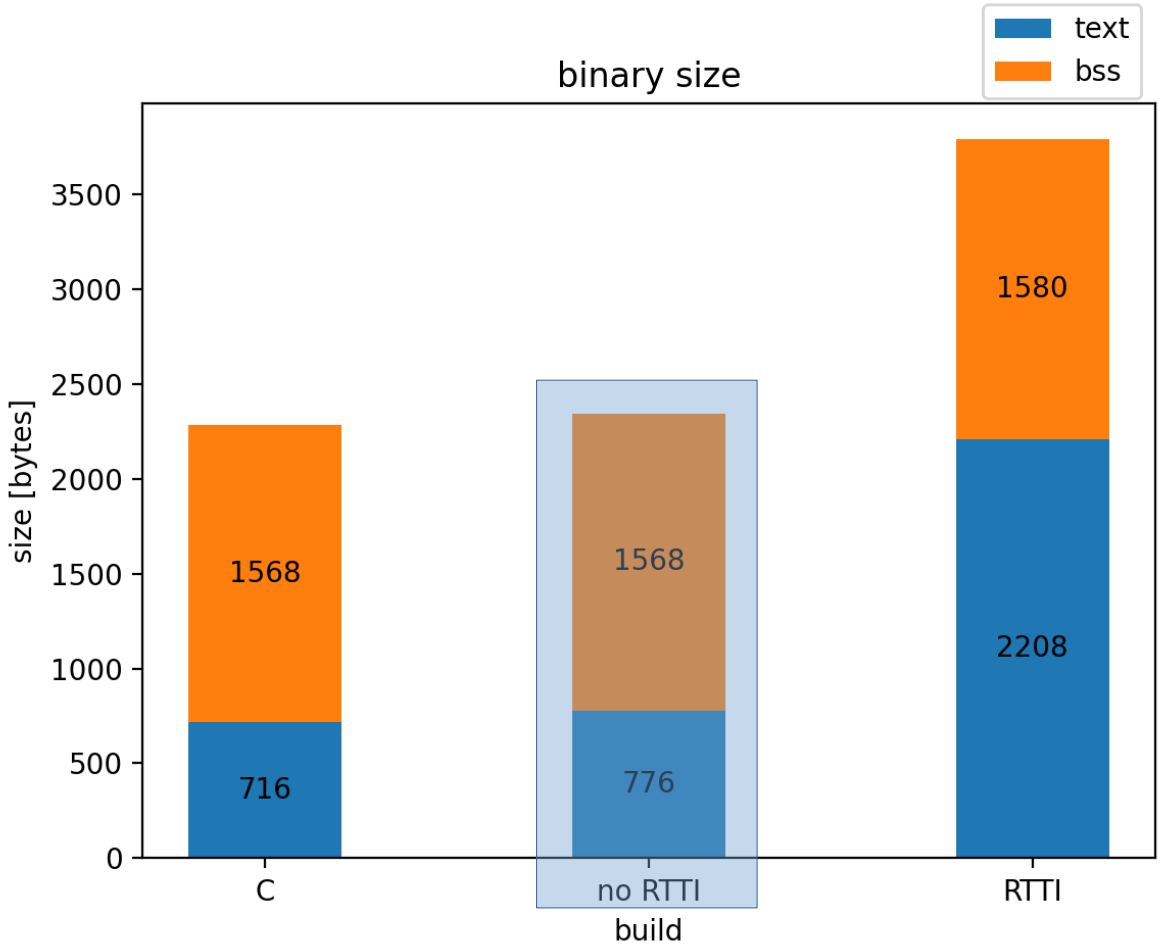


Run-**T**ime **T**ype **I**nformation

- _type_info sections in the binary
- Necessary for:
  - typeid
  - dynamic_cast

-fno-rtti

Dynamic Polymorphism

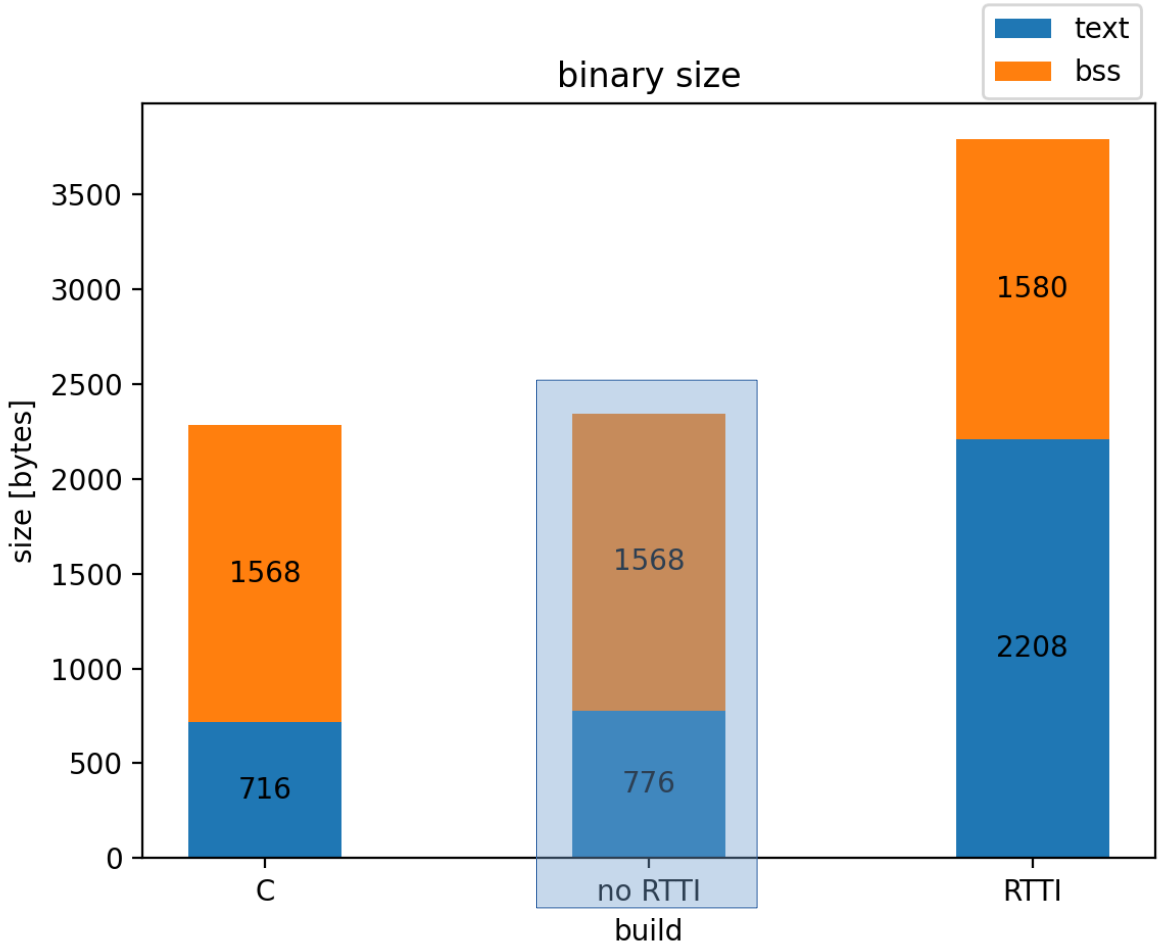


`set` and `reset` not inlined

vtable?

virtual function call?

Dynamic Polymorphism




```
CPin pin { GPIO_PIN_6 };
CLed led { &pin };

while (true) {
    led.toggle();
    delay(1000);
}
```

For statically allocated objects, the compiler can often determine the type at compile-time.
-> no virtual function calls (de-virtualization)

Dynamic Polymorphism

without de-virtualization

```
08000198 <_ZN4CPin5resetEv>: ...

080001b0 <_ZN4CPin3setEv>: ...
```

CPin vtable

```
0x08000258 : 0x080001b1
0x0800025c : 0x08000199
```

```
<main>:
push    {r0, r1, r2, lr}
ldr     r3, [pc, #28]    ; (80001e4 <main+0x20>)
mov     r2, sp
str     r3, [sp, #0]
movs    r3, #6
strb    r3, [r2, #4]
ldr     r3, [sp, #0]
mov     r0, sp
ldr     r3, [r3, #0]
blx     r3
ldr     r3, [sp, #0]
mov     r0, sp
ldr     r3, [r3, #4]
blx     r3
b.n     80001d0 <main+0xc>
Nop     ; (mov r8, r8)
.word   0x08000258
```

Dynamic Polymorphism

without de-virtualization

```
08000198 <_ZN4CPin5resetEv>: ...

080001b0 <_ZN4CPin3setEv>: ...
```

CPin vtable

```
0x08000258 : 0x080001b1
0x0800025c : 0x08000199
```

```
<main>:
push    {r0, r1, r2, lr}
ldr     r3, [pc, #28]    ; (80001e4 <main+0x20>)
mov     r2, sp
str     r3, [sp, #0]
movs    r3, #6
strb    r3, [r2, #4]
ldr     r3, [sp, #0]
mov     r0, sp
ldr     r3, [r3, #0]
blx     r3
ldr     r3, [sp, #0]
mov     r0, sp
ldr     r3, [r3, #4]
blx     r3
b.n     80001d0 <main+0xc>
Nop     ; (mov r8, r8)
.word   0x08000258
```

Load the vtable address in r3.

Dynamic Polymorphism

without de-virtualization

```
08000198 <_ZN4CPin5resetEv>: ...

080001b0 <_ZN4CPin3setEv>: ...
```

CPin vtable

```
0x08000258 : 0x080001b1
0x0800025c : 0x08000199
```

```
<main>:
push    {r0, r1, r2, lr}
ldr     r3, [pc, #28]    ; (80001e4 <main+0x20>)
mov     r2, sp
str     r3, [sp, #0]
movs    r3, #6
strb    r3, [r2, #4]
ldr     r3, [sp, #0]
mov     r0, sp
ldr     r3, [r3, #0]
blx     r3
ldr     r3, [sp, #0]
mov     r0, sp
ldr     r3, [r3, #4]
blx     r3
b.n     80001d0 <main+0xc>
Nop     ; (mov r8, r8)
.word   0x08000258
```

Load the address of the set method from the vtable into r3.

Dynamic Polymorphism

without de-virtualization

```
08000198 <_ZN4CPin5resetEv>: ...

080001b0 <_ZN4CPin3setEv>: ...
```

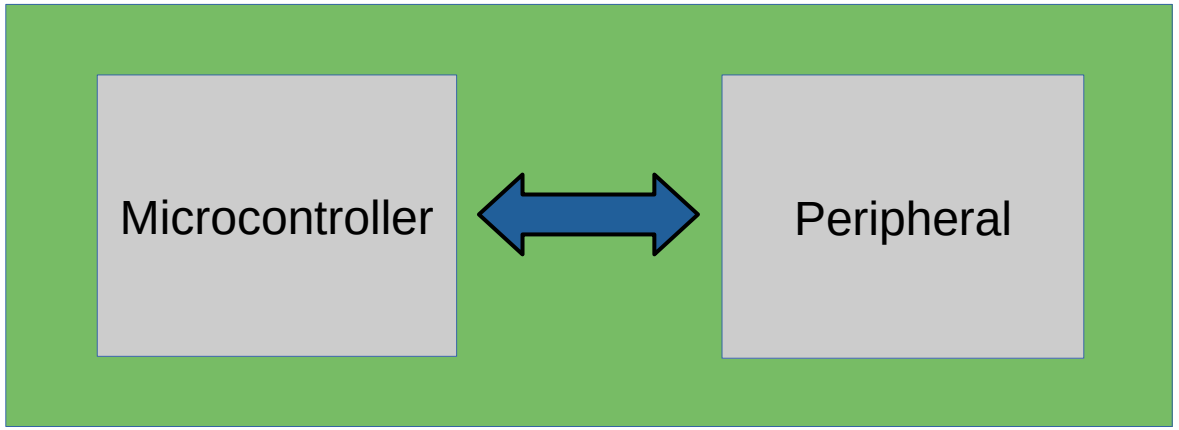
CPin vtable

```
0x08000258 : 0x080001b1
0x0800025c : 0x08000199
```

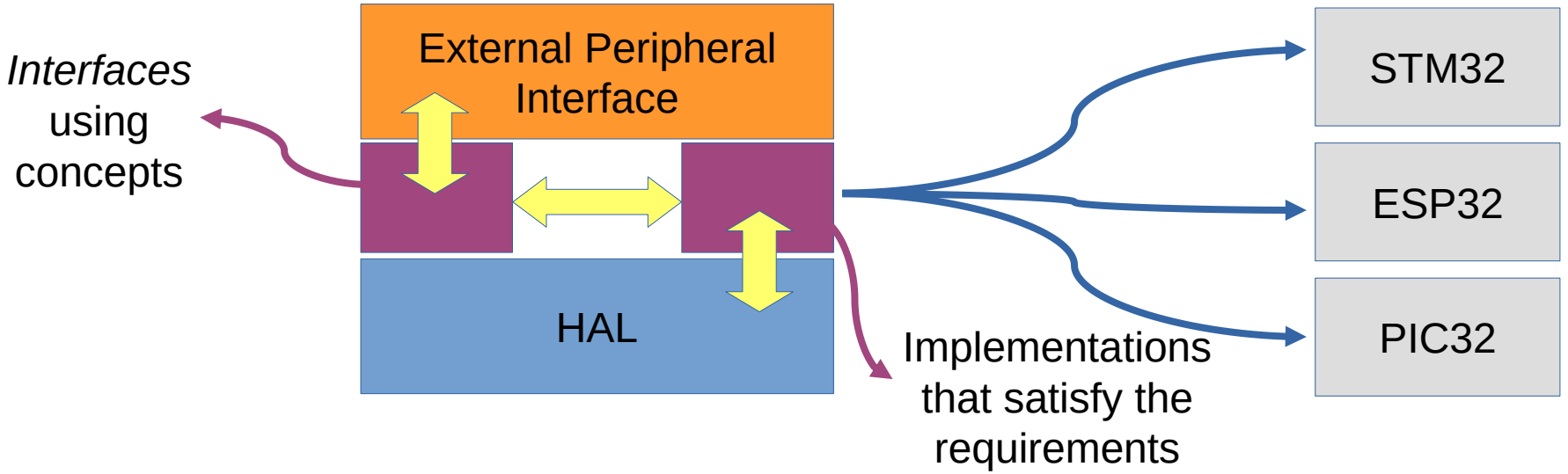
Call the set method via the address in r3.

```
<main>:
push    {r0, r1, r2, lr}
ldr     r3, [pc, #28]    ; (80001e4 <main+0x20>)
mov     r2, sp
str     r3, [sp, #0]
movs    r3, #6
strb    r3, [r2, #4]
ldr     r3, [sp, #0]
mov     r0, sp
ldr     r3, [r3, #0]
blx     r3
ldr     r3, [sp, #0]
mov     r0, sp
ldr     r3, [r3, #4]
blx     r3
b.n     80001d0 <main+0xc>
Nop     ; (mov r8, r8)
.word   0x08000258
```

Moving Higher

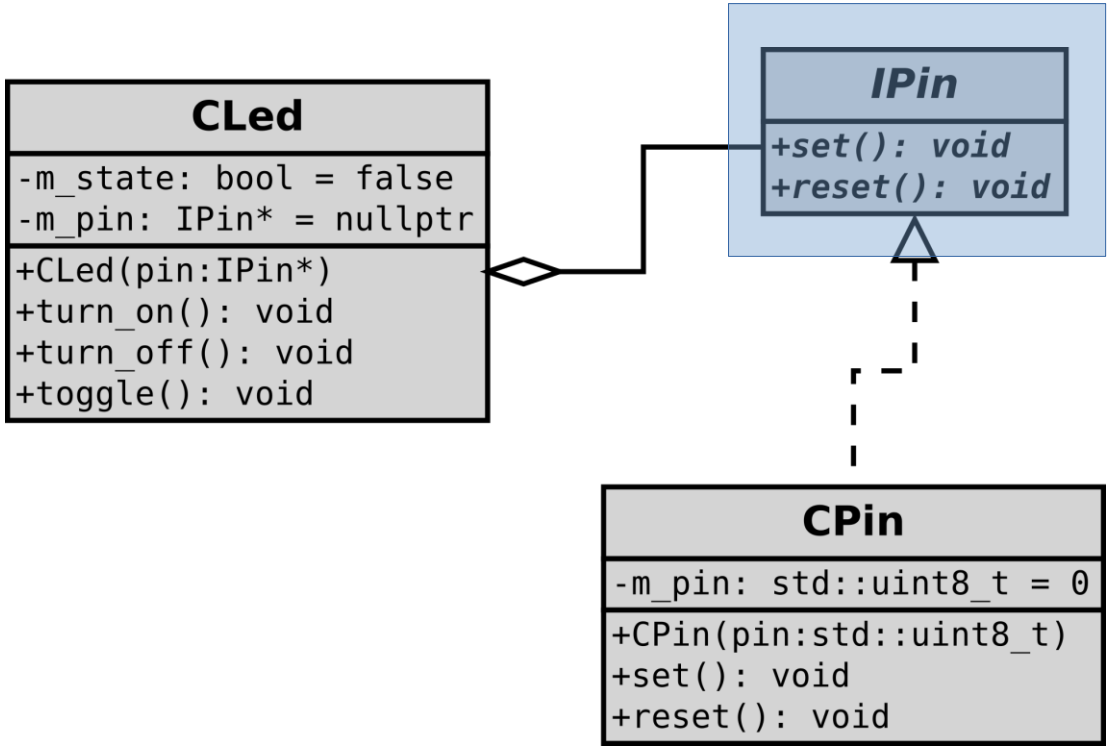


Static Polymorphism



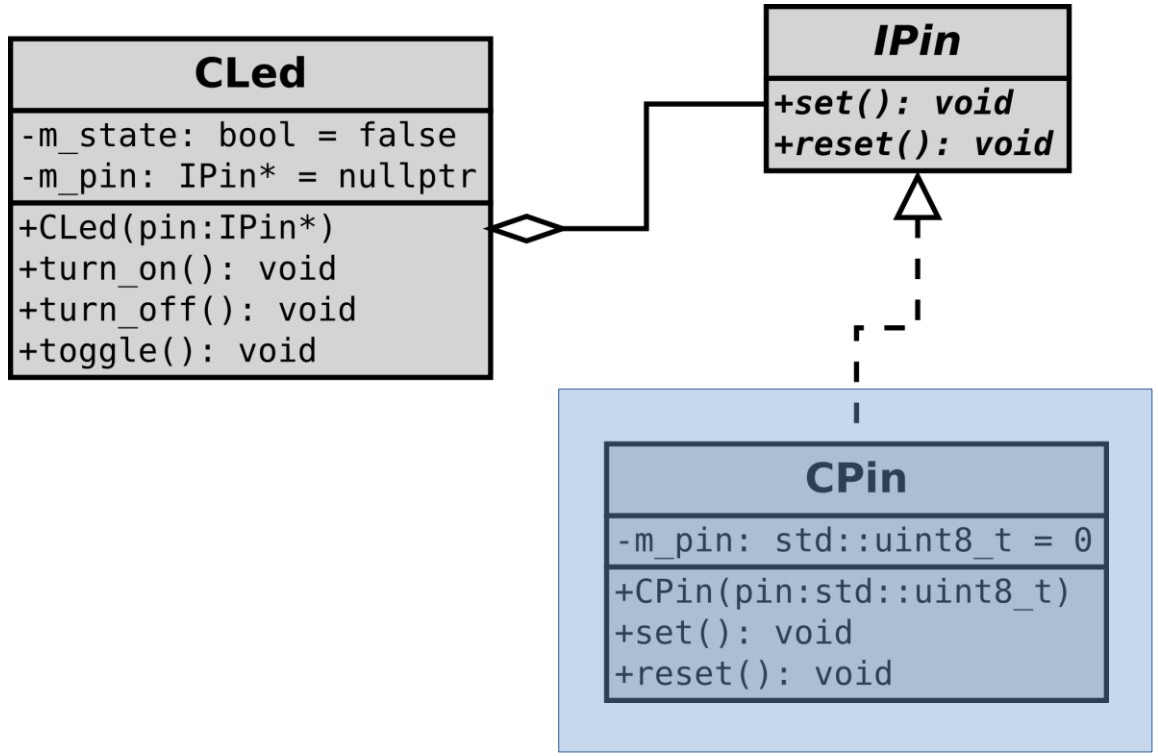
Static Polymorphism

```
template <typename T>
concept IPin = requires (T pin) {
    { pin.set() } -> std::same_as<void>;
    { pin.reset() } -> std::same_as<void>;
};
```



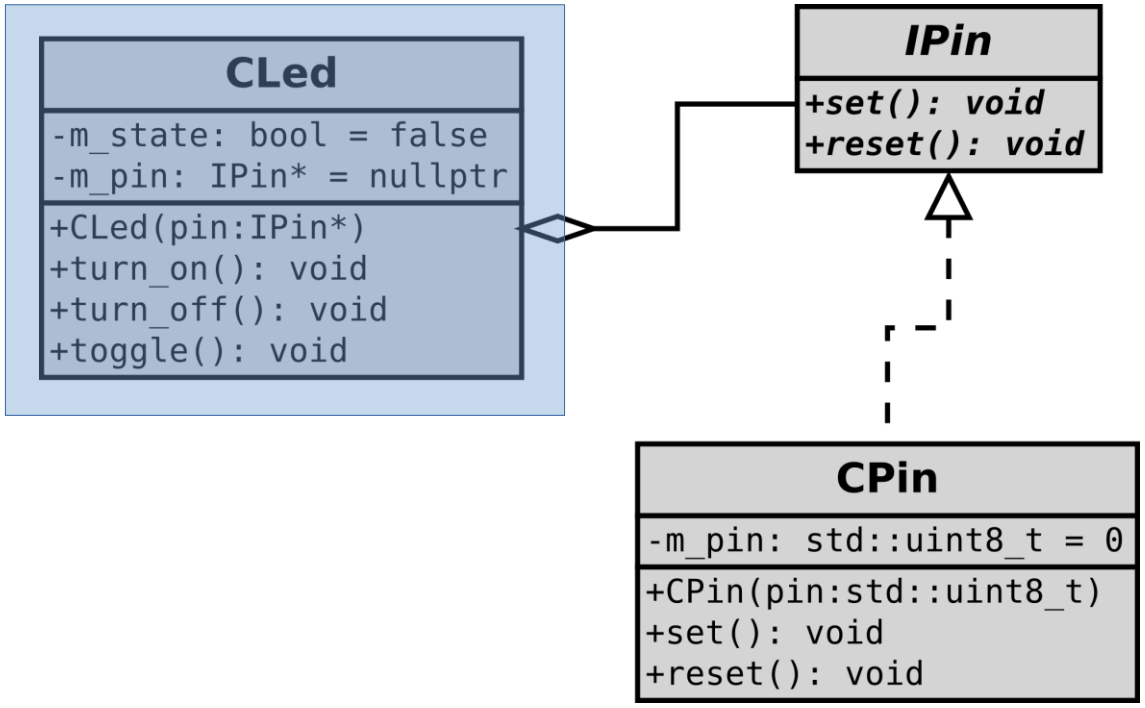
Static Polymorphism

```
class CPin {
private:
    std::uint8_t m_pin { 0 };
public:
    CPin () = delete;
    CPin (std::uint8_t pin) : m_pin(pin) {}
    void set () const {
        CBitSetResetRegister::set_pin(m_pin);
    }
    void reset () const {
        CBitSetResetRegister::reset_pin(m_pin);
    }
};
static_assert(IPin<CPin>); // optional
```



Static Polymorphism

```
template <IPin TPin>
class CLed {
private:
    bool m_state { false };
    TPin* m_pin { nullptr };
public:
    CLed () = delete;
    CLed (TPin* pin) : m_pin(pin) {
        m_pin->reset();
    }
    /* ... */
};
```



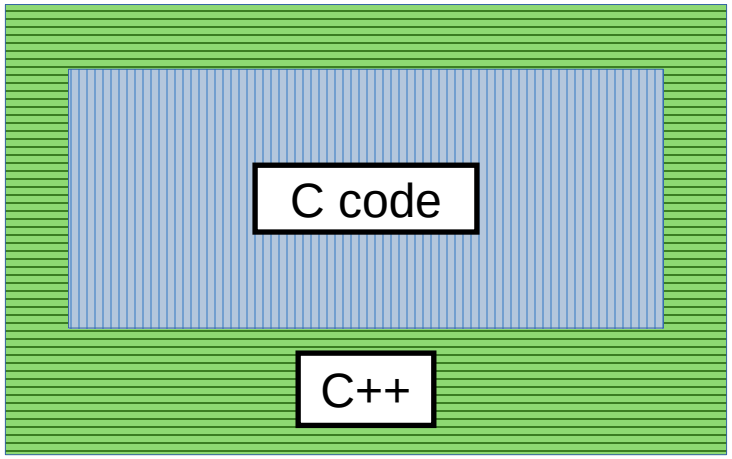
Static Polymorphism

Compiles to the same binary.

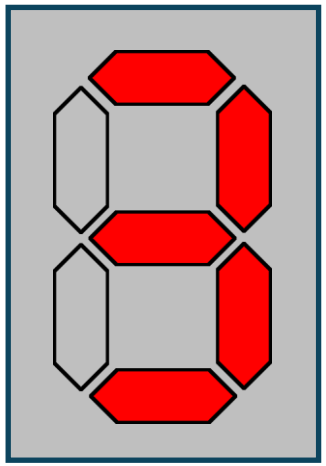
Architecture Matters

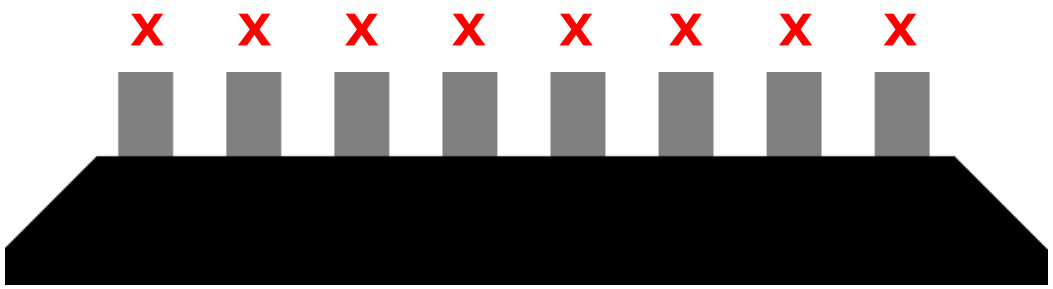
Building Layers of Abstractions

Wrapper

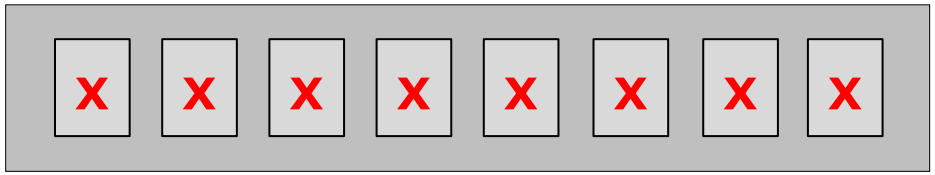


Architecture Matters

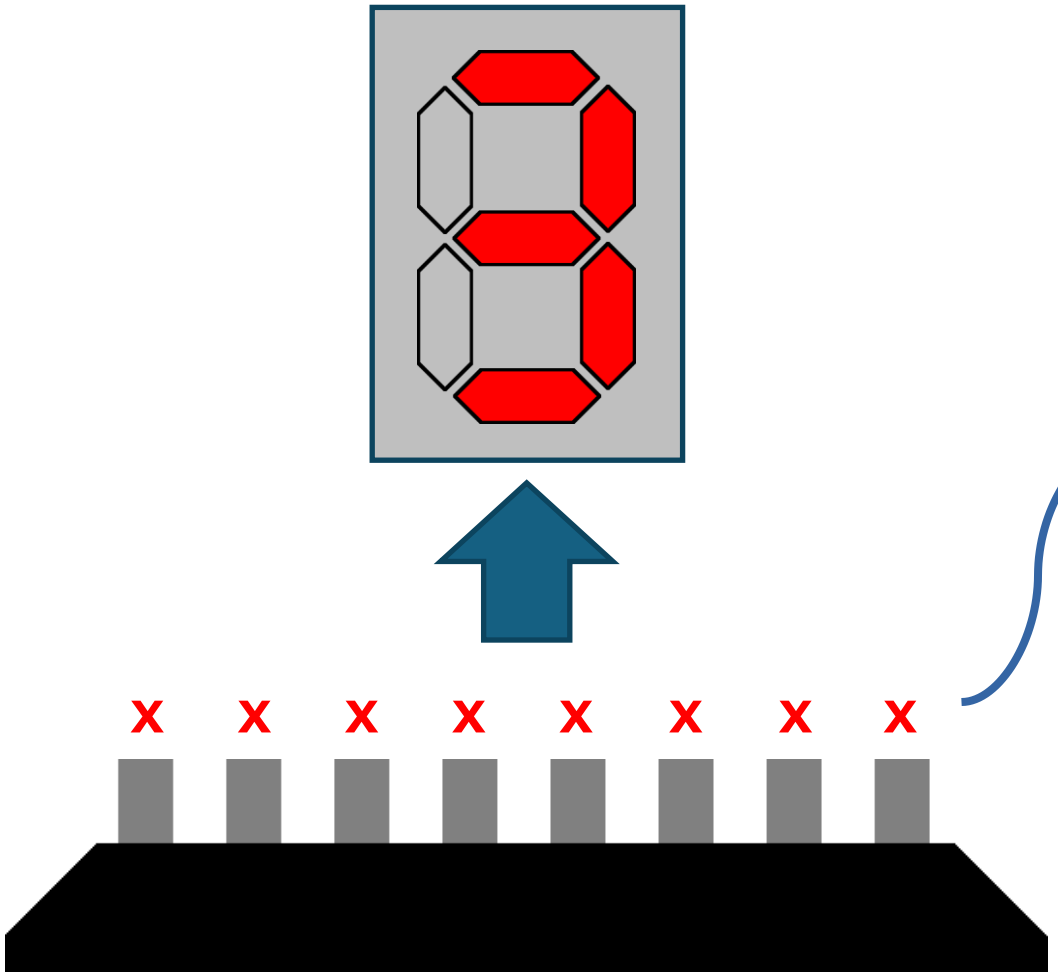




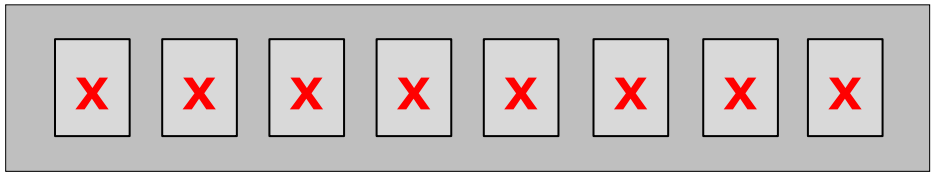
Output Data Register



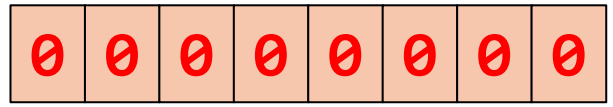
Architecture Matters



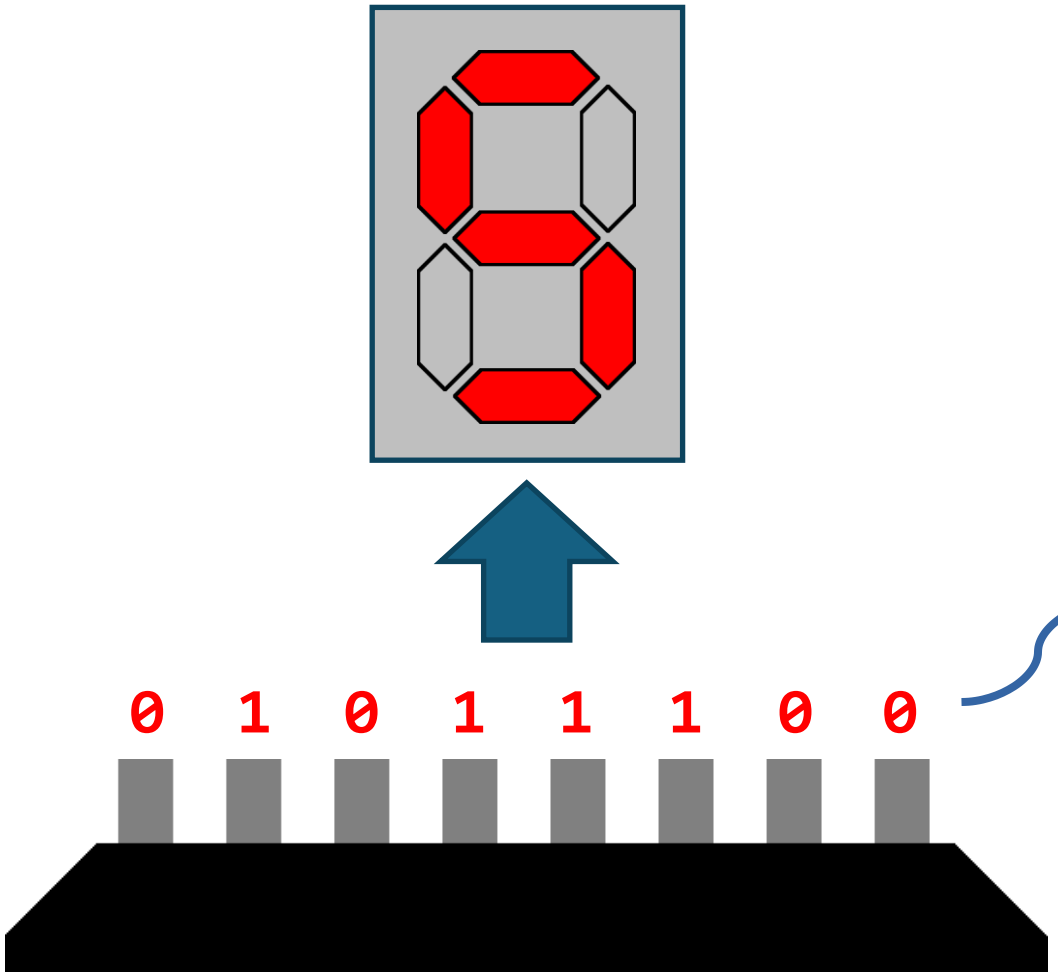
Output Data Register



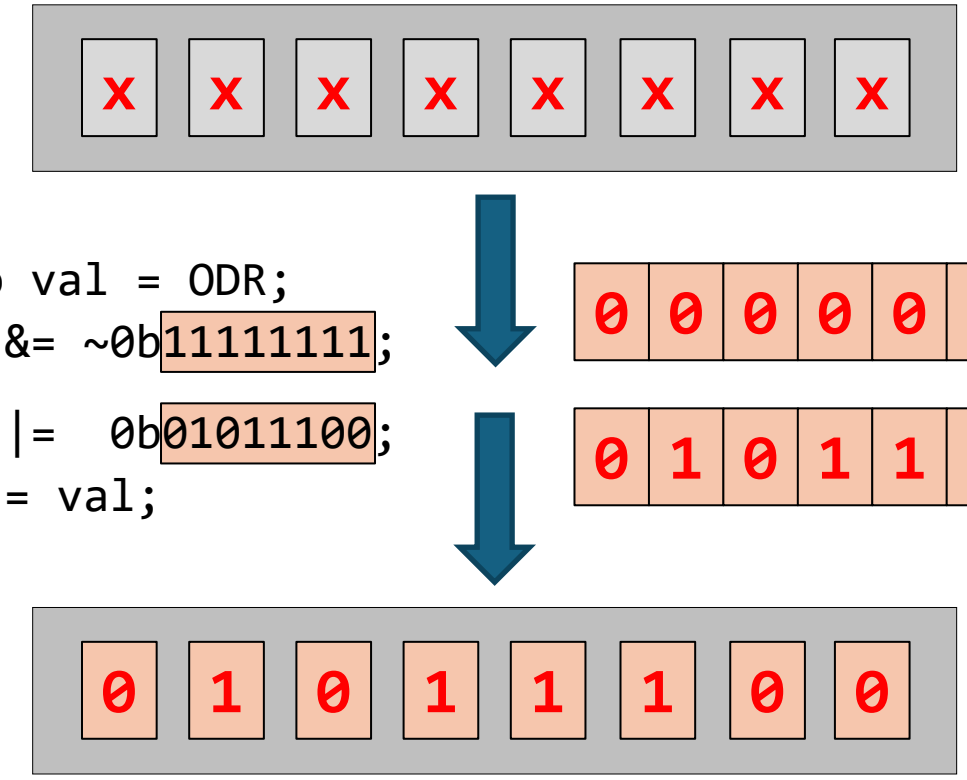

```
auto val = ODR;
val &= ~0b11111111;
```



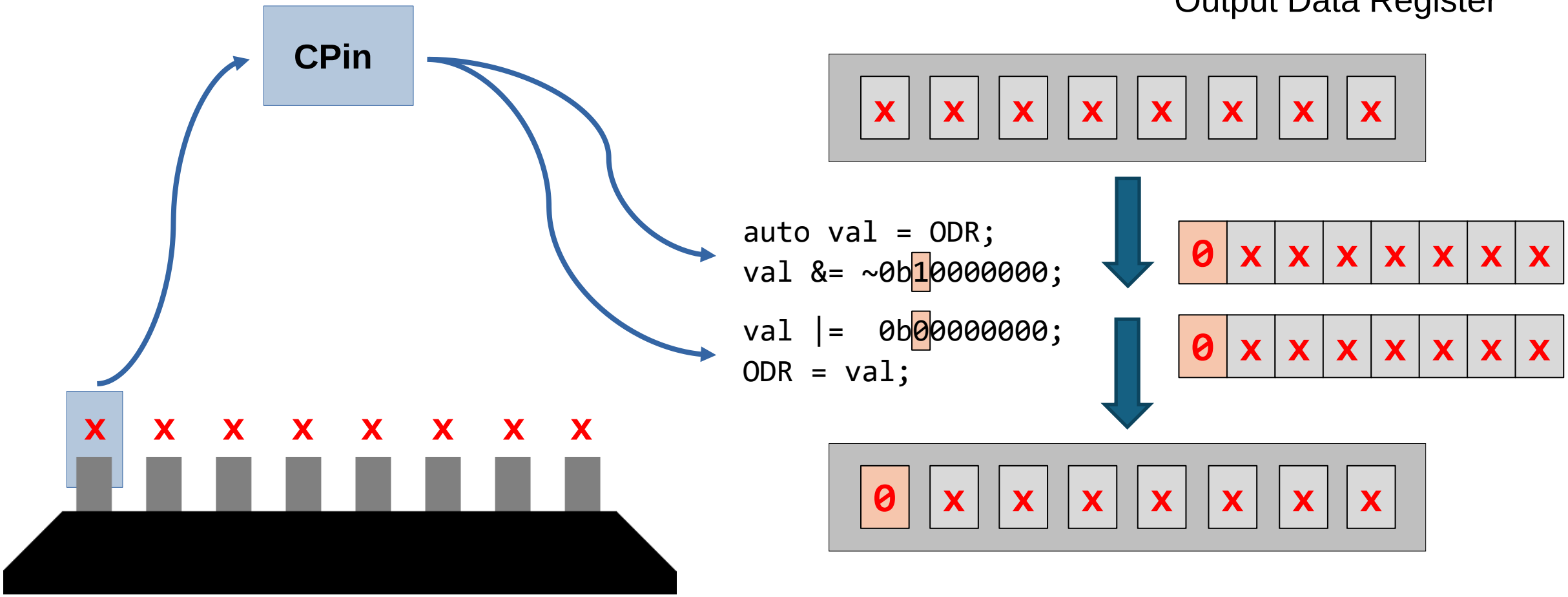
Architecture Matters



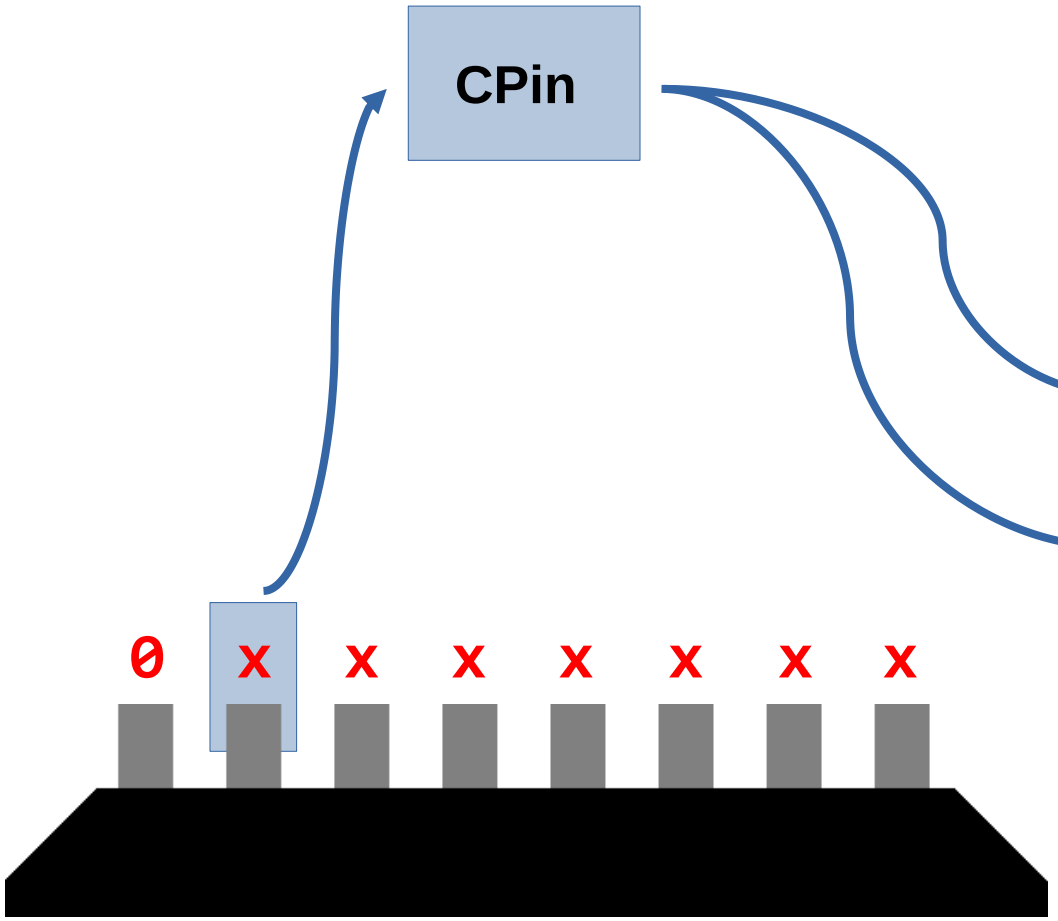
Output Data Register



Architecture Matters

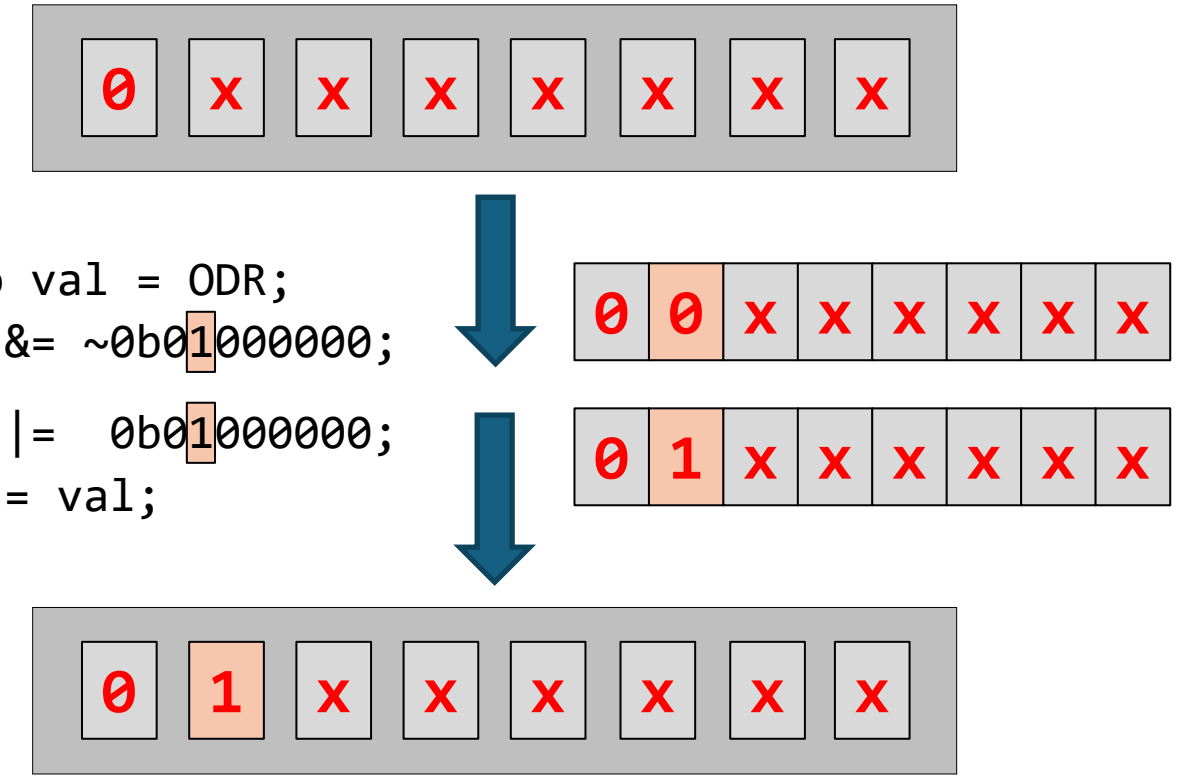


Architecture Matters

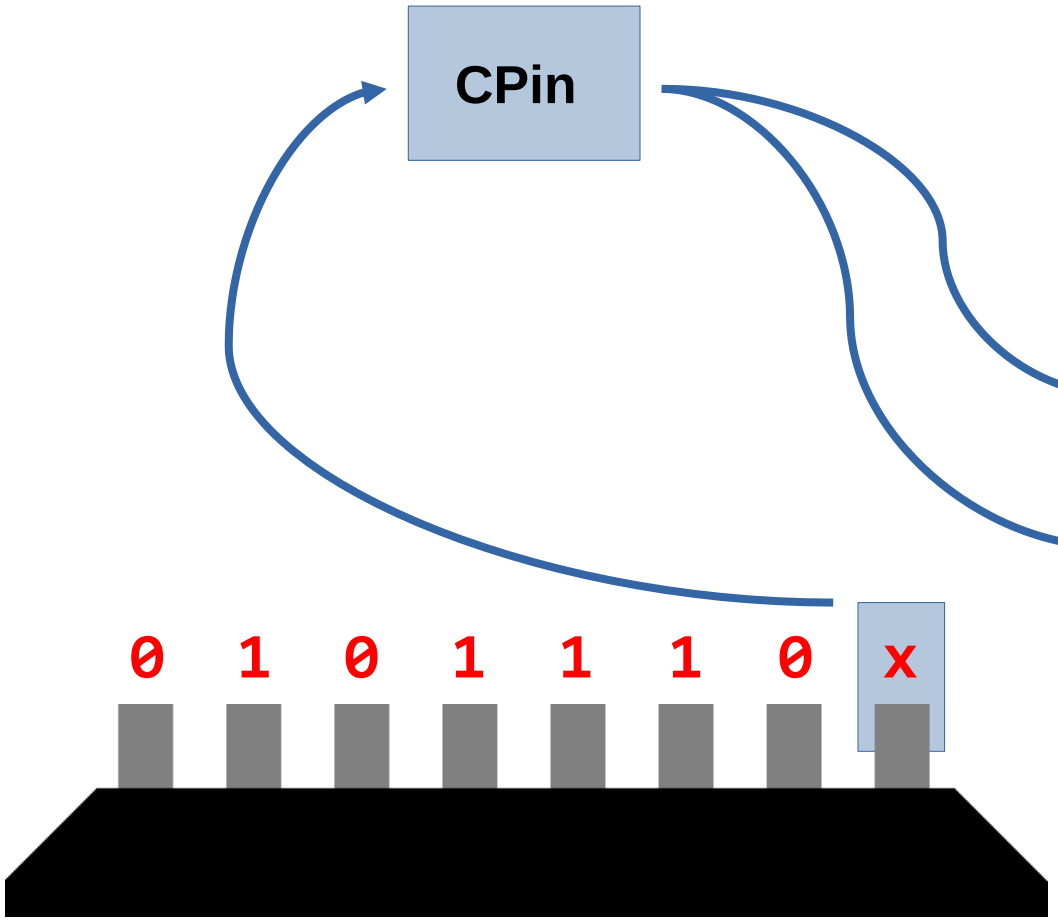



```
auto val = ODR;
val &= ~0b01000000;

val |=  0b01000000;
ODR = val;
```

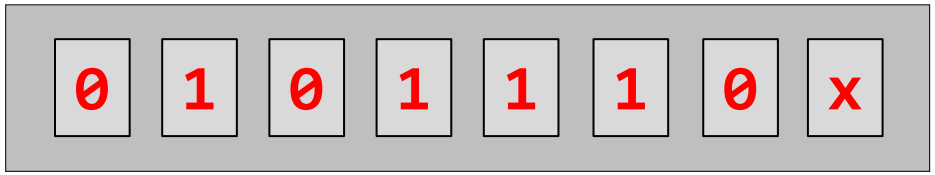
Output Data Register



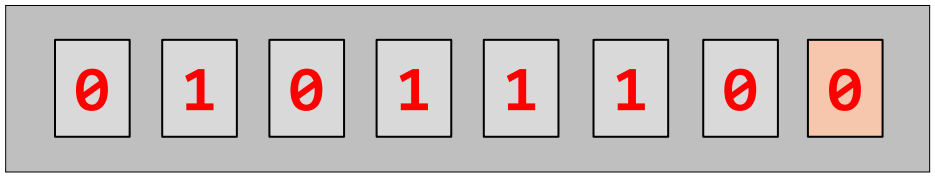
Architecture Matters



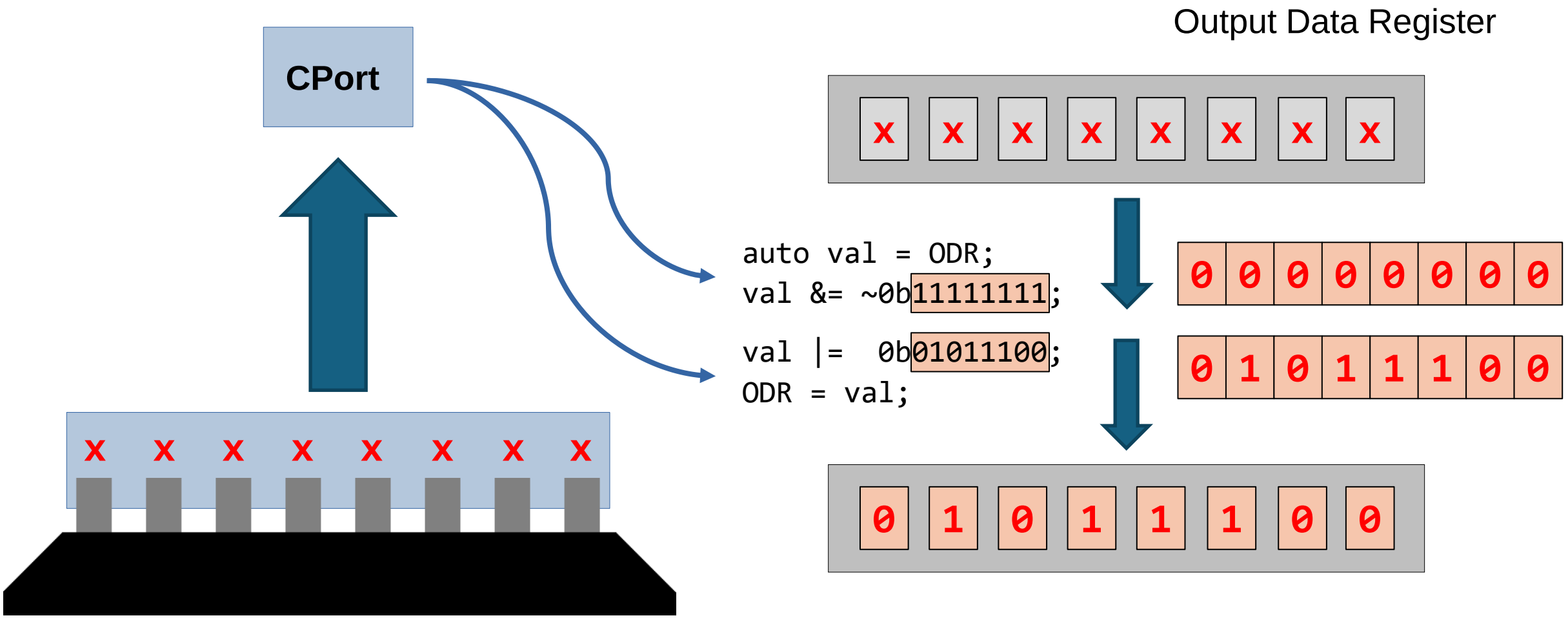
Output Data Register




```
auto val = ODR;
val &= ~0b00000001;
val |=  0b00000000;
ODR = val;
```



Architecture Matters



Dealing with Inefficiencies

Inefficiencies in HAL

```c
typedef struct {
    uint32_t pin;
    GPIO_Modes mode;
} GPIO_InitStruct;

void GPIO_Init(GPIO_InitStruct* conf) {
    uint32_t temp;

    /* check the values */
    if (!IS_GPIO_PIN(conf->pin)) { return; }
    if (!IS_GPIO_MODE(conf->mode)) { return; }

    /* configure the GPIO based on the settings */
    if (conf->mode == GPIO_MODE_OUTPUT) {
        temp = OSPEEDR;
        temp &= ~(OSPEEDR_MASK << (conf->pin * 2u));
        temp |= (GPIO_SPEED_FREQ_LOW << (conf->pin * 2u));
        OSPEEDR = temp;
        /* ... */
    }
    /* ... */
}
```

Enums are basically integers with no value restrictions

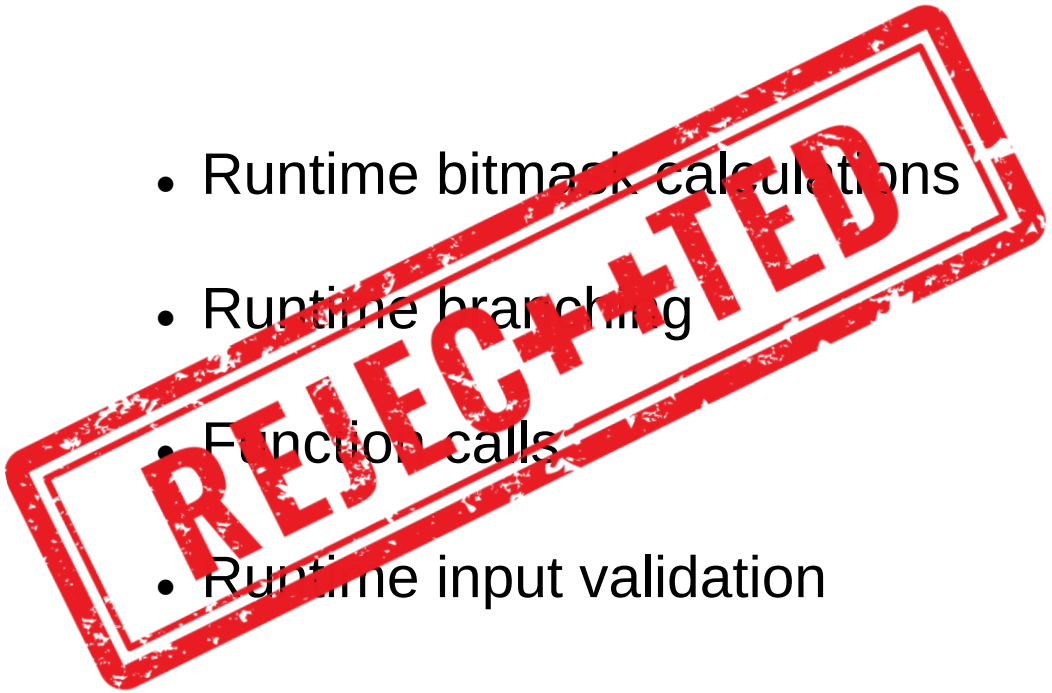
Runtime branching based on input parameters

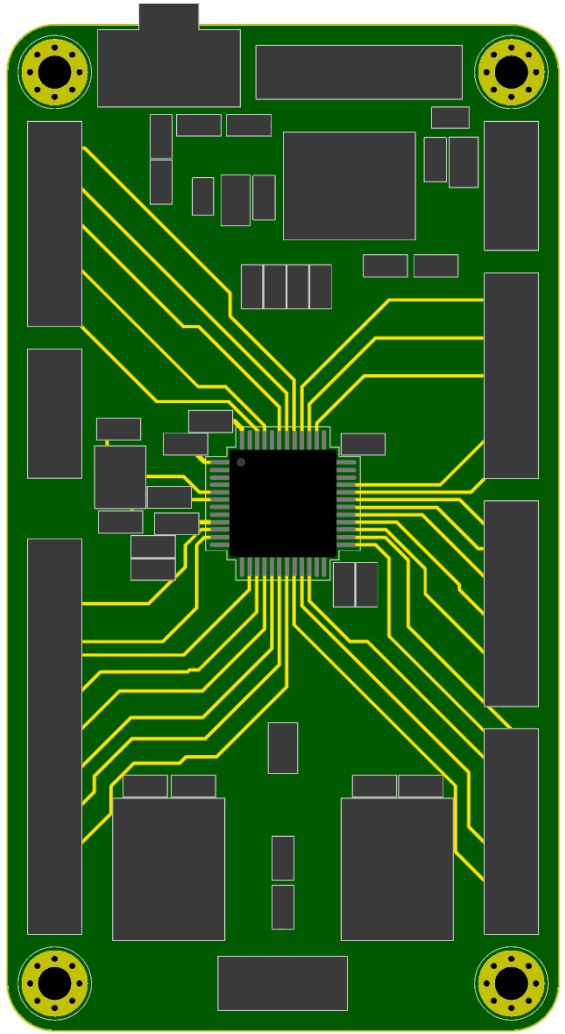
Runtime bitmask calculations

Function calls

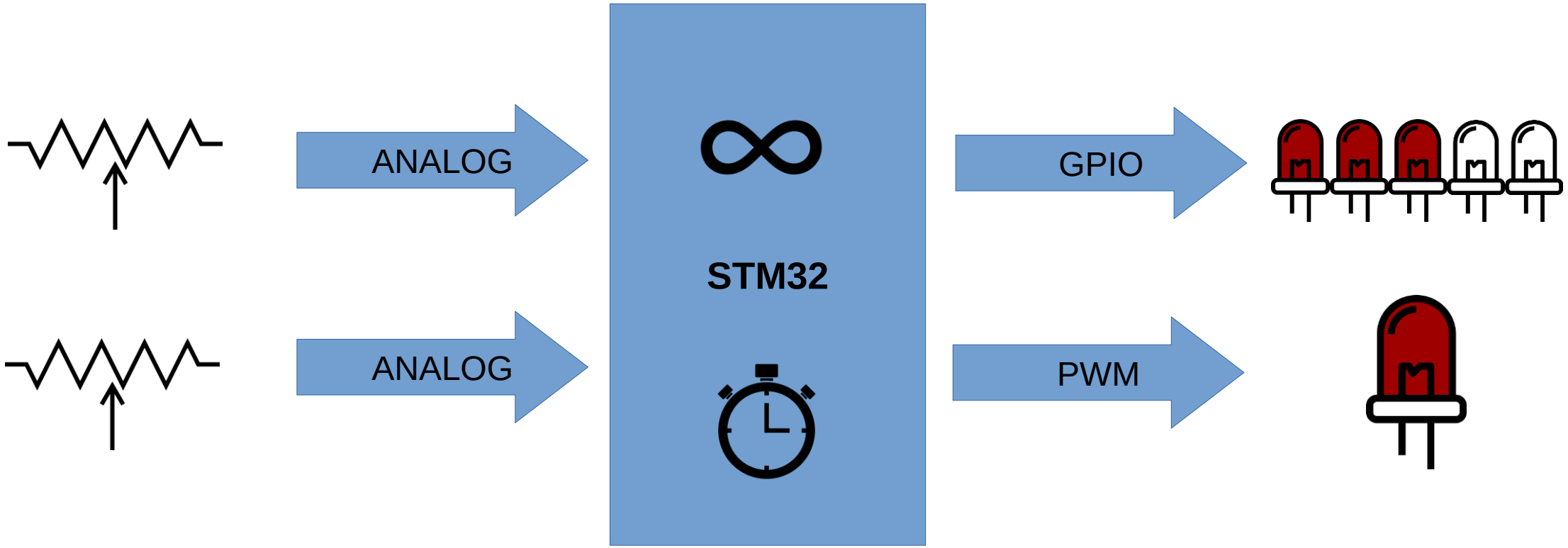
Inefficiencies in HAL

- Runtime bitmask calculations
- Runtime branching
- Function calls
- Runtime input validation



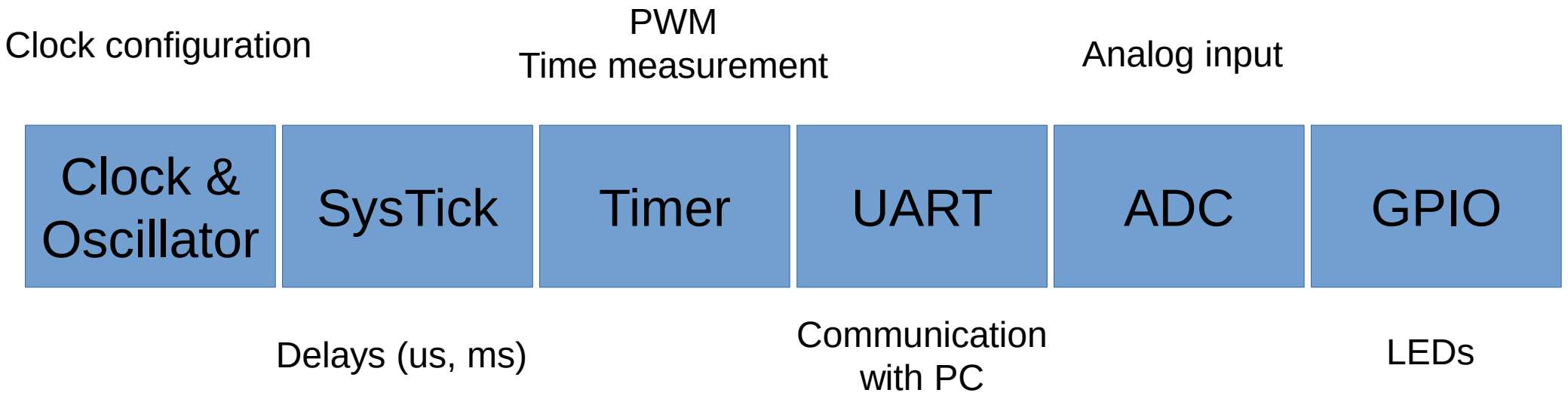


Setup



Modules

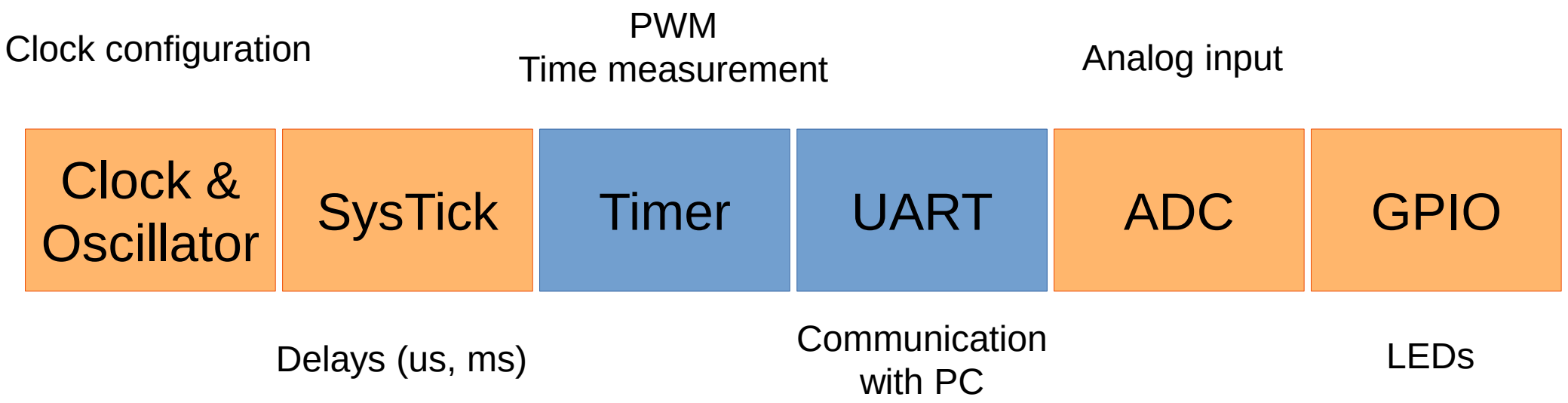
C


Clock configuration
PWM
Time measurement
Analog input
Clock & Oscillator
SysTick
Timer
UART
ADC
GPIO
Delays (us, ms)
Communication with PC
LEDs

Modules

C

C++


Clock configuration
PWM
Time measurement
Analog input
Clock & Oscillator
SysTick
Timer
UART
ADC
GPIO
Delays (us, ms)
Communication with PC
LEDs

Enumerations

```
enum class modes : std::uint32_t {
    input = 0b00,
    output = 0b01,
    alt_func = 0b10,
    analog = 0b11
};
```

```
enum class ports : std::uint8_t {
    port_f,
    port_d,
    port_c,
    port_b,
    port_a
};
```

Enumerations

- Not plain integers anymore
- No implicit conversion between enum values and integral types
  - Explicit static_cast required
  - Harder to misuse accidentally

Templates & Concepts

```
template <modes mode>
concept is_valid_mode = (
    (mode == modes::input) ||
    (mode == modes::output) ||
    (mode == modes::alt_func) ||
    (mode == modes::analog)
);
```

```
template <pins pin>
concept is_valid_pin = (
    is_valid_low_pin<pin> || is_valid_high_pin<pin>
);
```

```
template <pins... pin>
concept are_valid_pins = (is_valid_pin<pin> && ...);
```

Template parameters

- Compile time parameter passing

Concepts

- Set of requirements on template arguments
- Compile time parameter validation

Immediate Functions

- Executed at compile time
- Useful for e.g., bitmask calculations

```
template <modes mode, pins... pin>
requires (are_valid_pins<pin ...> && is_valid_mode<mode>)
consteval std::uint32_t moder_value () {
    return (... | (static_cast<std::uint32_t>(mode) <<
                   (static_cast<std::uint32_t>(pin) * 2)));
}

template <pins... pin>
requires (are_valid_pins<pin ...>)
consteval std::uint32_t moder_bitmask () {
    return (... | (GPIO_MODER_MODER0 <<
                   (static_cast<std::uint32_t>(pin) * 2)));
}
```

Compile-time Branching

```
template <GpioInitConfig conf, pins... pin>
requires (is_valid_gpio_config<conf> && are_valid_pins<pin...>)
static inline void configure_pins() {
    static_assert((sizeof...(pin) > 0), "No pins provided.");
    if constexpr ((conf.mode == modes::output) || (conf.mode == modes::alt_func)) {
        /* ... */
    }
    if constexpr (conf.mode != modes::analog) {
        /* ... */
    }
    if constexpr (conf.mode == modes::alt_func) {
        /* ... */
    }
    /* ... */
}
```

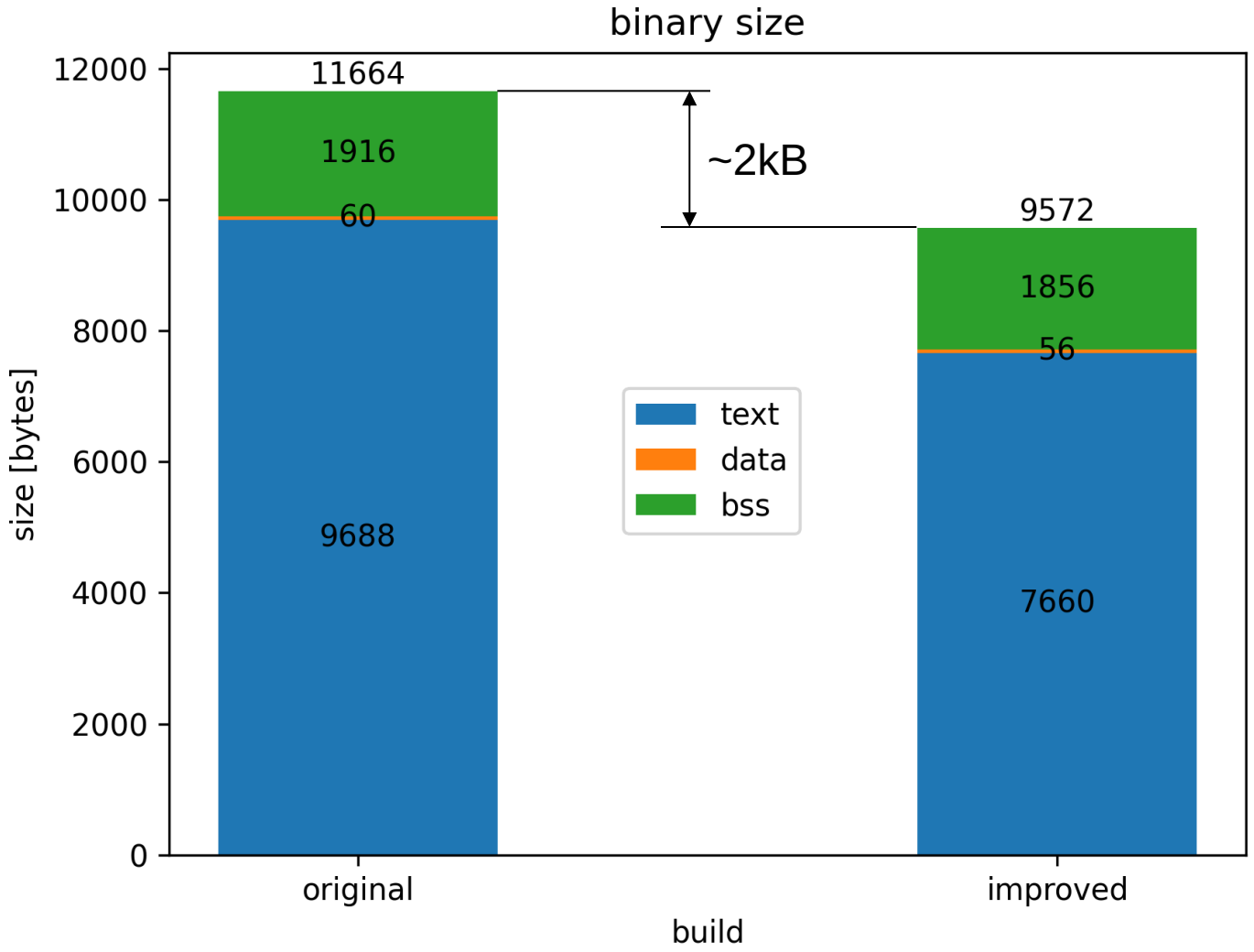
Eliminates:

- Unused code
- Comparisons
- Jump instructions

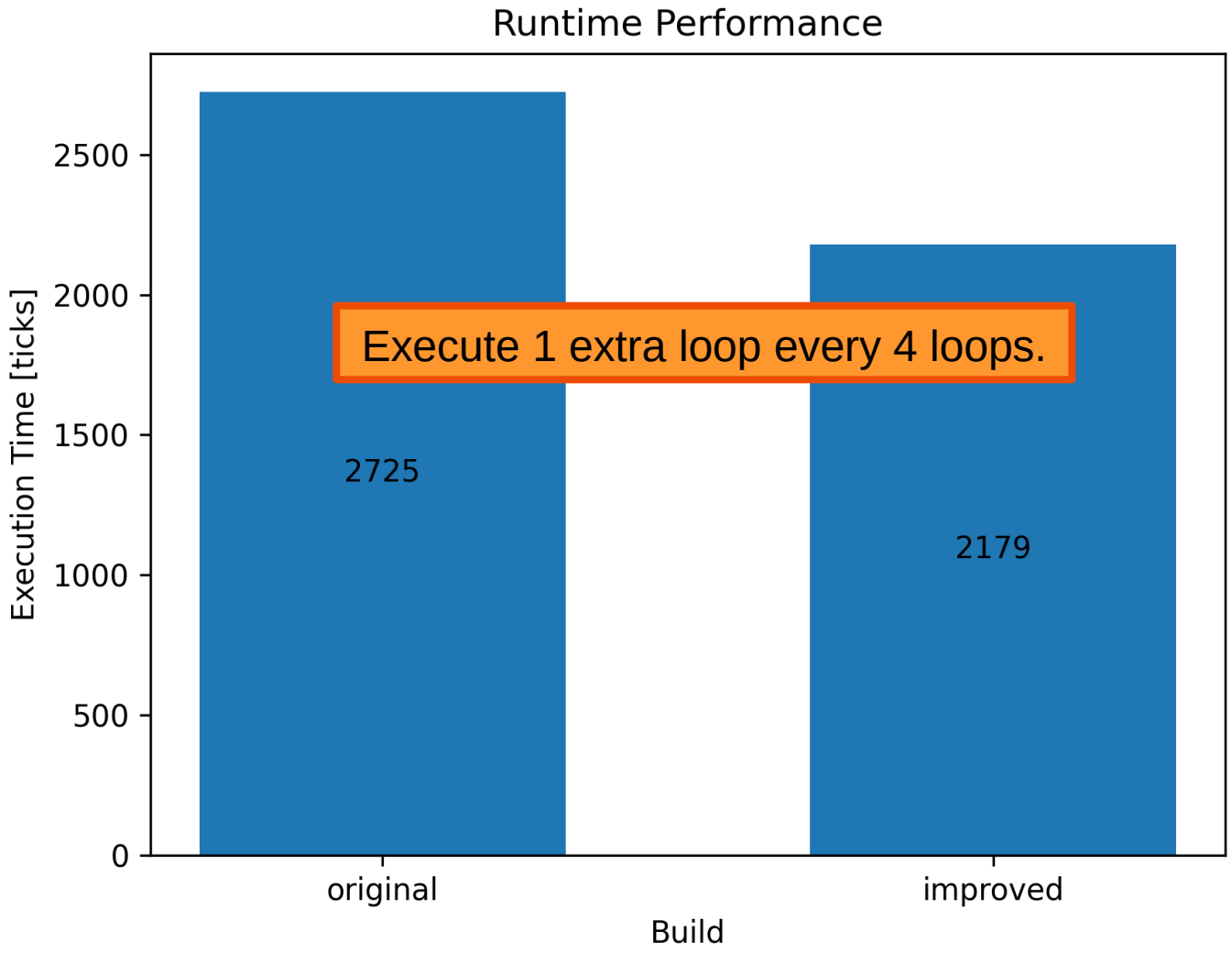
Result

- Compile time parameter validation 
- Compile time bitmask calculations 
- Compile time branching 
- Complex functions -> couple of register operations 

Result


binary size
size [bytes]
12000
10000
8000
6000
4000
2000
0
11664
1916
60
9688
9572
1856
56
7660
~2kB
text
data
bss
original
improved
build

Result


Runtime Performance
Execution Time [ticks]
2500
2000
1500
1000
500
0
Execute 1 extra loop every 4 loops.
2725
2179
original
improved
Build

Code Bloat

non-type template parameter

```
hal::rcc::CRcc::configure_oscillator<oscillator_config_1>();
hal::rcc::CRcc::configure_oscillator<oscillator_config_2>();
hal::rcc::CRcc::configure_oscillator<oscillator_config_3>();
```

```
_ZN3hal3rcc4CRcc20configure_oscillatorIX...
_ZN3hal3rcc4CRcc20configure_oscillatorIX...
_ZN3hal3rcc4CRcc20configure_oscillatorIX...
```

Binary size

Compilation time

Conclusions

Zero-cost abstractions

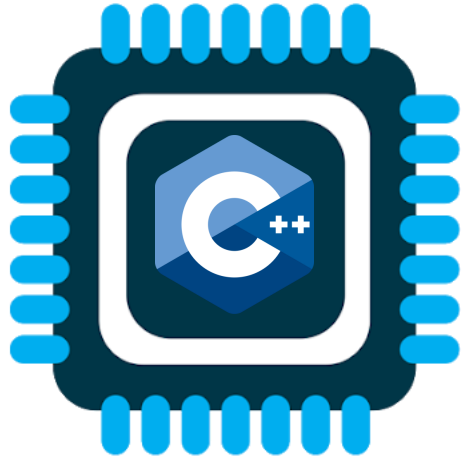
- Encapsulation
- Inheritance
- Dynamic Polymorphism
- Static Polymorphism

Architecture matters at runtime

Negative-cost abstractions

- template
- concept
- constexpr
- consteval

More C++ for the embedded world!



Thank You!

Marcell Juhasz

marcelljuhasz.com
github.com/juhaszmarcell96
linkedin.com/in/juhaszmarcell
marcell.juhasz96@gmail.com