The fastest JVM is the C++26 compiler

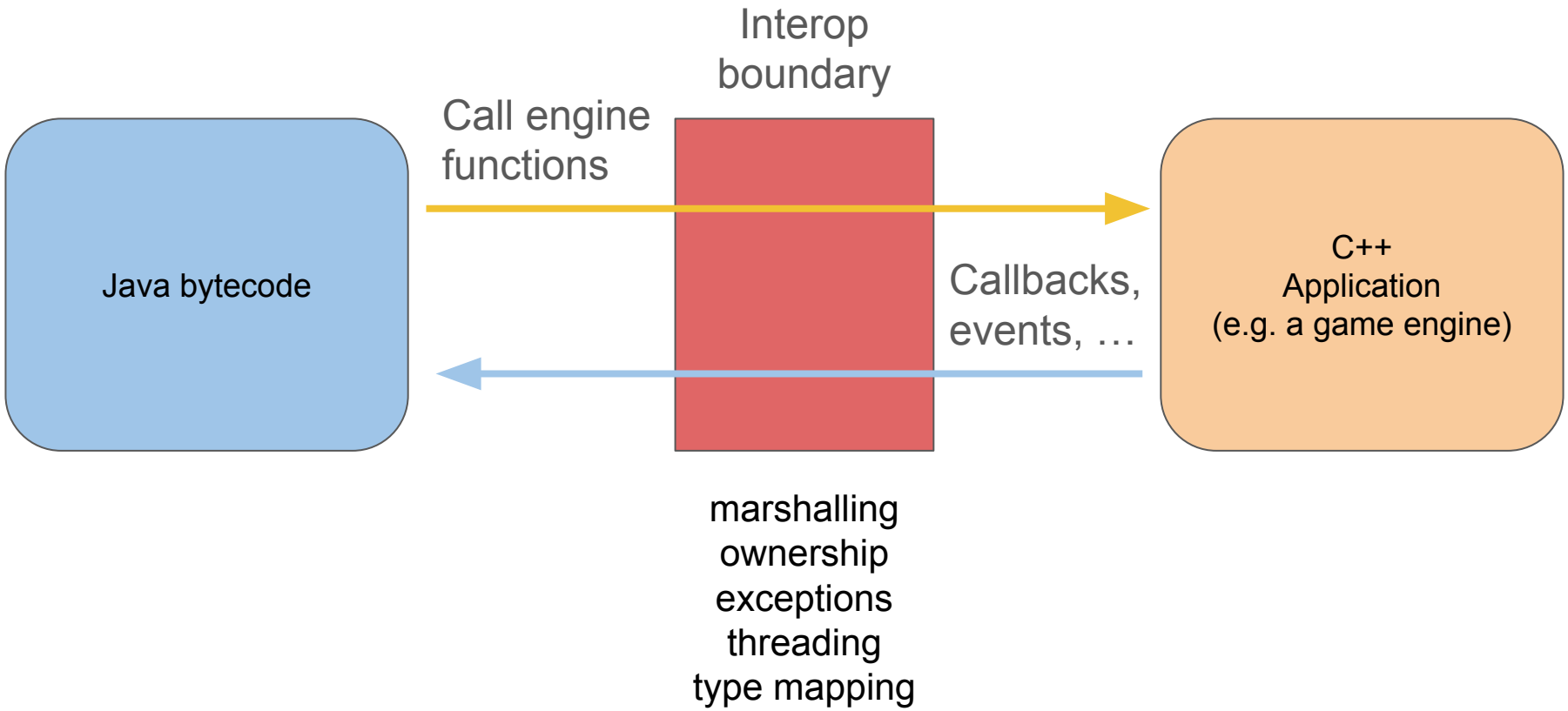
A static reflection adventure

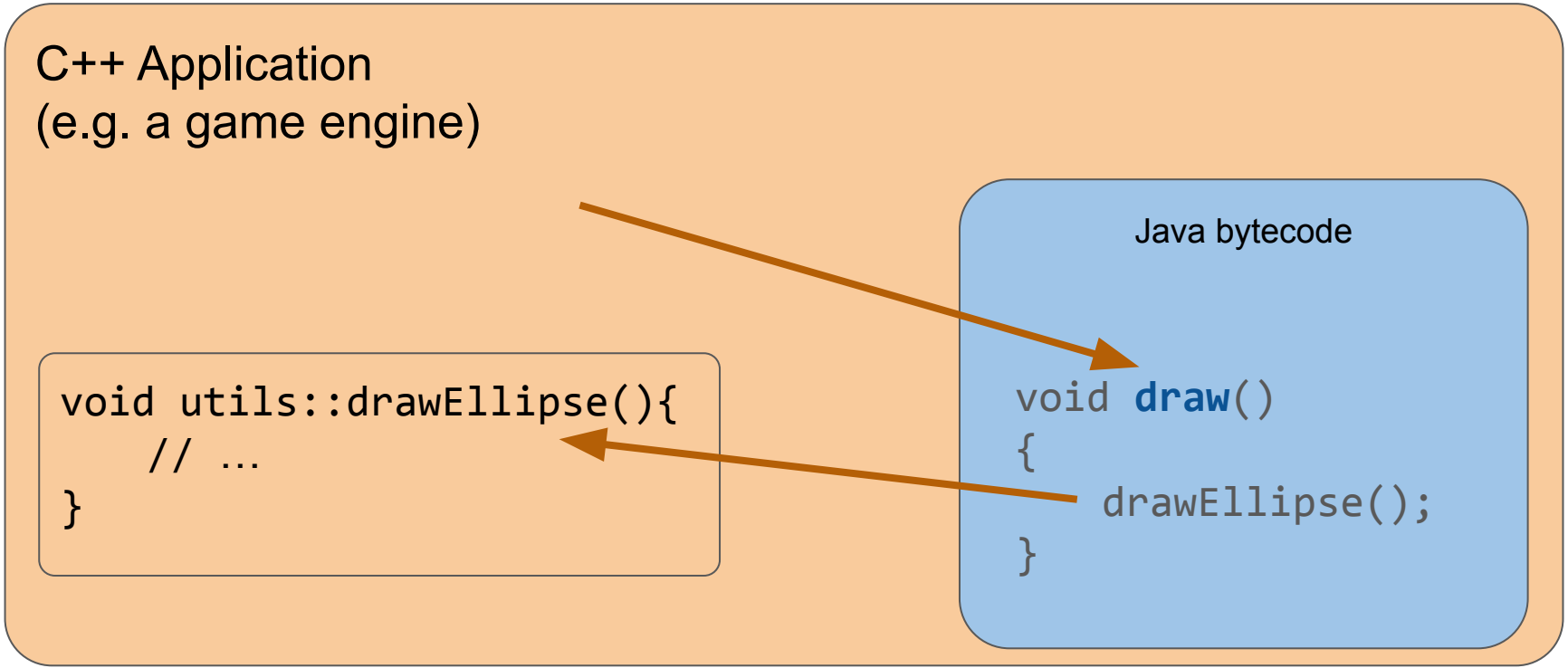
Koen Samyn

After the talk you will be able to:

- **Shift** runtime execution to compile time using `constexpr` and `consteval`.
- **Control** compile-time code generation using `if constexpr`.
- **Manipulate** program structures via static reflection using `std::meta::substitute`.
- **Combine** these C++26 features to compile Java bytecode directly into zero-overhead native machine code.

Part 0 - why


Interop
boundary
Call engine
functions
Java bytecode
Callbacks,
events, …
C++
Application
(e.g. a game engine)
marshalling
ownership
exceptions
threading
type mapping


C++ Application
(e.g. a game engine)
Java bytecode
void utils::drawEllipse(){
// …
}
void draw()
{
drawEllipse();
}

Part 1 - it all adds up

A simple Java program

```
int op1 = 5;
int op2 = 2;
int sum = op1 + op2;
```

```
int op1 = 5;
int op2 = 2;
int sum = op1 + op2;
```

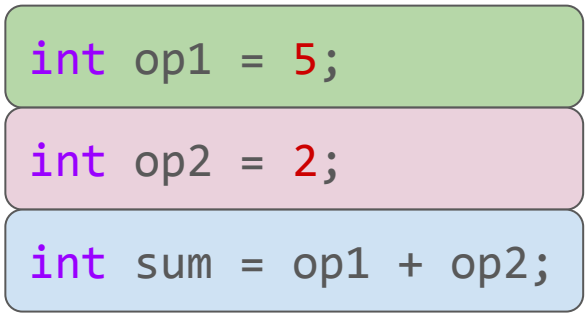
```
iconst_5
istore_0
```

```
int op1 = 5;
int op2 = 2;
int sum = op1 + op2;
```

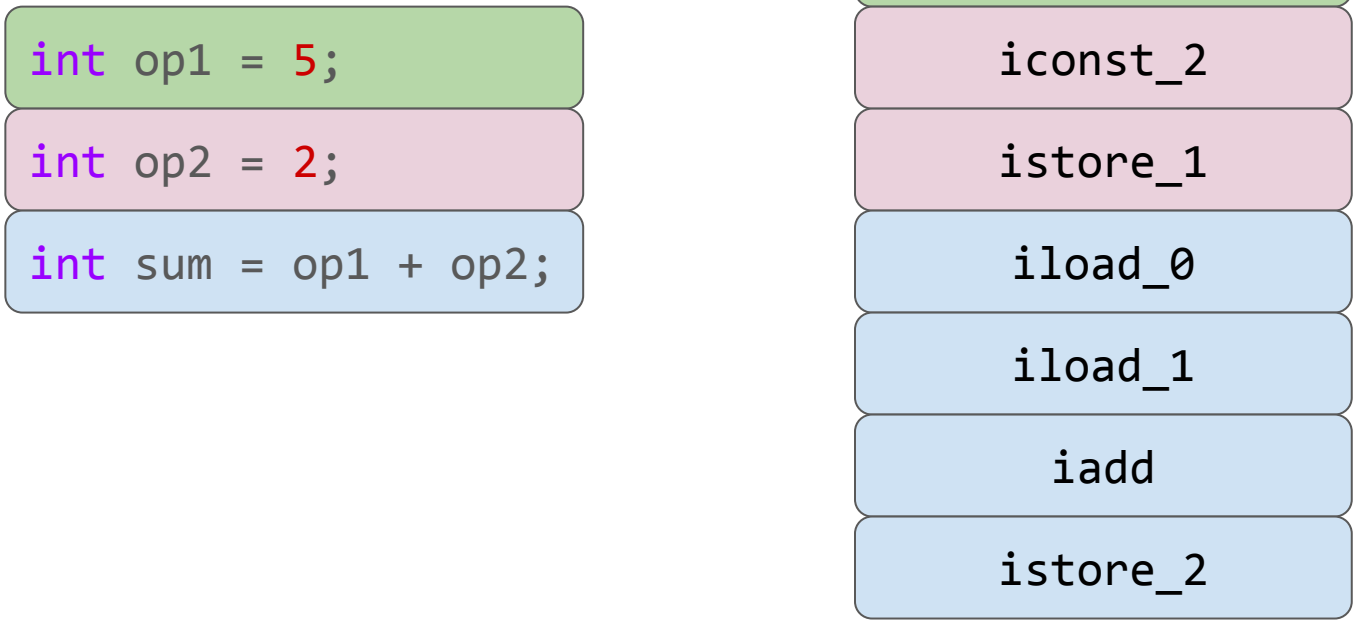
```
iconst_5
istore_0
```

```
int op1 = 5;
int op2 = 2;
int sum = op1 + op2;
```

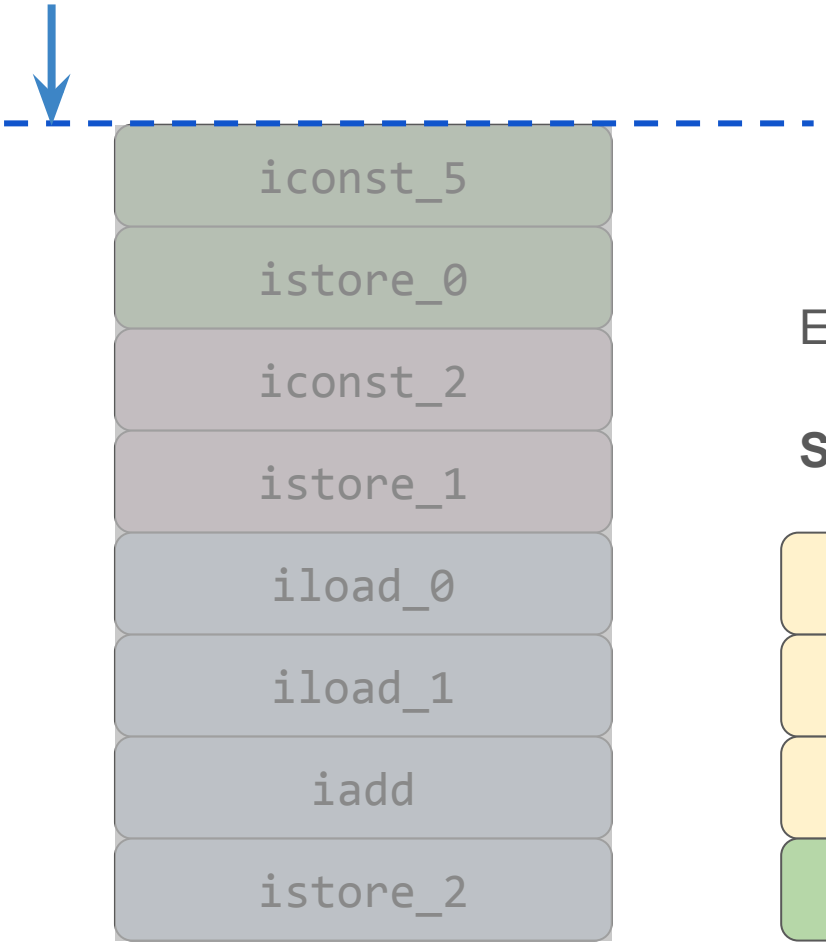
```
iconst_5
istore_0
iconst_2
istore_1
```


int op1 = 5;
int op2 = 2;
int sum = op1 + op2;

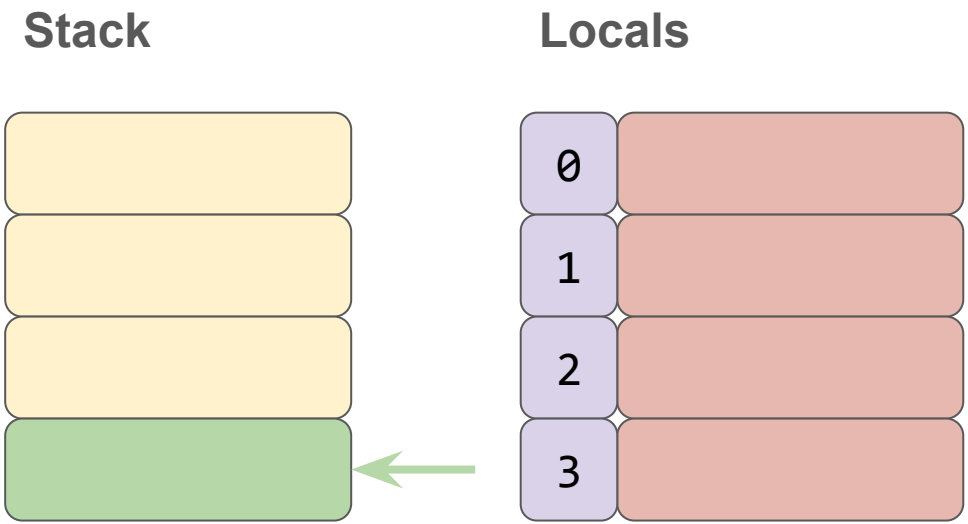

iconst_5
istore_0
iconst_2
istore_1

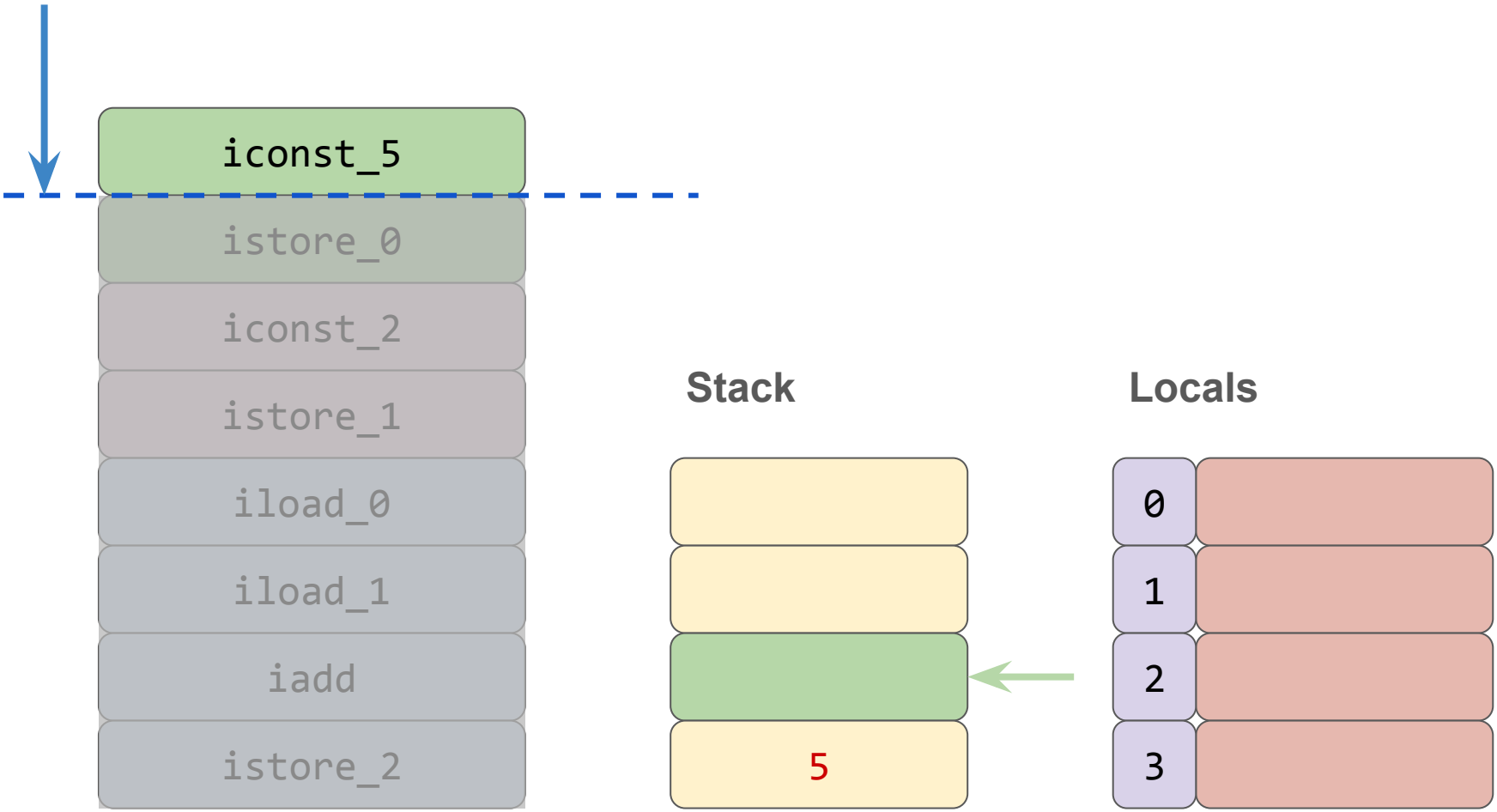

int op1 = 5;
int op2 = 2;
int sum = op1 + op2;
iconst_5
istore_0
iconst_2
istore_1
iload_0
iload_1
iadd
istore_2

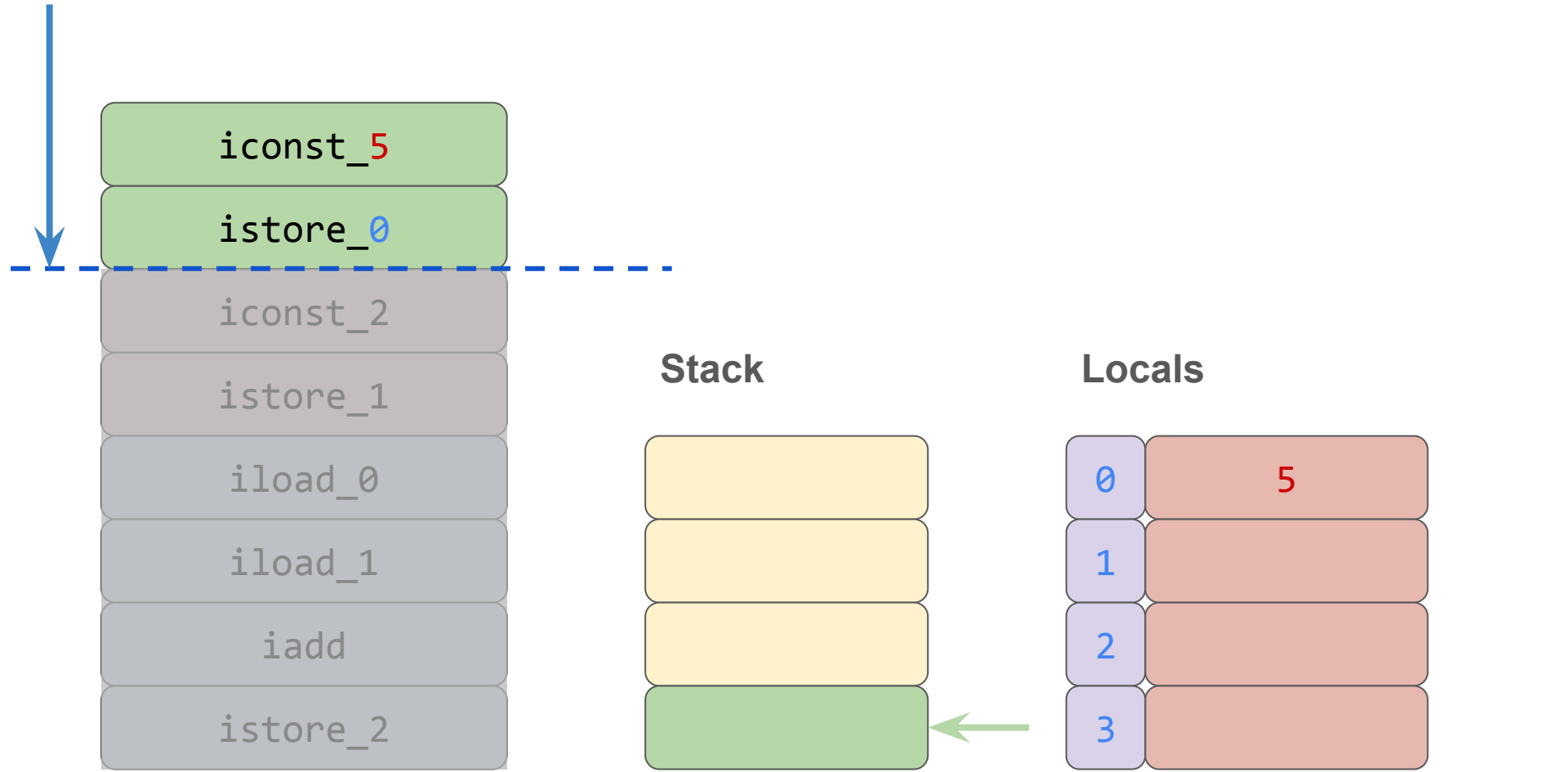
Bytecode execution

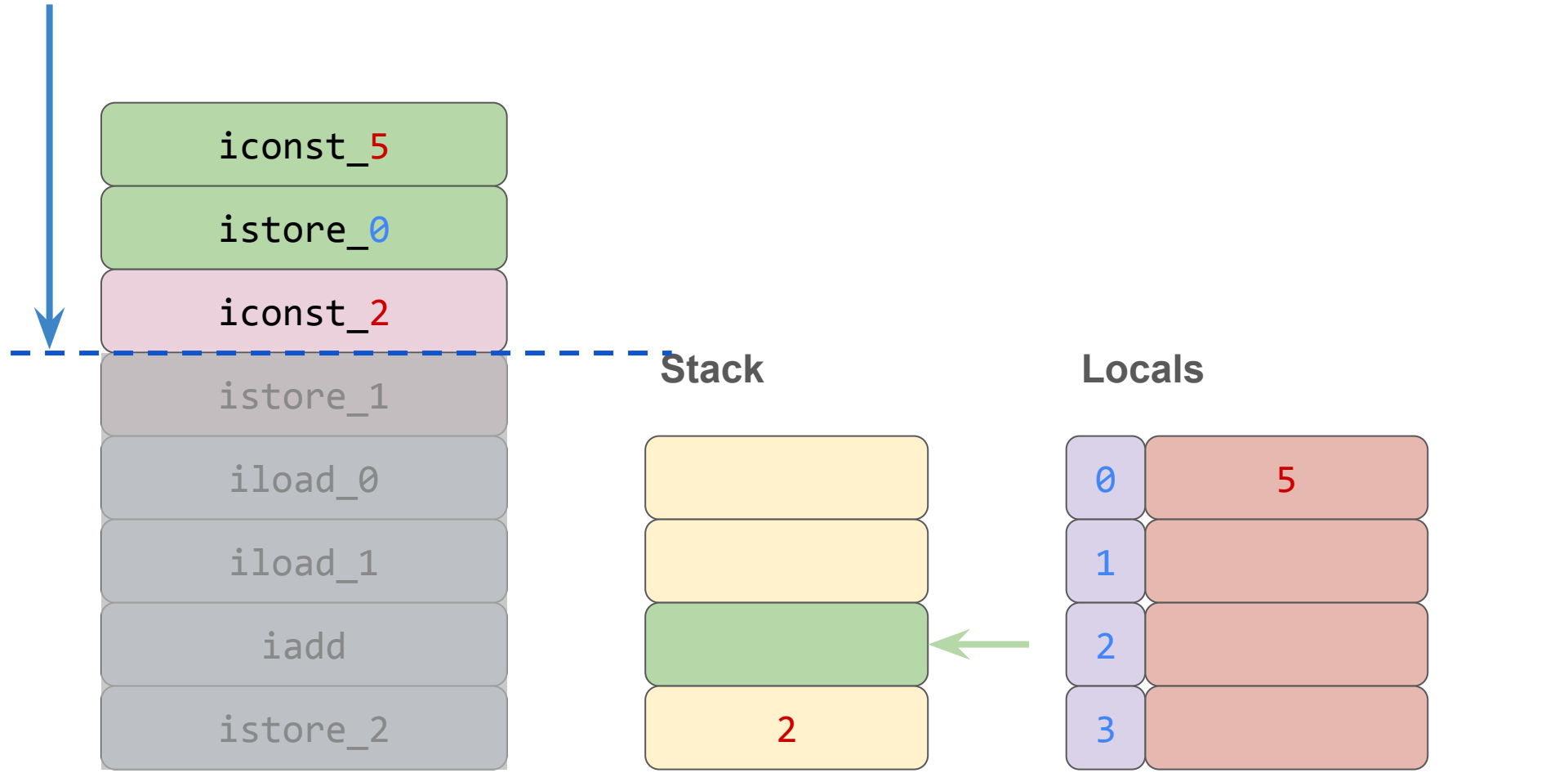


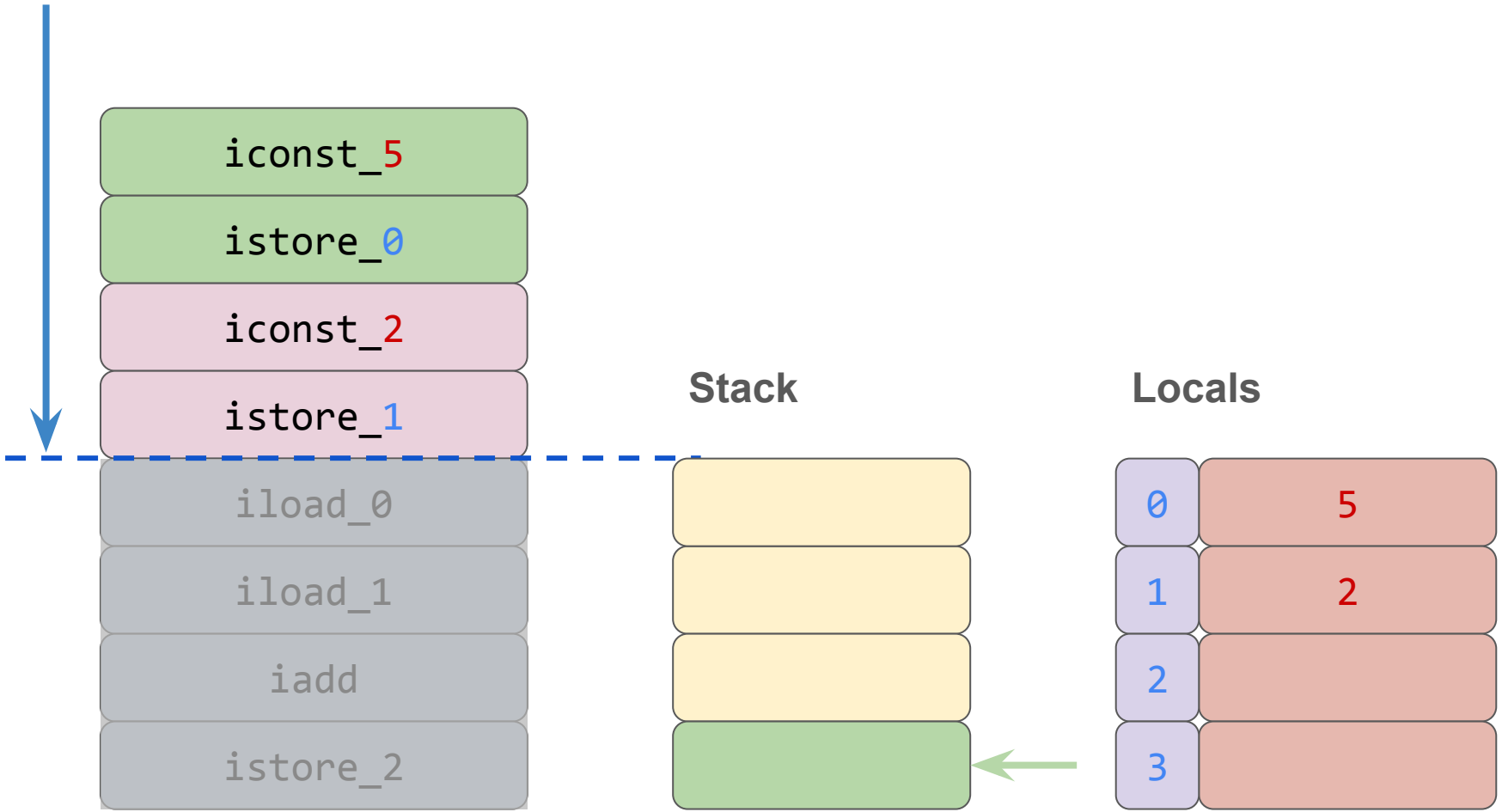
Empty stack pointer convention

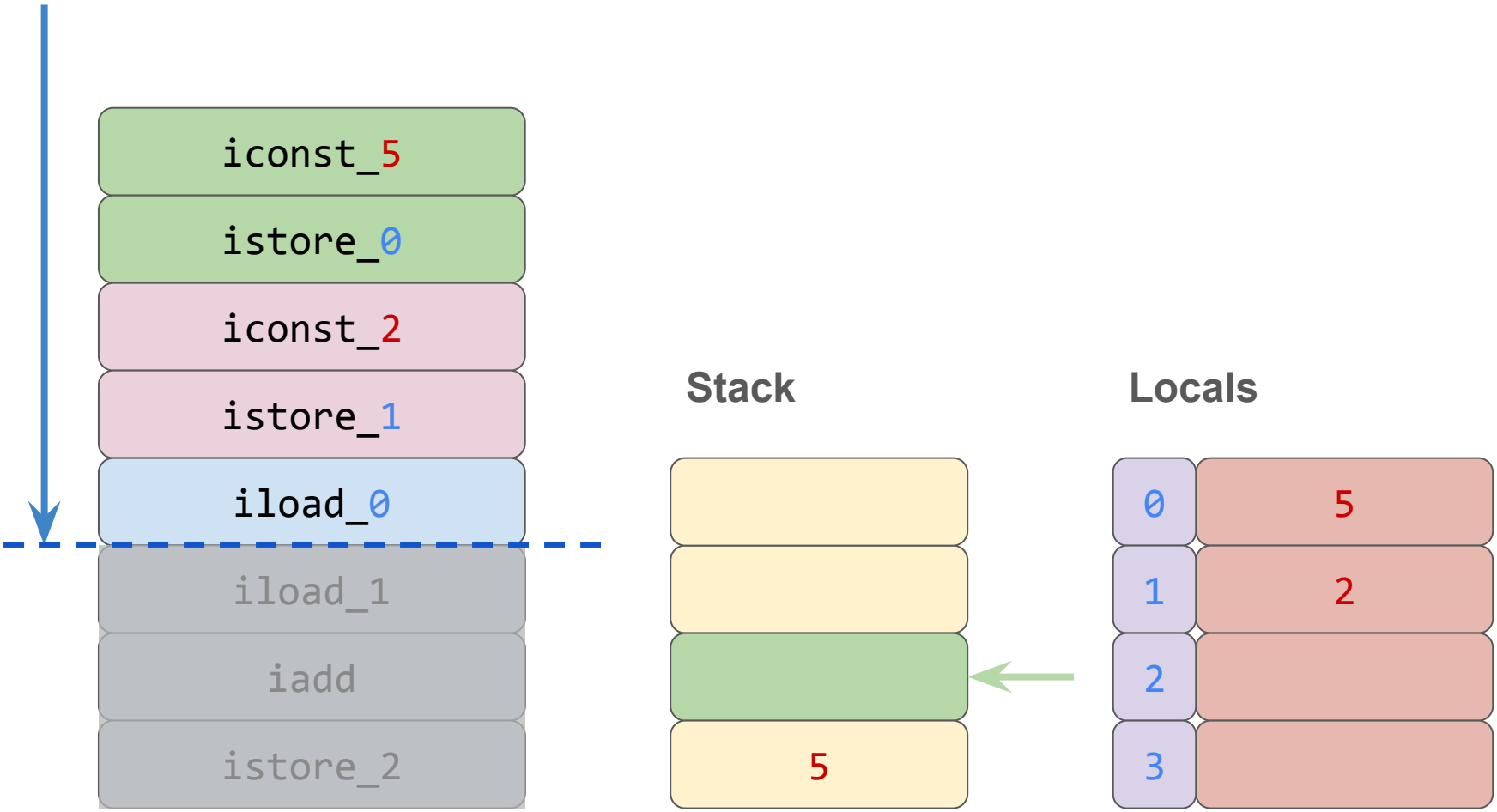


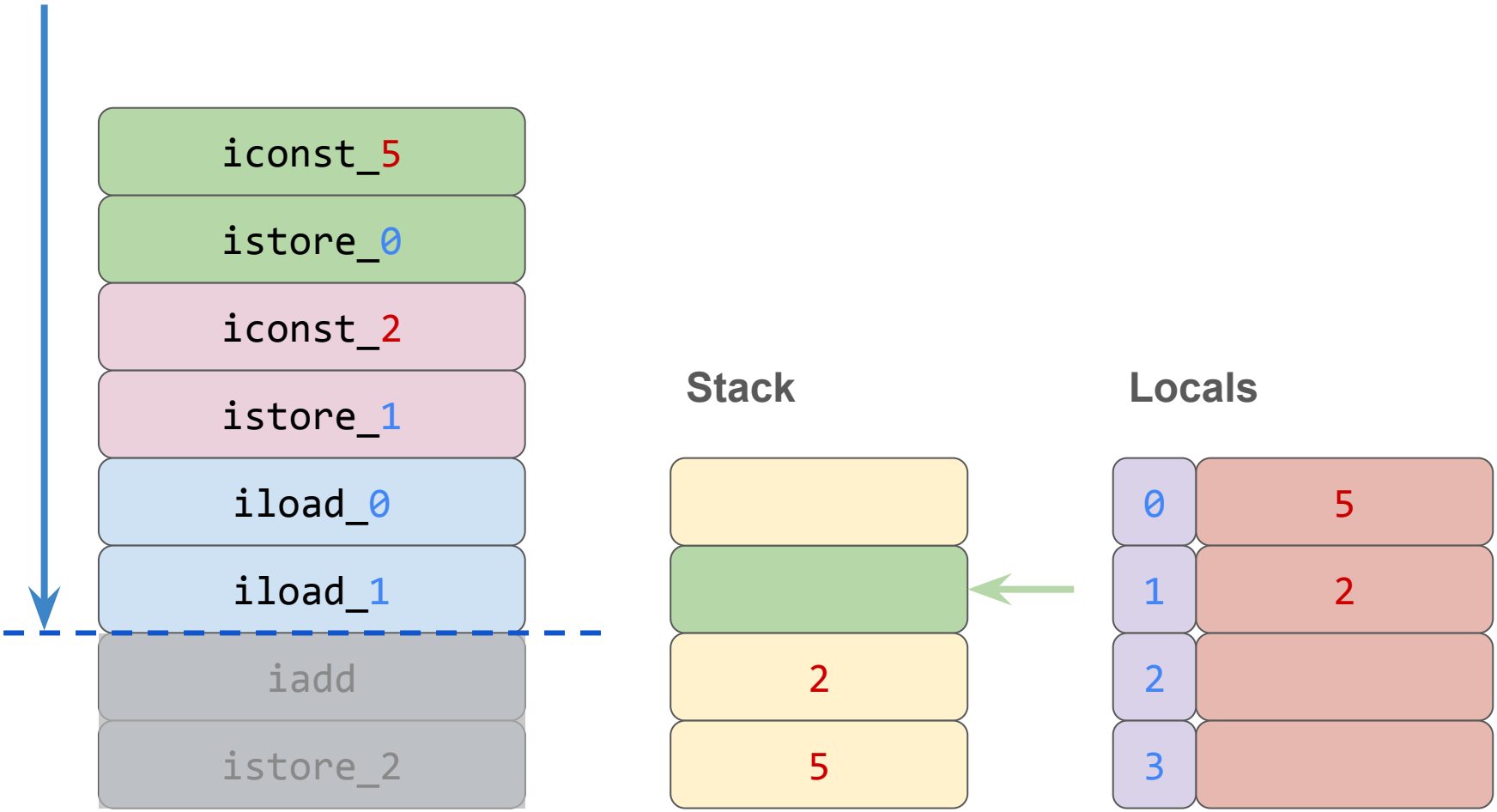

iconst_5
istore_0
iconst_2
istore_1
iload_0
iload_1
iadd
istore_2
Stack
5
Locals
0
1
2
3

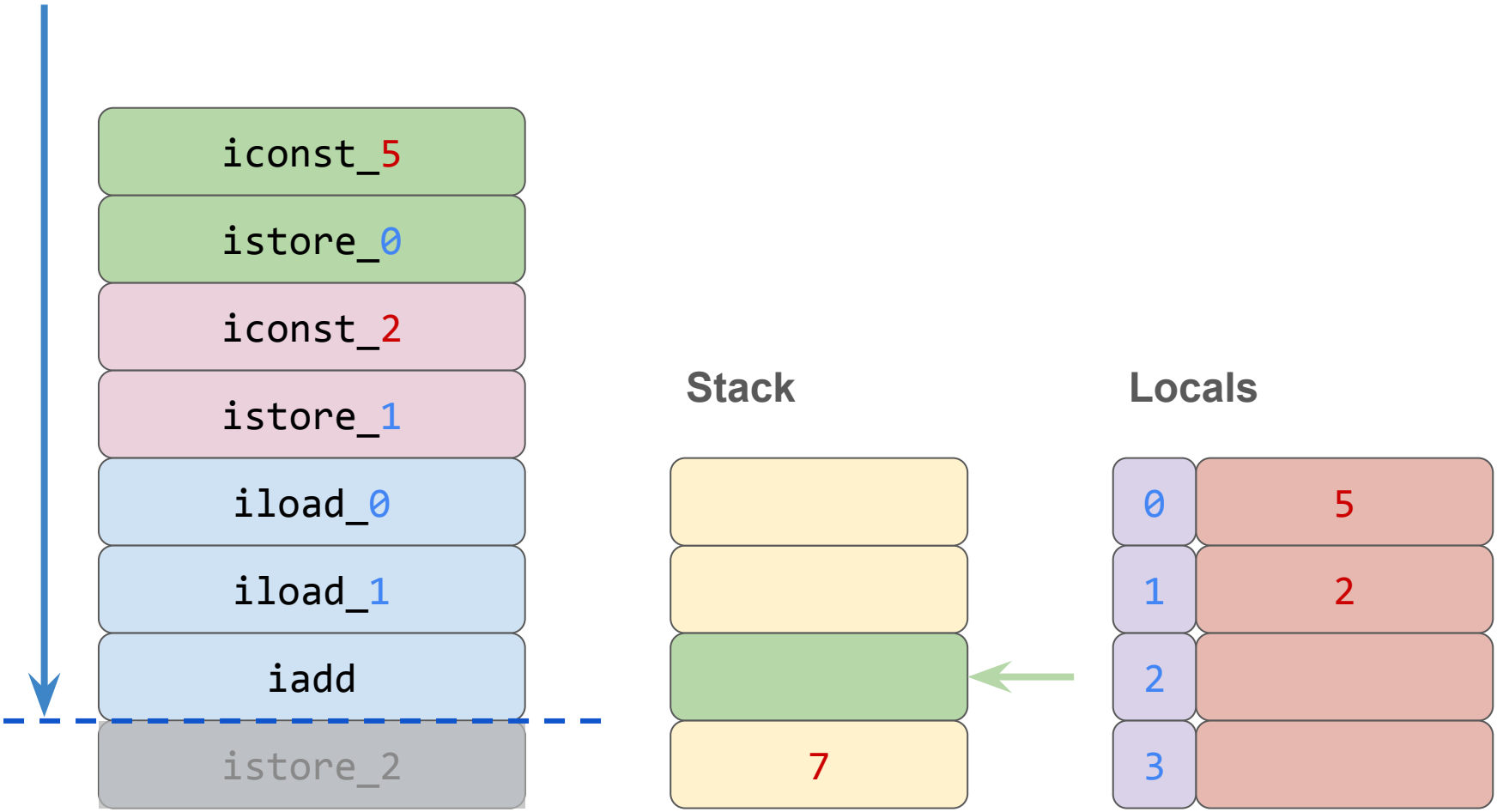

iconst_5
istore_0
iconst_2
istore_1
iload_0
iload_1
iadd
istore_2
Stack
Locals
0
5
1
2
3

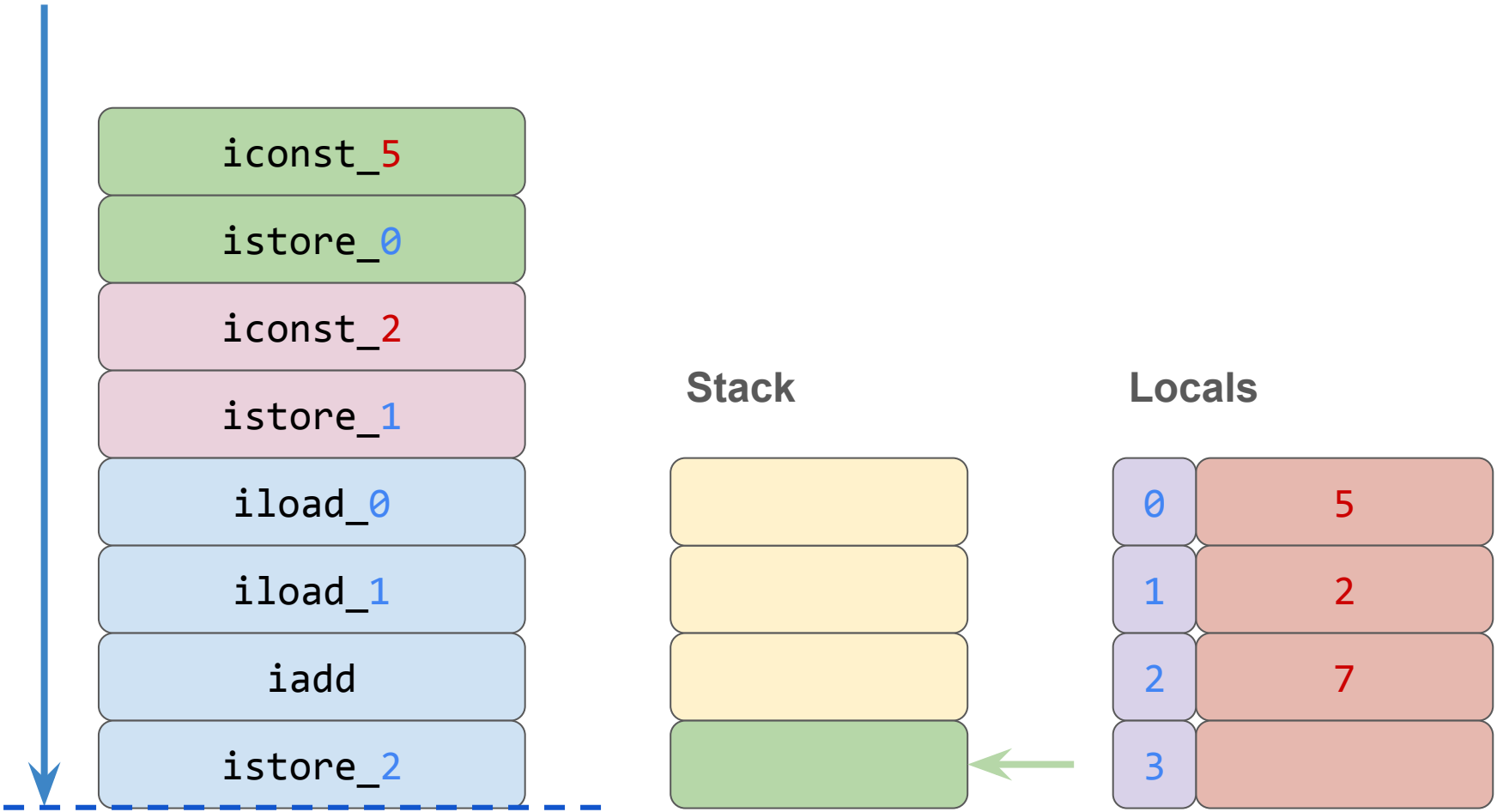

iconst_5
istore_0
iconst_2
istore_1
iload_0
iload_1
iadd
istore_2
Stack
2
Locals
0
5
1
2
3


iconst_5
istore_0
iconst_2
istore_1
iload_0
iload_1
iadd
istore_2
Stack
Locals
0
5
1
2
2
3


iconst_5
istore_0
iconst_2
istore_1
iload_0
iload_1
iadd
istore_2
Stack
5
Locals
0
5
1
2
2
3


iconst_5
istore_0
iconst_2
istore_1
iload_0
iload_1
iadd
istore_2
Stack
2
5
Locals
0
5
1
2
2
3


iconst_5
istore_0
iconst_2
istore_1
iload_0
iload_1
iadd
istore_2
Stack
7
Locals
0
5
1
2
2
3


iconst_5
istore_0
iconst_2
istore_1
iload_0
iload_1
iadd
istore_2
Stack
Locals
0
5
1
2
2
7
3

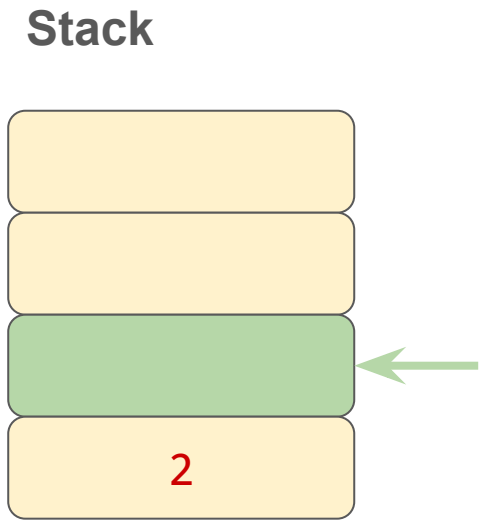
Part 2 - Interpret the bytecode in C++

Emulation principle

```
struct VM {
    std::array<int32_t, 4> stack{};
    std::array<int32_t, 4> locals{};
    uint8_t programCounter{0};
    uint8_t stackPointer{0};
};
```

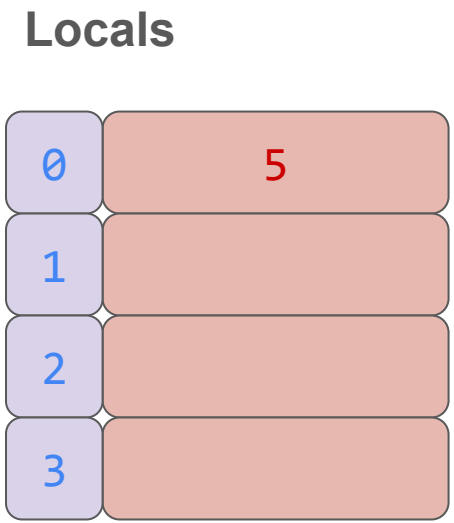
Push / pop

```
void push ( int32_t value ) {
    stack[stackPointer++] = value;
}

int32_t pop () {
    return stack[--stackPointer];
}
```

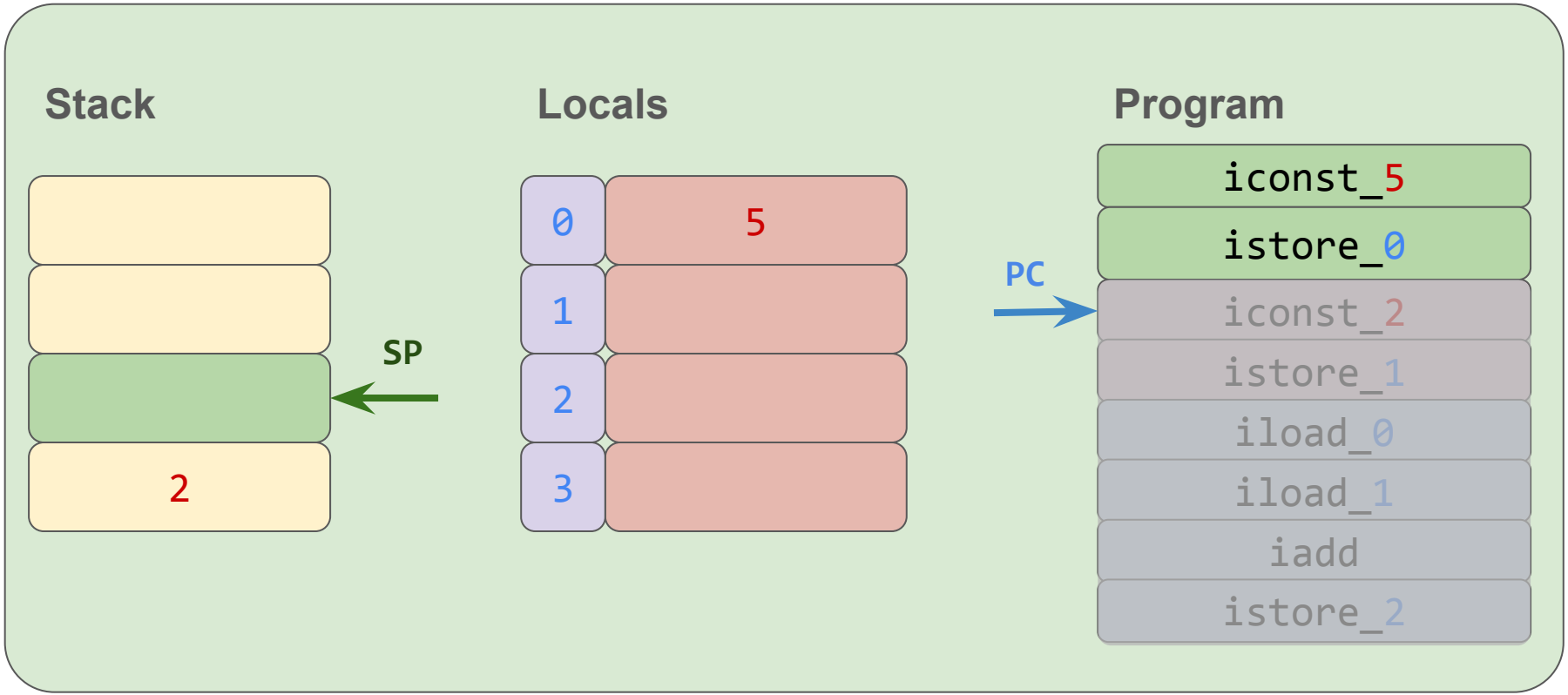


Store / load

```
void store( uint8_t idx ) {
    int32_t value = pop();
    locals[idx] = value;
}

void load ( uint8_t idx ) {
    push(locals[idx]);
}
```



VM

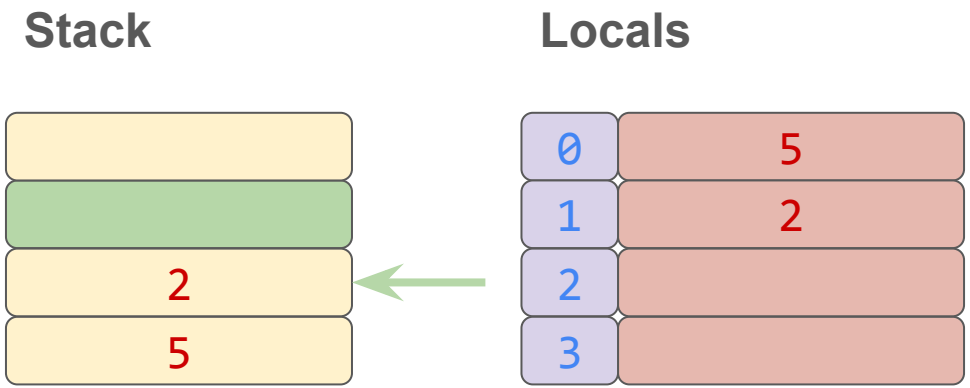

Stack
Locals
Program
SP
PC
2
0
5
1
2
3
iconst_5
istore_0
iconst_2
istore_1
iload_0
iload_1
iadd
istore_2

Execute function

```
void execute(VM& vm, OpCode op)
{
    switch(op){
        // Constants
        case ICONST_2: vm.push(2);break;
        case ICONST_5: vm.push(5);break;
        // Store stack constant into locals
        case ISTORE_0: vm.store(0);break;
        case ISTORE_1: vm.store(1);break;
        case ISTORE_2: vm.store(2);break;
        // Load from locals into stack
        case ILOAD_0: vm.load(0);break;
        case ILOAD_1: vm.load(1);break;
```

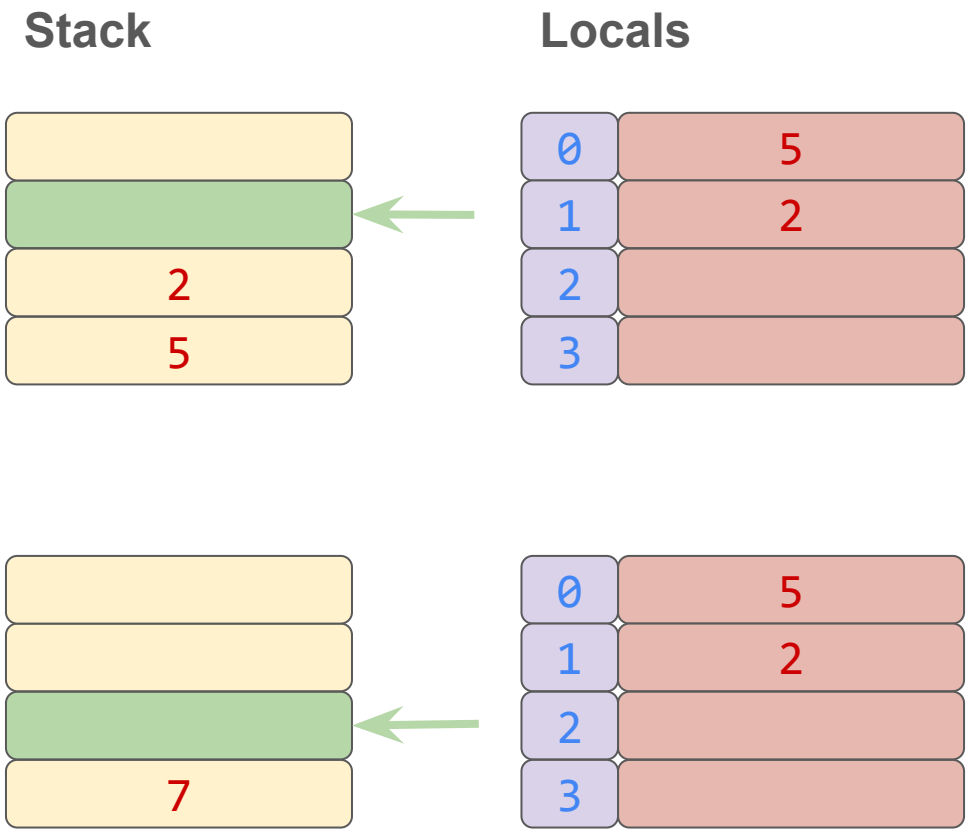
Execute function

```
        case IADD:{
            int32_t op1 = vm.pop();
            int32_t op2 = vm.pop();
            vm.push(op1+op2);
            break;
        }
    }
    vm.programCounter++;
    vm.print();
}
```



Execute function

```
        case IADD:{
            int32_t op1 = vm.pop();
            int32_t op2 = vm.pop();
            vm.push(op1+op2);
            break;
        }
    }
    vm.programCounter++;
    vm.print();
}
```



Main function (https://godbolt.org/z/6oovoY1hT)

```
int main(){
    VM vm;
    while(vm.programCounter < program.size())
    {
        OpCode toExecute = program[vm.programCounter];
        execute(vm, toExecute);
    }
    return vm.locals[2];
}
```

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 7
```

Main function (https://godbolt.org/z/6oovoY1hT)

```
"main":
        sub     rsp, 56
        pxor    xmm0, xmm0
        xor     edx, edx
        xor     eax, eax
        mov     DWORD PTR [rsp+32], 0
        movaps  XMMWORD PTR [rsp], xmm0
        movaps  XMMWORD PTR [rsp+16], xmm0
.L18:
        movzx   ecx, al
        mov     rdi, rsp
        mov     DWORD PTR [rsp+24], edx
        mov     esi, DWORD PTR "program"[0+rcx*4
        mov     BYTE PTR [rsp+32], al
        call    "execute(VM&, OpCode)"
        movzx   eax, BYTE PTR [rsp+32]
        mov     edx, DWORD PTR [rsp+24]
        cmp     al, 7
        jbe     .L18
        mov     eax, edx
        add     rsp, 56
        ret
```

Compiled with GCC 16.1 -O2

Part 2b - Remove the program counter

```
int main(){
    VM vm;
    for( OpCode op : program )
    {
        execute(vm, op);
    }
    return vm.locals[2];
}
```

```
106  "main":
107          sub     rsp, 56
108          pxor    xmm0, xmm0
109          xor     edx, edx
110          xor     eax, eax
111          mov     DWORD PTR [rsp+32], 0
112          movaps  XMMWORD PTR [rsp], xmm0
113          movaps  XMMWORD PTR [rsp+16], xmm0
114  .L18:
115          movzx   ecx, al
116          mov     rdi, rsp
117          mov     DWORD PTR [rsp+24], edx
118          mov     esi, DWORD PTR "program"[0+rcx*4
119          mov     BYTE PTR [rsp+32], al
120          call    "execute(VM&, OpCode)"
121          movzx   eax, BYTE PTR [rsp+32]
122          mov     edx, DWORD PTR [rsp+24]
123          cmp     al, 7
124          jbe     .L18
125          mov     eax, edx
126          add     rsp, 56
127          ret
```

Part 3 - Templates and if constexpr

Idea :

1. encode the OpCode as a non type template parameter (NTTP) to an execute function
2. Instantiate the execute template
3. Store the function pointers in
    a. an array of function pointers
    b. a compile-time parameter pack of function pointers

Execute function with if constepxr

```
template<OpCode op>
void execute(VM& vm)
{
    if constexpr(op == ICONST_0) {
        vm.push(0);
    }else if constexpr(op == ICONST_1) {
        vm.push(1);
    }else if constexpr(op == ICONST_2) {
        vm.push(2);
    }
    // ...
}
```

Program

```
int main(){
    using InstructionFn = void(*)(VM&);
    std::array<InstructionFn,8> program = {
        execute<ICONST_5>,
        execute<ISTORE_0>,
        execute<ICONST_2>,
        execute<ISTORE_1>,
        execute<ILOAD_0>,
        execute<ILOAD_1>,
        execute<IADD>,
        execute<ISTORE_2>
    };
```

Program

```
int main(){
    using InstructionFn = void(*)(VM&);
    std::array<InstructionFn,8> program = {
        execute<ICONST_5>,
        execute<ISTORE_0>,
        execute<ICONST_2>,
        execute<ISTORE_1>,
        execute<ILOAD_0>,
        execute<ILOAD_1>,
        execute<IADD>,
        execute<ISTORE_2>
    };


    VM vm;
    for (auto instruction : program) {
        instruction(vm);
    }
    return vm.locals[2];
}
```

https://godbolt.org/z/bbf4c98sM

Clang result (-O2)

```
1  main:
2          mov     eax, 7
3          ret
```

https://godbolt.org/z/bbf4c98sM

Clang result (-O2)

```
1   main:
2           mov     eax, 7
3           ret
```

What is the magic that enables this ?

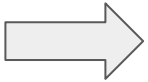
SRA - scalar replacement of aggregates

```
struct Frame {
     int stack[8];
     int sp;
     int result;
};


Frame frame{};
frame.stack[frame.sp++] = 3;
frame.stack[frame.sp++] = 4;
frame.result = frame.stack[--frame.sp]
             + frame.stack[--frame.sp];
return frame.result;
```

SRA - scalar replacement of aggregates

```
struct Frame {
     int stack[8];
     int sp;
     int result;
};
```

SRA

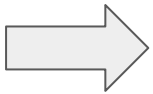


```
Frame frame{};
frame.stack[frame.sp++] = 3;
frame.stack[frame.sp++] = 4;
frame.result = frame.stack[--frame.sp]
             + frame.stack[--frame.sp];
return frame.result;
```

```
int a = 3;
int b = 4;
int result = a + b;
return result;
```

GCC Result (-O3)

```
"main":
        push    rbx
        sub     rsp, 112
        movq    xmm0, QWORD PTR .LC0[rip]
        movhps  xmm0, QWORD PTR .LC1[rip]
        mov     DWORD PTR [rsp+32], 0
        lea     rbx, [rsp+48]
        movaps  XMMWORD PTR [rsp+48], xmm0
        movq    xmm0, QWORD PTR .LC2[rip]
        movhps  xmm0, QWORD PTR .LC3[rip]
        movaps  XMMWORD PTR [rsp+64], xmm0
        movq    xmm0, QWORD PTR .LC4[rip]
        movhps  xmm0, QWORD PTR .LC5[rip]
        movaps  XMMWORD PTR [rsp+80], xmm0
        movq    xmm0, QWORD PTR .LC6[rip]
        movhps  xmm0, QWORD PTR .LC7[rip]
        movaps  XMMWORD PTR [rsp+96], xmm0
        pxor    xmm0, xmm0
        movaps  XMMWORD PTR [rsp], xmm0
        movaps  XMMWORD PTR [rsp+16], xmm0
.L11:
        mov     rdi, rsp
        call    [QWORD PTR [rbx]]
        add     rbx, 8
        lea     rax, [rsp+112]
        cmp     rax, rbx
        jne     .L11
        mov     eax, DWORD PTR [rsp+24]
        add     rsp, 112
        pop     rbx
        ret
```



Not optimized!

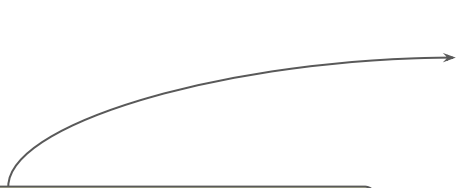
Compile time sequence

```
using InstructionFn = void(*)(VM&);

template<InstructionFn ... Instructions>
void block(VM& vm) {
    (Instructions(vm), ...);
}
```

https://godbolt.org/z/4EP5nhK3x

Compile time sequence

```
using InstructionFn = void(*)(VM&);

template<InstructionFn ... Instructions>
void block(VM& vm) {
    (Instructions(vm), ...);
}
```



https://godbolt.org/z/4EP5nhK3x

Compile time sequence

```
using InstructionFn = void(*)(VM&);

template<InstructionFn ... Instructions>
void block(VM& vm) {
    (Instructions(vm), ...);
}
```

Non Type Template Parameter pack of function pointers

Comma fold expression

https://godbolt.org/z/4EP5nhK3x

Compile time sequence

```
int main(){
    InstructionFn program = block<
        execute<ICONST_5>,
        execute<ISTORE_0>,
        execute<ICONST_2>,
        execute<ISTORE_1>,
        execute<ILOAD_0>,
        execute<ILOAD_1>,
        execute<IADD>,
        execute<ISTORE_2>
    >;

    VM vm;
    program(vm);
    return vm.local(2);
}
```

Summary so far

Version	Opcode	Sequence	Clang	GCC
Interpreter	runtime	runtime	❌	❌
Function pointers and for loop	compile-time	runtime	✅	❌
Fold expression	compile-time	compile-time	✅	✅

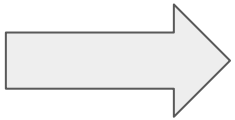
Part 4 - Reflection

Goal

Program

```
iconst_5
istore_0
iconst_2
istore_1
iload_0
iload_1
iadd
istore_2
```

Reflection



```
int main(){
    InstructionFn program = block<
        execute<ICONST_5>,
        execute<ISTORE_0>,
        execute<ICONST_2>,
        execute<ISTORE_1>,
        execute<ILOAD_0>,
        execute<ILOAD_1>,
        execute<IADD>,
        execute<ISTORE_2>
    >;

    VM vm;
    program(vm);
    return vm.local(2);
}
```

Reflection operators

^^

Reflection operator

Colloquially : cat-ears operator

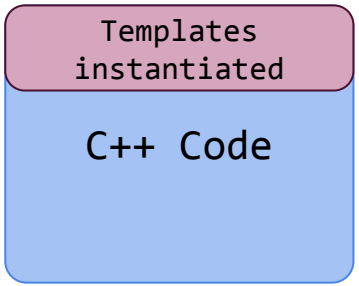
Returns **metadata** from variables or types.

[: ... :]

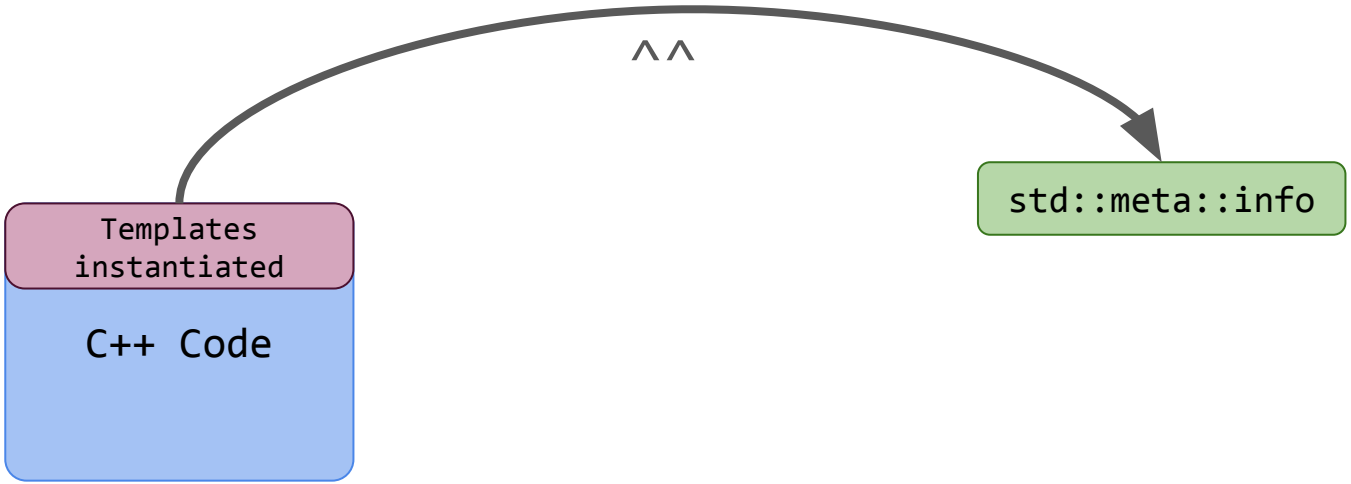
Splice operator

Use this **metadata** in a context.

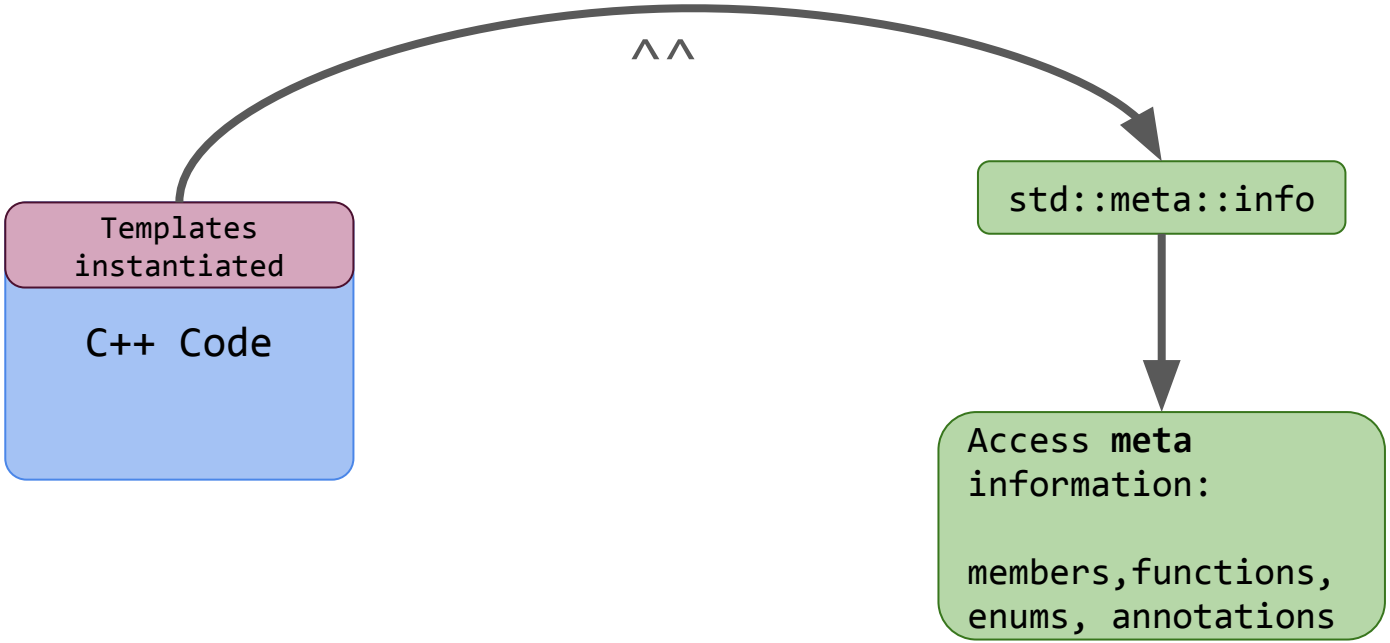
Reflection overview



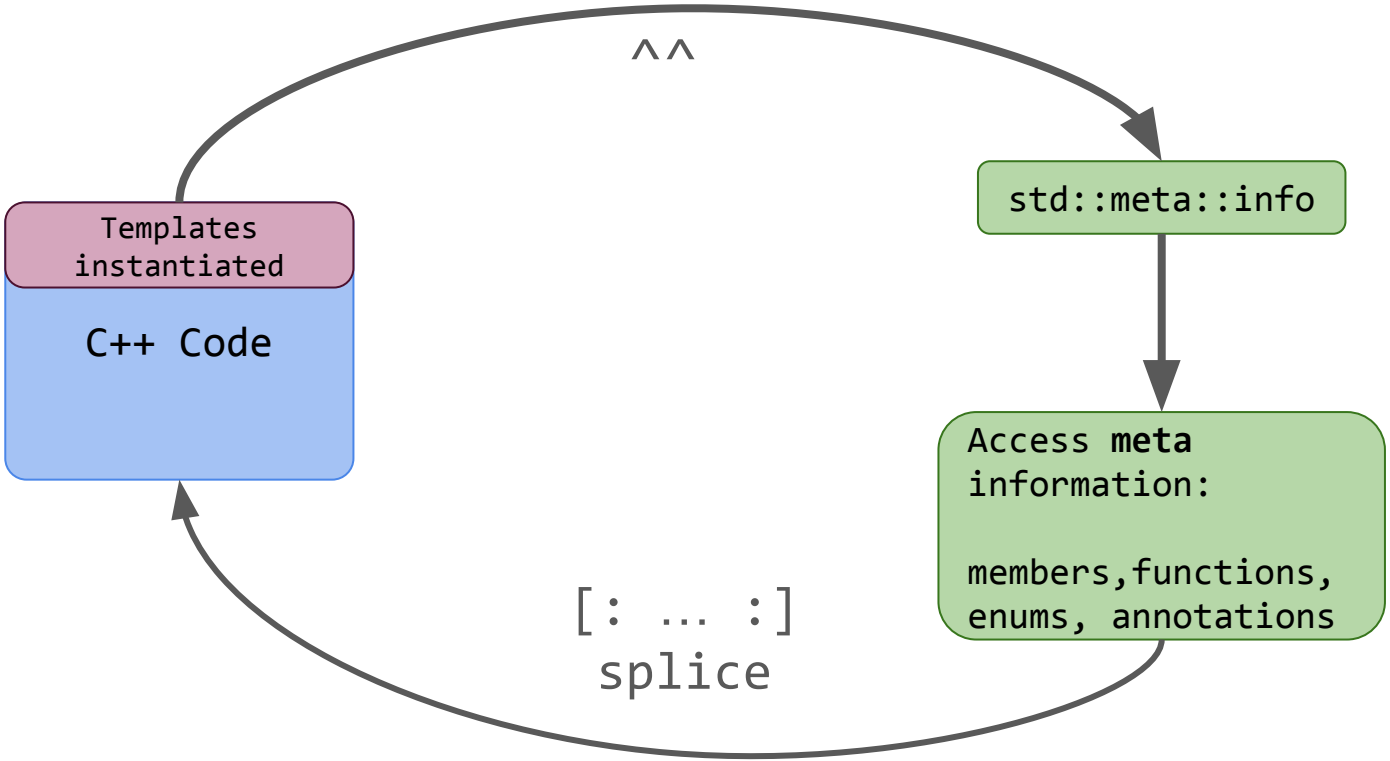
Reflection overview - reflection operator



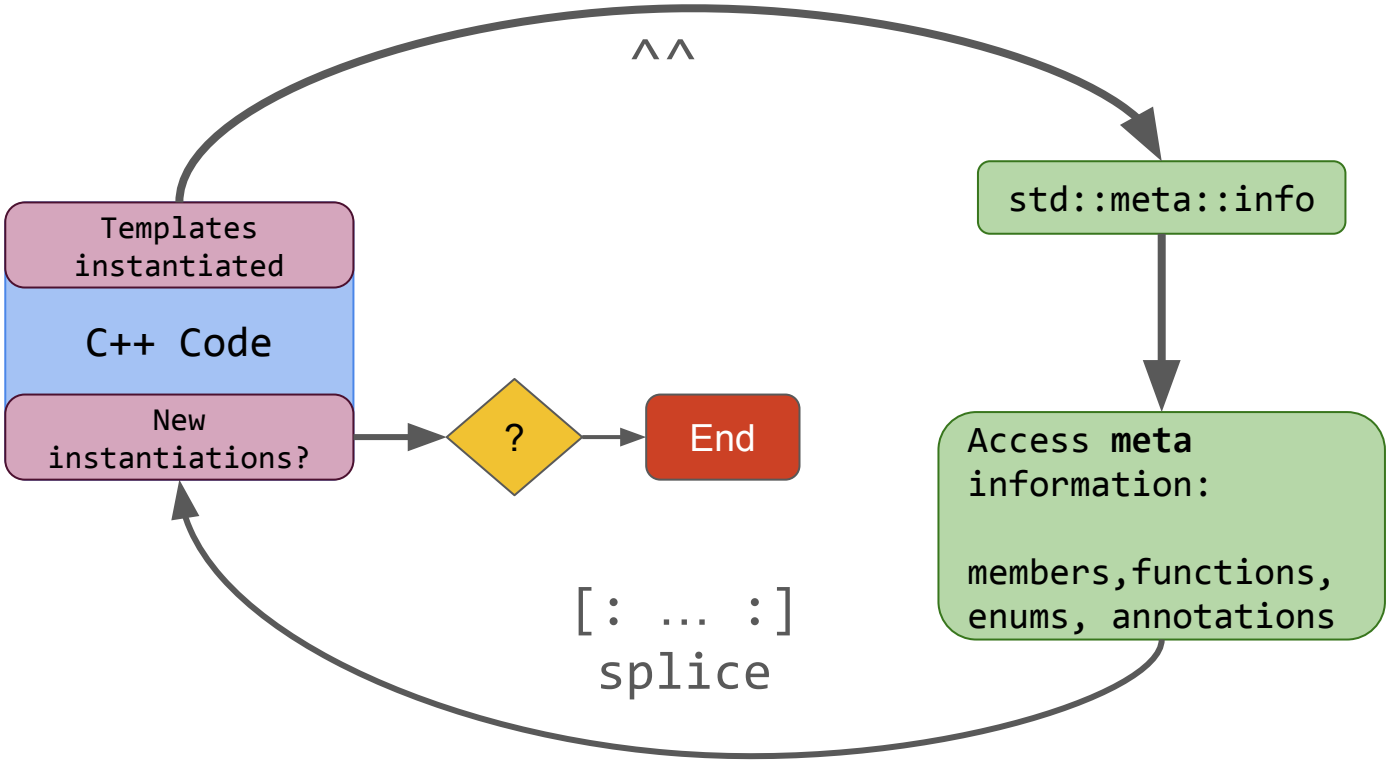
Reflection overview - query functions



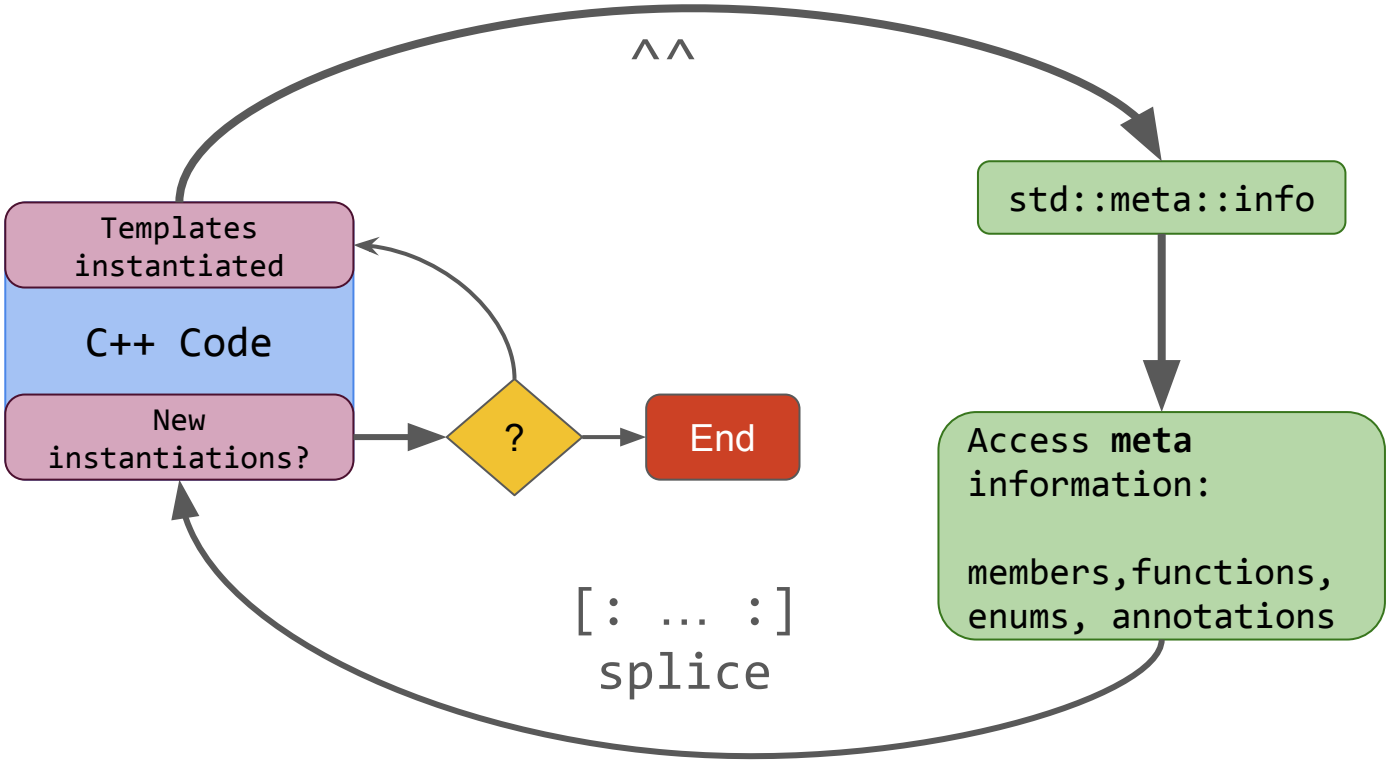
Reflection overview - splice operator



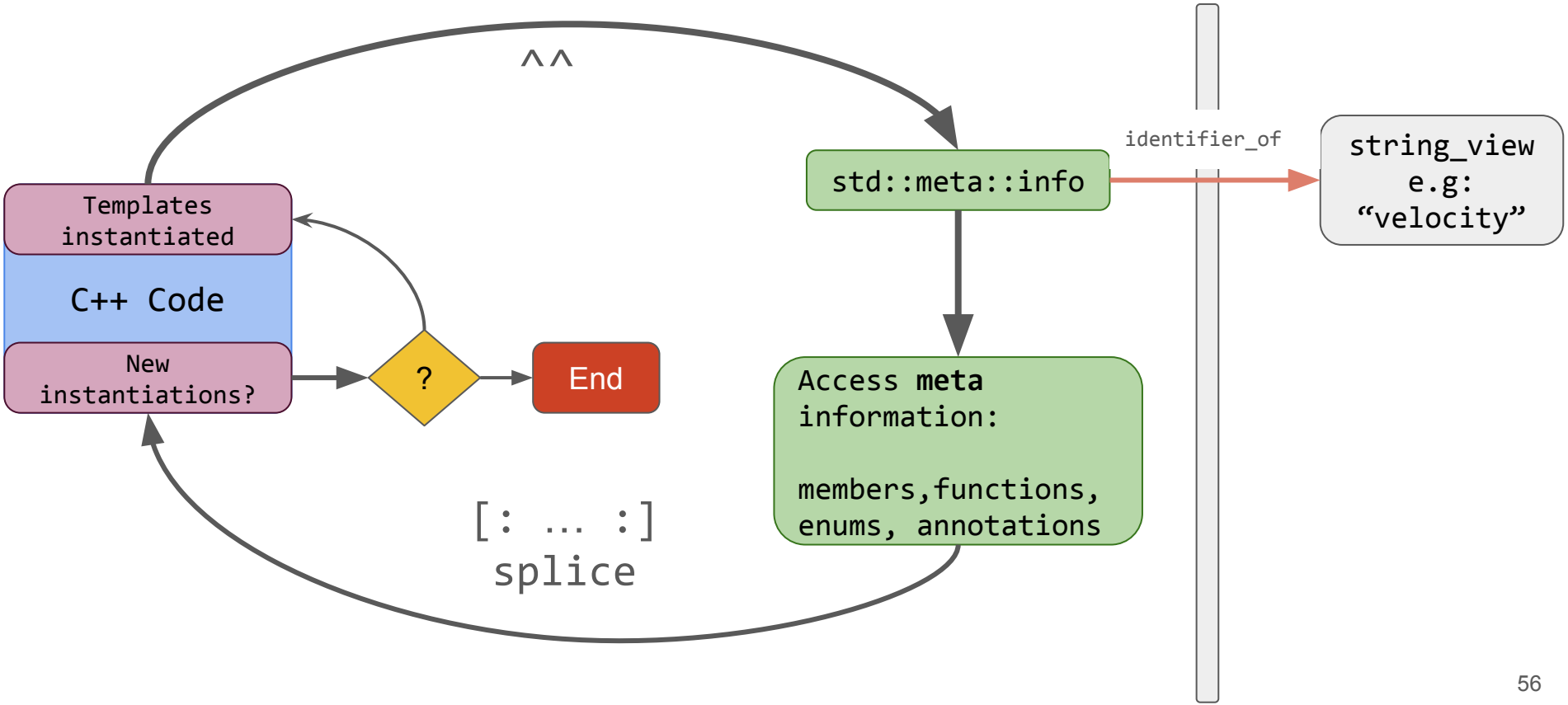
Reflection overview - end of reflection



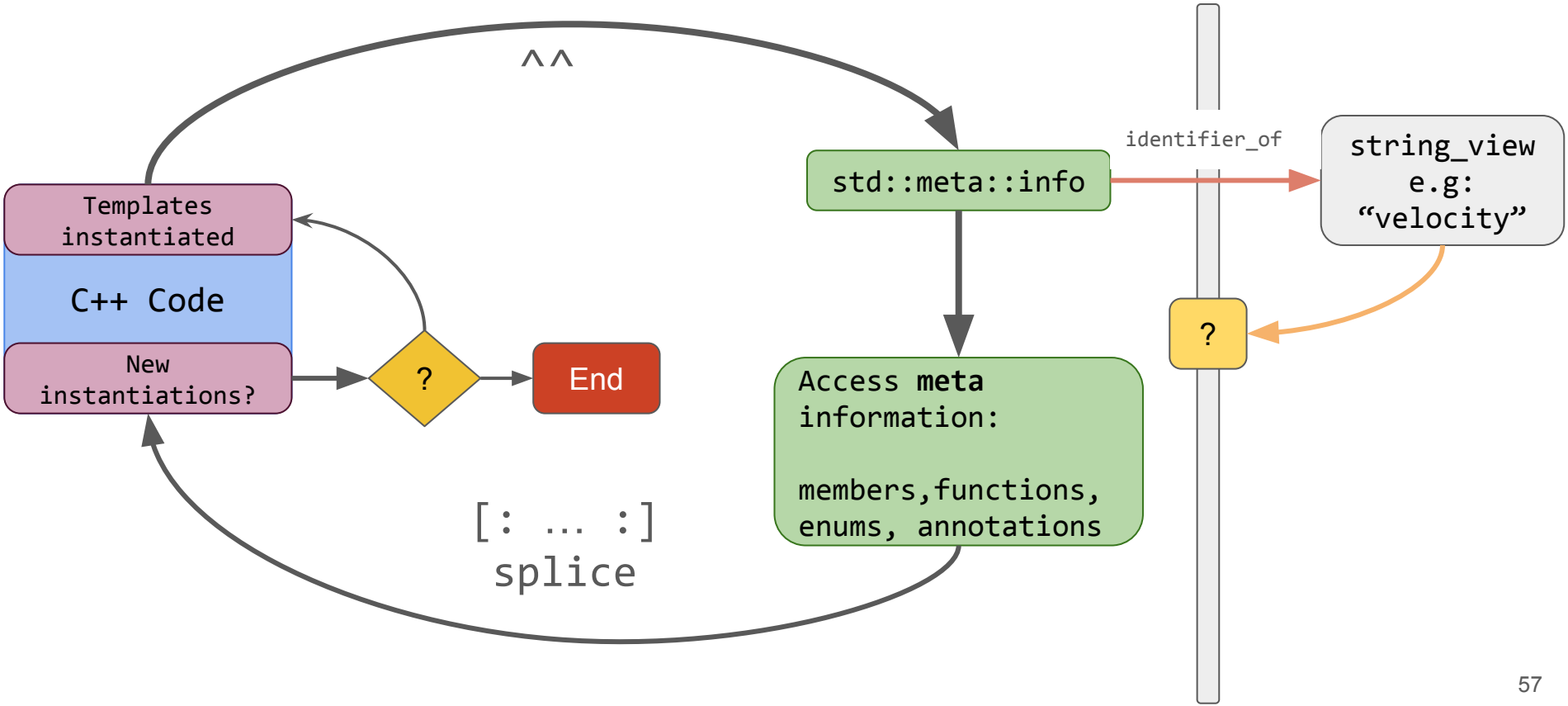
Reflection overview - loop possibility



Reflection overview - escaping out of reflection land



Reflection overview - getting back into reflection land



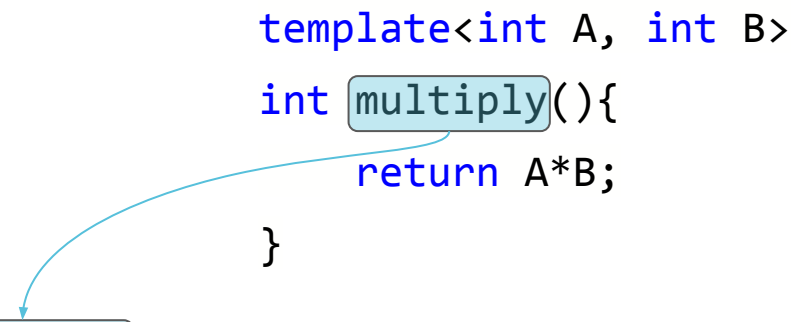
Let’s reflect something

```
template<int A, int B>
int multiply(){
    return A*B;
}

int main(){
    int result = multiply<3, 4>();
    return result;
}
```

Let’s reflect something

How to reflect the multiply function:

```
std::meta::info executeInfo = ^^multiply;
```



Let's reflect something

```
template<int A, int B>
int multiply(){
    return A*B;
}
```

How to reflect the multiply function:

```
constexpr std::meta::info rMultiply = ^^multiply;
```

Reflect the constants so they can be used as template arguments:

```
constexpr std::meta::info rA = std::meta::reflect_constant(3);
constexpr std::meta::info rB = std::meta::reflect_constant(4);
```

Reflect and finally splice

```
template<int A, int B>
int multiply(){
    return A*B;
}
```

How to reflect the `multiply` function:

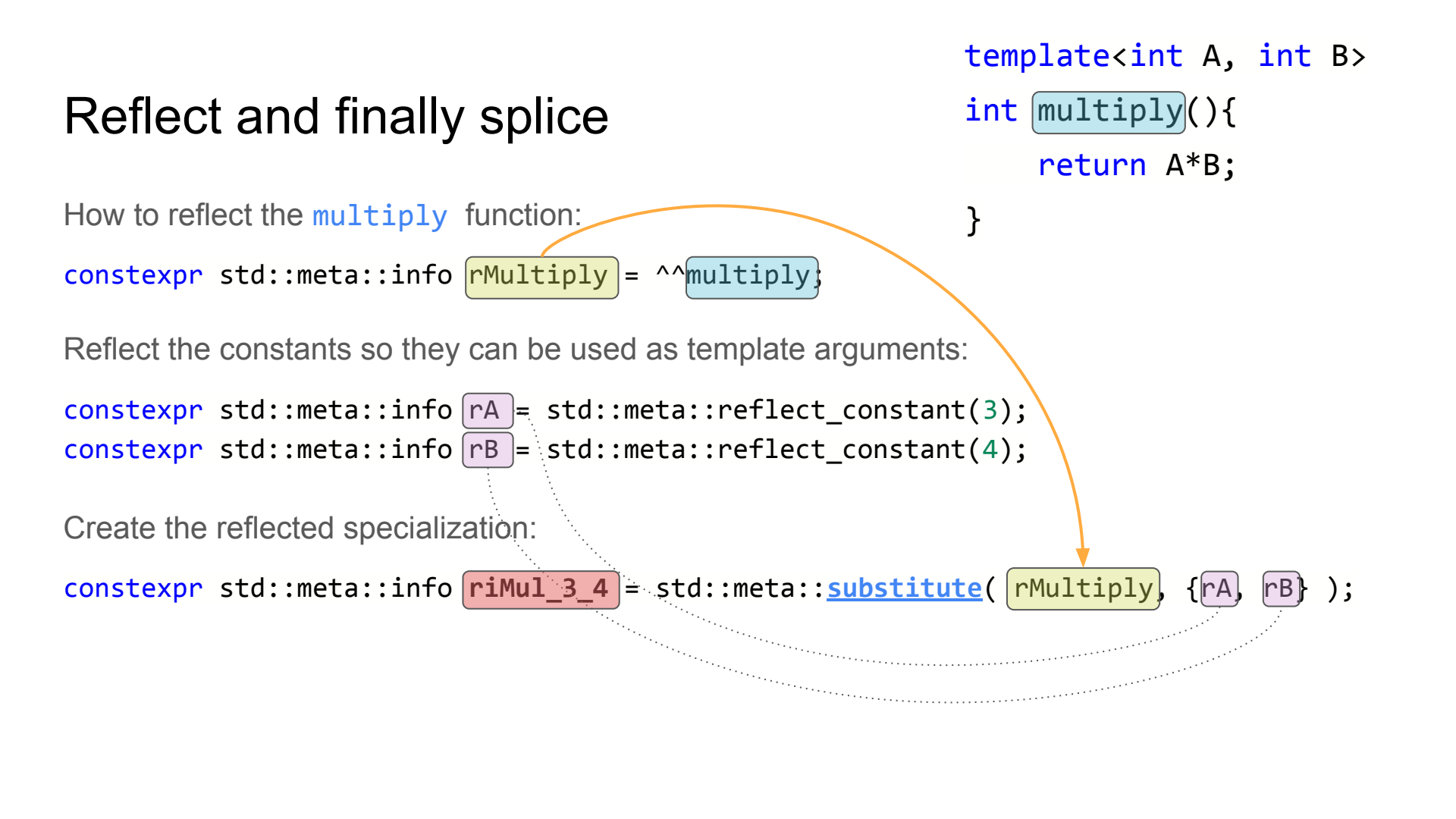
```
constexpr std::meta::info rMultiply = ^^multiply;
```

Reflect the constants so they can be used as template arguments:

```
constexpr std::meta::info rA = std::meta::reflect_constant(3);
constexpr std::meta::info rB = std::meta::reflect_constant(4);
```

Create the reflected specialization:

```
constexpr std::meta::info riMul_3_4 = std::meta::substitute( rMultiply, {rA, rB} );
```



Reflect and finally splice

```
template<int A, int B>
int multiply(){
    return A*B;
}
```

How to reflect the `multiply` function:

```
constexpr std::meta::info rMultiply = ^^multiply;
```

Reflect the constants so they can be used as template arguments:

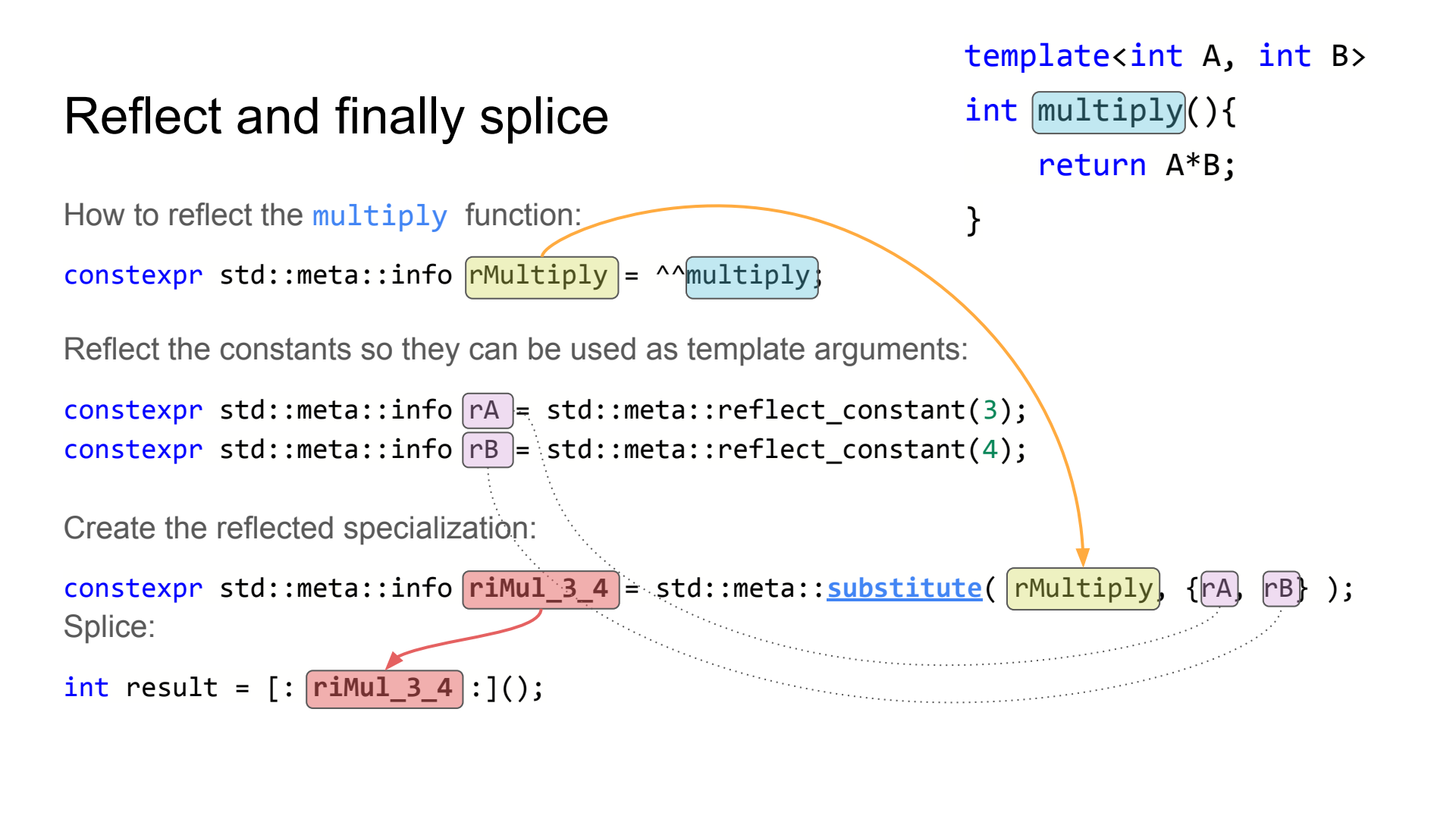
```
constexpr std::meta::info rA = std::meta::reflect_constant(3);
constexpr std::meta::info rB = std::meta::reflect_constant(4);
```

Create the reflected specialization:

```
constexpr std::meta::info riMul_3_4 = std::meta::substitute( rMultiply, {rA, rB} );
```

Splice:

```
int result = [: riMul_3_4 :]();
```



How to interpret the splice

```
int result1 = [: riMul_3_4 :] ();
```

as if

```
int result2 = multiply<3,4> ();
```

After the splice, normal C++ rules apply

```
using FnPtr = int(*)();

FnPtr fp1 = [:riMultiply:];
```

as if

```
FnPtr fp2 = multiply<3,4>;
```

Part 5 - transforming bytecode

constexpr

requests that the value of:

- a variable
- structured binding(since C++26)
- or function

is

evaluated at compile-time

enables use in constant expressions

consteval

specifies that:

- a **function**
- a **function** template

is

an *immediate function*

requires that every call to the function is evaluated at compile time.

Consteval function

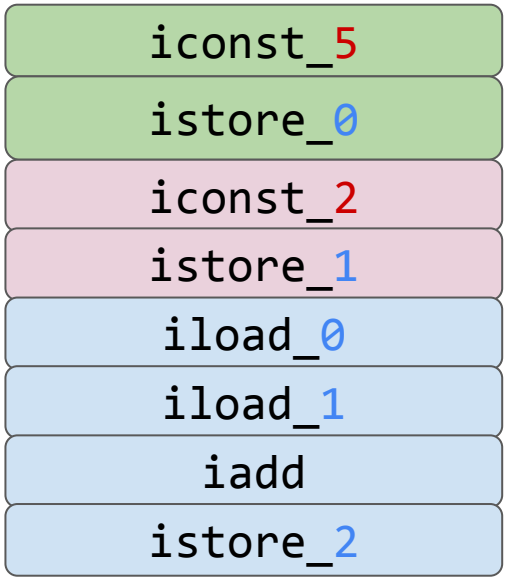
- A consteval function can use heap allocated memory.
- Must be **transient**
  - cleaned up before the constexpr finishes evaluating.

```
constexpr result = some_consteval_function();
```

- some_consteval_function can use heap allocated memory.
- this heap allocated memory must be cleaned up before storing the **result**.

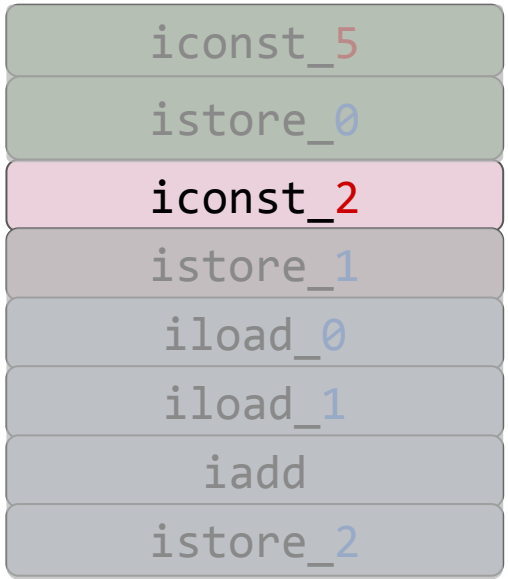
Bytecode program

Program




```
inline constexpr
std::array<std::uint8_t, 8> program {
    ICONST_5,
    ISTORE_0,
    ICONST_2,
    ISTORE_1,
    ILOAD_0,
    ILOAD_1,
    IADD,
    ISTORE_2
};
```

Program

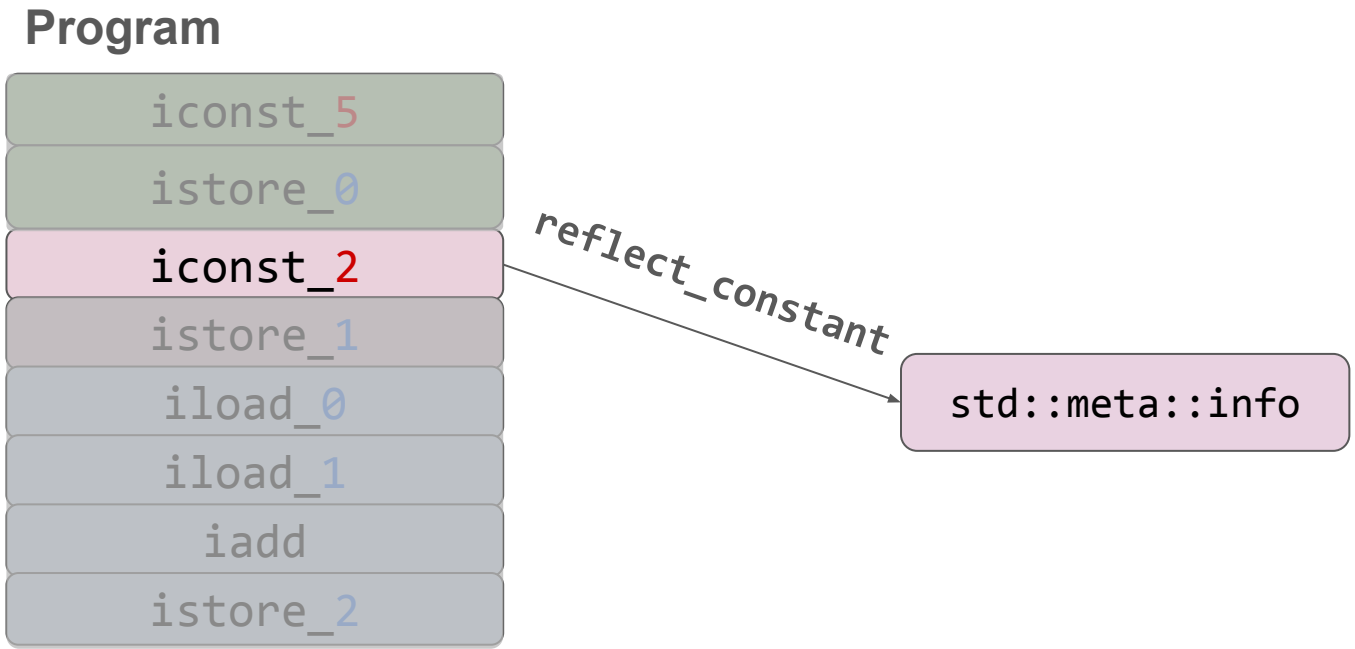


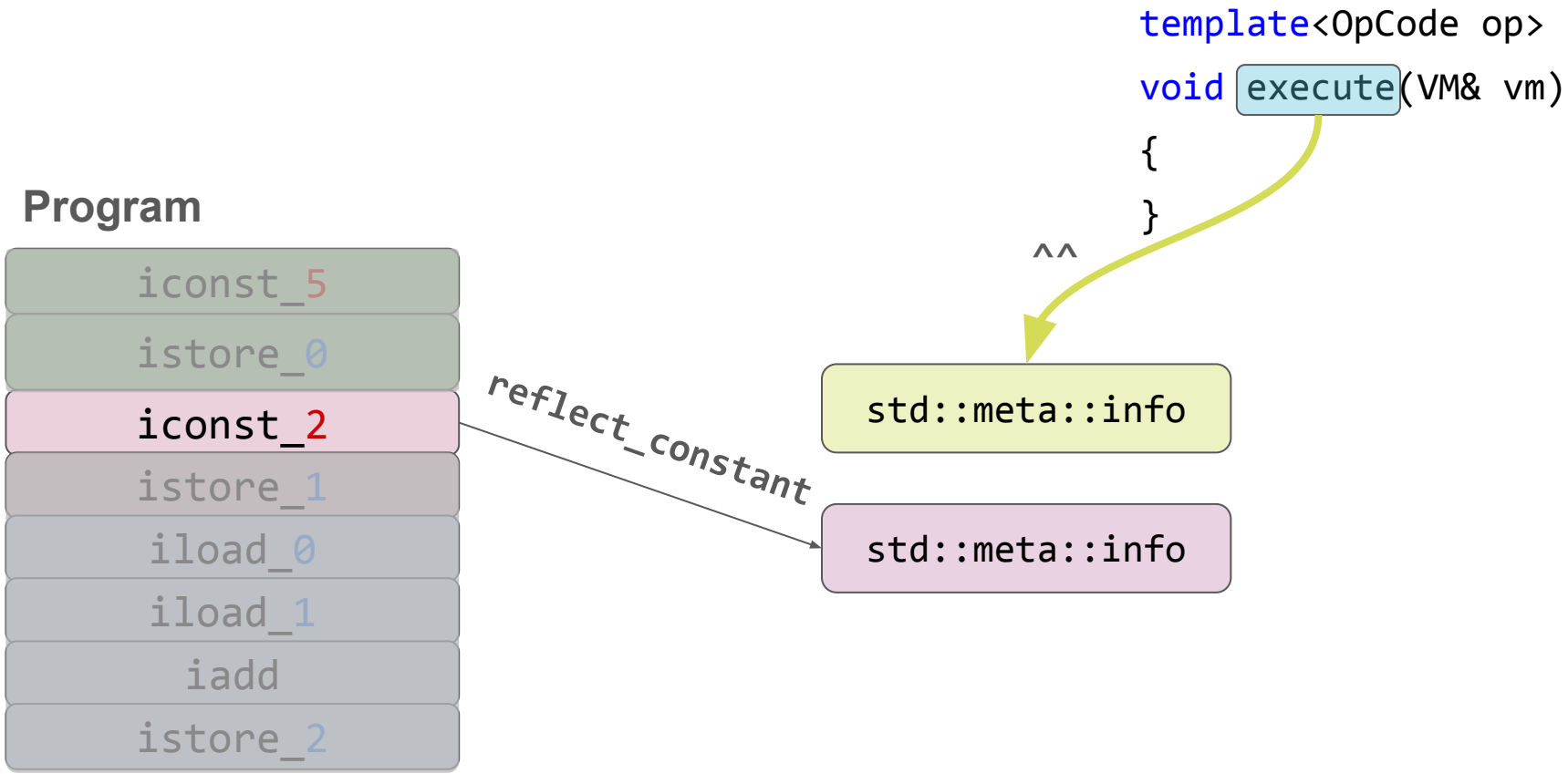

```
InstructionFn pFN = execute<ICONST_2>;
```

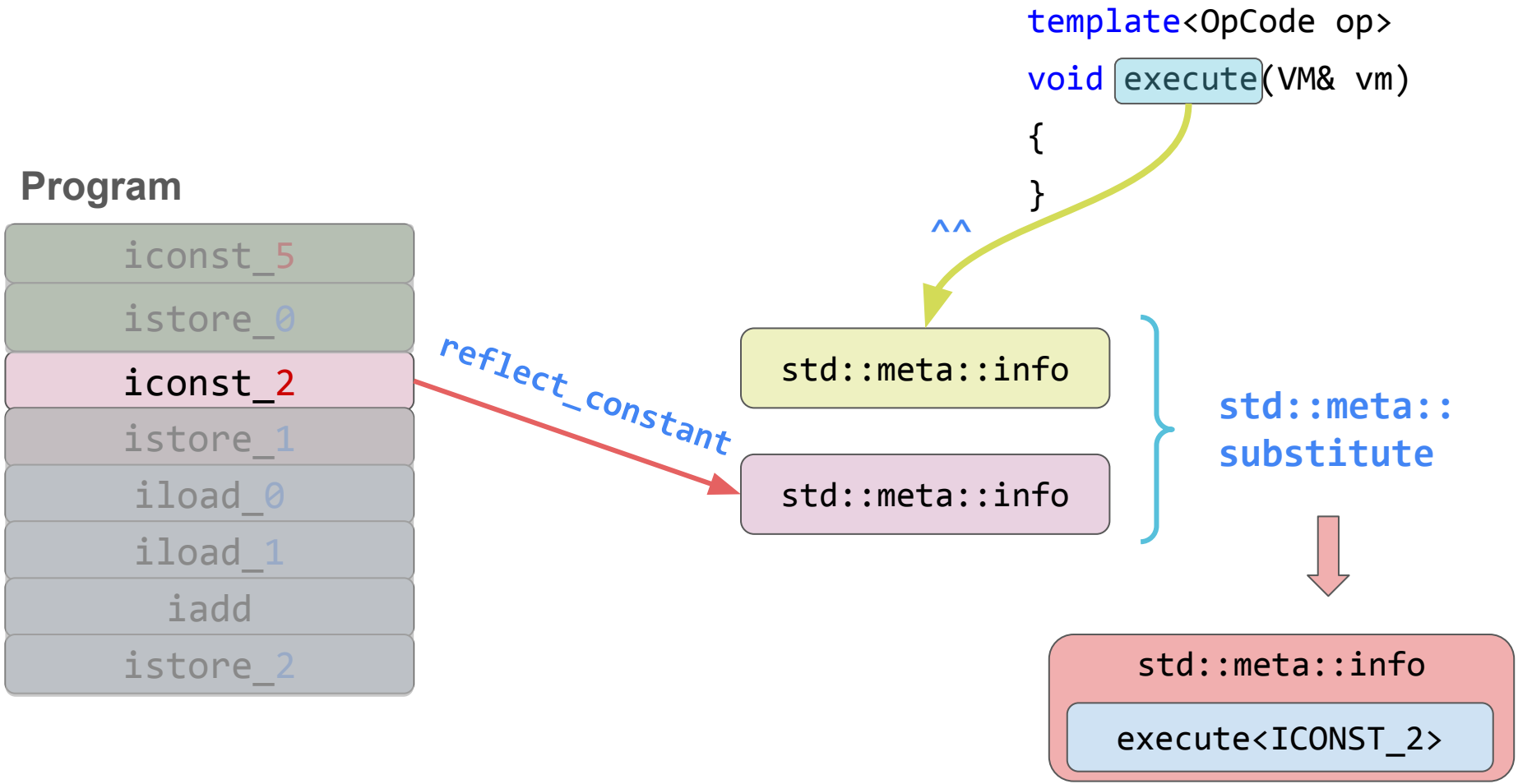
Program

```
iconst_5
istore_0
iconst_2
istore_1
iload_0
iload_1
iadd
istore_2
```

```
InstructionFn pFN = &Execute<ICONST_2>;
```


Program
iconst_5
istore_0
iconst_2
istore_1
iload_0
iload_1
iadd
istore_2
reflect_constant
std::meta::info


template<OpCode op>
void execute(VM& vm)
{
}
^^
Program
iconst_5
istore_0
iconst_2
istore_1
iload_0
iload_1
iadd
istore_2
reflect_constant
std::meta::info
std::meta::info


template<OpCode op>
void execute(VM& vm)
{
}
^^
Program
iconst_5
istore_0
iconst_2
istore_1
iload_0
iload_1
iadd
istore_2
reflect_constant
std::meta::info
std::meta::info
std::meta::
substitute
std::meta::info
execute<ICONST_2>

Reflected specialization

```
template<OpCode op>
void execute(VM& vm)
{
}
```

```
consteval std::meta::info make_execute_info(OpCode op) {
    return std::meta::substitute(
        ^^execute,
        { std::meta::reflect_constant(op) }
    );
}
```

Reflected block specialization

```
using InstructionFn = void(*)(VM&);

template<InstructionFn ... Instructions>
void block(VM& vm) {
    (Instructions(vm), ...);
}
```

Reflected block specialization

```
template<auto const& ByteCode >
consteval std::meta::info make_block_info() {




}
```

```
inline constexpr
std::array<std::uint8_t, 8> program {
    ICONST_5,
    ISTORE_0,
    ICONST_2,
    ISTORE_1,
    ILOAD_0,
    ILOAD_1,
    IADD,
    ISTORE_2
};
```

Reflected block specialization

```
template<auto const& ByteCode>
consteval std::meta::info make_block_info() {
    std::vector<std::meta::info> instructions;









}
```

Reflected block specialization

```
template<auto const& ByteCode>
consteval std::meta::info make_block_info() {
    std::vector<std::meta::info> instructions;

    for (auto byte : ByteCode) {
        auto op = static_cast<OpCode>(byte);
        instructions.push_back(make_execute_info(op));
    }



}
```

Reflected block specialization

```
template<auto const& ByteCode>
consteval std::meta::info make_block_info() {
    std::vector<std::meta::info> instructions;

    for (auto byte : ByteCode) {
        auto op = static_cast<OpCode>(byte);
        instructions.push_back(make_execute_info(op));
    }

    return std::meta::substitute(
        ^^block, instructions
    );
}
```

Execute the block

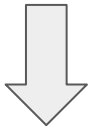
```
inline constexpr
std::array<std::uint8_t, 8> program {
    ICONST_5,
    ISTORE_0,
    ICONST_2,
    ISTORE_1,
    ILOAD_0,
    ILOAD_1,
    IADD,
    ISTORE_2
};
```

```
int main(){
    VM vm;
    [: make_block_info<program>() :](vm);
    return vm.local(2);
}
```

Execute the block

```
inline constexpr
std::array<std::uint8_t, 8> program {
    ICONST_5,
    ISTORE_0,
    ICONST_2,
    ISTORE_1,
    ILOAD_0,
    ILOAD_1,
    IADD,
    ISTORE_2
};
```

```
int main(){
    VM vm;
    [: make_block_info<program>() :](vm);
    return vm.local(2);
}
```



```
1   main:
2           mov     eax, 7
3           ret
```

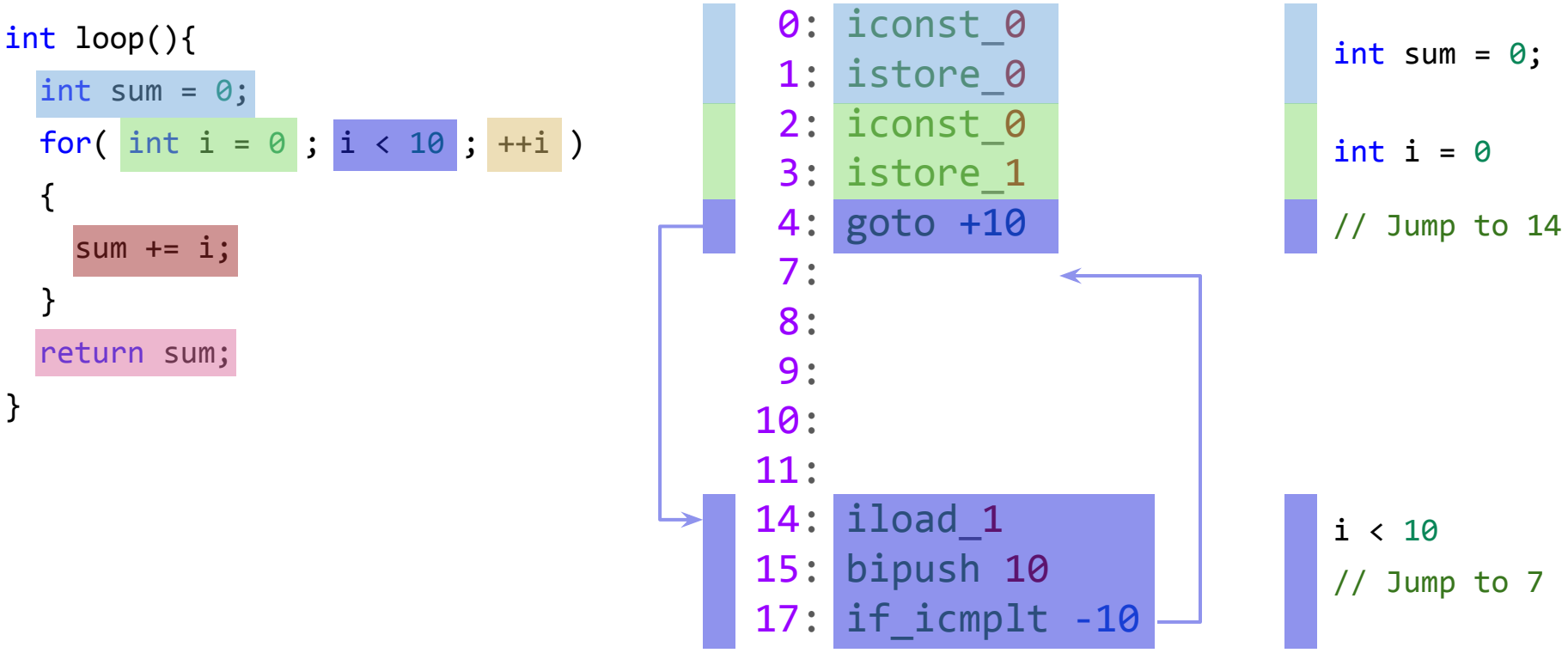
Part 6 - What about jumps and loops?

```
int loop(){
  int sum = 0;
  for( int i = 0 ; i < 10 ; ++i )
  {
    sum += i;
  }
  return sum;
}
```

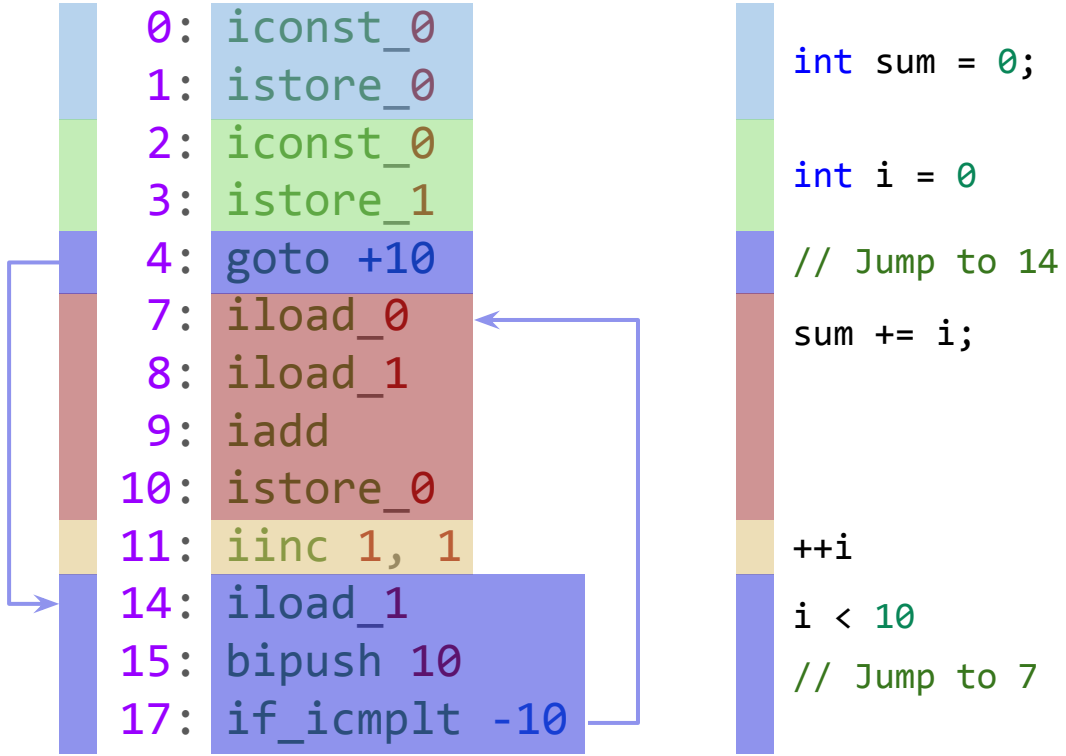
```
int loop(){
  int sum = 0;
  for( int i = 0 ; i < 10 ; ++i )
  {
    sum += i;
  }
  return sum;
}
```

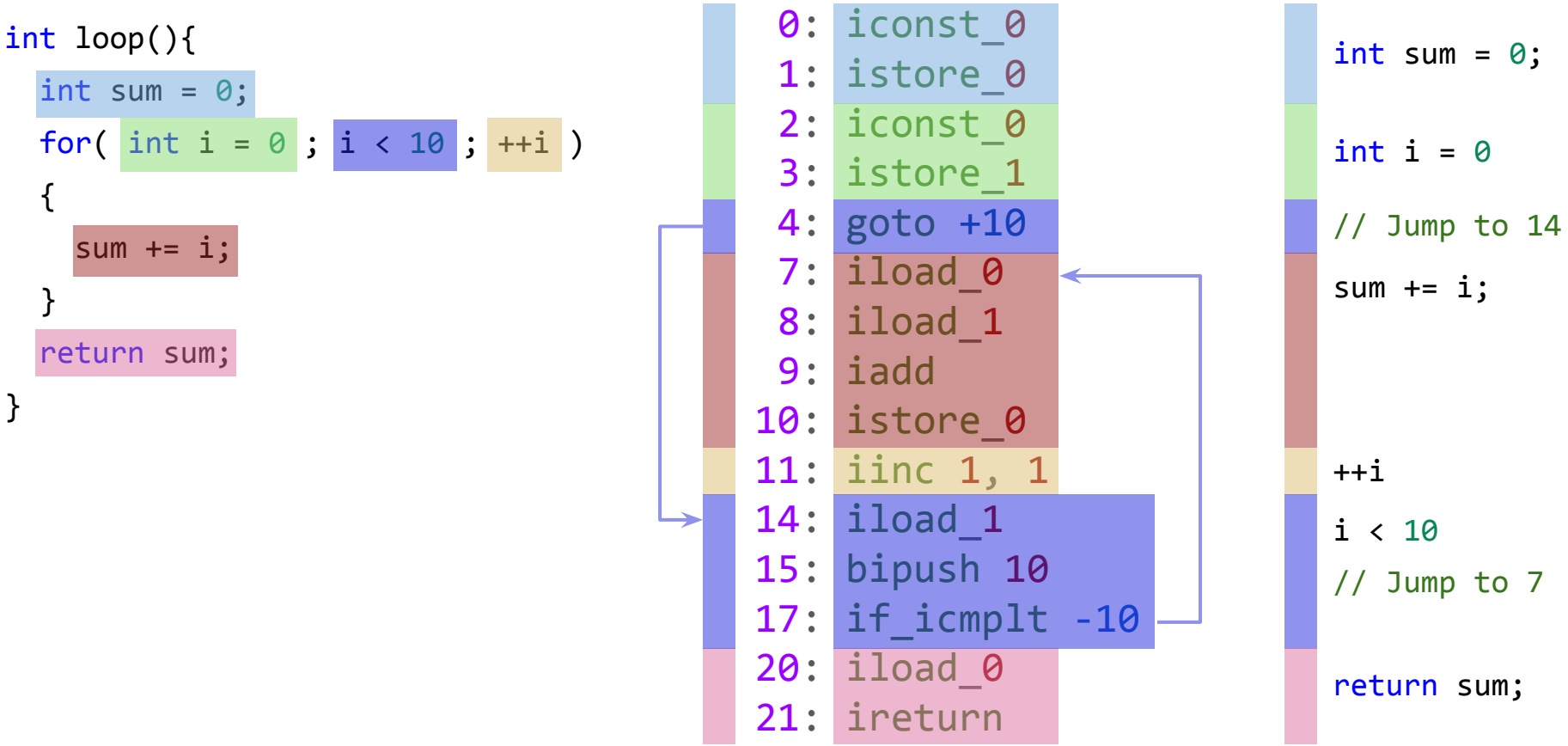
```
0: iconst_0
1: istore_0
2: iconst_0
3: istore_1
```

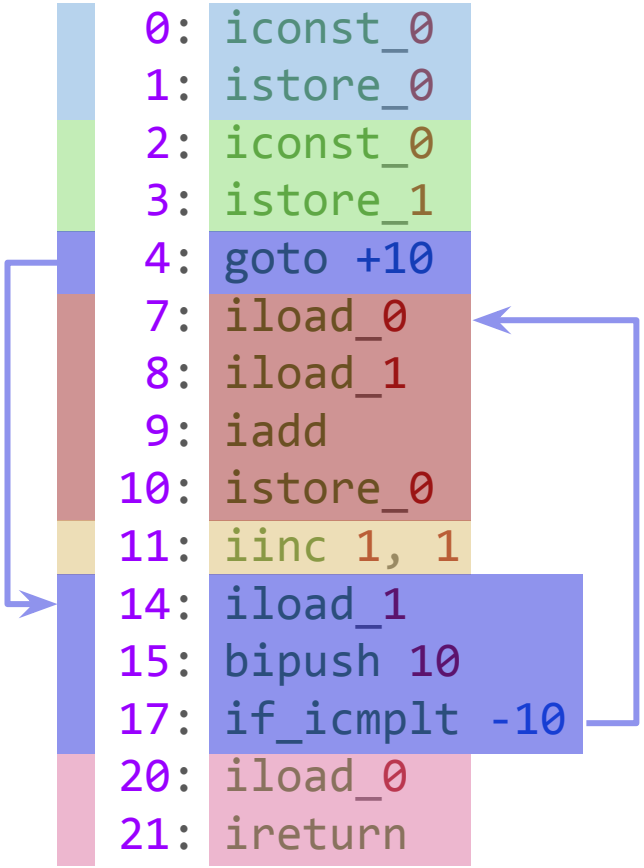
```
int sum = 0;

int i = 0
```


int loop(){
int sum = 0;
for(int i = 0 ; i < 10 ; ++i)
{
sum += i;
}
return sum;
}
0: iconst_0
1: istore_0
2: iconst_0
3: istore_1
4: goto +10
7:
8:
9:
10:
11:
14: iload_1
15: bipush 10
17: if_icmplt -10
int sum = 0;
int i = 0
// Jump to 14
i < 10
// Jump to 7

```
int loop(){
  int sum = 0;
  for( int i = 0 ; i < 10 ; ++i )
  {
    sum += i;
  }
  return sum;
}
```




int loop(){
int sum = 0;
for(int i = 0 ; i < 10 ; ++i)
{
sum += i;
}
return sum;
}
0: iconst_0
1: istore_0
2: iconst_0
3: istore_1
4: goto +10
7: iload_0
8: iload_1
9: iadd
10: istore_0
11: iinc 1, 1
14: iload_1
15: bipush 10
17: if_icmplt -10
20: iload_0
21: ireturn
int sum = 0;
int i = 0
// Jump to 14
sum += i;
++i
i < 10
// Jump to 7
return sum;




```
int main() {
    CPU cpu;
    block <
        execute<ICONST_0>, execute<ISTORE_0>,
        execute<ICONST_0>, execute<ISTORE_1>
    >(cpu);
    loop <
        icmplt<execute<ILOAD_1>, execute<BIPUSH, 10>>,
        block <
            execute<ILOAD_0>, execute<ILOAD_1>,
            execute<IADD>,
            execute<ISTORE_0>,
            execute<IINC, 1, 1>
        >
    >(cpu);
    execute<ILOAD_0>(cpu);
    execute<IRETURN>(cpu);
    return cpu.result;
}
```

```
int main() {
    CPU cpu;
    block <
        execute<ICONST_0>, execute<ISTORE_0>,
        execute<ICONST_0>, execute<ISTORE_1>
    >(cpu);
    loop <
        icmplt<execute<ILOAD_1>, execute<BIPUSH, 10>>,
        block <
            execute<ILOAD_0>, execute<ILOAD_1>,
            execute<IADD>,
            execute<ISTORE_0>,
            execute<IINC, 1, 1>
        >
    >(cpu);
    execute<ILOAD_0>(cpu);
    execute<IRETURN>(cpu);
    return cpu.result;
}
```

```
1   main:
2           mov     eax, 45
3           ret
```

Part 7 - Injecting functions

define_aggregate

```
class Incomplete;

define_aggregate( ^^Incomplete,  { /* datamembers */ });
```

Define_aggregate limitations

Only data members

Only **nonstatic** data members

→ Not possible to define functions

Define_aggregate limitations

Only data members

Only **nonstatic** data members

→ Not possible to define functions

→ Code injection is coming in C++29

Part 8 - Summary

Interpreter mode

```
inline constexpr
std::array<std::uint8_t, 8> program {
    ICONST_5,
    ISTORE_0,
    ICONST_2,
    ISTORE_1,
    ILOAD_0,
    ILOAD_1,
    IADD,
    ISTORE_2
};
```

```
Interpreter
```

```
switch( opcode ){
    // …
}
```

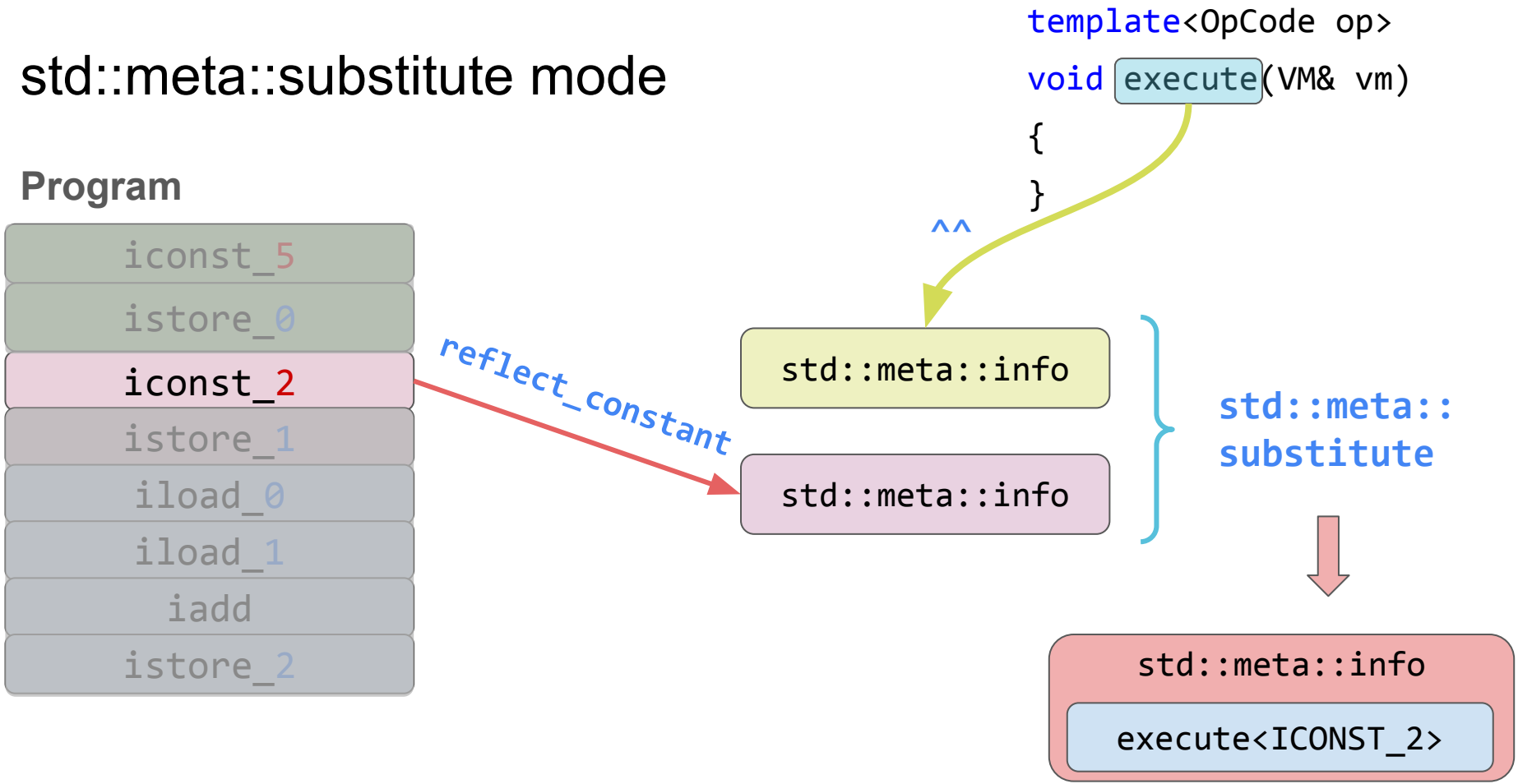
Handcrafted compile time mode

```
using InstructionFn = void(*)(VM&);
    std::array<InstructionFn,8> program = {
        execute<ICONST_5>,
        execute<ISTORE_0>,
        execute<ICONST_2>,
        execute<ISTORE_1>,
        execute<ILOAD_0>,
        execute<ILOAD_1>,
        execute<IADD>,
        execute<ISTORE_2>
    };
```

Compile time

if constexpr

std::meta::substitute mode



Conclusions

Future work


Java Class File Viewer
version 1.0 - Koen Samyn - 2026
Class Header
class: LoopTest
super: java/lang/Object
version: 70.0
access: super
magic: 0xCAFEBABE
Methods
name descriptor stack locals
<init> ()V 1 1
testFor ()V 2 3
sum (II)I 2 3
sum (III)I 2 4
Constant Pool
kind value
1 Ref class 2
2 Class java/lang/Object
3 Name and type <init>()V
4 Utf8 java/lang/Object
5 Utf8 <init>
6 Utf8 ()V
7 Ref class 8
8 Class LoopTest
9 Name and type offsetI
10 Utf8 LoopTest
11 Utf8 offset
12 Utf8 I
testFor
pc bytecode
0 iconst_0
1 istore_1
2 iconst_0
3 istore_2
4 iload_2
5 bipush 10
7 if_icmpge 20
10 iload_1
11 iload_2
12 iadd
13 istore_1
14 iinc 2, 1
17 goto 4
Local variables
value
0 this
1
2
Stack
value
0
1
Use up and down arrow keys to select a method.