

+ 24

# C++ Under the Hood

## Internal Class Mechanisms

CHRIS RYAN



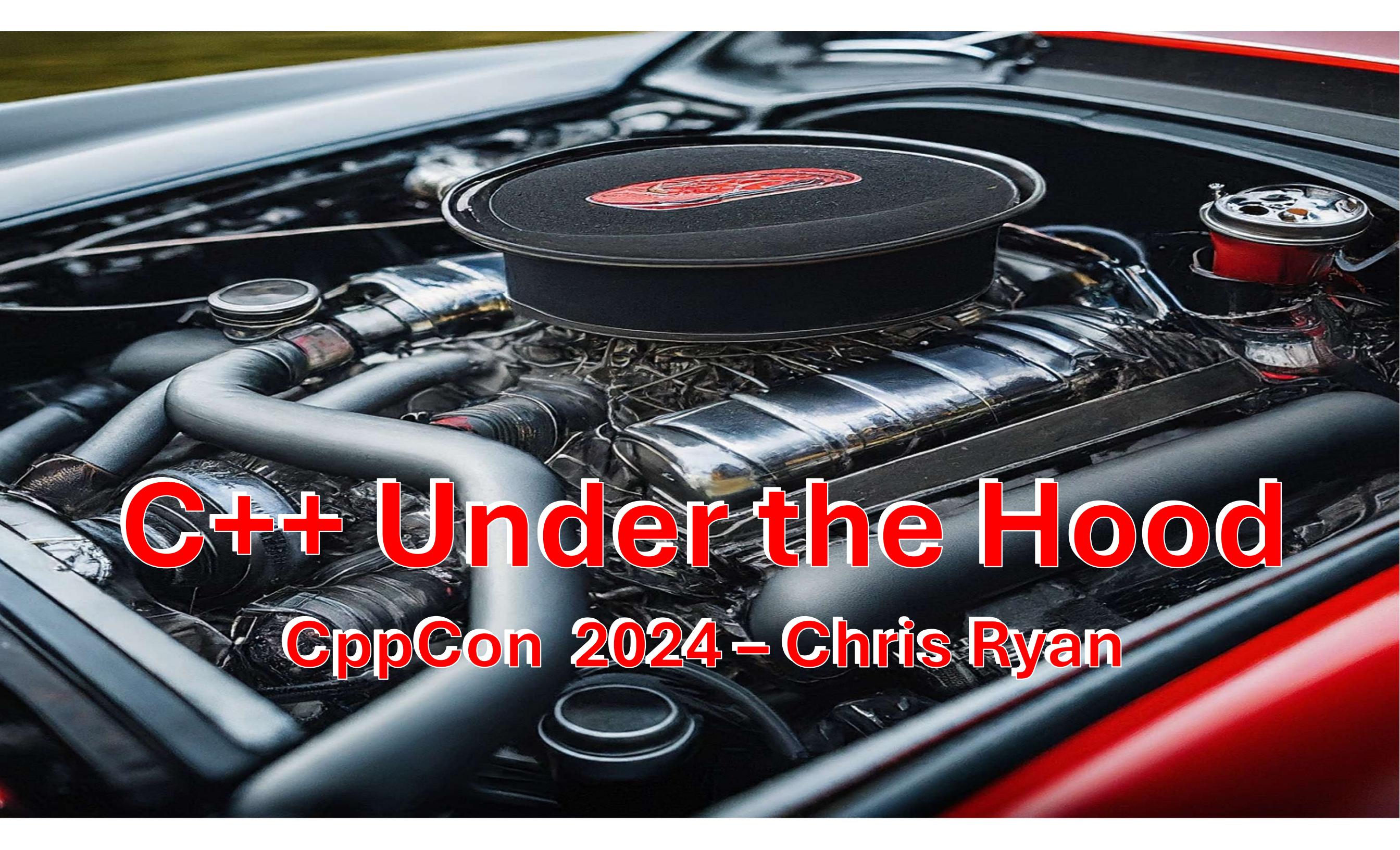
**Cppcon**

The C++ Conference

20  
24



September 15 - 20



# C++ Under the Hood

CppCon 2024 – Chris Ryan

# C++ Under the Hood: (Internal Class Mechanisms)

I find it much easier to use something correctly,  
if I can understand what it is doing inside and why.

We are going to talk about internal C++ class mechanisms:

- C++ Onion, inheritance and polymorphic mechanisms,
- Member Data Pointers,
- Member Function Pointers,
- \* Stack Frame / Base Pointer mechanics
- \* Construction/Destruction order,
- \* Running code before and after main(),

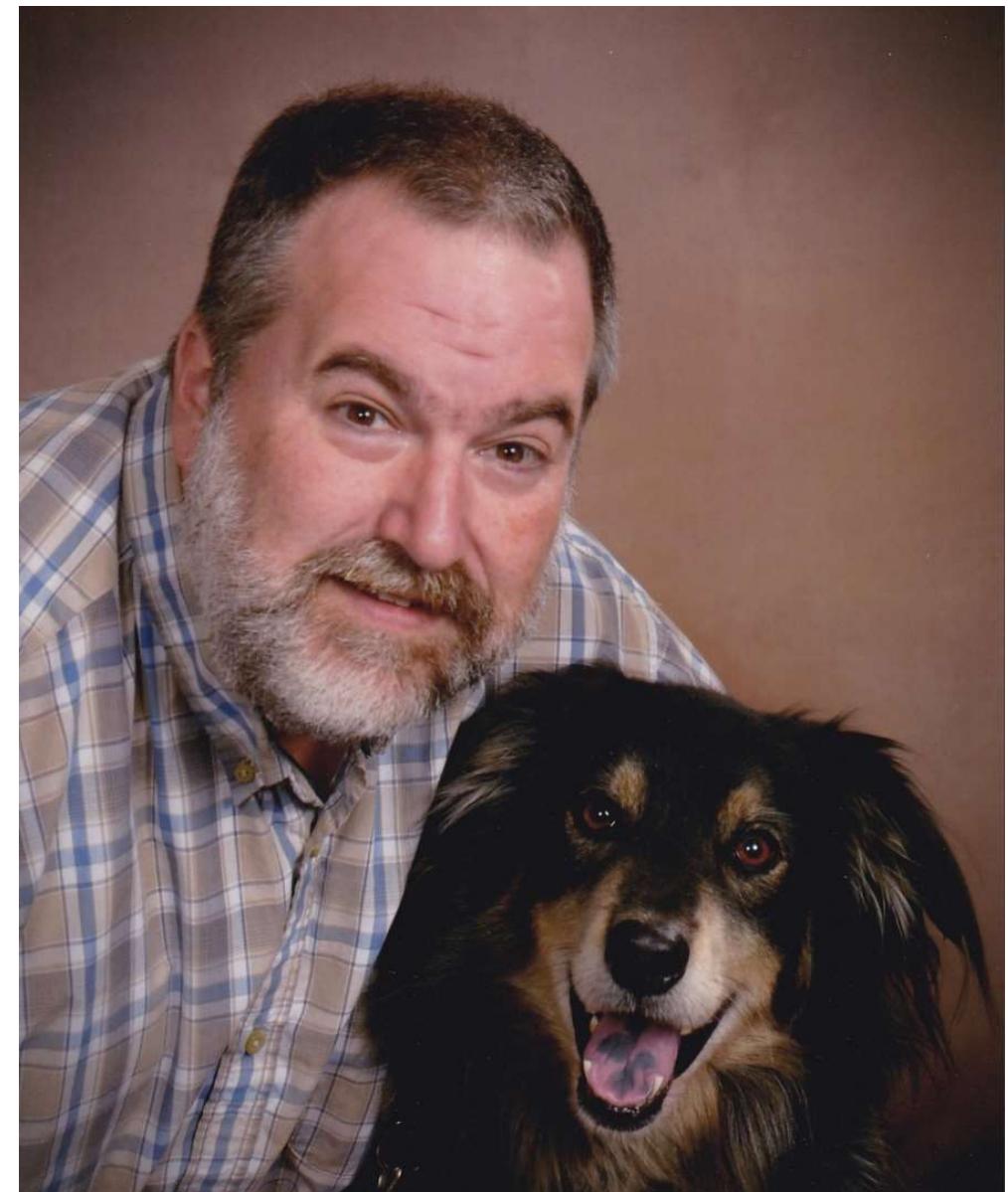
# C++ Under the Hood: (Internal Class Mechanisms)

About Me: Chris Ryan, Bellevue, Washington, USA

[Linkedin.com/in/chrisr98008](https://www.linkedin.com/in/chrisr98008)

(No blind invites! Tell me how we know each other)

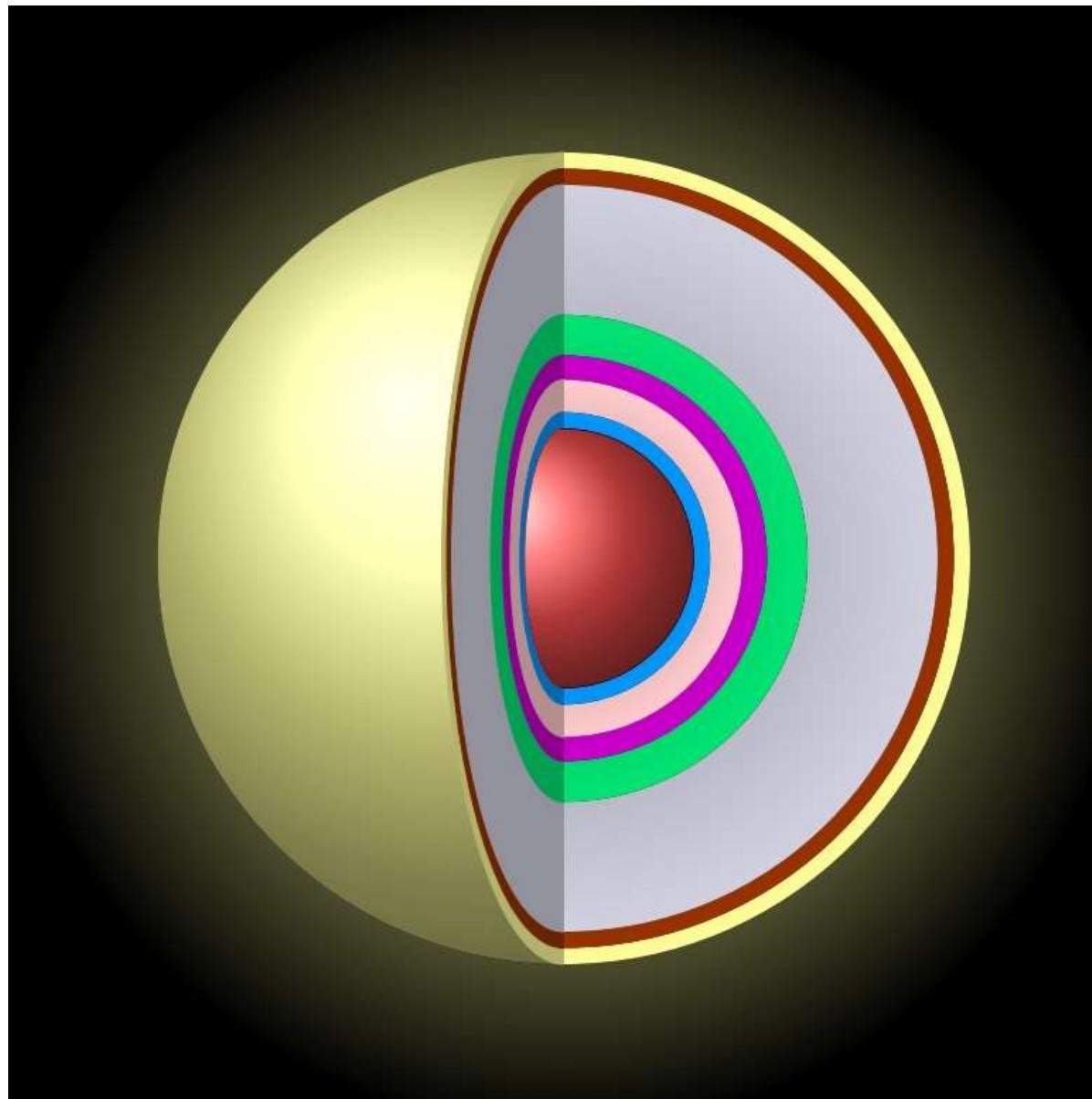
- Classically trained in Hardware and Software Engineering,
- Well experienced in Modern C++ (& Classic ‘C’ \*\*\*),
- ISO C++ Standards Committee member/WG21, EWG.
- Extremely Large Scale/Complex Problem Spaces,
- (on hiatus from\*\*\*) Embedded/Firmware (large & small),
- Prefer Windows, because that is what I know,
- Win32/64 SDK/MFC, \*\*\*Embedded/Firmware, Bare Metal, Linux
- Worked in many domains:
  - Number Crunching
  - Business / Internet
  - Networking / Protocols
  - Operating System Internals
  - Data Engines
  - Scientific / Medical
  - Engineering / Robotics
- **I believe in reducing complexity through simplification**



# C++ Under the Hood – The Onion



# C++ Under the Hood – The Onion



# C++ Under the Hood – The Onion



# C++ Under the Hood – The Onion (abused inheritance, low cohesion)



# C++ Under the Hood – Inheritance vs. Aggregation / Composite

## Inheritance

- **Is-a**

## Aggregation / Composite

- **Has-a**

A Person **has a** Name

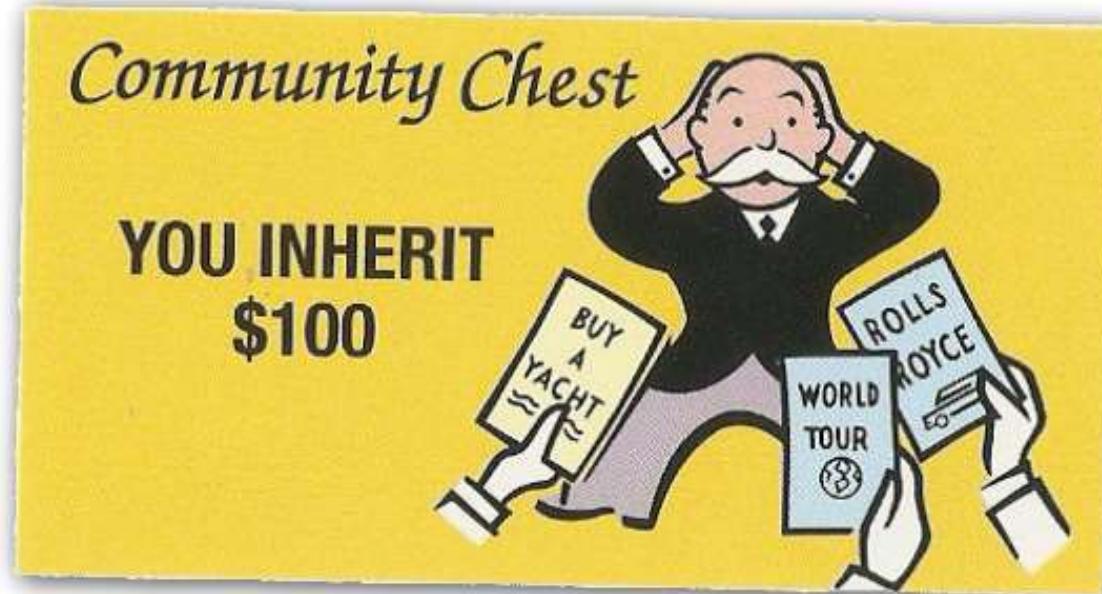
A Person **is NOT a** Name

An Employee **is a** Person

```
struct Person
{
    string name;
    date    dob;
    // ...
};

struct Employee : Person
{
    uint    id;
    // ...
};
```

# C++ Under the Hood – Inheritance & Construction



# C++ Under the Hood – Inheritance Hierarchy

```
struct A {
    int a{};    string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};    string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};    string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};
```

# C++ Under the Hood – \_\_v\_table\_ptr

```
struct A {
    hidden v_table* __v_table_ptr;
    int a{};    string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};    string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};    string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};
```

# C++ Under the Hood – \_\_v\_table\_ptr / v\_table

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }
    hidden v_table* __v_table_ptr;
    int a{};      string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }
    int b{};      string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }
    int c{};      string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};
```

# C++ Under the Hood – C'tor (Constructor Implementation)

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }                      A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }                      B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                             static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }                      C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

# C++ Under the Hood – D'tor (Destructor Implementation)

```
struct A {
    hidden v_table* __v_table_ptr;
    int a{};    string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};    string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};    string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};

static A::'v_table'[] = { A::~A(), A::Foo() }
A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()

static B::'v_table'[] = { B::~B(), B::Foo() }
B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()

static C::'v_table'[] = { C::~C(), C::Foo() }
C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Construction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```



# C++ Under the Hood – Construction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                               static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

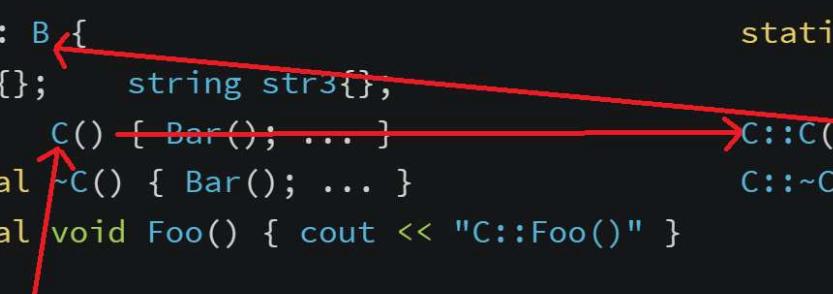


# C++ Under the Hood – Construction (construct base members)

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }
    hidden v_table* __v_table_ptr;
    int a{};      string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }
    int b{};      string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {                                static C::'v_table'[] = { C::~C(), C::Foo() }
    int c{};      string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};
```



# C++ Under the Hood – Construction (construct base members)

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }
    hidden v_table* __v_table_ptr;
    int a{};      string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }
    int b{};      string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {                           static C::'v_table'[] = { C::~C(), C::Foo() }
    int c{};      string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};
```

The diagram illustrates the construction of virtual tables (v-tables) for classes A, B, and C. Red arrows point from the inheritance lines in the C code to the corresponding constructor definitions in the B and A v-tables.

- Class A:** Contains a hidden v\_table pointer, a constructor A(), a destructor ~A(), and a virtual Foo() method.
- Class B:** Inherits from A. It contains an constructor B(), a destructor ~B(), and a virtual Foo() method.
- Class C:** Inherits from B. It contains an constructor C(), a destructor ~C(), and a virtual Foo() method.

Each class has a static v-table array defined at the top of its code block. The arrays contain the addresses of the class's own constructor (~A(), ~B(), ~C()) and its own Foo() method. The B and A v-tables also include the addresses of their respective constructors and destructors.

# C++ Under the Hood – Construction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }
    hidden v_table* __v_table_ptr;
    int a{};      string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }
    int b{};      string str2{};
    B() { Bar(); ... } → B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {                                static C::'v_table'[] = { C::~C(), C::Foo() }
    int c{};      string str3{};
    C() { Bar(); ... } → C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};
```

The diagram illustrates the construction of v-tables for classes A, B, and C. Each class has a static v-table array containing its own destructor and a pointer to the v-table of its base class. Red arrows and lines highlight the inheritance relationship and the placement of constructor definitions in the v-tables.

# C++ Under the Hood – Construction (construct base members)

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }
    hidden v_table* __v_table_ptr;
    int a{};      string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {                                static B::'v_table'[] = { B::~B(), B::Foo() }
    int b{};      string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {                                 static C::'v_table'[] = { C::~C(), C::Foo() }
    int c{};      string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};
```

The diagram illustrates the construction of base members for classes A, B, and C. Red arrows point from the base class definitions to their respective constructor definitions in the code above. Specifically, an arrow points from the B definition to the B::B() constructor, and another arrow points from the C definition to the C::C() constructor.

# C++ Under the Hood – Construction (construct base members)

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }
    hidden v_table* __v_table_ptr;
    int a{};      string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }
    int b{};      string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {                               static C::'v_table'[] = { C::~C(), C::Foo() }
    int c{};      string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};
```

The diagram illustrates the construction of base members for classes A, B, and C. Red arrows point from the base class names (A, B, C) to their respective constructor definitions. Red lines also connect the base class names to the derived class names.

# C++ Under the Hood – Construction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};      string str1{};  
    A() { Bar(); ... } → A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }  
    virtual ~A() { Bar(); ... }                 A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};      string str2{};  
    B() { Bar(); ... } → B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }  
    virtual ~B() { Bar(); ... }                 B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                               static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};      string str3{};  
    C() { Bar(); ... } → C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }  
    virtual ~C() { Bar(); ... }                 C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

# C++ Under the Hood – Construction (construct members)

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};     string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};     string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};     string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

The code illustrates the construction of objects and their v-tables. It defines three classes: A, B, and C. Each class has a hidden v\_table pointer and a constructor that calls Bar(). The destructor for each class also calls Bar(). The Foo() method is virtual in all three classes. The v-tables are defined as arrays containing the addresses of the dtor and Foo methods. The dtor entries include the base class pointer (A::~A(), B::~B(), or C::~C()) and the address of the dtor itself. The Foo() entries are the addresses of the Foo() methods in each class.

# C++ Under the Hood – Construction (set this v\_table)

```
struct A {
    int a{};      string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};      string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};      string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};

static A::'v_table'[] = { A::~A(), A::Foo() }
A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()

static B::'v_table'[] = { B::~B(), B::Foo() }
B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()

static C::'v_table'[] = { C::~C(), C::Foo() }
C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Polymorphism During Construction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

# C++ Under the Hood – Polymorphism During Construction

```
struct A {
    hidden v_table* __v_table_ptr;
    int a{};    string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};    string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};    string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};

static A::'v_table'[] = { A::~A(), A::Foo() }
A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()

static B::'v_table'[] = { B::~B(), B::Foo() }
B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()

static C::'v_table'[] = { C::~C(), C::Foo() }
C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Polymorphism During Construction

```
struct A {
    hidden v_table* __v_table_ptr;
    int a{};    string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};    string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};    string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};

static A::'v_table'[] = { A::~A(), A::Foo() }

A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()

static B::'v_table'[] = { B::~B(), B::Foo() }

B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()

static C::'v_table'[] = { C::~C(), C::Foo() }

C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Polymorphism During Construction

```
struct A {
    hidden v_table* __v_table_ptr;
    int a{};    string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};    string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};    string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};

static A::'v_table'[] = { A::~A(), A::Foo() }

A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()

static B::'v_table'[] = { B::~B(), B::Foo() }

B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()

static C::'v_table'[] = { C::~C(), C::Foo() }

C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Construction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

# C++ Under the Hood – Construction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};      string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};      string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};      string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};  
  
A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }  
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()  
  
B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }  
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()  
  
C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }  
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Construction (construct members)

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                               static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

# C++ Under the Hood – Construction (set this v\_table)

```
struct A {
    hidden v_table* __v_table_ptr;
    int a{};    string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};    string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};    string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};

static A::'v_table'[] = { A::~A(), A::Foo() }

A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()

static B::'v_table'[] = { B::~B(), B::Foo() }

B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()

static C::'v_table'[] = { C::~C(), C::Foo() }

C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Polymorphism During Construction

```
struct A {
    hidden v_table* __v_table_ptr;
    int a{};    string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};    string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};    string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};

static A::'v_table'[] = { A::~A(), A::Foo() }
A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()

static B::'v_table'[] = { B::~B(), B::Foo() }
B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()

static C::'v_table'[] = { C::~C(), C::Foo() }
C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Construction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                               static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

# C++ Under the Hood – Construction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }
    hidden v_table* __v_table_ptr;
    int a{};      string str1{};
    A() { Bar(); ... }                         A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
    virtual ~A() { Bar(); ... }                 A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }

};

struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }
    int b{};      string str2{};
    B() { Bar(); ... }                         B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
    virtual ~B() { Bar(); ... }                B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()
    virtual void Foo() { cout << "B::Foo()" }

};

struct C : B {                             static C::'v_table'[] = { C::~C(), C::Foo() }
    int c{};      string str3{};
    C() { Bar(); ... }                        C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
    virtual ~C() { Bar(); ... }               C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
    virtual void Foo() { cout << "C::Foo()" }

};
```

# C++ Under the Hood – Construction (construct members)

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

# C++ Under the Hood – Construction (set this v\_table)

```
struct A {
    hidden v_table* __v_table_ptr;
    int a{};    string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};    string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};    string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};

static A::'v_table'[] = { A::~A(), A::Foo() }

A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()

static B::'v_table'[] = { B::~B(), B::Foo() }

B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()

static C::'v_table'[] = { C::~C(), C::Foo() }

C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Polymorphism During Construction

```
struct A {
    hidden v_table* __v_table_ptr;
    int a{};    string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};    string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};    string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};

static A::'v_table'[] = { A::~A(), A::Foo() }

A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()

static B::'v_table'[] = { B::~B(), B::Foo() }

B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()

static C::'v_table'[] = { C::~C(), C::Foo() }

C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Construction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

# C++ Under the Hood – Construction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                               static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

# C++ Under the Hood – Fully Constructed

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};  
  
A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }  
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()  
  
B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }  
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()  
  
C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }  
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Destruction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```



# C++ Under the Hood – Destruction (set this v\_table)

```
struct A {
    hidden v_table* __v_table_ptr;
    int a{};    string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};    string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};    string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};

static A::'v_table'[] = { A::~A(), A::Foo() }

A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()

static B::'v_table'[] = { B::~B(), B::Foo() }

B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()

static C::'v_table'[] = { C::~C(), C::Foo() }

C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Polymorphism During Destruction

```
struct A {
    hidden v_table* __v_table_ptr;
    int a{};    string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};    string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};    string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};

static A::'v_table'[] = { A::~A(), A::Foo() }

A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()

static B::'v_table'[] = { B::~B(), B::Foo() }

B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()

static C::'v_table'[] = { C::~C(), C::Foo() }

C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Destruction (destruct members)

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }
    hidden v_table* __v_table_ptr;
    int a{};      string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }
    int b{};      string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

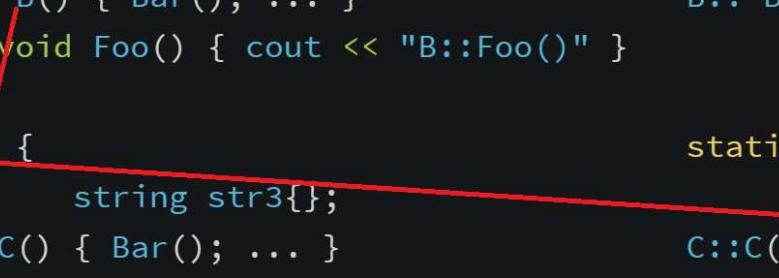
struct C : B {                               static C::'v_table'[] = { C::~C(), C::Foo() }
    int c{};      string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};
```

# C++ Under the Hood – Destruction (destruct base members)

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                               static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

# C++ Under the Hood – Destruction (destruct base members)

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```



# C++ Under the Hood – Destruction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }
    hidden v_table* __v_table_ptr;
    int a{};      string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }
    int b{};      string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {                                static C::'v_table'[] = { C::~C(), C::Foo() }
    int c{};      string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};
```

# C++ Under the Hood – Destruction (set this v\_table)

```
struct A {
    hidden v_table* __v_table_ptr;
    int a{};    string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};    string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};    string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};

static A::'v_table'[] = { A::~A(), A::Foo() }

A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()

static B::'v_table'[] = { B::~B(), B::Foo() }

B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()

static C::'v_table'[] = { C::~C(), C::Foo() }

C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Polymorphism During Destruction

```
struct A {
    hidden v_table* __v_table_ptr;
    int a{};    string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};    string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};    string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};

static A::'v_table'[] = { A::~A(), A::Foo() }
A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()

static B::'v_table'[] = { B::~B(), B::Foo() }
B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()

static C::'v_table'[] = { C::~C(), C::Foo() }
C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Destruction (destruct members)

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }
    hidden v_table* __v_table_ptr;
    int a{};      string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }
    int b{};      string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }
    int c{};      string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};
```

# C++ Under the Hood – Destruction (destruct base members)

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

The code illustrates the destruction of base class members when an object of derived class C is destroyed. The destructor for C (~C()) is shown to call the destructors of its base classes B (~B()) and A (~A()), along with its own destructor (~C()). The v-table pointers for each class point to their respective v-tables, which contain the addresses of these member destruction functions.

# C++ Under the Hood – Destruction (destruct base members)

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }
    hidden v_table* __v_table_ptr;
    int a{};      string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }
    int b{};      string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }
    int c{};      string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};
```

A red line highlights the inheritance relationship from class B to class A, starting from the colon in the B definition and extending to the A definition.

# C++ Under the Hood – Destruction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }
    hidden v_table* __v_table_ptr;
    int a{};      string str1{};
    A() { Bar(); ... }                         A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
    virtual ~A() { Bar(); ... }                 A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {                               static B::'v_table'[] = { B::~B(), B::Foo() }
    int b{};      string str2{};
    B() { Bar(); ... }                         B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
    virtual ~B() { Bar(); ... }                B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {                             static C::'v_table'[] = { C::~C(), C::Foo() }
    int c{};      string str3{};
    C() { Bar(); ... }                        C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
    virtual ~C() { Bar(); ... }               C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
    virtual void Foo() { cout << "C::Foo()" }
};
```

# C++ Under the Hood – Destruction (set this v\_table)

```
struct A {
    int a{};      string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }

};

struct B : A {
    int b{};      string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }

};

struct C : B {
    int c{};      string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }

};

static A::'v_table'[] = { A::~A(), A::Foo() }
A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()

static B::'v_table'[] = { B::~B(), B::Foo() }
B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()

static C::'v_table'[] = { C::~C(), C::Foo() }
C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Polymorphism During Destruction

```
struct A {
    hidden v_table* __v_table_ptr;
    int a{};    string str1{};
    A() { Bar(); ... }
    virtual ~A() { Bar(); ... }
    virtual void Foo() { cout << "A::Foo()" }
    void Bar() { Foo(); }
};

struct B : A {
    int b{};    string str2{};
    B() { Bar(); ... }
    virtual ~B() { Bar(); ... }
    virtual void Foo() { cout << "B::Foo()" }
};

struct C : B {
    int c{};    string str3{};
    C() { Bar(); ... }
    virtual ~C() { Bar(); ... }
    virtual void Foo() { cout << "C::Foo()" }
};

static A::'v_table'[] = { A::~A(), A::Foo() }

A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()

static B::'v_table'[] = { B::~B(), B::Foo() }

B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()

static C::'v_table'[] = { C::~C(), C::Foo() }

C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood – Destruction (destruct members)

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

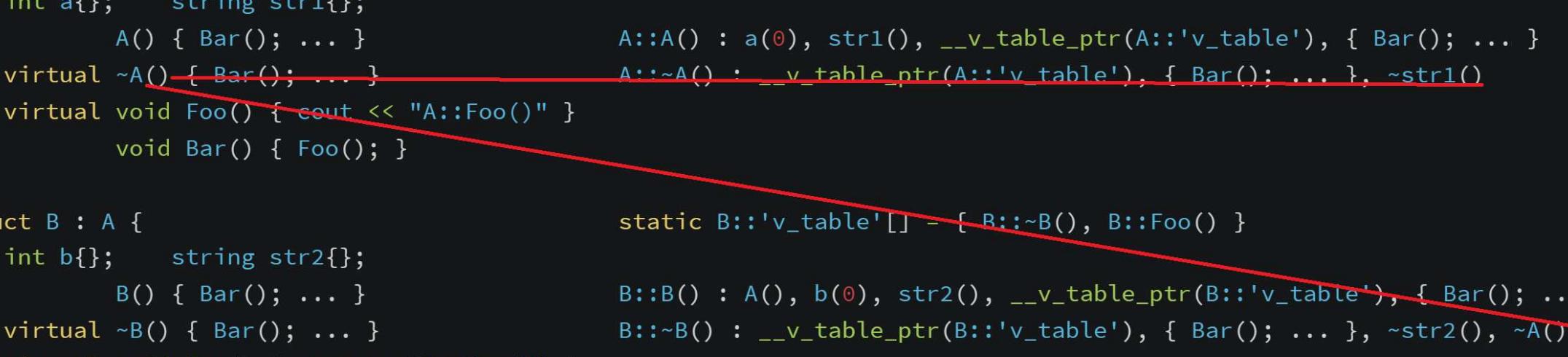
The code illustrates the v-tables and their destruction members for classes A, B, and C. Each class has a static v-table array containing its dtor and Foo member function. The dtor is a copy of the base class's dtor, and the Foo member function is a copy of the base class's Foo function. The v-table pointer is also present in each class.

# C++ Under the Hood – Destruction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

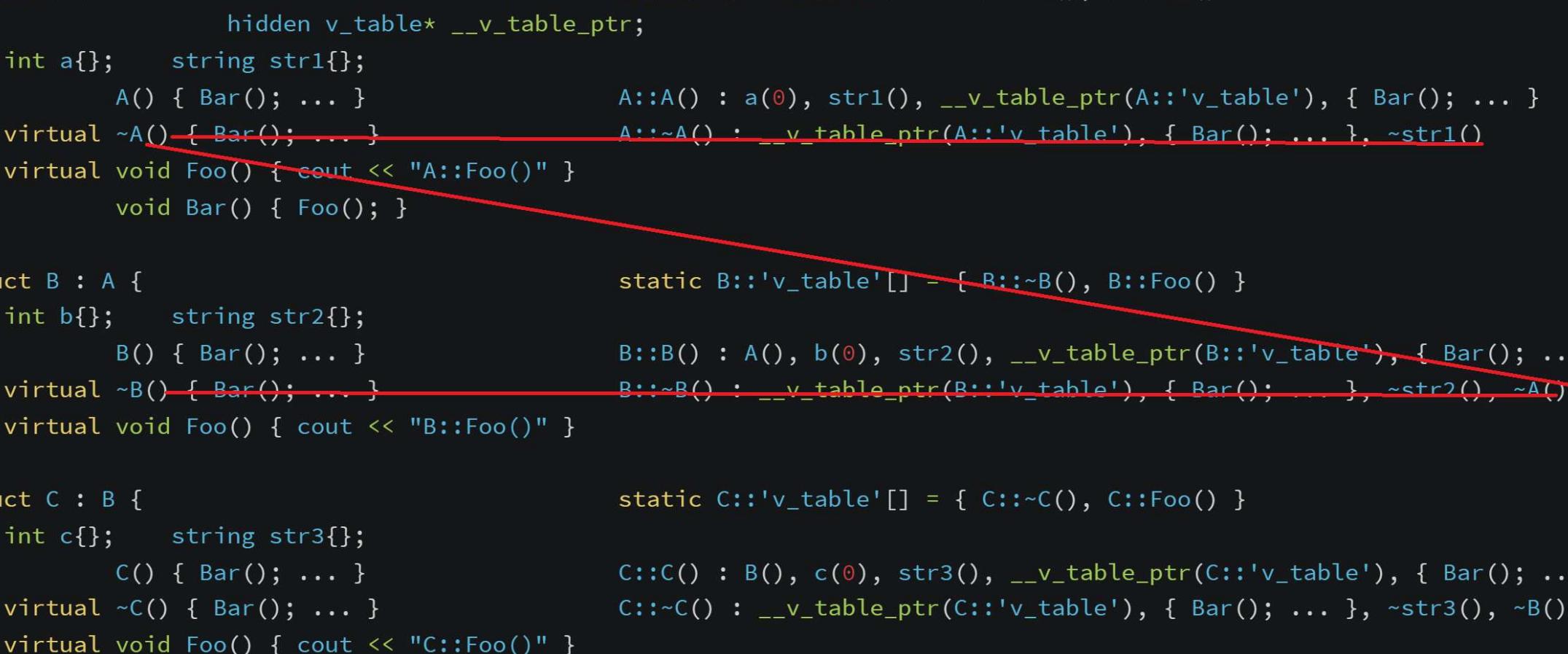
# C++ Under the Hood – Destruction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```



# C++ Under the Hood – Destruction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```



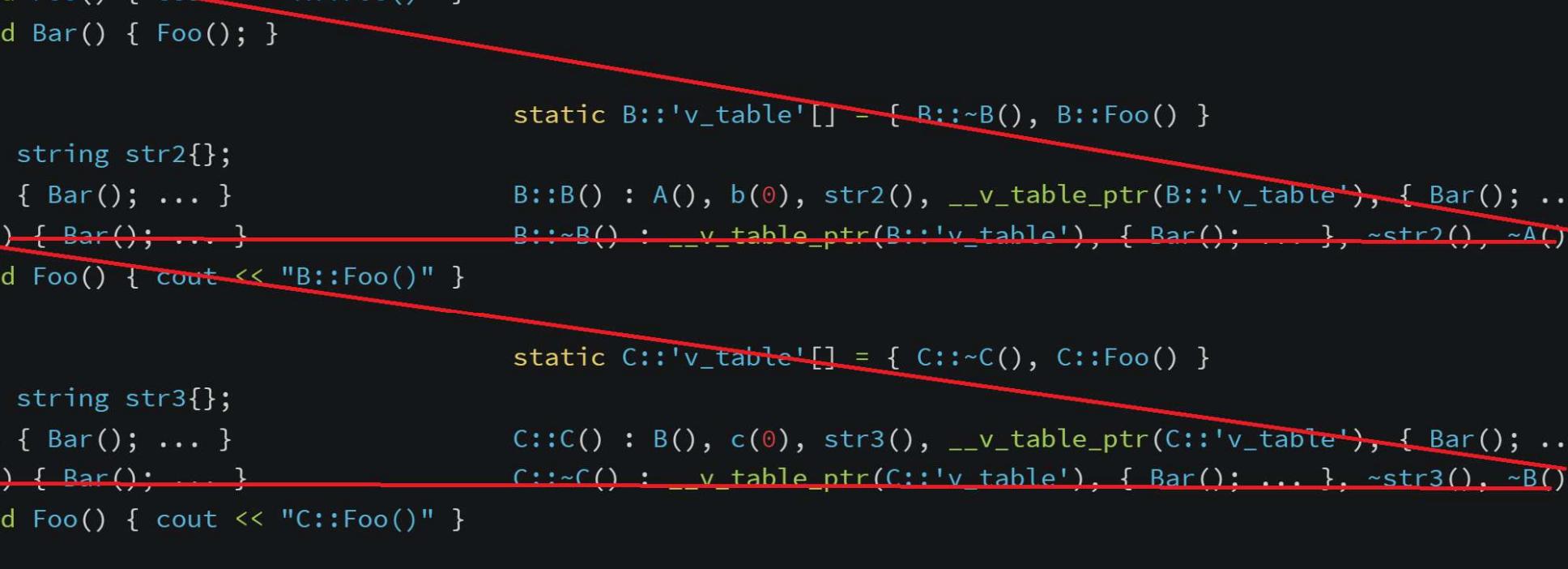
# C++ Under the Hood – Destruction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```

The diagram illustrates the inheritance of v-table pointers in C++. It shows three classes: A, B, and C. Each class has its own v-table pointer (labeled `__v_table_ptr`) which points to a static v-table array. The v-table arrays for A, B, and C contain the addresses of their respective dtor and Foo methods. Red arrows point from the `__v_table_ptr` of each class to the `__v_table_ptr` of its base class. The base class `__v_table_ptr`s are crossed out with red lines.

# C++ Under the Hood – Destruction

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};
```



# C++ Under the Hood – Fully Destructed

```
struct A {                                     static A::'v_table'[] = { A::~A(), A::Foo() }  
    hidden v_table* __v_table_ptr;  
    int a{};    string str1{};  
    A() { Bar(); ... }  
    virtual ~A() { Bar(); ... }  
    virtual void Foo() { cout << "A::Foo()" }  
    void Bar() { Foo(); }  
};  
struct B : A {                                     static B::'v_table'[] = { B::~B(), B::Foo() }  
    int b{};    string str2{};  
    B() { Bar(); ... }  
    virtual ~B() { Bar(); ... }  
    virtual void Foo() { cout << "B::Foo()" }  
};  
struct C : B {                                     static C::'v_table'[] = { C::~C(), C::Foo() }  
    int c{};    string str3{};  
    C() { Bar(); ... }  
    virtual ~C() { Bar(); ... }  
    virtual void Foo() { cout << "C::Foo()" }  
};  
  
A::A() : a(0), str1(), __v_table_ptr(A::'v_table'), { Bar(); ... }  
A::~A() : __v_table_ptr(A::'v_table'), { Bar(); ... }, ~str1()  
  
B::B() : A(), b(0), str2(), __v_table_ptr(B::'v_table'), { Bar(); ... }  
B::~B() : __v_table_ptr(B::'v_table'), { Bar(); ... }, ~str2(), ~A()  
  
C::C() : B(), c(0), str3(), __v_table_ptr(C::'v_table'), { Bar(); ... }  
C::~C() : __v_table_ptr(C::'v_table'), { Bar(); ... }, ~str3(), ~B()
```

# C++ Under the Hood

## ‘C’ Pointer types

**int \***

**Pointer to Data**

**int(\*)(int)**

**Pointer to Function**

## Member Pointer types

**int Foo::\***

**Pointer to Member Data**

**int(Foo::\*)(int)**

**Pointer to Member Function**

# C++ Under the Hood - ‘C’ Style Pointers - Pointer to Data(aka object)

```
struct Foo {
    int i{};
};

int* Bar(int* p) {
    (*p)++;
    return p;
}

int main()
{
    int i{2};
    Foo foo{3};

    int* p1 = &i;
    int* p2 = &foo.i;
    int* p3 = Bar(p1);

    return *p2 + *p3;
}
```

[godbolt.org/z/rKzdTsejz](https://godbolt.org/z/rKzdTsejz)

# C++ Under the Hood

## 'C' Pointer types

`int *`

Pointer to Data

`int(*)(int)`

**Pointer to Function**

## Member Pointer types

`int Foo::*`

Pointer to Member Data

`int(Foo::*)(int)`

Pointer to Member Function

# C++ Under the Hood

`int`      `(* foo )`      `( int , int );`

Return type specification for the target function

declares a function pointer  
"foo" is the variable name  
Parameter types specified for the target function.

# C++ Under the Hood - ‘C’ Style Pointers - Pointer to Function (PFN)

```
int Foo(int i)
{
    return i++;
}

int Bar(int(*pfn)(int), int i)
{
    return pfn(i);
}

int main()
{
    return Bar(&Foo, 1);
}
```

[godbolt.org/z/hvYM1no1b](https://godbolt.org/z/hvYM1no1b)

# C++ Under the Hood – ‘C’ Style Pointers - Pointer to Function (PFN)

```
int Bar(int(*pFN)(int))
{
    return pFN(10);
}

int Foo(int i)
{
    return i * (i+1) / 2;
}

int main()
{
    int(*pFN)(int) = &Foo;

    return Bar(pFN);
}
```

[godbolt.org/z/Y1c7eWjb6](https://godbolt.org/z/Y1c7eWjb6)

# C++ Under the Hood – ‘C’ Style Pointers - Pointer to Function (PFN)

```
typedef int(*PFN)(int);

int Bar(PFN pFN)
{
    return pFN(10);
}

int Foo(int i)
{
    return i * (i+1) / 2;
}

int main()
{
    PFN pFN = &Foo;

    return Bar(pFN);
}
```

[godbolt.org/z/s9cvWaEq8](https://godbolt.org/z/s9cvWaEq8)

# C++ Under the Hood – ‘C’ Style Pointers - Pointer to Function (PFN)

```
typedef int(PFN)(int);

int Bar(PFN pFN)
{
    return pFN(10);
}

int Foo(int i)
{
    return i * (i+1) / 2;
}

int main()
{
    PFN *pFN = &Foo;

    return Bar(pFN);
}
```

[godbolt.org/z/s9cvWaEq8](https://godbolt.org/z/s9cvWaEq8)

# C++ Under the Hood - ‘C’ Style Pointers - Pointer to Function (PFN)

```
typedef int(PFN)(int);

int Bar(PFN pFN)
{
    return pFN(10);
}

int Foo(int i)
{
    return i * (i+1) / 2;
}

int main()
{
    PFN *pFN = Foo;

    return Bar(pFN);
}
```

# C++ Under the Hood - ‘C’ Style Pointers – qsort(...) Pointer to Function (PFN)

```
int compare(const void* a, const void* b) {
    return (*(int*)a - *(int*)b);
}

int main() {
    int arr[] = {10, 5, 4, 6, 9};
    int count = sizeof(arr) / sizeof(arr[0]);

    qsort(arr, count, sizeof(arr[0]), compare);

    printf("Sorted array: ");
    for (int i = 0; i < count; ++i) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

[godbolt.org/z/vc8PzYcs3](https://godbolt.org/z/vc8PzYcs3)

# C++ Under the Hood

## ‘C’ Pointer types

`int *`

Pointer to Data

`int(*)(int)`

Pointer to Function

## Member Pointer types

`int Foo::*`

Pointer to Member Data

`int(Foo::*)(int)`

**Pointer to Member Function**

# C++ Under the Hood - ‘C’ Style Pointers - Pointer to Member Function

```
typedef int(*PFN)(int);

struct Foo
{
    int i;
    int Bar(int j) { return i+j; }
};

int main()
{
    PFN pFN = &Foo::Bar;

    Foo obj{10};

    return (obj.*pFN)(20);
}
```

# C++ Under the Hood - 'C' Style Pointers - Pointer to Member Function

```
typedef int(*PFN)(int);

struct Foo
{
    int i;
    int Bar(int j) { return i+j; }
};

int main()
{
    PFN pFN = &Foo::Bar;

    Foo obj{10};

    return (obj.*pMF)(20);
}
```

[godbolt.org/z/jfr9dMj7G](https://godbolt.org/z/jfr9dMj7G)

# C++ Under the Hood – C++ Pointer to Member Function

```
struct Foo;  
typedef int(Foo::*PMF)(int);  
  
struct Foo  
{  
    int i;  
    int Bar(int j) { return i+j; }  
};  
  
int main()  
{  
    PMF pMF = &Foo::Bar;  
  
    Foo obj{10};  
  
    return (obj.*pMF)(20);  
}
```

[godbolt.org/z/eY1n7a97x](https://godbolt.org/z/eY1n7a97x)

# C++ Under the Hood – C++ Pointer to (virtual) Member Function

```
struct Foo;
typedef int(Foo::*PMF)(int);
struct Foo
{
    int i;
    Foo(int i) : i(i) {}
    virtual int Bar(int j) { return i+j; }
};
struct Foo2 : Foo
{
    Foo2(int i) : Foo(i) {}
    int Bar(int j) override { return i*j; }
};
int main()
{
    PMF pMF = &Foo::Bar;
    Foo2 obj{10};
    return (obj.*pMF)(20);
}
```

[godbolt.org/z/qWhE5ErKz](https://godbolt.org/z/qWhE5ErKz)

# C++ Under the Hood – C++ Pointer to (virtual) Member Function

```
struct Foo;
typedef int(Foo::*PMF)(int);
struct Foo
{
    int i;
    Foo(int i) : i(i) {}
    virtual int Bar(int j) { return i+j; }
};
struct Foo2 : Foo
{
    Foo2(int i) : Foo(i) {}
    int Bar(int j) override { return i*j; }
};
int main()
{
    PMF pMF = &Foo::Bar;
    Foo2 obj{10};
    Foo *pFoo = &obj;
    return (pFoo->*pMF)(20);
}
```

[godbolt.org/z/vTfM143d7](https://godbolt.org/z/vTfM143d7)

# C++ Under the Hood – C++ Pointer to (virtual) Member Function

```
struct Foo;
using PMF = int(Foo::*)(int);
struct Foo
{
    int i;
    Foo(int i) : i(i) {}
    virtual int Bar(int j) { return i+j; }
};
struct Foo2 : Foo
{
    Foo2(int i) : Foo(i) {}
    int Bar(int j) override { return i*j; }
};
int main()
{
    PMF pMF = &Foo::Bar;
    Foo2 obj{10};
    Foo *pFoo = &obj;
    return (pFoo->pMF)(20);
}
```

[godbolt.org/z/188fcronz](https://godbolt.org/z/188fcronz)

# C++ Under the Hood – C++ Pointer to (virtual) Member Function

```
struct Foo
{
    using PMF = int(Foo::*)(int);

    int i;
    Foo(int i) : i(i) {}
    virtual int Bar(int j) { return i+j; }
};

struct Foo2 : Foo
{
    Foo2(int i) : Foo(i) {}
    int Bar(int j) override { return i*j; }
};

int main()
{
    Foo::PMF pMF = &Foo::Bar;
    Foo2 obj{10};
    Foo *pFoo = &obj;
    return (pFoo->*pMF)(20);
}
```

# C++ Under the Hood – C++ Pointer to (virtual) Member Function

```
struct Foo
{
    using PMF = int(Foo::*)(int);

    int i;
    Foo(int i) : i(i) {}
    virtual int Bar(int j) { return i+j; }

};

struct Foo2 : Foo
{

    Foo2(int i) : Foo(i) {}
    int Bar(int j) override { return i*j; }
};

int main()
{
    Foo::PMF pMF = &Foo::Bar;
    Foo2 obj{10};
    Foo *pFoo = &obj;
    return (pFoo->*pMF)(20);
}
```

# C++ Under the Hood – C++ Pointer to (virtual) Member Function

```
struct Foo
{
    using PMF = int(Foo::*)(int);
    V_table* pV_table;
    static V_table Foo::v_table[] = { &Foo::Bar, };

    int i;
    Foo(int i) : i(i) {}
    virtual int Bar(int j) { return i+j; }

};

struct Foo2 : Foo
{
    static V_table Foo2::v_table[] = { &Foo2::Bar, };

    Foo2(int i) : Foo(i) {}
    int Bar(int j) override { return i*j; }
};

int main()
{
    Foo::PMF pMF = &Foo::Bar;
    Foo2 obj{10};
    Foo *pFoo = &obj;
    return (pFoo->*pMF)(20);
}
```

# C++ Under the Hood – C++ Pointer to (virtual) Member Function

```
struct Foo
{
    using PMF = int(Foo::*)(int);
    V_table* pV_table;
    static V_table Foo::v_table[] = { &Foo::Bar, };

    int i;
    Foo(int i) : i(i) {}

    virtual int Bar(int j) { return i+j; }

    int Foo::Bar_Thunk(int i) { return (this->pV_table[0])(int i); }
};

struct Foo2 : Foo
{
    static V_table Foo2::v_table[] = { &Foo2::Bar, };

    Foo2(int i) : Foo(i) {}

    int Bar(int j) override { return i*j; }
};

int main()
{
    Foo::PMF pMF = &Foo::Bar; // = &Foo::Bar_Thunk
    Foo2 obj{10};
    Foo *pFoo = &obj;
    return (pFoo->*pMF)(20);
}
```

# C++ Under the Hood – C++ Pointer to (virtual) Member Function

```
struct Foo
{
    using PMF = int(Foo::*)(int);
    V_table* pV_table;
    static V_table Foo::v_table[] = { &Foo::Bar, };

    int i;
    Foo(int i) : i(i) {}

    virtual int Bar(int j) { return i+j; }

    int Foo::Bar_Thunk(int i) { return (this->pV_table[0])(int i); }

};

struct Foo2 : Foo
{
    Foo2(int i) : Foo(i) {}

    int Bar(int j) override { return i*j; }

};

int main()
{
    Foo::PMF pMF = &Foo::Bar; // = &Foo::Bar_Thunk
    Foo2 obj{10};
    Foo *pFoo = &obj;
    return (pFoo->*pMF)(20);
}
```

# C++ Under the Hood – C++ Pointer to (virtual) Member Function

```
struct Foo
{
    using PMF = int(Foo::*)(int);
    V_table* pV_table;
    static V_table Foo::v_table[] = { &Foo::Bar, };

    int i;
    Foo(int i) : i(i) {}

    virtual int Bar(int j) { return i+j; }

    int Foo::Bar_Thunk(int i) { return (this->pV_table[0])(int i); }

};

struct Foo2 : Foo
{
    Foo2(int i) : Foo(i) {}

    int Bar(int j) override { return i*j; }

};

int main()
{
    Foo::PMF pMF = &Foo::Bar; // = &Foo::Bar_Thunk
    Foo2 obj{10};
    Foo *pFoo = &obj;
    return (pFoo->*pMF)(20);
}
```

# C++ Under the Hood – C++ Pointer to (virtual) Member Function

```
struct Foo
{
    using PMF = int(Foo::*)(int);

    int i;
    Foo(int i) : i(i) {}
    virtual int Bar(int j) { return i+j; }

};

struct Foo2 : Foo
{
    Foo2(int i) : Foo(i) {}
    int Bar(int j) override { return i*j; }
};

int main()
{
    Foo::PMF pMF = &Foo::Bar; // = &Foo::Bar_Thunk
    Foo2 obj{10};
    Foo *pFoo = &obj;
    return (pFoo->*pMF)(20);
}
```

# C++ Under the Hood – C++ Pointer to (virtual) Member Function

```
struct Foo
{
    using PMF = int(Foo::*)(int);

    int i;
    Foo(int i) : i(i) {}
    virtual int Bar(int j) { return i+j; }
};

struct Foo2 : Foo
{
    Foo2(int i) : Foo(i) {}
    int Bar(int j) override { return i*j; }
};

int main()
{
    Foo::PMF pMF = &Foo::Bar; // = &Foo::Bar_Thunk
    Foo2 obj{10};
    Foo *pFoo = &obj;
    return (pFoo->*pMF)(20);
}
```

The diagram illustrates the memory layout for pointer-to-member-functions (pMF). It shows a pointer `pMF` pointing to the `vtable` entry for the `Bar` member function. This entry is a thunk function named `Bar_Thunk`, which then calls the actual implementation in the `vtable` of the `Foo2` class.

# C++ Under the Hood – C++ Pointer to (virtual) Member Function

```
struct Foo
{
    using PMF = int(Foo::*)(int);

    int i;
    Foo(int i) : i(i) {}
    virtual int Bar(int j) { return i+j; }
};

struct Foo2 : Foo
{
    Foo2(int i) : Foo(i) {}
    int Bar(int j) override { return i*j; }
};

int main()
{
    Foo::PMF pMF = &Foo::Bar; // = &Foo::Bar_Thunk
    Foo2 obj{10};
    Foo *pFoo = &obj;
    return (pFoo->*pMF)(20);
}
```

The diagram illustrates the v-table pointer mechanism. A red line points from the declaration of `pV_table` to its definition in the `Bar_Thunk` function. Another red line points from the `v_table` array in the `Bar_Thunk` function to the first element, which is the address of the `Bar` function. A third red line points from the `v_table` array in the `Bar_Thunk` function to the second element, which is the address of the `Bar_Thunk` function.

# C++ Under the Hood – C++ Pointer to (virtual) Member Function

```
struct Foo
{
    using PMF = int(Foo::*)(int);

    int i;
    Foo(int i) : i(i) {}
    virtual int Bar(int j) { return i+j; }
};

struct Foo2 : Foo
{
    Foo2(int i) : Foo(i) {}
    int Bar(int j) override { return i*j; }
};

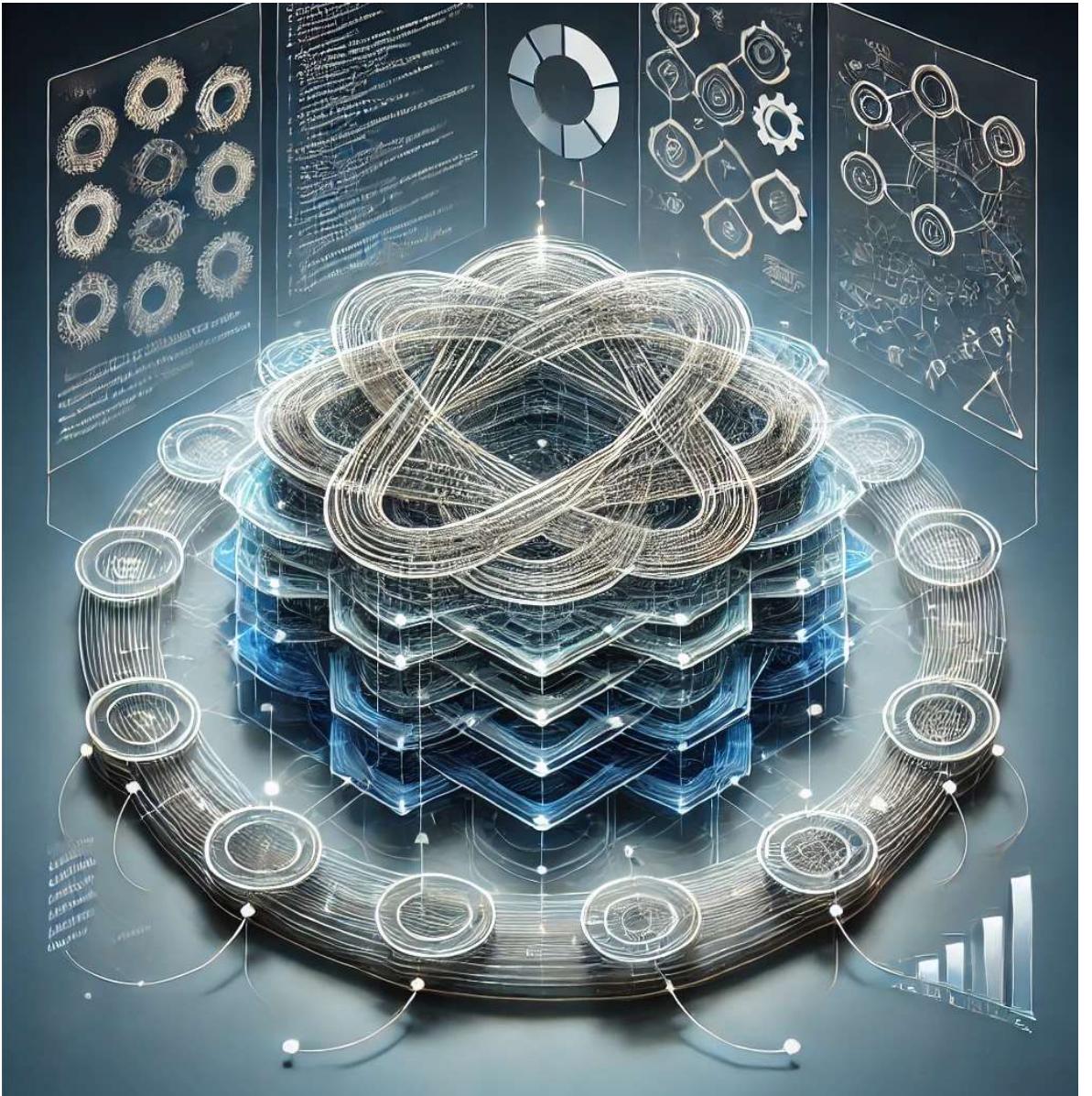
int main()
{
    Foo::Pmf pMf = &Foo::Bar; // = &Foo::Bar_Thunk
    Foo2 obj{10};
    Foo *pFoo = &obj;
    return (pFoo->*pMf)(20);
}
```

The diagram illustrates the memory layout of the `Foo` class. It shows the `v_table` pointer and the `Bar` function pointer. The `v_table` pointer points to the `v_table` array, which contains a pointer to the `Bar` function. The `Bar` function pointer points to the `Bar` thunk function, which then calls the actual `Bar` implementation via the `pV_table`.

# C++ Under the Hood

## Fundamental Theorem of Software Engineering:

Any problem can be solved by adding a layer of indirection.



# C++ Under the Hood

## ‘C’ Pointer types

`int *`

Pointer to Data

`int(*)(int)`

Pointer to Function

## Member Pointer types

`int Foo::*`

**Pointer to Member Data**

`int(Foo::*)(int)`

Pointer to Member Function

# C++ Under the Hood – C++ Pointer to Member Data

```
int main()
{
    struct Foo { int a, b, c, d; };

    Foo foo1{ 2, 3, 5, 7};
    Foo foo2{11, 13, 17, 19};

    Foo* pFoo1 = &foo1;
    Foo* pFoo2 = &foo2;

    cout << "foo1.a = " << foo1.a << '\n';           //foo1.a = 2
    cout << "foo2.a = " << foo2.a << '\n';           //foo2.a = 11

    cout << "pFoo1->c = " << pFoo1->c << '\n';     //foo1.c = 5
    cout << "pFoo2->c = " << pFoo2->c << '\n';     //foo2.c = 17
}
```

# C++ Under the Hood – C++ Pointer to Member Data

```
int main()
{
    struct Foo { int a, b, c, d; };

    Foo foo1{ 2, 3, 5, 7};
    Foo foo2{11, 13, 17, 19};

    Foo* pFoo1 = &foo1;
    Foo* pFoo2 = &foo2;

    int Foo::* p = &Foo::a;
    cout << "foo1.*p = " << foo1.*p << '\n';           //foo1.*p = 2 (aka foo1.a)
    cout << "foo2.*p = " << foo2.*p << '\n';           //foo2.*p = 11 (aka foo2.a)

    p = &Foo::c;
    cout << "pFoo1->*p = " << pFoo1->*p << '\n';     //pFoo1->*p = 5 (aka foo1.c)
    cout << "pFoo1->*p = " << pFoo2->*p << '\n';     //pFoo2->*p2 = 17 (aka foo2.c)
}
```

# C++ Under the Hood

**N U L L**

# C++ Under the Hood



# C++ Under the Hood

nullptr

# C++ Under the Hood



# C++ Under the Hood – C++ Pointer to Member Data

```
int main()
{
    struct Foo { int a, b, c, d; };
    int Foo::* p;

    p = nullptr;    cout << "(p) = " << p << '\n';           //0 (zero, aka false) unassigned
    p = &Foo::a;    cout << "(p) = " << p << '\n';           //1 (one, aka true) assigned
    p = &Foo::b;    cout << "(p) = " << p << "\n\n";         //1 (one, aka true) assigned
}

}
```

# C++ Under the Hood – C++ Pointer to Member Data

```
int main()
{
    struct Foo { int a, b, c, d; };
    int Foo::* p;

    p = nullptr;    cout << "(p) = " << p << '\n';           //0 (zero, aka false) unassigned
    p = &Foo::a;    cout << "(p) = " << p << '\n';           //1 (one, aka true) assigned
    p = &Foo::b;    cout << "(p) = " << p << "\n\n";          //1 (one, aka true) assigned

    p = &Foo::a;    cout << "*(∫*)&p = " << *(∫*)&p << '\n';   //p = 0
    p = &Foo::b;    cout << "*(∫*)&p = " << *(∫*)&p << '\n';   //p = 4
    p = &Foo::c;    cout << "*(∫*)&p = " << *(∫*)&p << '\n';   //p = 8
    p = &Foo::d;    cout << "*(∫*)&p = " << *(∫*)&p << "\n\n"; //p = 12

}
```

# C++ Under the Hood – C++ Pointer to Member Data

```
int main()
{
    struct Foo { int a, b, c, d; };
    int Foo::* p;

    p = nullptr;    cout << "(p) = " << p << '\n';           //0 (zero, aka false) unassigned
    p = &Foo::a;    cout << "(p) = " << p << '\n';           //1 (one, aka true) assigned
    p = &Foo::b;    cout << "(p) = " << p << "\n\n";          //1 (one, aka true) assigned

    p = &Foo::a;    cout << "*(∫*)&p = " << *(∫*)&p << '\n';   //p = 0
    p = &Foo::b;    cout << "*(∫*)&p = " << *(∫*)&p << '\n';   //p = 4
    p = &Foo::c;    cout << "*(∫*)&p = " << *(∫*)&p << '\n';   //p = 8
    p = &Foo::d;    cout << "*(∫*)&p = " << *(∫*)&p << "\n\n"; //p = 12

    p = nullptr;    cout << "(p) = " << p << '\n';           //0 (zero, aka false) unassigned
                    cout << "*(∫*)&p = " << *(∫*)&p << '\n';           //p = -1
}
```

# C++ Under the Hood – C++ Pointer to Member Data (pointer to object + offset)

```
int main()
{
    struct Foo { int a, b, c, d; };

    Foo foo1{ 2, 3, 5, 7};
    Foo foo2{11, 13, 17, 19};

    Foo* pFoo1 = &foo1;
    Foo* pFoo2 = &foo2;

    int Foo::* p = &Foo::a;
    cout << "foo1.*p = " << foo1.*p << '\n';           //foo1.*p = 2 (aka foo1.a)
    cout << "foo2.*p = " << foo2.*p << '\n';           //foo2.*p = 11 (aka foo2.a)

    p = &Foo::c;
    cout << "pFoo1->*p = " << pFoo1->*p << '\n';     //pFoo1->*p = 5 (aka foo1.c)
    cout << "pFoo1->*p = " << pFoo2->*p << '\n';     //pFoo2->*p2 = 17 (aka foo2.c)
}
```

# C++ Under the Hood - ‘C’ Style Pointers – qsort(...) Pointer to Function (PFN)

```
int compare(const void* a, const void* b) {
    return (*(int*)a - *(int*)b);
}

int main() {
    int arr[] = {10, 5, 4, 6, 9};
    int count = sizeof(arr) / sizeof(arr[0]);

    qsort(arr, count, sizeof(arr[0]), compare);

    printf("Sorted array: ");
    for (int i = 0; i < count; ++i) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

[godbolt.org/z/vc8PzYcs3](https://godbolt.org/z/vc8PzYcs3)

# C++ Under the Hood – C++ style Sort(...) with lambda [](l&, r&){ l < r }

```
struct Foo { int a, b, c, d; };

void Sort(std::vector<Foo>& data) {
    std::sort(std::begin(data), std::end(data),
              [] (auto& lhs, auto& rhs){ return lhs.a < rhs.a; });
}

int main()
{
    auto data = GetData();

    Sort(data);
    Output(data);
}
```

[godbolt.org/z/8vvr9brEo](https://godbolt.org/z/8vvr9brEo)

# C++ Under the Hood – C++ style Sort(...) with lambda +[](l&, r&){ l < r}

```
struct Foo { int a, b, c, d; };

void Sort(std::vector<Foo>& data, bool(*compare)(Foo&, Foo&)) {
    std::sort(std::begin(data), std::end(data), compare);
}

int main()
{
    auto data = GetData();

    Sort(data, +[](Foo& lhs, Foo& rhs)->bool { return lhs.a < rhs.a; });
    Output(data);

    Sort(data, +[](Foo& lhs, Foo& rhs)->bool { return lhs.b < rhs.b; });
    Output(data);

    Sort(data, +[](Foo& lhs, Foo& rhs)->bool { return lhs.c < rhs.c; });
    Output(data);

    Sort(data, +[](Foo& lhs, Foo& rhs)->bool { return lhs.d < rhs.d; });
    Output(data);
}
```

[godbolt.org/z/48aTYosKr](https://godbolt.org/z/48aTYosKr)

# C++ Under the Hood – C++ Pointer to Member Data

```
struct Foo { int a, b, c, d; };
//...
template<typename T>
void Sort(auto& data, int T::* key) {
    std::sort(std::begin(data), std::end(data),
              [key](auto& lhs, auto& rhs){ return lhs.*key < rhs.*key; });
}
int main()
{
    auto data = GetData();

    Sort(data, &Foo::a);      std::cout<<"a:\n";      Output(data);
    Sort(data, &Foo::b);      std::cout<<"b:\n";      Output(data);
    Sort(data, &Foo::c);      std::cout<<"c:\n";      Output(data);
    Sort(data, &Foo::d);      std::cout<<"d:\n";      Output(data);
}
```

# C++ Under the Hood – C++ Pointer to Member Data

```
template<typename T>
bool compare(auto& lhs, auto& rhs, auto T::* key, auto...keys) {
    if (lhs.*key < rhs.*key) return true;
    if (lhs.*key > rhs.*key) return false;
    if constexpr (sizeof...(keys)>0)
        return compare(lhs, rhs, keys...);
    return false;
}

void Sort(auto& data, auto ...keys) {
    std::sort(std::begin(data), std::end(data),
              [keys...](auto& lhs, auto& rhs)
              { return compare(lhs, rhs, keys...); });
}

int main()
{
    std::vector<Data> data = GetData();

    std::cout << "Sort based on b\n";
    Shuffle(data);           Sort(data, &Data::b, &Data::a);           Output(data);

    std::cout << "Sort based on c,d,e\n";
    Shuffle(data);           Sort(data, &Data::c, &Data::d, &Data::e);           Output(data);
}
```

# C++ Under the Hood – Inheritance vs. Aggregation

## Inheritance

- **Is-a**

## Aggregation

- **Has-a**

A Person **is NOT a** Name

A Person **has a** Name

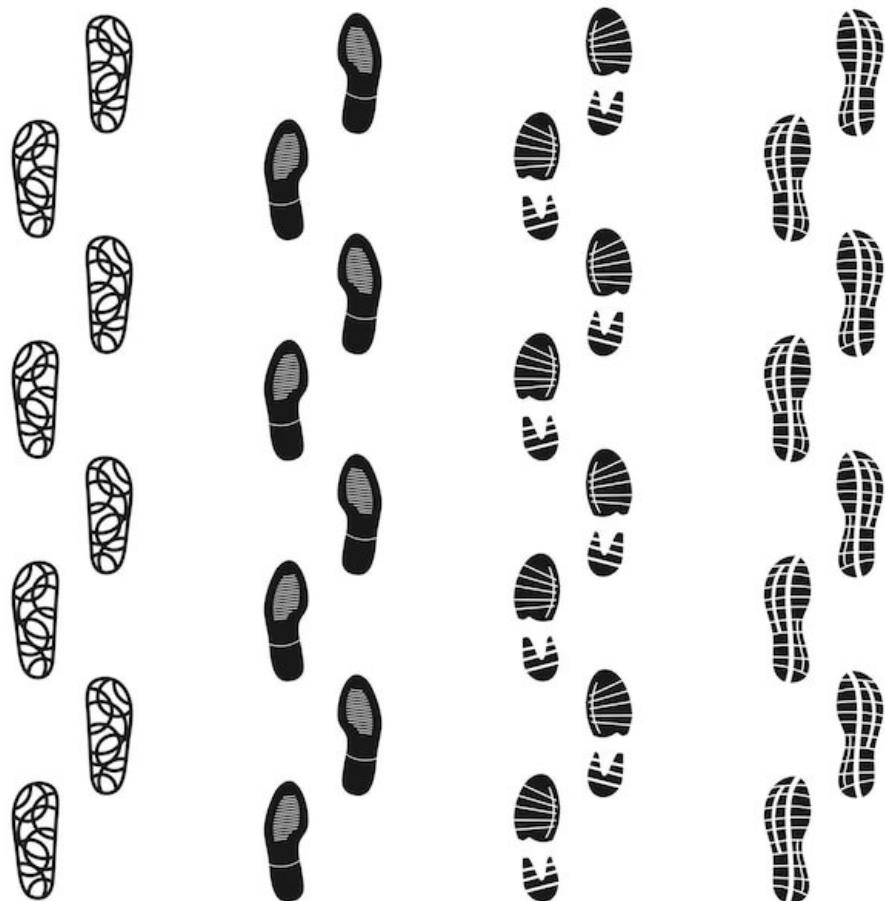
An Employee **is a** Person

```
struct Person
{
    string name;
    date    dob;
    // ...
};

struct Employee : Person
{
    uint    id;
    // ...
};
```

# C++ Under the Hood

## Object Memory Footprint



## v-table Mechanism



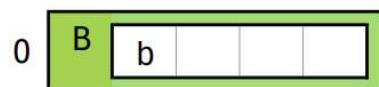
# C++ Under the Hood

```
struct A{  
    int a;  
    int Foo(...){...};  
};
```



A::Foo(...){...}

```
struct B{  
    int b;  
    int Foo(...){...};  
};
```



B::Foo(...){...}

```
struct C{  
    int c;  
    int Foo(...){...};  
};
```



C::Foo(...){...}

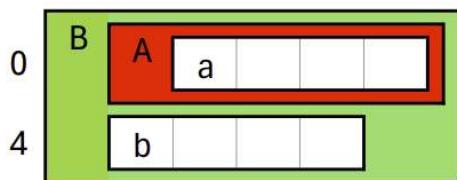
# C++ Under the Hood

```
struct A{  
    int a;  
    int Foo(...){...};  
};
```



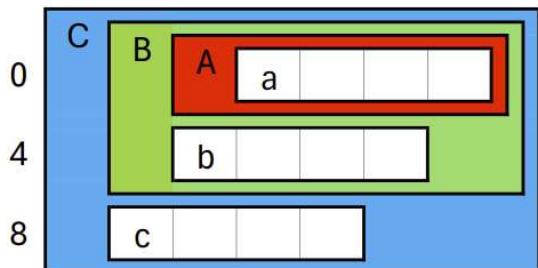
A::Foo(...){...}

```
struct B : A{  
    int b;  
    int Foo(...){...};  
};
```



B::Foo(...){...}

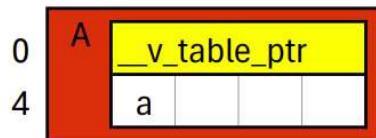
```
struct C : B{  
    int c;  
    int Foo(...){...};  
};
```



C::Foo(...){...}

# C++ Under the Hood

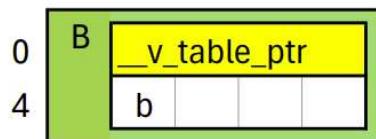
```
struct A {  
    int a;  
    A(...){...};  
    virtual int Bar(...){...};  
    int Foo(...){...};  
};
```



A::'v\_table'= {A::Bar};

A::A(...){...}  
A::Bar(...){...}  
A::Foo(...){...}

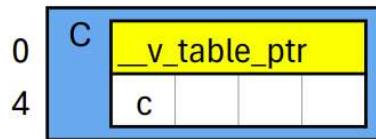
```
struct B {  
    int b;  
    B(...){...};  
    virtual int Bar(...){...};  
    int Foo(...){...};  
};
```



B::'v\_table'= {B::Bar};

B::B(...){...}  
B::Bar(...){...}  
B::Foo(...){...}

```
struct C {  
    int c;  
    C(...){...};  
    virtual int Bar(...){...};  
    int Foo(...){...};  
};
```

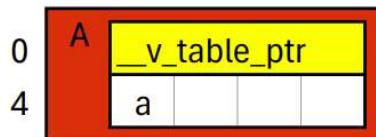


C::'v\_table'= {C::Bar};

C::C(...){...}  
C::Bar(...){...}  
C::Foo(...){...}

# C++ Under the Hood

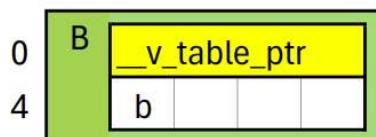
```
struct A{
    int a;
    A(...){...};
    virtual ~A(){...};
    virtual int Bar(...){...};
    int Foo(...){...};
};
```



A::\_\_v\_table' = {A::~A, A::Bar};

A::A(...){...}  
A::~A() {...}  
A::Bar(...){...}  
A::Foo(...){...}

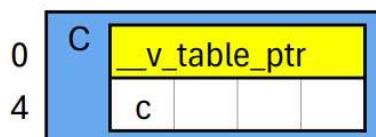
```
struct B{
    int b;
    B(...){...};
    virtual ~B(){...};
    virtual int Bar(...){...};
    int Foo(...){...};
};
```



B::\_\_v\_table' = {B::~B, B::Bar};

B::B(...){...}  
B::~B() {...}  
B::Bar(...){...}  
B::Foo(...){...}

```
struct C{
    int c;
    C(...){...};
    virtual ~C(){...};
    virtual int Bar(...){...};
    int Foo(...){...};
};
```

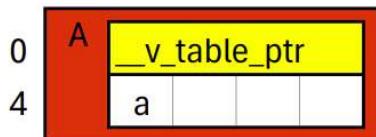


C::\_\_v\_table' = {C::~C, C::Bar};

C::C(...){...}  
C::~C() {...}  
C::Bar(...){...}  
C::Foo(...){...}

# C++ Under the Hood

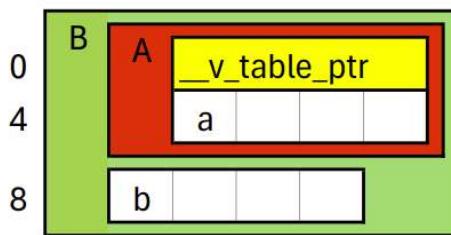
```
struct A{
    int a;
    A(...){...};
    virtual ~A(){...};
    virtual int Bar(...){...};
    int Foo(...){...};
};
```



A::\_\_v\_table' = {A::~A, A::Bar};

A::A(...){...}  
A::~A(){...}  
A::Bar(...){...}  
A::Foo(...){...}

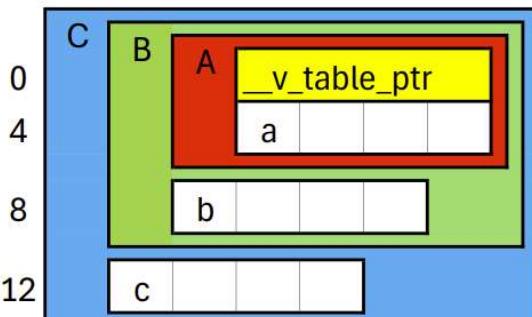
```
struct B : A{
    int b;
    B(...){...};
    virtual ~B(){...};
    virtual int Bar(...){...};
    int Foo(...){...};
};
```



B::\_\_v\_table' = {B::~B, B::Bar};

B::B(...){...}  
B::~B(){...}  
B::Bar(...){...}  
B::Foo(...){...}

```
struct C : B{
    int c;
    C(...){...};
    virtual ~C(){...};
    virtual int Bar(...){...};
    int Foo(...){...};
};
```

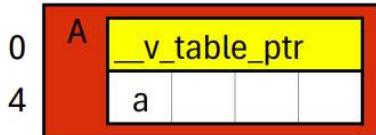


C::\_\_v\_table' = {C::~C, C::Bar};

C::C(...){...}  
C::~C(){...}  
C::Bar(...){...}  
C::Foo(...){...}

# C++ Under the Hood

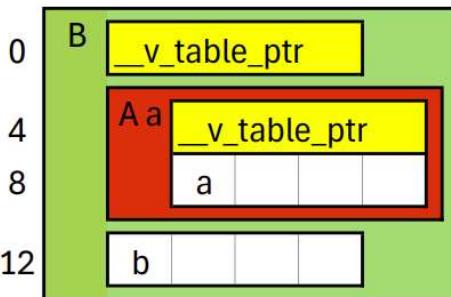
```
struct A{
    int a;
    A(...){...};
    virtual ~A() {...};
    virtual int Bar(...){...};
    int Foo(...){...};
};
```



A::'v\_table'= {A::~A, A::Bar};

A::A(...){...}  
A::~A() {...}  
A::Bar(...){...}  
A::Foo(...){...}

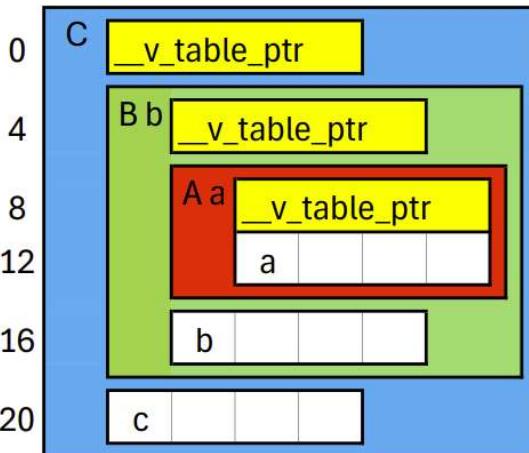
```
struct B{
    A a;
    int b;
    B(...){...};
    virtual ~B() {...};
    virtual int Bar(...){...};
    int Foo(...){...};
};
```



B::'v\_table'= {B::~B, B::Bar};  
A::'v\_table'= {A::~A, A::Bar};

B::B(...){...}  
B::~B() {...}  
B::Bar(...){...}  
B::Foo(...){...}

```
struct C{
    B b;
    int c;
    C(...){...};
    virtual ~C() {...};
    virtual int Bar(...){...};
    int Foo(...){...};
};
```

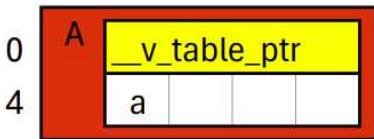


C::'v\_table'= {C::~C, C::Bar};  
B::'v\_table'= {B::~B, B::Bar};  
A::'v\_table'= {A::~A, A::Bar};

C::C(...){...}  
C::~C() {...}  
C::Bar(...){...}  
C::Foo(...){...}

# C++ Under the Hood

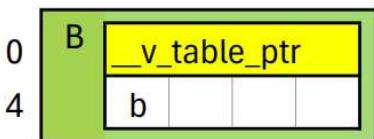
```
struct A {  
    int a;  
    A(...){...};  
    virtual ~A(){...};  
    virtual int Bar(...){...};  
};
```



A::\_\_v\_table' = {A::~A, A::Bar};

A::A(...){...}  
A::~A(){...}  
A::Bar(...){...}

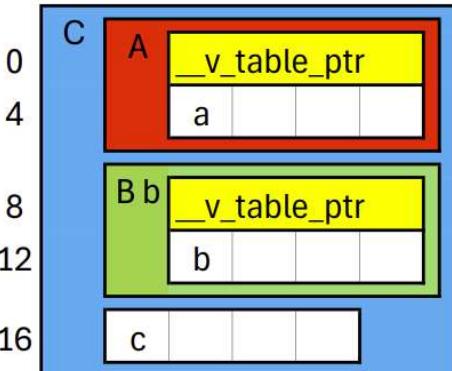
```
struct B {  
    int b;  
    B(...){...};  
    virtual ~B(){...};  
    virtual int Bar(...){...};  
    int Foo(...){...};  
};
```



B::\_\_v\_table' = {B::~B, B::Bar};

B::B(...){...}  
B::~B(){...}  
B::Bar(...){...}  
B::Foo(...){...}

```
struct C : A {  
    B b;  
    int c;  
    C(...){...};  
    virtual ~C(){...};  
    virtual int Bar(...){...};  
};
```



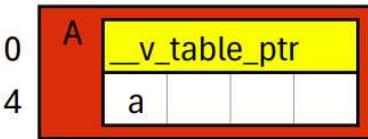
C::A::\_\_v\_table' = {C::~C, C::Bar};

C::C(...){...}  
C::~C(){...}  
C::Bar(...){...}

B::B::\_\_v\_table' = {B::~B, B::Bar};

# C++ Under the Hood

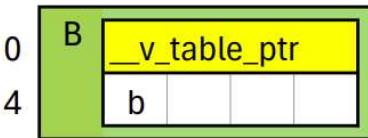
```
struct A {
    int a;
    A(...){...};
    virtual ~A(){...};
    virtual int Bar(...){...};
};
```



A::\_\_v\_table' = {A::~A, A::Bar};

A::A(...){...}  
A::~A(){...}  
A::Bar(...){...}

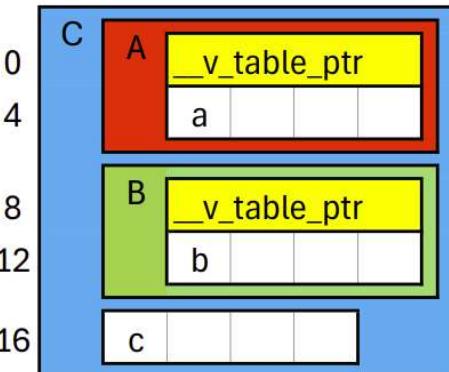
```
struct B {
    int b;
    B(...){...};
    virtual ~B(){...};
    virtual int Bar(...){...};
    int Foo(...){...};
};
```



B::\_\_v\_table' = {B::~B, B::Bar};

B::B(...){...}  
B::~B(){...}  
B::Bar(...){...}  
B::Foo(...){...}

```
struct C : A, B {
    int c;
    C(...){...};
    virtual ~C(){...};
    virtual int Bar(...){...};
};
```



C::A::\_\_v\_table' = {C::~C, C::Bar};

C::C(...){...}  
C::~C(){...}  
C::Bar(...){...}  
C::~C\_Adj8() {...}  
C::Bar\_Adj8() {...}

C::B::\_\_v\_table' = {C::~C\_Adj8, C::Bar\_Adj8};

# C++ Under the Hood – Multiple this pointers

```
struct A {
    int a{};
    A() { cout << "A::A()" this=" << this << "\n"; }
    virtual ~A() { cout << "A::~A()" this=" << this << "\n"; }
    virtual void Bar() { cout << "A::Bar()" this=" << this << "\n"; }
};

struct B {
    int b{};
    B() { cout << "B::B()" this=" << this << "\n"; }
    virtual ~B() { cout << "B::~B()" this=" << this << "\n"; }
    void Foo() { cout << "B::Foo()" this=" << this << "\n"; }
};

struct C : A ,B {
    int c{};
    C() { cout << "C::C()" this=" << this << "\n"; }
    virtual ~C() { cout << "C::~C()" this=" << this << "\n"; }
    virtual void Bar() { cout << "C::Bar()" this=" << this << "\n"; Foo(); }
};

int main() {
    cout << "\nC* pC = new C;\n";
    cout << "npA=" << pA << " pA->Bar();\n";
    cout << "npC=" << pC << " pC->Foo(); \n";
    cout << "npB=" << pB << " delete pB;\n";
}

C* pC = new C;
A* pA = pC;
pA->Bar();
pC->Foo();
B* pB = pC;
delete pB;
```

C\* pC = new C;  
A::A() this=009E85C8  
B::B() this=009E85D0  
C::C() this=009E85C8

pA=009E85C8 pA->Bar();  
C::Bar() this=009E85C8  
B::Foo() this=009E85D0

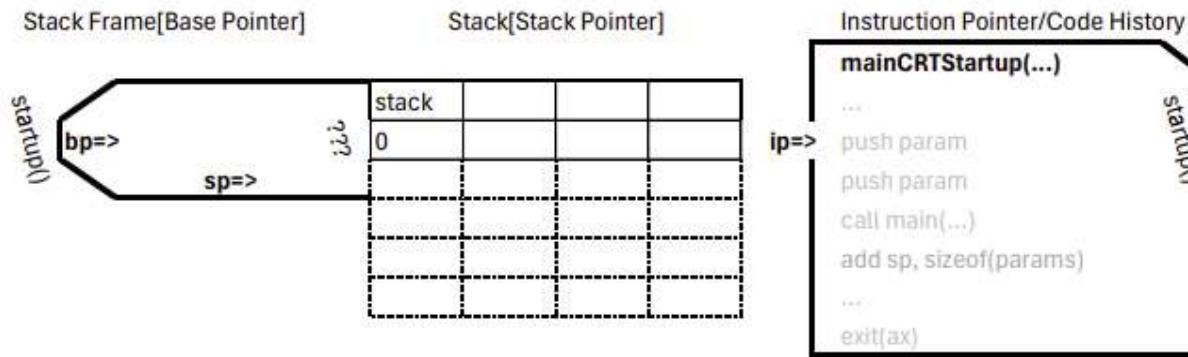
pC=009E85C8 pC->Foo();  
B::Foo() this=009E85D0

pB=009E85D0 delete pB;  
C::~C() this=009E85C8  
B::~B() this=009E85D0  
A::~A() this=009E85C8

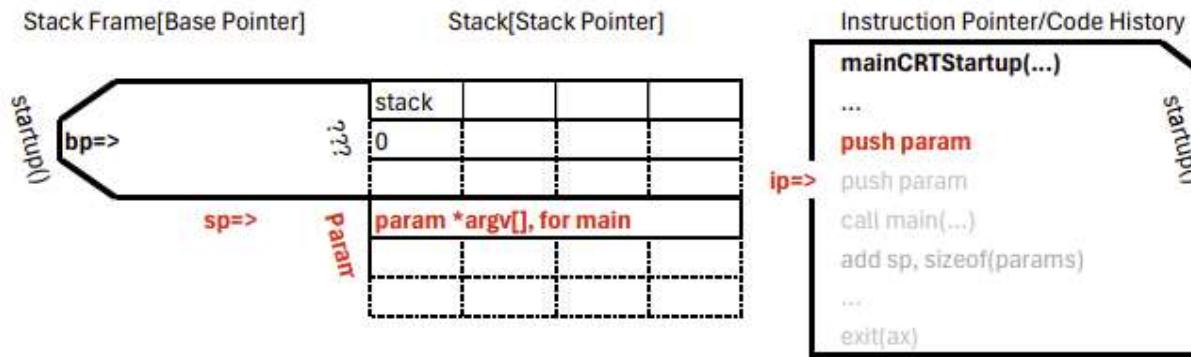
# C++ Under the Hood – Stacks of Frames



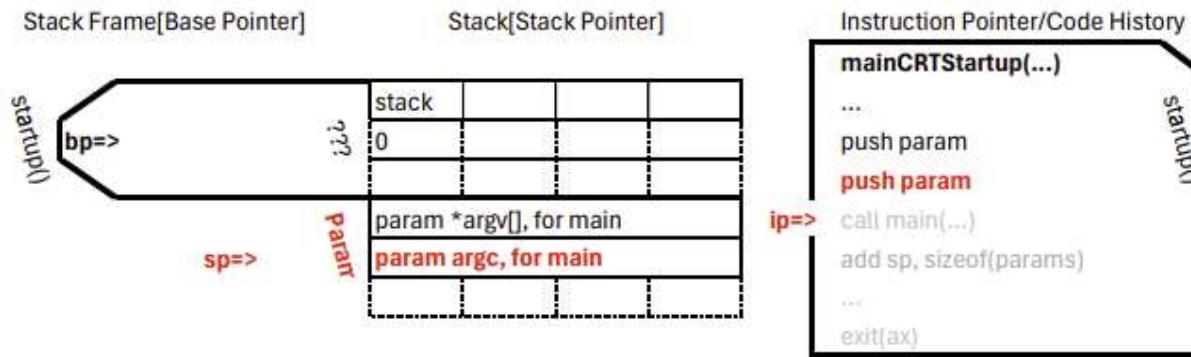
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



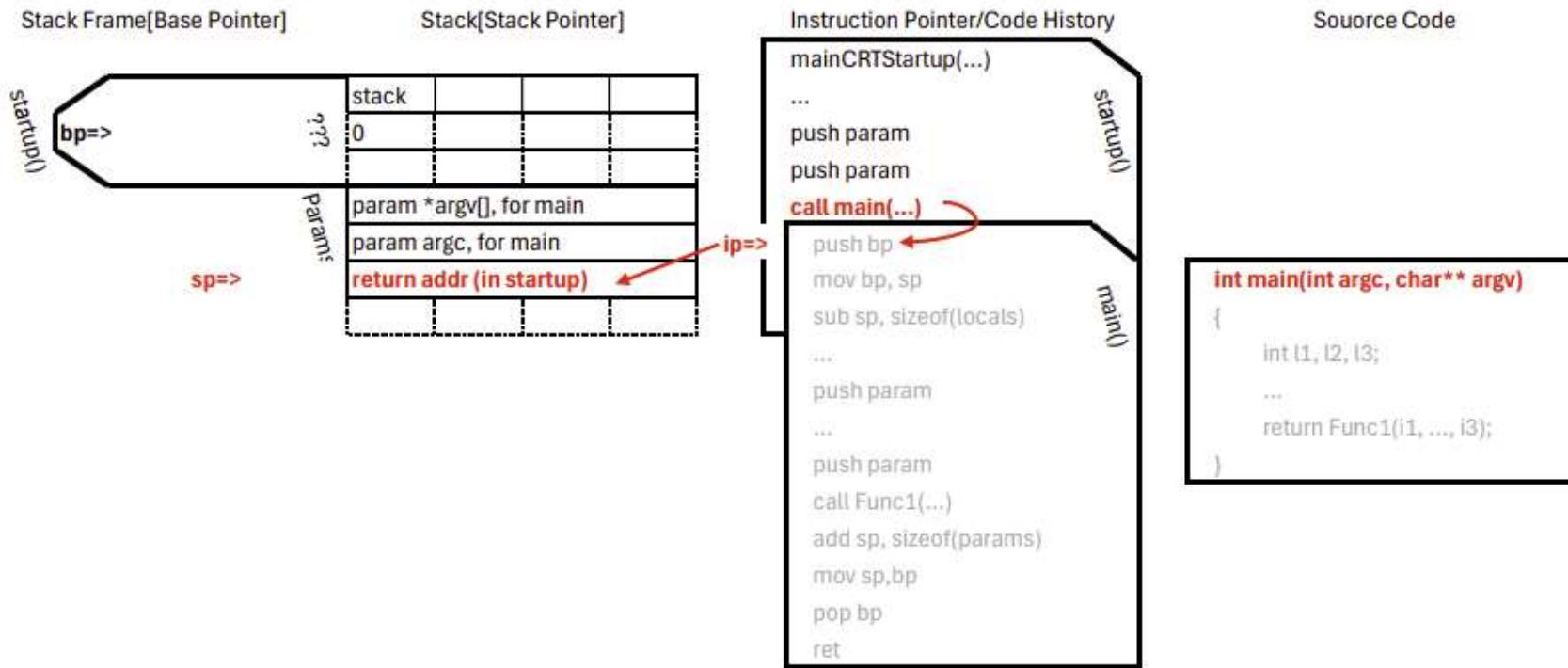
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



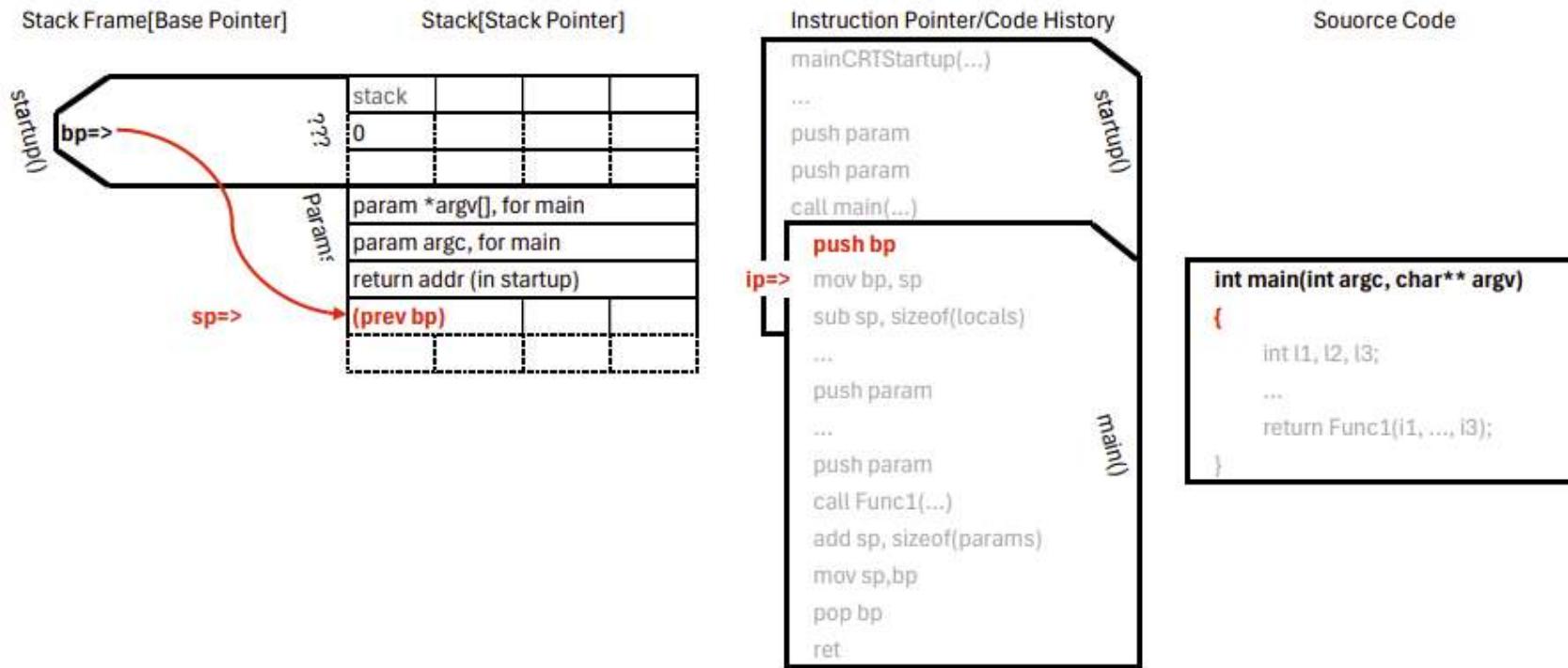
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



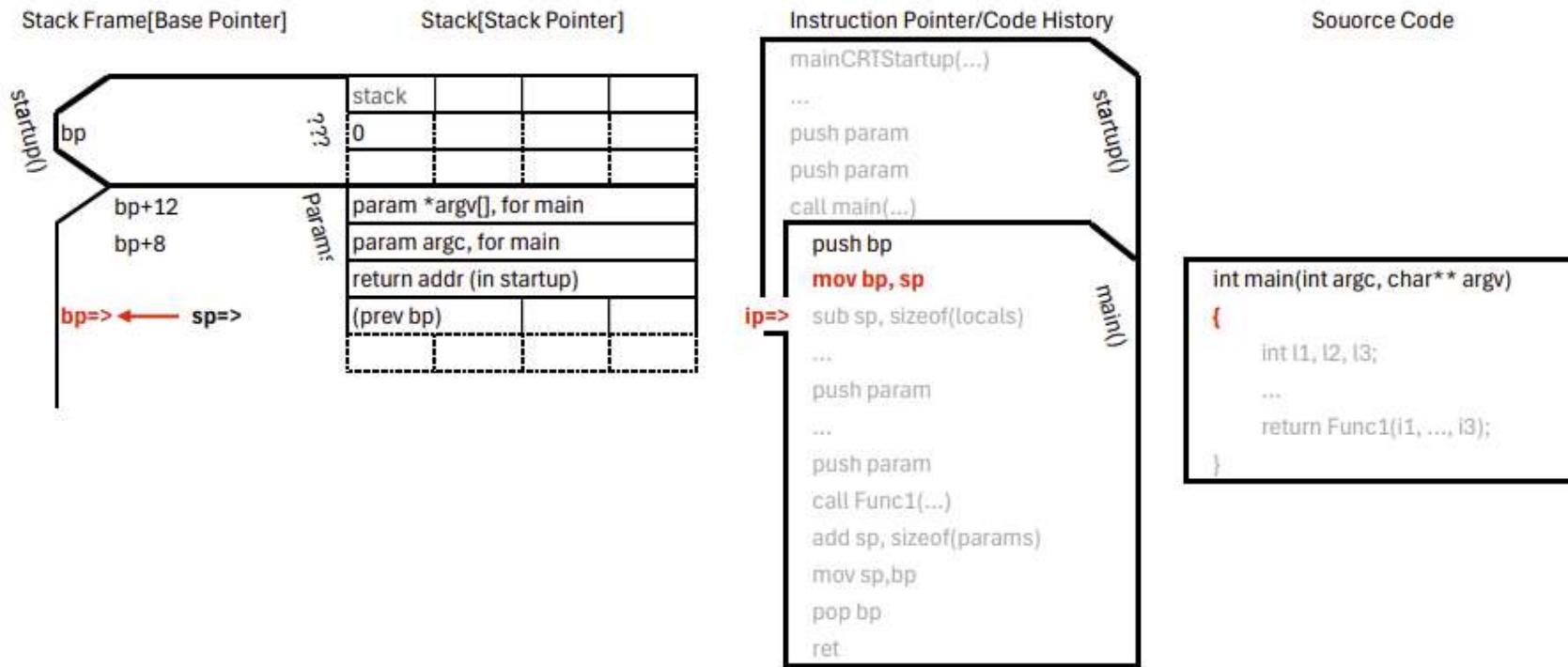
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



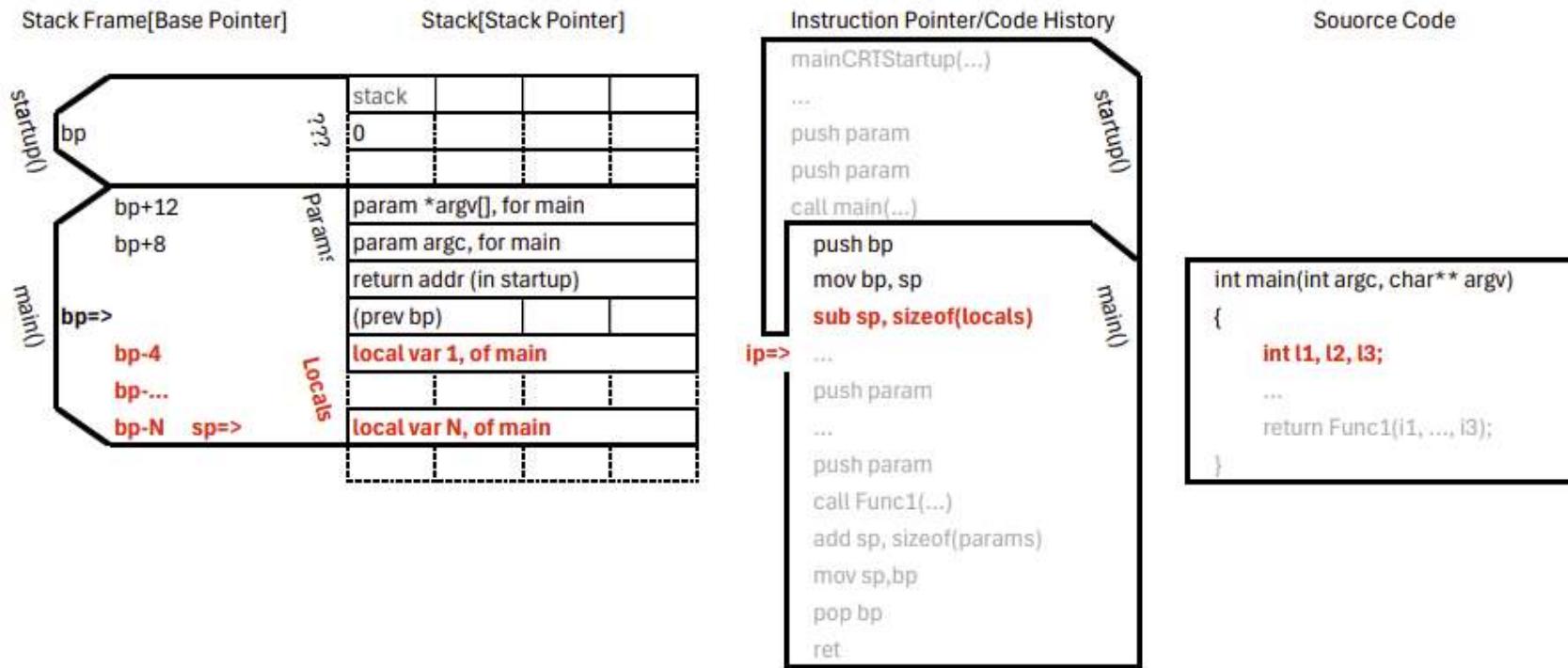
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



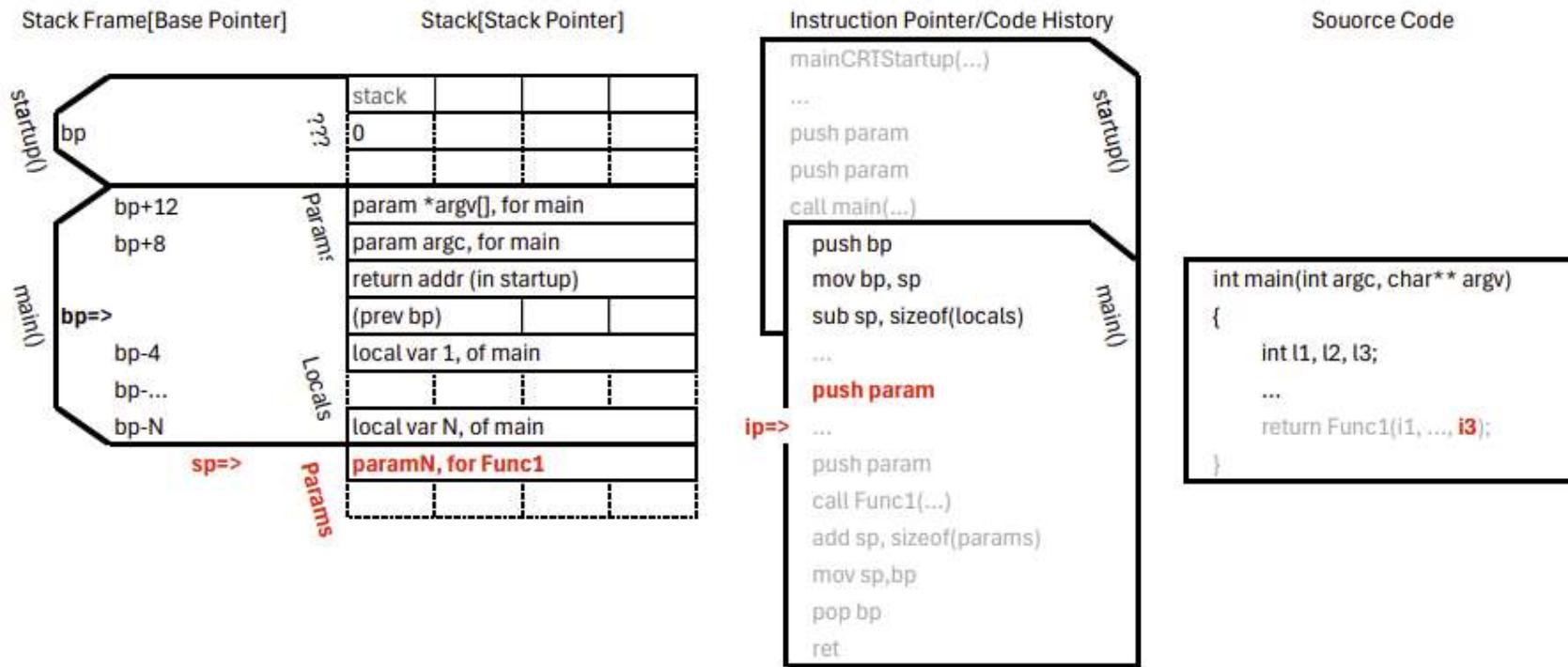
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



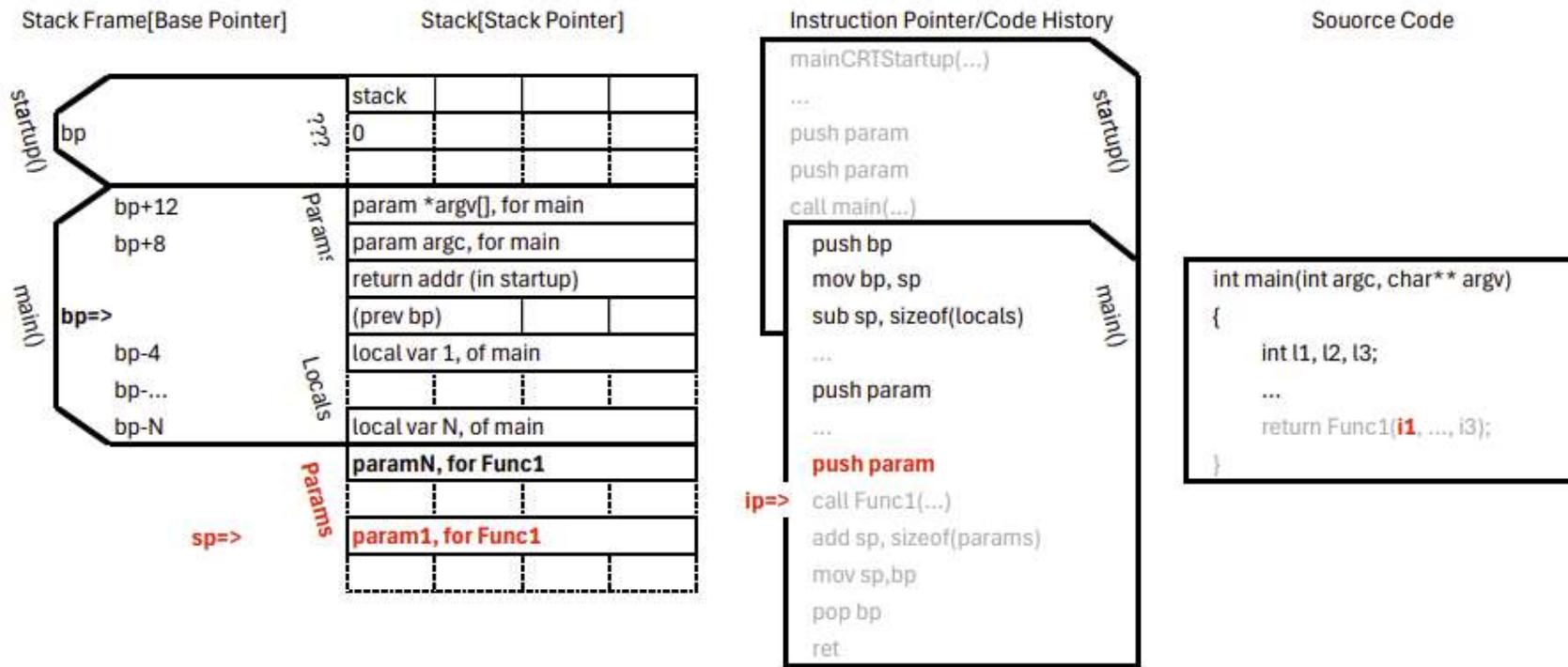
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



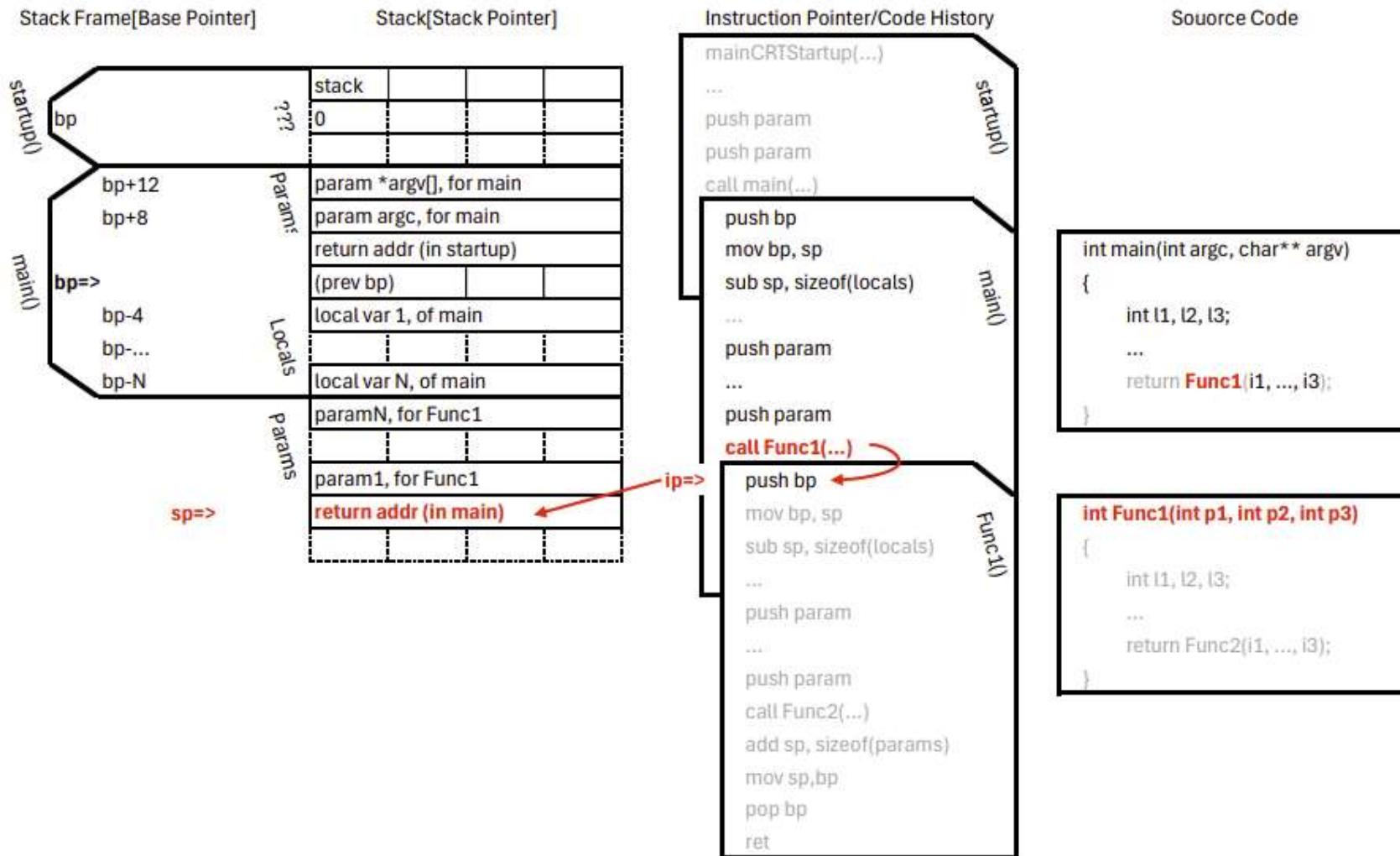
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



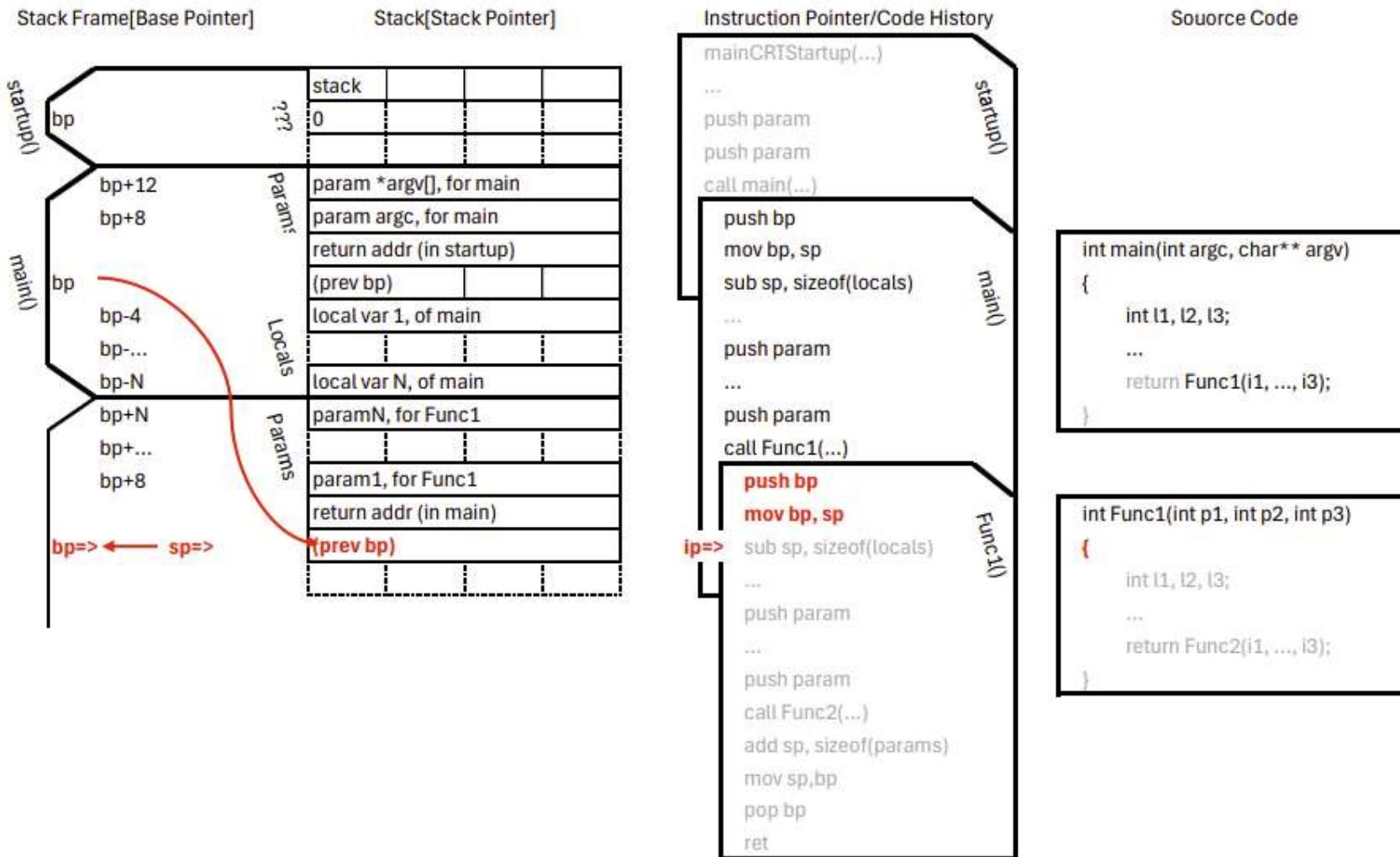
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



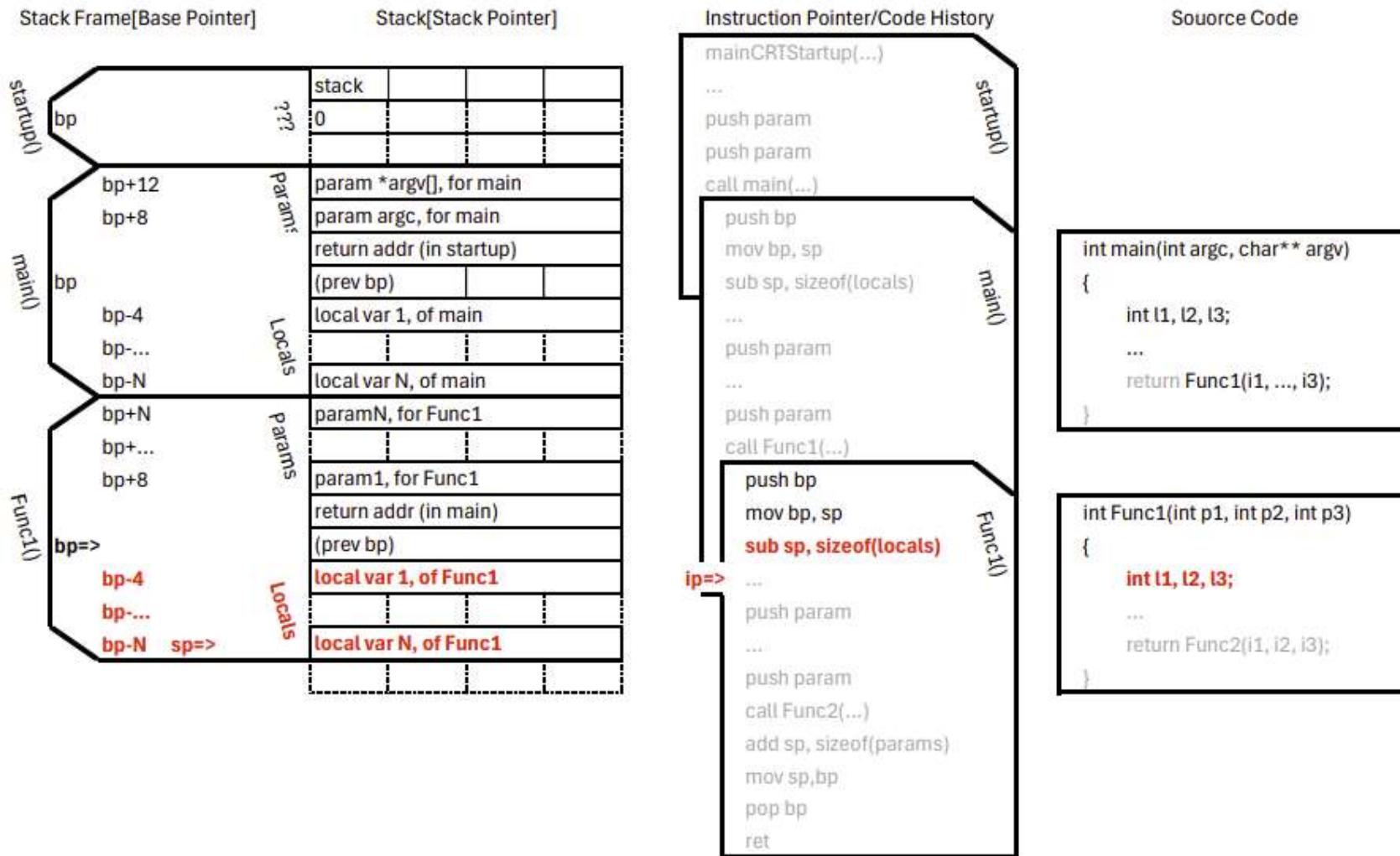
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



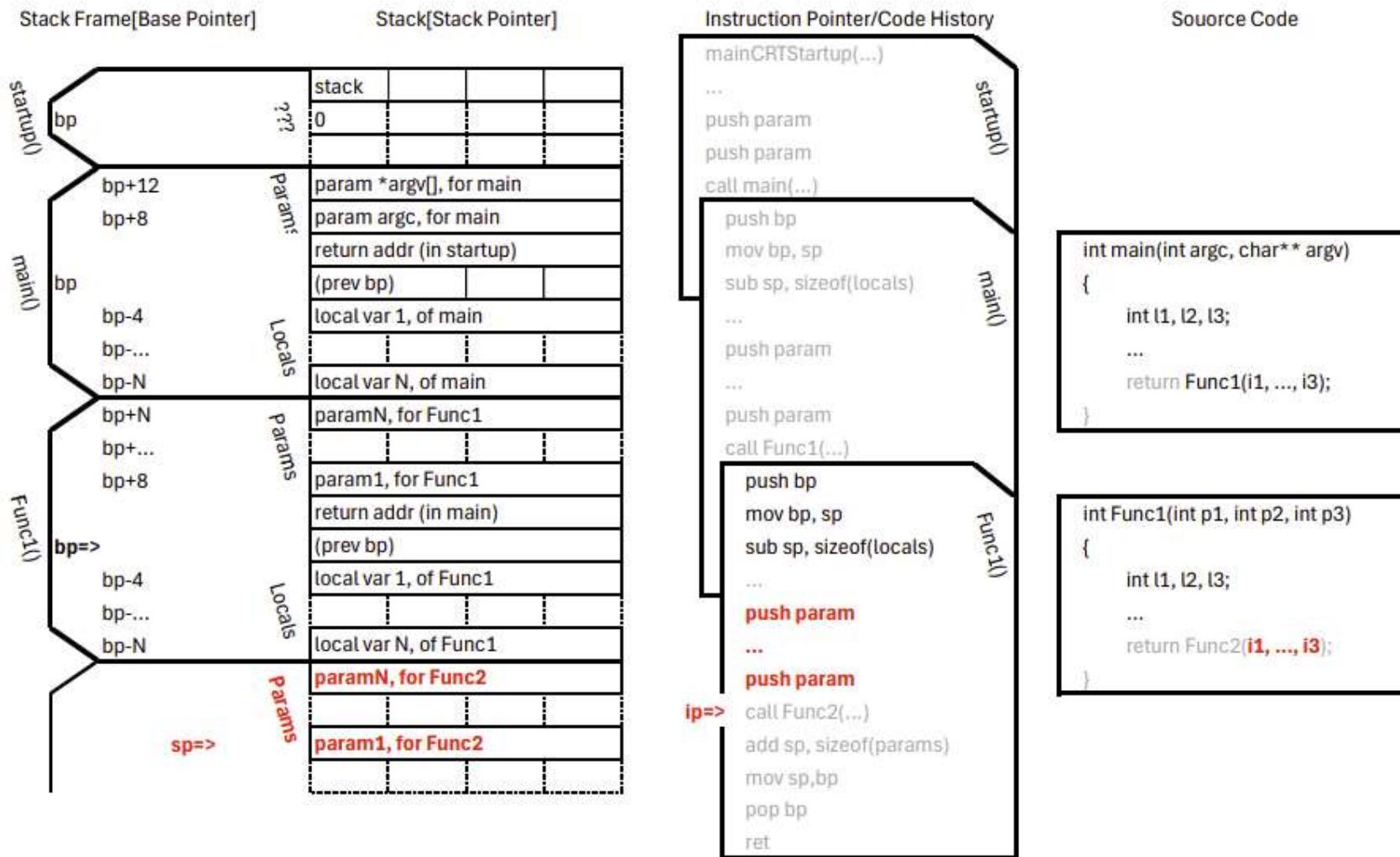
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



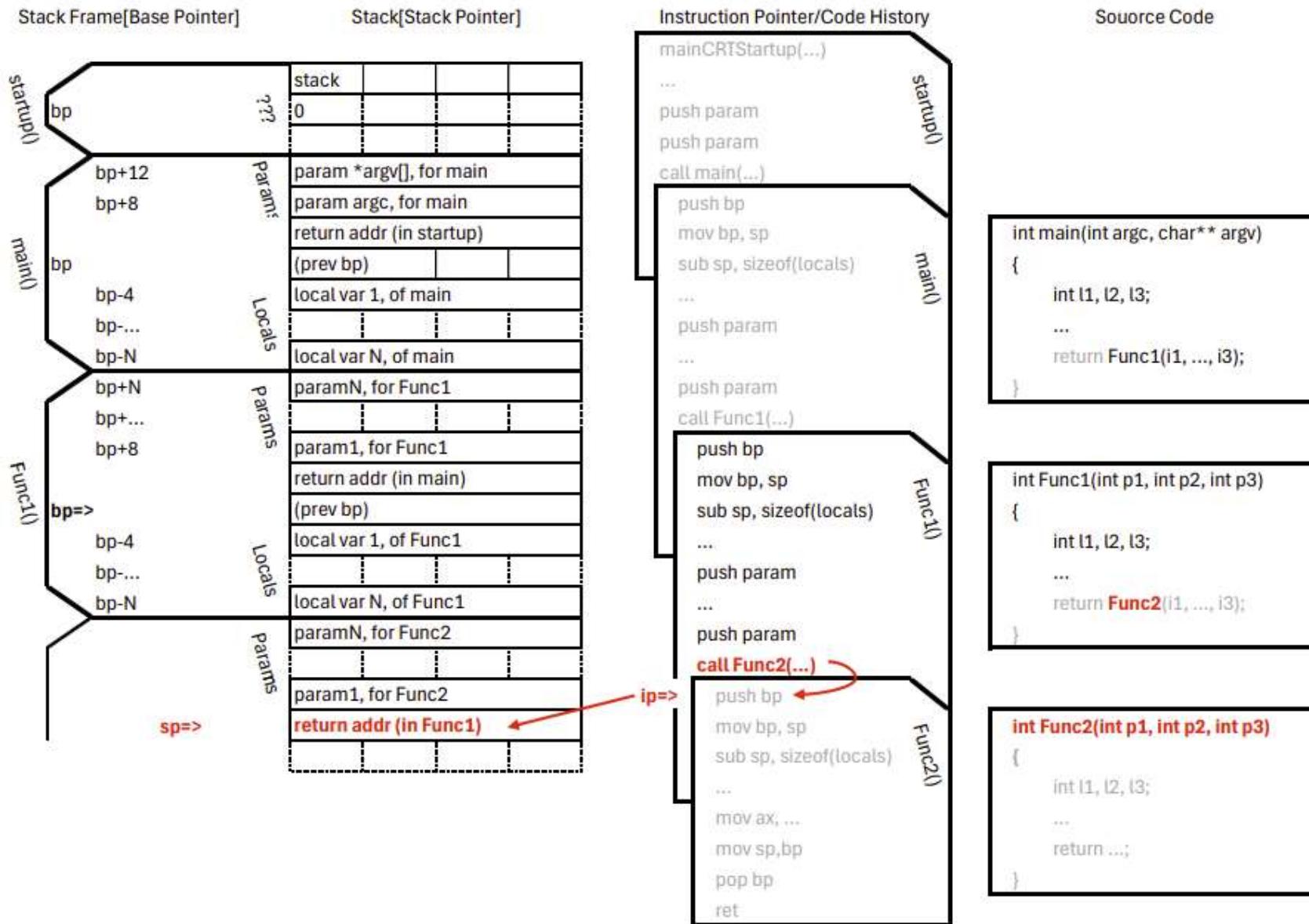
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



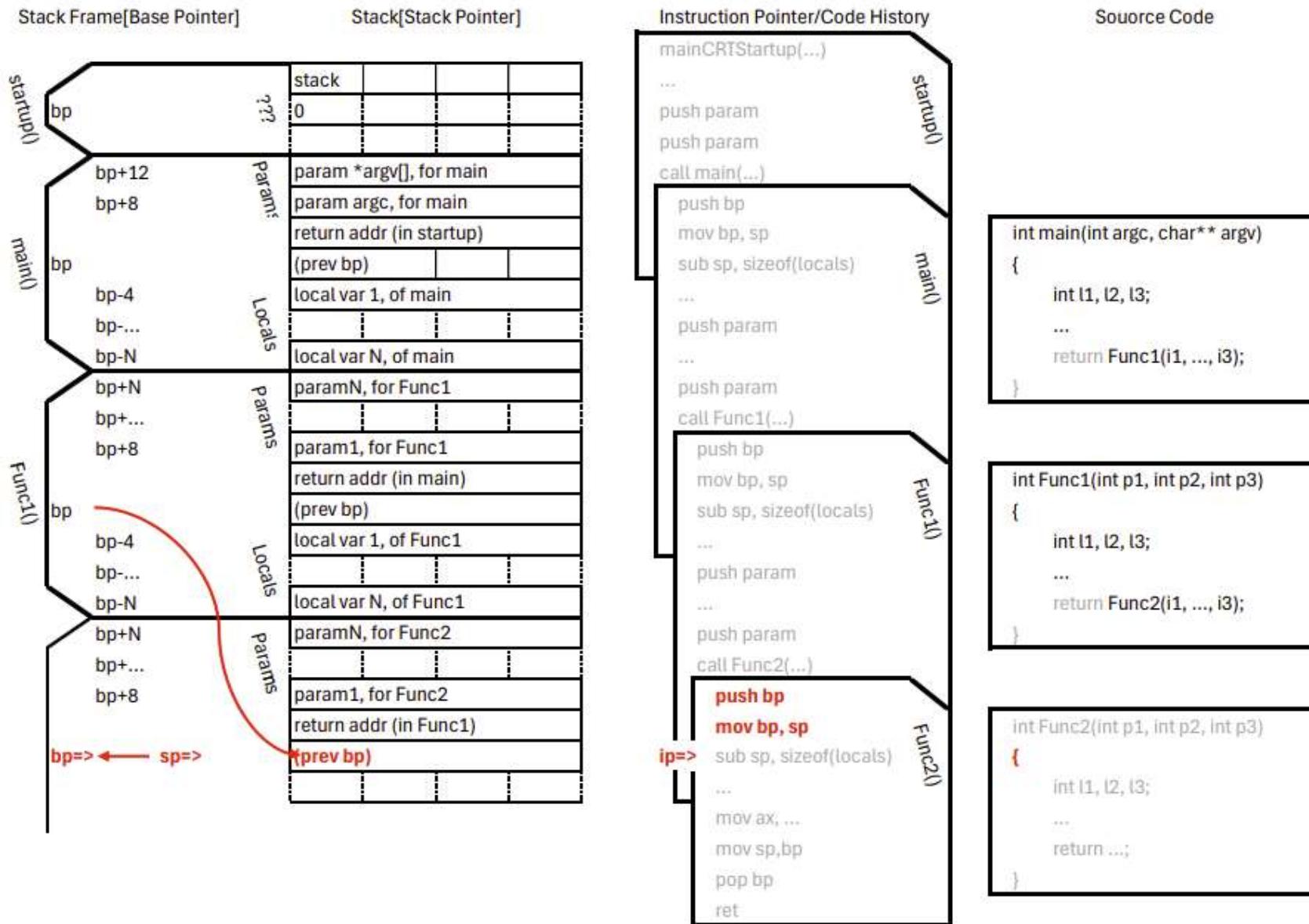
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



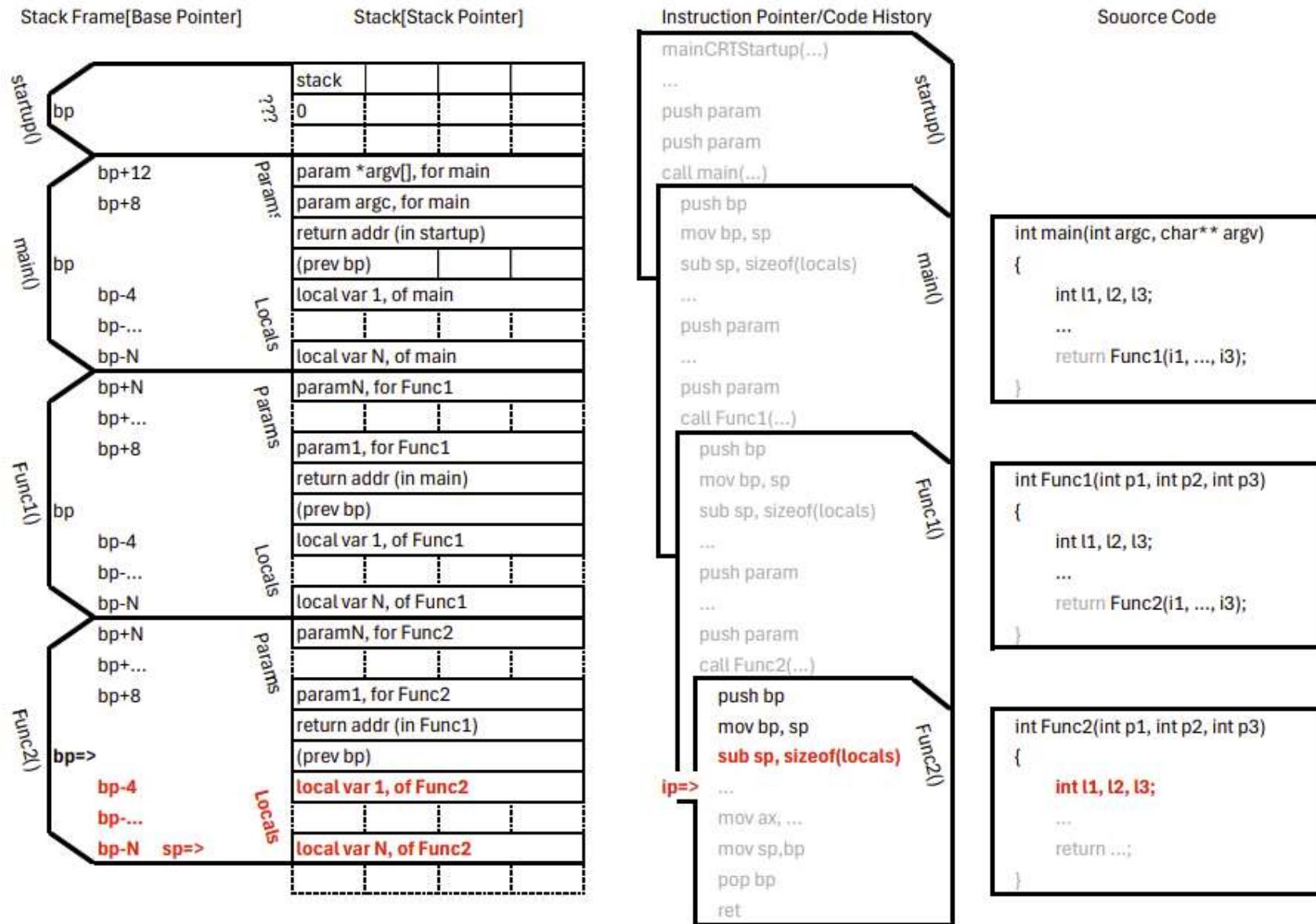
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



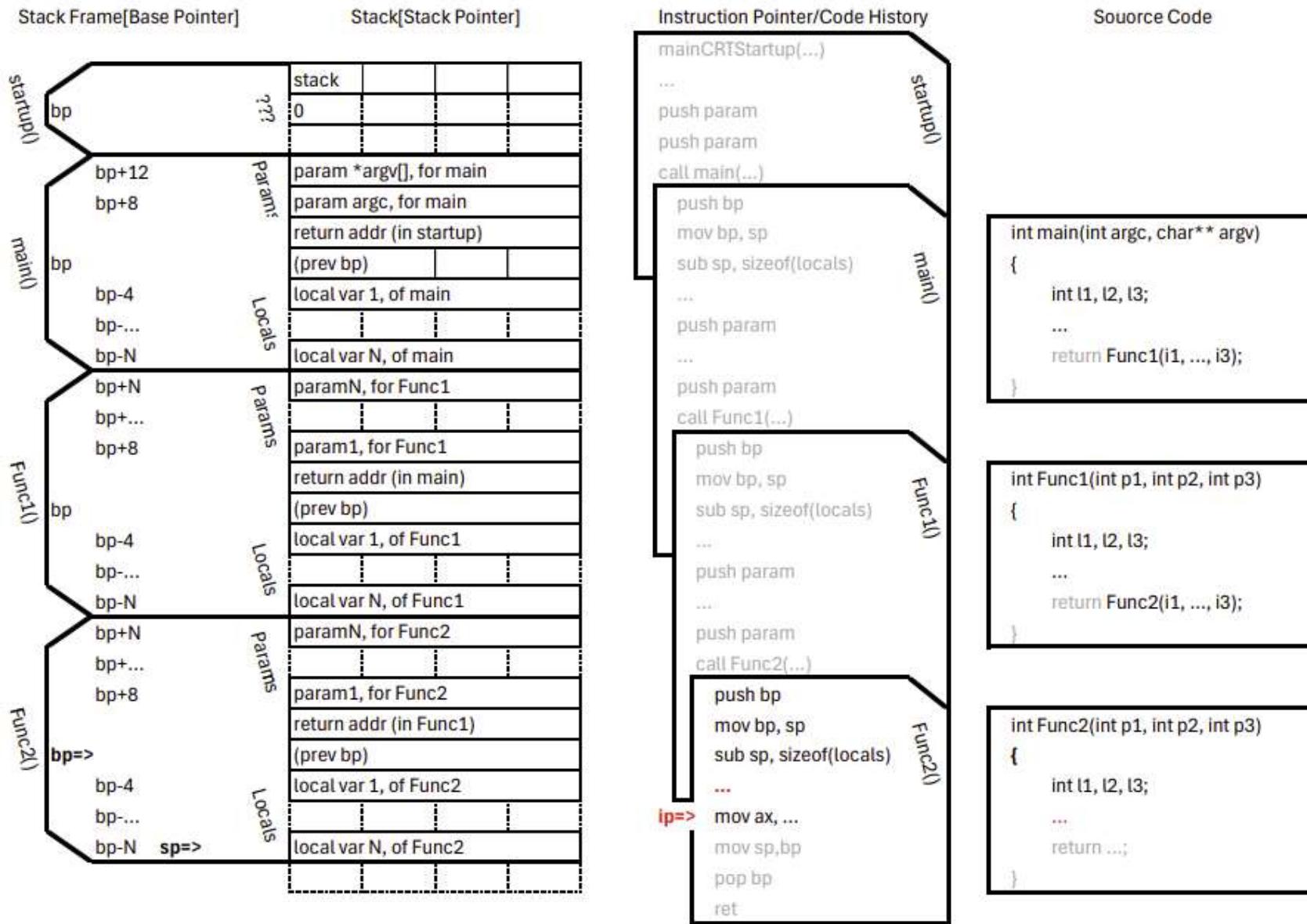
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



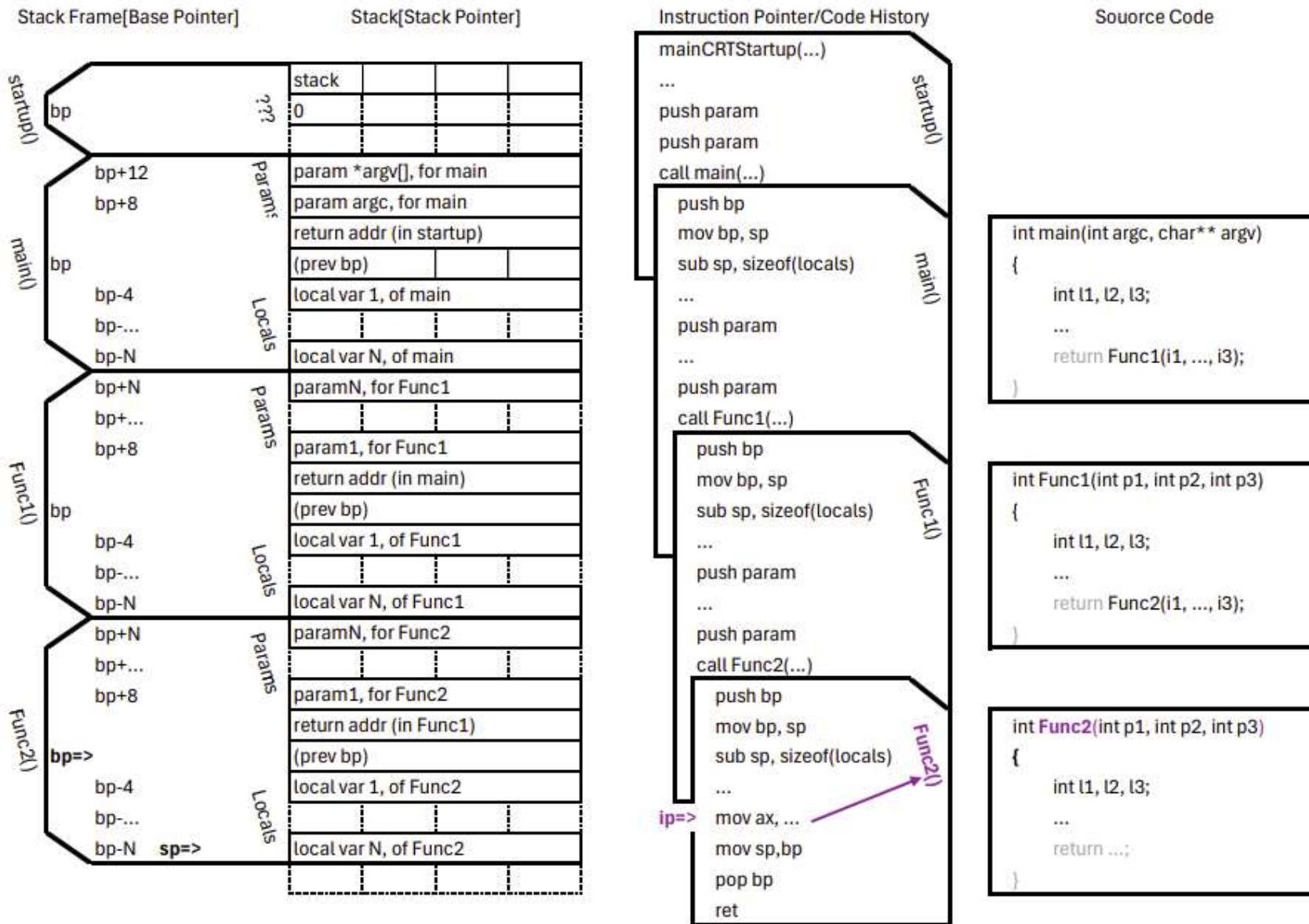
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



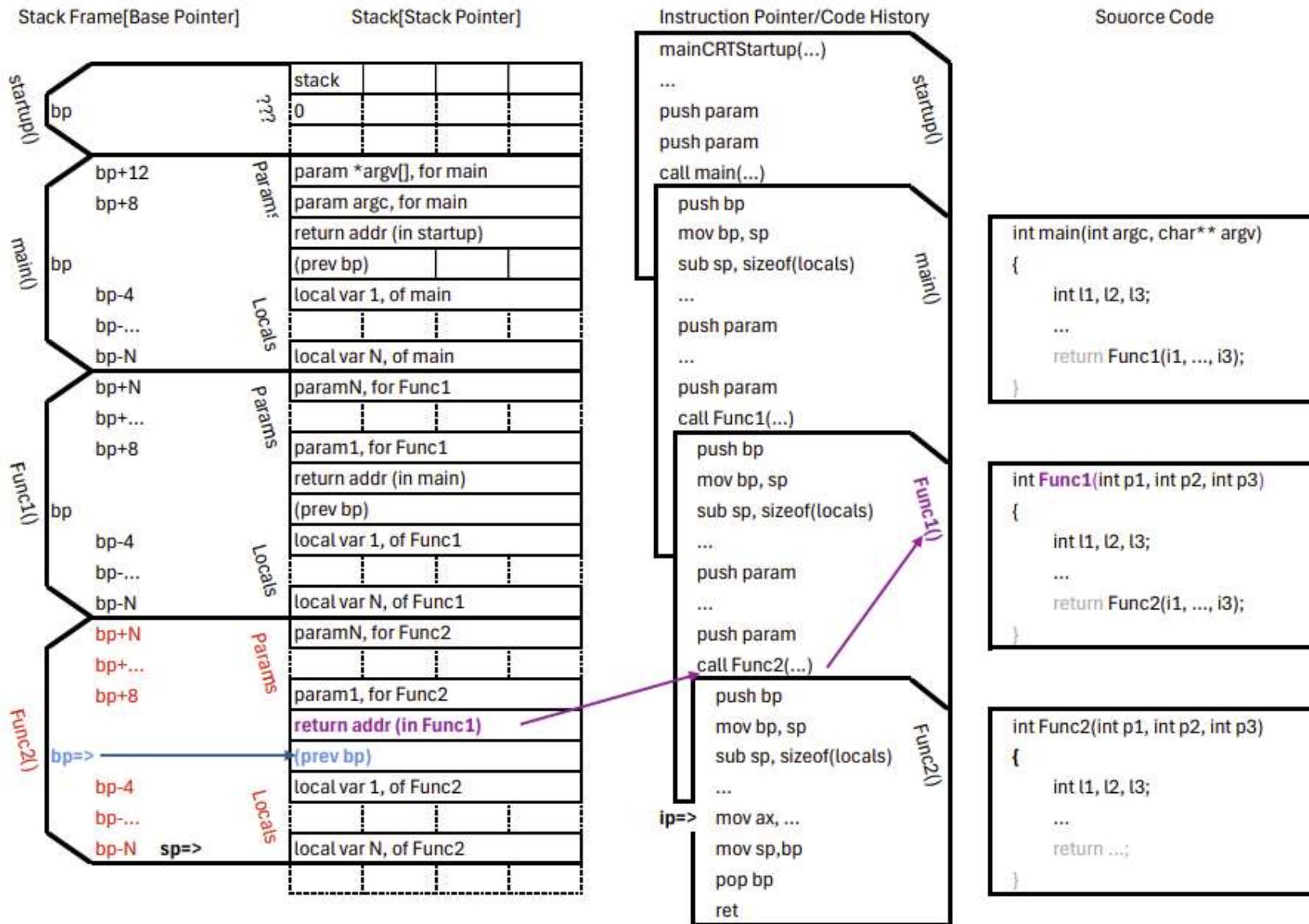
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



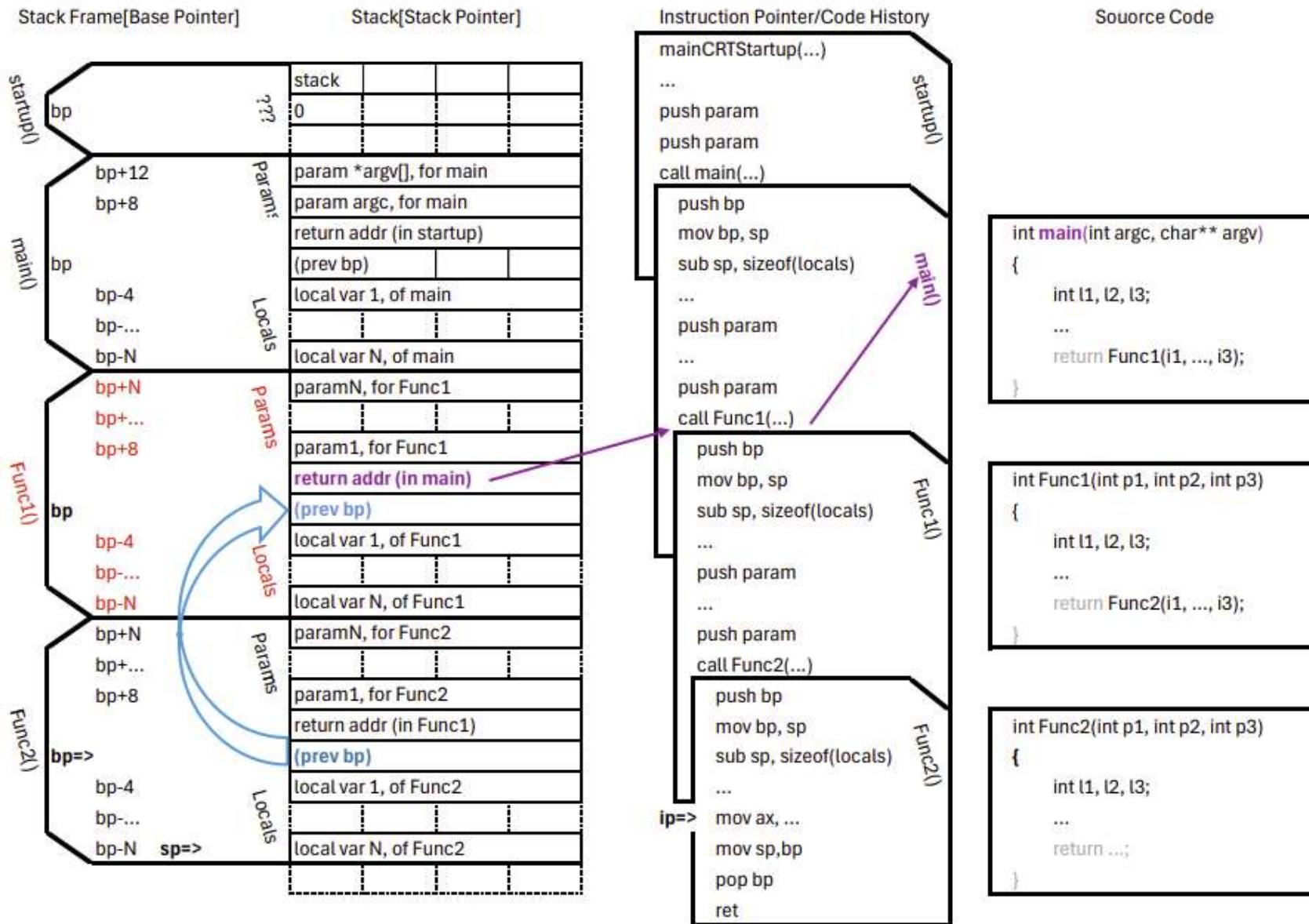
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



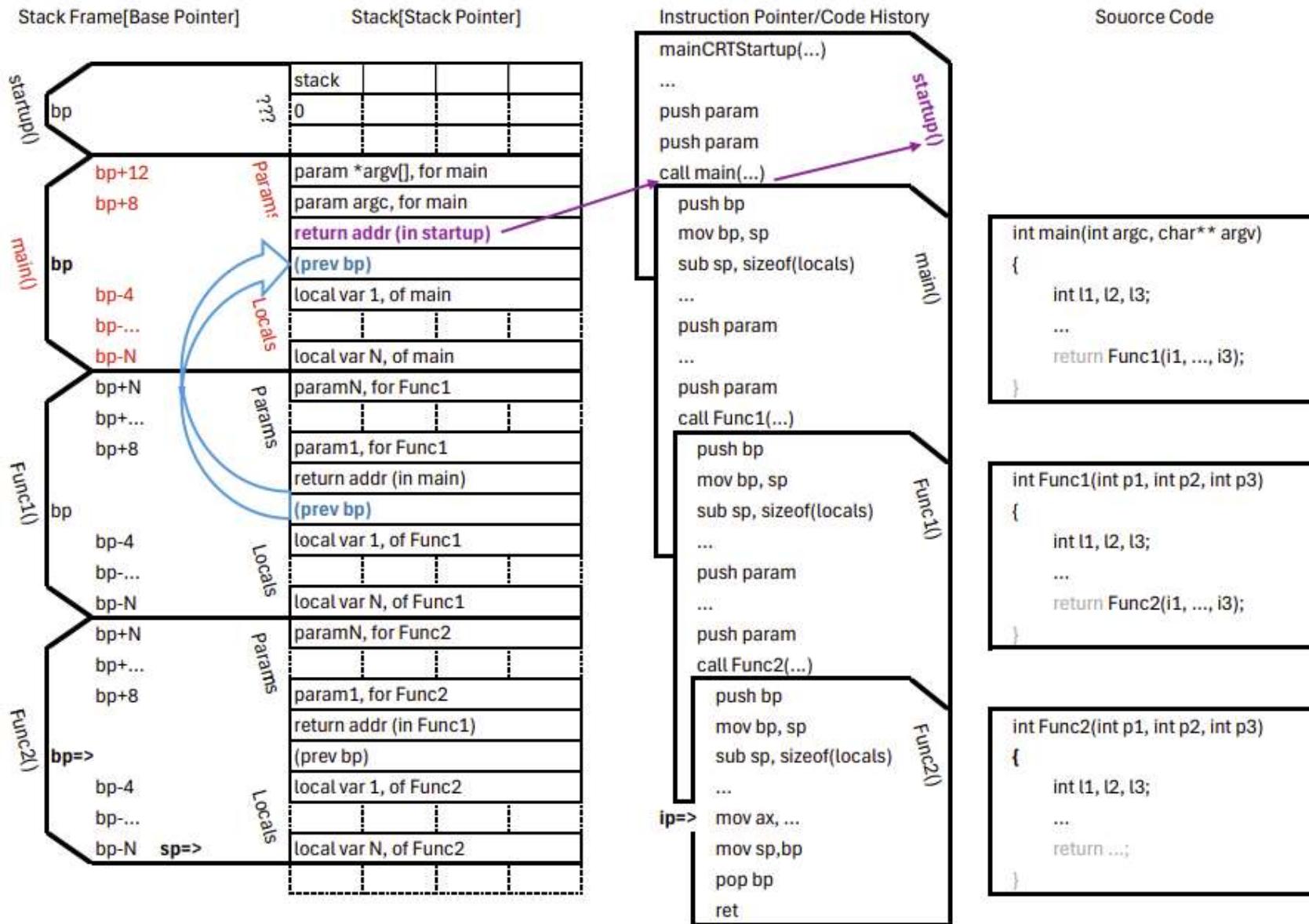
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



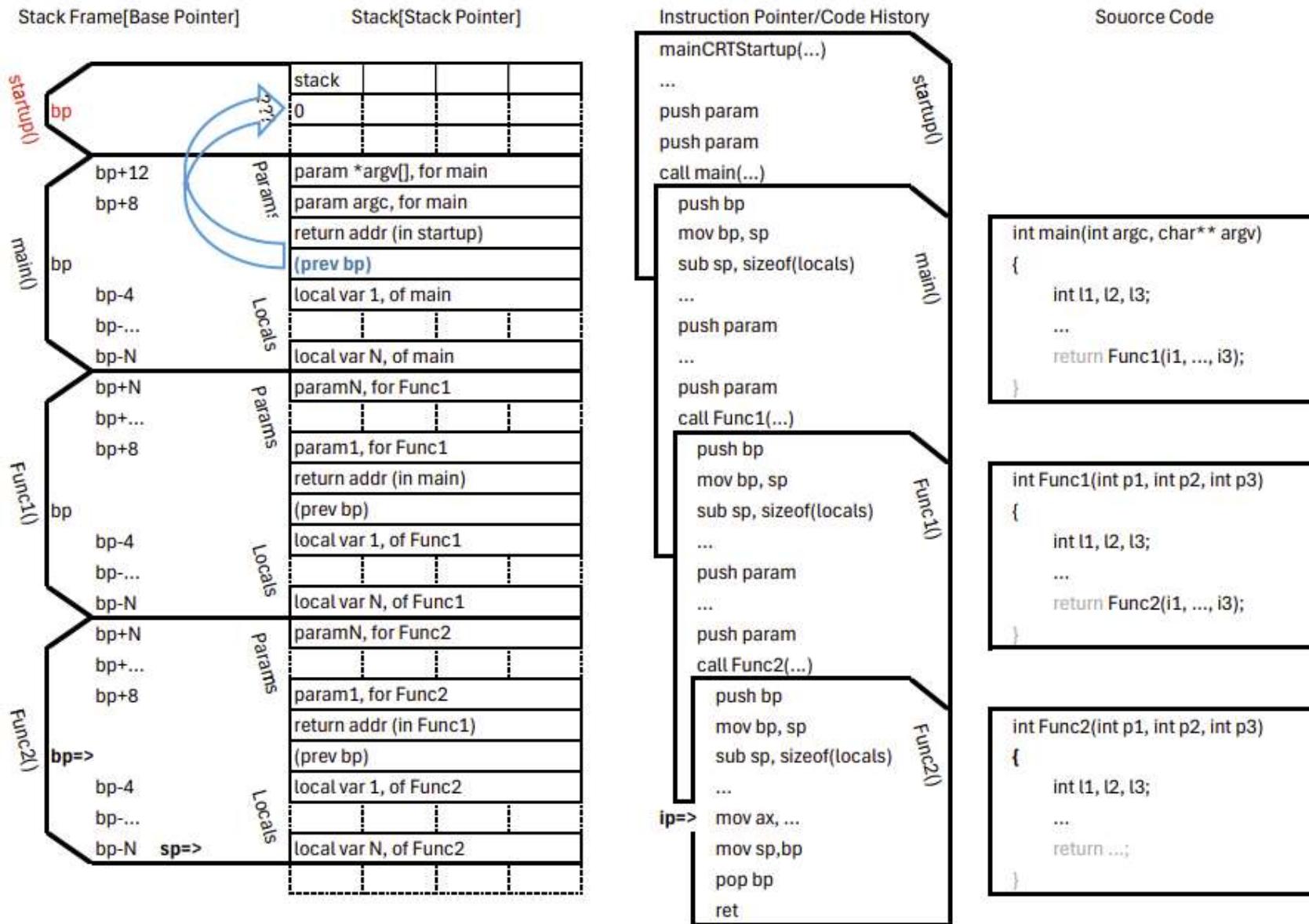
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



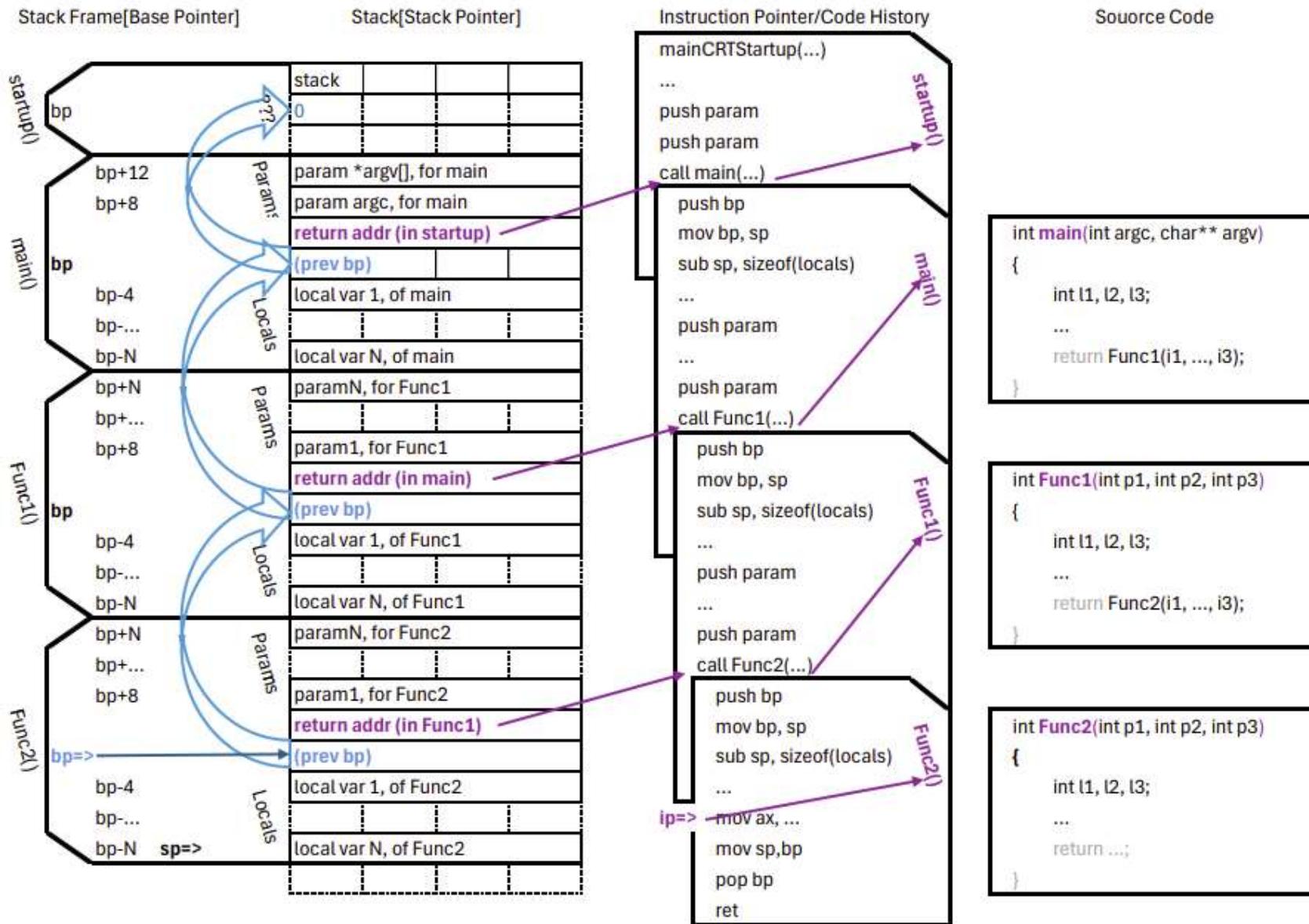
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



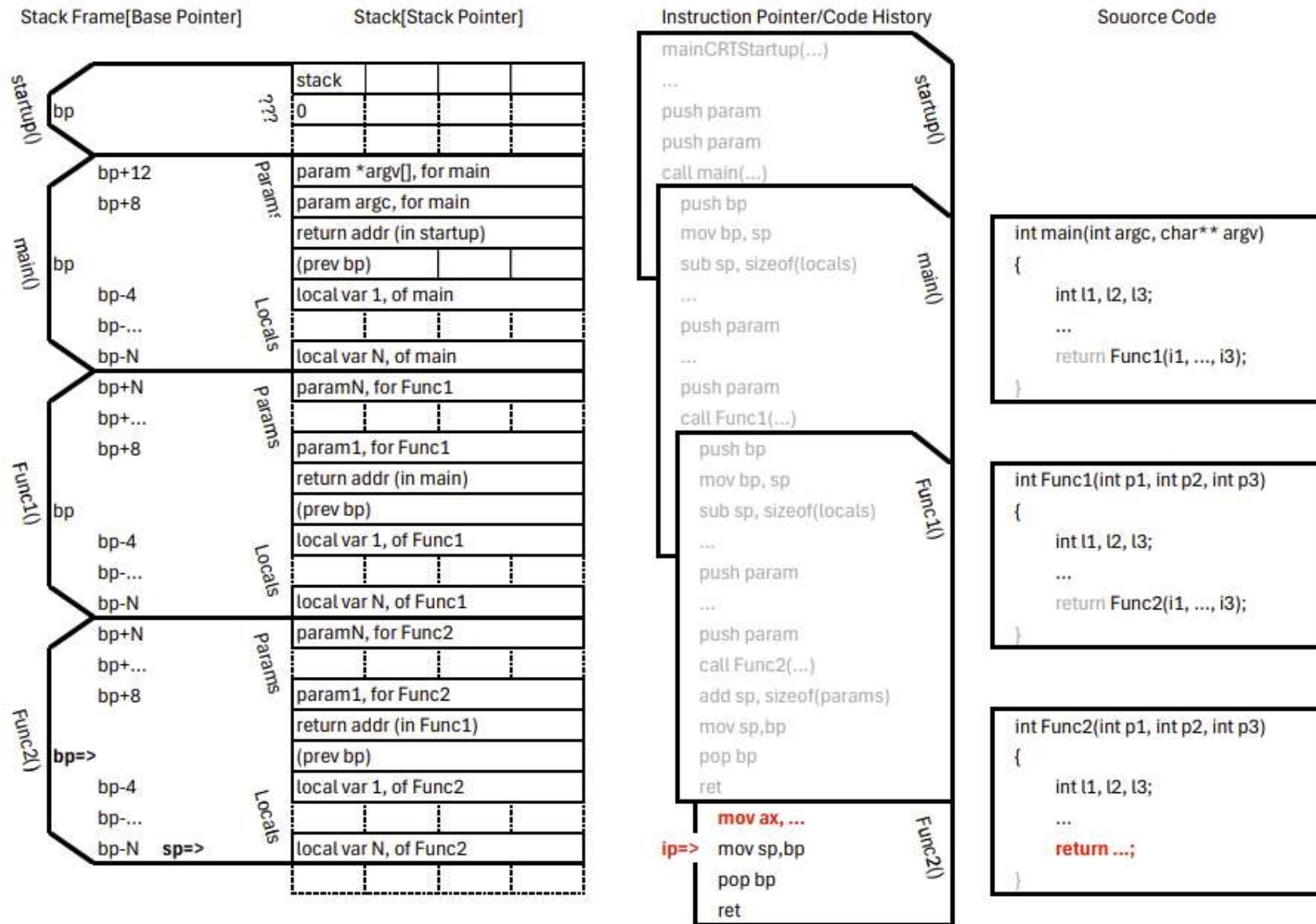
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



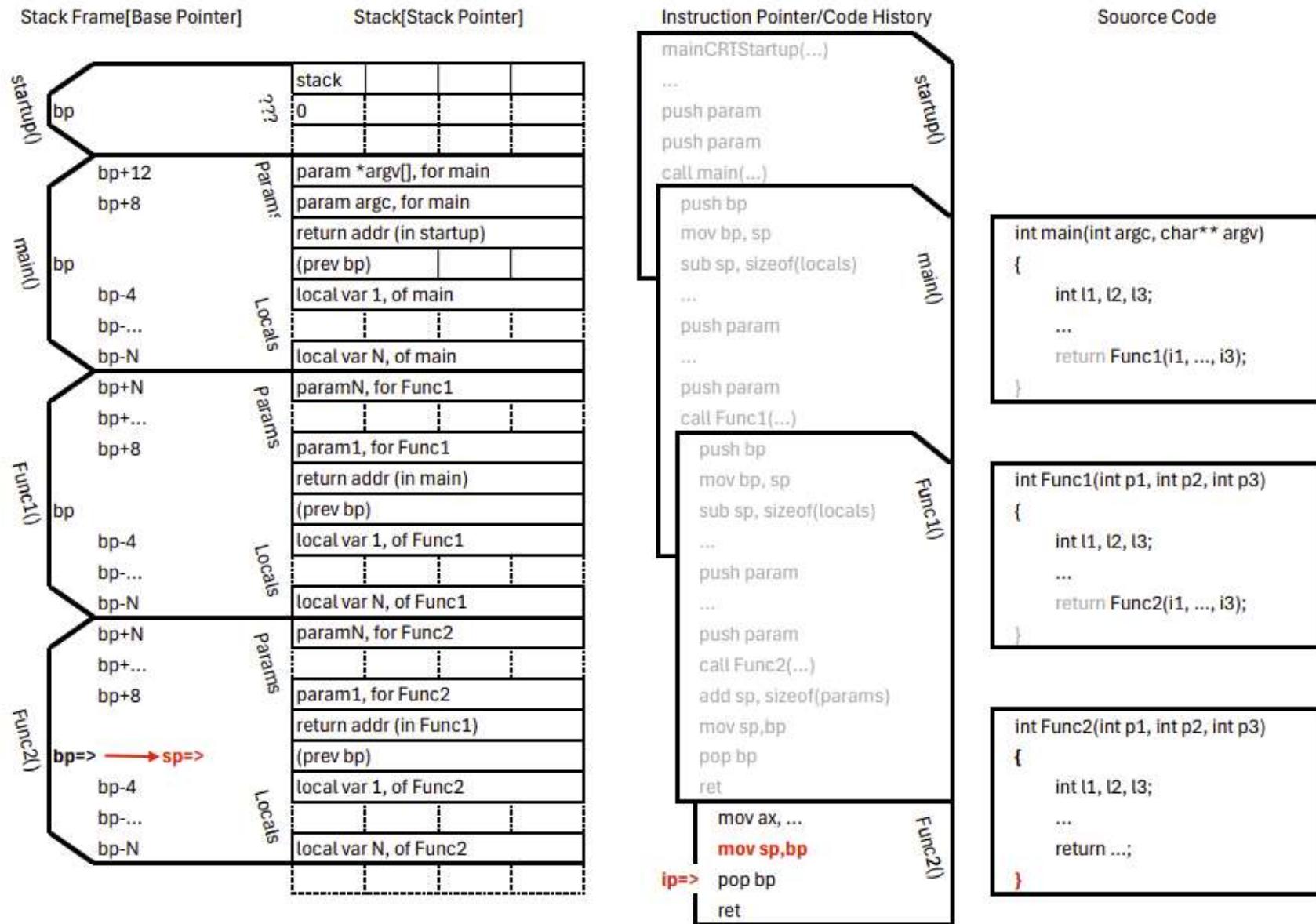
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



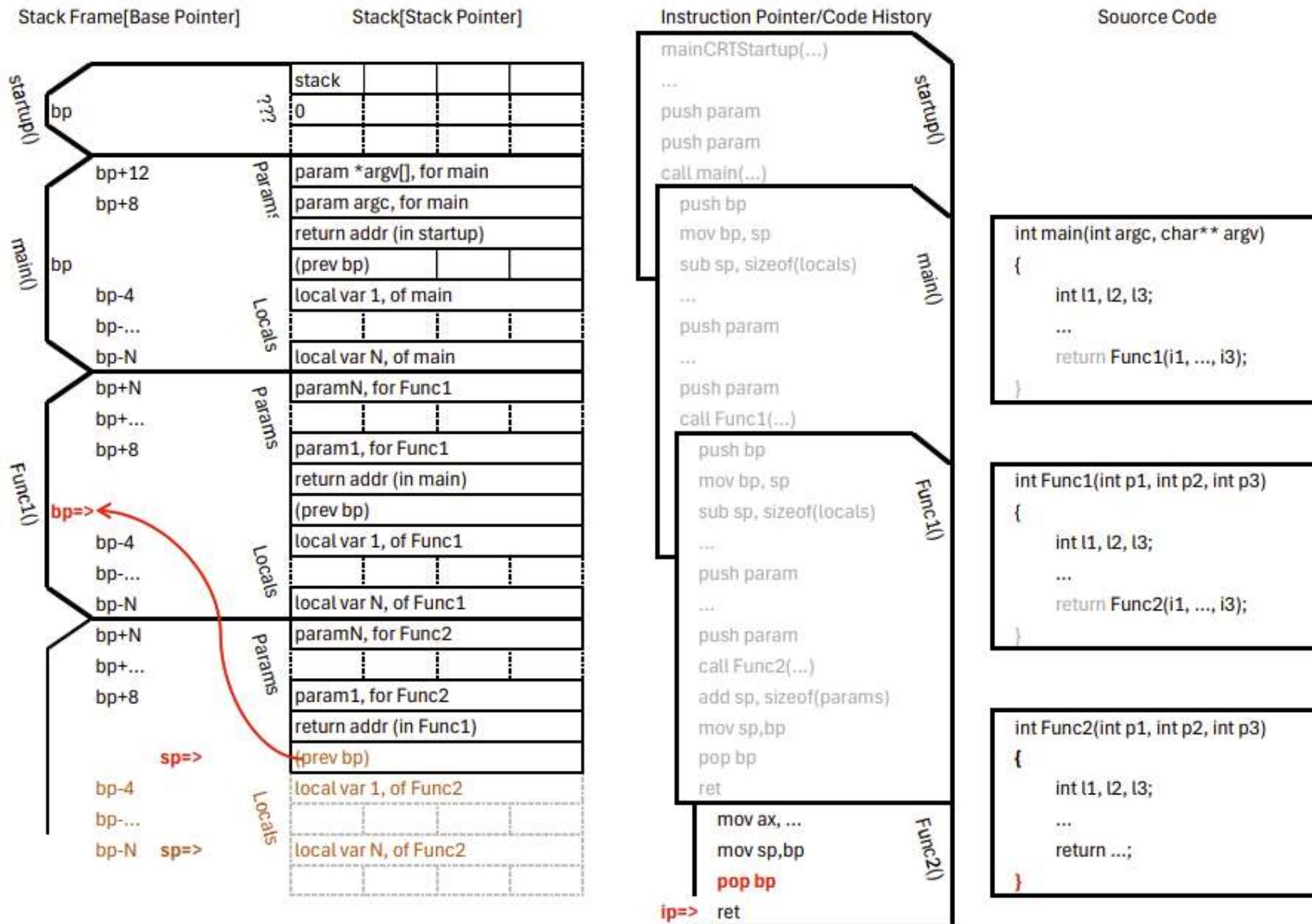
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



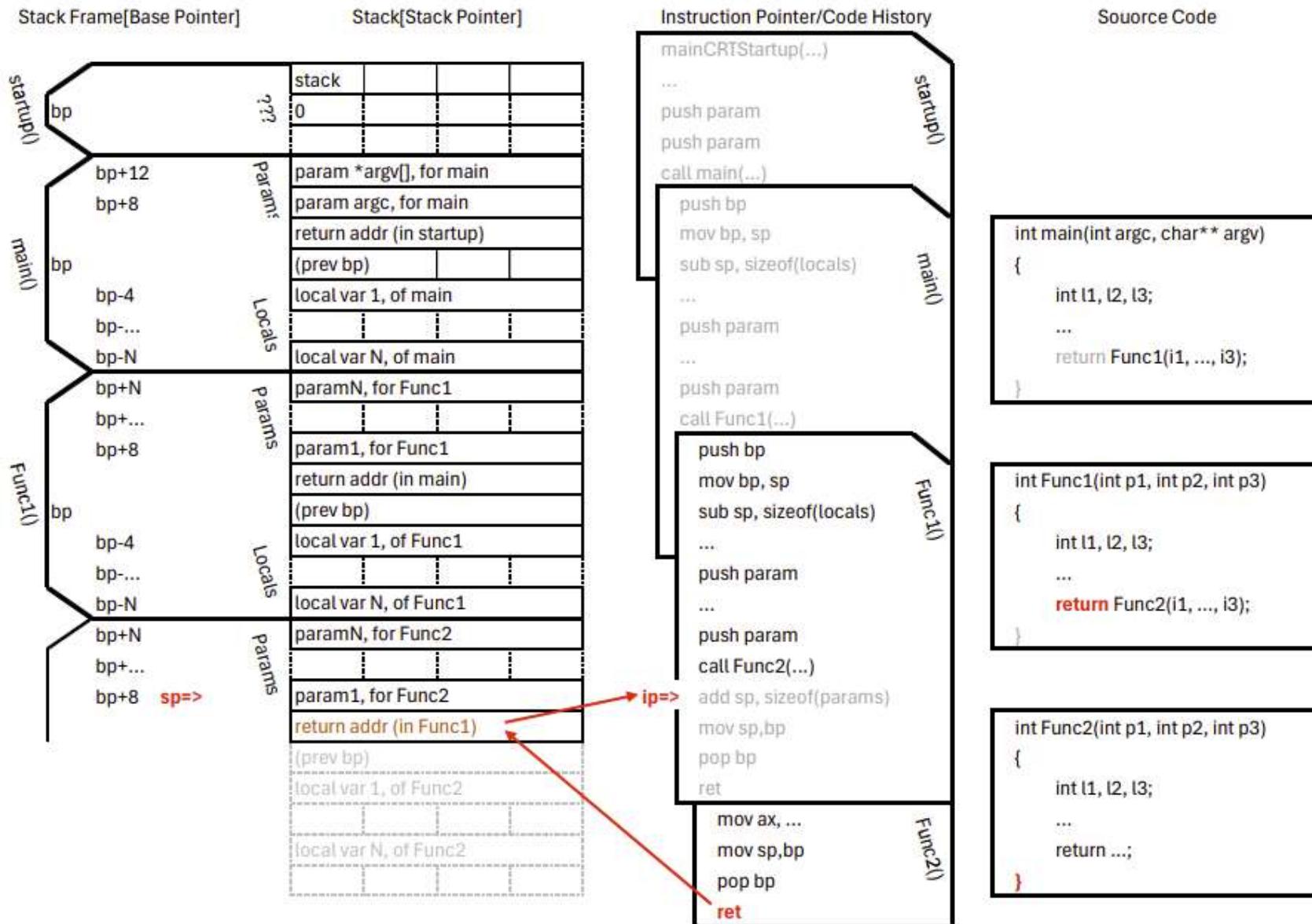
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



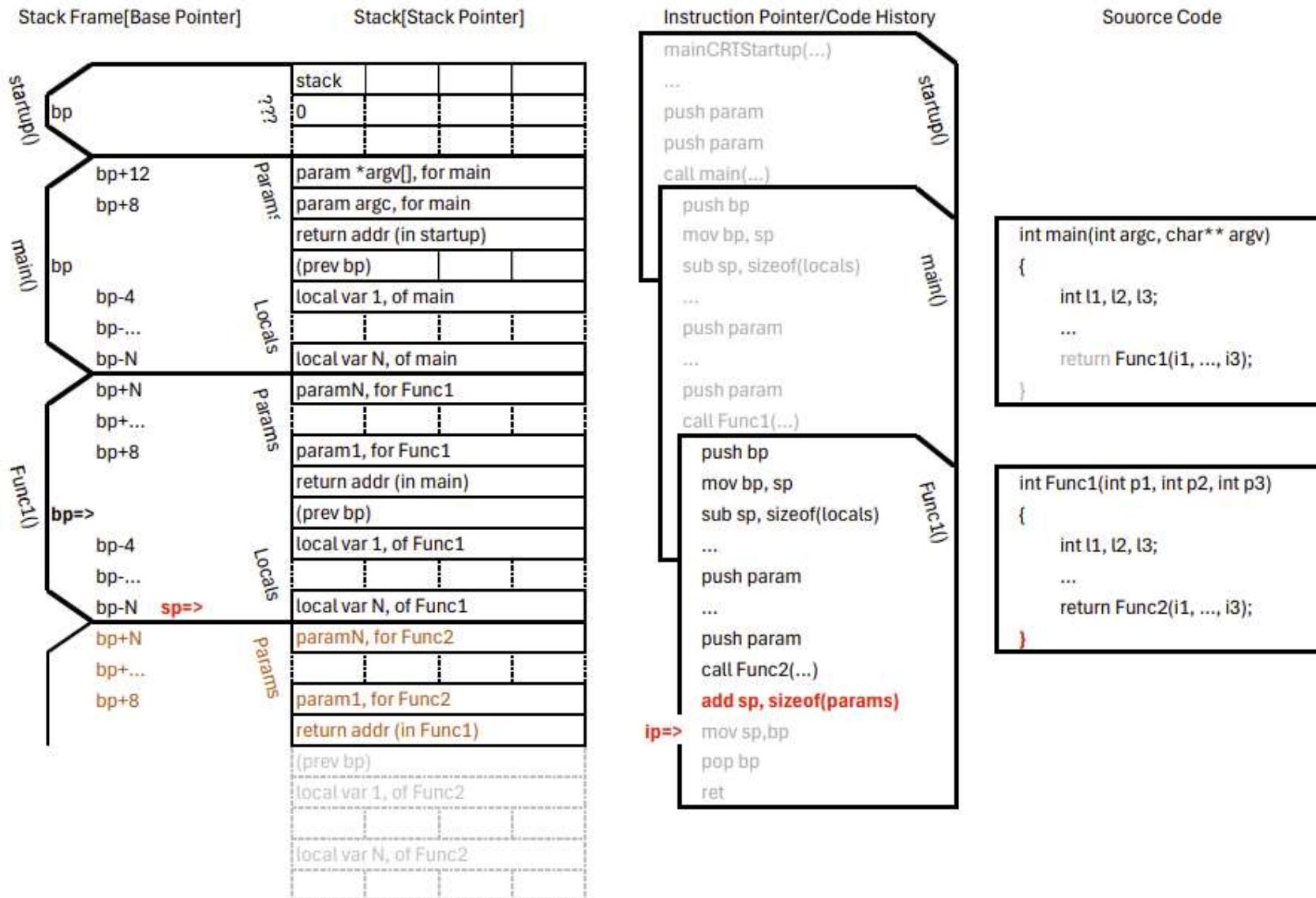
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



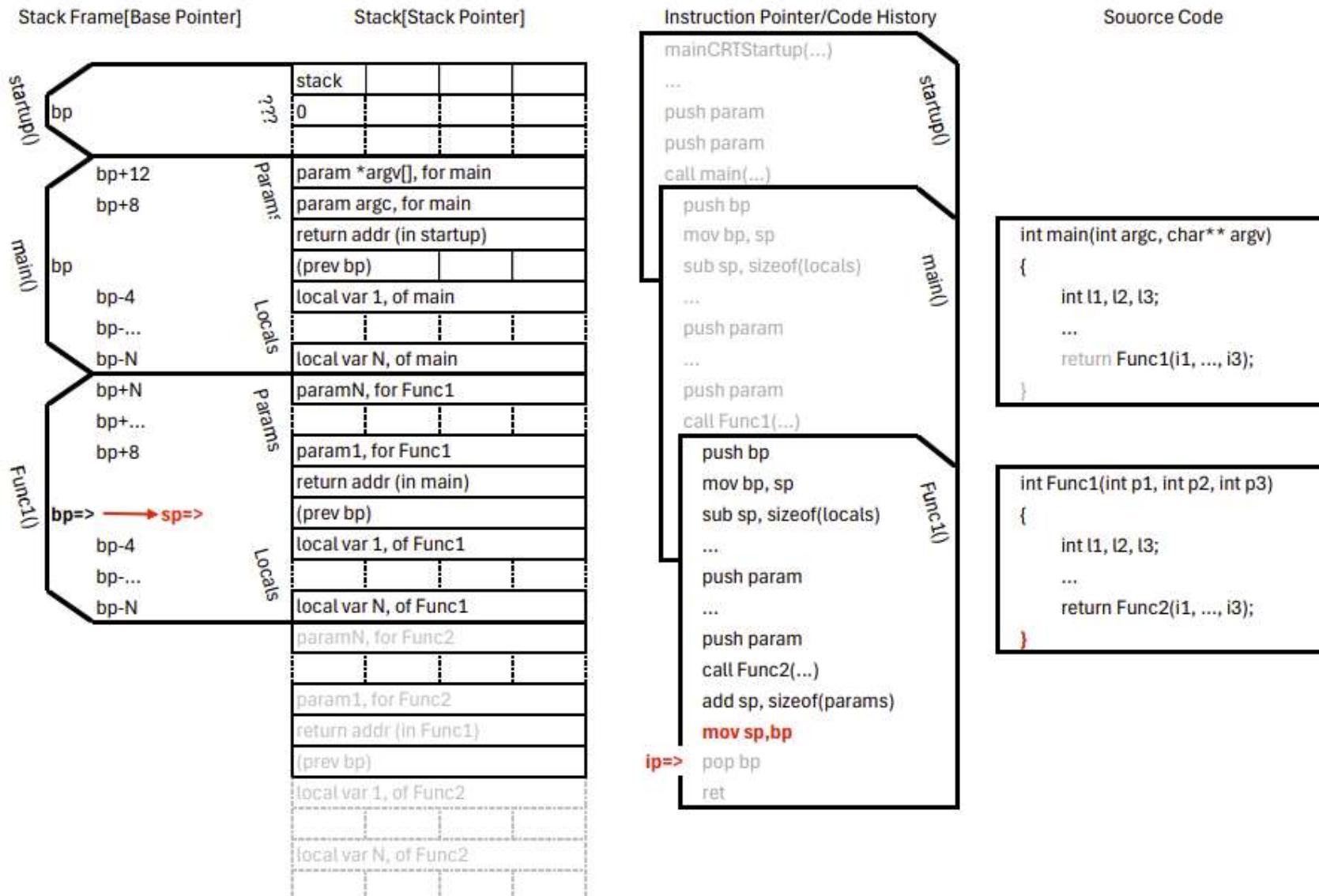
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



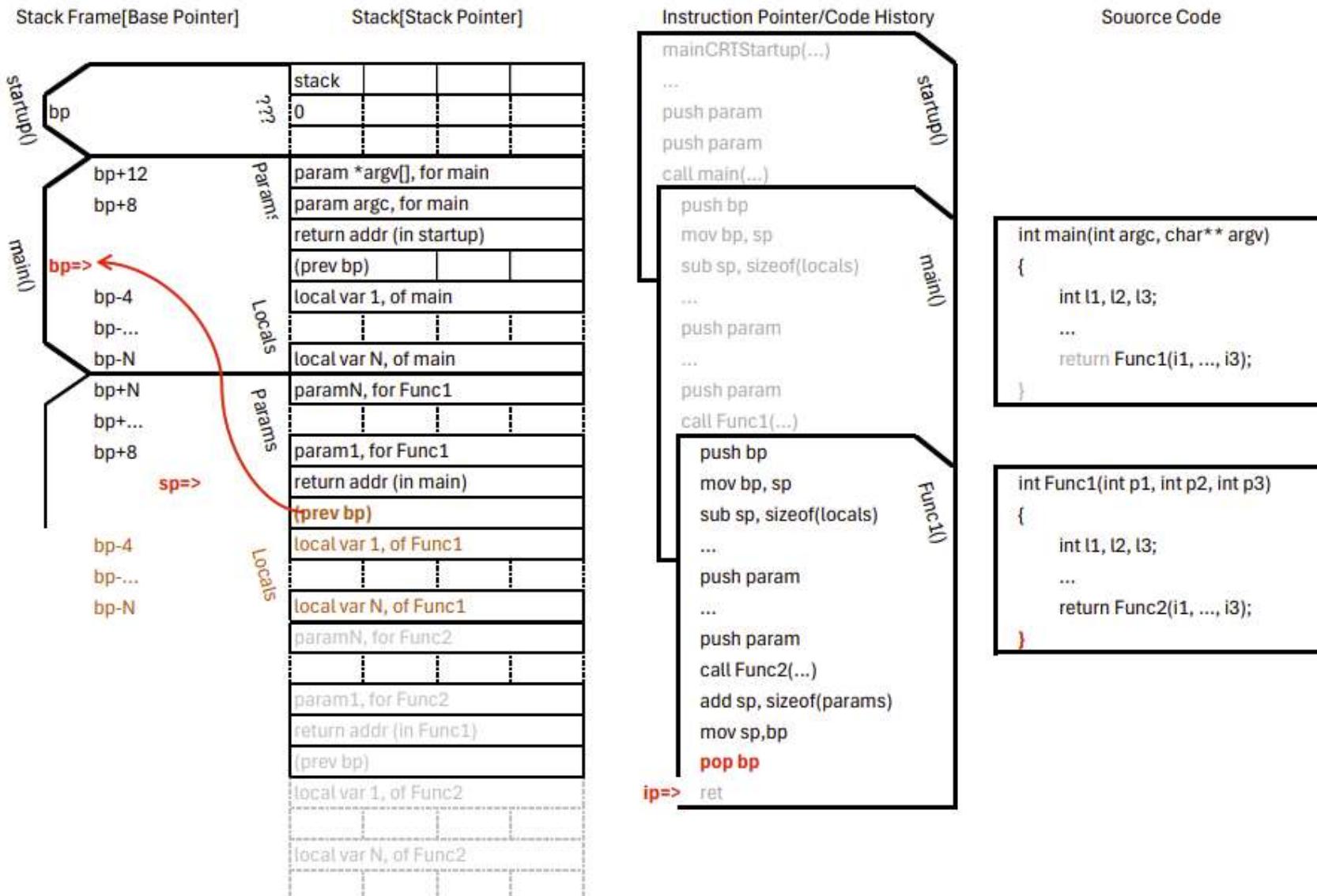
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



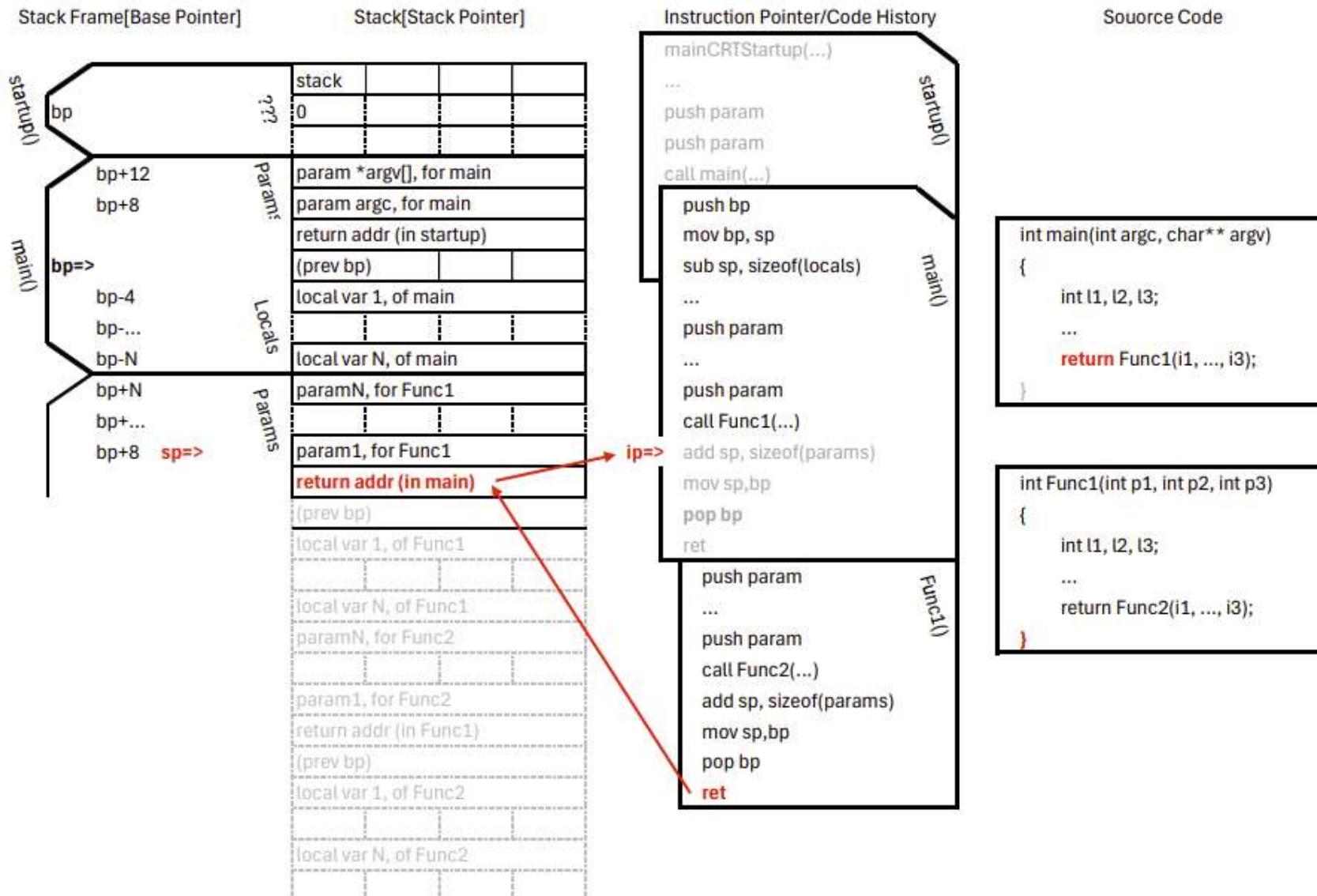
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



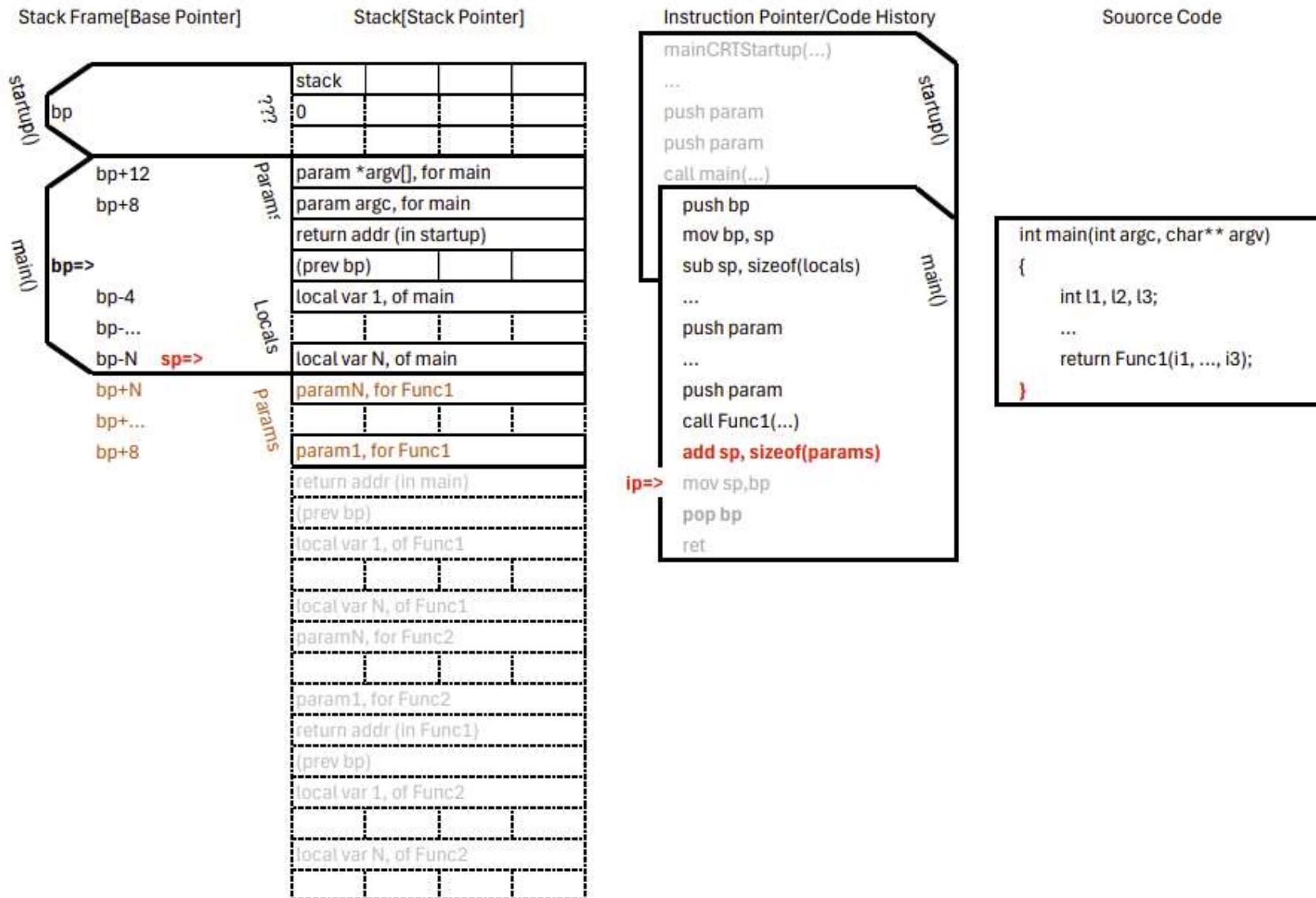
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



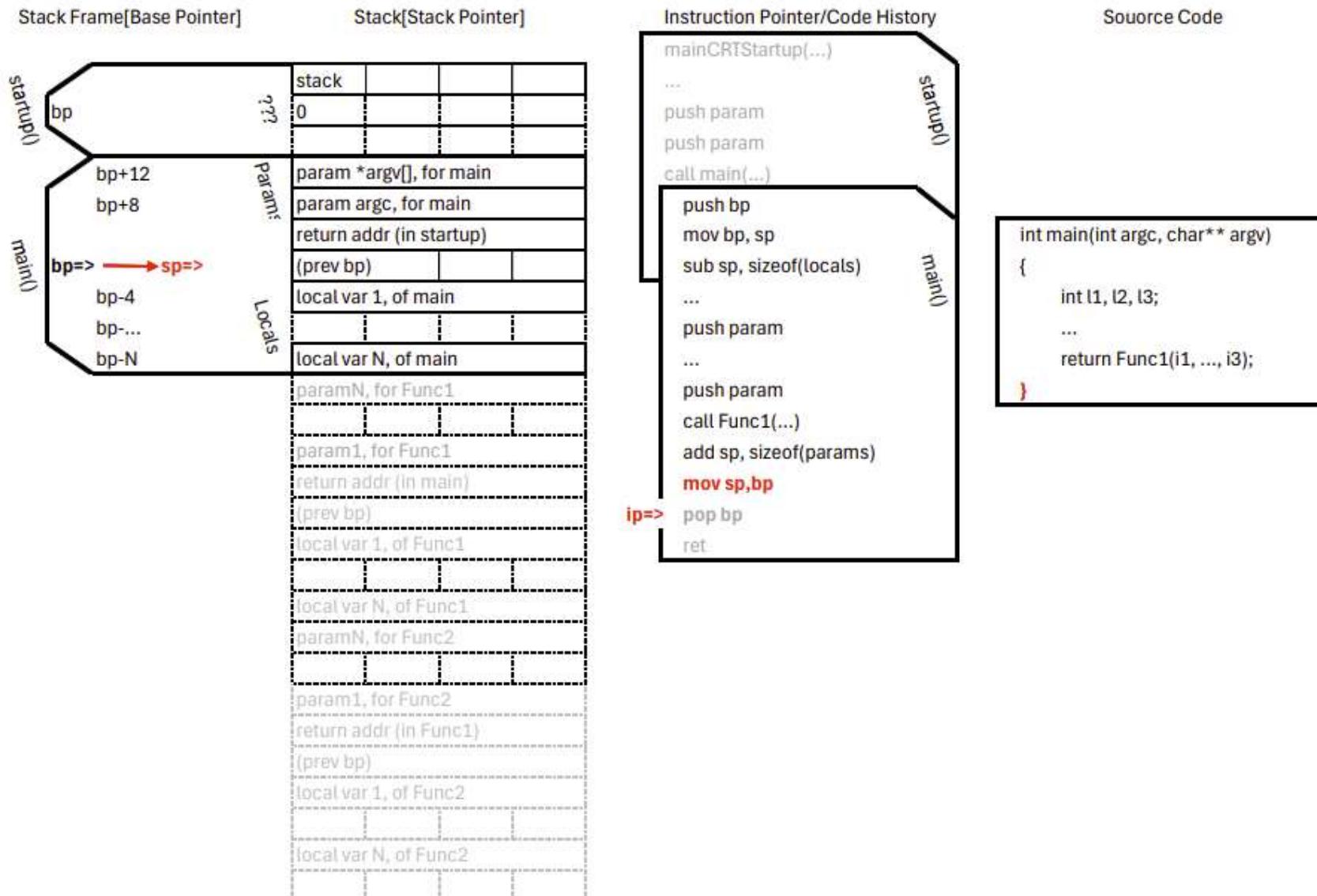
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



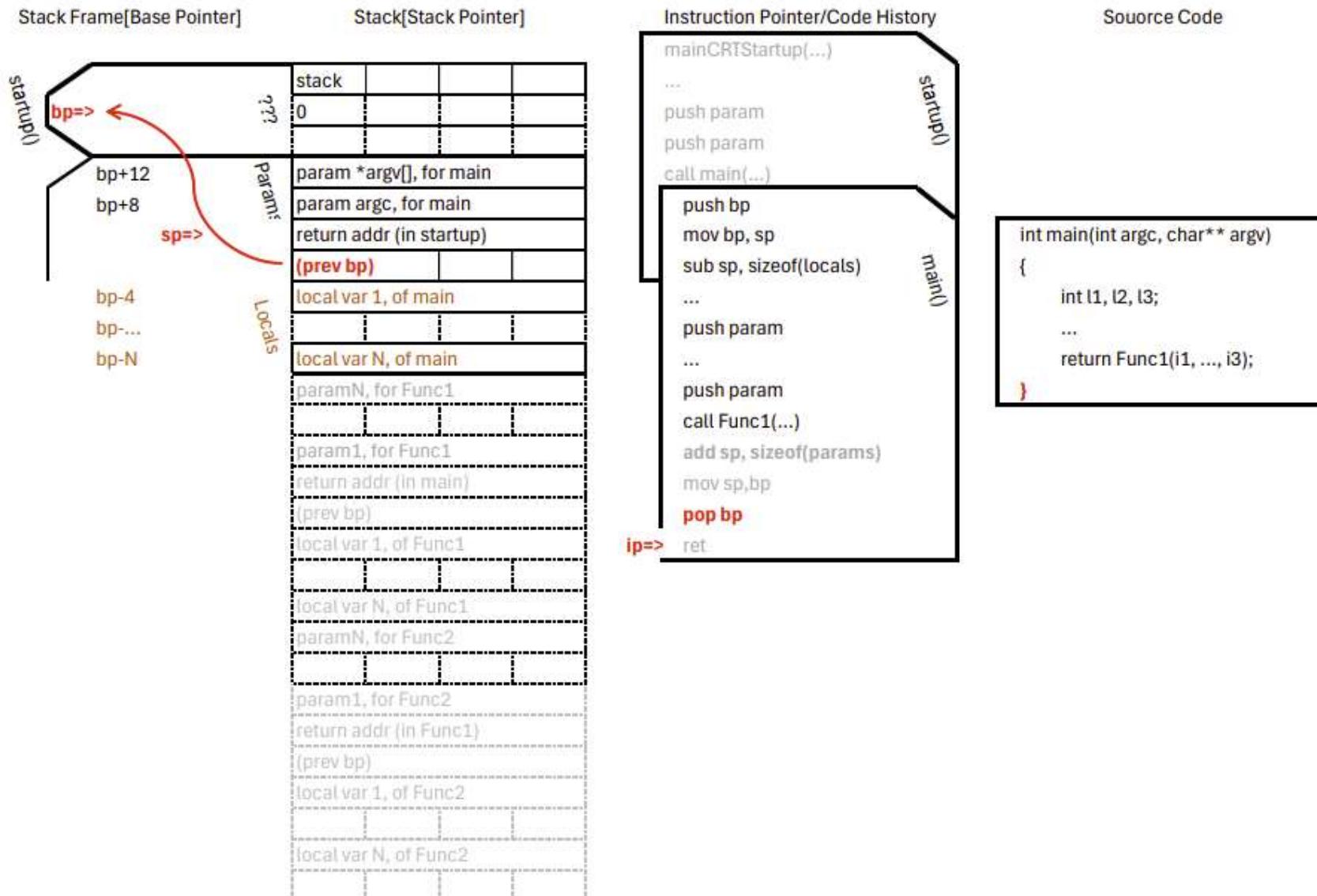
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



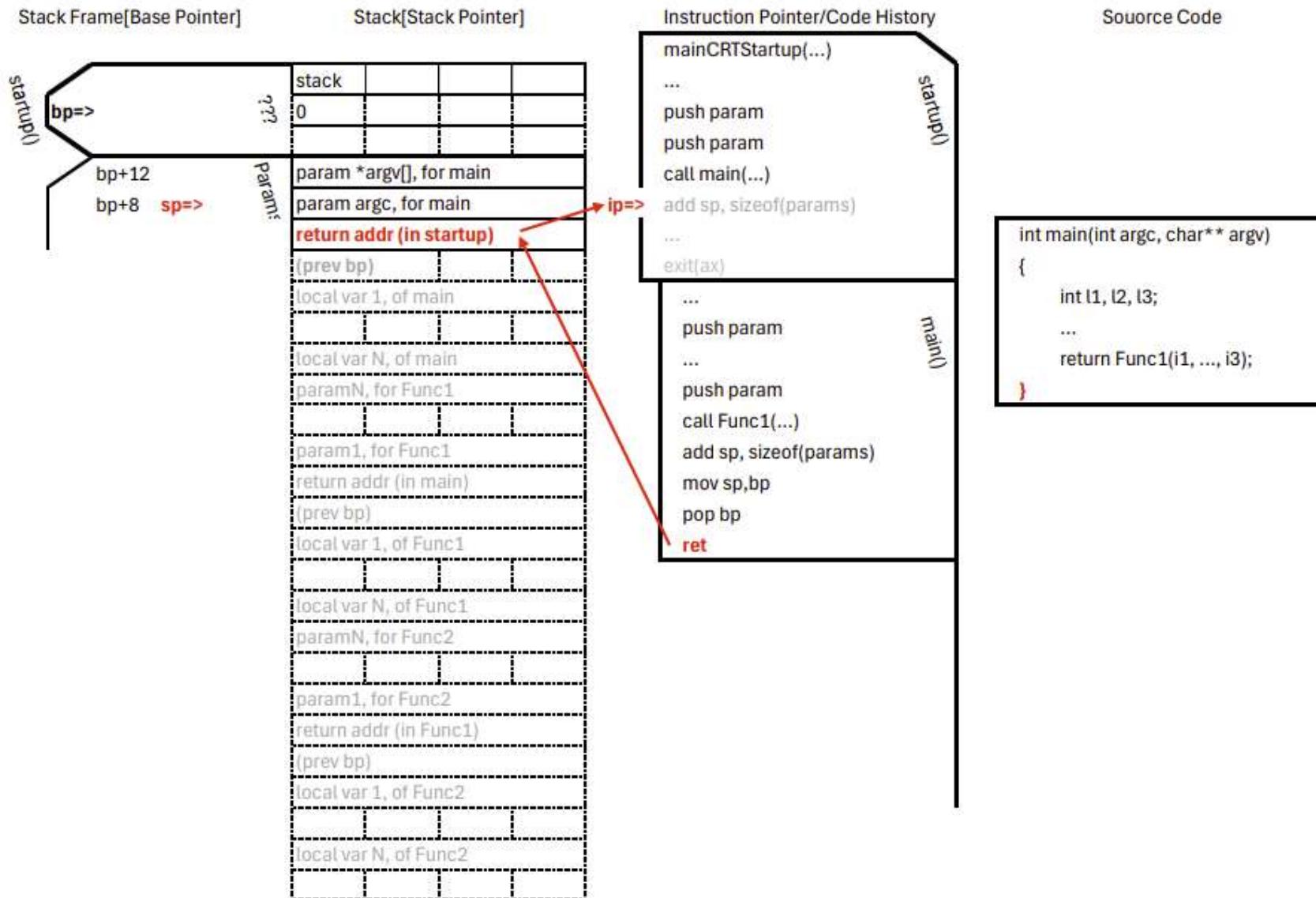
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



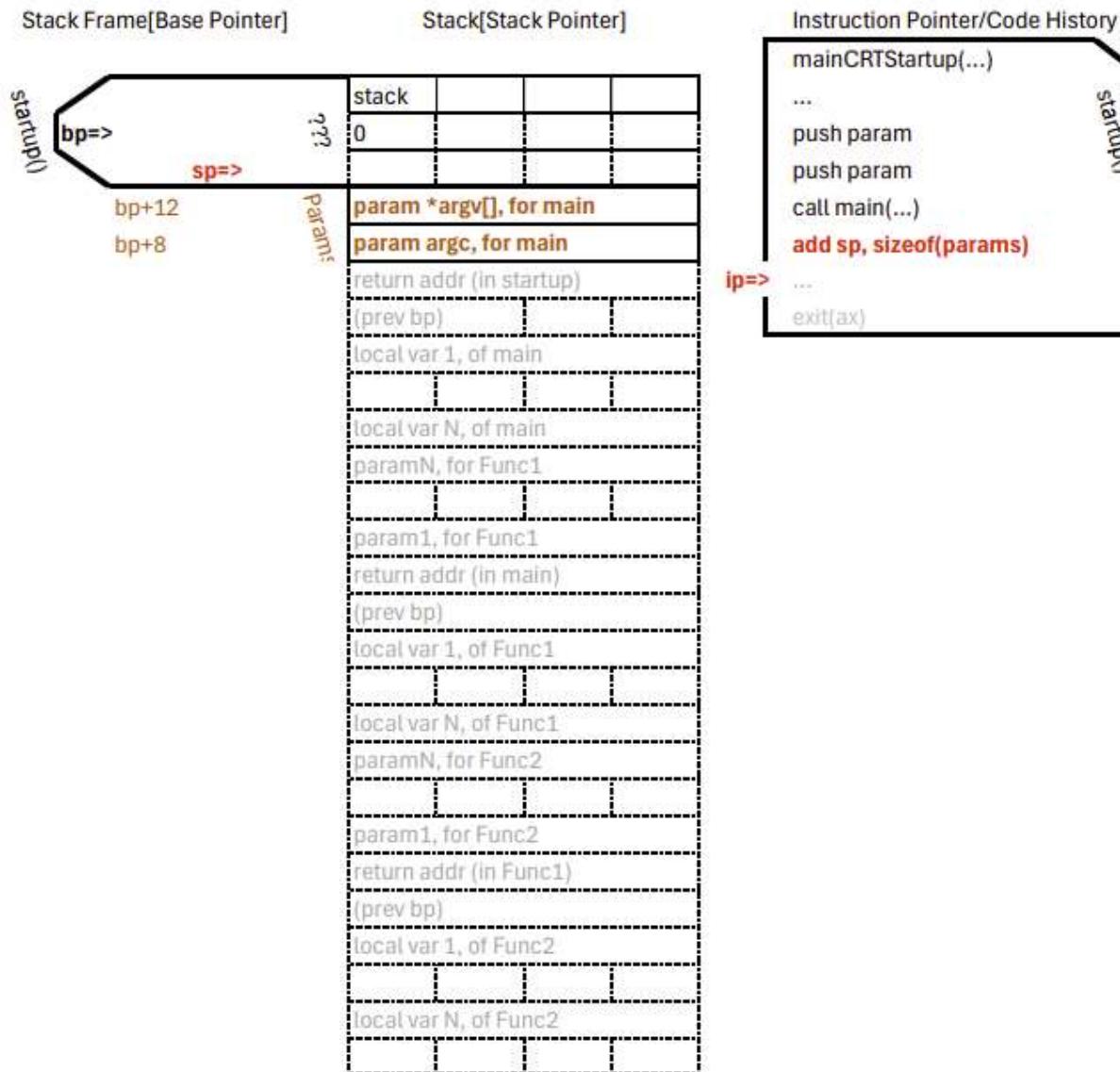
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



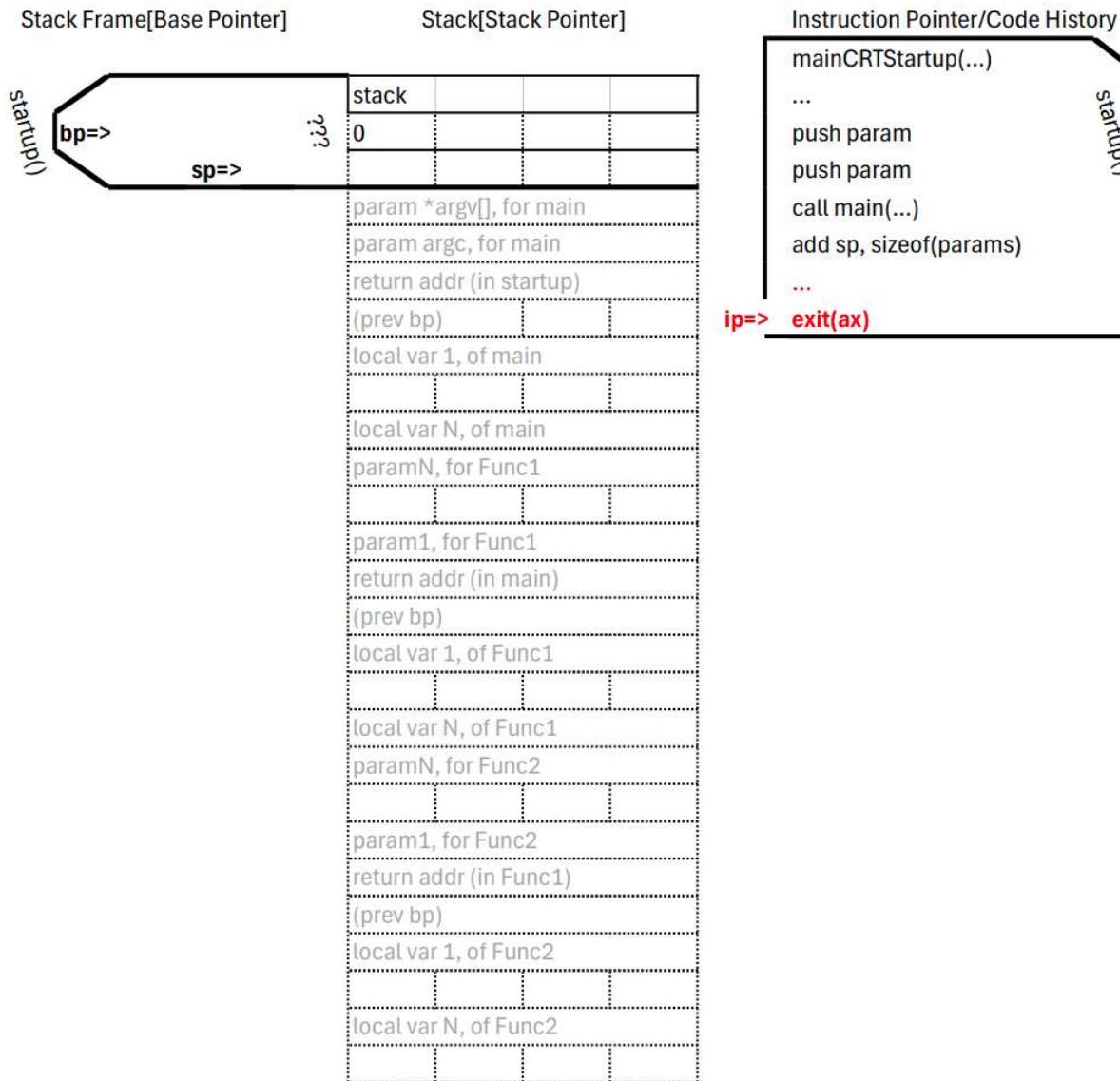
# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



# C++ Under the Hood – Stack Frame / Base Pointer Mechanics



# C++ Under the Hood - What did we learn today ?

- Inheritance vs. Aggregation:
  - Memory layout,
  - v-table pointer placement.
- The first shall be last,
- C'tor, D'tor, Virtual Mechanism Mechanics:
  - Set up the v-table pointer on the opening curly bracket of the C'tor,
  - Set up the v-table pointer on the opening curly bracket of the D'tor,
  - Inside-Out/Outside-In.
- Pointers vs Member Pointers:
  - `int *` vs `int Foo::*`
  - `int(*)(int)` vs `int(Foo::*)(int)`
  - Magic of a `nullptr` (not always ~~θ~~).
- Stack Frames/Base Pointer Mechanics
  - How to walk the stack

# C++ Under the Hood Questions?



## Thank You!

[Linkedin.com/in/chrisr98008](https://www.linkedin.com/in/chrisr98008)  
(No blind invites!  
Tell me how we know each other)

C++ Under the Hood

**Bonus slides below**

# C++ Under the Hood



# C++ Under the Hood – Constructor/Destructor Order

```
#include <iostream>
using std::cout;

struct Foo
{
    Foo() { cout << "Foo::Foo()    this=" << this << "\n"; }
    ~Foo() { cout << "~Foo::Foo()   this=" << this << "\n"; }
};

int main()
{
    cout << "new Foo[5];\n";
    Foo* pFoo = new Foo[5];
    cout << "Foo foo[5];\n";
    Foo foo[5];
    cout << "delete[] pFoo;\n";
    delete[] pFoo;
    cout << "//end of scope main()\n";
}
```

[godbolt.org/z/vdPs8sfYc](https://godbolt.org/z/vdPs8sfYc)

# C++ Under the Hood – Constructor/Destructor Order

```
#include <iostream>
using std::cout;

struct Foo
{
    Foo() { cout << "Foo::Foo()"    this=" << this << "\n"; }
    ~Foo() { cout << "~Foo::Foo()"   this=" << this << "\n"; }
};

int main()
{
    cout << "new Foo[5];\n";
    Foo* pFoo = new Foo[5];
    cout << "Foo foo[5];\n";
    Foo foo[5];
    cout << "delete[] pFoo;\n";
    delete[] pFoo;
    cout << "//end of scope main()\n";
}
```

```
new Foo[5];
Foo::Foo()      this=0x12ca2c8
Foo::Foo()      this=0x12ca2c9
Foo::Foo()      this=0x12ca2ca
Foo::Foo()      this=0x12ca2cb
Foo::Foo()      this=0x12ca2cc
Foo foo[5];
Foo::Foo()      this=0x7ffe8c8246c3
Foo::Foo()      this=0x7ffe8c8246c4
Foo::Foo()      this=0x7ffe8c8246c5
Foo::Foo()      this=0x7ffe8c8246c6
Foo::Foo()      this=0x7ffe8c8246c7
delete[] pFoo;
~Foo::Foo()      this=0x12ca2cc
~Foo::Foo()      this=0x12ca2cb
~Foo::Foo()      this=0x12ca2ca
~Foo::Foo()      this=0x12ca2c9
~Foo::Foo()      this=0x12ca2c8
//end of scope main()
~Foo::Foo()      this=0x7ffe8c8246c7
~Foo::Foo()      this=0x7ffe8c8246c6
~Foo::Foo()      this=0x7ffe8c8246c5
~Foo::Foo()      this=0x7ffe8c8246c4
~Foo::Foo()      this=0x7ffe8c8246c3
```

# C++ Under the Hood – Constructor/Destructor Order

```
#include <iostream>
using std::cout;

struct Foo
{
    Foo() { cout << "Foo::Foo()"    this=" << this << "\n"; }
    ~Foo() { cout << "~Foo::Foo()"   this=" << this << "\n"; }
};

int main()
{
    cout << "Foo foo3[3];\n";
    Foo foo3[3];
    cout << "Foo foo6[6];\n";
    Foo foo6[6];
    cout << "//end of scope main()\n";
}
```

```
Foo foo3[3];
Foo::Foo()  this=0x7fff617bb12d
Foo::Foo()  this=0x7fff617bb12e
Foo::Foo()  this=0x7fff617bb12f
Foo foo6[6];
Foo::Foo()  this=0x7fff617bb127
Foo::Foo()  this=0x7fff617bb128
Foo::Foo()  this=0x7fff617bb129
Foo::Foo()  this=0x7fff617bb12a
Foo::Foo()  this=0x7fff617bb12b
Foo::Foo()  this=0x7fff617bb12c
//end of scope main()
~Foo::Foo()  this=0x7fff617bb12c
~Foo::Foo()  this=0x7fff617bb12b
~Foo::Foo()  this=0x7fff617bb12a
~Foo::Foo()  this=0x7fff617bb129
~Foo::Foo()  this=0x7fff617bb128
~Foo::Foo()  this=0x7fff617bb127
~Foo::Foo()  this=0x7fff617bb12f
~Foo::Foo()  this=0x7fff617bb12e
~Foo::Foo()  this=0x7fff617bb12d
```

# C++ Under the Hood – Constructor/Destructor Order

```
struct Foo {  
    Foo() { cout << "Foo::Foo()\tthis=" << this << "\n"; }  
    ~Foo() { cout << "~Foo::Foo()\tthis=" << this << "\n"; }  
    void In() { cout << "Foo::In()\tthis=" << this << "\n"; }  
};  
Foo& Bar() {  
    static Foo foo; ←  
    return foo;  
}  
static Foo foo; ←  
  
struct Qaz {  
    static inline Foo foo; ←  
    Qaz() { cout << "Qaz::Qaz()\tthis=" << this << "\n"; foo.In(); }  
    ~Qaz() { cout << "~Qaz::Qaz()\tthis=" << this << "\n"; foo.In(); }  
} qaz; ←  
  
int main()  
{  
    cout << "\nint main()\n";  
    cout << "Bar().In();\n";     Bar().In();  
    cout << "foo.In();\n";      foo.In();  
    cout << "\n//end of scope main()\n\n";  
}
```

The diagram illustrates the execution flow and the order of constructor and destructor calls for the given C++ code. It uses colored arrows and boxes to map memory addresses to specific objects and their lifetimes.

**Execution Flow:**

- Initial State:** The program starts at the beginning of `main()`.
- Object Creation:**
  - `foo` (a `Foo` object) is created on the stack. Its address is `this=0x4041a8`. A red arrow points from this address to the `foo` variable in the `main()` frame.
  - `bar()` is called, creating a temporary `Bar` object. Its address is `this=0x4041a9`. A blue arrow points from this address to the `bar()` call in the `main()` frame.
  - `qaz` (a `Qaz` object) is created on the stack. Its address is `this=0x404198`. A green arrow points from this address to the `qaz` variable in the `main()` frame.
- Function Calls:**
  - `Bar().In()` is called. Its address is `this=0x4041a9`. A blue arrow points from this address to the `bar()` call in the `main()` frame.
  - `foo.In()` is called. Its address is `this=0x404199`. A purple arrow points from this address to the `foo` variable in the `main()` frame.
  - `Qaz::Qaz()` is called. Its address is `this=0x404199`. A purple arrow points from this address to the `qaz` variable in the `main()` frame.
  - `Foo::In()` is called. Its address is `this=0x4041a8`. A red arrow points from this address to the `foo` variable in the `main()` frame.
- Scope Exit:**
  - `~Qaz::Qaz()` is called. Its address is `this=0x404198`. A green arrow points from this address to the `qaz` variable in the `main()` frame.
  - `~Bar()` is called. Its address is `this=0x4041a9`. A blue arrow points from this address to the `bar()` call in the `main()` frame.
  - `~foo` (destruction of the local `foo`) is called. Its address is `this=0x4041a9`. A blue arrow points from this address to the `foo` variable in the `main()` frame.
  - `~Foo::Foo()` is called. Its address is `this=0x4041a8`. A red arrow points from this address to the `foo` variable in the `main()` frame.
- Final State:** The program ends at the end of the `main()` function.

[godbolt.org/z/3vahsaEr9](https://godbolt.org/z/3vahsaEr9)

# C++ Under the Hood - structure static footprint

```
struct A {
    auto& Get() { return map; }
private:
    std::map<int,int> map;
};

struct B {
    static auto& Get() { return map; }
private:
    inline static std::map<int,int> map;
};

struct C {
    static auto& Get() { return *map; }
private:
    inline static auto map = std::make_shared<std::map<int,int>>();
};

struct D {
    static auto& Get() { static std::map<int,int> map; return map; }
};

struct E {
    static auto& Get() { static auto map = std::make_shared<std::map<int,int>>(); return *map; }
};
```

[godbolt.org/z/563sbr99r](https://godbolt.org/z/563sbr99r)

# C++ Under the Hood - Pre-/Post-main() TODO

```
int a = print("global int a");
auto A = Print("global auto A");

auto shared = std::make_shared<Print>(
    "global shared make_shared<Print>");

C c;
auto d = c.Get2();

int main() {
    cout<<"main()\n";
    auto shared = std::make_shared<Print>(
        "main local shared make_shared<Print>");
    auto e = c.Get3();
    cout<<"end main()\n";
}
```

ctor: inline static C::p1  
func:global int a  
ctor: global auto A  
ctor: global shared make\_shared<Print>  
func:C::Get2()  
ctor: C::Get2()  
dtor: C::Get2()  
ctor: inline static Get1() C::p2  
main()  
ctor: main local shared make\_shared<Print>  
func:C::Get3()  
ctor: C::Get3()  
dtor: C::Get3()  
ctor: inline static C::Get2() C::p3  
end main()  
dtor: inline static C::Get2() C::p3  
dtor: main local shared make\_shared<Print>  
dtor: inline static C::Get2() C::p3  
dtor: inline static Get1() C::p2  
dtor: inline static Get1() C::p2  
dtor: global shared make\_shared<Print>  
dtor: global auto A  
dtor: inline static C::p1

# C++ Under the Hood