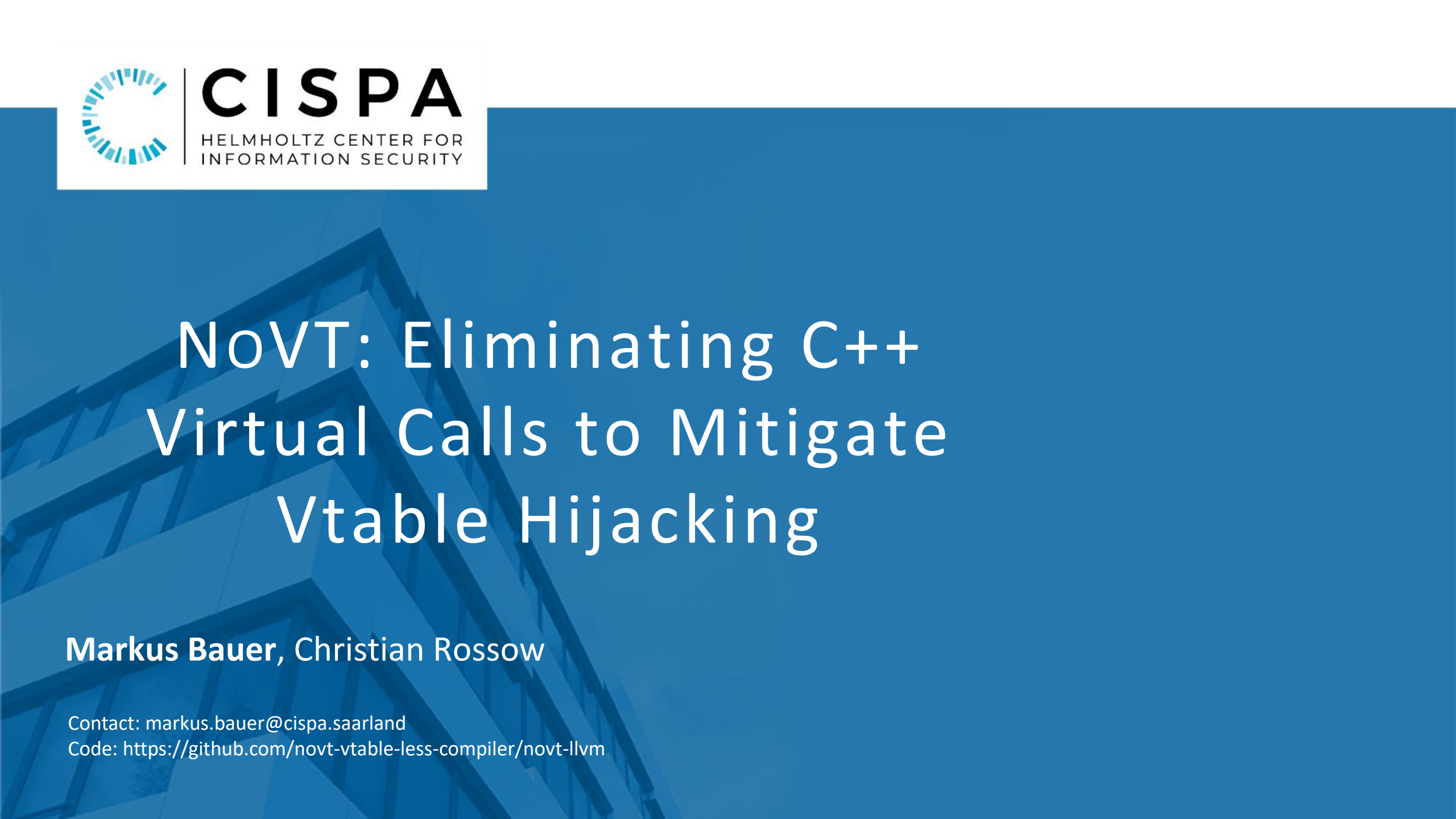


CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

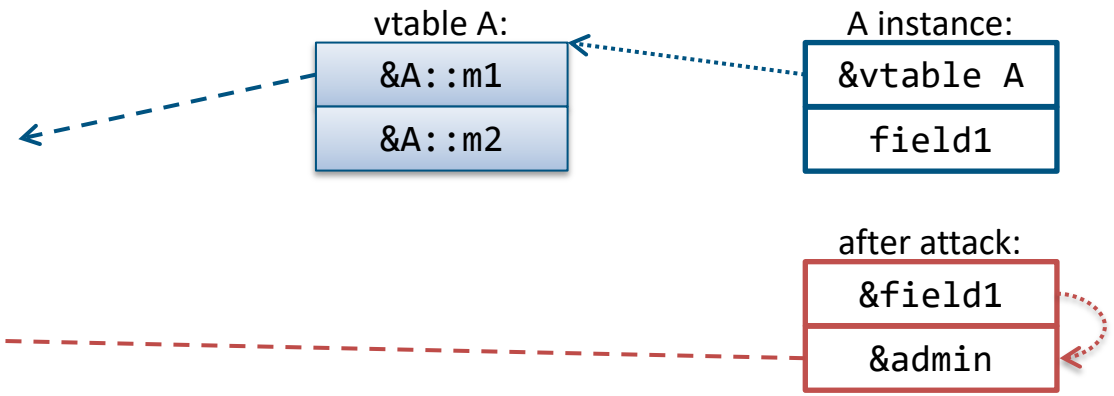
NoVT: Eliminating C++ Virtual Calls to Mitigate Vtable Hijacking

Markus Bauer, Christian Rossow

Contact: markus.bauer@cispa.saarland
Code: https://github.com/novt-vtable-less-compiler/novt-llvm

Attacks against Vtables – Vtable Hijacking

```
class A {
    long field1;
    virtual void m1() { … }
    virtual long m2() { … }
};

void admin() { … }
```



- Today, attackers target mainly vtable pointers during exploitation
- Large software might contain >50,000 of vtables with even more pointers
- We present NoVT, which removes this attack surface

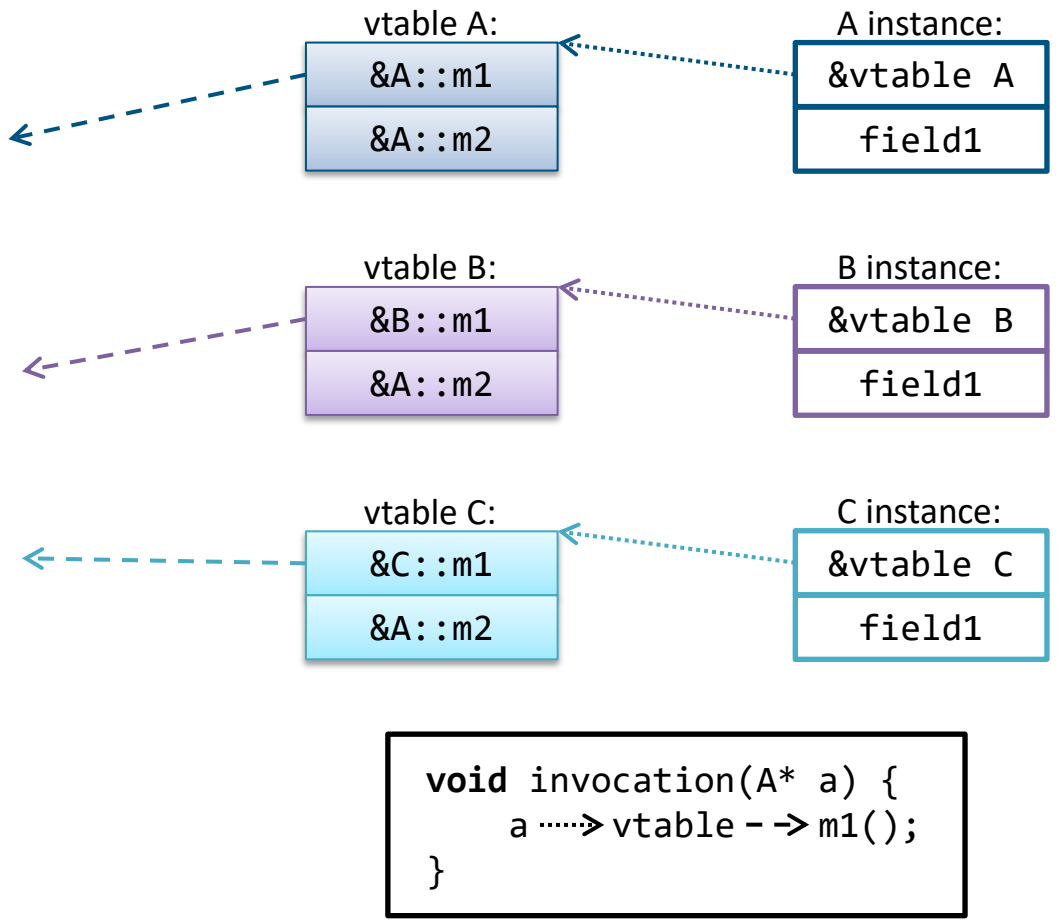
Virtual Function Tables (vtables)

```
class A {
    long field1;
    virtual void m1() { … }
    virtual long m2() { … }
};

class B : public A {
    void m1() override { … }
};

class C : public A {
    void m1() override { … }
};

void invocation(A* a) {
    a->m1();
}
```

3 possible functions to call

Virtual Function Tables (vtables)

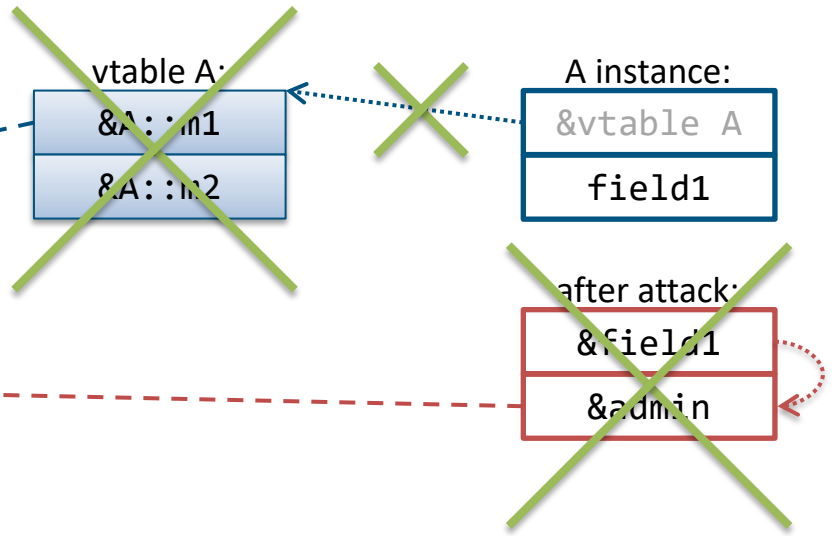
```
class A {
    long field1;
    virtual void m1() { … }
    virtual long m2() { … }
};

class B : public A {
    void m1() override { … }
};

class C : public A {
    void m1() override { … }
};

void invocation(A* a) {
    a->m1();
}
```




```
void invocation(A* a) {
    a ····> vtable -> m1();
}
```

NoVT: Vtable-Less C++ Compiler

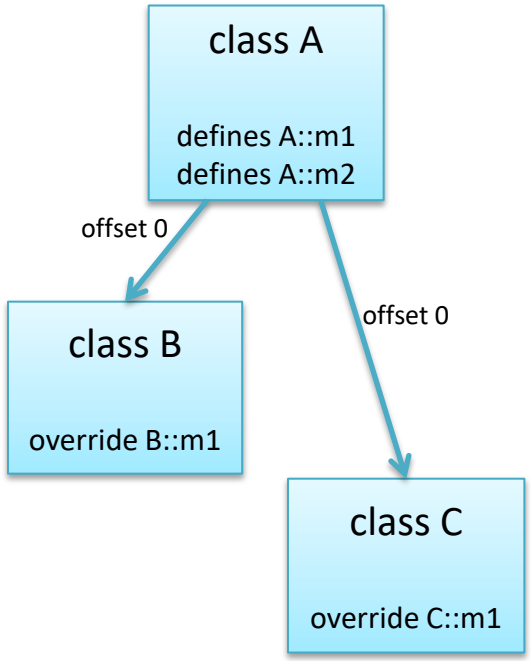
```
class A {
    long field1;
    virtual void m1() { … }
    virtual long m2() { … }
};

void admin() { … }
```



- NoVT eliminates vtables → vtable corruption impossible
- But how to dispatch methods?

NoVT: Overview

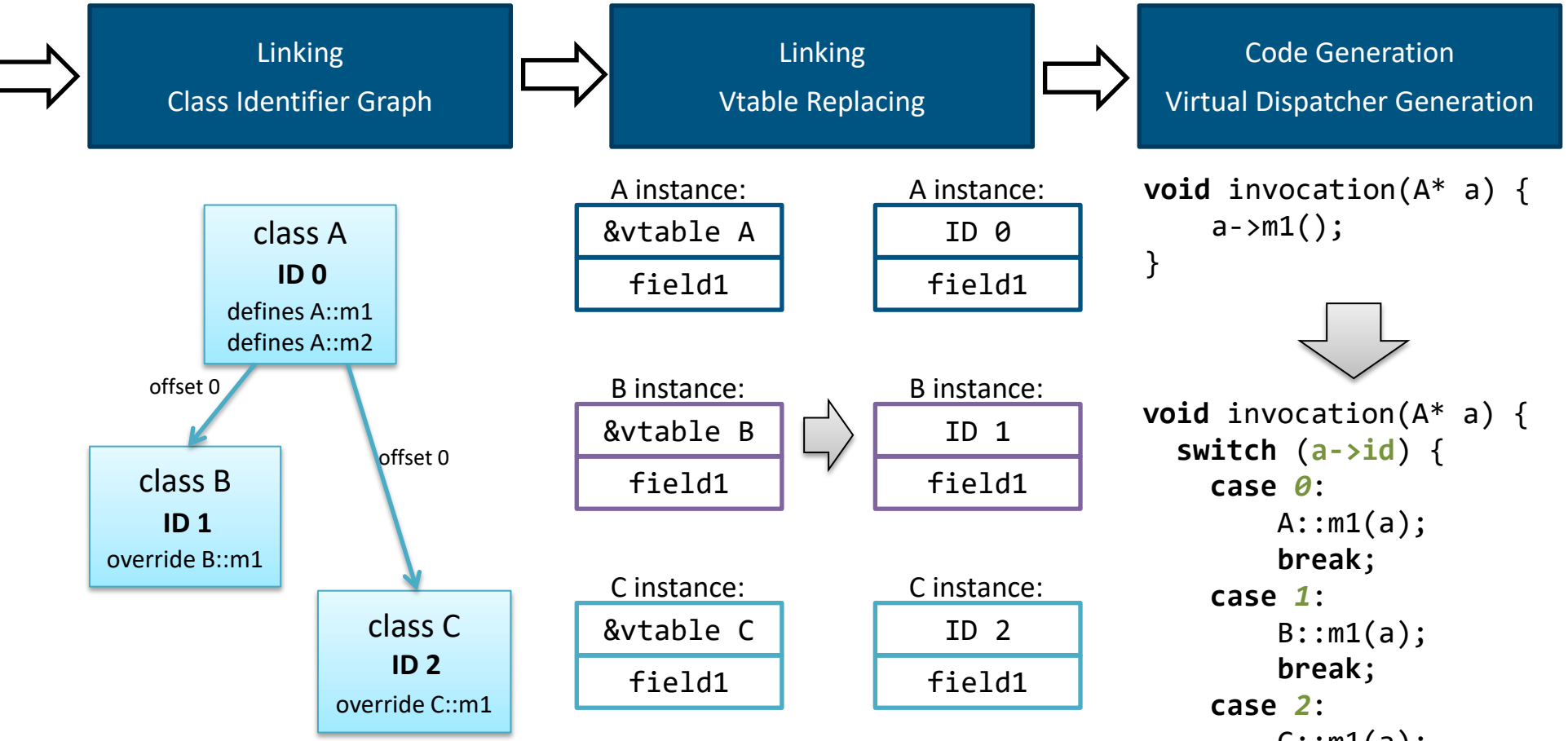

```
class A {
    long field1;
    virtual void m1();
    virtual long m2();
};

class B : public A {
    void m1() override;
};

class C : public A {
    void m1() override;
};
```




```
void invocation(A* a) {
    a->m1();
}
```

```
void invocation(A* a) {
  switch (class_of(a)) {
    case A:
        A::m1(a);
        break;
    case B:
        B::m1(a);
        break;
    case C:
        C::m1(a);
        break;
    default:
        abort();  // memory corruption detected!
  }
}
```

NoVT: Overview

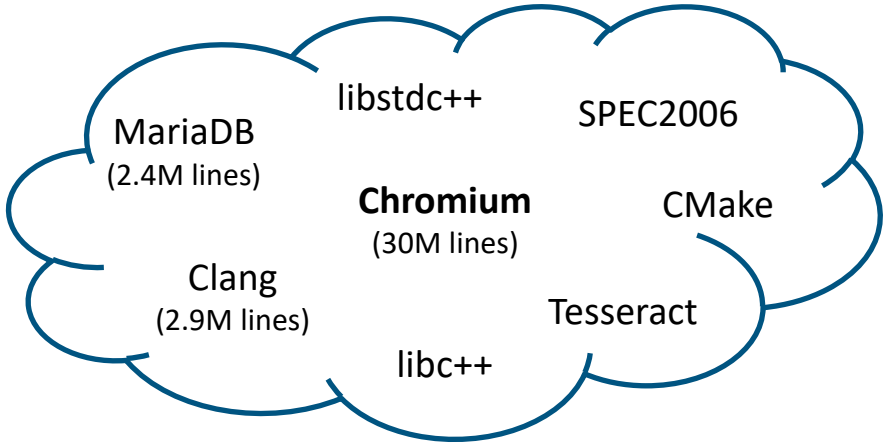



```
void invocation(A* a) {
    a->m1();
}
```

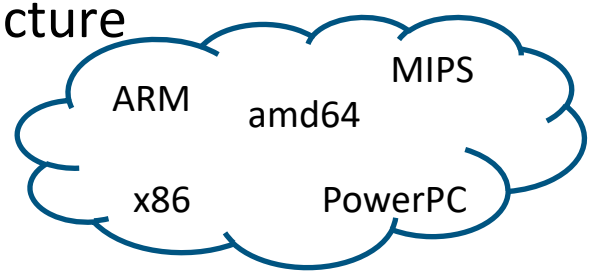
```
void invocation(A* a) {
  switch (a->id) {
    case 0:
        A::m1(a);
        break;
    case 1:
        B::m1(a);
        break;
    case 2:
        C::m1(a);
        break;
    default:
        abort();  // memory corruption detected!
  }
}
```

NoVT: Compatibility

- Prototype based on Clang / LLVM 10
- Tested on many large software projects



- No requirements on OS or CPU architecture



- Compilation is fast

  ≤ 3 seconds for all programs except Chromium

- Binaries usually get slightly smaller

  Direct calls allow better compiler optimizations to apply

- Limited dynamic linking / loading

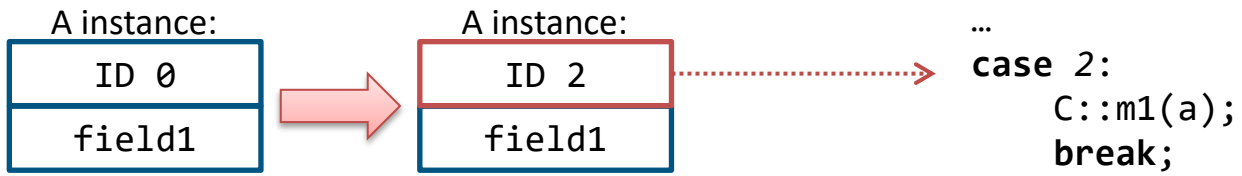
→ https://github.com/novt-vtable-less-compiler/novt-llvm

NoVT: Security

- NoVT protects all usages of vtables

  (Virtual dispatch, virtual memory offsets, dynamic casts, RTTI)

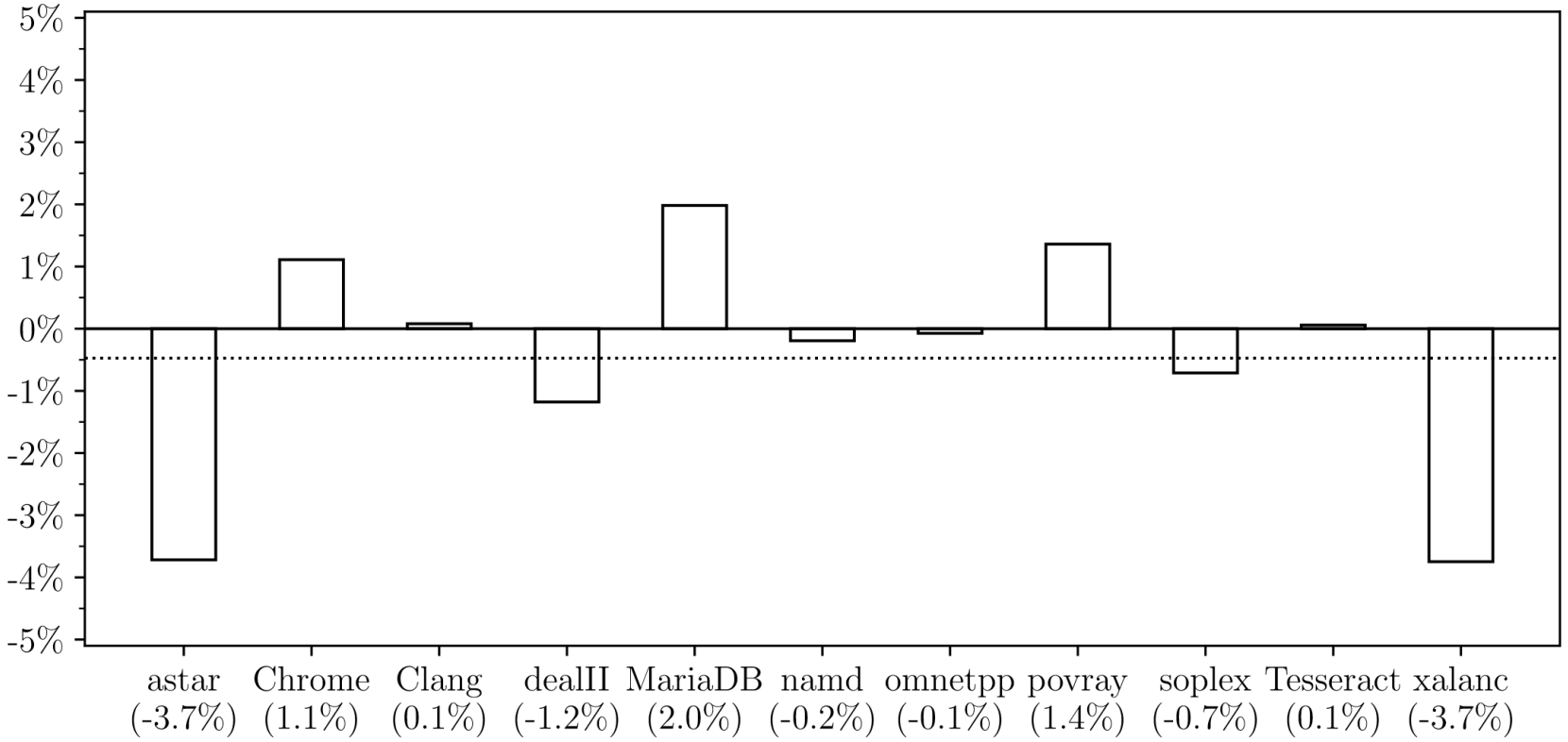
- Remaining risk: ID tampering



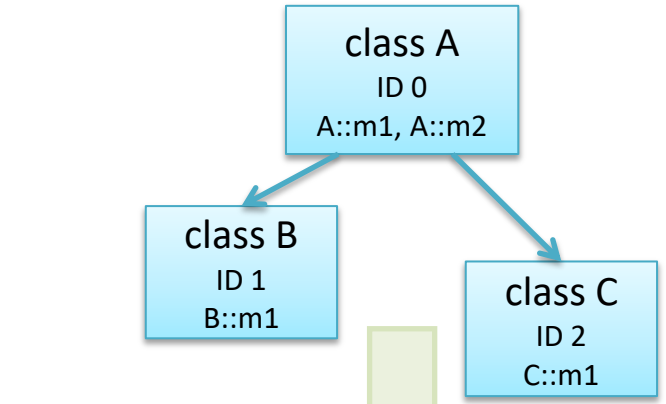
  - But: Minimal set of possible methods, no arbitrary code
  - NoVT's protection is optimal w.r.t. C++ type system

NoVT: Performance

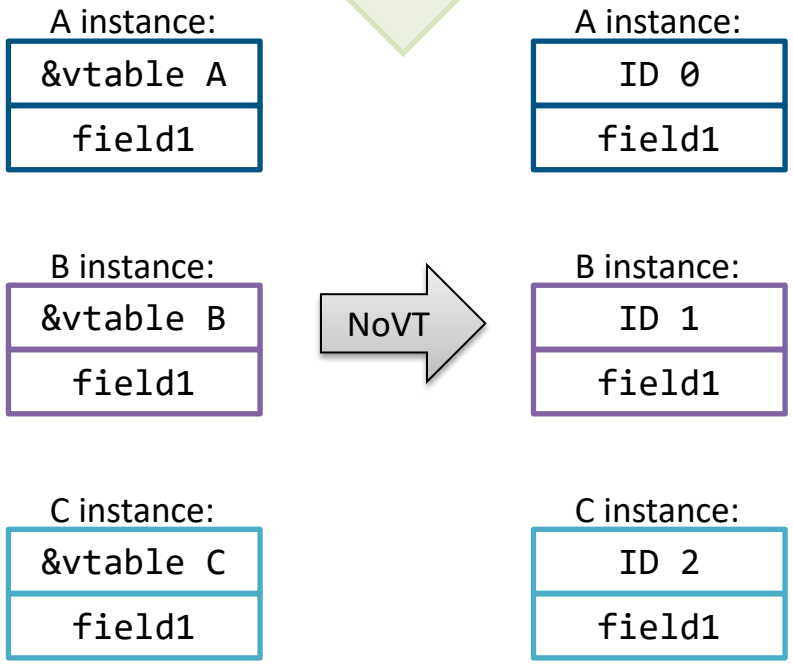
- On average, programs get 0.9% *faster*
  - No checks necessary
  - Branches can be faster than memory lookup
  - More suitable compiler optimizations



NoVT: Summary




```
class A {
    long field1;
    virtual void m1() { … }
    virtual long m2() { … }
};

void invocation(A* a) {
    a->m1();
}
```




```
void dispatch_A_m1(A* a) {
  switch (a->id) {
    case 0:
        A::m1(a);
        break;
    case 1:
        B::m1(a);
        break;
    case 2:
        C::m1(a);
        break;
    default:
        // memory corruption detected!
        abort();
    }
}
```

Contact: markus.bauer@cispa.saarland
Code: https://github.com/novt-vtable-less-compiler/novt-llvm