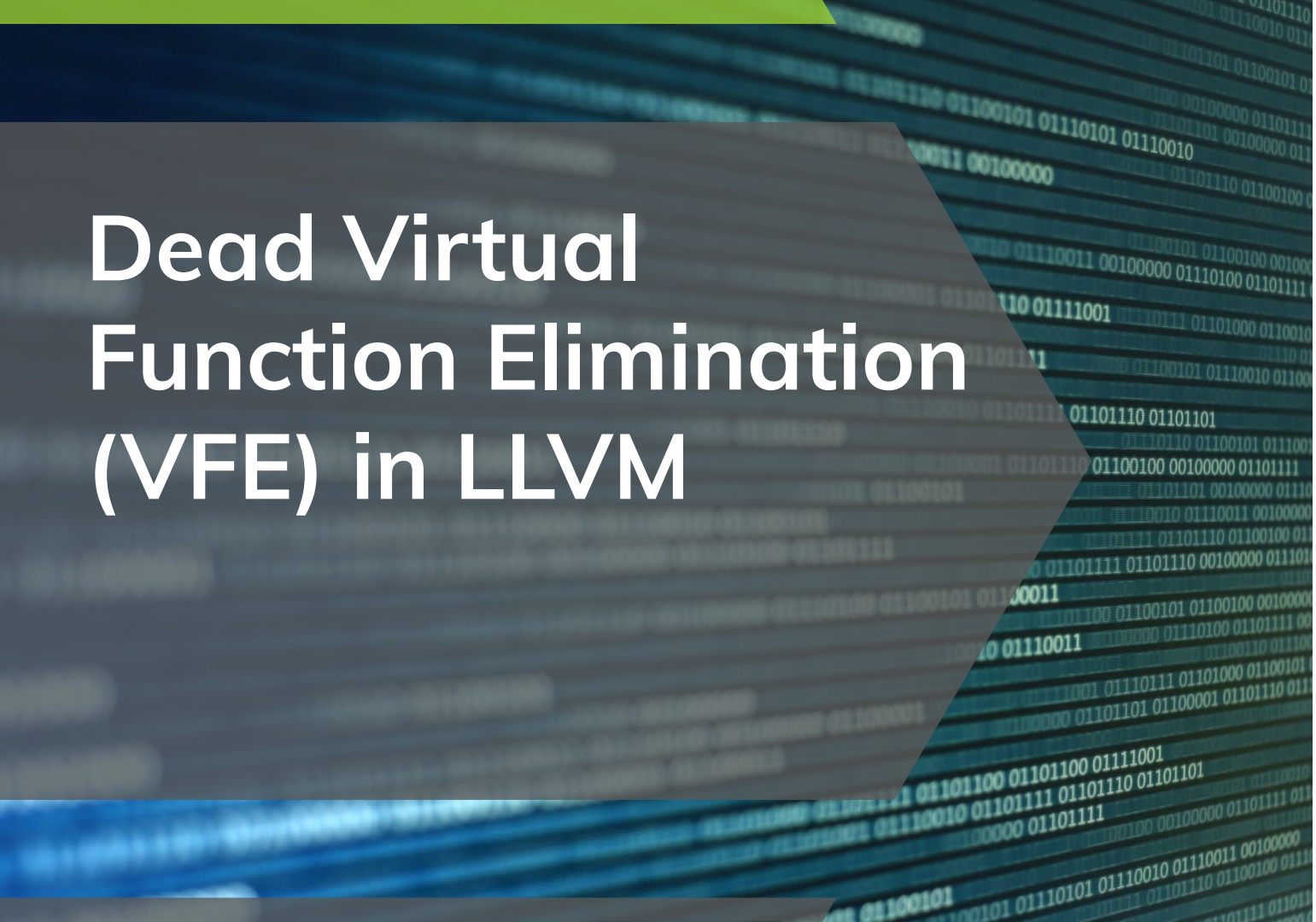
Dead Virtual Function Elimination (VFE) in LLVM



Linaro

An example we'd like to optimise

```
struct A {
  virtual int foo() { return 1; }
  virtual int bar() { return 2; }
};

struct B : A {
  virtual int foo() { return 3; }
  virtual int bar() { return 4; }
};

A* make_A() { return new A(); }
B* make_B() { return new B(); }

int call_1(A* p) { return p->foo(); }
int call_2(B* p) { return p->bar(); }
```

An example we'd like to optimise

```
struct A {
  virtual int foo() { return 1; }
  virtual int bar() { return 2; }
};

struct B : A {
  virtual int foo() { return 3; }
  virtual int bar() { return 4; }
};

A* make_A() { return new A(); }
B* make_B() { return new B(); }

int call_1(A* p) { return p->foo(); }
int call_2(B* p) { return p->bar(); }
```

Could call A::foo or B::foo

Can only call B::bar

An example we'd like to optimise

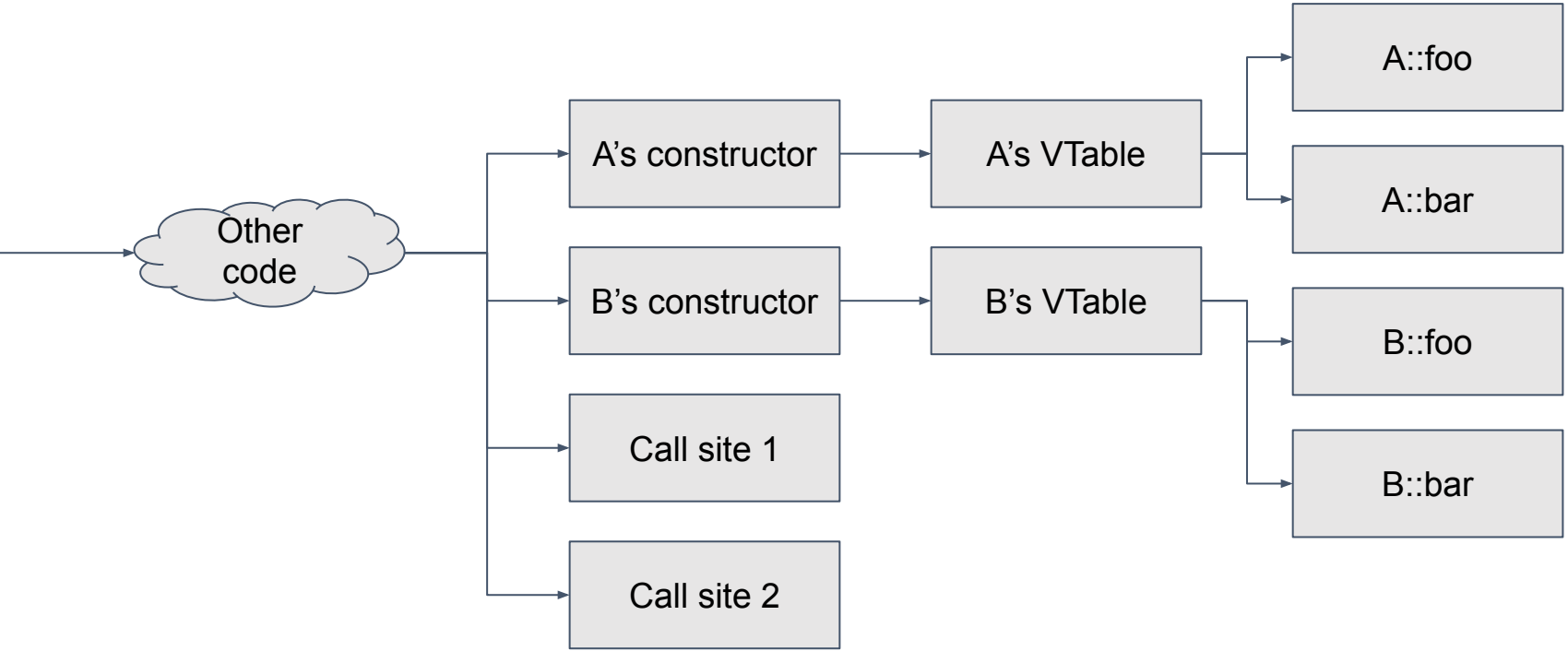
```
struct A {
  virtual int foo() { return 1; }
  virtual int bar() { return 2; }
};

struct B : A {
  virtual int foo() { return 3; }
  virtual int bar() { return 4; }
};

A* make_A() { return new A(); }
B* make_B() { return new B(); }

int call_1(A* p) { return p->foo(); }
int call_2(B* p) { return p->bar(); }
```

No calls to this function

Could call A::foo or B::foo

Can only call B::bar

What does this looks like to GlobalDCE?



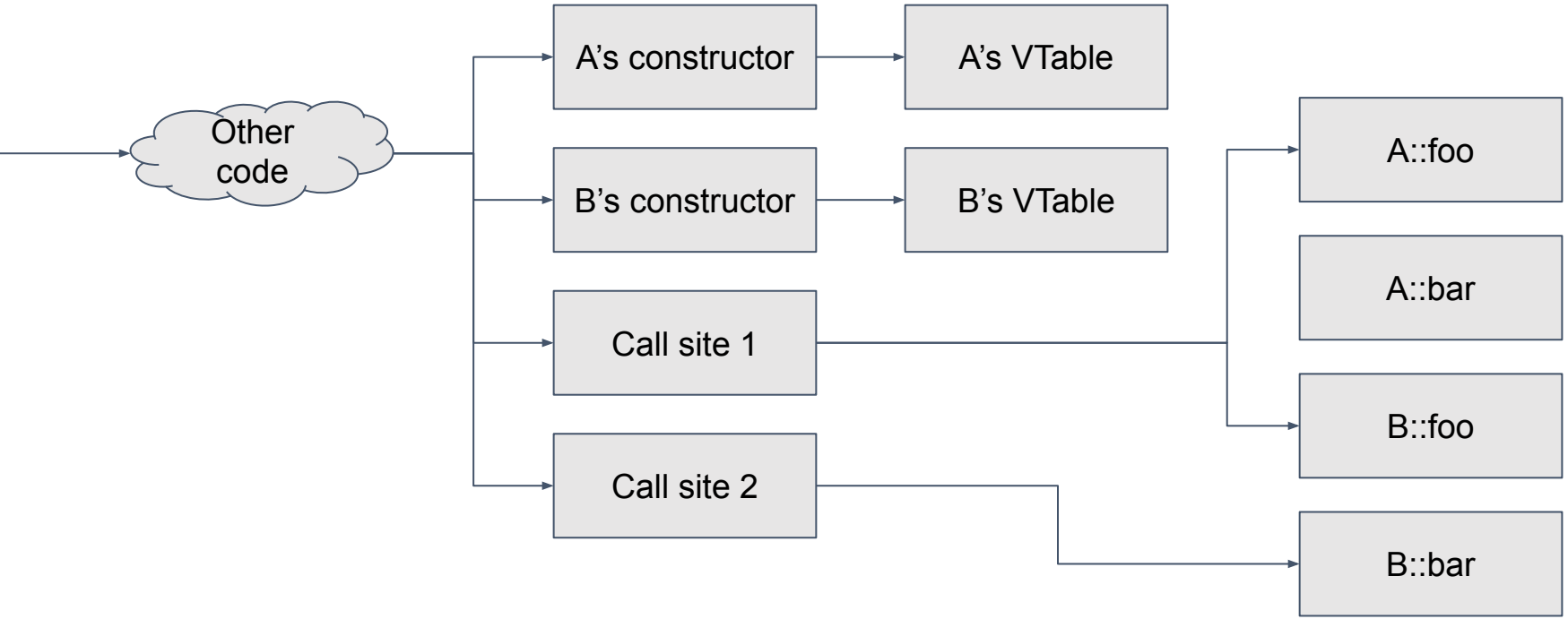
Type metadata

- Optionally added to vtables, virtual call sites
- Provides link between call sites and vtable slots
- Currently used for devirtualisation, control flow integrity
- Needs some changes to call sites to allow VFE

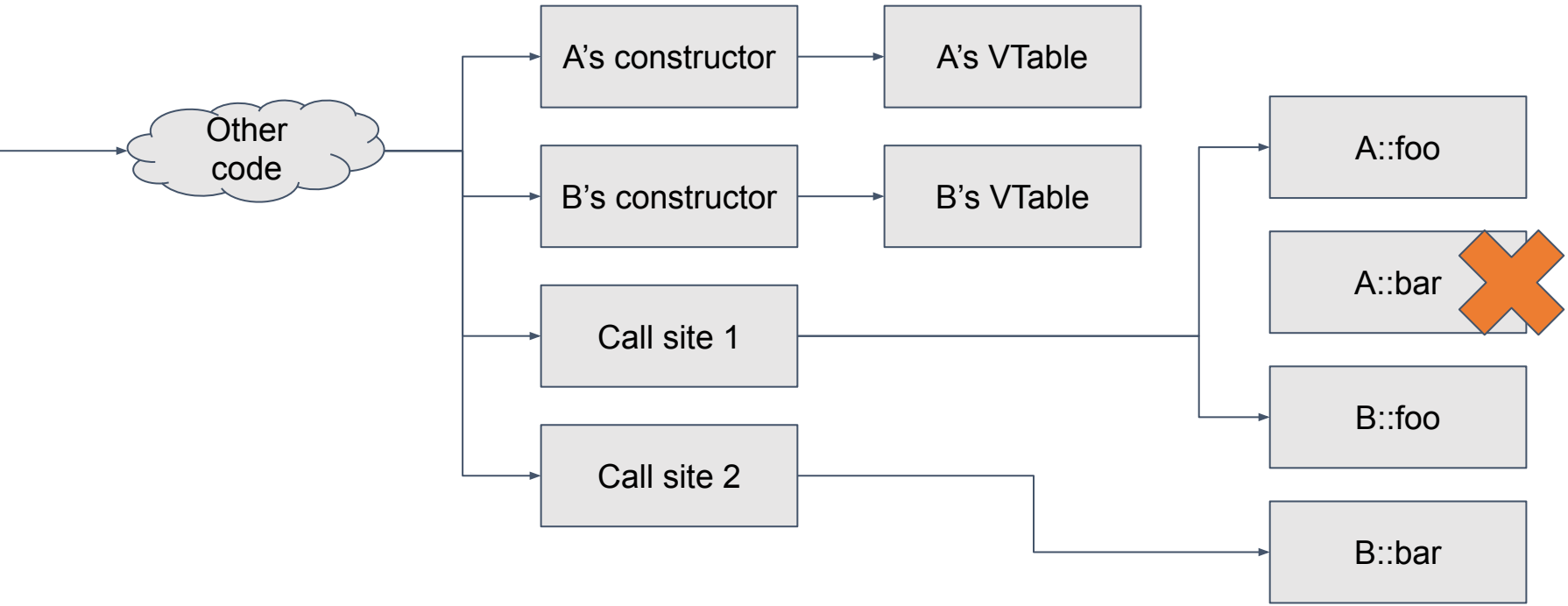
```
@_ZTV1A = constant { [4 x i8*] } ...,  !type !0
@_ZTV1B = constant { [4 x i8*] } ...,  !type !0, !type !4

!0 = !{i64 16, !"_ZTS1A"}
!4 = !{i64 16, !"_ZTS1B"}

%vtable1 = load i8*, i8** %0, align 8, !tbaa !9
%1 = tail call { i8*, i1 } @llvm.type.checked.load(
                                  i8* %vtable1, i32  8, metadata !"_ZTS1B")
```

Linaro

How does this change GlobalDCE's view?



How does this change GlobalDCE's view?

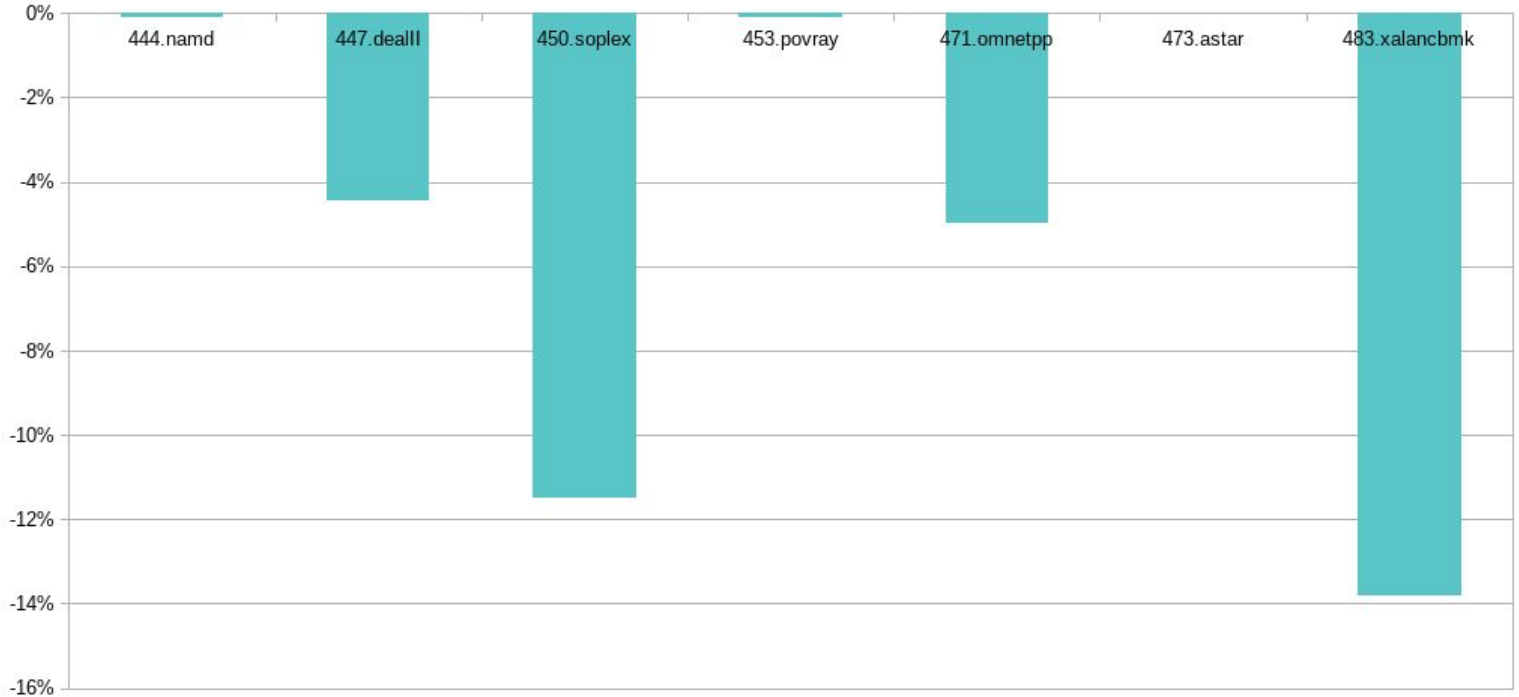


When is this optimisation valid?

- Need to see every possible call site involving a class
- Generally requires `-fvisibility=hidden`, and LTO
- All base classes (with some exceptions) must also be hidden
  - Otherwise, more virtual calls could be outside the LTO unit
- Added new `!vcall_visibility` metadata to represent this

Benchmark - SPEC2006 (C++ subset)

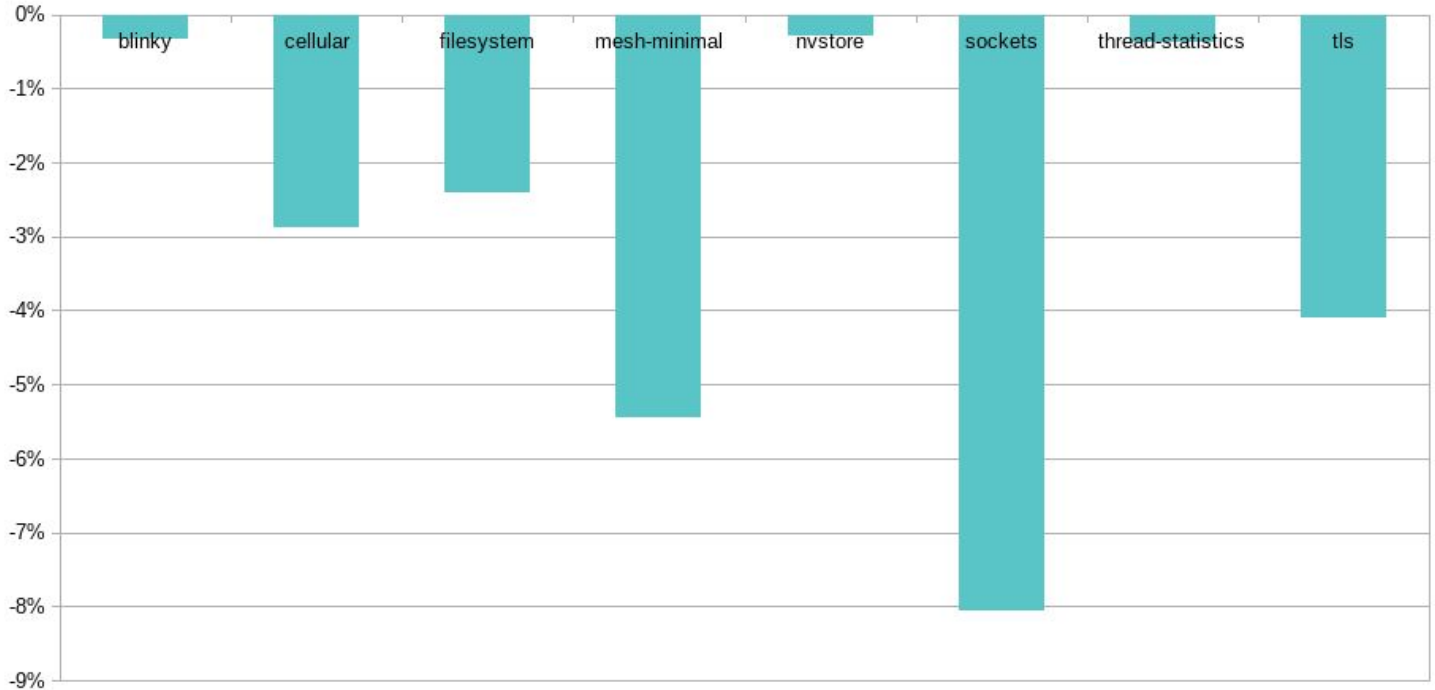
Code size change, compared to -Oz -flto -fvisibility=hidden



Linaro

Benchmark - mbed-os examples

Code size change, compared to -Oz -flto -fvisibility=hidden



Linaro

Future work

- ThinLTO
- Dead RTTI elimination