How Named Value Return Optimization Works

A **caller allocates space** for an instance
- (usually on the stack),
- then **passes a pointer** to the instance,
- and the **function fills in the bits**.

Technically,
- a **copy constructor should be called**,
- **but** that call (or calls) is **often omitted**
- (even if the constructor has side effects).

The optimization is thus technically incorrect, but it's widely used for performance.*

* Newer C++ standards define it to be correct.

What Should Be Called for an Addition?

As an example, assume that we have

```
complex operator+ (const complex& x,
                   const complex& y);
```

and a variable

```
complex a;
```

And then **we write**

```
complex b = a + a;
```

What should happen?

Addition in Theory Implies Three Constructors

```
complex b = a + a;
```

What should happen?

In theory, an **instance**
- is **constructed within `operator+`**, then
- **copy constructed as the return value**,
- then **copy constructed again as `b`**.

Addition in Practice Usually Calls One Constructor

```
complex b = a + a;
```

What should happen?

In practice,
- **variable `b` resides in the caller's stack frame**,
- so a **pointer to `b` is passed to `operator+`**,
- and **`operator+` constructs `b`**.

