Copy Elision

Presentation by Jon Kalb
Based on work by
Dave Abrahams

Copy Elision

The compiler is allowed to elide copies where results are "as if" copies were made. Return Value Optimization (commonly called RVO) is one such instance.

- Caller allocates space on stack for return value, passes the address to callee
- Callee constructs result *directly* in that space

Return Value Optimization

How many parameters are passed to the function f?

```
std::string f()
{
  std::string a("A");
 int b{23};
  …
  return a;
}
```

Return Value Optimization

How many parameters are passed to the function f?

```
std::string f()
{
  std::string a("A");
 int b{23};
  …
  return a;
}
```

C++ programmers answer *"None,"*

Return Value Optimization

How many parameters are passed to the function f?

```
std::string f()
{
  std::string a("A");
 int b{23};
  …
  return a;
}
```

C++ programmers answer *"None,"*
but assembly language programmers answer *"One."*
Why?

Return Value Optimization

How many parameters are passed to the function f?

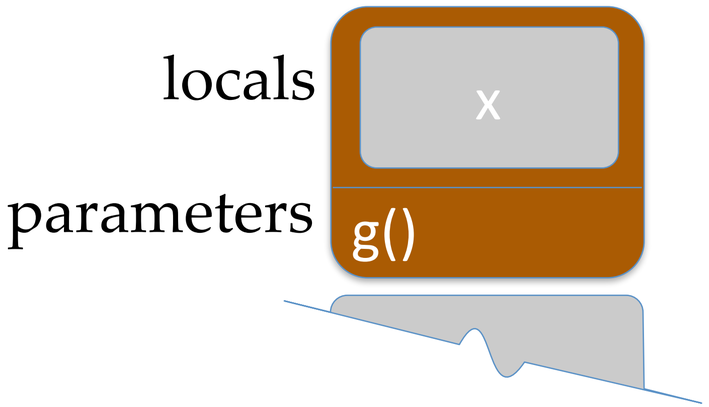
```
std::string f()
{
  std::string a("A");
 int b{23};
  …
  return a;
}
```

C++ programmers answer *"None,"*
but assembly language programmers answer *"One."*
Why?

The function is passed the address where the results should be written.

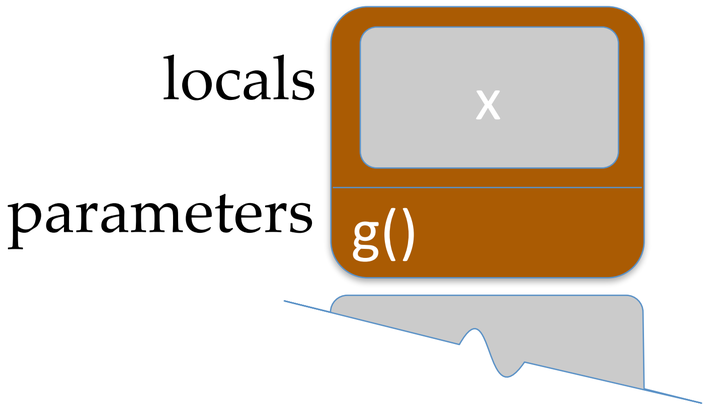
Return Value Unoptimized

```
std::string f()
{
    std::string a{"A"};
    int b{23};
    …
    return a;
}

void g()
{
    std::string x{f()};
}
```

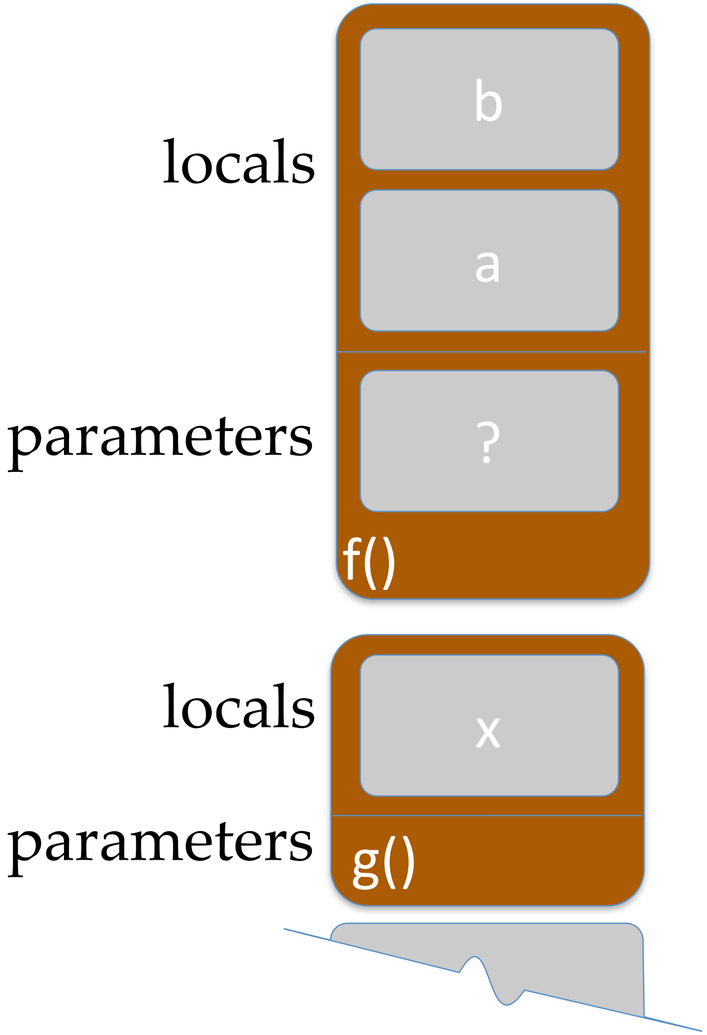


Return Value Unoptimized

```
std::string f()
{
    std::string a{"A"};
    int b{23};
    …
    return a;
}

void g()
{
    std::string x{f()};
}
```

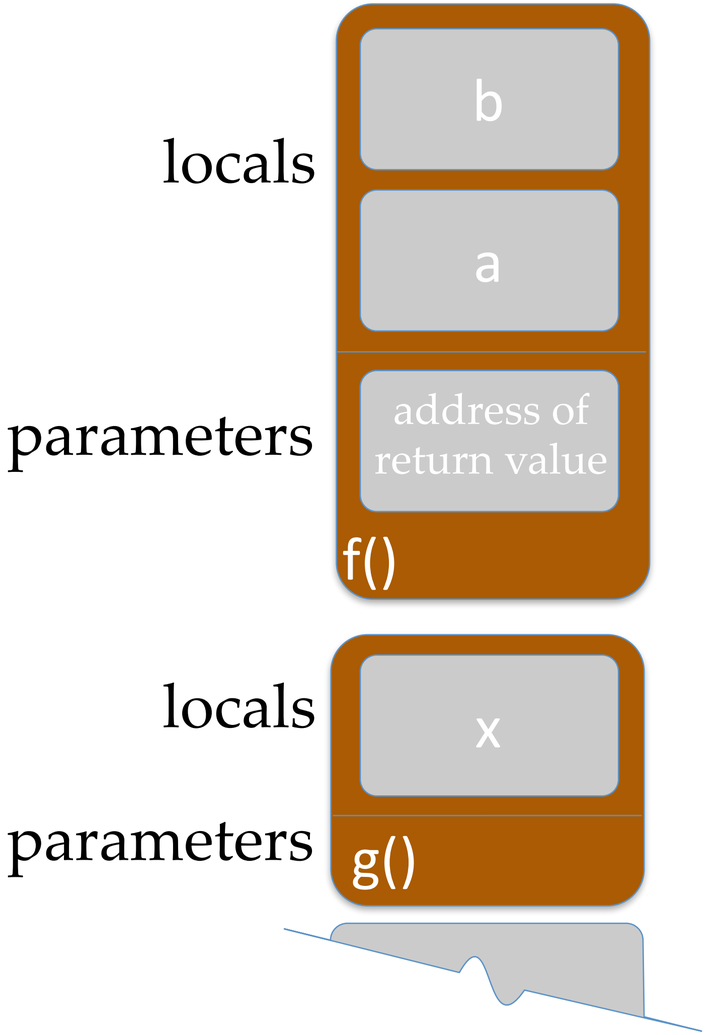


Return Value Unoptimized



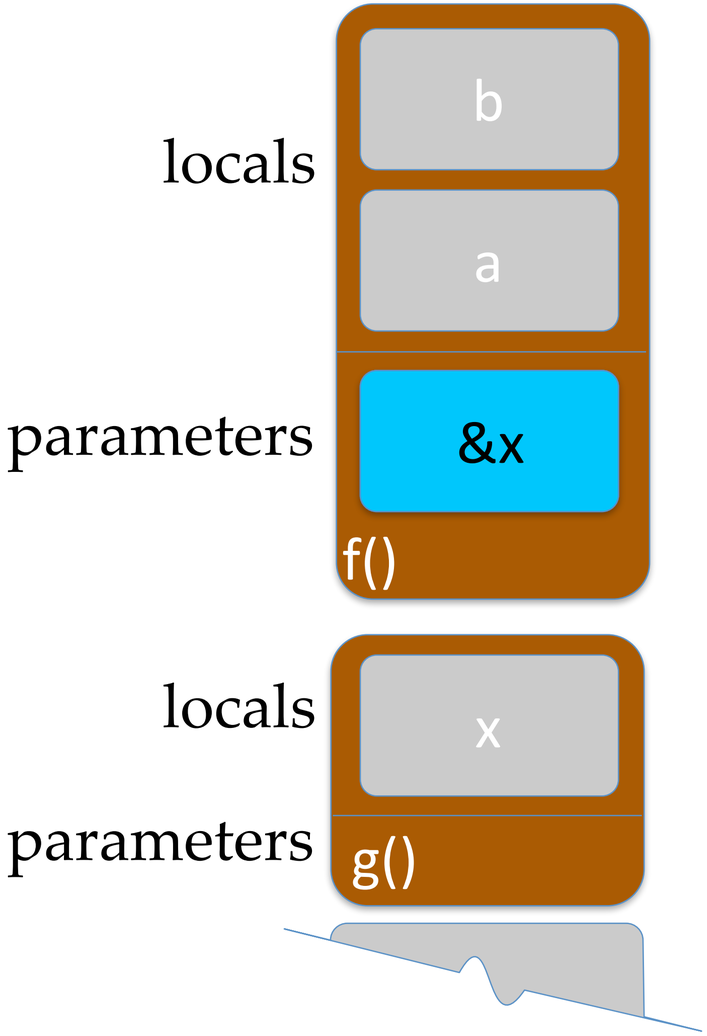

```
std::string f()
{
     std::string a{"A"};
     int b{23};
     …
     return a;
}

void g()
{
     std::string x{f()};
}
```

Return Value Unoptimized



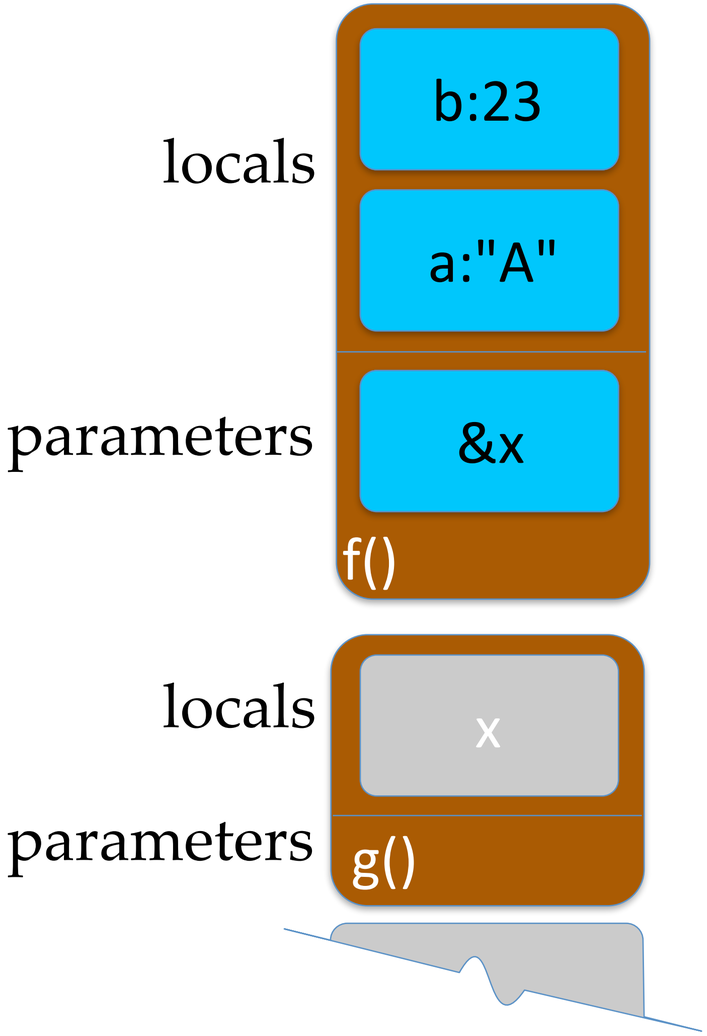

```
std::string f()
{
    std::string a{"A"};
    int b{23};
    …
    return a;
}

void g()
{
    std::string x{f()};
}
```

Return Value Unoptimized



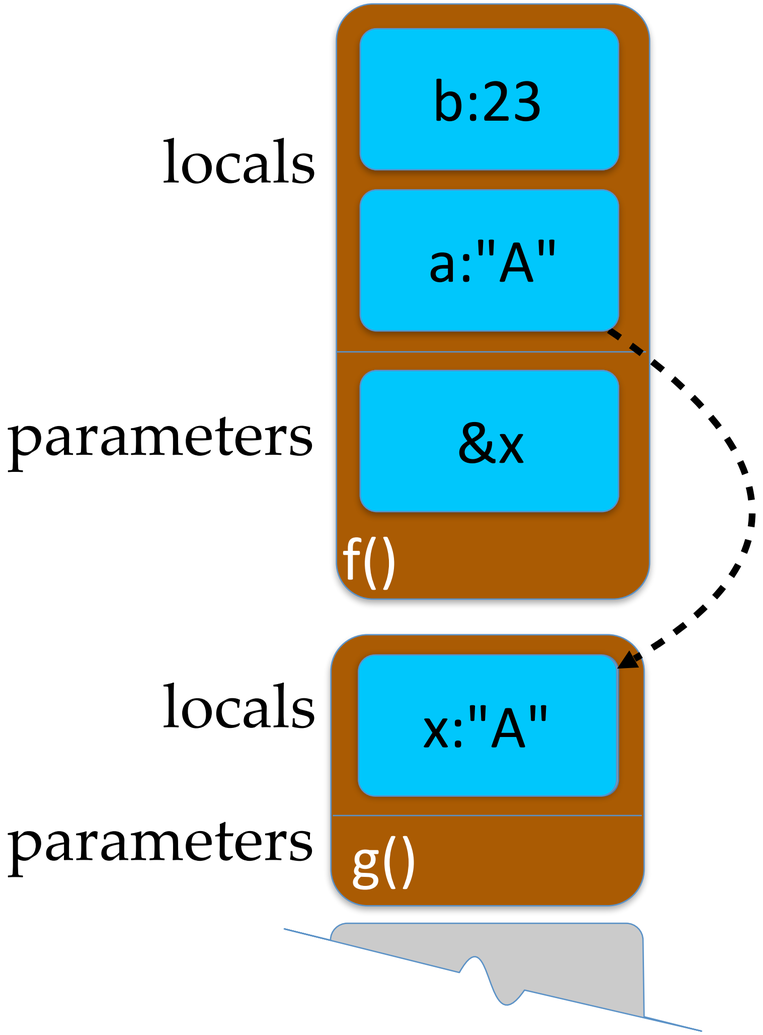

```
std::string f()
{
    std::string a{"A"};
    int b{23};
    …
    return a;
}

void g()
{
    std::string x{f()};
}
```

Return Value Unoptimized




```
std::string f()
{
     std::string a{"A"};
     int b{23};
     ...
     return a;
}

void g()
{
     std::string x{f()};
}
```

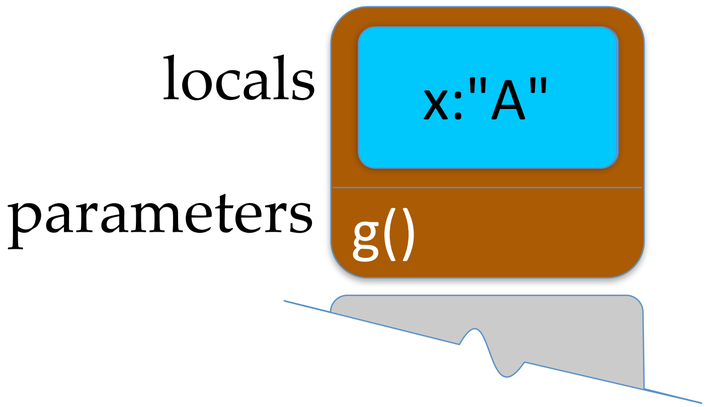
Return Value Unoptimized




```
std::string f()
{
     std::string a{"A"};
     int b{23};
     ...
     return a;
}

void g()
{
     std::string x{f()};
}
```

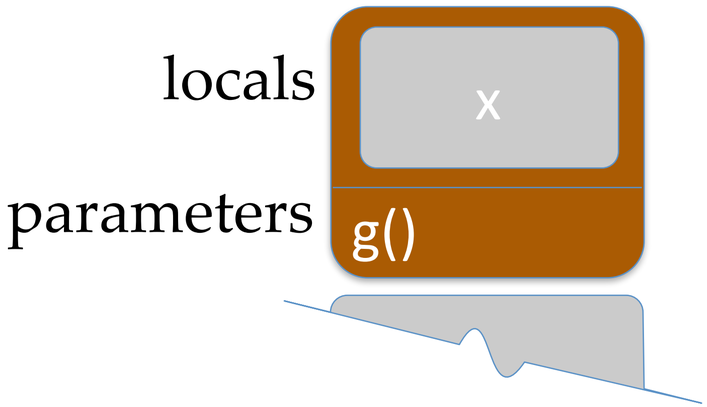
Return Value Unoptimized

```
std::string f()
{
    std::string a{"A"};
    int b{23};
    …
    return a;
}

void g()
{
    std::string x{f()};
}
```



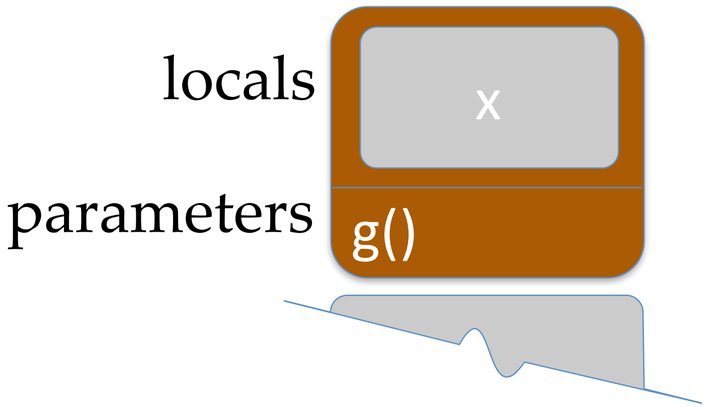
Return Value Optimization

```
std::string f()
{
    std::string a{"A"};
    int b{23};
    …
    return a;
}

void g()
{
    std::string x{f()};
}
```

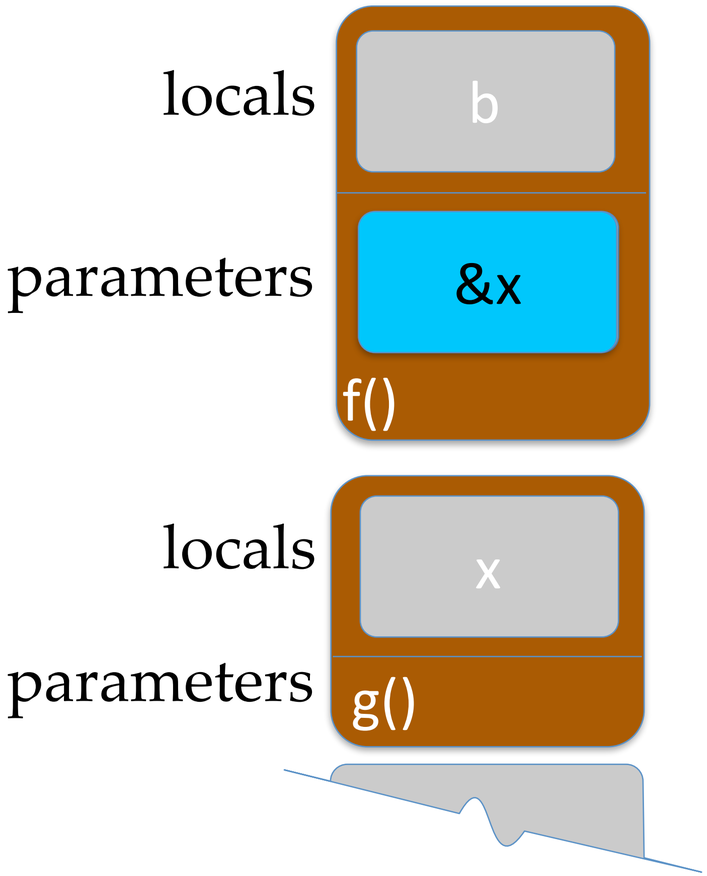


Return Value Optimization

```
std::string f()
{
    std::string a{"A"};
    int b{23};
    ...
    return a;
}

void g()
{
    std::string x{f()};
}
```

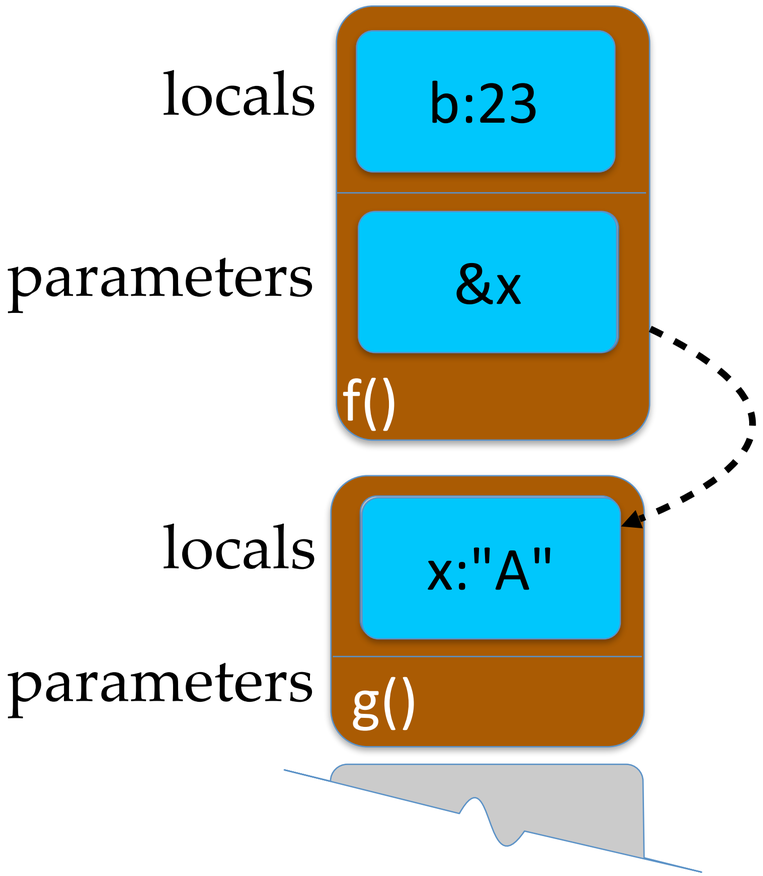


Return Value Optimization




```
std::string f()
{
     std::string a{"A"};
     int b{23};
     …
     return a;
}

void g()
{
     std::string x{f()};
}
```

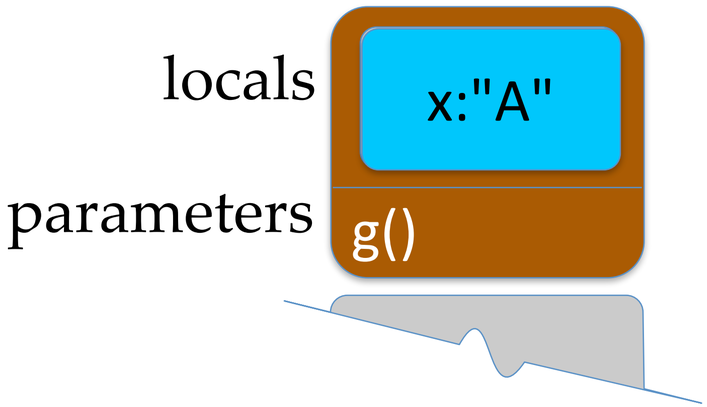
Return Value Optimization




```
std::string f()
{
     std::string a{"A"};
     int b{23};
     ...
     return a;
}

void g()
{
     std::string x{f()};
}
```

Return Value Optimization

```
std::string f()
{
    std::string a{"A"};
    int b{23};
    …
    return a;
}

void g()
{
    std::string x{f()};
}
```



Return Value Optimization

Return Value Optimization

Can RVO be used here?

```
Widget g()
{
  Widget a, b;
  …
 if (pred(some_value))
    return a;
 else
    return b;
}
```

Return Value Optimization

Can RVO be used here?

```
Widget g()
{
  Widget a, b;
  …
 if (pred(some_value))
    return a;
 else
    return b;
}
```

Or here:

```
Widget f()
{
  Widget a;
  …
  return pred(some_value)? a: Widget{};
}
```

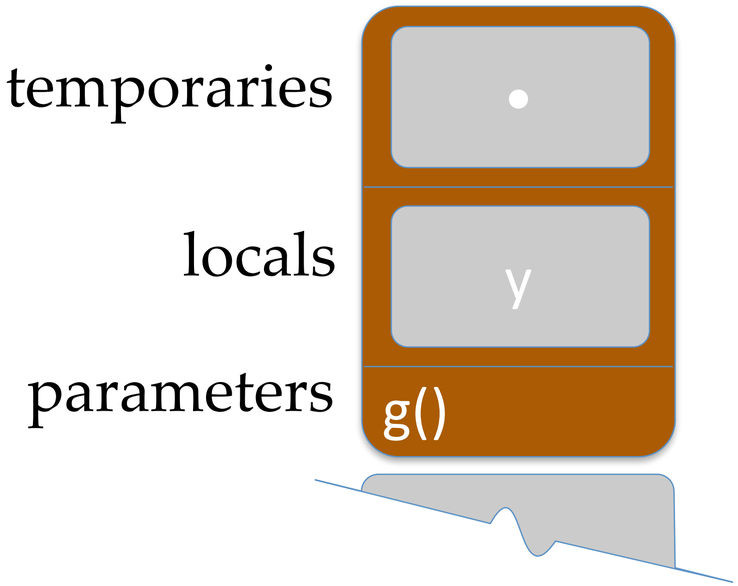
Pass-By-Value Copy Elision

Passing temporaries by value is another copy elision opportunity.

Pass-by-value implies callee can change its *copy* of the argument without being observed by caller

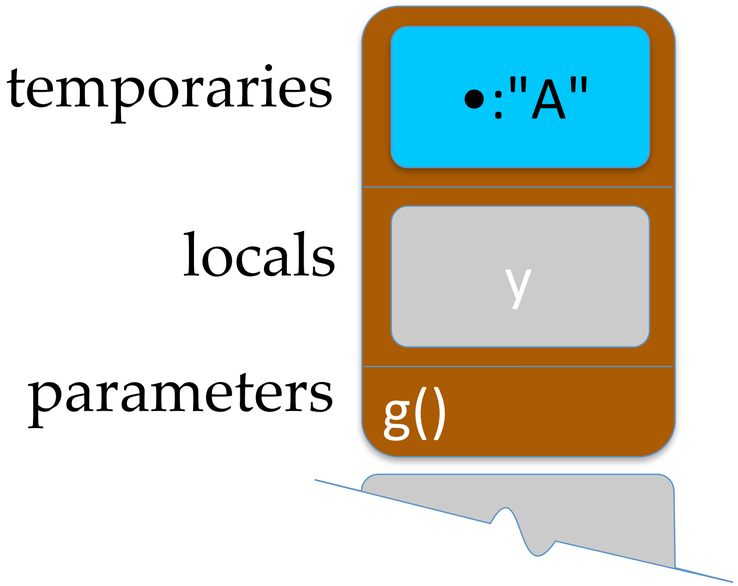
- Caller allocates space for callee's by-value parameters on stack.
- Any *lvalue* arguments get copied into that space (no elision happens)
- Any *rvalue* arguments are simply constructed in that space to begin with

Unoptimized Argument Passing



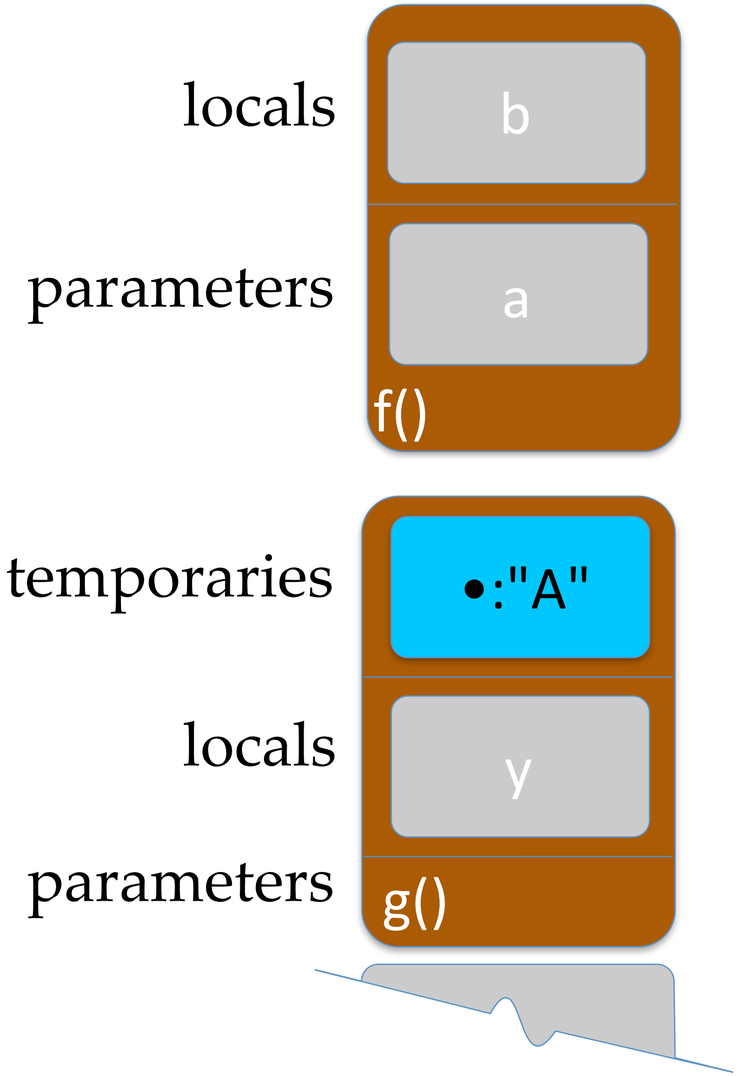

```
void f( std::string a )
{
    int b{23};
    …
    return;
}

void g()
{
    f(std::string{"A"});
    std::vector<int> y;
}
```

Unoptimized Argument Passing



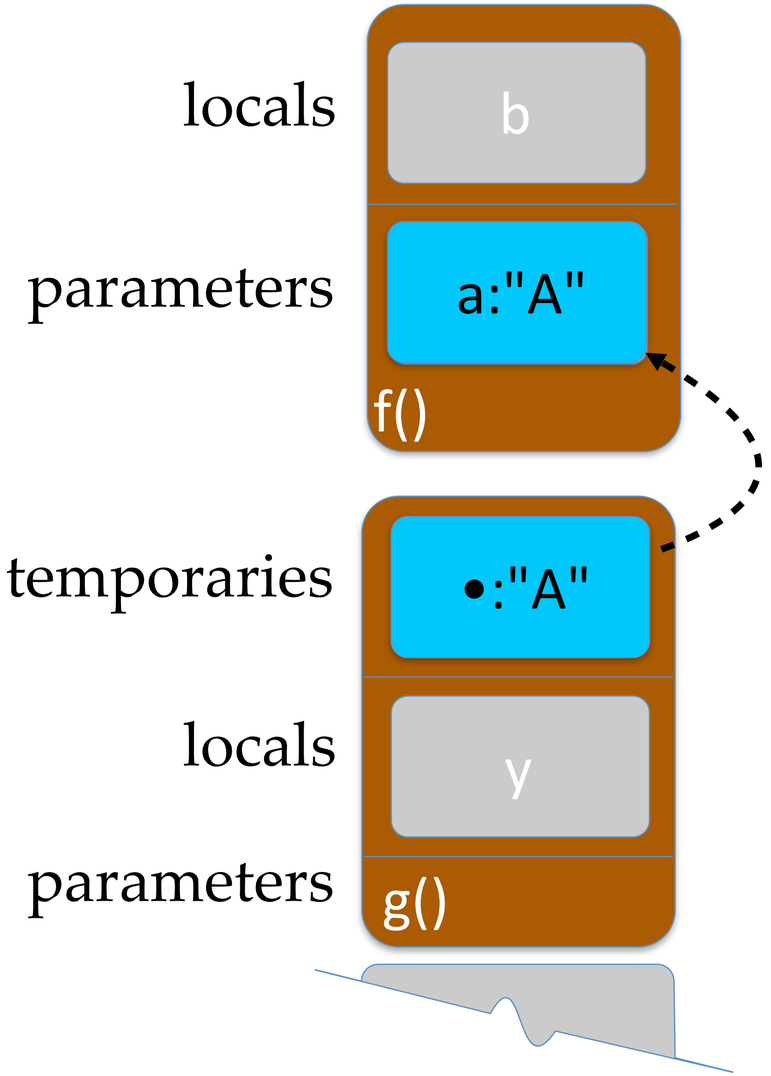

```
void f( std::string a )
{
    int b{23};
    …
    return;
}

void g()
{
    f(std::string{"A"});
    std::vector<int> y;
}
```

Unoptimized Argument Passing



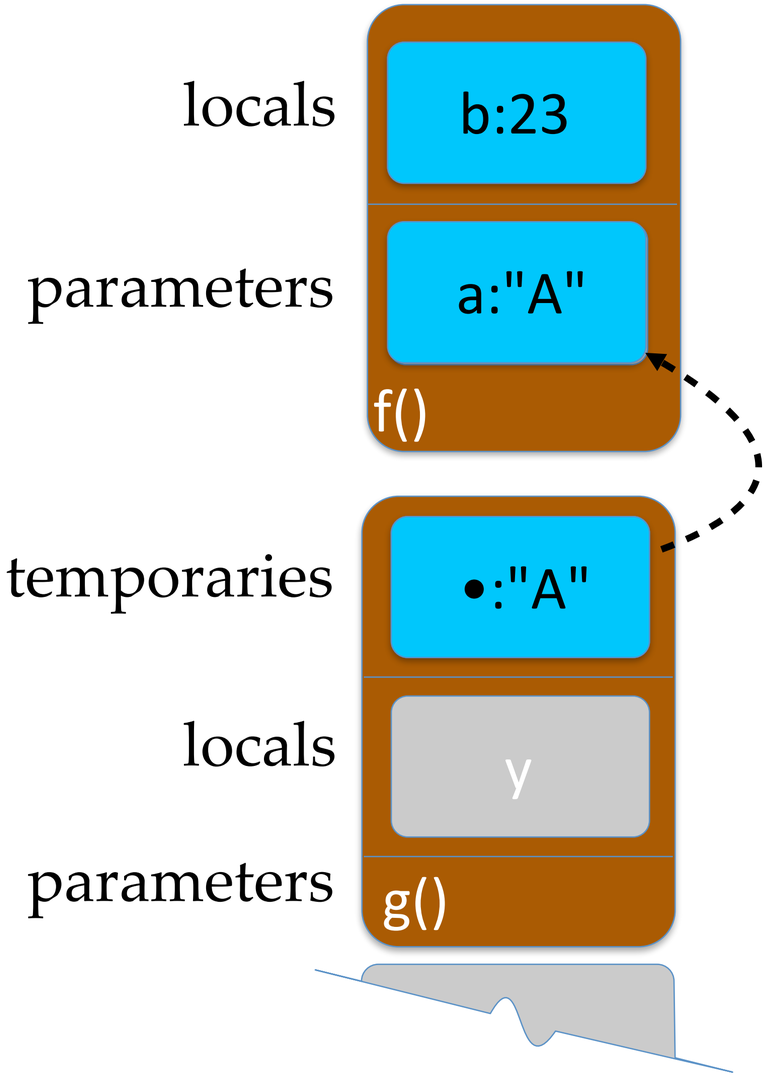

```
void f( std::string a )
{
     int b{23};
     …
     return;
}

void g()
{
     f(std::string{"A"});
     std::vector<int> y;
}
```

Unoptimized Argument Passing



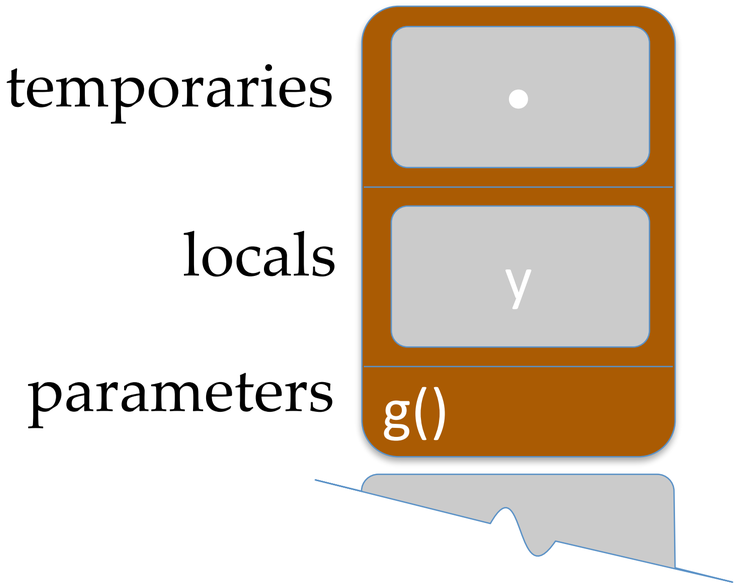

```
void f( std::string a )
{
     int b{23};
     …
     return;
}

void g()
{
     f(std::string{"A"});
     std::vector<int> y;
}
```

Unoptimized Argument Passing




```
void f( std::string a )
{
     int b{23};
     …
     return;
}

void g()
{
     f(std::string{"A"});
     std::vector<int> y;
}
```

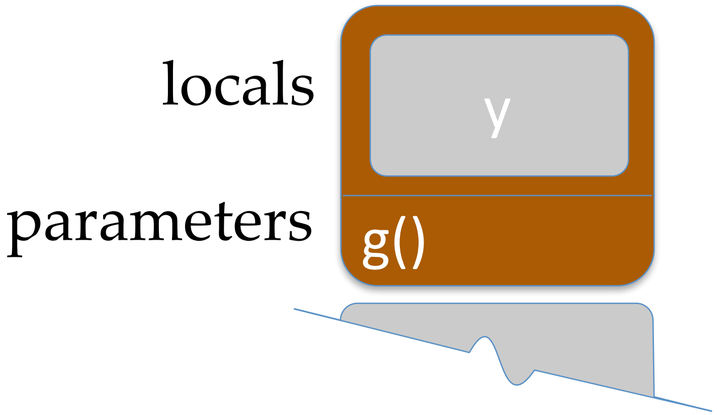
Unoptimized Argument Passing



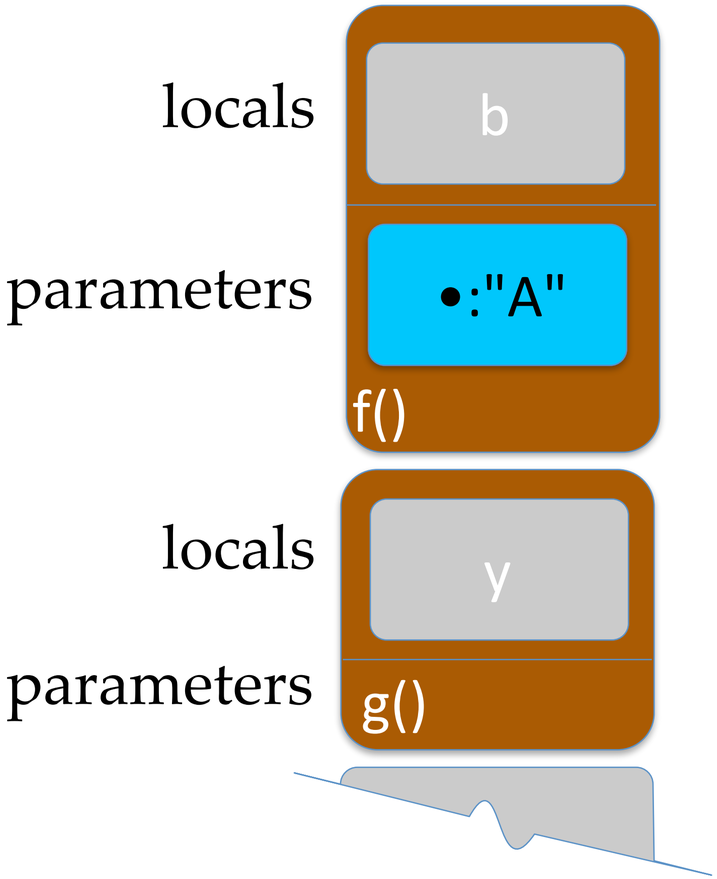

```
void f( std::string a )
{
    int b{23};
    …
    return;
}

void g()
{
    f(std::string{"A"});
    std::vector<int> y;
}
```

Copy Elision: Argument Passing

```
void f( std::string a )
{
    int b{23};
    …
    return;
}

void g()
{
    f(std::string{"A"});
    std::vector<int> y;
}
```

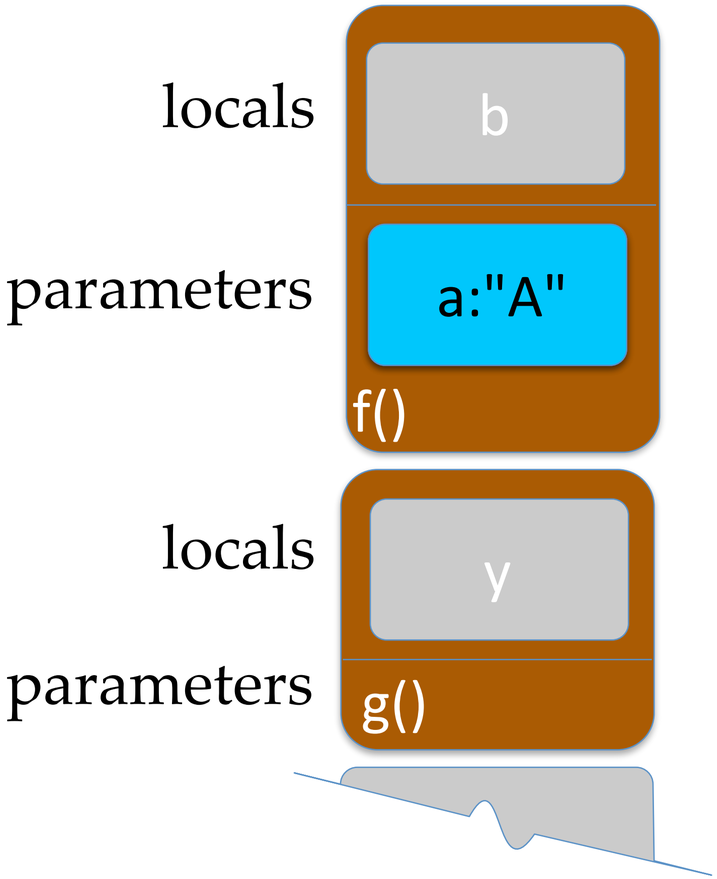


Copy Elision: Argument Passing



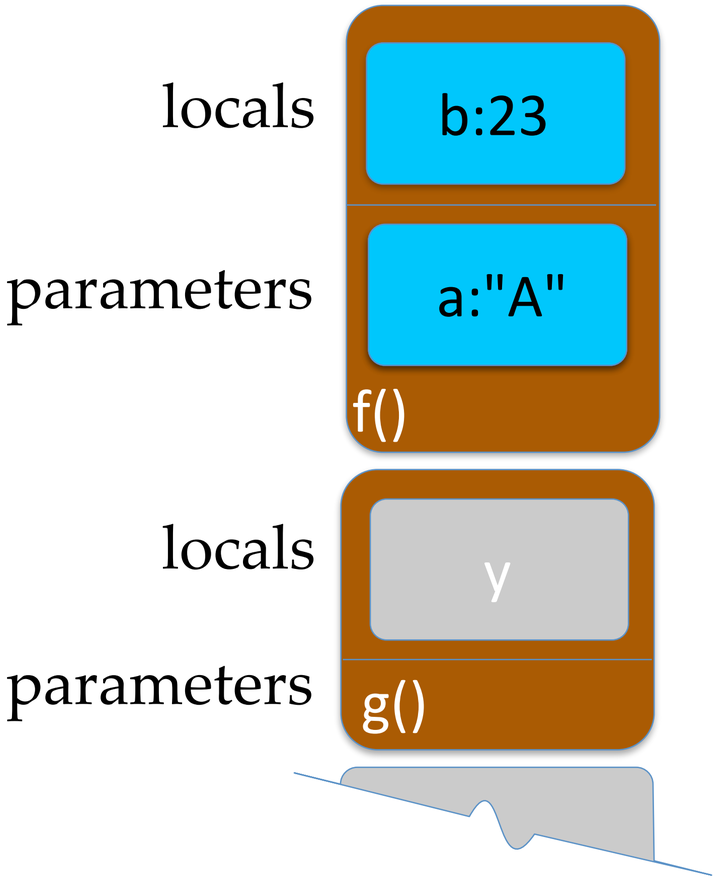

```
void f( std::string a )
{
     int b{23};
     …
     return;
}

void g()
{
     f(std::string{"A"});
     std::vector<int> y;
}
```

Copy Elision: Argument Passing




```
void f( std::string a )
{
    int b{23};
    …
    return;
}

void g()
{
    f(std::string{"A"});
    std::vector<int> y;
}
```

Copy Elision: Argument Passing




```
void f( std::string a )
{
     int b{23};
     …
     return;
}

void g()
{
     f(std::string{"A"});
     std::vector<int> y;
}
```

Copy Elision: Argument Passing

```
void f( std::string a )
{
    int b{23};
    …
    return;
}

void g()
{
    f(std::string{"A"});
    std::vector<int> y;
}
```