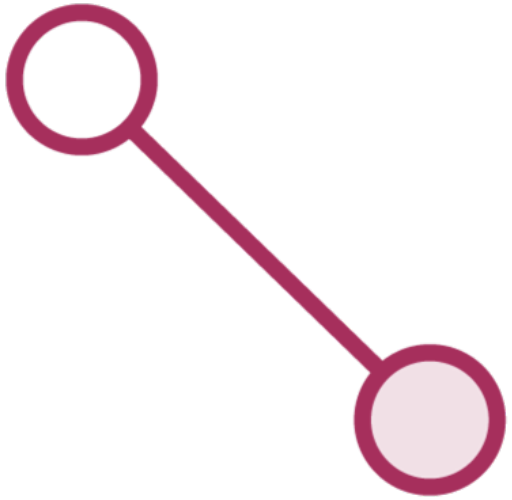
Move Semantics

Kate Gregory

@gregcons www.gregcons.com/kateblog

Move Semantics



Some objects have a pointer to data somewhere else

Copying this data to another object takes time

If you don't need it any more, why bother?

Temporaries

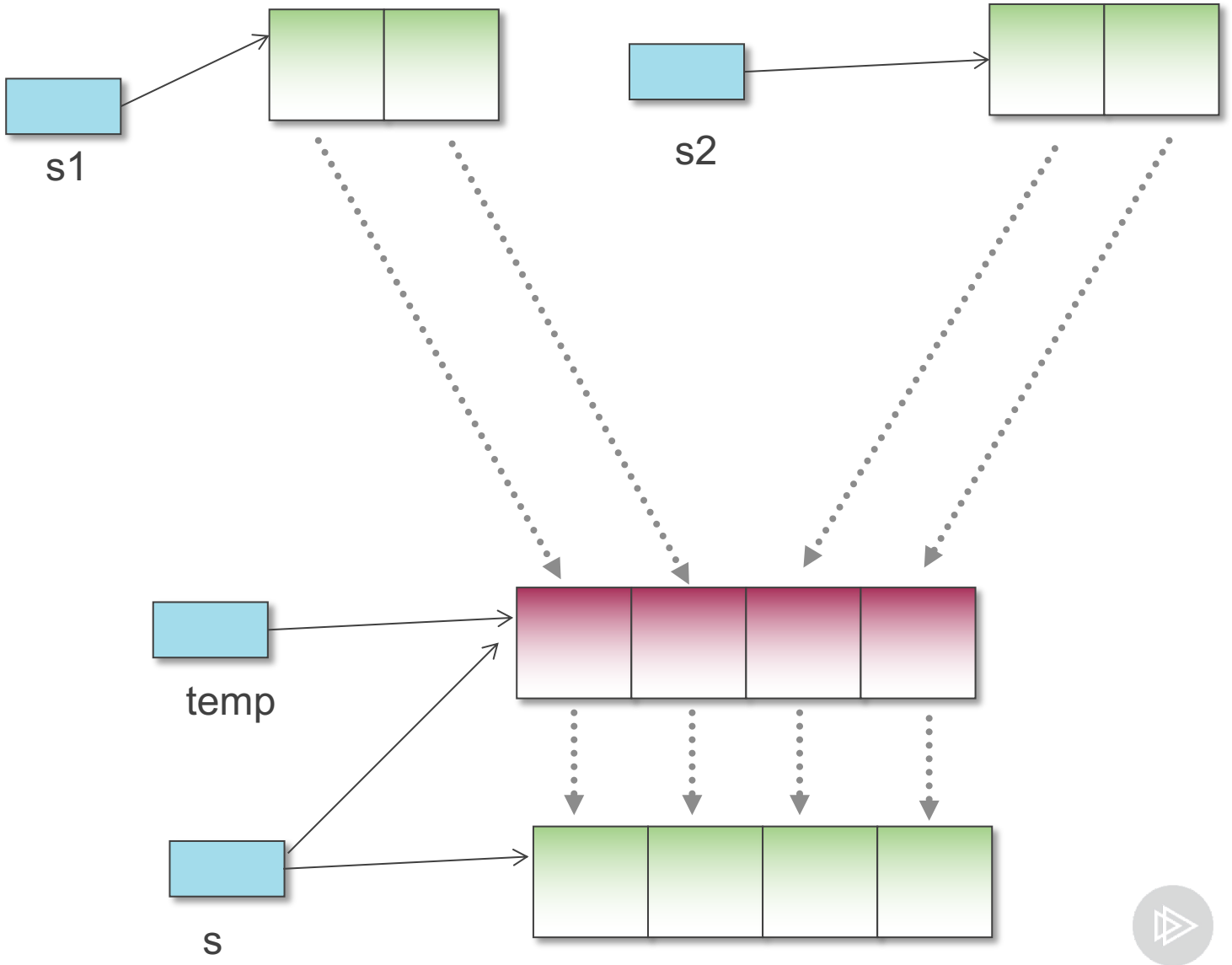
```
string s = s1 +
s2 + s3 + s4;
```

Temporaries like (s3+s4) have no purpose later in the app

Moving is a huge speedup

Moving

string s = s1 + s2;



Rvalue References

If you write a class called Resource, a function can take a Resource parameter many ways:

- Resource x – a solid instance of Resource
- Resource * px – a pointer to some solid instance of Resource
- Resource & rx –a reference to some solid instance of Resource
- Resource && rrx - an rvalue reference to some disappearing instance of Resource

What's an Rvalue?

```
x = 3;
x = a + b;
```

Name refers to RHS, but actual definition is more subtle

- It is something ephemeral
- Perfect for stealing from

Move Constructor, Move Assignment Operator

Like copy constructor and assignment operator

Faster: “steal” elements from the reference they are passed

- Eg make my pointer point the same place as the reference’s pointer

Should leave the passed reference in a valid state

- Empty, null pointer, etc

Already implemented for standard library classes like string and vector

- If your types are movable, vector resizing (and much more) will be faster

```
Resource(Resource&& r);

Resource::Resource(
      Resource&& r):
      name(std::move(r.name))

{}
```

◀ Move constructor

```
Resource& operator=( Resource&& r);

Resource& Resource::operator=(Resource&&
r)

{
        if (this != &r)
        {
                name = std::move(r.name);
                r.name.clear();
        }
        return *this;
}
```

◀ Move assignment operator

◀ Just in case

The Rules You Think You Know Might Be Wrong

Pass by value

- If a temporary is passed to the function, it might be moved, not copied

Return by value

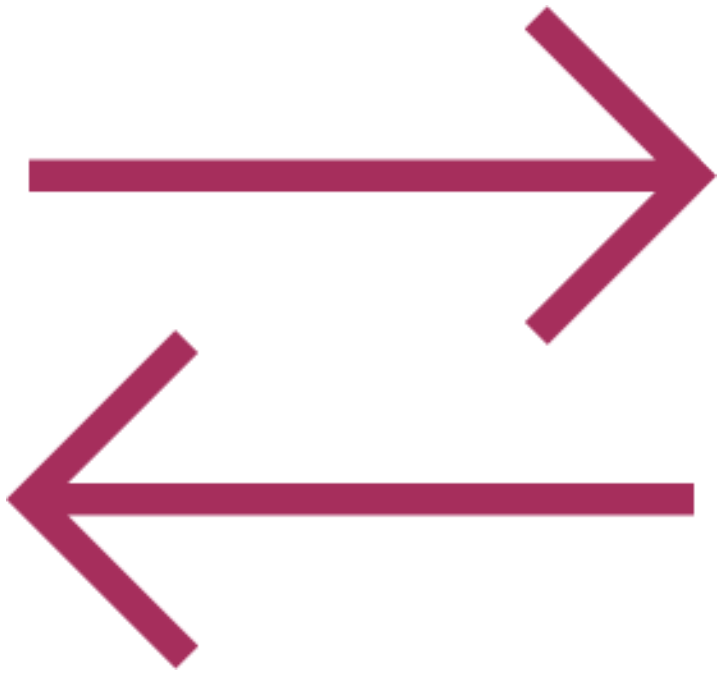
- That local variable is about to go out of scope – compiler will move it

Is vector slow or not?

- When copying becomes moving, the heuristics change

Is it inefficient to build a string from many little pieces?

- Not if you move the pieces as you go



std::move doesn't move anything

It's just a cast

name = std::move(r.name);

It causes the compiler to choose move constructors or move assignment operators

- They might move something

Rule of ...

Rule of 3

If you write a destructor, copy constructor, or assignment operator, write all three

Rule of 5

Add move constructor and move assignment operator to those three

Rule of 0

Use member variables that manage themselves (eg vector, unique_ptr) and write nothing

Summary

Move semantics dramatically boost perf

- May write your own move constructor

If you need to move something: std::move

- Casts a regular reference to an rvalue reference so the right overload is used
- Move a unique_ptr into and out of a collection

Standard Library uses move semantics

- All the collections
- string