Move operations and rvalue references

Wojciech Śmigaj

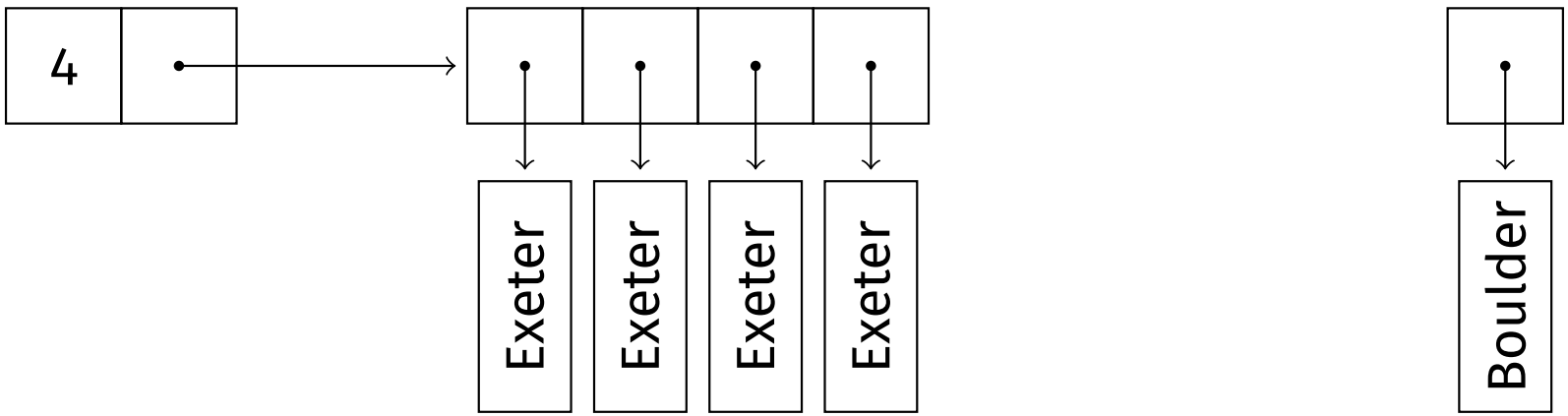
February 13, 2020

Motivation: unnecessary copies in C++03

Met Office

In C++03, appending to a full vector causes a lot of copy operations.

```
std::vector<std::string> v(4, "Exeter");
v.push_back("Boulder");
```

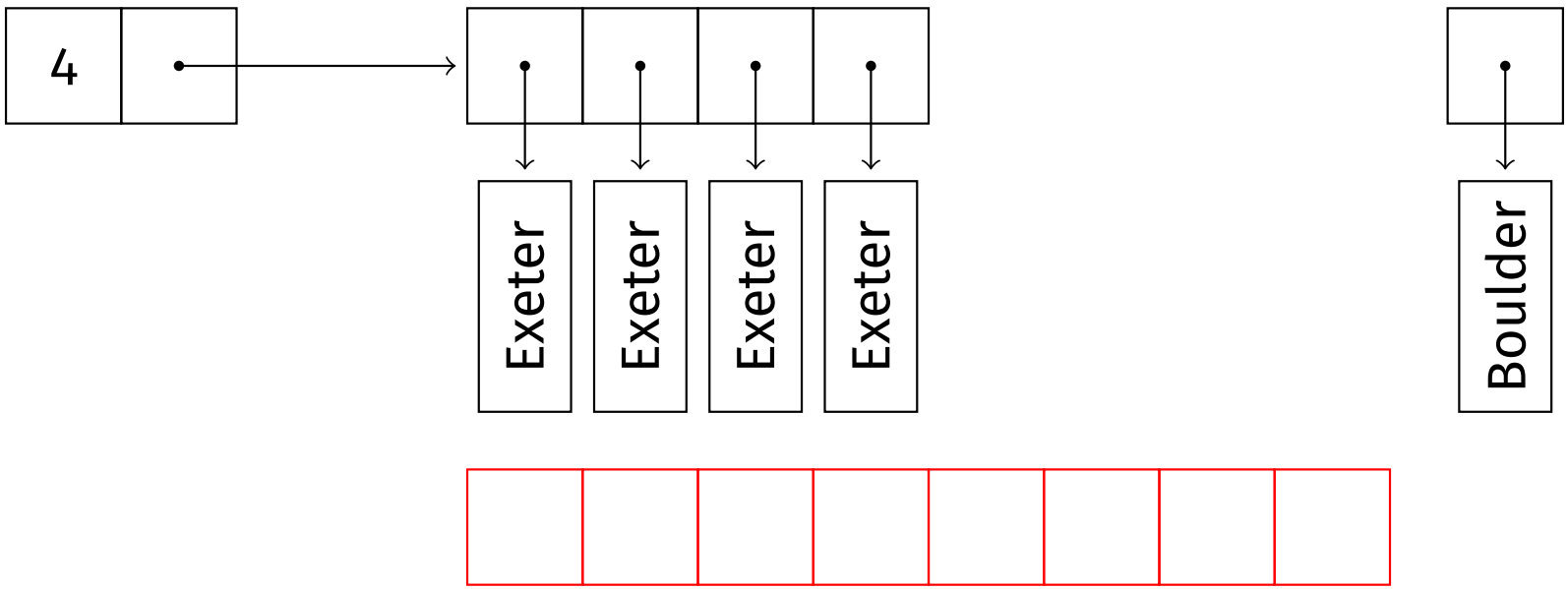


Motivation: unnecessary copies in C++03

Met Office

In C++03, appending to a full vector causes a lot of copy operations.

```
std::vector<std::string> v(4, "Exeter");
v.push_back("Boulder");
```

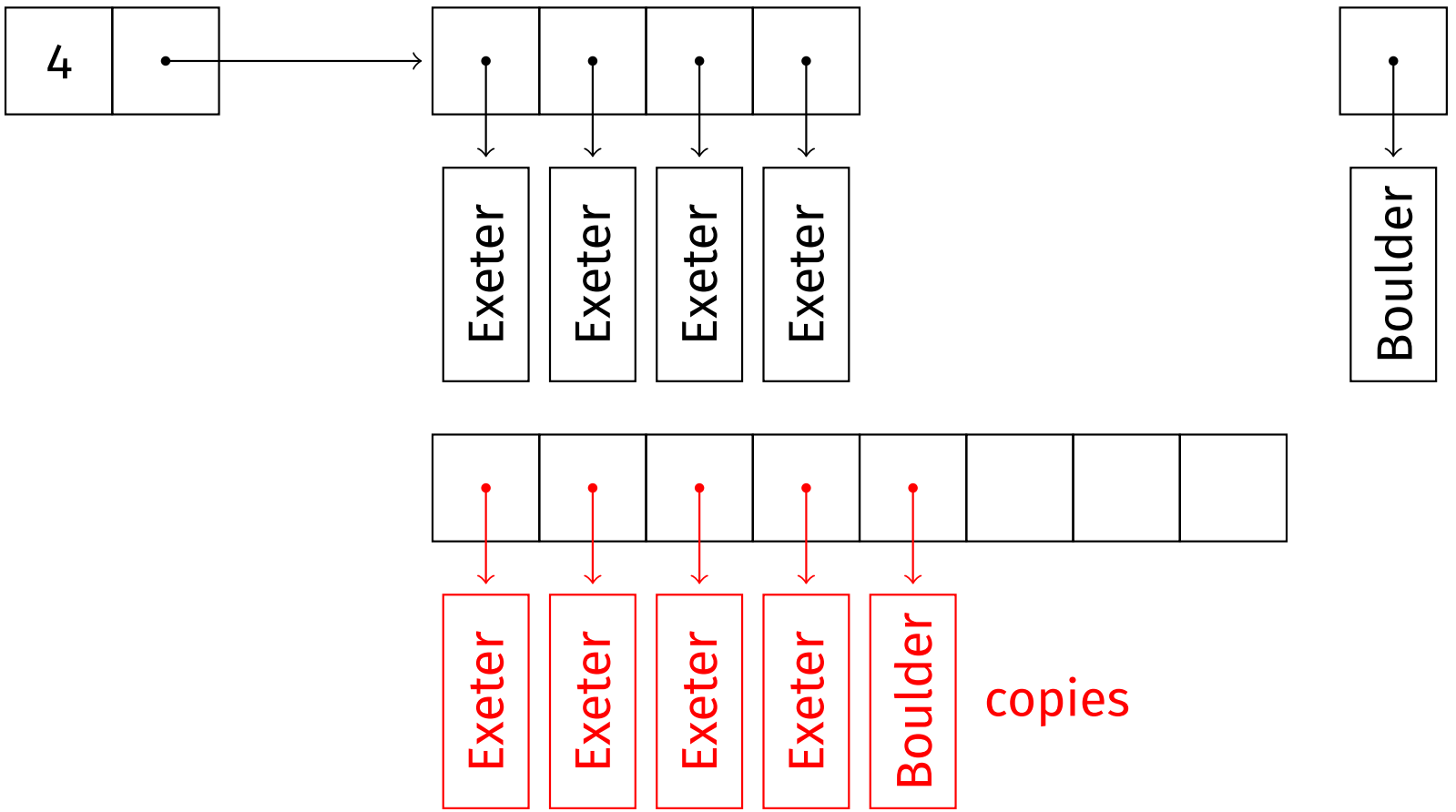


Motivation: unnecessary copies in C++03

Met Office

In C++03, appending to a full vector causes a lot of copy operations.

```
std::vector<std::string> v(4, "Exeter");
v.push_back("Boulder");
```

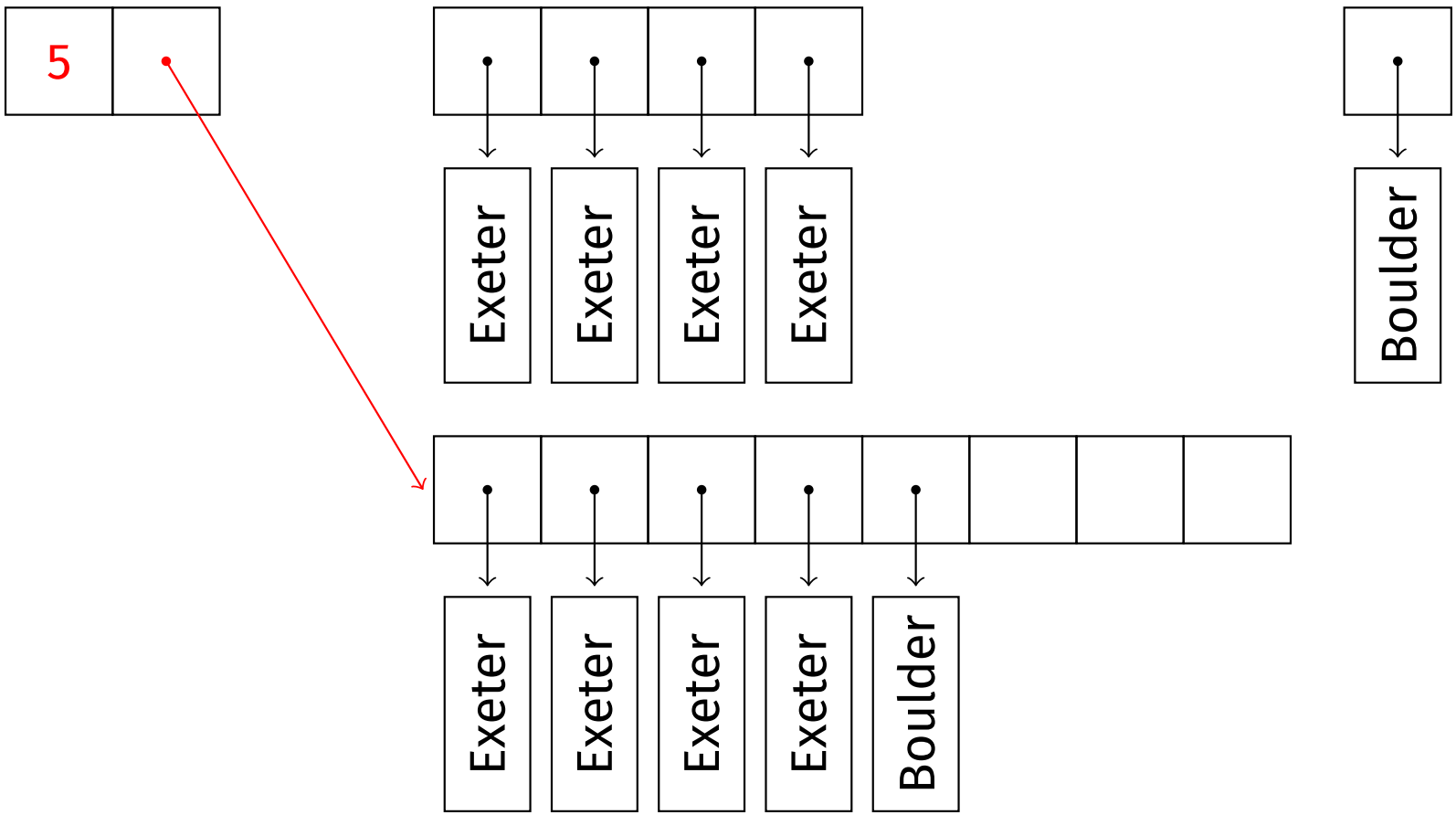


Motivation: unnecessary copies in C++03

Met Office

In C++03, appending to a full vector causes a lot of copy operations.

```
std::vector<std::string> v(4, "Exeter");
v.push_back("Boulder");
```

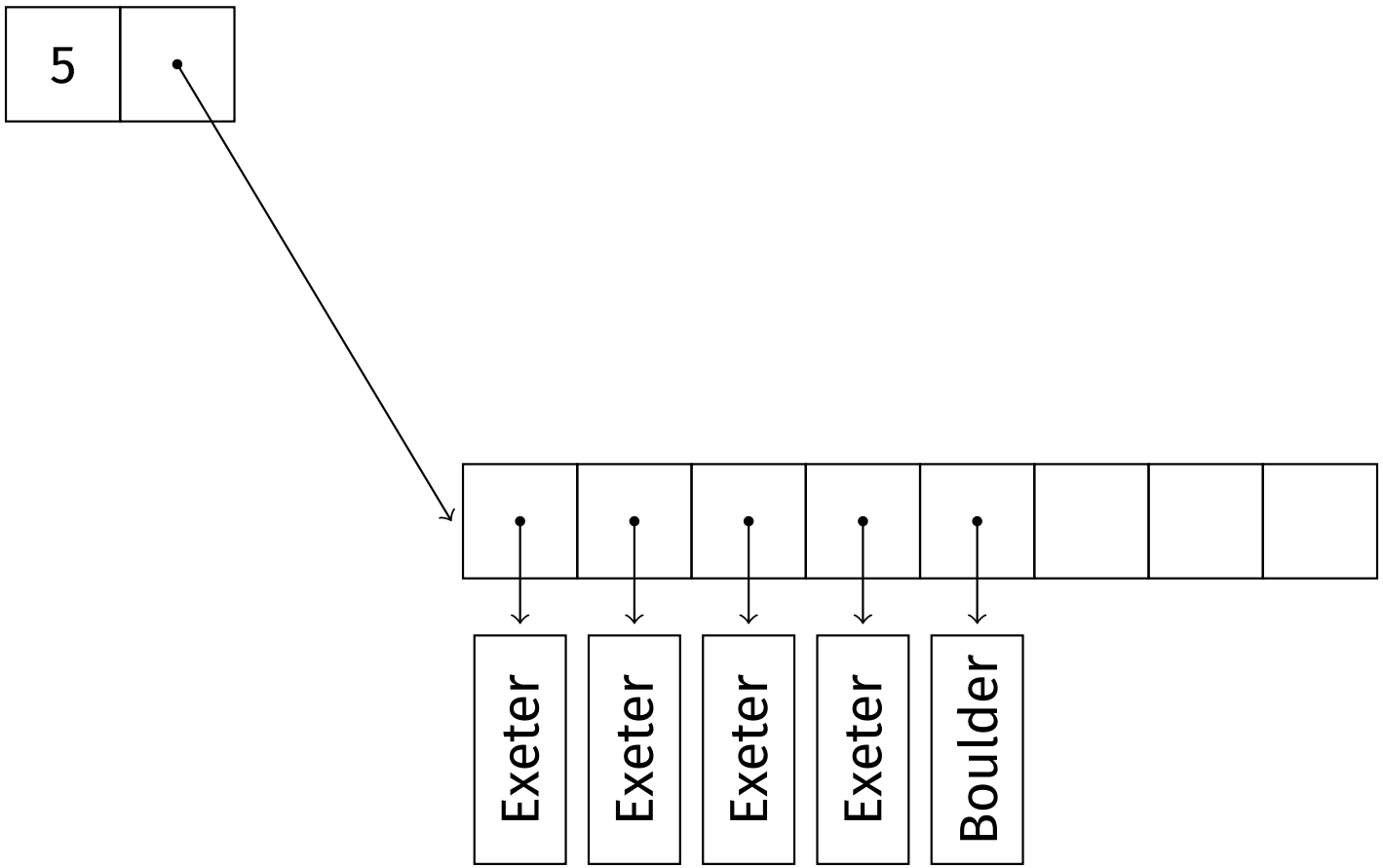


Motivation: unnecessary copies in C++03

Met Office

In C++03, appending to a full vector causes a lot of copy operations.

```
std::vector<std::string> v(4, "Exeter");
v.push_back("Boulder");
```

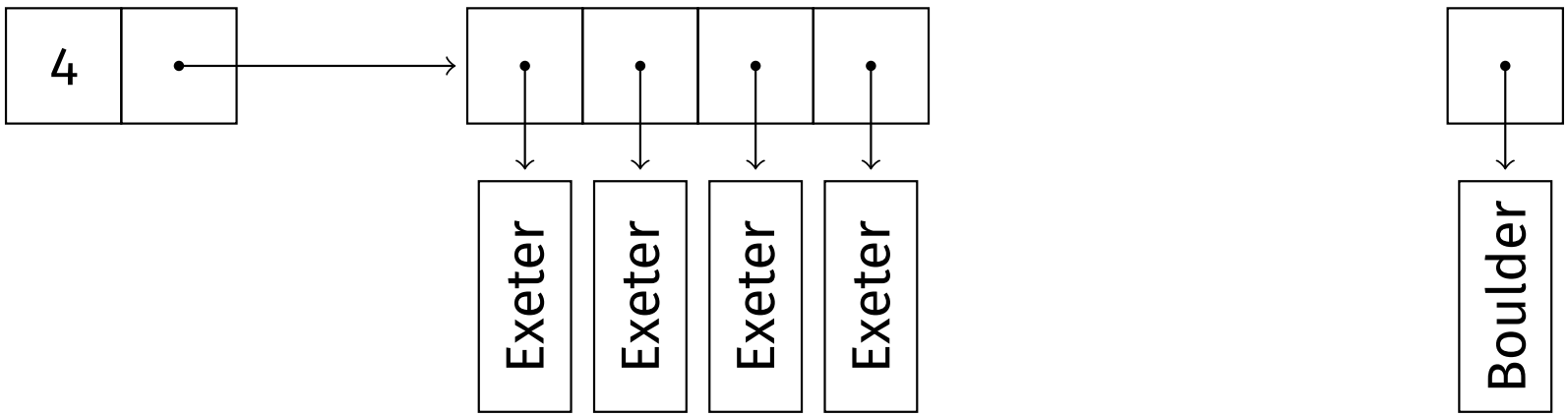


Motivation: unnecessary copies in C++03

Met Office

It would be more efficient to move the strings to the newly allocated block.

```
std::vector<std::string> v(4, "Exeter");
v.push_back("Boulder");
```

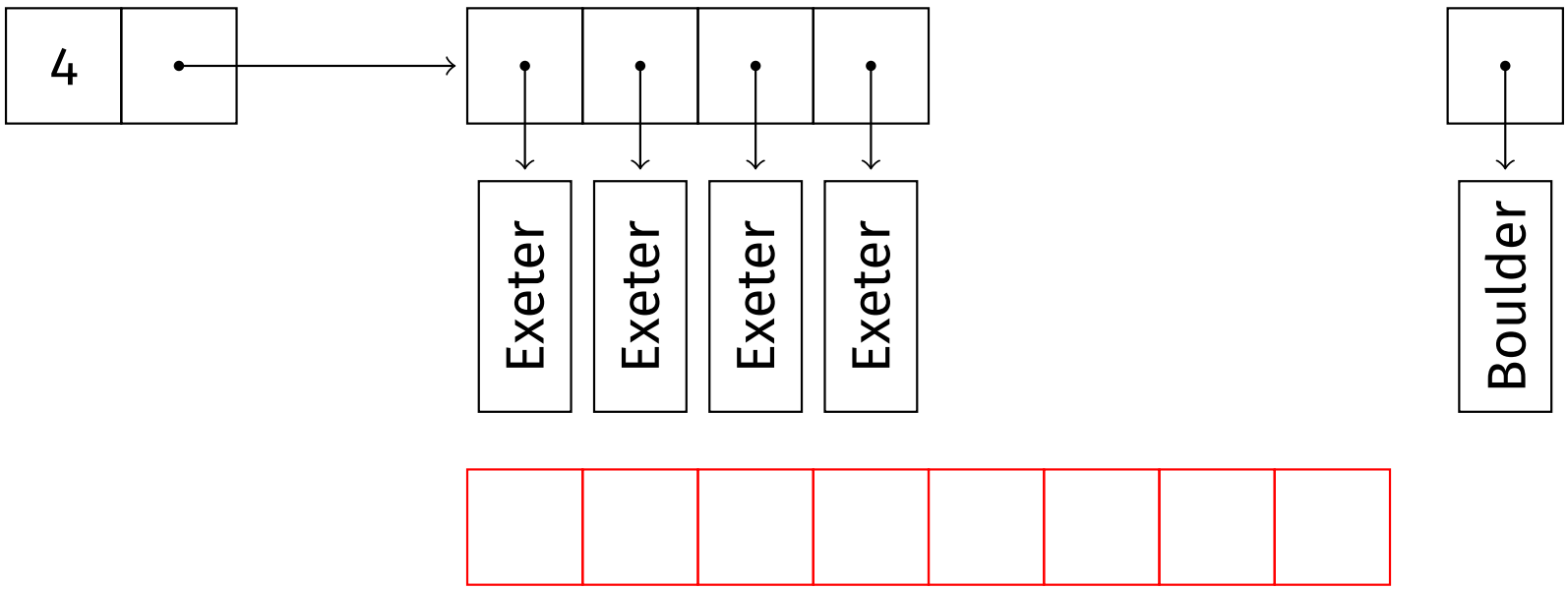


Motivation: unnecessary copies in C++03

Met Office

It would be more efficient to move the strings to the newly allocated block.

```
std::vector<std::string> v(4, "Exeter");
v.push_back("Boulder");
```

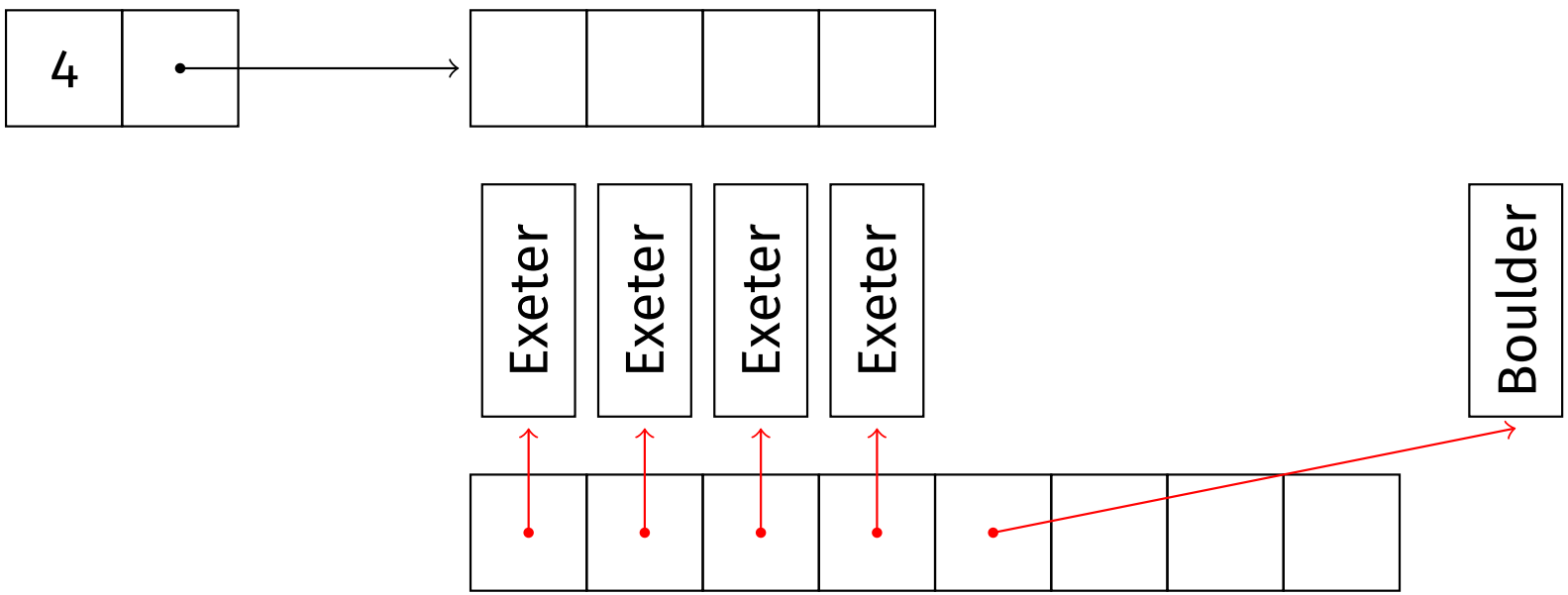


Motivation: unnecessary copies in C++03

Met Office

It would be more efficient to move the strings to the newly allocated block.

```
std::vector<std::string> v(4, "Exeter");
v.push_back("Boulder");
```

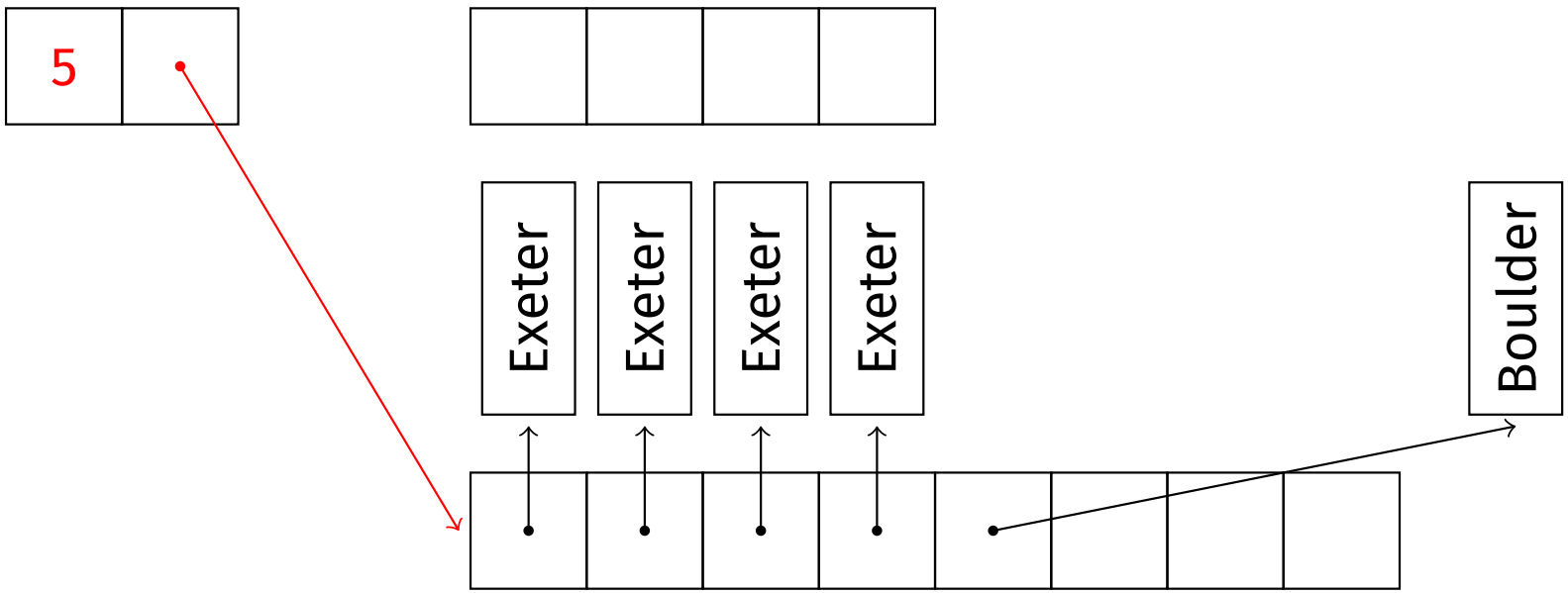


Motivation: unnecessary copies in C++03

Met Office

It would be more efficient to move the strings to the newly allocated block.

```
std::vector<std::string> v(4, "Exeter");
v.push_back("Boulder");
```

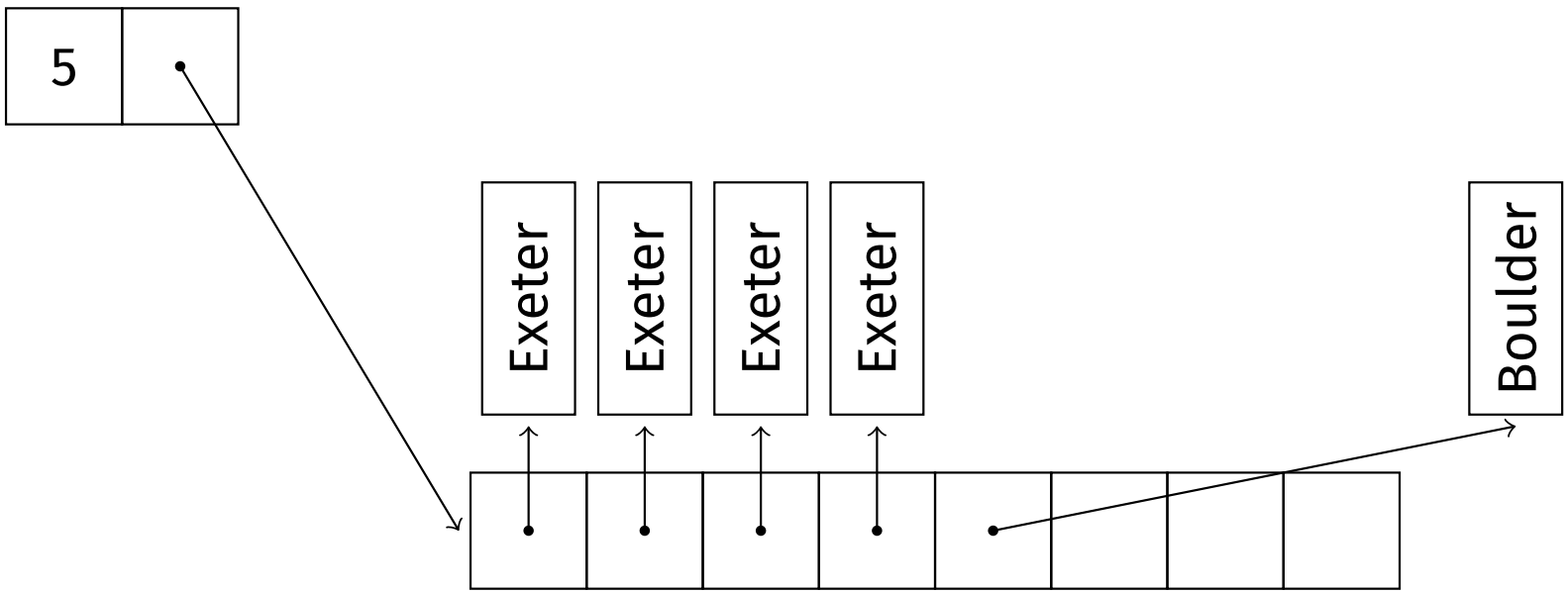


Motivation: unnecessary copies in C++03

Met Office

It would be more efficient to move the strings to the newly allocated block.

```
std::vector<std::string> v(4, "Exeter");
v.push_back("Boulder");
```



C++11 introduces the concept of rvalue references, making it possible to recognise and profit from move opportunities.

Value categories

Met Office

- Expressions in C++ are either lvalues or rvalues.
- Lvalue expressions evaluate to persistent values whose address can be taken, such as
  - names of variables
  - calls to functions returning lvalue references (`T&`, `const T&`)

```
size_t i = 5;
std::vector<int> v(10);
const size_t &ref = i; // i is an lvalue
v.at(ref) = 25;        // v, ref and v.at(ref) too
```

- All other expressions are rvalues. Intuitively, rvalues are typically unnamed temporary objects.

```
int a = 1, b = 2;
int c = (a + b);              // (a + b) is an rvalue,
double x = std::cos(0.0); // 0 and std::cos(0.0) too
```

- In general, value movement is safe only if the source is an rvalue.

Rvalue references

Met Office

C++11 introduces the concept of rvalue references. Unlike “normal” references (*lvalue references*), they may be bound only to rvalues.

Example: Move constructors.

```
class C {
 public:
  C(const C &other);      // copy constructor
  C(C &&other) noexcept; // move constructor
};

C makeC();

int main() {
  C c1;
  C c2(c1);              // lvalue: copy
  C c3(makeC());         // rvalue: move
  C c4(std::move(c1)); // rvalue: move
}
```

Move constructors

Met Office

It is very uncommon to have to implement a move constructor or assignment operator manually. The compiler automatically generates them for each class that fulfils certain conditions, and the default implementation typically does the right thing.

```
class Person {
 public:
  std::string name_;
  int age_;
};
```

The compiler-generated move constructor looks like this:

```
Person::Person(Person &&other) noexcept
  // Call the move constructors of all member variables
  : name_(std::move(other.name_)),
    age_(std::move(other.age_))
{}
```

Requirements

Met Office

Neither a move constructor nor an assignment operator are automatically generated if:

- the class has any user-declared copy constructors, copy assignment operators or destructors
- the class has any data members that can't be moved
- any base class can't be moved.

This restriction applies even if the user-declared special functions are empty or declared as `= default`!

Consequences for JEDI

Met Office

A lot of classes in JEDI declare

- an empty destructor and/or
- a copy constructor that prints a tracing message in addition to copying data members

and are thus non-movable.

Examples: `oops::Variables`, `eckit::LocalConfiguration`, `ioda::ObsVector`.

Guidelines

Met Office

Rule of zero

Avoid having to implement any of the special member functions.

Rule of five

If you have to declare any of the following special member functions:

- destructor
- copy constructor
- move constructor
- copy assignment operator
- move assignment operator

you should normally declare all of them (possibly as `= default` or `= delete`).