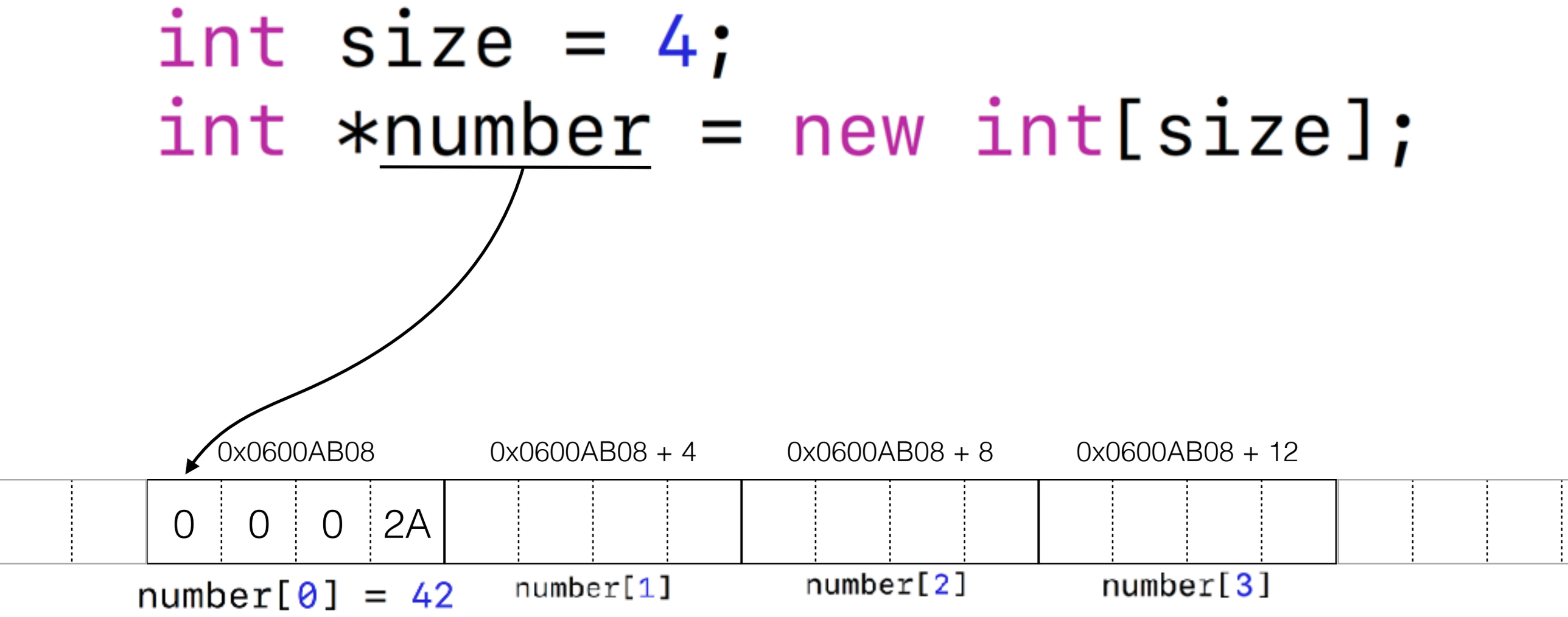
Управление памятью в современном C++

Мусатов Денис
Axmor
21 апреля 2017

Указатель

```
int size = 4;
int *number = new int[size];
```



Динамическое выделение памяти в C++

- new
- delete
- delete[]

Пример #1

```
class Widget { /* ... */ };

void foo()
{
    int n = 42;
    Widget *foo = new Widget();
    char *name = new char[n];
    int *suchWowNumber = new int(1729);

    // ...

    delete foo;
    delete[] name; // Нужно не забыть []!
    delete suchWowNumber;
}
```

Пример #2

```
bool importantCondition();

void foo()
{
    int n = 42;
    Widget *foo = new Widget();
    char *name = new char[n];
    int *suchWowNumber = new int(1729);

    // ...

    if (importantCondition())
        return;

    // ... Ещё 20 строк ...

    delete foo;
    delete[] name;
    delete suchWowNumber;
}
```

Пример #3

```
extern void sometimesThrows();

void foo()
{
    int n = 42;
    Widget *foo = new Widget();
    char *name = new char[n];
    int *suchWowNumber = new int(1729);

    // ...

    sometimesThrows();

    // ... Ещё 20 строк ...

    delete foo;
    delete[] name;
    delete suchWowNumber;
}
```

На помощь пришел C++11

std::unique_ptr

std::shared_ptr

std::weak_ptr

std::unique_ptr

Предназначен для управлением ресурсами с единственным владельцем

Создание std::unique_ptr

```
struct Foo
{
    explicit Foo(std::string name) : name(name) {}

private:
    std::string name;
};
```

Создание std::unique_ptr

```
struct Foo
{
    explicit Foo(std::string name) : name(name) {}

private:
    std::string name;
};
```

- С помощью new:

```
auto object = std::unique_ptr<Foo>(new Foo("widget"));
```

Создание std::unique_ptr

```
struct Foo
{
    explicit Foo(std::string name) : name(name) {}

private:
    std::string name;
};
```

- ~~С помощью new:~~

```
~~auto object = std::unique_ptr<Foo>(new Foo("widget"));~~
```

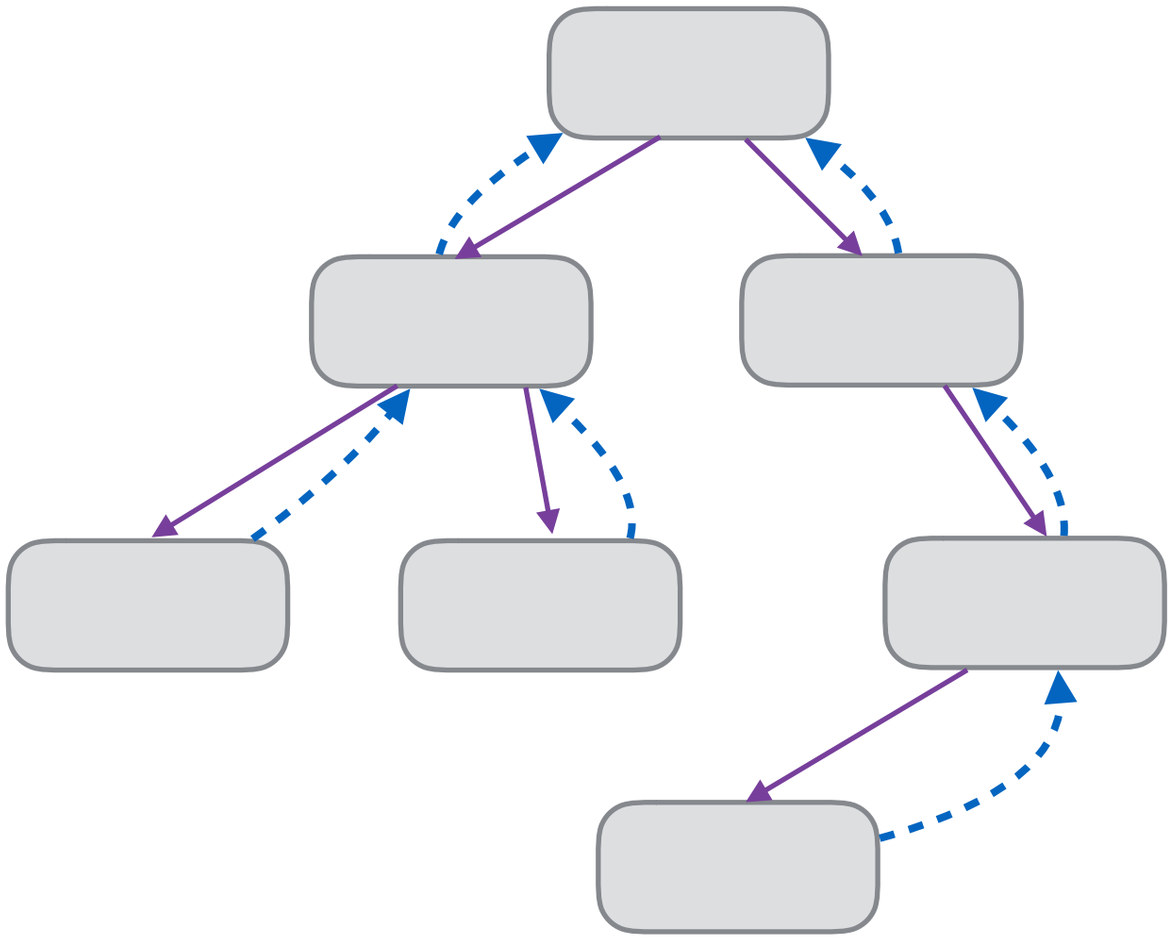
- С помощью std::make_unique (C++14): ✔

```
auto object = std::make_unique<Foo>("widget");
```

std::unique_ptr. Пример

```
std::unique_ptr<File> makeFileNamed(std::string name)
{
    FileType type = utils::extension2type(utils::getExtension(name));
    switch (type) {
        case FileType::Fasta:
            return std::make_unique<FASTAFile>(name);

        case FileType::Fastq:
            return std::make_unique<FASTQFile>(name);

        case FileType::Csv:
            return std::make_unique<CsvFile>(name);

        case FileType::Tsv:
            return std::make_unique<TsvFile>(name);

        default:
            return nullptr;
    }
}
```

std::unique_ptr. Дерево



std::unique_ptr. Дерево

```
struct Tree {
    struct Node {
        std::unique_ptr<Node> left;
        std::unique_ptr<Node> right;

        Node* parent;
    };

    std::unique_ptr<Node> root;
};
```

std::unique_ptr
Следует запомнить

- Имеет размер обычного указателя:

```
sizeof(std::unique_ptr<Foo>) == sizeof(Foo *)
```

- Такой же быстрый, как и new – delete

- При необходимости можно создать std::shared_ptr

std::shared_ptr

Используются для управления общими ресурсами

Является указателем, подсчитывающим количество ссылок, указывающих на объект. Когда это количество становится равным 0, объект автоматически удаляется.

Создание std::shared_ptr

- С помощью new:

```
auto object = std::shared_ptr<Foo>(new Foo("widget"));
```

Создание std::shared_ptr

- С помощью new:

```
auto object = std::shared_ptr<Foo>(new Foo("widget"));
```

- С помощью std::make_shared:

```
auto object = std::make_shared<Foo>("widget");
```

std::shared_ptr. Пример

```
std::shared_ptr<Foo> anotherPointer;

{
    auto object = std::make_shared<Foo>("widget");
    anotherPointer = object; // счетчик ссылок +1
}
// Объект, на который указывал object, ещё существует

/* ... */

anotherPointer = nullptr; // Тут счётчик стал =0
// Объект тут же был удален,
// память освобождена
```

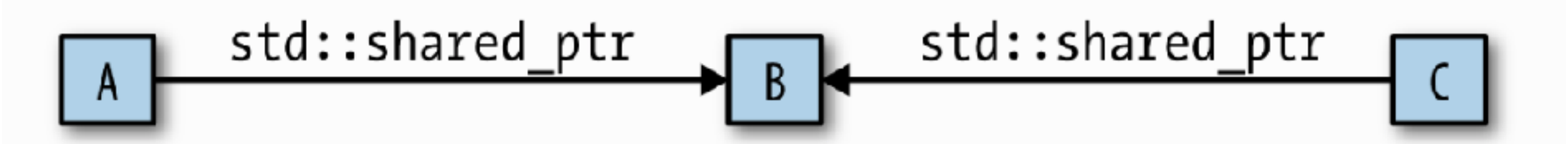
std::shared_ptr
Следует запомнить

- Позволяет достичь предсказуемой автоматической сборки мусора
- Нужно остерегаться образования циклических ссылок
- Все операции со счетчиком ссылок атомарны
- Размер указателя – вдвое больше обычного

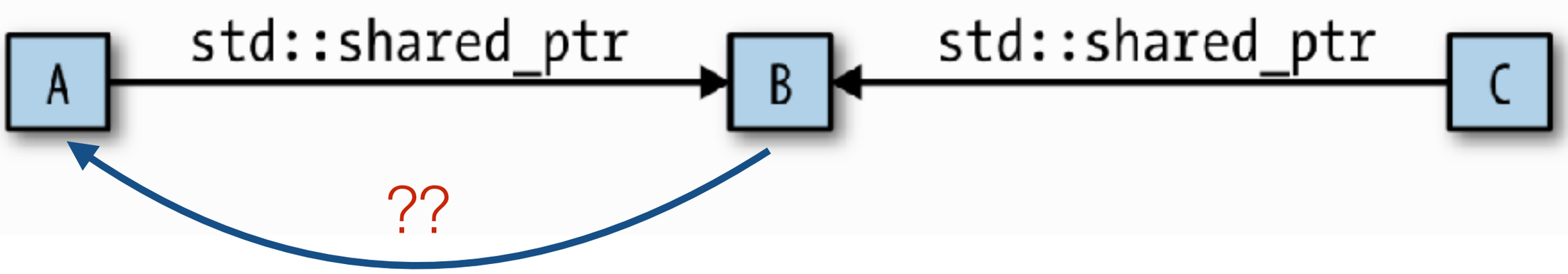
std::weak_ptr

Указывает на объект, хранящийся по std::shared_ptr, но при этом не увеличивает счетчик ссылок на тот объект

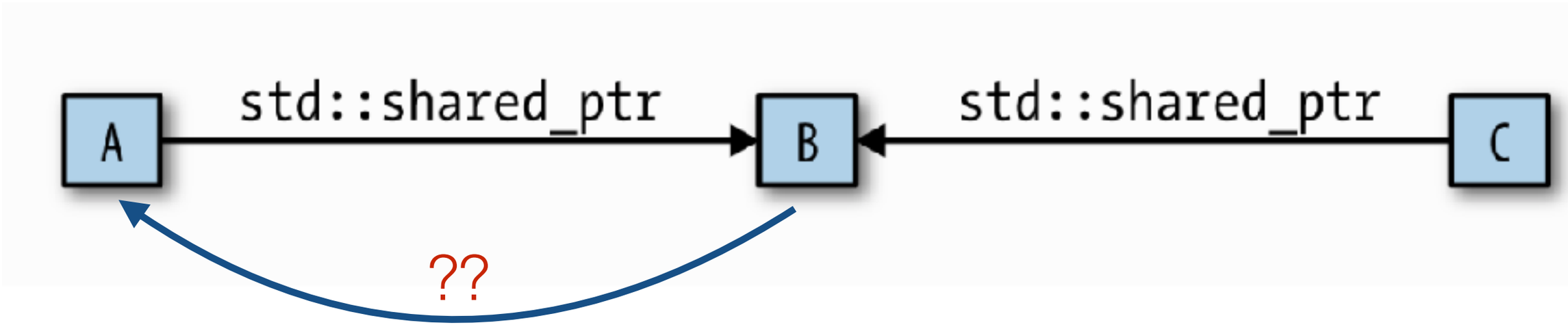
std::weak_ptr



std::weak_ptr

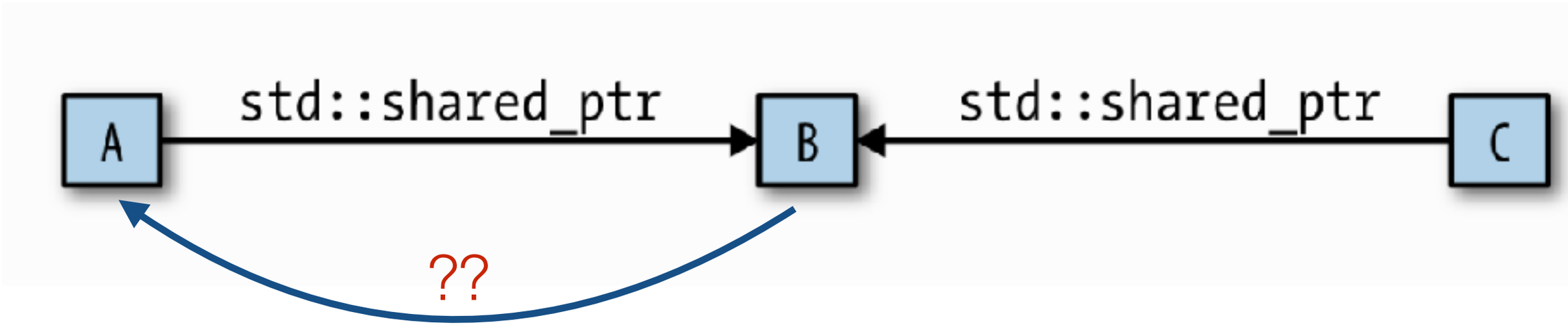


std::weak_ptr

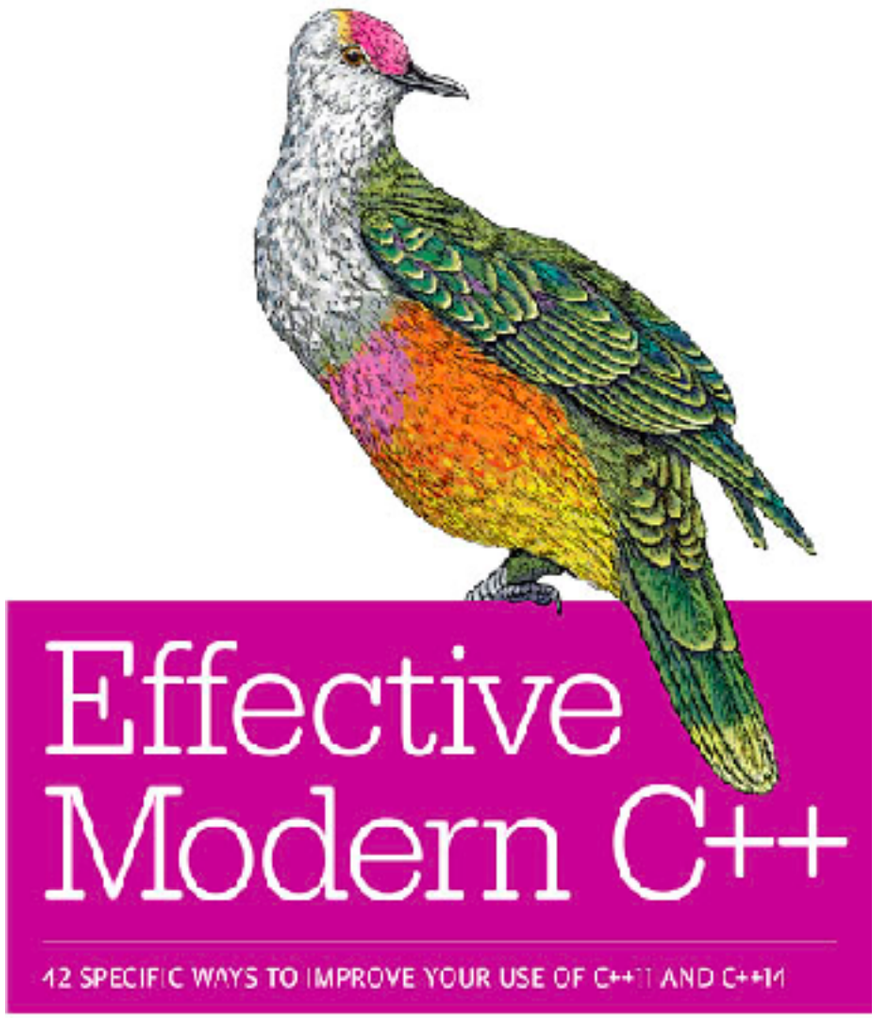


- Использовать обычный указатель A*
- Использовать std::shared_ptr<A>
- Использовать std::weak_ptr

std::weak_ptr



- ~~Использовать обычный указатель A*~~
- ~~Использовать std::shared_ptr<A>~~
- Использовать std::weak_ptr ✓



Effective Modern C++

42 Specific Ways to Improve Your Use of C++11 and C++14 (2014)

Автор: Scott Meyers

Q&A