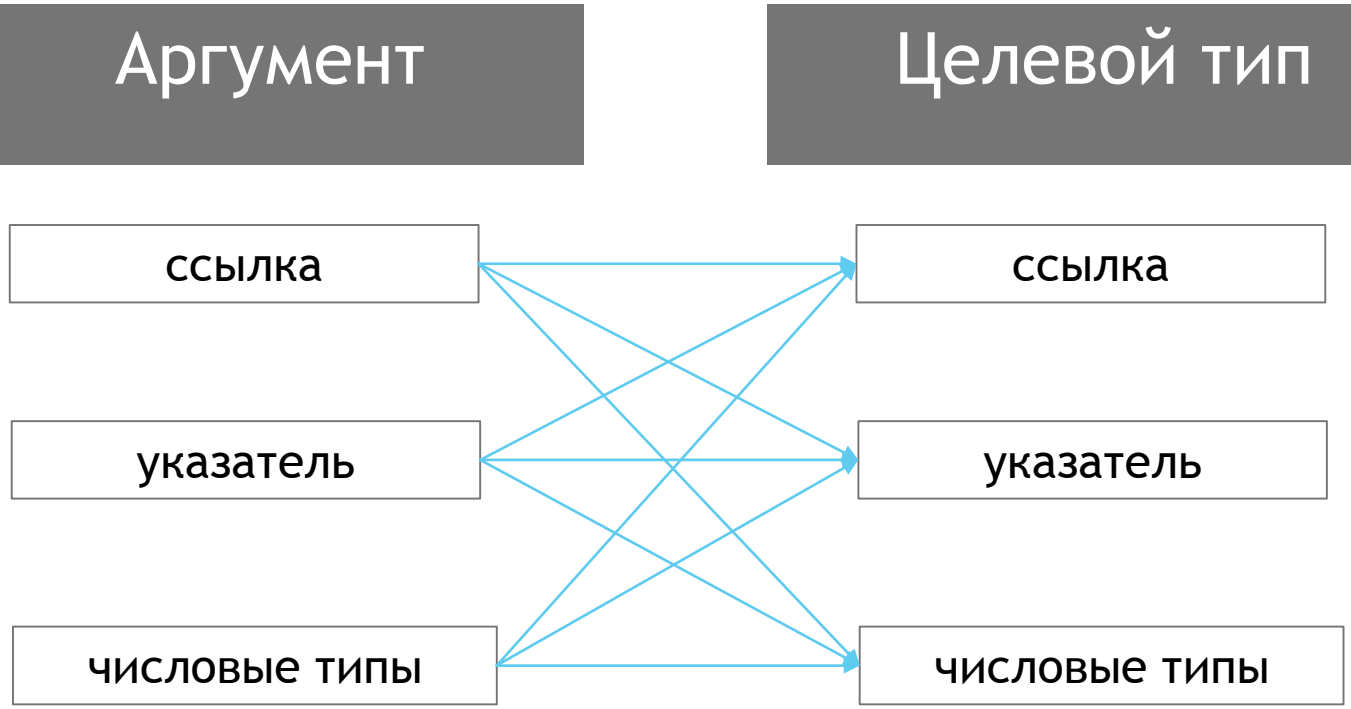
Приведение типов и пространства имен

Тема 9

Операции приведения типов

- `reinterpret_cast <целевой_тип> (аргумент)`
- `const_cast <целевой_тип> (аргумент)`
- `static_cast <целевой тип> (аргумент)`
- `dynamic_cast <целевой_тип> (аргумент)`

reinterpret_cast



Приведение несвязанных типов

Использование reinterpret_cast

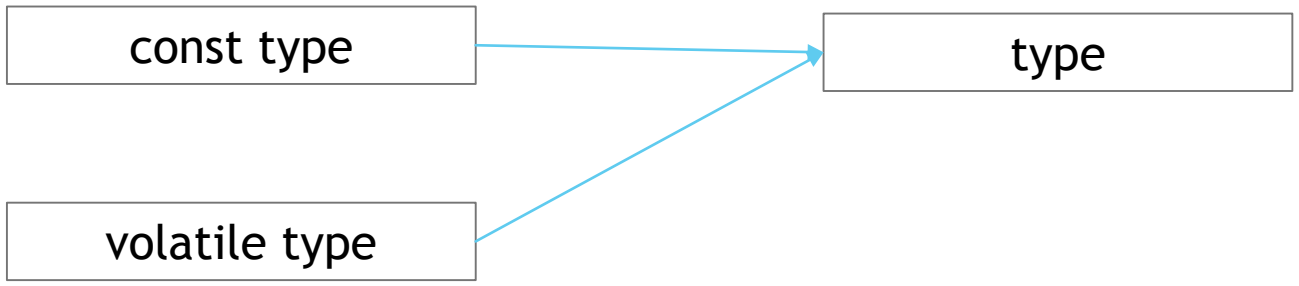
```
int i = 7;

int *ip = &i;

int temp = reinterpret_cast<int>(ip);

std::cout << "Pointer value is "<< ip << std::endl;

std::cout << "Representation of a pointer as int is "
          << hex << temp << std::endl;

std::cout << "Convert it back and dereference:"
          << reinterpret_cast<int*>(temp) << std::endl;
```

Использование reinterpret_cast

```
class A {};
class B {};
int main(){
    A a;
    A&ref = a;
    A* p = &ref;
    B & r = reinterpret_cast<B&>(p);
    B b;
    ref  = reinterpret_cast<A&>(b);
    B * pp = reinterpret_cast<B*>(p);
}
```

const_cast

Аргумент	Целевой тип



Использование const_cast

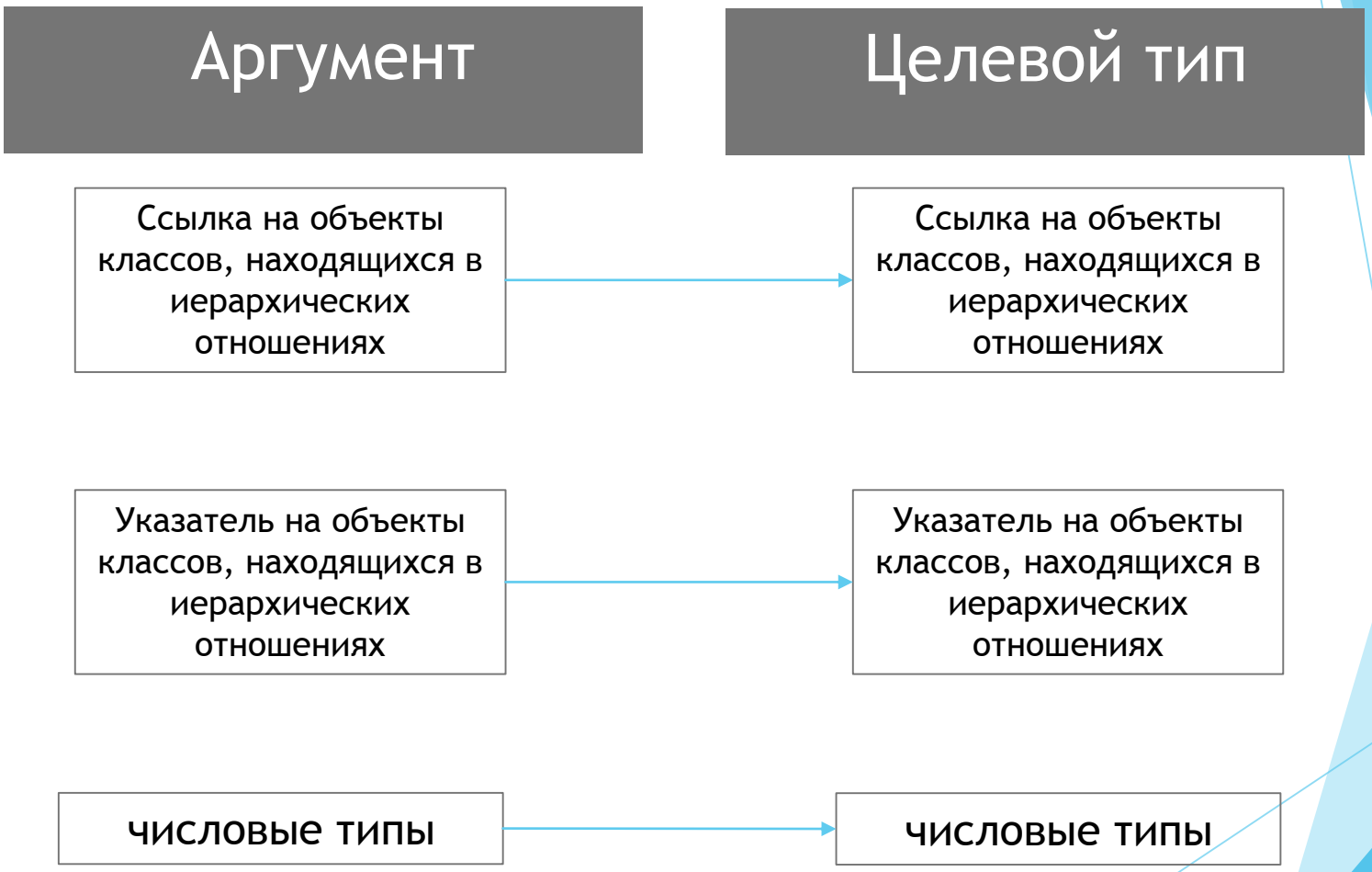
```
const char *ip = nullptr;
ip = new char[20];
strcpy(const_cast<char*>(ip), "New const string");
std::cout << ip << std::endl;
strcpy(const_cast<char*>ip, "Test");
std::cout << ip << std::endl;
delete [] ip;
```

Использование const_cast

```
void f(const Employee* pemp)
{
    Employee *pp = const_cast< Employee*>(pemp);
    pp->new_name("Petya");
}
```

```
const Employee emp("Vasya",...);
f(&emp); //???
```

static_cast


Аргумент
Целевой тип
Ссылка на объекты классов, находящихся в иерархических отношениях
Ссылка на объекты классов, находящихся в иерархических отношениях
Указатель на объекты классов, находящихся в иерархических отношениях
Указатель на объекты классов, находящихся в иерархических отношениях
числовые типы
числовые типы

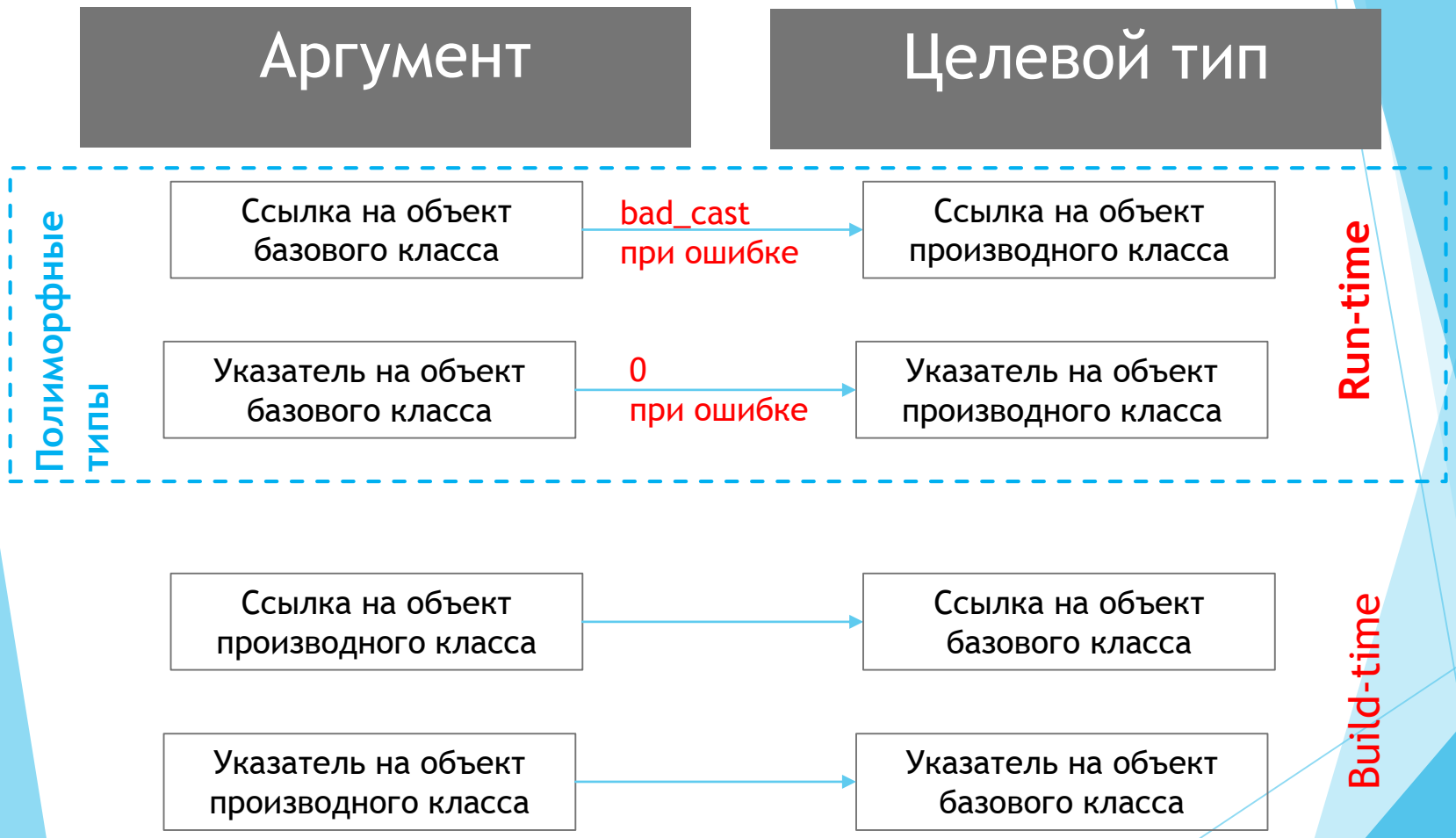
Использование static_cast

```
class Manager : public Employee {
    struct Subordinate    {
        Employee* emp;
        std::string todo;
        Date start, finish, deadline;
    };
public:
    bool SetBonus();
private:
    Subordinate ** arr;
protected:
    Subordinate* Arr(i);
};
```

Использование static_cast

```
bool Supervisor::SetBonus(Manager *man)
{
    ...
    if((static_cast<Manager*>(Arr(i)->emp))->SetBonus())
        *Arr(i)->emp + Arr(i)->emp->GetSalary() / 3;
    ...
}
```

dynamic_cast



Использование dynamic_cast

```
B b(22), *pb = &b;
A a, *ppa=&a;
A &ra = dynamic_cast<A&>(b);  // Ссылка на b как базовый объект.
A *pa = dynamic_cast<A*>(&b); // Указатель на b как базовый объект.
std::cout << "Derived object: " << b.i << std::endl;
std::cout << "Downcasting pointer to pointer: "
          << dynamic_cast<B*>(pa)->i << std::endl;  // Приведение указателей.
try{
  cout << "Downcasting referense to referense: "
       << dynamic_cast<B&>(ra).i<< std::endl;       // Приведение к ссылке.
}
catch(bad_cast)
{}
std::cout << "Downcasting reference to object: ";
std::cout << static_cast<B*>(ppa)->i<< std:: endl;  // Приведение к объекту.
```

Использование dynamic_cast

```
class Employee {
public:
    virtual ~Employee(){}
    ...
};
```

Использование dynamic_cast

```
class Manager : public Employee {
    struct Subordinate {
        Employee* emp;
        std::string todo;
        Date start, finish, deadline;
    };
public:
    virtual bool SetBonus();
private:
    Subordinate ** arr;
protected:
    Subordinate* Arr(i);
};
```

Использование dynamic_cast

```
int main()
{
    Employee * arr[10];
    ...
    Supervisor* p = dynamic_cast<Supervisor *>(arr[i]);
    if(p != nullptr)
        p->SetBonus();
}
```

Пространства имен

- Задают логическую организацию данных в программе
- Позволяют локализовать идентификаторы
- Определяют область видимости
- Одно пространство имен может распространяться на несколько файлов
- В одном файле может быть несколько пространств имен
- Пространства имен можно вкладывать друг в друга

Создание пространства имен

```
namespace name
{
    class Date{…};
    int n;
    void function(void);
}
```

Использование пространства имен

- **`using`**-объявление
- **`using`**-директива

using-объявление

```
using name::Date;
using name::n;
```

```
Date date;
n = 15;
name::function();
```

using-директива

```
using namespace name;
```

```
n = 15;
Date date;
function();
```

Псевдонимы пространства имен

```
namespace First
{
        namespace Second
        {
                namespace Third
                {
                        int test;
                }
        }
}
```

```
namespace FST = First::Second::Third;
int s = FST::test;
```

Объединение и отбор пространств имен

file1.h

```
namespace A{ void f(); void test();}
```

file2.h

```
namespace B{ class String{}; void foo();}
```

file3.h

```
namespace C
{
    using namespace A;
    using namespace B;
    void g();
}
```

test.cpp

```
using namespace C;
int main()
{
        f();
        g();
        String s;
}
```

Объединение и отбор пространств имен

file1.h

```
namespace A{ void f(); void test();}
```

file2.h

```
namespace B{ class String{}; void foo();}
```

file3.h

```
namespace C
{
    using A::test;
    using B::foo;
    void g();
}
```

test.cpp

```
using namespace C;
int main()
{
    f();        //!!!
    g();
    String s;   //!!!
}
```

Неименованные пространства имен

Неименованное пространство имен ограничивает идентификаторы рамками файла, в котором они объявлены

```
namespace
{
        int source;
        char * stroka;
}
```

Конец