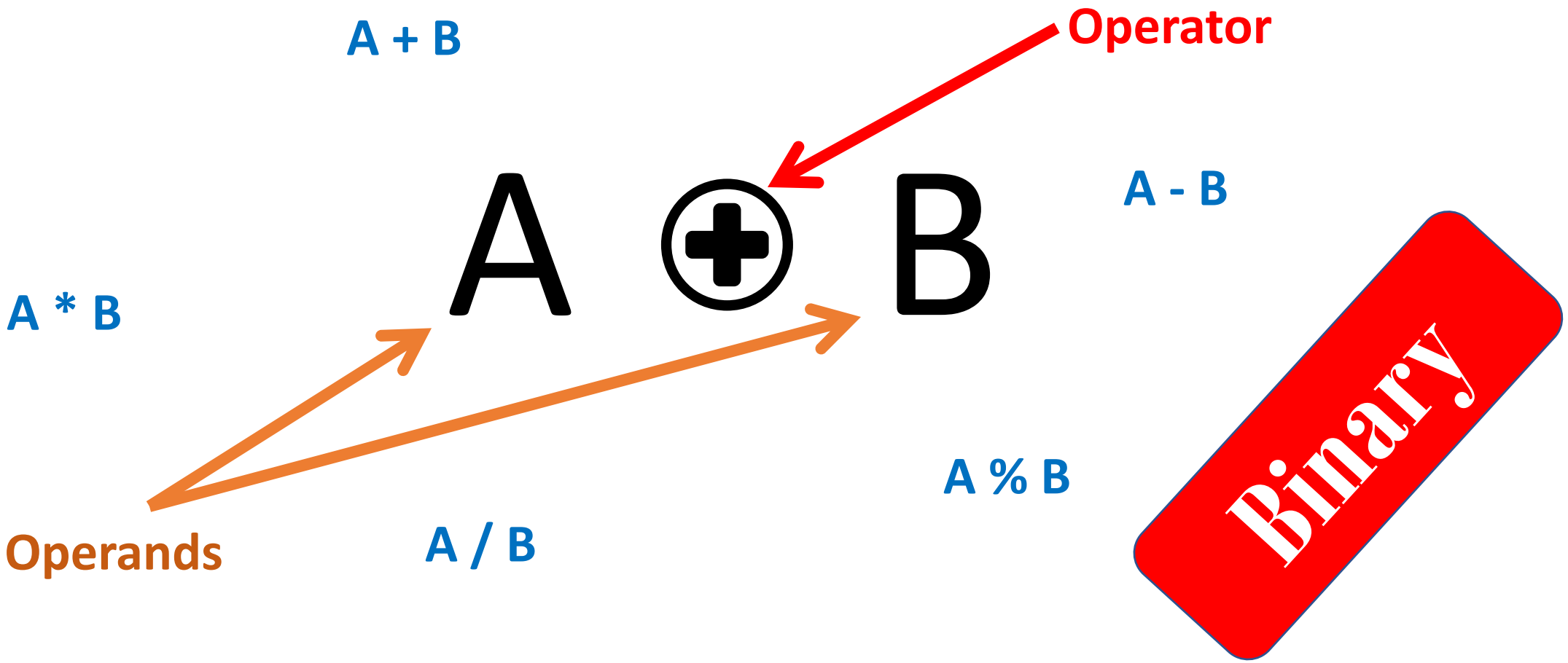
Operators

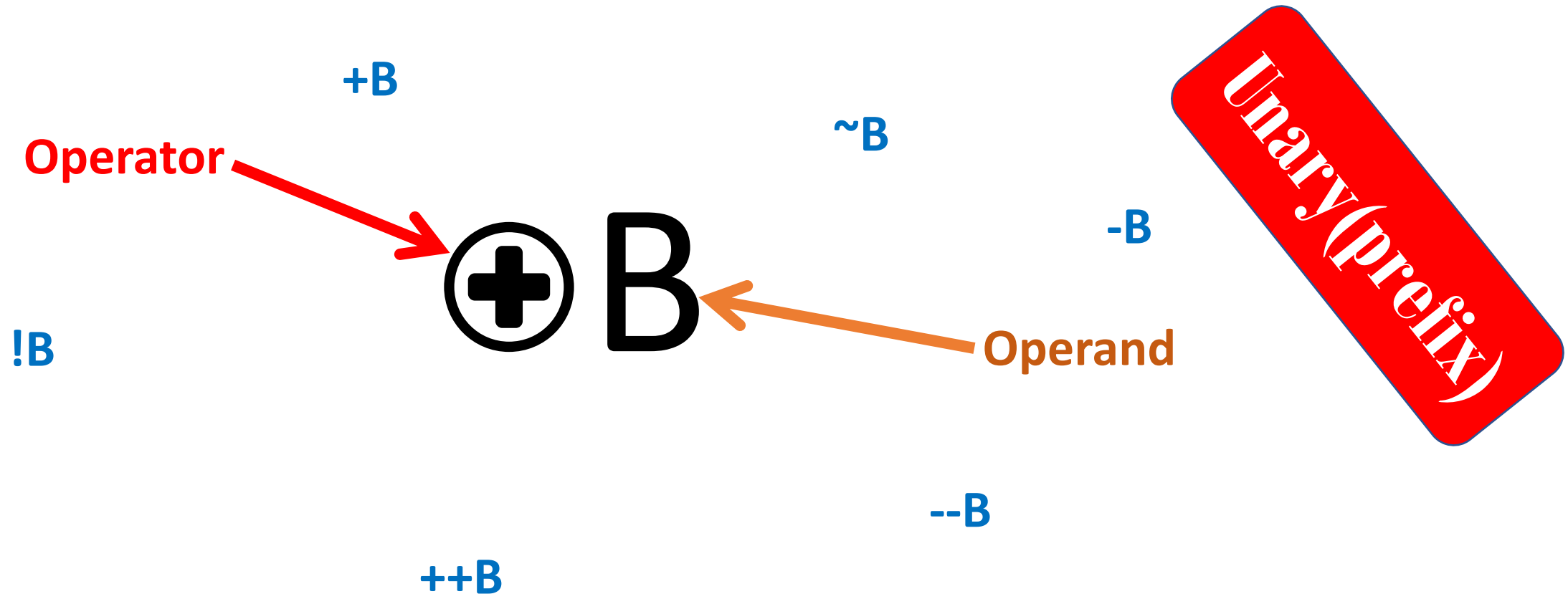
Operator types

- Binary Operators
- Unary Operators
  - Prefix
  - Post fix
- Indexing operator
- Type conversion operator

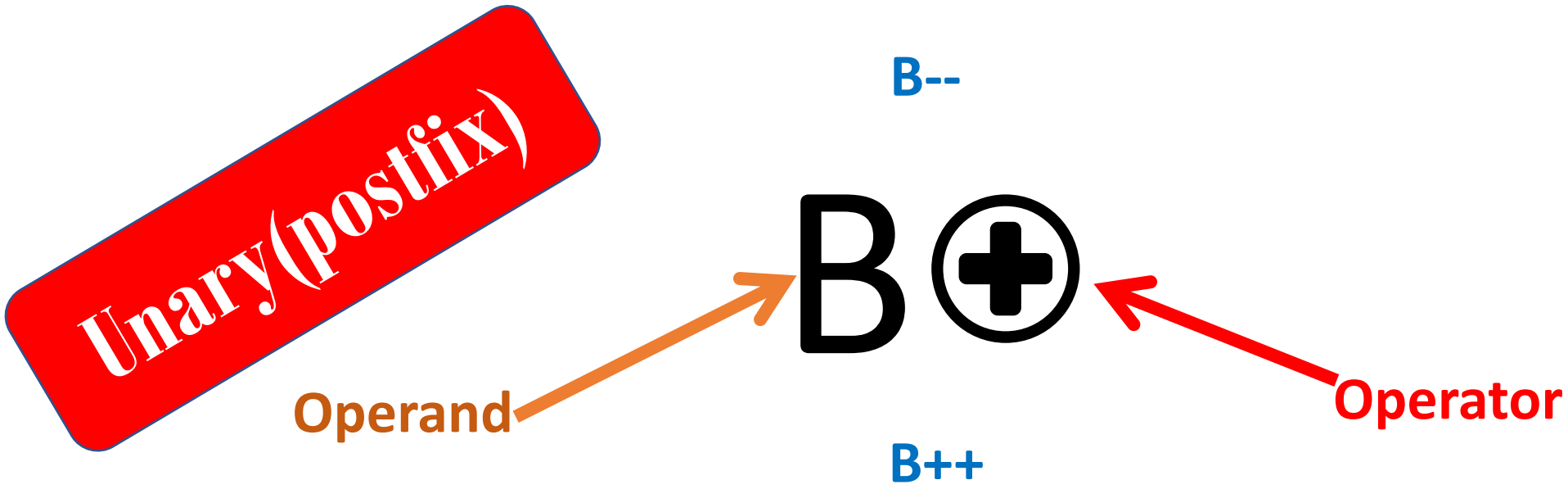
Operators and Operands



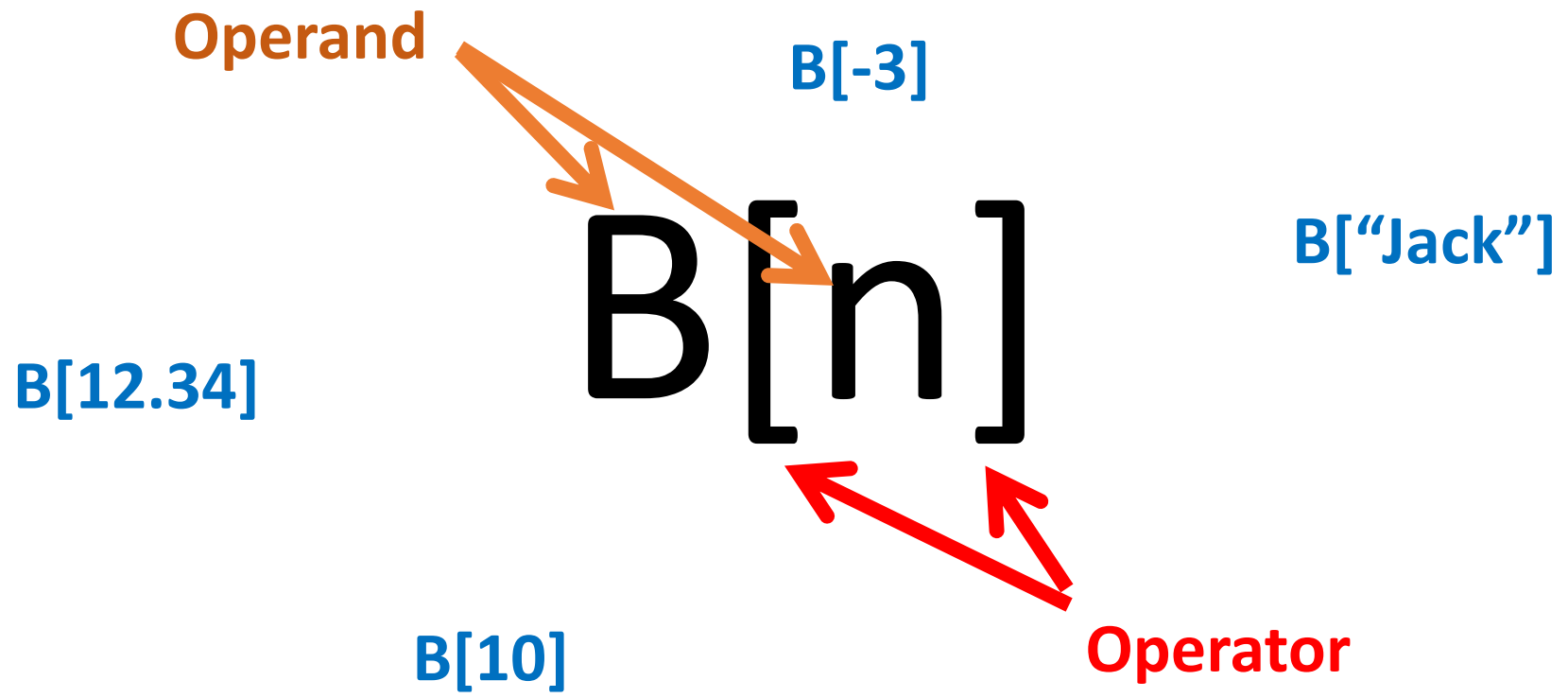
Operators and Operands



Operators and Operands

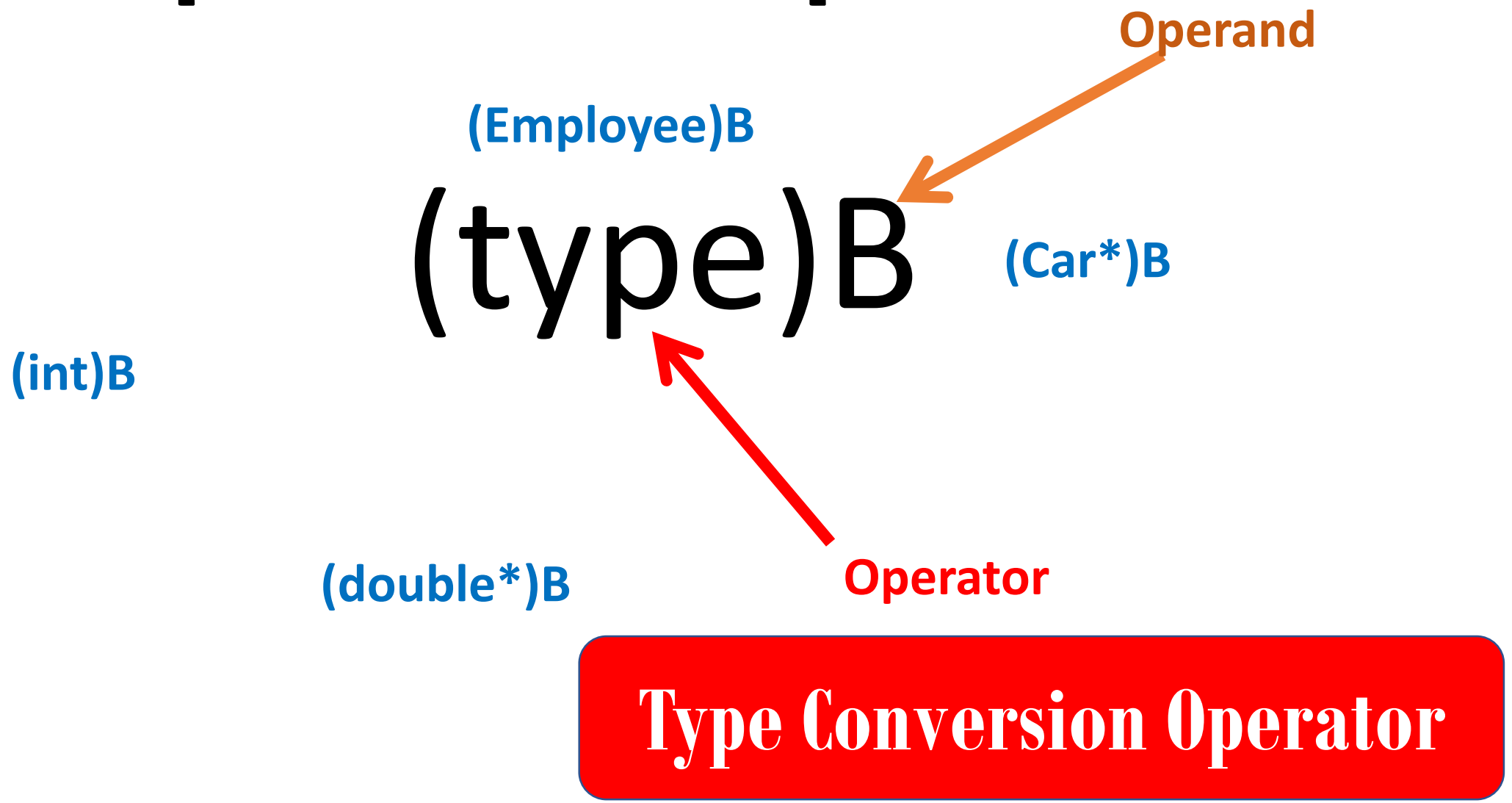


Operators and Operands



Indexing Operator

Operators and Operands



Reminder: What is overloading?

To give

an already existing function

a new meaning!

by reimplementing it;

using its name and changing its arguments!

What is operator overloading?

To give

an already existing operator

a new meaning!

by reimplementing it;

using its name and changing its arguments!

What is operator overloading?

To give

an already existing operator

a new meaning!

by reimplementing it;

using its name and changing its arguments!

Reimplement an operator?

How do I do that?

C++

created

function representations

for all types of operators.

Two types of function representation for Operators

1. **Member operators** **(preferred)**

   The operator functions are member functions of classes involved with the operation. Your should always try to use this first

2. **Non-member or standalone operators**

   (these are also called helper functions or helper operators)

   The operator functions are NOT member functions. They are standalone functions that accept operands as arguments.
   You should not use these unless you have to.

Two types of function representation for Operators

1. **Member operators** **(preferred)**

   The operator functions are member functions of classes involved with the operation. Your should always try to use this first

2. **Non-member or standalone operators**

   (these are also called helper functions or helper operators)

   The operator functions are NOT member functions. They are standalone functions that accept operands as arguments.
   You should not use these unless you have to.

Member operators Implementation

A ⊕ B


```
Foo A;
Faa B;
Whatever Foo::operator⊕(Faa B);
// *this is A (With side-effect, A will change)
A += B
Whatever Foo::operator⊕(Faa B)const;
// *this is A (Without side-effect, A is read only)
A + B
```

Member operators Implementation

⊕B

```
Faa B;
Whatever Faa::operator⊕()const;
//*this is B (Without side-effect, B is read only )
```

!B

```
Whatever Faa::operator⊕();
//*this is B
(With side-effect, B will change; very unusual )
```

No example

Member operators Implementation

B ⊕

```
Faa B;
Whatever Faa::operator⊕(int)const;
//*this is B (Without side-effect, B is read only )
```

!B

```
Whatever Faa::operator⊕(int);
//*this is B
(With side-effect, B will change; very unusual )
```

No example

Index operator Implementation

B[n]

```
Faa B;
Whatever Faa::operator[] (int n)const;
//*this is B (Without side-effect, B is read only )
B[5]
Whatever Faa::operator[] (int n);
//*this is B
(With side-effect, B will change )
B[5]
```

Indexing Operator

Type Conversion operator Implementation

(Foo)B

```
Faa B;
Faa::operator Foo()const;
//*this is B (Without side-effect, B is read only )
B[5]
Faa::operator Foo();
//*this is B
(With side-effect, B will change )
B[5]
```

Type Conversion Operator