const saga

```
int x=2;
int& a = x;   // ALIAS
```

References

A reference variable is an alias, that is, another name for an already existing variable. Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.

References vs Pointers

References are often confused with pointers but three major differences between references and pointers are –

- You cannot have NULL references. You must always be able to assume that a reference is connected to a legitimate piece of storage.
- Once a reference is initialized to an object, it cannot be changed to refer to another object. Pointers can be pointed to another object at any time.
- A reference must be initialized when it is created. Pointers can be initialized at any time.

References

Creating References in C++

Think of a variable name as a label attached to the variable's location in memory. You can then think of a reference as a second label attached to that memory location. Therefore, you can access the contents of the variable through either the original variable name or the reference. For example, suppose we have the following example –

```
int i = 17;
```

We can declare reference variables for i as follows.

```
int& r = i;
```

Read the & in these declarations as **reference**. Thus, read the first declaration as "r is an integer reference initialized to i" and read the second declaration as "s is a double reference initialized to d.". Following example makes use of references on int and double –

References

```
// primitive variables
int i;
double d;
// reference variables
int& r = i;
double& s = d;
i = 5;
cout << "Value of i : " << i << endl; // 5
cout << "Value of i reference : " << r << endl; // 5
d = 6.7;
cout << "Value of d : " << d << endl; // 6.7
cout << "Value of d reference : " << s << endl; // 6.7
```

Passing Parameters as References

```
// function declaration: reference parameters
void swap(int& x, int& y);

int main () {
  // local variable declaration
  int a = 1; int b = 2;
  cout << "Before swap, value of a :" << a << endl; // 1
  cout << "Before swap, value of b :" << b << endl; // 2
  // calling a function to swap the values
  swap(a, b);
  cout << "After swap, value of a :" << a << endl;  // 2
  cout << "After swap, value of b :" << b << endl;  // 1
}

void swap(int& x, int& y) {
  int temp = x;
  x = y;
  y = temp
}
```

Returning References

A C++ program can be made easier to read and maintain by using references rather than pointers. A C++ function can return a reference in a similar way as it returns a pointer.

When a function returns a reference, it returns an implicit pointer to its return value. This way, a function can be used on the left side of an assignment statement. For

Returning References

```
int v[] = {3, 2, 6, 8, 5};

int& setValue(int i) {
  return v[i]; // return a reference to i-th element
}

int main () {
 for (int i = 0; i < 5; i++ )
   cout << v[i] << " "; // 3 2 6 8 5
 cout << endl;
 setValue(1) = 9; setValue(3) = 7;
 for (int i = 0; i < 5; i++ )
   cout << v[i] << " "; // 3 9 6 9 5
}
```

Returning References

When returning a reference, be careful that the object being referred to does not go out of scope. So it is not legal to return a reference to local var.

```
int& fun(int& a) {
  int q;
  //! return q; // Compile time error
  return a;     // Safe, a is live outside this scope
}
```

Back to const saga

const saga

```
int x=2;
int& a = x;  // ALIAS
int& a1 = 2; // ILLEGALE
a=5;
int y=3;
a=y;         // LEGALE, r-valore di y assegnato a
             // l-valore di x
```

```
int x=2;
int* p = &x;
*p=5;
int y=3;
p=&y;        // LEGALE
```

const saga

```
int x=2;
int * const p = &x;
*p=5;    // LEGALE
int y=3;
p=&y;    // ILLEGALE
```

```
int x=2;
int & const r = x; // ILLEGALE (tipo illegale)
```

const saga

```
int x=2;
const int* p = &x;
*p=5;    // ILLEGALE
int y=3;
p=&y;    // LEGALE
```

```
int x=2;
const int *const p = &x;
*p=5;    // ILLEGALE
int y=3;
p=&y;    // ILLEGALE
```

const saga

```
int x=2;
const int& r = x; // RIFERIMENTO A TIPO COSTANTE
r=5;              // ILLEGALE
int y=3;
r=y;              // ILLEGALE
```

```
const int& r = 4; // LEGALE
r=5;              // ILLEGALE
int y=3;
r=y;              // ILLEGALE
```

```
const int & const r=2; // ILLEGALE (tipo illegale)
```

const saga

```
void fun(const int& r);
// PASSAGGIO PER RIFERIMENTO (A TIPO) COSTANTE
int x=2;
fun(x); // LEGALE
fun(4); // LEGALE ←
```

```
const int& fun() { return 4; /* LEGALE */ }
// RITORNA RIFERIMENTO (A TIPO) COSTANTE

fun()=5;     // ILLEGALE
```

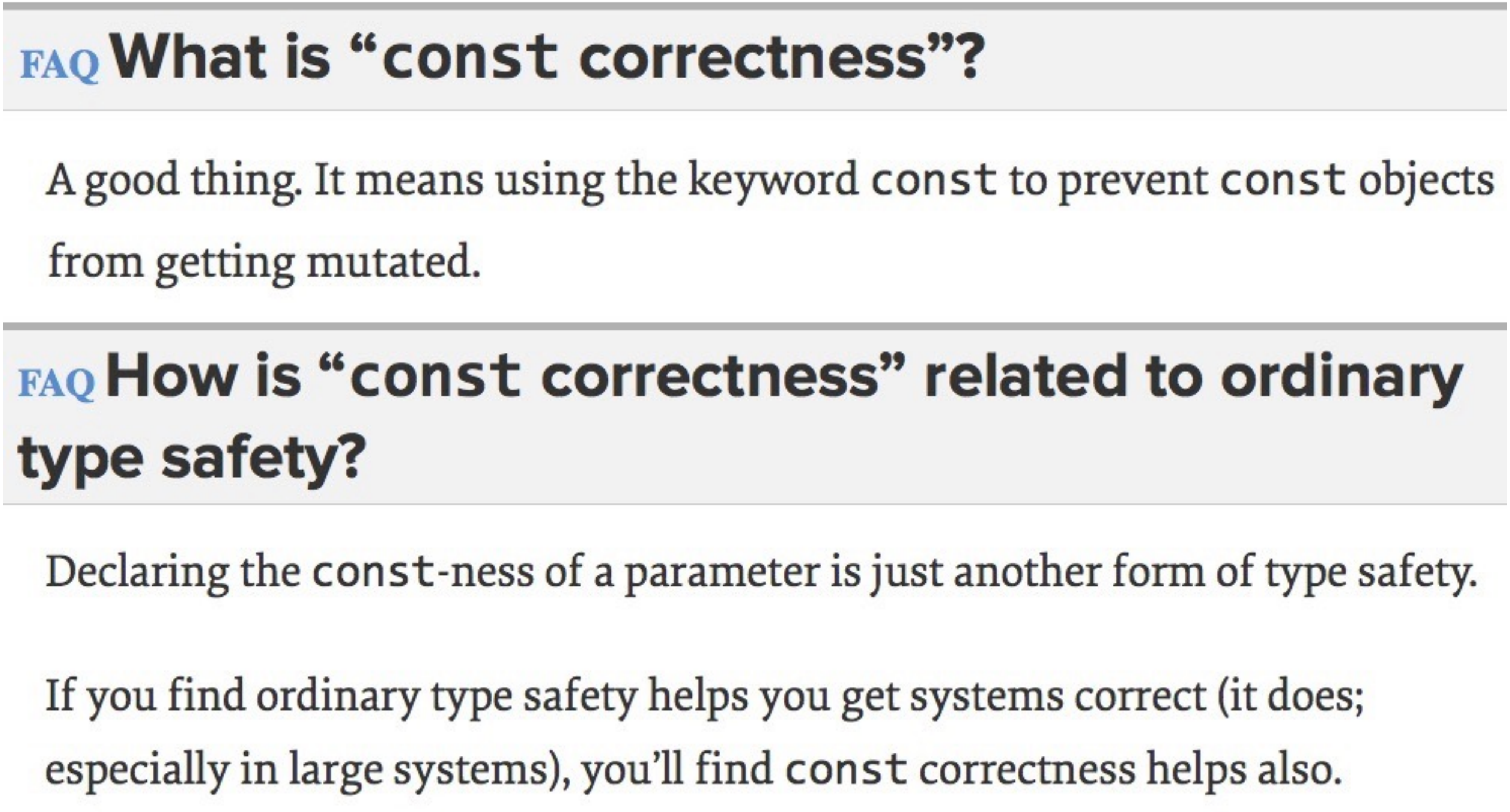
const saga

```
void fun_ref(const int& r);
// VERSUS
void fun_ptr(const int* p);
int x=2;
fun_ref(x);
// VERSUS
fun_ptr(&x);

fun_ref(4);  // LEGALE
// VERSUS
fun_ptr(&4); // ILLEGALE
```

const saga

```
void fun1(const Big& r); // PER RIFERIMENTO COSTANTE
// VERSUS
void fun2(Big v);        // PER VALORE

Big b(...);
fun1(b);     // copia di un riferimento a Big
// VERSUS
fun2(b);     // costruttore di copia di Big
```

const saga



FAQ What is "const correctness"?

A good thing. It means using the keyword `const` to prevent `const` objects from getting mutated.

FAQ How is "const correctness" related to ordinary type safety?

Declaring the `const`-ness of a parameter is just another form of type safety.

If you find ordinary type safety helps you get systems correct (it does; especially in large systems), you'll find `const` correctness helps also.

const saga

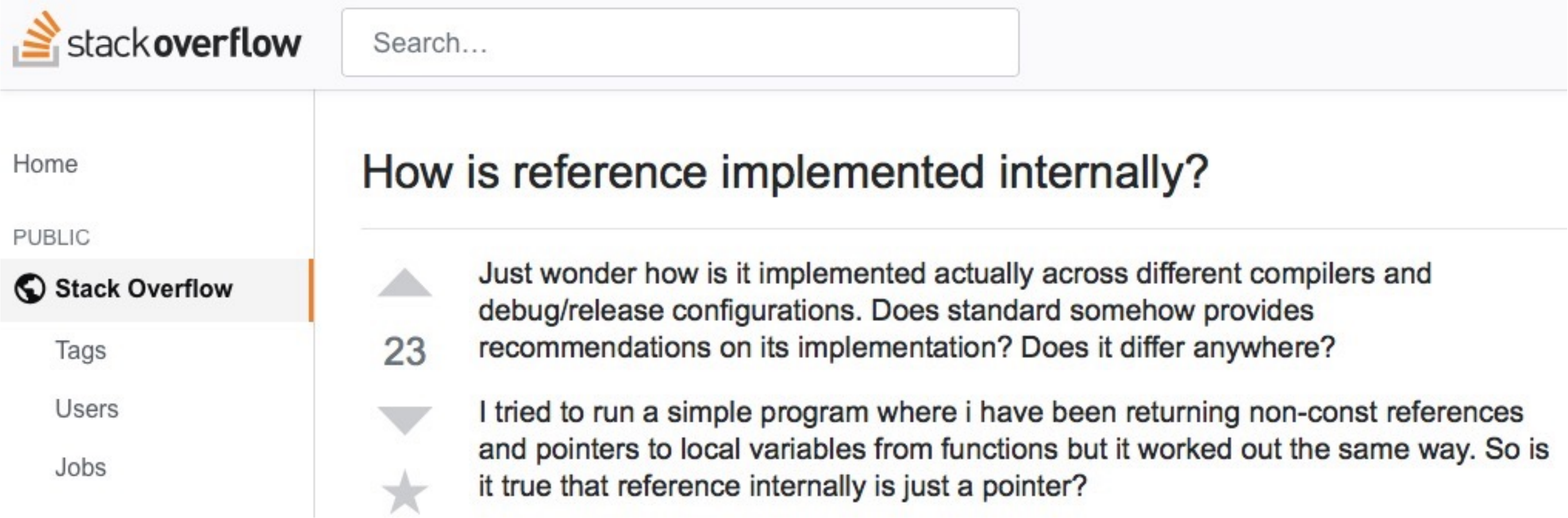
Come sono implementati i reference?

Lo standard C++ non lo prevede, dipende quindi dal compilatore.

In pratica: (quasi sempre) mediante puntatori.

const saga

Come sono implementati i reference?



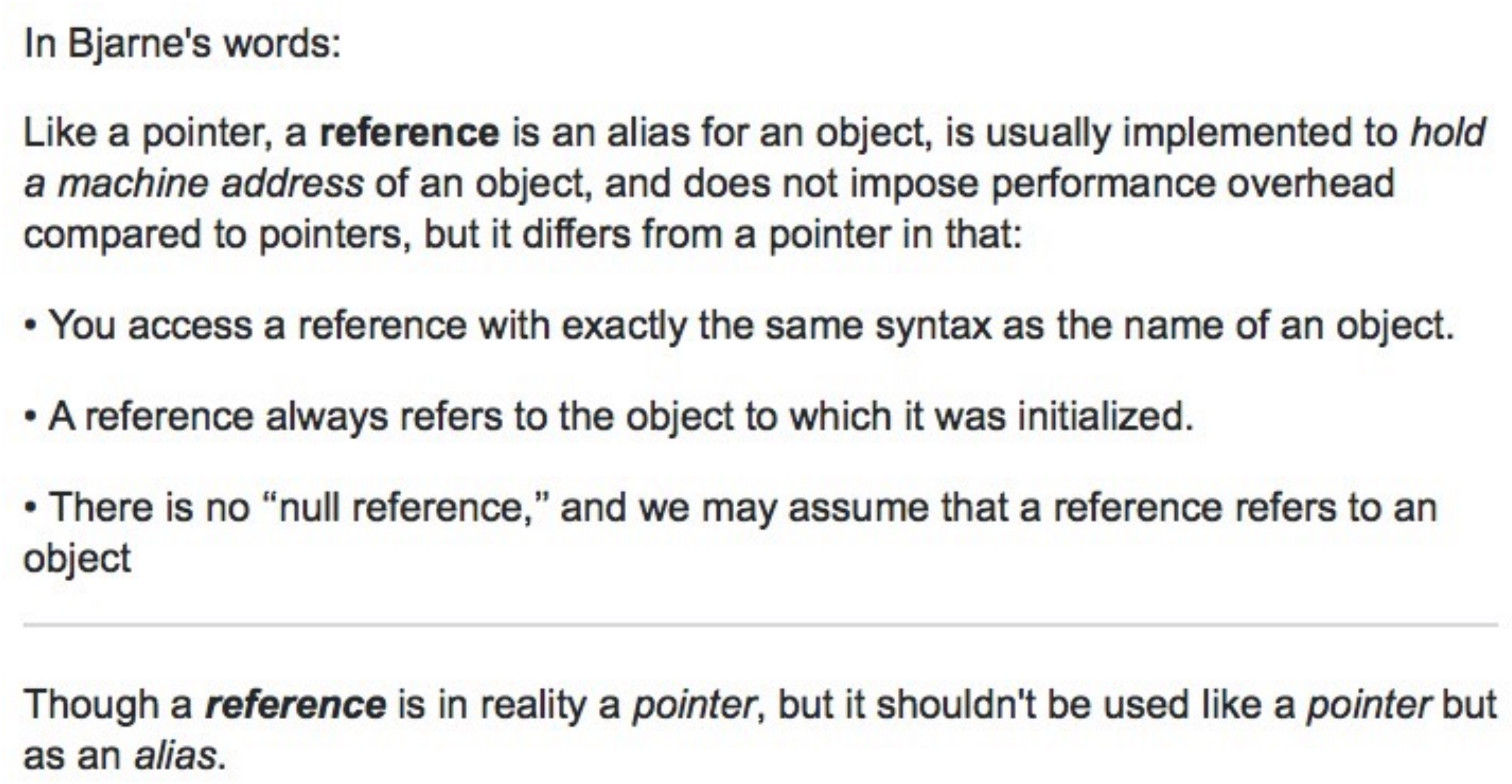
How is reference implemented internally?

Just wonder how is it implemented actually across different compilers and debug/release configurations. Does standard somehow provides recommendations on its implementation? Does it differ anywhere?

I tried to run a simple program where i have been returning non-const references and pointers to local variables from functions but it worked out the same way. So is it true that reference internally is just a pointer?

const saga

Come sono implementati i reference?



In Bjarne's words:

Like a pointer, a **reference** is an alias for an object, is usually implemented to *hold a machine address* of an object, and does not impose performance overhead compared to pointers, but it differs from a pointer in that:

- You access a reference with exactly the same syntax as the name of an object.

- A reference always refers to the object to which it was initialized.

- There is no "null reference," and we may assume that a reference refers to an object

---

Though a ***reference*** is in reality a *pointer*, but it shouldn't be used like a *pointer* but as an *alias*.

Warning

```
const int& f() {return 4;}
int main() { f(); }

g++ ex.cpp
warning: returning reference to temporary [-Wreturn-local-addr]
 const int& f() {return 4;}
                        ^

clang ex.cpp
warning: returning reference to local temporary object [-Wreturn-stack-address]
const int& f() {return 4;}
                       ^
```

Parametro per valore VS riferimento costante

```
class C {
  int a[1000]; // 4*1000 bytes
};

bool byValue(C x) {return true;}
bool byConstReference(const C& x) {return true;}

int main() {
  C obj;
  for(int i=0; i<10000000; i++) byValue(obj);          // 3.368 sec
  for(int i=0; i<10000000; i++) byConstReference(obj); // 0.031 sec
                                                       // 108x
}
```