This rule prevents multiple `extern` declarations from initializing a variable in different ways.

A variable in an `extern` declaration always has static storage duration. The scope of the variable depends on the declaration's placement. If the declaration is inside a block, the variable has block scope; otherwise, it has file scope:

Q&A

```
                                  static storage duration
extern int i;                     file scope
                                  ? linkage

void f(void)
{
                                  static storage duration
  extern int j;                   block scope
                                  ? linkage
}
```

Determining the linkage of an `extern` variable is a bit harder. If the variable was declared `static` earlier in the file (outside of any function definition), then it has internal linkage. Otherwise (the normal case), the variable has external linkage.

The `register` Storage Class

Using the `register` storage class in the declaration of a variable asks the compiler to store the variable in a register instead of keeping it in main memory like other variables. (A *register* is a storage area located in a computer's CPU. Data stored in a register can be accessed and updated faster than data stored in ordinary memory.) Specifying the storage class of a variable to be `register` is a request, not a command. The compiler is free to store a `register` variable in memory if it chooses.

The `register` storage class is legal only for variables declared in a block. A `register` variable has the same storage duration, scope, and linkage as an `auto` variable. However, a `register` variable lacks one property that an `auto` variable has: since registers don't have addresses, it's illegal to use the `&` operator to take the address of a `register` variable. This restriction applies even if the compiler has elected to store the variable in memory.

`register` is best used for variables that are accessed and/or updated frequently. For example, the loop control variable in a `for` statement is a good candidate for `register` treatment:

```
int sum_array(int a[], int n)
{
  register int i;
  int sum = 0;

  for (i = 0; i < n; i++)
    sum += a[i];
  return sum;
}
```