Race Conditions

Joseph Hallett

bristol.ac.uk

What are they?

Computers can do more than one thing at once

Sometimes the order things get done in can lead to bugs…

Sometimes these bugs can have security related consequences…

Simple Example

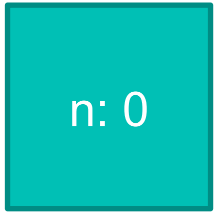
Here a really simple increment function

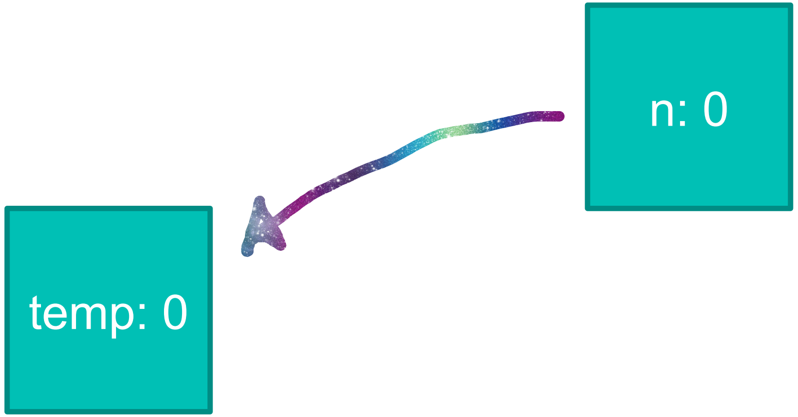
```
void increment(int *n) {
        int temp;

        temp = *n;
        temp += 1;
        *n = temp;
}
```

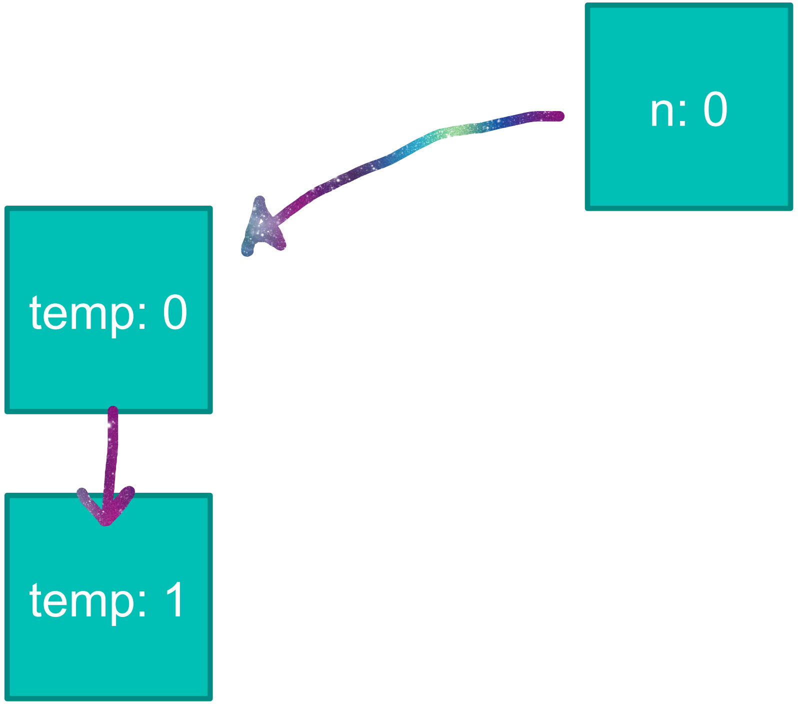
It is not thread safe though!

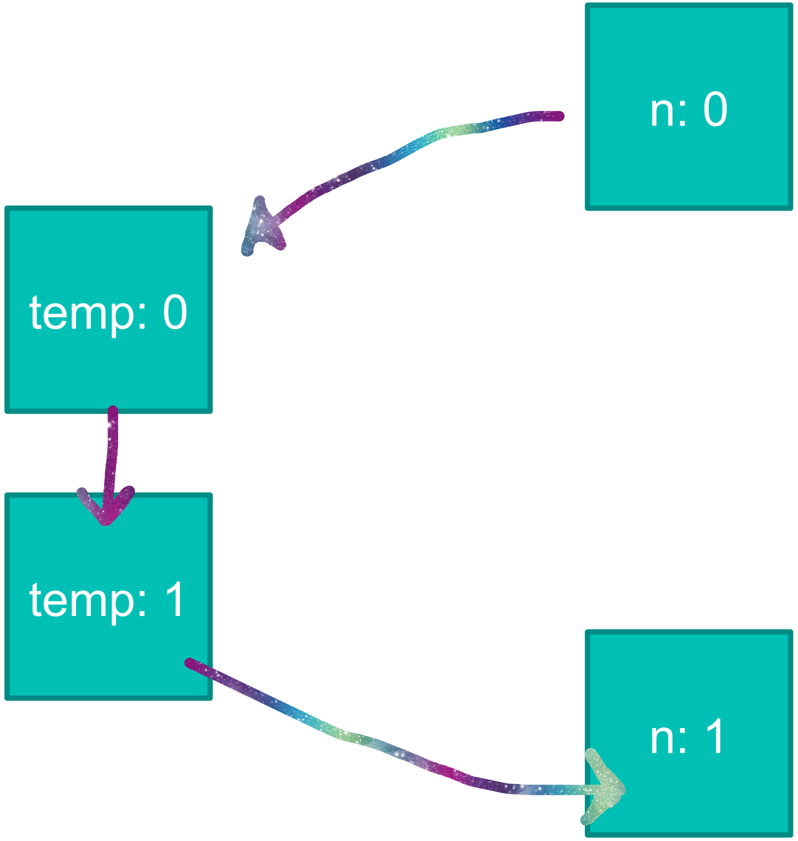
    If we are not careful, we can lose increments…

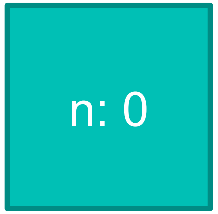
bristol.ac.uk

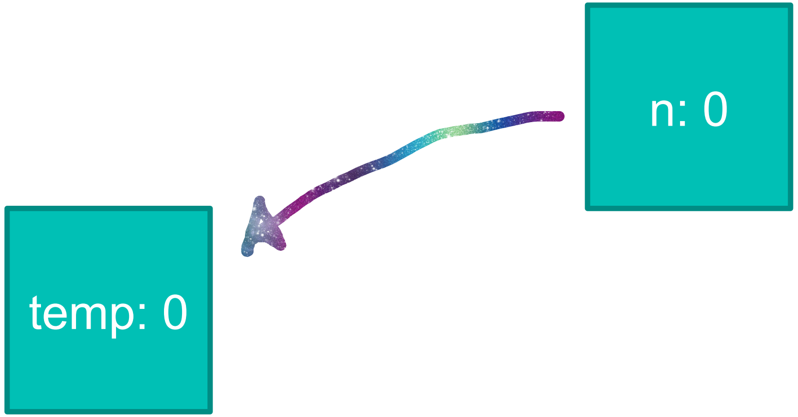

n: 0

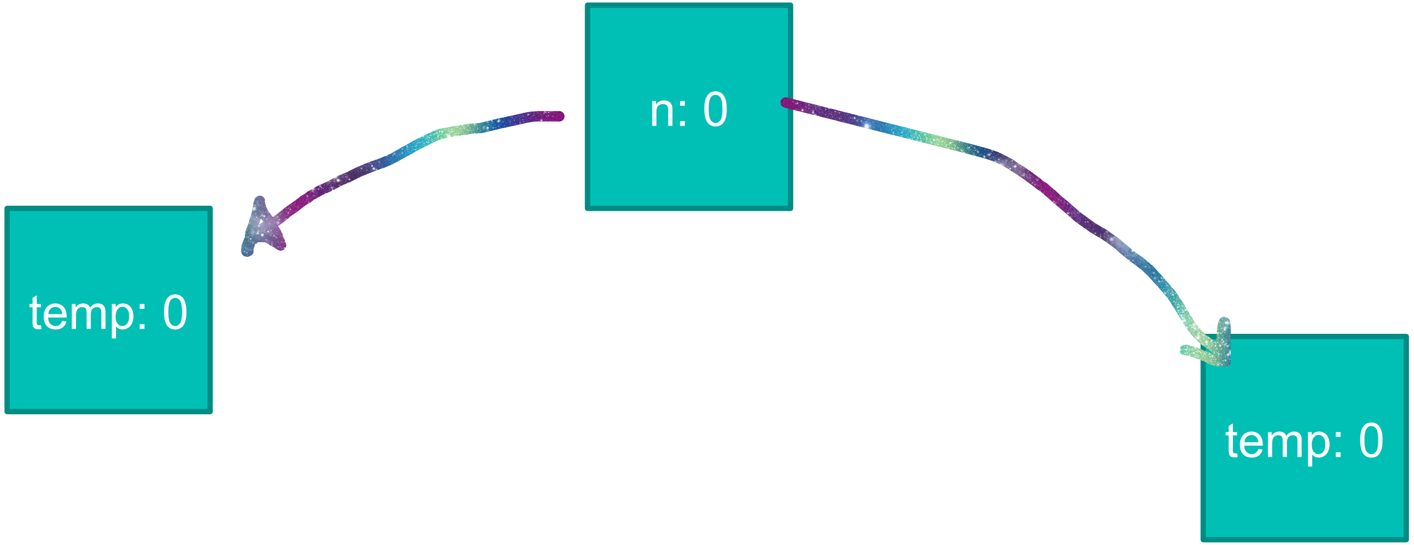

n: 0
temp: 0

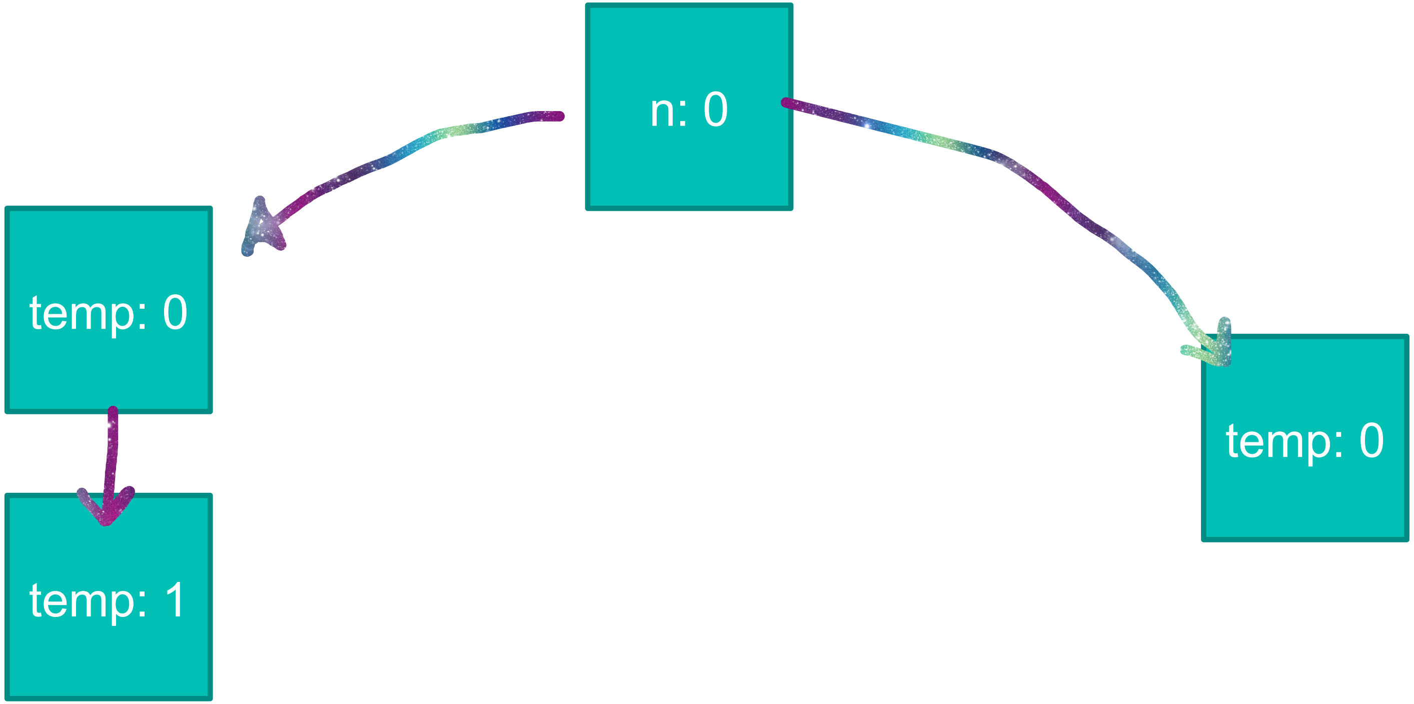

n: 0
temp: 0
temp: 1


n: 0
temp: 0
temp: 1
n: 1

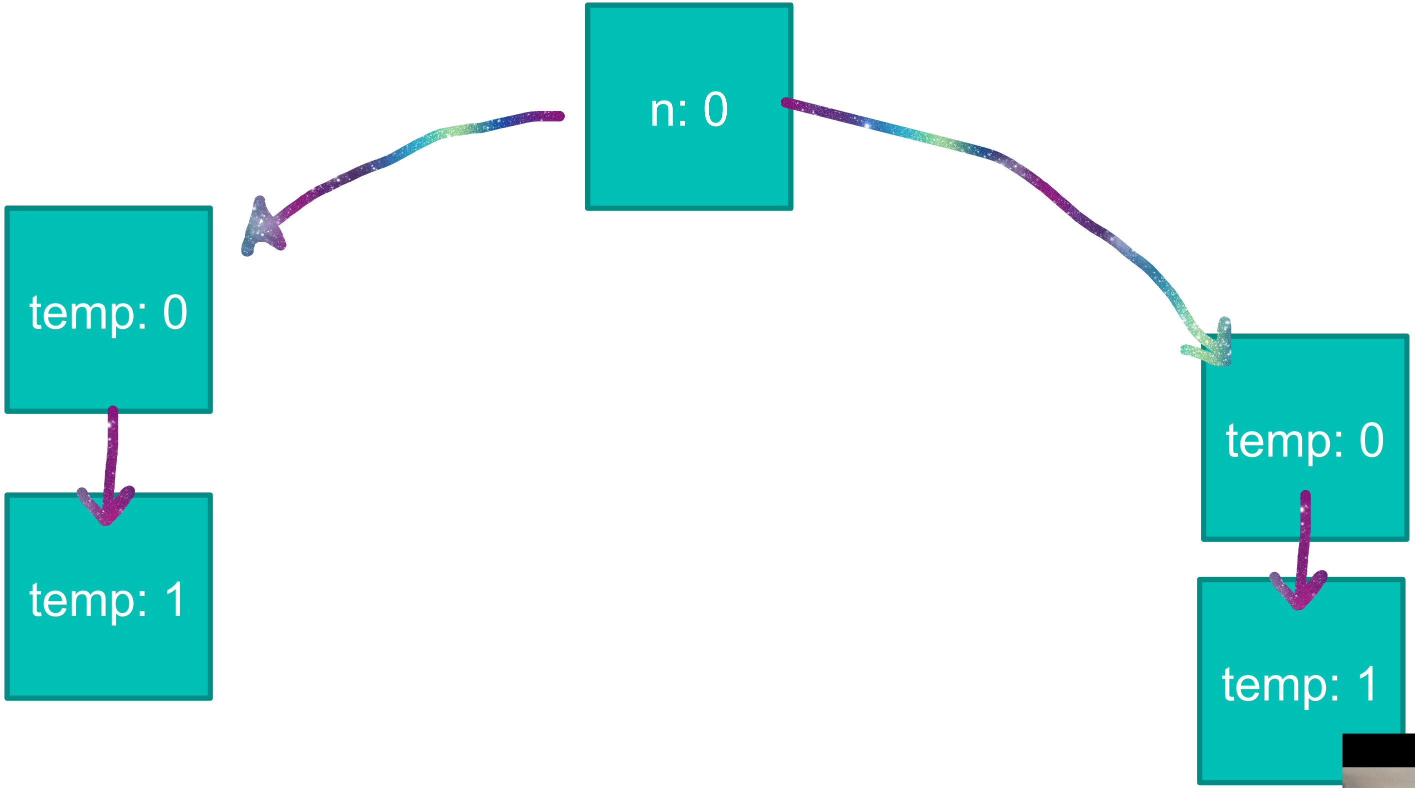

n: 0

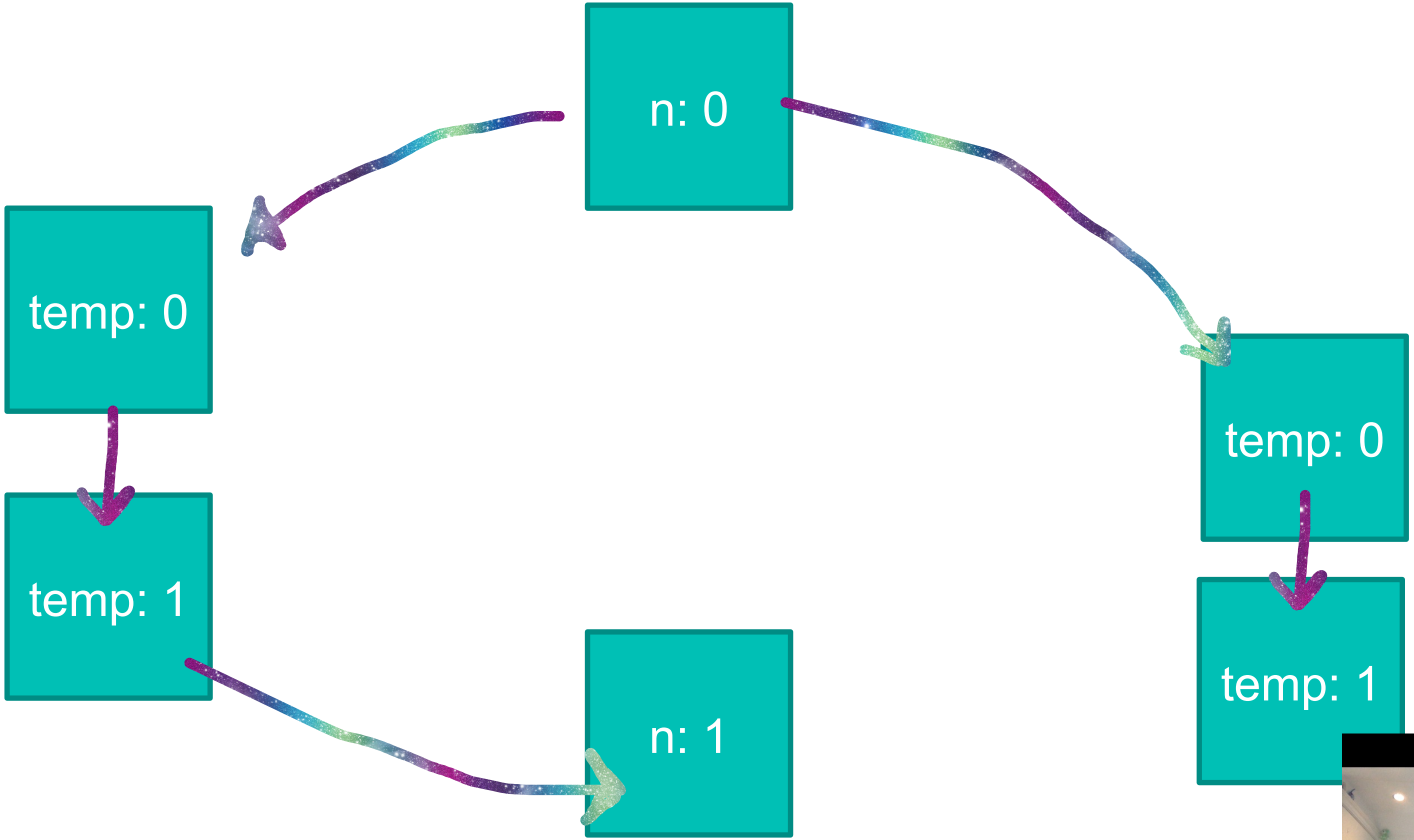

n: 0
temp: 0

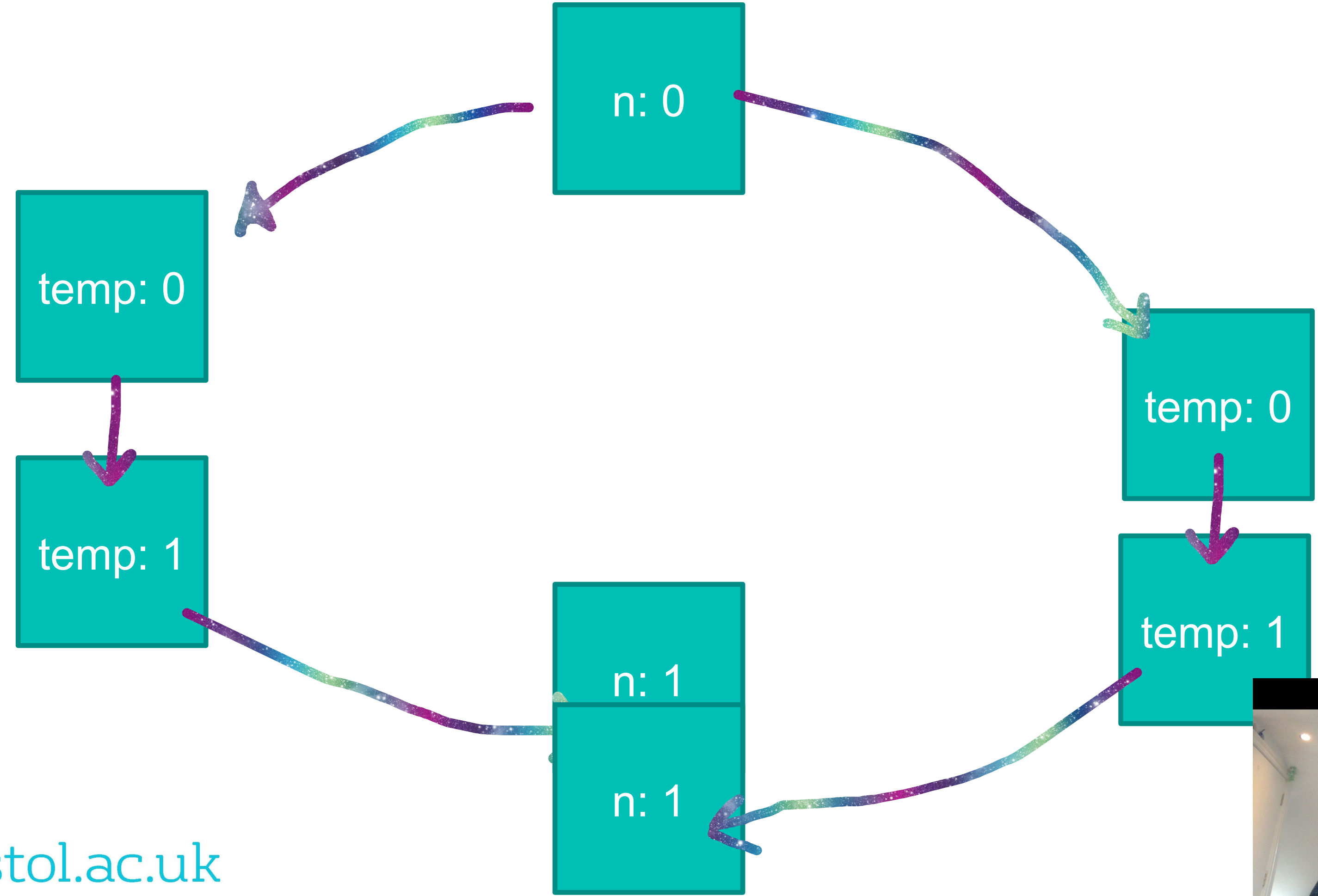

n: 0
temp: 0
temp: 0



bristol.ac.uk


n: 0
temp: 0
temp: 1
temp: 0
temp: 1


n: 0
temp: 0
temp: 1
n: 1
temp: 0
temp: 1


n: 0
temp: 0
temp: 1
temp: 0
temp: 1
n: 1
n: 1

Simple Example

Oh no we've called increment twice but only incremented once

- Correctness issue
- Well understood

But what's the security issue?

Access and Open

The **access**() system call checks the accessibility of the file named by the path argument for the access permissions indicated by the mode argument. The value of mode is either the bitwise-inclusive OR of the access permissions to be checked (R_OK for read permission, W_OK for write permission, and X_OK for execute/search permission), or the existence test (F_OK).

Suppose we have a suid-program that does controlled writes as root. It checks using access if your *real* user can write to a file, then does the writing as root

Access and Open

```
if (access("/tmp/X", W_OK)) {
        f = open("/tmp/X");
        write_to_file(f);
} else {
        printf("Nope.");
}
```

Access and Open

start $\longrightarrow$

```
if (access("/tmp/X", W_OK)) {
        f = open("/tmp/X");
        write_to_file(f);
} else {
        printf("Nope.");
}
```

Access and Open

check succeeds $\longrightarrow$

```
if (access("/tmp/X", W_OK)) {
        f = open("/tmp/X");
        write_to_file(f);
} else {
        printf("Nope.");
}
```

bristol.ac.uk

Access and Open

check succeeds $\longrightarrow$

```
if (access("/tmp/X", W_OK)) {
        f = open("/tmp/X");
        write_to_file(f);
} else {
        printf("Nope.");
}
```

```
$ rm /tmp/x
$ ln -s /etc/passwd /tmp/x
```

Access and Open

/tmp/x is a different file
… but were still on this

```
if (access("/tmp/X", W_OK)) {
        f = open("/tmp/X");
        write_to_file(f);
} else {
        printf("Nope.");
}
```

```
$ rm /tmp/x
$ ln -s /etc/passwd /tmp/x
```

man 2 access (bottom)

```
SECURITY CONSIDERATIONS
     The result of access() should not be used to make an actual access control decision,
     since its response, even if correct at the moment it is formed, may be outdated at
     the time you act on it.  access() results should only be used to pre-flight, such as
     when configuring user interface elements or for optimization purposes.  The actual
     access control decision should be made by attempting to execute the relevant system
     call while holding the applicable credentials, and properly handling any resulting
     errors; and this must be done even though access() may have predicted success.

     Additionally, set-user-ID and set-group-ID applications should restore the effective
     user or group ID, and perform actions directly rather than use access() to simulate
     access checks for the real user or group ID.
```

How do we avoid

Basically, the same as any other race condition…
Use synchronization around time-of-check and time-of use

Be *really* careful…
Some static analysis tools can spot these issues sometimes…

bristol.ac.uk