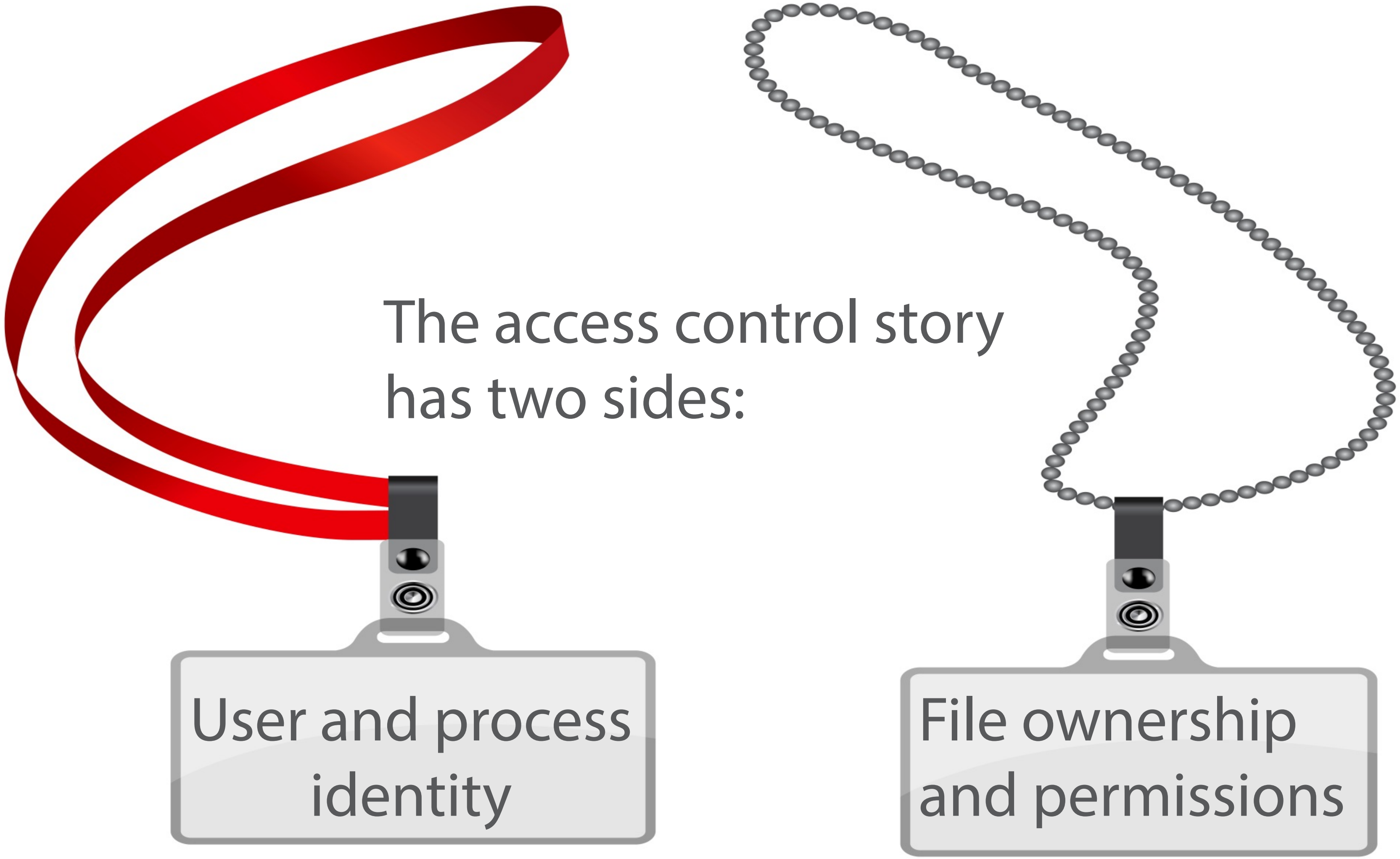
Controlling Access, Identity and Permissions

Chris Brown

In This Module …



The access control story has two sides:

User and process identity

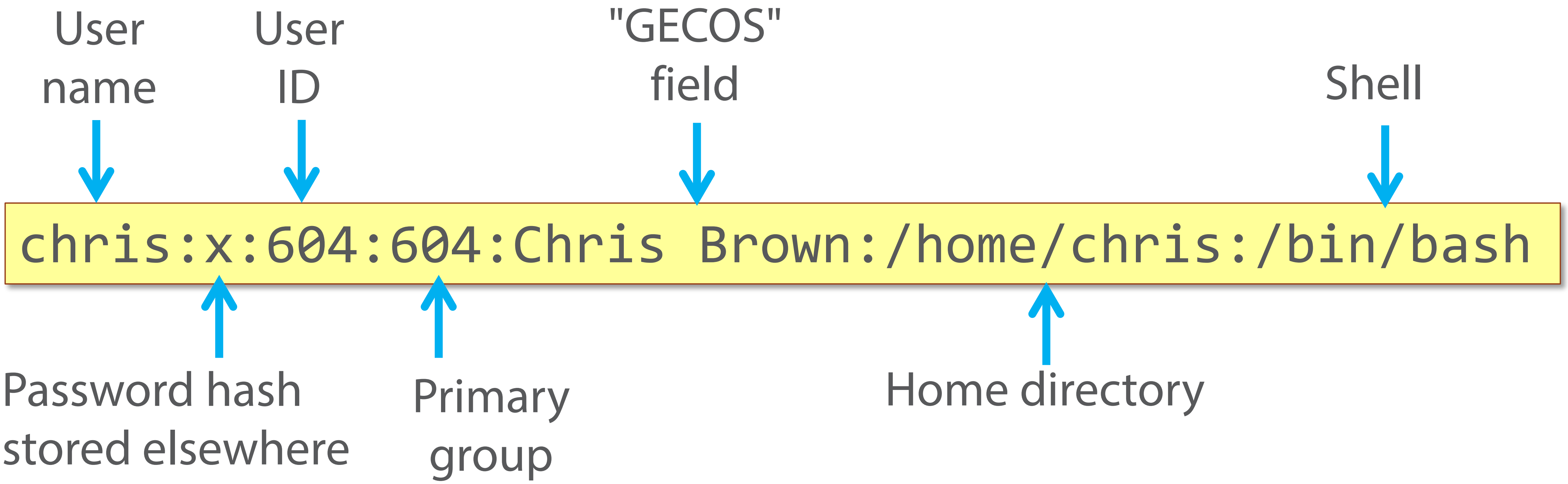
File ownership and permissions

User Identity
Enumerating and
searching user accounts

Process Identity
Getting & setting
Real vs. Effective User ID

File permissions
and ownership

User Identity — the Password File



```
chris:x:604:604:Chris Brown:/home/chris:/bin/bash
```

User name

User ID

"GECOS" field

Shell

Password hash stored elsewhere

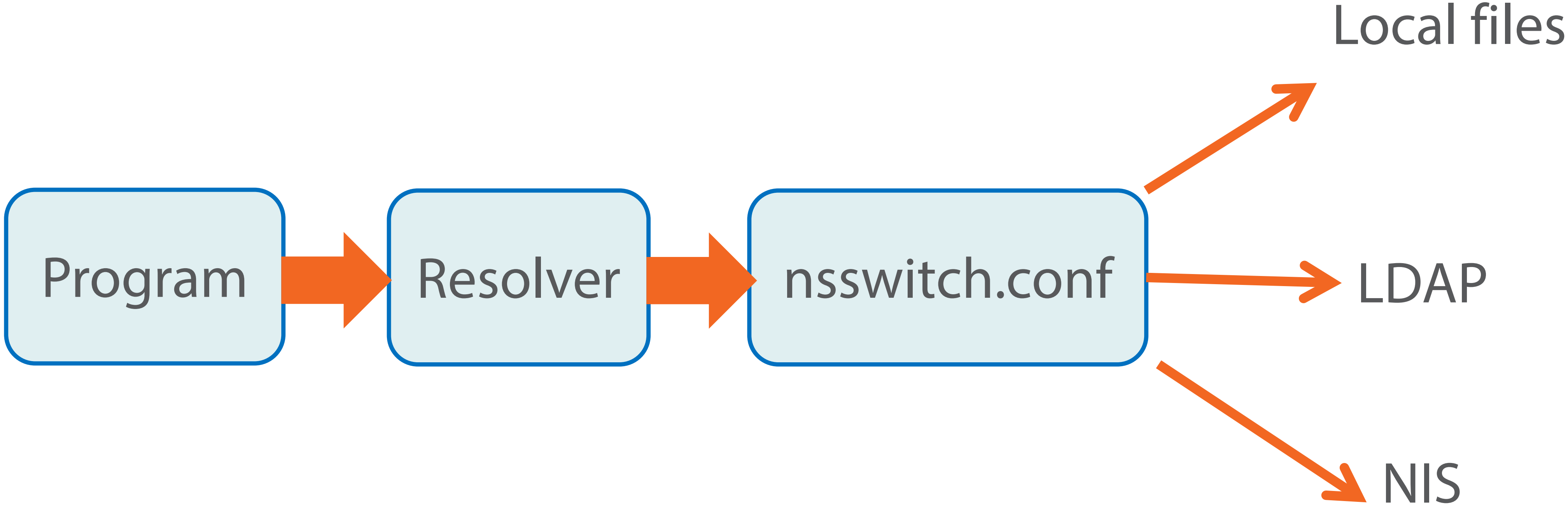
Primary group

Home directory

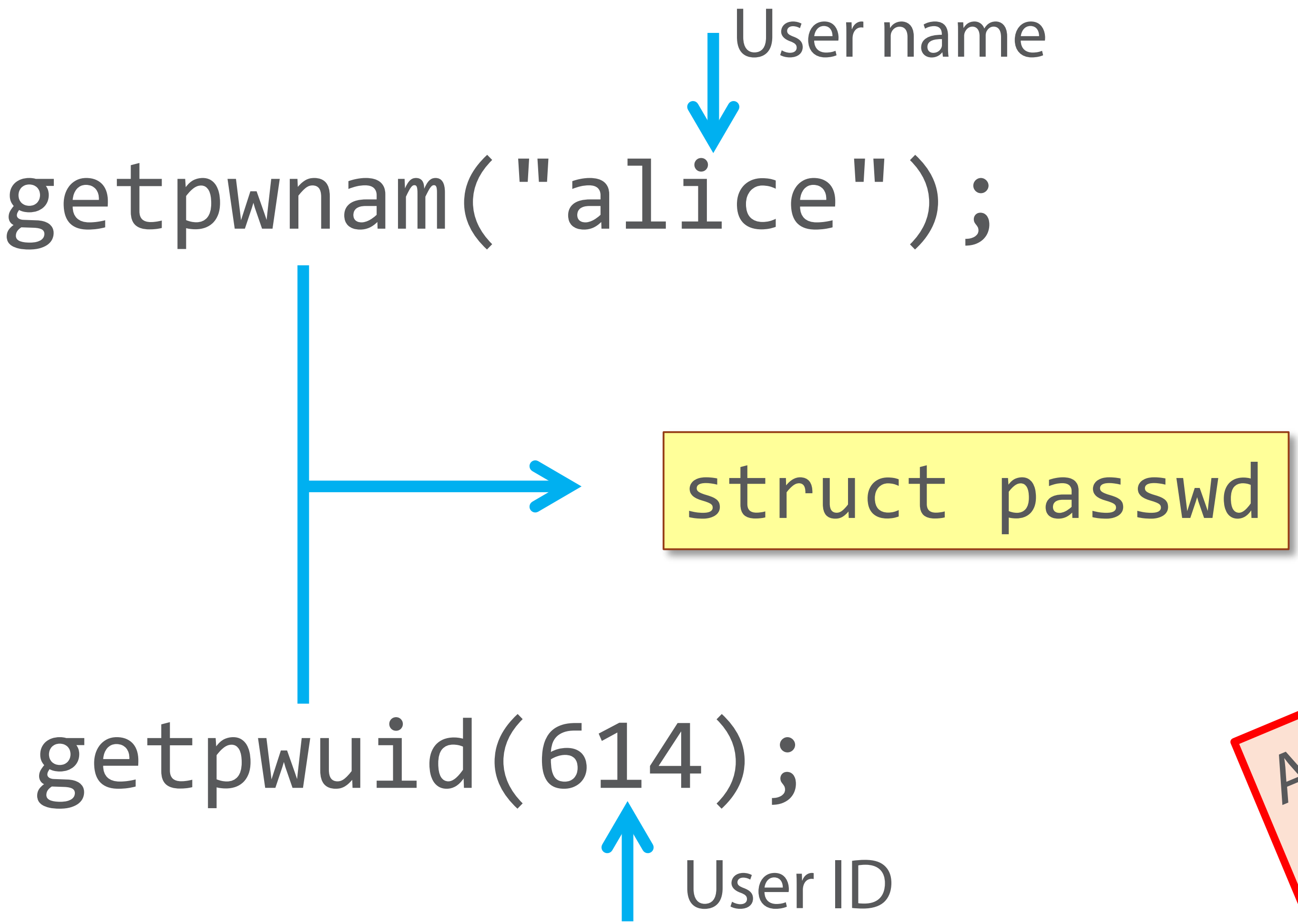
The passwd Structure

```
struct passwd {
  char *pw_name;    /* username */
  char *pw_passwd;  /* user password */
  uid_t pw_uid;     /* user ID */
  gid_t pw_gid;     /* group ID */
  char *pw_gecos;   /* user information */
  char *pw_dir;     /* home directory */
  char *pw_shell;   /* shell program */
};
```

Querying User Identity



Querying User Identity



User name

```
getpwnam("alice");
```

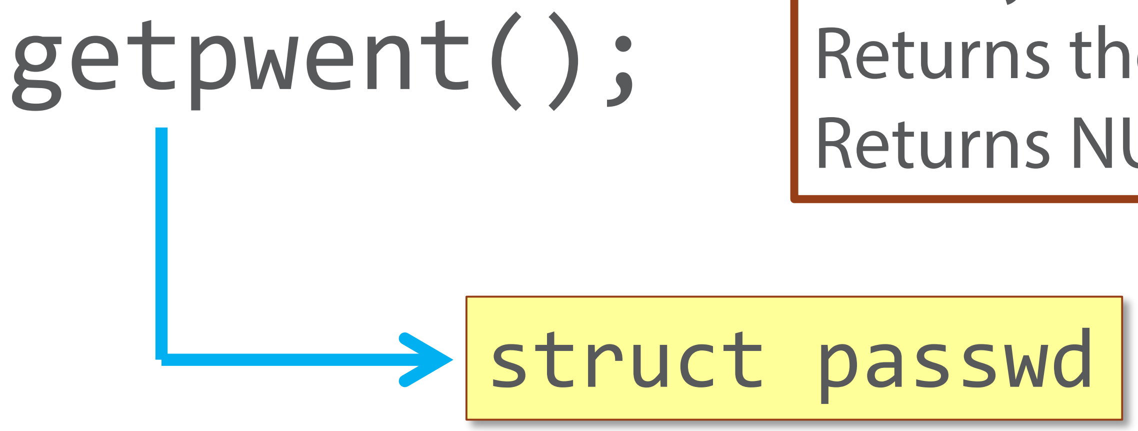
```
struct passwd
```

```
getpwuid(614);
```

User ID

A null pointer is returned if the user name or UID is not found

Enumerating User Accounts



```
getpwent();
```

```
struct passwd
```

Usually used in a loop
Returns the next account from the database
Returns NULL at the end

```
setpwent();
```

"Rewind" to the beginning

Enumerating User Accounts — Example

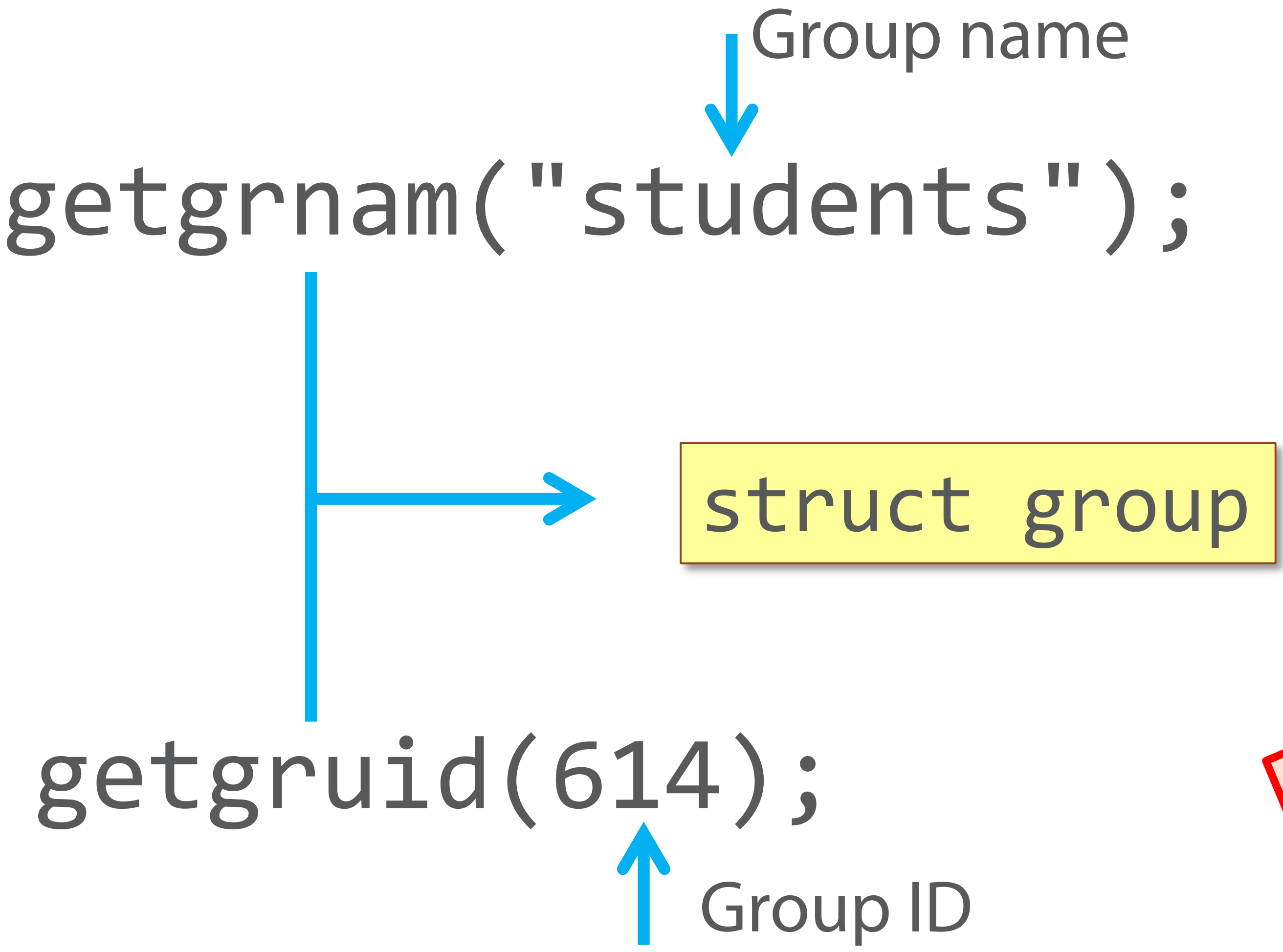
```
/* Lists accounts with uid >= 1000 */

#include <stdio.h>
#include <pwd.h>

void main(int argc, char *argv[])
{
  struct passwd *u;

  while ((u = getpwent()) != NULL) {
    if (u->pw_uid >= 1000)
      printf("%s\n", u->pw_name);
  }
}
```

Querying Groups



Group name

```
getgrnam("students");
```

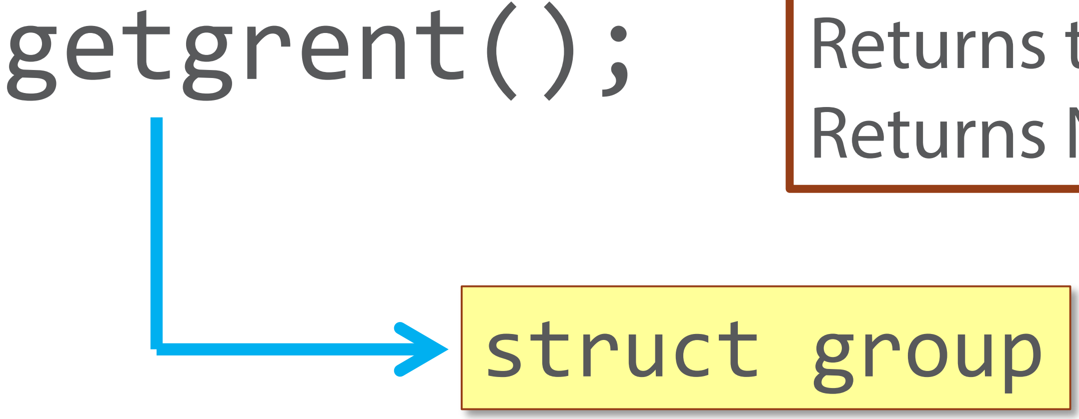
```
struct group
```

```
getgruid(614);
```

Group ID

A null pointer is returned if the group name or UID is not found

Enumerating Groups



```
getgrent();
```

```
struct group
```

Usually used in a loop
Returns the next group from the database
Returns NULL at the end

```
setgrent();
```

"Rewind" to the beginning

The group structure

```
struct group {
  char *gr_name;    /* group name */
  char *gr_passwd; /* group password */
  gid_t gr_gid;     /* group ID */
  char **gr_mem;    /* group members */ };
```

Process Identity

Real User ID

Effective User ID

Process Identity

A process inherits its "real" user identity across a `fork()` and an `exec()` but ...

If the program being exec'd has the `setuid` bit turned on, it runs with the effective ID of its owner

Access permission checks are made against the *effective ID*

Bit value 04000 in the mode

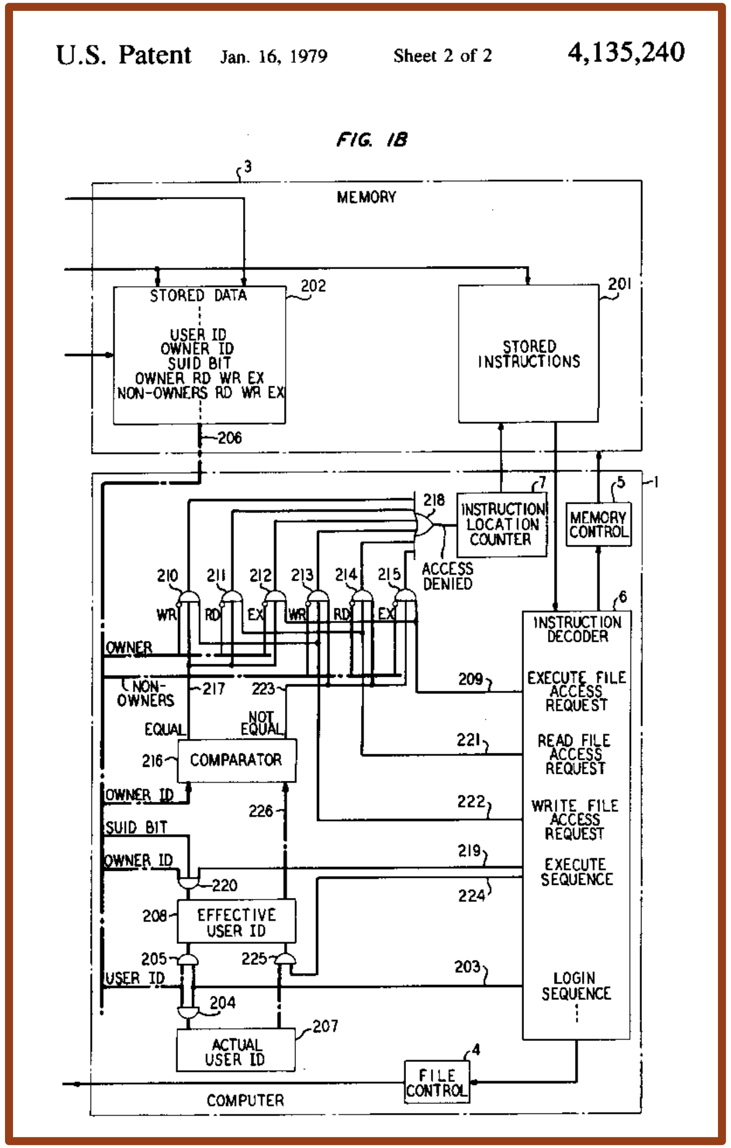
Set using `chmod u+s ...`

Historical Note — the "setuid" Patent

The setuid mechanism was invented by Dennis Ritchie

Patented by Bell Labs (his employer)
  - algorithm expressed as a logic diagram

Patent was placed in the public domain



U.S. Patent Jan. 16, 1979 Sheet 2 of 2 4,135,240

FIG. 1B

Getting and Setting Process Identity

A process may discover its real and effective user and group IDs

	User ID	Group ID
Real	`getuid()`	`getgid()`
Effective	`geteuid()`	`getegid()`

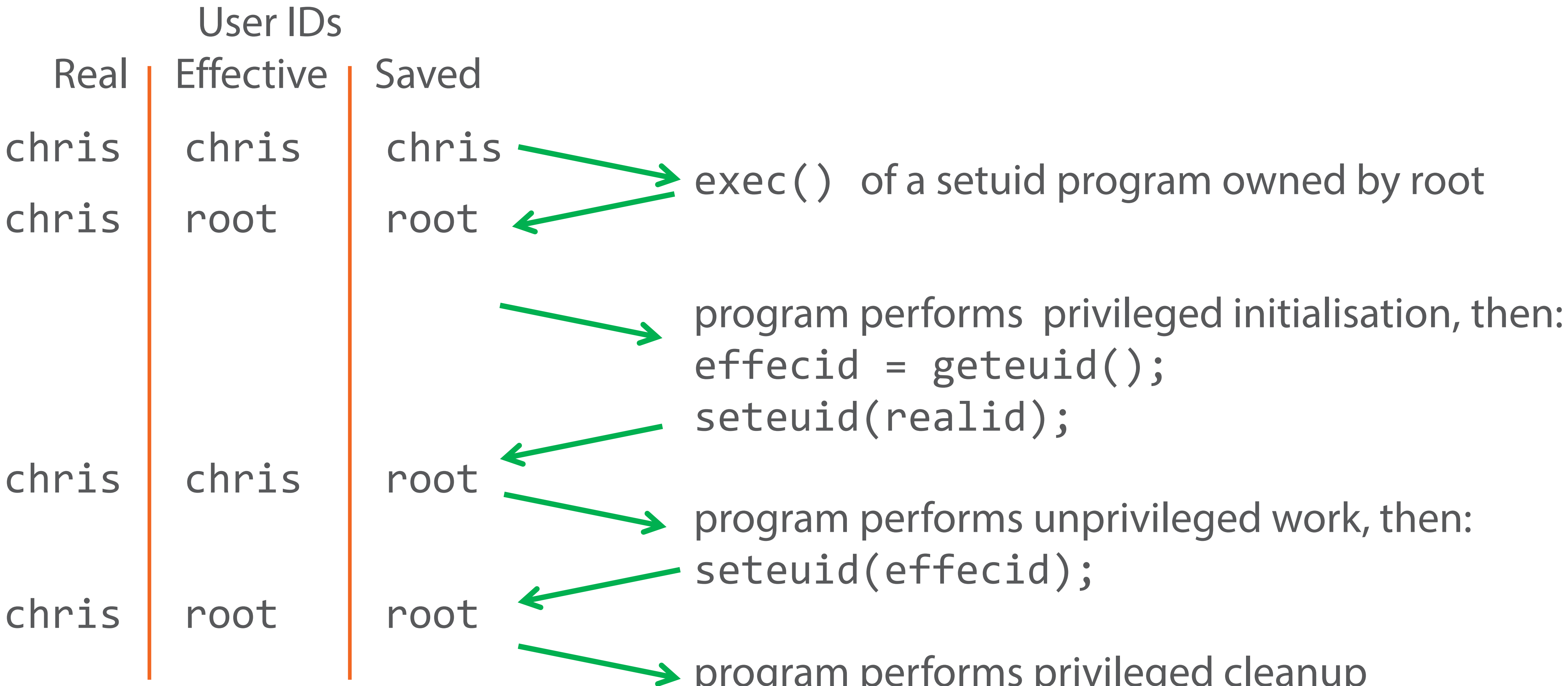
A process remembers its initial effective ID (after an `exec()`)

- called the *saved set-user-ID*

A process may switch its effective user ID between the real ID and the saved set-user-ID:

```
seteuid(uid)
```

Changing Identity on the Fly



	User IDs	
Real	Effective	Saved
chris	chris	chris
chris	root	root
chris	chris	root
chris	root	root

`exec()` of a setuid program owned by root

program performs privileged initialisation, then:

```
effecid = geteuid();
seteuid(realid);
```

program performs unprivileged work, then:

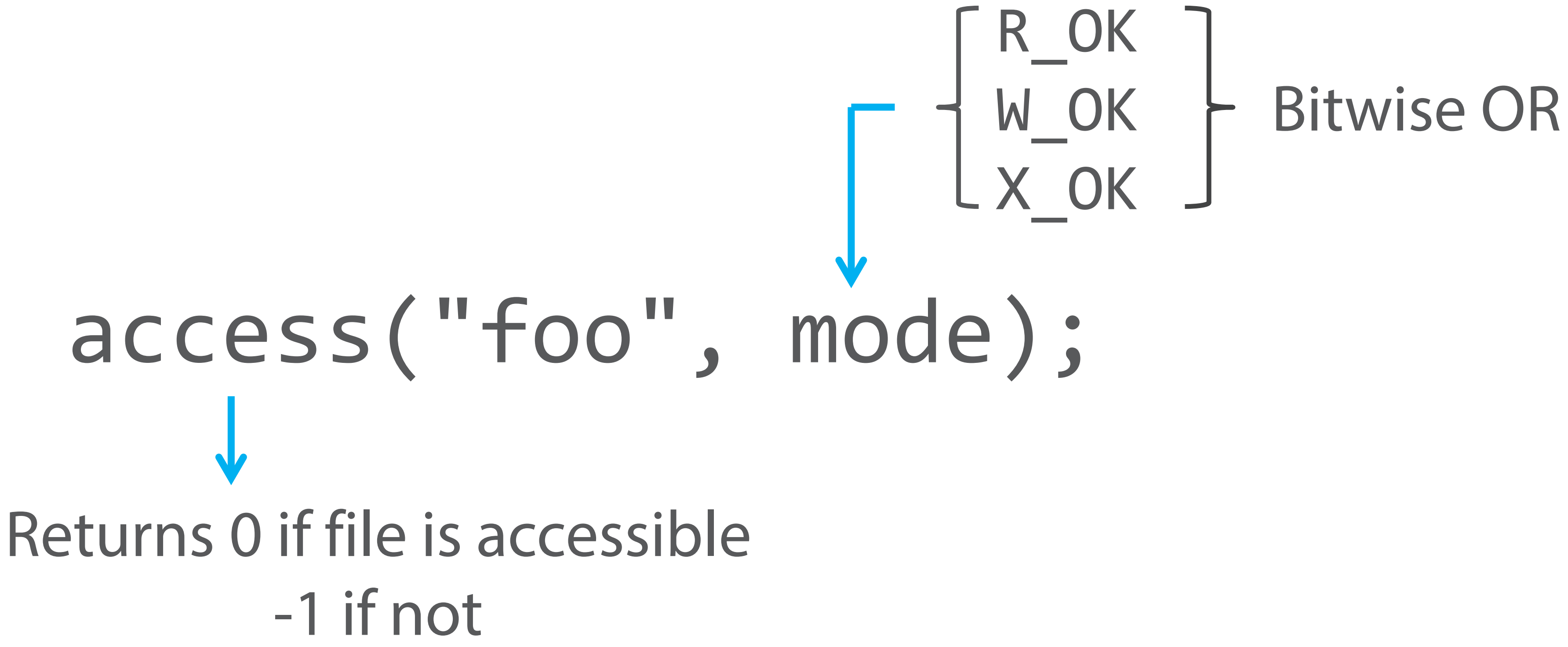
```
seteuid(effecid);
```

program performs privileged cleanup

Testing File Accessibility

`open()` checks file permissions against the effective UID

`access()` checks file permissions against the real UID



R_OK
W_OK
X_OK

Bitwise OR

```
access("foo", mode);
```

Returns 0 if file is accessible
-1 if not

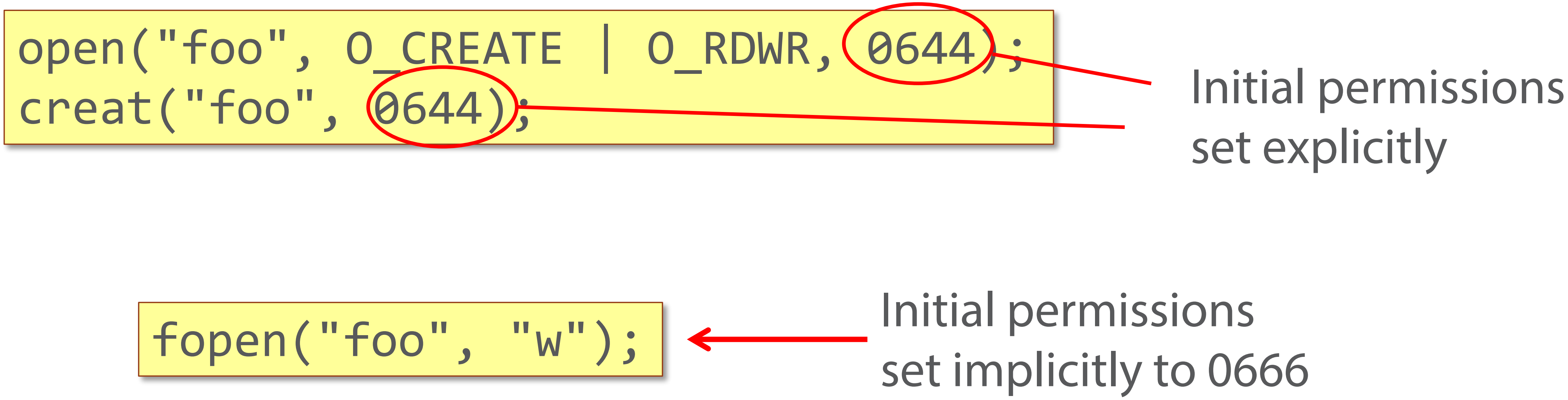
File Permissions

Initial file permissions

Limiting permissions with umask

Changing file permissions

Establishing Initial File Permissions



```
open("foo", O_CREATE | O_RDWR, 0644);
creat("foo", 0644);
```

Initial permissions set explicitly

```
fopen("foo", "w");
```

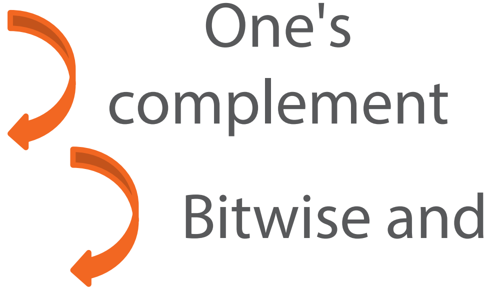
Initial permissions set implicitly to 0666

Limiting Permission with umask

umask is a bit mask of permissions *not* to be assigned

Worked example:

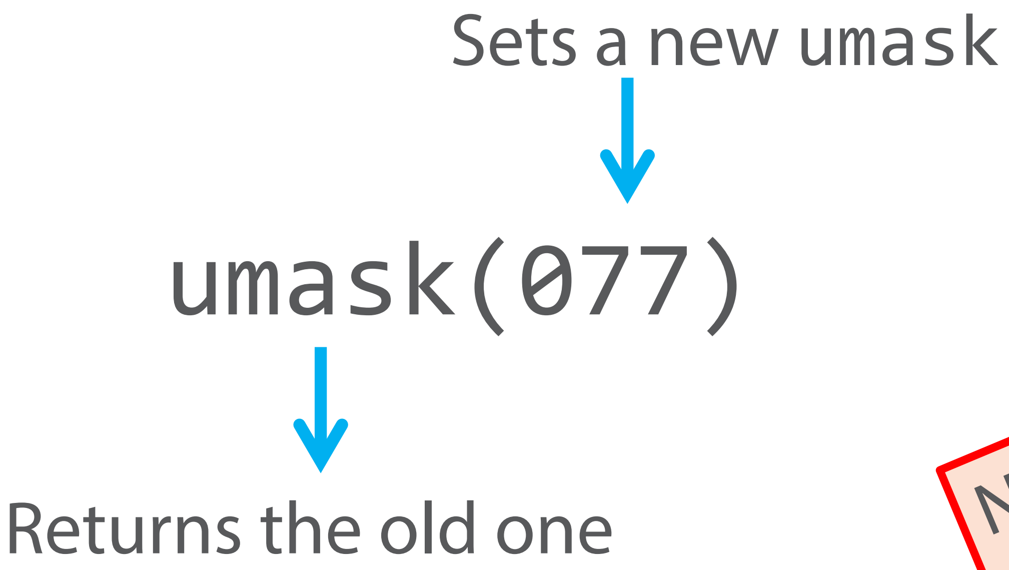
```
umask            022    000 010 010
~umask                  111 101 101
mode requested   666    110 110 110
mode assigned    644    110 100 100
```



One's complement

Bitwise and

Getting and Setting umask



Sets a new umask

```
umask(077)
```

Returns the old one

Note: umask is applied when a file is created. Changing it will not affect file permissions retrospectively

Changing File Permissions

```
chmod("foo", 0600);
```

File permissions can be changed by:

- The file's owner
- Root

Symbolic Constants

Symbolic constants in <stat.h> are useful in specifying the mode

s	g	t	r	w	x	r	w	x	r	w	x
S_ISUID	S_ISGID	S_ISVTX	S_IRUSR	S_IWUSR	S_IXUSR	S_IRGRP	S_IWGRP	S_IXGRP	S_IROTH	S_IWOTH	S_IXOTH

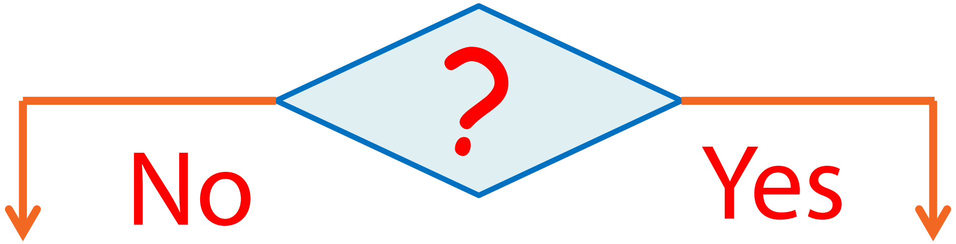
```
chmod("foo", S_IRUSR | S_IWUSR | S_IRGRP);
```

File Ownership

The owner of a new file is the effective UID of the process that creates it

The rules concerning the *group* of a new file are more complicated:

Does the parent directory have the `setgid` bit set?



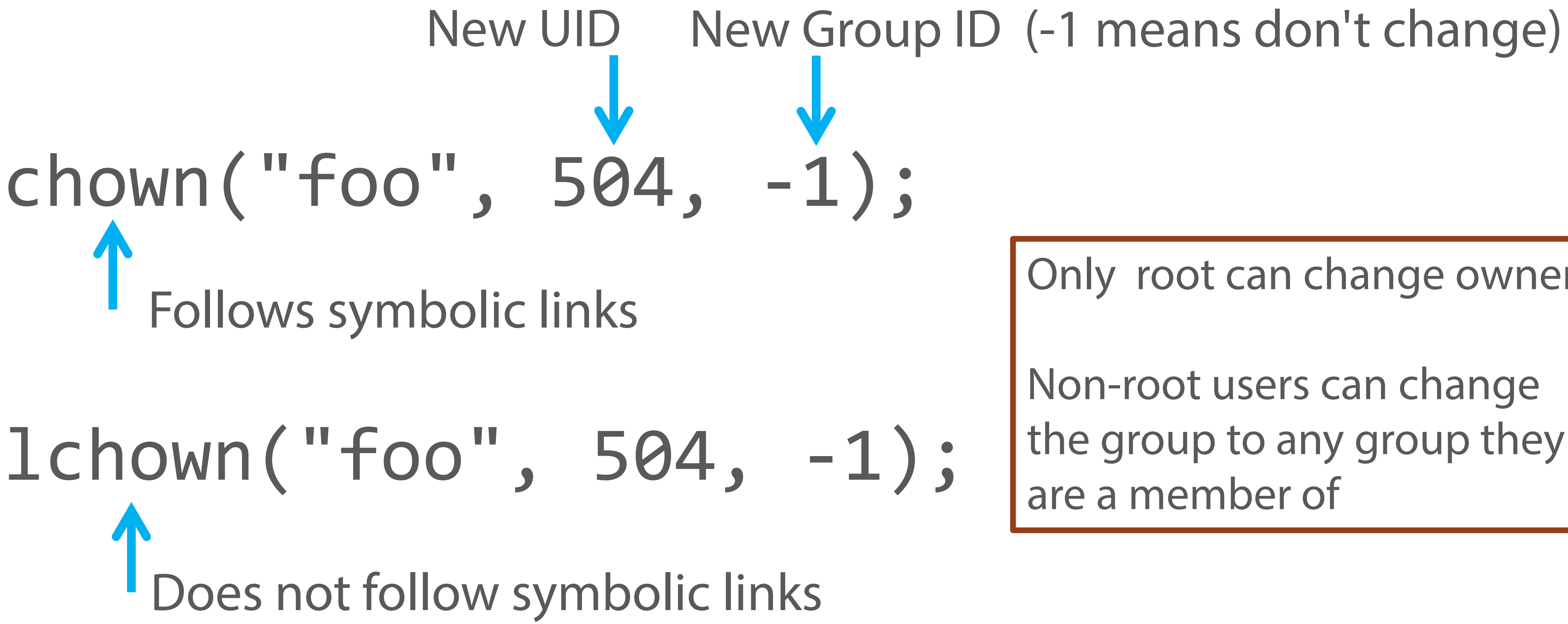
No

Group of file is the effective GID of the process

Yes

Group of file is inherited from the parent directory

Changing Ownership



New UID

New Group ID (-1 means don't change)

```
chown("foo", 504, -1);
```

Follows symbolic links

```
lchown("foo", 504, -1);
```

Does not follow symbolic links

Only root can change ownership

Non-root users can change the group to any group they are a member of

Changing Ownership

If you know the user name of the intended owner, use `getpwnam()` to map it to the numeric ID:

```
/* Make Alice the owner of file foo */

struct passwd *u;
if ((u = getpwnam("alice")) == NULL)
  printf("unknown user\n");
else
  chown("foo", u->pw_uid);
```

Determining Ownership

The stat() call discussed in module 3 returns a file's UID
-- use getpwuid() to map it to a user name

```
/* Display the owner of file foo */
struct stat buf;
struct passwd *u;
stat("foo", &buf);
if ((u = getpwuid(buf.st_uid)) == NULL)
  printf("Unknown user\n");
else
  printf("Owned by %s\n", u->pw_name);
```

Module Summary

User Identity and Accounts

Real and effective process identity

File permissions and ownership

Coming up in the Next Module

Signals

Signal types and their uses

Sending signals

Writing signal handlers

Seven things to do with signals