Filesystems

Inodes

The design of the traditional Unix filesystem is based on the *inode*

Filesystems

Inodes

The design of the traditional Unix filesystem is based on the *inode*

Each file has its own inode

Filesystems

Inodes

The design of the traditional Unix filesystem is based on the *inode*

Each file has its own inode

The inode is a fixed size structure (stored on disk) that contains all the information about a file, its *metadata*

Filesystems

Inodes

Information in the inode includes

Filesystems

Inodes

Information in the inode includes

- Timestamps. Dates and times this file was last accessed and last modified

Filesystems

Inodes

Information in the inode includes

- Timestamps. Dates and times this file was last accessed and last modified
- Ownership. The userid of the owner of this file, for protection purposes

Filesystems

Inodes

Information in the inode includes

- Timestamps. Dates and times this file was last accessed and last modified
- Ownership. The userid of the owner of this file, for protection purposes
- Size. How big the file is currently

Filesystems

Inodes

Information in the inode includes

- Timestamps. Dates and times this file was last accessed and last modified
- Ownership. The userid of the owner of this file, for protection purposes
- Size. How big the file is currently
- Type. Whether this is a *plain file*, or a directory, or some other kind of *special file*

Filesystems

Inodes

Information in the inode includes

- Timestamps. Dates and times this file was last accessed and last modified
- Ownership. The userid of the owner of this file, for protection purposes
- Size. How big the file is currently
- Type. Whether this is a *plain file*, or a directory, or some other kind of *special file*
- Access permissions. Who can read or write or run this file (if it is a program)

Filesystems

Inodes

Information in the inode includes

- Timestamps. Dates and times this file was last accessed and last modified
- Ownership. The userid of the owner of this file, for protection purposes
- Size. How big the file is currently
- Type. Whether this is a *plain file*, or a directory, or some other kind of *special file*
- Access permissions. Who can read or write or run this file (if it is a program)
- Reference count. The number of names this file has

Filesystems

Inodes

Information in the inode includes

- Timestamps. Dates and times this file was last accessed and last modified
- Ownership. The userid of the owner of this file, for protection purposes
- Size. How big the file is currently
- Type. Whether this is a *plain file*, or a directory, or some other kind of *special file*
- Access permissions. Who can read or write or run this file (if it is a program)
- Reference count. The number of names this file has
- Pointers to areas on the disk where the actual data lives

Filesystems

Inodes

Notice that filenames are *not* in the inode

Filesystems

Inodes

Notice that filenames are *not* in the inode

Filenames are stored in directories

Filesystems

Inodes

Notice that filenames are *not* in the inode

Filenames are stored in directories

A directory is essentially just a list of names of files and subdirectories, together with their inode numbers

Filesystems

Inodes

Notice that filenames are *not* in the inode

Filenames are stored in directories

A directory is essentially just a list of names of files and subdirectories, together with their inode numbers

Name	Inode
`foo.c`	23
`ff.html`	42
`mydata`	7

Filesystems

Inodes

Notice that filenames are *not* in the inode

Filenames are stored in directories

A directory is essentially just a list of names of files and subdirectories, together with their inode numbers

Name	Inode
`foo.c`	23
`ff.html`	42
`mydata`	7

Originally just a table, these days clever datastructures are used to manage the large numbers of names we use

Filesystems

Inodes

This is how a file can have many names: multiple directory entries referring to the same inode

Filesystems

Inodes

This is how a file can have many names: multiple directory entries referring to the same inode

As a consequence a file cannot know its own name(s) as the names are independent of the file

Filesystems

Inodes

This is how a file can have many names: multiple directory entries referring to the same inode

As a consequence a file cannot know its own name(s) as the names are independent of the file

In some sense, the inode number is the true name of the file

Filesystems

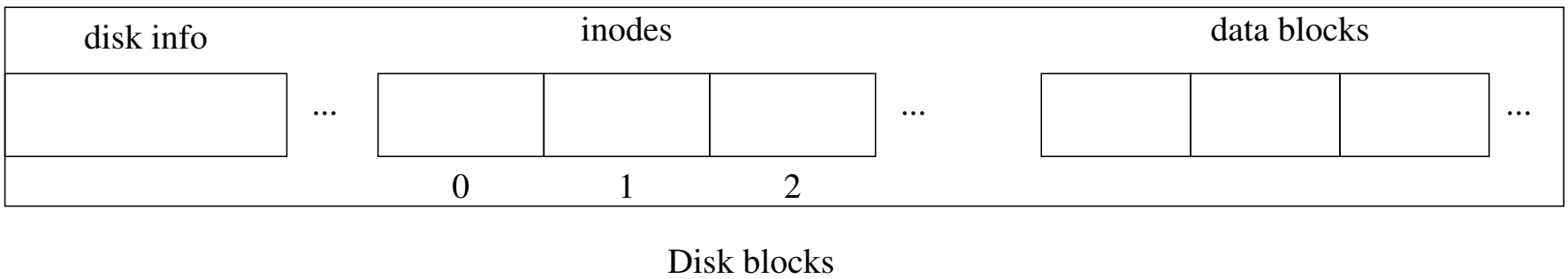
Inodes

As inodes are a fixed size, it is easy to put them in a simple array on disk and just refer to them by their index in the array: the *inode number*

Filesystems

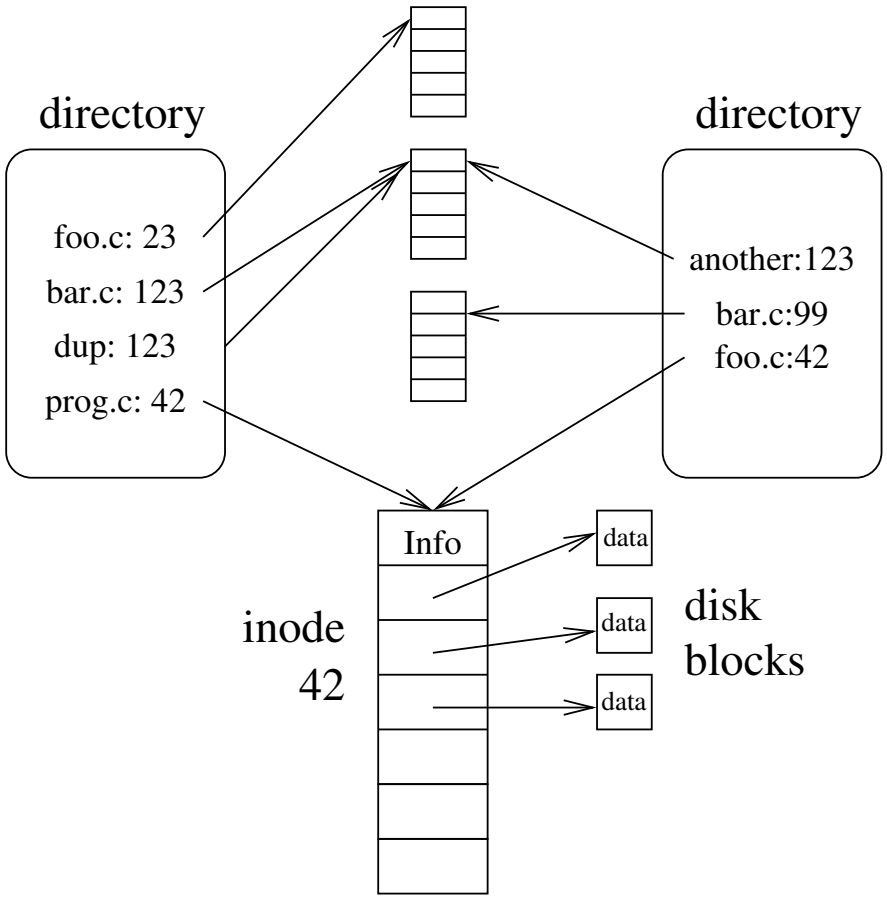
Inodes

As inodes are a fixed size, it is easy to put them in a simple array on disk and just refer to them by their index in the array: the *inode number*



Filesystems

Inodes



Filesystems

Inodes

There are a couple of special names always to be found in every directory

Filesystems

Inodes

There are a couple of special names always to be found in every directory

The name `..` refers back to the parent directory. This allows us to crawl up the hierarchy until we reach the root

Filesystems

Inodes

There are a couple of special names always to be found in every directory

The name `..` refers back to the parent directory. This allows us to crawl up the hierarchy until we reach the root

So `../foo/bar.c` is a name of a file in a sibling directory: up; across; then down in the hierarchy

Filesystems

Inodes

There are a couple of special names always to be found in every directory

The name `..` refers back to the parent directory. This allows us to crawl up the hierarchy until we reach the root

So `../foo/bar.c` is a name of a file in a sibling directory: up; across; then down in the hierarchy

The `..` of `/` is `/`

Filesystems

Inodes

There are a couple of special names always to be found in every directory

The name `..` refers back to the parent directory. This allows us to crawl up the hierarchy until we reach the root

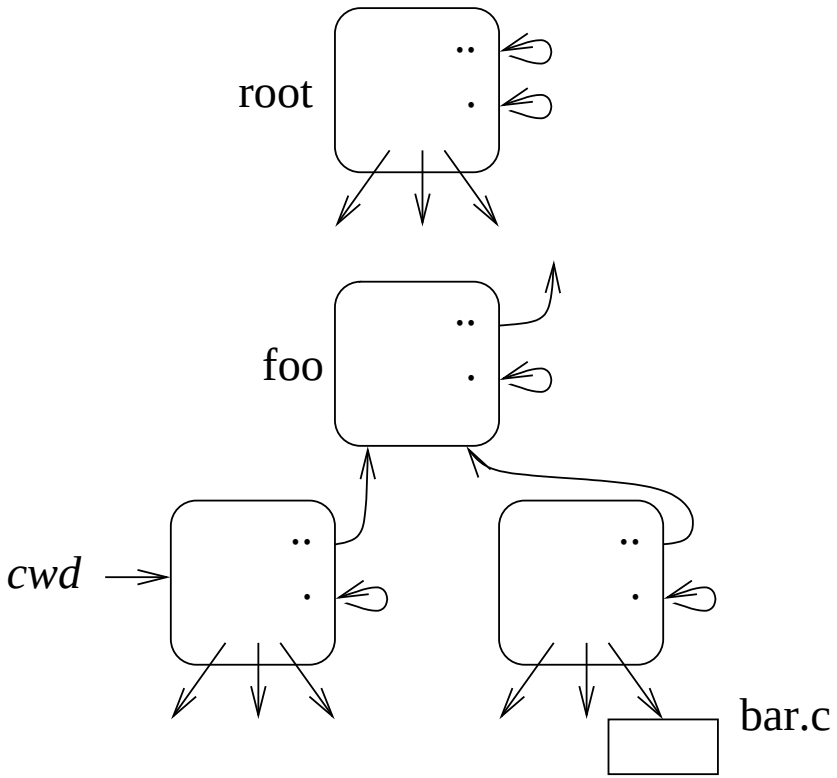
So `../foo/bar.c` is a name of a file in a sibling directory: up; across; then down in the hierarchy

The `..` of `/` is `/`

The name `.` refers a directory back to itself. This often turns out to be useful to do

Filesystems

Inodes



Filesystems

Inodes

The inode contains a reference count: the number of names the file/inode has

Filesystems

Inodes

The inode contains a reference count: the number of names the file/inode has

If the count drops to zero, the OS can remove the file

Filesystems

Inodes

The inode contains a reference count: the number of names the file/inode has

If the count drops to zero, the OS can remove the file

Deleting a file is a matter of

Filesystems

Inodes

The inode contains a reference count: the number of names the file/inode has

If the count drops to zero, the OS can remove the file

Deleting a file is a matter of

- Removing the name reference in the relevant directory

Filesystems

Inodes

The inode contains a reference count: the number of names the file/inode has

If the count drops to zero, the OS can remove the file

Deleting a file is a matter of

- Removing the name reference in the relevant directory
- Decrementing the reference count in the inode

Filesystems

Inodes

The inode contains a reference count: the number of names the file/inode has

If the count drops to zero, the OS can remove the file

Deleting a file is a matter of

- Removing the name reference in the relevant directory
- Decrementing the reference count in the inode
- If the count reaches 0, we can free the inode and the disk blocks it refers to

Filesystems

Inodes

In fact, each time a program opens a file the OS increments the count; and decrements it when the program closes the file (possibly when the program exits)

Filesystems

Inodes

In fact, each time a program opens a file the OS increments the count; and decrements it when the program closes the file (possibly when the program exits)

So it is possible for a program to create a new file (inc); open it (inc); delete it (dec); and still be able to read and write to it

Filesystems

Inodes

In fact, each time a program opens a file the OS increments the count; and decrements it when the program closes the file (possibly when the program exits)

So it is possible for a program to create a new file (inc); open it (inc); delete it (dec); and still be able to read and write to it

The file will only disappear when the program ends (dec)

Filesystems

Inodes

In fact, each time a program opens a file the OS increments the count; and decrements it when the program closes the file (possibly when the program exits)

So it is possible for a program to create a new file (inc); open it (inc); delete it (dec); and still be able to read and write to it

The file will only disappear when the program ends (dec)

No other process can see this file: there is no name in any directory