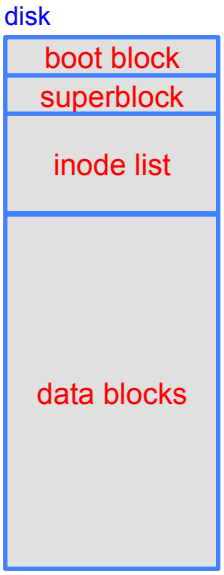
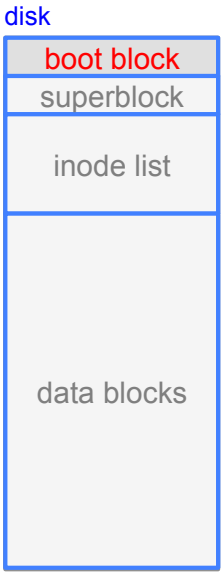
UNIX File System Implementation

- Simple implementation of original UNIX File System
- File System Layout
- File-System Information and Free-Space Management
- I-nodes and I-node Management
- Naming and Directories
- Evaluation

File-System Layout of UNIX FS

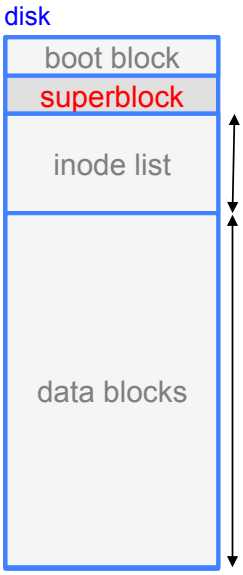


File-System Layout of UNIX FS



Boot Block: Contains bootstrap code for operating system.

File-System Layout of UNIX FS

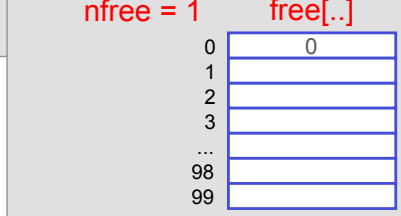


Superblock: Contains information about state of file system, such as size, free space info, etc.

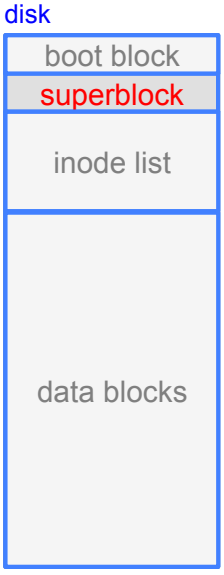
```
struct {
  ushort isize;
  ushort fsize;
  ushort nfree;
  ushort free[100];
  ushort ninode;
  ushort inode[100];
  char flock, ilock, fmod;
  ushort time[2];
}
```

Free Space Management

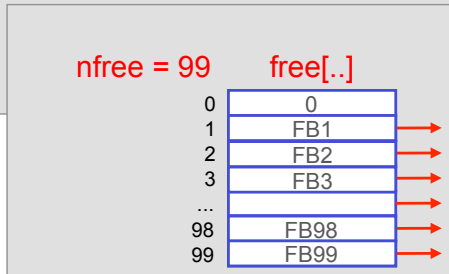
Superblock: Contains information about state of file system, such as size, free space info, etc.



Free Space Management

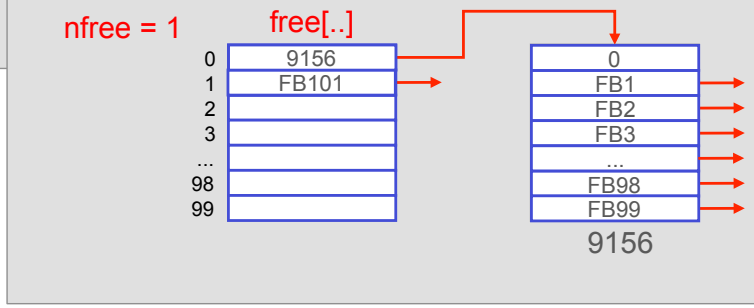


Superblock: Contains information about state of file system, such as size, free space info, etc.

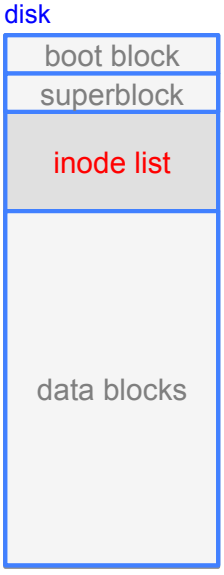


Free Space Management

Superblock: Contains information about state of file system, such as size, free space info, etc.

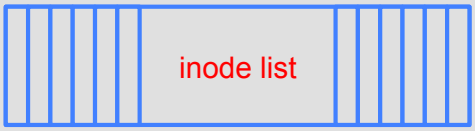


File-System Layout of UNIX FS

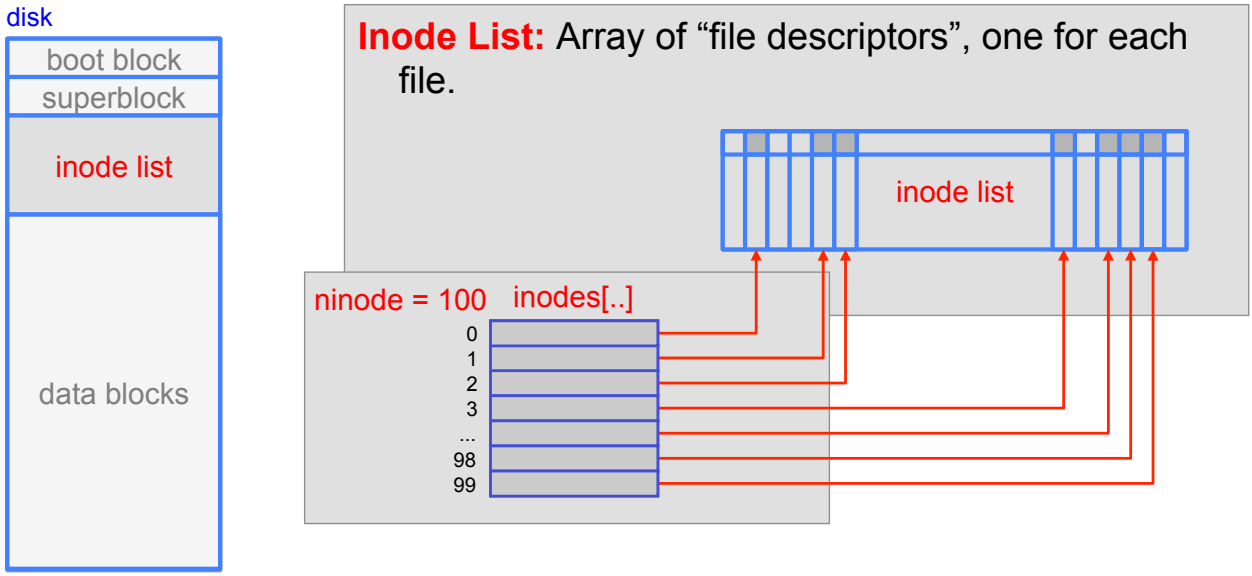


Inode List: Array of meta-data records, one for each file.

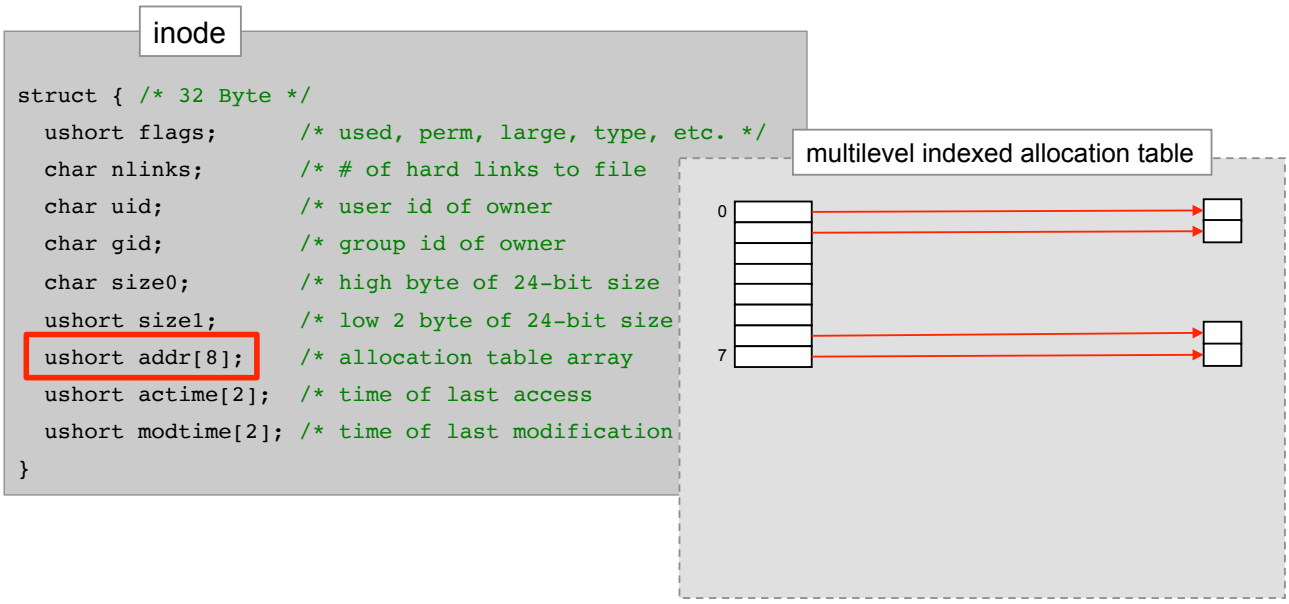
```
struct { /* 32 Byte */
  ushort flags;
  char nlinks;
  char uid;
  char gid;
  char size0;
  ushort size1;
  ushort addr[8];
  ushort actime[2];
  ushort modtime[2];
}
```



File-System Layout of UNIX FS



UNIX inodes

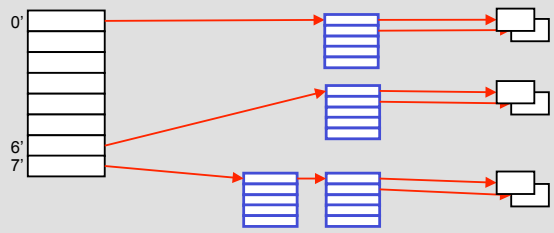


UNIX inodes

inode

```
struct { /* 32 Byte */
  ushort flags;      /* used, perm, large, type, etc. */
  char nlinks;       /* # of hard links to file
  char uid;          /* user id of owner
  char gid;          /* group id of owner
  char size0;        /* high byte of 24-bit size
  ushort size1;      /* low 2 byte of 24-bit size
  ushort addr[8];    /* allocation table array
  ushort actime[2];  /* time of last access
  ushort modtime[2]; /* time of last modification
}
```

multilevel indexed allocation table

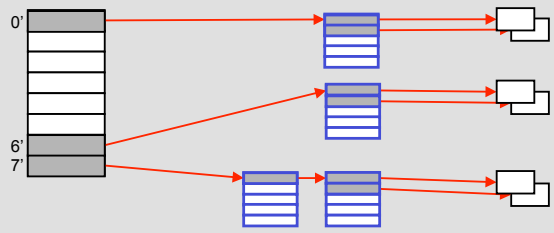


UNIX inodes

inode

```
struct { /* 32 Byte */
  ushort flags;      /* used, perm, large, type, etc. */
  char nlinks;       /* # of hard links to file
  char uid;          /* user id of owner
  char gid;          /* group id of owner
  char size0;        /* high byte of 24-bit size
  ushort size1;      /* low 2 byte of 24-bit size
  ushort addr[8];    /* allocation table array
  ushort actime[2];  /* time of last access
  ushort modtime[2]; /* time of last modification
}
```

multilevel indexed allocation table



UNIX inodes

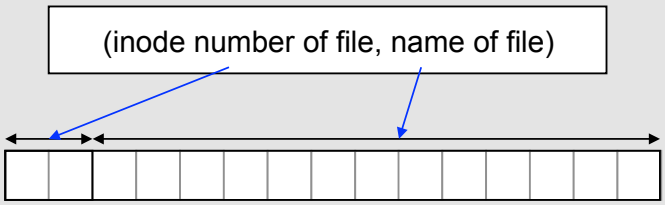
inode

```
struct { /* 32 Byte */
  ushort flags;      /* used, perm, larg
  char nlinks;       /* # of hard links
  char uid;          /* user id of owner
  char gid;          /* group id of owne
  char size0;        /* high byte of 24-
  ushort size1;      /* low 2 byte of 24
  ushort addr[8];    /* allocation table
  ushort actime[2];  /* time of last acc
  ushort modtime[2]; /* time of last modification     */
}
```

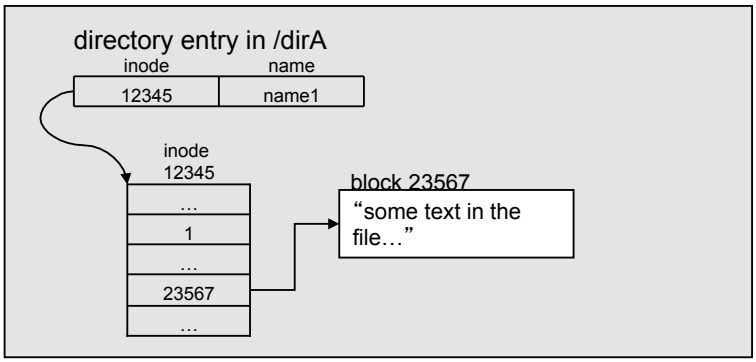
where is the file name?!

Name information is contained in separate **Directory Files**, which contain entries of type:

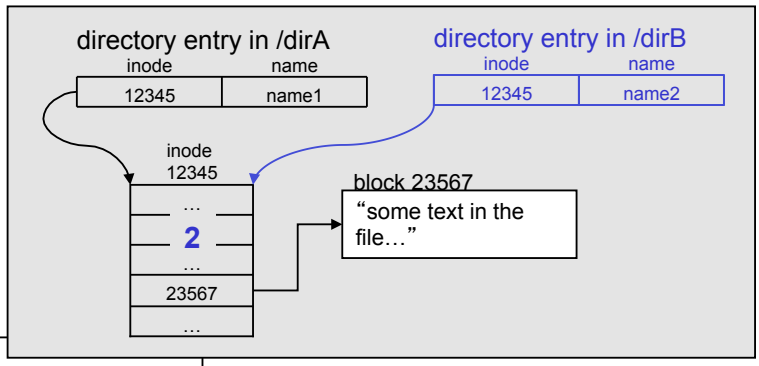
(inode number of file, name of file)



Files can have multiple Names (Hard Links)



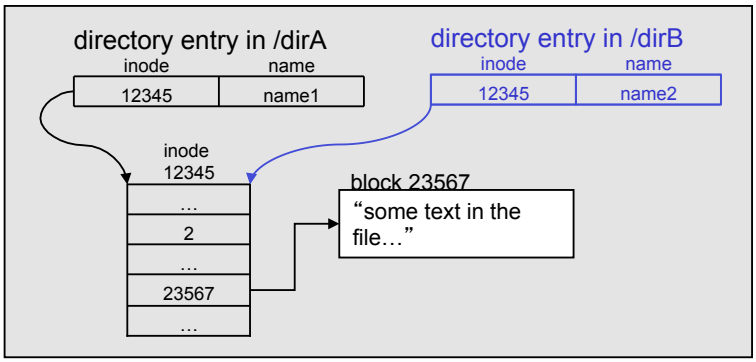
Files can have multiple Names (Hard Links)



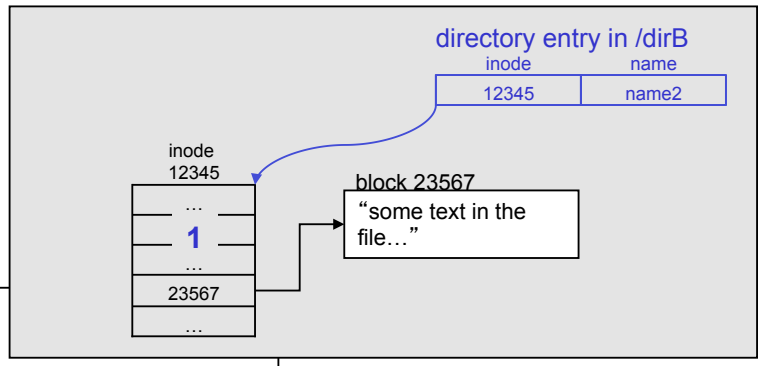
The **link** system call:

```
#include <stdio.h>
#include<unistd.h>

if (link("/dirA/name1", "/dirB/name2") == -1)
  perror("failed to make new link in /dirB");
```

Deleting Files

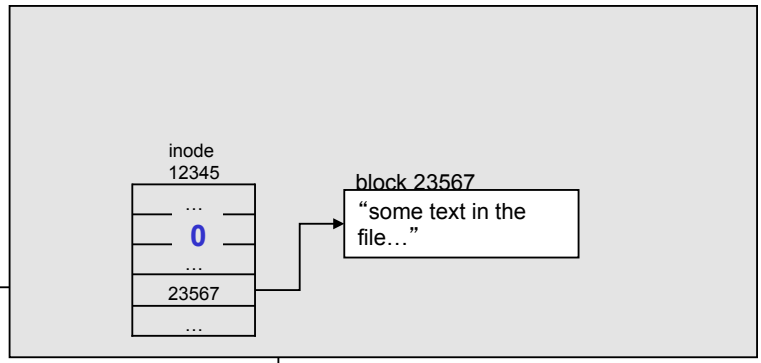


Deleting Files



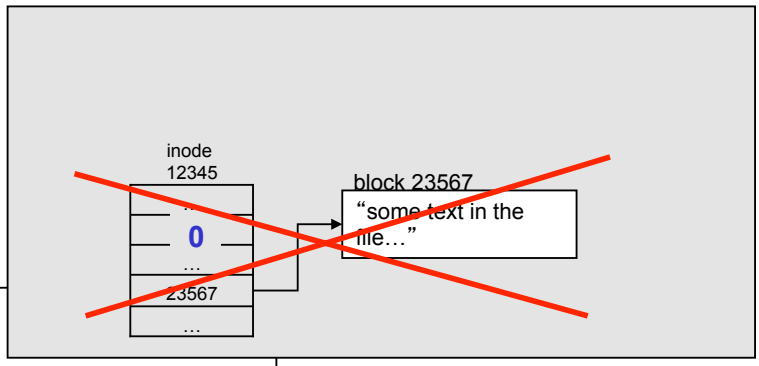

```
#include <stdio.h>
#include<unistd.h>

if (unlink("/dirA/name1") == -1)
  perror("failed to delete link in /dirA");
```

Deleting Files



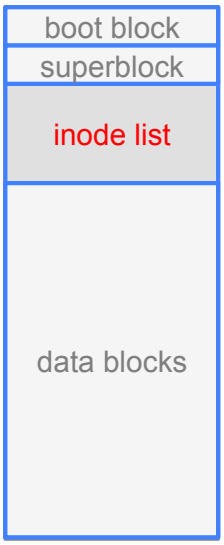

```
#include <stdio.h>
#include<unistd.h>

if (unlink("/dirA/name1") == -1)
  perror("failed to delete link in /dirA");

if (unlink("/dirB/name2") == -1)
  perror("failed to delete link in /dirB");
```

Deleting Files

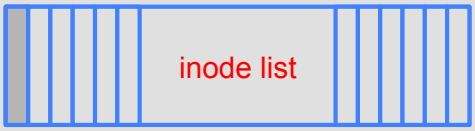



```
#include <stdio.h>
#include<unistd.h>

if (unlink("/dirA/name1") == -1)
  perror("failed to delete link in /dirA");

if (unlink("/dirB/name2") == -1)
  perror("failed to delete link in /dirB");
```

File-System Layout of UNIX FS



Inode List: Array of "file descriptors", one for each file.



Pros/Cons of Original Unix File System

Pros:

- Data in small files can be accessed directly from the inode.
  - One read to fetch inode, another to fetch first data block.
- Larger files can be accessed efficiently
- No external fragmentation.

Cons:

- Original file system uses 512-byte blocks.
- Inodes kept separately from data, causing long seeks to access data.
- Inodes of files in a common directory not kept together, causing low performance when searching directories.
- Data blocks of a file are not stored together.
- Free list quickly scrambles, increasing overhead of finding free blocks.

Pros/Cons of Original Unix File System

Pros:

- Data in small files can be accessed directly from the inode.
  - One read to fetch inode, another to fetch first data block.
- Larger files can be accessed efficiently
- No ext

Measurements of original file system used as little as 2% of the available disk bandwidth!

Cons:

- Original file system uses 512-byte blocks.
- Inodes kept separately from data, causing long seeks to access data.
- Inodes of files in a common directory not kept together, causing low performance when searching directories.
- Data blocks of a file are not stored together.
- Free list quickly scrambles, increasing overhead of finding free blocks.

UNIX File System Implementation

- Simple implementation of original UNIX File System
- File System Layout
- Free-Space Management
- I-nodes and I-node Management
- Evaluation