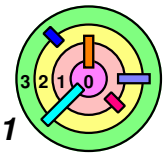
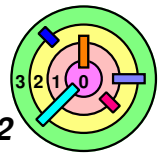
Housekeeping (Lecture 15 - 10/16/2013)

- Kernel #1 due at 11:45pm on Friday, 10/25/2013
  - if you have code from a previous semester, be very careful and *not copy any code from it*
    - it's best if you just get rid of it
- Post questions about the kernel #1 to *class Google Group*
  - your classmates may have better answers!
    - and faster turn-around
  - you should state what you have found so other students know that you have tried (and therefore, more willing to help)



Access Protection

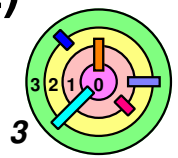
- OS needs to make sure that only authorized processes are allowed access to system resources
  - various ways to provide this
- Unix (and many other systems, such as Windows) associates with files some indication of which *security principals* are allowed access
  - along with what sort of access is allowed
- A *security principal* is normally a user or group of users
  - a "user" can be an identity used by processes performing system functions
  - each running process can have several security principals associated with it
    - all processes have a user identification and a set of group identifications
    - for Sixth-Edition Unix, only one user ID and one group ID



Access Protection

- Each file has associated with it a set of access permissions
  - for each of the 3 classes of principals, what sorts of operations on the file are allowed
  - the 3 classes are:
    - *user:* owner of the file
    - *group:* group owner of the file
    - *others:* everyone else
  - the operations are grouped into 3 classes:
    - *read:* can read a file or directory
    - *write:* can write a file or directory
    - *execute:* one must have *execute permission* for a *directory* in order to *follow a path through it*

- *Rules for checking permissions*
  1) determines the *smallest class of principals the requester belongs to* (user being smallest and others being largest)
  2) then it checks for appropriate permissions with that class



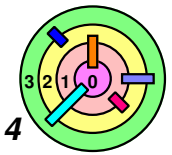
Permissions Example

```
% ls -lR
.:
total 2
drwxr-x--x  2 bill   adm     1024 Dec 17 13:34 A
drwxr-----  2 bill   adm     1024 Dec 17 13:34 B

./A:
total 1
-rw-rw-rw-  1 bill   adm      593 Dec 17 13:34 x

./B:
total 2
-r--rw-rw-  1 bill   adm      446 Dec 17 13:34 x
-rw----rw-  1 trina  adm      446 Dec 17 13:45 y
```

- Suppose that `bill` and `trina` are members of the `adm` group and `andy` is not
  1) Q: May `andy` list the contents of directory `A`?



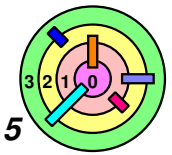
Permissions Example

```
% ls -lR
.:
total 2
drwxr-x--x   2 bill    adm     1024 Dec 17 13:34 A
drwxr-----   2 bill    adm     1024 Dec 17 13:34 B

./A:
total 1
-rw-rw-rw-   1 bill    adm      593 Dec 17 13:34 x

./B:
total 2
-r--rw-rw-   1 bill    adm      446 Dec 17 13:34 x
-rw----rw-   1 trina   adm      446 Dec 17 13:45 y
```

- Suppose that `bill` and `trina` are members of the `adm` group and `andy` is not
  1) Q: May `andy` list the contents of directory `A`?
     A: No



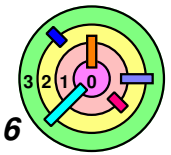
Permissions Example

```
% ls -lR
.:
total 2
drwxr-x--x  2 bill   adm     1024 Dec 17 13:34 A
drwxr-----  2 bill   adm     1024 Dec 17 13:34 B

./A:
total 1
-rw-rw-rw-  1 bill   adm      593 Dec 17 13:34 x

./B:
total 2
-r--rw-rw-  1 bill   adm      446 Dec 17 13:34 x
-rw----rw-  1 trina  adm      446 Dec 17 13:45 y
```

- Suppose that `bill` and `trina` are members of the `adm` group and `andy` is not
  2) Q: May `andy` read `A/x`?



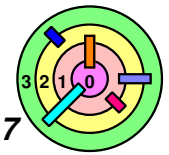
Permissions Example

```
% ls -lR
.:
total 2
drwxr-x--x  2 bill   adm     1024 Dec 17 13:34 A
drwxr-----  2 bill   adm     1024 Dec 17 13:34 B

./A:
total 1
-rw-rw-rw-  1 bill   adm      593 Dec 17 13:34 x

./B:
total 2
-r--rw-rw-  1 bill   adm      446 Dec 17 13:34 x
-rw----rw-  1 trina  adm      446 Dec 17 13:45 y
```

- Suppose that `bill` and `trina` are members of the `adm` group and `andy` is not
  2) Q: May `andy` read `A/x`?
     A: Yes



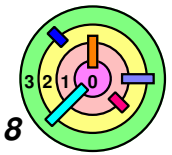
Permissions Example

```
% ls -lR
.:
total 2
drwxr-x--x  2 bill   adm     1024 Dec 17 13:34 A
drwxr-----  2 bill   adm     1024 Dec 17 13:34 B

./A:
total 1
-rw-rw-rw-  1 bill   adm      593 Dec 17 13:34 x

./B:
total 2
-r--rw-rw-  1 bill   adm      446 Dec 17 13:34 x
-rw----rw-  1 trina  adm      446 Dec 17 13:45 y
```

- Suppose that `bill` and `trina` are members of the `adm` group and `andy` is not
  3) Q: May `trina` list the contents of directory `B`?



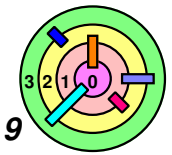
Permissions Example

```
% ls -lR
.:
total 2
drwxr-x--x  2 bill   adm     1024 Dec 17 13:34 A
drwxr-----  2 bill   adm     1024 Dec 17 13:34 B

./A:
total 1
-rw-rw-rw-  1 bill   adm      593 Dec 17 13:34 x

./B:
total 2
-r--rw-rw-  1 bill   adm      446 Dec 17 13:34 x
-rw----rw-  1 trina  adm      446 Dec 17 13:45 y
```

- Suppose that `bill` and `trina` are members of the `adm` group and `andy` is not
  - 3) Q: May `trina` list the contents of directory `B`?
    A: Yes



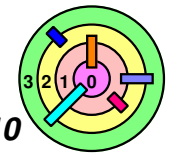
Permissions Example

```
% ls -lR
.:
total 2
drwxr-x--x  2 bill   adm     1024 Dec 17 13:34 A
drwxr-----  2 bill   adm     1024 Dec 17 13:34 B

./A:
total 1
-rw-rw-rw-  1 bill   adm      593 Dec 17 13:34 x

./B:
total 2
-r--rw-rw-  1 bill   adm      446 Dec 17 13:34 x
-rw----rw-  1 trina  adm      446 Dec 17 13:45 y
```

- Suppose that `bill` and `trina` are members of the `adm` group and `andy` is not
  4) Q: May `trina` modify `B/y`?



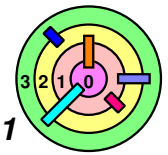
Permissions Example

```
% ls -lR
.:
total 2
drwxr-x--x  2 bill   adm     1024 Dec 17 13:34 A
drwxr-----  2 bill   adm     1024 Dec 17 13:34 B

./A:
total 1
-rw-rw-rw-  1 bill   adm      593 Dec 17 13:34 x

./B:
total 2
-r--rw-rw-  1 bill   adm      446 Dec 17 13:34 x
-rw----rw-  1 trina  adm      446 Dec 17 13:45 y
```

- Suppose that `bill` and `trina` are members of the `adm` group and `andy` is not
  4) Q: May `trina` modify `B/y`?
     A: No



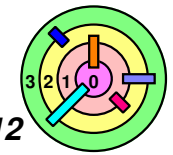
Permissions Example

```
% ls -lR
.:
total 2
drwxr-x--x  2 bill   adm     1024 Dec 17 13:34 A
drwxr-----  2 bill   adm     1024 Dec 17 13:34 B

./A:
total 1
-rw-rw-rw-  1 bill   adm      593 Dec 17 13:34 x

./B:
total 2
-r--rw-rw-  1 bill   adm      446 Dec 17 13:34 x
-rw----rw-  1 trina  adm      446 Dec 17 13:45 y
```

- Suppose that `bill` and `trina` are members of the `adm` group and `andy` is not
  5) Q: May `bill` modify `B/x`?



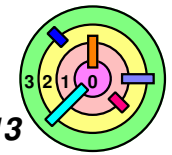
Permissions Example

```
% ls -lR
.:
total 2
drwxr-x--x  2 bill   adm     1024 Dec 17 13:34 A
drwxr-----  2 bill   adm     1024 Dec 17 13:34 B

./A:
total 1
-rw-rw-rw-  1 bill   adm      593 Dec 17 13:34 x

./B:
total 2
-r--rw-rw-  1 bill   adm      446 Dec 17 13:34 x
-rw----rw-  1 trina  adm      446 Dec 17 13:45 y
```

- Suppose that `bill` and `trina` are members of the `adm` group and `andy` is not
  5) Q: May `bill` modify `B/x`?
     A: No



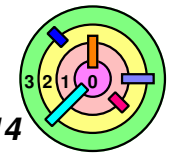
Permissions Example

```
% ls -lR
.:
total 2
drwxr-x--x  2 bill   adm     1024 Dec 17 13:34 A
drwxr-----  2 bill   adm     1024 Dec 17 13:34 B

./A:
total 1
-rw-rw-rw-  1 bill   adm      593 Dec 17 13:34 x

./B:
total 2
-r--rw-rw-  1 bill   adm      446 Dec 17 13:34 x
-rw----rw-  1 trina  adm      446 Dec 17 13:45 y
```

- Suppose that `bill` and `trina` are members of the `adm` group and `andy` is not
  6) Q: May `bill` read `B/y`?



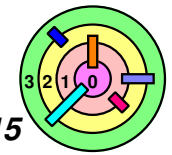
Permissions Example

```
% ls -lR
.:
total 2
drwxr-x--x  2 bill   adm     1024 Dec 17 13:34 A
drwxr-----  2 bill   adm     1024 Dec 17 13:34 B

./A:
total 1
-rw-rw-rw-  1 bill   adm      593 Dec 17 13:34 x

./B:
total 2
-r--rw-rw-  1 bill   adm      446 Dec 17 13:34 x
-rw----rw-  1 trina  adm      446 Dec 17 13:45 y
```

- Suppose that `bill` and `trina` are members of the `adm` group and `andy` is not
  6) Q: May `bill` read `B/y`?
     A: No

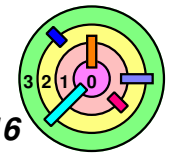


Open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *path, int options [, mode_t mode])
```

options

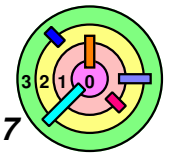
- O_RDONLY open for reading only
- O_WRONLY open for writing only
- O_RDWR open for reading and writing
- O_APPEND set the file offset to end of file prior to each write
- O_CREAT if the file does not exist, then create it, setting its mode to mode adjusted by umask
- O_EXCL: if O_EXCL and O_CREAT are set, then open fails if the file exists
- O_TRUNC delete any previous contents of the file
- O_NONBLOCK don't wait if I/O cannot be done immediately



Setting File Permissions

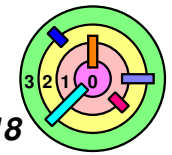
```
#include <sys/types.h>
#include <sys/stat.h>
int chmod(const char *path, mode_t mode)
```

- sets the file permissions of the given file to those specified in `mode`
- only the owner of a file and the superuser may change its permissions
- nine combinable possibilities for mode (read/write/execute for user, group, and others)
- `S_IRUSR` (0400), `S_IWUSR` (0200), `S_IXUSR` (0100)
- `S_IRGRP` (040), `S_IWGRP` (020), `S_IXGRP` (010)
- `S_IROTH` (04), `S_IWOTH` (02), `S_IXOTH` (01)
  - note: numeric prefix of 0 means the number is in octal format



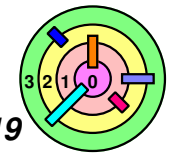
Creating a File

- Use either `open` or `creat`
  - `open(const char *pathname, int flags, mode_t mode)`
    - flags must include O_CREAT
  - `creat(const char *pathname, mode_t mode)`
  - `open` is preferred
- The `mode` parameter helps specify the permissions of the newly created file
  - permissions = `mode & ~umask`



Umask

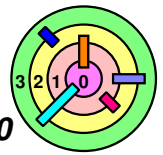
- Standard programs create files with "maximum needed permissions" as *mode*
  - compilers: `0777`
  - editors: `0666`
- Per-process parameter, *umask*, used to *turn off* undesired permission bits
  - e.g., turn off all permissions for others, write permission for group: set umask to `027`
  - compilers: permissions = `0777 & ~(027) = 0750`
  - editors: permissions = `0666 & ~(027) = 0640`
  - set with `umask()` system call or (usually) `umask` shell command



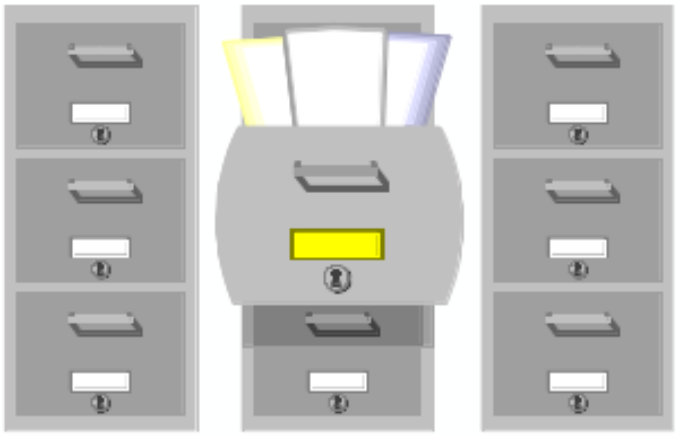
Ch 6: File Systems

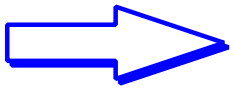
Bill Cheng

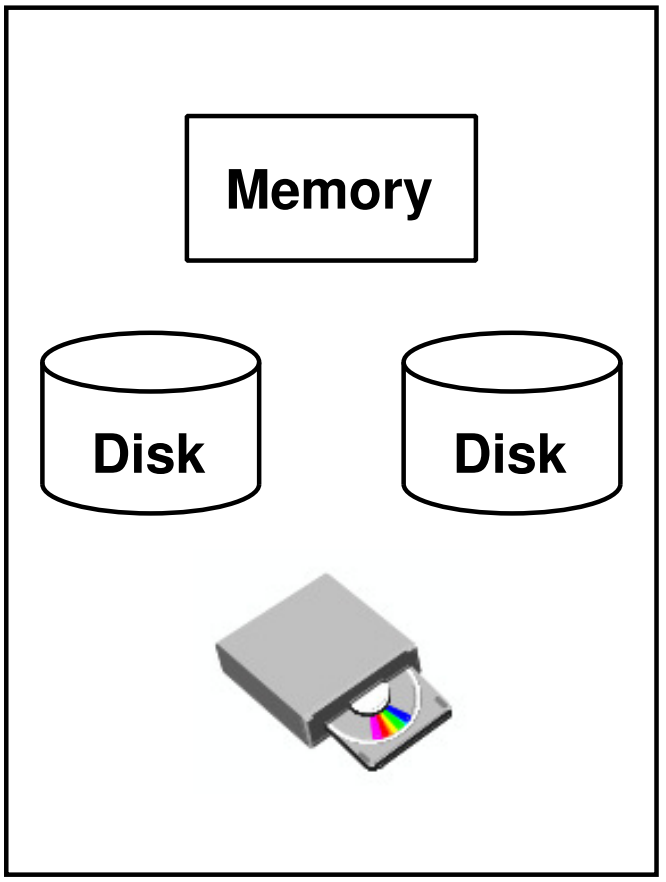
http://merlot.usc.edu/cs402-f13

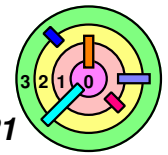


Files



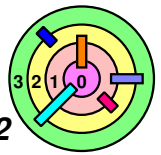






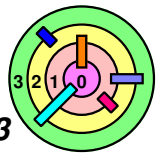
Requirements

- Permanent storage
  - resides on disk (or alternatives)
  - survives software and hardware crashes
    - (including loss of disk?)
- Quick, easy, and efficient
  - satisfies needs of most applications
    - how do applications use permanent storage?



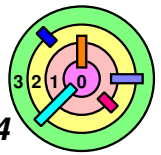
Applications

- Software development
  - text editors
  - linkers and loaders
  - source-code control
- Document processing
  - editing
  - browsing
- Web stuff
  - serving
  - browsing
- Program execution
  - paging



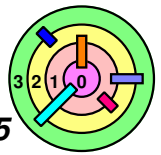
Needs

- Directories
  - convenient naming
  - fast lookup
- File access
  - sequential is very common!
  - "random access" is relatively rare



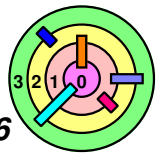
6.1 The Basics of File Systems

- *UNIX's S5FS*
- Disk Architecture
- Problems with S5FS
- Improving Performance
- Dynamic Inodes

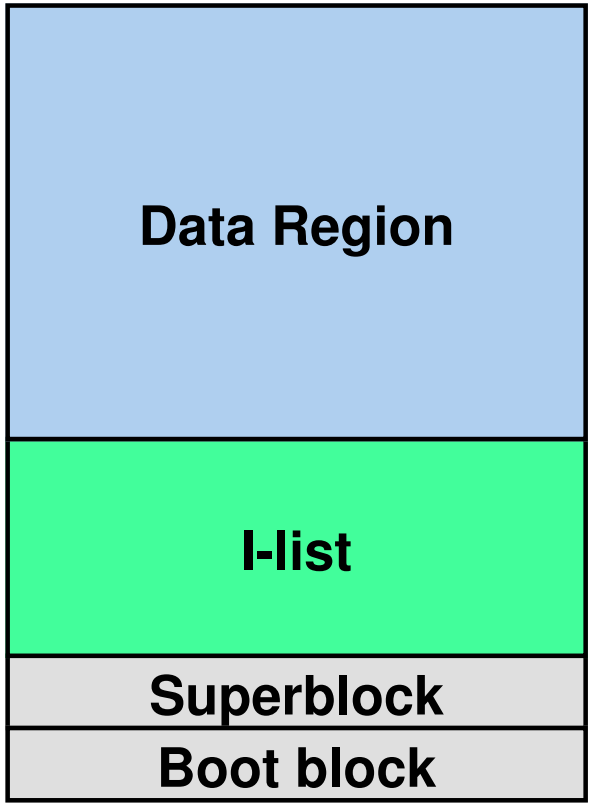


S5FS

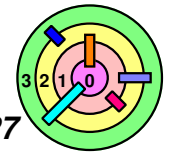
- A simple file system
  - slow
  - not terribly tolerant to crashes
  - reasonably efficient in space
    - no compression
- Concerns
  - on-disk data structures
    - file representation
    - free space



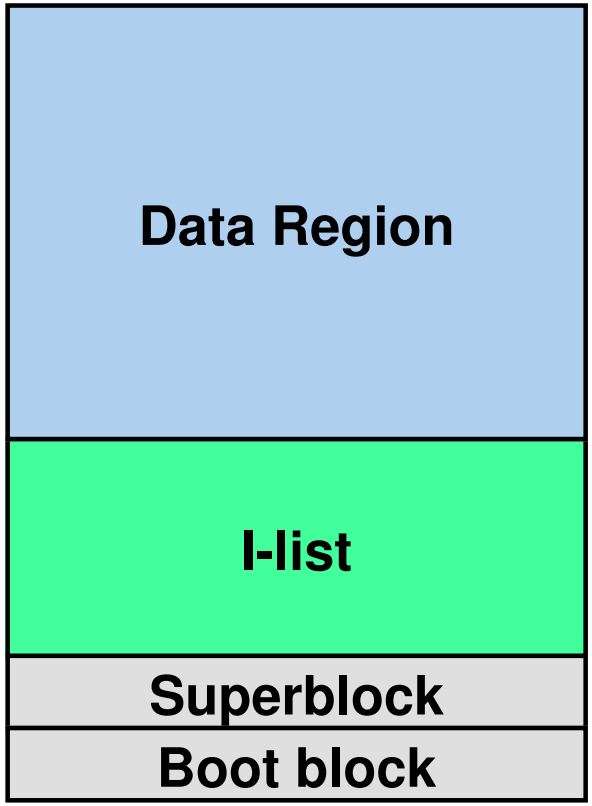
S5FS Layout



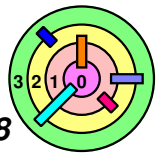
- A disk is simply an array of blocks of 1KB each (old Unix: 512B)
- A "linear view" (1-D array of blocks) of the disk



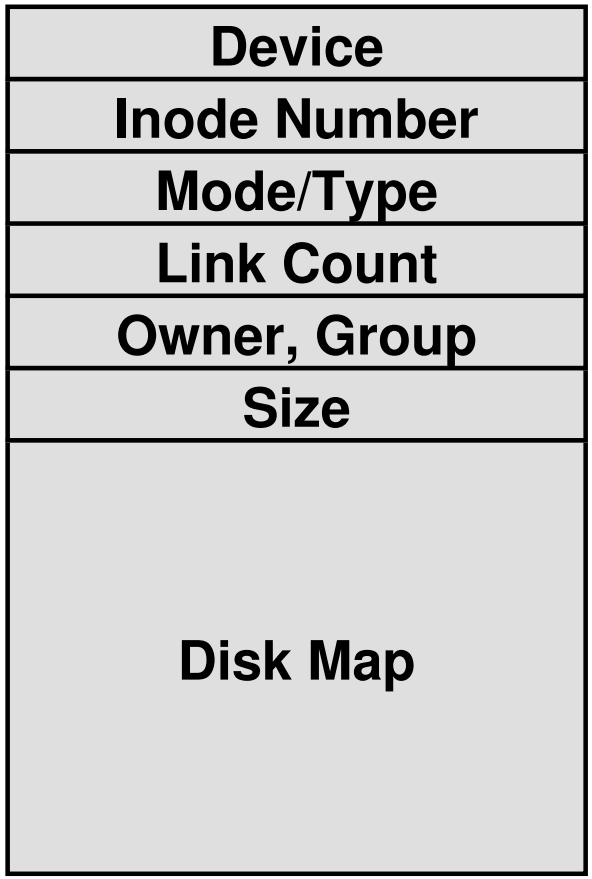
S5FS Layout

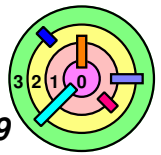


- The *superblock*
  - describes the layout of the rest of the file system
  - contains the *heads* of the *free lists*
- The *i-list* is an *array* of *index nodes (inodes)*
  - each representing a *file*



S5FS: Inode

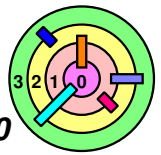




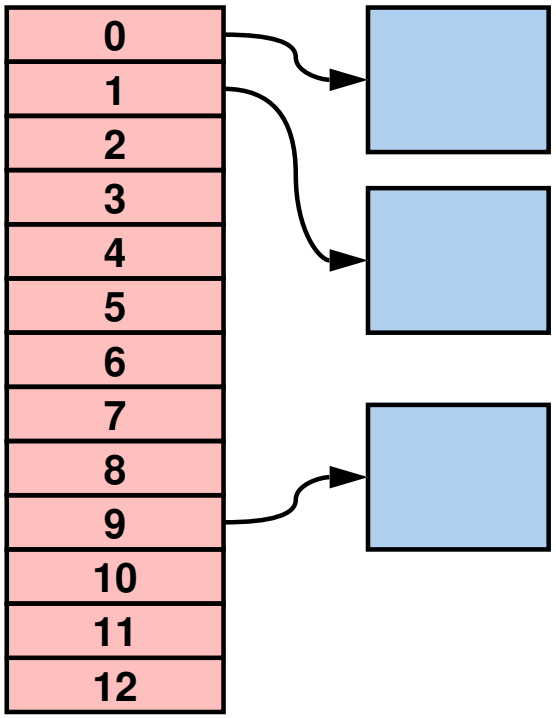
Disk Map

- assuming blocksize = 1KB

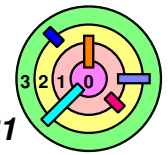
0
1
2
3
4
5
6
7
8
9
10
11
12



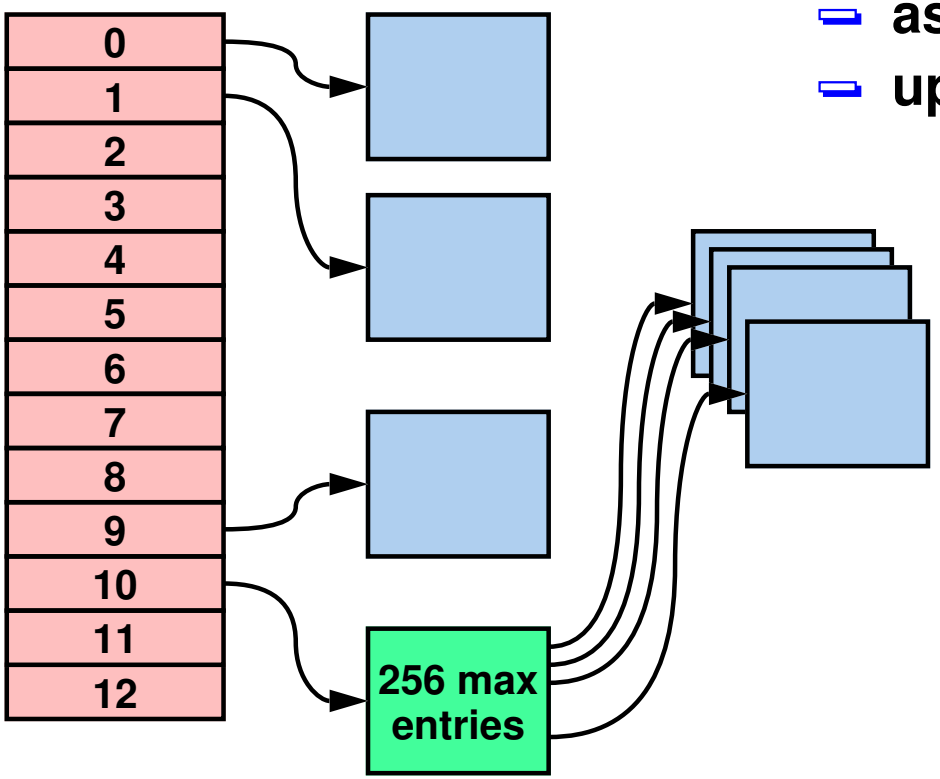
Disk Map



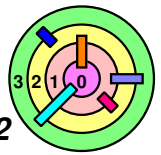
- assuming blocksize = 1KB
- up to 10KB



Disk Map

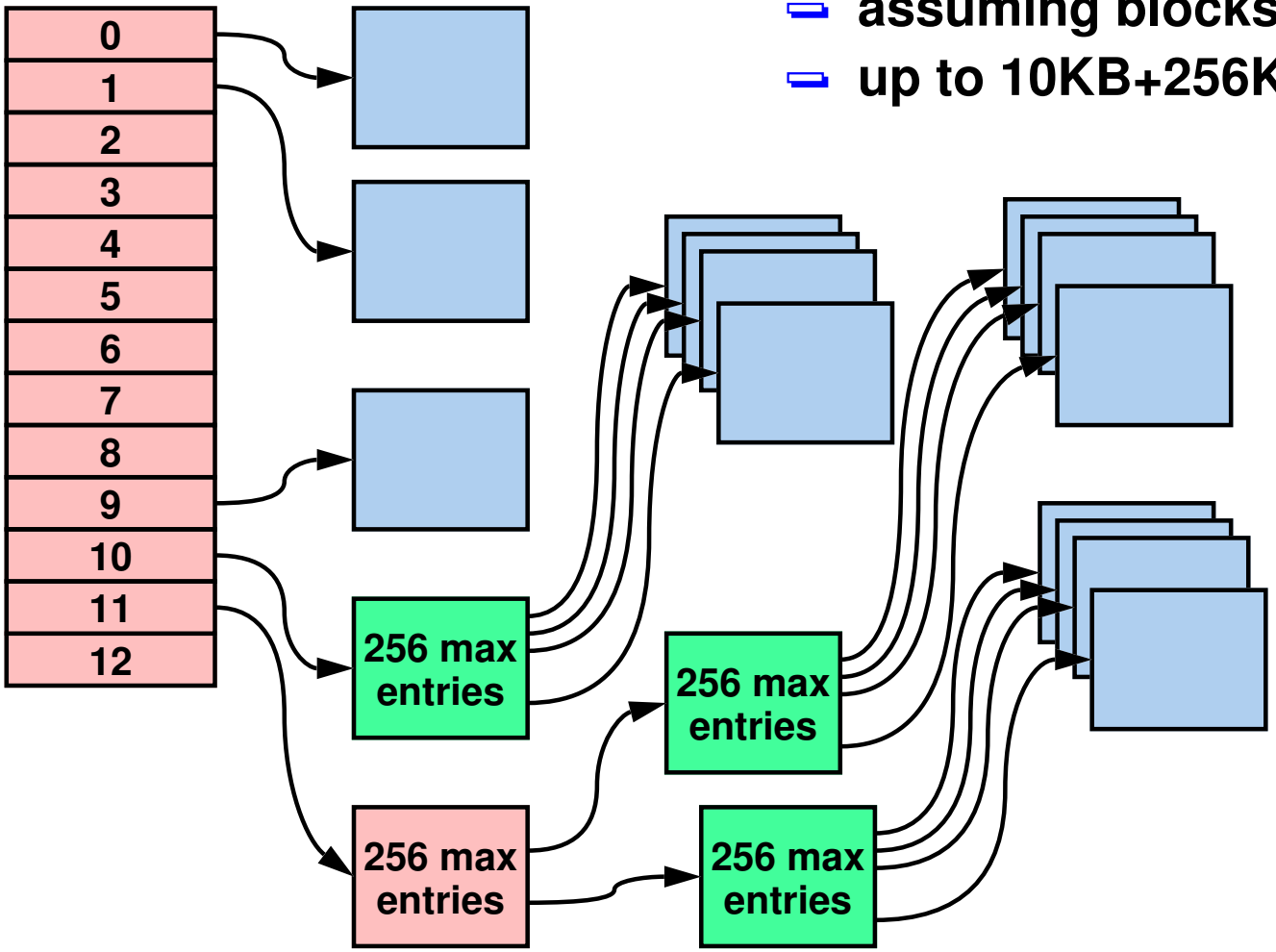


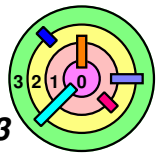
- assuming blocksize = 1KB
- up to 10KB+256KB



Disk Map

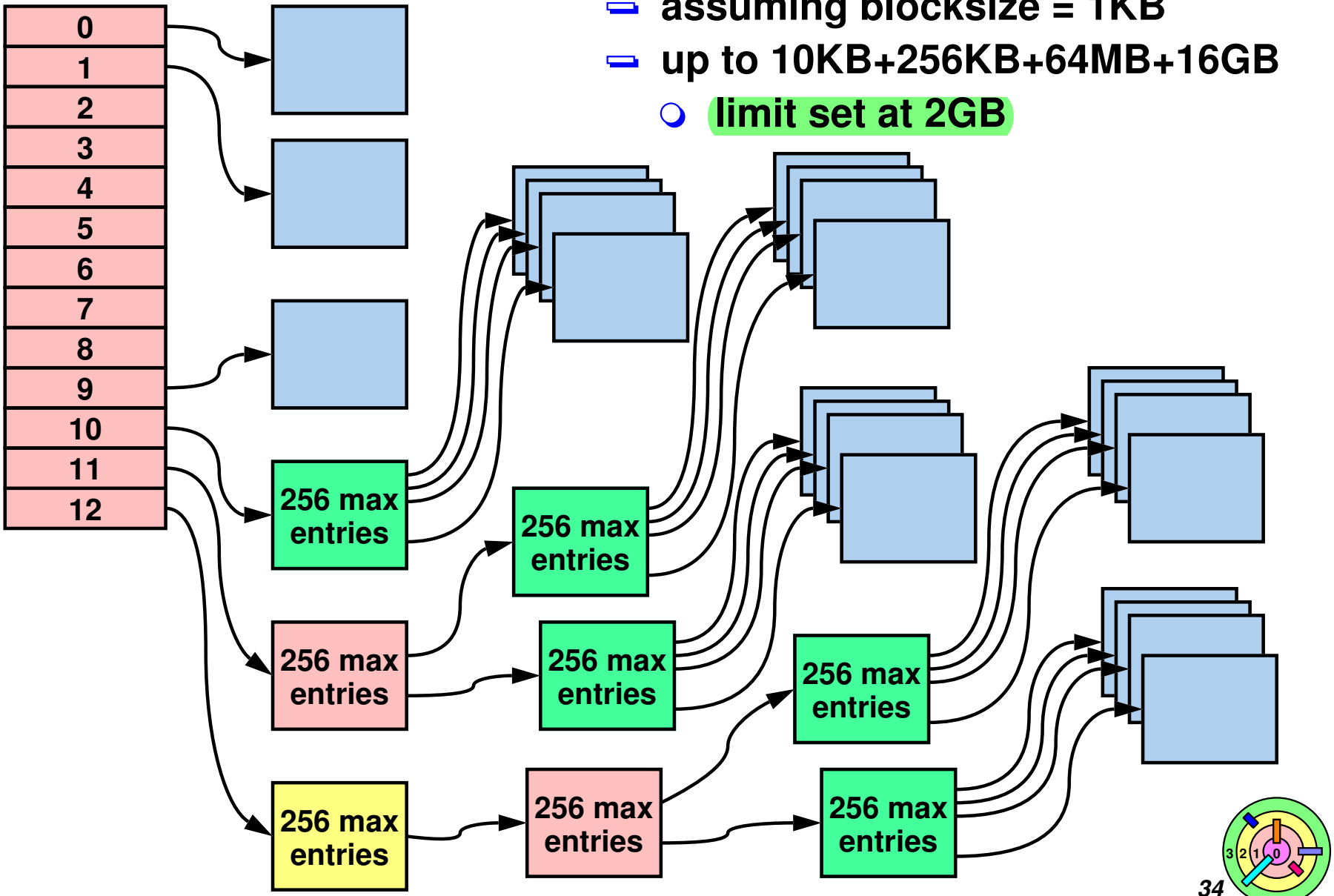
- assuming blocksize = 1KB
- up to 10KB+256KB+64MB



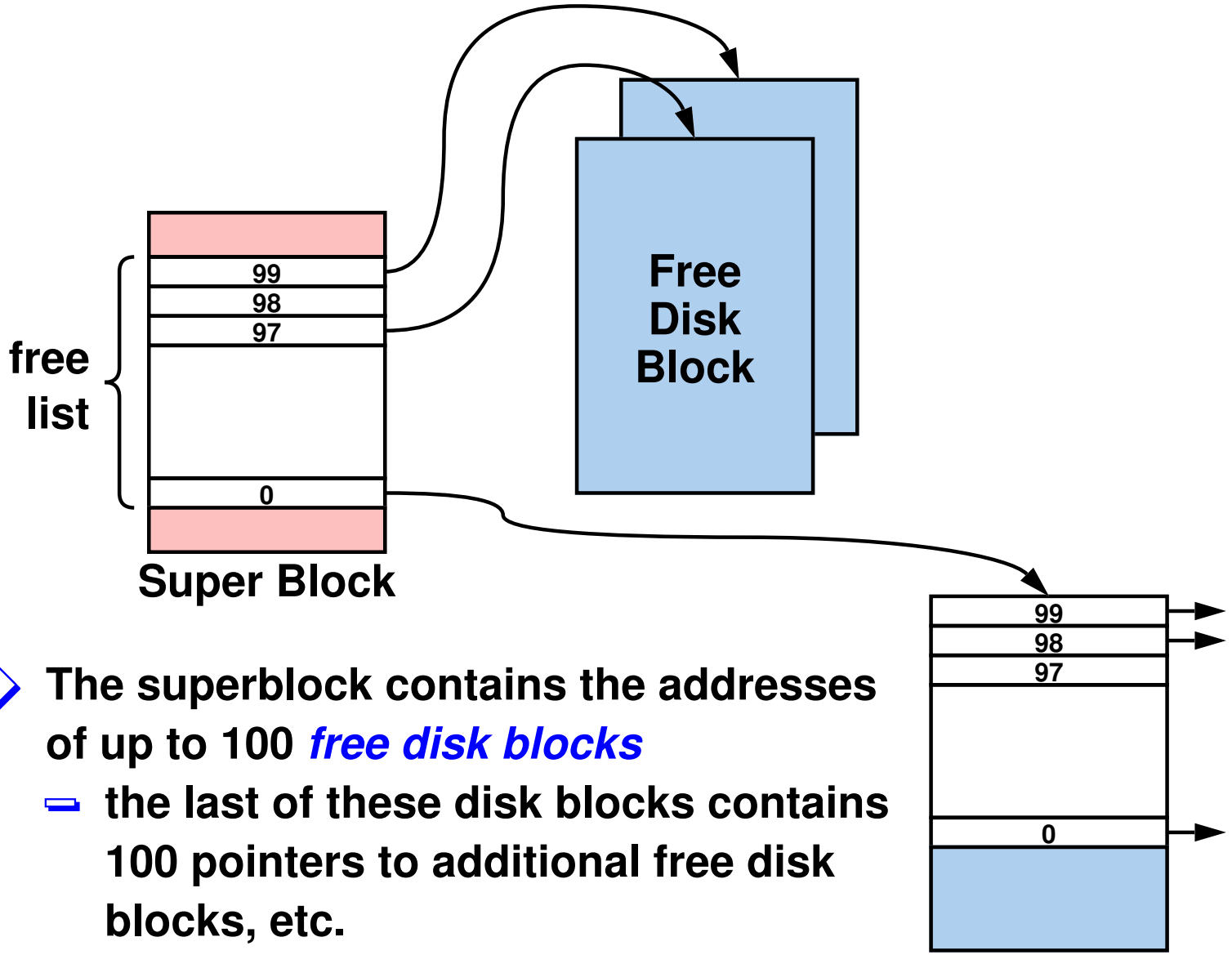


Disk Map

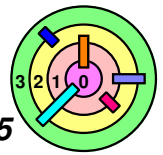
- assuming blocksize = 1KB
- up to 10KB+256KB+64MB+16GB
  - limit set at 2GB



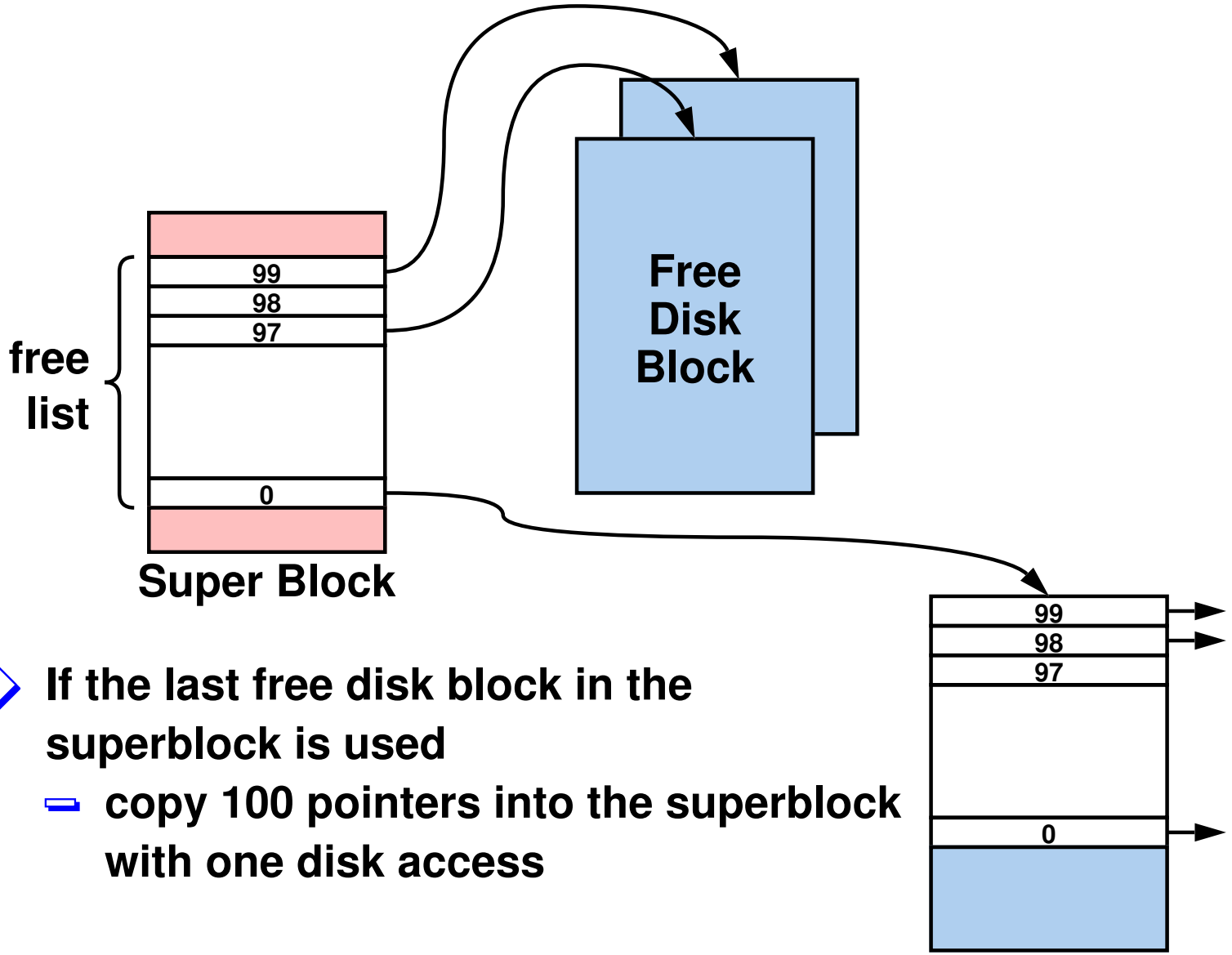
S5FS Free List



- The superblock contains the addresses of up to 100 *free disk blocks*
  - the last of these disk blocks contains 100 pointers to additional free disk blocks, etc.
  - can find *all* the free disk blocks this way

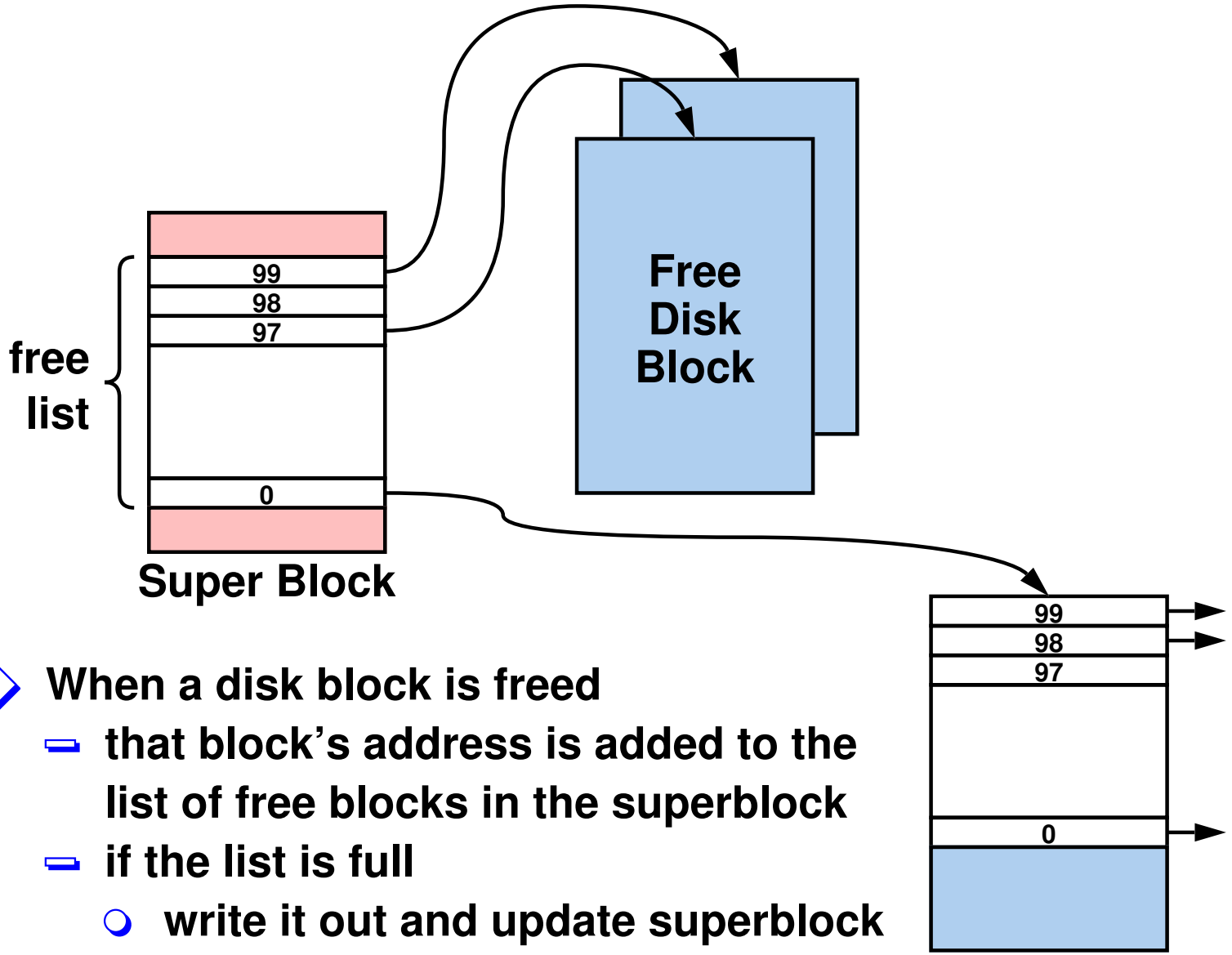


S5FS Free List

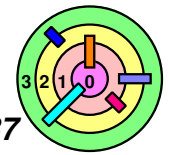


- If the last free disk block in the superblock is used
  - copy 100 pointers into the superblock with one disk access

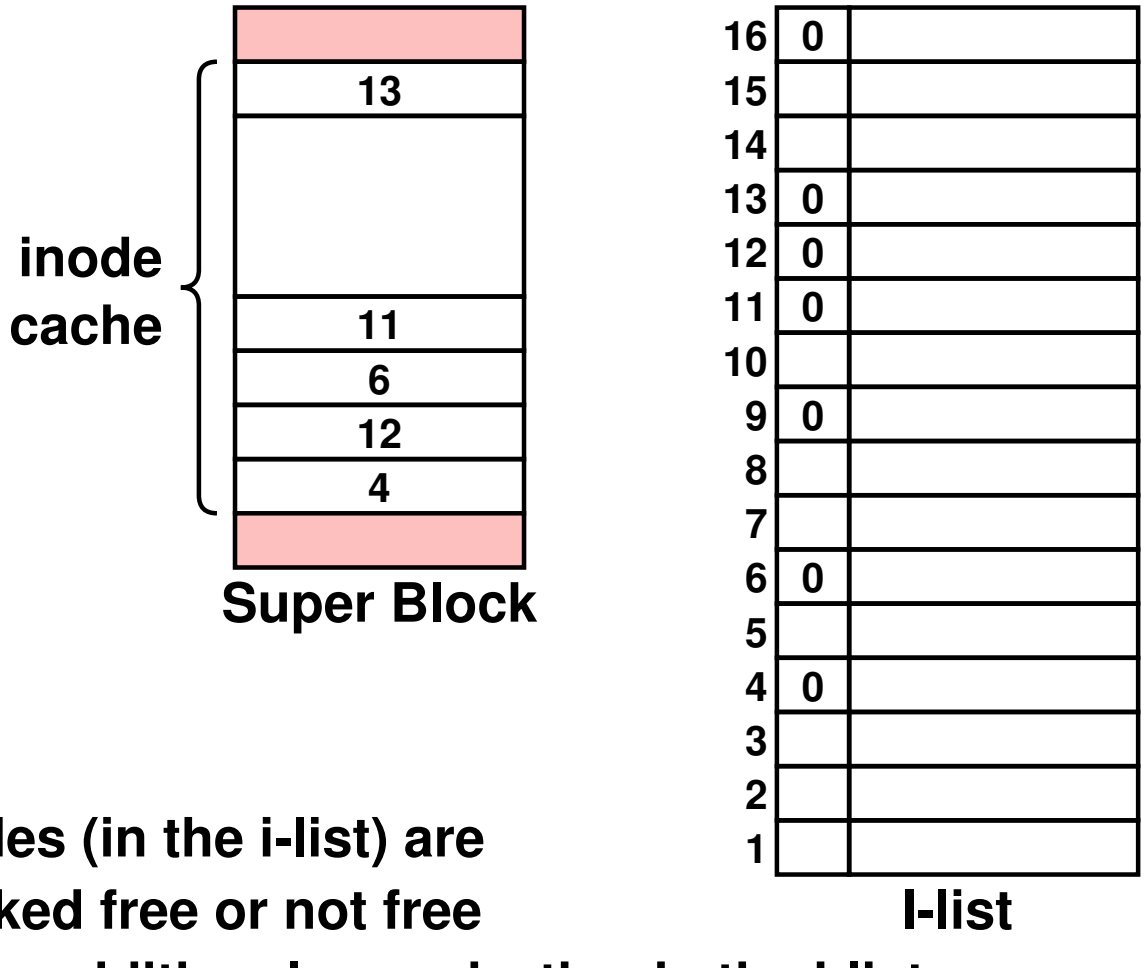
S5FS Free List



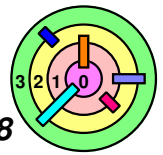
- When a disk block is freed
  - that block's address is added to the list of free blocks in the superblock
  - if the list is full
    - write it out and update superblock



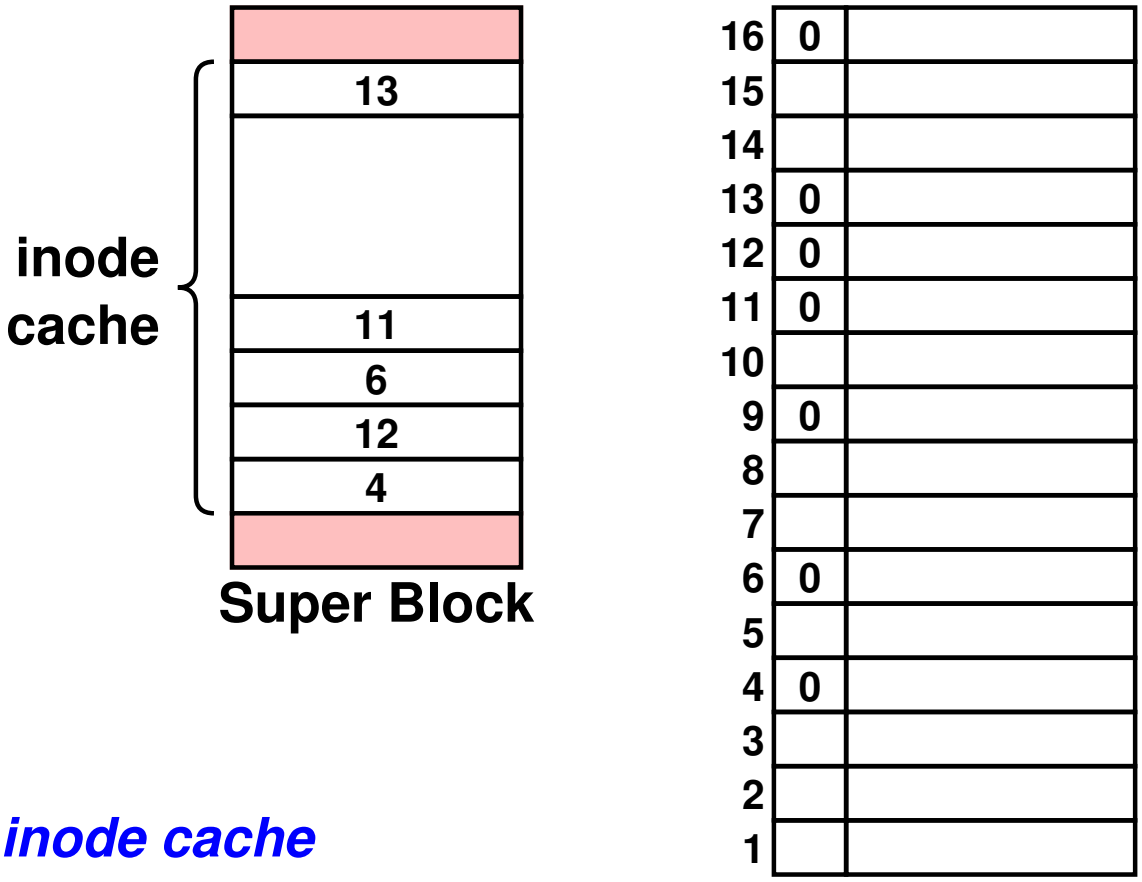
S5FS Free Inode List



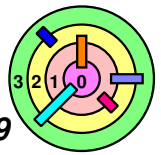
- Inodes (in the i-list) are marked free or not free
  - no additional organization in the i-list
  - the superblock *caches* free inodes *(*i.e., in the *inode cache)*



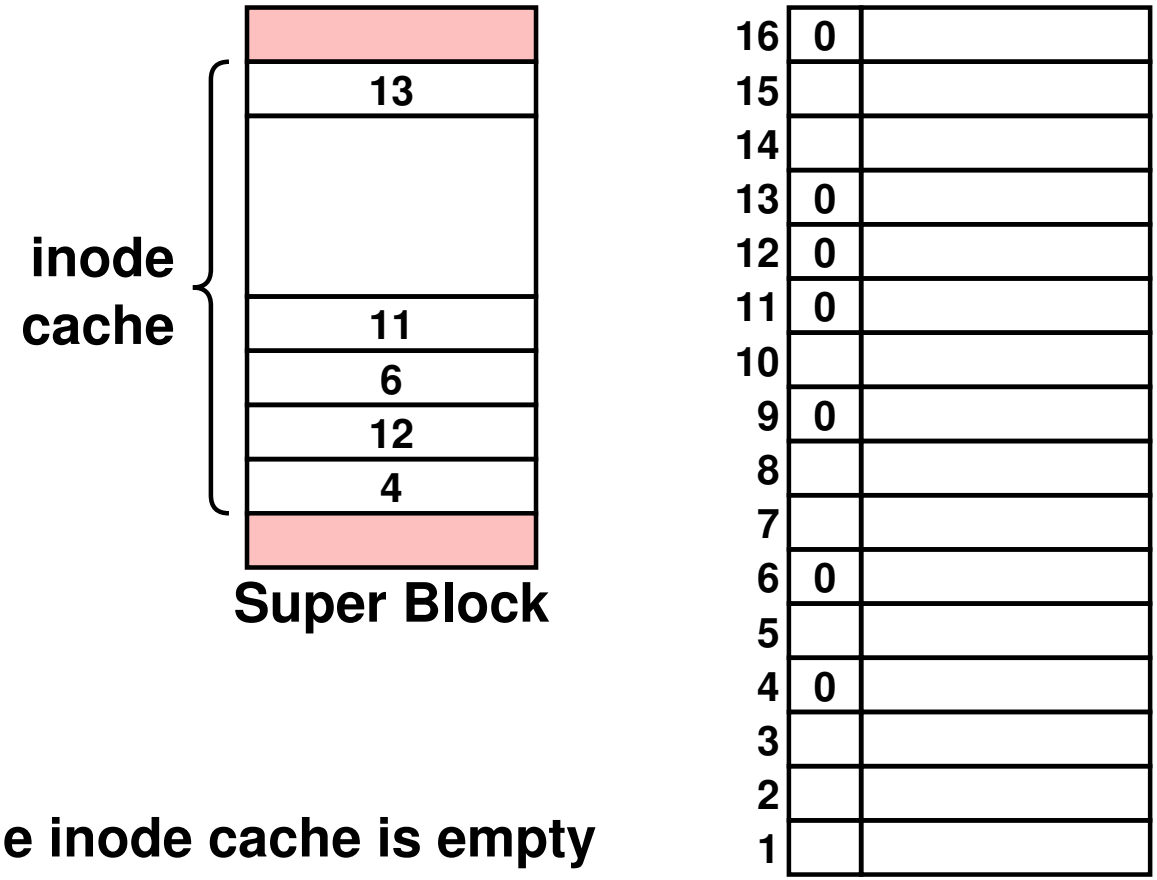
S5FS Free Inode List



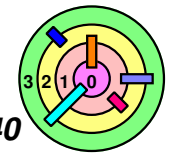
- The *inode cache*
  - to allocate an inode, simply mark it not free and remove it from the inode cache
  - to free an inode, simply mark it free and add to the inode cache if there is room



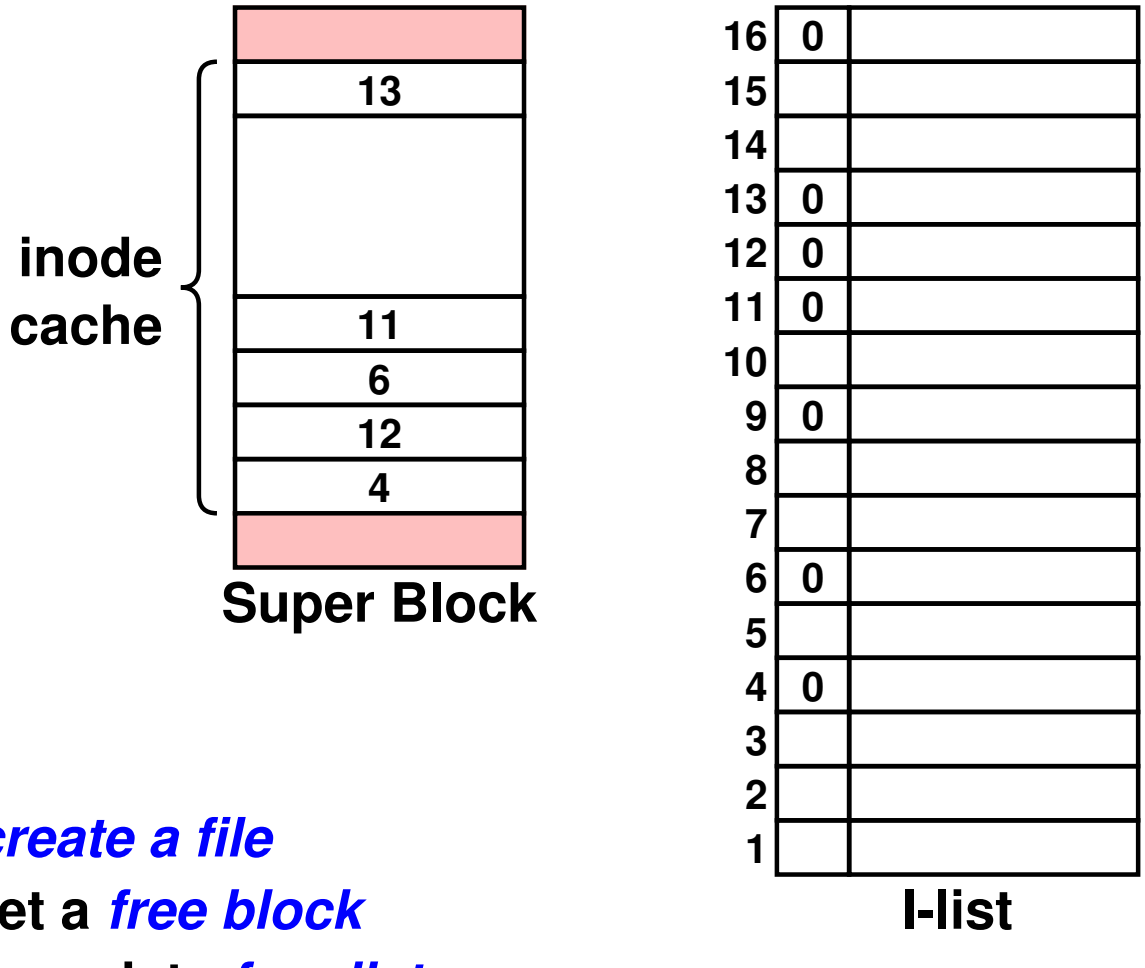
S5FS Free Inode List



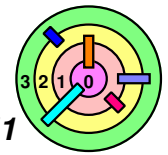
- If the inode cache is empty
  - scan the i-list to refill it
    - to help out with the scan, the inode cache contains the index of the first free inode in the i-list
      - need to maintain this entry when necessary



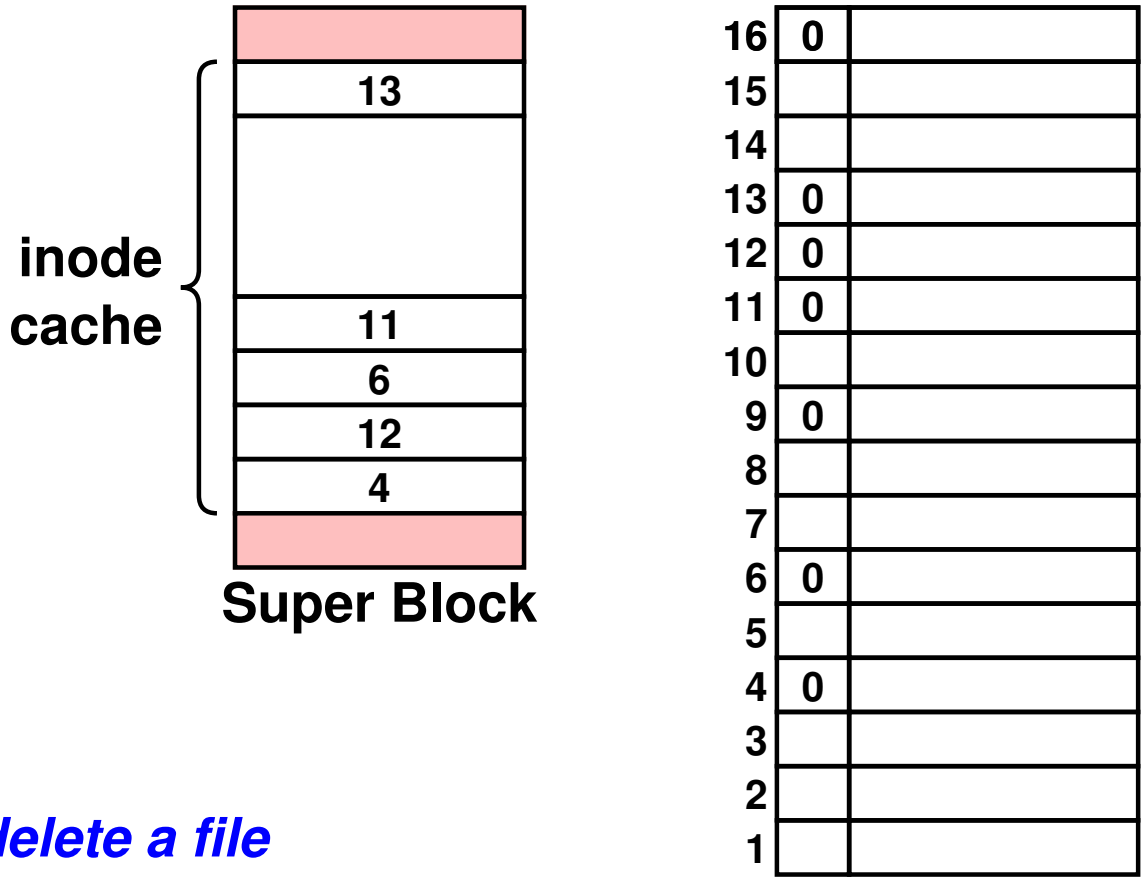
S5FS Free Inode List



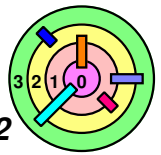
- To *create a file*
  - get a *free block*
    - update *free list*
  - get a *free inode*
    - update *i-list* and *inode cache*



S5FS Free Inode List

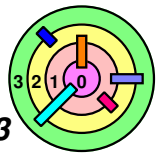


- To *delete a file*
  - add disk block(s) to *free list*
  - mark inode free in *i-list* and may be update *inode cache*



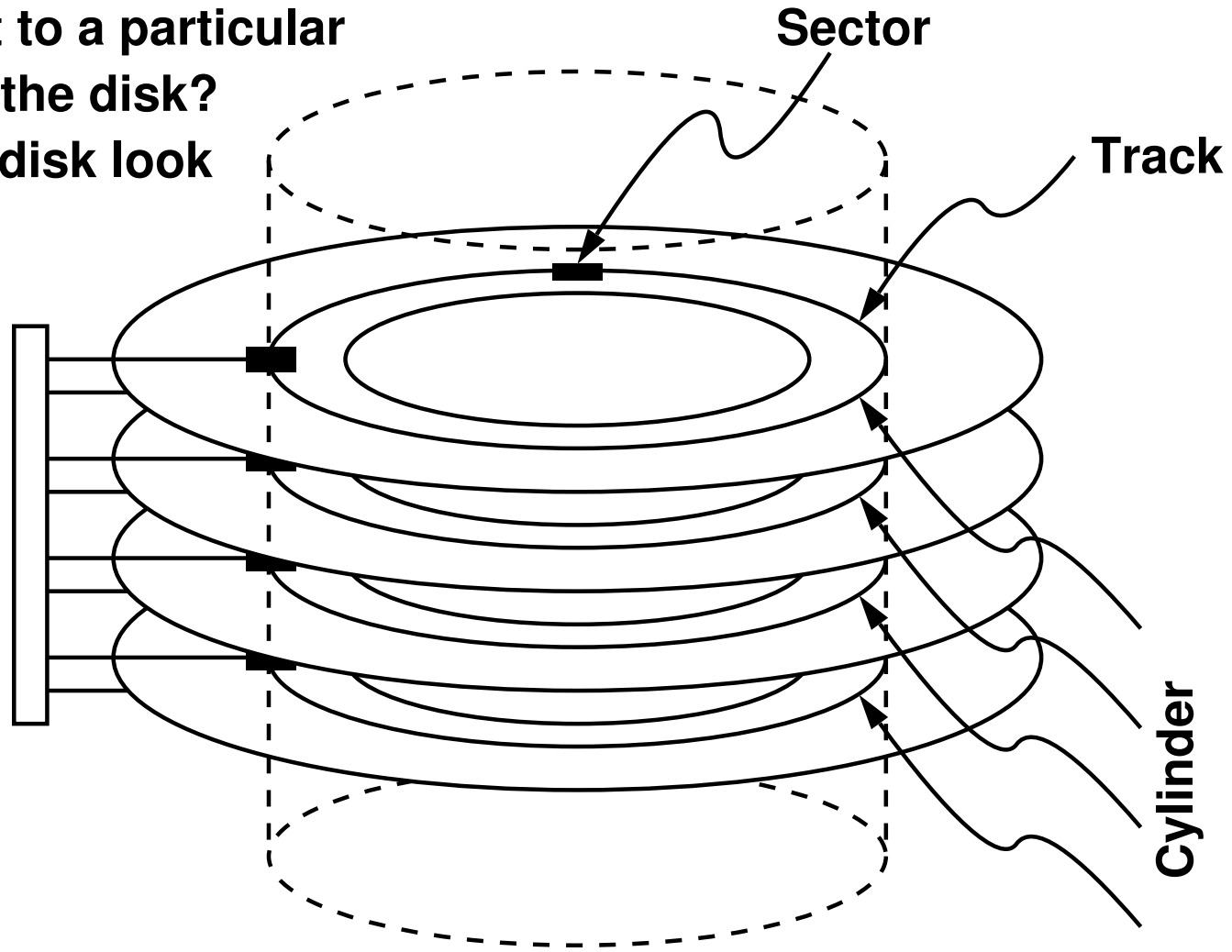
6.1 The Basics of File Systems

- UNIX's S5FS
- *Disk Architecture*
- Problems with S5FS
- Improving Performance
- Dynamic Inodes

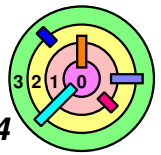


Disk Architecture

- How do you get to a particular byte of data on the disk?
  - what does a disk look like?



- Clearly, this looks nothing like the S5FS layout
  - or any logical file system layout



Rhinopias Disk Drive

Rotation speed	10,000 RPM
Number of surfaces	8
Sector size	512 bytes
Sectors/track	500-1000; 750 average
Tracks/surface	100,000
Storage capacity	307.2 billion bytes
Average seek time	4 milliseconds
One-track seek time	.2 milliseconds
Maximum seek time	10 milliseconds