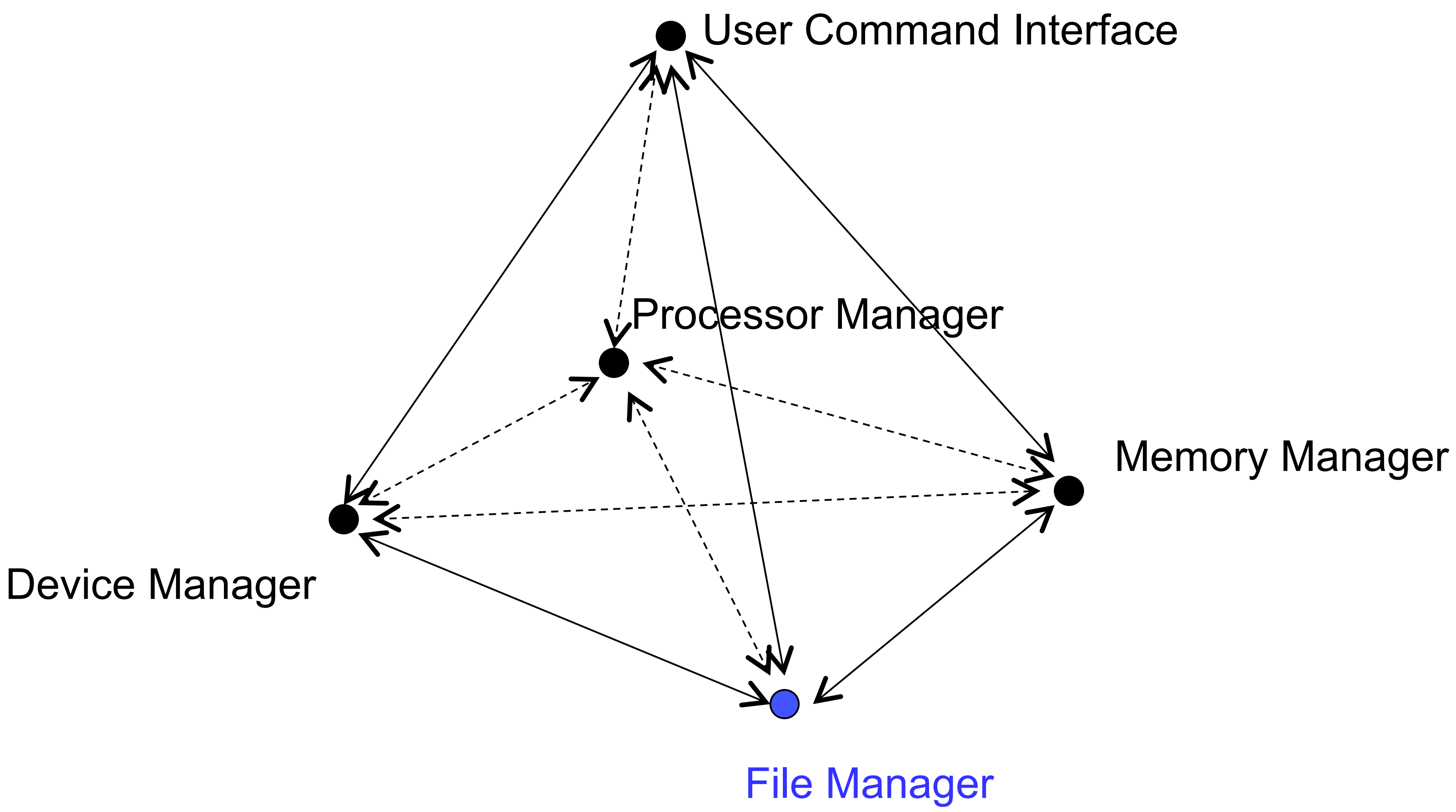
Comp 204: Computer Systems and Their Implementation

Lecture 17: Files and Filestore Allocation Policies

Today

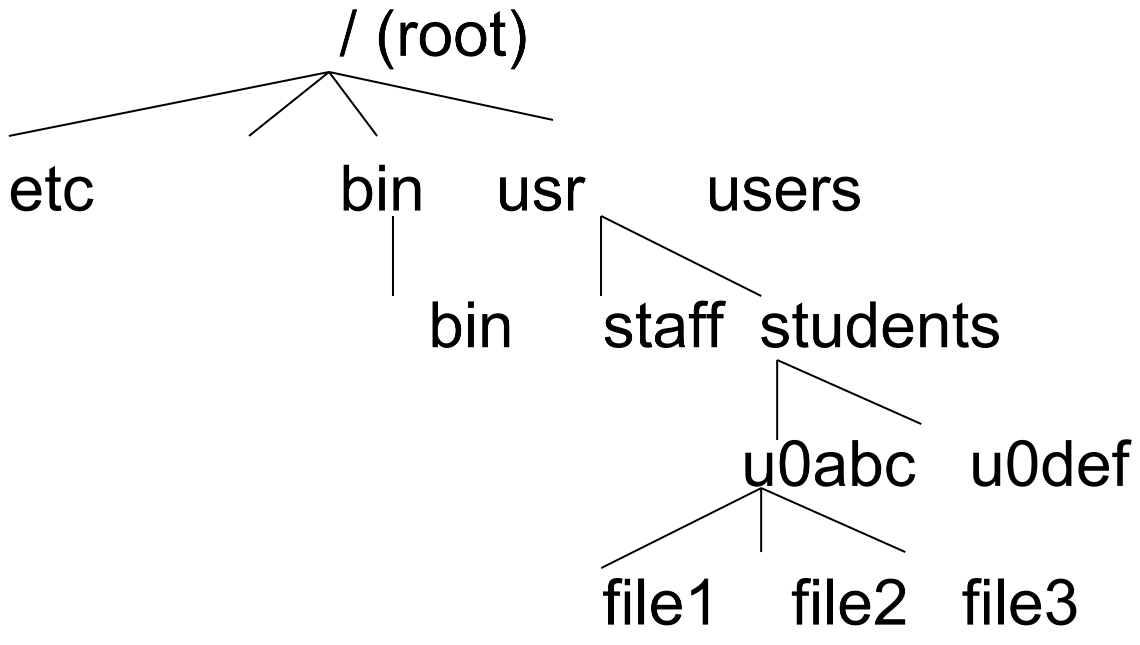
- Files
  - Introduction
  - Filestore allocation policies
    - Contiguous allocation
    - Linked allocation
    - File allocation table
    - Indexed allocation
  - Links
  - File deletion

Operating System – An Abstract View



Files & I/O

- Most filesystems are tree-structured (e.g. Unix)



Directories

- Non-leaf nodes are directories
  - contain list of filenames plus further info about each file
- Example: Unix
  - directory entry consists of filename and inode number
  - inode number references an inode - a file descriptor

Unix Inodes

- An inode contains info such as
  - file owner
  - permissions
  - modify and access times
  - size
  - type (regular, special, etc.)
  - location on disk

UNIX Permissions

- Permissions are read, write and execute (rwx) for user, group, others (ugo)
  - e.g. rwxr--r-- (rwx for owner, r for everyone else)
- Permissions can be altered with chmod
  - e.g. chmod go-r prog1

File Types

- Some systems (e.g. Windows) use file extension to indicate application
  - e.g. .doc, .ps, .ppt, .html
- Other systems more basic
- On UNIX, can try to execute any file
  - Exec will look for ‘magic number’ at head of valid executable binary file
  - If number not present, exec looks for “#!” followed by name of program to execute
    - e.g. #!/bin/ksh
  - Otherwise, assumes file is shell script and creates instance of user’s preferred shell to process it

Filestore Allocation

- Filestore divided into fixed-size blocks
- Blocks not currently allocated to files are maintained in a free list
- Several strategies for allocating blocks
  - contiguous
  - linked
  - file allocation table (FAT)
  - indexed

The Free List

- Can be a simple bit vector
  - e.g. 1100111100000…
    where 1=in use, 0=free
- Simple and efficient if kept in memory
- Problems:
  - Needs writing to disk every so often
  - May be huge for large disks
    - e.g. 80GB disk with 512-byte blocks needs 20MB for free list
- An alternative is to chain all free blocks into a linked list

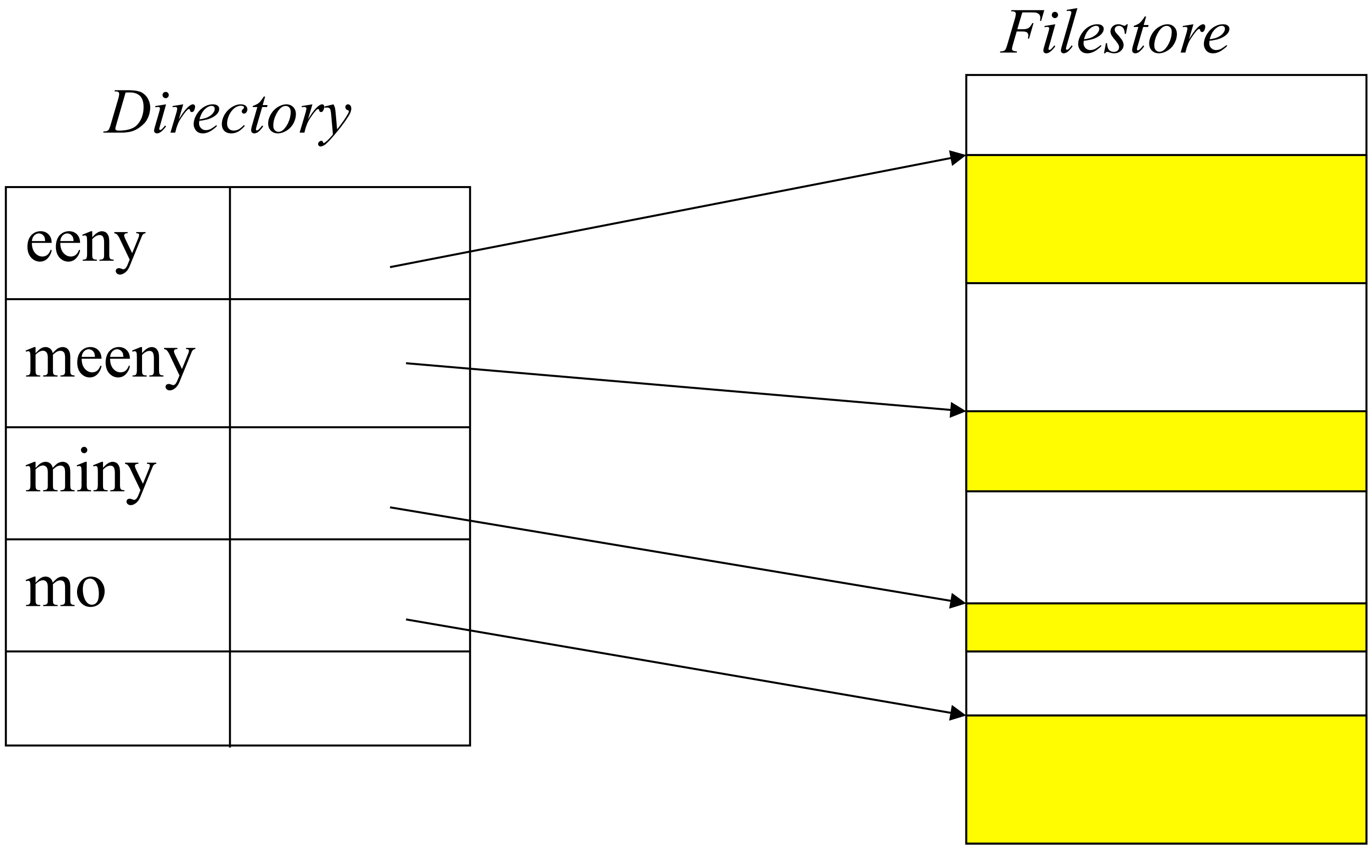
Exercise

A free list is represented as A2A0.

How many blocks are free?

Contiguous Allocation

- Each file allocated a contiguous set of blocks

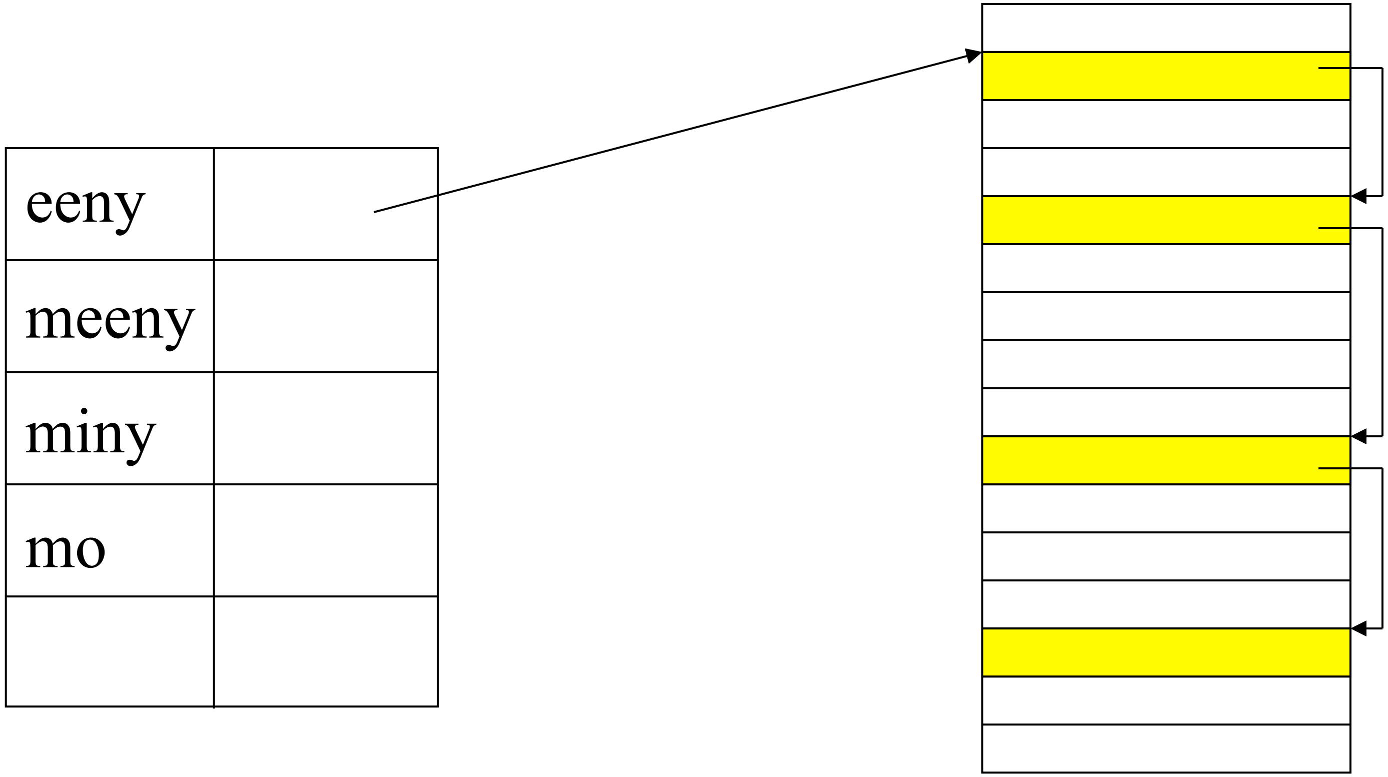


Contiguous Allocation

- Location information consists simply of *start block, no. of blocks*
- Advantages
  - fast, both for sequential and direct access
- Problems
  - fragmentation (c.f. linear memory)
    - may need regular compaction
  - number of blocks to allocate
  - file growth

Linked Allocation

- Each block contains a pointer to the next



Linked Allocation

- Advantages
  - easy to grow/shrink files
  - no fragmentation
- Problems
  - blocks widely dispersed
  - sequential access less efficient
  - direct access even worse
    - requires *n* reads to get to block *n*
  - danger of pointer corruption
- Can improve things by allocating blocks in clusters
  - but this worsens internal fragmentation

Question

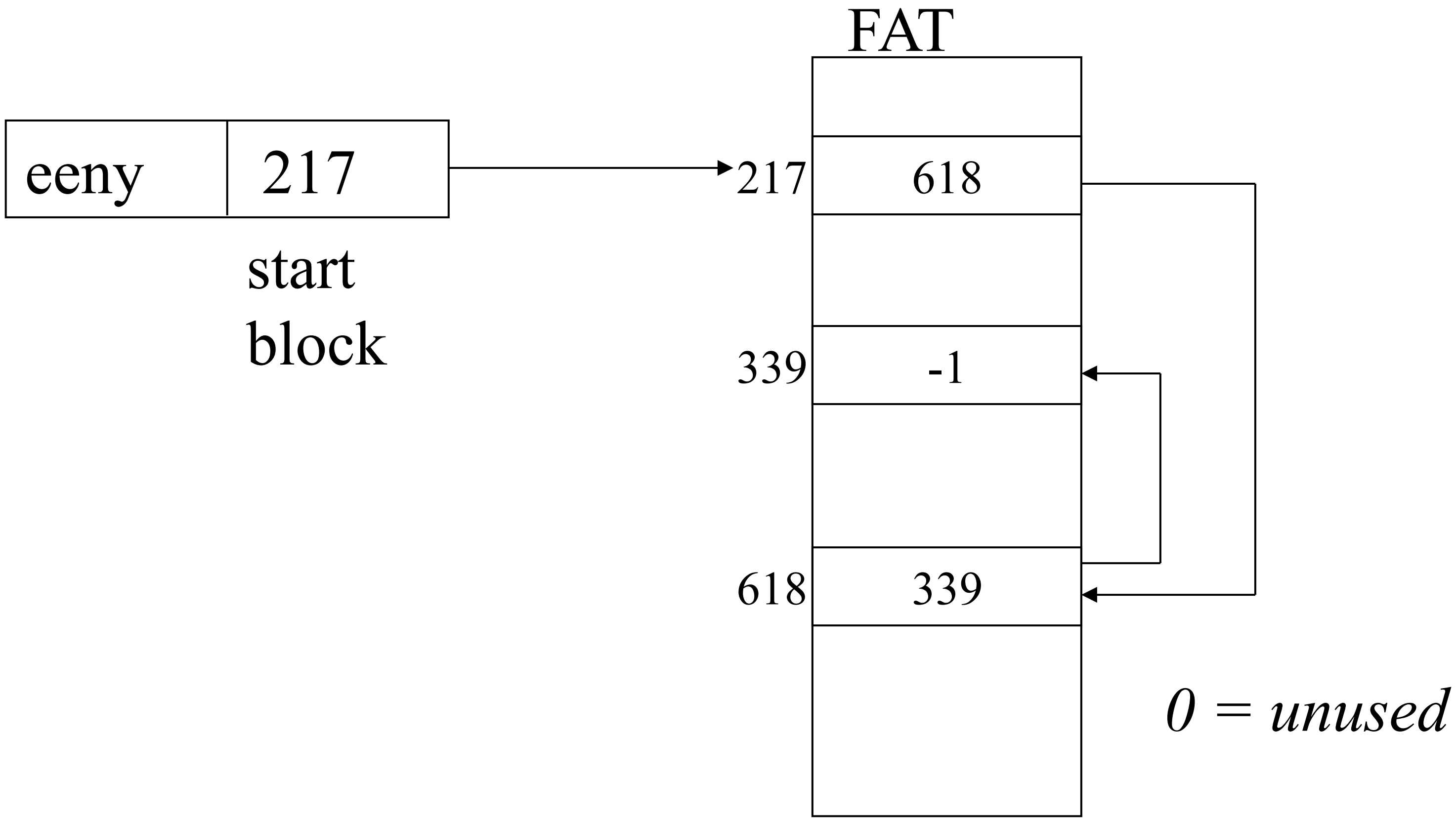
- Which of the following is true about linked filestore allocation versus contiguous allocation?

a) Linked is slower for both sequential and direct access.
b) Linked is faster for both sequential and direct access.
c) Linked is faster for sequential access, but slower for direct access.
d) Linked is slower for sequential access, but faster for direct access.
e) The performance for linked and contiguous is roughly the same for both forms of access.

Answer: a
Linked is slower for both sequential and direct access

File Allocation Table (FAT)

- Block allocations held in table located at start of partition

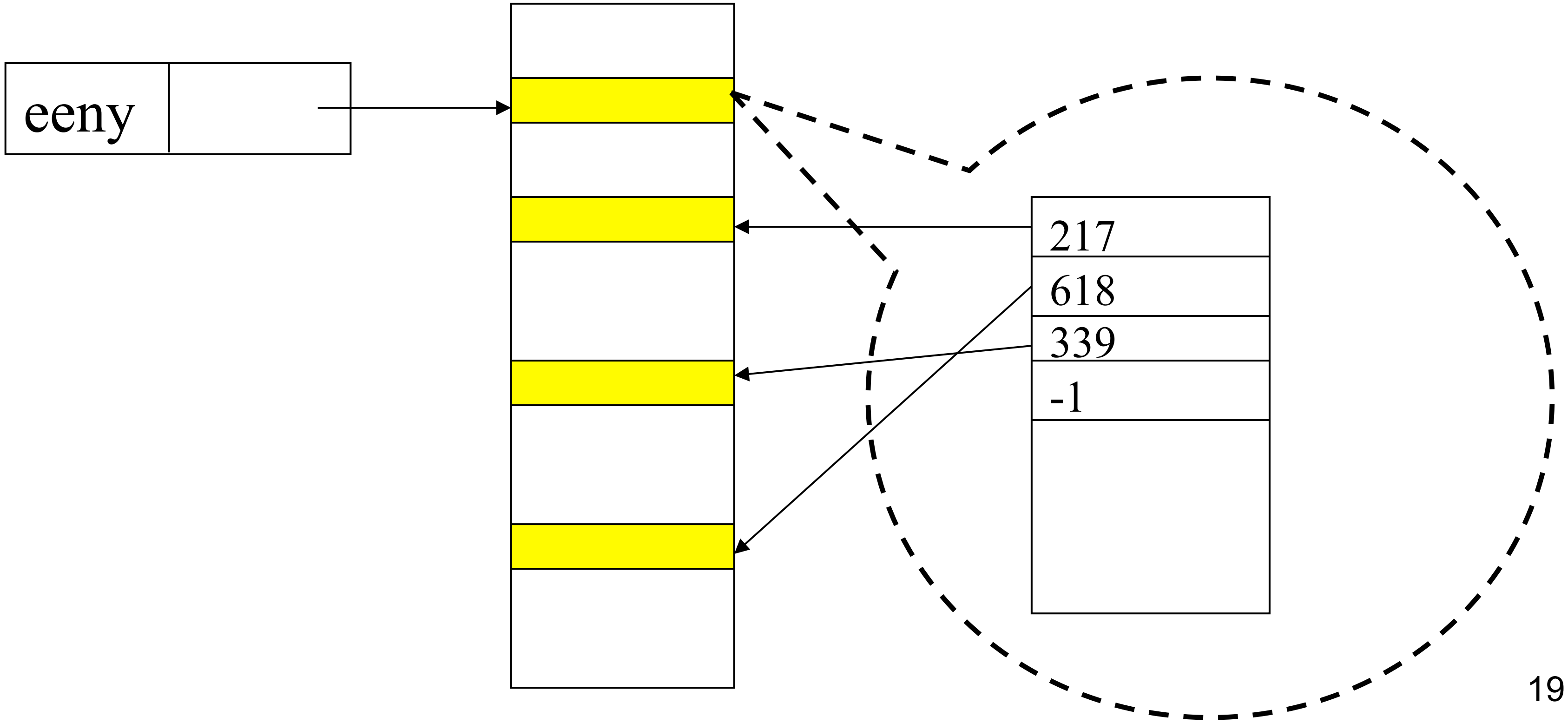


FAT

- Advantages
  - all pointer info held in one place
  - easier to protect
  - no need for separate free list
  - direct access much more efficient
- Problems
  - may require drive head to shift constantly between FAT and file blocks
  - FAT may become huge for large disks
    - clustering may help

Indexed Allocation

- First block holds index to all other blocks



Indexed Allocation

- Advantages
  - each file's pointer info held in one place
  - very efficient for both sequential and direct access
- Problems
  - blocks may still be widely dispersed
  - can run out of pointers for large files
    - may have to chain several index blocks together
- Example: Windows NTFS
  - Has a Master File Table (MFT) using a form of indexed allocation
  - However, small files contained entirely within MFT itself

Links

- It is possible for several filenames to point to the same file contents
  - e.g. Unix, the command

    ```
    $ ln /users/katie/public/prog.c myprog.c
    ```

    creates a new directory entry called myprog.c, with the same inode number as prog.c

File Deletion

- To delete a file:
  - remove directory entry
  - add allocated blocks to free list (garbage collection)

Question

- When should a deleted file ***not*** be garbage collected?

a) When there are multiple links to it.

b) When the file contains program code.

c) When there is only one copy of the file.

d) When the file is a system file rather than a user file.

e) When the file contains encrypted data.

Answer: a
When there are multiple links to it.

Log-Structured File Systems

- A system crash can cause loss of data and inconsistencies
- Log-structured file systems address issues of consistency checking
  - Based on database log-based recovery algorithms
- Each update to the file system is recorded as a transaction and all transactions are written to a log
  - When a transaction is written to the log it is considered to be committed
  - However, the file system may not yet be updated
- Transactions in the log are asynchronously written to the file system then removed from the log once the modification is complete
- If the system crashes all transactions that remain in the log must still be completed