Outline

- Practical, Transparent Operating System Support for Superpages Juan Navarro, Sitaram Iyer, Peter Druschel, and Alan Cox , In Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)

- Chapter 11: File System Interface

- Chapter 12: File System Implementation

- File System Trace Analysis
  - This project started out as a OS course project and ended in SOSP!!
- Distributed File system trace analysis

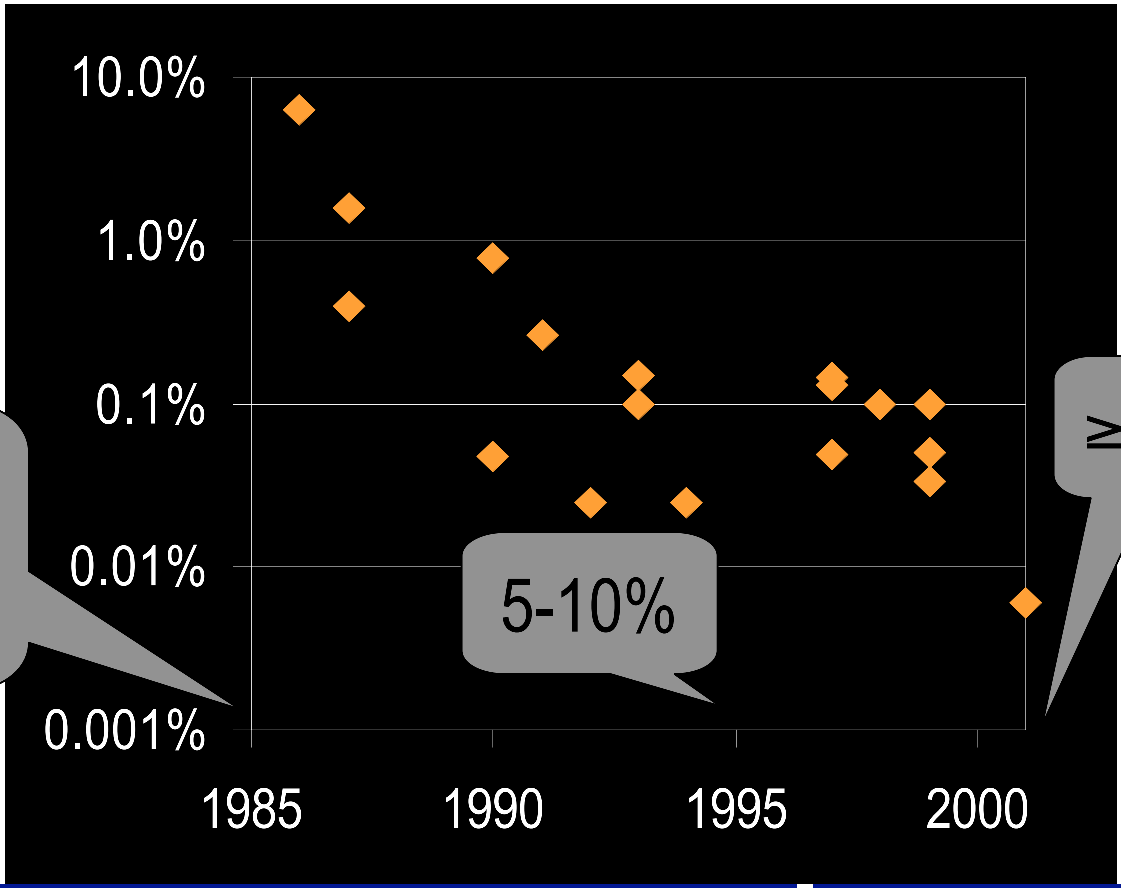
Superpages

- Increasing cost in TLB miss overhead
  - growing working sets
  - TLB size does not grow at same pace
- Processors now provide superpages
  - one TLB entry can map a large region
- OSs have been slow to harness them
  - no transparent superpage support for apps

TLB coverage trend

TLB coverage as percentage of main memory

Factor of 1000
decrease in
15 years



TLB miss overhead: □5%

5-10%

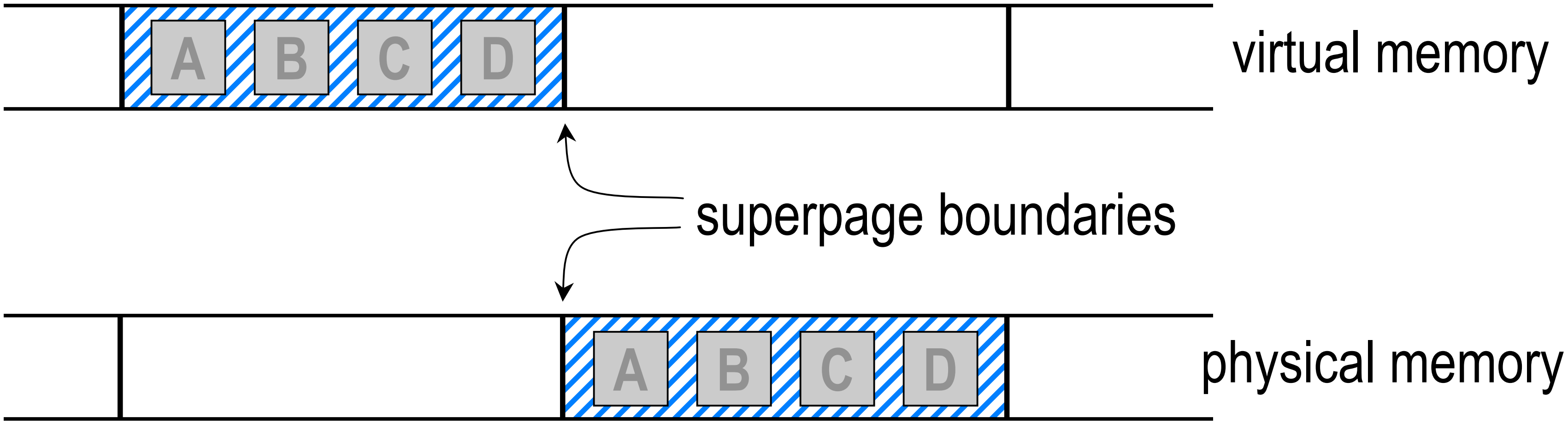
≥30%

Why multiple superpage sizes

- Improvements with only one superpage size vs. all sizes

	64KB	512KB	4MB	All
FFT	1%	0%	**55%**	55%
galgel	**28%**	**28%**	1%	29%
mcf	24%	31%	22%	**68%**

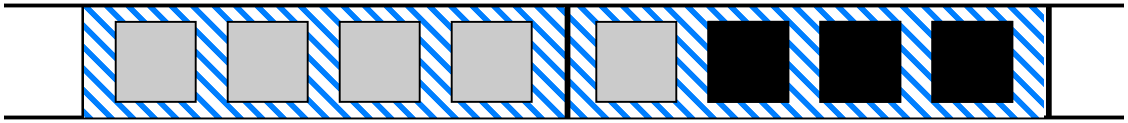
Issue 1: superpage allocation



- How / when / what size to allocate?

Issue 2: promotion

- Promotion: create a superpage out of a set of smaller pages
  - mark page table entry of each base page
- When to promote?



Create sm
May was

Forcibly populate pages?
May cause internal fragmentation.

to touch pages?
ortunity to increase
TLB coverage.

Issue 3: demotion

- Demotion: convert a superpage into smaller pages
- when page attributes of base pages of a superpage become non-uniform
- during partial pageouts

Issue 4: fragmentation

- Memory becomes fragmented due to
  - use of multiple page sizes
  - persistence of file cache pages
  - scattered wired (non-pageable) pages
- Contiguity: contended resource
- OS must
  - use contiguity restoration techniques
  - trade off impact of contiguity restoration against superpage benefits

File system interface

- File attributes
  - Name: only information kept in human-readable form
  - Type: Explicit or inferred
  - Location: pointer to file location on device.
  - Size: current file size
  - Protection: controls who can do reading, writing, executing
  - Time, date, user identification
- Operations: open, close, read, write, seek, append, delete, rename …
- Access: sequential, random, structured (indexed)
- Directory: Single, Two-level, Tree-structure, Acyclic
- Remote file systems: NFS, AFS, …

Consistency Semantics

- UFS: Writes to open file are visible immediately to other users that have this file open at the same time
- AFS: Writes to an open file by a user are not visible immediately to other users that have the same file open simultaneously

- Semantics depend on the cost of providing consistency vs scalability

Protection

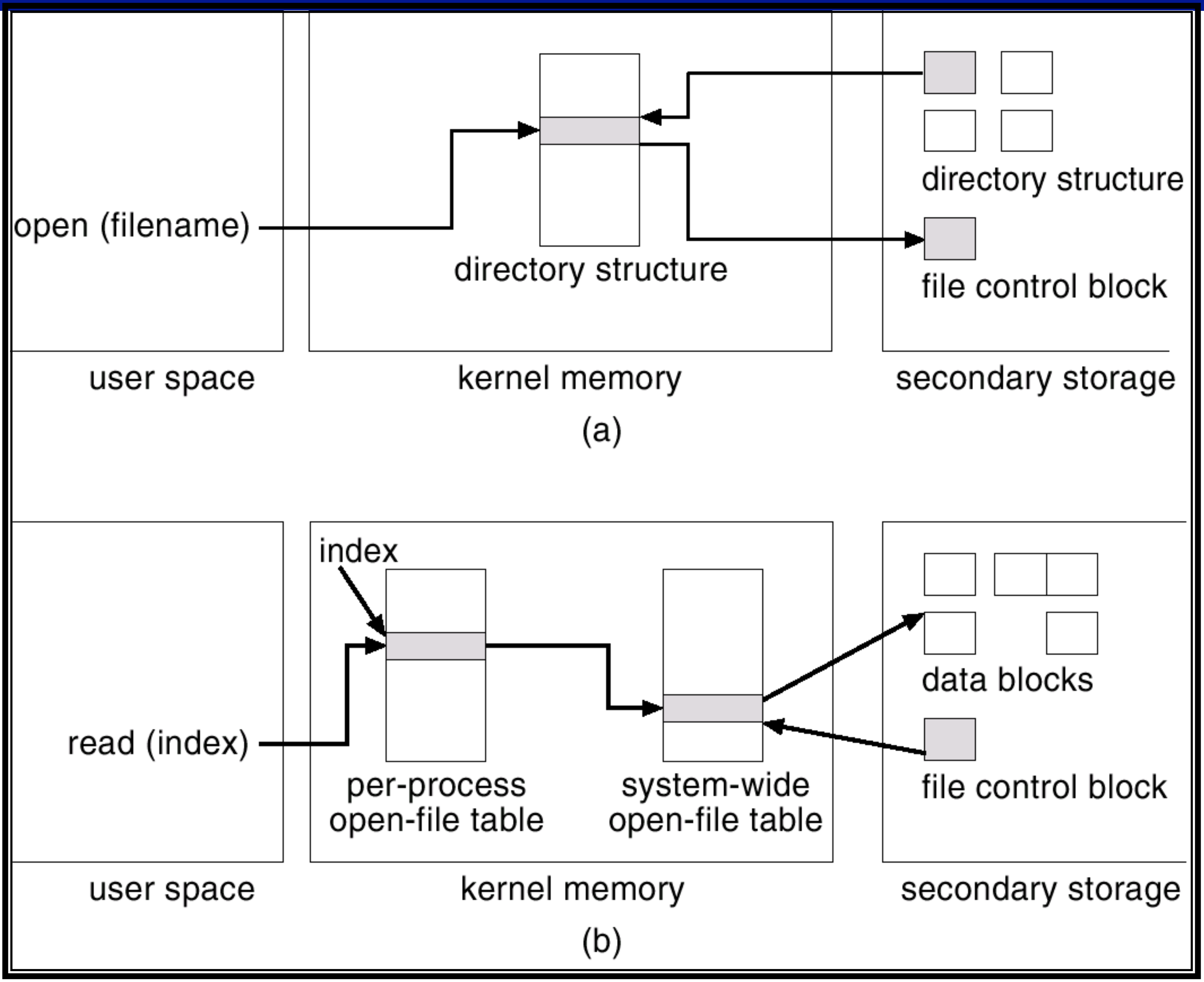
- Read, Write, Execute, Append, Delete, List etc using “owner, group, other” UNIX model or Access control lists (ACL) of NT, AFS

```
-rw-rw----    1 surendar mail      7384093 Sep 12 02:15 /var/mail/surendar
```

ACL:

```
Access list for . is
Normal rights:
  system:administrators rlidwka
  system:anyuser l
  surendar rlidwka
```

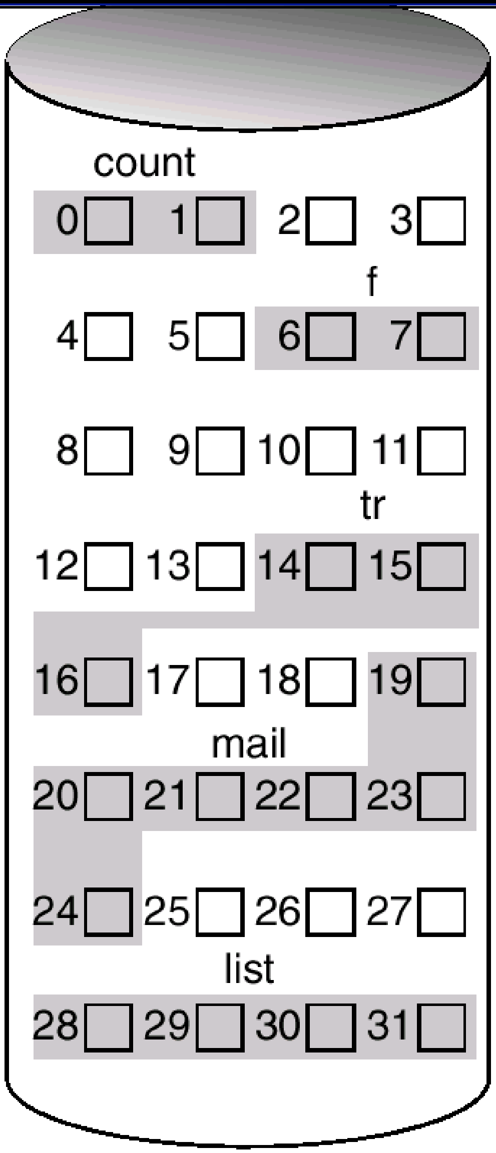
File system structure



File system implementation

- Virtual File Systems - easily change underlying storage mechanisms to local or remote
- Directory Implementation: Linear list, Hash table
- Allocation: Contiguous (fragmentation), linked allocation (FAT), Indexed Allocation (single level, multilevel)
- Free space management: Bit vector, Linked list, Grouping, Counting
- Recovery and consistency checking

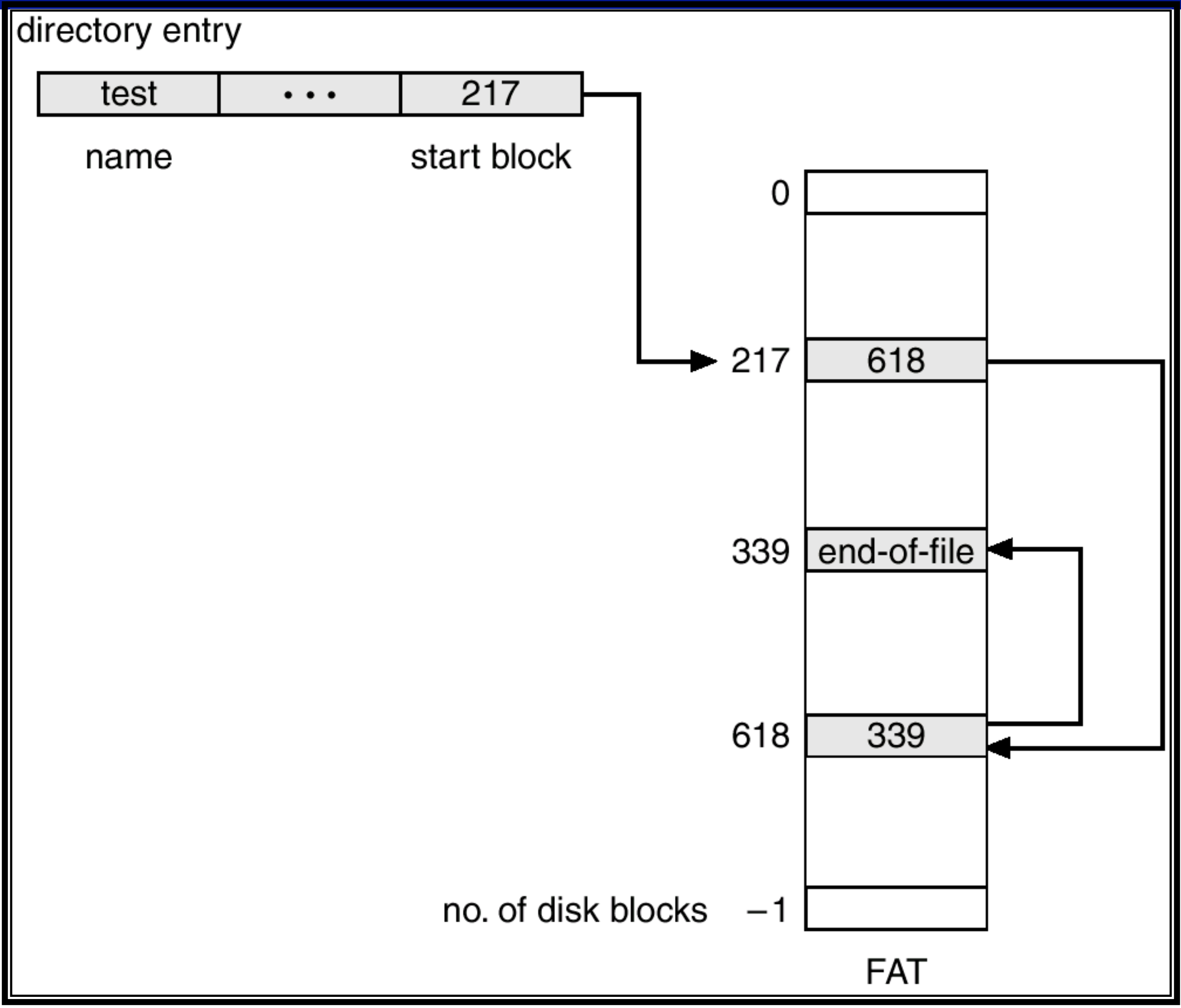
Contiguous allocation



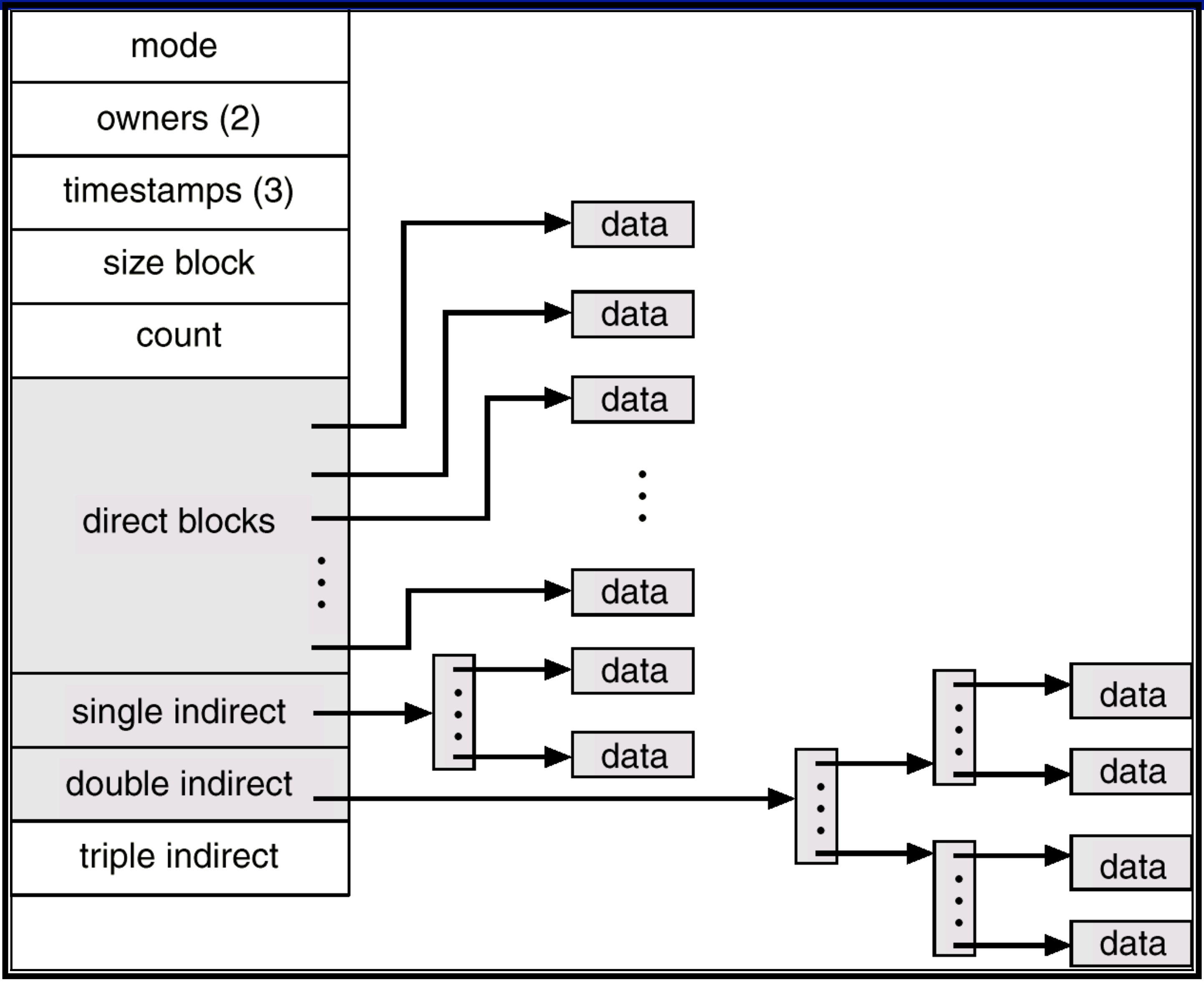
directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

File allocation table (FAT)



Unix - Indexed allocation



Backups

- Backup and restore: Towers of Hanoi style levels
  - Full+Incremental backups
  - Level 0 - full
  - level n takes all changes since last, lower numbered level
  - E.g. 0, 5, 6, 3, 5, 6
  - Full, 5-0, 6-5, 3-0, 5-3, 6-5
- Log structured file system
  - Provide transactional guarantees for critical meta-data
- Pathname translation:
  - Multilevel directories in order from the root
  - Caching to improve performance
  - /usr/local/bin/ls would be /, /usr, /usr/local, /usr/local/bin ..

LFS

- **Log structured** (or journaling) file systems record each update to the file system as a **transaction**.

- All transactions are written to a **log**. A transaction is considered **committed** once it is written to the log. However, the file system may not yet be updated.

- The transactions in the log are asynchronously written to the file system. When the file system is modified, the transaction is removed from the log.

- If the file system crashes, all remaining transactions in the log must still be performed.

Trace driven analysis of the UNIX file system

- Rather old, but seminal. Influenced much of file system design for a long time

- Studies like these are extremely important to understand how typical users are using a file system so that you can tune for performance

- The key is to trace the “typical user population”.
  - Academics do not have access to commercial work loads
  - Chicken and Egg syndrome: Users perform certain tasks because current systems perform poorly.
    - E.g. users may backup their work into a separate file every so often because of poor consistency guarantees.
    - UNIX vi editor saves files by deleting old file, creating a new file with the same name, writing all the data and then closing. If the system crashes after creating and write, before close, data is left in buffers which are lost, leading to a 0 byte file. It happened a lot and so programs create backup files often.

Important conclusions

- Most files are small; whole file transfer and open for short intervals. Most files are short lived. Caching really works.
- UNIX used files as intermediate data transfer mechanisms:
  - E.g. compiler
    - Preprocessor reads .c file -> .i file
    - CC1 reads .i -> .asm file and deletes .i file
    - Assembler reads .asm -> .o file and deletes .asm file
    - Linker reads .o -> executable and deletes .o file
- One solution: Make /tmp an in-memory file system

```
df /tmp
Filesystem          1k-blocks      Used Available Use% Mounted on
swap                 1274544      1776   1272768   1% /tmp
```

Most files are read sequentially

- UNIX provides no support for structured files
- Applications that provide structured access (data bases) use raw file interface and by-pass operating systems
- Solution:
  - Read-ahead to improve performance

Most file accesses are to the same directory

- UNIX has a hierarchical file system structure
- Typical academic users compile, word process from one directory and so all accesses are to a single directory
- File systems such as Coda, AFS have notions of volumes, cells that capture this
- Does this apply to Windows?

Most files are small

- On a departmental machine with 8 MB of main memory, what else do you expect

- Is it true now with our Netscape, xemacs, IE, Power point etc?

- Relatively, it may still be true. On a 60 GB hard disk, 1 MB file may be “small”