Disk Allocation

> Access patterns, allocation +/-, file meta-data caching

> Two basic diffs from mem mgmt
correctness: crashes   Performance: disk's nature

Access Patterns

> Sequential access
file read/written in order
very common
⇒ much knowledge to OS (optimization)
prediction

> file: name ≥ (abstraction) array of bytes (built on blocks)
[RMW]

> meta-data: stuff about file
must also be on disk

> directories: organize files
just a special file!
data:

"a"	3

inode #

> Random access
address some arbitrary block in file
harder to predict patterns

> Keyed Index (Associative)
return data assoc. w/ key value – Database
"higher level"
Usually not in OS

Workloads

> File Characteristics
{ Most files small
{ most of disk allocated to large files

⇒ keep per-file overheads low
good bandwidth for large files

> Common operations
{ read file ⇒ read i-node too
{ access files in same directory @ same time }
(e.g. make)

Free Space Mgmt

> Bit Maps
  array of bits, one per block
  perm. on disk, but keep in mem too (cache) ?

> Try to "co-allocate" "related" blocks
  when empty, easy
  when full, ⇒ time spent searching

> Partial Solution
  always keep some % of disk free
  (not available to users)

Strategies

> Progression
  simple ⇒ . . .

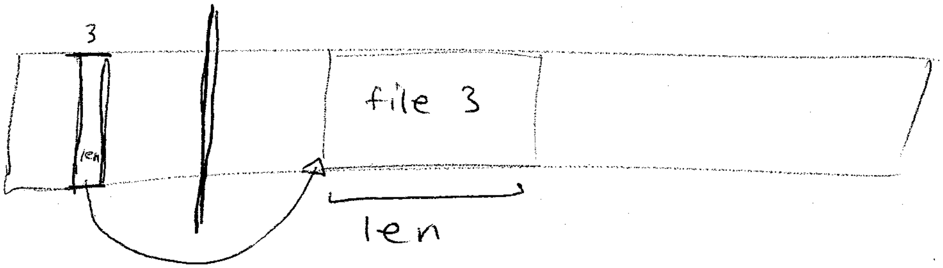
> keep in mind
  fragmentation } space/time
  ability to extend file] functionality
  access time } performance
  space

Time
  few seeks
Space
  overheads of structures
  fragmentation
Functionality
  extend file . . .

Contiguous

Allocate files like segmented memory (base + length)

> specify length @ creation time
> find space by examining bit-map (first, best fit)
> ~~inode~~/meta-data : base + length



OS360

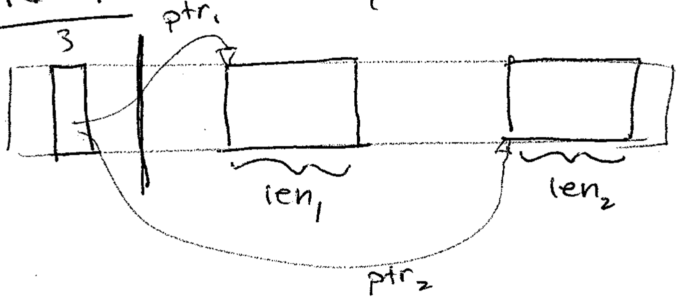
+)
easy to get at data (offset + base)
sequential access efficient

-)
Fragmentation (which kind ?)
⇒ solution: off-line compaction
Hard to predict requirements (size) @ creation time
⇒ yes) cp no) audio stream off of the web
how to extend file?

Extent-Based

> Multiple contiguous regions / file

> Meta-data : array of <ptr, len> pairs [fixed number]



Q) How to find offset x?

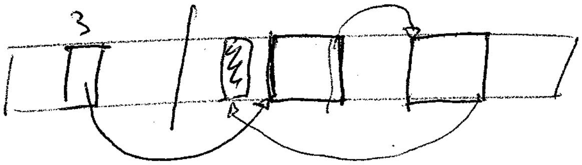
+) files can (w/o compaction, etc.) grow over time
eases external fragmentation

-) fixed # of extents
frag. still an issue

Linked - Allocation

> File kept as linked list of fixed sized blocks

> Meta-data : pointer to first block



e.g. TOPS-10, Alto

+) easy to extend file
external frag: gone

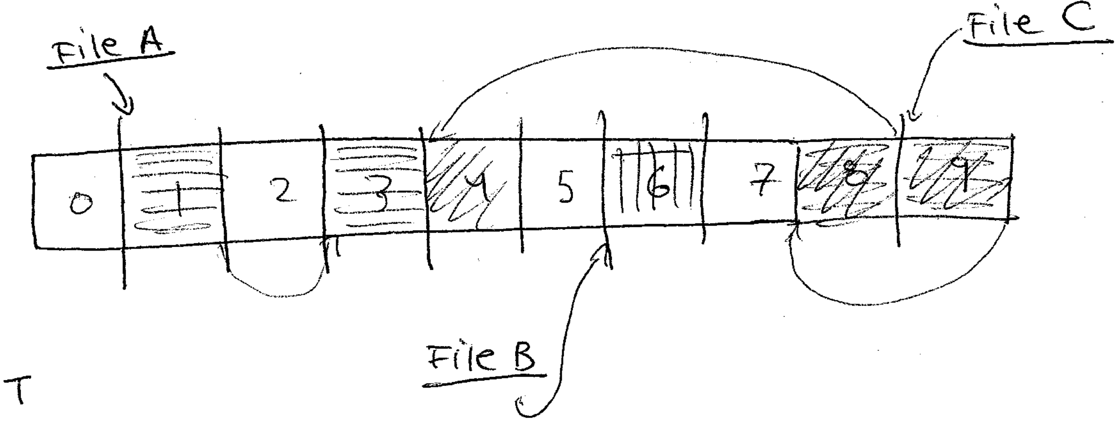
-)
> random access ? (impossible)
> seqentiol access
many seeks
> slight space overhead per block
> internal fragmentation

Variation: FAT Table

Beginning of disk: File Alloc for all files

1 entry / block

Linked list of entries / file



FAT

0	Free
1	~~A~~ 3
2	Free
3	EOF
4	EOF
5	Free
6	EOF
7	Free
8	4
9	8

"Directory"

"A"	1
"B"	6
"C"	9

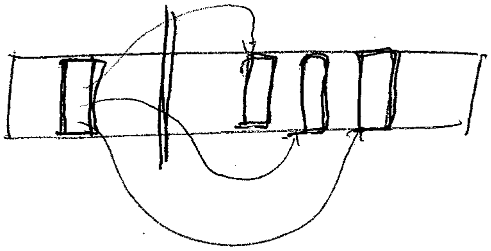
Key: [Cache FAT in memory]

+ > w/ caching, better than linked list

− > could have to read two blocks for each read

Indexed Allocation

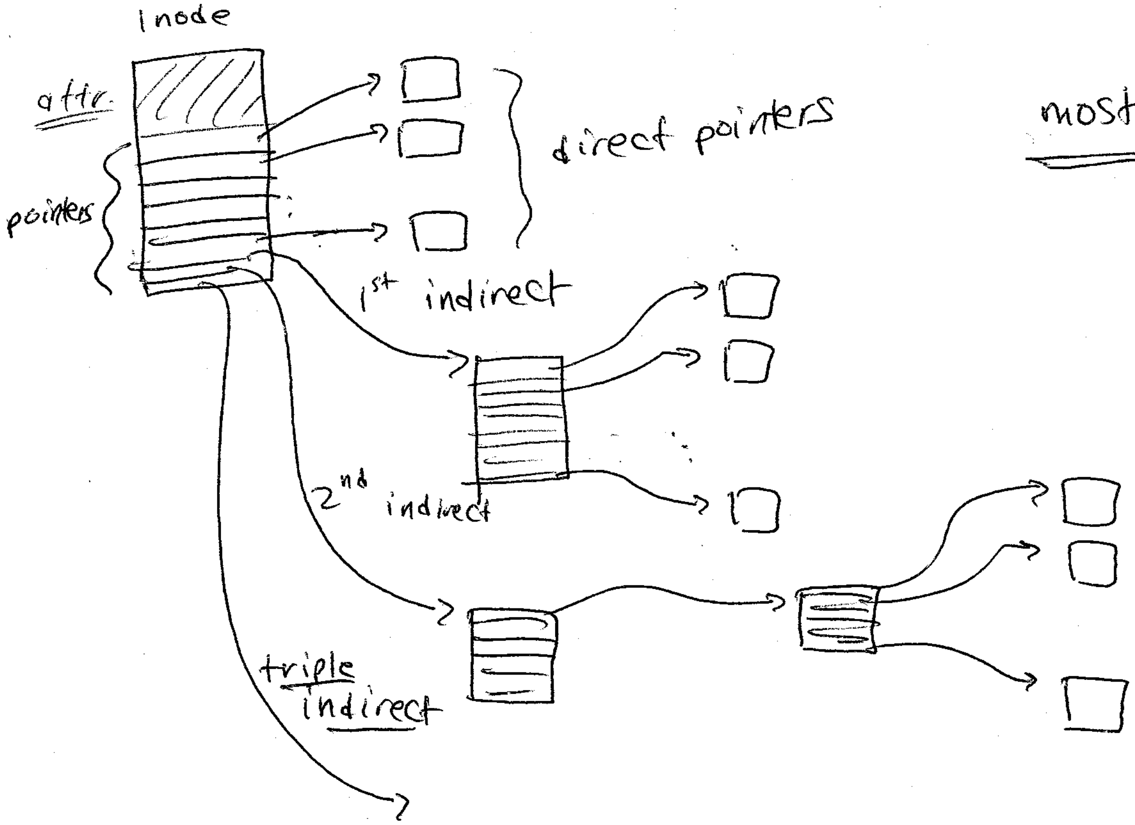
> Meta - Data : Array of block pointers



+) No ext. frag. supports random access

-) lots of seeks for seq. access

Multi-Level Indexed Files (tree like)



most Unix

+) very general
random access
large files supported

-) many lookups => large files
still true : layout of file is important