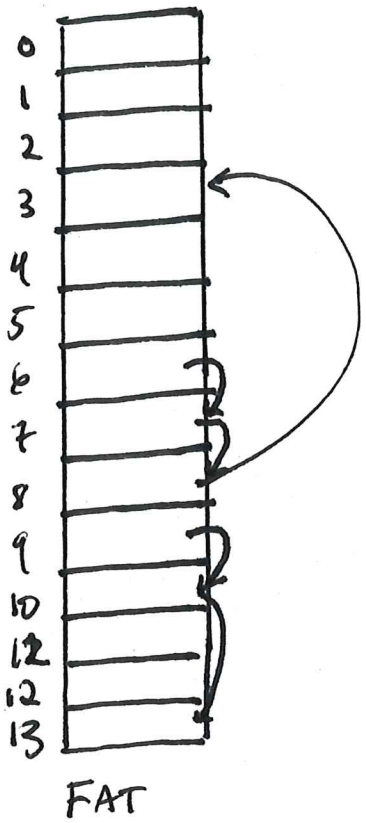
Lec 20

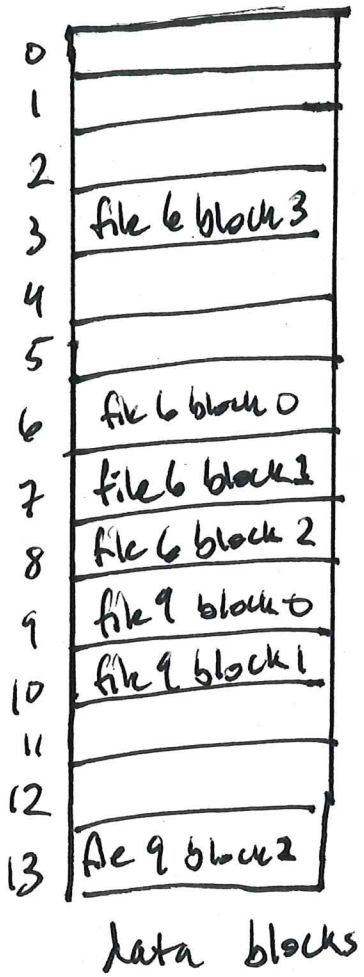
# File Layout

# File allocation table (FAT)

- widely used on USB thumb drives
- index structure: linked list
- file table: linear map of all blocks on disk
  - each file is a linked list of blocks

# FAT Example




Pros: - simple
- all operations easy to implement

Cons: - slow
- small file access is slow
- random access is slow
- fragmentation

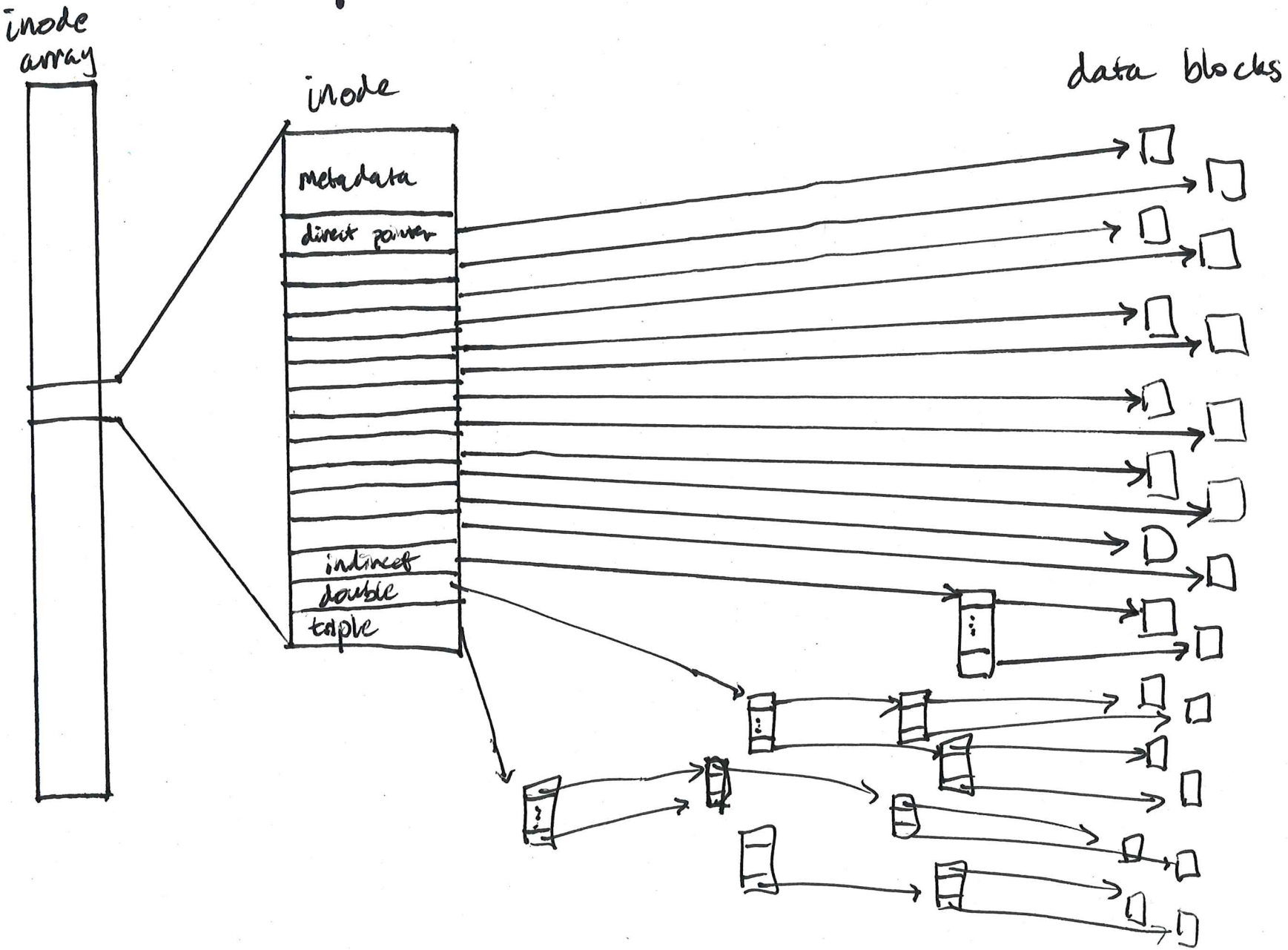
- 32 bit FAT entry for every block on disk
- file number is index into FAT
- file is a linked list of FAT entries

# Unix Fast File System (FFS)

- inode table
  - somewhat analogous to FAT
  - inumber: index into inode table

- inode:
  - metadata (owner, access permissions, ...)
  - 12 data pointers ("direct")
    - w/ 4Kb block size, max 48Kb file size
  - 13th pointer is "indirect": points to a page of 1000 ptrs!
    ↳ 4Mb
  - 14th pointer is "double indirect": points to page of indirect
    ↳ 4Gb
  - 15th pointer is "triple indirect": ...
    ↳ 4Tb

# FFS example


inode array
inode
metadata
direct pointers
indirect
double
triple
data blocks

# FFS discussion

- small files → shallow tree
  - low space overhead
- large files → deep tree
  - fast random access
- sparse files: only fill in needed parts of tree
- Locality
  - block group stores related files on nearby tracks
    - files in same directory in same group; subdirs diff group
  - store inode table partitioned across block groups
  - first fit allocation

# FFS pros/cons

Pros:

- efficient storage for all file sizes
- decent locality for all file sizes
- metadata locality

Cons:

- inefficient for tiny files (1 byte of data = 4 Kb (inode + data block))
- inefficient encoding of contiguous regions
- reserve 10-20% space to avoid fragmentation