CS-446/646

Fast File System

C. Papachristos
Robotic Workers (RoboWork) Lab
University of Nevada, Reno

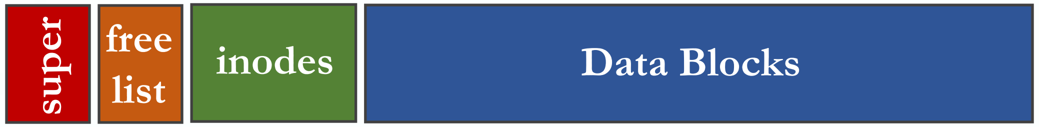
Fast File System

Original Unix *Filesystem*

- From Bell Labs by Ken Thompson

Simple and elegant:

Unix Disk Layout



- Components
  - *Data Blocks*
  - *inodes (inode Table)*
  - *Freelist*
  - *Superblock*
    - Specifies number of *Blocks* in *Filesystem*, counts Max # of *Files*, has Pointer to *Head* of *Freelist*

Problem: slow

- Only gets 2% of Disk maximum (20Kb/s) even for *Sequential* Disk Transfers

Fast File System

Original Unix *Filesystem*

- Why so slow?
- Problem 1: *Blocks* too small (512 Bytes)
  - *inode Table* too large
  - Requires more *Indirect Blocks*
  - *Transfer Rate* low (get one *Block* at time)
- Problem 2: Unorganized *Freelist*
  - Consecutive *File Blocks* not close together
  - Pay *Seek* cost even for *Sequential* Access
  - *Aging*: Becomes *Fragmented* over time
- Problem 3: Poor *Locality*
  - *inode Table* far from *Data Blocks*
  - *inodes* for *Directories* not close together
  - Poor performance doing enumeration (e.g. `ls`, `grep foo *.c`)

Fast File System

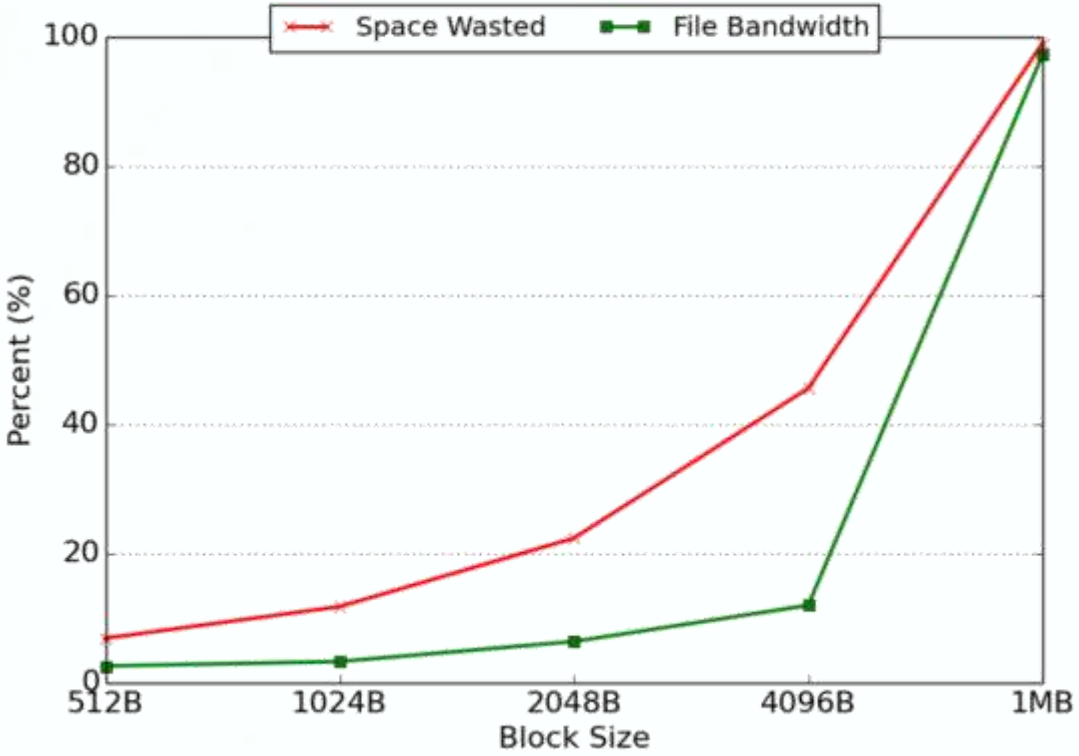
Unix *Fast File System* (FFS)

- Designed by a Berkeley research group for the BSD UNIX
  - Seminal *Filesystems* paper to read: "*A Fast File System for UNIX*", McKusick et al.
- Approach:
  - Measure state of the art *Filesystems*
  - Identify and understand the fundamental problems
  - The original *Filesystem* treats Disks like *Random-Access Memory* !
  - Build a better *Filesystem*
- Idea: Design *Filesystem* structures and Allocation Polices to be "Disk-aware"
- Next: Performance problems and how FFS fixes them

Fast File System

Problem 1: Blocks Too Small

Measurement:

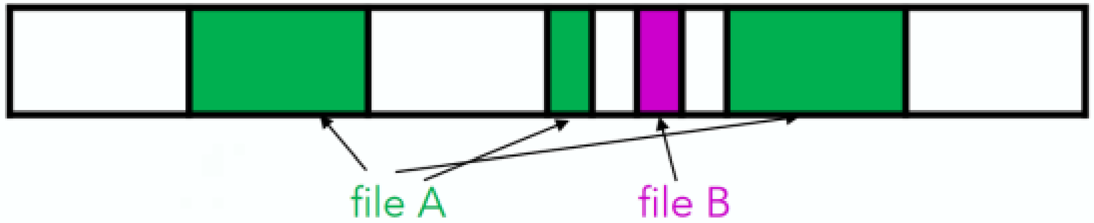


- ➢ Larger *Block* increases Bandwidth, but how to deal with *Wastage* (/*Internal Fragmentation*)?
  - ➢ Use idea from **`malloc()`**: Split unused portion

Fast File System

FFS Solution: *Fragments*

- BSD FFS:
  - Has large *Block Size* (4096B or 8192B)
  - Allow large *Block* to be chopped into smaller ones called "*Fragments*"
  - Ensure *Fragments* only used either for a) small *Files* or b) Ends of *Files*



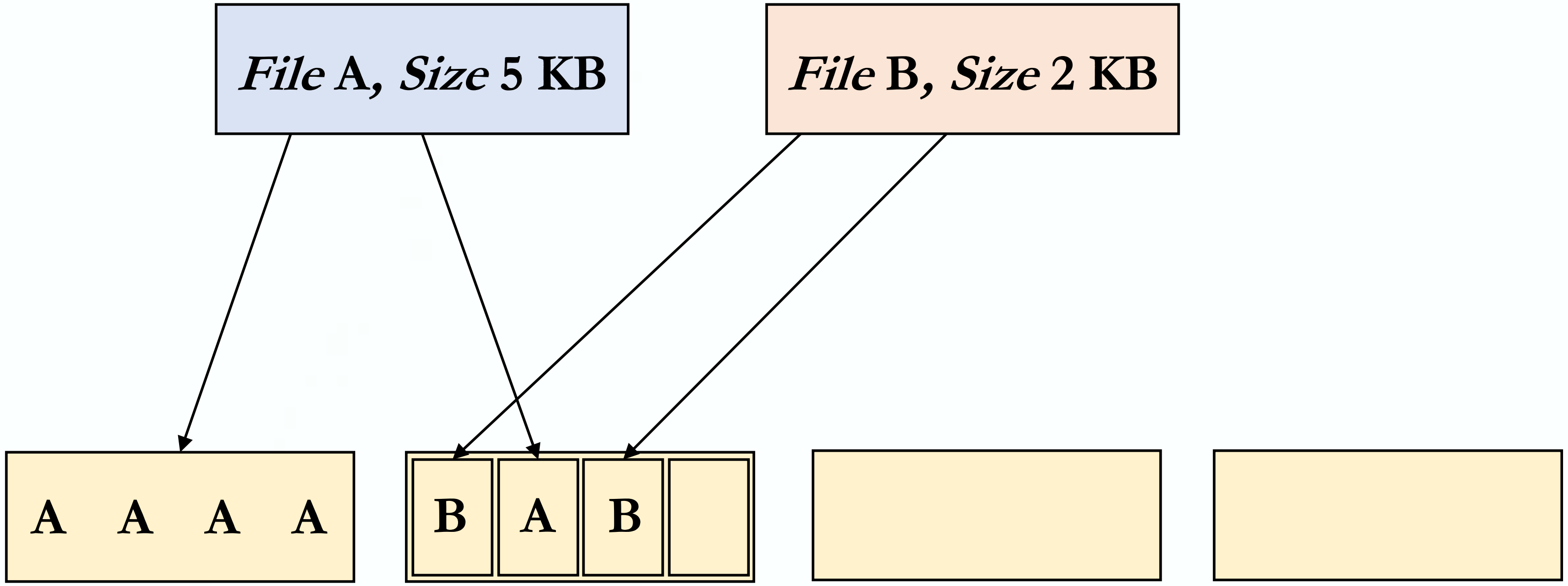
- *Fragment Size* specified at the time that the *Filesystem* is created
- Limit number of *Fragments* per *Block* to 2, 4, or 8

Advantages:

- High *Transfer Speed* for larger *Files*
- Low *Wasted* Space for a) small *Files* or b) Ends of *Files*

Fast File System

Fragments Example



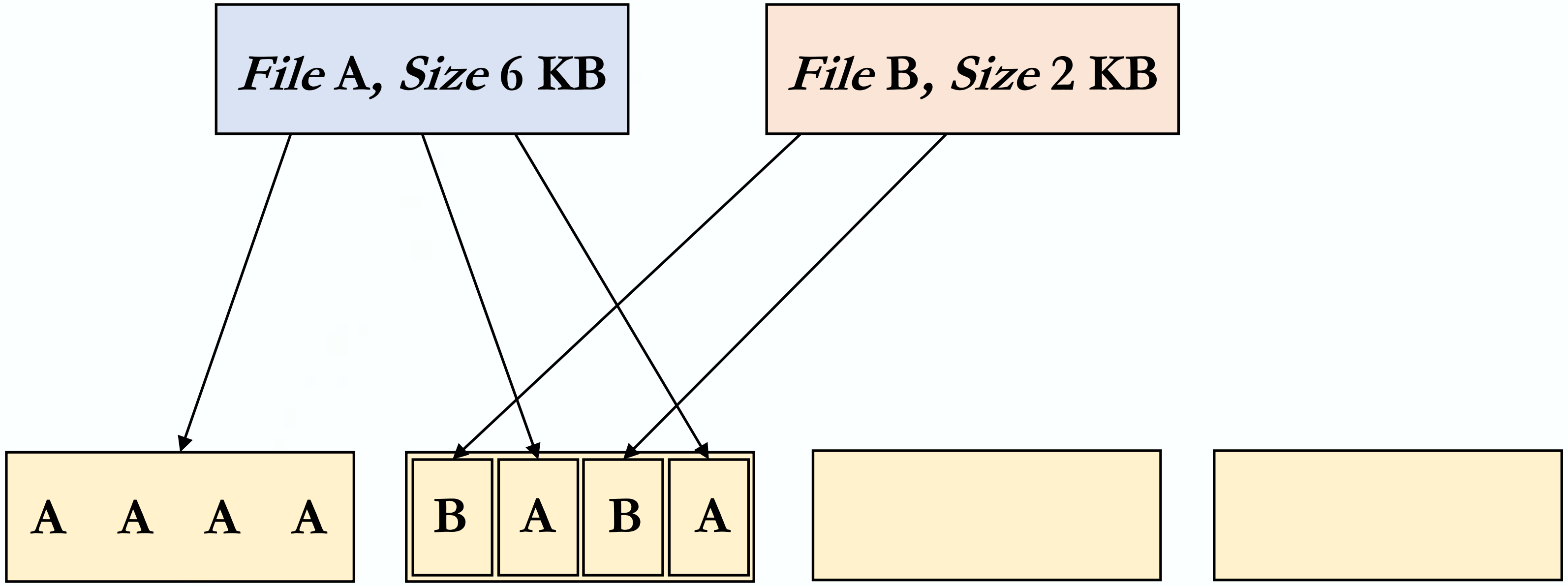
Block Size: 4 KB

Fragment Size: 1KB

Fast File System

Fragments Example

```
write(fd1, "A"); // append A to first file
```

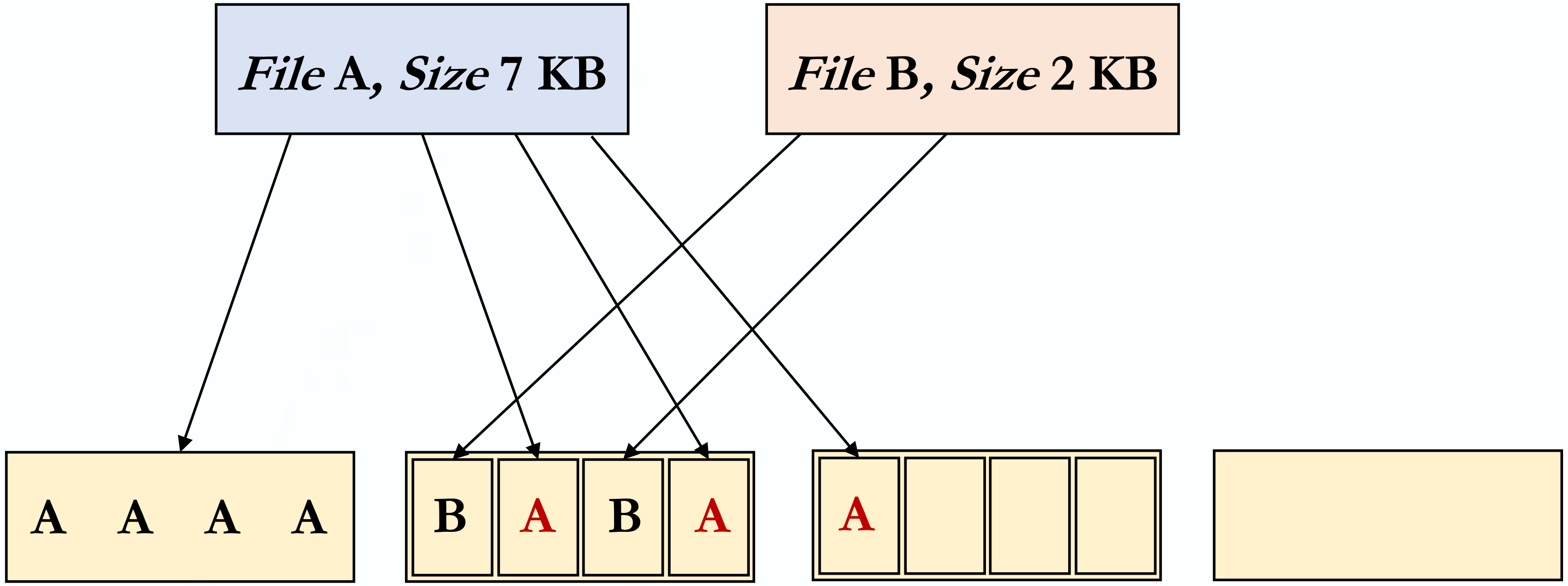


Block Size: 4 KB
Fragment Size: 1KB

Fast File System

Fragments Example

```
write(fd1, "A"); // append A to first file
write(fd1, "A");
```



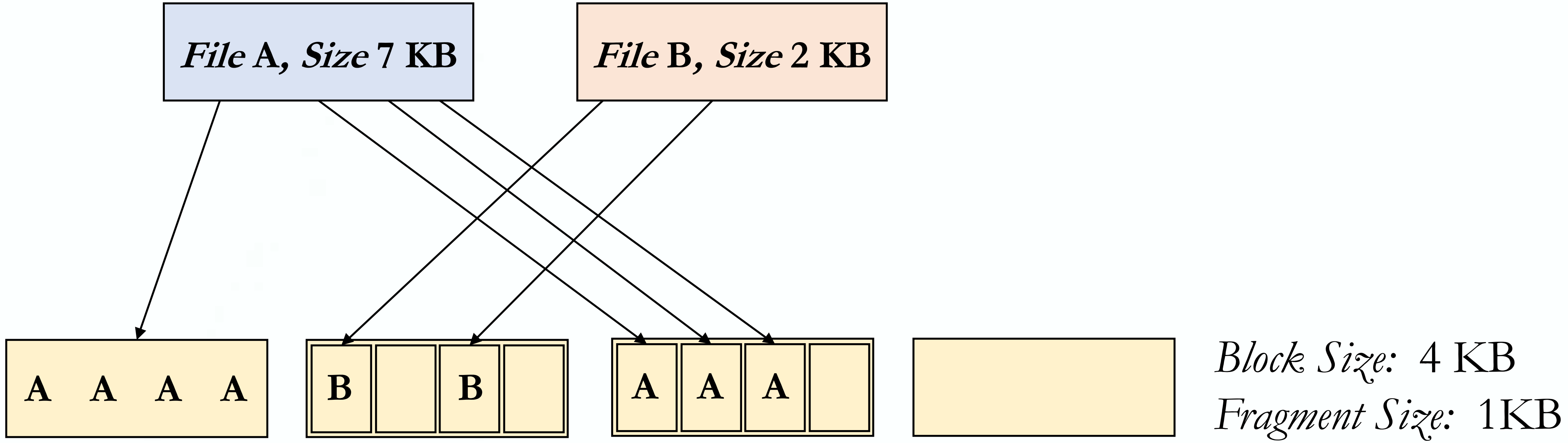
Block Size: 4 KB
Fragment Size: 1KB

- But, not allowed to use *Fragments* across multiple *Blocks*…

Fast File System

Fragments Example

```
write(fd1, "A"); // append A to first file
write(fd1, "A");
```



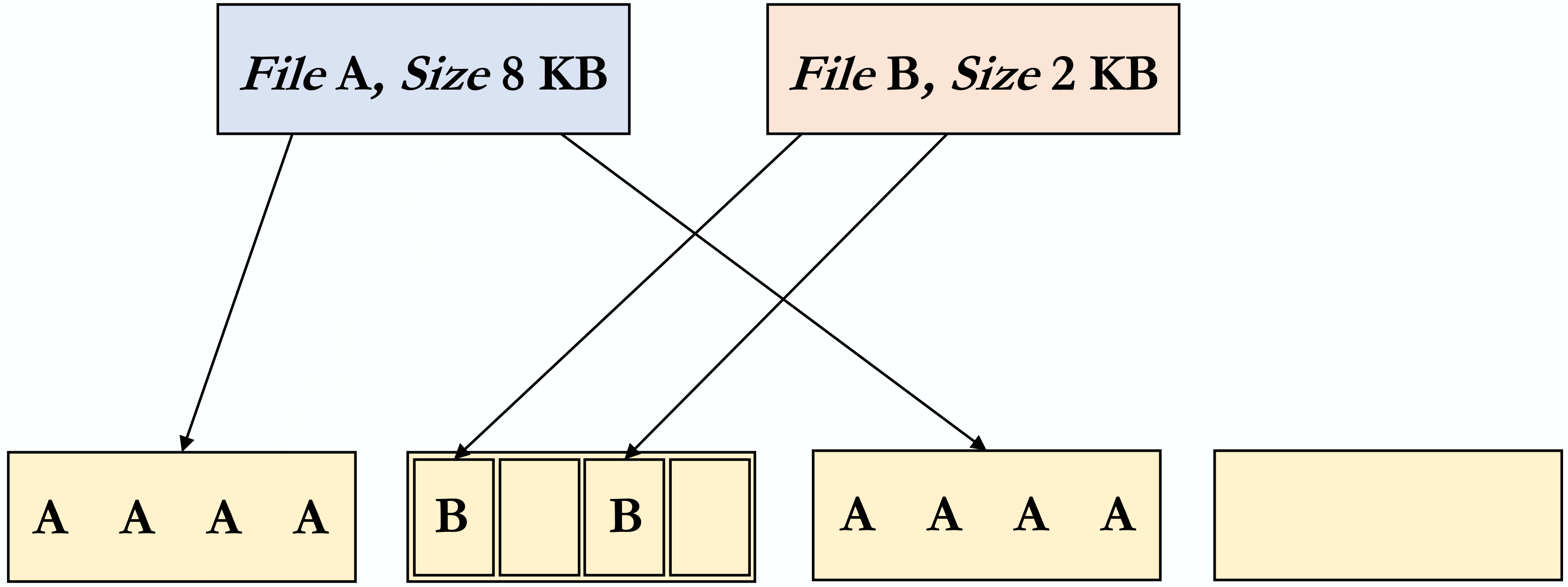
Block Size: 4 KB
Fragment Size: 1KB

- … so, copy old *Fragments* to new *Block*
- Any new Data will use remaining *Fragments*

Fast File System

Fragments Example

```
write(fd1, "A"); // append A to first file
write(fd1, "A");
write(fd1, "A");
```



File A, *Size* 8 KB

File B, *Size* 2 KB

A A A A

B B

A A A A

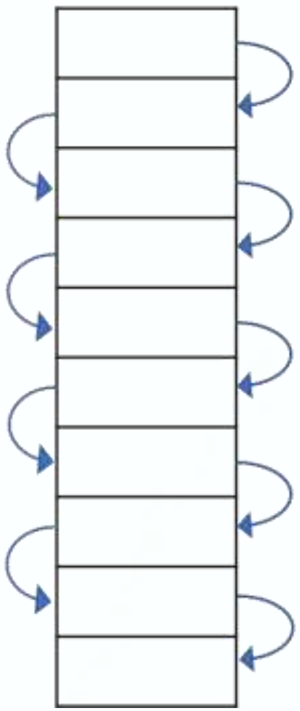
Block Size: 4 KB

Fragment Size: 1KB

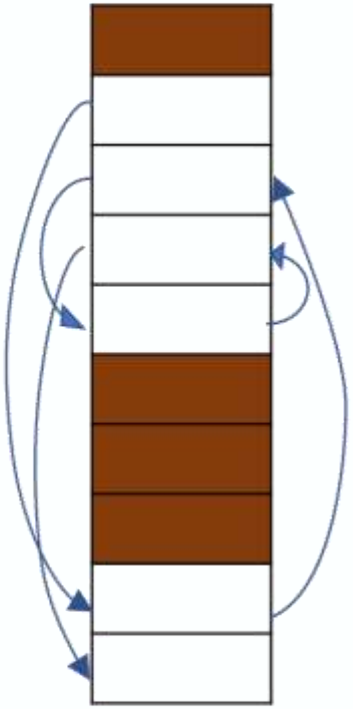
Fast File System

Problem 2: Unorganized *Freelist*

- Leads to random-like allocation of *Sequential File Blocks* over time



Initial Performance good



Gets worse over time

Measurement:

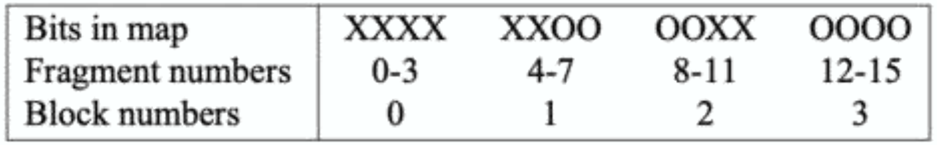
- New *Filesystem*: 17.5% of Disk Bandwidth
- Few weeks old: 3% of Disk Bandwidth

Fast File System

Solutions for Unorganized *Freelist*

- Periodic Disk *Defragmentation*
  - Cons: Locks-up Disk Bandwidth during operation
- Keep adjacent *Free Blocks* together on *Freelist*
  - Cons: Costly to maintain
- FFS Solution: *Bitmap* of *Free Blocks*
  - Each bit indicates whether *Block* is *Free*
    - e.g. 1010101111111000001111111000101100
  - Easier to find Contiguous *Free Blocks*
  - Small, so usually keep entire thing in *Memory*
  - Time to find *Free Blocks* increases if fewer *Free Blocks* are available
  - Consideration:

    Also handle *Block Fragments*

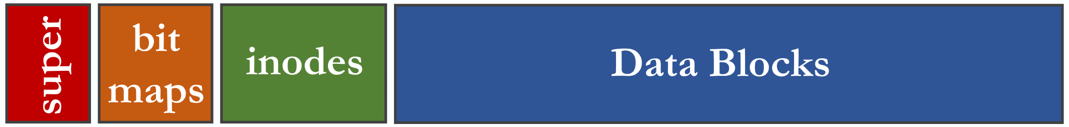


Bits in map	XXXX	XXOO	OOXX	OOOO
Fragment numbers	0-3	4-7	8-11	12-15
Block numbers	0	1	2	3

Fast File System

FFS Solution: *Bitmap* of *Free Blocks*

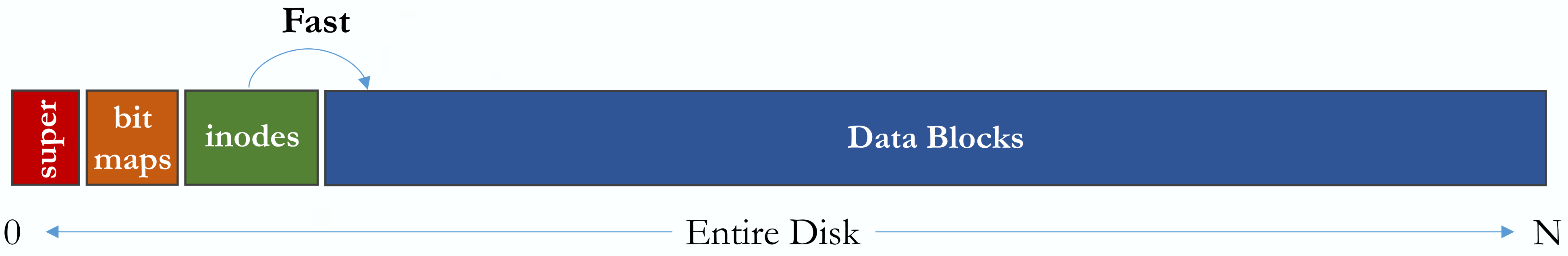
- Usually keep entire *Bitmap* in Memory:
  - 4 GiB disk / 4 KiB *Blocks* → *Bitmap* Size: 1 Mi *Entries* → 1 Mbit = 125 KB
- Allocate *Block* close to *Block* **`x`**?
  - Check for *Blocks* near **`bmap[x/32]`** (e.g. assuming **`int32_t bmap[125*1024]`**)
  - If Disk almost empty, will likely find one near
  - As Disk becomes full, search becomes more expensive and less effective
- Trade Space for Time (*Search Time*, *File Access Time*)
  - Instead of *Freelist* (effectively just a Pointer to *Head*), use *Bitmap* of *Free Blocks*

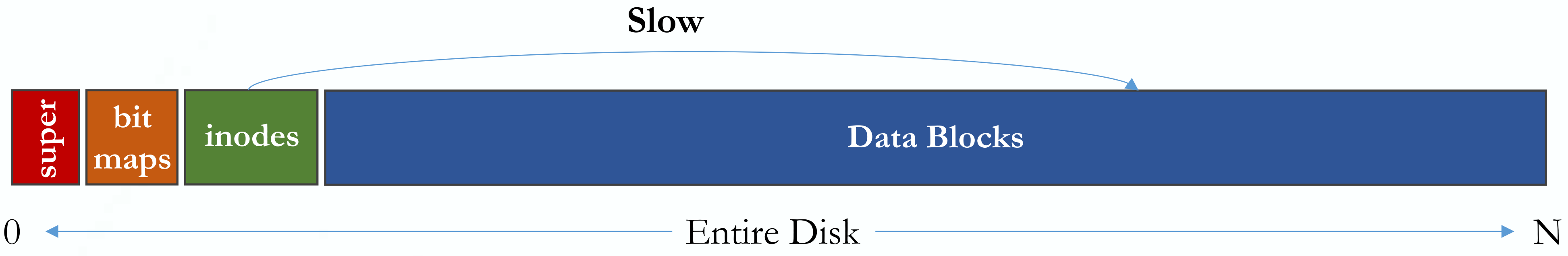


Fast File System

Problem 3: Poor *Locality*

- Desired to keep *inode* close to *Data Block*

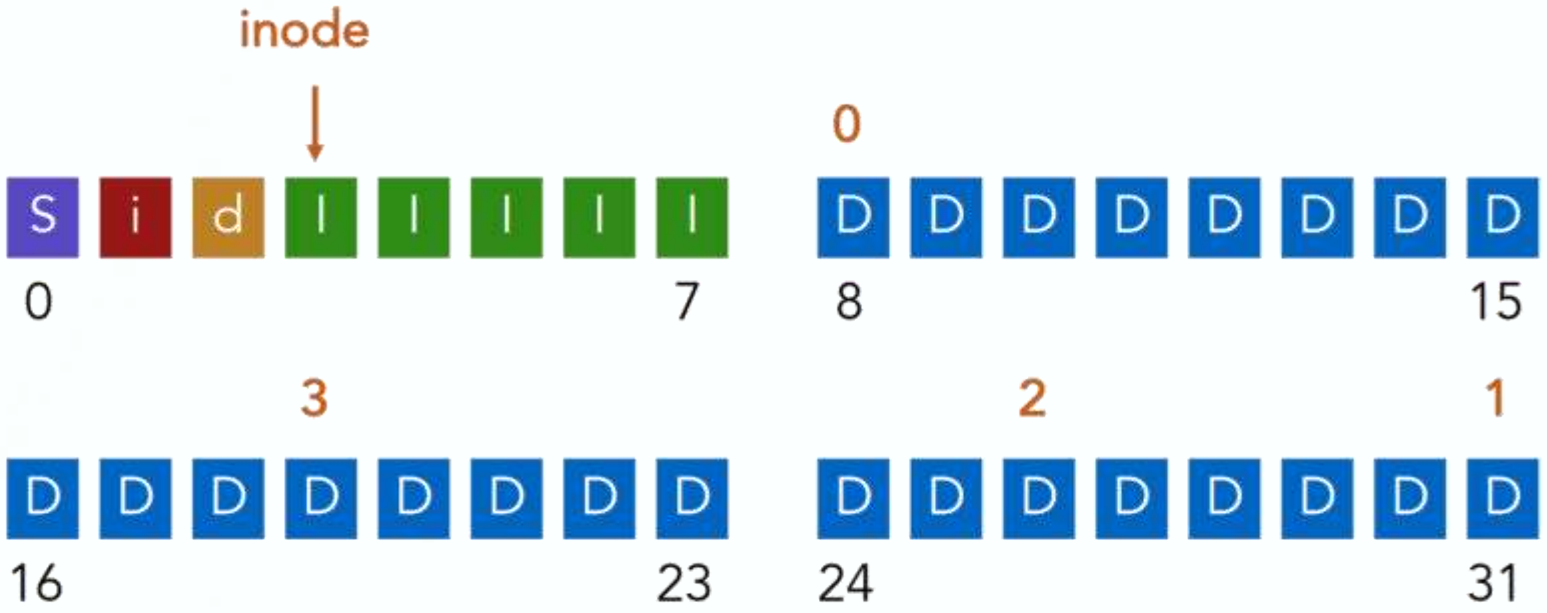




Fast File System

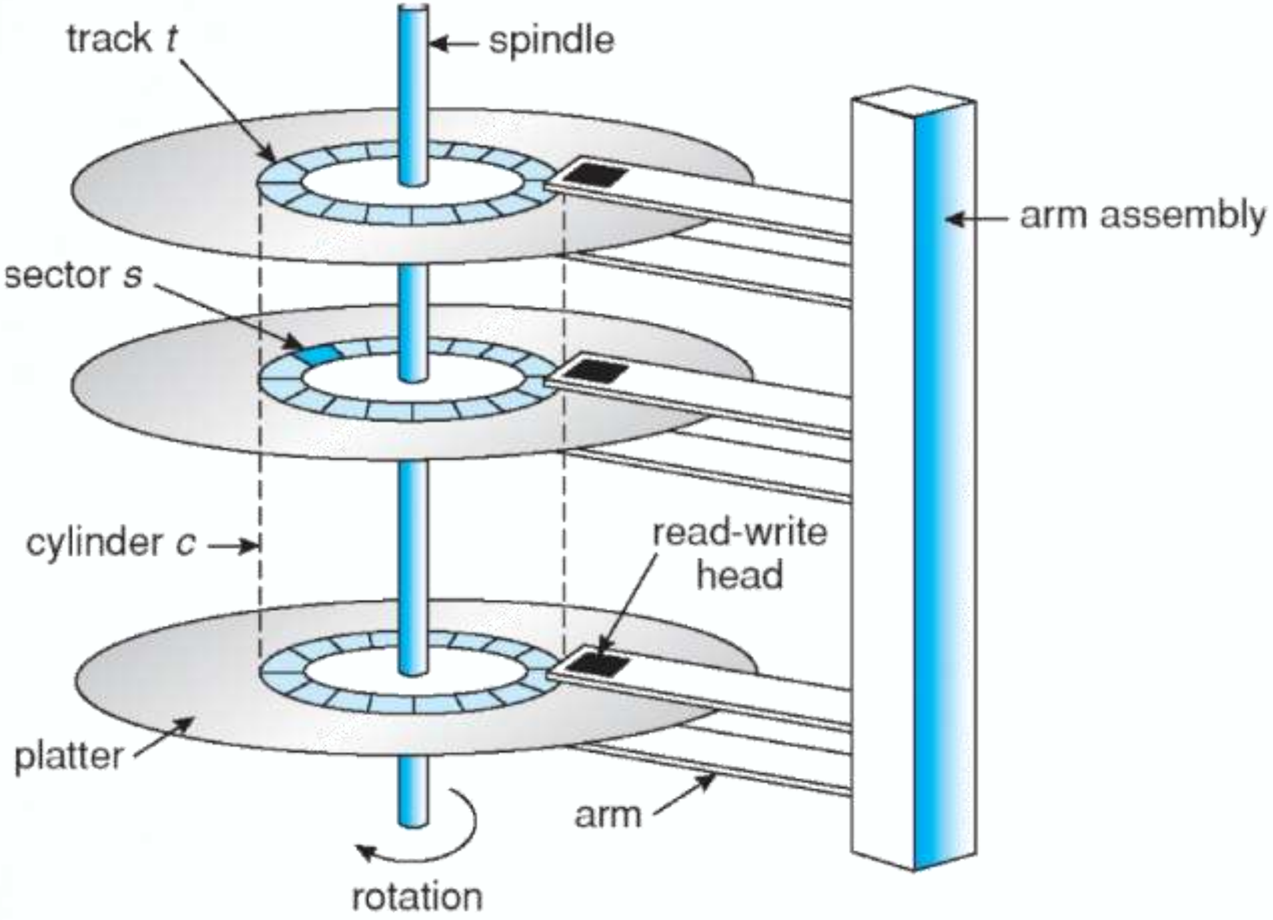
Problem 3: Poor *Locality*

- Desired to keep *inode* close to *Data Block*
- Example bad Layout and Access Sequence:



Fast File System

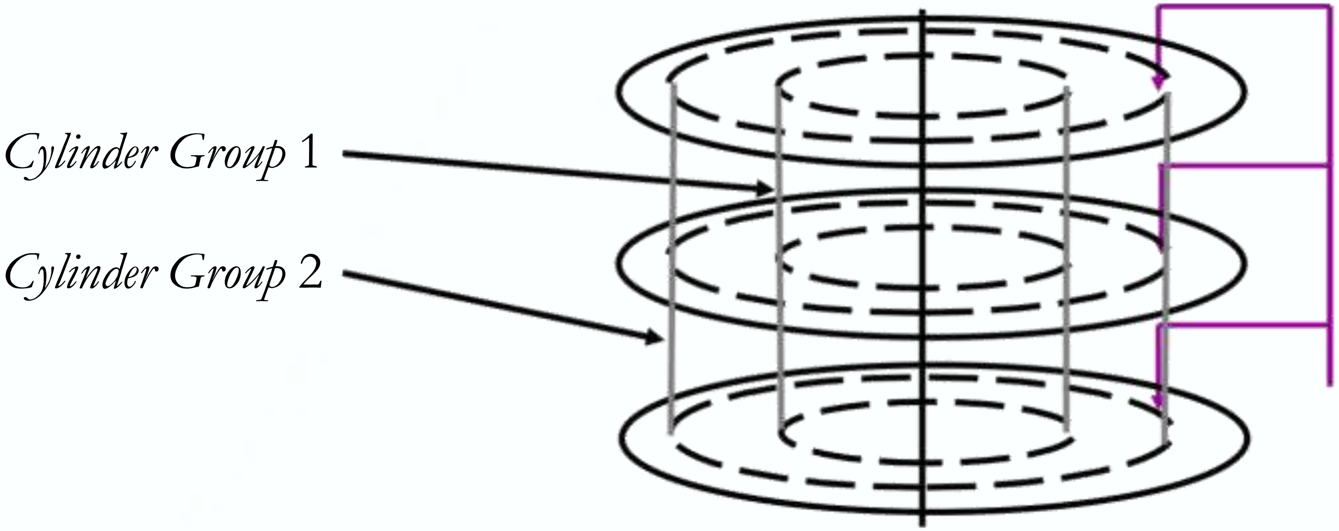
Remember: ***Cylinders, Tracks, & Sectors***



Fast File System

FFS Solution: *Cylinder Group*

- ➢ Group sets of consecutive *Cylinders* into "*Cylinder Groups*"



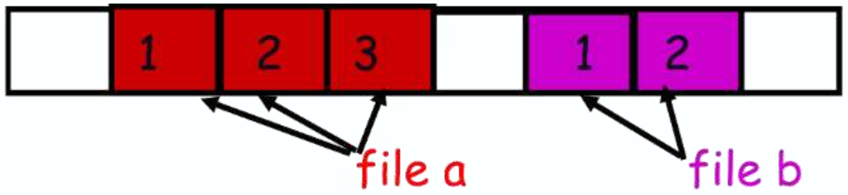
Key Concepts:
- ➢ Can access any *Block* in a *Cylinder* without performing a *Seek*
- ➢ Next fastest place is adjacent *Cylinder*

- ➢ Try to put everything related in same *Cylinder Group*
- ➢ Try to put everything unrelated in different *Groups*

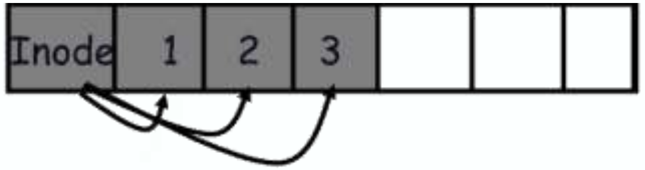
Fast File System

Clustering in FFS

- Try to put *Sequential Blocks* in adjacent *Sectors*
  - Assumption: If Accessing one *Block*, probably will need to Access next *Block* as well



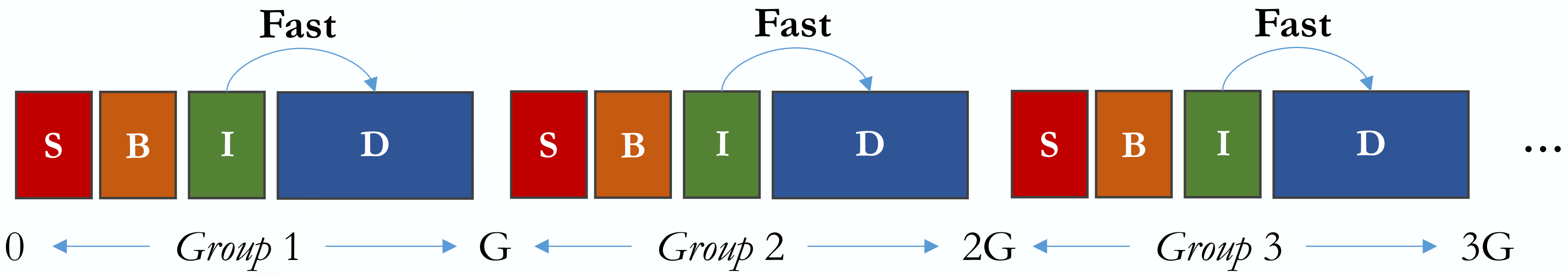
- Try to keep *inode* in same *Cylinder* as *File Data*:
  - Assumption: If accessing an *inode*, most likely will also access the *File*'s Data too



- Try to keep all *inodes* in a *Directory* in same *Cylinder Group*
  - Assumption: If accessing one Name, frequently need to access many locally, e.g. `ls -l`)

Fast File System

Resulting FFS Disk Layout:



Try to keep *inodes* close to *Data Blocks*

- Use *Groups* across Disk
- Strategy: Allocate *inodes* and *Data Blocks* in same *Cylinder Group*
- Each *Cylinder Group* basically a mini-Unix *Filesystem*

- Utility of multiple *Superblocks:*
  - *Reliability*: Superblock contains general *Filesystem* description, necessary to mount it
  - Each *Group*'s *S* is a copy the FFS *Superblock*

Fast File System

FFS Results

- Performance improvements:
  - Able to get 20-40% of Disk Bandwidth for large *Files*
  - 10-20x of original Unix *Filesystem*
  - Stable over *Filesystem* lifetime
  - Better small *File* performance
    - *Locality:* Small *Files* take up *Fragments* of same *Block*
- Other enhancements
  - Long *Filenames*
  - Parameterization
  - Maintains *Free Space* reserve (10%) that only admin can allocate *Blocks* from

CS-446/646

Time for Questions !