ILLINOIS TECH

College of Computing

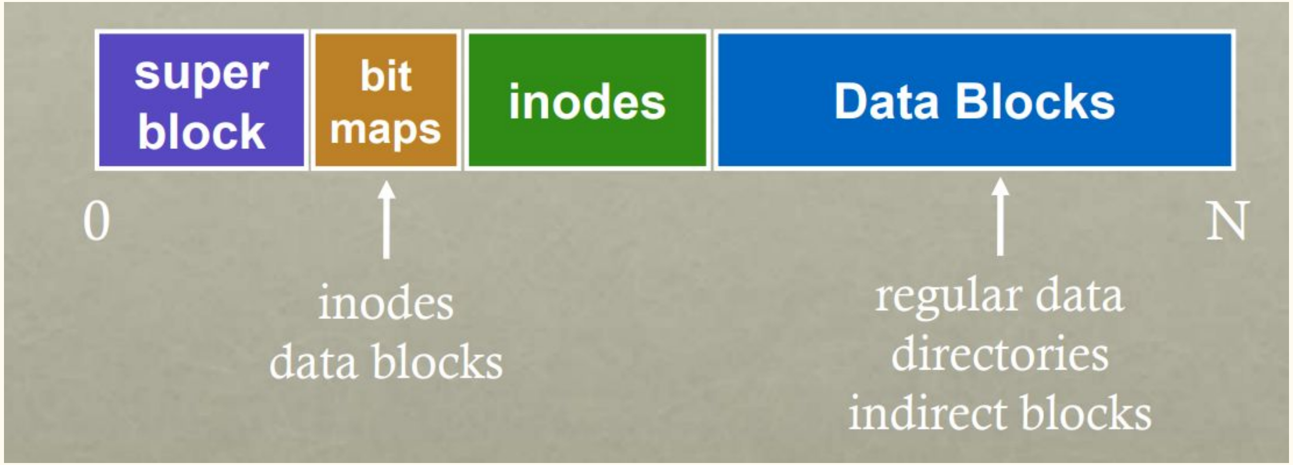
CS 450 Operating Systems
Fast File System

Yue Duan

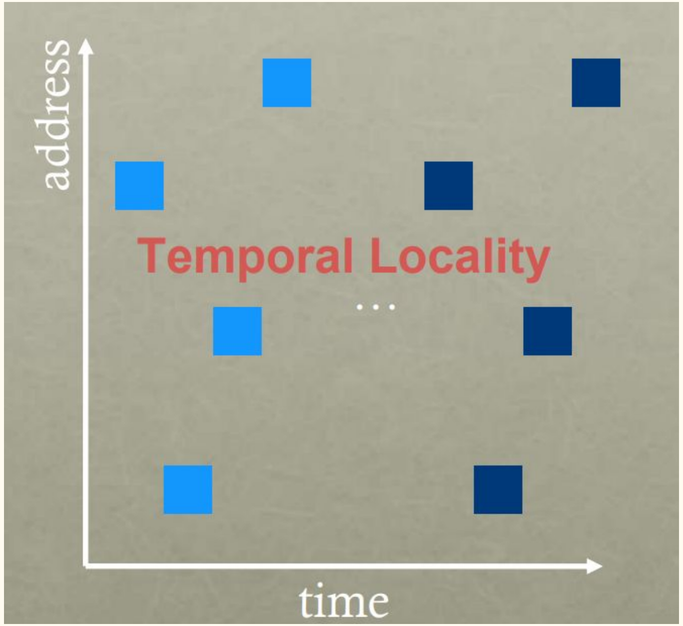
File System Case Study

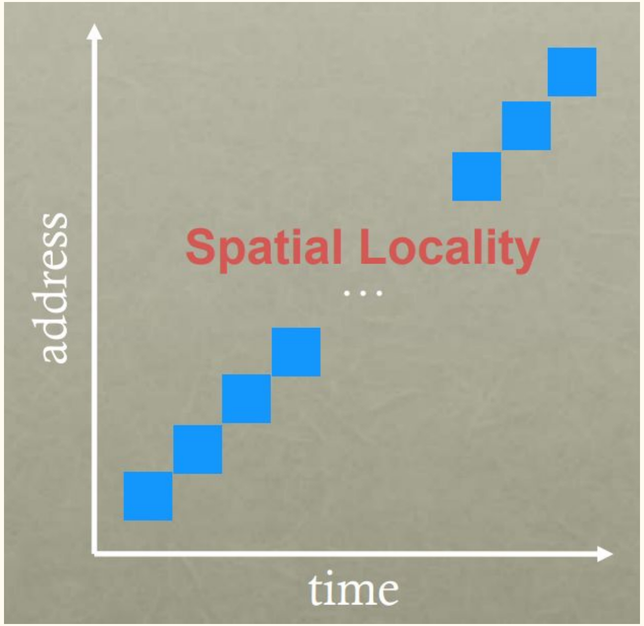
- Local
  - **FFS: Fast File System**
  - LFS: Log-Structured File System
- Network
  - **NFS: Network File System**
  - AFL: Andrew File System

Recap: Basic Layout



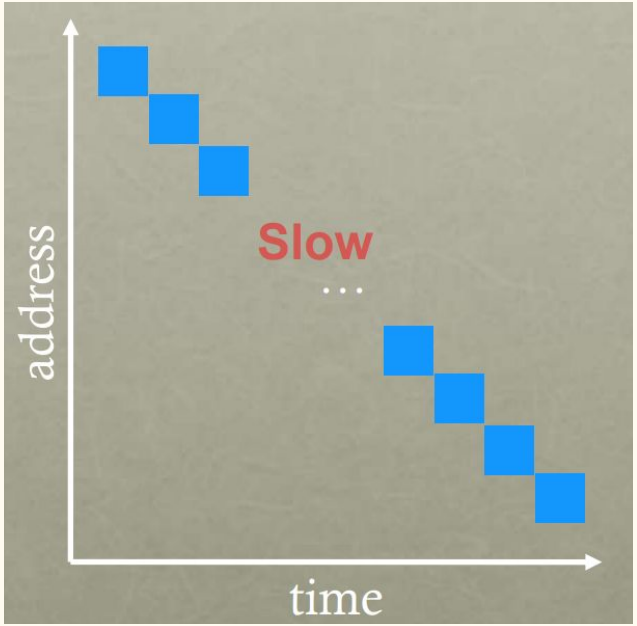
Locality Types

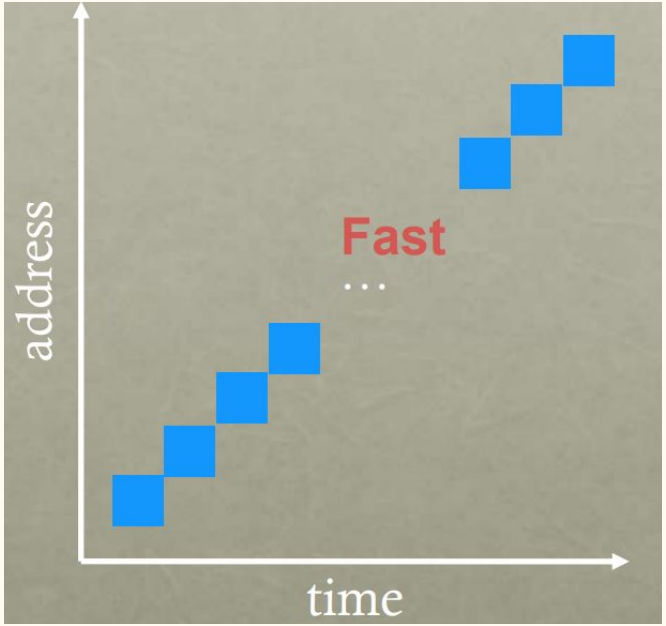




- Which type of locality is most interesting with a disk?

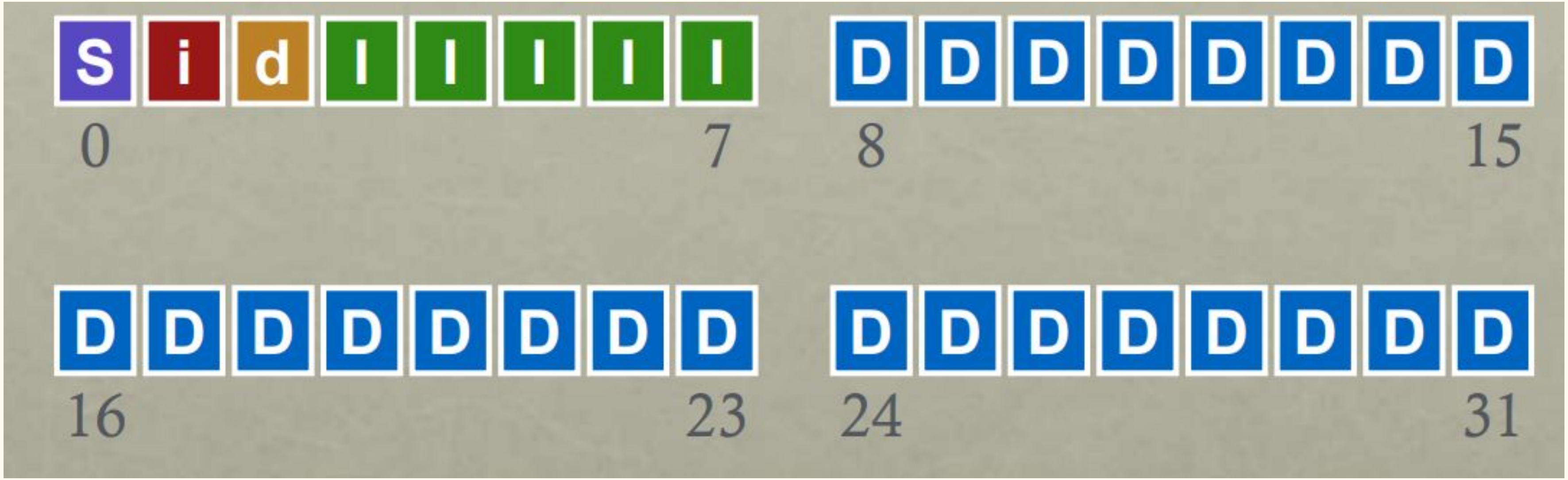
Order Matters





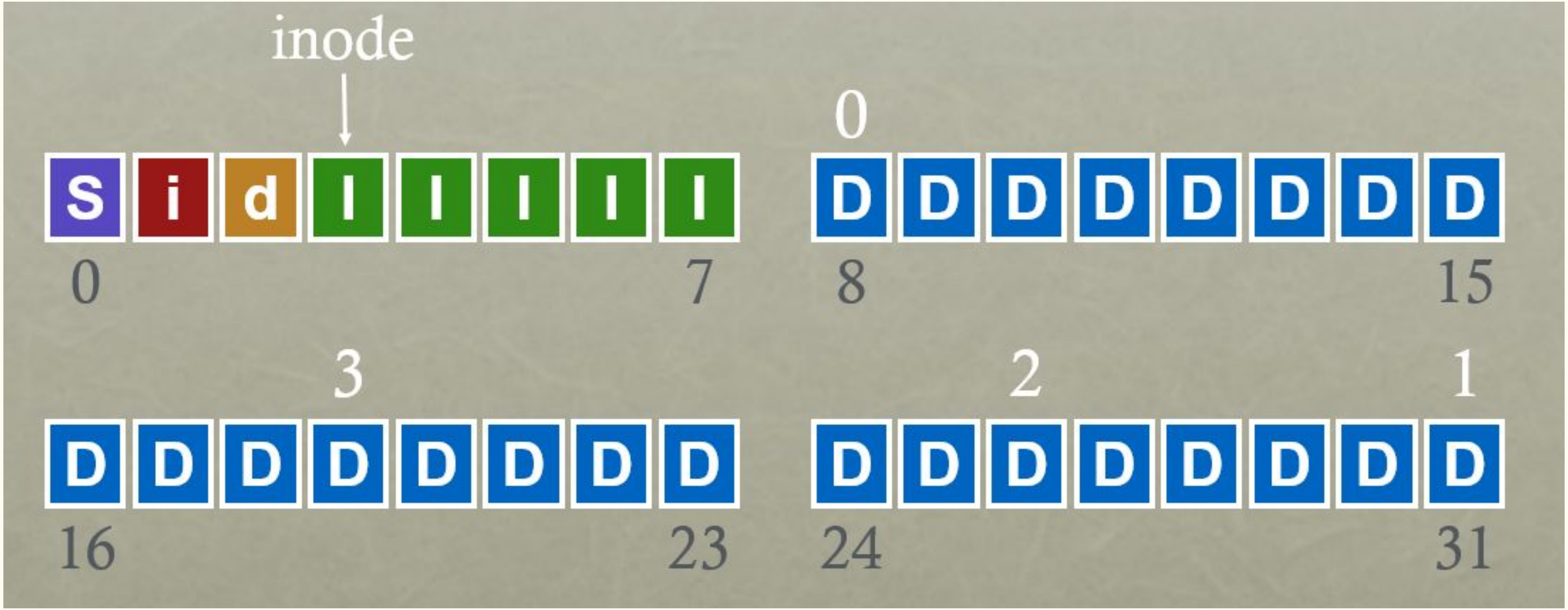
- Implication for disk schedulers?

Policy: Choose Inode, Data Blocks

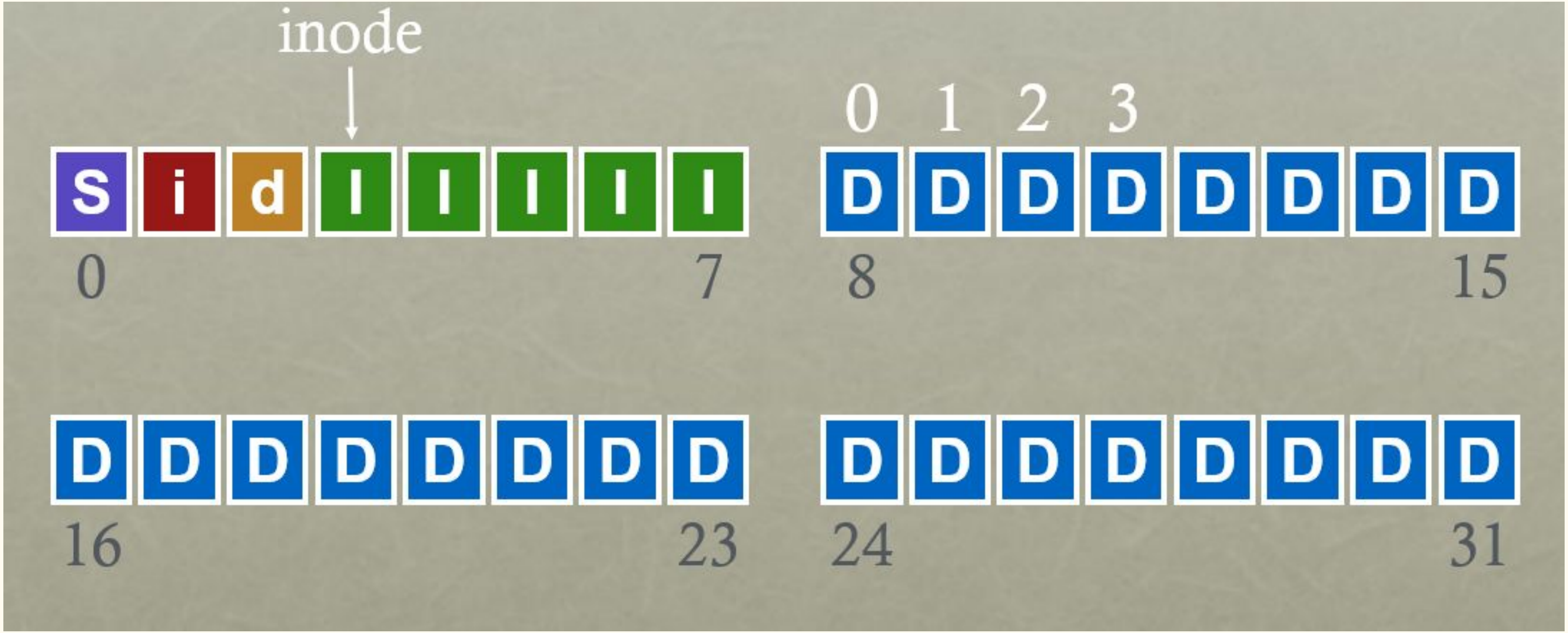


- Assuming all free, which should be chosen?

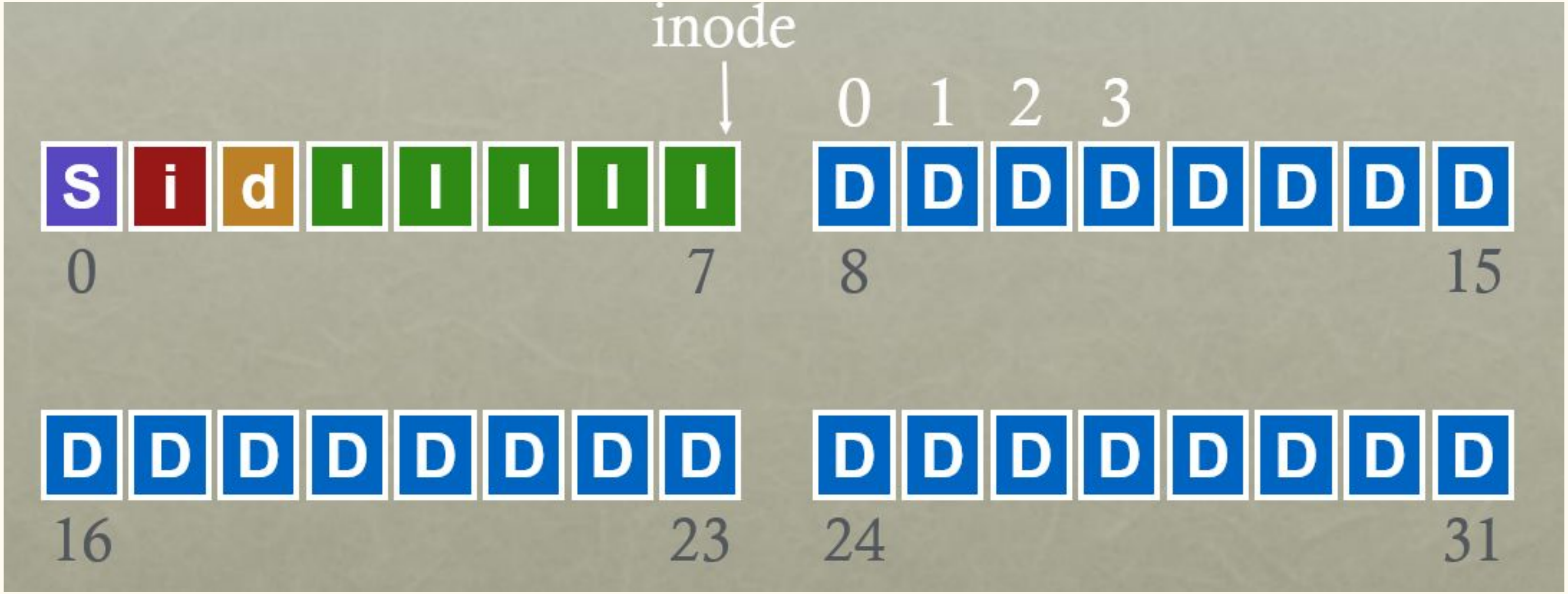
Bad File Layout



Better File Layout



Best File Layout



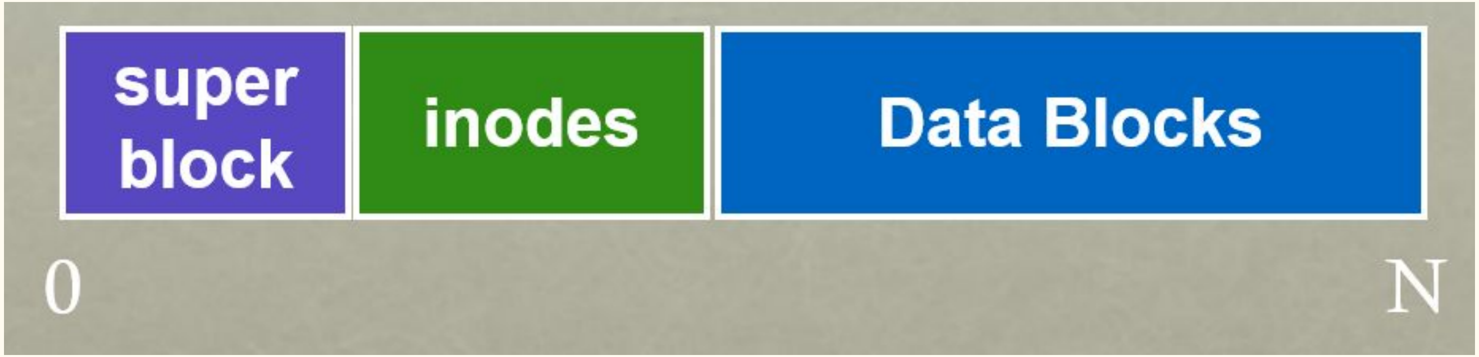
- Can’t do this for all files

Fast File System

- System Building
  - 1. identify existing state of the art
  - 2. measure it, identify and understand problems
  - 3. get idea (solutions often flow from deeply understanding problem)
  - 4. build it
- Measure then build!

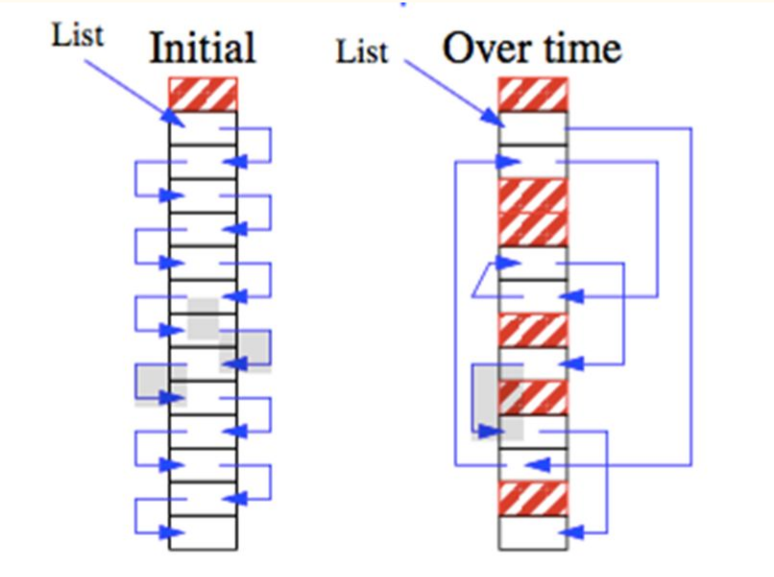
Measure Old FS

- State of the art: original UNIX file system
- Measure throughput for whole sequential file reads/writes
- Compare to theoretical max, which is the **disk bandwidth**
- Old UNIX file system: achieved only **2%** of potential. Why?



Measurement 1: Aging

- What is performance before/after aging?
    - New FS: 17.5% of disk bandwidth
    - Few weeks old: 3% of disk bandwidth
- Problem: FS becomes fragmented over time
    - free list makes contiguous chunks hard to find



Measurement 2: Block Size

- How does block size affect performance?
  - try doubling it!
- Result: Performance **more** than doubled
- Why double the performance?
  - logically adjacent blocks not physically adjacent
  - only half as many seeks+rotations now required
- Why **more** than double the performance?
  - smaller blocks require more indirect blocks

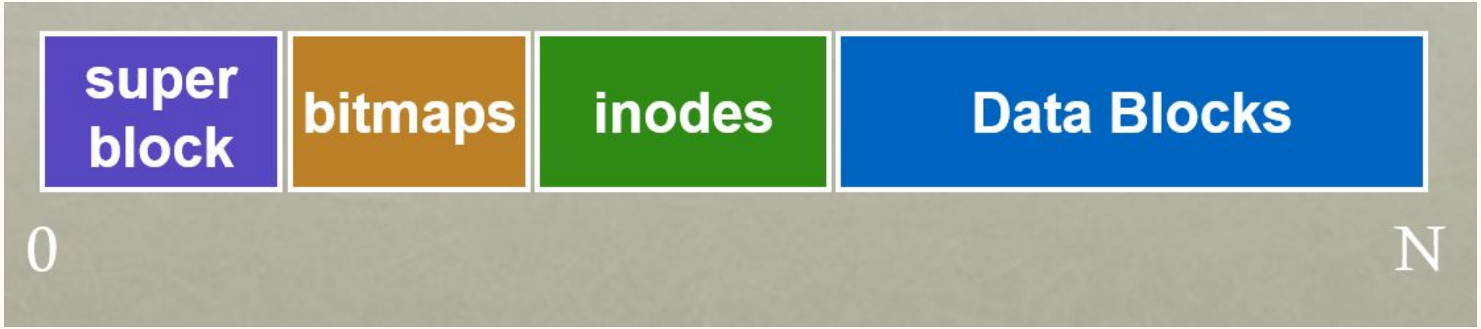
Old FS Summary

- Free list becomes scrambled ===> random allocations
- Small blocks (512 bytes)
- Blocks laid out poorly
  - long distance between inodes/data
  - related inodes not close to one another
    - inodes in same directory (ls –l)
- Result: 2% of potential performance! (and worse over time)
- **Problem**: old FS treats disk like RAM!

Solution: Disk-Awareness

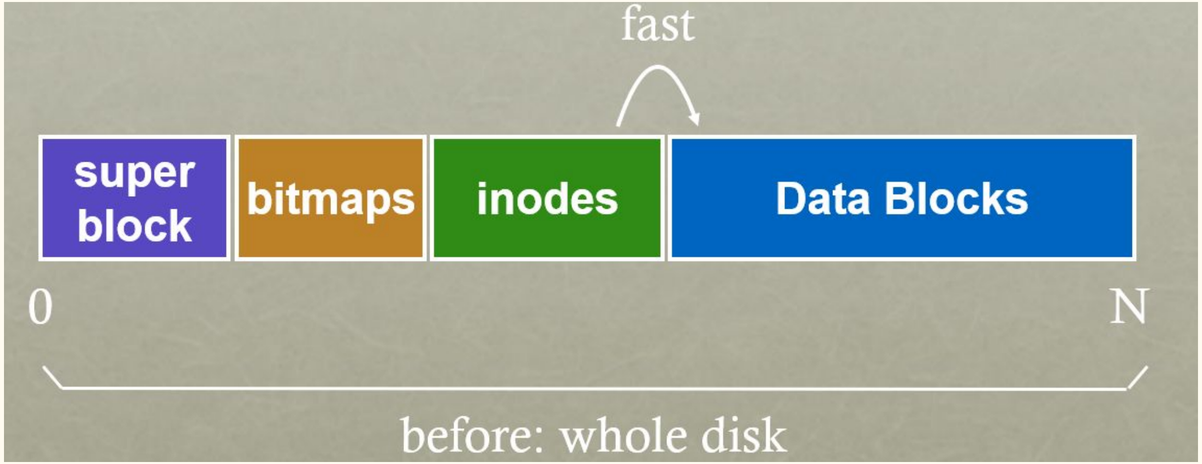
- Primary File System Design Questions:
  - **Where** to place meta-data and data on disk?
  - **How** to use big blocks without wasting space?

Placement Technique 1: Bitmaps



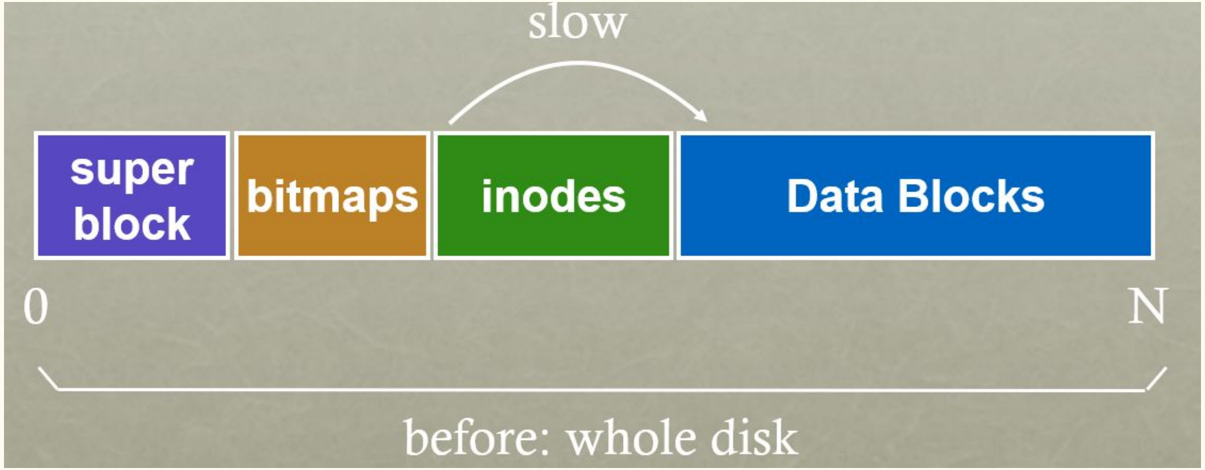
- Use bitmaps instead of free list
- Provides better speed, with more global view
- Faster to find contiguous free blocks

Placement Technique 2: Groups



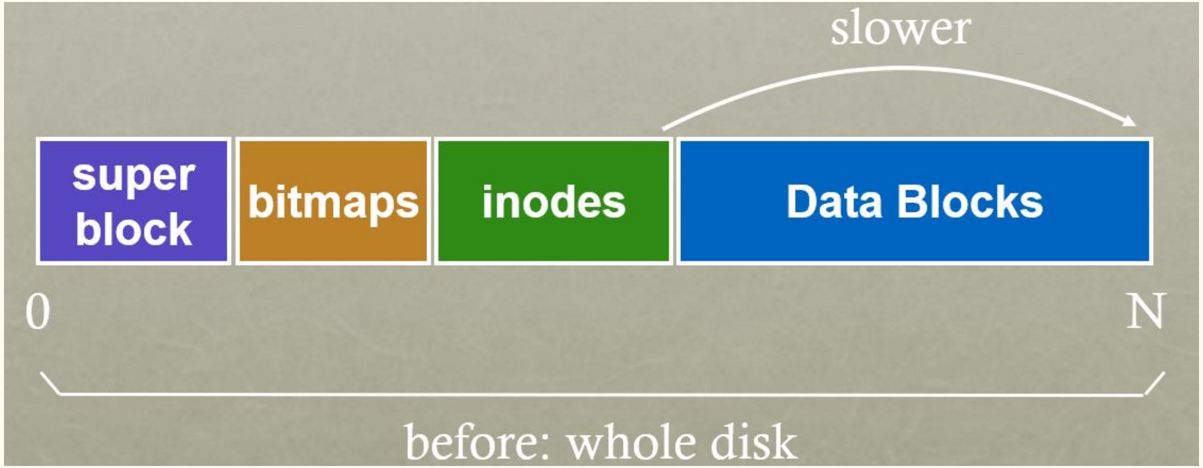
- How to keep inode close to data?

Placement Technique 2: Groups



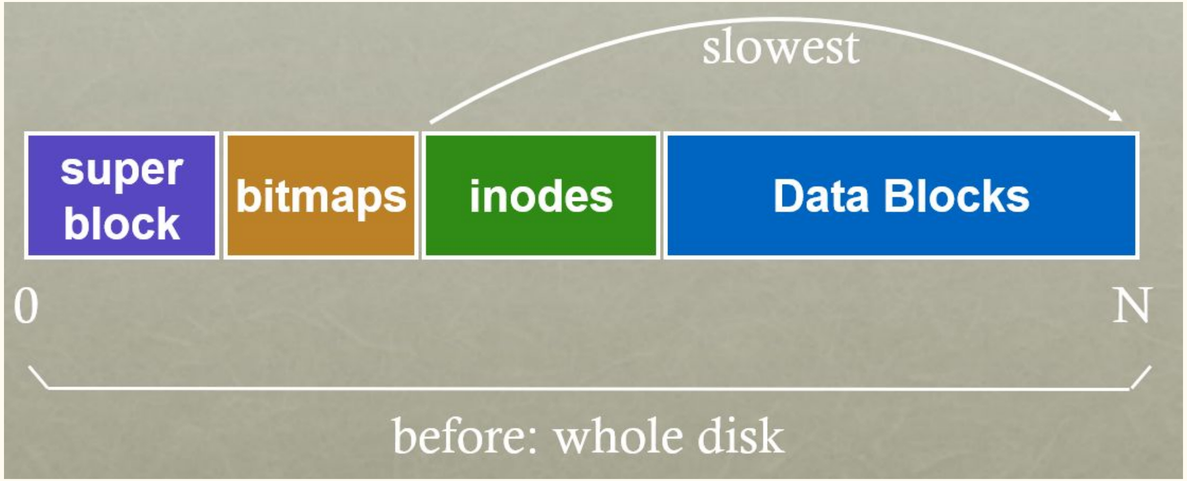
- How to keep inode close to data?

Placement Technique 2: Groups



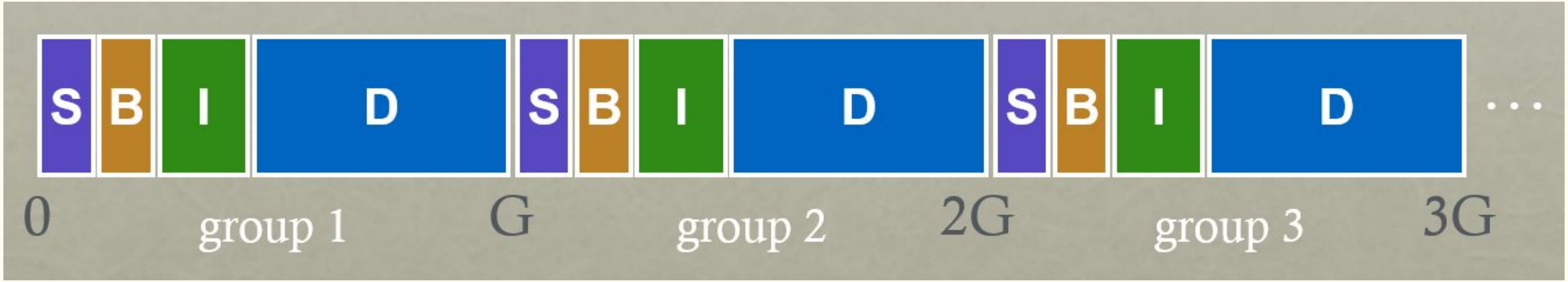
- How to keep inode close to data?

Placement Technique 2: Groups



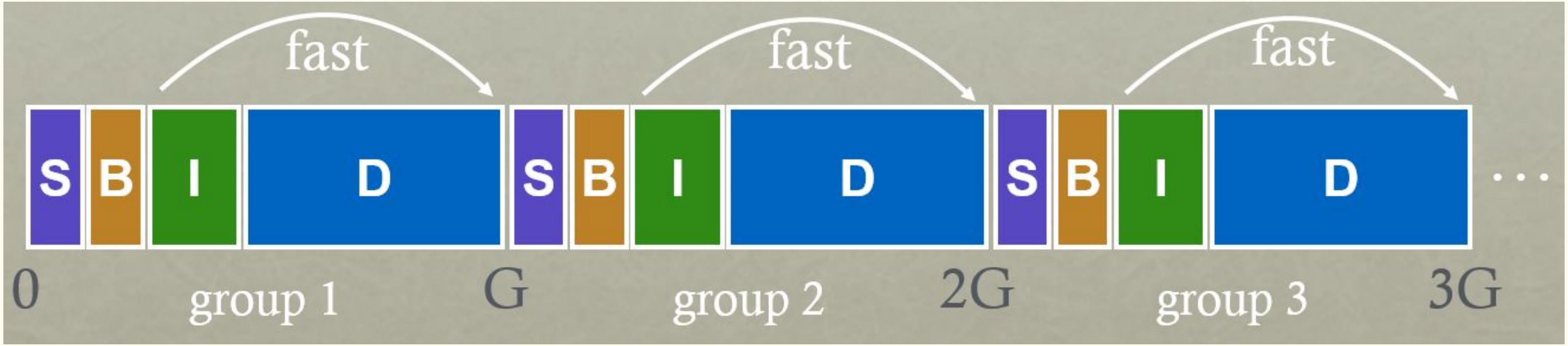
- How to keep inode close to data?

Placement Technique 2: Groups



- How to keep inode close to data?
  - key idea: keep inode close to data
  - use groups across disks
  - try to place inode and data in same group

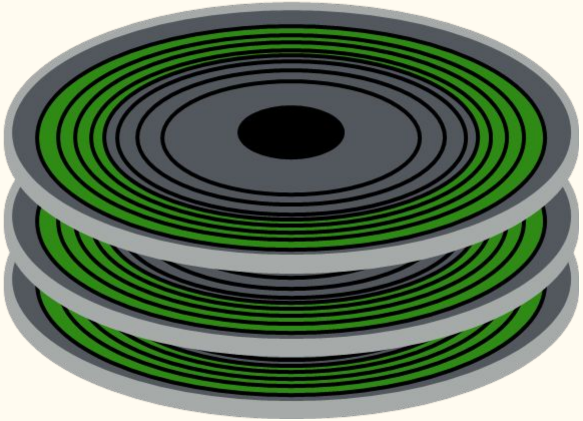
Placement Technique 2: Groups



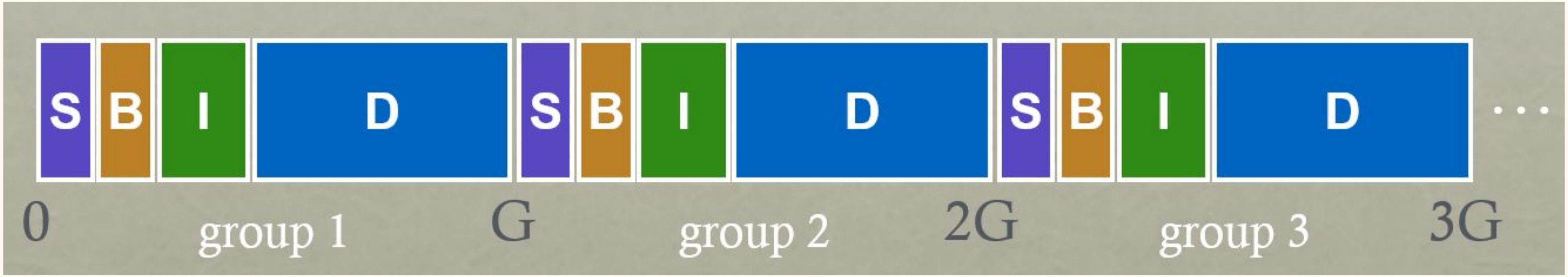
- Strategy: allocate inodes and data blocks in same group.

Groups

- In FFS, groups were ranges of cylinders
  - called **cylinder group**
- In ext2-4, groups are ranges of blocks
  - called **block group**

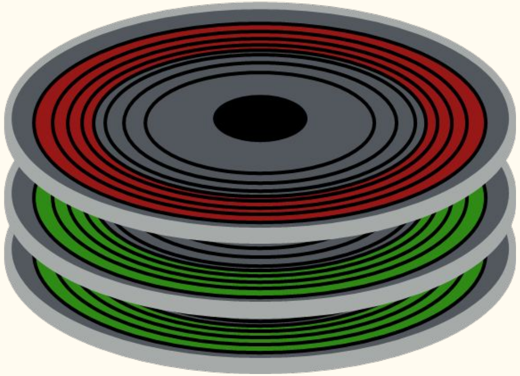


Placement Technique 3: Super Rotation



- Is it useful to have multiple replicated super blocks?
  - Yes, if some (but not all) fail.

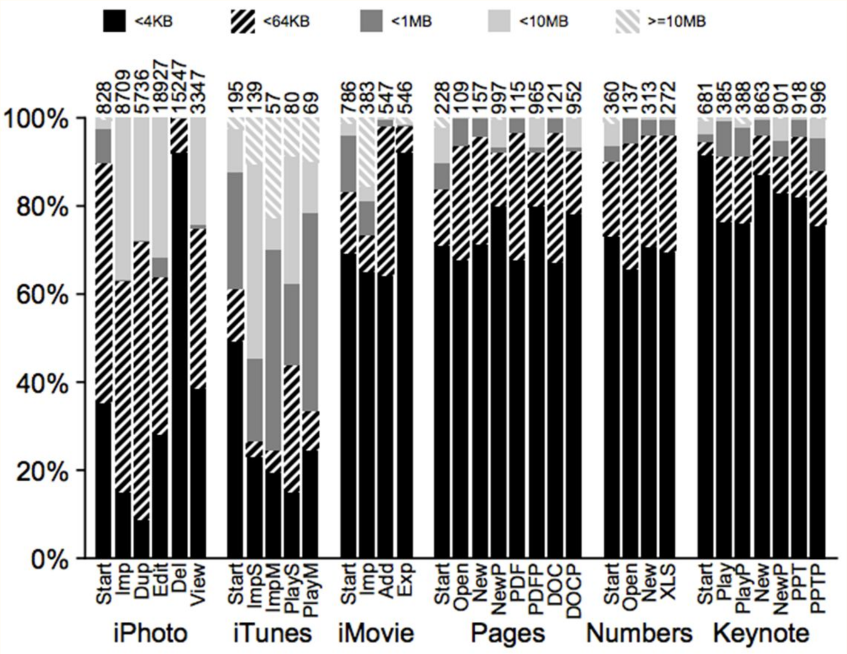
Problem



- Old FS: All super-block copies are on the **top platter**
- Correlated failures! What if **top platter** dies?
- Solution:
  - for each group, store super-block at different offset
  - super rotation:
    - use multiple super blocks for each group using different offset

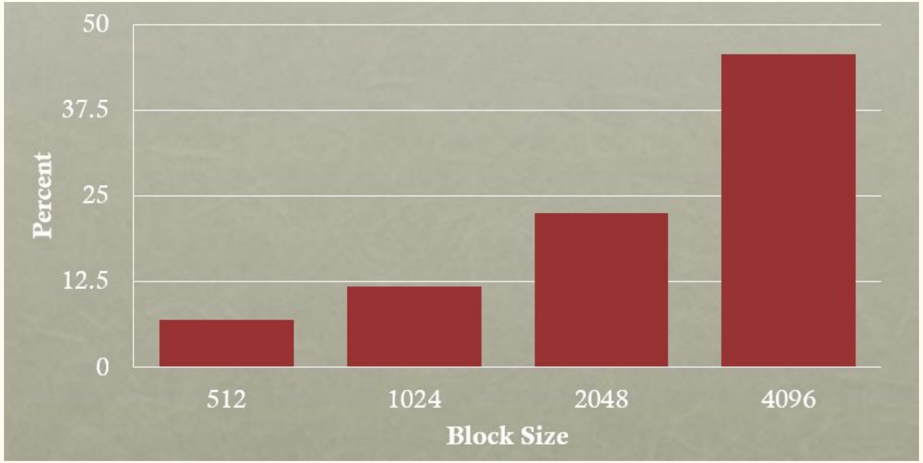
Technique 4: Larger Blocks

- Observation: Doubling block size for old FS over doubled performance
- Why not make blocks huge?

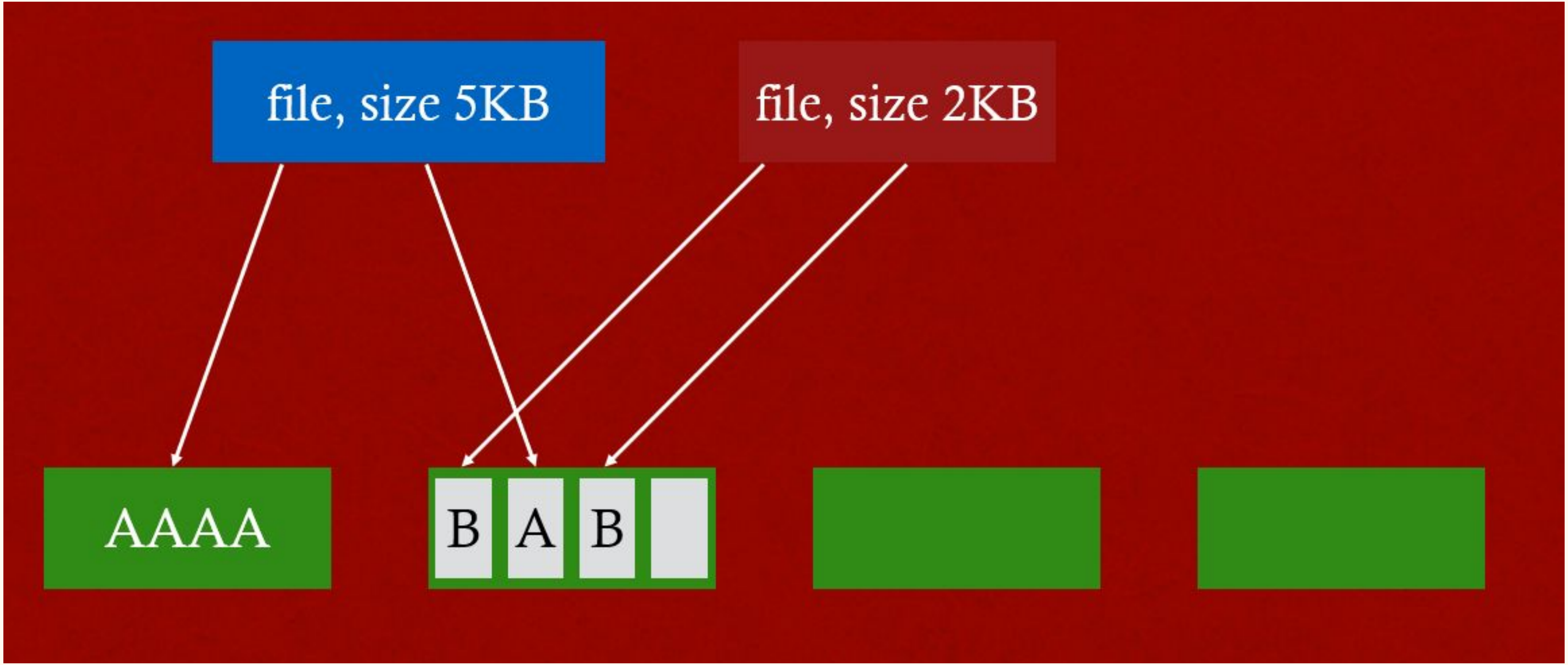


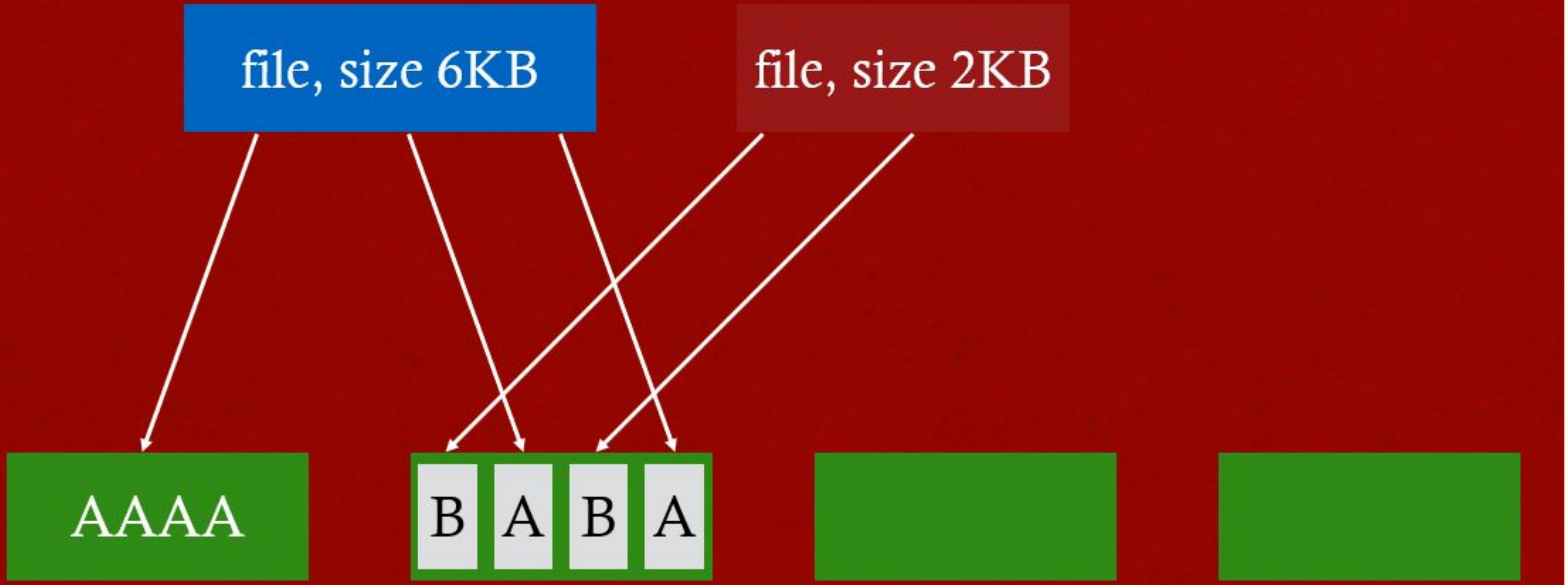
- Most file are very small, even today!
- Large block:
  - lots of waste due to internal fragment in most blocks

Solution: Fragments

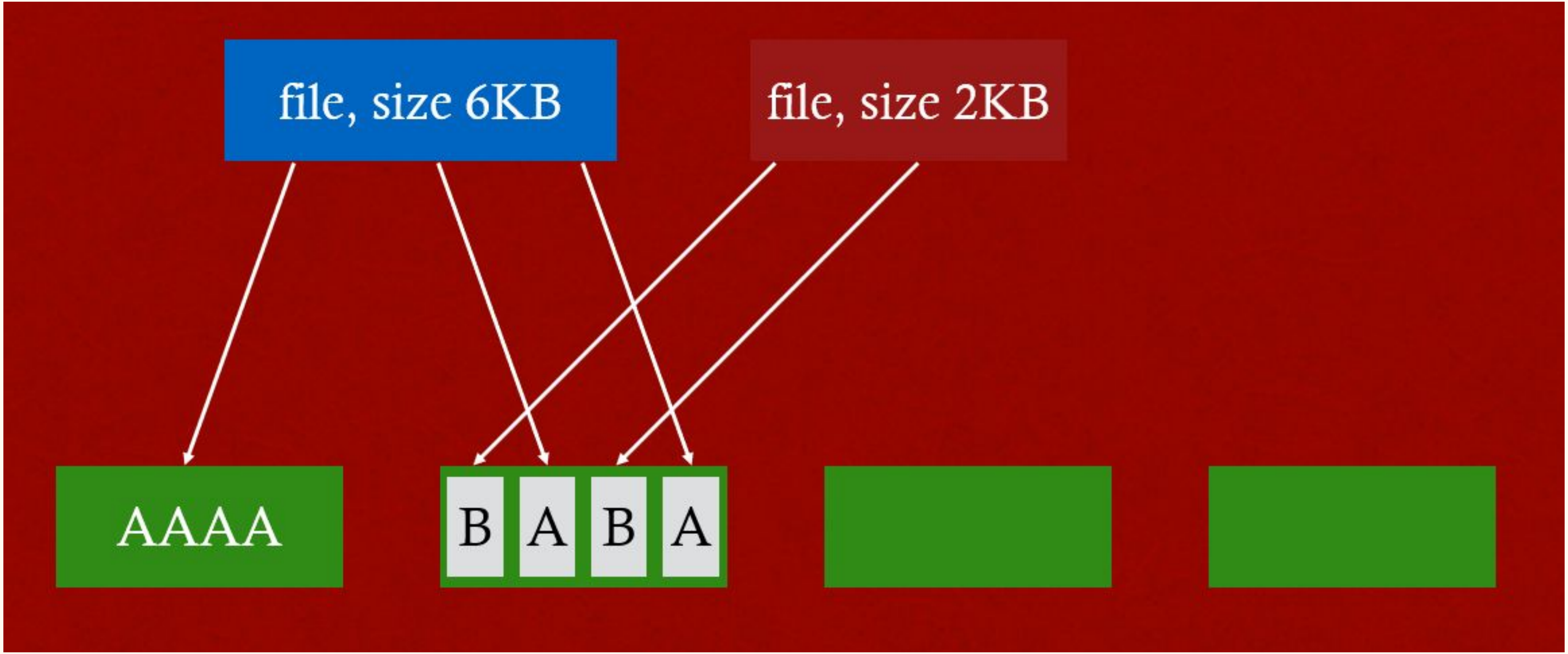


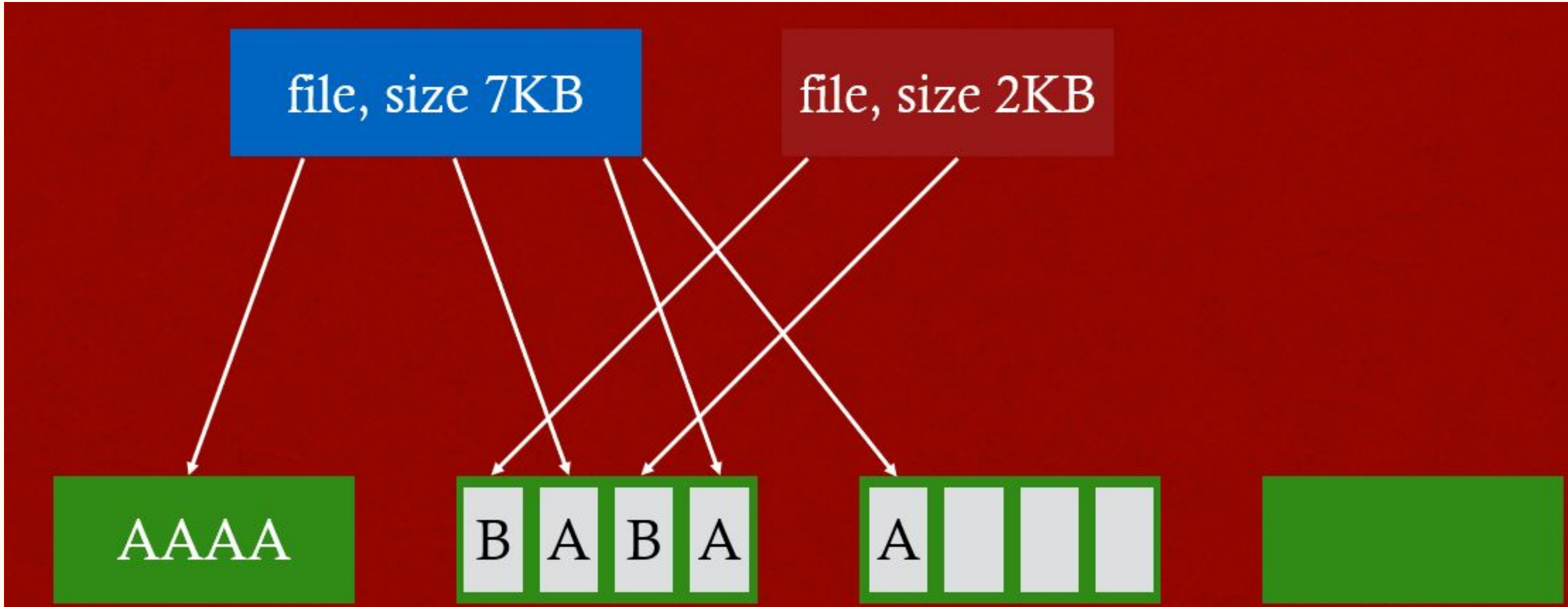
- Solution:
  - hybrid – combine best of large blocks and best of small blocks
- Use large block when file is large enough
- Introduce “fragment” for files that use parts of blocks
- Only tail of file uses fragments


file, size 5KB
file, size 2KB
AAAA
B
A
B



append A to first file

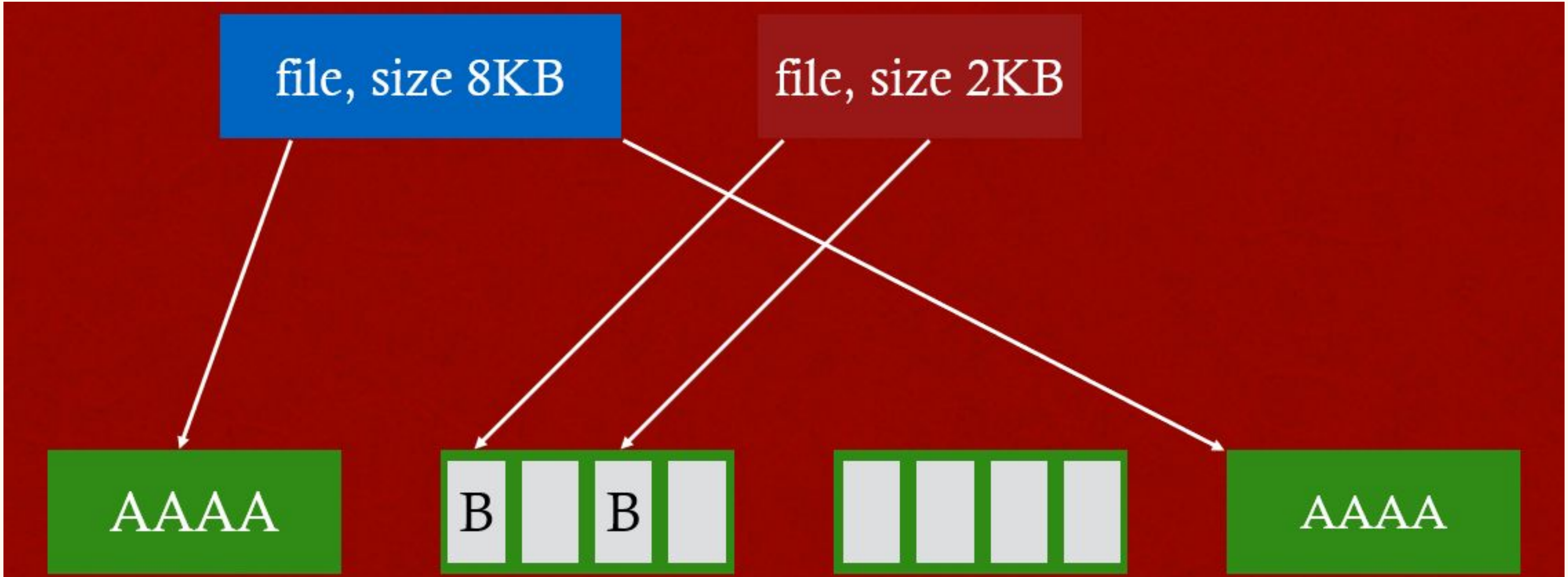

file, size 6KB
file, size 2KB
AAAA
B
A
B
A



append A to first file

Not allowed to use fragments across multiple blocks!

What to do instead?



file, size 8KB

file, size 2KB

AAAA

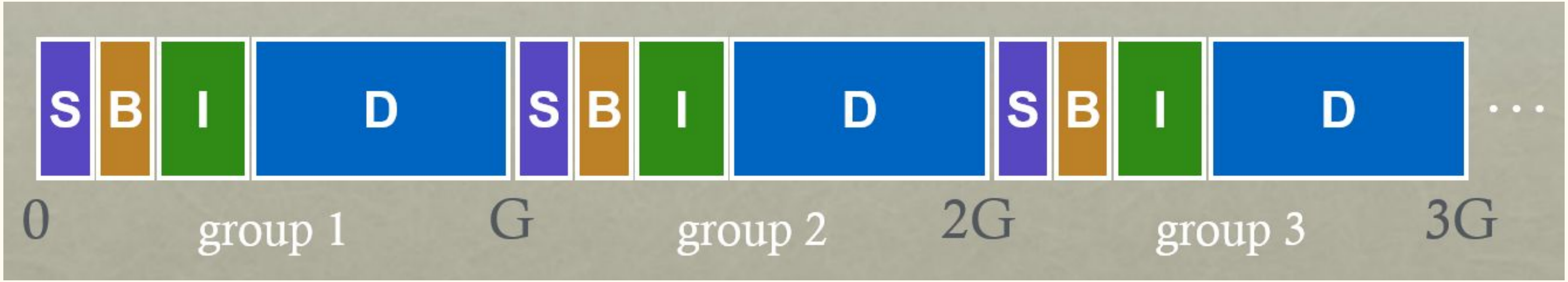
B

B

AAAA

append A to first file,
copy to fragments to new block

Smart Allocation Policy



- Where should new inodes and data blocks go?
    - strategy: put related pieces of data near each other.
- Rules:
    - put directory entries near directory inodes
    - put inodes near directory entries
    - put data blocks near inodes.

Smart Allocation Policy

- Strategy: put related pieces of data near each other.
- Rules:
  - put directory entries near directory inodes
  - put inodes near directory entries
  - put data blocks near inodes.
- Sound good?
- Problem:
  - file system is one big tree
  - all directories and files have a common root
  - all data in same FS is related in some way
- Trying to put everything near everything else

Revised Strategy

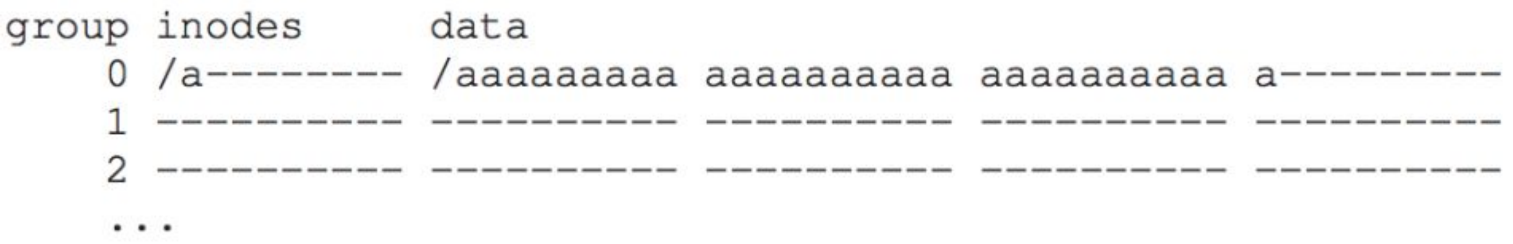
- Put more-related pieces of data near each other
- Put less-related pieces of data **far** from each other

Preferences

- File inodes:
  - allocate in **same** group with dir
- Dir inodes:
  - allocate in **new** group with fewer used inodes than average group
- First data block:
  - allocate **near** inode
- Other data blocks:
  - allocate **near** previous block

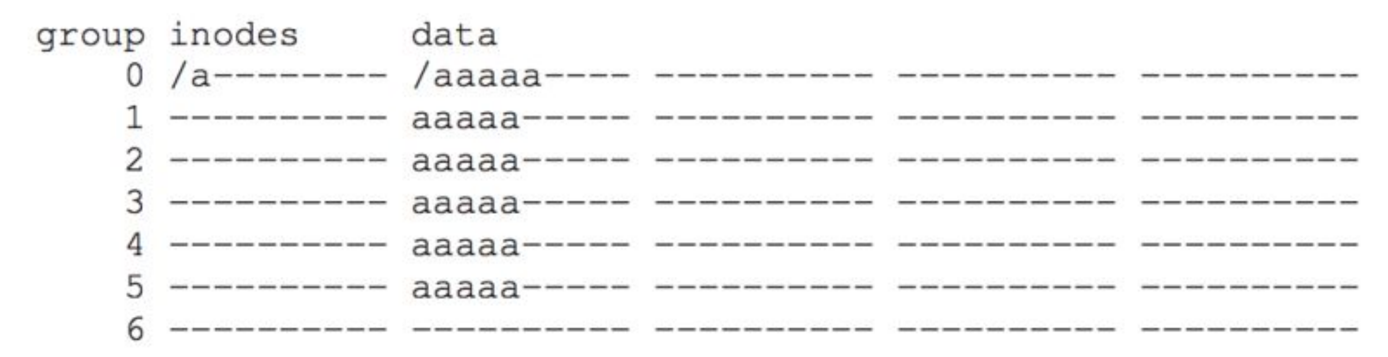
Problem: Large Files

- Single large file can fill nearly all of a group
- Displaces data for many small files
- Better to do one seek for large file
  - than one seek for each of many small files
- Define “large” as requiring an **indirect block**



```
group inodes     data
    0 /a-------- /aaaaaaaaa aaaaaaaaaa aaaaaaaaaa a---------
    1 ---------- ---------- ---------- ---------- ----------
    2 ---------- ---------- ---------- ---------- ----------
    ...
```

Revised Strategy



```
group inodes     data
    0 /a-------- /aaaaa---- ---------- ---------- ----------
    1 ---------- aaaaa----- ---------- ---------- ----------
    2 ---------- aaaaa----- ---------- ---------- ----------
    3 ---------- aaaaa----- ---------- ---------- ----------
    4 ---------- aaaaa----- ---------- ---------- ----------
    5 ---------- aaaaa----- ---------- ---------- ----------
    6 ---------- ---------- ---------- ---------- ----------
```

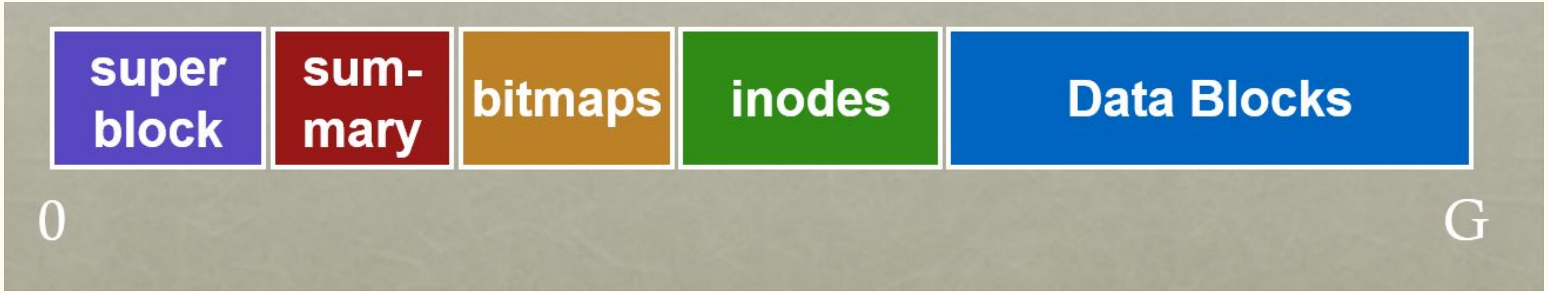
- Large files: where to cut the tree and start growing into another group?
- Starting at indirect (e.g., after 48 KB) put blocks in a new block group.
- Each chunk corresponds to one indirect block
- Block size 4KB, 4 byte per address
  - => 1024 address per indirect block
  - 1024*4KB = 4MB contiguous “chunk”

Preferences

- File inodes:
  - allocate in **same** group with dir
- Dir inodes:
  - allocate in **new** group with fewer used inodes than average group
- First data block:
  - allocate **near** inode
- Other data blocks:
  - allocate **near** previous block
- Large file data blocks:
  - after 48KB, go to **new** group
  - move to **another** group (w/ fewer than avg blocks) every subsequent 1MB.

Group Descriptor (Summary Block)

- How does file system know which new group to pick?
  - summary block: tracks number of free inodes and data blocks



Conclusion

- FFS inspired modern files systems, including ext2 and ext3
- First disk-aware file system
  - bitmaps
  - locality groups
  - rotated superblocks
  - large blocks
  - fragments
  - smart allocation policy
  - ability to rebuild free lists after crash (fsck)

THANK YOU!