File System Consistency and Exam Review

CS439: Principles of Computer Systems

April 6, 2015

Last Time

- File System Implementation
  - Directories
    - Designs
    - How they work
  - Finding files on disk (FFS)
  - Disk Layout
- NTFS
- File System Consistency
  - Sources of Inconsistency
  - Maintaining Consistency/Fixing Inconsistencies

Today's Agenda

- Transactions in the File System
- Journaling File Systems
- Copy on Write File Systems
- RAID
- Exam Review

File System Fault Tolerance

UNIX Approach: Another Problem

- What if we need multiple file operations to occur as a unit?
  - If you transfer money from one account to another, you need to update the two account files as a unit!
- What if we need *atomicity*?

Solution: *Transactions*

Transactions (Review)

- *Transactions* group actions together so that they are:
  - *atomic*: they all happen or they all don't
  - *serializable*: transactions appear to happen one after the other
  - *durable*: once it happens, it sticks
- Critical sections give us atomicity and serializability, but not durability

Achieving Durability (Review)

To get durability, we need to be able to:

- *Commit*: indicate when a transaction is finished
- *Roll back*: recover from an *aborted* transaction
  - If we have a failure in the middle of a transaction, we need to be able to undo what we have done so far
- In other words, we do a set of operations tentatively.
  - If we get to the commit stage, we are okay.
  - If not, roll back operations as if the transaction never happened.

Implementing Transactions (Review)

- Key idea: Turn multiple disk updates into a single disk write!

```
begin transaction
    x = 300
    y = 512
Commit
```

- Keep *write-ahead* (or *redo*) log on disk of all changes in the transaction
- The log records everything the OS does (or tries!) to do
- Once the OS writes both changes on the log, the transaction is committed
- Then *write-behind* changes to the disk, logging all writes
- If the crash comes after a commit, the log is replayed

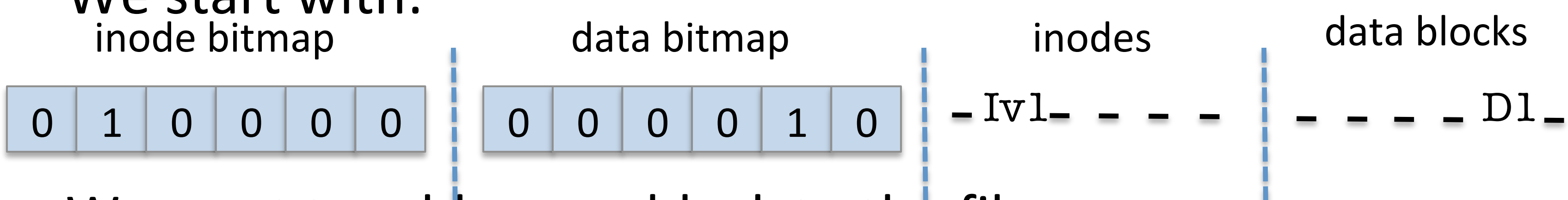
Transactions in File Systems

Most file systems now use *write-ahead logging*

- known as *journaling file systems*
- write all metadata changes to a transaction log before sending any changes to disk
  - file changes are: update directory, allocate blocks, etc.
  - transactions are: create directory, delete file, etc.
- eliminates the need for `fsck` after a crash
- In the event of a crash, read the log.
  - If no log, then all updates made it to disk, do nothing
  - If the log is not complete (no commit), do nothing
  - If the log is completely written (committed), apply any changes that are left to disk

Data Journaling: An Example

- We start with:



- We want to add a new block to the file
- Three easy steps
  - Write to the log 5 blocks: TxBegin | Iv2 | B2 | D2 | TxEnd
    - Write each record to a block, so it is atomic
  - Write the blocks for Iv2, B2, D2 to the FS proper
  - Mark the transaction free in the journal
- What happens if we crash before the log is updated?
  - no commit, nothing to disk---ignore changes!
- What happens if we crash after the log is updated?
  - replay changes in log back to disk

Data Journaling: An Example
Plain Text

- We start with:
  - Inode bitmap: 0 1 0 0 0 0
  - Data bitmap: 0 0 0 0 1 0
  - Inodes: _ [v] _ _ _ _
  - Data blocks: _ _ _ _ D1 _
- We want to add a new block to the file
- 3 easy steps
  - Write to the log 5 blocks: TxBegin | Iv2 | B2 | D2 | TxEnd
    - Write each record to a block, so it's atomic
  - Write the blocks for Iv2, B2, D2 to the FS proper
  - Mark the transaction free in the journal
- What happens if we crash before the log is updated?
  - No commit, nothing to disk---ignore changes!
- What happens if we crash after the log is updated?
  - Replay changes in log back to disk

Journaling and Write Order

- Issuing the 5 writes to the log sequentially is slow `TxBegin | Iv2 | B2 | D2 | TxEnd`
- Issue at once and transform in a single sequential write
- Problem: disk can schedule writes out of order
  - First write `TxBegin`, `Iv2`, `B2`, `TxEnd`
  - Then write `D2`
- Syntactically, transaction log looks fine, even with nonsense in place of `D2`!
- Set a Barrier before `TxEnd`
  - `TxEnd` must block until data on disk

Transactions in File Systems

- Advantages:
  - Reliability
  - Asynchronous write-behind
- Disadvantages:
  - All data is written twice!

Copy-on-Write File Systems

- Data and metadata not updated in place, but written to new location
  - Transforms random writes to sequential writes
- Several motivations
  - Small writes are expensive
  - Small writes are expensive on RAID (more soon)
    - Expensive to update a single block (4 disk I/O) but efficient for entire stripes
  - Caches filter reads
  - Widespread adoption of flash storage
    - Wear leveling, which spreads writes across all cells, important to maximize flash life
    - COW techniques used to virtualize block addresses and redirect writes to cleared erasure blocks
  - Large capacities enable versioning

iClicker Question

Where on disk would you put the journal for a journaling file system?

A. Anywhere
B. Outer rim
C. Inner rim
D. Middle
E. Wherever the inodes are

RAID

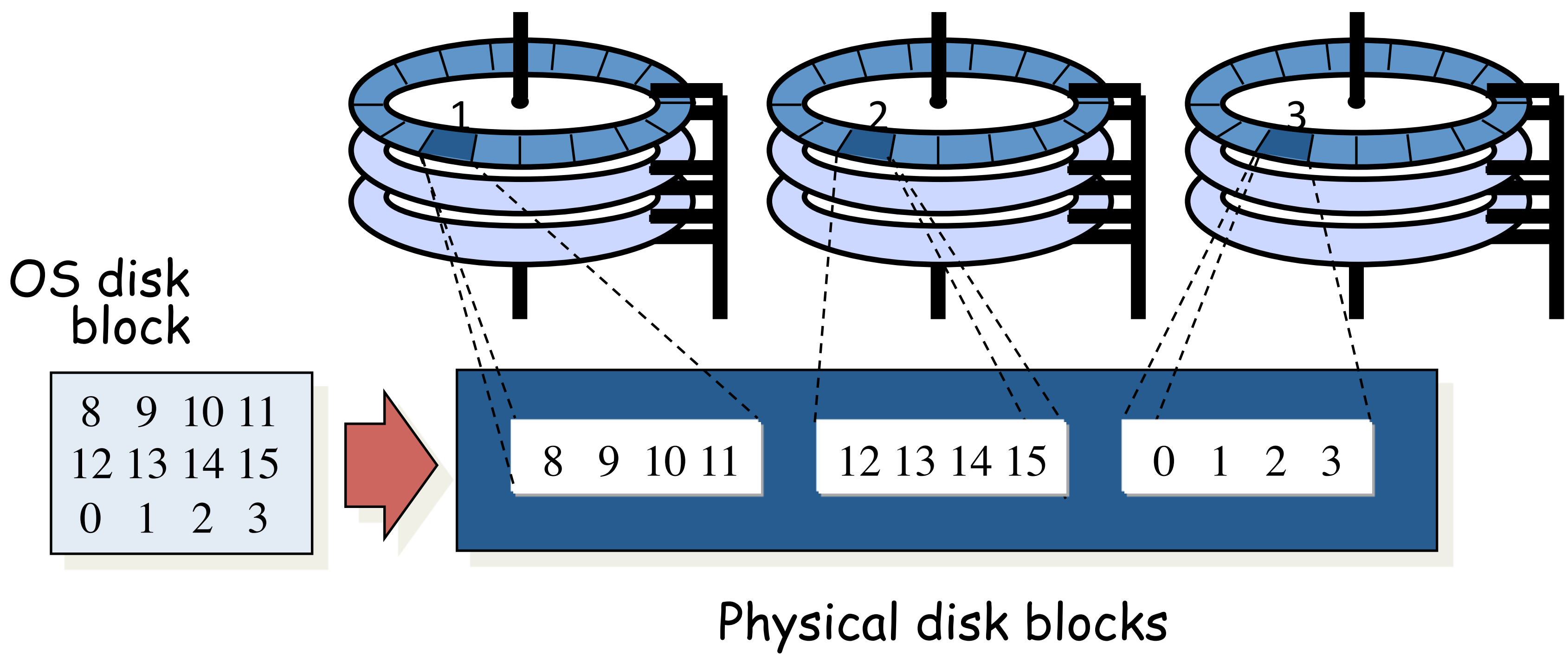
RAID

- *Redundant Array of Inexpensive Disks*
- Disks are cheap, so put many (10s to 100s) of them in one box to increase storage, performance, and availability
- Data plus some redundant information is striped across disks
- Performance and reliability depend on how precisely it is striped
- 5 different levels
  - 0 improves performance
  - 1 improves reliability
  - 3 improves reliability
  - 4 & 5 improve both

RAID-0: Increasing Throughput

Disk striping (RAID-0)

- Blocks broken into sub-blocks that are stored on separate disks
- Higher disk bandwidth
- Poor reliability
  - Failure of a single disk would cause data loss



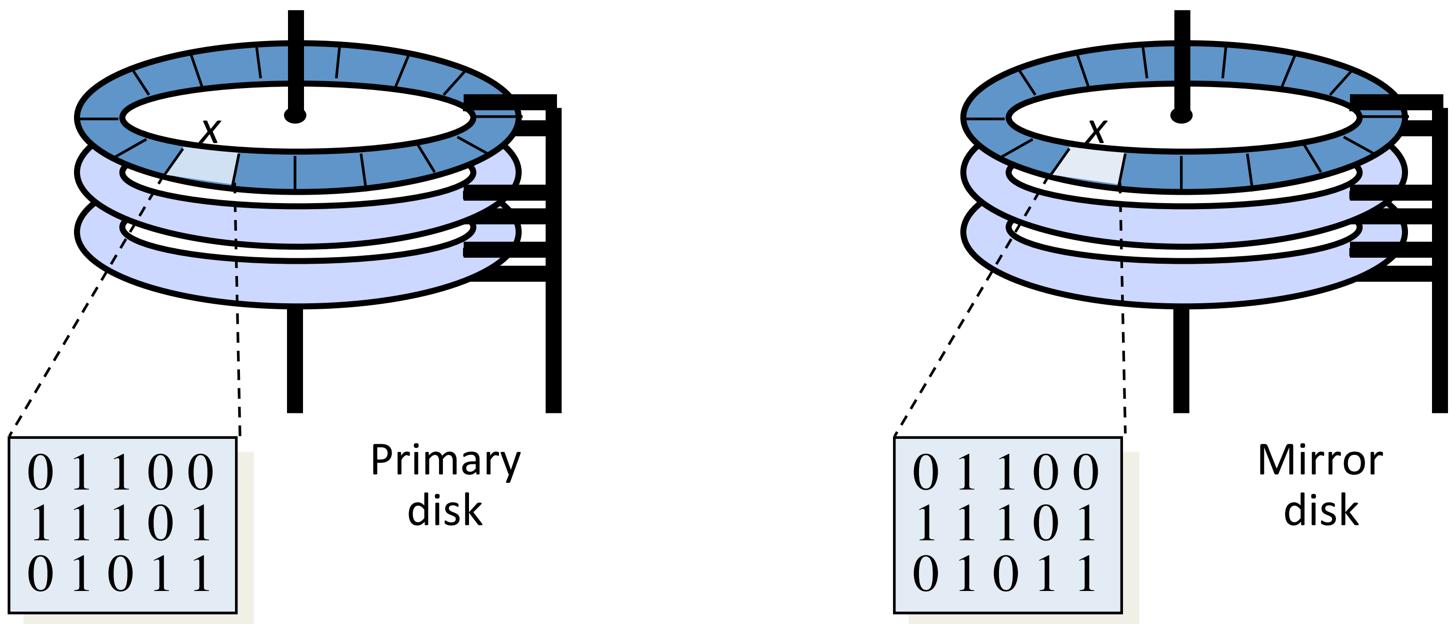
RAID-0: Increasing Throughput Plain Text

- Blocks broken into sub-blocks that are stored on separate disks
- Higher disk bandwidth
- Poor reliability
  - Failure of a single disk would cause the loss of data
- Example:
  - OS disk block that holds data: 8 9 10 11 12 13 14 15 0 1 2 3
    - Information 8 9 10 11 stored on disk 1
    - Information 12 13 14 15 stored on disk 2
    - Information 0 1 2 3 stored on disk 3

RAID-1: Mirrored Disks

To increase disk reliability, we must introduce redundancy

- Simple scheme: *Write to both disks, read from either.*
- On failure, use surviving disk
- Expensive: must write each change twice



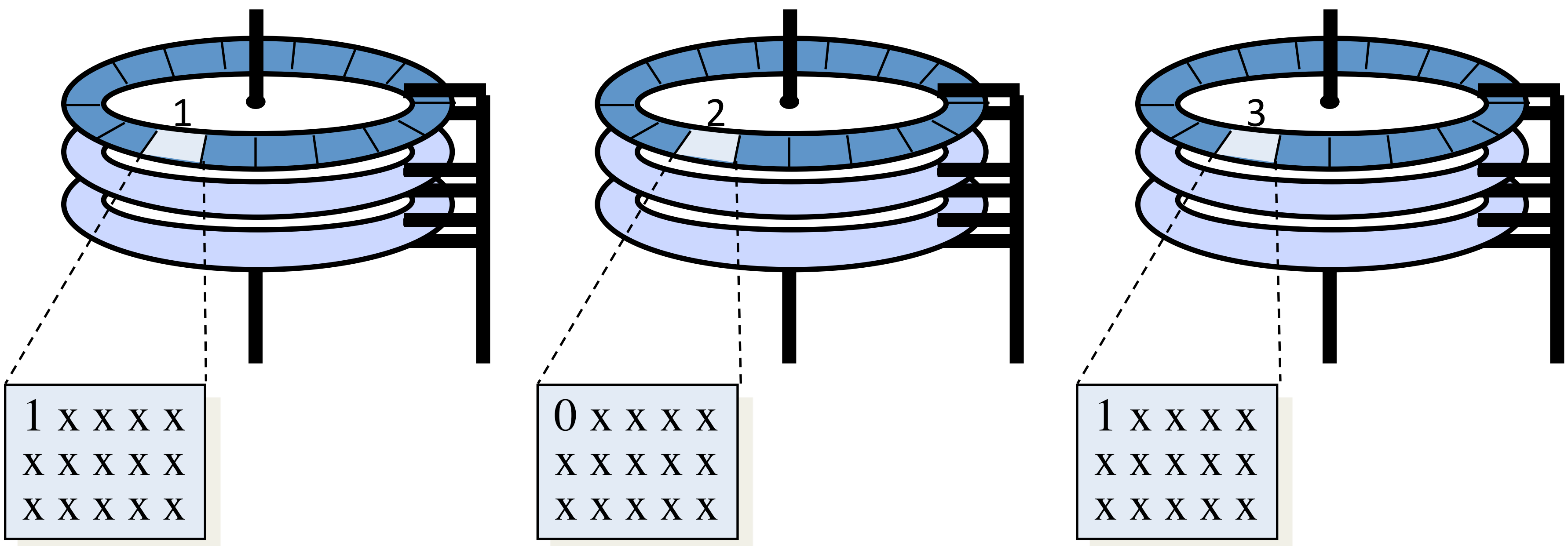
RAID-1: Mirrored Disks Plain Text

To increase disk reliability, we must introduce redundancy

- Simple scheme: write to both disks, read from either
  - Have 2 disks that each hold all the data for the file system
  - Read from whichever has the head closer to the right spot
- On failure, use surviving disk
- Expensive: have to write each change twice
- Disks marked as “primary” and “mirror”

RAID-3

- Byte-striped with parity
  - Bytes written to same spot on each disk
- Reads access all data disks
- Writes accesses all data disks plus parity disk
- Disk controller can identify faulty disk
  - Single parity disk can detect and correct errors
- Example: storing the byte-string 101 in a RAID-3 system



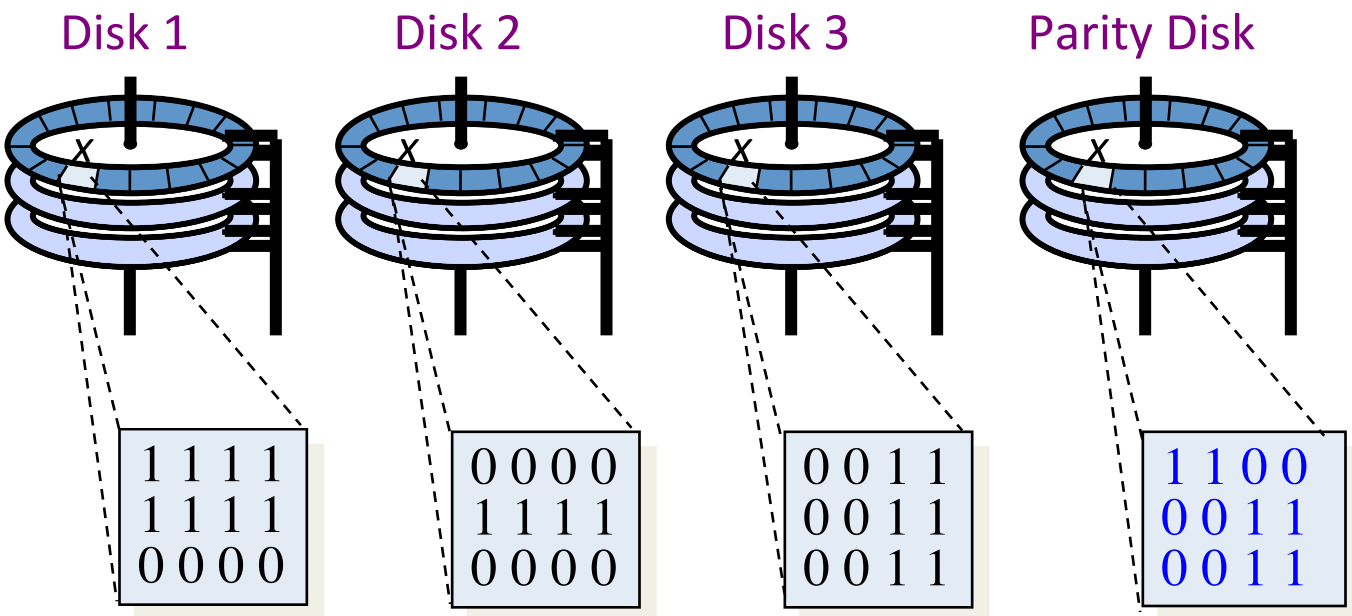
RAID-3: Plain Text

- Byte-striped with parity
  - Bytes written to same spot on each disk
- Reads access all data disks
- Writes access all data disks plus parity disk
- Disk controller can identify faulty disk
  - single parity disk can detect and correct errors
- Example: storing the byte-string 101 in RAID-3 system with four disks
  - Store 1 on disk 1, 0 on disk 2 and the 2nd 1 on disk 3
  - Parity on fourth disk
    - parity – evenness/oddness of the bits in the string

RAID-4

- Block striped with parity
  - Blocks written to same spot on each disk
- Combines RAID-0 and RAID-3
  - Reading a block accesses a single disk
  - Writing always accesses parity disk
    - Heavy load on parity disk
- Disk controller can identify faulty disk
  - Single parity disk can detect and correct errors

RAID-4 layout:



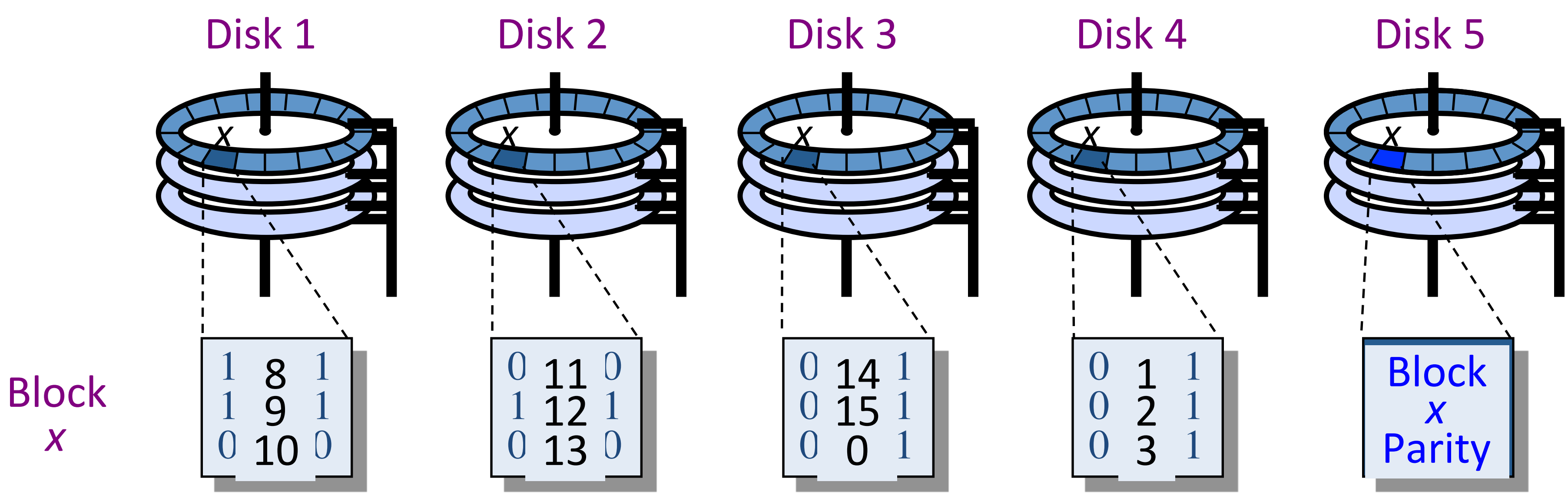
RAID-4: Plain Text

- Block striped with parity
  - Instead of splitting bytes across separate disks you split on block boundaries
  - Blocks written to same spot on each disk
- Combines RAID-0 and RAID-3
  - Reading a block accesses a single block
  - Writing always accesses parity disk
    - Heavy load on parity disk
- Disk controller can identify faulty disk
  - Single parity disk can detect and correct errors

RAID-5

Block Interleaved Distributed Parity

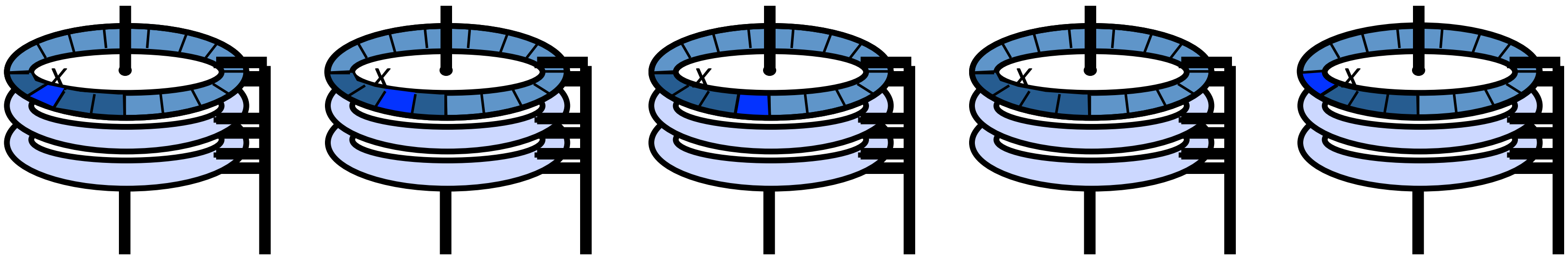
- No single disk dedicated to parity
- Parity and data distributed across all disks



RAID-5: Plain Text

- Block interleaved distributed parity
  - No single disk dedicated to parity
  - Parity and data distributed across all disks
    - So parity bits spread across multiple disks
- Example with 5 disks:
  - 4 blocks written to 4 disks (one to each disk)
  - 5th disk writes parity block
  - Block in same spot on each disk

RAID-5 Example



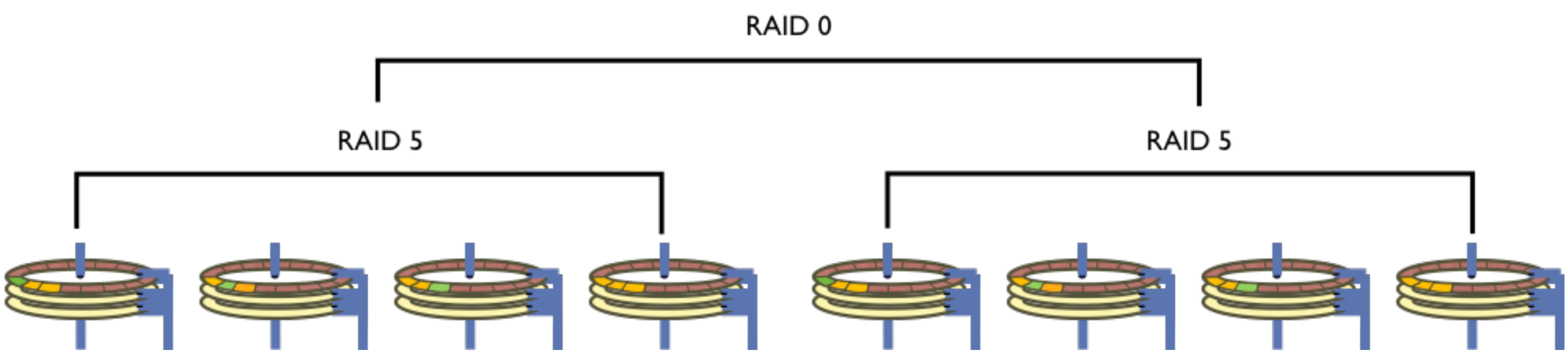
	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
Block *x*	1 8 1 1 9 1 0 10 0	0 11 0 1 12 1 0 13 0	0 14 1 0 15 1 0 0 1	0 1 1 0 2 1 0 3 1	Block *x* Parity
Block *x*+1	Block *x*+1 Parity	0 *a* 0 1 *b* 1 0 *c* 0	0 *d* 1 0 *e* 1 0 *f* 1	0 *g* 1 0 *h* 1 0 *i* 1	1 *j* 1 0 *k* 0 0 *l* 0
Block *x*+2	1 *m* 1 1 *n* 1 0 *o* 0	Block *x*+2 Parity	0 *p* 1 0 *q* 1 0 *r* 1	0 *s* 1 0 *t* 1 0 *u* 1	1 *v* 1 0 *w* 0 0 *x* 0
Block *x*+3	1 *y* 1 1 *z* 1 0 *aa* 0	0 *bb* 0 1 *cc* 1 0 *dd* 0	Block *x*+3 Parity	0 *ee* 1 0 *ff* 1 0 *gg* 1	1 *hh* 1 0 *ii* 0 0 *jj* 0

RAID-5 Example: Text Description

- Has 5 separate disks
  - 4 sets of blocks are written
  - In total, 3 blocks of data + 1 parity block are written to each disk
- First set of blocks:
  - 4 data blocks written to disks 1, 2, 3, 4
  - Parity block written to disk 5
- Second set of blocks:
  - 4 data blocks written to disks 2, 3, 4, 5
  - Parity block written to disk 1
- Third set of blocks:
  - 4 data blocks written to disks 3, 4, 5, 1
  - Parity block written to disk 2
- Fourth set of blocks:
  - 4 data blocks written to disks 4, 5, 1, 2
  - Parity block written to disk 3
- Note that in this example, disk 4 does not write a parity block. If the example were to be extended by one data set, it would be disk 4's turn.

RAID-10 and RAID-50

- RAID-10
  - stripes (RAID-0) across reliable logical disks, implemented as mirrored disks (RAID-1)
- RAID-50
  - stripes (RAID-0) across groups of disks with block interleaved distributed parity (RAID-5)



RAID-10 and RAID-50: Plain Text

- RAID-10
  - Stripes (RAID-0) across reliable logical disks, implemented as mirrored disks (RAID-1)
- RAID-50
  - Stripes (RAID-0) across groups of disks with block interleaved distributed parity (RAID-5)
  - Example: Write is striped (RAID-0) to two sets of disks implemented RAID-5.

Summary

Transactions can be used to provide atomicity in the file system.

Exam Review and Procedures

Exam Review

He who asks is a fool for five minutes; he who does not ask remains a fool forever.

- *Anonymous Chinese Proverb*

iClicker Question

What might be on the exam?

A. Information from lectures and reading

B. Coding questions

C. Concept questions (general understanding/ thought)

D. All of the above (and more!)

Exam Procedures

- Arrive on time
  - No one may start the exam after the first person leaves
- Bring your UT ID
- Find your EID and assigned seat on the chart outside the classroom
- Do not enter the room until told to do so
- When you enter, proceed to your seat

Exam Procedures

- Leave all extra paper, electronics, hats, etc. in your bag.
- Do not begin the exam until told to do so
- Raise your hand to ask questions
- When finished,
  - turn in exam and all scratch paper to myself or the proctor
  - present your ID.

iClicker Question

What should you bring to the exam?

A. A writing utensil and your ID

B. Nothing

My Best Advice

Do NOT panic!

You have been taught how to do each question, and you can do it.

Announcements

- Exam Wednesday!
  - UTC 2.122A 7p-9p
- Class on Wednesday is shortened and optional
  - 10a-11a and 12p-1p
  - Review sessions (driven by your questions!)
  - Any student may attend either section
- No discussion sections this week
- My Wednesday office hours are canceled
- Project 3 is posted due Friday, 4/17