Quiz 3: The Final Frontier

- **9-10:30AM on Thursday, May 19, 2005**
- **Johnson Ice Rink**
- All material from Lecture 16 (April 4) through Recitation 26 (May 12)
- Bring your notes!
- Print out the Unison paper and bring it.

Atomicity Concepts

Chapters 9 and 10

LFS, System R, Chocolate, Unison, Durability

LFS: Motivation

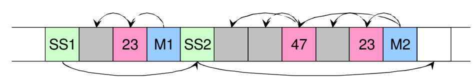
RAM is cheap, so:

→ The buffer cache will be large

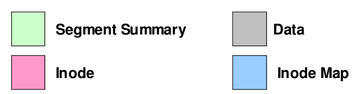
→ Reads will be “absorbed” by the buffer cache

→ Let’s design a filesystem that makes writes *really* fast

LFS: On-Disk Layout




```
mkdir(“/etc”, 0);
fd = open(“/etc/group”, O_RDWR | O_CREAT);
write(fd, buf, 5000);
```



LFS: Observations

- LFS uses checkpoints to decrease recovery time
  - Checkpoint region points to all blocks in the inode map
- LFS outperforms SunOS FS for
  - Small writes
  - Many file creates

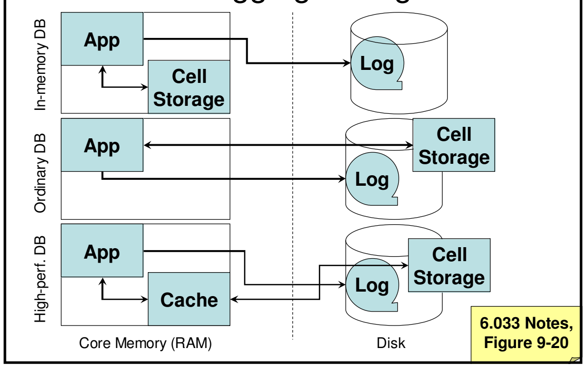
LFS: Coping With a Finite Disk

- Divide disk into **segments** of size *s*
  - Time to write *s* bytes >> rotational + seek latency
  - *s* << buffer cache size
- Idea is to write whole segments at once
- **Cleaner** runs periodically
  - Bottom line: no one really knows the cleaning overhead

Database Terminology

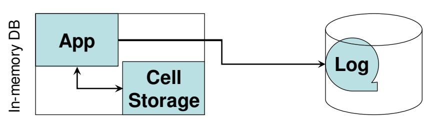
6.033 Definition	System R Terminology	Meaning
Recoverable	"Atomic"	Do it all or not at all.
Isolated	"Consistent"	Do it all before or all after.
Atomic	N/A	Recoverable and isolated.
Consistent	N/A	App-specified invariant is preserved.

Common Logging Configurations



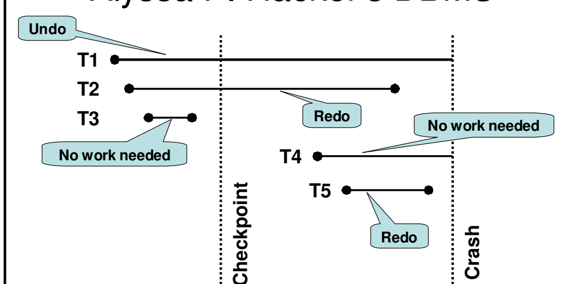
Alyssa P. Hacker's DBMS

- On-disk log records transactions
- Reference copy of all data in RAM
- Checkpoint: write entire database state to the log
- Recovery: start from last chpkted state



6.033 Notes, Problem 9.9b

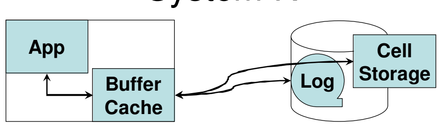
Alyssa P. Hacker's DBMS



What if T3 nested in T2, T2 nested in T1?

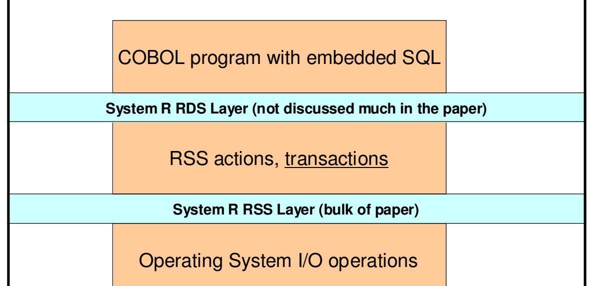
Time

System R

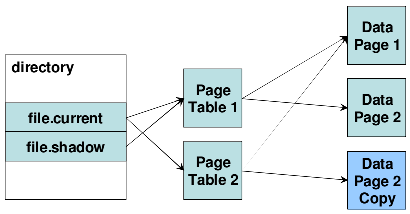


- Take-home design points:
  - System R uses shadow files <u>and</u> write-ahead logging (WAL) to make transactions recoverable and isolated
  - Writes go through the buffer cache, flushed to disk when necessary

System R



System R Shadow Files



- FILE SAVE: file.shadow ← file.current
- FILE RESTORE: file.current ← file.shadow

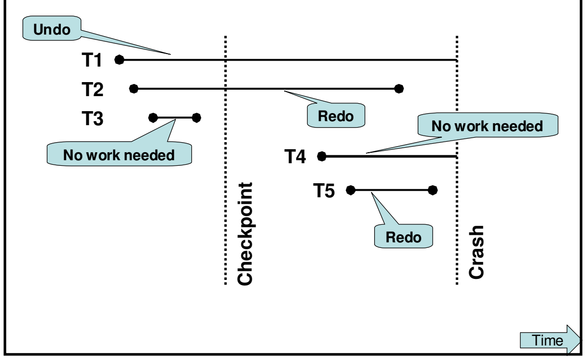
System R Write-Ahead Logging

- Commit
- Checkpoint
- How is write-ahead logging useful?
- "Golden Rule" of Recoverability
  - Never modify the only copy of data

System R Checkpoint and Recovery

- Checkpoint:
  - Write checkpoint log record
  - FILE SAVE every shadow file
  - Remember log address of checkpoint record
- Recovery:
  - FILE RESTORE files to their shadowed versions
  - Determine losers, winners
  - Undo or redo as necessary

System R Recovery



IBM IMS Database System

- Version 1 (1968) Isolation Protocol
  - A transaction may read only data that has been written by previously committed transactions.
  - A transaction must acquire a lock for every data item that it will **write**.

6.033 Notes, Problem 9.3

IBM IMS Database System

Initially, x=3 and y=4

Intent: T1 assigns y=x; T2 assigns x=y

1 BEGIN (*T1*);	*1* BEGIN (*T2*);
2 ACQUIRE (*lock* **of** *y*);	*2*
3 *temp1* ← *x*;	*3*
4	*4* ACQUIRE (*lock* **of** *x*);
5	*5* *temp2* ← *y*;
6	*6* *x* ← *temp2*;
7 *y* ← *temp1*;	*7*
8 COMMIT (*T1*);	*8*
9	*9* COMMIT (*T2*);

Values after this execution completes? Have we achieved isolation?

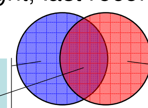
Atomicity Concepts

Chapters 9 and 10

LFS, System R, Chocolate, **Unison**, Durability

Reconciling Two Filesystems

- Quiesce the filesystems to be reconciled
- Given *left*, *right*, last reconcile time=*lrt*



if (modtime(left) > lrt) then
 copy left to right
else
 delete left

if (modtime(right) > lrt) then
 copy right to left
else
 delete right

if (modtime(left) > lrt AND modtime(right) > lrt) then
 /* conflict */
else if (modtime(left) > lrt) then
 copy left to right
else if (modtime(right) > lrt) then
 copy right to left

6.033 Notes, Chapter 10D

Unison

- Reconciles a file system on a remote host
- Optimistic vs. pessimistic concurrency control
- State-based vs. log-based concurrency control
- Detecting changes
  - Modification time
  - inode number
  - Cryptographic fingerprint

Protecting Information

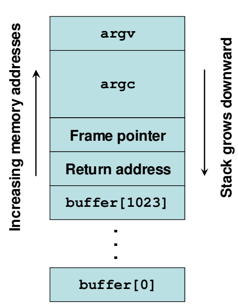
Chapter 11 and Appendices, **Slammer**, DoS, Reflections on Trusting Trust, Why Cryptosystems Fail, Lampsons's Hints for System Design

Slammer: Buffer Overruns

How <u>not</u> to read input into your program:

```
int main(int argc,
         char **argv) {
  char buffer[1024];
  gets(buffer);
  return 0;
}
```

This program could have read input from the network instead of the keyboard, resulting in a <u>remote</u> exploit!



Slammer: Design and Lessons

- Slammer exploited a similar buffer overrun in MS SQL Server 2000
- Very simple exploit program
  - Send identical attack packets to random IPs, as fast as possible
- Exponential attack rate
- Lesson to users: close unused ports
- Lesson to OS vendors: be secure by default

Internet Denial of Service

- TCP SYN flooding
  - Solution: SYN cookies: push burden onto client
- Reflectors
  - ICMP Smurf attack
    - Solution: ingress filtering