ECE566
Enterprise Storage Architecture

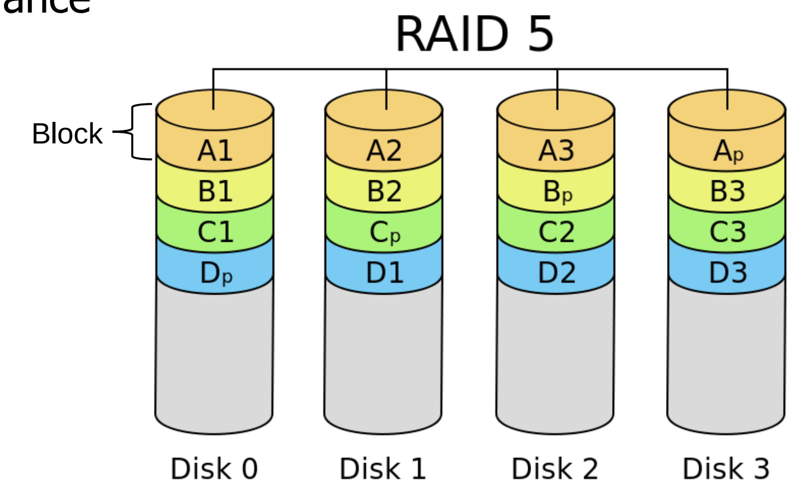
Spring 2025

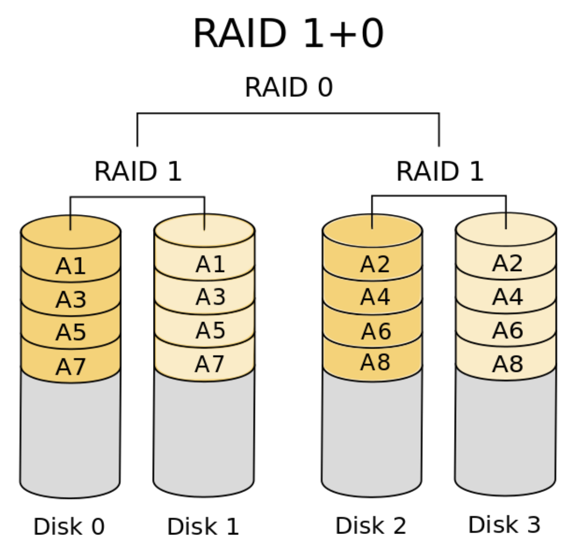
The Rest-of-Course Overture
(Preparation for your project proposal)

Tyler Bletsch
Duke University

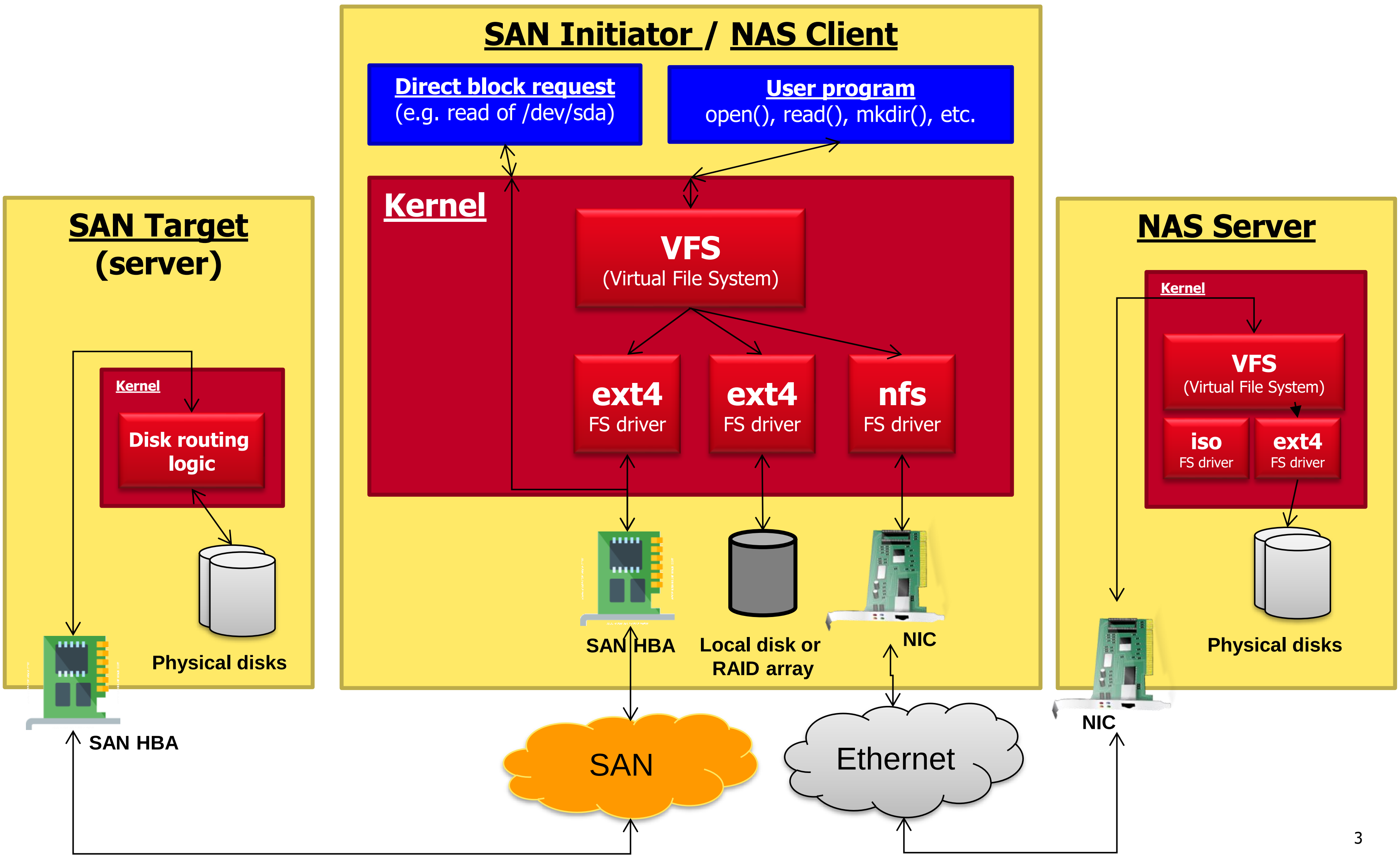
RAID

- Combine disks,
  - **Striping** to make aggregate scale in performance
  - **Redundancy** to survive failures
- RAID levels
  - RAID 0: Striping
  - RAID 1: Mirroring
  - RAID 4: Parity
  - RAID 5: Distributed parity
  - RAID 6: Dual parity
  - RAID 10, 50, 60, etc.: Combinations



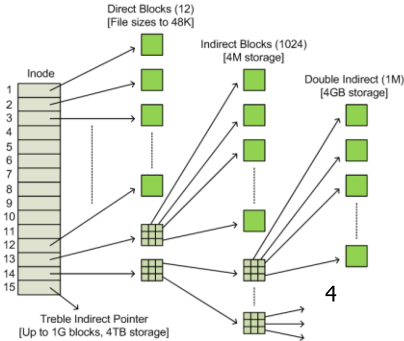


NAS and SAN block diagram



Filesystems

- Take open/close/read/write/mkdir/rm/etc, translate to read block / write block
- Responsibilities:
  - Allocation among files (files are created, grown, shrunk, destroyed)
    - Identify and manage *free* blocks
  - Metadata, including security (owner, timestamp, permissions, etc.)
  - Directory hierarchy
- Key filesystem innovations:
  - **Inode**-based layout (good efficiency/scalability)
  - **Journaling** (recover from crashes safely)
  - **Logging** (high-efficiency writes by appending everything)
  - **Indirected** designs (snapshots, deduplication, etc.)



Storage efficiency

- Find ways to put fewer bytes on disk while still satisfying all IO requests

More efficient RAID

Snapshot/clone

Zero-block elimination

Thin provisioning

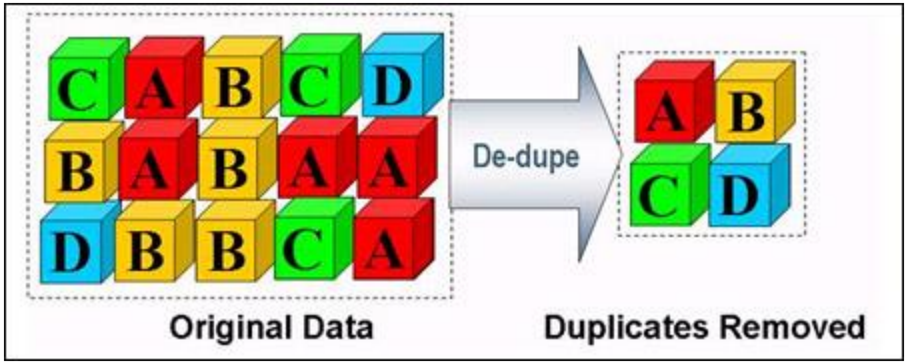
Deduplication

Compression

"Compaction" (partial zero block elimination)

Deduplication

- Identify redundant data; only store it once

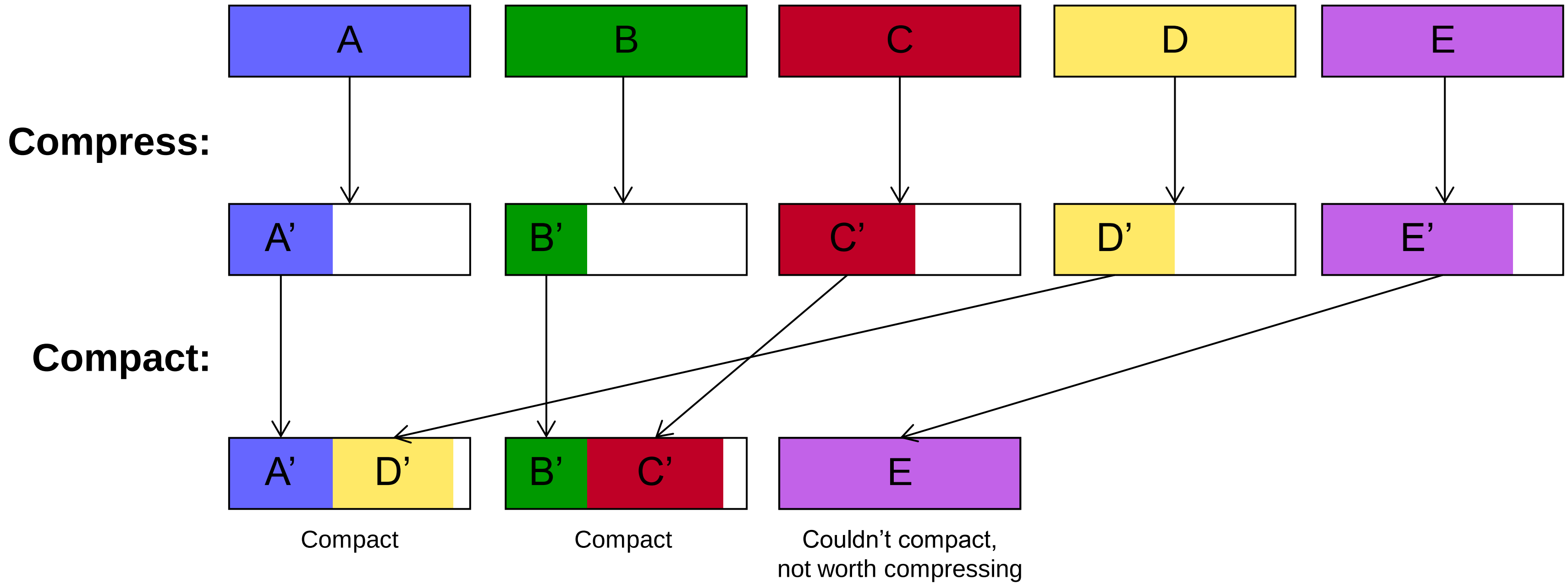


- Simplified algorithm:
  - Split the file in to chunks
  - Hash each chunk with a big hash
  - If hashes match, data matches:
    - Replace this with a reference to the matching data
  - Else:
    - It's new data, store it.
- Lots of design decisions to look at in the details…

Figure from http://www.eweek.com/c/a/Data-Storage/How-to-Leverage-Data-Deduplication-to-Green-Your-Data-Center/

Compression with compaction

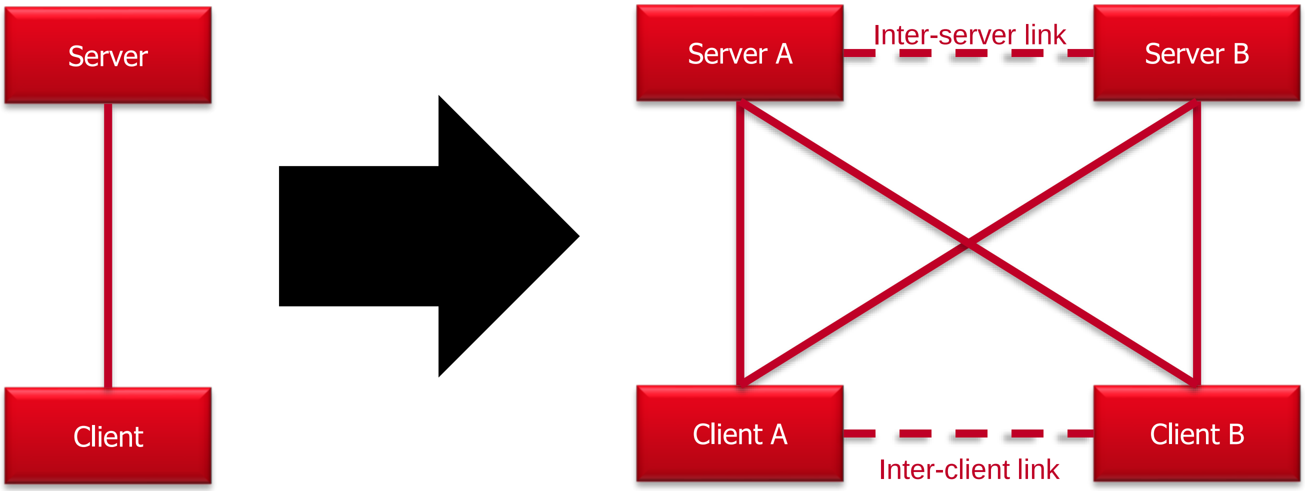
- Compression with simple compaction



- Data block pointers are now {block_num, offset, length}

High availability

- Eliminate single points of failure!
  - Disk failure → RAID redundancy
  - Server failure → Server clustering
  - Link failure → Multipathing
  - Etc…
- Interesting part is *how* the system works now that there's 2+ of whatever there used to be one of…



Disaster recovery

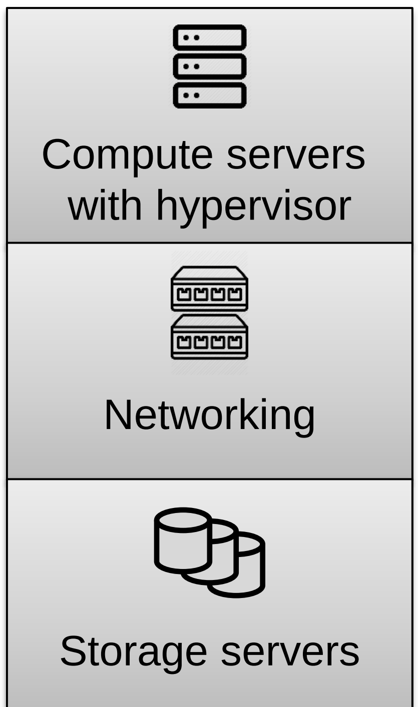
- If our high availability redundancy is overwhelmed, that's a disaster.
- How to recover?
    - Keep extra hardware (easy)
    - Keep good backups (harder)
- Backups must:
    - Be **non-modifiable** and record changes **over time**, in a **separate place**, **automatically**, with **separate credentials**, with continuous **reports/alerts** and **testing**.

Storage Array – Source site

Storage Array – Remote site

Virtualization

- Virtualize each layer of stack to pool resources; individual systems stop mattering
- Fundamental concept: **aggregate physically** and **separate logically**



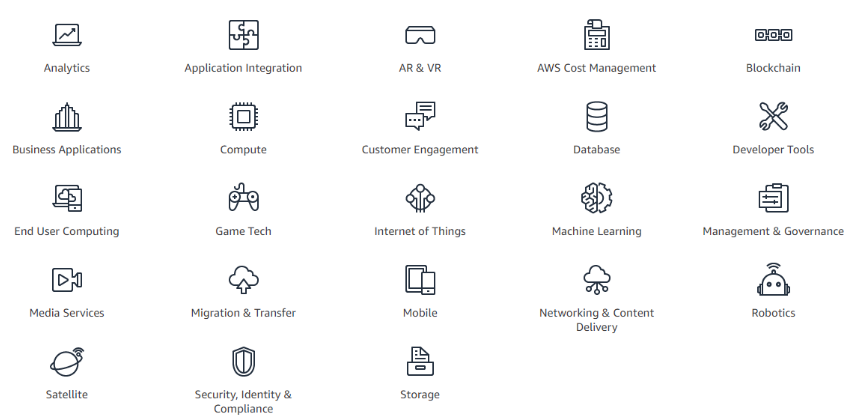
Aggregate: Cluster disk-less interchangeable servers
Separate: Run virtual machines (VMs) that can freely migrate

Aggregate: Switches paired and interconnected with cables
Separate: Virtual LANs (VLANs) separate traffic flows

Aggregate: Disks combined with RAID and linear mapping
Separate: Logical volumes created on top

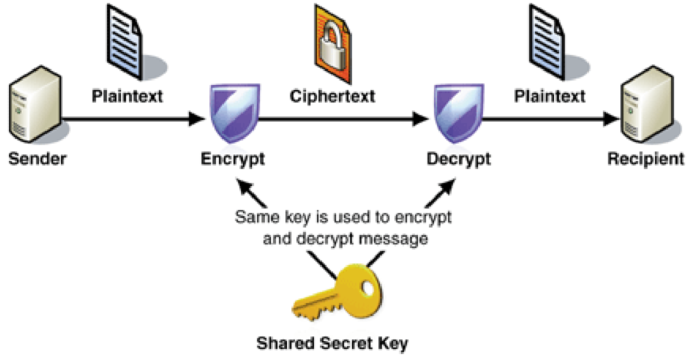
Cloud

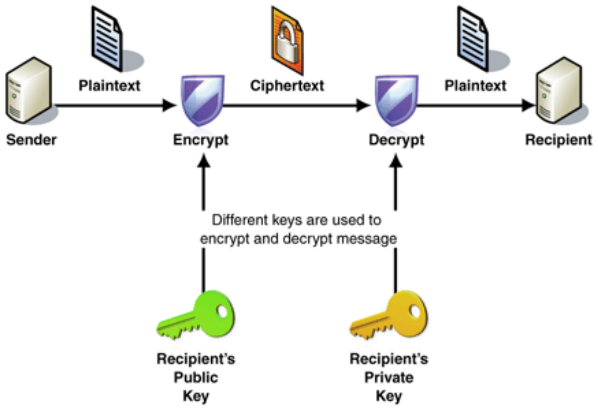
- Basically the virtualization stuff, but:
  - You're careful with separation security
  - You rent pieces of the stack to users (either internal or external)
- Variety of cloud services out there – many ripe for an interesting project!
  - Traditional Infrastructure-as-a-Service providers (Amazon, Azure, Vultr, Linode, Digital ocean, etc.)
  - Amazon S3 (object storage)
  - Amazon EBS and EFS (Amazon's SAN and NAS offerings)
  - Amazon has a ton of weird/specific offerings too...



Security

- Kinds of encryption:
  Secret key (symmetric) & Public key (asymmetric)





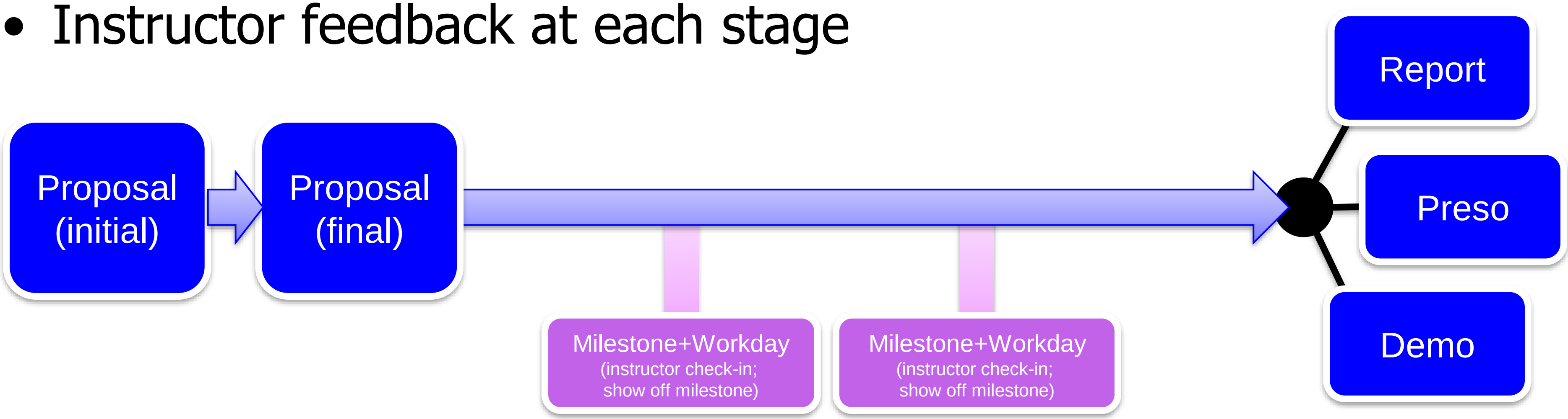
- SEPARETELY, two main places to use encryption:
  In-flight (on network link) & At rest (on disk)

- Also have to worry about <u>authentication</u> (who are you?) and <u>access control</u> (are you allowed to do that?)

Course project discussion

The course project

- Half-semester effort in some area of storage
- Several choices (plus choose-your-own)
- Instructor feedback at each stage



- Any stage can result in a need for **resubmission** (grade withheld pending a second attempt).

- **See [course site project page](#) for details**

Example projects

- **Availability/recoverability**
  - RAID at the filesystem level
  - Mirroring to second system (or cloud?)
- **Network-accessibility**
  - Make a network filesystem
  - Store to cloud service
- **Storage efficiency**
  - Filesystem deduplication
  - Filesystem compression
- **Performance***
  - Minimal-seek on disk data structures
  - Caching with read-ahead
  - Hybrid SSD+HDD filesystem
- **Security**
  - Access control list support
  - Per-user at-rest file encryption

*** Hard to realize benefit with FUSE/BUSE**

Wildcard projects

- Special purpose file system (e.g. MP3 transcoding)
- Custom block device instead of file system:
  - Custom RAID
  - Custom SAN
  - Block-level encryption
  - Block-level compression
  - Block-level deduplication

Project idea
Write-once file system

Write-once file system (WOFS)

- Normal file system
  - Read/write
  - Starts empty, evolves over time
  - Simplest implementation isn't simple
    - Fragmentation and indirection

- Write-once file system
  - Read-only
  - Starts "full", created with a body of data
  - Simple implementation
    - No fragmentation, little indirection

What is a WOFS for?

- CD/DVD images
  - *"Master" the image with the content in /mydir*
    ```
    $ mkisofs -o my.iso /home/user/mydir
    ```
  - *Write the disc image directly onto the burner*
    ```
    $ cdrecord my.iso
    ```
- Ramdisk images (e.g. cramfs, squashfs, etc.)

Major parts of a WOFS

- **Mastering program:**
  `$ mkwofs myfilesystem.img data/`

- **Mounting program (FUSE):**
  `$ wofsmount myfilesystem.img dir/`
  `$ ls dir/`
  …

- *Mounting program must not "extract" data at load time – data is retrieved from the image as read requests are handled!*

Project idea

Dropbox “Smart sync” support for Linux

Project idea
Network file system with caching

Network File System without Special Sauce

- Simple idea:
  Put IO system calls over the network

- Complex consequences:
  - Stateful or stateless?
  - Caching? Cache coherency?
  - What server? How many servers?
  - Data compression?
  - Data reduction, e.g. "Low-bandwidth File System" (http://pdos.csail.mit.edu/papers/lbfs:sosp01/lbfs.pdf)

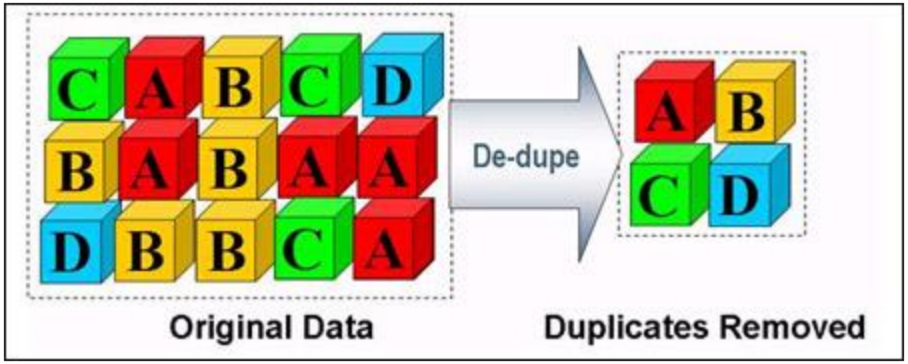
An *interesting* network file system

- A basic network filesystem is basic OS stuff
- Yours must also have one of:
  - Read caching and write-behind caching
  - Read caching and read-ahead optimization
  - Distributed storage over multiple servers
  - Compression
  - “Low-bandwidth file system” features
    - (Persistent disk cache, basically dedupe-on-the-wire)
  - Something else?

Project idea
Deduplication

Deduplication

- Will be covered later, here's the short version



- Split the file in to chunks
- Hash each chunk with a big hash
- If hashes match, data matches:
  - Replace this with a reference to the matching data
- Else:
  - It's new data, store it.

Figure from http://www.eweek.com/c/a/Data-Storage/How-to-Leverage-Data-Deduplication-to-Green-Your-Data-Center/

Common deduplication data structures

- Metadata:
  - Directory structure, permissions, size, date, etc.
  - Each file's contents are stored as a **list of hashes**
- Data pool:
  - A flat table of hashes and the data they belong to
  - Must keep a reference count to know when to free an entry

Design decisions

- **Eager** or **lazy**?

- **Fixed-** or **variable-sized** blocks?
  - Variable size via Rabin-Karp Fingerprinting

<u>Project idea</u>
Special-case file system

Special-case file system

- Sometimes “general purpose” is *too* general

- Example motivations:
  - Can we exploit a workload’s peculiar access pattern?
  - Can we examine the data to present new organizational structures?
  - Can we map non-filesystem information into the file system?

Tips to keep in mind

- Performance: Disk seeks are the enemy!
  - Often, “Minimize seeks” = “Optimize performance”
- Metadata: Many files have metadata not usually exposed to the file system, such as JPEG EXIF tags, MP3 ID3 tags, DOC/DOCX author tags, etc.
- Anything can be a filesystem. You can have a file system represent:
  - A git server
  - An email account
  - A web server
  - A physical system (e.g. “Internet of Things”)
  - A database (e.g. via the Duke registration system public API)
  - More!

Project idea

File system performance survey

File system performance survey

- Storage systems are enormously complex with many pieces affecting overall performance
  - Filesystem (ext3, ntfs, etc.)
  - Filesystem configuration (journaling, alignment, etc.)
  - Workload (benchmarks)
  - Underlying devices (SSD, HDD, and also RAID)
- It is useful to characterize how different configurations perform under different workloads

How to approach the problem

- Get hardware
  - Such as **your server**!!
- Define your test variables
- Build a test harness
  - Automate all testing, it will run for days!
  - Automate data collation – don't scrape numbers by hand!
  - Get it all into a giant spreadsheet
- Data mining – find knowledge in the data
- Detailed write up of interesting conclusions

<u>Project idea</u>
Hybrid HDD/SSD system

Hybrid storage

- SSD is expensive per GB, cheap for random IO performance
- HDD is the opposite

- Can develop a software that gets best of both worlds

- Examples:
  - SSD as cache for HDD
  - SSD as write buffer for HDD
  - Auto-migrate "hot" data to SSD, "cold" data to HDD
  - Identify random workloads, migrate to SSD
- Mechanism:
  - File system (e.g. with FUSE)
  - Virtual block device (e.g. via BUSE)

Evaluation

- Must include:
  - Benchmark of your system against pure HDD and pure SSD systems.
  - Measurement of FUSE overhead
  - Cost/benefit analysis based on HDD and SSD costs
  - All of the above must be conducted against a good cross-section of workloads

Project idea
Storage workload characterization

Storage workload capture

- In storage sizing, need to characterize workload
- Workload may be confidential or too complex to migrate

- Project: Use a technique to *record* a storage workload
  - Example 1: take a trace of read/write ops; need to *anonymize*, then be able to replay operations with equivalent performance
  - Example 2: monitor I/O ops, characterize nature of workload, then be able to simulate a request stream with similar characteristics

- Will need to prove the accuracy of your technique with statistical analysis across variety of workloads

<u>Project idea</u>
Cloud storage tiering

Cloud storage tier

- Cloud storage (e.g. Amazon S3) is useful, generally pretty cheap
- Downside: internet latency and bandwidth

- Can develop a storage system which migrates “cold” or otherwise lower-priority data out to a cloud service, brings it back live on demand without user interaction
  - Optional enhancements:
    - Intelligent prediction algorithm for migration
    - Encryption for cloud-exported data
    - Compression for cloud-exported data
  - Can be implemented at block level or file system level

BRAINSTORMING

Brainstorming

- Take an existing storage paradigm
  - Local storage (DAS)
  - NAS
  - SAN
  - RAID
  - Cloud storage (e.g. S3)
  - Cluster filesystems
- …or take one of the project ideas given.

- SCAMPER it

SCAMPER

S	**S**ubstitute
C	**C**ombine
A	**A**dapt
M	**M**odify / **M**agnify
P	**P**urpose
E	**E**liminate
R	**R**earrange / **R**everse

Where did that lead you?