# CS152
# Computer Architecture and Engineering
# Lecture 24

## I/O and Storage Systems

April 29, 2004

John Kubiatowicz (www.cs.berkeley.edu/~kubitron)

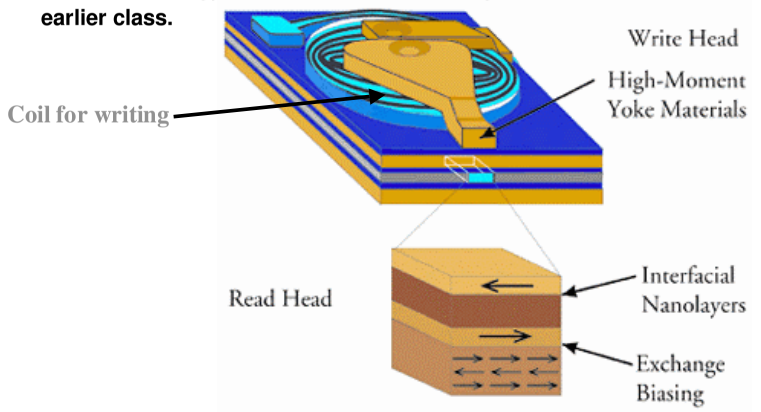
lecture slides: http://inst.eecs.berkeley.edu/~cs152/

## Recap: Nano-layered Disk Heads

° Special sensitivity of Disk head comes from "Giant Magneto-Resistive effect" or (GMR)

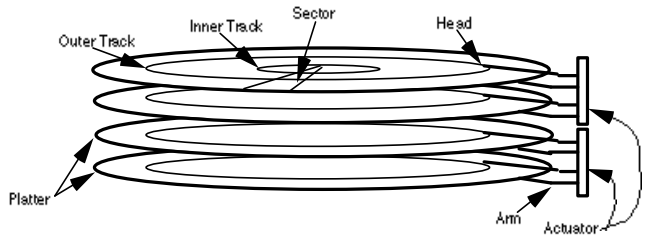
° IBM is leader in this technology

- Same technology as TMJ-RAM breakthrough we described in earlier class.






## Recap: Disk Device Terminology




Disk Latency = Queueing Time +
Controller time +
Seek Time + Rotation Time + Xfer Time

*Order of magnitude times for 4K byte transfers:*

Average Seek: 8 ms or less

Rotate: 4.2 ms @ 7200 rpm

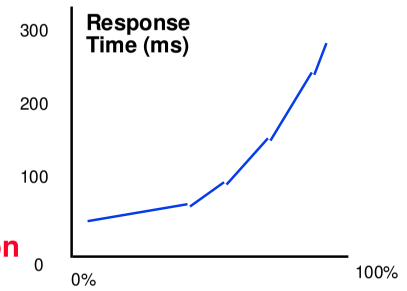
Xfer: 1 ms @ 7200 rpm

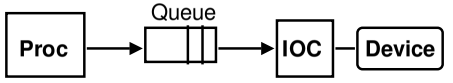


## Disk I/O Performance

Metrics:
Response Time
Throughput

latency goes as $T_{ser} \times u/(1-u)$
u = utilization




Response time = Queue + Device Service time

## Introduction to Queueing Theory

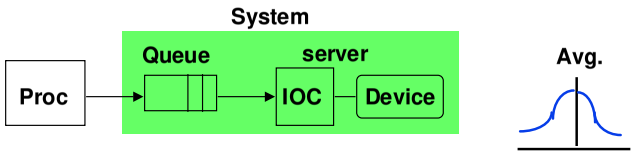
° Queueing Theory applies to long term, steady state behavior ⇒ Arrival rate = Departure rate

° Little's Law:
Mean number tasks in system = arrival rate x mean reponse time
- Observed by many, Little was first to prove
- Simple interpretation: you should see the same number of tasks in queue when entering as when leaving.

° Applies to any system in equilibrium, as long as nothing in black box is creating or destroying tasks

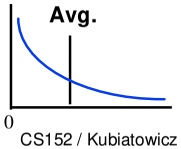


## A Little Queuing Theory: Use of random distributions



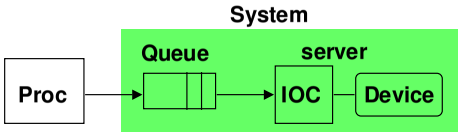

° Server spends a variable amount of time with customers
- Weighted mean $m1 = (f1 \times T1 + f2 \times T2 + ... + fn \times Tn)/F = \Sigma\ p(T) \times T$
- $\sigma^2 = (f1 \times T1^2 + f2 \times T2^2 + ... + fn \times Tn^2)/F - m1^2 = \Sigma\ p(T) \times T^2 - m1^2$
- *Squared coefficient of variance*: $C = \sigma^2/m1^2$
  - Unitless measure (100 ms² vs. 0.1 s²)

° Exponential distribution $C = 1$ : most short relative to average, few others long; 90% < 2.3 x average, 63% < average
Hypoexponential distribution $C < 1$ : most close to average, C=0.5 => 90% < 2.0 x average, only 57% < average
Hyperexponential distribution $C > 1$ : further from average C=2.0 => 90% < 2.8 x average, 69% < average

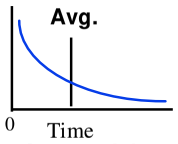





## A Little Queuing Theory: Variable Service Time




° Disk response times $C \approx 1.5$ (majority seeks < average)
° Yet usually pick $C = 1.0$ for simplicity
- Memoryless, exponential dist
- Many complex systems well described by memoryless distribution!




° Another useful value is average time must wait for server to complete current task: $m1(z)$
- *Called "Average Residual Wait Time"*
- Not just 1/2 x m1 because doesn't capture variance
- Can derive $m1(z) = 1/2 \times m1 \times (1 + C)$
- *No variance ⇒ C= 0 => m1(z) = 1/2 x m1*
- *Exponential ⇒ C= 1 => m1(z) = m1*



## A Little Queuing Theory: Average Wait Time

° Calculating average wait time in queue $T_q$:
- All customers in line must complete; avg time: $m1 \equiv T_{ser} = 1/\mu$
- If something at server, it takes to complete on average $m1(z)$
  - Chance server is busy = $u = \lambda/\mu$; average delay is $u \times m1(z)$

$T_q = u \times m1(z) + L_q \times T_{ser}$
$T_q = u \times m1(z) + \lambda \times T_q \times T_{ser}$ — Little's Law
$T_q = u \times m1(z) + u \times T_q$ — Defn of utilization (u)
$T_q \times (1 - u) = m1(z) \times u$
$T_q = m1(z) \times u/(1-u) = T_{ser} \times \{1/2 \times (1+C)\} \times u/(1 - u))$

**Notation:**

| | |
|---|---|
| $\lambda$ | average number of arriving customers/second |
| $T_{ser}$ | average time to service a customer |
| $u$ | server utilization (0..1): $u = \lambda \times T_{ser}$ |
| $T_q$ | average time/customer in queue |
| $L_q$ | average length of queue: $L_q = \lambda \times T_q$ |
| m1(z) | average residual wait time = $T_{ser} \times \{1/2 \times (1+C)\}$ |

## A Little Queuing Theory: M/G/1 and M/M/1

° Assumptions so far:
- System in equilibrium
- Time between two successive arrivals in line are random
- Server can start on next customer immediately after prior finishes
- No limit to the queue: works First-In-First-Out
- Afterward, all customers in line must complete; each avg $T_{ser}$

° Described "memoryless" or Markovian request arrival (M for C=1 exponentially random), General service distribution (no restrictions), 1 server: *M/G/1 queue*

° When Service times have C = 1, *M/M/1 queue*

$$T_q = T_{ser} \times u / (1 - u)$$

$T_{ser}$ average time to service a customer
$u$ server utilization (0..1): $u = \lambda \times T_{ser}$
$T_q$ average time/customer in queue



## A Little Queuing Theory: An Example

° Processor sends 10 x 8KB disk I/Os per second, requests & service exponentially distrib., avg. disk service = 20 ms
- This number comes from disk equation: Service time = Ave seek + ave rot delay + transfer time + ctrl overhead

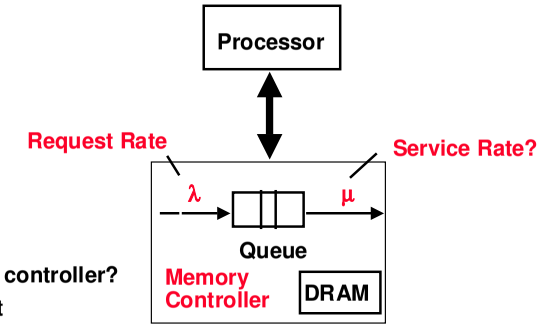
° On average, how utilized is the disk?
- What is the number of requests in the queue?
- What is the average time spent in the queue?
- What is the average response time for a disk request?

° Notation:

$\lambda$ average number of arriving customers/second = 10
$T_{ser}$ average time to service a customer = 20 ms (0.02s)
$u$ server utilization (0..1): $u = \lambda \times T_{ser}$ = 10/s x .02s = 0.2
$T_q$ average time/customer in queue = $T_{ser} \times u / (1 - u)$
= 20 x 0.2/(1-0.2) = 20 x 0.25 = 5 ms (0 .005s)
$T_{sys}$ average time/customer in system: $T_{sys} = T_q + T_{ser}$ = 25 ms
$L_q$ average length of queue: $L_q = \lambda \times T_q$
= 10/s x .005s = 0.05 requests in queue
$L_{sys}$ average # tasks in system: $L_{sys} = \lambda \times T_{sys}$ = 10/s x .025s = 0.25



## Memory System I/O Performance




° Pipelined Bus with queue at controller?
- Time to transfer request
- $T_{queue}$= Queueing Delay+service time
- Time to transfer data

° DRAM has DETERMINISTIC service time
- $T_{ser} = t_{RAC} + (n-1) * t_{PC} + t_{precharge}$
- $T_q = m1(z) \times u/(1-u) = T_{ser} \times \{1/2 \times (1+C)\} \times u/(1 - u))$ with C=0



## Administrivia

° Go to the "Projects" link and describe your project (By Friday)

° Thursday: Sections in lab again (119 Cory)
- Discuss your design document
- Send draft to TA tonight

° Midterm II next Wednesday
- 5:30 – 8:30 in 306 Soda Hall
  - Pizza afterwards
- Topics
  - Pipelining
  - Caches/Memory systems
  - Buses and I/O (Disk equation)
  - Queueing theory
- Can bring 1 page of notes and calculator
  - Handwitten, double-sided (CLOSED BOOK!)

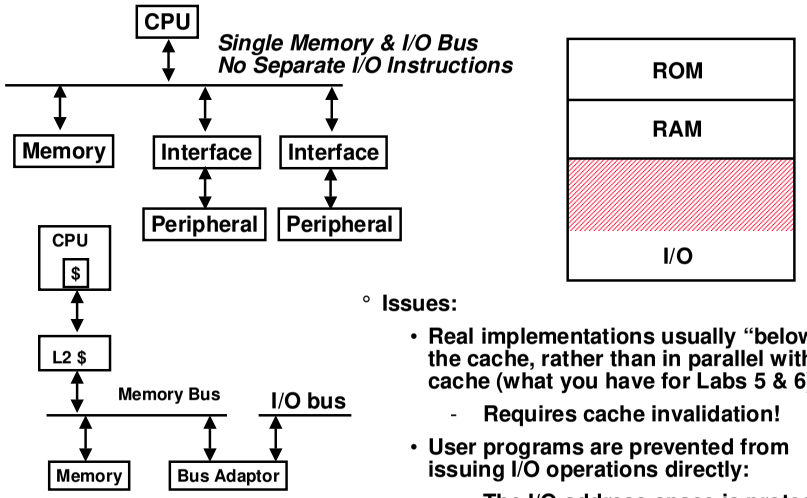
° Oral Report
- Powerpoint
- 15 minute presentation, 5 minutes for questions

## Giving Commands to I/O Devices

- Two methods are used to address the device:
  - Special I/O instructions
  - Memory-mapped I/O
- Special I/O instructions specify:
  - Both the device number and the command word
    - Device number: the processor communicates this via a set of wires normally included as part of the I/O bus
    - Command word: this is usually send on the bus's data lines
- Memory-mapped I/O:
  - Portions of the address space are assigned to I/O device
  - Read and writes to those addresses are interpreted as commands to the I/O devices



## Memory Mapped I/O




- Issues:
  - Real implementations usually "below" the cache, rather than in parallel with the cache (what you have for Labs 5 & 6)
    - Requires cache invalidation!
  - User programs are prevented from issuing I/O operations directly:
    - The I/O address space is protected by the address translation

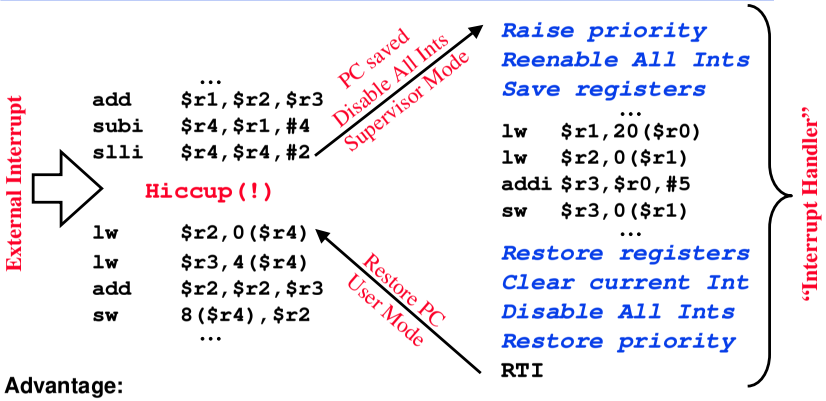


## I/O Device Notifying the OS

- The OS needs to know when:
  - The I/O device has completed an operation
  - The I/O operation has encountered an error
- This can be accomplished in two different ways
  - I/O Interrupt:
    - Whenever an I/O device needs attention from the processor, it interrupts the processor from what it is currently doing.
  - Polling:
    - The I/O device put information in a status register
    - The OS periodically check the status register

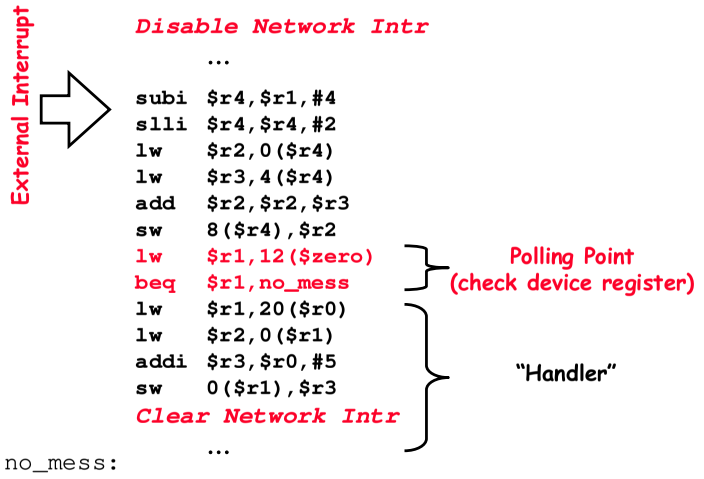


## Example: Device Interrupt




- Advantage:
  - User program progress is only halted during actual transfer
- Disadvantage, special hardware is needed to:
  - Cause an interrupt (I/O device)
  - Detect an interrupt (processor)
  - Save the proper states to resume after the interrupt (processor)

## Alternative: Polling





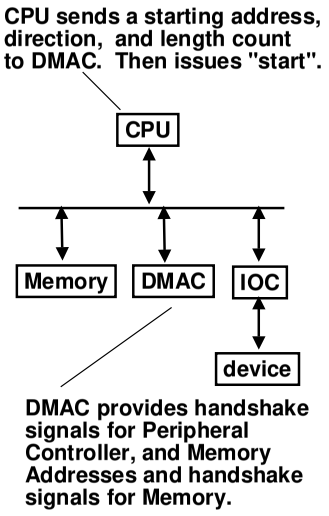

## Polling is faster/slower than Interrupts

- Polling is faster than interrupts because
  - Compiler knows which registers in use at polling point. Hence, do not need to save and restore registers (or not as many).
  - Other interrupt overhead avoided (pipeline flush, trap priorities, etc).
- Polling is slower than interrupts because
  - Overhead of polling instructions is incurred regardless of whether or not handler is run. This could add to inner-loop delay.
  - Device may have to wait for service for a long time.
- When to use one or the other?
  - Multi-axis tradeoff
    - Frequent/regular events good for polling, *as long as device can be controlled at user level.*
    - Interrupts good for infrequent/irregular events
    - Interrupts good for ensuring regular/predictable service of events.



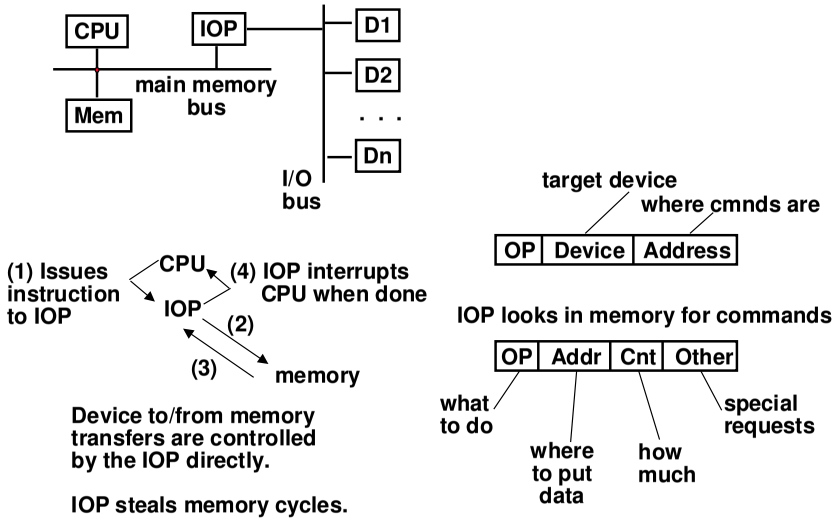
## Delegating I/O Responsibility from the CPU: DMA

- Direct Memory Access (DMA):
  - External to the CPU
  - Act as a maser on the bus
  - Transfer blocks of data to or from memory without CPU intervention






## Delegating I/O Responsibility from the CPU: IOP

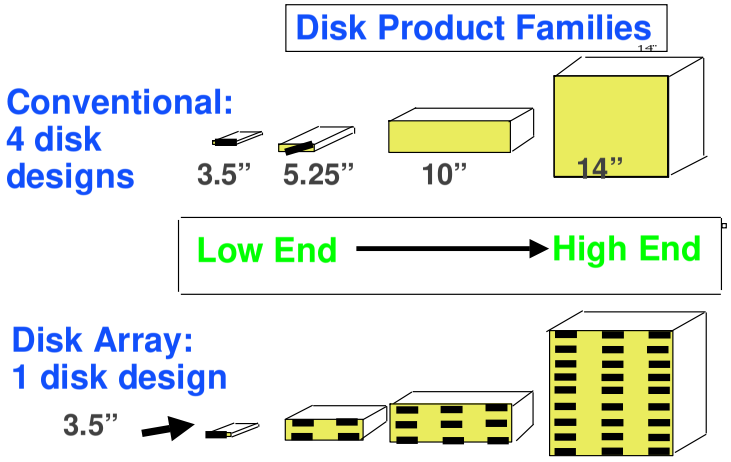


## Reliability and Availability

- Two terms that are often confused:
  - Reliability: Is anything broken?
  - Availability: Is the system still available to the user?
- Availability can be improved by adding hardware:
  - Example: adding ECC on memory
- Reliability can only be improved by:
  - Better environmental conditions
  - Building more reliable components
  - Building with fewer components
    - Improve availability may come at the cost of lower reliability
- Durability: Will the data last forever?



## Manufacturing Advantages of Disk Arrays






## Array Reliability

- Reliability of N disks = Reliability of 1 Disk ÷ N

  50,000 Hours ÷ 70 disks = 700 hours

  Disk system MTTF: Drops from 6 years to 1 month!

- Arrays (without redundancy) too unreliable to be useful!

**Hot spares support reconstruction in parallel with access: very high media availability can be achieved**



## *Redundant* Arrays of Disks

- Files are "striped" across multiple spindles
- Redundancy yields high data availability

  Disks will fail

  Contents reconstructed from data redundantly stored in the array

  → Capacity penalty to store it

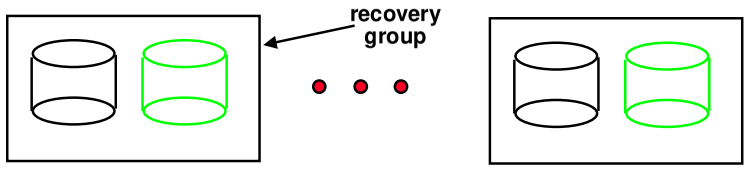
  → Bandwidth penalty to update

*Techniques:*

- Mirroring/Shadowing (high capacity cost)
- Horizontal Hamming Codes (overkill)
- Parity & Reed-Solomon Codes
- Failure Prediction (no capacity overhead!)

*VaxSimPlus — Technique is controversial*

## RAID 1: Disk Mirroring/Shadowing

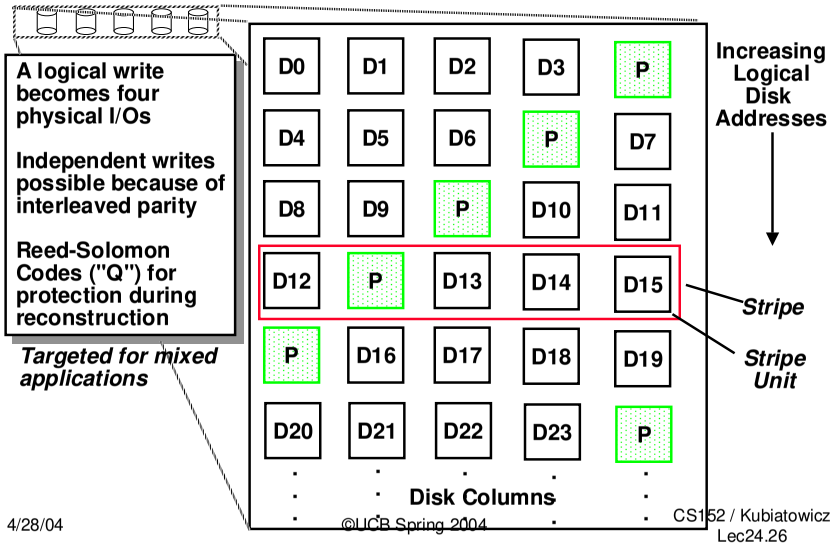



- Each disk is fully duplicated onto its "shadow"
  Very high availability can be achieved
- Bandwidth sacrifice on write:
  Logical write = two physical writes
- Reads may be optimized
- Most expensive solution: 100% capacity overhead

*Targeted for high I/O rate , high availability environments*



## RAID 5+: High I/O Rate Parity

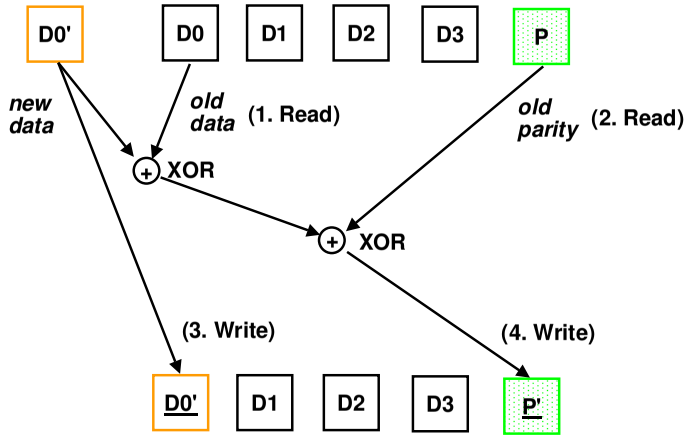





## Problems of Disk Arrays: Small Writes

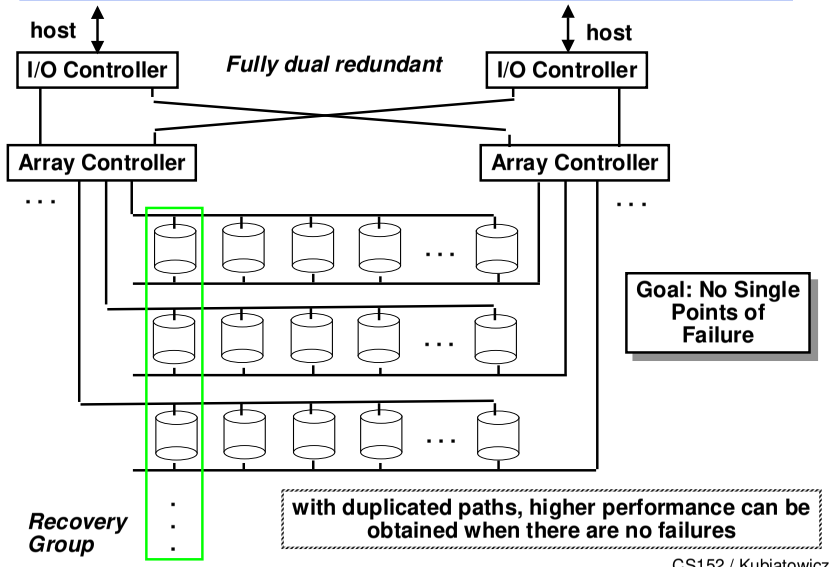
*RAID-5: Small Write Algorithm*

1 Logical Write = 2 Physical Reads + 2 Physical Writes

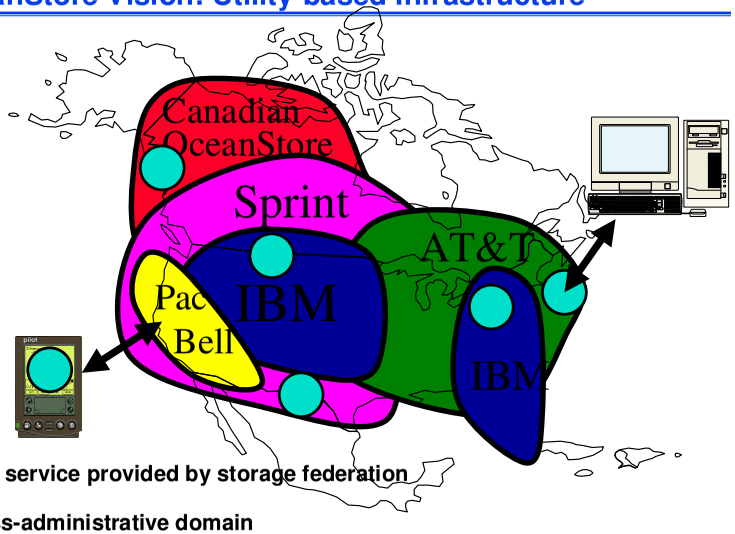





## System-Level Availability



## OceanStore Vision: Utility-based Infrastructure




- Data service provided by storage federation
- Cross-administrative domain
- Contractual Quality of Service ("someone to sue")



## What are the advantages of a utility?

- For Clients:
  - Outsourcing of Responsibility
    - Someone else worries about quality of service
  - Better Reliability
    - Utility can muster greater resources toward durability
    - System not disabled by local outages
    - Utility can focus resources (manpower) at security-vulnerable aspects of system
  - Better data mobility
    - Starting with secure network model⇒sharing
- For Utility Provider:
  - Economies of scale
    - Dynamically redistribute resources between clients
    - Focused manpower can serve many clients simultaneously

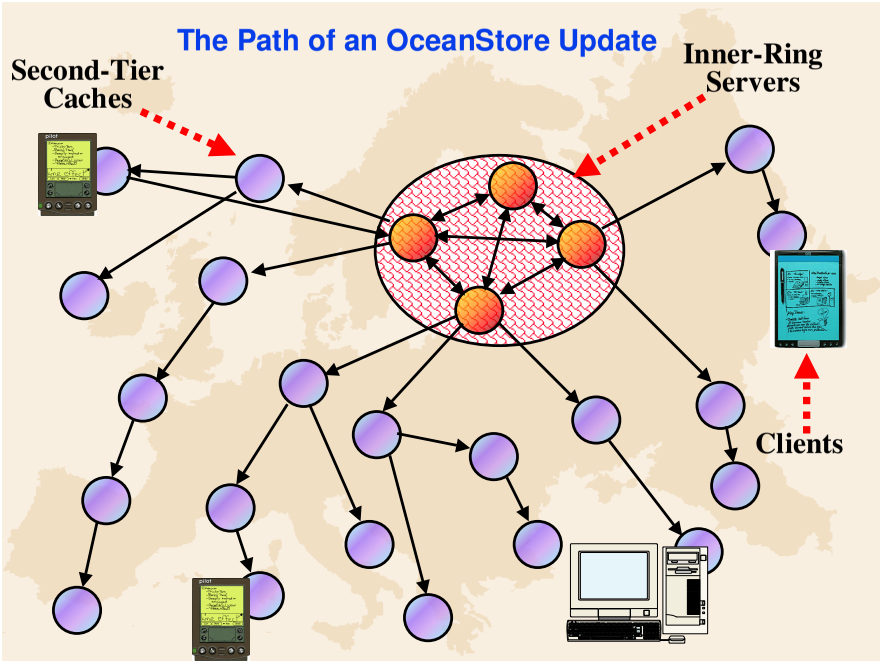


## Two Types of OceanStore Data

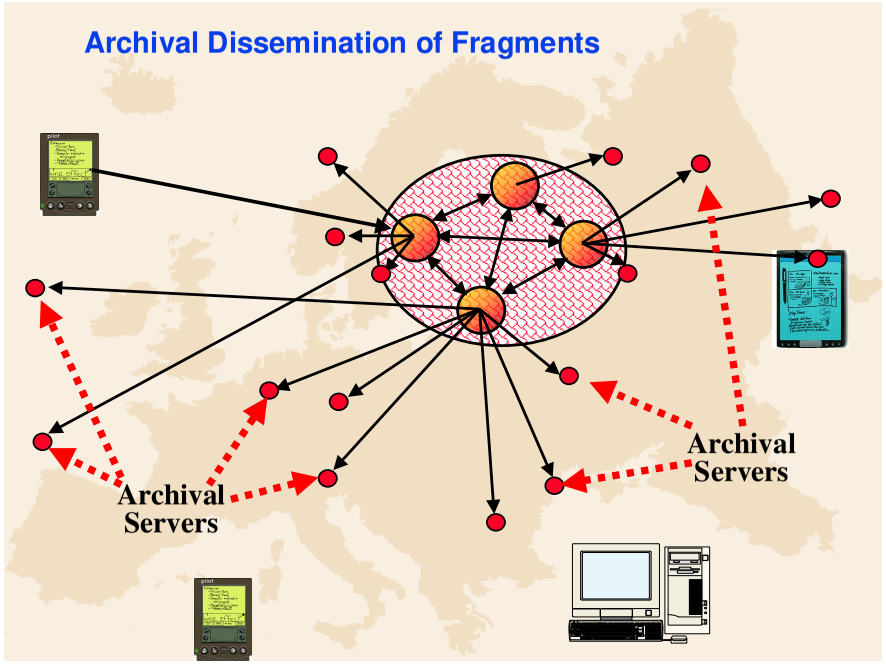
- *Active Data:* "Floating Replicas"
  - Per object virtual server
  - Interaction with other replicas for consistency
  - May appear and disappear like bubbles
- *Archival Data:* OceanStore's Stable Store
  - m-of-n coding: Like hologram
    - Data coded into *n* fragments, any *m* of which are sufficient to reconstruct (e.g m=16, n=64)
    - Coding overhead is proportional to n÷m (e.g 4)
  - Fragments are cryptographically self-verifying
- Most data in the OceanStore is archival!



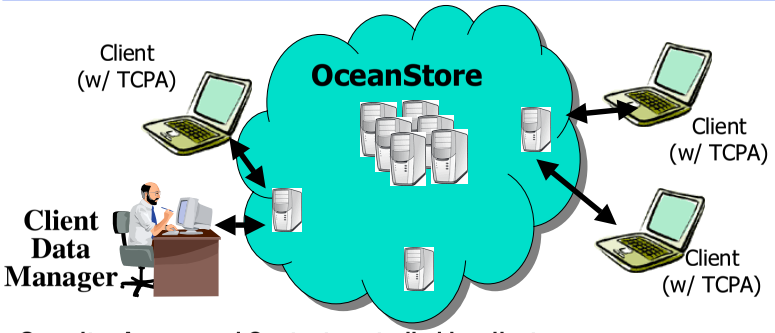
## The Path of an OceanStore Update



## Archival Dissemination of Fragments



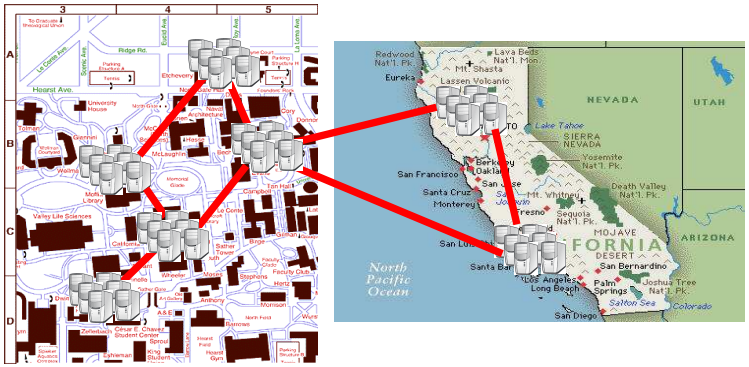

## Secure Object Storage




- Security: Access and Content controlled by client
  - Privacy through data encryption
  - Optional use of cryptographic hardware for revocation
  - Authenticity through hashing and active integrity checking
- Flexible self-management and optimization:
  - Performance and durability
  - Efficient sharing



## The Berkeley PetaByte Archival Service




- OceanStore Concepts Applied to Tape-less backup
  - Self-Replicating, Self-Repairing, Self-Managing
  - No need for actual Tape in system
    - (Although could be there to keep with tradition)



## I/O Summary:

- I/O performance limited by weakest link in chain between OS and device
- Three Components of Disk Access Time:
  - Seek Time: advertised to be 8 to 12 ms. May be lower in real life.
  - Rotational Latency: 4.1 ms at 7200 RPM and 8.3 ms at 3600 RPM
  - Transfer Time: 2 to 12 MB per second
- I/O device notifying the operating system:
  - Polling: it can waste a lot of processor time
  - I/O interrupt: similar to exception except it is asynchronous
- Delegating I/O responsibility from the CPU:
  - DMA, or even IOP
- Queueing theory is important
  - 100% utilization means very large latency
  - Remember, for M/M/1 queue (exponential source of requests/service)
    - queue size goes as $u/(1-u)$
    - latency goes as $T_{ser} \times u/(1-u)$
  - For M/G/1 queue (more general server, exponential sources)
    - latency goes as $m1(z) \times u/(1-u) = T_{ser} \times \{1/2 \times (1+C)\} \times u/(1-u)$
- Redundancy + Repair is key to high reliability