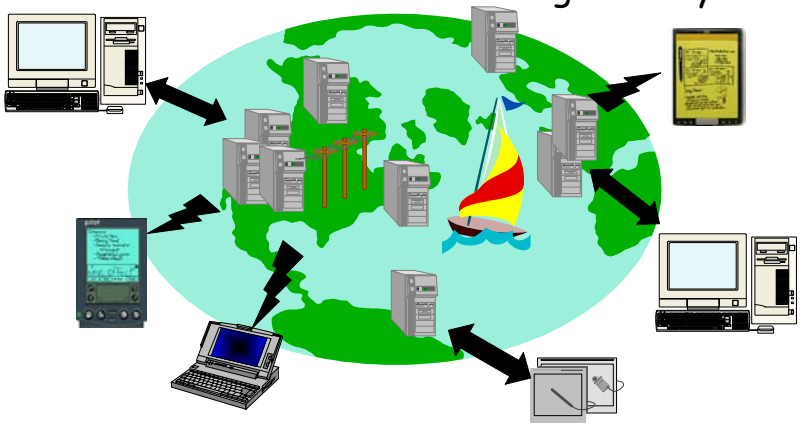
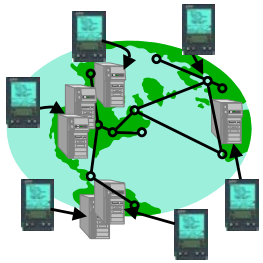
# OceanStore

Exploiting Peer-to-Peer for a Self-Repairing, Secure and Persistent Storage Utility



John Kubiatowicz
University of California at Berkeley

# OceanStore Context: Ubiquitous Computing

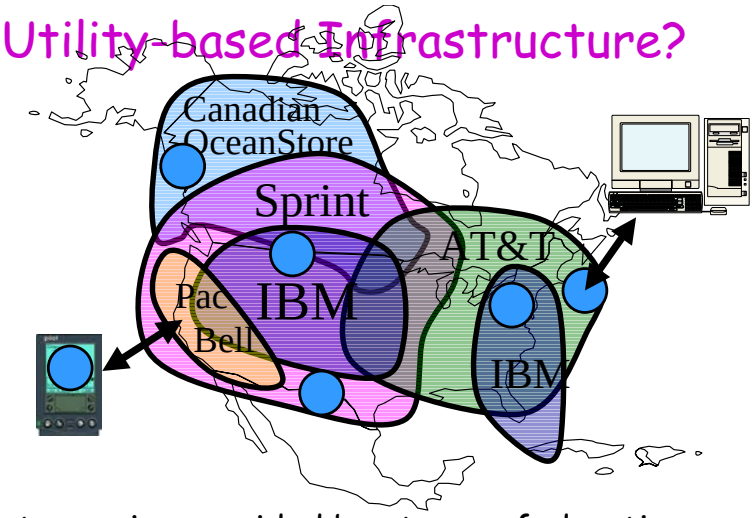


- Computing everywhere:
  - Desktop, Laptop, Palmtop
  - Cars, Cellphones
  - Shoes? Clothing? Walls?
- Connectivity everywhere:
  - Rapid growth of bandwidth in the interior of the net
  - Broadband to the home and office
  - Wireless technologies such as CMDA, Satelite, laser
- Where is persistent data????





# Utility-based Infrastructure?




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



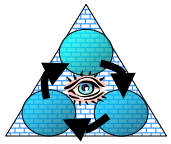


# What are the advantages of a utility?

- 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

# Key Observation: Want Automatic Maintenance

- Can't possibly manage billions of servers by hand!
- System should automatically:
  - Adapt to failure
  - Exclude malicious elements
  - Repair itself
  - Incorporate new elements



- System should be secure and private
  - Encryption, authentication
- System should preserve data over the long term (*accessible* for 1000 years):
  - Geographic distribution of information
  - New servers added from time to time
  - Old servers removed from time to time
  - Everything just works





# OceanStore: Everyone's Data, One Big Utility

"The data is just out there"

- How many files in the OceanStore?
  - Assume $10^{10}$ people in world
  - Say 10,000 files/person (very conservative?)
  - So $10^{14}$ files in OceanStore!
  - If 1 gig files (ok, a stretch), get 1 mole of bytes! (or a Yotta-Byte if you are a computer person)

Truly impressive number of elements…
… but small relative to physical constants

Aside: SIMS school: 1.5 Exabytes/year ($1.5 \times 10^{18}$)





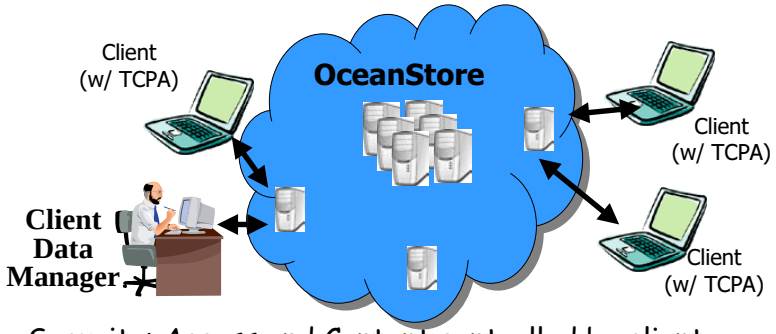
# Outline

- Motivation
- Applications We Have Now
  - What would it actually mean to have such a utility?
- Why Peer to Peer?
  - A new way of thinking?
- A Peek at OceanStore
  - Self-Verifying Data
  - Active Data Authentication
  - Archival Data Maintenance
- Prototype: It's Alive
- Conclusion





# Secure Object Storage

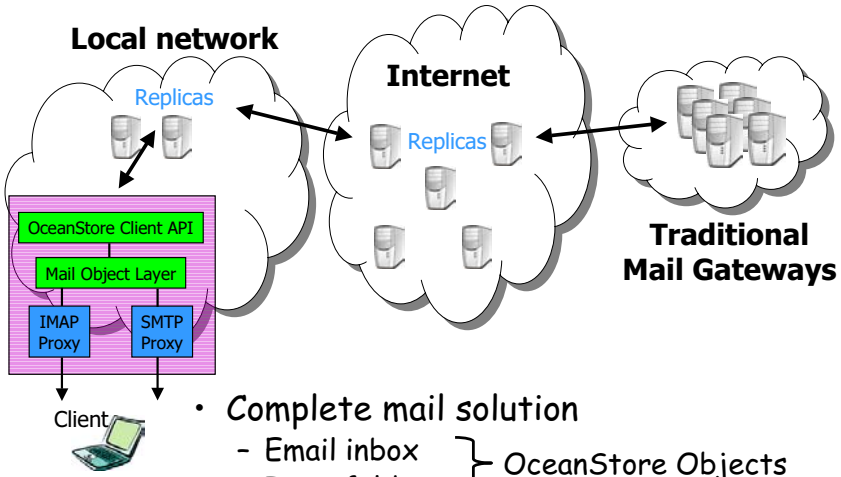



- Security: Access and Content controlled by client
  - Privacy through data encryption
  - Optional use of cryptographic hardware for revocation
  - Authenticity through hashing and active integrity checking
- Flexible self-management and optimization:
  - Performance and durability
  - Efficient sharing





# MINO: Wide-Area E-Mail Service



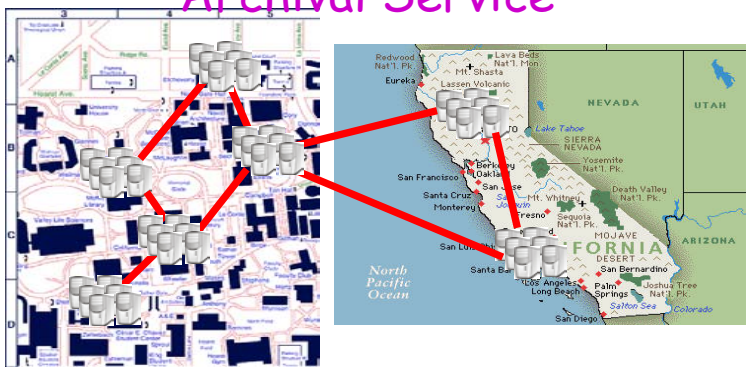

- Complete mail solution
  - Email inbox
  - Imap folders

} OceanStore Objects





# The Berkeley PetaByte Archival Service

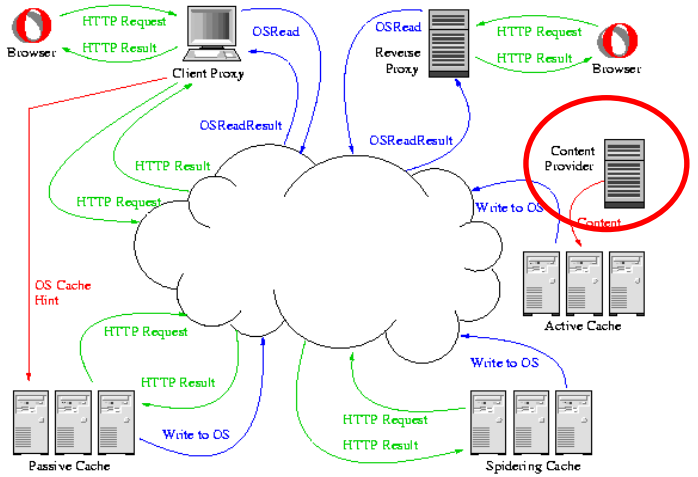



- OceanStore Concepts Applied to Tape-less backup
  - Self-Replicating, Self-Repairing, Self-Managing
  - No need for actual Tape in system
    - (Although could be there to keep with tradition)





# High-Performance, Authenticated Access to legacy Archives via HTTP



# Why Peer-to-Peer?



# Peer-to-Peer is:


- Old View:
  - A bunch of flakey high-school students stealing music
- New View:
  - A philosophy of systems design at extreme scale
  - Probabilistic design when it is appropriate
  - New techniques aimed at unreliable components
  - A rethinking (and recasting) of distributed algorithms
  - Use of Physical, Biological, and Game-Theoretic techniques to achieve guarantees



# OceanStore Assumptions

**Peer-to-peer**

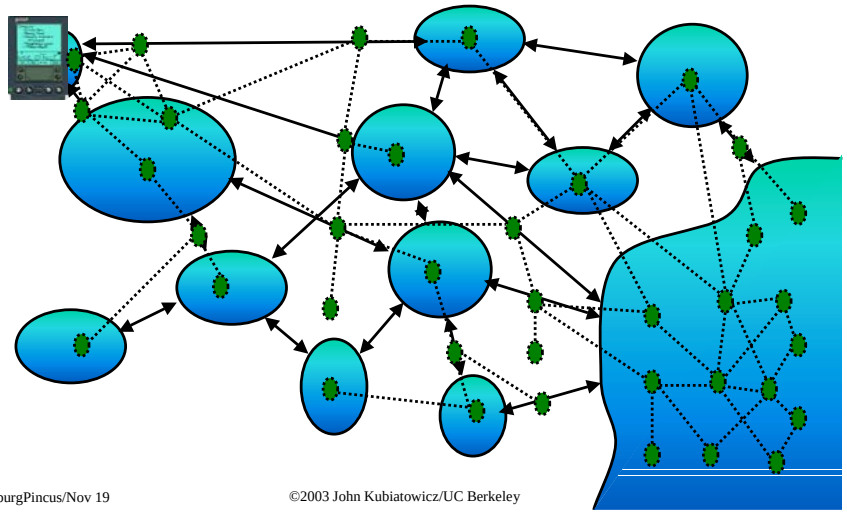
- Untrusted Infrastructure:
  - The OceanStore is comprised of untrusted components
  - Individual hardware has finite lifetimes
  - All data encrypted within the infrastructure
- Mostly Well-Connected:
  - Data producers and consumers are connected to a high-bandwidth network most of the time
  - Exploit multicast for quicker consistency when possible
- Promiscuous Caching:
  - Data may be cached anywhere, anytime

**Quality-of-Service**

- Responsible Party:
  - Some organization (*i.e. service provider)* guarantees that your data is consistent and durable
  - Not trusted with *content* of data, merely its *integrity*



# The Peer-To-Peer View: Irregular Mesh of "Pools"



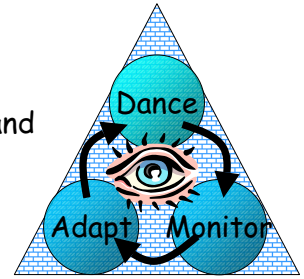
# The Thermodynamic Analogy

- Large Systems have a variety of *latent order*
  - Connections between elements
  - Mathematical structure (erasure coding, etc)
  - Distributions peaked about some desired behavior
- Permits "Stability through Statistics"
  - Exploit the behavior of aggregates (redundancy)
- Subject to Entropy
  - Servers fail, attacks happen, system changes
- Requires continuous repair
  - Apply energy (i.e. through servers) to reduce entropy



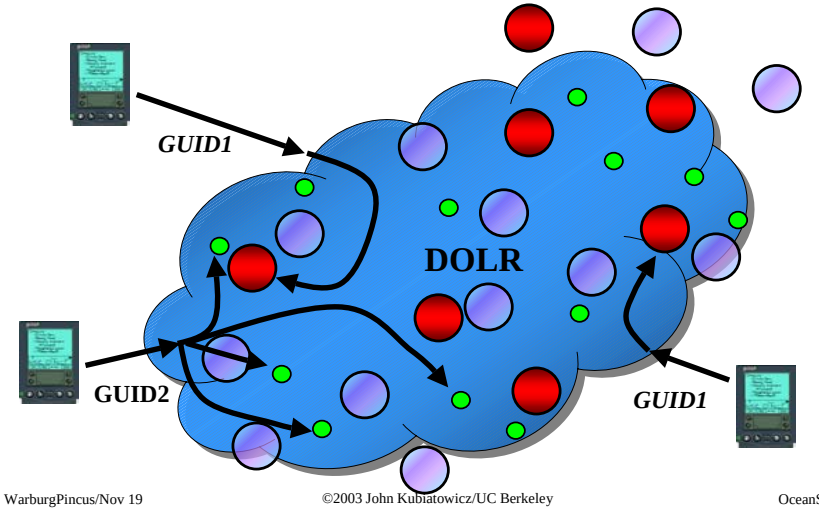
# The Biological Inspiration

- Biological Systems are built from (extremely) faulty components, yet:
  - They operate with a variety of component failures ⇒ Redundancy of function and representation
  - They have stable behavior ⇒ Negative feedback
  - They are self-tuning ⇒ Optimization of common case
- Introspective (Autonomic) Computing:
  - Components for performing
  - Components for monitoring and model building
  - Components for continuous adaptation



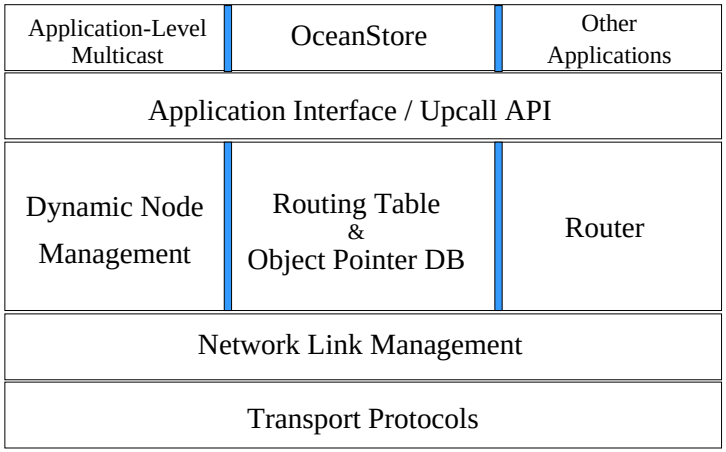



# Important Example: DOLR
## (Decentralized Object Location and Routing)

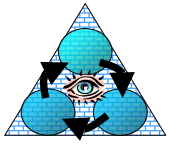





# Single Node *Tapestry* DOLR: Like a router



# It's Alive!

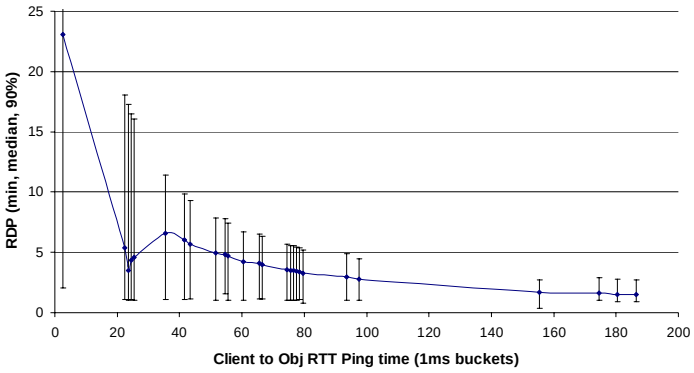


- Planet Lab testbed
  - 104 machines at 43 institutions, in North America, Europe, Australia (~ 60 machines utilized)
  - 1.26Ghz PIII (1GB RAM), 1.8Ghz PIV (2GB RAM)
  - North American machines (2/3) on Internet2
- Tapestry Java deployment
  - 50,000 lines of Java code
  - 6-7 nodes on each physical machine
  - IBM Java JDK 1.30
  - Node virtualization inside JVM and SEDA
  - Scheduling between virtual nodes increases latency





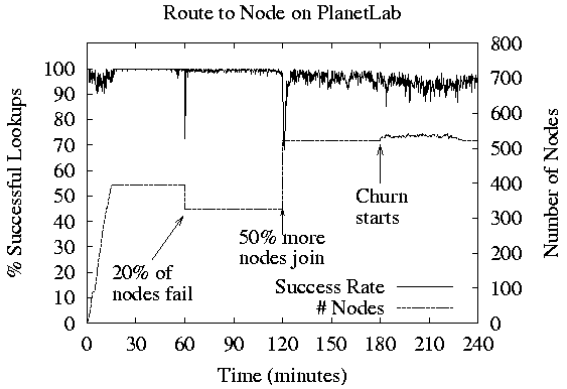
# Object Location with Tapestry








# Stability under extreme circumstances




(May 2003: 1.5 TB over 4 hours)

DOLR Model generalizes to many simultaneous apps

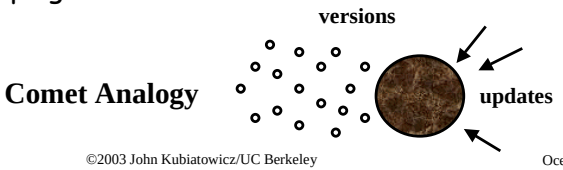




# Peek at OceanStore

# OceanStore Data Model

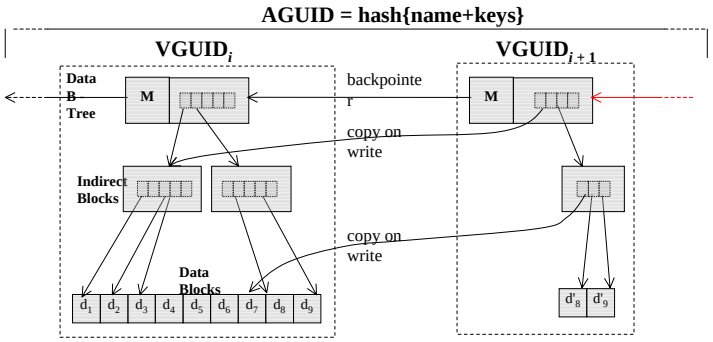
- Versioned Objects
  - Every update generates a new version
  - Can always go back in time (Time Travel)
- Each Version is Read-Only
  - Can have permanent name
  - Much easier to repair
- An Object is a signed mapping between permanent name and latest version
  - Write access control/integrity involves managing these mappings



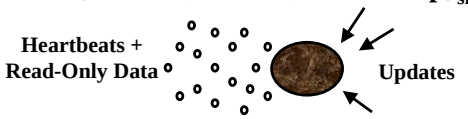





# Self-Verifying Objects



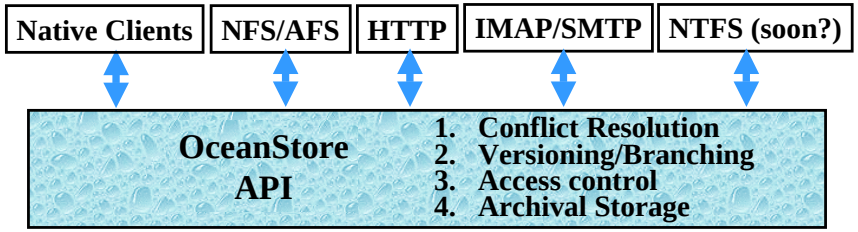

♥Heartbeat: {AGUID,VGUID, Timestamp}$_{signed}$








# OceanStore API: Universal Conflict Resolution



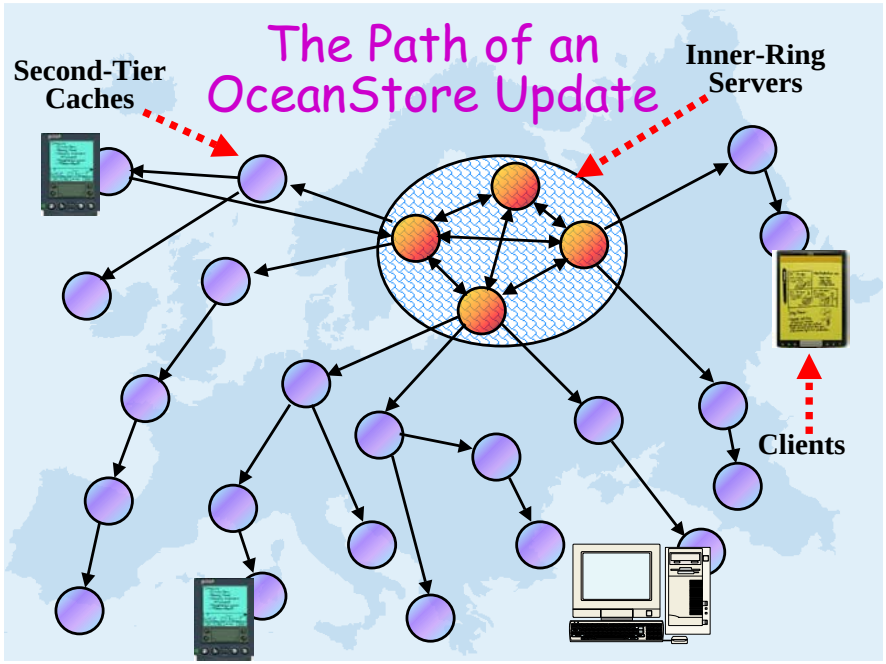

- Consistency is form of optimistic concurrency
  - Updates contain *predicate-action* pairs
  - Each predicate tried in turn:
    - If none match, the update is *aborted*
    - Otherwise, action of first true predicate is *applied*
- Role of Responsible Party (RP):
  - Updates submitted to RP which chooses total order





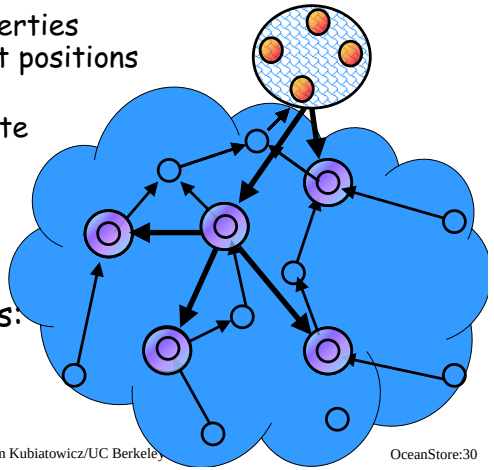
# Two Types of OceanStore Data

- *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)
    - Other parameter, *rate*, is 1/overhead
  - Fragments are cryptographically self-verifying
- Most data in the OceanStore is archival!

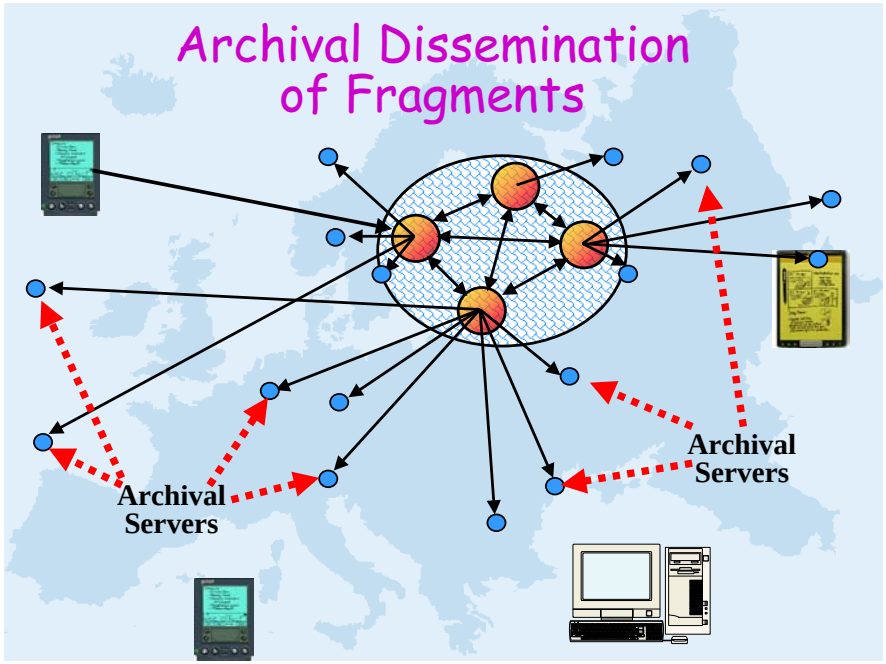



# Self-Organizing Soft-State Replication

- Simple algorithms for placing replicas on nodes in the interior
  - Intuition: locality properties of Tapestry help select positions for replicas
  - Tapestry helps associate parents and children to build multicast tree
- Preliminary results encouraging
- Current Investigations:
  - Game Theory
  - Thermodynamics



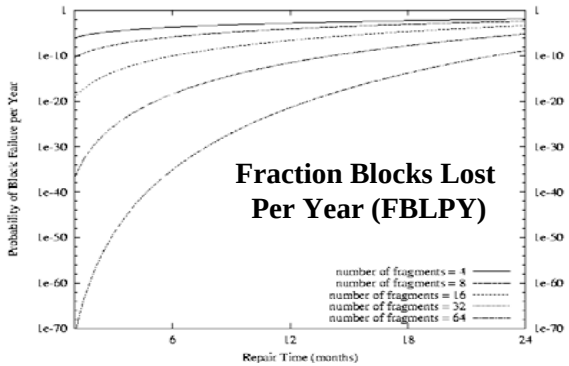





# Differing Degrees of Responsibility

- Inner-ring provides quality of service
  - Handles of live data and write access control
  - Focus utility resources on this vital service
  - Compromised servers must be detected quickly
- Caching service can be provided by anyone
  - Data encrypted and self-verifying
  - Pay for service "Caching Kiosks"?
- Archival Storage and Repair
  - Read-only data: easier to authenticate and repair
  - Tradeoff redundancy for responsiveness
- Could be provided by different companies!

# Aside: Why erasure coding? High Durability/overhead ratio!




- Exploit law of large numbers for durability!
- 6 month repair, FBLPY:
  - Replication: 0.03
  - Fragmentation: $10^{-35}$

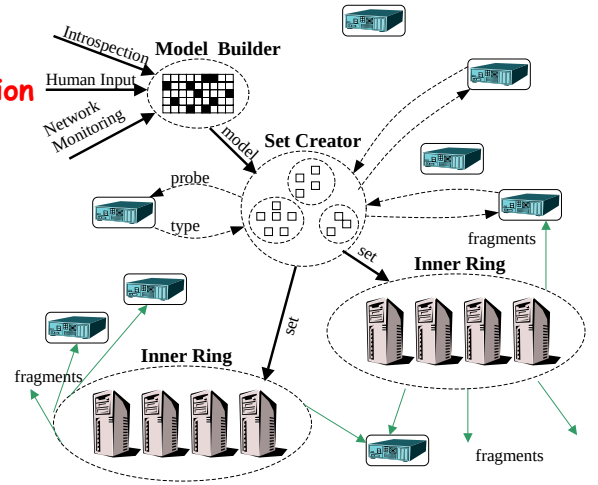


# The Dissemination Process: Achieving Failure Independence

**Anti-correlation Analysis/Models**
- **Mutual Information**
- **Human Input**
- **Data Mining**

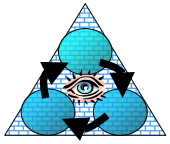
**Independent Set Generation**

**Effective Dissemination**



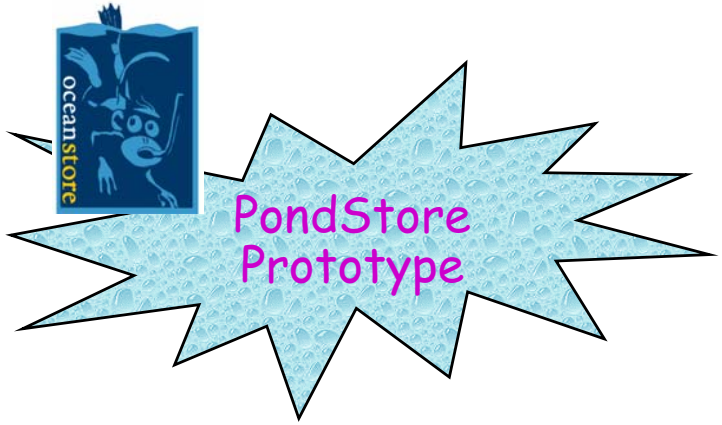



# Extreme Durability?



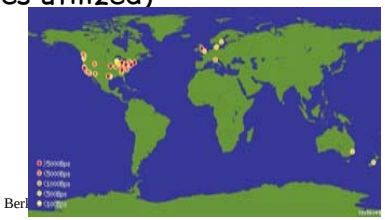
- Exploiting Infrastructure for Repair
  - DOLR permits efficient heartbeat mechanism to notice:
    - Servers going away for a while
    - Or, going away forever!
  - Continuous sweep through data also possible
  - Erasure Code provides Flexibility in Timing
- Data transferred from physical medium to physical medium
  - No "tapes decaying in basement"
  - Information becomes fully Virtualized
- Thermodynamic Analogy: Use of Energy (supplied by servers) to Suppress Entropy





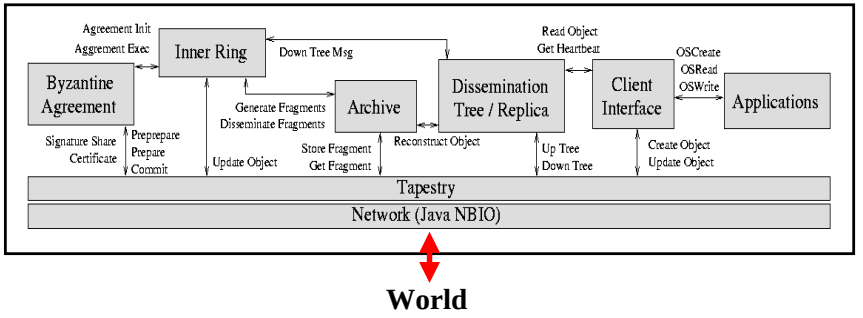
# OceanStore Prototype

- All major subsystems operational
  - Self-organizing Tapestry base
  - Primary replicas use Byzantine agreement
  - Secondary replicas self-organize into multicast tree
  - Erasure-coding archive
  - Application interfaces: NFS, IMAP/SMTP, HTTP
- 280K lines of Java (J2SE v1.3)
  - JNI libraries for cryptography, erasure coding
- PlanetLab Deployment (FAST 2003, "Pond" paper)
  - 104 machines at 43 institutions, in North America, Europe, Australia (~ 60 machines utilized)
  - 1.26Ghz PIII (1GB RAM), 1.8Ghz PIV (2GB RAM)
  - OceanStore code running with 1000 virtual-node emulations





# Event-Driven Architecture of an OceanStore Node




- Data-flow style
  - Arrows Indicate flow of messages
- Potential to exploit small multiprocessors at each physical node



# Closer Look: Write Cost

- Small writes
  - Signature dominates
  - Threshold sigs. slow!
  - Takes 70+ ms to sign
  - Compare to 5 ms for regular sigs.
- Large writes
  - Encoding dominates
  - Archive cost per byte
  - Signature cost per write
- The Future
  - Tentative Writes remove Inner-Ring from latency path

**PondStore Prototype**

| Phase | 4 kB write | 2 MB write |
|---|---|---|
| Validate | 0.3 | 0.4 |
| Serialize | 6.1 | 26.6 |
| Apply | 1.5 | 113.0 |
| Archive | 4.5 | 566.9 |
| Sign Result | 77.8 | 75.8 |

**Times in ms**



# Conclusions

- Peer-to-Peer System are a Hot Area
  - Large amounts of redundancy and connectivity
  - New Organizations for Large-Scale Systems
  - Thermodynamics of systems
  - Continuous Introspection and Repair
- Help the Infrastructure to Help you
  - Decentralized Object Location and Routing (DOLR)
  - Object-based Storage
  - Self-Organizing redundancy
  - Continuous Repair
- OceanStore properties:
  - Provides security, privacy, and integrity
  - Provides extreme durability
  - Lower maintenance cost through redundancy, continuous adaptation, self-diagnosis and repair

# For more info:
## http://oceanstore.org

- OceanStore vision paper for ASPLOS 2000
  - "OceanStore: An Architecture for Global-Scale Persistent Storage"
- Pond Implementation paper for FAST 2003
  - "Pond: the OceanStore Prototype"
- Tapestry algorithms paper (SPAA 2002):
  - "Distributed Object Location in a Dynamic Network"
- Tapestry deployment paper (JSAC, to appear)
  - "Tapestry: A Resilient Global-scale Overlay for Service Deployment"
- Bloom Filters for Probabilistic Routing (INFOCOM 2002):
  - "Probabilistic Location and Routing"
- CACM paper (February 2003):
  - "Extracting Guarantees from Chaos"