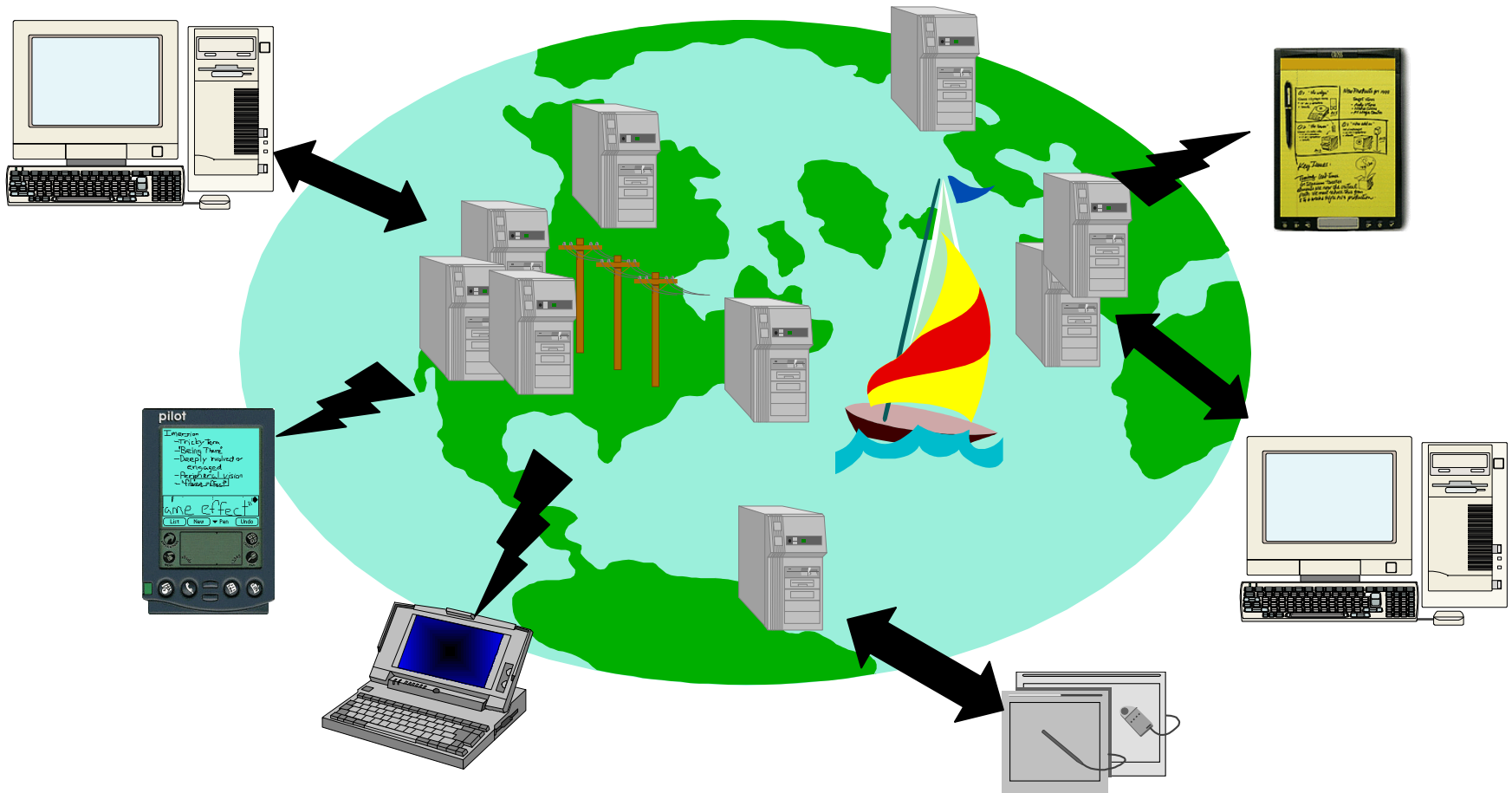


OceanStore

Global-Scale Persistent Storage



John Kubiawicz
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

Questions about information:

- Where is persistent information stored?
 - *Want: Geographic independence for availability, durability, and freedom to adapt to circumstances*
- How is it protected?
 - *Want: Encryption for privacy, signatures for authenticity, and Byzantine commitment for integrity*
- Can we make it indestructible?
 - *Want: Redundancy with continuous repair and redistribution for long-term durability*
- Is it hard to manage?
 - *Want: automatic optimization, diagnosis and repair*
- Who owns the aggregate resources?
 - *Want: Utility Infrastructure!*

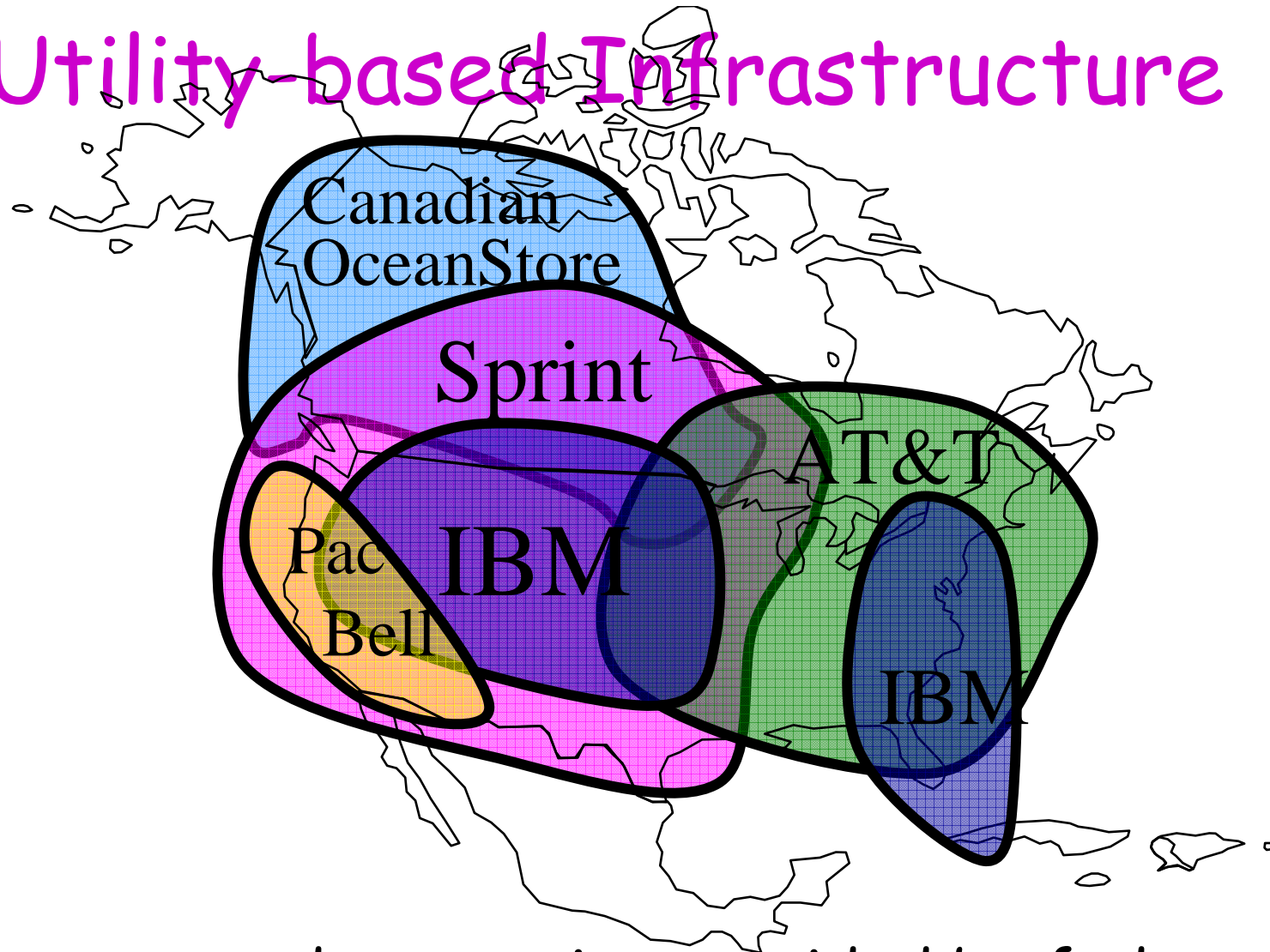
First Observation: Want Utility Infrastructure

- Mark Weiser from Xerox:
Transparent computing is the ultimate goal
 - Computers should disappear into the background
- In storage context:
 - Don't want to worry about backup
 - Don't want to worry about obsolescence
 - Need lots of resources to make data secure and highly available, BUT *don't want to own them*
 - Outsourcing of storage already becoming popular
- Pay monthly fee and your "data is out there"
 - Simple payment interface
 - ⇒ one bill from one company

Second Observation: Want Automatic Maintenance

- Can't possibly manage billions of servers by hand!
- System should automatically:
 - Adapt to failure
 - Repair itself
 - Incorporate new elements
- Can we guarantee data is *available* for 1000 years?
 - New servers added from time to time
 - Old servers removed from time to time
 - Everything just works
- *Many* components with *geographic* separation
 - System not disabled by natural disasters
 - Can adapt to changes in demand and regional outages
 - *Gain in stability through statistics*

Utility-based Infrastructure



- Transparent data service provided by federation of companies:
 - Monthly fee paid to one service provider
 - Companies buy and sell capacity from each other

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!

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

Aside: new results: 1.5 Exabytes/year (1.5×10^{18})

Outline

- Motivation
- Assumptions of the OceanStore
- Specific Technologies and approaches:
 - Naming
 - Routing and Data Location
 - Conflict resolution on encrypted data
 - Replication and Deep archival storage
 - Oceanic Data Market
 - Introspection for optimization and repair
- Conclusion

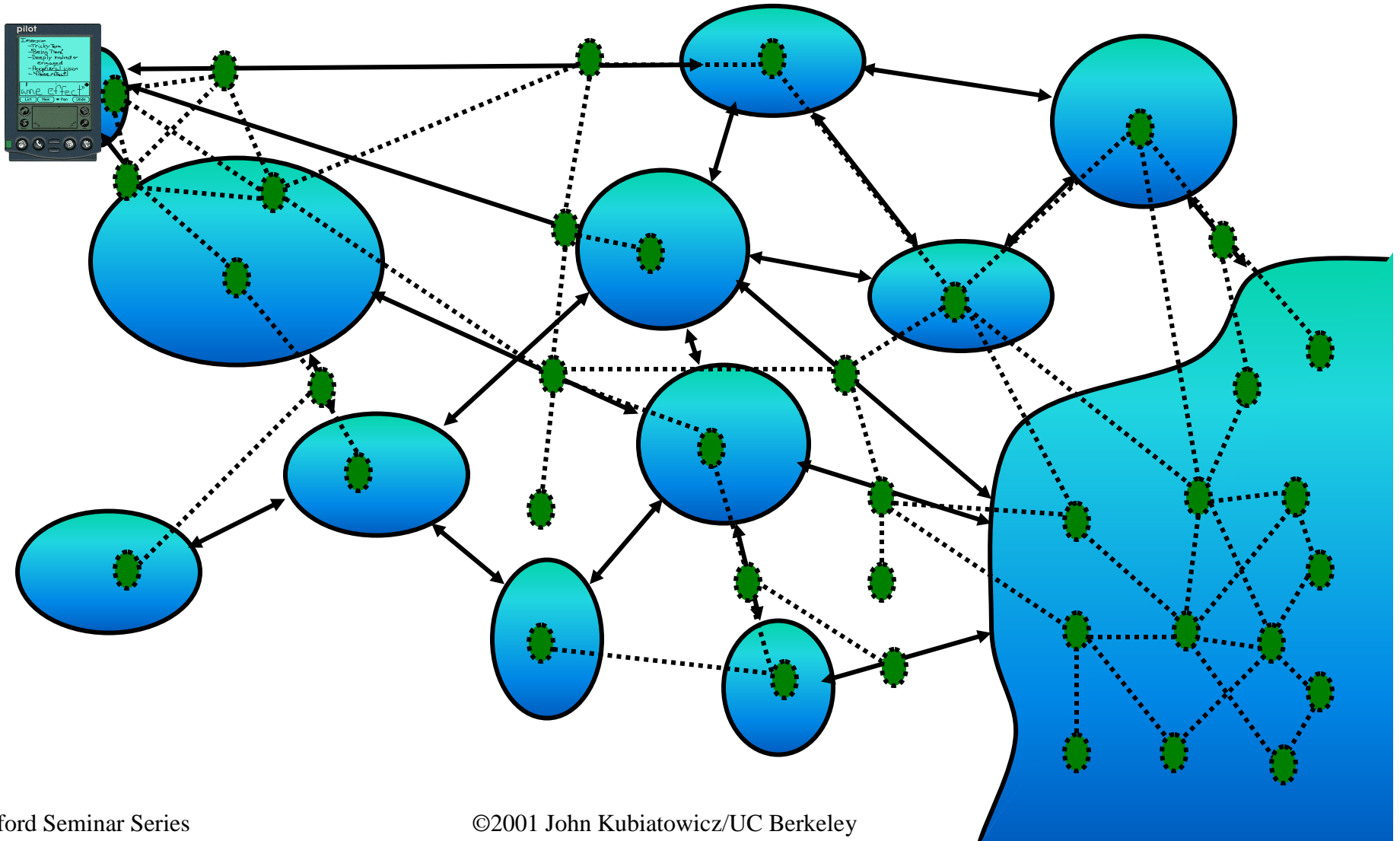
OceanStore Assumptions

- **Untrusted Infrastructure:**
 - The OceanStore is comprised of untrusted components
 - Only ciphertext within the infrastructure
- **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*
- **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
- **Optimistic Concurrency via Conflict Resolution:**
 - Avoid locking in the wide area
 - Applications use object-based interface for updates

Use of Moore's law gains

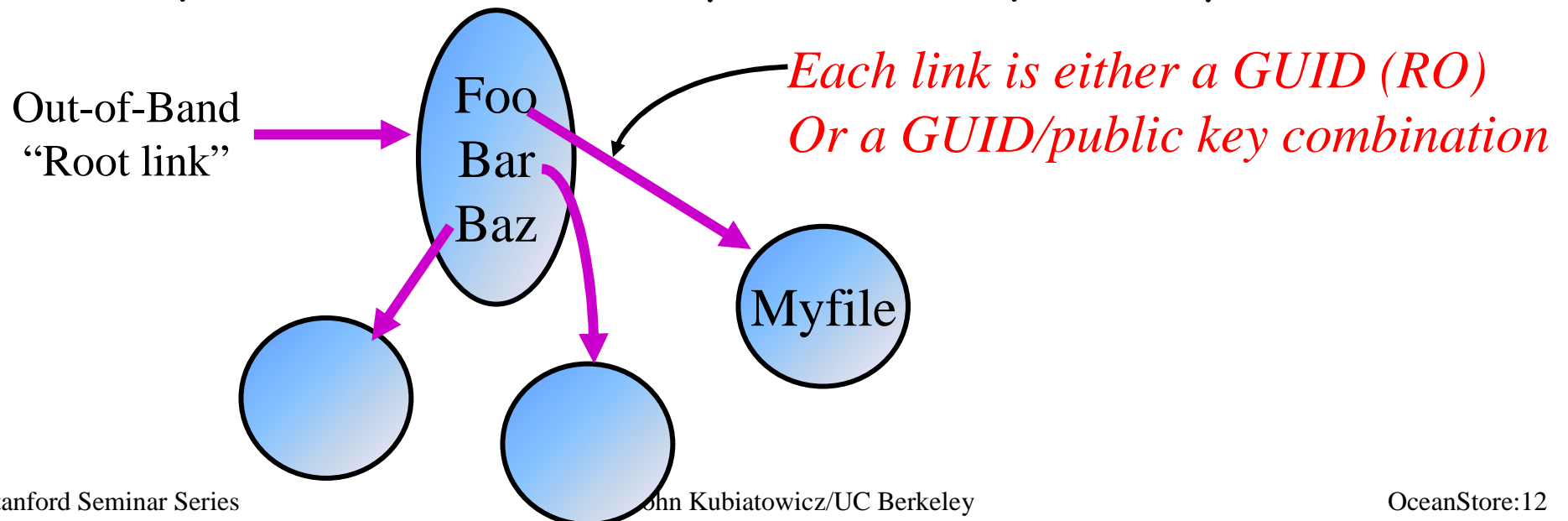
- Question: Can we use Moore's law gains for something other than just raw performance?
 - Growth in computational *performance*
 - Growth in network *bandwidth*
 - Growth in *storage* capacity
- Examples:
 - Stability through Statistics
 - Use of redundancy of servers, network packets, *etc.* in order to gain more predictable behavior
 - *Systems version of Thermodynamics!*
 - Extreme Durability (1000-year time scale?)
 - Use of erasure coding and continuous repair
 - Security and Authentication
 - Signatures and secure hashes in many places
 - Continuous dynamic optimization

Basic Structure: Irregular Mesh of "Pools"



Secure Naming

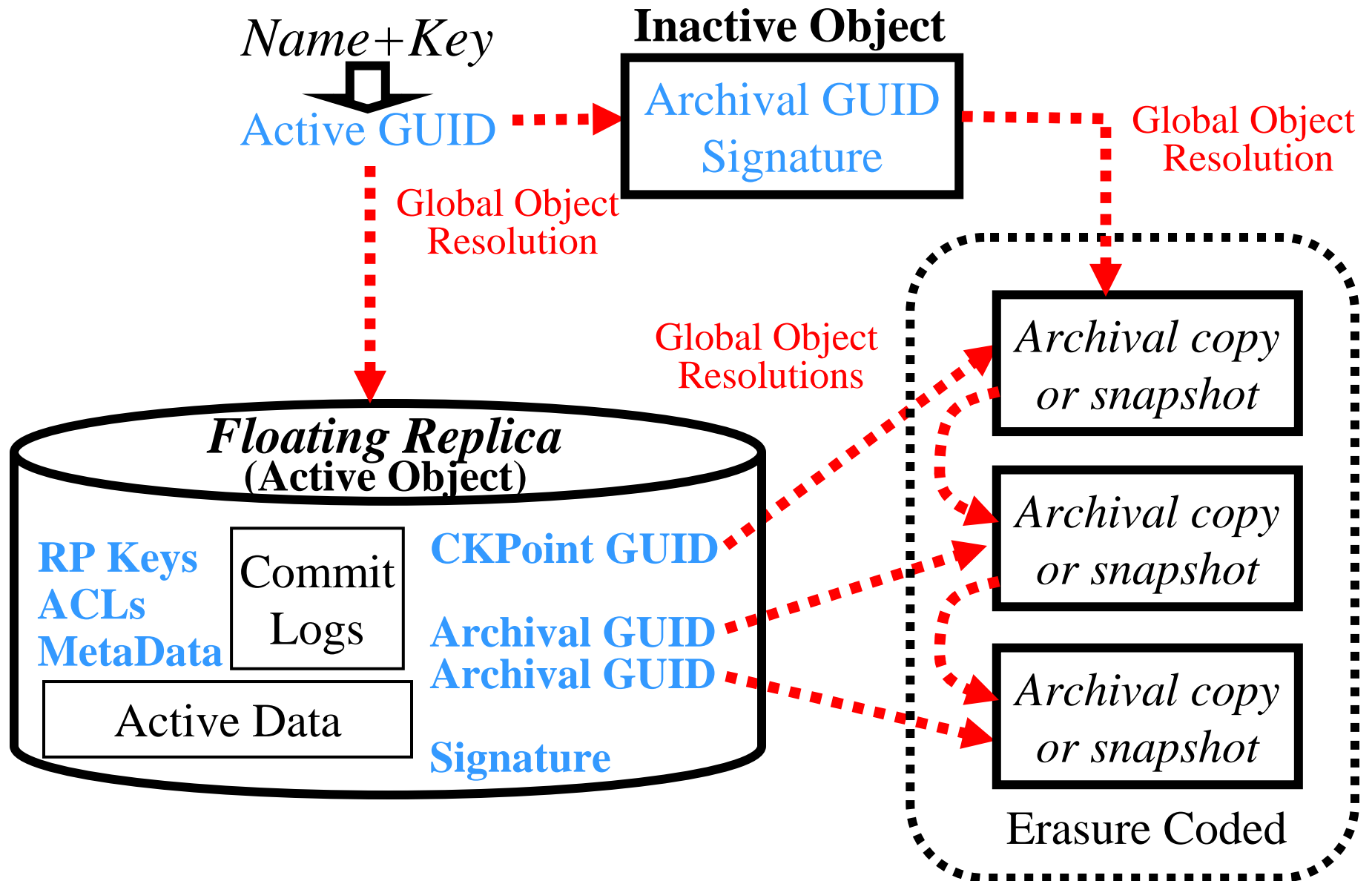
- Unique, location independent identifiers:
 - Every *version* of every unique entity has a permanent, *Globally Unique ID (GUID)*
 - All OceanStore operations operate on GUIDs
- Naming hierarchy:
 - Users map from names to GUIDs via hierarchy of OceanStore objects (*ala SDSI*)
 - Requires set of "root keys" to be acquired by user



Unique Identifiers

- Secure Hashing is key!
 - Use of 160-bit SHA-1 hashes over information provides uniqueness, unforgeability, and verifiability:
 - *Read-only data*: GUID is hash over actual information
 - Uniqueness and Unforgeability: the data is what it is!
 - Verification: check hash over data
 - *Changeable data*: GUID is combined hash over a human-readable name + public key
 - Uniqueness: GUID space selected by public key
 - Unforgeability: public key is indelibly bound to GUID
 - Verification: check signatures with public key
- Is 160 bits enough?
 - Birthday paradox requires over 2^{80} unique objects before collisions worrisome
 - Good enough for now

GUIDs \Rightarrow Secure Pointers



Routing and Data Location

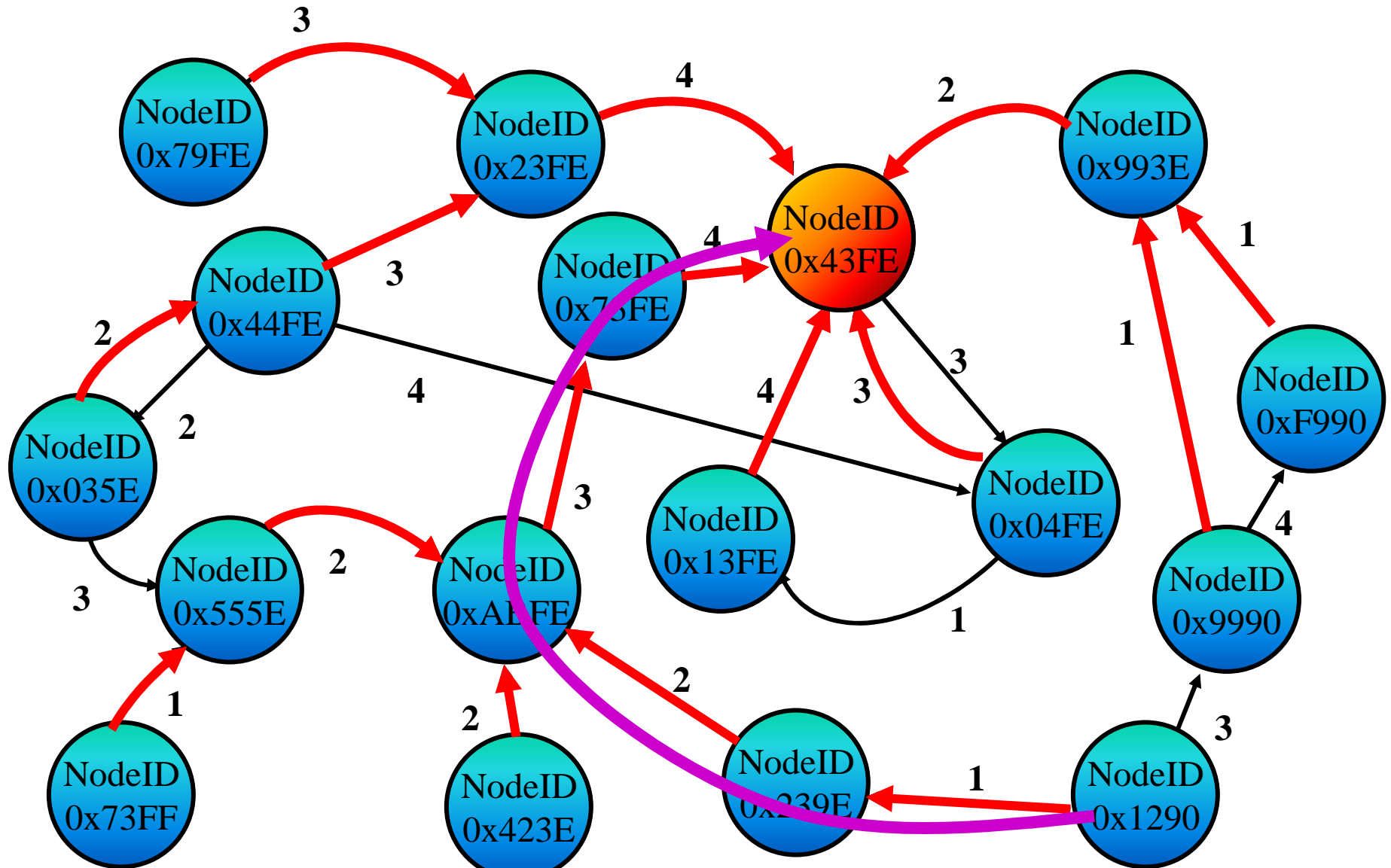
- Requirements:
 - Find data quickly, wherever it might reside
 - Locate nearby data without global communication
 - Permit rapid data migration
 - Insensitive to faults and denial of service attacks
 - Provide multiple routes to each piece of data
 - Route around bad servers and ignore bad data
 - Repairable infrastructure
 - Easy to reconstruct routing and location information
- Technique: Combined Routing and Data Location
 - Packets are addressed to GUIDs, not locations
 - Infrastructure gets the packets to their destinations and verifies that servers are behaving

Two-levels of Routing

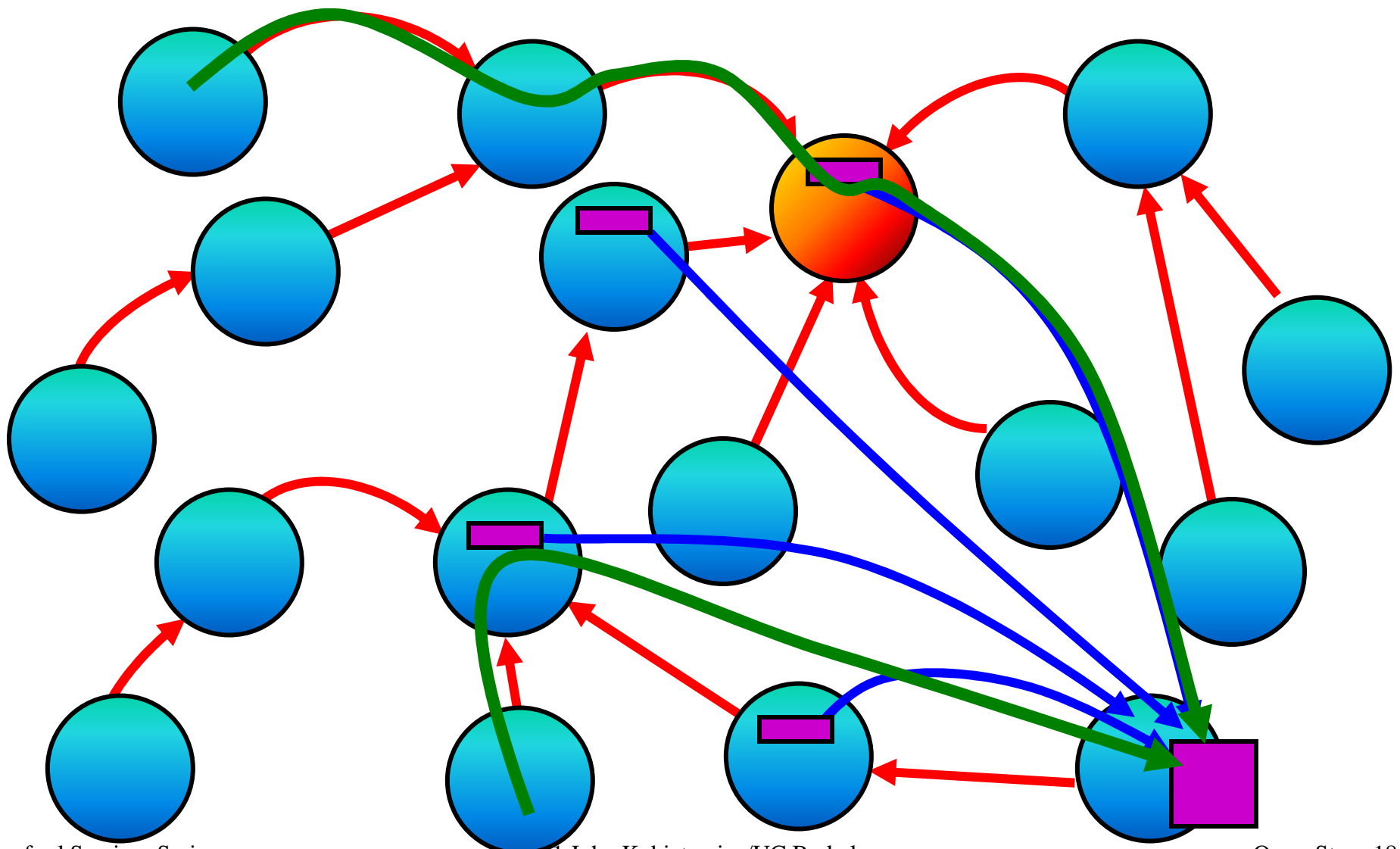
- Fast, probabilistic search for “routing cache”:
 - Built from *attenuated* bloom filters
 - Approximation to gradient search
 - *Not going to say more about this today*
- Redundant *Plaxton Mesh* used for underlying routing infrastructure:
 - Randomized data structure with locality properties
 - Redundant, insensitive to faults, and repairable
 - Amenable to continuous adaptation to adjust for:
 - Changing network behavior
 - Faulty servers
 - Denial of service attacks

Basic Plaxton Mesh

Incremental suffix-based routing



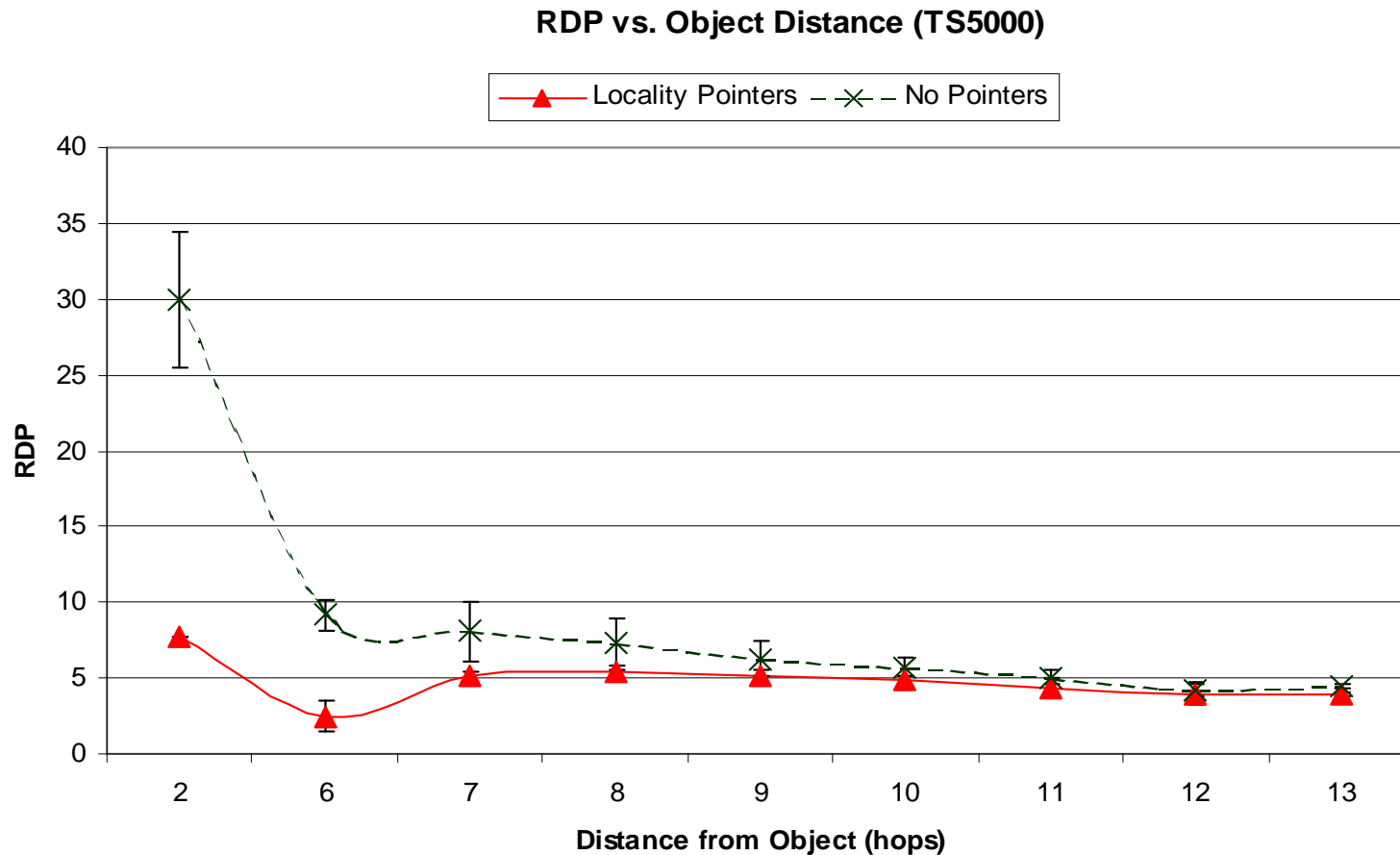
Use of Plaxton Mesh Randomization and Locality



Use of the Plaxton Mesh: The Tapestry Infrastructure

- As in original Plaxton scheme:
 - Scheme to directly map GUIDs to root node IDs
 - Replicas publish toward a document root
 - Search walks toward root until pointer located \Rightarrow *locality!*
- OceanStore enhancements for reliability:
 - Documents have multiple roots (Salted hash of GUID)
 - Each node has multiple neighbor links
 - Searches proceed along multiple paths
 - Tradeoff between reliability and bandwidth?
 - Routing-level validation of query results

How important are the pointers?



- Transit-Stub IP network of 5000 nodes
- 100 Tapestry overlay nodes

Automatic Maintenance

- All Tapestry state is Soft State
 - State maintained during transformations of network
 - Periodic restoration of state
- Self-Tuning of link structure
- Dynamic insertion:
 - New nodes contact small number of existing nodes
 - Integrate themselves automatically
 - Later, introspective optimization will move data to new servers
- Dynamic deletion:
 - Node detected as unresponsive
 - Pointer state routed around faulty node (signed deletion requests authorized by servers holding data)

OceanStore Consistency via Conflict Resolution

- Consistency is form of optimistic concurrency
 - An update packet contains a series of *predicate-action* pairs which operate on encrypted data
 - Each predicate tried in turn:
 - If none match, the update is *aborted*
 - Otherwise, action of first true predicate is *applied*
- Role of Responsible Party
 - All updates submitted to Responsible Party which chooses a final total order
 - Byzantine agreement with threshold signatures
- This is powerful enough to synthesize:
 - ACID database semantics
 - release consistency (build and use MCS-style locks)
 - Extremely loose (weak) consistency

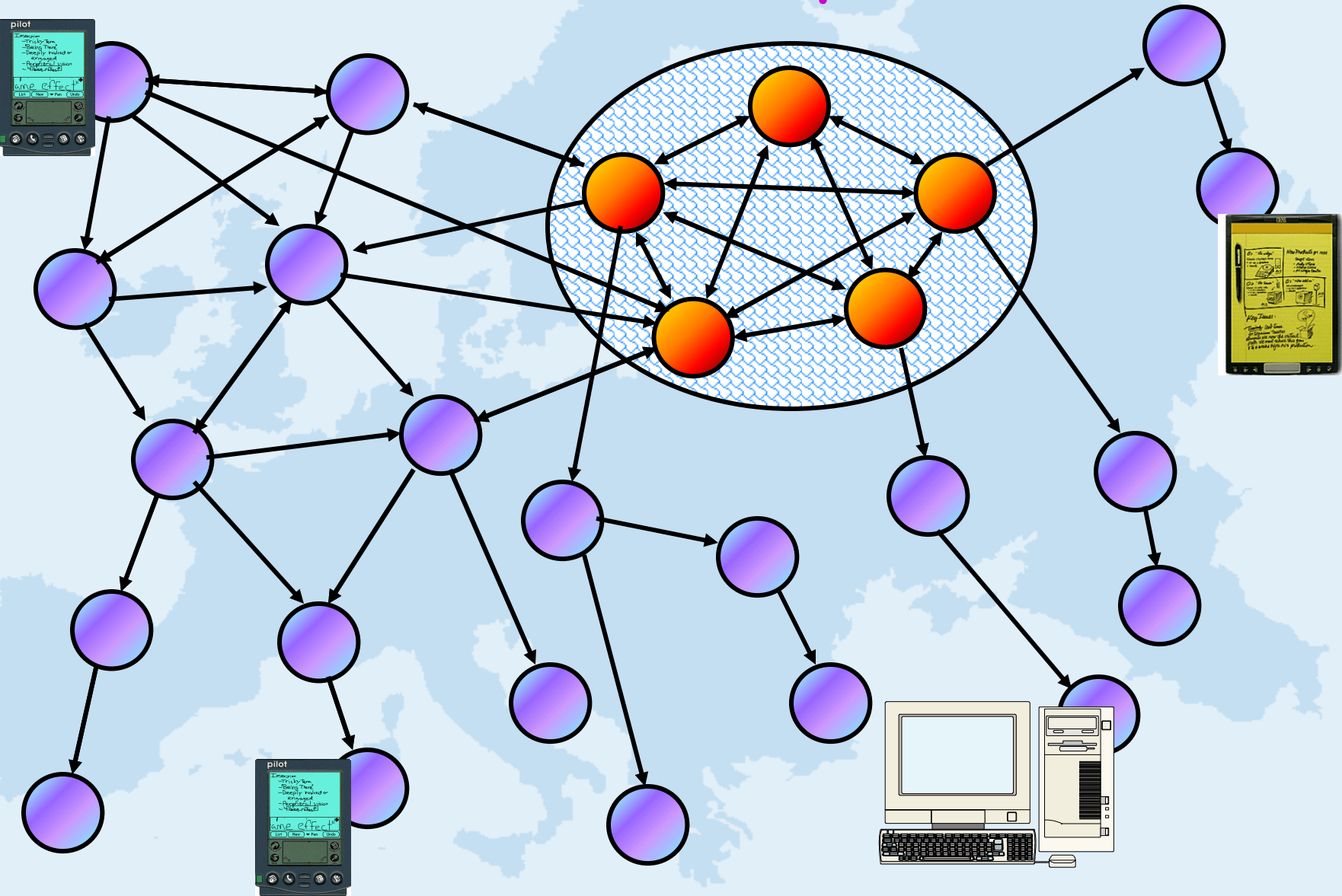
Oblivious Updates on Encrypted Data?

- Tentative Scheme:
 - Divide data into small blocks
 - Updates on a per-block basis
 - Predicates derived from techniques for searching on encrypted data
- Still exploring other options

TimeStamp
Client ID
{Pred1, Update1}
{Pred2, Update2}
{Pred3, Update3}
Client Signature

Unique Update ID is
hash over packet

The Path of an OceanStore Update



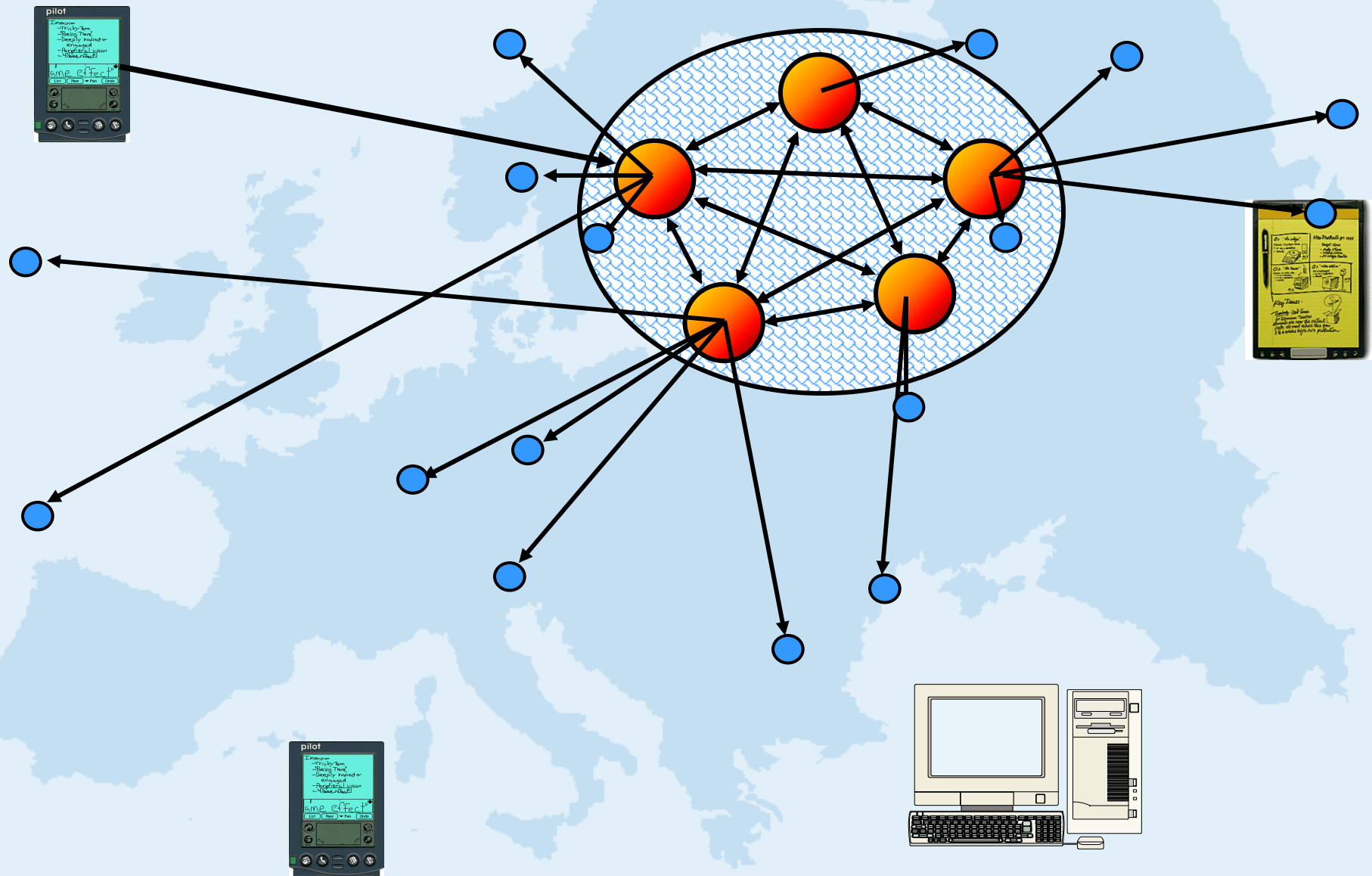
Automatic Maintenance

- Byzantine Commitment for inner ring:
 - Can tolerate up to 1/3 faulty servers in inner ring
 - Bad servers can be arbitrarily bad
 - Cost $\sim n^2$ communication
 - Continuous refresh of set of inner-ring servers
 - Proactive threshold signatures
 - Use of Tapestry \Rightarrow membership of inner ring unknown to clients
- Secondary tier self-organized into overlay dissemination tree
 - Use of Tapestry routing to suggest placement of replicas in the infrastructure
 - Automatic choice between *update* vs *invalidate*

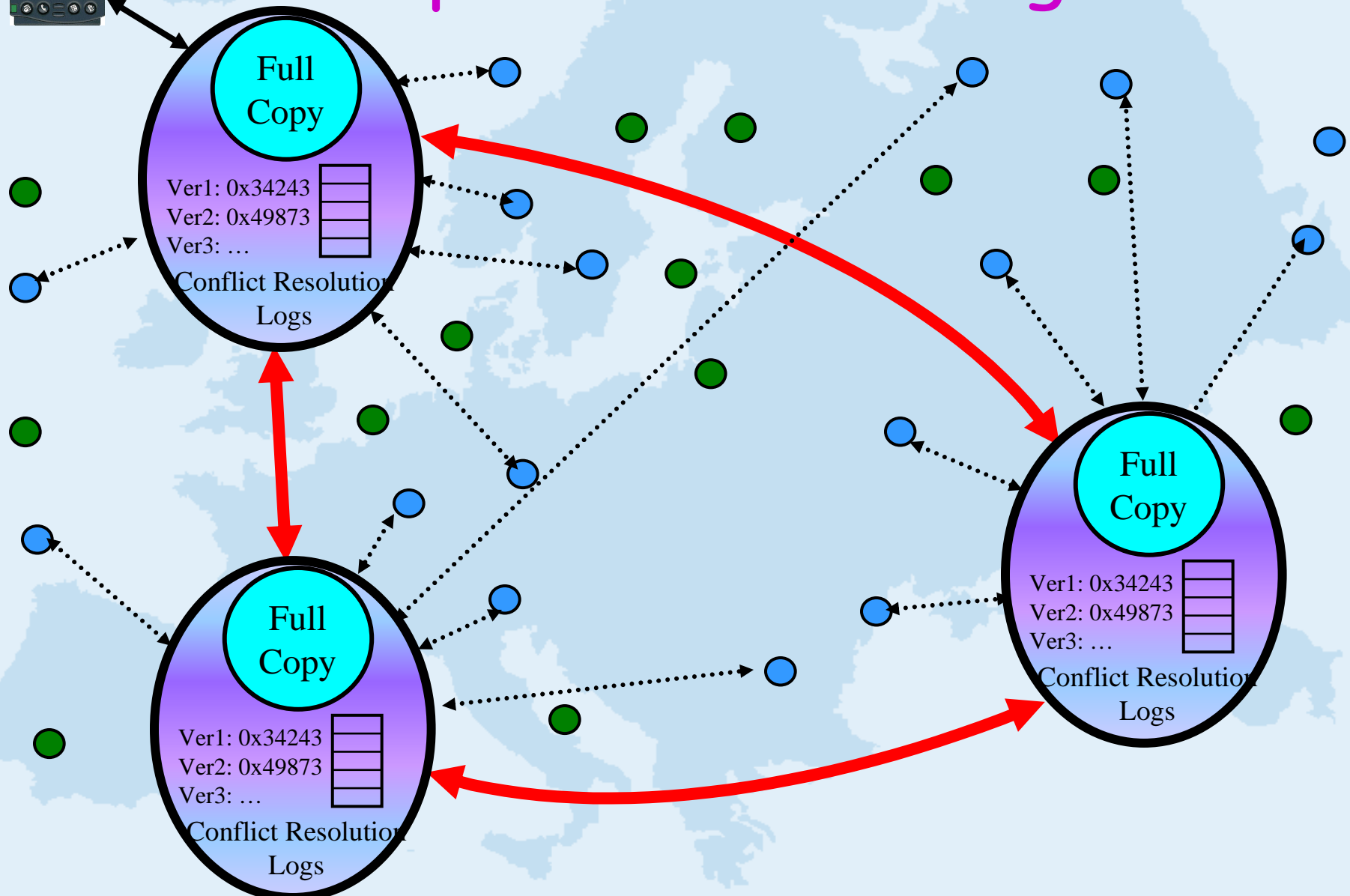
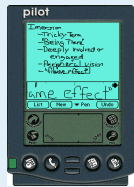
Data Coding Model

- Two distinct forms of data: active and archival
- *Active Data* in Floating Replicas
 - Per object virtual server
 - Logging for updates/conflict resolution
 - Interaction with other replicas to keep data consistent
 - *May appear and disappear like bubbles*
- *Archival Data* in Erasure-Coded Fragments
 - 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 $n \div m$ (e.g 4)
 - *Law of large numbers advantage to fragmentation*
 - Fragments are self-verifying
 - *OceanStore equivalent of stable store*
- Most data in the OceanStore is archival!

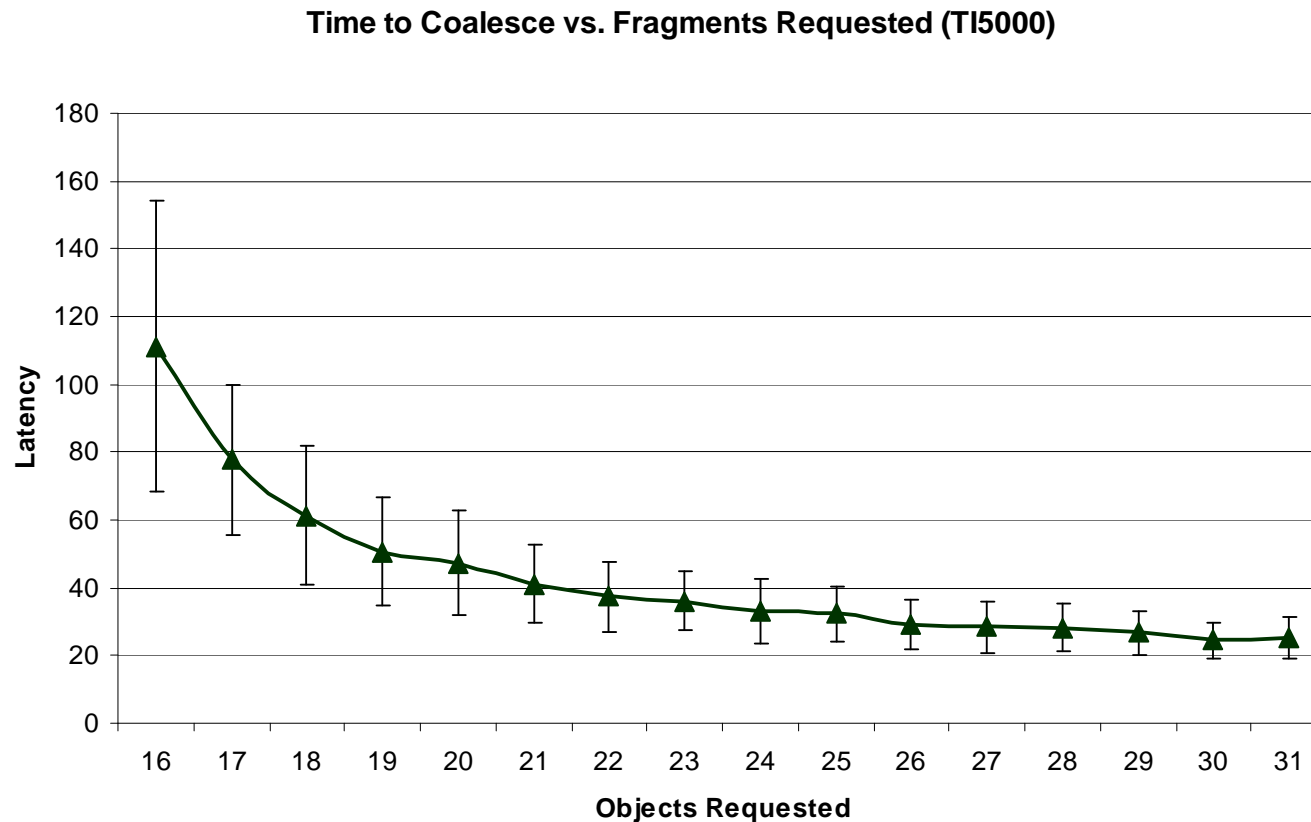
Archival Dissemination of Fragments



Floating Replica and Deep Archival Coding



Statistical Advantage of Fragments



- Latency *and* standard deviation reduced:
 - Memory-less latency model
 - Rate $\frac{1}{2}$ code with 32 total fragments

Automatic Maintenance

- Tapestry knows every location of fragment
 - Reverse pointer traversing permits multicast heartbeat from server of data to:
 - Everyone who has a pointer to data on that server
 - Or: Everyone who is one or two hops from server
 - Permits infrastructure to notice:
 - Servers going away for a while
 - Or, going away forever!
- Continuous sweep through data
 - Simple calculation of 1 mole repair
 - Rate $\frac{1}{4}$, 10 billion servers, 64 fragments,
 - 1 Mbit/sec bandwidth

Introspective Optimization

- Monitoring and adaptation of routing substrate
 - Optimization of Plaxton Mesh
 - Adaptation of second-tier multicast tree
- Continuous monitoring of access patterns:
 - Clustering algorithms to discover object relationships
 - Clustered prefetching: demand-fetching related objects
 - Proactive-prefetching: get data there *before* needed
 - Time series-analysis of user and data motion
- Continuous testing and repair of information
 - Slow sweep through all information to make sure there are sufficient erasure-coded fragments
 - Continuously reevaluate risk and redistribute data
 - Diagnosis and repair of routing and location infrastructure
 - *Provide for 1000-year durability of information?*

Can You Delete (Eradicate) Data?

- *Eradication* is antithetical to *durability!*
 - If you can eradicate something, then so can someone else! (denial of service)
 - Must have "eradication certificate" or similar
- Some answers:
 - Bays: limit the scope of data flows
 - Ninja Monkeys: hunt and destroy with certificate
- Related: Revocation of keys
 - Need hunt and re-encrypt operation
- Related: Version pruning
 - Temporary files: don't keep versions for long
 - Streaming, real-time broadcasts: Keep? Maybe
 - Locks: Keep? No, Yes, Maybe (auditing!)
 - Every key stroke made: Keep? For a short while?

First Implementation [Java]:

- Event-driven state-machine model
- Included Components
 - ✓ Initial floating replica design
 - Conflict resolution and Byzantine agreement
 - ✓ Routing facility (Tapestry)
 - Bloom Filter location algorithm
 - Plaxton-based locate and route data structures
 - ✓ Introspective gathering of tacit info and adaptation
 - Language for introspective handler construction
 - Clustering, prefetching, adaptation of network routing
 - ✓ Initial archival facilities
 - Interleaved Reed-Solomon codes for fragmentation
 - Methods for signing and validating fragments
- Target Applications
 - ✓ Unix file-system interface under Linux ("legacy apps")
 - Email application, proxy for web caches, streaming multimedia applications

OceanStore Conclusions

- OceanStore: everyone's data, one big utility
 - Global Utility model for persistent data storage
- OceanStore assumptions:
 - Untrusted infrastructure with a responsible party
 - Mostly connected with conflict resolution
 - Continuous on-line optimization
- OceanStore properties:
 - Provides security, privacy, and integrity
 - Provides extreme durability
 - Lower maintenance cost through redundancy, continuous adaptation, self-diagnosis and repair
 - Large scale system has good statistical properties

For more info:

- OceanStore vision paper for ASPLOS 2000
"OceanStore: An Architecture for Global-Scale Persistent Storage"
- OceanStore web site:
<http://oceanstore.cs.berkeley.edu/>