

Global-Scale Archival Goals

- Durability
 - Data is stored for centuries or longer.
- Verifiability
 - Data is not subject to substitution attacks.
- Availability
 - Data is accessible *most* of the time.
 - Where *most* is defined in n 9's of availability.
- Maintainability
 - System recovers from server and network failures.
 - Efficiently incorporates new resources.
- Atomicity
 - Updates are applied atomically.
- Privacy
 - Information is only visible to those who have access rights.
- Performance
 - Response time is bounded.



Long-Term Durability in OceanStore

Hakim Weatherspoon, Patrick R. Eaton, and John D. Kubiatowicz

<http://oceanstore.cs.berkeley.edu>



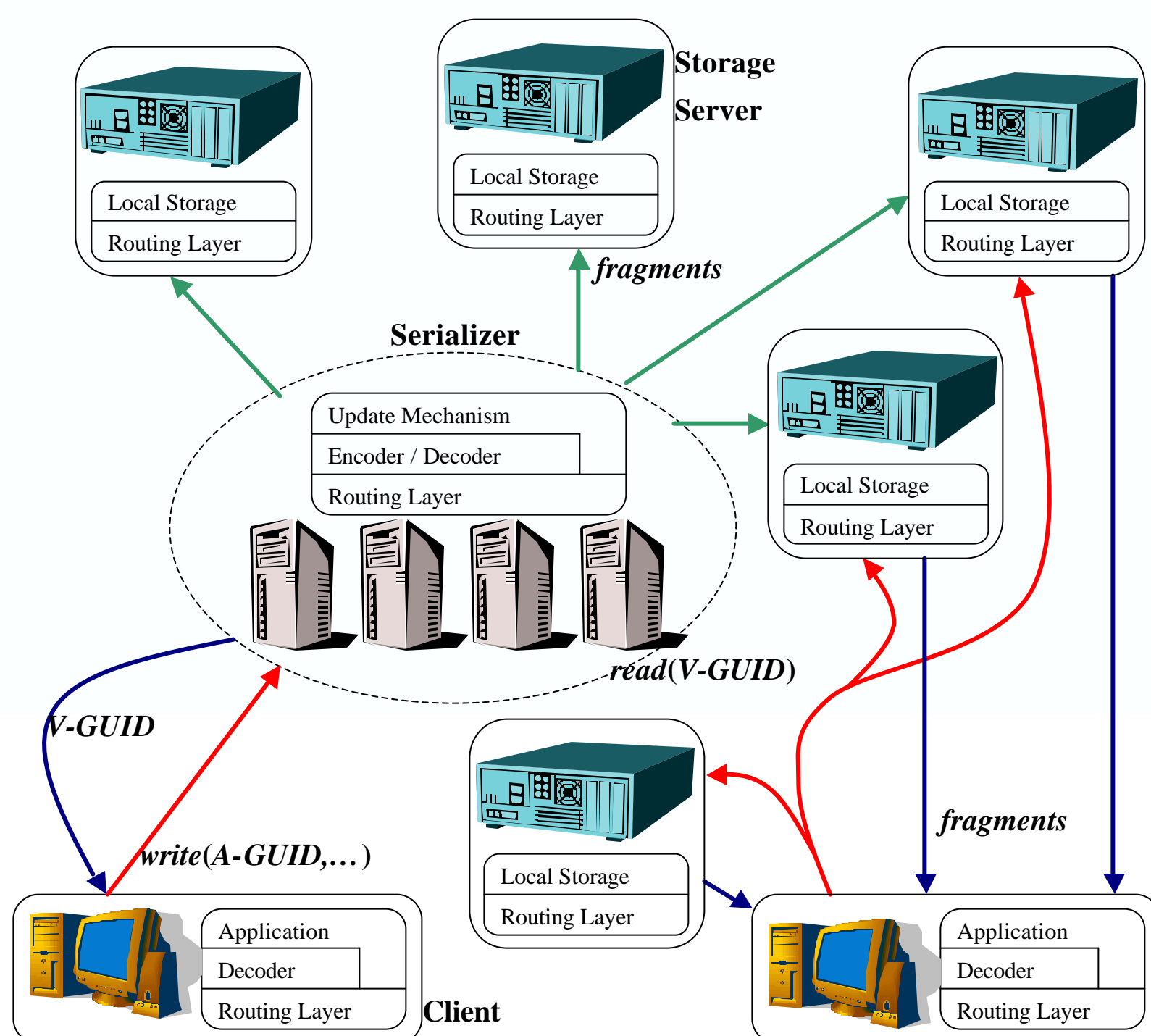
Durability

Archival Model

- Archive Data Structures.
 - *Archive* is a linearly ordered sequence of *versions*.
 - Each version is a read-only sequence of bytes.
 - E.g. an archive might be a *file*, a *directory*, or a *database record*.
- Naming.
 - Globally-Unique Identifier (GUID).
 - Archives are uniquely specified by *archive GUIDs* (A-GUIDs).
 - Within an archive, each version is specified by a *version GUID* (V-GUID).
 - Versions are immutable and provide for *time-travel*.
- Operations.
 - *Update Operations*.
 - Add versions to the end of the version sequence of a given archive.
 - *Read Operations*.
 - Read data from a specific version.
- *Serializer* provides consistency.
 - Entity in network that provides *atomicity*.
 - Provides an A-GUID to V-GUID mapping.
 - Creates a serial order over simultaneously submitted updates.
 - Verifies that the *client* has update privileges.
 - Atomically applies update to the archive and generates a new V-GUID.
 - Sends fragments from an update to *storage servers*.

Interface

- Generate new archive interface.
 - create(name, identity, keys) => A-GUID.
- Query Interface.
 - query(A-GUID, Specifier) => V-GUID.
 - Specifier => timestamp, version#, etc.
- Read interface.
 - read(V-GUID, offset, length) => data.
- Write interface.
 - write(A-GUID, data) => V-GUID.
 - append(A-GUID, data) => V-GUID.
 - replace(V-GUID, offset, data, allowbr) => V-GUID or null.
 - *allowbr* denotes whether operation allowed to generate branch.



Case for Erasure Codes

Background

- Erasure codes provide redundancy without overhead of replication.
 - Divide an object into m fragments.
 - Recode them into n fragments.
 - A rate $r = m/n$ code increases storage cost by a factor of $1/r$.
 - Key property is that original object can be reconstructed from *any* m fragments.
 - E.g. using an $r = 1/2$ code, divide a block into $m = 16$ fragments, and encode the original m fragments into $n = 64$ fragments.
 - Increases storage cost by a factor *four*.
- Example implementations
 - Reed-Solomon Codes.
 - Tornado Codes.
 - Interleaved Reed-Solomon.

Assumptions

- An archive is implemented on a collection of independently failing disks.
- Failed disks immediately replaced by new, blank ones.
- Each archival fragment for a given block is placed on a unique, randomly selected disk.
- A repair epoch.
 - Time period between a global sweep, where a repair process scans the system, attempting to restore redundancy.

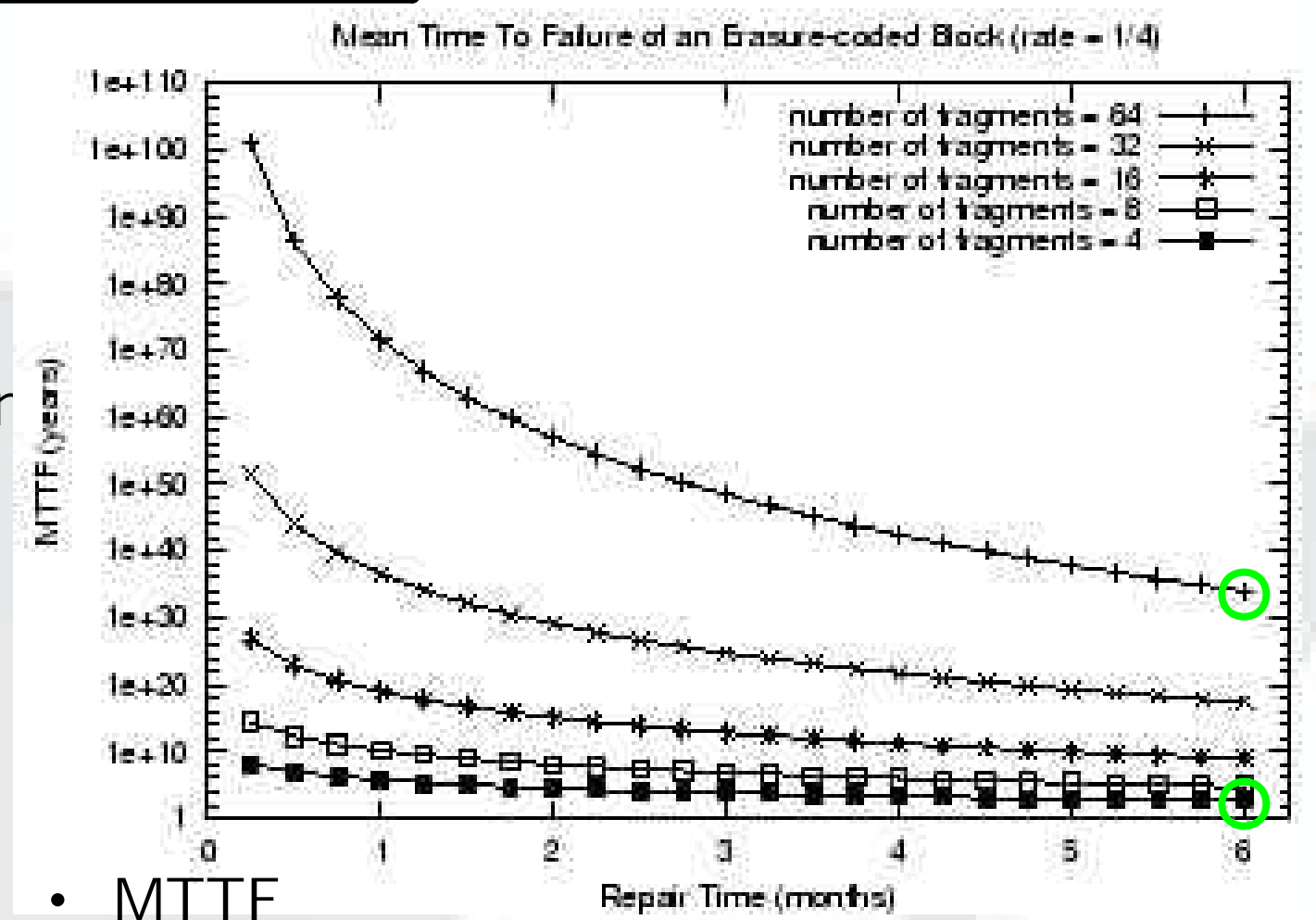
Availability

Exploits the statistical stability of a large number of components

$$P_o = \sum_{i=0}^{n-m} \binom{M}{i} \binom{N-M}{n-i} / \binom{N}{n}$$

- P_o - Probability that an object is available.
- n - total number of fragments.
- m - number of fragments needed for reconstruction.
- N - total number of machines in the world.
- M - number of currently unavailable machines.

- E.g. given 90% of a million machines availability:
 - $n = 16$ fragments, rate $r = 1/2$, yield 5 9's of availability.
 - $n = 32$ fragments, rate $r = 1/2$, yield 8 9's of availability.



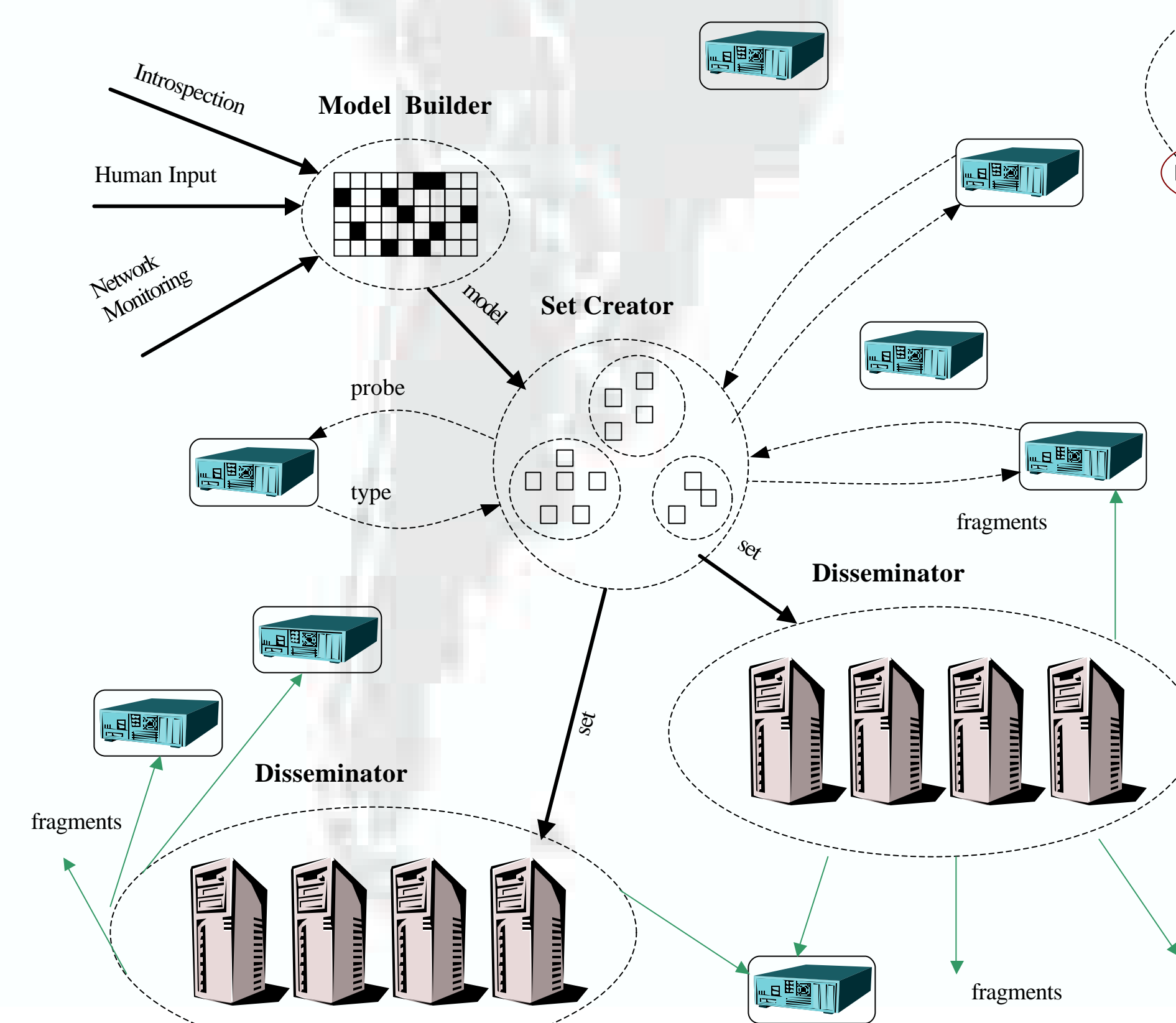
- MTTF
 - Grows exponentially with number of fragments.
 - Grows super-linearly with decreasing repair times.
 - E.g. MTTF = 10^{35} years for a particular block.
 - with $n = 64$ fragments, rate $r = 1/2$ and repair epoch $e = 6$ months.
 - MTTF = 35 years for replication with same storage cost and repair epoch!

Can this be Real?

- Three requirements must be met:
 - Failure Independence.
 - Efficient Repair.
 - Data Integrity.

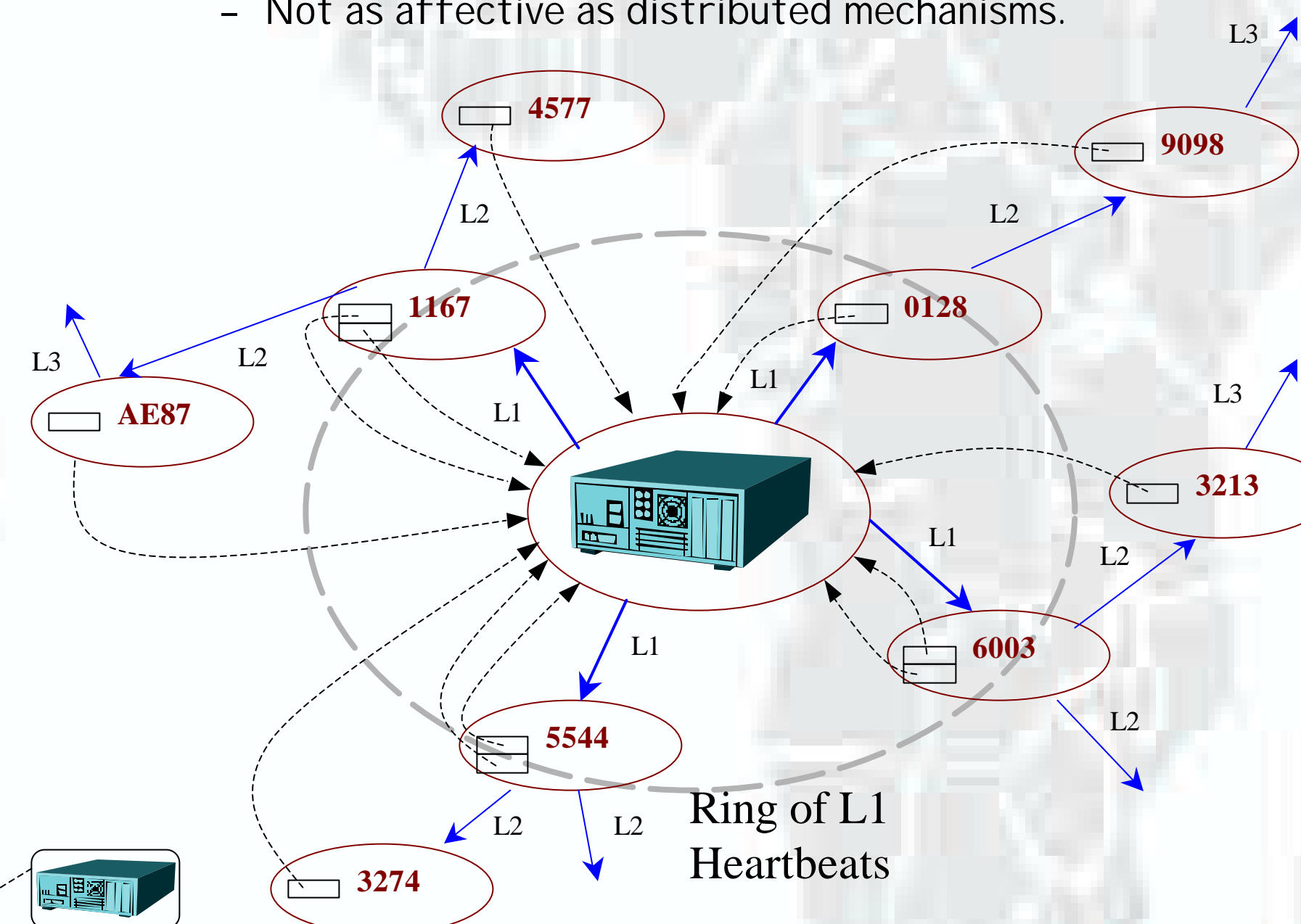
Failure Independence: Effective Dissemination

- Model Builder.
 - Takes input from various sources.
 - Builds a model of failure correlation.
- Set Creator.
 - Queries random nodes for properties.
 - Uses the model to compute *Dissemination Sets*.
 - Sets of storage servers that fail with low correlation.
- Disseminator.
 - Sends one fragment to each storage server in a set.



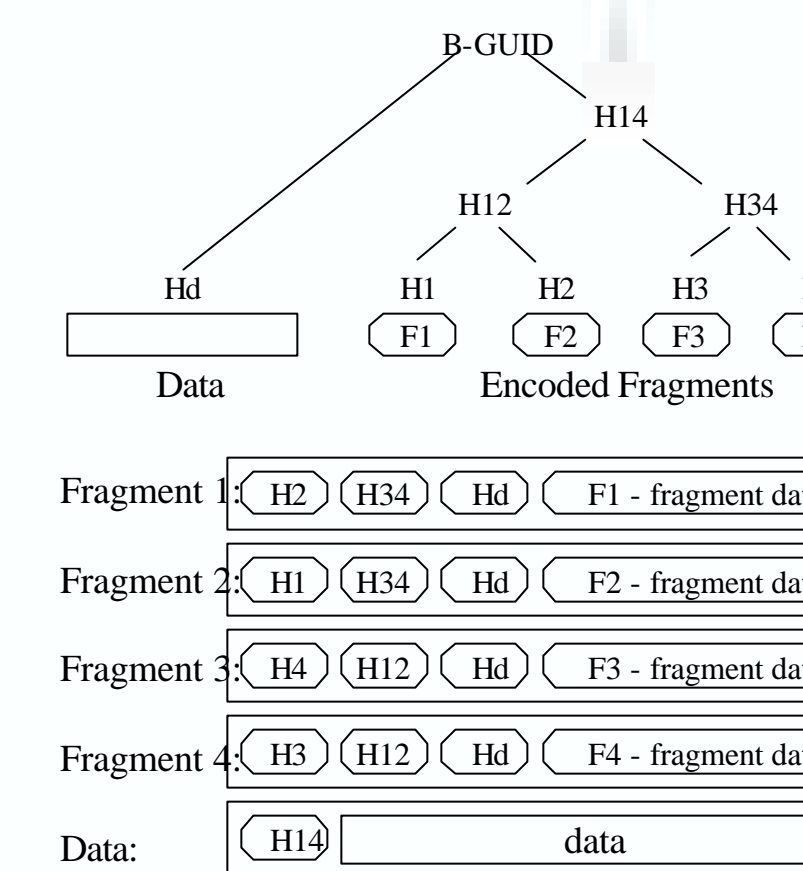
Efficient Repair

- Local.
 - Durability enhancement techniques such as RAID.
 - Servers proactively copy data to new disk.
 - Servers periodically verify the integrity of local data.
- Distributed.
 - Exploit Tapestry's distributed information and locality properties.
- Global.
 - Not as effective as distributed mechanisms.



Data Integrity

- Erasure Codes require precise identification of failed/corrupted fragments.
 - Use cryptographically secure hash algorithm to detect corrupted fragments.
- Verification Tree:
 - n is the number of fragments.
 - store $\log(n) + 1$ hashes with each fragment.
 - Total of $n(\log(n) + 1)$ hashes.
- Top hash is a *block GUID* (B-GUID).
 - Fragments and blocks are self-verifying.



Conclusion

- The OceanStore archive combines several techniques to satisfy the goals of a global-scale archival system.
 - Erasure codes provide durability and availability.
 - Verification trees provide verifiability
 - Introspective failure analysis, automatic repair, and location independent routing promote maintainability.
 - The serializer provides atomicity.
 - End-to-end encryption (not discussed in this poster) provides privacy.
- Result.
 - Archival storage that has the potential to store data indefinitely.
- Mechanisms are implemented in current prototype.
 - First prototype, code name *Puddle*.
 - Implemented *archival model*, NFS front-end using archival interface, and ran Andrew Benchmark with 400 client, serializers, and storage servers in network.
 - Second prototype, code name *Pond*.
 - Implemented on top of a Staged Event Driven Architecture.
 - Implementing Dissemination and repair algorithms.

Enabling Technology: Tapestry

Tapestry Operations

- Tapestry is a *location-independent* routing infrastructure.
 - Fragments and serializers are both named by opaque bit-strings (GUIDs).
 - Tapestry can perform location-independent routing of messages directly to objects using only GUIDs.
 - Tapestry is an IP overlay network that uses a distributed, fault-tolerant architecture to track the location of every object in the network.
 - Tapestry has two components: a *routing mesh* and a *distributed directory service*.
- Routing in Tapestry.
 - Nodes are connected to other nodes via neighbor links.
 - Any node can route to any other by resolving one digit at a time:
 - e.g. 1010 => 2218 => 9098 => 7598 => 4598
 - Each GUID is associated with one particular *Root* node.

