Memo (immutable msg)
Differences to ancestor memos

Projection based on known memos

Action

Slab behaviors:
- Store "memos" (in RAM or otherwise)
- Has specific storage/network/compute quotas (MB/GB/TB+)
- Track peering data with memo replicas (gossip, not all replicas)
- Report approx. expected lifetime (seconds~months+) to peers
- Calculate a "durability score" for all memos
- Push replicas of memos below durability-threshold to peers
- Send heartbeat packets to peer slabs
- Recalculate replication factor reduction for stale peers
- Accepts memo replicas from peers when below quotas
- Evicts LRU memos when network/storage quotas are exceeded
- Update peer slabs for memos leaving/entering

Subjects/Memos
- A subject is simply an enumeration, and exists only as a coalescence of it's projected memos
- Memos are immutable, originated by a slab
- The "HEAD" of a given subject consists of the most causally recent memos, which no other memos are yet known to descend.

Notes:
Term for memos that share the same topic which are non-identical
Explain replica peering vs referential peering

Slab 1
(A slab is similar to what most think of as a "node", except that you may have multiple slabs corresponding to different processor cores, or different services, for instance)

X1.1 (Existing Memo)
Create Index node X1 (various links)

All Slabs must initialize with at least one memo from the root index node. It needn't be fresh

R1.1 (Existing Memo)
Create Subject R1 – Color=Red

Last known "head" of Record R1 Gotten from previous transaction or pushed by a peer slab for durability assurance

Causal Context for Slab 1
(contains references to objects in the slab)

X1.1 (Existing Memo)
Create Index node X1 (various links)

X3.2 (New Memo)
Update Index node X3 to point to new head of Record R1 (R1.2)

X1.2 (New Memo)
Update Index node X1 to link to new head of Index Node X2 (X2.2)

Compaction
When context grows too large. May happen mid-transaction.

Create new parent index memos linking to child index memos from the causal context.

X2.2 (New Memo)
Update Index node X2 to link to new head of Index Node X3 (X3.2)

X1.2 (New Memo)
Update Root Index node X1 to link to new head of Index Node X2 (X2.2)

Replace context with latest head of root index node, plus any descendents of non-root nodes which are not referenced by same.

Transaction 1

T1.1 (New Memo)
Begin Transaction T1

R1.2 (New Memo)
Record1 – Set Color=Blue

Traverse main index to find relevant leaf node

Issue a new memo for leaf node, add to context

X3.2 (New Memo)
Update Index node X3 to include ref to new head of Record R1 (R1.2)

T1.1 (New Memo)
Commit Transaction T1

TODO:
1. review how index node memos are to be selectively acknowledged in order to keep traffic manageable.(Maybe merge lazily only on the basis of context swaps?)
2. show how keyframes are generated
3. show how projection works

Q: how do i serialize a causal context for api users?
A: broadcast memo hunt OR memo index

Q: how do I determine when to share causal context? (or otherwise how do I control merging such that its layered

NOTES:
* Memos reference beacons
* index keyframes associate beacon vector to each slot
* All keyframes associate beacon vector to the overall frame (consider compression schemes inside each keyframe?)
* Beacons never get handed off, they're born, and they die
* Beacons are selected by their longevity, consistency, and conformance to the target interval
* All memo propagation can be vicarious via plumtree
* keyframes are merged commutatively by comparing beacon vectors for each slot
* each slab may be a beacon
* beacons which are unpopular will tend to self-terminate
* beacons which are popular will tend to persist
* each slab's probability of activating its beacon function is inversely proportionate to the health of it's beacon set
* does plum tree allow for popularity estimation?

Upon traversal/retrieval, copies of the index memos, and any other retrieved memos are stored at least momentarily.

X2.1 (Existing Memo)
Create Index node X2 (various links)

X3.1 (Existing Memo)
Create Leaf node X3 (various links)

X1 - Root Node for Index 1

X1.2 (New Memo)
Update Index node X1 to link to new head of Index Node X2 (X2.2)

Descends

X1.1 (Existing Memo)
Create Index node X1 (various links)

X2 - Node for Index 1

X2.2 (New Memo)
Update Index node X2 to link to new head of Index Node X3 (X3.2)

Descends

X2.1 (Existing Memo)
Create Index node X2 (various links)

X3 - Leaf Node for Index 1

X3.2 (New Memo)
Update Index node X3 to point to new head of Record R1 (R1.2)

Descends

X3.1 (Existing Memo)
Create Leaf node X3 (various links)

Link

Link

Link

Link

Assumes fixed 3-tier unbalanced index tree with 1024 per slots each node (1024^3 = 1 Billion records)
Would likely be a 4-tier system. Balanced-tree indexing work may be possible, but is not presently included in this design