Skip Net (9/11/05)

Introduction

Based on Skip Lists

What are they?

An *alternative to binary trees*,
Binary trees work well with random input but do not work well say when inserting items in order

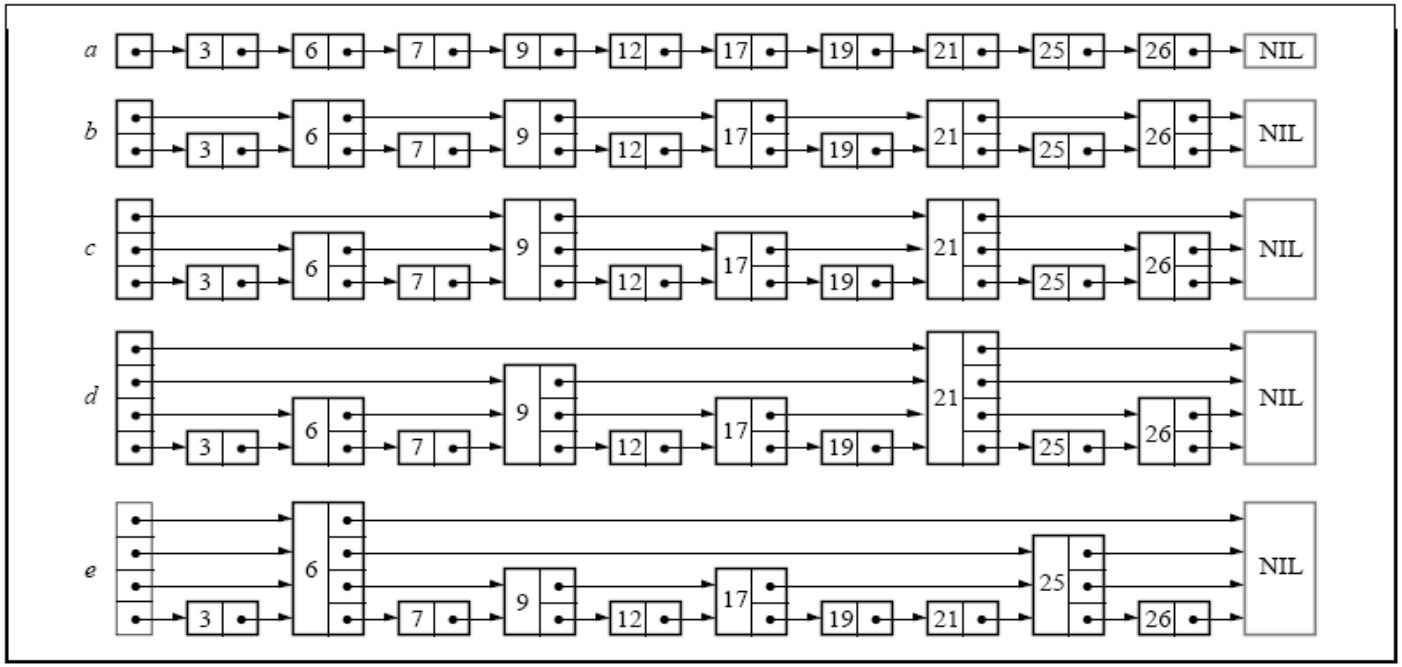
Thus in general, binary trees need to be *load balanced*

Can we avoid load balancing of binary trees by introducing some form of randomness in the load balancing procedure – what if the position (level) of a node in the tree is determined by a random number generator?

Skip List: a sorted linked list with extra links

Introduction

Skip List: a sorted linked list where some nodes have pointers that *skip over* many list elements, why?



How to use the extra pointers:

For example, in case (b), as long as you have not reached a node with key > search-key, use level 2 pointers, then use level 1, $\lceil n/2 \rceil + 1$

In general,

level = top-level,

while not found,

while key > search-key use *level*

level = *level* – 1 /*use lower level */

log n with only doubling the number of pointers

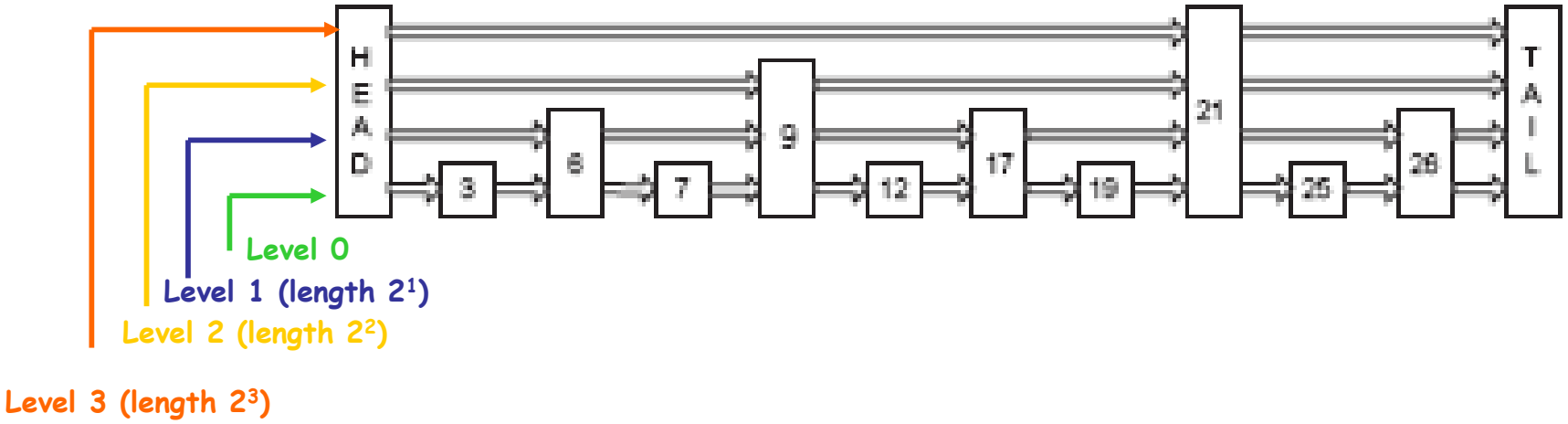
Skip List

a sorted linked list in which some nodes are supplemented with pointers that *skip over* many list elements

"perfect" Skip List: the height of the *ith* node is the exponent of the largest power-of-two that divides *i*.

Pointers at level *h* have length 2^h (i.e., they traverse 2^h nodes).

A perfect Skip List supports searches in $O(\log N)$ time.



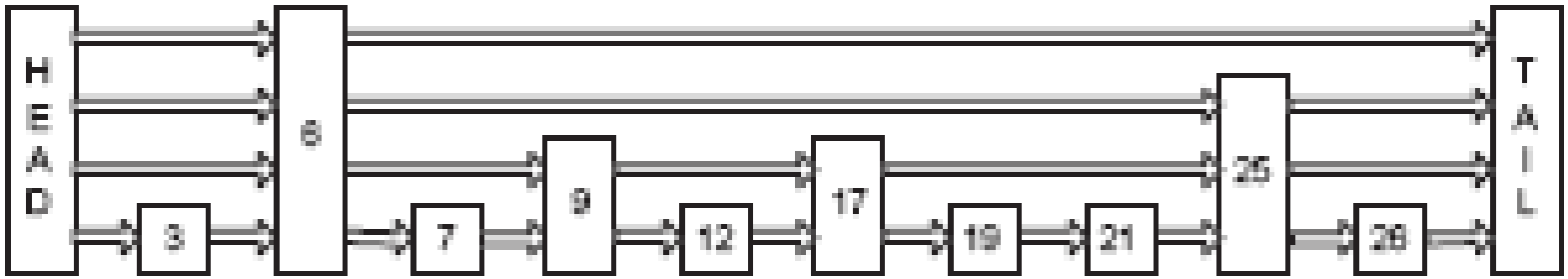
Probabilistic Skip List

Insertions and deletions are very expensive

A probabilistic scheme for determining node heights while maintaining *O*(log*N*) searches *with high probability*

Each node chooses a height such that the probability of choosing height *h* is $1/2^h$. (why? Hint: number of nodes at each level)

Thus, with probability $\frac{1}{2}$ a node has height 1, with probability $\frac{1}{4}$, it has height 2, etc.



How to use them in P2P

- *in-memory* data structure

link together *distributed computer nodes*

- traversed from its head node

traversal may *start from any node* in the system (ring)

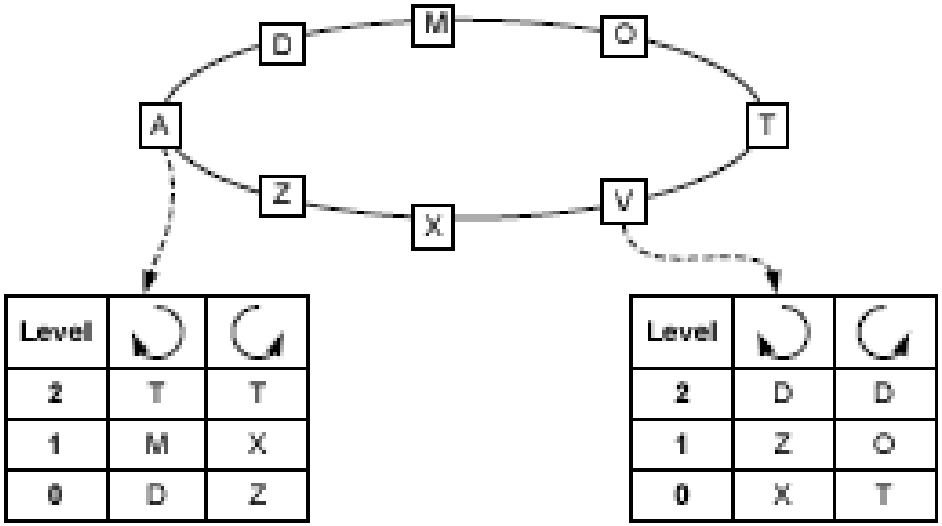
- a highly variable number of pointers per data record and a substantially different amount of traversal traffic at each data record

the state and processing overhead of all nodes roughly the *same*

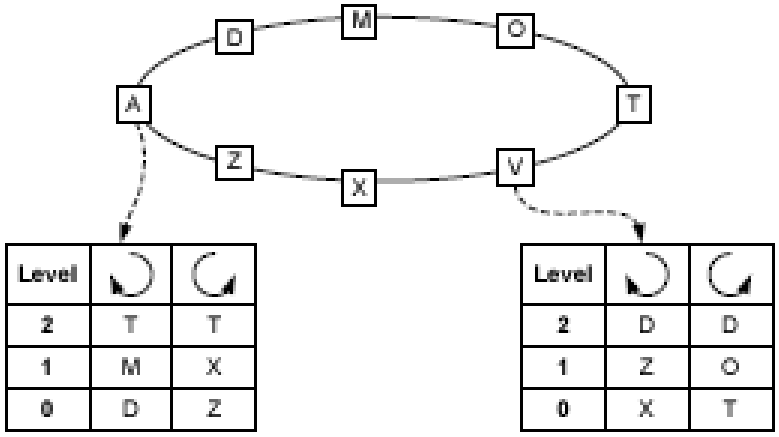
The Skip Net Structure

Replace data records with computer nodes, using the *string name IDs* of the nodes as the data record keys

Form a ring instead of a list (the ring is *doubly-linked* to enable path locality)



The Skip Net Structure (continued)



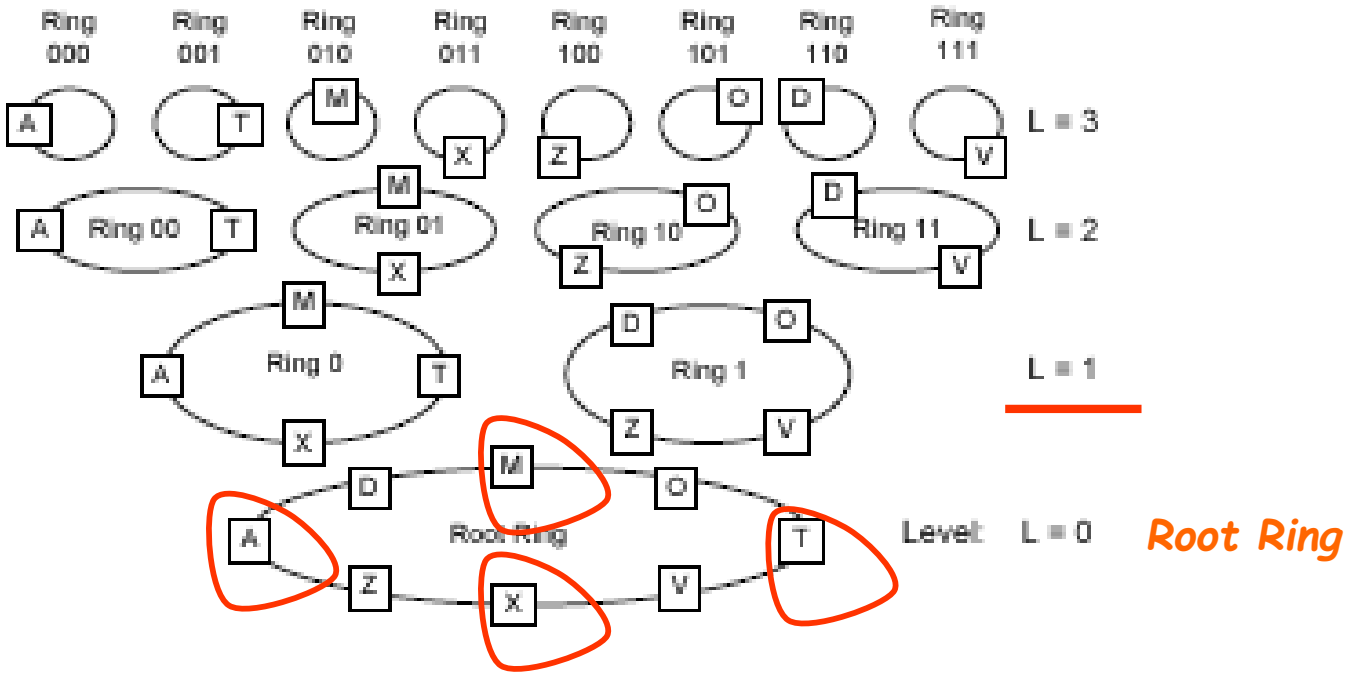
Each SkipNet node stores **2 logN** pointers, where N is the number of nodes in the overlay system

Routing Table or R-Table of a node (its set of pointers)

The pointers at level *h* of a given node's routing table point to nodes that are roughly 2^h nodes to the left and right of the given node.

"perfect" SkipNet: each level *h* pointer traverses exactly 2^h nodes.

The Skip Net Structure (continued)



All nodes are connected by the *root ring* formed by each node's pointers at level 0. – *Note that these nodes are sorted by Name ID.*

Pointers at level 1 point to nodes that are 2 nodes away -> the overlay nodes are divided into 2 disjoint rings.

Pointers at level 2 form 4 disjoint rings of nodes, and so forth.

Note: rings at level *h* + 1 are obtained by splitting a ring at level *h* into two disjoint sets, each ring containing every second member of the level *h* ring.

The Probabilistic Skip Net Structure

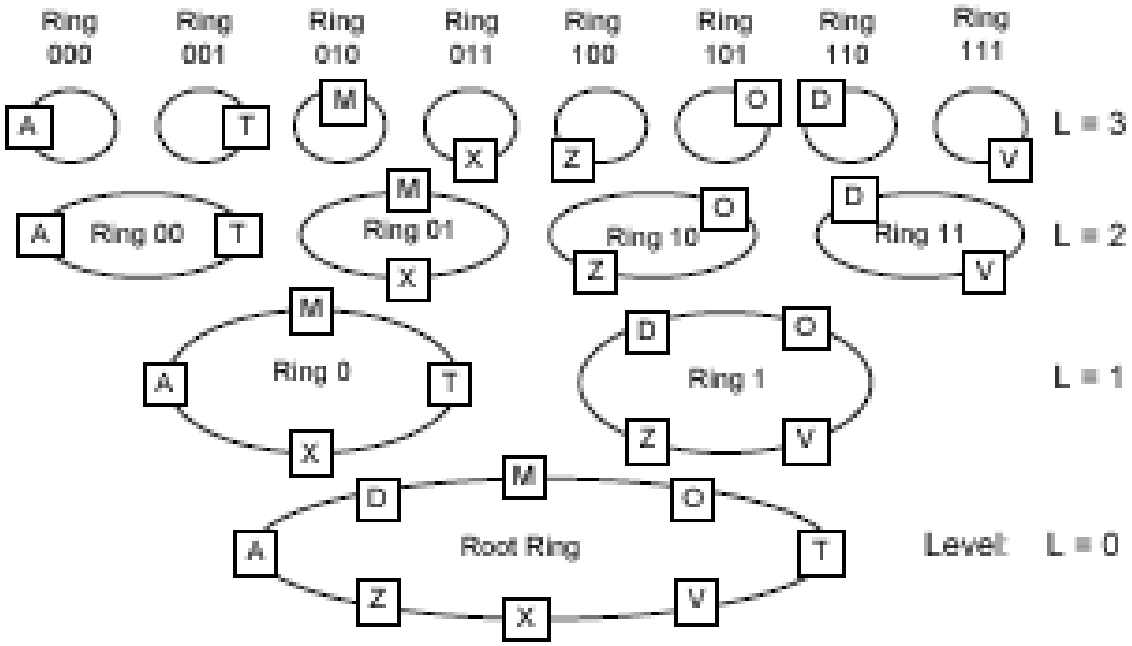
Each ring at level h is split into two rings by having each node *randomly and uniformly* decide to which of the two rings it belongs

- Insertion and deletion affects only *two other nodes in each ring* to which the node has randomly chosen to belong

- *A* pointer at level h still skips 2^h nodes in *expectation* and routing is possible in *O*(log N) forwarding hops *with high probability*

Skip Net's Numeric ID Space

Each node's random choice of ring memberships can be encoded as a unique binary number: the node's *numeric ID*



The first *h* bits of the number determine ring membership at level *h*.

For example, node *X*'s numeric ID is 011 and its membership at level 2 is determined by taking the first 2 bits of 011, which designate Ring 01.

The Probabilistic Skip Net Structure (continued)

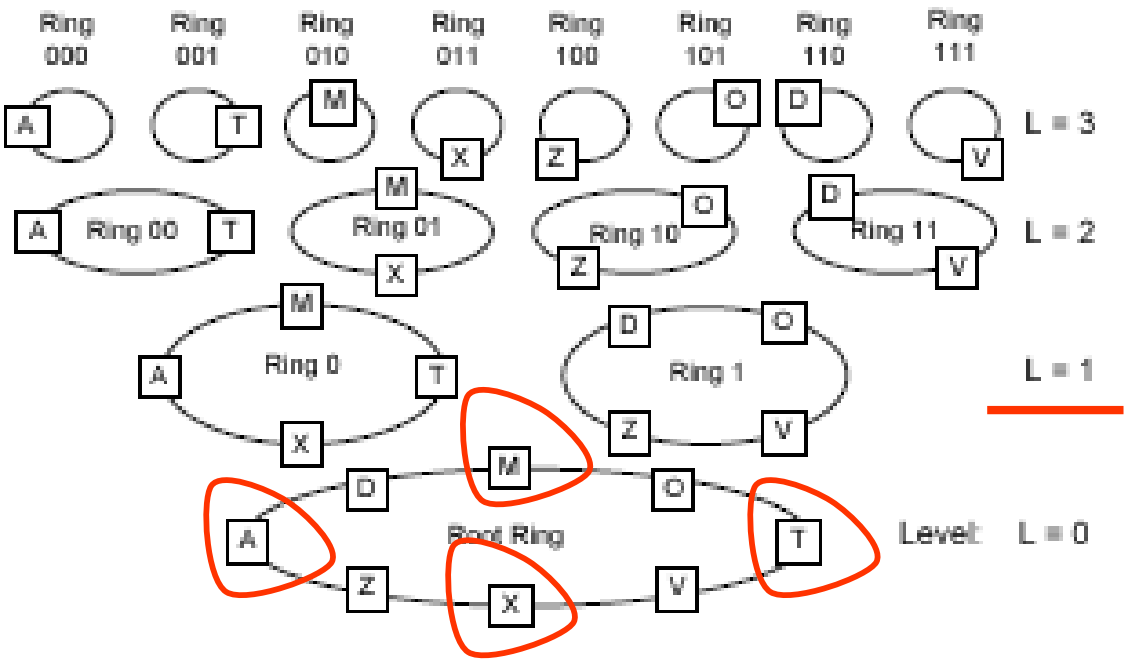
does not require hashing to generate nodes' numeric IDs; it only requires that numeric IDs are random and unique

Node IDs and Numeric IDs

Numeric IDs of nodes are unique; they can be thought of as *a second address space* maintained by the same SkipNet

- SkipNet's string (name) address space is populated by node ***name IDs*** that are *not* uniformly distributed throughout the space
- SkipNet's numeric address space is populated by node ***numeric IDs*** that *are* uniformly distributed

The Skip Net Structure (continued)



The root ring, at the bottom, is **sorted** by name ID and, collectively, the top-level rings are sorted by numeric ID.

For any given node, the SkipNet rings to which it belongs precisely form a Skip List.

If you construct a trie on all nodes' numeric IDs, the nodes of the resulting trie would be in one-to-one correspondence with the SkipNet rings.

Skip Net's Address Spaces

Skip Net maintains two address spaces for the nodes:

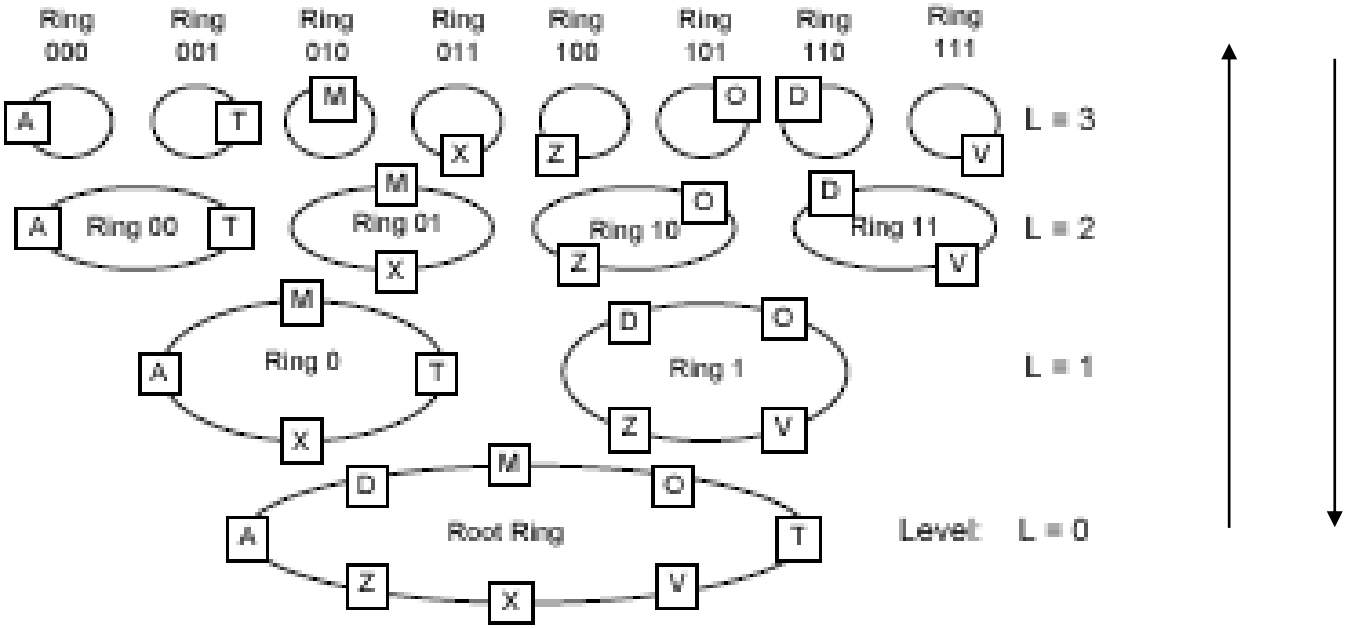
- Node ID space (which is not uniformly distributed through the address space)

- Numeric ID space (it does not require hashing, it suffices that they are random and unique)

CHORD and Probabilistic Skip Net

SkipNet's routing pointers are exponentially distributed in a manner similar to Chord's: The pointer at level *h* hops over 2^h nodes in expectation.

- Chord's routing pointers skip over 2^h in the numeric space
- SkipNet's pointers:
  - when considered from level 0 upward, skip over 2^h nodes in the name ID space
  - when considered from the top level downward, skip over 2^h nodes in the numeric ID space.



CHORD and Probabilistic Skip Net

Chord guarantees $O(\log N)$ routing and node insertion performance by *uniformly distributing node identifiers in its numeric address space*.
Hash(nodeID) $\in [0, 2^m]$

SkipNet guarantees $O(\log N)$ performance of node insertion and routing in both the name ID and numeric ID spaces by *uniformly distributing numeric IDs and leveraging the sorted order of name IDs.*

Skip Net

By using *names* instead of hashed identifiers to *order nodes*, natural locality based on the names of objects is preserved

Arrange content in name order rather than randomly

- Path locality, and
- Content locality

Q: Can we achieve the same using an order-preserving hash function?

Routing by Name ID

Similar with Skip Lists

Follow pointers that route closest to the intended destination

At each node, a message will be routed *along the highest level pointer* that does not point past the destination value.

Routing terminates when the message arrives at a node whose name ID is closest to the destination.

Since nodes are ordered by name ID along each ring and a message is never forwarded past its destination,
all nodes encountered during routing have name IDs between the source and the destination

Given a source node and a destination node, **if** destination > source, move clockwise **else** move counter clockwise

Routing by Name ID

Content locality

Use the name id of an item to store it (map it) to the p2p nodes

Use the "name" of the data item to store it at the node with the same name (or the same prefix)

Incorporating the node's name ID into a data name ID guarantees that the data item will be stored at that node

Example: doc-name -> kostas.microsoft.com/doc-name

Path locality

Use com.microsoft.kostas so that all nodes within the com.microsoft prefix share a single DNS suffix

Routing by Name ID (continued)

When a message originates at a node whose name ID shares a common prefix with the destination:

All nodes traversed by the message have name IDs that share that same prefix

If the source name ID and the destination name ID share no common prefix:
A message can be routed in either direction.

For the sake of fairness, randomly pick a direction so that nodes whose name IDs are near the middle of the sorted ordering do not get a disproportionately large share of the forwarding traffic.

Routing by Name ID (continued)

The number of message hops when routing by name ID is $O(\log N)$ with high probability.

The key observation of this scheme is that routing by name ID traverses only nodes whose name IDs share a non-decreasing prefix with the destination ID.

Routing by Numeric ID

Begin by examining nodes in the *level 0* ring until a node is found whose numeric ID matches the destination numeric ID in the 1st digit.

Jump up to this node's *level 1* ring, which also contains the destination node.

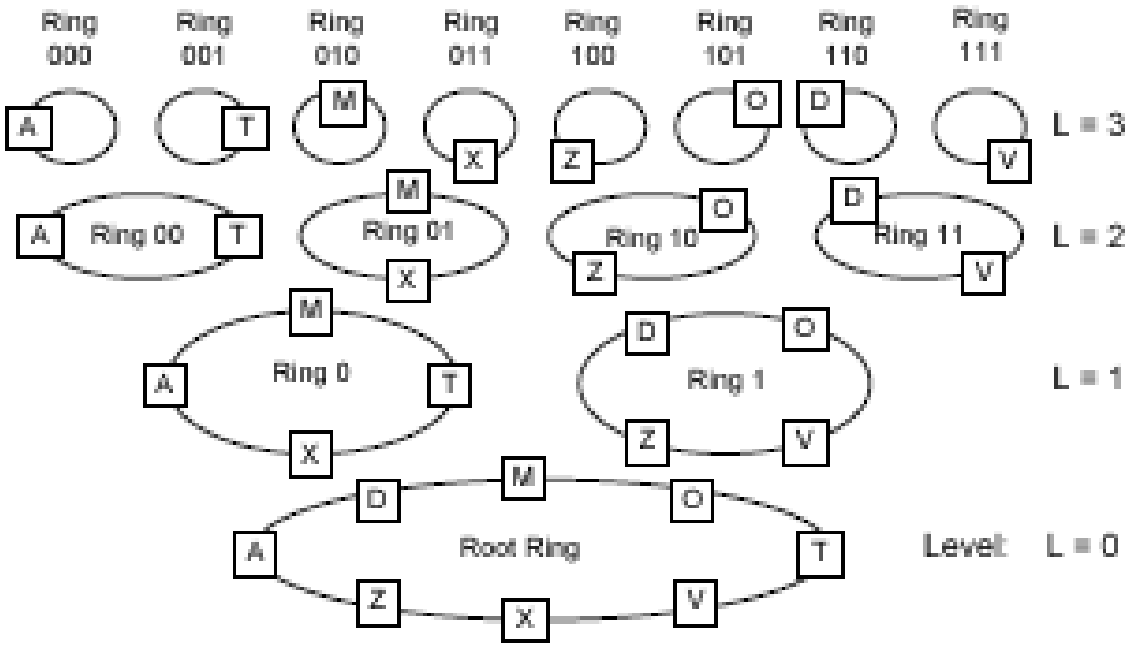
> Examine nodes in this level 1 ring until a node is found whose numeric ID matches the destination numeric ID in the 2nd digit.
>
> Again, this node's *level 2* ring must also contain the destination node, and thus the routing operation proceeds in this level 2 ring.

This procedure repeats until we cannot make any more progress — we have reached a ring at some level h such that none of the nodes in that ring share h + 1 digits with the destination numeric ID.

We must now deterministically choose one of the nodes in this ring to be the destination node.

Define the destination node to be the node whose numeric ID is numerically closest to destination numeric ID amongst all nodes in this highest ring.

Routing by Numeric ID (continued)



Example: Route a message from node A to destination 1011

A will first forward the message to node D because D is in ring 1.
D will then forward the message to node O because O is in ring 10.
O will forward the message to Z because it is not in ring 101.
Z will forward the message onward around the ring (and hence back) to O for the same reason.

Since none of the members of ring 10 belong to ring 101, node O will be picked as the final message destination because its numeric ID is closest to 1011 of all ring 10 members.

Routing

The number of message hops when routing by numeric ID is $O(\log N)$ with high probability.

- Route by name ID: from top (higher level h) downwards
- Route by numeric ID: from bottom (lowest level, level 0) upwards

Efficient searches by name ID, because for any given node, the SkipNet rings to which it belongs precisely form a Skip List.

Efficient searches by numeric ID, because, if you construct a trie on all nodes' numeric IDs, the nodes of the resulting trie would be in one-to-one correspondence with the SkipNet rings.

To map items
Use name ID (for content locality)
Use Numeric ID (for load balancing)

Node Join

A node wanting to join the network must know at least one node inside the network and sends a JOIN request to that node.

- Select a numeric ID either randomly or use hashing

Where to join (position in the rings):

- Phase 1: First, find your position at each level, the correct ring at the level (use Numeric ID)
- Phase 2: Then, find the correct position in the ring (must be sorted by name ID)

Node Join

Phase 1
First *find the toplevel ring* that corresponds to the newcomer's numeric ID.

This amounts to routing a message to the newcomer's numeric ID.

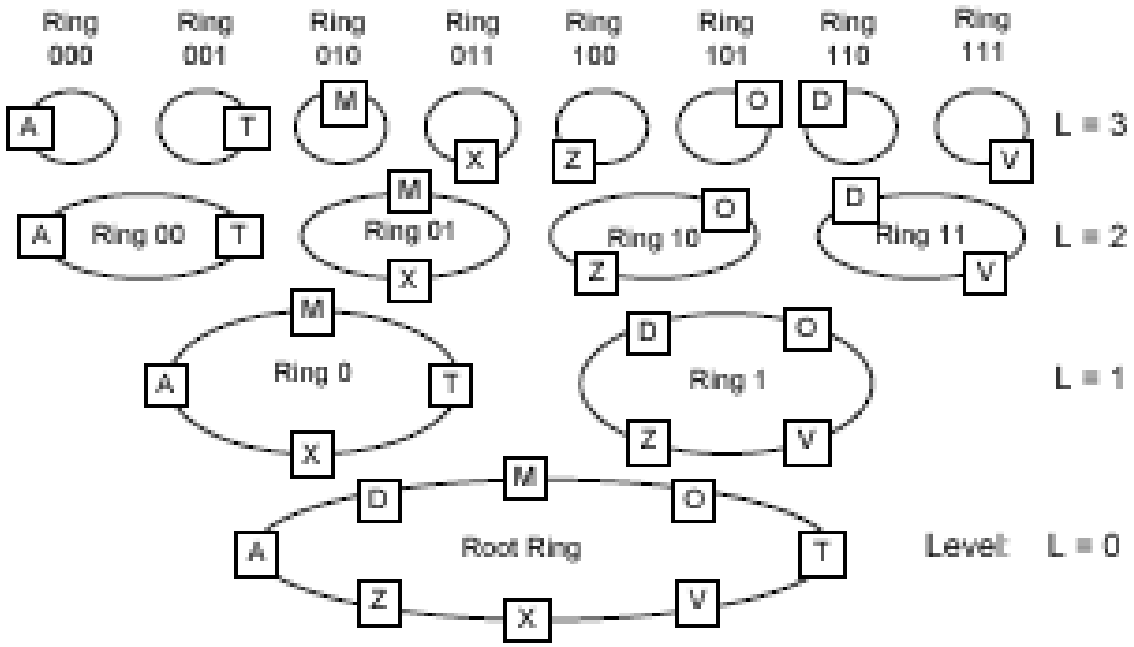
Phase 2 (the toplevel ring is found, search by name ID)
The newcomer then finds its neighbors in this toplevel ring, using a search by name ID within this ring only.

Starting from one of these neighbors, the newcomer searches for its name ID *at the next lower level* and thus finds its neighbors at this lower level.

This process is repeated for each level until the newcomer reaches the root ring.

For correctness, the existing nodes only point to the newcomer after it has joined the root ring; the newcomer then notifies its neighbors in each ring that it should be inserted next to them.

Node Join (continued)



Example: Insert node O

STEP 1
Node O initiates a search by *numeric ID* for its own ID (101) and ends up at node Z in ring 10 since that is the highest nonempty ring that shares a prefix with node O's numeric ID.

STEP 2:
Since Z is the only node in **ring 10**, Z concludes that it is both the clockwise and counter-clockwise neighbor of node O in this ring.

To find node O's neighbors in the next lower ring (**ring 1**), Z forwards the insertion message to D. Node D then concludes that D and V are the neighbors of node O in ring 1.

Similarly, node D forwards the insertion message to node M in the **root ring**, who concludes that node O's level 0 neighbors must be M and T.

STEP 3: The insertion message is returned to node O, who then instructs all of its neighbors to insert it into the rings.

Node Join (continued)

A newcomer searches for its neighbors at a certain level only after finding its neighbors at all higher levels.

As a result, the search by name ID will traverse only a few nodes within each ring to be joined: The range of nodes traversed at each level is limited to the range between the newcomer's neighbors at the next higher level.

Therefore, with high probability, a node join in SkipNet will traverse $O(\log N)$ hops

Node Departure

SkipNet can route correctly as long as the bottom level ring is maintained.

All pointers but the level 0 ones can be regarded as routing *optimization hints*, and thus are not necessary to maintain routing protocol correctness.

Therefore (like Chord and Pastry) SkipNet maintains and repairs the upper-level ring memberships by means of *a background repair process*.

In addition, when a node voluntarily departs from the SkipNet, it can *proactively notify* all of its neighbors to repair their pointers immediately.

To maintain the root ring correctly, each SkipNet node maintains a leaf set that points to *additional nodes along the root ring, for redundancy*.

In the current implementation a leaf set size of 16.

More next time