SKIP LIST & SKIP GRAPH

James Aspnes
Gauri Shah

Many slides have been taken from the original presentation by the authors

Definition of Skip List

A skip list for a set $\boldsymbol{L}$ of distinct (key, element) items is a series of linked lists $\boldsymbol{L}_0$, $\boldsymbol{L}_1$, … , $\boldsymbol{L}_h$ such that

Each list $\boldsymbol{L}_i$ contains the special keys $+\infty$ and $-\infty$
List $\boldsymbol{L}_0$ contains the keys of $\boldsymbol{L}$ in non-decreasing order
Each list is a subsequence of the previous one, i.e.,

$$\boldsymbol{L}_0 \supset \boldsymbol{L}_1 \supset \ \dots \supset \boldsymbol{L}_h$$

List $\boldsymbol{L}_h$ contains only the two special keys $+\infty$ and $-\infty$

Skip List

(Idea due to Pugh '90, CACM paper)

Dictionary based on a probabilistic data structure.

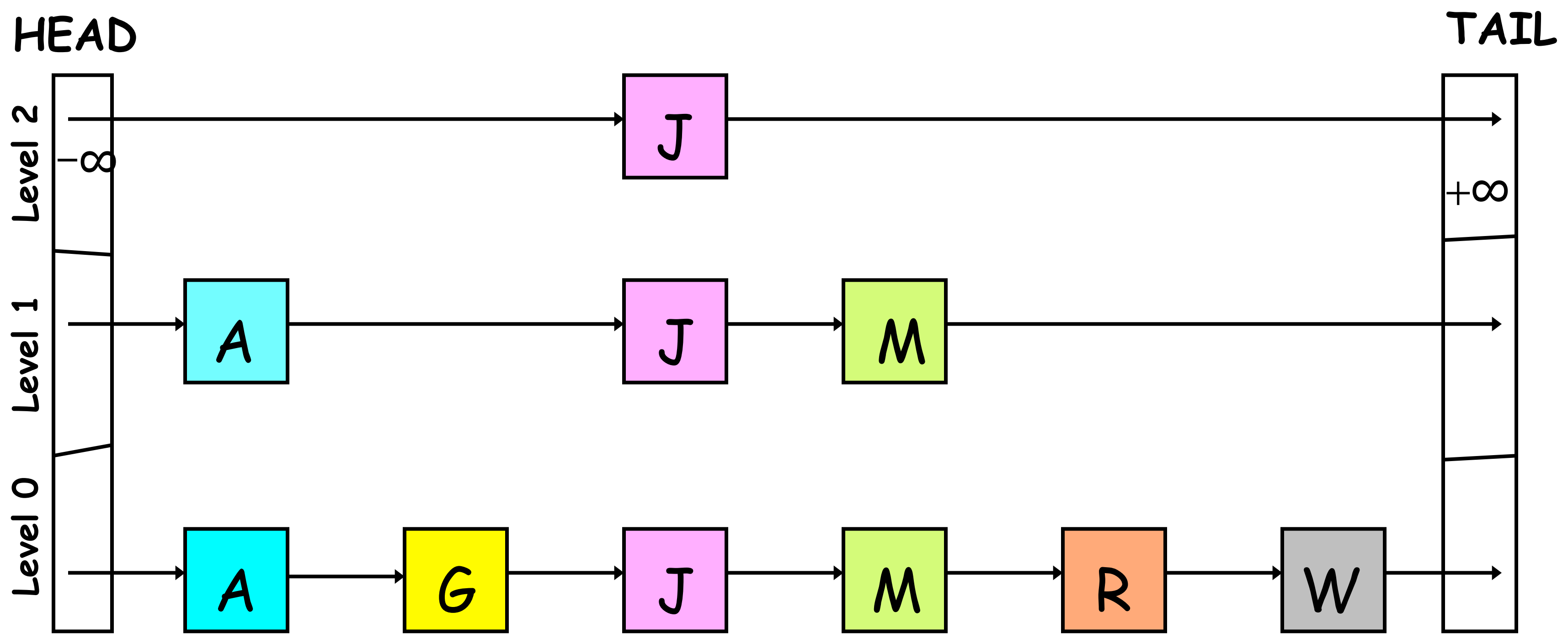
Allows efficient search, insert, and delete operations.

Each element in the dictionary typically stores additional useful information beside its search key. Example:

<student id. Transcripts>	[for University of Iowa]
<date, news>	[for Daily Iowan]

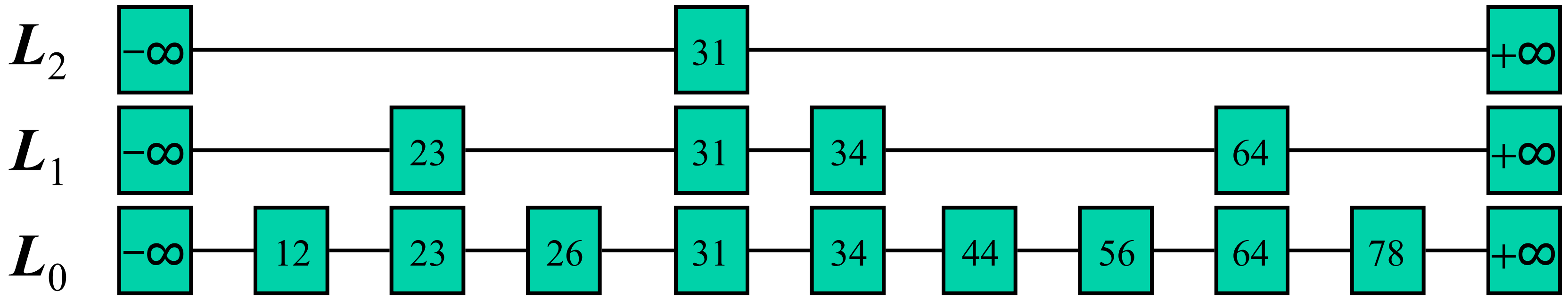
Probabilistic alternative to a balanced tree.

Skip List



Each node linked at higher level with probability 1/2.

Another example



Each element of L_i appears in $L._{i+1}$ with probability p.
Higher levels denote express lanes.

Searching in Skip List

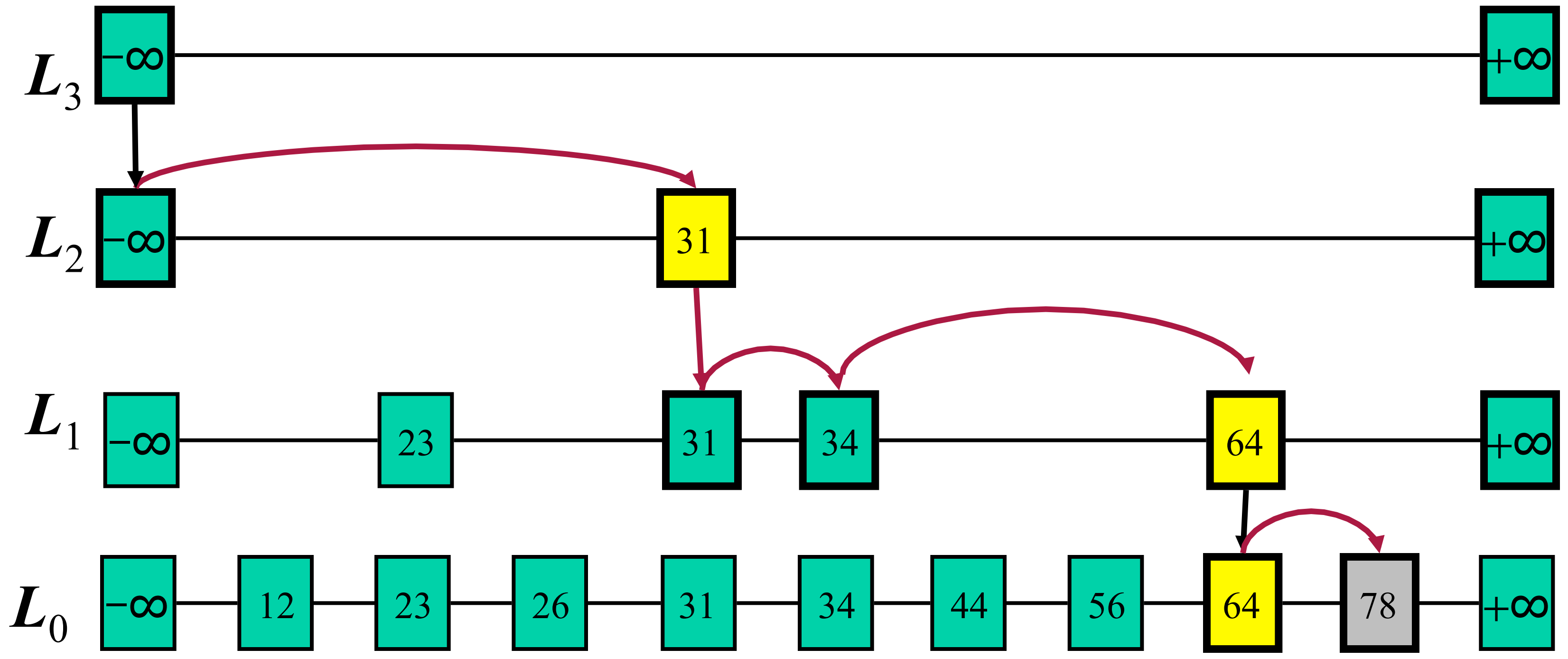
Search for a key ***x*** in a skip:

Start at the first position of the top list
At the current position ***P***, compare ***x*** with ***y*** **≥** key after P

x = ***y*** **->** return ***element*** (***after*** (***P***))
x > ***y*** -> "scan forward"
x < ***y*** -> "drop down"

– If we move past the bottom list, then no such key exists

Example of search for 78



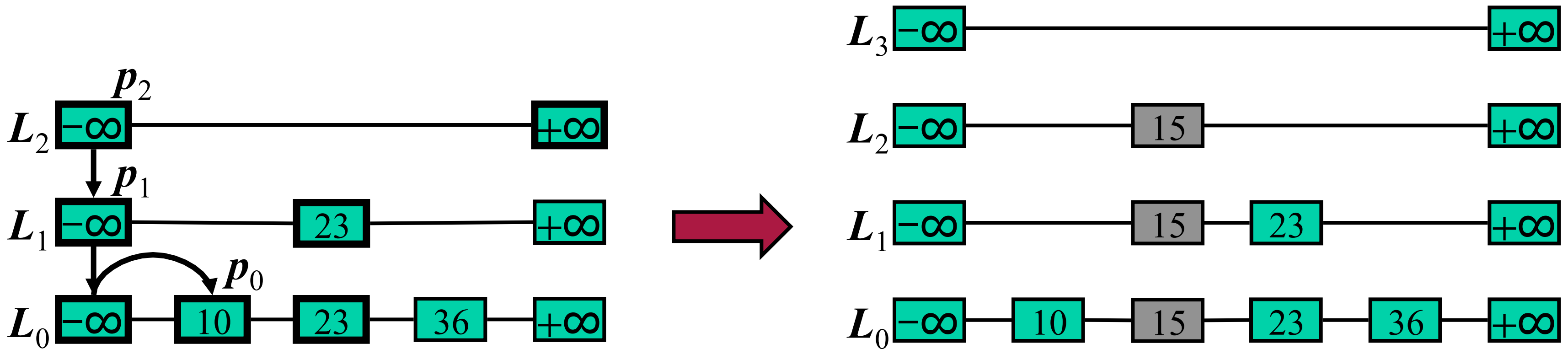
At L1 P is 64, the next element +∞ is bigger than 78, we drop down
At L0, 78 = 78, so the search is over.

Insertion

- The insert algorithm uses randomization to decide in how many levels the new item <k> should be added to the skip list.
- After inserting the new item at the bottom level flip a coin.
- If it returns tail, insertion is complete. Otherwise, move to next higher level and insert <k> in this level at the appropriate position, and repeat the coin flip.

Insertion Example

1) Suppose we want to insert 15
2) Do a search, and find the spot between 10 and 23
3) Suppose the coin come up "head" three times

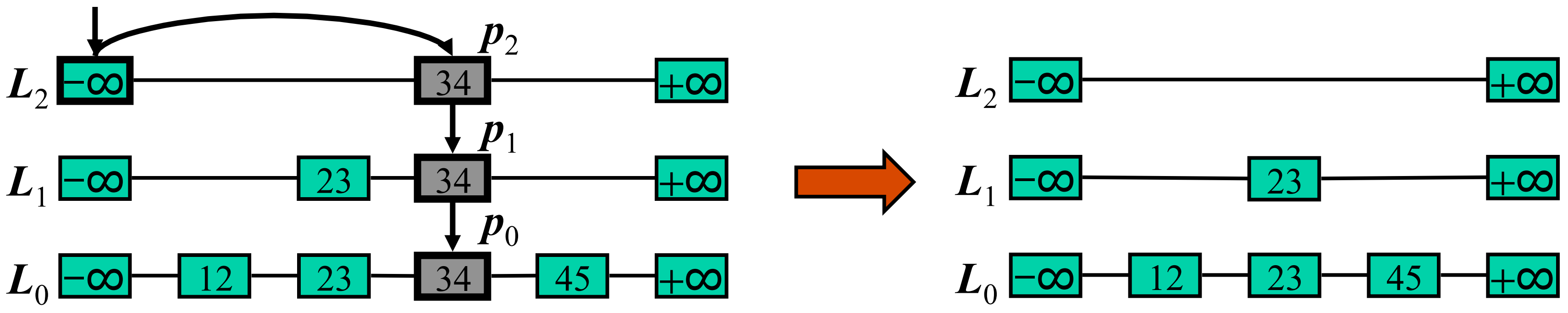


Deletion

- Search for the given key <k>. If a position with key <k> is not found, then no such key exists.
- Otherwise, if a position with key <k> is found (it will be definitely found on the bottom level), then we remove all occurrences of <k> from every level.
- If the uppermost level is empty, remove it.

Deletion Example

1) Suppose we want to delete 34
2) Do a search, find the spot between 23 and 45
3) Remove all occurrences of the key



O(1) pointers per node

Average number of pointers per node = O(1)

Total number of pointers

$$= 2.n + 2.\ n/2 + 2.\ n/4 + 2.\ n/8 + \ldots$$
$$= 4.n$$

So, the average number of pointers per node = 4

Number of levels

The number of levels = O(log n) w.h.p

Pr[a given element x is above level c log n] $= 1/2^{c \log n}$
$= 1/n^c$

Pr[any element is above level c log n] $= n.\ 1/n^c$
$= 1/n^{c-1}$

So the number of levels = O(log n) with high probability

Search time

Consider a skiplist with two levels L_0 and L_1. To search a key, first search L_1 and then search L_0.

Cost (i.e. search time) = length (L_1) + n / length (L_1)
Minimum when length (L_1) = n / length (L_1).

Thus length(L_1) = $(n)^{1/2}$, and cost = 2. $(n)^{1/2}$
(Three lists) minimum cost = 3. $(n)^{1/3}$
(Log n lists) minimum cost = log n. $(n)^{1/\log n}$ = 2.log n

Skip lists for P2P?

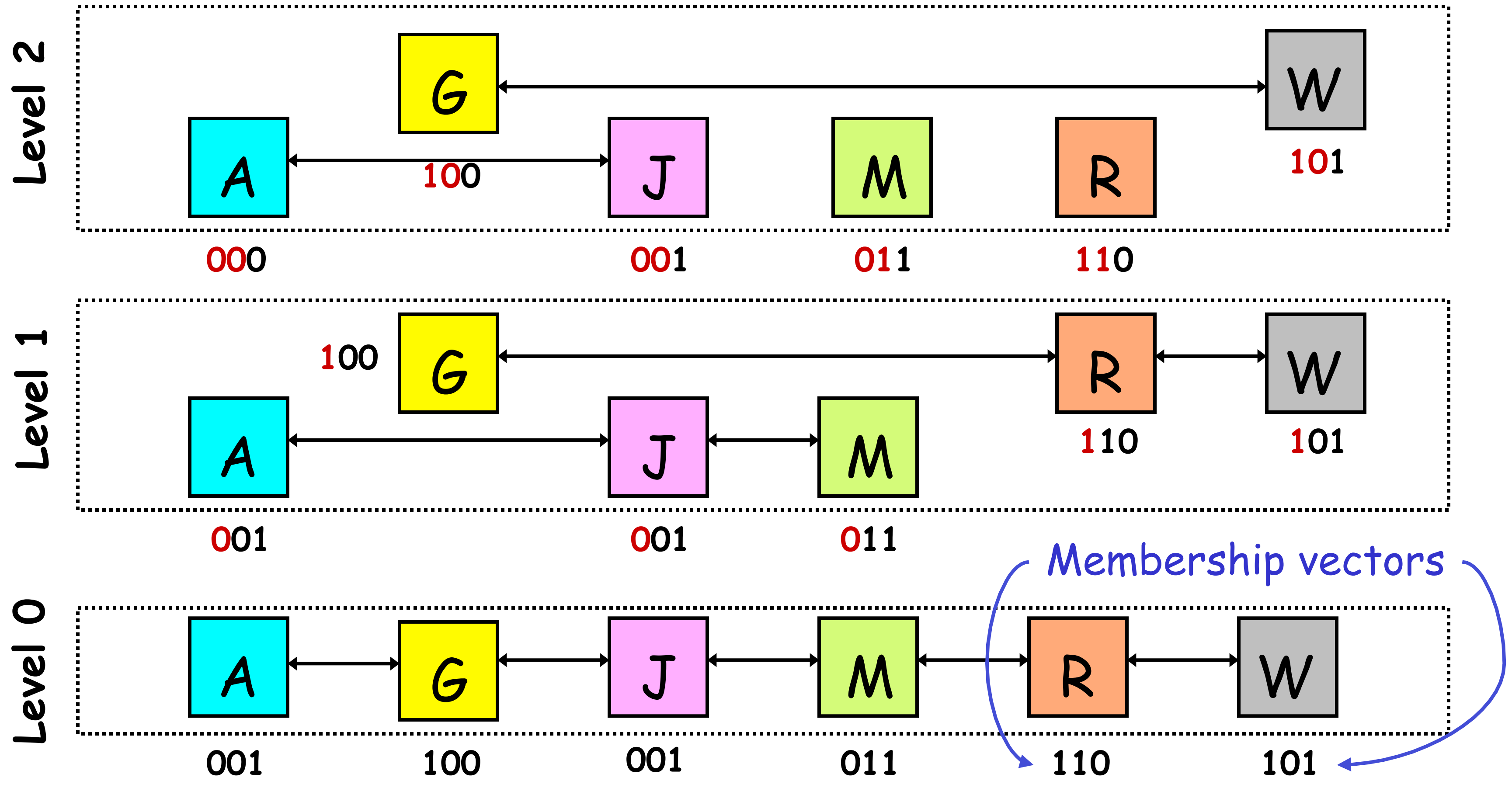
Advantages

- O(log n) expected search time.
- ***Retains locality.***
- Dynamic node additions/deletions.

Disadvantages

- Heavily loaded top-level nodes.
- Easily susceptible to failures.
- Lacks redundancy.

A Skip Graph



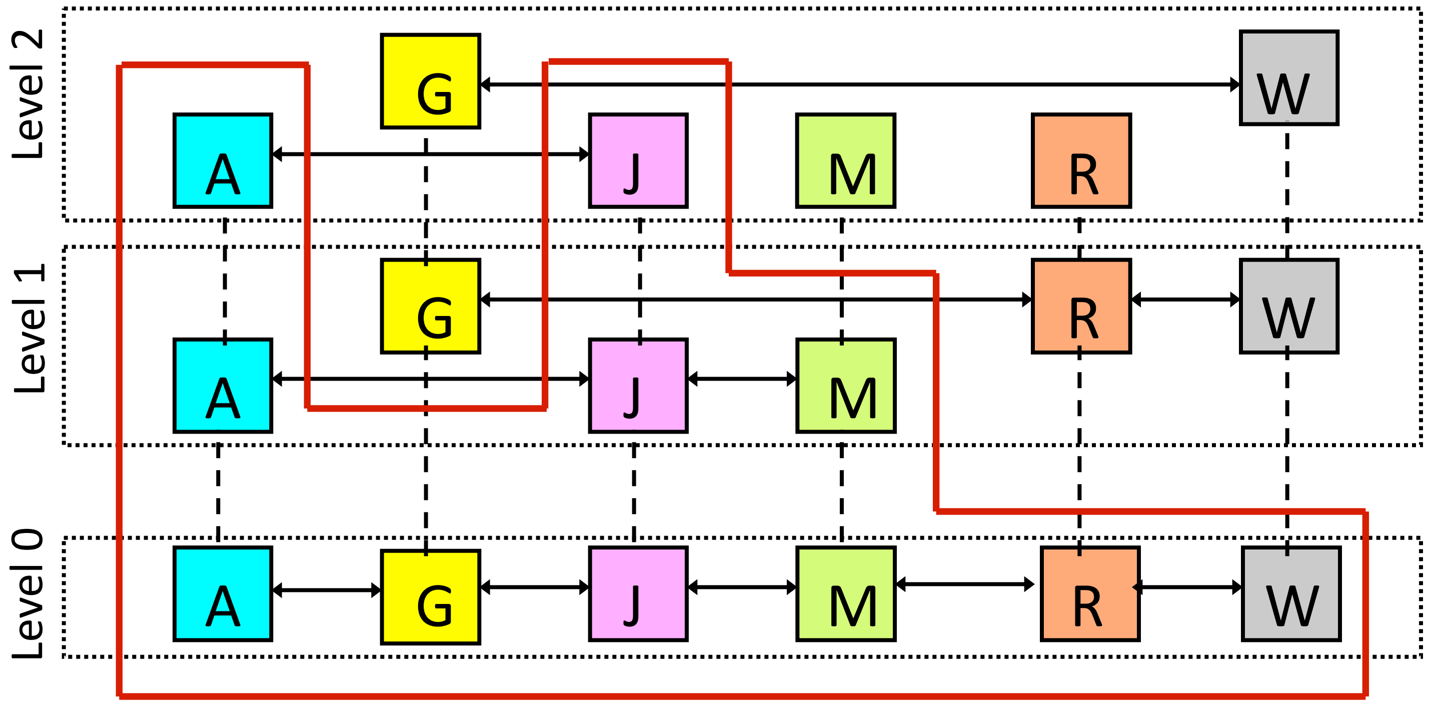
Link at level i to nodes with matching prefix of length i.
Think of a tree of skip lists that share lower layers.

Properties of skip graphs

1. Efficient Searching.
2. Efficient node insertions & deletions.
3. Independence from system size.
4. Locality and range queries.

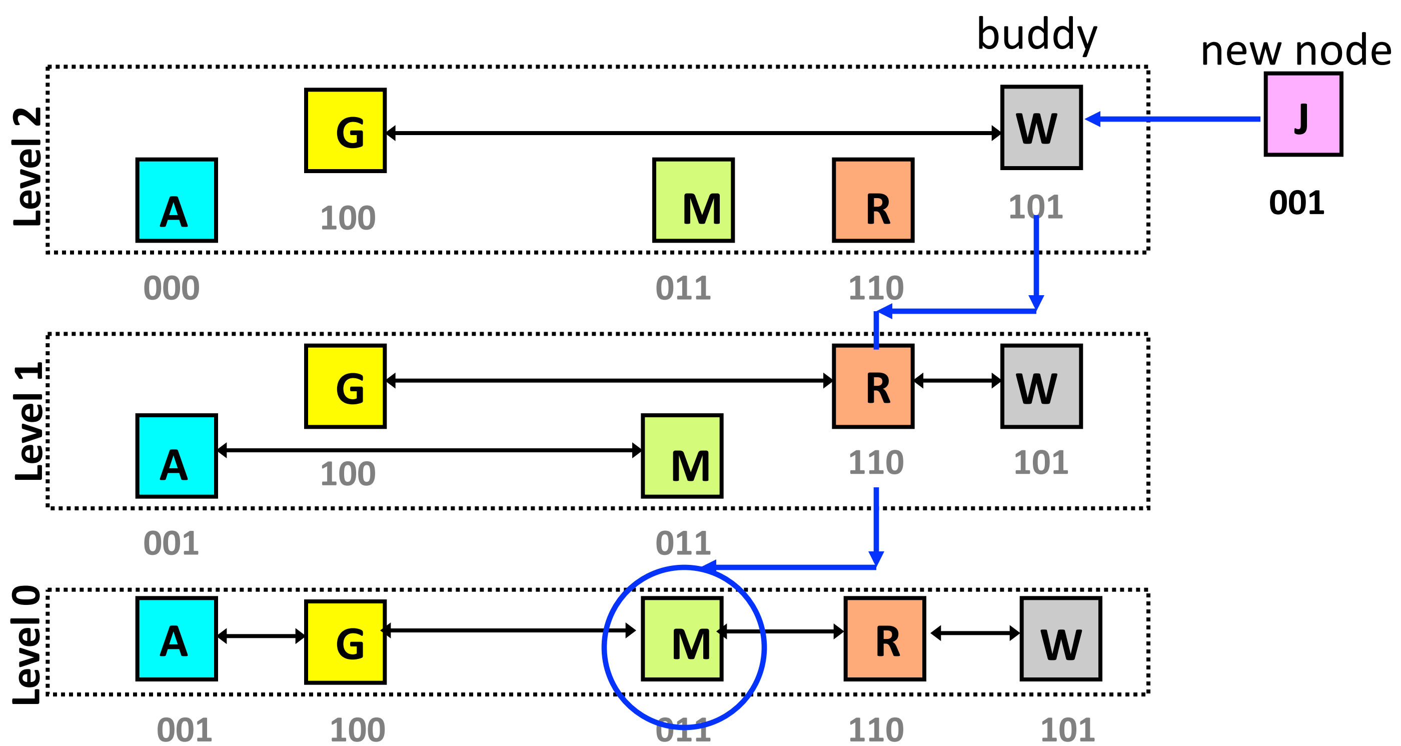
Searching: avg. O (log n)

Restricting to the lists containing the starting element of the search, we get a skip list.



Same performance as DHTs.

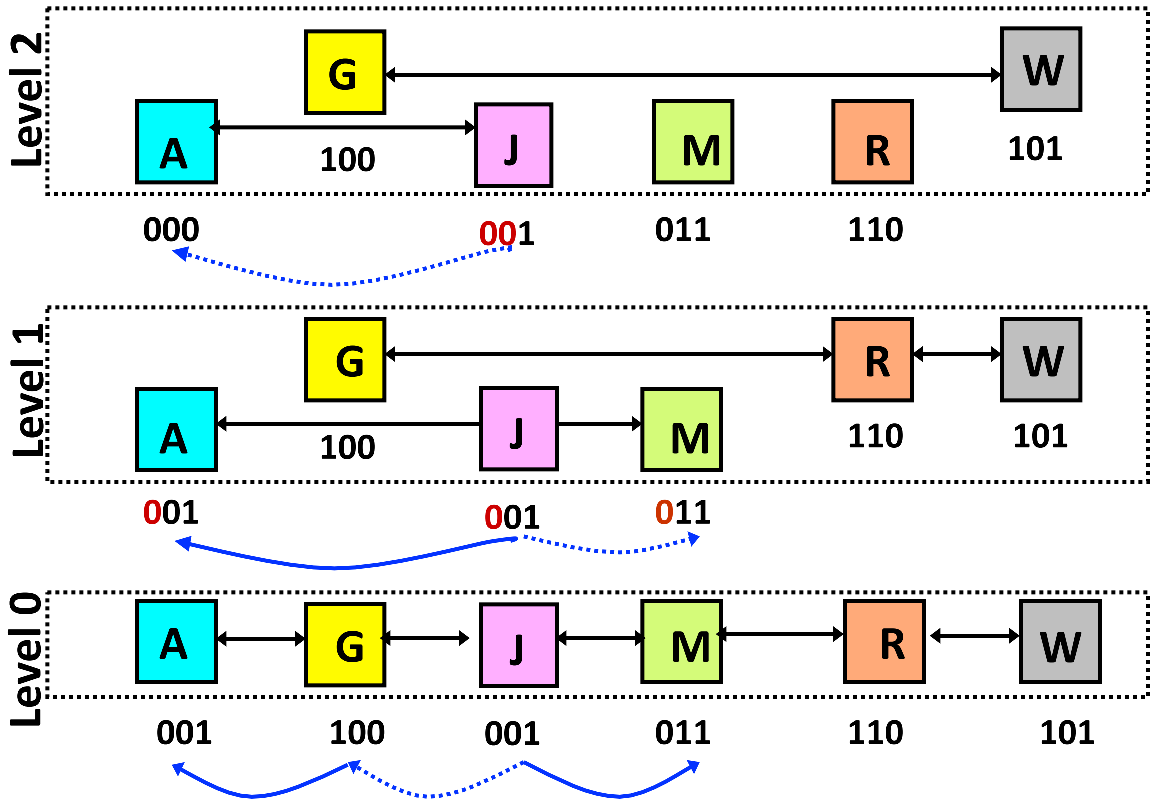
Node Insertion – 1



Starting at buddy node, find nearest key at level 0.
Takes O(log n) time on average.

Node Insertion - 2

At each level i, find nearest node with matching prefix of membership vector of length i.

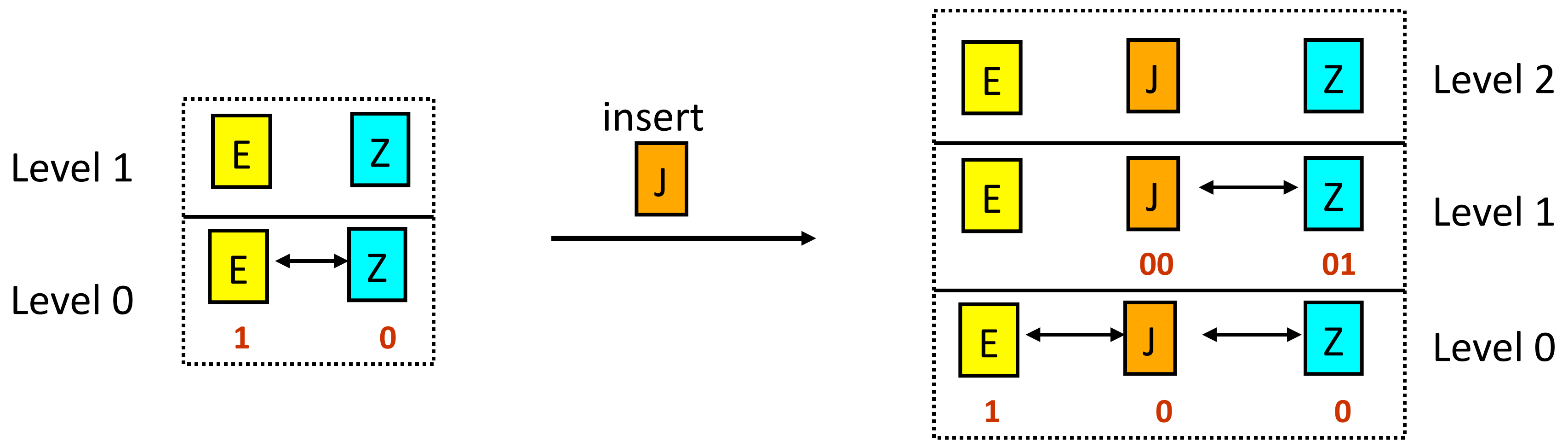


Total time for insertion: O(log n)

DHTs take: O($\log^2$n)

Independent of system size

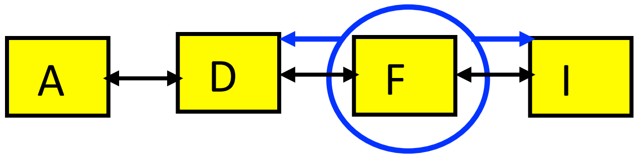
No need to know size of keyspace or number of nodes.



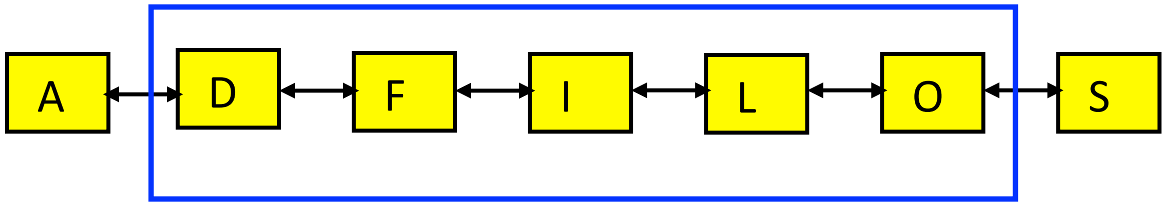
Old nodes extend membership vector as required with arrivals.
DHTs require knowledge of keyspace size initially.

Locality and range queries

- Find key < F, > F.
- Find largest key < x.
- Find least key > x.



- Find all keys in interval [D..O].



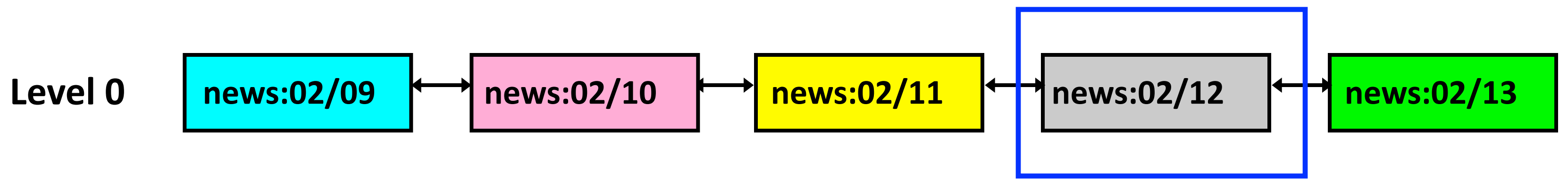
- Initial node insertion at level 0.

Applications of locality

Version Control

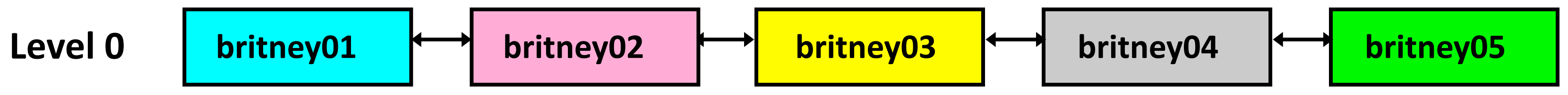
e.g. find latest news from yesterday.

$\longrightarrow$ find largest key < news: 02/13.



Data Replication

e.g. find any copy of some Britney Spears song.



DHTs cannot do this easily as hashing destroys locality.

So far…

✓Decentralization.

✓Locality properties.

✓O(log n) space per node.

✓O(log n) search, insert, and delete time.

✓Independent of system size.

Coming up…

- Load balancing.
- Tolerance to faults.
  - Random faults.
  - Adversarial faults.
- Self-stabilization.

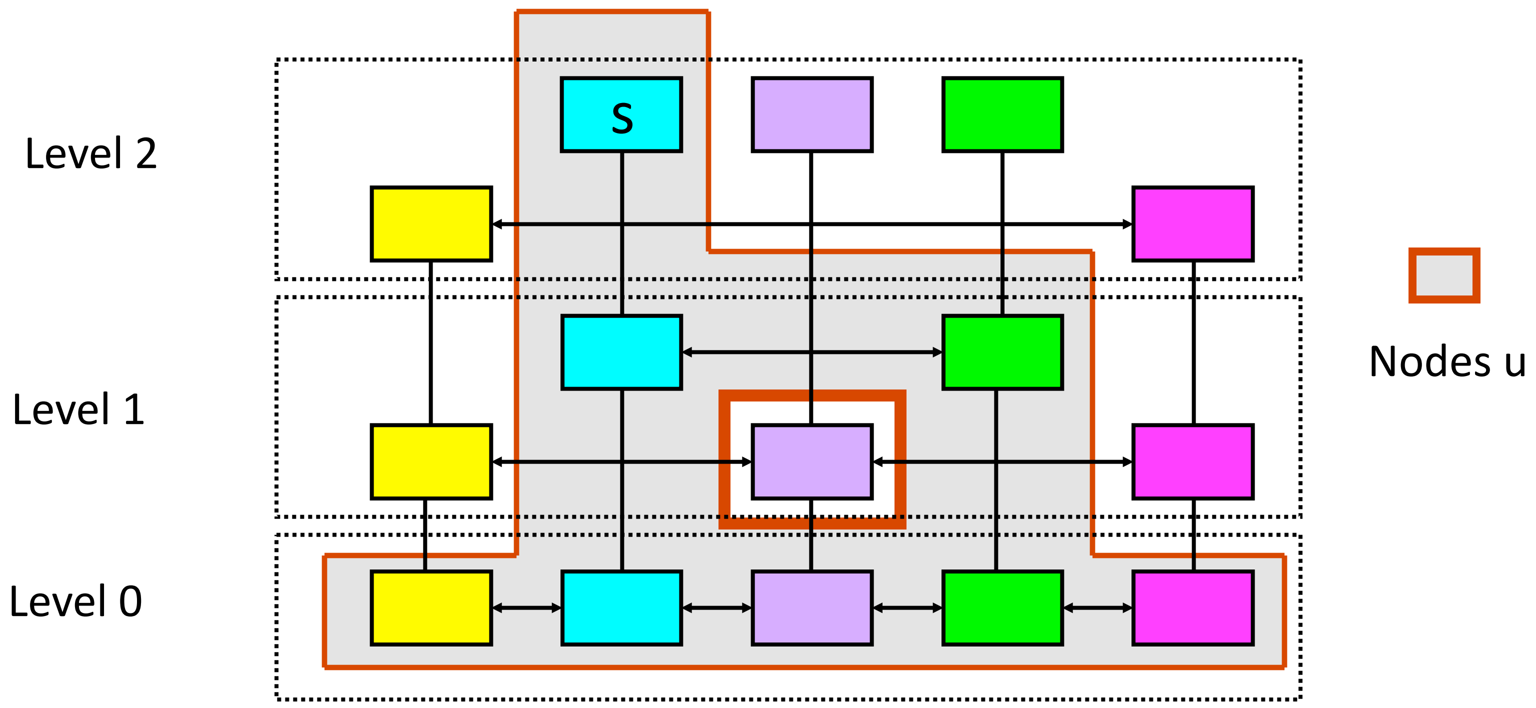
Load balancing

Interested in average load on a node u.
i.e. the number of searches from source
s to destination **t** that use node **u**.

Theorem: Let dist (u, t) = d. Then the probability that a search from s to t passes through u is < 2/(d+1).

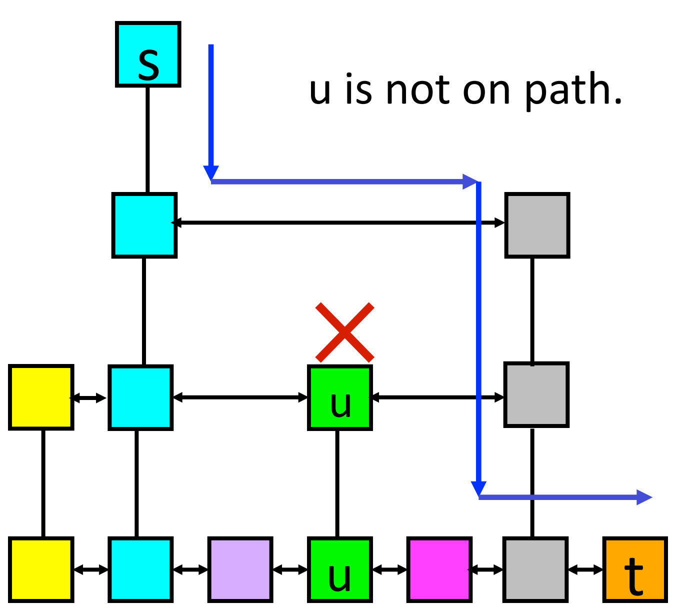
where V = {nodes v: u <= v <= t} and |V| = d+1.

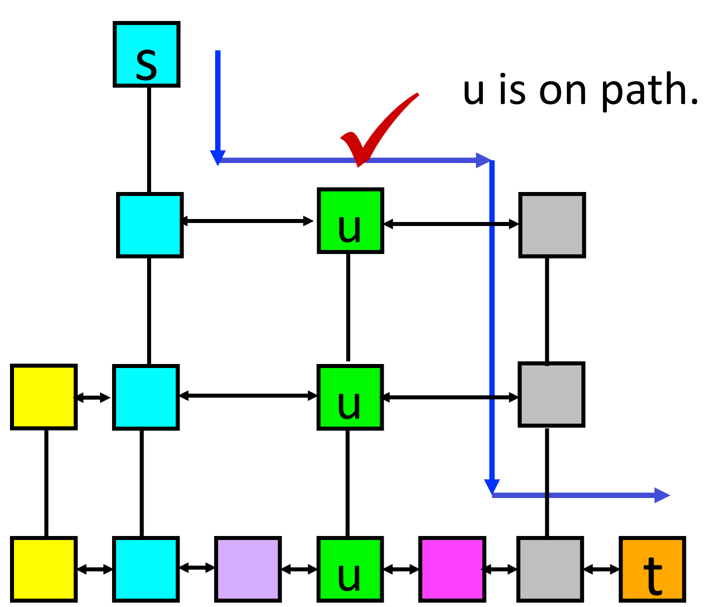
Observation



Node u is on the search path from s to t only if it is in the skip list formed from the lists of s at each level.

Tallest nodes





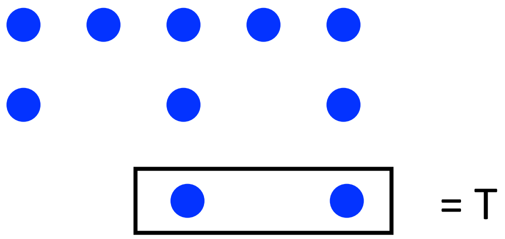
Node u is on the search path from s to t only if it is in T = the set of k tallest nodes in [u..t].

$$\Pr[u \in T] = \sum_{k=1}^{d+1} \Pr[|T|=k] \bullet k/(d+1) = E[|T|]\boxed{/(d+1)}.$$

Heights independent of position, so distances are symmetric.

Load on node u

Start with n nodes. Each node goes to next set with prob. 1/2.
We want expected size of T = last non-empty set.

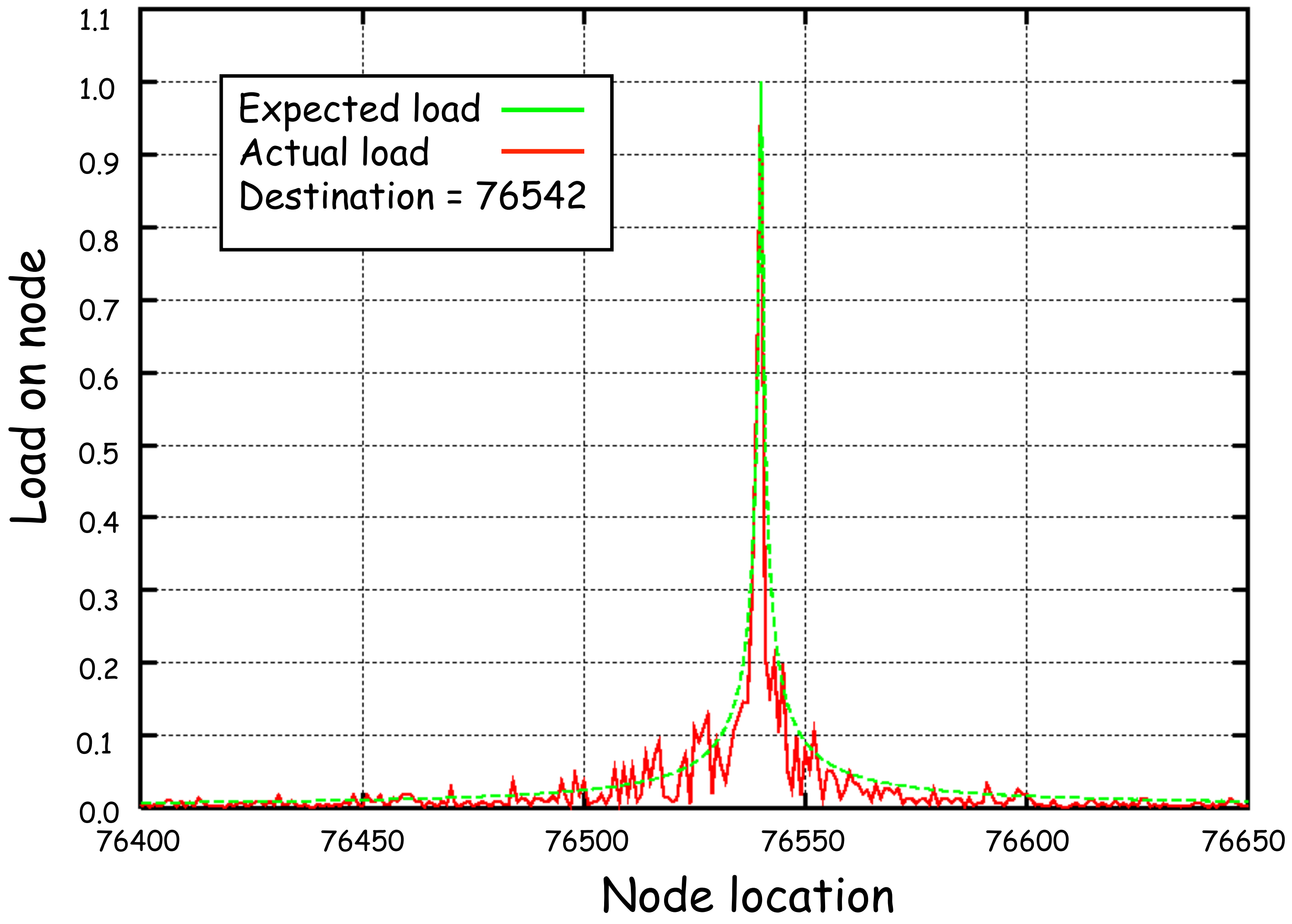


We show that: E[|T|] < 2.

Asymptotically: E[|T|] = 1/(ln 2) ± 2x10^{-5} ≃ 1.4427… [Trie analysis]

Average load on a node is inversely proportional to the distance from the destination.

Experimental result



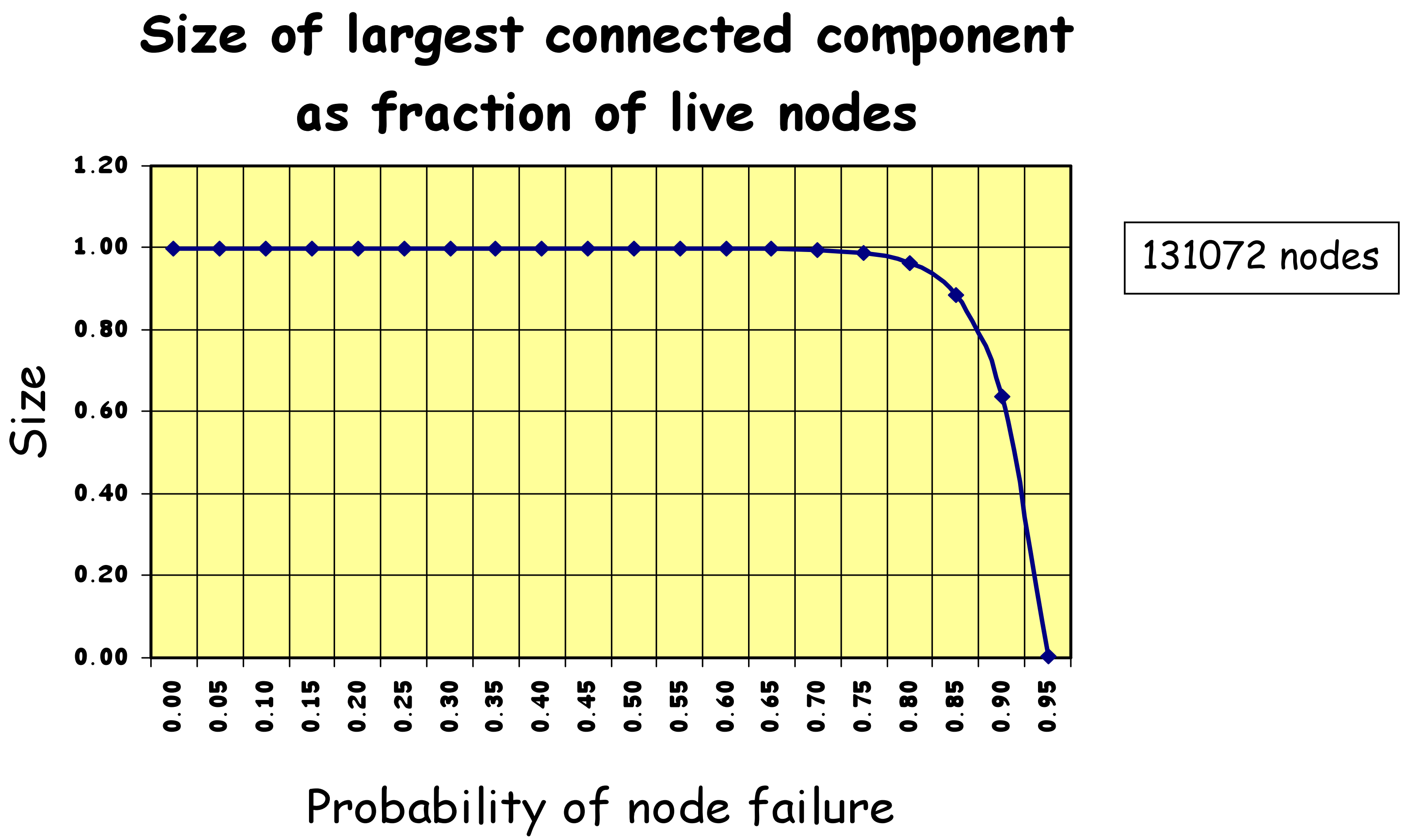
Fault tolerance

How do node failures affect skip graph performance?

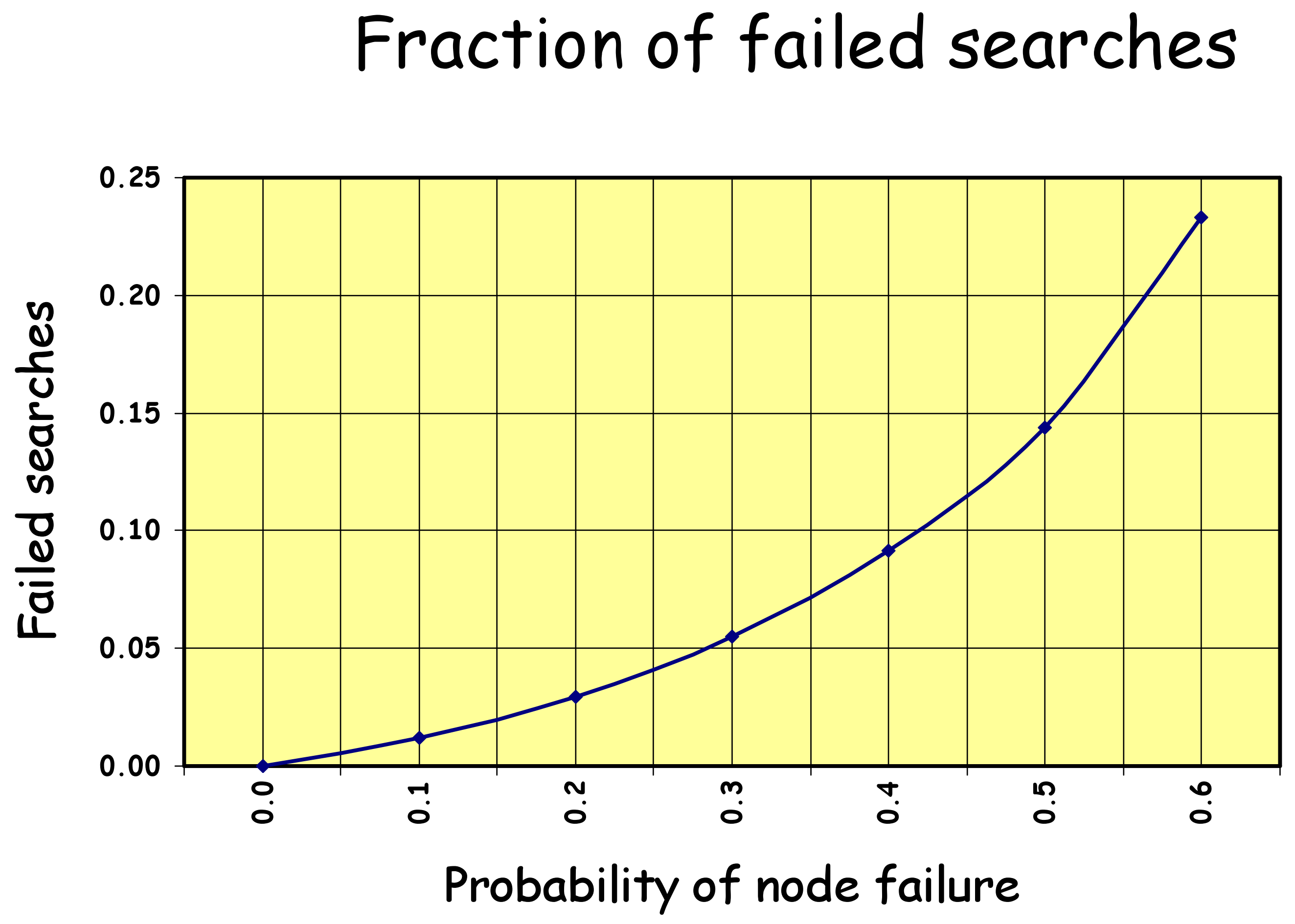
Random failures: Randomly chosen nodes fail.
Experimental results.

Adversarial failures: Adversary carefully chooses nodes that fail.
Bound on expansion ratio.

Random faults

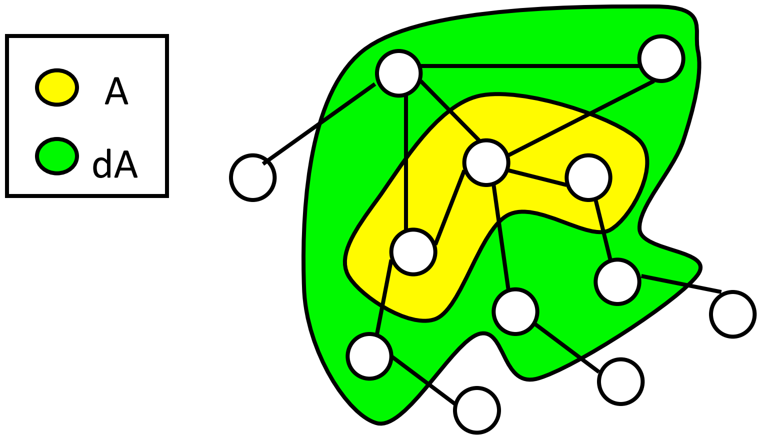


Searches with random failures



131072 nodes
10000 messages

Adversarial faults



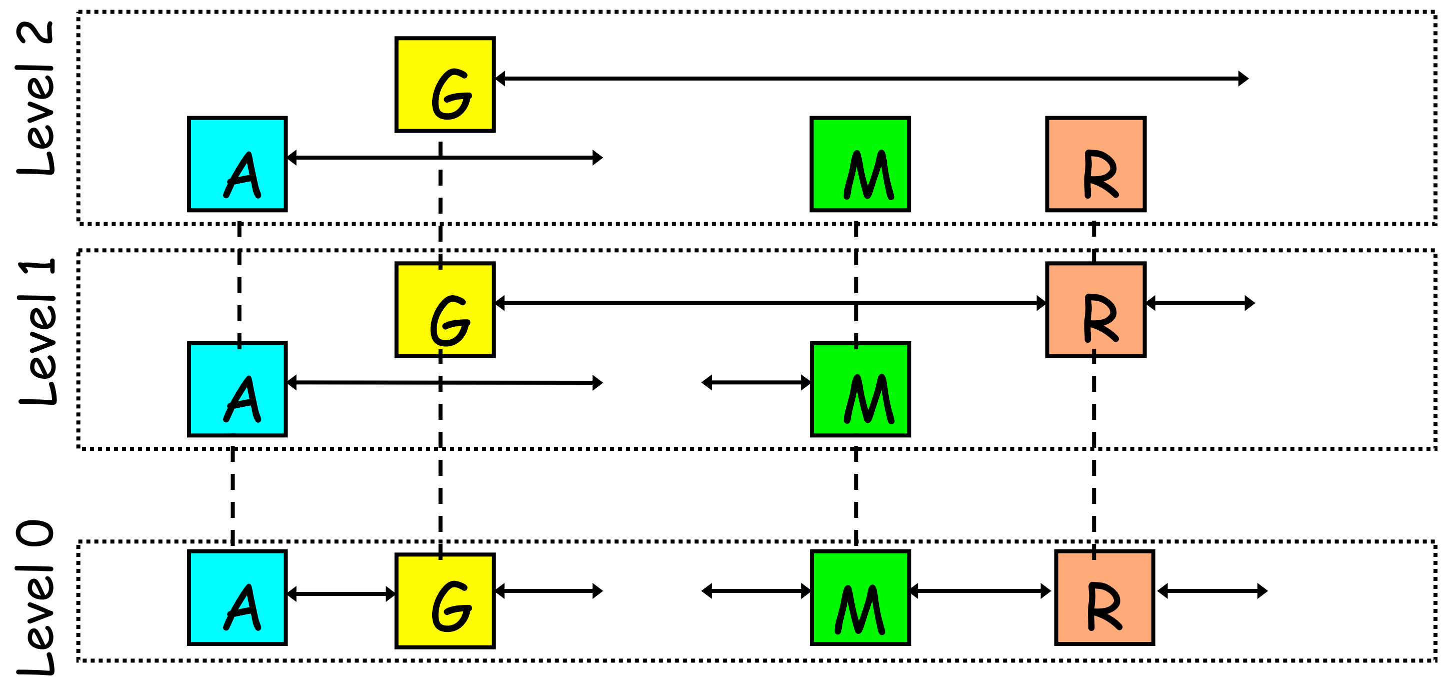
dA = nodes adjacent to A but not in A. To disconnect A all nodes in dA must be removed

Expansion ratio = min |dA|/|A|, 1 <= |A| <= n/2.

Theorem: A skip graph with n nodes has expansion ratio $=\Omega(1/\log n)$.

⇨ f failures can isolate only O(f•log n) nodes.

Need for repair mechanism



Node failures can leave skip graph in inconsistent state.

Ideal skip graph

Let xRi (xLi) be the right (left) neighbor of x at level i.

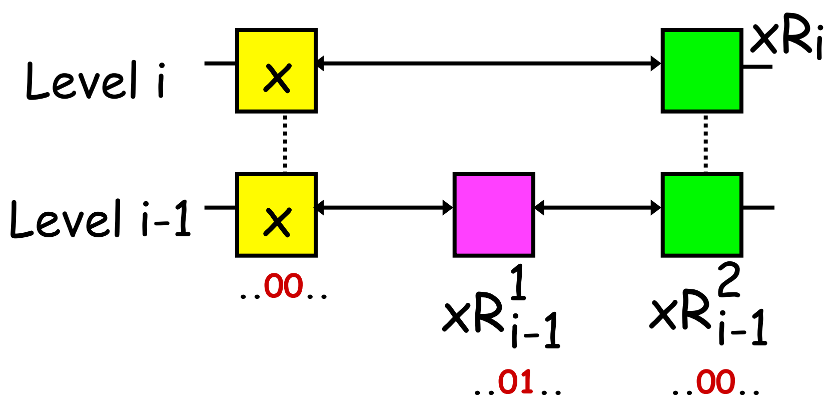
If xLi, xRi exist:

$$\boxed{\begin{array}{l} xL_i < x < xR_i. \\ xL_iR_i = xR_iL_i = x. \end{array}}$$

Invariant

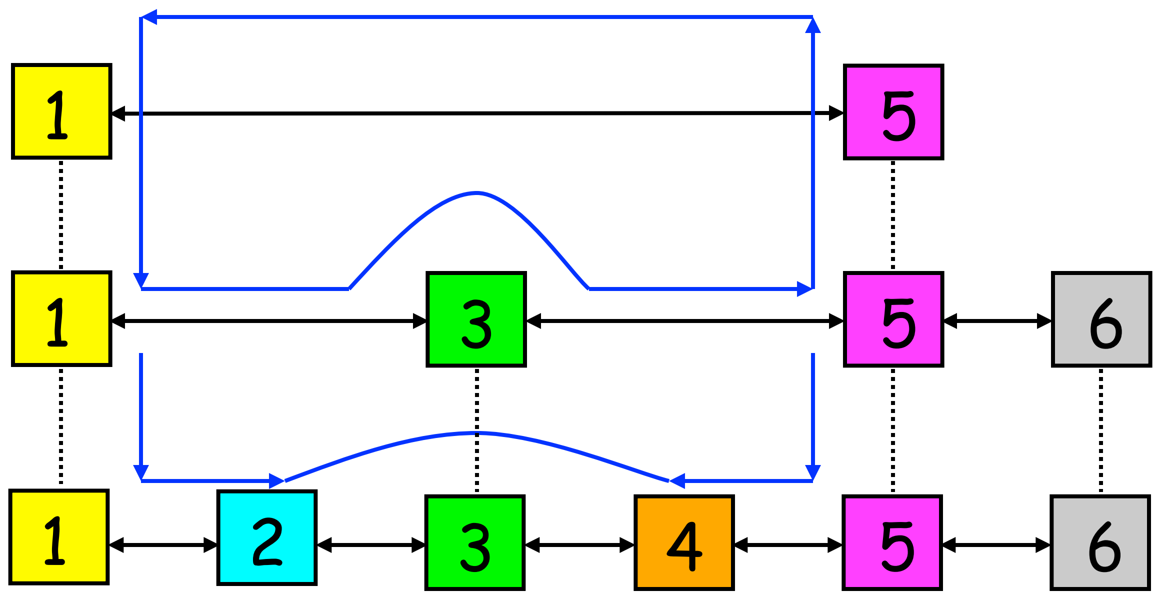
$$\boxed{\begin{array}{l} xL_i = xL^k_{i-1}. \\ xR_i = xR^k_{i-1}. \end{array}}$$

Successor constraints



Basic repair

If a node detects a missing neighbor, it tries to patch the link using other levels.

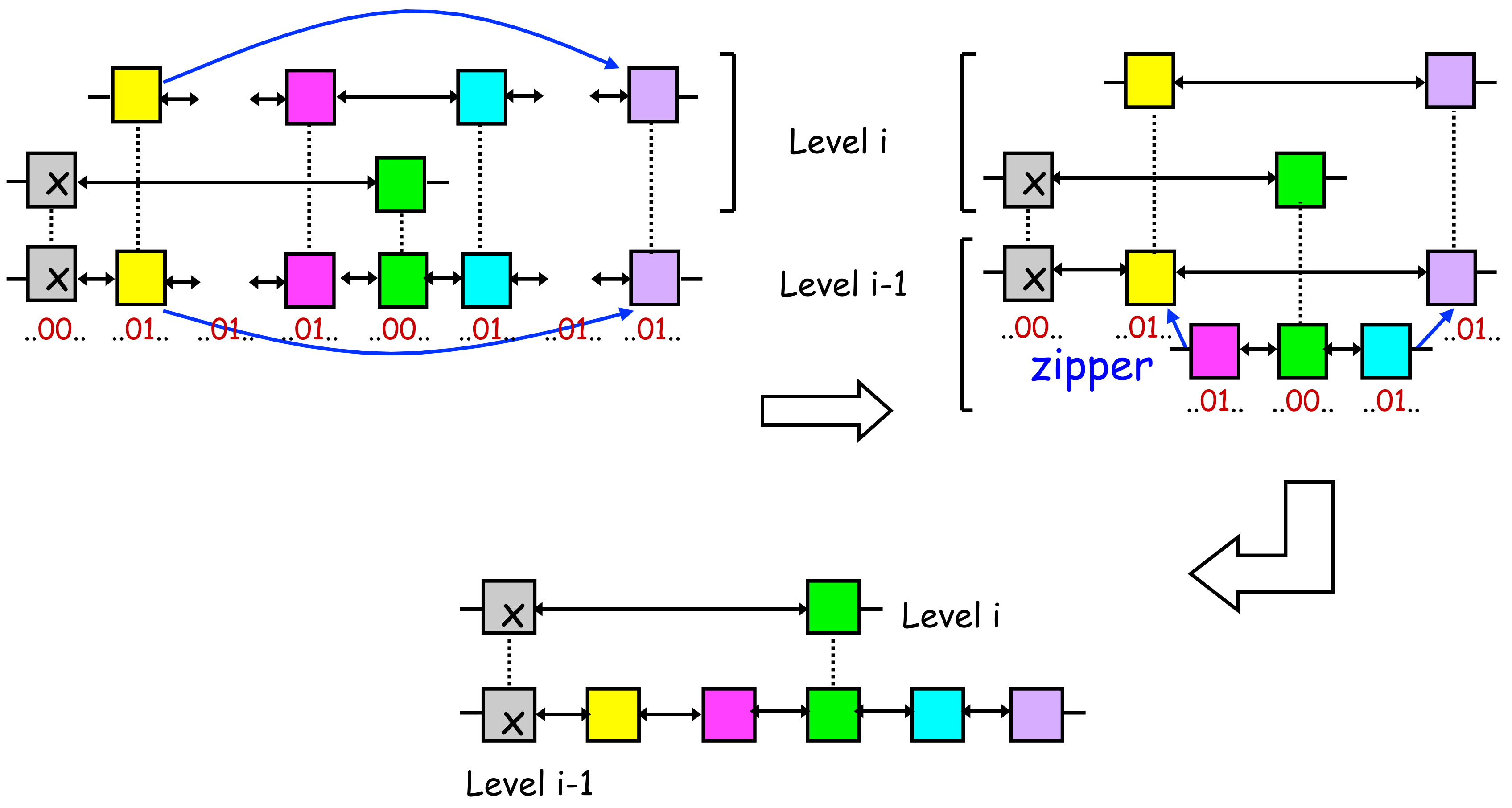


Also relink at other lower levels.

Successor constraints may be violated by node arrivals or failures.

Constraint violation

Neighbor at level i not present at level (i-1).



Conclusions

Similarities with DHTs

- Decentralization.
- O(log n) space at each node.
- O(log n) search time.
- Load balancing properties.
- Tolerant of random faults.

Differences

Property	DHTs	Skip Graphs
Insert/Delete time	$O(\log^2 n)$	$O(\log n)$
Locality	No	Yes
Repair mechanism	?	Partial
Tolerance of adversarial faults	?	Yes
Keyspace size	Reqd.	Not reqd.

Open Problems

- Design efficient repair mechanism.
- Incorporate geographical proximity.
- Study multi-dimensional skip graphs.
- Evaluate performance in practice.
- Study effect of byzantine failures.