Simultaneous Insertions in Tapestry

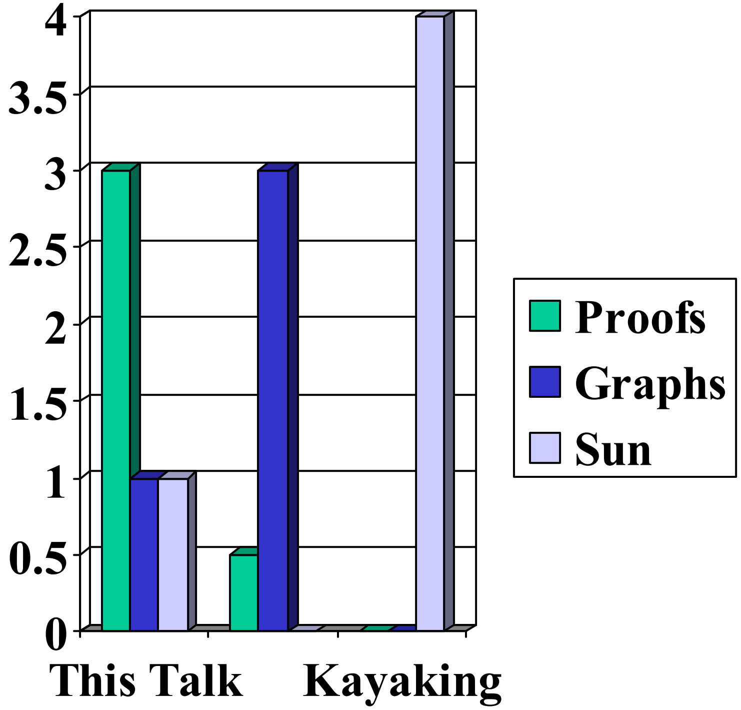
Kris Hildrum, UC Berkeley

hildrum@cs.berkeley.edu

Joint work with John Kubiatowicz, Satish Rao, and Ben Y. Zhao

This is going to be different…

- Please stop me if I'm confusing.
- This will be your only graph.
- Now for the hard (but very cool) stuff…



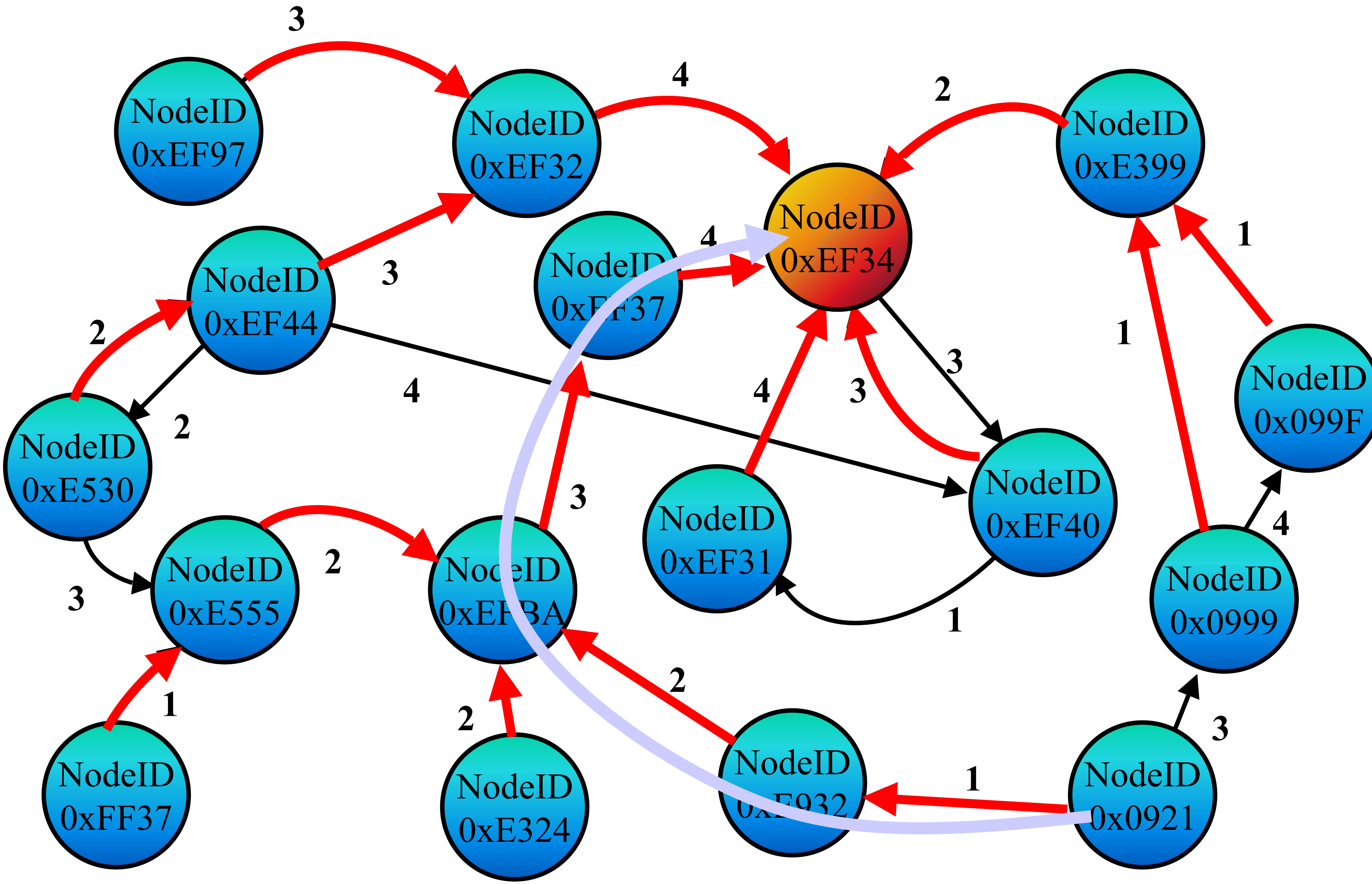
Related Work

(no, this wasn't in the original talk)

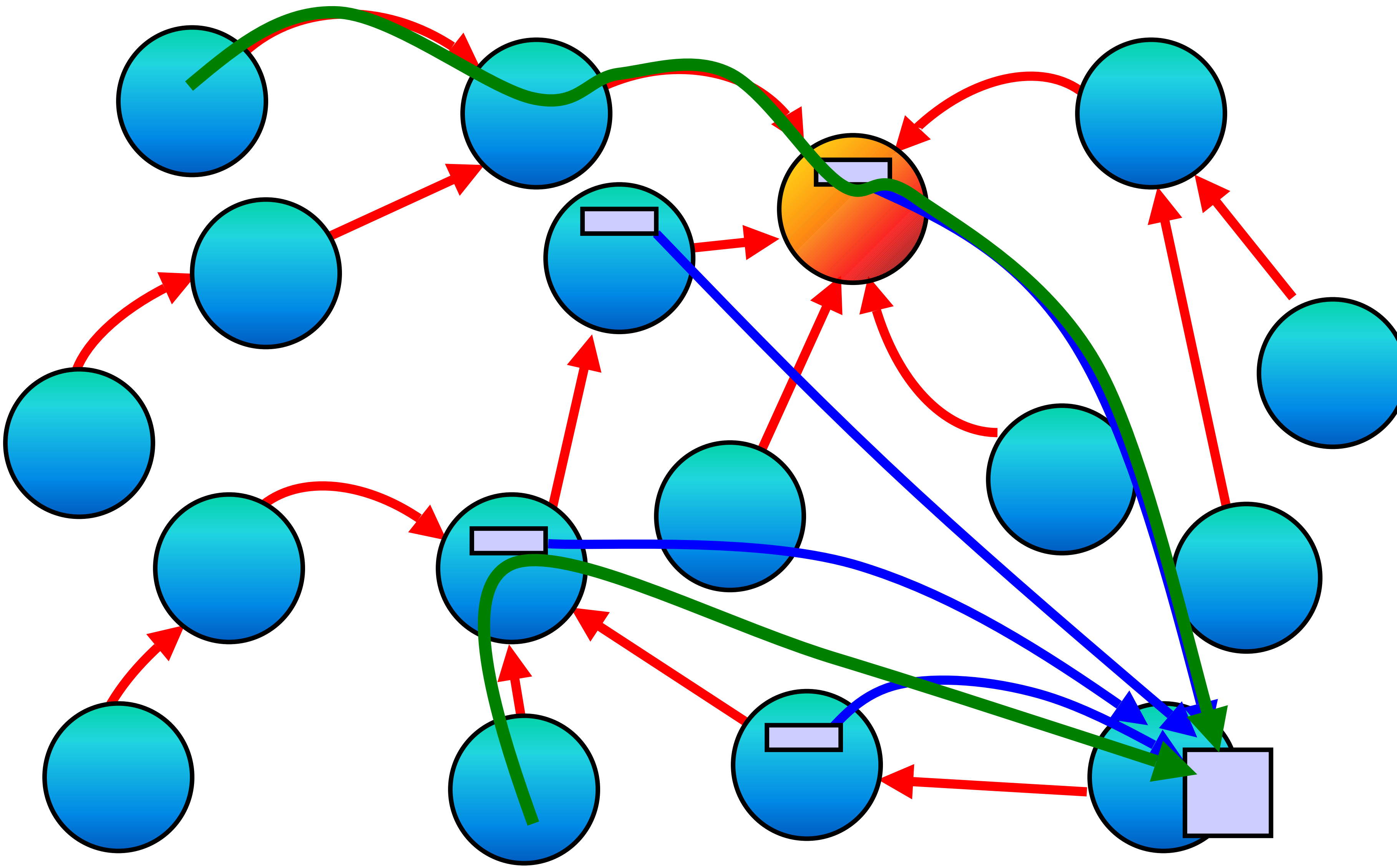
- Tapestry mesh inspired by paper by Plaxton, Rajaraman and Richa from SPAA 1997.
- Other peer-to-peer object location systems include
  - Chord
  - CAN
  - Pastry

Basic Tapestry Mesh

(from PRR97)



Use of Tapestry Mesh Randomization and Locality



Why simultaneous?

- Inserts will not always happen one at a time.
  - Not practical to have one gateway to serialize
- Most simultaneous inserts completely harmless (no interference), but handling bad ones correctly is important
- Assumptions:
  - No concurrent deletes (can be worked around)
  - Messages always arrive, though no guarantee on timely delivery

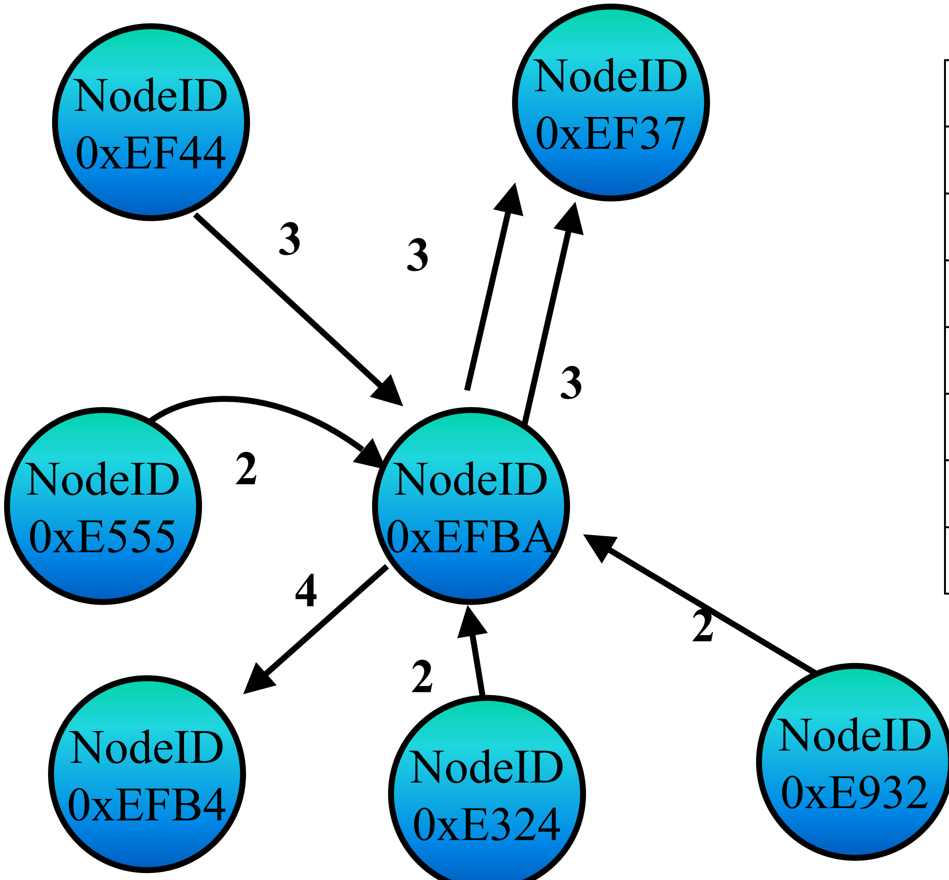
(Simultaneous) Insertion

- Find node with closest matching ID (surrogate) and get preliminary neighbor table
  - If surrogate's is hole-free, so is this one.
- Find all nodes that need to put new node in routing table via multicast
- Optimize neighbor table
  - Very tricky & fun, touched on here.

- Want:

No fillable holes.

Neighbor Table



Neighbor Map
For "2175" (Octal)

2170	210x	20xx	0xxx
2171	211x	2175	1xxx
2172	212x	22xx	2175
2173	213x	23xx	3xxx
ø	214x	24xx	4xxx
2175	215x	25xx	5xxx
2176	216x	26xx	6xxx
2177	2175	27xx	7xxx
4	3	2	1

Routing Levels

Need-to-know nodes

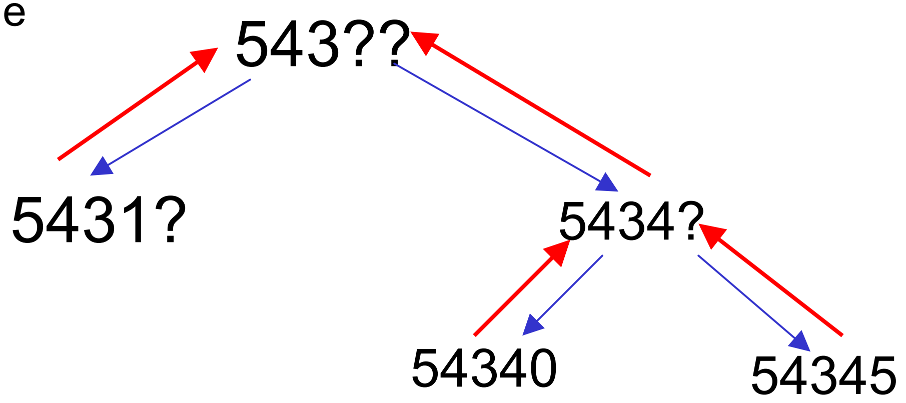
- Need-to-know = a node with a hole in neighbor table filled by new node
  - If 1234 is new node, and no 123s existed, must notify 12?? nodes
  - Acknowledged multicast to all matching nodes
- During this time, object requests may go either to new node or former surrogate, but old and new can forward requests
  - New node knows old destination
  - Once pointers moved, pre-insertion destination knows new node.

Acknowledged Multicast Algorithm

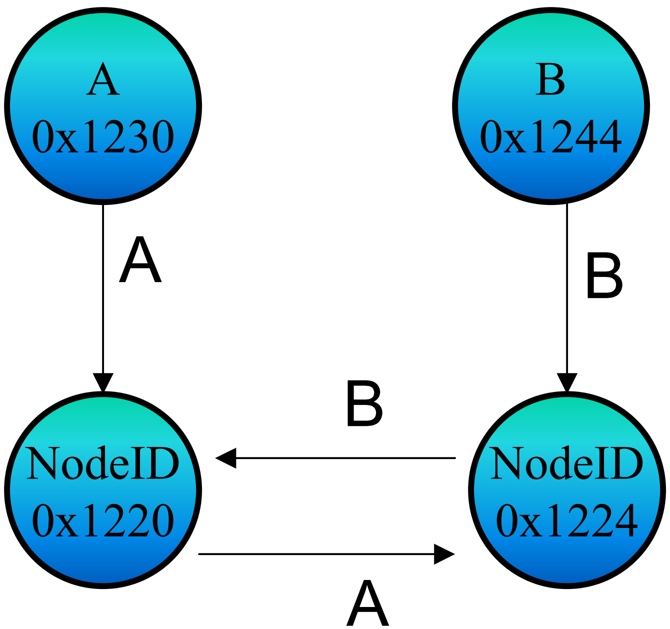
Locates & Contacts all nodes with a given prefix

- Create a tree based on IDs as we go
- Starting node knows when all nodes reached
- Nodes send acks when all children reached

The node then sends to **any** ?0345, **any** ?1345, **any** ?3345, etc. if possible



Multicast Breaks



- A is only 123
- B is only 124
- They need to find out about each other
- But they don't!

What Goes Wrong?

- Suppose A & B add themselves.
  - A is only 123
  - B is only 124
  - Both talk to same set (all 12 nodes)
  - 123 is a “Need-to-Know” node for 124 & vice-versa
  - But multicasts could pass each other…

But it Gets Worse…

- Suppose X has prefix 12.
- A=1231 arrives. X adds A to table.
- B =1232 arrives.
  - X adds B to table, drops A.
  - Sends B’s message to A.
- C = 1233 arrives.
  - X sends C’s message to B.
- B gets C’s message.
- A gets message about B’s.

A does not know about C!!

We Fill All Holes - Outline

- Multicast reaches all completely inserted or *core* nodes. (Lemma 1)
- Any same-hole insertion arriving at a node before A is found before A finishes its multicast. So A has found all such nodes by end. (Lemma 2)
- Any two different-hole insertions must find each other.

Locking Pointers

- Problem in same hole case:
  - multicast assumed that chosen node can forward message
  - Inserting nodes have incomplete information.

So…

- Pointers are added as "locked". When multicast for that node returns, pointers are unlocked.
- Multicasts are sent to one unlocked pointer and all locked pointers.
- Locked pointers may not be deleted.

Any unlocked pointer can reach all other unlocked pointers.

Suppose it is true for all unlocked pointers until A. Now consider next unlocked pointer.

- Knows all unlocked before its arrival, by hypothesis.
- Knows locked when A arrived, since A's message was sent to them.
- Knows later arrivals, since they must have sent message down A.

$\Rightarrow$ *If X sends to one unlocked and all locked, all nodes X has seen will get message.*

Modified Multicast

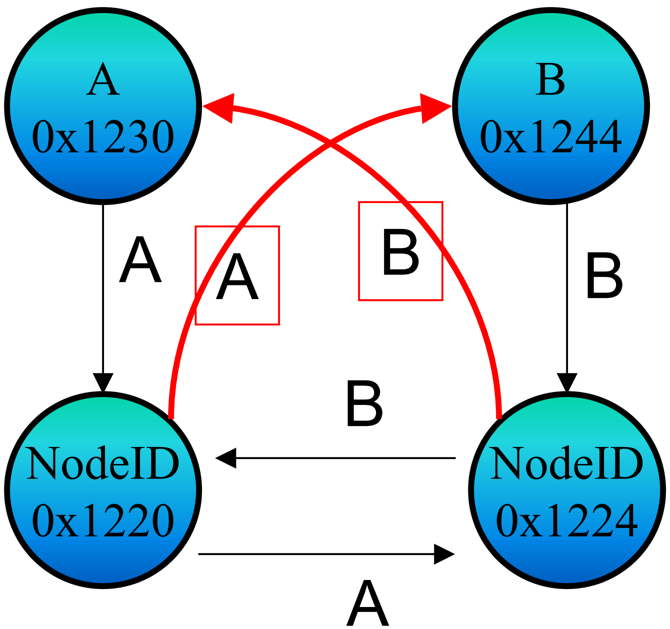
- Message now includes:
  - Hole node is filling
  - A “watch list” of unfilled holes in neighbor table
- Receivers now
  - Forward multicast to hole if hole filled
  - Send any nodes matching holes in watch list to originator

•We want:

When A finishes its multicast, it has informed all core need-to-know nodes and it knows all the core nodes it needs to. (no unfilled holes)

Two insertions *conflict* if there can be no agreement on which the order in which the insertions occurred.

New Multicast Fixes Problem



- A is only 123
- B is only 124
- They need to find out about each other
- A needs to arrive before B at only ONE node.

Proof

- Multicast reaches all completely inserted nodes. (Lemma 1)
- Any same-hole insertion arriving at a node before A is found before A finishes its multicast. So A has found all such nodes by end. (Follows from pointer locking)
- Any different-hole insertion must either arrive
  – Before or conflict (ok)
  – After (then A gets multicast)

Lemma 1: Core Nodes Reached

- Core node: multicast finished.
- Suppose some core node unreached. Consider X, which was supposed to send it towards core node.
  - X is not finished inserting. Cannot be, since X only fills holes.
  - X is done inserting. But it must not have a hole.

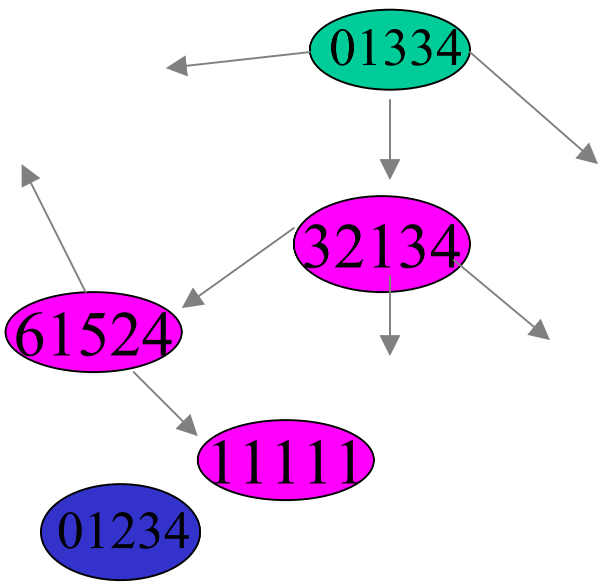
Finding Nearest Neighbor

- Let j be such that surrogate matches new node in last j digits of node ID
- G = surrogate

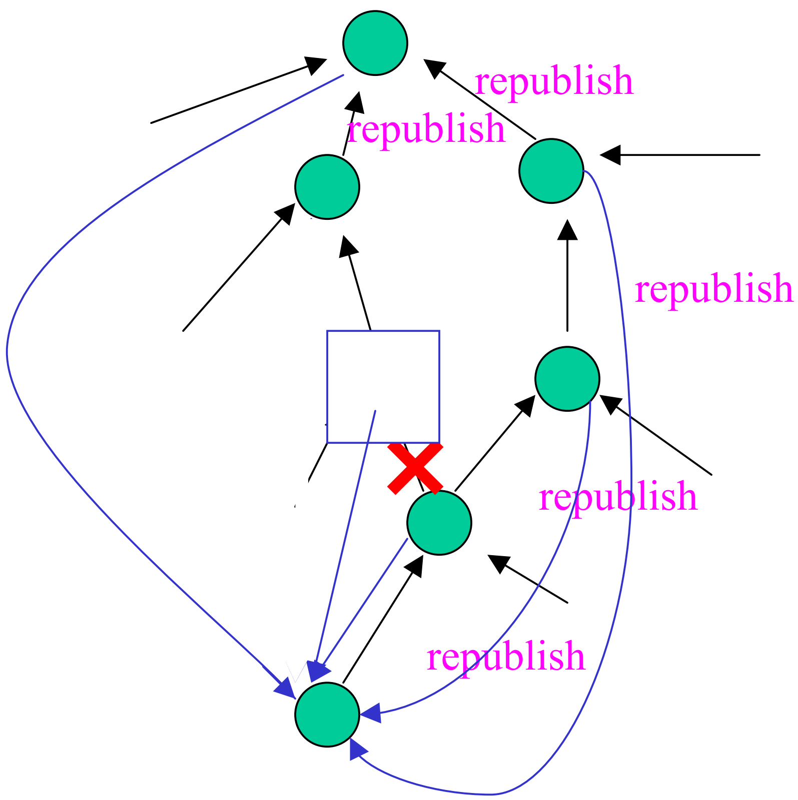
A. G sends j-list to new node; new node pings all nodes on j-list.

B. If one is closer, G = closest, goto A. If not, done with this level, and let j = j-1 and goto A.

j-list is closest k=O(log n) nodes matching in j digits



Delete



Conclusions

- Simultaneous insertion works.
- Deletion and details on insertion in paper.
- Questions:
  - How does delete interact with insert?
  - Can we make optimization algorithm better?