Distributed Hash Tables:
An Overview

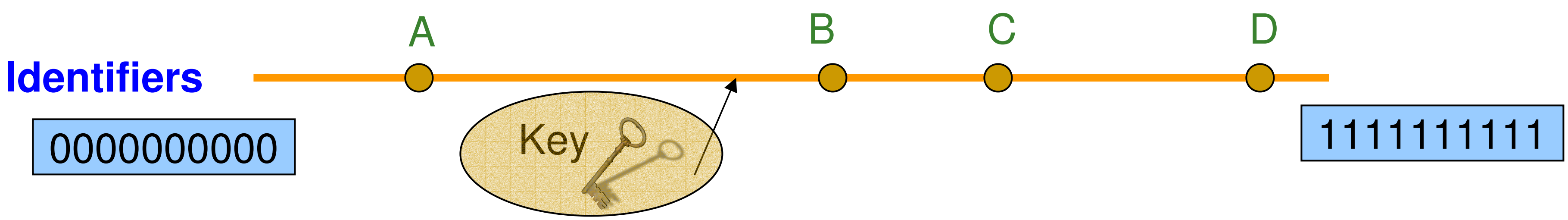
Ashwin Bharambe
Carnegie Mellon University

Definition of a DHT

- Hash table ➔ supports two operations
  - `insert(key, value)`
  - `value = lookup(key)`
- Distributed
  - Map hash-buckets to nodes
- Requirements
  - Uniform distribution of buckets
  - Cost of `insert` and `lookup` should *scale* well
  - Amount of local state (routing table size) should *scale* well

Fundamental Design Idea - I

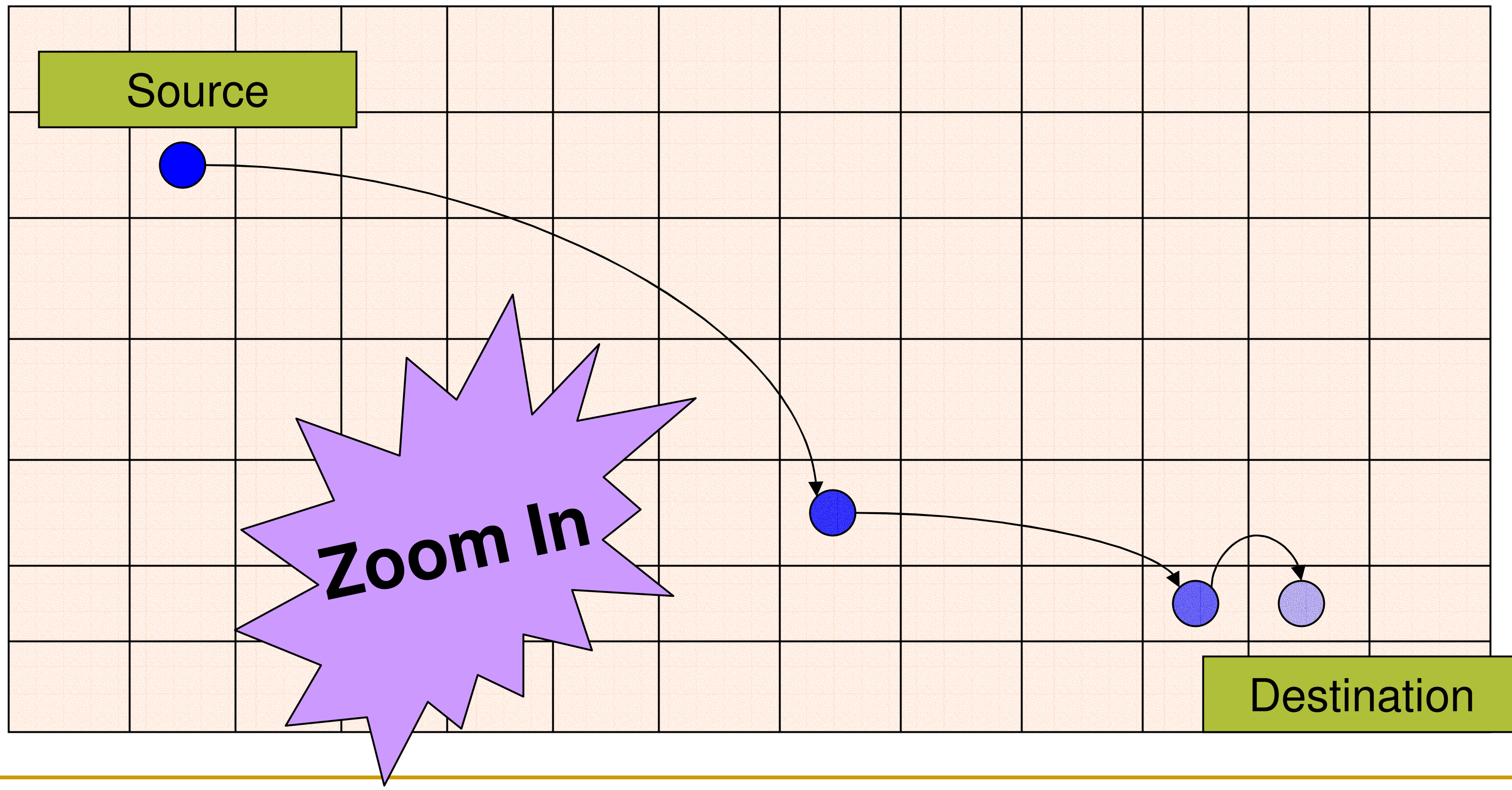
- Consistent Hashing
  - Map keys *and* nodes to an *identifier* space; implicit assignment of responsibility



- Mapping performed using hash functions (e.g., SHA-1)
  - Spread nodes and keys *uniformly* throughout

Fundamental Design Idea - II

- Prefix / Hypercube routing



But, there are so many of them!

- DHTs are hot!
- Scalability trade-offs
  - Routing table size at each node vs.
  - Cost of lookup and insert operations
- Simplicity
  - Routing operations
  - Join-leave mechanisms
- Robustness

Talk Outline

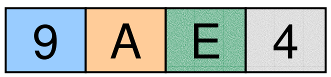
- DHT Designs
  - Plaxton Trees, Pastry/Tapestry
  - Chord
  - Overview: CAN, Symphony, Koorde, Viceroy, etc.
  - SkipNet
- DHT Applications
  - File systems, Multicast, Databases, etc.
- Conclusions / New Directions

Plaxton Trees [Plaxton, Rajaraman, Richa]

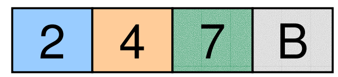
- Motivation
  - Access nearby copies of replicated objects
  - Time-space trade-off
    - Space = Routing table size
    - Time = Access hops

Plaxton Trees Algorithm

1. Assign labels to objects and nodes
   - using randomizing hash functions



Object

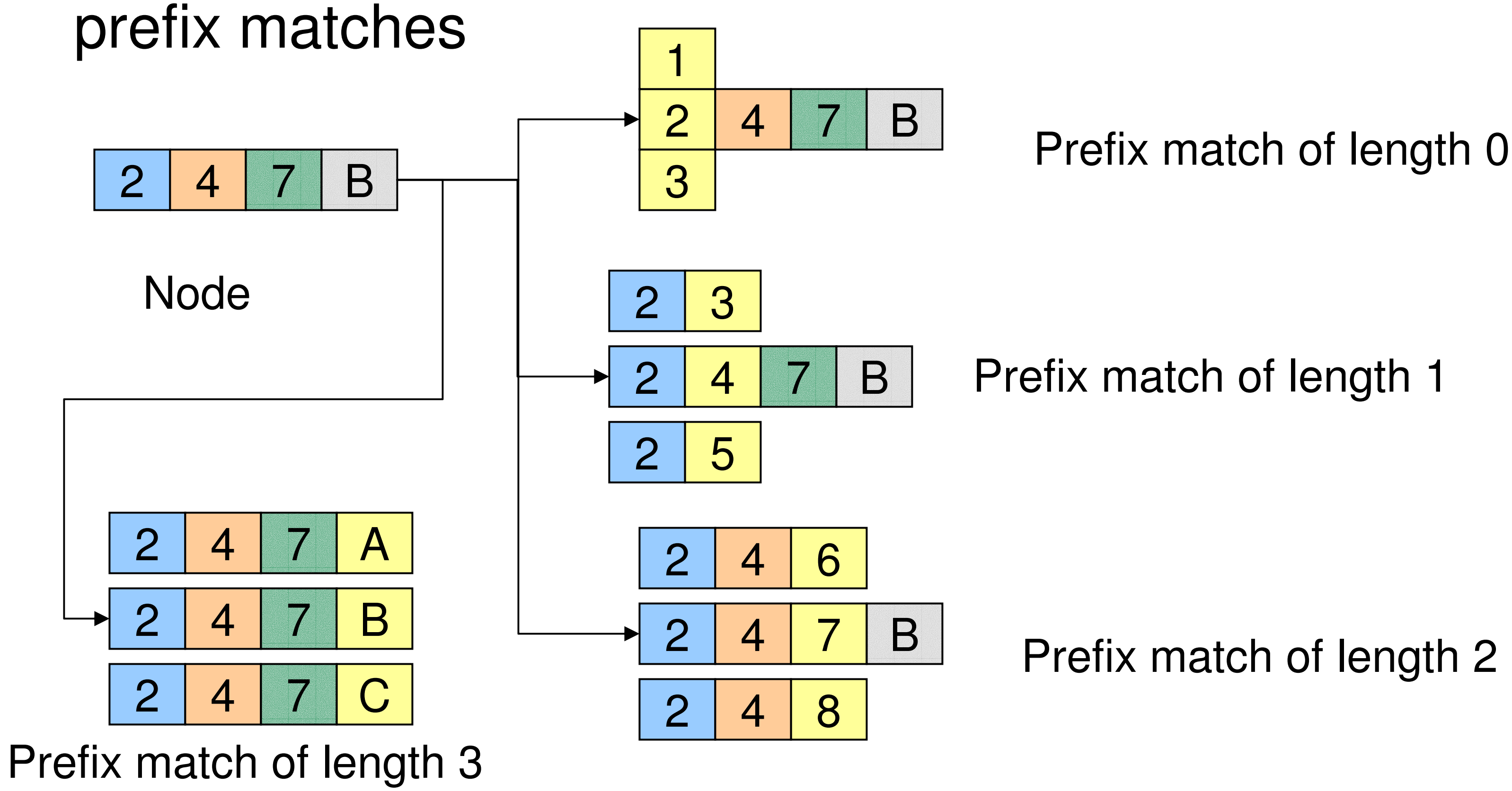


Node

Each label is of $\log_2^b n$ digits

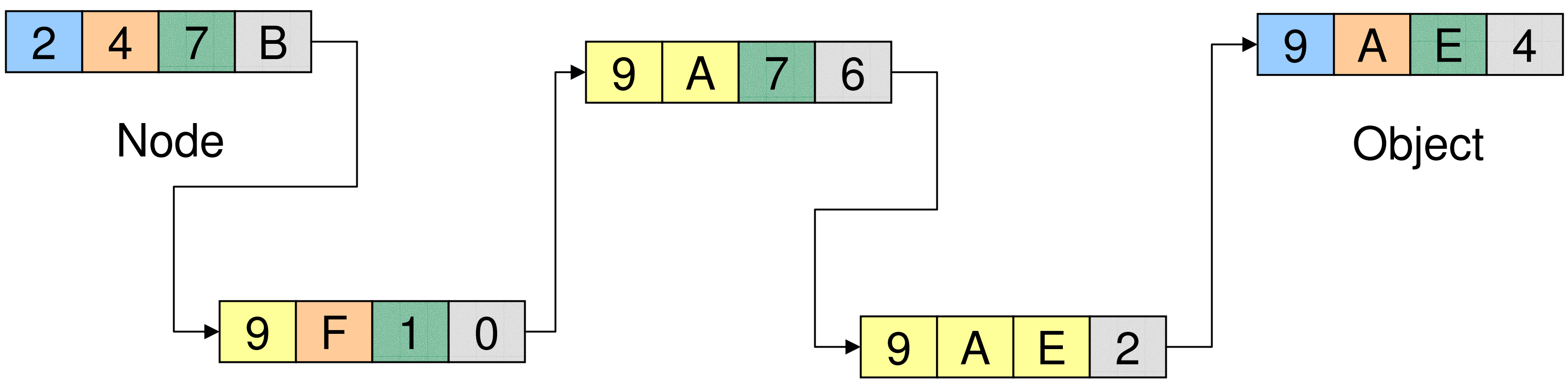
Plaxton Trees Algorithm

2. Each node knows about other nodes with varying prefix matches



Plaxton Trees
Object Insertion and Lookup

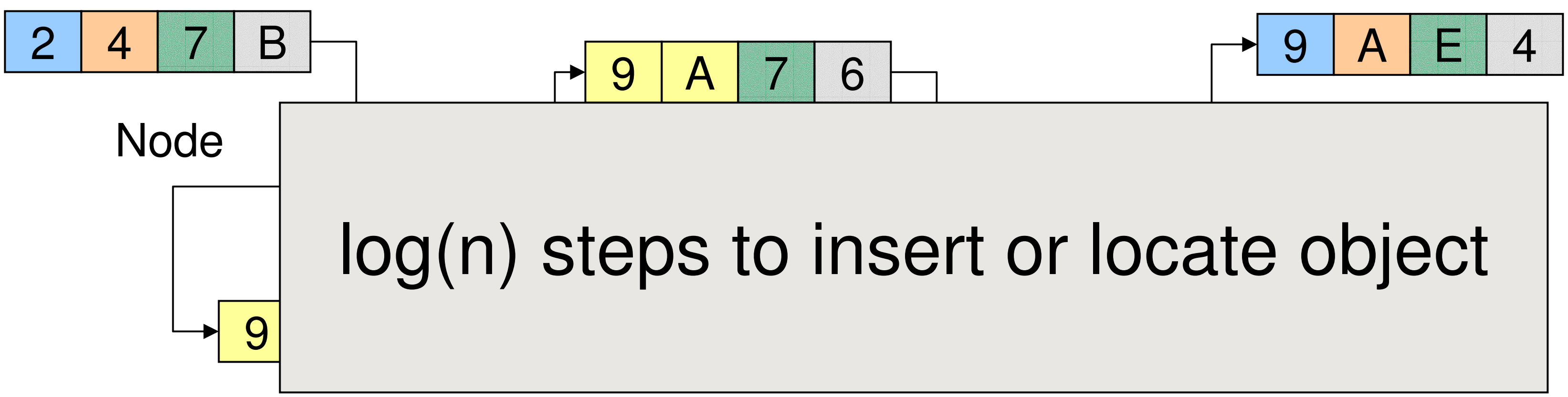
Given an object, route successively towards nodes with greater prefix matches



Store the object at each of these locations

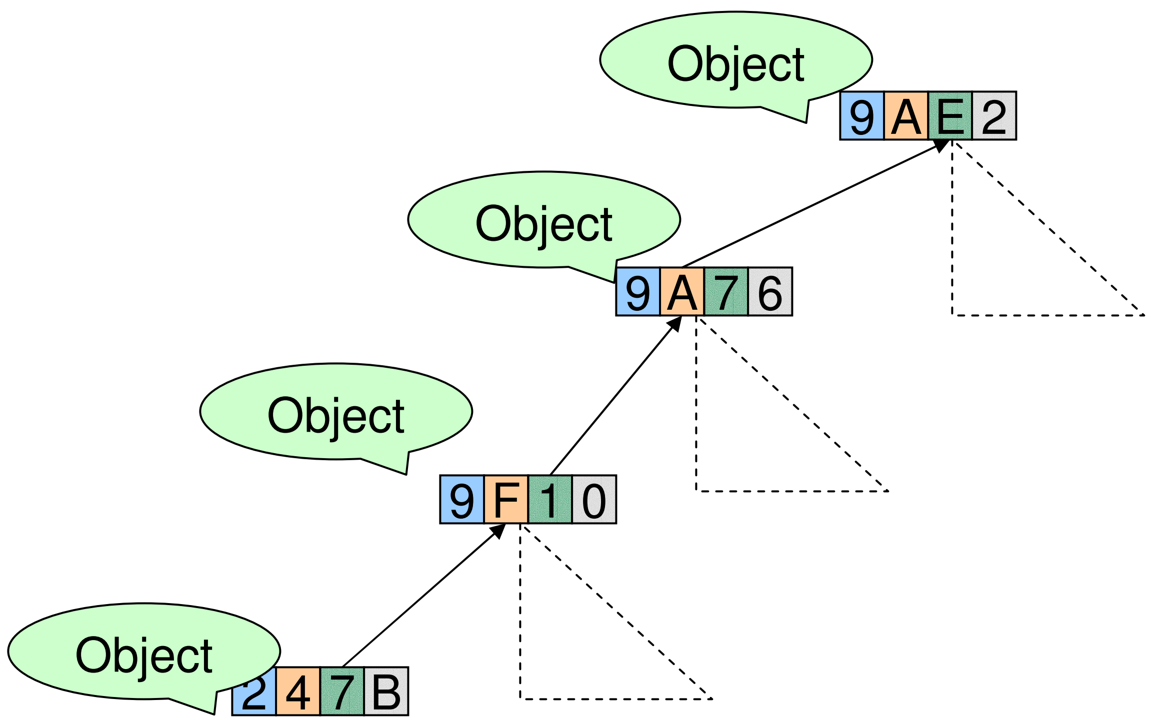
Plaxton Trees
Object Insertion and Lookup

Given an object, route successively towards nodes with greater prefix matches



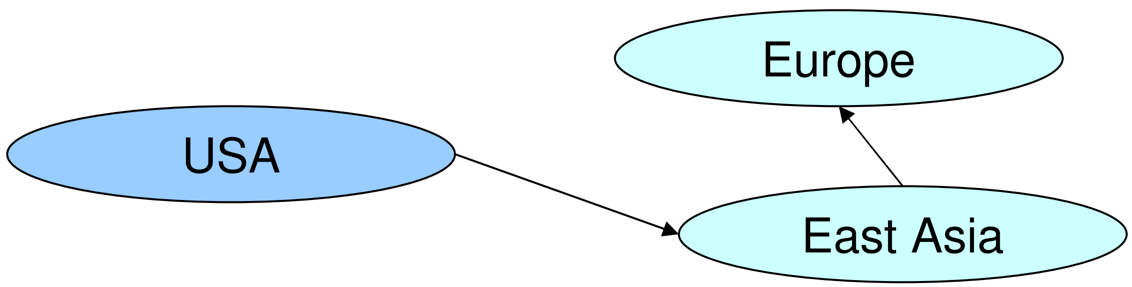
Store the object at each of these locations

Plaxton Trees
Why is it a tree?



Plaxton Trees
Network Proximity

- Overlay tree hops could be totally unrelated to the underlying network hops

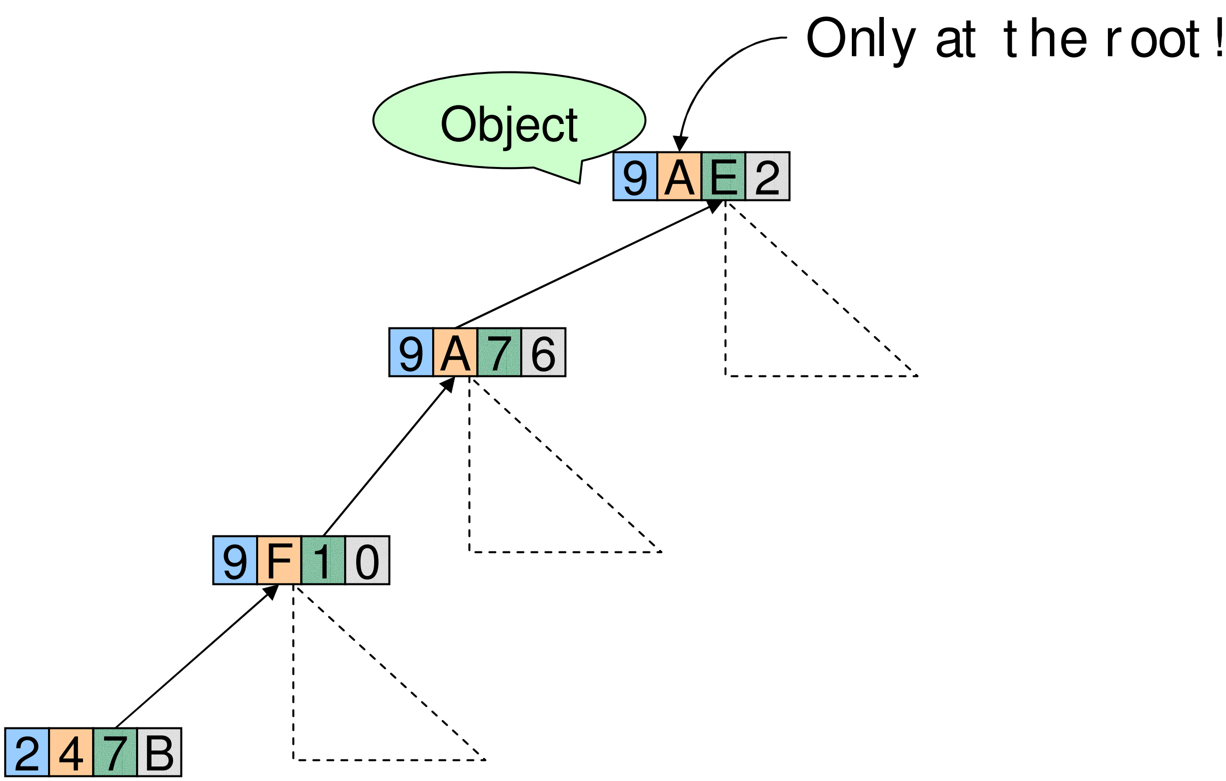


- Plaxton trees guarantee constant factor approximation!
  - Only when the topology is *uniform* in some sense

Pastry

- Based directly upon Plaxton Trees
- Exports a DHT interface
- Stores an object only at a node whose ID is *closest* to the object ID
- In addition to main routing table
  - Maintains *leaf set* of nodes
  - Closest L nodes (in ID space)
    - $L = 2^{(b+1)}$,typically -- one digit to left and right

Pastry



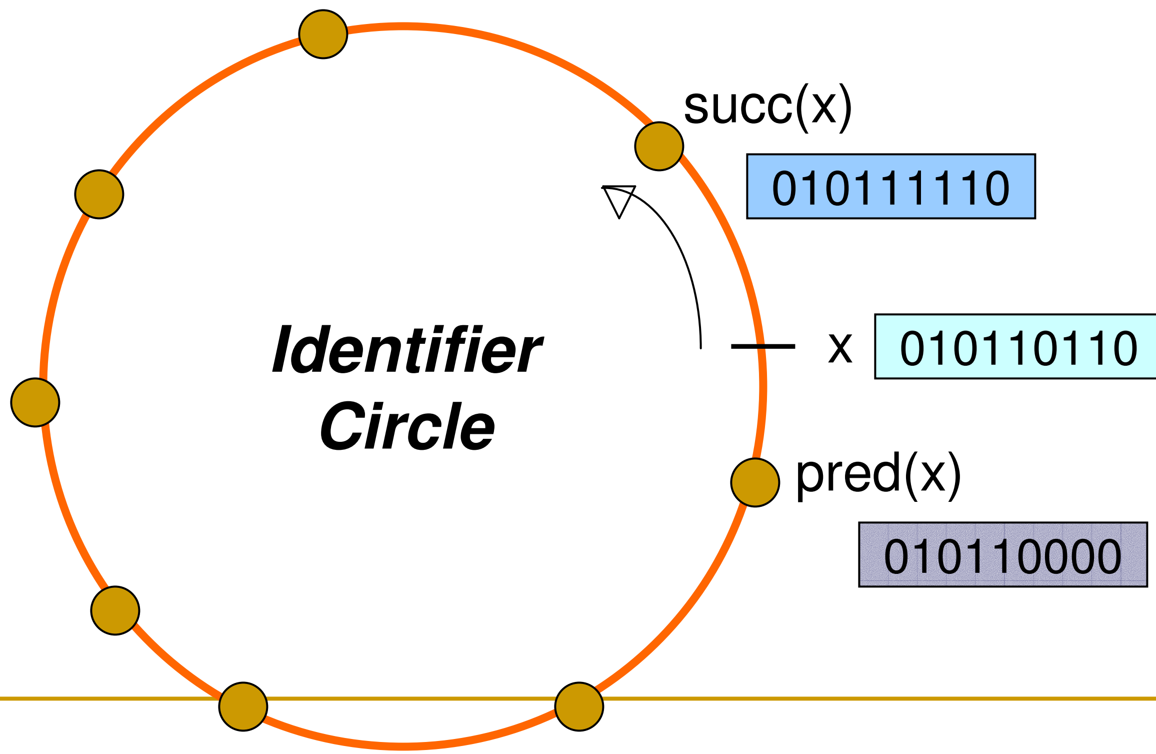
Key Insertion and Lookup = Routing to Root
➔ Takes O(log n) steps

Pastry
Self Organization

- Node join
  - Start with a node "close" to the joining node
  - Route a message to nodeID of new node
  - Take union of routing tables of the nodes on the path
- Joining cost: O(log n)
- Node leave
  - Update routing table
    - Query nearby members in the routing table
  - Update leaf set

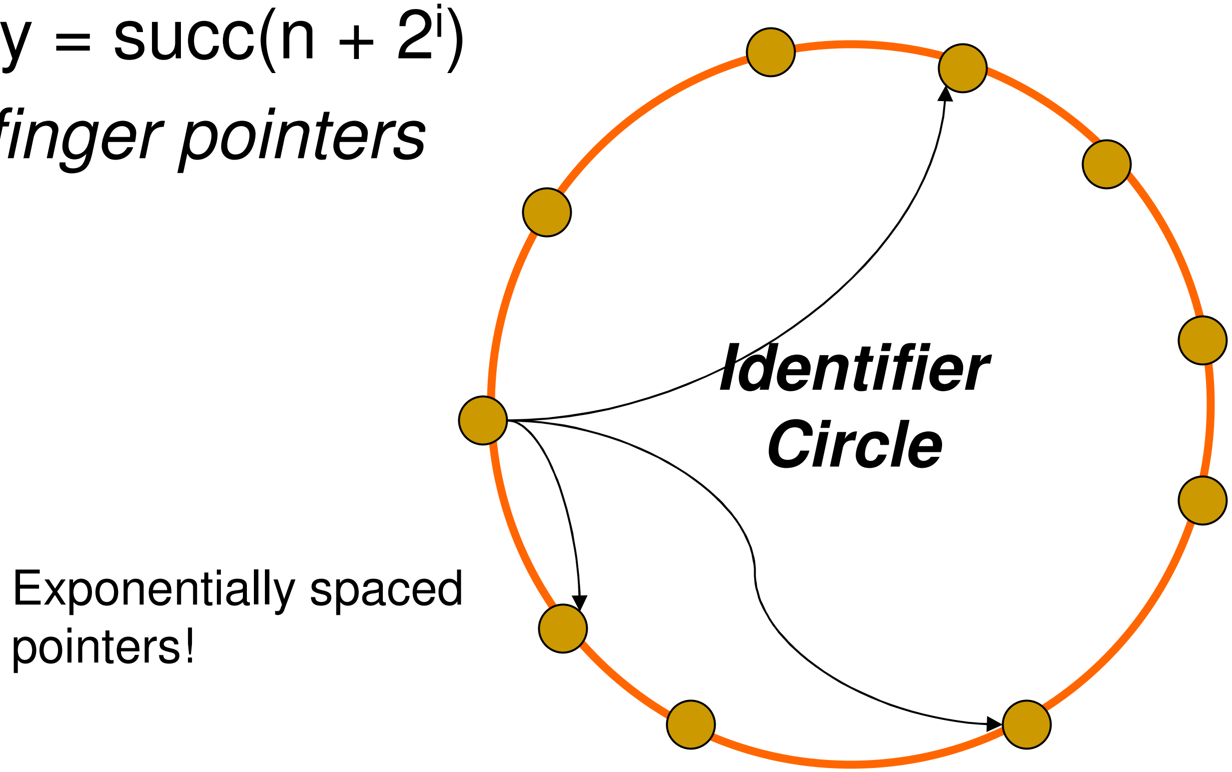
Chord [Karger, et al]

- Map nodes and keys to identifiers
  - Using randomizing hash functions
- Arrange them on a circle



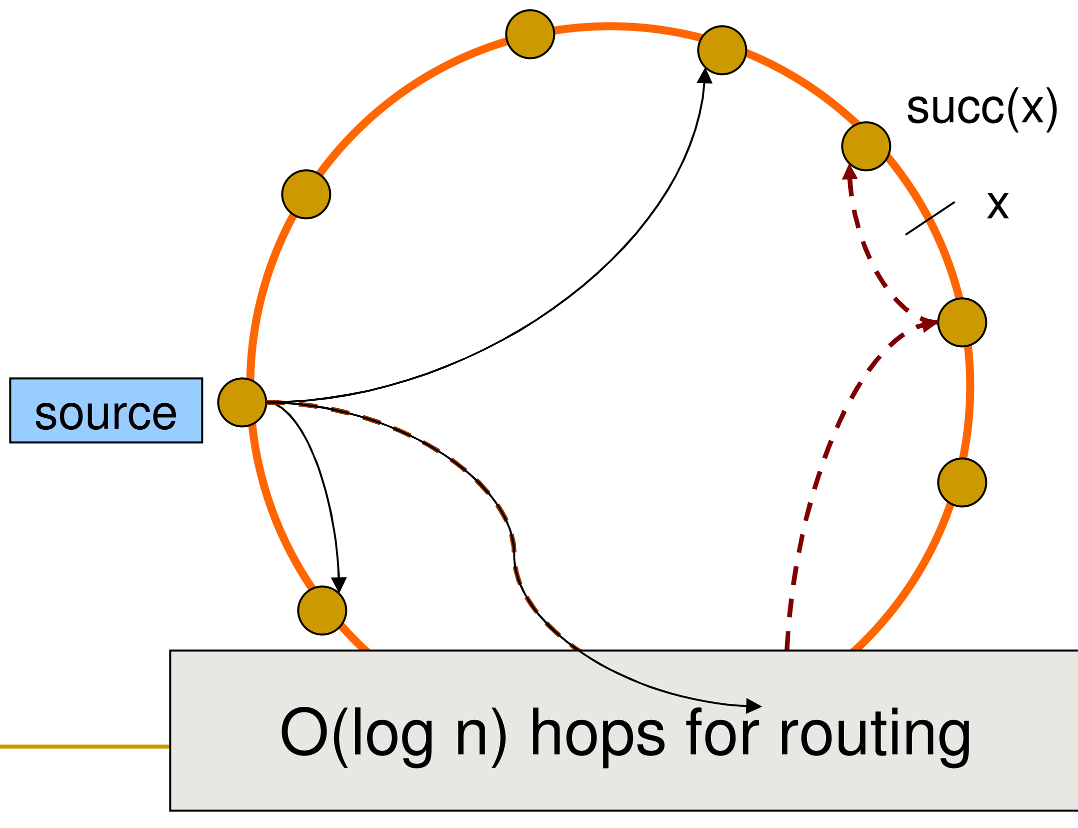
Chord
Efficient routing

- Routing table
  - i^{th} entry = succ(n + 2^i)
  - log(n) *finger pointers*



Chord
Key Insertion and Lookup

To insert or lookup a key ‘x’,
route to succ(x)



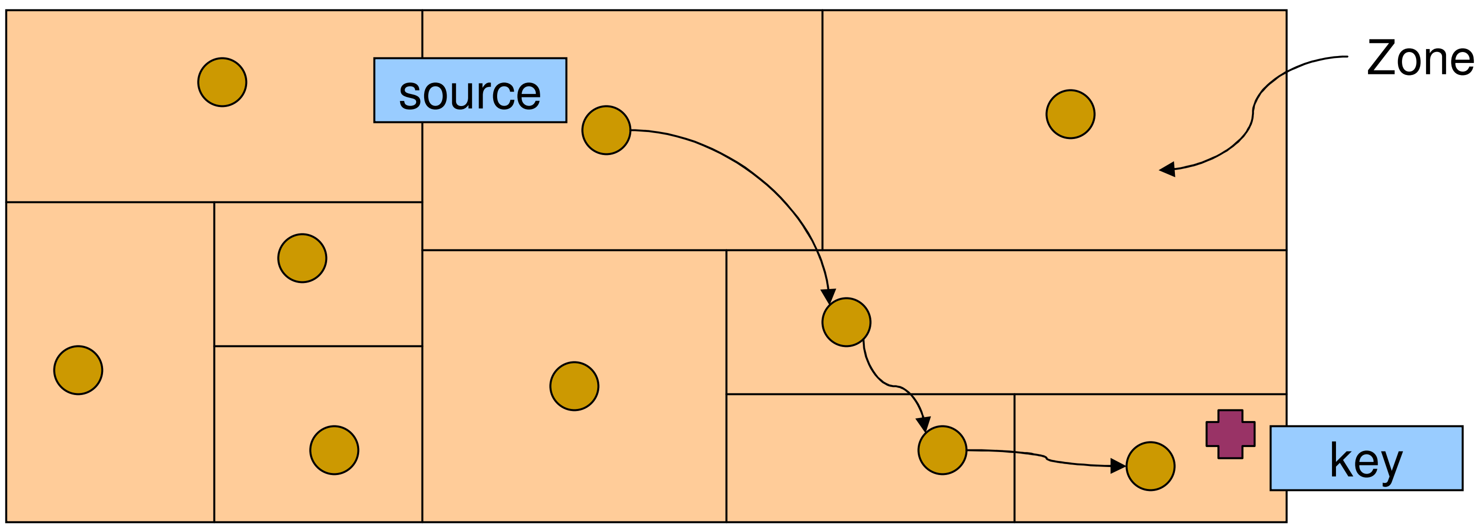
Chord

Self-organization

- Node join
  - Set up finger *i:* route to *succ(n + 2^i)*
  - log(n) fingers ⇒ O($\log^2$ n) cost
- Node leave
  - Maintain successor list for ring connectivity
  - Update successor list and finger pointers

CAN [Ratnasamy, et al]

- Map nodes and keys to *coordinates* in a multi-dimensional cartesian space



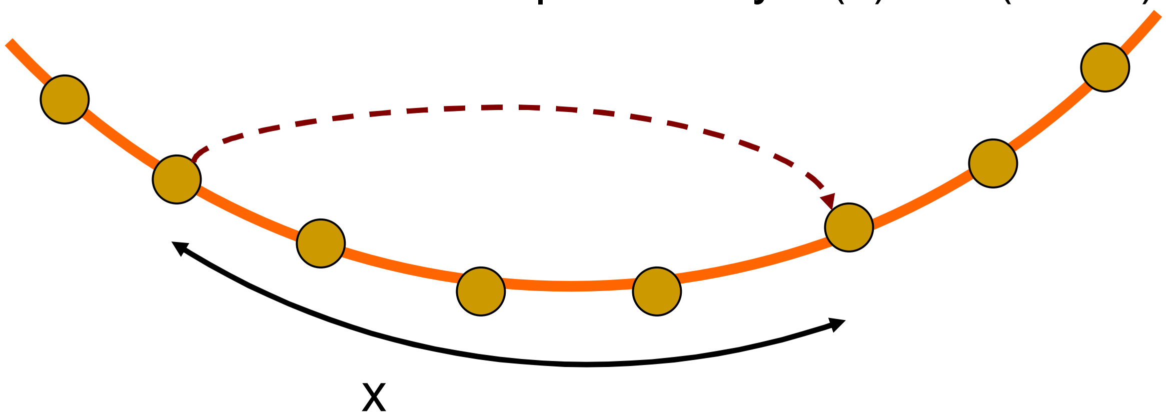
Routing through shortest Euclidean path

For d dimensions, routing takes $O(dn^{1/d})$ hops

Symphony [Manku, et al]

- Similar to Chord – mapping of nodes, keys
  - ‘k’ links are constructed *probabilistically!*

This link chosen with probability P(x) = 1/(x ln n)



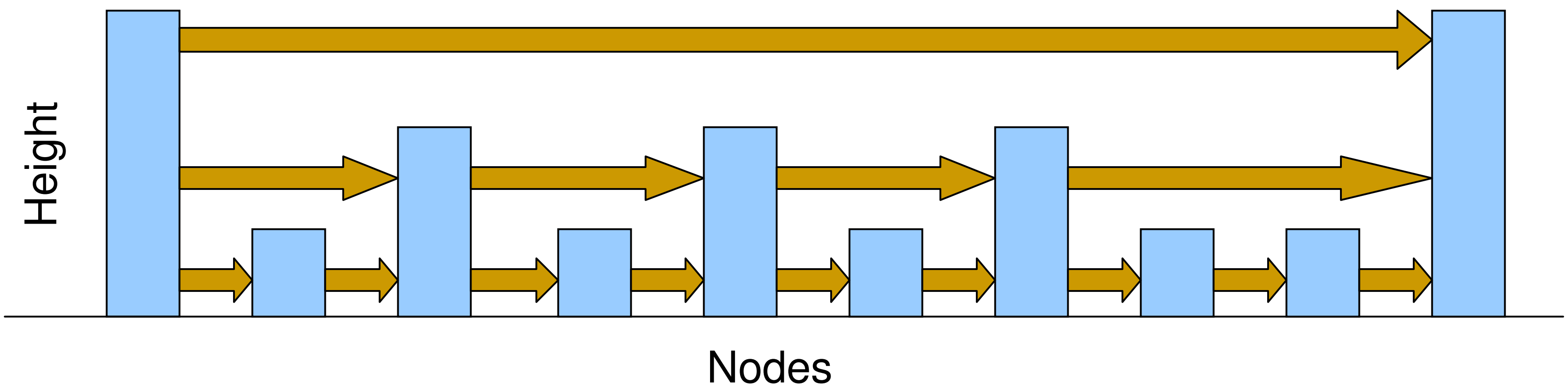
Expected routing guarantee: $O(1/k (\log^2 n))$ hops

SkipNet [Harvey, et al]

- Previous designs distribute data uniformly throughout the system
  - Good for load balancing
  - But, my data can be stored in Timbuktu!
  - Many organizations want stricter control over data placement
  - What about the routing path?
    - Should a Microsoft → Microsoft end-to-end path pass through Sun?

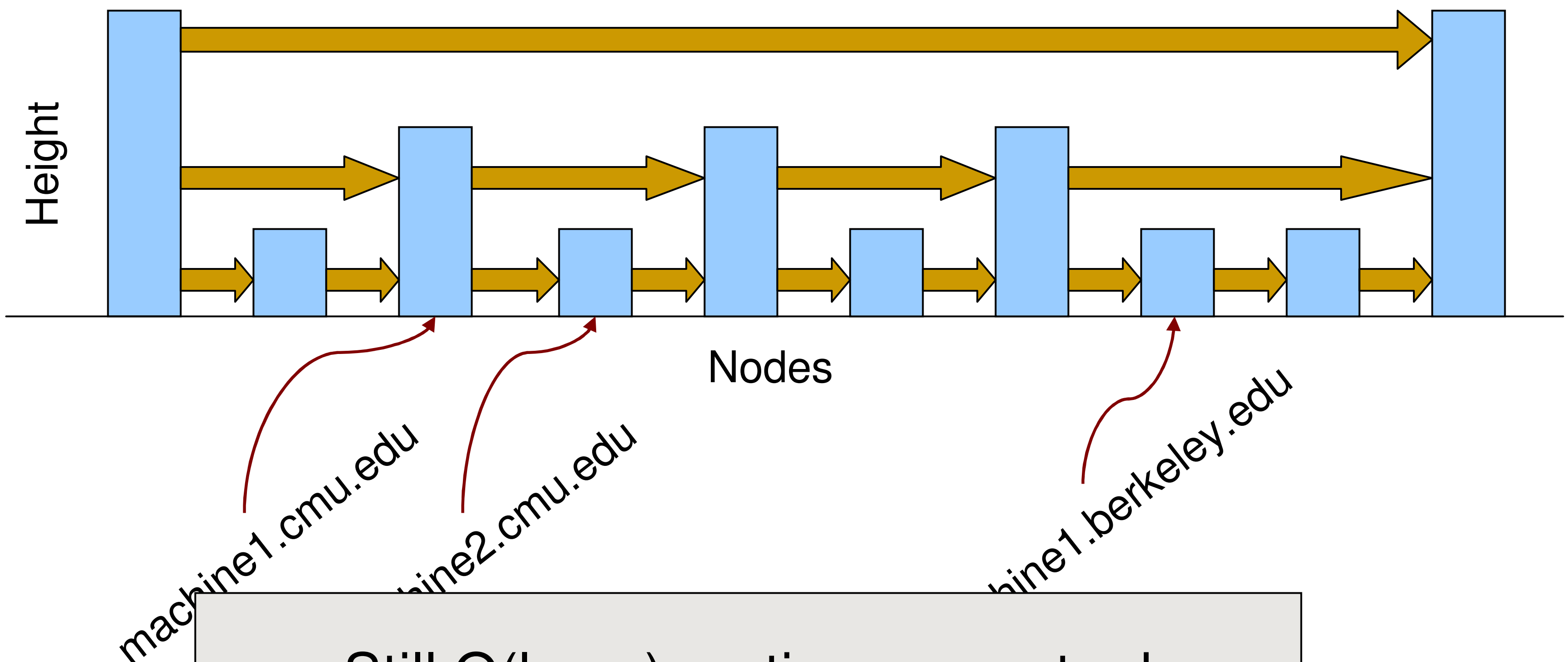
SkipNet
Content and Path Locality

Basic Idea: Probabilistic skip lists



- Each node choose a height at random
  - Choose height ‘h’ with probability $1/2^h$

SkipNet
Content and Path Locality



Still O(log n) routing guarantee!

- Nodes are *lexicographically sorted*

Summary (Ah, at last!)

	# Links per node	Routing hops
Pastry/Tapestry	$O(2^b \log_{2^b} n)$	$O(\log_{2^b} n)$
Chord	$\log n$	$O(\log n)$
CAN	d	$dn^{1/d}$
SkipNet	$O(\log n)$	$O(\log n)$
Symphony	k	$O((1/k) \log^2 n)$
Koorde	d	$\log_d n$
Viceroy	7	$O(\log n)$

Optimal (= lower bound)

What can DHTs do for us?

- Distributed object lookup
  - Based on object ID
- De-centralized file systems
  - CFS, PAST, Ivy
- Application Layer Multicast
  - Scribe, Bayeux, Splitstream
- Databases
  - PIER

De-centralized file systems

- CFS [Chord]
  - *Block* based read-only storage
- PAST [Pastry]
  - *File* based read-only storage
- Ivy [Chord]
  - *Block* based read-write storage

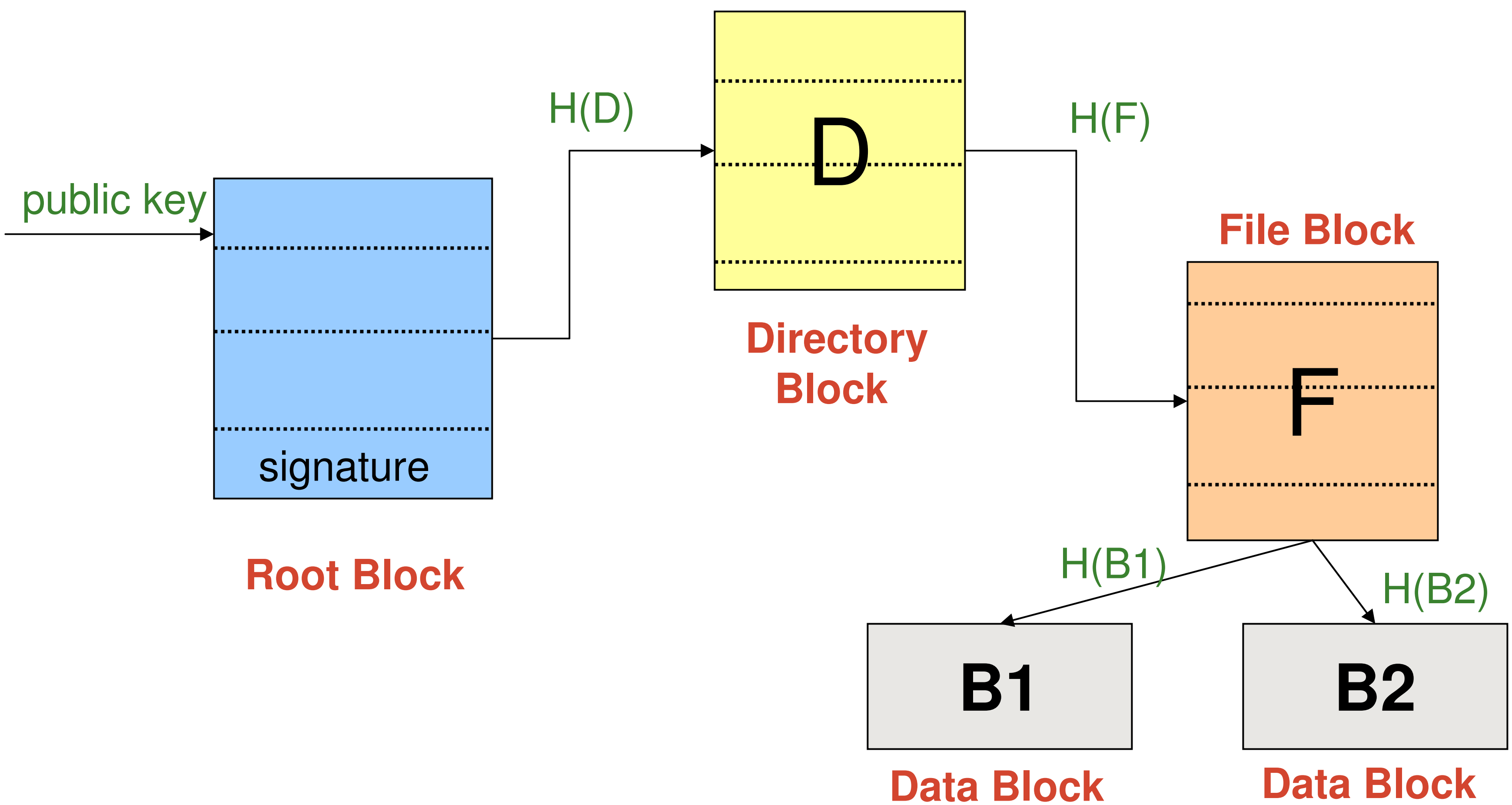
PAST

- Store file
  - Insert (filename, file) into Pastry
  - Replicate file at the leaf-set nodes
- Cache if there is empty space at a node

CFS

- Blocks are inserted into Chord DHT
  - insert(blockID, block)
  - Replicated at successor list nodes
- Read root block through public key of file system
- Lookup other blocks from the DHT
  - Interpret them to be the file system
- Cache on lookup path

CFS



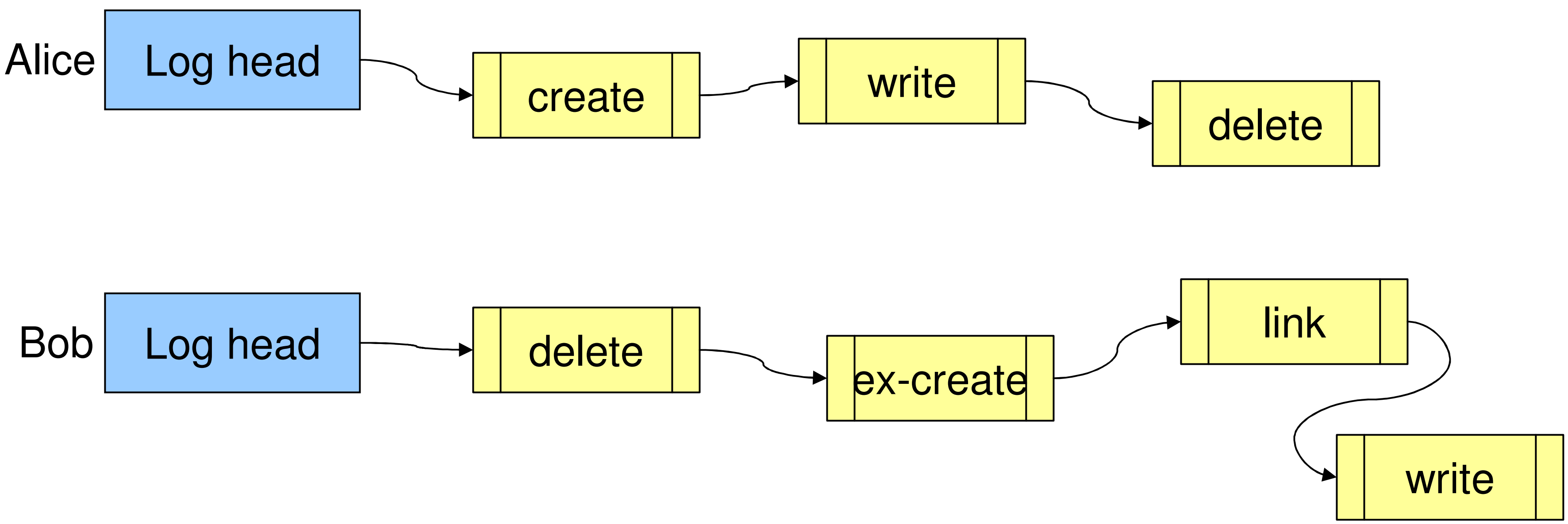
public key
signature
Root Block
H(D)
D
Directory Block
H(F)
File Block
F
H(B1)
H(B2)
B1
B2
Data Block
Data Block

CFS vs. PAST

- Block-based vs. File-based
  - Insertion, lookup and replication
- CFS has better performance for small popular files
  - Performance comparable to FTP for larger files
- PAST is susceptible to storage imbalances
  - Plaxton trees can provide it network locality

Ivy

- Each user maintains a log of updates
- To construct file system, scan logs of all users



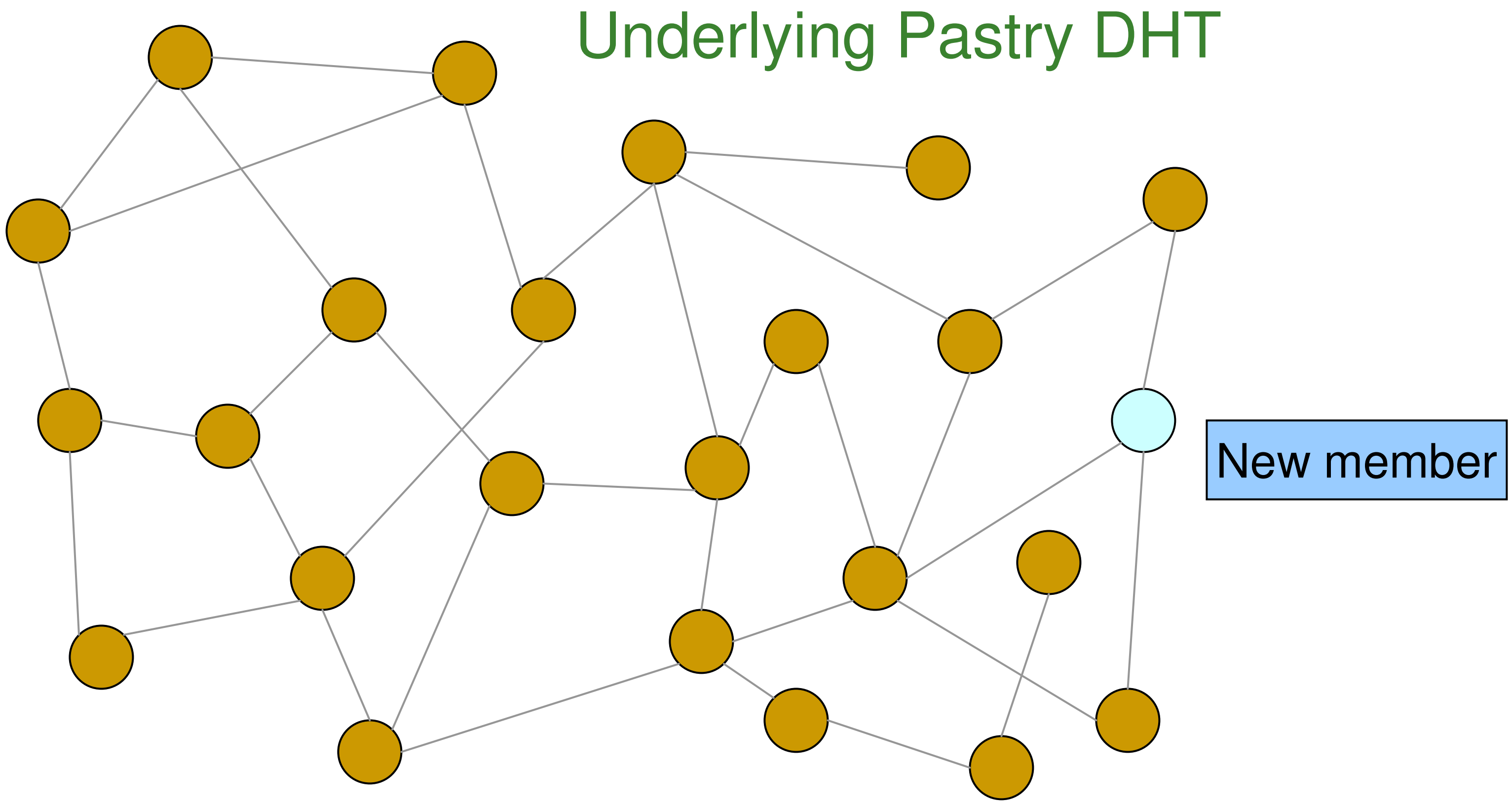
Ivy

- Starting from log head – stupid
  - Make periodic snapshots
- Conflicts will arise
  - For resolution, use any tactics (e.g., Coda’s)

Application Layer Multicast

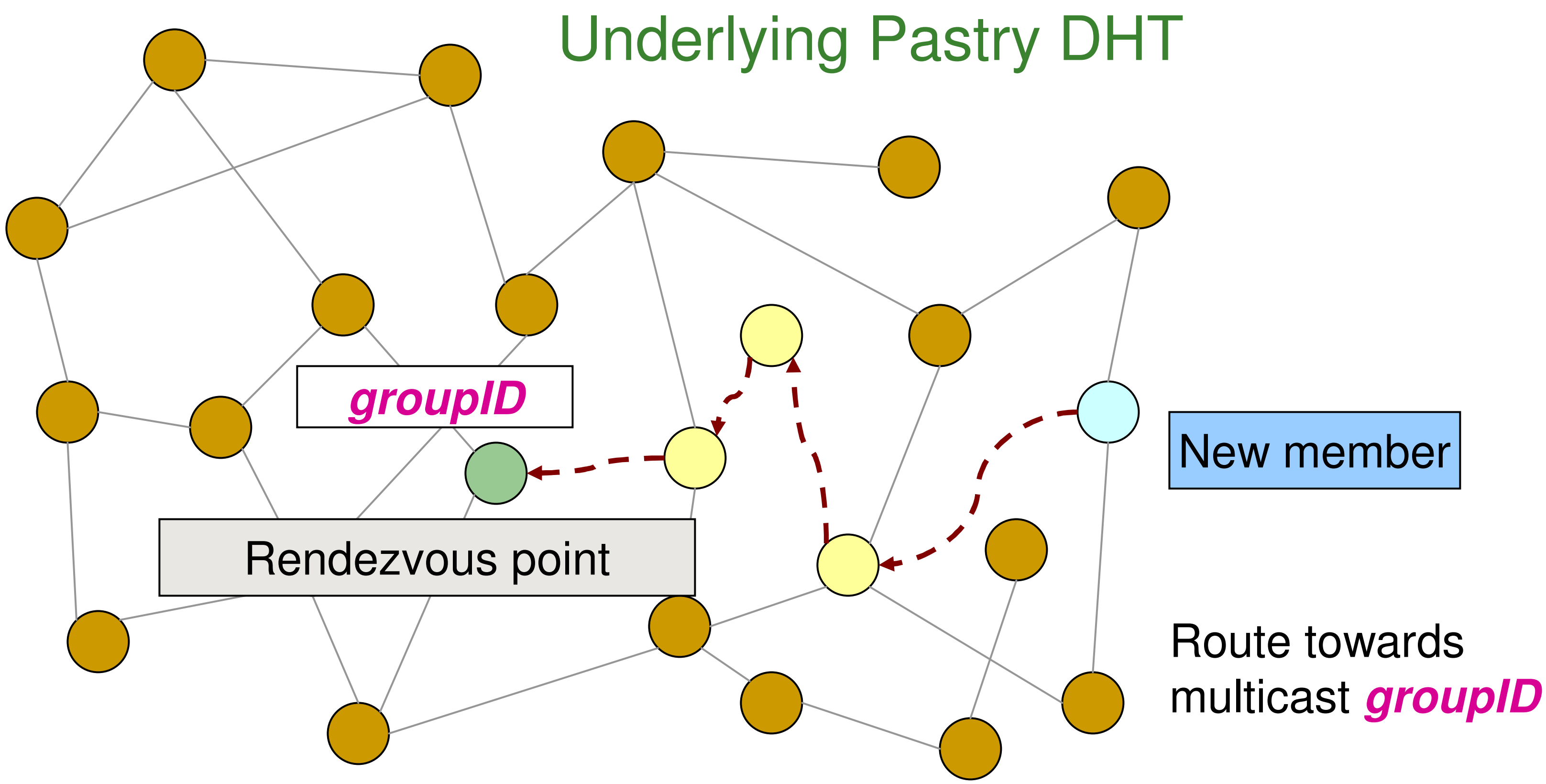
- Embed multicast tree(s) over the DHT graph
- Multiple source; multiple groups
  - Scribe
  - CAN-based multicast
  - Bayeux
- Single source; multiple trees
  - Splitstream

Scribe



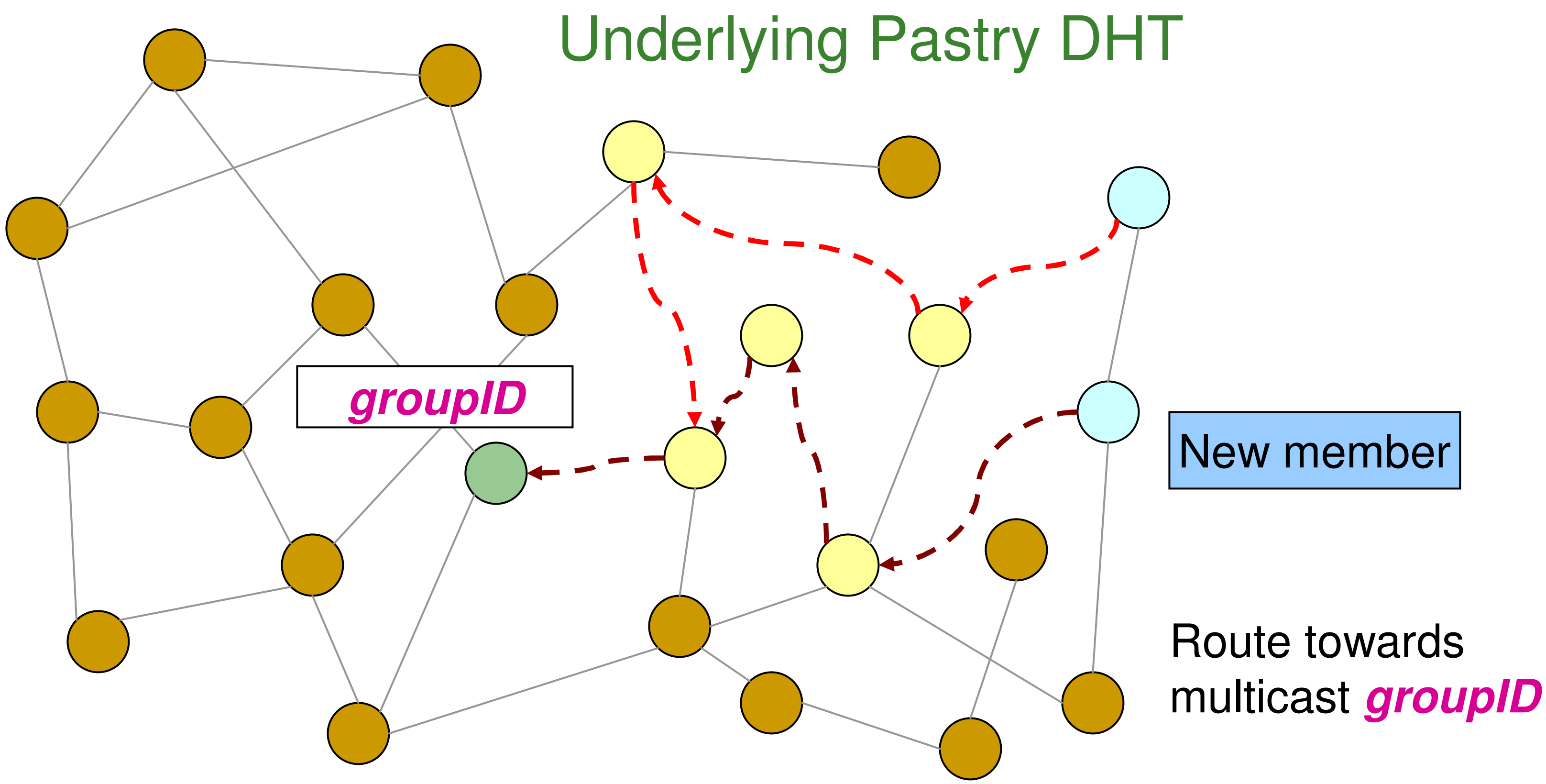
Scribe

Tree construction



Scribe

Tree construction



Scribe

Discussion

- Very scalable
  - Inherits scalability from the DHT
- Anycast is a simple extension
- How good is the multicast tree?
  - As compared to native IP multicast
  - Comparison to Narada
- Node heterogeneity not considered

SplitStream

- Single source, high bandwidth multicast
- Idea
  - Use multiple trees instead of one
  - Make them *internal-node-disjoint*
    - Every node is an internal node in only one tree
  - Satisfies bandwidth constraints
  - Robust
- Use cute Pastry prefix-routing properties to construct node-disjoint trees

Databases, Service Discovery

SOME OTHER TIME!

Where are we now?

- Many DHTs offering efficient and relatively robust routing
- Unanswered questions
  - Node heterogeneity
  - Network-efficient overlays vs. Structured overlays
    - Conflict of interest!
  - What happens with high user churn rate?
  - Security

Are DHTs a panacea?

- Useful primitive
- Tension between network efficient construction and uniform key-value distribution
- Does every non-distributed application use only hash tables?
  - Many rich data structures which cannot be built on top of hash tables alone
  - Exact match lookups are not enough
  - Does any P2P file-sharing system use a DHT?