A Scalable, Content-Addressable Network

Sylvia Ratnasamy,[1,2] Paul Francis,[3] Mark Handley,[1]

Richard Karp,[1,2] Scott Shenker[1]

[1] ACIRI [2] U.C.Berkeley [3] Tahoe Networks

Outline

- Introduction
- Design
- Evaluation
- Strengths & Weaknesses
- Ongoing Work

Internet-scale hash tables

- Hash tables
  - essential building block in software systems
- Internet-scale distributed hash tables
  - equally valuable to large-scale distributed systems?

Internet-scale hash tables

- Hash tables
  - essential building block in software systems
- Internet-scale distributed hash tables
  - equally valuable to large-scale distributed systems?
    - peer-to-peer systems
      - Napster, Gnutella, Groove, FreeNet, MojoNation...
    - large-scale storage management systems
      - Publius, OceanStore, PAST, Farsite, CFS ...
    - mirroring on the Web

Content-Addressable Network (CAN)

- CAN: Internet-scale hash table
- Interface
  - insert(key,value)
  - value = retrieve(key)

Content-Addressable Network (CAN)

- CAN: Internet-scale hash table
- Interface
  - insert(key,value)
  - value = retrieve(key)
- Properties
  - scalable
  - operationally simple
  - good performance (w/ improvement)

Content-Addressable Network (CAN)

- CAN: Internet-scale hash table
- Interface
  - insert(key,value)
  - value = retrieve(key)
- Properties
  - scalable
  - operationally simple
  - good performance
- Related systems: Chord/Pastry/Tapestry/Buzz/Plaxton ...

Problem Scope

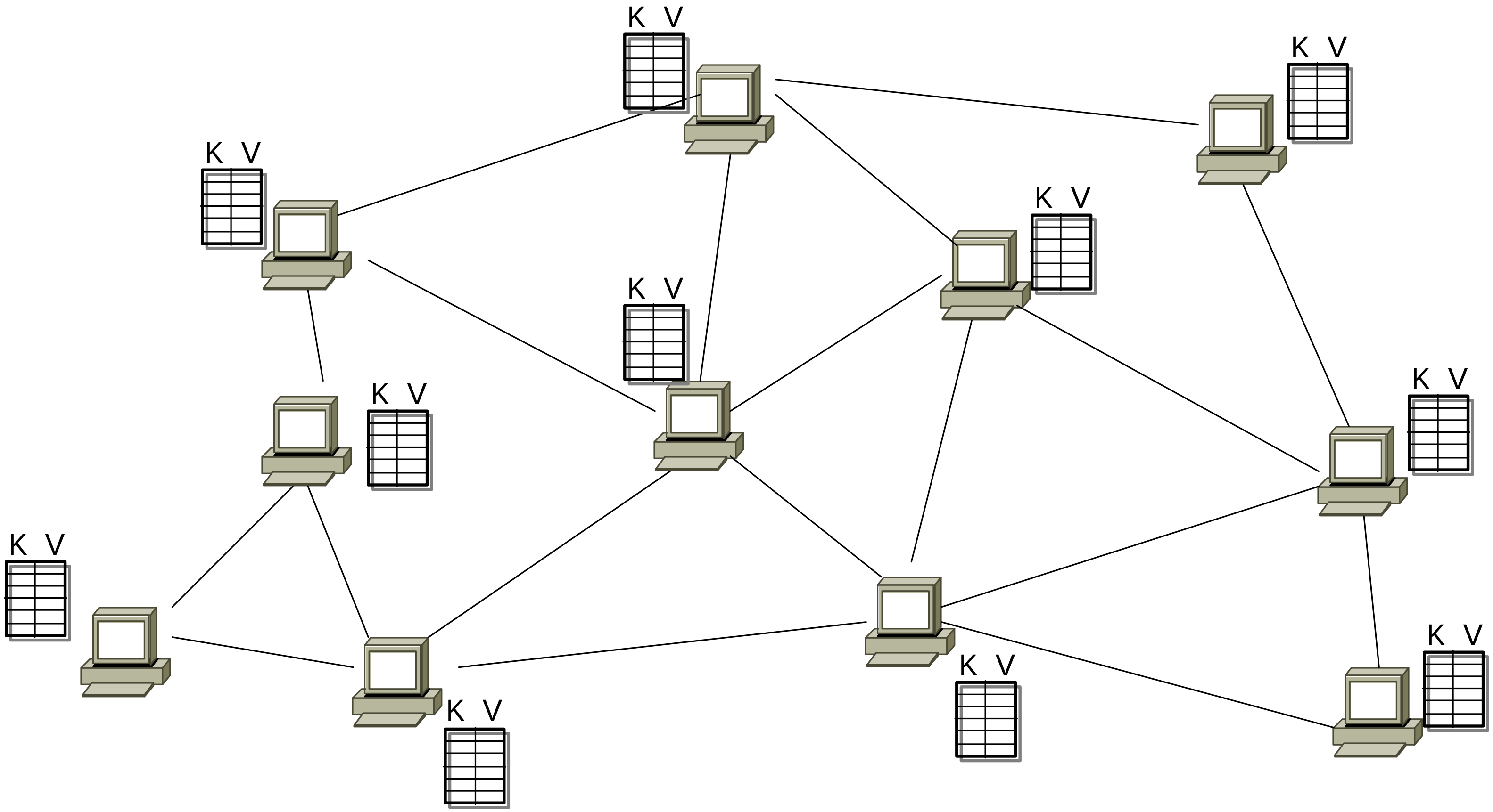
- Design a system that provides the interface
  - scalability
  - robustness
  - performance
  - security

- Application-specific, higher level primitives
  - keyword searching
  - mutable content
  - anonymity

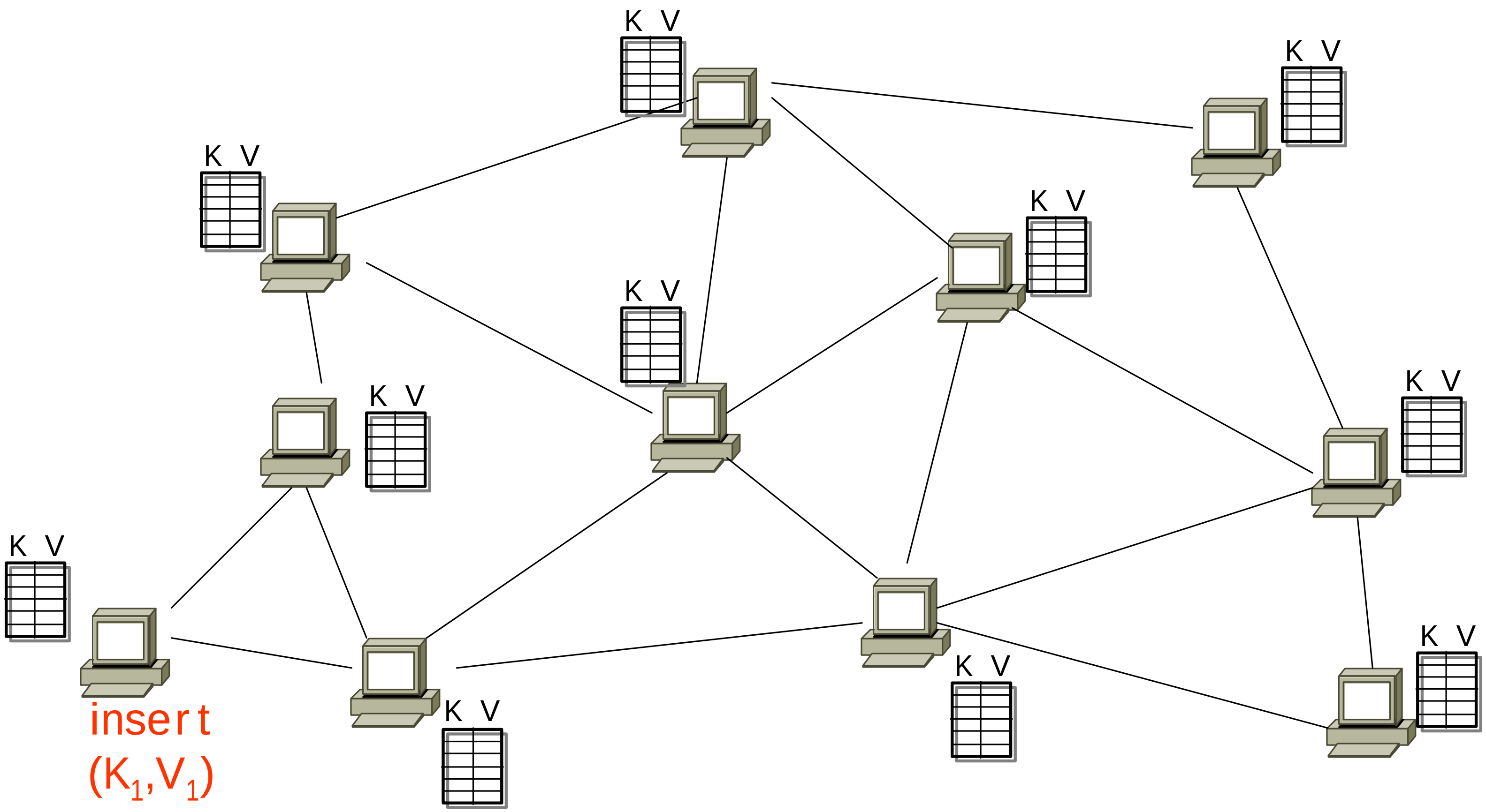
Outline

- Introduction
- Design
- Evaluation
- Strengths & Weaknesses
- Ongoing Work

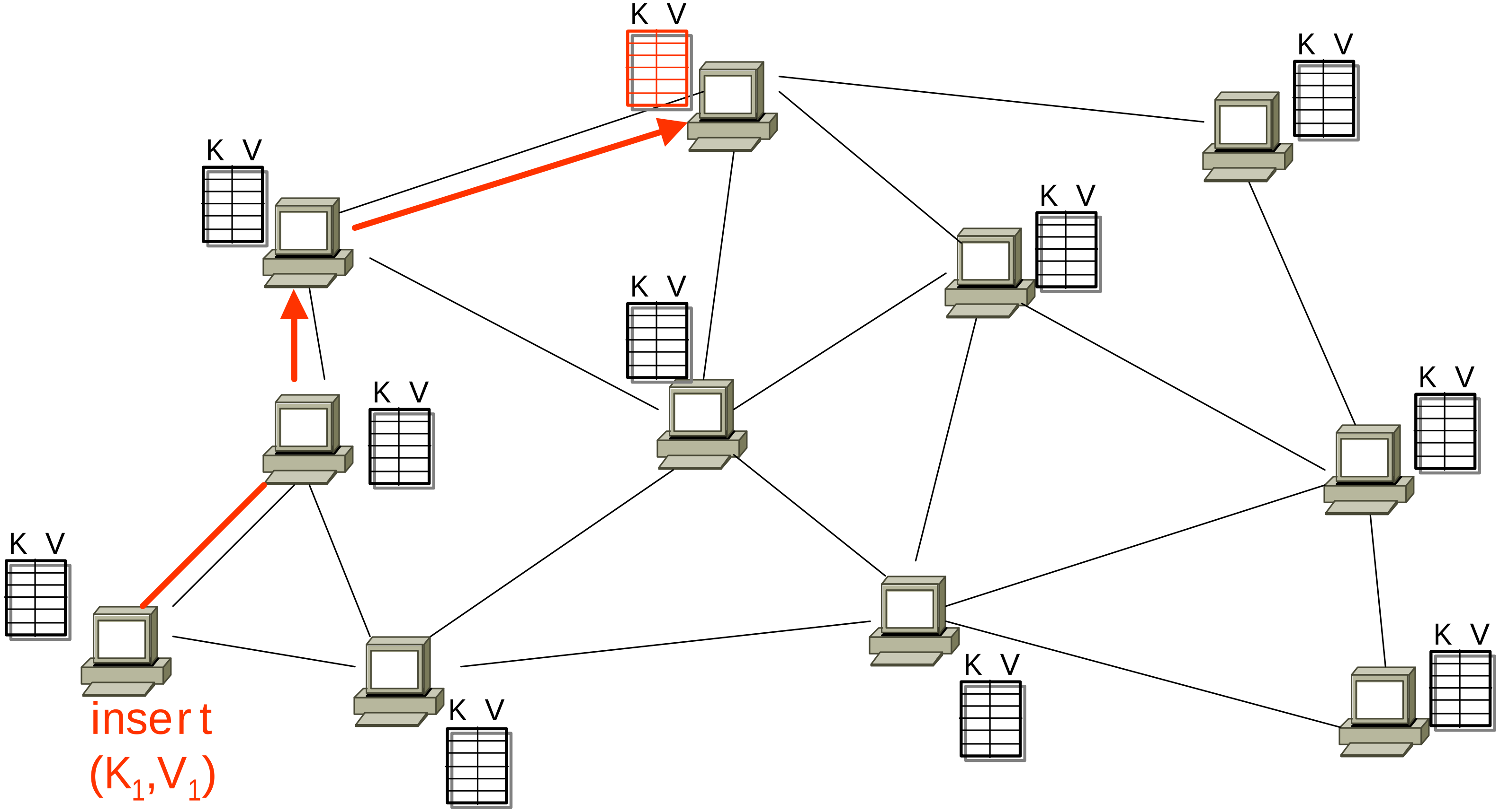
CAN: basic idea



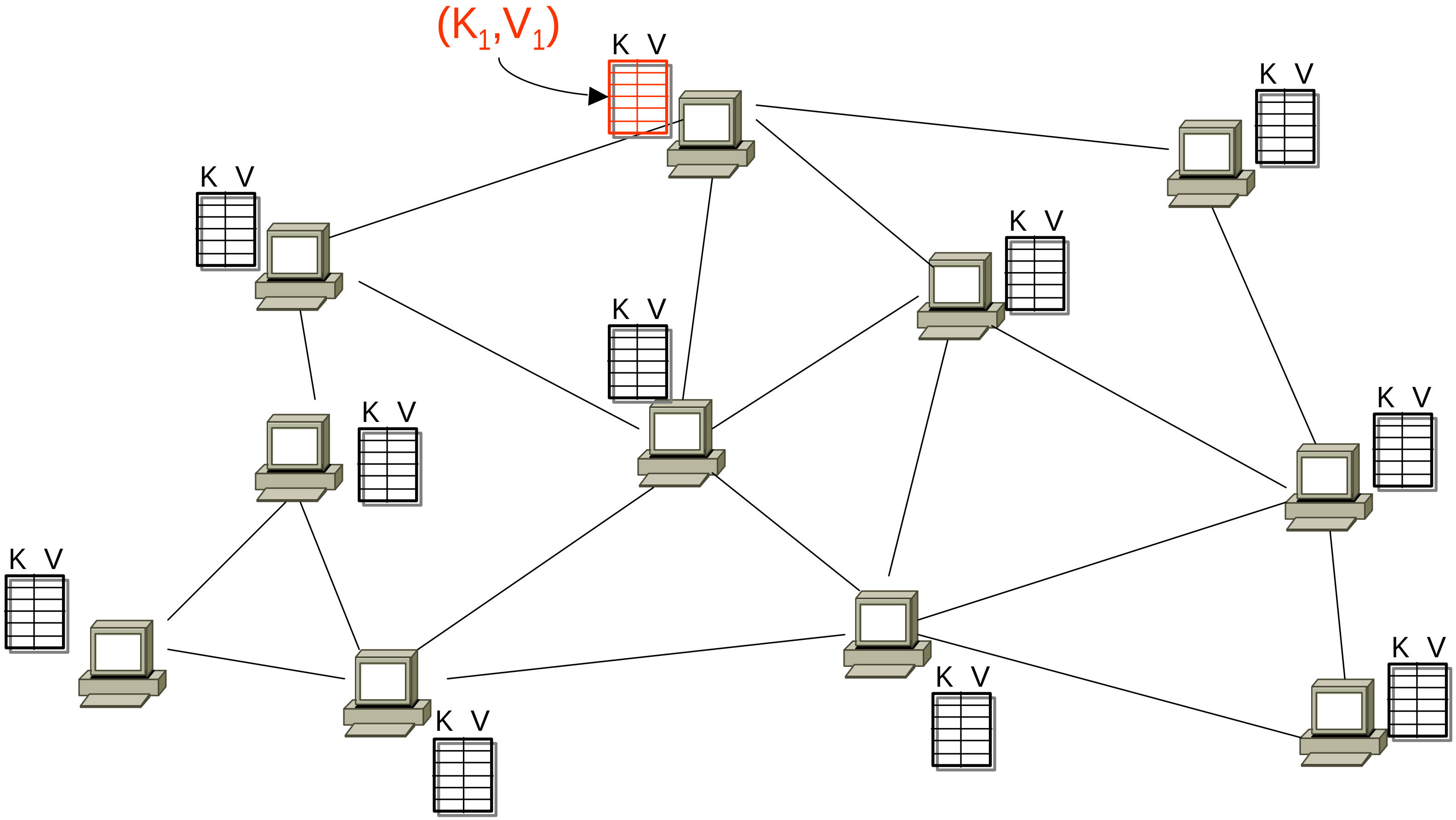
CAN: basic idea



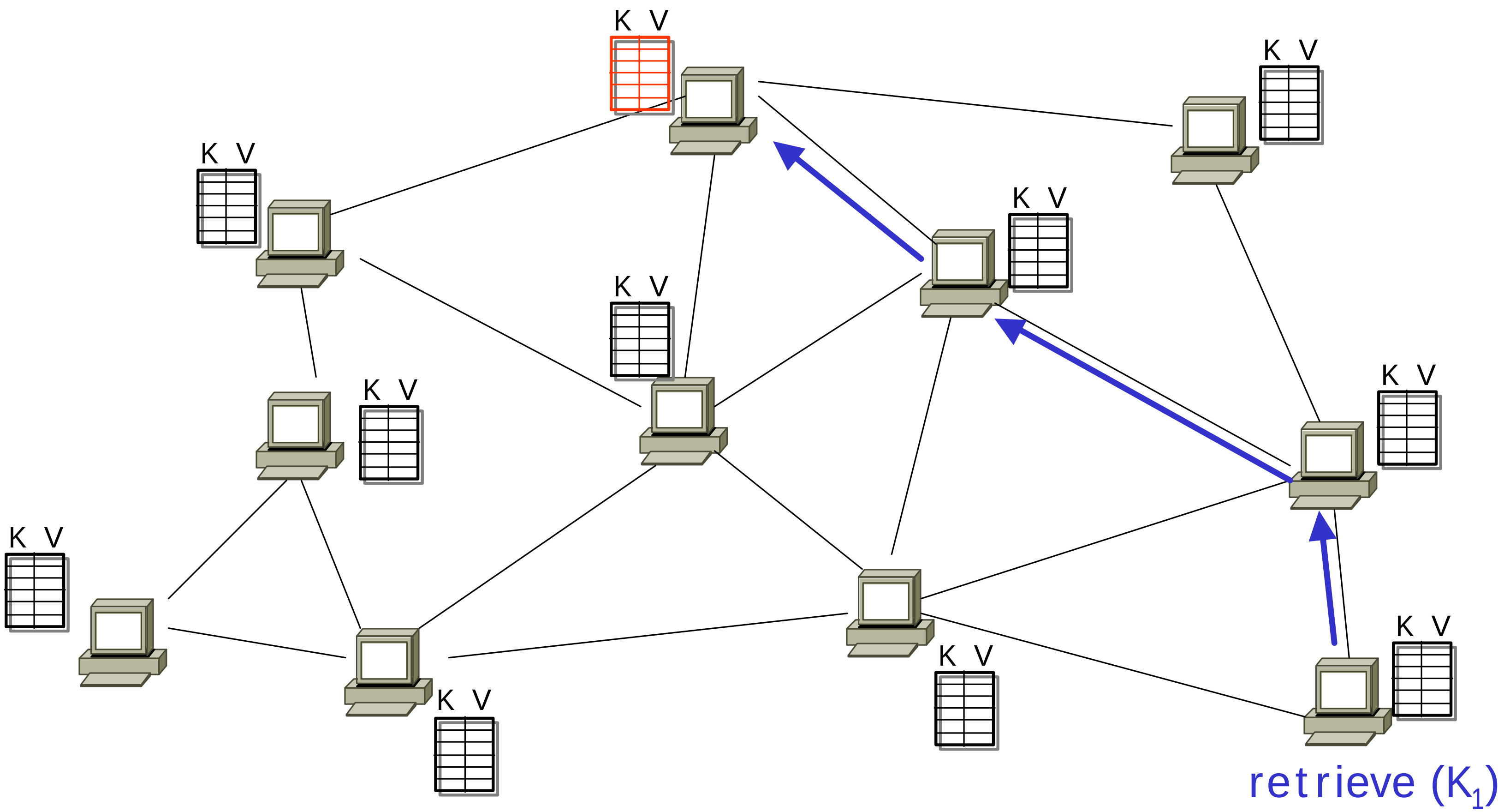
CAN: basic idea



CAN: basic idea



CAN: basic idea



CAN: solution

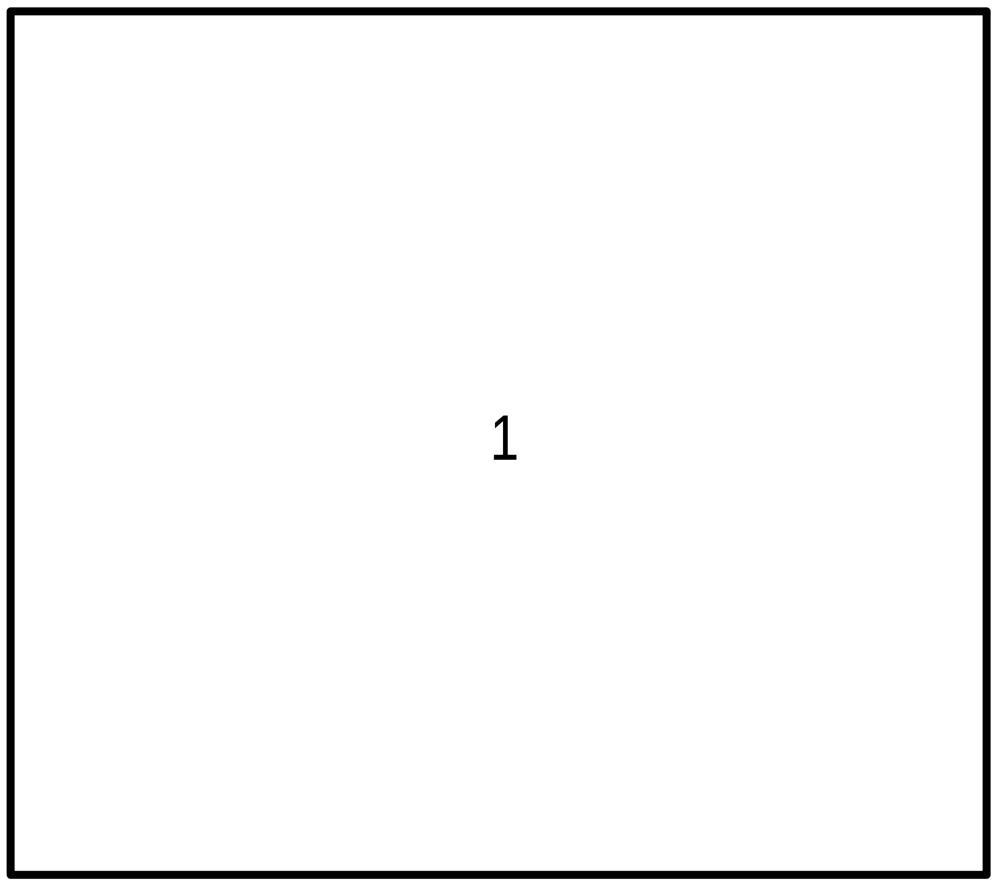
- virtual Cartesian coordinate space

- entire space is partitioned amongst all the nodes
  - every node “owns” a zone in the overall space

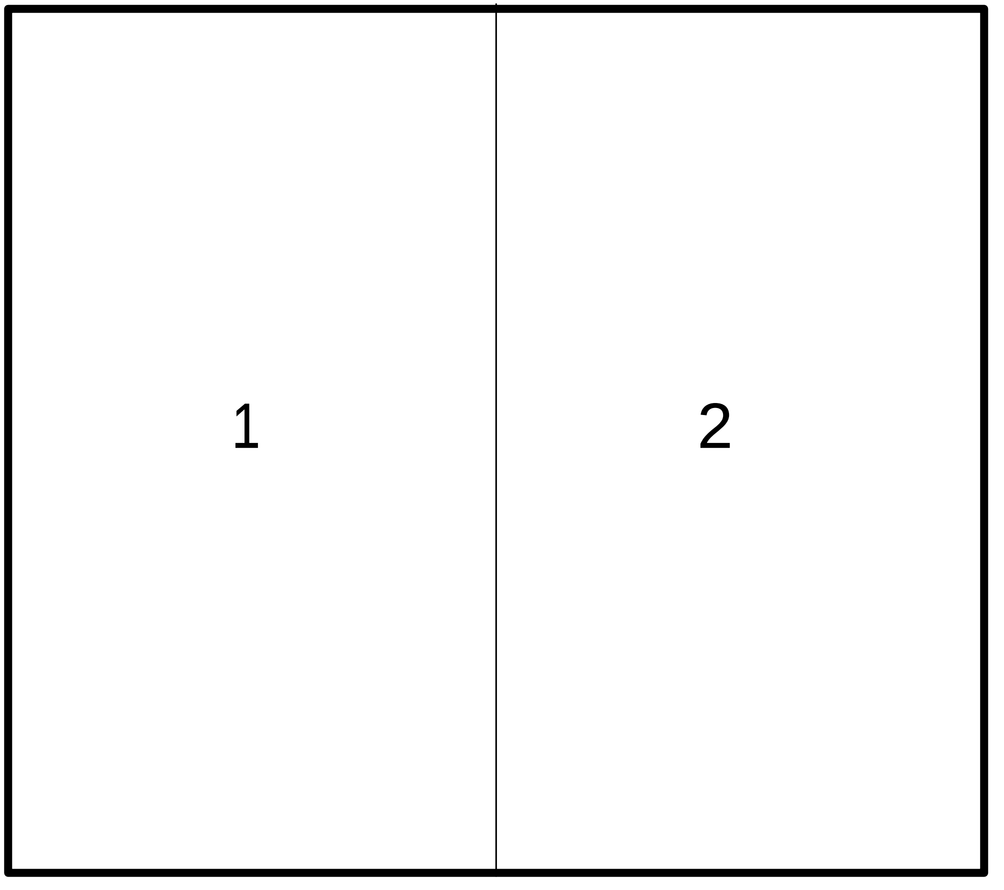
- abstraction
  - can store data at “points” in the space
  - can route from one “point” to another

- point = node that owns the enclosing zone

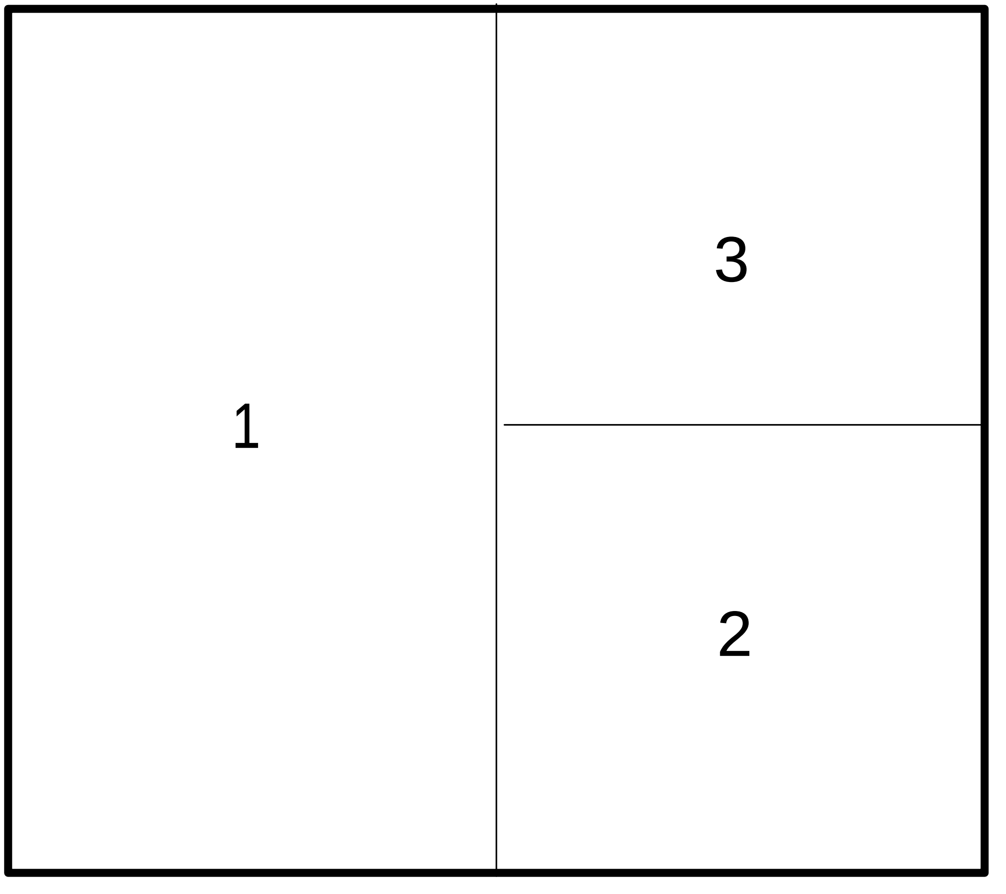
CAN: simple example



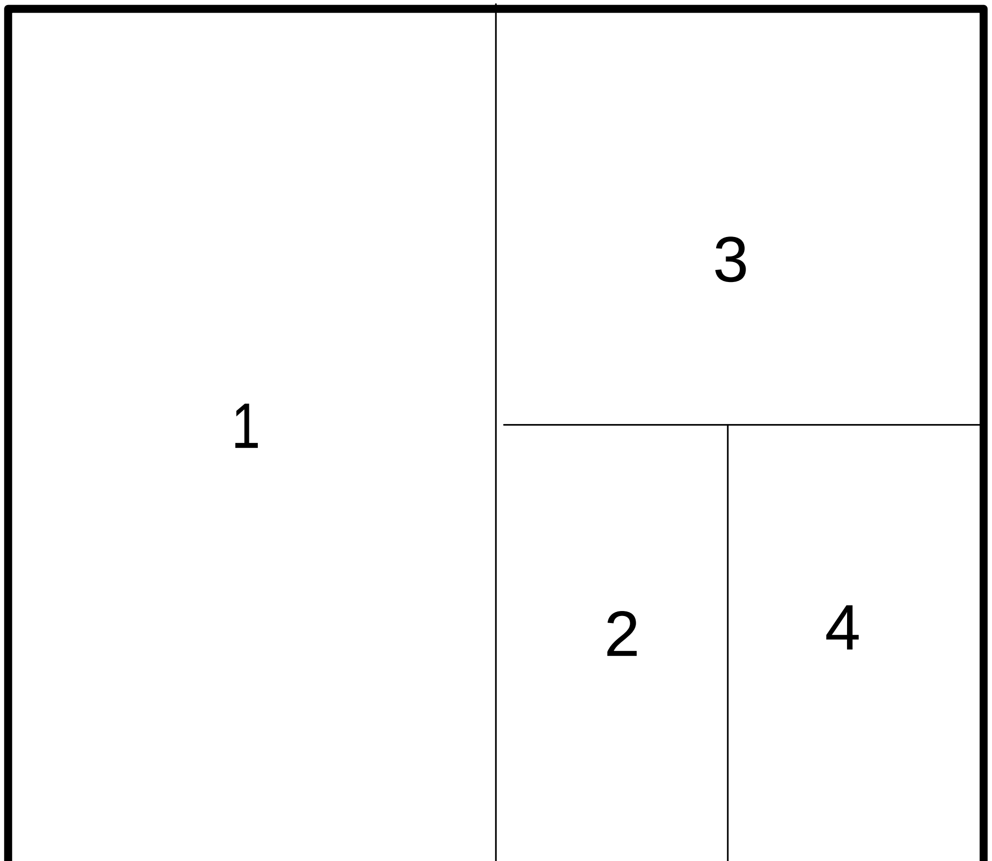
CAN: simple example



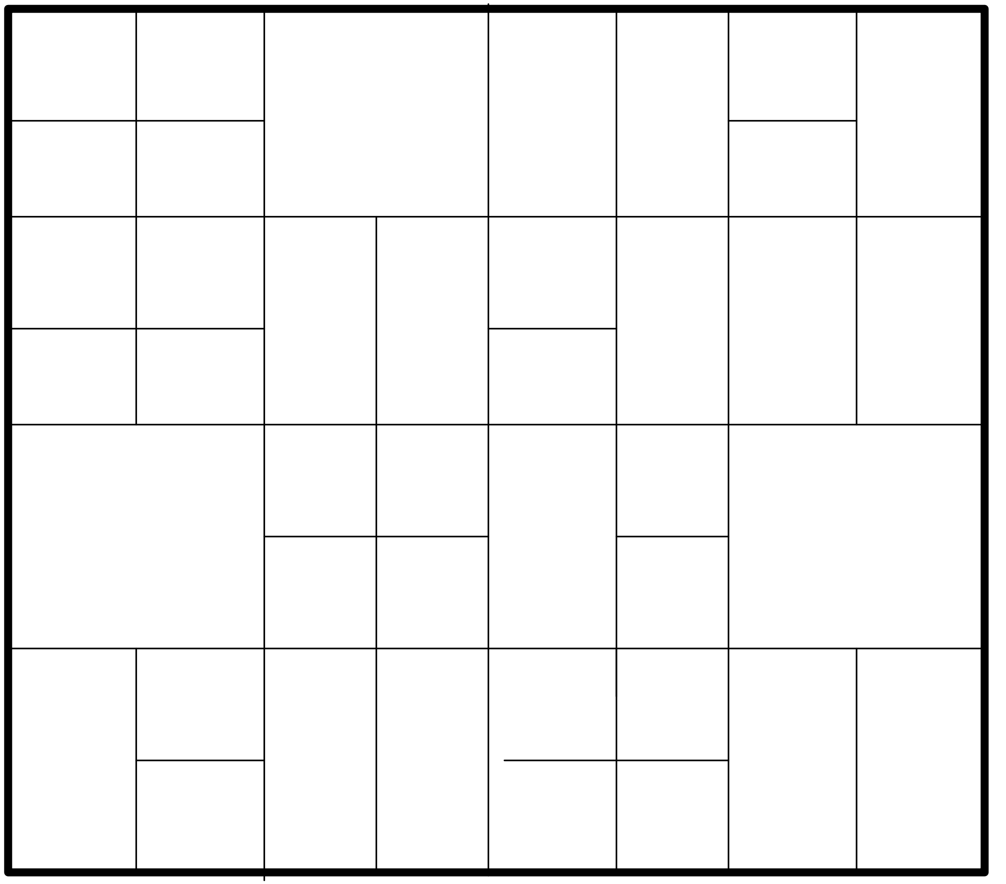
CAN: simple example



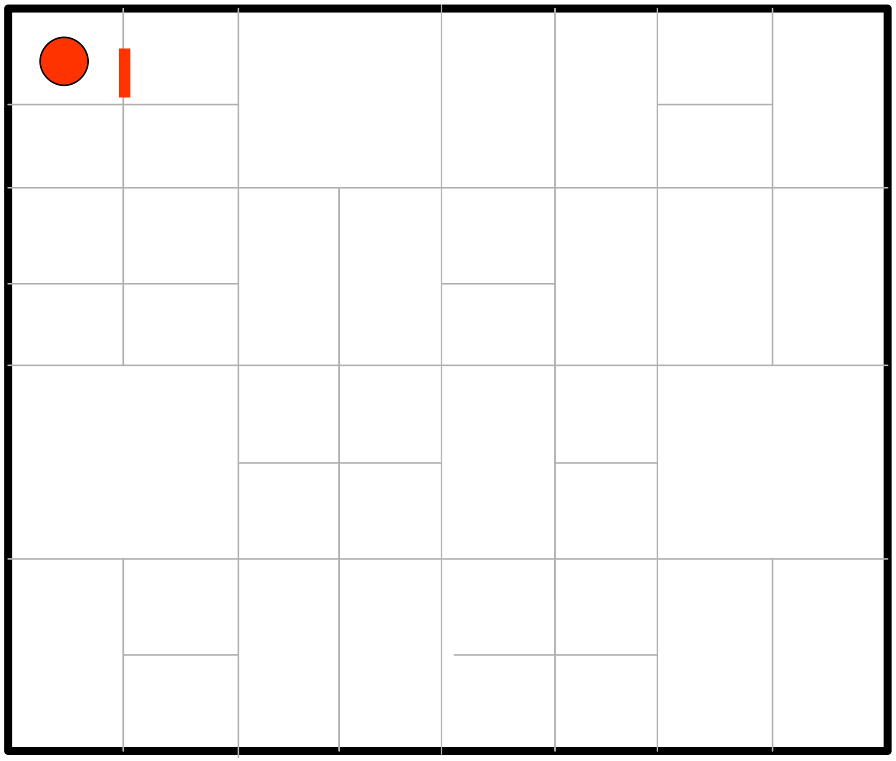
CAN: simple example



CAN: simple example

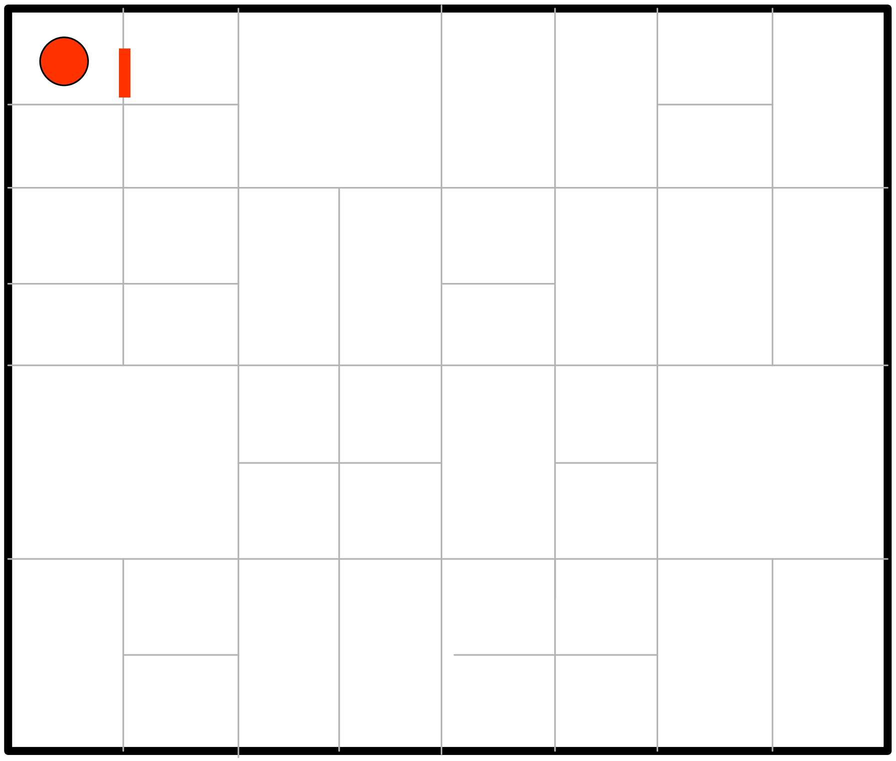


CAN: simple example



CAN: simple example

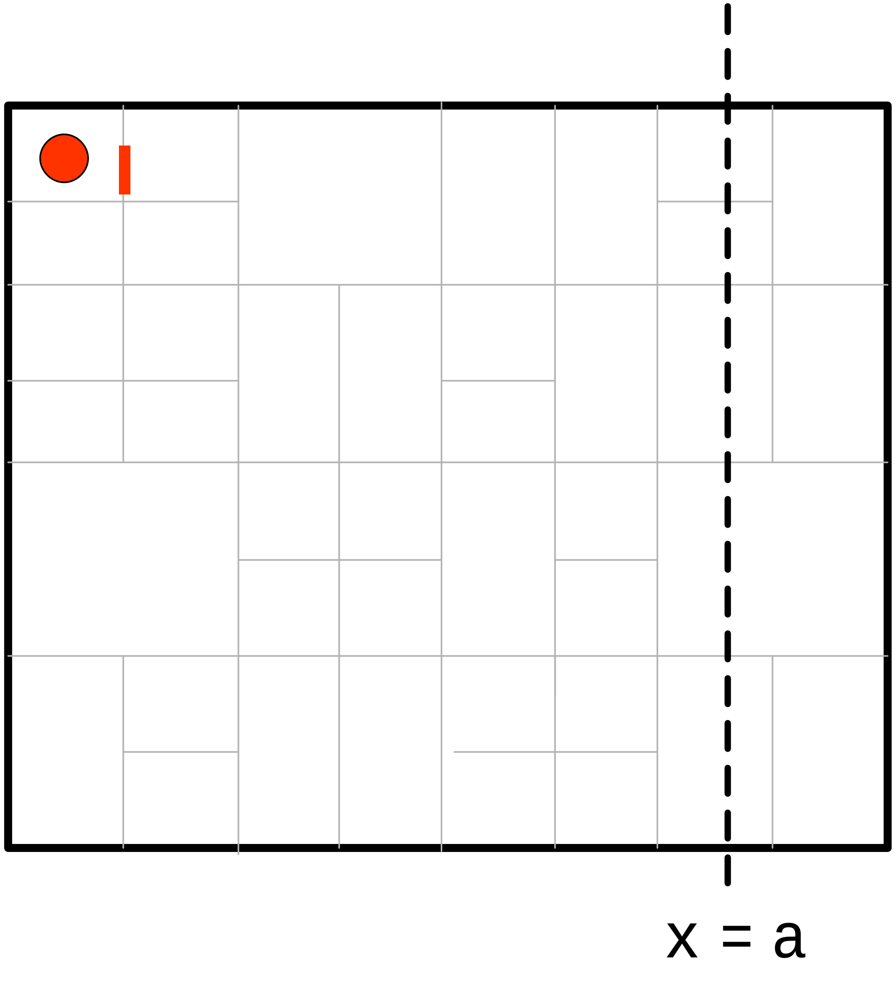
node I::insert(K,V)



CAN: simple example

node I::insert(K,V)

(1) $a = h_x(K)$

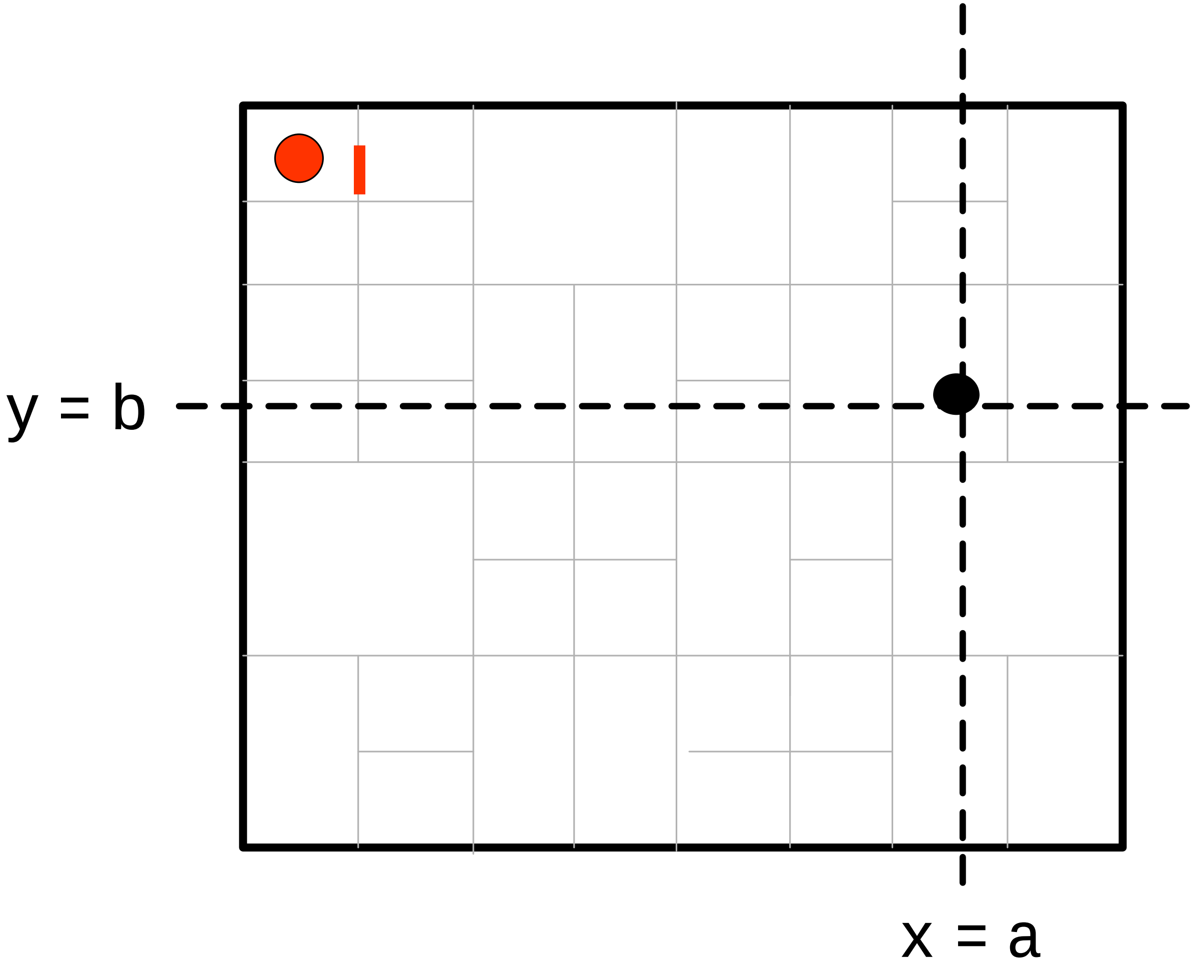


CAN: simple example

node I::insert(K,V)

(1) $a = h_x(K)$

$b = h_y(K)$



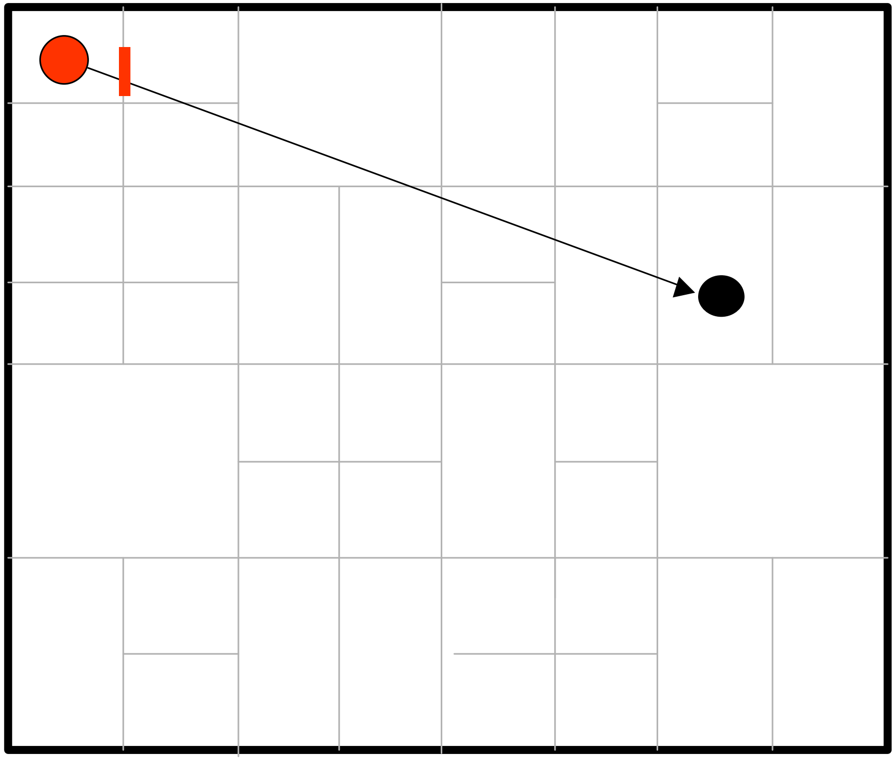
CAN: simple example

node I::insert(K,V)

(1) $a = h_x(K)$

$b = h_y(K)$

(2) route(K,V) -> (a,b)



CAN: simple example

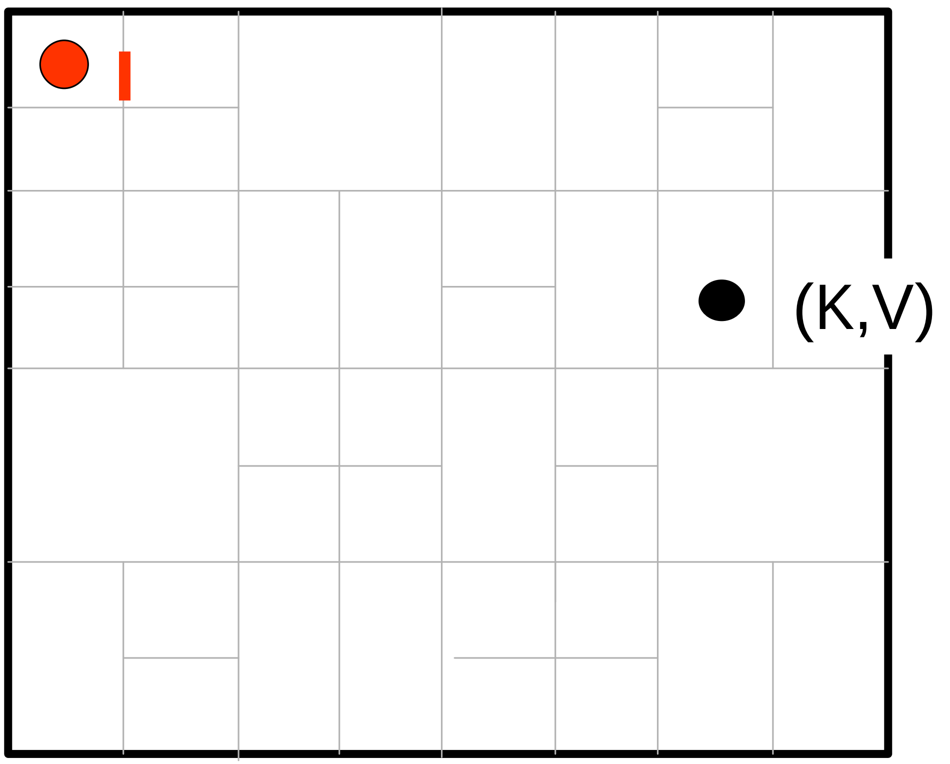
node I::insert(K,V)

(1) $a = h_x(K)$

$b = h_y(K)$

(2) route(K,V) -> (a,b)

(3) (a,b) stores (K,V)

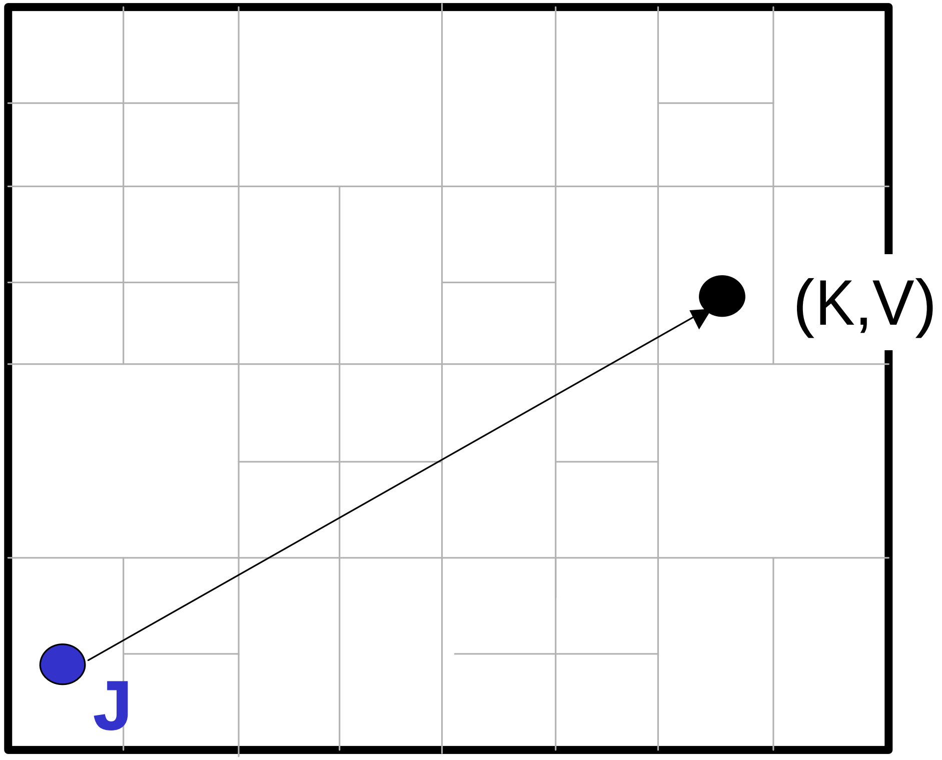


CAN: simple example

node J::retrieve(K)

(1) $a = h_x(K)$
$b = h_y(K)$

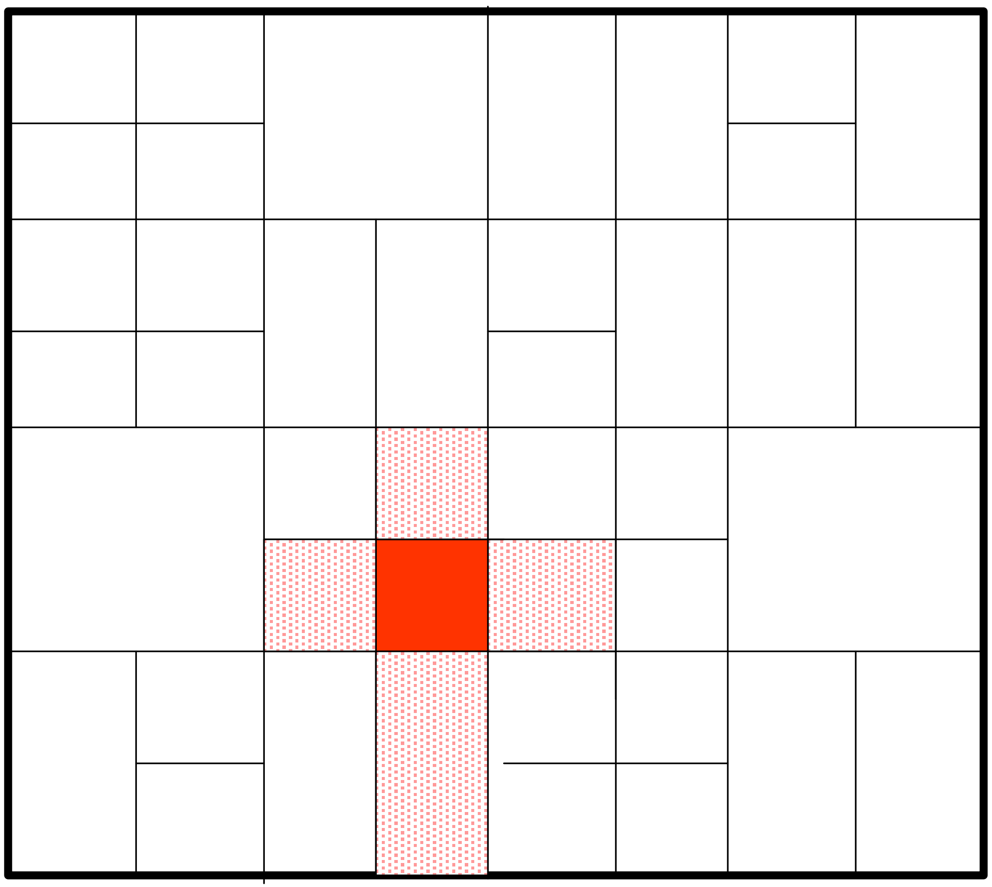
(2) route "retrieve(K)" to (a,b)



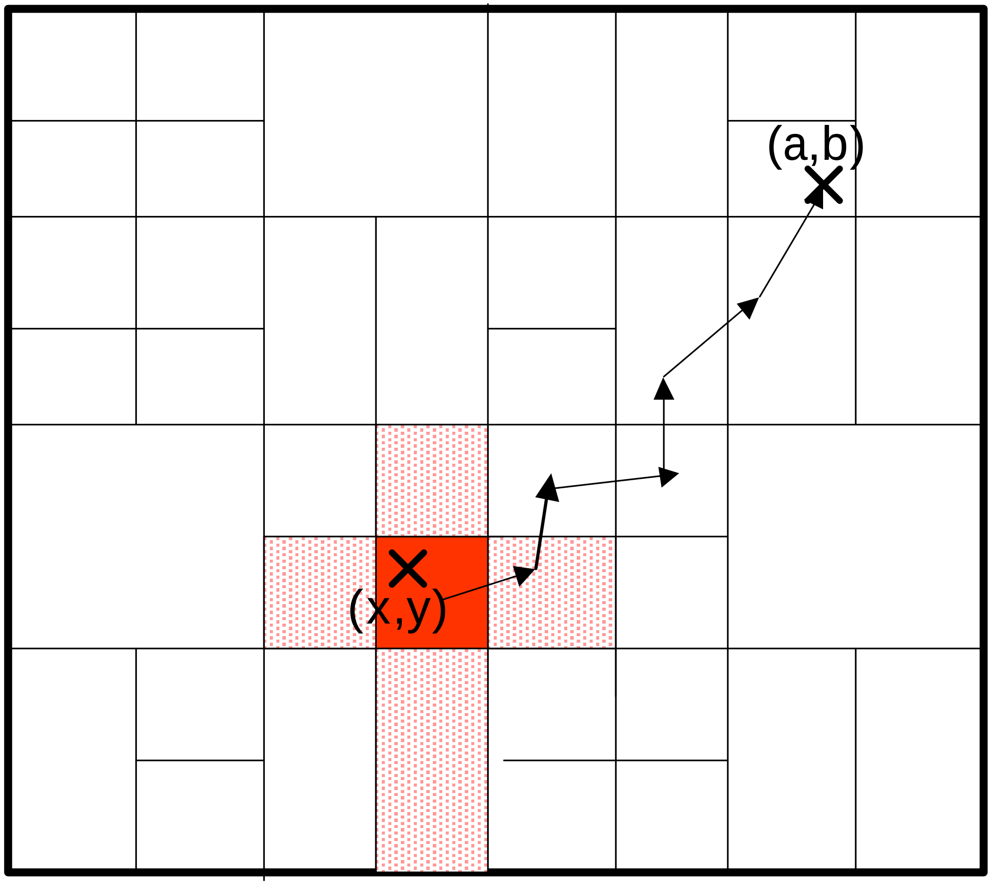
CAN

Data stored in the CAN is addressed by name (i.e. key), not location (i.e. IP address)

CAN: routing table



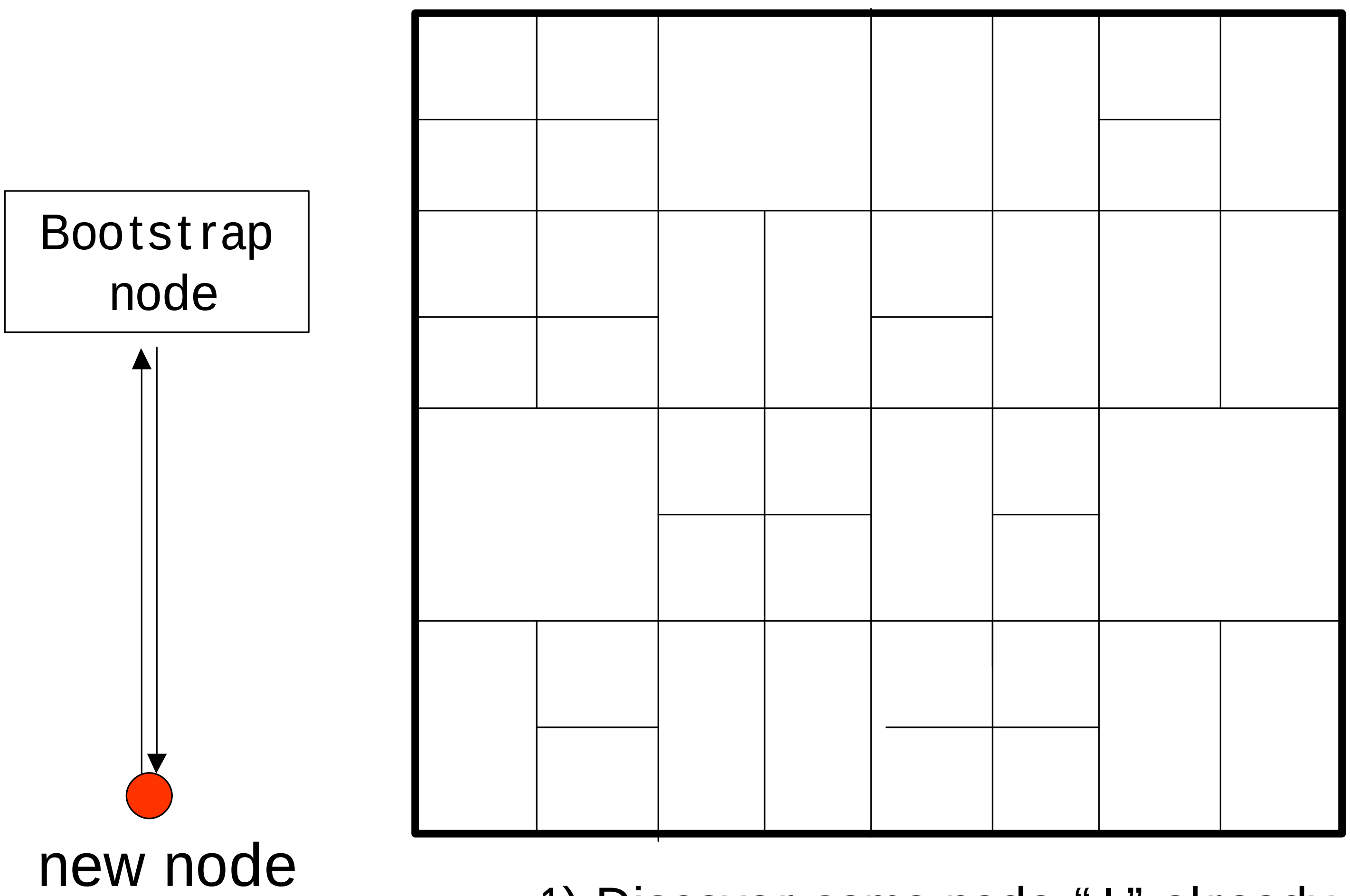
CAN: routing



CAN: routing

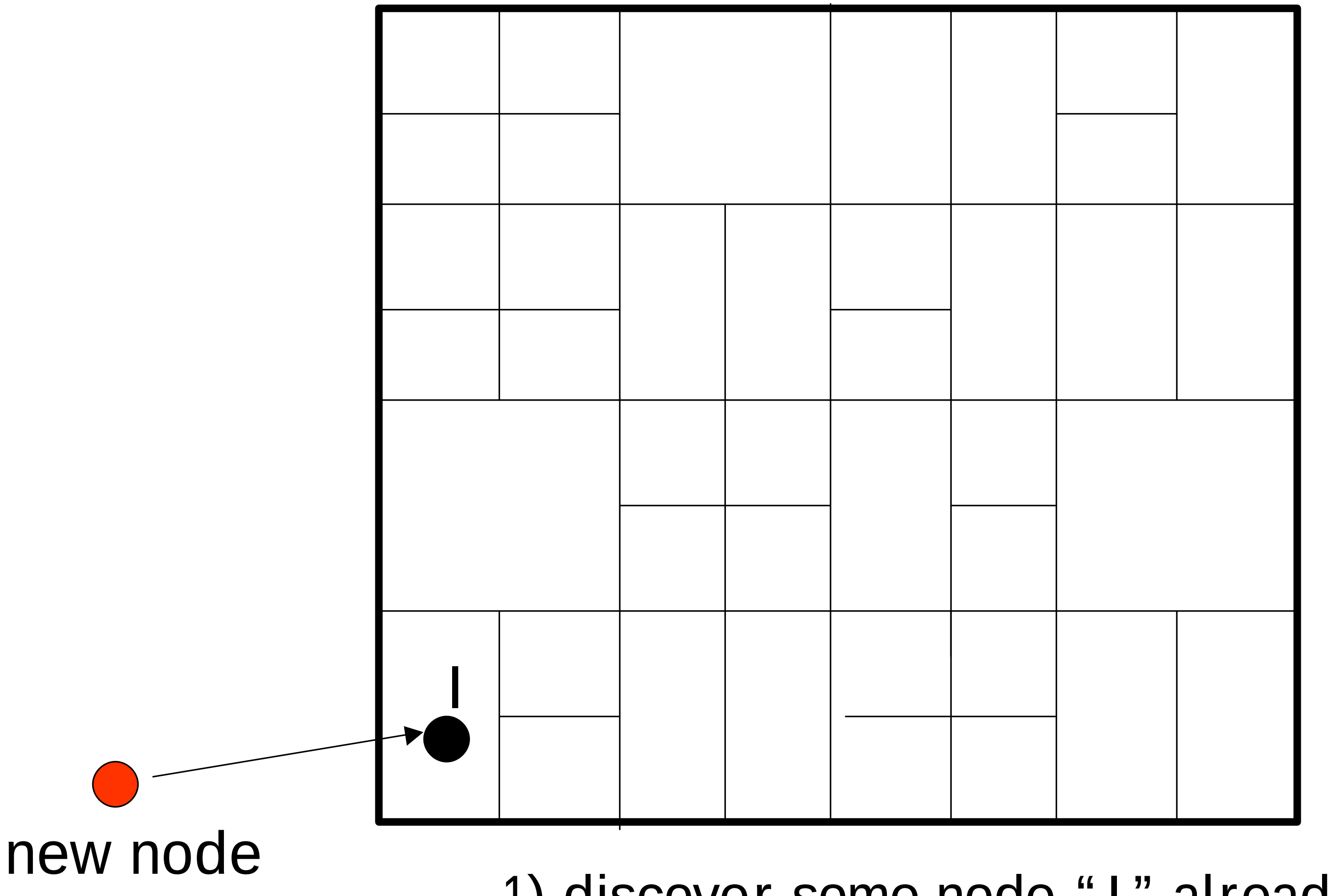
A node only maintains state for its immediate neighboring nodes

CAN: node insertion



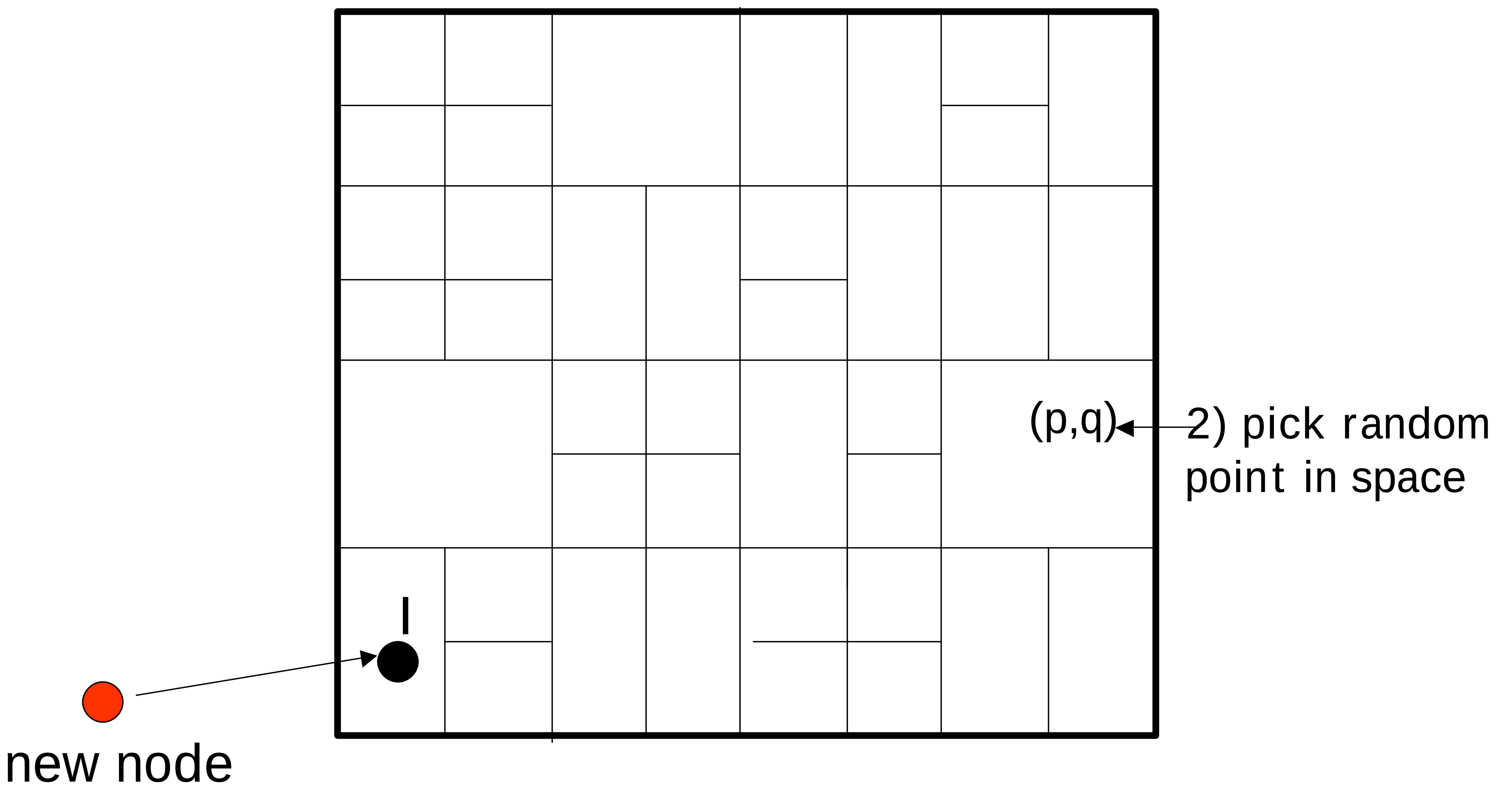
1) Discover some node “I” already in CAN

CAN: node insertion

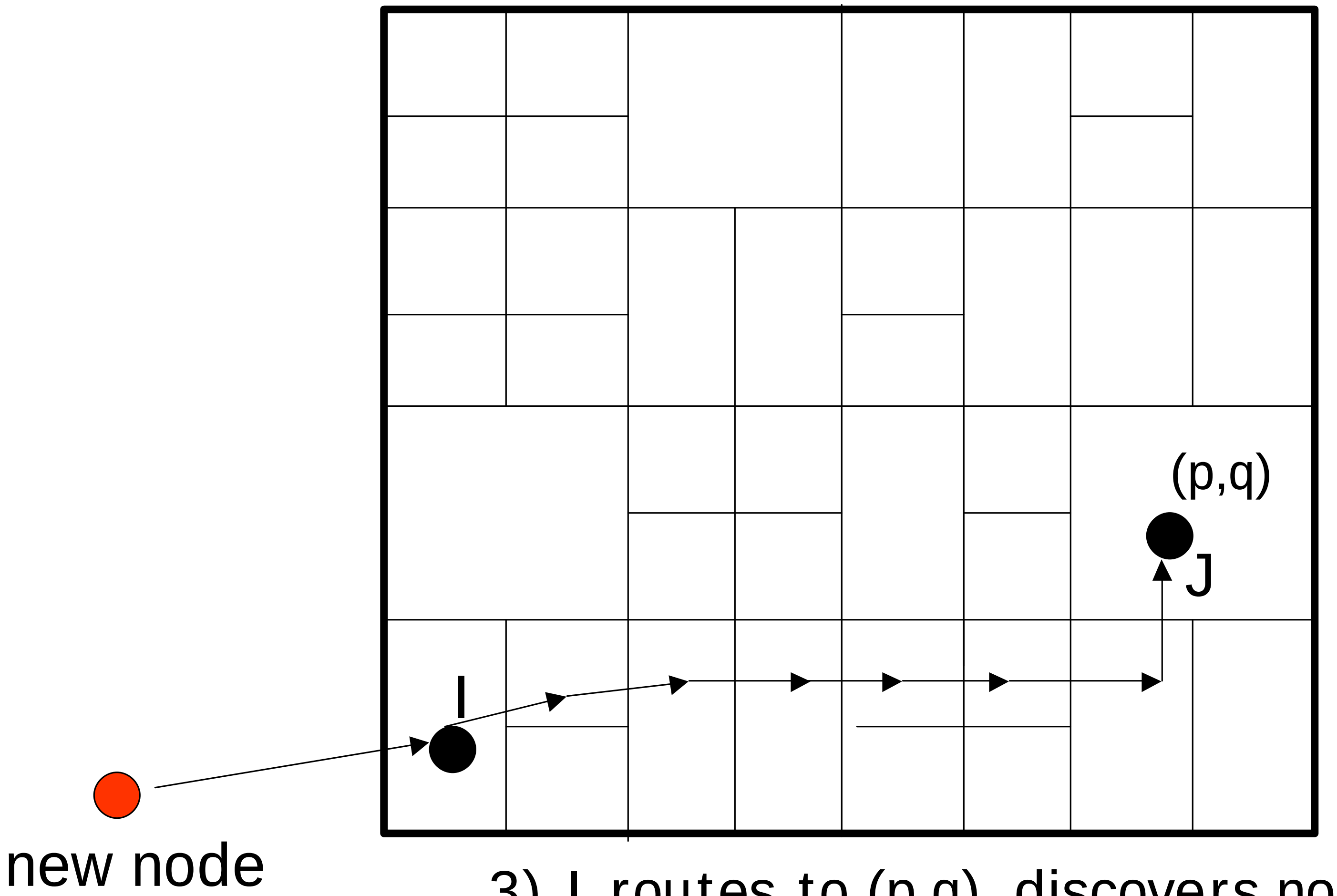


1) discover some node “ I ” already in CAN

CAN: node insertion

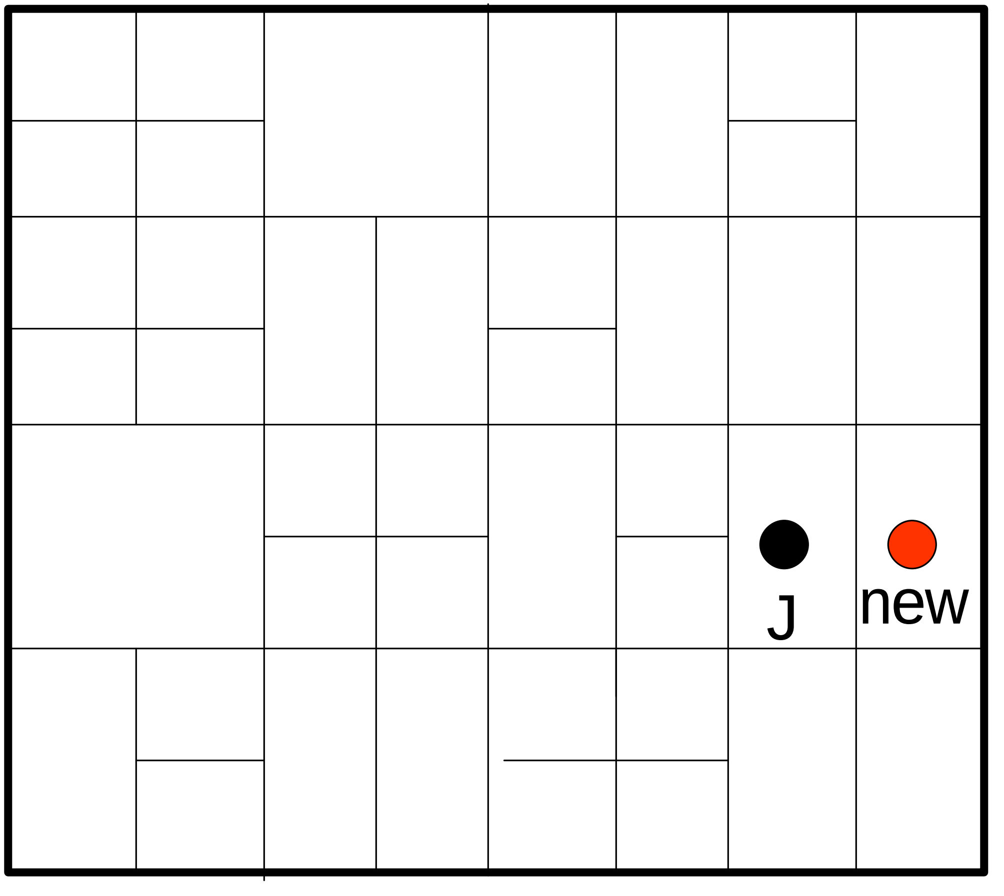


CAN: node insertion



3) I routes to (p,q), discovers node J

CAN: node insertion



4) split J's zone in half… new owns one half

CAN: node insertion

Inserting a new node affects only a single other node and its immediate neighbors

CAN: node failures

- Need to repair the space
  - recover database (weak point)
    - soft-state updates
    - use replication, rebuild database from replicas
  - repair routing
    - takeover algorithm

CAN: takeover algorithm

- Simple failures
  - know your neighbor's neighbors
  - when a node fails, one of its neighbors takes over its zone
- More complex failure modes
  - simultaneous failure of multiple adjacent nodes
  - scoped flooding to discover neighbors
  - hopefully, a rare event

CAN: node failures

Only the failed node's immediate neighbors are required for recovery

Design recap

- Basic CAN
  - completely distributed
  - self-organizing
  - nodes only maintain state for their immediate neighbors

- Additional design features
  - multiple, independent spaces (realities)
  - background load balancing algorithm
  - simple heuristics to improve performance

Outline

- Introduction
- Design
- Evaluation
- Strengths & Weaknesses
- Ongoing Work

Evaluation

- Scalability
- Low-latency
- Load balancing
- Robustness

CAN: scalability

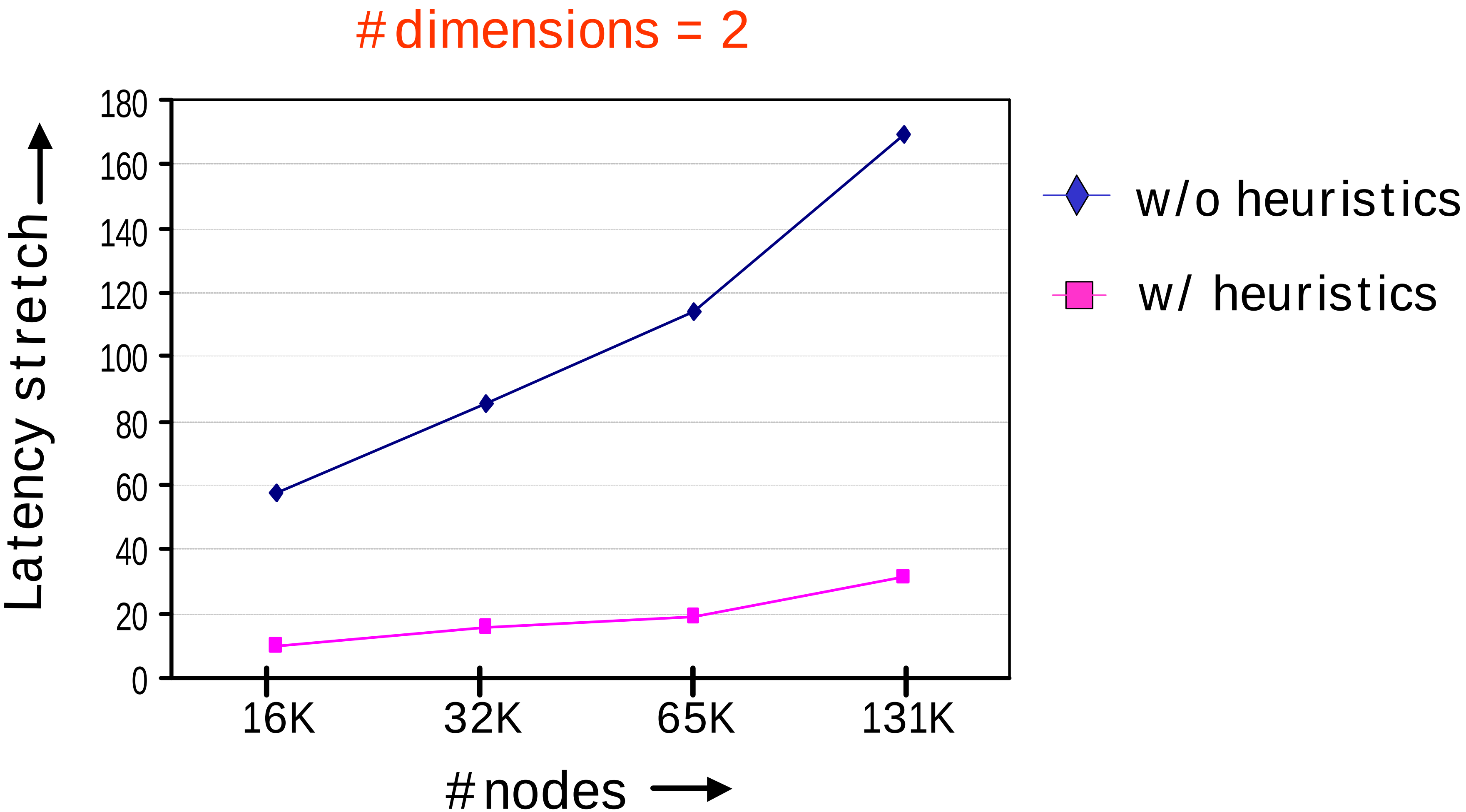
- For a uniformly partitioned space with **n** nodes and **d** dimensions
  - per node, number of neighbors is 2d
  - average routing path is $(dn^{1/d})/4$ hops
  - simulations show that the above results hold in practice
- Can scale the network without increasing per-node state
- Chord/Plaxton/Tapestry/Buzz
  - log(n) nbrs with log(n) hops

CAN: low-latency

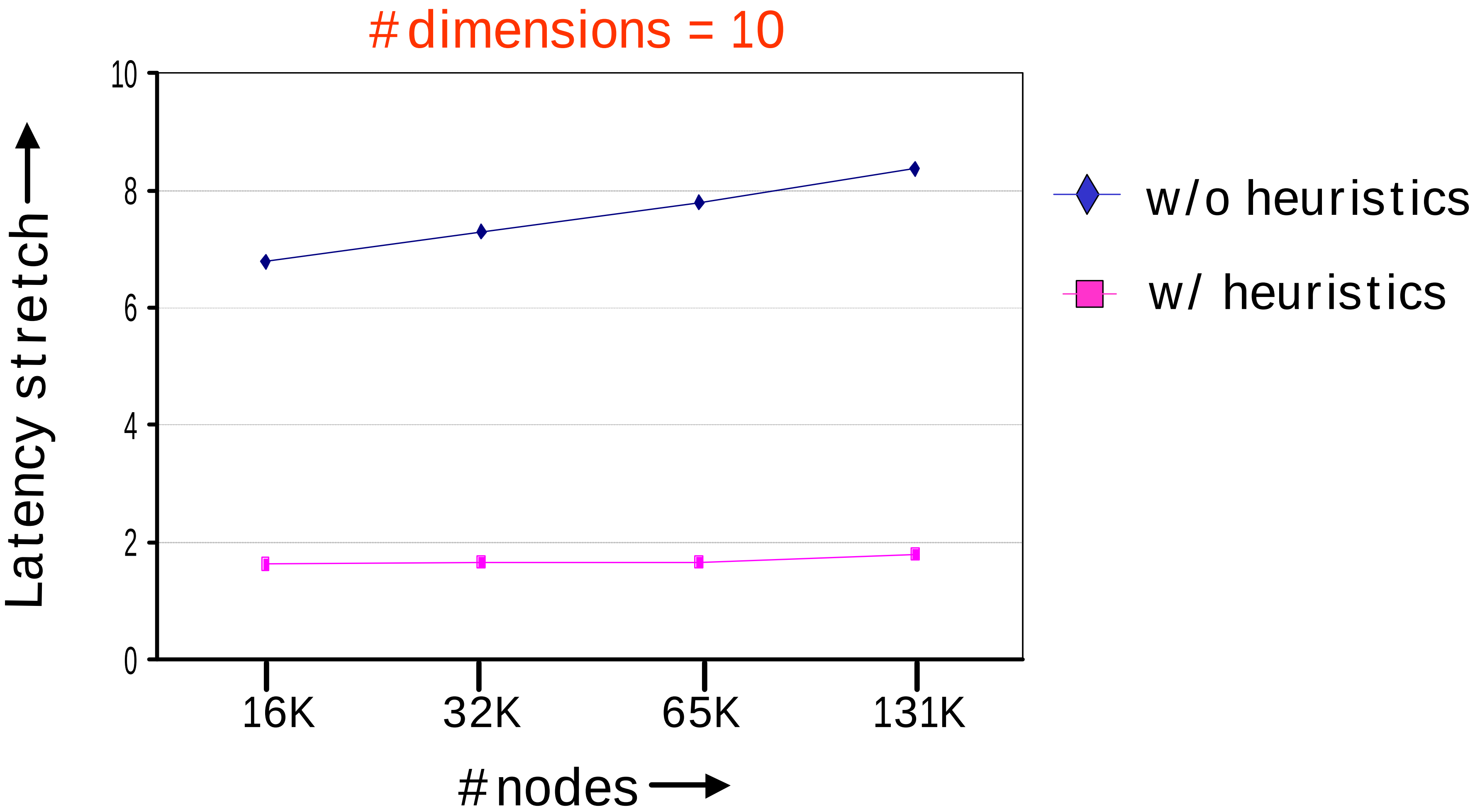
- Problem
  - latency stretch = $\frac{\text{(CAN routing delay)}}{\text{(IP routing delay)}}$
  - application-level routing may lead to high stretch

- Solution
  - increase dimensions, realities (reduce the path length)
  - Heuristics (reduce the per-CAN-hop latency)
    - RTT-weighted routing
    - multiple nodes per zone (peer nodes)
    - deterministically replicate entries

CAN: low-latency



CAN: low-latency



CAN: load balancing

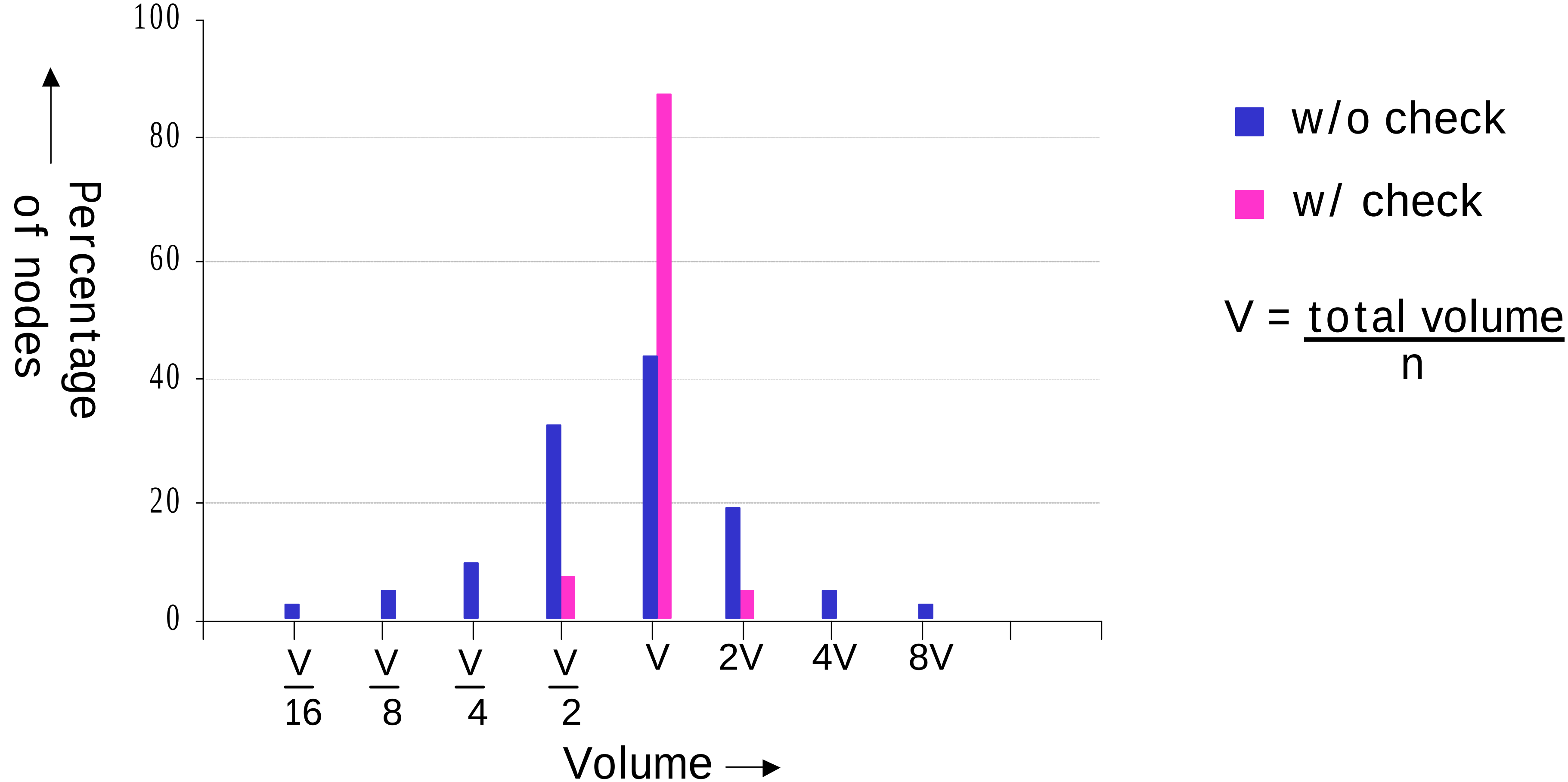
- Two pieces
  - Dealing with hot-spots
    - popular (key,value) pairs
    - nodes cache recently requested entries
    - overloaded node replicates popular entries at neighbors
  - Uniform coordinate space partitioning
    - uniformly spread (key,value) entries
    - uniformly spread out routing load

Uniform Partitioning

- Added check
  - at join time, pick a zone
  - check neighboring zones
  - pick the largest zone and split that one

Uniform Partitioning

65,000 nodes, 3 dimensions



CAN: Robustness

- Completely distributed
  - no single point of failure (not applicable to pieces of database when node failure happens)
- Not exploring database recovery (in case there are multiple copies of database)
- Resilience of routing
  - can route around trouble

Outline

- Introduction
- Design
- Evaluation
- Strengths & Weaknesses
- Ongoing Work

Strengths

- More resilient than flooding broadcast networks
- Efficient at locating information
- Fault tolerant routing
- Node & Data High Availability (w/ improvement)
- Manageable routing table size & network traffic

Weaknesses

- Impossible to perform a fuzzy search
- Susceptible to malicious activity
- Maintain coherence of all the indexed data (Network overhead, Efficient distribution)
- Still relatively higher routing latency
- Poor performance w/o improvement

Suggestions

- Catalog and Meta indexes to perform search function
- Extension to handle mutable content efficiently for web-hosting
- Security mechanism to defense against attacks

Outline

- Introduction
- Design
- Evaluation
- Strengths & Weaknesses
- Ongoing Work

Ongoing Work

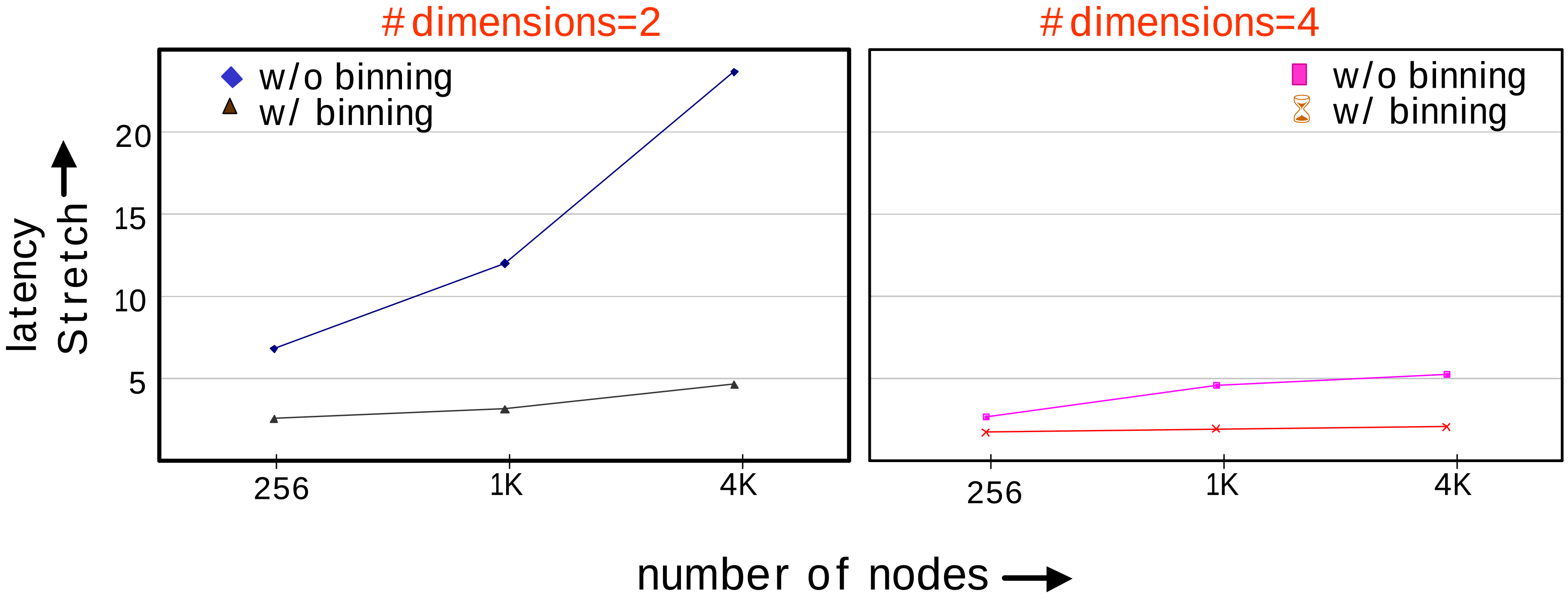
- Topologically-sensitive CAN construction
  - distributed binning

Distributed Binning

- Goal
  - bin nodes such that co-located nodes land in same bin
- Idea
  - well known set of landmark machines
  - each CAN node, measures its RTT to each landmark
  - orders the landmarks in order of increasing RTT
- CAN construction
  - place nodes from the same bin close together on the CAN

Distributed Binning

- 4 Landmarks (placed at 5 hops away from each other)
- naïve partitioning



Ongoing Work (cont'd)

- Topologically-sensitive CAN construction
  - distributed binning
- CAN Security (Petros Maniatis - Stanford)
  - spectrum of attacks
  - appropriate counter-measures

Ongoing Work (cont'd)

- CAN Usage
  - Application-level Multicast (NGC 2001)
  - Grass-Roots Content Distribution
  - Distributed Databases using CANs
    (J.Hellerstein, S.Ratnasamy, S.Shenker, I.Stoica, S.Zhuang)

Summary

- CAN
  - an Internet-scale hash table
  - potential building block in Internet applications
- Scalability
  - O(d) per-node state
- Low-latency routing
  - simple heuristics help a lot
- Robust
  - decentralized, can route around trouble