Multi-Attribute Range Queries on Read-Only DHT

Verdi March, Yong Meng Teo

Department of Computer Science
National University of Singapore
Email: [verdimar,teoym]@comp.nus.edu.sg

Outline

- Introduction to R-DHT
- Problem Statement
- Related Works
- Midas
  - Indexing
  - Range-Query Optimizations
- Analysis
- Conclusion

Introduction

- Goal: provide *lookup* service in large distributed systems with minimum dependency to a 3rd-party infrastructure
  - Effective : result guarantee (minimize false negative)
  - Efficient : short bounded lookup path length, scalable to #nodes

- DHT : distributed implementation of hash-table abstractions, i.e. ‹key, value›, *get*(key), and *put*(key, value)
  - Distributed file system (CFS, PAST)
  - Multicast (Scribe)
  - RSS distribution (Corona , FeedTree)
  - Grid resource discovery (*DGRID*, MAAN, Self-Organizing Condor, RIC, XenoSearch)

DHT Lookups

- User: lookup key *k*
- DHT: walk along a path to a certain direction

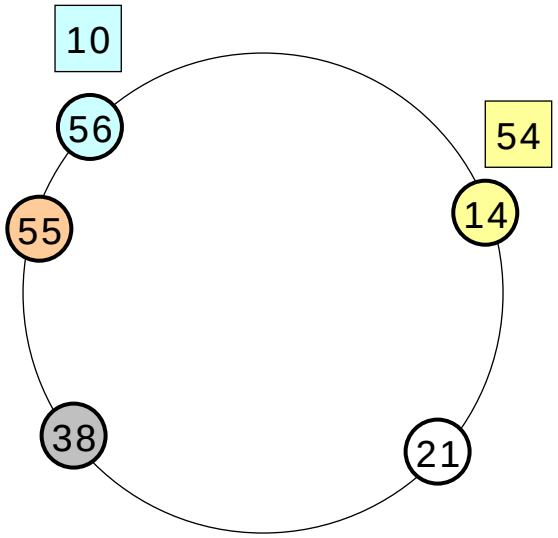
- User: I’ve walk 10 steps, and I haven’t see *k*
- DHT: Continue 10 steps.

- …

- User: I’ve been walking for a total of 50 steps
- DHT: Look around. If *k* is not around, then *k* does not exist

DHT Concepts

Data items are distributed across the overlay network, and this is controlled by the hash function.



Nodes under different adm. domain (e.g. commercial organization):

- Ownership, don't proactively "push" data
- Self-interest to protect investment

- Map keys to nodes. Keys (and values) are stored to the responsible nodes
  - Node = *bucket*
  - Locating a key is equals to locating the responsible node

Higher result guarantee

- Structured overlay network: *topology* + *nodes ordering*
  - Routing to a node in short bounded path length
  - Node maintains a small number of routing states

Scalability

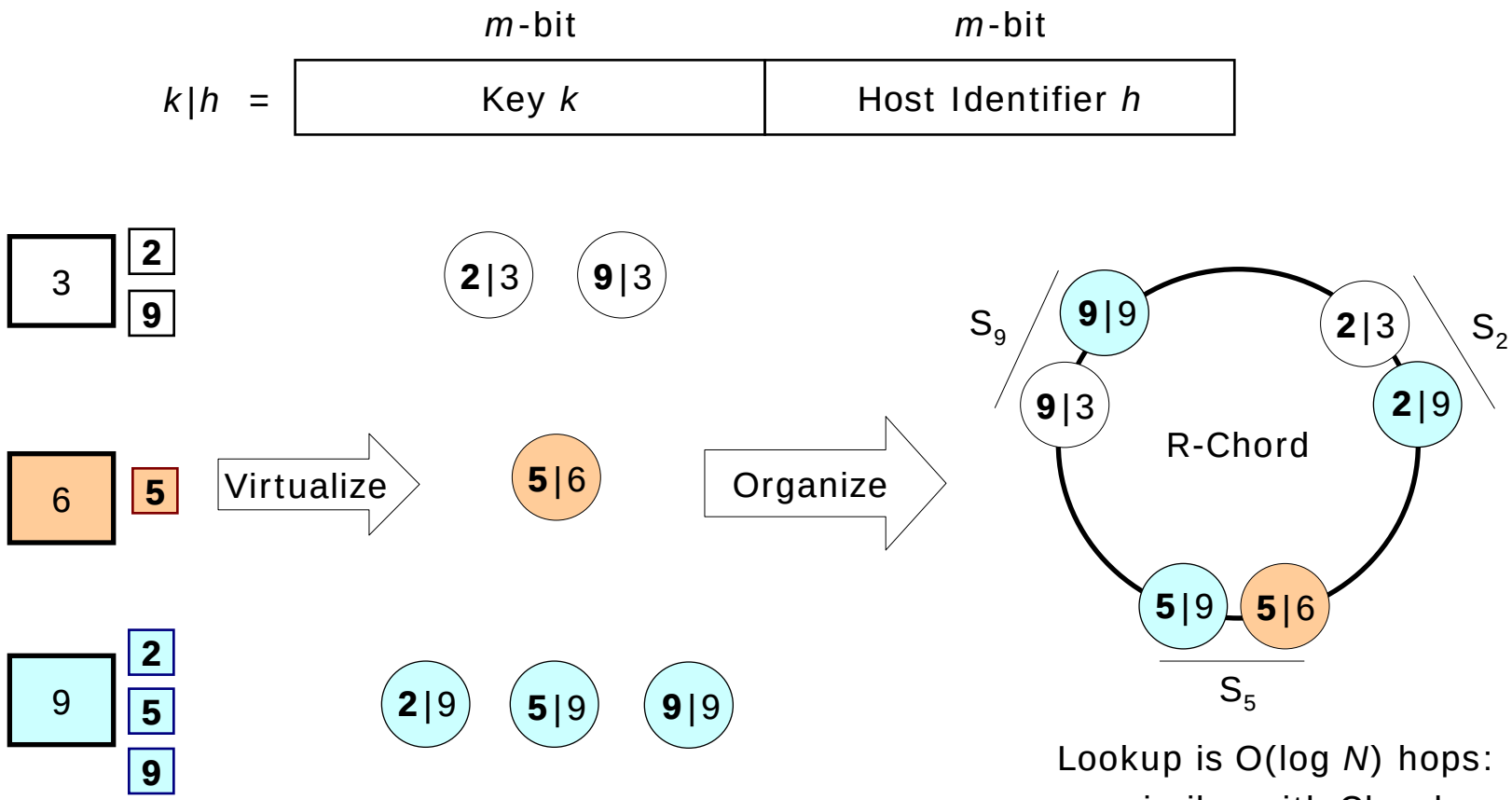
R-DHT Framework

- A class of DHT
  - Framework to turn existing DHT into a read-only version

Hash-Table Abstraction	Conventional DHT	R-DHT
Store	Yes	No
Lookup	Yes	Yes

- No distribution of key-value pairs
  - Each node stores only its own key-value pairs (data items)
- Keys are mapped to their original location

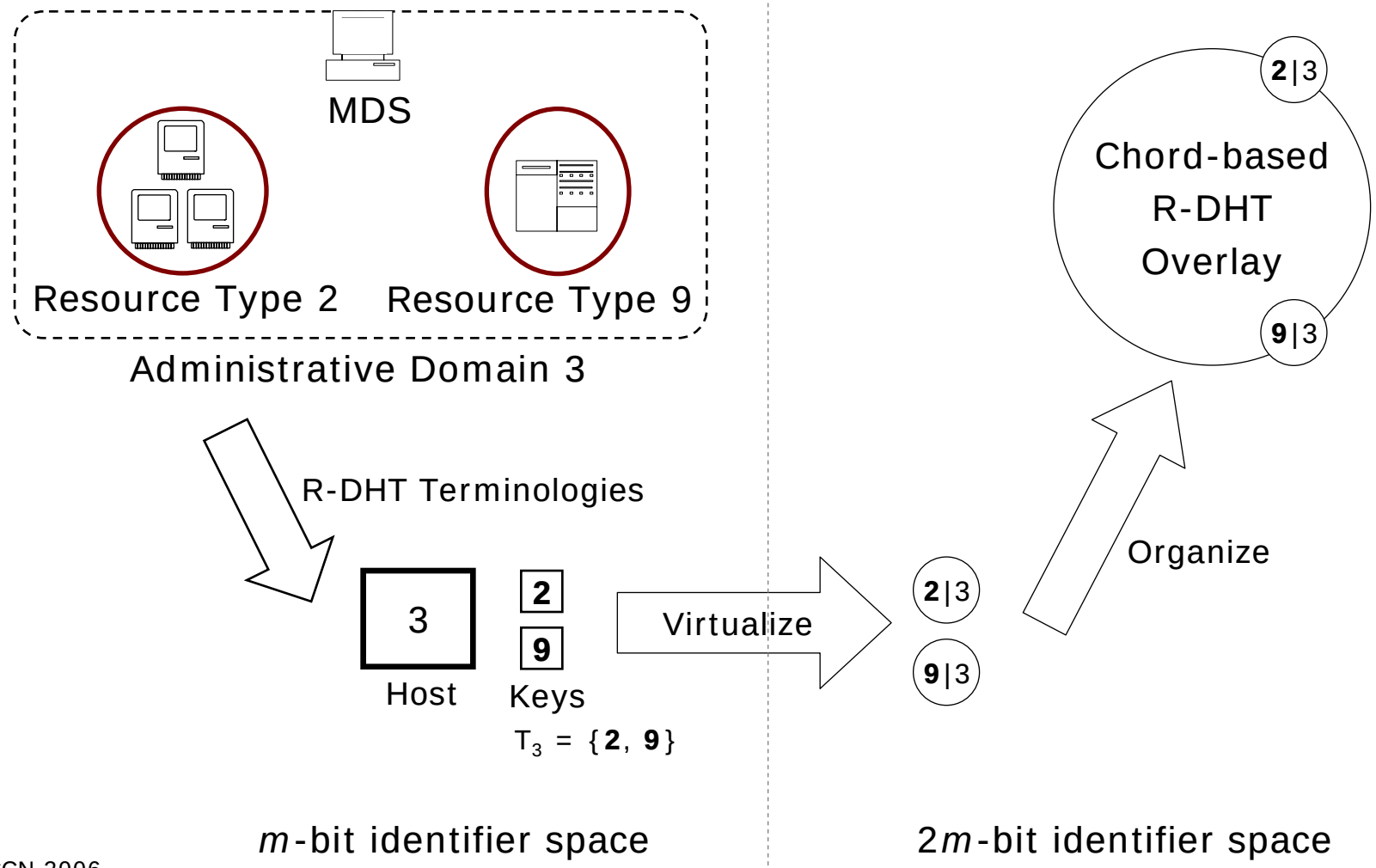
R-DHT



Lookup is O(log *N*) hops:

- similar with Chord
- N = #hosts

R-DHT Example



Outline

- Introduction to R-DHT
- **Problem Statement**
- Related Works
- Midas
  - Indexing
  - Range-Query Optimizations
- Analysis
- Conclusion

Multi-Attribute Resources

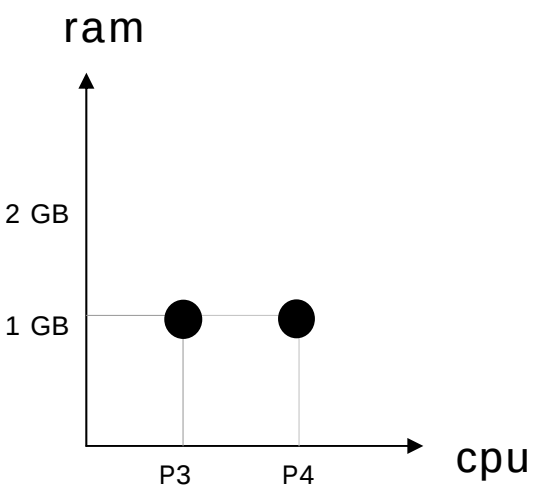
- Basic lookup operation in DHT supports only exact queries
  - lookup(3) to search resource type 3

- Ongoing research for efficient multi-attribute range queries in DHT
  - Resource type is described by *d* attributes: `cpu` and `ram`
  - A multi-attribute range query:
    - Find resources where {`cpu`=*, ‘1 GB’ ≤ `ram` ≤ ‘2 GB’}

Modeling Multi-Attribute Resource

- We index resources by their type (the *d* attributes)

- *d*-attribute resource type
  - *d*-dimensional attribute space
  - Dimension : attribute
  - Point : resource type (≥ 1 resource instances)



2-Dimensional Attribute Space

Proposed Scheme

- Objective: efficient *searching* through *multi-dimensional indexing* on top of R-DHT to answer multi-attribute range queries
  - Find {`cpu`='P3', '1 GB' ≤ `ram` ≤ '2 GB'}
- Our approach, Midas, is based on *d-to-one* mapping scheme
  - Multi-dimensional indexing of *resource types*
  - Search strategy to efficiently retrieve answers

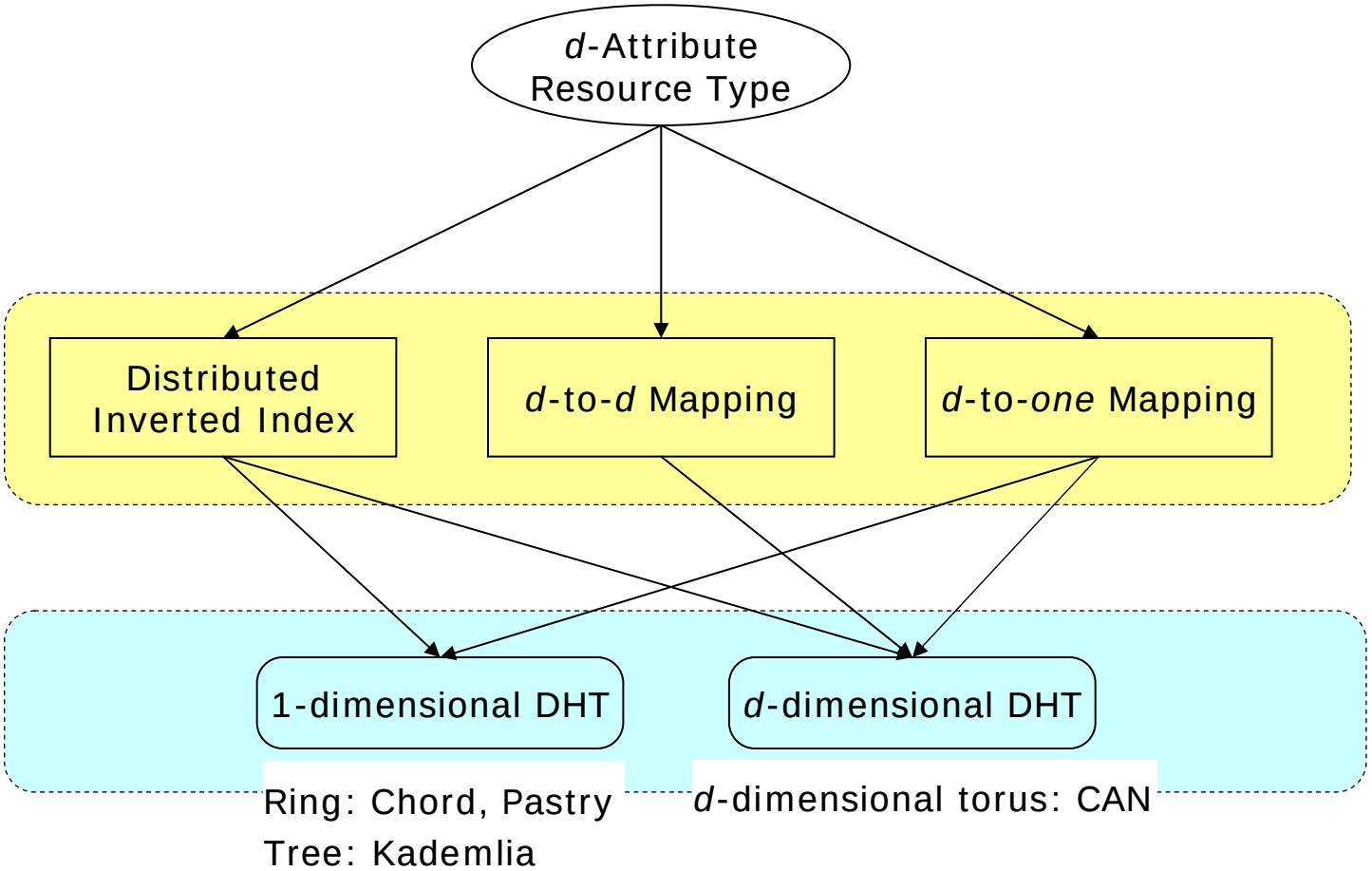
Contribution

- Midas scheme to support multi-attribute range queries on R-DHT
- Study on the implication of data-item distribution to the performance of multi-attribute range queries

Outline

- Introduction to R-DHT
- Problem Statement
- **Related Works**
- Midas
  - Indexing
  - Range-Query Optimizations
- Analysis
- Conclusion

Related Works (1)



Related Works (2)

- Distributed Inverted Index
  - MAAN (Cai et. al., 2004), CANDy (Bauer et. al., 2004), Harren 2002, KSS (Gnawali 2002), and MLP (Shi et. al., 2004)
- *d*-to-*d* Mapping
  - pSearch (Tang et. al., 2003), MURK (Ganesan et. al., 2004), and 2CAN (Agrawal et. al., 2005)
- *d*-to-*one* Mapping
  - Squid (Schmidt et. al., 2003), CONE (Agrawal et. al., 2005), ZNet (Shu et. al., 2005), SCRAP (Ganesan et. al., 2004), and CISS (Lee et. al., 2004)

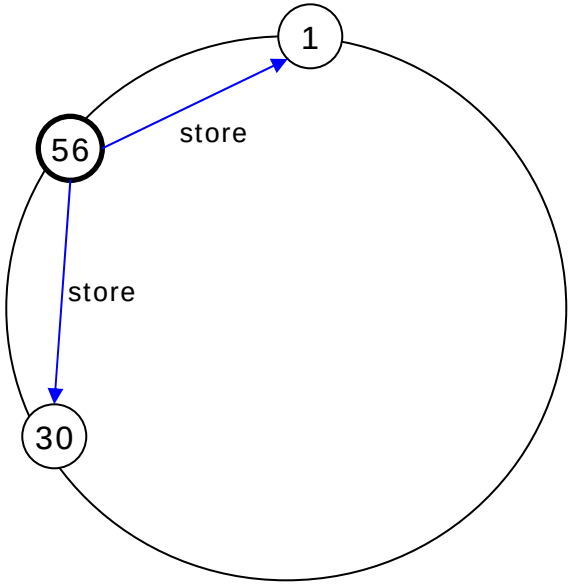
Distributed Inverted Index (1)

Resource R = {cpu='P3', ram=' 1GB'}

h('P3') = 1

h('1 GB') = 30

Order-Preserving Hashing

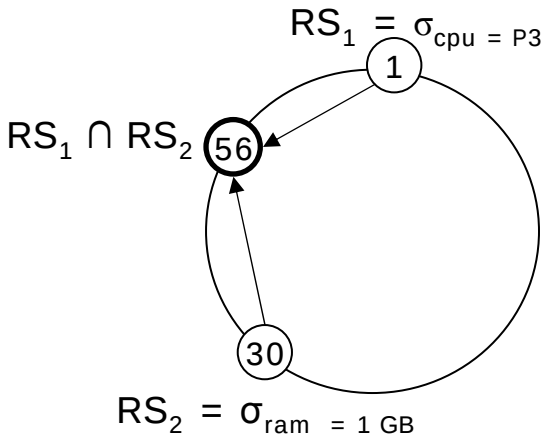


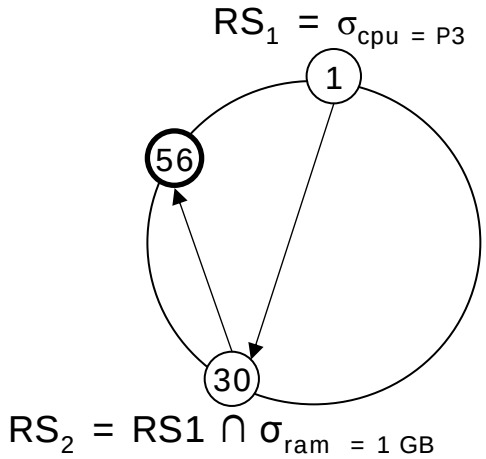
Indexing: store each key to the DHT

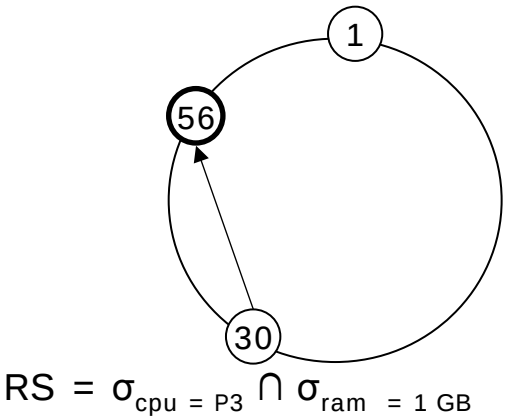
Distributed Inverted Index (2)

Find resource where {cpu='P3', ram=' 1GB'}

h('P3') = 1 | h('1 GB') = 30

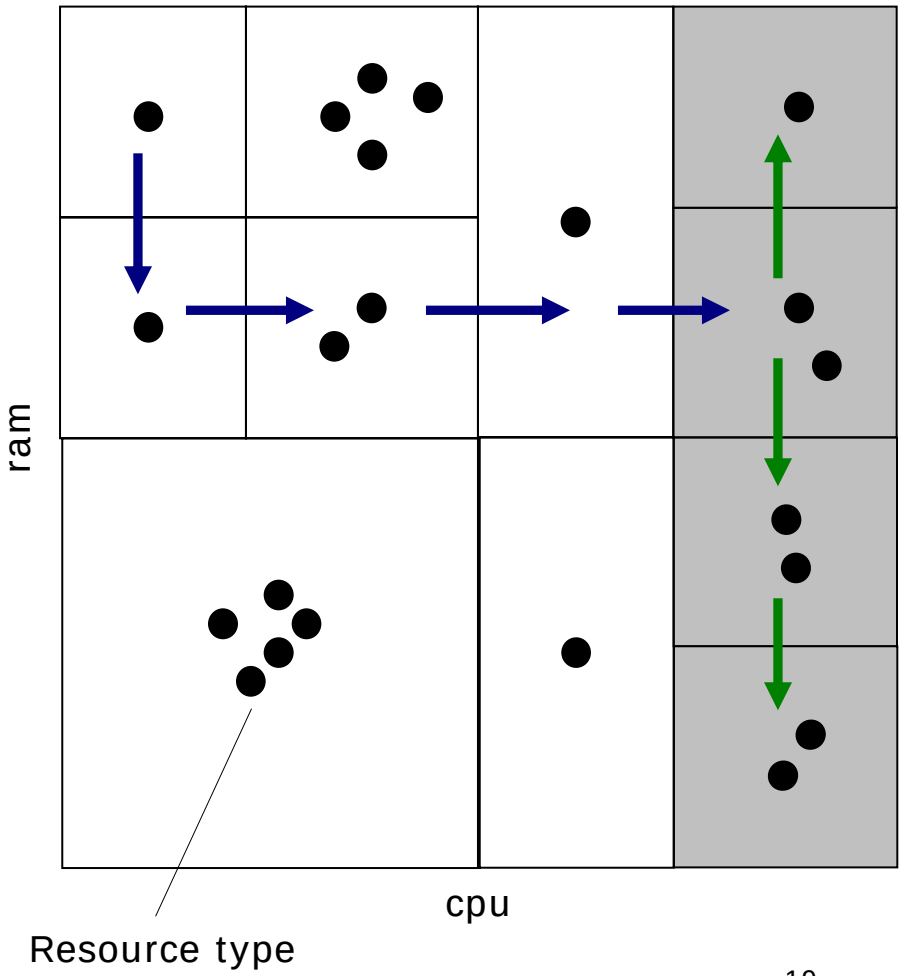




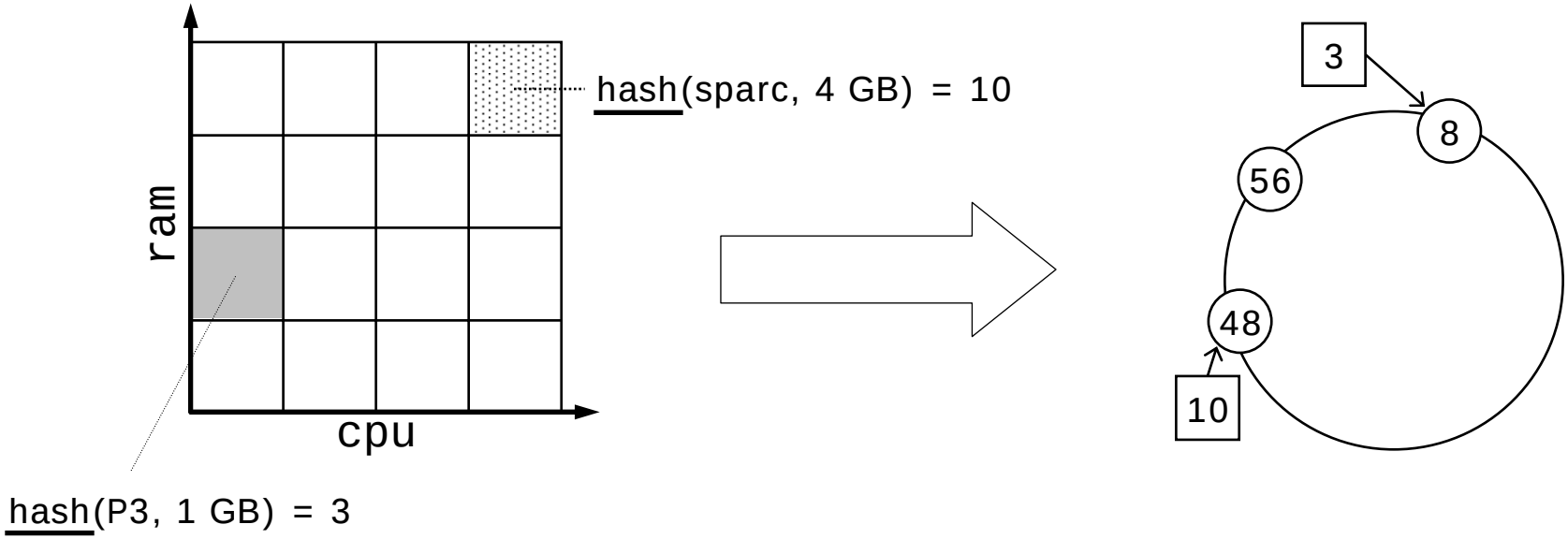


d-to-*d* Mapping

- Maps *d*-dimensional attribute space to *d*-dimensional DHT (CAN)
  - With the exception of 2CAN, which maps *d*-dimensional attribute space to 2*d*-dimensional CAN
- Range query is modeled as a region in *d*-dimensional space
- Route a search request to any point in the query region
- Flood to the remaining points in the region



d-to-*one* Mapping



Map point in *d*-dimensional space to one-dimensional key

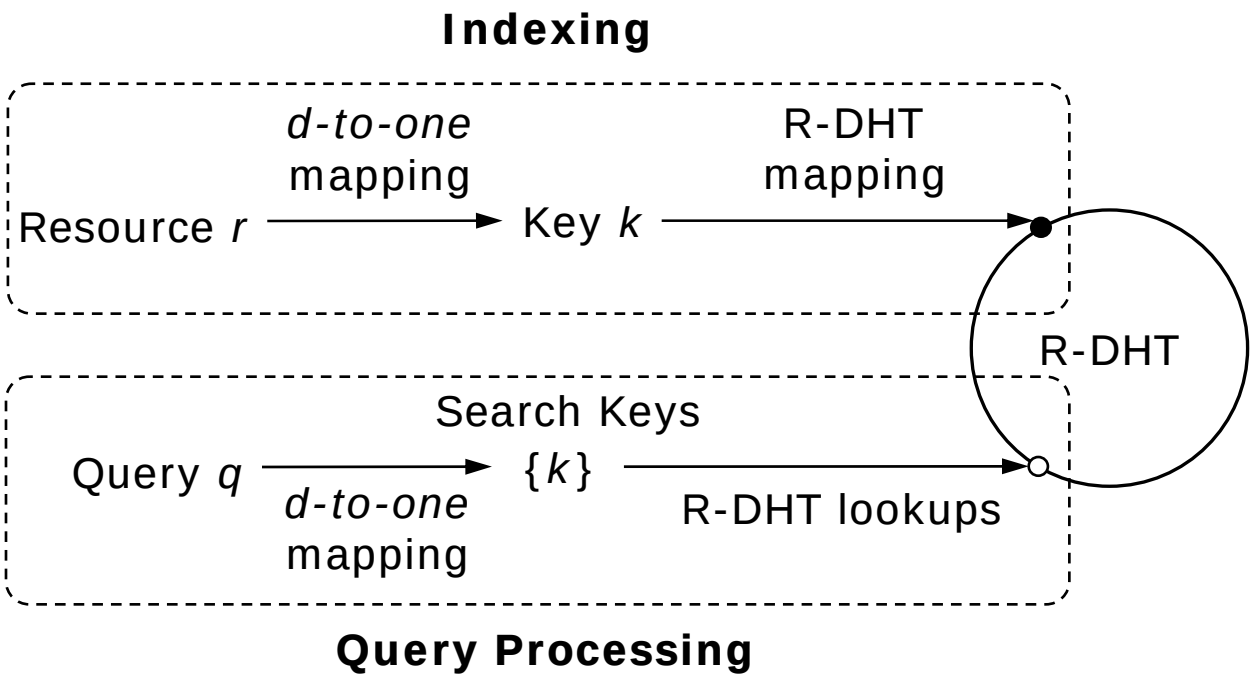
Store keys to DHT

For indexing resources and query processing

Outline

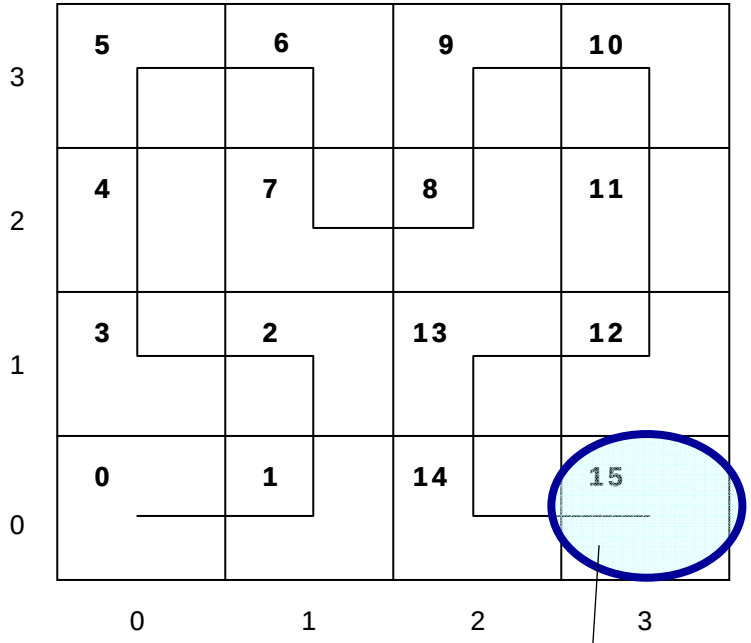
- Introduction to R-DHT
- Problem Statement
- Related Works
- **Midas**
  - **Indexing**
  - **Range-Query Optimizations**
- Analysis
- Conclusion

Midas Framework



Space-Filling Curve

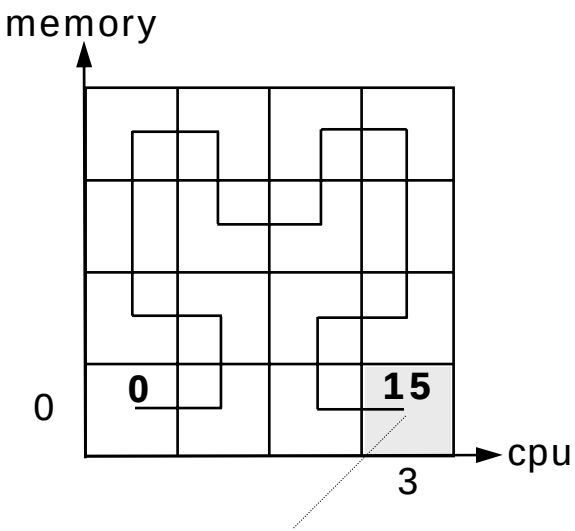
- Hilbert SFC is an example of *d*-to-*one* mapping function

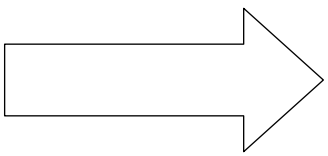


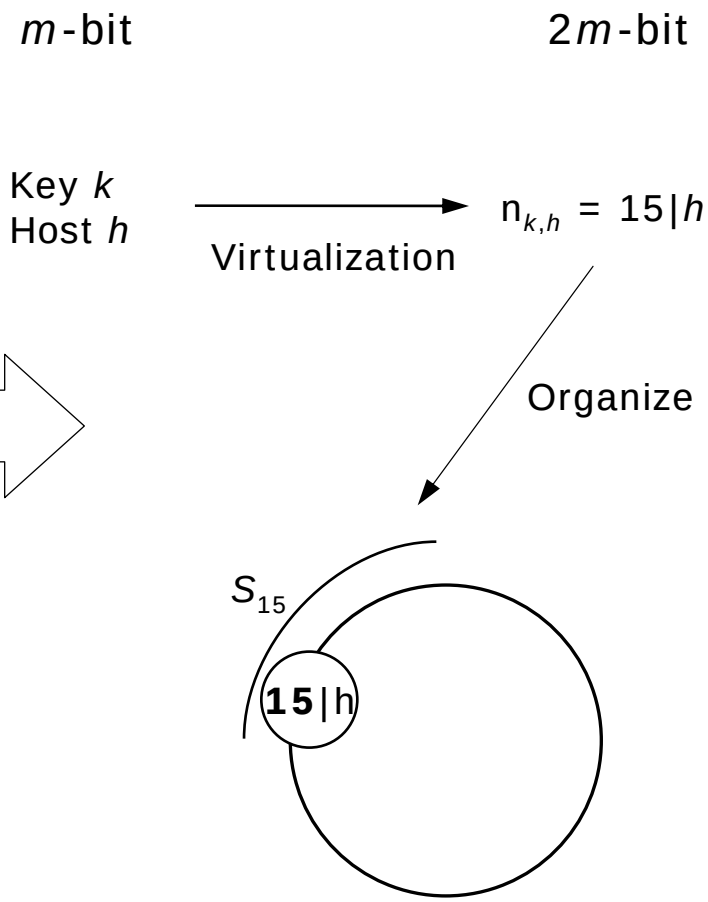
Hilbert

(3, 0) = 15

Indexing



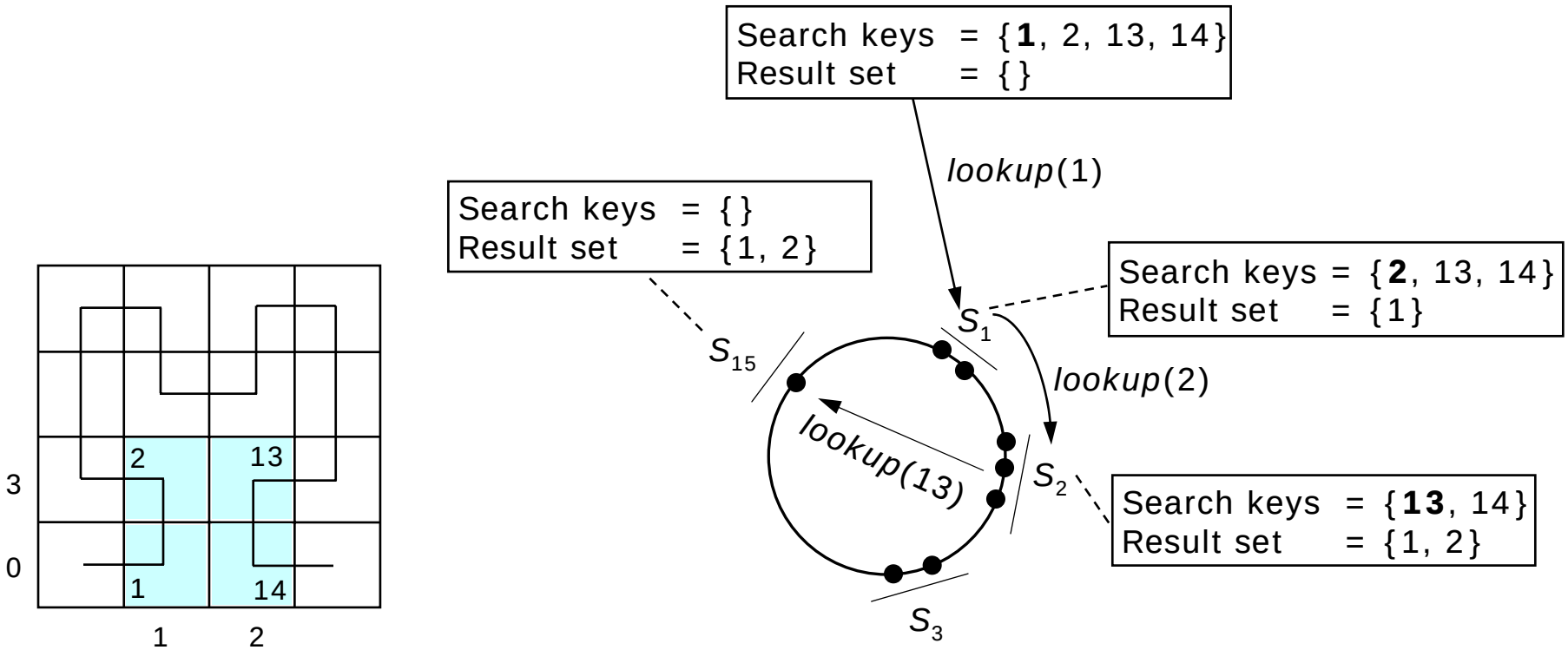




r = (cpu=*'P3'*, memory=*'1 GB'*)
= (3, 0)

k = 15

Query Processing



Outline

- Introduction to R-DHT
- Problem Statement
- Related Works
- Midas
  - Indexing
  - Range-Query Optimizations
- **Analysis**
- Conclusion

Experiment Setup

- Compare Midas on R-Chord and Chord
- Parameters
  - m = 16-bit
  - d = 3–4
  - K = 10,000–50,000
    - Keys follow normal distribution in d-dimensional space
  - N = 25,000
    - Each administrative domain has 4–10 resource types
  - Query selectivity = 1% (of 2^m)

Resiliency to Node Failures (1)

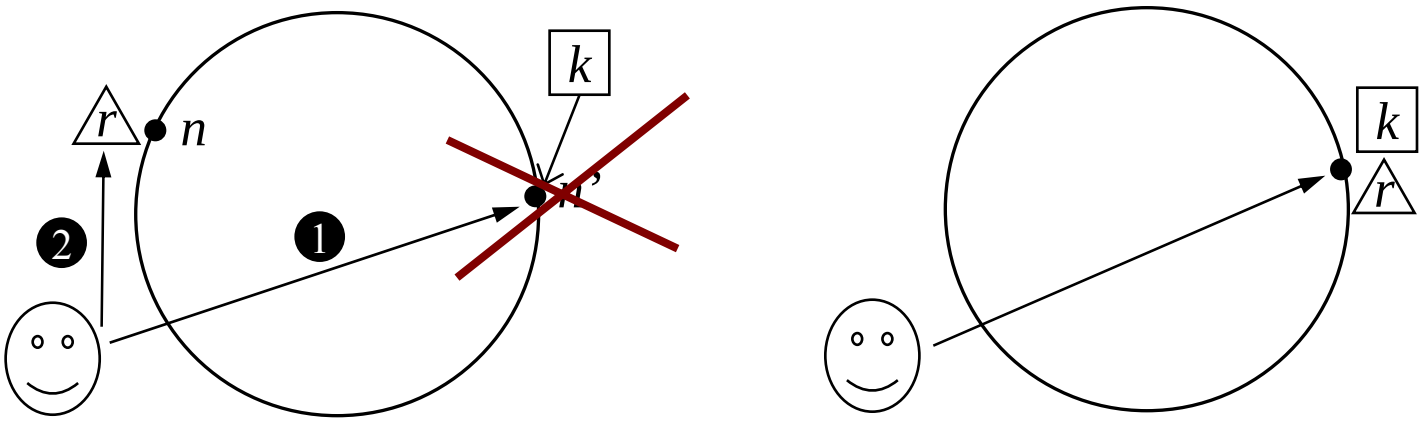
- Resiliency : ability to locate *available* resources when *FN* nodes fail simultaneously ($0 \leq F \leq 1$)
  - Resources are not replicated (i.e. we are not looking at resource *availability*)

F	K	% Keys Retrieved			
		$d = 3$		$d = 4$	
		Chord	R-Chord	Chord	R-Chord
20%	1,000	73	100	76	100
	10,000	75	100	76	100
	50,000	78	99	77	100
40%	1,000	45	99	47	99
	10,000	47	99	46	99
	50,000	51	97	47	98

- With R-Chord as the underlying infrastructure, nearly all keys are retrieved
  - Though no replication
- In Chord, without replication, # keys retrieved is affected by *F*

Resiliency to Node Failures (2)

- In R-DHT, node is responsible for only one key, i.e., its own resources
- In conventional DHT, node is responsible for several keys (even clusters), i.e., index other resources. When a node is down, it affects resources belonging to other nodes.



Query Cost

F	K	# Available Keys	
		$d = 3$	$d = 4$
20%	1,000	5	44
	10,000	49	346
	50,000	220	716
40%	1,000	5	44
	10,000	49	346
	50,000	206	668

F	K	# Nodes Visited			
		$d = 3$		$d = 4$	
		Chord	R-Chord	Chord	R-Chord
20%	1,000	154	63	202	221
	10,000	154	160	206	592
	50,000	154	357	207	945
40%	1,000	113	60	153	213
	10,000	113	154	149	580
	50,000	112	336	145	908

Expected size of result set is affected by *d*

- Size of *d*-dimensional space is fixed (2^m)
- Increasing *d* causes the space to be more compact and dense

- In Chord, cost is constant
  - Query selectivity (size of query region) is constant
- In R-Chord, cost is affected by size of result set

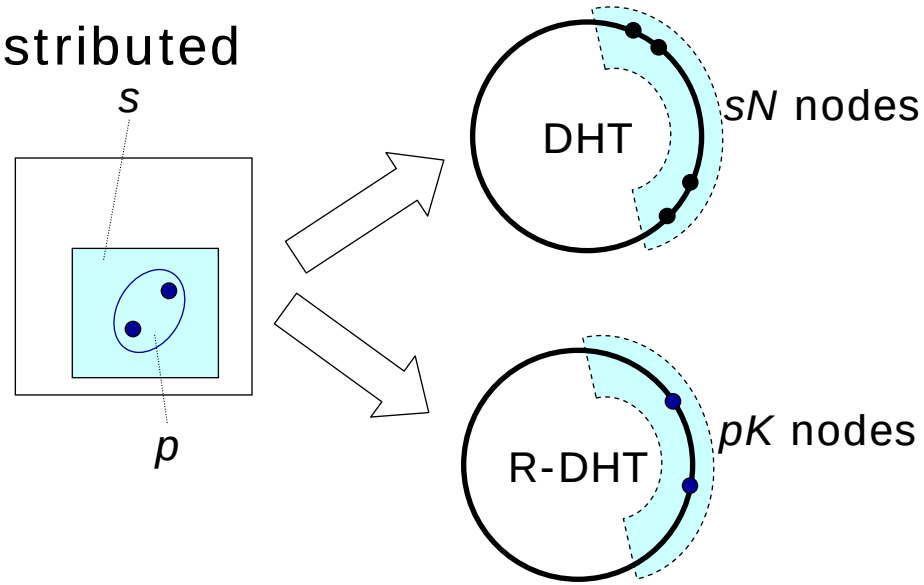
Query Cost

F	K	# Nodes Visited			
		$d = 3$		$d = 4$	
		Chord	R-Chord	Chord	R-Chord
20%	1,000	154	63	202	221
	10,000	154	160	206	592
	50,000	154	357	207	945
40%	1,000	113	60	153	213
	10,000	113	154	149	580
	50,000	112	336	145	908

- Cost = #hops visited
- In Chord, cost is constant
  - Query selectivity (size of query region) is constant
  - Cost is determined by size of overlay network (*N*)
- In R-Chord, cost is affected by size of result set (which in turn, is affected by *K*)
  - Performance hit is due to #lookups, **not** the cost (i.e. path length) of *individual* lookup

Query Cost (2)

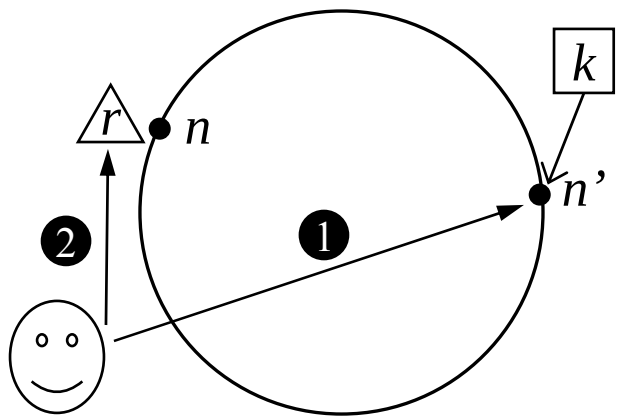
- Assume keys are uniformly distributed
- Query selectivity is $0 \leq s \leq 1$
- Result set contains *pK* keys
  - $0 \leq p \leq 1$



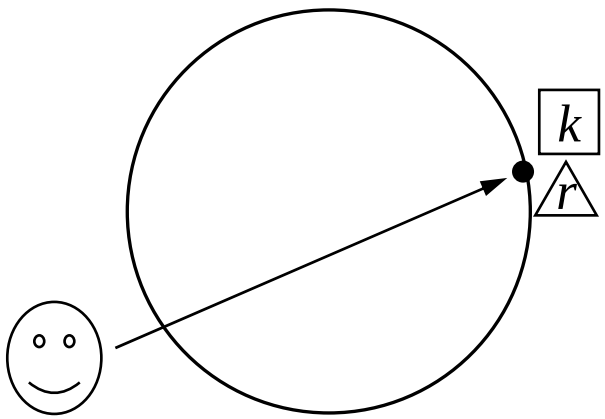
- Conventional DHT visits *sN* nodes, i.e. all nodes responsible for query region
- R-DHT visits *pK* nodes, i.e. equals to #keys (#answers)

Amortizing Query Cost

- Conventional DHT separates keys and resources
  - At the end, still need to contact the administrative domain that shares the resources



Conventional DHT



R-DHT

Query Performance under Churn

K	% Keys Retrieved			
	$d = 3$		$d = 4$	
	Chord	R-Chord	Chord	R-Chord
1,000	99	99	98	98
10,000	97	98	94	97
50,000	95	89	94	88

- Metrics: lookup resiliency under churn
- R-Chord performs reasonably well, considering that its overlay is larger (7x) than Chord.
- When *K* is increased, R-Chord cannot effectively exploits the segment-based overlay to support redundancy of routing tables.

Outline

- Introduction to R-DHT
- Problem Statement
- Related Works
- Midas
  - Indexing
  - Range-Query Optimizations
- Analysis
- **Conclusion**

Conclusion

- Midas
  - Indexing : resource type → key → R-DHT node
  - Query engine : incremental search + key elimination

- Implication of data-item distribution to performance of query processing
  - R-DHT achieves high lookup resiliency without requiring replication
  - R-DHT query cost is due to a higher number of lookup operations are needed
  - R-DHT is more suitable for large range queries with small result set
  - To improve query performance in R-DHT, allow selective data-item distributions