HORTON TABLES: FAST HASH TABLES FOR IN-MEMORY DATA-INTENSIVE COMPUTING

ALEX D. BRESLOW, DONG PING ZHANG, JOSEPH L. GREATHOUSE,
NUWAN JAYASENA, AND DEAN M. TULLSEN

6/23/2016

THE ROLE OF HASH TABLES
IN IN-MEMORY DATA-INTENSIVE COMPUTING

THE ROLE OF HASH TABLES
IN IN-MEMORY DATA-INTENSIVE COMPUTING

- Data stores and caches
  - Key-value stores (e.g., Memcached, Redis, and MongoDB)
  - Relational databases (e.g., MonetDB, HyPer, IBM DB2 with BLU)

THE ROLE OF HASH TABLES
IN IN-MEMORY DATA-INTENSIVE COMPUTING

- Data stores and caches
  - Key-value stores (e.g., Memcached, Redis, and MongoDB)
  - Relational databases (e.g., MonetDB, HyPer, IBM DB2 with BLU)

- Graphics
  - Accelerate computations by computing on hash tables that store sparse images, textures, or surfaces

THE ROLE OF HASH TABLES
IN IN-MEMORY DATA-INTENSIVE COMPUTING

- Data stores and caches
  - Key-value stores (e.g., Memcached, Redis, and MongoDB)
  - Relational databases (e.g., MonetDB, HyPer, IBM DB2 with BLU)
- Graphics
  - Accelerate computations by computing on hash tables that store sparse images, textures, or surfaces
- General data compression schemes used in common compression utilities

THE ROLE OF HASH TABLES
IN IN-MEMORY DATA-INTENSIVE COMPUTING

- Data stores and caches
  - Key-value stores (e.g., Memcached, Redis, and MongoDB)
  - Relational databases (e.g., MonetDB, HyPer, IBM DB2 with BLU)

- Graphics
  - Accelerate computations by computing on hash tables that store sparse images, textures, or surfaces

- General data compression schemes used in common compression utilities

- **In each of these fields, having a fast hash table is important.**

FOCUS OF THIS TALK

OPTIMIZING MEMORY ACCESSES IN FAST IN-MEMORY HASH TABLES

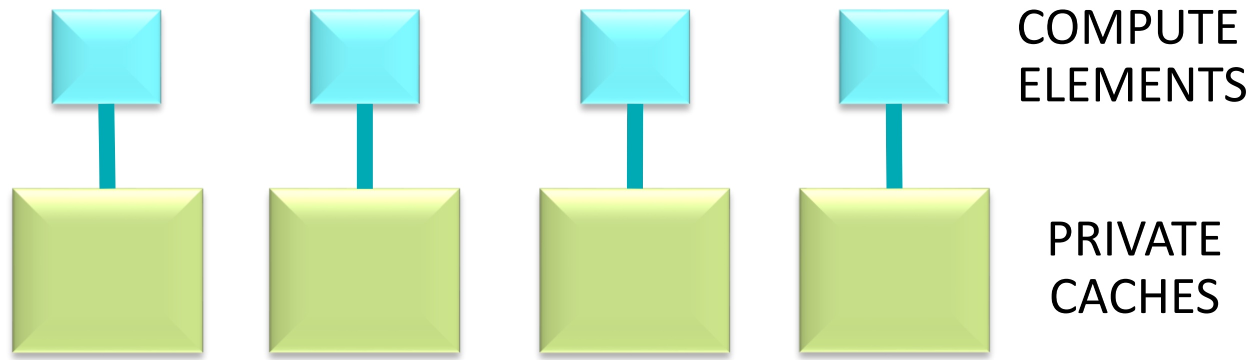
FOCUS OF THIS TALK

OPTIMIZING MEMORY ACCESSES IN FAST IN-MEMORY HASH TABLES

COMPUTE ELEMENTS

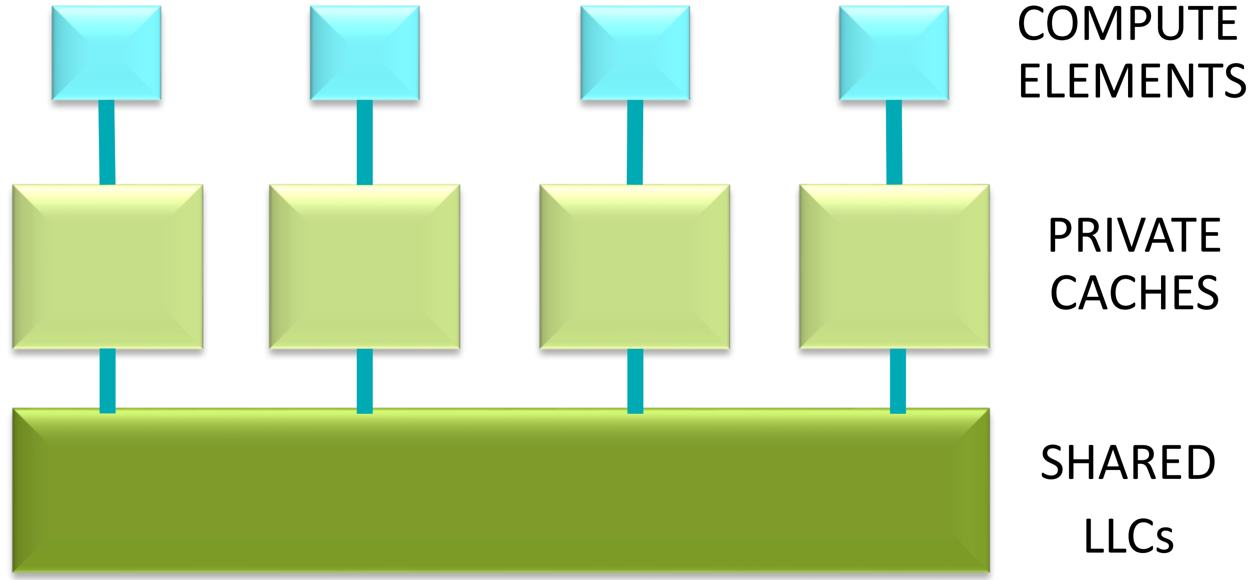
FOCUS OF THIS TALK

OPTIMIZING MEMORY ACCESSES IN FAST IN-MEMORY HASH TABLES



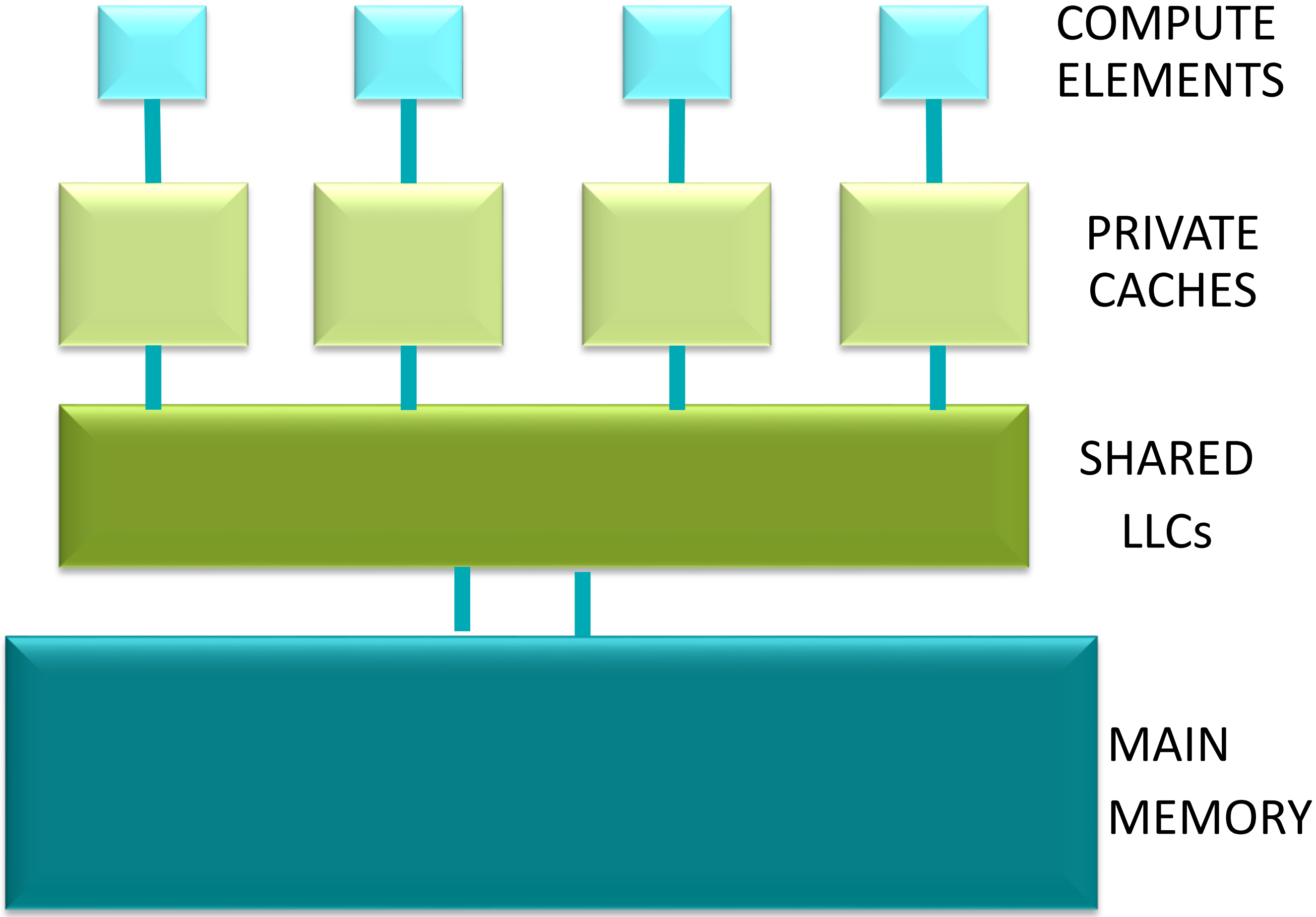
FOCUS OF THIS TALK

OPTIMIZING MEMORY ACCESSES IN FAST IN-MEMORY HASH TABLES



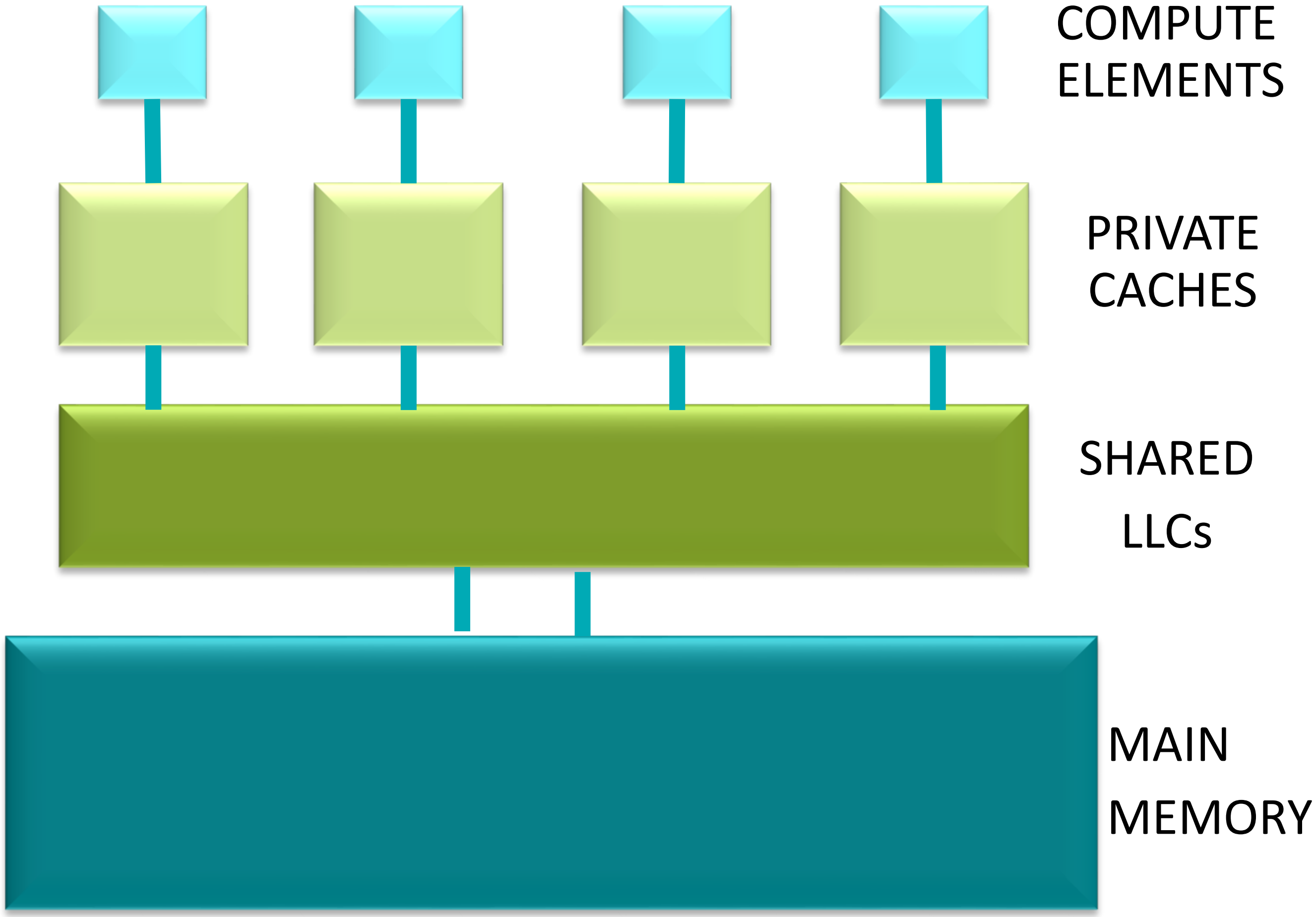
FOCUS OF THIS TALK

OPTIMIZING MEMORY ACCESSES IN FAST IN-MEMORY HASH TABLES



FOCUS OF THIS TALK

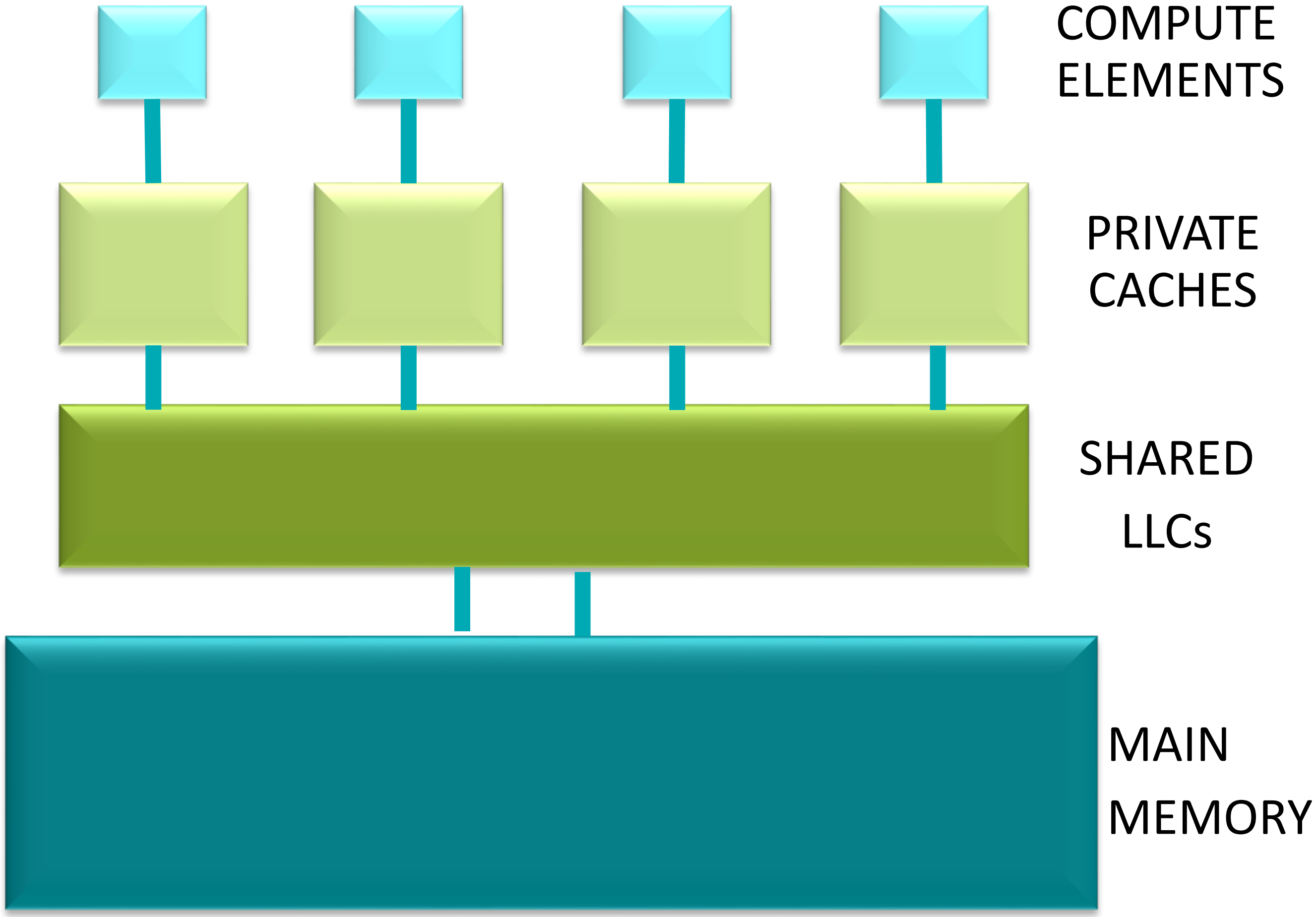
OPTIMIZING MEMORY ACCESSES IN FAST IN-MEMORY HASH TABLES



- Hash tables have poor temporal and spatial locality.

FOCUS OF THIS TALK

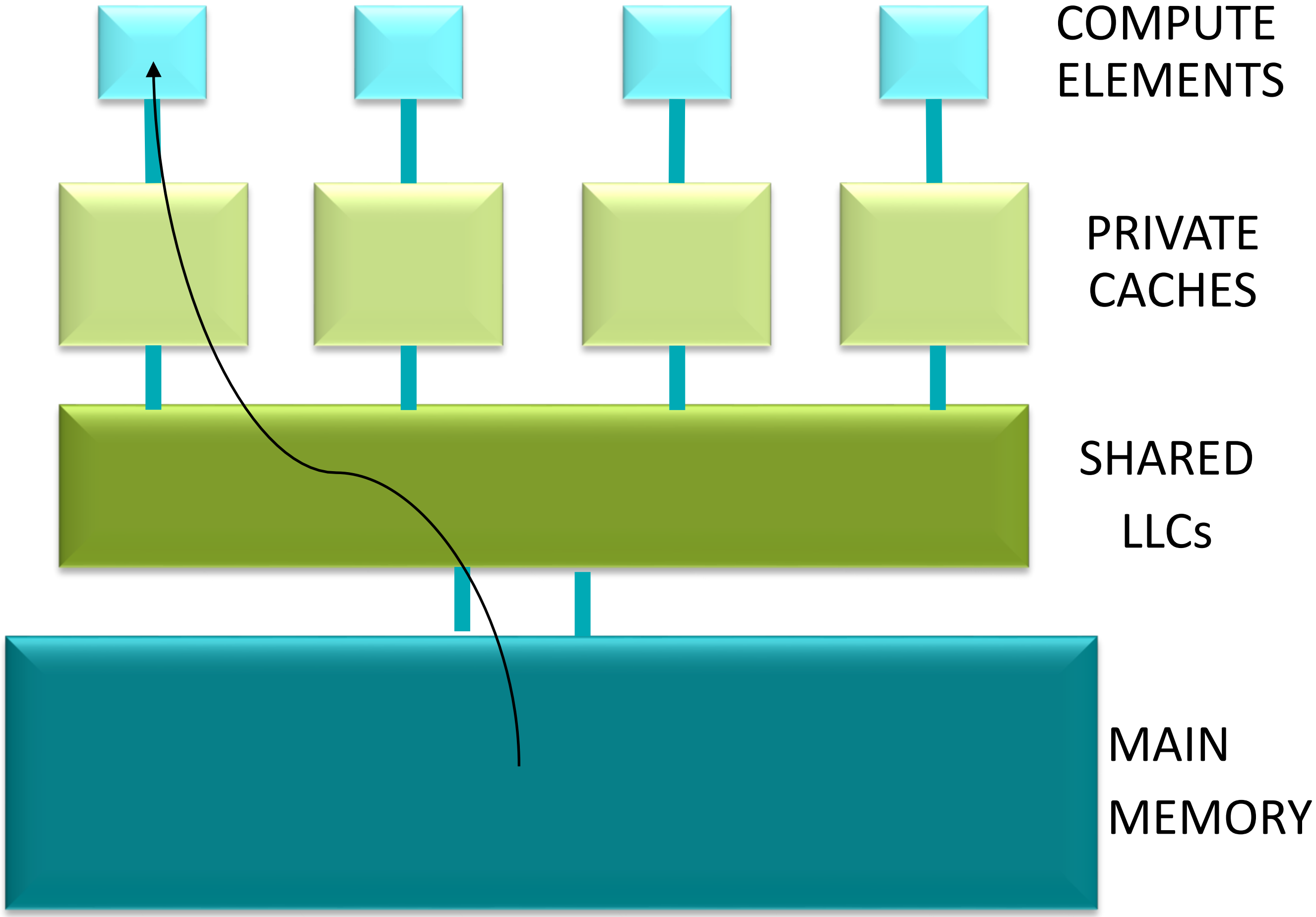
OPTIMIZING MEMORY ACCESSES IN FAST IN-MEMORY HASH TABLES



- Hash tables have poor temporal and spatial locality.
- In-memory hash tables often have hot working sets that are bigger than LLCs.

FOCUS OF THIS TALK

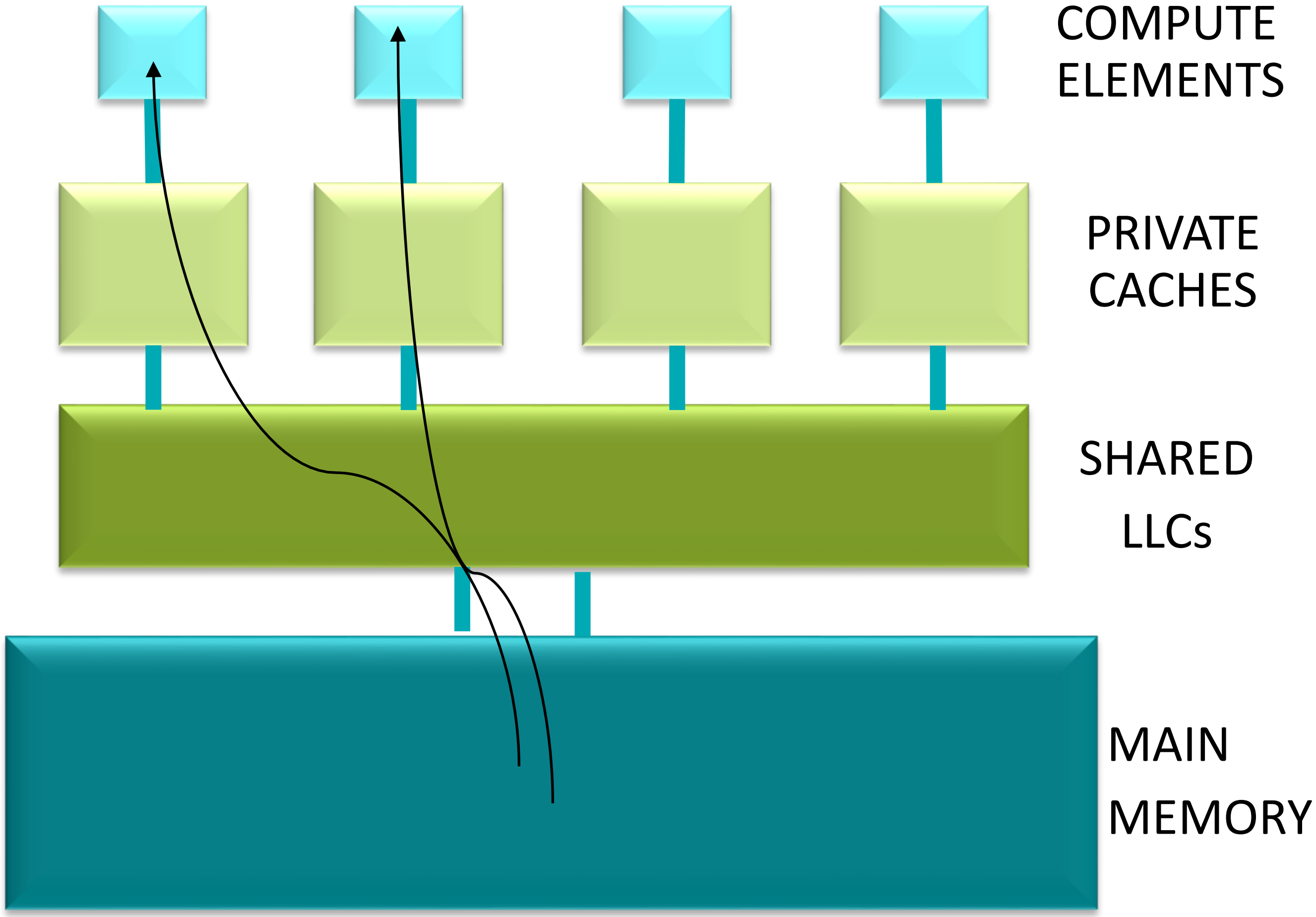
OPTIMIZING MEMORY ACCESSES IN FAST IN-MEMORY HASH TABLES



- Hash tables have poor temporal and spatial locality.
- In-memory hash tables often have hot working sets that are bigger than LLCs.

FOCUS OF THIS TALK

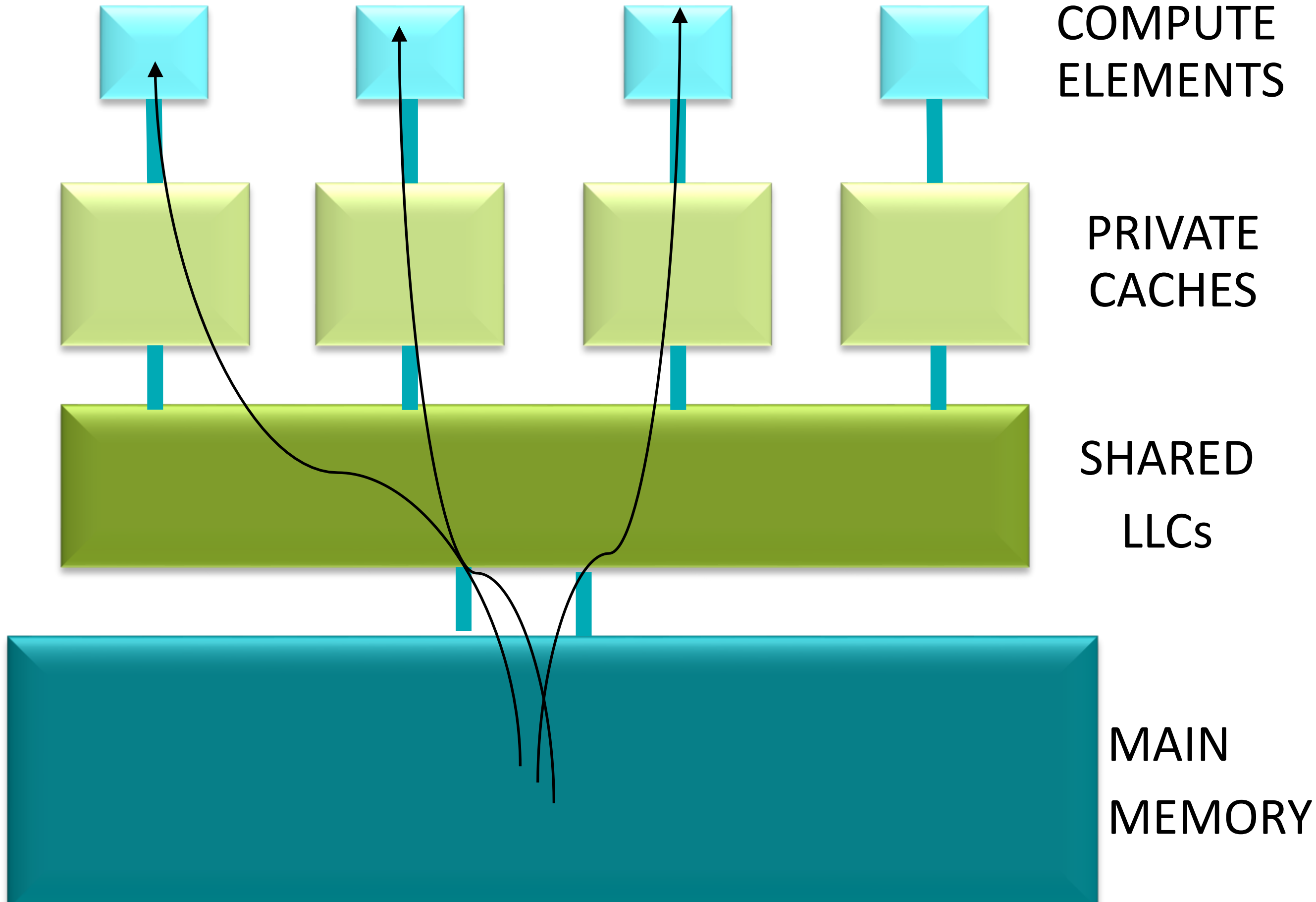
OPTIMIZING MEMORY ACCESSES IN FAST IN-MEMORY HASH TABLES



- Hash tables have poor temporal and spatial locality.
- In-memory hash tables often have hot working sets that are bigger than LLCs.

FOCUS OF THIS TALK

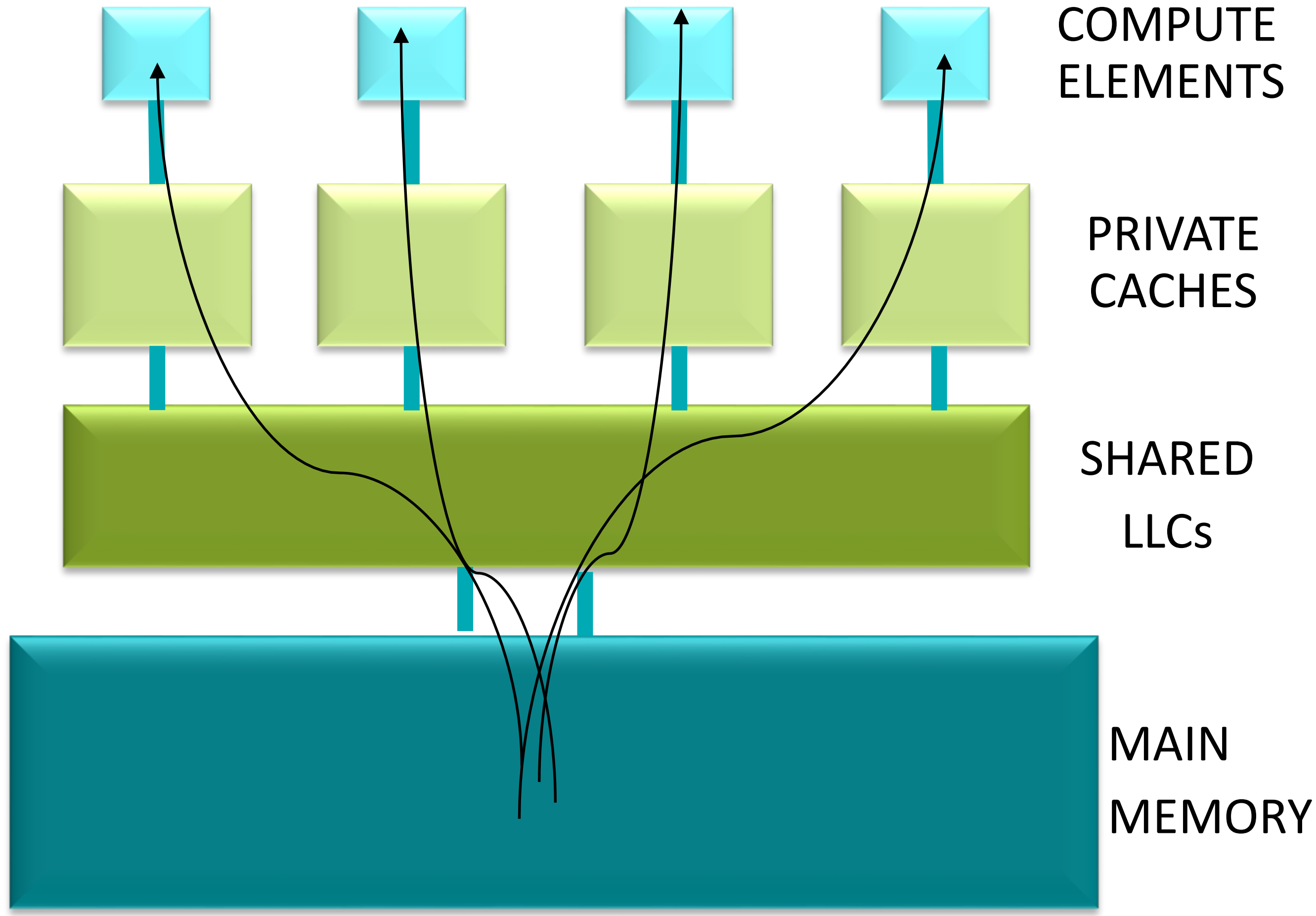
OPTIMIZING MEMORY ACCESSES IN FAST IN-MEMORY HASH TABLES



- Hash tables have poor temporal and spatial locality.
- In-memory hash tables often have hot working sets that are bigger than LLCs.

FOCUS OF THIS TALK

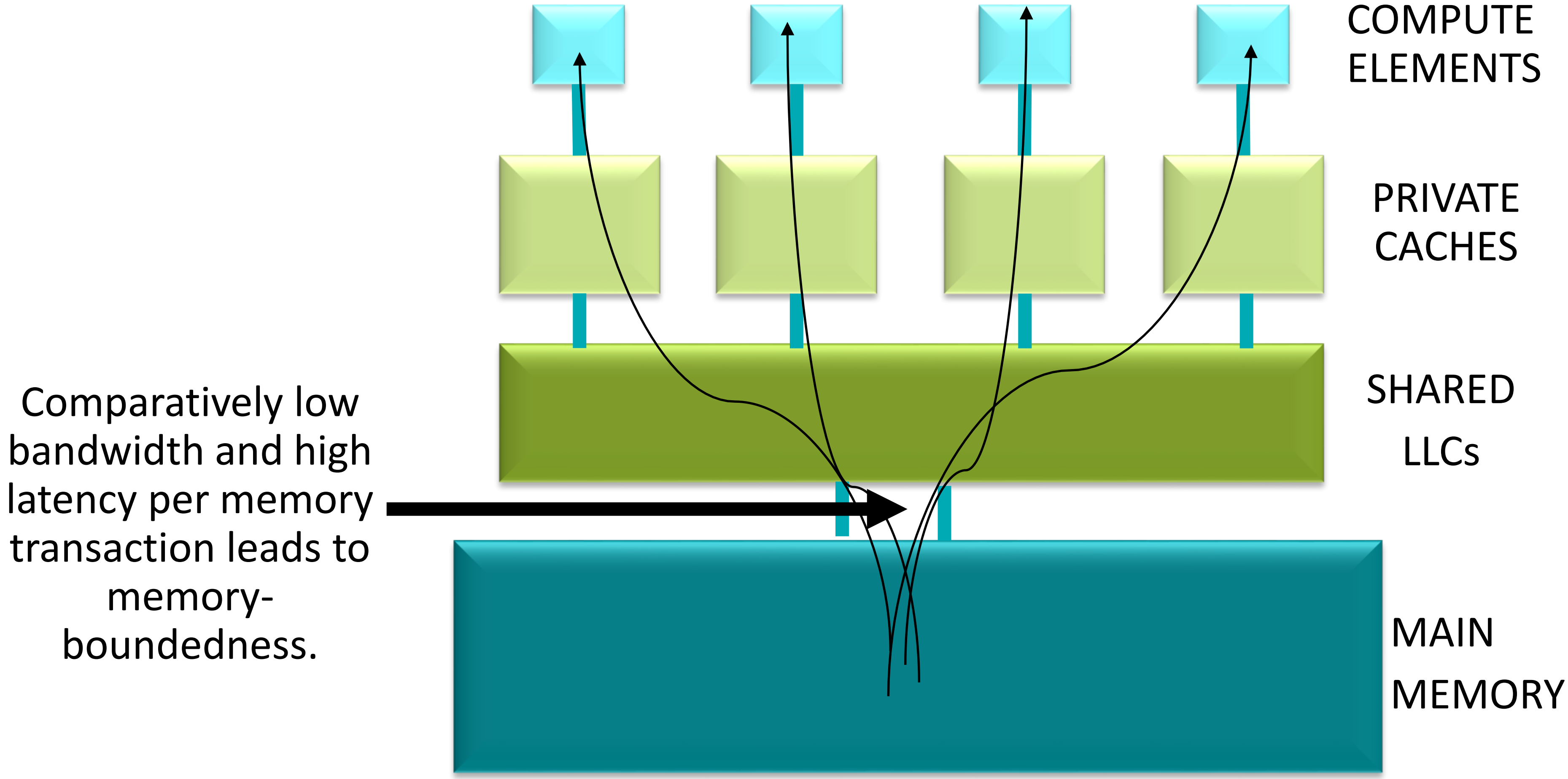
OPTIMIZING MEMORY ACCESSES IN FAST IN-MEMORY HASH TABLES



- Hash tables have poor temporal and spatial locality.
- In-memory hash tables often have hot working sets that are bigger than LLCs.

FOCUS OF THIS TALK

OPTIMIZING MEMORY ACCESSES IN FAST IN-MEMORY HASH TABLES



- Hash tables have poor temporal and spatial locality.
- In-memory hash tables often have hot working sets that are bigger than LLCs.

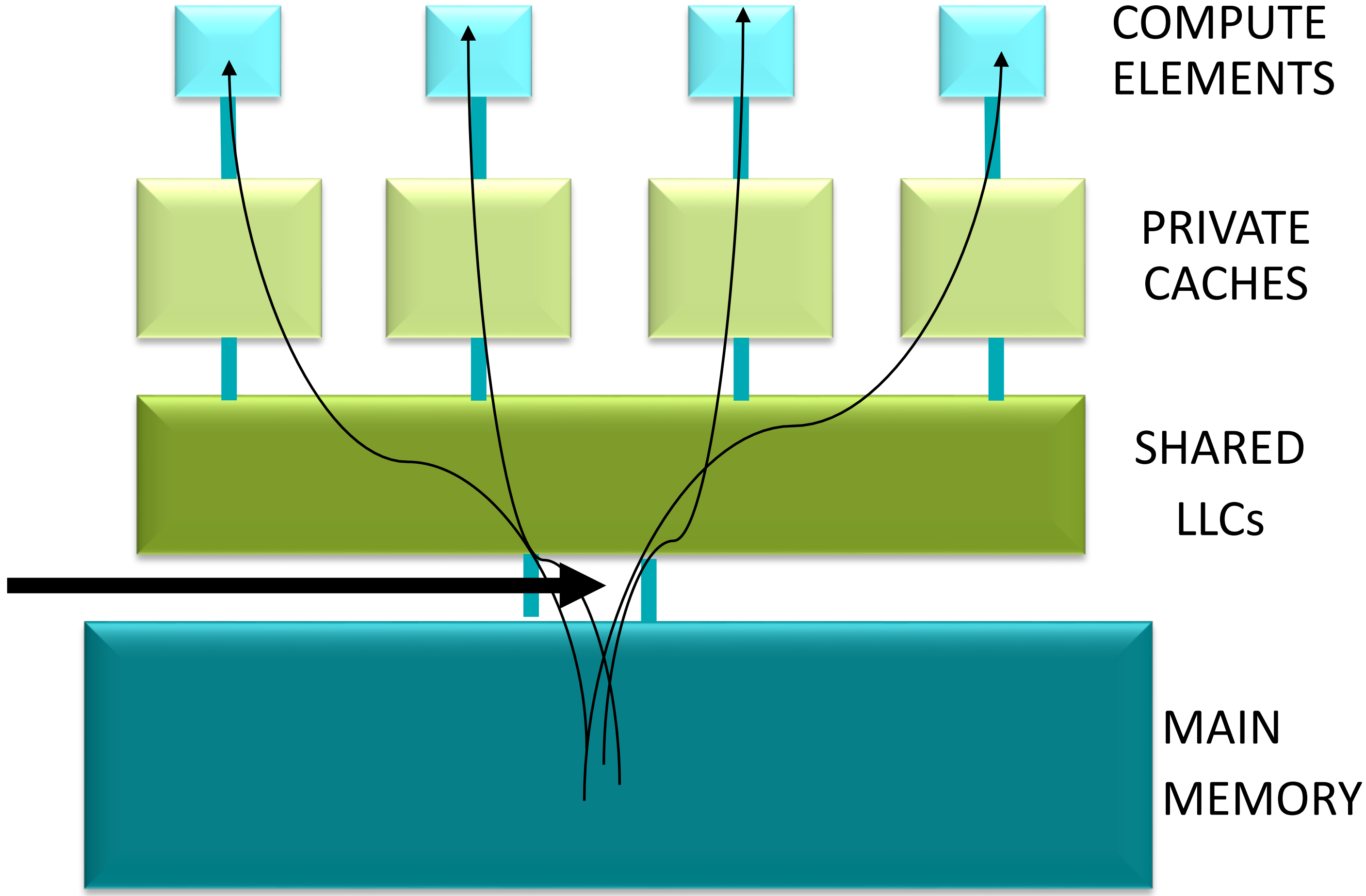
FOCUS OF THIS TALK

OPTIMIZING MEMORY ACCESSES IN FAST IN-MEMORY HASH TABLES

We need to aggressively optimize hash tables to be cognizant of this limitation.

↓

Comparatively low bandwidth and high latency per memory transaction leads to memory-boundedness.



- Hash tables have poor temporal and spatial locality.
- In-memory hash tables often have hot working sets that are bigger than LLCs.

BUCKETIZED CUCKOO HASH TABLES

BUCKETIZED CUCKOO HASH TABLES

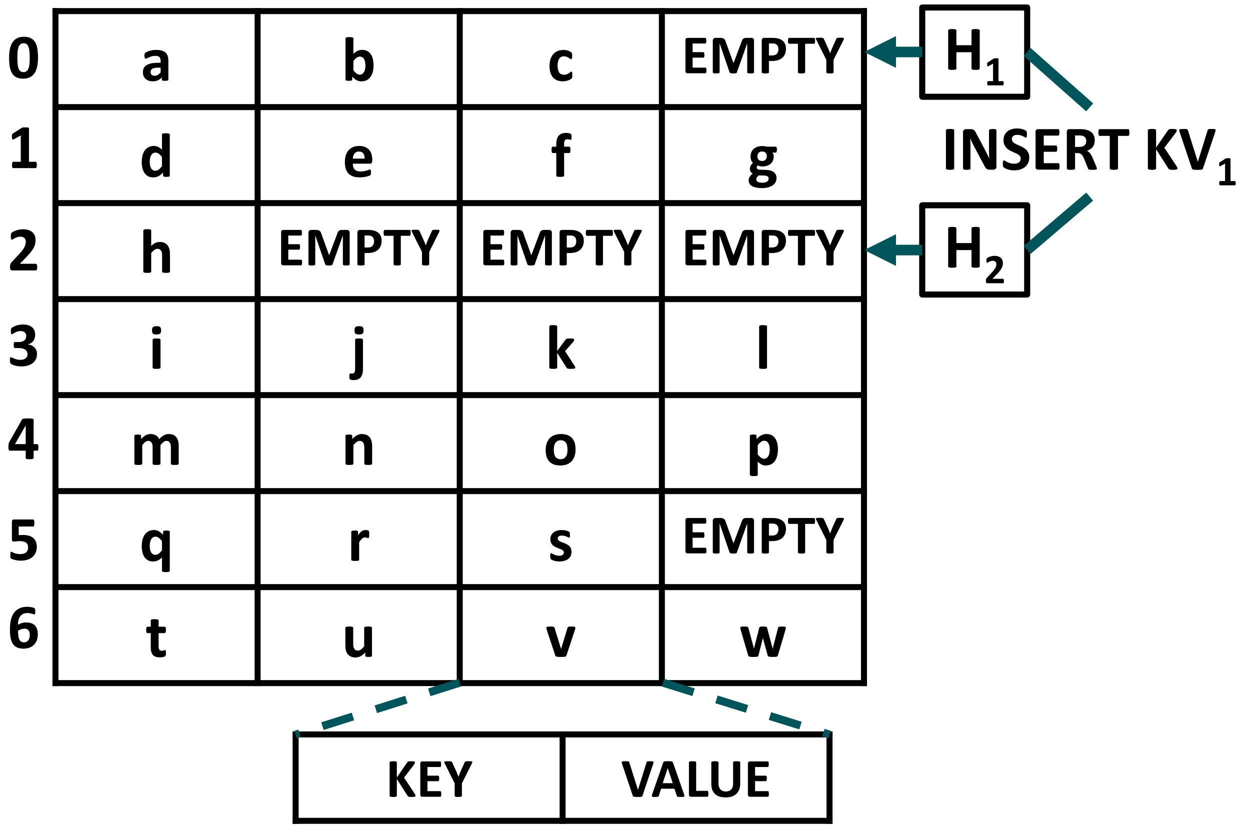
a	b	c	EMPTY
d	e	f	g
h	EMPTY	EMPTY	EMPTY
i	j	k	l
m	n	o	p
q	r	s	EMPTY
t	u	v	w

BUCKETIZED CUCKOO HASH TABLES

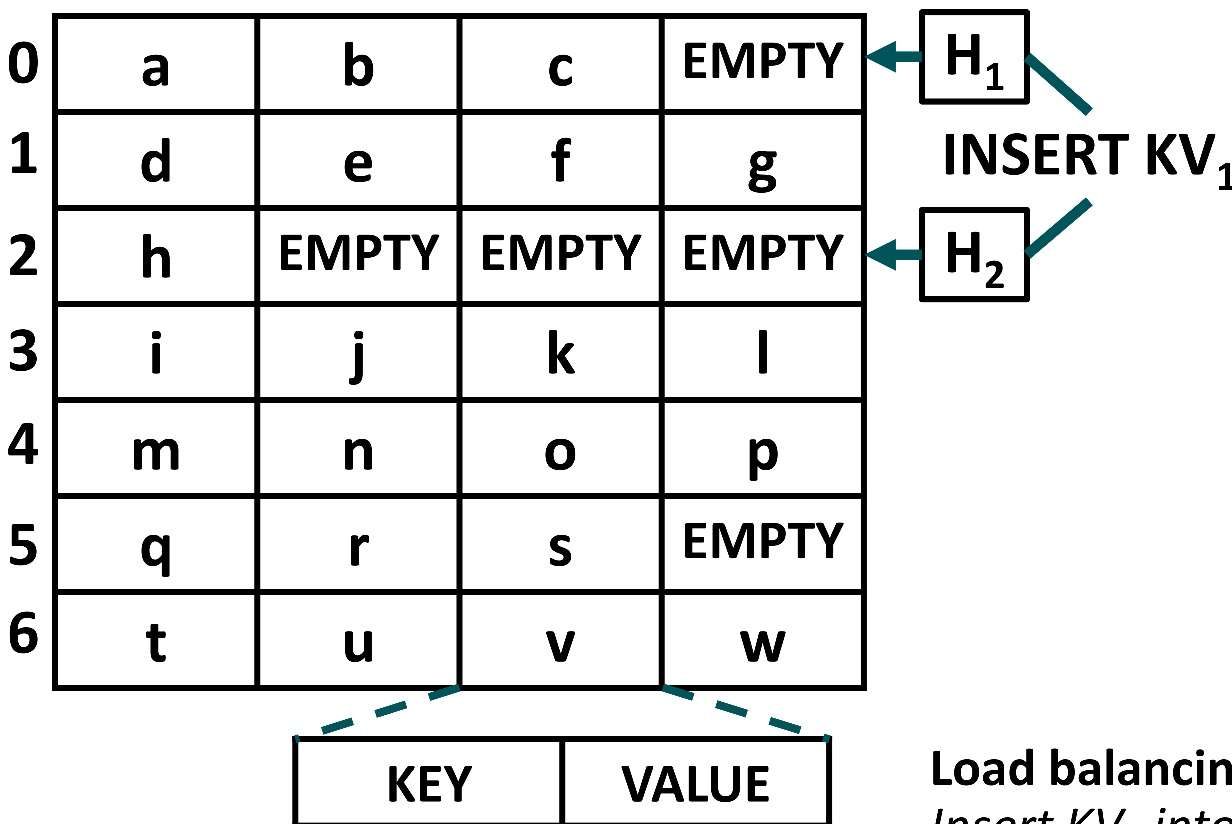
a	b	c	EMPTY
d	e	f	g
h	EMPTY	EMPTY	EMPTY
i	j	k	l
m	n	o	p
q	r	s	EMPTY
t	u	v	w

KEY	VALUE

BUCKETIZED CUCKOO HASH TABLES



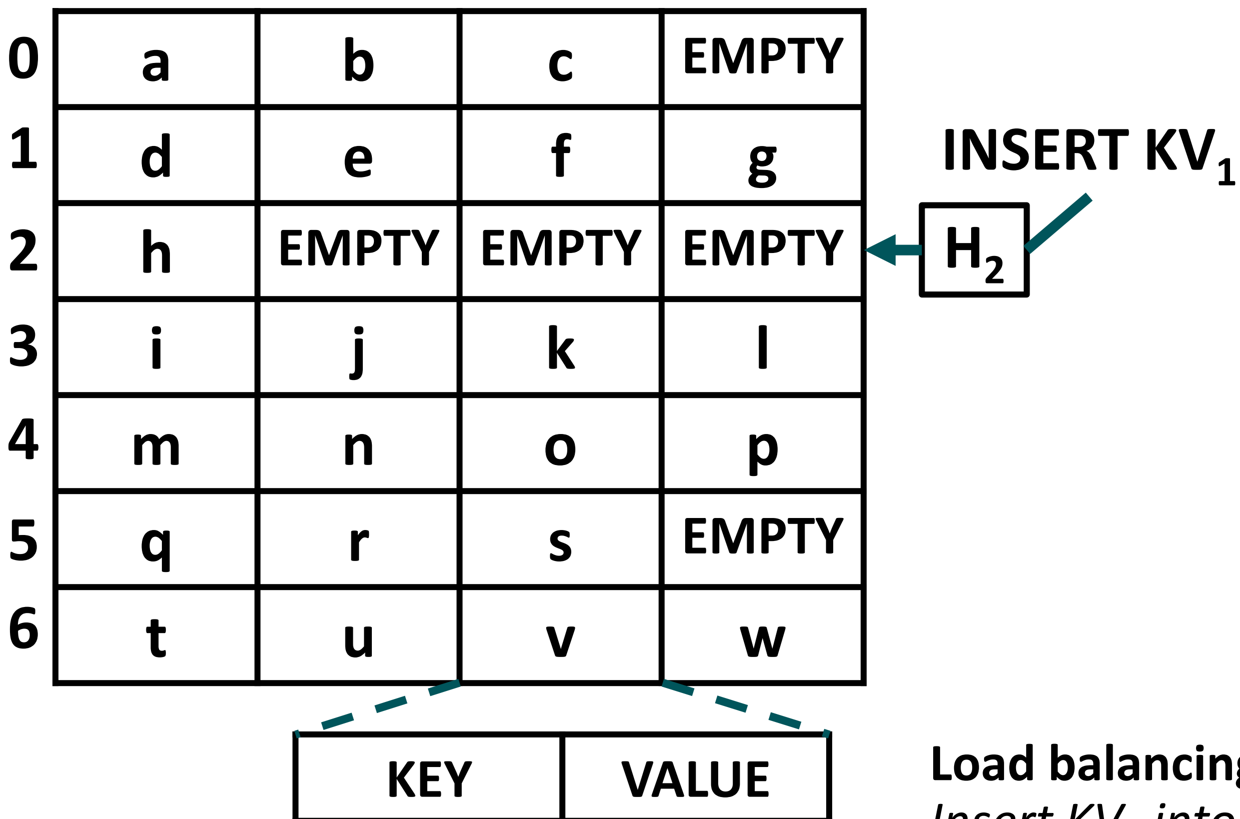
BUCKETIZED CUCKOO HASH TABLES



Load balancing

Insert KV_1 into the least full candidate bucket

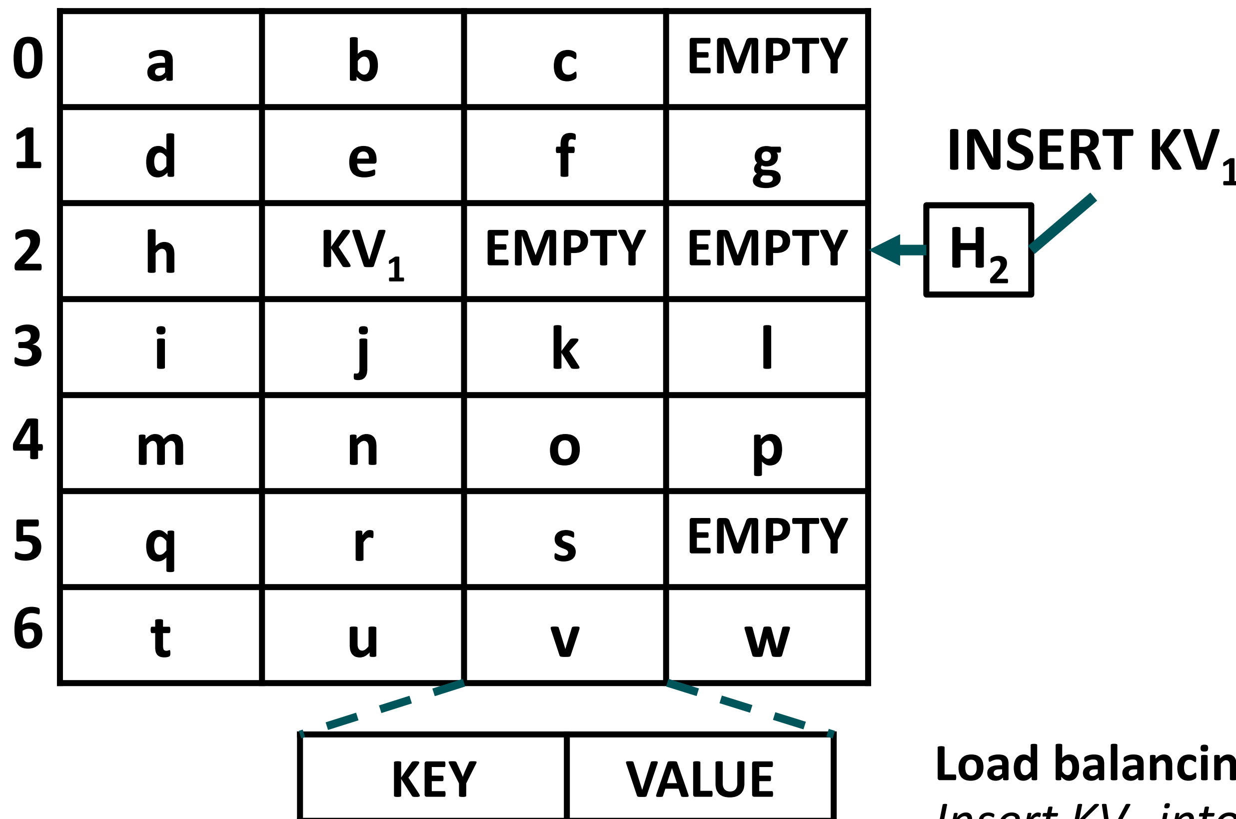
BUCKETIZED CUCKOO HASH TABLES



Load balancing

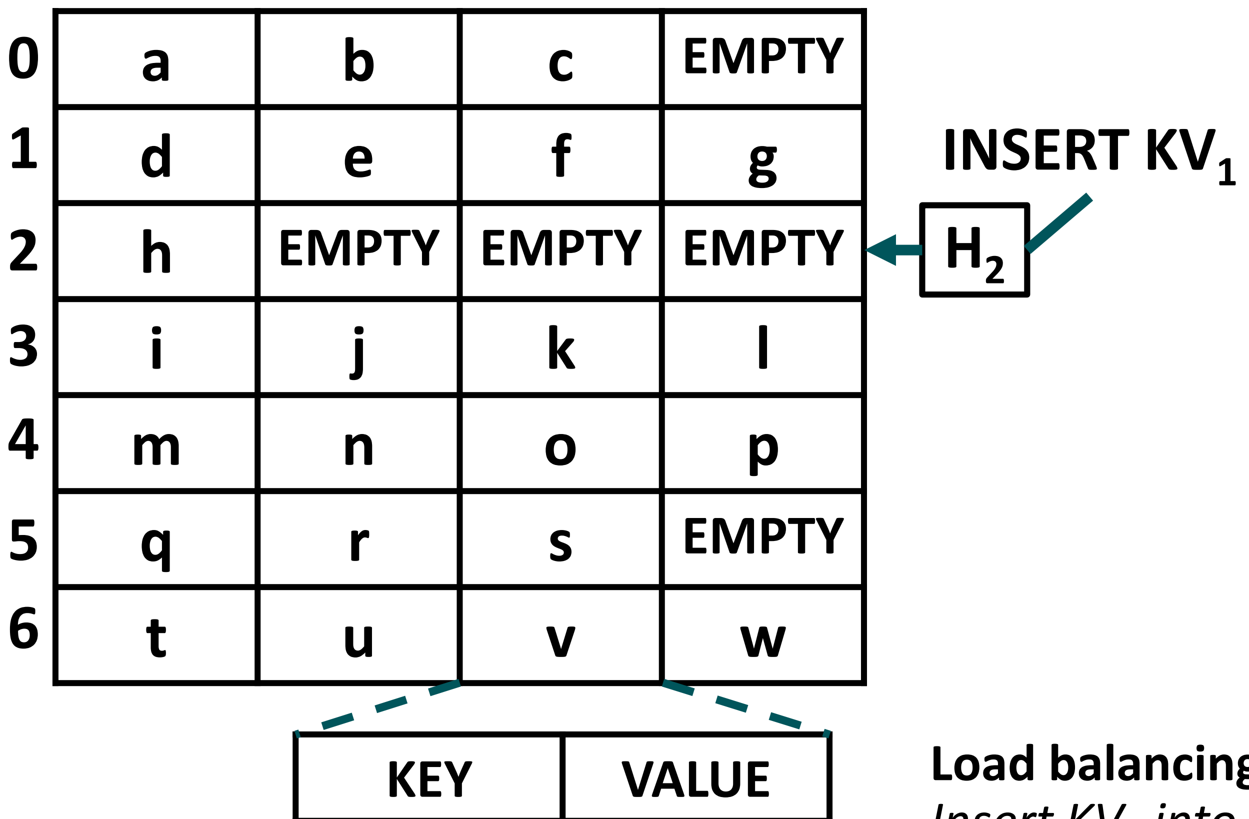
Insert KV_1 into the least full candidate bucket

BUCKETIZED CUCKOO HASH TABLES



Load balancing
Insert KV_1 into the least full candidate bucket

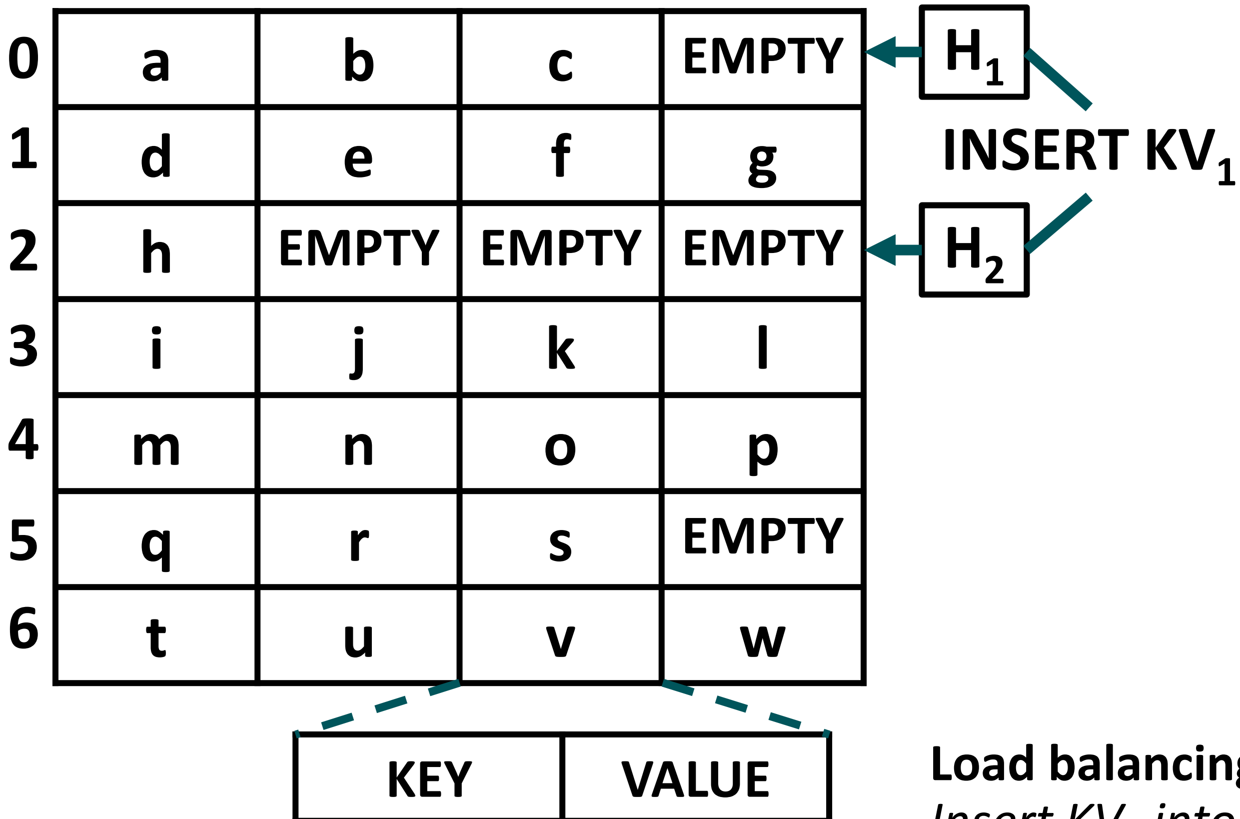
BUCKETIZED CUCKOO HASH TABLES



Load balancing

Insert KV_1 into the least full candidate bucket

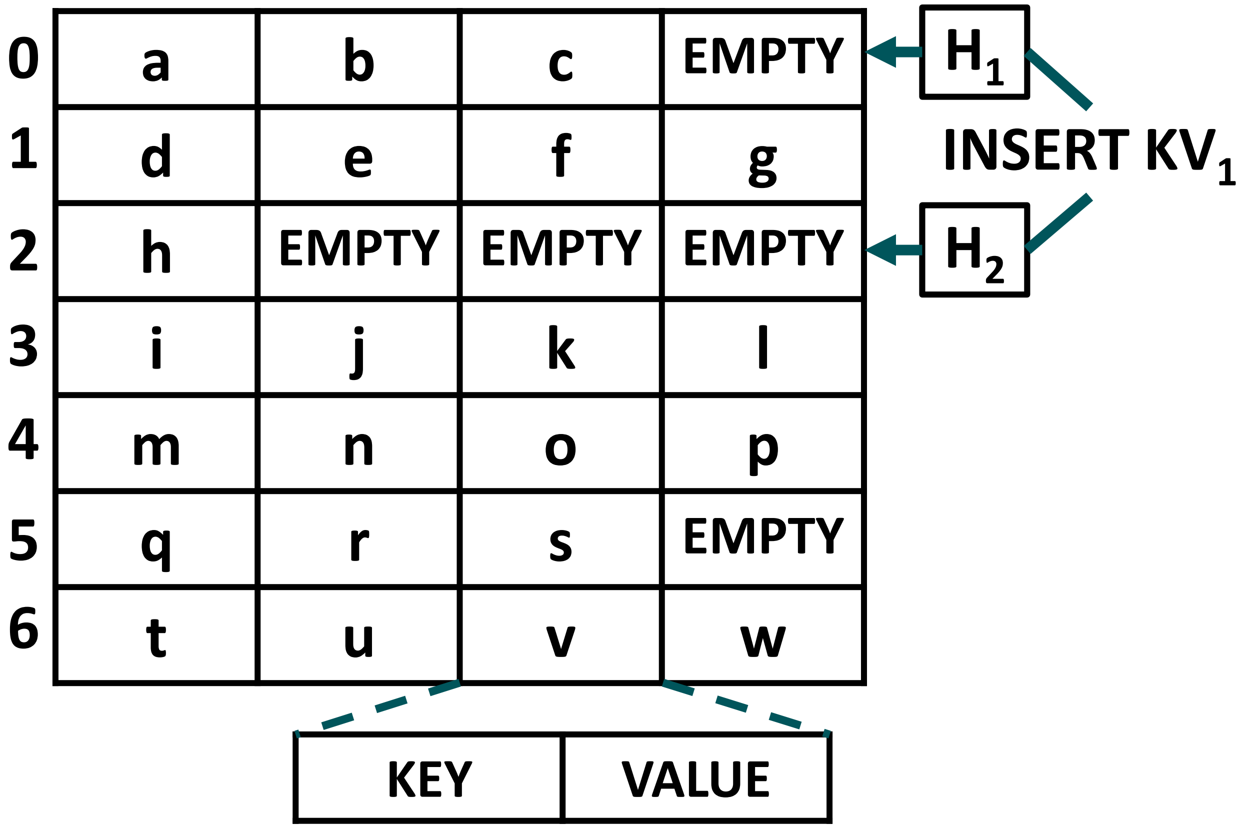
BUCKETIZED CUCKOO HASH TABLES



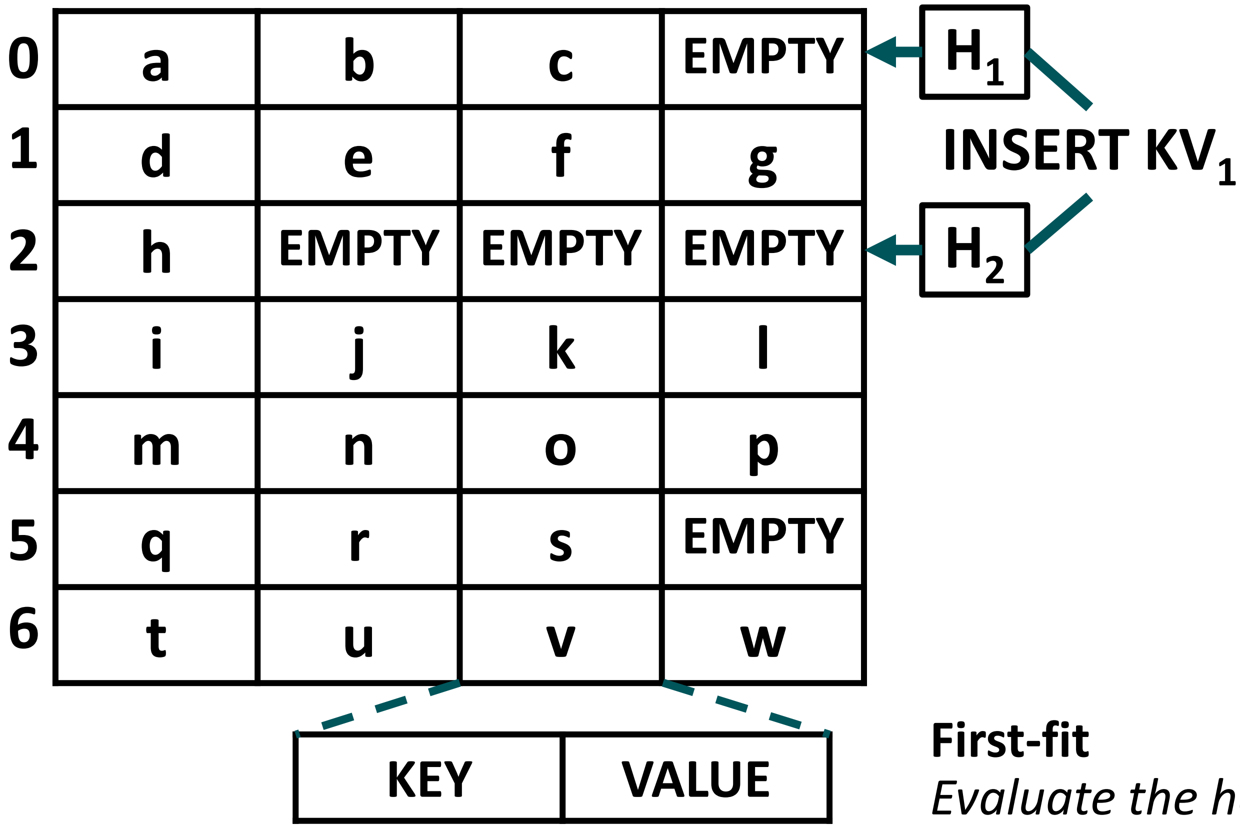
Load balancing

Insert KV_1 into the least full candidate bucket

BUCKETIZED CUCKOO HASH TABLES



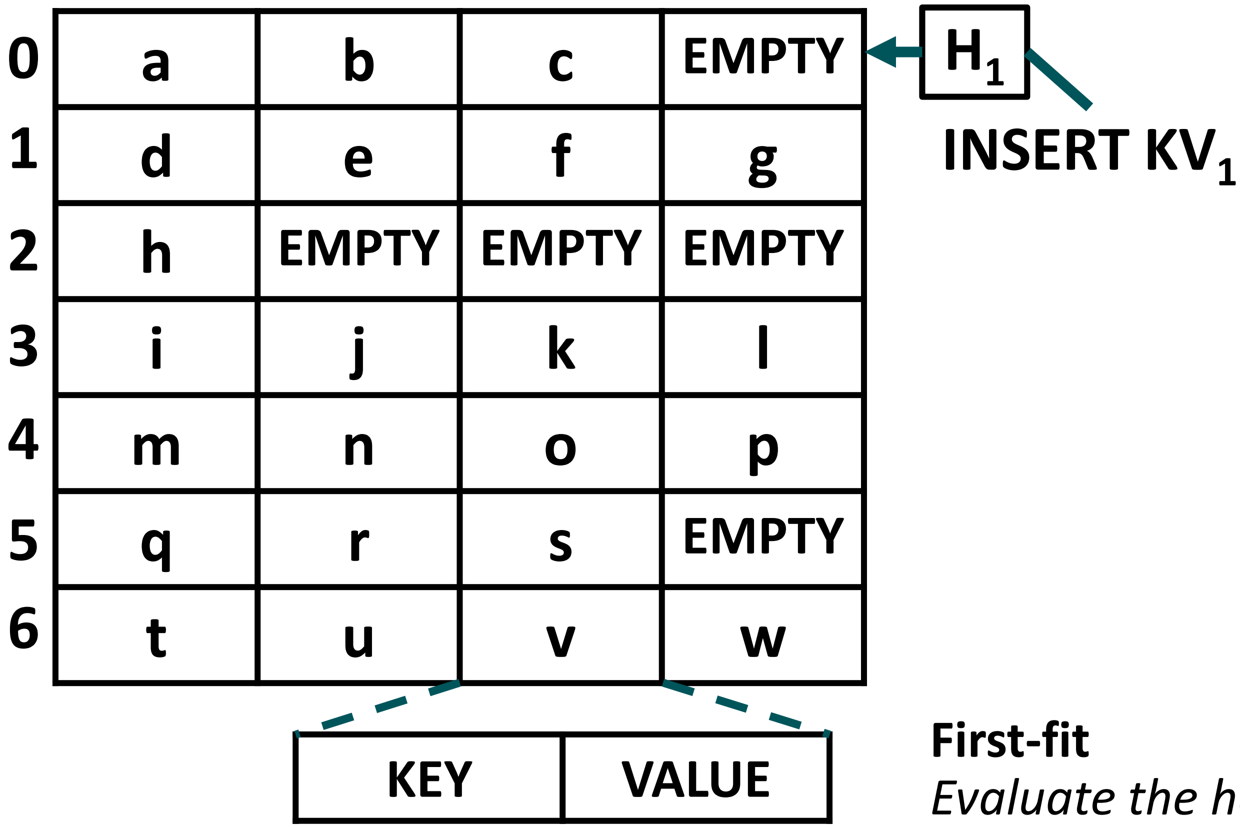
BUCKETIZED CUCKOO HASH TABLES



First-fit

Evaluate the hash functions in numerical order and insert KV_1 into the first candidate bucket with a free slot

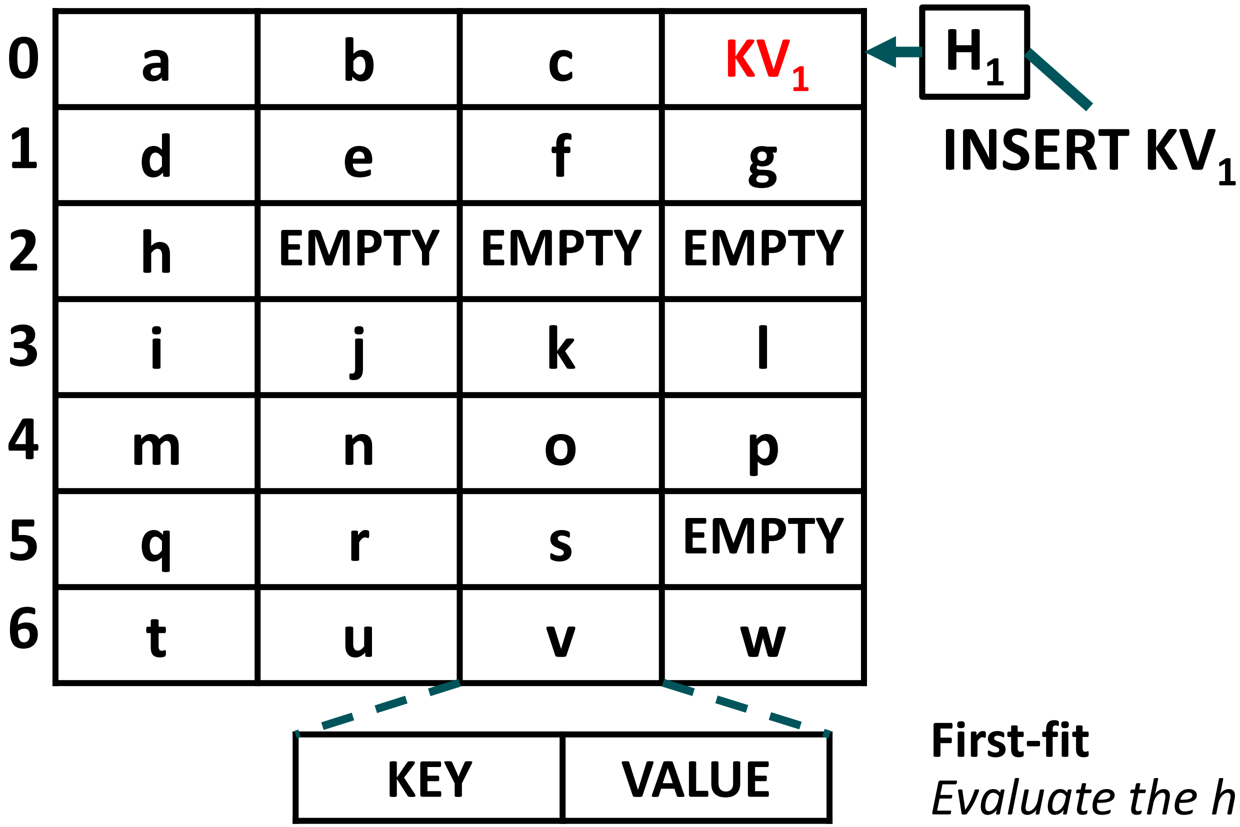
BUCKETIZED CUCKOO HASH TABLES



First-fit

Evaluate the hash functions in numerical order and insert KV_1 into the first candidate bucket with a free slot

BUCKETIZED CUCKOO HASH TABLES



First-fit

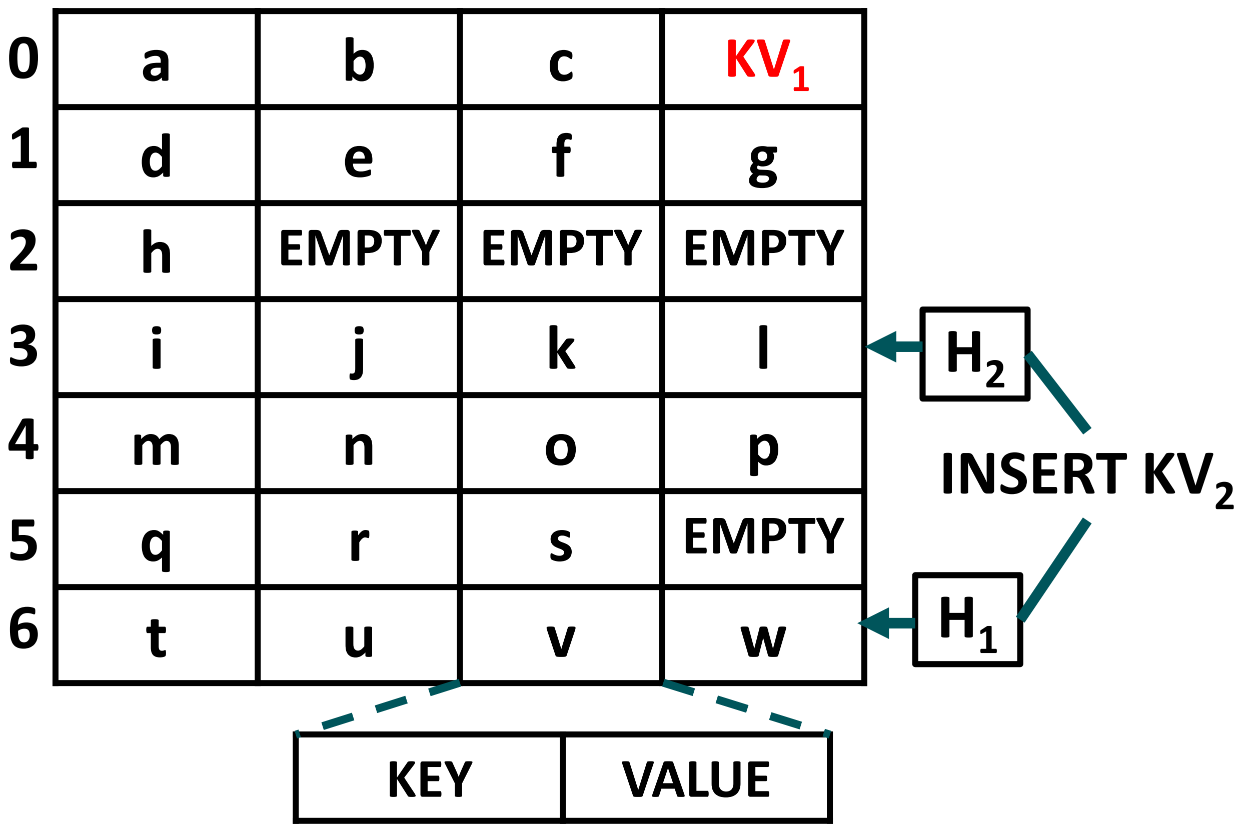
Evaluate the hash functions in numerical order and insert KV_1 into the first candidate bucket with a free slot

BUCKETIZED CUCKOO HASH TABLES

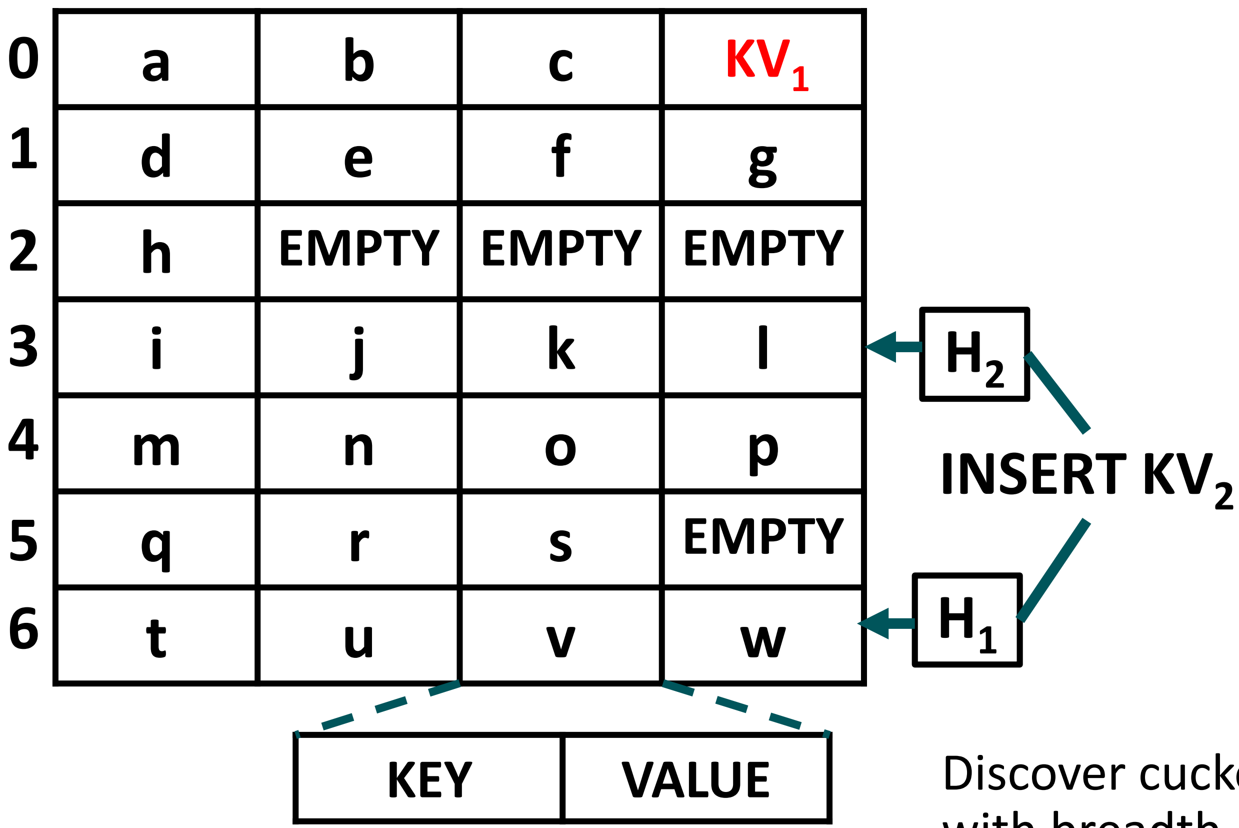
0	a	b	c	KV_1
1	d	e	f	g
2	h	EMPTY	EMPTY	EMPTY
3	i	j	k	l
4	m	n	o	p
5	q	r	s	EMPTY
6	t	u	v	w

KEY	VALUE

BUCKETIZED CUCKOO HASH TABLES

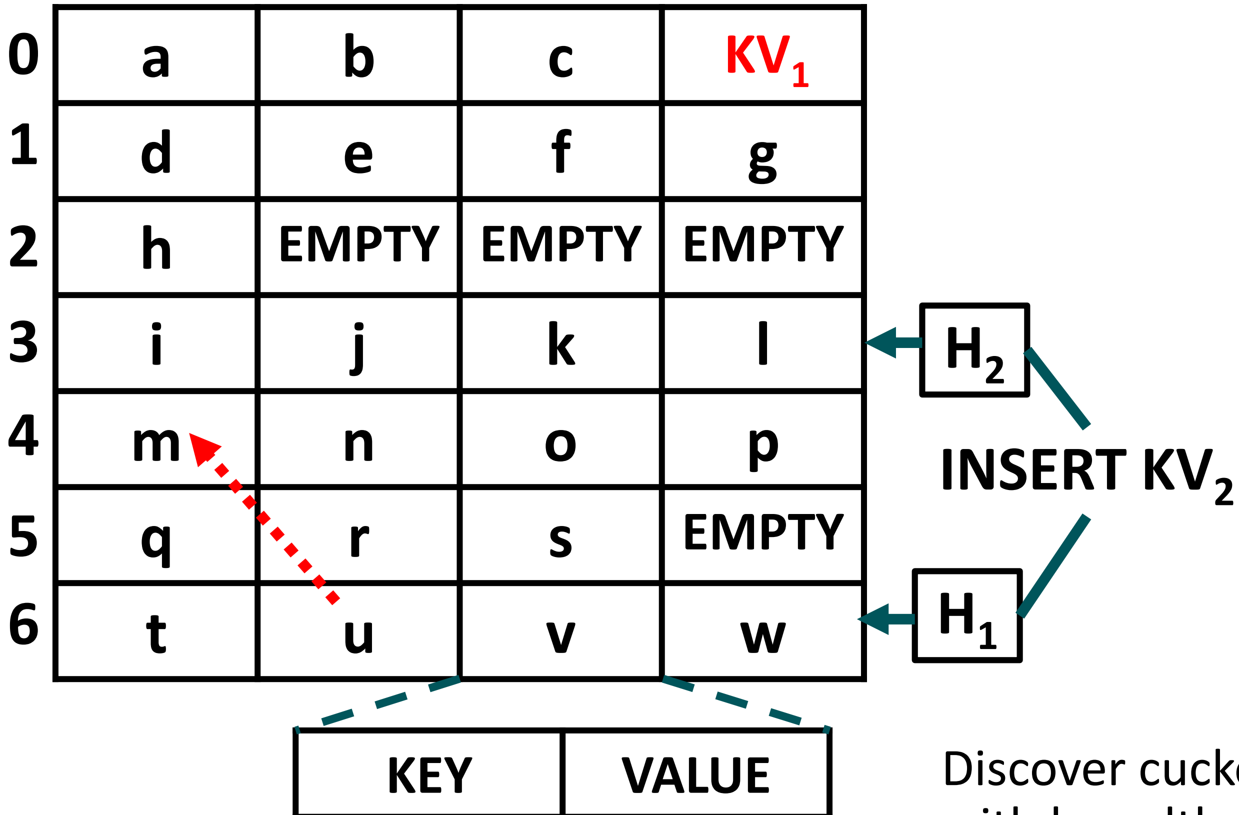


BUCKETIZED CUCKOO HASH TABLES



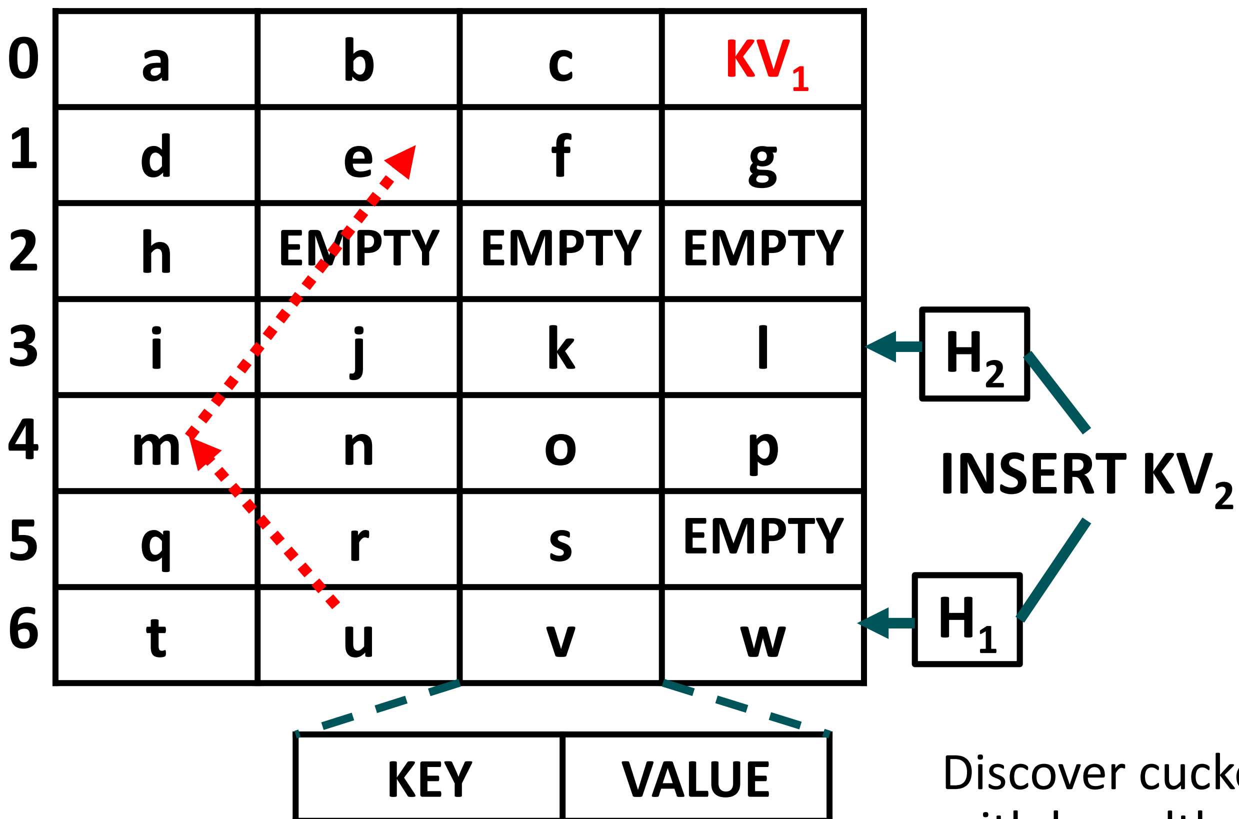
Discover cuckoo chain with breadth-first search

BUCKETIZED CUCKOO HASH TABLES



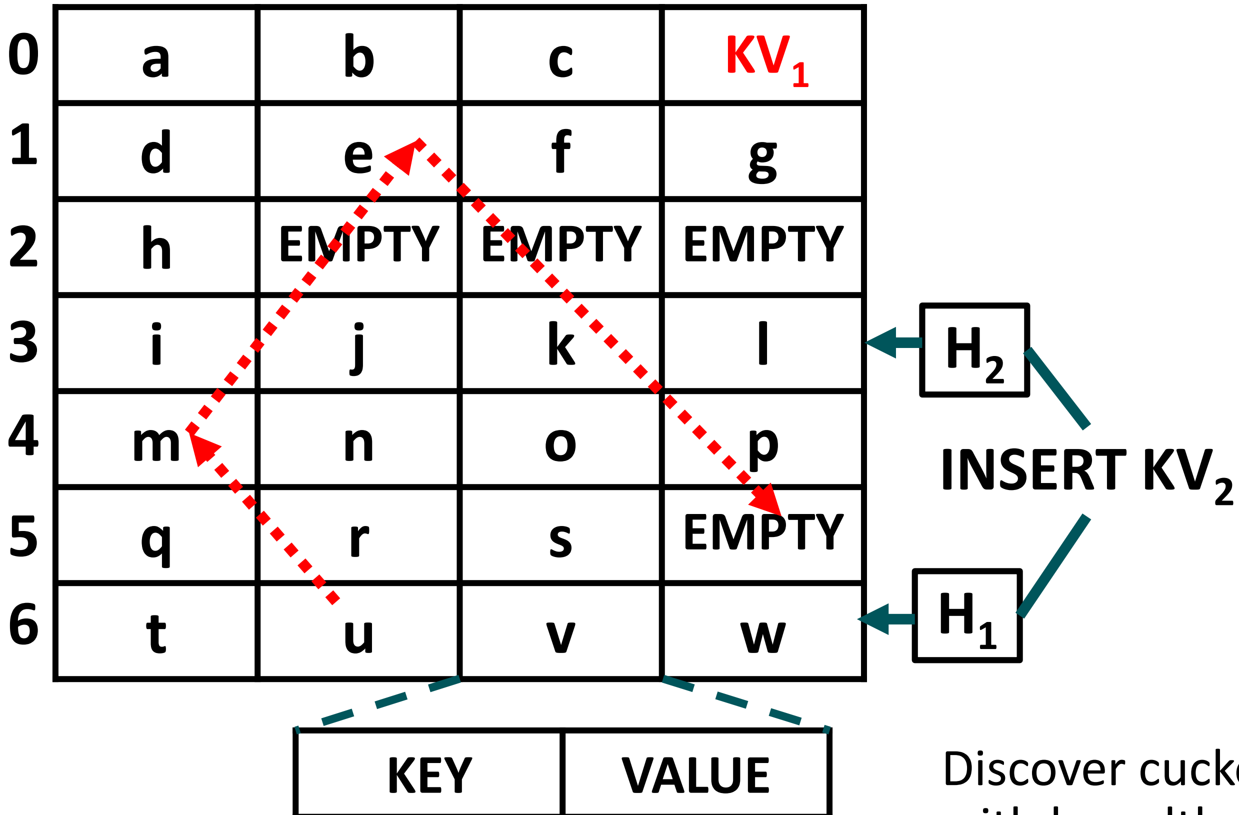
Discover cuckoo chain with breadth-first search

BUCKETIZED CUCKOO HASH TABLES



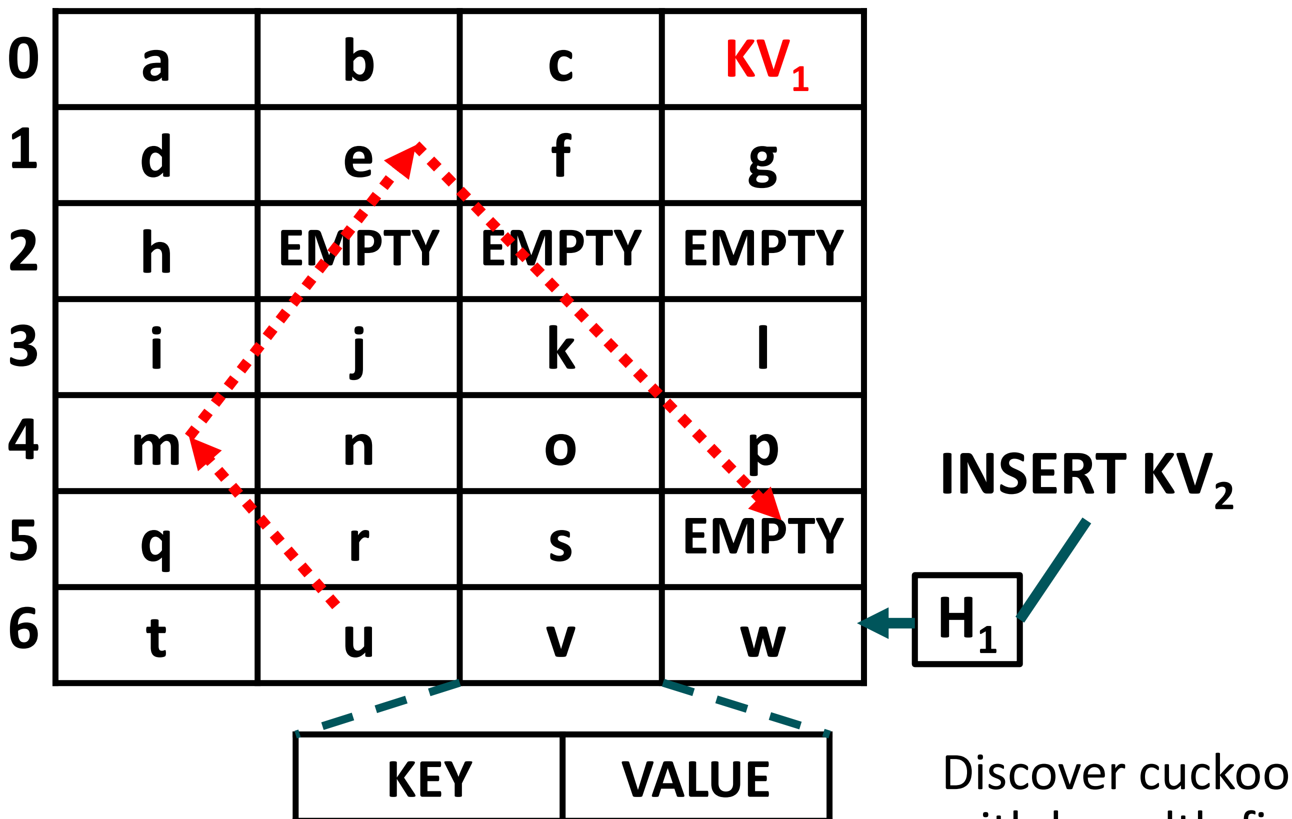
Discover cuckoo chain
with breadth-first search

BUCKETIZED CUCKOO HASH TABLES



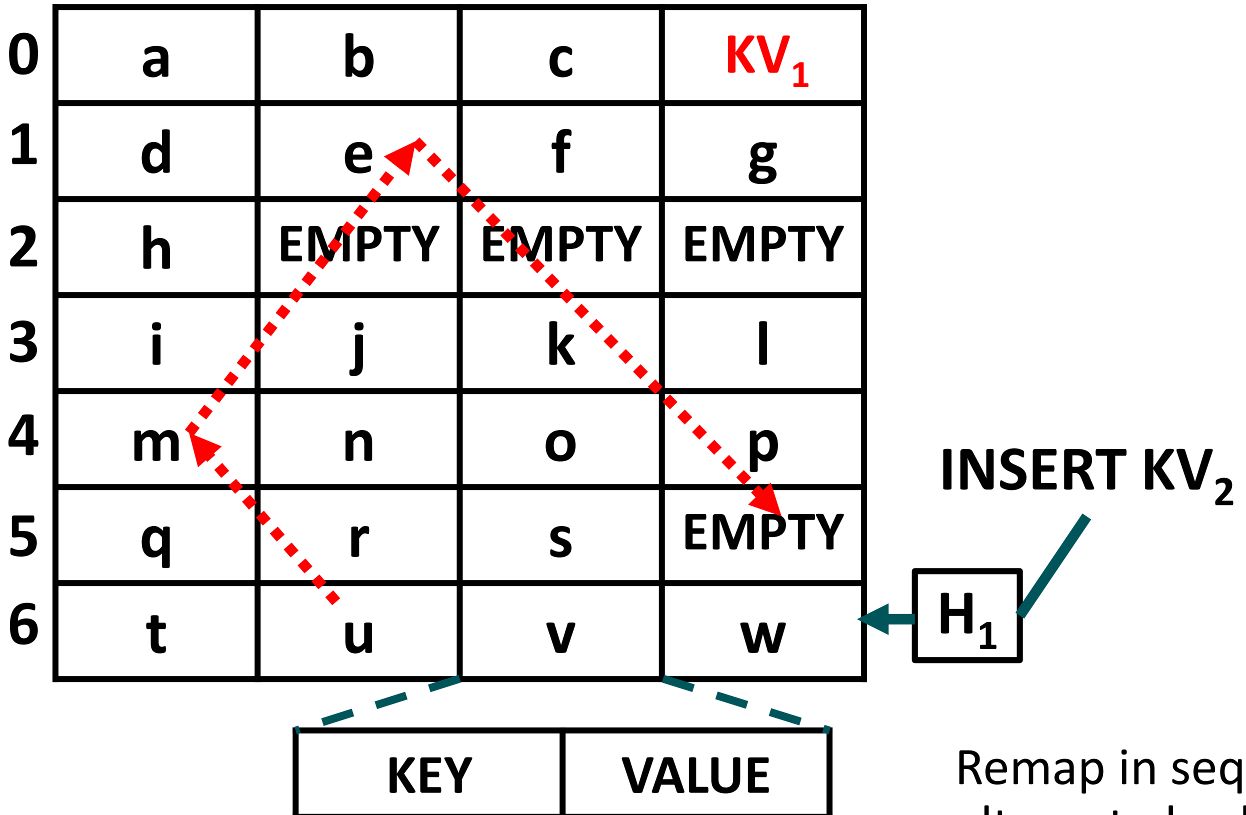
Discover cuckoo chain with breadth-first search

BUCKETIZED CUCKOO HASH TABLES



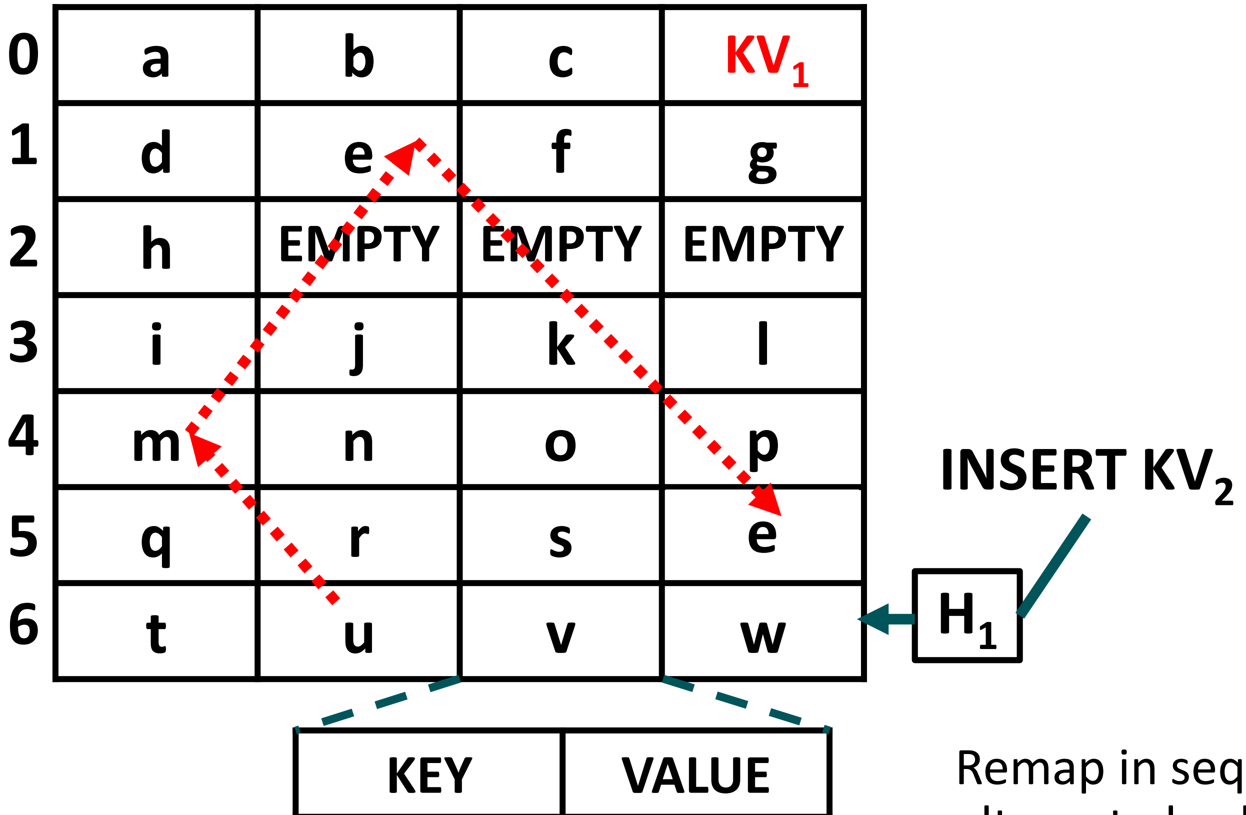
Discover cuckoo chain with breadth-first search

BUCKETIZED CUCKOO HASH TABLES



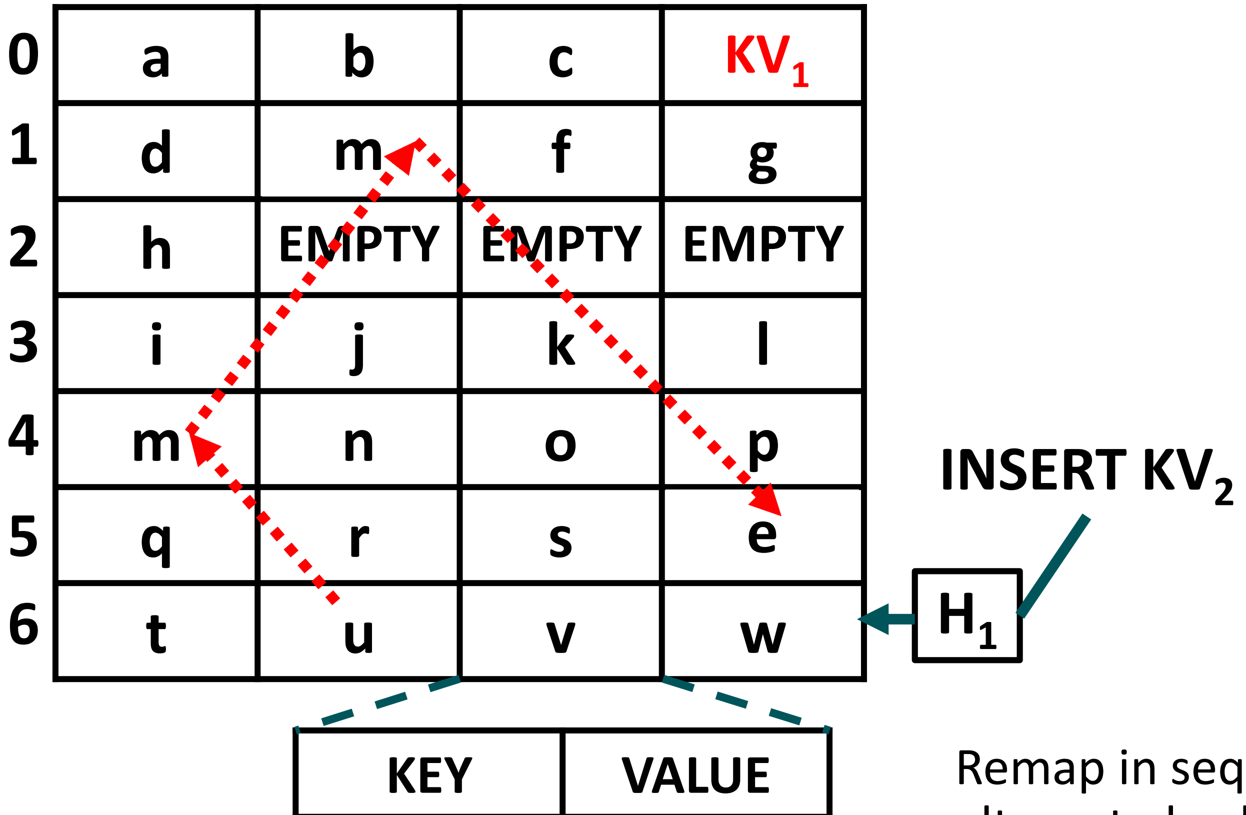
Remap in sequence with alternate hash function

BUCKETIZED CUCKOO HASH TABLES



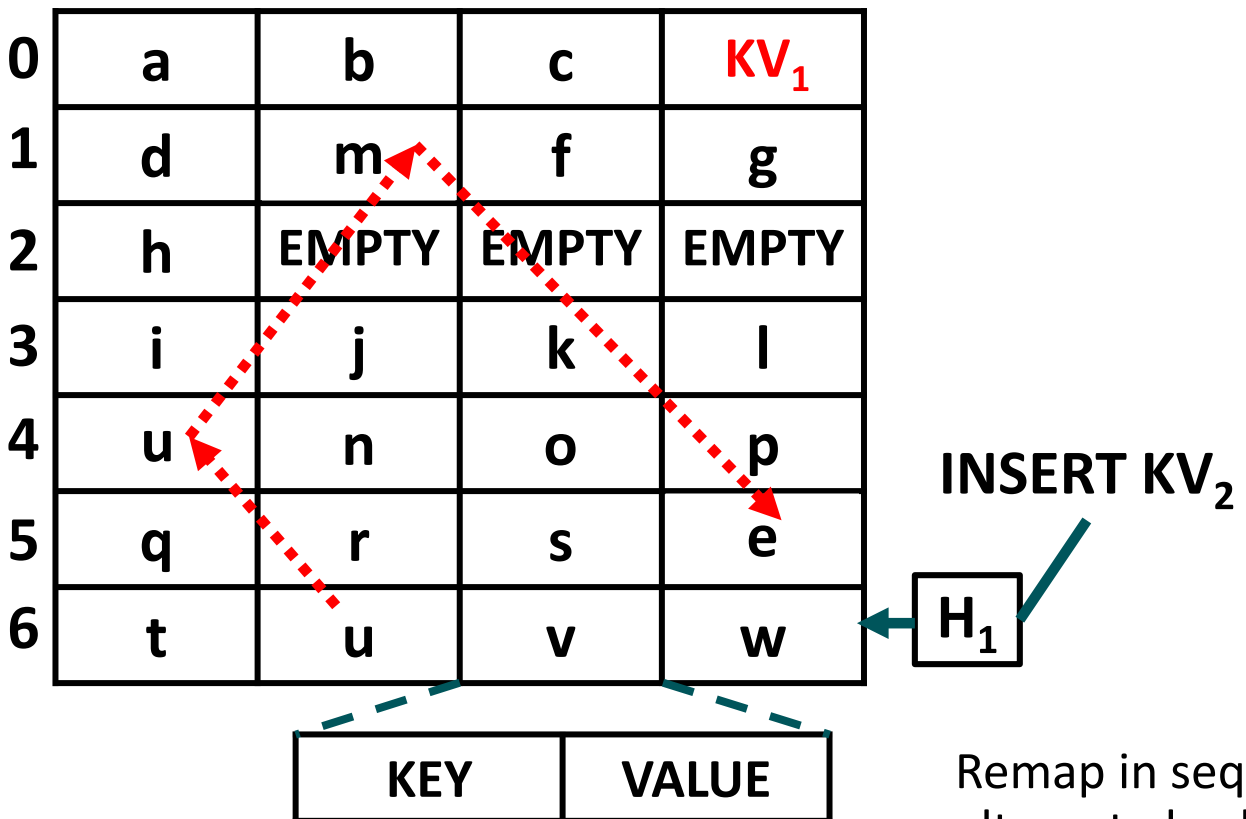
Remap in sequence with alternate hash function

BUCKETIZED CUCKOO HASH TABLES



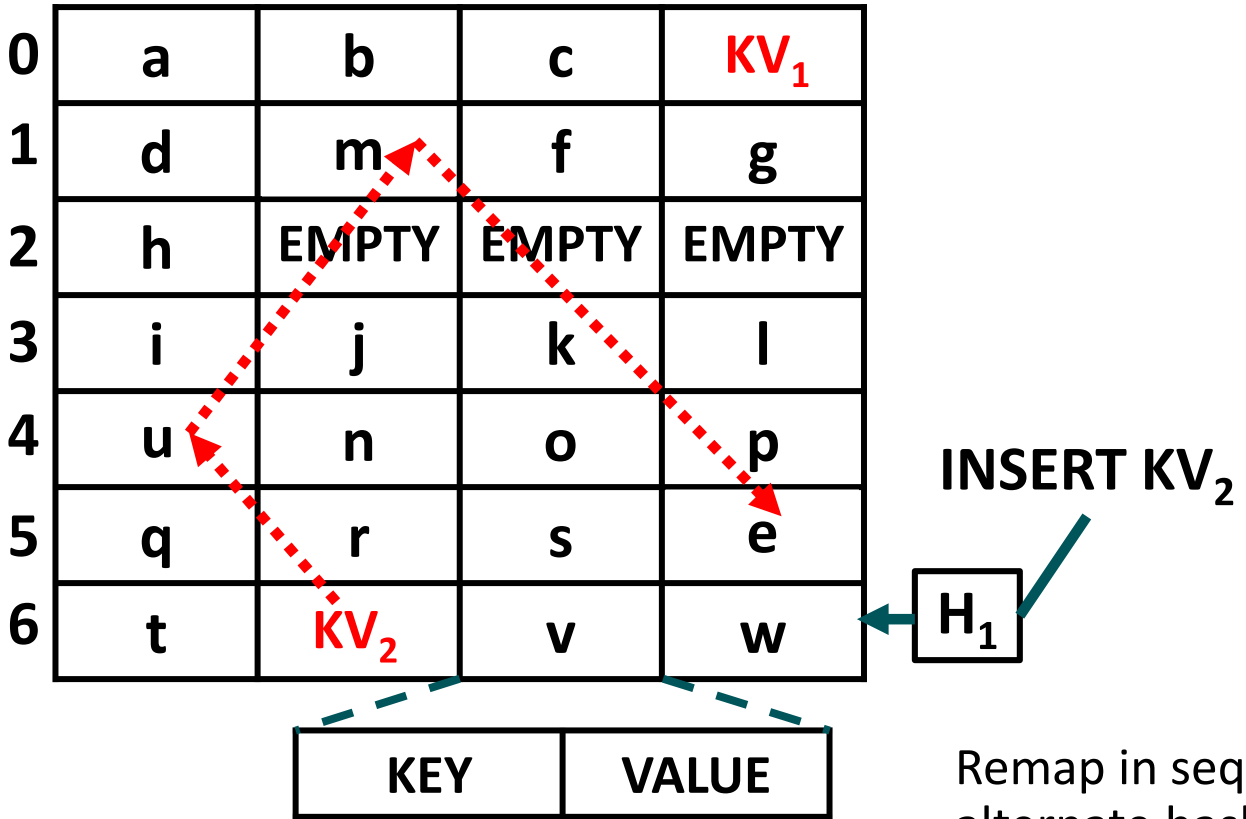
Remap in sequence with alternate hash function

BUCKETIZED CUCKOO HASH TABLES



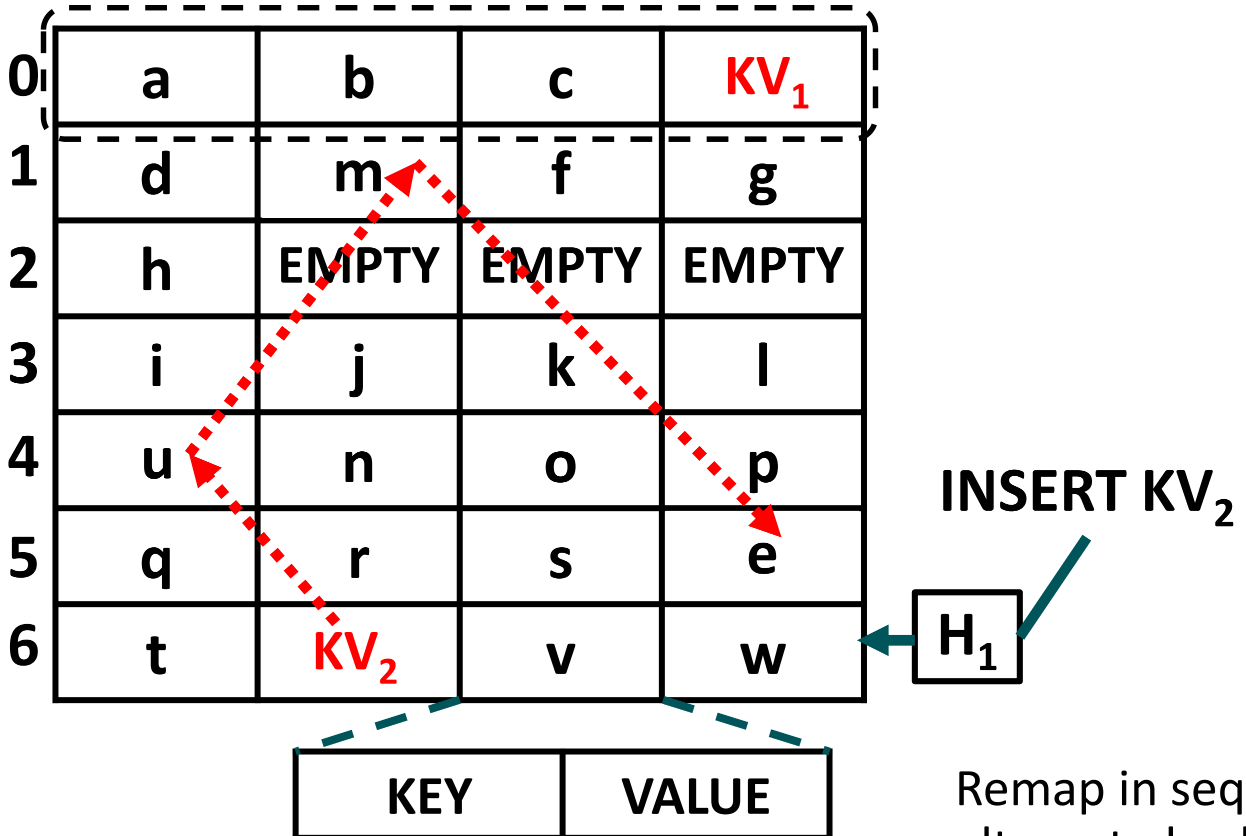
Remap in sequence with alternate hash function

BUCKETIZED CUCKOO HASH TABLES



Remap in sequence with alternate hash function

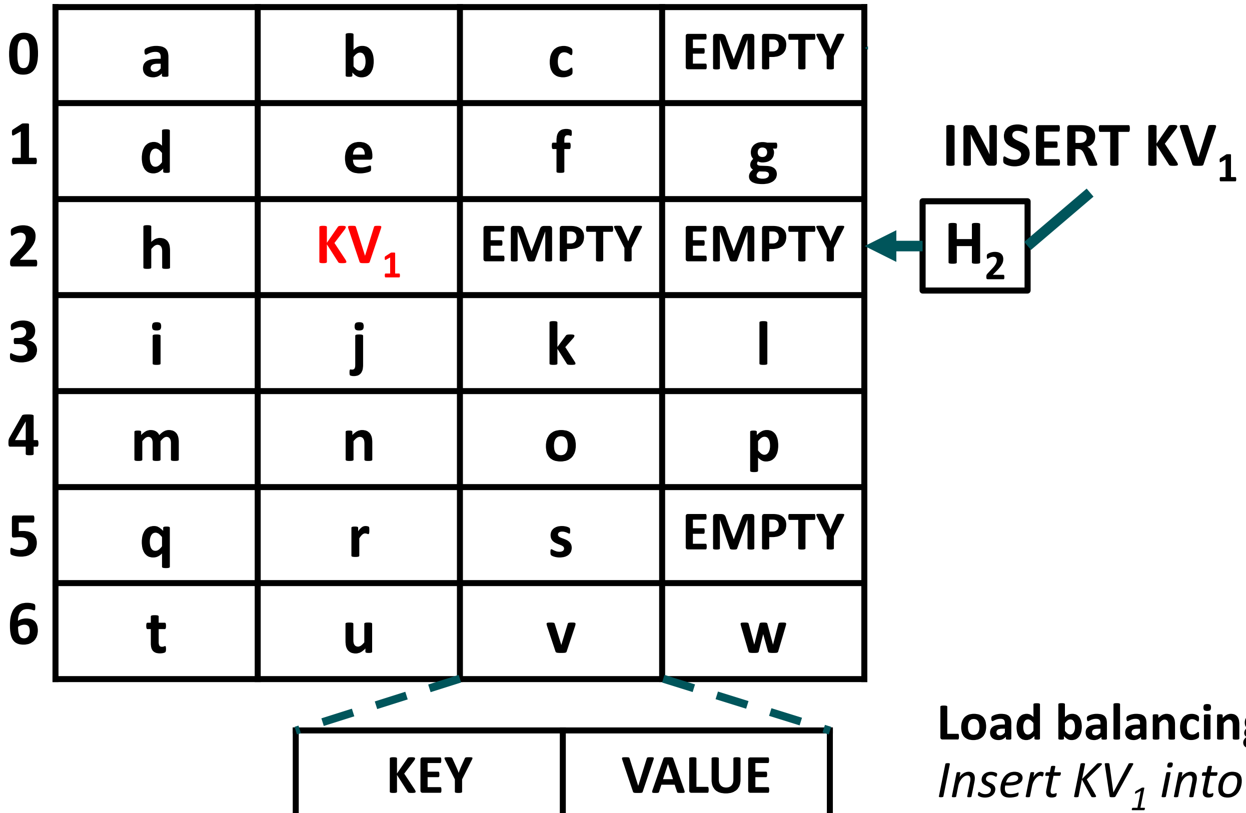
BUCKETIZED CUCKOO HASH TABLES



Remap in sequence with alternate hash function

- Each bucket is typically sized to one hardware cache line or less.
- Overwhelmingly, accesses to the bucket's cache line hit in the hardware caches during accesses to consecutive cells.

BUCKETIZED CUCKOO HASH TABLES
LOOKUPS AND LOAD BALANCING HEURISTIC

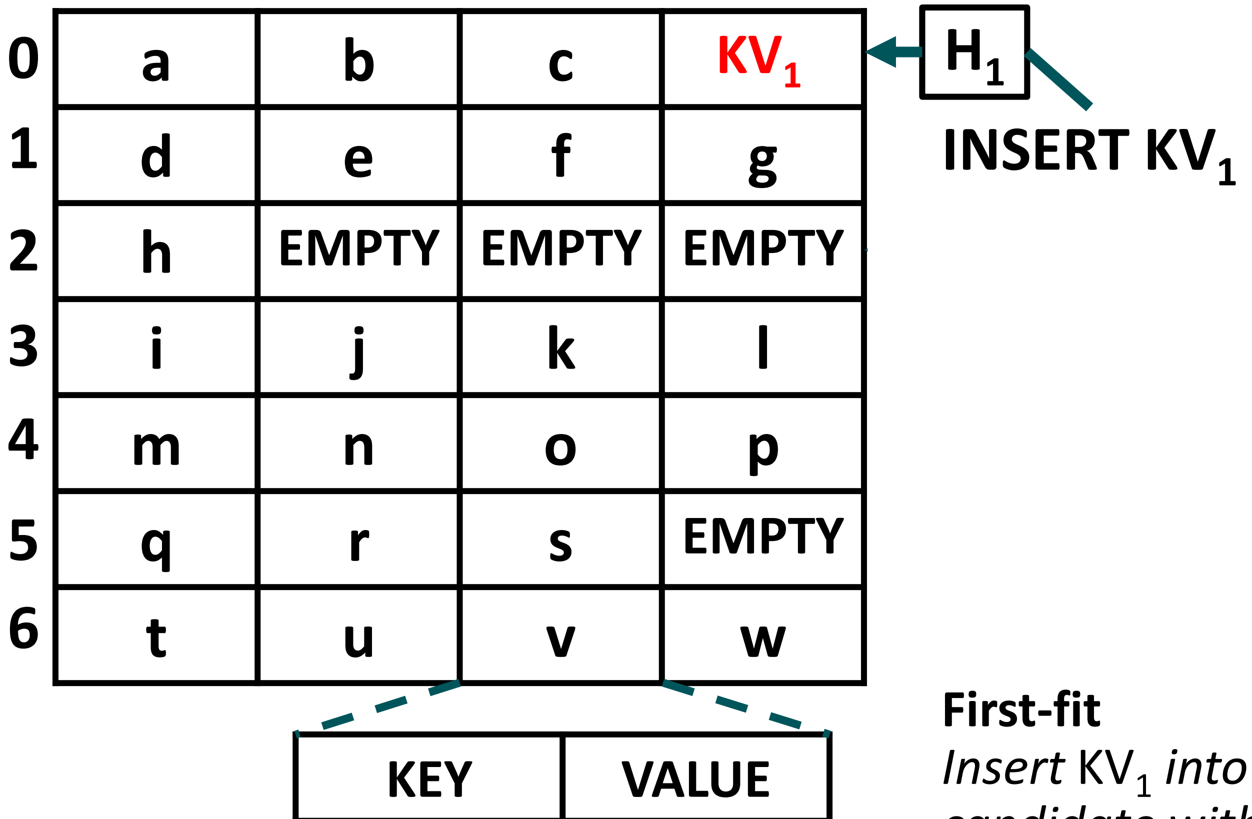


Load balancing
Insert KV_1 into the least full candidate bucket

- Expected Positive Lookup Cost Per Item in Buckets:
  1.5 = (0.5 Hashed by H_1) + 2 * (0.5 Hashed by H_2)
- Expected Negative Lookup Cost per Item in Buckets:
  2 (also worst-case)

BUCKETIZED CUCKOO HASH TABLES

LOOKUPS AND FIRST-FIT INSERTION HEURISTIC



First-fit
Insert KV_1 *into the first candidate with a free slot*

- Expected Positive Lookup Cost Per Item in Buckets:
  1 to 1.3ish depending on the table load factor and the slots per bucket
- Expected Negative Lookup Cost per Item in Buckets:
  2 (also worst-case)

BUCKETIZED CUCKOO HASH TABLES

BENEFITS OF FIRST-FIT

0	a	b	c	KV_1
1	d	m	f	g
2	h	EMPTY	EMPTY	EMPTY
3	i	j	k	l
4	u	n	o	p
5	q	r	s	e
6	t	KV_2	v	w

KEY	VALUE

BUCKETIZED CUCKOO HASH TABLES

BENEFITS OF FIRST-FIT

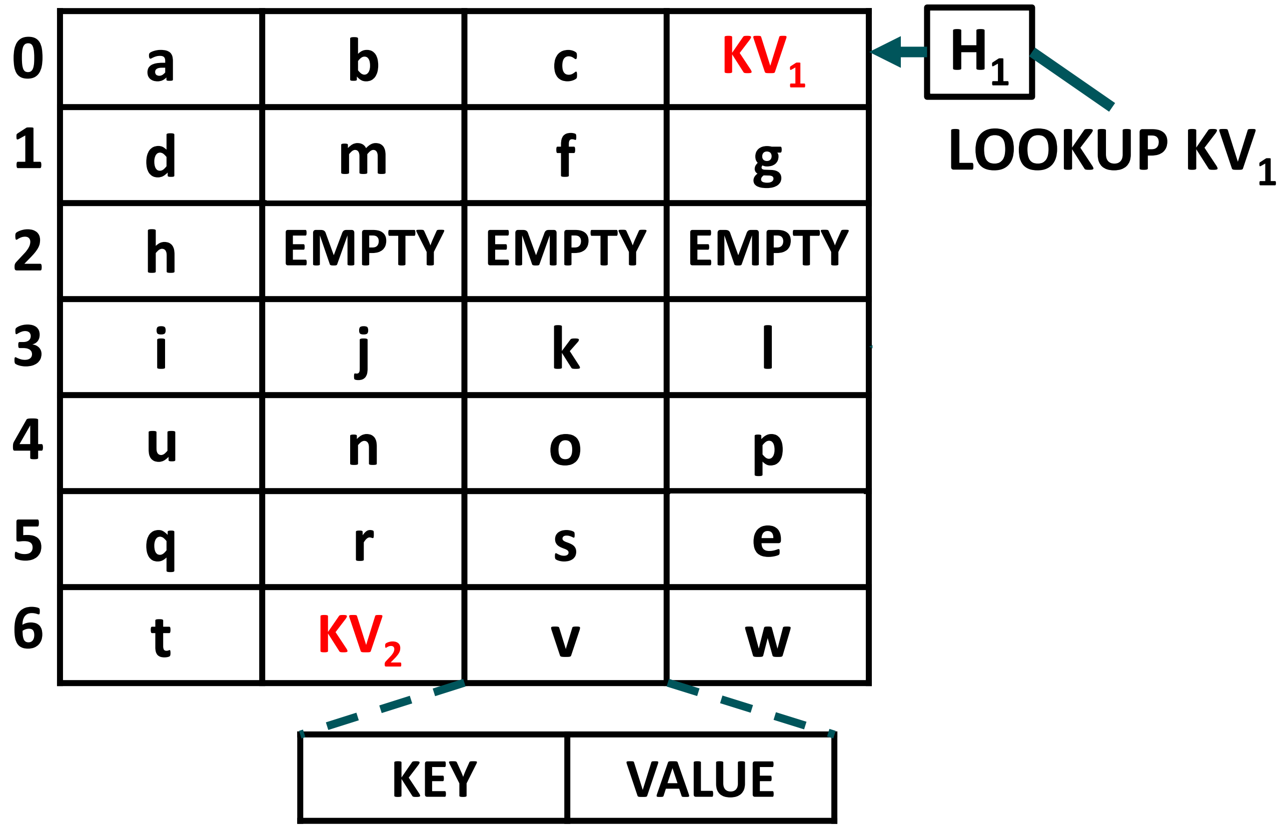
0	a	b	c	KV_1
1	d	m	f	g
2	h	EMPTY	EMPTY	EMPTY
3	i	j	k	l
4	u	n	o	p
5	q	r	s	e
6	t	KV_2	v	w

KEY	VALUE

LOOKUP KV_1

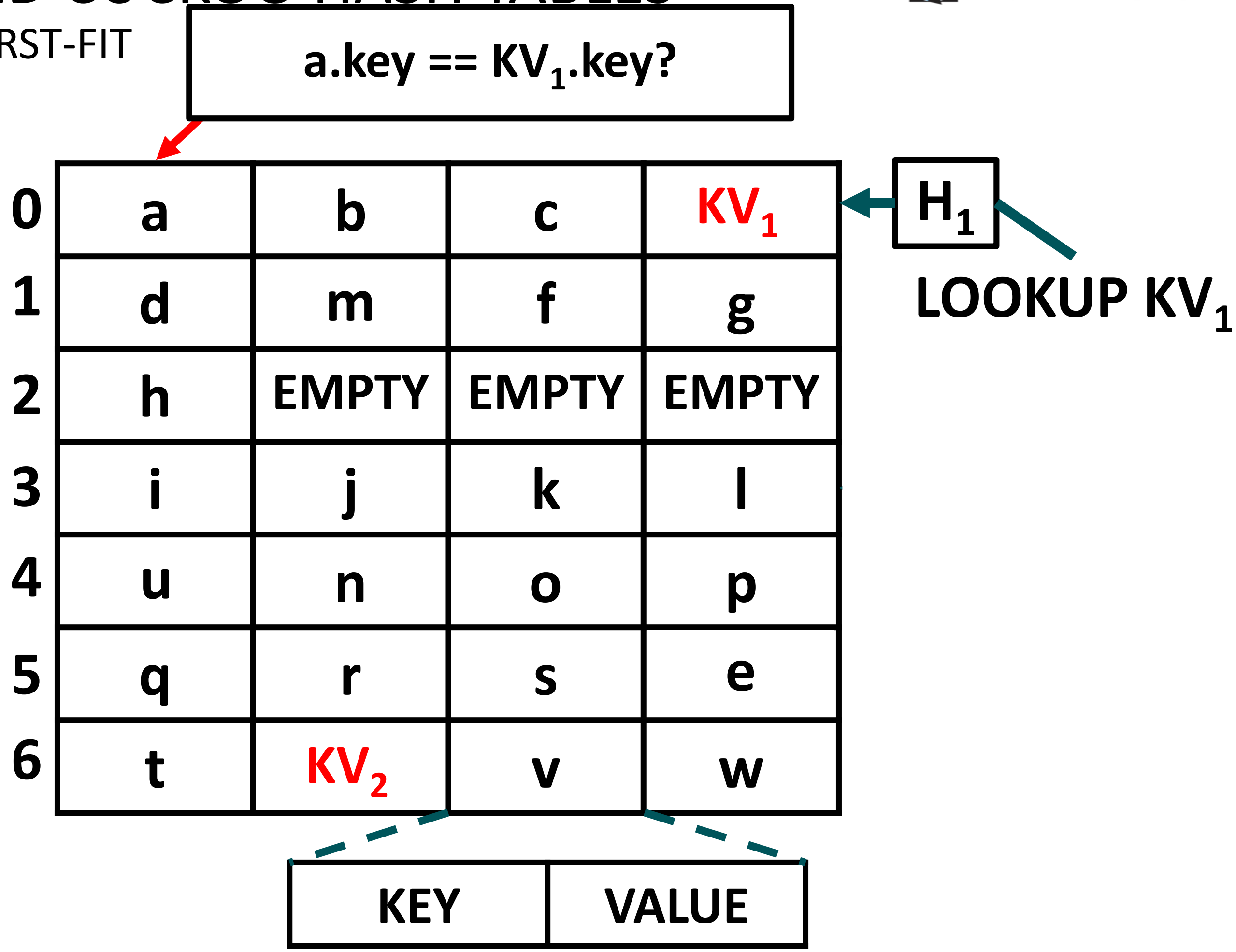
BUCKETIZED CUCKOO HASH TABLES

BENEFITS OF FIRST-FIT



BUCKETIZED CUCKOO HASH TABLES

BENEFITS OF FIRST-FIT



BUCKETIZED CUCKOO HASH TABLES

BENEFITS OF FIRST-FIT

a.key == KV_1.key?

0	a	b	c	KV_1
1	d	m	f	g
2	h	EMPTY	EMPTY	EMPTY
3	i	j	k	l
4	u	n	o	p
5	q	r	s	e
6	t	KV_2	v	w

H_1

LOOKUP KV_1

KEY	VALUE

BUCKETIZED CUCKOO HASH TABLES

BENEFITS OF FIRST-FIT

b .key == KV_1.key?

0	a	b	c	KV_1
1	d	m	f	g
2	h	EMPTY	EMPTY	EMPTY
3	i	j	k	l
4	u	n	o	p
5	q	r	s	e
6	t	KV_2	v	w

H_1

LOOKUP KV_1

KEY	VALUE

BUCKETIZED CUCKOO HASH TABLES

BENEFITS OF FIRST-FIT

b .key == KV_1.key?

0	a	b	c	KV_1
1	d	m	f	g
2	h	EMPTY	EMPTY	EMPTY
3	i	j	k	l
4	u	n	o	p
5	q	r	s	e
6	t	KV_2	v	w

H_1

LOOKUP KV_1

KEY	VALUE

BUCKETIZED CUCKOO HASH TABLES

BENEFITS OF FIRST-FIT

c .key == KV_1.key?

0	a	b	c	KV_1
1	d	m	f	g
2	h	EMPTY	EMPTY	EMPTY
3	i	j	k	l
4	u	n	o	p
5	q	r	s	e
6	t	KV_2	v	w

H_1

LOOKUP KV_1

KEY	VALUE

BUCKETIZED CUCKOO HASH TABLES

BENEFITS OF FIRST-FIT

c .key == KV_1.key?

0	a	b	c	KV_1
1	d	m	f	g
2	h	EMPTY	EMPTY	EMPTY
3	i	j	k	l
4	u	n	o	p
5	q	r	s	e
6	t	KV_2	v	w

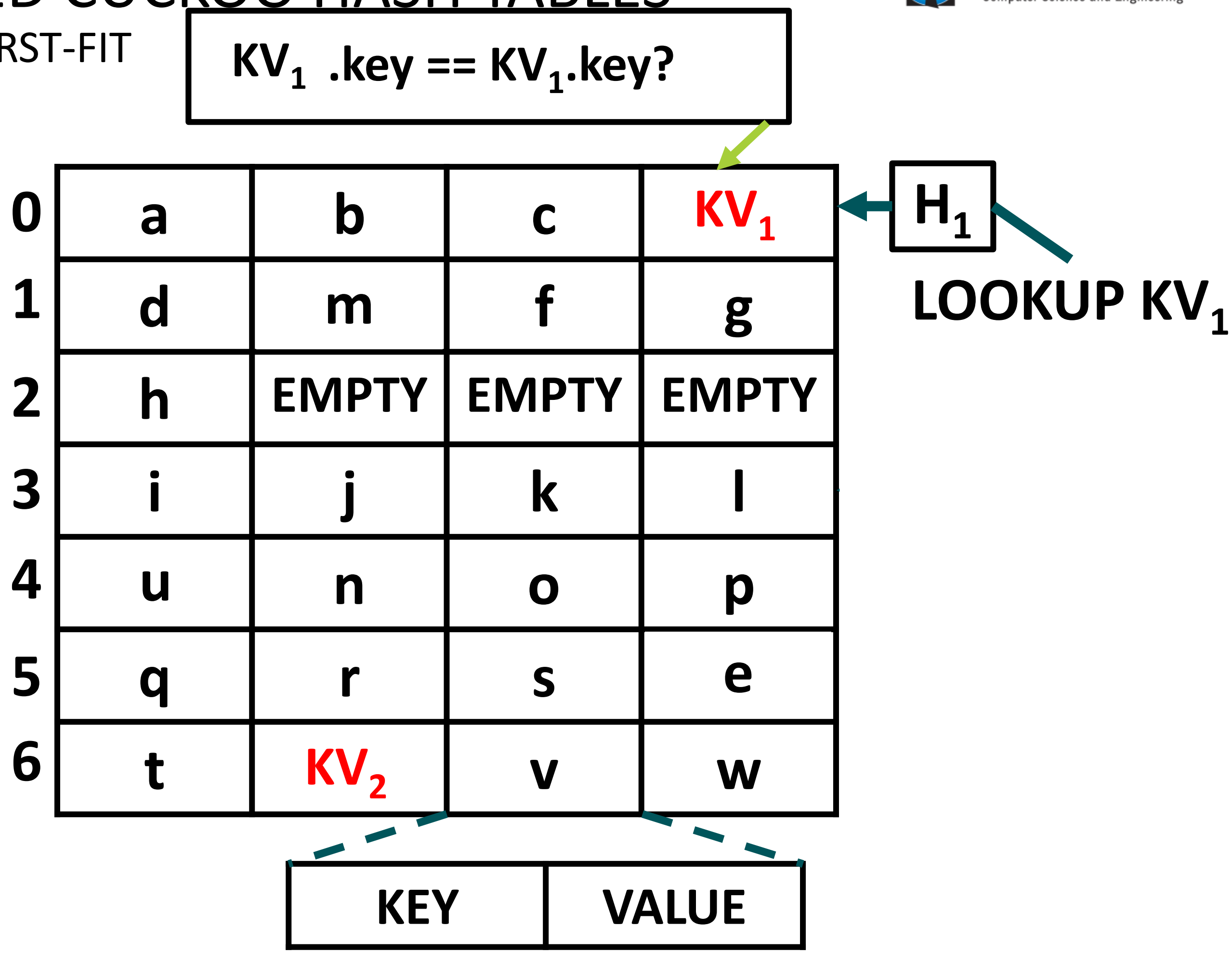
H_1

LOOKUP KV_1

KEY	VALUE

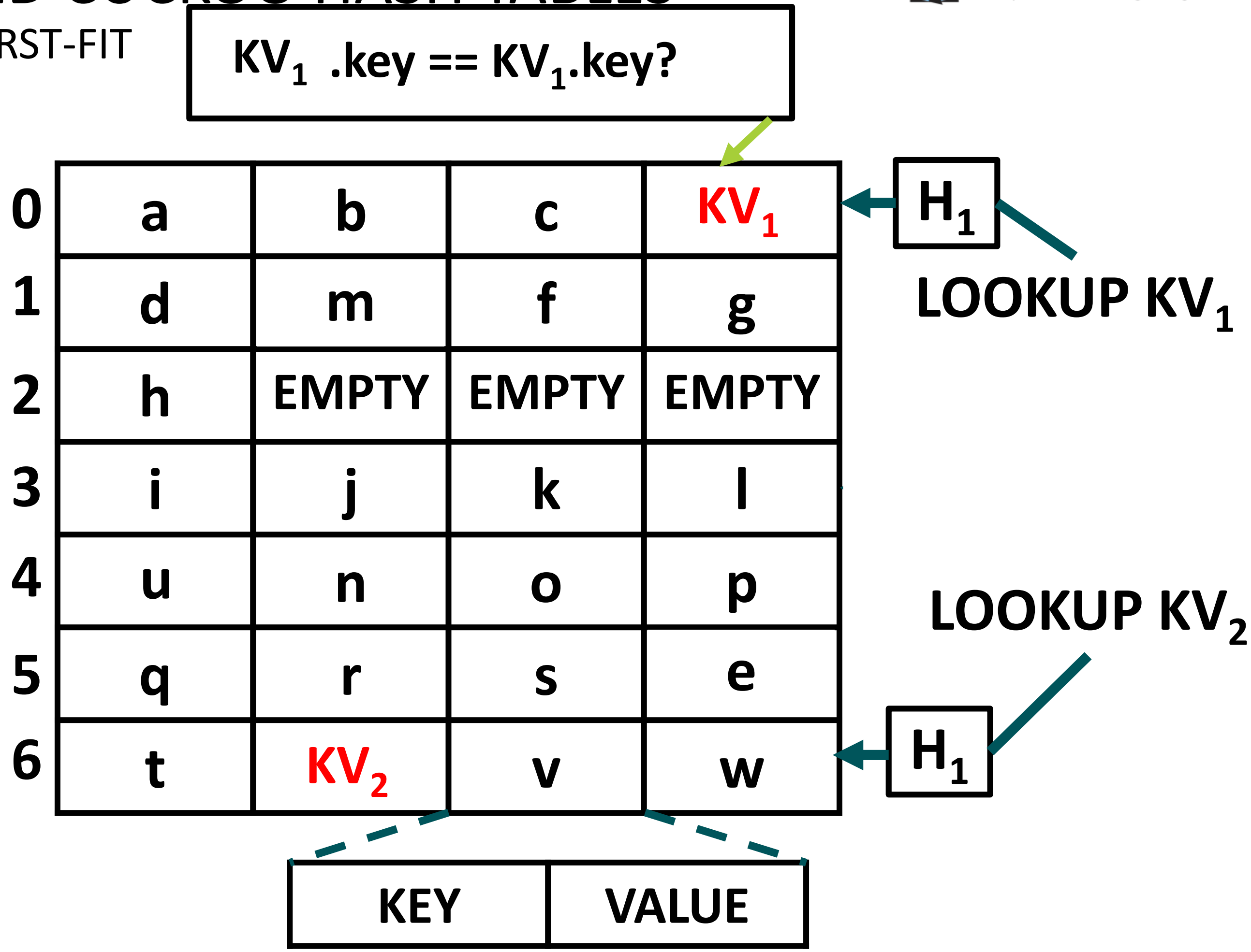
BUCKETIZED CUCKOO HASH TABLES

BENEFITS OF FIRST-FIT



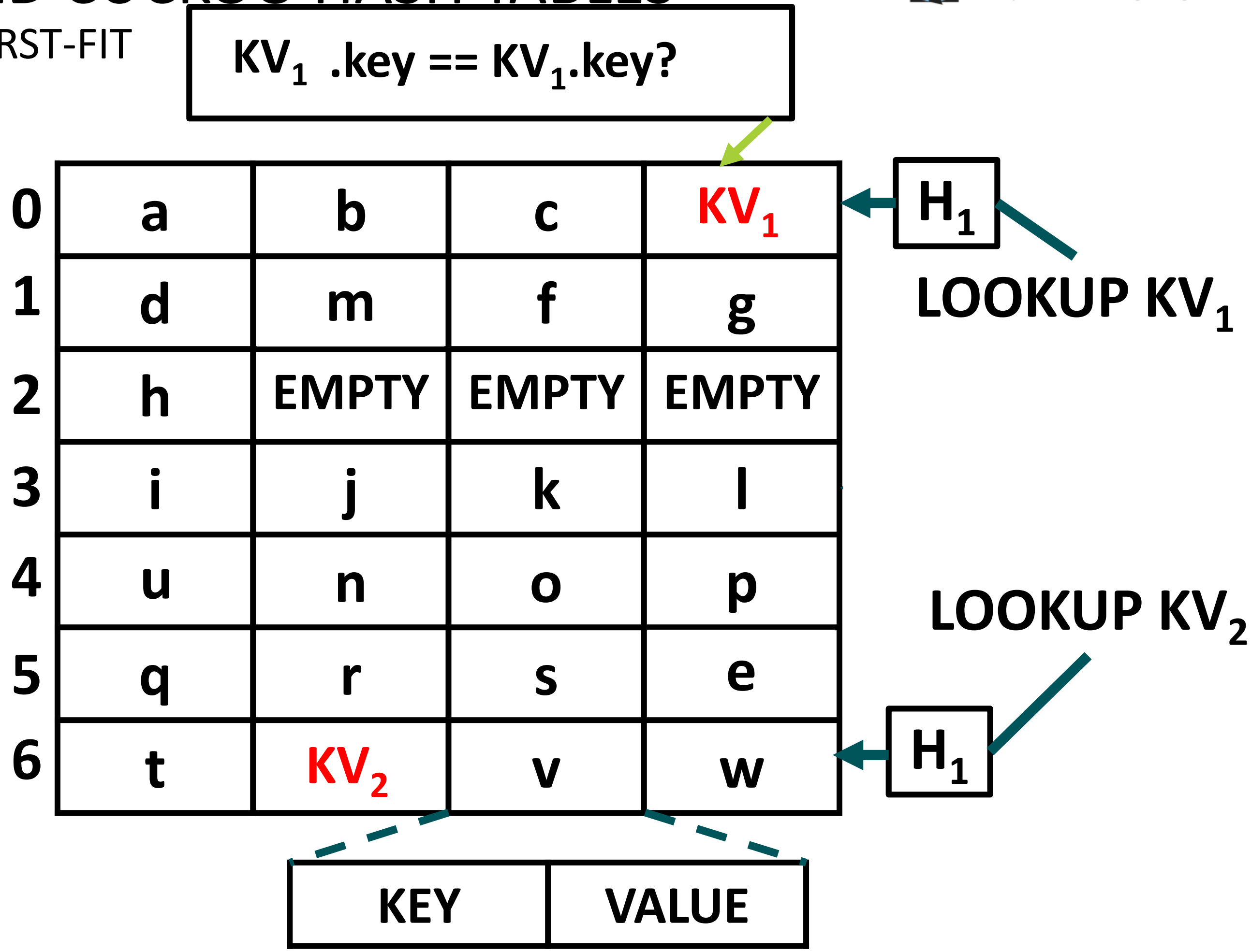
BUCKETIZED CUCKOO HASH TABLES

BENEFITS OF FIRST-FIT



BUCKETIZED CUCKOO HASH TABLES

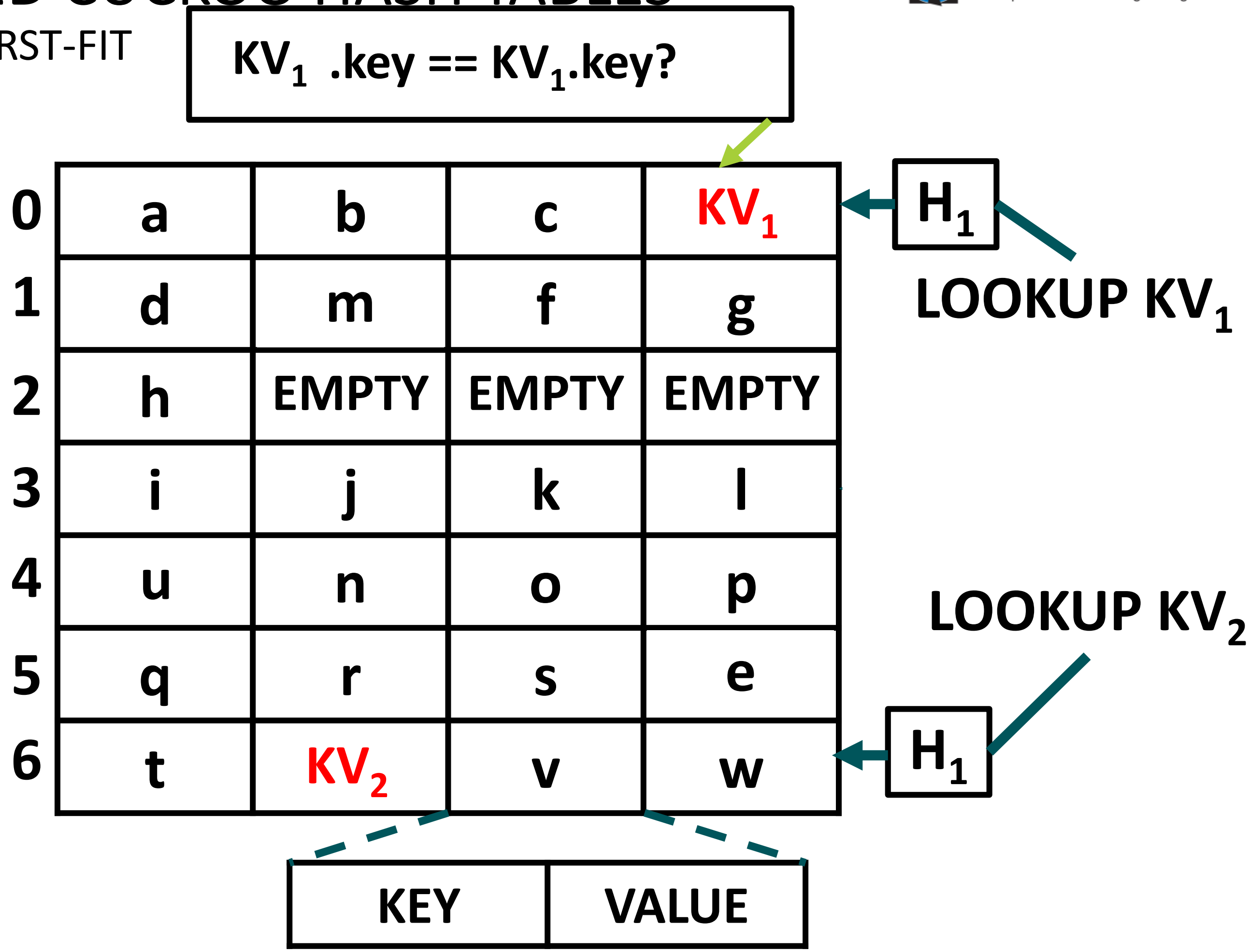
BENEFITS OF FIRST-FIT



- Positive Lookups:
  - First-fit gets us most of the way to 1.0 on positive lookups because most elements are hashed with H_1

BUCKETIZED CUCKOO HASH TABLES

BENEFITS OF FIRST-FIT



- Positive Lookups:
  - First-fit gets us most of the way to 1.0 on positive lookups because most elements are hashed with H_1
- But…

BUCKETIZED CUCKOO HASH TABLES

LIMITATIONS OF FIRST-FIT

0	a	b	c	KV_1
1	d	m	f	g
2	h	EMPTY	EMPTY	EMPTY
3	i	j	k	l
4	u	n	o	p
5	q	r	s	e
6	t	KV_2	v	w

KEY	VALUE

BUCKETIZED CUCKOO HASH TABLES
LIMITATIONS OF FIRST-FIT

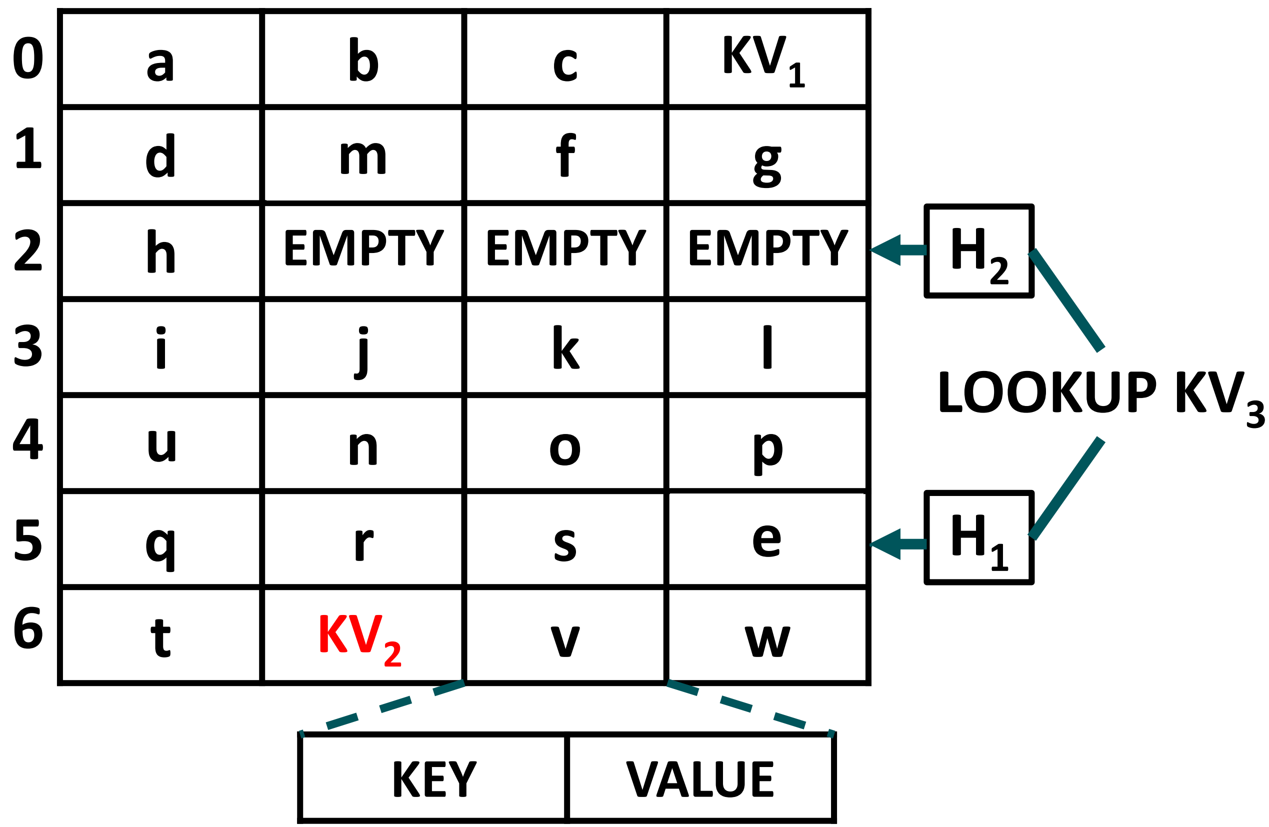
0	a	b	c	KV_1
1	d	m	f	g
2	h	EMPTY	EMPTY	EMPTY
3	i	j	k	l
4	u	n	o	p
5	q	r	s	e
6	t	KV_2	v	w

KEY	VALUE

- Expected Negative Lookup Cost per Item in Buckets:
  - First-fit doesn't address the comparatively expensive negative lookup cost. We still need to check all candidate buckets.

BUCKETIZED CUCKOO HASH TABLES

LIMITATIONS OF FIRST-FIT



- Expected Negative Lookup Cost per Item in Buckets:
  - First-fit doesn't address the comparatively expensive negative lookup cost. We still need to check all candidate buckets.

HORTON TABLES
DESIGN GOALS

HORTON TABLES
DESIGN GOALS

- Positive lookups that typically require accessing only 1 bucket per query
  - If buckets are at most a cache line in size, then only 1 cache line is accessed as well.

HORTON TABLES
DESIGN GOALS

- Positive lookups that typically require accessing only 1 bucket per query
  - If buckets are at most a cache line in size, then only 1 cache line is accessed as well.

- Negative lookups that typically require accessing only 1 bucket per query
  - If buckets are at most a cache line in size, then only 1 cache line is accessed as well.

HORTON TABLES
DESIGN GOALS

- Positive lookups that typically require accessing only 1 bucket per query
  - If buckets are at most a cache line in size, then only 1 cache line is accessed as well.

- Negative lookups that typically require accessing only 1 bucket per query
  - If buckets are at most a cache line in size, then only 1 cache line is accessed as well.

- Retain a worst-case lookup cost of 2 buckets (i.e., often 2 hardware cache lines)

HORTON TABLES
DESIGN GOALS

- Positive lookups that typically require accessing only 1 bucket per query
  - If buckets are at most a cache line in size, then only 1 cache line is accessed as well.

- Negative lookups that typically require accessing only 1 bucket per query
  - If buckets are at most a cache line in size, then only 1 cache line is accessed as well.

- Retain a worst-case lookup cost of 2 buckets (i.e., often 2 hardware cache lines)

- Achieve a load factor exceeding 0.95 (akin to a bucketized cuckoo hash table that uses 2 hash functions and 4-cell buckets)

HORTON TABLES

PRIMARY INSERTIONS AND LOOKUPS

0	8	5	EMPTY	EMPTY
1	33	EMPTY	15	2
2	35	18	22	EMPTY
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

HORTON TABLES

PRIMARY INSERTIONS AND LOOKUPS

0	8	5	EMPTY	EMPTY
1	33	EMPTY	15	2
2	35	18	22	EMPTY
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

- Horton tables start off as standard bucketized cuckoo hash tables

HORTON TABLES

PRIMARY INSERTIONS AND LOOKUPS

0	8	5	EMPTY	EMPTY
1	33	EMPTY	15	2
2	35	18	22	EMPTY
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

- Horton tables start off as standard bucketized cuckoo hash tables
- Like first-fit, they strongly bias inserts by using a *primary hash function* called $H_{primary}$

HORTON TABLES

PRIMARY INSERTIONS AND LOOKUPS

0	8	5	EMPTY	EMPTY
1	33	EMPTY	15	2
2	35	18	22	EMPTY
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

- Horton tables start off as standard bucketized cuckoo hash tables
- Like first-fit, they strongly bias inserts by using a *primary hash function* called $H_{primary}$
- Most positive lookups therefore only require accessing a single cache line

HORTON TABLES

PRIMARY INSERTIONS AND LOOKUPS

0	8	5	EMPTY	EMPTY
1	33	EMPTY	15	2
2	35	18	22	EMPTY
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

$H_{primary}$

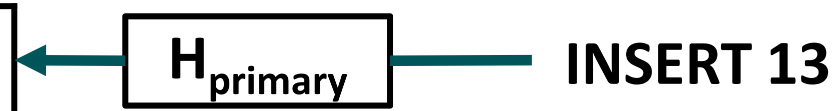
INSERT 13

- Horton tables start off as standard bucketized cuckoo hash tables
- Like first-fit, they strongly bias inserts by using a *primary hash function* called $H_{primary}$
- Most positive lookups therefore only require accessing a single cache line

HORTON TABLES

PRIMARY INSERTIONS AND LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	EMPTY
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY



- Horton tables start off as standard bucketized cuckoo hash tables
- Like first-fit, they strongly bias inserts by using a *primary hash function* called $H_{primary}$
- Most positive lookups therefore only require accessing a single cache line

HORTON TABLES

PRIMARY INSERTIONS AND LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	EMPTY
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

- Horton tables start off as standard bucketized cuckoo hash tables
- Like first-fit, they strongly bias inserts by using a *primary hash function* called $H_{primary}$
- Most positive lookups therefore only require accessing a single cache line

HORTON TABLES

PRIMARY INSERTIONS AND LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	EMPTY
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

$H_{primary}$ ← INSERT 16

- Horton tables start off as standard bucketized cuckoo hash tables
- Like first-fit, they strongly bias inserts by using a *primary hash function* called $H_{primary}$
- Most positive lookups therefore only require accessing a single cache line

HORTON TABLES

PRIMARY INSERTIONS AND LOOKUPS

0	**8**	**5**	**13**	**EMPTY**
1	**33**	**EMPTY**	**15**	**2**
2	**35**	**18**	**22**	**16**
3	**EMPTY**	**EMPTY**	**EMPTY**	**37**
4	**17**	**6**	**21**	**EMPTY**
5	**9**	**24**	**EMPTY**	**EMPTY**

$H_{primary}$ ← **INSERT 16**

- Horton tables start off as standard bucketized cuckoo hash tables
- Like first-fit, they strongly bias inserts by using a *primary hash function* called $H_{primary}$
- Most positive lookups therefore only require accessing a single cache line

HORTON TABLES

PRIMARY INSERTIONS AND LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	16
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

- Horton tables start off as standard bucketized cuckoo hash tables
- Like first-fit, they strongly bias inserts by using a *primary hash function* called $H_{primary}$
- Most positive lookups therefore only require accessing a single cache line

HORTON TABLES

PRIMARY INSERTIONS AND LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	16
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

$H_{primary}$

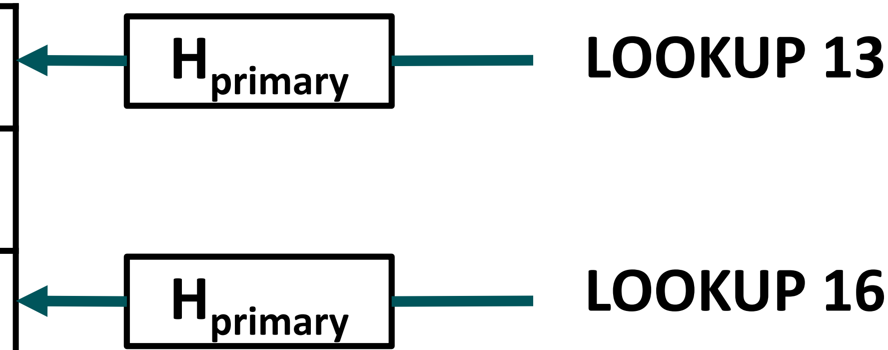
LOOKUP 13

- Horton tables start off as standard bucketized cuckoo hash tables
- Like first-fit, they strongly bias inserts by using a *primary hash function* called $H_{primary}$
- Most positive lookups therefore only require accessing a single cache line

HORTON TABLES

PRIMARY INSERTIONS AND LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	16
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY



- Horton tables start off as standard bucketized cuckoo hash tables
- Like first-fit, they strongly bias inserts by using a *primary hash function* called $H_{primary}$
- Most positive lookups therefore only require accessing a single cache line

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	16
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	16
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

$H_{primary}$

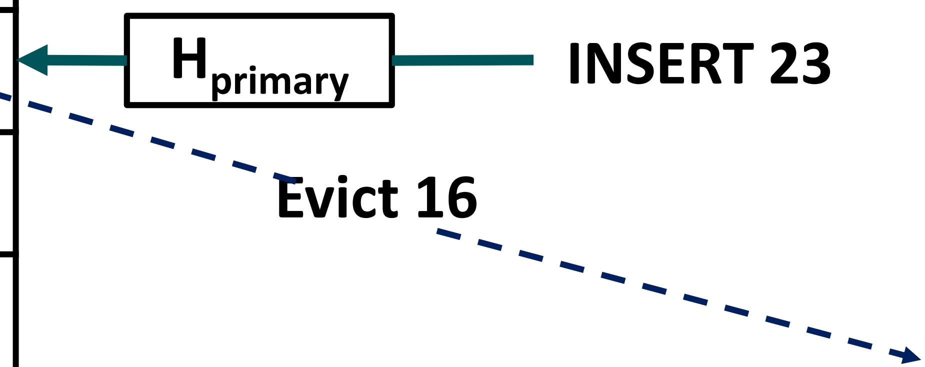
INSERT 23

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	16
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

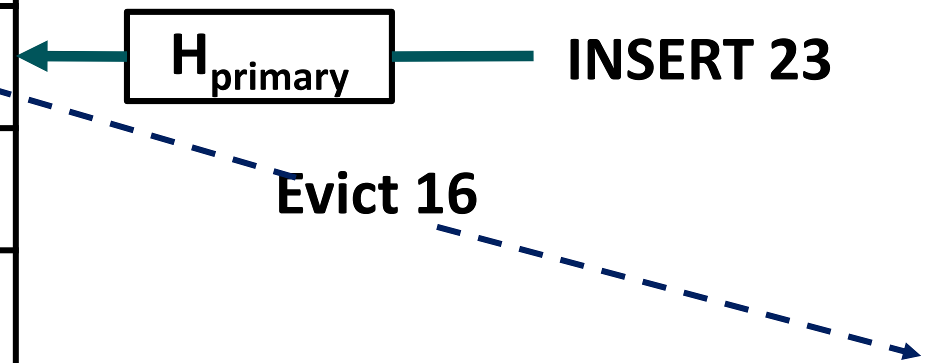


- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

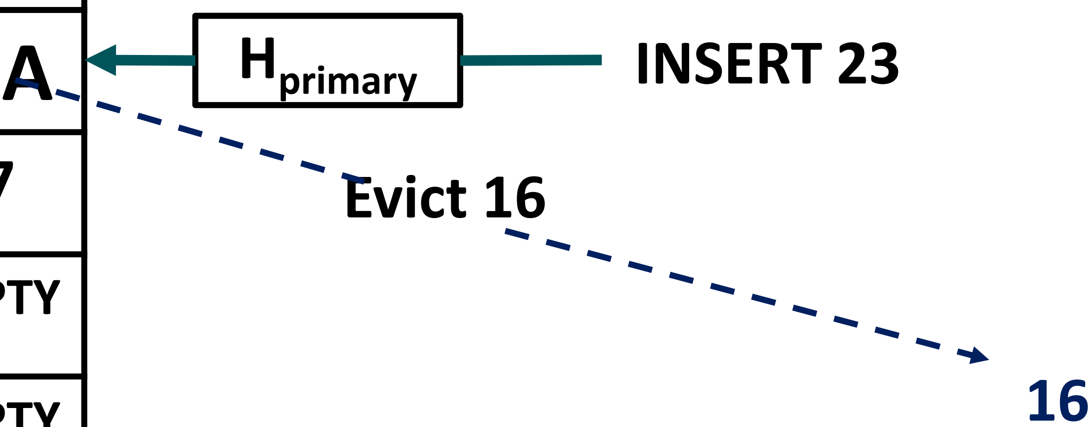


- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	**8**	**5**	**13**	**EMPTY**
1	**33**	**EMPTY**	**15**	**2**
2	**35**	**18**	**22**	**REA**
3	**EMPTY**	**EMPTY**	**EMPTY**	**37**
4	**17**	**6**	**21**	**EMPTY**
5	**9**	**24**	**EMPTY**	**EMPTY**



- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	**8**	**5**	**13**	**EMPTY**
1	**33**	**EMPTY**	**15**	**2**
2	**35**	**18**	**22**	**REA**
3	**EMPTY**	**EMPTY**	**EMPTY**	**37**
4	**17**	**6**	**21**	**EMPTY**
5	**9**	**24**	**EMPTY**	**EMPTY**

16

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

INSERT 23

16

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

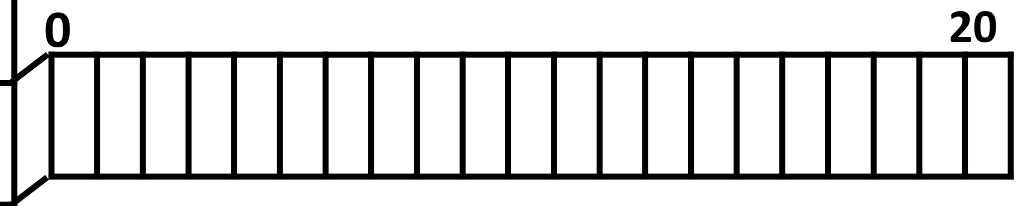
16

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY



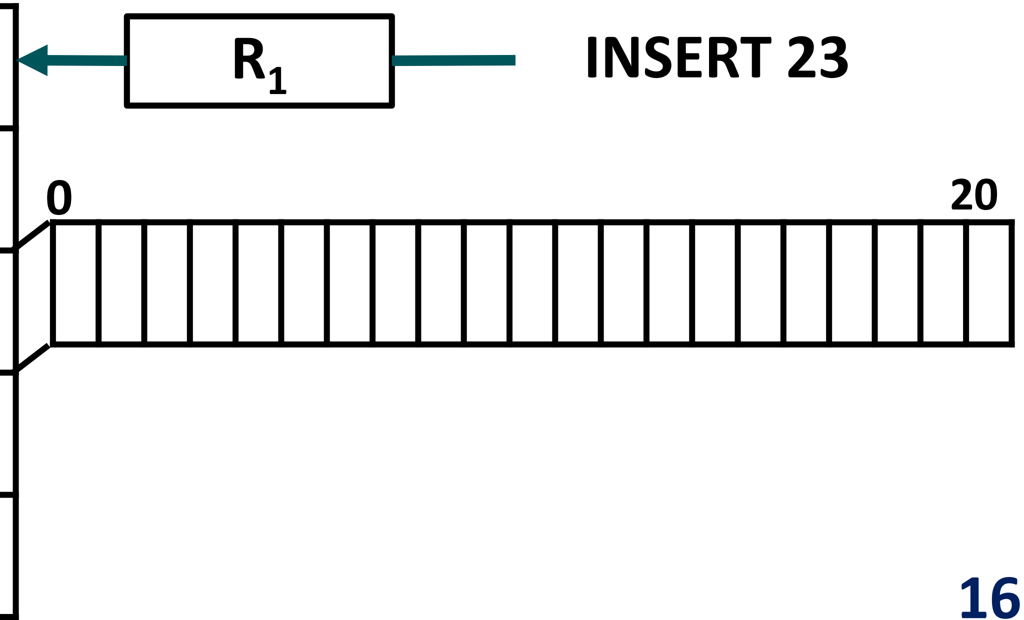
16

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY

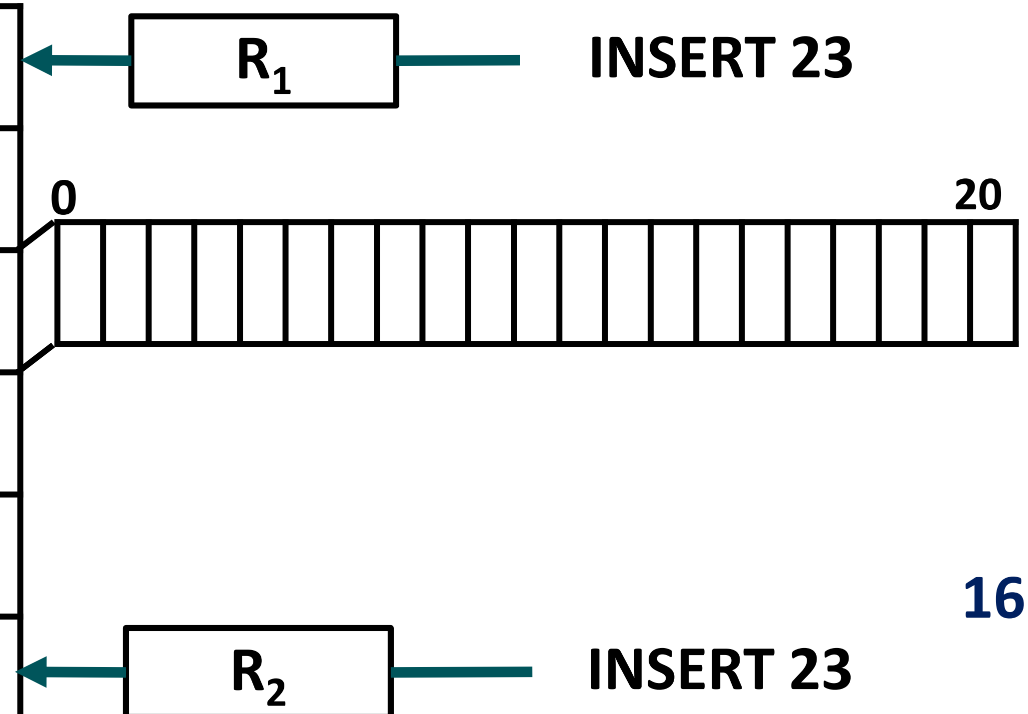


- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

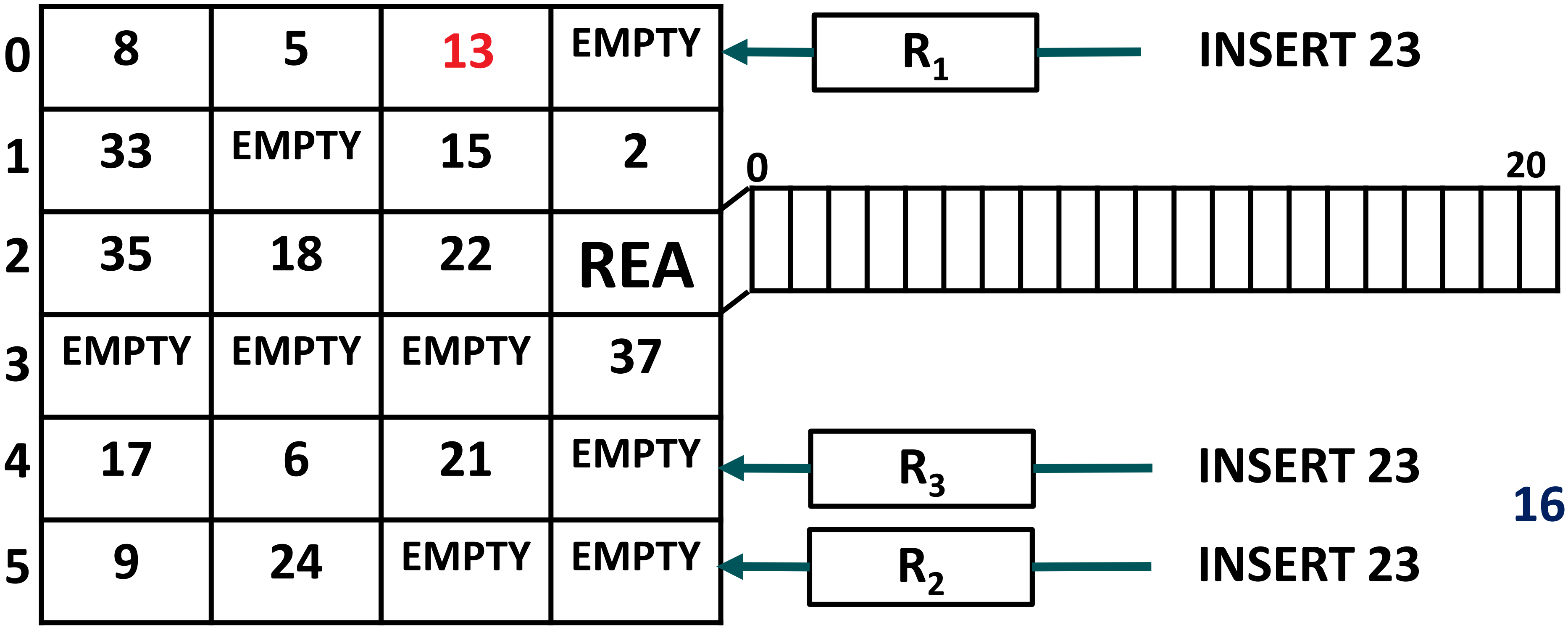
0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY



- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

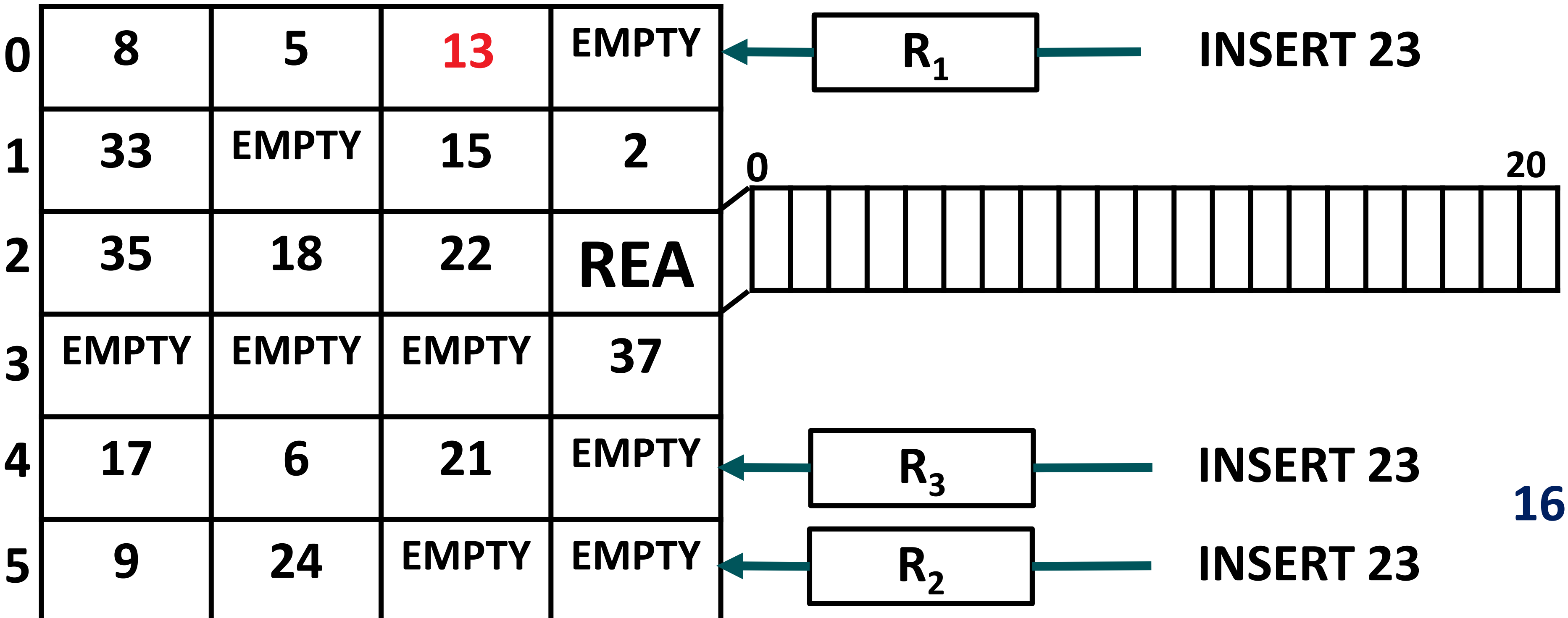
INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



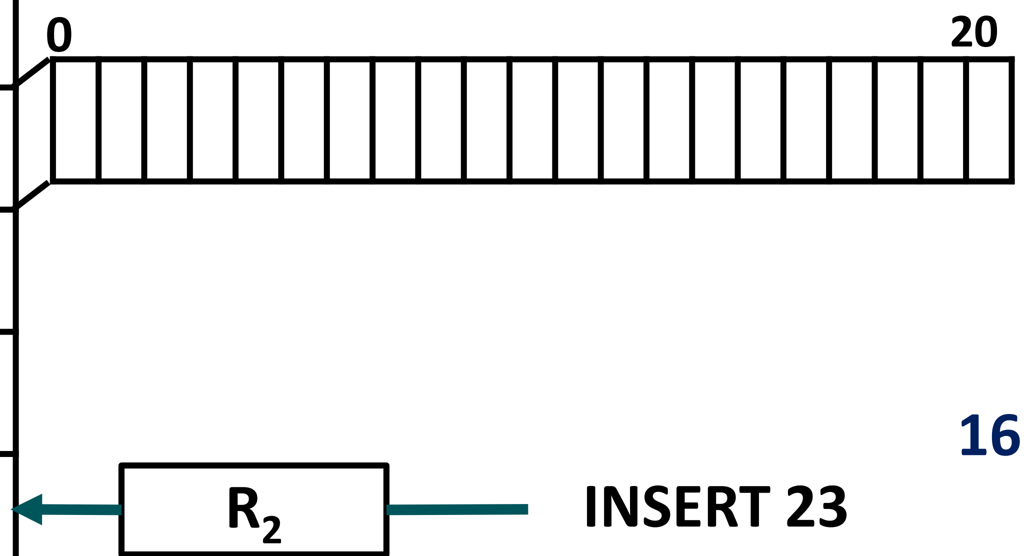
Use R_2 for inserting 23 because it maps 23 to least full secondary bucket candidate.

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	EMPTY	EMPTY



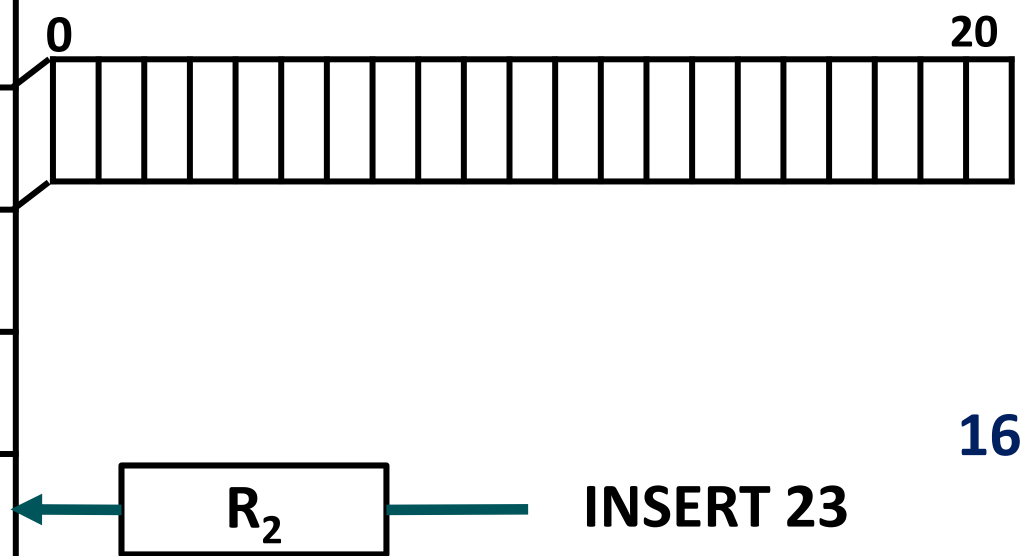
Use R_2 for inserting 23 because it maps 23 to least full secondary bucket candidate.

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

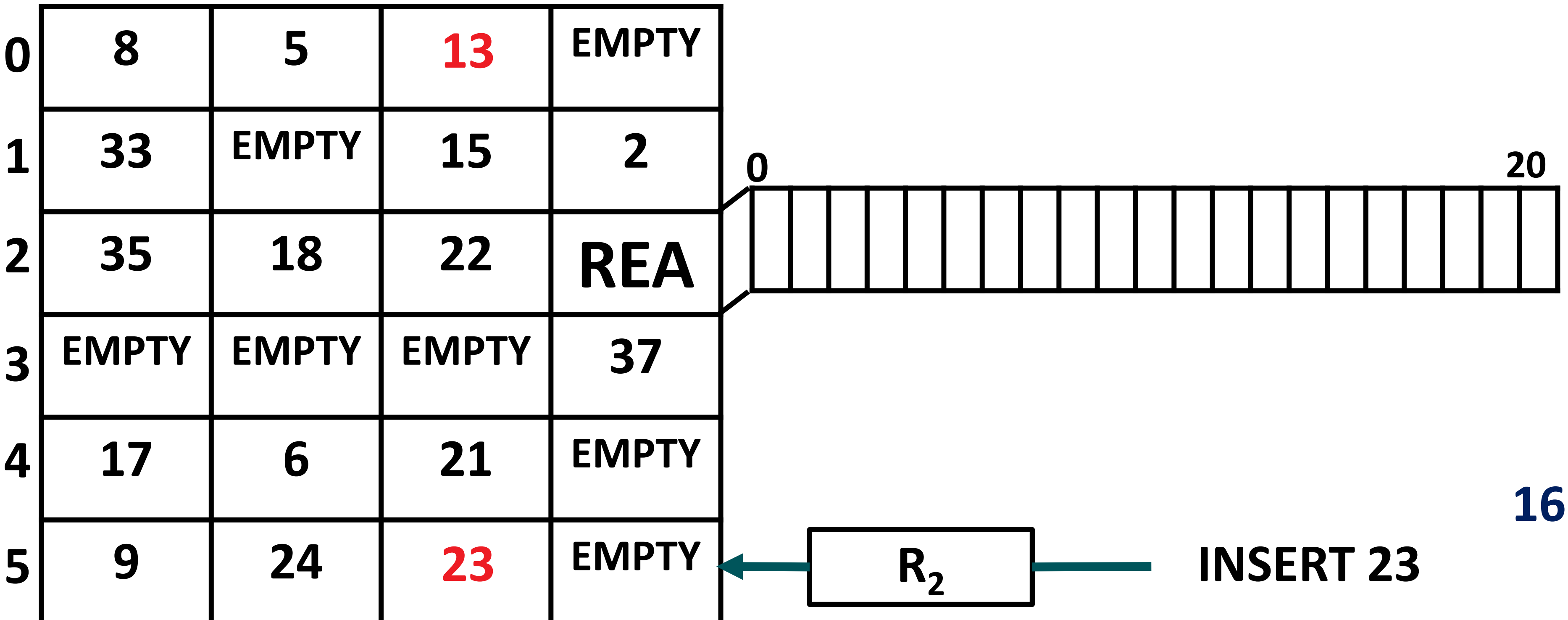


Use R_2 for inserting 23 because it maps 23 to least full secondary bucket candidate.

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



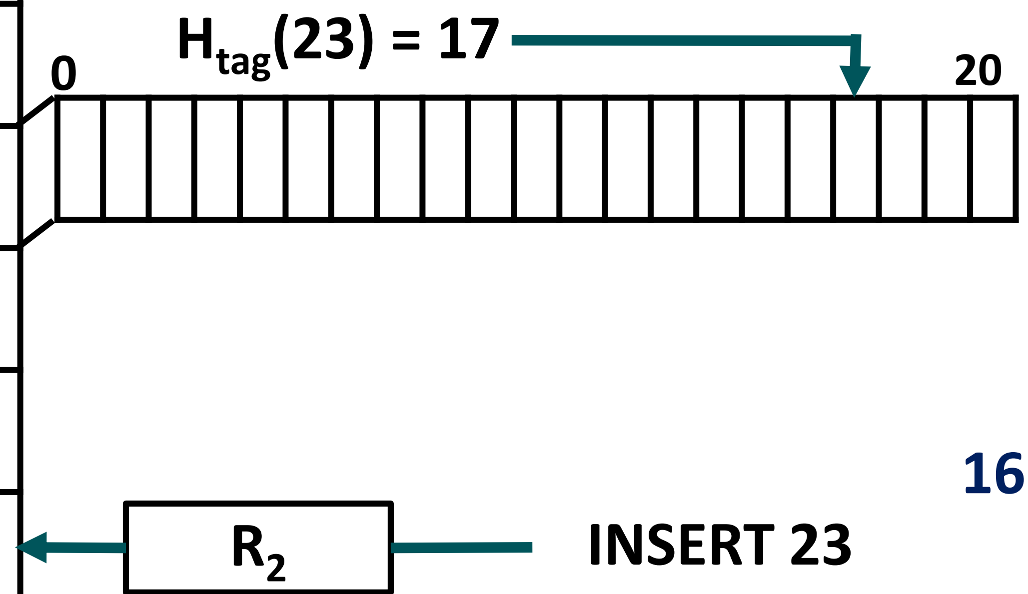
Compute index into remap entry array using H_{tag} with key as input

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



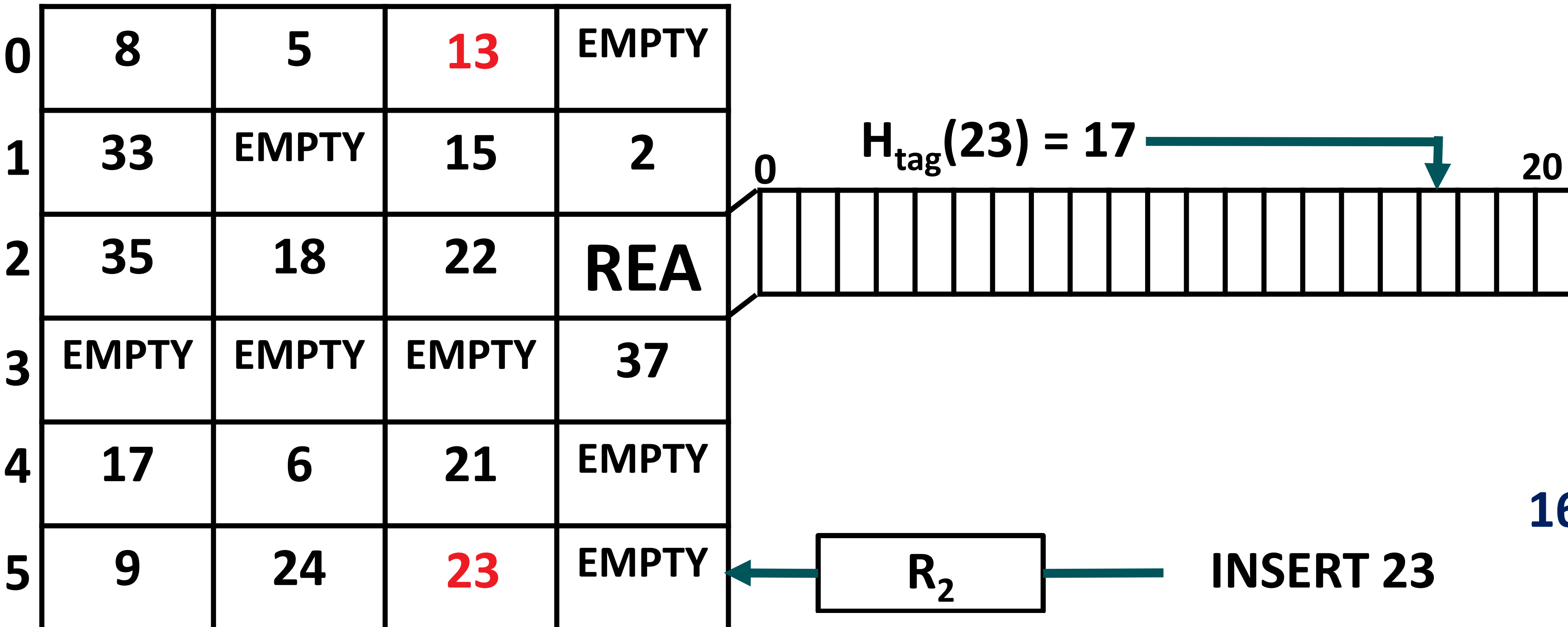
16

Compute index into remap entry array using H_{tag} with key as input

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



16

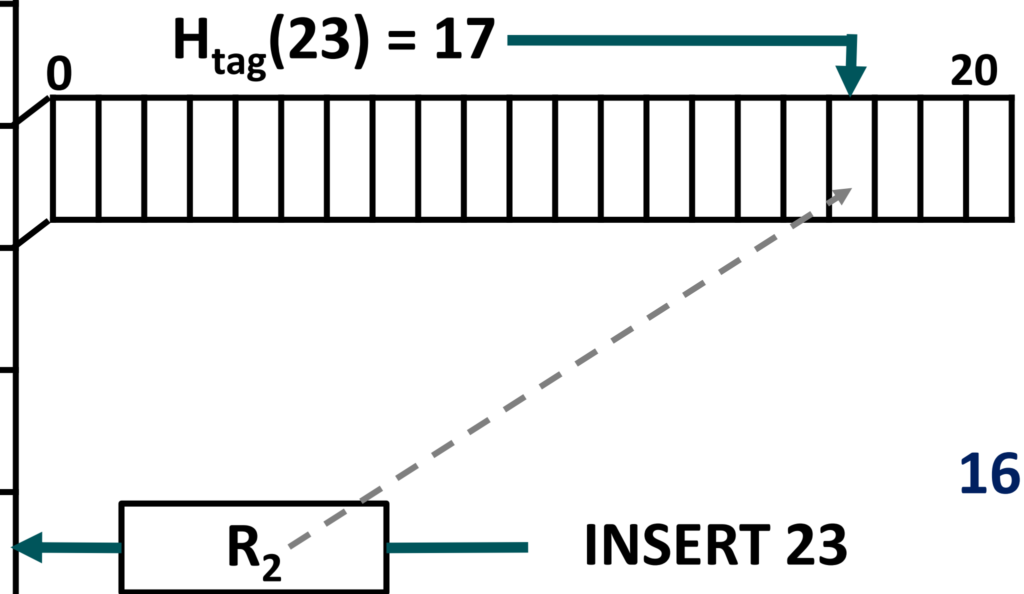
Store 2 at index $H_{tag}(23)=17$ to indicate that R_2 was used to remap 23

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



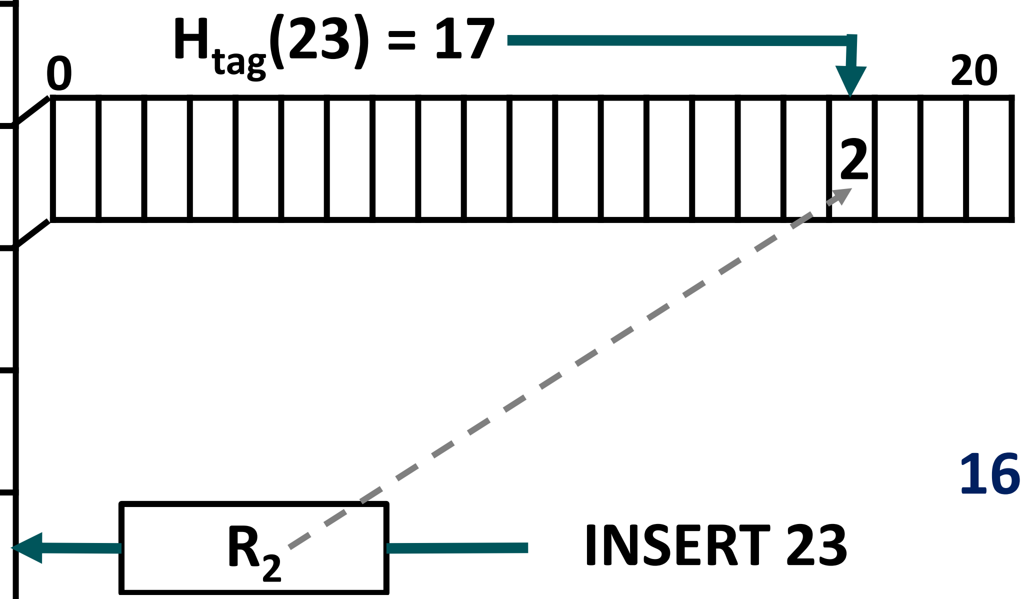
Store 2 at index $H_{tag}(23)=17$ to indicate that R_2 was used to remap 23

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



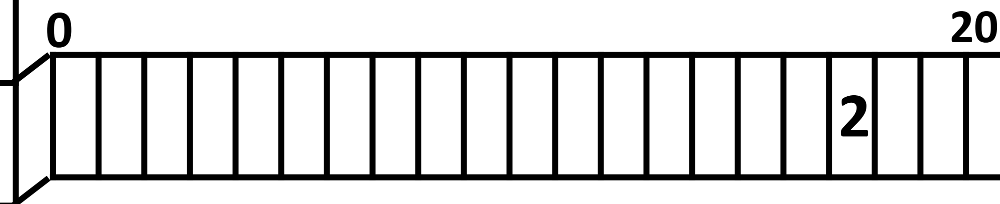
Store 2 at index $H_{tag}(23)=17$ to indicate that R_2 was used to remap 23

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



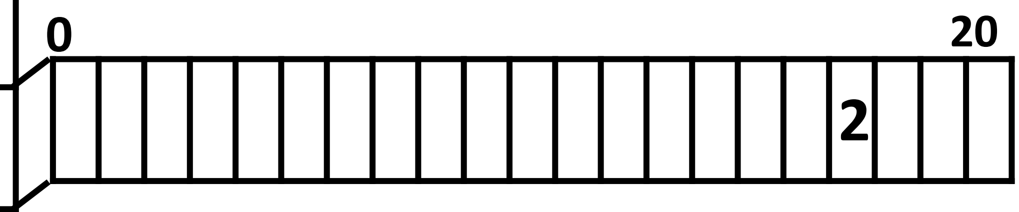
16

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



16

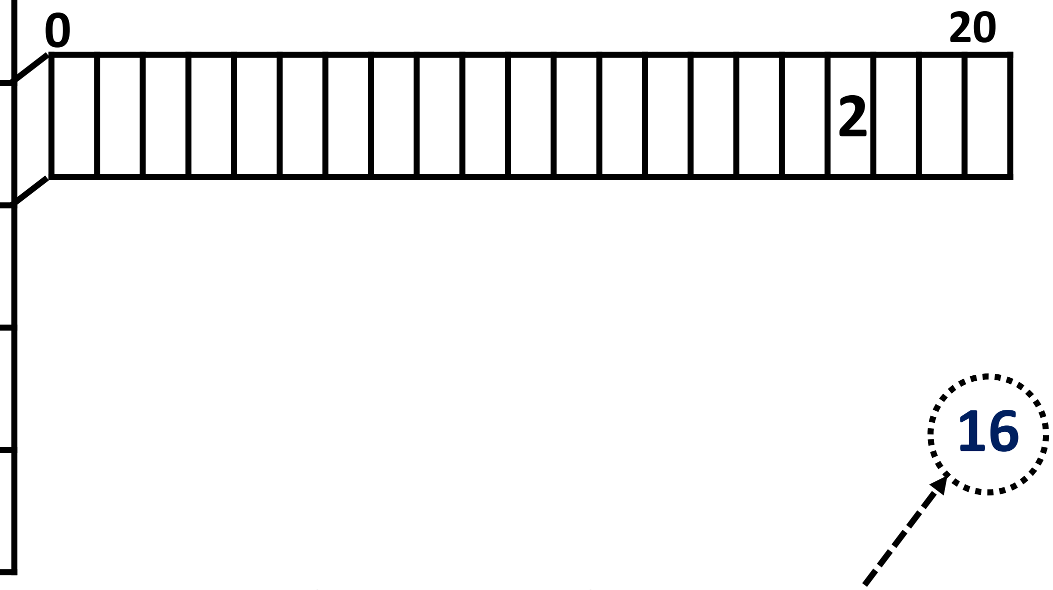
16 now also needs to be remapped to a secondary bucket.

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

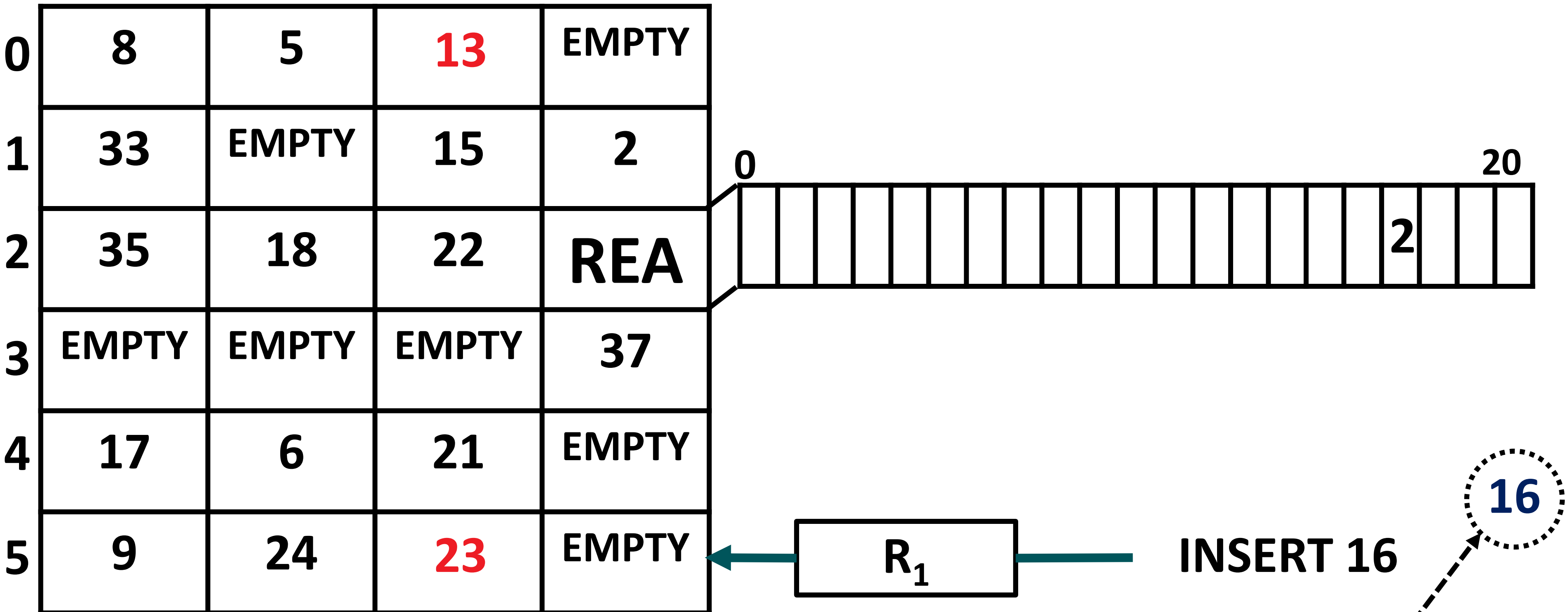


16 now also needs to be remapped to a secondary bucket.

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

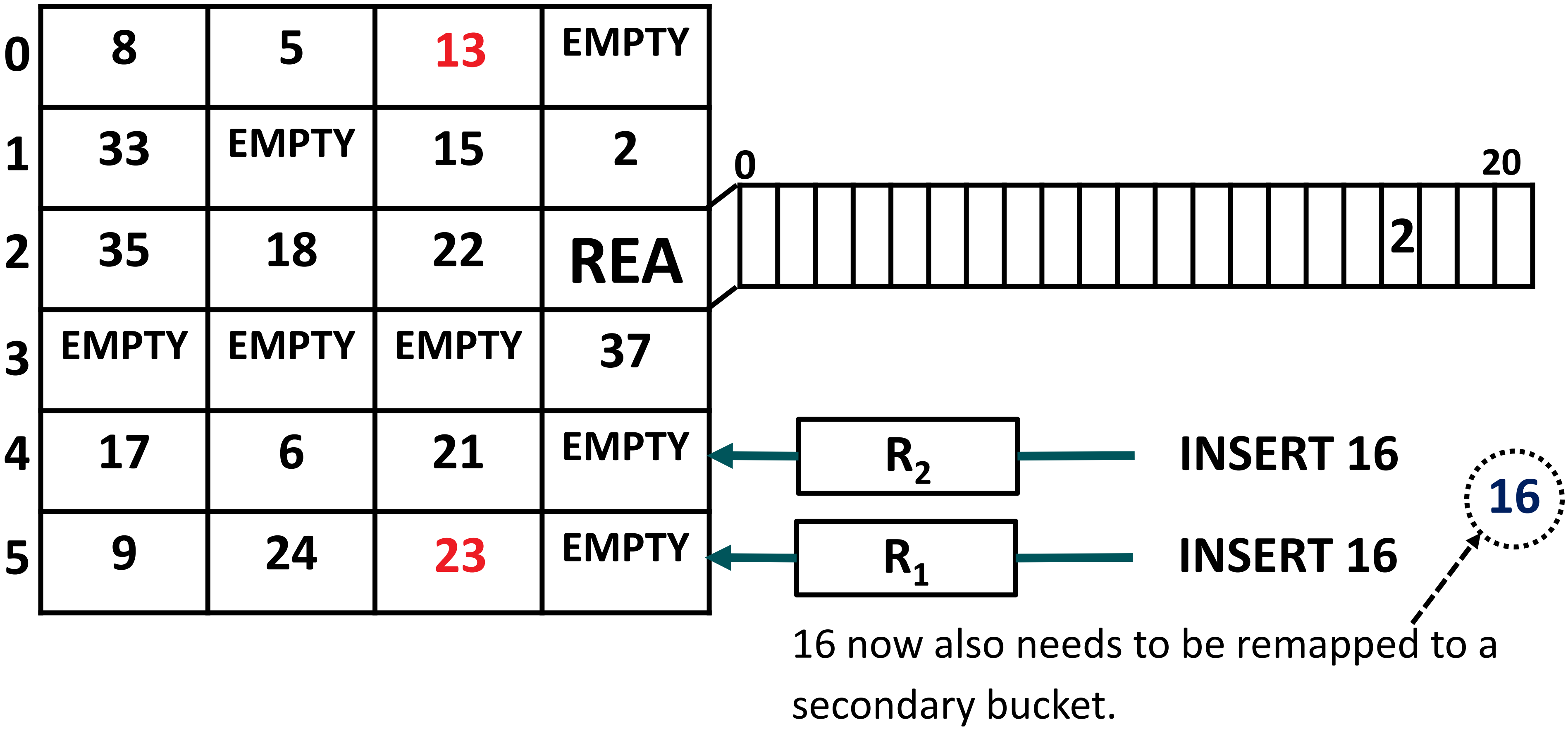


16 now also needs to be remapped to a secondary bucket.

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

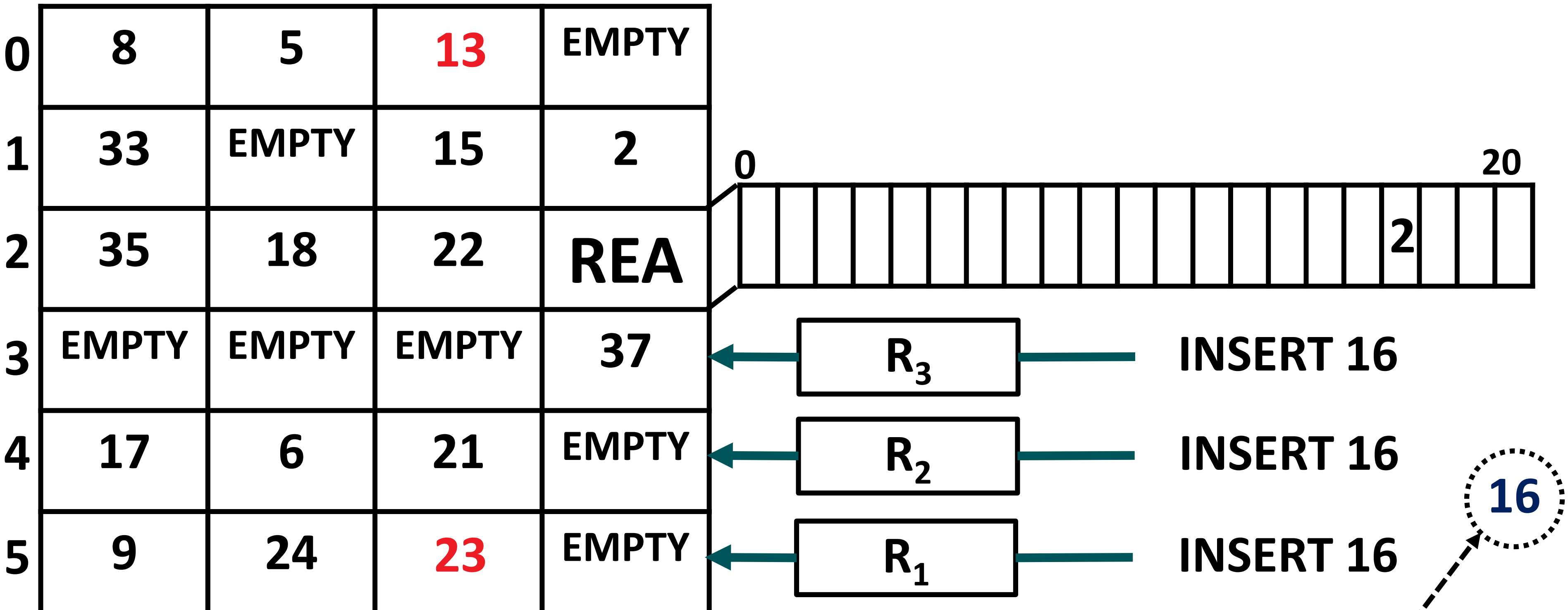
INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

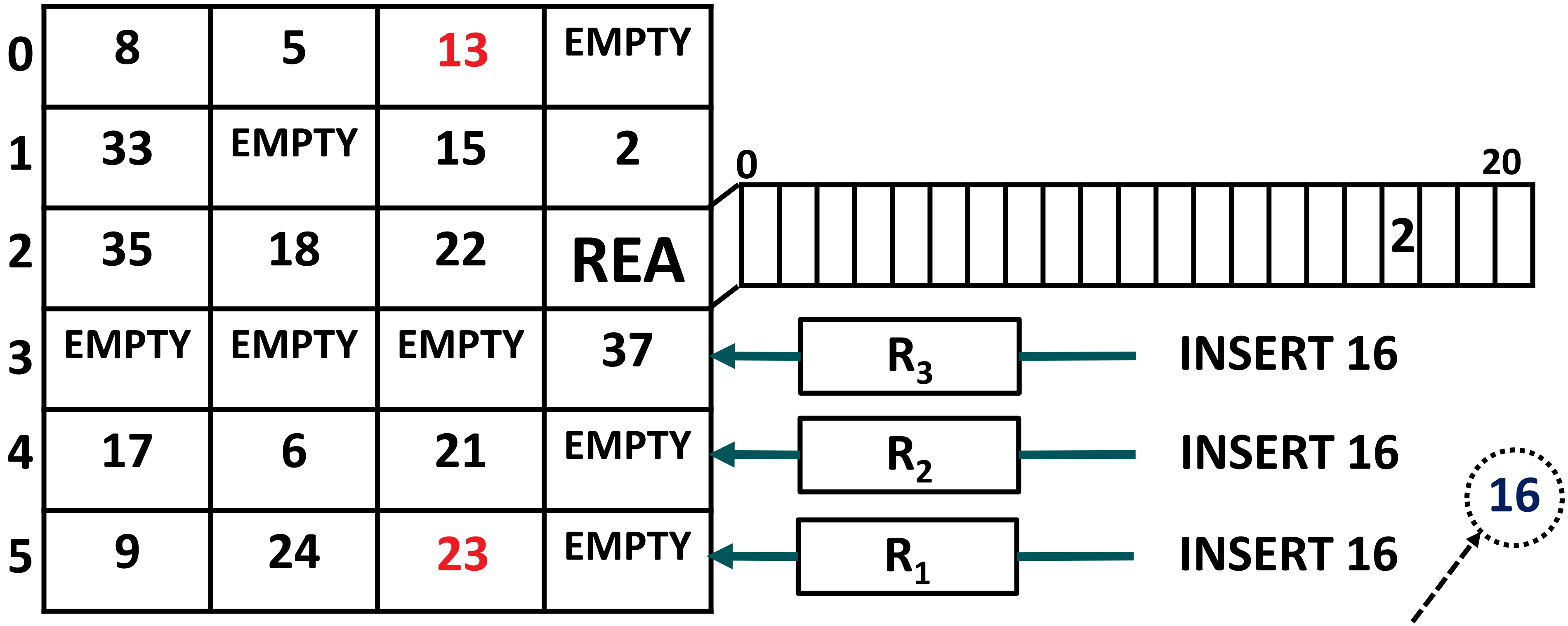


16 now also needs to be remapped to a secondary bucket.

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



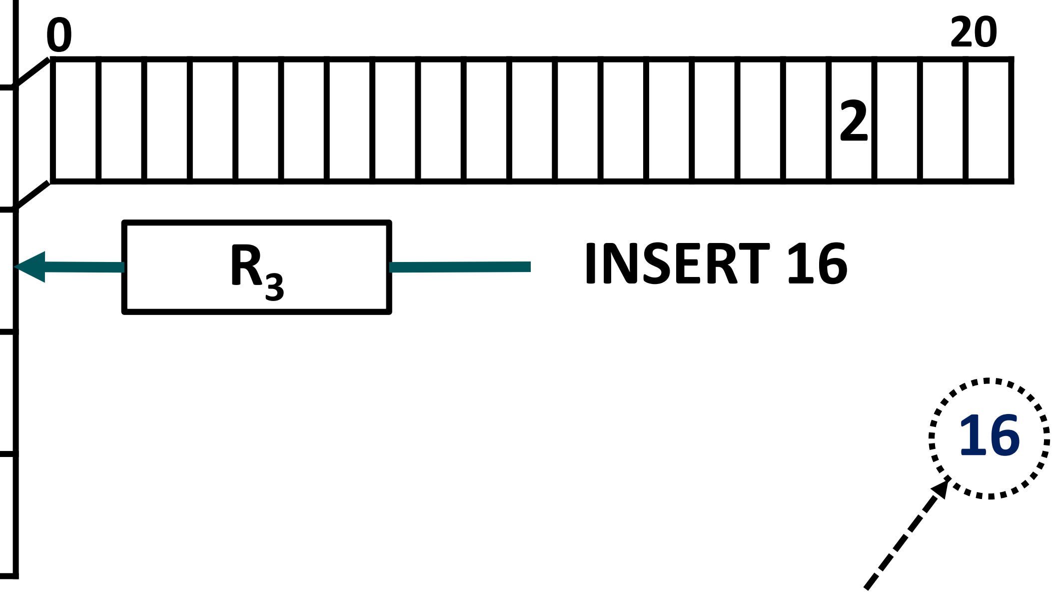
Use R_3 for inserting 16 because it maps 16 to least full secondary bucket candidate.

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



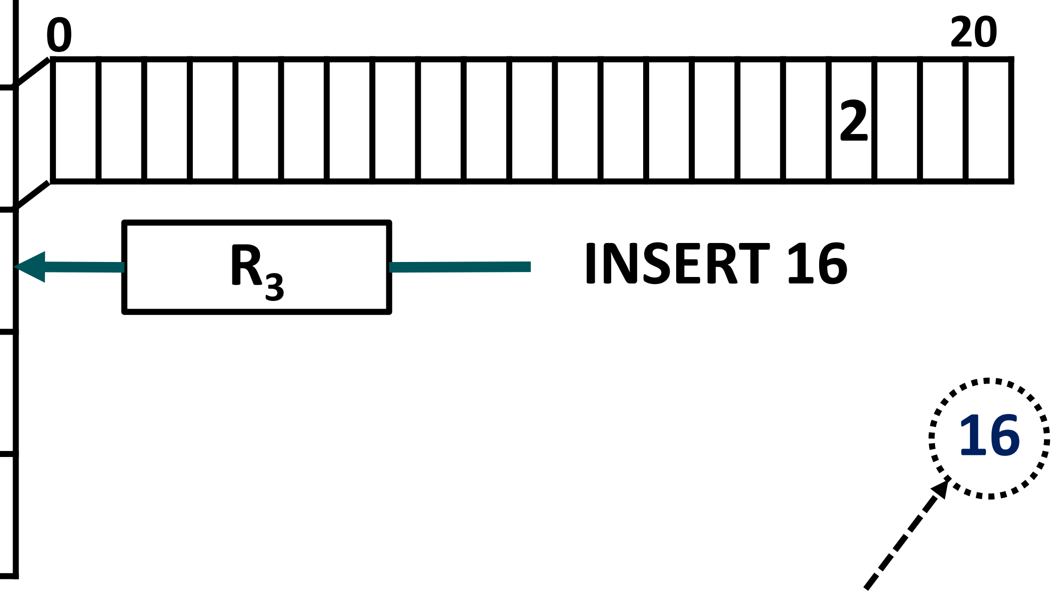
Use R_3 for inserting 16 because it maps 16 to least full secondary bucket candidate.

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



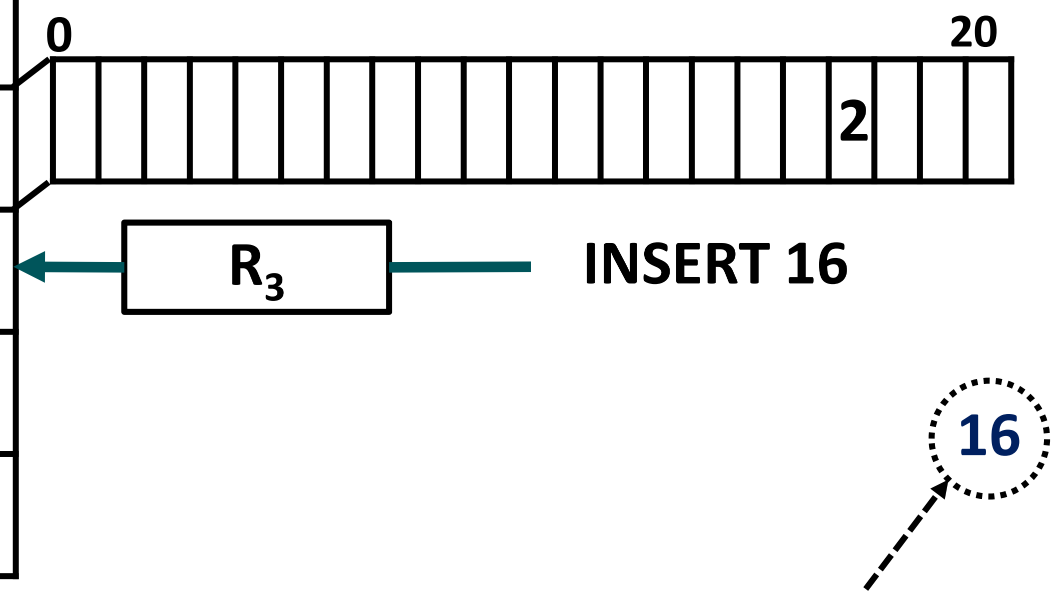
Use R_3 for inserting 16 because it maps 16 to least full secondary bucket candidate.

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



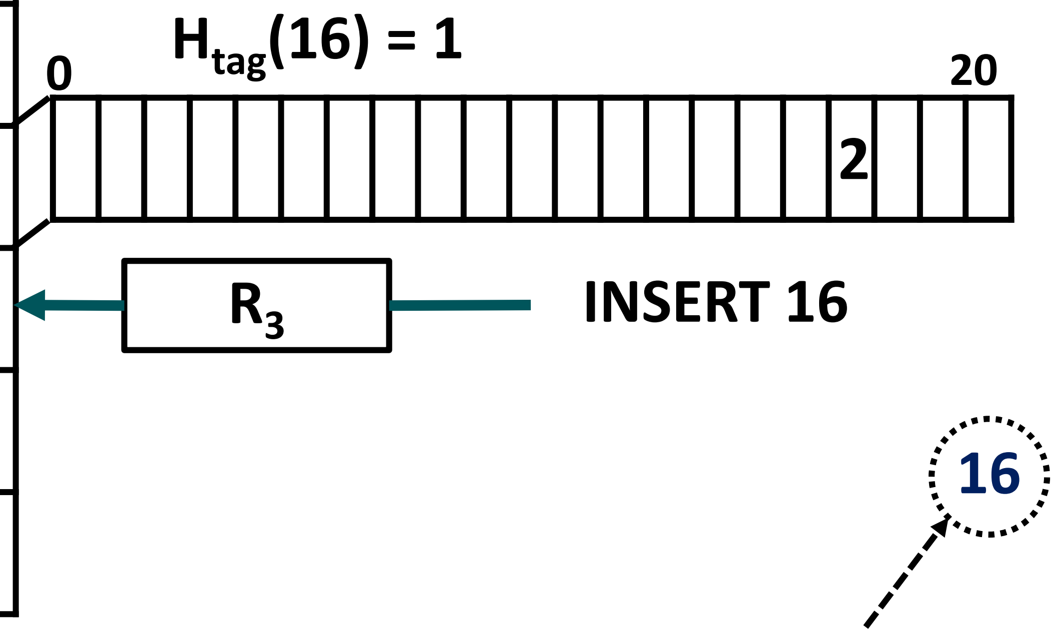
Compute index into remap entry array using H_{tag} with key as input

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



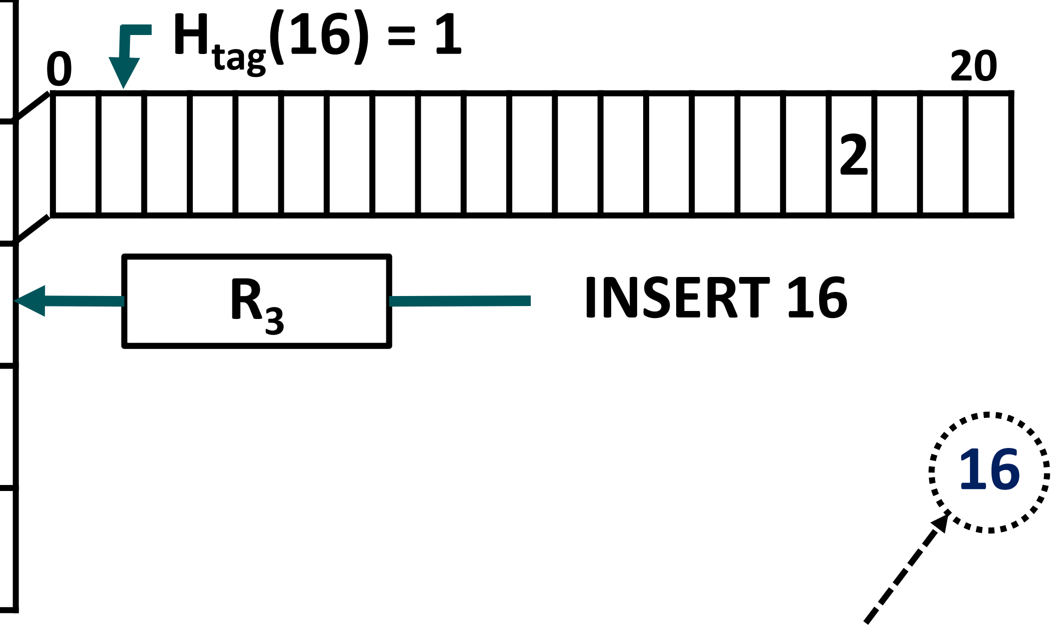
Compute index into remap entry array using H_{tag} with key as input

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



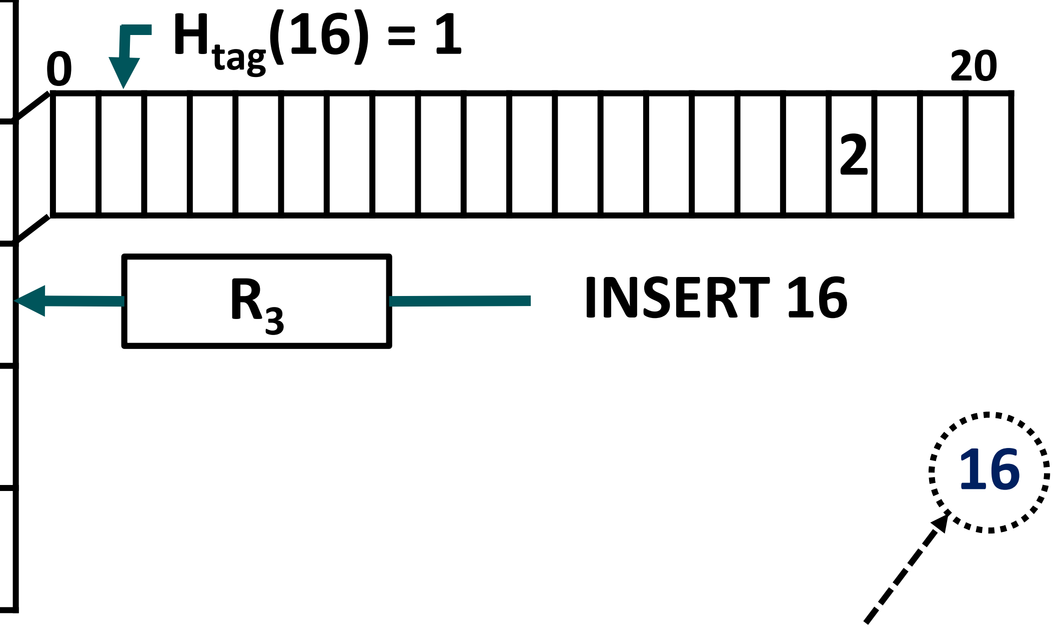
Compute index into remap entry array using H_{tag} with key as input

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



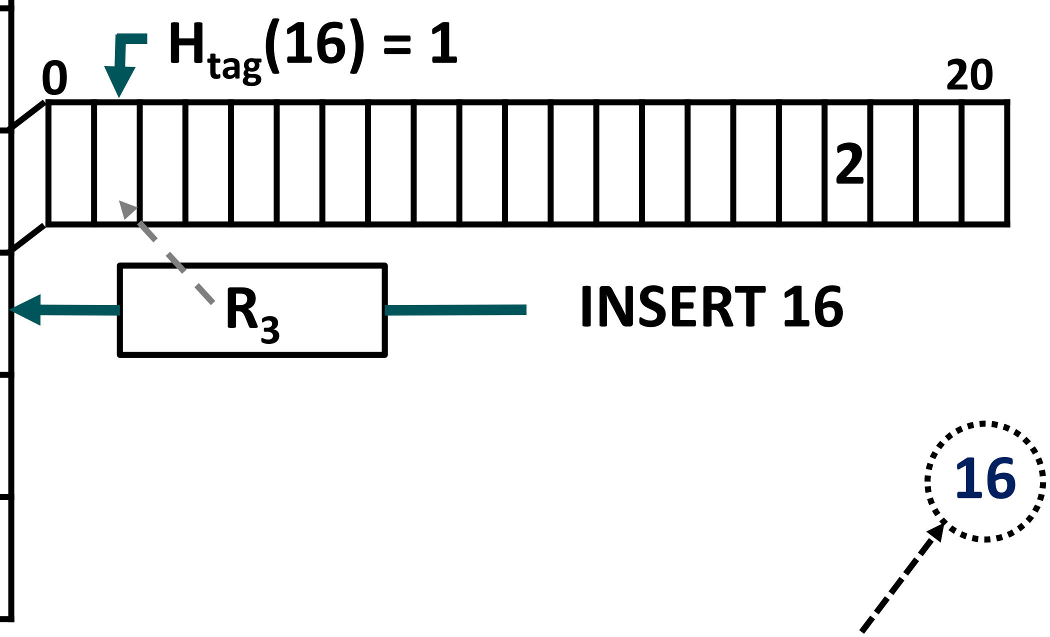
Store 3 at index $H_{tag}(16)=1$ to indicate that R_3 was used to remap 16

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

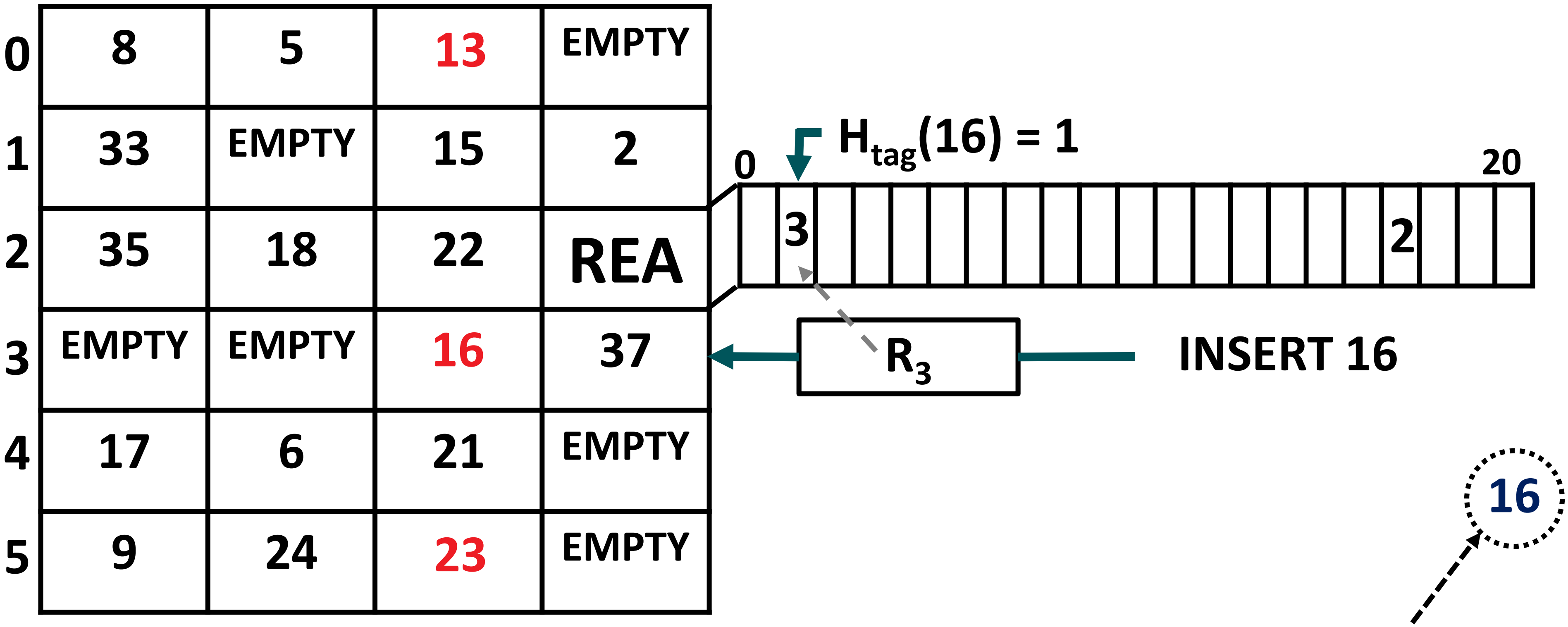
0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



Store 3 at index $H_{tag}(16)=1$ to indicate that R_3 was used to remap 16

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



Store 3 at index $H_{tag}(16)=1$ to indicate that R_3 was used to remap 16

- For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_1, R_2, and R_3) as an element in a *remap entry array* (REA), a sparse, in-bucket array that tracks remapped elements.

HORTON TABLES

RETRIEVING REMAPPED ITEMS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

- Remapped items can always be retrieved by accessing 2 buckets, even when many secondary hash functions are used
- E.g., when retrieving 16, we only access buckets 2 (primary bucket) and 3 (secondary bucket). We skip buckets 4 and 5 even though they were previously candidates.

HORTON TABLES

RETRIEVING REMAPPED ITEMS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

Compute primary hash function and examine primary bucket (bucket 2)

- Remapped items can always be retrieved by accessing 2 buckets, even when many secondary hash functions are used
- E.g., when retrieving 16, we only access buckets 2 (primary bucket) and 3 (secondary bucket). We skip buckets 4 and 5 even though they were previously candidates.

HORTON TABLES

RETRIEVING REMAPPED ITEMS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

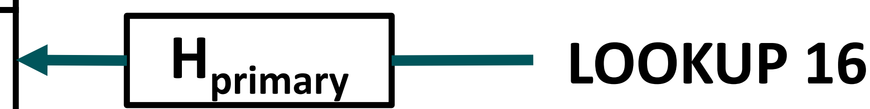
← $H_{primary}$ ← LOOKUP 16

Compute primary hash function and examine primary bucket (bucket 2)

- Remapped items can always be retrieved by accessing 2 buckets, even when many secondary hash functions are used
- E.g., when retrieving 16, we only access buckets 2 (primary bucket) and 3 (secondary bucket). We skip buckets 4 and 5 even though they were previously candidates.

HORTON TABLES
RETRIEVING REMAPPED ITEMS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



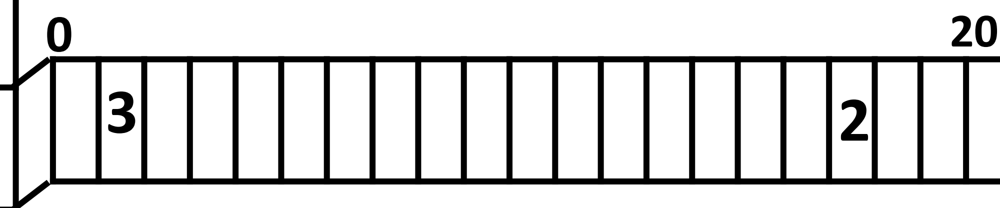
Determine 16 is not stored in its primary bucket and proceed to examine REA

- Remapped items can always be retrieved by accessing 2 buckets, even when many secondary hash functions are used
- E.g., when retrieving 16, we only access buckets 2 (primary bucket) and 3 (secondary bucket). We skip buckets 4 and 5 even though they were previously candidates.

HORTON TABLES

RETRIEVING REMAPPED ITEMS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



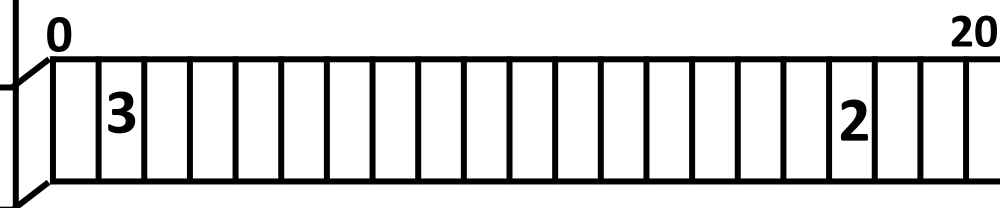
Determine 16 is not stored in its primary bucket and proceed to examine REA

- Remapped items can always be retrieved by accessing 2 buckets, even when many secondary hash functions are used
- E.g., when retrieving 16, we only access buckets 2 (primary bucket) and 3 (secondary bucket). We skip buckets 4 and 5 even though they were previously candidates.

HORTON TABLES

RETRIEVING REMAPPED ITEMS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



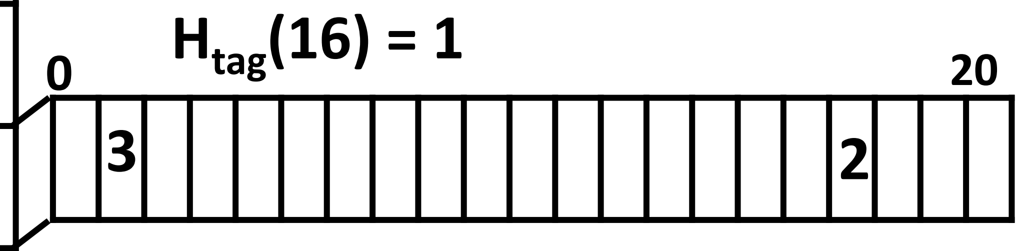
Compute index into remap entry array using H_{tag} with key as input

- Remapped items can always be retrieved by accessing 2 buckets, even when many secondary hash functions are used
- E.g., when retrieving 16, we only access buckets 2 (primary bucket) and 3 (secondary bucket). We skip buckets 4 and 5 even though they were previously candidates.

HORTON TABLES

RETRIEVING REMAPPED ITEMS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



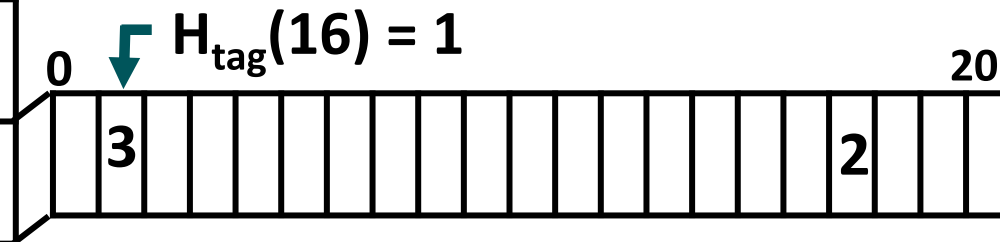
Compute index into remap entry array using H_{tag} with key as input

- Remapped items can always be retrieved by accessing 2 buckets, even when many secondary hash functions are used
- E.g., when retrieving 16, we only access buckets 2 (primary bucket) and 3 (secondary bucket). We skip buckets 4 and 5 even though they were previously candidates.

HORTON TABLES

RETRIEVING REMAPPED ITEMS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



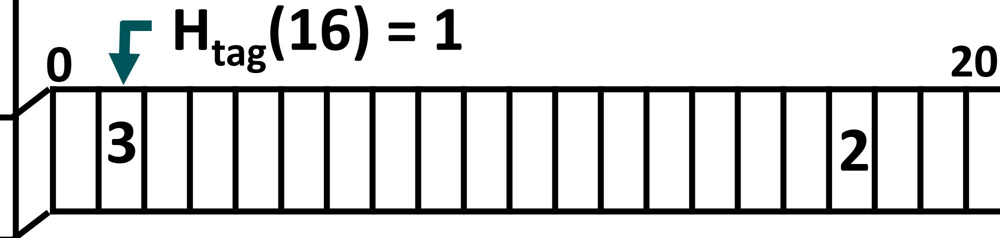
Compute index into remap entry array using H_{tag} with key as input

- Remapped items can always be retrieved by accessing 2 buckets, even when many secondary hash functions are used
- E.g., when retrieving 16, we only access buckets 2 (primary bucket) and 3 (secondary bucket). We skip buckets 4 and 5 even though they were previously candidates.

HORTON TABLES

RETRIEVING REMAPPED ITEMS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

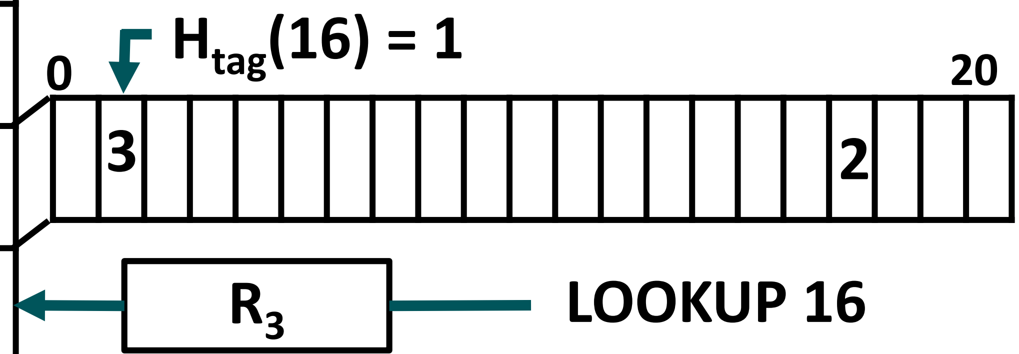


The remap entry shows R_3 was used to remap 16, so compute $R_3(16)$.

- Remapped items can always be retrieved by accessing 2 buckets, even when many secondary hash functions are used
- E.g., when retrieving 16, we only access buckets 2 (primary bucket) and 3 (secondary bucket). We skip buckets 4 and 5 even though they were previously candidates.

HORTON TABLES
RETRIEVING REMAPPED ITEMS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

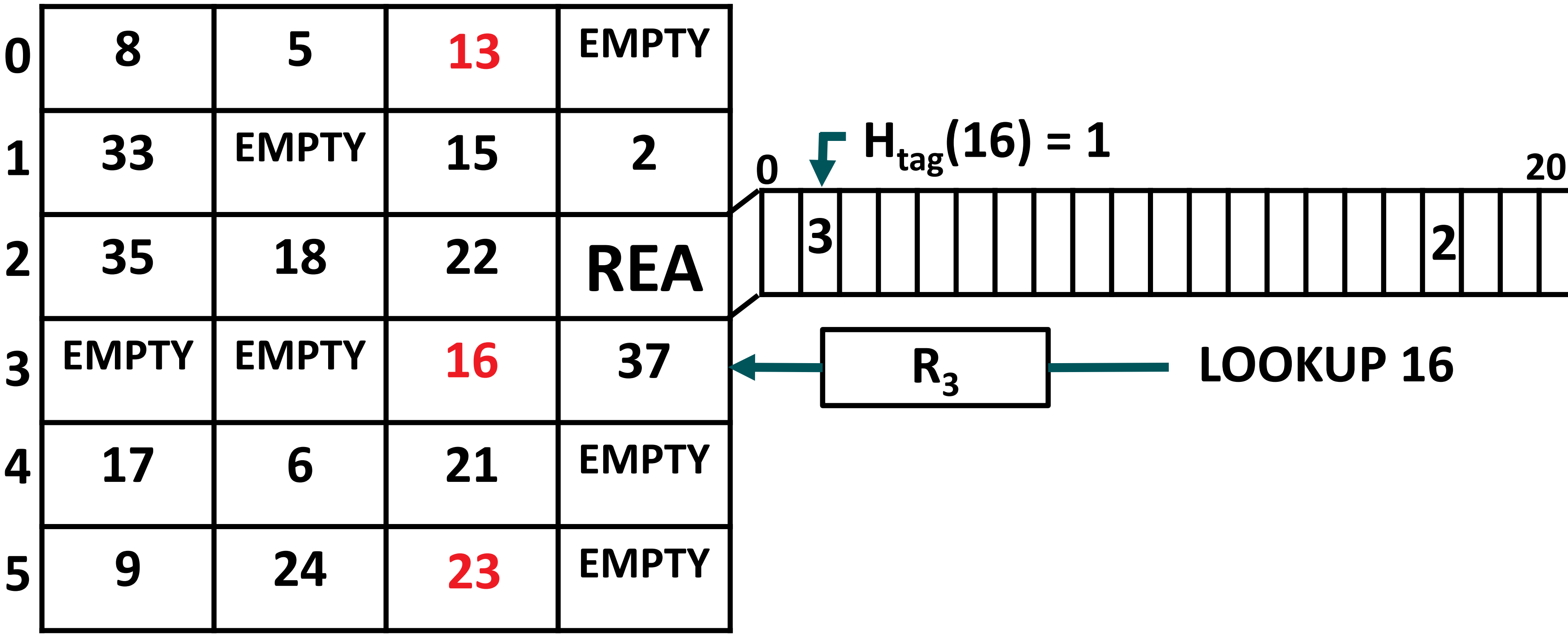


The remap entry shows R_3 was used to remap 16, so compute $R_3(16)$.

- Remapped items can always be retrieved by accessing 2 buckets, even when many secondary hash functions are used
- E.g., when retrieving 16, we only access buckets 2 (primary bucket) and 3 (secondary bucket). We skip buckets 4 and 5 even though they were previously candidates.

HORTON TABLES

RETRIEVING REMAPPED ITEMS

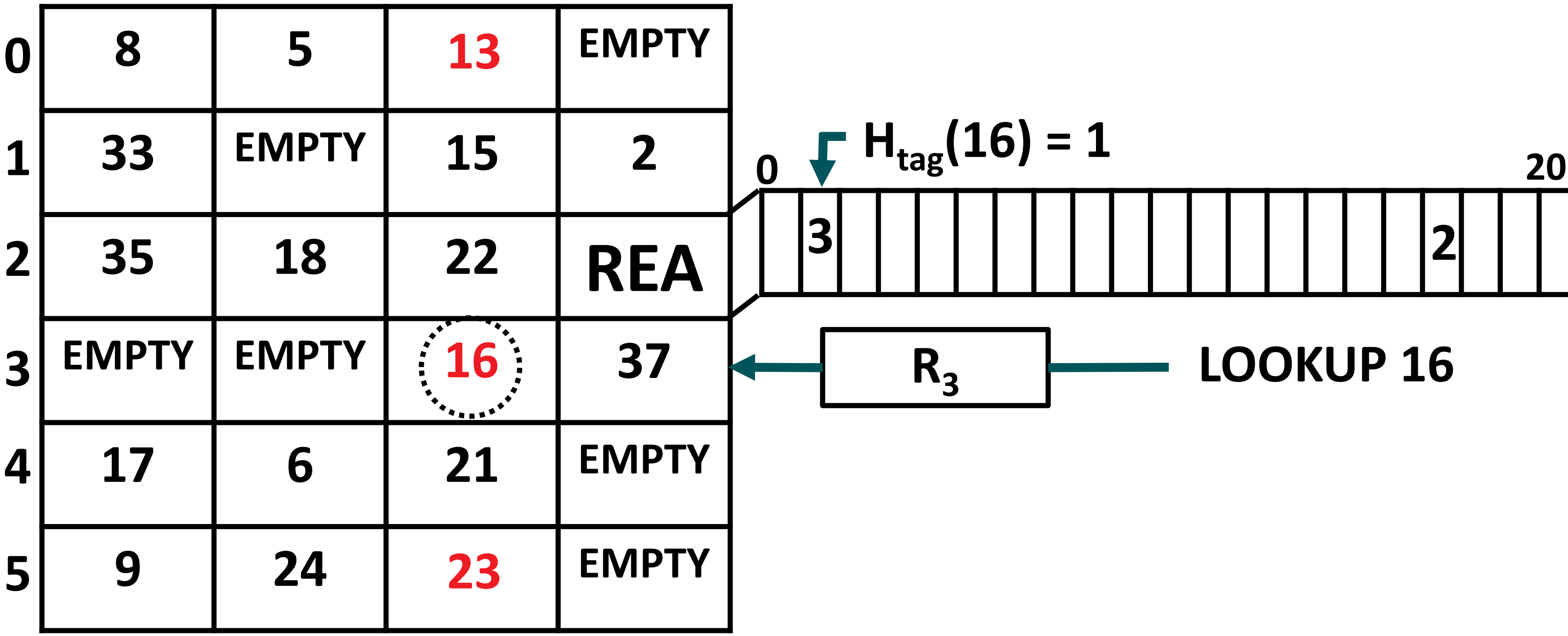


Retrieve 16 from bucket 3

- Remapped items can always be retrieved by accessing 2 buckets, even when many secondary hash functions are used
- E.g., when retrieving 16, we only access buckets 2 (primary bucket) and 3 (secondary bucket). We skip buckets 4 and 5 even though they were previously candidates.

HORTON TABLES

RETRIEVING REMAPPED ITEMS



Retrieve 16 from bucket 3

- Remapped items can always be retrieved by accessing 2 buckets, even when many secondary hash functions are used
- E.g., when retrieving 16, we only access buckets 2 (primary bucket) and 3 (secondary bucket). We skip buckets 4 and 5 even though they were previously candidates.

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

Most negative lookups only require accessing a single bucket

HORTON TABLES
NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

LOOKUP 25

Most negative lookups only require accessing a single bucket

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{primary}$ ← **LOOKUP 25**

Most negative lookups only require accessing a single bucket

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{primary}$

LOOKUP 25

Most negative lookups only require accessing a single bucket

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

Most negative lookups only require accessing a single bucket

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{primary}$ ← **LOOKUP 25**

Most negative lookups only require accessing a single bucket

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

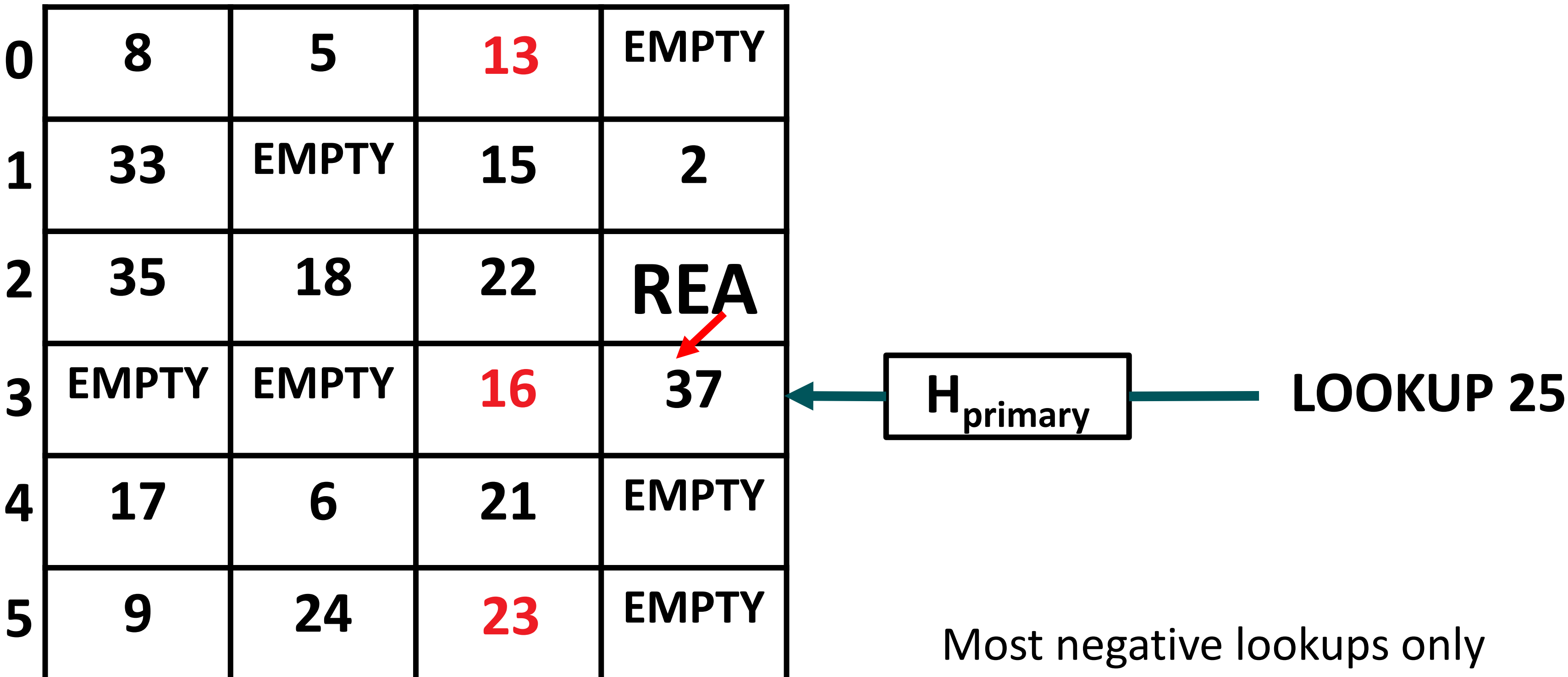
$H_{primary}$

LOOKUP 25

Most negative lookups only require accessing a single bucket

HORTON TABLES

NEGATIVE LOOKUPS



Most negative lookups only require accessing a single bucket

- Lookups where the primary bucket is a conventional BCHT bucket without remap entries only ever require examining 1 bucket and thus 1 cache line for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

Most negative lookups only require accessing a single bucket

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES
NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

LOOKUP 28

Most negative lookups only require accessing a single bucket

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

LOOKUP 28

Most negative lookups only require accessing a single bucket

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{primary}$ ← LOOKUP 28

Most negative lookups only require accessing a single bucket

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

Most negative lookups only require accessing a single bucket

- Lookups where the primary bucket’s final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES
NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{primary}$ ← LOOKUP 28

Most negative lookups only require accessing a single bucket

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{primary}$

LOOKUP 28

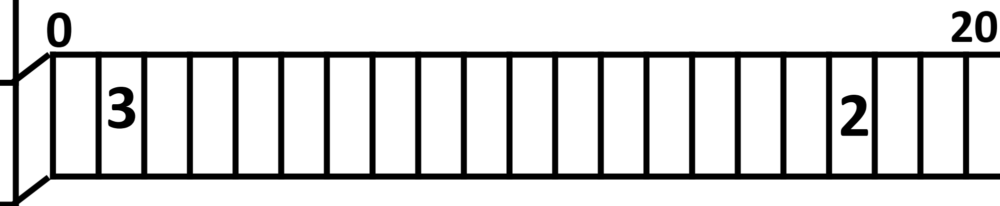
Most negative lookups only require accessing a single bucket

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



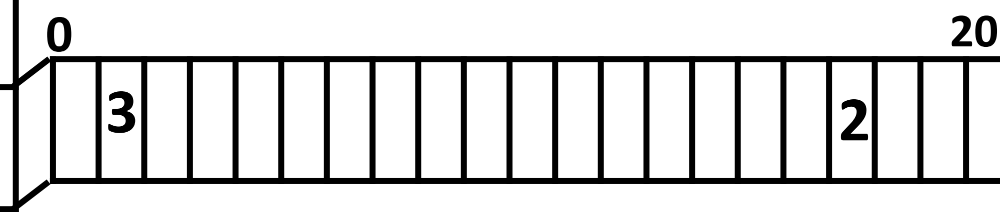
Most negative lookups only require accessing a single bucket

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



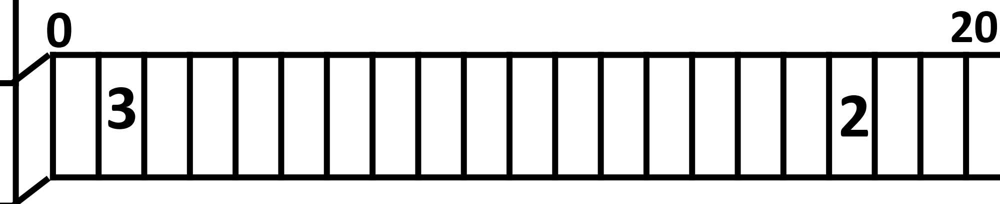
Determine 28 is not stored in its primary bucket (2) and proceed to examine REA

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



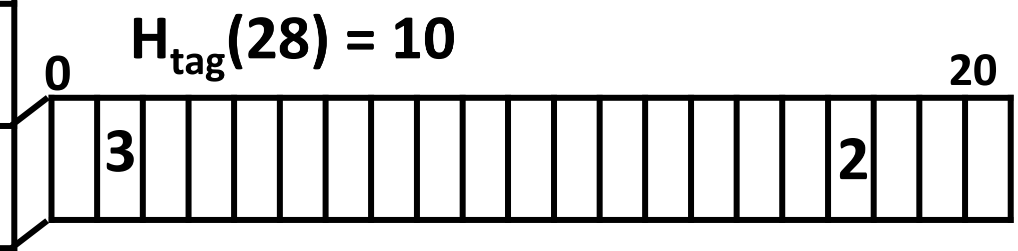
Compute index into remap entry array using H_{tag} with key as input

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



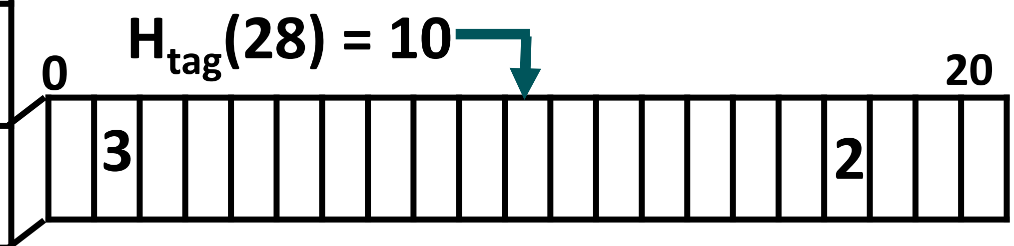
Compute index into remap entry array using H_{tag} with key as input

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



Examine 10th slot of remap entry array and see it is empty. The search can stop.

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS WITH TAG ALIAS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

Negative lookups only require accessing 2 buckets on a *tag alias*

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS WITH TAG ALIAS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

LOOKUP 7

Negative lookups only require accessing 2 buckets on a *tag alias*

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS WITH TAG ALIAS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

LOOKUP 7 → $H_{primary}$ → bucket 2

Negative lookups only require accessing 2 buckets on a *tag alias*

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS WITH TAG ALIAS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{primary}$

LOOKUP 7

Negative lookups only require accessing 2 buckets on a *tag alias*

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS WITH TAG ALIAS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

Negative lookups only require accessing 2 buckets on a *tag alias*

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS WITH TAG ALIAS

0	**8**	**5**	**13**	**EMPTY**
1	**33**	**EMPTY**	**15**	**2**
2	**35**	**18**	**22**	**REA**
3	**EMPTY**	**EMPTY**	**16**	**37**
4	**17**	**6**	**21**	**EMPTY**
5	**9**	**24**	**23**	**EMPTY**

$H_{primary}$ ← **LOOKUP 7**

Negative lookups only require accessing 2 buckets on a *tag alias*

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS WITH TAG ALIAS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{primary}$ ← **LOOKUP 7**

Negative lookups only require accessing 2 buckets on a *tag alias*

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS WITH TAG ALIAS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

LOOKUP 7 → $H_{primary}$ → bucket 2

Determine 7 is not stored in its primary bucket (2) and proceed to examine REA

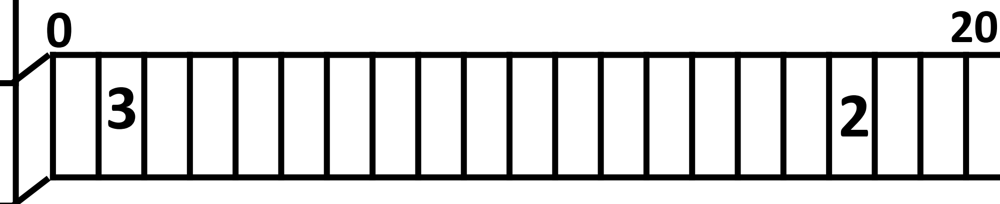
Negative lookups only require accessing 2 buckets on a *tag alias*

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS WITH TAG ALIAS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



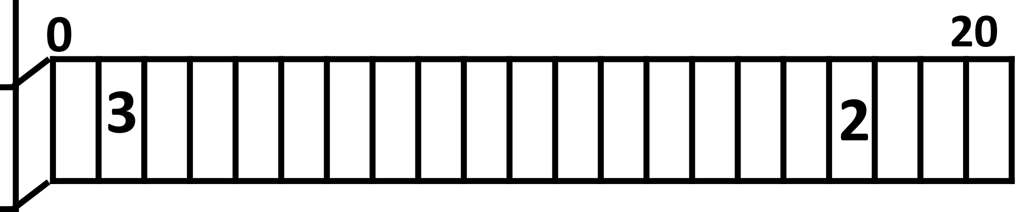
Negative lookup with a *tag alias*, e.g. 7 reads remap entry set by 23

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS WITH TAG ALIAS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



Compute index into remap entry array using H_{tag} with key as input

Negative lookup with a *tag alias*, e.g. 7 reads remap entry set by 23

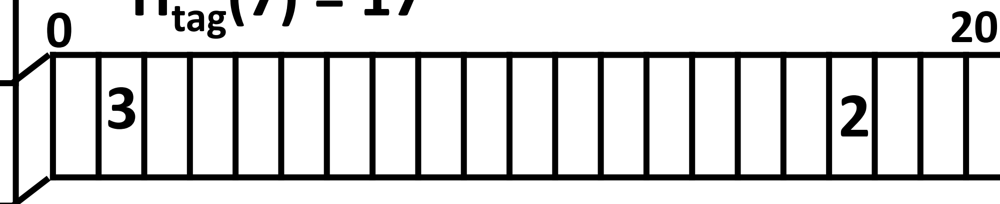
- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS WITH TAG ALIAS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{tag}(7) = 17$



Compute index into remap entry array using H_{tag} with key as input

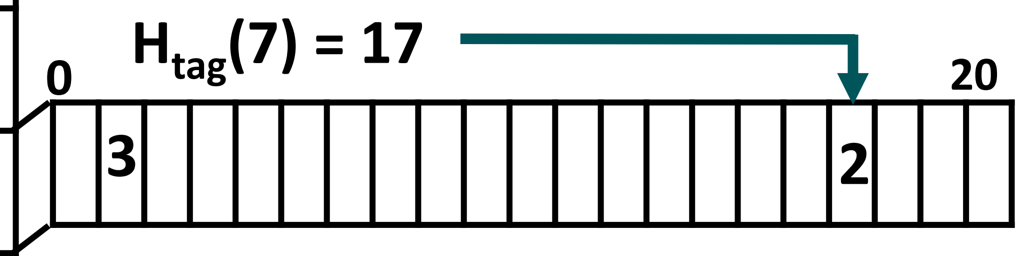
Negative lookup with a *tag alias*, e.g. 7 reads remap entry set by 23

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS WITH TAG ALIAS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



Examine 18th slot of remap entry array and see that R_2 was likely used to remap 7.

Negative lookup with a *tag alias*, e.g. 7 reads remap entry set by 23

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS WITH TAG ALIAS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

LOOKUP 7

Examine 18th slot of remap entry array and see that R_2 was likely used to remap 7.

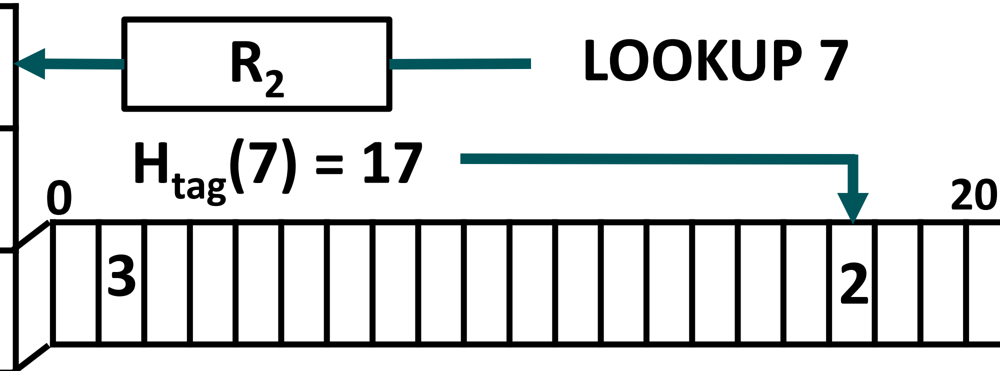
Negative lookup with a *tag alias*, e.g. 7 reads remap entry set by 23

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS WITH TAG ALIAS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



Examine 18th slot of remap entry array and see that R_2 was likely used to remap 7.

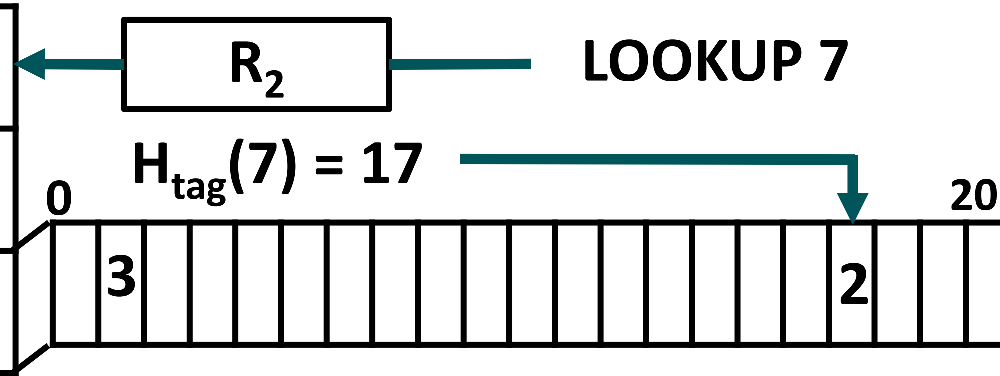
Negative lookup with a *tag alias*, e.g. 7 reads remap entry set by 23

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

HORTON TABLES

NEGATIVE LOOKUPS WITH TAG ALIAS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



Determine that no slots of secondary bucket (0) match 7, so stop looking.

Negative lookup with a *tag alias*, e.g. 7 reads remap entry set by 23

- Lookups where the primary bucket's final slot is converted into an REA often only require accessing 1 bucket and at most 2 for positive and negative queries alike

ADDITIONAL CONTENT IN THE PAPER

- Sharing of remap entries among multiple remapped elements while still permitting their deletion

- Further optimizations for improving lookup throughput

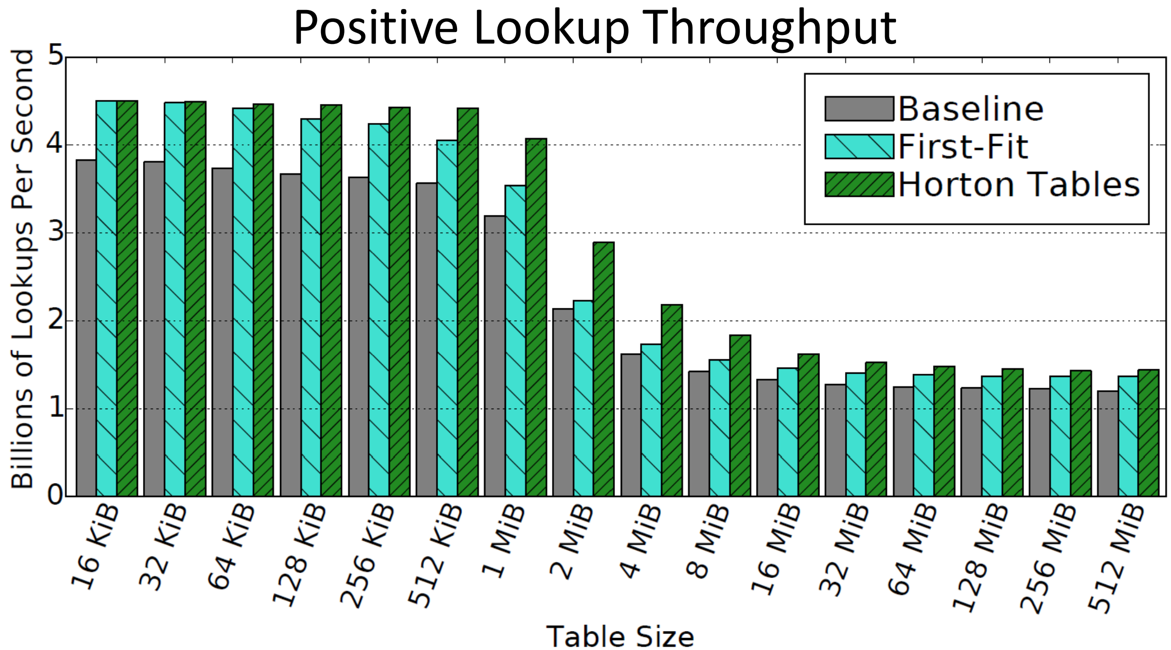
- Analytical models for lookups, insertions and deletions

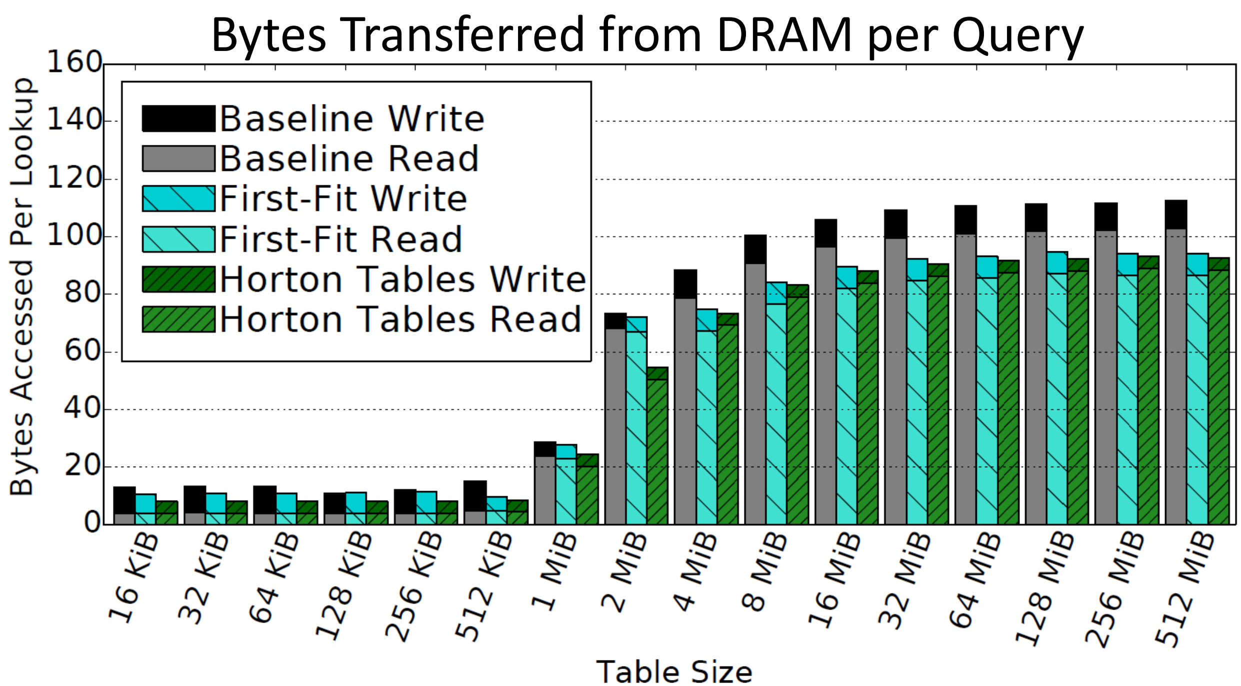
- More in-depth discussion of prior work and how Horton tables improves over first-fit for positive lookups

EXPERIMENTAL METHODOLOGY

- Conducted a series of analytical studies to determine 8-slots per bucket was a good design point (**more details in paper**)
  - Fills a 64-byte cache line with 8-byte entries
  - High load factors (>95% table can be filled with key-value pairs)
  - Positive lookups that typically access less than 1.18 buckets per query
  - Negative lookups that typically access less than 1.06 buckets per query
- Further analytical studies demonstrated that 21 entries per REA and 7 secondary functions is often more than sufficient for 8-slot buckets (**more details in paper**)
- Experimental studies conducted on an AMD Radeon™ R9 290X GPU

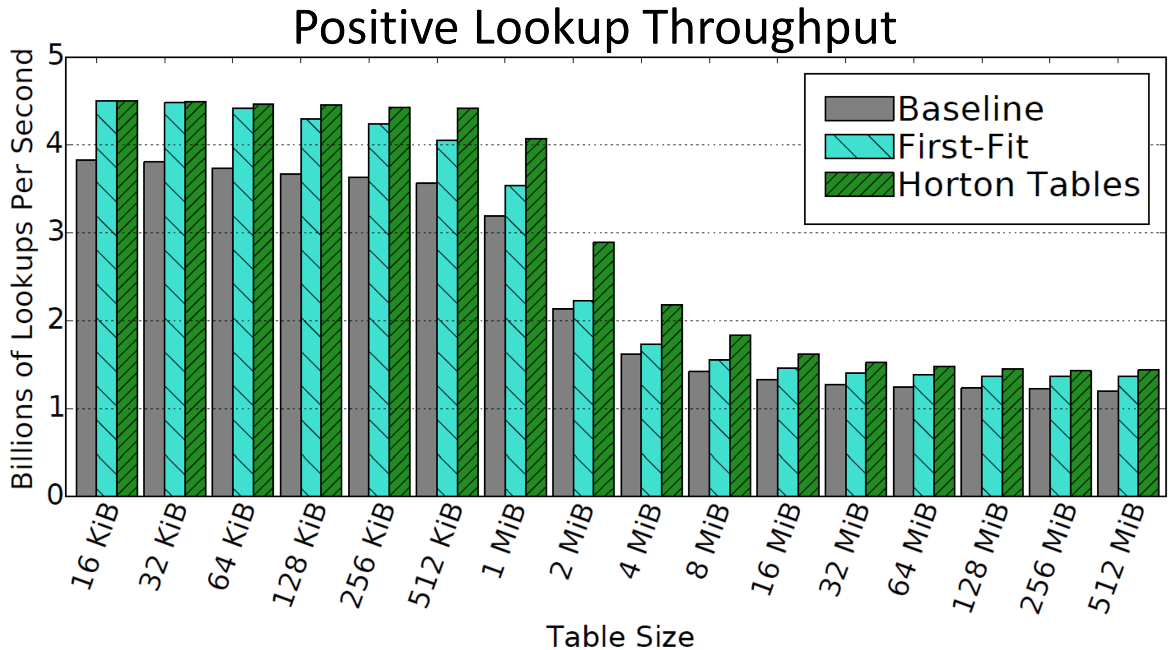
RESULTS
POSITIVE LOOKUPS

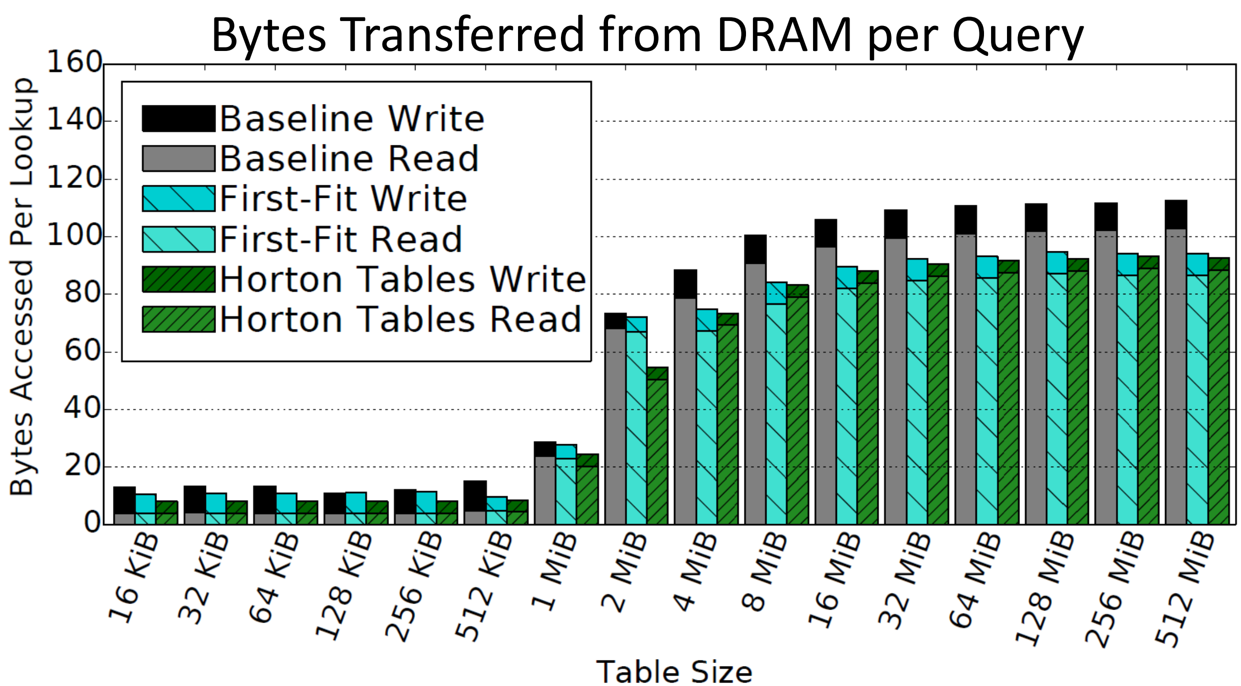




RESULTS
POSITIVE LOOKUPS

HIGHER IS BETTER

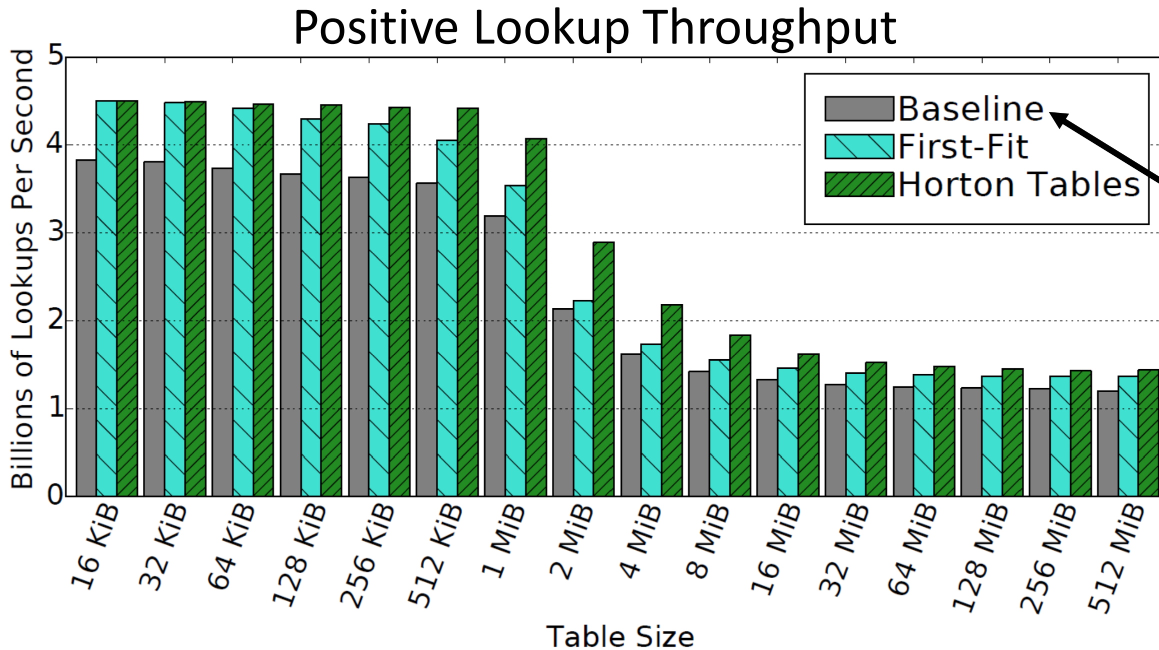




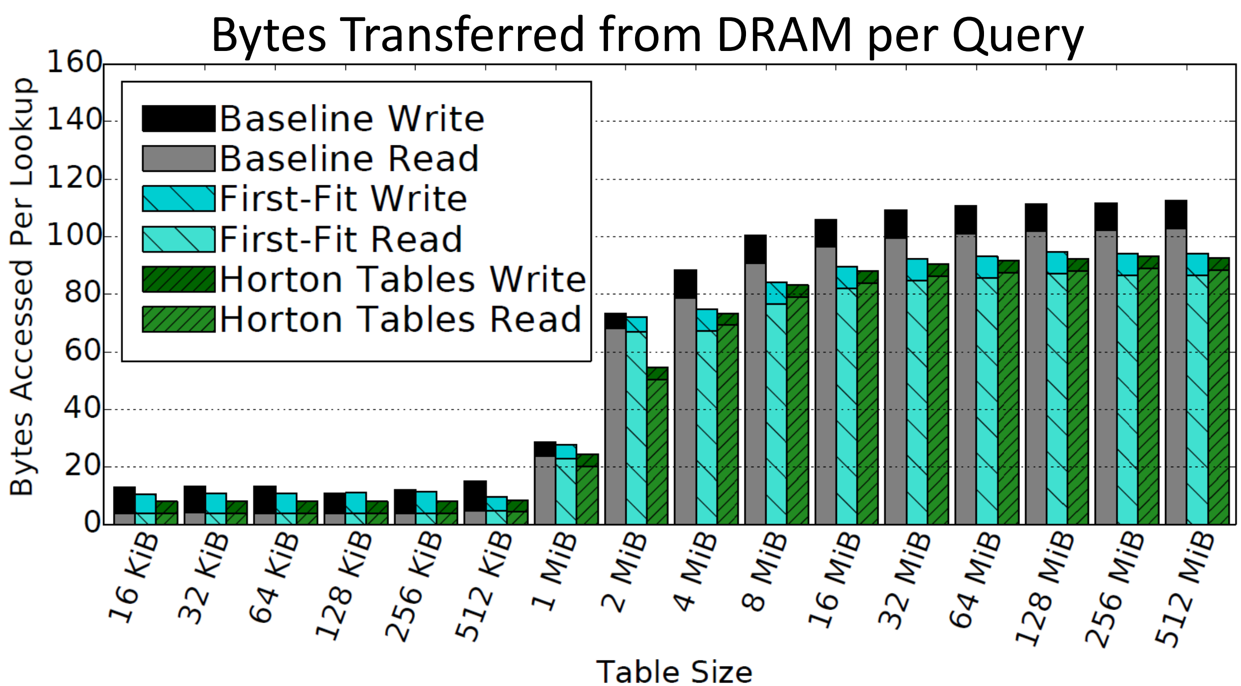
RESULTS

POSITIVE LOOKUPS

HIGHER IS BETTER



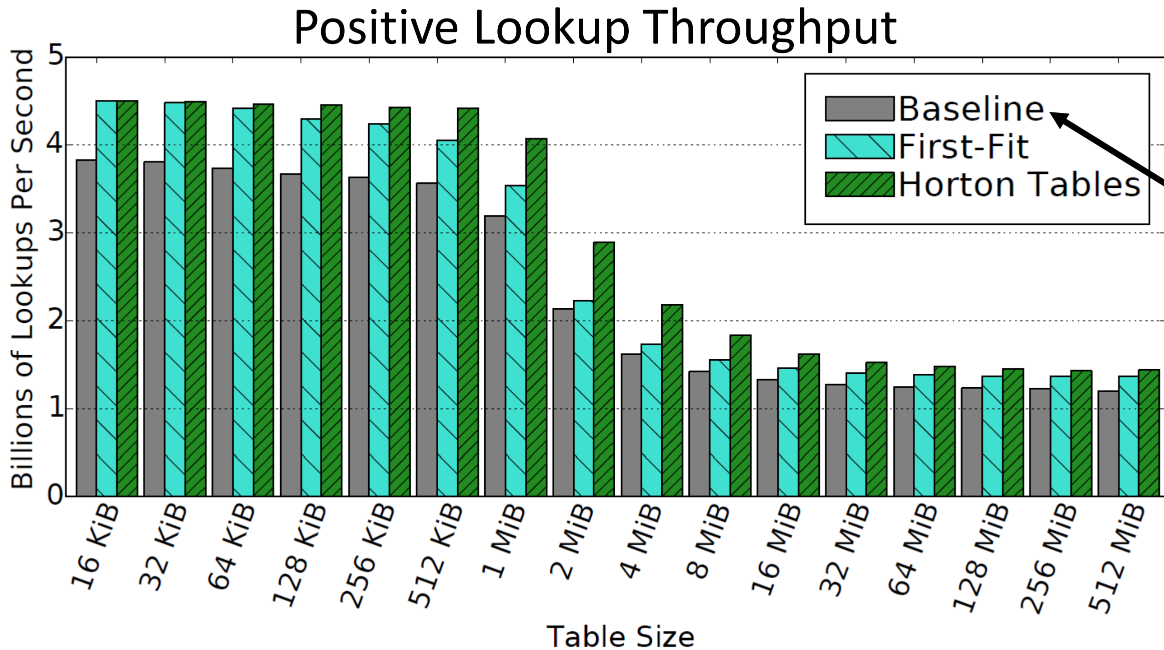
Load balancing insertion heuristic



RESULTS

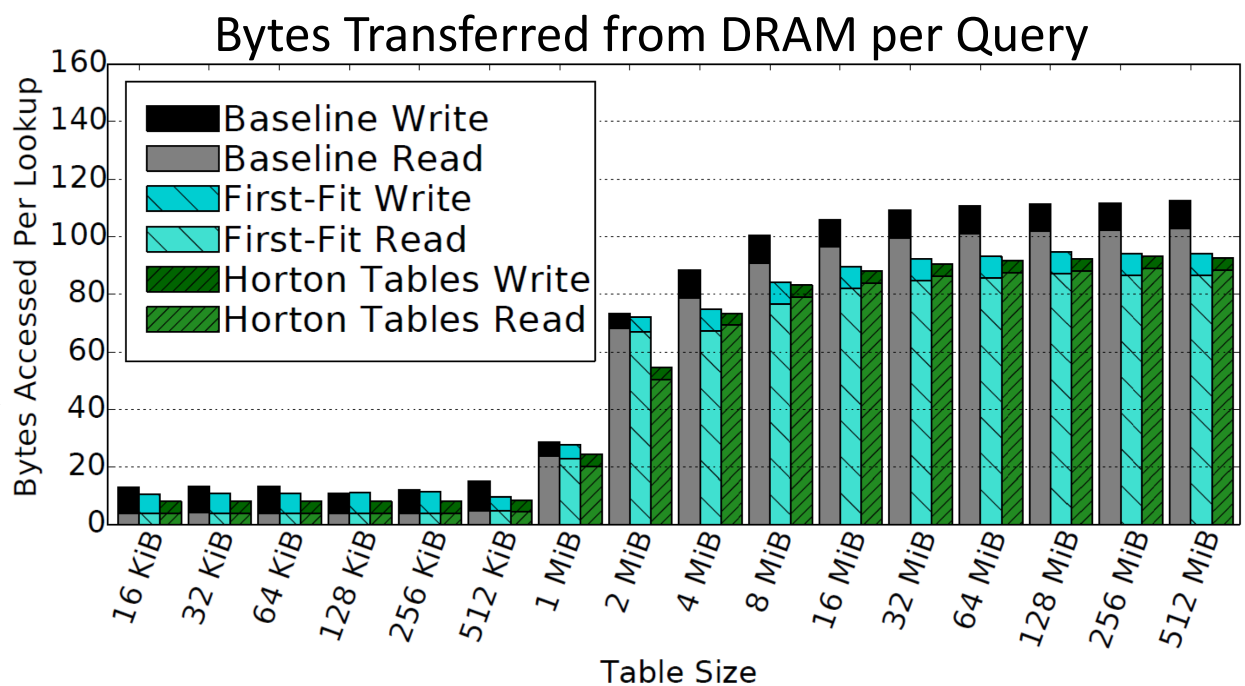
POSITIVE LOOKUPS

HIGHER IS BETTER



Load balancing insertion heuristic

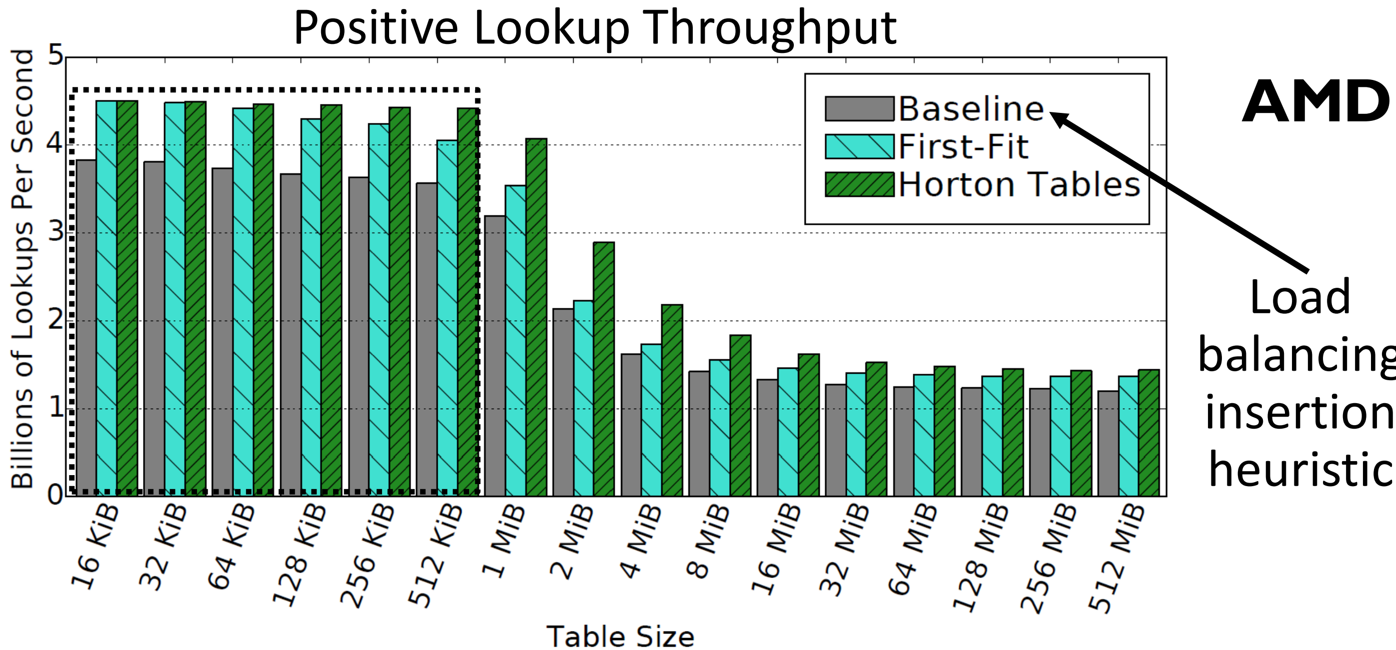
LOWER IS BETTER



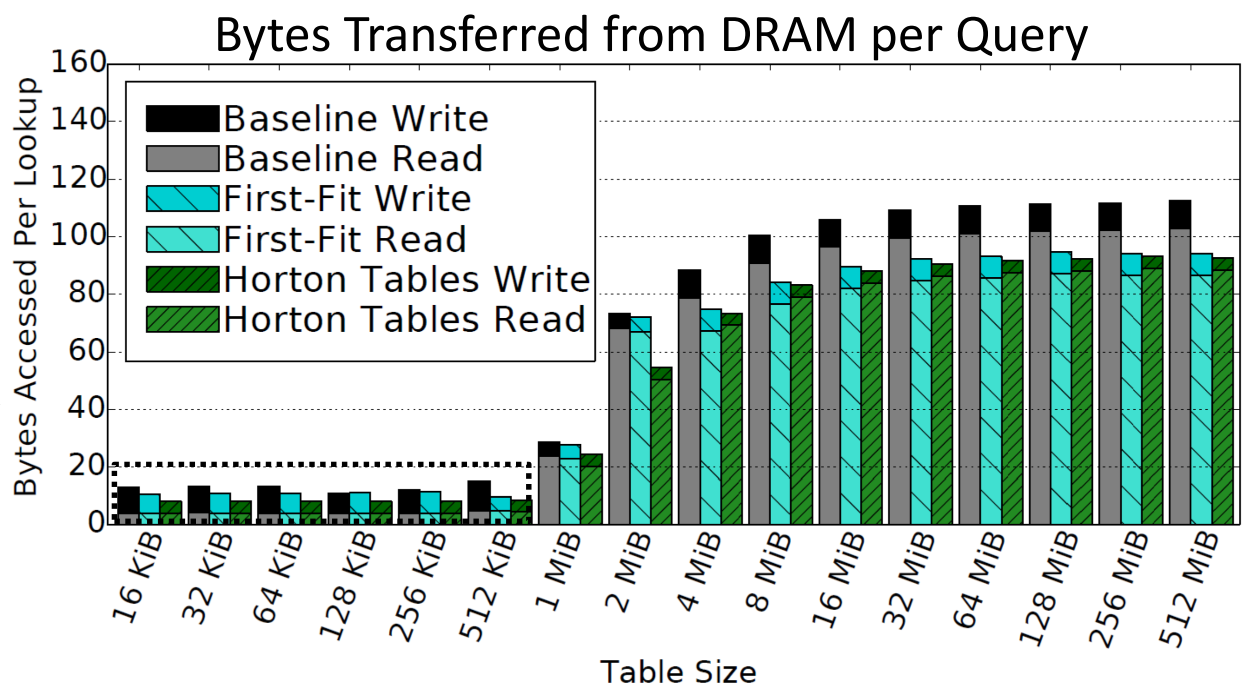
RESULTS

POSITIVE LOOKUPS

HIGHER IS BETTER

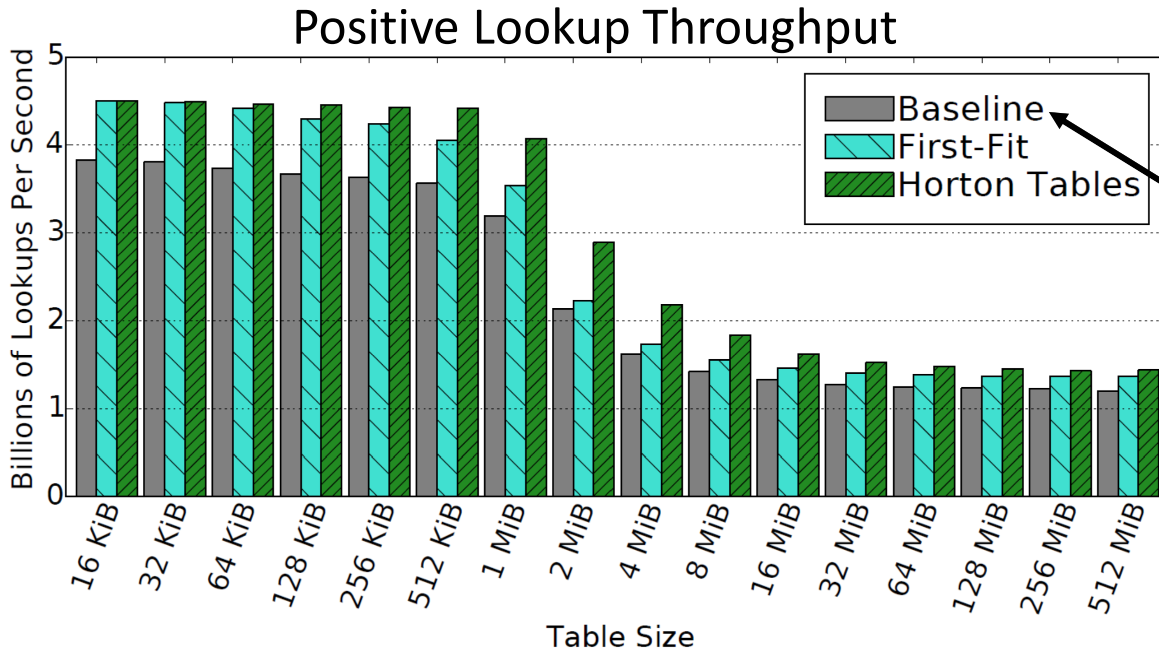


LOWER IS BETTER



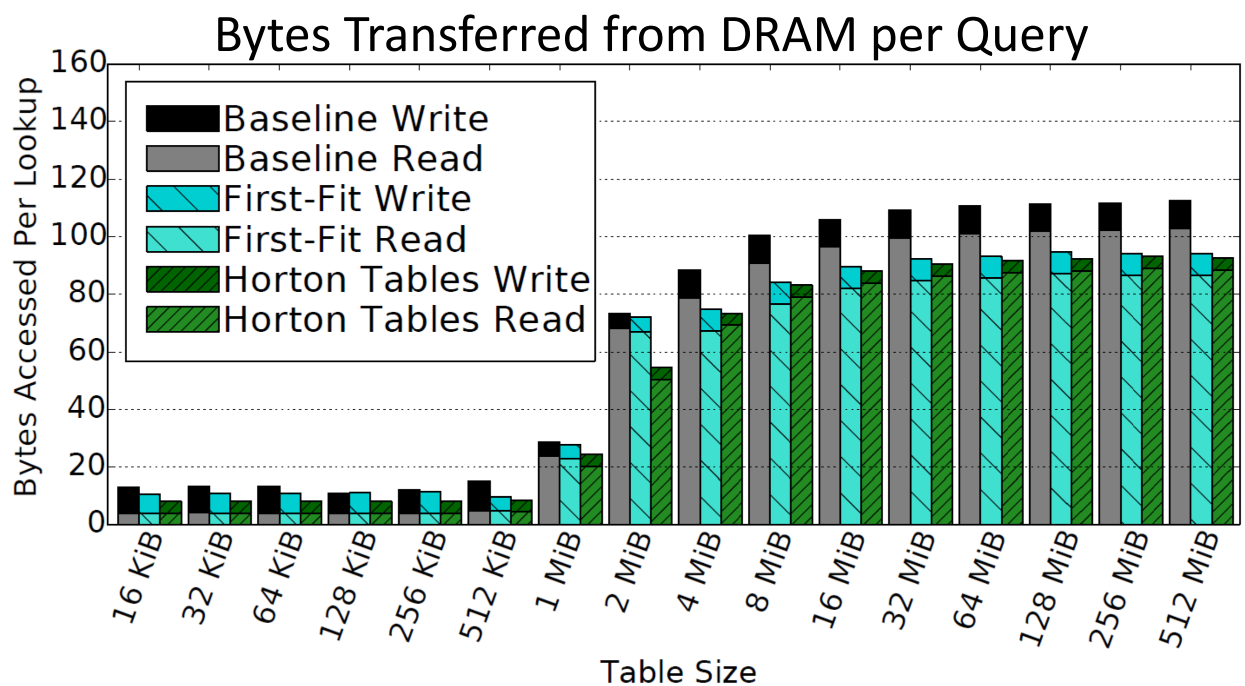
RESULTS
POSITIVE LOOKUPS

HIGHER IS BETTER



Load balancing insertion heuristic

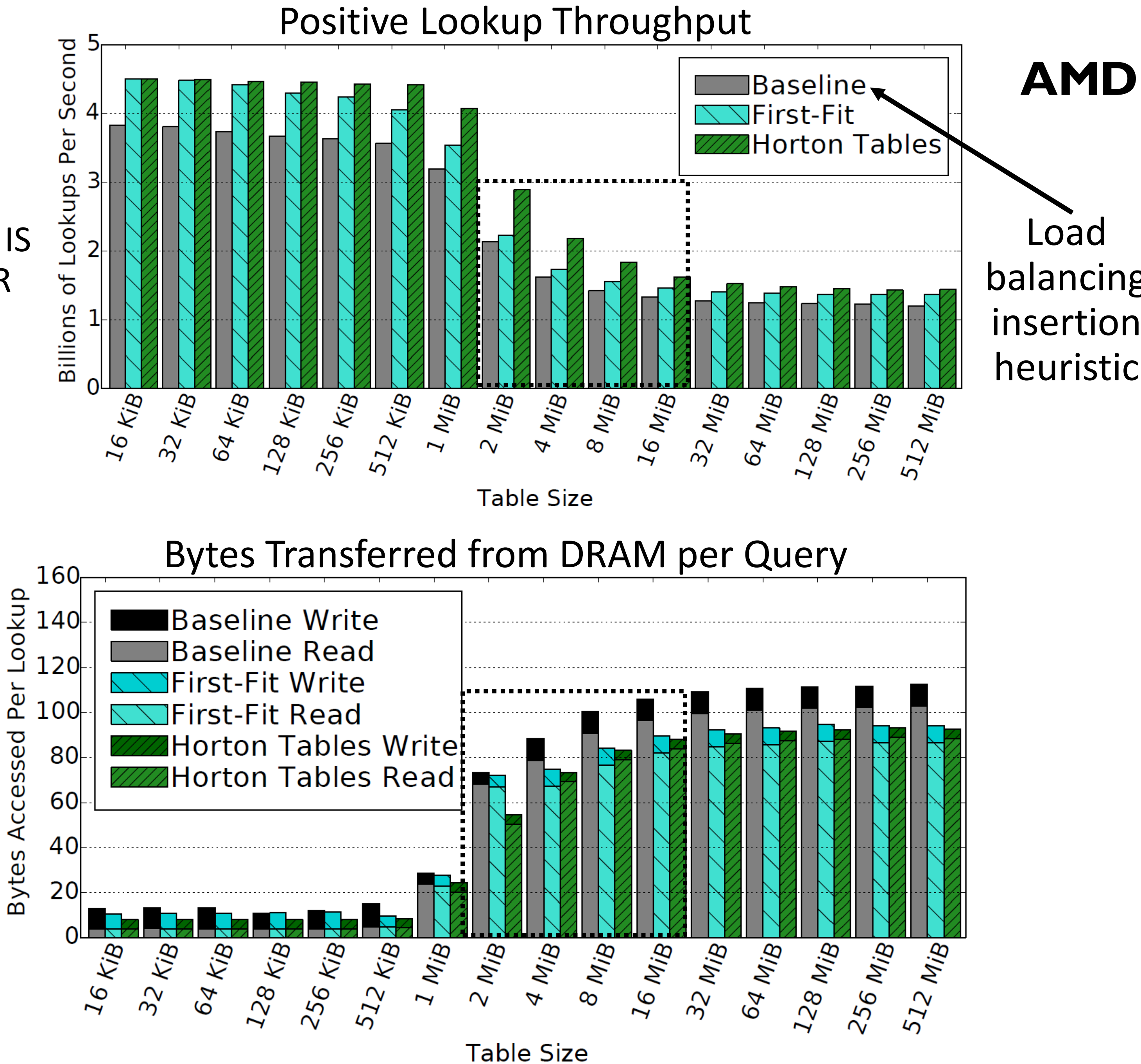
LOWER IS BETTER



RESULTS

POSITIVE LOOKUPS

HIGHER IS BETTER

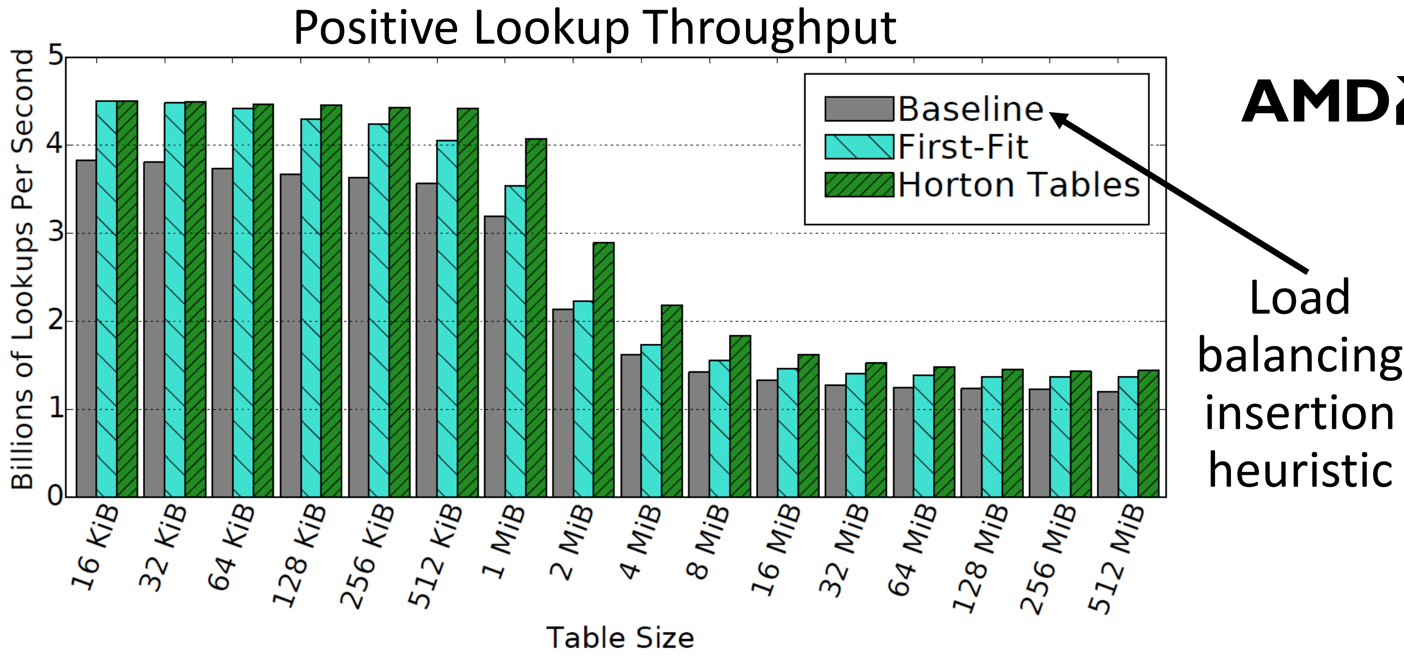


LOWER IS BETTER

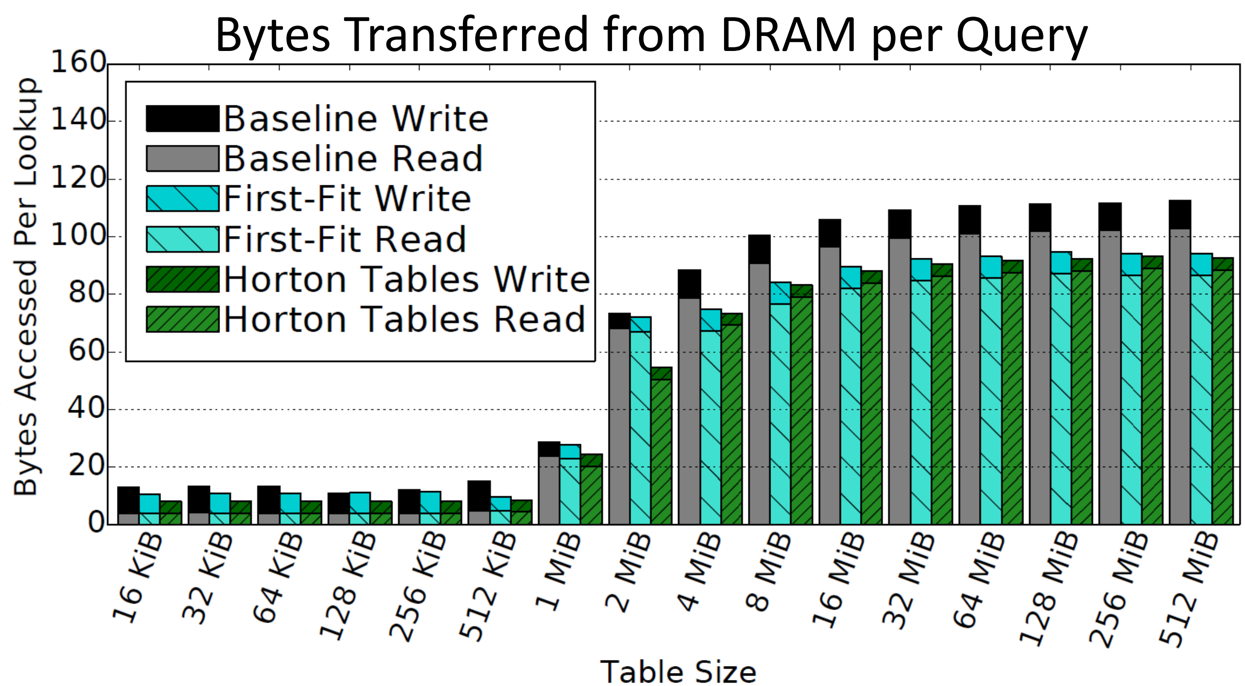
RESULTS

POSITIVE LOOKUPS

HIGHER IS BETTER



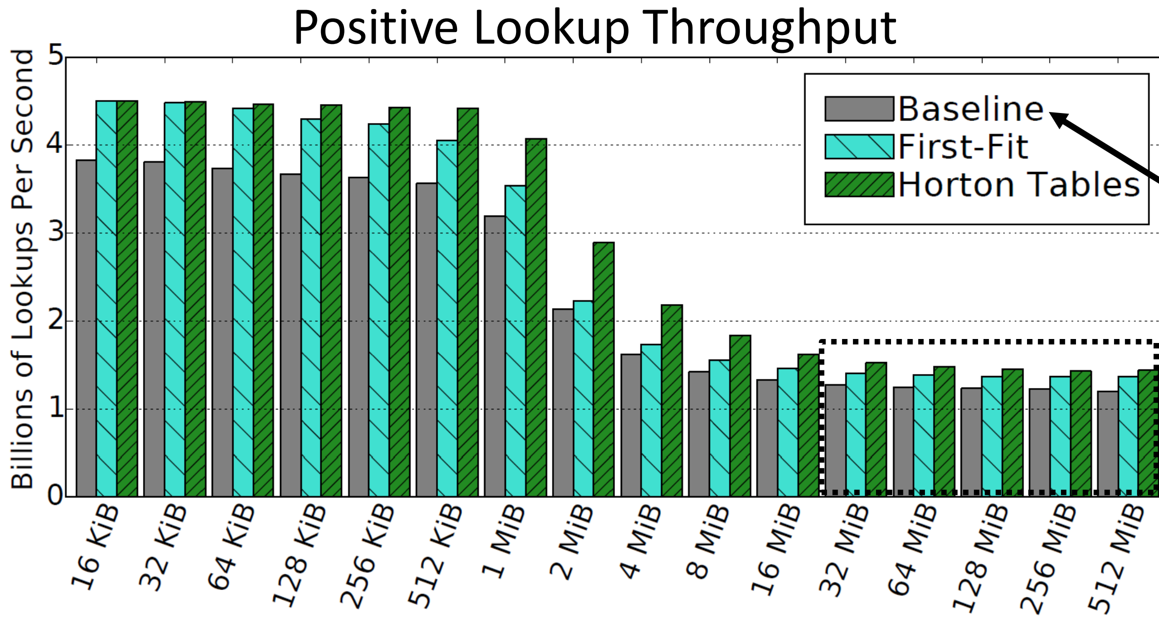
LOWER IS BETTER



RESULTS

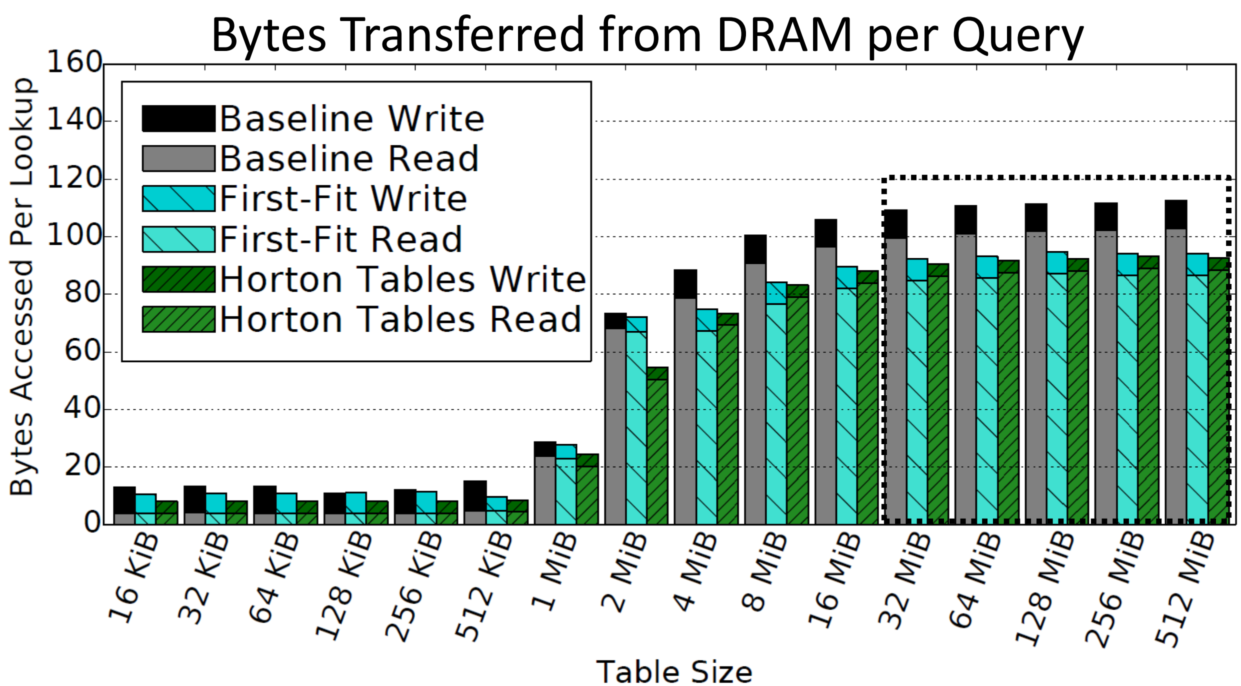
POSITIVE LOOKUPS

HIGHER IS BETTER



Load balancing insertion heuristic

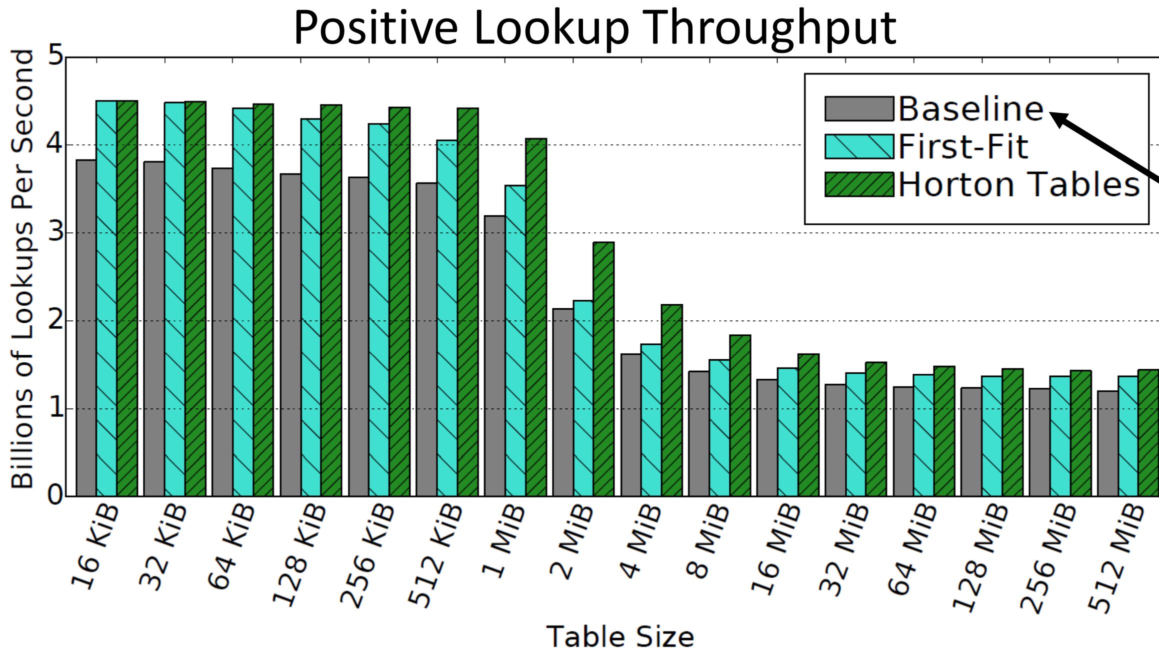
LOWER IS BETTER



RESULTS

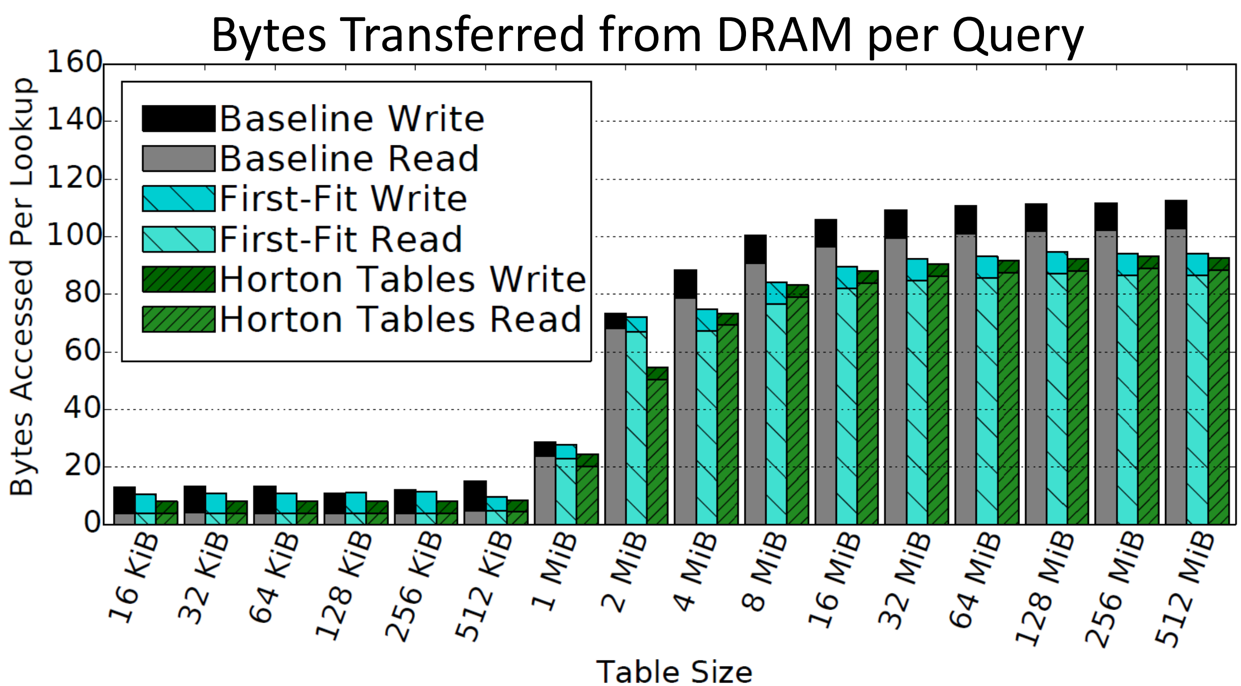
POSITIVE LOOKUPS

HIGHER IS BETTER

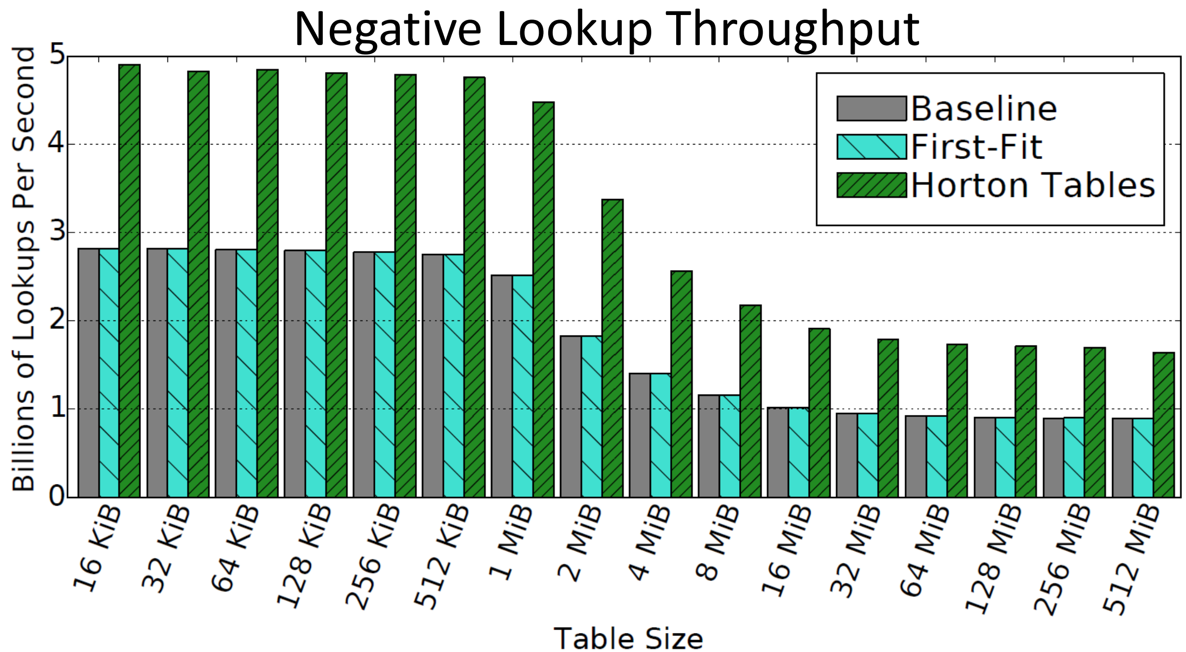


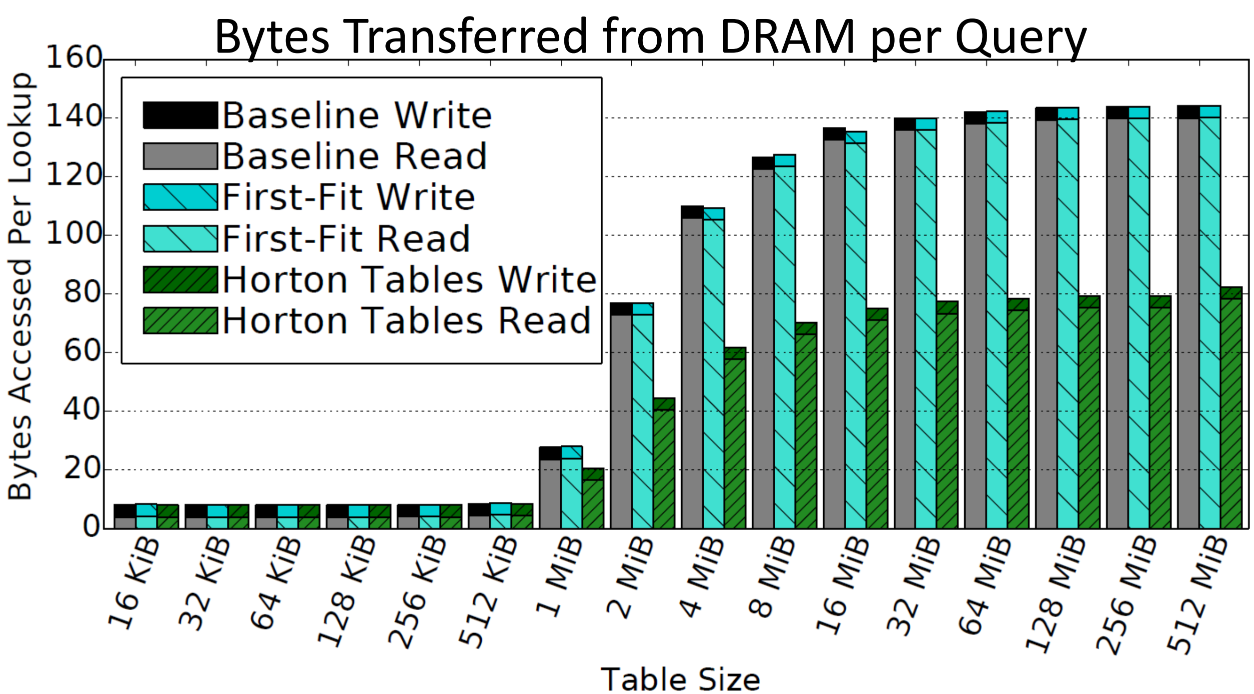
Load balancing insertion heuristic

LOWER IS BETTER



RESULTS
NEGATIVE LOOKUPS

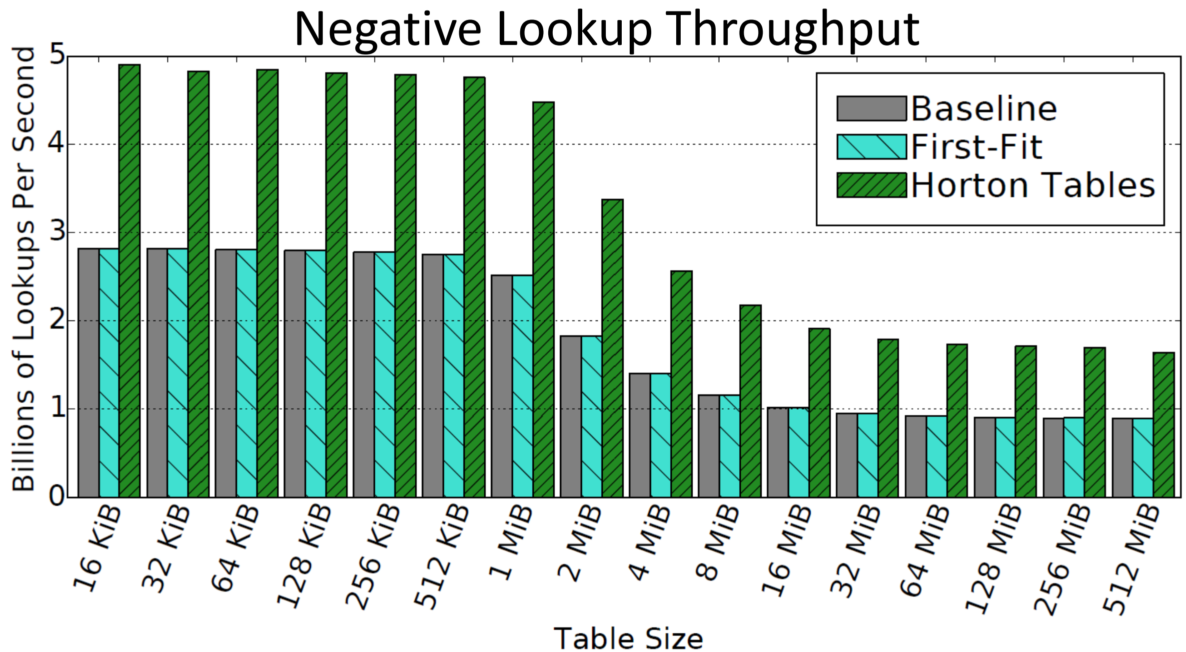


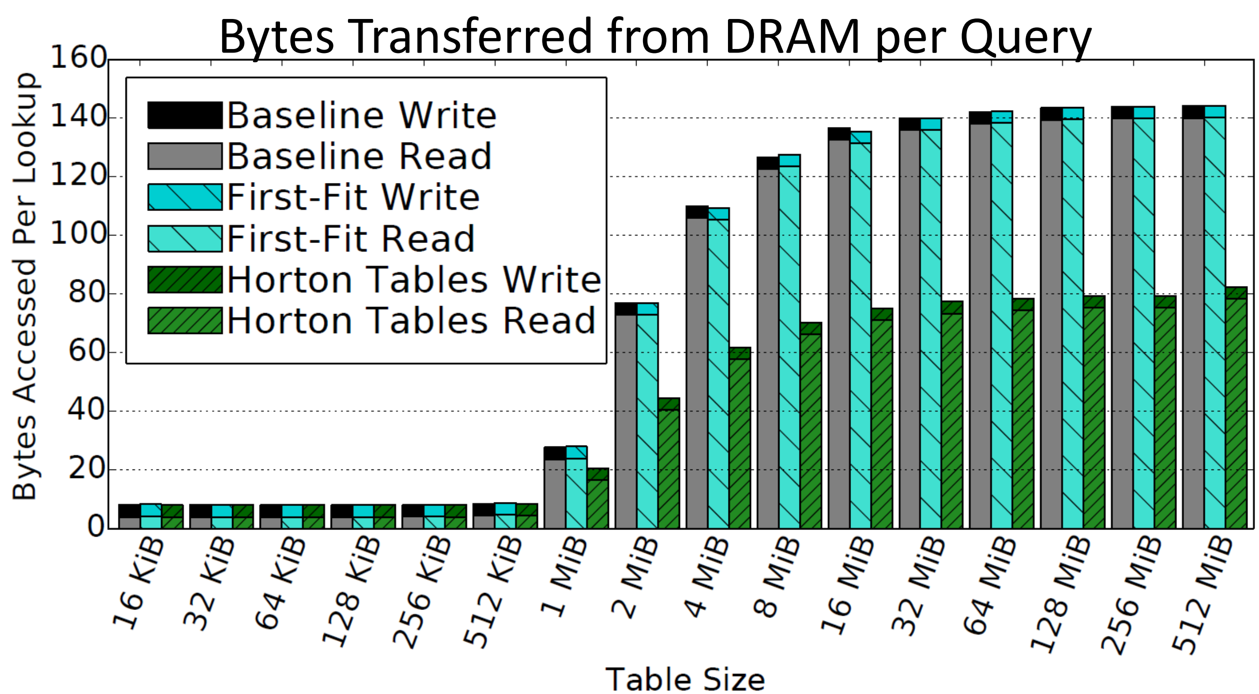


RESULTS

NEGATIVE LOOKUPS

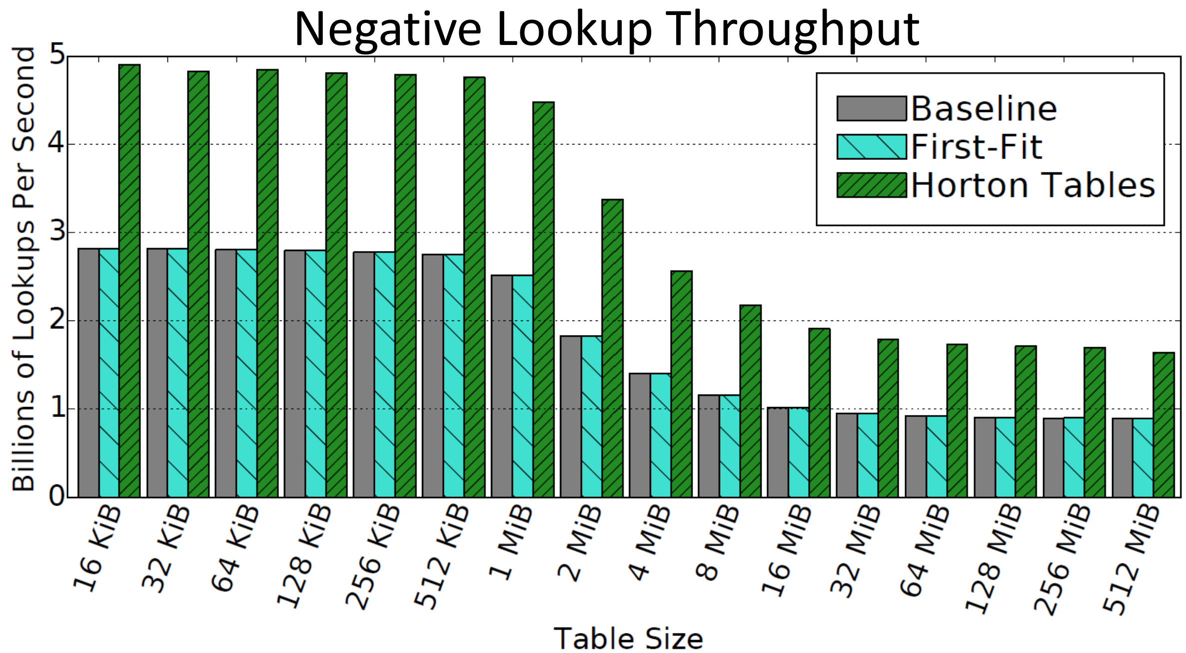
HIGHER IS BETTER



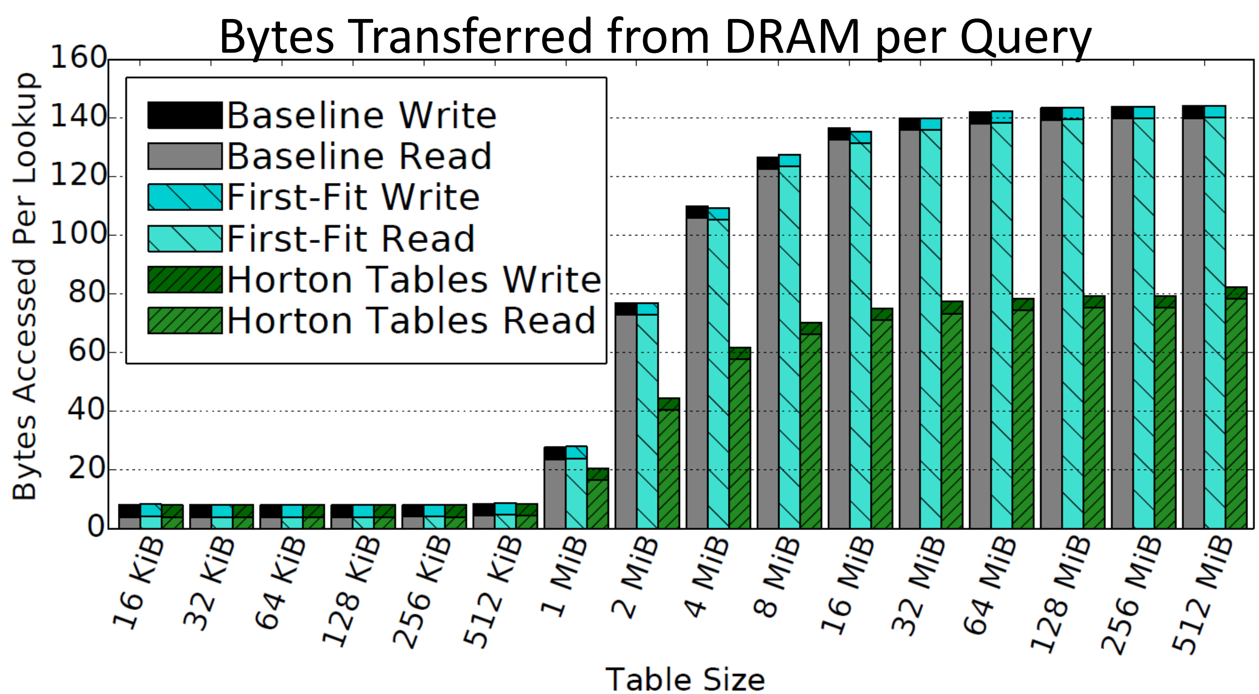


RESULTS
NEGATIVE LOOKUPS

HIGHER IS BETTER

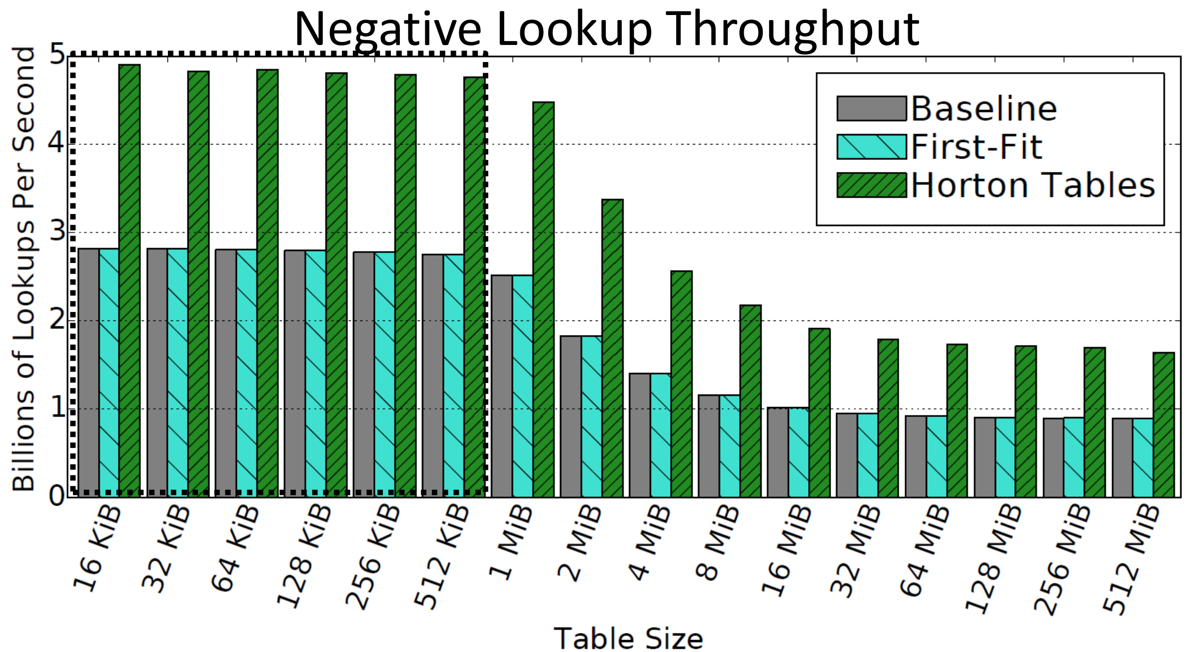


LOWER IS BETTER

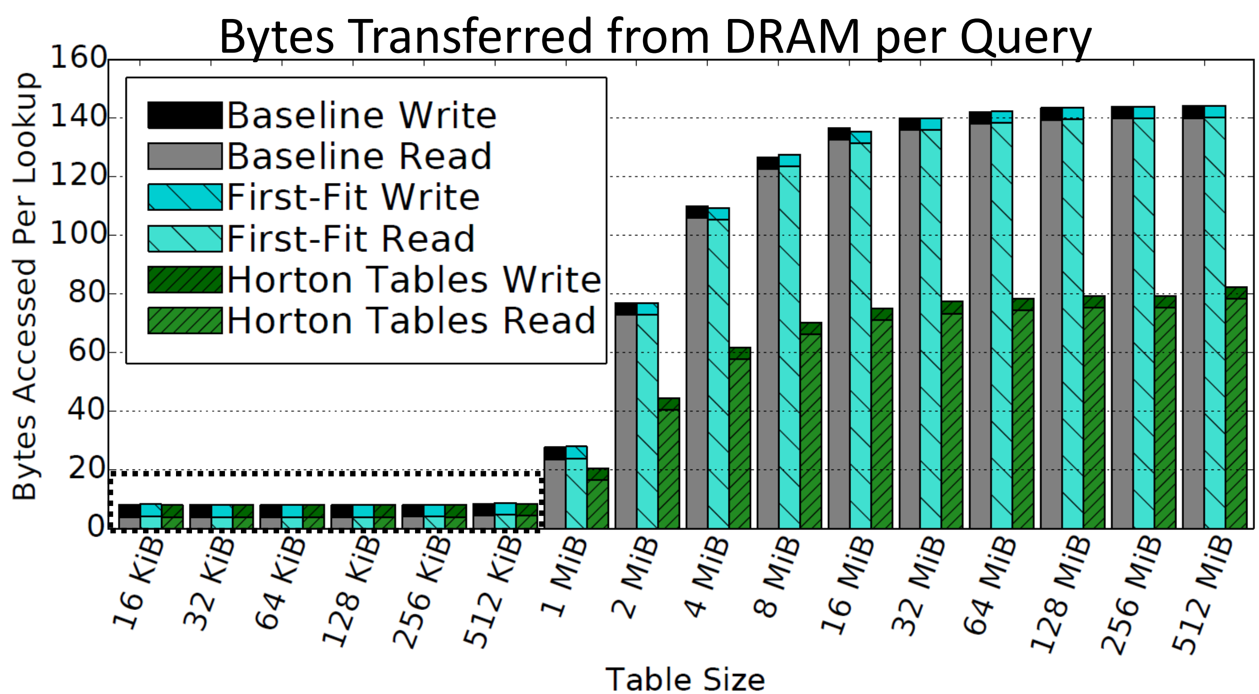


RESULTS
NEGATIVE LOOKUPS

HIGHER IS BETTER



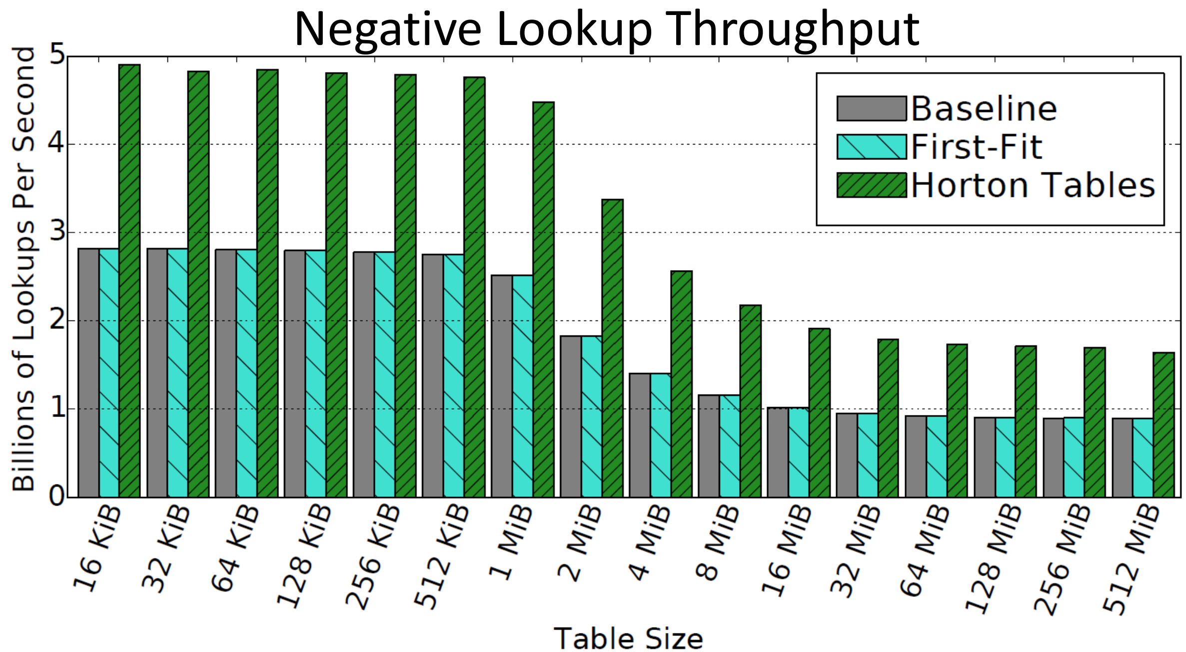
LOWER IS BETTER



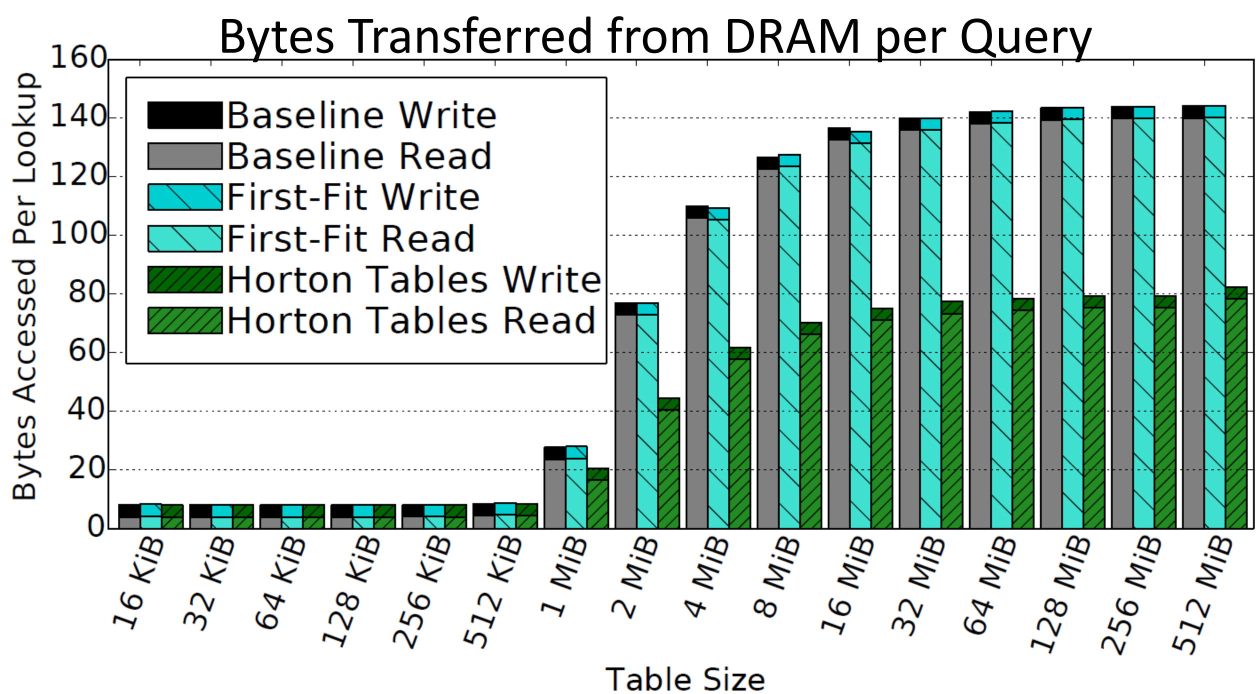
RESULTS

NEGATIVE LOOKUPS

HIGHER IS BETTER



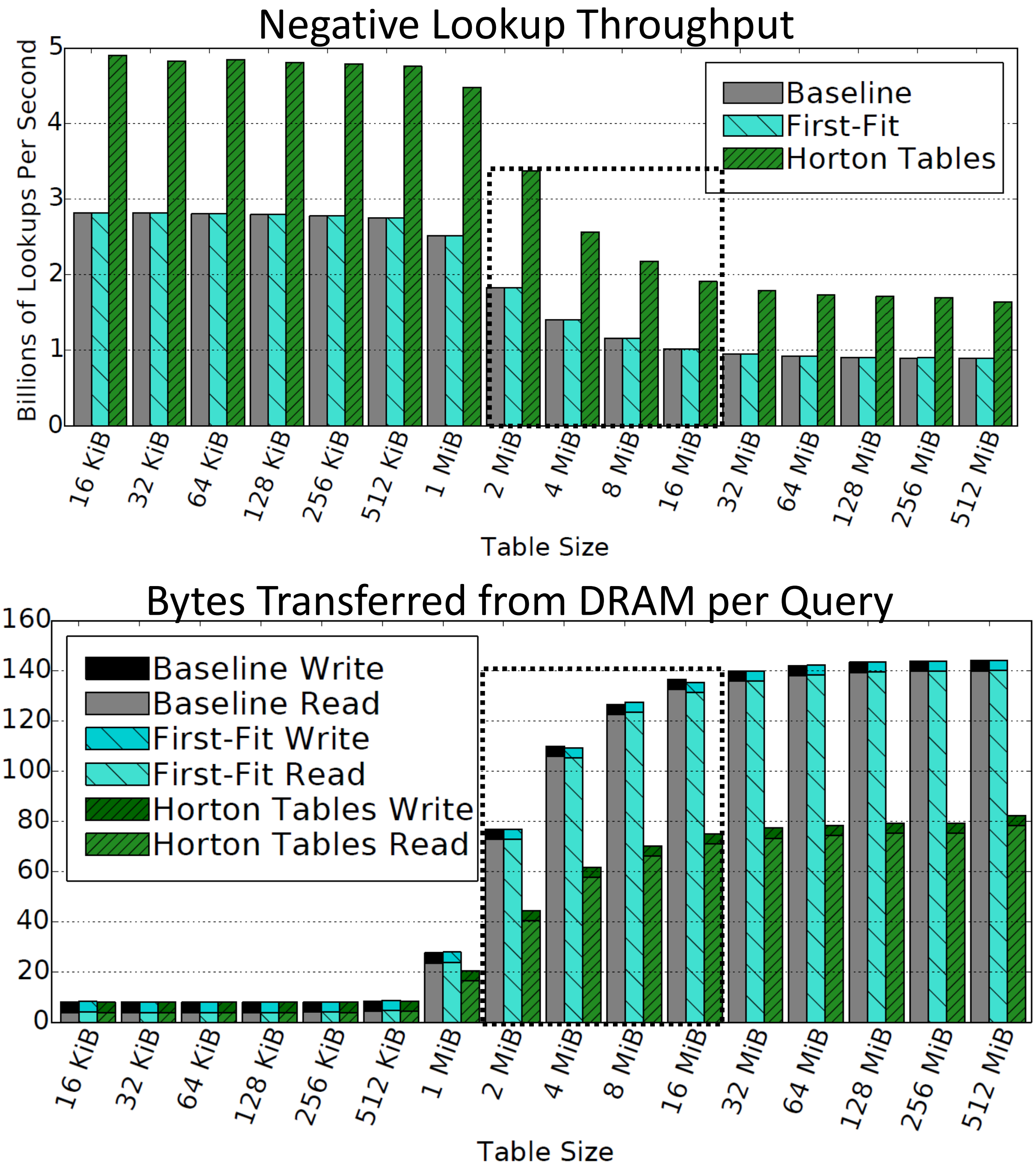
LOWER IS BETTER



RESULTS

NEGATIVE LOOKUPS

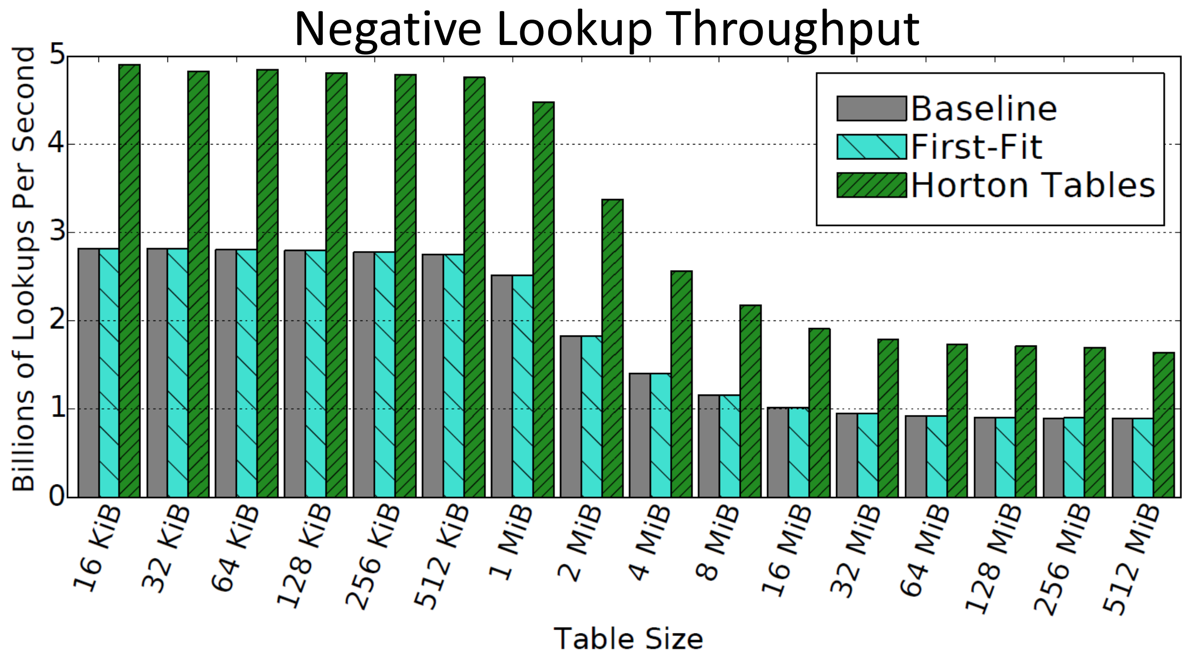
HIGHER IS BETTER



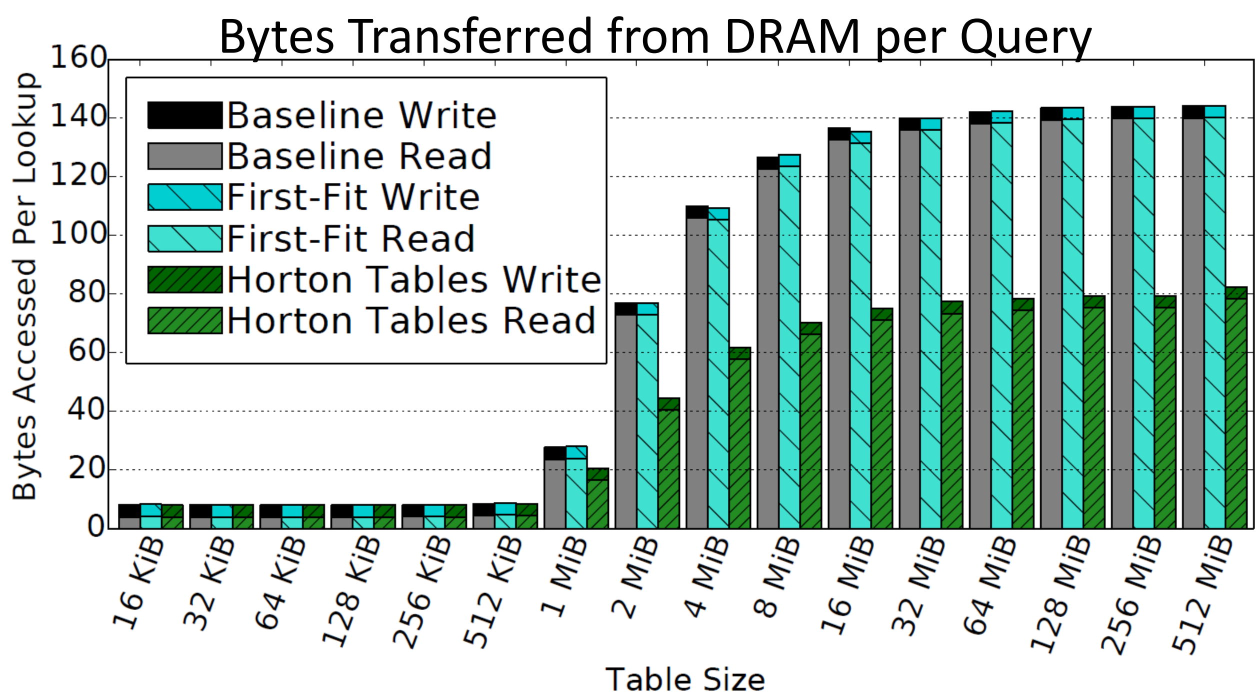
LOWER IS BETTER

RESULTS
NEGATIVE LOOKUPS

HIGHER IS BETTER

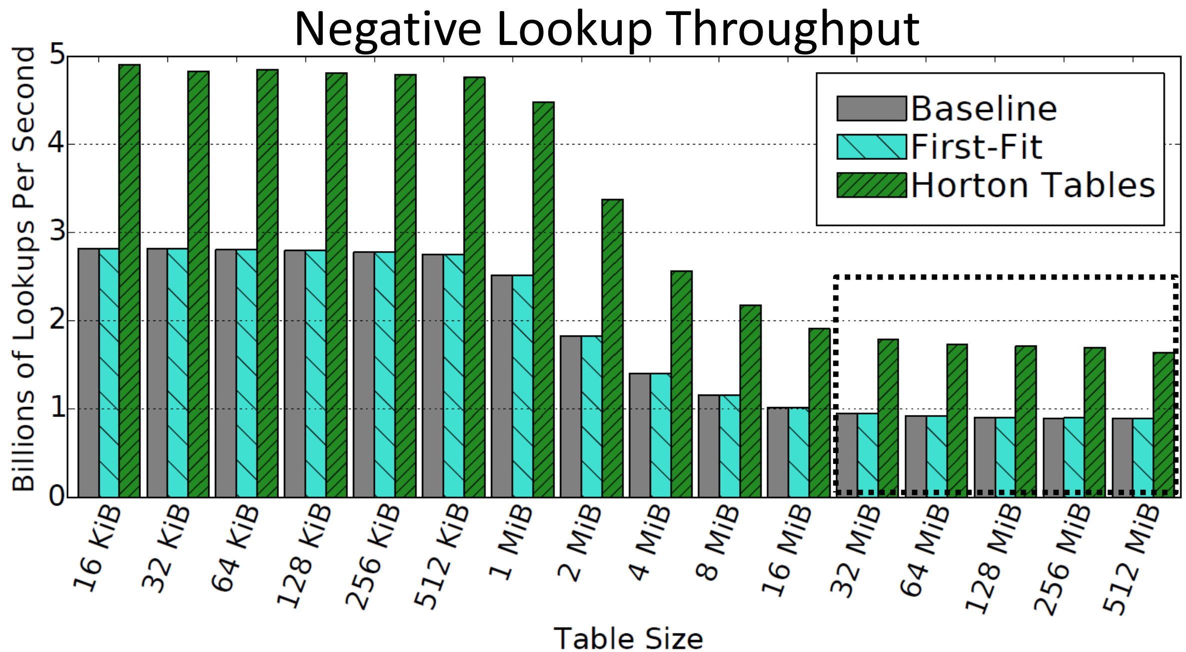


LOWER IS BETTER

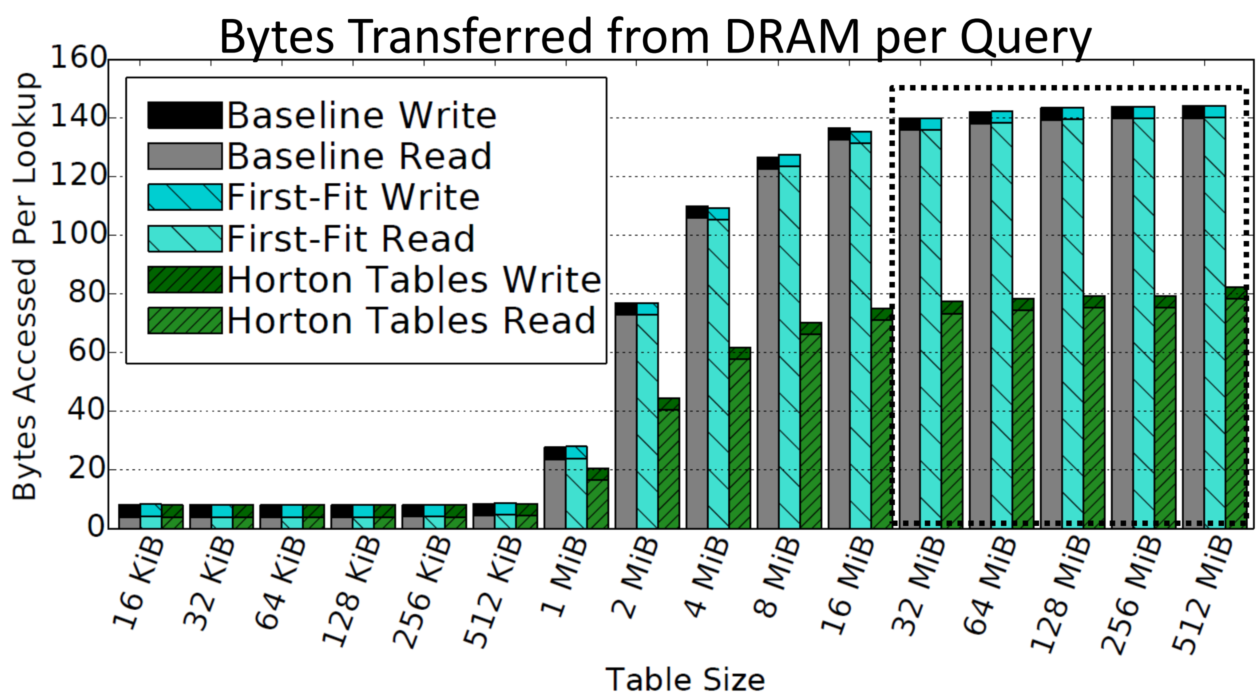


RESULTS
NEGATIVE LOOKUPS

HIGHER IS BETTER

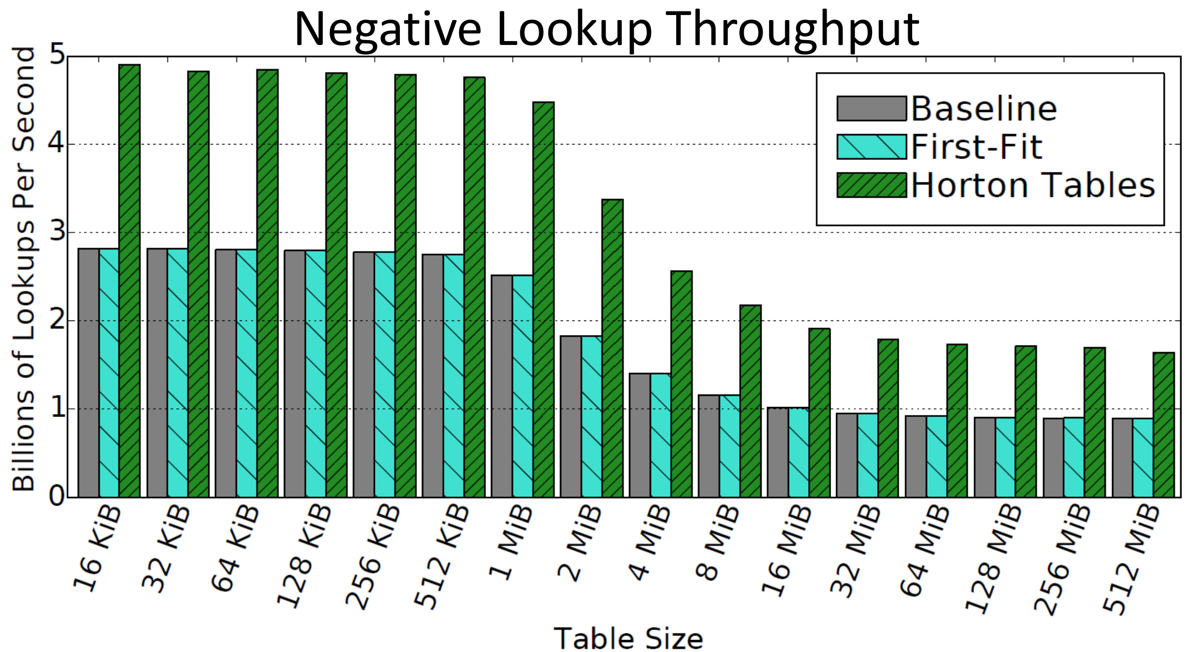


LOWER IS BETTER

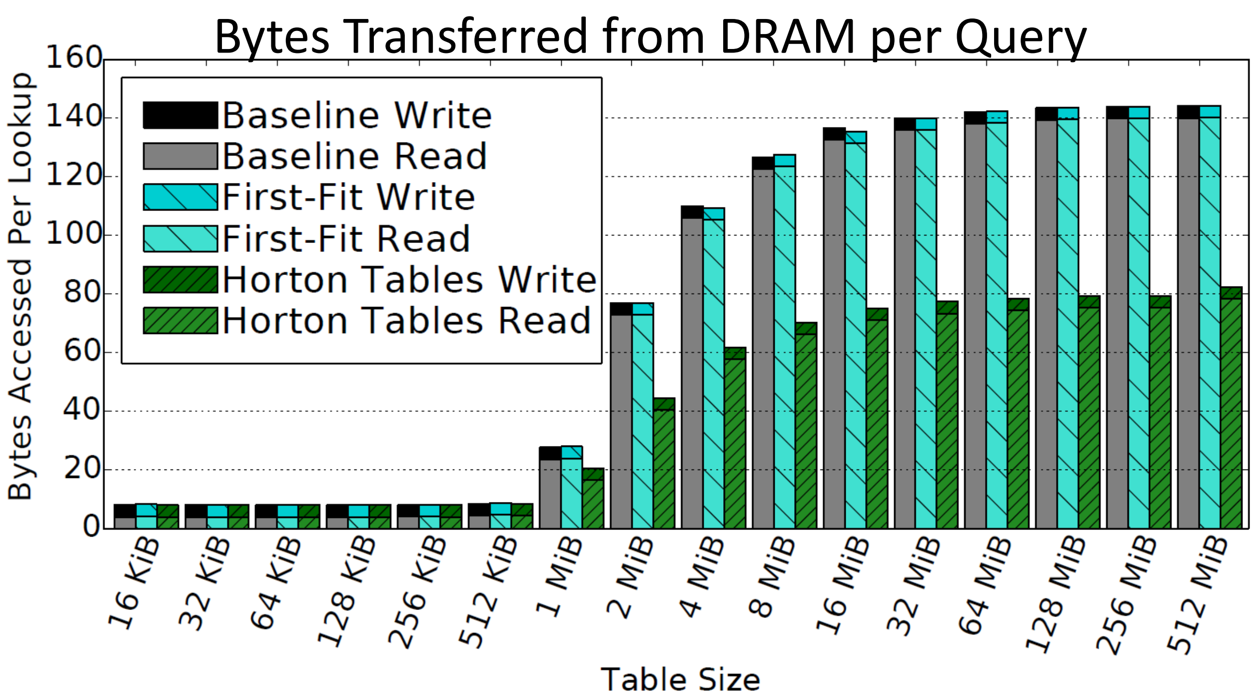


RESULTS
NEGATIVE LOOKUPS

HIGHER IS BETTER



LOWER IS BETTER



CONCLUSIONS FROM HORTON TABLES

CONCLUSIONS FROM HORTON TABLES

- Achieves lookup throughput that meets or exceeds prior approaches

CONCLUSIONS FROM HORTON TABLES

UCSDCSE Computer Science and Engineering AMD

- Achieves lookup throughput that meets or exceeds prior approaches
- The throughput improvement is achieved by reducing the number of cache lines that need to be accessed per lookup query to **at most 1.18 for positive lookups and 1.06 for negative lookups.**

CONCLUSIONS FROM HORTON TABLES

- Achieves lookup throughput that meets or exceeds prior approaches

- The throughput improvement is achieved by reducing the number of cache lines that need to be accessed per lookup query to **at most 1.18 for positive lookups and 1.06 for negative lookups.**

- **Reducing cache accesses yields corresponding throughput improvements of 5% to 35% and 73% to 89%, for pos. and neg. lookups, respectively, on a very full table.**

CONCLUSIONS FROM HORTON TABLES

UCSDCSE
Computer Science and Engineering

AMD

- Achieves lookup throughput that meets or exceeds prior approaches
- The throughput improvement is achieved by reducing the number of cache lines that need to be accessed per lookup query to **at most 1.18 for positive lookups and 1.06 for negative lookups.**
- **Reducing cache accesses yields corresponding throughput improvements of 5% to 35% and 73% to 89%, for pos. and neg. lookups, respectively, on a very full table.**
- Optimizing hash table algorithms is important because of their wide use throughout all segments of computing (e.g., scientific computing and databases, data compression, computer graphics and data visualization).

FUTURE WORK

- Evaluation of insertions and deletions and their optimization
  - Write- and update-heavy workloads should also benefit from Horton tables approach.

- Application of Horton tables to data warehousing and analysis applications
  - Database operators' implementations (e.g., hash joins and grouping hash tables)
  - Key-value stores

- Additional indices for speeding up lookups, insertions, and deletions

- Evaluation of Horton tables on new and emerging memory subsystems as well as tailoring the technique for persistent storage technologies such as SSDs

QUESTIONS?

Thanks for your attention.

DISCLAIMER & ATTRIBUTION

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

BACKUP SLIDES

HORTON TABLES

SHARING OF REMAP ENTRIES

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

- We permit a single remap entry to reference multiple remapped elements.
- Deleting remap entries is possible by having elements that share remap entries map to the same secondary bucket (see our paper for details).

HORTON TABLES

SHARING OF REMAP ENTRIES

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

INSERT 27

- We permit a single remap entry to reference multiple remapped elements.
- Deleting remap entries is possible by having elements that share remap entries map to the same secondary bucket (see our paper for details).

HORTON TABLES

SHARING OF REMAP ENTRIES

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{primary}$

INSERT 27

- We permit a single remap entry to reference multiple remapped elements.
- Deleting remap entries is possible by having elements that share remap entries map to the same secondary bucket (see our paper for details).

HORTON TABLES

SHARING OF REMAP ENTRIES

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

INSERT 27 → $H_{primary}$ → bucket 2

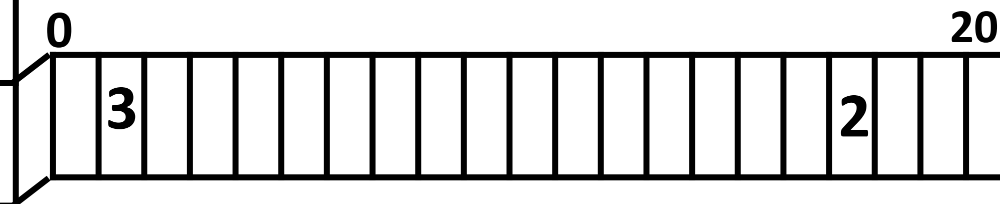
We conclude that bucket 2 has no free slots, so we need to remap it.

- We permit a single remap entry to reference multiple remapped elements.
- Deleting remap entries is possible by having elements that share remap entries map to the same secondary bucket (see our paper for details).

HORTON TABLES

SHARING OF REMAP ENTRIES

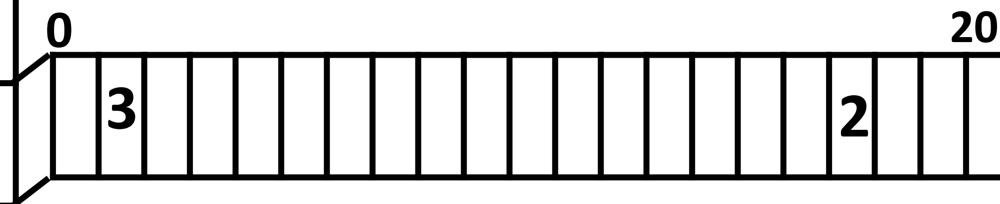
0	**8**	**5**	**13**	**EMPTY**
1	**33**	**EMPTY**	**15**	**2**
2	**35**	**18**	**22**	**REA**
3	**EMPTY**	**EMPTY**	**16**	**37**
4	**17**	**6**	**21**	**EMPTY**
5	**9**	**24**	**23**	**EMPTY**



HORTON TABLES

SHARING OF REMAP ENTRIES

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



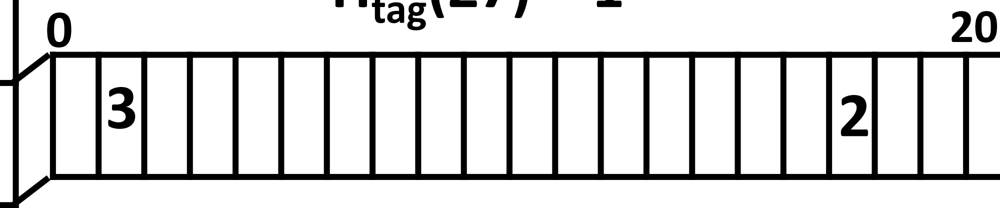
Compute the H_{tag} on the key

HORTON TABLES

SHARING OF REMAP ENTRIES

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{tag}(27) = 1$

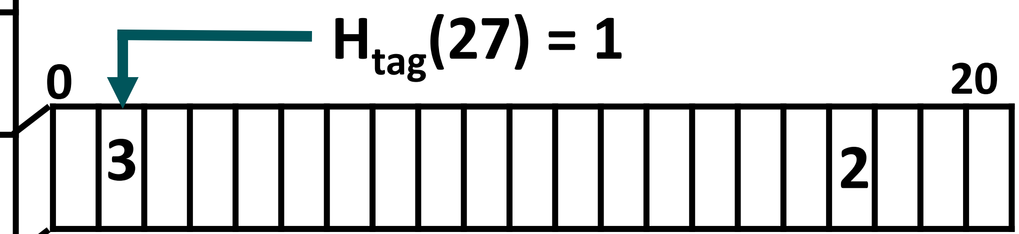


Compute the H_{tag} on the key

HORTON TABLES

SHARING OF REMAP ENTRIES

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

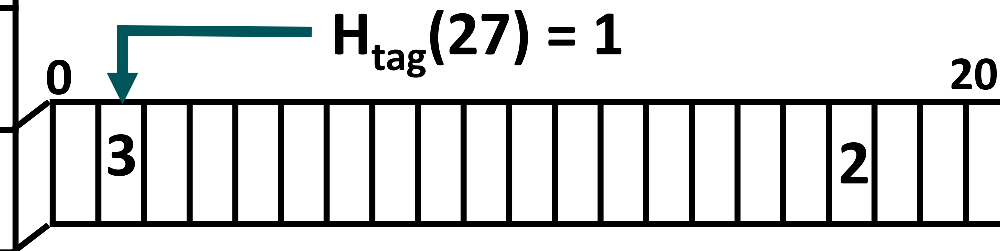


Compute the H_{tag} on the key

HORTON TABLES

SHARING OF REMAP ENTRIES

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

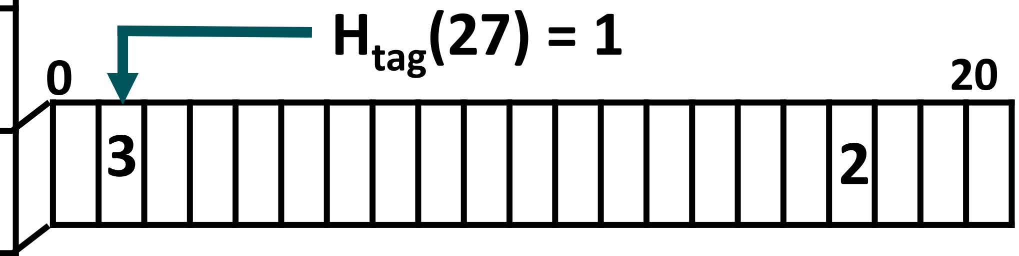


We see that the remap entry is set, so we try to use R_3 to insert 27.

HORTON TABLES

SHARING OF REMAP ENTRIES

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



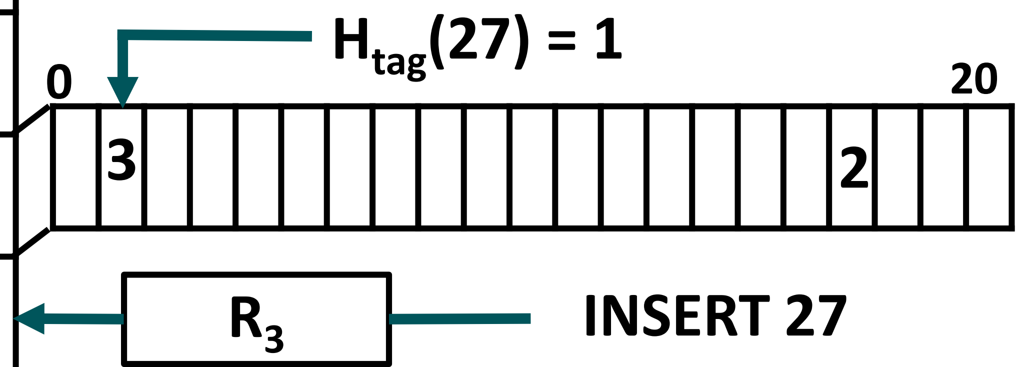
INSERT 27

We see that the remap entry is set, so we try to use R_3 to insert 27.

HORTON TABLES

SHARING OF REMAP ENTRIES

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

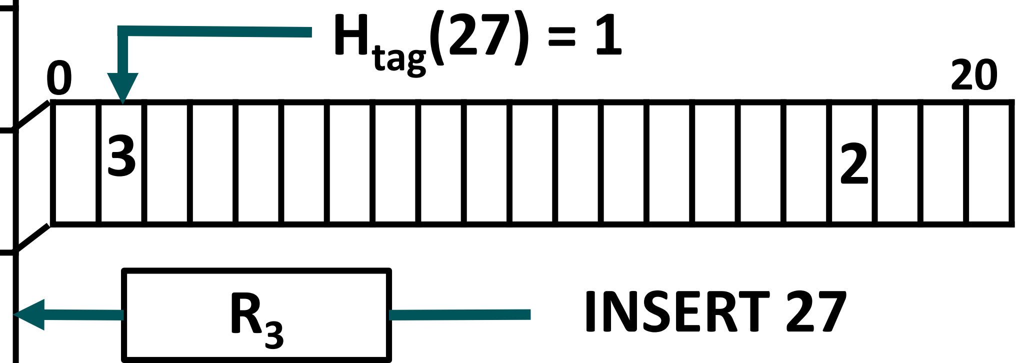


We see that the remap entry is set, so we try to use R_3 to insert 27.

HORTON TABLES

SHARING OF REMAP ENTRIES

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

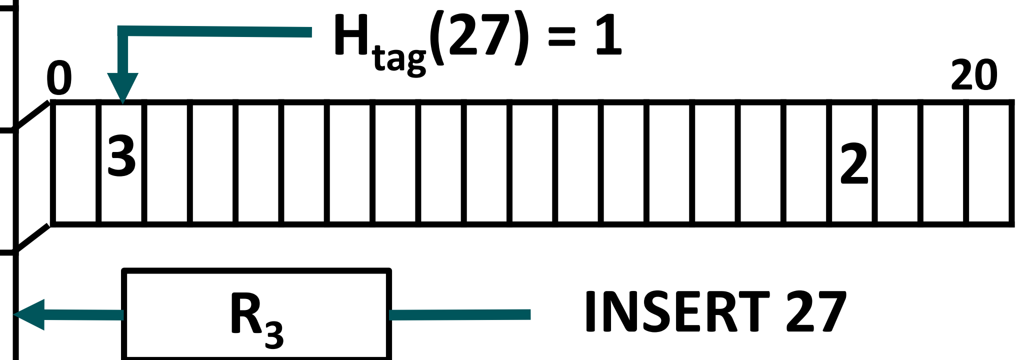


The insertion succeeds because the secondary bucket (3) has a free slot.

HORTON TABLES

SHARING OF REMAP ENTRIES

0	**8**	**5**	**13**	**EMPTY**
1	**33**	**EMPTY**	**15**	**2**
2	**35**	**18**	**22**	**REA**
3	**EMPTY**	**27**	**16**	**37**
4	**17**	**6**	**21**	**EMPTY**
5	**9**	**24**	**23**	**EMPTY**



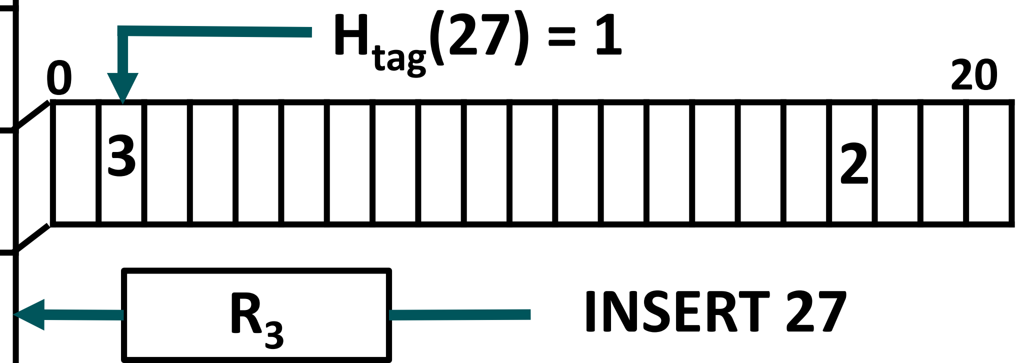
The insertion succeeds because the secondary bucket (3) has a free slot.

- If bucket 3 had been full, we could have swapped 27 with another item from 27’s primary bucket (e.g., 35 in bucket 2) and remapped that item instead.

HORTON TABLES

SHARING OF REMAP ENTRIES

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



The insertion succeeds because the secondary bucket (3) has a free slot.

- If bucket 3 had been full, we could have swapped 27 with another item from 27's primary bucket (e.g., 35 in bucket 2) and remapped that item instead.

HORTON TABLES

DELETING ELEMENTS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

DELETE 8

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES
DELETING ELEMENTS

0	8	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{primary}$

DELETE 8

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{primary}$

DELETE 8

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

DELETE 27

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{primary}$

DELETE 27

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{primary}$ ← **DELETE 27**

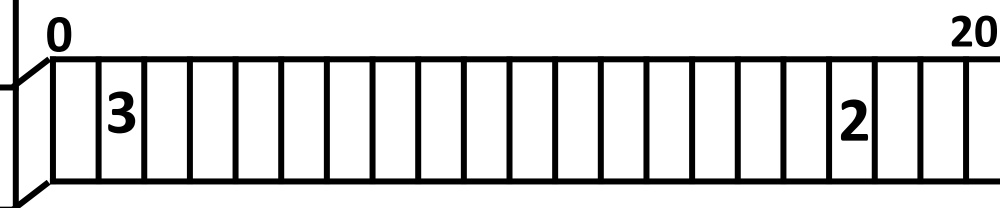
27 is not found in its primary bucket; we need to access the remap entry array.

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

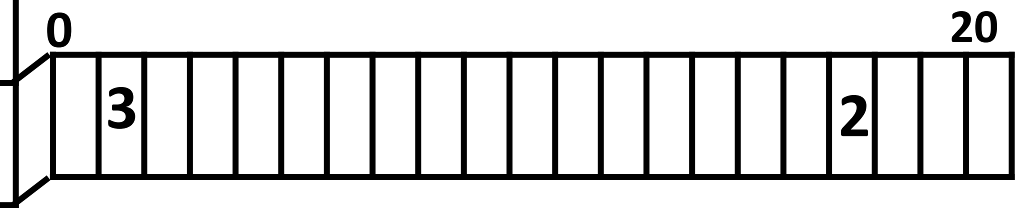


- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



27 is not found in its primary bucket; we need to access the remap entry array.

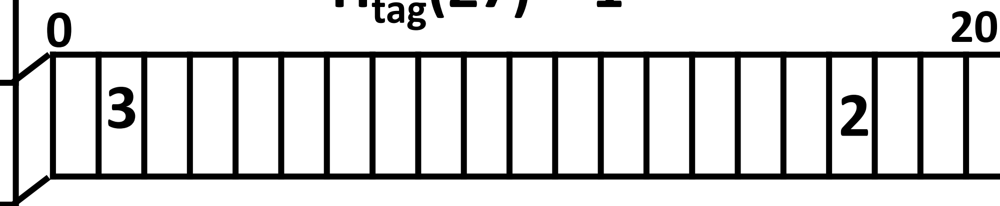
- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

$H_{tag}(27) = 1$



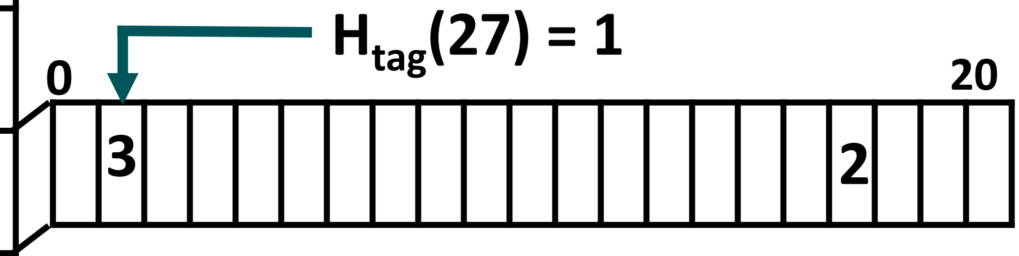
27 is not found in its primary bucket; we need to access the remap entry array.

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



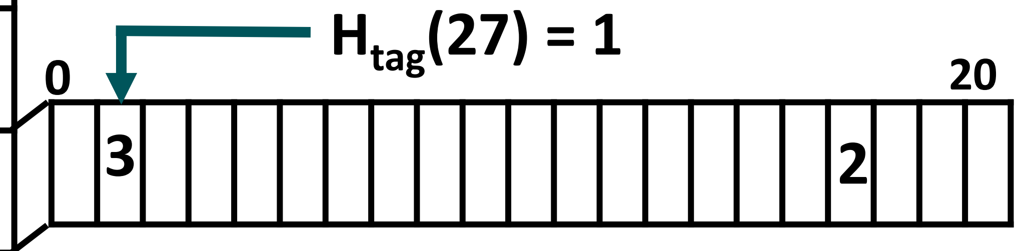
27 is not found in its primary bucket; we need to access the remap entry array.

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



DELETE 27

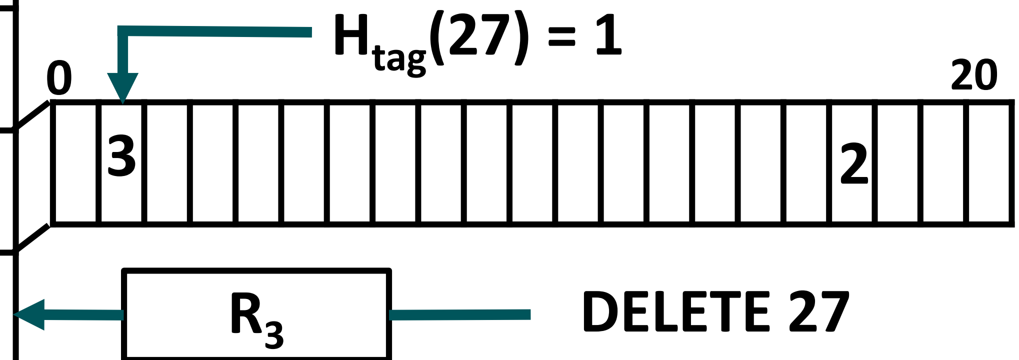
27 is not found in its primary bucket; we need to access the remap entry array.

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	27	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY

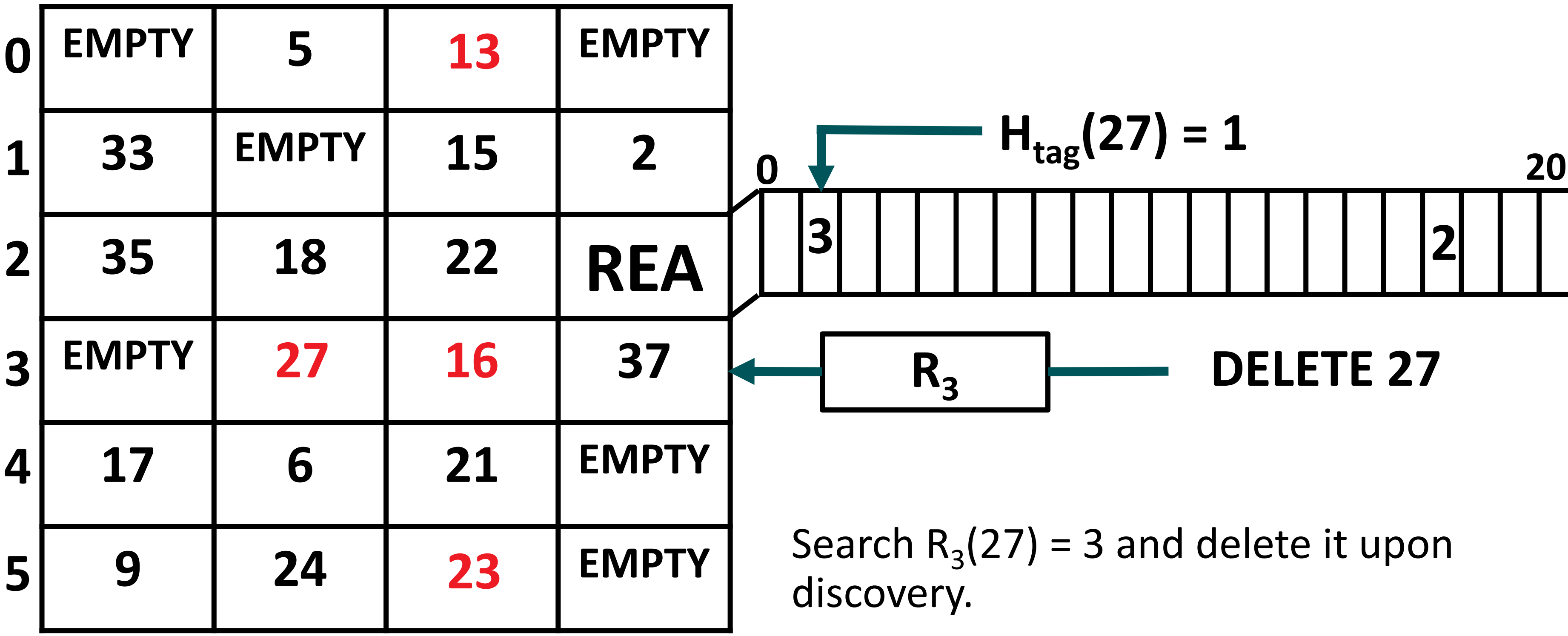


27 is not found in its primary bucket; we need to access the remap entry array.

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS



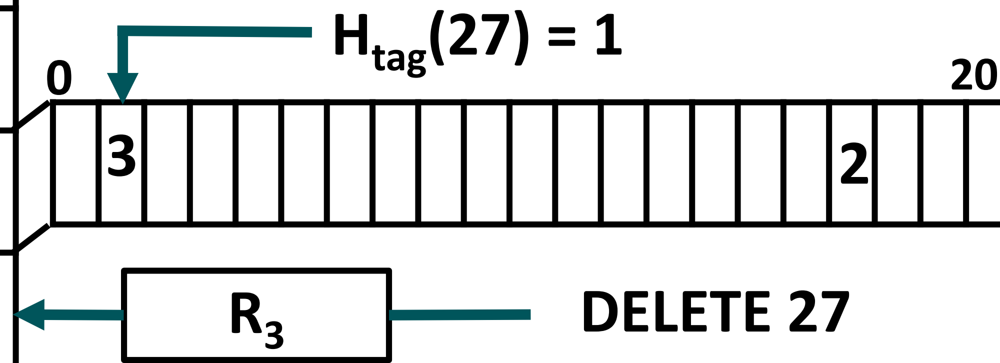
Search $R_3(27) = 3$ and delete it upon discovery.

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



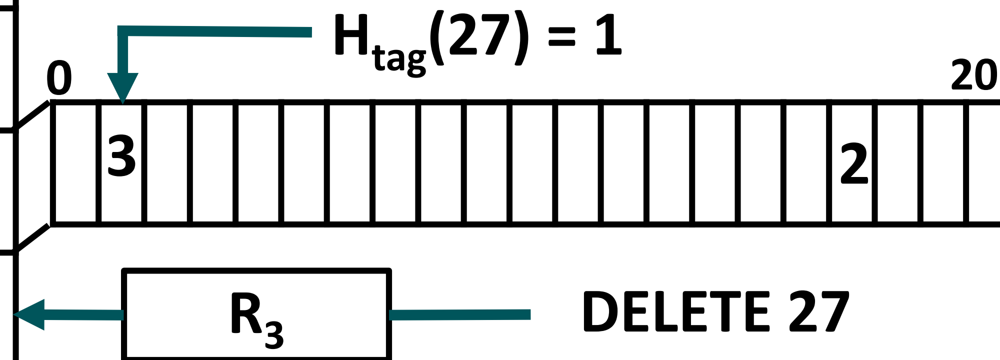
Search $R_3(27) = 3$ and delete it upon discovery.

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

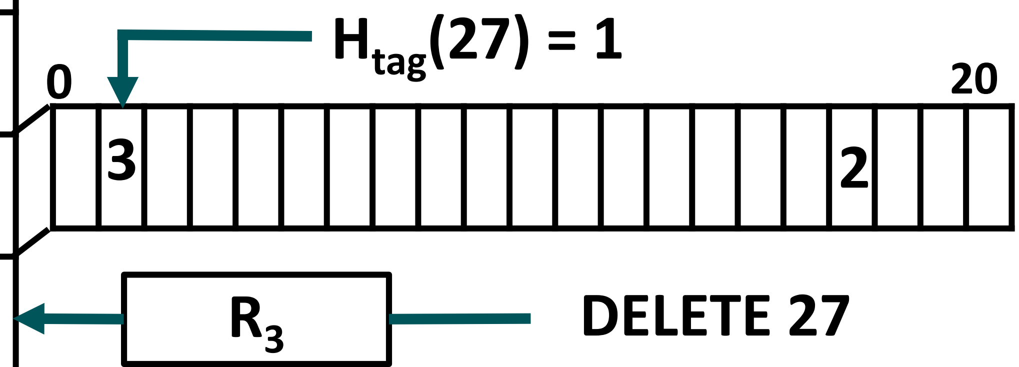
0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



Compute $H_{primary}$ on 16 and 37 and find that the remap entry is still active.

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



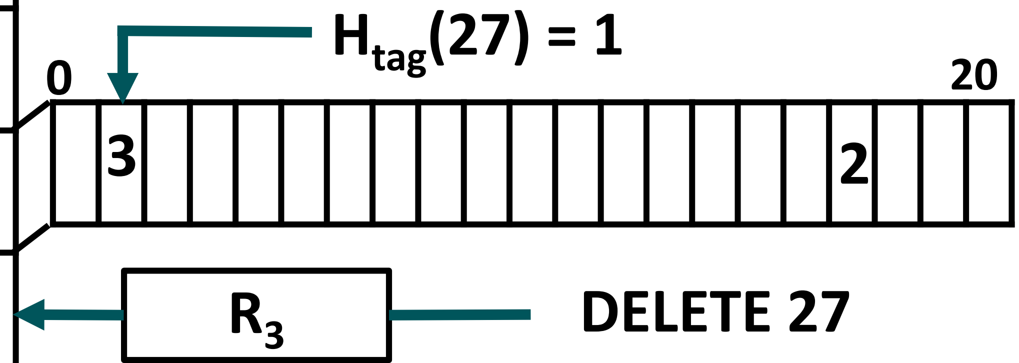
A subsequent deletion of 16 would cause the remap entry to be deleted.

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



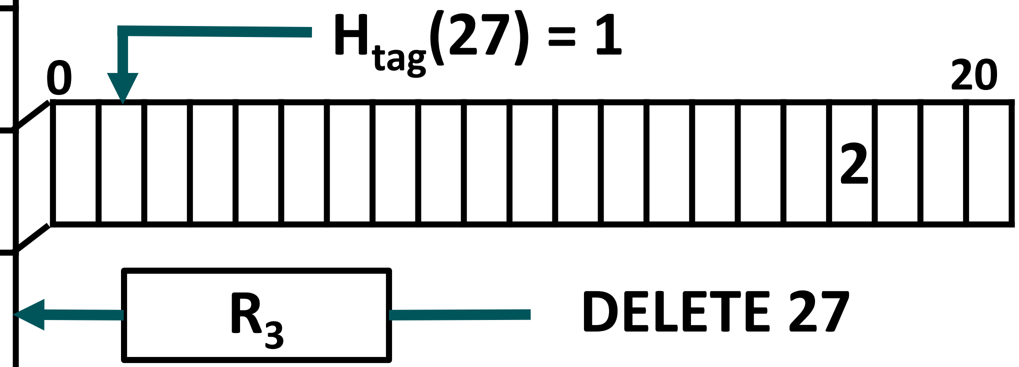
A subsequent deletion of 16 would cause the remap entry to be deleted.

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

DELETING ELEMENTS

0	EMPTY	5	13	EMPTY
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	EMPTY	37
4	17	6	21	EMPTY
5	9	24	23	EMPTY



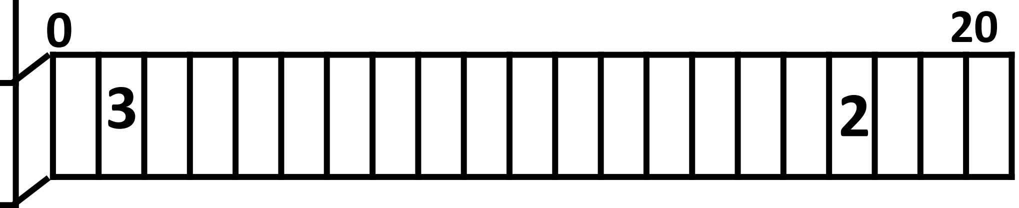
A subsequent deletion of 16 would cause the remap entry to be deleted.

- Deleting elements that are found in their primary bucket only requires accessing a single bucket
- A remapped element can be deleted by performing a secondary lookup followed by a deletion

HORTON TABLES

INSERTION WHERE ALL CANDIDATE SECONDARY BUCKETS ARE FULL

0	8	5	13	31
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	42
5	9	24	23	19

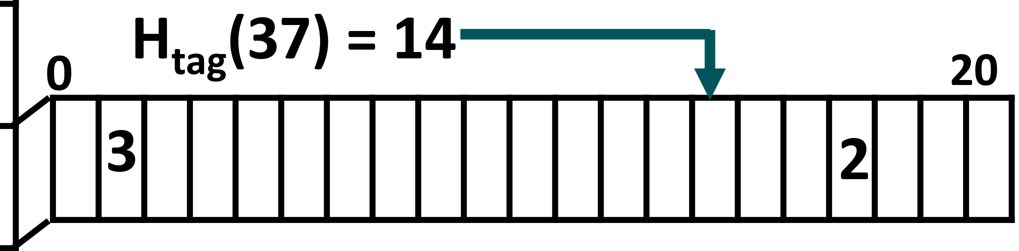


- Swap item with another from its primary bucket that can be remapped to a secondary bucket that is not full
- If this fails, then use cuckoo hashing
  - Preferably enforcing as we do that secondary items cannot displace primary items

HORTON TABLES

INSERTION WHERE ALL CANDIDATE SECONDARY BUCKETS ARE FULL

0	8	5	13	31
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	42
5	9	24	23	19



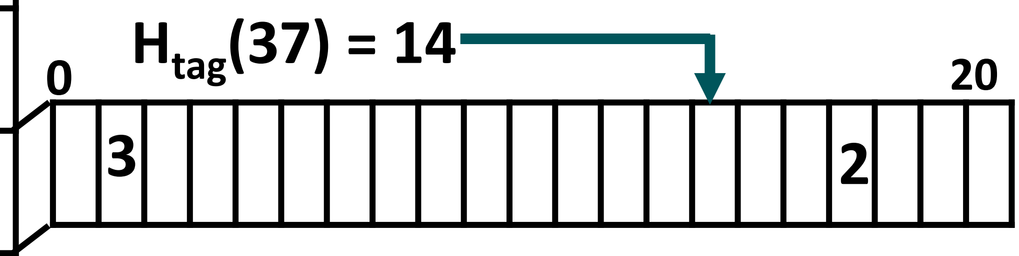
- Swap item with another from its primary bucket that can be remapped to a secondary bucket that is not full
- If this fails, then use cuckoo hashing
  - Preferably enforcing as we do that secondary items cannot displace primary items

HORTON TABLES

INSERTION WHERE ALL CANDIDATE SECONDARY BUCKETS ARE FULL

0	8	5	13	31
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	42
5	9	24	23	19

INSERT 37

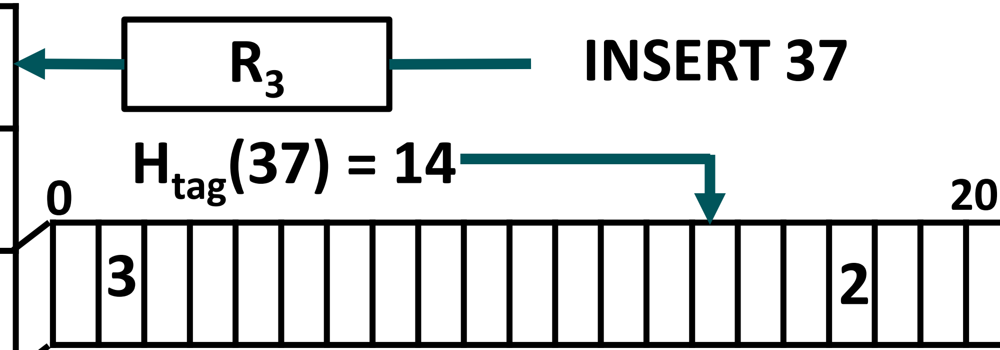


- Swap item with another from its primary bucket that can be remapped to a secondary bucket that is not full
- If this fails, then use cuckoo hashing
  - Preferably enforcing as we do that secondary items cannot displace primary items

HORTON TABLES

INSERTION WHERE ALL CANDIDATE SECONDARY BUCKETS ARE FULL

0	8	5	13	31
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	42
5	9	24	23	19

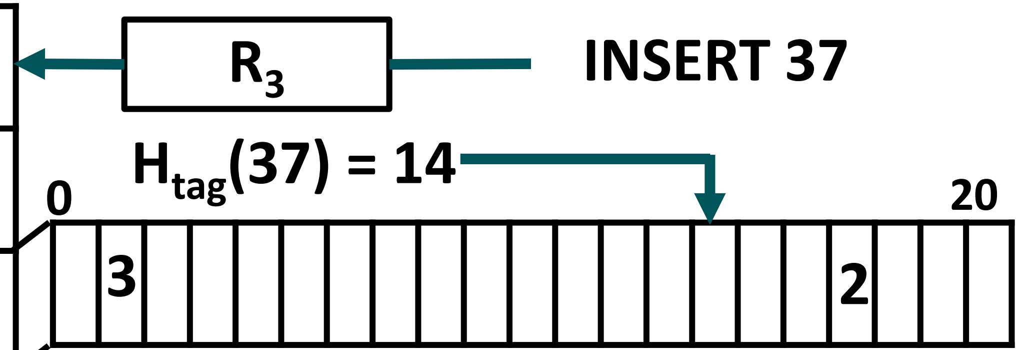


- Swap item with another from its primary bucket that can be remapped to a secondary bucket that is not full
- If this fails, then use cuckoo hashing
  - Preferably enforcing as we do that secondary items cannot displace primary items

HORTON TABLES

INSERTION WHERE ALL CANDIDATE SECONDARY BUCKETS ARE FULL

0	8	5	13	31
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	42
5	9	24	23	19



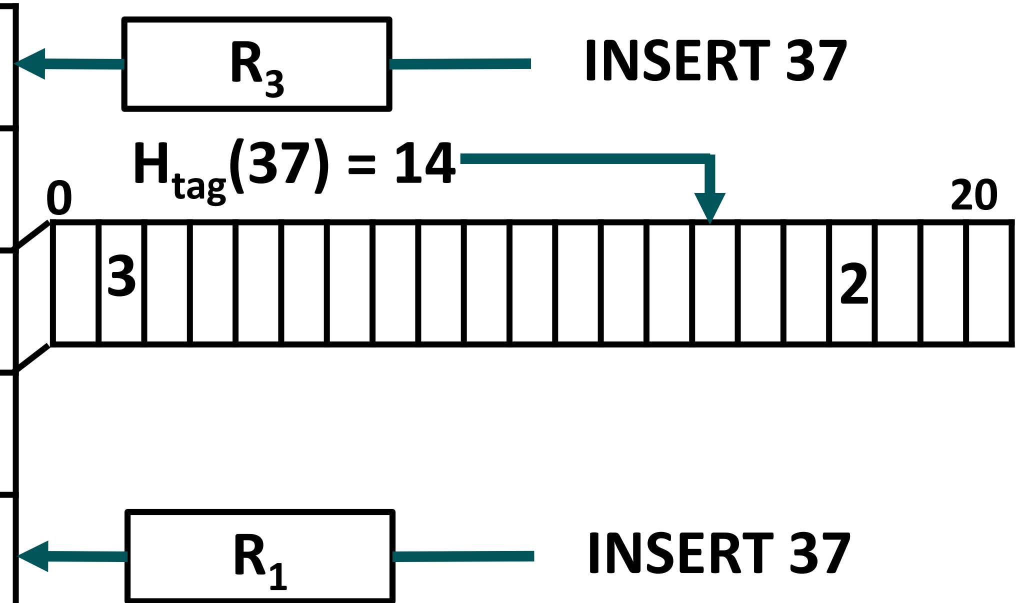
INSERT 37

- Swap item with another from its primary bucket that can be remapped to a secondary bucket that is not full
- If this fails, then use cuckoo hashing
  - Preferably enforcing as we do that secondary items cannot displace primary items

HORTON TABLES

INSERTION WHERE ALL CANDIDATE SECONDARY BUCKETS ARE FULL

0	8	5	13	31
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	42
5	9	24	23	19

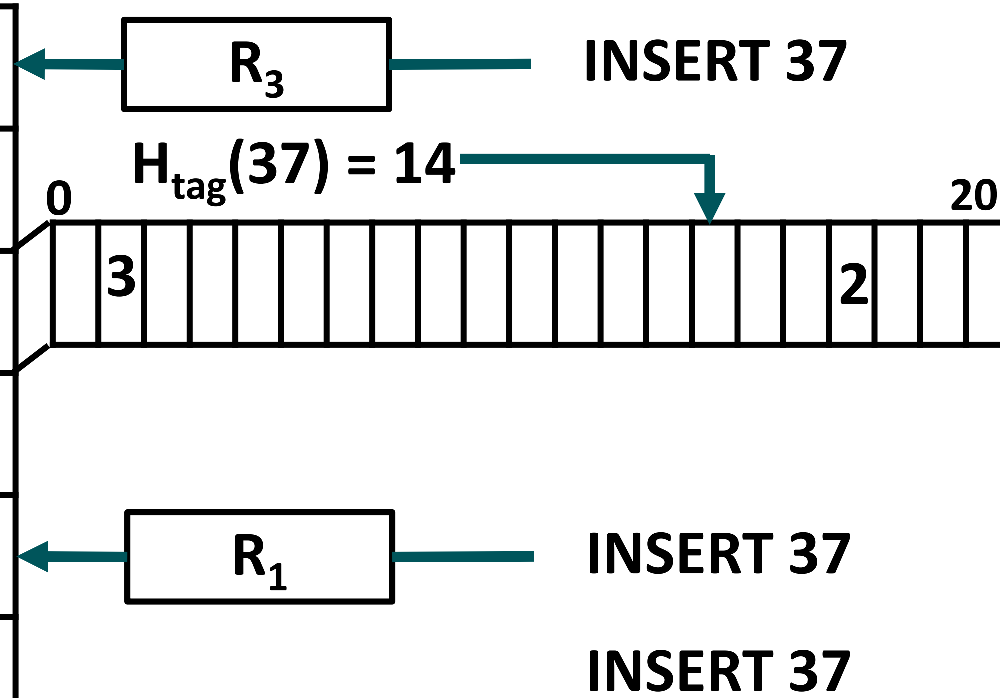


- Swap item with another from its primary bucket that can be remapped to a secondary bucket that is not full
- If this fails, then use cuckoo hashing
  - Preferably enforcing as we do that secondary items cannot displace primary items

HORTON TABLES

INSERTION WHERE ALL CANDIDATE SECONDARY BUCKETS ARE FULL

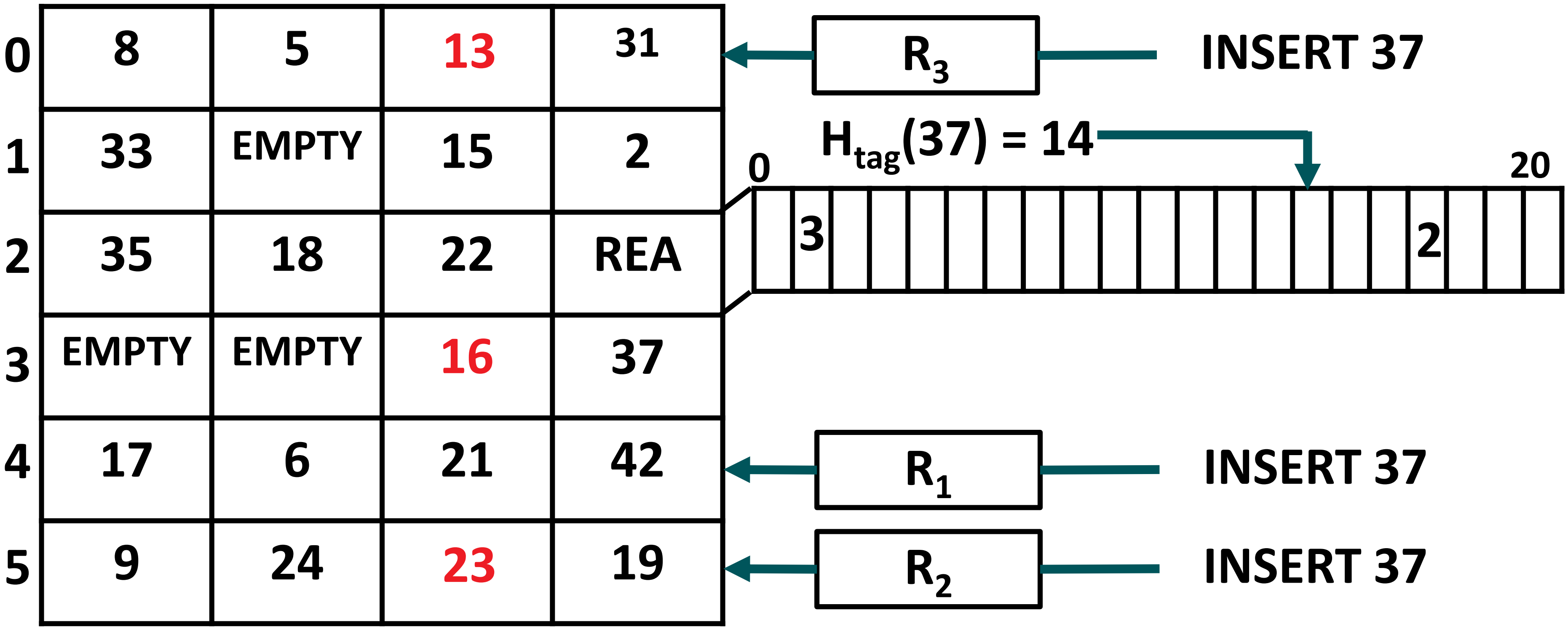
0	8	5	13	31
1	33	EMPTY	15	2
2	35	18	22	REA
3	EMPTY	EMPTY	16	37
4	17	6	21	42
5	9	24	23	19



- Swap item with another from its primary bucket that can be remapped to a secondary bucket that is not full
- If this fails, then use cuckoo hashing
  - Preferably enforcing as we do that secondary items cannot displace primary items

HORTON TABLES

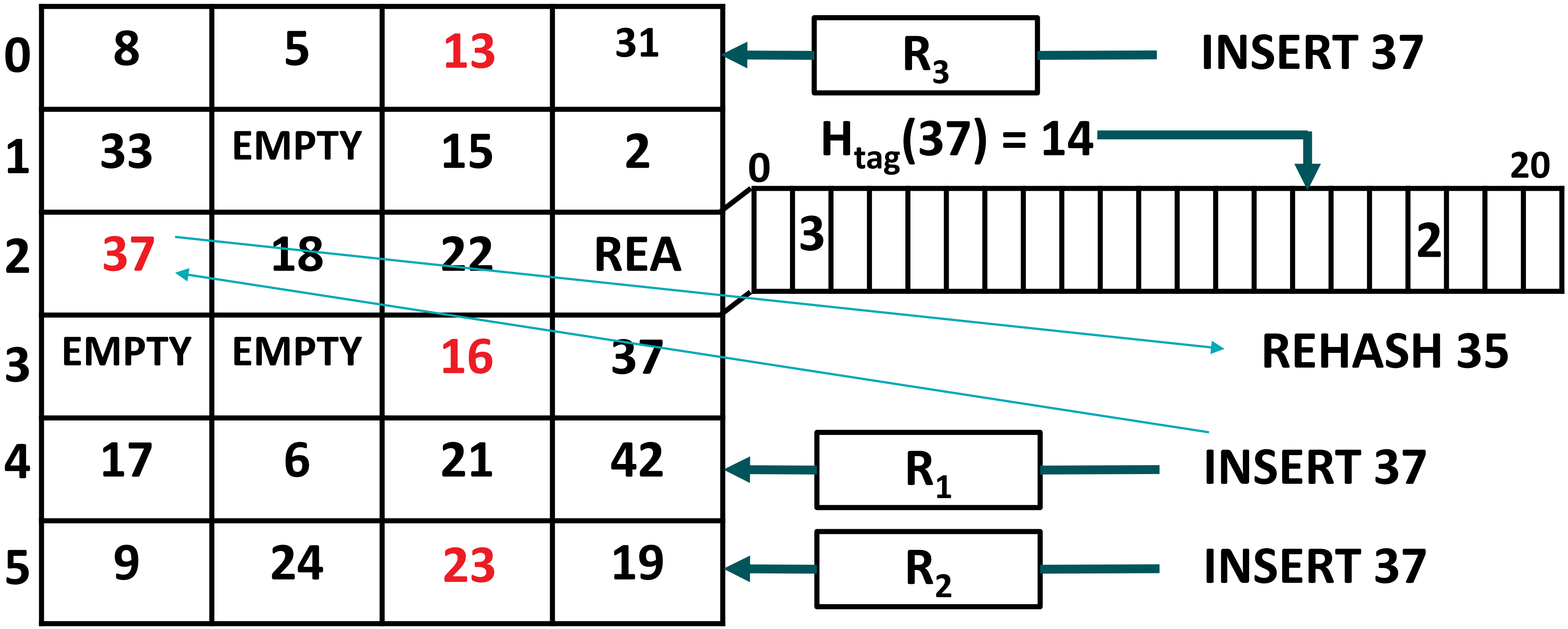
INSERTION WHERE ALL CANDIDATE SECONDARY BUCKETS ARE FULL



- Swap item with another from its primary bucket that can be remapped to a secondary bucket that is not full
- If this fails, then use cuckoo hashing
  - Preferably enforcing as we do that secondary items cannot displace primary items

HORTON TABLES

INSERTION WHERE ALL CANDIDATE SECONDARY BUCKETS ARE FULL



- Swap item with another from its primary bucket that can be remapped to a secondary bucket that is not full
- If this fails, then use cuckoo hashing
  - Preferably enforcing as we do that secondary items cannot displace primary items

END OF BACKUP SLIDES