



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

AMDA

IN IN-MEMORY DATA-INTENSIVE COMPUTING



- Data stores and caches
 - Key-value stores (e.g., Memcached, Redis, MongoDB)
 - Relational databases (e.g., MonetDB, HyPer, IBM DB2 with BLU)
 - Hash indexes
 - Join implementation: hash join and variants
 - Grouping: grouping hash table
 - Dictionary encoding



THE ROLE OF HASH TABLES

IN IN-MEMORY DATA-INTENSIVE COMPUTING



Data stores and caches

- Key-value stores (e.g., Memcached, Redis, MongoDB)
- Relational databases (e.g., MonetDB, HyPer, IBM DB2 with BLU)
 - Hash indexes
 - Join implementation: hash join and variants
 - Grouping: grouping hash table
 - Dictionary encoding

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, MongoDB)
 - Relational databases (e.g., MonetDB, HyPer, IBM DB2 with BLU)
 - Hash indexes
 - Join implementation: hash join and variants
 - Grouping: grouping hash table
 - Dictionary encoding
- Graphics
 - Accelerate computations by computing on hash tables that store sparse images, textures, or surfaces
- General data compression schemes used in common compression utilities

AMD

THE ROLE OF HASH TABLES

IN IN-MEMORY DATA-INTENSIVE COMPUTING



- Data stores and caches
 - Key-value stores (e.g., Memcached, Redis, MongoDB)
 - Relational databases (e.g., MonetDB, HyPer, IBM DB2 with BLU)
 - Hash indexes
 - Join implementation: hash join and variants
 - Grouping: grouping hash table
 - Dictionary encoding
- ▲ 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









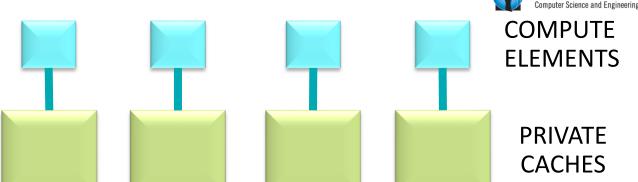




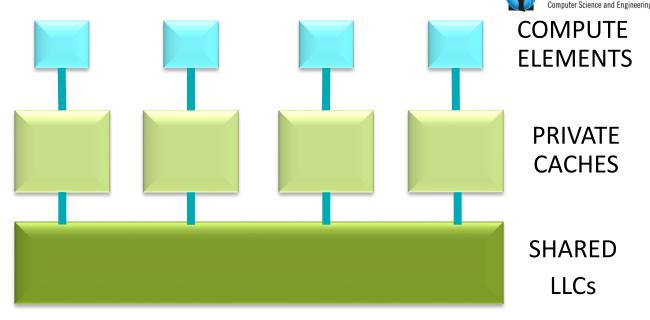




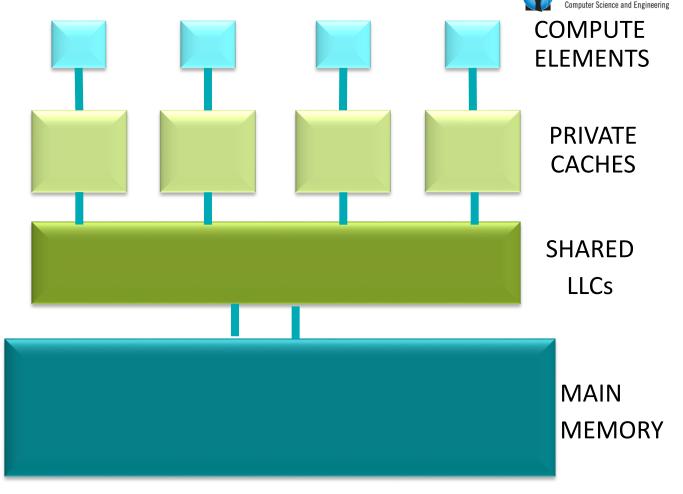






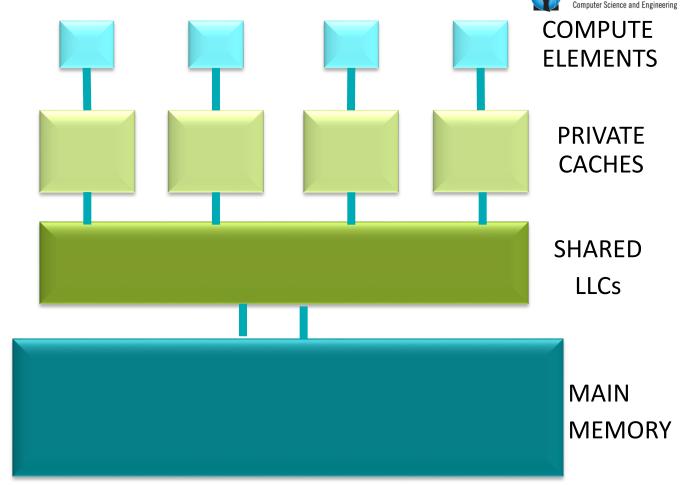




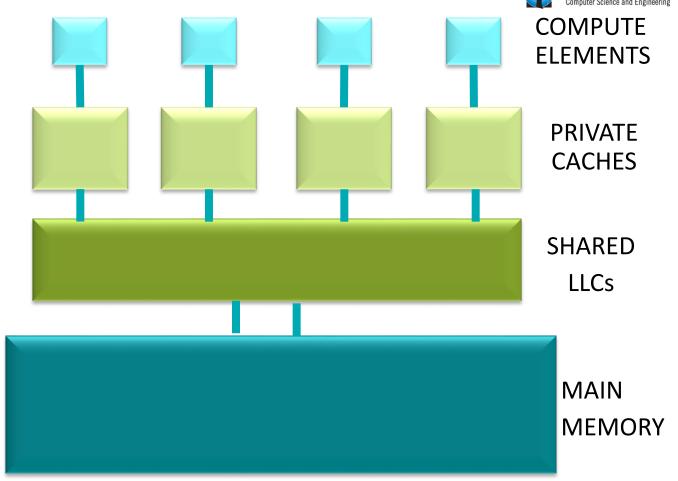




OPTIMIZING MEMORY ACCESSES IN FAST IN-MEMORY HASH TABLES

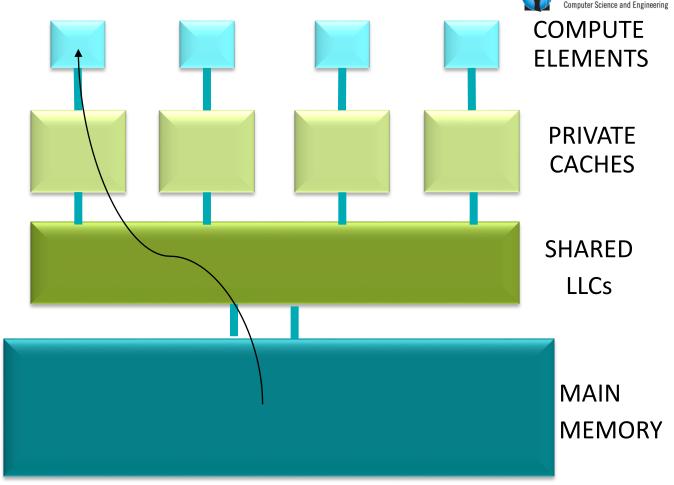






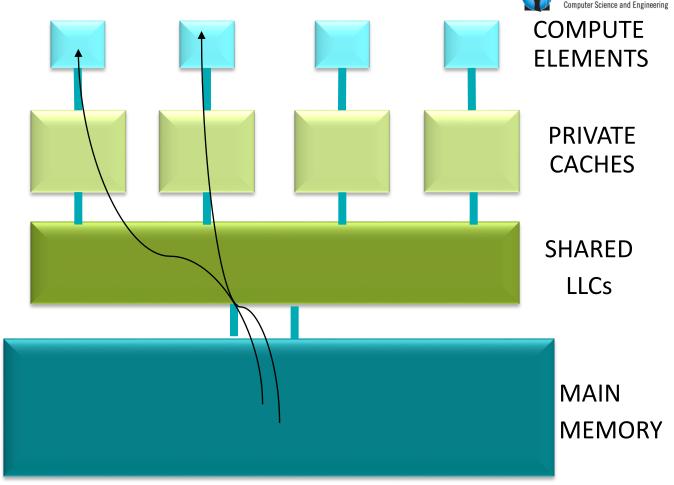
- Hash tables have poor temporal and spatial locality.
- ✓ In-memory hash tables often have hot working sets that are bigger than LLCs.





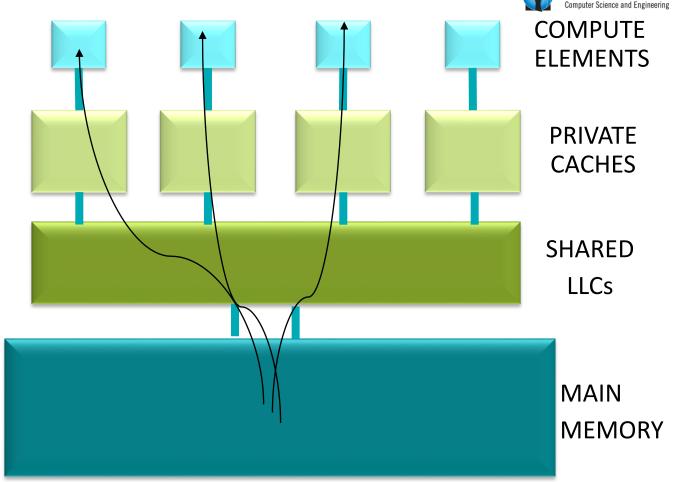
- Hash tables have poor temporal and spatial locality.
- ✓ In-memory hash tables often have hot working sets that are bigger than LLCs.





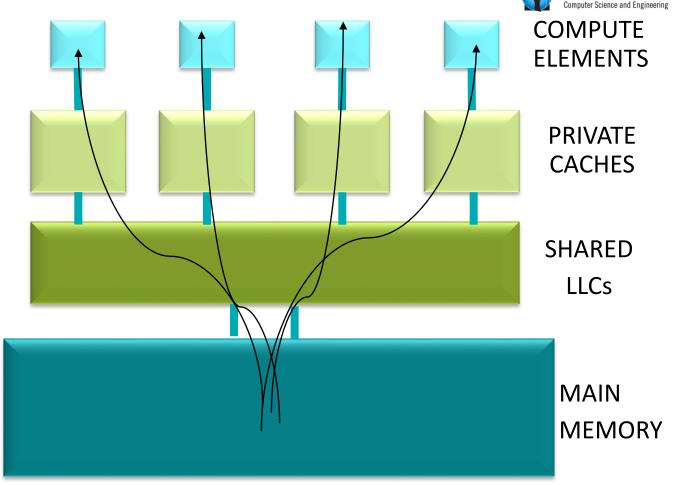
- Hash tables have poor temporal and spatial locality.
- ✓ In-memory hash tables often have hot working sets that are bigger than LLCs.





- ✓ In-memory hash tables often have hot working sets that are bigger than LLCs.





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



MAIN

MEMORY

FOCUS OF THIS TALK

OPTIMIZING MEMORY ACCESSES IN FAST IN-MEMORY HASH TABLES

COMPUTE ELEMENTS

PRIVATE CACHES

Comparatively low bandwidth and high

Comparatively low bandwidth and high latency per memory transaction leads to memoryboundedness



In-memory hash tables often have hot working sets that are bigger than LLCs.

AMD

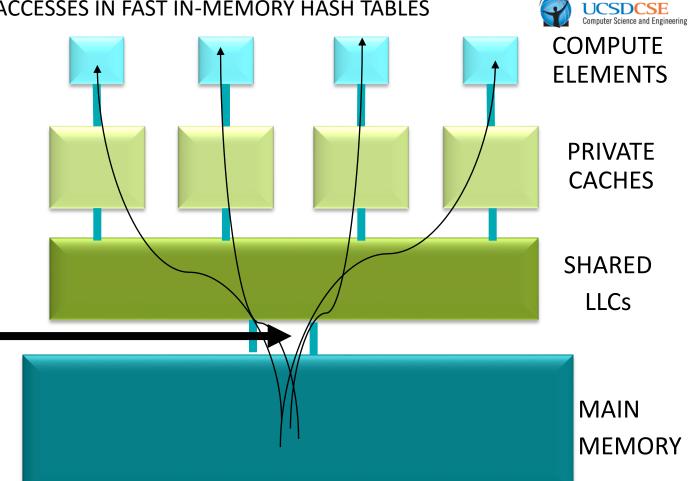
FOCUS OF THIS TALK

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 memoryboundedness



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







а	b	С	EMPTY
d	е	f	g
h	EMPTY	EMPTY	EMPTY
i	j	k	I
m	n	0	р
q	r	S	EMPTY
t	u	V	W

BUCKETIZED CUCKOO HASH TABLES



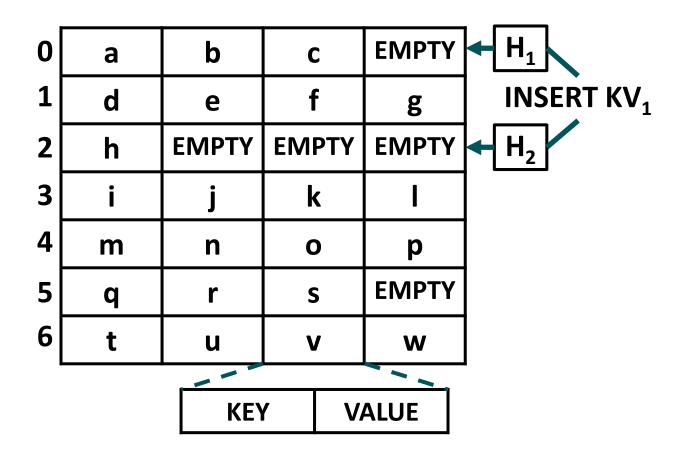


а	b	С	EMPTY
d	e	f	g
h	EMPTY	EMPTY	EMPTY
i	j	k	I
m	n	0	р
q	r	S	EMPTY
t	u	V	W
KEY			ALUE
	INE I		ALUE





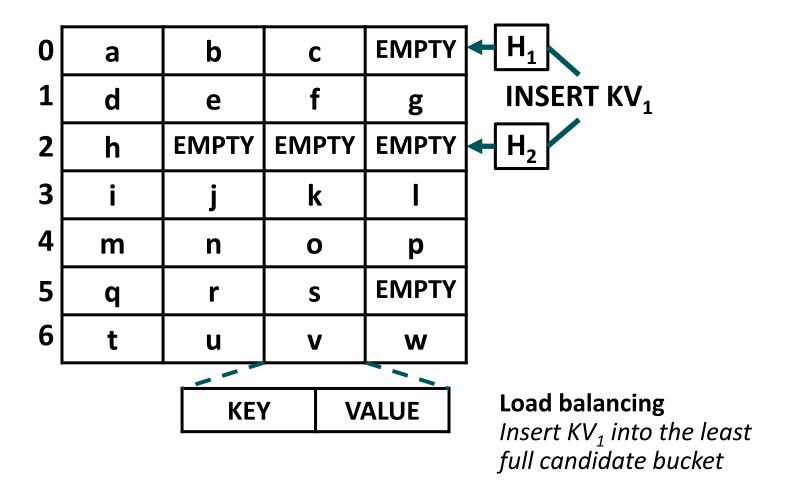








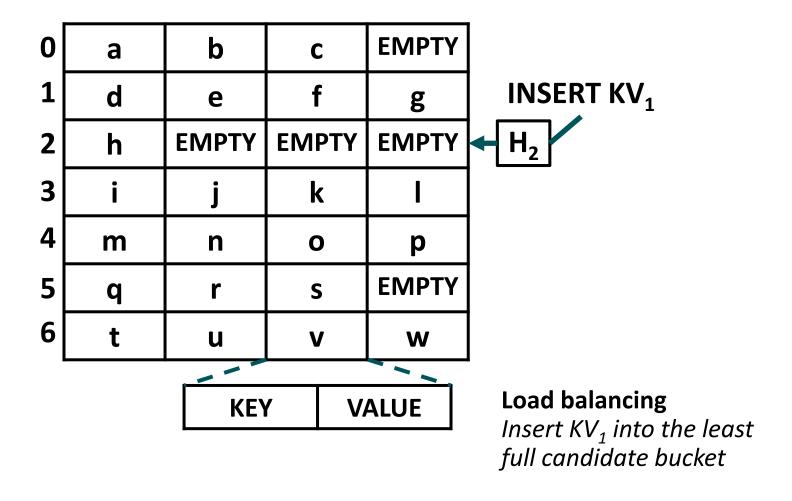








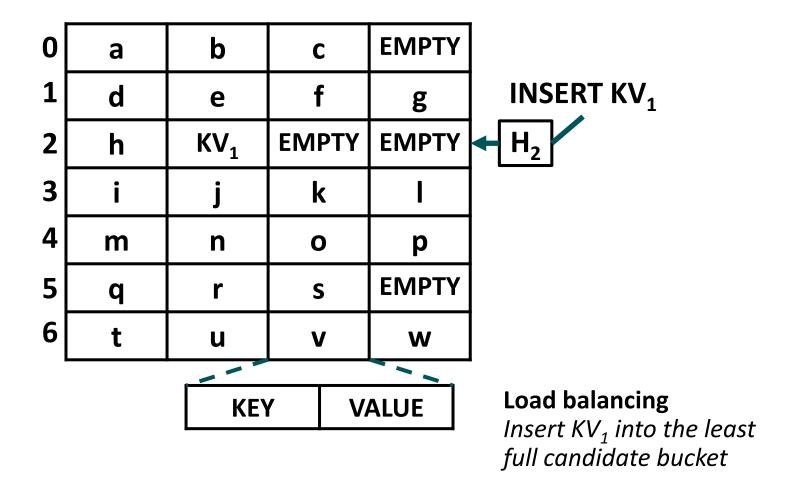








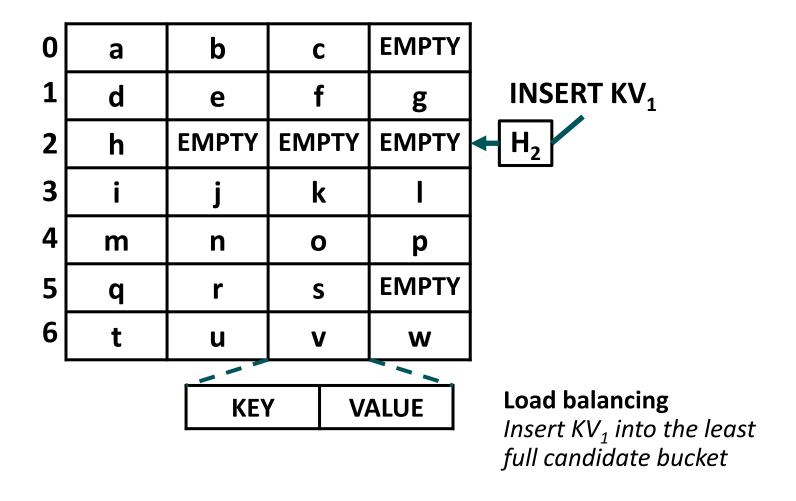








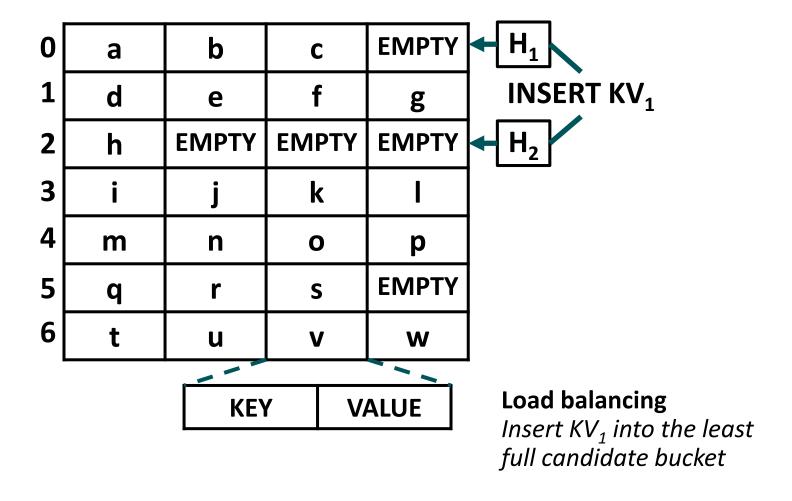








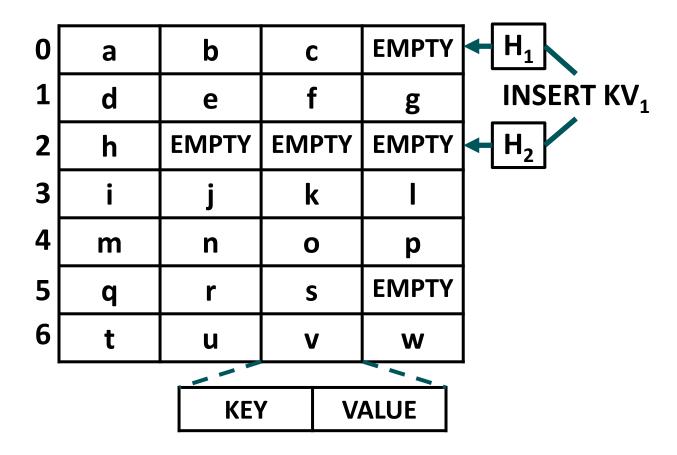








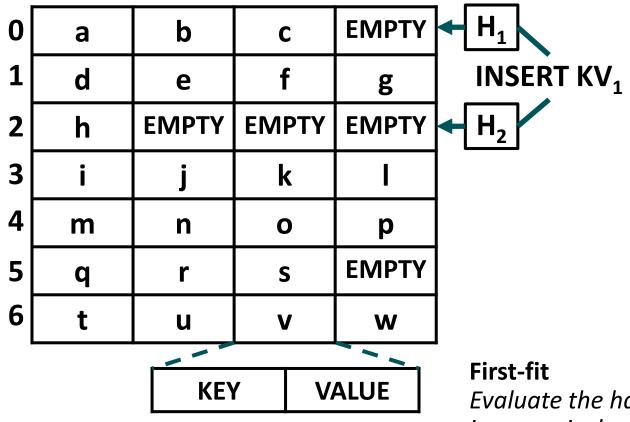












Evaluate the hash functions in numerical order and insert KV_1 into the first candidate bucket with a free slot







0	а	b	С	EMPTY	H_1
1	d	е	f	g	INSERT K
2	h	EMPTY	EMPTY	EMPTY	
3	i	j	k		
4	m	n	0	р	
5	q	r	S	EMPTY	
6	t	u	V	W	
·	KEY VALUE			First-fit <i>Evaluate the</i>	

e hash functions in numerical order and insert KV₁ into the first candidate bucket with a free slot







0	а	b	С	KV ₁	H ₁
1	d	е	f	g	INSERT KV
2	h	EMPTY	EMPTY	EMPTY	
3	i	j	k	I	
4	m	n	0	р	
5	q	r	S	EMPTY	
6	t	u	V	W	
	KEY VALUE			First-fit Evaluate the	

hash functions in numerical order and insert KV₁ into the first candidate bucket with a free slot

BUCKETIZED CUCKOO HASH TABLES



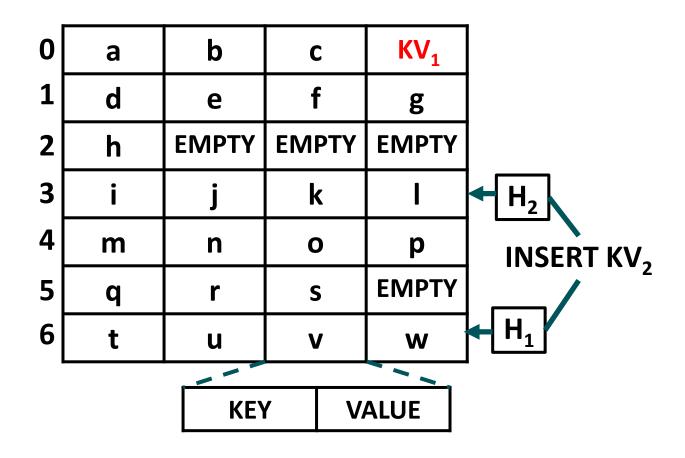


0	а	b	С	KV ₁
1	d	е	f	g
2	h	EMPTY	EMPTY	EMPTY
3	i	j	k	I
4	m	n	0	р
5	q	r	S	EMPTY
6	t	u	V	W
		KEY	ſ V	ALUE





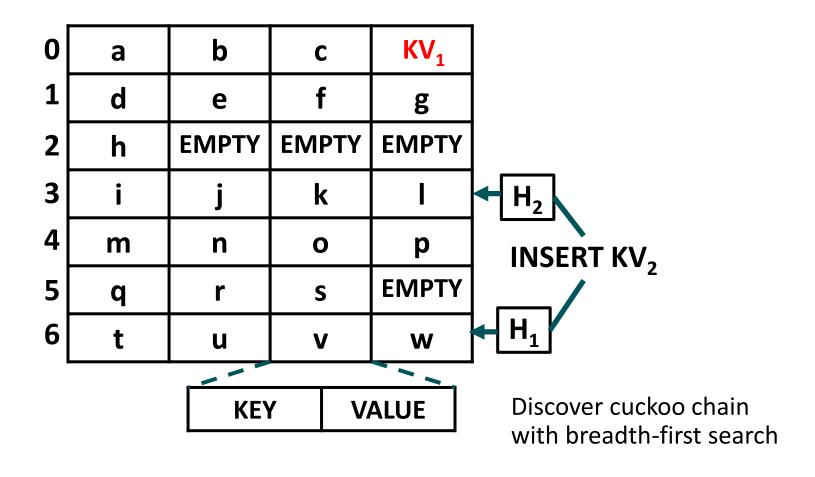








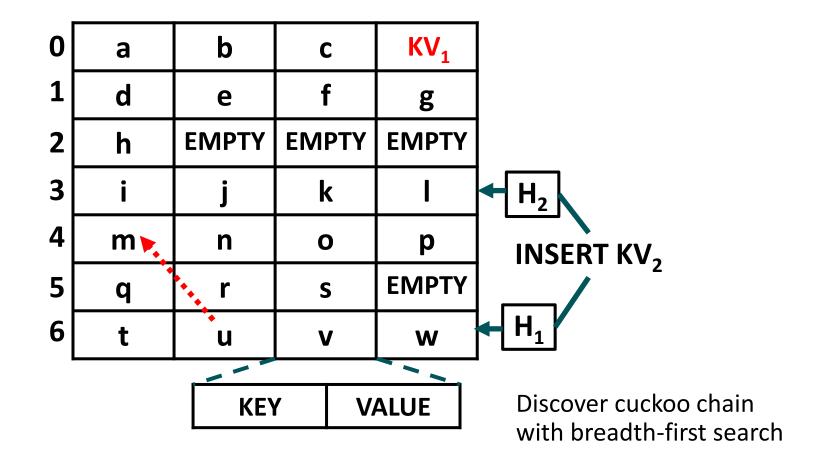








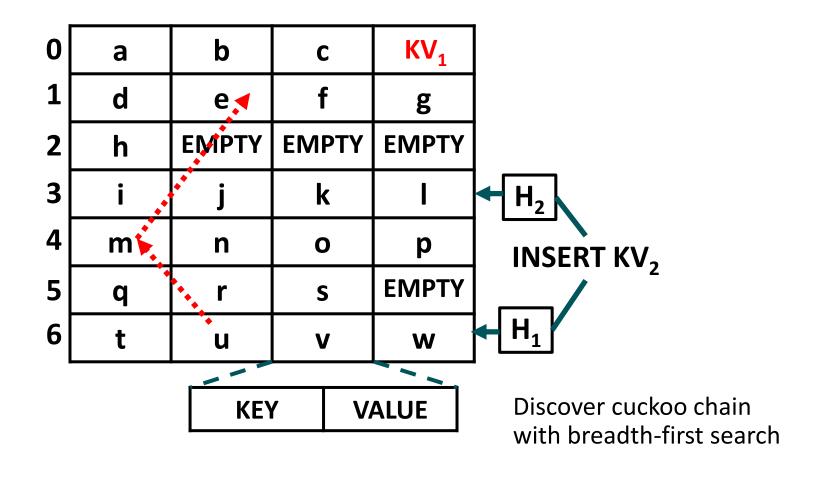








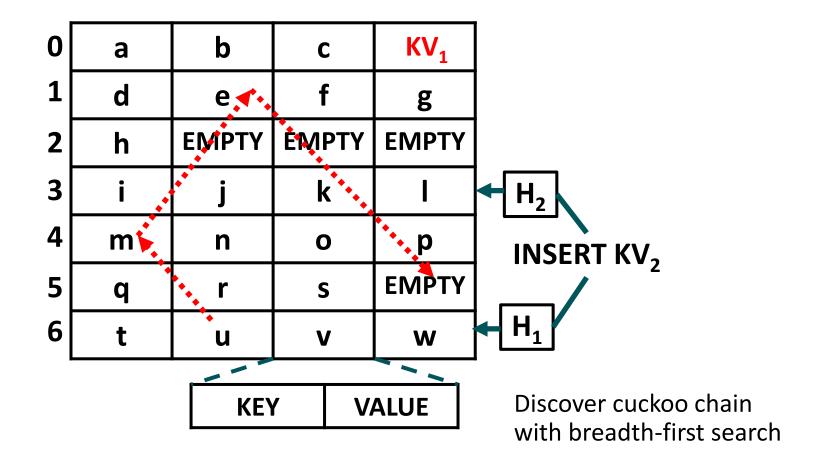








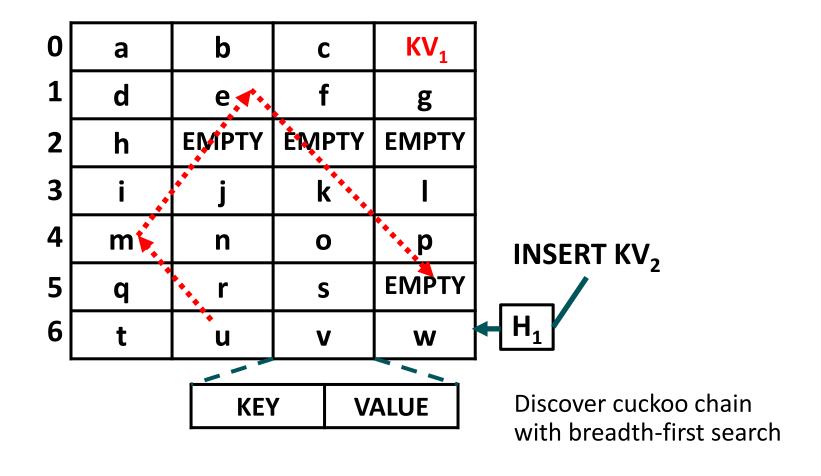








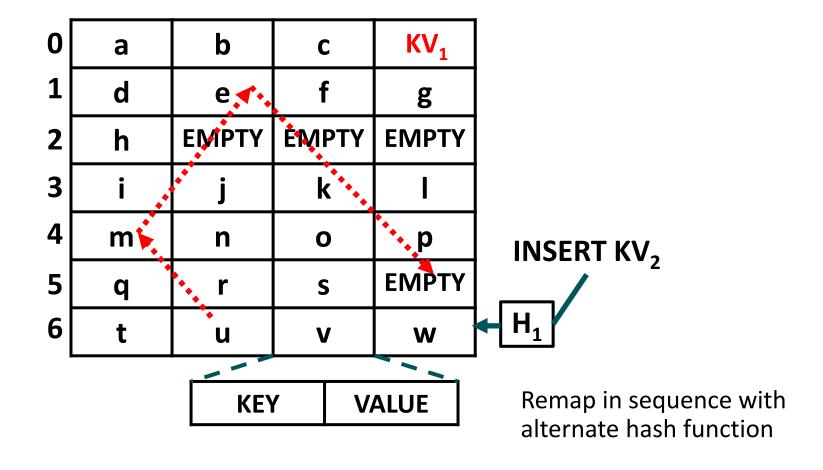








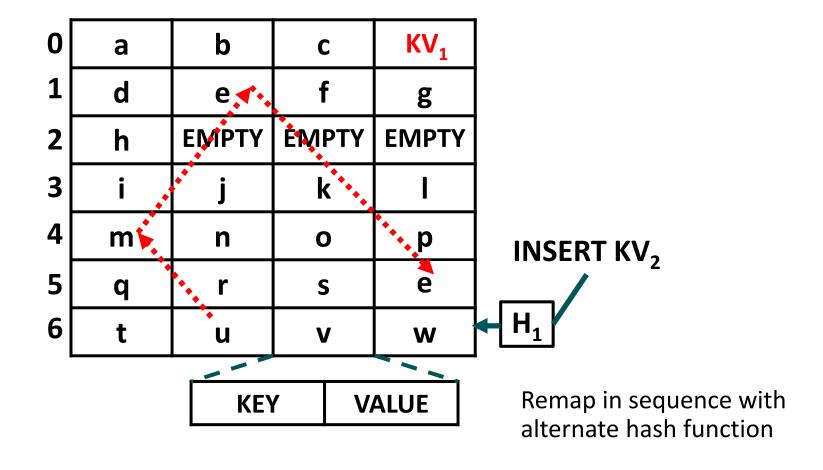








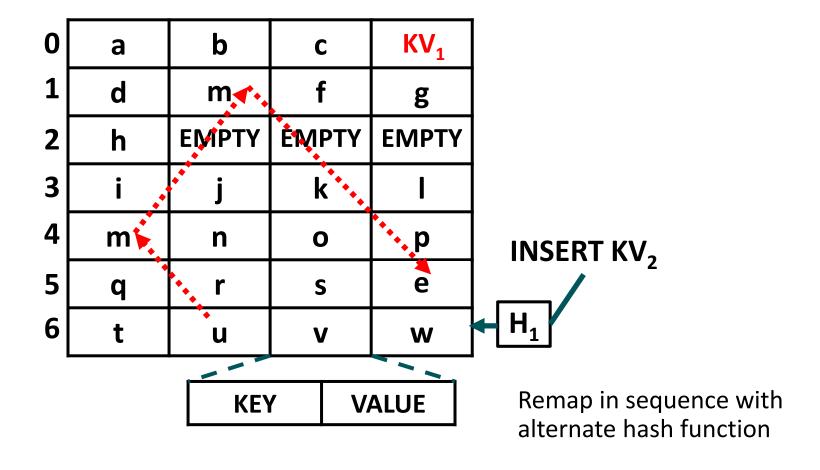








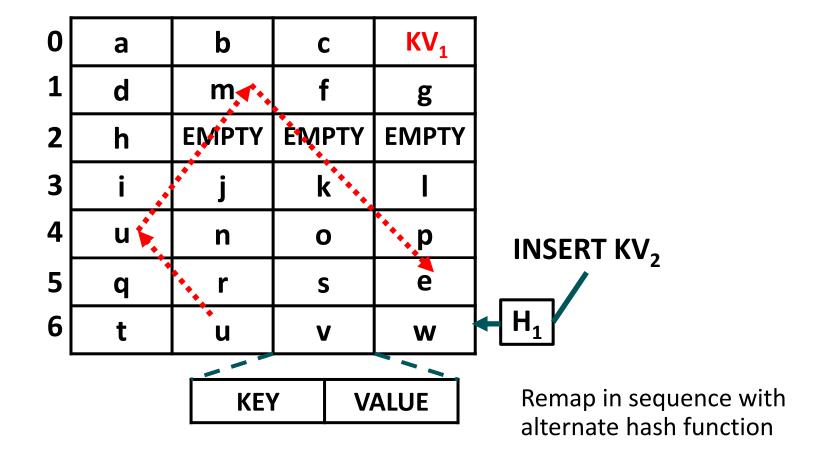








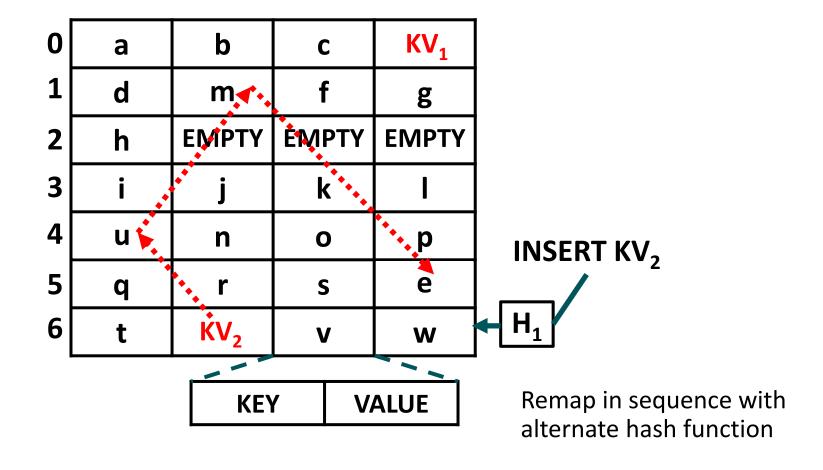






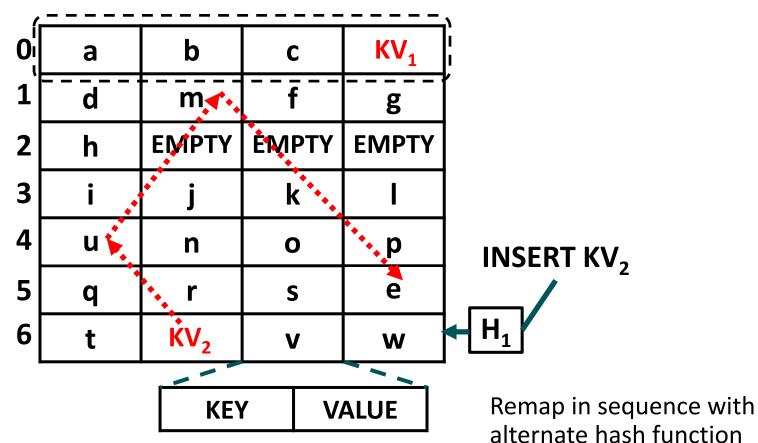










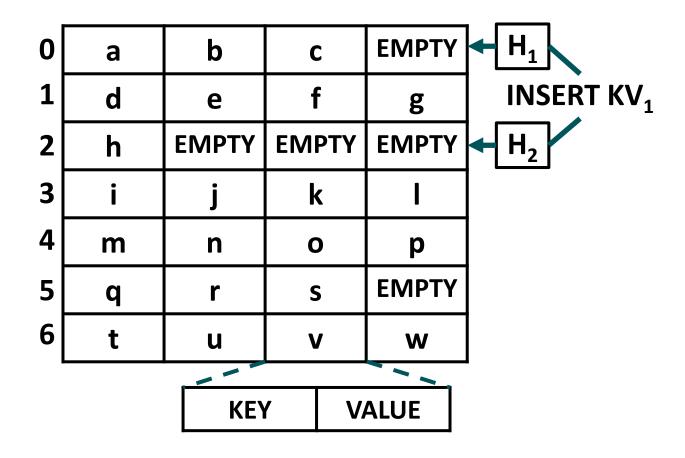


- Each bucket is typically sized to one hardware cache line or less.
- Overwhelmingly, accesses to the bucket's cache line hit in the hardware caches during accesses to consecutive cells.



LOOKUPS AND LOAD BALANCING HEURISTIC







LOOKUPS AND LOAD BALANCING HEURISTIC



				. —
а	b	С	EMPTY	→ H ₁
d	е	f	g	INSERT KV ₁
h	EMPTY	EMPTY	EMPTY	→ H ₂
i	j	k	I	
m	n	0	р	
q	r	S	EMPTY	
t	u	V	w	
				_
	KEY	Y V	ALUE	
	d h i m	d e h EMPTY i j m n q r t u	d e f h EMPTY EMPTY i j k m n o q r s t u v	d e f g h EMPTY EMPTY i j k l m n o p q r s EMPTY t u v w

Expected Positive Lookup Cost Per Item in Buckets: (Fraction of Items Hashed by H_1) + 2 * (Fraction of Items Hashed by H_2)



LOOKUPS AND LOAD BALANCING HEURISTIC



_					
0	а	b	С	EMPTY	$+$ H_1
1	d	е	f	g	INSERT KV ₁
2	h	EMPTY	EMPTY	EMPTY	→ H₂
3	i	j	k	I	
4	m	n	0	р	
5	q	r	S	EMPTY	
6	t	u	V	W	
-					_
		KEY	Y V	ALUE	

- Expected Positive Lookup Cost Per Item in Buckets: (Fraction of Items Hashed by H_1) + 2 * (Fraction of Items Hashed by H_2)
- Expected Negative Lookup Cost per Item in Buckets:



LOOKUPS AND LOAD BALANCING HEURISTIC



_				_	
0	а	b	С	EMPTY	H_1
1	d	е	f	g	INSERT KV ₁
2	h	EMPTY	EMPTY	EMPTY	H₂
3	i	j	k	I	
4	m	n	0	р	
5	q	r	S	EMPTY	
6	t	u	V	w	
					Load balancing
		KEY	/ V	ALUE	Insert KV ₁ into the least
D:	eti in landi	Cook Dow	Itaas in De		full candidate bucket

- Expected Positive Lookup Cost Per Item in Buckets: (Fraction of Items Hashed by H_1) + 2 * (Fraction of Items Hashed by H_2)
- Expected Negative Lookup Cost per Item in Buckets:



LOOKUPS AND LOAD BALANCING HEURISTIC



		_	_	_						
0	а	b	С	EMPTY						
1	d	е	f	g	INSERT KV ₁					
2	h	EMPTY	EMPTY	EMPTY	H₂					
3	i	j	k	ı						
4	m	n	0	р						
5	q	r	S	EMPTY						
6	t	u	v	w						
					Load balancing					
		KEY	V.	ALUE	Insert KV ₁ into the least					
Dasi	full candidate bucket									

- Expected Positive Lookup Cost Per Item in Buckets: (Fraction of Items Hashed by H_1) + 2 * (Fraction of Items Hashed by H_2)
- Expected Negative Lookup Cost per Item in Buckets:



LOOKUPS AND LOAD BALANCING HEURISTIC



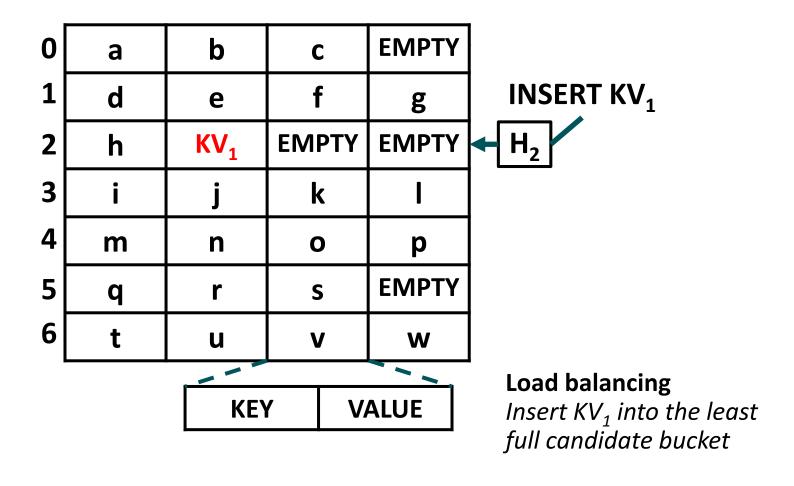
					_
0	а	b	С	EMPTY	
1	d	е	f	g	INSERT KV ₁
2	h	KV ₁	EMPTY	EMPTY	H ₂
3	i	j	k	ı	
4	m	n	0	р	
5	q	r	S	EMPTY	
6	t	u	v	w	
•					Load balancing
		KE	Y V	ALUE	Insert KV ₁ into the least
					full candidate bucket

- Expected Positive Lookup Cost Per Item in Buckets: (Fraction of Items Hashed by H_1) + 2 * (Fraction of Items Hashed by H_2)
- Expected Negative Lookup Cost per Item in Buckets:



LOOKUPS AND LOAD BALANCING HEURISTIC







LOOKUPS AND LOAD BALANCING HEURISTIC



			_		_
0	а	b	С	EMPTY	
1	d	е	f	g	INSERT KV ₁
2	h	KV ₁	EMPTY	EMPTY	H₂
3	i	j	k	ı	
4	m	n	0	р	
5	q	r	S	EMPTY	
6	t	u	V	w	
!					Load balancing
KEY VALUE					Insert KV ₁ into the least
Doci	tivo Looku	n Cost Dor	Itam in Du		full candidate bucket

■ Expected Positive Lookup Cost Per Item in Buckets: $1.5 = (0.5 \text{ Hashed by H}_1) + 2 * (0.5 \text{ Hashed by H}_2)$



LOOKUPS AND LOAD BALANCING HEURISTIC

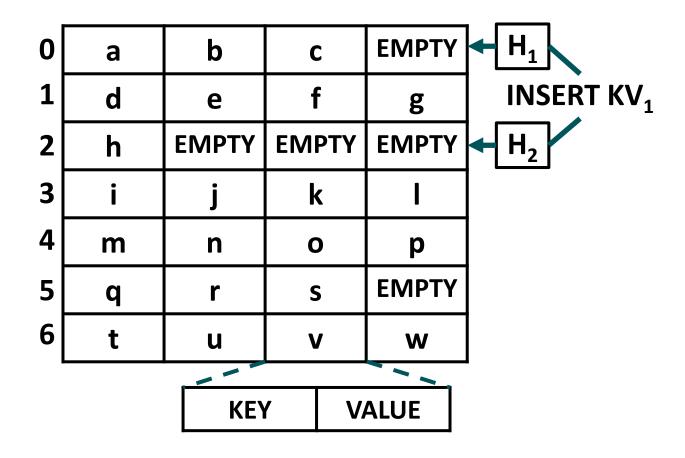


_					_
0	а	b	С	EMPTY	
1	d	е	f	g	INSERT KV ₁
2	h	KV ₁	EMPTY	EMPTY	H₂
3	i	j	k	1	
4	m	n	0	р	
5	q	r	S	EMPTY	
6	t	u	V	w	
					Load balancing
	KEY VALUE				Insert KV ₁ into the least
					full candidate bucket

- Expected Positive Lookup Cost Per Item in Buckets: $1.5 = (0.5 \text{ Hashed by H}_1) + 2 * (0.5 \text{ Hashed by H}_2)$
- ▲ Expected Negative Lookup Cost per Item in Buckets:2 (also worst-case)

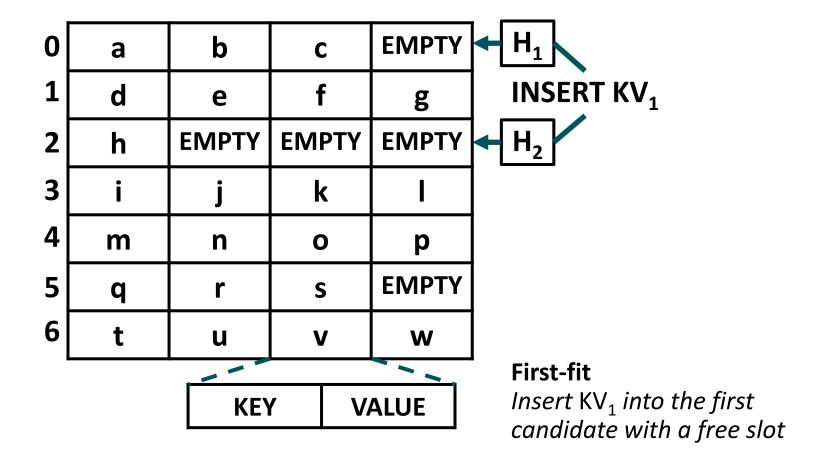
















0	а	b	С	EMPTY	→ H ₁
1	d	е	f	g	INSERT KV ₁
2	h	EMPTY	EMPTY	EMPTY	
3	i	j	k	1	
4	m	n	0	р	
5	q	r	S	EMPTY	
6	t	u	V	W	
•					First-fit
		KEY	Y V	ALUE	Insert KV ₁ into the first candidate with a free slot
					canalate with a free slot





0	а	b	С	KV ₁	H_1
1	d	е	f	g	INSERT KV ₁
2	h	EMPTY	EMPTY	EMPTY	
3	i	j	k	I	
4	m	n	0	р	
5	q	r	S	EMPTY	
6	t	u	V	w	
•					First-fit
		KEY	/ V	ALUE	Insert KV ₁ into the first
					candidate with a free slot



LOOKUPS AND FIRST-FIT INSERTION HEURISTIC



0	а	b	С	KV ₁	H_1
1	d	е	f	g	INSERT KV ₁
2	h	EMPTY	EMPTY	EMPTY	
3	i	j	k	I	
4	m	n	0	р	
5	q	r	S	EMPTY	
6	t	u	V	W	
•					First-fit
		KEY	/ V	ALUE	Insert KV ₁ into the firs candidate with a free
					Sallalaace With a fice

Expected Positive Lookup Cost Per Item in Buckets:

1 to 1.3ish depending on the table load factor and the slots per bucket



LOOKUPS AND FIRST-FIT INSERTION HEURISTIC



0	а	b	С	KV ₁	H_1
1	d	е	f	g	INSERT KV ₁
2	h	EMPTY	EMPTY	EMPTY	
3	i	j	k	I	
4	m	n	0	р	
5	q	r	S	EMPTY	
6	t	u	V	W	
•					First-fit
		KEY	/ V	ALUE	Insert KV ₁ into the firs candidate with a free
					Sallalaace With a fice

- 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:

AMD





_		_		
0	а	b	С	KV ₁
1	d	m	m f	
2	h	EMPTY	EMPTY	EMPTY
3	i	j	k	I
4	u	n	0	р
5	q	r	S	е
6	t	KV ₂	٧	W
•				
		KEY	Y V	ALUE



BENEFITS OF FIRST-FIT



0	а	b	С	KV ₁
1	d	m	f	g
2	h	EMPTY	EMPTY	EMPTY
3	i	j	k	I
4	u	n	0	р
5	q	r	S	е
6	t	KV ₂	V	W
•				
		KEY	/ V	ALUE

LOOKUP KV₁

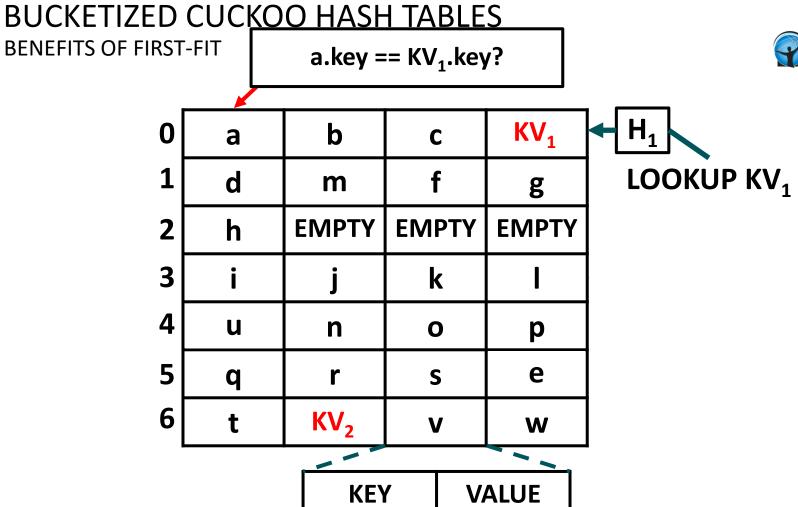


BENEFITS OF FIRST-FIT



_		_	-		
0	а	b	С	KV ₁	→ H ₁
1	d	m	f	g	LOOKUP KV ₁
2	h	EMPTY	EMPT	Y EMPTY	
3	i	j	k	ı	
4	u	n	0	р	
5	q	r	S	е	
6	t	KV ₂	V	w	
•					_
KEY			1	VALUE	







BENEFITS OF FIRST-FIT

a.key == KV₁.key?



			_		_
0	а	b	С	KV ₁	→ H ₁
1	d	m	f	g	LOOKUP KV ₁
2	h	EMPTY	EMPTY	EMPTY	
3	i	j	k	ı	
4	u	n	0	р	
5	q	r	S	е	
6	t	KV ₂	V	w	
•					_
		KEY	Y V	ALUE	



BENEFITS OF FIRST-FIT

b.key == KV₁.key?



				- —
а	b	С	KV ₁	H ₁
d	m	f	g	LOOKUP KV ₁
h	EMPTY	EMPT	YEMPTY	
i	j	k	ı	
u	n	0	р	
q	r	S	е	
t	KV ₂	V	w	
				_
	KEY	1	VALUE	
	d h i u	d m h EMPTY i j u n q r t KV2	d m f h EMPTY EMPTY i j k u n o q r s t KV ₂ v	d m f g h EMPTY EMPTY EMPTY i j k l u n o p q r s e t KV ₂ v w



BENEFITS OF FIRST-FIT

b.key == KV₁.key?



		_	_			
0	а	b	С		KV ₁	→ H ₁
1	d	m	f		g	LOOKUP KV ₁
2	h	EMPTY	ЕМРТ	Υ	EMPTY	
3	i	j	k		I	
4	u	n	О		р	
5	q	r	S		е	
6	t	KV ₂	V		W	
•						
		KEY	1	V	ALUE	



BENEFITS OF FIRST-FIT

c .key == KV₁.key?



_					·	
0	а	b	С		KV ₁	H₁
1	d	m	f		g	LOOKUP KV ₁
2	h	EMPTY	EMP	TY	EMPTY	
3	i	j	k		1	
4	u	n	0		р	
5	q	r	S		e	
6	t	KV ₂	V		W	
						_
		KEY	1	V	ALUE	



BENEFITS OF FIRST-FIT

c .key == KV_1 .key?



		_	_			
0	а	b	С		KV ₁	→ H ₁
1	d	m	f		g	LOOKUP KV ₁
2	h	EMPTY	ЕМРТ	Υ	EMPTY	
3	i	j	k		I	
4	u	n	О		р	
5	q	r	S		е	
6	t	KV ₂	V		W	
•						
		KEY	1	V	ALUE	



BENEFITS OF FIRST-FIT

 KV_1 .key == KV_1 .key?



0	а	b	С		KV ₁	→ H ₁
1	d	m	f		g	LOOKUP KV ₁
2	h	EMPTY	EMP	ΓΥ	EMPTY	
3	i	j	k		Ι	
4	u	n	0		р	
5	q	r	S		е	
6	t	KV ₂	V		W	
		KEY	1	V	ALUE	



BENEFITS OF FIRST-FIT

 KV_1 .key == KV_1 .key?



_					_
0	а	b	С	KV ₁	→ H ₁
1	d	m	f	g	LOOKUP KV ₁
2	h	EMPTY	EMPTY	EMPTY	
3	i	j	k	I	
4	u	n	0	р	LOOKUP KV ₂
5	q	r	S	е	
6	t	KV ₂	V	W	H_1
					_
		KEY			



BUCKETIZED CUCKOO HASH TABLES BENEFITS OF FIRST-FIT KV_1 .key == KV_1 .key? KV₁ b 0 a C LOOKUP KV₁ d m g **EMPTY EMPTY EMPTY** h 3 k 4 u n 0 p LOOKUP KV₂ 5 e S q 6 KV₂ W **KEY VALUE**

▲ Positive Lookups:

 First-fit gets us most of the way to 1.0 on positive lookups because most elements are hashed with H₁



BUCKETIZED CUCKOO HASH TABLES BENEFITS OF FIRST-FIT KV_1 .key == KV_1 .key? KV_1 b 0 a C LOOKUP KV₁ d m g **EMPTY EMPTY EMPTY** h 3 k 4 u n 0 p LOOKUP KV₂ 5 e S q 6 KV₂ W

▲ Positive Lookups:

 First-fit gets us most of the way to 1.0 on positive lookups because most elements are hashed with H₁

VALUE

KEY

■ But...

AMD



LIMITATIONS OF FIRST-FIT

0	а	b	С	KV ₁
1	d	m	f	g
2	h	EMPTY	EMPTY	EMPTY
3	i	j	k	I
4	u	n	0	р
5	q	r	S	е
6	t	KV ₂	V	W
_				
		KEY	/ V	ALUE



LIMITATIONS OF FIRST-FIT



0	а	b	С	KV ₁	
1	d	m	f	g	
2	h	EMPTY	EMPTY	EMPTY	
3	i	j	k	I	
4	u	n	0	р	
5	q	r	S	е	
6	t	KV ₂	V	W	
		KEY	/ V	ALUE	

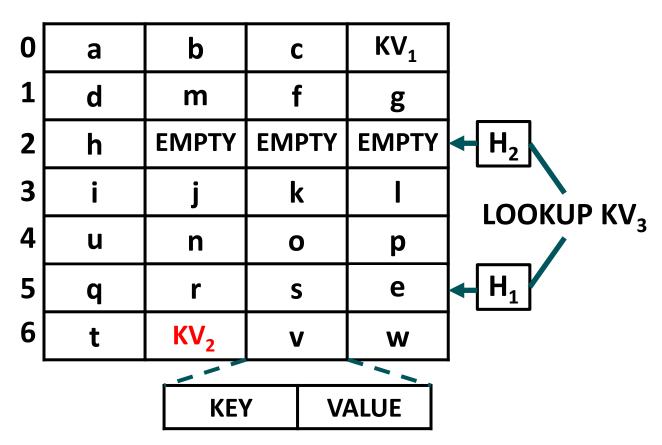
- Expected Negative Lookup Cost per Item in Buckets:
 - First-fit doesn't address the comparatively expensive negative lookup cost. We still need to check all candidate buckets.



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.



AMD



- 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.



AMDA



- 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.



AMDA



- 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)



AMDA



- 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)

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.



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



Like first-fit, they strongly bias inserts by using a primary hash function called H_{primary}



HORTON TABLES PRIMARY INSERTIONS AND LOOKUPS

UCSDCSE Computer Science and Engineering
Computer Science and Engineering

_				
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.



			_	_	_			
0	8	5	EMPTY	EMPTY	—	H _{prin}	nary	INSERT :
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

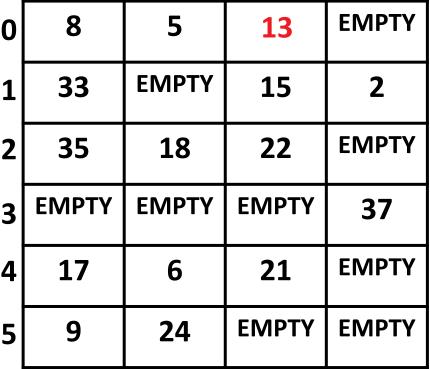


					=	
0	8	5	13	EMPTY	H _{primary} INSER	Г1
1	33	EMPTY	15	2		
2	35	18	22	EMPTY		
3	EMPTY	EMPTY	EMPTY	37		
4	17	6	21	EMPTY		
5	9	24	EMPTY	EMPTY		

- ▲ 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.



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



		_			Com
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	
2	35	18	22	EMPTY	H _{primary} INSERT 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



PRI	MARY	NSERTIONS	S AND LOO	KUPS
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

INSERT 16 primary

- 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

UCSDCSE Computer Science and Engineering
Computer Science and Engineering

_				
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





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





_					Computer Science
0	8	5	13	EMPTY	H _{primary} LOOKUP 13
1	33	EMPTY	15	2	
2	35	18	22	16	H _{primary} LOOKUP 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



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

UCSDCSE Computer Science and Engineerin	g
--	---

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



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



_					Comput
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	
2	35	18	22	16	H _{primary} INSERT 23
3	EMPTY	EMPTY	EMPTY	37	
4	17	6	21	EMPTY	
5	9	24	EMPTY	EMPTY	

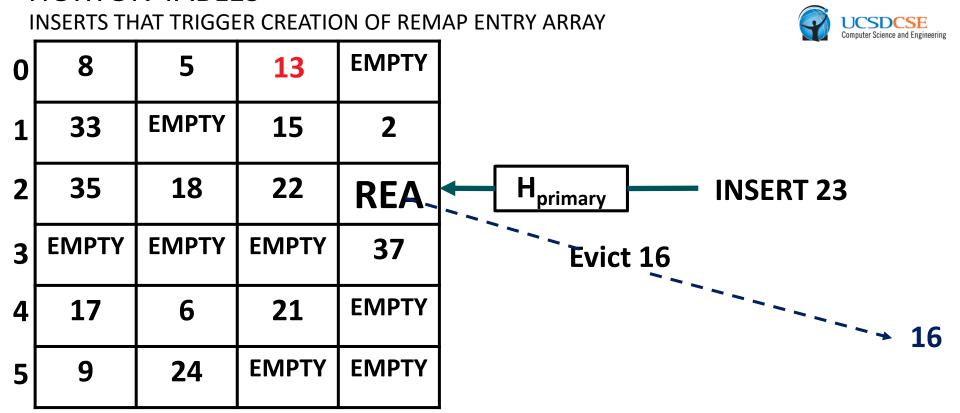


INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY **EMPTY** 8 5 **13** 0 **EMPTY 15** 33 2 $H_{primary}$ **35** 18 22 16_ **INSERT 23 EMPTY EMPTY EMPTY 37** Evict 16 **EMPTY 17** 6 21 **EMPTY EMPTY** 9 24 5



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY **EMPTY** 8 5 **13** 0 **EMPTY 15** 33 $H_{primary}$ **35** 18 22 **INSERT 23** REA **EMPTY EMPTY EMPTY 37** Evict 16 **EMPTY 17** 6 21 **EMPTY EMPTY** 24 9 5







INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY **EMPTY** 8 5 **13** 0 **EMPTY 15** 33 $H_{primary}$ **35** 18 22 **INSERT 23 REA EMPTY EMPTY EMPTY 37 EMPTY 17** 6 21 4 **16 EMPTY EMPTY** 9 24 5



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



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

UCSDCSE Computer Science and Engineering
--

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



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

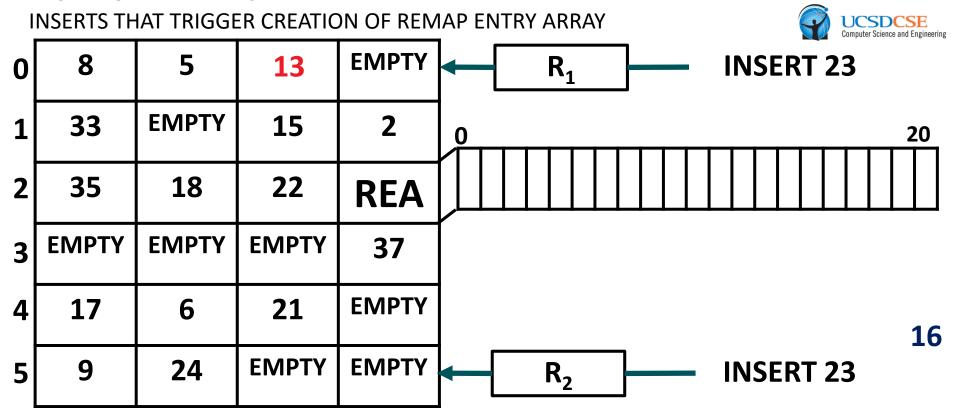
neerin
neer

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

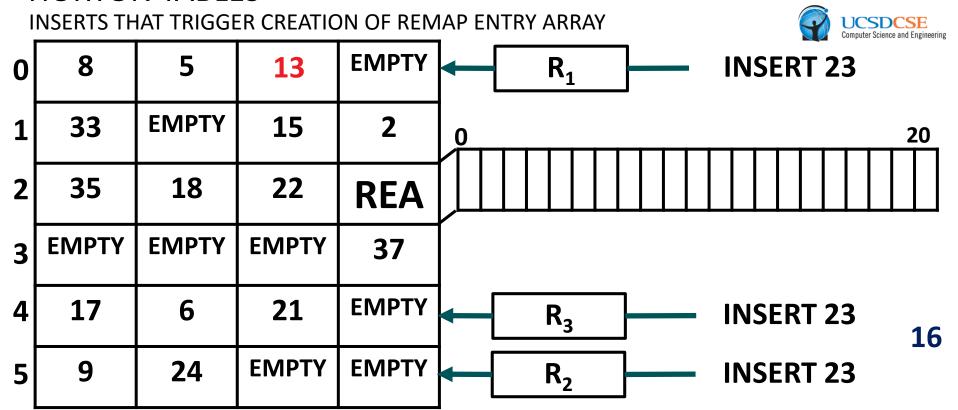


HORTON TABLES INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY **EMPTY** 8 5 **13** R_1 **INSERT 23** 0 **EMPTY 15** 33 2 20 **35** 18 22 **REA EMPTY EMPTY EMPTY 37 EMPTY 17** 6 **21** 4 **16 EMPTY EMPTY** 24 9 5

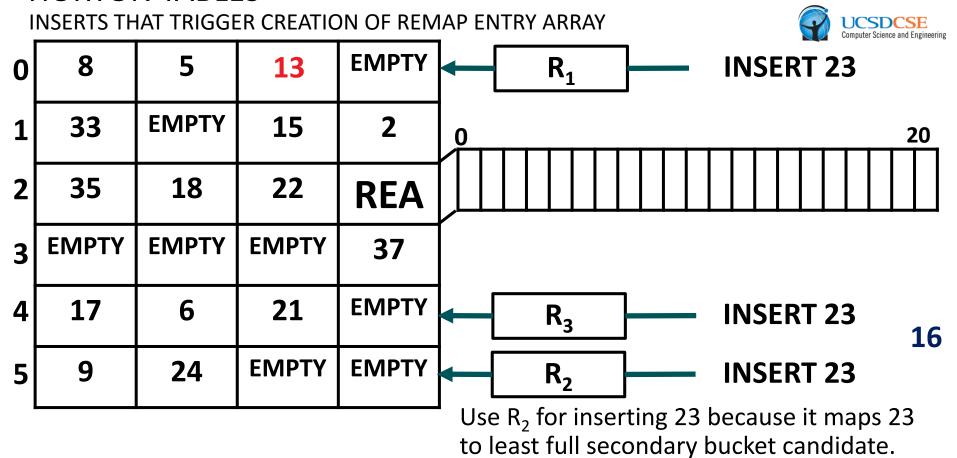




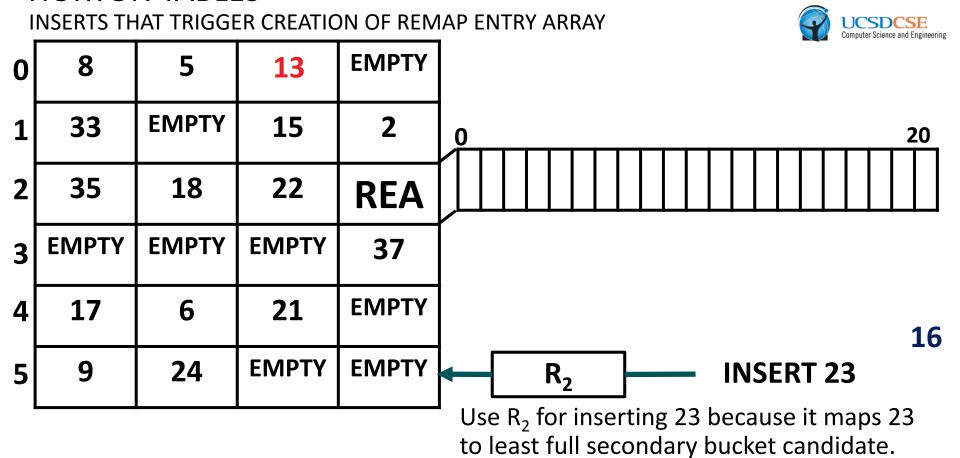




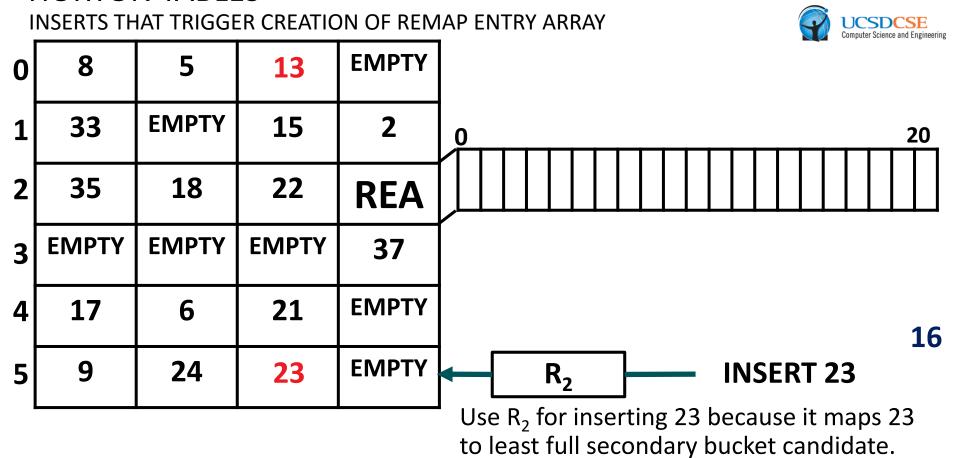




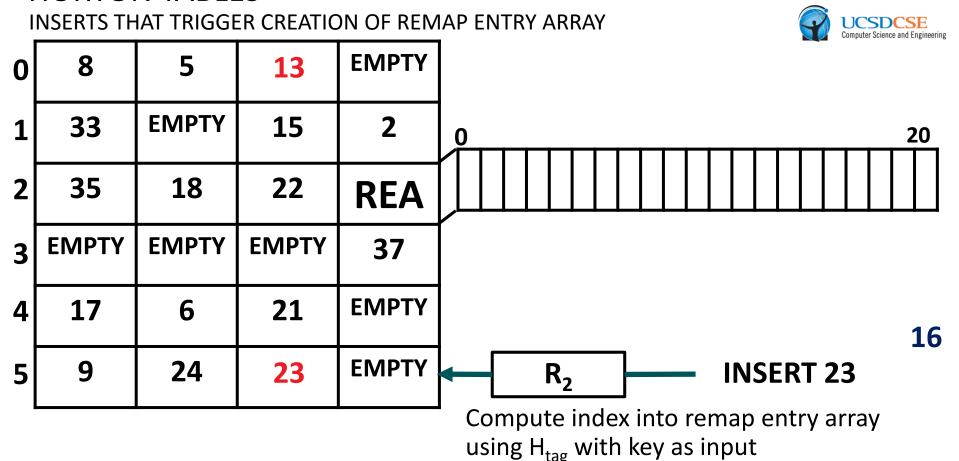














INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



					Computer Science	e a
0	8	5	13	EMPTY		
1	33	EMPTY	15	2	$_{0}$ $H_{tag}(23) = 17$	
2	35	18	22	REA		
3	EMPTY	EMPTY	EMPTY	37		
4	17	6	21	EMPTY		
5	9	24	23	EMPTY	R ₂ INSERT 23	
					Compute index into remap entry array using H _{tag} with key as input	



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



_		_	_	_	Computer Science	and En
0	8	5	13	EMPTY		
1	33	EMPTY	15	2	$_{0}$ $H_{tag}(23) = 17$	2
2	35	18	22	REA		
3	EMPTY	EMPTY	EMPTY	37		
4	17	6	21	EMPTY		
5	9	24	23	EMPTY	R ₂ INSERT 23	•
				L	Store 2 at index $H_{tag}(23)=17$ to indicate that R_2 was used to remap 23	



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



_					Computer Science	and Engir
0	8	5	13	EMPTY		
1	33	EMPTY	15	2	$_{0}$ $H_{tag}(23) = 17$	20
2	35	18	22	REA		
3	EMPTY	EMPTY	EMPTY	37		
4	17	6	21	EMPTY		1
5	9	24	23	EMPTY	R ₂ INSERT 23	_
!					Store 2 at index $H_{tag}(23)=17$ to indicate that R_2 was used to remap 23	



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



		_	_	_	Configure Science	allu Eligi
0	8	5	13	EMPTY		
1	33	EMPTY	15	2	$_{0}$ $H_{tag}(23) = 17$	20
2	35	18	22	REA		
3	EMPTY	EMPTY	EMPTY	37		
4	17	6	21	EMPTY		1
5	9	24	23	EMPTY	R ₂ INSERT 23	_
ı					Store 2 at index $H_{tag}(23)=17$ to indicate that R_2 was used to remap 23	



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

_			_	_	Computer Science and Engineer
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	20
2	35	18	22	REA	2
3	EMPTY	EMPTY	EMPTY	37	
4	17	6	21	EMPTY	16
5	9	24	23	EMPTY	10



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



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

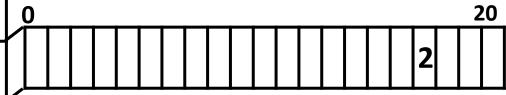
16 now also needs to be remapped to a secondary bucket.



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

UCSDCSE
Computer Science and Engineerin

0	8	5	13	EMPTY	
1	33	EMPTY	15	2	0_
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.



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



					obilipator obtained and Engineering
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	0 20
2	35	18	22	REA	
3	EMPTY	EMPTY	EMPTY	37	
4	17	6	21	EMPTY	16
5	9	24	23	EMPTY	R ₁ INSERT 16
					16 now also needs to be remarked to a

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



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



0	8	5	13	EMPTY	
1	33	EMPTY	15	2	0 20
2	35	18	22	REA	
3	EMPTY	EMPTY	EMPTY	37	
4	17	6	21	EMPTY	R ₂ INSERT 16
5	9	24	23	EMPTY	R ₁ INSERT 16
					16 now also poods to be remarked to a

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



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

					Computer Science and Engineering
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	0 20
2	35	18	22	REA	
3	EMPTY	EMPTY	EMPTY	37	R ₃ INSERT 16
4	17	6	21	EMPTY	R_2 INSERT 16

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

INSERT 16

▲ For buckets that overflow, we remap surplus elements using one of many secondary hash functions and register its numerical identifier (e.g., R_2 , R_5 , R_7) as an element in a remap entry array (REA), a sparse, in-bucket array that tracks remapped elements.

23

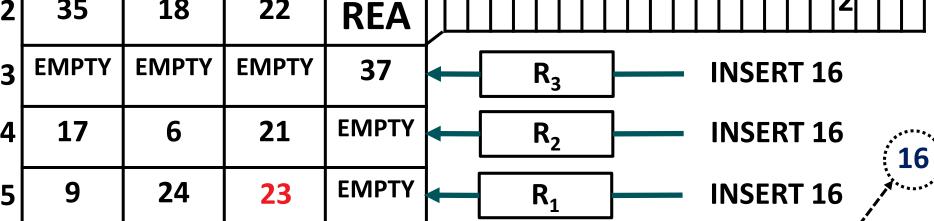
EMPTY

9

24



0 8 5 13 EMPTY
1 33 EMPTY 15 2 20
2 35 18 22 REΛ



Use R₃ 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., R₂, R₂, R₂) 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



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY **EMPTY** 8 5 **13** 0 **EMPTY** 33 **15** 20 **35** 18 22 **REA EMPTY EMPTY EMPTY 37** R_3 **INSERT 16 EMPTY 17** 6 **21 EMPTY** 9 24 23

Use R₃ for inserting 16 because it maps 16 to least full secondary bucket candidate.



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY **EMPTY** 8 5 **13** 0 **EMPTY** 33 **15** 20 **35** 18 22 **REA EMPTY EMPTY 16 37** R_3 **INSERT 16 EMPTY 17** 6 **21 EMPTY** 9 24 23

Use R₃ for inserting 16 because it maps 16 to least full secondary bucket candidate.



24

INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

					Computer Science and Engineering
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	0 20
2	35	18	22	REA	
3	EMPTY	EMPTY	16	37	R ₃ INSERT 16
4	17	6	21	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_2 , R_5 , R_7) as an element in a remap entry array (REA), a sparse, in-bucket array that tracks remapped elements.

23

EMPTY



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



0	8	5	13	EMPTY	Computer Science and Engineering
1	33	EMPTY	15	2	$_{0}$ $H_{tag}(16) = 1$ $_{20}$
2	35	18	22	REA	
3	EMPTY	EMPTY	16	37	R ₃ INSERT 16
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	,

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



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



0	8	5	13	EMPTY	Computer Science and Engineering
1	33	EMPTY	15	2	$_{0}$ \downarrow $H_{tag}(16) = 1$
2	35	18	22	REA	
3	EMPTY	EMPTY	16	37	R ₃ INSERT 16
4	17	6	21	EMPTY	16
5	9	24	23	EMPTY	

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



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



				_	Computer Science and Eng
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	$_{0}$ \downarrow $H_{tag}(16) = 1$
2	35	18	22	REA	
3	EMPTY	EMPTY	16	37	R ₃ INSERT 16
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



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY

UCSDCSE
Computer Science and Engineerin

					Computer Science and Engineering
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	$_{0}$ \downarrow $H_{tag}(16) = 1$
2	35	18	22	REA	
3	EMPTY	EMPTY	16	37	INSERT 16
4	17	6	21	EMPTY	16
5	9	24	23	EMPTY	,

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



INSERTS THAT TRIGGER CREATION OF REMAP ENTRY ARRAY



				_	Computer Science and En
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	$_{0}$ $+$ $H_{tag}(16) = 1$
2	35	18	22	REA	
3	EMPTY	EMPTY	16	37	INSERT 16
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



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.



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.



RETRIEVING REMAPPED ITEMS

UCSDCSE Computer Science and Engineer

					Computer Sci
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	
2	35	18	22	REA	H _{primary} LOOKUP 16
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.



RETRIEVING REMAPPED ITEMS



				Computer Sc
8	5	13	EMPTY	
33	EMPTY	15	2	
35	18	22	REA	H _{primary} LOOKUP 16
EMPTY	EMPTY	16	37	
17	6	21	EMPTY	
9	24	23	EMPTY	
	33 35 EMPTY 17	33 EMPTY 35 18 EMPTY EMPTY 17 6	33 EMPTY 15 35 18 22 EMPTY EMPTY 16 17 6 21	33 EMPTY 15 2 35 18 22 REA EMPTY EMPTY 16 37 17 6 21 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.



RETRIEVING REMAPPED ITEMS

UCSDCSE Computer Science and Engineering
Computer Science and Engineerin

o	8	5	13	EMPTY	Computer Science a	illu El
1	33	EMPTY	15	2	0	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.



RETRIEVING REMAPPED ITEMS

					Computer Science and Engineeri
8	5	13	EMPTY		
33	EMPTY	15	2	0	20
35	18	22	REA		2
EMPTY	EMPTY	16	37		
17	6	21	EMPTY		
9	24	23	EMPTY		
	33 35 EMPTY 17	33 EMPTY 35 18 EMPTY EMPTY 17 6	33 EMPTY 15 35 18 22 EMPTY EMPTY 16 17 6 21	33 EMPTY 15 2 35 18 22 REA EMPTY EMPTY 16 37 17 6 21 EMPTY	33 EMPTY 15 2 35 18 22 REA EMPTY EMPTY 16 37 17 6 21 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.



20

HORTON TABLES

RETRIEVING REMAPPED ITEMS

0	8	5	13	EMPTY		
1	33	EMPTY	15	2	$_{0}$ $H_{tag}(16) = 1$	
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.



20

HORTON TABLES

RETRIEVING REMAPPED ITEMS

9	UCSDCSE Computer Science and Engineerin

0	8	5	13	EMPTY	
1	33	EMPTY	15	2	0 H _{tag} (16) = 1
2	35	18	22	REA	3 1
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.



RETRIEVING REMAPPED ITEMS

	UCSDCSE Computer Science and Engineering
50	Computer Science and Engineering

_						Computer Science
0	8	5	13	EMPTY		
1	33	EMPTY	15	2	$_{0}$ \downarrow $H_{tag}(16) = 1$	
2	35	18	22	REA		2
3	EMPTY	EMPTY	16	37		
4	17	6	21	EMPTY		
5	9	24	23	EMPTY		

The remap entry shows R₃ 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.



RETRIEVING REMAPPED ITEMS

9	UCSDCSE Computer Science and Engineerin
	Computer Science and Engineering

_					Computer Science and Engineering
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	$_{0}$ \downarrow $H_{tag}(16) = 1$ $_{20}$
2	35	18	22	REA	
3	EMPTY	EMPTY	16	37	R ₃ LOOKUP 16
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	

The remap entry shows R₃ 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.



RETRIEVING REMAPPED ITEMS



_					Computer Science and Engli
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	$_{0}$ \downarrow $H_{tag}(16) = 1$
2	35	18	22	REA	
3	EMPTY	EMPTY	16	37	R ₃ LOOKUP 16
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	

The remap entry shows R₃ 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.



RETRIEVING REMAPPED ITEMS

UCSDCSE
Computer Science and Engineering

				Computer Science and Engineering
8	5	13	EMPTY	
33	EMPTY	15	2	$_{0}$ \downarrow $H_{tag}(16) = 1$
35	18	22	REA	
EMPTY	EMPTY	16	37	R ₃ LOOKUP 16
17	6	21	EMPTY	
9	24	23	EMPTY	
	33 35 EMPTY 17	33 EMPTY 35 18 EMPTY EMPTY 17 6	33 EMPTY 15 35 18 22 EMPTY EMPTY 16 17 6 21	33 EMPTY 15 2 35 18 22 REA EMPTY 16 37 17 6 21 EMPTY

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.



RETRIEVING REMAPPED ITEMS

UCSDCSE
Computer Science and Engineerin

_					Computer Science and Engineerin
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	$_{0}$ \downarrow $H_{tag}(16) = 1$
2	35	18	22	REA	
3	EMPTY	EMPTY	16	37	R ₃ LOOKUP 16
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	
•					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.

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







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



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



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



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



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



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



NEGATIVE LOOKUPS

	DCSDCSE omputer Science and Engineering
--	--

_					Computer Science a
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	
2	35	18	22	REA	
3	EMPTY	EMPTY	16	37	H _{primary} LOOKUP 25
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	Most negative lookups only require accessing a single bucket
l					



NEGATIVE LOOKUPS

				_	o impactor o o to trop
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	
2	35	18	22	REA	
3	EMPTY	EMPTY	16	37	H _{primary} LOOKUP 25
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	Most negative lookups only require accessing a single bucket

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



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



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



NEGATIVE LOOKUPS

erin
eı

0	8	5	13	EMPTY	Computer
1	33	EMPTY	15	2	
2	35	18	22	REA	H _{primary} LOOKUP 28
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	Most negative lookups only require accessing a single bucket



NEGATIVE LOOKUPS

UCSDCSE Computer Science and Engineering
Computer Science and Engineering

0	8	5	13	EMPTY	Computer
1	33	EMPTY	15	2	
2	35	18	22	REA	H _{primary} LOOKUP 28
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	Most negative lookups only require accessing a single bucket



NEGATIVE LOOKUPS

?	UCSDCSE Computer Science and Engineering
	Computer Science and Engineerin

	8	5	13	EMPTY	Computer Sci
0	0	,	15		
1	33	EMPTY	15	2	
2	35	18	22	REA	H _{primary} LOOKUP 28
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	Most negative lookups only require accessing a single bucket



NEGATIVE LOOKUPS

UCSDCSE Computer Science and Engineerin
Computer Science and Engineering

0	8	5	13	EMPTY	Computer Sci
1	33	EMPTY	15	2	
2	35	18	22	REA	H _{primary} LOOKUP 28
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	Most negative lookups only require accessing a single bucket



NEGATIVE LOOKUPS

erin
eı

0	8	5	13	EMPTY	Computer
1	33	EMPTY	15	2	
2	35	18	22	REA	H _{primary} LOOKUP 28
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	Most negative lookups only require accessing a single bucket



NEGATIVE LOOKUPS

0	8	5	13	EMPTY	Computer Science and Engine
1	33	EMPTY	15	2	0 20
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



NEGATIVE LOOKUPS

0	8	5	13	EMPTY	Computer Science and
1	33	EMPTY	15	2	0 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



NEGATIVE LOOKUPS

0	8	5	13	EMPTY													
1	33	EMPTY	15	2	<u></u>						_	_	_	_		 20)
2	35	18	22	REA		3									2		
3	EMPTY	EMPTY	16	37													
4	17	6	21	EMPTY													
1				ì	1												

Compute index into remap entry array

using H_{tag} with key as input

▲ Lookups where the primary bucket is *Type B*, buckets where the final slot is converted into an REA, often only require accessing 1 bucket and at most 2 for positive and negative queries alike

EMPTY

23

24



NEGATIVE LOOKUPS

UCSDCSE Computer Science and Engineerin
UCSDCSE Computer Science and Engineering

0	8	5	13	EMPTY	Computer Science and En
1	33	EMPTY	15	2	$_{0}$ $H_{tag}(28) = 10$
2	35	18	22	REA	3
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 LOOKUPS

UCSDCSE Computer Science and Engineerin
Computer Science and Engineeri

0	8	5	13	EMPTY	Computer Science and Engine
1	33	EMPTY	15	2	$_{0}$ $H_{tag}(28) = 10_{20}$
2	35	18	22	REA	3
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	Examine 10 th slot of remap entry array and see it is empty. The search can stop.



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



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*



NEGATIVE LOOKUPS WITH TAG ALIAS

					Compute
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	
2	35	18	22	REA	H _{primary} LOOKUP 7
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	Negative lookups only require accessing 2 buckets on a <i>tag alias</i>



NEGATIVE LOOKUPS WITH TAG ALIAS

					Computer
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	
2	35	18	22	REA	H _{primary} LOOKUP 7
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



LOOKUP 7

HORTON TABLES

NEGATIVE L	TAG ALI	AS
0 8	3 EI	MPTY

 $H_{primary}$

EMPTY 33 **15** 2 18 **35** 22 **REA EMPTY EMPTY 16 37 EMPTY 17** 6 21 4 **EMPTY** 9 **24** 23 5

Negative lookups only require accessing 2 buckets on a tag alias



NEGATIVE LOOKUPS WITH TAG ALIAS

_					Compute
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	
2	35	18	22	REA	H _{primary} LOOKUP 7
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	Negative lookups only require accessing 2 buckets on a <i>tag alias</i>



NEGATIVE LOOKUPS WITH TAG ALIAS

UCSDCSE Computer Science and Engineeri
--

0	8	5	13	EMPTY	Compute
1	33	EMPTY	15	2	
2	35	18	22	REA	H _{primary} LOOKUP 7
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	Negative lookups only require accessing 2 buckets on a <i>tag alias</i>



NEGATIVE LOOKUPS WITH TAG ALIAS

UCSDCSE Computer Science and Engineering
--

0	8	5	13	EMPTY	Computer Science and Engl
1	33	EMPTY	15	2	
2	35	18	22	REA	H _{primary} LOOKUP 7
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	Determine 7 is not stored in its primary bucket (2) and proceed to examine REA
5	9	24	23	EMPTY	Negative lookups only require accessing 2 buckets on a tag alias



NEGATIVE LOOKUPS WITH TAG ALIAS

	0	Е	12	EMPTY	
U	8	5	13	LIVIT	
1	33	EMPTY	15	2	0 20
2	35	18	22	REA	3
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	Negative lookup with a <i>tag alias</i> (e.g., 7 reads remap entry set by 23)



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

	0									 _	2	!O
1		3								2		

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)



NEGATIVE LOOKUPS WITH TAG ALIAS

20

0	8	5	13	EMPTY	
1	33	EMPTY	15	2	$_{0}$ $H_{tag}(7) = 17$
2	35	18	22	REA	3
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	Compute index into rema using H _{tag} with key as inpu
5	9	24	23	EMPTY	Negative lookup with a <i>tag</i> (e.g., 7 reads remap entry s

ap entry array ut

g alias set by 23)



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	

0	H _t	ag	(7) =	= :	17	,					7	2	20
	3											2		

Examine 18^{th} 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)



NEGATIVE LOOKUPS WITH TAG ALIAS

9	UCSDCSE Computer Science and Engineerin

20

0	8	5	13	EMPTY	LOOKUP 7
1	33	EMPTY	15	2	$_{0}$ $H_{tag}(7) = 17$
2	35	18	22	REA	3
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	Examine 18 th slot of remap entry ar see that R ₂ was likely used to remap
5	9	24	23	EMPTY	Negative lookup with a <i>tag alias</i> (e.g., 7 reads remap entry set by 23

ntry array and remap 7.

lias by 23)



NEGATIVE LOOKUPS WITH TAG ALIAS

					Computer Science and Engineering
0	8	5	13	EMPTY	R ₂ LOOKUP 7
1	33	EMPTY	15	2	$_{0}$ $H_{tag}(7) = 17$ $_{20}$
2	35	18	22	REA	3
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	Examine 18 th slot of remap entry array and see that R ₂ was likely used to remap 7.
5	9	24	23	EMPTY	Negative lookup with a <i>tag alias</i> (e.g., 7 reads remap entry set by 23)



NEGATIVE LOOKUPS WITH TAG ALIAS

0	8	5	13	EMPTY	R ₂ LOOKUP 7
1	33	EMPTY	15	2	$_{0}$ H _{tag} (7) = 17 $_{20}$
2	35	18	22	REA	3
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	Determine that no slots of secondary bucket (0) match 7, so stop looking.
5	9	24	23	EMPTY	Negative lookup with a <i>tag alias</i> (e.g., 7 reads remap entry set by 23)

ADDITIONAL CONTENT IN THE PAPER





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

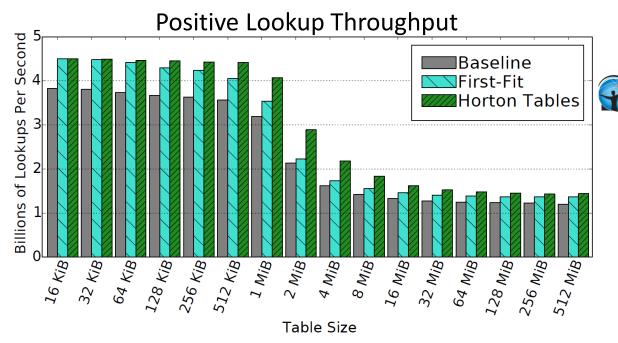
EXPERIMENTAL METHODOLOGY





- ▲ 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 RadeonTM R9 290X GPU

RESULTS POSITIVE LOOKUPS



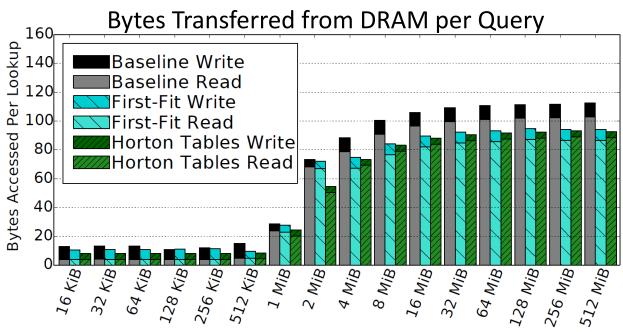
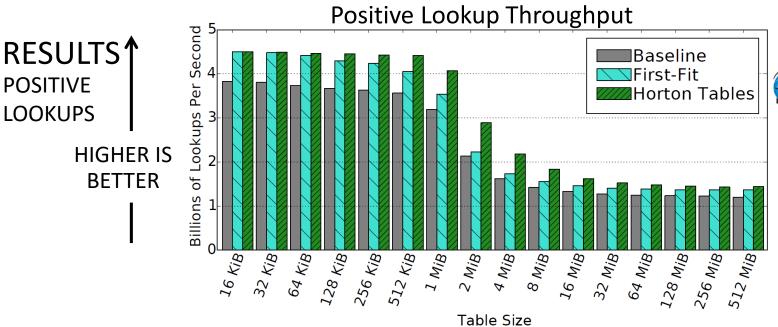
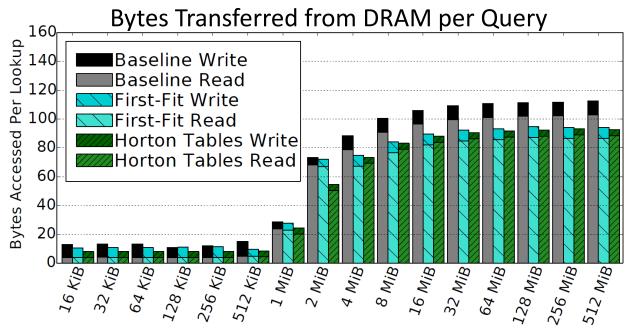


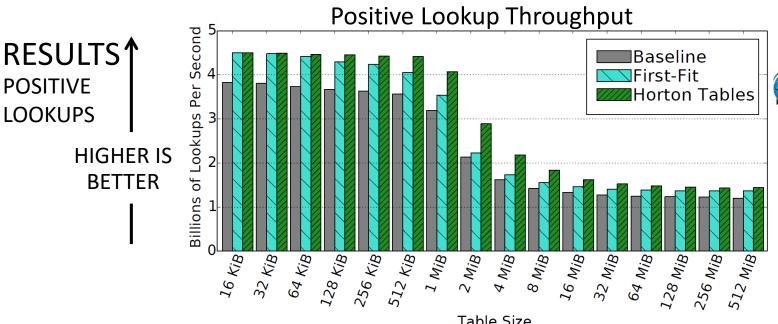
Table Size

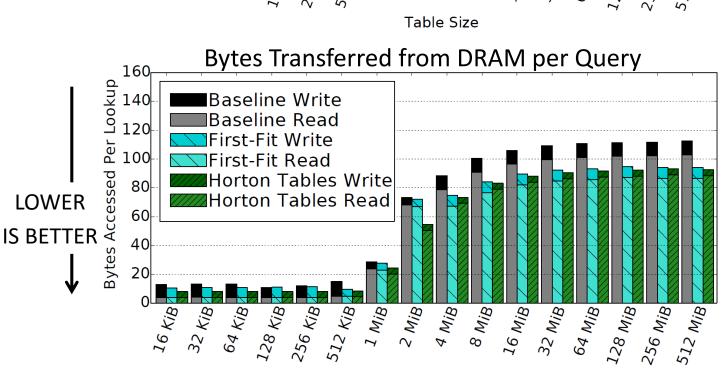


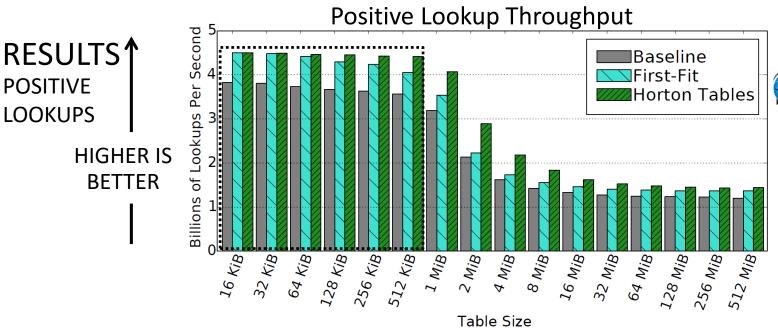






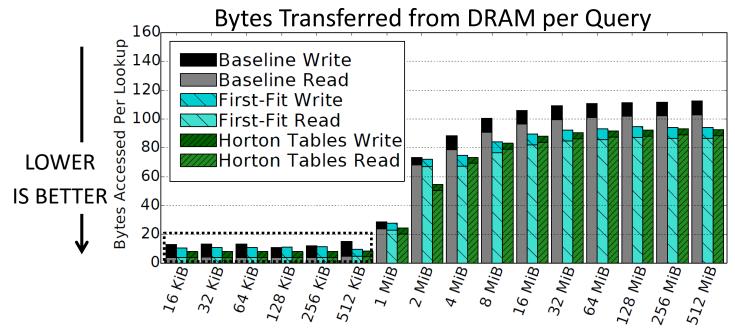


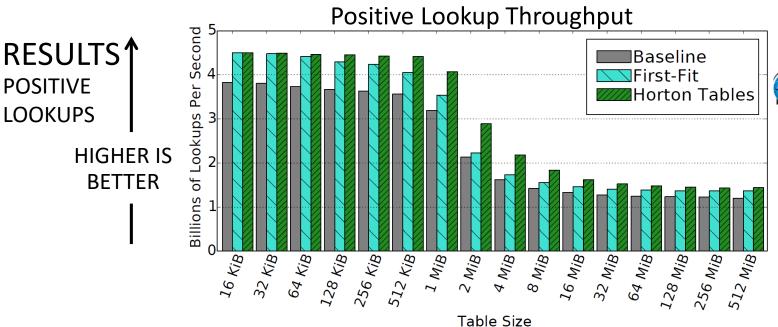






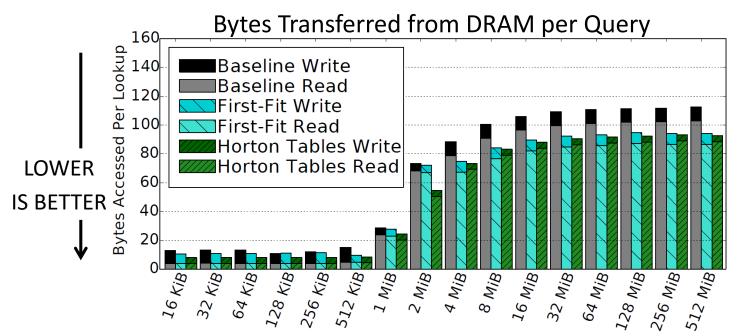


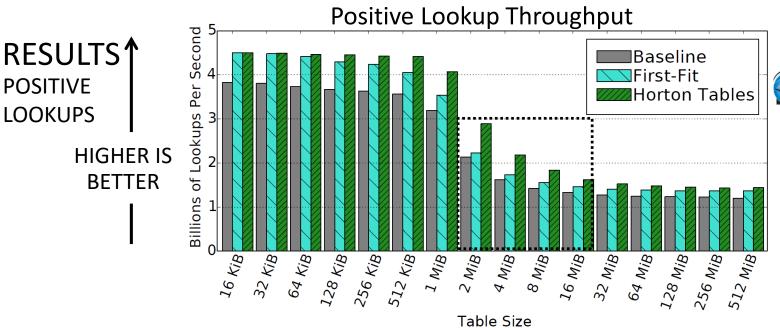




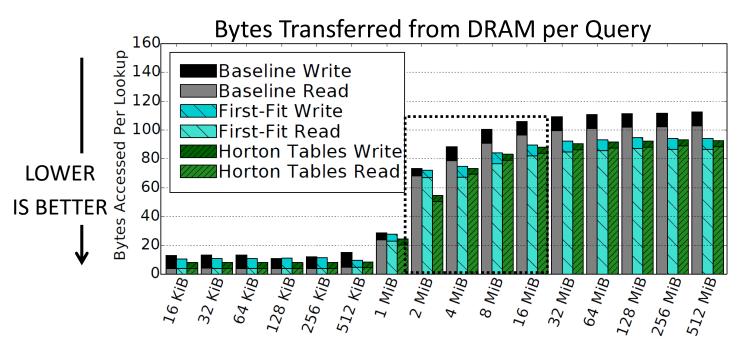


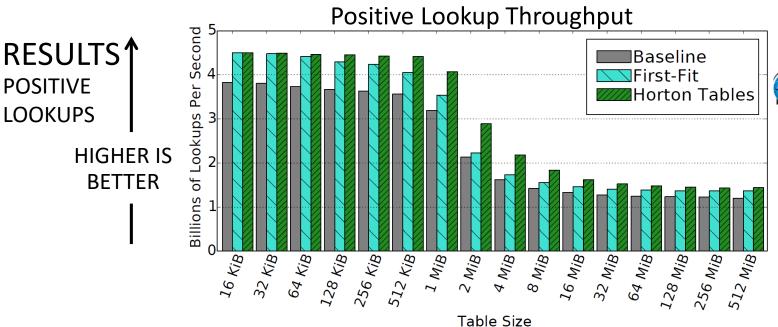






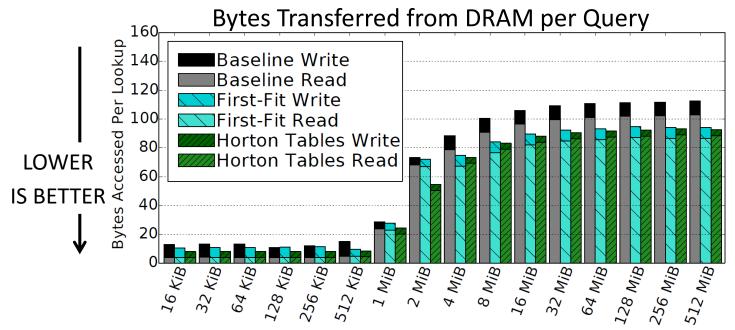


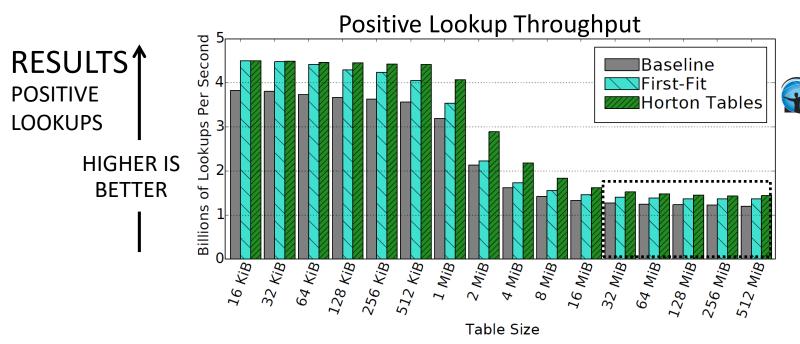


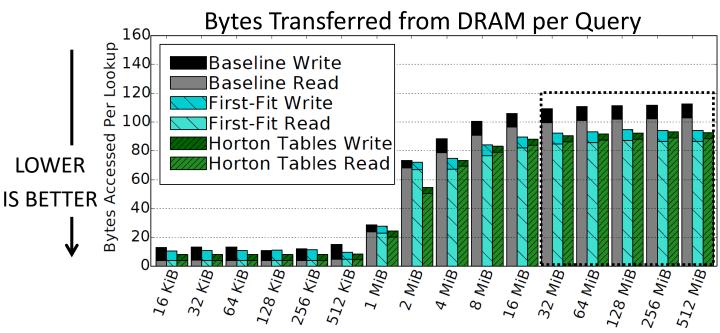


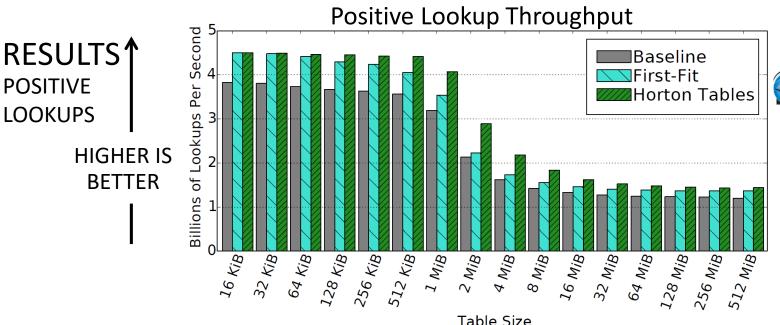


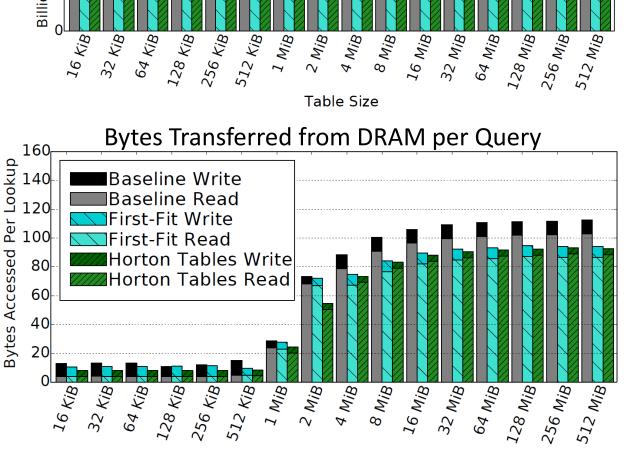








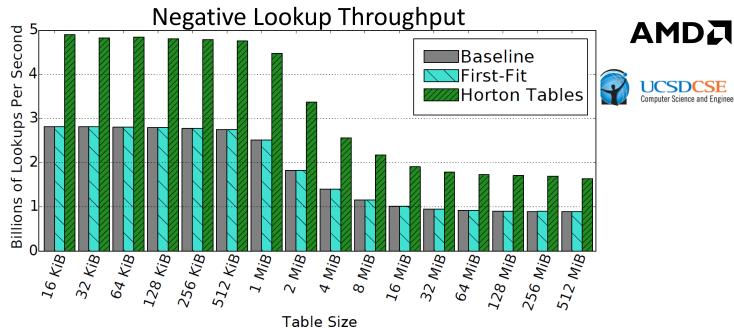


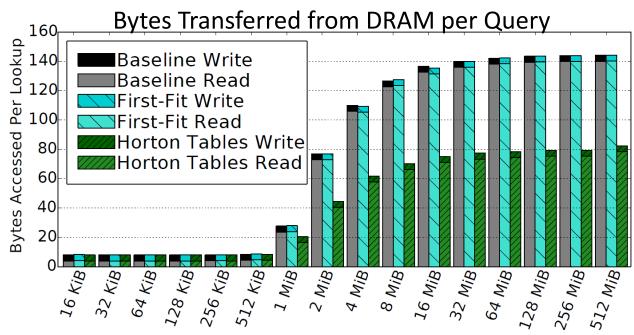


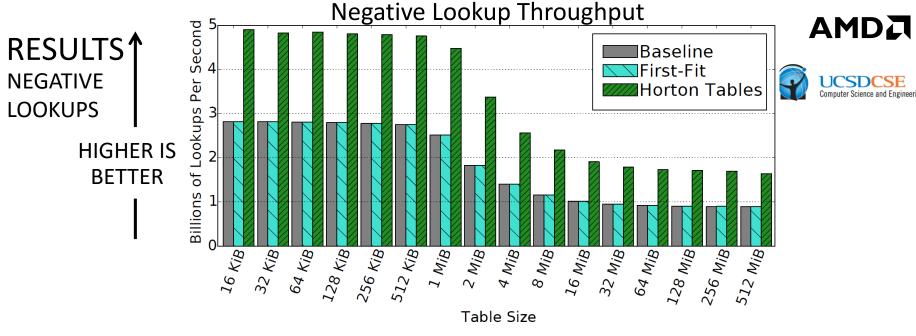
LOWER

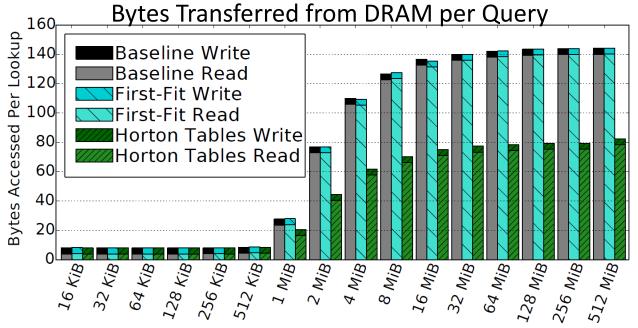
IS BETTER

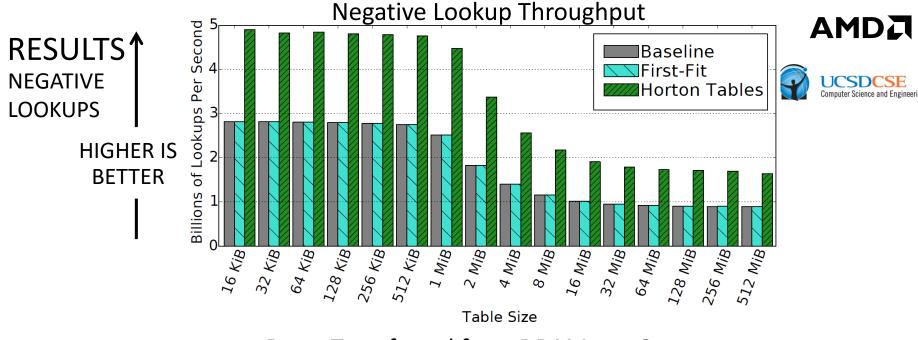
RESULTS NEGATIVE LOOKUPS

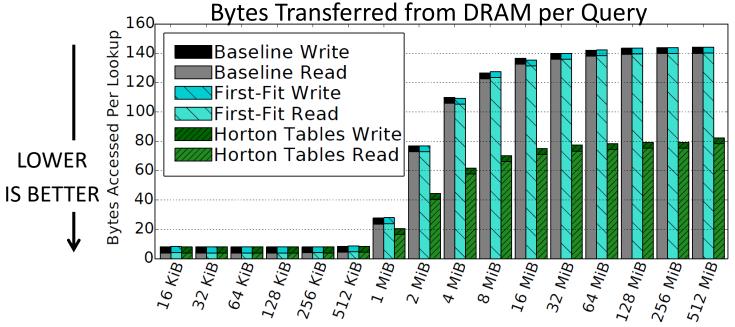


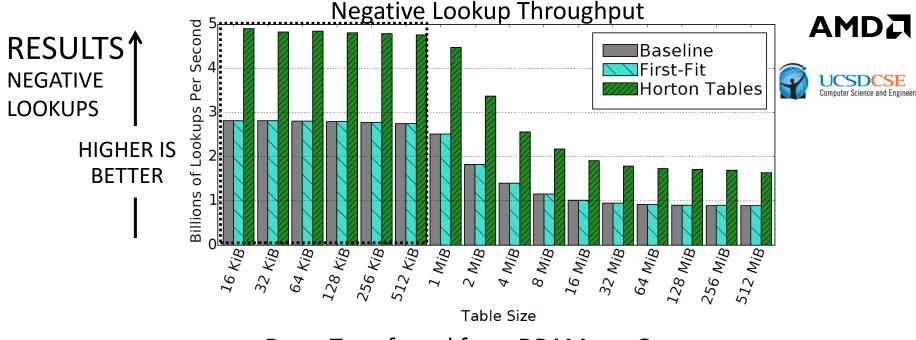


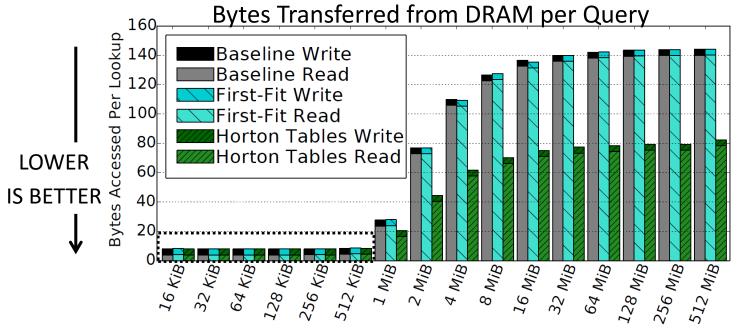


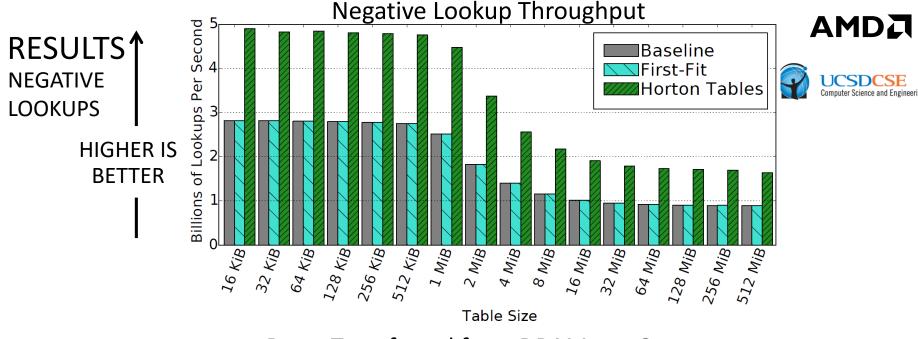


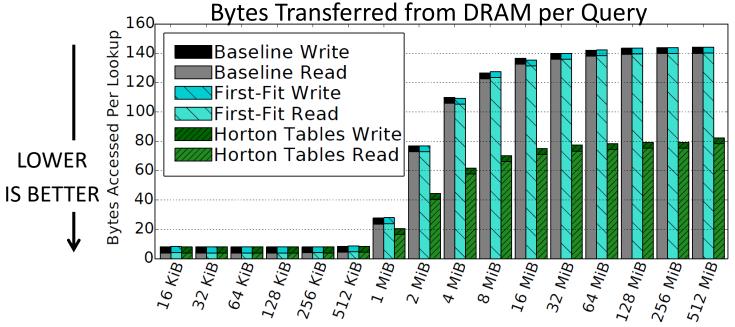


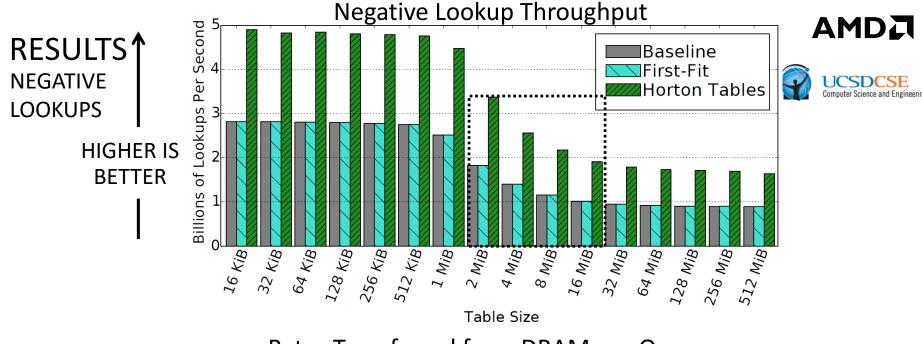


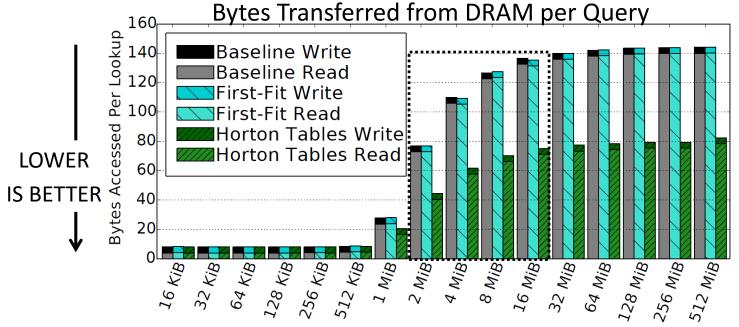


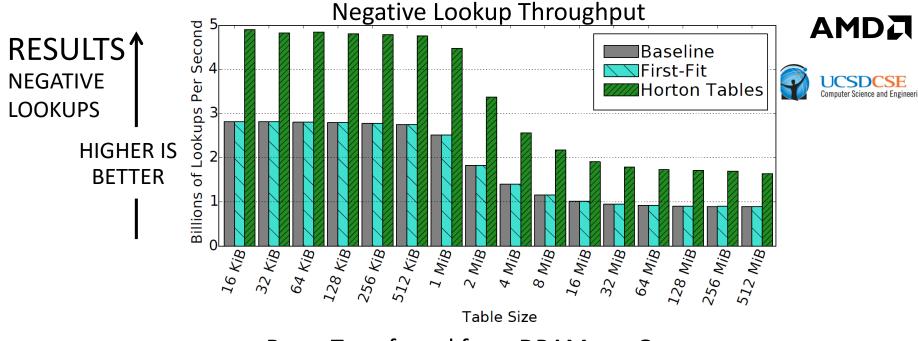


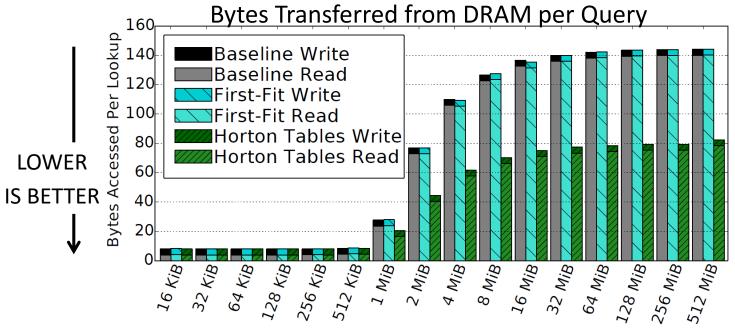


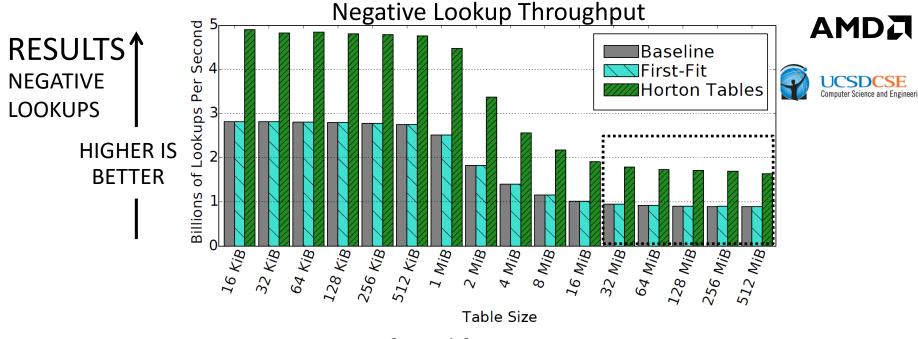


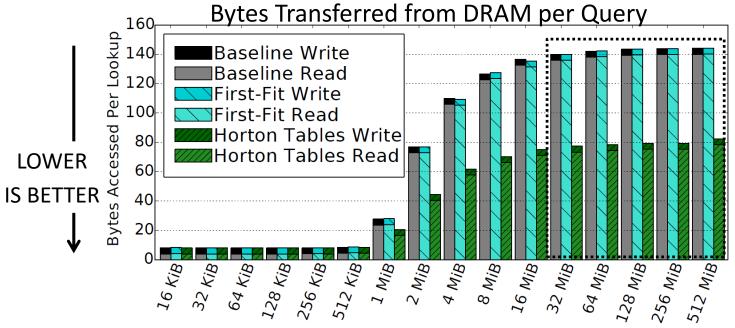


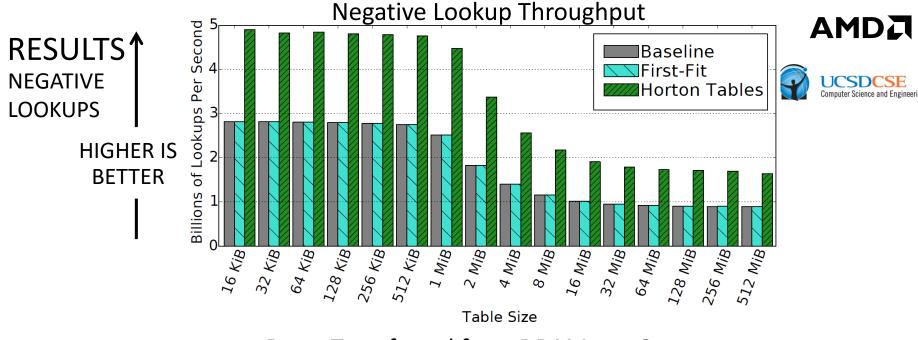


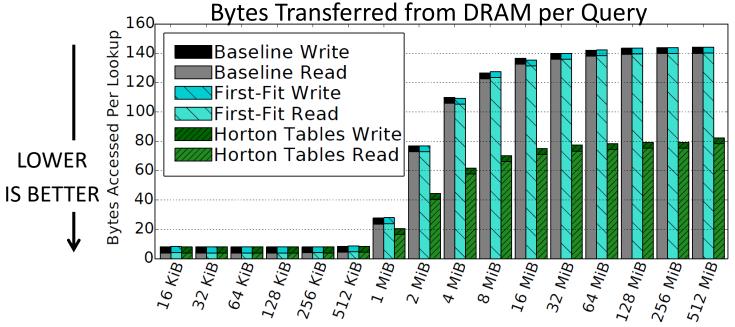
















▲ Achieves lookup throughput that meets or exceeds prior approaches





- Achieves lookup throughput that meets or exceeds prior approaches
- ▲ 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





- Achieves lookup throughput that meets or exceeds prior approaches
- ▲ 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.





- Achieves lookup throughput that meets or exceeds prior approaches
- ▲ 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





- Achieves lookup throughput that meets or exceeds prior approaches
- ▲ 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





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

© 2016 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, AMD Radeon, and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.



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).



Computer Science and Engineering
Computer Science and Engineering

_						Compute
0	8	5	13	EMPTY		
1	33	EMPTY	15	2		
2	35	18	22	REA	H _{primary} INSERT	27
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).



0	8	5	13	EMPTY	Computer Science and
1	33	EMPTY	15	2	
2	35	18	22	REA	H _{primary} INSERT 27
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	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).



0	8	5	13	EMPTY		Computer Science and Engine
1	33	EMPTY	15	2	0	20
2	35	18	22	REA		2
3	EMPTY	EMPTY	16	37		
4	17	6	21	EMPTY		
5	9	24	23	EMPTY		



HORTON TABLES

SHARING OF REMAP ENTRIES

	UCSDCSE Computer Science and Engineerin
--	--

0	8	5	13	EMPTY	
1	33	EMPTY	15	2	0
2	35	18	22	REA	3 1
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	Compute the H _{tag} on the k

key



eerin
)

		_		ENADTY	Computer Science and Engineering
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	$H_{tag}(27) = 1$
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



	UCSDCSE Computer Science and Engineerin
--	--

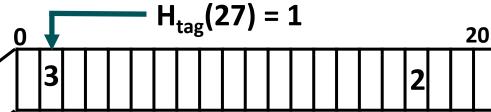
0	8	5	13	EMPTY	Computer Science and Engine
1	33	EMPTY	15	2	$H_{tag}(27) = 1$
2	35	18	22	REA	
3	EMPTY	EMPTY	16	37	
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	Compute the H _{tag} on the key



SHARING OF REMAP ENTRIES

SE d Engineering
SE d Engineer

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



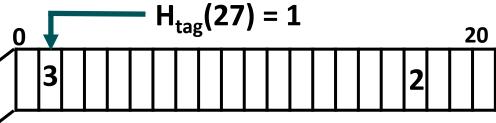
We see that the remap entry is set, so we try to use R_3 to insert 27.



SHARING OF REMAP ENTRIES

SE d Engineering
SE d Engineer

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.





_					Computer Science and Engineerii
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	$H_{\text{tag}}(27) = 1$
2	35	18	22	REA	
3	EMPTY	EMPTY	16	37	R ₃ INSERT 27
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.





0	8	5	13	EMPTY	
1	33	EMPTY	15	2	$H_{tag}(27) = 1$
2	35	18	22	REA	
3	EMPTY	27	16	37	R ₃ INSERT 27
4	17	6	21	EMPTY	
5	9	24	23	EMPTY	The insertion succeeds because the secondary bucket (3) has a free slot.



SHARING OF REMAP ENTRIES

UCSDCSE Computer Science and Engineering
Computer Science and Engineering

					Computer Science and Engineering
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	$H_{\text{tag}}(27) = 1$
2	35	18	22	REA	
3	EMPTY	27	16	37	R ₃ INSERT 27
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 (2) (e.g., 35) and remapped that item instead.



UCSDCSE Computer Science and Engineering
Computer Science and Engineering

				_	
0	8	5	13	EMPTY	
1	33	EMPTY	15	2	$H_{tag}(27) = 1$
2	35	18	22	REA	
3	EMPTY	27	16	37	R ₃ INSERT 27
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 (2) (e.g., 35) and remapped that item instead.
- ▲ Alternatively, we could try to remap both 27 and 16 to another shared bucket with a different secondary hash function, but this is more likely to fail.



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

	UCSDCSE Computer Science and Engineering
DELETE	8

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





					1			
0	8	5	13	EMPTY		H _{primar}	у	DELETE 8
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





				_				Compt
0	EMPTY	5	13	EMPTY	— [H _{primary}	,]——	DELETE 8
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



_	JEEE I III VOI								
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



UCSDCSE Computer Science and Engineerin

_				_	Cor	ompute
0	EMPTY	5	13	EMPTY		
1	33	EMPTY	15	2		
2	35	18	22	REA	H _{primary} DELETE 27	7
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



UCSDCSE Computer Science and Engineering
Computer Science and Engineering

_					Computer Science and Enginee
0	EMPTY	5	13	EMPTY	
1	33	EMPTY	15	2	
2	35	18	22	REA	H _{primary} DELETE 27
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



					_							computer of	iciice and	Liigilieeli
0	8	5	13	EMPTY										
1	33	EMPTY	15	2	0		 	_	_	_	 			20
2	35	18	22	REA		Ш						2		
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



	OMPUTE Science and Engineering
--	--------------------------------

0	EMPTY	5	13	ЕМРТҮ		computer Science and Engi
1	33	EMPTY	15	2	0	20
2	35	18	22	REA		2
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



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

0										2	20
	3								2		

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



_				_	Computer Science and Engin
0	EMPTY	5	13	EMPTY	
1	33	EMPTY	15	2	$H_{tag}(27) = 1$
2	35	18	22	REA	3
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



DELETING ELEMENTS

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

0	Ţ	$\mathbf{H}_{tag}(27) = 1$									0				
	3												2		

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



0	EMPTY	5	13	EMPTY	Computer Science and Engineeri
1	33	EMPTY	15	2	$H_{tag}(27) = 1$
2	35	18	22	REA	
3	EMPTY	27	16	37	DELETE 27
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





_				_	Computer Science and Engineerin
0	EMPTY	5	13	EMPTY	
1	33	EMPTY	15	2	$H_{\text{tag}}(27) = 1$
2	35	18	22	REA	
3	EMPTY	27	16	37	R ₃ DELETE 27
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





1				1	Computer Science and Engineering
0	EMPTY	5	13	EMPTY	
1	33	EMPTY	15	2	$H_{tag}(27) = 1$
2	35	18	22	REA	
3	EMPTY	27	16	37	R ₃ DELETE 27
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





0	EMPTY	5	13	EMPTY	Computer Science and Engineering
1	33	EMPTY	15	2	$H_{tag}(27) = 1$
2	35	18	22	REA	
3	EMPTY	EMPTY	16	37	R ₃ DELETE 27
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





_				_	Computer Science and Engineerin
0	EMPTY	5	13	EMPTY	
1	33	EMPTY	15	2	$H_{\text{tag}}(27) = 1$
2	35	18	22	REA	3
3	EMPTY	EMPTY	16	37	R ₃ DELETE 27
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





_					Computer Science and Engineerin
0	EMPTY	5	13	EMPTY	
1	33	EMPTY	15	2	$H_{\text{tag}}(27) = 1$
2	35	18	22	REA	
3	EMPTY	EMPTY	16	37	R ₃ DELETE 27
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





					Computer Science and Engineering
0	EMPTY	5	13	EMPTY	
1	33	EMPTY	15	2	$H_{\text{tag}}(27) = 1$
2	35	18	22	REA	
3	EMPTY	EMPTY	EMPTY	37	R ₃ DELETE 27
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





_					Computer Science and Engineerin
0	EMPTY	5	13	EMPTY	
1	33	EMPTY	15	2	$H_{\text{tag}}(27) = 1$
2	35	18	22	REA	
3	EMPTY	EMPTY	EMPTY	37	R ₃ DELETE 27
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



END OF BACKUP