Zombie Hashing

Reanimating tombstones in a Graveyard

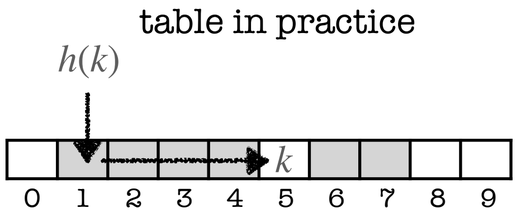
Yuvaraj Chesetti*, Benwei Shi*, Jeff M. Phillips, Prashant Pandey

Northeastern University

THE UNIVERSITY OF UTAH®

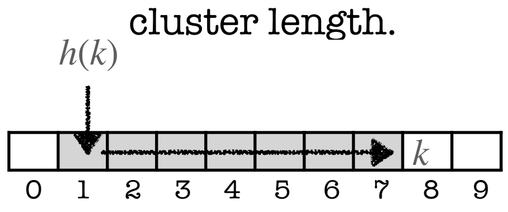
Primary Clustering effects in Linear Probing Hash Tables

One of the fastest hash table in practice



Sequential access -> Data locality

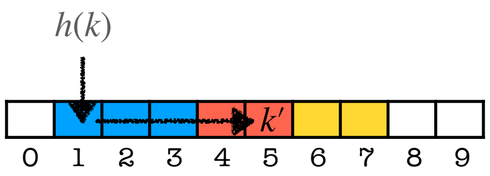
Operations are bounded by cluster length.



However, clusters grow longer more inserts, an effect known as **primary clustering**

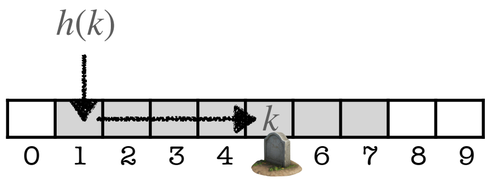
Mitigating primary clustering

Ordered Linear Probing (RobinHood Hashing)



Items are clustered by their home slot. Query can terminate if h(k')>h(k). Inserts are still bounded by cluster

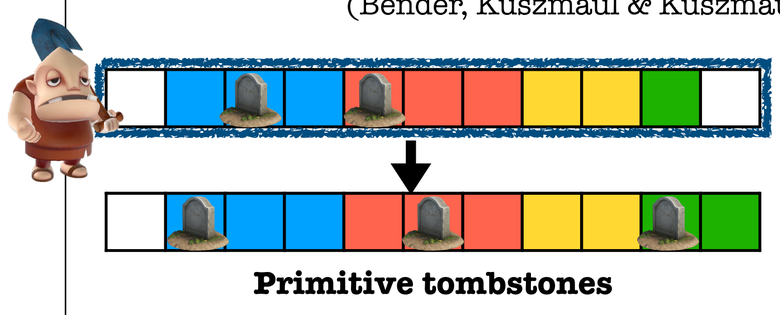
Tombstones (Deletion Markers)



Tombstones have anti-clustering effects. Requires **periodic cleanup** of tombstones

Graveyard hashing

(Bender, Kuszmaul & Kuszmaul, FOCS 2023)

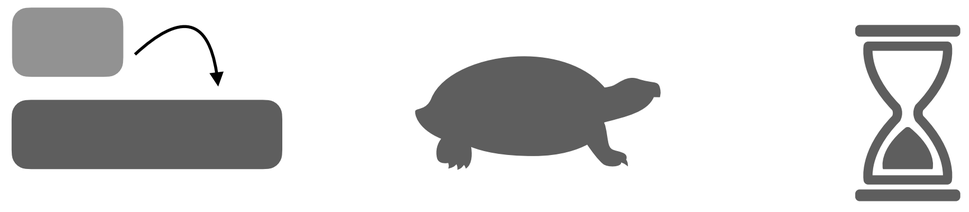


Primitive tombstones

Ordered Linear Probing + Tombstone redistribution.

Periodically rebuild and place **Primitive Tombstones**

Trade-offs



Over allocate
Low space efficiency

Non-linear probing
Low cache efficiency

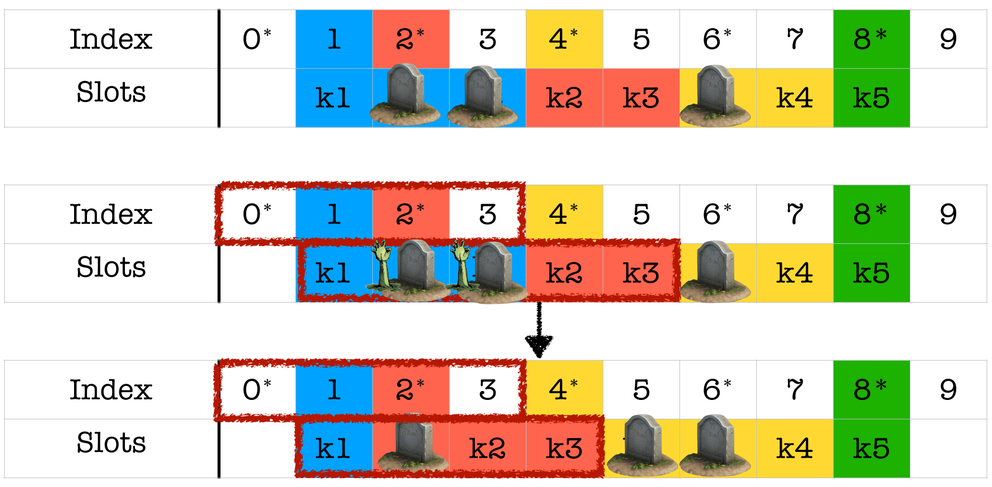
Periodically Rebuild
High 99p latency

These tradeoffs matter significantly for in-memory caches, network stream monitoring and metagenomic applications

Research Question: Can we overcome the tradeoff between space and performance?

ZombieHT: Deamortized Rebuild Schedule

Rebuild a small part of the table after each update



Primitive tombstone positions must contain a tombstone after rebuild window passes through it.
*Here every 2nd slot is a primitive tombstone position

Current Rebuild Window: Redistribute tombstones in these runs after an insert/delete.

Rebuild process: Leave tombstone at primitive tombstone positions and push excess out for next rebuild.

Rebuild window progresses and is rebuilt after another insert/ delete.

Excess tombstones pushed to empty slots are removed. (Rebuilding maintains RobinHood hashing invariant)

Additional tombstones are introduced if needed.

Space Efficiency

Key hash is divided into quotient and remainder. Quotient is stored implicitly as home slot and marked with metadata bits.

Index	q0	q1	q2	q3	q4	q5	q6	q7	q8	q9
occupied		1	1		1				1	
runend		0	0	1	0	1	0	1	1	
tombstone		0	1	1	0	0	1	0	0	
remainder		r1			r2	r3		r4	r5	
keys		k1			k2	k3		k4	k5	

Hash Table	Space
ZombieHT	92.12%
CuckooHT	74.96%
ZombieHT(V)	70.55%
AbslHT	70.55%
IcebergHT	62.71%
CLHT	35.66%

ZombieHT: Quotient-Filter based design with high space efficiency and $O(x)$ performance.

ZombieHT(V): High performance through **vectorized instructions** on an **unordered linear probing** based design. Rebuilds entire cluster and has $O(x^2)$ performance. Based on AbslHT, a production hash table by Google.

Code hosted at: https://github.com/saltsystemslab/ZombieHT

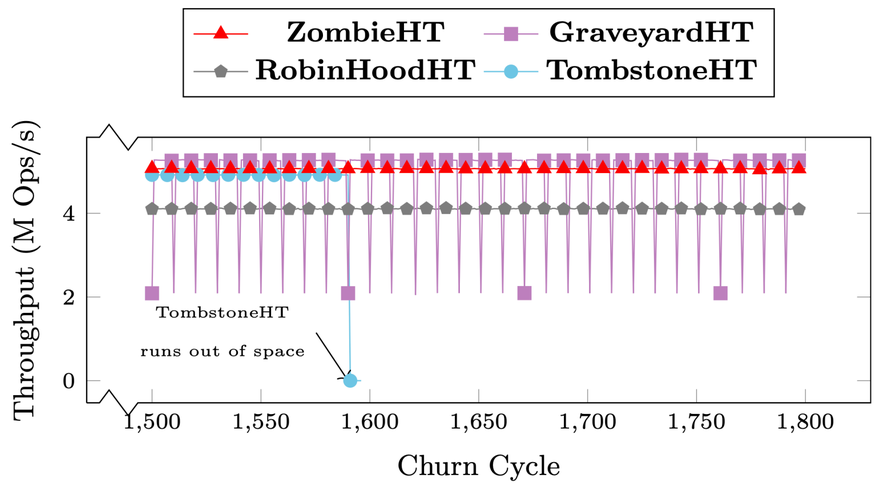
Only $O(x)$ work done per operation with no rebuilds by choosing primitive tombstone slot distance and rebuild window size as constant factor of x, where hash table with load factor $\alpha = 1 - 1/x$

Variant	Insert	Delete	Query	Rebuild?
Linear Probing	$\Theta(x^2)$	$\Theta(x^2)$	$\Theta(x^2)$	No
RobinHood	$\Theta(x^2)$	$\Theta(x^2)$	$\Theta(x)$	No
RobinHood + T.S	$\tilde{\Theta}(x^{1.5})$	$\Theta(x)$	$\Theta(x)$	Yes
Graveyard	$\Theta(x)$	$\Theta(x)$	$\Theta(x)$	Yes
Zombie Hashing	$\Theta(x)$	$\Theta(x)$	$\Theta(x)$	No

Experiments measure **throughput** and **latency** at **95% load factor** for churn **cycles** with **50/50 inserts and deletes**

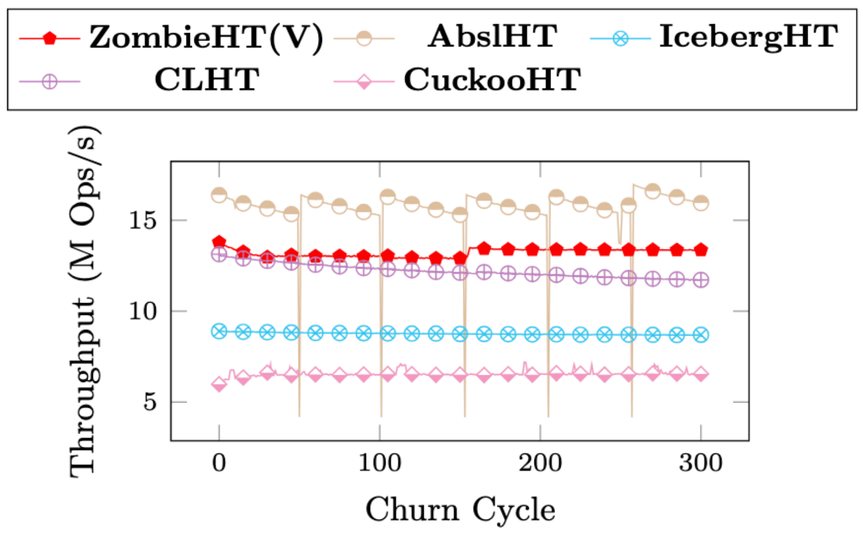
ZombieHT

Max Latency (us)	Insert	Delete	Query
ZombieHT(C)	349.90	68.04	74.47
Graveyard	2173487.96	58.47	70.25



ZombieHT(V)

Max Latency (us)	Insert	Delete	Query
ZombieHT(V)	140.79	71.22	70.03
AbslHT	1308525.04	36.29	73.84



ZombieHT: Consistent high throughput with **NO** interruptions **AND** high space utilization!