

# Verified double-hashing HashTable

Martin VASSOR

DSLab, EPFL



January 10, 2017

# Outline

## Introduction

## Implementation

- Modifications

- Performance evaluation

- Performance results

## Verification

- What to prove ?

- Proof steps

## Conclusion

- Hash Table software

- Remaining work

- Side effects

# Outline

## Introduction

## Implementation

- Modifications

- Performance evaluation

- Performance results

## Verification

- What to prove ?

- Proof steps

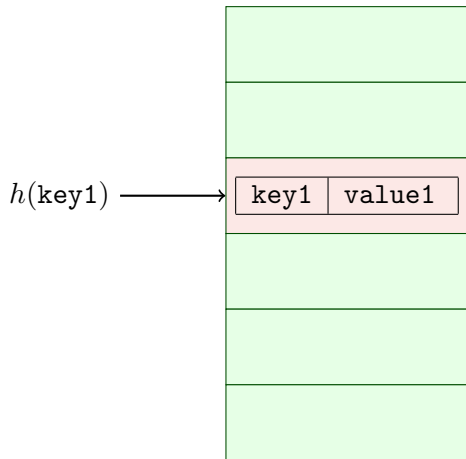
## Conclusion

- Hash Table software

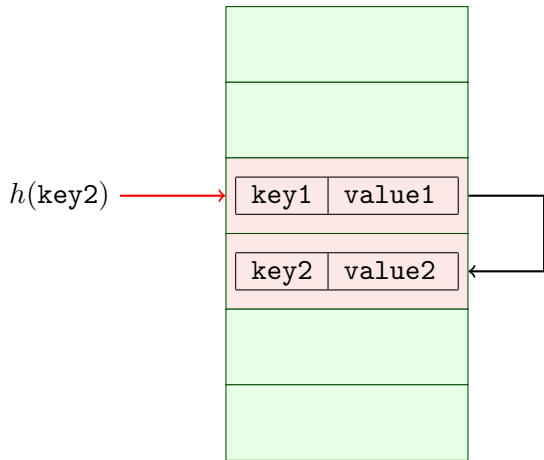
- Remaining work

- Side effects

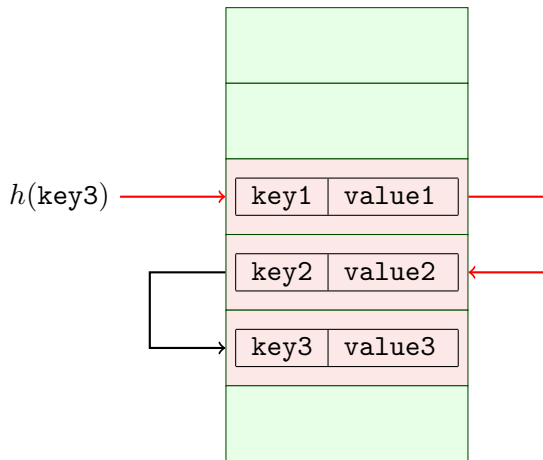
# Naive hash table



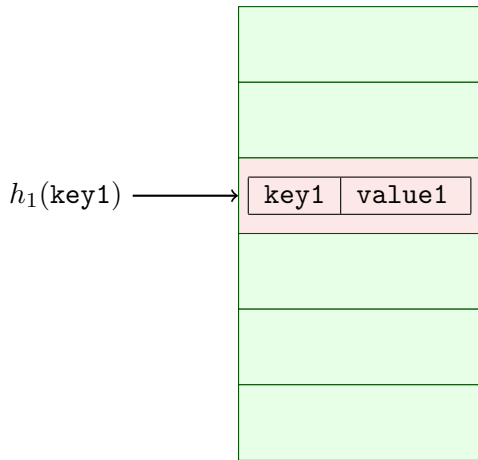
## Naive hash table



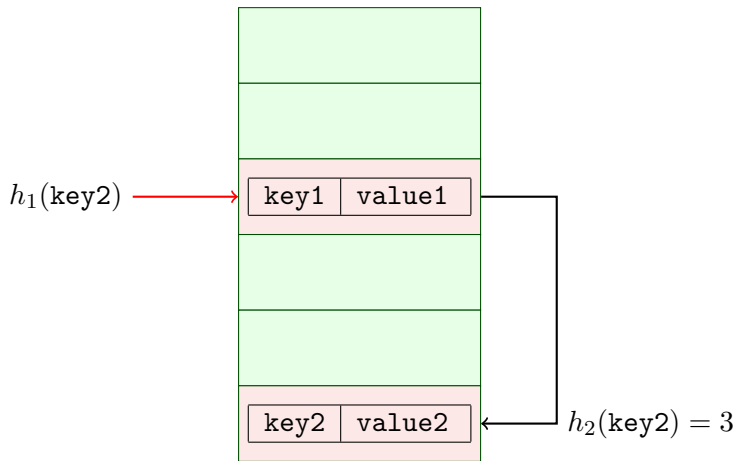
# Naive hash table



# Double hashing

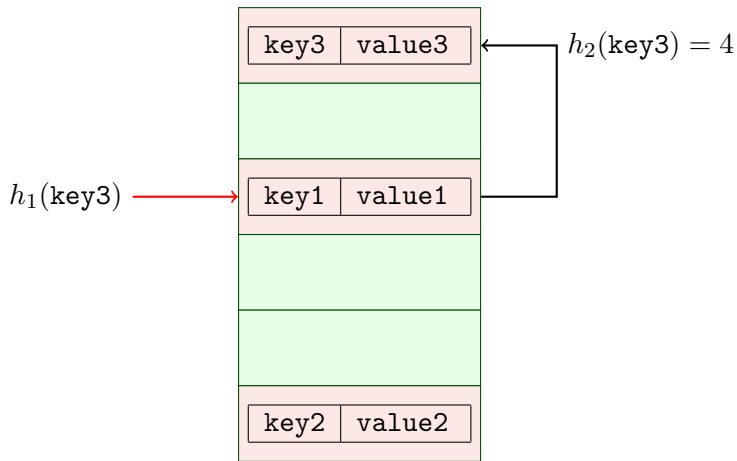


# Double hashing





# Double hashing

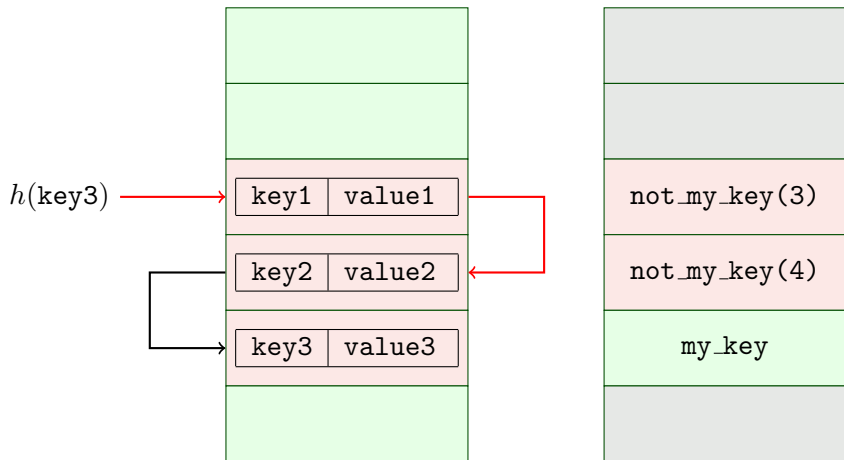


## Provided implementation

- ▶ A naive implementation
- ▶ `findEmpty`, `findKey` perform the loops.

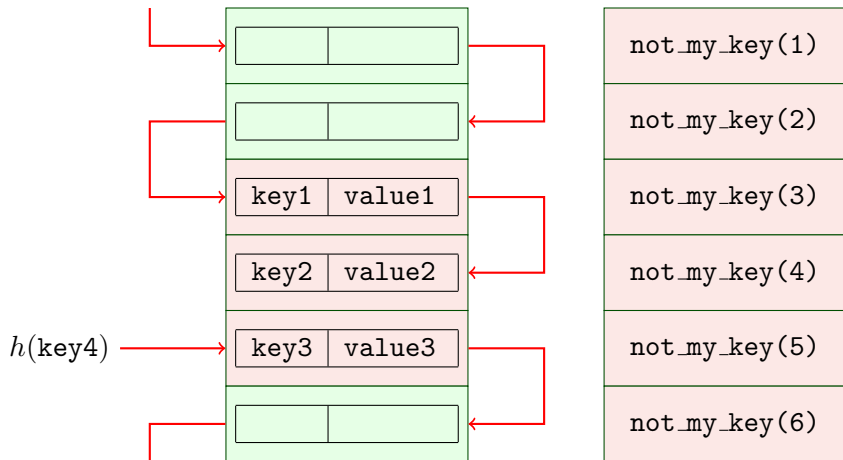
# Provided verification

Example: successful search of `key3`



# Provided verification

Example: unsuccessful search of `key4`



## Provided verification

Part before and after “ $\forall i. \text{not\_my\_key}(i) = \text{true}$ ” provided.

For insertion:

- ▶ Same idea
- ▶ Property: `findEmpty`

# Outline

## Introduction

## Implementation

- Modifications

- Performance evaluation

- Performance results

## Verification

- What to prove ?

- Proof steps

## Conclusion

- Hash Table software

- Remaining work

- Side effects

# Modifications

- ▶ 64 bits hashes.

offset	entry
--------	-------

Except type changes, only **for** loops modified.

# Performance evaluation

- ▶ Build a benchmark tool.
- ▶ Size, number of accesses, load, read/write ratio, etc...
- ▶ Converter to C file.
- ▶ First warms-up, then measures when target load is reached.

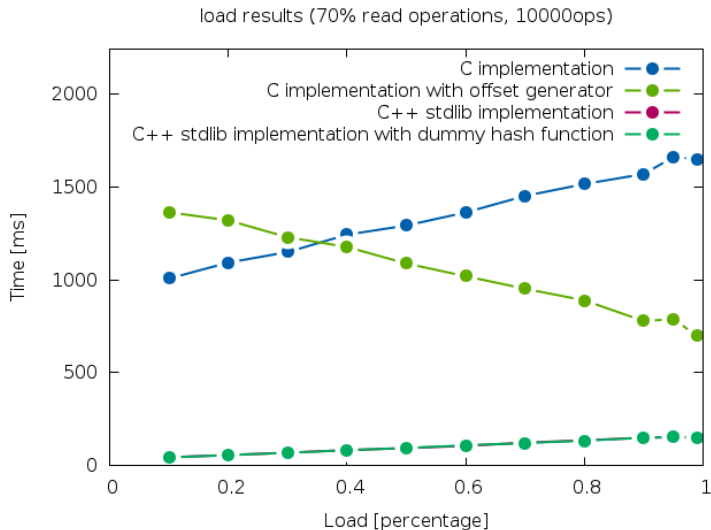
```
test_load.sh length read_ratio load1 [load2...]
```



# Evaluation cases

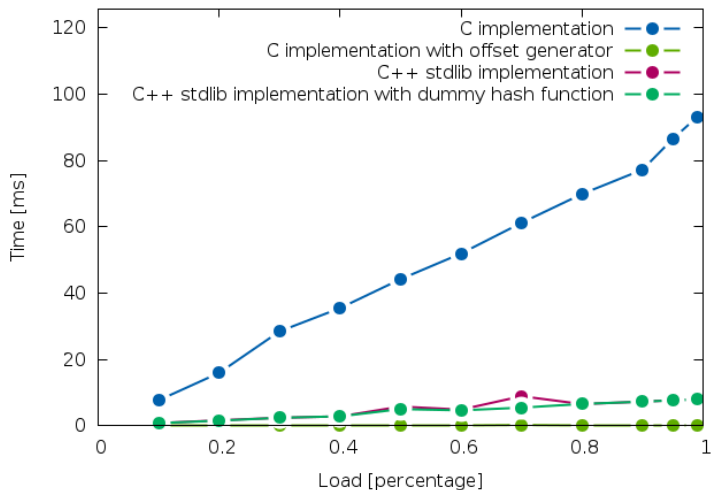
- ▶ Worst case: searching a non existing element.
1. Allow searching non existing element.
  2. Search only existing element.

# Result



## Result – only existing

load results (70% read operations, 1000ops, access only existing elements)



# Outline

## Introduction

## Implementation

- Modifications

- Performance evaluation

- Performance results

## Verification

- What to prove ?

- Proof steps

## Conclusion

- Hash Table software

- Remaining work

- Side effects

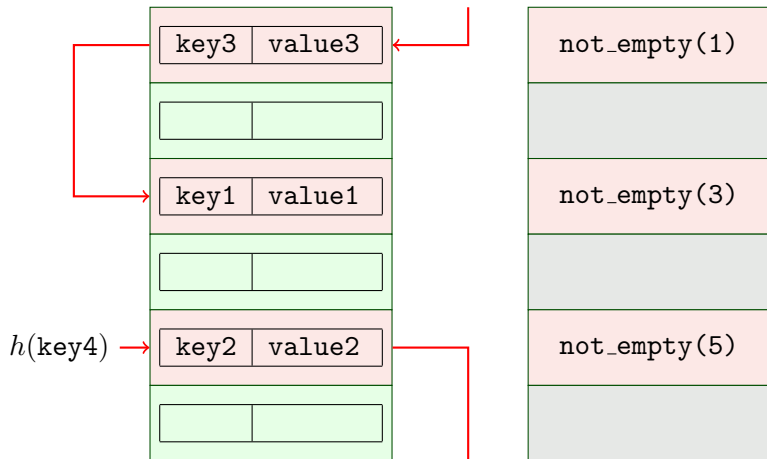
# What to prove ?

Goal: show that increment by offset covers all the map.

- ▶ Not always true (chinese remainder theorem).
- ▶ Requires: **offset** and **capacity** coprime ( $\gcd = 1$ ) (necessary and sufficient).

## What to prove ?

Insert **key4**:  $h_1(\text{key4}) = 5$ ,  $h_2(\text{key4}) = 2$ : search empty ?



## Proof steps

If the number of iteration is less than the capacity:

- ▶ Build and updated a `list<option<nat>>` with the same pattern.
- ▶ Each cell is:
  - ▶ `some(n)` if accessed after  $n$  iterations.
  - ▶ `none` if not accessed.
- ▶ Apply Chinese Remainder Theorem.
- ▶ Deduce that only `none` are updated to `some`.
- ▶ Hence, the number of `some` is the number of iteration.
- ▶ For `capacity` iteration, all cells are `some`.

## Proof steps

If `some(n)`, then `prop(start+offset*n % capa)`.

stripe(capacity=7, offset=2, iter=7)	none	
	some(4)	prop(2) holds
	some(1)	prop(3) holds
	some(5)	prop(4) holds
	some(2)	prop(5) holds
	none	
	some(3)	prop(7) holds



## Proof steps

stripe(capacity=7, offset=2, iter=7)	some(7)
	some(4)
	some(1)
	some(5)
	some(2)
	some(6)
	some(3)

⇒ `count_some = iter = 7`

⇒ All cells are **some**.

# Outline

## Introduction

## Implementation

- Modifications

- Performance evaluation

- Performance results

## Verification

- What to prove ?

- Proof steps

## Conclusion

- Hash Table software

- Remaining work

- Side effects

# Hash-Table software

- ▶ Efficient (when key is present).
- ▶ Formally verified.
- ▶ Requires `capacity` and `offset` coprime.

# Remaining work<sup>1</sup>

- ▶ Two lemmas:
  - ▶ Multiplication associativity:  $(n1 \cdot n2) \cdot n3 = n1 \cdot (n2 \cdot n3)$
  - ▶ Coprime factorization:  $a \perp c \wedge b \perp c \Rightarrow (a \cdot b) \perp c$
- ▶ Some typing errors

---

<sup>1</sup>Updated 10/01/2017@3am

## Side effects

- ▶ 6 commits in Verifast tree (`long long` support).
- ▶ 9 issues on Verifast.
- ▶ A random access sequence generator & benchmark.

# Q&A