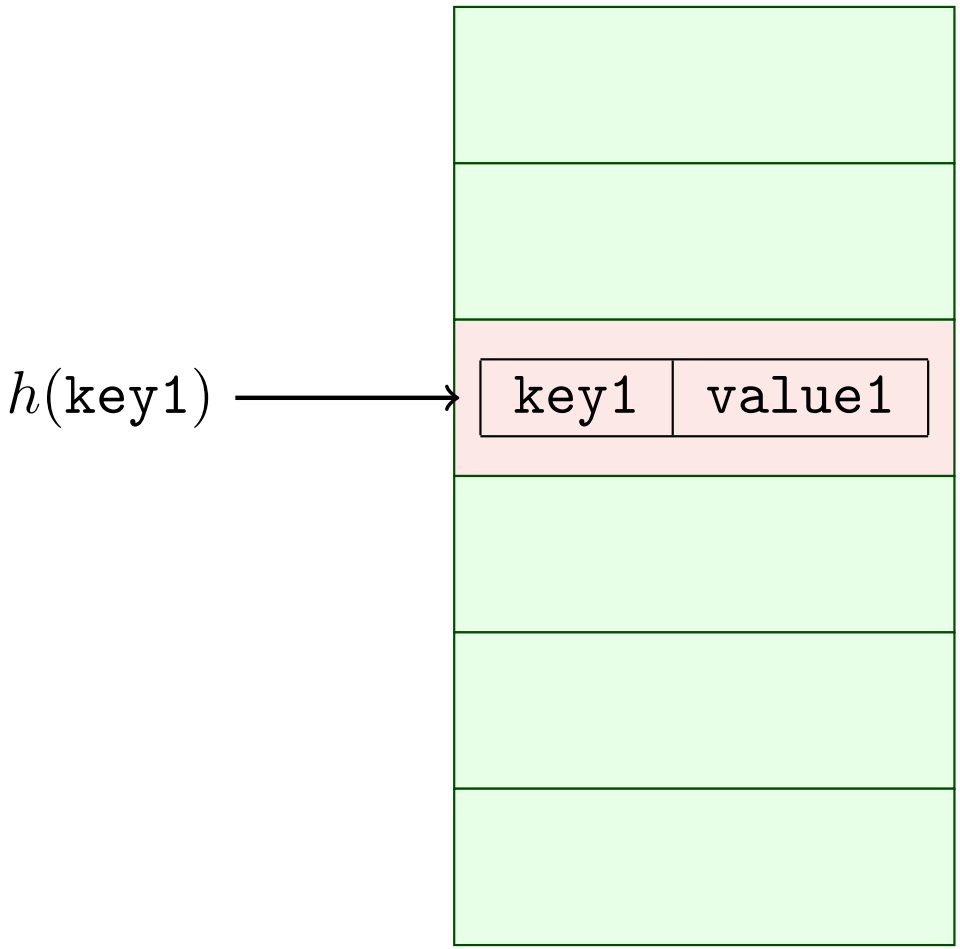
Martin Vassor

DSLab, EPFL

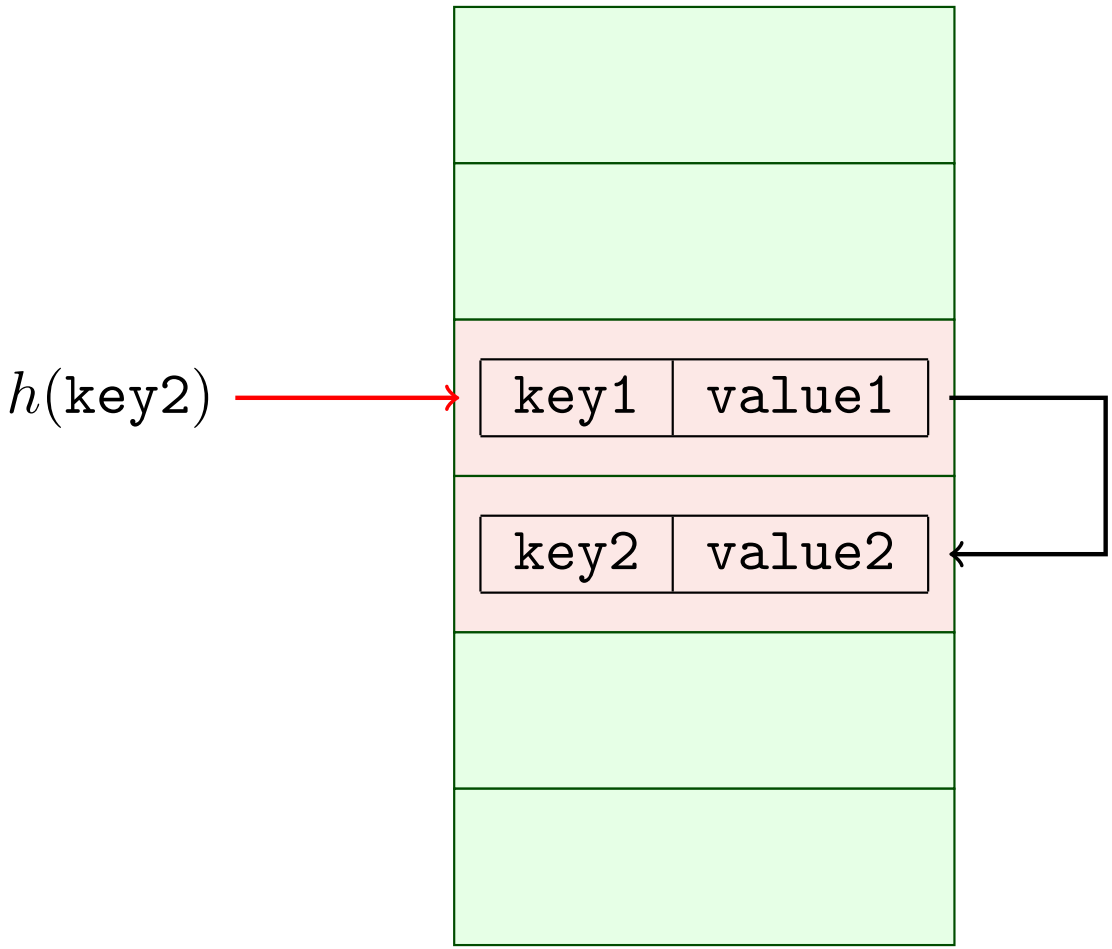
January 8, 2017

Outline

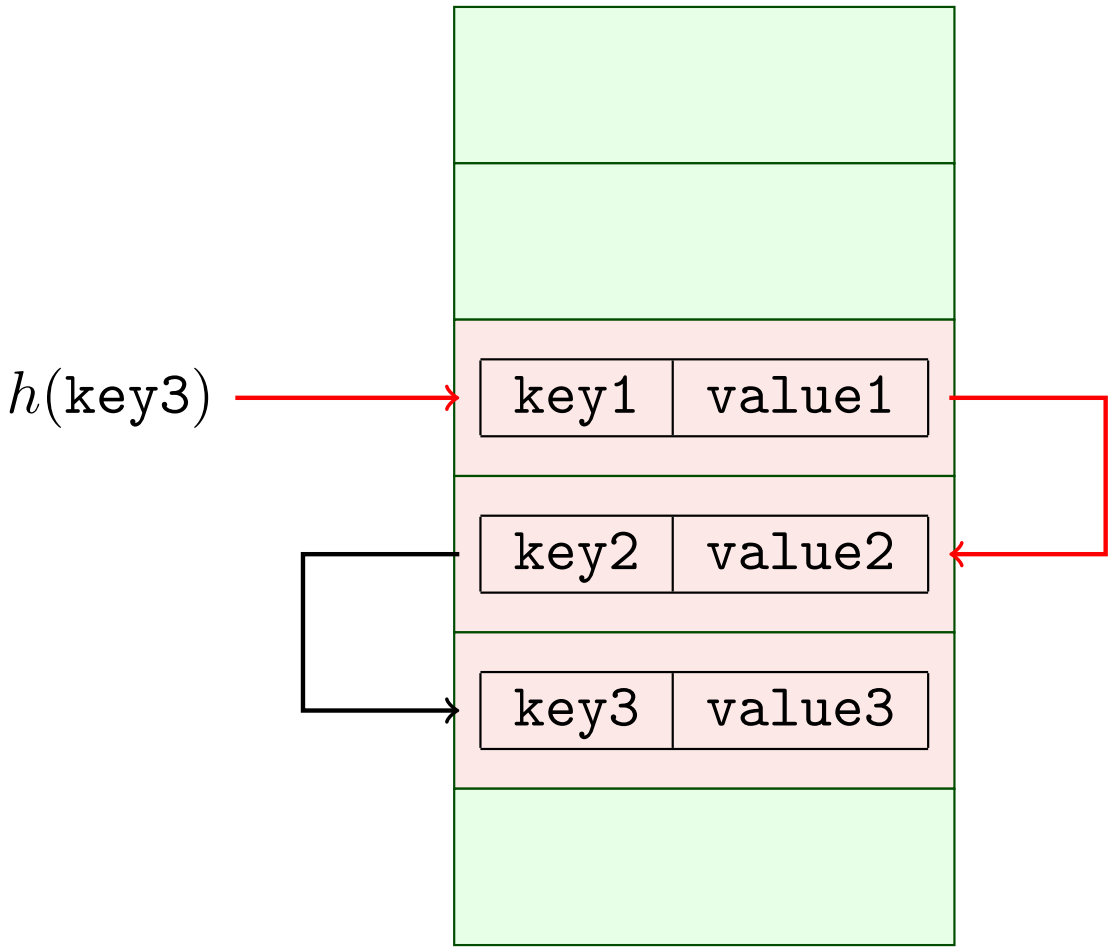
Naive hash table



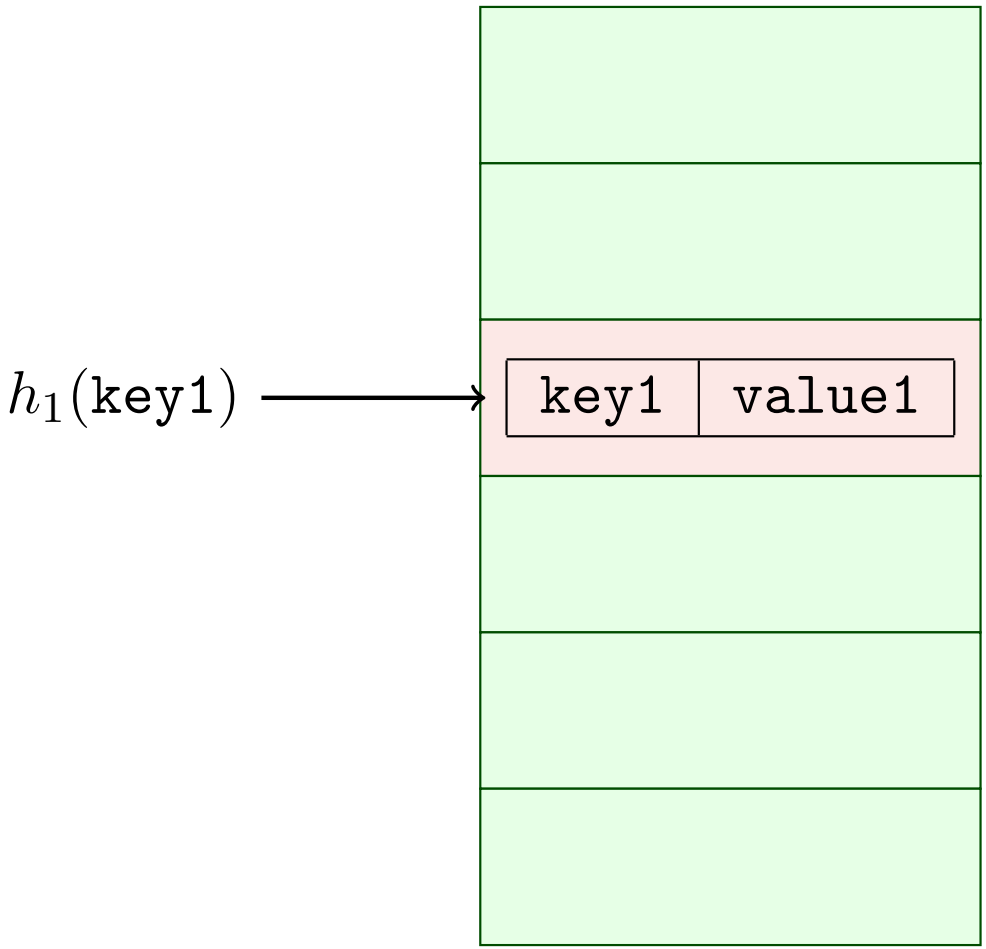
Naive hash table



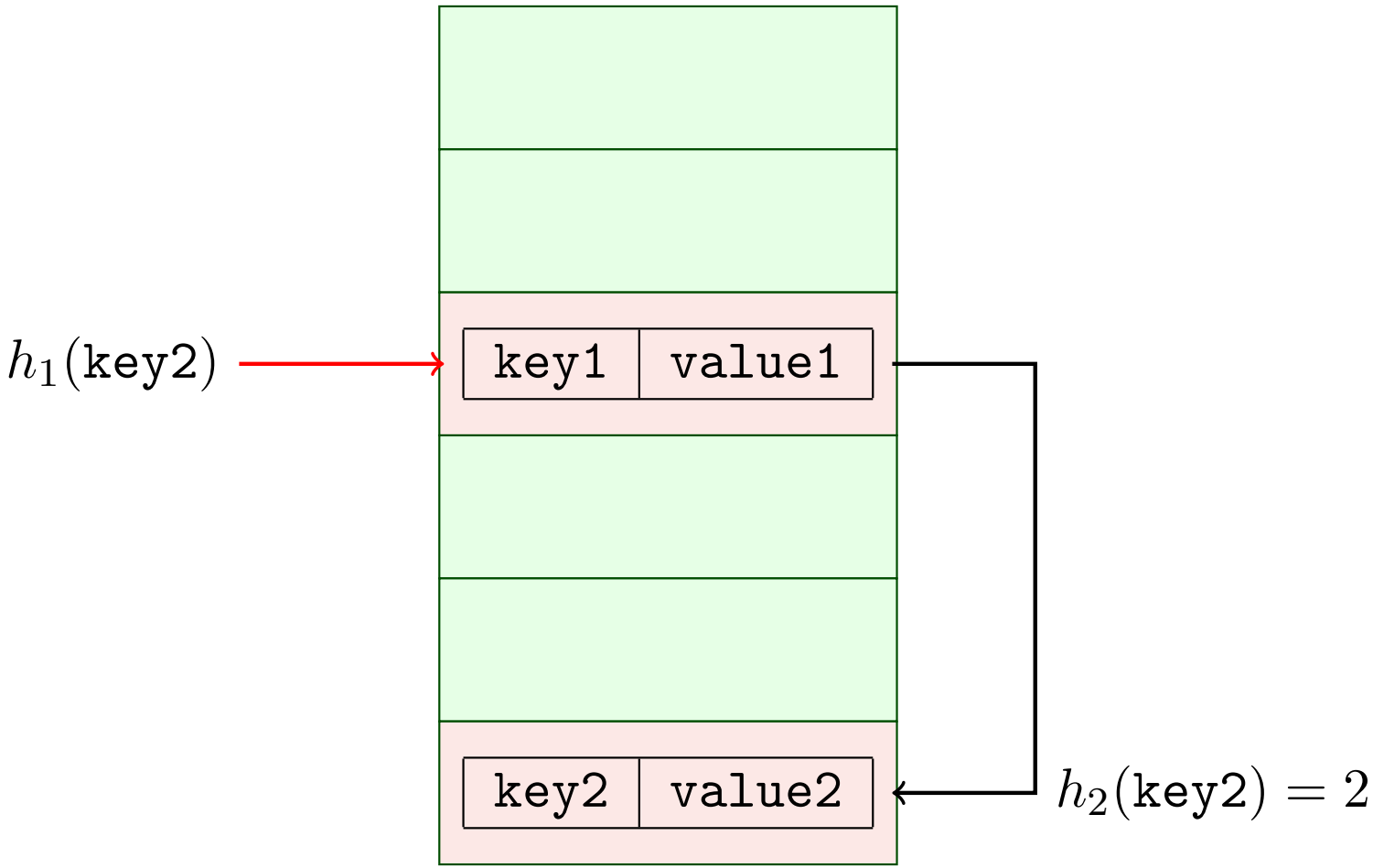
Naive hash table



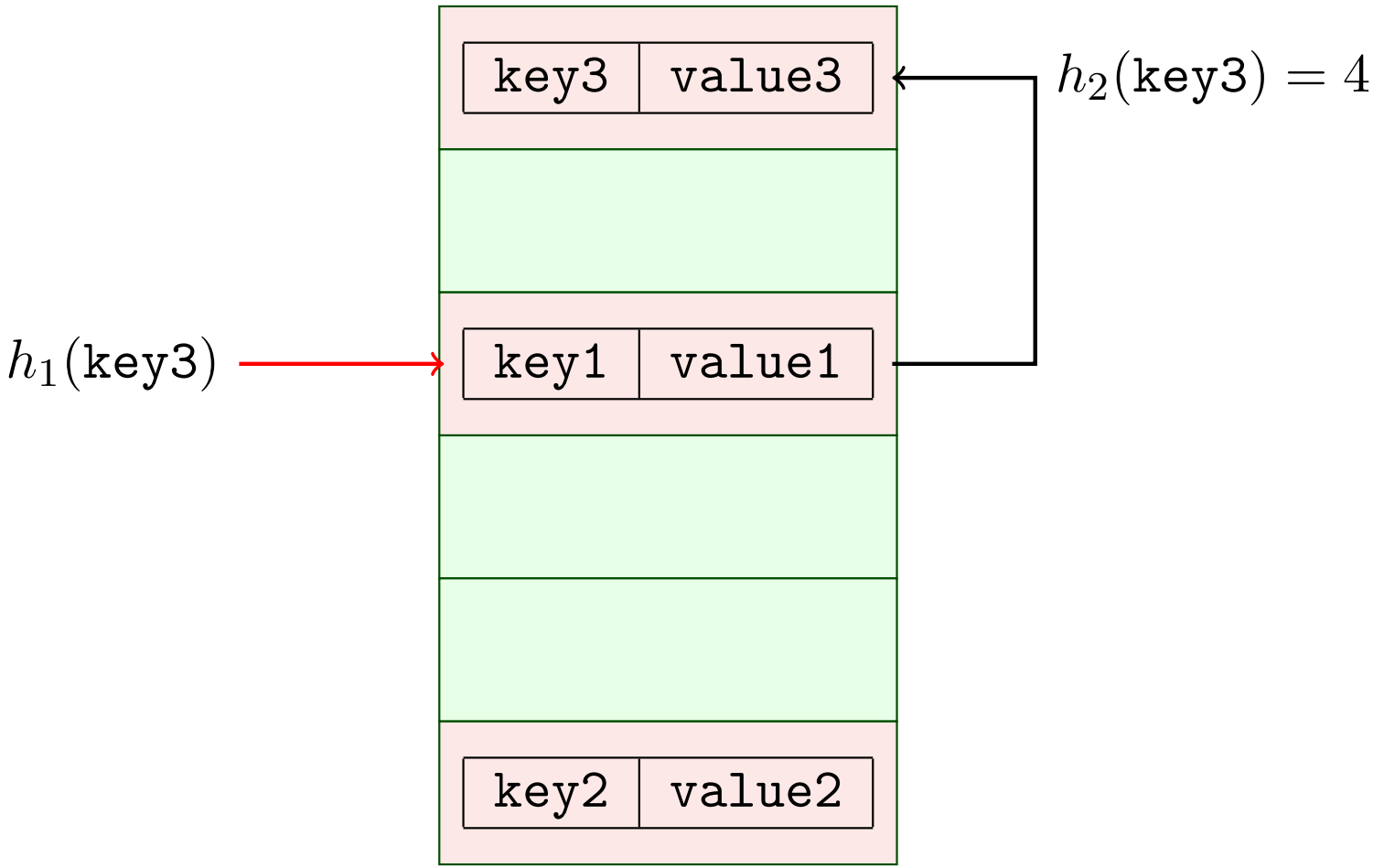
Double hashing



Double hashing



Double hashing



Provided implementation

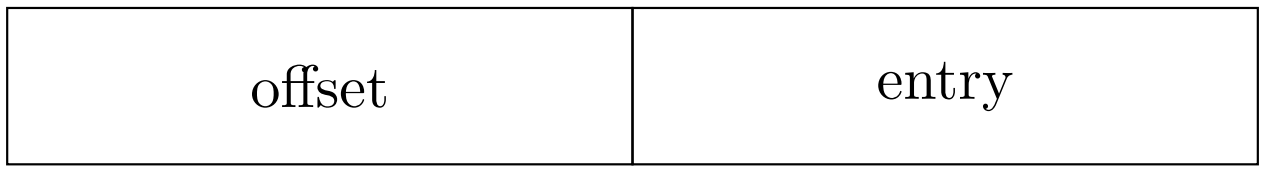
findEmpty, findKey

Provided verification

Outline

Modifications

- 64 bits hashes.



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.

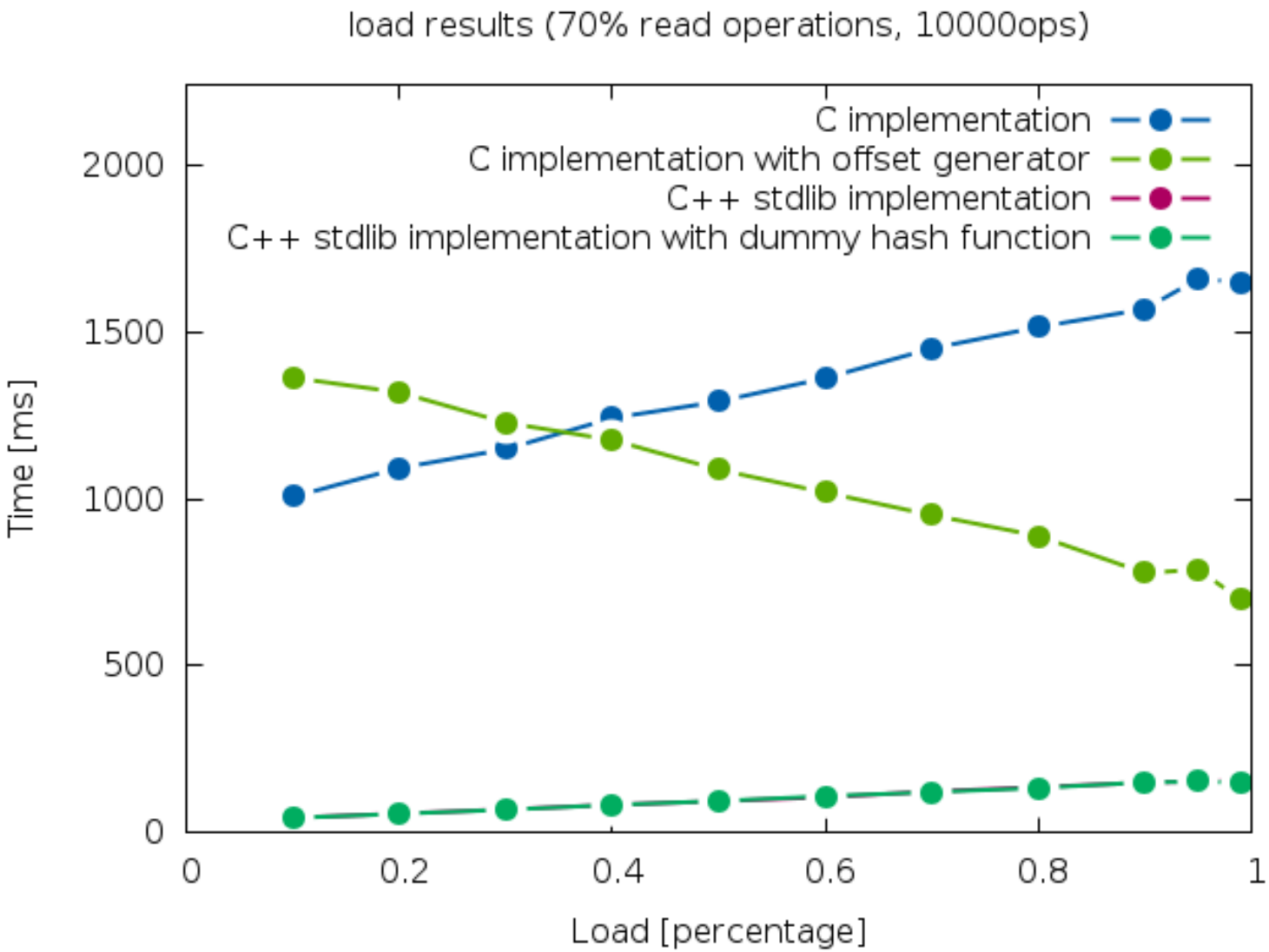
```
test_load.sh length read_ratio load1 [load2...]
```

Evaluation cases

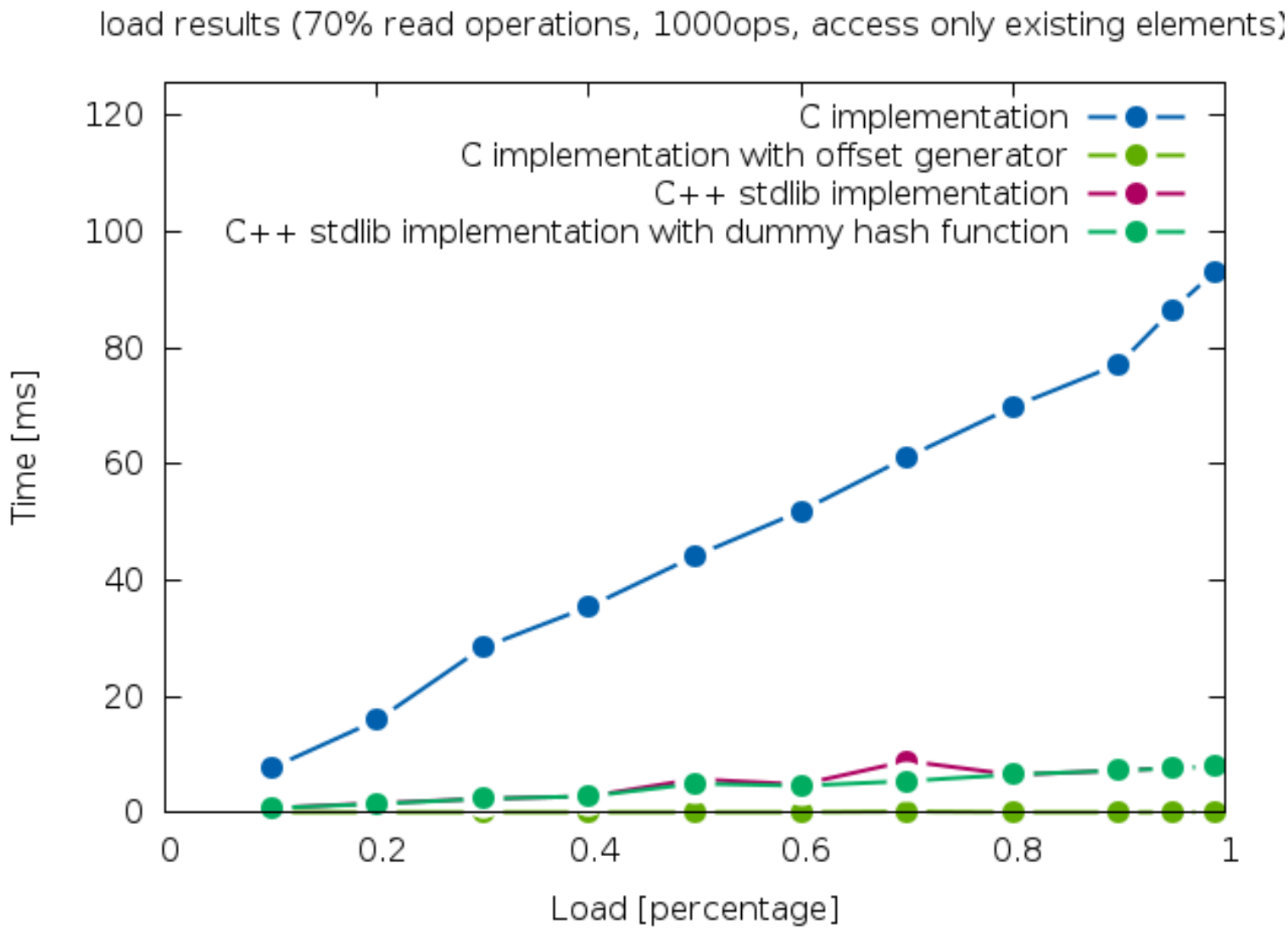
- Worst case: searching a non existing element.

1. Allow searching non existing element.
2. Search only existing element.

Result


load results (70% read operations, 10000ops)
C implementation
C implementation with offset generator
C++ stdlib implementation
C++ stdlib implementation with dummy hash function
Time [ms]
2000
1500
1000
500
0
0
0.2
0.4
0.6
0.8
1
Load [percentage]

Result – only existing



Outline

Outline

Hash-Table software

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

Side effects

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

Q&A