A Tale of Two Hash Tables

Vlad Roubtsov, 2017

A wise Chinese man once said:

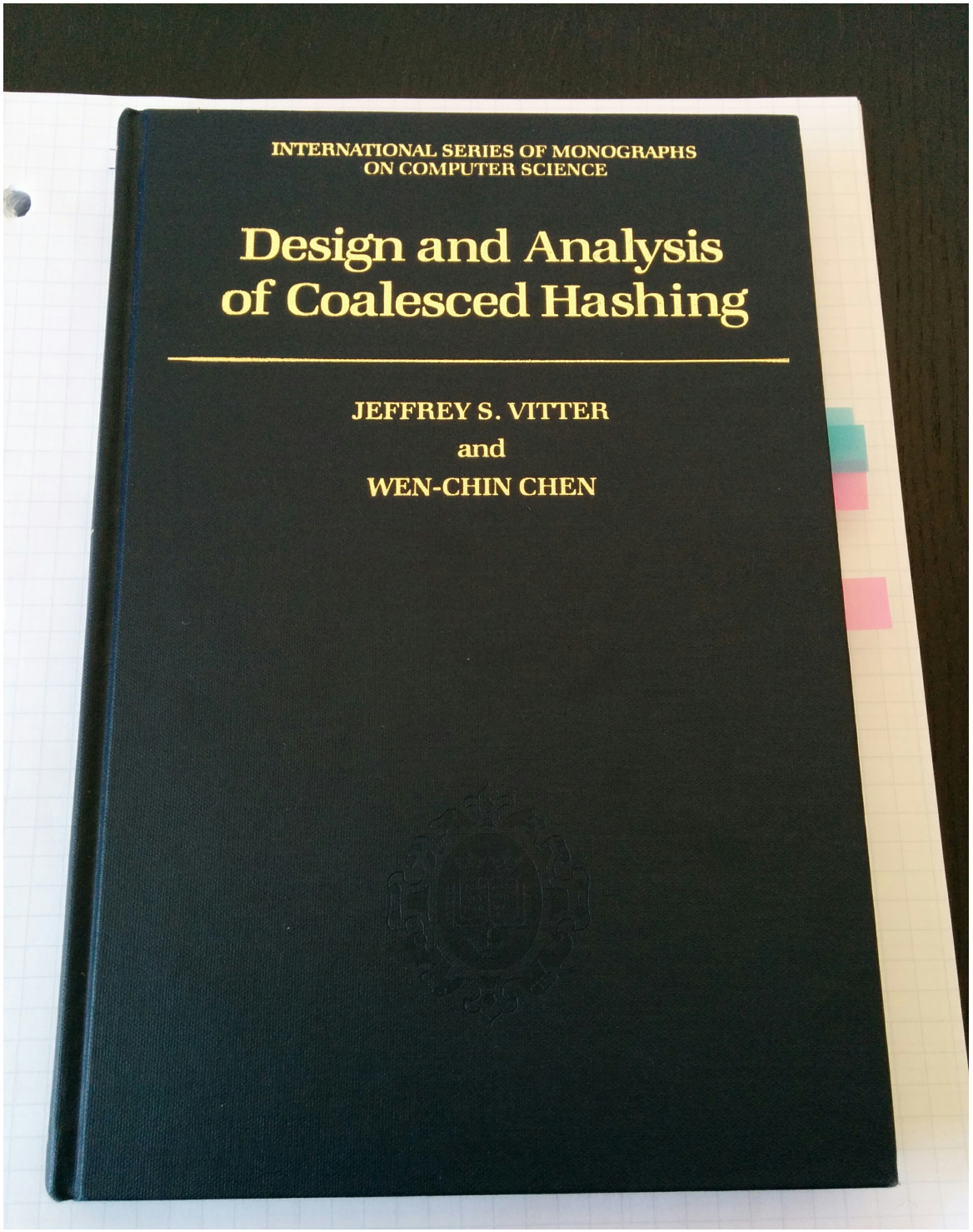
"***Everything is an Array***"

(YY, Tolo Technologies)

Want:

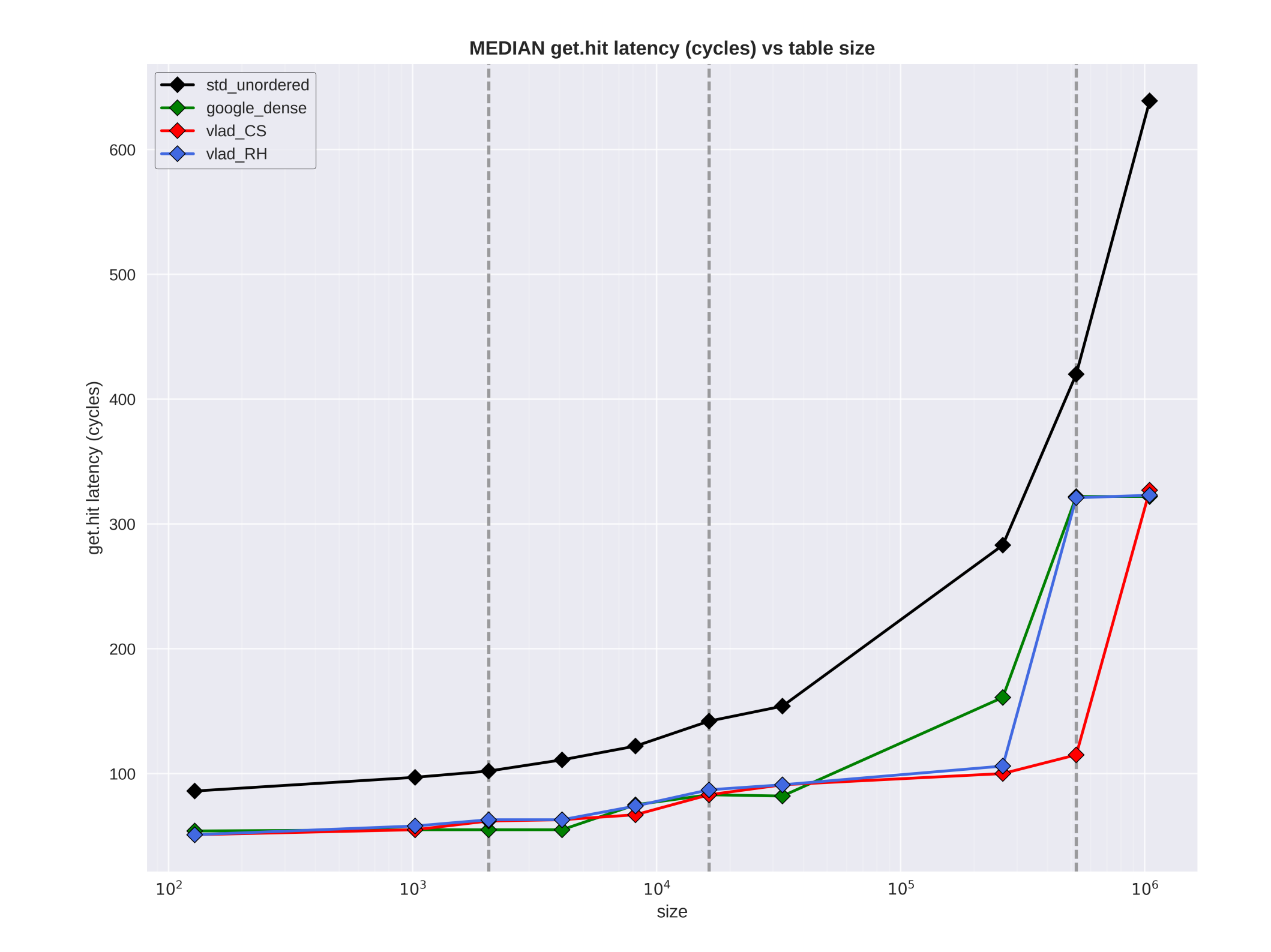
- Want a fast hashtable for data compression, order books, etc
- Want memory compactness
- Want low latency for all ops, ideally all the way to load factors of ~100%
- Try not to require “special” K or V
- std::unordered_map is messed up

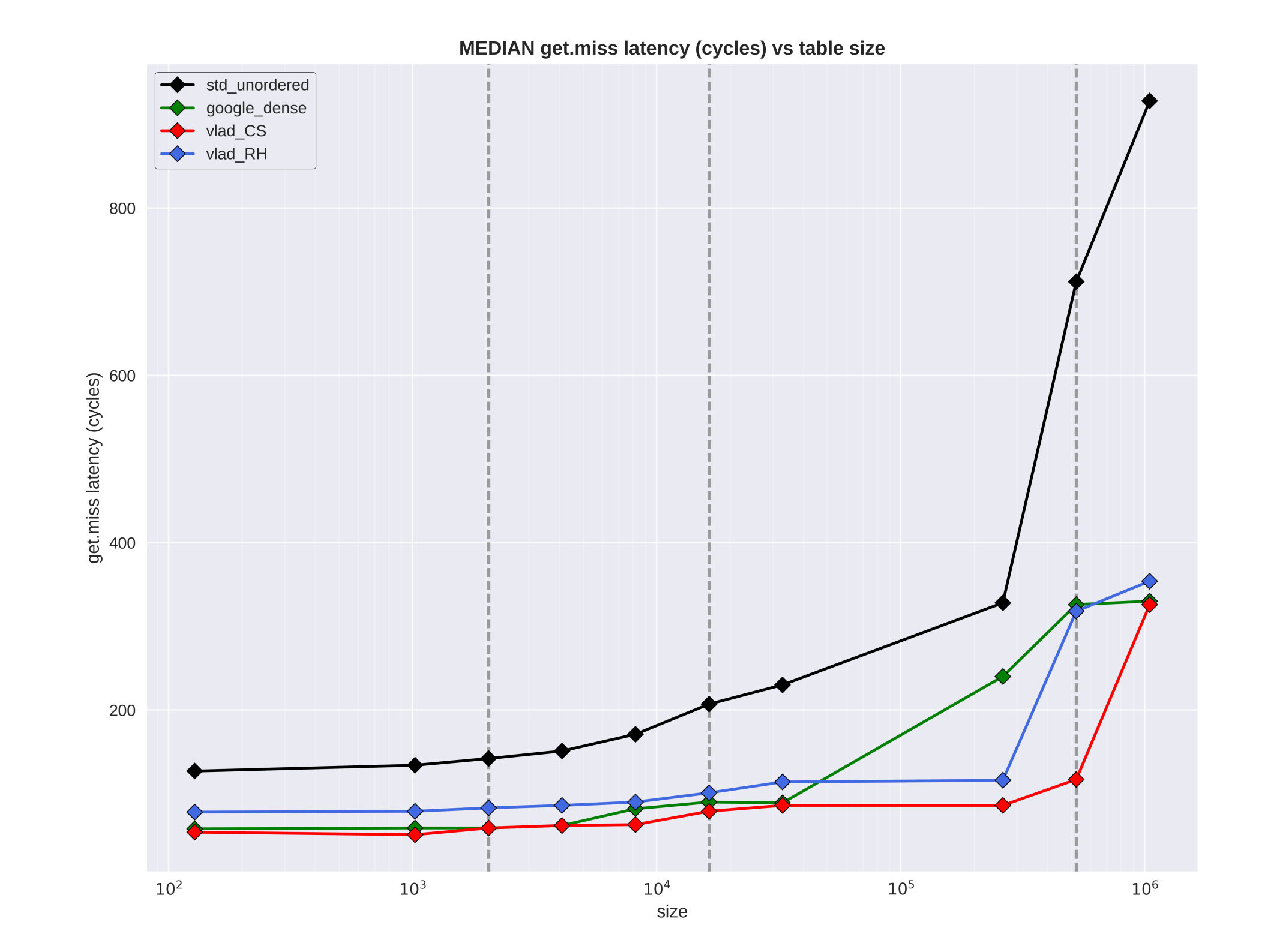
Books have written about (coalesced) hashing...

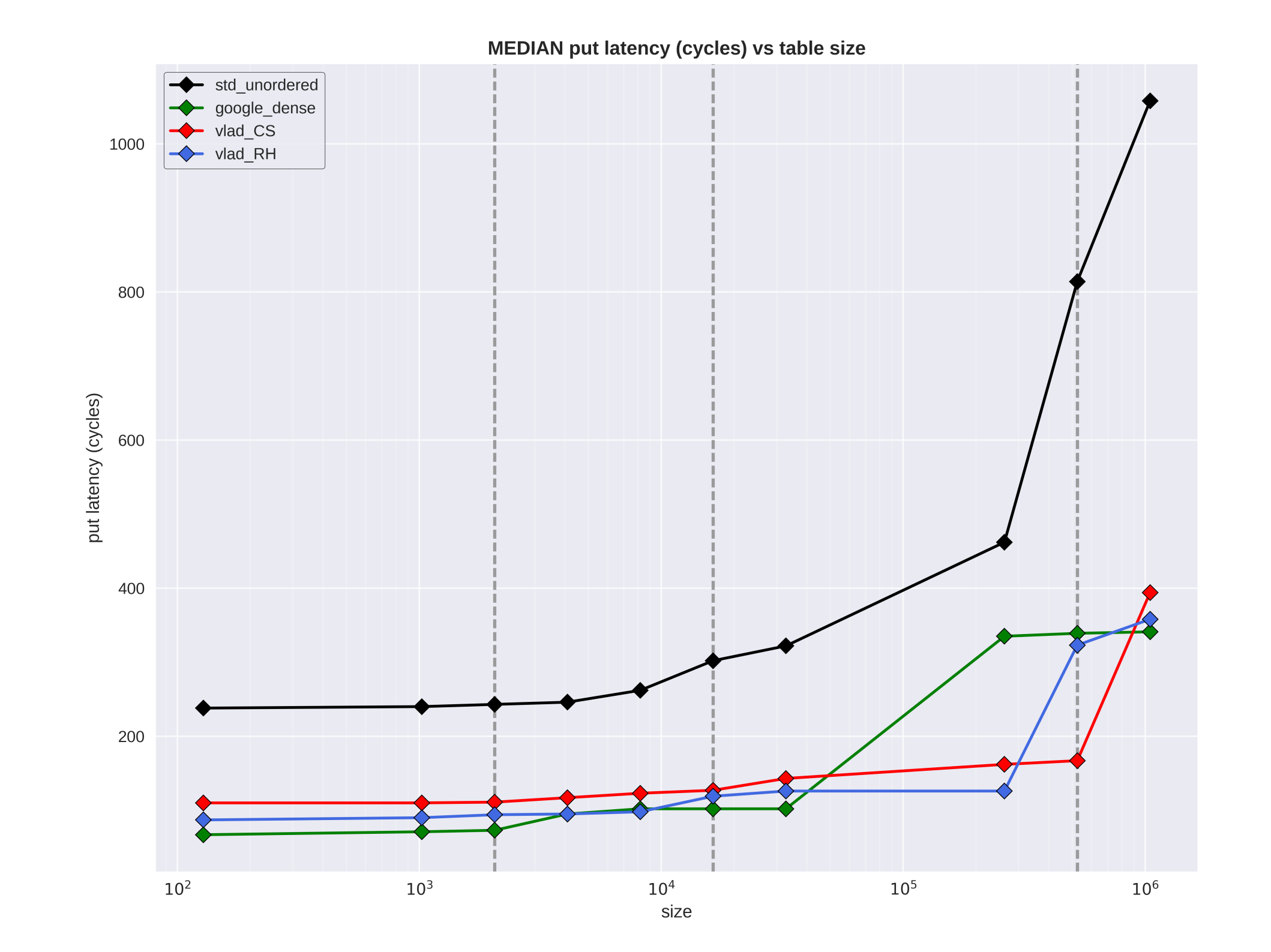

INTERNATIONAL SERIES OF MONOGRAPHS
ON COMPUTER SCIENCE
Design and Analysis
of Coalesced Hashing
JEFFREY S. VITTER
and
WEN-CHIN CHEN

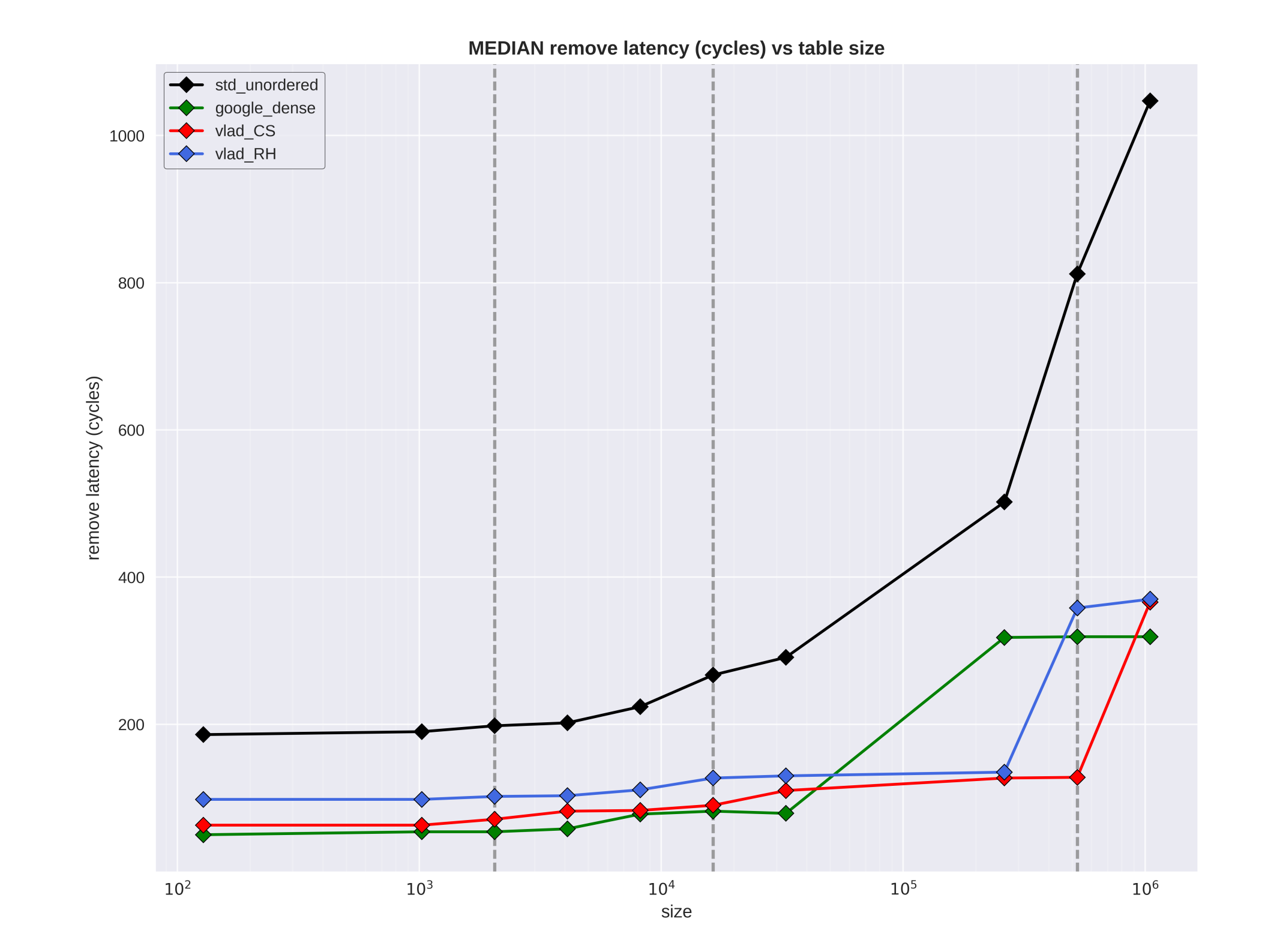
Keywords:

- Coalesced hashing
- Hopscotch hashing
- Brent's variant
- Robin Hood hashing
- Tombstone deletion
- Backshift deletion


MEDIAN get.hit latency (cycles) vs table size
std_unordered
google_dense
vlad_CS
vlad_RH
get.hit latency (cycles)
100
200
300
400
500
600
10^2
10^3
10^4
10^5
10^6
size

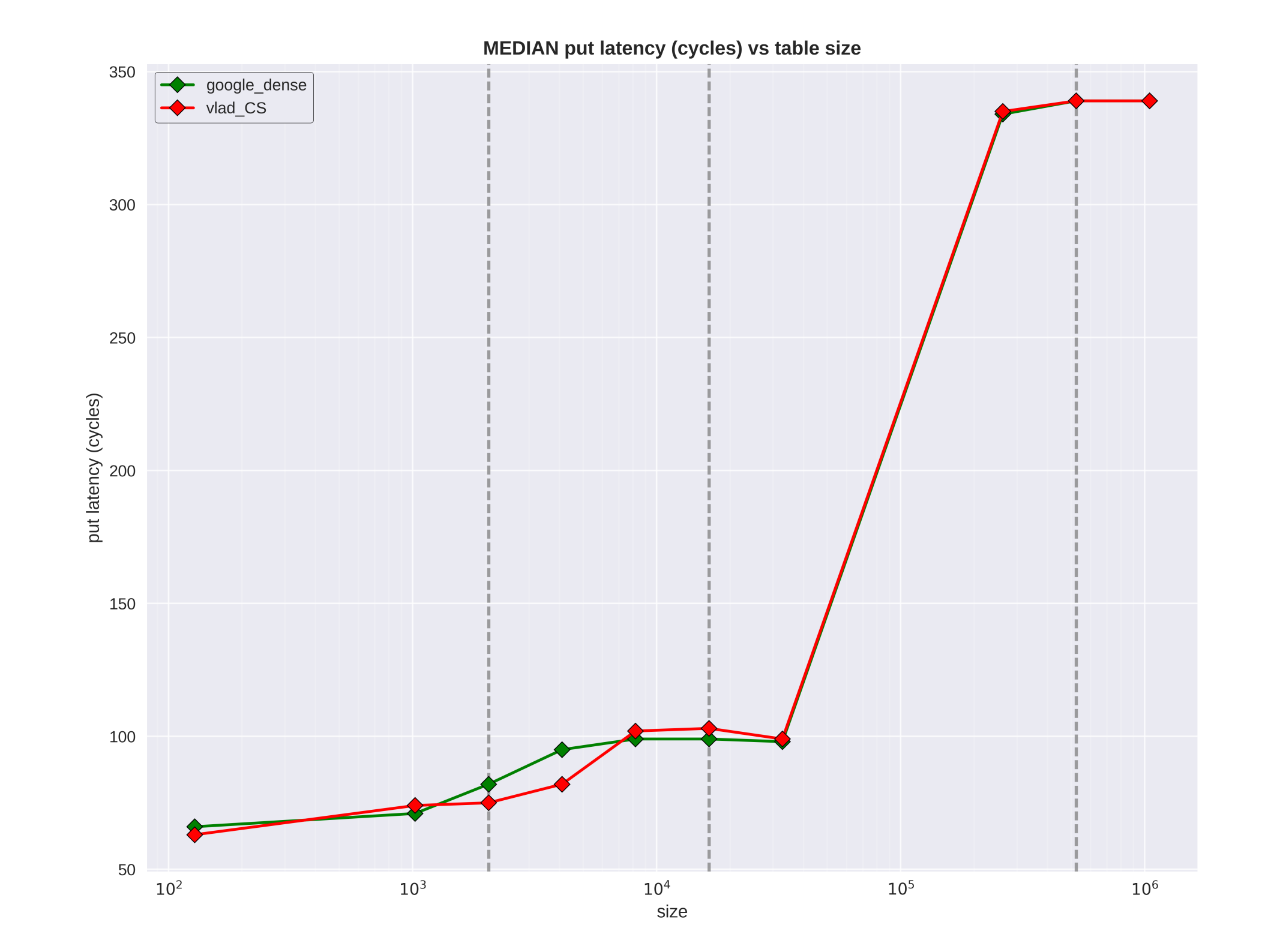

MEDIAN get.miss latency (cycles) vs table size
std_unordered
google_dense
vlad_CS
vlad_RH
get.miss latency (cycles)
800
600
400
200
10^2
10^3
10^4
10^5
10^6
size

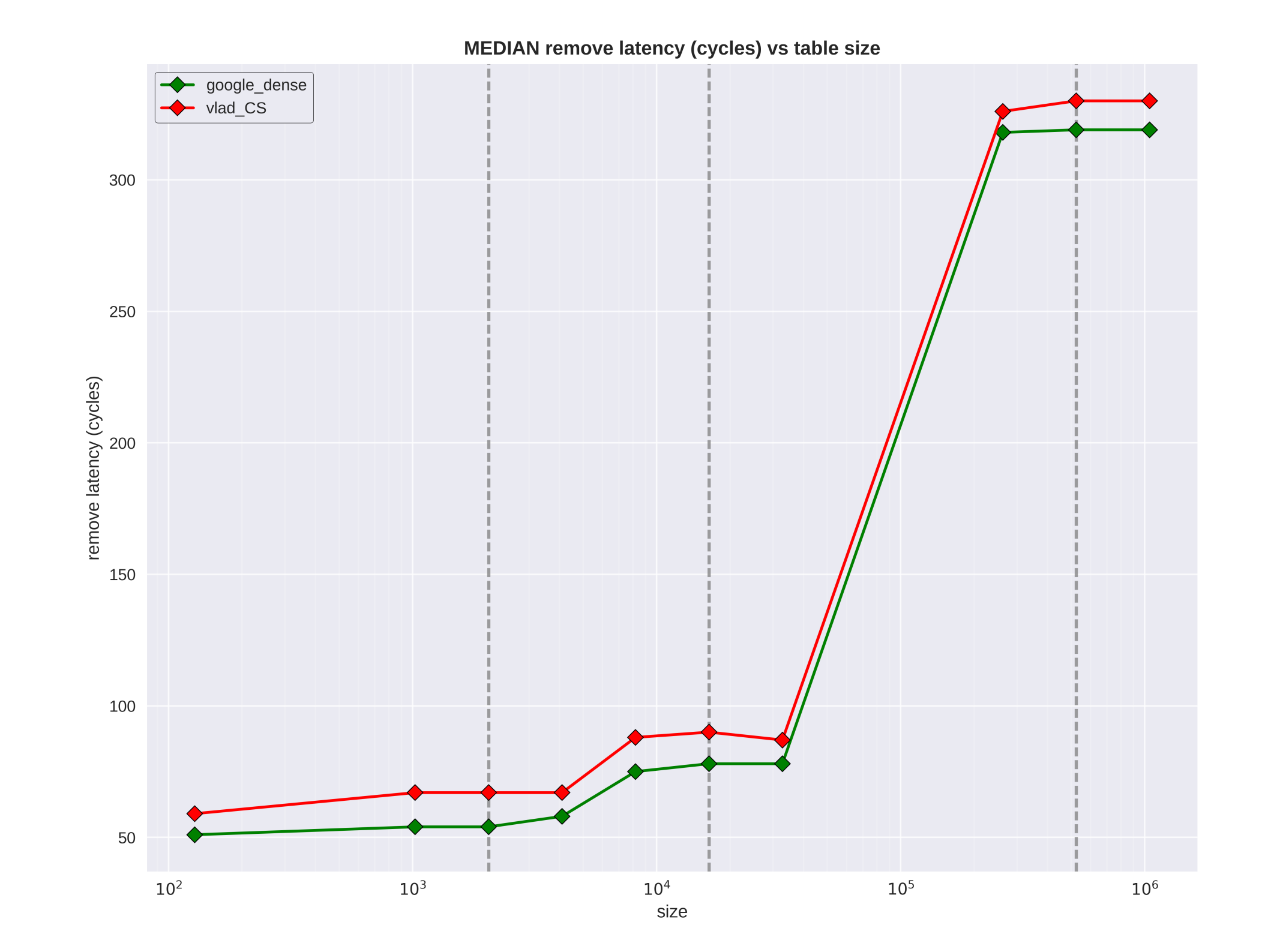

MEDIAN put latency (cycles) vs table size
std_unordered
google_dense
vlad_CS
vlad_RH
put latency (cycles)
1000
800
600
400
200
10^2
10^3
10^4
10^5
10^6
size


MEDIAN remove latency (cycles) vs table size
std_unordered
google_dense
vlad_CS
vlad_RH
remove latency (cycles)
1000
800
600
400
200
10^2
10^3
10^4
10^5
10^6
size

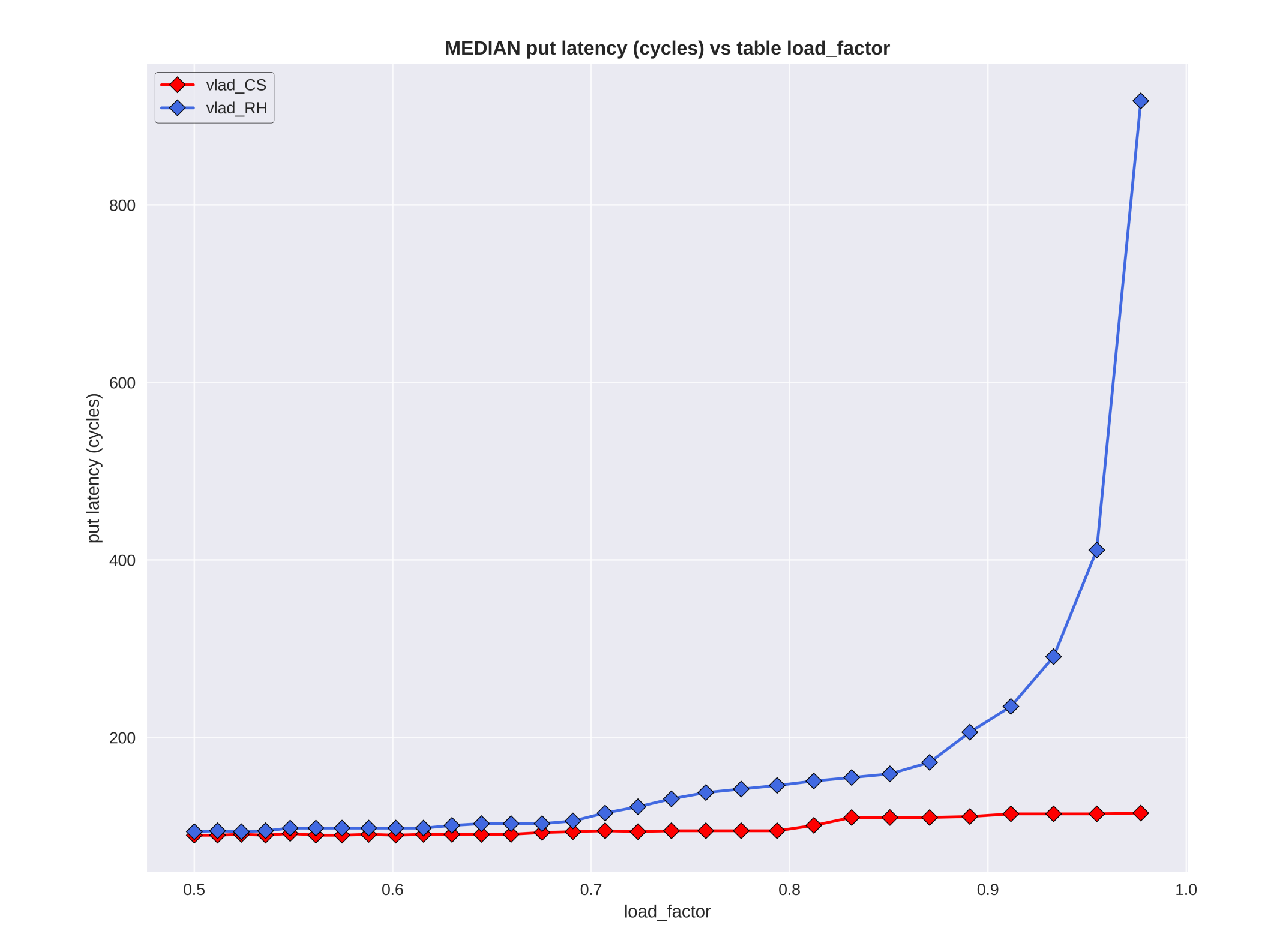
But, but, but! Is google_dense_map cheating?

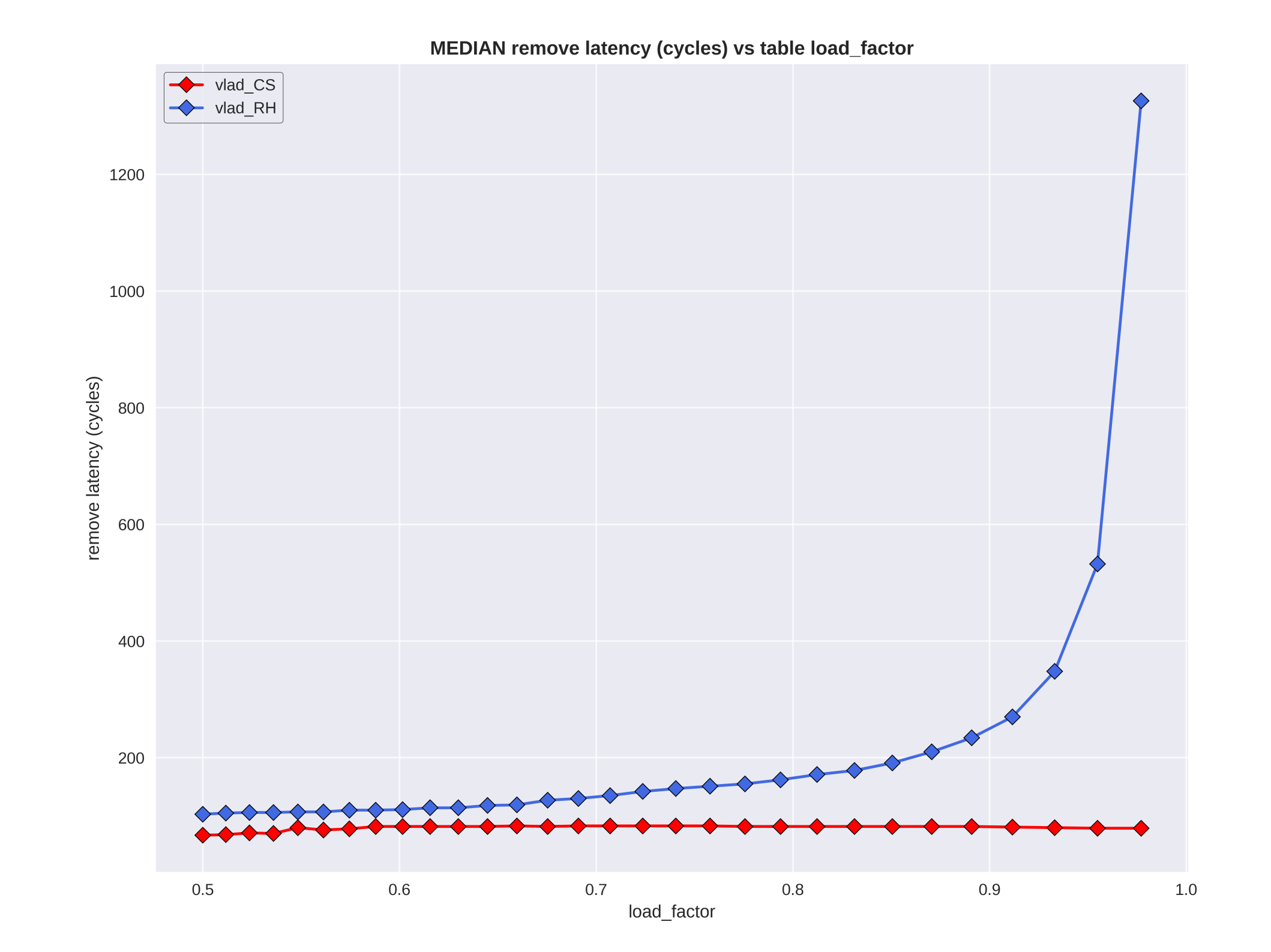
- Given expected size, chosen load factors:
  - std::unordered_map ~100%
  - google_dense_map **~25%**
  - vlad_CS ~100%
  - vlad_RH ~50%
- Let's try mano a mano comparison that is fair (same load factors):


MEDIAN put latency (cycles) vs table size
google_dense
vlad_CS
put latency (cycles)
size
350
300
250
200
150
100
50
10^2
10^3
10^4
10^5
10^6


MEDIAN remove latency (cycles) vs table size
google_dense
vlad_CS
remove latency (cycles)
300
250
200
150
100
50
10^2
10^3
10^4
10^5
10^6
size

Robin Hood scheme seems to have issues with nearly full tables...


MEDIAN put latency (cycles) vs table load_factor
vlad_CS
vlad_RH
put latency (cycles)
200
400
600
800
0.5
0.6
0.7
0.8
0.9
1.0
load_factor


MEDIAN remove latency (cycles) vs table load_factor
vlad_CS
vlad_RH
remove latency (cycles)
200
400
600
800
1000
1200
0.5
0.6
0.7
0.8
0.9
1.0
load_factor

Lessons:

- Main benefit of **chained scatter** *without* coalescing: simpler and faster deletions
- “Short pointers” and sentinels rock
- Reordering structs for memory compactness rocks
- Power-of-2 sizing still rules unless have a “fast mod” alternative
- Linear probing still sucks
- Back or front shifting not so great in practice – consider tombstones instead