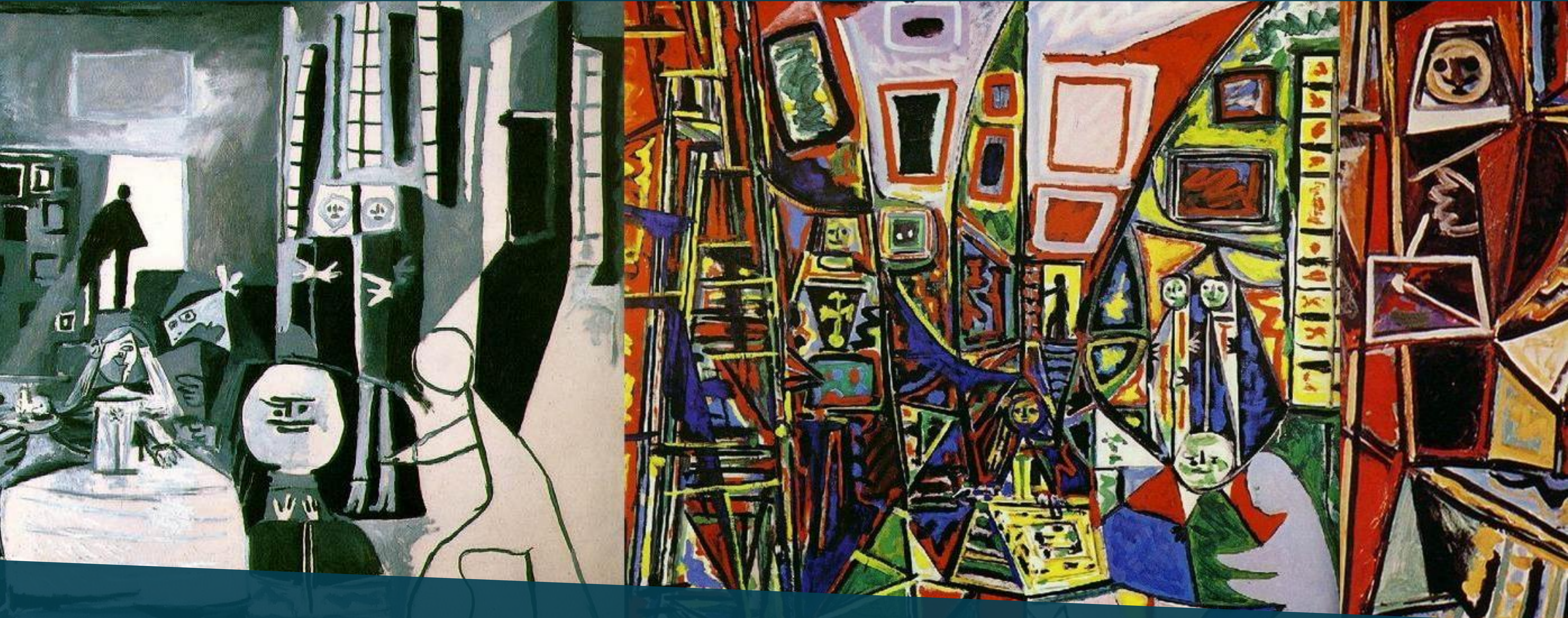
More than a rehash



`using std::cpp 2023`

Joaquín M López Muñoz <joaquin.lopezmunoz@gmail.com>

Madrid, April 2023

It's good to be back

Development Plan for Boost.Unordered

Development Plan for Boost.Unordered

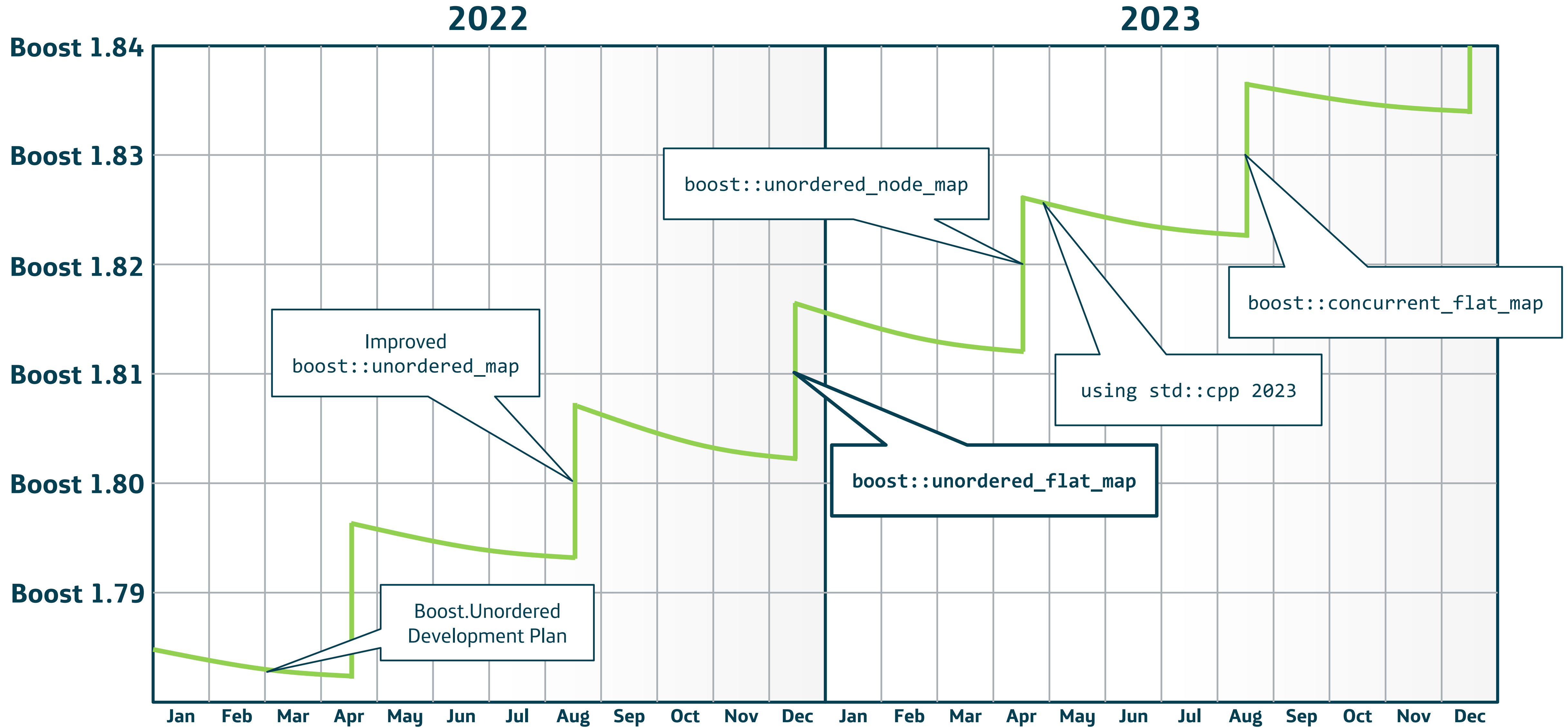
*"**Boost.Unordered** implements the TR1/C++11 unordered containers as proposed by Matt Austern in N1456. Section III.B in the paper explains why **closed addressing** is assumed."*

*"The arguments there were valid at the time the paper was written (2004), but the state of the art has advanced since then, and the hash tables currently in use have chosen **open addressing**."*

*"Therefore, there is room for **adding more containers to Unordered** that implement these additional hash table variations. Our primary aim will be **competitive performance** in each category […]"*

Because it is there

Boost.Unordered timeline



cpplang.slack.com #boost-unordered

Christian Mazakas

Peter Dimov

Sam Darwin

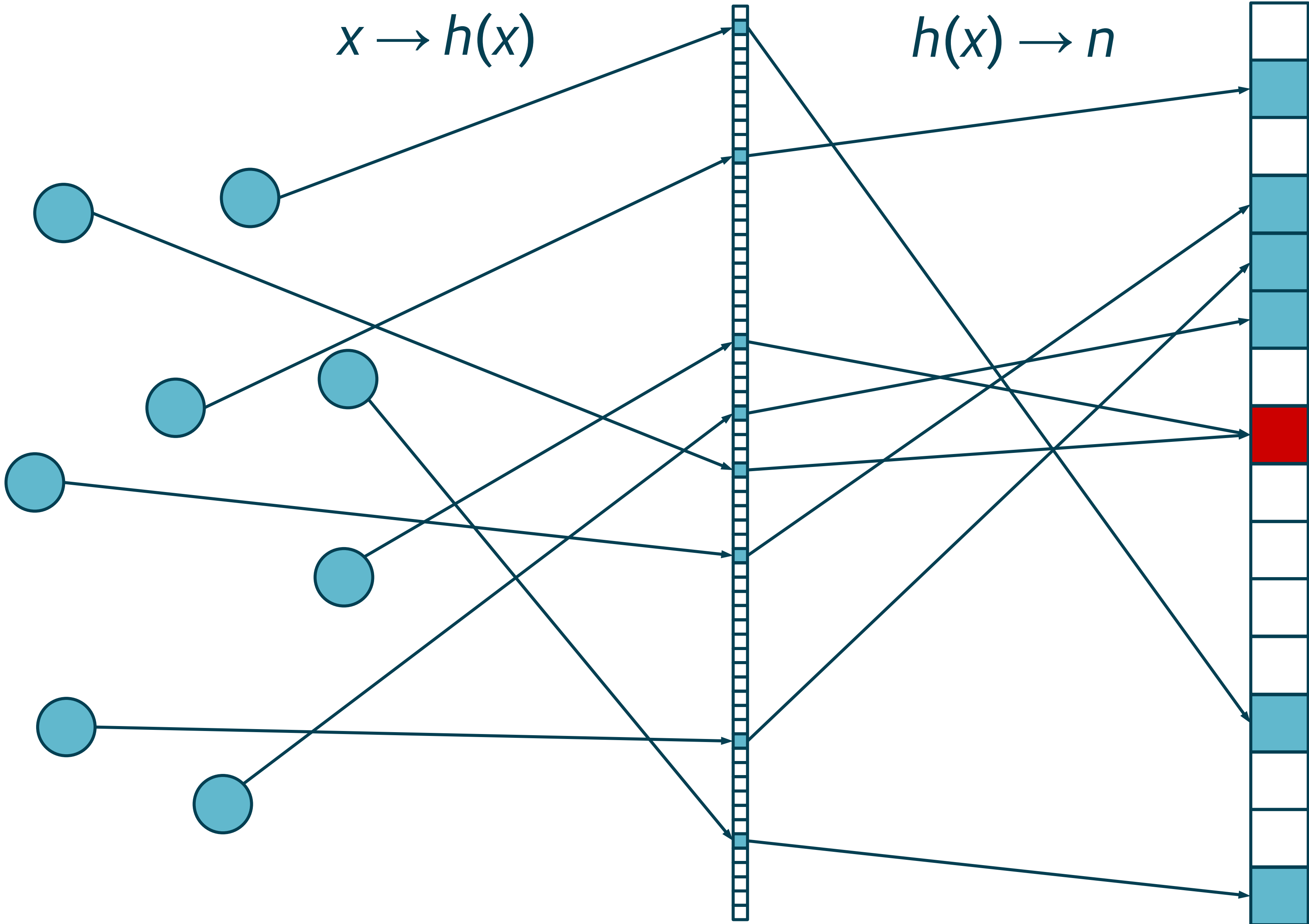
Martin Leitner-Ankerl

Pavel Odintsov

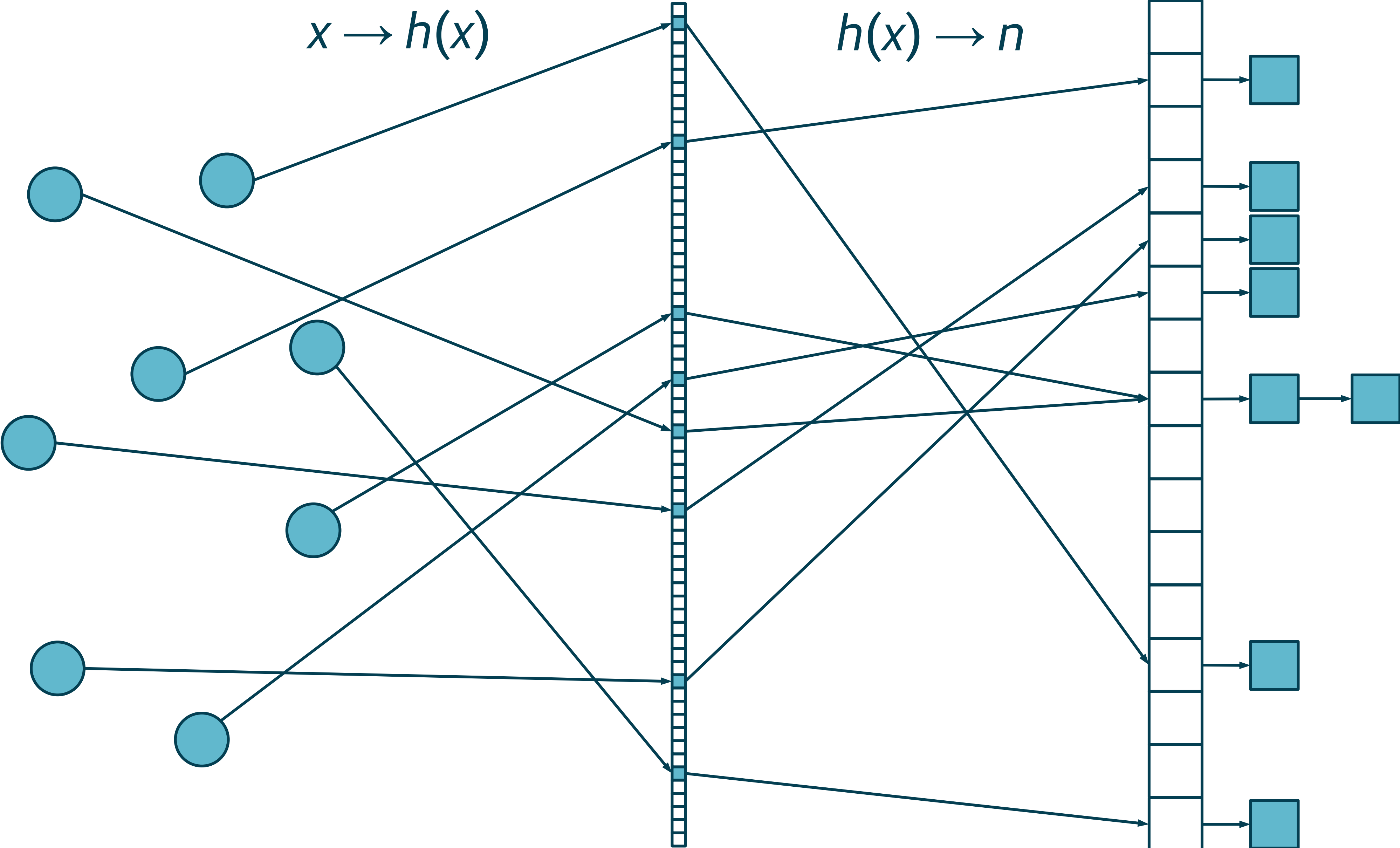
This work is funded by

cppalliance.org

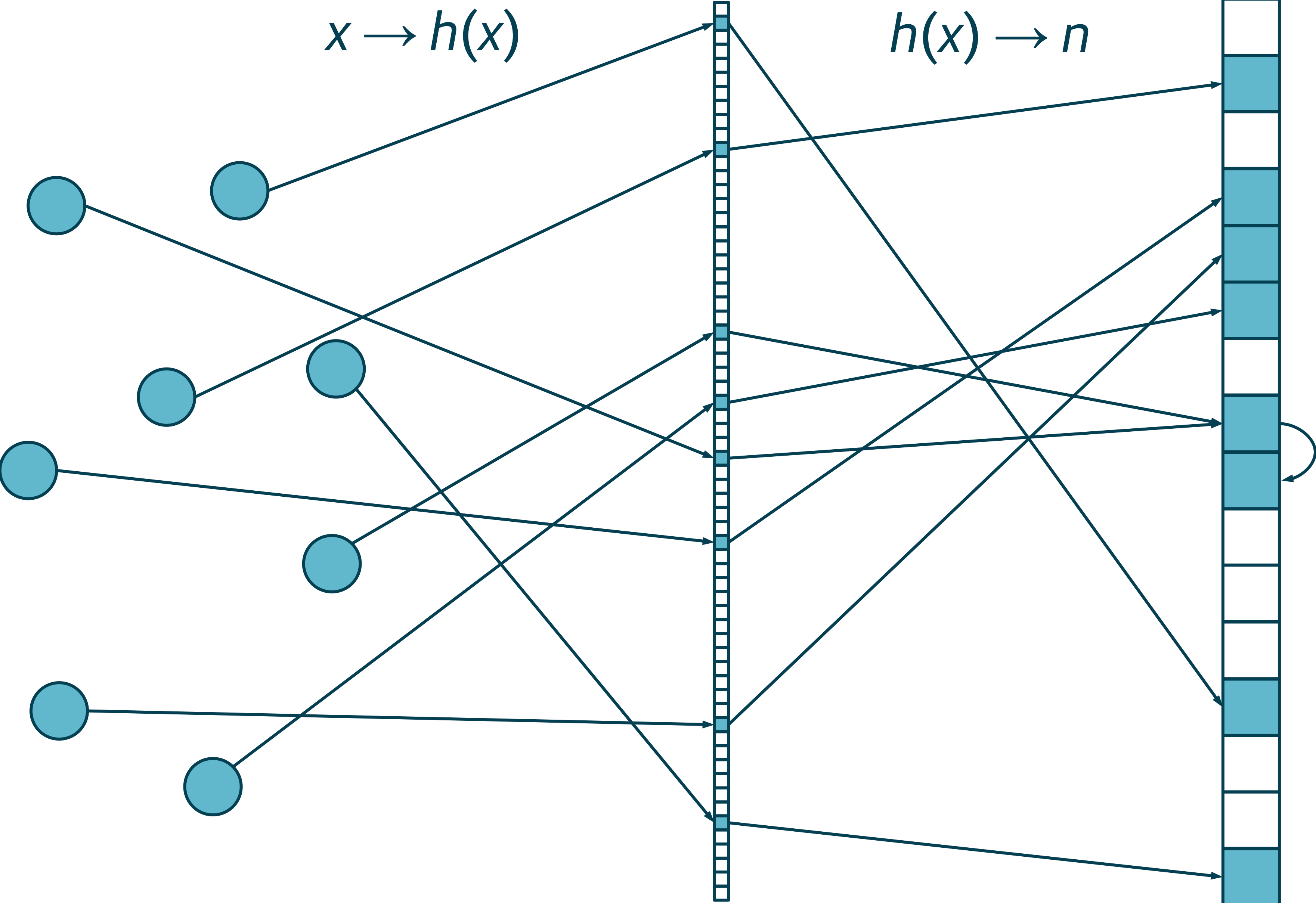
Hash tables in a slide



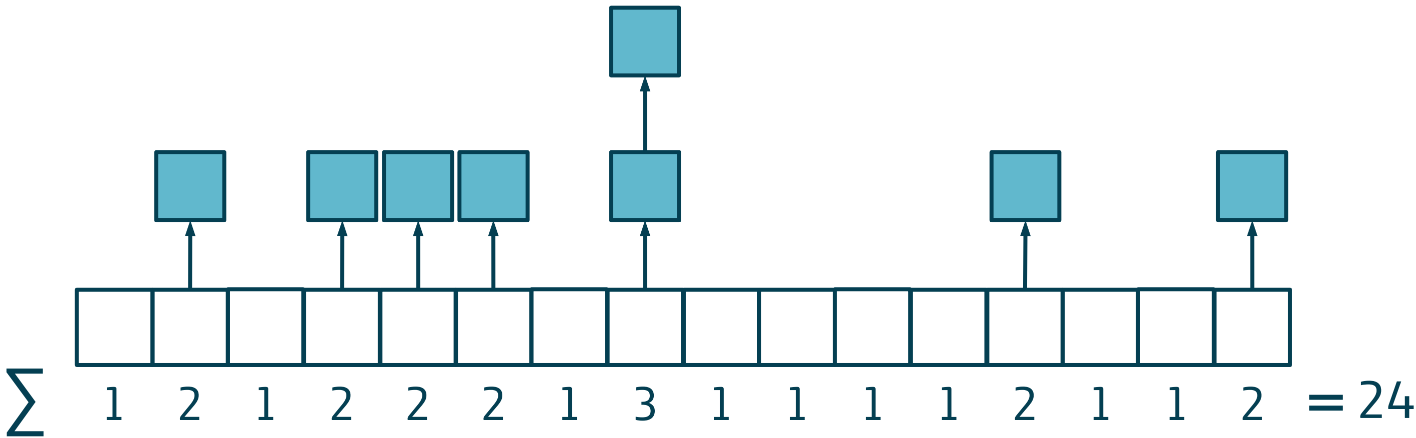
Closed addressing



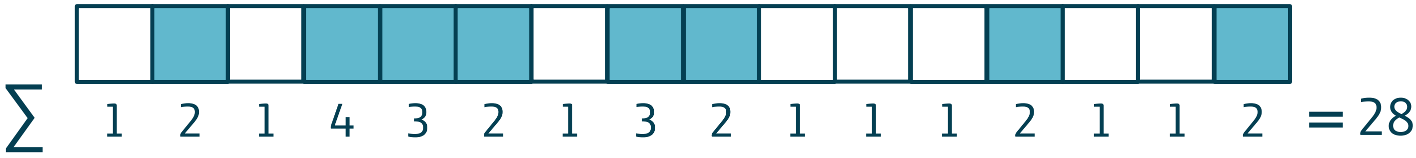
Open addressing



Clustering



$$APL_{unsuccessful} = 24/16 = 1.5$$



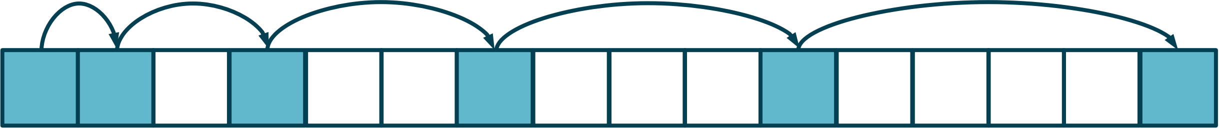
$$APL_{unsuccessful} = 28/16 = 1.75$$

Probing, tombstones and relocations

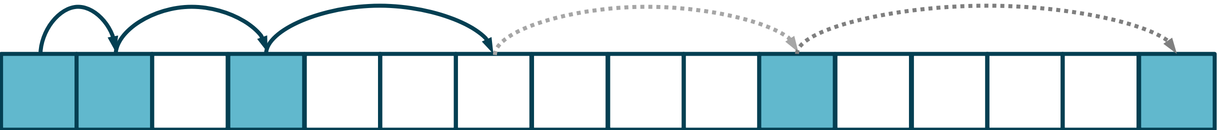
$$\text{APL}_{\text{successful}} = \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$$

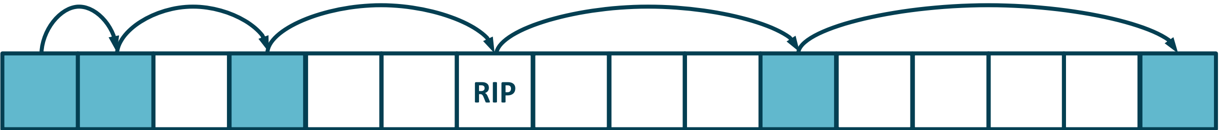
$$\text{APL}_{\text{unsuccessful}} = \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$$

Primary and secondary clustering

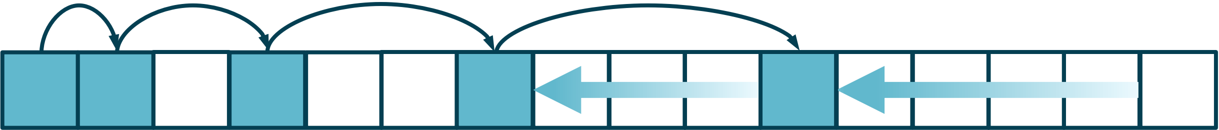


APL better than linear probing
Secondary clustering



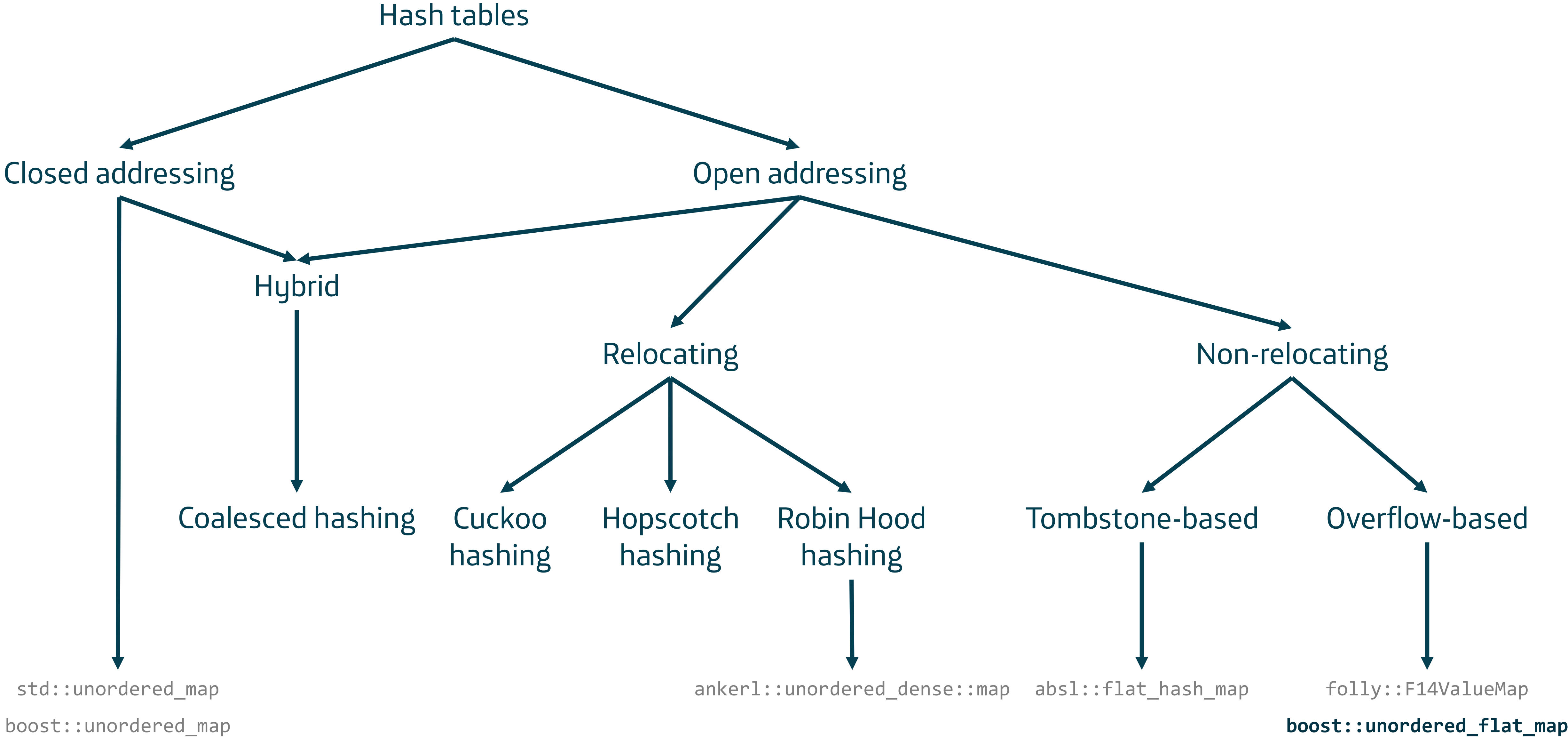


APL never decreases



Several algorithms, increased complexity

A taxonomy of tables



Closed vs open addressing

Closed addressing

- Pointer stability
- Works with poor-quality hash functions
- Supports high load factors (≥ 1.0)

Open addressing

- No pointer stability (in principle)
- Requires high-quality hash functions
- APL $\to \infty$ as load factor $\to 1.0$

Closed vs open addressing

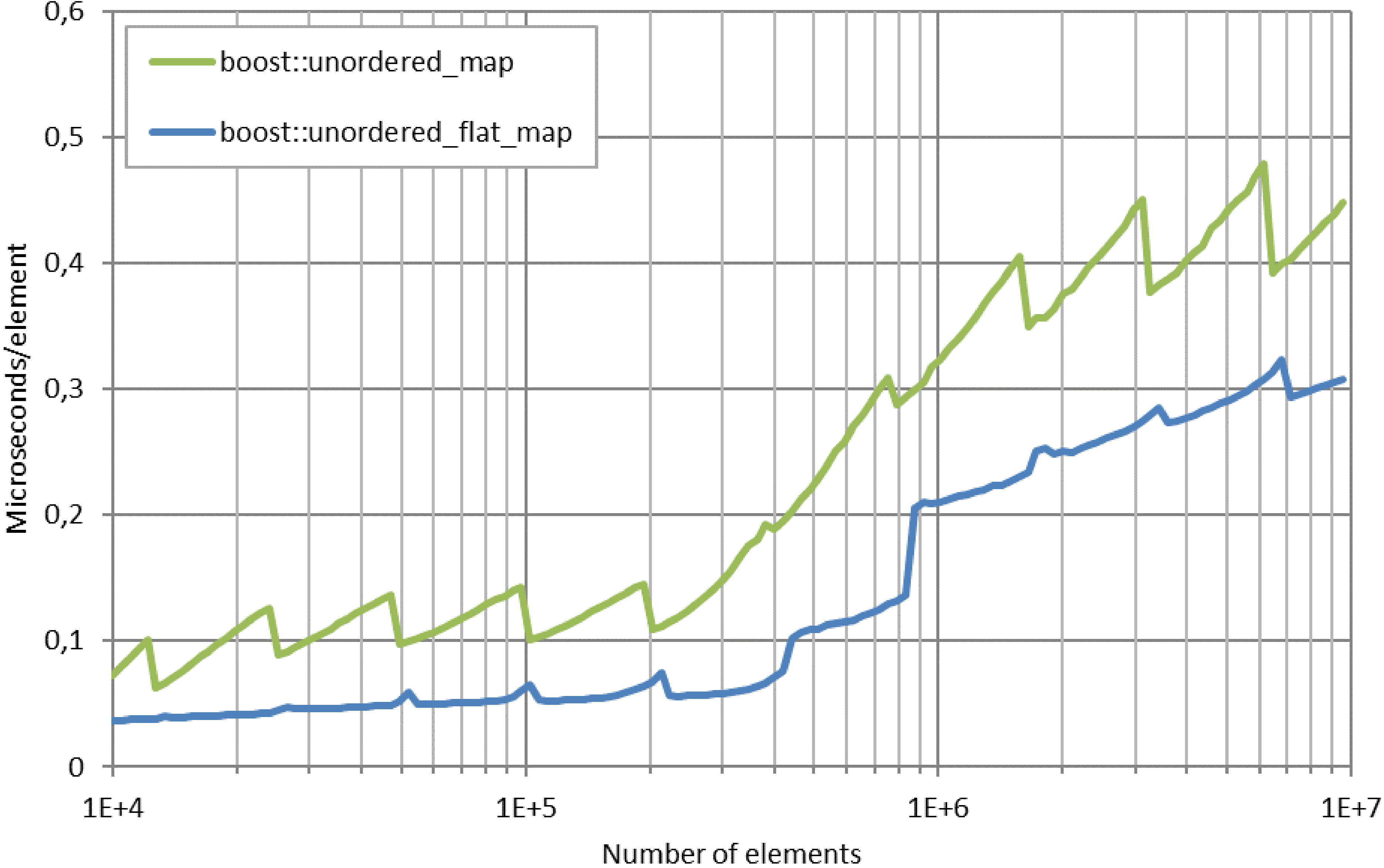
Closed addressing

- Pointer stability
- Works with poor-quality hash functions
- Supports high load factors (≥ 1.0)
- One allocation per node
- Very poor cache locality

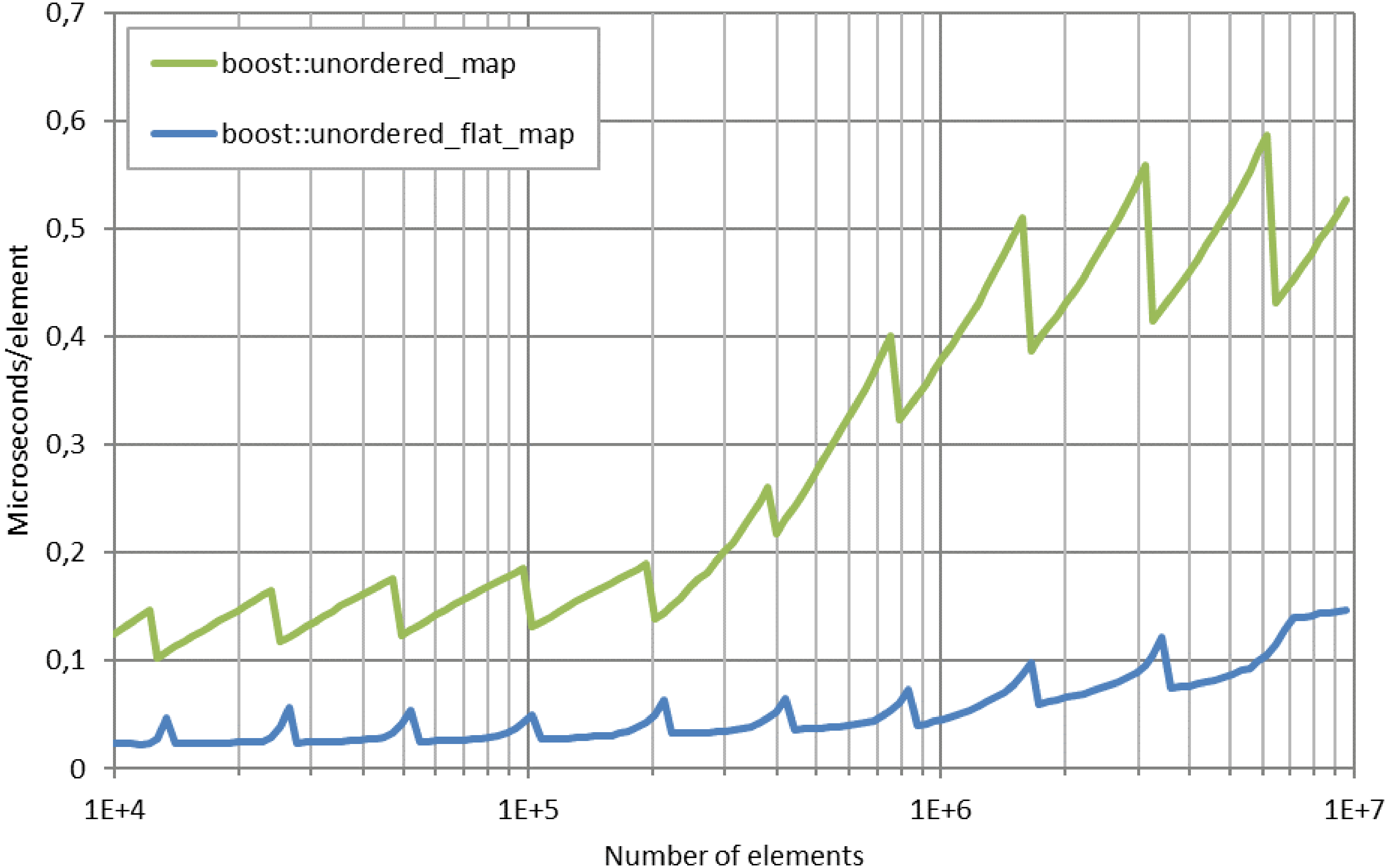
Open addressing

- No pointer stability (in principle)
- Requires high-quality hash functions
- APL $\rightarrow \infty$ as load factor $\rightarrow 1.0$
- Amortized $O(1)$ allocation (in principle)
- Good to very good cache locality

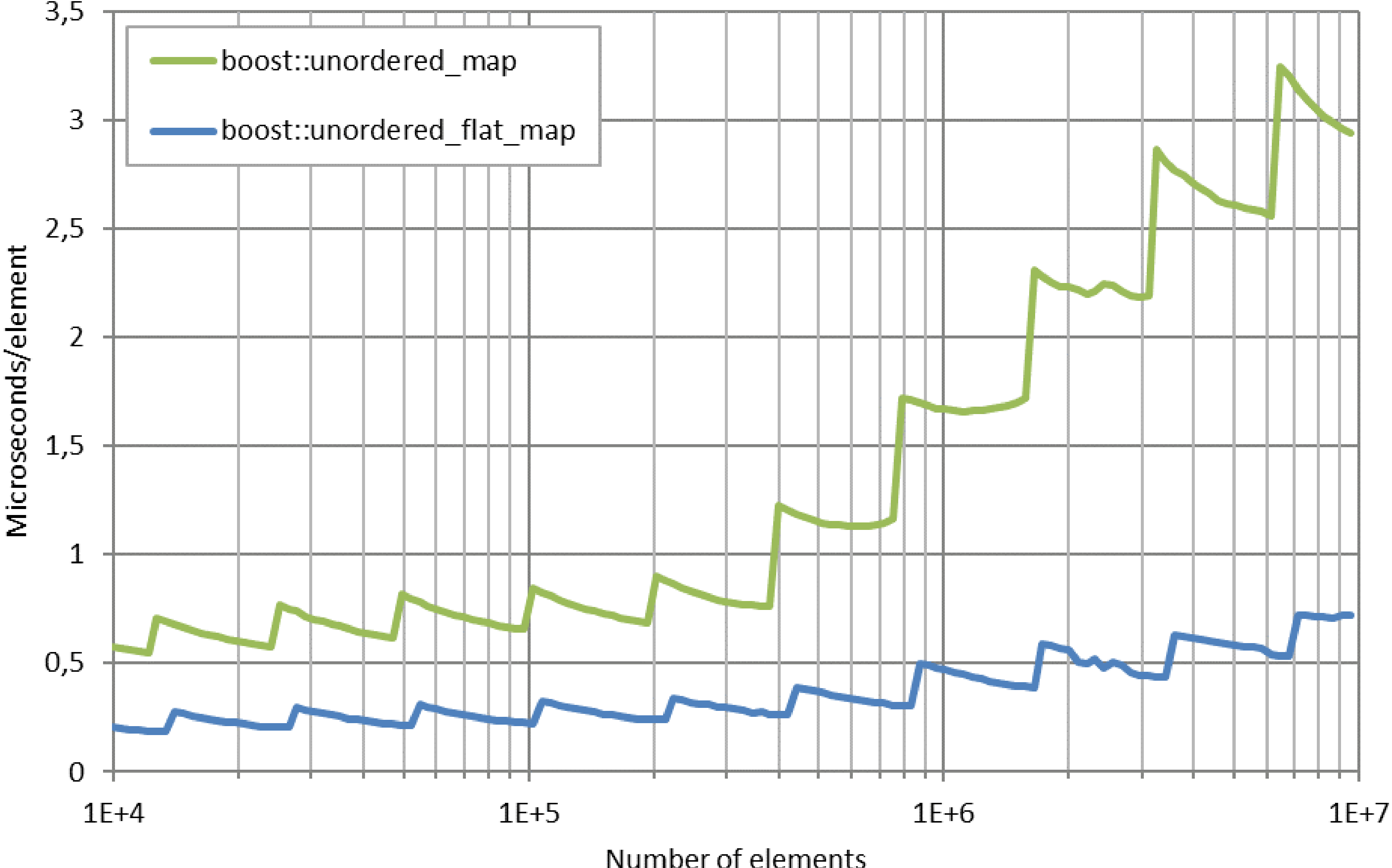
Closed vs open addressing: successful lookup



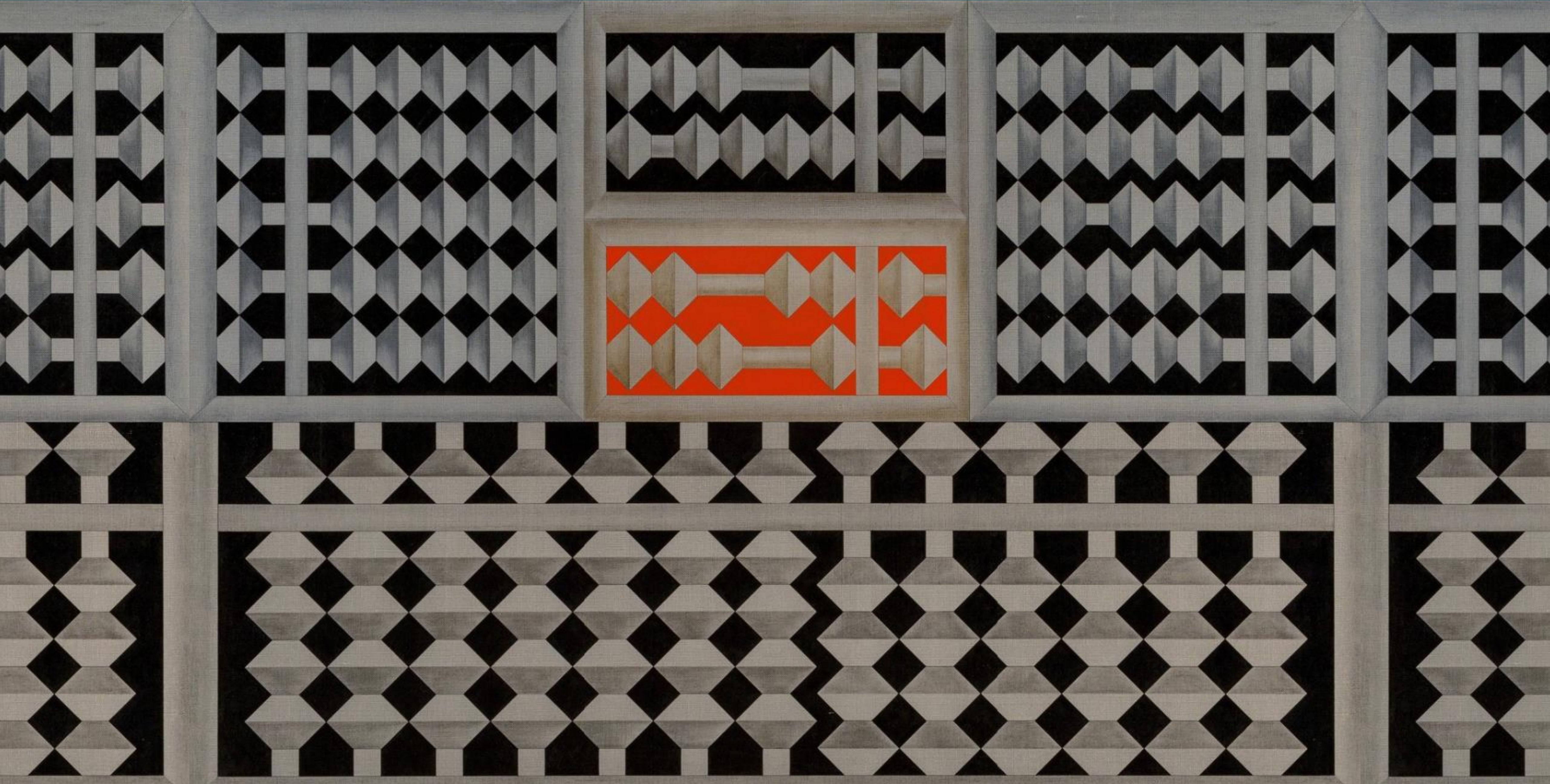
Closed vs open addressing: unsuccessful lookup



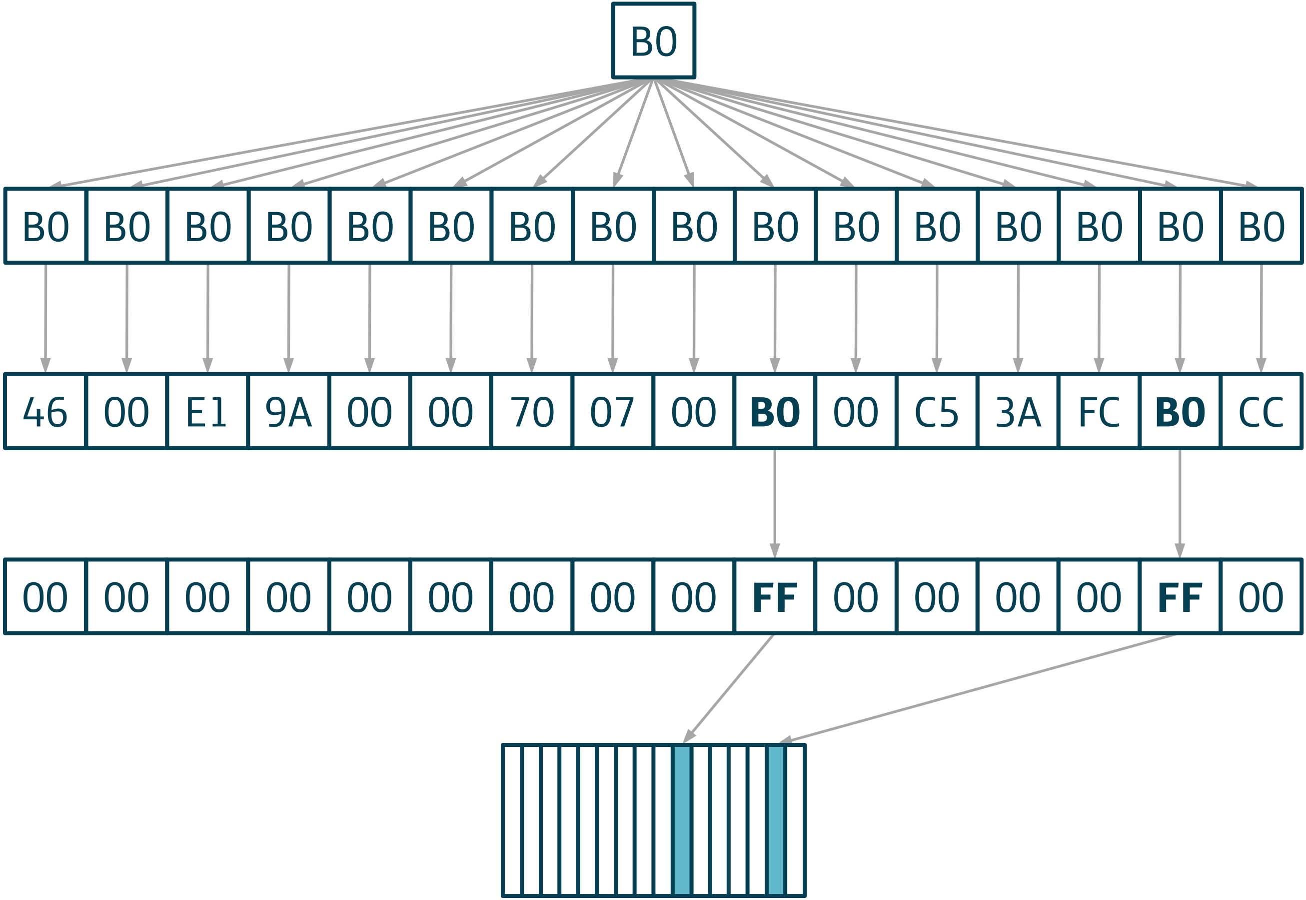
Closed vs open addressing: insertion



Enter SIMD



Parallel reduced hash matching

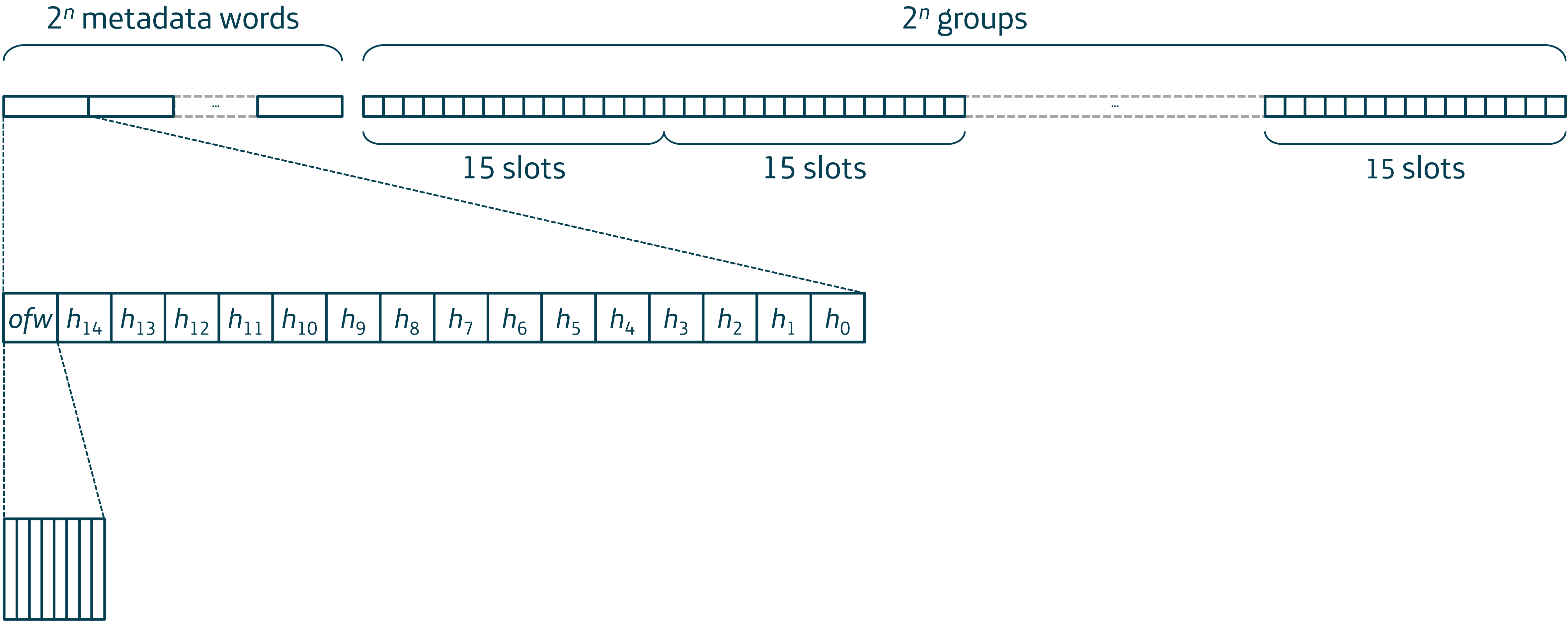


Parallel reduced hash matching

- SIMD and open addressing: match made in heaven
- Slot array + metadata array
- Some metadata values reserved for special markers (empty, tombstone, etc.)
  - $N =$ number of slots compared in one SIMD match operation
  - $b =$ actual reduced hash payload (bits)
  - $E(\#\text{ false positives}) = \dfrac{\alpha N}{2^b}$

Into `boost::unordered_flat_map`

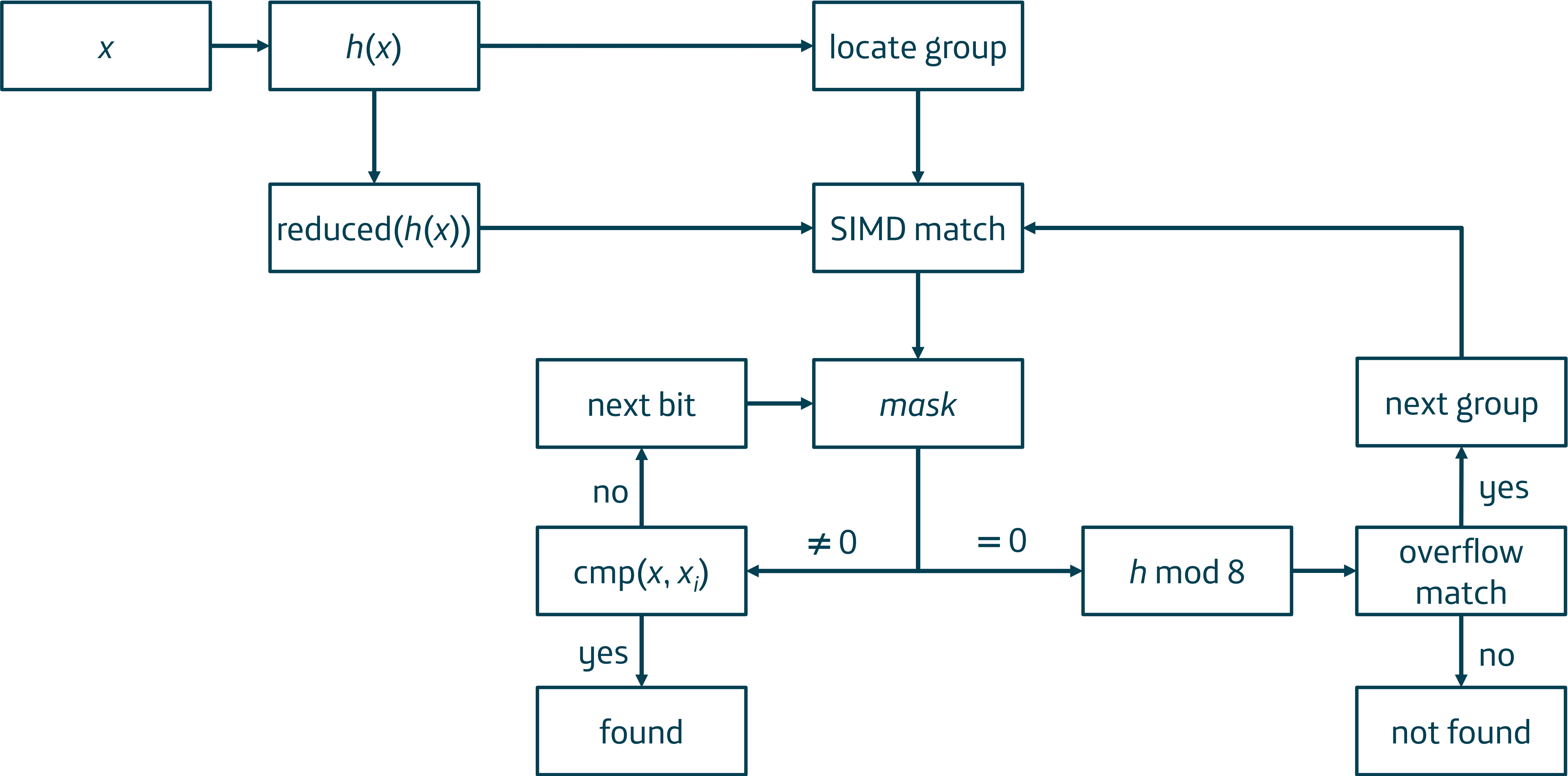
Data layout



Mechanics

- Group-level quadratic probing
  - Intra-group insertion: lowest slot available
- Reduced hash
  - slot empty $h_i \leftarrow 0$
  - sentinel $h_i \leftarrow 1$
  - otherwise $h_i \leftarrow \text{reduced}(h(x)) \in [2, 255]$ (7.989 bits of info)
- Overflow byte
  - x overflowed $\mathit{ofw}[h(x) \bmod 8] \leftarrow 1$

Lookup



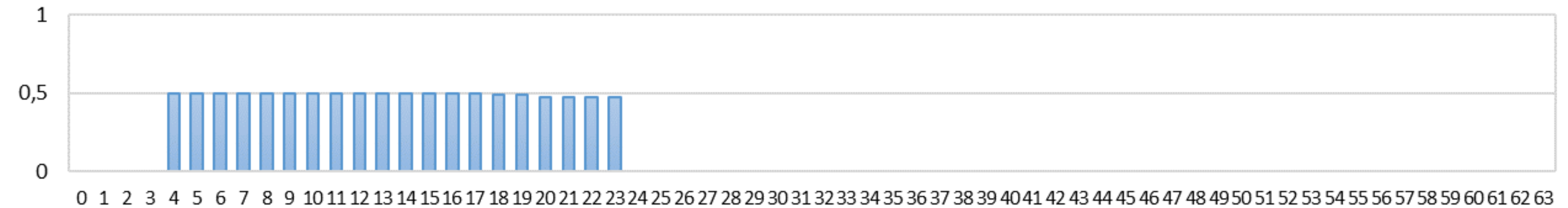
Fighting clustering

- $x \rightarrow h(x) \rightarrow h' = \mathrm{mix}(h(x))$

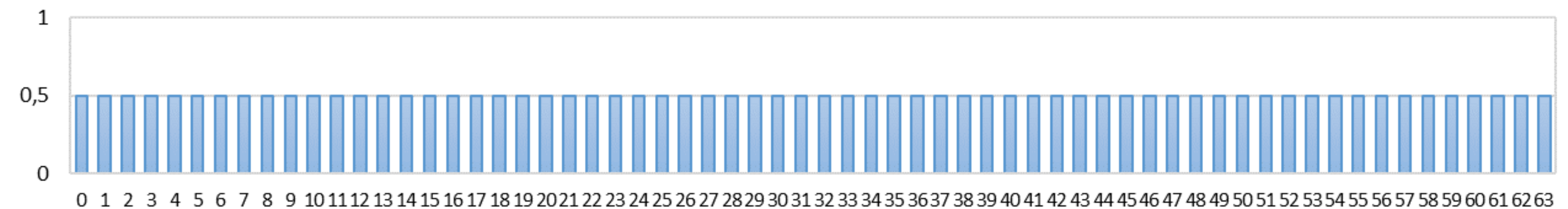
  $a \leftarrow h \;\mathrm{mulx}\; C,\;\; C = \left\lfloor \frac{2^{64}}{\varphi} \right\rfloor, \varphi = \frac{1+\sqrt{5}}{2}$

  $h' \leftarrow \mathrm{high}(a) \;\mathrm{xor}\; \mathrm{low}(a)$

- input $= \{\, 16 \cdot n \mid n = 0, \ldots, 10^6 - 1\}$
  - hash = identity



  - with post-mixing



Competition

`boost::unordered_flat_map` vs `absl::flat_hash_map`

`boost::unordered_flat_map`

- Elements per group: 15
- Probing: quadratic, group-level
- Hash mapping: group-level
- SIMD matching (SSE2, Neon)
- Reduced hash
  - Special values: empty, sentinel
  - Payload: 7.989 bits
- Probe termination: via overflow byte

`absl::flat_hash_map`

- Elements per group: 16
- Probing: quadratic, group-level
- Hash mapping: slot-level
- SIMD matching (SSE2, Neon)
- Reduced hash
  - Special values: empty, sentinel, tombstone
  - Payload: 7 bits
- Probe termination: empty slots in group

Occupancy distribution

$\alpha = 0.4375$

boost

absl

$\alpha = 0.65625$

boost

absl

$\alpha = 0.875$

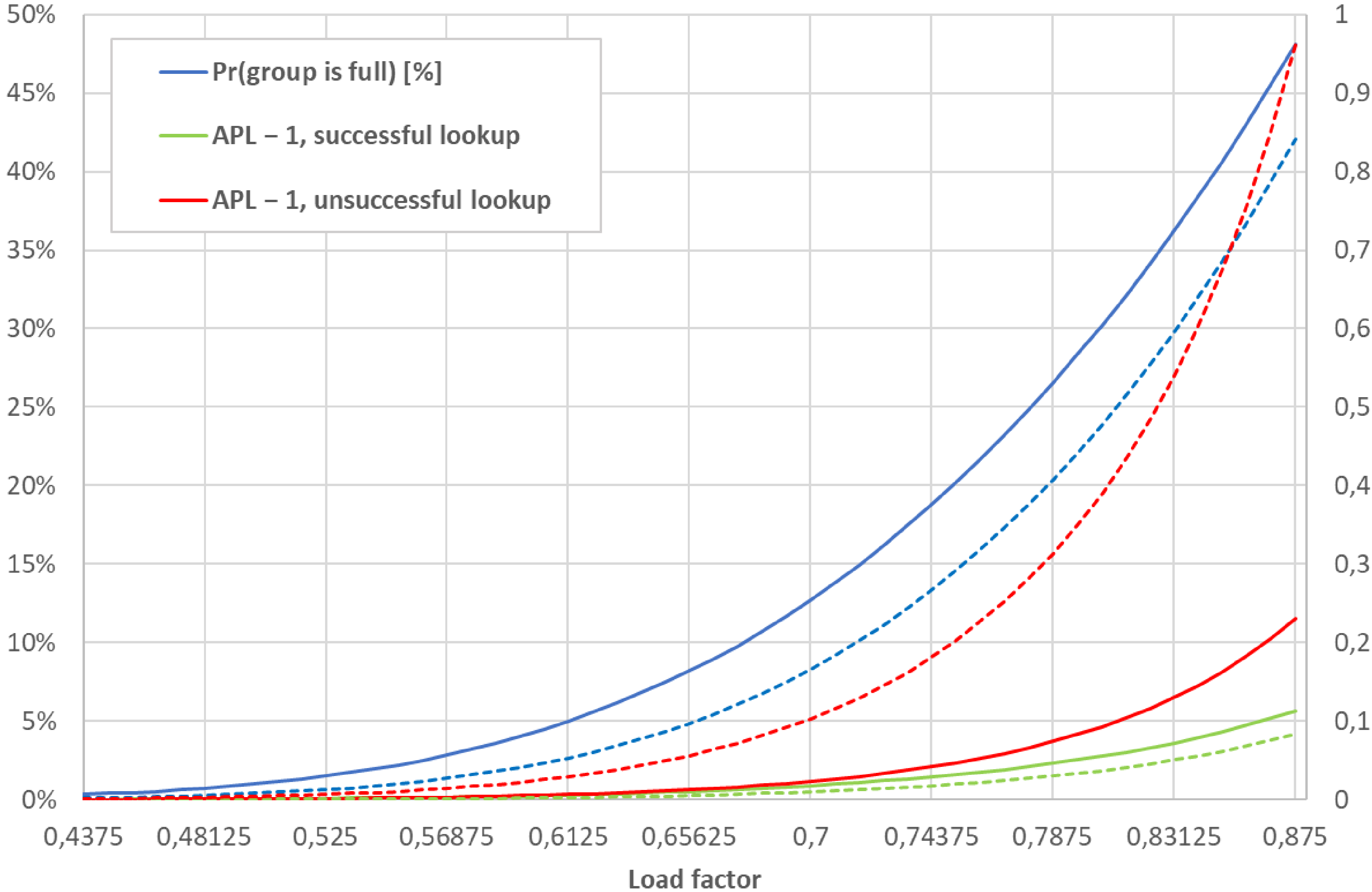
boost

absl

Stats

boost

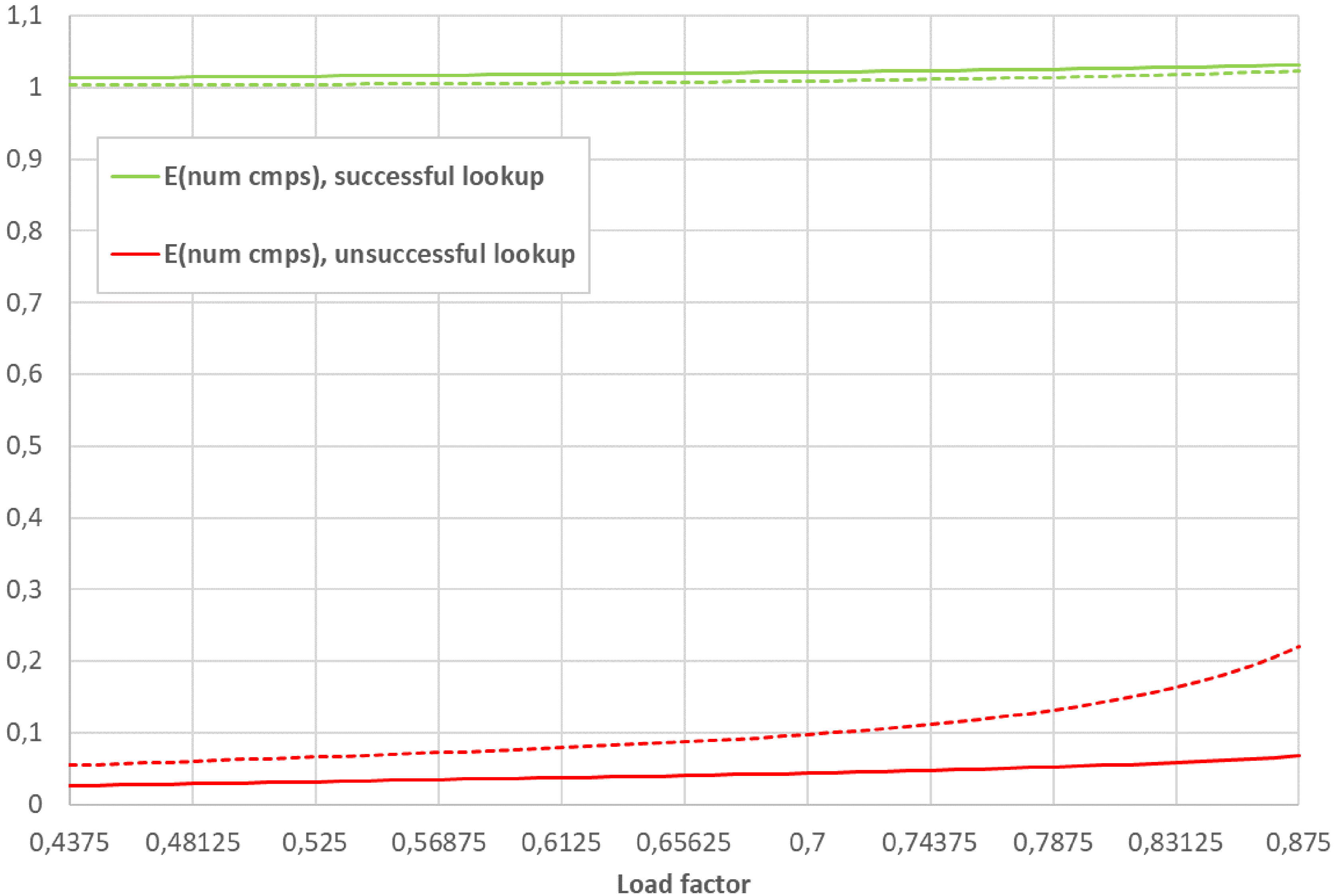
absl



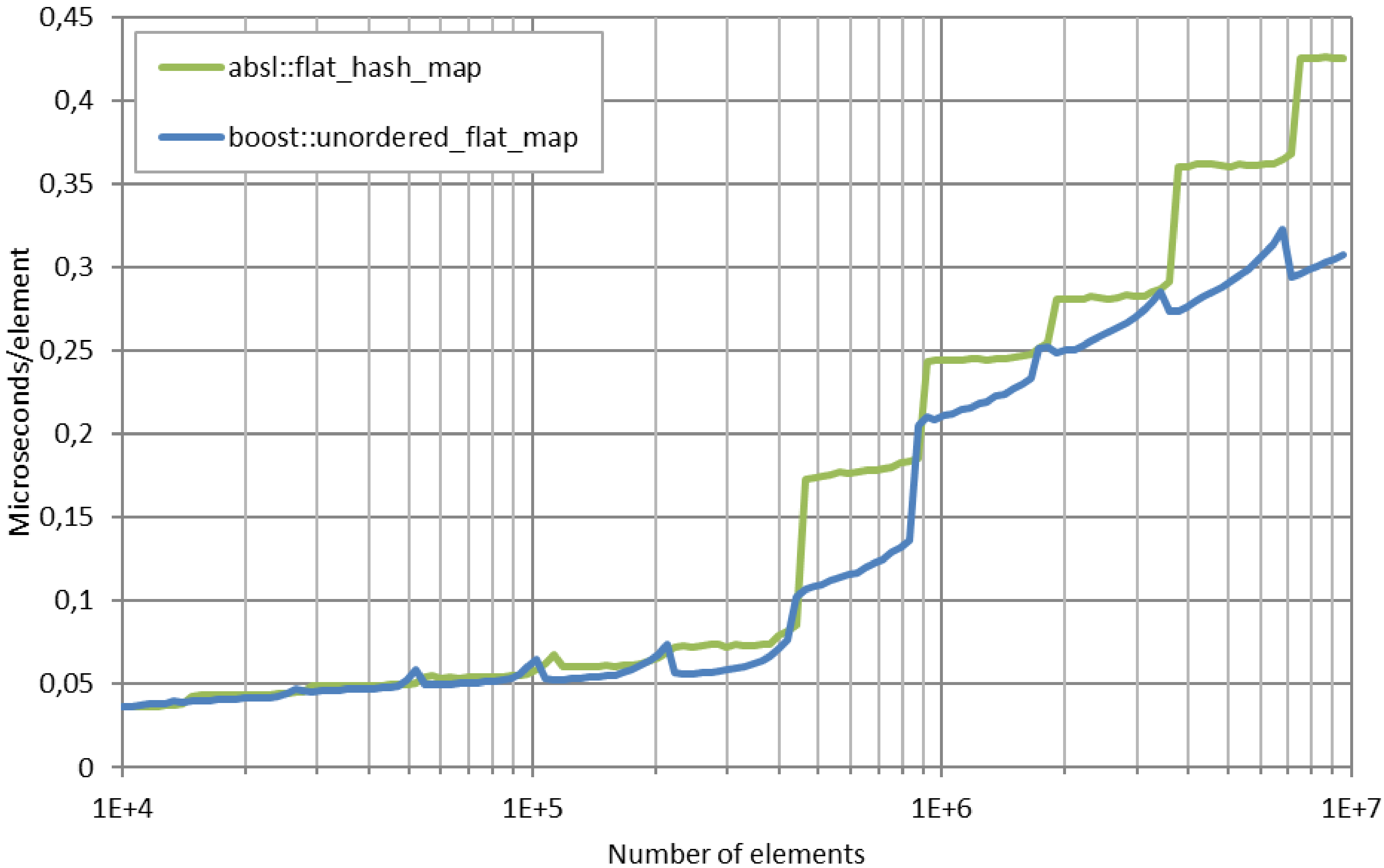
Stats

——— **boost**

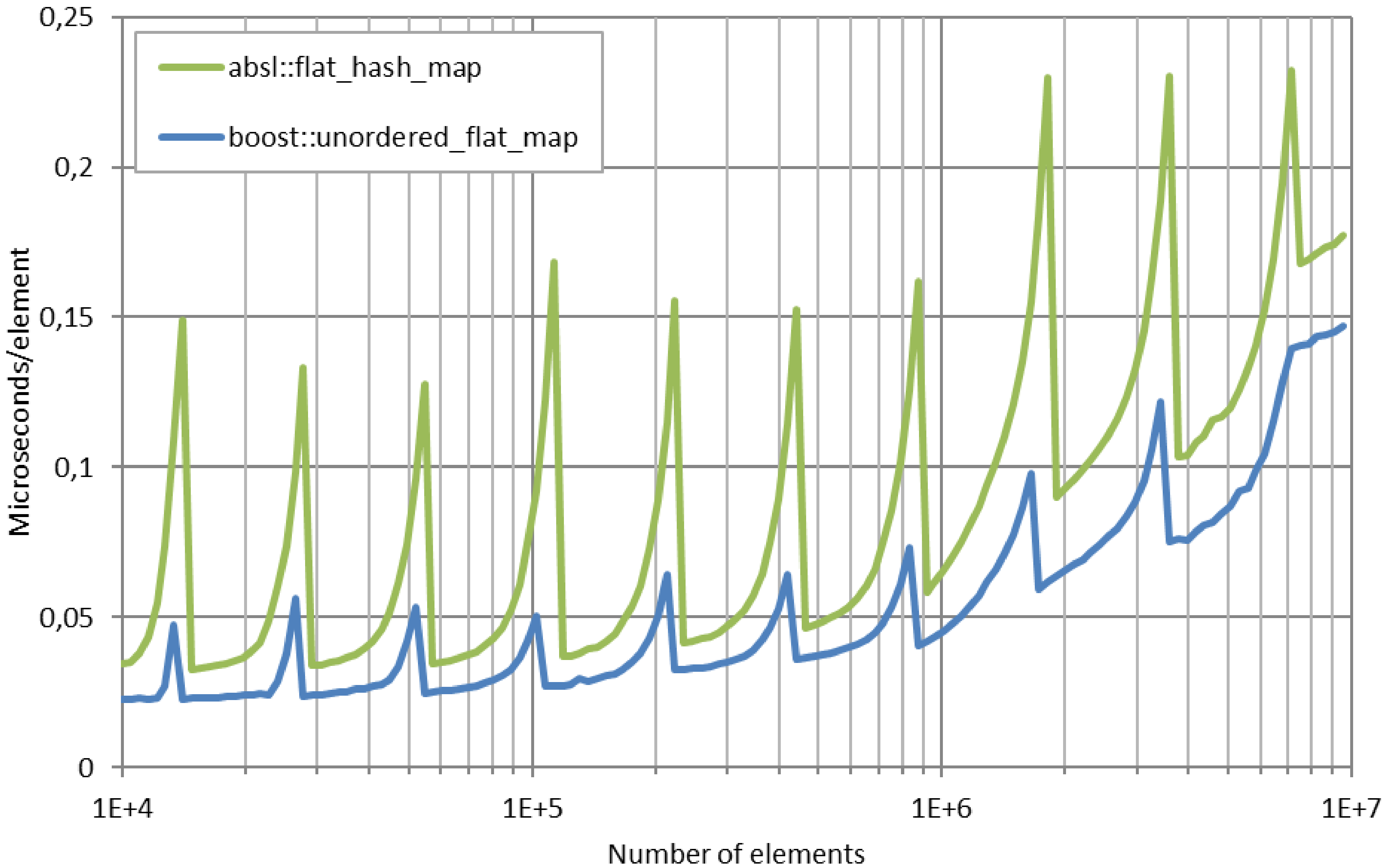
- - - - **absl**



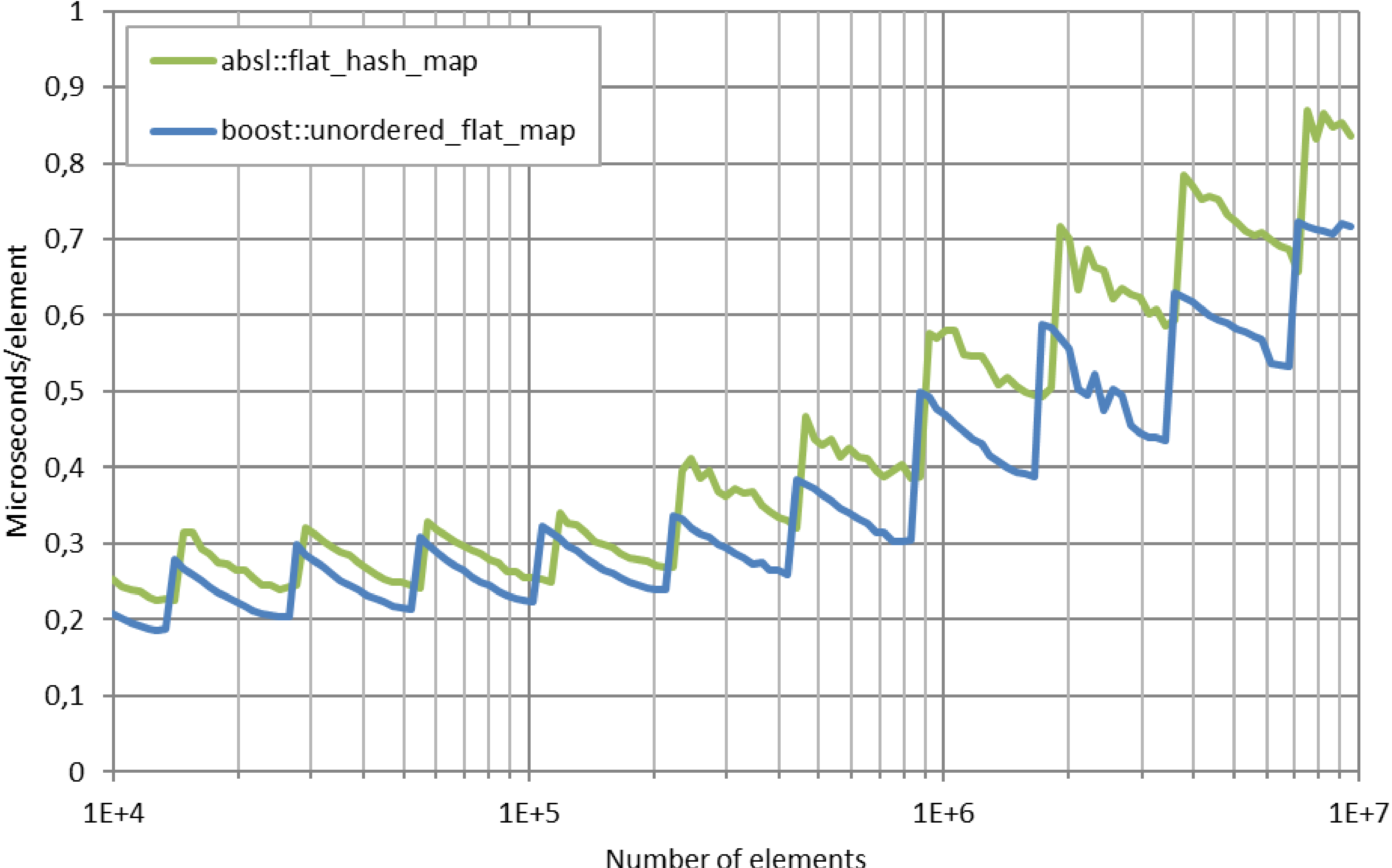
Successful lookup



Unsuccessful lookup



Insertion



Martin's benchmarks (github.com/martinus/map_benchmark)

		GameOfLife_growing	GameOfLife_stabilizing	knucleotide	RandomDistinct2	RandomInsertErase	RandomFind_200	RandomFind_2000	RandomFind_500000	RandomInsertEraseStrings	RandomFindString	RandomFindString_1000000	Memory
boost::unordered_flat_map	std::hash	101	100	125	193	100	132	129	100	105	140	117	158
gtl::flat_hash_map	std::hash	116	131	104	221	116	147	154	121	124	160	145	167
absl::flat_hash_map	robin_hood::hash	135	136	122	223	142	143	137	120	115	149	140	168
emhash8::HashMap	ankerl::unordered_dense::hash	158	146	234	271	144	115	109	127	136	129	149	172
emhash7::HashMap	mumx/std::hash	158	146	192	124	118	124	119	161	152	218	169	226
ska::bytell_hash_map	std::hash	160	164	134	132	146	148	145	161	151	227	192	167
ankerl::unordered_dense::map	ankerl::unordered_dense::hash	145	120	188	273	168	151	152	140	122	128	149	199
robin_hood::unordered_flat_map	std::hash	214	169	259	144	203	144	141	148	122	146	138	167
ankerl::unordered_dense::segmented_map	std::hash	156	131	221	277	168	162	159	141	145	155	121	173
folly::F14ValueMap	std::hash				235	182	183	178	151	133	154	149	149
ska::flat_hash_map	std::hash	135	135	100	131	109	106	105	172	139	243	196	451
rigtorp::HashMap	robin_hood::hash	172	170	191	132	136	130	143	180	156	241	182	300
tsl::robin_map	robin_hood::hash	182	180	242	171	143	133	131	193	141	211	169	451
robin_hood::unordered_node_map	mumx/std::hash	266	224	311	327	164	157	157	159	130	146	122	224
jg::dense_hash_map	mumx/std::hash	166	167	282	319	158	106	106	184	181	230	196	300
boost::unordered_node_map	std::hash	189	181	131	613	186	131	131	108	124	138	106	403
tsl::hopscotch_map	mumx/std::hash	233	224	191	159	162	174	252	224	165	236	200	300
tsl::sparse_map	ankerl::unordered_dense::hash	205	203	279	303	264	141	167	173	242	203	248	108
folly::F14NodeMap	std::hash				570	224	148	146	131	134	148	115	416
absl::node_hash_map	mumx/std::hash	242	239	115	669	209	139	141	129	137	153	126	437
spp::sparse_hash_map	absl::Hash	382	393	250	384	379	189	196	195	229	212	255	119
boost::unordered_map	mumx/std::hash	395	359	296	719	256	142	163	236	182	238	215	376
std::unordered_map	std::hash	402	296	225	599	495	351	342	412	278	309	349	371

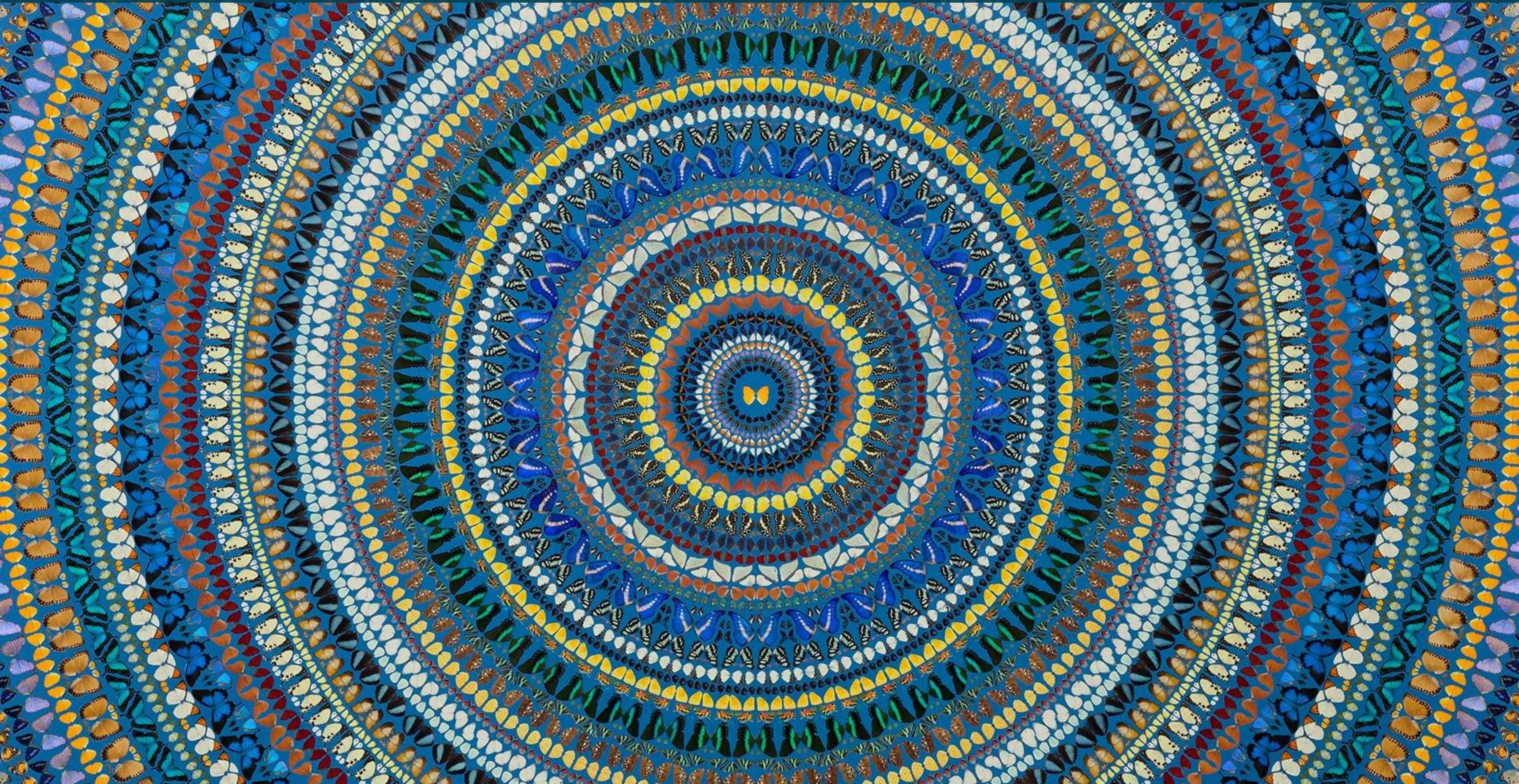
*All tests run with Clang 15.0.7 in Linux. Results normalized columnwise to best =100.

Migrating to `boost::unordered_flat_map`

Migrating to `boost::unordered_flat_map`

- Key and T must be `MoveConstructible`
- No pointer stability (or use `boost::unordered_node_map`)
- `begin` is not constant time
- `erase(iterator)` returns `void`
- Maximum load factor can't be changed
- No bucket API, no `extract`
- Otherwise, enjoy!

Teaser: `boost::concurrent_flat_map`



Teaser: `boost::concurrent_flat_map`

Uniform load

80
70
60
50
40
30
20
10
0

Millions of updates per second

tbb::concurrent_hash_map
boost::concurrent_flat_map

1 4 7 10 13 16 19

Number of threads

Biased load

80
70
60
50
40
30
20
10
0

Millions of updates per second

tbb::concurrent_hash_map
boost::concurrent_flat_map

1 4 7 10 13 16 19

Number of threads

Conclusions

Conclusions

- Boost.Unordered is providing a selection of hashmaps to suit everyone's needs
  - Fully C++ standard-compliant — `unordered_map`
  - Fastest, based on open addressing + SIMD — `unordered_flat_map`
  - Open addressing + SIMD + pointer stability — `unordered_node_map`
  - Concurrent (coming soon) — `concurrent_flat_map`
- `boost::unordered_flat_map` among fastest hashmaps in the market
  - We've looked under the hood to learn why
- Migrating from `std::unordered_map` gives away some functionality, watch out
- Stay tuned to #boost-unordered in cpplang.slack.com

More than a rehash

Thank you

github.com/joaquintides/usingstdcpp2023
github.com/boostorg/unordered
boost.org

“Las meninas”, © 1957 Pablo Ruiz Picasso; “El abrazo”, © 1976 Juan Veronés; “De sterrennacht”, © 1889 Vincent van Gogh; “Abacus No. 250”, © 1971 Yoshio Sekine; “Scène d'Orphée”, © Jean Cocteau; “Les joueurs de football”, © 1908 Henri Rousseau; “Water Lilies #1”, © 2023 Ai Weiwei; “Noble Path”, © 2019 Damien Hirst; “Una investigación” or “El Dr. Simarro en el laboratorio”, © 1897 Joaquín Sorolla

`using std::cpp 2023`

Joaquín M López Muñoz <joaquin.lopezmunoz@gmail.com>
Madrid, April 2023