How to create in-process KV-storage

Alexey Maslov <alexey.y.maslov@gmail.com>

What?

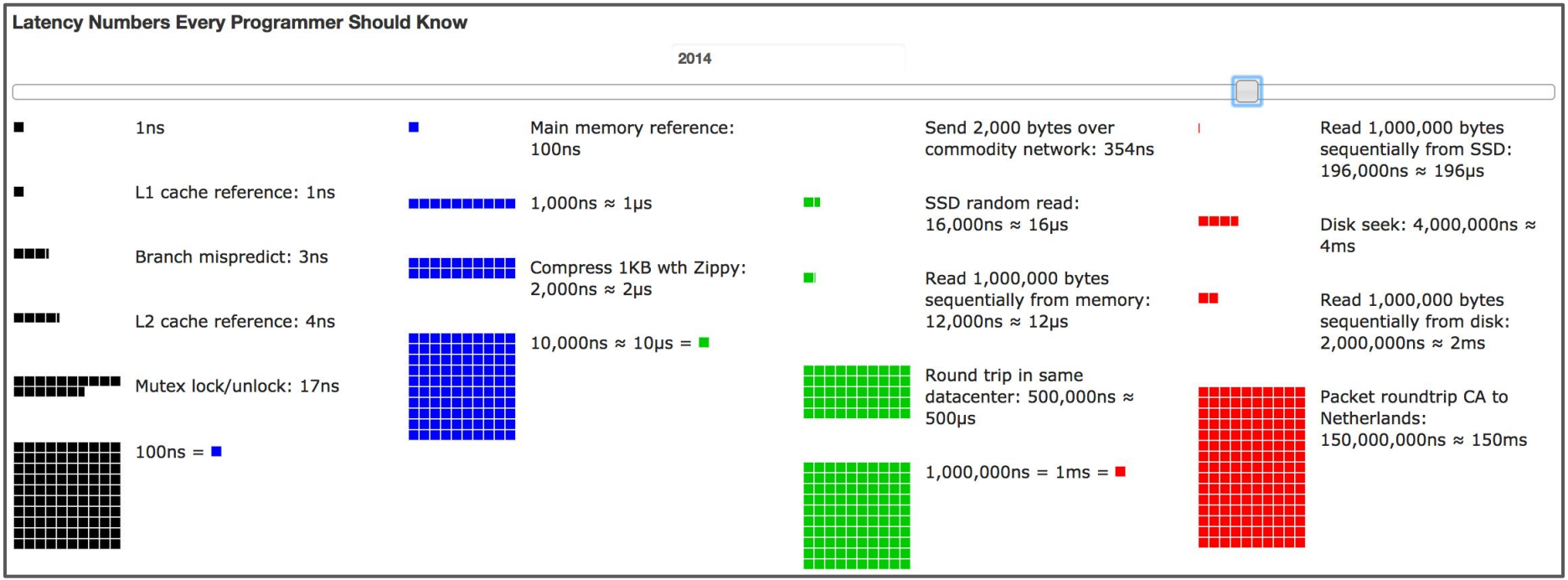
In-process storage is an architecture of embedded databases systems

Key-Value storage is a dictionary

So, **In-process KV-storage** is a dictionary database that is embedded in the application (golang)

https://en.wikipedia.org/wiki/Embedded_database

Why?



https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

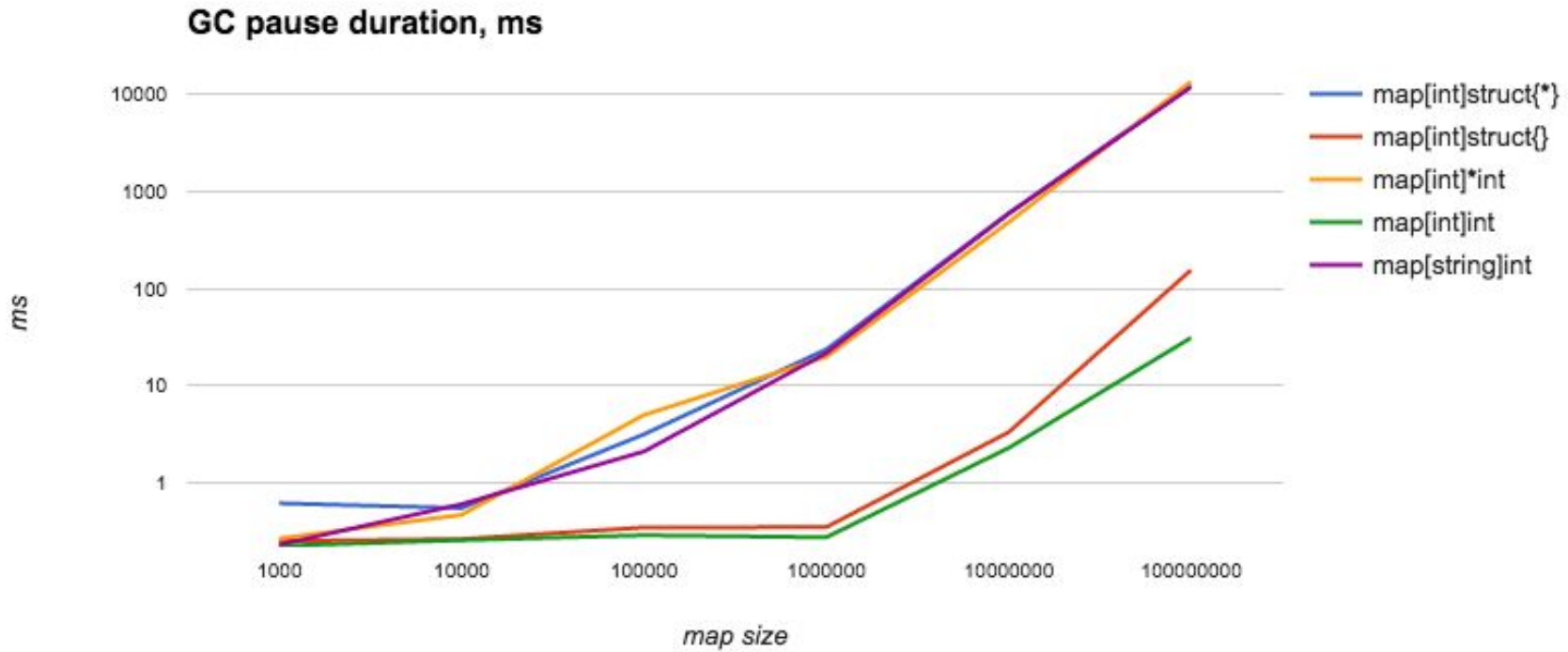
Interface

CRUD:

- *create* (**string**, **data**)
- *read* (**string**) **data**
- *update* (**string**, **data**)
- *delete* (**string**) “done”

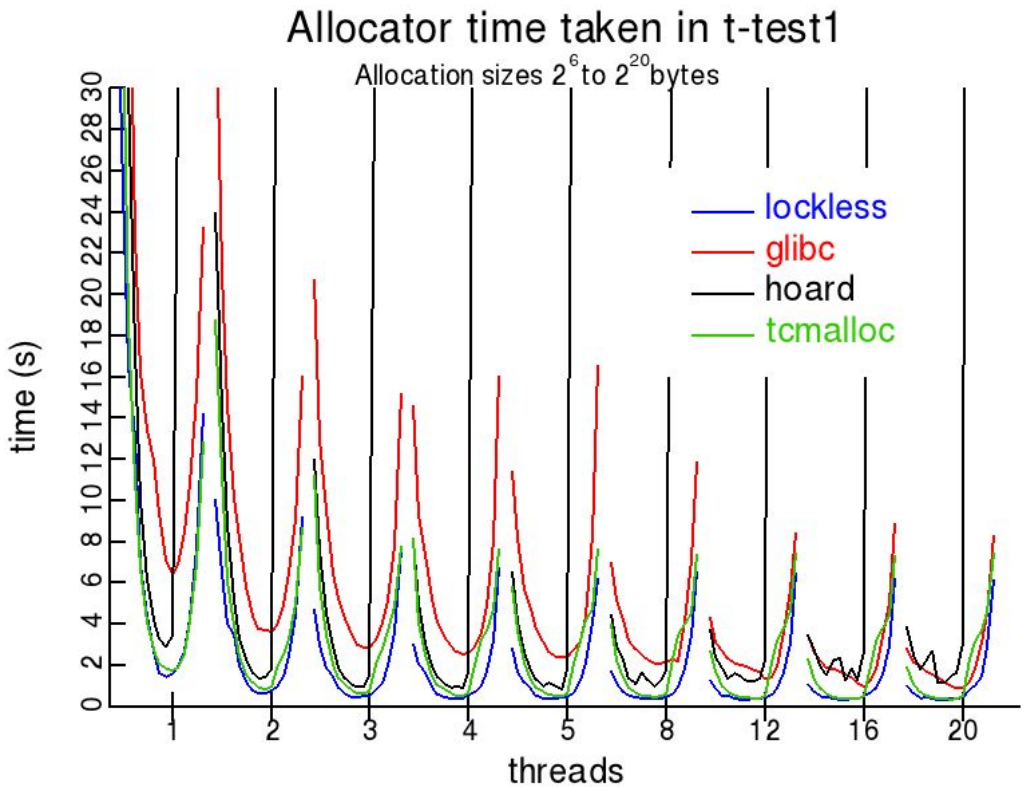
https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

GC pause on map



http://allegro.tech/2016/03/writing-fast-cache-service-in-go.html

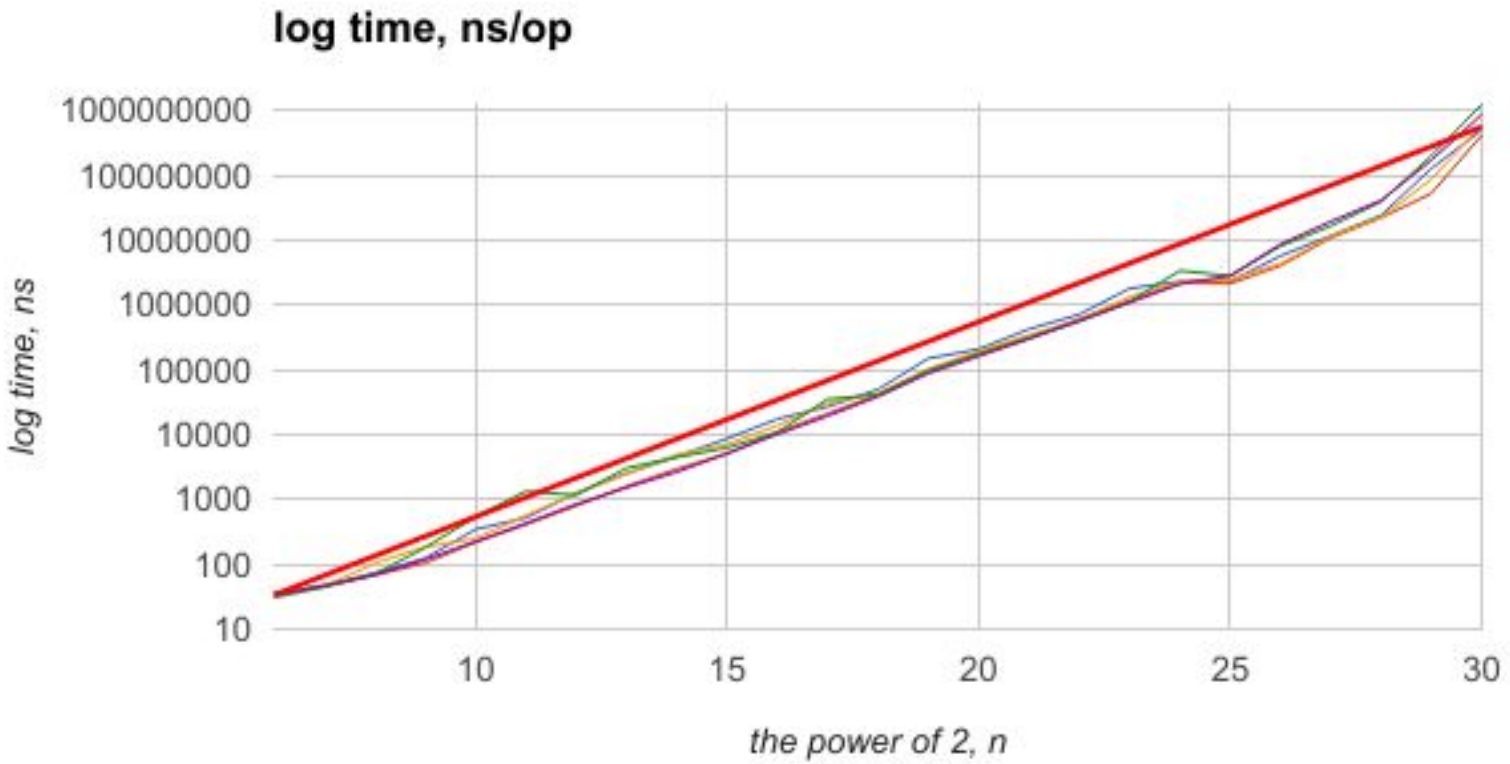
Allocations and deallocations



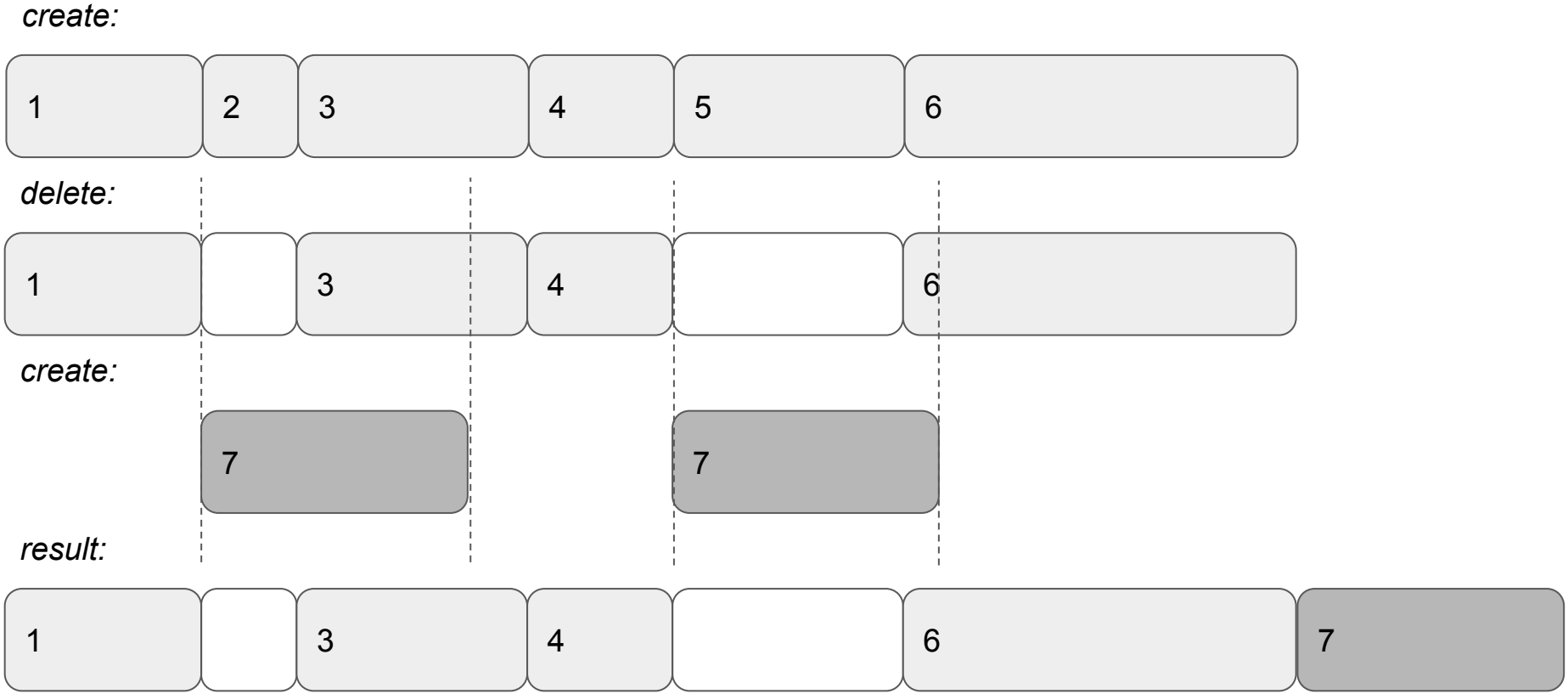
To show how fast the Lockless memory allocator is compared to others, we use the t-test1 benchmark. This benchmark is given a total amount of memory to use, and a maximum size of allocation to make. The benchmark chooses random sizes of memory below this maximum to call malloc(), calloc(), free(), realloc() and memalign(). The memory space is split into bins, and a specified number of parallel threads are used to access each bin. When a thread is done with a bin, it releases it, and acquires another. This causes the benchmark to test memory allocated on one thread, and freed in another.

http://locklessinc.com/benchmarks_allocator.shtml

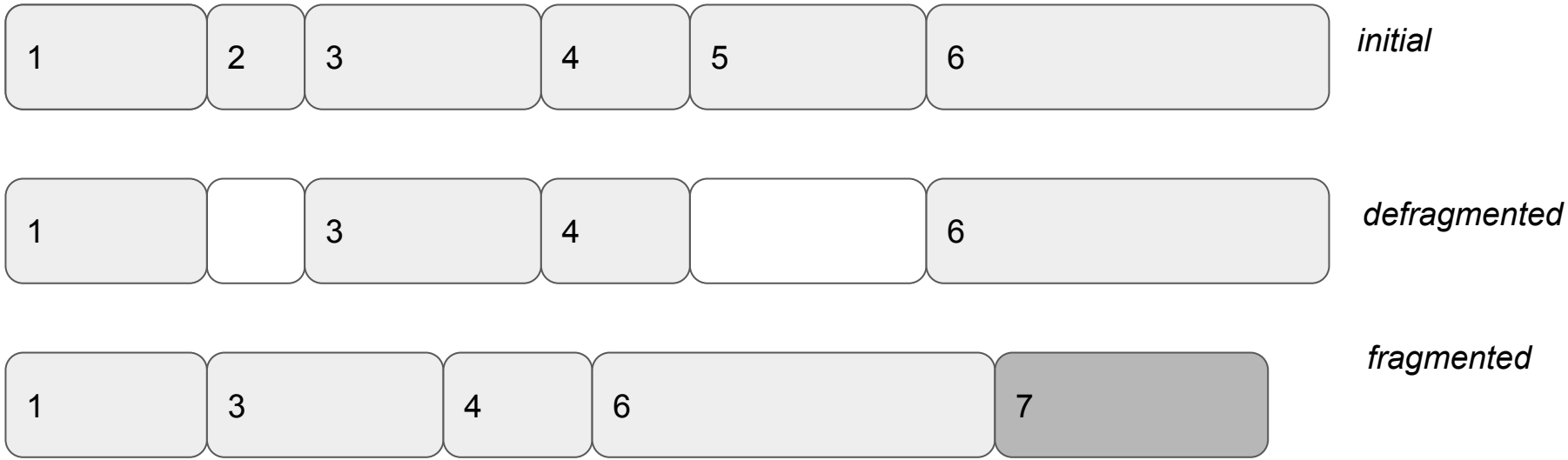
Memory allocation time



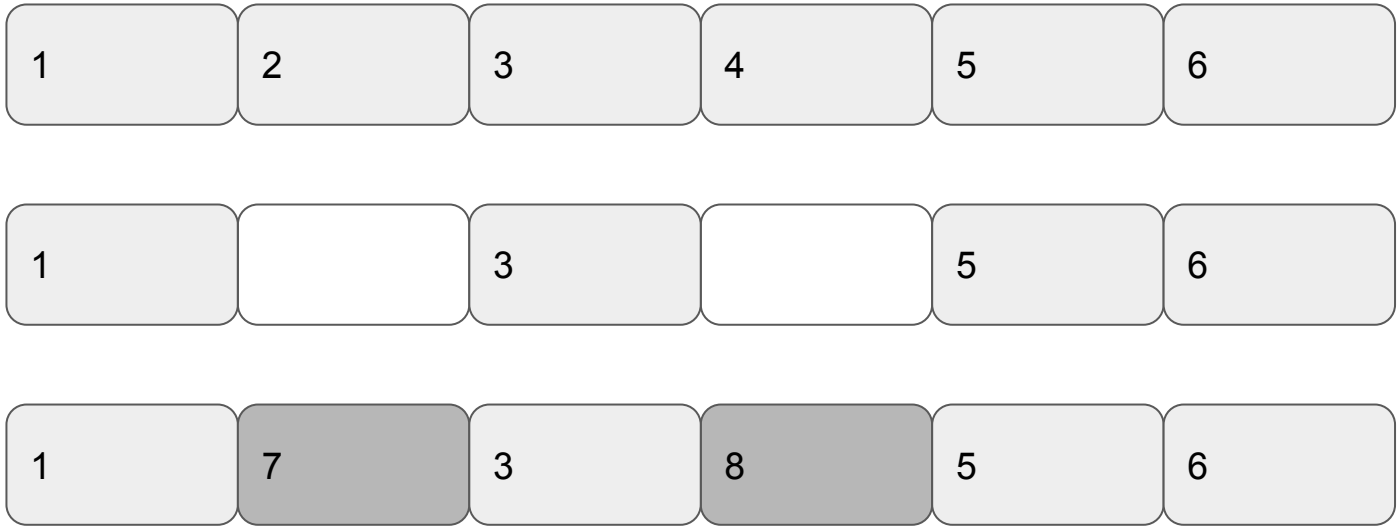
Fragmentation



Defragmentation



Persistent block size



Serialization

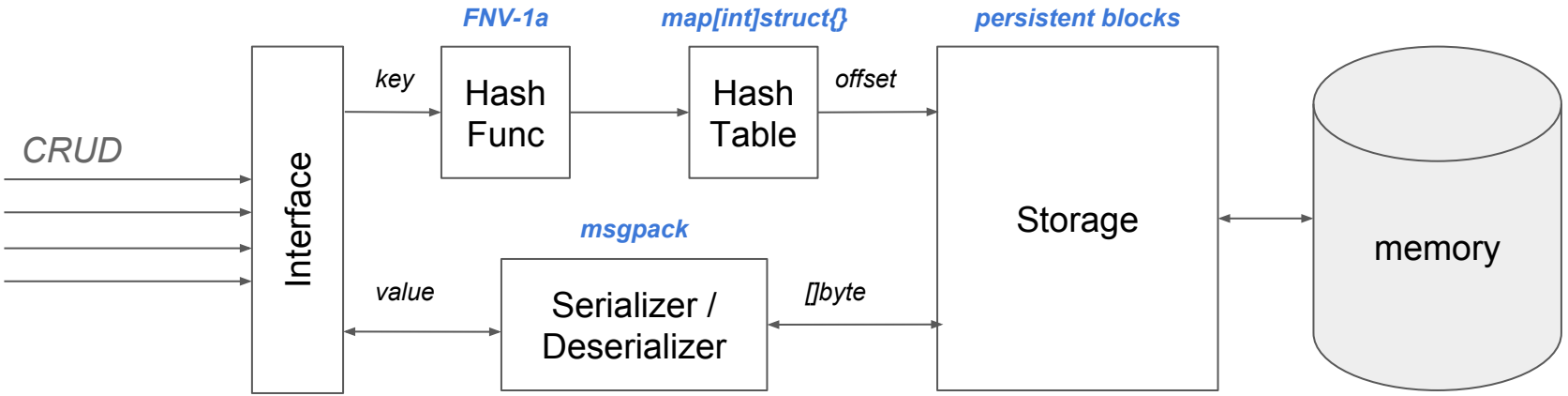
benchmark	iter	time/iter	bytes alloc	allocs
BenchmarkMsgpMarshal-8	5000000	269 ns/op	128 B/op	1 allocs/op
BenchmarkMsgpUnmarshal-8	3000000	498 ns/op	112 B/op	3 allocs/op
BenchmarkVmihailencoMsgpackMarshal-8	1000000	1910 ns/op	352 B/op	5 allocs/op
BenchmarkVmihailencoMsgpackUnmarshal-8	1000000	2154 ns/op	352 B/op	13 allocs/op
BenchmarkJsonMarshal-8	500000	3714 ns/op	1232 B/op	10 allocs/op
BenchmarkJsonUnmarshal-8	500000	4125 ns/op	416 B/op	7 allocs/op
BenchmarkEasyJsonMarshal-8	1000000	1920 ns/op	784 B/op	5 allocs/op
BenchmarkEasyJsonUnmarshal-8	1000000	1659 ns/op	160 B/op	4 allocs/op
BenchmarkBsonMarshal-8	1000000	1886 ns/op	392 B/op	10 allocs/op
BenchmarkBsonUnmarshal-8	500000	2487 ns/op	248 B/op	21 allocs/op
BenchmarkGobMarshal-8	1000000	1282 ns/op	48 B/op	2 allocs/op
BenchmarkGobUnmarshal-8	1000000	1306 ns/op	112 B/op	3 allocs/op
BenchmarkXdrMarshal-8	500000	2740 ns/op	455 B/op	20 allocs/op
BenchmarkXdrUnmarshal-8	1000000	2041 ns/op	239 B/op	11 allocs/op
BenchmarkUgorjiCodecMsgpackMarshal-8	500000	3316 ns/op	2753 B/op	8 allocs/op
BenchmarkUgorjiCodecMsgpackUnmarshal-8	500000	3289 ns/op	3008 B/op	6 allocs/op
BenchmarkUgorjiCodecBincMarshal-8	500000	3237 ns/op	2785 B/op	8 allocs/op
BenchmarkUgorjiCodecBincUnmarshal-8	500000	3631 ns/op	3168 B/op	9 allocs/op
BenchmarkSerealMarshal-8	300000	4453 ns/op	912 B/op	21 allocs/op
BenchmarkSerealUnmarshal-8	500000	3889 ns/op	1008 B/op	34 allocs/op
BenchmarkBinaryMarshal-8	1000000	1841 ns/op	256 B/op	16 allocs/op
BenchmarkBinaryUnmarshal-8	1000000	1945 ns/op	336 B/op	22 allocs/op
BenchmarkFlatbuffersMarshal-8	3000000	411 ns/op	0 B/op	0 allocs/op
BenchmarkFlatBuffersUnmarshal-8	5000000	354 ns/op	112 B/op	3 allocs/op
BenchmarkCapNProtoMarshal-8	2000000	578 ns/op	56 B/op	2 allocs/op
BenchmarkCapNProtoUnmarshal-8	3000000	515 ns/op	200 B/op	6 allocs/op
BenchmarkCapNProto2Marshal-8	1000000	1427 ns/op	244 B/op	3 allocs/op
BenchmarkCapNProto2Unmarshal-8	1000000	1325 ns/op	320 B/op	6 allocs/op
BenchmarkHproseMarshal-8	1000000	1294 ns/op	479 B/op	8 allocs/op
BenchmarkHproseUnmarshal-8	1000000	1715 ns/op	320 B/op	10 allocs/op
BenchmarkProtobufMarshal-8	1000000	1554 ns/op	200 B/op	7 allocs/op
BenchmarkProtobufUnmarshal-8	1000000	1055 ns/op	192 B/op	10 allocs/op
BenchmarkGoprotobufMarshal-8	2000000	746 ns/op	312 B/op	4 allocs/op
BenchmarkGoprotobufUnmarshal-8	2000000	978 ns/op	432 B/op	9 allocs/op
BenchmarkGogoprotobufMarshal-8	10000000	211 ns/op	64 B/op	1 allocs/op
BenchmarkGogoprotobufUnmarshal-8	5000000	289 ns/op	96 B/op	3 allocs/op
BenchmarkColferMarshal-8	10000000	178 ns/op	64 B/op	1 allocs/op
BenchmarkColferUnmarshal-8	10000000	235 ns/op	112 B/op	3 allocs/op
BenchmarkGencodeMarshal-8	5000000	212 ns/op	80 B/op	2 allocs/op
BenchmarkGencodeUnmarshal-8	5000000	273 ns/op	112 B/op	3 allocs/op
BenchmarkGencodeUnsafeMarshal-8	10000000	135 ns/op	48 B/op	1 allocs/op
BenchmarkGencodeUnsafeUnmarshal-8	10000000	198 ns/op	96 B/op	3 allocs/op

Time/iteration TOP5:

1. Gencode
2. Colfer
3. Gogoprotobuf
4. Msgpack
5. FlatBuffers

https://github.com/alecthomas/go_serialization_benchmarks

Summary



https://github.com/alxmsl/ipkvs

Results. GC overhead

```
 sources.go $: go run
src/github.com/allegro/bigcache/caches_bench/caches_gc_overhead_comparsion.go
Number of entries:  20000000
GC pause for bigcache:  16.062938ms
GC pause for freecache:  9.465ms
GC pause for map:  5.327027052s

 sources.go $: go run
src/github.com/alxmsl/ipkvs/benches/gc_ipkvs/msgpack/msgpack_bench.go -l
20000000
5.270392ms
```

Results. Comparison

```
 sources.go $: go test -bench=. -benchmem
src/github.com/alxmsl/ipkvs/benches/comparision/comparision_bench_test.go
BenchmarkGoCacheSet-4           2000000          746 ns/op             108 B/op          2 allocs/op
BenchmarkFreeCacheSet-4         3000000          467 ns/op              11 B/op          1 allocs/op
BenchmarkBigCacheSet-4          2000000          689 ns/op             117 B/op          0 allocs/op
BenchmarkCacheSetMsgPack-4      1000000          924 ns/op             295 B/op          4 allocs/op
BenchmarkCacheSet-4             3000000          397 ns/op             116 B/op          0 allocs/op
PASS
ok   command-line-arguments   11.682s
```

Thank you