```
ReJSON = {
    "id":          "old dog",
    "activity":    "new trick"
}
```

Itamar Haber
@itamarhaber

redisday
TLV 2017

redislabs
home of redis

What do Chuck Norris, JSON & Redis have in common?

They're everywhere.

"Any application that *can* be written in JavaScript, *will* eventually be written in JavaScript."

Attwod's Law

"Any database that *can* store JSON, *will* eventually store JSON."

Orthodox storage of JSON in Redis

With Redis' core data structures you can store JSON

1. **Raw in String keys:** the document is stored in serialized form
   a. Lua may be used to encode/decode on the fly
   b. MessagePack is option with additional encoding/decoding
2. **Decomposed in Hash keys:** the data is deserialized to key-value pairs

Raw JSON in String keys - DEMO!

```
127.0.0.1:6379> SET rawjson '{"foo": "bar",
"ans": 42}'
OK
127.0.0.1:6379> GET rawjson
"{\"foo\": \"bar\", \"ans\": 42}"
```

Raw JSON String keys (orthodox #1)

- Advantages
  - Data is stored serialized - perfect for opaque caching, i.e. entire "BLOB" read/write
  - Medium memory overhead (JSON is readable)
- Disadvantages
  - Element access is impossible - entire bulk must be read, processed and possibly written back by the client. This adds traffic, latency and complexity to application code.
  - Modifications are therefore not atomic

Raw JSON in String keys with Lua (#1.a)

```
$ cat json-get-path.lua
local js = redis.call('GET', KEYS[1])
local v = cjson.decode(js)
-- Parse the path
local r = ...

local rjs = cjson.encode(r)
return rjs
```

Raw JSON/MessagePack String keys and Lua

- Additional advantages
  - Elements are accessible
  - Updates are atomic
  - MessagePack has lower memory overhead and is faster (vs. JSON)
- Disadvantages
  - Access time depends on JSON's size, or O(N)
  - Lua isn't for everyone and introduces more code to maintain

Decomposed Hash Keys (orthodox #2)

```
127.0.0.1:6379> HSET decomposed foo bar
(integer) 1
127.0.0.1:6379> HSET decomposed ans 42
(integer) 1
...
```

Decomposed Hash Keys

- Advantages
  - Elements are accessible in O(1)
- Disadvantages
  - No native way to decode/encode to/from JSON/Hash means a client-side or Lua implementation
  - No nesting means only for flat objects (dictionaries)
  - Only String/"Number" data types
  - Redis Hash memory overheads

ReJSON = Redis + JSON

ReJSON in one slide [Preview Release]

- A custom JSON data type for Redis (v4 Modules API)
- Keys can contain any valid JSON value
  - Scalars, objects or arrays
  - Nested or not
- Data is stored decoded in binary format
- JSONPath-like syntax for direct access to elements
- Strongly-typed atomic commands

ReJSON - basic SET and GET

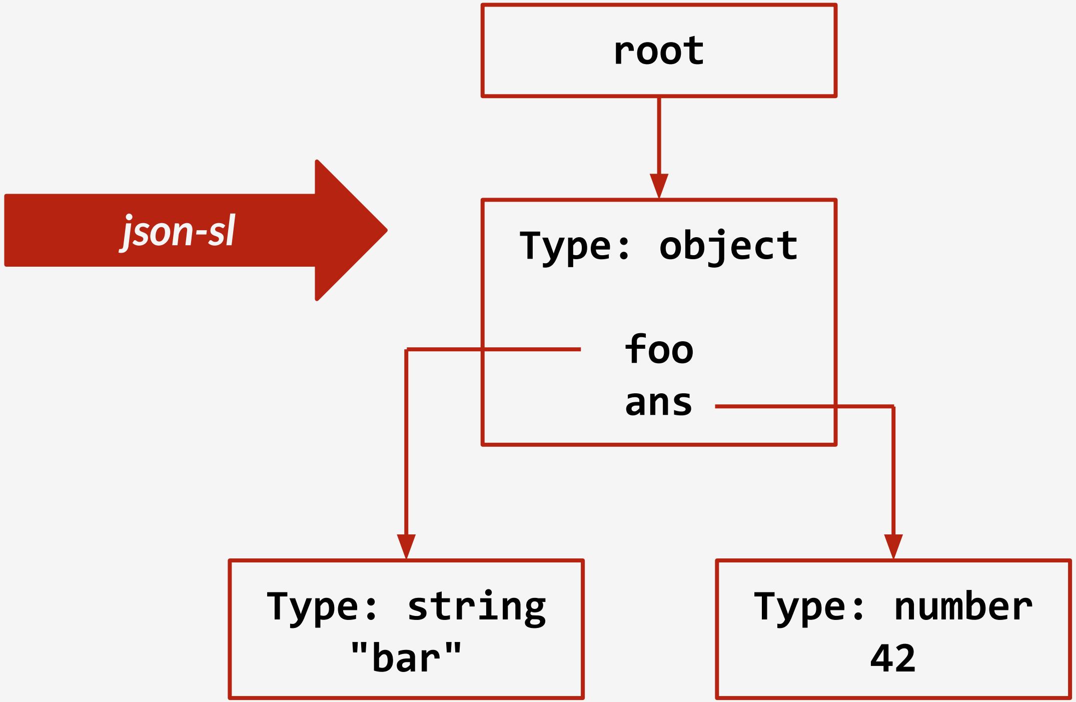
```
127.0.0.1:6379> JSON.SET scalar . '"Hello JSON!"'
OK
127.0.0.1:6379> JSON.SET object . '{"foo": "bar",
"ans": 42}'
OK
127.0.0.1:6379> JSON.GET object
"{\"foo\":\"bar\",\"ans\":42}"
127.0.0.1:6379> JSON.GET object .ans
"42"
```

ReJSON - who's the prettiest of them all?

```
127.0.0.1:6379> ^C
$ redis-cli --raw
127.0.0.1:6379> JSON.GET object INDENT "\t"
NEWLINE "\n" SPACE " "
{
   "foo": "bar",
   "ans": 42
}
```

JSON value -> ReJSON tree data structure

```
{
   "foo": "bar",
   "ans": 42
}
```



json-sl

root

Type: object

foo
ans

Type: string
"bar"

Type: number
42

ReJSON for storing JSON data

- Advantages
    - Full and intuitive JSON support
    - Works with any Redis client, no extra coding needed
    - Elements are efficiently accessible by path (read/write)
- Disadvantages
    - Serializing the value to JSON is "expensive"
    - Higher memory overhead (vs. serialized form)

A note about paths

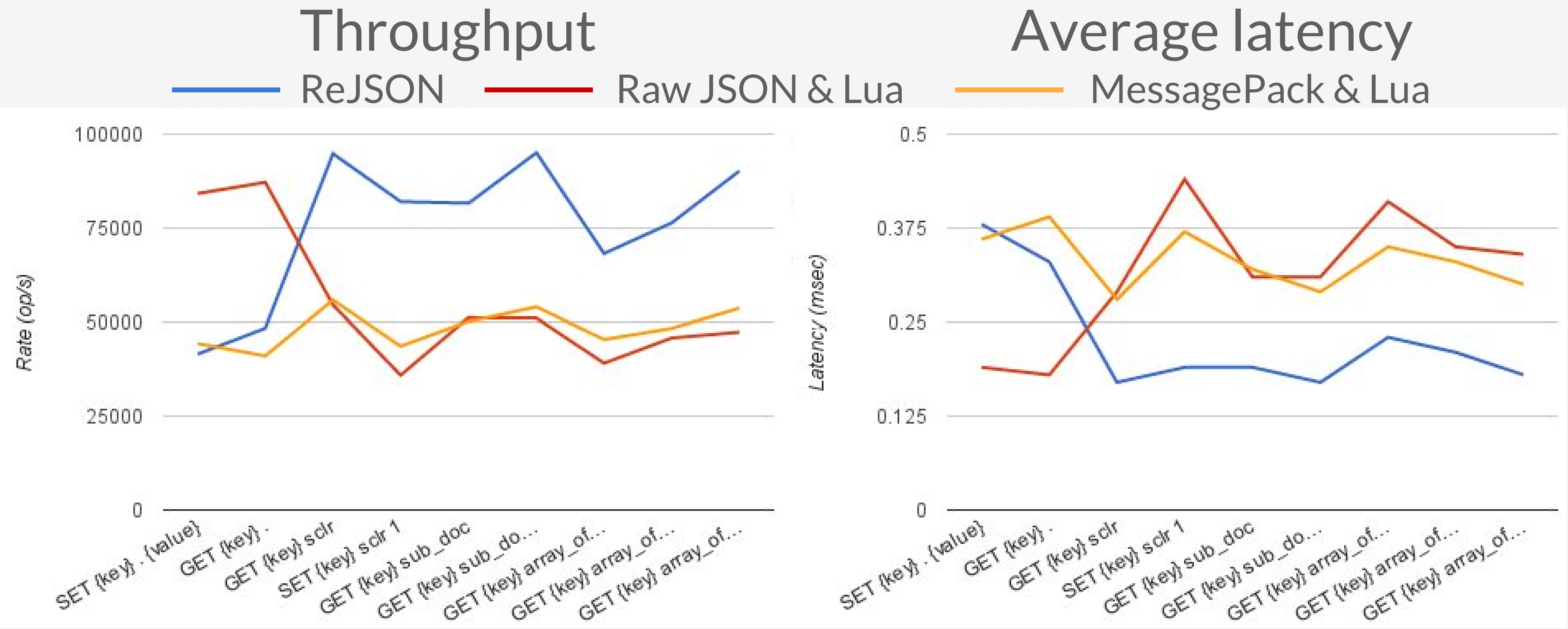
There are at least two standards for JSON paths…

… which means there is no standard for JSON paths.

ReJSON implements a subset of the seemingly more popular JSONPath "standard", basically:

- Canonical, dot-separated notation
- Brackets denote keys or list indices
- Example: `.foo.bar[0]`

Performance: 380 bytes, 3 nesting levels



Throughput

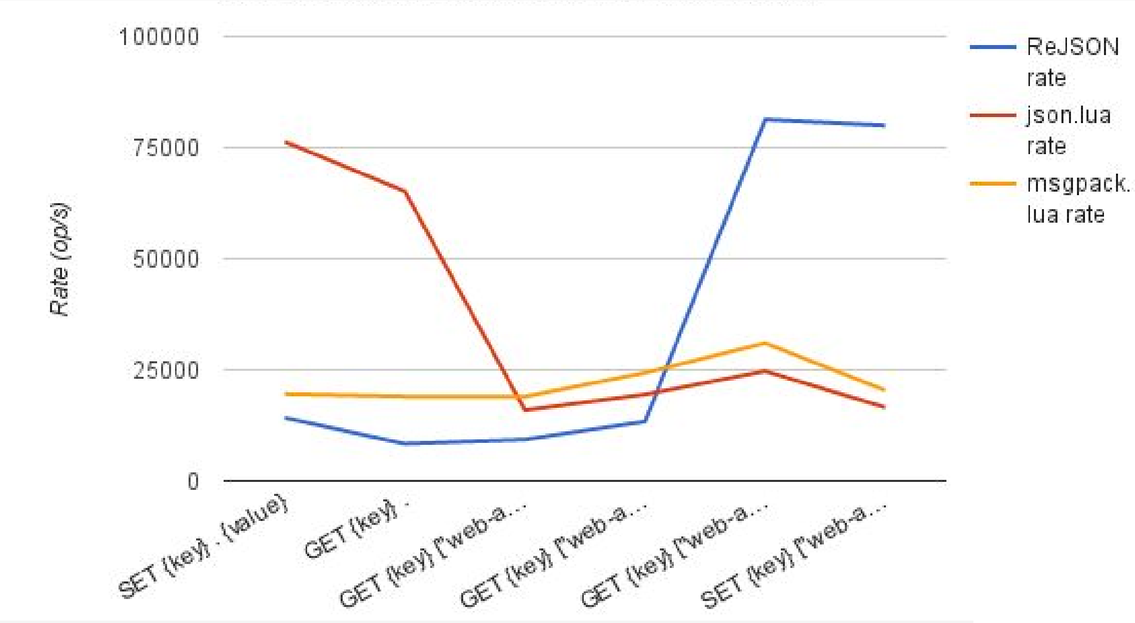
Average latency

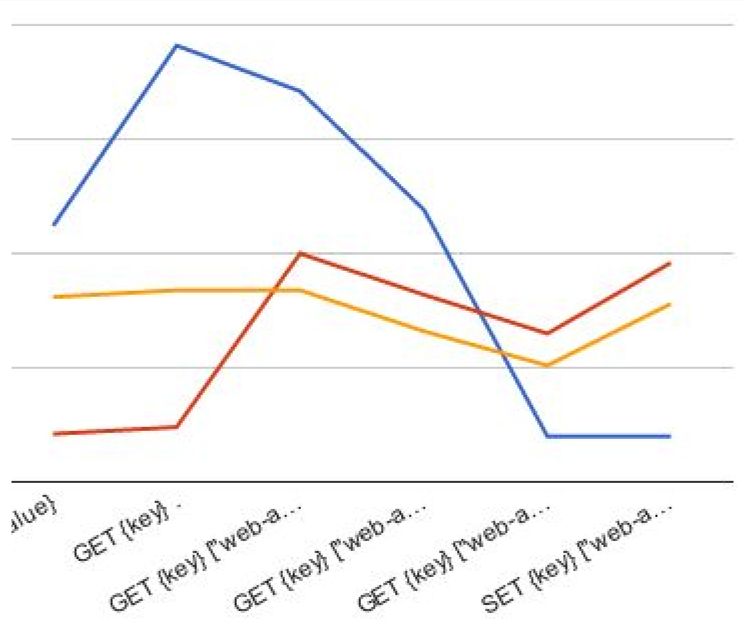
Performance: 3468 bytes, 3 nesting levels

Throughput

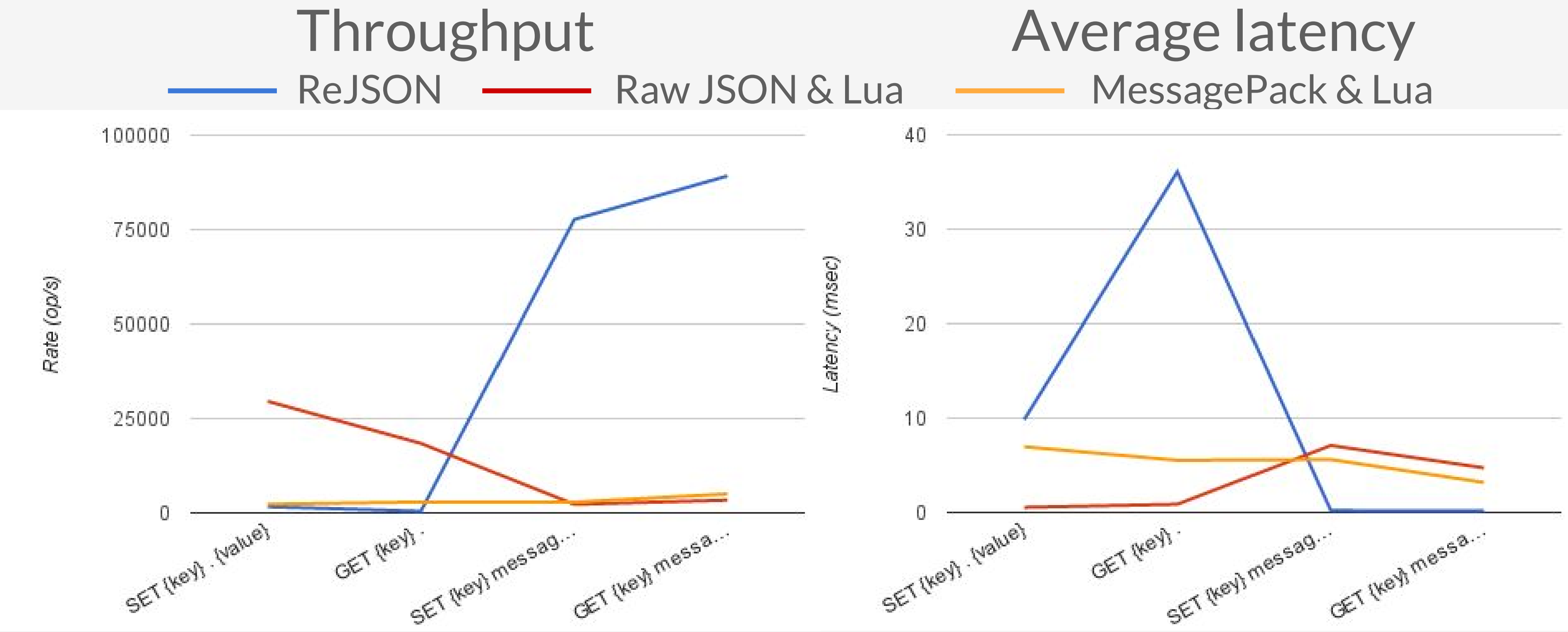
Average latency

ReJSON — Raw JSON & Lua — MessagePack & Lua





Performance: 39491 bytes, 3 nesting levels



Throughput

Average latency

redislabs
home of redis

ReJSON commands

General	`JSON.DEL, JSON.GET, JSON.MGET, JSON.SET & JSON.TYPE`
Numbers	`JSON.NUMINCRBY & JSON.NUMMULTBY`
Strings	`JSON.STRAPPEND & JSON.STRLEN`
Objects	`JSON.OBJKEYS & JSON.OBJLEN`
Arrays	`JSON.ARRAPPEND, JSON.ARRINDEX, JSON.ARRINSERT, JSON.ARRLEN, JSON.ARRPOP & JSON.ARRTRIM`
Other	`JSON.RESP`

Where and when

Source code: https://github.com/RedisLabsModules/rejson

Documention: https://redislabsmodules.github.io/rejson

- Now: preview release
- Future: data compression, schema validation, secondary indices, querying & more
- Your feature and pull requests are welcome :)

Thank you, woof!