Creating a Custom Serialization Format

Scott Mansfield (@sgmansfield)
Senior Software Engineer
Netflix

NETFLIX

What are we doing here?

1. Motivations
2. Queries
3. The Format
4. Performance
5. Future

NETFLIX

1. Motivations

"The field is too in love with horribly inefficient frameworks. Writing network code and protocols is now considered too low level for people."

- jnordwick (Hacker News)

Motivations

- Computers make meaning out of voltages
- Serialization is everywhere
  - Network protocols
  - Video encoding
  - Machine code
  - HTTP/2 headers
  - Hard drive communication
  - Video display
- Engineers should know what's inside the black box

Motivations

- JSON is the de facto serialization format
- Common pattern:
  1. Get entire document
  2. Inflate serialized data
  3. Walk data structure & extract
- New pattern:
  1. Query the document
  2. Get only the data you need
  3. Still need to inflate

Motivations

- Query capabilities over JSON documents
- Documents stored as a byte array

JSON Document (Augmented)

```
{
  "null"    : null,
  "boolean" : true,
  "integer" : 1,
  "float"   : 2.3,
  "string"  : "a string",
  "array"   : [4, 5, 6],
  "map"     : {"foo": 1}
}
```

2. Queries

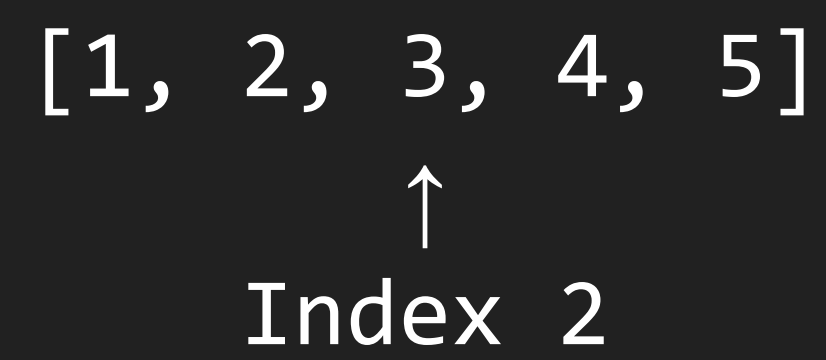
Query Types

- Array Index
- Array Slice
- Array Iteration

- Map Access
- Map Keys
- Map Iteration

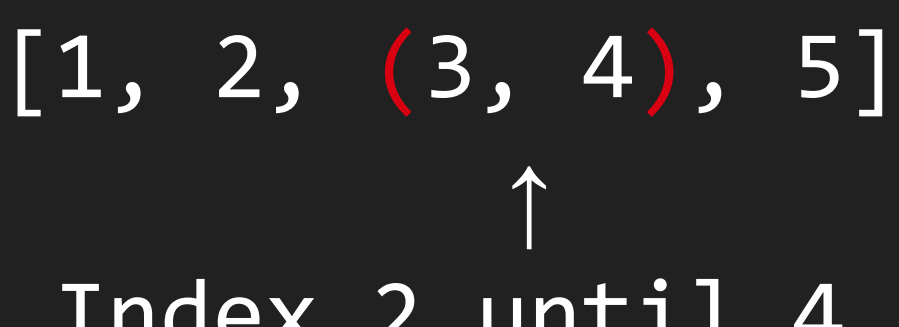
Array Index

```
Query:  [2]
Result: 3
```



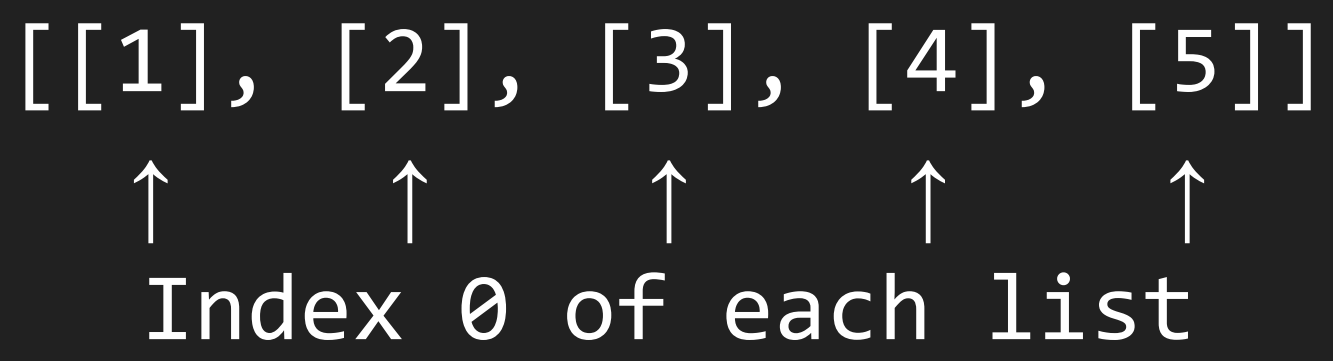
Array Slice

```
Query:  [2:-1]
Result: [3, 4]
```



Array Iteration

```
Query:   .a[] [0]
Result: [1,2,3,4,5]
```



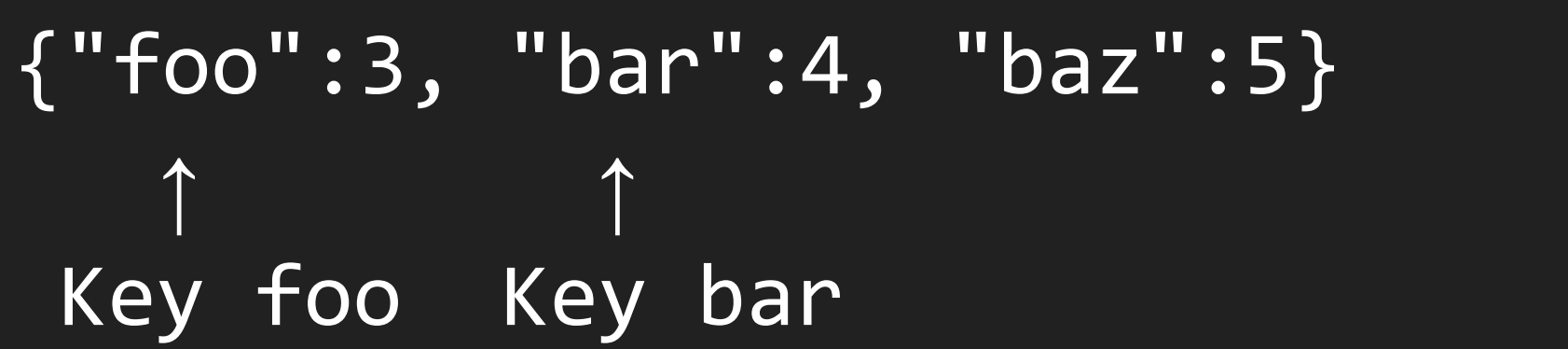
Map Access (Single)

```
Query:  .foo
Result: 3
```

```
{"foo": 3, "bar": 4}
   ↑
 Key foo
```

Map Access (Multiple)

```
Query:   .foo|bar
Result: {"foo":3, "bar":4}
```



Map Keys

```
Query:  keys
Result: ["foo", "bar"]
```

```
{"foo": 3, "bar": 4}
   ↑          ↑
   Map Keys
```

Map Iteration

```
Query:  .m[] [0]
Result: {"foo": 3, "bar": 4}

      {"foo": [3], "bar": [4]}
                ↑          ↑
        Index 0 of each array value
```

Example

```
{"foo": {"k1": [3,4]},
 "bar": {"k1": [5,6]}}

Query:   .m[] .k1 [0]
Result: {"foo": 3, "bar": 5}
```

Example

```
{"foo": {"1":1, "2":2, "3":3},
 "bar": {"4":4, "5":5, "6":6}}

Query:  .m[] keys
Result: {"foo": ["1","2","3"],
         "bar": ["4","5","6"] }
```

3. The Format

NETFLIX

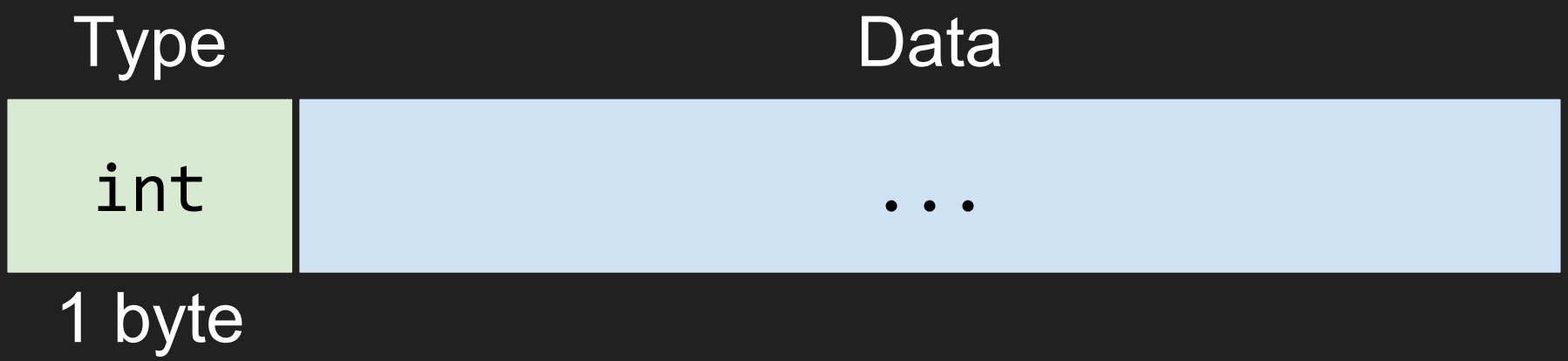
Types

Augmented JSON == JSON + integers

- Scalars
  - Null
  - Boolean
  - Integer (64 bit)
  - Float (64 bit)
  - String
- Composites
  - Array
  - Map

General Format

Every record starts with a single byte for the type:

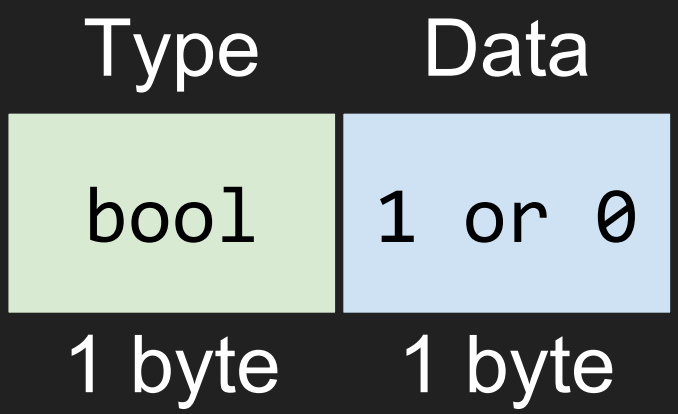


Scalars

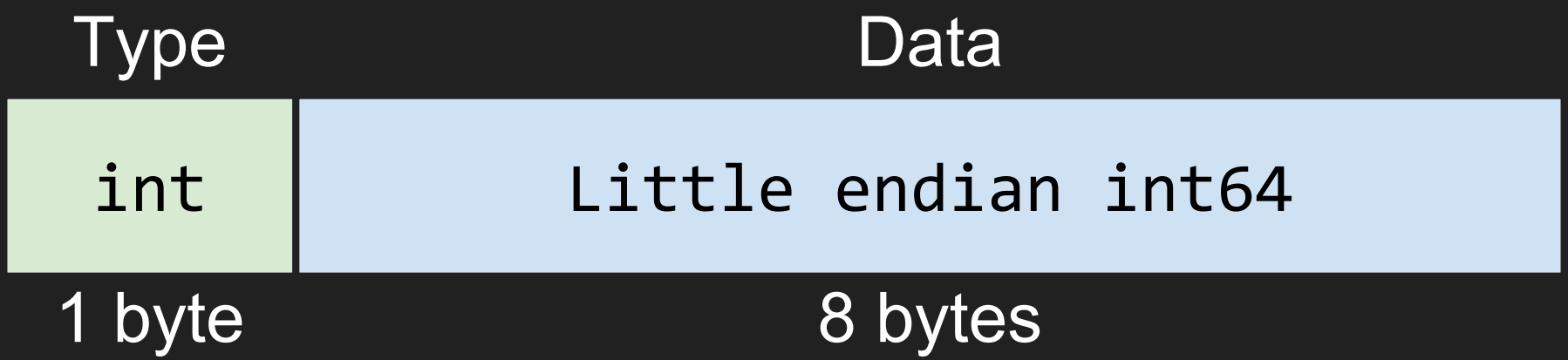
- Null
- Boolean
- Integer (64 bit)
- Float (64 bit)
- String

Scalar: Null

Scalar: Boolean

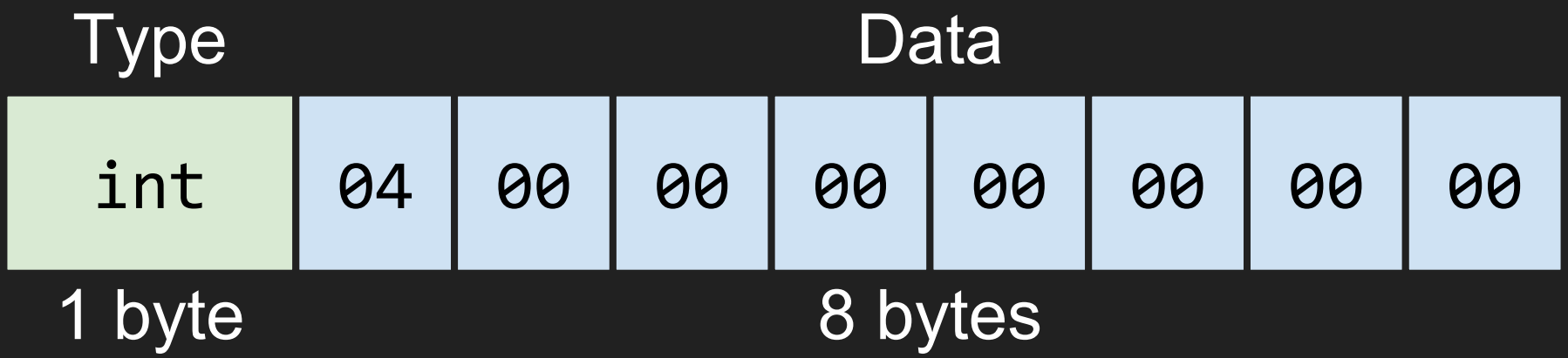


Scalar: Integer

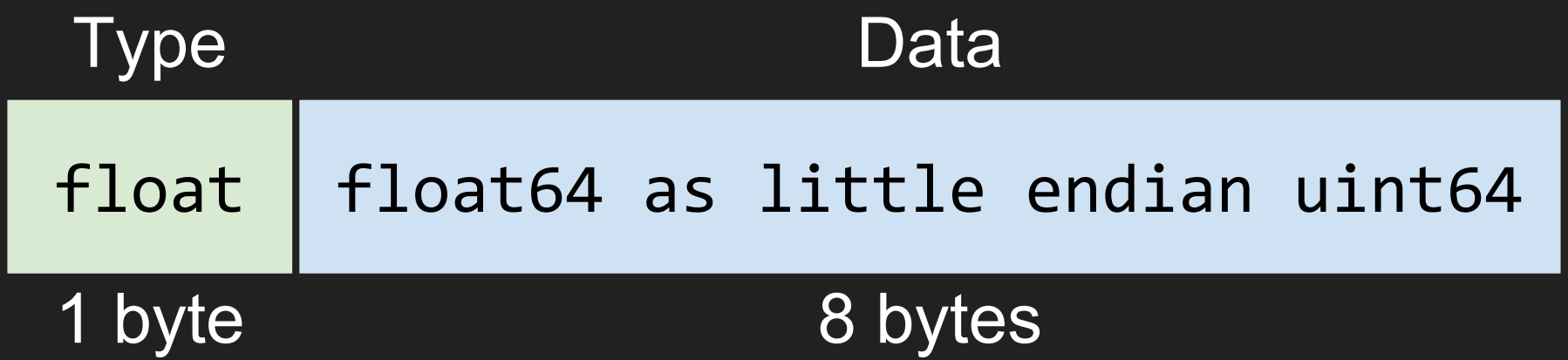


Scalar: Integer (example)

`4 = 0x0000_0000_0000_0004`

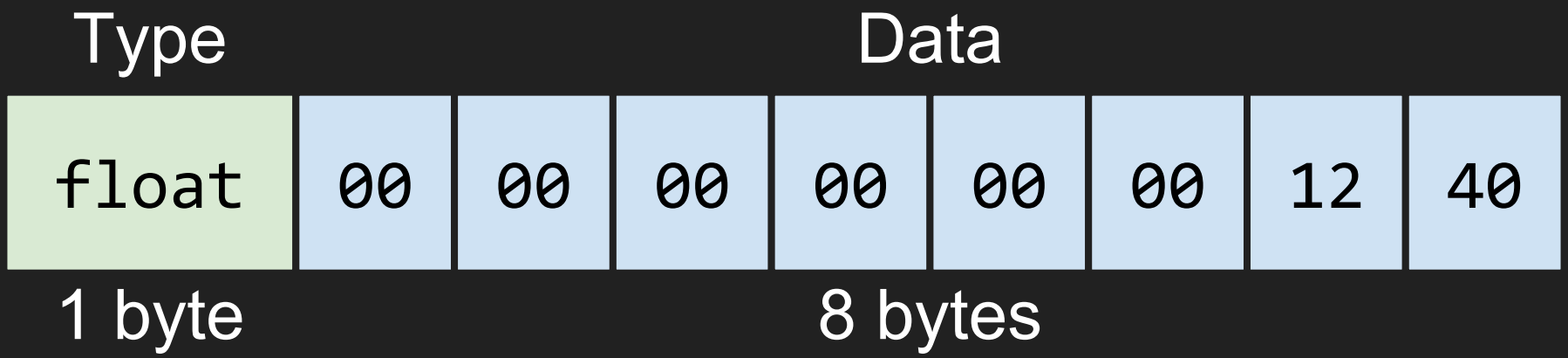


Scalar: Float

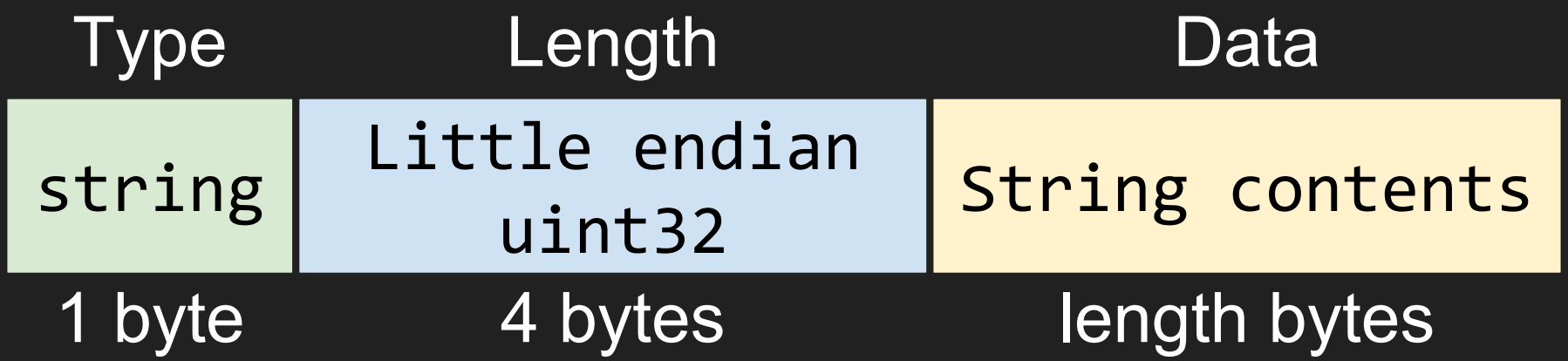


Scalar: Float (example)

4.5 = `0x4012_0000_0000_0000`

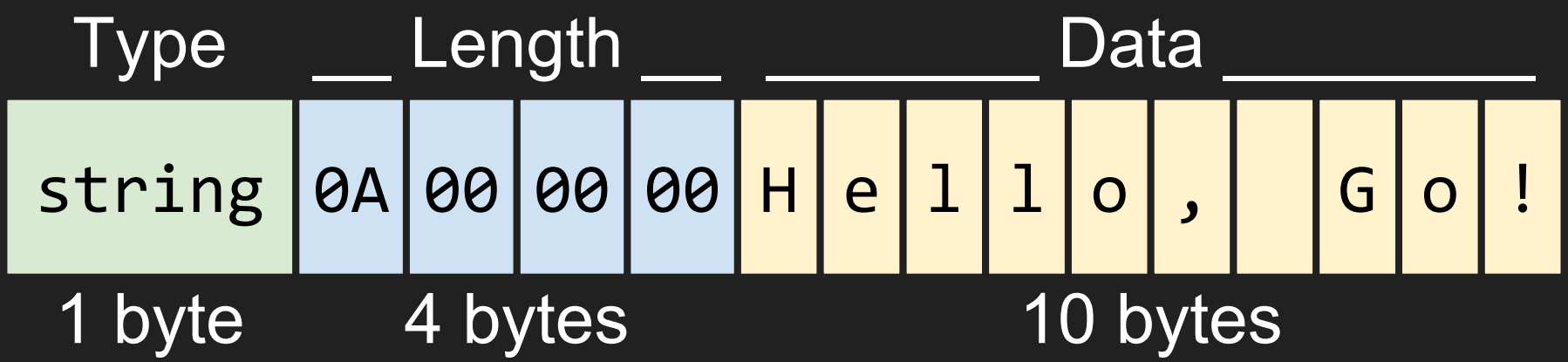


Scalar: String



Scalar: String (example)

"Hello, Go!" Length: `10 = 0x0000_000A`

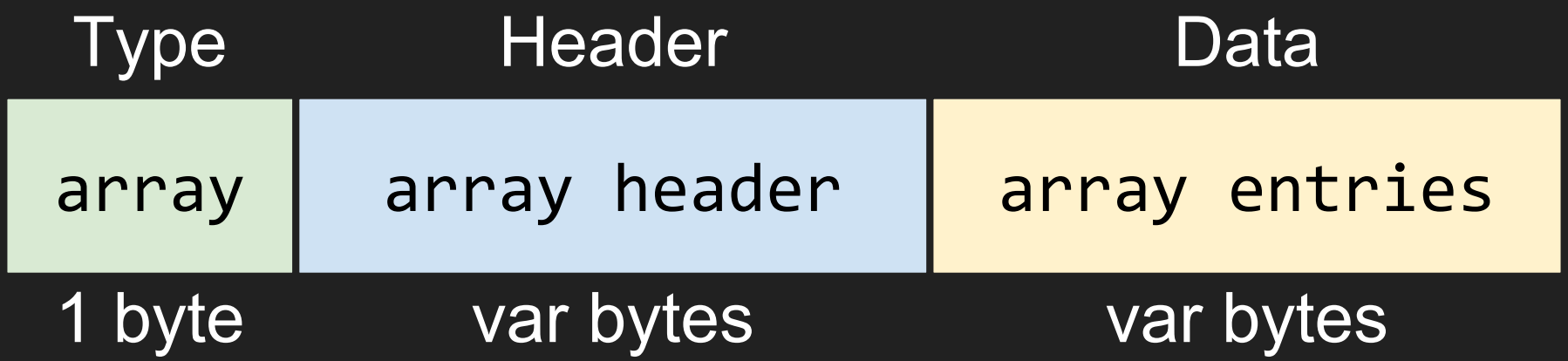


Composites

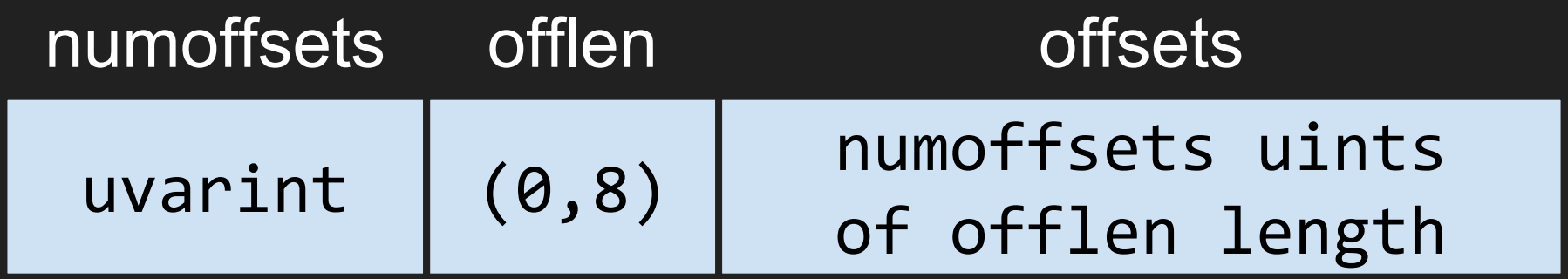
Recursive - contained data are defined by this same format

- Array
- Map

Composite: Array



Composite: Array - Header



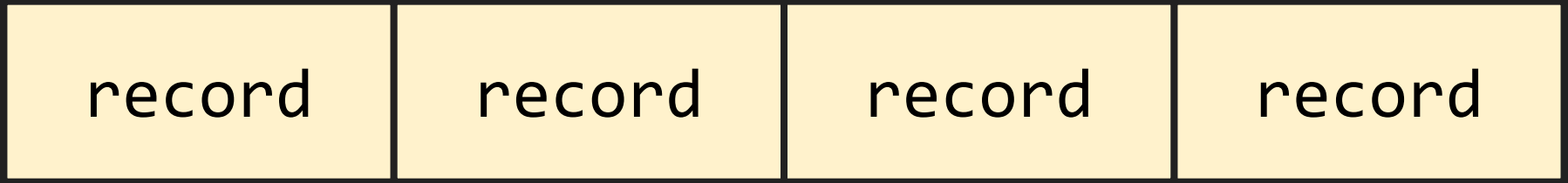
Composite: Array - Header offsets

2 or more offsets

Each offlen bytes

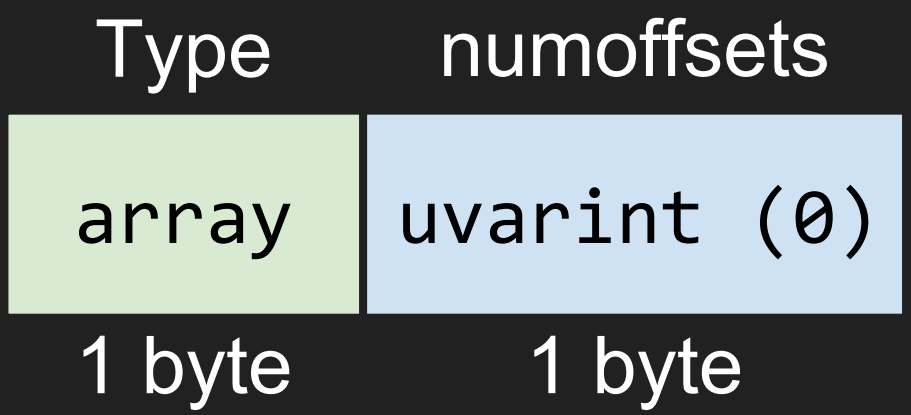
Composite: Array - Data

1 or more records



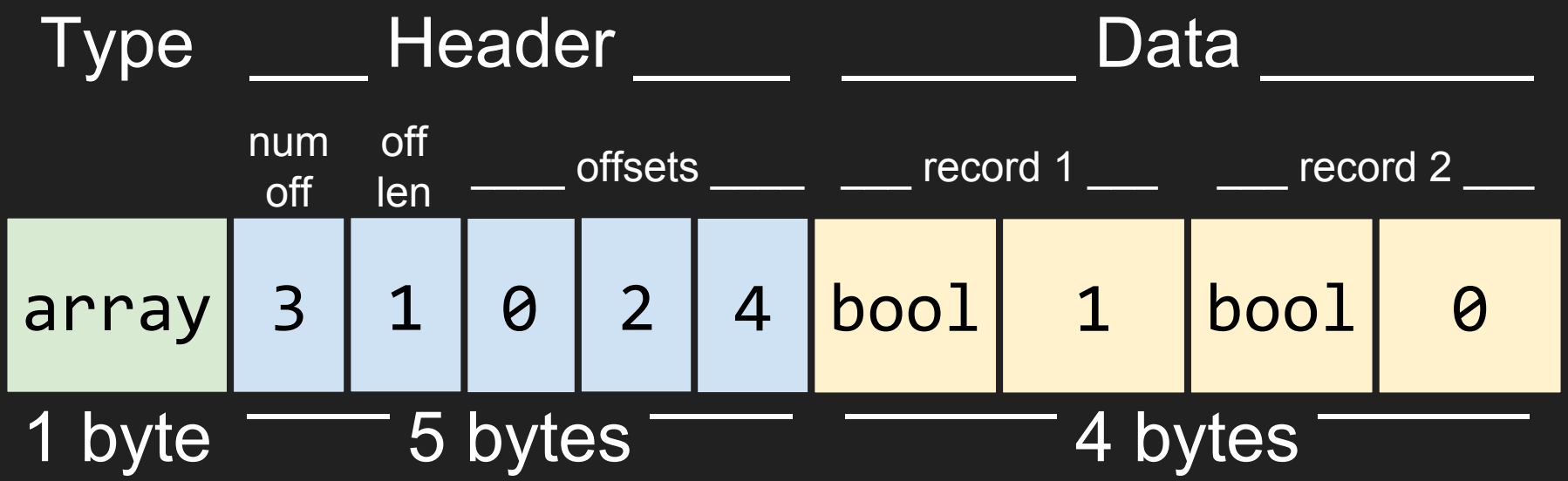
Each var bytes

Composite: Empty Array



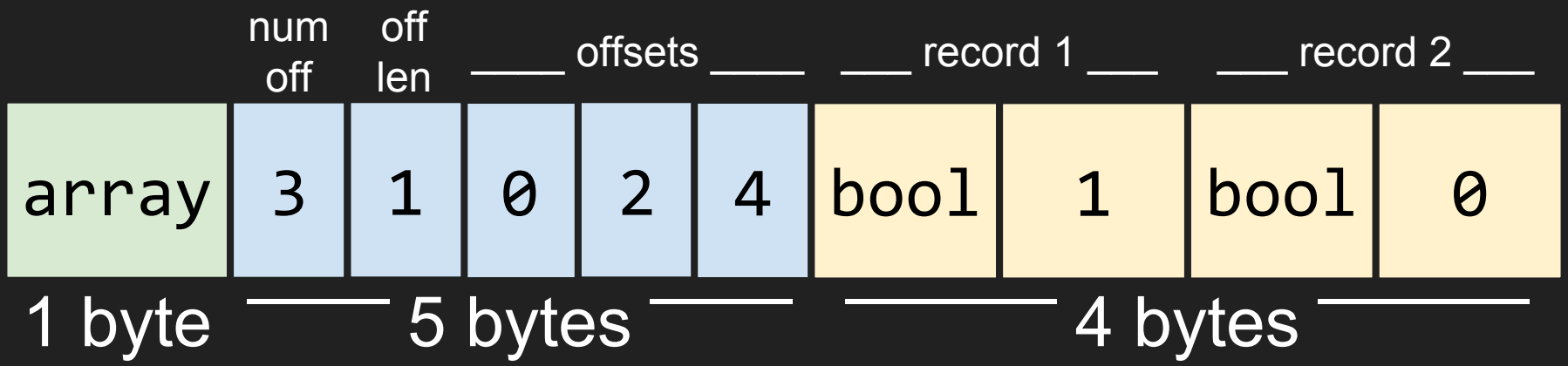
Composite: Array (example)

`[true, false]`

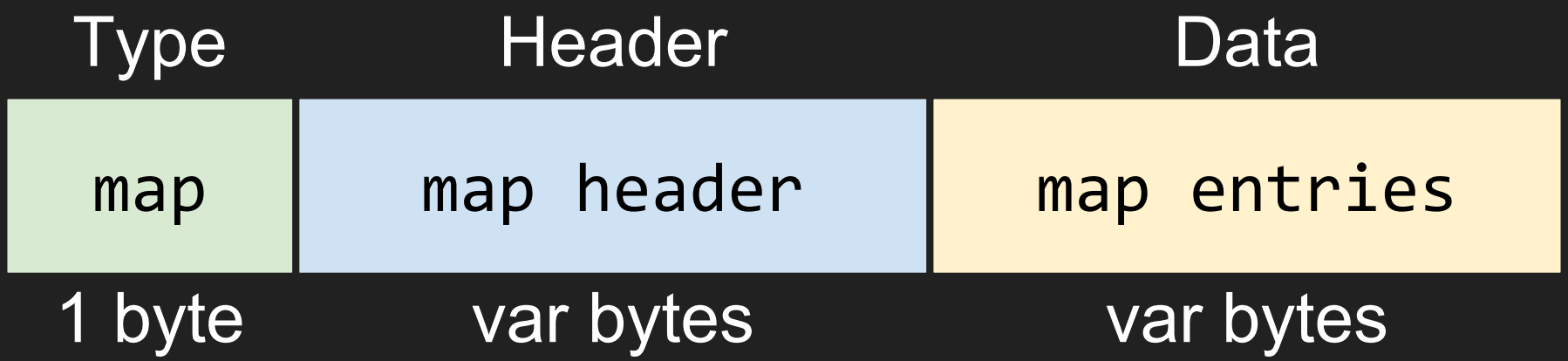


Composite: Array (example, slicing)

`[true, false]`



Composite: Map

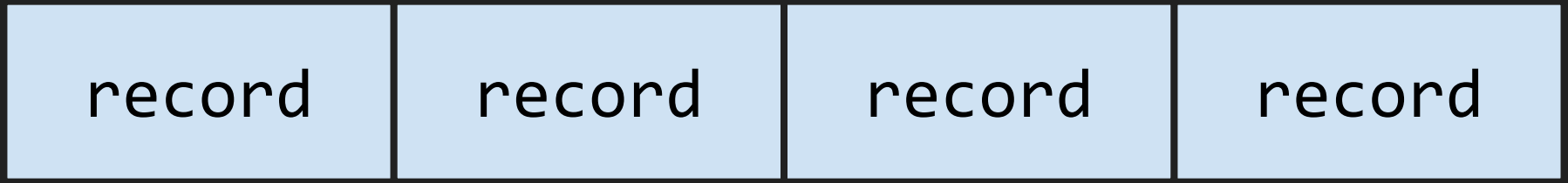


Composite: Map - Header

num recs	offlen	lenlen	header records
`uvarint`	`(0,8)`	`(0,8)`	`num recs` `header records`
var bytes	1 byte	1 byte	∝num recs

Composite: Map - Header

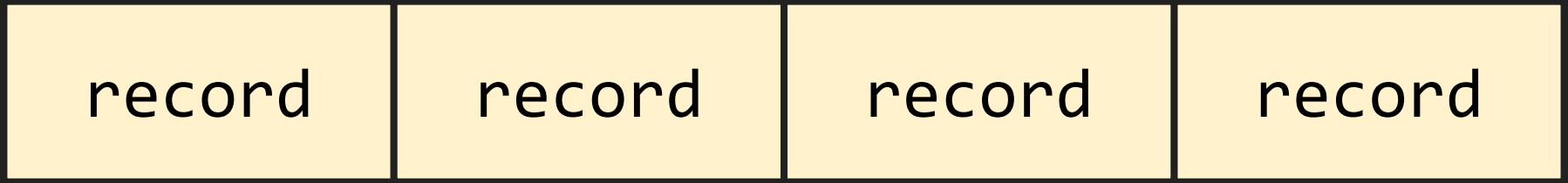
1 or more header records



Each 4 + offlen + lenlen bytes

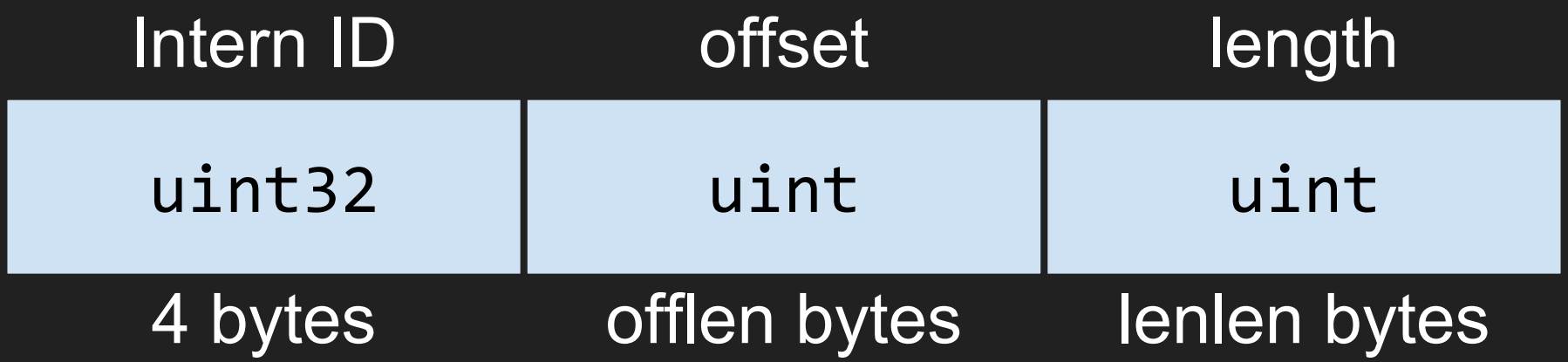
Composite: Map - Data

1 or more records



Each var bytes

Composite: Map - Header Record



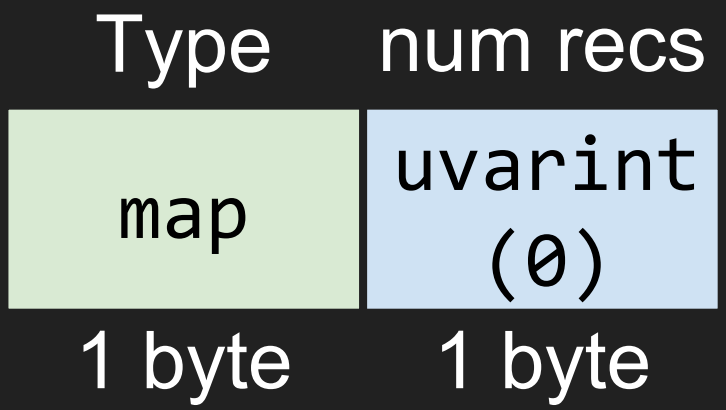
Composite: Map - Interned Keys

- Map keys are assigned a unique uint32 ID
- IDs are shared by identical strings
- Forward and reverse mappings stored next to the data
- Example:
  - `"true"  → 1`
  - `"false" → 2`

Composite: Map - Header

header records

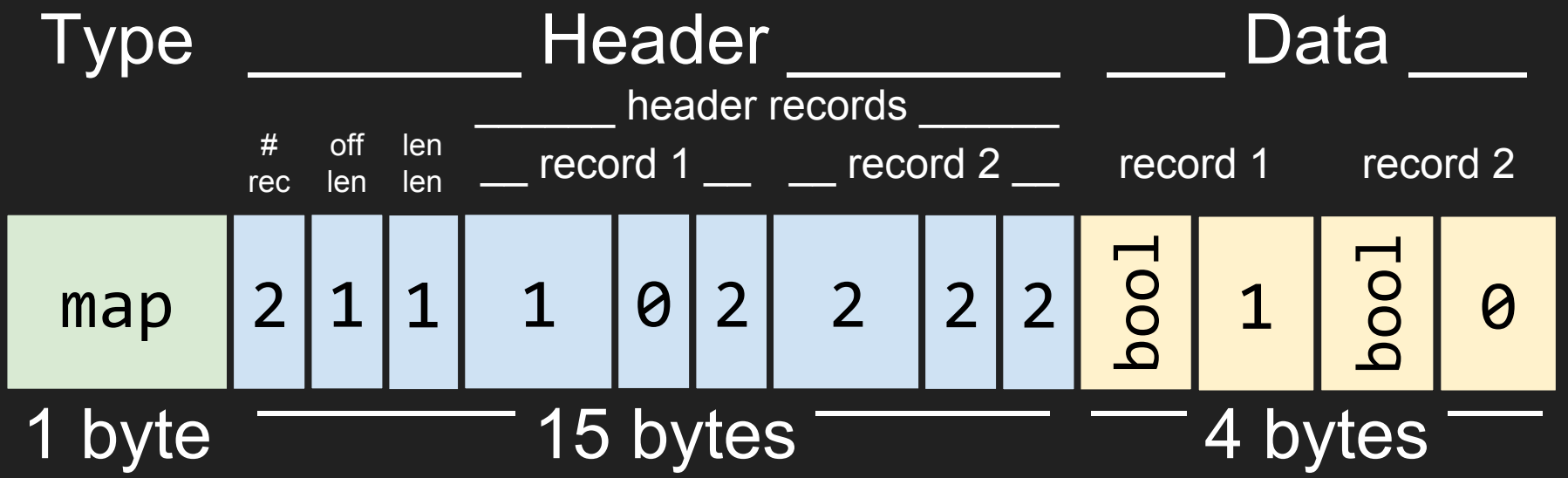
Composite: Empty Map



```
"true"  → 1
"false" → 2
```

Composite: Map (example)

```
{"false":false, "true":true}
```



NETFLIX

4. Performance

How fast is it?

It depends

… on:

- How much data you ask for
- How complex the query is
- How many CPU's
- Speed of the underlying data storage

Scalars

Serialize

Type	time/op	
Null	64.3 ns ± 2%	
Boolean	71.6 ns ± 1%	
Int	75.7 ns ± 0%	
Float	75.4 ns ± 1%	
String	88.6 ns ± 1%	"foobar"

Deserialize

Type	time/op
Null	16.0 ns ± 1%
Boolean	23.9 ns ± 1%
Int	26.6 ns ± 1%
Float	27.1 ns ± 1%
String	70.1 ns ± 1%

NETFLIX

Composites: Serialize

Type	# elems	time/op	time/op (ns)
Array	0	115 ns ± 0%	115 ns
Array	1	273 ns ± 1%	273 ns
Array	10	900 ns ± 1%	900 ns
Array	100	5.42 µs ± 1%	5420 ns
Array	1000	43.7 µs ± 1%	43700 ns
Array	10000	453 µs ± 1%	453000 ns
Array	100000	5.35 ms ± 1%	5350000 ns
Array	1000000	54.0 ms ± 3%	54000000 ns
Map	0	87.2 ns ± 1%	87 ns
Map	1	608 ns ± 1%	608 ns
Map	10	3.39 µs ± 1%	3390 ns
Map	100	34.1 µs ± 1%	34100 ns
Map	1000	374 µs ± 0%	374000 ns
Map	10000	4.37 ms ± 1%	4370000 ns
Map	100000	58.7 ms ± 2%	58700000 ns
Map	1000000	866 ms ± 4%	866000000 ns

Composites: Deserialize

Type	# elems	time/op	time/op (ns)
Array	0	136 ns ± 1%	136 ns
Array	1	201 ns ± 0%	201 ns
Array	10	588 ns ± 2%	588 ns
Array	100	4.05 µs ± 3%	4050 ns
Array	1000	38.1 µs ± 1%	38100 ns
Array	10000	380 µs ± 2%	380000 ns
Array	100000	3.81 ms ± 1%	3810000 ns
Array	1000000	39.9 ms ± 2%	39900000 ns
Map	0	158 ns ± 0%	158 ns
Map	1	361 ns ± 0%	361 ns
Map	10	1.97 µs ± 0%	1970 ns
Map	100	21.3 µs ± 0%	21300 ns
Map	1000	261 µs ± 1%	261000 ns
Map	10000	2.67 ms ± 1%	2670000 ns
Map	100000	38.3 ms ± 2%	38300000 ns
Map	1000000	757 ms ± 3%	757000000 ns

NETFLIX

Composites: Queries

Type	# elems	time/op
Array Get	1	25.9 ns ± 7%
Array Get	10	26.4 ns ± 6%
Array Get	100	26.6 ns ± 6%
Array Get	1000	26.3 ns ± 6%
Array Get	10000	26.3 ns ± 8%
Array Get	100000	26.0 ns ± 4%
Array Get	1000000	26.2 ns ± 7%
Map Get	1	35.3 ns ± 1%
Map Get	10	64.7 ns ± 0%
Map Get	100	74.6 ns ± 1%
Map Get	1000	121 ns ± 1%
Map Get	10000	157 ns ± 0%
Map Get	100000	221 ns ± 2%
Map Get	1000000	375 ns ± 1%

Type	# elems	time/op
Array Slice	1	70.1 ns ± 1%
Array Slice	10	73.9 ns ± 4%
Array Slice	100	73.7 ns ± 3%
Array Slice	1000	73.0 ns ± 2%
Array Slice	10000	73.4 ns ± 3%
Array Slice	100000	75.6 ns ± 3%
Array Slice	1000000	73.4 ns ± 2%
Map Keys	1	662 ns ± 9%
Map Keys	10	2.11 µs ± 8%
Map Keys	100	17.4 µs ± 8%
Map Keys	1000	173 µs ± 8%
Map Keys	10000	2.28 ms ± 4%
Map Keys	100000	35.6 ms ± 5%
Map Keys	1000000	348 ms ± 7%

5. Future

NETFLIX

In Progress & Future Work

- Replace simple scalar values
- Append to arrays
- Add new keys to a map
- Other ops (inc, dec, etc)
- Compression

Thank You

@sgmansfield
smansfield@netflix.com
techblog.netflix.com

NETFLIX