
Meta

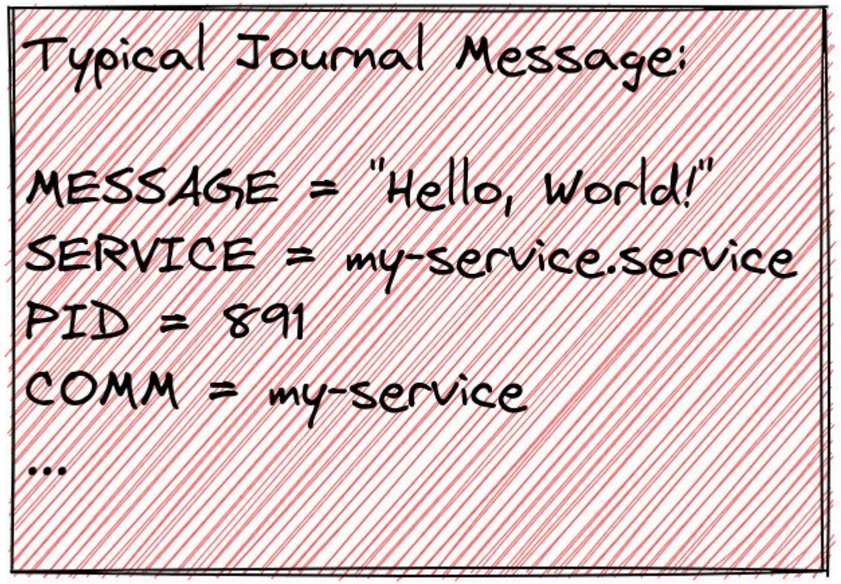
Slimming down the journal

Daan De Meyer
Software Engineer

Meta

Journal Messages

- Key = Value pairs
- MESSAGE field => Log message (unique)
- Other fields => Metadata (duplicated)
- Large fields are supported (theoretically)



Journal Format

Journal Format Requirements

- Should allow querying all messages with a specific field = value
  - Without a specific field = value ?
  - With/Without a specific field ?
- Should support storing large data in fields
- Should be resistant to corruption
- Should support in-line compression
- Should be space-efficient on disk

Entry
- Header
- Timestamp
- ...
- Boot ID

Items
- Hash
- Data offset
- Hash
- Data offset
- Hash
- Data offset
...

Field
- Header
- Head Data Offset
- ...

Payload
Field

Data
- Header
- Hash
- Entry Array Offset
- Next Data Offset

Payload
Field = Value

Entry Array
- Header
- Next Entry Array Offset

Items
- Entry offset
- Entry offset
- Entry offset
- Entry offset
...

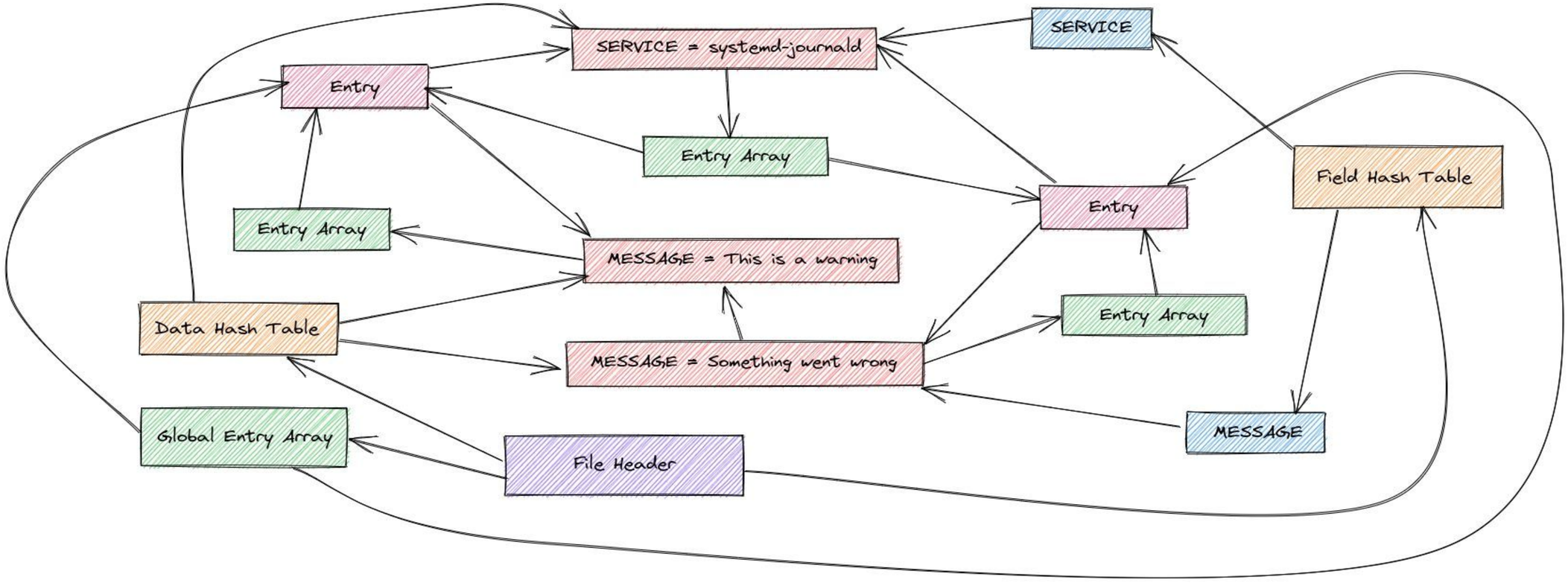
Hash Table
- Header

Items
- Hash Item
- Hash Item
- Hash Item
- Hash Item
...

Header
- Type
- Size
- Flags
- ...

Hash Item
- Head Hash Offset
- Tail Hash Offset

File Header
- Tail Object Offset
- Data Hash Table Offset
- Field Hash Table Offset
- Global Entry Array Offset
- ...



Journal Format Problems

- Much larger than the equivalent plaintext logs
  - Caused by all the extra metadata fields
  - Data object offset => 64-bit
  - Assuming 20 fields per entry => 120 bytes per entry => Larger than most log messages
- Grepping is much slower than the equivalent plaintext logs
  - Not discussed further in this talk

Optimizations

32-bit Object Offsets

- Issue: Object offsets => 64-bit
- Observation: Can be made 32-bit if offsets <= 4G
- Fix: Limit journal files to 4G
- Fix: Limit offsets to 32-bit where convenient
- Halves the size used by Entry Array objects

```
Before:

OBJECT TYPE       ENTRIES SIZE
Unused            0       0B
Data              3610336 595.7M
Field             5310    285.2K
Entry             3498326 1.2G
Data Hash Table   29      103.1M
Field Hash Table  29      151.3K
Entry Array       605991  1011.6M
Tag               0       0B
Total             7720021 2.9G

After:

OBJECT TYPE       ENTRIES SIZE
Unused            0       0B
Data              3562667 591.0M
Field             3971    213.6K
Entry             3498566 1.2G
Data Hash Table   20      71.1M
Field Hash Table  20      104.3K
Entry Array       582647  505.0M
Tag               0       0B
Total             7647891 2.4G
```

Relative Entry Seqnum

- Issue: Seqnum => 64-bit
- Observation: Journal files will never have billions of entries
- Fix: Store the entry object seqnum as a 32-bit offset to the header seqnum

Object Offset Trie

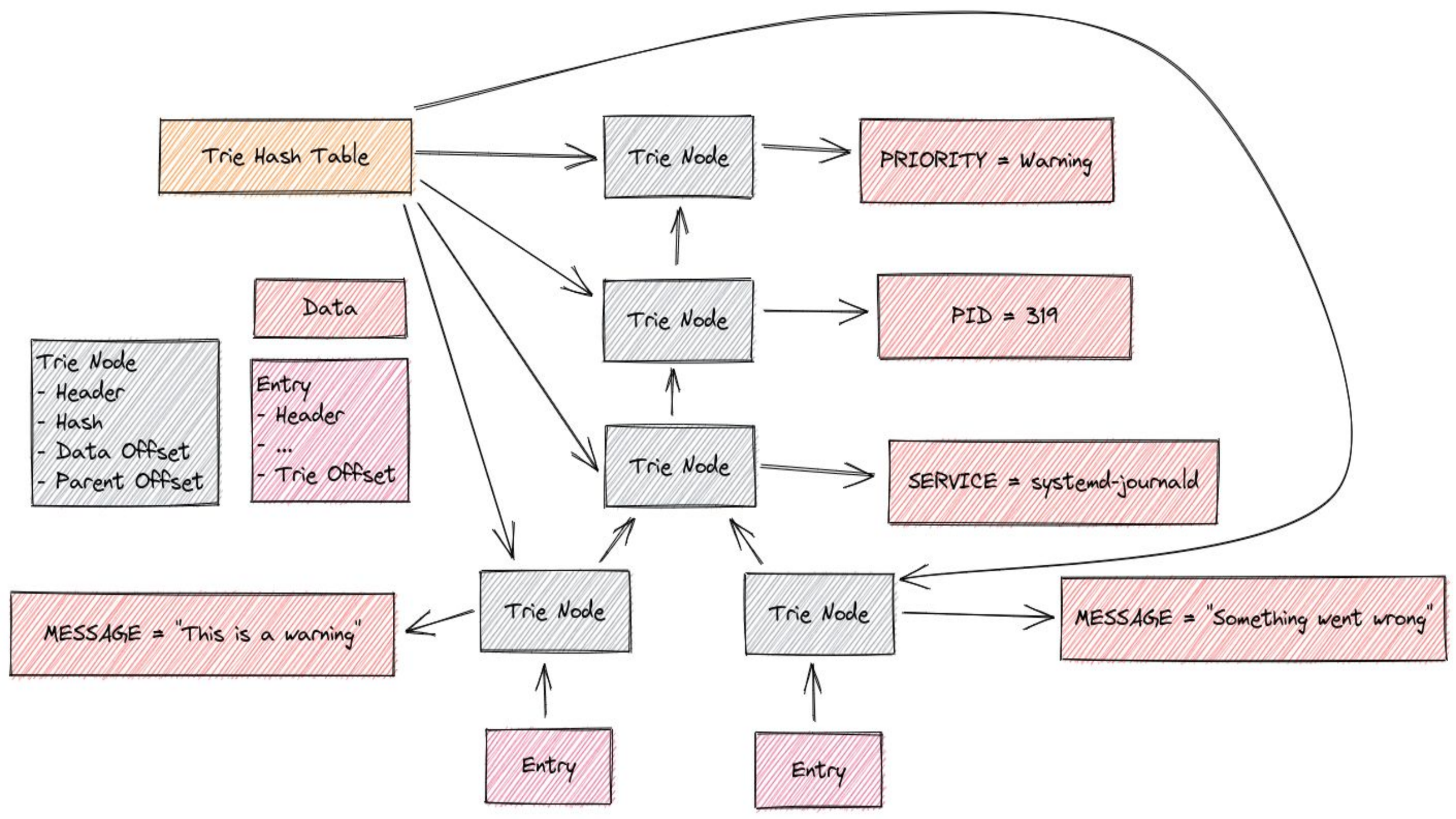
- Issue: Duplicated offsets in Entry objects
- Observation: Journal entries often include the same fields
- Fix: Use a trie to reduce offset duplication
- Disadvantage: Extra random accesses when looping over offsets

```
Before:

OBJECT TYPE      ENTRIES SIZE
Unused           0       0B
Data             3562667 591.0M
Field            3971    213.6K
Entry            3498566 1.2G
Data Hash Table  20      71.1M
Field Hash Table 20      104.3K
Entry Array      582647  505.0M
Tag              0       0B
Total            7647891 2.4G

After:

OBJECT TYPE      ENTRIES  SIZE
Unused           0        0B
Data             3521895  587.0M
Field            3140     169.4K
Entry            3499118  240.2M
Data Hash Table  14       49.7M
Field Hash Table 14       73.0K
Entry Array      577350   499.5M
Tag              0        0B
Trie Node        5767903  220.0M
Trie Hash Table  14       74.6M
Total            13369448 1.6G
```


Trie Hash Table
Trie Node
PRIORITY = Warning
Trie Node
PID = 319
Trie Node
SERVICE = systemd-journald
Trie Node
MESSAGE = "This is a warning"
Trie Node
MESSAGE = "Something went wrong"
Entry
Entry
Data
Trie Node
- Header
- Hash
- Data Offset
- Parent Offset
Entry
- Header
- ...
- Trie Offset

Cache tail entry arrays

- Issue: Appending to Entry Array chains requires iterating the chain
- Fix: Cache tail Entry Array object
- 73s => 40s to copy a 4G journal to compact mode

Configurable Field Deduplication

- Issue: Unique fields are always allocated their own Data object
- Fix: Add environment variable to configure field deduplication

```
Before:

OBJECT TYPE      ENTRIES  SIZE
Unused           0        0B
Data             3521895  587.0M
Field            3140     169.4K
Entry            3499118  240.2M
Data Hash Table  14       49.7M
Field Hash Table 14       73.0K
Entry Array      577350   499.5M
Tag              0        0B
Trie Node        5767903  220.0M
Trie Hash Table  14       74.6M
Total            13369448 1.6G

After:

OBJECT TYPE      ENTRIES SIZE
Unused           0       0B
Data             1022925 95.3M
Field            2808    151.3K
Entry            3499976 667.7M
Data Hash Table  13      46.2M
Field Hash Table 13      67.8K
Entry Array      492907  576.7M
Tag              0       0B
Trie Node        1758648 67.0M
Trie Hash Table  13      69.3M
Total            6777303 1.4G
```

Configurable Field Indexing

- Issue: All fields are indexed
- Fix: Add environment variable to configure field indexing

```
Before:

OBJECT TYPE      ENTRIES SIZE
Unused           0       0B
Data             1022925 95.3M
Field            2808    151.3K
Entry            3499976 667.7M
Data Hash Table  13      46.2M
Field Hash Table 13      67.8K
Entry Array      492907  576.7M
Tag              0       0B
Trie Node        1758648 67.0M
Trie Hash Table  13      69.3M
Total            6777303 1.4G

After:

SYSTEMD_JOURNAL_NON_INDEXED_FIELDS="_CAP_E

OBJECT TYPE      ENTRIES SIZE
Unused           0       0B
Data             987596  92.5M
Field            2127    114.8K
Entry            3500464 667.8M
Data Hash Table  9       32.0M
Field Hash Table 9       46.9K
Entry Array      241932  200.3M
Tag              0       0B
Trie Node        1752229 66.8M
Trie Hash Table  9       47.9M
Total            6484375 1.0G
```

Future

Dictionary Compression

- Trie is great for active journal files
- For archived journal files, we could do ZSTD dictionary compression of Entry item offsets
- Would require rewriting the journal file when archiving

Entry Array Concatenation

- If we rewrite a journal file when archiving, we can concatenate all Entry Array chains into single Entry Array objects

Questions?

THANK YOU FOR YOUR TIME

Daan De Meyer
twitter.com/daanjdemeyer