# MongoDB Internals

mongoDB

# Why Pop the Hood?

- Understanding data safety
- Estimating RAM / disk requirements
- Optimizing performance

# Storage Layout

mongoDB

# Directory Layout

```
drwxr-xr-x    136          Nov 19 10:12    journal
-rw-------    16777216     Oct 25 14:58    test.0
-rw-------    134217728    Mar 13  2012    test.1
-rw-------    268435456    Mar 13  2012    test.2
-rw-------    536870912    May 11  2012    test.3
-rw-------    1073741824   May 11  2012    test.4
-rw-------    2146435072   Nov 19 10:14    test.5
-rw-------    16777216     Nov 19 10:13    test.ns
```

# Directory Layout

- Aggressive pre-allocation (always 1 spare file)
- There is one namespace file per db which can hold 18000 entries per default
- A namespace is a collection or an index

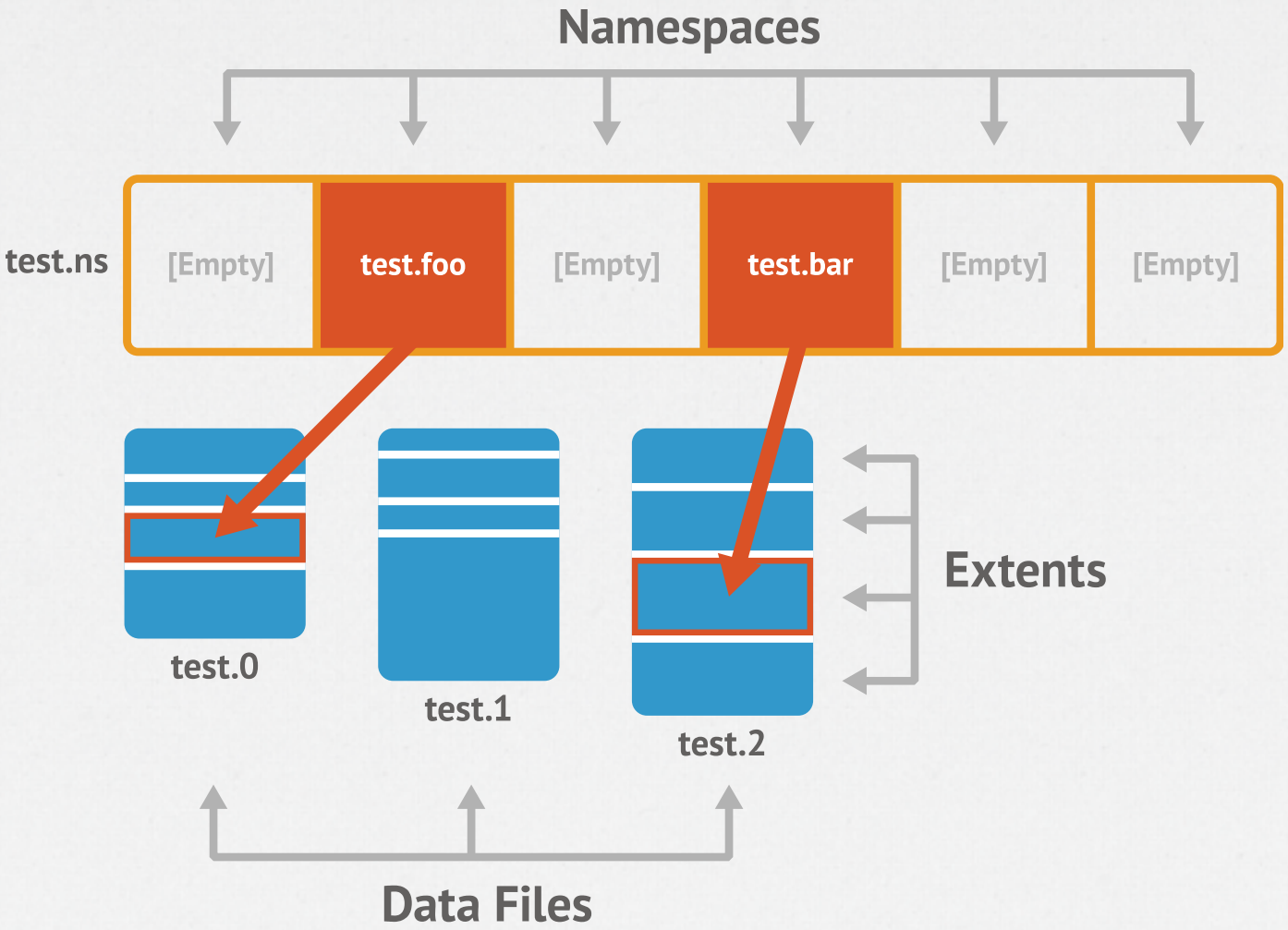
# Tuning with Options

- Use --*directoryperdb* to separate dbs into own folders which allows to use different volumes (isolation, performance)
- Use --*smallfiles* to keep data files smaller
- If **using many databases**, use –*nopreallocate* and --*smallfiles* to reduce storage size
- If **using thousands of collections & indexes**, increase namespace capacity with --*nssize*

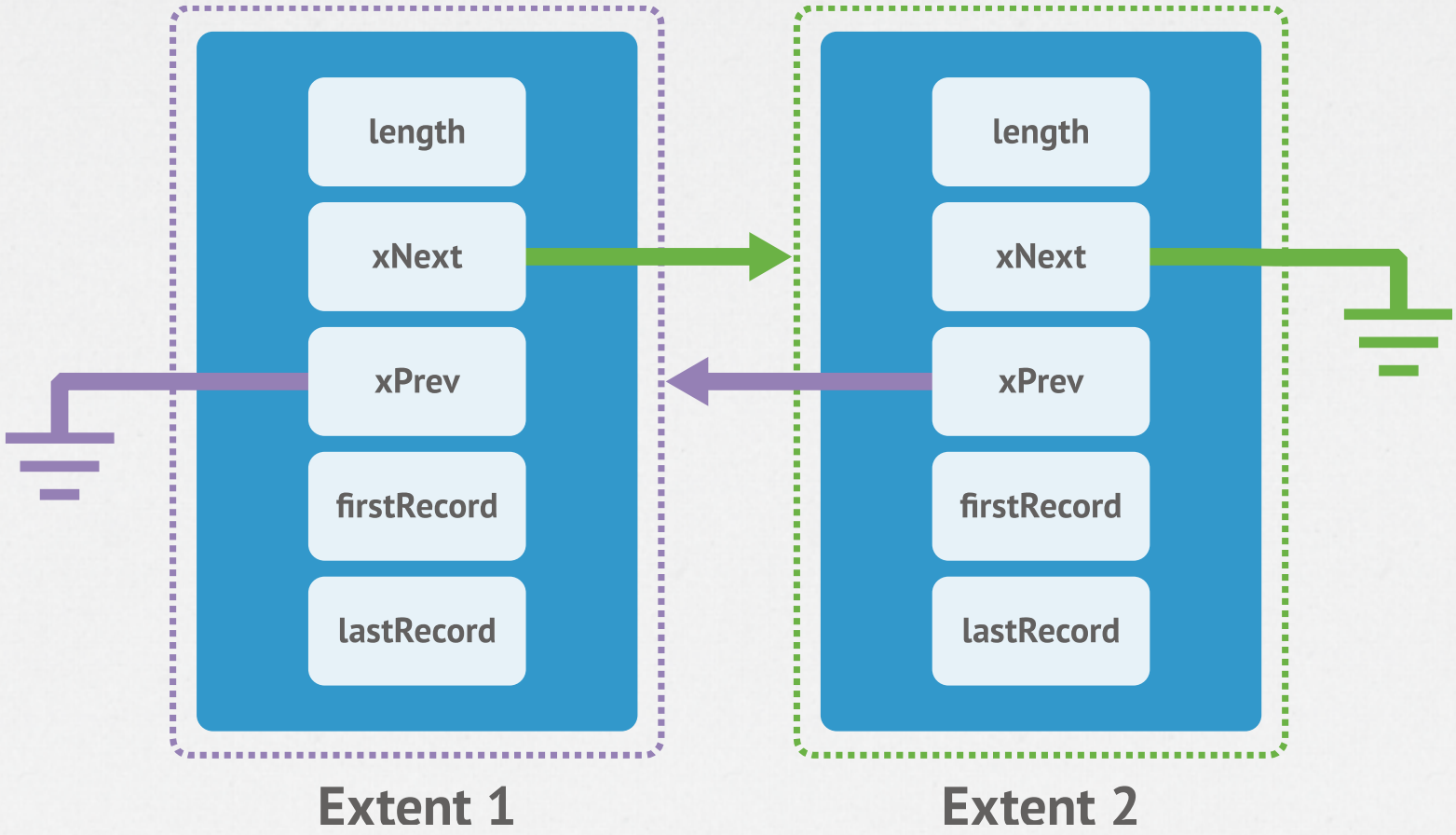
# Internal Structure

mongoDB

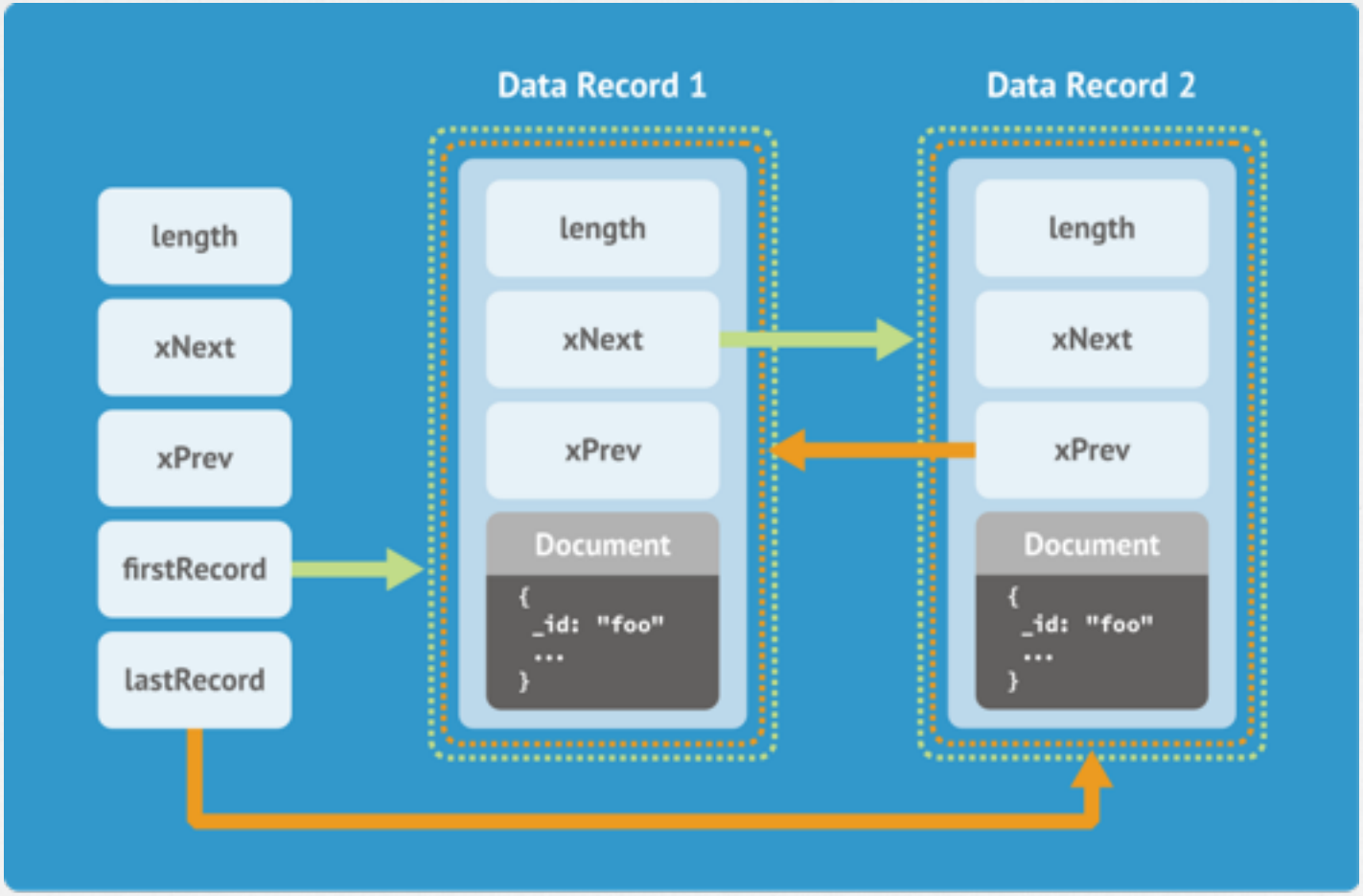
# Internal File Format



# Extent Structure



# Extents and Records



# To Sum Up: Internal File Format

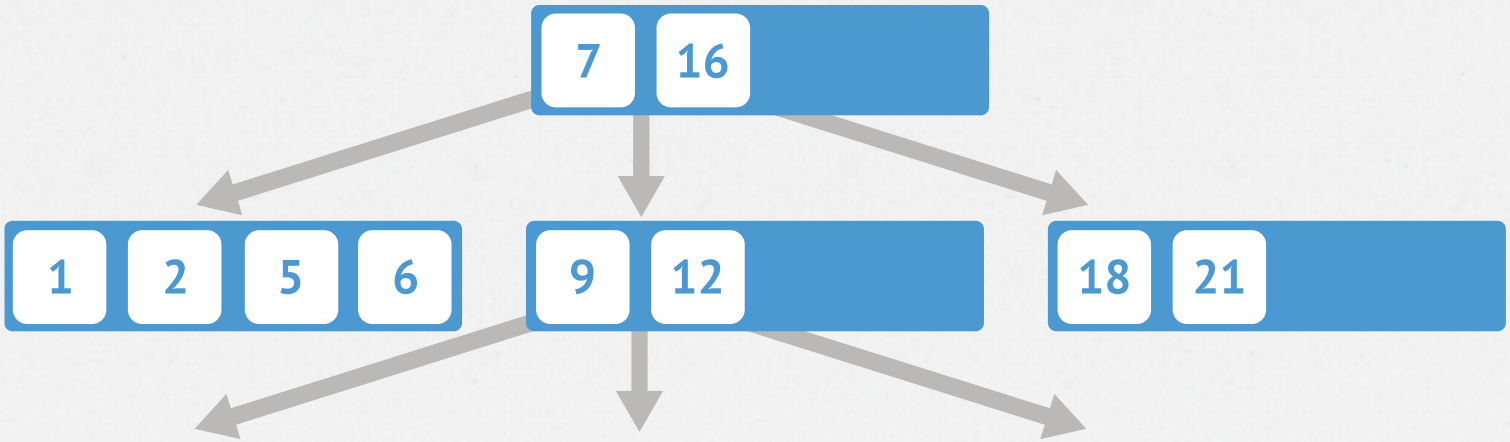
- Files on disk are broken into extents which contain the documents
- A collection has **one or more extents**
- Extent grow exponentially **up to 2GB**
- Namespace entries in the *ns* file point to the **first extent** for that collection

# What About Indexes?

mongoDB

# Indexes

- Indexes are **BTree** structures serialized to disk
- They are stored in the **same files as data** but using **own extents**



# The DB Stats

```
> db.stats()
{
    "db" : "test",
    "collections" : 22,
    "objects" : 17000383, ## number of documents
    "avgObjSize" : 44.336902762720l1,
    "dataSize" : 753744328,  ## size of data
    "storageSize" : 1159569408, ## size of all containing
extents
    "numExtents" : 81,
    "indexes" : 85,
    "indexSize" : 624204896, ## separate index storage
size
    "fileSize" : 4176478208, ## size of data files on disk
    "nsSizeMB" : 16,
    "ok" : 1
}
```

# The Collection Stats

```
> db.large.stats()
{
    "ns" : "test.large",
    "count" : 5000000, ## number of documents
    "size" : 280000024, ## size of data
    "avgObjSize" : 56.0000048,
    "storageSize" : 409206784, ## size of all containing
extents
    "numExtents" : 18,
    "nindexes" : 1,
    "lastExtentSize" : 74846208,
    "paddingFactor" : 1, ## amount of padding
    "systemFlags" : 0,
    "userFlags" : 0,
    "totalIndexSize" : 162228192, ## separate index storage
size
    "indexSizes" : {
        "_id_" : 162228192
    },
    "ok" : 1
}
```

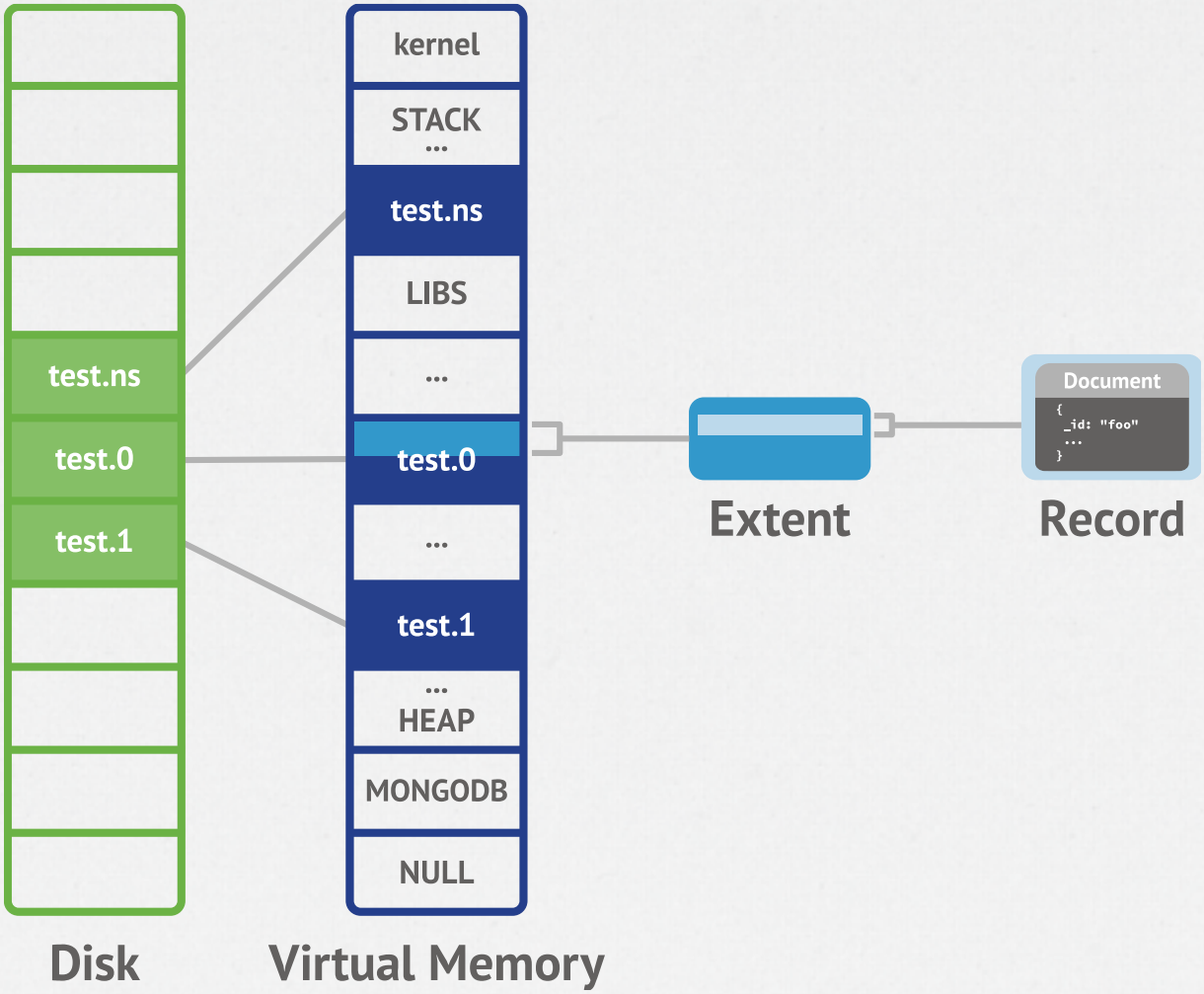
# What's Memory Mapping?

mongoDB

# Memory Mapped Files

- All data files are **memory mapped** to Virtual Memory by the OS
- MongoDB **just reads / writes to RAM** in the filesystem cache
- **OS takes care of the rest!**
- Virtual process size = total files size + overhead (connections, heap)
- If journal is on, the virtual size will be roughly doubled

# Virtual Address Space



# Memory Map, Love It or Hate It

- Pros:
  - No complex memory / disk code in MongoDB, **huge win!**
  - The OS is very good at caching for **any type** of storage
  - Least Recently Used behavior
  - Cache **stays warm** across MongoDB restarts
- Cons:
  - RAM usage is affected by disk **fragmentation**
  - RAM usage is affected by high **read-ahead**
  - LRU behavior does not prioritize things (like indexes)

# How Much Data is in RAM?

- **Resident memory** the **best indicator** of how much data in RAM
- Resident is: process overhead (connections, heap) + FS pages in RAM that were accessed
- Means that it resets to 0 upon restart even though **data is still in RAM** due to FS cache
- Use *free* command to check on FS cache size
- Can be affected by **fragmentation** and **read-ahead**

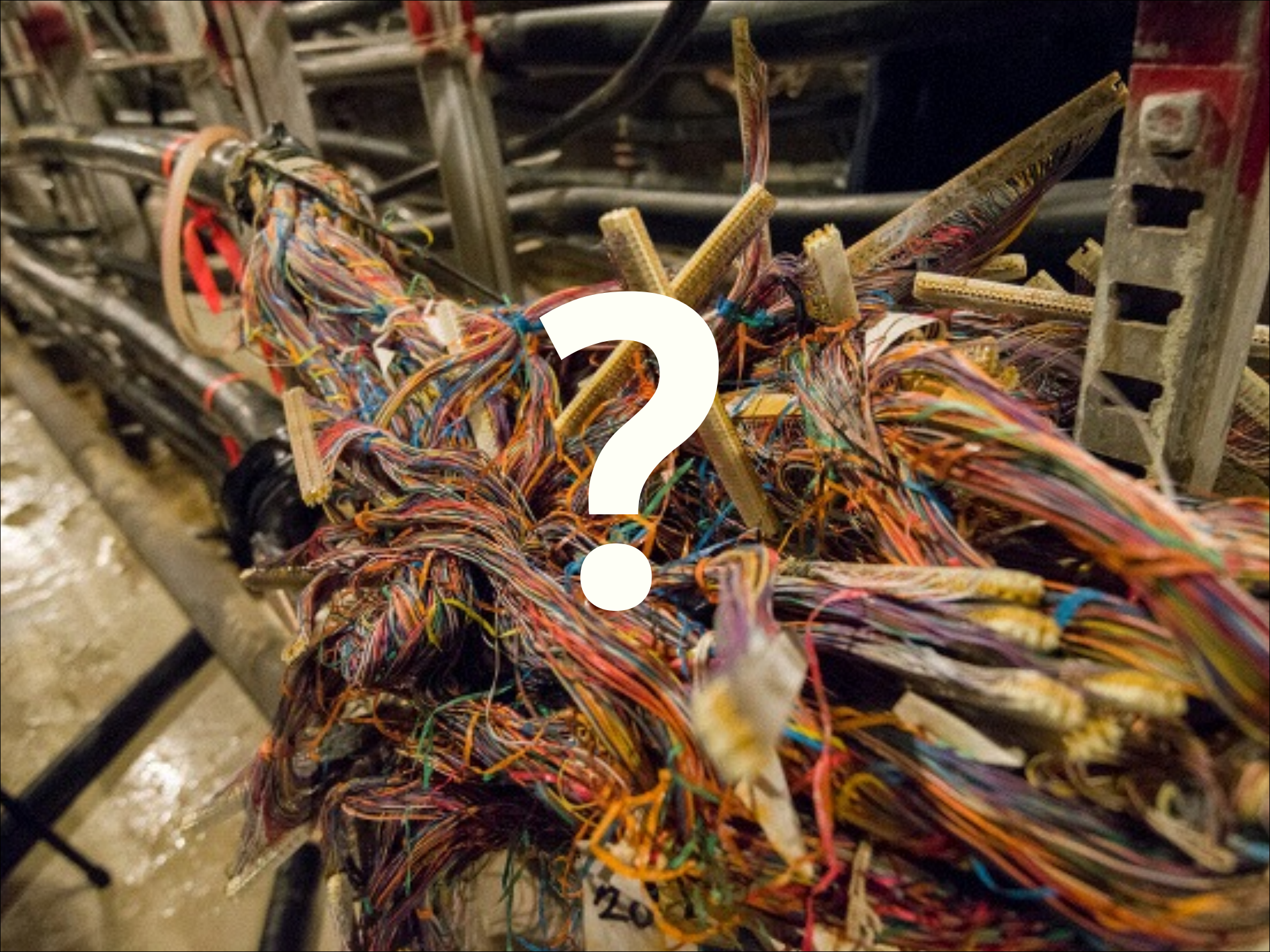
# Journaling

mongoDB

# The Problem

Changes in memory mapped files are not applied in order and different parts of the file can be from different points in time

You want a consistent point-in-time snapshot when restarting after a crash

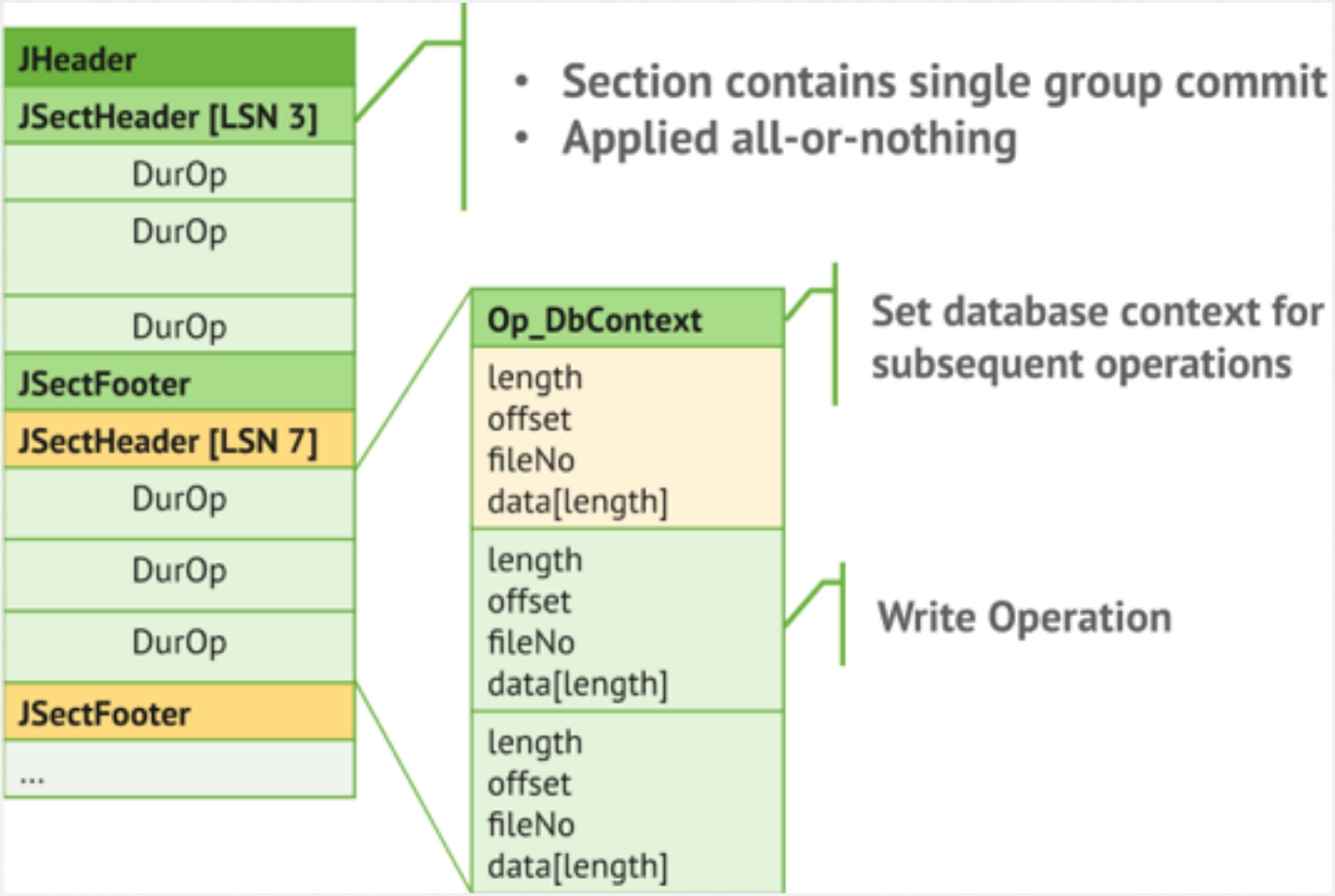



corruption

# Solution – Use a Journal

- Data gets written to a journal before making it to the data files
- Operations written to a journal buffer in RAM that gets flushed every 100ms by default or 100MB
- Once the **journal** is written to disk, the **data** is safe
- Journal **prevents corruption** and allows **durability**
- Can be turned off, **but don't!**

# Journal Format



# Can I Lose Data on a Hard Crash?

- Maximum data loss is 100ms (journal flush). This can be reduced with *–journalCommitInterval*
- For **durability** (data is on disk when ack'ed) use the JOURNAL_SAFE write concern ("j" option).
- Note that replication can reduce the data loss further. Use the REPLICAS_SAFE write concern ("w" option).
- As write guarantees **increase**, latency **increases**. To maintain performance, use **more connections**!

# What is the Cost of a Journal?

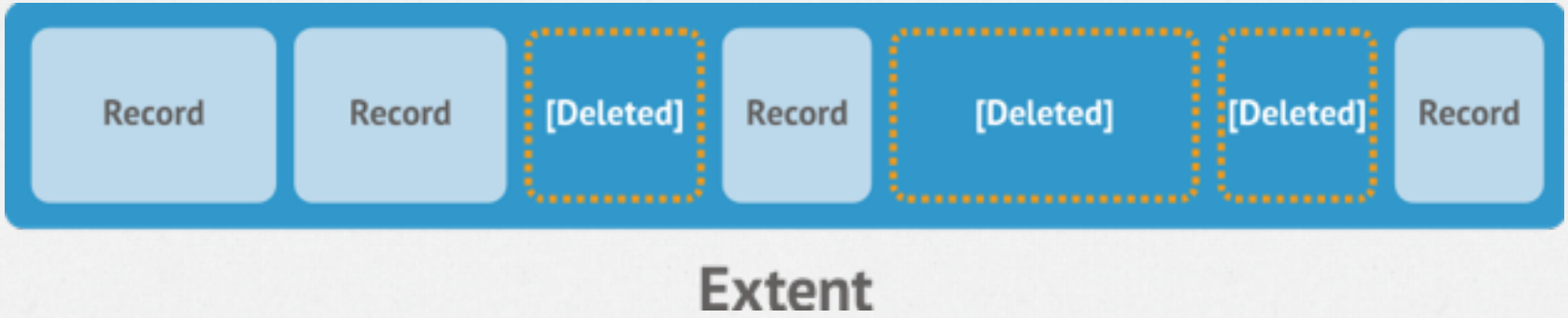
- On read-heavy systems, **no impact**
- Write performance is **reduced by 5-30%**
- If using separate drive for journal, **as low as 3%**
- For apps that are write-heavy (1000+ writes per server) there can be slowdown due to mix of journal and data flushes. Use **a separate drive**!

# Fragmentation

mongoDB

# What it Looks Like

Both on disk **and** in RAM!



# Fragmentation

- Files can get fragmented over time if *remove()* and *update()* are issued.
- It gets worse if documents have varied sizes
- Fragmentation **wastes disk space and RAM**
- Also makes writes **scattered and slower**
- Fragmentation can be checked by comparing *size* to *storageSize* in the collection's *stats*.

# How to Combat Fragmentation

- *compact* command (maintenance op)
- **Normalize** schema more (documents don't grow)
- **Pre-pad** documents (documents don't grow)
- Use **separate collections** over time, then use *collection.drop()* instead of *collection.remove(query)*
- *--usePowerOf2sizes* option makes disk buckets more reusable

# In Review

mongoDB

# In Review

- Understand disk layout and footprint
- See how much data is actually in RAM
- Memory mapping is cool
- Answer how much data is ok to lose
- Check on fragmentation and avoid it
- https://github.com/10gen-labs/storage-viz