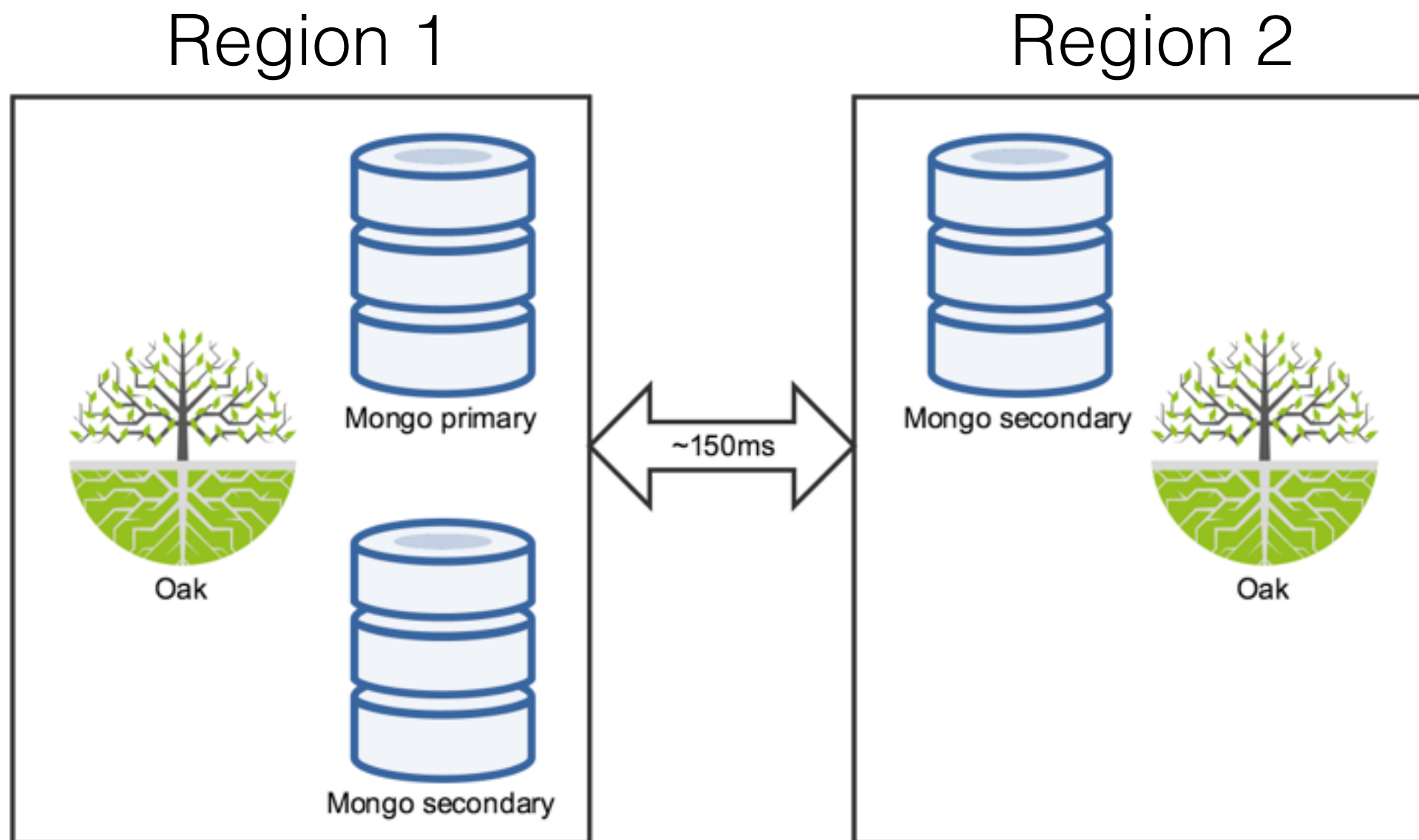


# Improvements for clustered Oak setup

OAK-3865 & OAK-4412

# Cluster setup



# Keep it local

- We'll focus on optimising the “remote” instance in region 2
- Avoid reading from primary:
  - use cache
  - **read from the local secondary - OAK-3865**
- Avoid writing
  - bulk updates ✓
  - local indexing data - OAK-4412

# Secondary read preference

- It isn't possible to always read from the secondary
- It may not contain the most recent changes that we're expecting
  - changes done by the local instance
  - external changes that we've already seen

# Current state

- We'll read a document from the secondary instance if:
  - the parent is already cached and
  - the parent is at least 6h old
- It means that the document hasn't been modified for 6h, so probably it has been already replicated

# Proposal - OAK-3865

- Each Oak instance keeps track of the most recent revisions it has ever seen
- There's a background process that connects to each secondary Mongo instance directly
- If the lastRev map of root is as recent as the most recent revision we have ever seen, **it's safe to use the secondary**
- The secondary doesn't have to be perfectly up-to-date - it only has to contain the changes we're already aware of

These are the latest revisions accessed by the Oak instance C<sub>1</sub>. The read preference can be set to “secondary” only if all the secondary Mongos already contains these revisions.

Oak instance C<sub>1</sub>

C <sub>1</sub> :	R <sub>1.05</sub>
C <sub>2</sub> :	R <sub>2.12</sub>
C <sub>3</sub> :	R <sub>3.56</sub>

The secondary 2 doesn't contain R<sub>1.05</sub>, so we can't use the “secondary” read preference on Oak C<sub>1</sub>.

Mongo primary

C <sub>1</sub> :	R <sub>1.01</sub>	R <sub>1.02</sub>	R <sub>1.03</sub>	R <sub>1.04</sub>	R <sub>1.05</sub>
C <sub>2</sub> :	R <sub>2.12</sub>	R <sub>2.13</sub>	R <sub>2.14</sub>	R <sub>2.15</sub>	R <sub>2.16</sub>
C <sub>3</sub> :	R <sub>3.56</sub>	R <sub>3.57</sub>	R <sub>3.58</sub>	R <sub>3.59</sub>	R <sub>3.60</sub>

Mongo secondary 1

C <sub>1</sub> :	R <sub>1.01</sub>	R <sub>1.02</sub>	R <sub>1.03</sub>	R <sub>1.04</sub>	R <sub>1.05</sub>
C <sub>2</sub> :	R <sub>2.12</sub>	R <sub>2.13</sub>	R <sub>2.14</sub>	R <sub>2.15</sub>	R <sub>2.16</sub>
C <sub>3</sub> :	R <sub>3.56</sub>	R <sub>3.57</sub>	R <sub>3.58</sub>	R <sub>3.59</sub>	R <sub>3.60</sub>

Mongo secondary 2

C <sub>1</sub> :	R <sub>1.01</sub>	R <sub>1.02</sub>	R <sub>1.03</sub>	R <sub>1.04</sub>	R <sub>1.05</sub>
C <sub>2</sub> :	R <sub>2.12</sub>	R <sub>2.13</sub>	R <sub>2.14</sub>	R <sub>2.15</sub>	R <sub>2.16</sub>
C <sub>3</sub> :	R <sub>3.56</sub>	R <sub>3.57</sub>	R <sub>3.58</sub>	R <sub>3.59</sub>	R <sub>3.60</sub>

Mongo instances contains document revisions created by 3 Oaks. We can get the current Mongo state reading the `_lastRevs` for the root document.

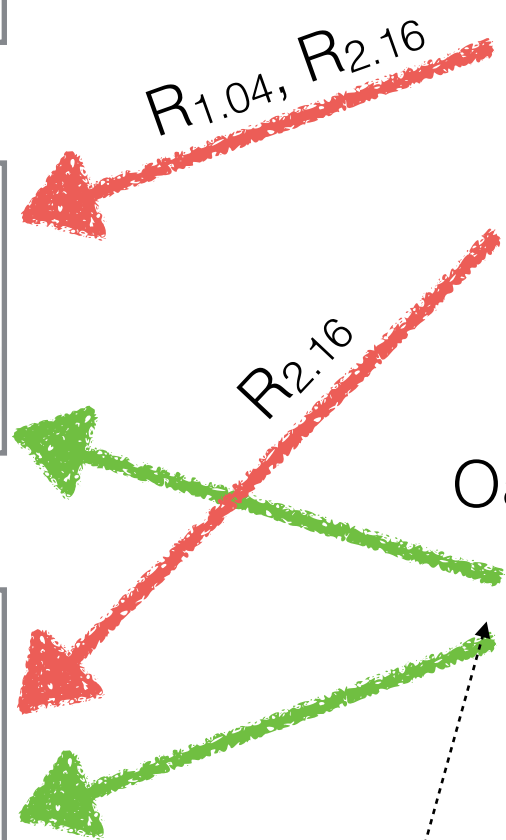
Oak instance C<sub>2</sub>

C <sub>1</sub> :	R <sub>1.04</sub>
C <sub>2</sub> :	R <sub>2.16</sub>
C <sub>3</sub> :	R <sub>3.58</sub>

Oak instance C<sub>3</sub>

C <sub>1</sub> :	R <sub>1.02</sub>
C <sub>2</sub> :	R <sub>2.13</sub>
C <sub>3</sub> :	R <sub>3.60</sub>

All secondaries are up-to-date with the current state of repository from the C<sub>3</sub> perspective. We can use the “secondary” read preference here.



# Document age

- An extra benefit - we can improve the implementation of the `DocumentStore#find(..., maxAge)` support
- The background tracker process provides the maximum age of documents on each secondary MongoDB
- We can exactly decide whether the `maxAge` constraint applies to the secondary instance (and it can be used) or not (and we have to go to primary)



# Local changes improvement

- We can track local changes separately, storing their document id and revision
- When `find(id)` is called and the id is saved in the local changes tracker, we can check if it has been already replicated to all secondaries
- If so, it can be removed from the tracker and the secondary instance can be used

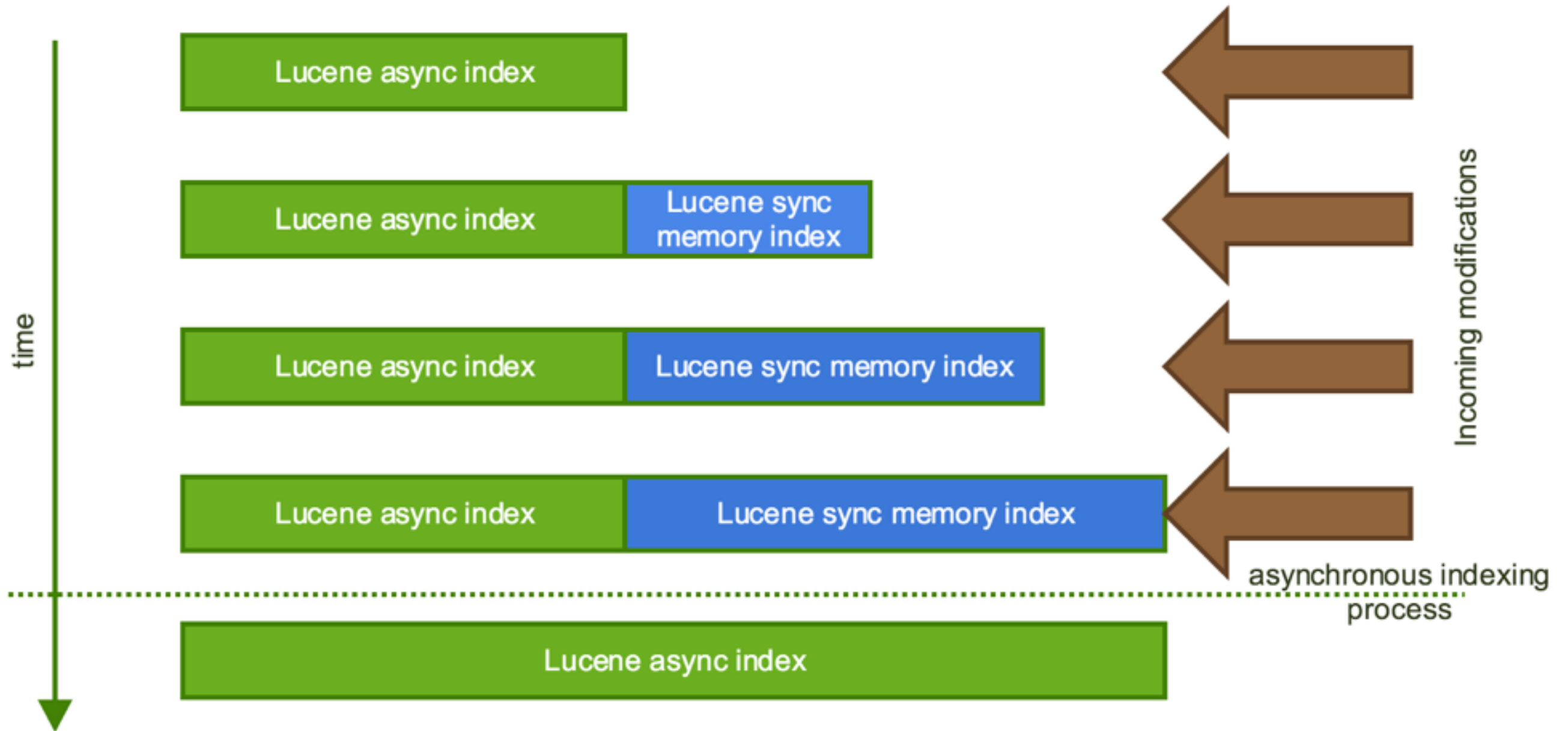
# Keep it local

- We'll focus on the Singapore instance
- Avoid reading from primary:
  - use cache
  - read from the local secondary - OAK-3865
- Avoid writing
  - bulk updates ✓
  - **local indexing data - OAK-4412**

# Synchronous index

- Property indexing process is a heavy writer
- Usually it works synchronously (within the client HTTP request)
- We can't replace it with an asynchronous Lucene cache as the customer code expects the changes are visible immediately in the JCR query results

# Hybrid approach



# Local Lucene index

- The local index can be stored either in memory or in local files (Lucene provides a MMAP solution)
- Observer is used to update the index with local and external changes

# Index definitions

- The index definition node is used to store the indexing metadata
- To keep the local indexing off-the-repository, an in-memory branch of the indexing definitions is created