Advanced usage of indexes in Oracle Coherence

Alexey Ragozin
alexey.ragozin@gmail.com
Nov 2011

Presentation overview

- Structure of Coherence index
- How `IndexAwareFilter` works
- Multiple indexes in same query
- Custom index provider API (since 3.6)
- Embedding Apache Lucene into data grid

Creation of index

Attribute extractor, used to identify index later

```
QueryMap.addIndex(
  ValueExtractor extractor,
  boolean ordered,
  Comparator comparator)
```

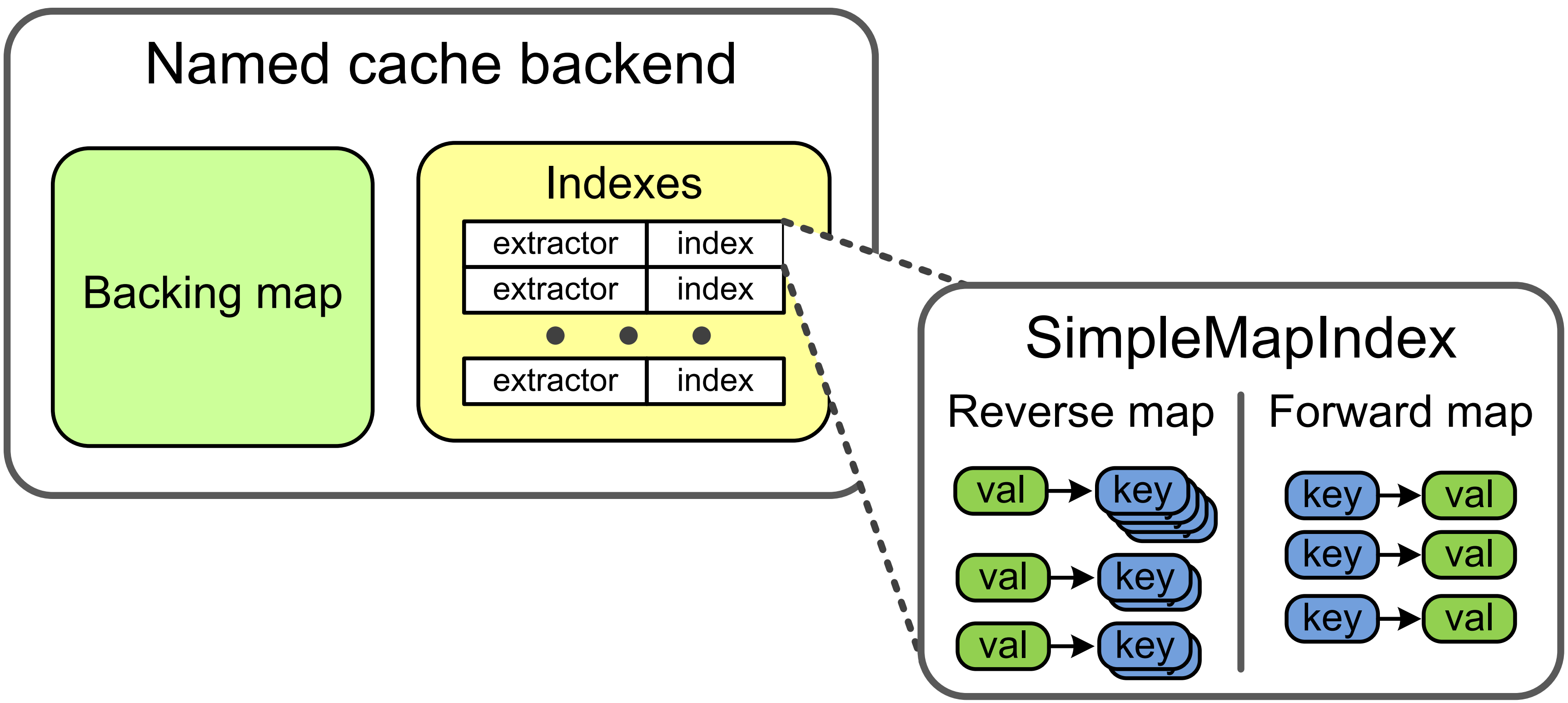
Index configuration

Using of query API

```
public interface QueryMap extends Map {
 Set keySet(Filter f);
 Set entrySet(Filter f);
 Set entrySet(Filter f, Comparator c);
 ...
}

public interface InvocableMap extends Map {
 Map invokeAll(Filter f, EntryProcessor agent);
 Object aggregate(Filter f, EntryAggregator agent);
 ...
}
```

Indexes at storage node



Indexes at storage node

- All indexes created on cache are stored in map

Custom extractors should obey `equals/hashCode` contract!

- Reverse map is used to speed up filters
- Forward map is used to speed up aggregators

`QueryMap.Entry.extract(…)` **is using index, if available**

Indexes at storage node

- Index structures are stored in heap
  - and may consume a lot of memory
- For partitioned scheme
  - keys in index are **binary blobs**,
  - **regular object**, otherwise
- Indexes will keep your key in heap even if you use off heap backing map
- Single index for all **primary partitions** of cache on single node

How filters use indexes?

```
interface IndexAwareFilter extends EntryFilter {
  int calculateEffectiveness(Map im, Set keys);
  Filter applyIndex(Map im, Set keys);
}
```

- applyIndex(…) is called by cache service on top level filter
- calculateEffectiveness(…) may be called by compound filter on nested filters
- each node executes index individually
- **For complex queries execution plan is calculated ad hoc, each compound filter calculates plan for nested filters**

Example: equalsFilter

Filter execution (call to applyIndex())

- Lookup for matching index using **extractor** instance as key
- If index found,
  - ✓ lookup index reverse map for **value**
  - ✓ intersect provided **candidate set** with **key set** from reverse map
  - ✓ return **null** – candidate set is accurate, no object filtering required
- else (no index found)
  - ✓ return **this** – all entries from **candidate set** should be deserialized and evaluated by filter

Multiple indexes in same query

Example: ticker=IBM & side=B

```
new AndFilter(
   new EqualsFilter("getTicker", "IBM"),
   new EqualsFilter("getSide", 'B'))
```

Execution plan

- call applyIndex(…) on first nested filter
  – only entries with ticker IBM are retained in candidate set
- call applyIndex(…) on second nested filter
  – only entries with side=B are retained in candidate set
- return candidate set

Index performance

PROs

- using of inverted index
- no deserialization overhead

CONs

- very simplistic cost model in index planner
- candidate set is stored in hash tables (intersections/unions may be expensive)
- high cardinality attributes may cause problems

Compound indexes

Example: ticker=IBM & side=B

- Index per attribute

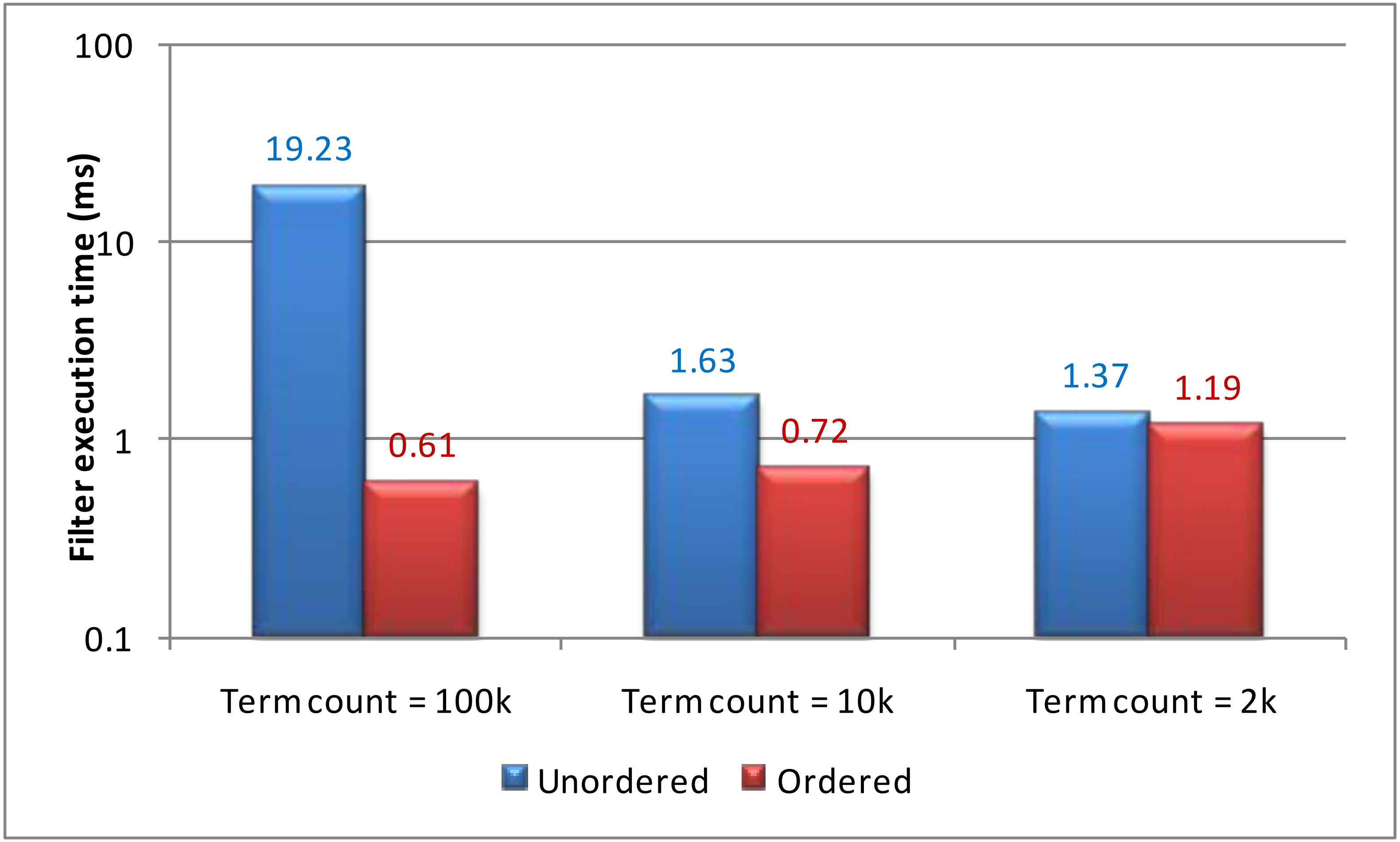
```
new AndFilter(
  new EqualsFilter("getTicker", "IBM"),
  new EqualsFilter("getSide", 'B'))
```

- Index for compound attribute

```
new EqualsFilter(
  new MultiExtractor("getTicker, getSide"),
  Arrays.asList(new Object[]{"IBM", 'B'}))
```

For index to be used, filter's extractor should match extractor used to create index!

Ordered indexes vs. unordered



Custom indexes since 3.6

```
interface IndexAwareExtractor
          extends ValueExtractor {

  MapIndex createIndex(
       boolean ordered,
       Comparator comparator,
       Map indexMap,
       BackingMapContext bmc);

  MapIndex destroyIndex(Map indexMap);
}
```

Ingredients of customs index

- Custom index extractor
- Custom index class (extends `MapIndex`)
- Custom filter, aware of custom index

+

- Thread safe implementation
- Handle both binary and object keys gracefully
- Efficient insert (index is updates synchronously)

Why custom indexes?

Custom index implementation is free to use any advanced data structure tailored for specific queries.

- NGram index – fast substring based lookup
- Apache Lucene index – full text search
- Time series index – managing versioned data

Using Apache Lucene in grid

Why?

- Full text search / rich queries
- Zero index maintenance

PROs

- Index partitioning by Coherence
- Faster execution of many complex queries

CONs

- Slower updates
- Text centric

Lucene example

Step 1. Create document extractor

```
// First, we need to define how our object will map
// to field in Lucene document
LuceneDocumentExtractor extractor = new LuceneDocumentExtractor();
extractor.addText("title", new ReflectionExtractor("getTitle"));
extractor.addText("author", new ReflectionExtractor("getAuthor"));
extractor.addText("content", new ReflectionExtractor("getContent"));
extractor.addText("tags", new ReflectionExtractor("getSearchableTags"));
```

Step 2. Create index on cache

```
// next create LuceneSearchFactory helper class
LuceneSearchFactory searchFactory = new LuceneSearchFactory(extractor);
// initialize index for cache, this operation actually tells coherence
// to create index structures on all storage enabled nodes
searchFactory.createIndex(cache);
```

Lucene example

Now you can use Lucene queries

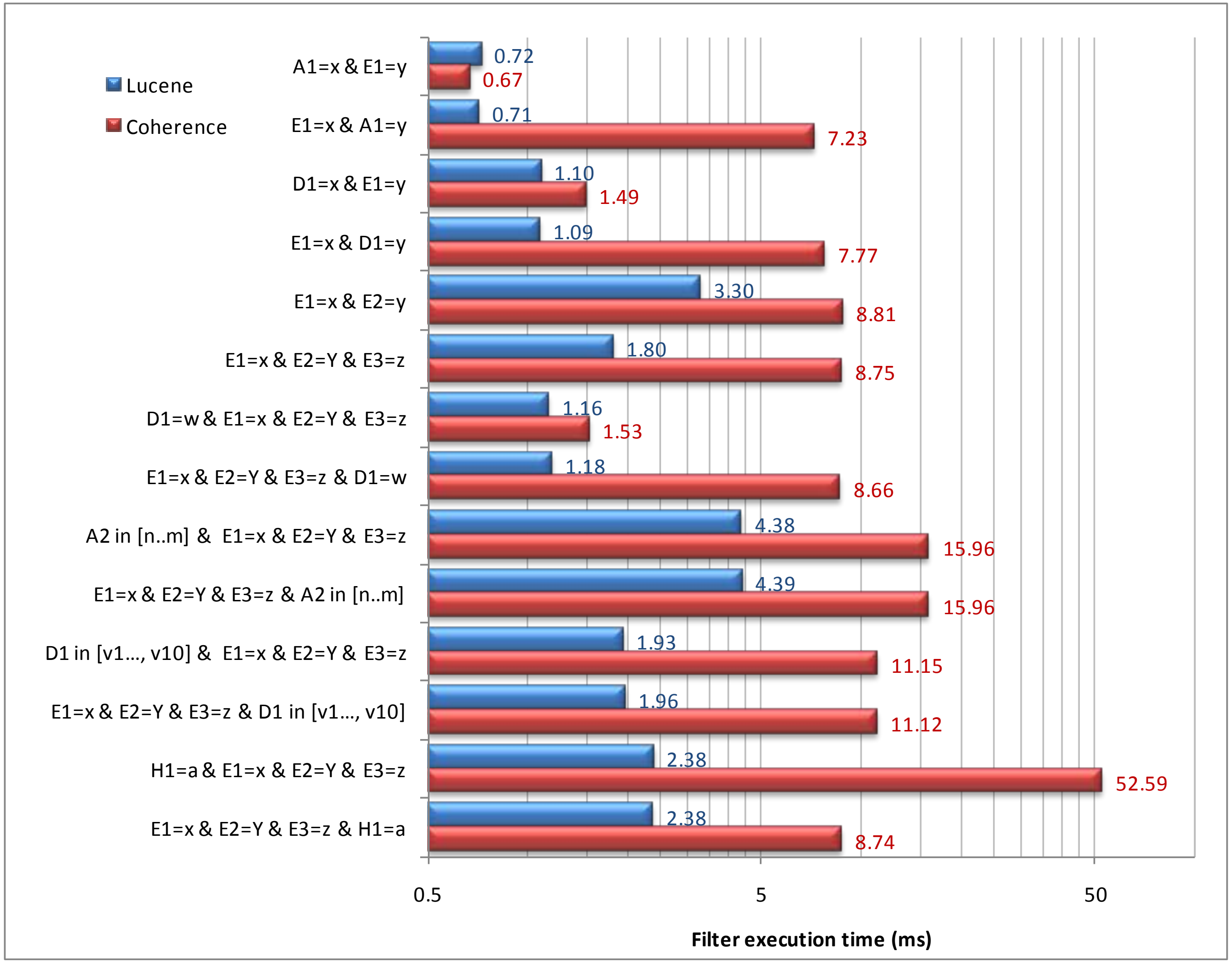
```
// now index is ready and we can search Coherence cache
// using Lucene queries
PhraseQuery pq = new PhraseQuery();
pq.add(new Term("content", "Coherence"));
pq.add(new Term("content", "search"));
// Lucene filter is converted to Coherence filter
// by search factory
cache.keySet(searchFactory.createFilter(pq));
```

Lucene example

You can even combine it with normal filters

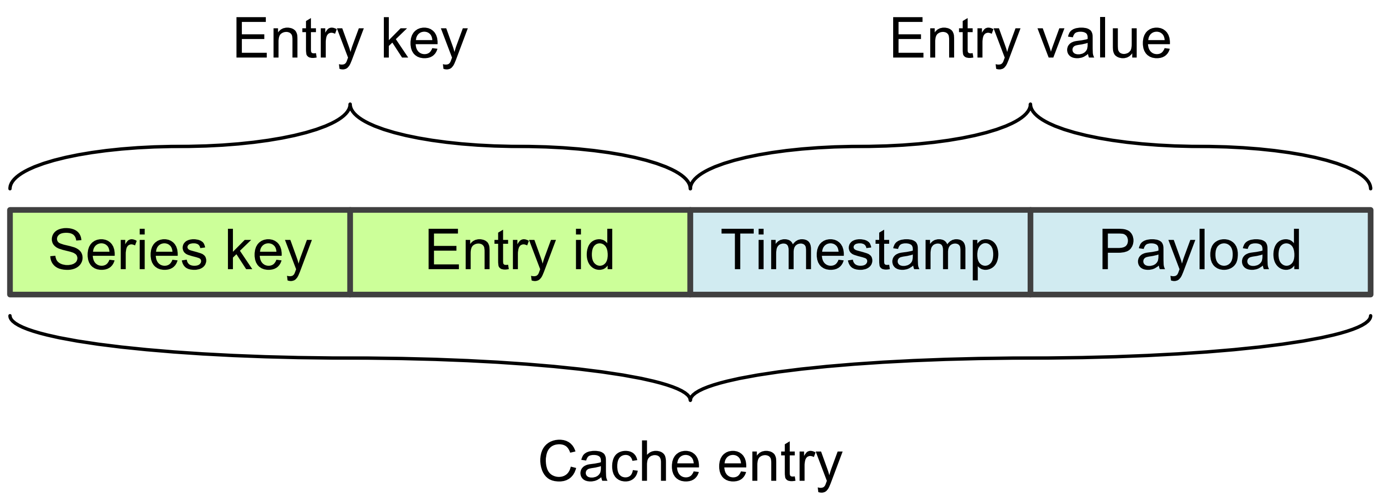
```
// You can also combine normal Coherence filters
// with Lucene queries
long startDate
  = System.currentTimeMillis() - 1000 * 60 * 60 * 24;
// last day
long endDate = System.currentTimeMillis();
BetweenFilter dateFilter
  = new BetweenFilter("getDateTime", startDate, endDate);
Filter pqFilter = searchFactory.createFilter(pq);
// Now we are selecting objects by Lucene query and apply
// standard Coherence filter over Lucene result set
cache.keySet(new AndFilter(pqFilter, dateFilter));
```

Lucene search performance



Time series index

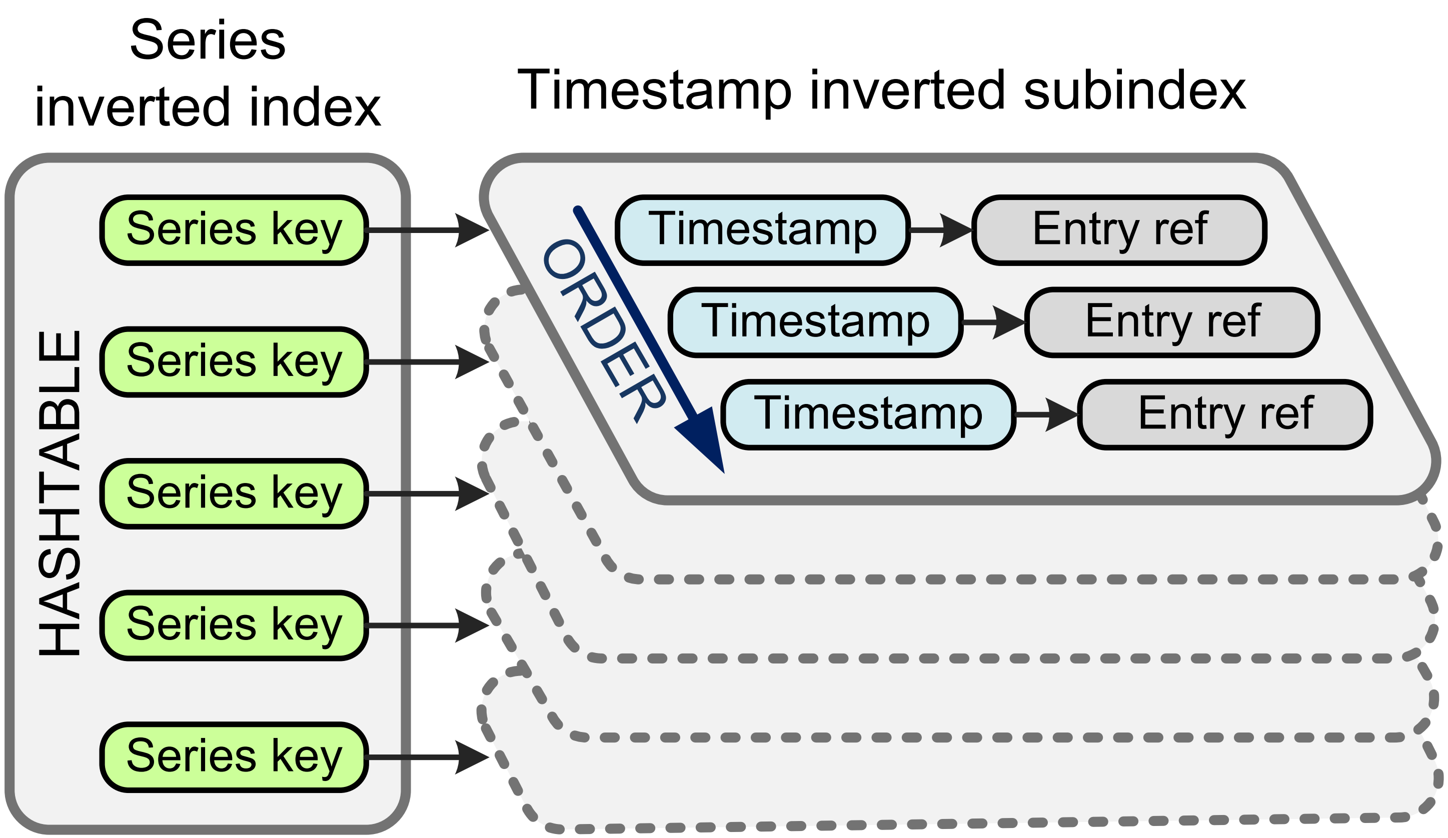
Special index for managing versioned data



Getting last version for series *k*

```
select * from versions where series=k and version =
   (select max(version) from versions where key=k)
```

Time series index



Thank you

http://aragozin.blogspot.com
- my articles
http://code.google.com/p/gridkit
- my open source code

Alexey Ragozin
alexey.ragozin@gmail.com