Apache Usergrid
Internals

Sungju Jin
sungju@apache.org

INDEX

INDEX

1. What is Apache Usergrid?

- **Designed for multi-tenancy & operational predictability.**
  Built on Java 7, Jersey & Apache Cassandra, with SDKs for iOS, Android, HTML5/JS, node.js, Ruby, Java, .NET, PHP — and so much more.

- https://github.com/usergrid/usergrid
- Creator, Ed Anuff (http://www.anuff.com)
- Since 2011.10.03 ~

apache usergrid_

The BaaS *not* made for Hipsters

Designed for multi-tenancy & operational predictability. Built on Java 7, Jersey & Apache Cassandra, with SDKs for iOS, Android, HTML5/JS, node.js, Ruby, Java, .NET, PHP – and so much more. Open source since 2011.

Currently undergoing incubation at the Apache Software Foundation

http://usergrid.incubator.apache.org/

Users

Sign up users, log in, reset passwords and more, in just one API call. You can put users in groups, assign roles or permissions, let users follow each other and access everything via OAuth 2.0, without writing a single line of server code.

Data

If you can express it in JSON, we can store it! Underneath everything is stored in a standard Cassandra instance, but we've added the ability to retrieve data via SQL, do graph queries, and even joins.

Files

Our asset storage can handle anything from text files to videos of several terrabytes, with automatic content-detection and full URL access control. In the back, everything goes Amazon S3 or other preferred cloud file store.

SDKs

Java-based

Trusted

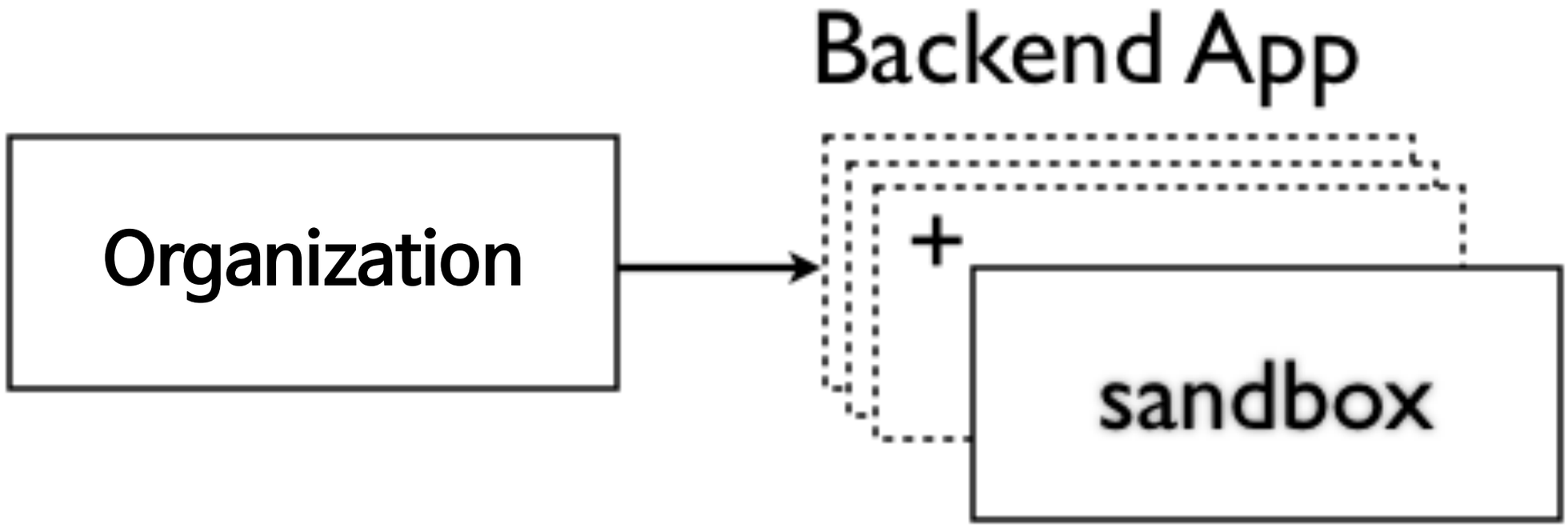
1. What is Apache Usergrid?

History

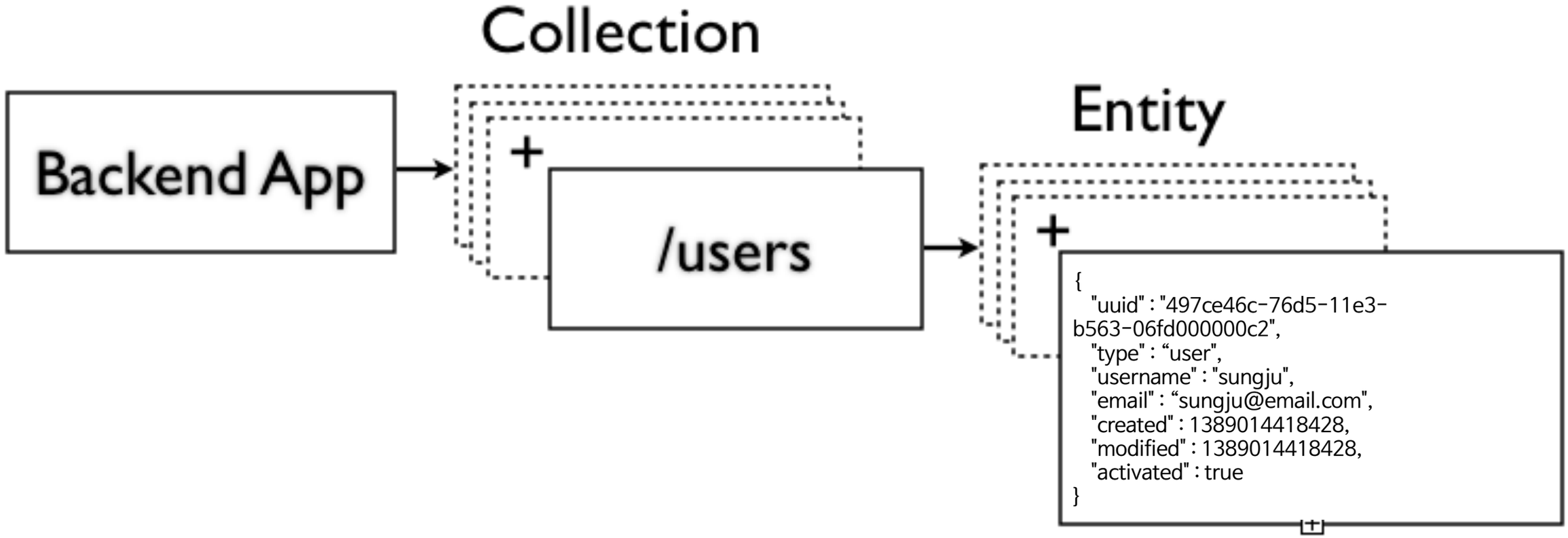
2011.10.03	Open sourced
2012.01.18 ~	Acquired by Apigee
2013.10 ~	Joined Apache incubator project

INDEX

2. Basic Concepts



2. Basic Concepts



2. Basic Concepts

Resource	POST create	GET read	PUT update	DELETE delete
/items	create a new item	list items	bulk update items	delete all items
/items/ipad	error	show ipad	update ipad	delete ipad

2. Basic Concepts

curl -X POST -i -H "Authorization: Bearer {auth_key}" -d '{"username":"bob","email":"bob@company.com"}' "**https://api.domain.io/my-org-id/my-app-id/users**"


```
{
   "action": "post",
   "application": "81c5c8b8-136a-11e2-8ed5-4061867ca222",
   "params": {},
   "path": "/users",
   "uri": "https://api.domain.io/my-org-id/my-app-id/users",
   "entities": [
      {
         "uuid": "37a71adc-136c-11e2-8ed5-4061867ca222",
         "type": "user",
         "created": 1349936628563,
         "modified": 1349936628563,
         "activated": true,
         "email": "bob@company.com",
         "picture": "https://www.gravatar.com/avatar/217195a2032ff3c42cf8711bd6334b0f",
         "username": "bob"
      }
   ],
   "timestamp": 1349936628483,
   "duration": 180,
   "organization": "my-org-id",
```

2. Basic Concepts

curl -X POST -i -H "Authorization: Bearer {auth_key}" -d '{"username":"bob","email":"bob@company.com"}' "**https://api.domain.io/my-org-id/my-app-id/users**"


```
{
   "action": "post",
   "application": "81c5c8b8-136a-11e2-8ed5-4061867ca222",
   "params": {},
   "path": "/users",
   "uri": "https://api.domain.io/my-org-id/my-app-id/users",
   "entities": [
      {
         "uuid": "37a71adc-136c-11e2-8ed5-4061867ca222",
         "type": "user",
         "created": 1349936628563,
         "modified": 1349936628563,
         "activated": true,
         "email": "bob@company.com",
         "picture": "https://www.gravatar.com/avatar/217195a2032ff3c42cf8711bd6334b0f",
         "username": "bob"
      }
   ],
   "timestamp": 1349936628483,
   "duration": 180,
   "organization": "my-org-id",
```

2. Basic Concepts

curl -X POST -i -H "Authorization: Bearer {auth_key}" -d '{"username":"bob","email":"bob@company.com"}' "**https://api.domain.io/my-org-id/my-app-id/users**"


```
{
   "action": "post",
   "application": "81c5c8b8-136a-11e2-8ed5-4061867ca222",
   "params": {},
   "path": "/users",
   "uri": "https://api.domain.io/my-org-id/my-app-id/users",
"entities": [
   {
      "uuid": "37a71adc-136c-11e2-8ed5-4061867ca222",
      "type": "user",
      "created": 1349936628563,
      "modified": 1349936628563,
      "activated": true,
      "email": "bob@company.com",
     "picture": "https://www.gravatar.com/avatar/217195a2032ff3c42cf8711bd6334b0f",
      "username": "bob"
   }
],
"timestamp": 1349936628483,
"duration": 180,
"organization": "my-org-id",
```


collection

application

organization

2. Basic Concepts

Relationship

POST	https://api.domain.io/devices	Create device
POST	https://api.domain.io/users	Create user
POST	https://api.domain.io/users/{user}/devices	Create device to user relationship
POST	https://api.domain.io/devices/{device}/users	Create user to device relationship

INDEX

3.Restful APIs

Categories of API

Application API	• Token • Predefined Collections • Users, Groups, Roles, Activities, Devices, Events, Folders, Assets • Collections
Management API	• Token • Admin Users • Organizations

3.Restful APIs

Collections (other than users, groups, and roles)

URI	Verb	Content Types	Action
/{org_id}/{app_id}/	GET	application/json	Retrieve all collections
/{org_id}/{app_id}/{collection}	POST	application/json	Create a new entity or collection
/{org_id}/{app_id}/{collection}/{uuid|name}	GET	application/json	Retrieve an entity
/{org_id}/{app_id}/{collection}/{uuid|name}	PUT	application/json	Update an entity
/{org_id}/{app_id}/{collection}/{uuid|name}	DELETE	application/json	Delete an entity
/{org_id}/{app_id}/{collection}?{query}	GET	application/json	Query a collection

3.Restful APIs

Collections (other than users, groups, and roles)

URI	Verb	Content Types	Action
/{org_id}/{app_id}/{collection}/{entity_id}/{relationship}?{query}	GET	application/json	Query an entity's collections or connections
/{org_id}/{app_id}/{collection}/{first_entity_id}/{relationship}/{second_entity_id} or /{org_id}/{app_id}/{collection}/{first_entity_id}/{relationship}/{second_entity_type}/{second_entity_id}	POST	application/json	Add an entity to a collection or create a connection
/{org_id}/{app_id}/{collection}/{first_entity_id}/{relationship}/{second_entity_id} or /{org_id}/{app_id}/{collection}/{first_entity_id}/{relationship}/{second_entity_type}/{second_entity_id}	DELETE	application/json	Remove an entity from a collection or delete a connection

3.Restful APIs

Access Token

URI	Verb	Content Types	Action
/management/token '{"grant_type":"client_credentials","client_id":"{client_id}","client_secret":"client_secret"}'	POST	application/json	Obtain an access token (access type = organization)
/management/token '{"grant_type":"password","username":"{username}":"password"="{password}"}'	POST	application/json	Obtain an access token (access type = admin user)
/management/{org_id}/{app_id}/token '{"grant_type":"client_credentials","client_id":"{client_id}","client_secret":"{client_secret}"}'	POST	application/json	Obtain an access token (access type = application)
/management/{org_id}/{app_id}/token '{"grant_type":"password","username":"{username}","password":"{password}"}'	POST	application/json	Obtain an access token (access type = application user)

3.Restful APIs

Organizations

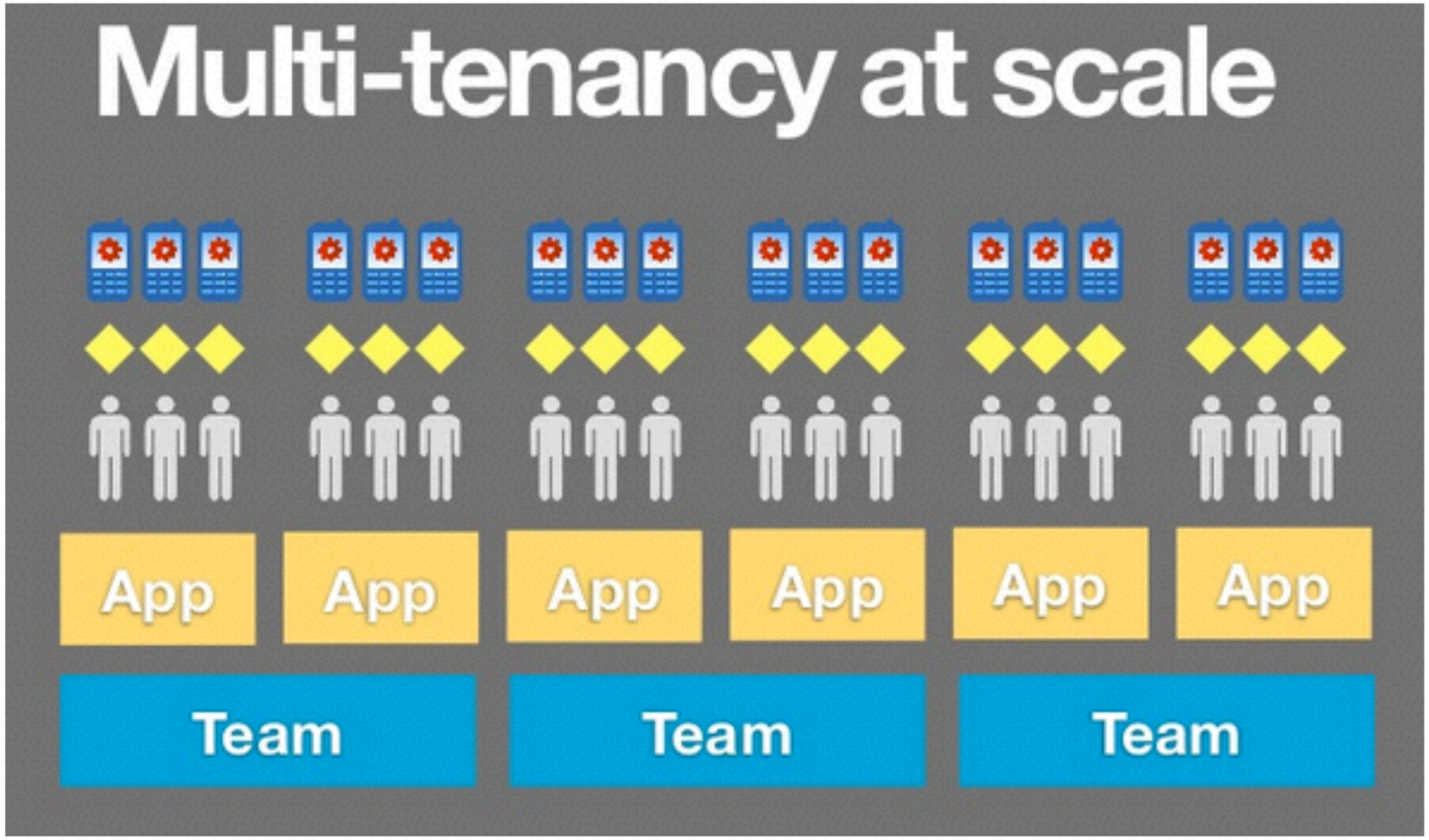
URI	Verb	Content Types	Action
/management/organizations|orgs	POST	application/json	Create an organization
/management/organizations|orgs/{org_name}|{uuid}	GET	application/json	Retrieve an organization
/management/organizations|orgs/{org_name}|{uuid}/activate?token={token}&confirm={confirm_email}	GET	application/json	Activate an organization
/management/organizations|orgs/{org_name}|{uuid}/reactivate	GET	application/json	Reactivate an organization
/management/organizations|orgs/{org_name}|{uuid}/credentials	POST	application/json	Generate organization client credentials

INDEX

4.Architecture

Design Goal

- Designed for multi-tenancy
- Scalability



4.Architecture

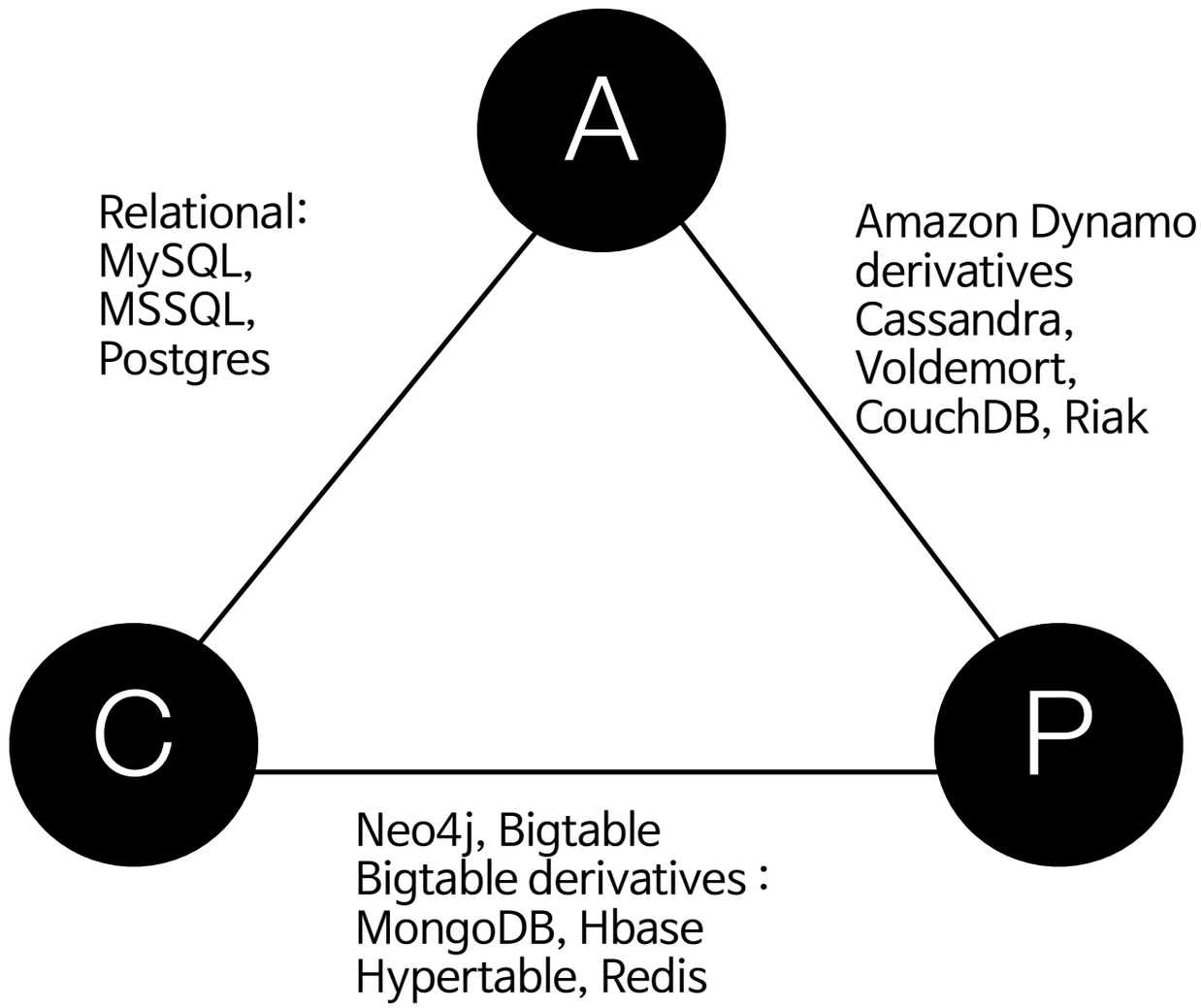
Design background

- **Brewer's theorem**

  Consistency

  Availability

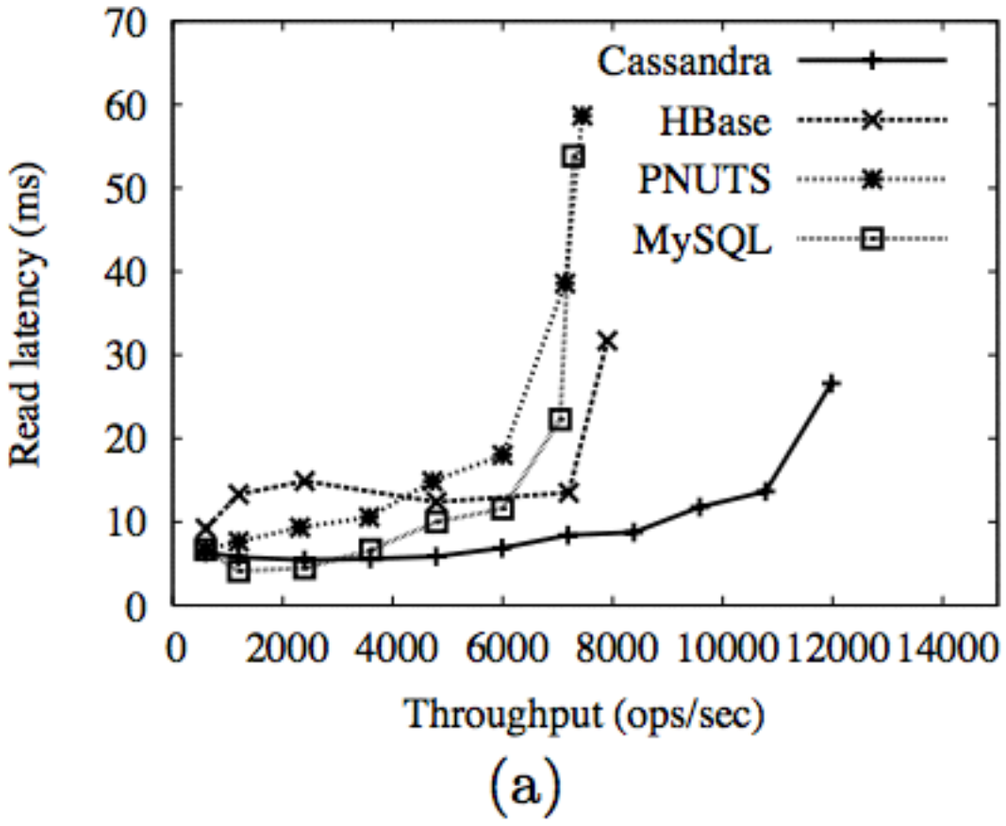
  Partition tolerance



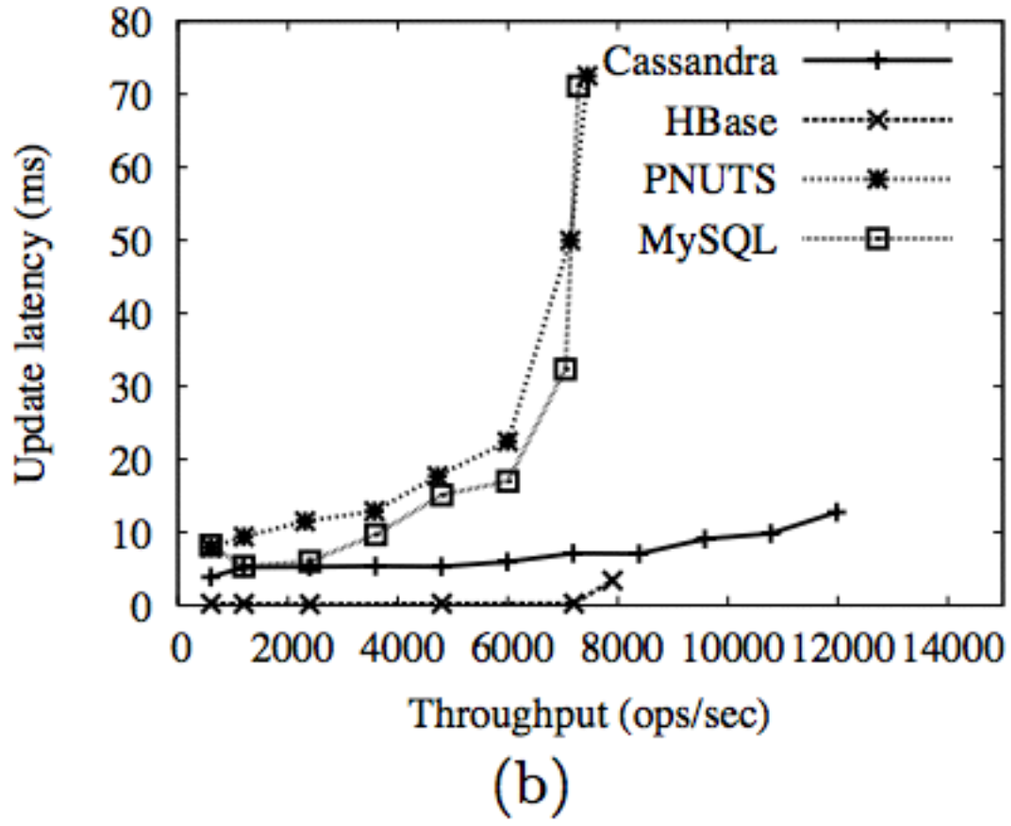
4.Architecture

Design background

- Benchmark



(a)



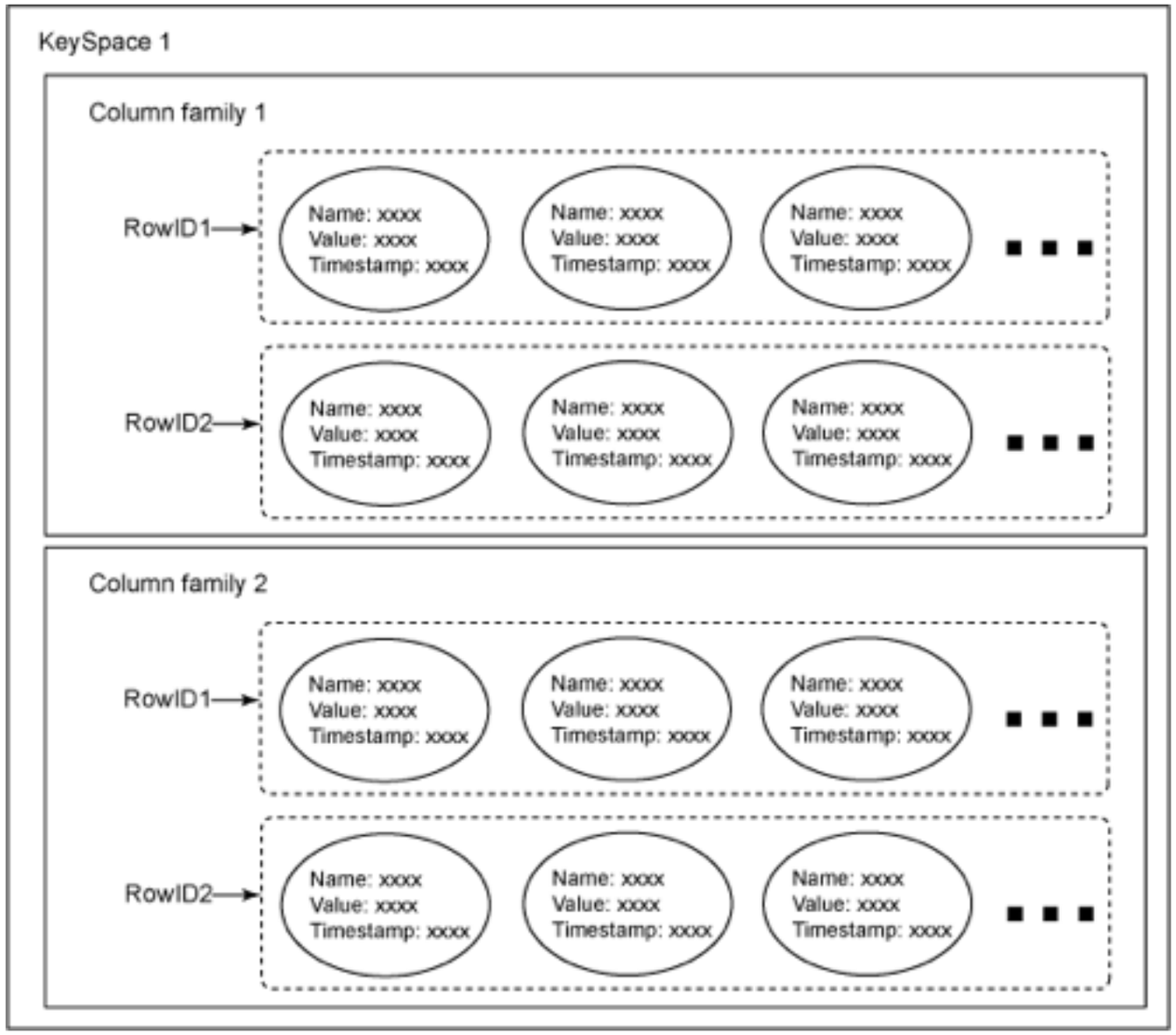
(b)

Workload A—update heavy: (a) read operations, (b) update operations. Throughput in this (and all figures) represents total operations per second, including reads and writes.

References : Benchmarking Cloud Serving Systems with YCSB

4.Architecture

Design background



http://www.ibm.com/developerworks/library/os-apache-cassandra/

4.Architecture

Design background

User	Cluster size	Node count	Usage	Now
~~Facebook~~	~~>200~~	~~?~~	~~Inbox search~~	Abandoned, Moved to HBase
Cisco WebEx	?	?	User feed, activity	
Netflix	200 TB	750 (50 cluster)		
Apigee	?	?	Data store	
Formspring	? (26 million account with 10m responses per day)	?	Social-graph data	
Urban airship, Rackspace, Open X, Twitter (preparing move to), ebay				

References :
http://planetcassandra.org/Company/ViewCompany
http://en.wikipedia.org/wiki/Apache_Cassandra

4.Architecture

Design background

	Separate Database	Shared Database
Shared Schema	- Scalability + Isolation + Simple	+ Scalability - No Isolation - Complicated
Separate Schema	- Scalability + Isolation + Not Complicated	- Scalability - No Isolation - Complicated

4.Architecture

Design background

	Separate Database	Shared Database
Shared Schema	Expensive	Interesting
Separate Schema	Very Expensive	Unwieldy

4. Architecture

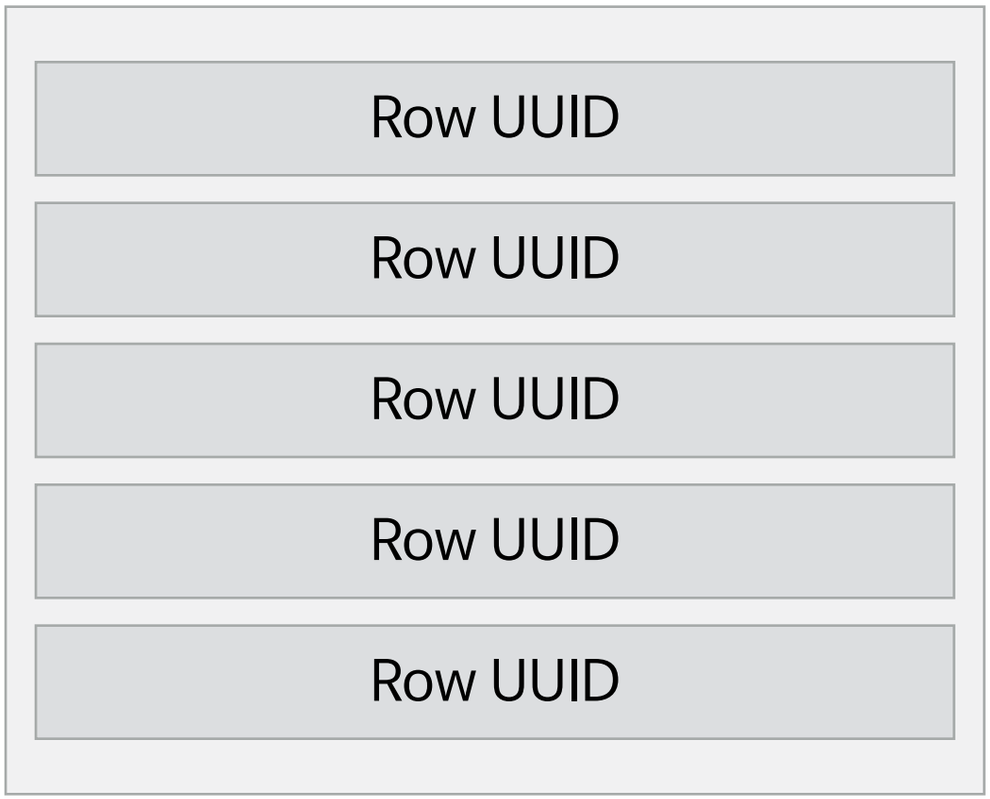
Design background

	Separate Database	Shared Database
Shared Schema	− Scalability + Isolation + Simple	+ Scalability − No Isolation − Complicated
Separate Schema	− Scalability + Isolation + Not Complicated	− Scalability − No Isolation − Complicated

4.Architecture

Design background

Conventional Row Keys In Single Keyspace



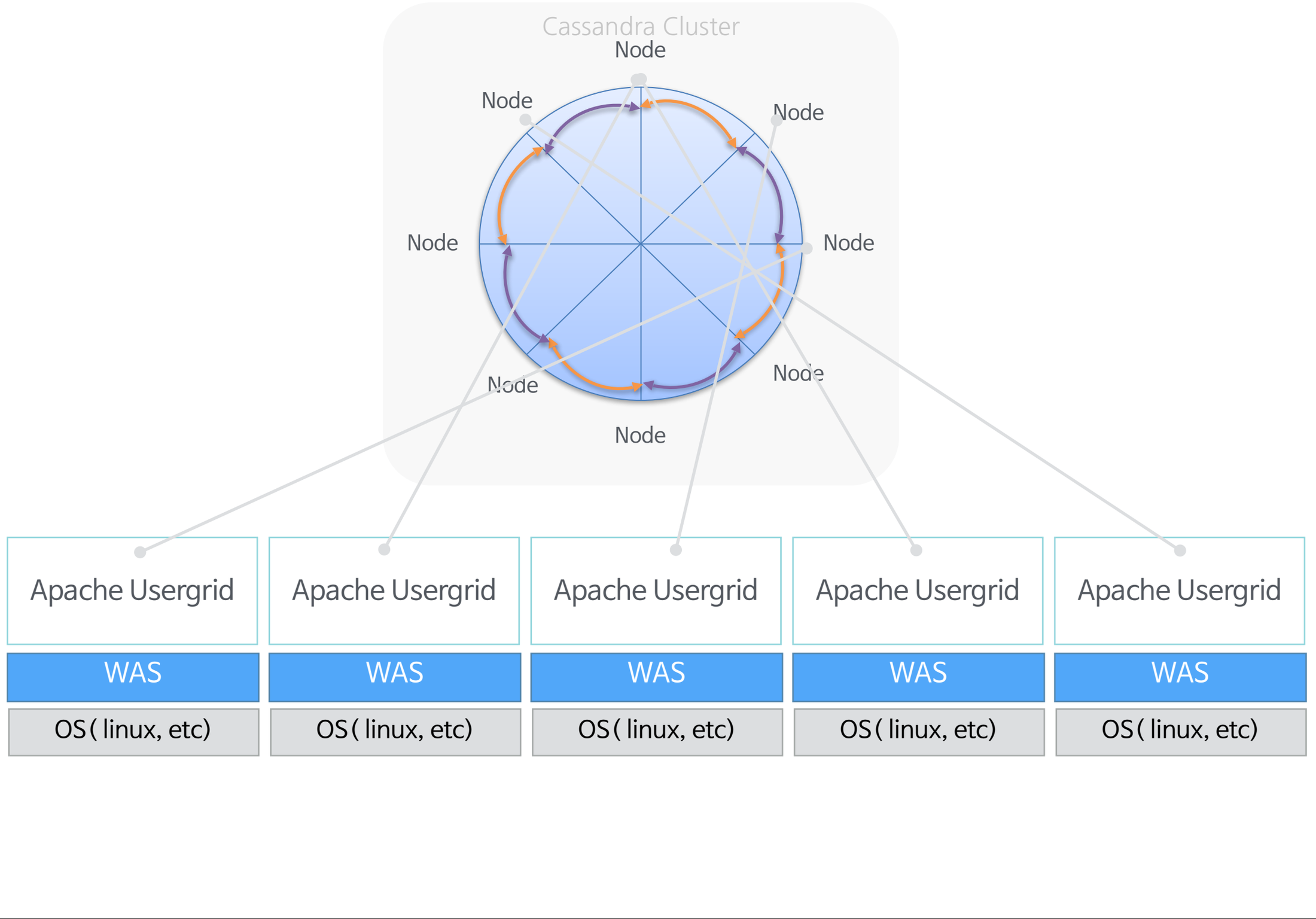
4.Architecture

Design background

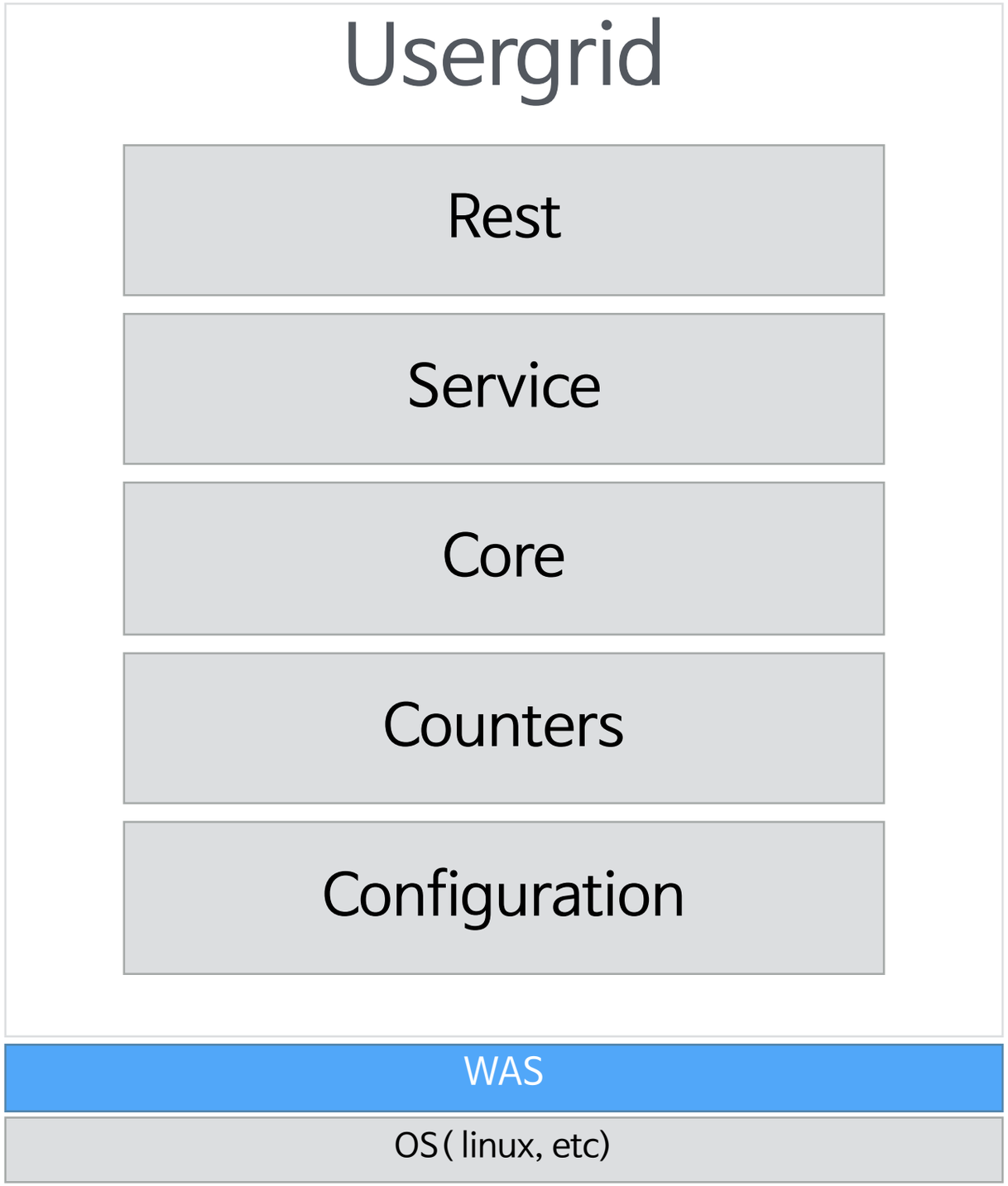
Multi-tenant Row Keys In Shared Keyspace

Tenant ID	Row UUID
Tenant ID	Row UUID
Tenant ID	Row UUID
Tenant ID	Row UUID
Tenant ID	Row UUID

4.Architecture



4.Architecture

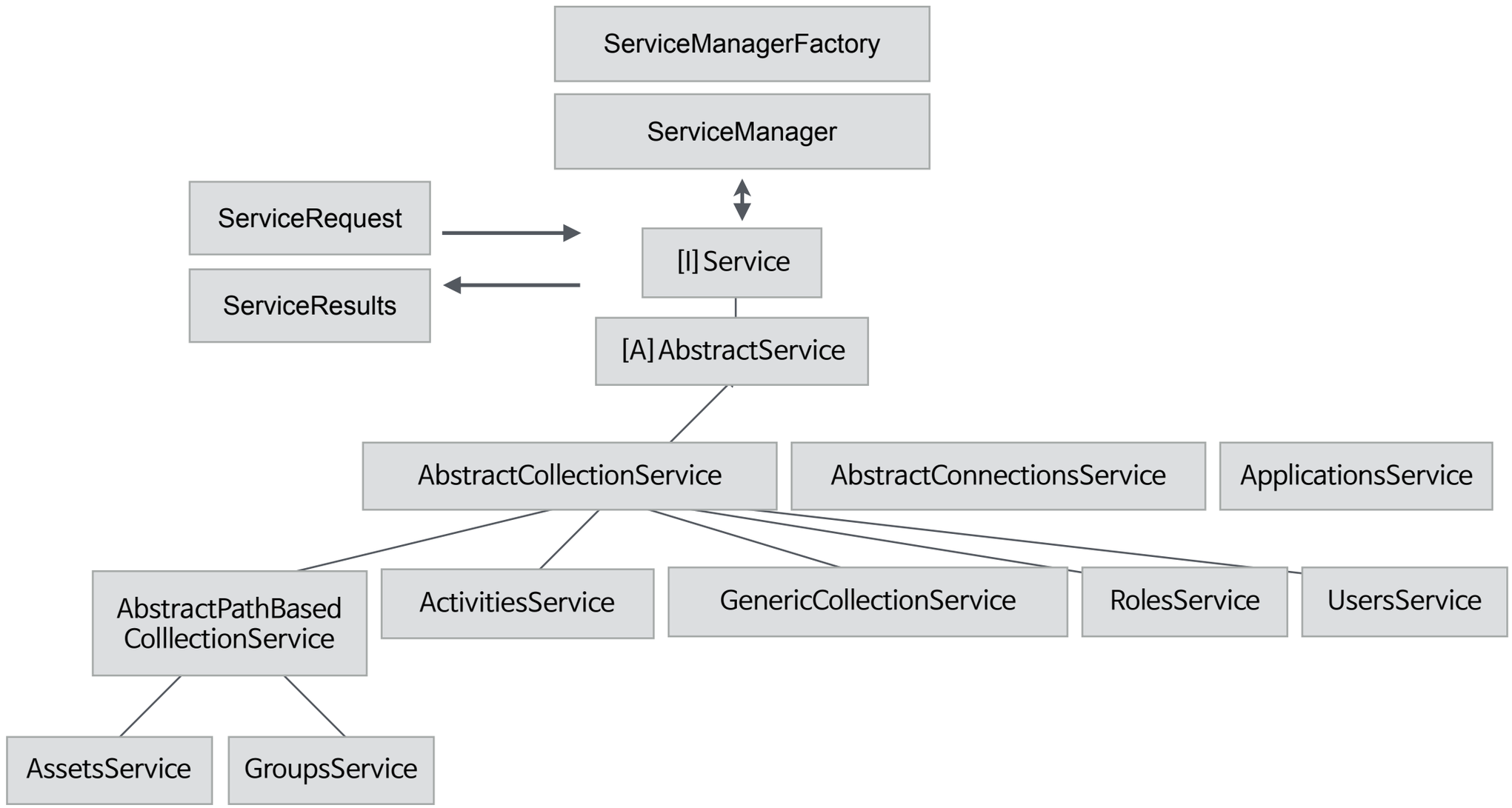


4. Architecture

Usergrid

- Rest
- Service
- Core
- Counters
- Configuration

Service



4. Architecture

Usergrid

- Rest
- Service
- Core
- Counters
- Configuration

Core

EntityManagerFactory

EntityManagerFactory.getEntityManager(UUID applicationId)

EntityManager

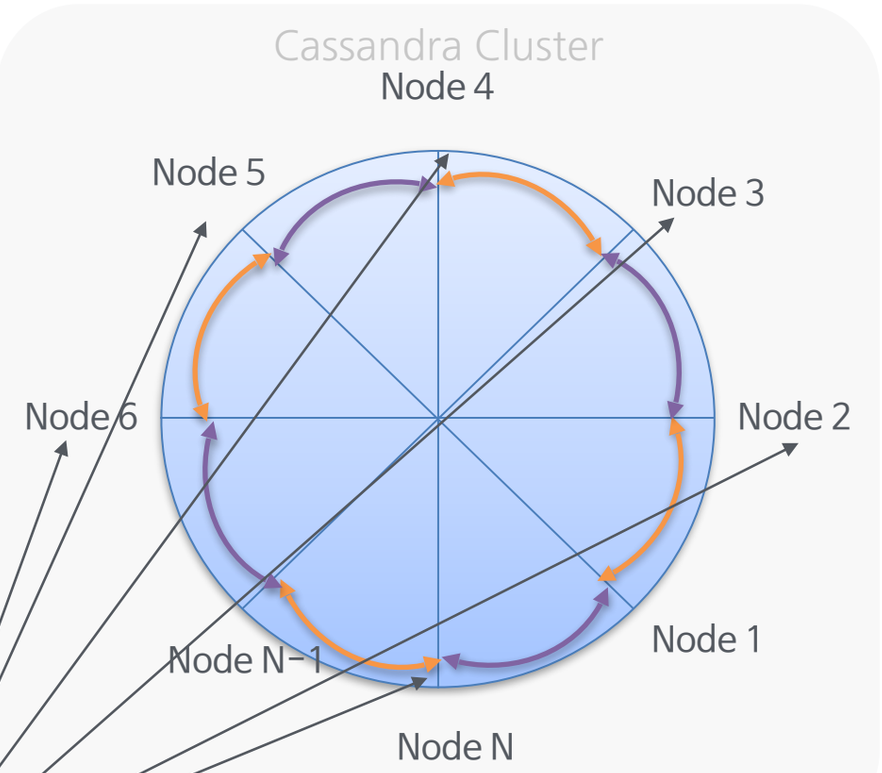
EntityManager.getRelationManager(EntityRef entityRef)

- EntityFactory
- QueueManagerFactory
- GeoIndexManager
- Schema
- IndexBucketScanner
- IndexBucketLocator
- LockManager
- QueryProcessor

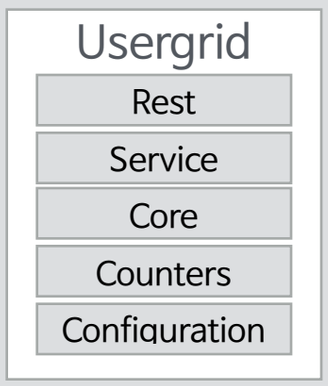
RelationManager

CassandraService

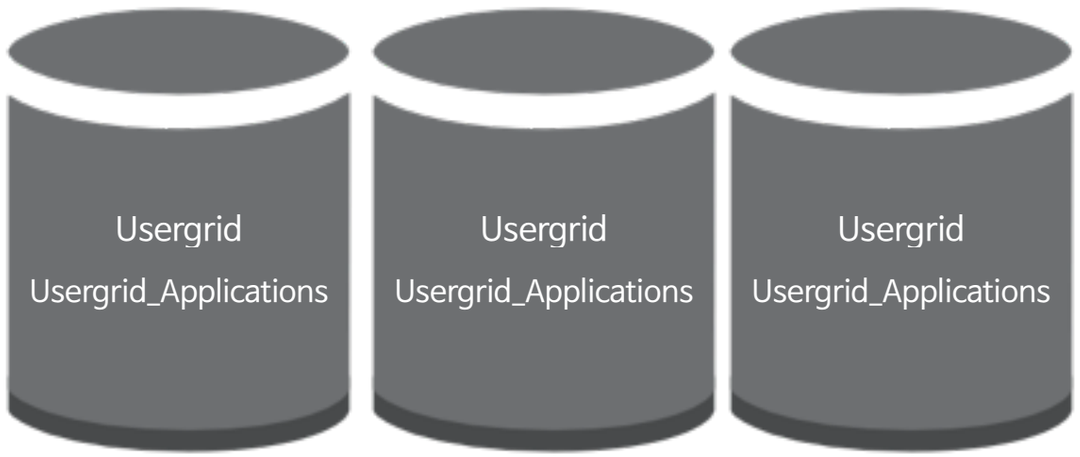
Hector



4.Architecture



Schema on Cassandra (Physical Layer)



KeySpaces

Usergrid

- Applications
- Properties
- Tokens

Usergrid_Applications

- Application_Aggregate_Counters
- Application_Roles
- Entity_Aliases
- Entity_Composite_Dictionaries
- Entity_Connections
- Entity_Counters
- Entity_Dictionaries
- Entity_Id_Sets
- Entity_Index
- Entity_Index_Entries
- Entity_Properties
- Consumer_Queue_Messages_Properties
- Entity_Metadata
- MQ_Consumers
- MQ_Counters
- MQ_Property_Index
- MQ_Property_Index_Entries
- Queue_Dictionaries
- Queue_Inbox
- Queue_Properties
- Queue_Subscribers
- Queue_Subscriptions

INDEX

5. An Overview of Data Processing

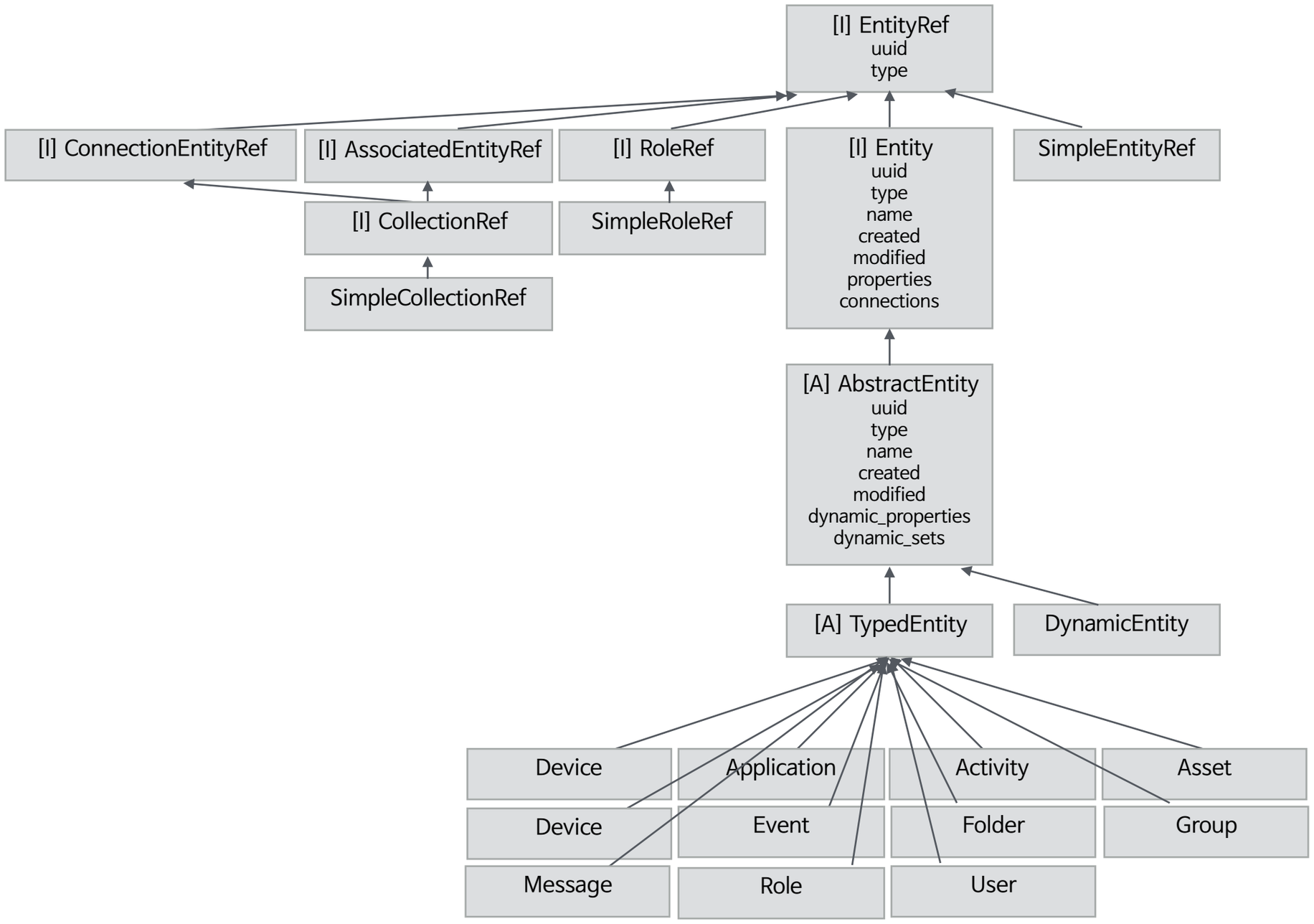
- **Metadata**
  - Information of Organizations, Applications
  - Information of Collections
  - Information of Entity
- **Real data**
  - Name, Value

5. An Overview of Data Processing

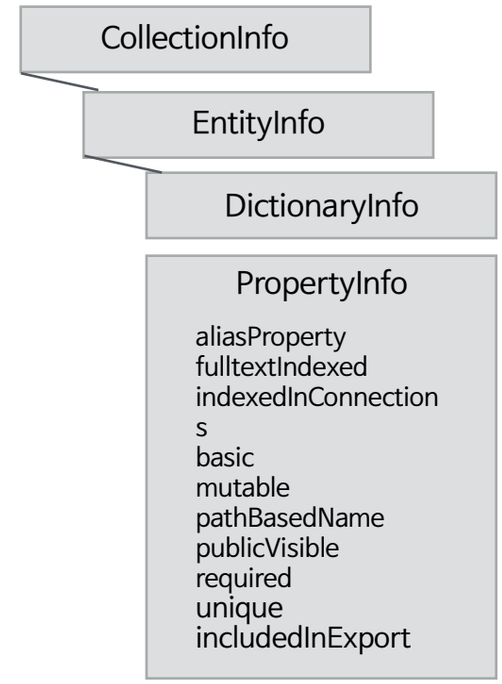
- **Collections**
  - Organization 〉Application 〉Collections
  - Schema
    - Unique
    - Indexes (option, fulltextindexed)
      - Alias : Another lookup property
  - Information of added to the entity in collection
- **Entity**
  - Information of property
  - Property name
  - Property value

5. An Overview of Data Processing

Entity



Entity Schema



5. An Overview of Data Processing

Collections

- Predefined Collections
  - Users, Groups, Roles, Activities, Devices, Events, Folders, Assets

- Dynamic Collections
  - uuid (unique, required)
  - name (alias)
  - created (required)
  - modified (required)
  - … (indexed, fulltextIndexed)

5. An Overview of Data Processing

Collections

Predefined Collections

- **Application**
  - name (required, indexed, alias, unique)
  - title
  - accesstokenttl
  - description
  - collections
  - counters
  - activated
  - disabled
  - modified (required)
  - … (indexed, fulltextIndexed)
  - allowOpenRegistration, registrationRequiresEmailConfirmation, registrationRequiresAdminApproval, notifyAdminOfNewUsers
  - activities, assets, events, folders, groups, users, devices

5. An Overview of Data Processing

Collections

Predefined Collections

- **Users**
  - username (required, indexed, fulltextIndexed, alias, unique)
  - email (indexed, unique)
  - name (indexed, fulltextIndexed)
  - activated (indexed)
  - deactivated (indexed)
  - confirmed (indexed)
  - disabled (indexed)
  - picture (indexed)
  - created (required)
  - modified (required)
  - … (indexed, fulltextIndexed)
  - connections, permissions, credentials, groups (linkedCollection), devices (linkedCollection), activities (linkedCollection), feed (linkedCollection), roles (linkedCollection)

5. An Overview of Data Processing

Collections

Predefined Collections

- **Devices**
  - name (required, indexed, alias, unique)
  - created (required)
  - modified (required)
  - … (indexed, fulltextIndexed)
- users (linkedCollection)

5. An Overview of Data Processing

Collections

Predefined Collections

- Groups
  - path (required, indexed, alias, unique)
  - created (required)
  - modified (required)
  - … (indexed, fulltextIndexed)
- users(linkedCollection), connections, credentials, permissions, activities(linkedCollection), feed(linkedCollection), roles(linkedCollection)

5. An Overview of Data Processing

Collections

Predefined Collections

- Role
  - name (required, indexed, alias, unique)
  - roleName (indexed)
  - title (indexed)
  - inactivity (indexed)
  - created (required)
  - modified (required)
  - … (indexed, fulltextIndexed)
  - permissions
  - users (linkedCollection)
  - groups (linkedCollection)

INDEX

INDEX

6. How it works: CRUD

Create an entity

```
Map<String, Object> properties = new
LinkedHashMap<String, Object>();
properties.put("username", "sungju");

Entity user = entityManager.create("user", properties);
```

6. How it works: CRUD

Core methods to create an entity

EntityManagerImpl.batchCreate(Mutator⟨ByteBuffer⟩ m, String entityType, Class⟨A⟩ entityClass, Map⟨String, Object⟩ properties, UUID importId, UUID timestampUuid)

EntityManagerImpl.batchSetProperty(Mutator⟨ByteBuffer⟩ batch, Entity entity, String propertyName, Object propertyValue, boolean force, boolean noRead, UUID timestampUuid)

RelationManagerImpl.batchUpdatePropertyIndexes(Mutator⟨ByteBuffer⟩ batch, String propertyName, Object propertyValue, boolean entitySchemaHasProperty, boolean noRead, UUID timestampUuid)

RelationManagerImpl.batchStartIndexUpdate(Mutator⟨ByteBuffer⟩ batch, Entity entity, String entryName, Object entryValue, UUID timestampUuid, boolean schemaHasProperty, boolean isMultiValue, boolean removeListEntry, boolean fulltextIndexed, boolean skipRead)

6. How it works: CRUD

Process of creating an entity

1. Precondition

    • All commands used batch_mutate in the Cassandra thrift API, so you should care to set thrift buffer size.

2. Pre-processing

- Get an alias property from org.usergrid.persistence.Schema (such as ‘name’)
- Generate a collection type (Inflector.pluralize)
- Generate Bucket ID (SimpleBucketIdLocator, 20 buckets)
- Generate collection key : {applicationId}:collections: {collection_name}:{bucketID}
- Set default properties (uuid, type, created, modified)
- Checking required properties(Loop through all the properties)

6. How it works: CRUD

Process of creating an entity (con't)

3. Handling metadata
    1. Entity UUID
    2. Property names (dictionary)
4. Loop through all the properties
    1. Checking alias property
    2. Checking unique property
    3. Save property

6. How it works: CRUD

Process of creating an entity (con't)

4.4. Save property (EntityManagerImpl.batchSetProperty)

1) Get Entity Info from org.usergrid.persistence.Schema

2) Validate value (type casting)

3) If property value is null, do delete

4) If property is mutation, skip the process

5) Check required property

6) Check unique property

7) Check alias property

8) Indexing property if it needed

9) Saving real data

6. How it works: CRUD

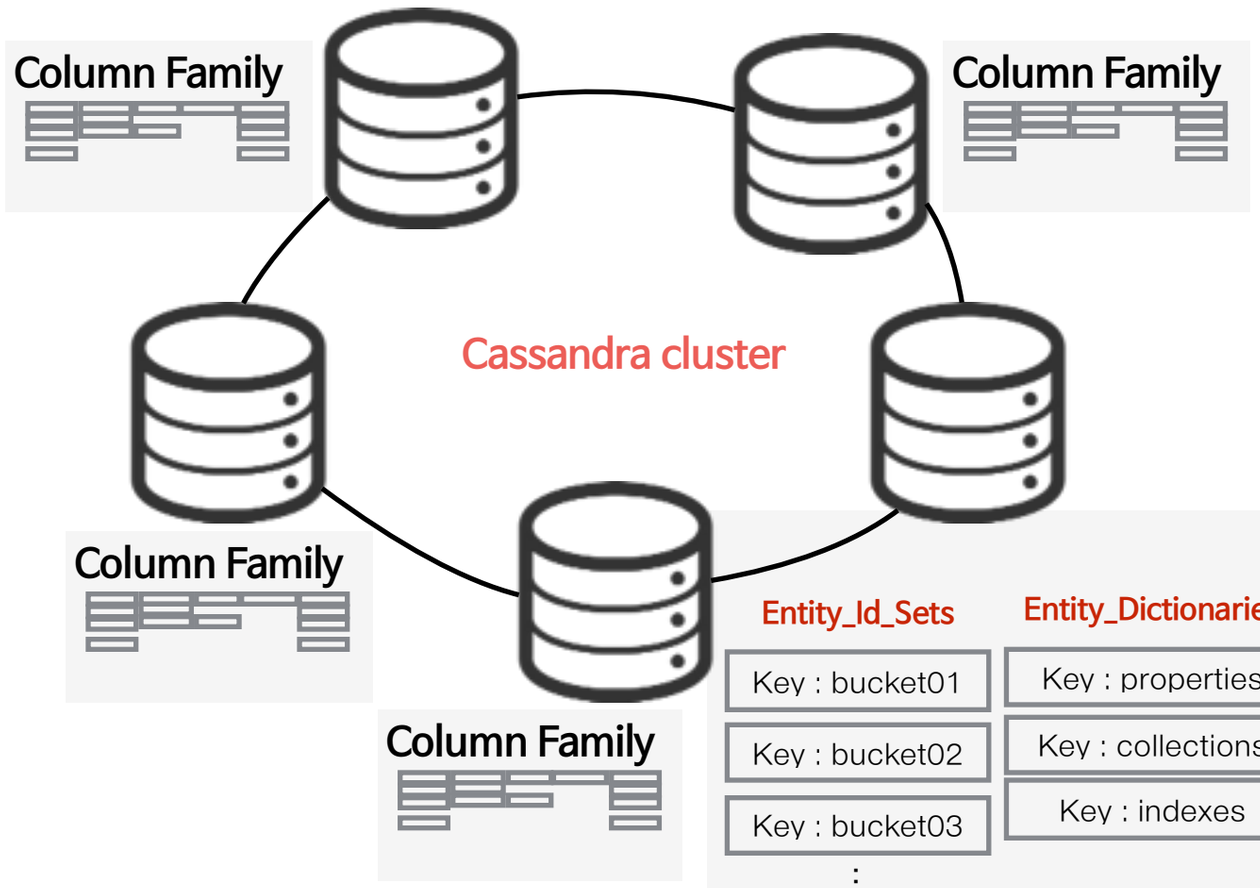
Process of creating an entity (con't)

4.4.9. Indexing

1) Background
    1) http://anuff.com/2010/07/secondary-indexes-in-cassandra/
    2) http://anuff.com/2011/07/cassandra-summit-sf-2011-presentation/
    3) http://anuff.com/2011/02/indexing-in-cassandra/
2) Index Type - COLLECTION, CONNECTION, GEO
3) Get tokens based on lucene 3.0 (token separator, space)
    1) fulltextIndex = false, (Key="name", Value="Sungju Jin")
    2) fulltextIndex = true, (Key="name", Value="Sungju Jin"), (Key="name**.keyword**", Values="sungju, jin")
4) Delete all matching index in Entity_Index from the entries retrieved from Entity_Index_Entries in the previous step
5) Delete all columns in Entity_Index_Entries that were previously retrieved
6) Insert new entry in Entity_Index
7) Insert new entry in Entity_Index_Entries

6. How it works: CRUD

Usergrid on Cassandra (Physical Layer)



Column Family

Entity_Id_Sets	Entity_Dictionaries	Entity_Unique	Entity_Properties	Entity_Index	Entity_Index_Entries
Key : bucket01	Key : properties	Key	Key(uuid)	Key : bucket01	Key
Key : bucket02	Key : collections	Entity_Aliases		Key : bucket02	
Key : bucket03	Key : indexes	Key		Key : bucket03	
⋮				⋮	
Key : bucket20				Key : bucket20	

Entity_Composite_Dictionaries

Key : container_entities

6. How it works: CRUD

- **Entity_Dictionaries**
  - Key : uuid + “properties”
  - Column
    - Name : propertyName, Value : Null
  - Key: uuid + "collections"
  - Column
    - Name : collection_name, Value : Null
  - Key: uuid + "indexes"
  - Column
    - Name : collection_name, Value : Null
- **Entity_Id_Sets**
  - Key : applicationUuid:collections :collectionName:bucketId
  - Column
    - Name : uuid, Value : Null
- **Entity_Aliases**
  - Key : appId + aliasType + alias
  - Column
    - Name : "entityId", Value : entityId
    - Name : "entityType", Value : entityType
    - Name : "aliasType", Value : aliasType
    - Name : "alias", Value : alias
- **Entity_Unique**
  - Key : appId:collectionName:propertyName:value
  - Column
    - Name : entityId, Value : Null
- **Entity_Composite_Dictionaries**
  - Key : itemId + "container_entities"
  - Column
    - Name : List (application + collection_name + applicationId), Value : Null
- **Entity_Index**
  - Key : uuid, collectionName, path, bucketId
  - Column :
    - Name : Index type code, value, uuid, timestamp uuid, Value : Null
- **Entity_Index_Entries**
  - Key : entityId
  - Column :
    - Name : entryName, code, val, timestampUuid, name, Value : Null
- **Entity_Properties**
  - Key : uuid
  - Column
    - Name : propertyName
    - Value : entityType, propertyName, propertyValue

6. How it works: CRUD

Create entity

```
Map<String, Object> properties = new
LinkedHashMap<String, Object>();
properties.put("username", "sungju");

Entity user = entityManager.create("user", properties);
```

6. How it works: CRUD

Keys

- appId : 81185cd1-5d6e-11e3-b1d1-06a6fa0000b9
- entityId : 0177c265-a596-11e3-b4cd-406c8f07e929
- bucketId : 153127065114422308558518573442956951 47
- collection_key : 81185cd1-5d6e-11e3-b1d1-06a6fa0000b9:collections:users:153127065114422308558518573442956

Entity_Id_Sets

Key : 81185cd1-5d6e-11e3-b1d1-06a6fa0000b9:collections:users:153127065114422308558518573442956951 47

Column Name/Value : 0177c265-a596-11e3-b4cd-406c8f07e929, Null

Entity_Aliases

Key : CassandraPersistenceUtils.aliasID("81185cd1-5d6e-11e3-b1d1-06a6fa0000b9", "user", "sungju")

Column Name/Value : entityId, 0177c265-a596-11e3-b4cd-406c8f07e929

Column Name/Value : entityType, user

Column Name/Value : aliasType, user

Column Name/Value : alias, sungju

Entity_Unique

Key : 81185cd1-5d6e-11e3-b1d1-06a6fa0000b9:users:username:sungju

Column Name/Value : 0177c265-a596-11e3-b4cd-406c8f07e929, Null

6. How it works: CRUD

Keys

- appId : 81185cd1-5d6e-11e3-b1d1-06a6fa0000b9
- entityId : 0177c265-a596-11e3-b4cd-406c8f07e929
- bucketId : 153127065114422308558518573442956951 47
- collection_key : 81185cd1-5d6e-11e3-b1d1-06a6fa0000b9:collections:users:153127065114422308558518573442956

Entity_Id_Sets

Key : 81185cd1-5d6e-11e3-b1d1-06a6fa0000b9:collections:users:153127065114422308558518573442956951 47

Column Name/Value : 0177c265-a596-11e3-b4cd-406c8f07e929, Null

Entity_Aliases

Key : CassandraPersistenceUtils.aliasID("81185cd1-5d6e-11e3-b1d1-06a6fa0000b9", "user", "sungju")

Column Name/Value : entityId, 0177c265-a596-11e3-b4cd-406c8f07e929

Column Name/Value : entityType, user

Column Name/Value : aliasType, user

Column Name/Value : alias, sungju

Entity_Unique

Key : 81185cd1-5d6e-11e3-b1d1-06a6fa0000b9:users:username:sungju

Column Name/Value : 0177c265-a596-11e3-b4cd-406c8f07e929, Null

6. How it works: CRUD

Keys

- **appId : 81185cd1-5d6e-11e3-b1d1-06a6fa0000b9**
- **entityId : 0177c265-a596-11e3-b4cd-406c8f07e929**
- **bucketId : 153127065114422308558518573442956951 47**
- **collection_key : 81185cd1-5d6e-11e3-b1d1-06a6fa0000b9:collections:users:153127065114422308558518573442956**

Entity_Dictionaries

Key : 81185cd1-5d6e-11e3-b1d1-06a6fa0000b9:collections

Column Name/Value : 0177c265-a596-11e3-b4cd-406c8f07e929, Null

Entity_Composite_Dictionaries

Key : 0177c265-a596-11e3-b4cd-406c8f07e929:container_entities

Column Name/Value : List〈application, users, 81185cd1-5d6e-11e3-b1d1-06a6fa0000b9〉, Null

Entity_Properties

Key : 0177c265-a596-11e3-b4cd-406c8f07e929

Column Name/Value : username | user, username, sungju

6. How it works: CRUD

Keys

- appId : 81185cd1-5d6e-11e3-b1d1-06a6fa0000b9
- entityId : 0177c265-a596-11e3-b4cd-406c8f07e929
- bucketId : 153127065114422308558518573344295695147
- collection_key : 81185cd1-5d6e-11e3-b1d1-06a6fa0000b9:collections:users:153127065114422308558518573344295695

Entity_Index

Key : 81185cd1-5d6e-11e3-b1d1-06a6fa0000b9:users:username:059549414211164231106090556300559437001

Column Name/Value : 1, "sungju", 0177c265-a596-11e3-b4cd-406c8f07e929, f48df154-a595-11e3-b4cd-406c8f07e929

Entity_Index_Entries

Key : 0177c265-a596-11e3-b4cd-406c8f07e929

Column Name/Value : List("username", 1, "sungju", timestampUuid, name)

IndexCode

Bytes - 0

String - 1

UUID -2

INT - 3

INDEX

6. How it works: CRUD

Update an entity

```
UUID uuid = UUID.fromString(“0177c265-a596-11e3-b4cd-406c8f07e929”);
Entity entity = EntityFactory.newEntity(uuid,”users”);
entity.setProperty("username", "sungju1");
```

```
Entity user = entityManager.update(entity);
```

INDEX

6. How it works: CRUD

Delete an entity

```
UUID uuid = UUID.fromString("0177c265-a596-11e3-b4cd-406c8f07e929");
Entity entity = EntityFactory.newEntity(uuid,"users");

Entity user = entityManager.delete(entity);
```

INDEX

6. How it works: CRUD

Read an entity

1. Get an entity by UUID
2. Get an entity by Alias
3. Retrieve entities
4. Retrieve entities by Query

6. How it works: CRUD

Read an entity 〉Get an entity by UUID

1) Get columns from **Entity_Properties**

6. How it works: CRUD

Read an entity 〉Get an entity by Alias

1) Get columns from **Entity_Aliases**
2) Get columns from **Entity_Properties**

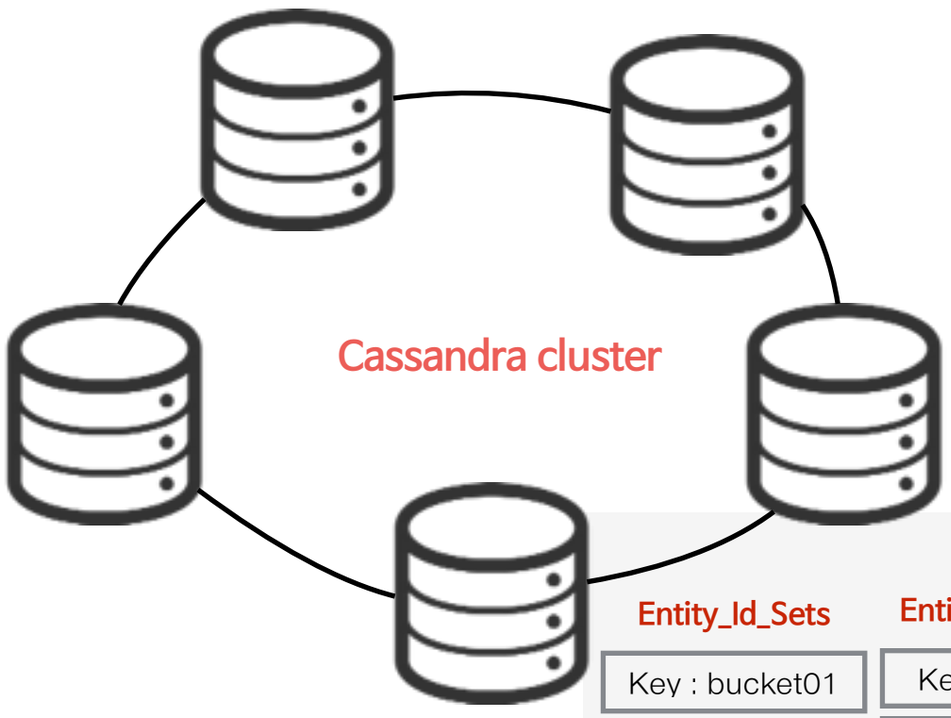
6. How it works: CRUD

Read an entity 〉Retrieve entities

1) Get columns from **Entity_Id_Sets**
2) Get columns from **Entity_Properties**

6. How it works: CRUD

Read an entity 〉Retrieve entities by Query



Column Family

Entity_Id_Sets

Key : bucket01
Key : bucket02
Key : bucket03
⋮
Key : bucket20

Entity_Dictionaries

Key : properties
Key : collections
Key : indexes

Entity_Unique

Key

Entity_Aliases

Key

Entity_Properties

Key(uuid)

Entity_Index

Key : bucket01
Key : bucket02
Key : bucket03
⋮
Key : bucket20

Entity_Index_Entries

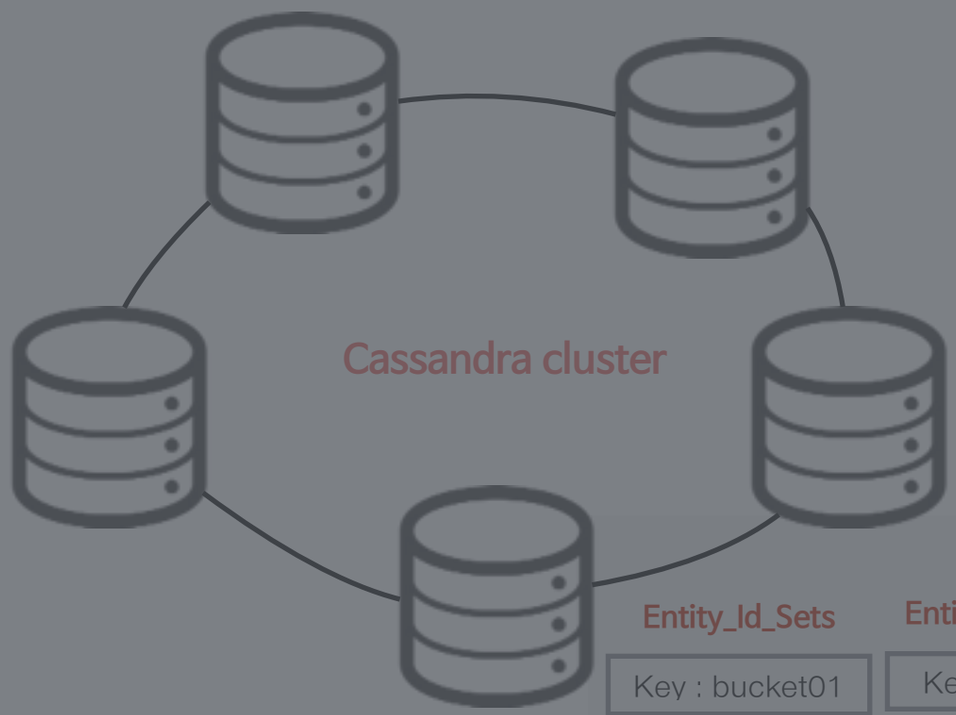
Key

Entity_Composite_Dictionaries

Key : container_entities

6. How it works: CRUD

Read an entity 〉Retrieve entities by Query



Column Family

Entity_Id_Sets

Key : bucket01
Key : bucket02
Key : bucket03
⋮
Key : bucket20

Entity_Dictionaries

Key : properties
Key : collections
Key : indexes

Entity_Unique

Key

Entity_Aliases

Key

Entity_Properties

Key(uuid)

Entity_Index

Key : bucket01
Key : bucket02
Key : bucket03
⋮
Key : bucket20

Entity_Index_Entries

Key

Entity_Composite_Dictionaries

Key : container_entities

6. How it works: CRUD

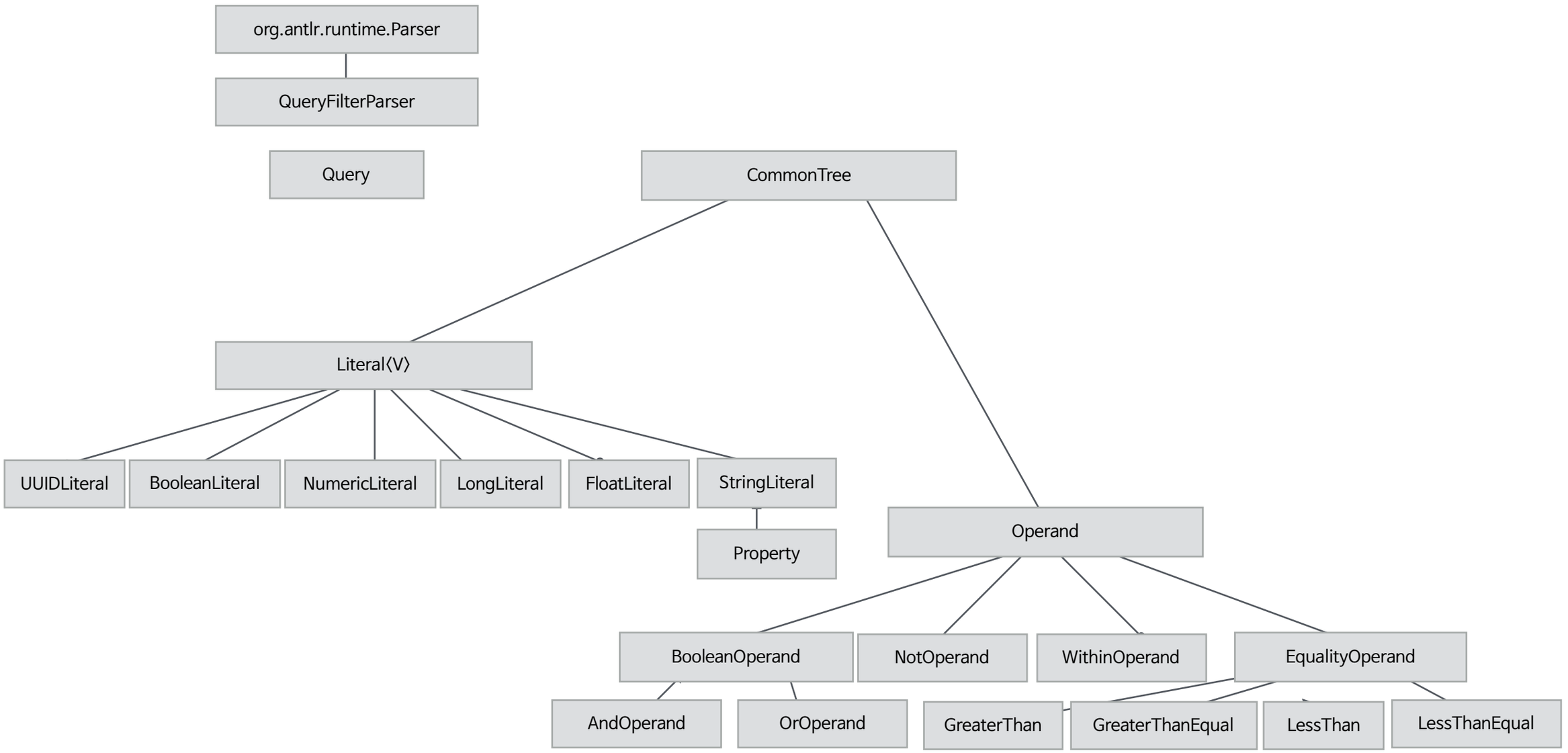
Read an entity 〉Retrieve entities by Query

1) Parse query parameter using ANTLR (represent Query class)
2) Represent QueryNode from Query class at QueryProcessor.process
3) Visit nodes in SearchVisitor.visit and then fetch data and merge, sort
    1) Algorithm : Merge Sort
        1) Fetch data and then sort in memory

6. How it works: CRUD

Read an entity 〉Retrieve entities by Query

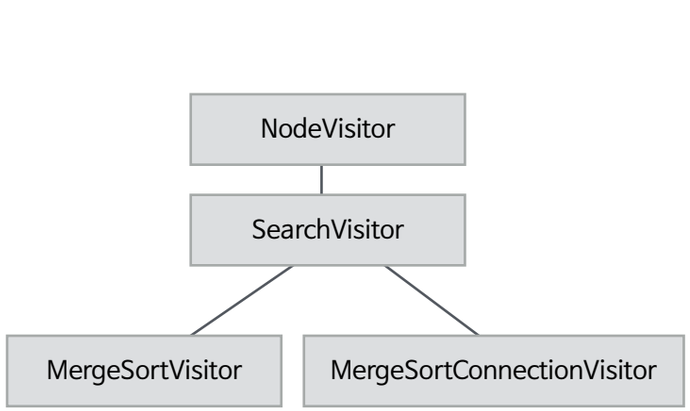
1) Parse query parameter using ANTLR (represent Query class)

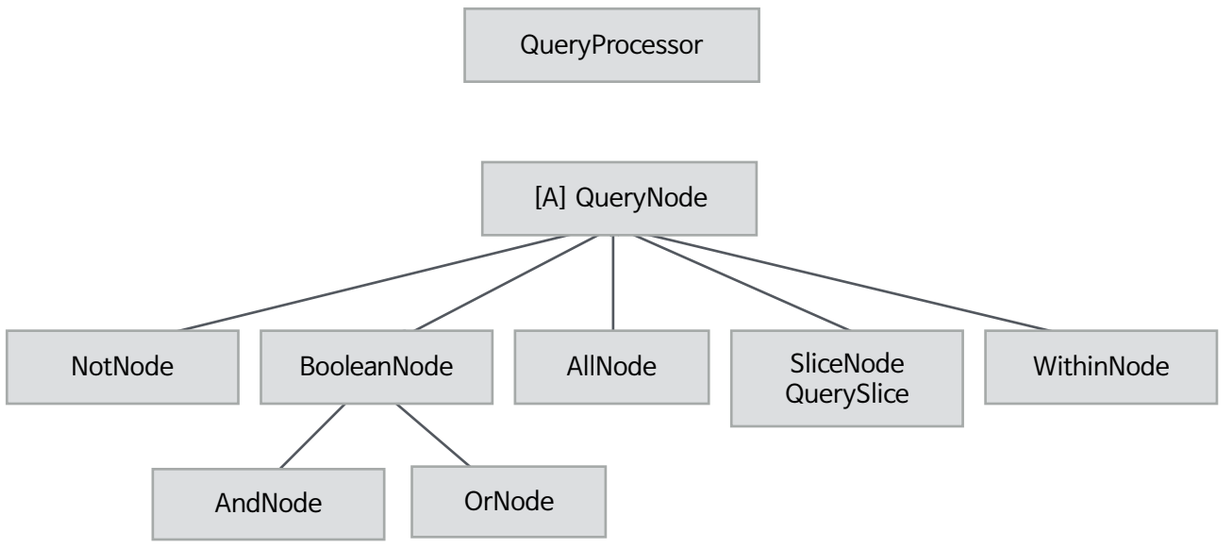


6. How it works: CRUD

Read an entity 〉Retrieve entities by Query

1) Parse query parameter using ANTLR (represent Query class)
2) Represent QueryNode from Query class at QueryProcessor.process
3) Visit nodes in SearchVisitor.visit and then fetch data and merge, sort



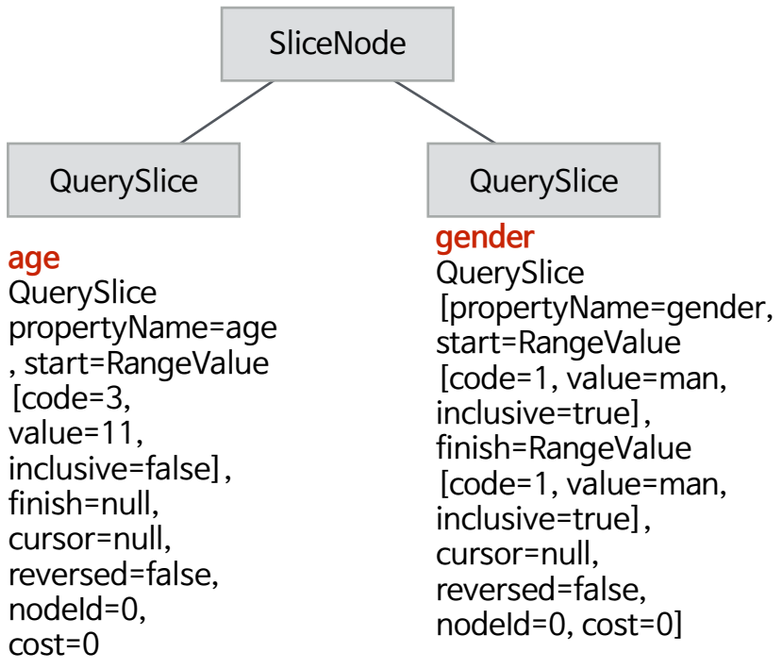


6. How it works: CRUD

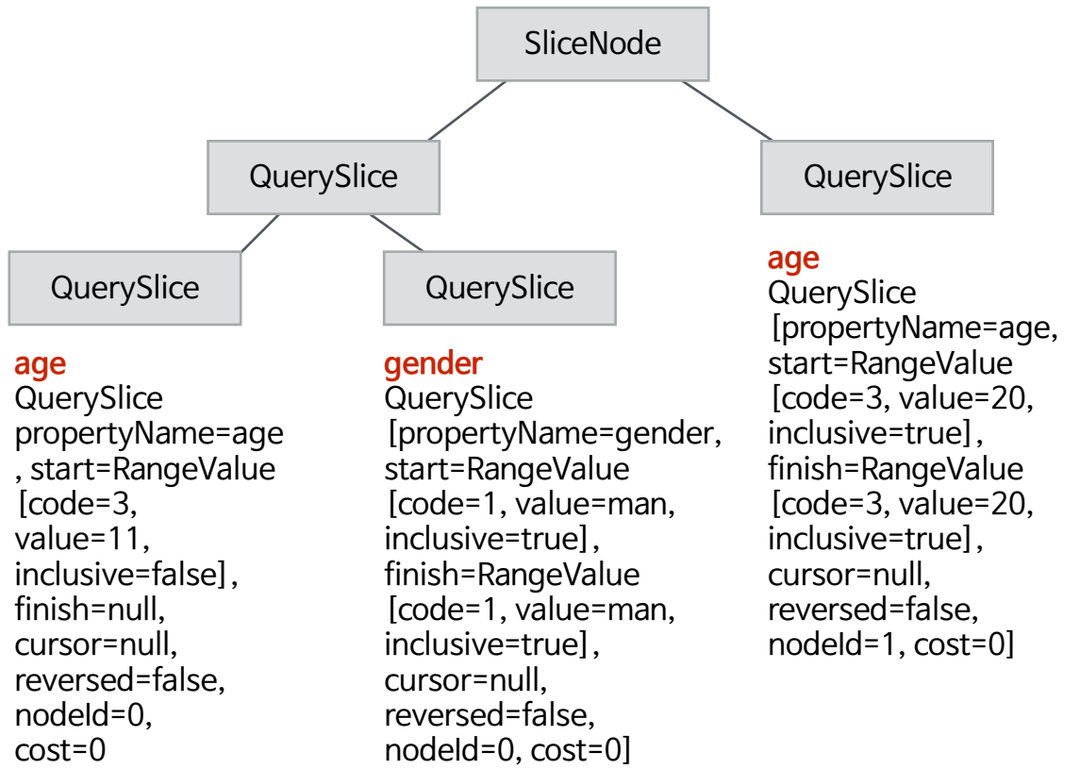
Read an entity 〉Retrieve entities by Query

1) Parse query parameter using ANTLR (represent Query class)
2) Represent QueryNode from Query class at QueryProcessor.process
3) Visit nodes in SearchVisitor.visit and then fetch data and merge, sort

ex) select * where age 〉11 and gender = 'man'



ex) select * where age 〉11 and gender = ‘man' or age = 20



6. How it works: CRUD

Read an entity 〉Retrieve entities by Query

1) Parse query parameter using ANTLR (represent Query class)
2) Represent QueryNode from Query class at QueryProcessor.process
3) Visit nodes in SearchVisitor.visit and then fetch data and merge, sort
    1) If logical operand(AND, OR, etc) is null && order by field == where field
       Simple Query
    else
       Complex Query
        1) select * where category = 'cat' order by category desc - Simple Query
        2) select * where category = 'cat' order by created desc - Complex Query
        3) select * where category = 'cat' AND girl = true order by category desc - Complex Query

6. How it works: CRUD

Read an entity 〉Retrieve entities by Query

Retrieve data through cassandra command (get, multiget, get_slice, multiget_slice)

Query : select * where gender = 'man' order by created desc

Entity_Index

Key : bucket01

Key : bucket02

Key : bucket03

⋮

Key : bucket20

IndexCode

- Bytes - 0
- String - 1
- UUID -2
- INT - 3

Key : appID:collection:property:bucket0

81….9:users:gender:0595….000

Columns

Index code	Property value	Entity Id	Time UUID
1	man	0177……01	…
1	women	0177……10	…
1	man	0177……20	…
1	women	0177……02	…

Key : appID:collection:property:bucket19

81….9:users:gender:0595….019

Columns

Index code	Property value	Entity Id	Time UUID
1	man	0177……03	…
1	women	0177……14	…
1	man	0177……16	…
1	women	0177……29	…

6. How it works: CRUD

Read an entity 〉Retrieve entities by Query

Algorithm I. Merge Sort

- Get entity ids from Entity_Index
- Get all properties from Entity_Properties
- Sort all entities by property in memory
- Limitation
  - Fetch size of multiget_slice
  - Slow performance if the fetch size of multiget_slice is large

Thanks

Do you want to contribute?

http://usergrid.incubator.apache.org/docs/contribute-code/

How to Contribute Code

contribute to this article on github

Usergrid is an open source project developed by folks like you. Anybody can contribute, but we do have some rules in place and we do like to review and discuss changes in public on our mailing lists and on our Github account.

We use Github as our code review and collaboration system and we use the Apache Git repo as our official repository of record, that's the code that we release when we make an official release.

We have two slightly different procedures for small fixes and for more significant changes.

For small fixes

We welcome small "drive-by" contributions from impatient developers! If you just have a small fix that you want to contribute then these are the steps you should follow to get your changes into Usergrid.

- **Fork usergrid/usergrid**. Fork this repo: https://github.com/usergrid/usergrid. You will work in your fork. Clone it to a directory on your computer and...
- **Make your changes** in your fork and don't forget to create tests for those changes.

References

- Open Source Mobile Backend on Cassandra
  http://www.slideshare.net/edanuff/open-source-mobile-backend-on-cassandra
- Cassandra Indexing Techniques
  http://anuff.com/2011/07/cassandra-summit-sf-2011-presentation/
- Secondary indexes in Cassandra
  http://anuff.com/2010/07/secondary-indexes-in-cassandra/
- Indexing in Cassandra
  http://anuff.com/2011/02/indexing-in-cassandra/
- “An Overview of Apache Cassandra” - Jihyun An(KT)