Apache Cassandra

INF 551

Wensheng Wu

History

- Early version developed at Facebook
- 2008: open-sourced
- 2009: became an Apache incubator project
- 2010: graduated to a top-level project

Installation

- wget http://mirrors.ibiblio.org/apache/cassandra/3.10/apache-cassandra-3.10-bin.tar.gz
- tar xvf apache-cassandra-3.10-bin.tar.gz
- cd apache-cassandra-3.10-bin

Starting Cassandra server

- bin/cassandra
  - Start cassandra server in background
- bin/cassandra –f
  - Start it in the foreground

Jdk requirement

- Cassandra 3.10 requires jdk1.8 or above
- Check jdk version
  - java -version
- Upgrade to java8 (if you have earlier version installed)
  - sudo yum remove java-devel
  - sudo yum install java-devel $\searrow$

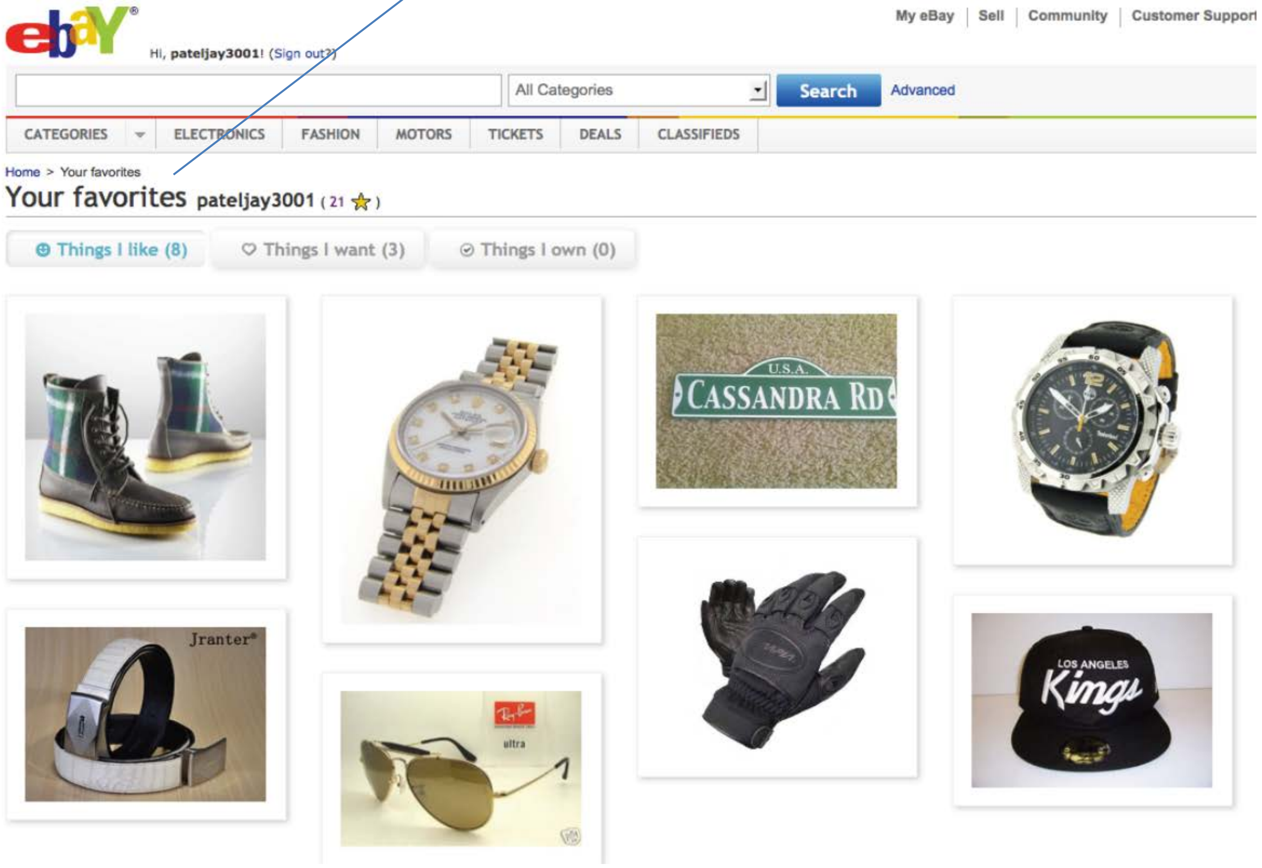
Amazon installs Java 8 now

Use cases

- Ebay
- Facebook
- Netflix
- Twitter
- …

A great variety of items with very different attributes

Now called "collections"



Cassandra

- An extensible record (wide column) store
  - Columns are grouped into column families
  - Column family ~ table (in RDBMS)
  - Each row belongs to a column family
- Rows are stored on disk in SSTables (sorted string table)
- Similar to Google's Bigtable

Sorted string table (SSTable)

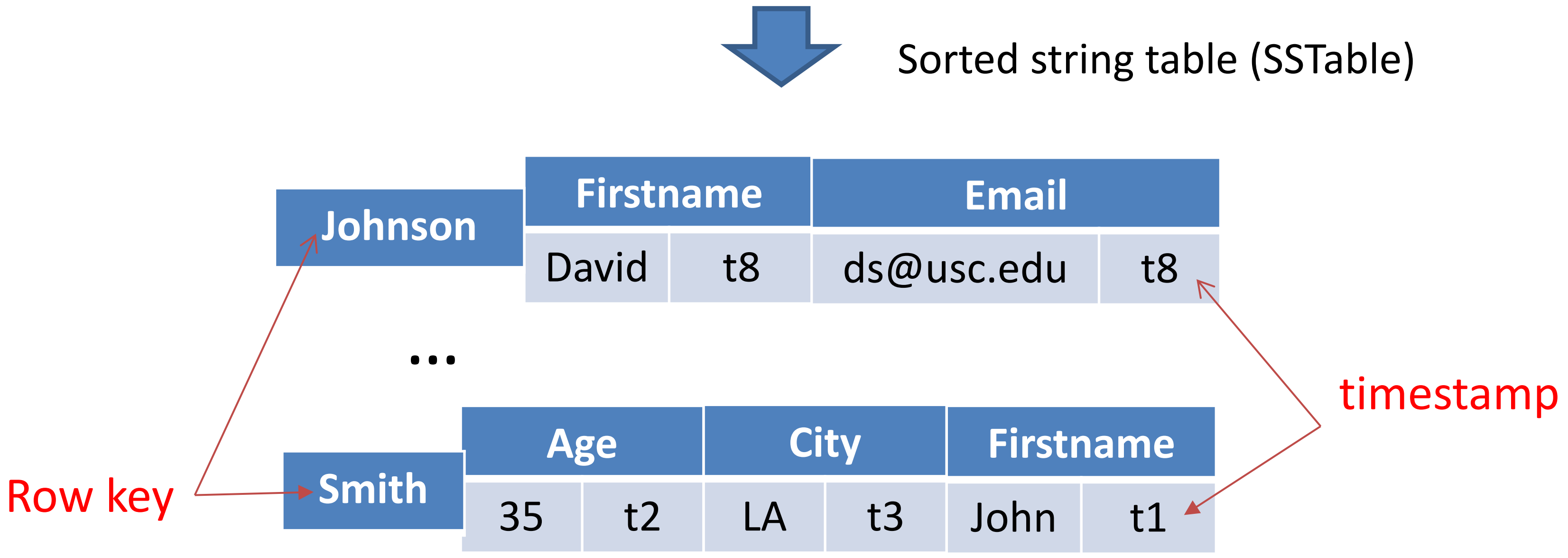
- SSTable stores rows of a table
  - Rows are sorted by row key
- Each row starts with a row key, followed by
  - A sorted list of columns (e.g., sorted by column name or timestamp)
- Each column contains:
  - Column name, column value, timestamp

SSTable

- Immutable
  - I.e., can not be modified, once created
- Two ways it can be created:
  - Flush in-memory data stored in memtable (via a process called minor compaction)
  - Merge a set of SSTables for the same column family (via major compaction)

Example: users table

lastname	firstname	age	email	city	gender
Smith	John	35		LA	
Johnson	David		ds@usc.edu		
Lou	Mary	44		LA	F
Lopez		38	lopez@gmail.com		F



Compared to RDBMS

lastname	firstname	age	email	city	gender
Smith	John	35	NULL	LA	NULL
Johnson	David	NULL	ds@usc.edu	NULL	NULL
Lou	Mary	44	NULL	LA	F
Lopez	NULL	38	lopez@gmail.com	NULL	F

NULL values need to be explicitly stored

Memtable & minor compaction

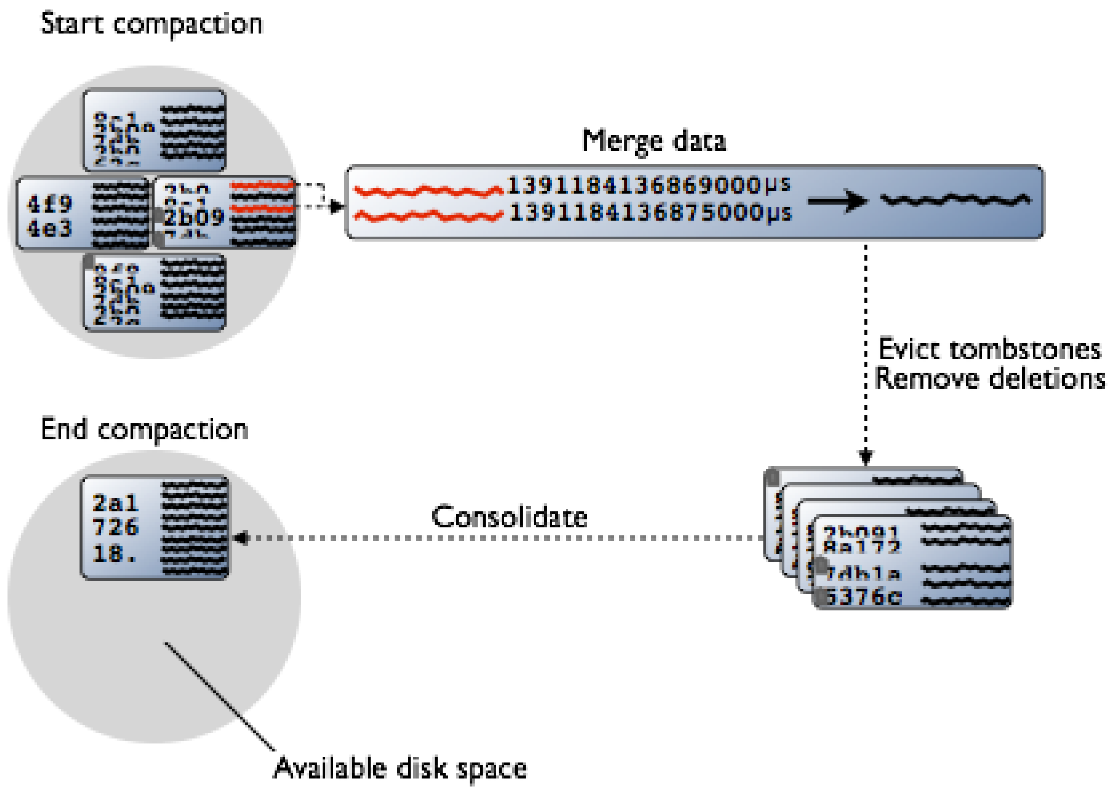
- In-memory structure holding new data & updates
- Typically one memtable per column family
- Flushed to disk as a new SSTable
  - when the size of the memtable exceeds some threshold
- This process is called minor compaction
  - It releases buffer pages & shrink memory usage

Major compaction

- Merge multiple SSTables into a single one
  - So that the number of SSTables won't grow out of bound
- Old data are removed & disk space is reclaimed
- Periodically performed in background

Major compaction

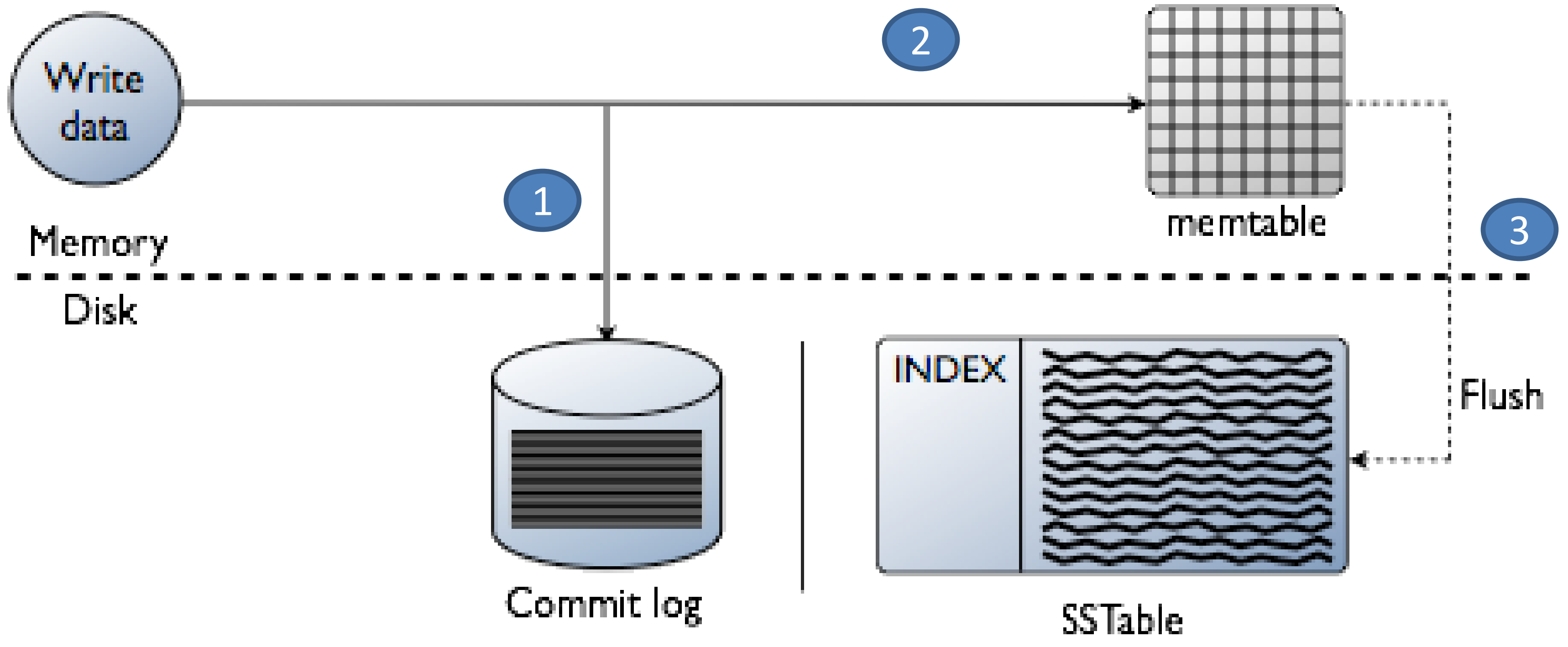
- Merging can be done efficiently
  - Since rows in SSTables are sorted by key



SSTable index

- Each SSTable has an index
  - Efficient lookup of row content from row key
- Location of index
  - Newer version: stored in a separate file
  - Old version: stored after all rows of table (in the same data file)

Write path


Write data
2
memtable
1
3
Memory
Disk
INDEX
Flush
Commit log
SSTable

Write path

- Write: insert/delete/update

1. A log entry is appended to a commit log file
2. Write data to memtable & acknowledge completion to client
3. When memtable is full, flush it as a new SSTable & purge corresponding entries from commit log
4. Periodically, merge SSTables of the same column family

Observations

- Write is fast
  - Only disk I/O is to write to commit log

- No overwrite or random write
  - Commit log and SSTables are both append-only

Read path

- Content of row is now distributed among
  - Memtable
  - Multiple SSTables

=> Read is expensive than write & may require:
  - disk access (to locate SSTables that contain fragments of row)
  - merging (row content in memtable and SSTables)

Keyspace (Cassandra)

- Analogous to a database in RDBMS
- Container for column families
- May specify different replication strategies for different keyspaces

Replication strategy

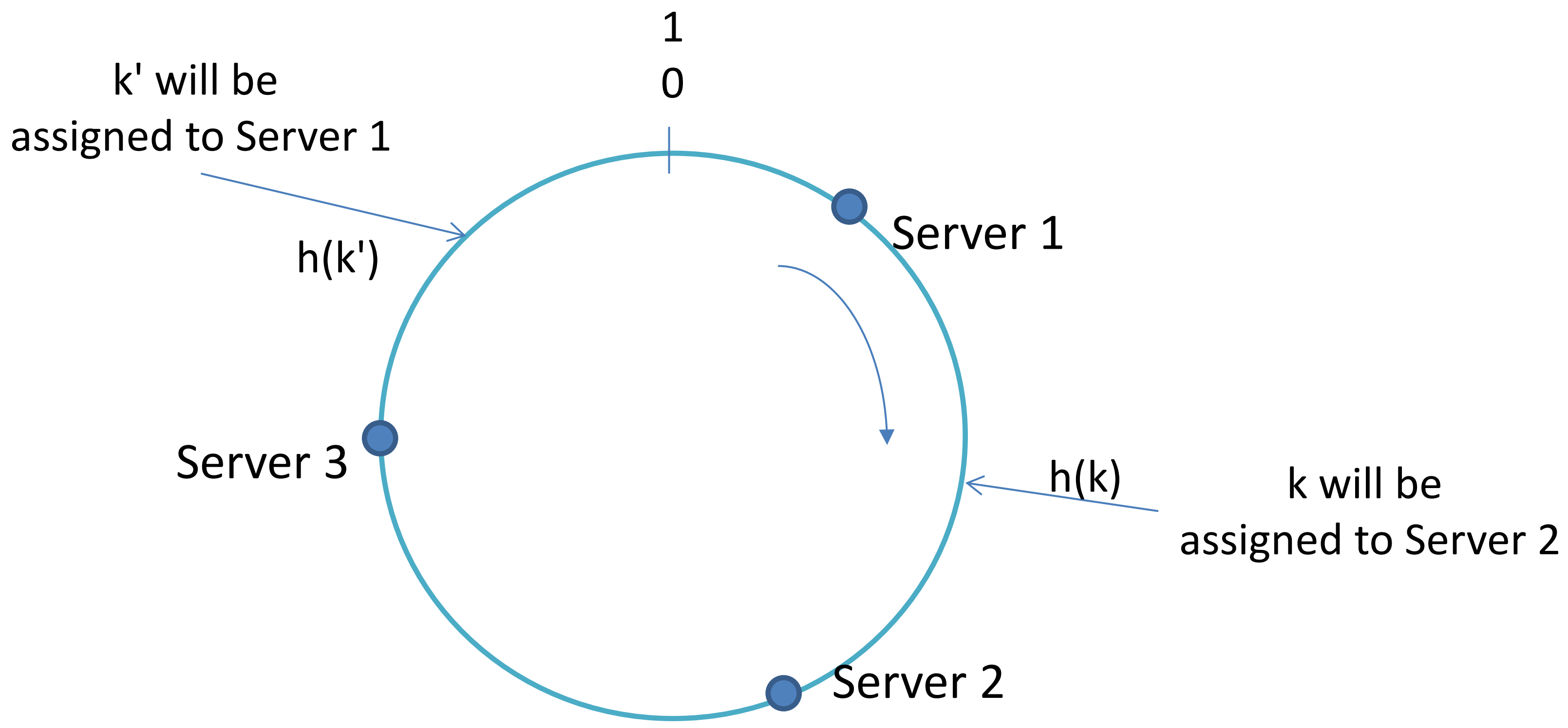
- How many replica?
- How to distribute them among nodes & data centers?

Simple replication strategy

- All replicas are in the same data center
- First replica on a node decided by consistent hashing
- Additional replica on next nodes clockwise in the ring
- Not rack-aware

Consistent hashing

- Recall Amazon DynamoDB



Network topology strategy

- For deployment in multiple data centers
  - Can specify # of replicas in each center
- Rack-aware:
  - In a center, additional replicas are placed on a node in different racks
- Note: nodes in the same rack tend to fail together
  - due to shared power, cooling, and networking

CQL

- Cassandra Query Language

- bin/cqlsh
  - Start CQL interactive shell

Keyspace

- DESC KEYSPACES;
  - List all keyspaces in the cluster

- CREATE KEYSPACE inf551 WITH replication = { 'class': 'SimpleStrategy', 'replication_factor': 1};

Keyspace

- desc inf551;
  - Show information about the keyspace inf551
- drop keyspace inf551;
- USE inf551;

Working with tables

- CREATE TABLE users (

  lastname text, -- UTF-8 string

  firstname text,

  age int,

  email text,

  city text,

  gender text,

  PRIMARY KEY (lastname));

Working with tables

- desc tables; -- list all tables
- desc users; -- list details of table "users"

- drop table users;

Column family

- create columnfamily user1(id int primary key, name varchar);

Equivalent to:

- create table user1(id int primary key, name varchar);

Query table

- SELECT * FROM users;
- SELECT * FROM users WHERE lastname= 'Smith';

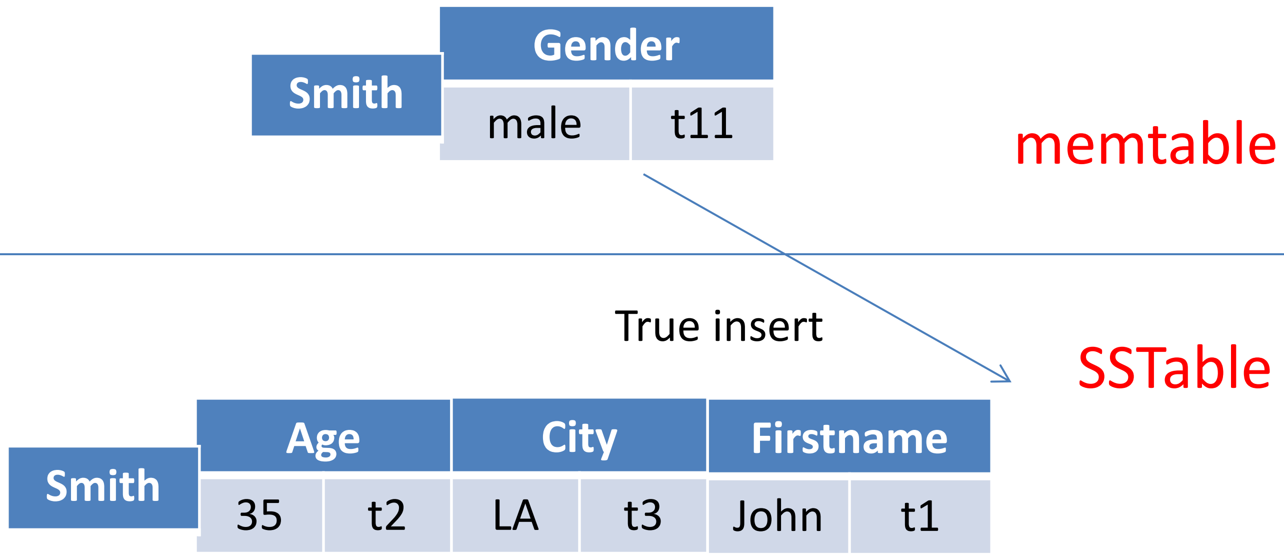
insert

- insert into users (lastname, age, city, firstname) values ('Smith', 35, 'LA', 'John');

- Note strings in CQL need to be single-quoted

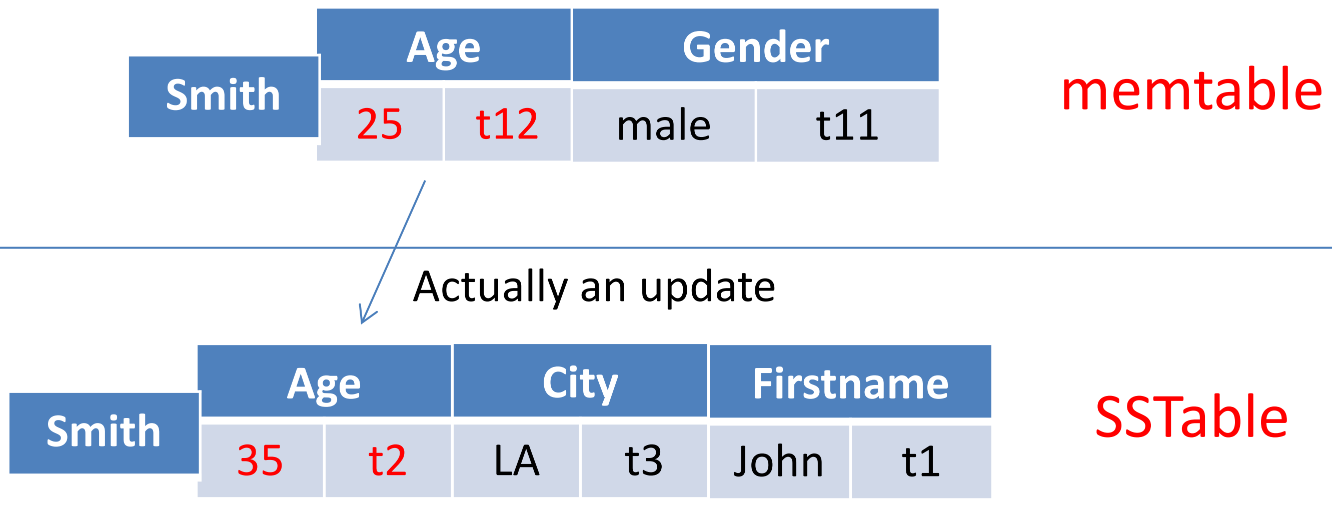
Insert

- insert into users (lastname, gender) values ('Smith', 'male');
  - Note that it does not check content of SSTable



Insert but actually an update

- insert into users (lastname, age) values ('Smith', 25);
  - This insert is actually an update (of age in SSTable)



memtable

	Age		Gender	
Smith	25	t12	male	t11

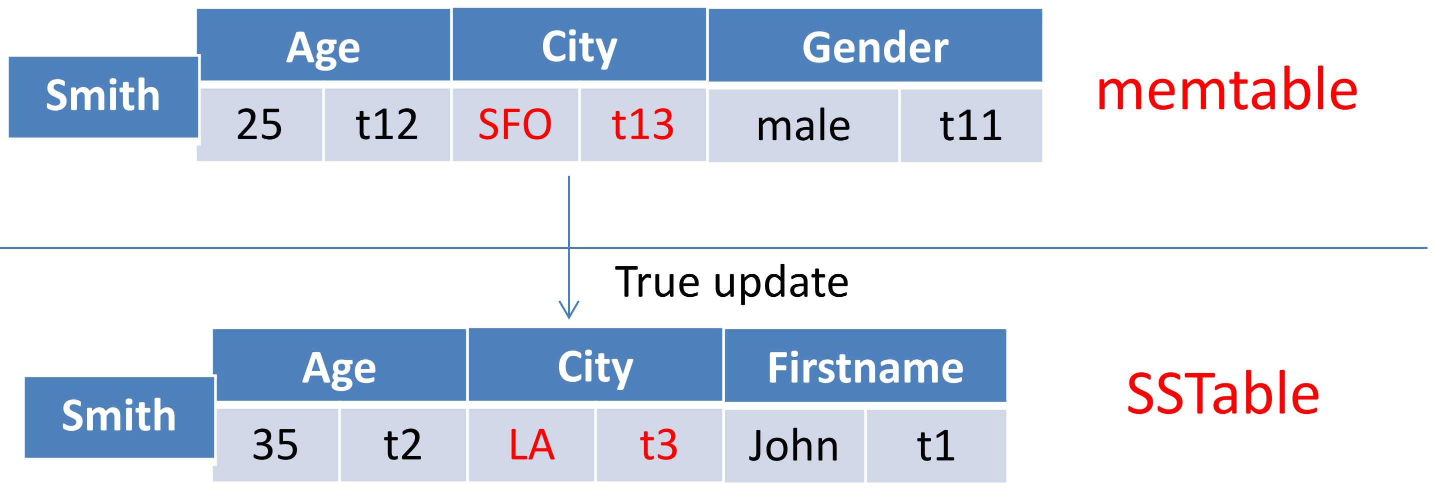
Actually an update

SSTable

	Age		City		Firstname	
Smith	35	t2	LA	t3	John	t1

Update

- update users set city = 'SFO' where lastname = 'Smith';



	Age		City		Gender	
Smith	25	t12	SFO	t13	male	t11

	Age		City		Firstname	
Smith	35	t2	LA	t3	John	t1

Update but actually an insert

- update users set email = 'john@usc.edu' where lastname = 'Smith';

memtable

	Age		City		Gender		Email	
Smith	25	t12	SFO	t13	male	t11	john@usc.edu	t14

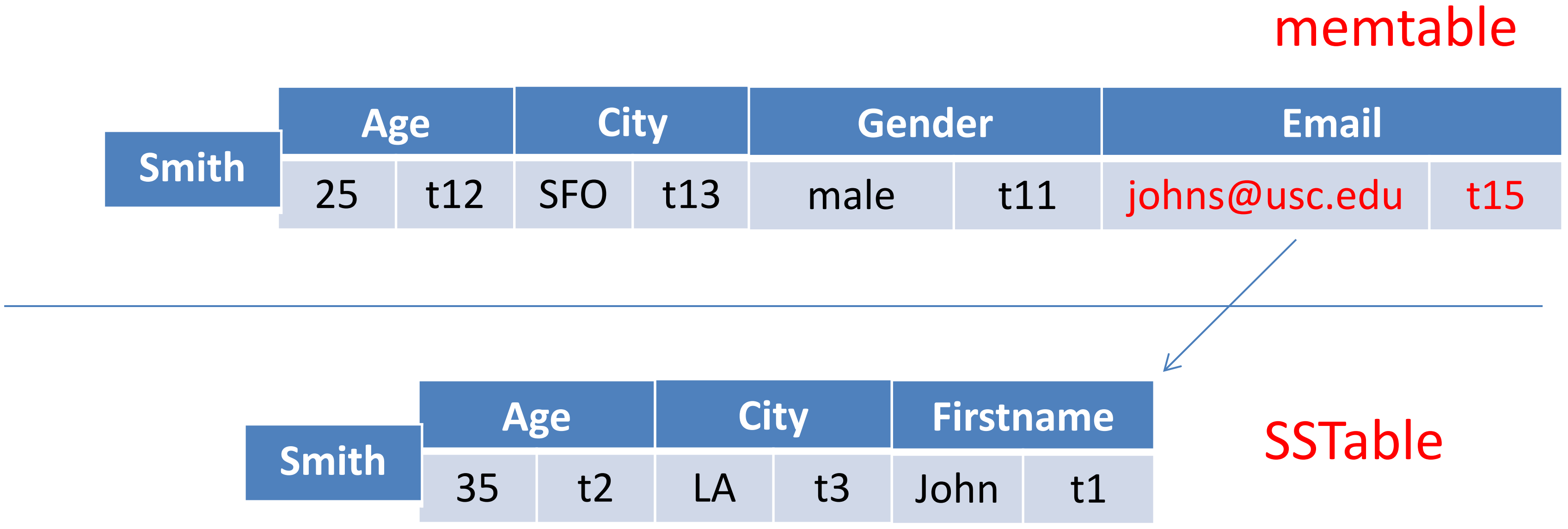
Actually an insert

	Age		City		Firstname	
Smith	35	t2	LA	t3	John	t1

SSTable

Insert but actually an update

- insert into users(lastname, email) values ('Smith', 'johns@usc.edu');



	Age		City		Gender		Email	
Smith	25	t12	SFO	t13	male	t11	johns@usc.edu	t15

	Age		City		Firstname	
Smith	35	t2	LA	t3	John	t1

Upsert

- Both update and insert are implemented as upsert
- Update if exists; otherwise, insert
- Insert if not exists yet; otherwise, update

Delete

- delete city from users where lastname = 'Smith';

	age		city		gender	
Smith	25	t12	[Tombstone]	t14	male	t11

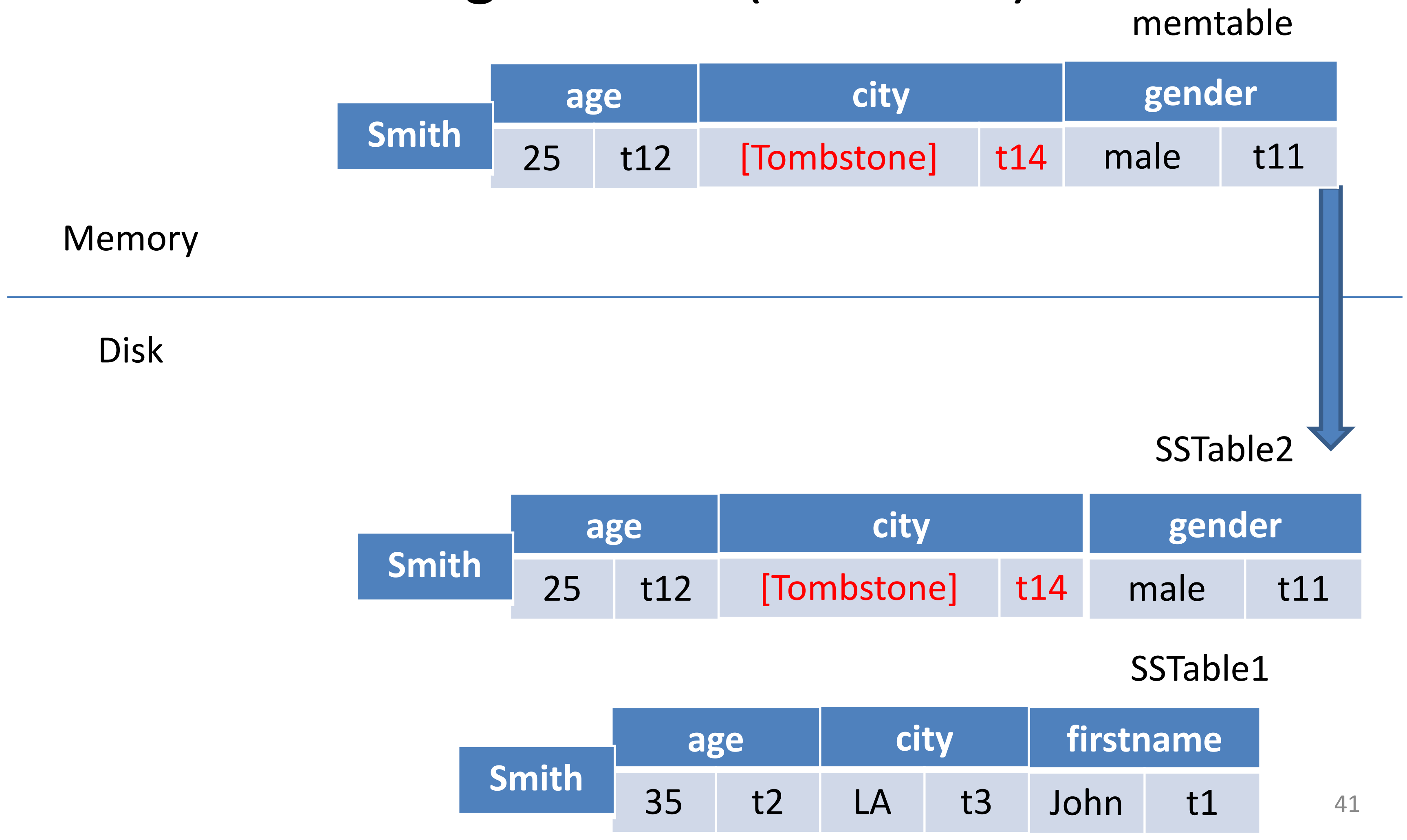
- Note this deletes a specific column
- When the last column of a row is deleted
  - The entire row will be removed!

Delete an entire row

- delete from users where lastname = 'Smith';
- A tombstone will be placed for the row

Minor compaction

- Note an existing SSTable (SSTable1) for users



memtable

	age		city		gender	
Smith	25	t12	[Tombstone]	t14	male	t11

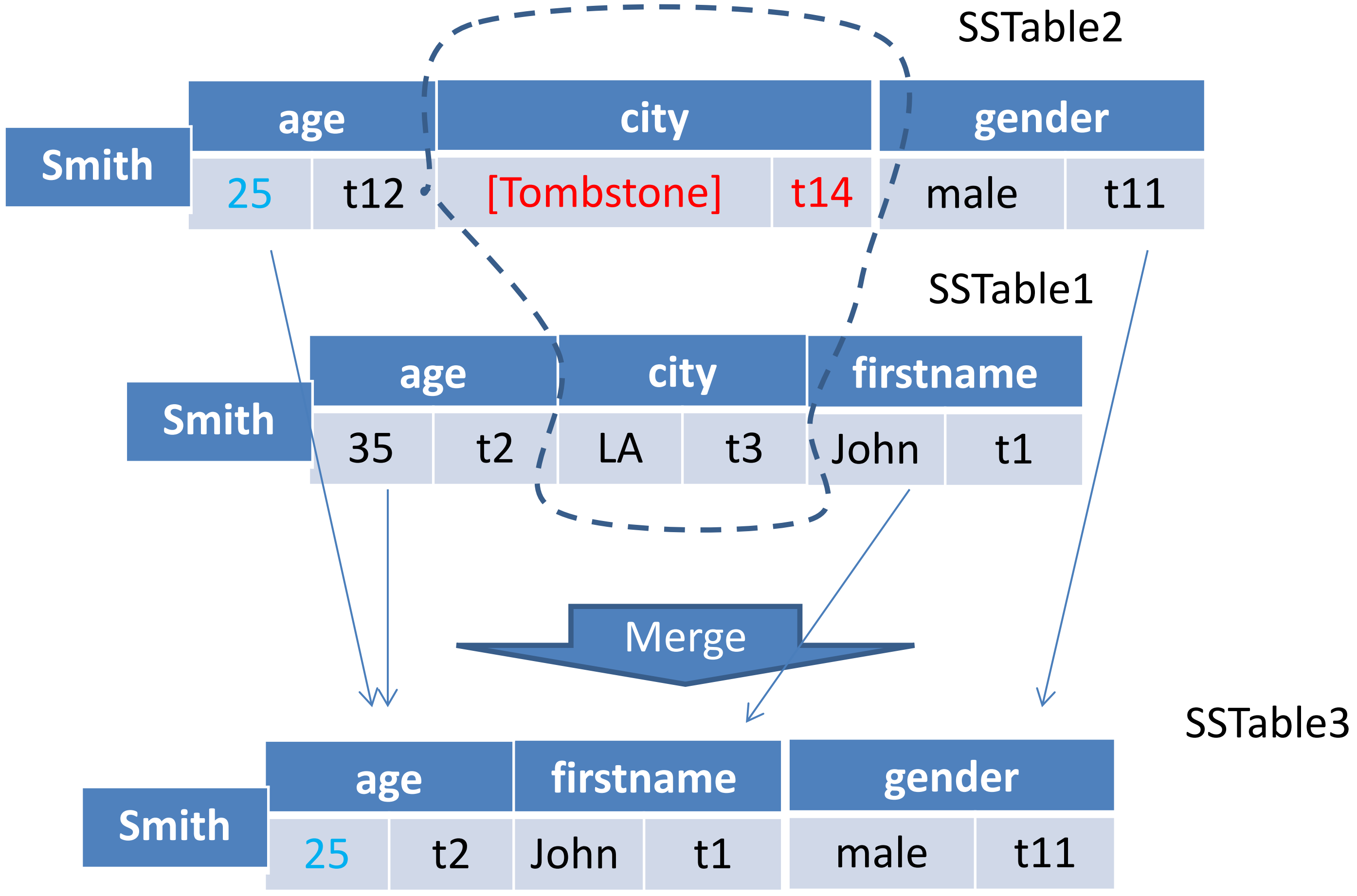
SSTable2

	age		city		gender	
Smith	25	t12	[Tombstone]	t14	male	t11

SSTable1

	age		city		firstname	
Smith	35	t2	LA	t3	John	t1

Major compaction



Range query

- Range query is not supported
- SELECT * FROM users WHERE lastname > 'Doe';
  - This does not work

Inequality query

- Inequality query not supported either
- SELECT * FROM users WHERE lastname != 'Doe';
  - This does not work

Query non-key attribute

- select * from users where age = 25;
  - This query is not supported
  - age is a non-key column
- Need to create a secondary index on age first

Secondary Index

- create index age_idx on users(age);
  - drop index age_idx;
- select * from users where age = 25;
  - This now works
- select * from users where age > 20;
  - This does not work; no range query

Compound key

- A primary key that contains multiple columns

- 1st column is the partition key
  - Decides how rows are distributed among nodes

- Remaining are clustering columns (similar to sort key in DynamoDB)
  - Decides how rows with same partition key are stored
  - Default: ascending (but may change it, see next)

Example

- CREATE TABLE playlists (

```
    id uuid,
    song_order int,
    song_id uuid,
    title text,
    album text,
    artist text,
    PRIMARY KEY (id, song_order)
);
```

Partition key

Clustering column
Default order: 1, 2, 3, 4, …

Change default clustering order

- CREATE TABLE playlists (

```
    id uuid,
    song_order int,
    song_id uuid,
    title text,
    album text,
    artist text,
    PRIMARY KEY (id, song_order)
) WITH CLUSTERING ORDER BY (song_order DESC);
```

Change default order

Example data

- INSERT INTO playlists (id, song_order, song_id, title, artist, album) VALUES (62c36092-82a1-3a00-93d1-46196ee77204, 1, a3e64f8f-bd44-4f28-b8d9-6938726e34d4, 'La Grange', 'ZZ Top', 'Tres Hombres');

- INSERT INTO playlists (id, song_order, song_id, title, artist, album) VALUES (62c36092-82a1-3a00-93d1-46196ee77204, 2, 8a172618-b121-4136-bb10-f665cfc469eb, 'Moving in Stereo', 'Fu Manchu', 'We Must Obey');

- INSERT INTO playlists (id, song_order, song_id, title, artist, album) VALUES (62c36092-82a1-3a00-93d1-46196ee77204, 3, 2b09185b-fb5a-4734-9b56-49077de9edbf, 'Outside Woman Blues', 'Back Door Slam', 'Roll Away');

Order by

- select *

  from playlists

  where id = 62c36092-82a1-3a00-93d1-
46196ee77204

  order by song_order desc;

Not supported

- select *

  from playlists

  where id = 62c36092-82a1-3a00-93d1-46196ee77204

  order by album;

- Album is not a clustering column

Data types in Cassandra

Data type	Value	Example
boolean, float, int, bigint, text, varchar, etc.	As usual	
list	Collection of ordered elements, e.g., list<int>	[12, 14]
map	List of key-value pairs, e.g., map<text, int>	{'home': 123, 'office': 456}
set	Collection of elements, e.g., set<int>	{12, 14}
UDT	User-defined	create type …

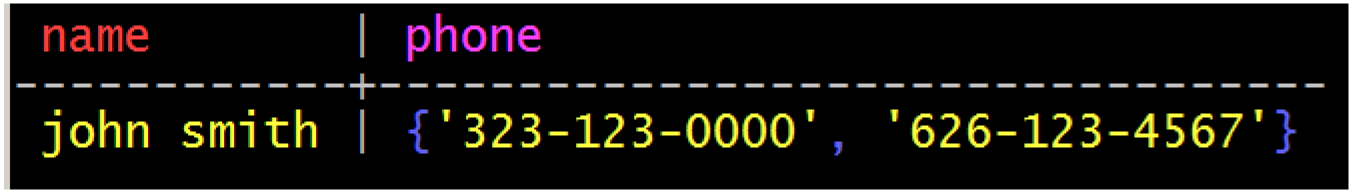
Collection data type

- List, map, set

- create table person (
  ```
  name text primary key,
  phone set<text>);
  ```

Example

- insert into person(name, phone) values ('john smith', {'626-123-4567', '323-123-0000'});



```
 name        | phone
-------------+----------------------------------
 john smith  | {'323-123-0000', '626-123-4567'}
```

Other notable features of Cassandra

- Tunable consistency
  - Can specify how many replicas need to be consistent for a write to be complete
  - To balance between consistency and latency
- No join
- No foreign key

Readings

- Bigtable: A Distributed Storage System for Structured Data
  - http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf

- Cassandra: a decentralized structured storage system
  - https://www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf