An Introduction to Cassandra Database

JHUG Meetup

A Gentle Introduction to Cassandra Database

Petros Kaklamanis

December 2016, May 2017

GENERAL

What is Cassandra (I)

AGILE ACTORS

- Fast Distributed DB
- High Availability
- Near-Linear Horizontal Scalability

C*

What is Cassandra (II)

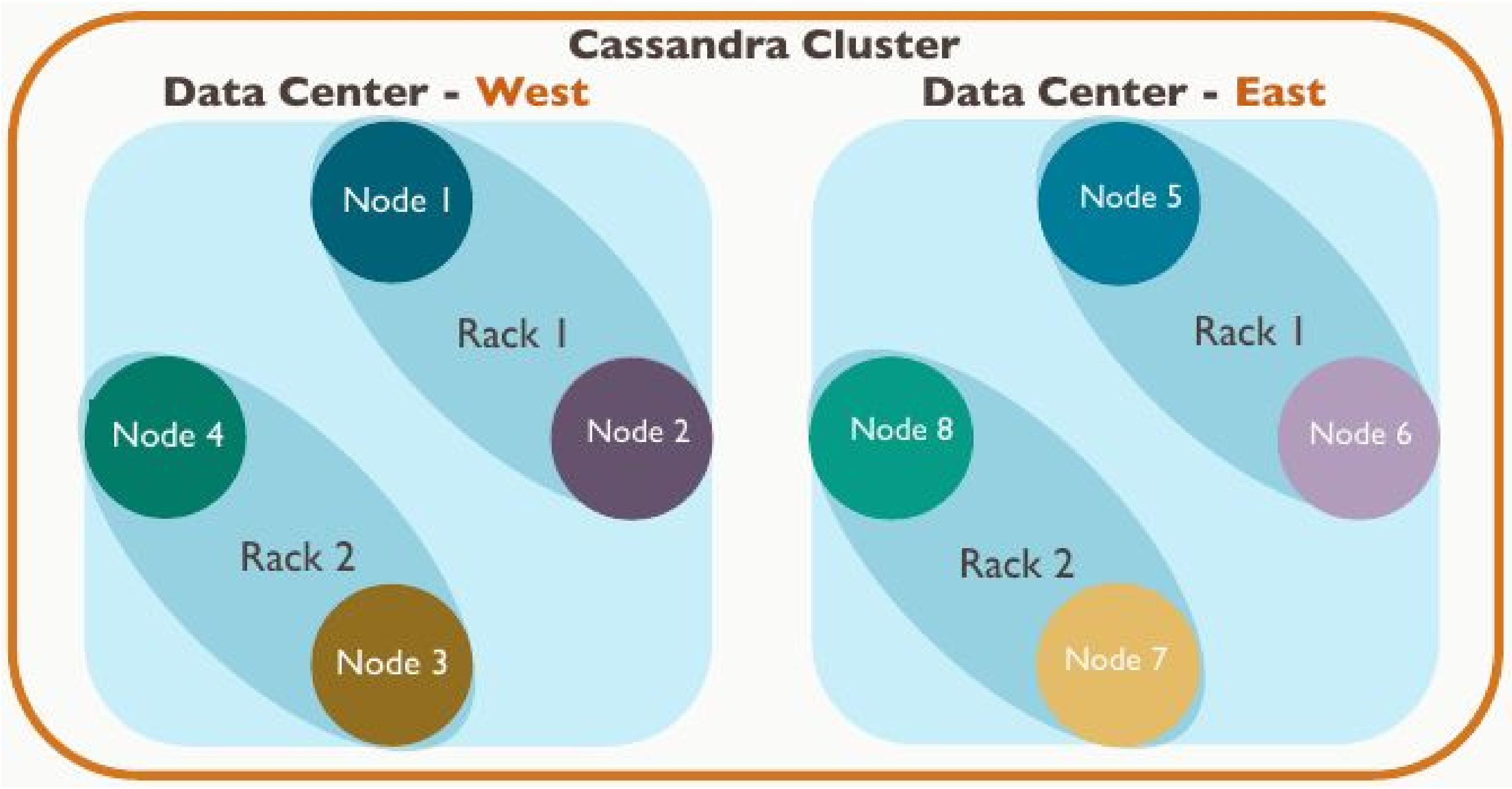
- Predictable Performance
- Fault Tolerance (Peer-to-Peer)
- Cannot replace RDBMS ad hoc
    - Data Model is different
    - Transaction mechanism is different
        - it is not ACID
- Current version is 3.10, released in February 2017

Origins

- Google Big Table
  - Storage Model
- Amazon Dynamo
  - Distribution backbone
- Facebook integrated these two (2008)
  - Later released as Cassandra
  - Nowadays an Apache project

Structure (I)

AGILE ACTORS

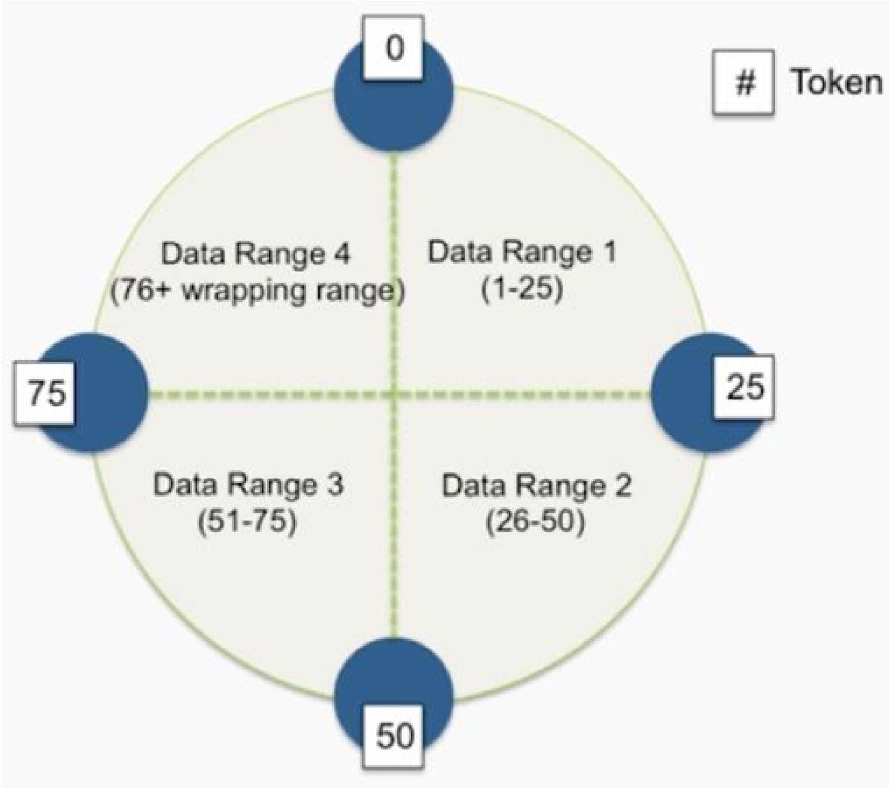


C*

Structure (II)

AGILE ACTORS

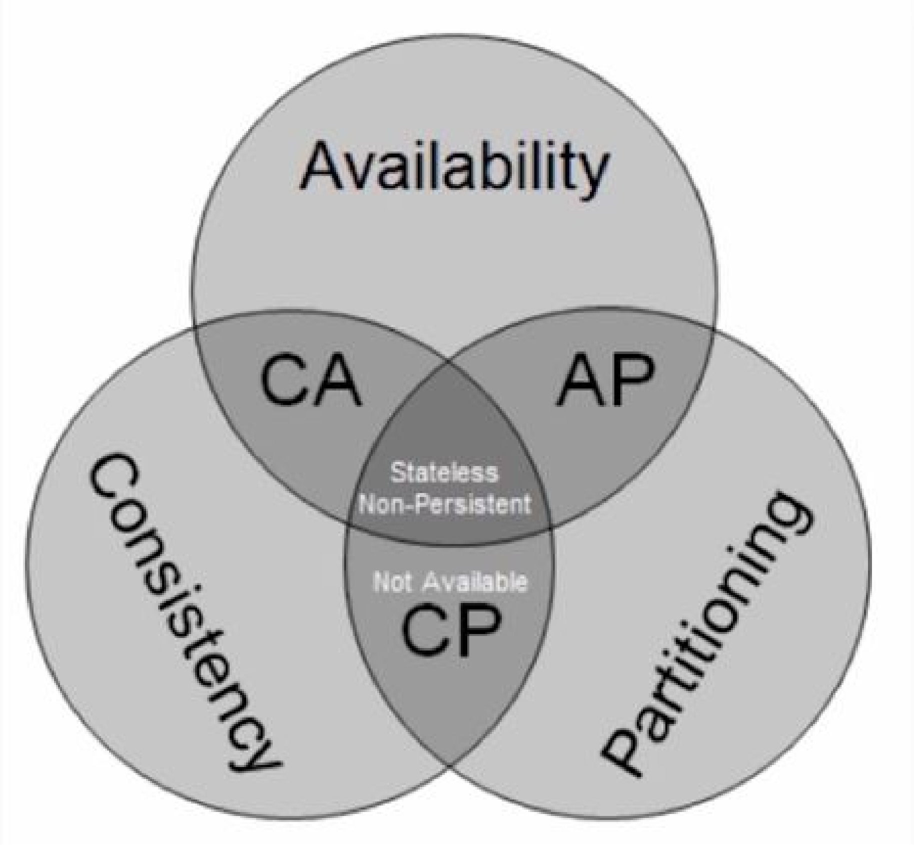
- Hash Ring
- P2P
- Data partitioning
- Replication across peers



C*

CAP theorem (I)

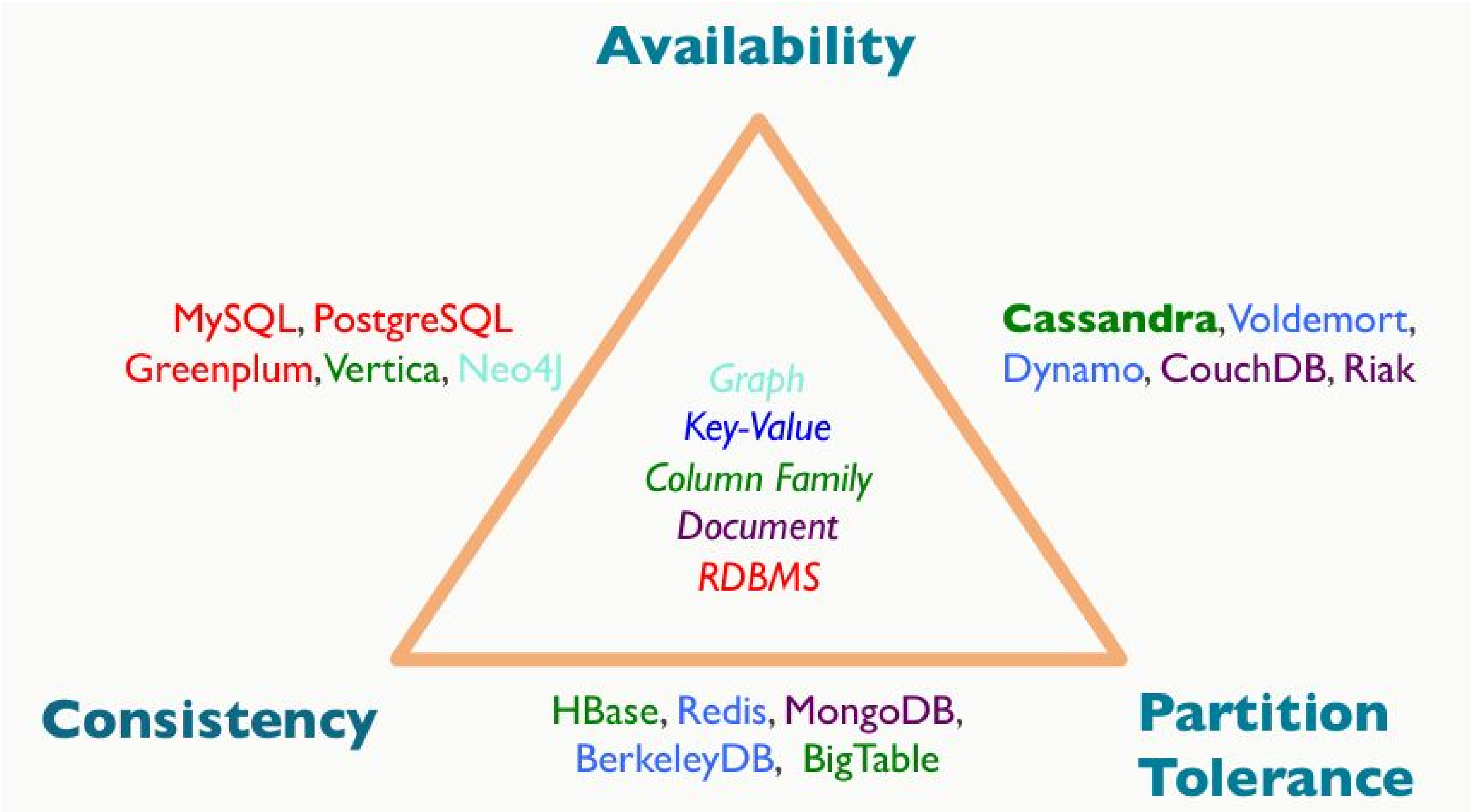
- Consistency Availability Partitioning trade-off
- Partitioning = Partition Tolerance = same network or not
- C* design choice:
  - A, P over C



C*

CAP theorem (II)

AGILE ACTORS



C*

Common Use Cases

- From: http://www.datastax.com/tag/use-case/
- Product Catalog and Playlist
- Recommendation and Personalization
- Fraud Detection
- Messaging
- IOT and Sensor Data
- Marketing and Advertising
- Social Media and Networking

INTERNALS

Replication Factor

AGILE ACTORS

- Definition “how many copies of our data, do exist in a cluster” (RF)
- Data is always replicated
- RF is defined and configured for a KeySpace per Data Center
- A KeySpace is a “collection of Tables”

Multiple Data Centers

- DCs can be physical or logical
- Asynchronous replication to other DCs
- CREATE KEYSPACE hospital

```
WITH REPLICATION = {
    'class' : 'SimpleStrategy',
    'replication_factor' : 3
};
```

Consistency Level

AGILE ACTORS

- Definition “How many replicas respond Properly to a query” in order to consider the query successful
  - A query can be a Read or a Write
- Examples: ALL, QUORUM, ONE
- Consistency Level (CL) affects performance and availability (fault-tolerance)
- CL is configured per query
  - This enables using C* even in **CA**P mode

Consistency Level Details

AGILE ACTORS

- Several are available
- Defined per request, by default ONE

Name	Description	Usage
ANY (writes only)	Write to any node, and store *hinted handoff* if all nodes are down.	Highest availability and lowest consistency (writes)
ALL	Check all nodes. Fail if any is down.	Highest consistency and lowest availability
ONE (TWO, THREE)	Check closest node to coordinator.	Highest availability and lowest consistency (reads)
QUORUM	Check quorum of available nodes.	Balanced consistency and availability

C*

Consistency Level Trade-Off

- Consistency Level ALL
  - **Consistent Read**,

    Highest latency, Lowest availability
- Consistency Level ONE
  - Maybe inconsistent Read,

    Lowest latency, **Highest availability**
- Consistency Level QUORUM
  - Consistent Read (if both Read/Write are QUORUM), Medium latency, Medium availability

Immediate Consistency (I)

- Immediate Consistency
  - Reads always return the most recent data
- We achieve this by configuring
  - CL per Read, Write
  - RF per KeySpace
- It must hold:
- Practically, does it worth it?
  - CL ONE is enough in most cases

Immediate Consistency (II)

Configuration examples for a Cluster with 4 Nodes:

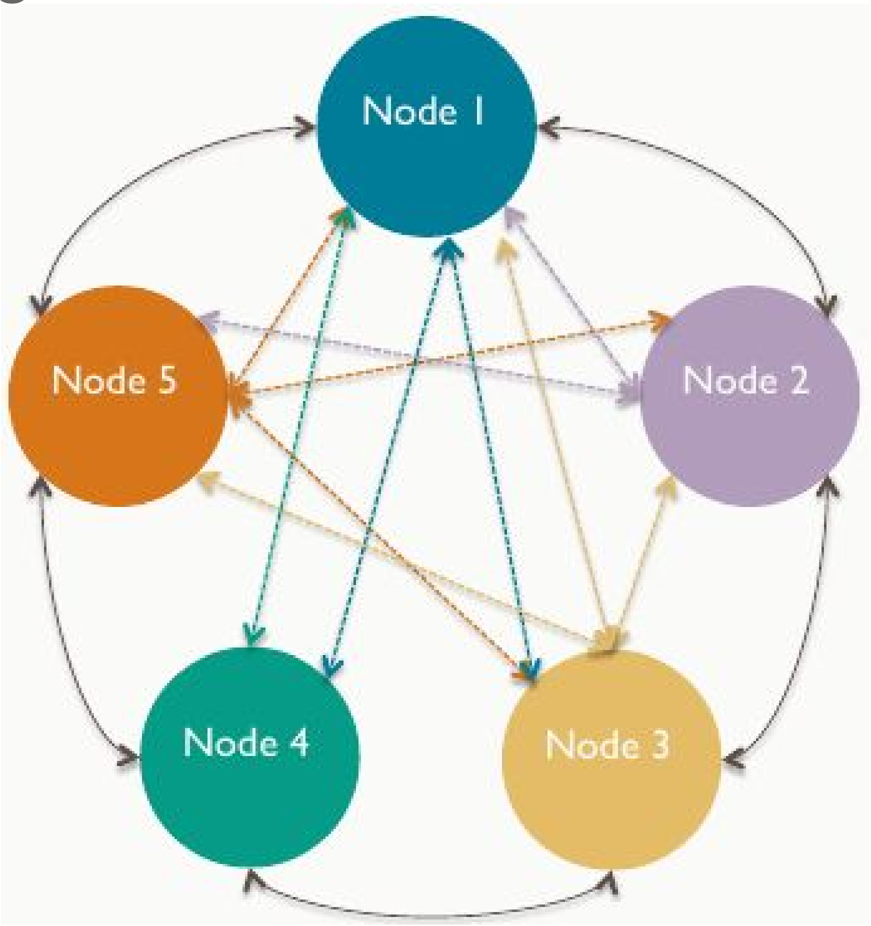
1. Frequent Read operations:
   a. RF = 3
   b. CL_{Read} = QUORUM, CL_{Write} = QUORUM
2. Frequent Write operations:
   a. RF = 3
   b. CL_{Read} = ALL, CL_{Write} = ONE

Cluster internal communication

- Nodes continuously communicate and exchange information
- Two central mechanisms
    - Gossip
    - Snitch

Every one second, each Node contacts 1 to 3 others, sending and requesting timestamped updates about known Nodes

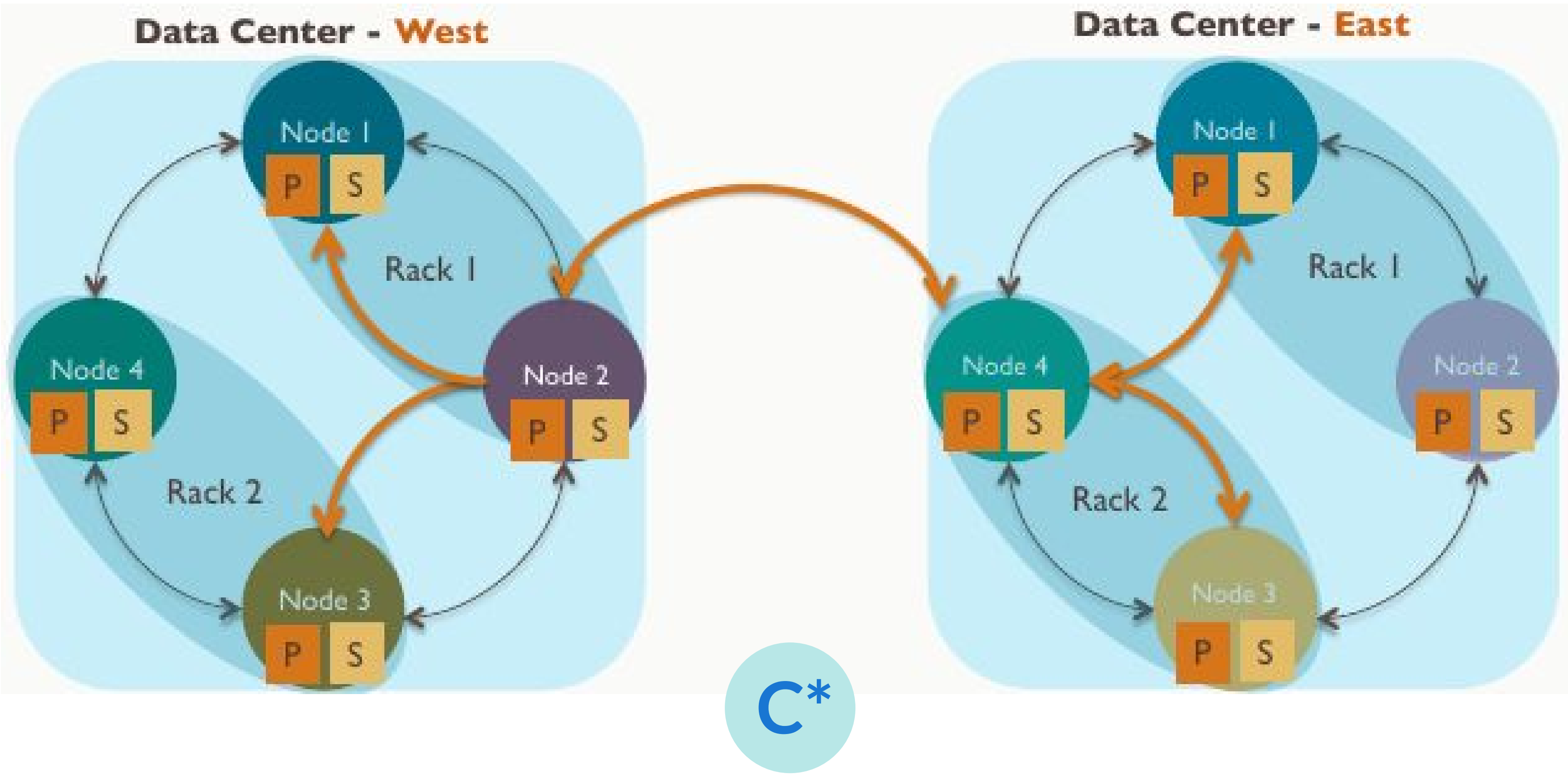
- states
- locations



C*

AGILE ACTORS

This is how Nodes know about the rack and data center topology



CONFIGURATION FILES & TOOLS

Installation

- Requirements for CPU, RAM, HDD
- Operating System
- NTP – C* requires synchronized clocks
- Disable memory swaps
- Java: Oracle JDK
- Network configuration
- C* installation
- C* configuration

Distributions

AGILE ACTORS

- Apache Cassandra http://cassandra.apache.org/
- DataStax Community Edition (DSC)
  - Additional tools for managing a Cluster
- DataStax Enterprise Edition (DSE)
  - More features than DSC, better for Analytics
  - Special program for start-ups
- http://www.datastax.com/products/datastax-enterprise

Configuration Files

- Located under **$CASSANDRA_HOME/conf/**
  - Example: dsc-cassandra-2.1.10/conf/
- Most important files:
  - cassandra.yaml
  - cassandra-env.sh
  - logback.xml
  - cassandra-rackdc.properties
  - cassandra-topology.properties

C* Tools

- Located under
  - **$CASSANDRA_HOME/bin/**
  - **$CASSANDRA_HOME/tools/**
- Tools
  - nodetool
  - cqlsh
  - cassandra-stress
  - sstable2json, json2sstable
  - Cassandra Cluster Management – CCM (DataStax)
  - DevCenter (DataStax)

CASSANDRA DATA MODEL

C* Data Model (I)

- Data is stored and organized in a Column Family
- A Column Family is comprised of Rows
- A Row is the smallest unit that stores related data

C* Data Model (II)

- A Partition (old name: RowKey) uniquely identifies a Row in a Column Family

- It stores data in Cells

- Cell parts
  - column name
  - column value
  - data creation timestamp

- Maximum cell size (column value)
  - 2 GB in theory
  - 100MB in praxis

C* Data Model (III)

1. A Table is a 2D view of a column family
    a. A table has Partitions
    b. A Partition may be a single row or multiple rows
2. A Partition Key uniquely identifies a Partition
    a. Can be composite
    b. It is hashed by the partitioner system to determine which Node will store it

C* Data Model (IV)

- A Primary Key uniquely identifies a row
  - Can be composite
  - It is comprised of two parts
    - the Partition Key
    - optionally, further columns

- Data Definition Language (DDL) describes Tables, Partition Keys, Primary Keys

Data in Clustering Columns (I)

AGILE ACTORS

For table Videos below:

```
CREATE TABLE Videos
( id INT, name TEXT, year INT, runtime INT,
 PRIMARY KEY ((year), name)
);
```

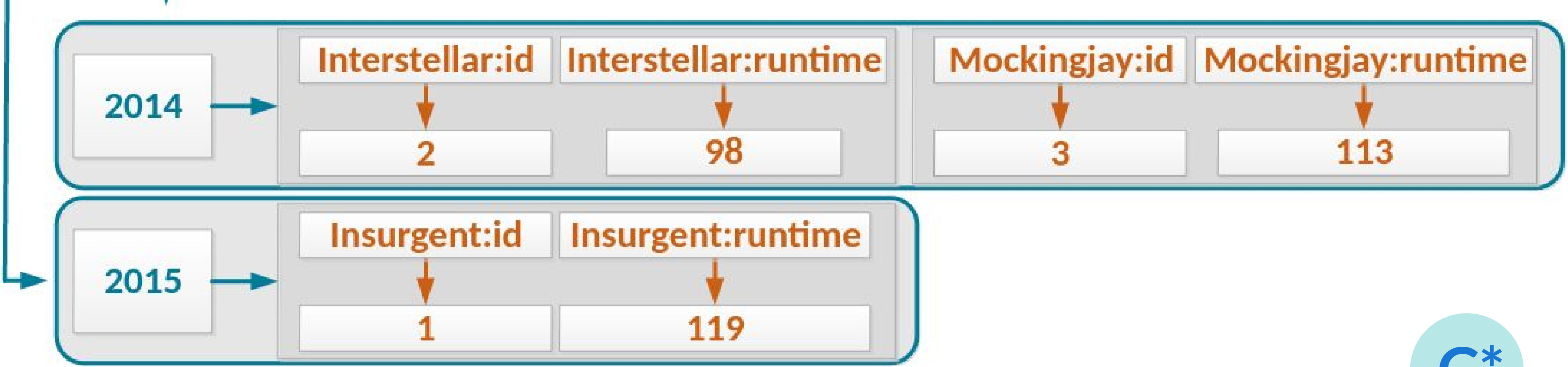
year	name	id	runtime
2015	Insurgent	1	119
2014	Interstellar	2	98
2014	Mockingjay	3	122

C*

Data in Clustering Columns (II)

Clustering columns divide Rows among partitions

year	name	id	runtime
2015	Insurgent	1	119
2014	Interstellar	2	98
2014	Mockingjay	3	122



AGILE ACTORS

CASSANDRA QUERY LANGUAGE

Cassandra Query Language

AGILE ACTORS

- Language for communicating with the C* DB
- Abbreviated as CQL
- Similar to SQL
- It can create, modify, delete tables and data

CQL Basic Data Types

AGILE ACTORS

CQL Type	Constants	Description
ASCII	strings	US-ASCII character string
BIGINT	integers	64-bit signed long
BLOB	blobs	Arbitrary bytes (no validation), expressed as hexadecimal
BOOLEAN	booleans	true or false
COUNTER	integers	Distributed counter value (64-bit long)
DECIMAL	integers, floats	Variable-precision decimal
DOUBLE	integers	64-bit IEEE-754 floating point
FLOAT	integers, floats	32-bit IEEE-754 floating point
INET	strings	IP address string in IPv4 or IPv6 format*
INT	integers	32-bit signed integer
LIST	n/a	A collection of one or more ordered elements
MAP	n/a	A JSON-style array of literals: { literal : literal, literal : literal ... }
SET	n/a	A collection of one or more elements
TEXT	strings	UTF-8 encoded string
TIMESTAMP	integers, strings	Date plus time, encoded as 8 bytes since epoch
TUPLE	n/a	Up to 32k fields
UUID	uuids	A UUID in standard UUID format
TIMEUUID	uuids	Type 1 UUID only (CQL 3)
VARCHAR	strings	UTF-8 encoded string
VARINT	integers	Arbitrary-precision integer

C*

CQL: Create Table

CREATE TABLE cars_by_cost (

- brand TEXT, // part of Partition Key
- model TEXT, // part of Partition Key
- cost DECIMAL, // Clustering Key
- merchant TEXT,
- PRIMARY KEY ((brand, model), cost)

) WITH CLUSTERING ORDER BY (cost ASC);

CQL: Modify Table

- ALTER TABLE cars_by_cost ADD cc INT;
- ALTER TABLE cars_by_cost

  ALTER cc TYPE BIGINT;
  - Types must be compatible
- ALTER TABLE cars_by_cost DROP cc;

CQL: Remove or Empty Table

- To fully remove a Table:
- DROP TABLE cars_by_cost;

- To clear all data (delete all Partitions) from a Table – but spare the Table:
- TRUNCATE cars_by_cost;

CQL: Read Data (I)

AGILE ACTORS

- General syntax:

- **SELECT columns**
  FROM table
  WHERE relations
  ORDER BY clustering_column ASC/DESC
  LIMIT number
  ALLOW FILTERING;

CQL: Read Data (II)

- Typical cases
  - Beware, these examples include Anti-Patterns!

- SELECT brand, merchant FROM cars_by_cost;
  - Avoid retrieving all partitions and rows unless absolutely necessary

- SELECT * FROM cars_by_cost;
  - Avoid retrieving all columns unless necessary

CQL: Read Data (III)

- SELECT * FROM cars_by_cost
  WHERE brand = 'b' AND model = 'm';
  - To retrieve a partition, values for **all** partition columns are needed

- SELECT * FROM cars_by_cost
  WHERE brand = 'b' AND
  model = 'm' AND cost < 1000;
  - To retrieve a row, values for **all** partition and clustering columns (primary key) are needed

CQL: Read Data (IV)

AGILE ACTORS

- Trick of ALLOW FILTERING
  - Allows scanning over all partitions and the predicate needs not give values for all partition columns
  - May lead to slow queries with large result set

CQL: Indexing to Read Data

AGILE ACTORS

- Used to allow queries on normal columns
- Two types of Indexes
  - Secondary Indexes
    - traditional
  - Custom Indexes: SASI
    - SSTable Attached Secondary Index
    - Since C* v3.4
- Indexes usage is NOT a spontaneous decision, but a well thought one

CQL: Secondary Indexes

For exact matches

- **CREATE INDEX** merchant_idx
  ON cars_by_cost (merchant);
  - SELECT * FROM cars_by_cost
    WHERE **merchant = 'm'**;
  - SELECT * FROM cars_by_cost
    WHERE brand = 'b' AND **merchant = 'm'**;
- **DROP INDEX** merchant_idx;

CQL: Custom/SASI Indexes (I)

For partial matches:

```
CREATE CUSTOM INDEX merchant_idx
ON cars_by_cost (merchant)
USING 'org.apache.cassandra.index.sasi.SASIIndex'
WITH OPTIONS = {
   'mode': 'CONTAINS',
   'analyzer_class': 'org.apache.cassandra.index.sasi
                         .analyzer.NonTokenizingAnalyzer',
   'case_sensitive': 'false'
};
```

CQL: Custom/SASI Indexes (II)

Query using a SASI Index:

- **SELECT** *

  FROM cars_by_cost

  WHERE merchant **LIKE** '%son%' ;

CQL: Additional Functions

- Aggregation related
  - count(), min(), max(), sum(), avg(), ...
- Time related
  - now(), dateof(), ...
- Blob conversion related
  - bigintAsBlob, blobAsBigint, ...
- User Defined Functions are also possible!
  - To be executed within C*, thus written in Java

CQL: Create (Insert) Data

- INSERT INTO

  cars_by_cost (brand, model, cost, merchant)

  VALUES ('volvo', 'xc90', 9999, 'daves');

- What does it do?
  - Creates non-existing partitions
  - **But also updates existing partitions**

CQL: Update Data

- UPDATE cars_by_cost
  SET merchant = 'pauls'
  WHERE brand = 'volvo'
  AND model = 'xc90'
  AND cost = 9999;
- What does it do?
  - Updates existing partitions
  - **But also creates non-existing partitions**

CQL: Upsert Data

- Insert and Update have the notion of Upsert
  - Update or Insert
- Why?
  - Because of the way data is organized into Clustering columns

CQL: Delete Data (I)

AGILE ACTORS

- Deleting a Partition
- DELETE FROM cars_by_cost

  WHERE brand = 'b' AND model = 'm';

- Deleting a Row
- DELETE FROM cars_by_cost

  WHERE brand = 'b'

  AND model = 'm'

  AND cost = 1000;

CQL: Delete Data (II)

AGILE ACTORS

- Deleting (setting to NULL) a cell from a Row
- DELETE merchant FROM cars_by_cost

  WHERE brand = 'b'

  AND model = 'm'

  AND cost = 1000;
- To clear all data (delete all Partitions) from a Table – but spare the Table:
- TRUNCATE cars_by_cost;

CQL: Time-To-Live (TTL) (I)

AGILE ACTORS

- Used in INSERT and UPDATE commands
- Column values in commands with TTL are automatically marked as deleted after the specified amount of time has expired
- Any subsequent update of the column resets the TTL to the new value specified in the update

CQL: Time-To-Live (TTL) (II)

- Expressed in seconds

- INSERT INTO

  cars_by_cost (brand, model, cost, merchant)

  VALUES ('volvo', 'xc90', 9999, 'daves')

  USING TTL 86400;

ACID & TRANSACTIONS

ACID

- Not in the usual RDBMS sense
- Atomicity
    - Per Partition
- Consistency
    - Configurable via CL
- Isolation
    - Per Partition
- Durability
    - Write operations are indeed persisted

Lightweight Transactions

AGILE ACTORS

- Two ways to accomplish, as
  - Compare-And-Set (CAS) operations
  - Batch Statements

- Both affect performance
  - Decrement

Compare-And-Set Ops. (I)

AGILE ACTORS

- It performs a Read operation, checks a Condition, and if that one holds, proceeds with the Write operation

- All atomically

Compare-And-Set Ops. (II)

- INSERT INTO
  cars_by_cost (brand, model, cost, merchant)
  VALUES ('volvo', 'xc90', 9999, 'daves')
  IF NOT EXISTS;
- UPDATE cars_by_cost
  SET merchant = 'pauls'
  WHERE brand = 'volvo'
  AND model = 'xc90'
  AND cost = 9999
  IF EXISTS;

Compare-And-Set Ops. (III)

- UPDATE accounts

  SET balance = 2000.0

  WHERE id = 1

  IF balance = 1000.0;

Batch Statements (I)

AGILE ACTORS

- BATCH statement
- Offers Atomicity for a series of operations
  - Write-Operations
    - INSERT, UPDATE, DELETE
  - All these operations receive the same timestamp
  - Order of operations is NOT guaranteed
- Does NOT offer isolation
  - Other statements can read/write data affected by the batch

Batch Statements (II)

AGILE ACTORS

- CREATE TABLE accounts1 (
  id INT, balance **DECIMAL**,
  PRIMARY KEY ((id)));

- **BEGIN BATCH**
  UPDATE accounts1
  SET balance = 1500.0 WHERE id = 1;
  UPDATE accounts1
  SET balance = 2500.0 WHERE id = 3;
  APPLY BATCH;

Batch Statements (III)

AGILE ACTORS

- CREATE TABLE accounts2 (
  id INT, balance **COUNTER**,
  PRIMARY KEY ((id)));

- **BEGIN COUNTER BATCH**
  UPDATE accounts2
  SET balance = balance + 500 WHERE id = 1;
  UPDATE accounts2
  SET balance = balance - 500 WHERE id = 3;
  APPLY BATCH;

AGILE ACTORS

DATA MODELING FOR CASSANDRA DB

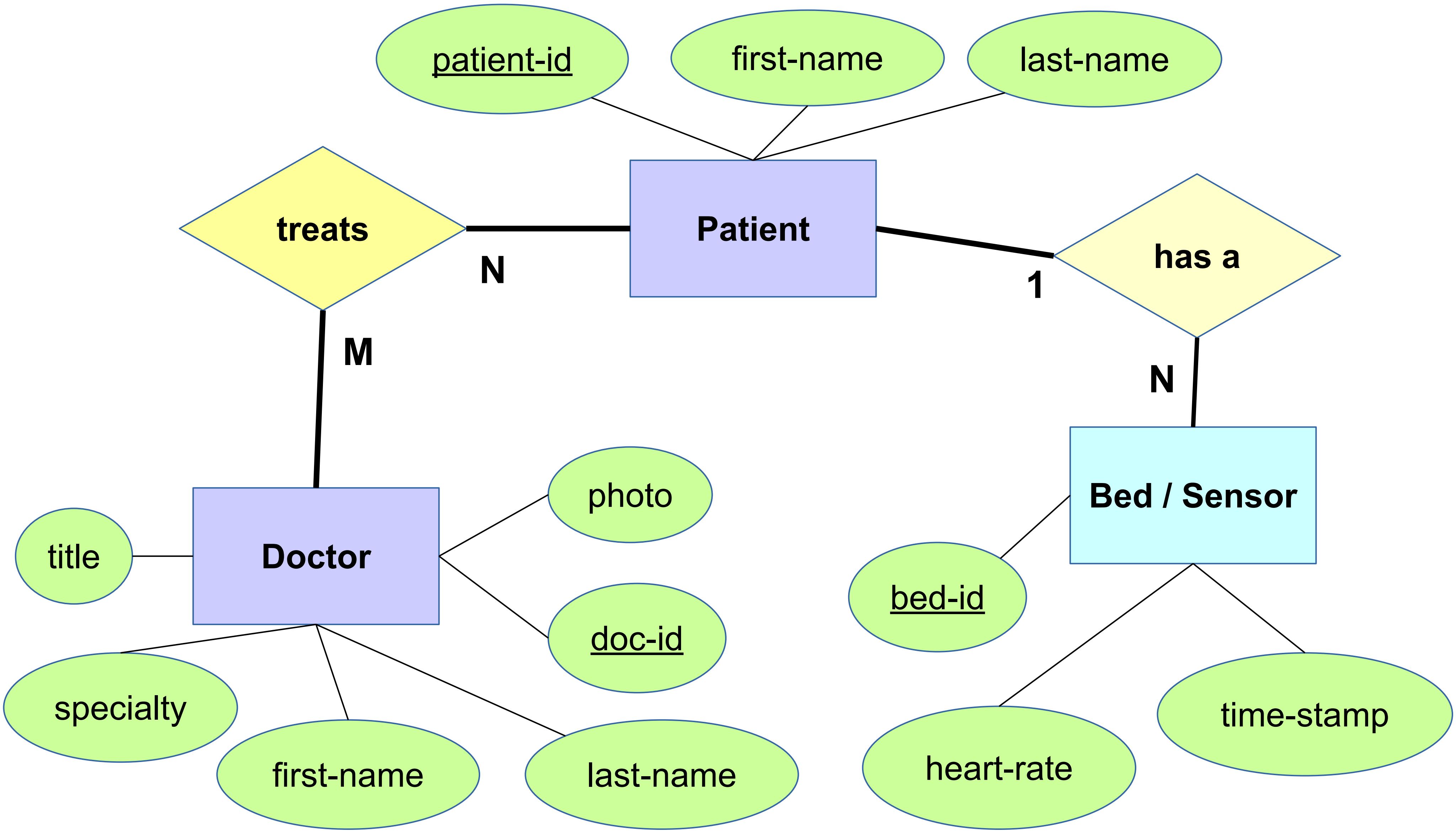
Data Modeling (DM)

AGILE ACTORS

1. Conceptual Data Model
2. Query-Driven Schema Design
    - Access Patterns
3. Logical Data Model
4. Analysis for Partition Size and Data Duplication
5. Physical Data Model
    - CQL

- Toy Application related to heart rate measurements of patients in a hospital
- We start with a Conceptual Model and some Queries in the form of Requirements

DM: Conceptual Data Model



DM: Queries – Reqs. (I)

1) Retrieve all information for a Doctor given his/her full name

2) Retrieve all information excluding the photograph (picture) for a Doctor given his/her full name

3) Retrieve the names and ids of the Doctors treating a given Patient, who is known by his/her patient-id. Have them ordered alphabetically.

DM: Queries – Reqs. (II)

4) Find the average heart rate for a given Patient – known by his/her patient-id – on a single given date and given time range.

5) Find the average heart rate for a given Patient – known by his/her patient-id – on a given date range.
Assume that the date range will have at most ten days.

DM: Queries – set 1

1) Retrieve all information for a Doctor given his/her full name

2) Retrieve all information excluding the photograph (picture) for a Doctor given his/her full name

DM: Logical Data Model (set 1)

AGILE ACTORS

- Q1, Q2:
    - last_name TEXT → Partition Key column (*)
    - first_name TEXT → Partition Key column (*)
    - doc_id INT → Clustering column (**)
    - specialty TEXT
    - title TEXT
    - photo BLOB

- (*) We search by it

- (**) Needed for uniqueness across a row - case of two different doctors with same names

- Q1, Q2:
  - A photo is only needed for Q1 but can slow down Q2
  - We can duplicate data to make Q2 faster
- Q1 =
  [last_name **K**, first_name **K**, doc_id **C**↑,
  specialty, title, **photo**]
- Q2 =
  [last_name **K**, first_name **K**, doc_id **C**↑,
  specialty, title]

3) Retrieve the names and ids of the Doctors treating a given Patient, who is known by his/her patient-id. Have them ordered alphabetically.

DM: Logical Data Model (set 2)

- Q3:
  - patient_id TEXT → Partition Key column (*)
  - doc_last_name TEXT → Clustering column (**)
  - doc_first_name TEXT → Clustering column (**)
  - doc_id INT → Clustering column (***)

- (*) We search by it
- (**) We order the result by them
- (***) Needed for uniqueness across a row - Corner case of two different doctors with same names treating one patient

DM: Analysis (set 2)

AGILE ACTORS

- Q3: No surprises
- Q3 =
  [patient_id **K** ,
  doc_last_name **C**↑,
  doc_first_name **C**↑,
  doc_id **C**↑]

4) Find the average heart rate for a given Patient – known by his/her patient-id – on a single given date and given time range

5) Find the average heart rate for a given Patient – known by his/her patient-id – on a given date range. Assume that the date range will have at most ten days.

DM: Logical Data Model (set 2)

AGILE ACTORS

- Q4, Q5:
  - patient_id TEXT → Partition Key column (*)
  - patient_last_name TEXT
  - patient_first_name TEXT
  - bed_id TEXT
  - when TIMESTAMP → Clustering column (**)
  - heart_rate INT

- (*) We search by it
- (**) We perform range-search by it

DM: Analysis (set 2)

- Q4, Q5:
  - We have at most 1 measurement per minute, that is 1440 per day, 14 400 for ten days
  - The range is defined by the timestamp is quite large
  - Although we keep a full timestamp, we only query for days or hours
  - Information about the patient's names is repeated

DM: Analysis (set 2)

AGILE ACTORS

- Q4, Q5 =

- [patient_id K,
 patient_last_name S,
 patient_first_name S,
 bed_id,
 when_date TIMESTAMP C↓,
 when_day_minutes INT C↓,
 heart_rate]

- This is good enough for bounded Partition size

DM: Physical and CQL

AGILE ACTORS

- Q1) TABLE docs_w_photos_by_name
- Q2) TABLE docs_by_name
- Q3) TABLE docs_by_patient
- Q4, Q5) TABLE heart_rate_by_patient_and_time

CQL source code about them in the accompanying toy application

FINAL WORDS

Final Words (I)

AGILE ACTORS

- Choose between Apache Cassandra and DataStax Enterprise
- Consult the CQL help pages when in doubt
- Consult your driver's manual pages when in doubt
    - For example, for the accompanying Java/Spring application:

      http://docs.datastax.com/en/developer/java-driver/3.1/manual/
- Prototype your Table Schemas using CQLSH or DevCenter
- Configure Availability and Consistency wisely
- Keep Scalability in mind; Do Data Dimensioning

Final Words (II)

AGILE ACTORS

- Choose drivers, libraries and frameworks that will improve productivity and testability regarding development
- Load test your Application early enough
  - Fine-tune Cassandra or the Application if needed
- Consider Data Migration early enough
- If you have any doubts about the benefits of using Cassandra, proceed to develop an Application Prototype and experiment with it

Accompanying Toy Applications

- Both Toy Applications are about the ‘hospital’ example
- First application is a Java RESTful Web Service that make direct use of the Cassandra Driver
  - https://github.com/pek-github/cassandra-slides
  - Refer to file README.md there for more information
- Second application is a RESTful Web Service that makes use of Spring Boot and Spring Data Cassandra
  - https://github.com/pek-github/SpringCassandra
  - Refer to file README.md there for more information

MORE ABOUT CASSANDRA DB

Get Started (I)

AGILE ACTORS

- Apache C* - http://cassandra.apache.org/
- DataStax C* - http://www.datastax.com/
- DataStax C* Drivers and Tools

  https://academy.datastax.com/downloads/welcome
- DataStax and Apache C* Drivers -
  - DataStax
  - Apache

Get Started (II)

AGILE ACTORS

- DataStax and Apache CQL Documentation
  - http://docs.datastax.com/en/cql/3.3/cql/cqlIntro.html
  - http://cassandra.apache.org/doc/latest/cql/

- **DataStax Startup Program**

  http://www.datastax.com/datastax-enterprise-for-startups

Learn More

AGILE ACTORS

- DataStax Cassandra Academy
  - https://academy.datastax.com/
- Stackoverflow Tags:
  - cassandra, datastax, datastax-enterprise
- Books
  - visit Amazon

Get Involved

- Meetup Groups

  http://www.meetup.com/

- Join a DataStax Group:

  https://www.meetup.com/Athens-Cassandra-Users/

References – Sources (I)

AGILE ACTORS

These (already mentioned) web pages:

- http://cassandra.apache.org/
- http://www.datastax.com/
- http://docs.datastax.com/en/cql/3.3/cql/cqlIntro.html
- http://cassandra.apache.org/doc/latest/cql/
- https://academy.datastax.com/

References – Sources (II)

AGILE ACTORS

- These additional articles and web pages:
    - Leslie Lamport, "Time, clocks, and the ordering of events in a distributed system", http://research.microsoft.com/en-us/um/people/lamport/pubs/time-clocks.pdf
    - Mark Burgess, "Deconstructing the 'CAP theorem' for CM and DevOps", http://markburgess.org/blog_cap.html

References - Acknowledgments

Tables and Graphics in slides marked with C*
are copyrighted work of DATASTAX
and used here after their permission.
We really thank them for that.

Let's Talk!

AGILE ACTORS

THANK YOU !