LAMBDA ARCHITECTURE

Speed layer: Real-time views

BIS2013

Database Systems

Prof. Duc Khanh Tran, PhD

1st Hai Vo

2nd Huy Huynh

3rd Tin Ho

Agenda

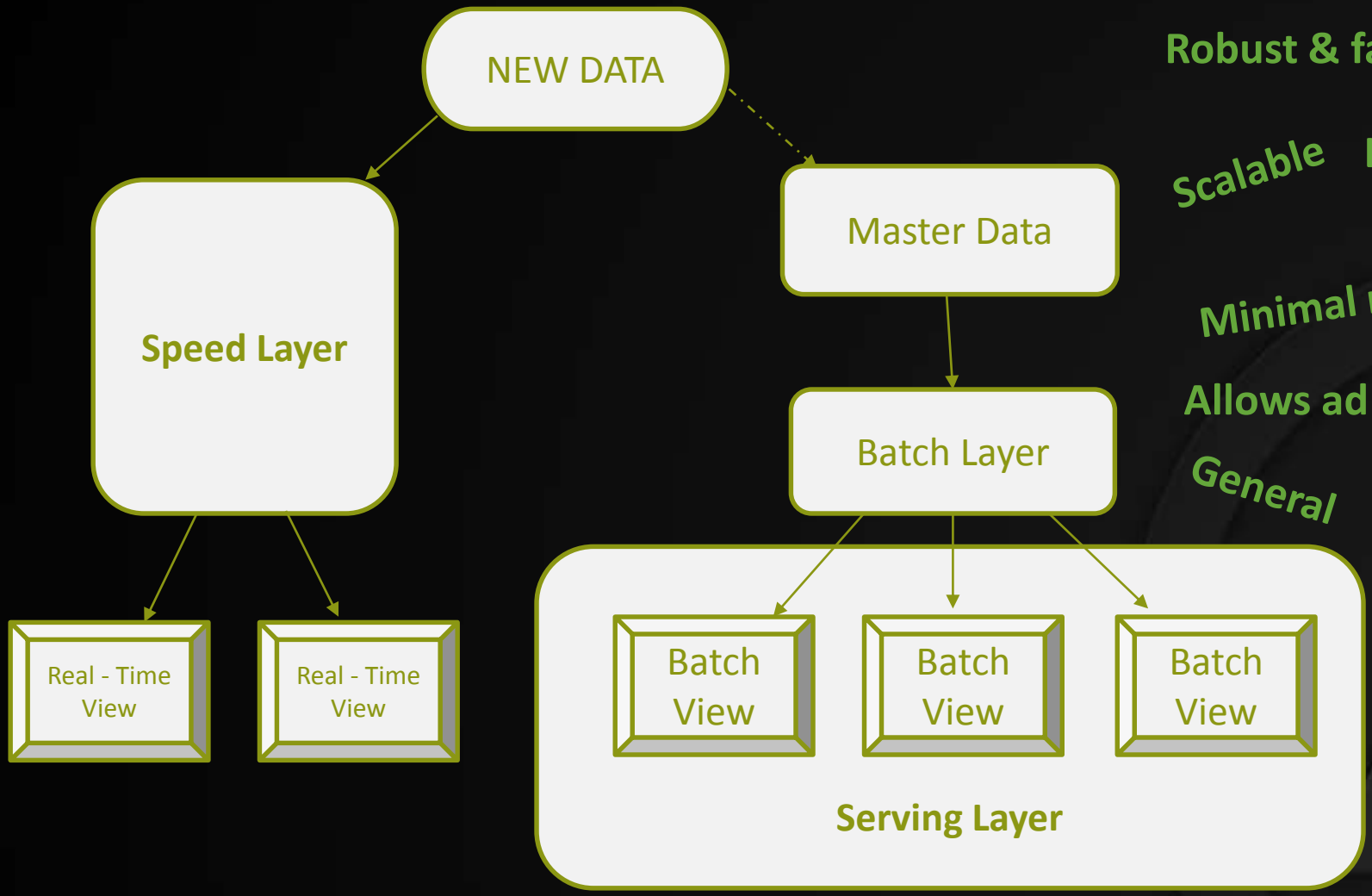
Hai Vo	The Speed Layer
	Computing & Storing real-time views
Huy Huynh	Challenges of incremental algorithms
	Asynchronous versus synchronous updates
Tin Ho	Expiring real-time views
	Cassandra: an example speed layer view
	Conclusion

Motivation

The update latency requirements vary a great deal between applications.

Some applications require updates to **propagate immediately**, while in other applications a latency of a few hours is fine.

Motivation



Robust & fault-tolerant

Scalable

Low latency reads

Minimal maintenance

Allows ad hoc queries

General

Extensible

Debuggable

Speed Layer vs Batch Layer

The speed layer is similar to the batch layer in that it produces views based on data it receives.

Incremental updates as opposed to **recomputation updates**.

Only produces views on **recent data** vs produces views on the **entire dataset**.

Speed Layer

Processing data on a **smaller scale**

Requires databases that support random reads and **random writes**

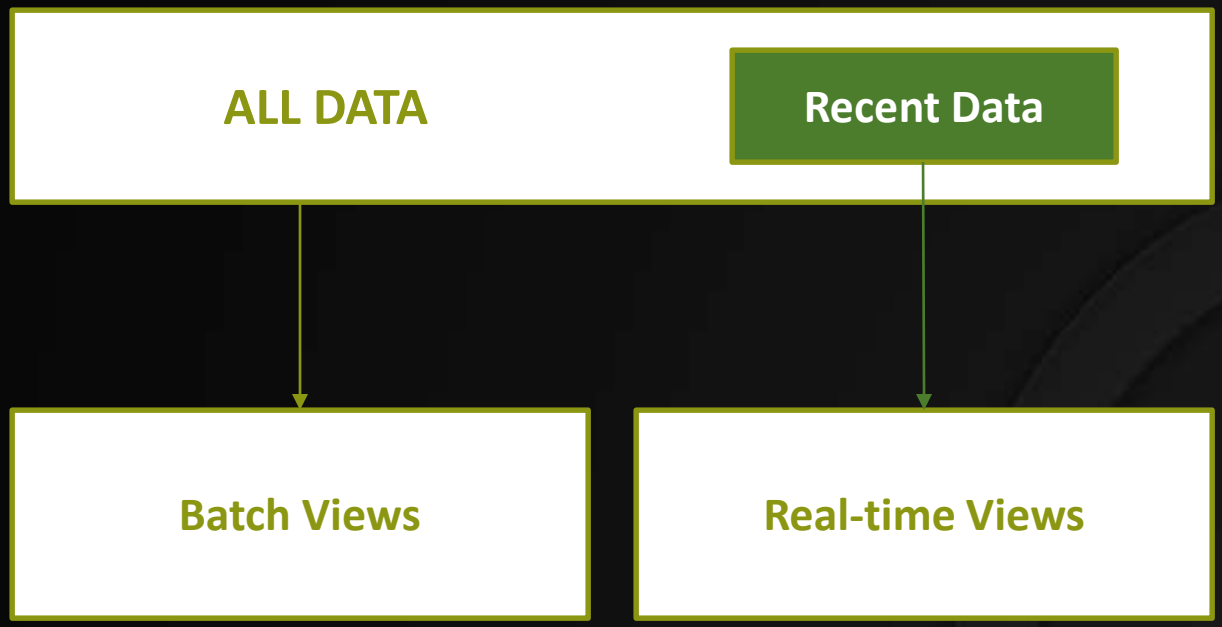
More **complex** and thus more **prone to error.**

The speed layer views are **transient**.

Two major facets of the speed layer

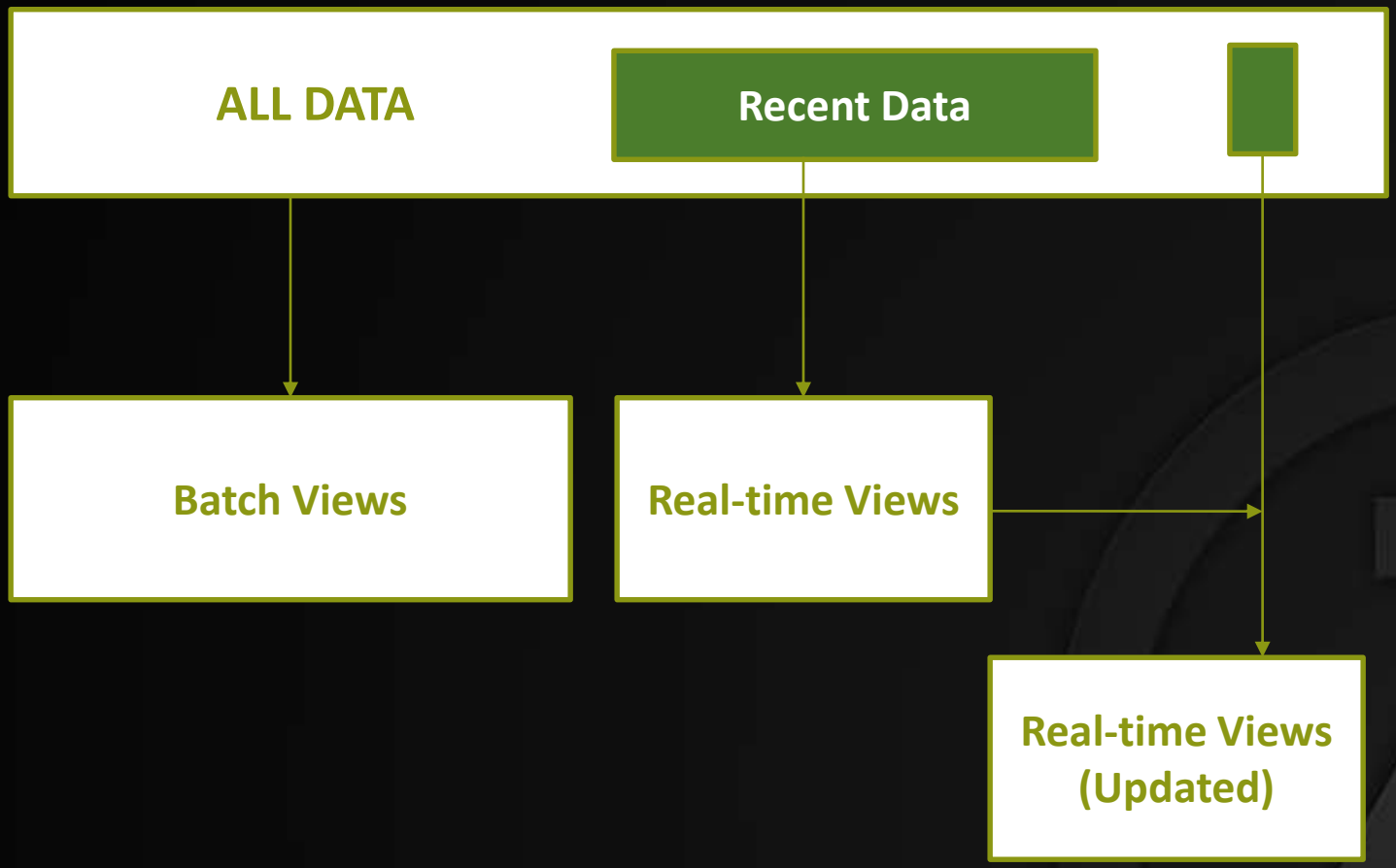
Storing the real-time views and **processing** the incoming data stream so as to update those views

Computing Real-time Views



Real-time View = function (recent data)

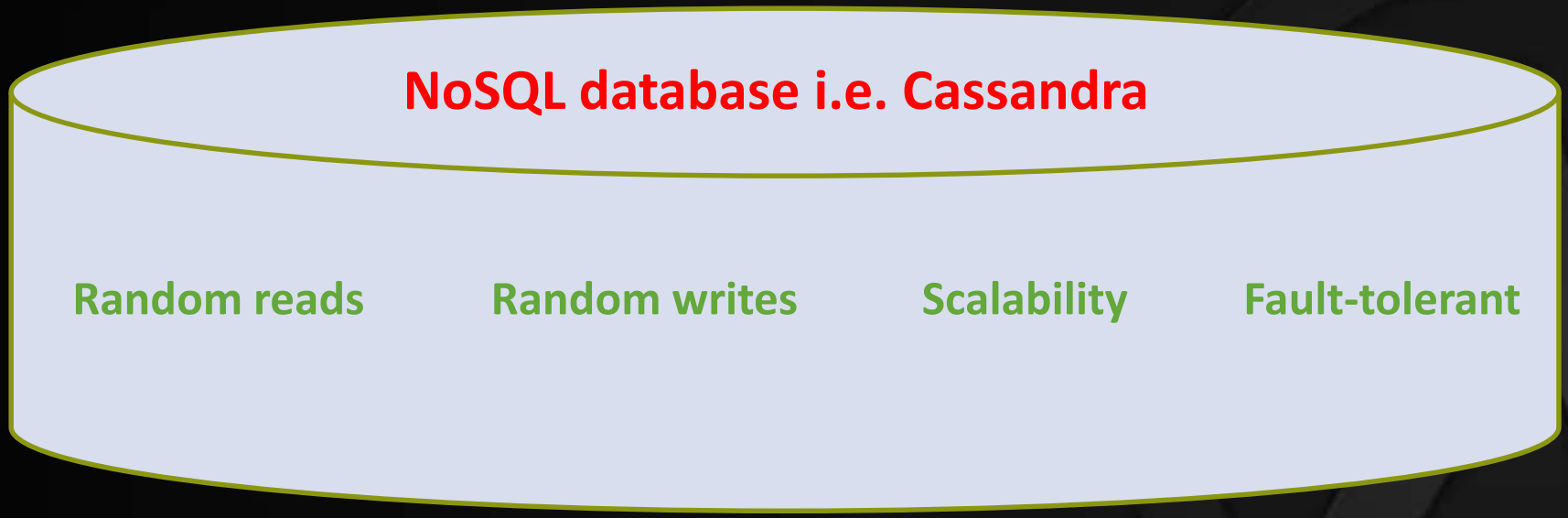
Computing Real-time Views



Real-time View = function (new data, previous real-time view)

Storing Real-time Views

Low-latency random reads, and using incremental algorithms necessitates low-latency random updates



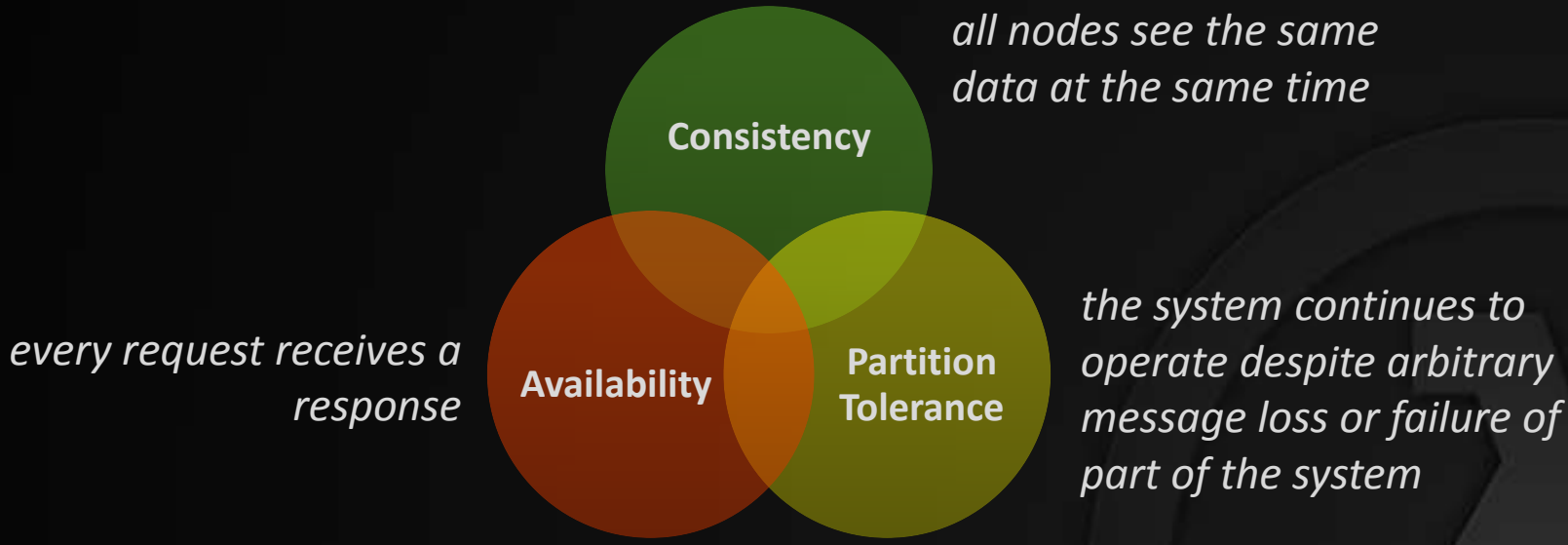
Challenges of incremental computation

Incremental algorithms are less general and less human fault-tolerant, but higher performance

Challenge in a real-time context: the interaction between *incremental algorithms* and the CAP theorem

The CAP Theorem

"You can have at most two of Consistency, Availability, and Partition-tolerance"



"When a distributed data system is **Partitioned**, it can be **Consistent** or **Available** but **not both**"

C vs A

When you choose consistency, sometimes a query will receive an error instead of an answer

When you choose availability, at best you will have where **eventual consistency**

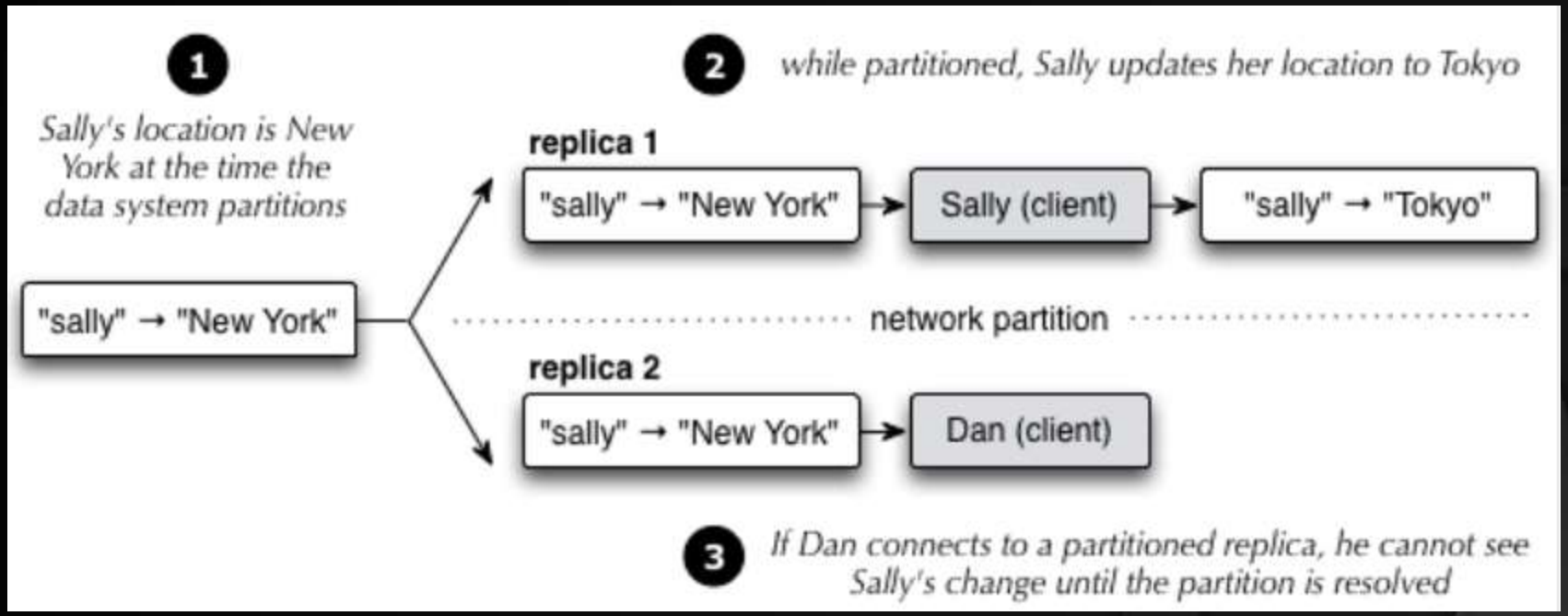
Replication in Distributed system

Sally	New York
Joe	Paris
Maria	Tokyo

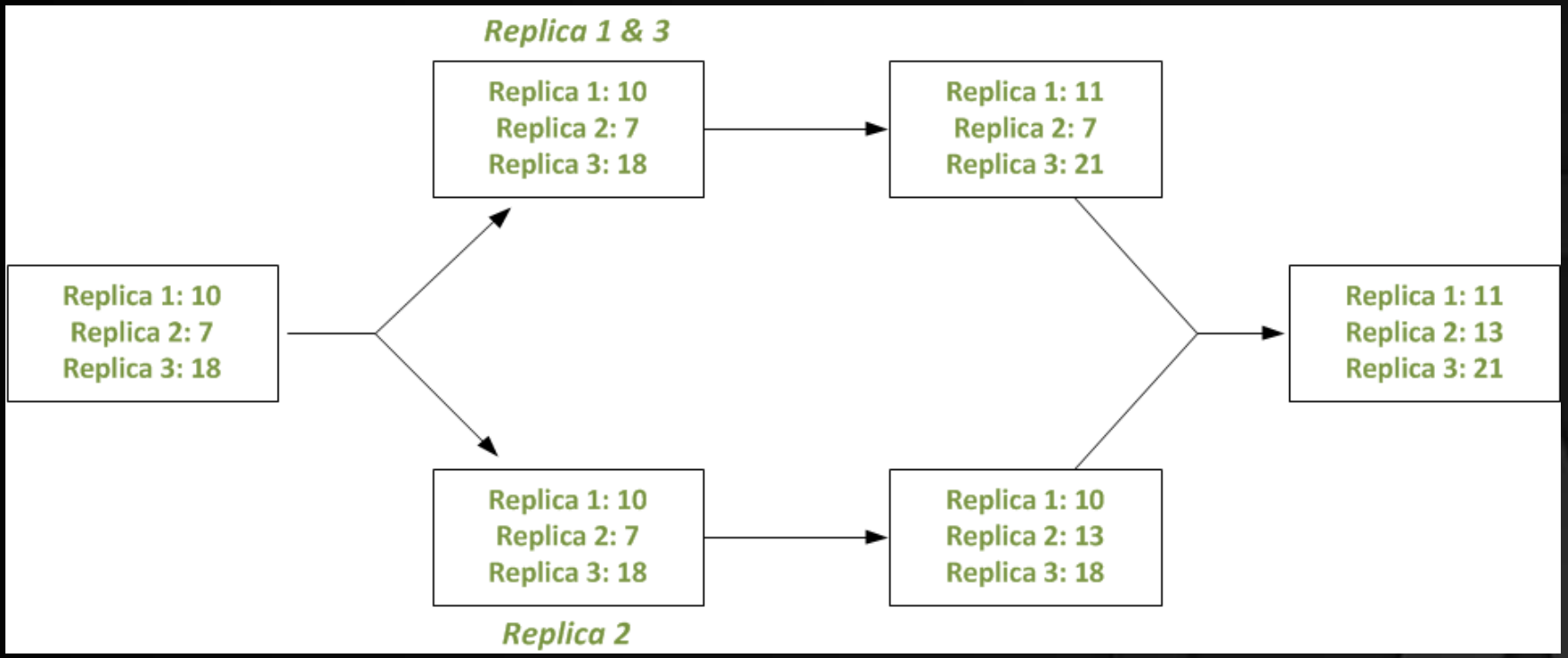
Sally	New York
Maria	Tokyo

Sally	New York
Maria	Tokyo

Eventual consistency



Eventual consistency counting



Interaction between the CAP theorem and incremental algorithms

Complexity in a real-time eventually consistent context

Read and repair algorithms are needed

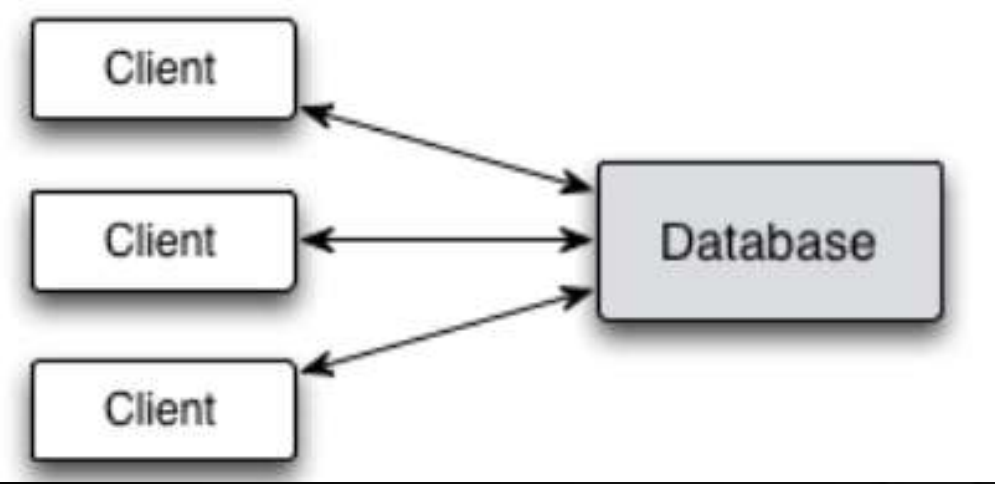
Keep calm with Lambda

Unfortunately there is no escape from this complexity if you want eventual consistency in the speed layer

If the real-time view becomes corrupted, the batch/serving layers will later automatically correct the mistake in the serving layer views

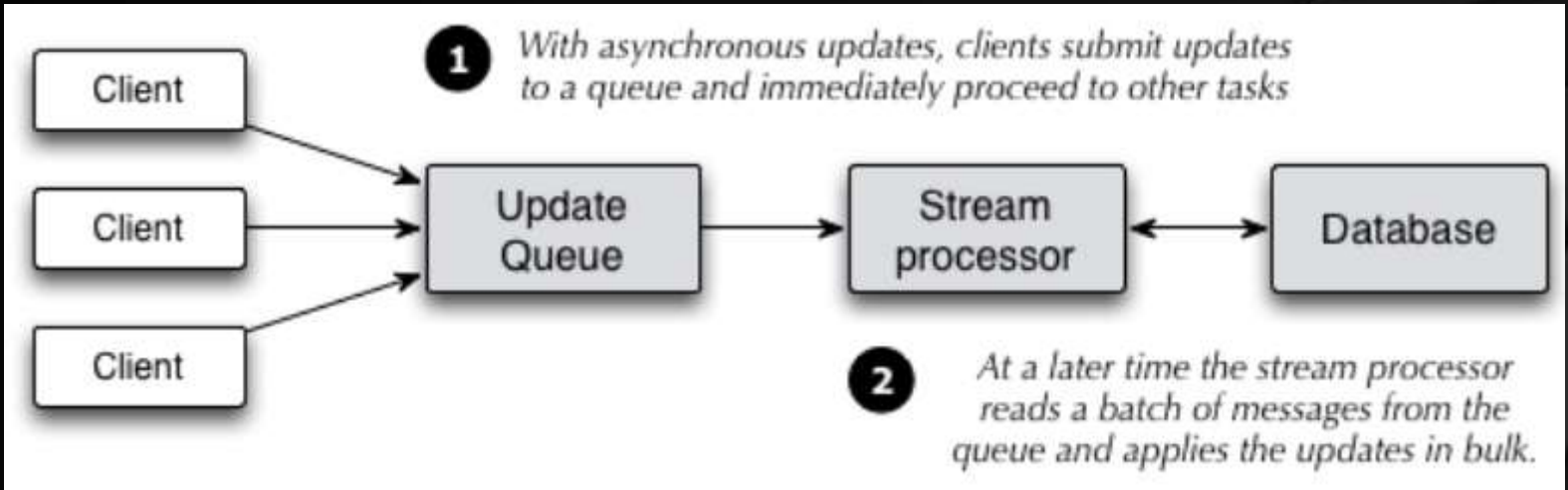
Synchronous updates

The application issues a request directly to the database and blocks until the update is processed



Asynchronous updates

Asynchronous update requests are placed in a queue with the updates occurring at a later time



Asynchronous versus synchronous updates

Synchronous	Asynchronous
Fast	Slower
Coordinate the update with other aspects of the application	Not coordinate
May overload the database	Not overload the database

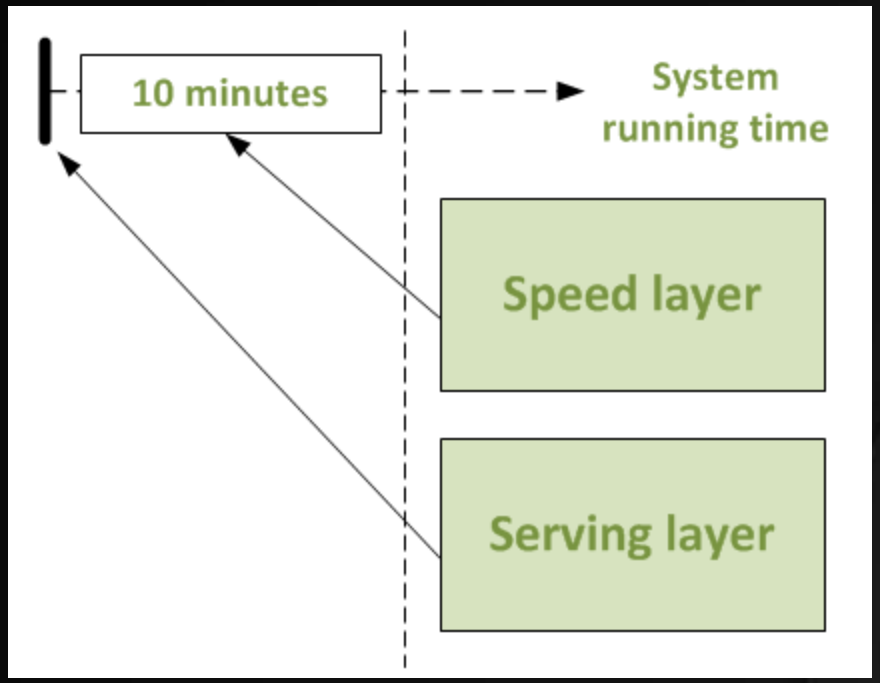
Expiring real-time views

Since the simpler batch and serving layers continuously override the speed layer, the speed layer views only need to represent data yet to be processed by the batch computation workflow

Expiring real-time views

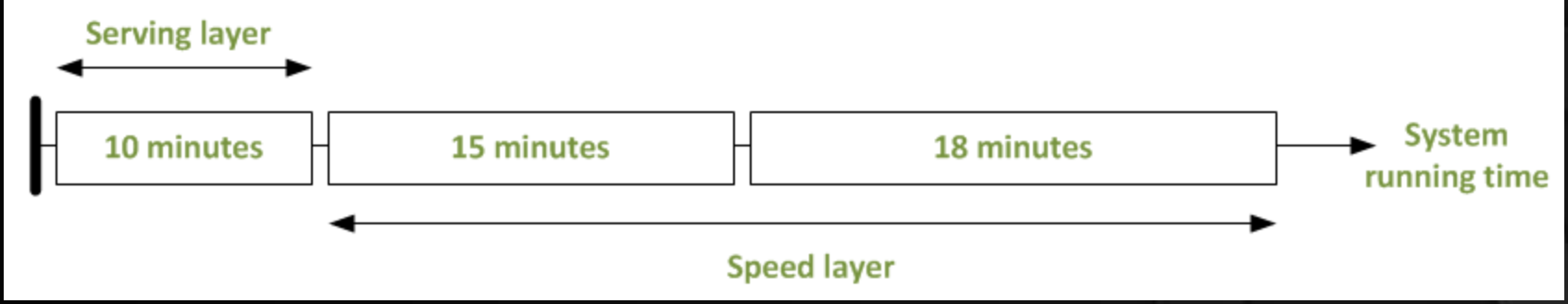
Instead, we present a generic approach for expiring speed layer views that works regardless of the speed layer databases being used. To understand this approach, let's first get an understanding of what exactly needs to be expired each time the serving layer is updated

Expiring real-time views

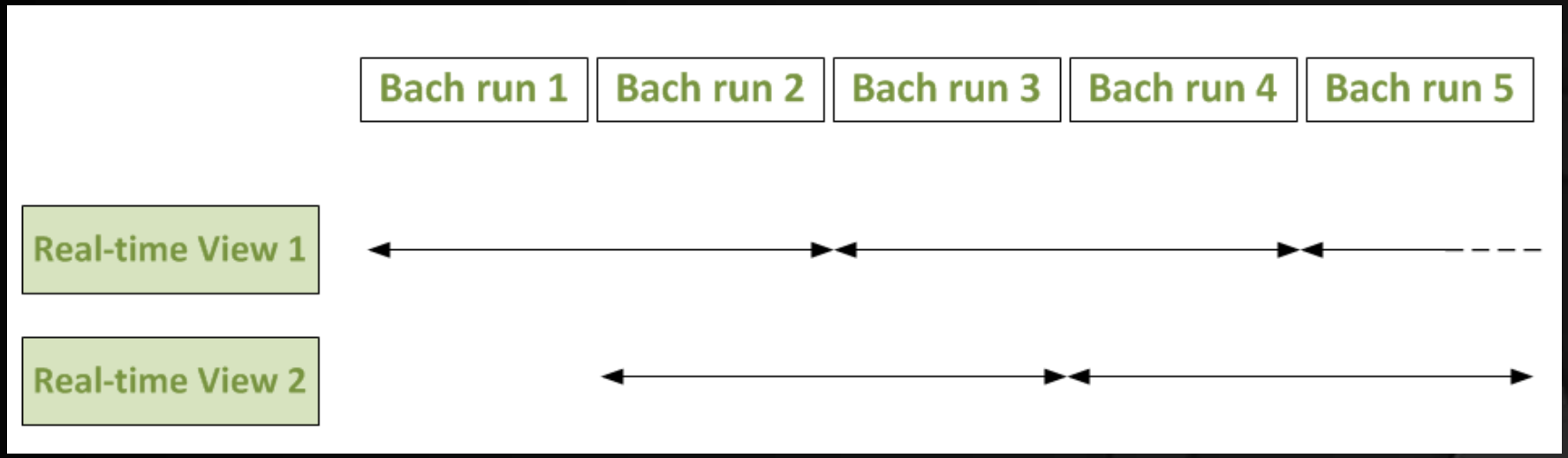


Expiring real-time views

Expiring real-time views



Expiring real-time views



Cassandra's data model

What do you have ever heard about it?

- Column oriented/family database
- Key value based database
- NOSql database
- Real time operational data store
- Distributed database
- ...

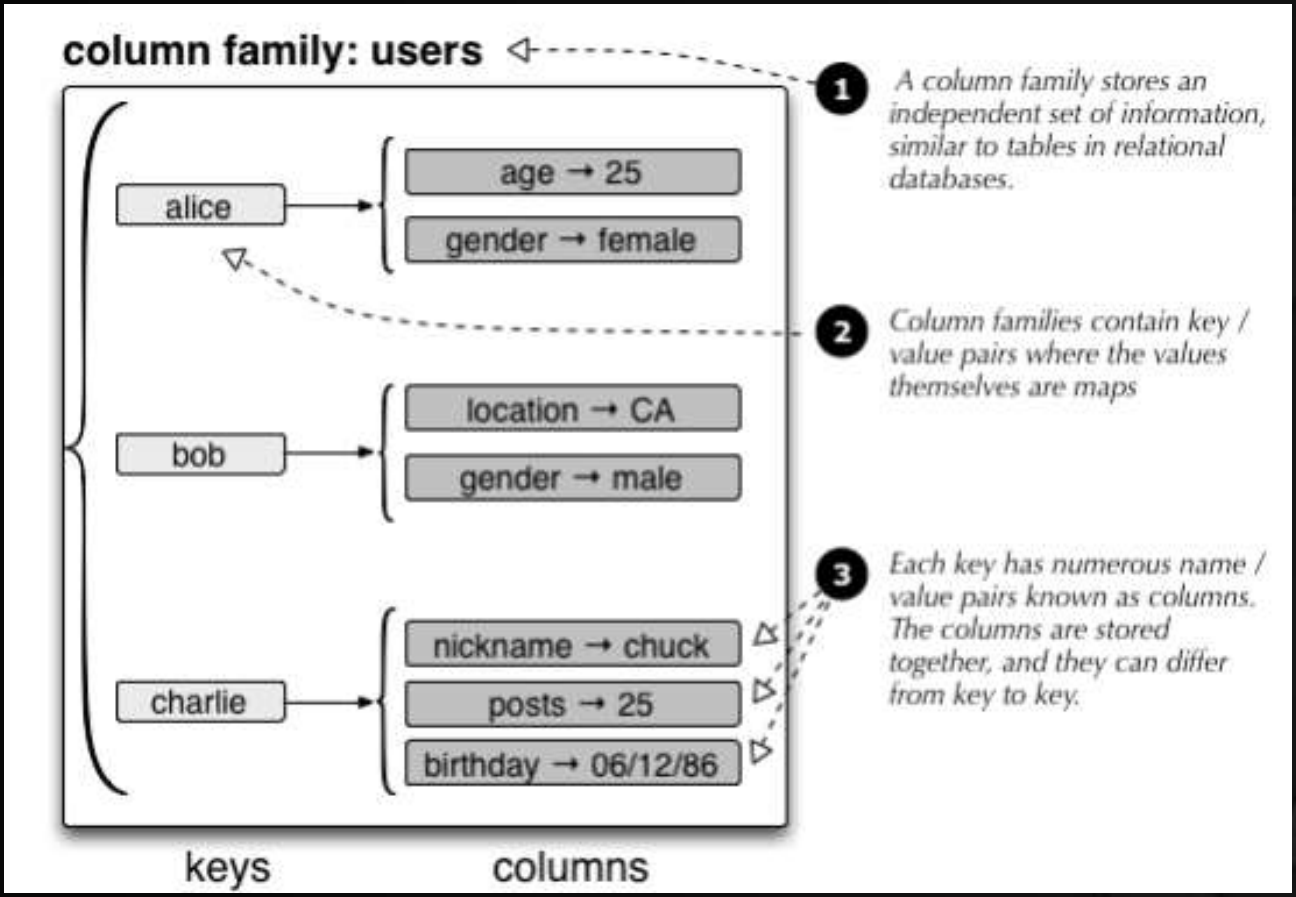
Cassandra's data model

Some definitions:

- Key space: is the outermost container for data in Cassandra.
- Column families: is a container for a collection of rows. Each row contains ordered columns.
- Columns: is the most basic unit of data in Cassandra data model. Consists of <key,value,timestamp> triplet.

Cassandra's data model

Column family example:



Cassandra's data model

Cassandra features make it suitable for real-time view:

- Partitions keys among nodes in cluster:
  - RandomPatitioner and OrderPreservingPatitioner
- Composite columns.

Using Cassandra!

Conclusion

Theoretical model of the speed layer

CAP theorem

Incremental computation in stead of batch computation

Store speed layer view (real-time)

References

Big Data, Principles and best practices of scalable real-time data system – Nathan Marz