Big Data Processing and Applications – Lecture 7

Ioana Ciuciu
ioana.ciuciu@ubbcluj.ro

Course structure

Date	Course/Week	Title
03.10.2025	1	Introduction to Data Science and Big Data (Part 1)
10.10.2025	2	Introduction to Data Science and Big Data (Part 2)
17.10.2025	3	Industrial standards for data mining projects. Big data case studies from industry – invited lecture from Bosch
24.10.2025	4	Data systems and the lambda architecture for big data
31.10.2025	5	Lambda architecture: batch layer
07.11.2025	6	Lambda architecture: serving layer
14.11.2025	7	Lambda architecture: speed layer
	8	NoSQL Solutions for Big Data – invited lecturer from UBB
	9	Data Ingestion
	10	Introduction to SPARK – invited lecture from Bosch
	11	Data visualization
	12	Presentation research essays
	13	Presentation research essays + Project Evaluation during Seminar
	14	Presentation research essays + Project Evaluation during Seminar

Slight modifications in the structure are possible

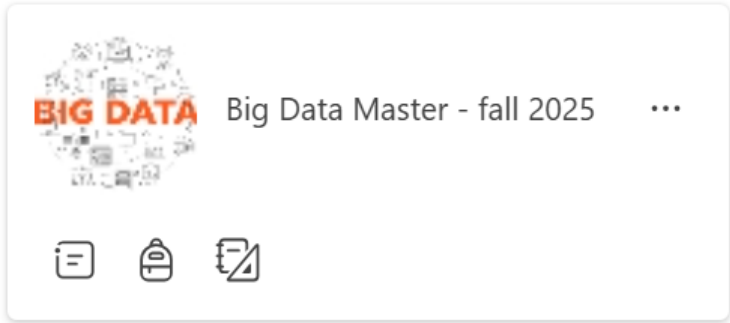
Semester Project

- Team-based (2-5 students with precise roles)
- Multidisciplinary: 3 CS Master Programes (HPC, SI, DS) + Master in Bioinformatics
- Real use cases: collaboration with local IT industry - TBA
- High degree of autonomy in selecting the topic and proposing the solution
- Implementation prototype
- Prototype demo & evaluation – student workshop (last seminar – weeks 13 & 14)
- *Best projects – disseminated in various events (TBD), scientifically disseminated (workshops/conferences/journals) AND/ OR possibility for a dissertation thesis*

Evaluation

- The final grade will be computed as follows:
  - 50% semester project (must be >= 5)
  - 50% research presentation or written exam (must be >= 5)

- Semester project
  - Details available on the course team
    - MS Team: **Big Data Master - fall 2025**
    - Access code: **j1exenq**



Agenda

Lambda architecture: Speed Layer (Chapter 12)

- Computing real time views
- Storing real time views
- Asynchronous vs synchronous updates
- Expiring real time views

Additional support material: **Nathan Marz, James Warren**, Big Data: *Principles and Best Practices of Scalable Real Time Data Systems*

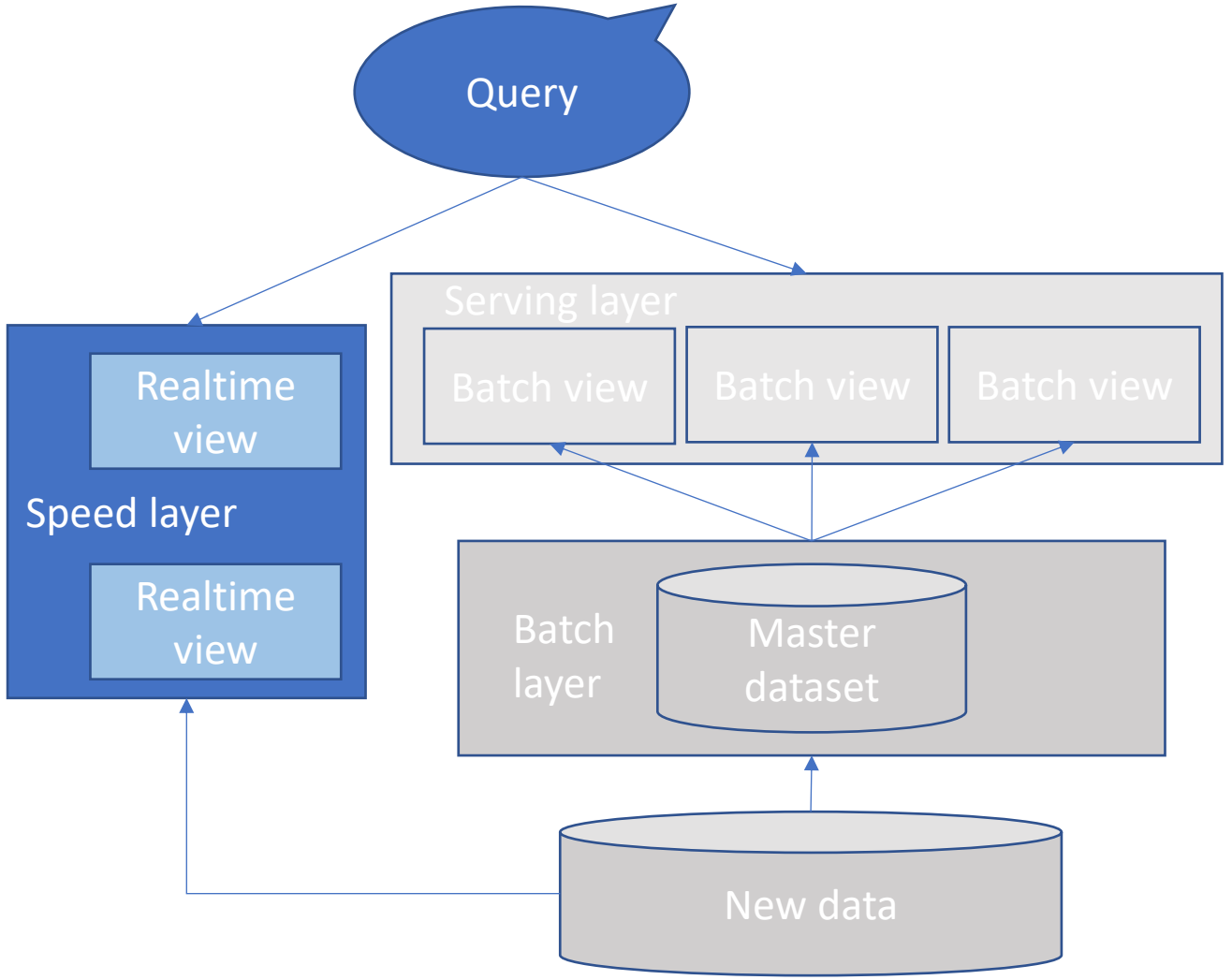
Lambda architecture: speed layer

Lambda architecture: serving layer

Speed layer: implements the equation below

Realtime view = function (new data)

Speed layer: its goal is to ensure new data is represented in the query functions as quickly as required by the application requirements.



Speed layer vs batch layer

Speed layer is similar to the batch layer in the sense they both produce views on the data they receive.

Speed layer	Batch layer
Updates are based on *incremental computations*	Updates are based on *batch computations (or recomputations)*.
Only produces views on recent data.	Produces views on the entire dataset.

Speed layer

Advantages:

- The speed layer is only responsible for data yet to be included in the serving layer views meaning at most a few hour old; it is vastly smaller than the master dataset
- The speed layer views are transient; once the data is absorbed into the serving layer, it can be discarded from the speed layer

Drawbacks:

- Requires databases that support **random reads and random writes**
- They are more complex and error prone; this is handled by the transient nature of the speed views

Two major topics to be discussed:

1. Storing the real-time views
2. Processing the incoming data stream so as to update the real-time views

Illustrating the limitation of batch layer for real time applications

Example: real time visualization of the S&P 500 index

The S&P 500 index, or simply the S&P, is a stock market index that measures the stock performance of 500 large companies listed on stock exchanges in the United States. It is one of the most commonly followed equity indices.

S&P 500 index = sum(P*i* x Q*i*)/Divisor

P*i* is the price of stock

Q*i* is the number of shared for each stock

Divisor is a value that is updated daily to keep the index consistent

Requirements:

- Real time updates
- Visualizations at different scales
- Customized visualizations (***e.g.*** opening price, closing price, mean price etc.)
- Customized history visualization

Illustrating the limitation of batch layer for real time applications

Illustrating the limitation of batch layer for real time applications

Example: real time visualization of the S&P 500 index

Implementation using the batch layer

Raw data

Raw data comes every second

```
datapoint
{
company_name
timestamp
stock_price
no_of_shares
}
```

Batch layer

Storage

Compute

S&P 500 index = sum(P*i* x Q*i*)/Divisor

Batch views

Second_view

Minute_view

Hourly_view

Daily_view

Montly_view

Yearly_view

Visualization

Illustrating the limitation of batch layer for real time applications

Example: real time visualization of the S&P 500 index

Implementation using the batch layer

Second_view requires to compute the S&P 500 function for 126,144,000 data points every 15 seconds

126,144,000 = 4 (per minute) x 60 (minutes) x 24 (hours) x 365 (days) x 60 (years)

Precomputing and indexing 126,144,000 values can be unfeasible and unpractical.

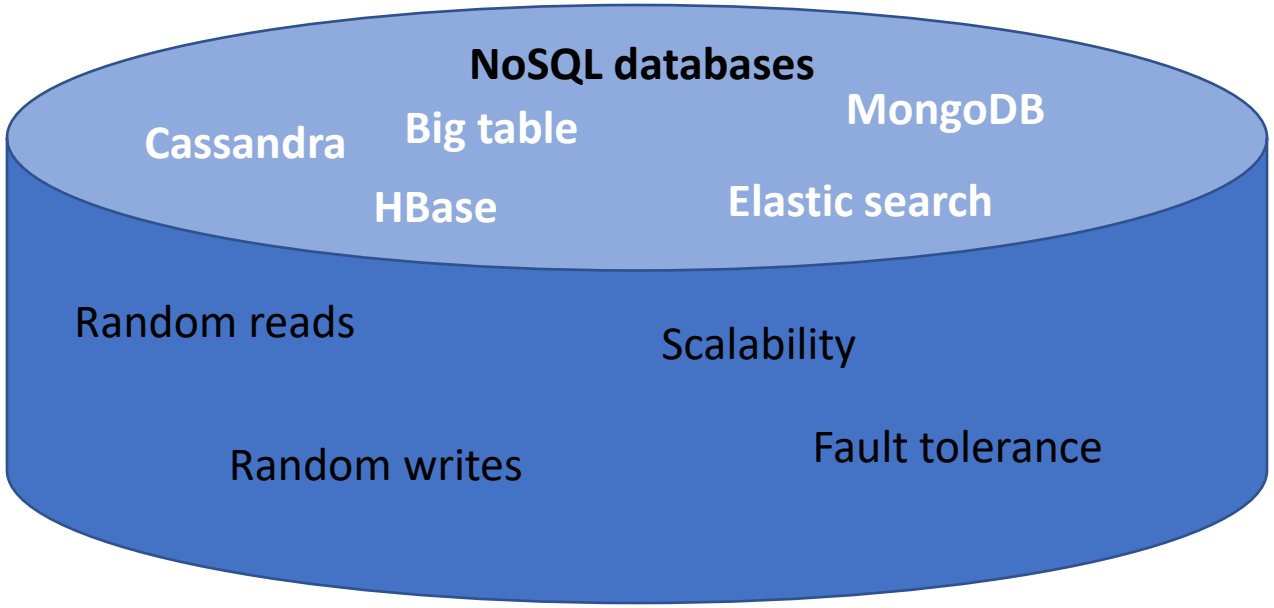
Storing real time views

Databases for the speed layer must meet the following requirements:

Requirements	Motivation
Random reads	Real-time views should support fast random reads to answer queries quickly. This means that data is contains must be **indexed**.
Random writes	To support incremental updates, it must be possible to **modify** real-time views with low latency.
Scalability	Real-time views should scale with the amount of data they store and the read/write rates required by the application. This implies that real-time views can be **distributed** across many machines.
Fault tolerance	If a disk or a machine crashes, the real-time view should continue to work normally. Fault tolerance is ensured by **replicating data** across machines.

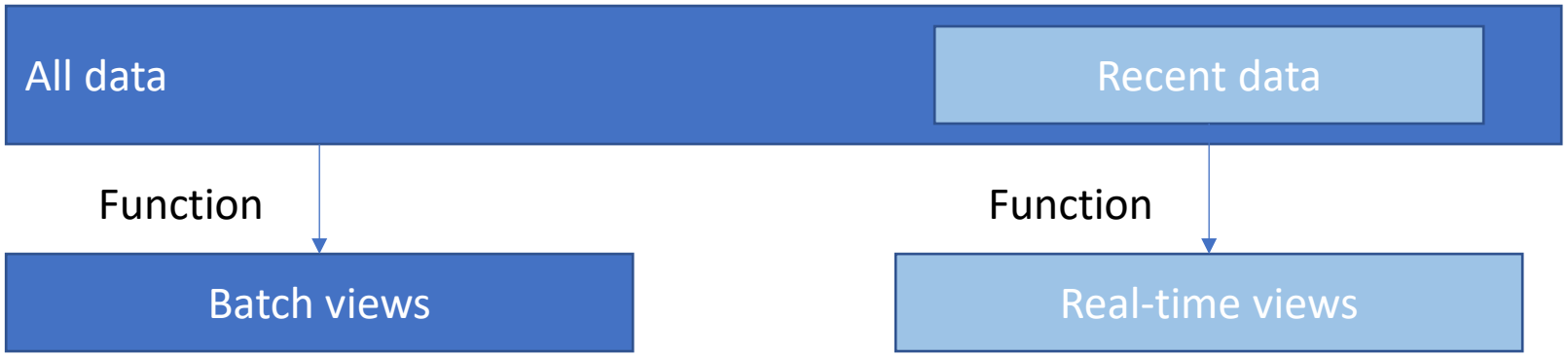
Storing real time views

Databases for the speed layer must meet the following requirements:



Computing real time views

Mirroring the batch/serving layer computation



This strategy is suited for applications that accept latency of the order of few minutes. Not suited for applications requiring latency of on the order of seconds/milliseconds.

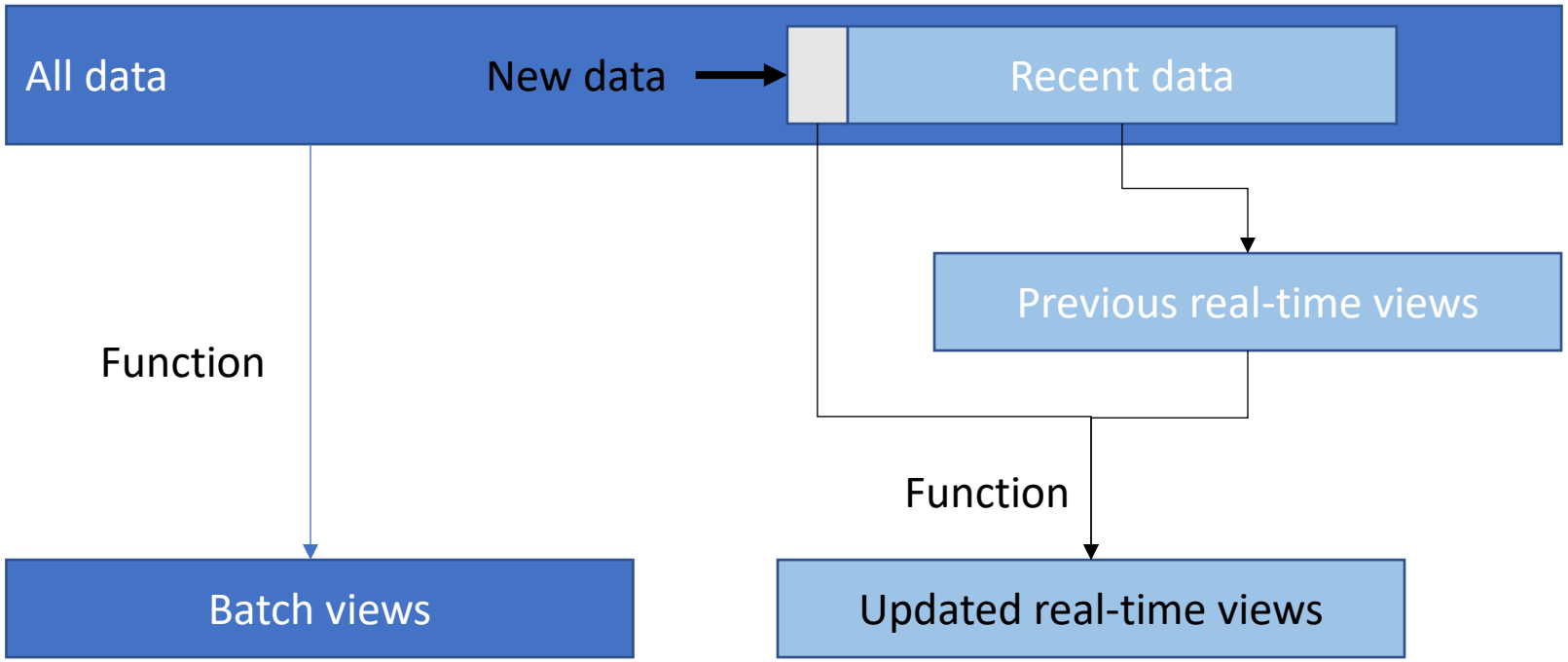
Example: application receives 32 GB new data per day.

Speed layer is responsible of processing data for at most 6 h ~ 8GB

Assuming 1 data unit ~ 100 bytes - > 86,000,000 data units to be processed - > high latency

Computing real time views

Incremental computation strategy



Main idea: update the real-time view as the data comes in, and reuse the work previously done to produce the views.

Computing real time views

Challenges of incremental computation strategy

Incremental algorithms vs recomputation algorithms

Recall

1. Incremental algorithms are less general ☹
2. Incremental algorithms are less fault tolerant ☹
3. Incremental algorithms provide with much higher performance ☺

Incremental algorithms main challenge:

- Related to the interaction of the incremental algorithms with ***CAP theorem***

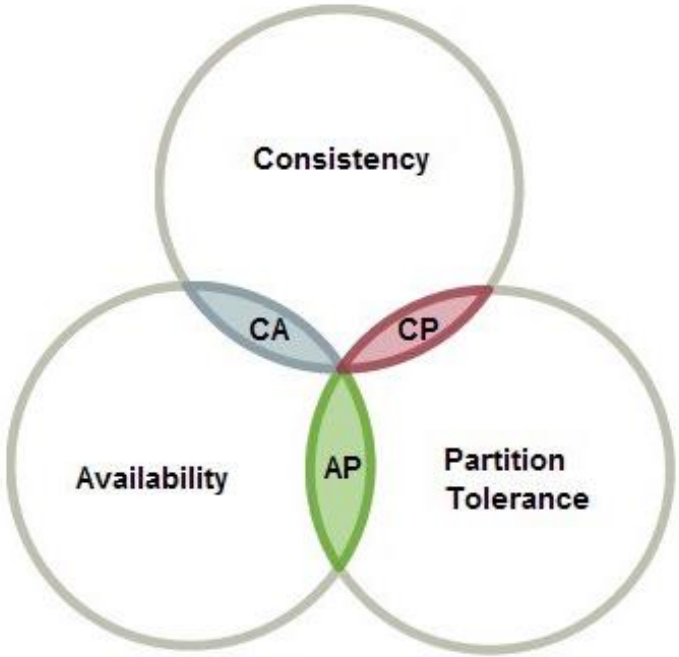
Computing real time views

Recall

CAP theorem:

- ***CAP theorem*** *is about the* ***trade-off between consistency*** *where reads are guaranteed to incorporate all previous writes, and* ***availability*** *where every query receives an answer instead of an error*
- *CAP theorem states that* ***"you can have at most two of consistency, availability and partition tolerance" - >*** *this could lead to confusing interpretations because a CA system is non-sense in distributed systems*

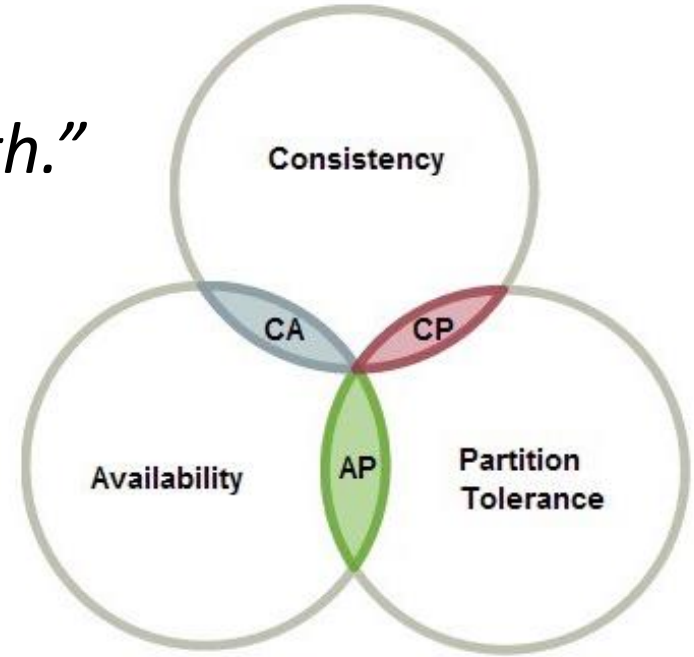
- *Actually the theorem is all about* ***what happens when the system is partitioned***



Computing real time views

CAP theorem in a different view:

"When a distributed system is partitioned,
*it can be **either consistent** or **available** but not both."*



- *When consistency is favored, sometimes a query will receive an error*
- *When availability is favored, reads may return results not up to date*

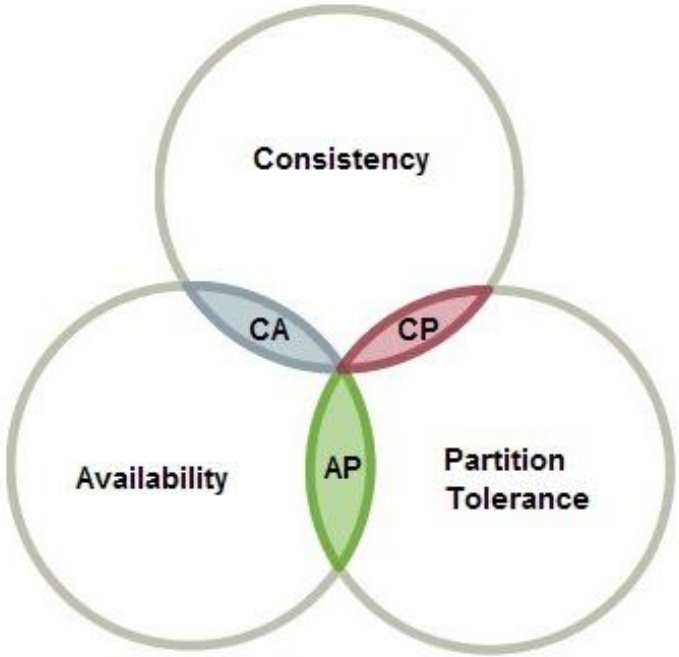
- *For highly available systems, the best that can be obtained in terms of consistency is **eventual consistency** -> the system returns consistency once the network partition ends*

Computing real time views

Understanding the validity of CAP theorem

Assume a distributed key/value database **with no replication ->** where each node in cluster is responsible for disjoint sets of keys

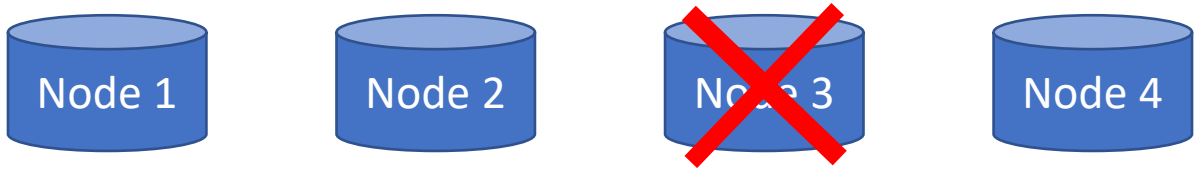
$$Key\_set1 \cap Key\_set2 \cap Key\_set3 \cap Key\_set4 = \emptyset$$



Computing real time views

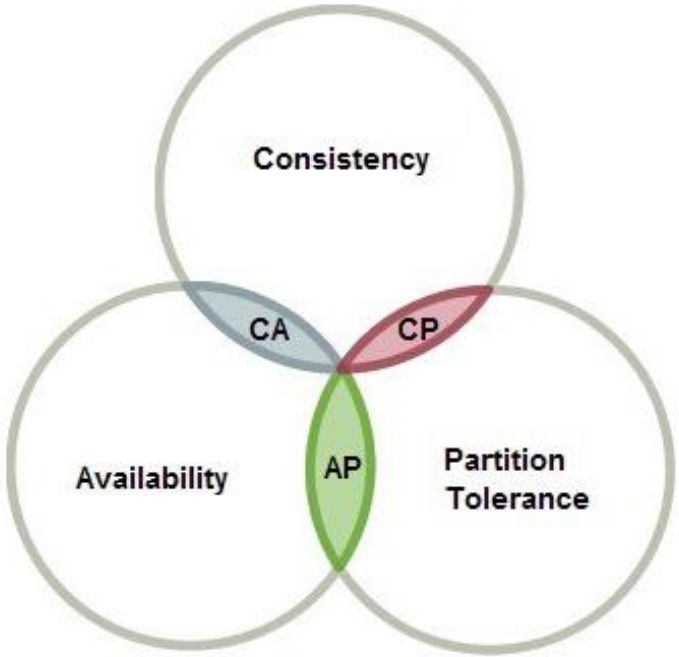
Understanding the validity of CAP theorem

Assume a distributed key/value database where each node in cluster is responsible for disjoint sets of keys **-> no replication**



$Key\_set1 \cap Key\_set2 \cap Key\_set3 \cap Key\_set4 = \emptyset$

If one machine becomes unavailable - > you cannot read/write data to that node

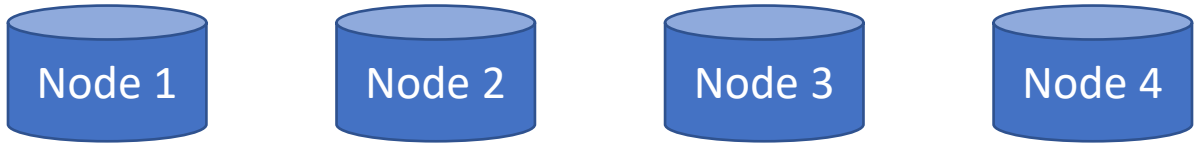


Solution: make the system fault tolerant by replicating data

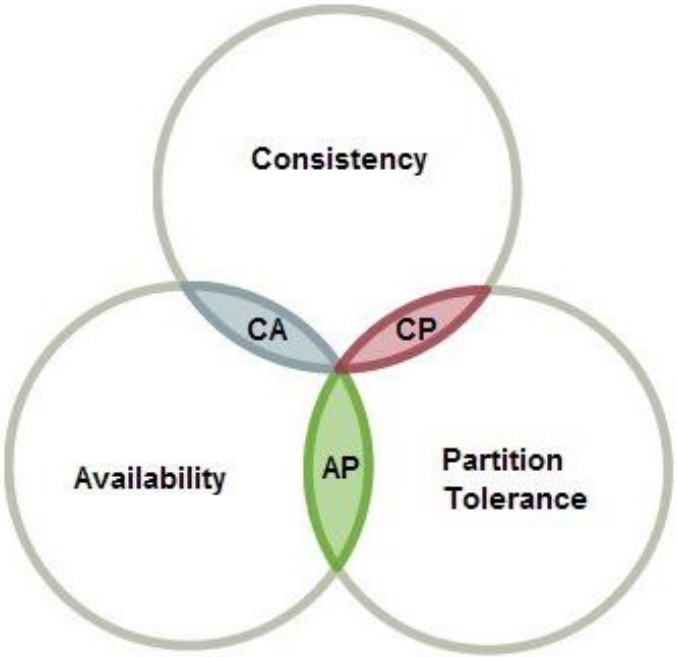
Computing real time views

Understanding the validity of CAP theorem

Assume a distributed key/value database with a replication factor of 3



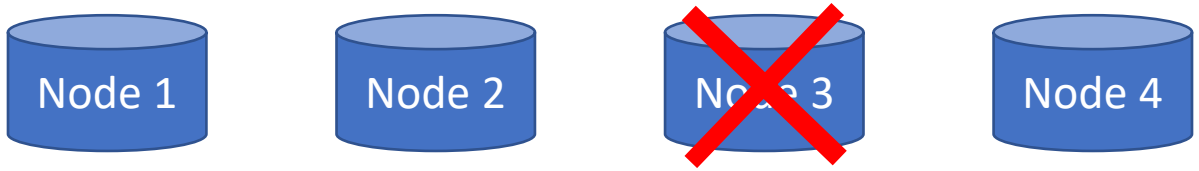
$$Key\_set1 \cap Key\_set2 \cap Key\_set3 \cap Key\_set4 \neq \emptyset$$



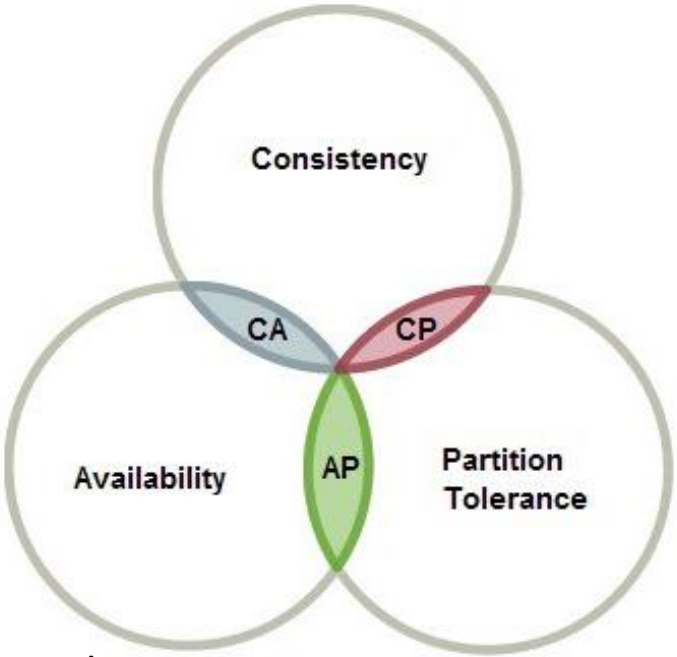
Computing real time views

Understanding the validity of CAP theorem

Assume a distributed key/value database with a replication factor of 3



$Key\_set1 \cap Key\_set2 \cap Key\_set3 \cap Key\_set4 \neq \emptyset$



If one machine becomes unavailable - > you can retrieve data from a different location but …

The important question to be answered is how to handle writes when the system is partitioned?

Computing real time views

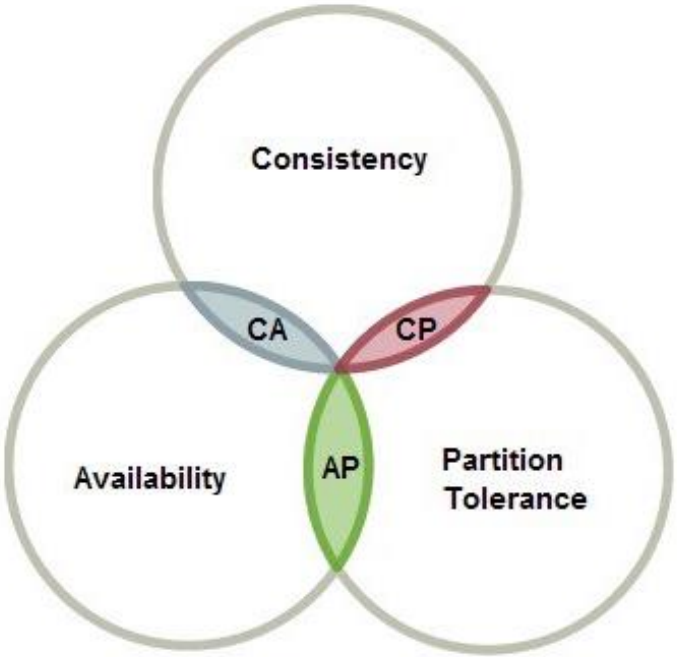
Understanding the validity of CAP theorem

Strategies to handle writes when the system is partitioned

Data loss

1. **Updates are not performed unless all replicas can be updated at once**
   - This is a consistent strategy, but all data will not be available
   - -> CP systems

2. **Update whatever replicas are available and synchronize the replicas once the partition is resolved** – partial update strategy
   - The system becomes available but not consistent
   - -> AP systems

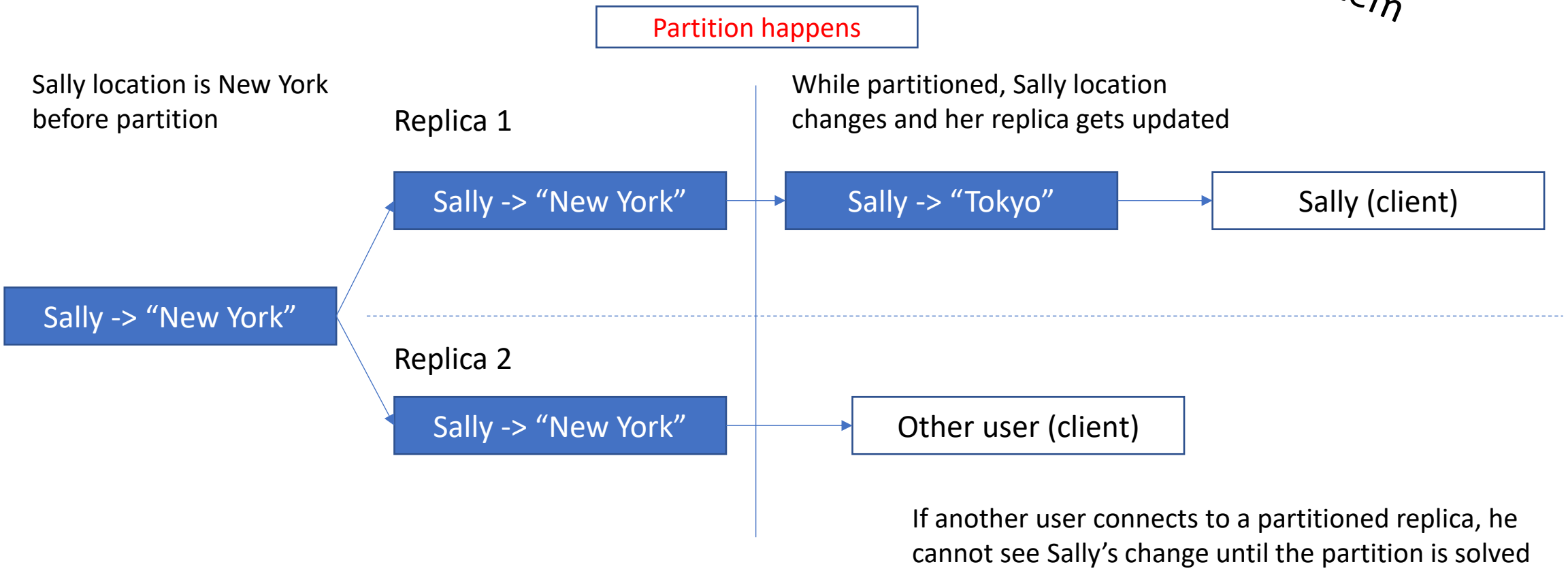


The batch layer and the serving layer chose availability over consistency

Computing real time views

Achieving eventual consistency in highly available system

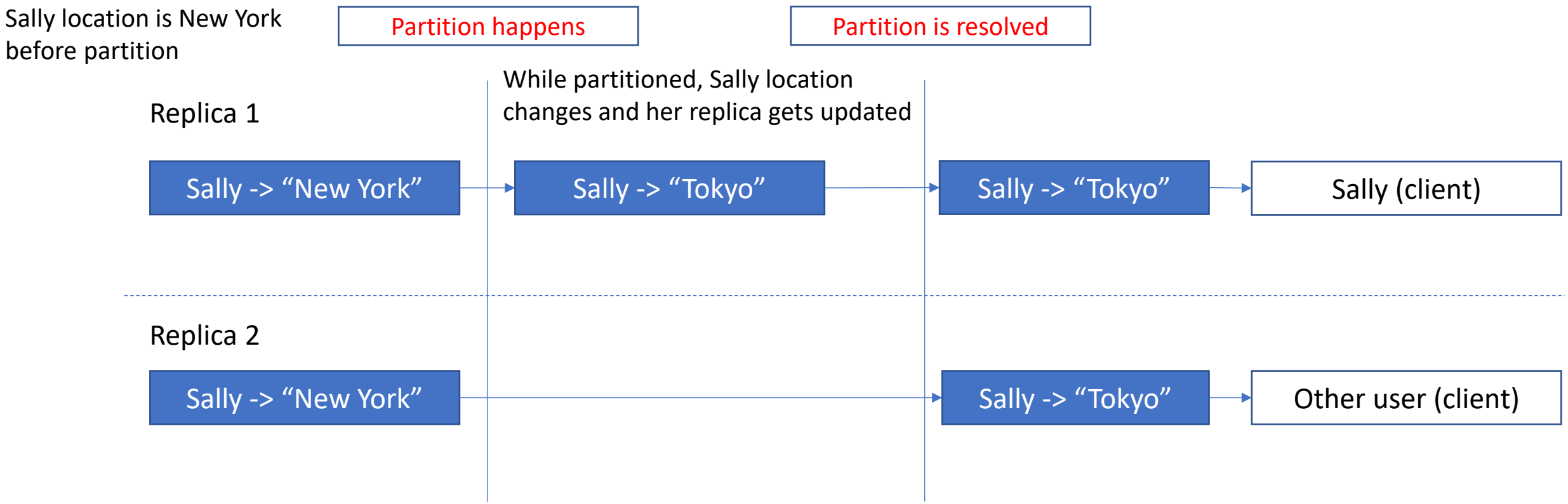
AP system



Computing real time views

Achieving eventual consistency in highly available system

AP system



Computing real time views

Achieving eventual consistency in highly available system

Example: Implementing eventual consistent counting

Scenario description

1. Assume counts are stored as values
2. Assume the network is partitioned, the partitions evolve independently and then the partition is corrected
3. Once the partition is corrected, the replicas have to be merged
4. Assume one replica has a count of 110 and the other 105

Question: what the new value should be?

Main cause for confusion: it is not known when the partitions the replicas started to diverge

Computing real time views

Achieving eventual consistency in highly available system

Example: Implementing eventual consistent counting

Scenario description

1. Assume counts are stored as values
2. Assume the network is partitioned, the partitions evolve independently and then the partition is corrected
3. Once the partition is corrected, the replicas have to be merged
4. Assume one replica has a count of 110 and the other 105

Question: what the new value should be?

Answer:

- If the replicas start diverging at 0 then the new value should be 110 + 105
- If they start diverging at 105, the new value should be 110
- The real answer should be in between these limits

Computing real time views

Achieving eventual consistency in highly available system

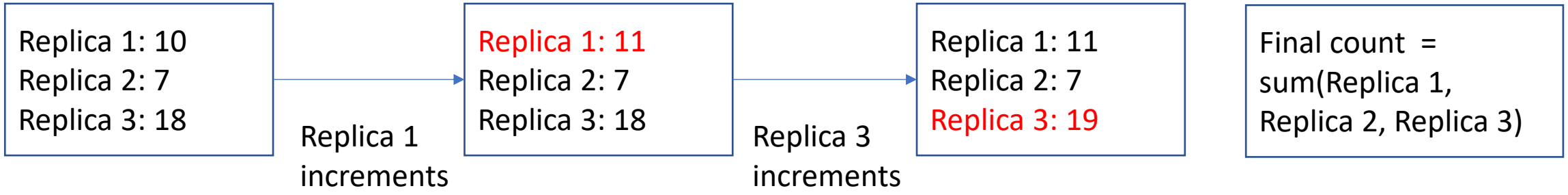
Example: Implementing eventual consistent counting

The solution to implement consistent counting is to use structures called ***conflict free replicated data types (CRDTs)***

CRDTs are data structures which can be replicated across multiple computers in a network, where the replicas can be updated independently and concurrently, without coordination, between the replicas, and where it is always mathematically possible to resolve inconsistencies that might come up.

How CRDTs work?

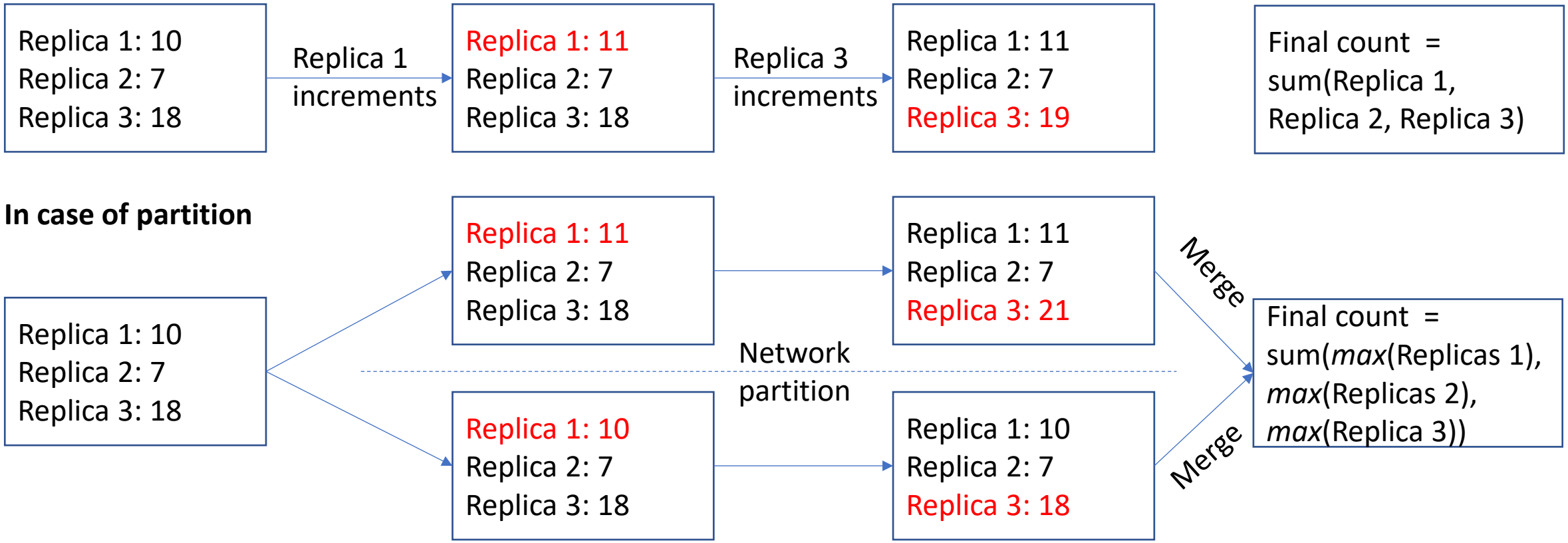
G-Counter is a CRDT which supports only increments



Computing real time views

Achieving eventual consistency in highly available system

How CRDTs work?



Computing real time views

Achieving eventual consistency in highly available system

Summary

1. Computation on the speed layer are much more complex to be implemented to achieve eventual consistency because:
   - Performing the computation is not sufficient
   - A strategy for repairing also needs to be implemented

2. The complexity cannot be avoided

3. However real time views are transient - if the real time view becomes corrupted the batch and serving layer will correct the mistake. Therefore the corruption is only temporary.

Computing real time views

Updating the real-time views

Synchronous updates vs. **Asynchronous updates**

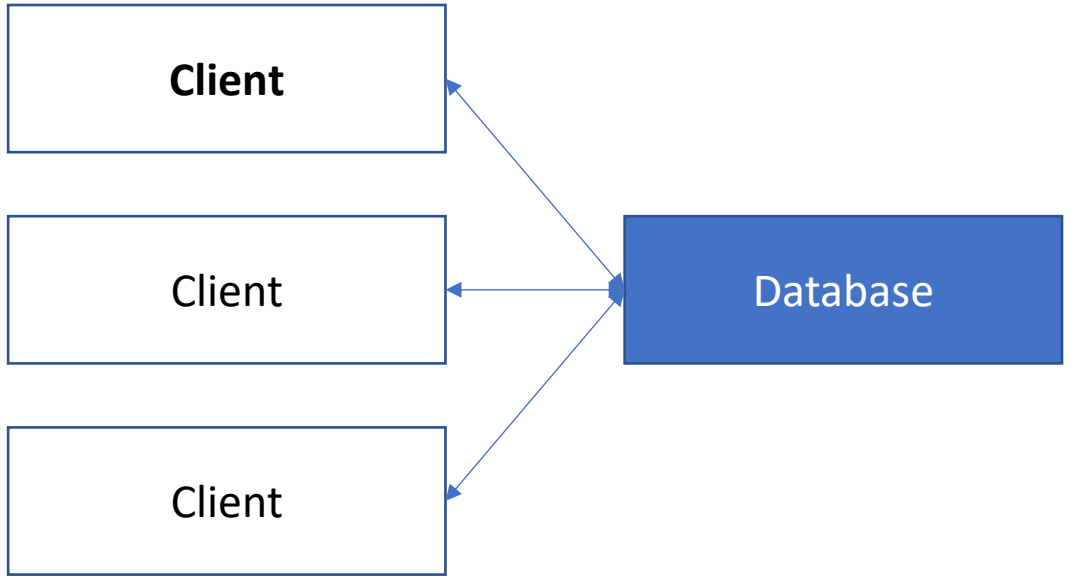
Asynchronous vs synchronous updates

Synchronous updates

Principle: the application issues a request to the database and blocks until the request is solved

Properties:

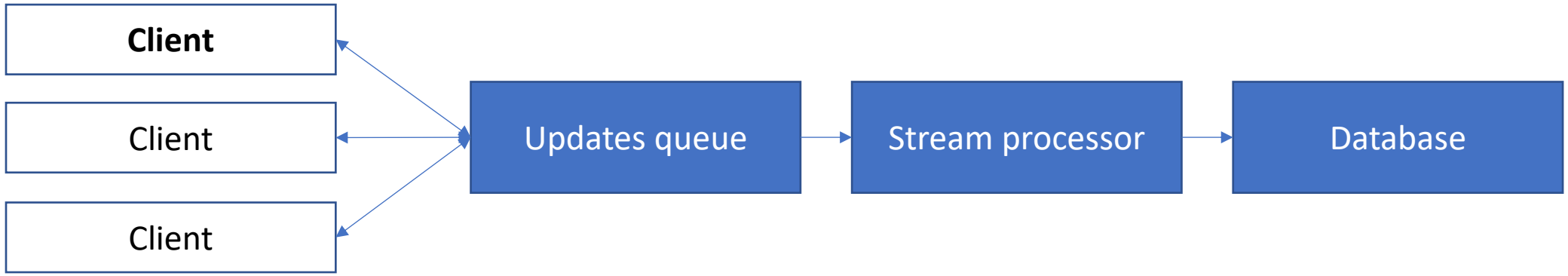
1. Sync updates communicate directly with the database
2. Sync updates are fast
3. They facilitate the coordination of the update with other aspects of the application (e.g. displaying a spinning cursor while waiting for the update to be completed)



Asynchronous vs synchronous updates

Asynchronous updates

Principle: the requests are placed in a queue and the updates are processed at a later time.



Properties (drawbacks):

1. Async updates introduce delays from a few milliseconds to a few seconds (or longer in case of an excess of requests)
2. Async updates are slower than sync updates
3. It is impossible to coordinate them with other actions

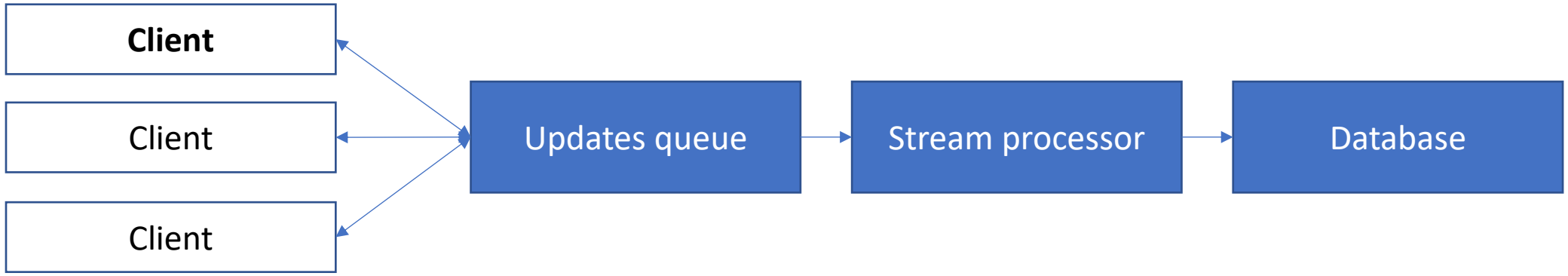
Asynchronous vs synchronous updates

Asynchronous updates

Principle: the requests are placed in a queue and the updates are processed at a later time.

Properties (advantages):

1. Increased throughput (you can read multiple messages from the queue and perform batch updates on the database)
2. Async updates handle varying load (requests are buffered in the queue).



Asynchronous vs synchronous updates

Synchronous vs Asynchronous updates

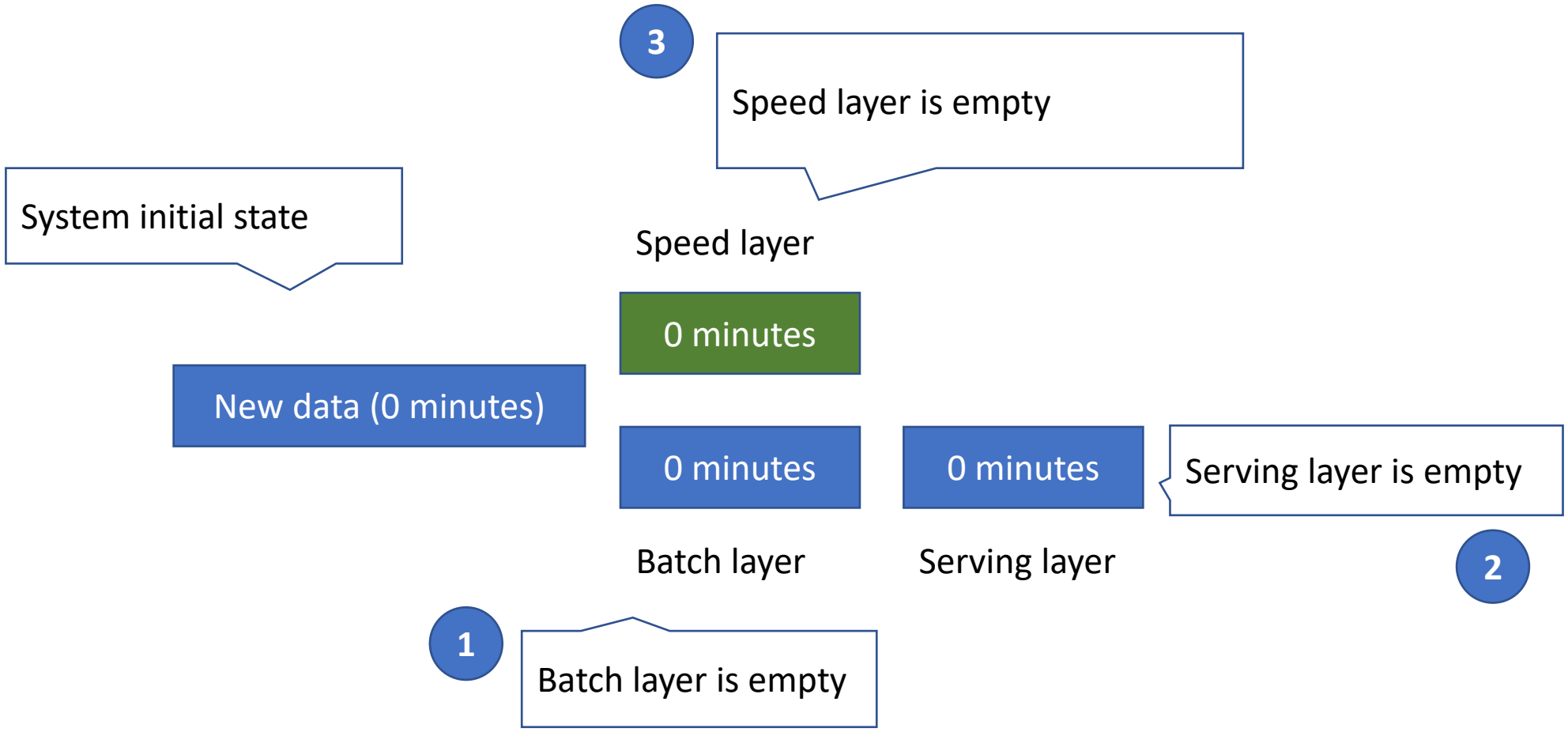
Synchronous updates	Asynchronous updates
Suited for transactional systems that interact with the users and require coordination with the user interface	Suited for analytics oriented workloads or workloads not requiring coordination
They are fast	They could introduce delays
Could overload the database is case of high load	They handle varying load

Expiring real time views

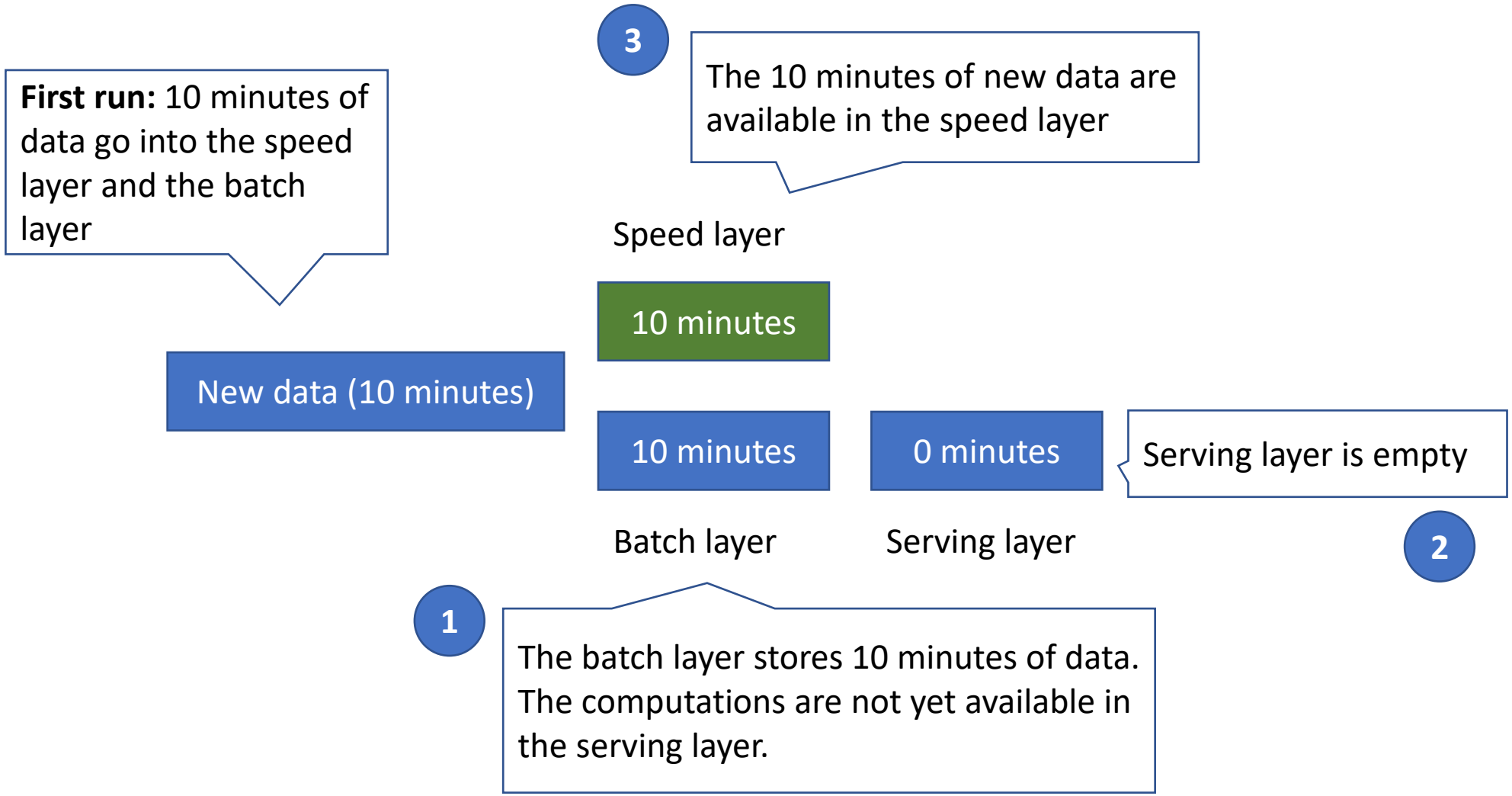
The batch layer and the serving layer constantly override the speed layer, therefore the speed layer only needs to represent data yet to be processed by the batch computation

Part of the processing results stored in the speed layer can be discarded once the batch layer finished a new run.

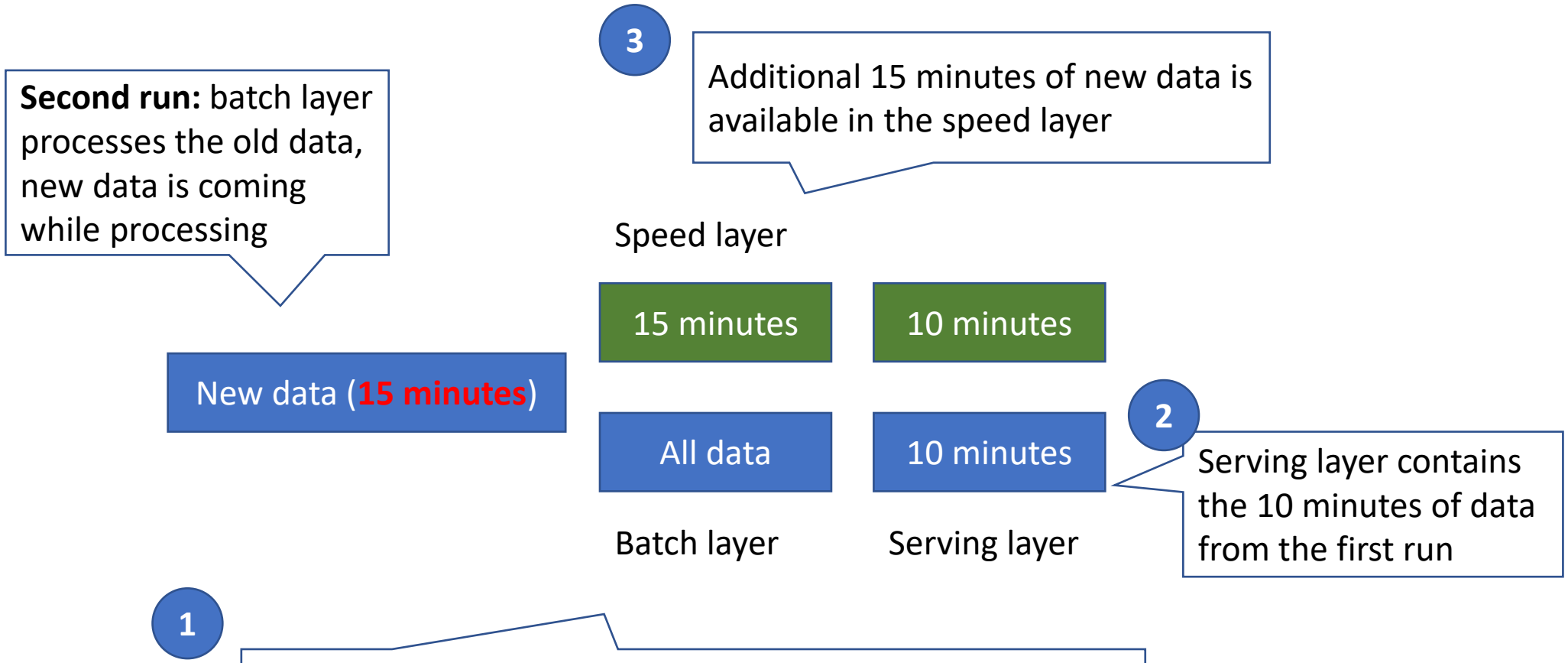
Expiring real time views



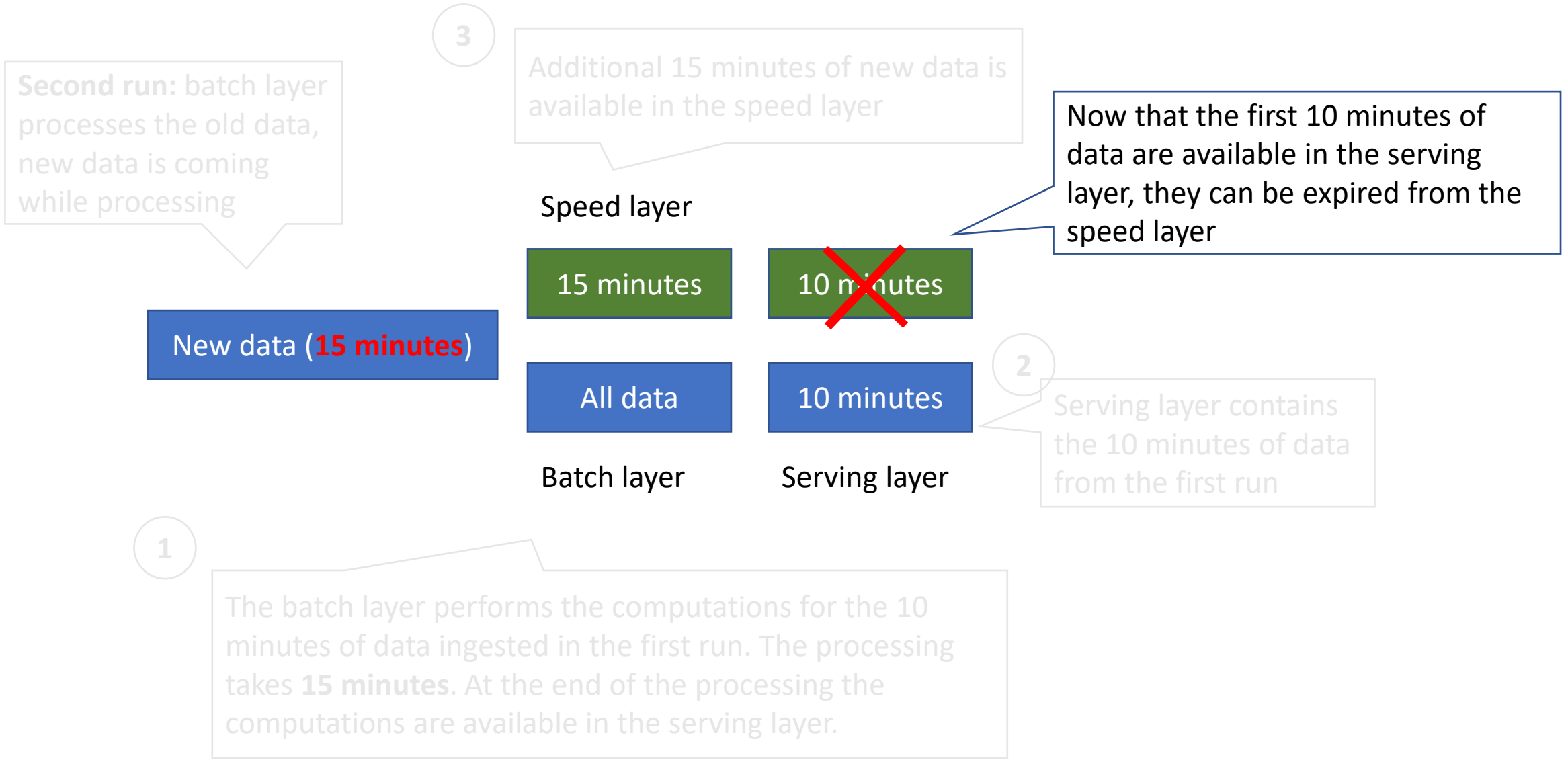
Expiring real time views



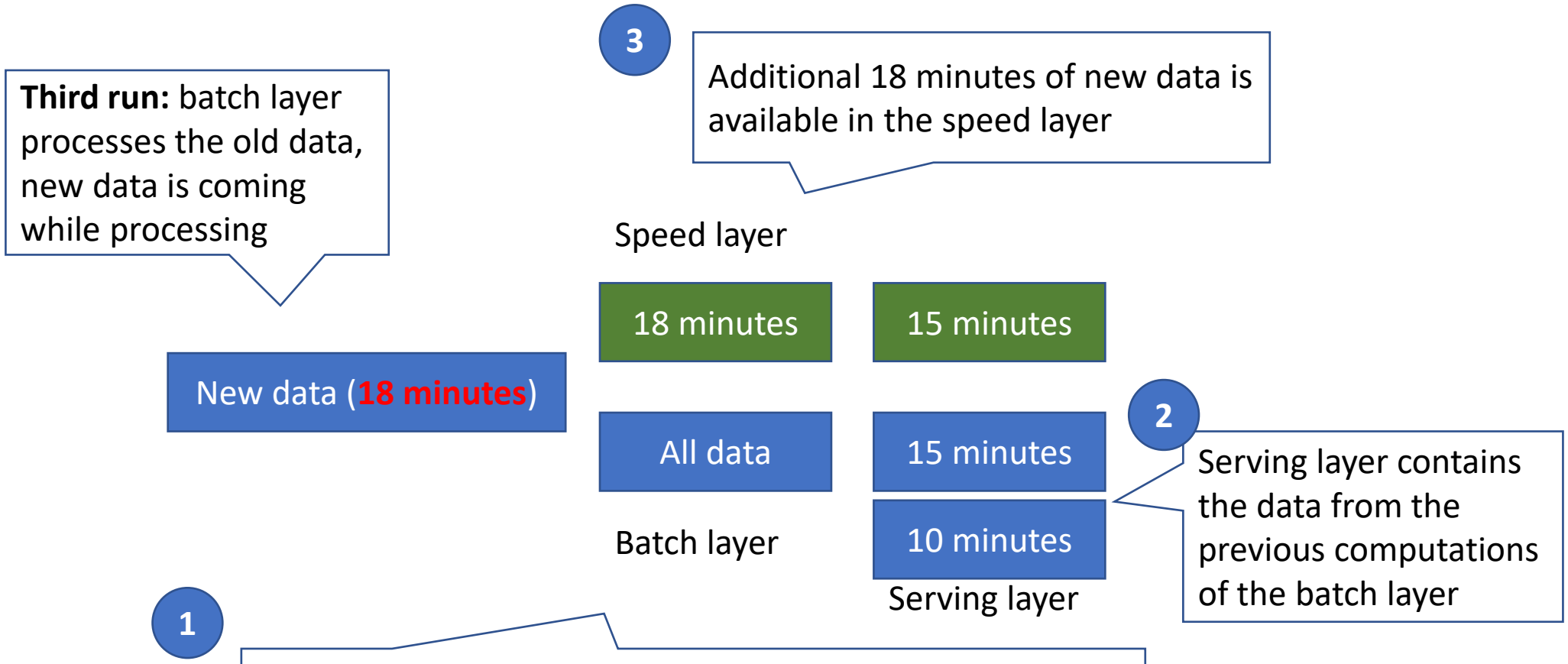
Expiring real time views



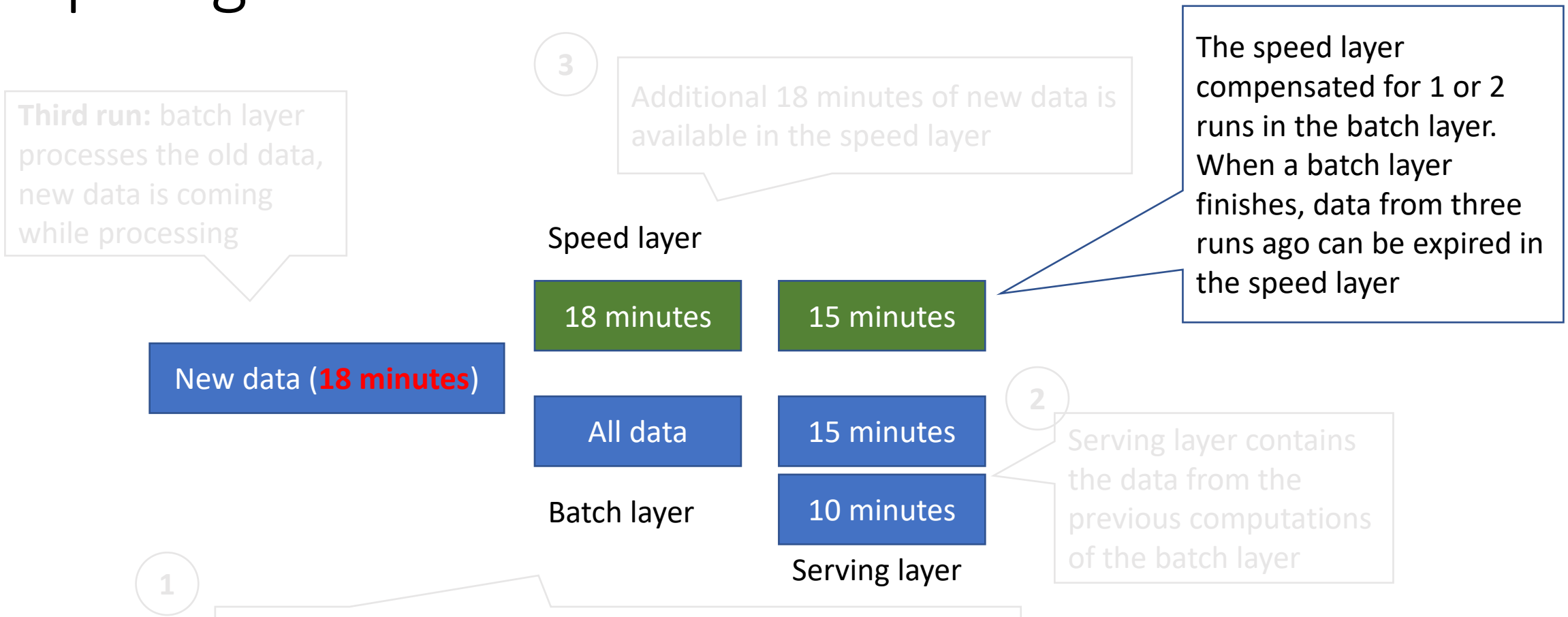
Expiring real time views



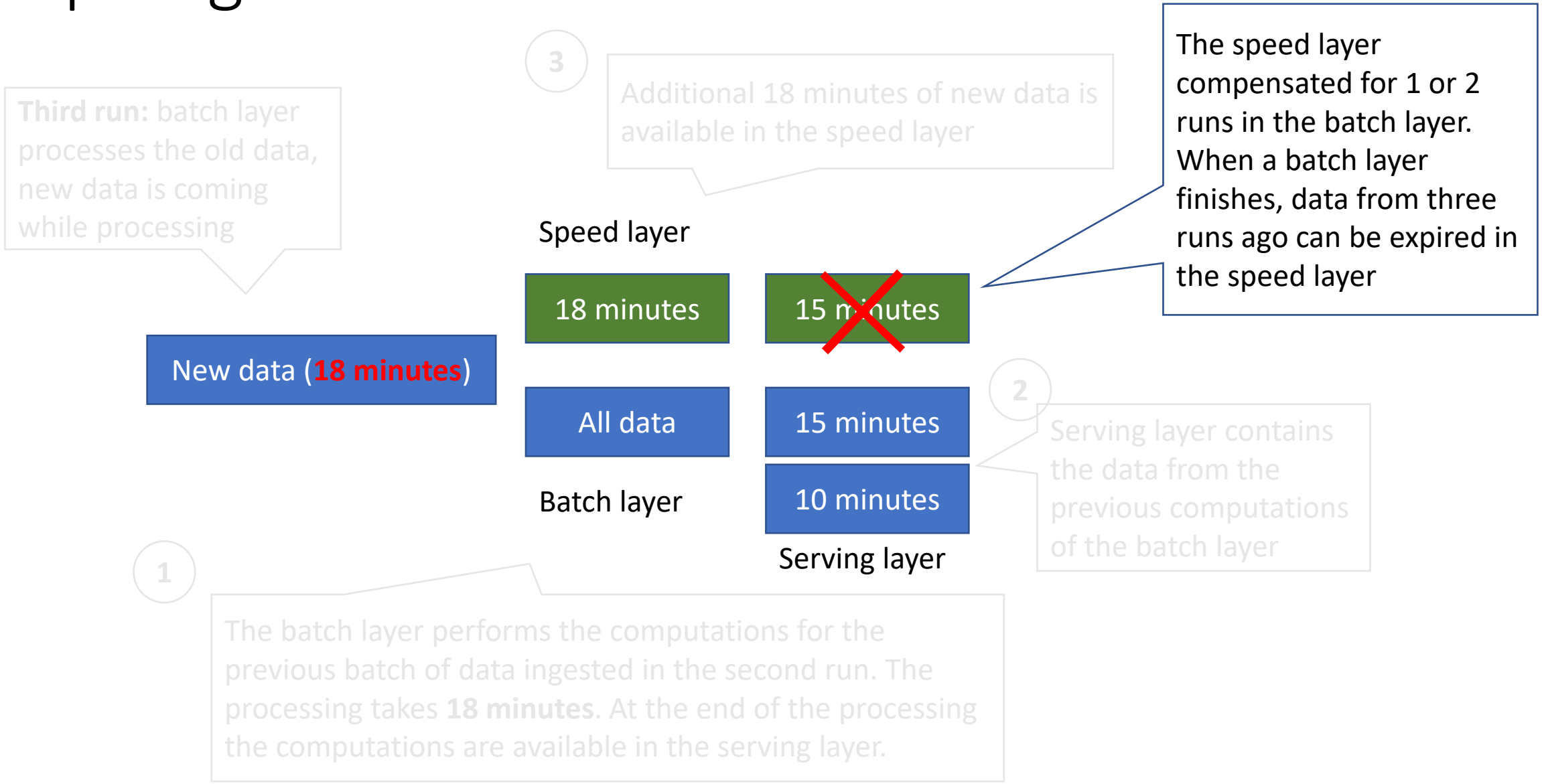
Expiring real time views



Expiring real time views



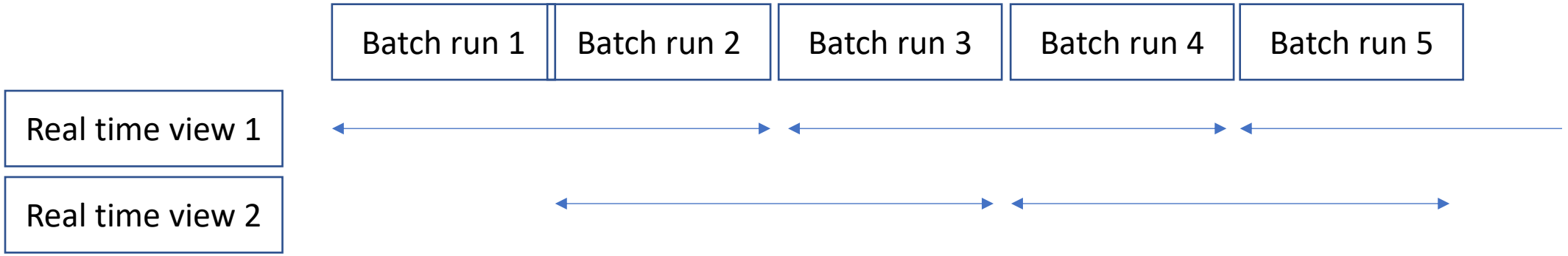
Expiring real time views



Expiring real time views

Strategy to expire real time views

Maintain two real time views and alternate clearing them after each batch layer run



One of the real time view will always contain the right data to compensate for the serving layer views

After each batch run the application should switch to reading from the real time view with more data.

This strategy does introduce redundancy but it is an acceptable price for a general solution to expire real time views

Speed layer – take home

- Computations on the speed layer – interaction between incremental algorithms and the CAP theorem
- Updating the real time views: synchronous vs asynchronous
- Expiring the real time views
- Example of databases for the speed layer