Programming at Scale: Consistency

cs378h

Today

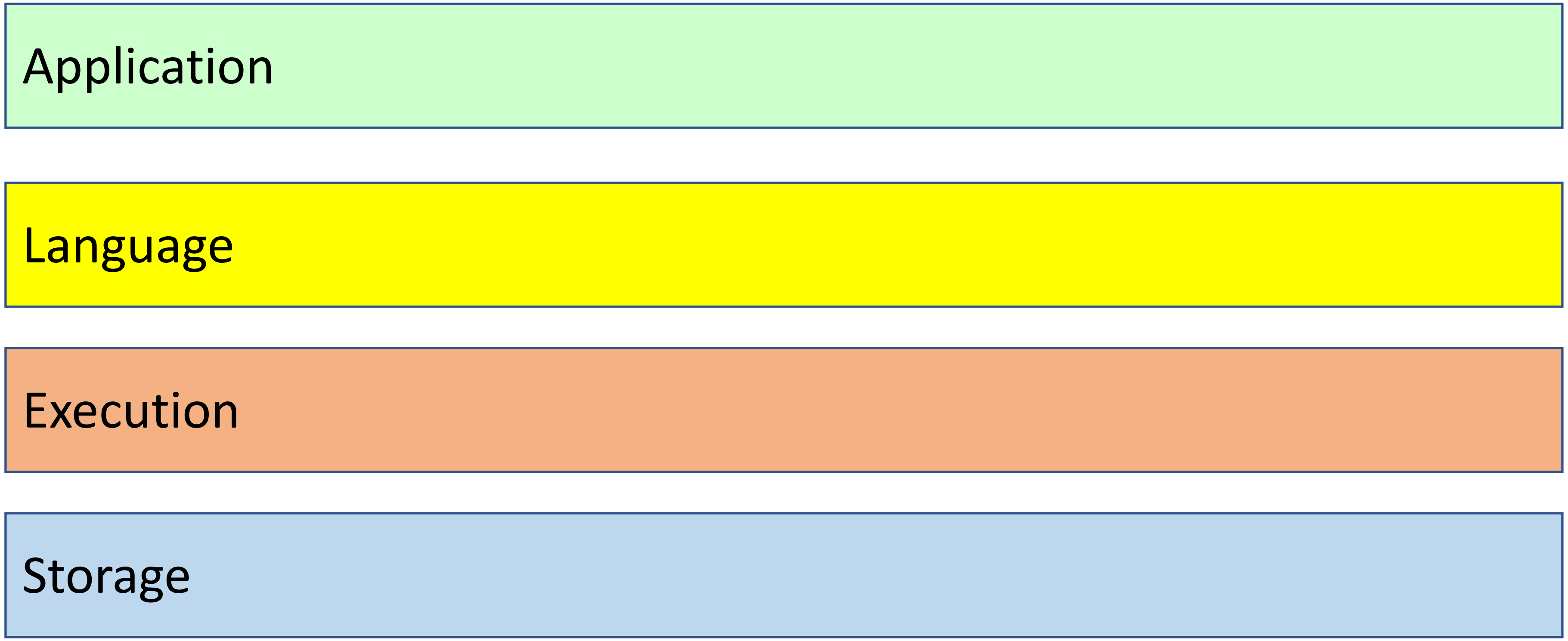
Questions?

Administrivia

Agenda:

- Concurrency & Consistency at Scale

Data-Parallel Computation Systems



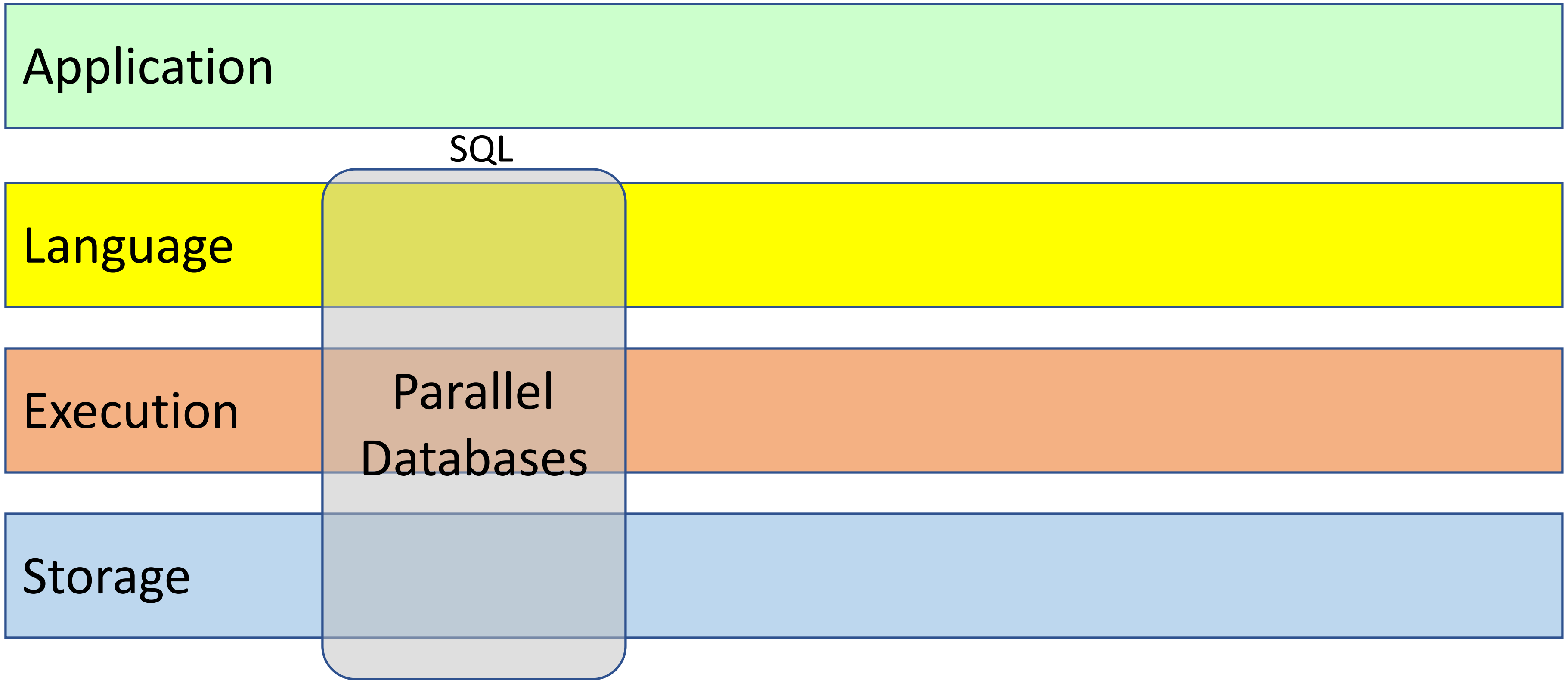
Application

Language

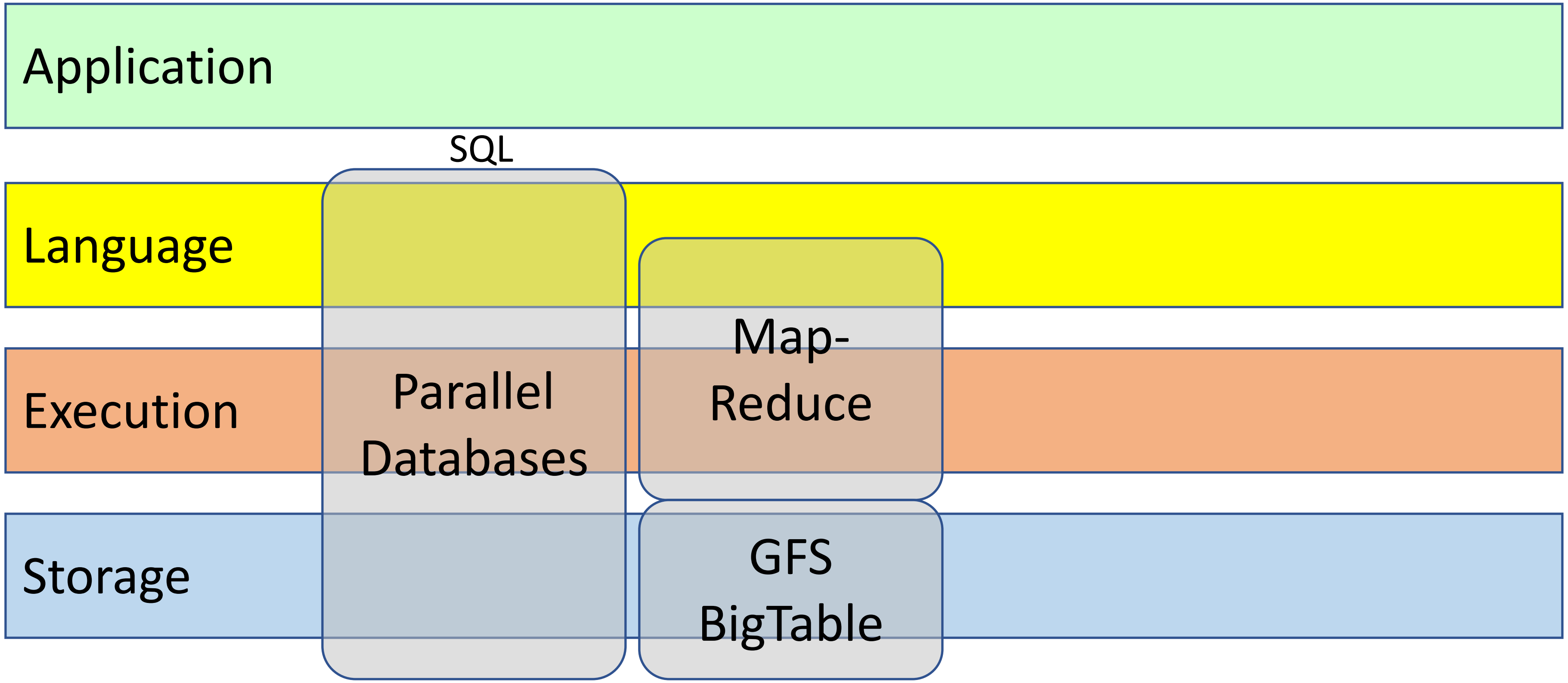
Execution

Storage

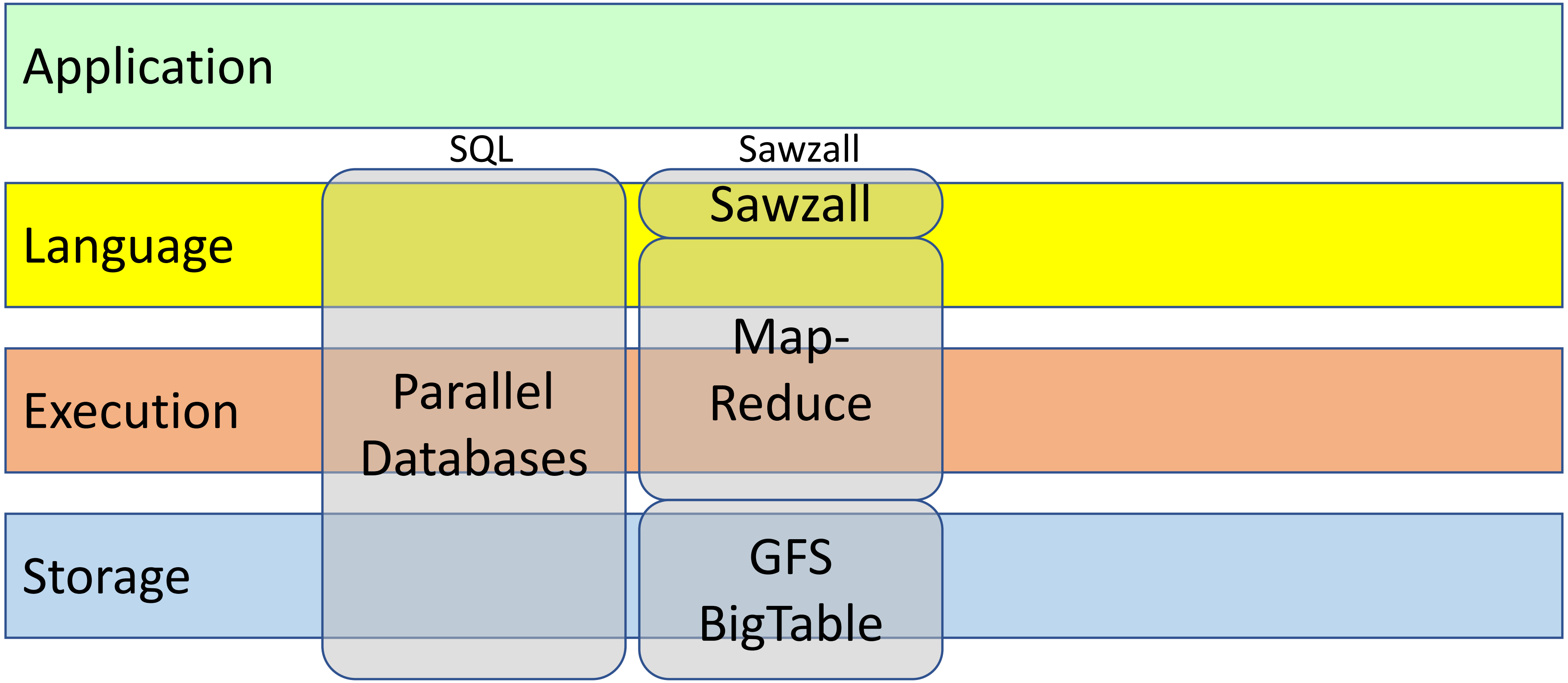
Data-Parallel Computation Systems



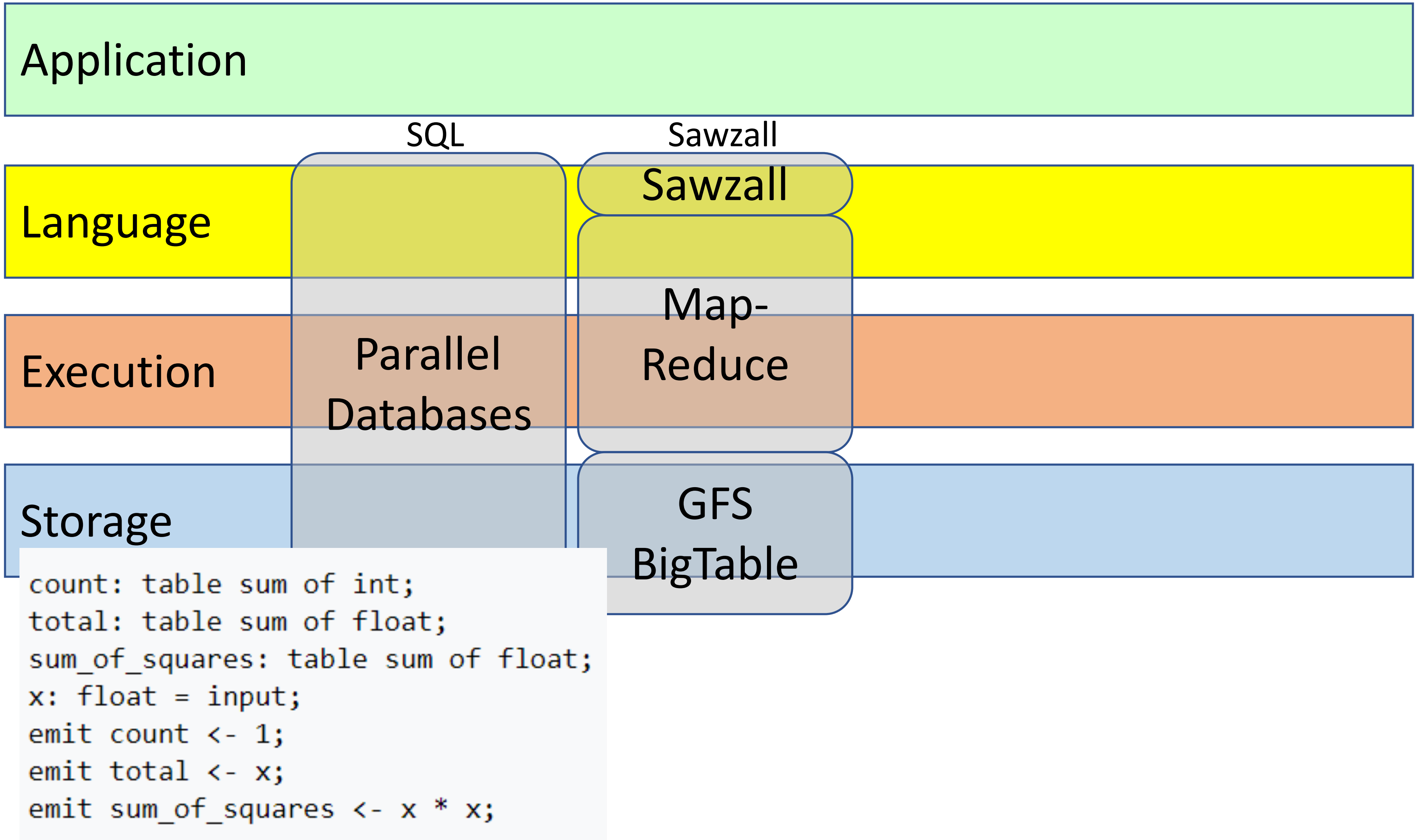
Data-Parallel Computation Systems



Data-Parallel Computation Systems

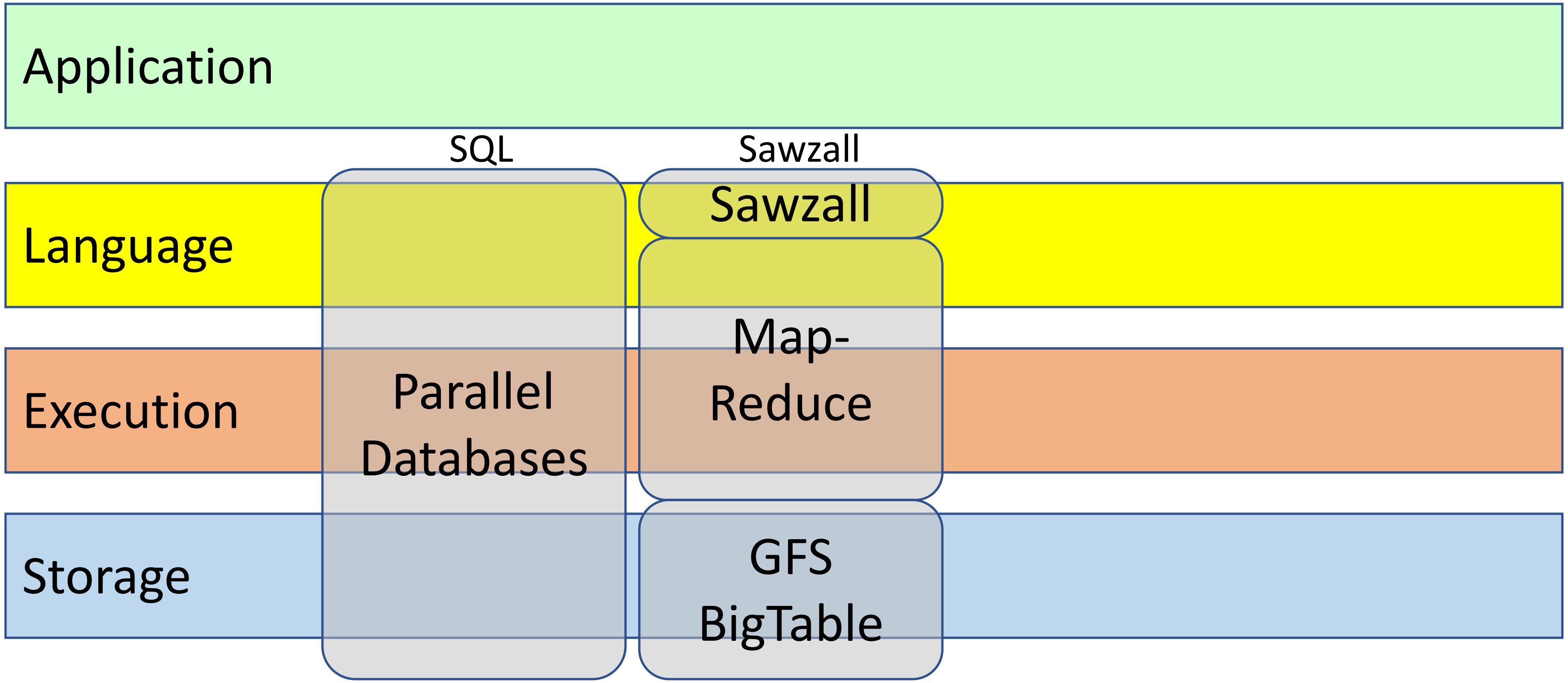


Data-Parallel Computation Systems

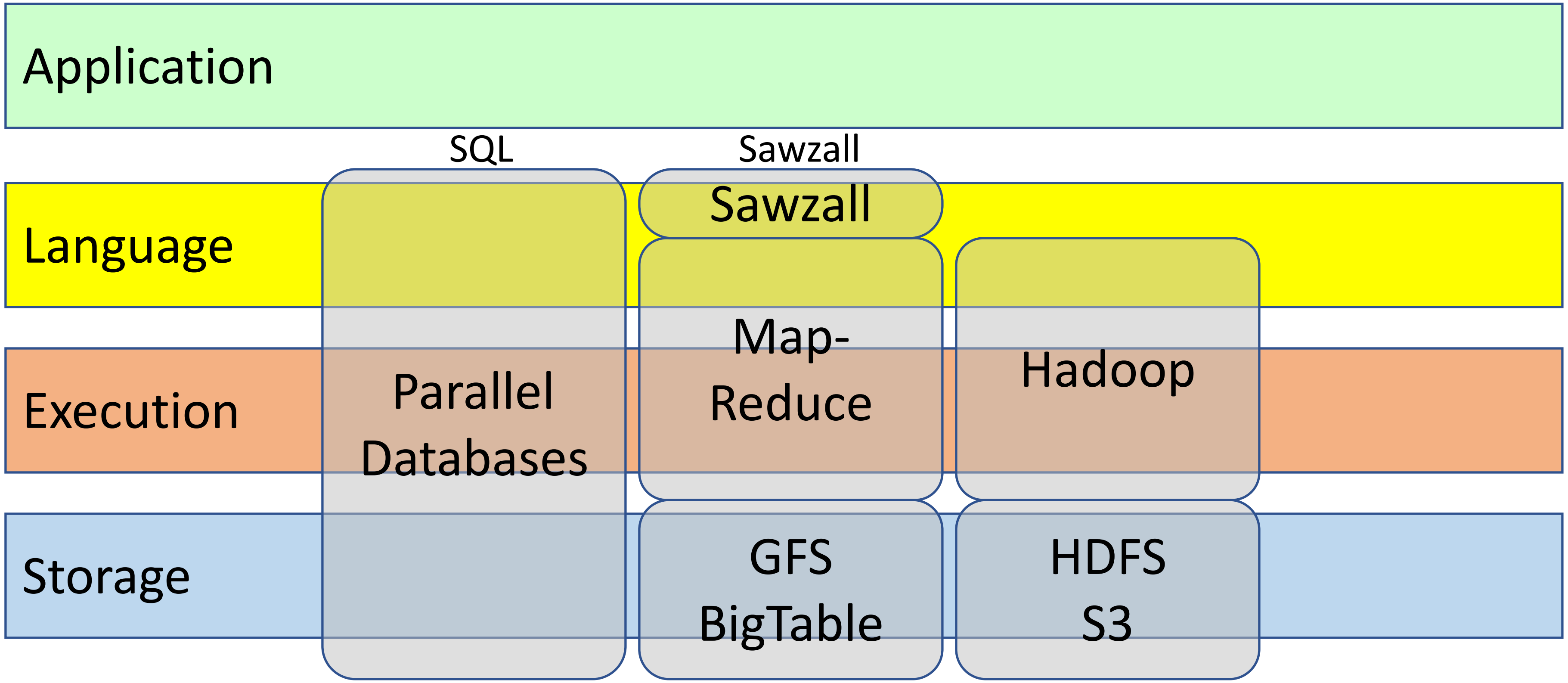


```
count: table sum of int;
total: table sum of float;
sum_of_squares: table sum of float;
x: float = input;
emit count <- 1;
emit total <- x;
emit sum_of_squares <- x * x;
```

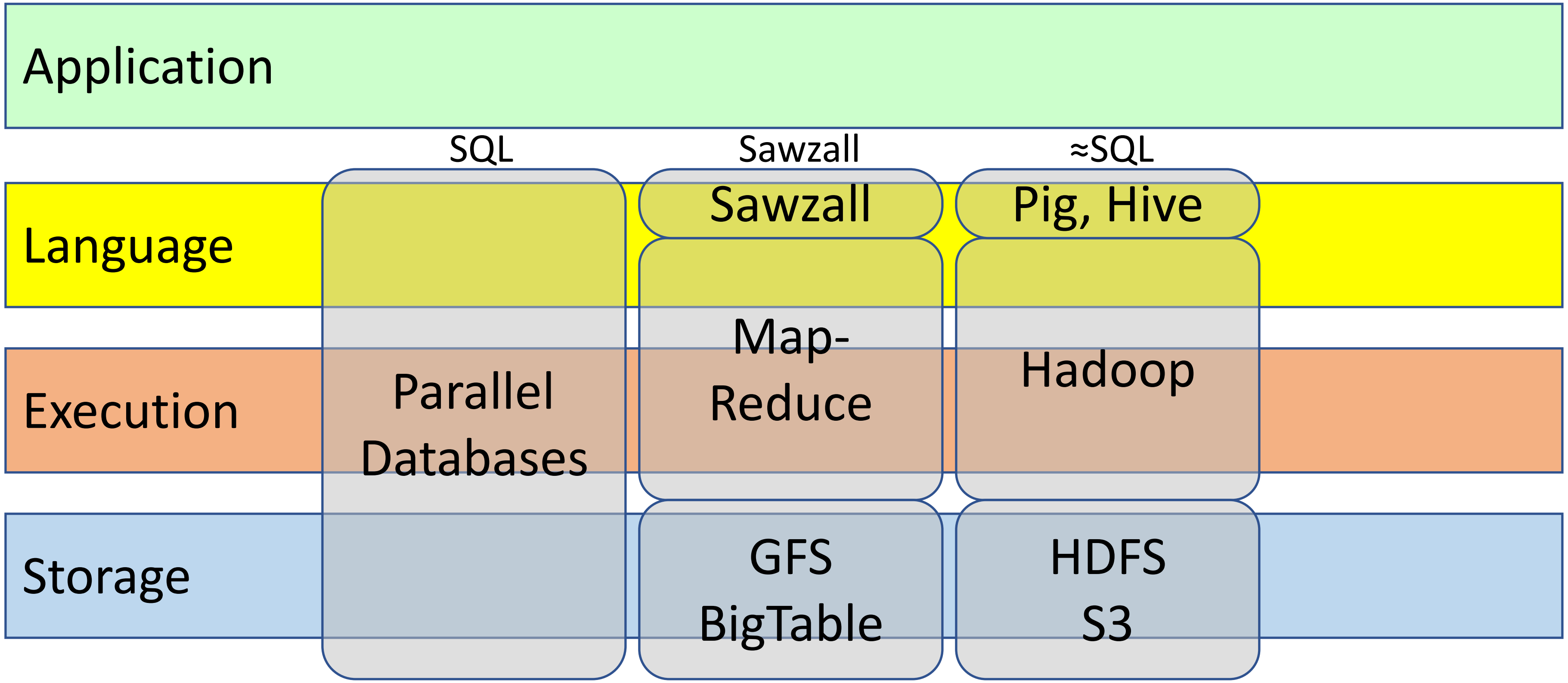
Data-Parallel Computation Systems



Data-Parallel Computation Systems



Data-Parallel Computation Systems



Application

Language

Execution

Storage

SQL

Parallel Databases

Sawzall

Sawzall

Map-Reduce

GFS BigTable

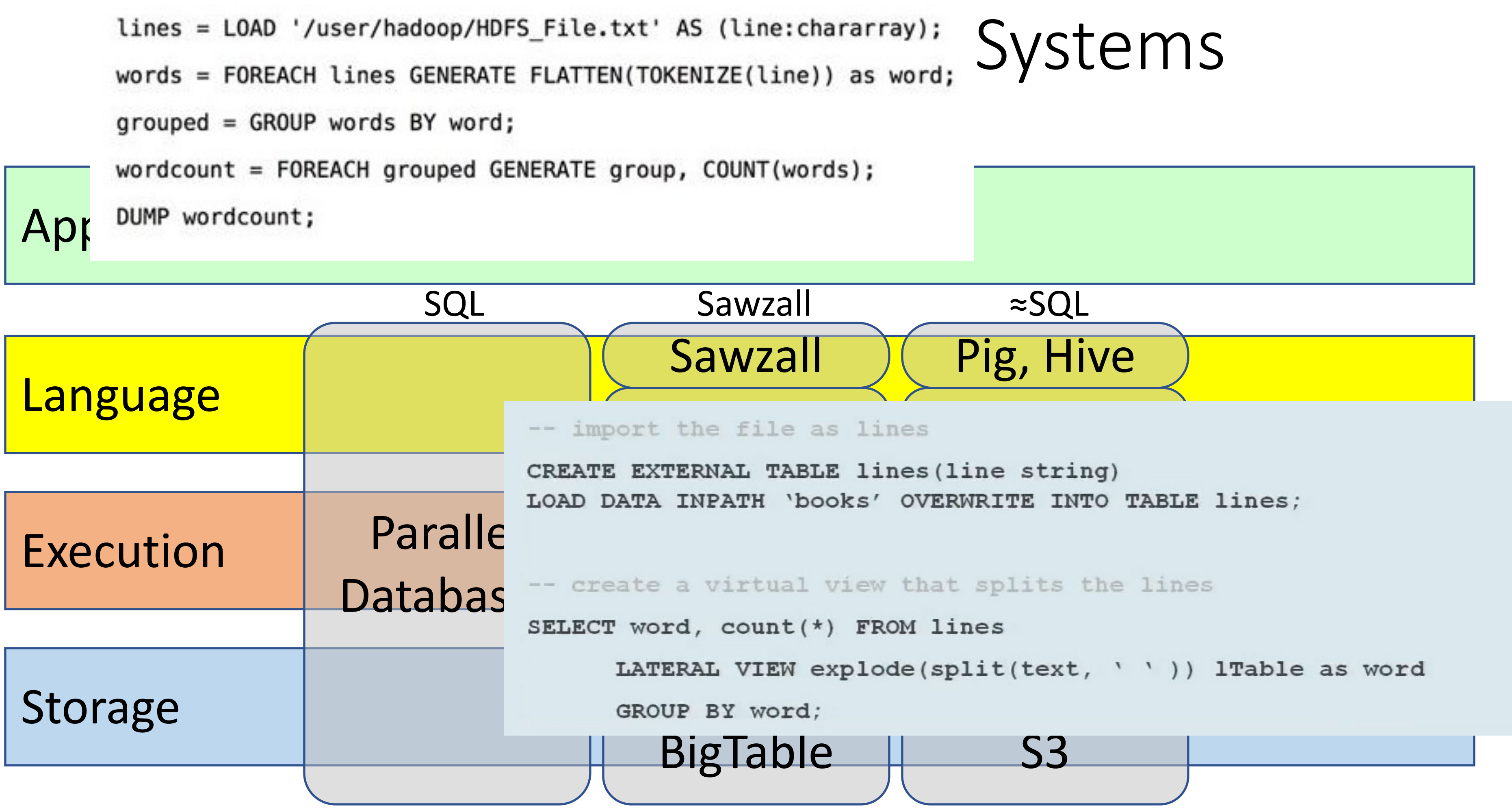
≈SQL

Pig, Hive

Hadoop

HDFS S3

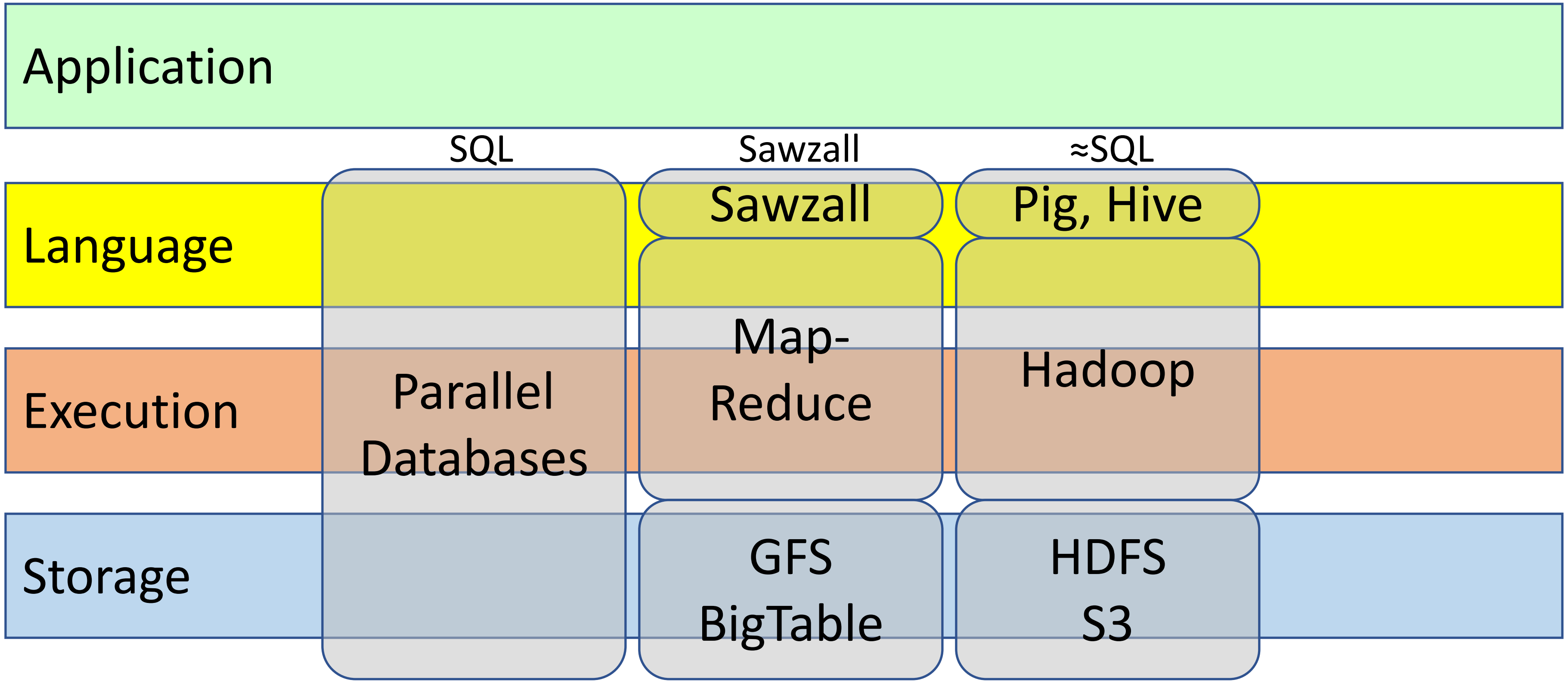
Systems



```
lines = LOAD '/user/hadoop/HDFS_File.txt' AS (line:chararray);

words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;

grouped = GROUP words BY word;

wordcount = FOREACH grouped GENERATE group, COUNT(words);

DUMP wordcount;
```

```
-- import the file as lines
CREATE EXTERNAL TABLE lines(line string)
LOAD DATA INPATH 'books' OVERWRITE INTO TABLE lines;


-- create a virtual view that splits the lines
SELECT word, count(*) FROM lines
       LATERAL VIEW explode(split(text, ' ' )) lTable as word
       GROUP BY word;
```

Data-Parallel Computation Systems



Application

Language

Execution

Storage

SQL

Parallel Databases

Sawzall

Sawzall

Map-Reduce

GFS BigTable

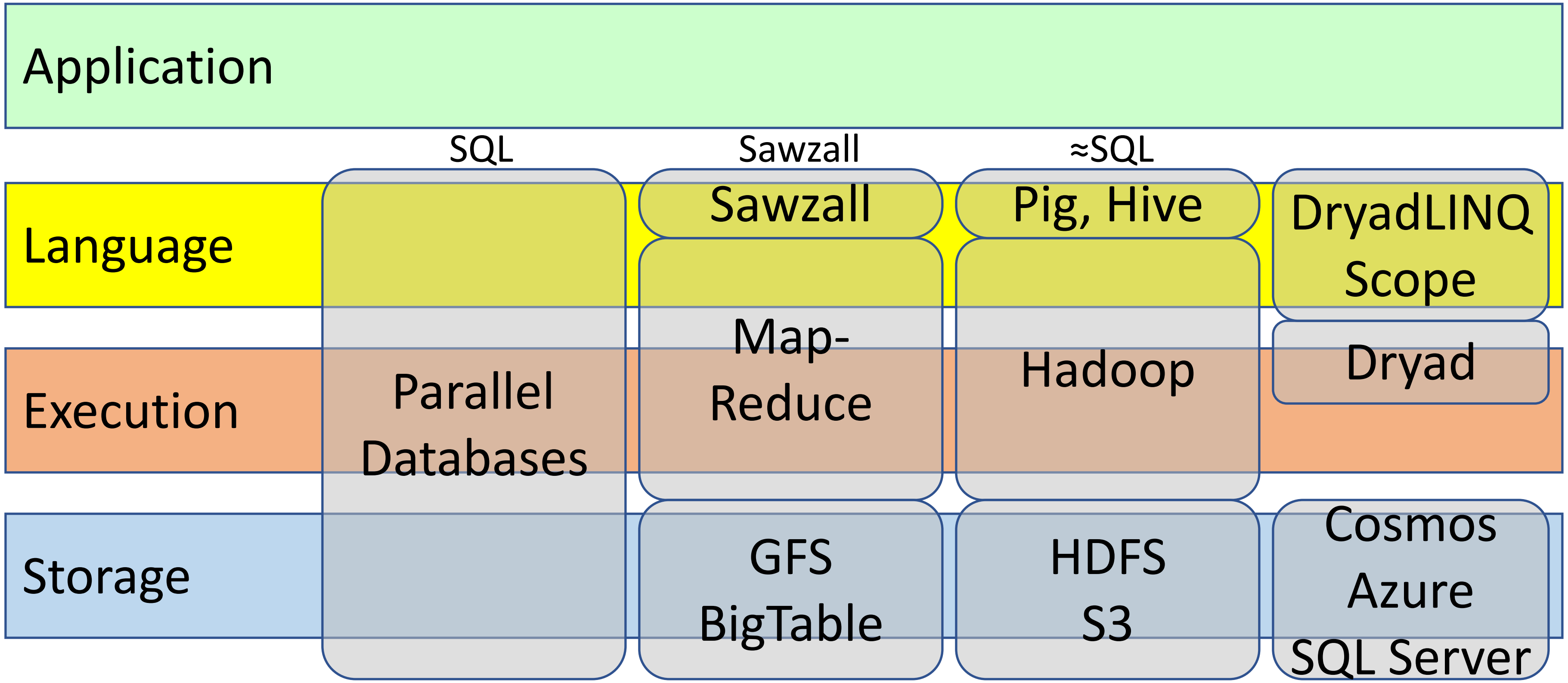
≈SQL

Pig, Hive

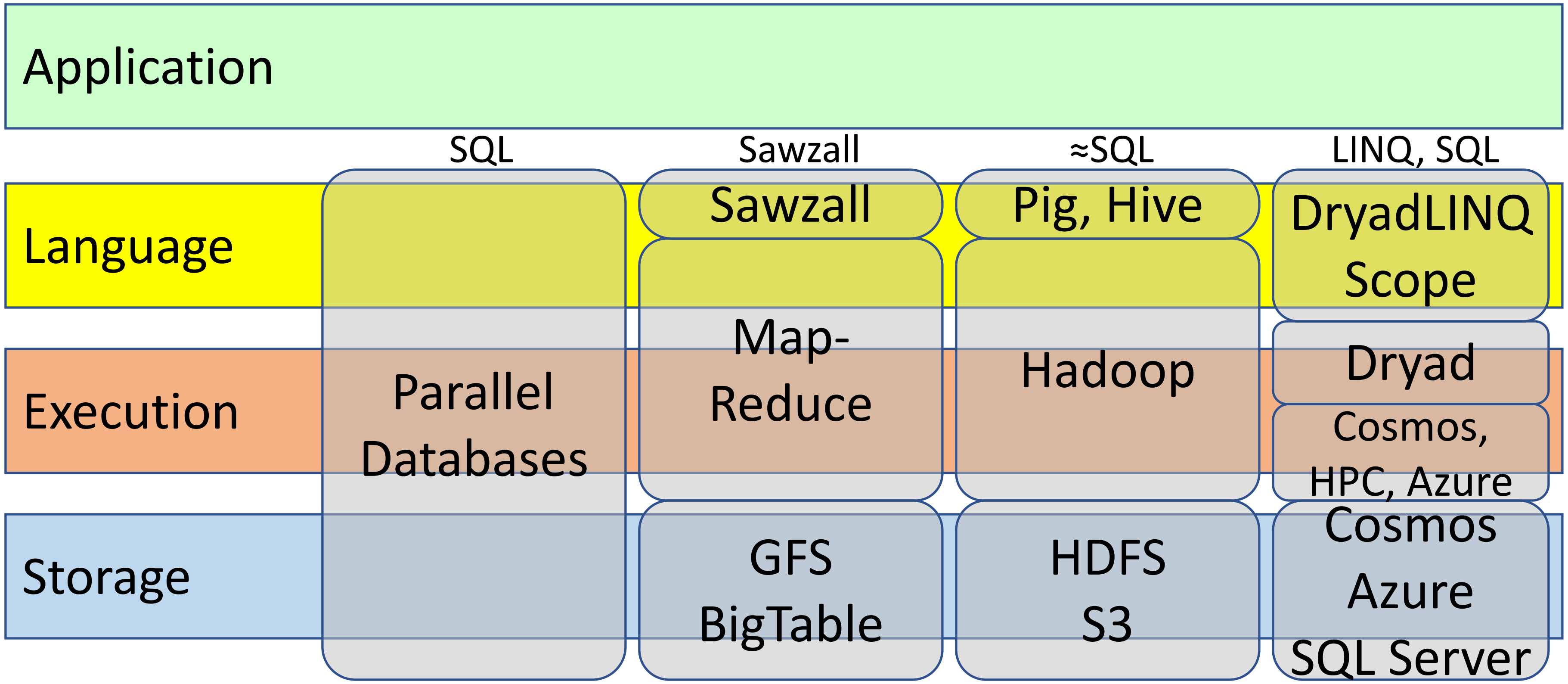
Hadoop

HDFS S3

Data-Parallel Computation Systems



Data-Parallel Computation Systems



Application

SQL

Sawzall

≈SQL

LINQ, SQL

Language

Sawzall

Pig, Hive

DryadLINQ

Scope

Execution

Parallel

Databases

Map-

Reduce

Hadoop

Dryad

Cosmos,

HPC, Azure

Storage

GFS

BigTable

HDFS

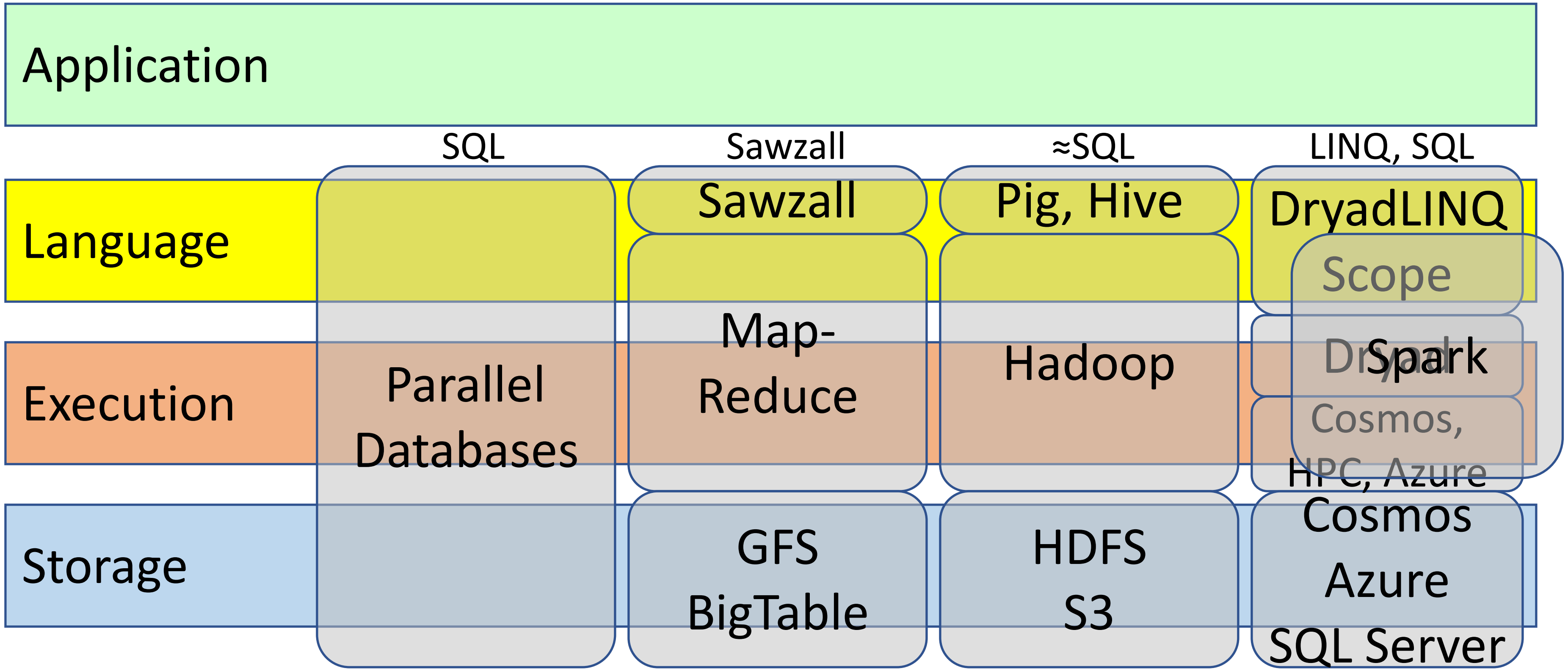
S3

Cosmos

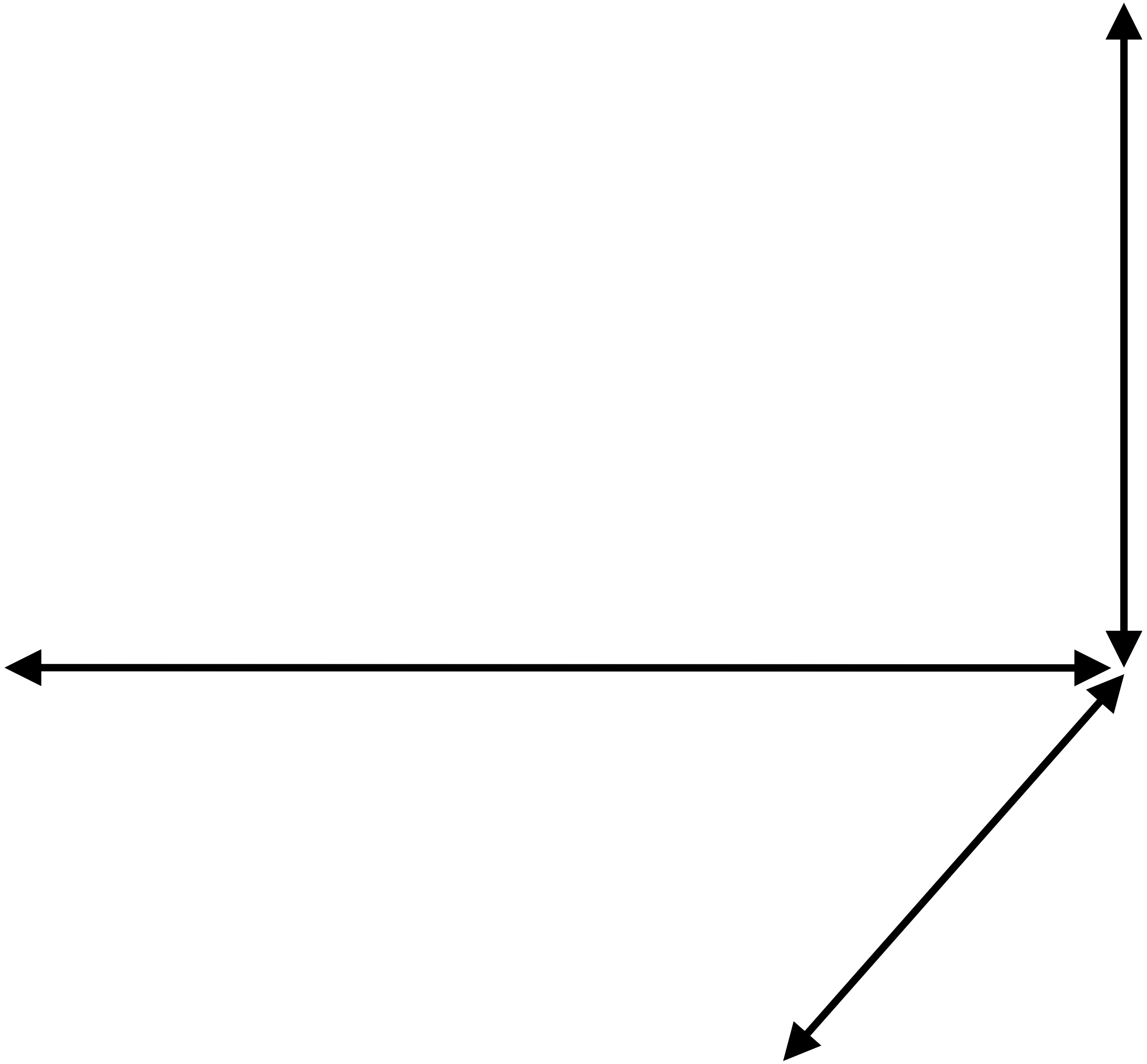
Azure

SQL Server

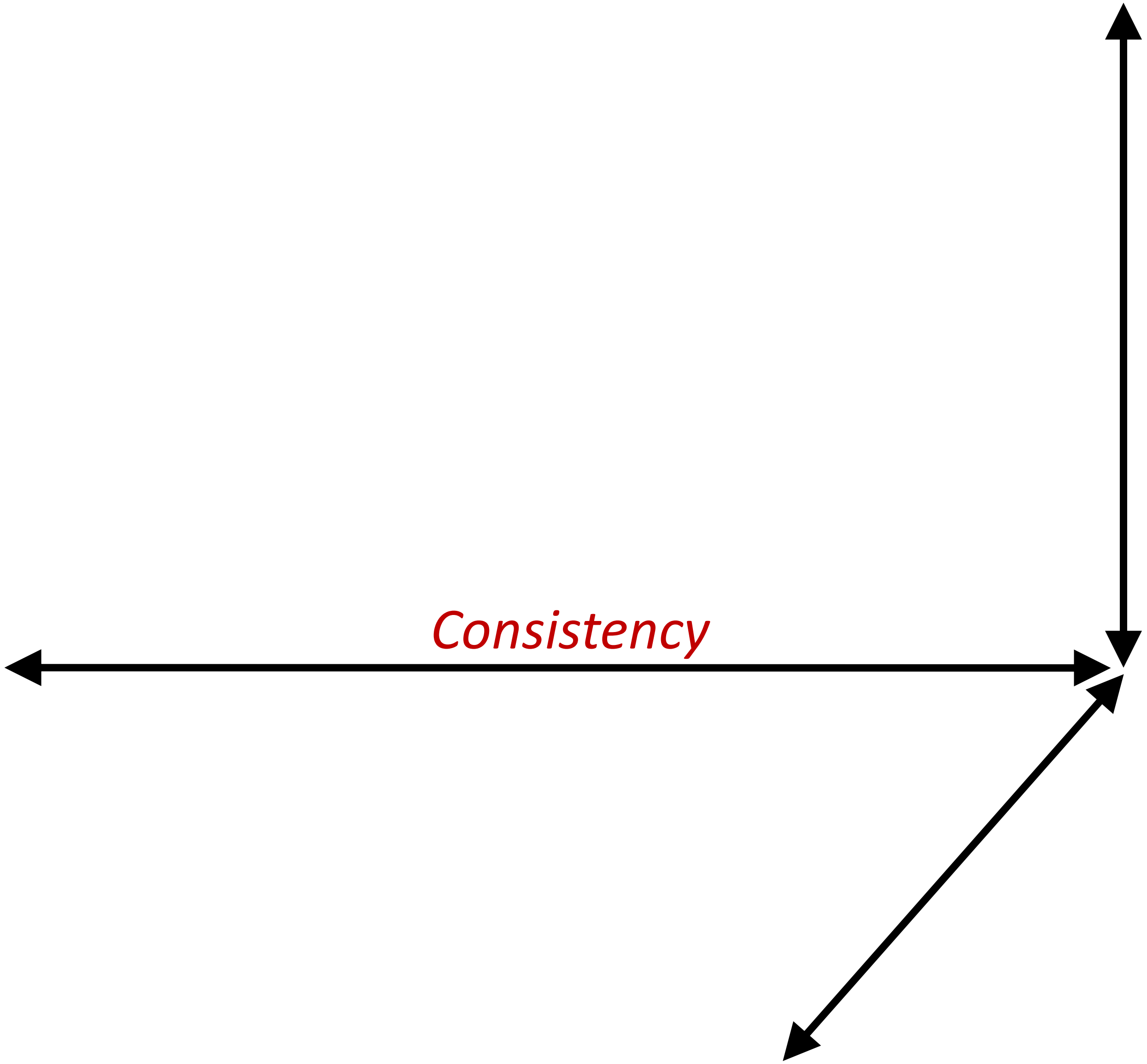
Data-Parallel Computation Systems



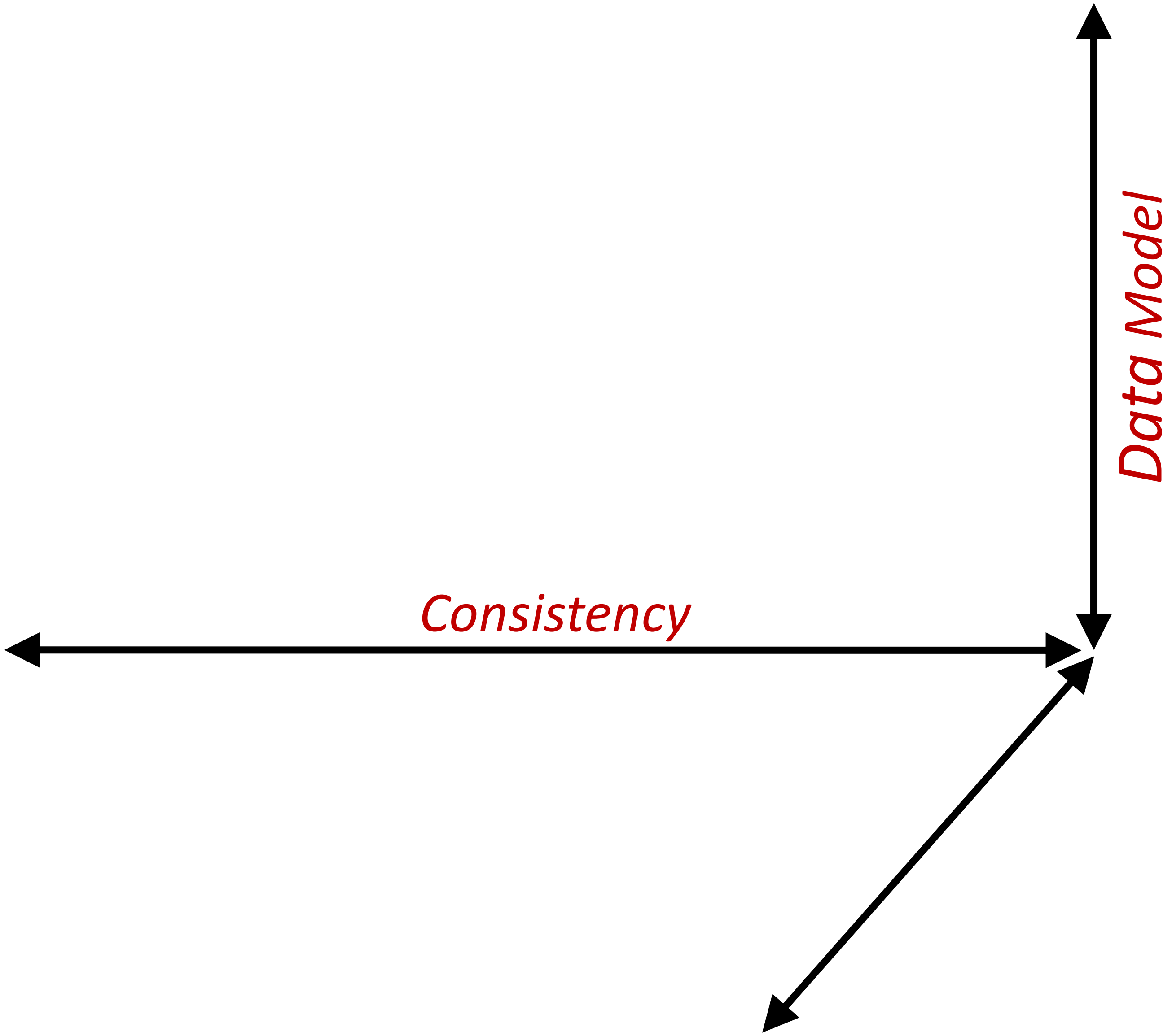
(Yet) Another Framework



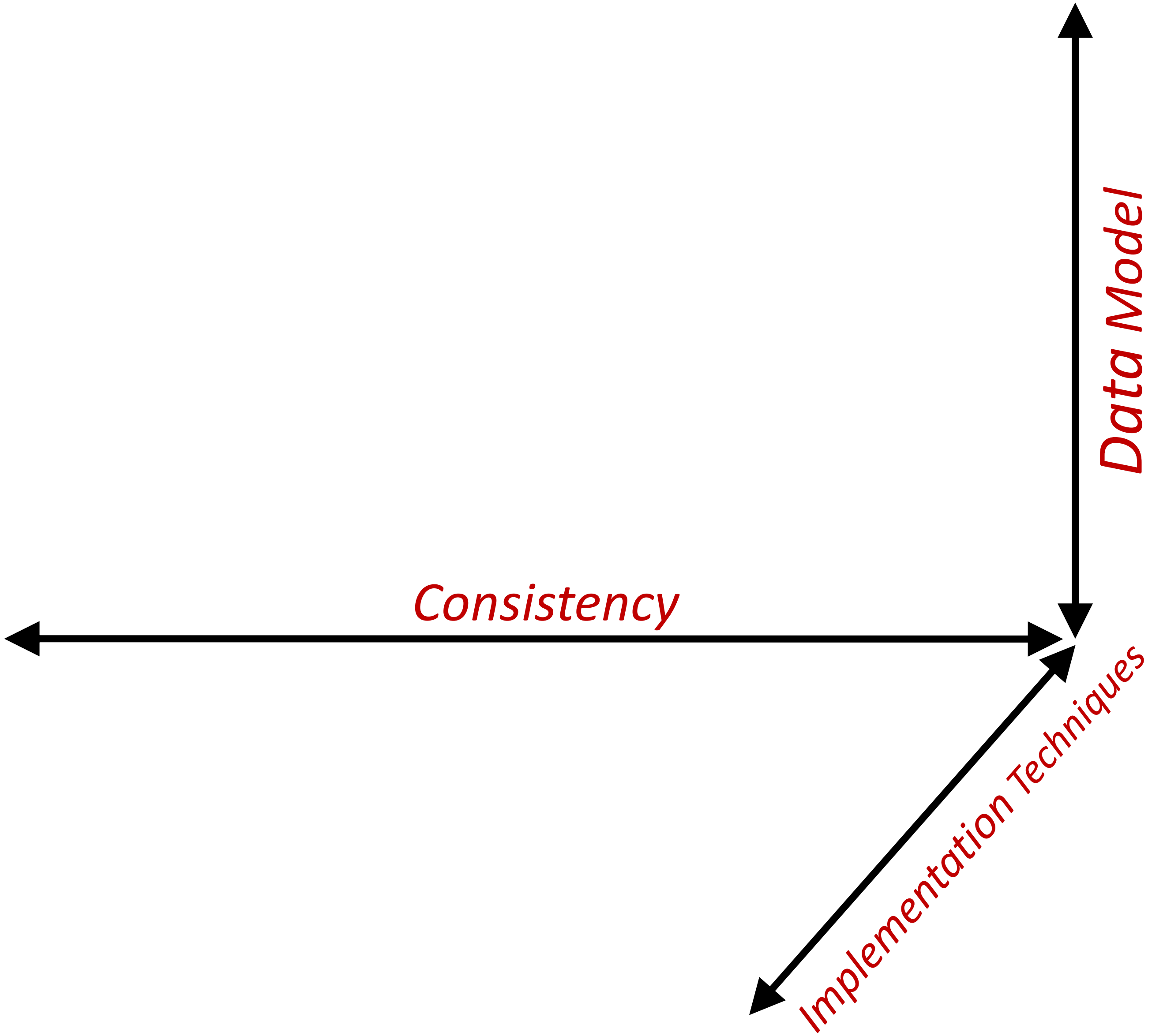
(Yet) Another Framework



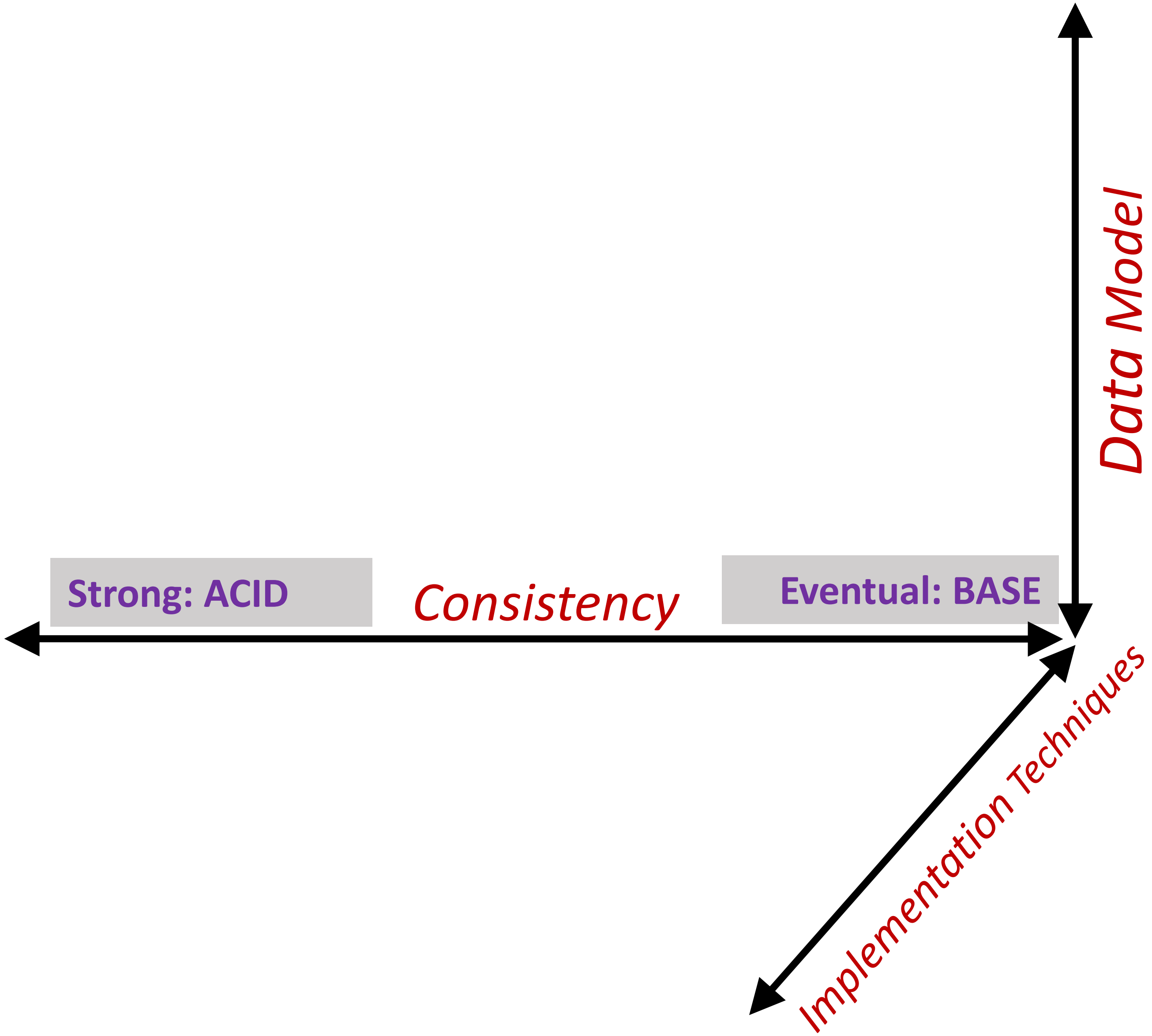
(Yet) Another Framework



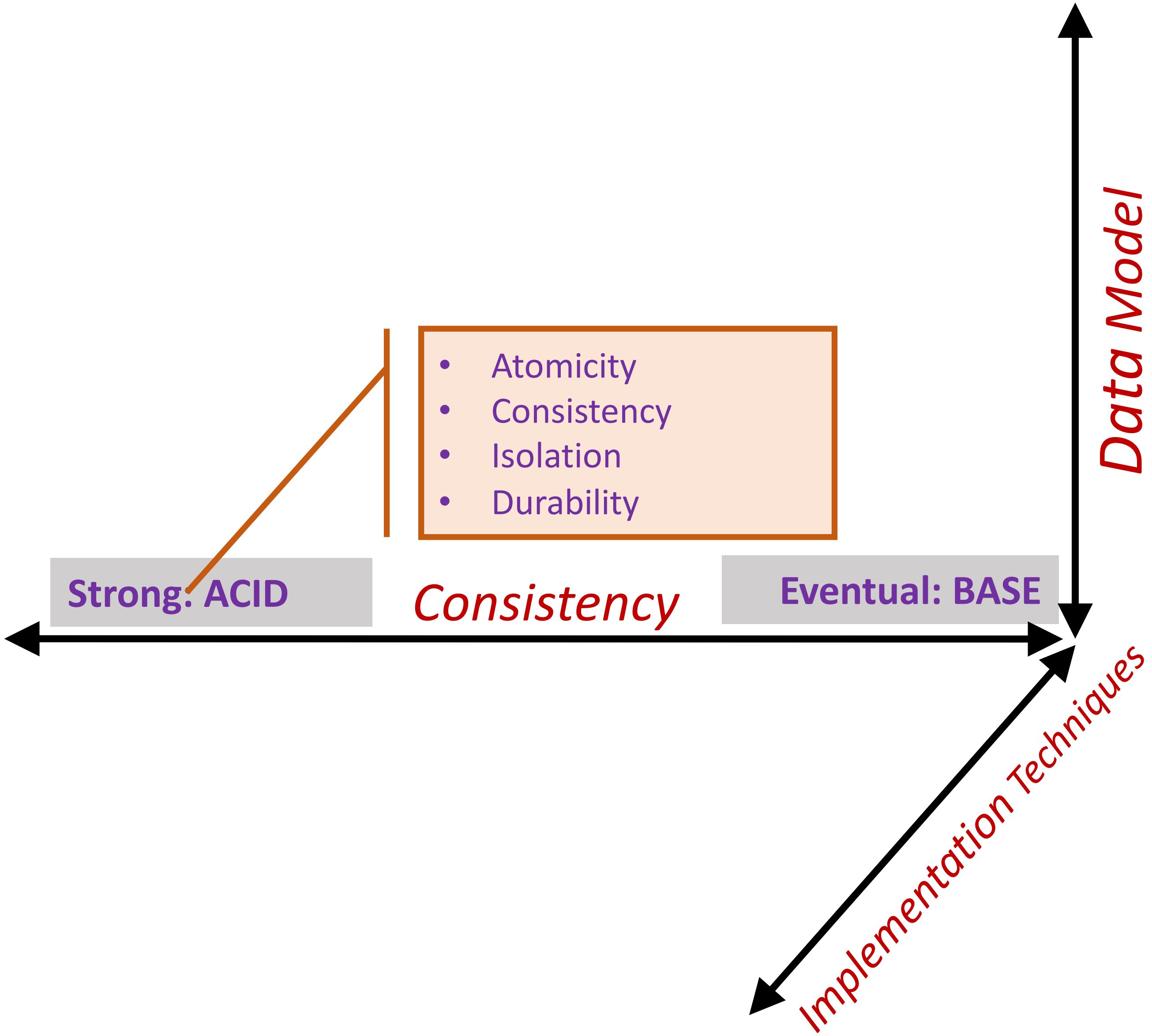
(Yet) Another Framework



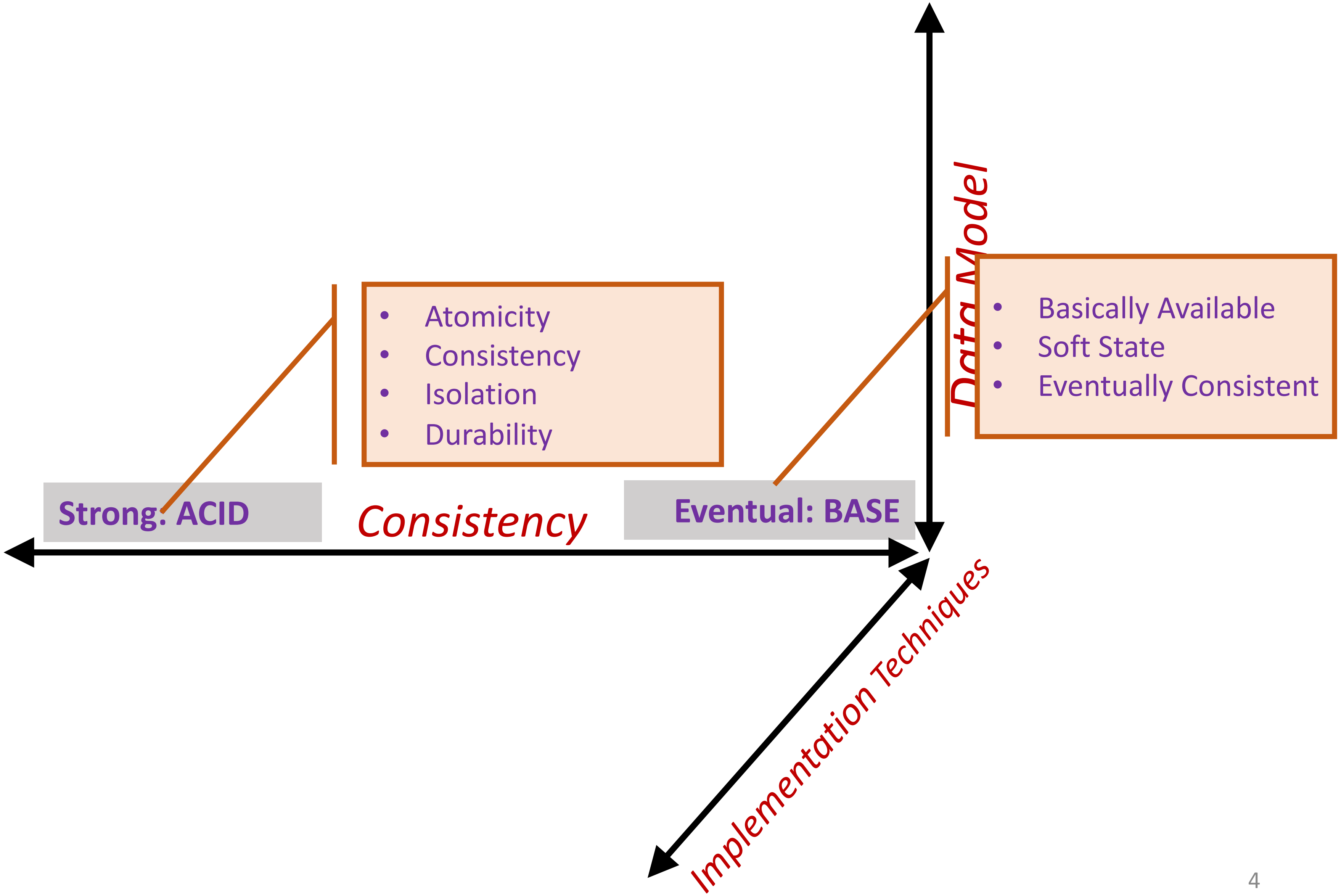
(Yet) Another Framework



(Yet) Another Framework



(Yet) Another Framework



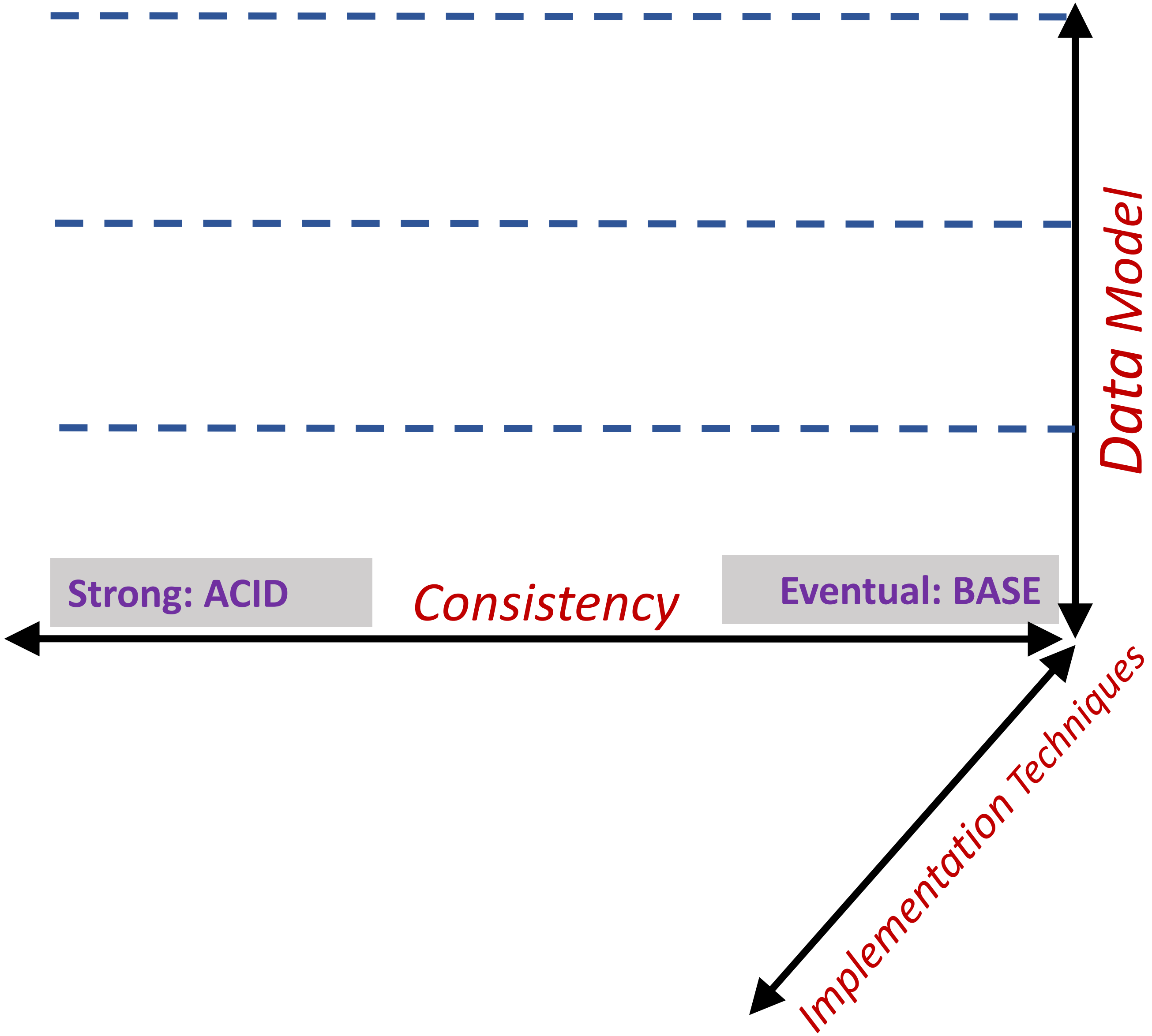
Strong: ACID

- Atomicity
- Consistency
- Isolation
- Durability

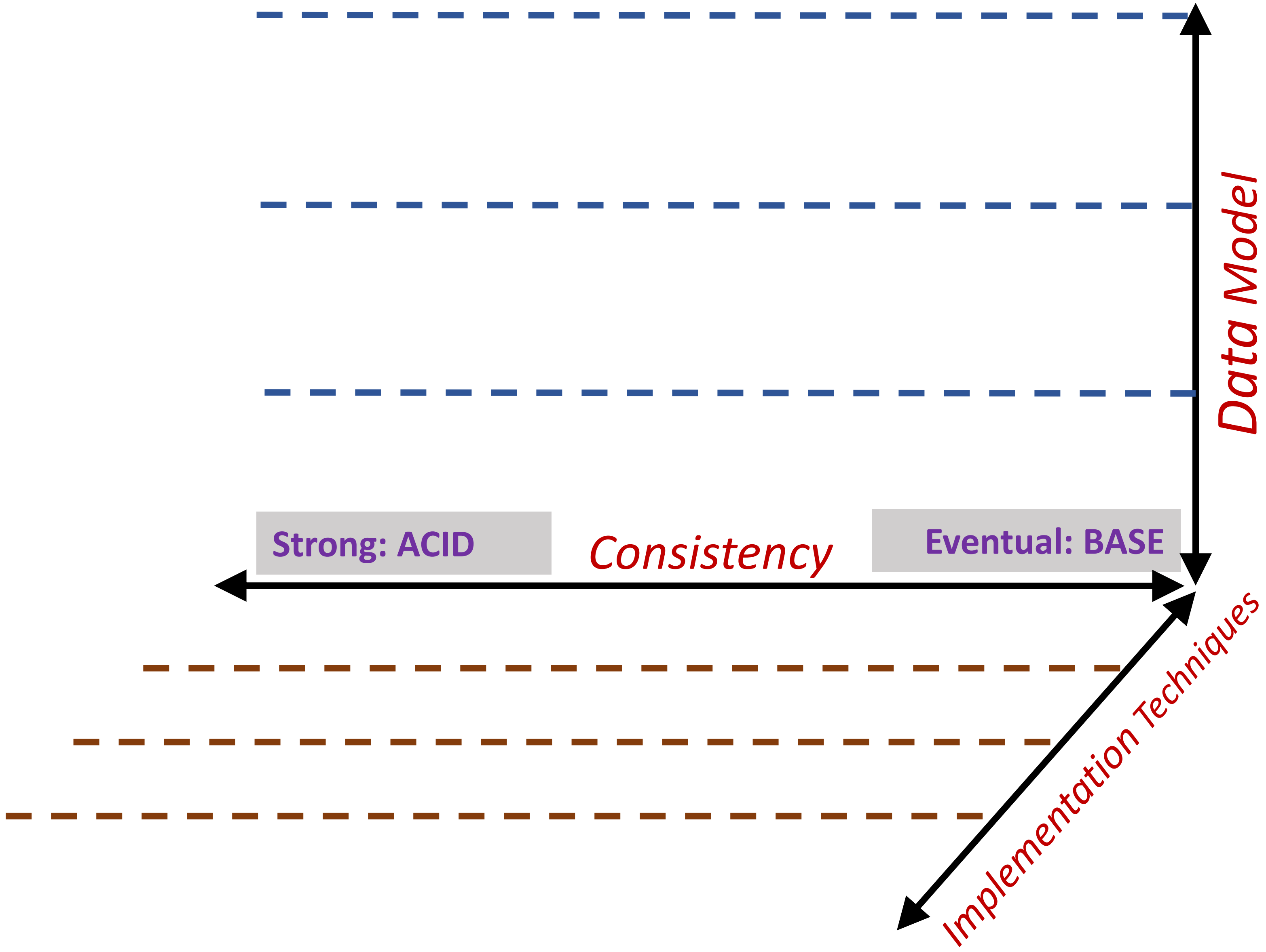
Eventual: BASE

- Basically Available
- Soft State
- Eventually Consistent

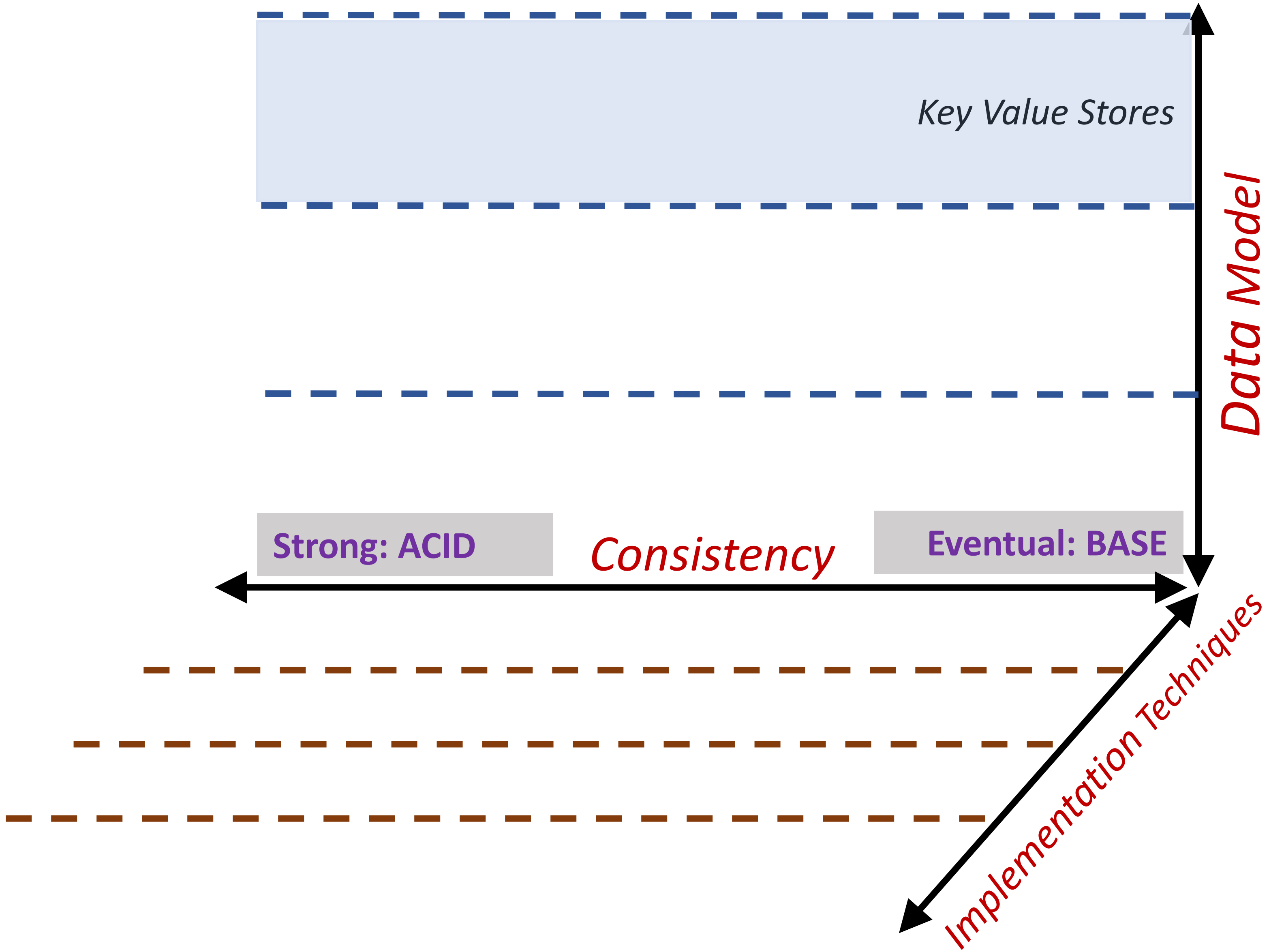
(Yet) Another Framework



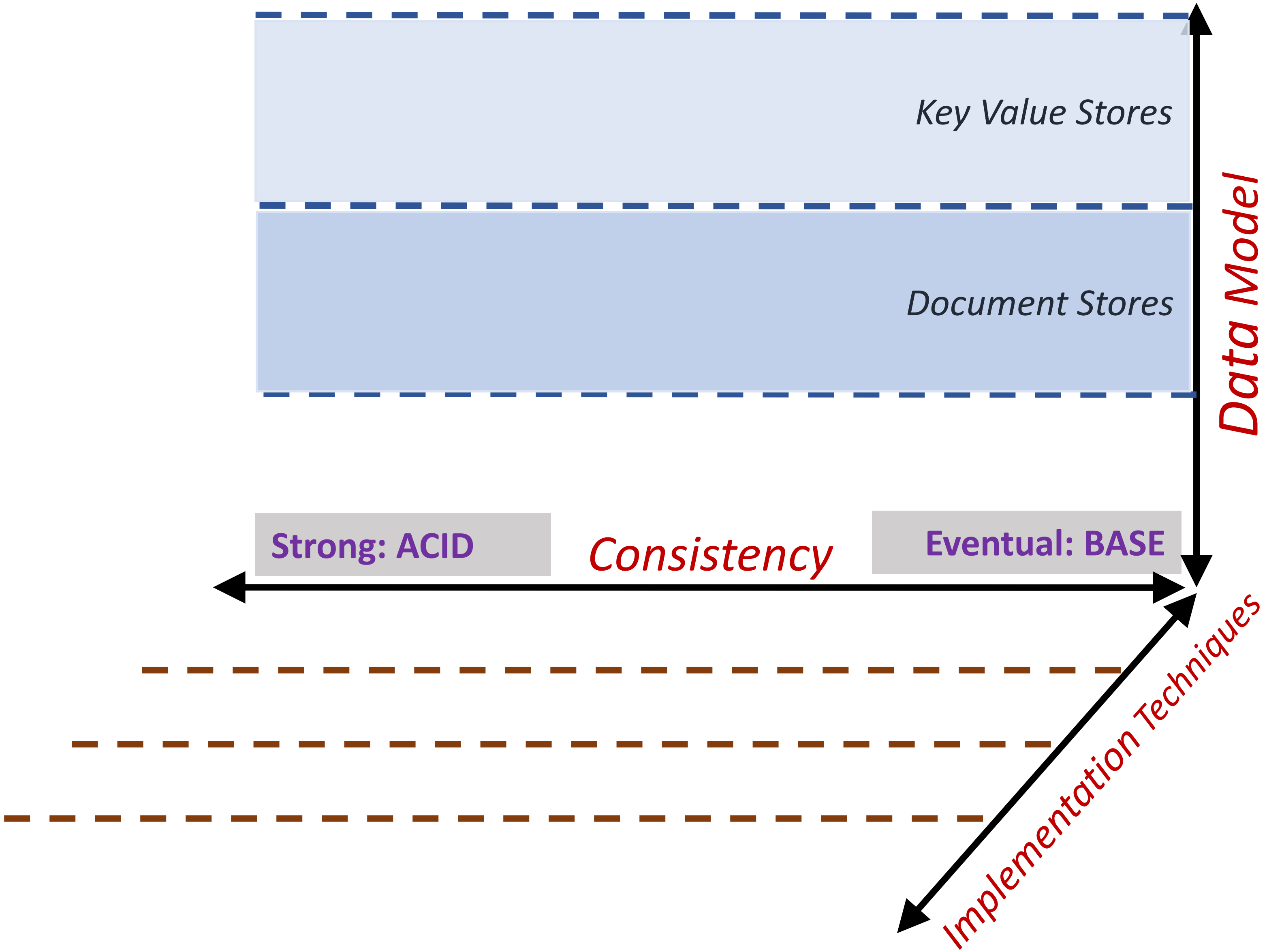
(Yet) Another Framework



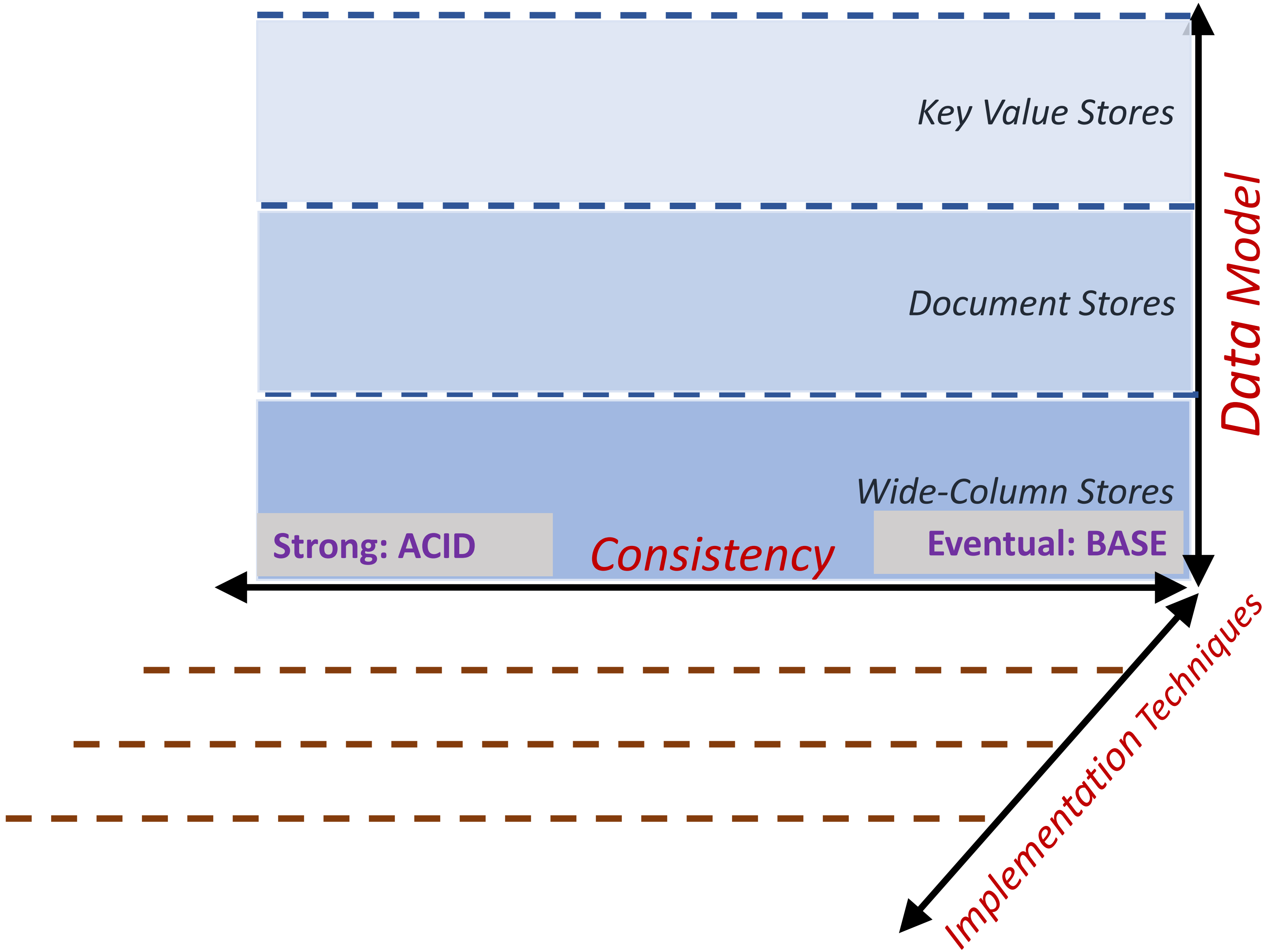
(Yet) Another Framework



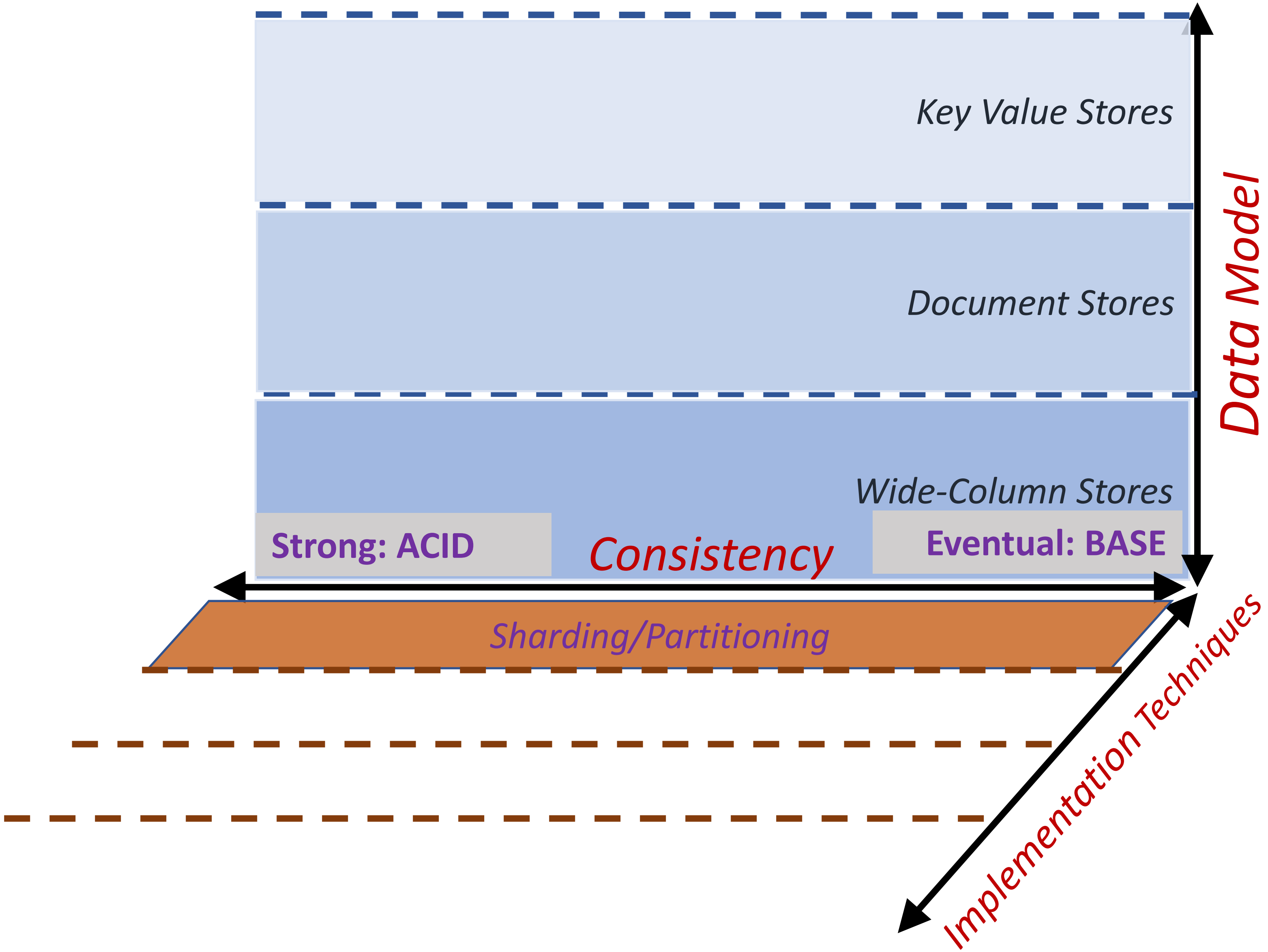
(Yet) Another Framework



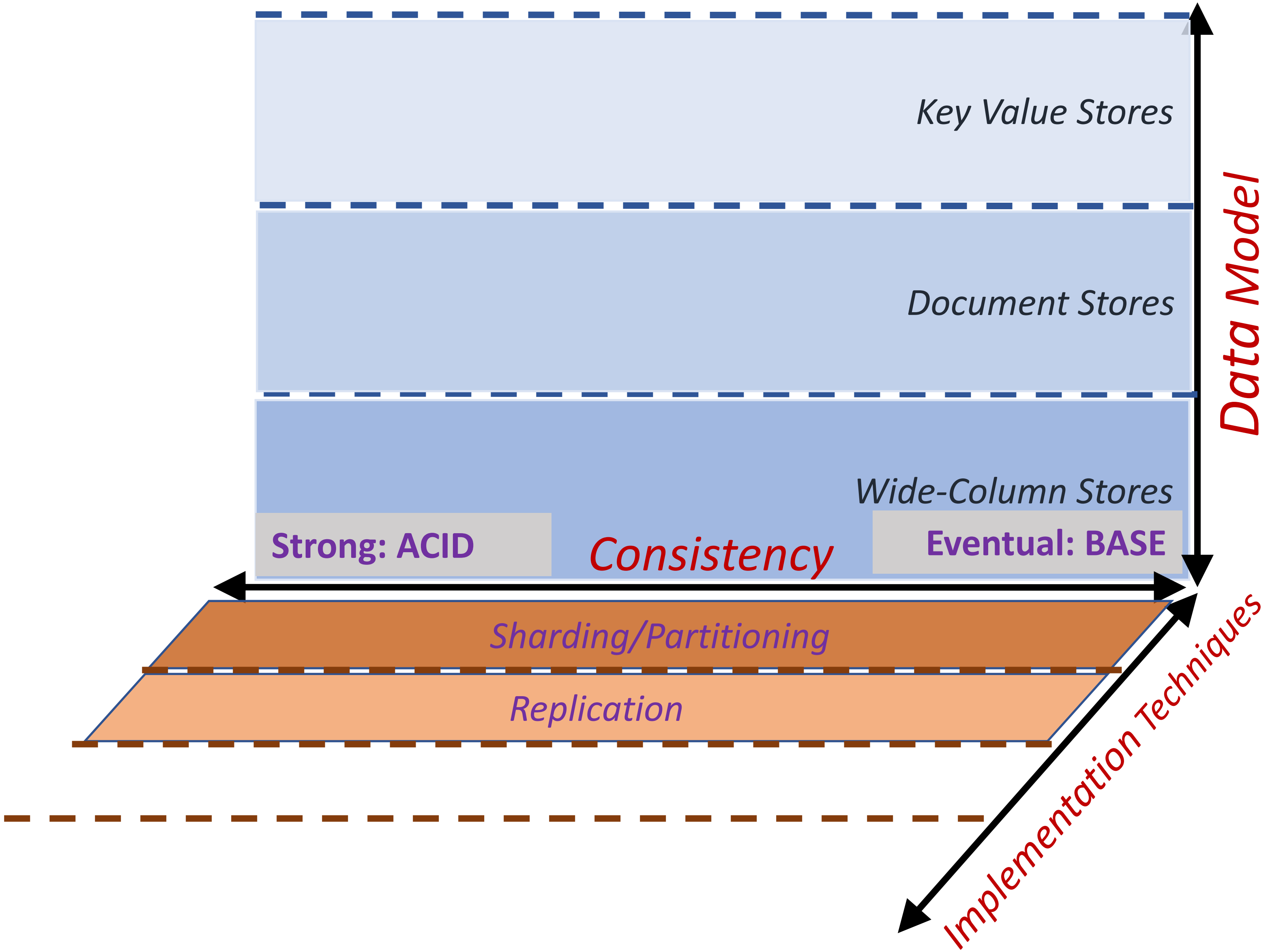
(Yet) Another Framework



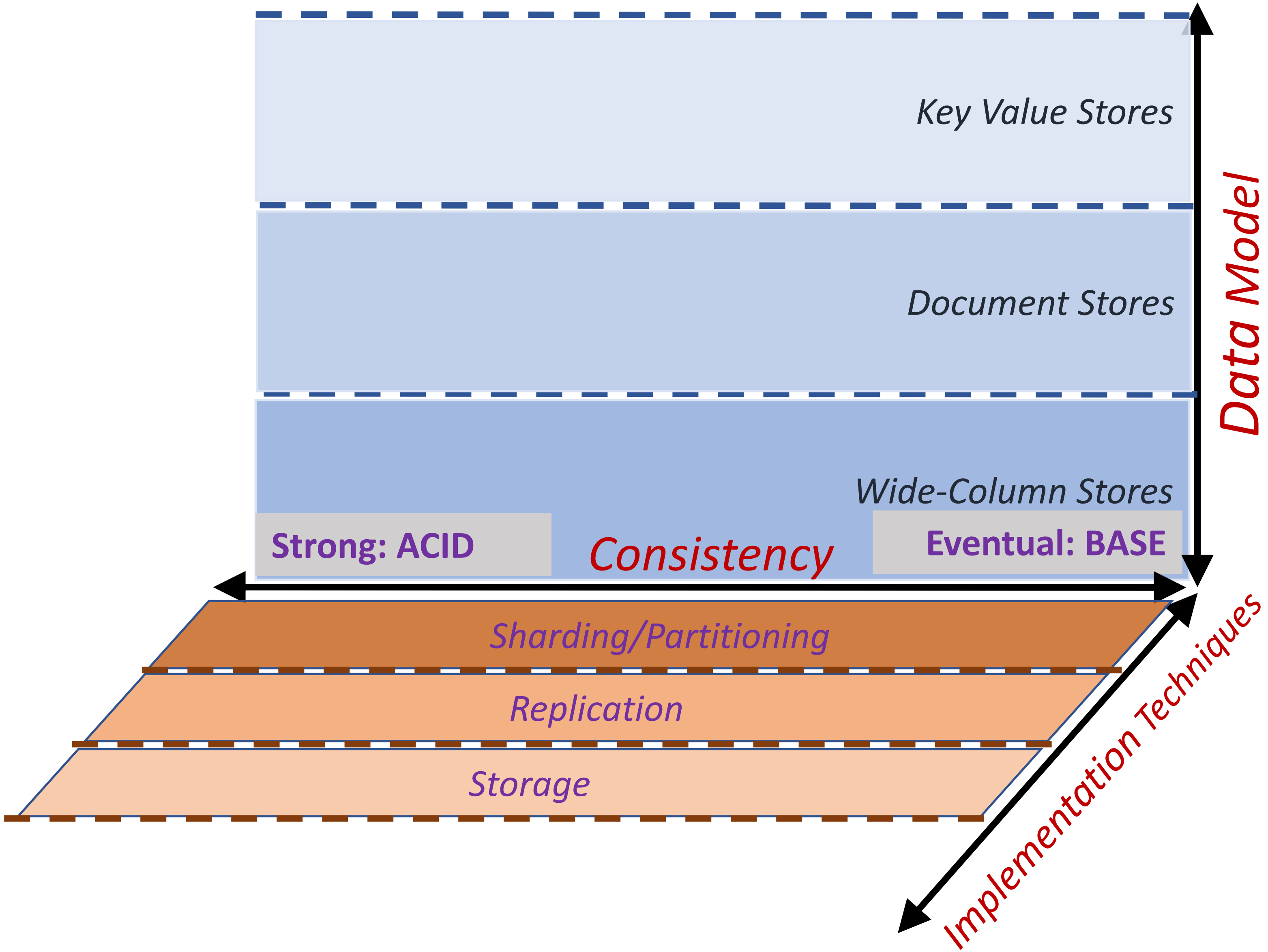
(Yet) Another Framework



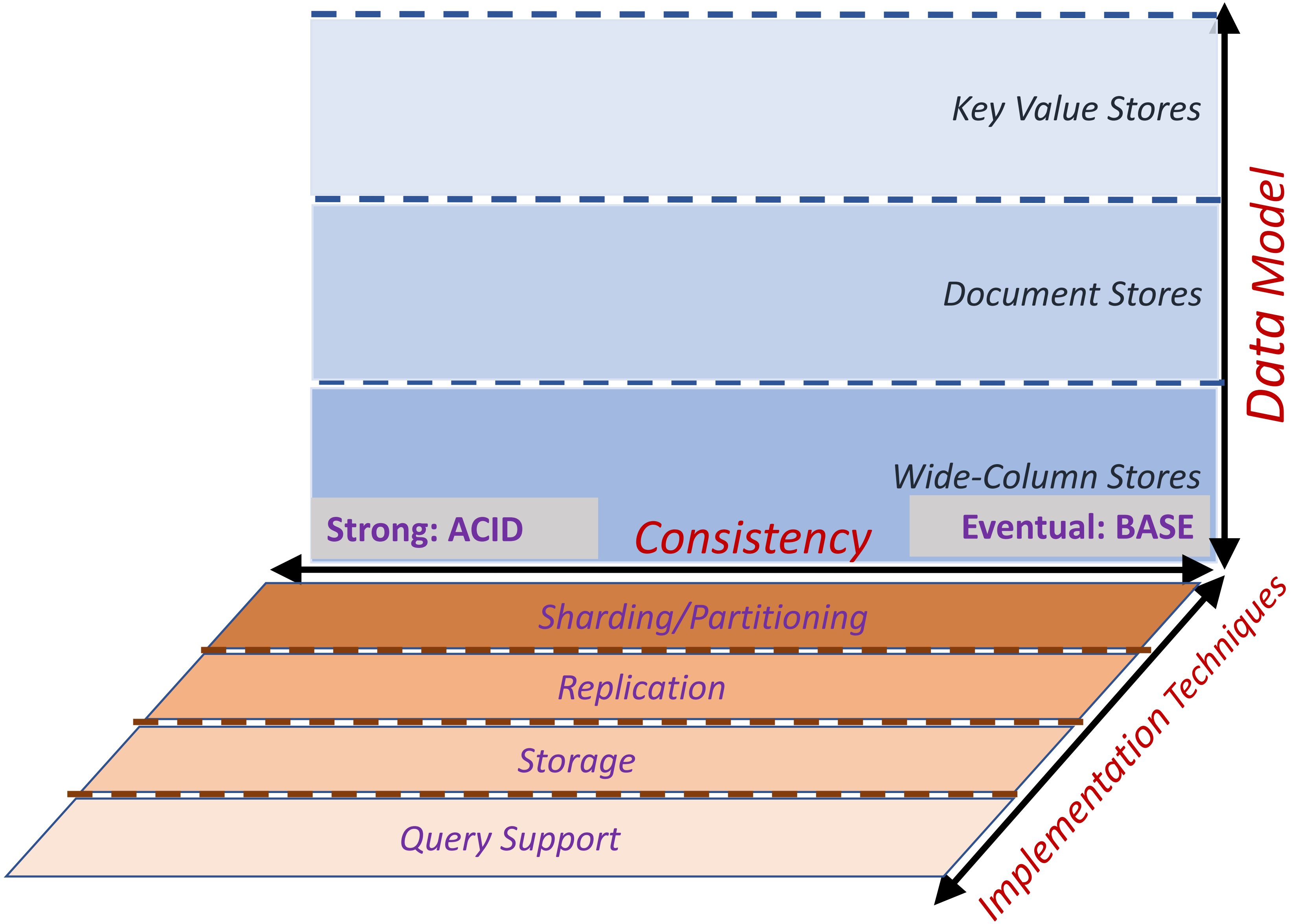
(Yet) Another Framework



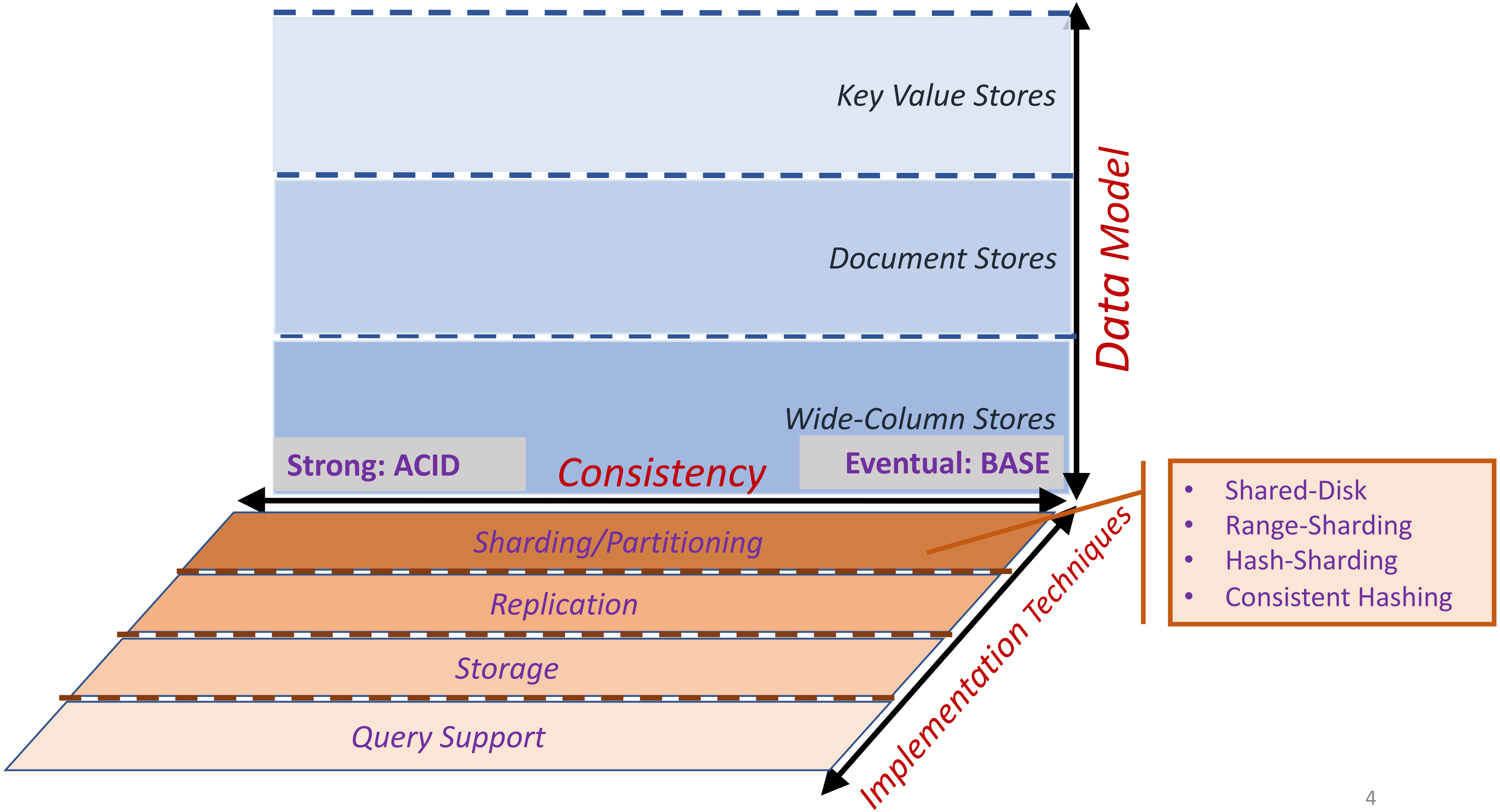
(Yet) Another Framework



(Yet) Another Framework

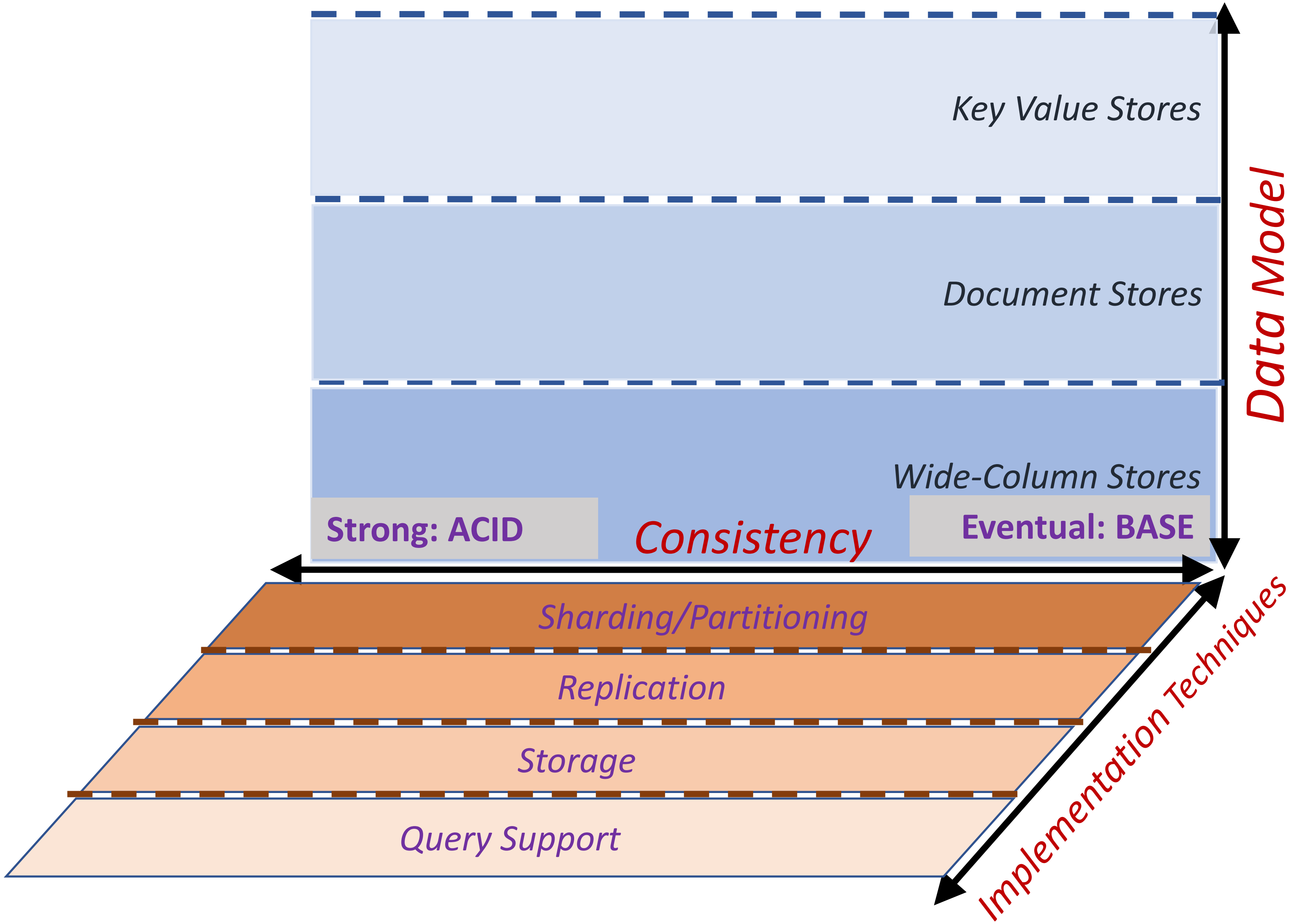


(Yet) Another Framework

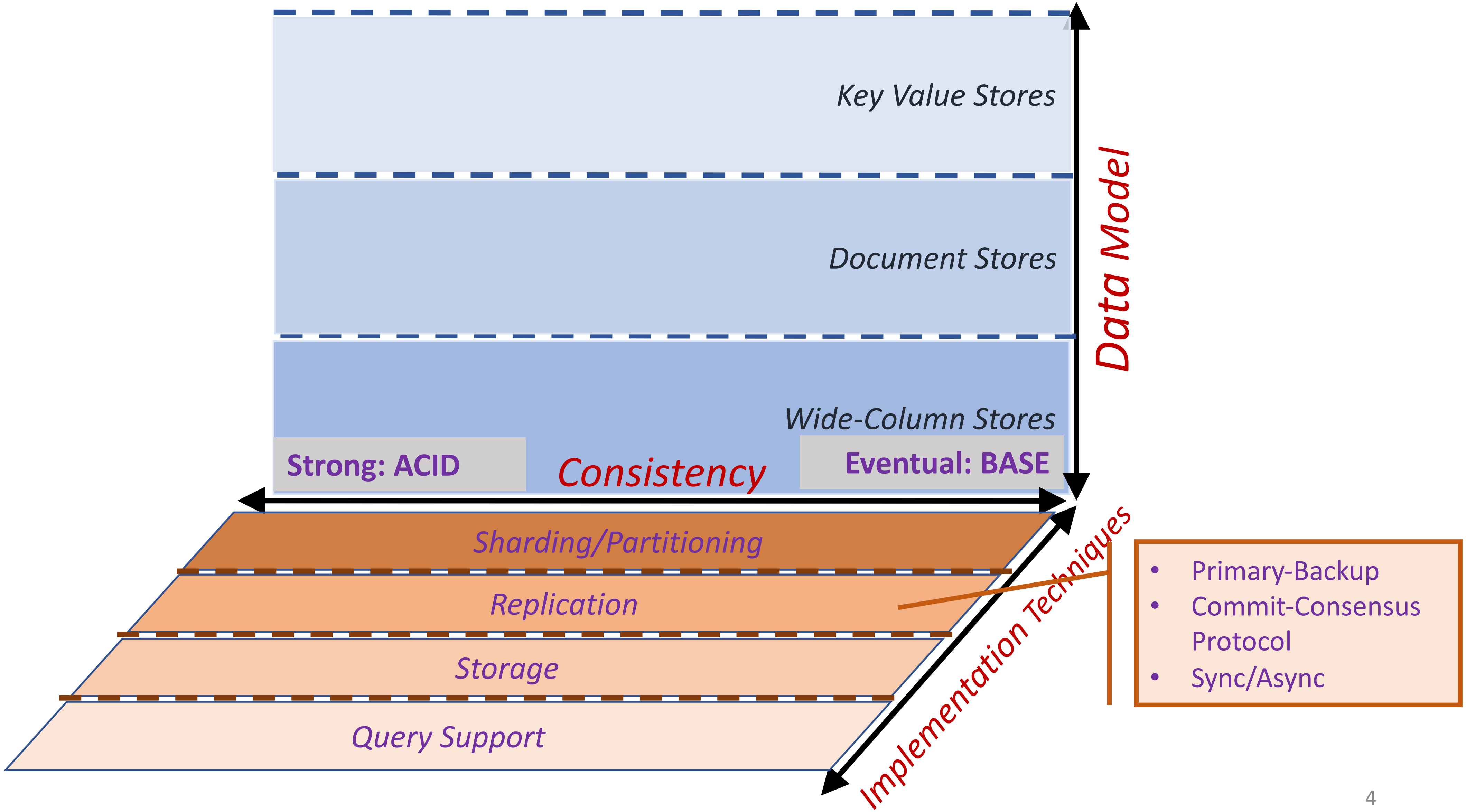


- Shared-Disk
- Range-Sharding
- Hash-Sharding
- Consistent Hashing

(Yet) Another Framework

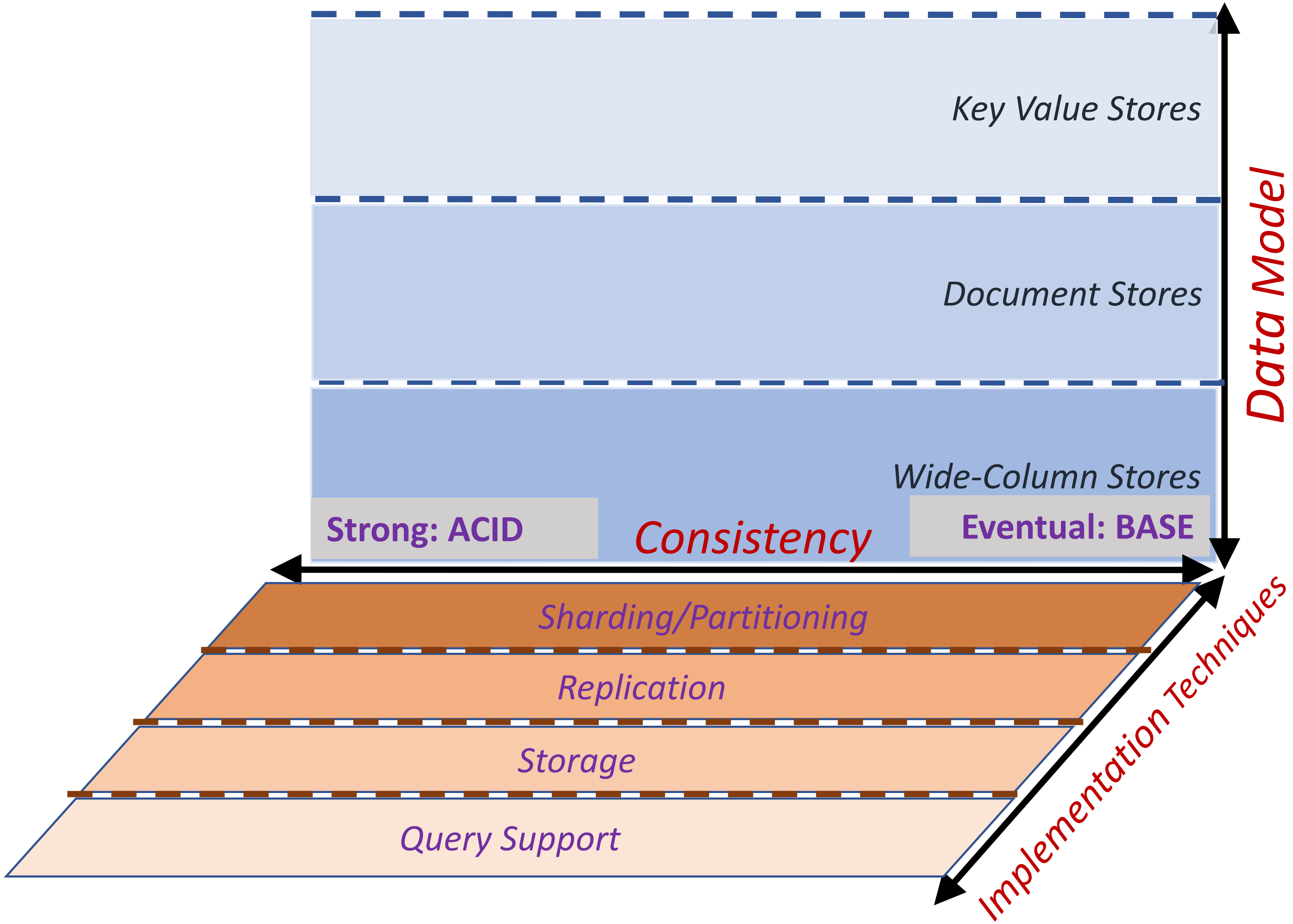


(Yet) Another Framework

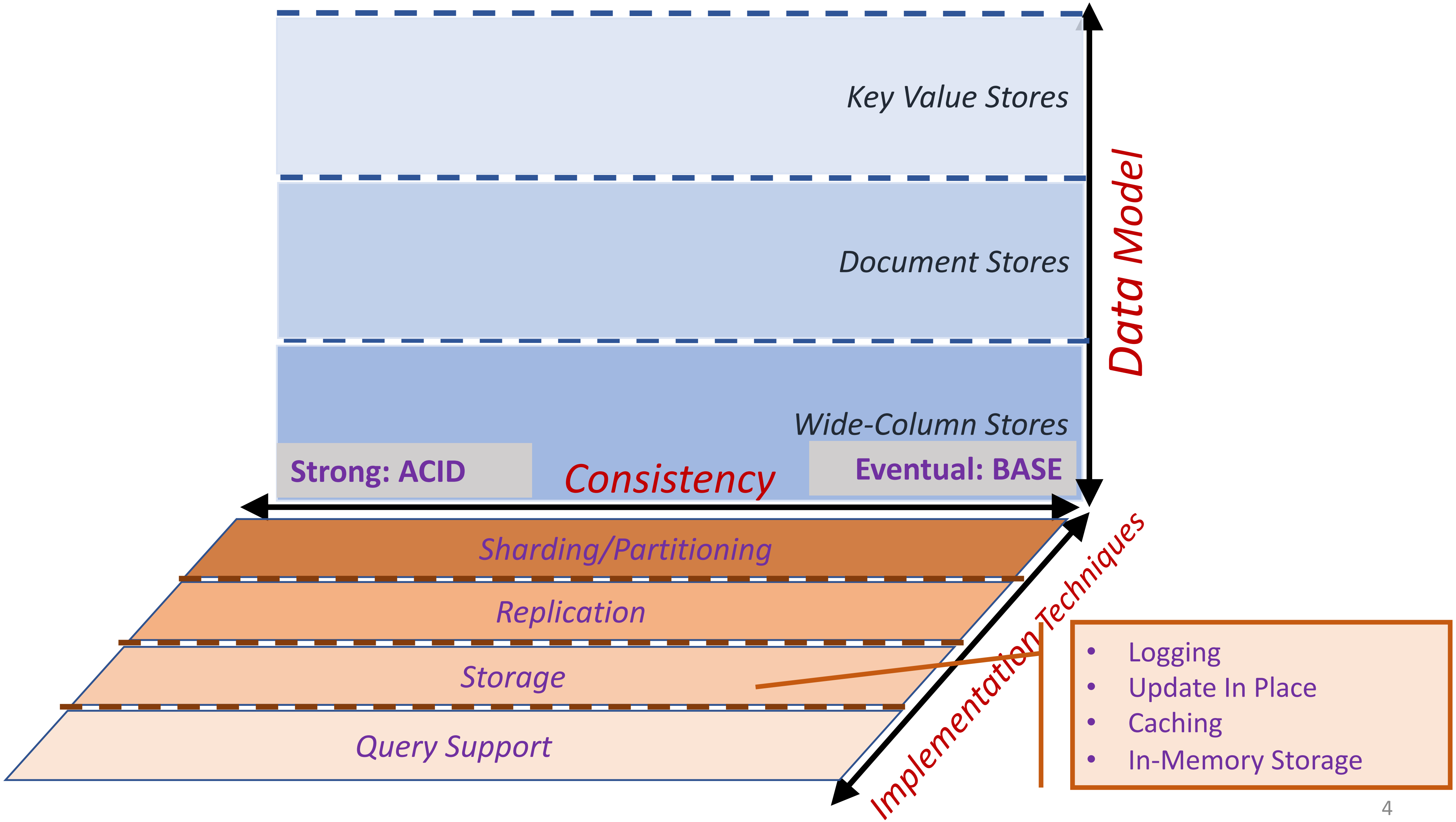


- Primary-Backup
- Commit-Consensus Protocol
- Sync/Async

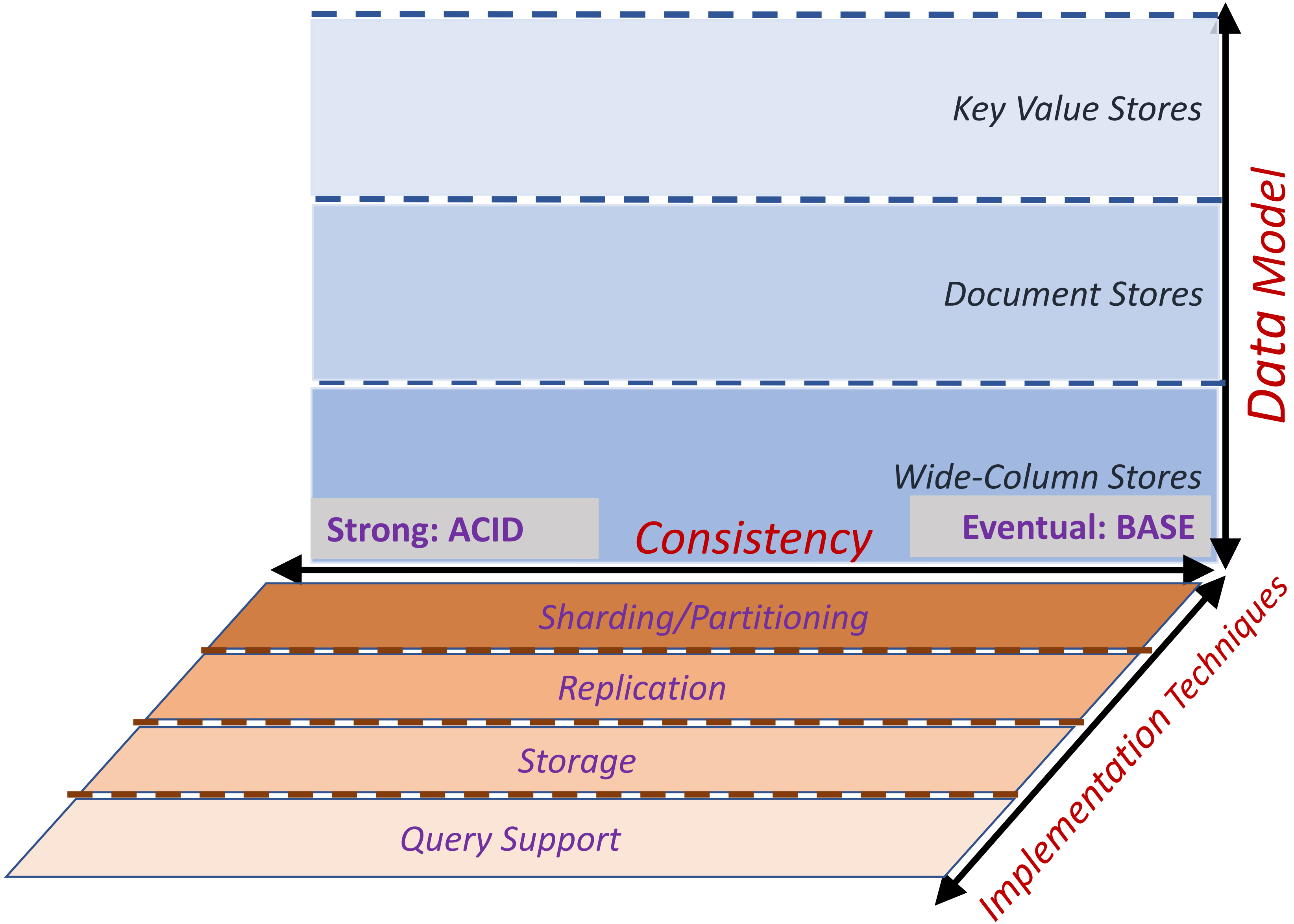
(Yet) Another Framework



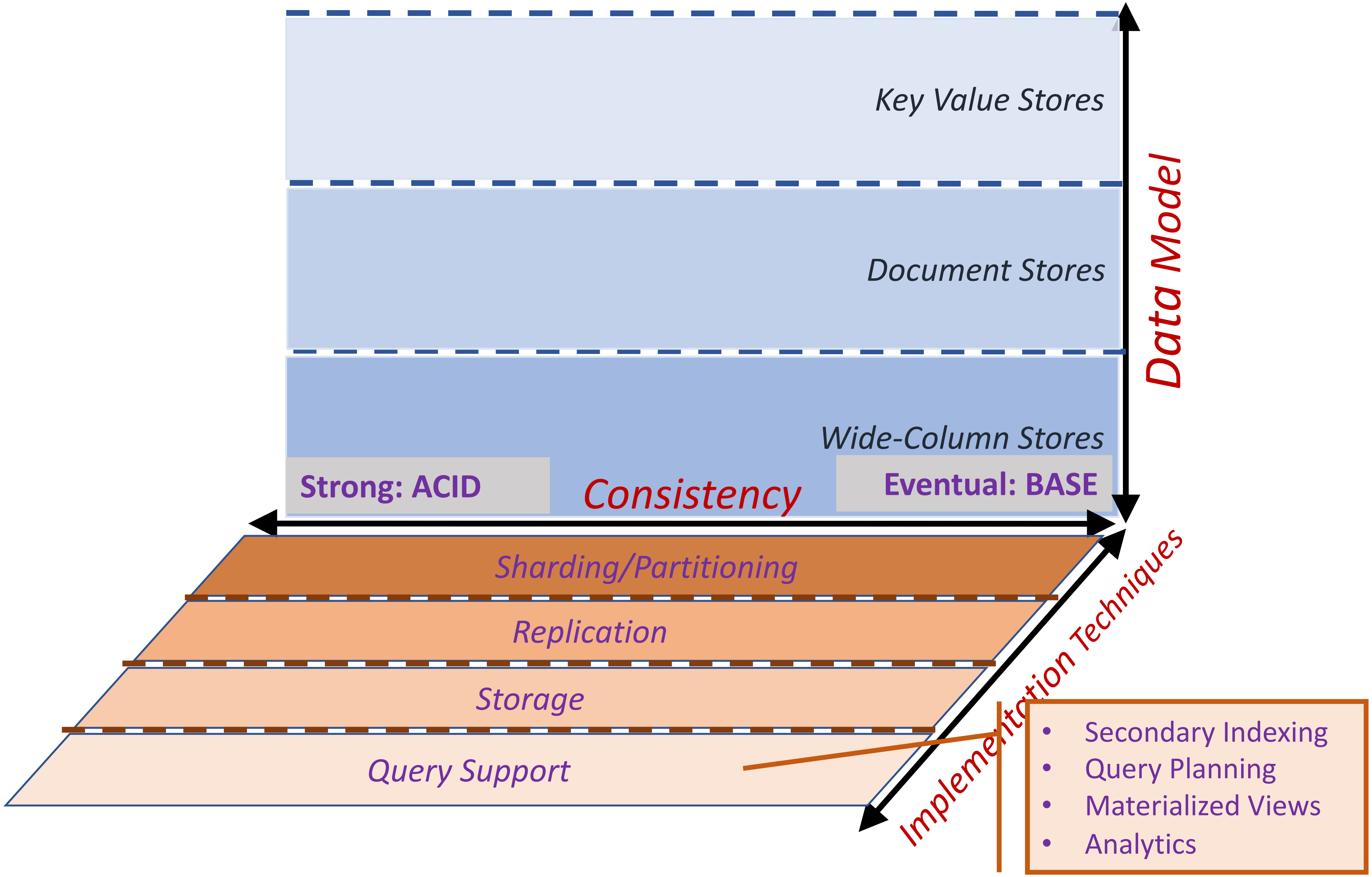
(Yet) Another Framework



(Yet) Another Framework

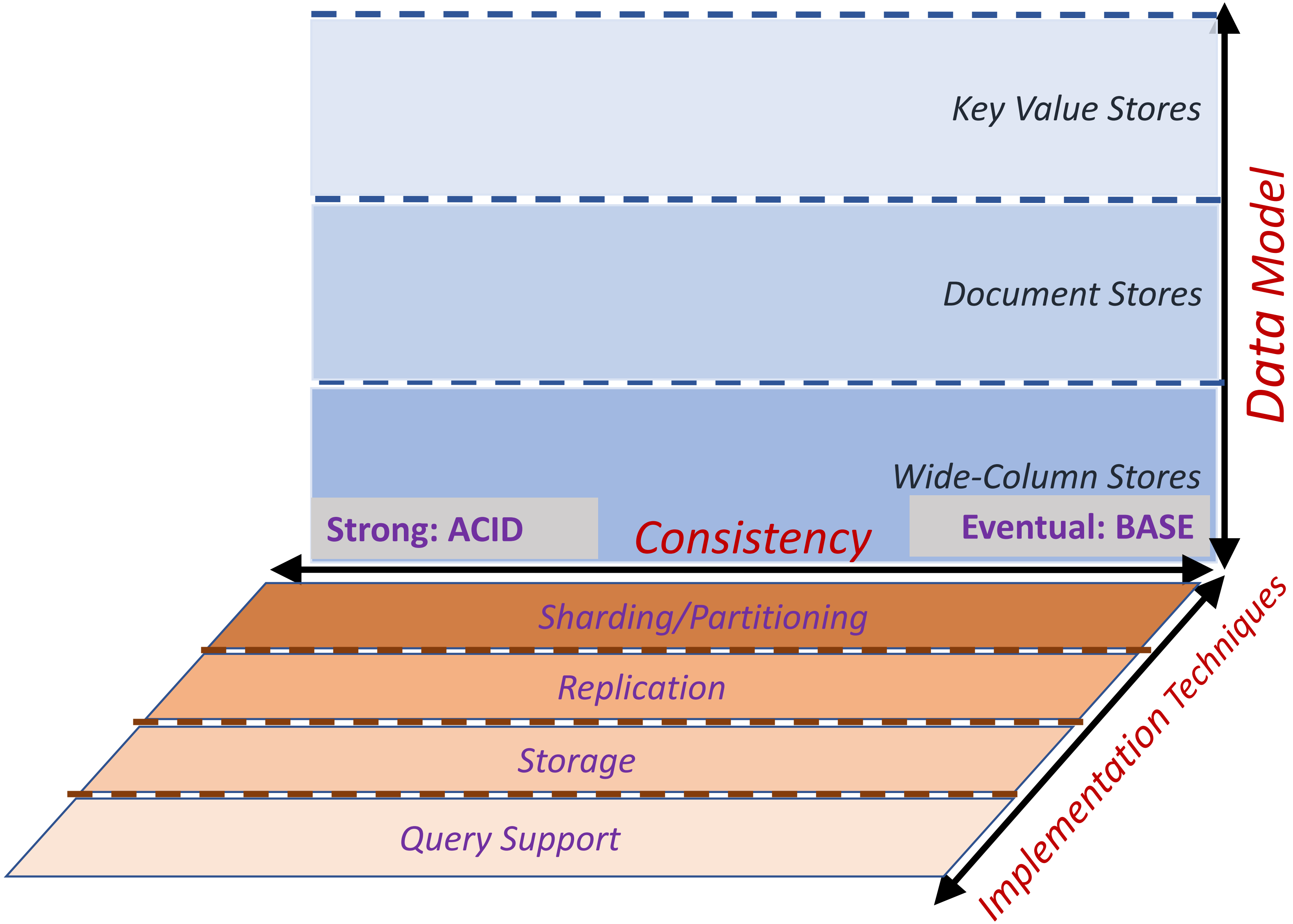


(Yet) Another Framework



- Secondary Indexing
- Query Planning
- Materialized Views
- Analytics

(Yet) Another Framework



(Yet) Another Framework

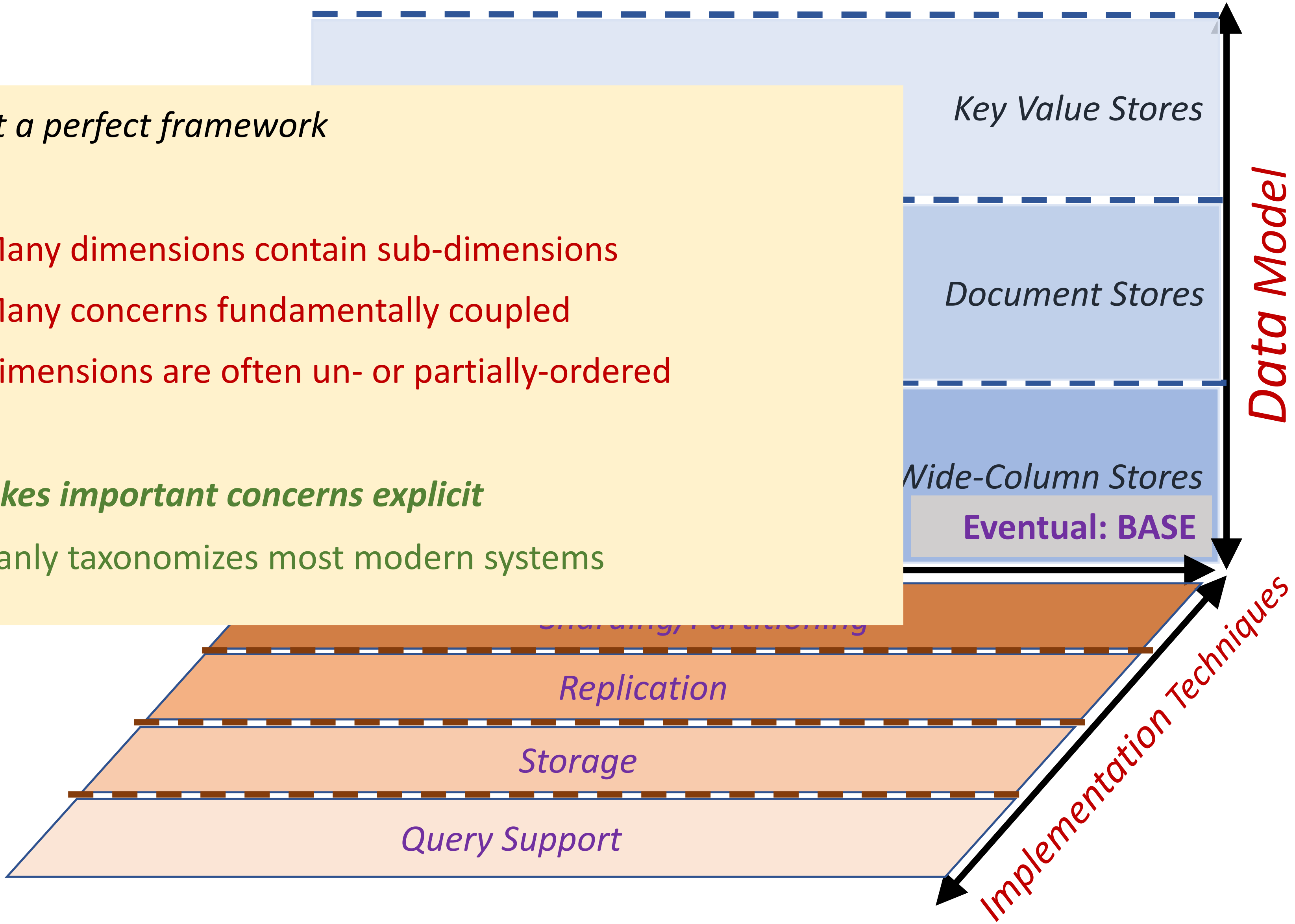
Still not a perfect framework

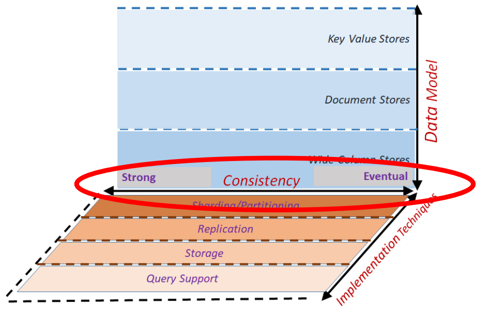
Cons:

- Many dimensions contain sub-dimensions
- Many concerns fundamentally coupled
- Dimensions are often un- or partially-ordered

Pros:

- ***Makes important concerns explicit***
- Cleanly taxonomizes most modern systems

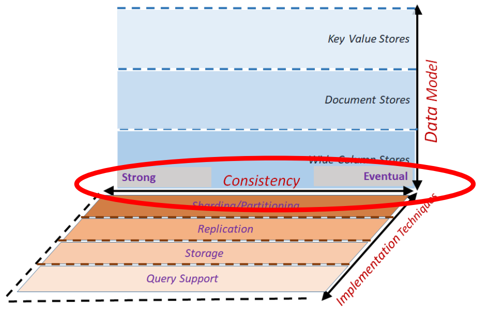


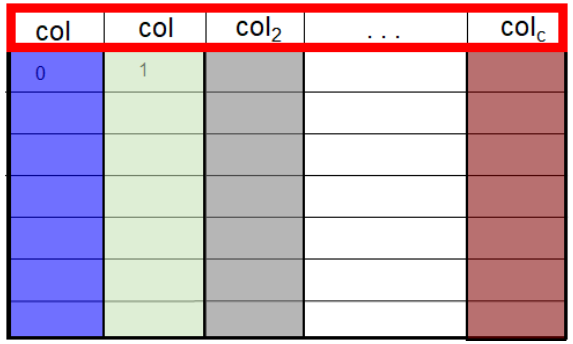


Consistency

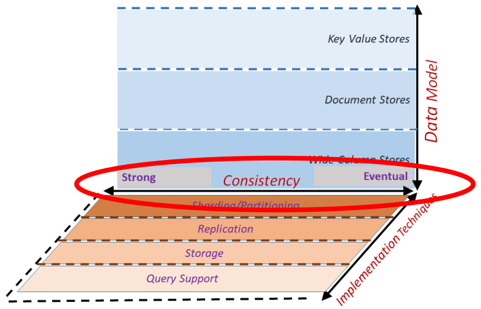
col	col	col_2	. . .	col_c
0	1			

Consistency

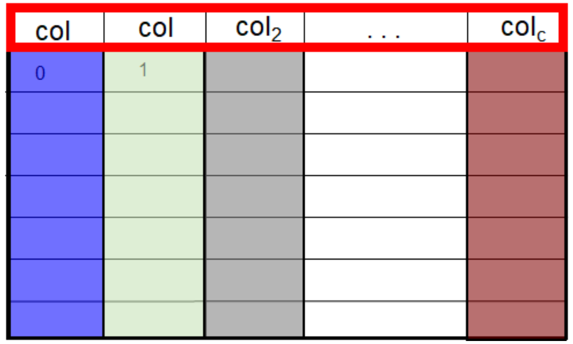




How to keep data in sync?



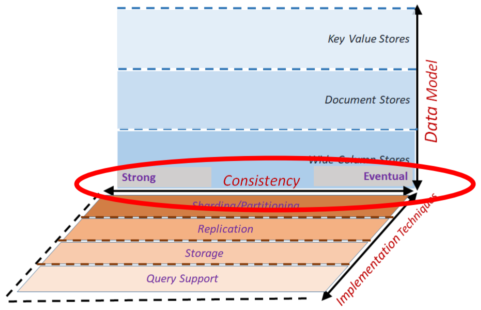
Consistency

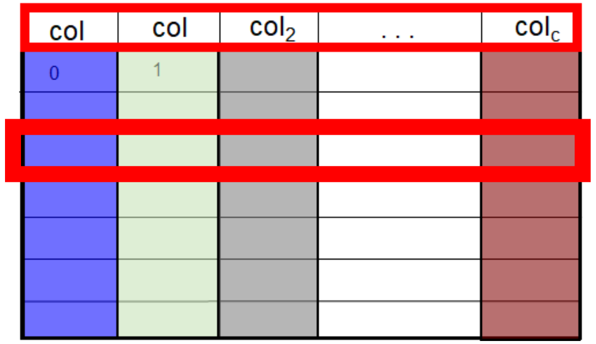


How to keep data in sync?

- Partitioning → single row spread over multiple machines

Consistency

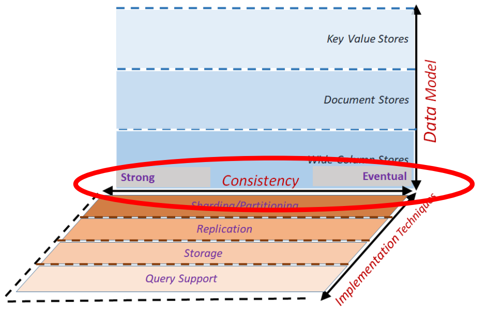


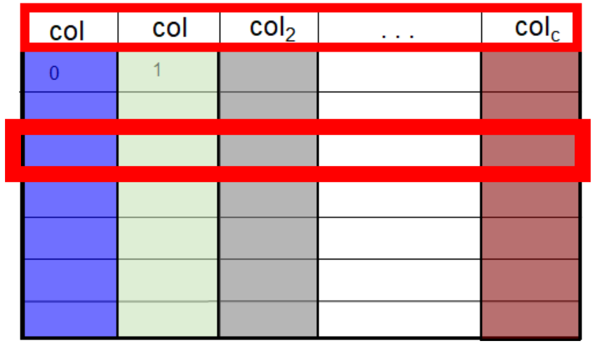


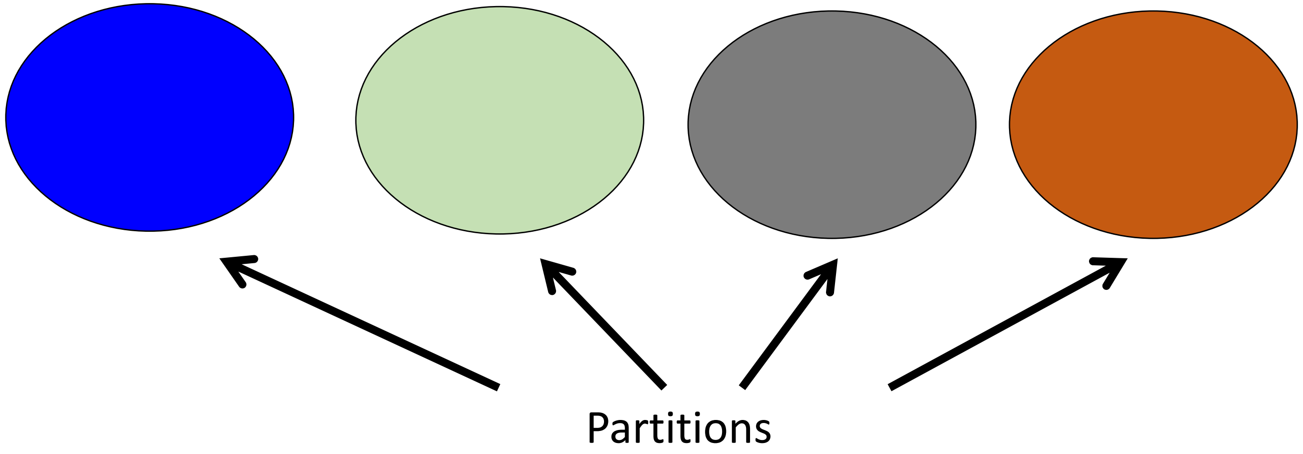
How to keep data in sync?

- Partitioning → single row spread over multiple machines

Consistency



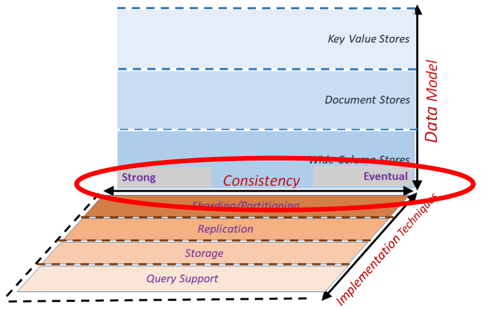


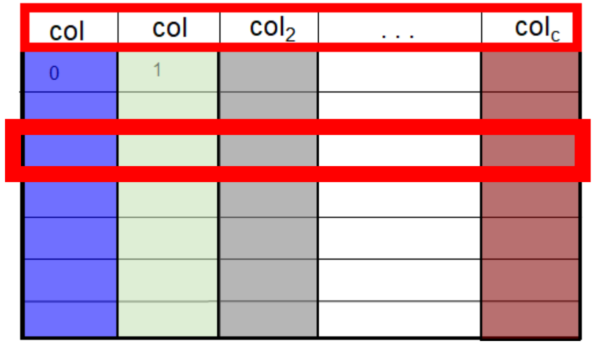


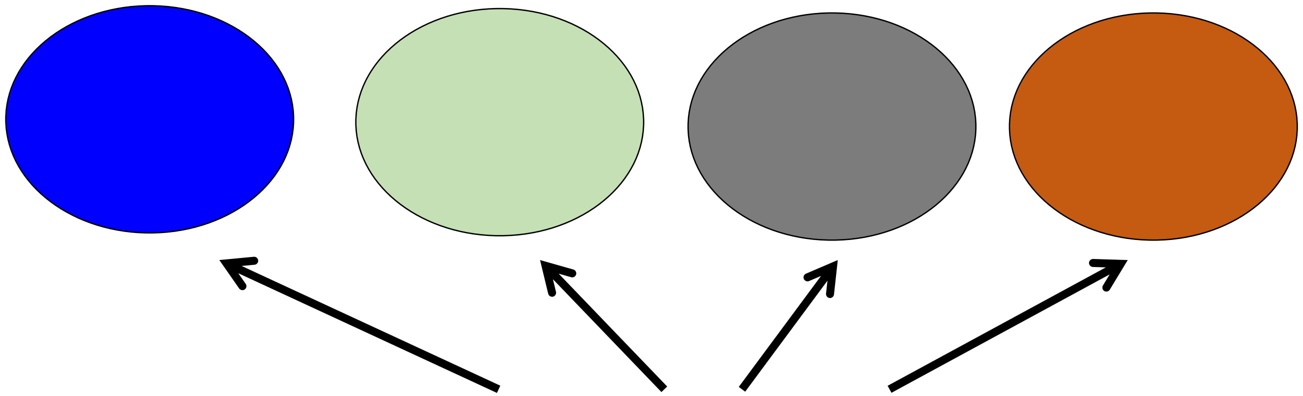
How to keep data in sync?

- Partitioning → single row spread over multiple machines

Consistency

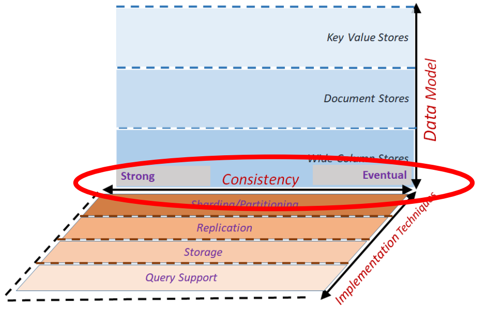




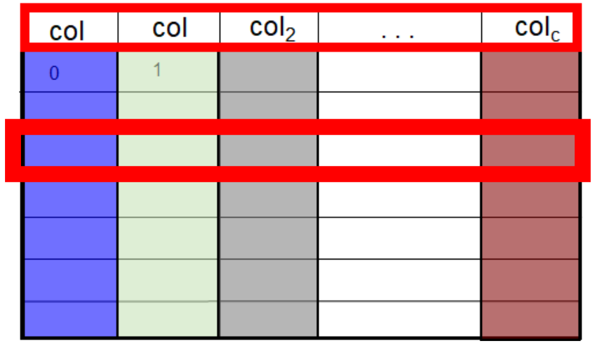


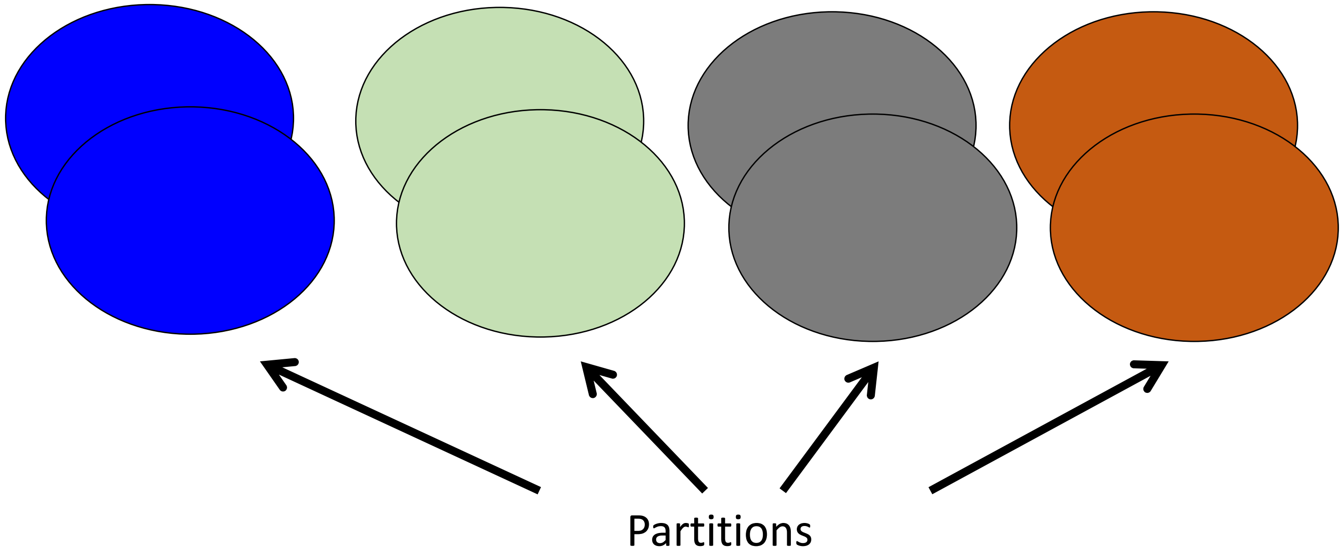
How to keep data in sync?

- Partitioning → single row spread over multiple machines
- Redundancy → single datum spread over multiple machines



Consistency

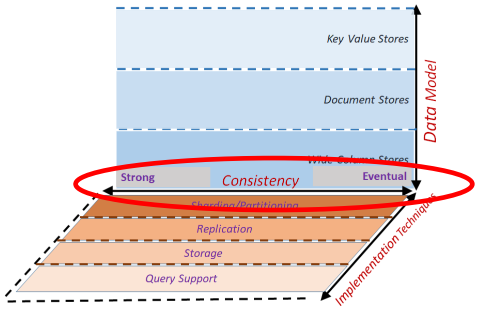


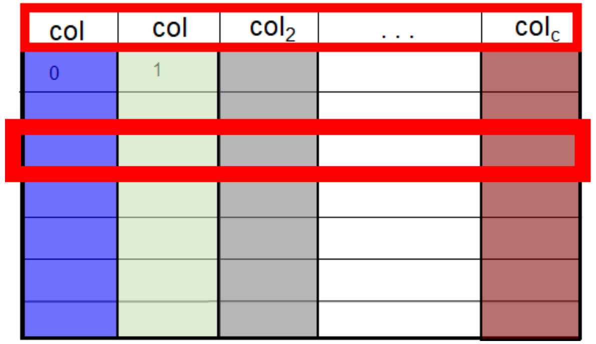


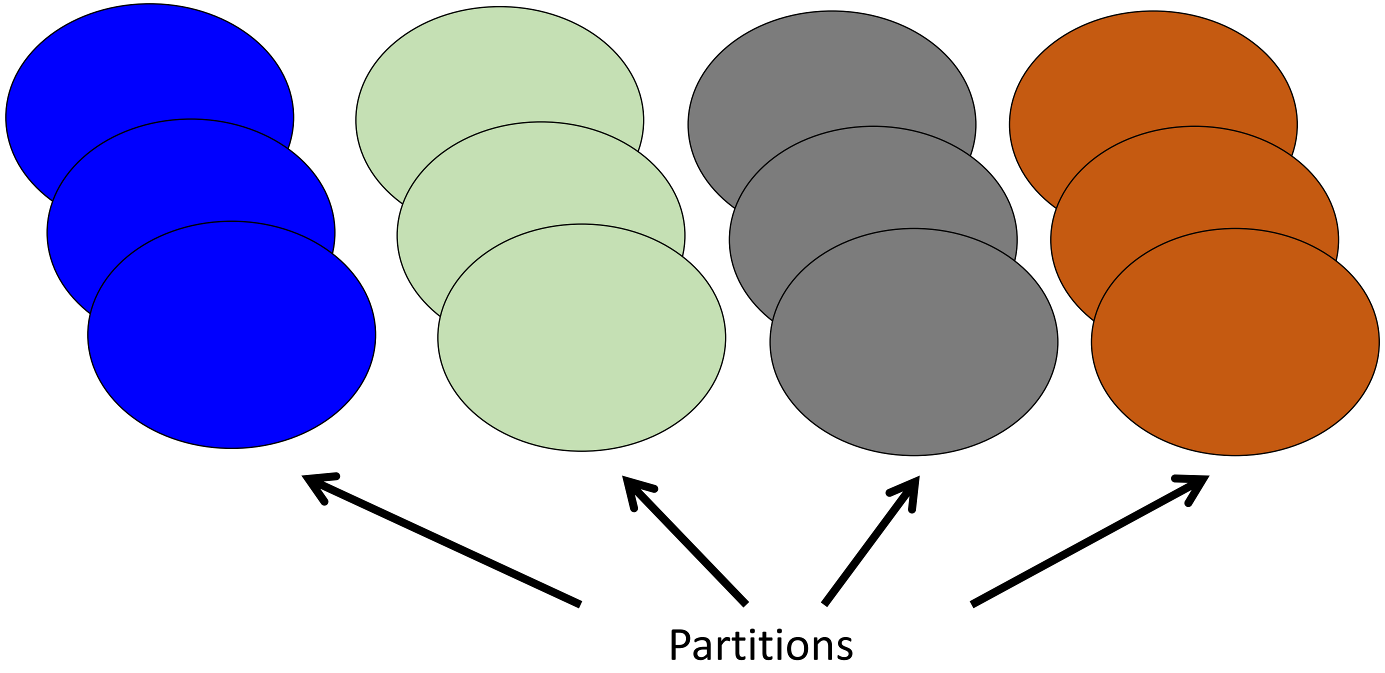
How to keep data in sync?

- Partitioning → single row spread over multiple machines
- Redundancy → single datum spread over multiple machines

Consistency

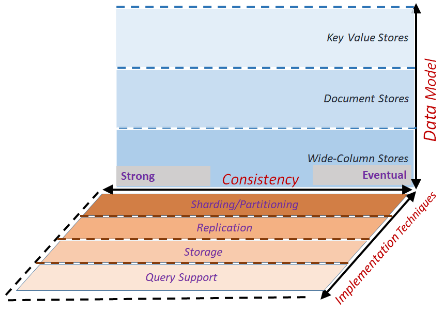




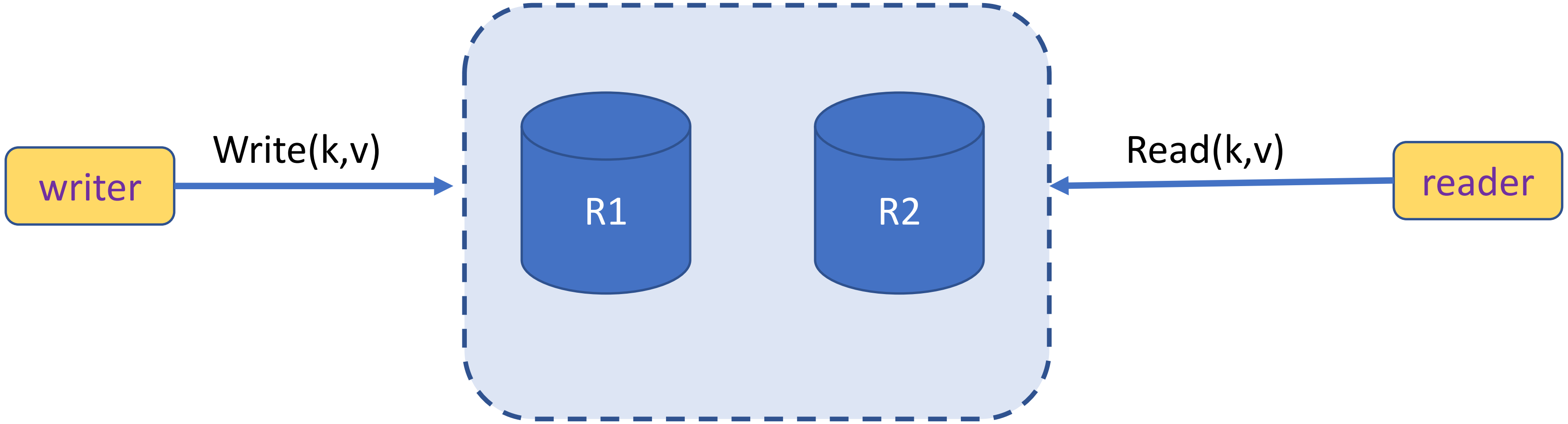


How to keep data in sync?

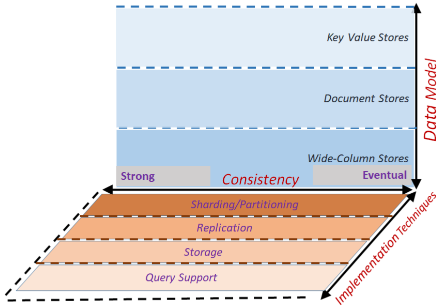
- Partitioning → single row spread over multiple machines
- Redundancy → single datum spread over multiple machines

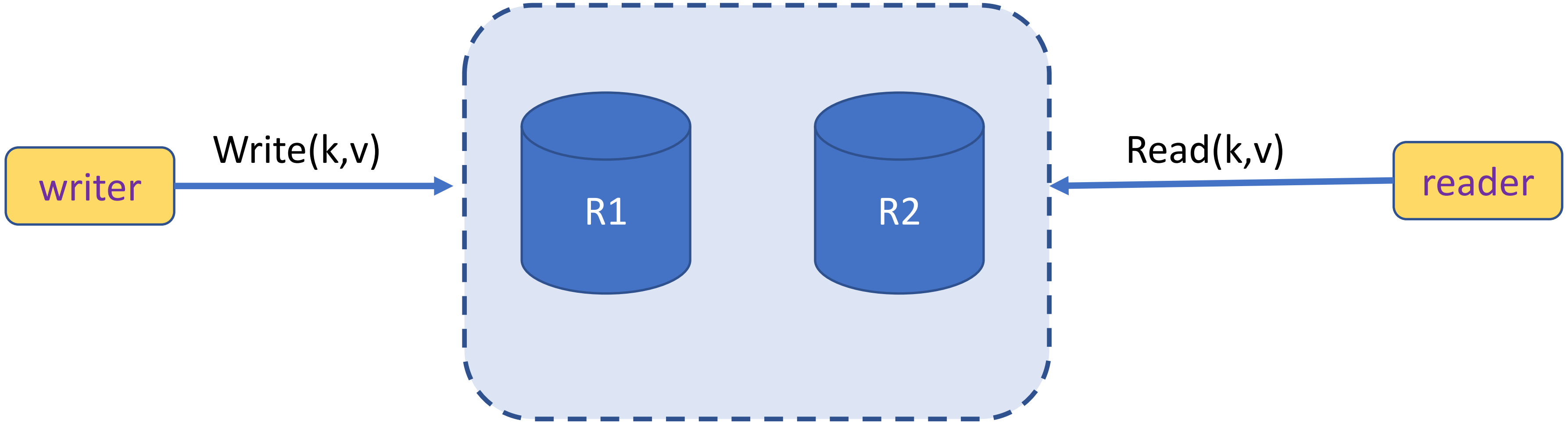


Consistency: the core problem

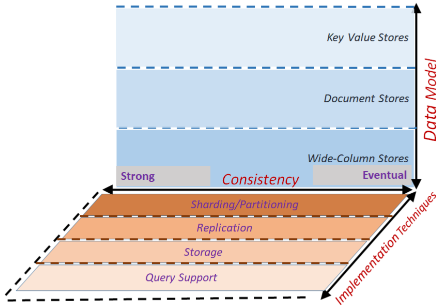


Consistency: the core problem

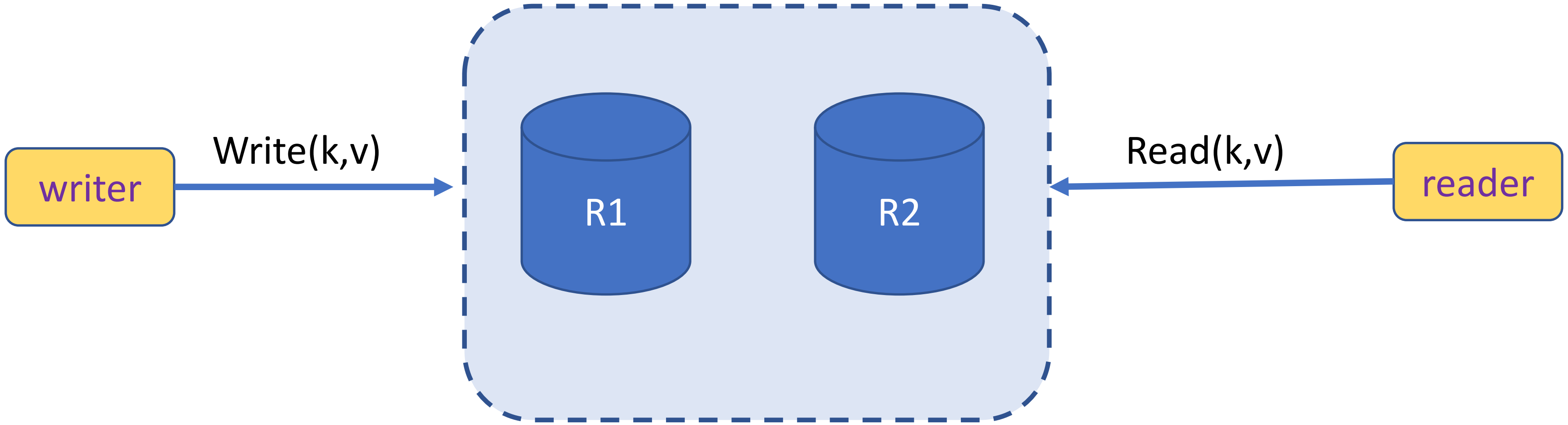




- Clients perform reads and writes

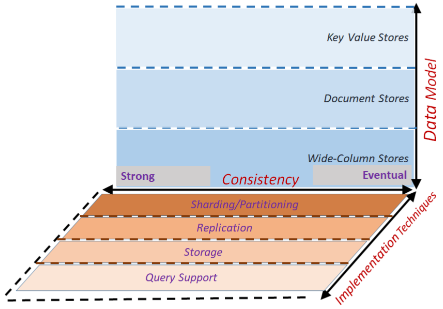


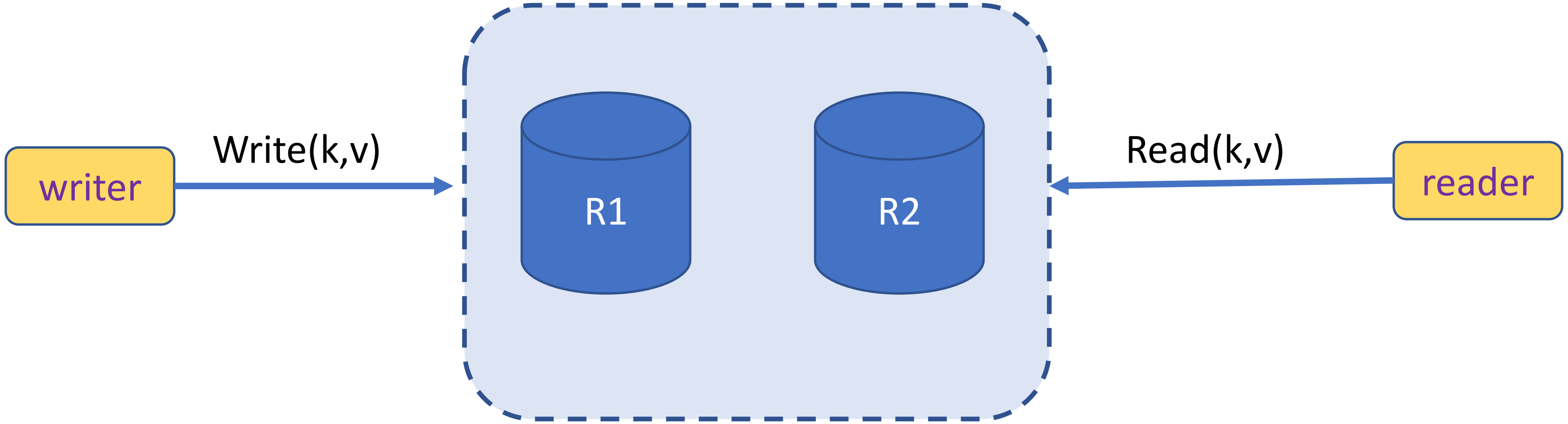
Consistency: the core problem



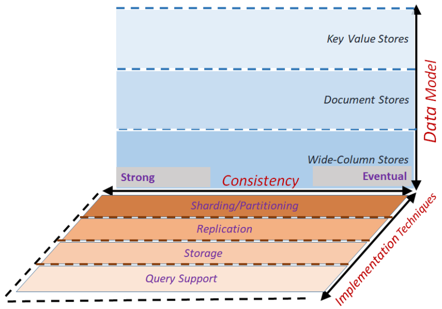
- Clients perform reads and writes
- Data is replicated among a set of servers

Consistency: the core problem

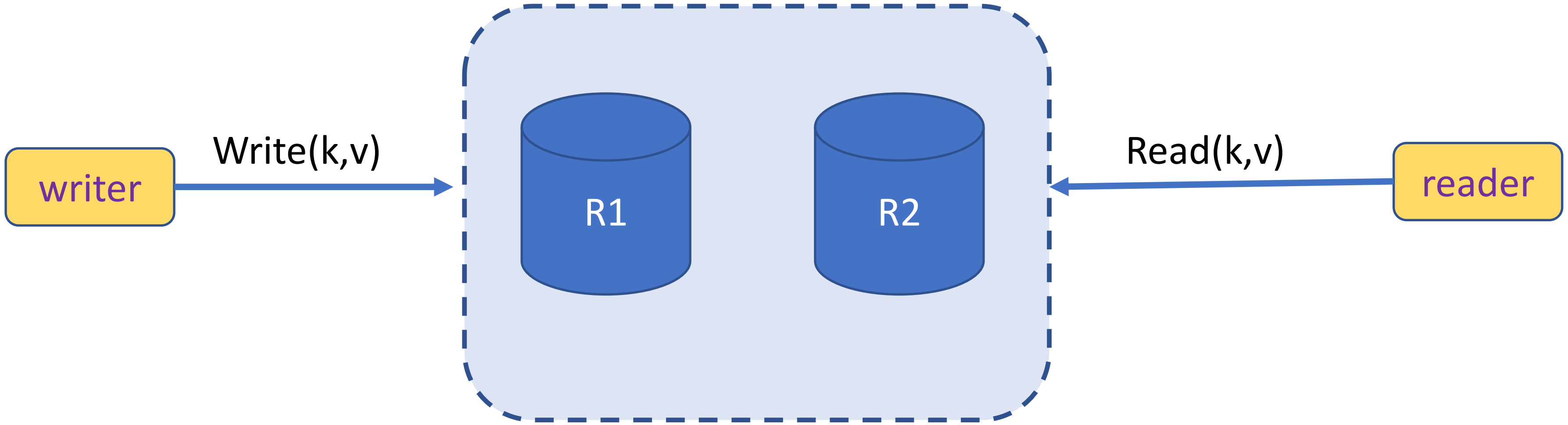




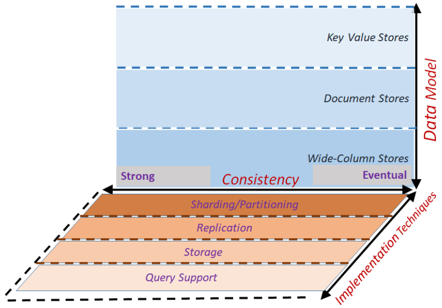
- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers



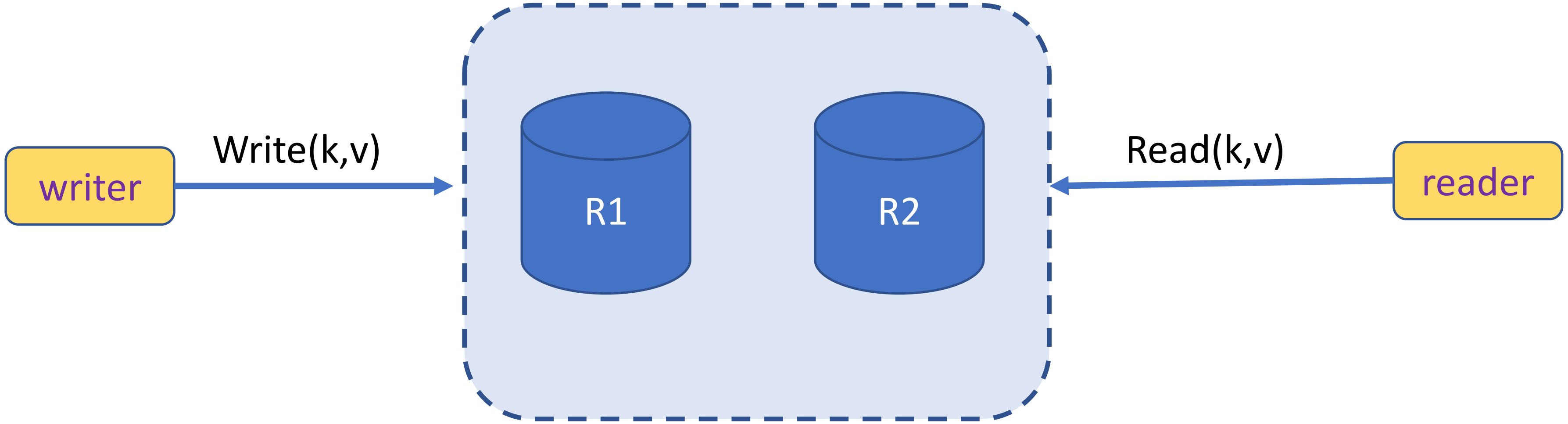
Consistency: the core problem



- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes

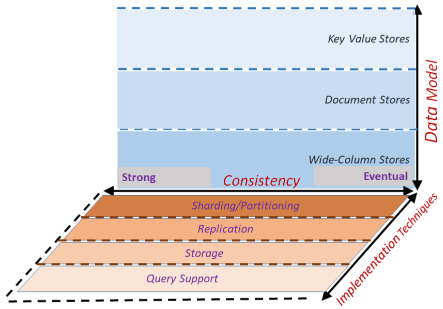


Consistency: the core problem

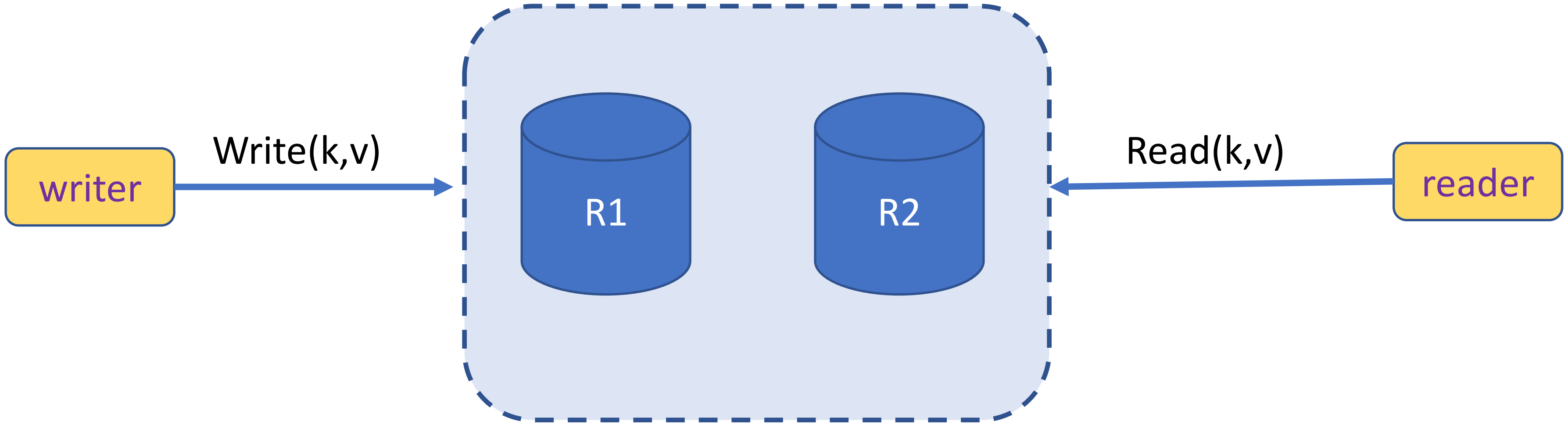


- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes

- How should we *implement* write?



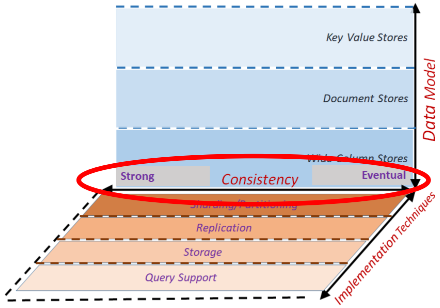
Consistency: the core problem



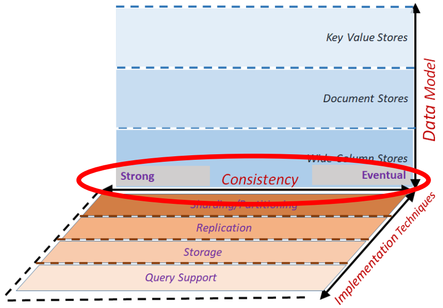
- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes

- How should we *implement* write?
- How to *implement* read?

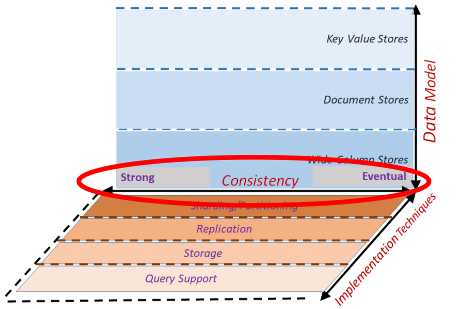
Consistency: CAP Theorem



Consistency: CAP Theorem



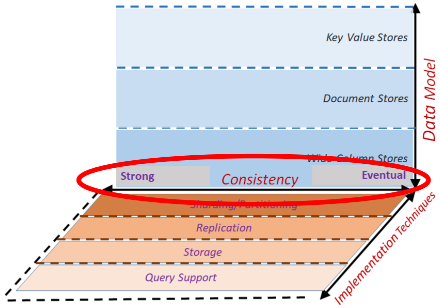
- A distributed system can satisfy at most 2/3 guarantees of:



Consistency: CAP Theorem

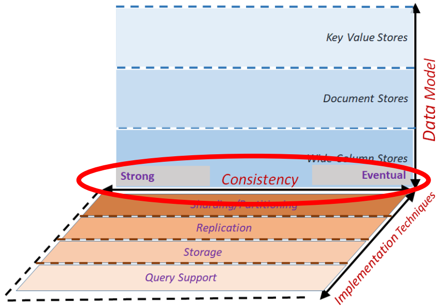
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:

Consistency: CAP Theorem



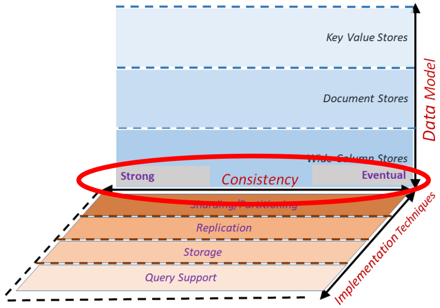
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client

Consistency: CAP Theorem



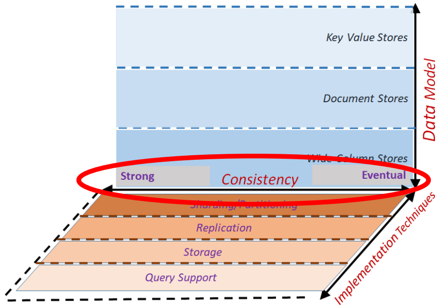
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:

Consistency: CAP Theorem



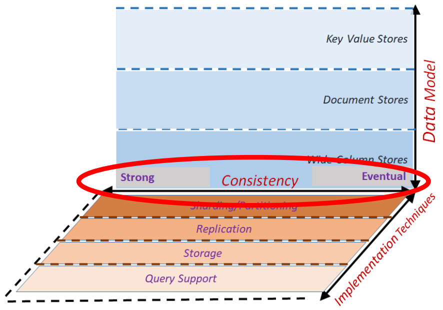
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly

Consistency: CAP Theorem



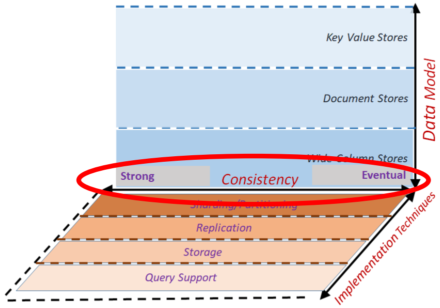
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**:

Consistency: CAP Theorem



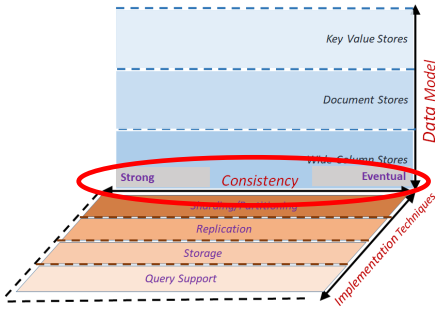
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**:
     - system continues to work in spite of network partitions

Consistency: CAP Theorem



- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**:
     - system continues to work in spite of netwo

Consistency: CAP Theorem



- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**:
     - system continues to work in spite of netwo

Why care about CAP Properties?

Availability

- Reads/writes complete reliably and quickly.
- E.g. Amazon, each ms latency → $6M yearly loss.

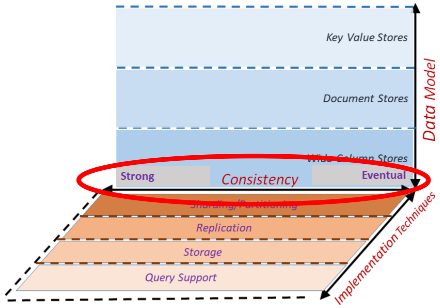
Partitions

- Internet router outages
- Under-sea cables cut
- rack switch outage
- *system should continue functioning normally!*

Consistency

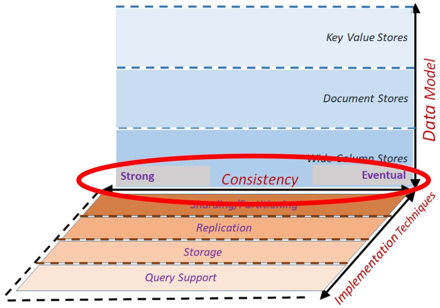
- all nodes see same data at any time, or reads return latest written value by any client.
- ***This basically means correctness!***

Consistency: CAP Theorem



- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**:
     - system continues to work in spite of netwo

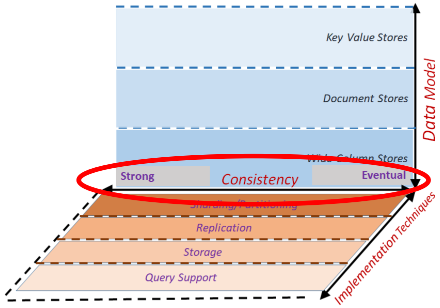
Consistency: CAP Theorem



- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**:
     - system continues to work in spite of netwo

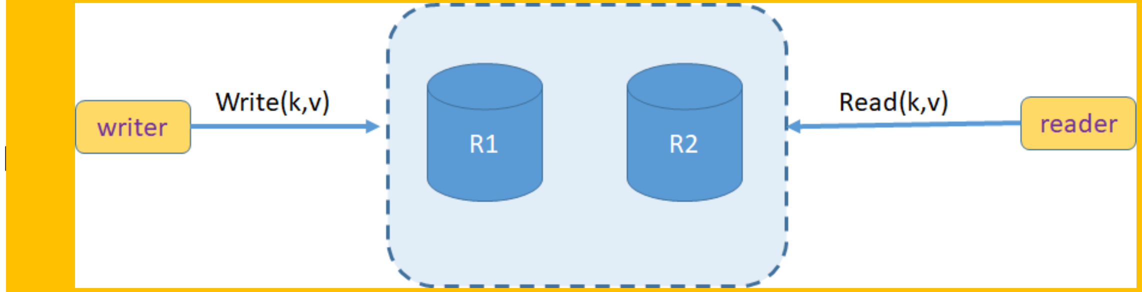
Why is this “theorem” true?

Consistency: CAP Theorem

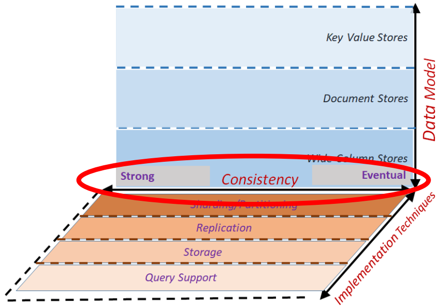


- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**:
     - system continues to work in spite of netwo

Why is this “theorem” true?

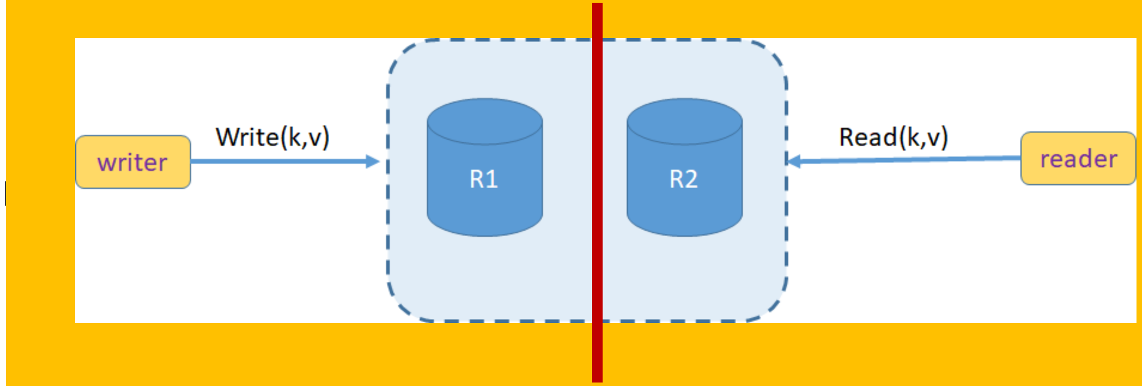


Consistency: CAP Theorem

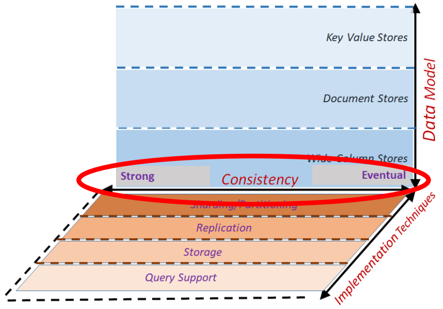


- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**:
     - system continues to work in spite of netwo

Why is this "theorem" true?

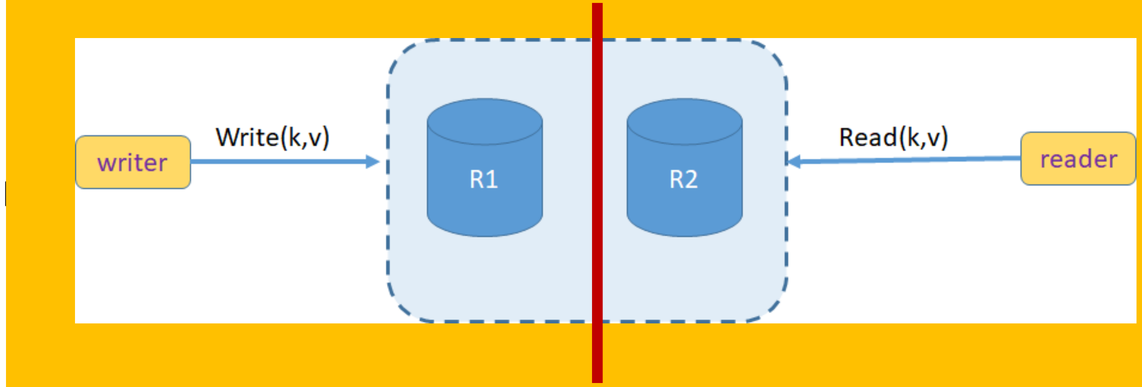


Consistency: CAP Theorem



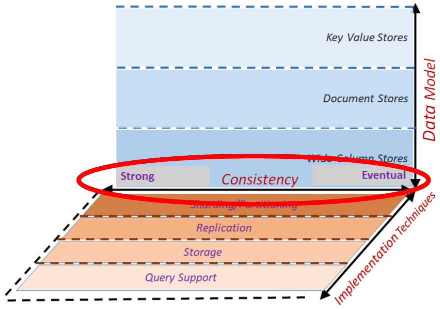
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**:
     - system continues to work in spite of netwo

Why is this "theorem" true?



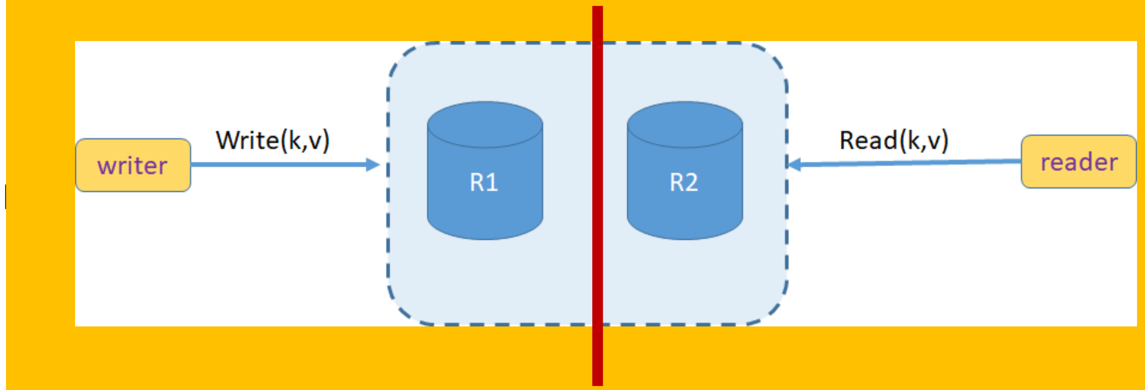
if(partition) { keep going } → !consistent && available

Consistency: CAP Theorem



- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**:
     - system continues to work in spite of netwo

Why is this “theorem” true?

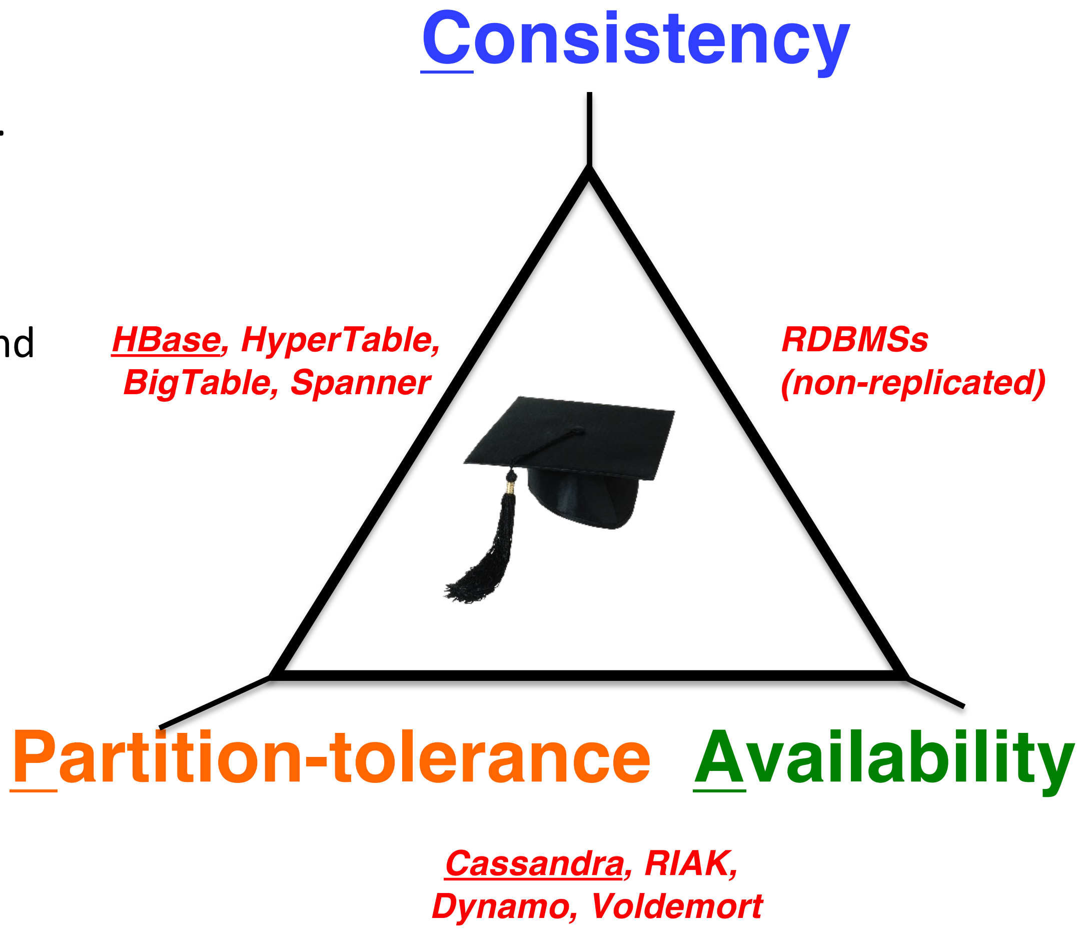


if(partition) { keep going } → !consistent && available

if(partition) { stop } → consistent && !available

CAP Implications

- A distributed storage system can achieve at most two of C, A, and P.
- When partition-tolerance is important, you have to choose between consistency and availability



Consistency

HBase, HyperTable, BigTable, Spanner

RDBMSs (non-replicated)

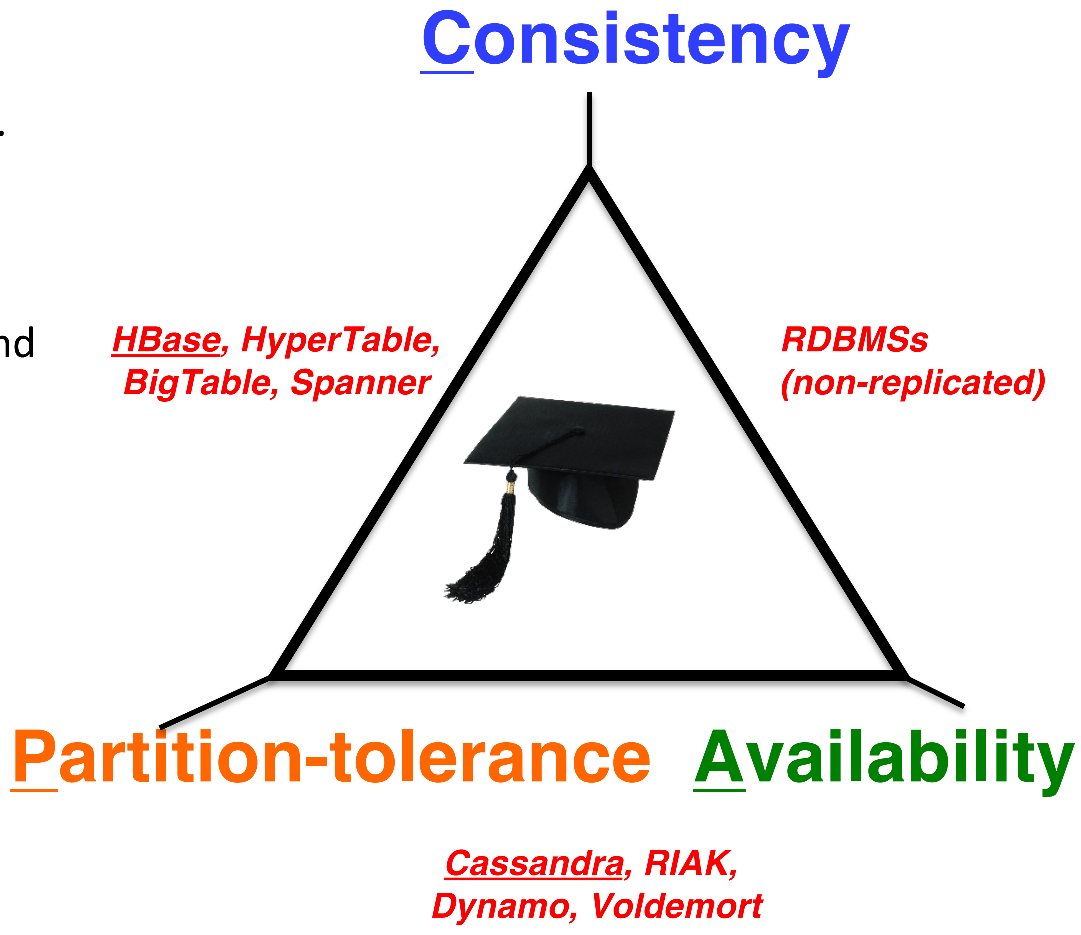
Partition-tolerance

Availability

Cassandra, RIAK, Dynamo, Voldemort

CAP Implications

- A distributed storage system can achieve at most two of C, A, and P.
- When partition-tolerance is important, you have to choose between consistency and availability



Consistency

HBase, HyperTable, BigTable, Spanner

RDBMSs (non-replicated)

Partition-tolerance

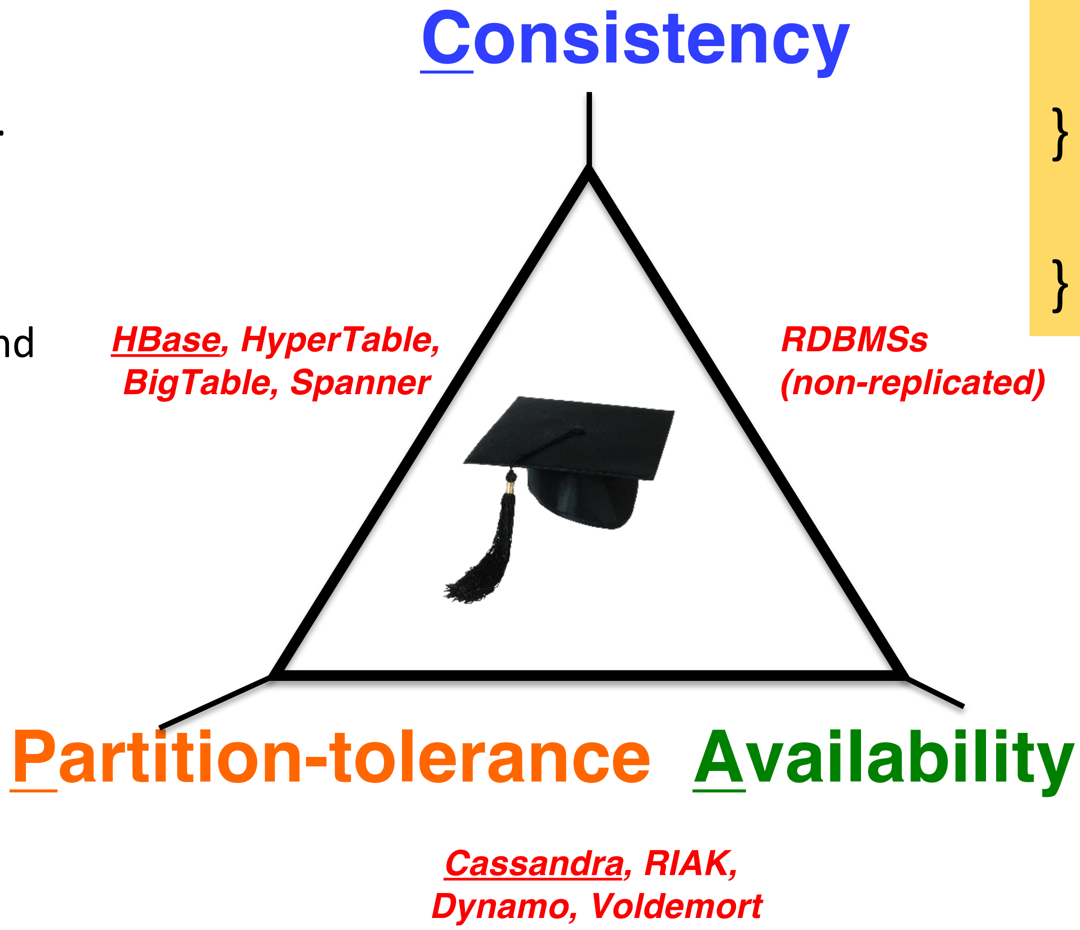
Availability

Cassandra, RIAK, Dynamo, Voldemort

CAP is flawed

CAP Implications

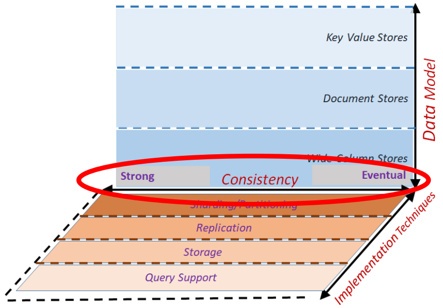
- A distributed storage system can achieve at most two of C, A, and P.
- When partition-tolerance is important, you have to choose between consistency and availability

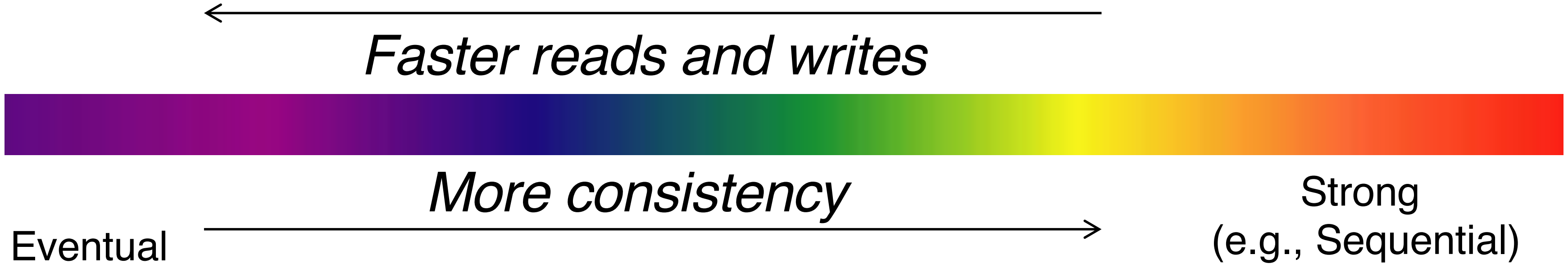

Consistency
HBase, HyperTable, BigTable, Spanner
RDBMSs (non-replicated)
Partition-tolerance
Availability
Cassandra, RIAK, Dynamo, Voldemort

PACELC:

```
if(partition) {
    choose A or C
} else {
    choose latency or consistency
}
```

Consistency Spectrum





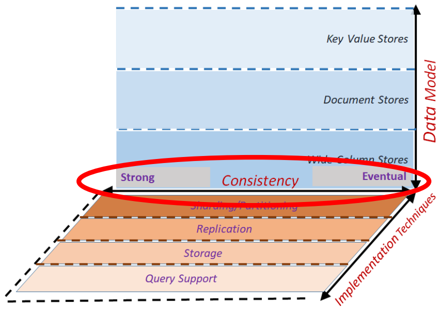
Faster reads and writes

More consistency

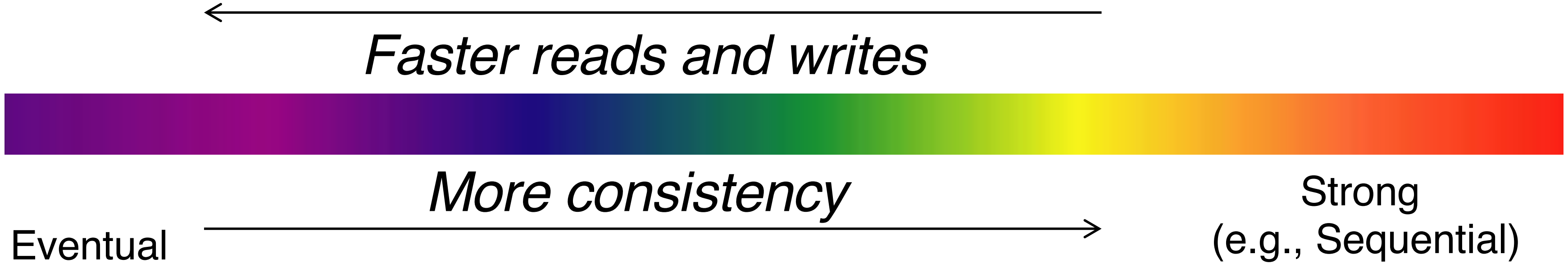
Eventual

Strong
(e.g., Sequential)

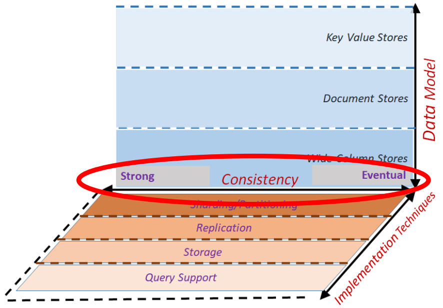
Spectrum Ends: Eventual Consistency



- Eventual Consistency
  - If writes to a key stop, all replicas of key will converge
  - Originally from Amazon’s Dynamo and LinkedIn’s Voldemort systems

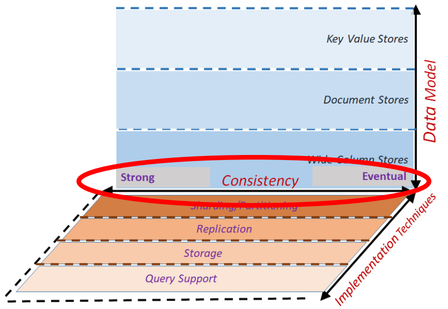


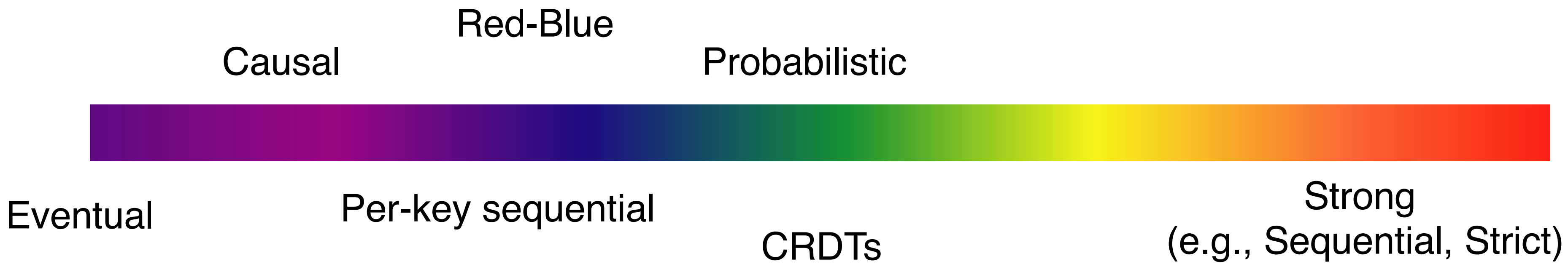
Spectrum Ends: Strong Consistency



- **Strict/Strong:**
  - Absolute time ordering of all shared accesses, reads always return last write
- **Linearizability**:
  - Each operation is visible (or available) to all other clients in real-time order
- **Sequential Consistency** [Lamport]:
  - *"… the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.*
  - After the fact, find a “reasonable” ordering of the operations (can re-order operations) that obeys sanity (consistency) at all clients, and across clients.
- **ACID** properties

Many *Many* Consistency Models





Red-Blue

Causal

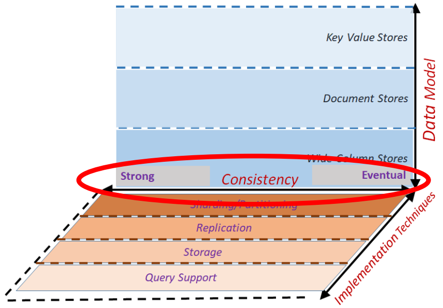
Probabilistic

Eventual

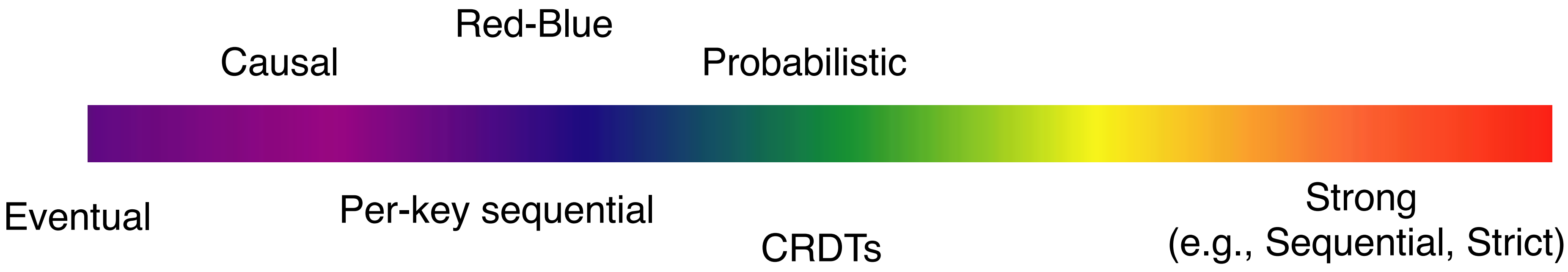
Per-key sequential

CRDTs

Strong
(e.g., Sequential, Strict)



Many *Many* Consistency Models



Red-Blue

Causal

Probabilistic

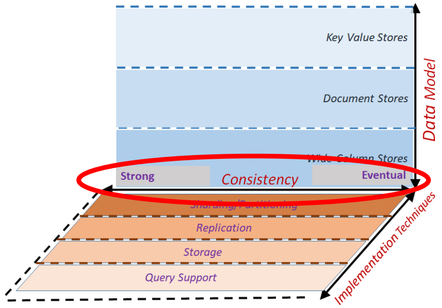
Eventual

Per-key sequential

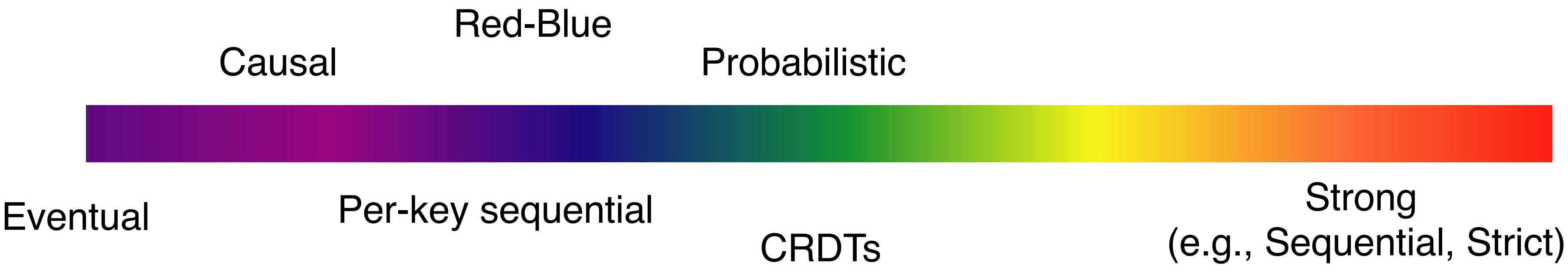
CRDTs

Strong
(e.g., Sequential, Strict)

- Amazon S3 – **eventual** consistency
- Amazon Simple DB – **eventual** or strong
- Google App Engine – **strong** or eventual
- Yahoo! PNUTS – **eventual** or strong
- Windows Azure Storage – **strong** (or eventual)
- Cassandra – **eventual** or strong (if R+W > N)
- ...



Many *Many* Consistency Models



Red-Blue

Causal

Probabilistic

Eventual

Per-key sequential

CRDTs

Strong
(e.g., Sequential, Strict)

- Amazon S3 – **eventual** consistency
- Amazon Simple DB – **eventual** or strong
- Google App Engine – **strong** or eventual
- Yahoo! PNUTS – **eventual** or strong
- Windows Azure Storage – **strong** (or eventual)
- Cassandra – **eventual** or strong (if R+W > N)
- …

Question: How to choose what to use or support?

Review: Some Consistency Guarantees

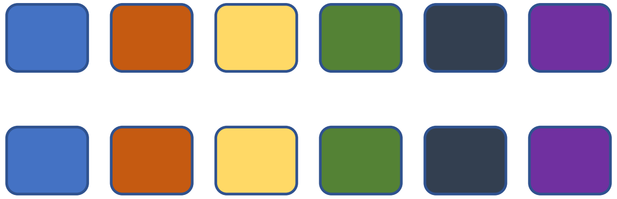
Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.

Review: Some Consistency Guarantees

🟦 🟧 🟨 🟩 ⬛ 🟪

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.

Review: Some Consistency Guarantees



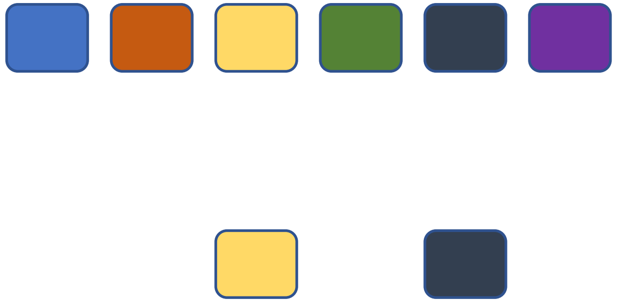
Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.

Review: Some Consistency Guarantees

🟦 🟧 🟨 🟩 ⬛ 🟪

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.

Review: Some Consistency Guarantees



Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.

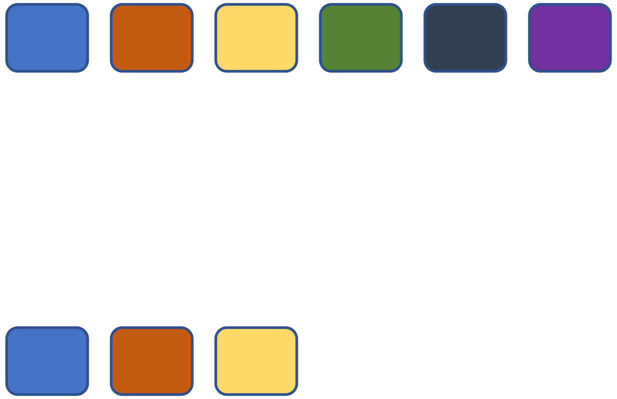
Review: Some Consistency Guarantees

🟦 🟧 🟨 🟩 ⬛ 🟪

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.

Review: Some Consistency Guarantees

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.

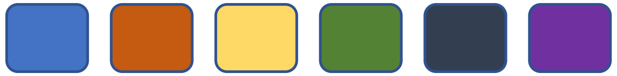


Review: Some Consistency Guarantees

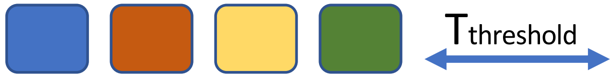
🟦 🟧 🟨 🟩 ⬛ 🟪

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.

Review: Some Consistency Guarantees



Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.

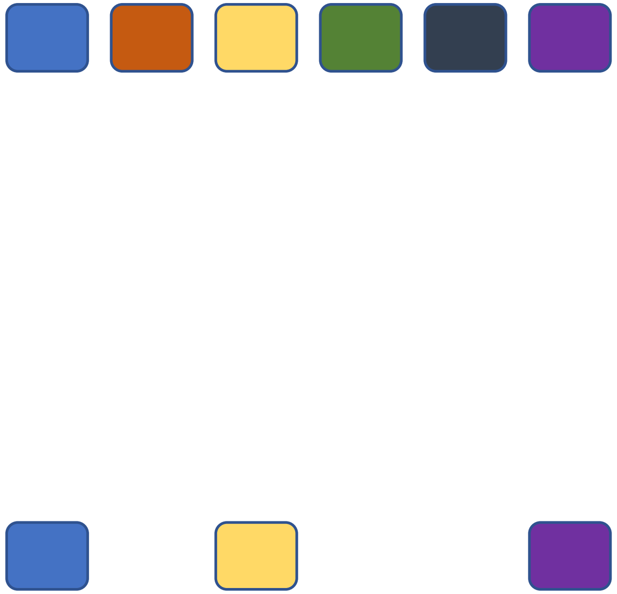


Review: Some Consistency Guarantees

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.

Review: Some Consistency Guarantees

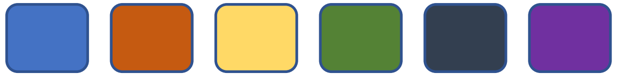
Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.



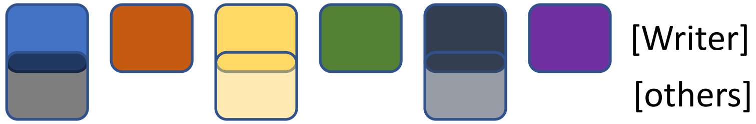
Review: Some Consistency Guarantees

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.

Review: Some Consistency Guarantees

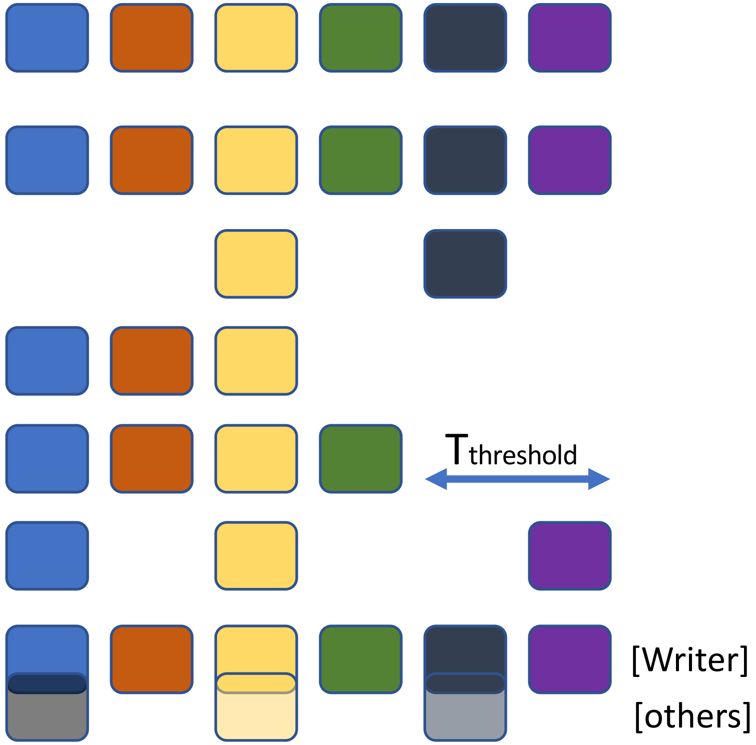


Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.



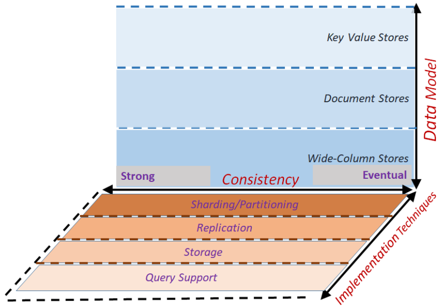
Review: Some Consistency Guarantees

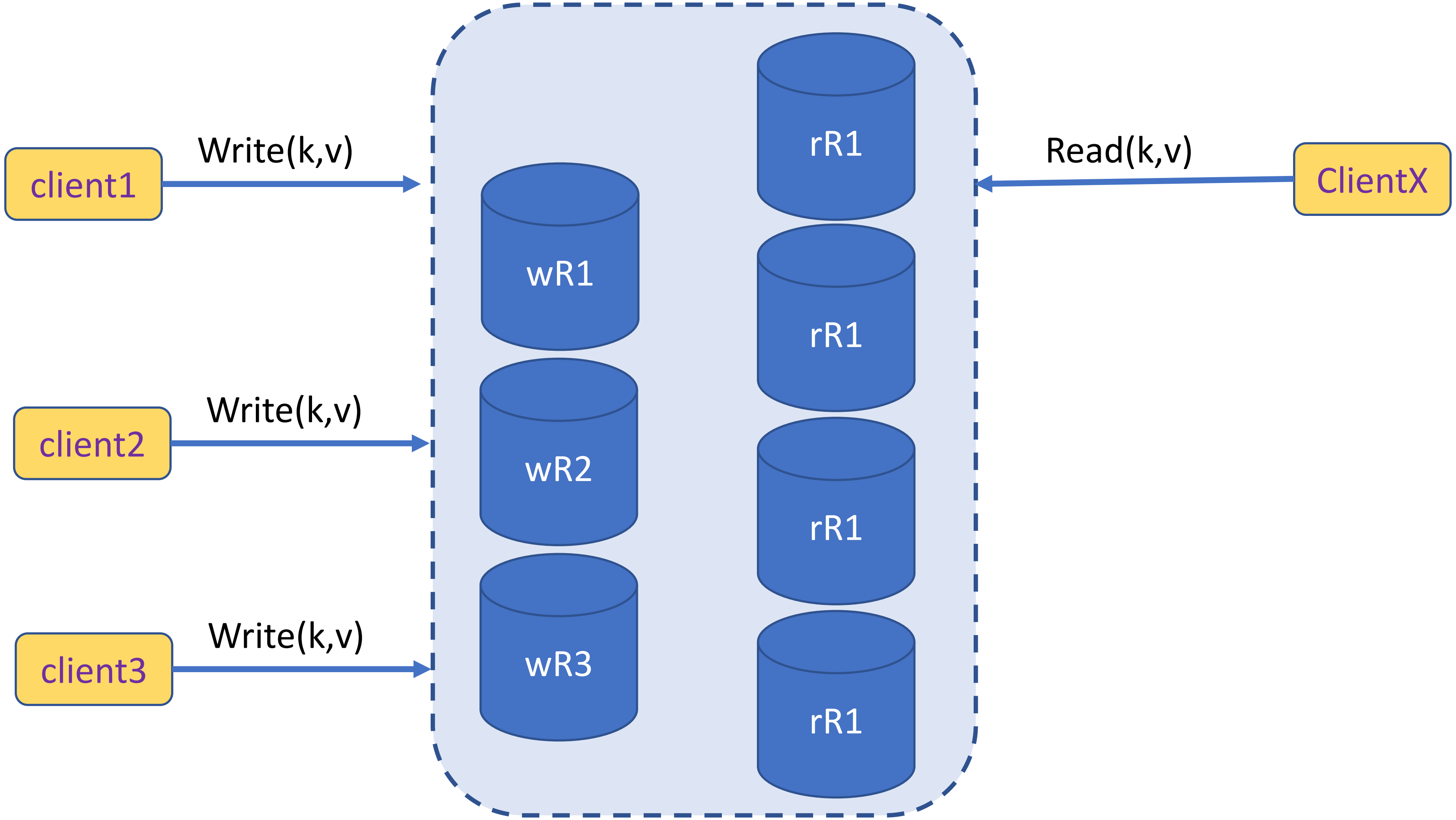
Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.

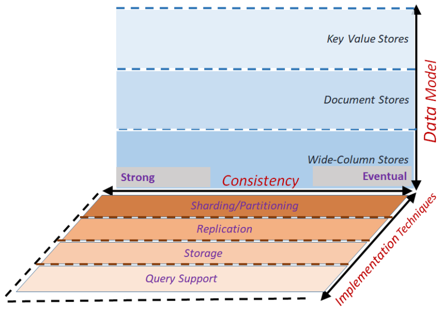


Hold the phone!

How can all these different guarantees come up?

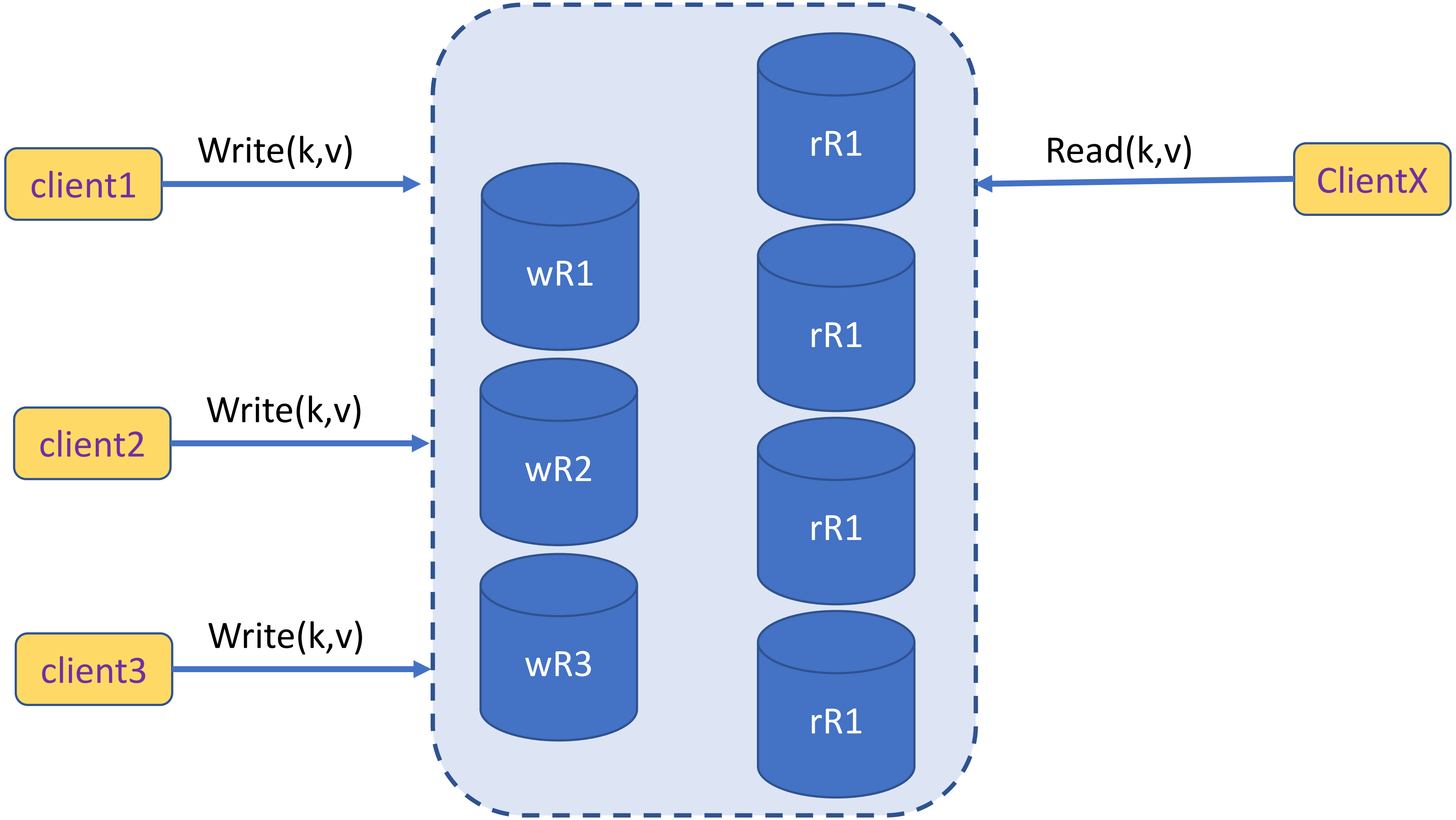


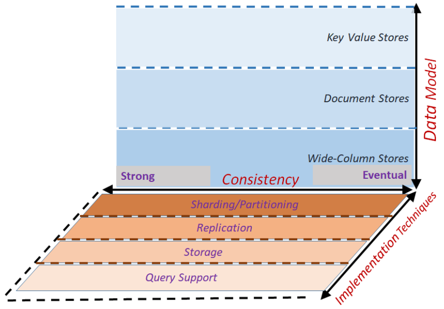




Hold the phone!

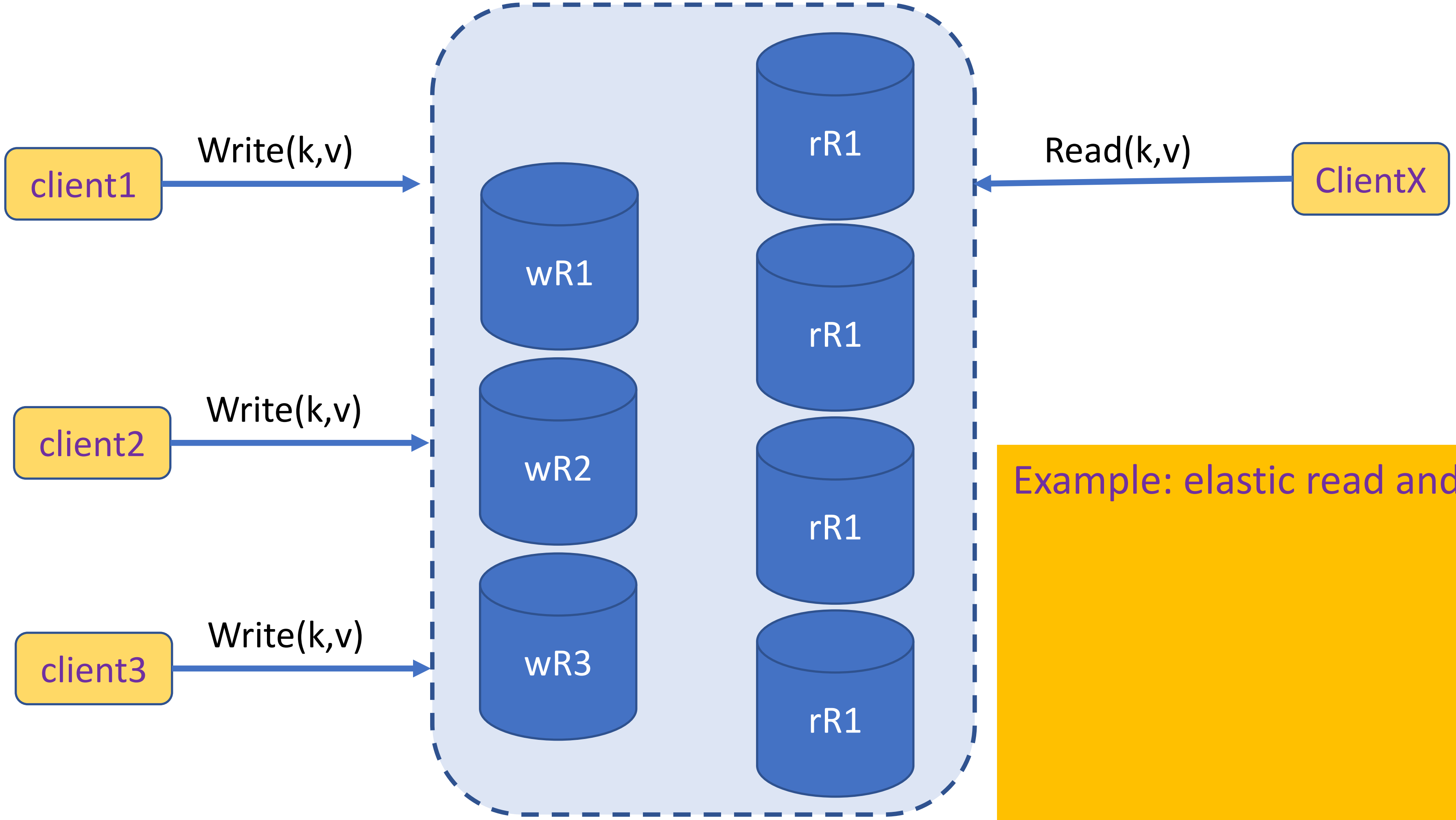
How can all these different guarantees come up?



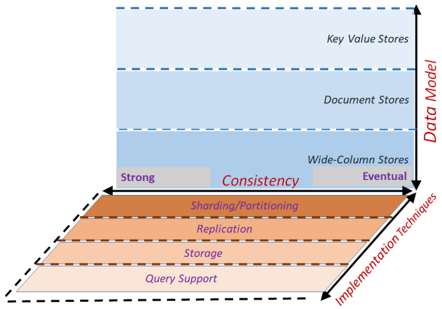


Hold the phone!

How can all these different guarantees come up?

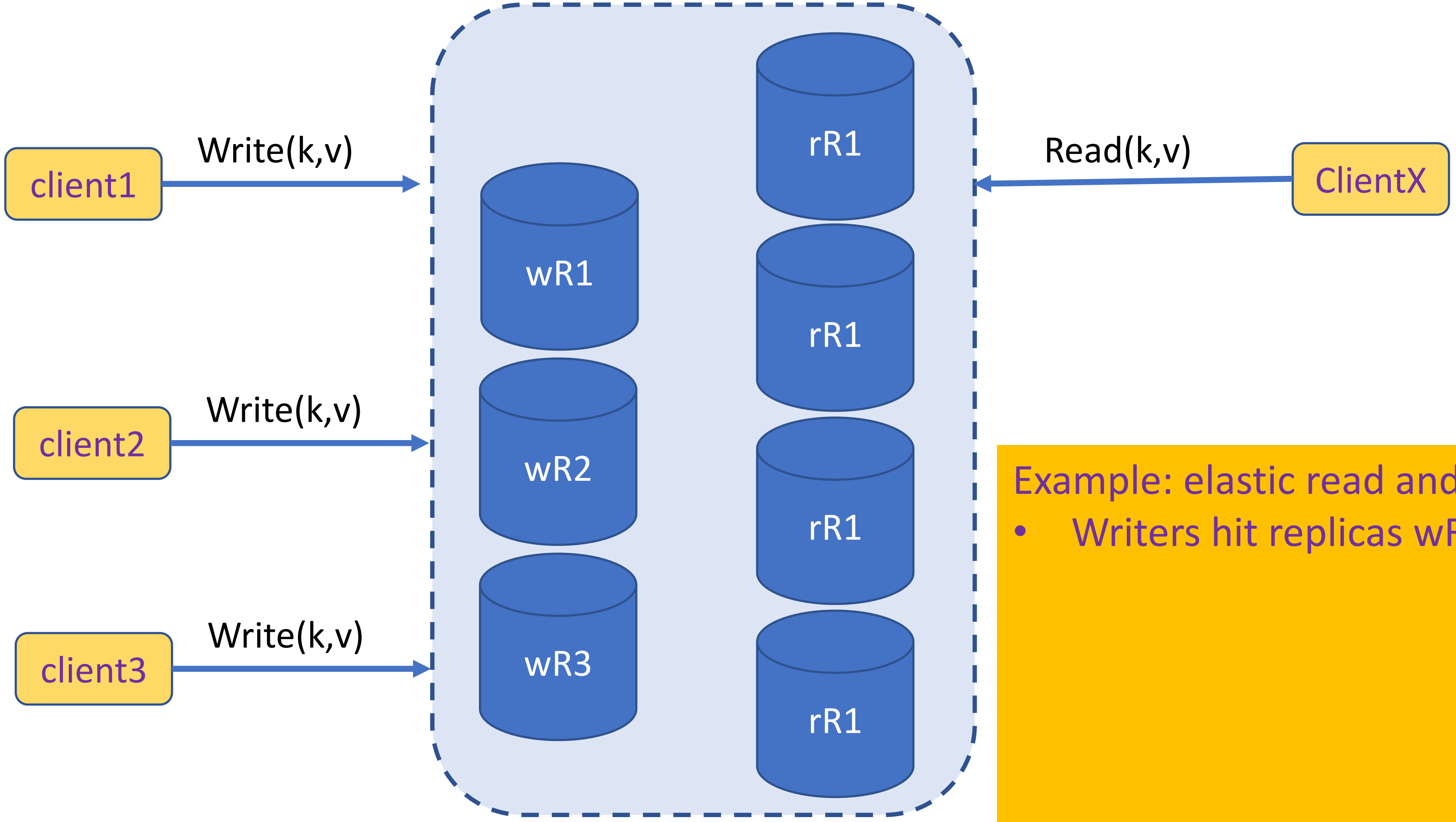


Example: elastic read and writes



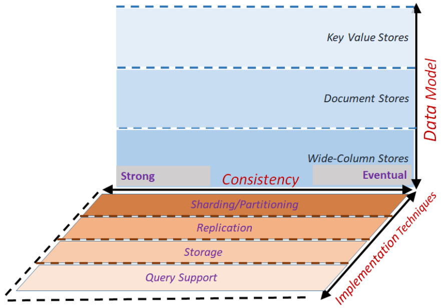
Hold the phone!

How can all these different guarantees come up?



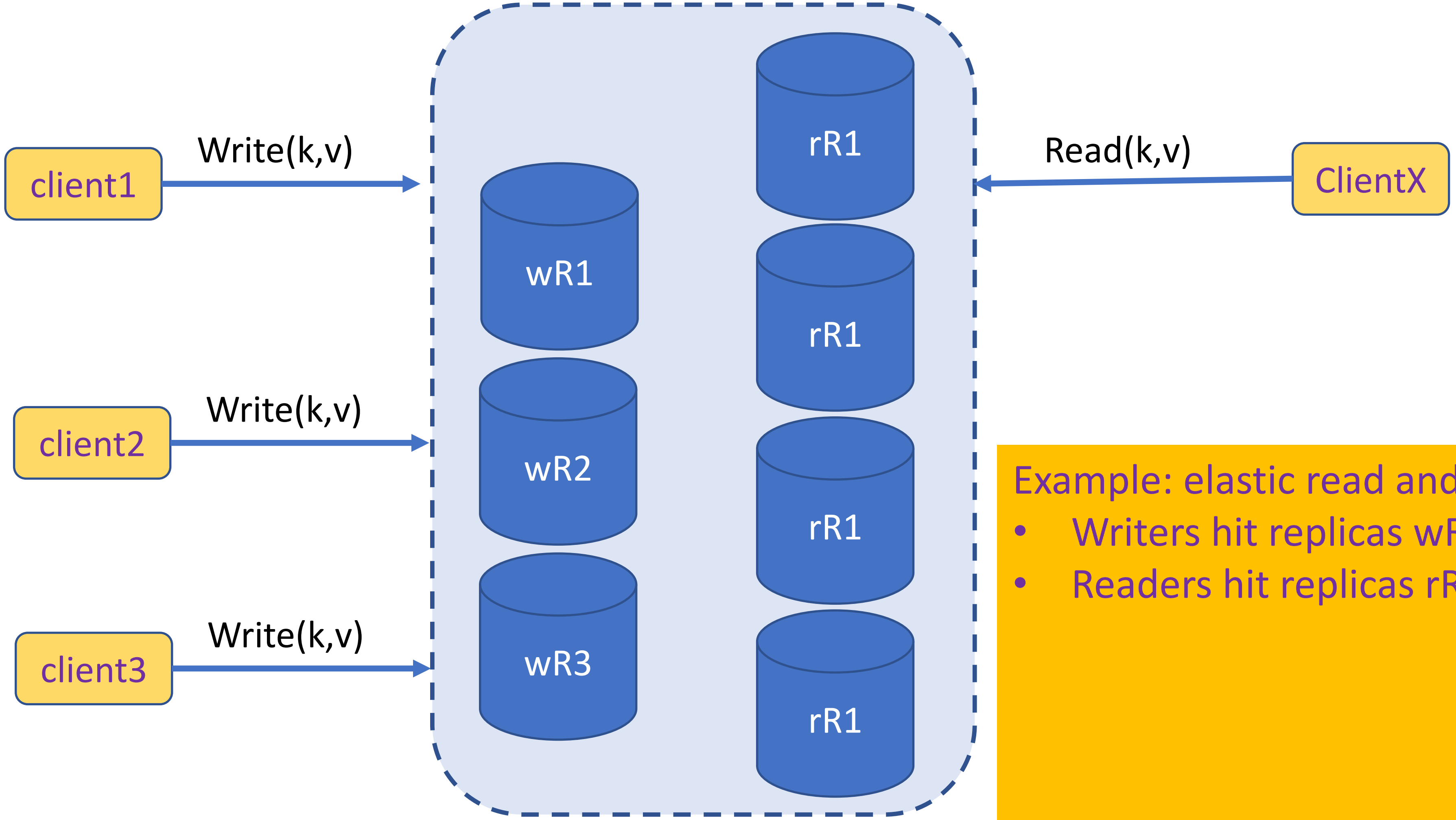
Example: elastic read and writes

- Writers hit replicas wR1..wR3



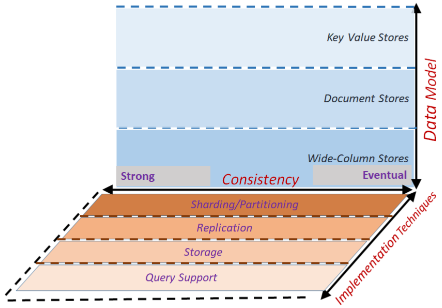
Hold the phone!

How can all these different guarantees come up?



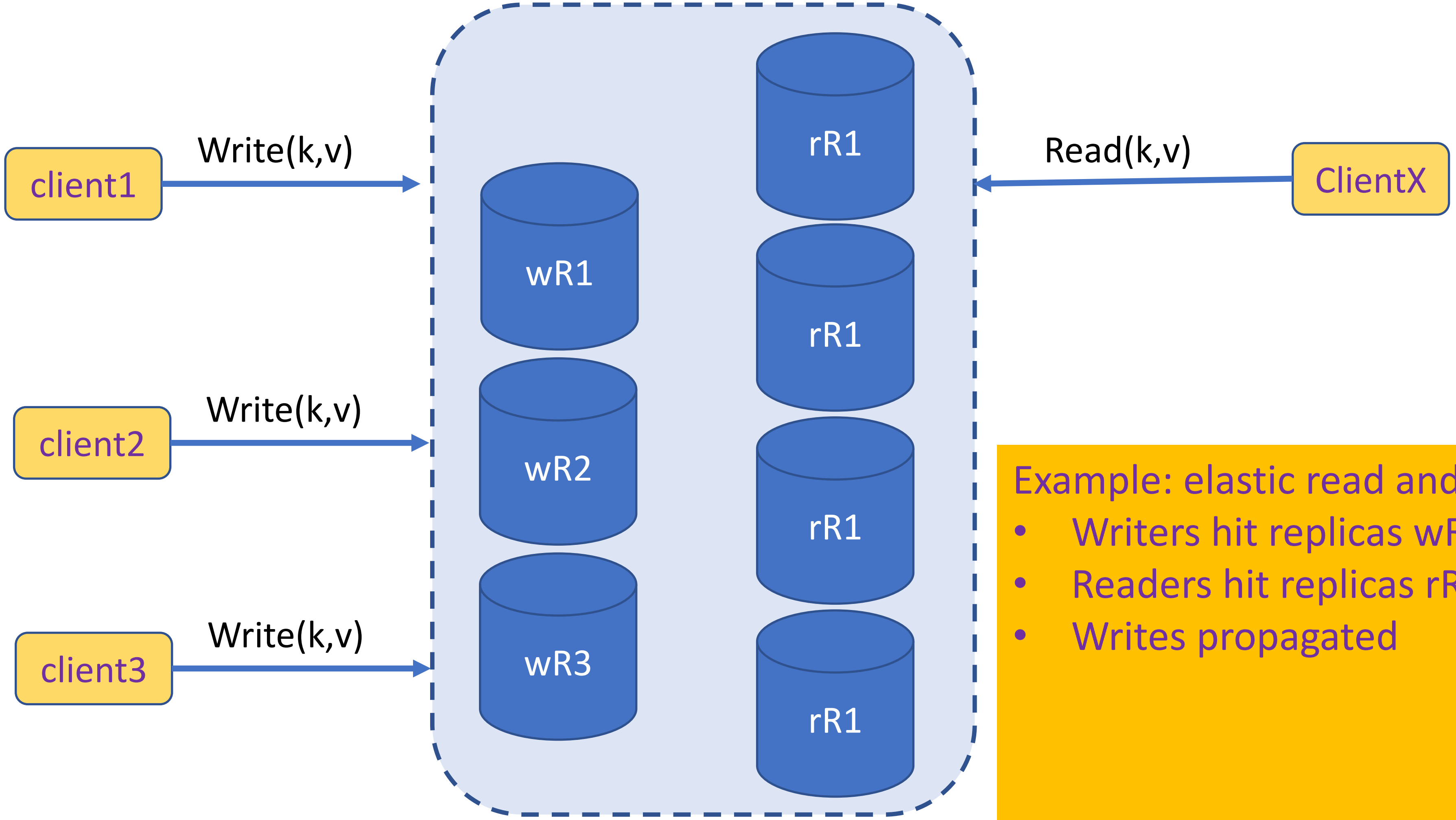
Example: elastic read and writes

- Writers hit replicas wR1..wR3
- Readers hit replicas rR1..rR4



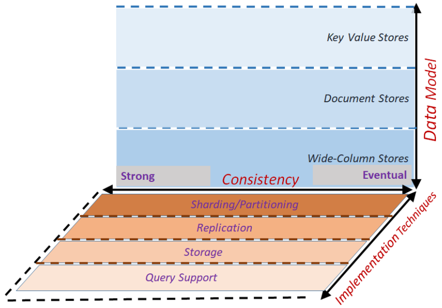
Hold the phone!

How can all these different guarantees come up?



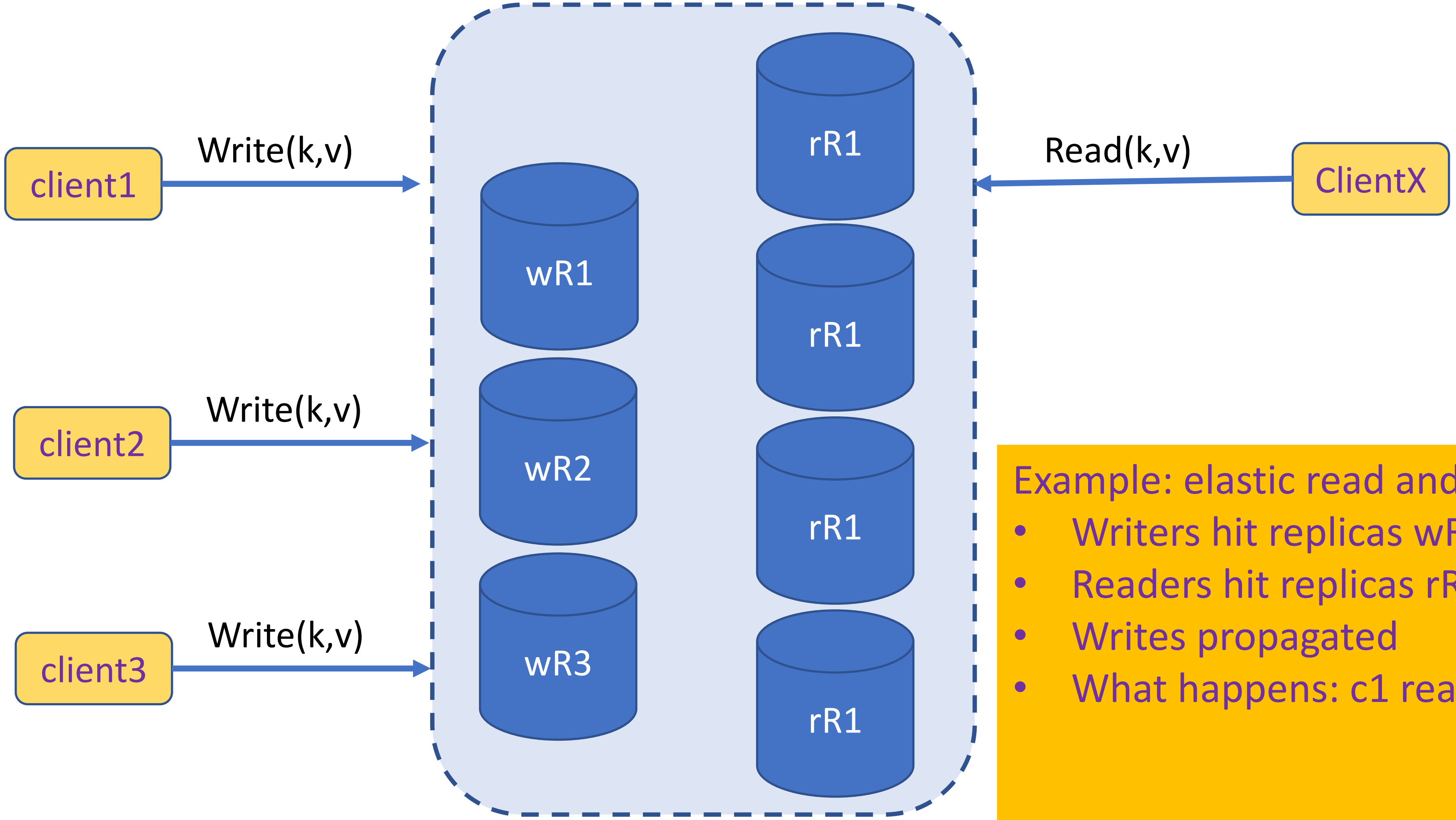
Example: elastic read and writes

- Writers hit replicas wR1..wR3
- Readers hit replicas rR1..rR4
- Writes propagated



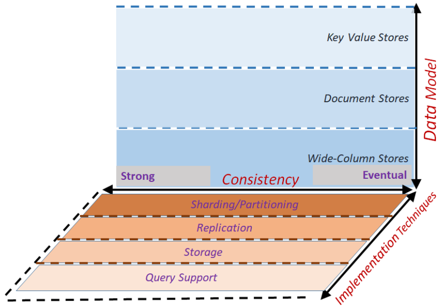
Hold the phone!

How can all these different guarantees come up?



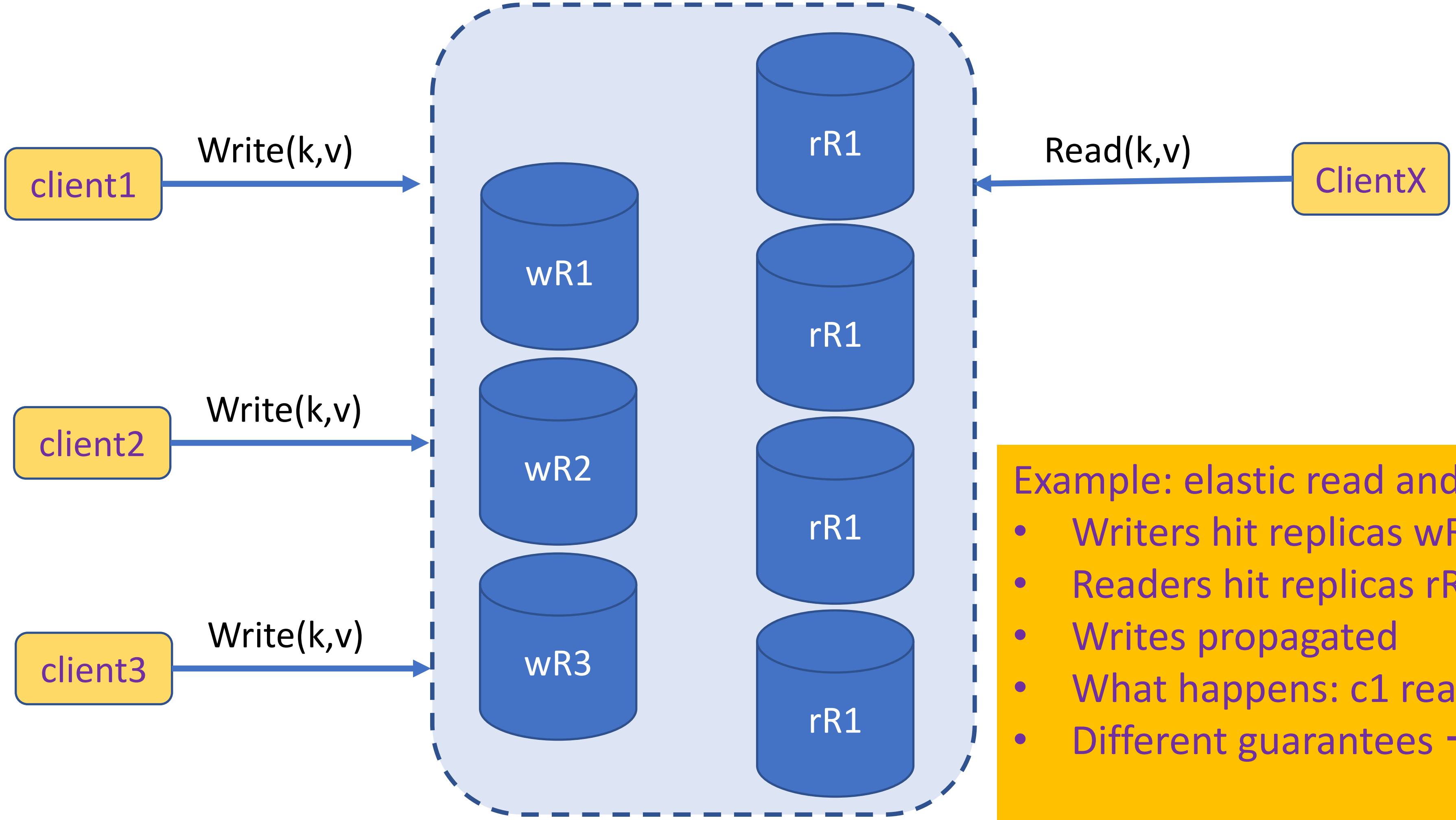
Example: elastic read and writes

- Writers hit replicas wR1..wR3
- Readers hit replicas rR1..rR4
- Writes propagated
- What happens: c1 reads own writes?



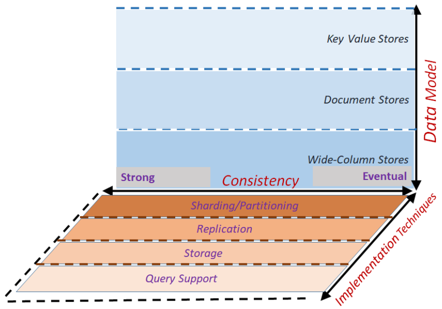
Hold the phone!

How can all these different guarantees come up?



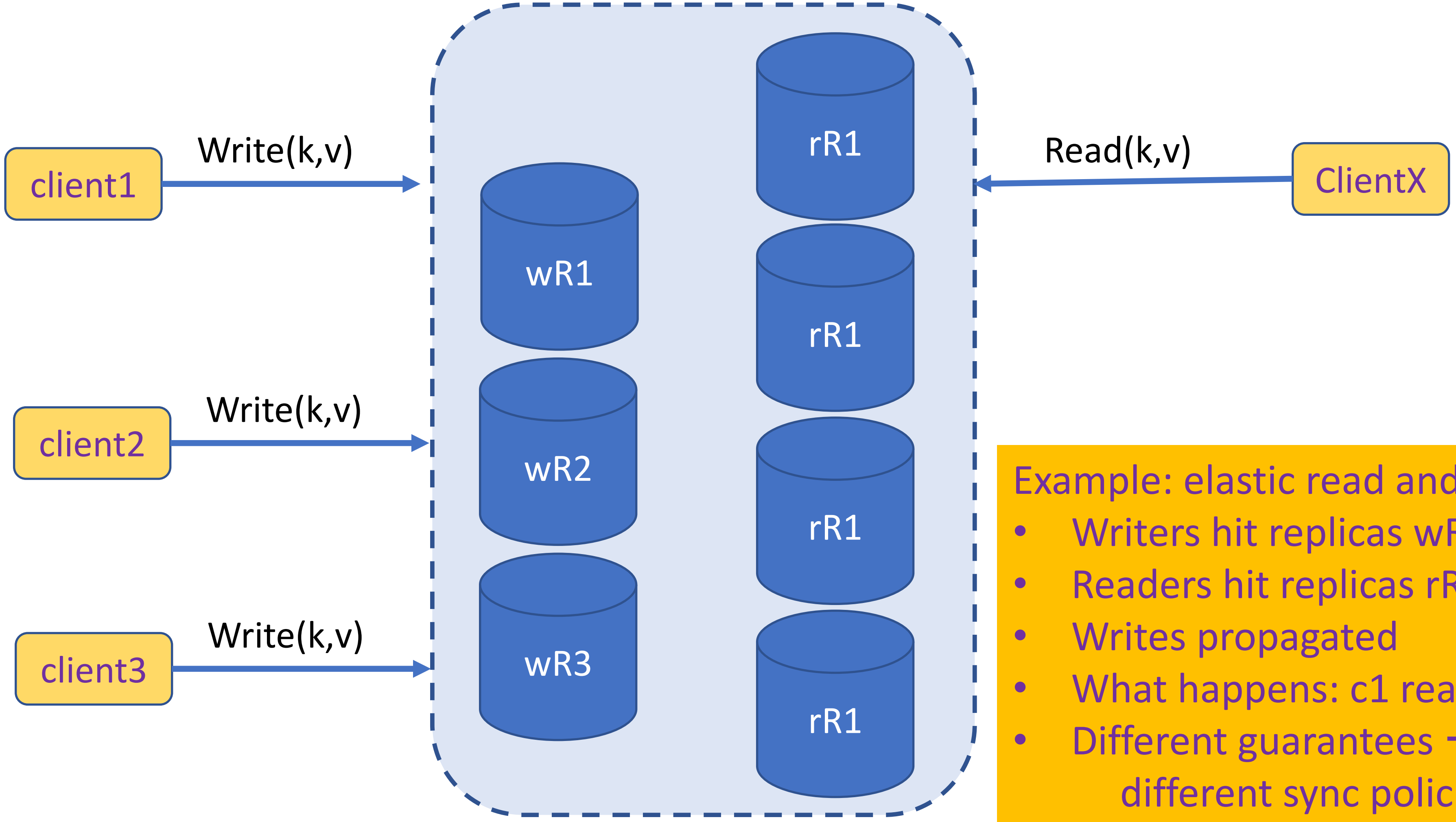
Example: elastic read and writes

- Writers hit replicas wR1..wR3
- Readers hit replicas rR1..rR4
- Writes propagated
- What happens: c1 reads own writes?
- Different guarantees →



Hold the phone!

How can all these different guarantees come up?

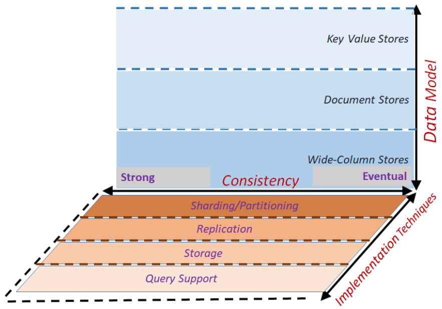


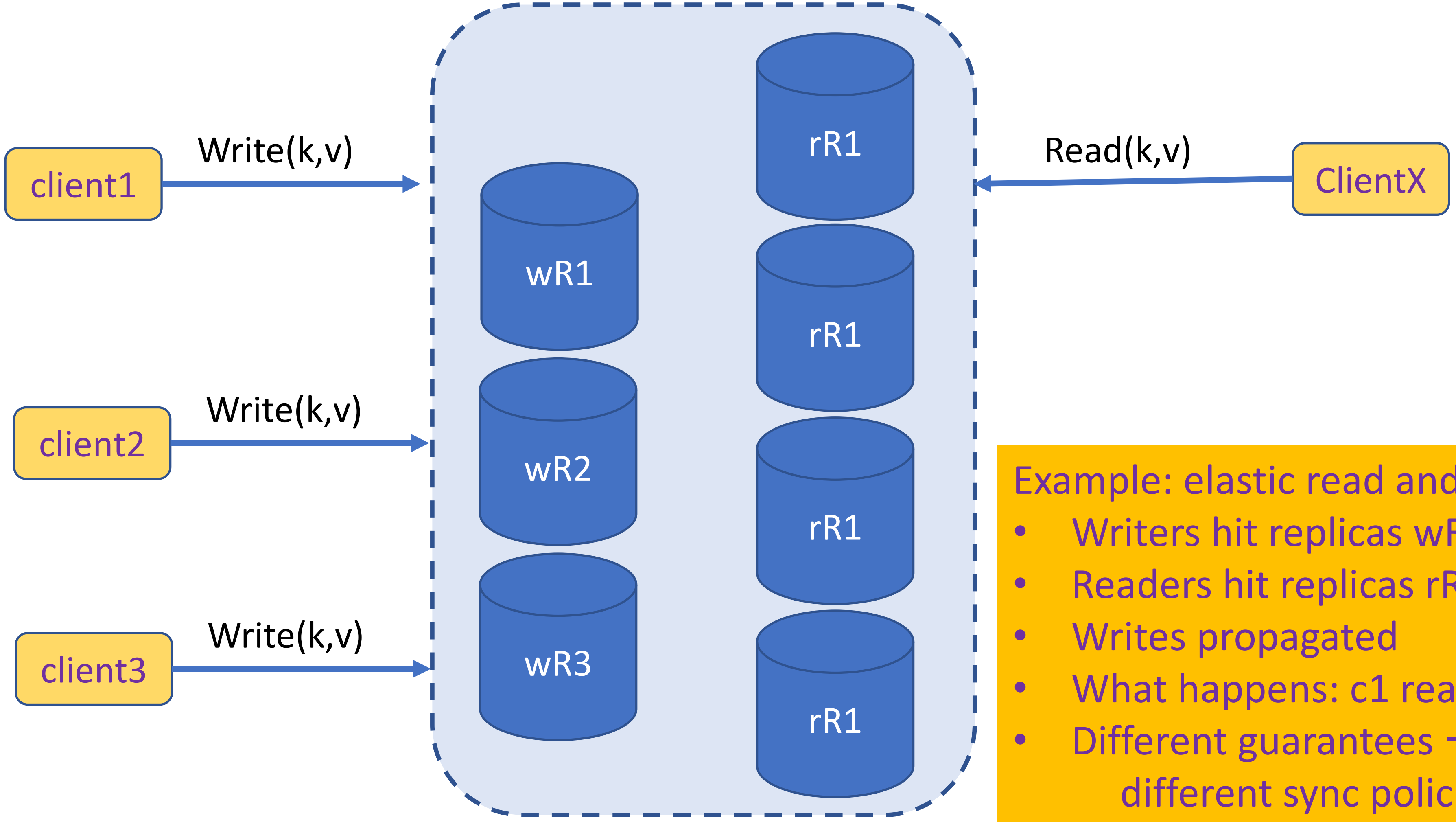
Example: elastic read and writes

- Writers hit replicas wR1..wR3
- Readers hit replicas rR1..rR4
- Writes propagated
- What happens: c1 reads own writes?
- Different guarantees → different sync policies

Hold the phone!

How can all these different guarantees come up?





Example: elastic read and writes

- Writers hit replicas wR1..wR3
- Readers hit replicas rR1..rR4
- Writes propagated
- What happens: c1 reads own writes?
- Different guarantees → different sync policies different w/r routing policies

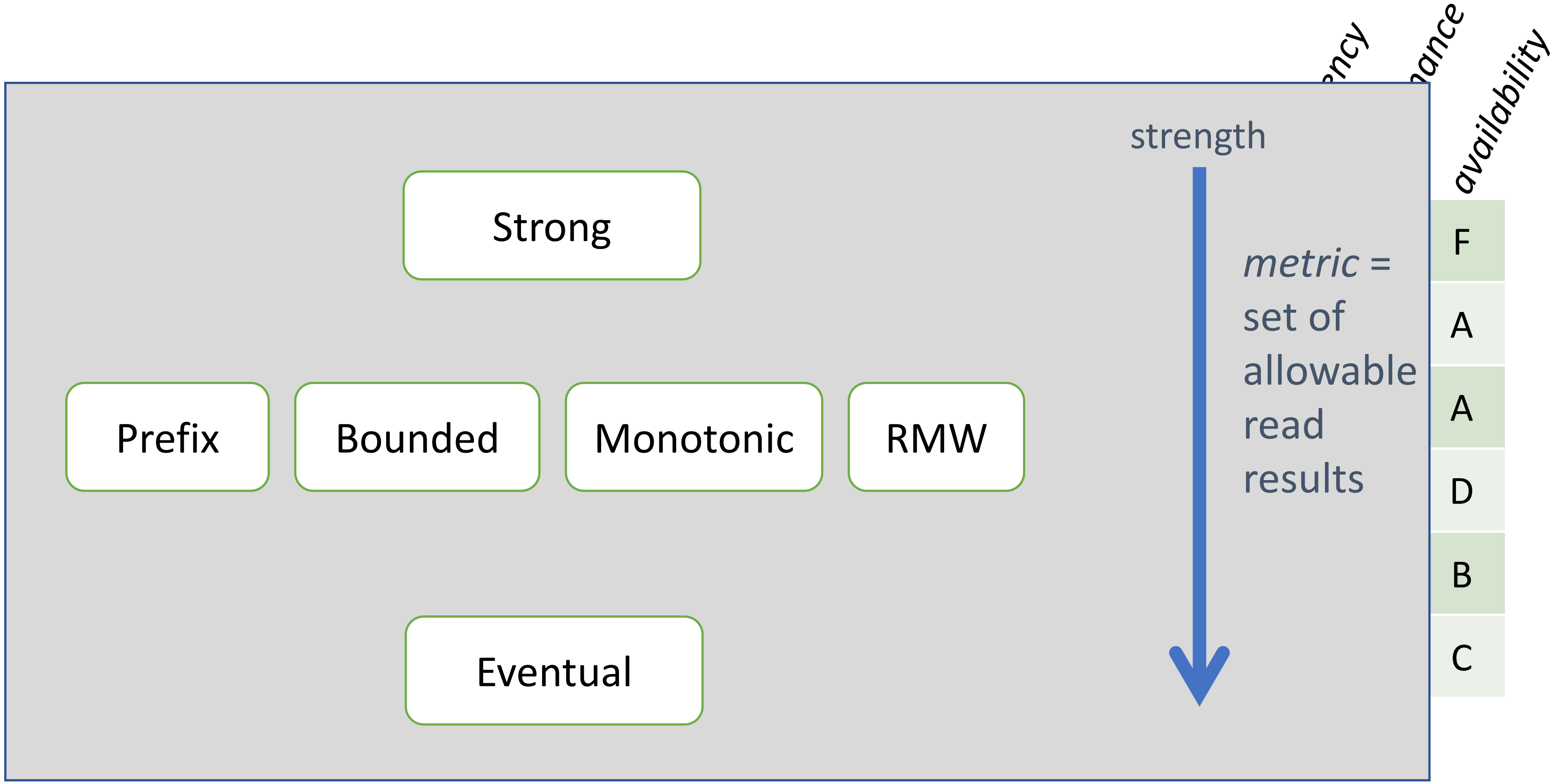
Some Consistency Guarantees

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.

Some Consistency Guarantees

		consistency	performance	availability
Strong Consistency	See all previous writes.	A	D	F
Eventual Consistency	See subset of previous writes.	D	A	A
Consistent Prefix	See initial sequence of writes.	C	B	A
Bounded Staleness	See all “old” writes.	B	C	D
Monotonic Reads	See increasing subset of writes.	C	B	B
Read My Writes	See all writes performed by reader.	C	C	C

Some Consistency Guarantees



ency

ance

availability

strength

Strong

Prefix

Bounded

Monotonic

RMW

Eventual

metric = set of allowable read results

F

A

A

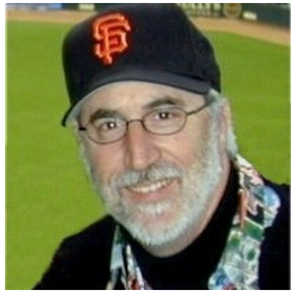
D

B

C

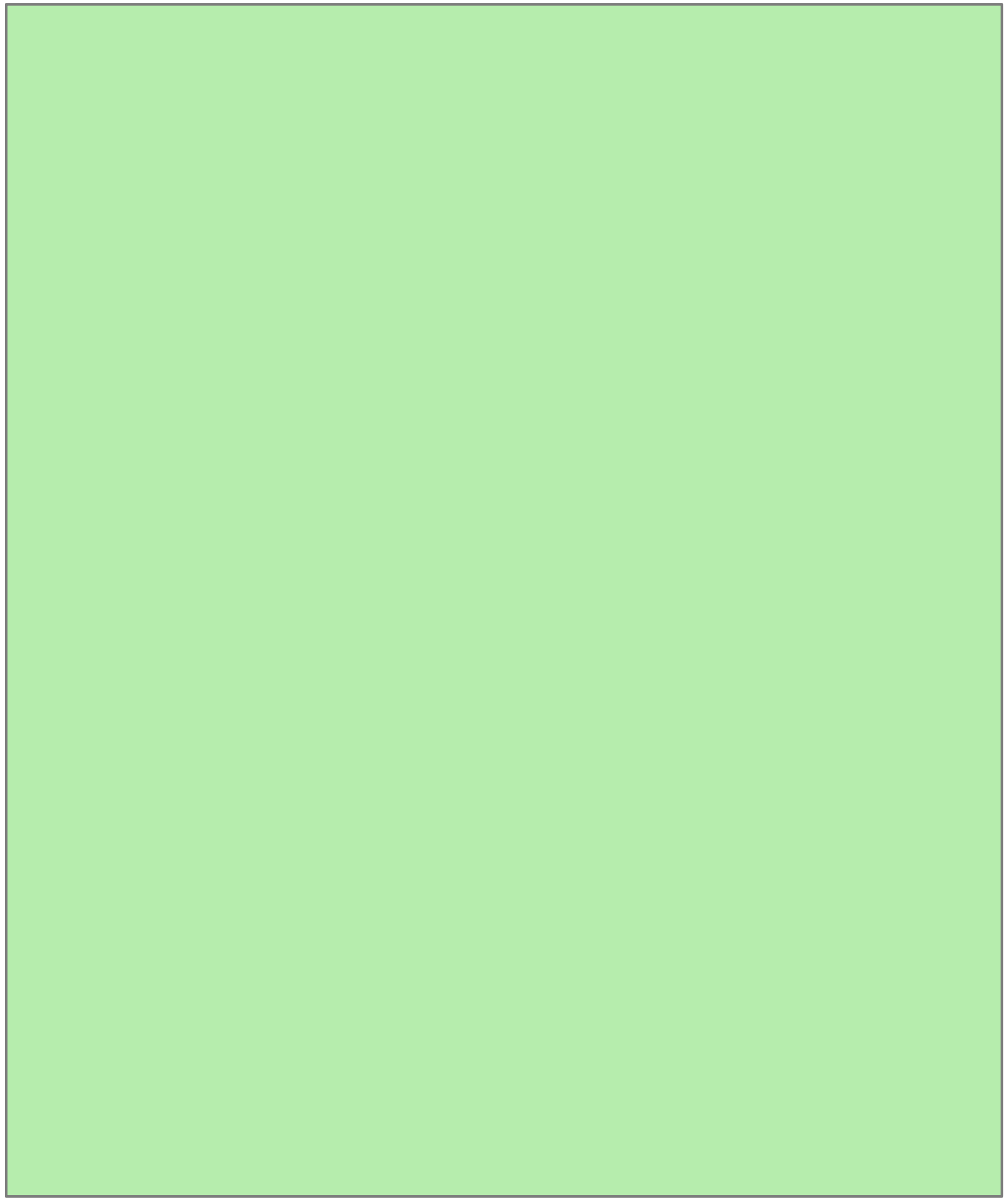
The Game of Soccer

The Game of Soccer



The Game of Soccer

The Game of Soccer



The Game of Soccer

```
for half = 1 .. 2  {
```

The Game of Soccer

```
for half = 1 .. 2  {
  while half not over {
```

The Game of Soccer

```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
```

The Game of Soccer

```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
      for each goal {
```

The Game of Soccer

```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
      for each goal {
        if visiting-team-scored {
```

The Game of Soccer

```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
      for each goal {
        if visiting-team-scored {
          score = Read (“visitors”);
```

The Game of Soccer

```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
      for each goal {
        if visiting-team-scored {
          score = Read (“visitors”);
          Write (“visitors”, score + 1);
```

The Game of Soccer

```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
      for each goal {
        if visiting-team-scored {
          score = Read (“visitors”);
          Write (“visitors”, score + 1);
        } else {
```

The Game of Soccer

```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
      for each goal {
        if visiting-team-scored {
          score = Read (“visitors”);
          Write (“visitors”, score + 1);
        } else {
          score = Read (“home”);
```

The Game of Soccer

```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
      for each goal {
        if visiting-team-scored {
          score = **Read** (“visitors”);
          **Write** (“visitors”, score + 1);
        } else {
          score = **Read** (“home”);
          **Write** (“home”, score + 1);
```

The Game of Soccer

```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
      for each goal {
        if visiting-team-scored {
           score = Read (“visitors”);
           Write (“visitors”, score + 1);
        } else {
           score = Read (“home”);
           Write (“home”, score + 1);
        } } }
```

The Game of Soccer

```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
      for each goal {
        if visiting-team-scored {
           score = Read (“visitors”);
           Write (“visitors”, score + 1);
        } else {
           score = Read (“home”);
           Write (“home”, score + 1);
        } } }
hScore = Read(“home”);
```

The Game of Soccer

```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
      for each goal {
        if visiting-team-scored {
          score = **Read** (“visitors”);
          **Write** (“visitors”, score + 1);
        } else {
          score = **Read** (“home”);
          **Write** (“home”, score + 1);
        } } }
hScore = **Read**(“home”);
vScore = **Read**(“visit”);
```

The Game of Soccer

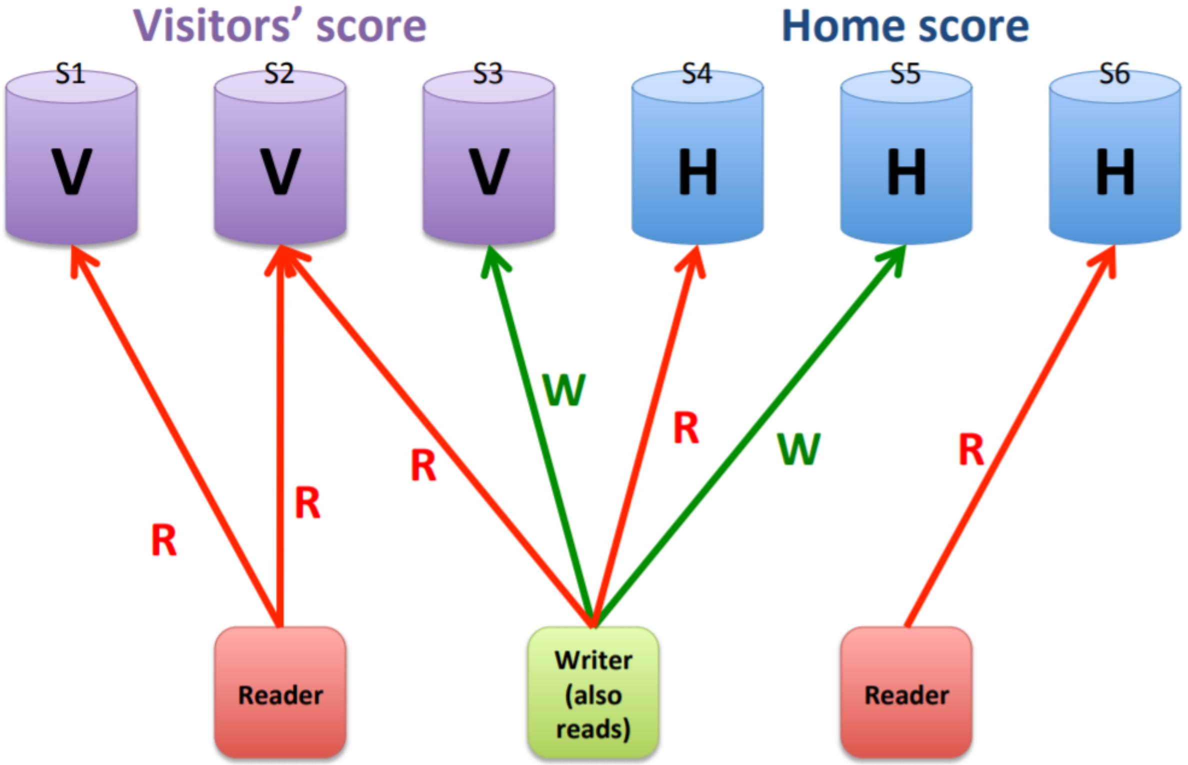
```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
      for each goal {
        if visiting-team-scored {
          score = Read ("visitors");
          Write ("visitors", score + 1);
        } else {
          score = Read ("home");
          Write ("home", score + 1);
        } } }
hScore = Read("home");
vScore = Read("visit");
if (hScore == vScore)
```

The Game of Soccer

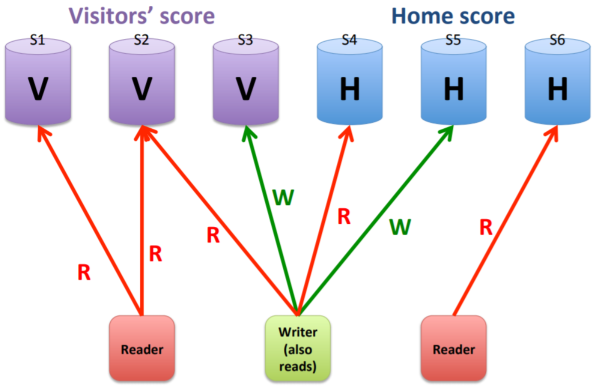
```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
      for each goal {
        if visiting-team-scored {
           score = Read (“visitors”);
           Write (“visitors”, score + 1);
        } else {
           score = Read (“home”);
           Write (“home”, score + 1);
        } } }
hScore = Read(“home”);
vScore = Read(“visit”);
if (hScore == vScore)
    play-overtime
```

The Game of Soccer

```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
      for each goal {
        if visiting-team-scored {
          score = Read (“visitors”);
          Write (“visitors”, score + 1);
        } else {
          score = Read (“home”);
          Write (“home”, score + 1);
        } } }
hScore = Read(“home”);
vScore = Read(“visit”);
if (hScore == vScore)
   play-overtime
```



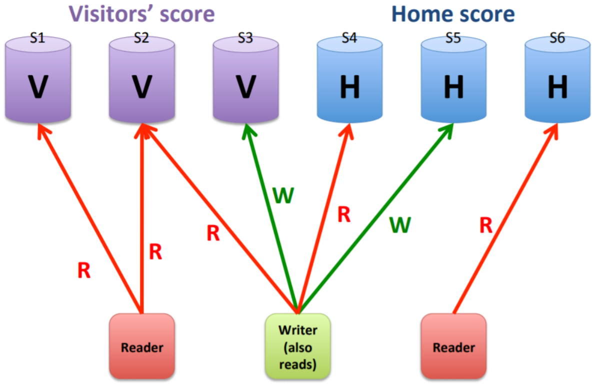
Official Scorekeeper



```
score = Read ("visitors");
Write ("visitors", score + 1);
```

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

Official Scorekeeper

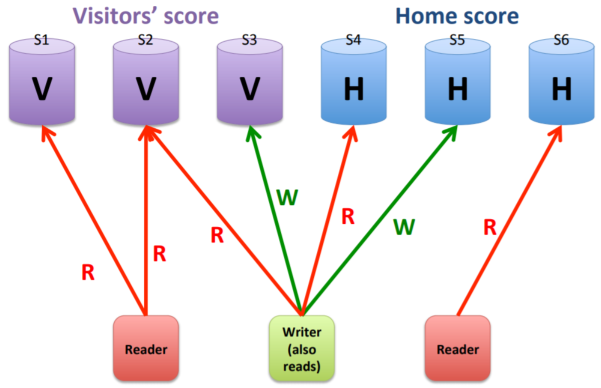


```
score = Read ("visitors");
Write ("visitors", score + 1);
```

Desired consistency?

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

Official Scorekeeper



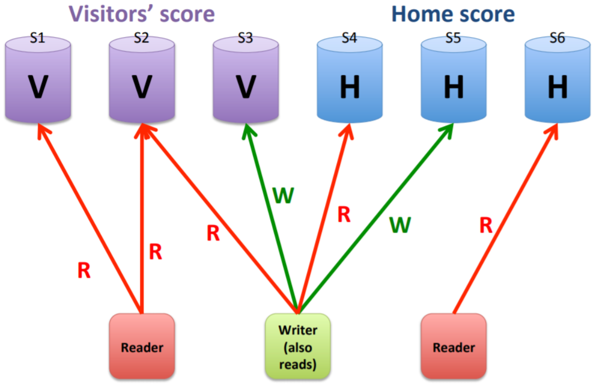
```
score = Read ("visitors");
Write ("visitors", score + 1);
```

Desired consistency?

Strong

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

Official Scorekeeper



```
score = Read ("visitors");
Write ("visitors", score + 1);
```

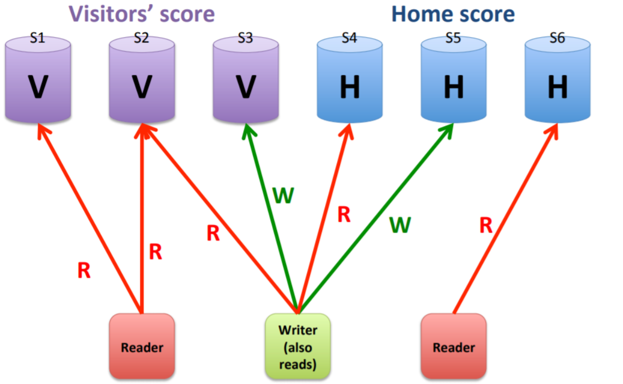
Desired consistency?

Strong

= Read My Writes!

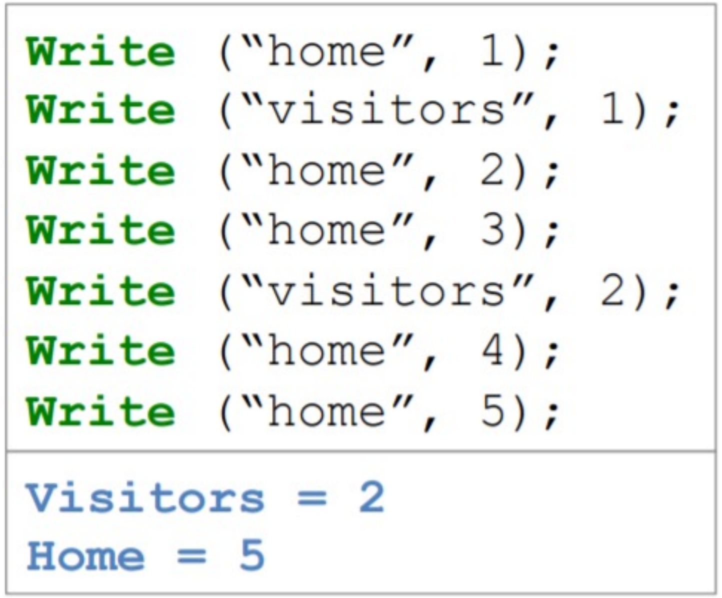
Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

Official Scorekeeper



score = **Read** ("visitors");

Write ("visitors", score + 1);



```
Write ("home", 1);
Write ("visitors", 1);
Write ("home", 2);
Write ("home", 3);
Write ("visitors", 2);
Write ("home", 4);
Write ("home", 5);
```

```
Visitors = 2
Home = 5
```

Desired consistency?

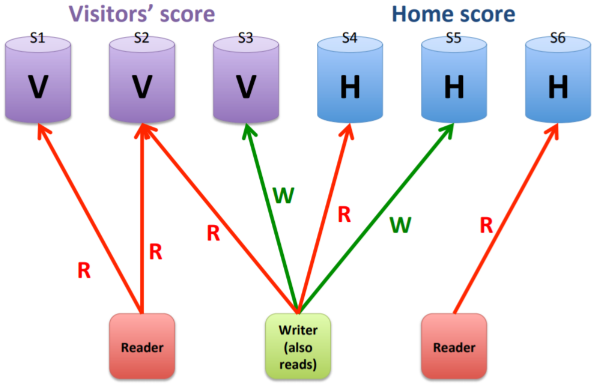
Strong

= Read My Writes!

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

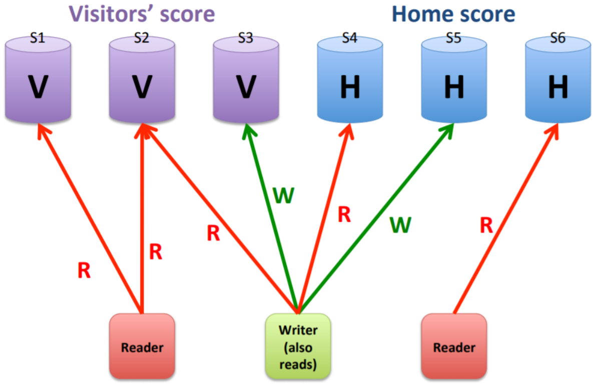
Referee

```
vScore = Read ("visitors");
hScore = Read ("home");
if vScore == hScore
      play-overtime
```



Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

Referee

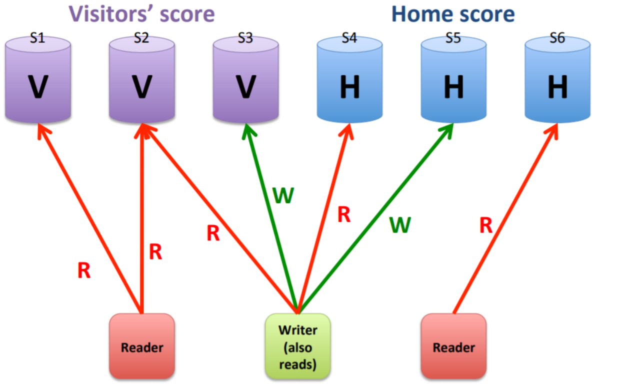


```
vScore = Read ("visitors");
hScore = Read ("home");
if vScore == hScore
      play-overtime
```

Desired consistency?

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

Referee



```
vScore = **Read** (“visitors”);
hScore = **Read** (“home”);
if vScore == hScore
      play-overtime
```

Desired consistency?

Strong consistency

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all “old” writes.

Radio Reporter

```
do {
    BeginTx();
        vScore = Read (“visitors”);
        hScore = Read (“home”);
    EndTx();
    report vScore and hScore;
    sleep (30 minutes);
}
```

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all “old” writes.

Radio Reporter

```
do {
    BeginTx();
        vScore = Read (“visitors”);
        hScore = Read (“home”);
    EndTx();
    report vScore and hScore;
    sleep (30 minutes);
}
```

Desired consistency?

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all “old” writes.

Radio Reporter

```
do {
    BeginTx();
        vScore = Read (“visitors”);
        hScore = Read (“home”);
    EndTx();
    report vScore and hScore;
    sleep (30 minutes);
}
```

Desired consistency?

Consistent Prefix

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all “old” writes.

Radio Reporter

```
do {
    BeginTx();
        vScore = Read (“visitors”);
        hScore = Read (“home”);
    EndTx();
    report vScore and hScore;
    sleep (30 minutes);
}
```

Desired consistency?

Consistent Prefix

Monotonic Reads

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all “old” writes.

Radio Reporter

```
do {
    BeginTx();
        vScore = Read (“visitors”);
        hScore = Read (“home”);
    EndTx();
    report vScore and hScore;
    sleep (30 minutes);
}
```

Desired consistency?

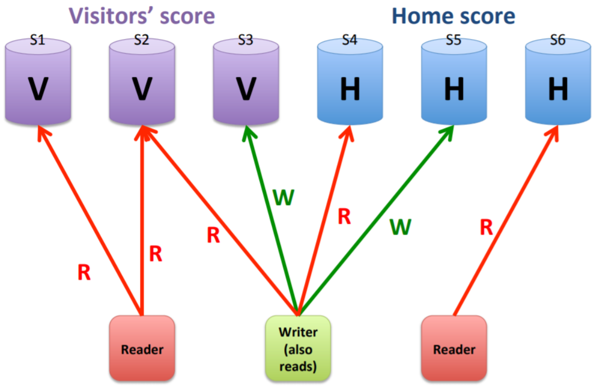
Consistent Prefix

Monotonic Reads

or Bounded Staleness

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all “old” writes.

Radio Reporter



```
do {
    BeginTx();
        vScore = Read ("visitors");
        hScore = Read ("home");
    EndTx();
    report vScore and hScore;
    sleep (30 minutes);
}
```

Desired consistency?

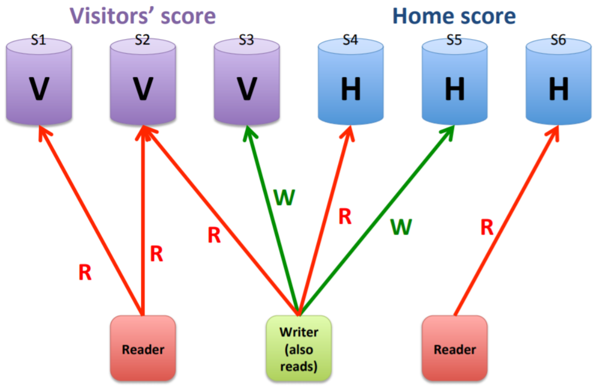
Consistent Prefix

Monotonic Reads

or Bounded Staleness

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

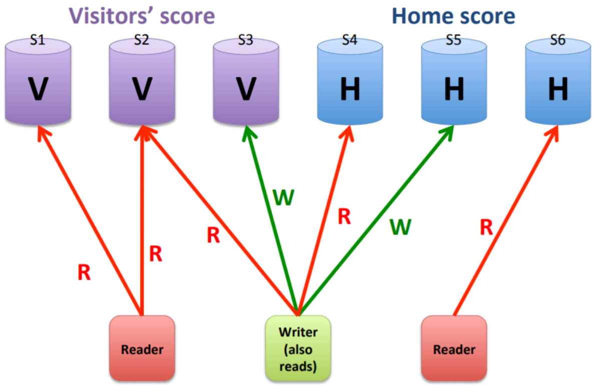
Sportswriter



```
 While not end of game {
      drink beer;
      smoke cigar;
}
go out to dinner;
vScore = Read ("visitors");
hScore = Read ("home");
write article;
```

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all “old” writes.

Sportswriter

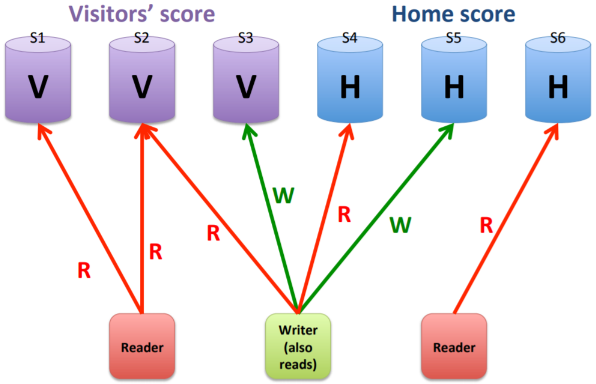


```
While not end of game {
    drink beer;
    smoke cigar;
}
go out to dinner;
vScore = Read (“visitors”);
hScore = Read (“home”);
write article;
```

Desired consistency?

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all “old” writes.

Sportswriter



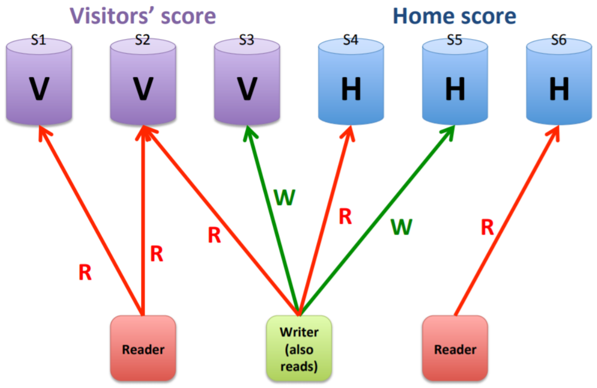
```
While not end of game {
      drink beer;
      smoke cigar;
}
go out to dinner;
vScore = Read ("visitors");
hScore = Read ("home");
write article;
```

Desired consistency?

Eventual

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

Sportswriter



```
While not end of game {
    drink beer;
    smoke cigar;
}
go out to dinner;
vScore = Read ("visitors");
hScore = Read ("home");
write article;
```

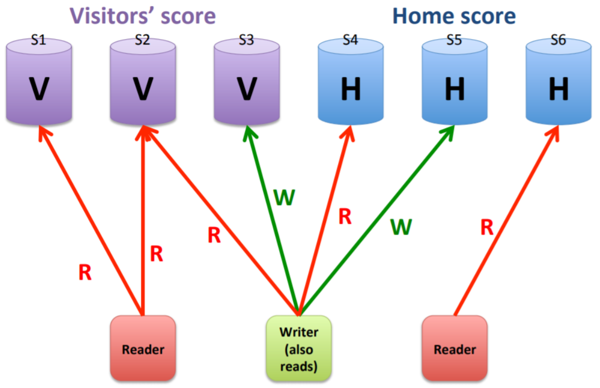
Desired consistency?

Eventual

Bounded Staleness

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

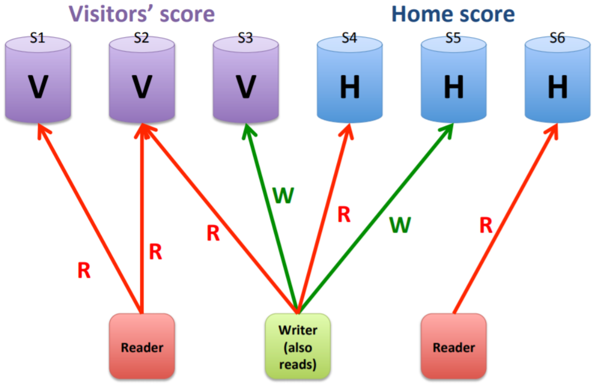
Statistician



```
Wait for end of game;
score = Read ("home");
stat = Read ("season-goals");
Write ("season-goals", stat + score);
```

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

Statistician

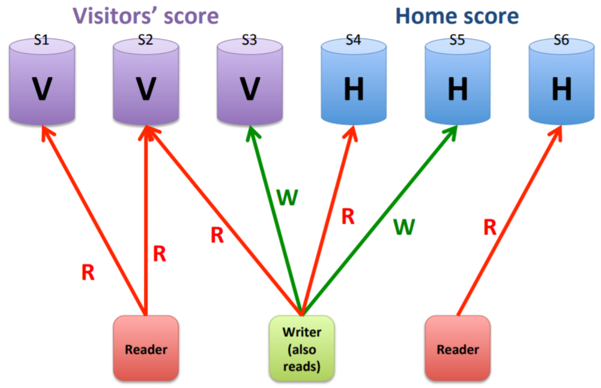


```
Wait for end of game;
score = Read ("home");
stat = Read ("season-goals");
Write ("season-goals", stat + score);
```

Desired consistency?

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

Statistician



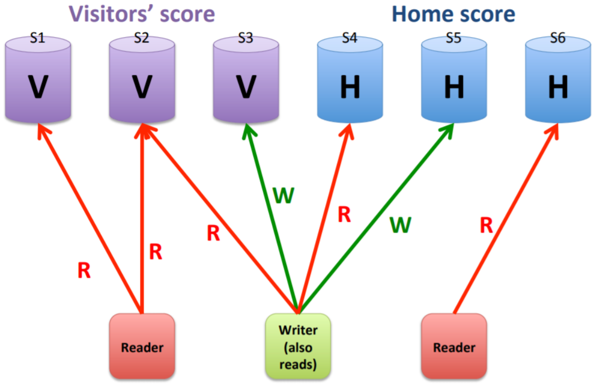
```
Wait for end of game;
score = Read ("home");
stat = Read ("season-goals");
Write ("season-goals", stat + score);
```

Desired consistency?

Strong Consistency (1st read)

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

Statistician



```
Wait for end of game;
score = Read ("home");
stat = Read ("season-goals");
Write ("season-goals", stat + score);
```

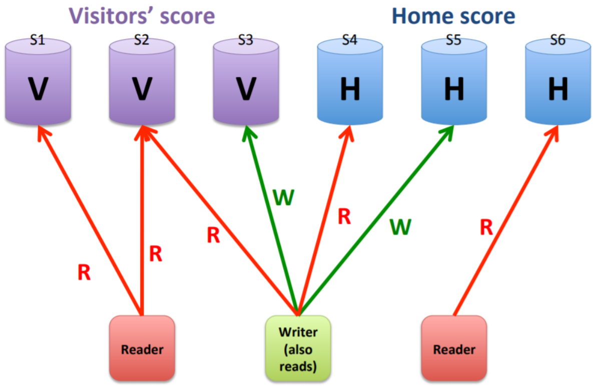
Desired consistency?

Strong Consistency (1st read)

Read My Writes (2nd read)

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

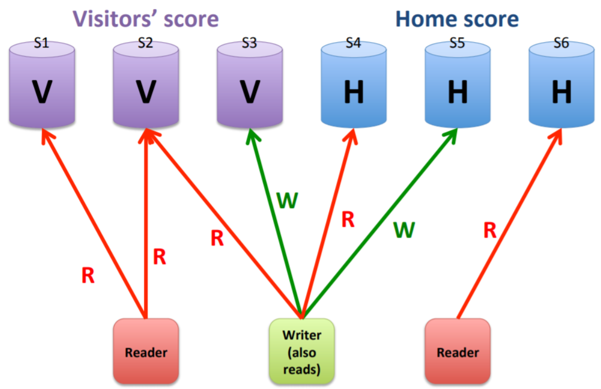
Stat Watcher



```
do {
        stat = Read ("season-goals");
        discuss stats with friends;
        sleep (1 day);
}
```

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

Stat Watcher

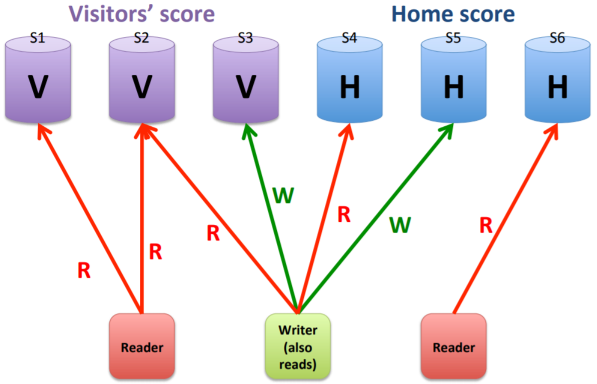


```
do {
      stat = Read ("season-goals");
      discuss stats with friends;
      sleep (1 day);
}
```

Desired consistency?

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

Stat Watcher



```
do {
      stat = **Read** ("season-goals");
      discuss stats with friends;
      sleep (1 day);
}
```

Desired consistency?

Eventual Consistency

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

Official scorekeeper:

```
score = Read ("visitors");
Write ("visitors"
```

Read My Writes

Referee:

Strong Consistency

Radio reporter:

```
do {
    vScore = Read ("visitors");
    hScore = Read ("home");
    report vScore and hSc
    sleep (30 minutes)
}
```

Consistent Prefix

Monotonic Reads

Sportswriter:

```
 While not end of game {
     drink beer;
     smoke cigar;
}
go out to dinner;
vScore = Read ("visit
hScore = R        );
write article;
```

Bounded Staleness

Statistician:

```
Wait for end of game;
score = Read ("home");
stat = Read ("season-goals");
Write ("season-goals", stat +
```

Strong Consistency

Read My Writes

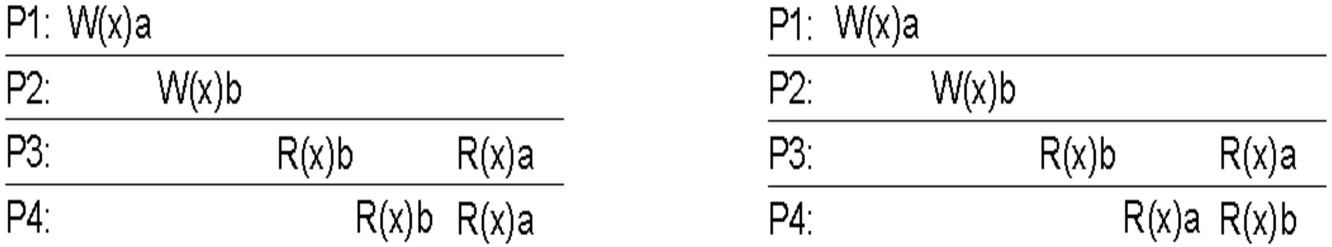
Stat watcher:

```
stat = Read ("season-runs")
discuss sta
```

Eventual Consistency

Sequential Consistency

- weaker than strict/strong consistency
  - All operations are executed in *some* sequential order
  - each process issues operations in program order
    - Any valid interleaving is allowed
    - All agree on the same interleaving
    - Each process preserves its program order



P1:	W(x)a				
P2:		W(x)b			
P3:			R(x)b		R(x)a
P4:				R(x)b	R(x)a

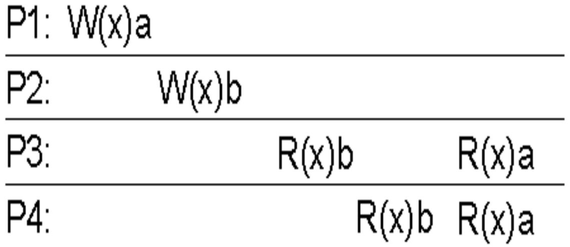
(a)

P1:	W(x)a				
P2:		W(x)b			
P3:			R(x)b		R(x)a
P4:				R(x)a	R(x)b

(b)

Sequential Consistency

- weaker than strict/strong consistency
  - All operations are executed in *some* sequential order
  - each process issues operations in program order
    - Any valid interleaving is allowed
    - All agree on the same interleaving
    - Each process preserves its program order



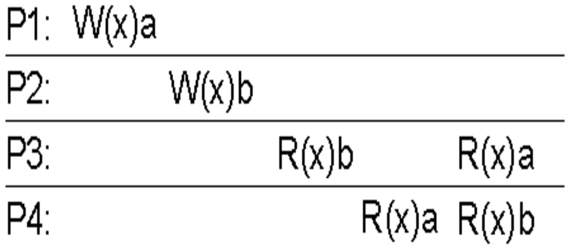
P1: W(x)a

P2: W(x)b

P3: R(x)b R(x)a

P4: R(x)b R(x)a

)



P1: W(x)a

P2: W(x)b

P3: R(x)b R(x)a

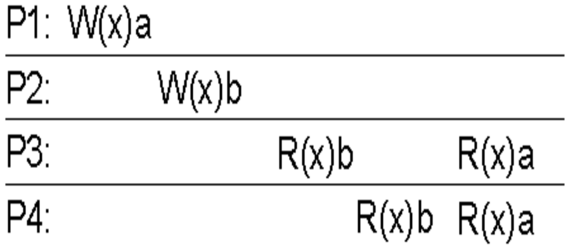
P4: R(x)a R(x)b

(b)

- ***Why is this weaker than strict/strong?***

Sequential Consistency

- weaker than strict/strong consistency
  - All operations are executed in *some* sequential order
  - each process issues operations in program order
    - Any valid interleaving is allowed
    - All agree on the same interleaving
    - Each process preserves its program order



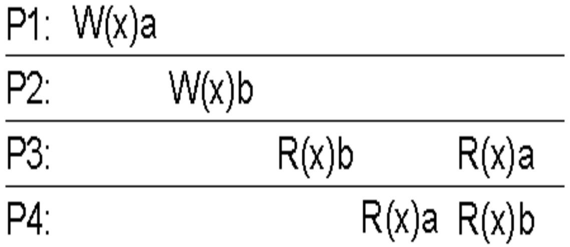
P1: W(x)a

P2: W(x)b

P3: R(x)b R(x)a

P4: R(x)b R(x)a

)



P1: W(x)a

P2: W(x)b

P3: R(x)b R(x)a

P4: R(x)a R(x)b

(b)

- ***Why is this weaker than strict/strong?***
- ***Nothing is said about "most recent write"***

Linearizability

Linearizability

- Assumes sequential consistency *and*
  - If TS(x) < TS(y) then OP(x) should precede OP(y) in the sequence
  - Stronger than sequential consistency
  - Difference between linearizability and serializability?
    - Granularity: reads/writes versus transactions

Linearizability

- Assumes sequential consistency *and*
  - If TS(x) < TS(y) then OP(x) should precede OP(y) in the sequence
  - Stronger than sequential consistency
  - Difference between linearizability and serializability?
    - Granularity: reads/writes versus transactions
- Example:
  - Stay tuned…relevant for lock free data structures
  - Importantly: *a property of concurrent objects*

Causal consistency

Causal consistency

- Causally related writes seen by all processes in same order.

Causal consistency

- Causally related writes seen by all processes in same order.
    - *Causally?*

Causal consistency

- Causally related writes seer
    - *Causally?*

Causal:

If a write produces a value that
causes another write, they are causally related

```
X = 1
if(X > 0) {
   Y = 1
}
```

Causal consistency → all see X=1, Y=1 in same order

Causal consistency

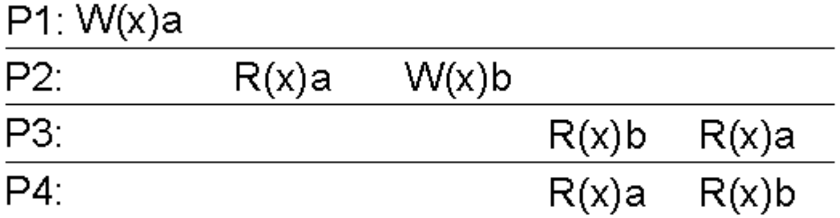
- Causally related writes seen by all processes in same order.
  - *Causally?*

Causal consistency

- Causally related writes seen by all processes in same order.
  - *Causally?*
  - *Concurrent* writes may be seen in different orders on different machines

Causal consistency

- Causally related writes seen by all processes in same order.
  - *Causally?*
  - *Concurrent* writes may be seen in different orders on different machines

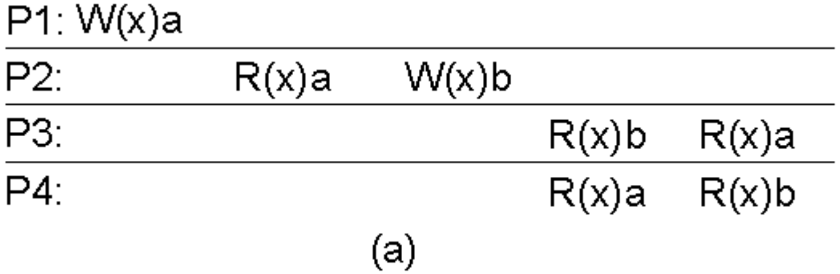


P1:	W(x)a				
P2:		R(x)a	W(x)b		
P3:				R(x)b	R(x)a
P4:				R(x)a	R(x)b

(a)

Causal consistency

- Causally related writes seen by all processes in same order.
    - *Causally?*
    - *Concurrent* writes may be seen in different orders on different machines



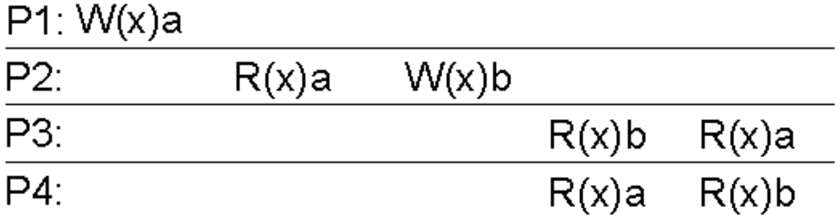
P1:	W(x)a				
P2:		R(x)a	W(x)b		
P3:				R(x)b	R(x)a
P4:				R(x)a	R(x)b

(a)

Not permitted

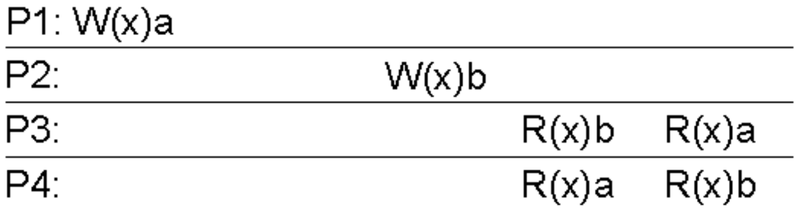
Causal consistency

- Causally related writes seen by all processes in same order.
    - *Causally?*
    - *Concurrent* writes may be seen in different orders on different machines



P1:	W(x)a			
P2:	R(x)a	W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

(a)



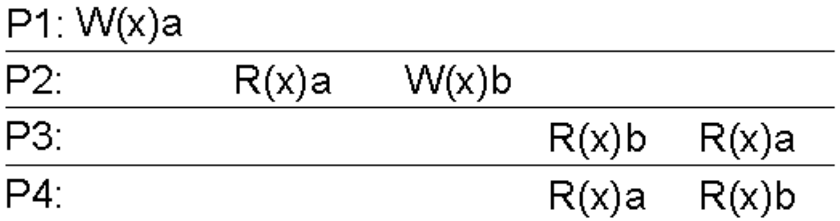
P1:	W(x)a			
P2:		W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

(b)

Not permitted

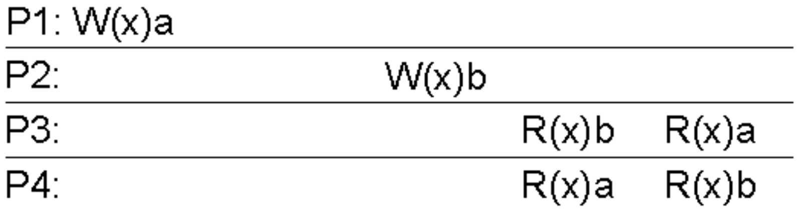
Causal consistency

- Causally related writes seen by all processes in same order.
  - *Causally?*
  - *Concurrent* writes may be seen in different orders on different machines



```
P1: W(x)a
P2:           R(x)a     W(x)b
P3:                              R(x)b   R(x)a
P4:                              R(x)a   R(x)b
```

(a)



```
P1: W(x)a
P2:                 W(x)b
P3:                          R(x)b   R(x)a
P4:                          R(x)a   R(x)b
```

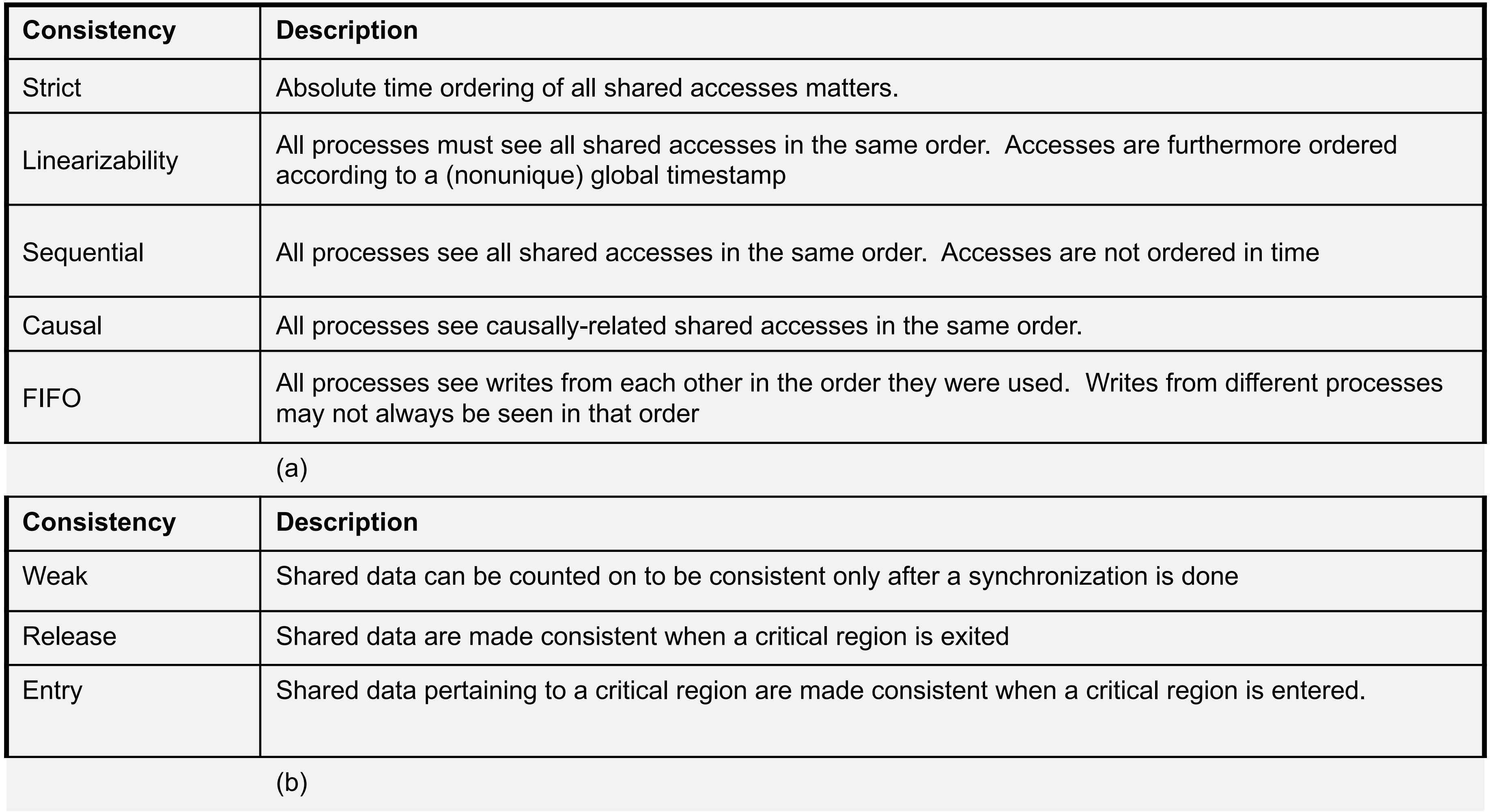
(b)

Not permitted

Permitted

Consistency models summary

Consistency models summary



Consistency	Description
Strict	Absolute time ordering of all shared accesses matters.
Linearizability	All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp
Sequential	All processes see all shared accesses in the same order. Accesses are not ordered in time
Causal	All processes see causally-related shared accesses in the same order.
FIFO	All processes see writes from each other in the order they were used. Writes from different processes may not always be seen in that order

(a)

Consistency	Description
Weak	Shared data can be counted on to be consistent only after a synchronization is done
Release	Shared data are made consistent when a critical region is exited
Entry	Shared data pertaining to a critical region are made consistent when a critical region is entered.

(b)