Programming at Fast Scale: Consistency + Lock Freedom

cs378h

Today

Questions?

Administrivia

- Project Proposal Due Today!

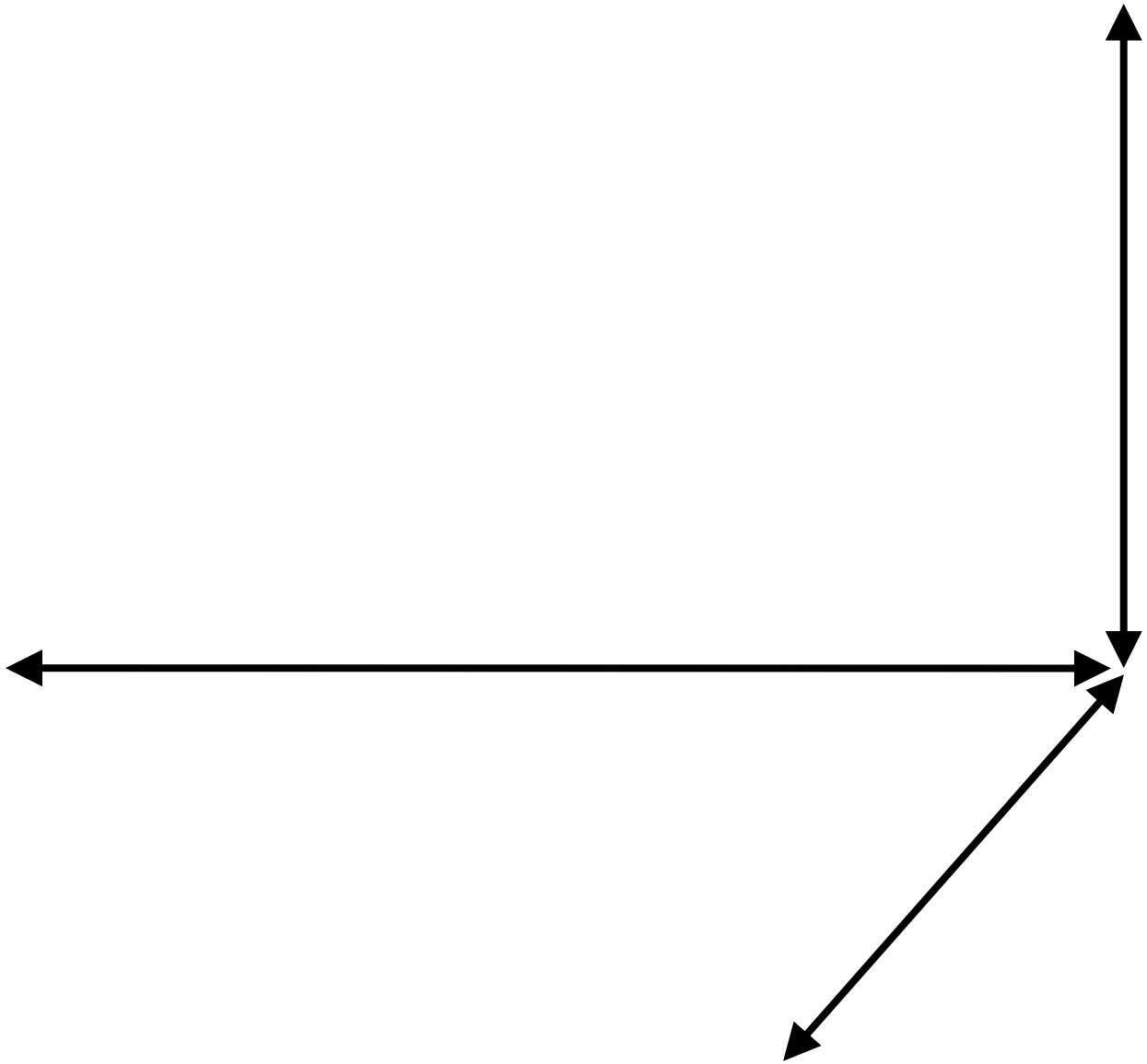
Agenda:

- Consistency
- Lock Freedom

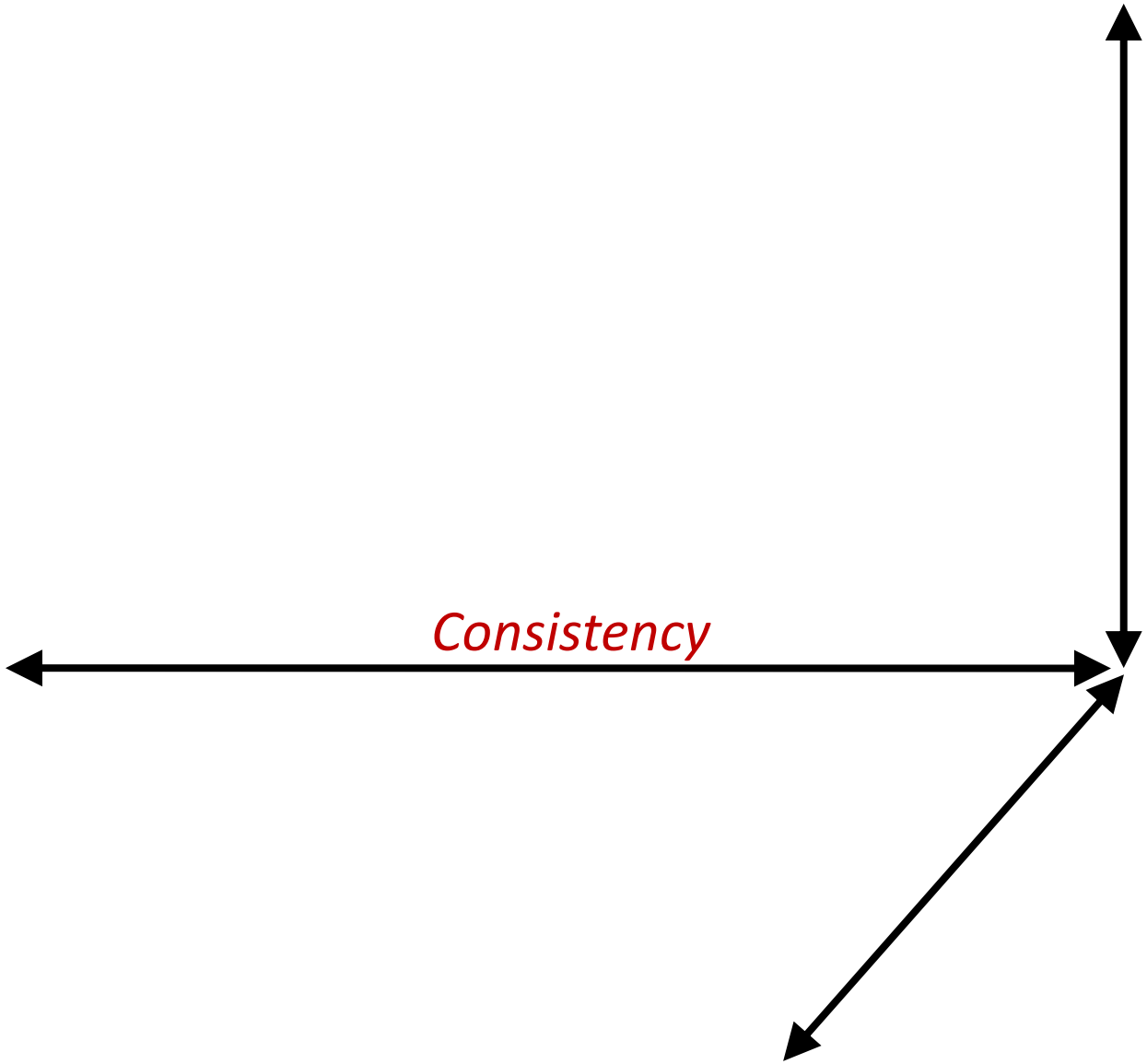
Faux Quiz Questions

- What is the CAP theorem? What does “PACELP” stand for and how does it relate to CAP?
- What is the difference between ACID and BASE?
- Why do NoSQL systems claim to be more horizontally scalable than RDMBSes? List some features NoSQL systems give up toward this goal?
- What is eventual consistency? Give a concrete example of how of why it causes a complex programming model (relative to a strongly consistent model).
- Compare and contrast Key-Value, Document, and Wide-column Stores
- Define and contrast the following consistency properties:
  - strong consistency, eventual consistency, consistent prefix, monotonic reads, read-my-writes, bounded staleness
- What is causal consistency?
- What is chain replication?
- What is obstruction freedom, wait freedom, lock freedom?
- How can one compose lock free data structures?
- Why should I want a lock free hash table instead of a fine-grain lock-based one?
- What is the difference between linearizability and strong consistency? Between linearizability and serializability?
- What is the ABA problem? Give an example.
- How do lock-free data structures deal with the “inconsistent view” problem?

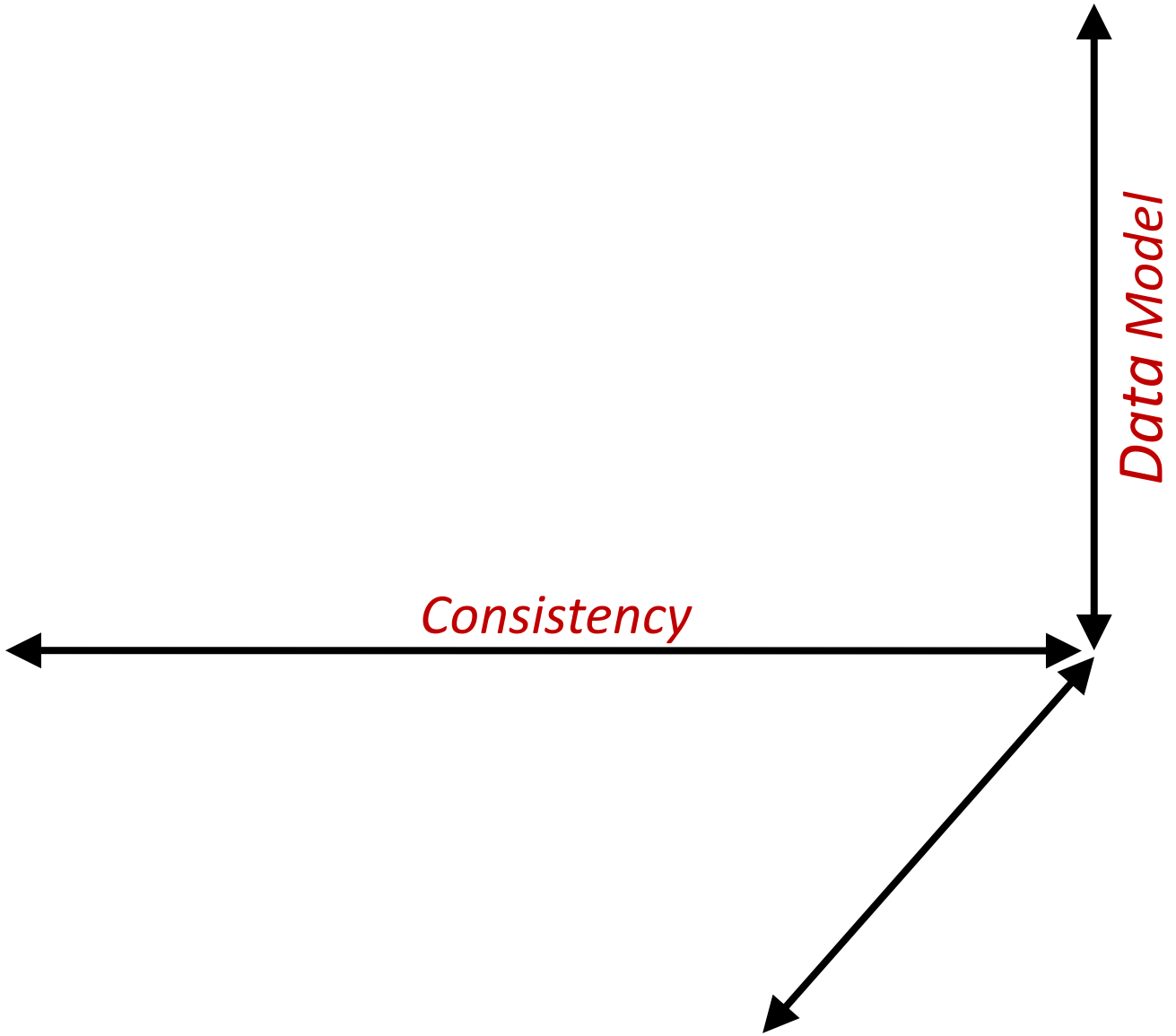
Another Framework



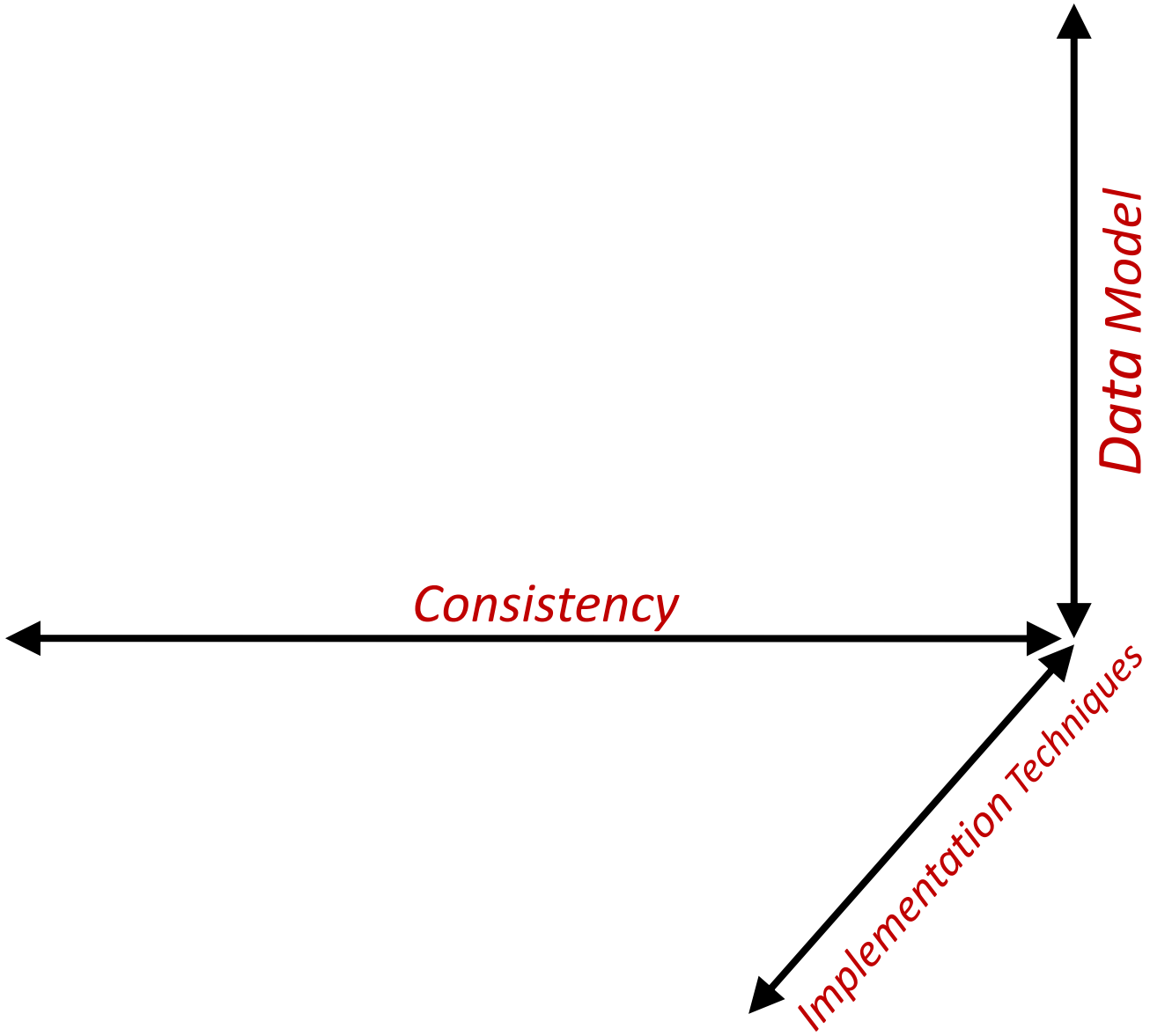
Another Framework



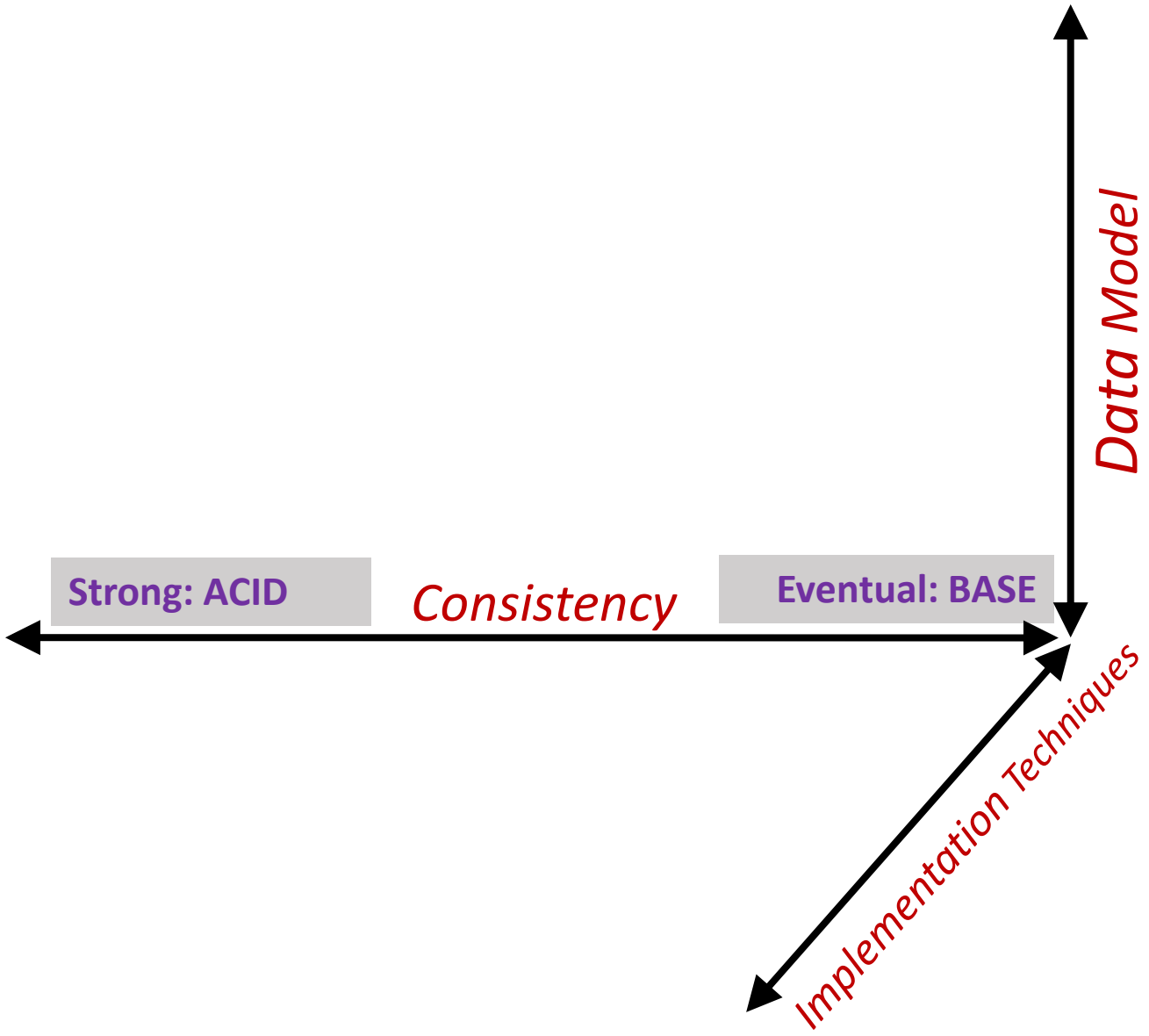
Another Framework



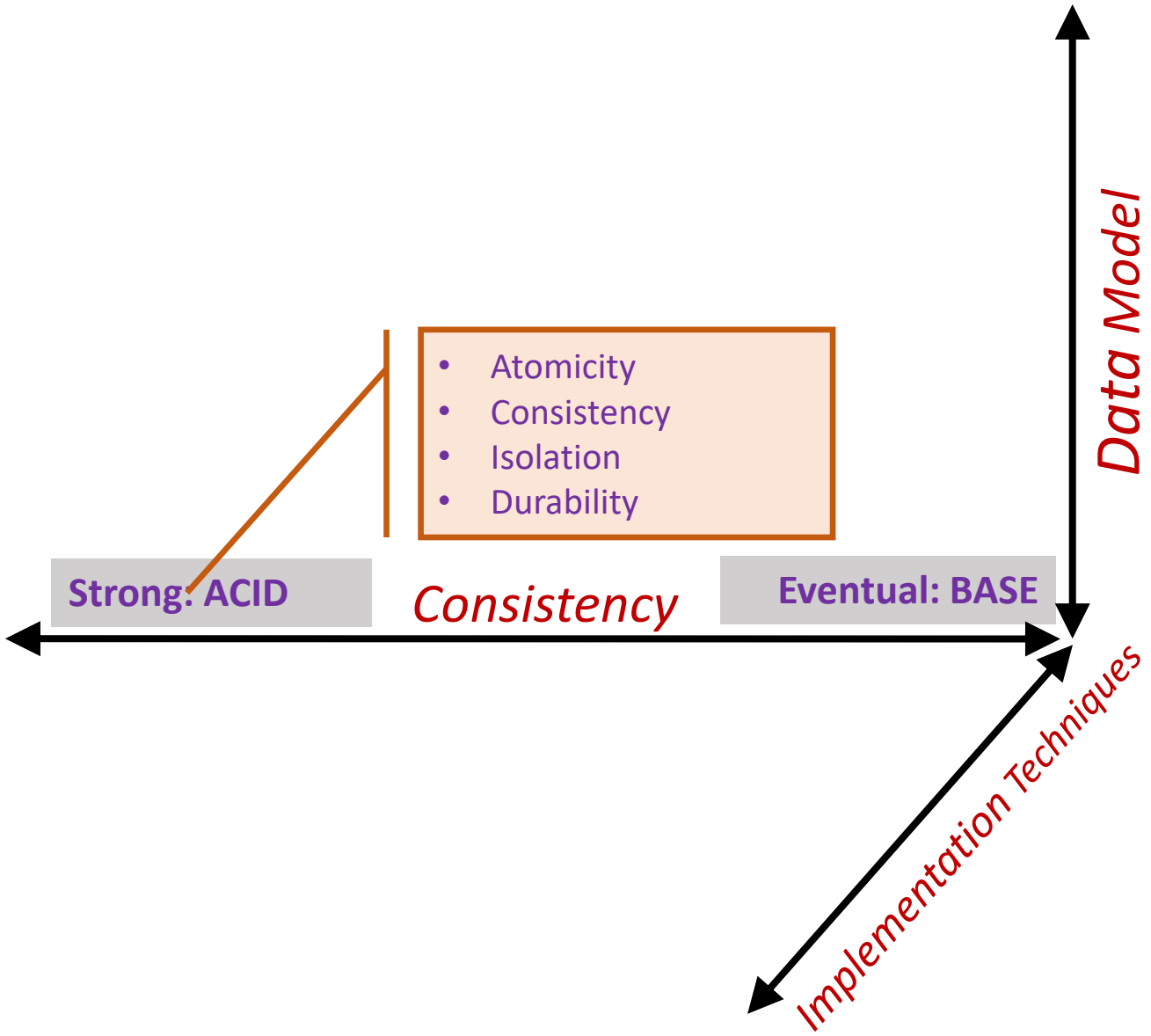
Another Framework



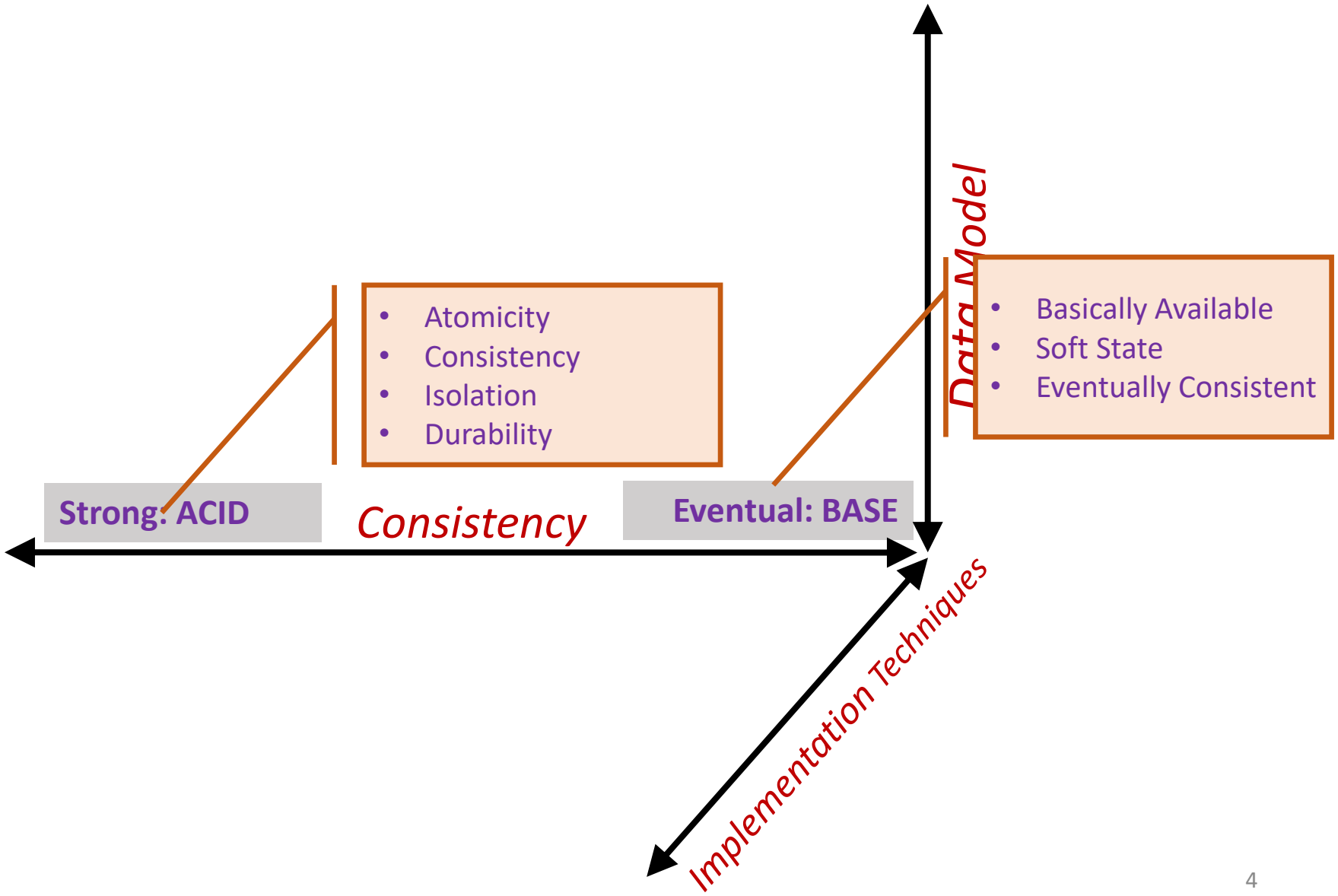
Another Framework



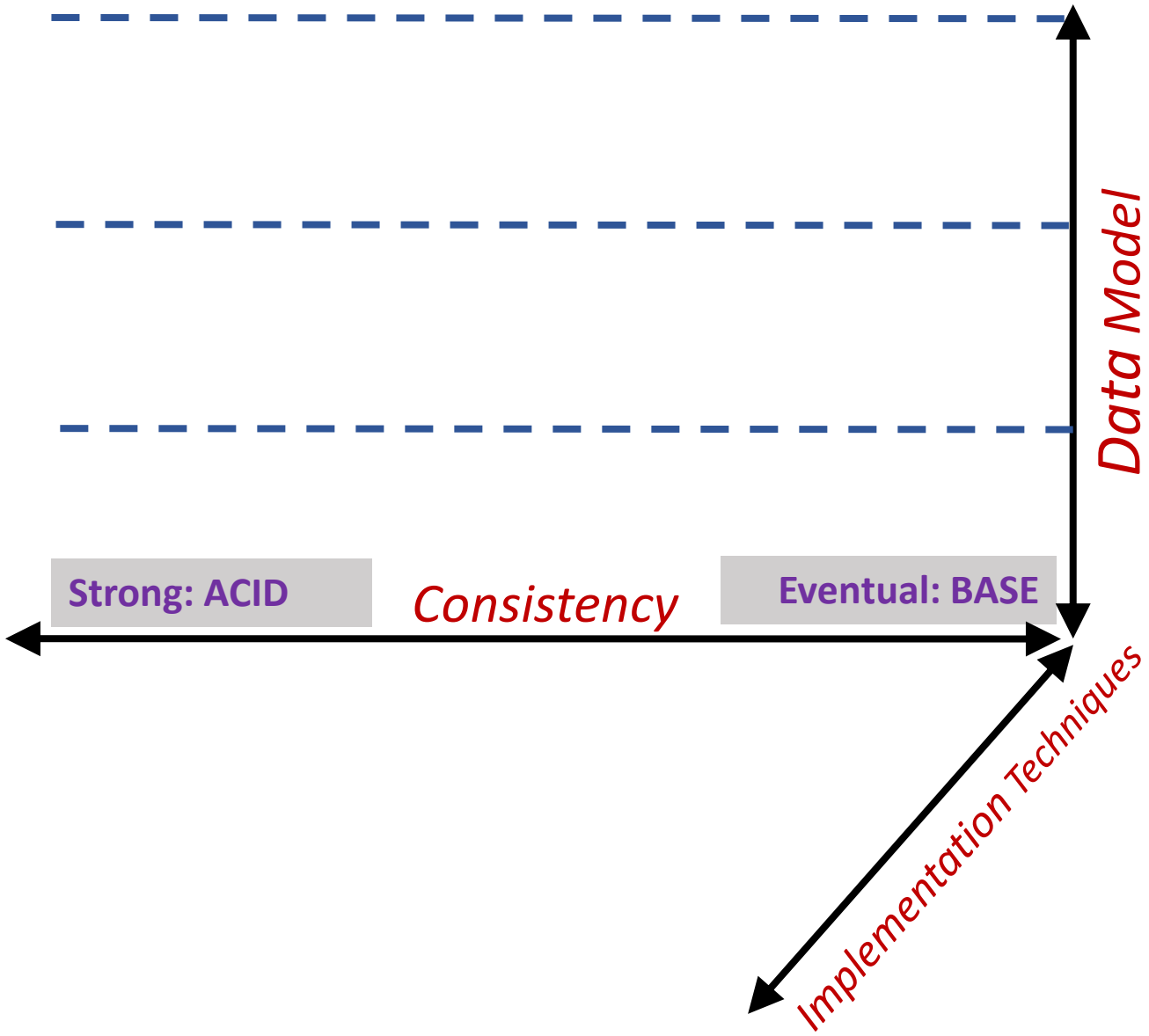
Another Framework



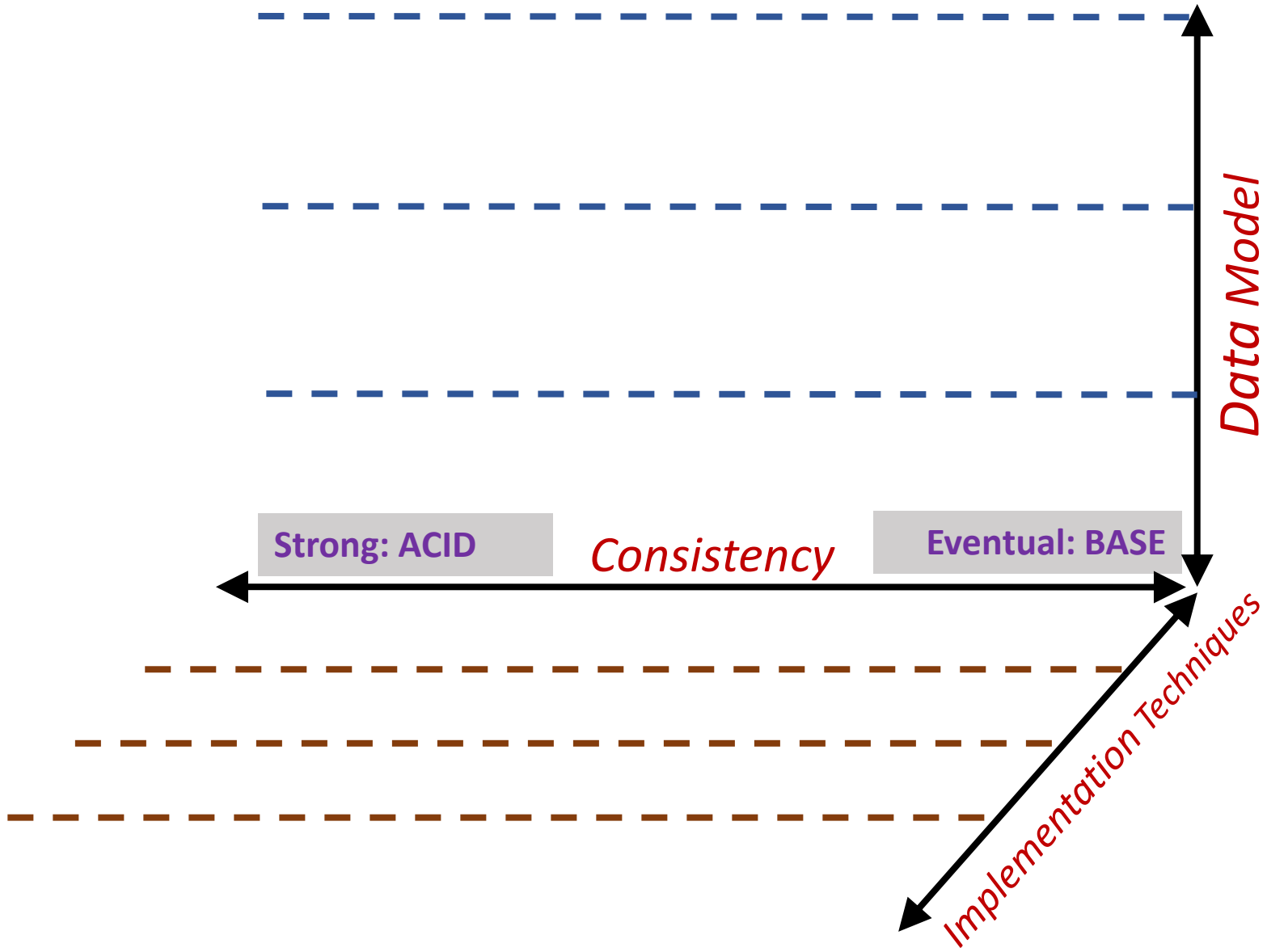
Another Framework



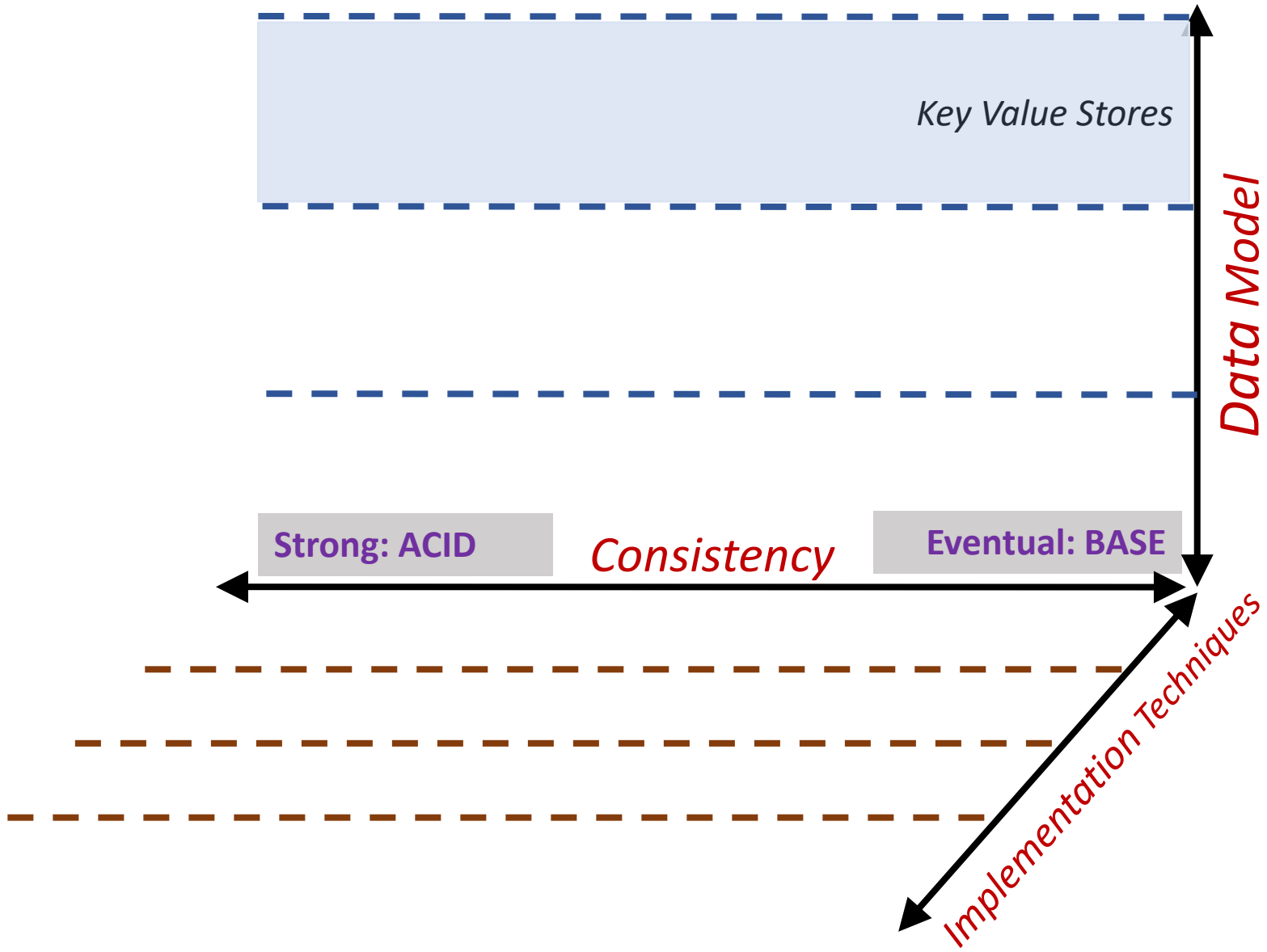
Another Framework



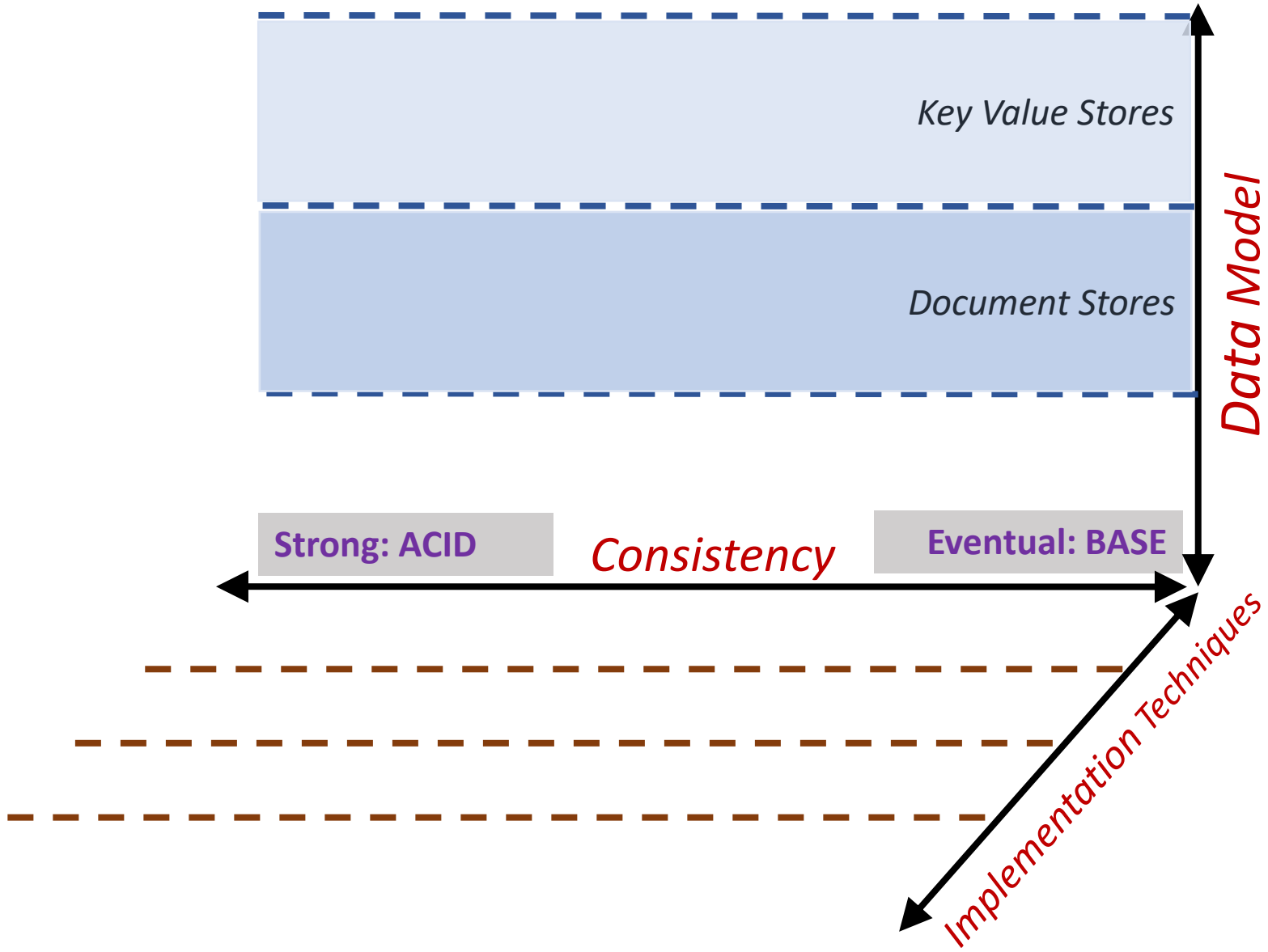
Another Framework



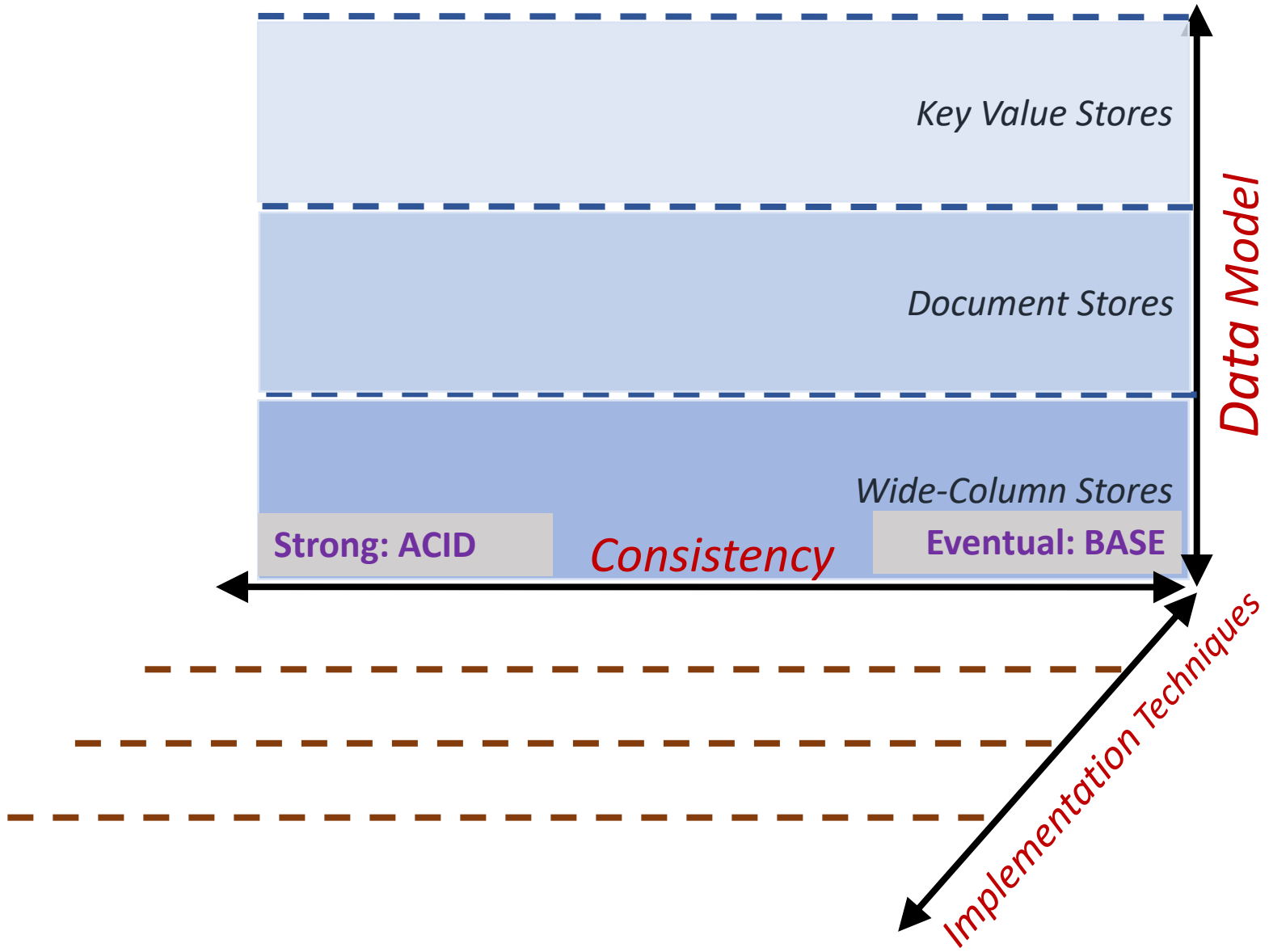
Another Framework



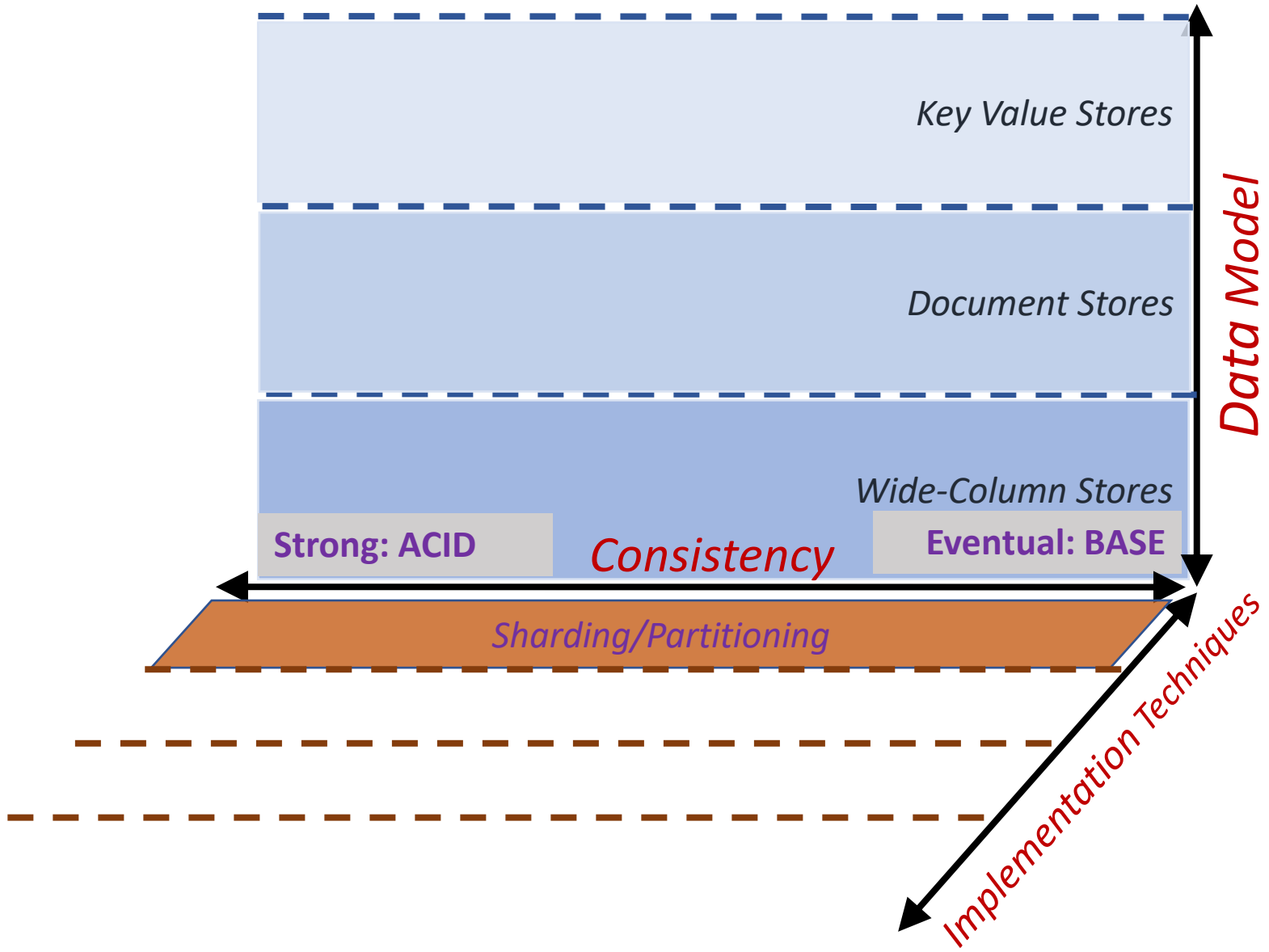
Another Framework



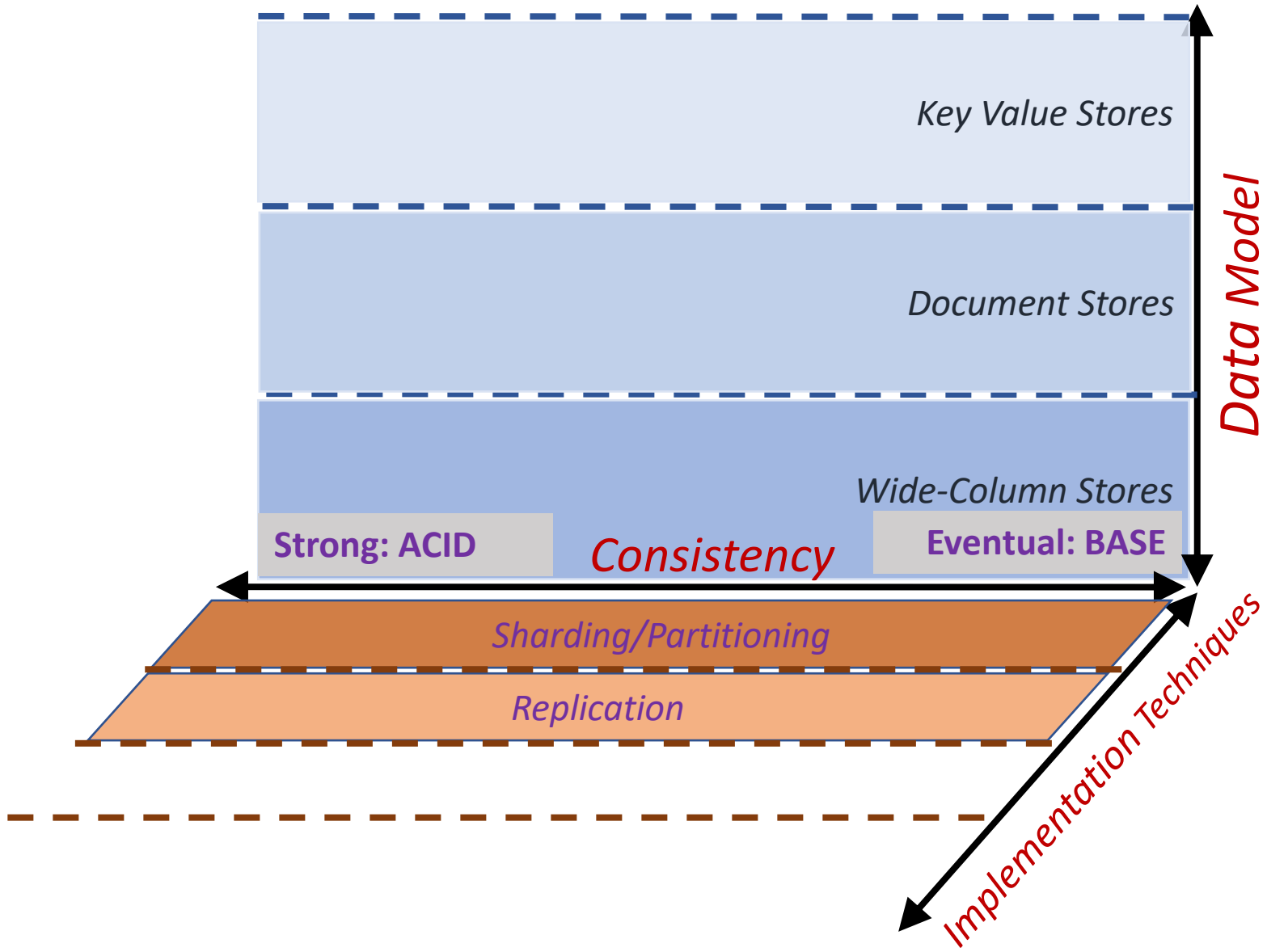
Another Framework



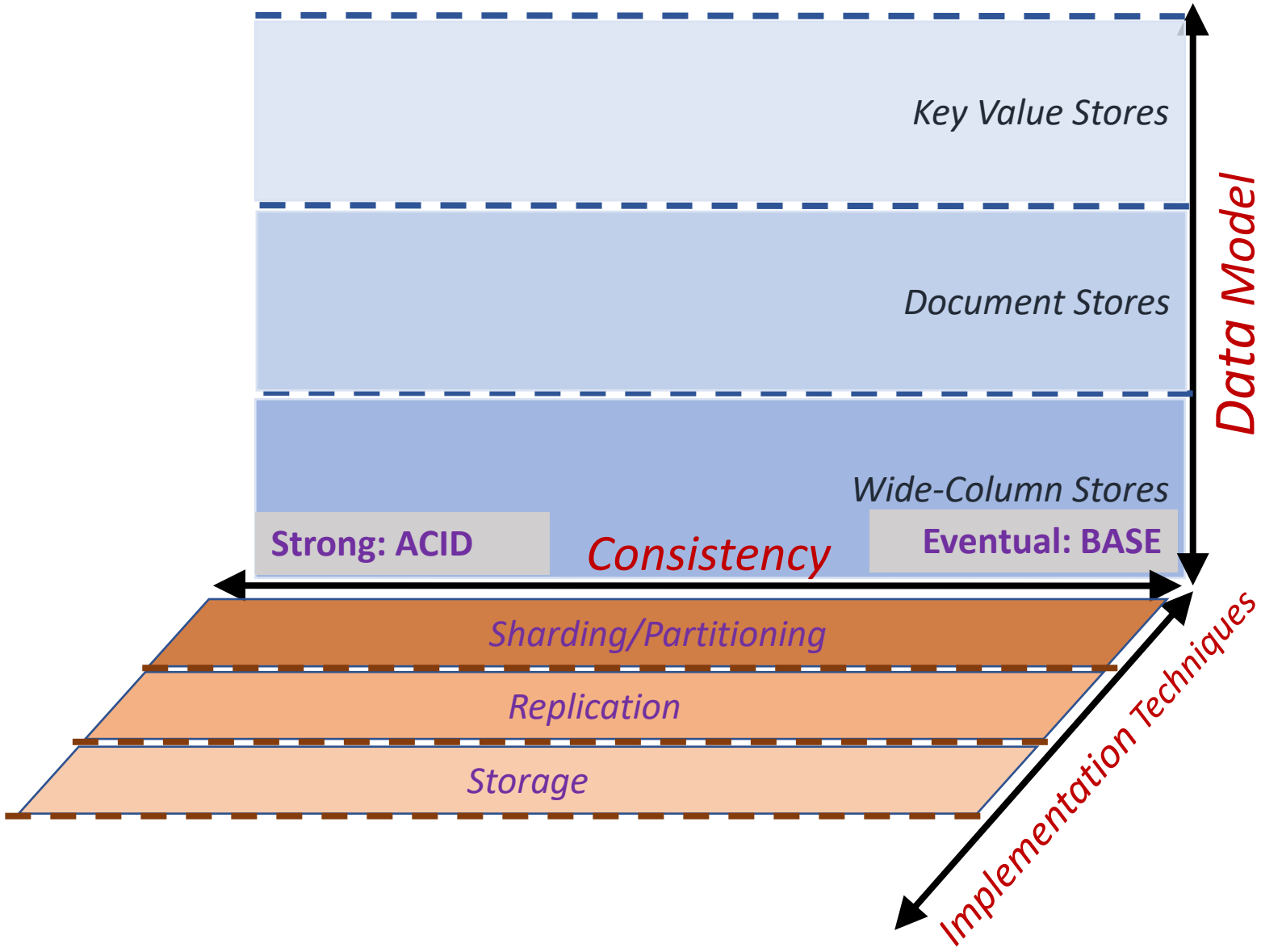
Another Framework



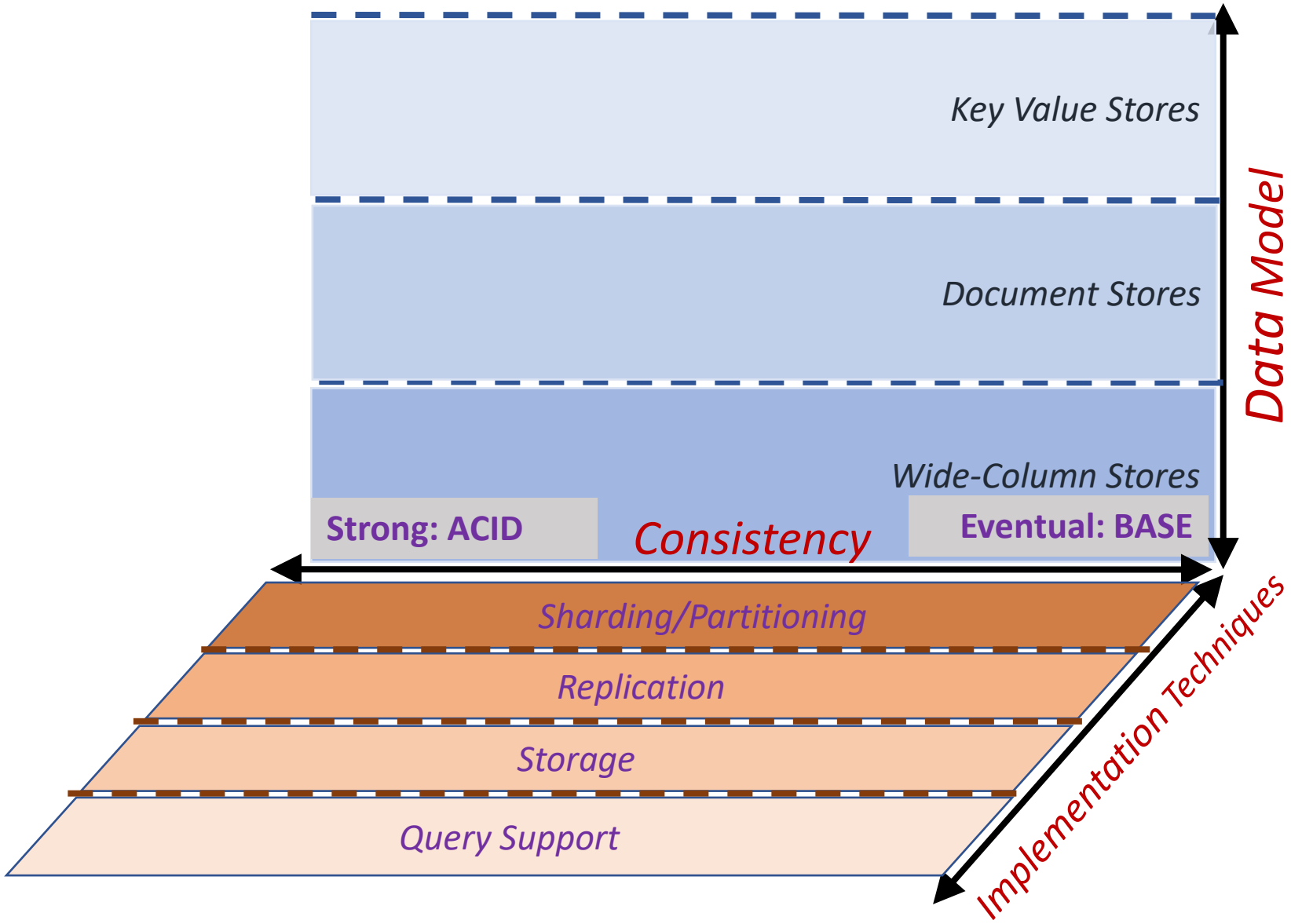
Another Framework



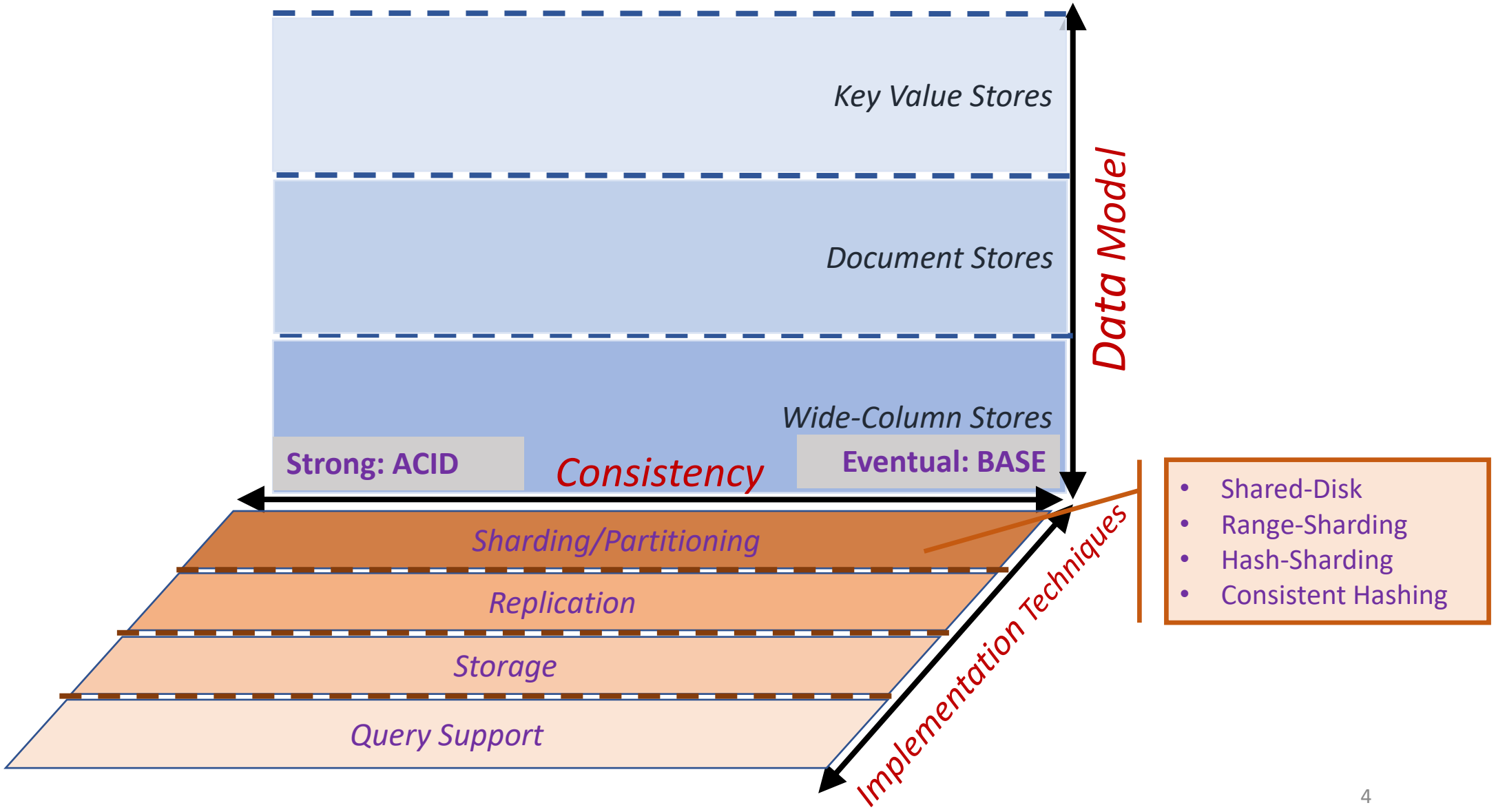
Another Framework



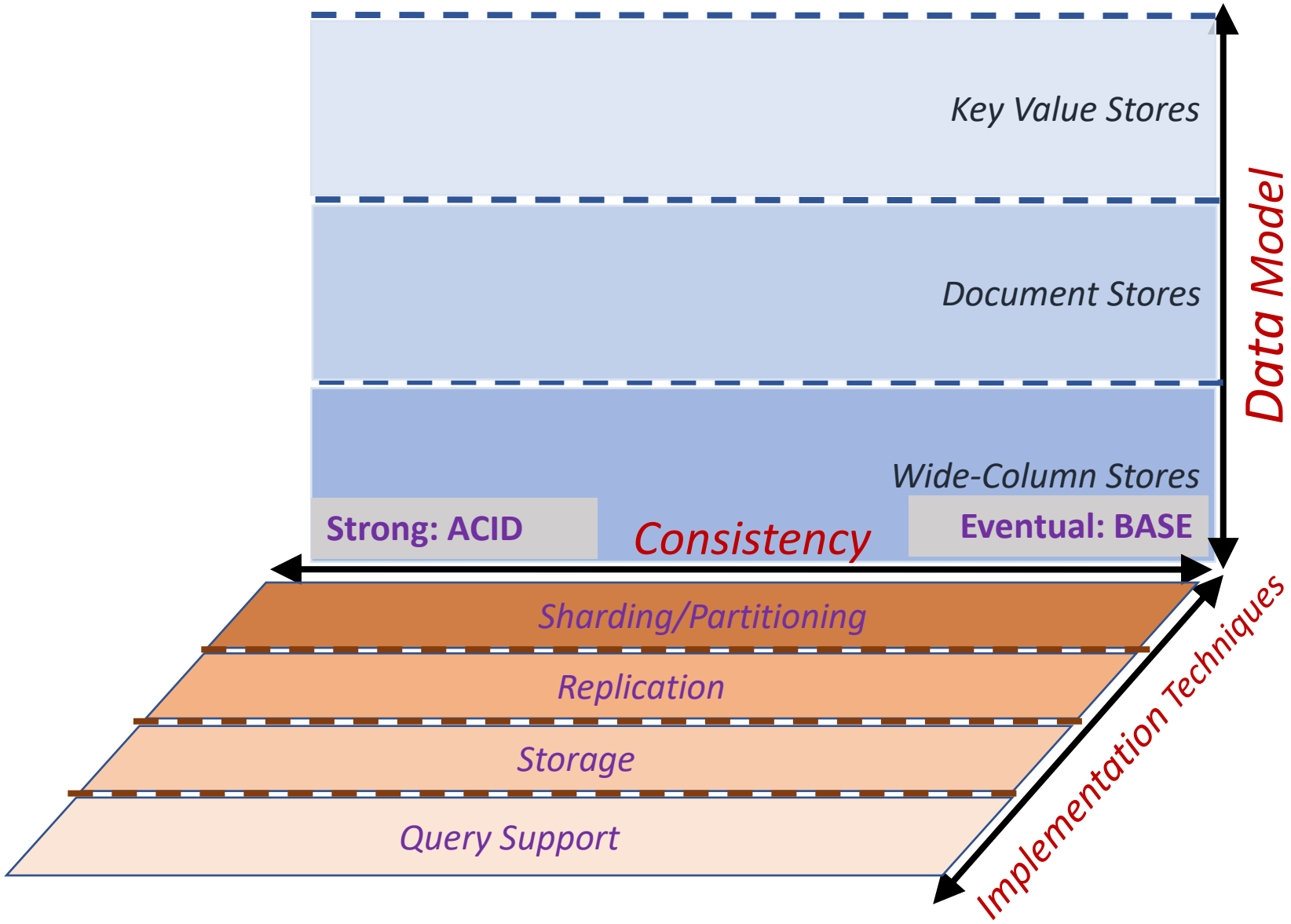
Another Framework



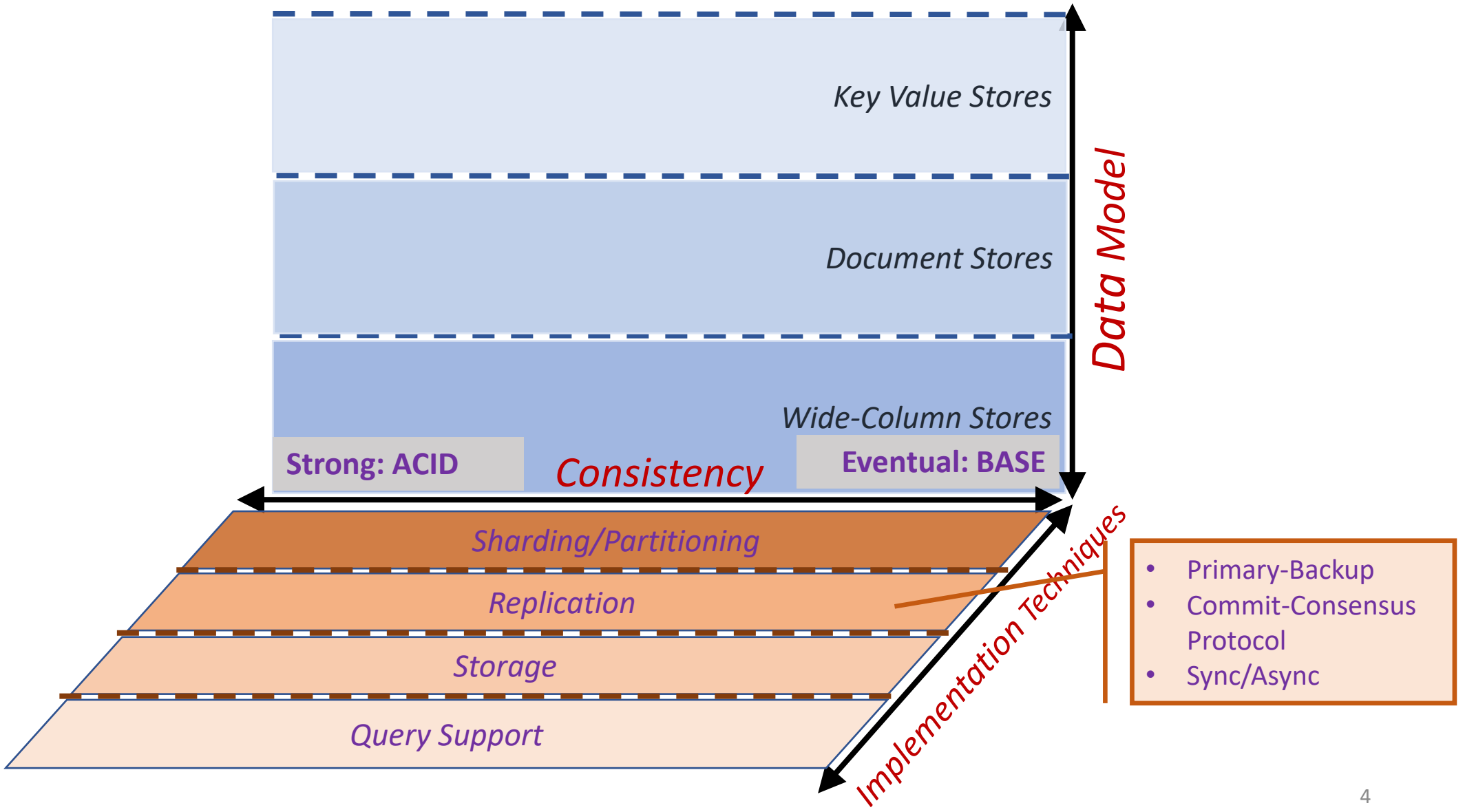
Another Framework



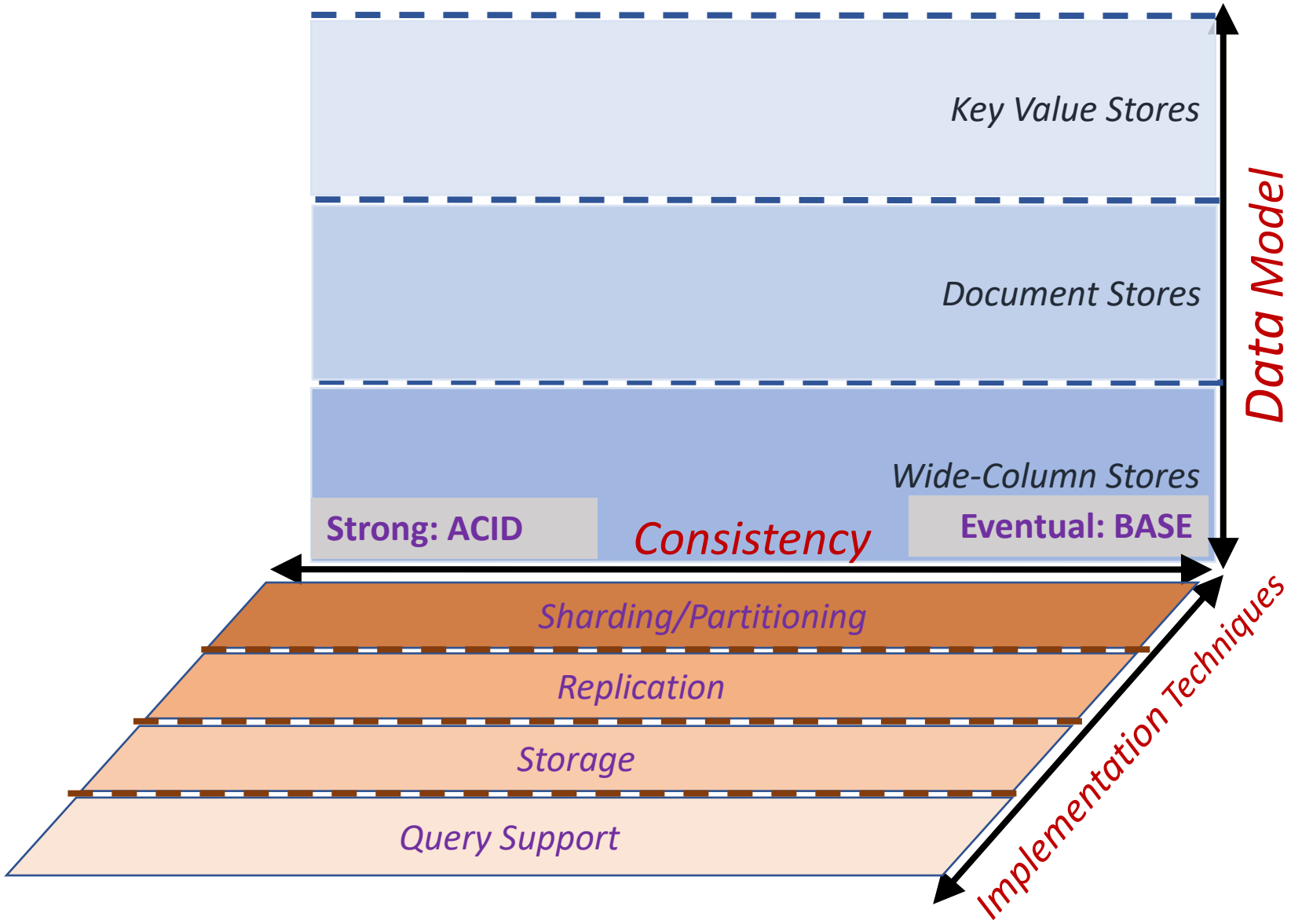
Another Framework



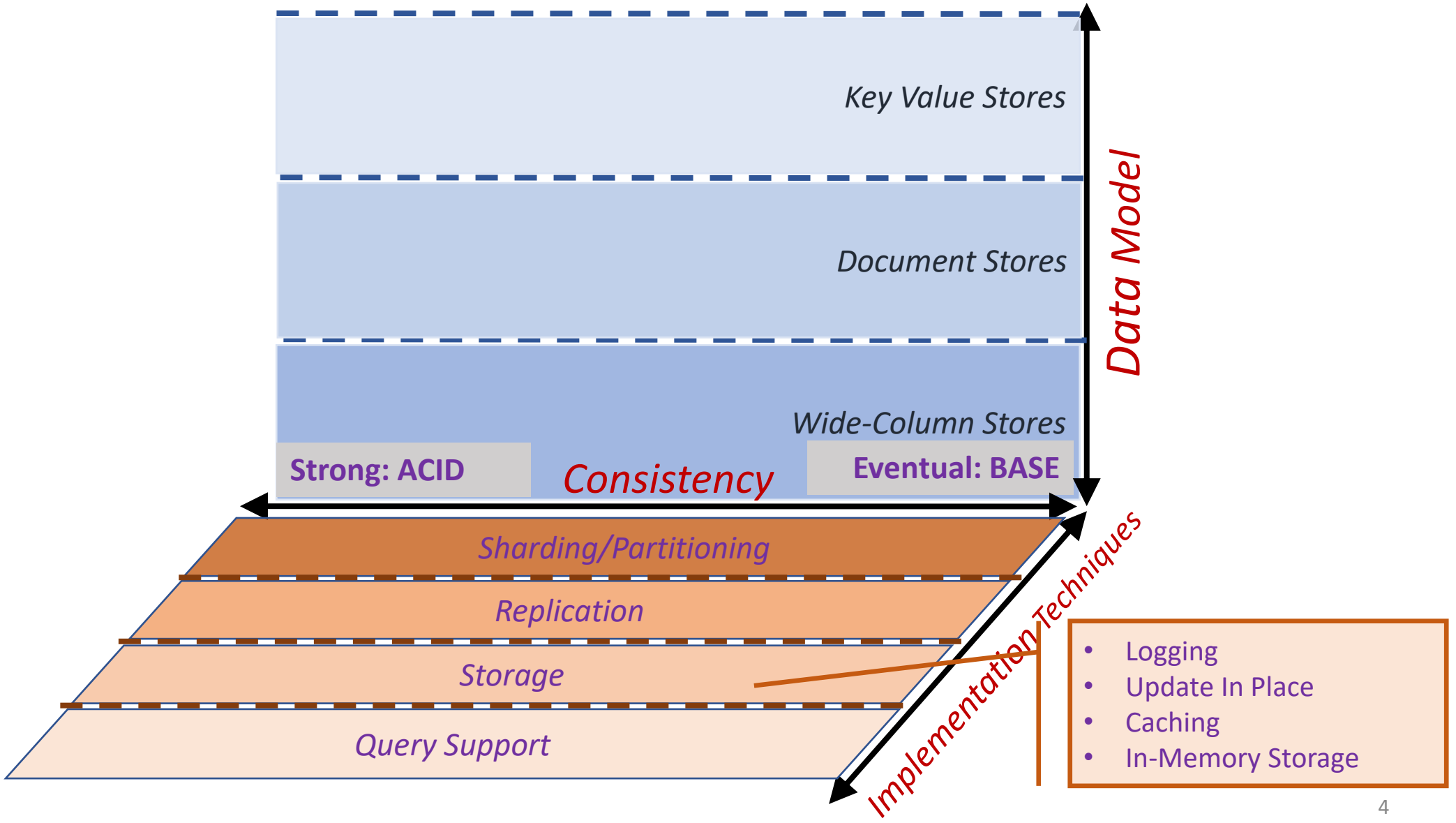
Another Framework



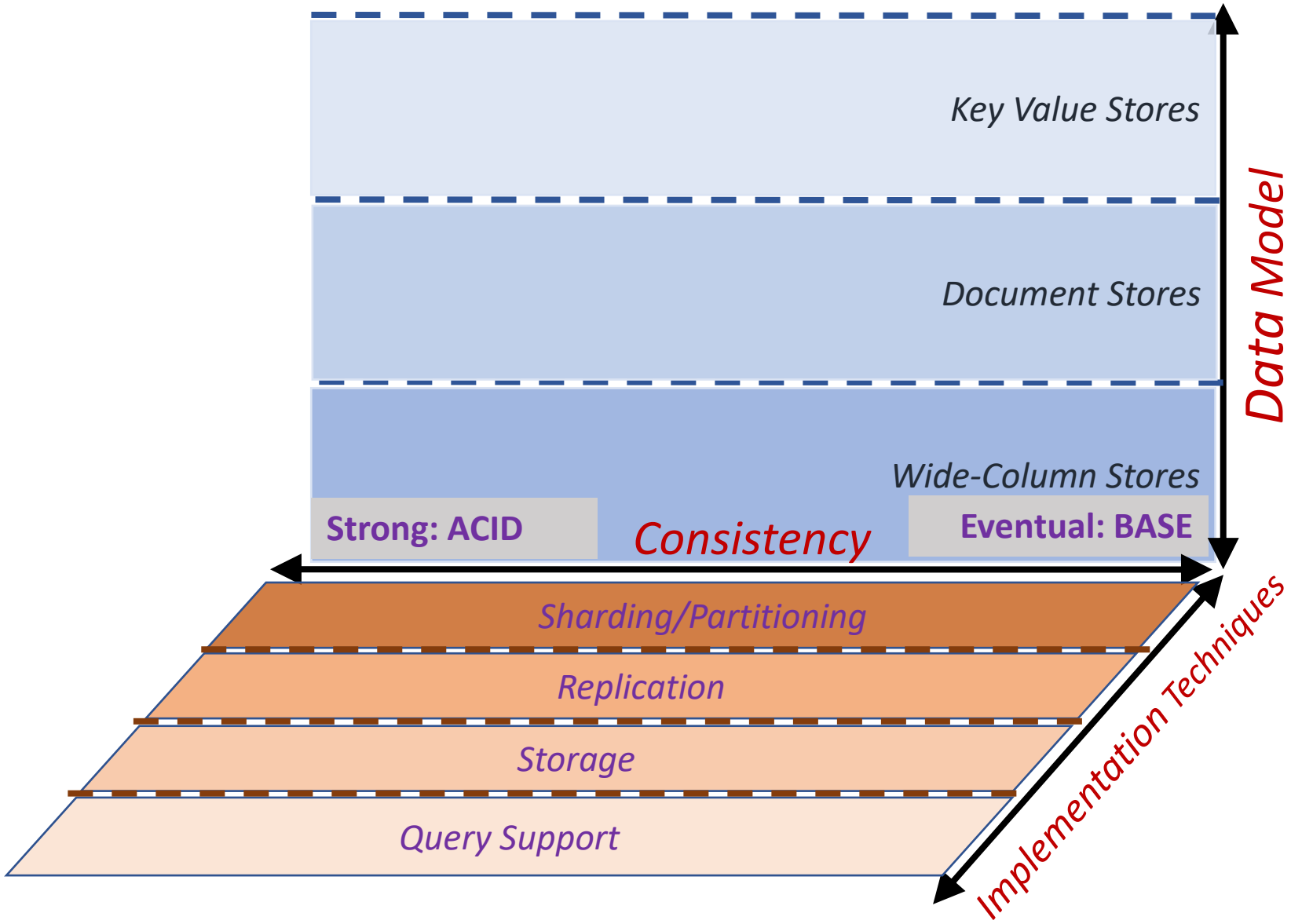
Another Framework



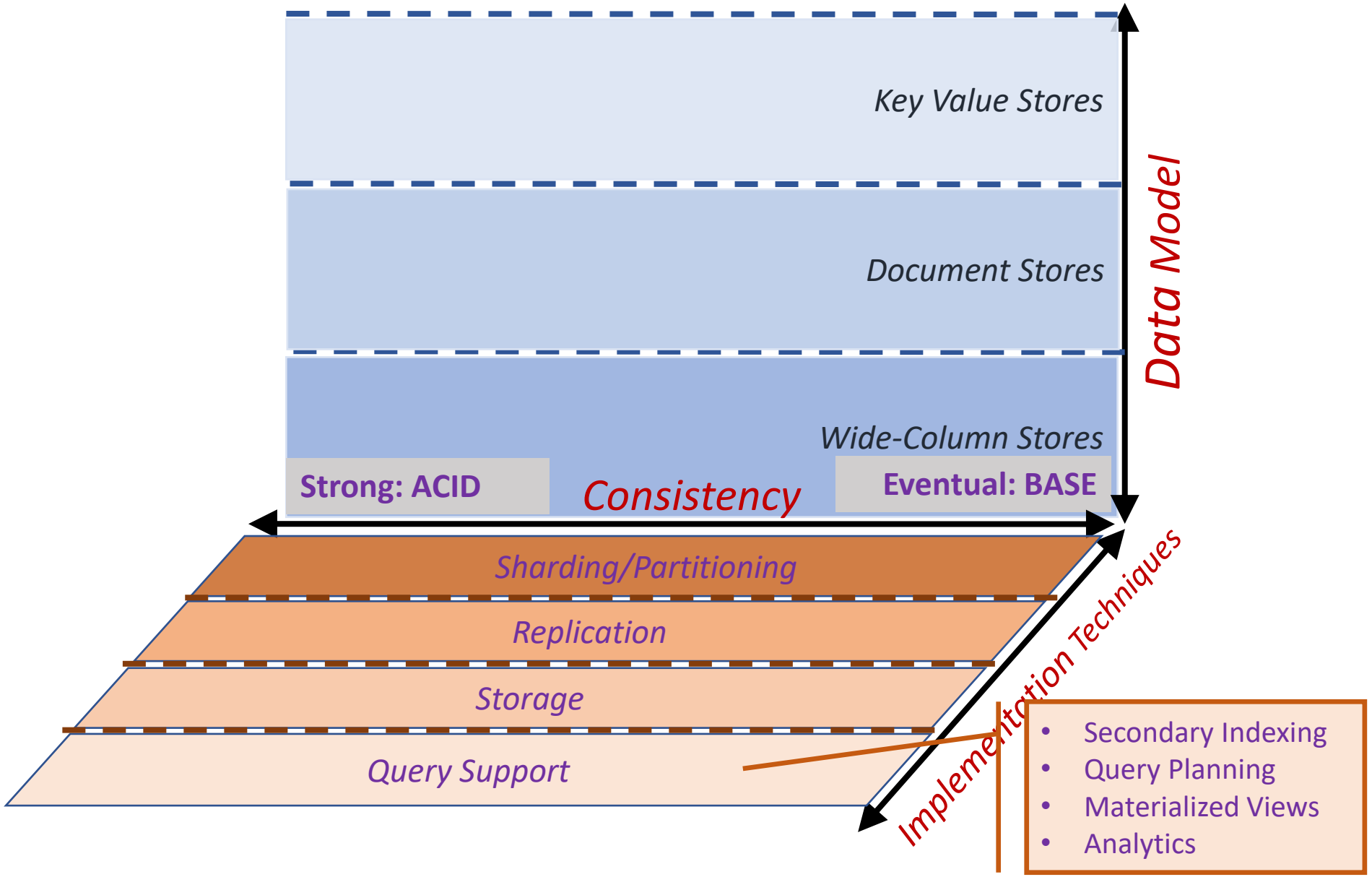
Another Framework



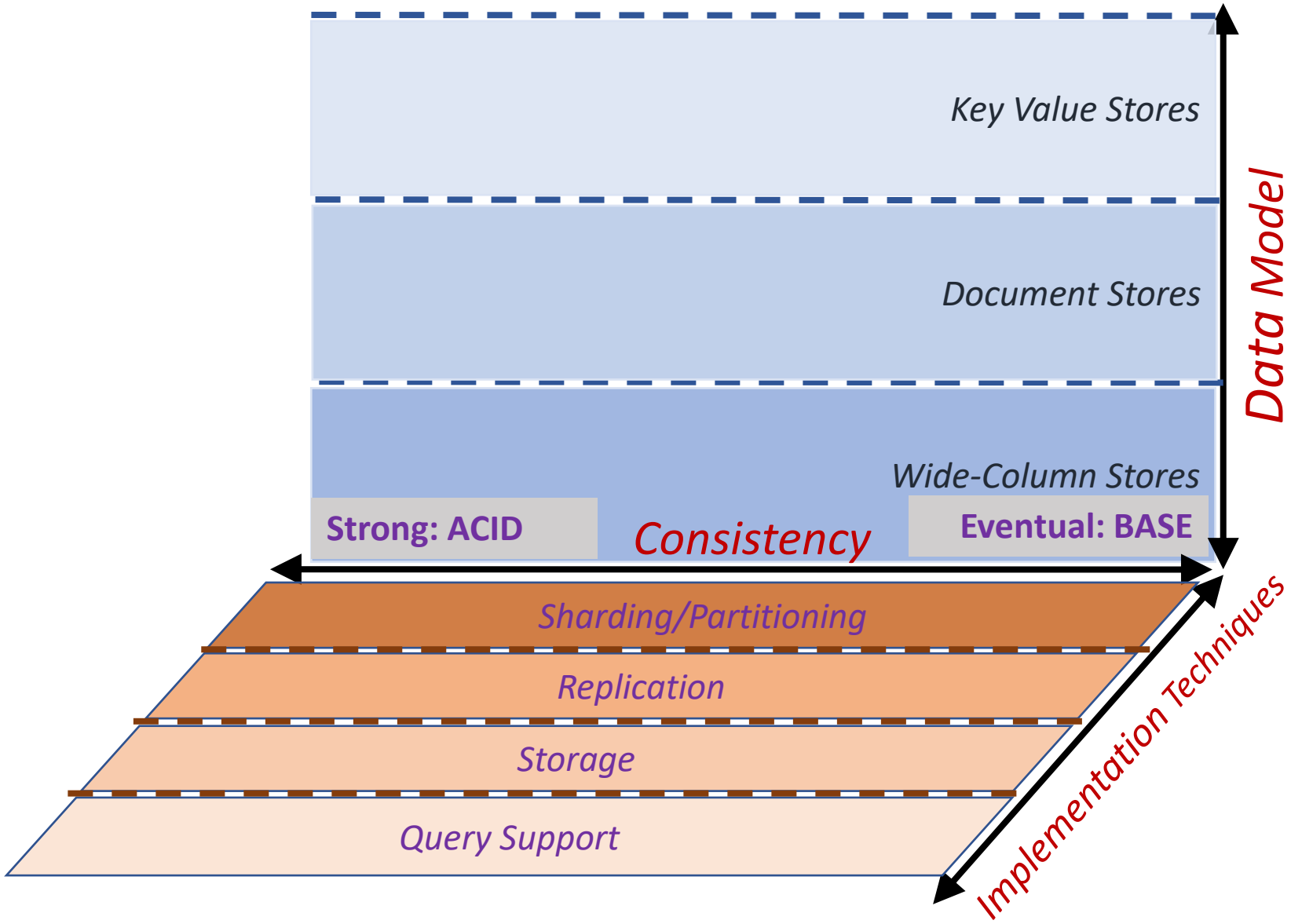
Another Framework



Another Framework



Another Framework



Another Framework

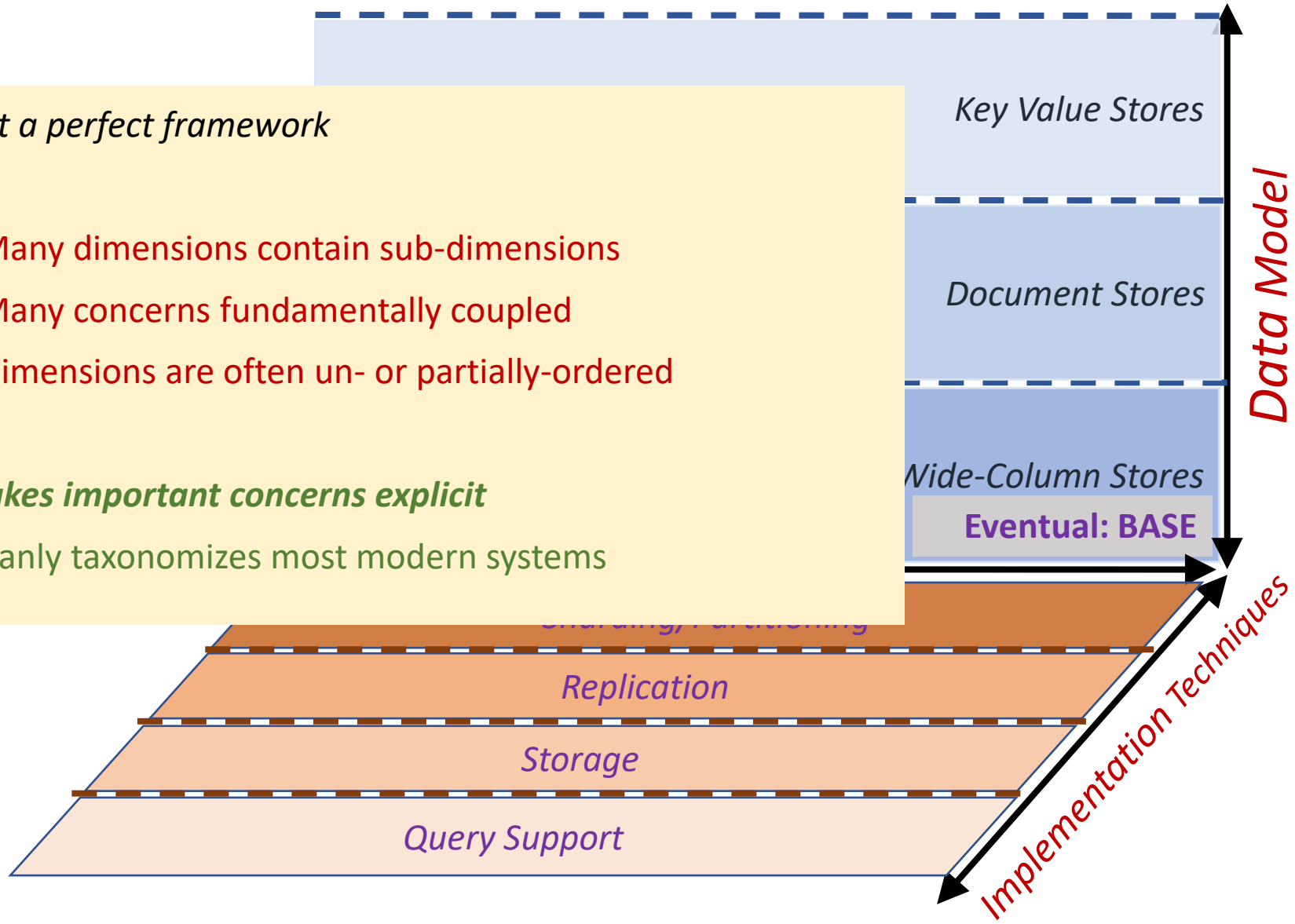
Still not a perfect framework

Cons:

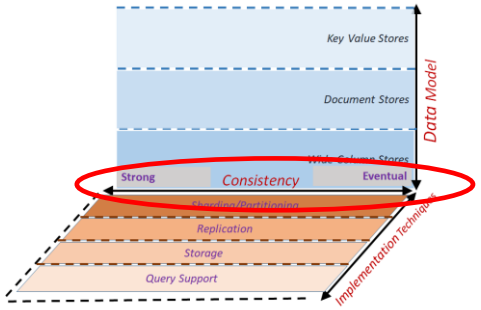
- Many dimensions contain sub-dimensions
- Many concerns fundamentally coupled
- Dimensions are often un- or partially-ordered

Pros:

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

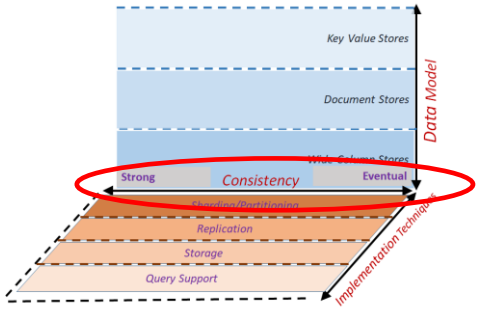


Consistency



col	col	col_2	. . .	col_c
0	1			

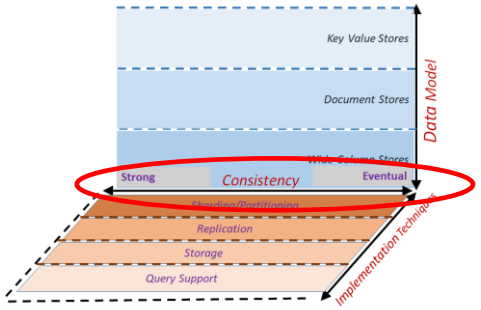
Consistency



col	col	col_2	. . .	col_c
0	1			

How to keep data in sync?

Consistency

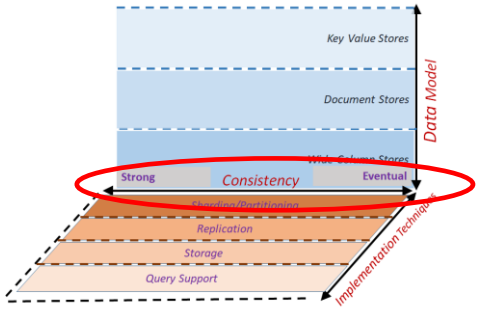


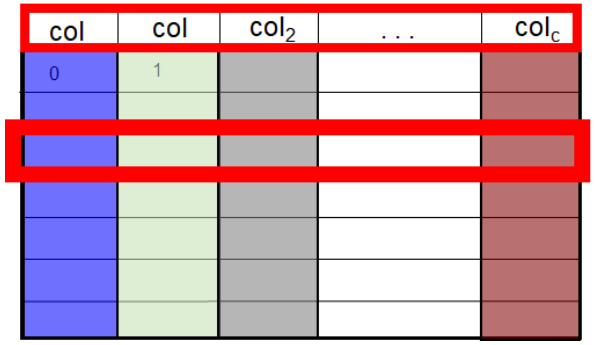
col	col	col_2	. . .	col_c
0	1			

How to keep data in sync?

- Partitioning → single row spread over multiple machines

Consistency

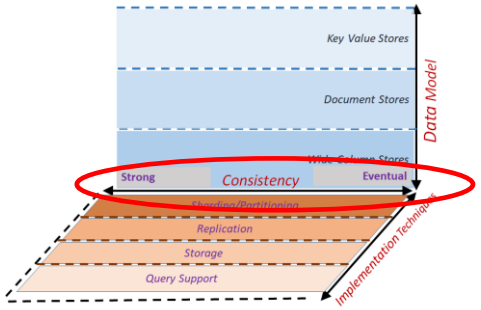


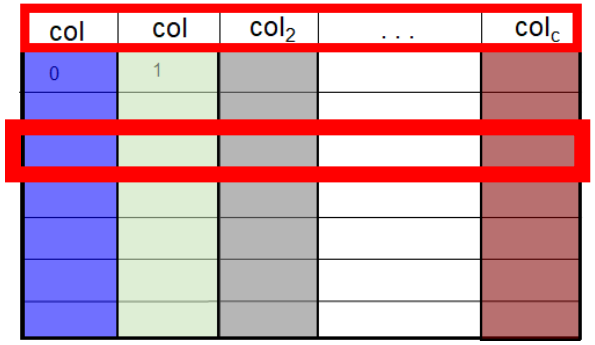


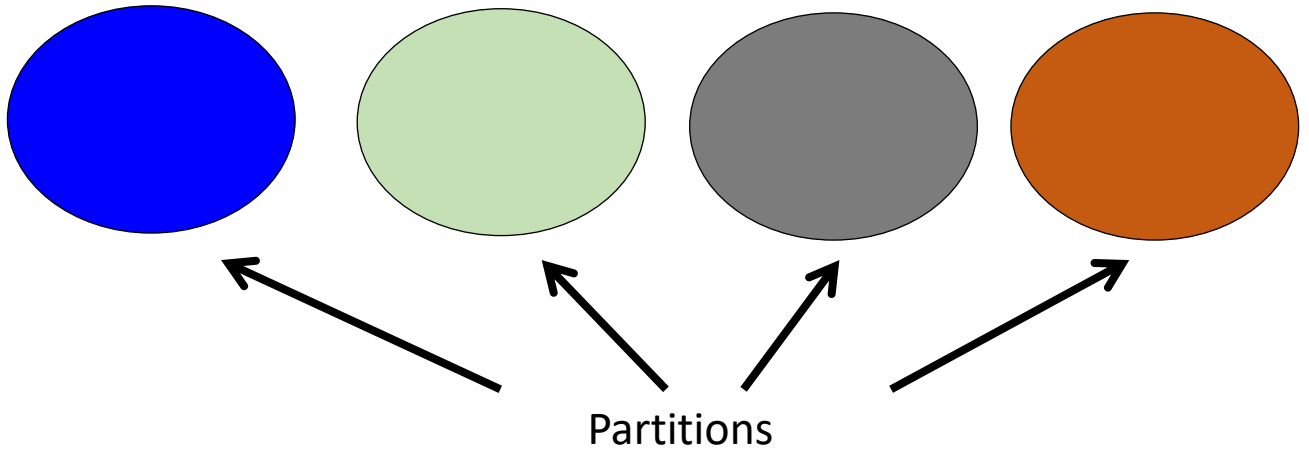
How to keep data in sync?

- Partitioning → single row spread over multiple machines

Consistency



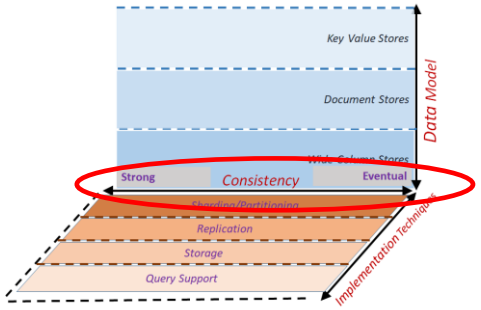


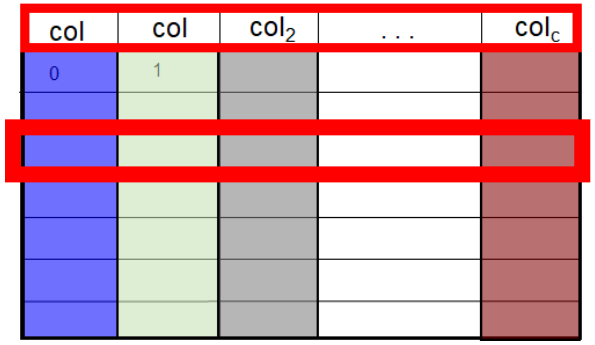


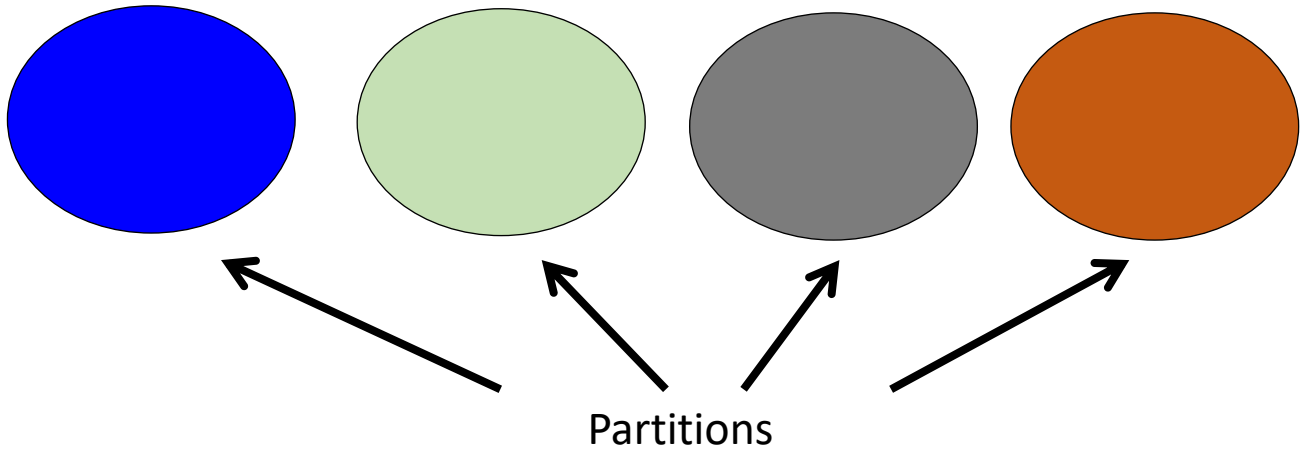
How to keep data in sync?

- Partitioning → single row spread over multiple machines

Consistency



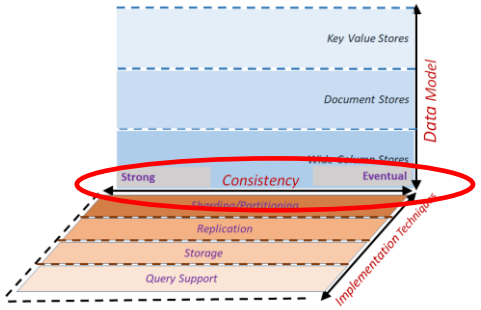


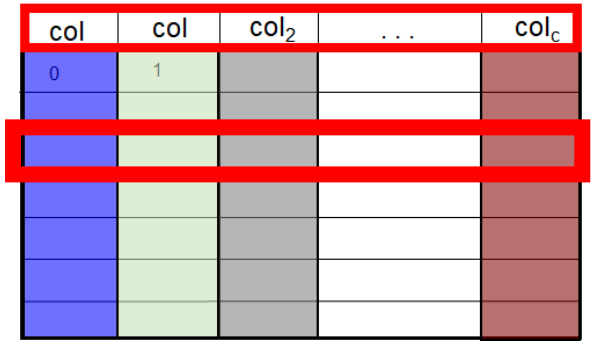


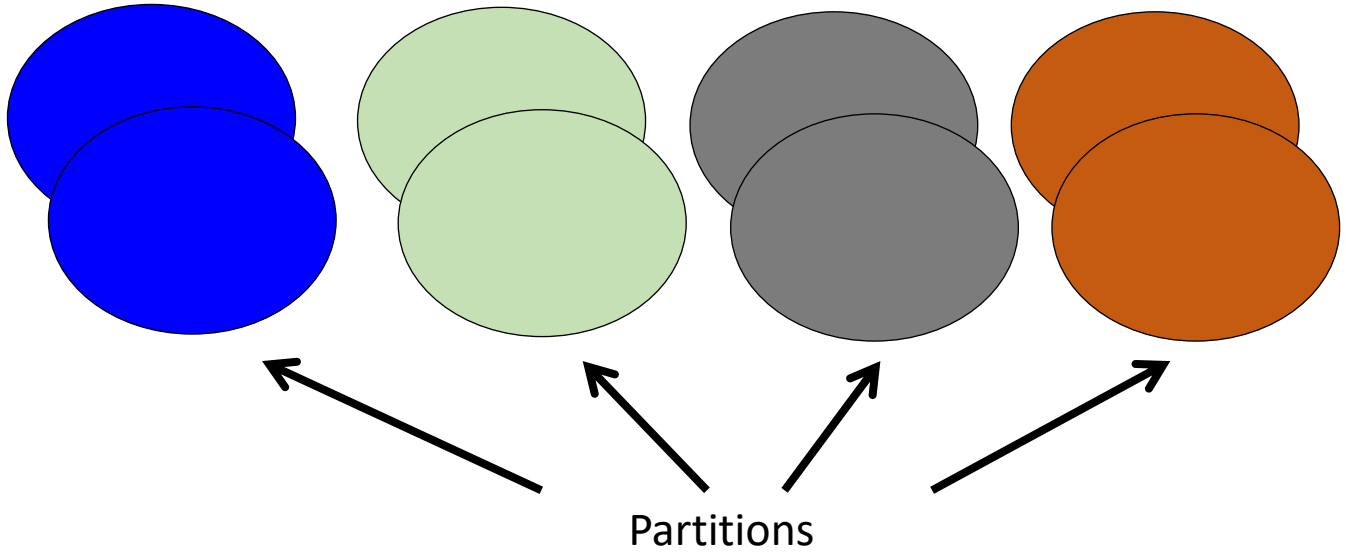
How to keep data in sync?

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

Consistency



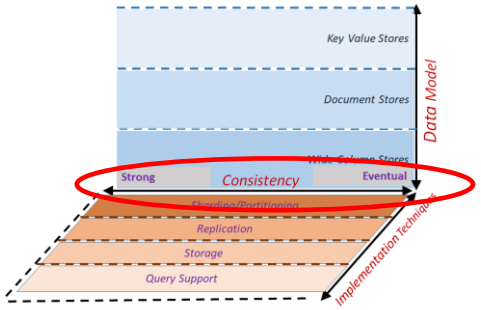


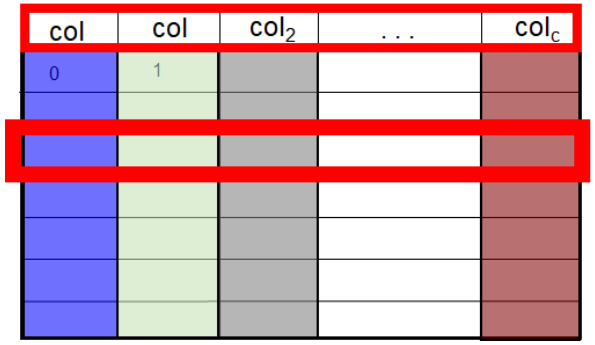


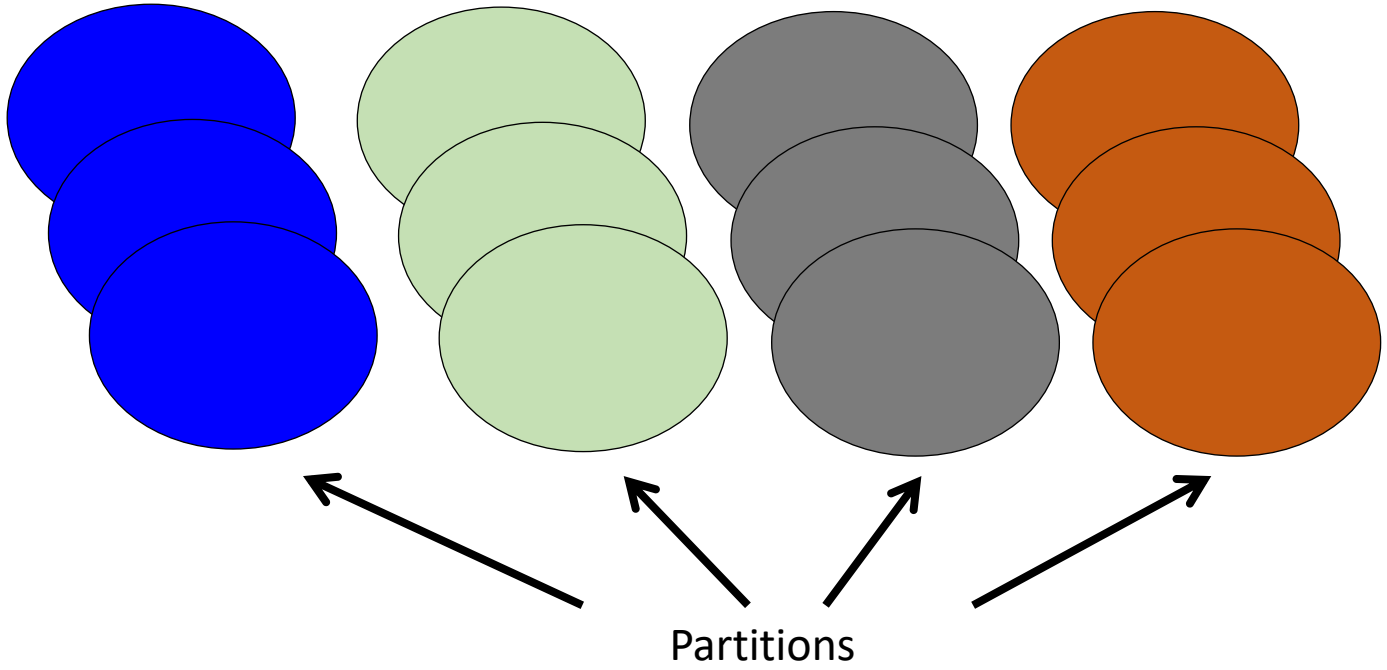
How to keep data in sync?

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

Consistency



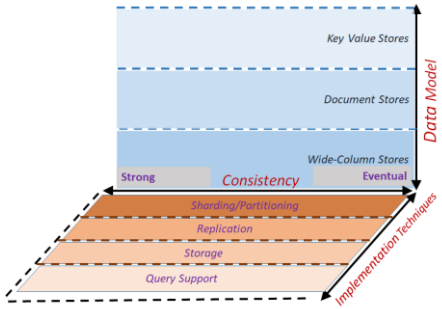


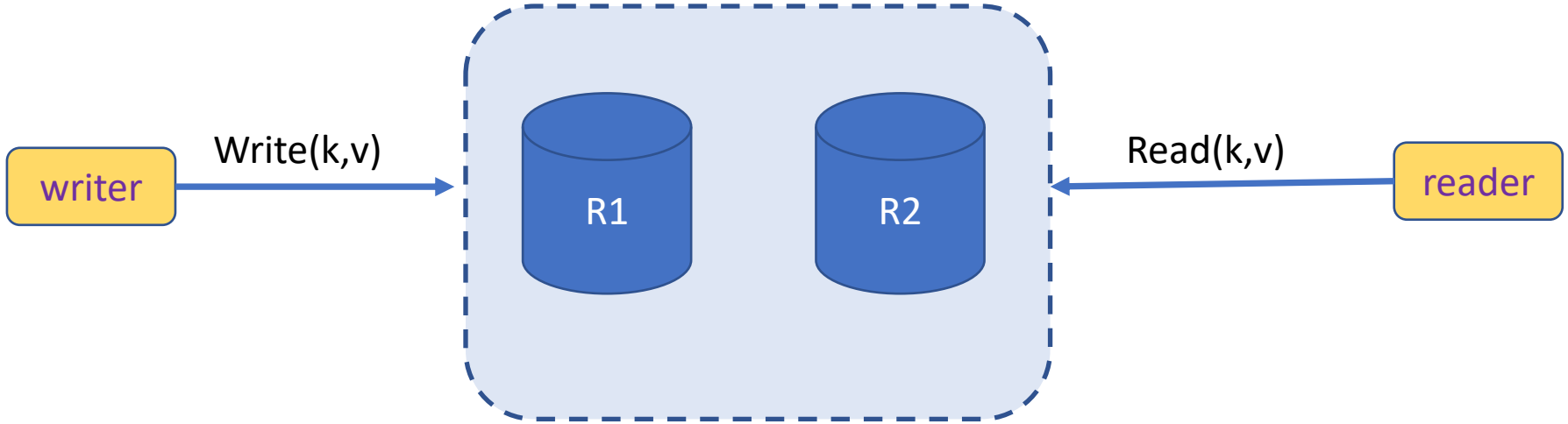


How to keep data in sync?

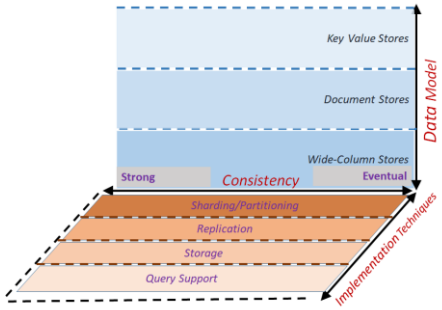
- Partitioning → single row spread over multiple machines
- Redundancy → single datum spread over multiple machines

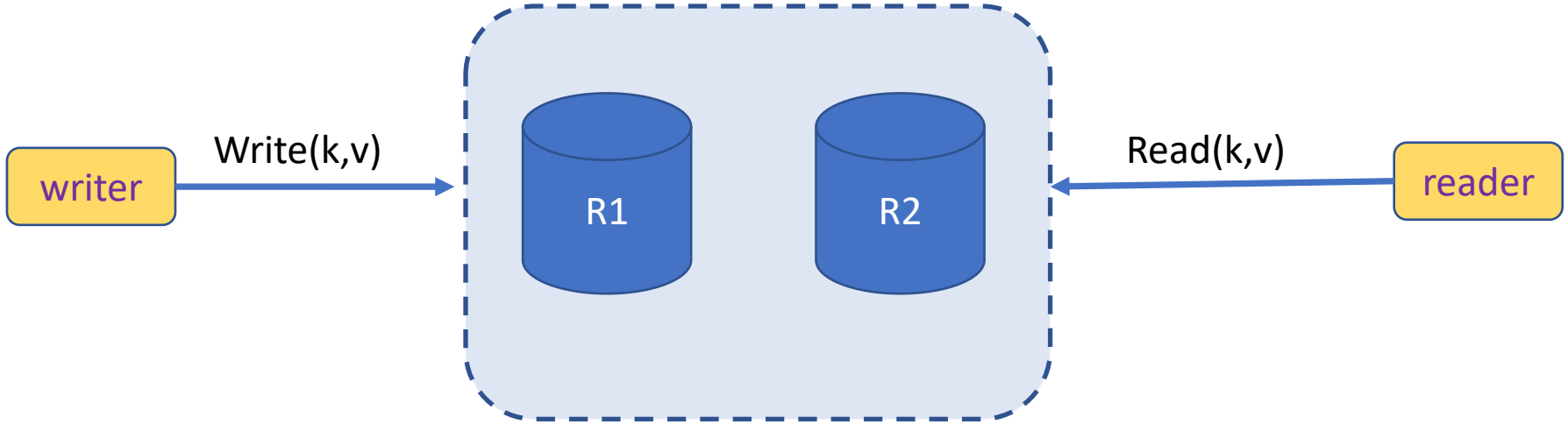
Consistency: the core problem





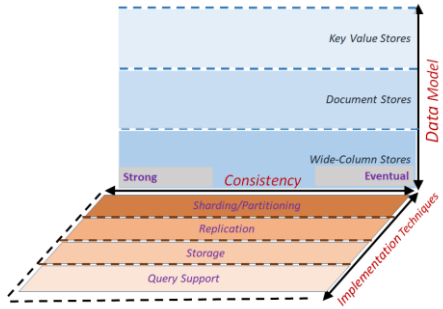
Consistency: the core problem

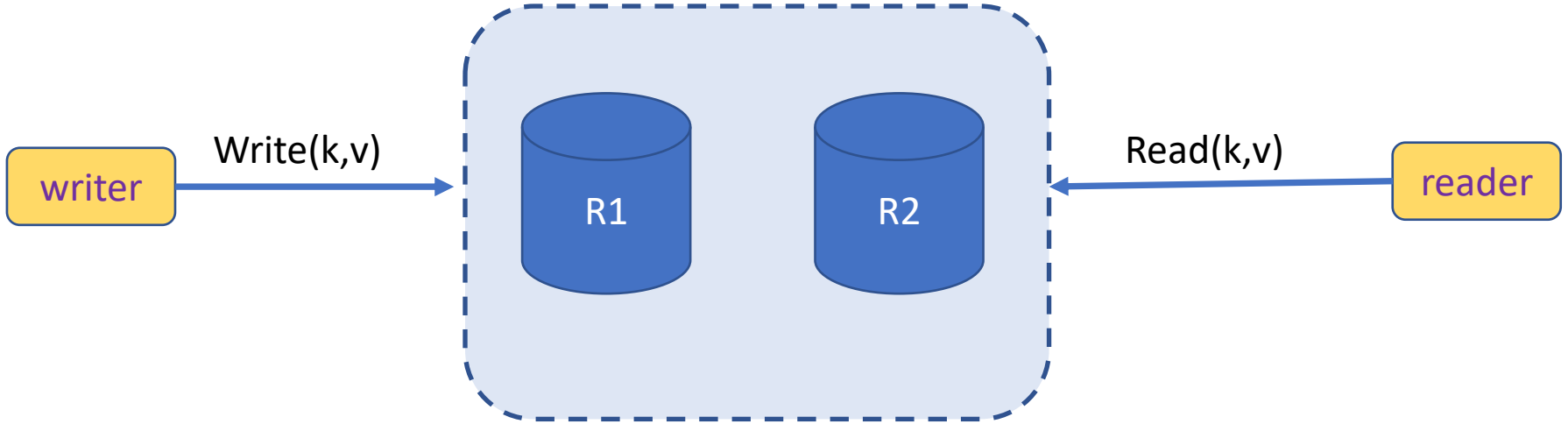




- Clients perform reads and writes

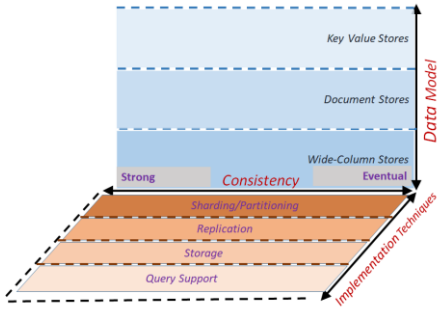
Consistency: the core problem

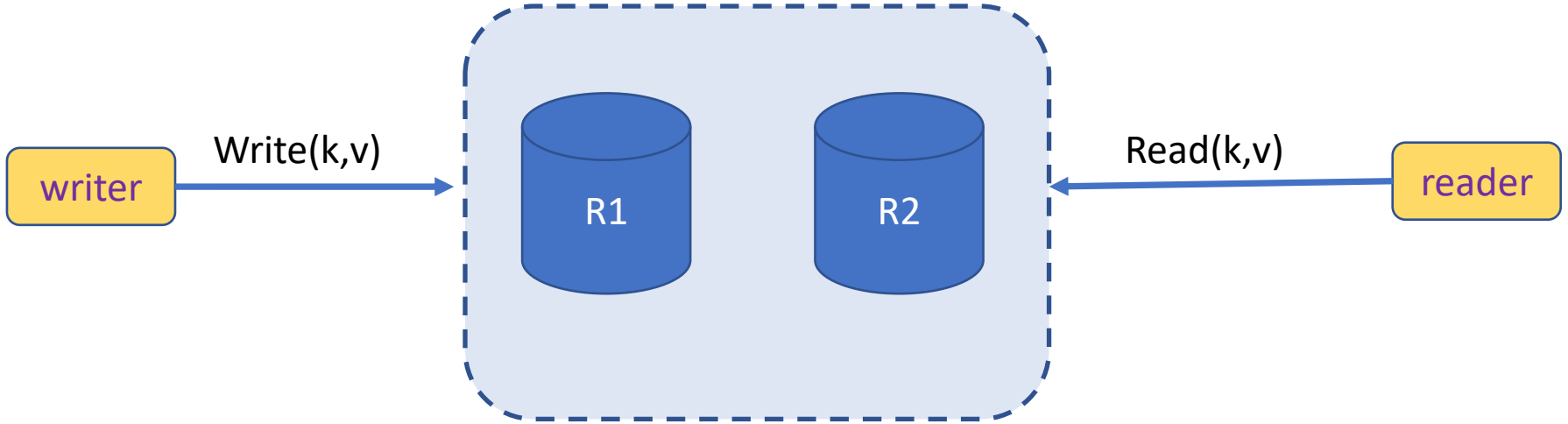




- Clients perform reads and writes
- Data is replicated among a set of servers

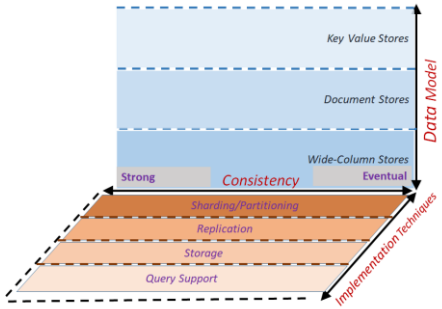
Consistency: the core problem

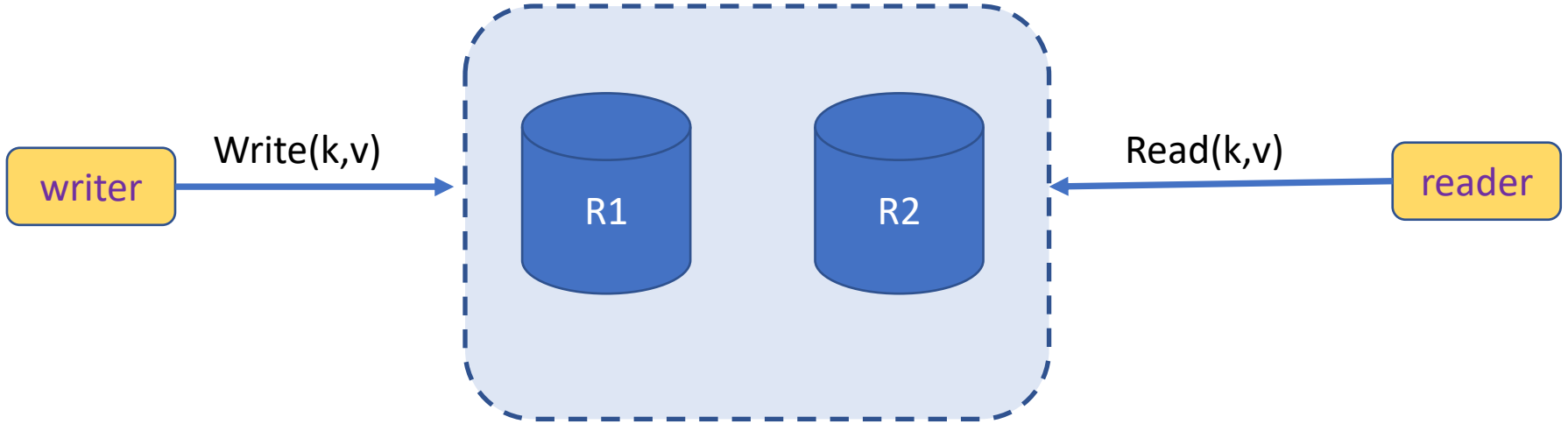




- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers

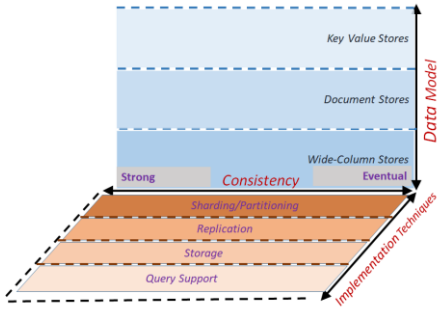
Consistency: the core problem

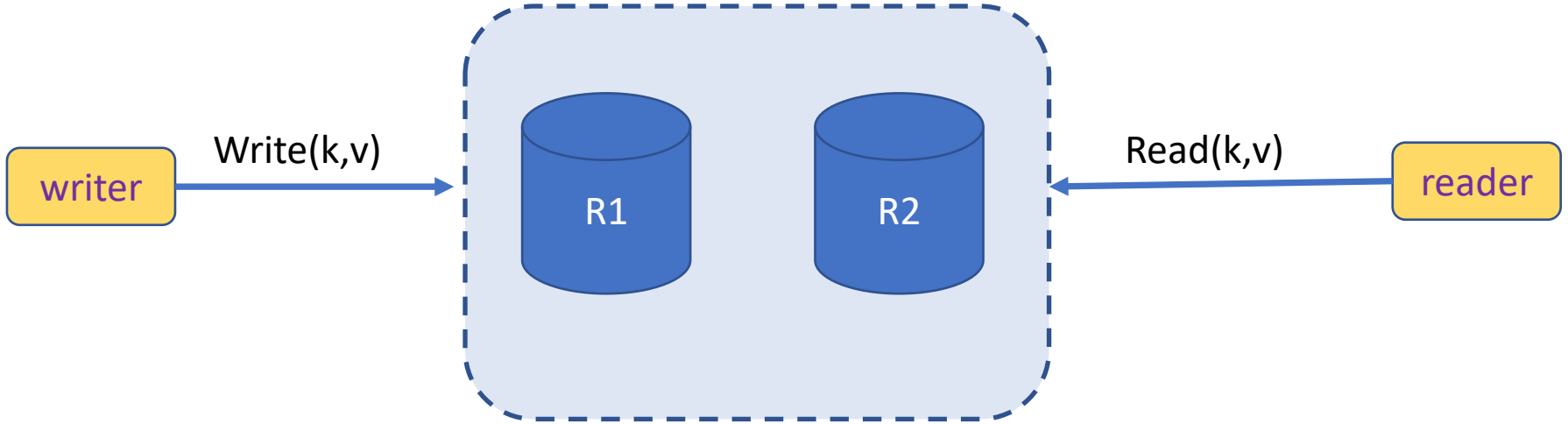




- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes

Consistency: the core problem

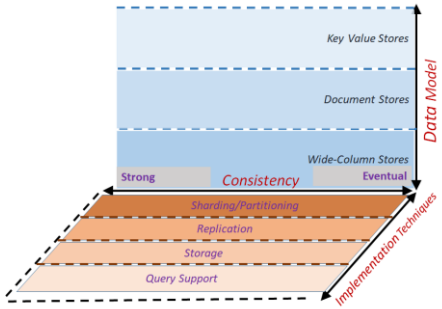


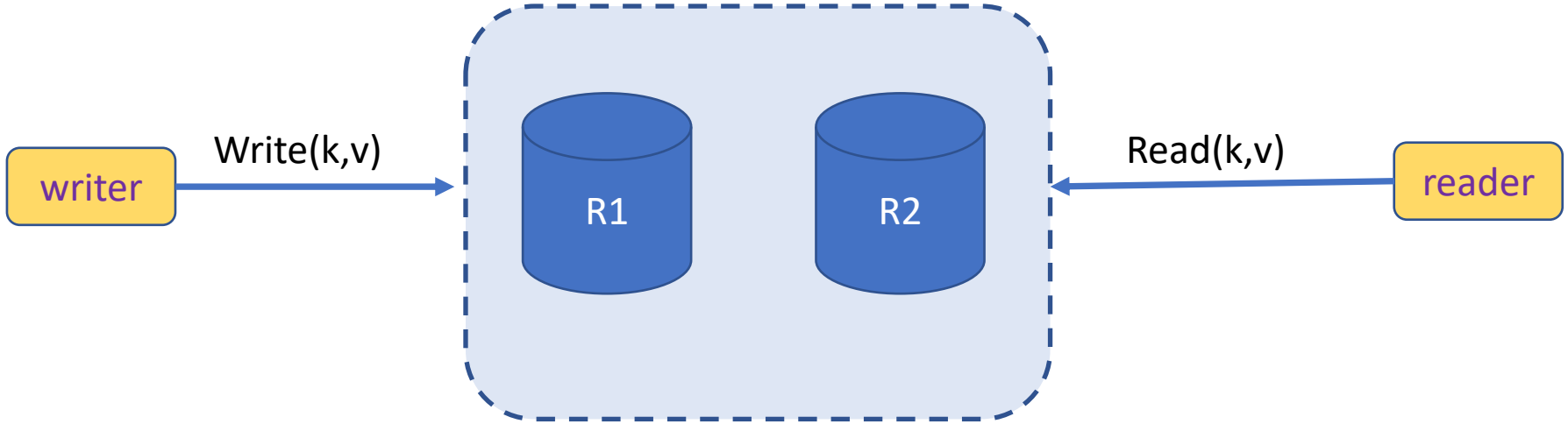


- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes

- How should we *implement* write?

Consistency: the core problem

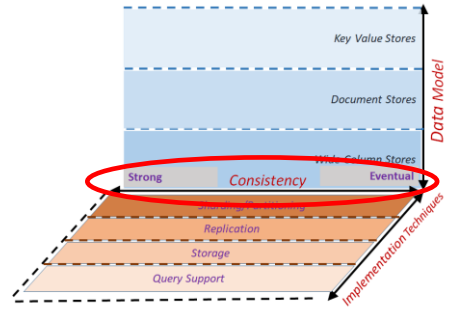




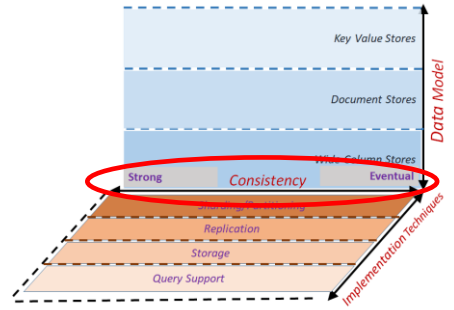
- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes

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

Consistency: CAP Theorem

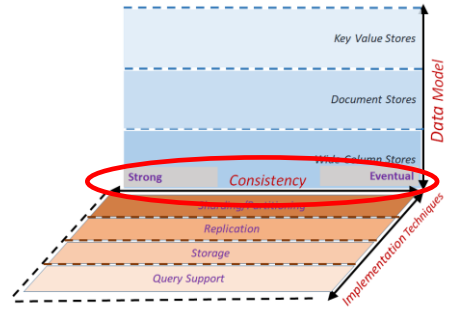


Consistency: CAP Theorem



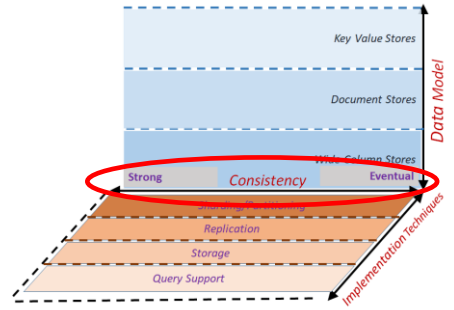
- A distributed system can satisfy at most 2/3 guarantees of:

Consistency: CAP Theorem



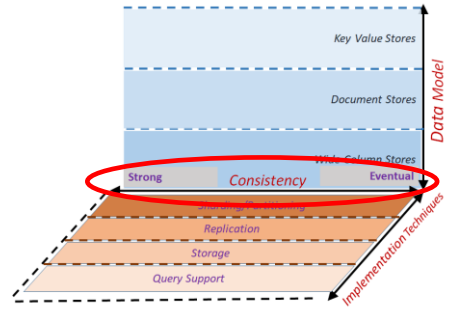
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:

Consistency: CAP Theorem



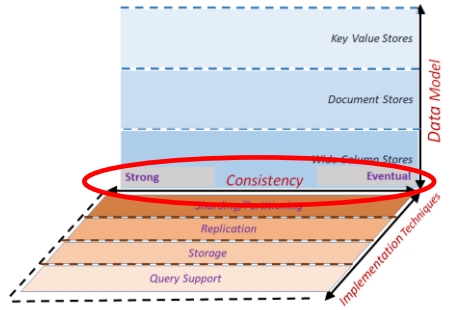
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client

Consistency: CAP Theorem



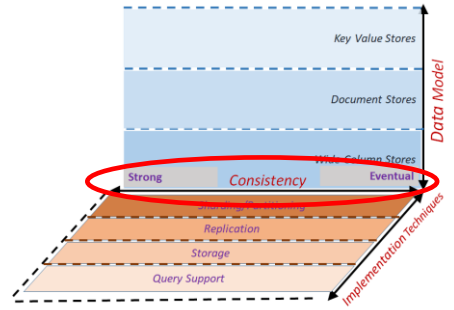
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:

Consistency: CAP Theorem



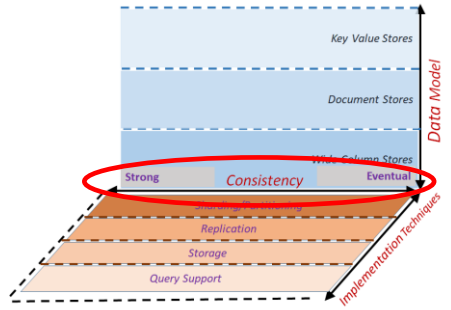
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly

Consistency: CAP Theorem



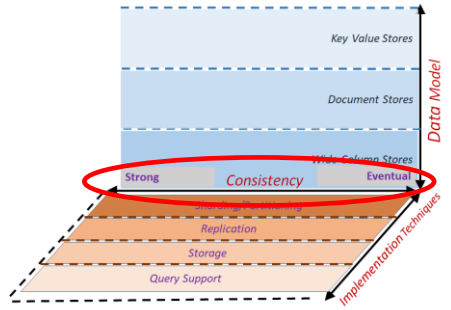
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**:

Consistency: CAP Theorem



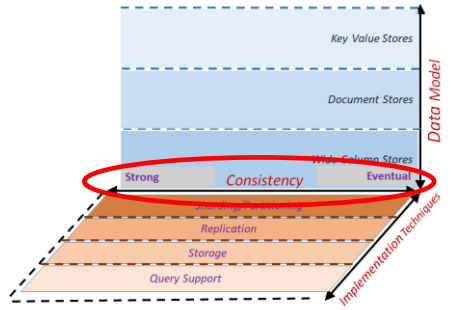
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**: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

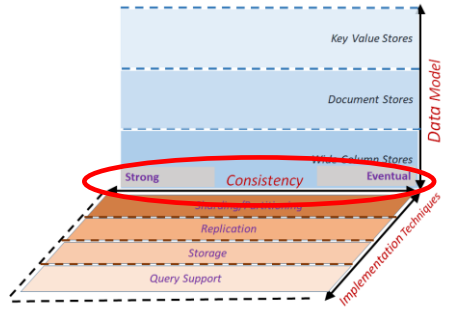
Partitions

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

Consistency

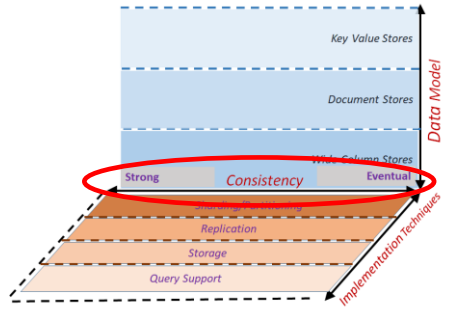
- all nodes see same data at any time, or reads return latest written value by any client.
- ***This basically means correctness!***

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

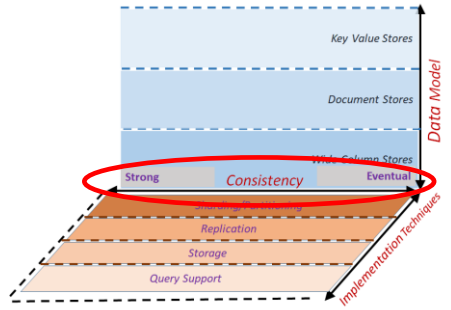
Consistency: CAP Theorem



- 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

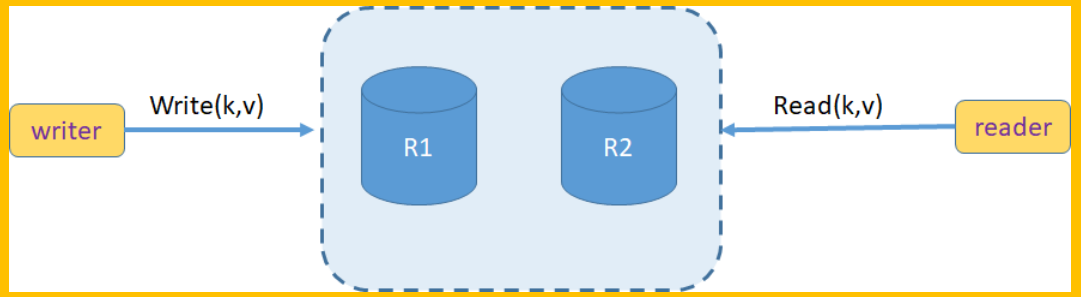
Why is this "theorem" true?

Consistency: CAP Theorem

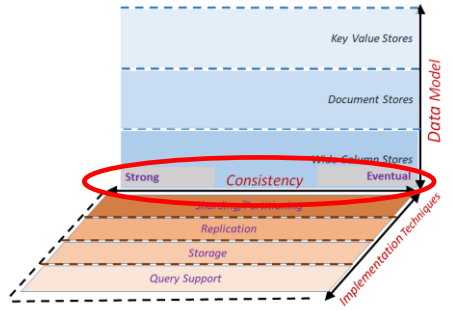


- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**: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

Why is this "theorem" true?

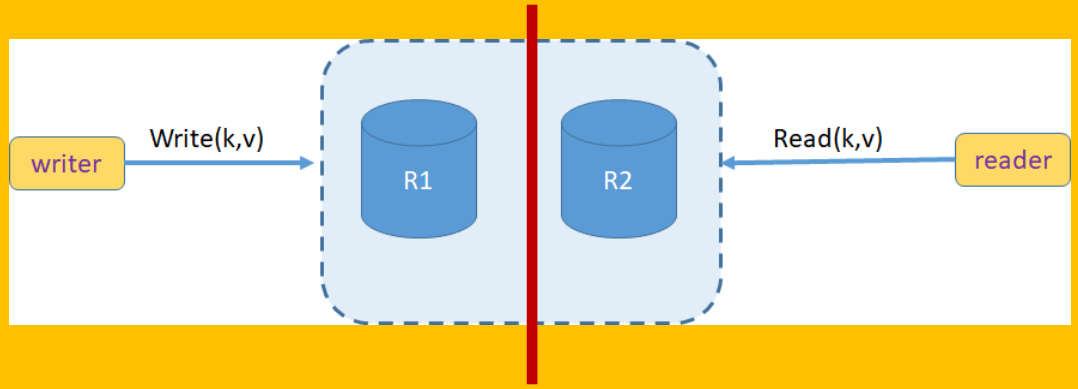


Consistency: CAP Theorem

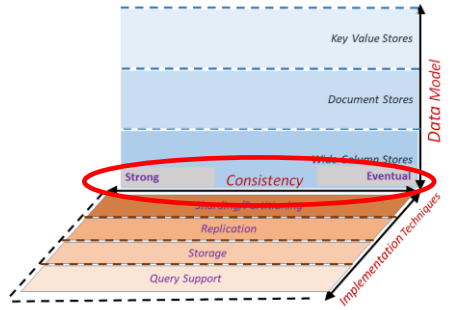


- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**: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

Why is this "theorem" true?

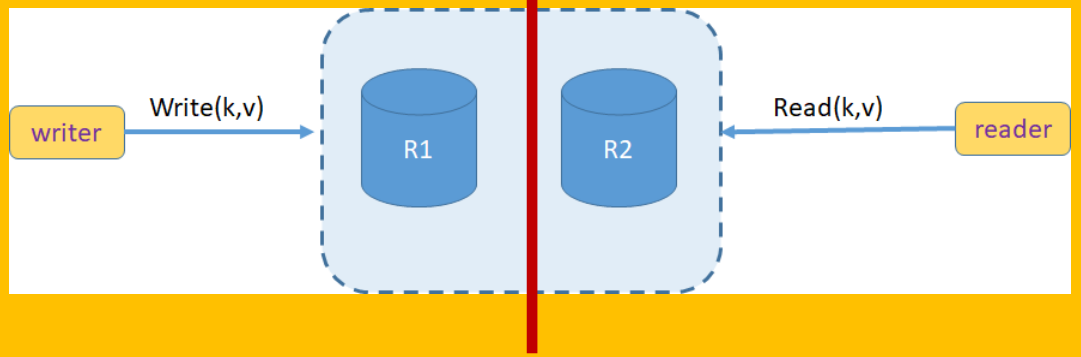


Consistency: CAP Theorem



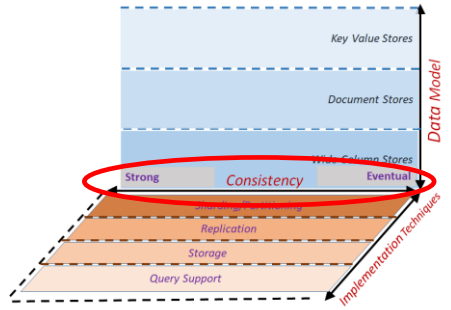
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**: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

Why is this "theorem" true?



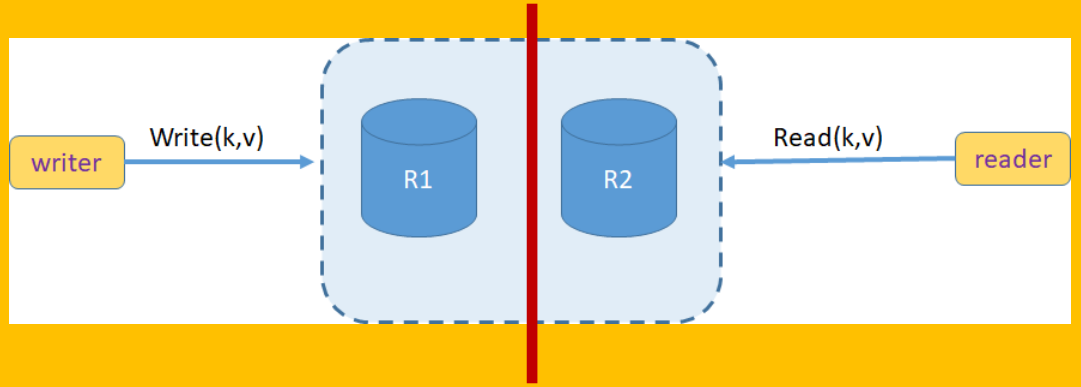
if(partition) { keep going } → !consistent && available

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

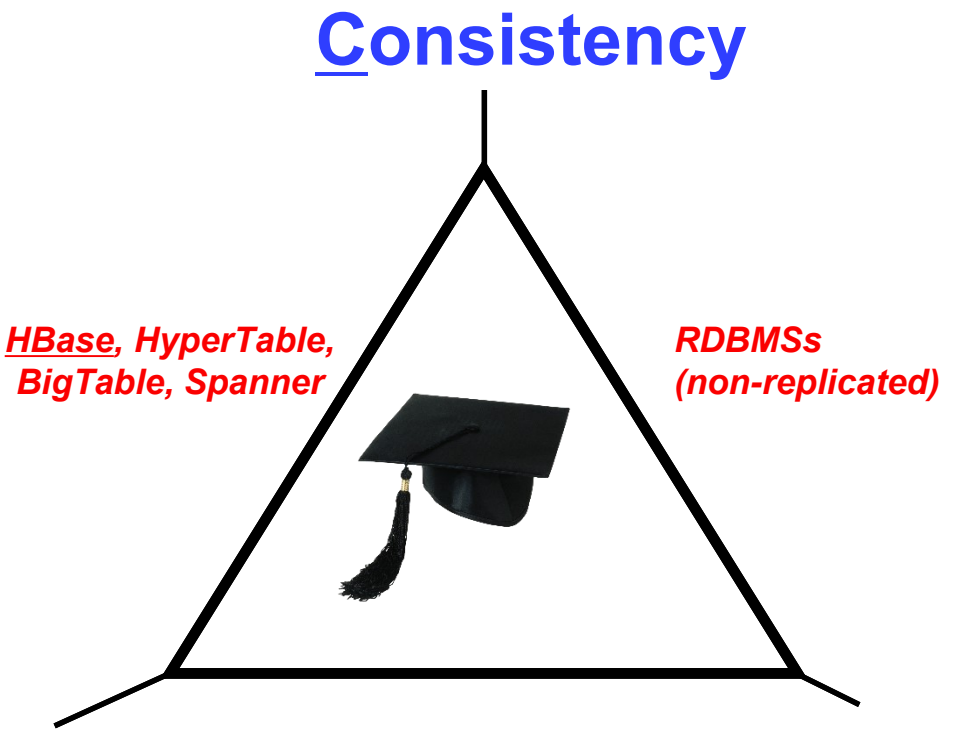
Why is this "theorem" true?



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

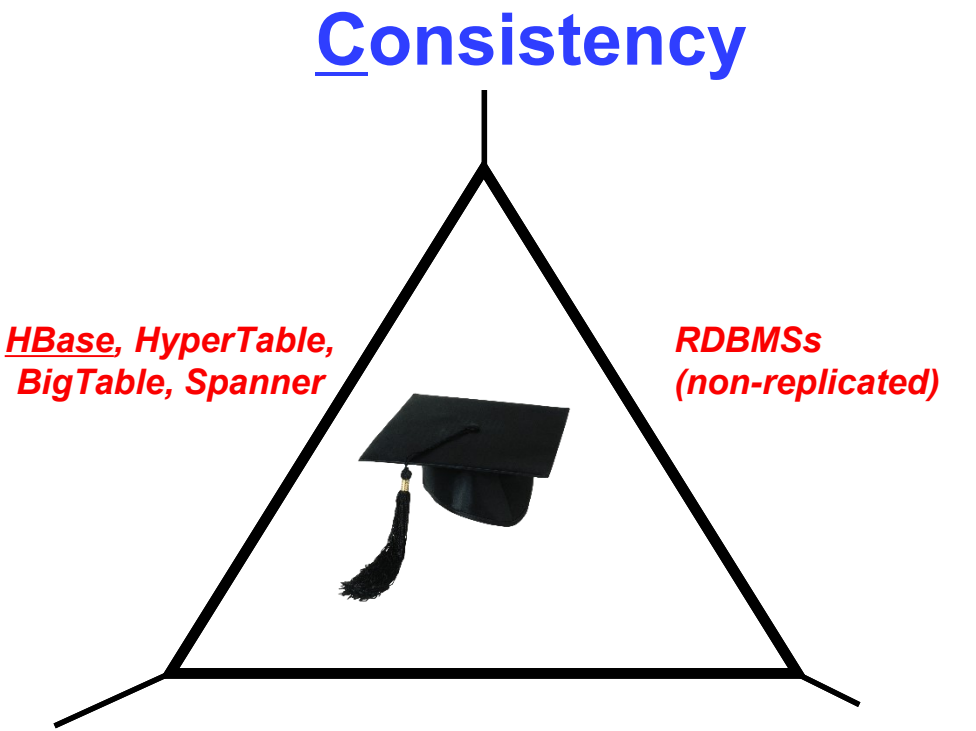
CAP Implications

- 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



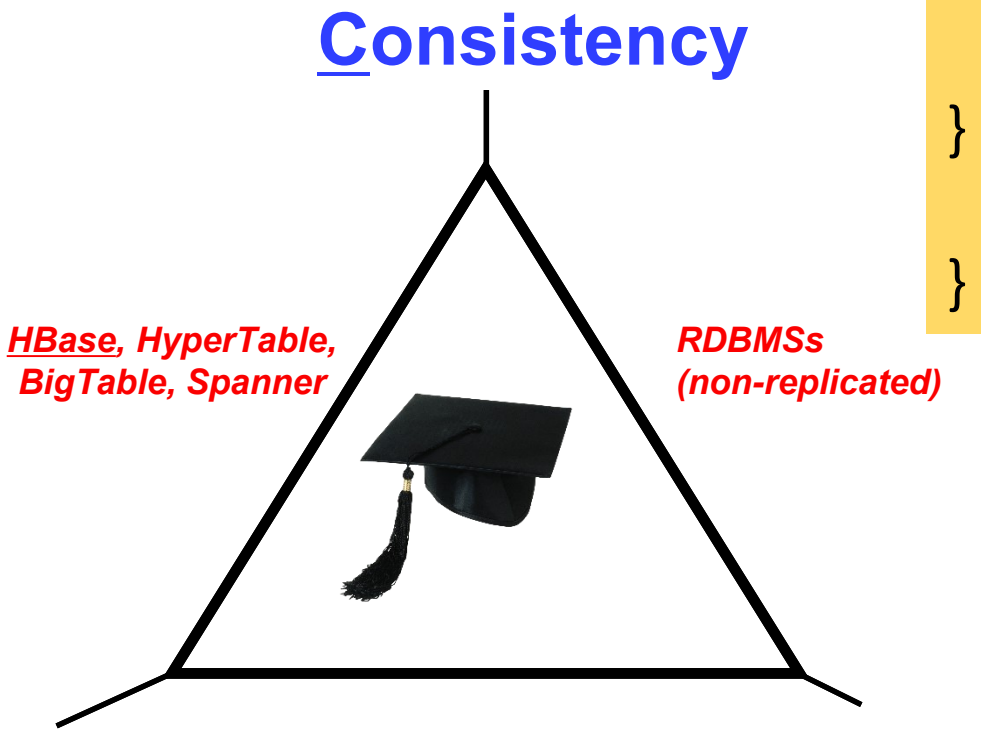
CAP Implications

- 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



CAP Implications

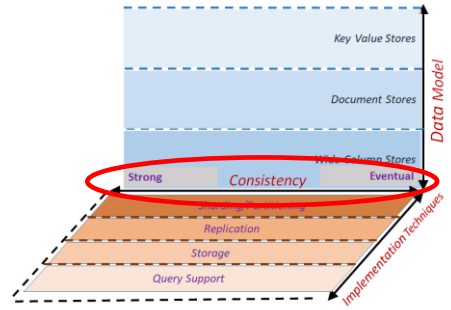
- A distributed storage system can achieve at most two of C, A, and P.
- When partition-tolerance is important, you have to choose between consistency and availability

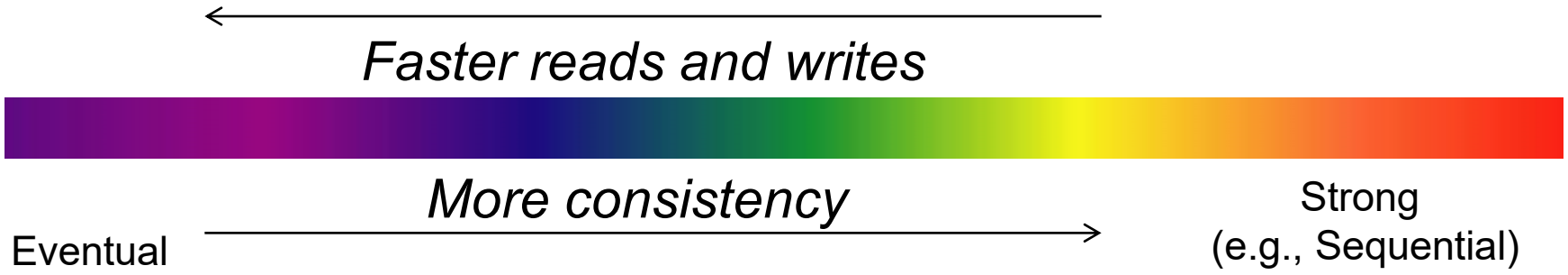


PACELC:

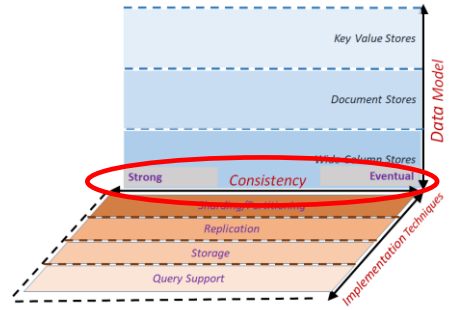
```
if(partition) {
   choose A or C
} else {
   choose latency or consistency
}
```

Consistency Spectrum

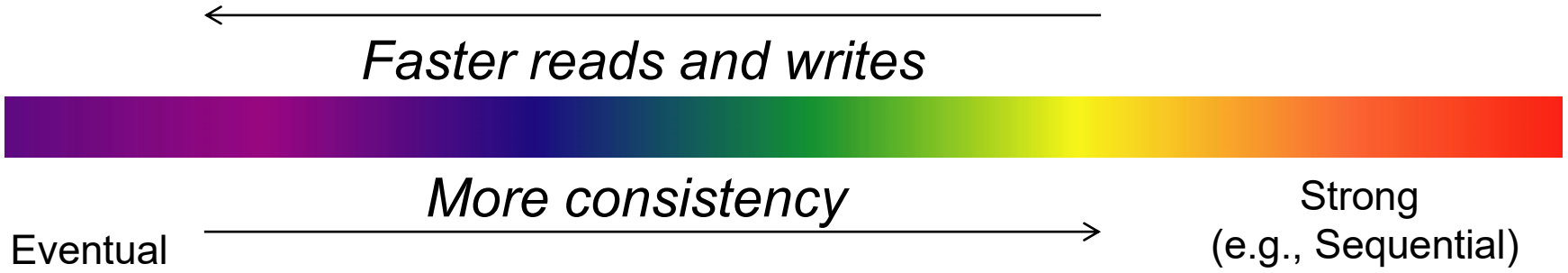




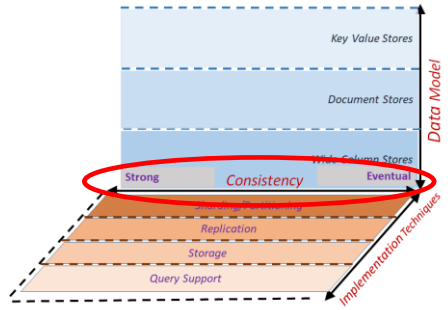
Spectrum Ends: Eventual Consistency



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

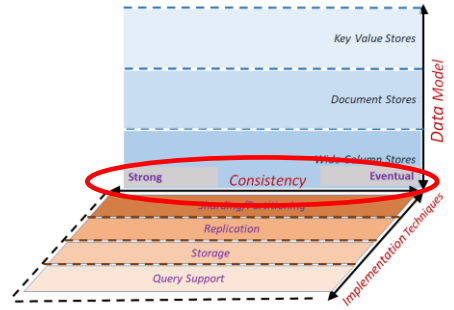


Spectrum Ends: Strong Consistency



- **Strict:**
  - 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

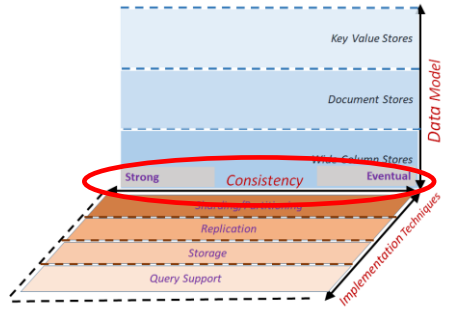
Eventual

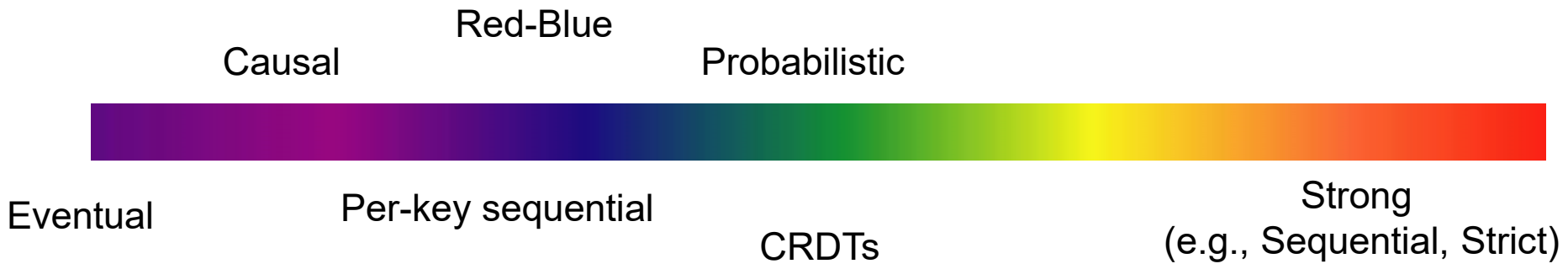
Per-key sequential

CRDTs

Strong
(e.g., Sequential, Strict)

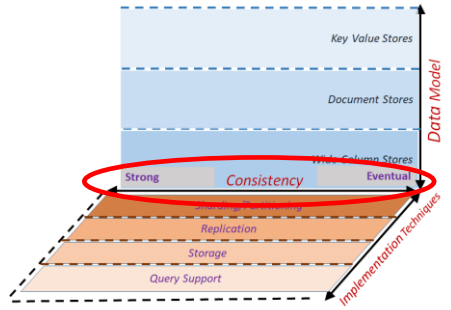
Many *Many* Consistency Models

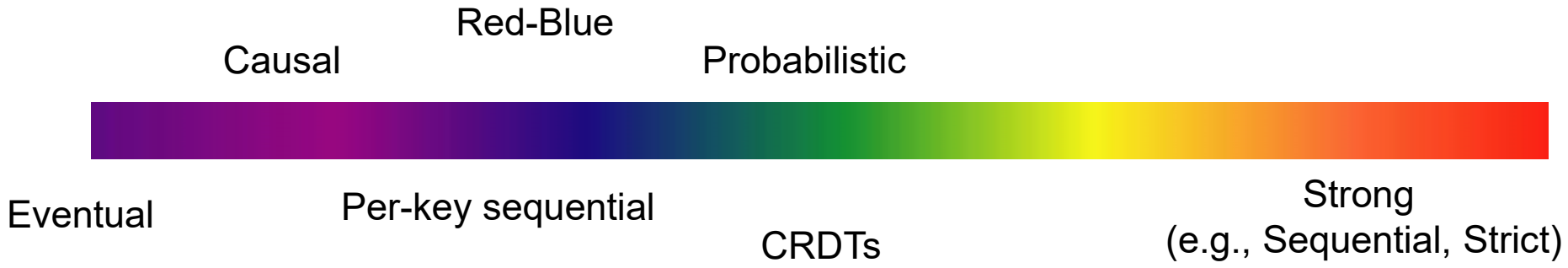




- Amazon S3 – **eventual** consistency
- Amazon Simple DB – **eventual** or strong
- Google App Engine – **strong** or eventual
- Yahoo! PNUTS – **eventual** or strong
- Windows Azure Storage – **strong** (or eventual)
- Cassandra – **eventual** or strong (if R+W > N)
- …

Many *Many* Consistency Models





- 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?

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