Projects
End-of-semester Review

cs378h

Outline/Administrivia

- Questions?
- Comments on Exam
- Project presentations x 2
- Review
  - Can someone please act as scribe?
  - Requested review content:
    - NoSQL/databases
    - ACID vs. BASE
    - Linearizability vs. Serializability
    - Spark
      - Pros/cons wrt page rank / indexing
      - Pros/cons wrt multi-core parallelism

Project Presentations

- Emily & Abby
- Ryan & Patrick
- Any last minute additions?

What is NoSQL?

- Next Generation Compute/Storage engines (databases)
  - **non-relational**
  - **distributed**
  - **open-source**
  - **horizontally scalable**
- One view: “no” → elide SQL/database functionality to achieve scale
- Another view: “NoSQL” is actually misleading.
  - more appropriate term is actually “Not Only SQL”

What is NoSQL?

- Next Generation Compute/Storage engines (databases)
  - **non-relational**
  - **distributed**
  - **open-source**
  - **horizontally scalable**
- One view: “no” → elide SQL/da
- Another view: “NoSQL” is actua
  - more appropriate term is actually “Not Only SQL”

What NoSQL gives up in exchange for scale:

- Relationships between entities are non-existent
- Limited or no ACID transactions
- No standard language for queries (SQL)
- Less structured

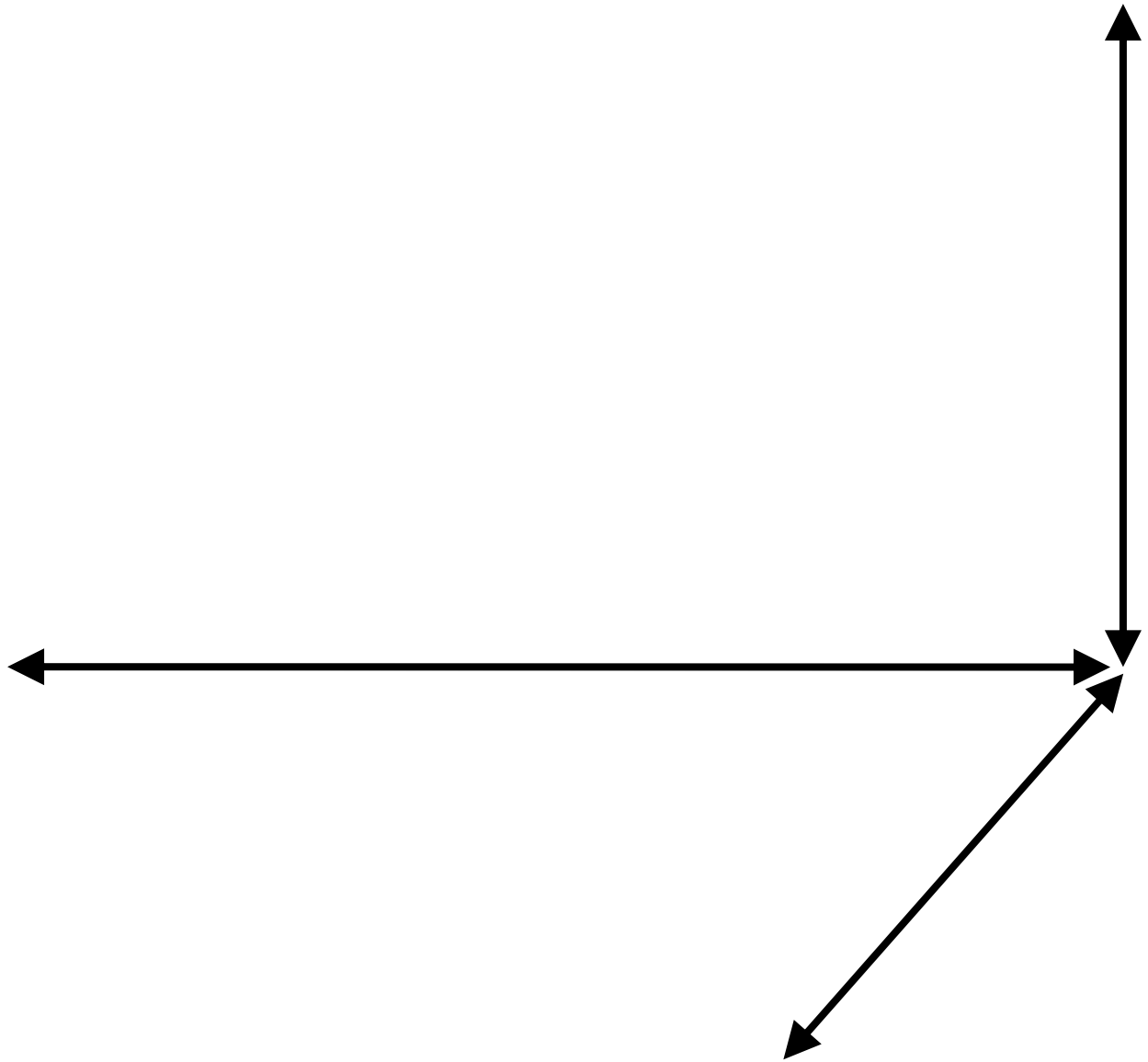
What is NoSQL?

- Next Generation Compute/Storage engines (databases)
  - **non-relational**
  - **distributed**
  - **open-source**
  - **horizontally scalable**
- One view: "no" → elide SQL/da
- Another view: "NoSQL" is actua
  - more appropriate term is actually "No

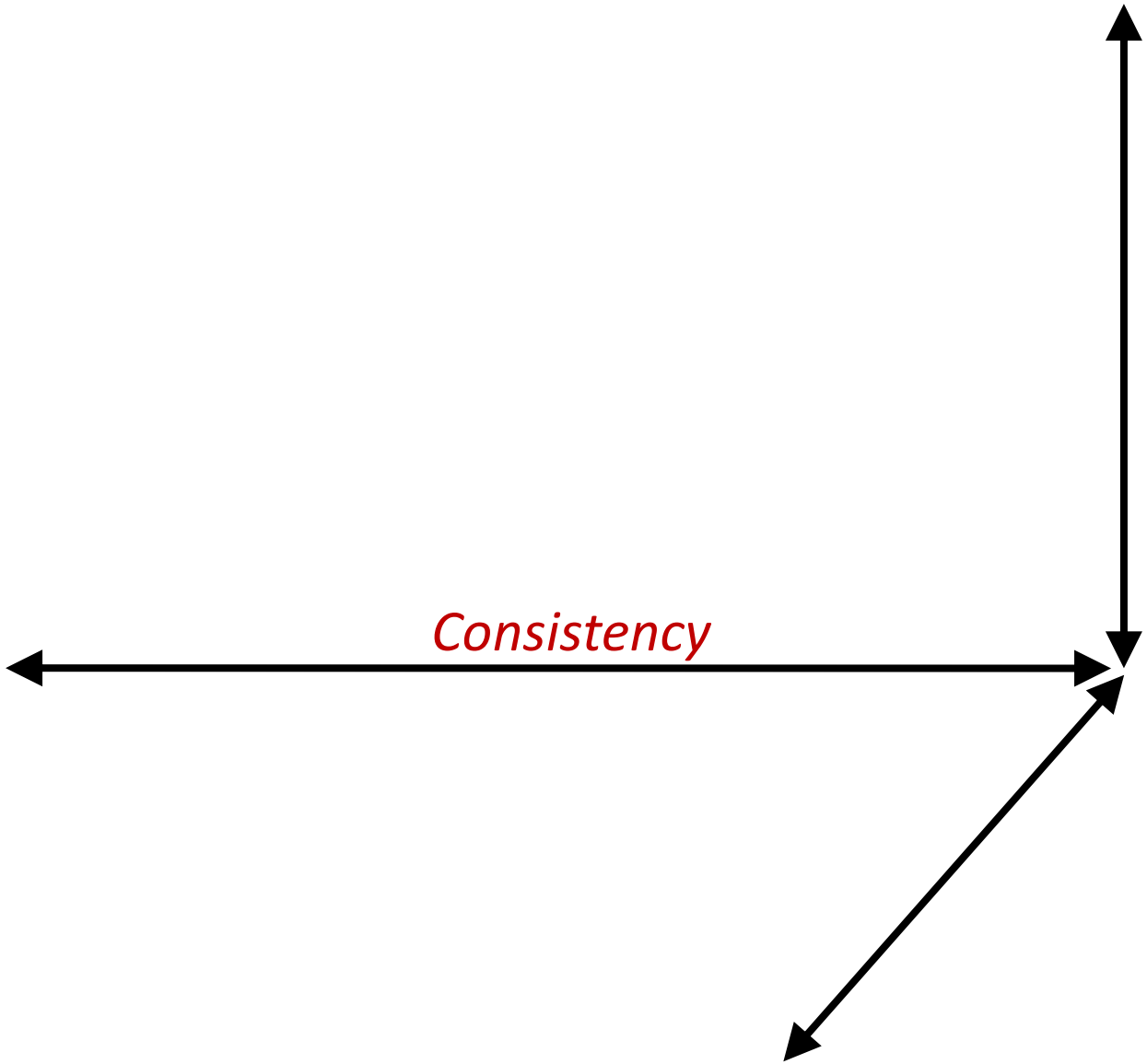
What NoSQL gives up in exchange for scale:

- *Why talk about NoSQL in concurrency class?*
- Principle
  - Most tradeoffs are a ***direct result*** of concurrency
- Practice
  - NoSQL systems are ubiquitous
- Relevant aspects
  - scale/performance tradeoff space
  - Correctness/programmability tradeoff space

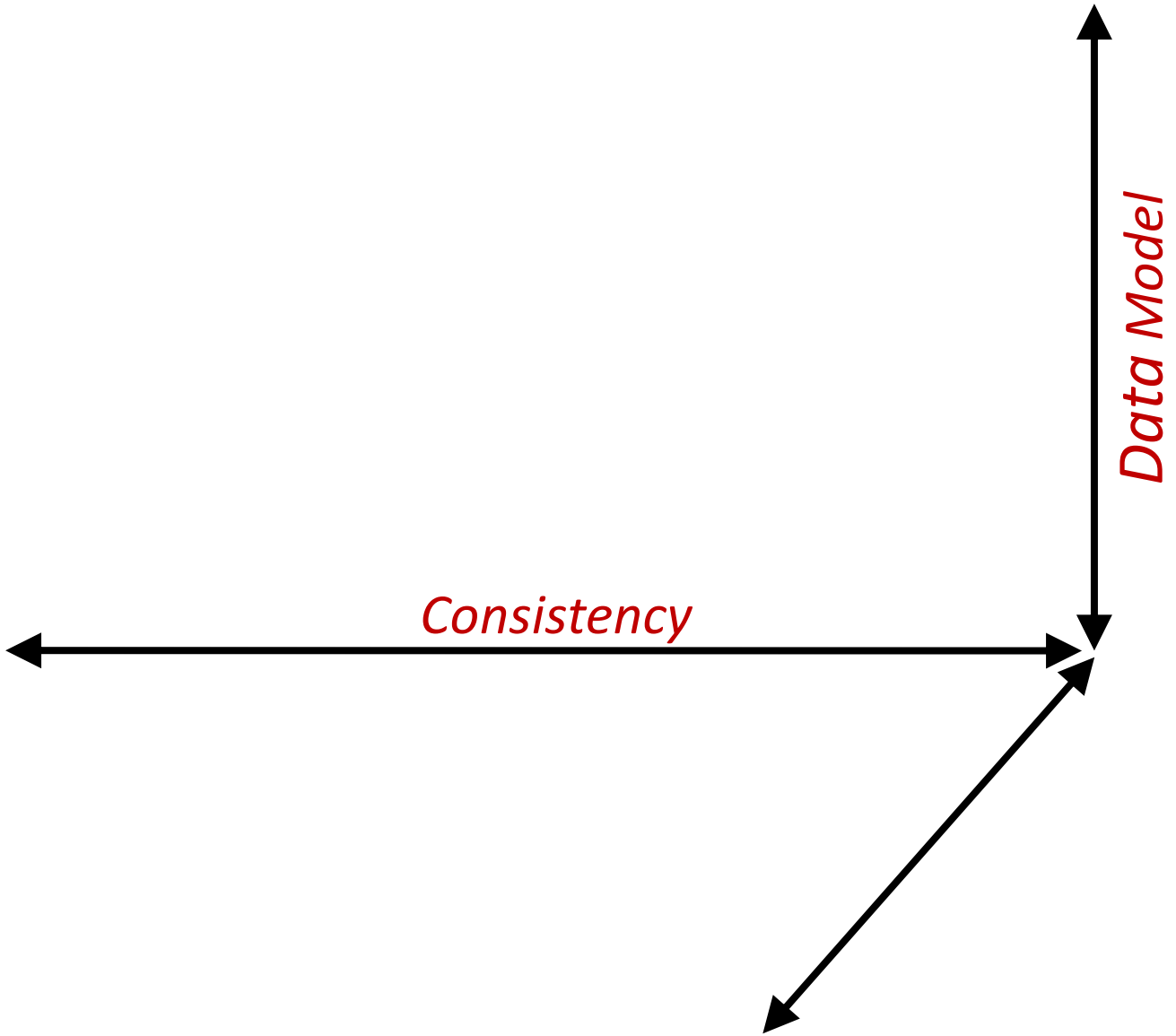
Review: noSQL Taxonomy



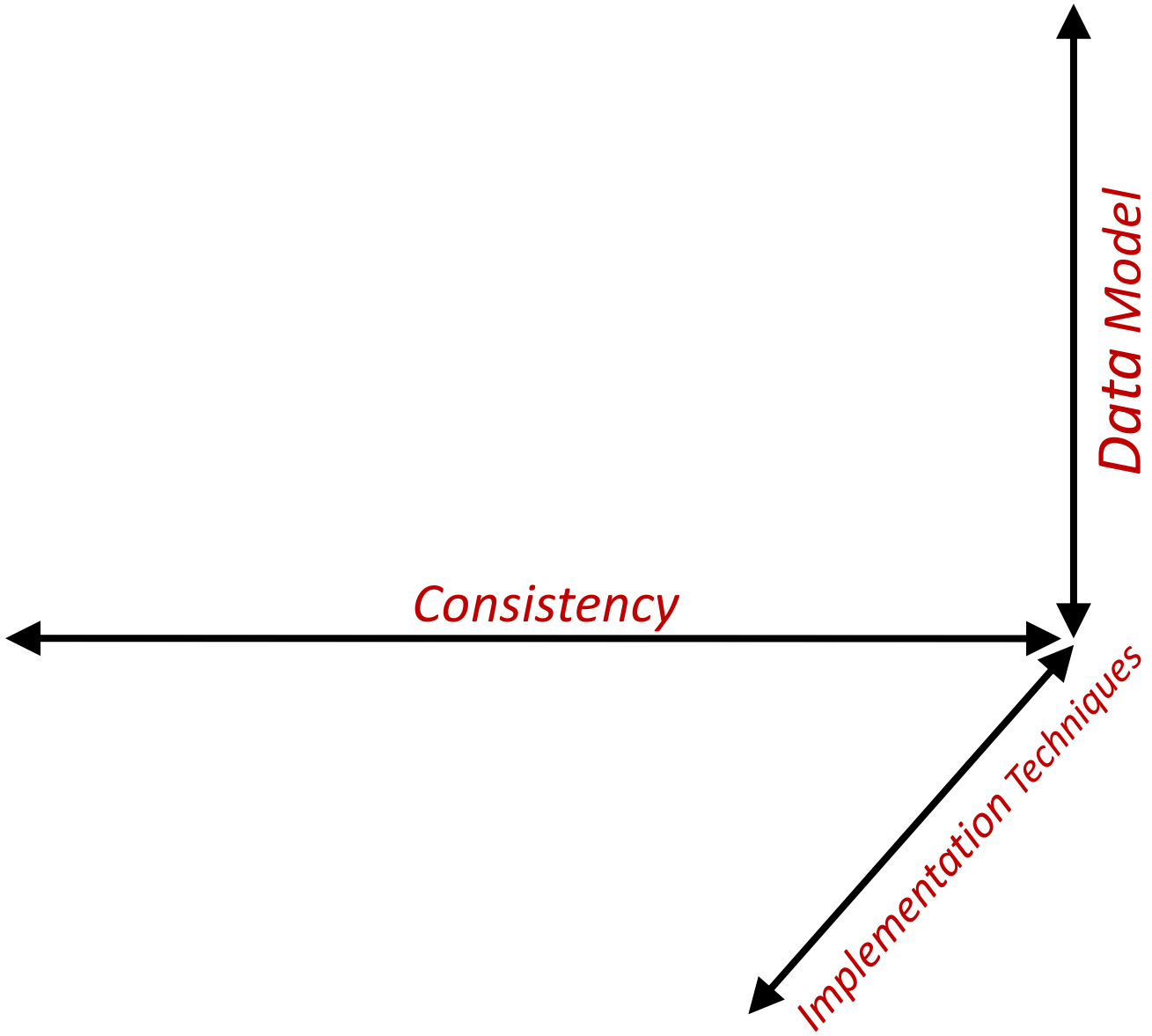
Review: noSQL Taxonomy



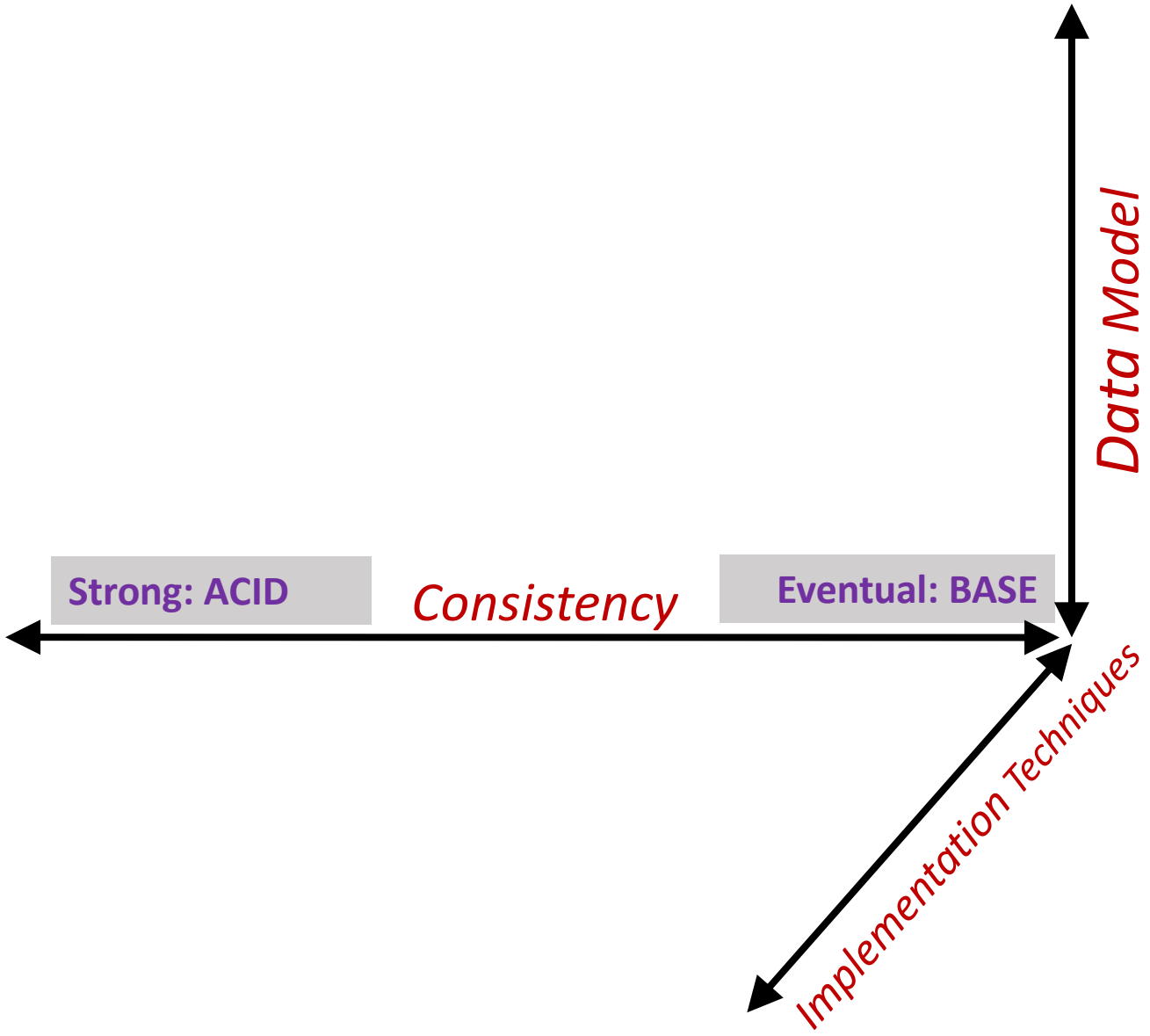
Review: noSQL Taxonomy



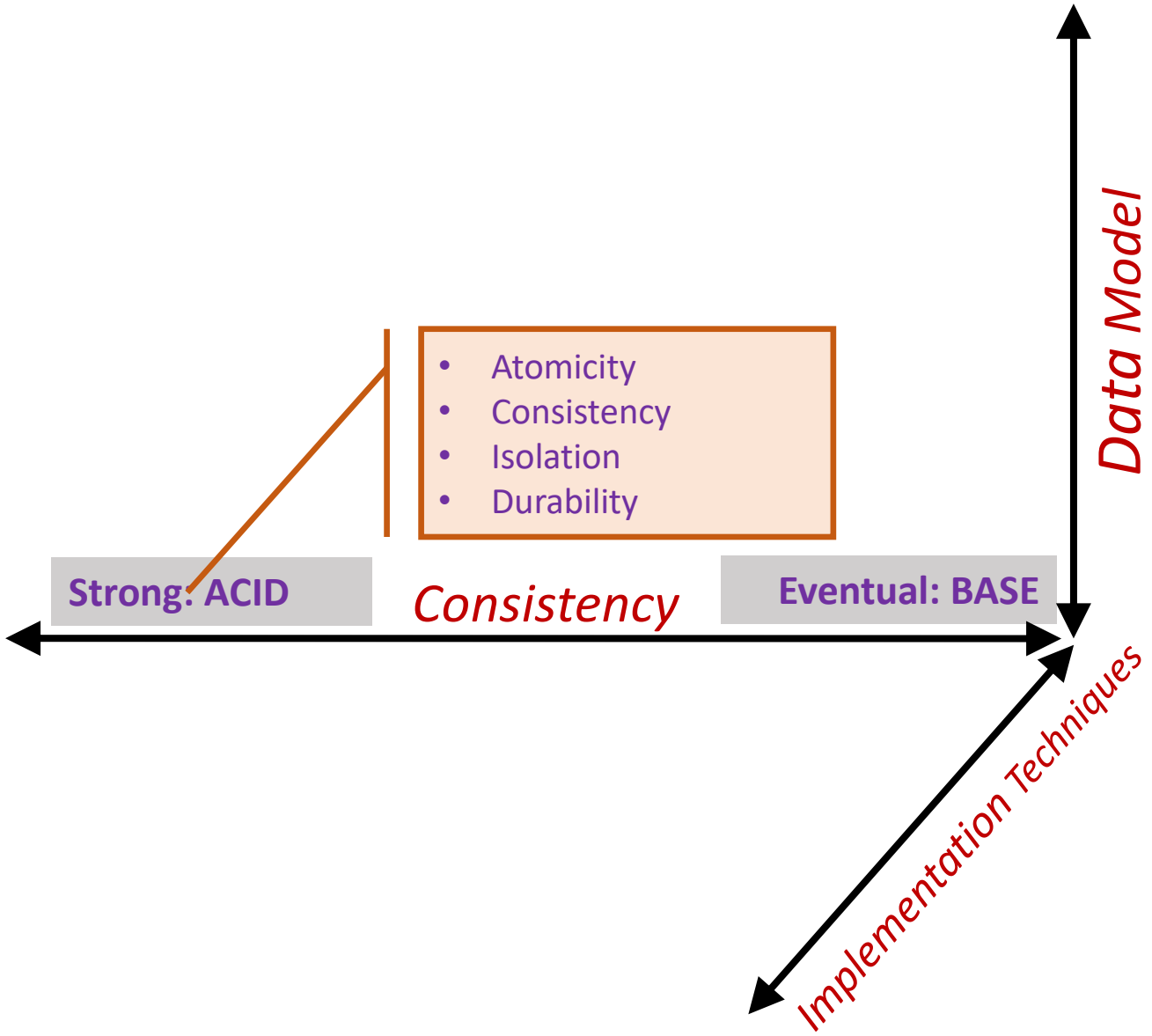
Review: noSQL Taxonomy



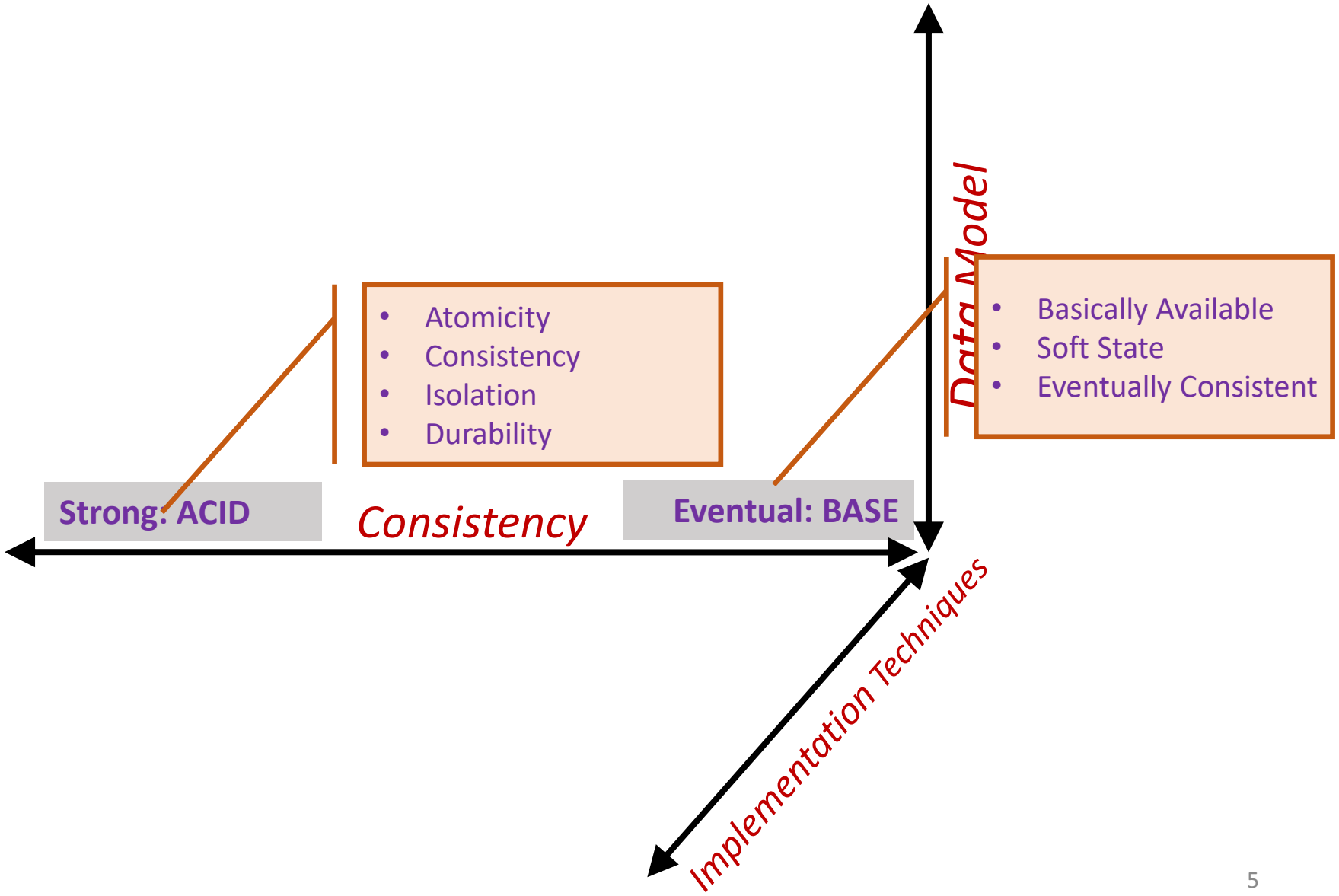
Review: noSQL Taxonomy



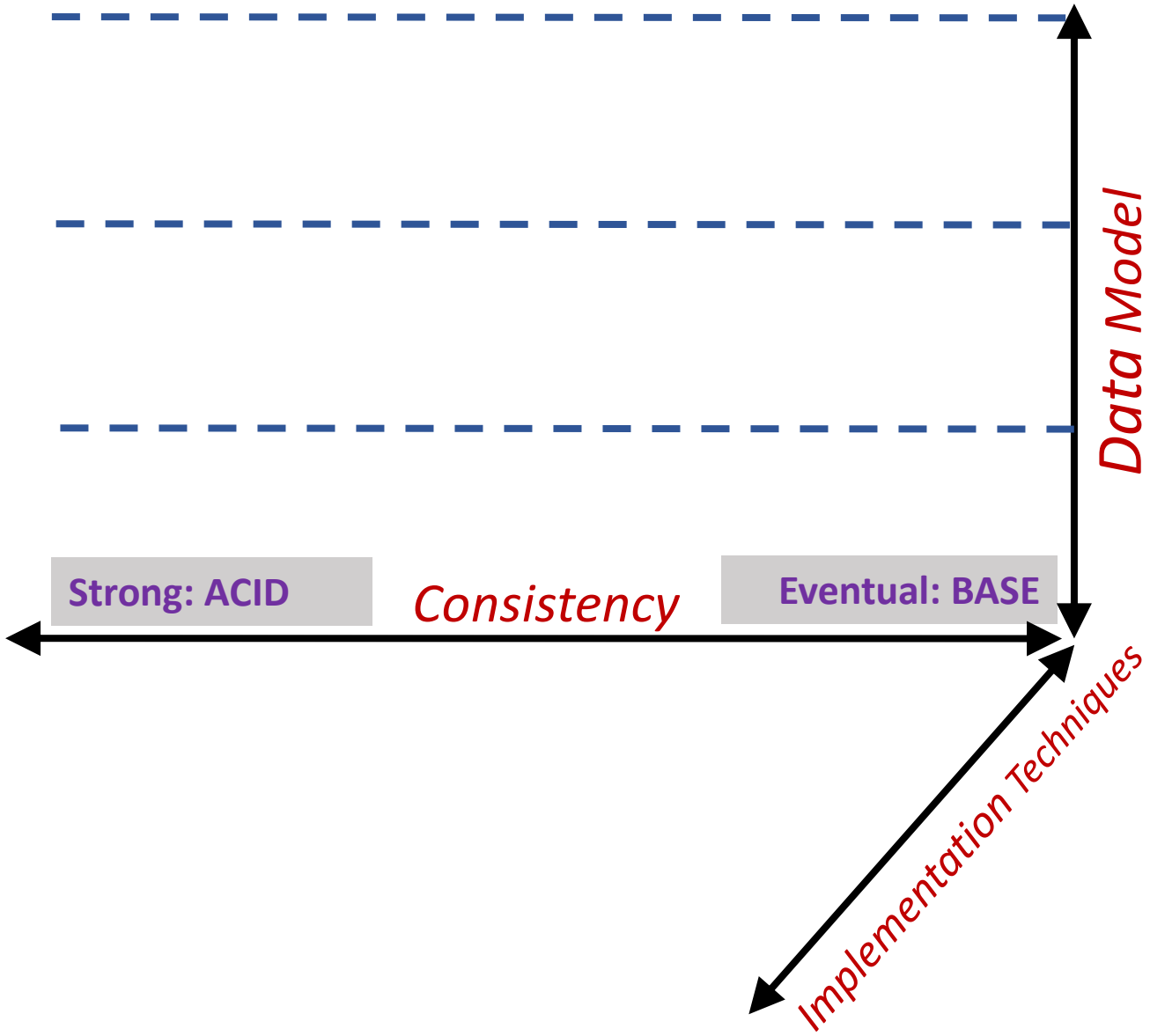
Review: noSQL Taxonomy



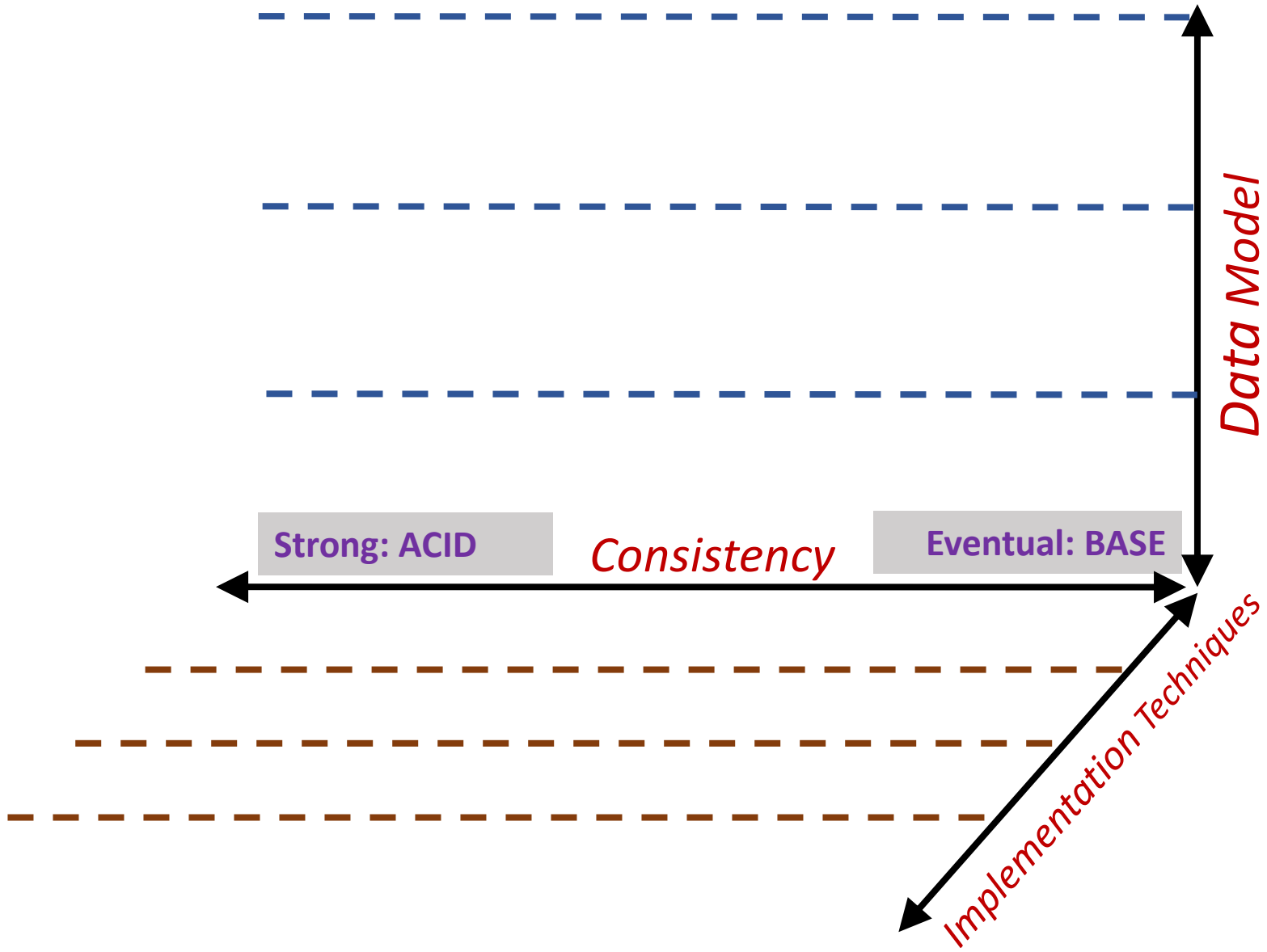
Review: noSQL Taxonomy



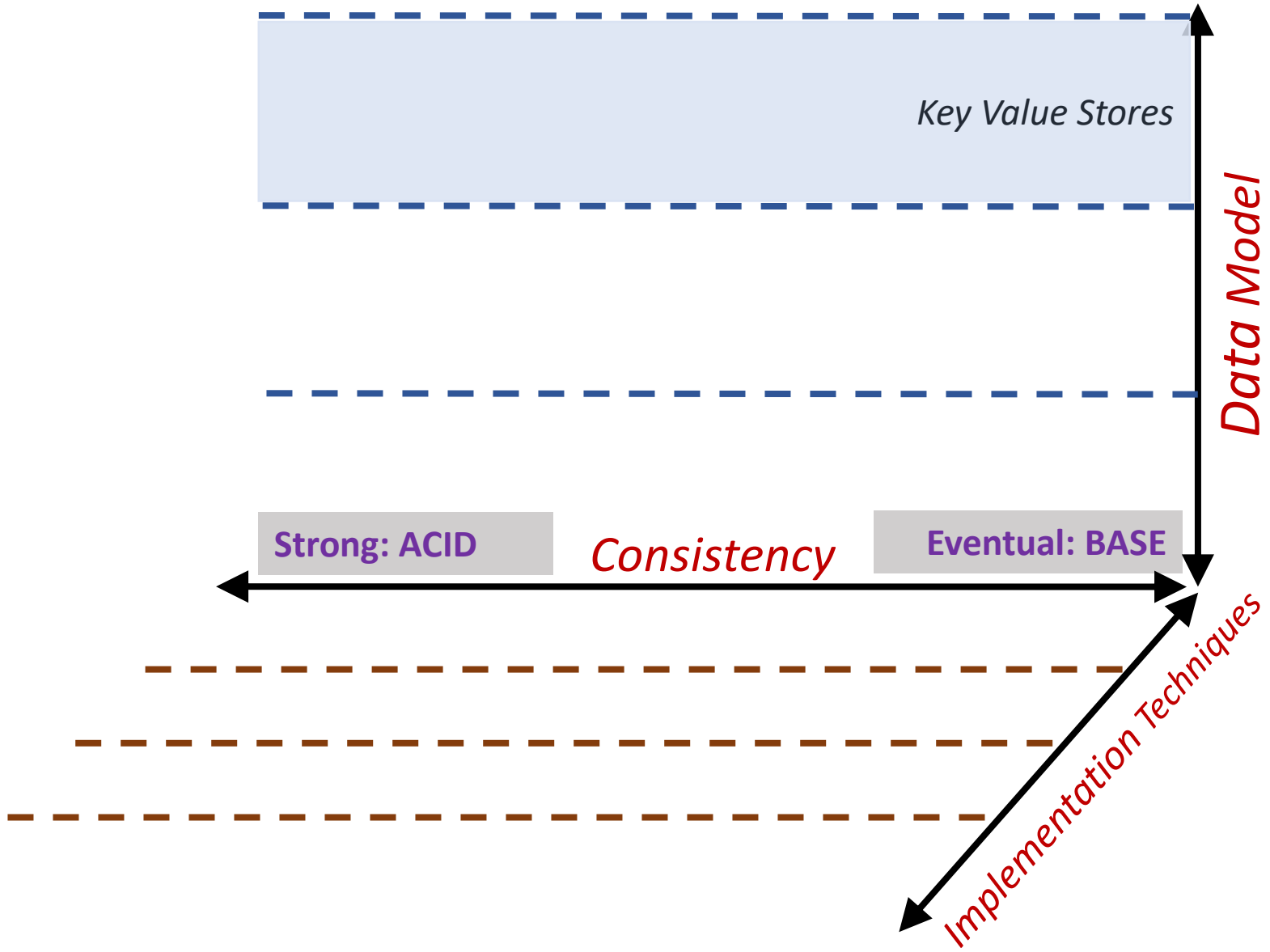
Review: noSQL Taxonomy



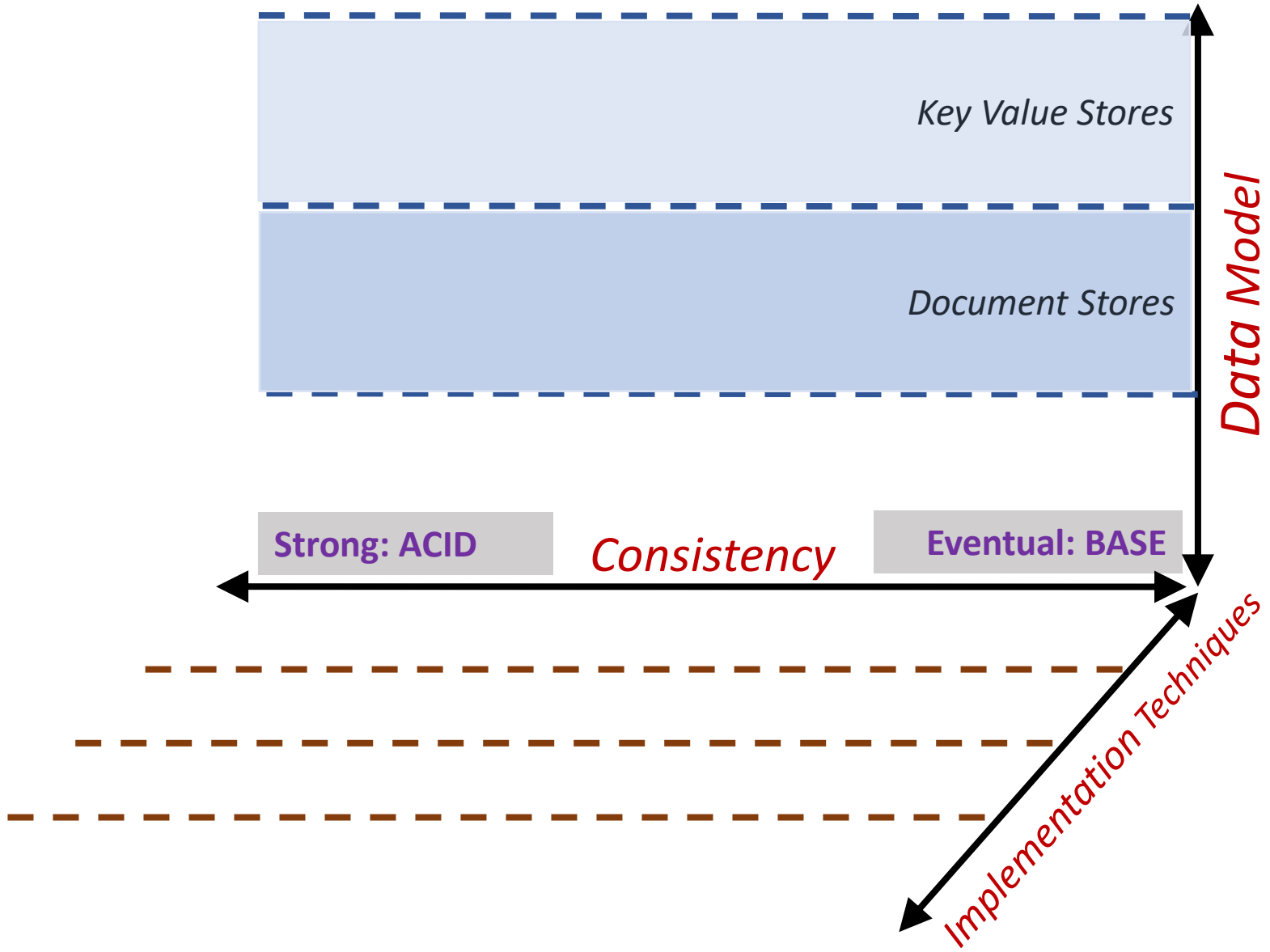
Review: noSQL Taxonomy



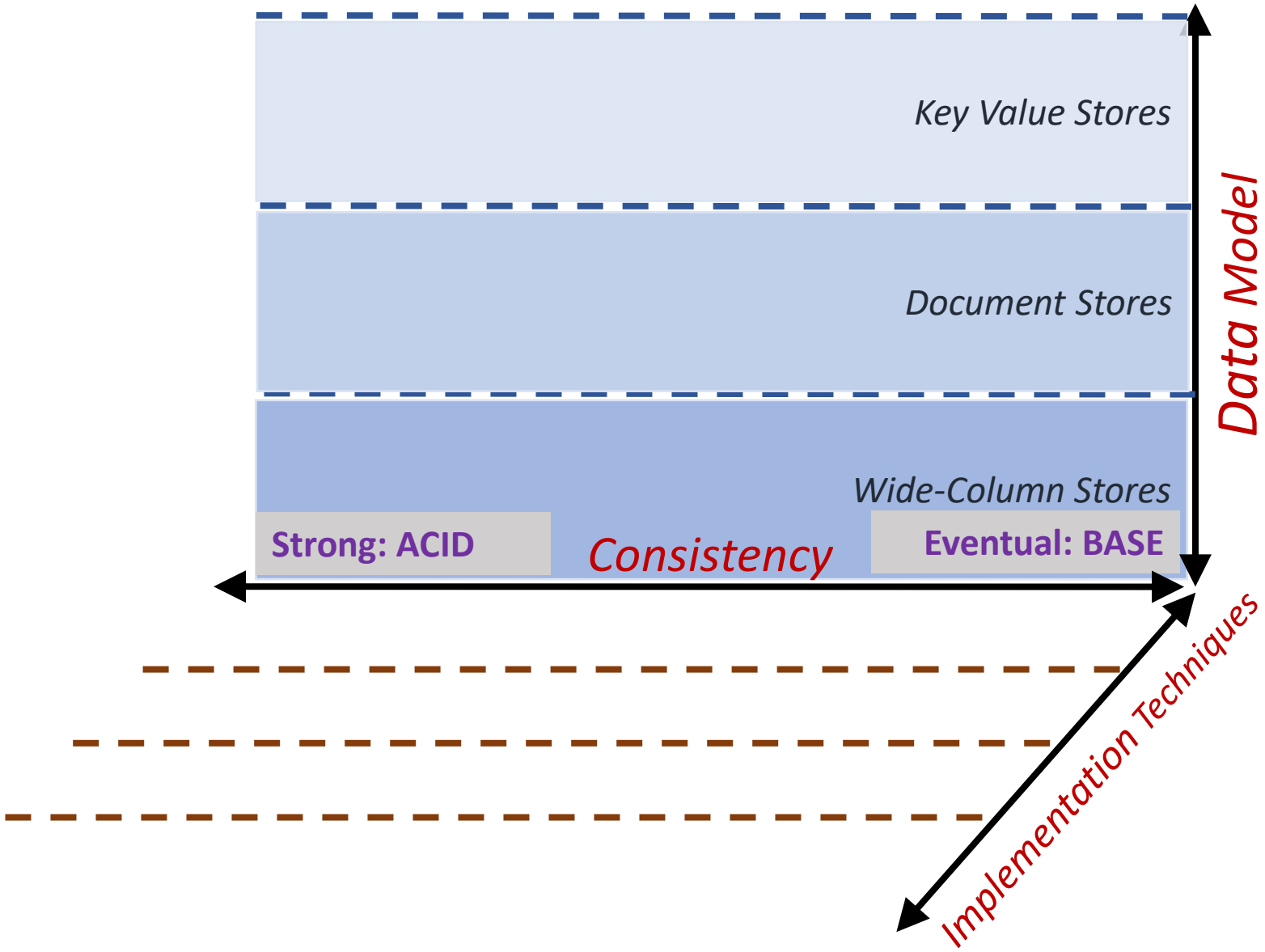
Review: noSQL Taxonomy



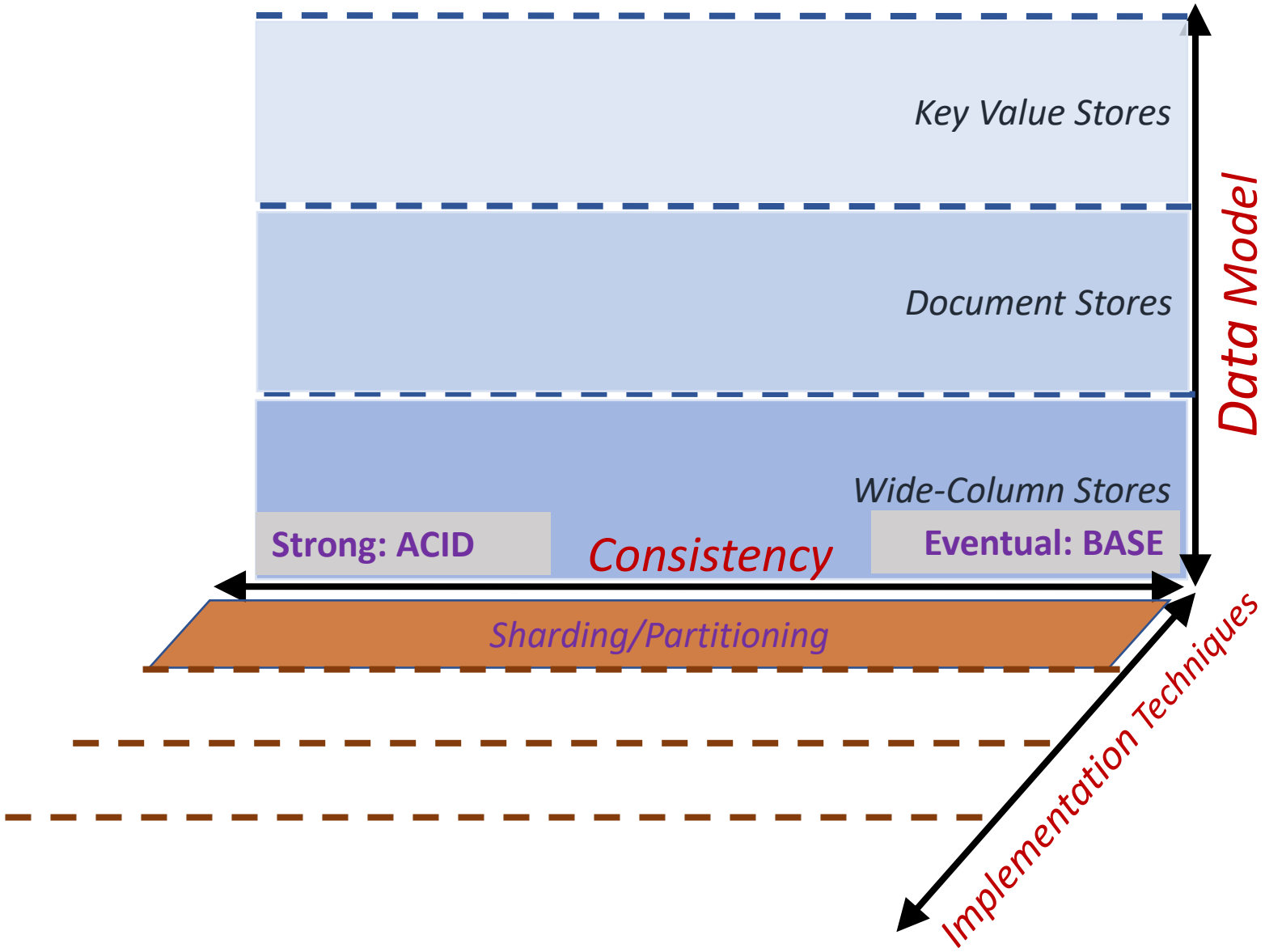
Review: noSQL Taxonomy



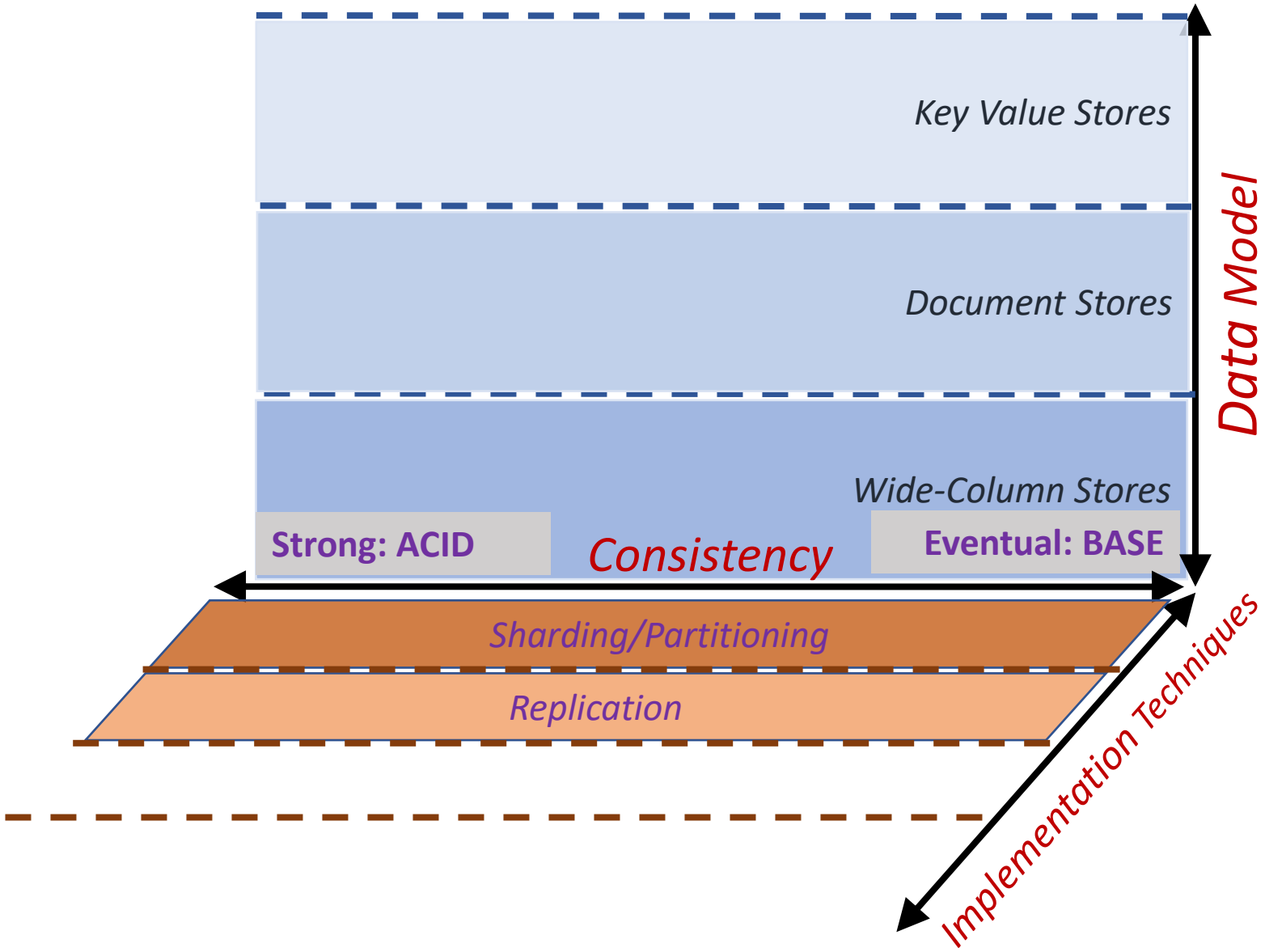
Review: noSQL Taxonomy



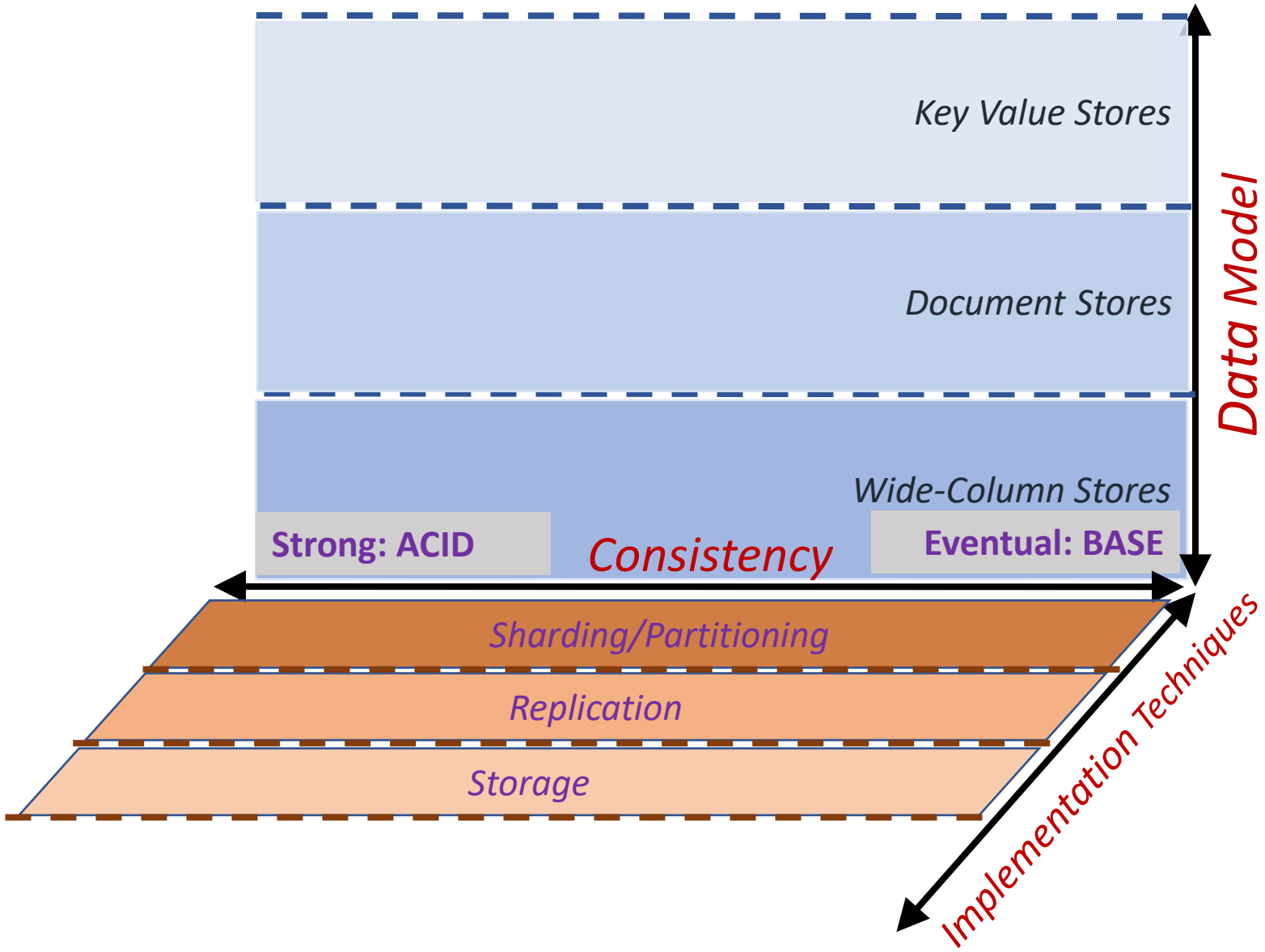
Review: noSQL Taxonomy



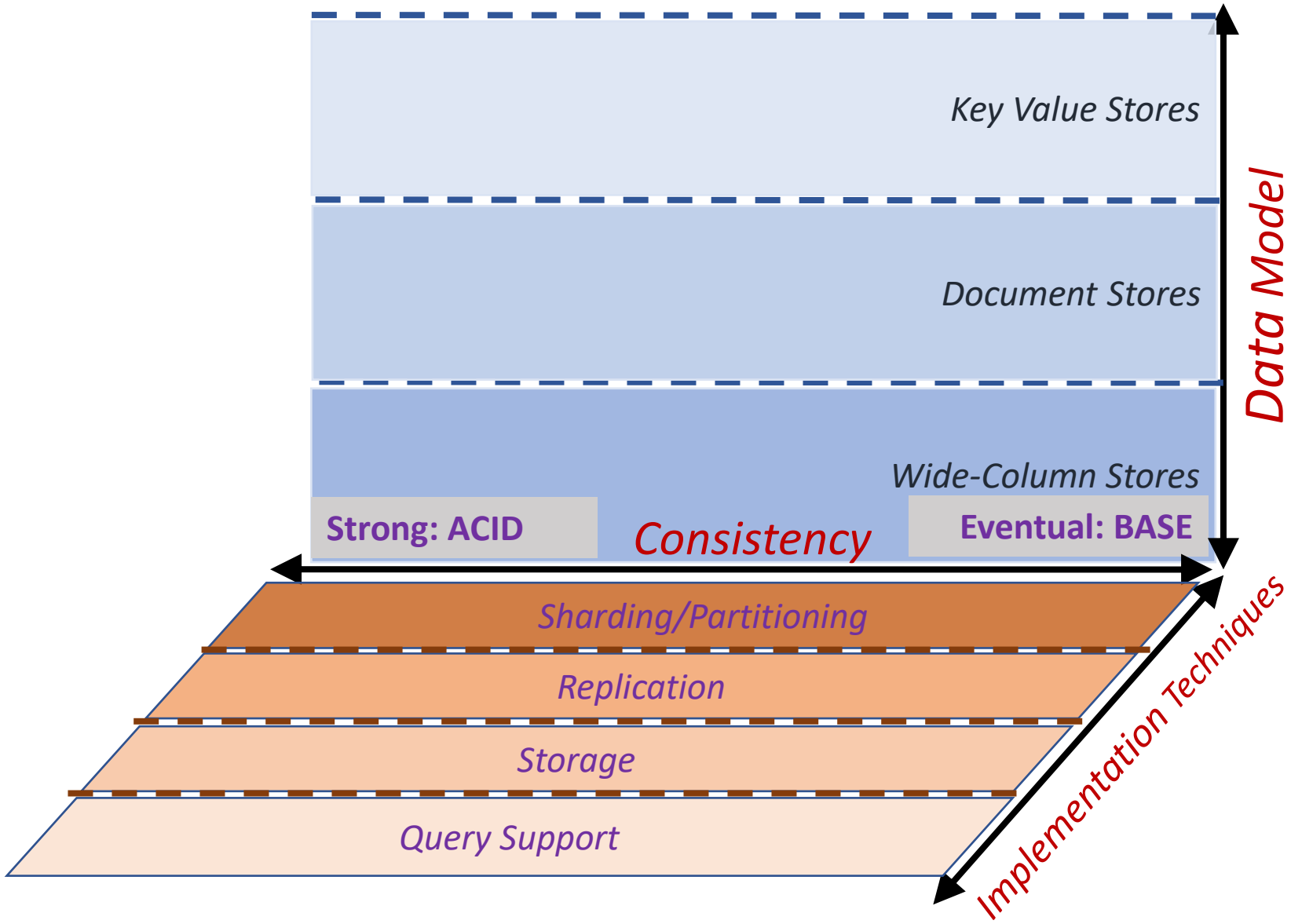
Review: noSQL Taxonomy



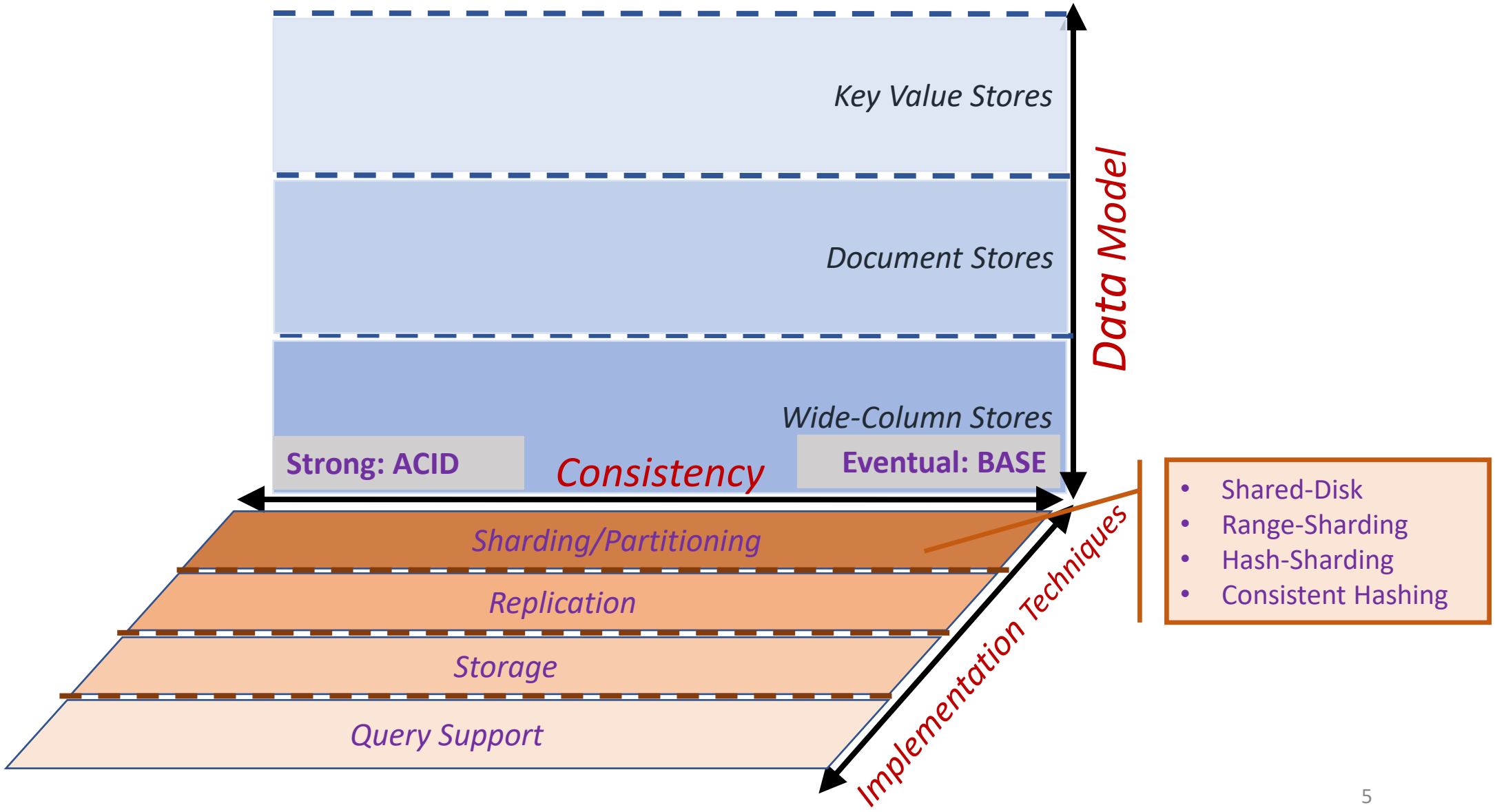
Review: noSQL Taxonomy



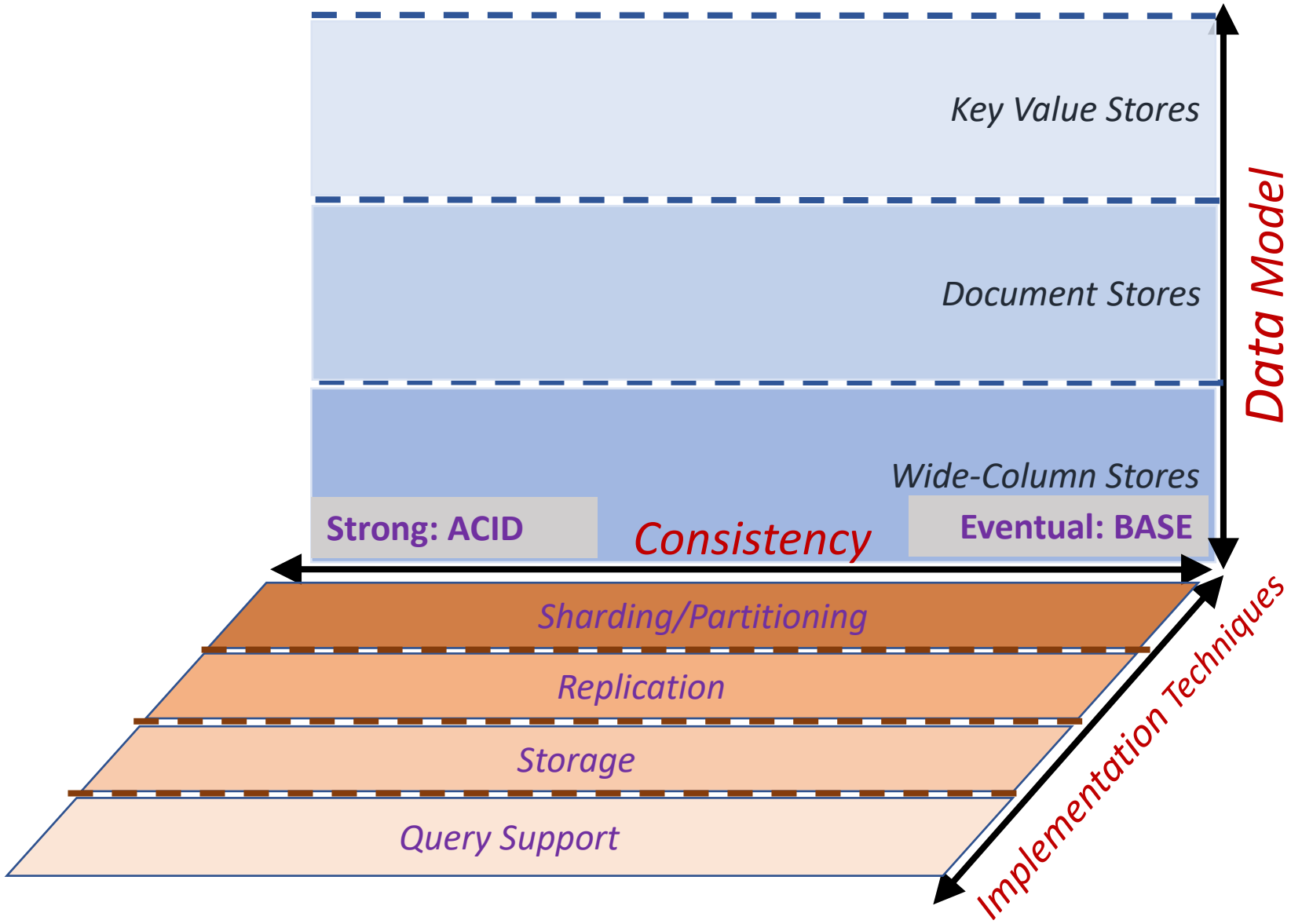
Review: noSQL Taxonomy



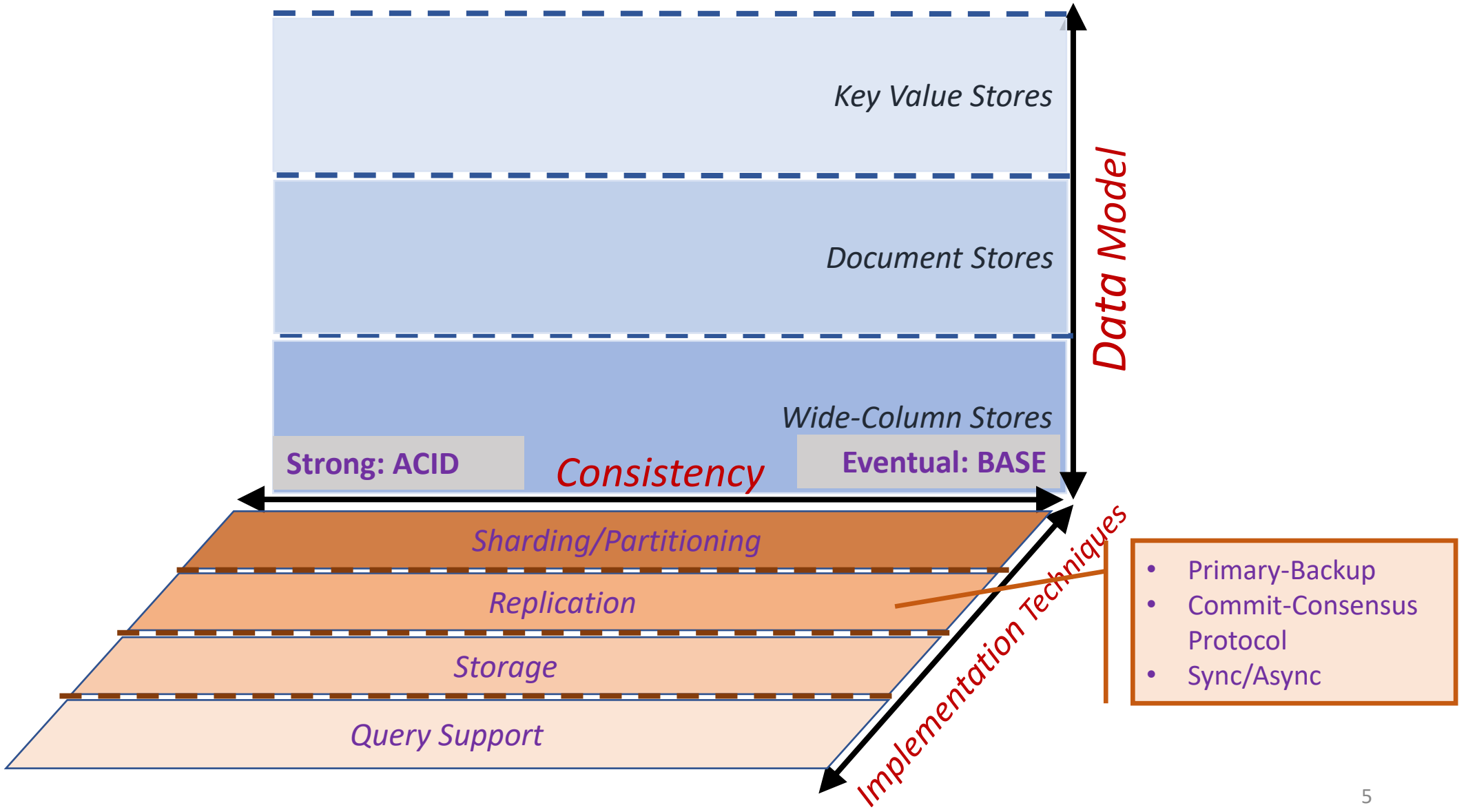
Review: noSQL Taxonomy



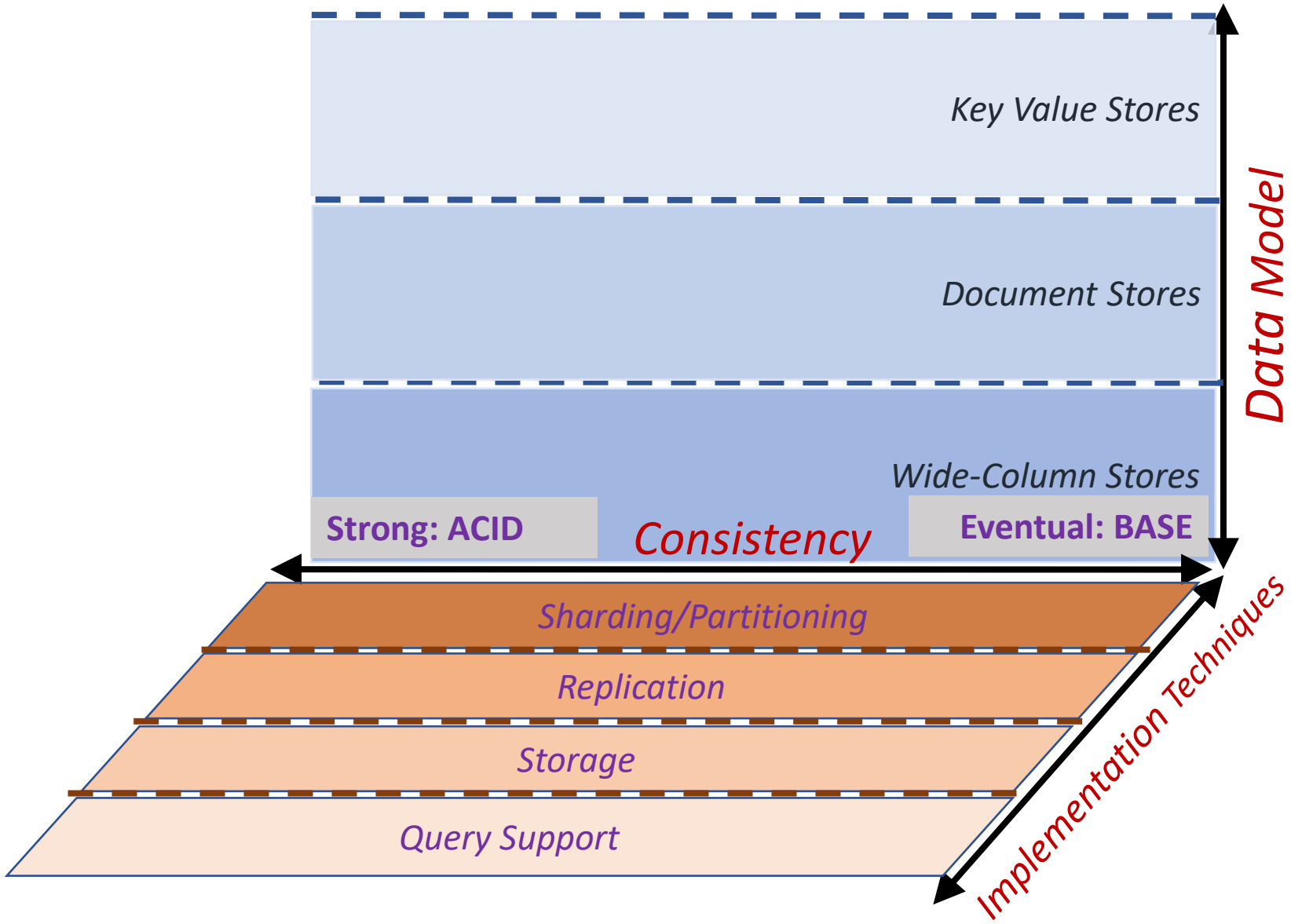
Review: noSQL Taxonomy



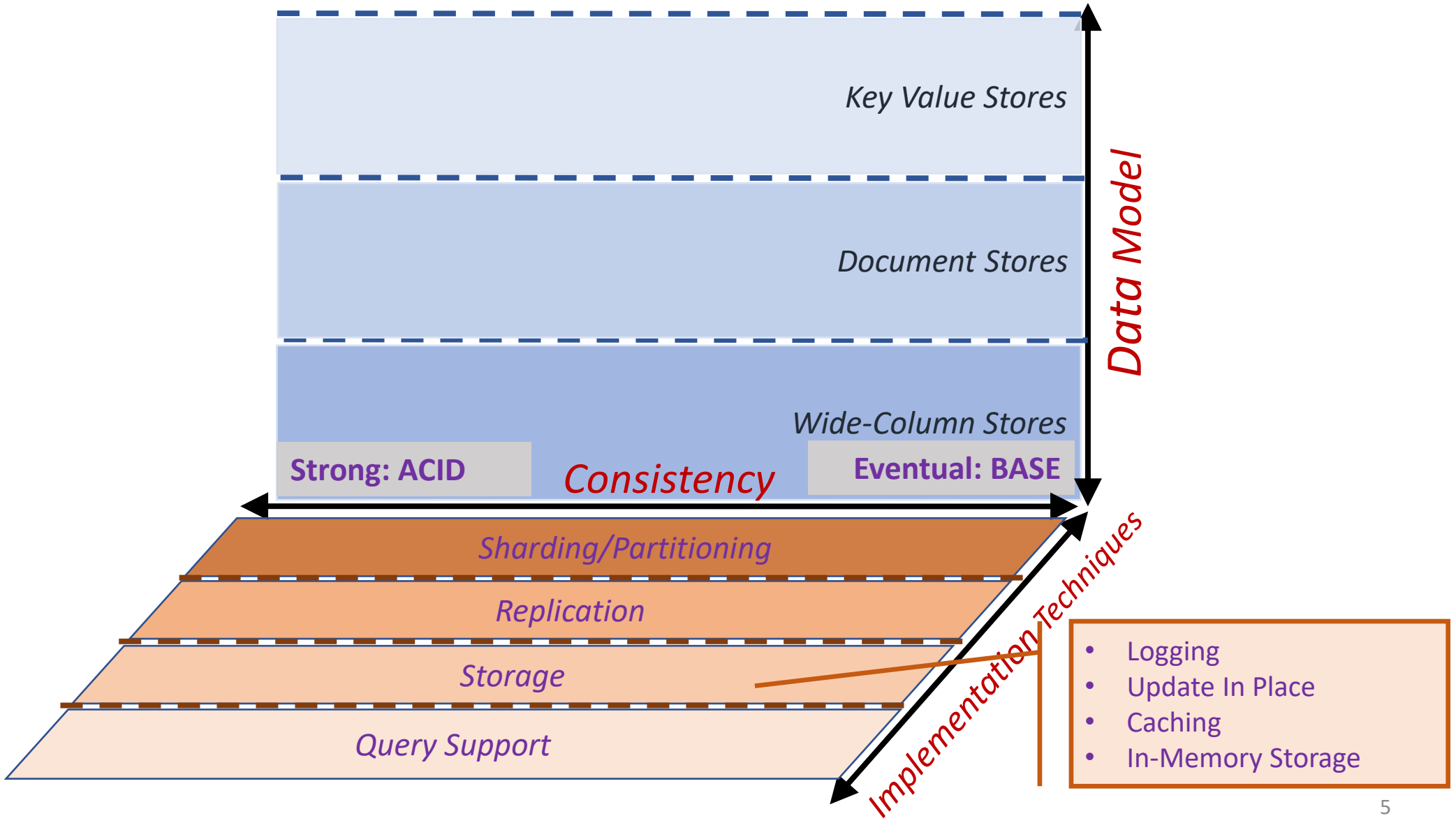
Review: noSQL Taxonomy



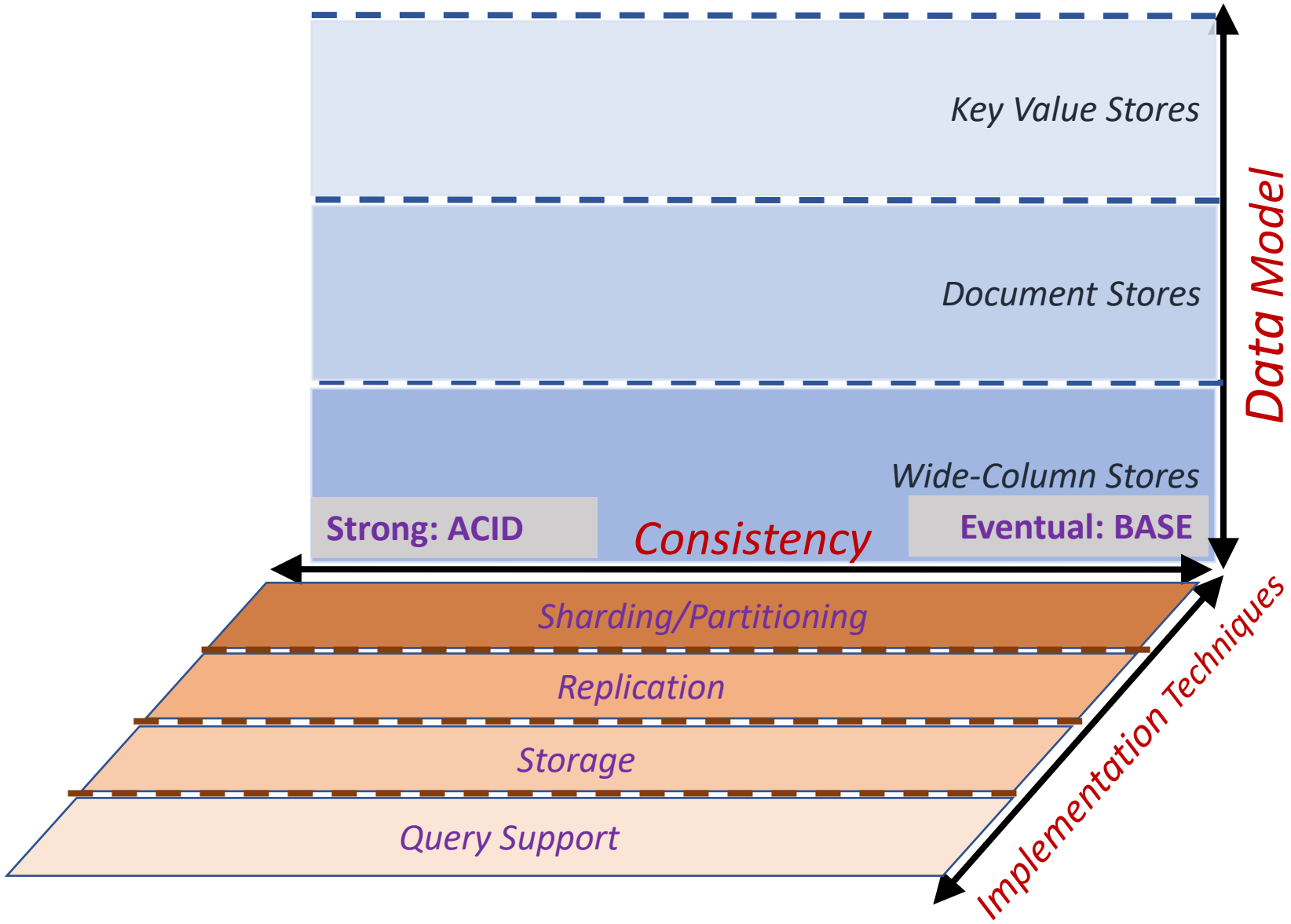
Review: noSQL Taxonomy



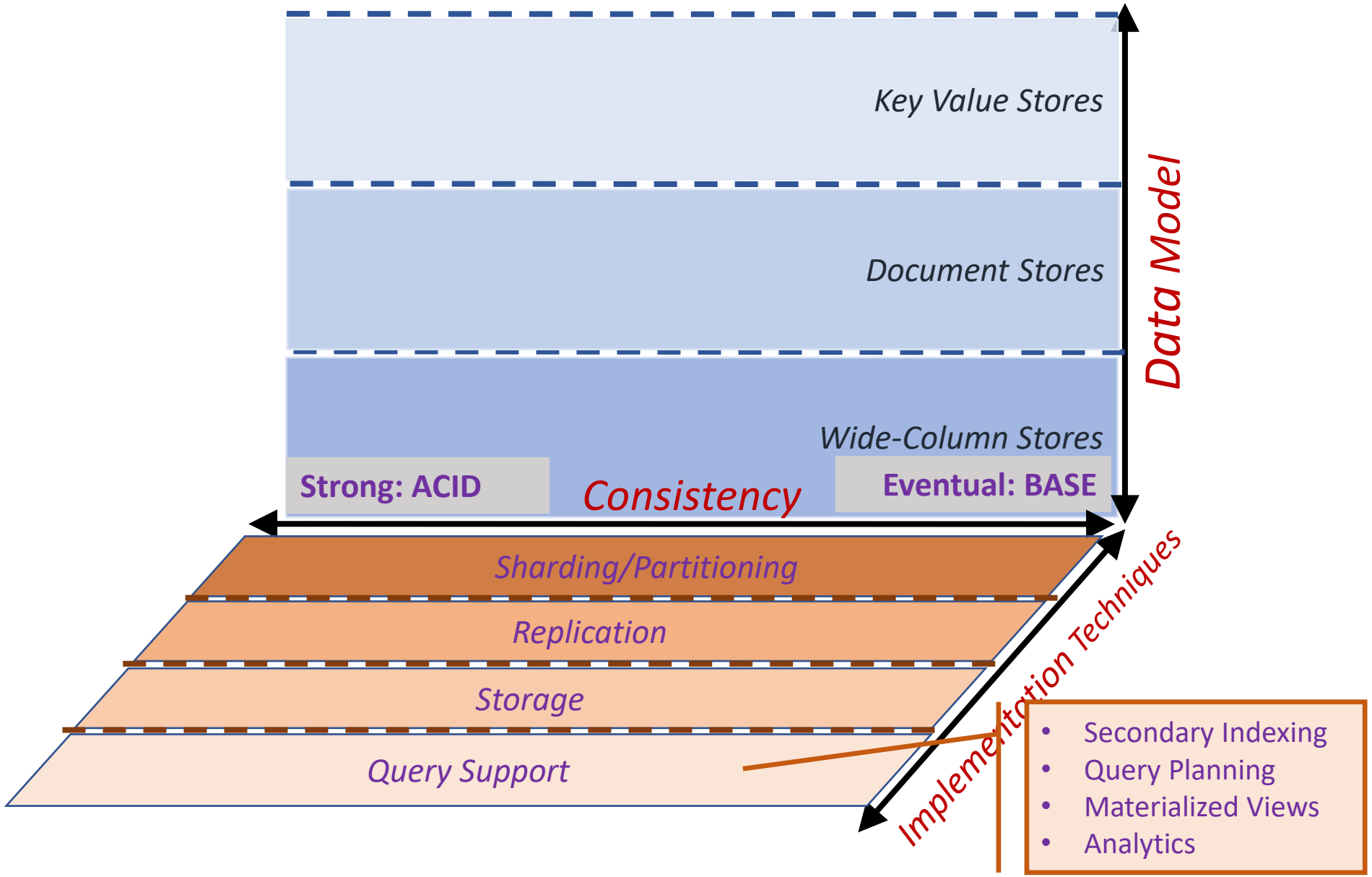
Review: noSQL Taxonomy



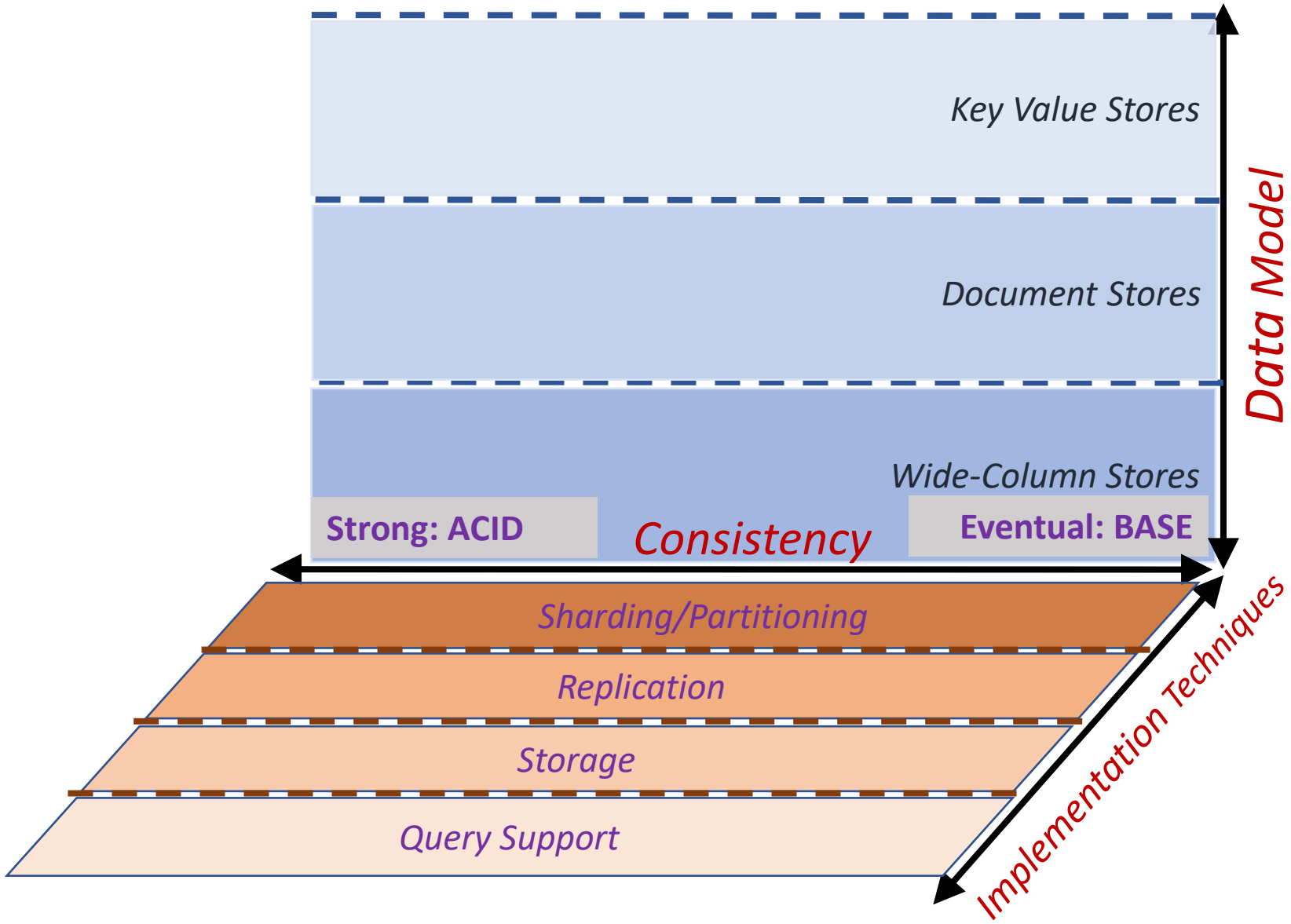
Review: noSQL Taxonomy



Review: noSQL Taxonomy



Review: noSQL Taxonomy



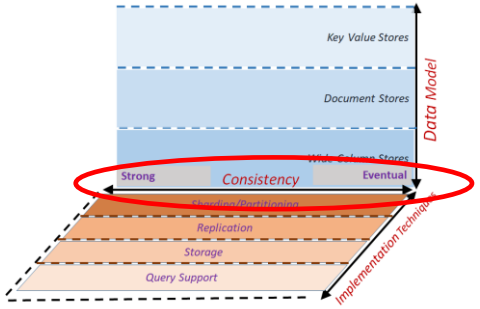
NoSQL faux quiz:

- What is the CAP theorem? What does “PACELC” 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

NoSQL faux quiz:

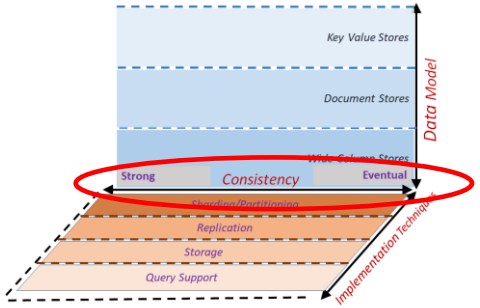
- What is the CAP theorem? What does “PACELC” stand for and how does it relate to CAP?
- What is the difference between ACID and BASE?
- Why do NoSQL systems claim to be more horizontally scalable than RDMBSes? ~~List some features NoSQL systems give up toward this goal?~~
- What is eventual consistency? 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

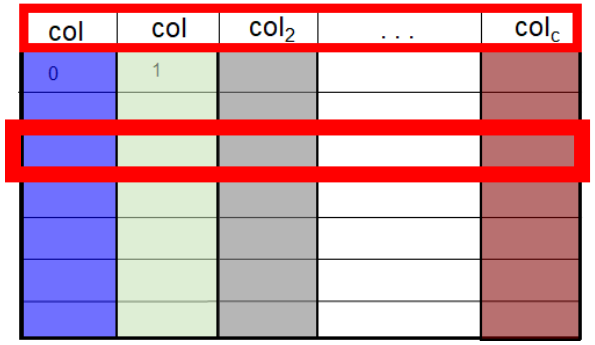
Consistency



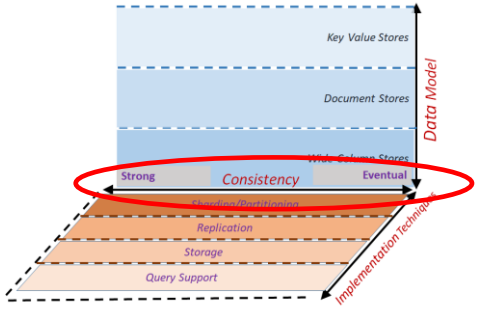
col	col	col_2	. . .	col_c
0	1			

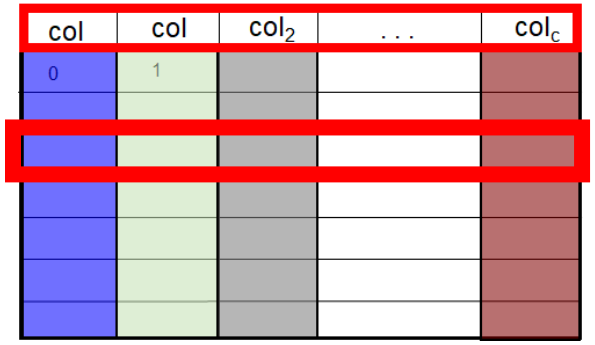
Consistency

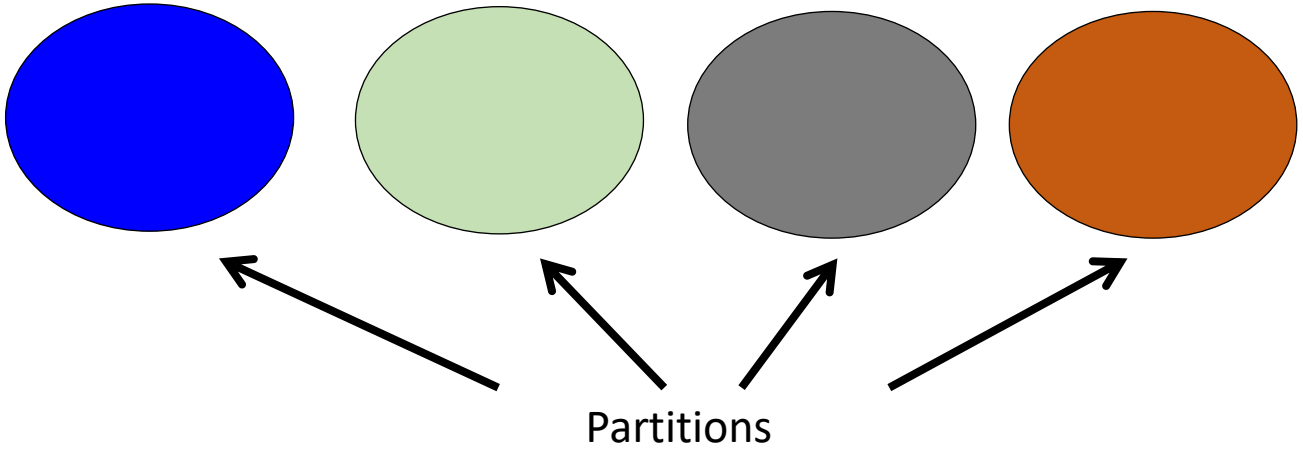




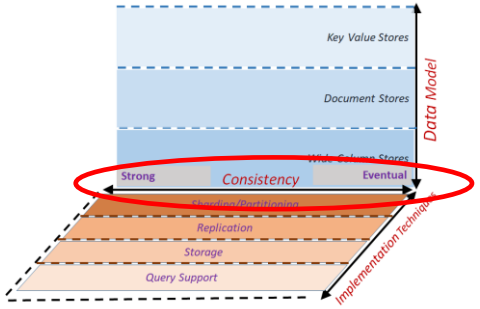
Consistency

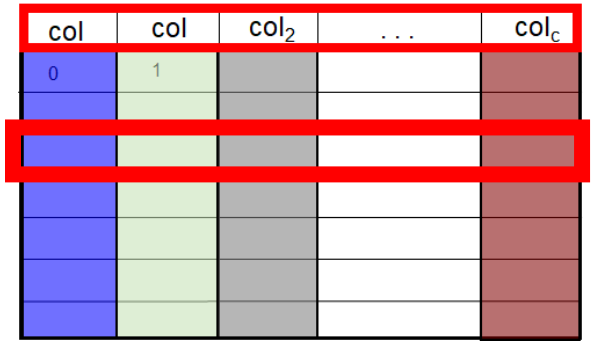


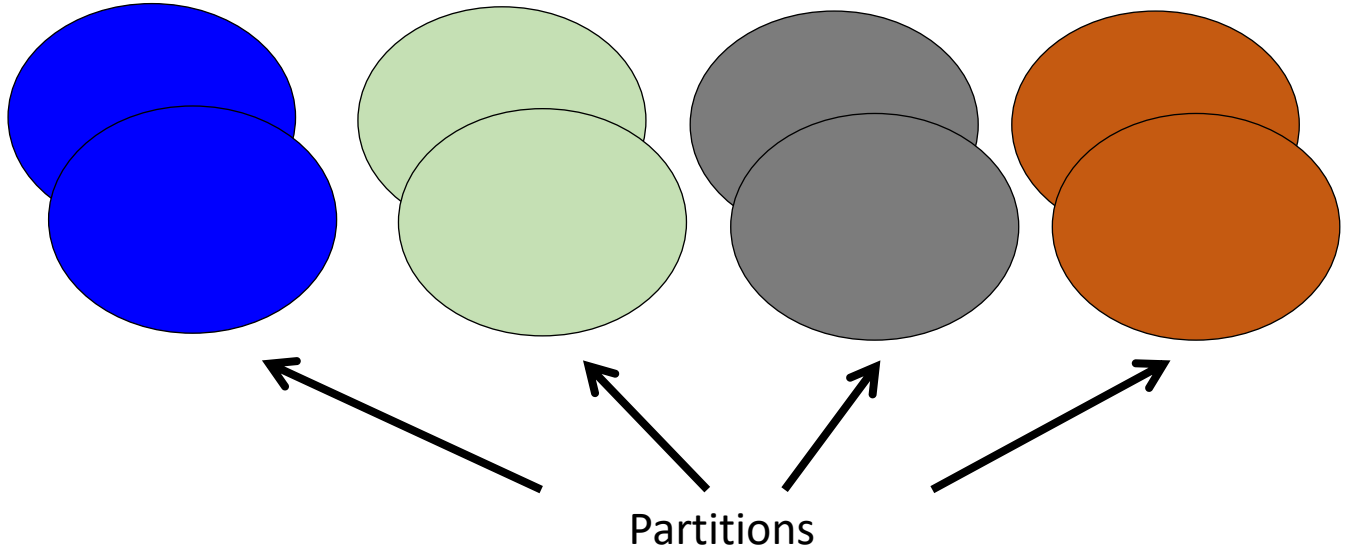




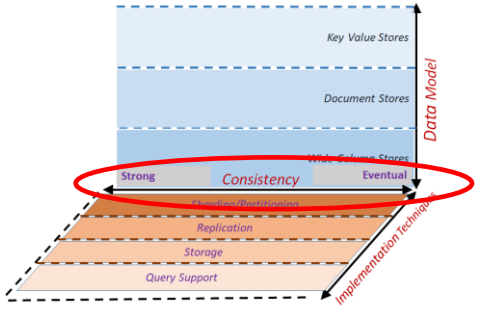
Consistency

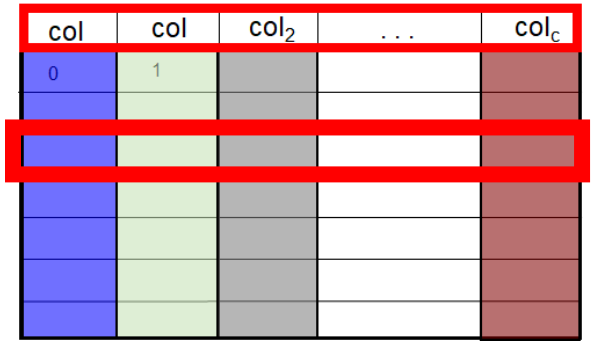


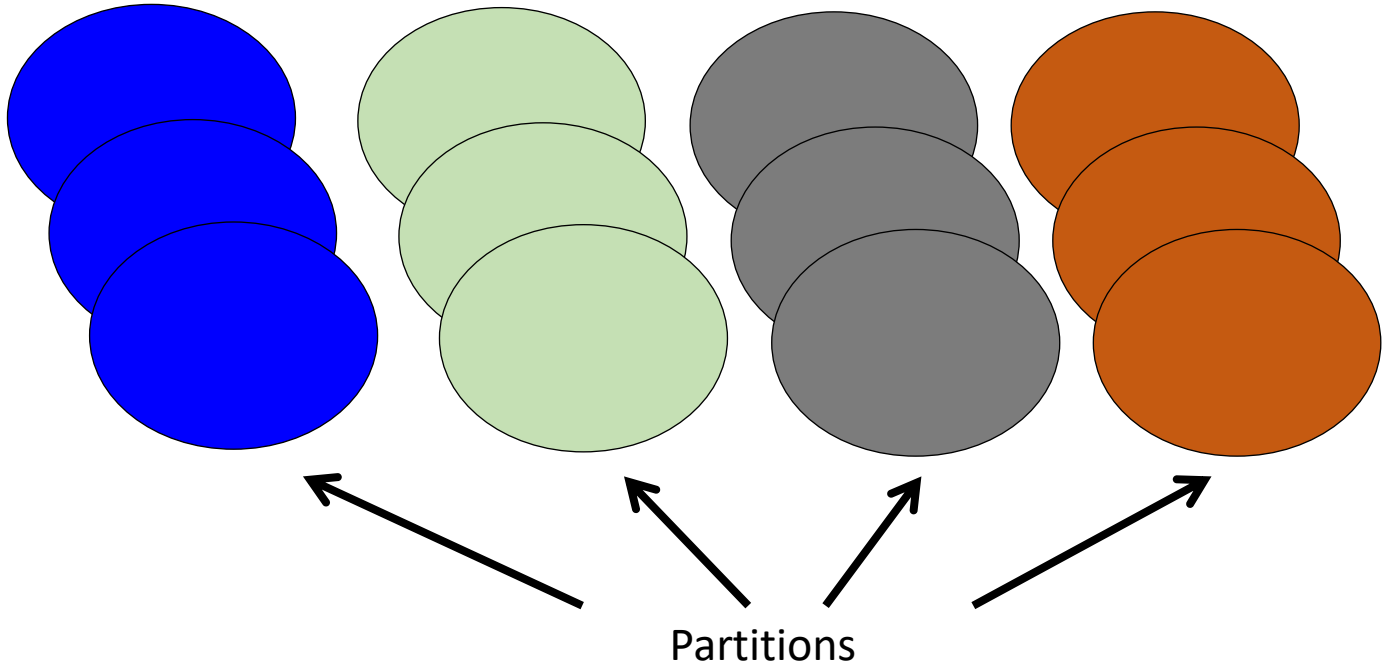




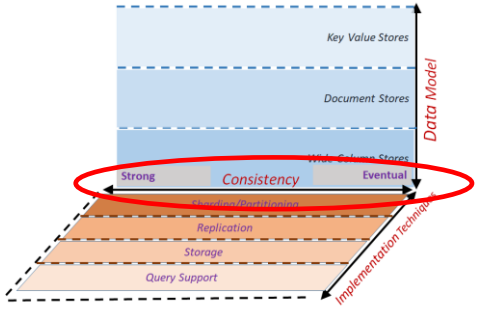
Consistency

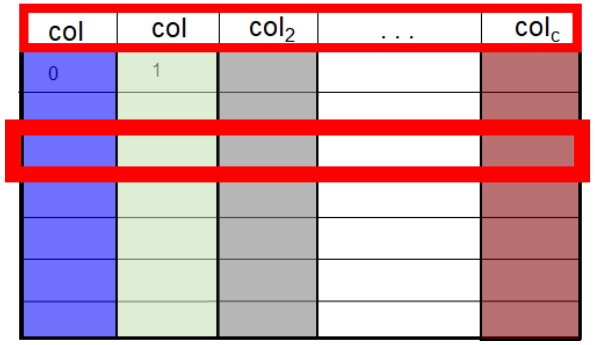


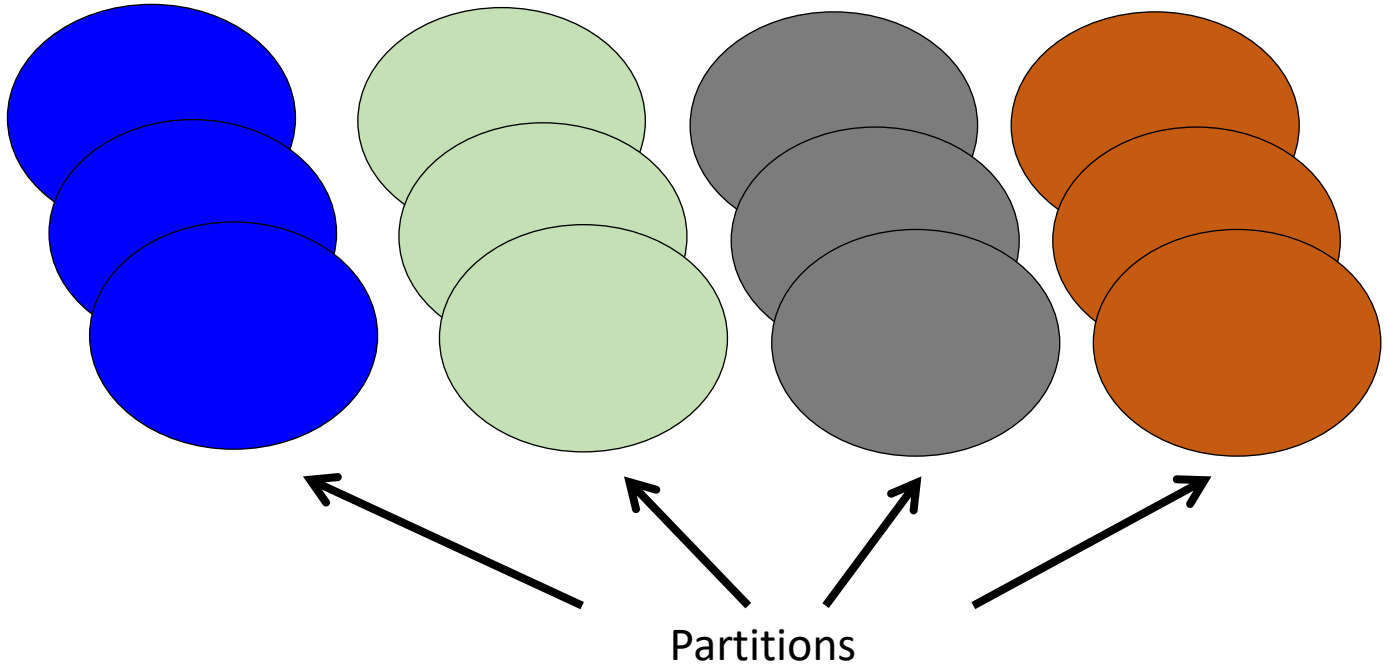




Consistency

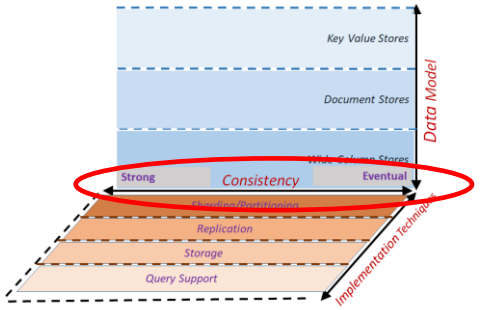


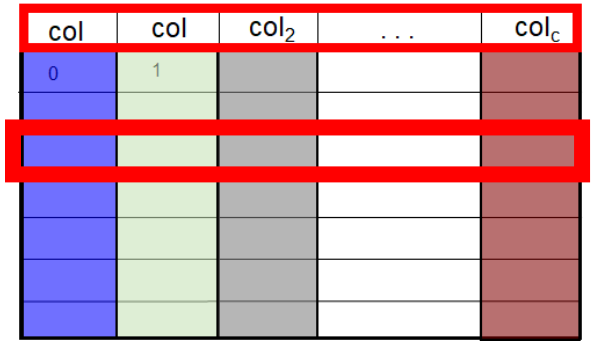


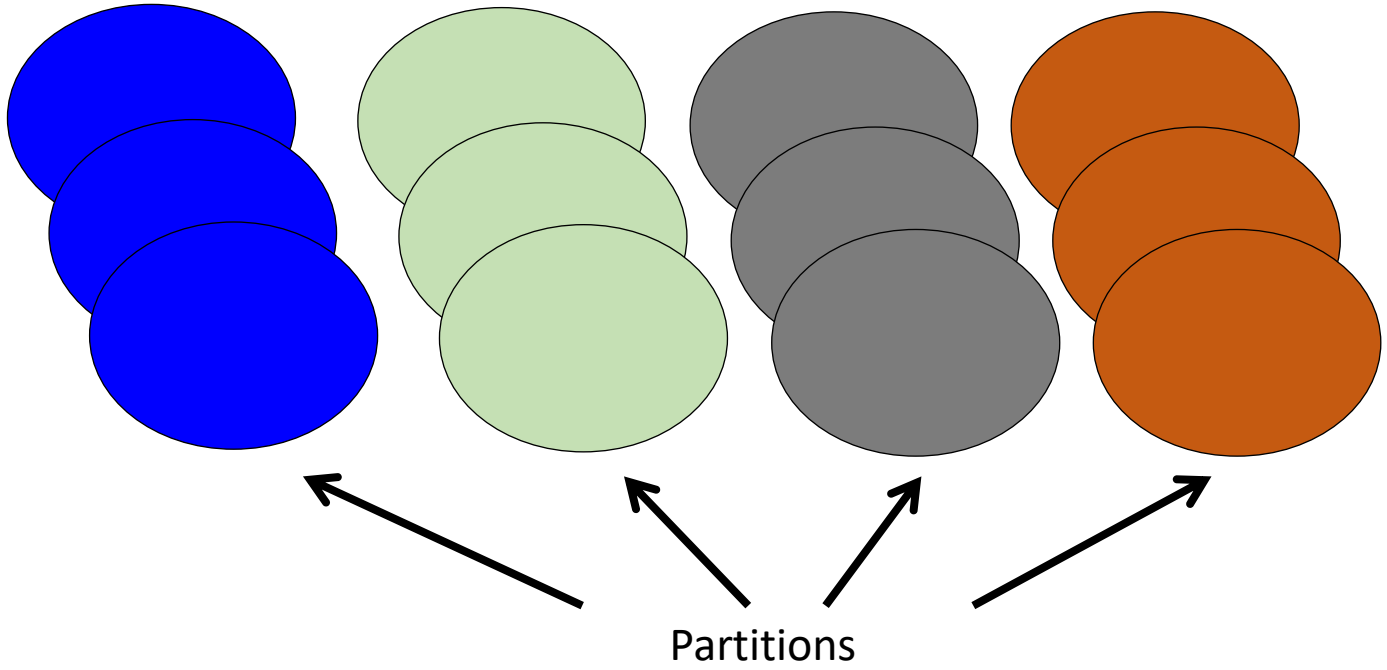


- Clients perform reads and writes

Consistency

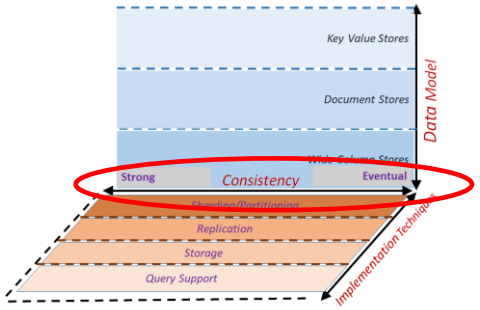


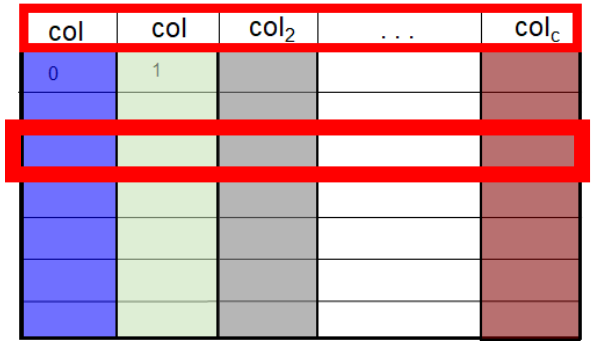


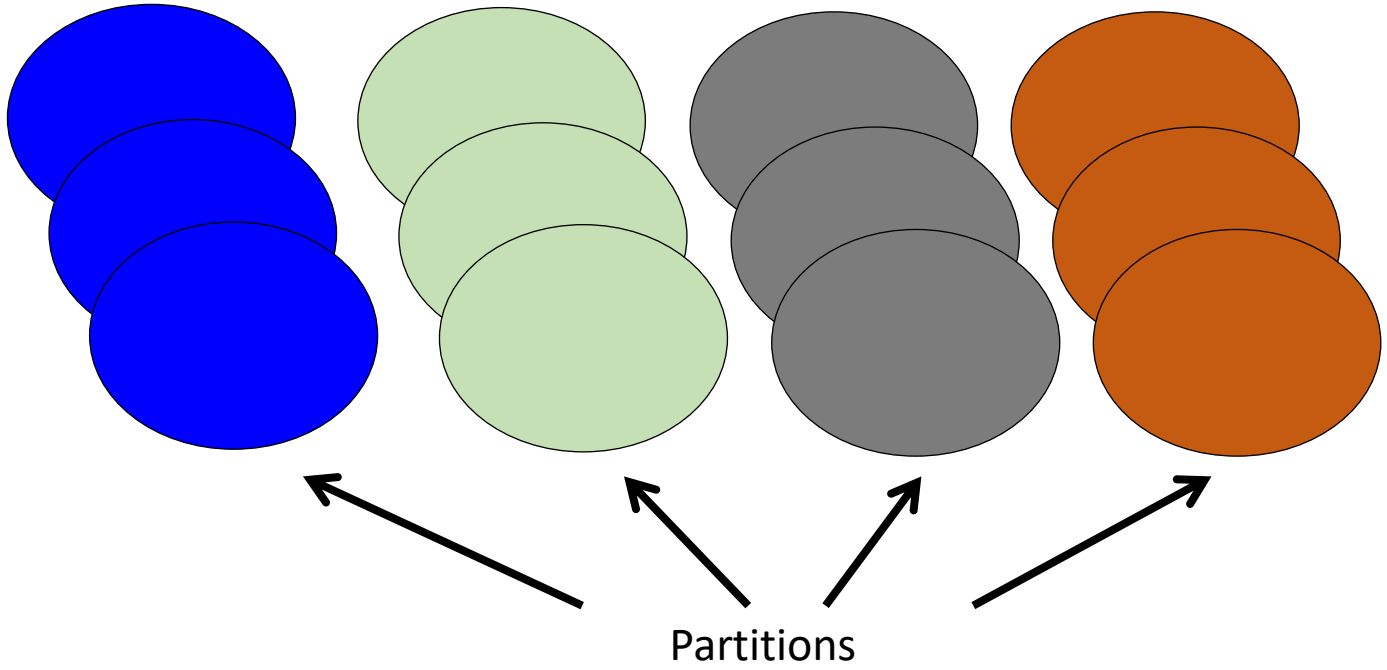


- Clients perform reads and writes
- Data is replicated among a set of servers

Consistency

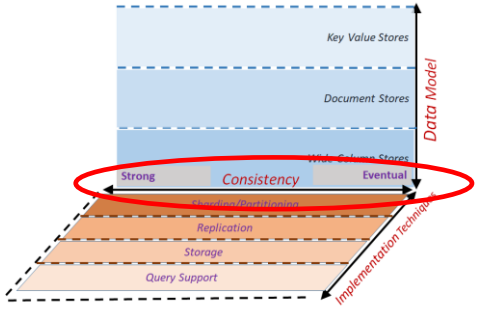


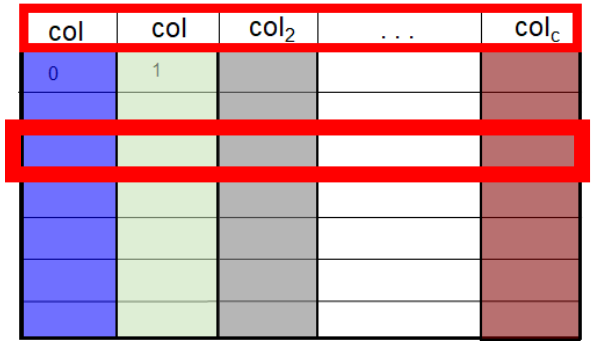


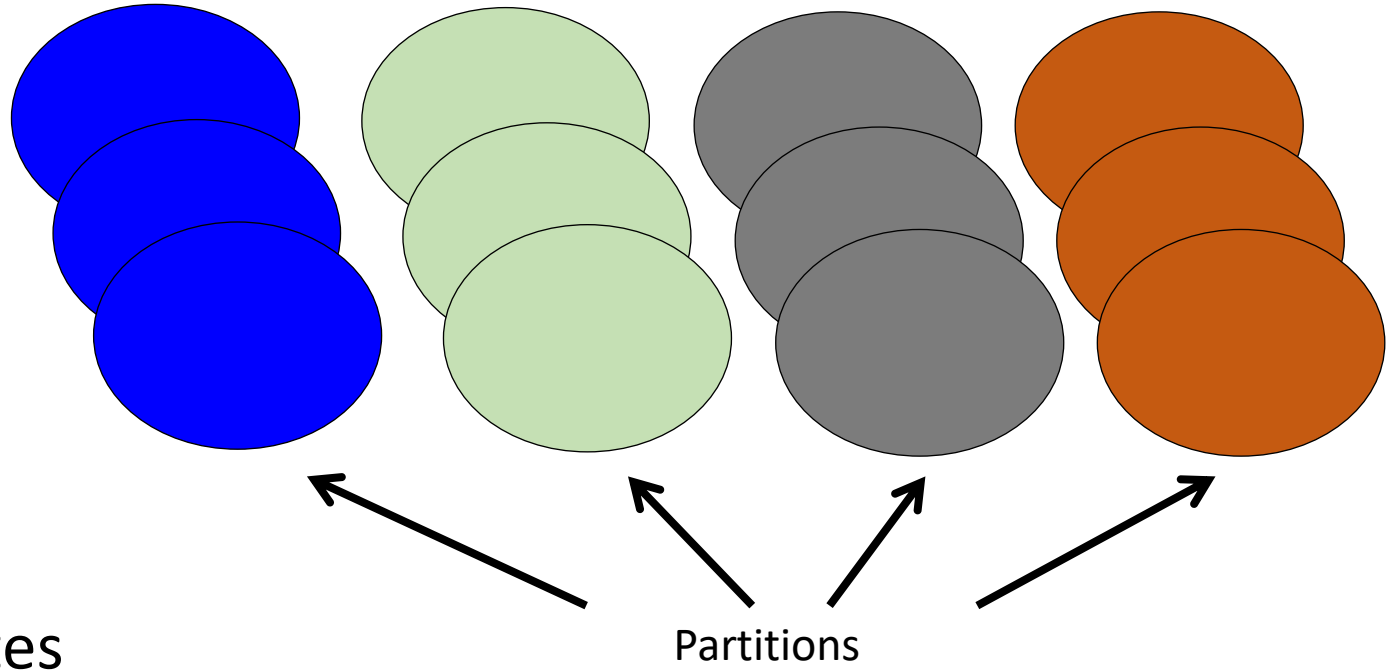


- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers

Consistency

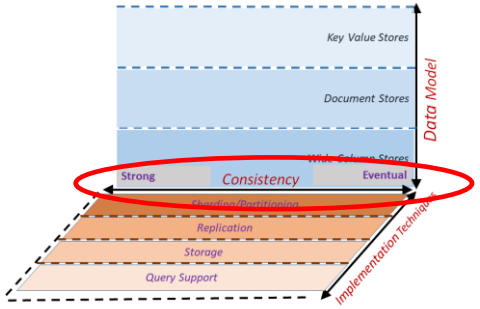


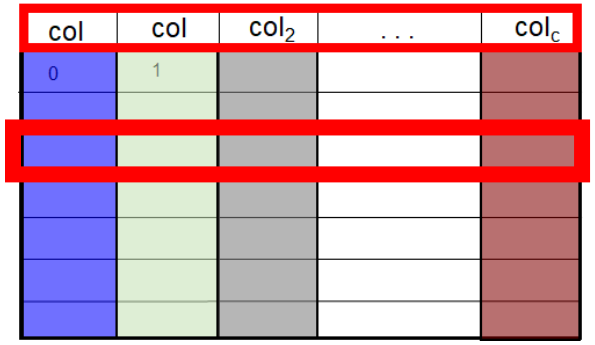


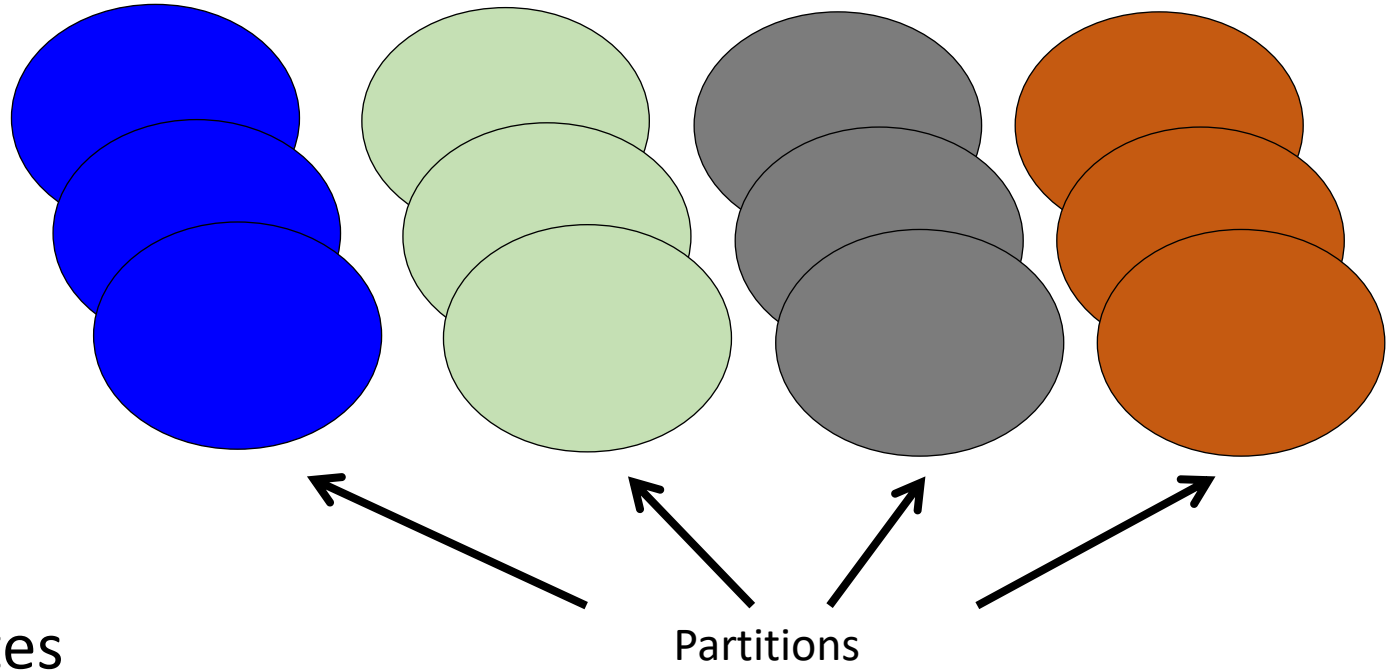


- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes

Consistency

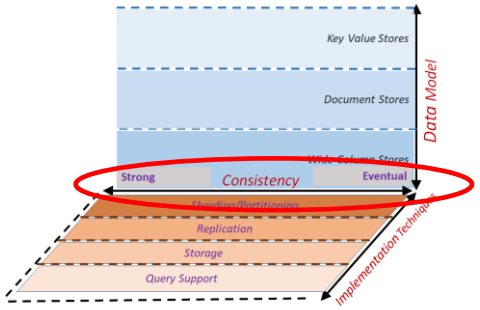


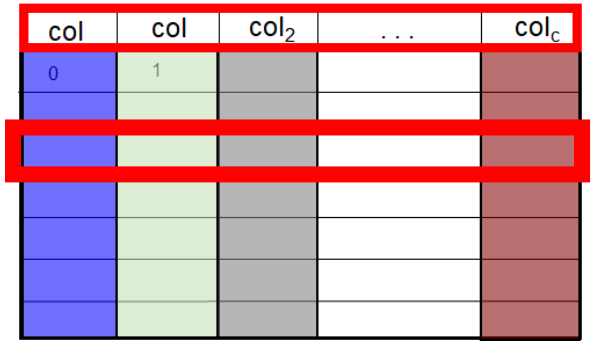


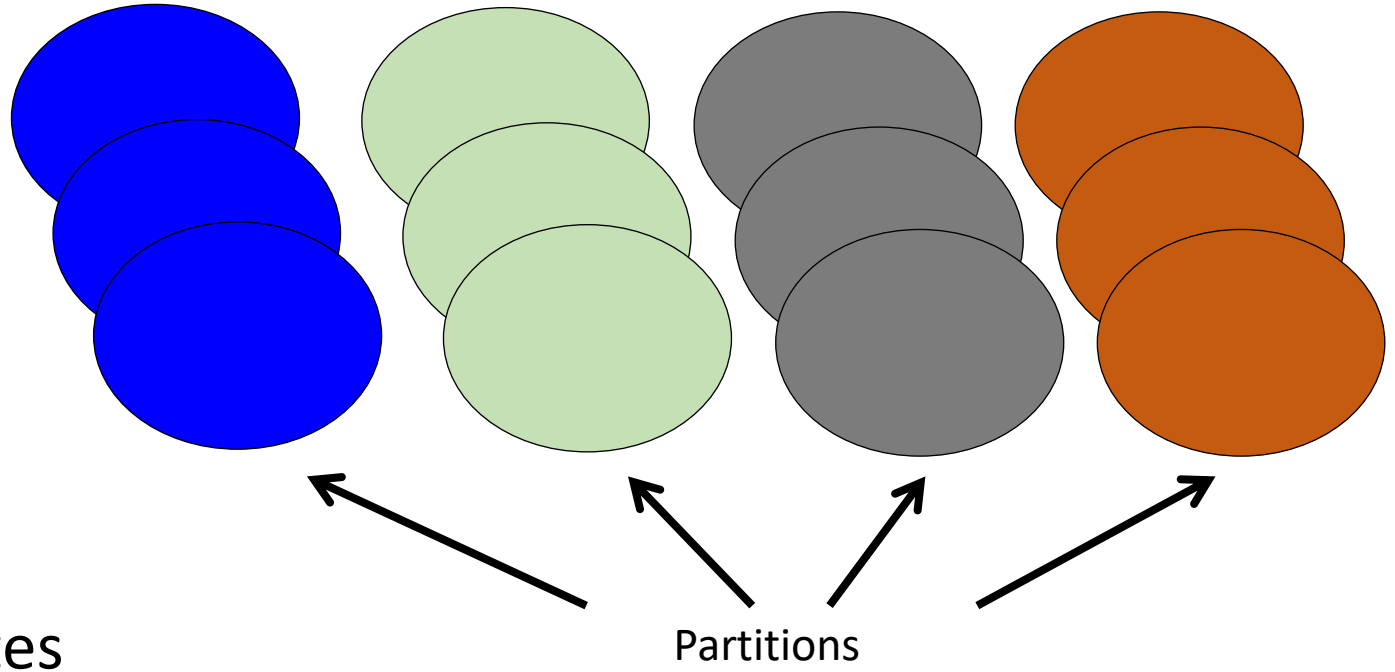


- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes
- How to keep data in sync?

Consistency





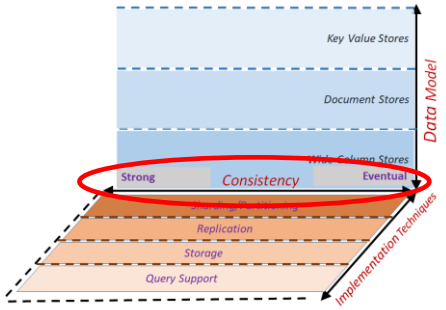


- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writ
- How to keep data in sync?

Consistency != Correctess

- consistency: no internal contradictions
- Correct: higher-level property
- Inconsistency → code does wrong things

Consistency Spectrum

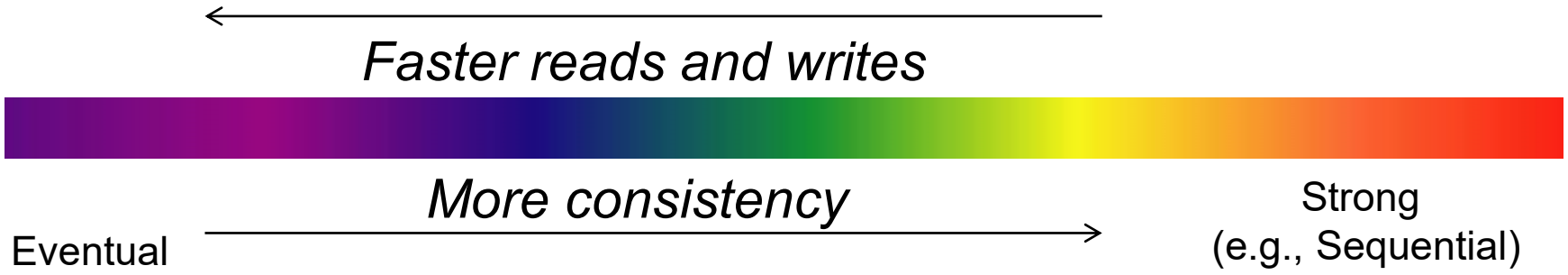


- Eventual Consistency
  - If writes to a key stop, all replicas of key will converge
  - Originally from Amazon's Dynamo and LinkedIn's Voldemort systems

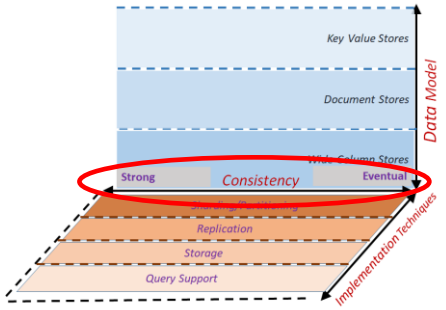
BASE:

- Basically Available
- Soft State
- Eventually Consistent

- **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



Consistency Spectrum



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

BASE:

- Basically Available
- Soft State
- Eventually Consistent

- **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

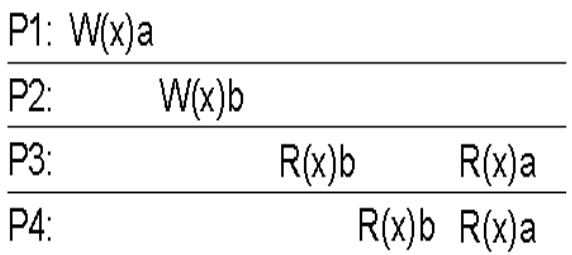
Eventual: BASE ← *Faster reads and writes* → **Strong: ACID**

More consistency

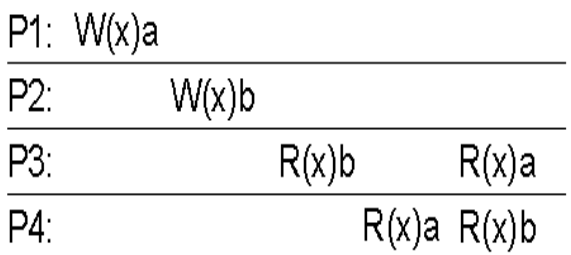
Eventual → Strong (e.g., Sequential)

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



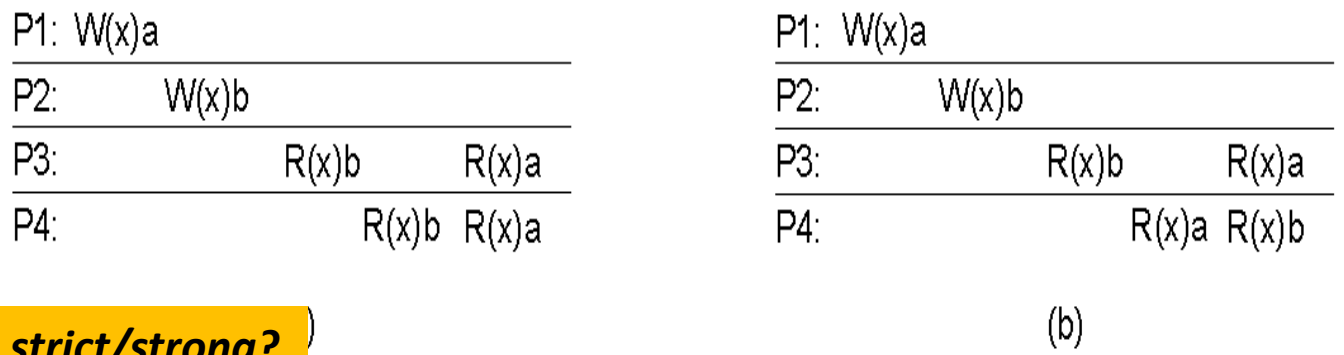
(a)



(b)

Sequential Consistency

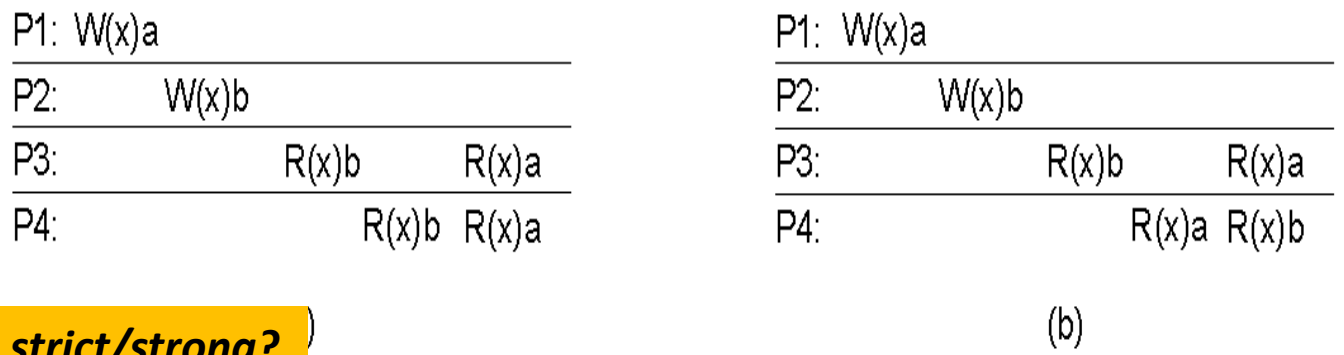
- weaker than strict/strong consistency
  - All operations are executed in *some* sequential order
  - each process issues operations in program order
    - Any valid interleaving is allowed
    - All agree on the same interleaving
    - Each process preserves its program order



- ***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



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

Linearizability vs. Serializability

http://www.bailis.org/blog/linearizability-versus-serializability/

Linearizability vs. Serializability

- Linearizability assumes sequential consistency *and*
  - If TS(x) < TS(y) then OP(x) should precede OP(y) in the sequence
  - Stronger than sequential consistency

http://www.bailis.org/blog/linearizability-versus-serializability/

Linearizability vs. Serializability

- 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

http://www.bailis.org/blog/linearizability-versus-serializability/

Linearizability vs. Serializability

- 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:

http://www.bailis.org/blog/linearizability-versus-serializability/

Linearizability vs. Serializability

- 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:

•Single-operation, single-object, real-time order

http://www.bailis.org/blog/linearizability-versus-serializability/

Linearizability vs. Serializability

- 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:

•Single-operation, single-object, real-time order

•Talks about order of ops on single object (e.g. atomic register)

http://www.bailis.org/blog/linearizability-versus-serializability/

Linearizability vs. Serializability

- 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:

- Single-operation, single-object, real-time order
- Talks about order of ops on single object (e.g. atomic register)
- Ops should appear instantaneous, reflect real time order

http://www.bailis.org/blog/linearizability-versus-serializability/

Linearizability vs. Serializability

- 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:

- Single-operation, single-object, real-time order
- Talks about order of ops on single object (e.g. atomic register)
- Ops should appear instantaneous, reflect real time order

Serializability:

http://www.bailis.org/blog/linearizability-versus-serializability/

Linearizability vs. Serializability

- 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:

- Single-operation, single-object, real-time order
- Talks about order of ops on single object (e.g. atomic register)
- Ops should appear instantaneous, reflect real time order

Serializability:

- Talks about groups of 1 or more ops on one or more objects

http://www.bailis.org/blog/linearizability-versus-serializability/

Linearizability vs. Serializability

- 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:

- Single-operation, single-object, real-time order
- Talks about order of ops on single object (e.g. atomic register)
- Ops should appear instantaneous, reflect real time order

Serializability:

- Talks about groups of 1 or more ops on one or more objects
- Txns over multiple items equivalent to serial order of txns

http://www.bailis.org/blog/linearizability-versus-serializability/

Linearizability vs. Serializability

- 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:

•Single-operation, single-object, real-time order

•Talks about order of ops on single object (e.g. atomic register)

•Ops should appear instantaneous, reflect real time order

Serializability:

•Talks about groups of 1 or more ops on one or more objects

•Txns over multiple items equivalent to serial order of txns

•Only requires *some* equivalent serial order

http://www.bailis.org/blog/linearizability-versus-serializability/

Linearizability vs. Serializability

- 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:
•Single-operation, single-object, real-time order
•Talks about order of ops on single object (e.g. atomic register)
•Ops should appear instantaneous, reflect real time order

Serializability:
•Talks about groups of 1 or more ops on one or more objects
•Txns over multiple items equivalent to serial order of txns
•Only requires *some* equivalent serial order

Serializability + Linearizability == "Strict Serializability"
- Txn order equivalent to some serial order ***that respects real time order***
- Linearizability: degenerate case of Strict Ser: txns are single op single object

http://www.bailis.org/blog/linearizability-versus-serializability/

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 seen
  - *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

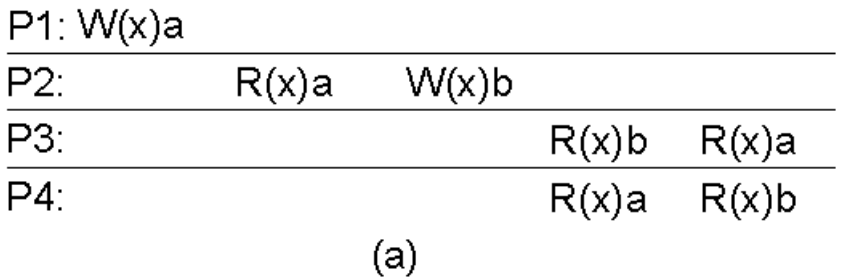
- Causally related writes seen by all processes in same order.
  - *Causally?*

Causal consistency

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

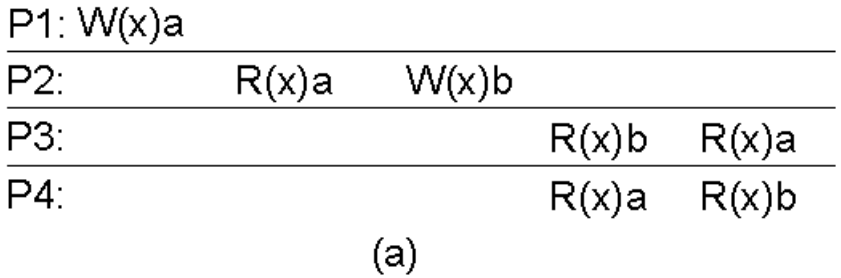
Causal consistency

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



Causal consistency

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

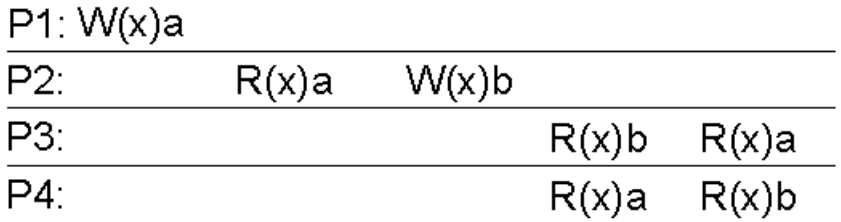


(a)

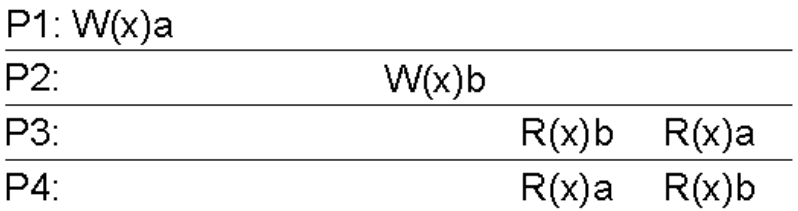
Not permitted

Causal consistency

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



(a)

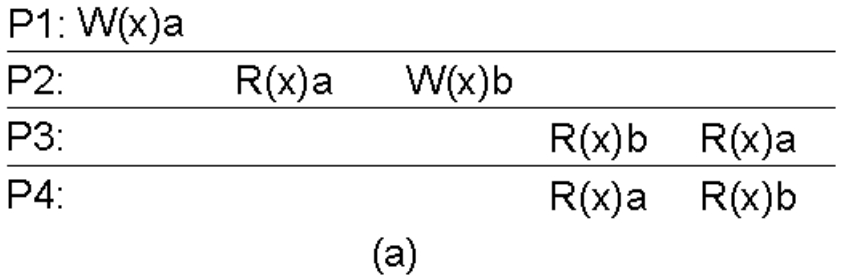


(b)

Not permitted

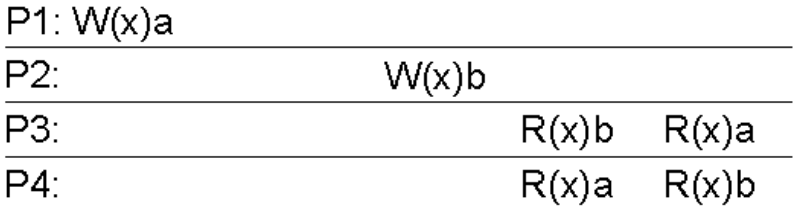
Causal consistency

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



(a)

Not permitted



(b)

Permitted

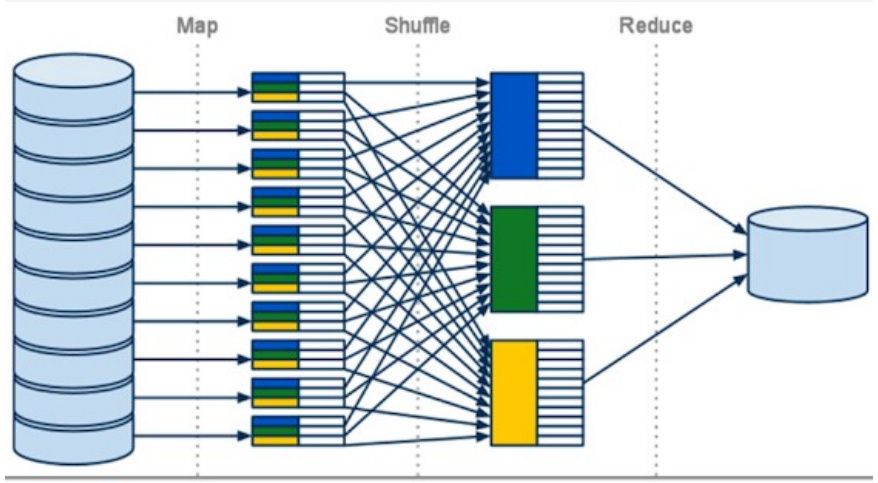
Dataflow

Dataflow

- MR is a ***dataflow*** engine

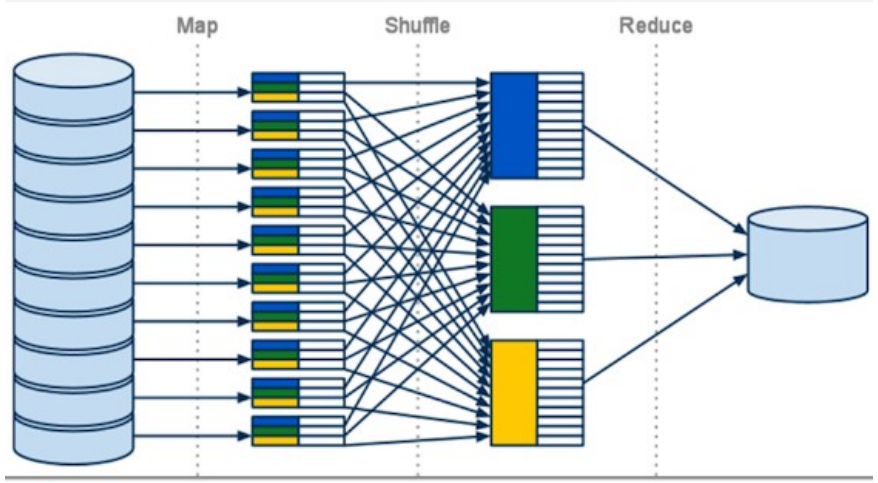
Dataflow

- MR is a ***dataflow*** engine



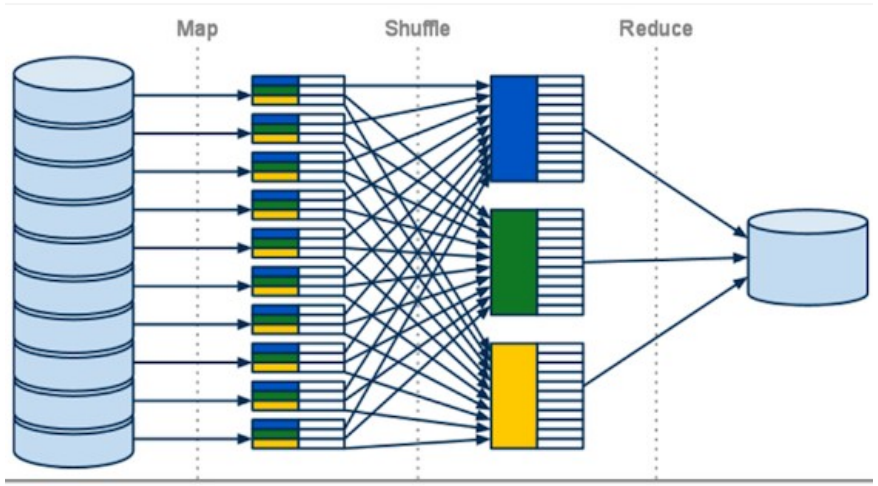
Dataflow

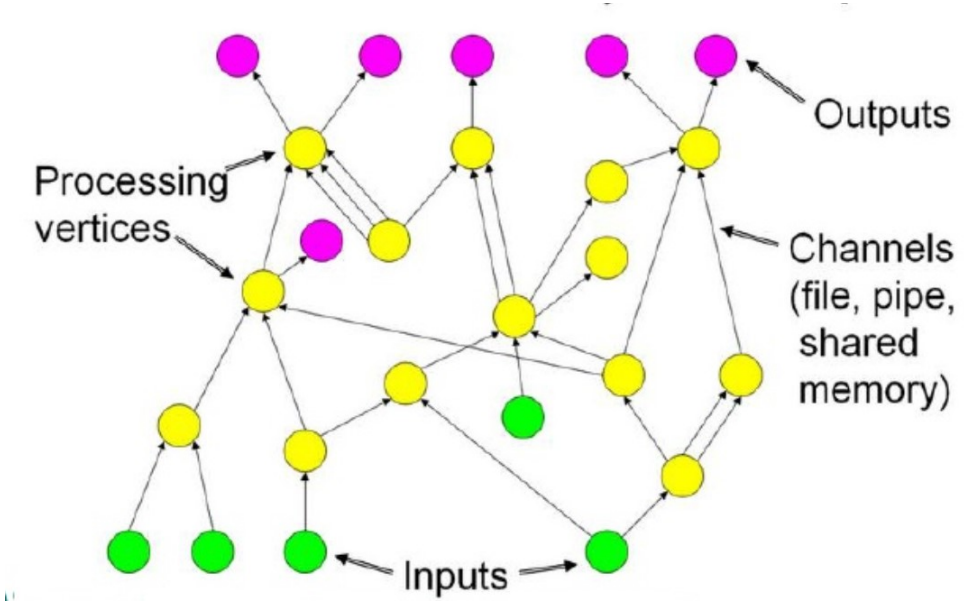
- MR is a ***dataflow*** engine
- So are Lots of others
  - Dryad
  - DryadLINQ
  - Dandelion
  - CIEL
  - GraphChi/PowerGraph/Pregel
  - Spark



Dataflow

- MR is a ***dataflow*** engine
- So are Lots of others
  - Dryad
  - DryadLINQ
  - Dandelion
  - CIEL
  - GraphChi/PowerGraph/Pregel
  - Spark



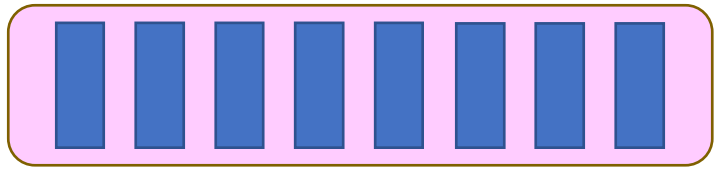


Spark faux quiz (5 min, any 2):

- What is the difference between *transformations* and *actions* in Spark?
- Spark supports a persist API. When should a programmer want to use it? When should she [not] use use the "*RELIABLE*" flag?
- Compare and contrast fault tolerance guarantees of Spark to those of MapReduce. How are[n't] the mechanisms different?
- Is Spark a good system for indexing the web? For computing page rank over a web index? Why [not]?
- List aspects of Spark's design that help/hinder multi-core parallelism relative to MapReduce. If the issue is orthogonal, explain why.

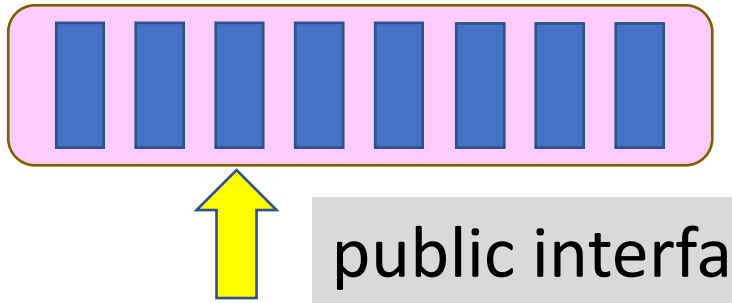
Collections and Iterators

class Collection<T> : IEnumerable<T>;



Collections and Iterators

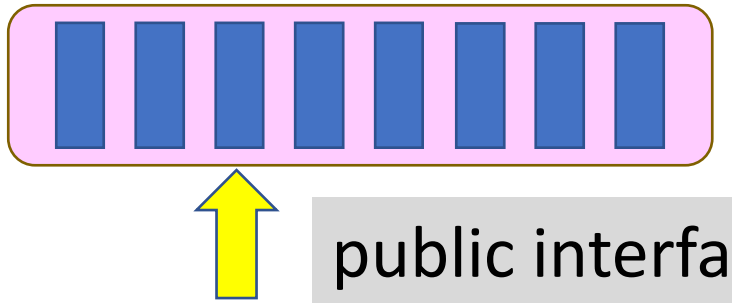
class Collection<T> : IEnumerable<T>;



```
public interface IEnumerable<T>  {
        IEnumerator<T> GetEnumerator();
}
```

Collections and Iterators

class Collection<T> : IEnumerable<T>;

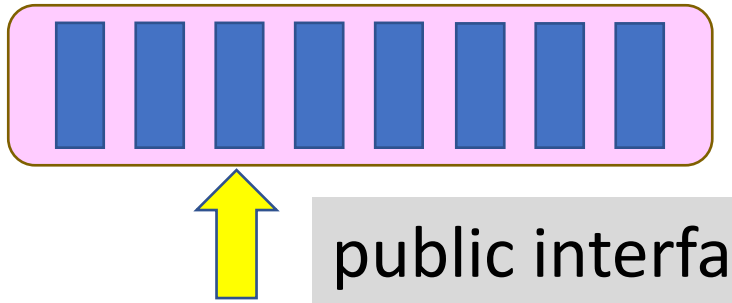


```
public interface IEnumerable<T>  {
        IEnumerator<T> GetEnumerator();
}
```

```
public interface IEnumerator <T> {
        T Current { get; }
        bool MoveNext();
        void Reset();
}
```

Collections and Iterators

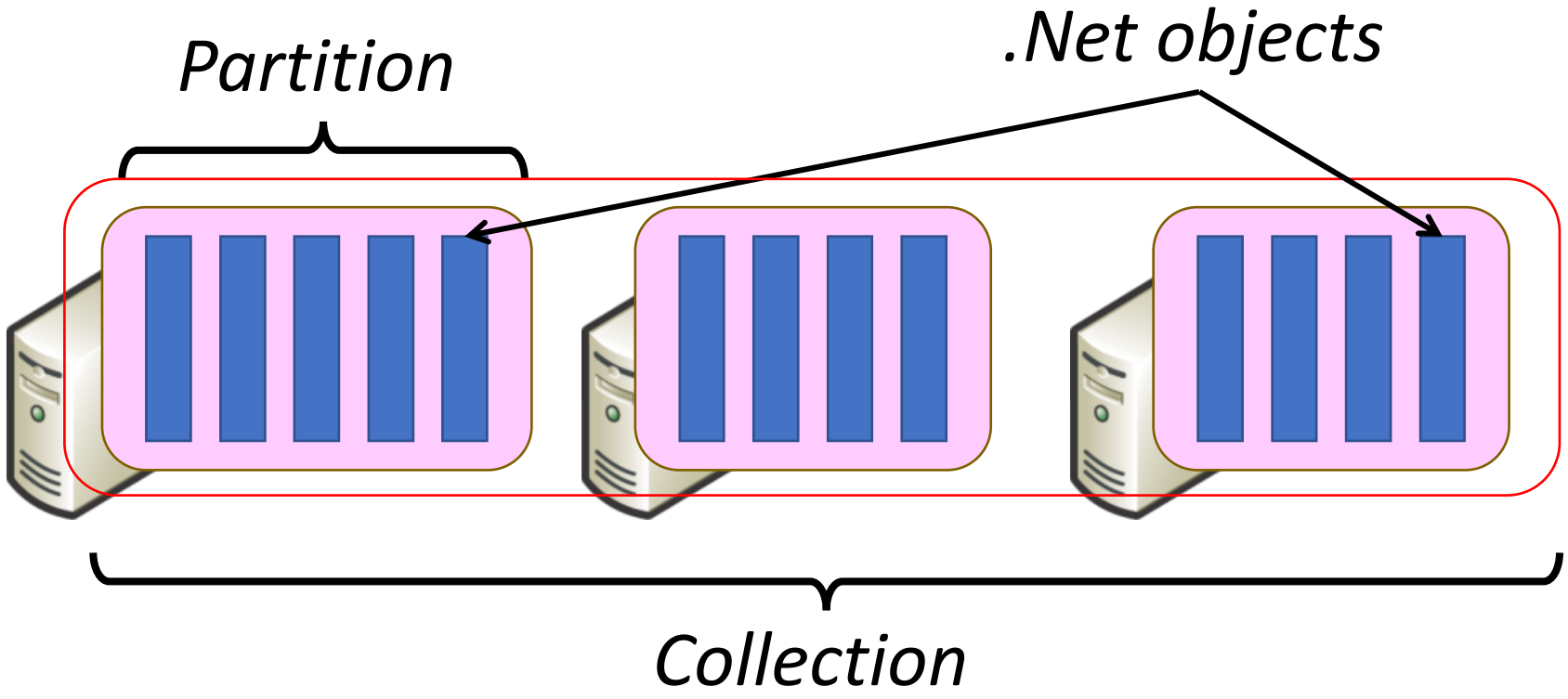
class Collection<T> : IEnumerable<T>;



```
public interface IEnumerable<T>  {
        IEnumerator<T> GetEnumerator();
}
```

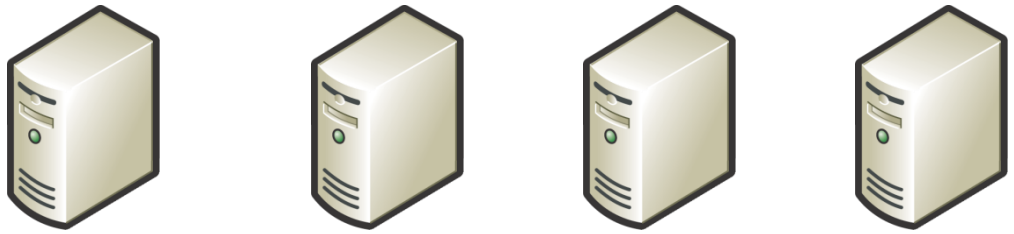
```
public interface IEnumerator <T> {
        T Current { get; }
        bool MoveNext();
        void Reset();
}
```

DryadLINQ Data Model

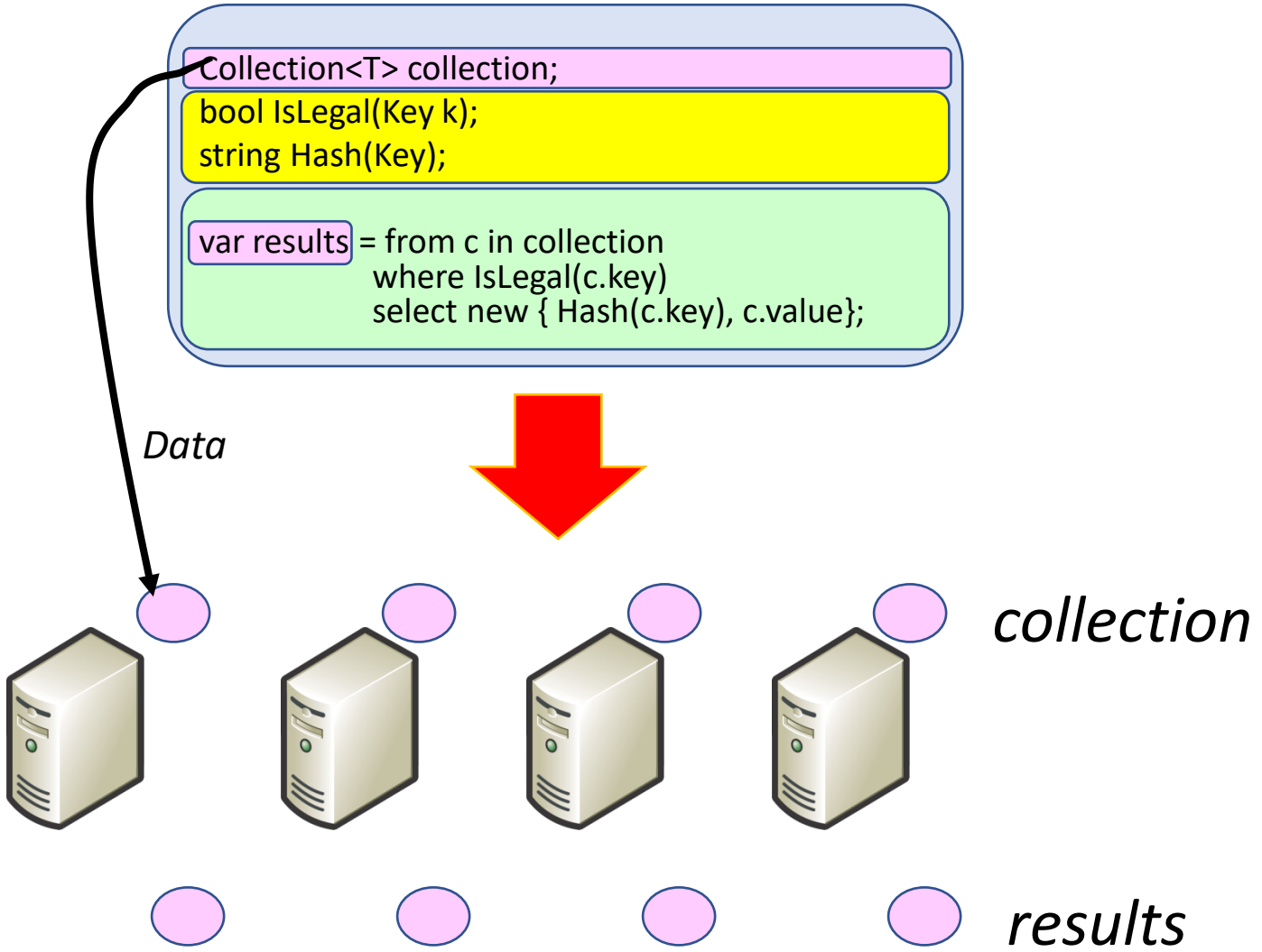


DryadLINQ = LINQ + Dryad

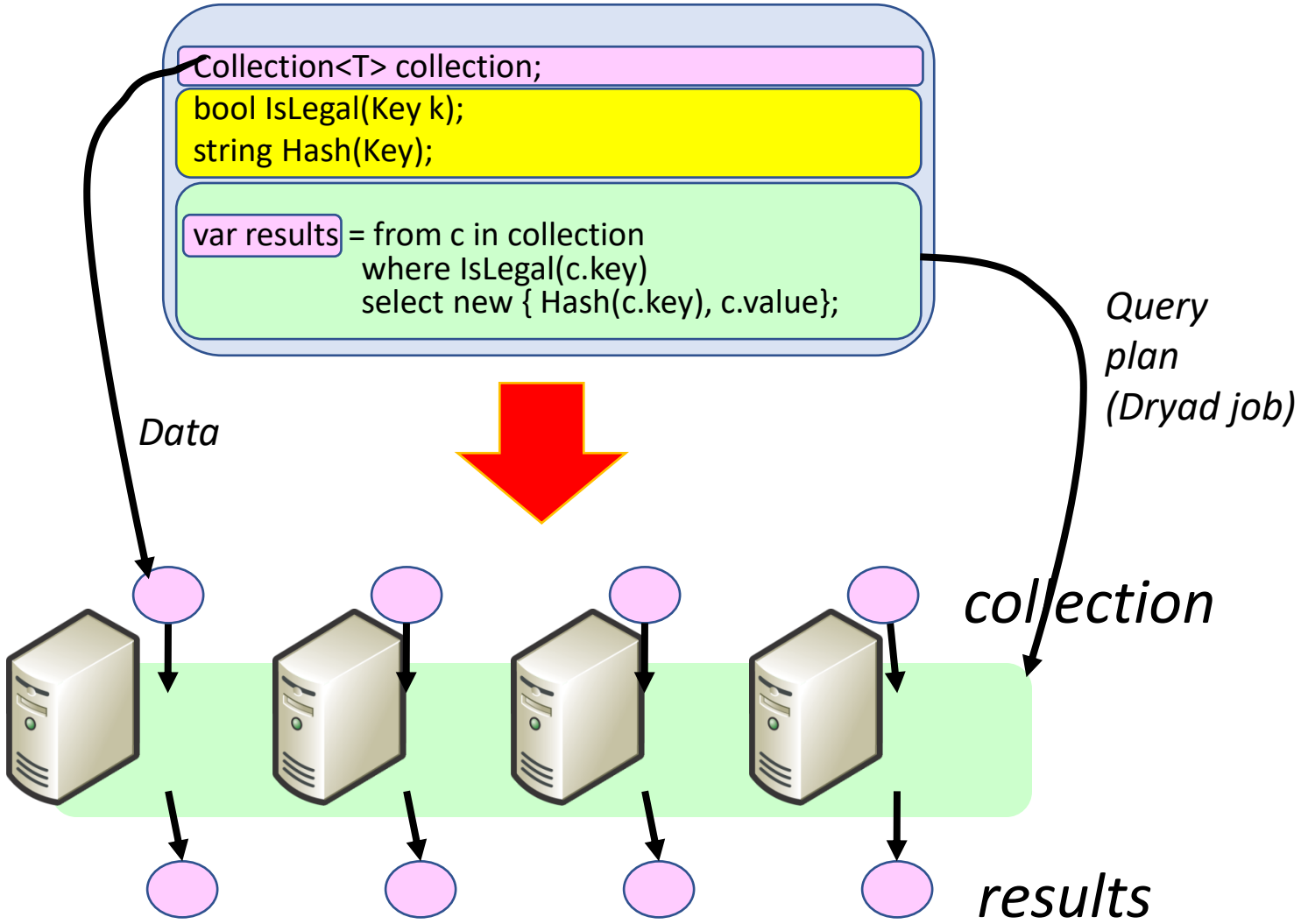
```
Collection<T> collection;
bool IsLegal(Key k);
string Hash(Key);

var results = from c in collection
              where IsLegal(c.key)
              select new { Hash(c.key), c.value};
```



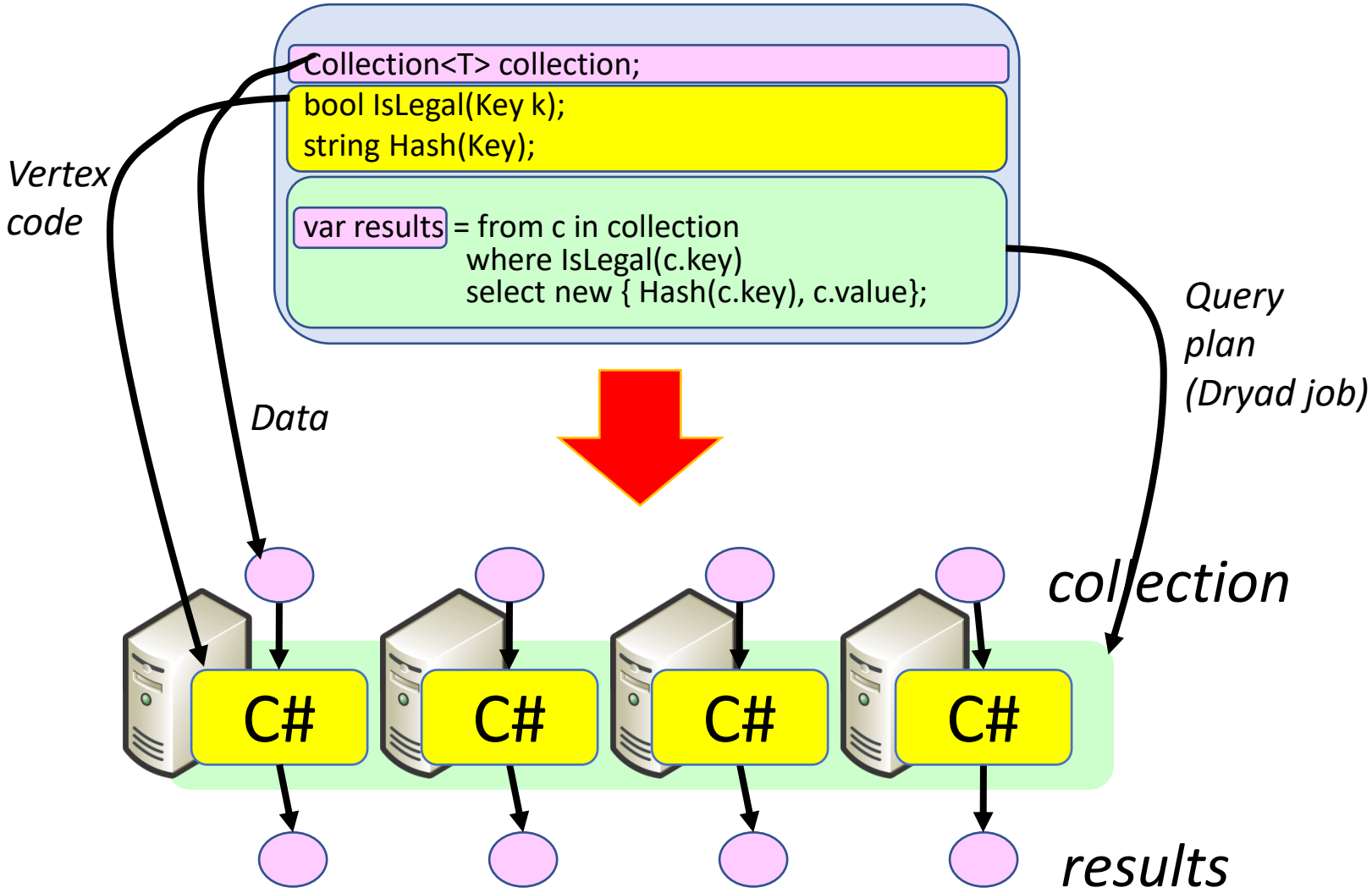
DryadLINQ = LINQ + Dryad



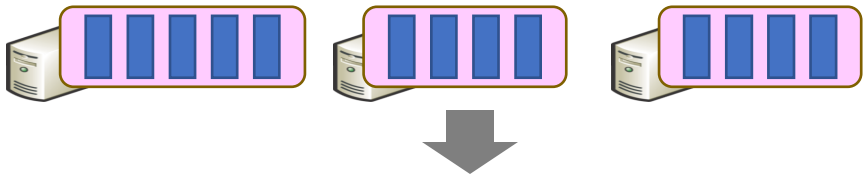
DryadLINQ = LINQ + Dryad



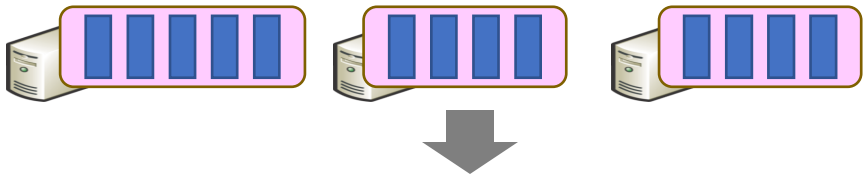
DryadLINQ = LINQ + Dryad



Language Summary



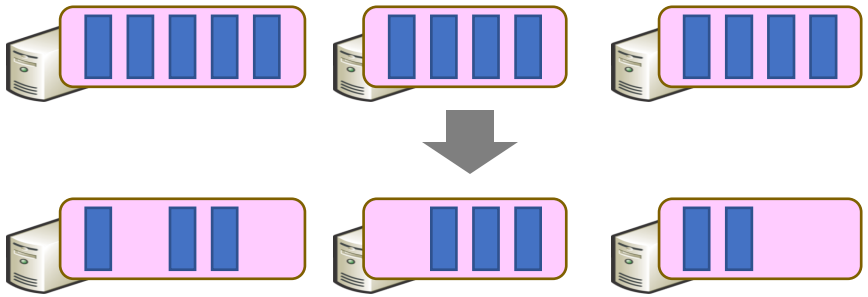
Language Summary



Where

Language Summary

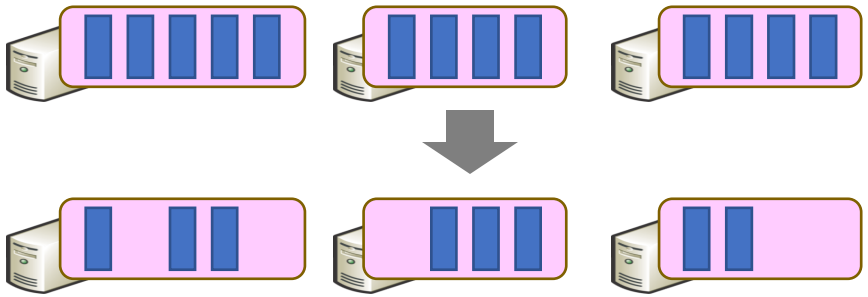
Where



Language Summary

Where

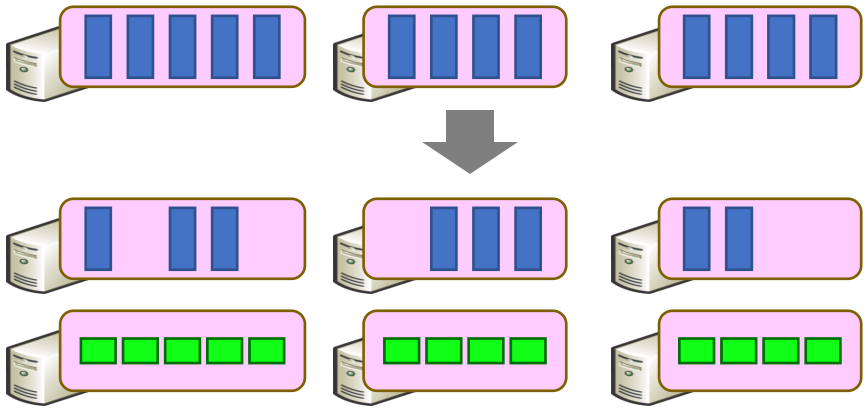
Select



Language Summary

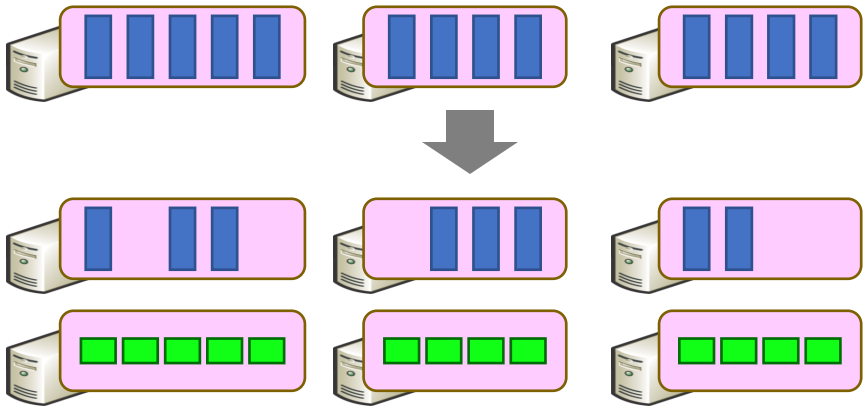
Where

Select



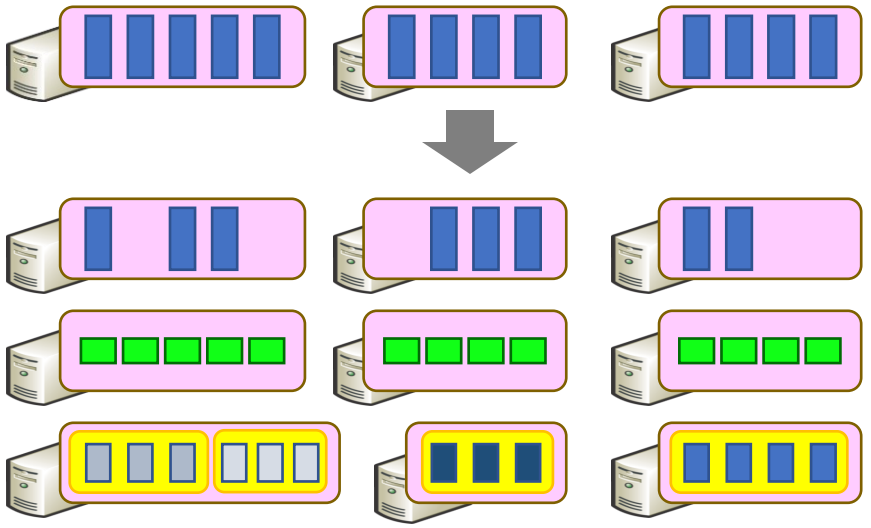
Language Summary

Where
Select
GroupBy



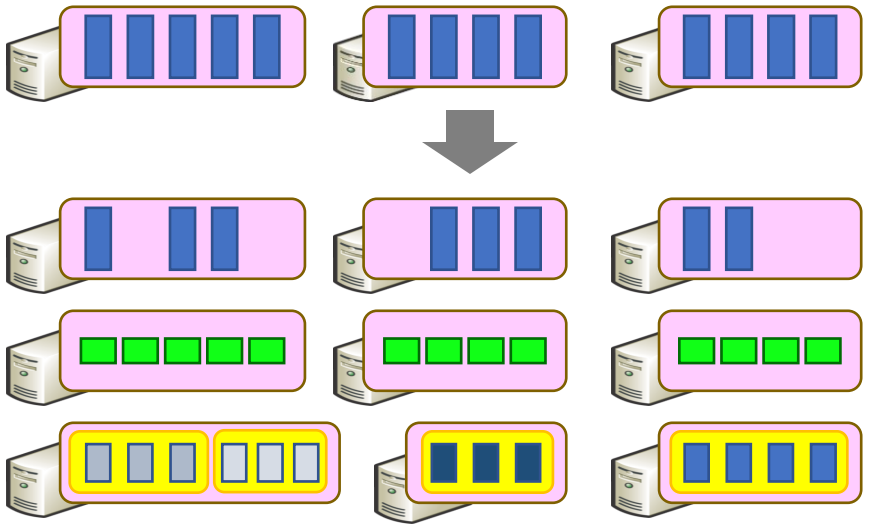
Language Summary

Where
Select
GroupBy



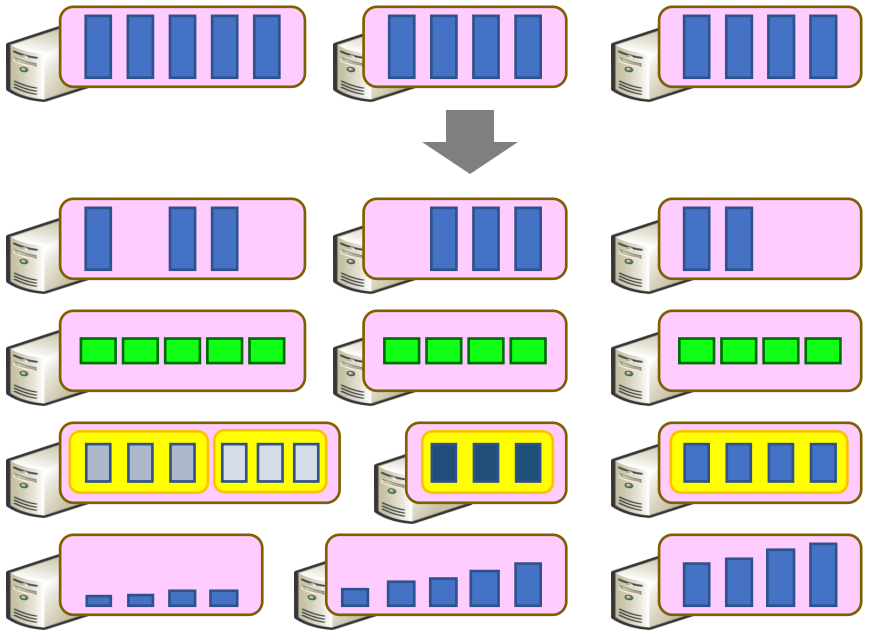
Language Summary

Where
Select
GroupBy
OrderBy



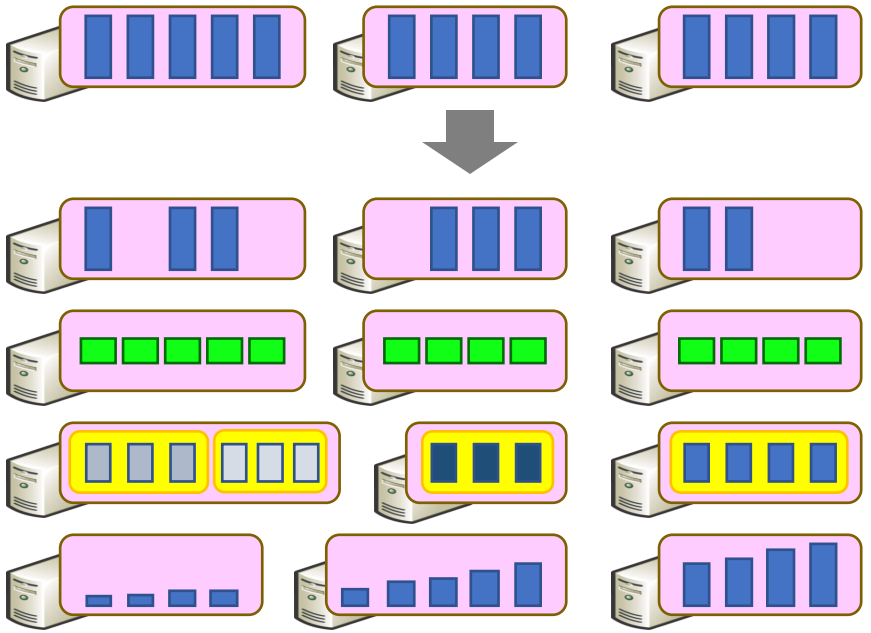
Language Summary

Where
Select
GroupBy
OrderBy



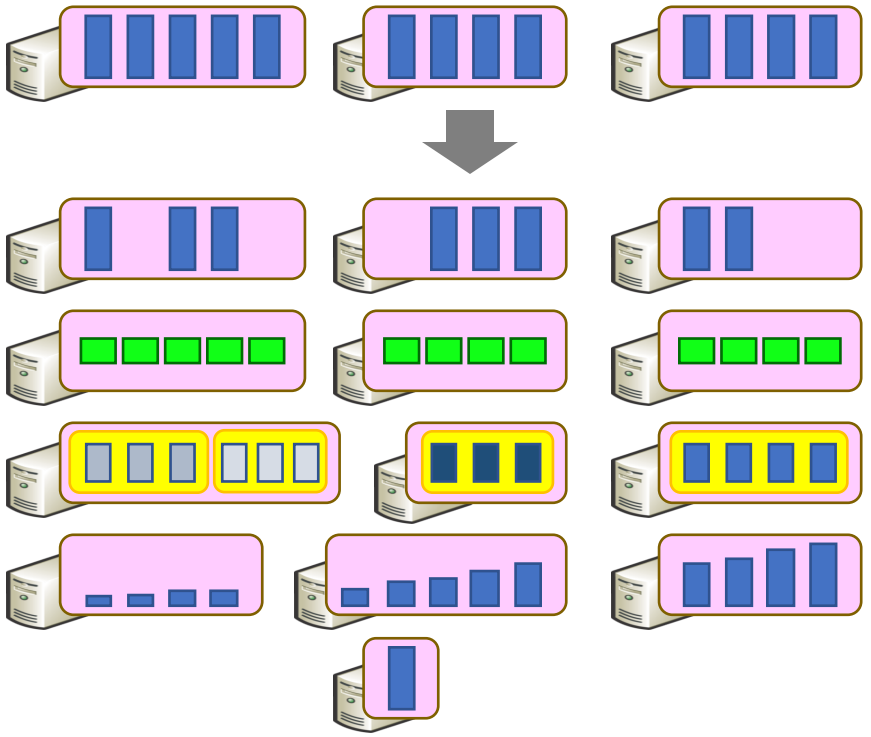
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate



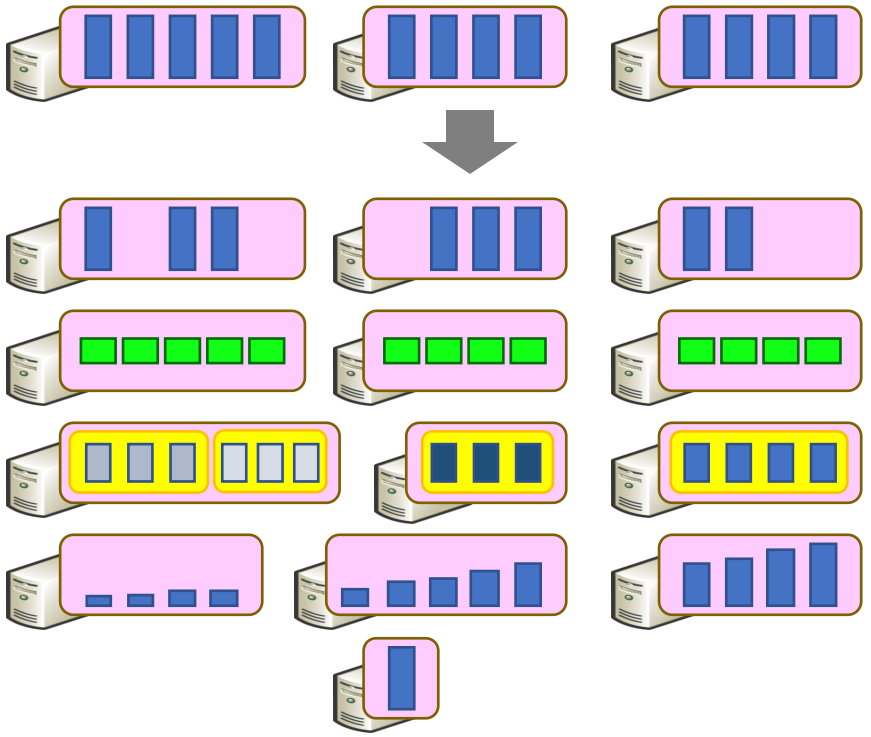
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate



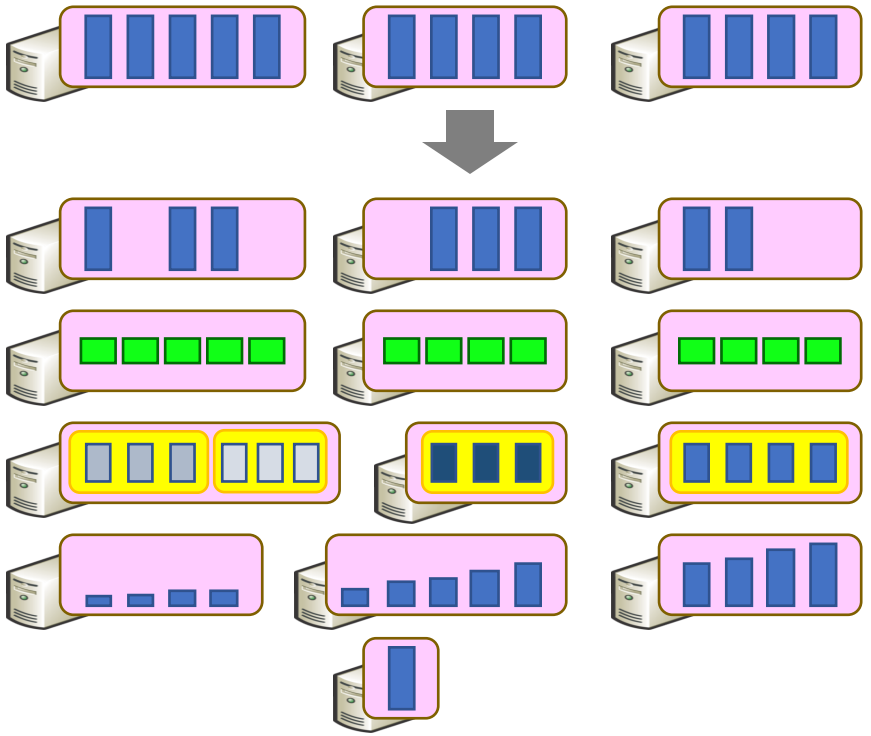
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Join



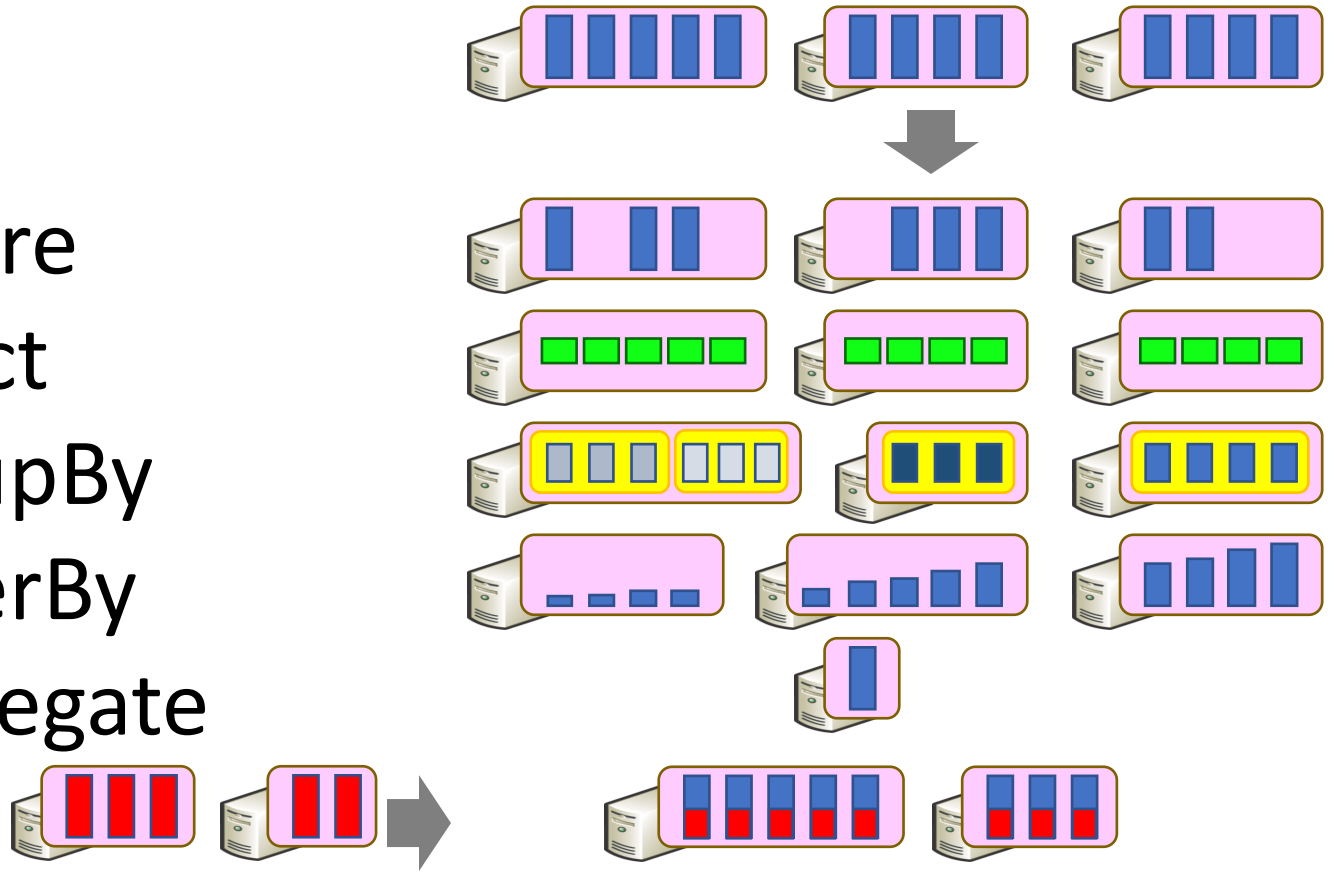
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Join



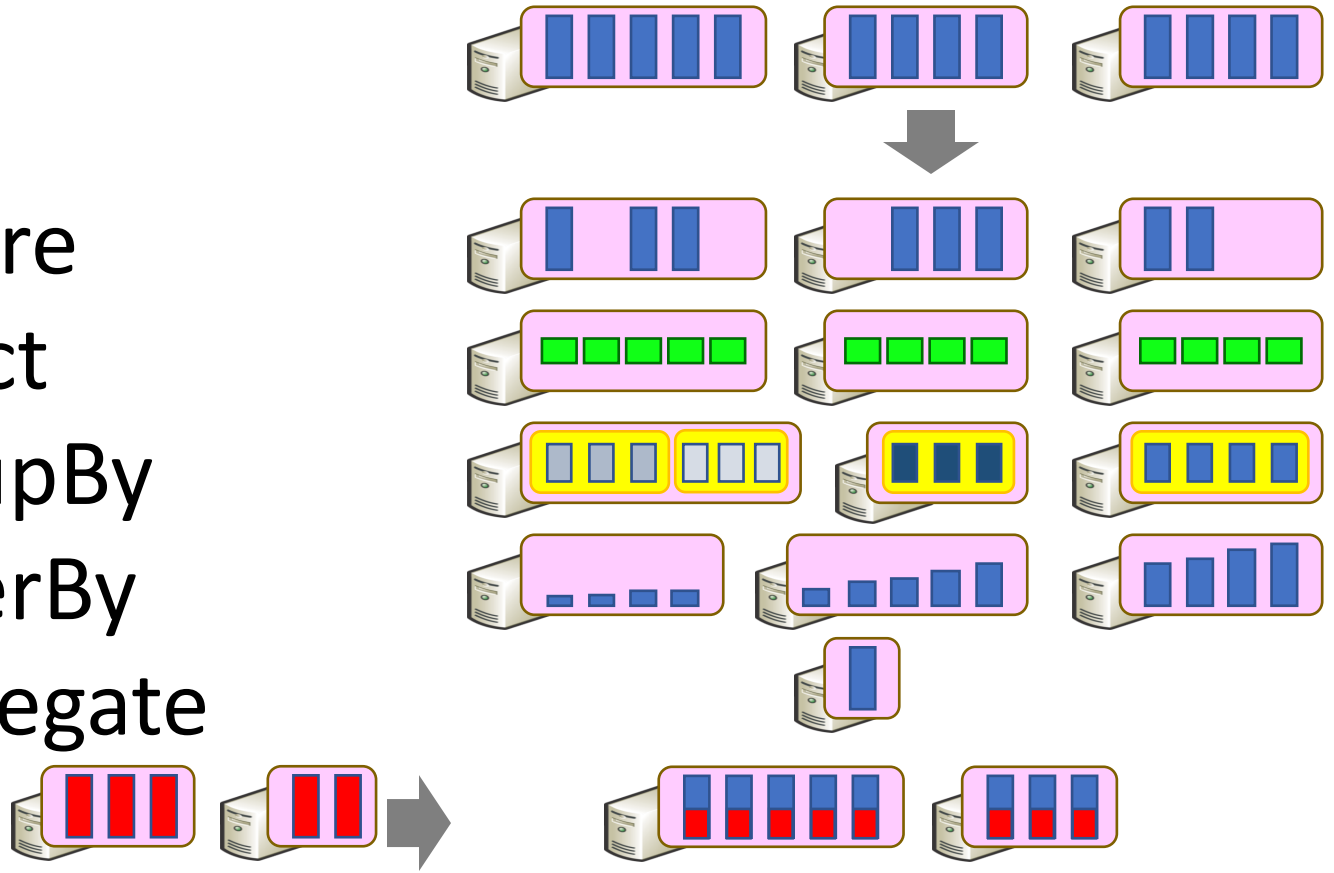
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Join



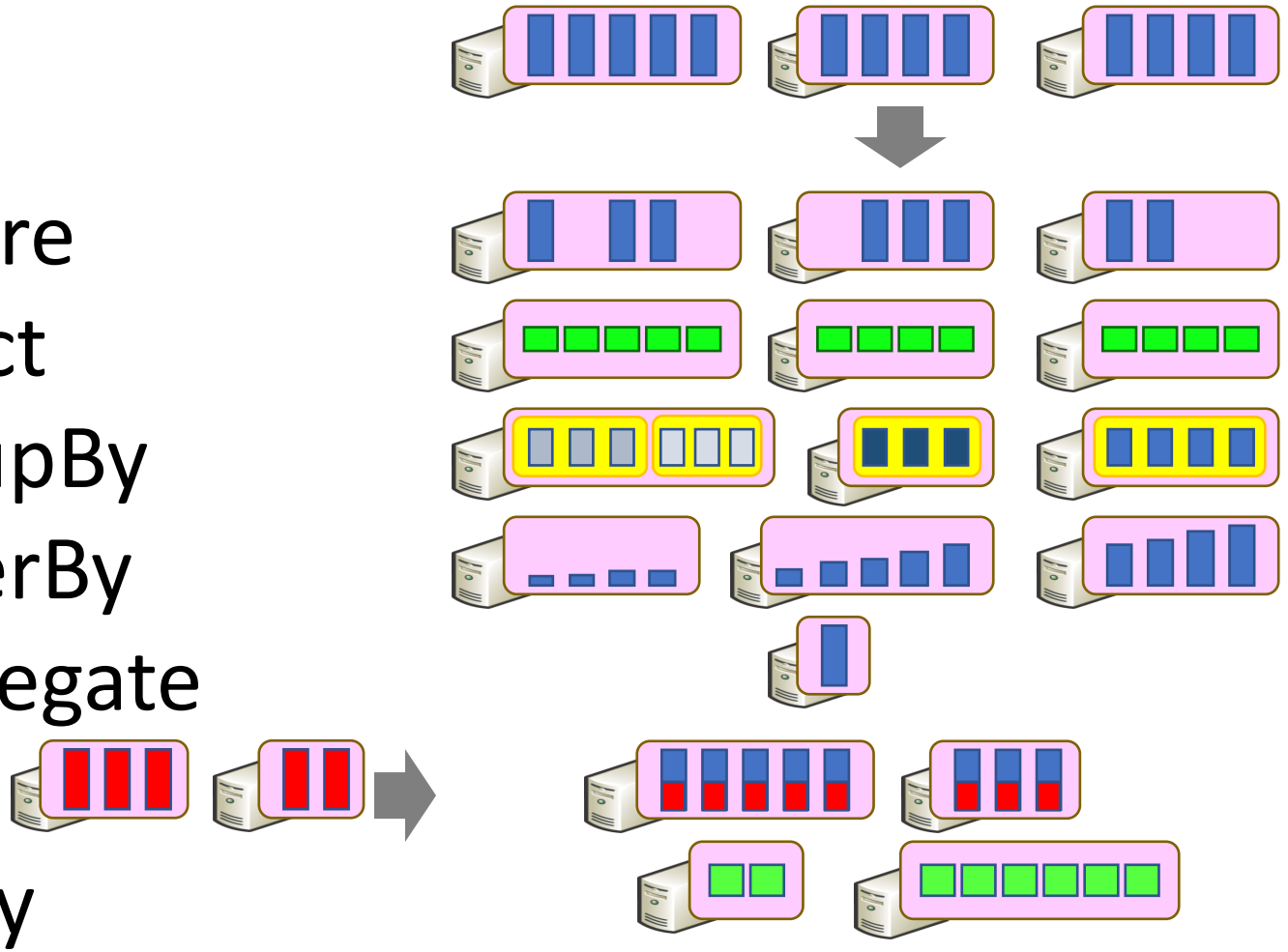
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Join
Apply



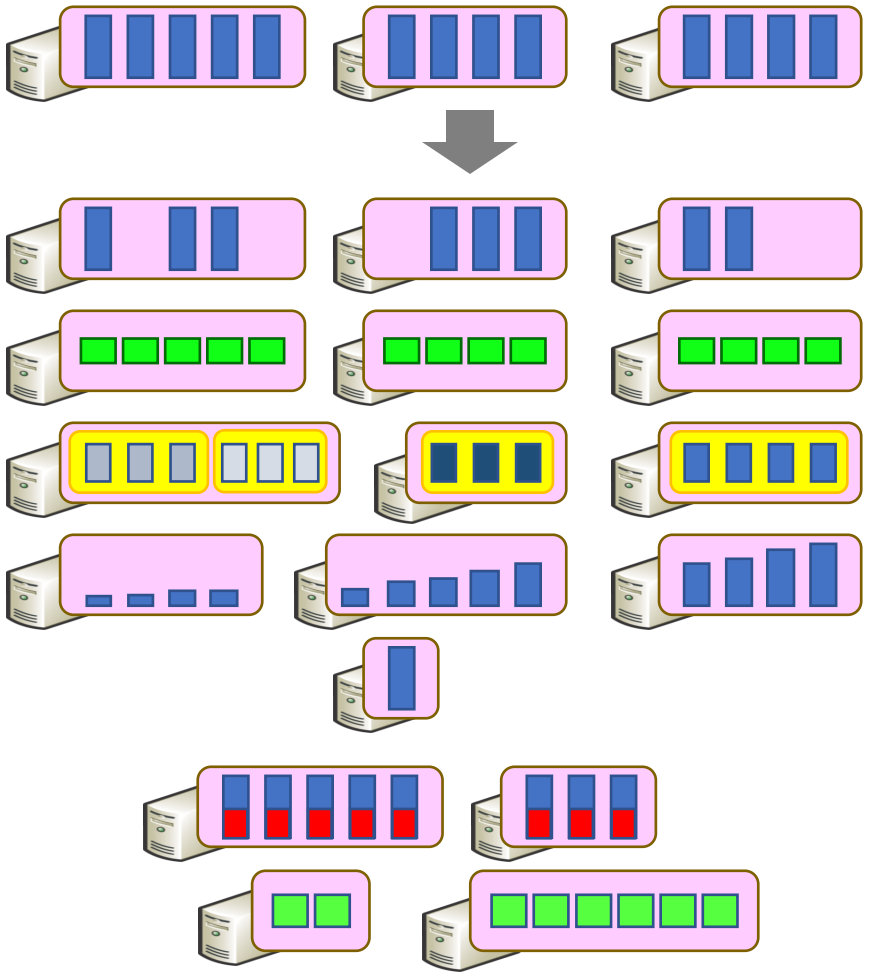
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Join
Apply



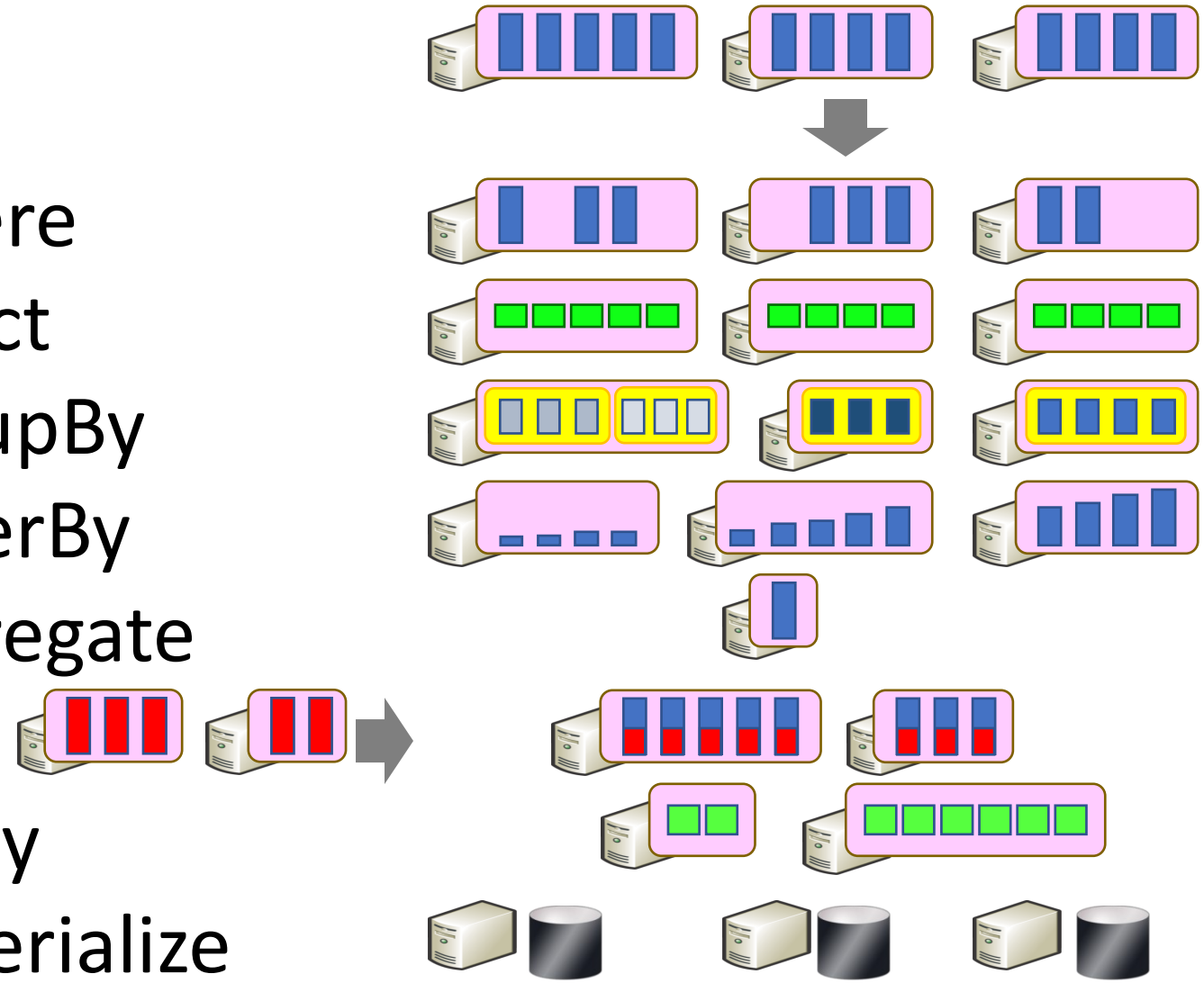
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Join
Apply
Materialize



Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Join
Apply
Materialize



Example: Histogram

```
public static IQueryable<Pair> Histogram(
      IQueryable<LineRecord> input, int k)
{
   var words = input.SelectMany(x => x.line.Split(' '));
   var groups = words.GroupBy(x => x);
   var counts = groups.Select(x => new Pair(x.Key, x.Count()));
   var ordered = counts.OrderByDescending(x => x.count);
   var top = ordered.Take(k);
   return top;
}
```

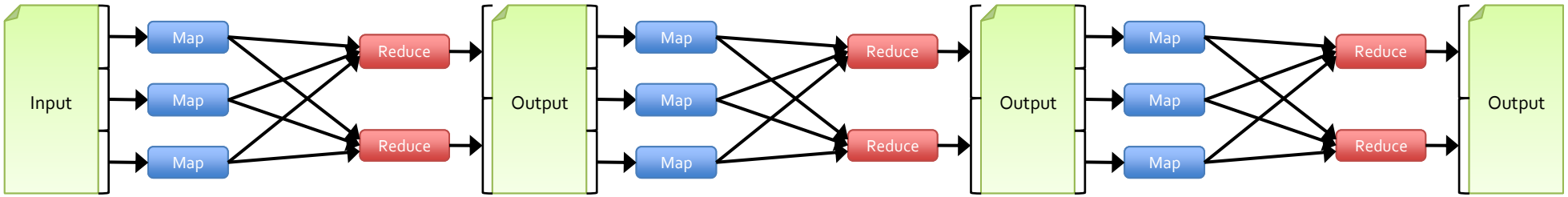
“A line of words of wisdom”
[“A”, “line”, “of”, “words”, “of”, “wisdom”]
[[“A”], [“line”], [“of”, “of”], [“words”], [“wisdom”]]
[{“A”, 1}, {“line”, 1}, {“of”, 2}, {“words”, 1}, {“wisdom”, 1}]
[{“of”, 2}, {“A”, 1}, {“line”, 1}, {“words”, 1}, {“wisdom”, 1}]
[{“of”, 2}, {“A”, 1}, {“line”, 1}]

Iterative Computations: PageRank

1. Start each page with a rank of 1
2. On each iteration, update each page's rank to

$$\Sigma_{i \in neighbors} rank_i / |neighbors_i|$$

```
links = // RDD of (url, neighbors) pairs
ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  ranks = links.join(ranks).flatMap {
    (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }.reduceByKey(_ + _)
}
```

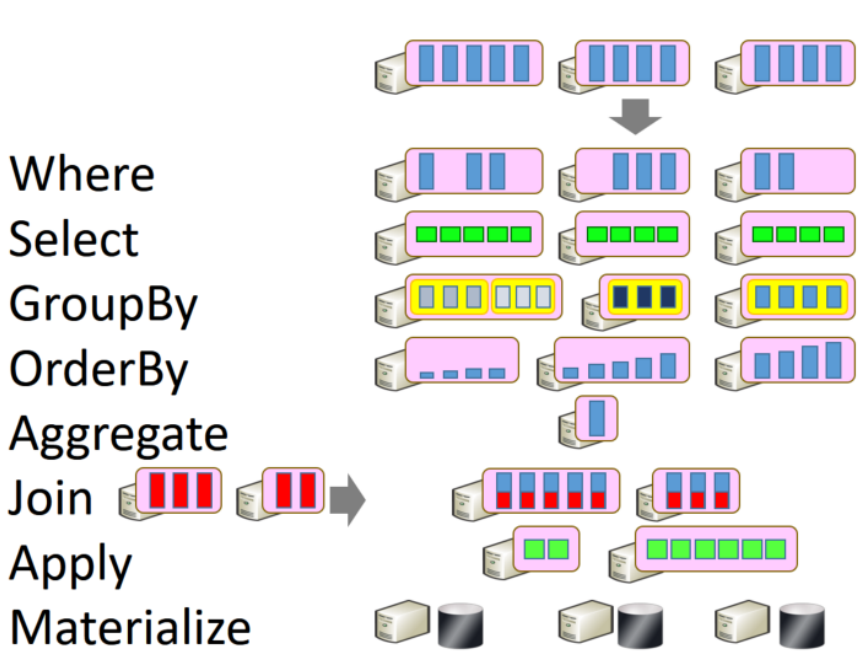


RDD Operations

Transformations (define a new RDD)	**Parallel operations** (return a result to driver)
map filter sample union groupByKey reduceByKey join persist/*cache* …	reduce collect count save lookupKey …

RDD Operations

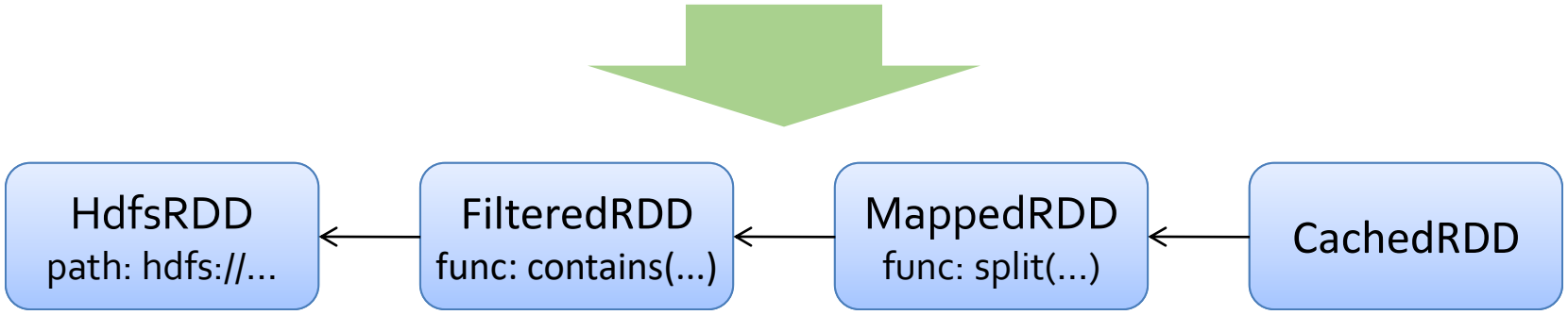
Transformations (define a new RDD)	**Parallel operations** (return a result to driver)
map filter sample union groupByKey reduceByKey join persist/*cache* …	reduce collect count save lookupKey …



RDD Fault Tolerance

- RDDs maintain *lineage* information that can be used to reconstruct lost partitions
- Ex:

```
cachedMsgs = textFile(...).filter(_.contains("error"))
                          .map(_.split('\t')(2))
                          .persist()
```



RDDs vs Distributed Shared Memory

Concern	RDDs	Distr. Shared Mem.
Reads	Fine-grained	Fine-grained
Writes	Bulk transformations	Fine-grained
Consistency	Trivial (immutable)	Up to app / runtime
Fault recovery	Fine-grained and low-overhead using lineage	Requires checkpoints and program rollback
Straggler mitigation	Possible using speculative execution	Difficult
Work placement	Automatic based on data locality	Up to app (but runtime aims for transparency)