Programming at Scale: Dataflow and Consistency

cs378h

Today

Questions?

Administrivia

- Rust lab due today!
- Project Proposal Due Thursday!

Agenda:

- Dataflow Wrapup
- Concurrency & Consistency at Scale

Review: K-Means

```
public void kmeans() {
  while(…) {
    for each point
      find_nearest_center(point);
    for each center
      compute_new_center(center)
  }
}
```

Review: K-Means

map (bracket around the "for each point" loop)

```
public void kmeans() {
  while(...) {
    for each point
      find_nearest_center(point);
    for each center
      compute_new_center(center)
  }
}
```

Review: K-Means

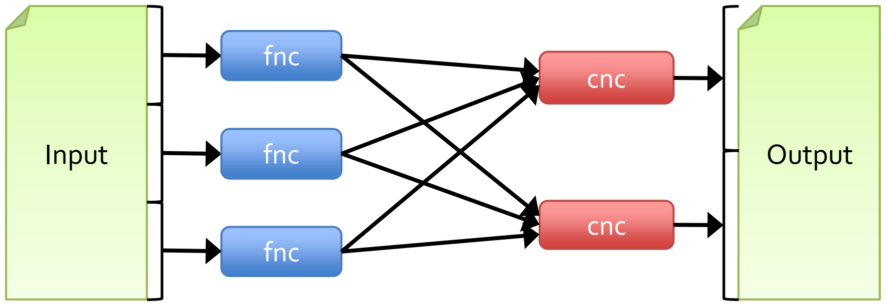
```
public void kmeans() {
   while(…) {
map  {  for each point
           find_nearest_center(point);
reduce { for each center
           compute_new_center(center)
   }
}
```

Review: K-Means

```
public void kmeans() {
  while(…) {
    for each point
      find_nearest_center(point);
    for each center
      compute_new_center(center)
  }
}
```

map: for each point / find_nearest_center(point);

reduce: for each center / compute_new_center(center)

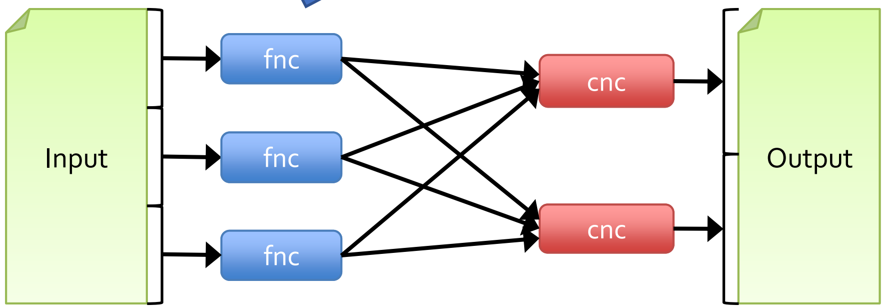


Review: K-Means

```
public void kmeans() {
  while(…) {
    for each point
      find_nearest_center(point);
    for each center
      compute_new_center(center)
  }
}
```

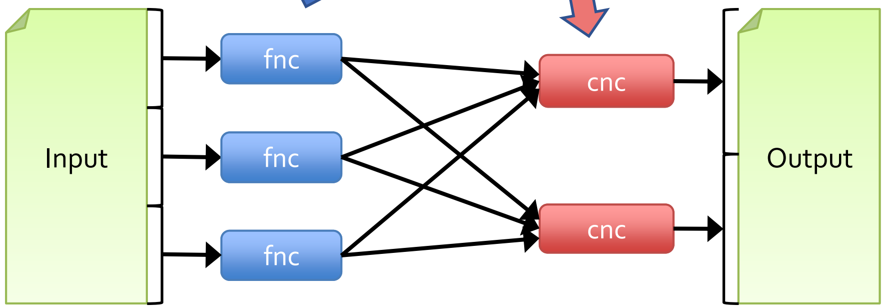
map

reduce



Review: K-Means

```
public void kmeans() {
  while(...) {
    for each point            <- map
      find_nearest_center(point);
    for each center           <- reduce
      compute_new_center(center)
  }
}
```

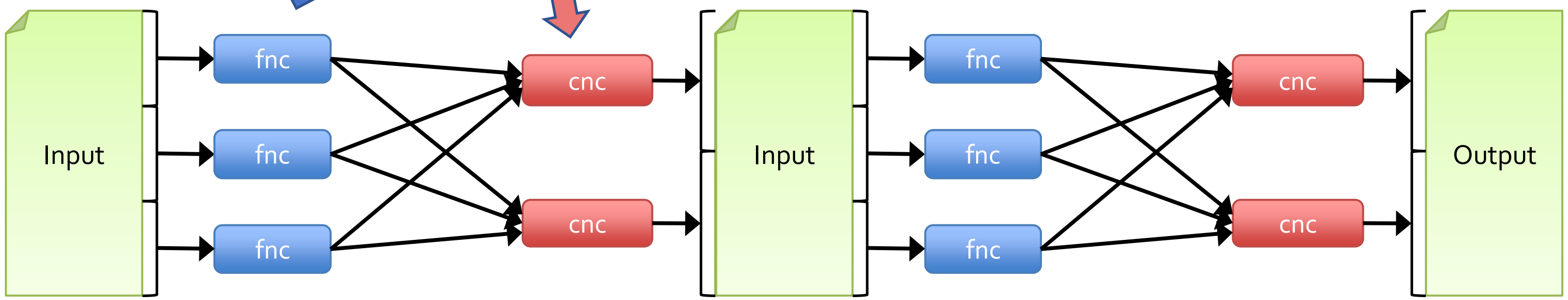


Review: K-Means

map: for each point find_nearest_center(point);

reduce: for each center compute_new_center(center)

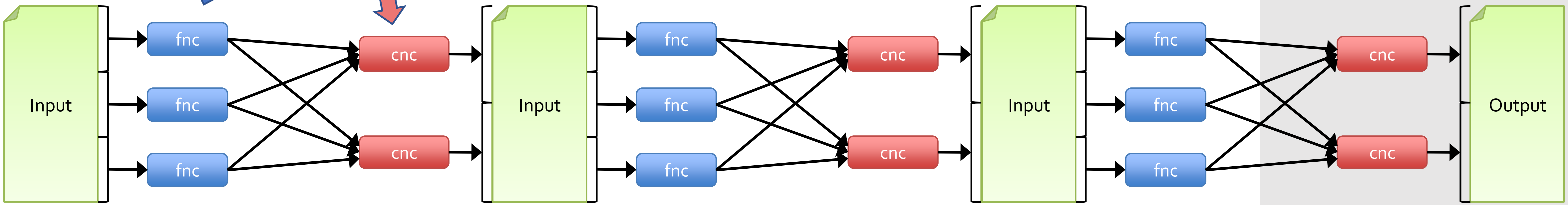
```
public void kmeans() {
   while(…) {
      for each point
         find_nearest_center(point);
      for each center
         compute_new_center(center)
   }
}
```



Review: K-Means

map / reduce

```
public void kmeans() {
  while(…) {
    for each point
      find_nearest_center(point);
    for each center
      compute_new_center(center)
  }
}
```

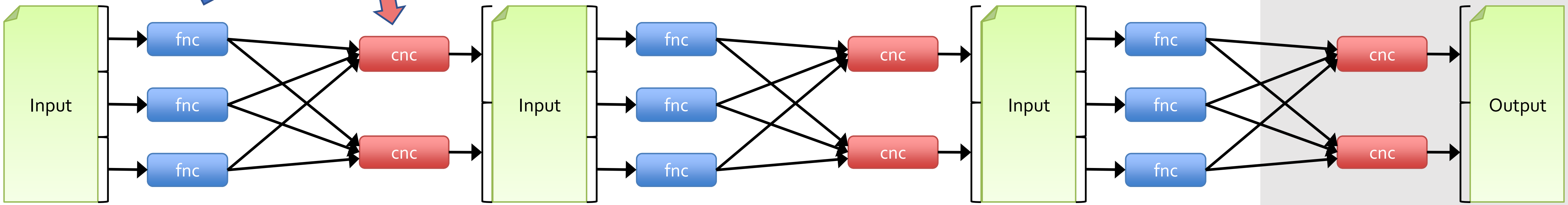


Review: K-Means

```
public void kmeans() {
   while(…) {
map     for each point
          find_nearest_center(point);
reduce  for each center
          compute_new_center(center)
   }
}
```

```
/*
 * Map: find minimum distance center for point, emit to reducer
 */
@Override
public void map(LongWritable key, Text value,
                OutputCollector<DoubleWritable, DoubleWritable> output,
                Reporter reporter) throws IOException {
    String line = value.toString();
    double point = Double.parseDouble(line);
    double min1, min2 = Double.MAX_VALUE, nearest_center = mCenters.get(0);
    // Find the minimum center from a point
    for (double c : mCenters) {
        min1 = c - point;
        if (Math.abs(min1) < Math.abs
            nearest_center = c;
            min2 = min1;
        }
    }
    // Emit the nearest center and th
    output.collect(new DoubleWritable
            new DoubleWritable(point)
}
```

```
/*
 * Reduce: collect all points per center and calculate
 * the next center for those points
 */
@Override
public void reduce(
        DoubleWritable key, Iterator<DoubleWritable> values,
        OutputCollector<DoubleWritable, Text> output, Reporter reporter)
        throws IOException {
    double newCenter;
    double sum = 0;
    int no_elements = 0;
    String points = "";
    while (values.hasNext()) {
        double d = values.next().get();
        points = points + " " + Double.toString(d);
        sum = sum + d;
        ++no_elements;
    }

    // We have a new center now
    newCenter = sum / no_elements;

    // Emit new center and point
    output.collect(new DoubleWritable(newCenter), new Text(points));
}
```



Review: K-Means

```
public void kmeans() {
  while(…) {
    for each point                          (map)
      find_nearest_center(point)
    for each center                         (reduce)
      compute_new_center(center)
  }
}
```

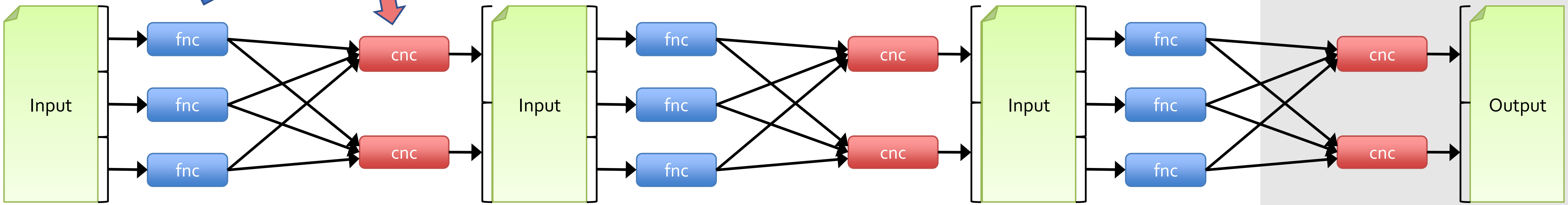
```
/*
 * Map: find minimum distance center for point, emit to reducer
 */
@Override
public void map(LongWritable key, Text value,
                OutputCollector<DoubleWritable, DoubleWritable> output,
                Reporter reporter) throws IOException {
    String line = value.toString();
    double point = Double.parseDouble(line);
    double min1, min2 = Double.MAX_VALUE, nearest_center = mCenters.get(0);
    // Find the minimum center from a point
    for (double c : mCenters) {
        min1 = c - point;
        if (Math.abs(min1) < Math.abs
            nearest_center = c;
            min2 = min1;
}
```

```
/*
 * Reduce: collect all points per center and calculate
 * the next center for those points
 */
...
oubleWritable> values,
, Text> output, Reporter reporter)
...
    while (values.hasNext()) {
        double d = values.next().get();
        points = points + " " + Double.toString(d);
        sum = sum + d;
        ++no_elements;
    }

    // We have a new center now
    newCenter = sum / no_elements;

    // Emit new center and point
    output.collect(new DoubleWritable(newCenter), new Text(points));
}
```

Key idea: ***adapt workload to parallel patterns***

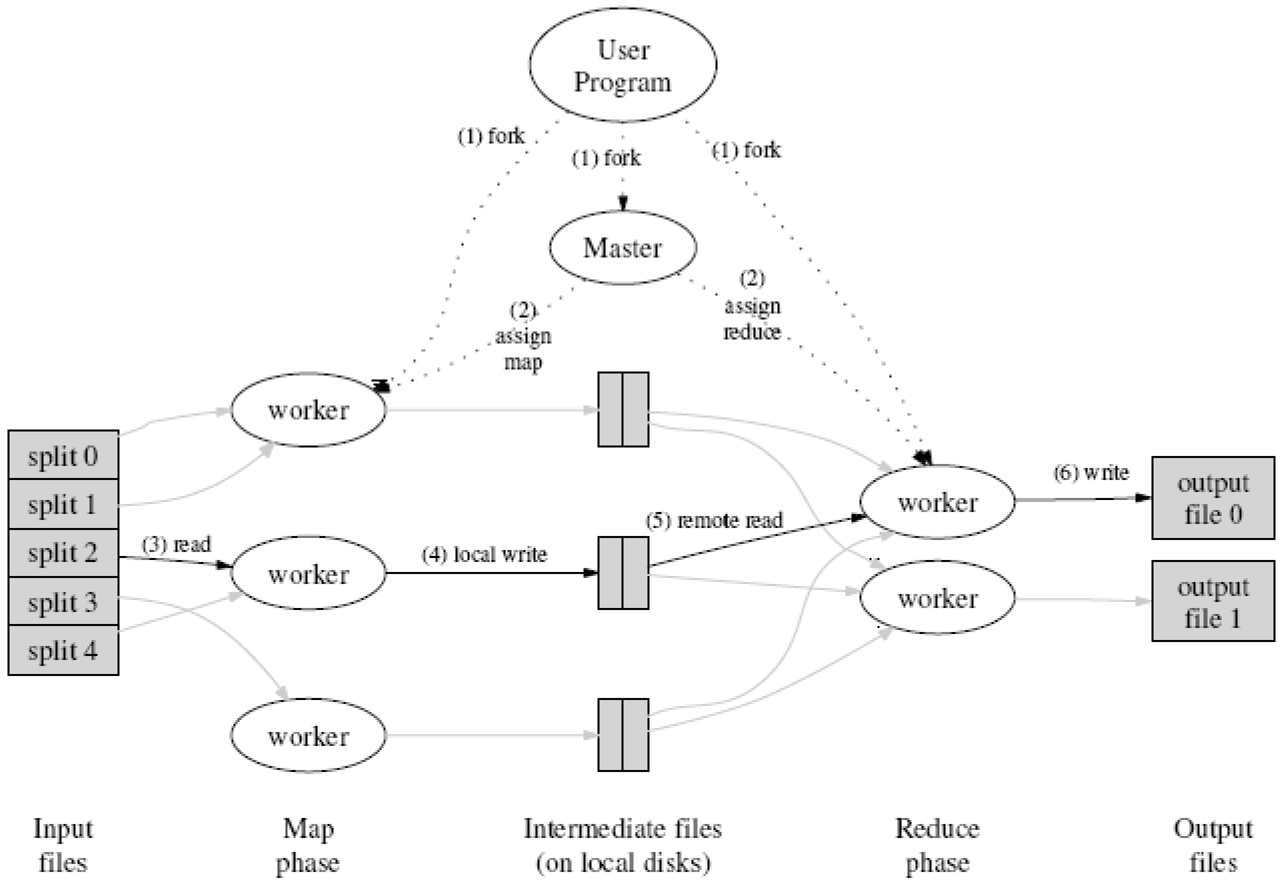
Questions:

- What kinds of computations can this express?
- What other patterns could we use?

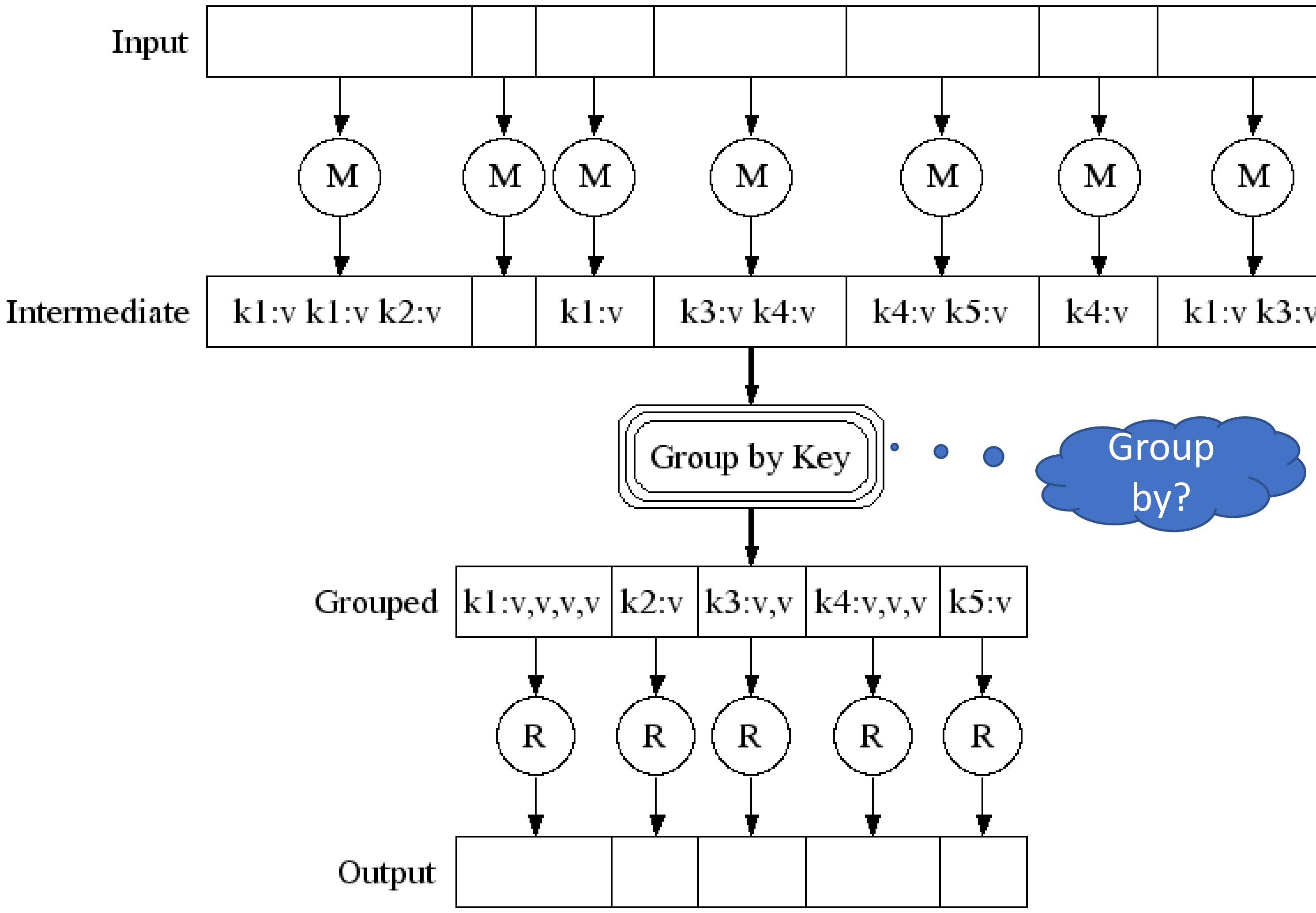


How Does Parallelization Work?

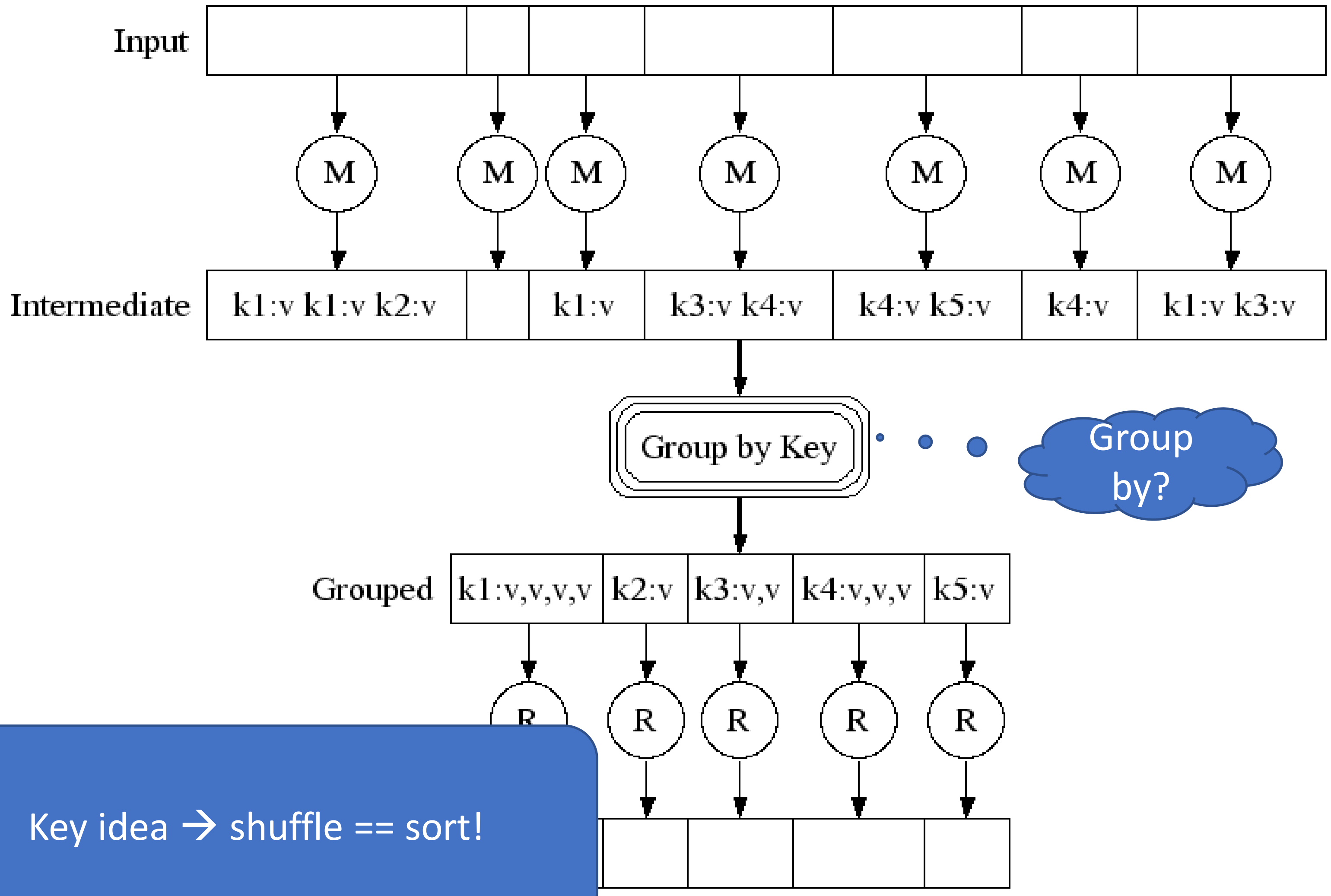
INPUT FILE(s)



Execution


Input
M
Intermediate
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Group by Key
Group by?
Grouped
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
R
Output

Execution



Key idea → shuffle == sort!

Task Granularity And Pipelining

|map tasks| >> |machines| -- why?

Task Granularity And Pipelining

|map tasks| >> |machines| -- why?

Minimize fault recovery time

Pipeline map with other tasks

Easier to load balance dynamically

MapReduce: not without Controversy

MapReduce: A major step backwards | The Database Column

http://databasecolumn.vertica.com/database-innovation/mapreduce-a-major-step-backwards/ September 6, 2011

on Jan 17 in Database architecture, Database history, Database innovation posted by DeWitt

[Note: Although the system attributes this post to a single author, it was written by David J. DeWitt and Michael Stonebraker]

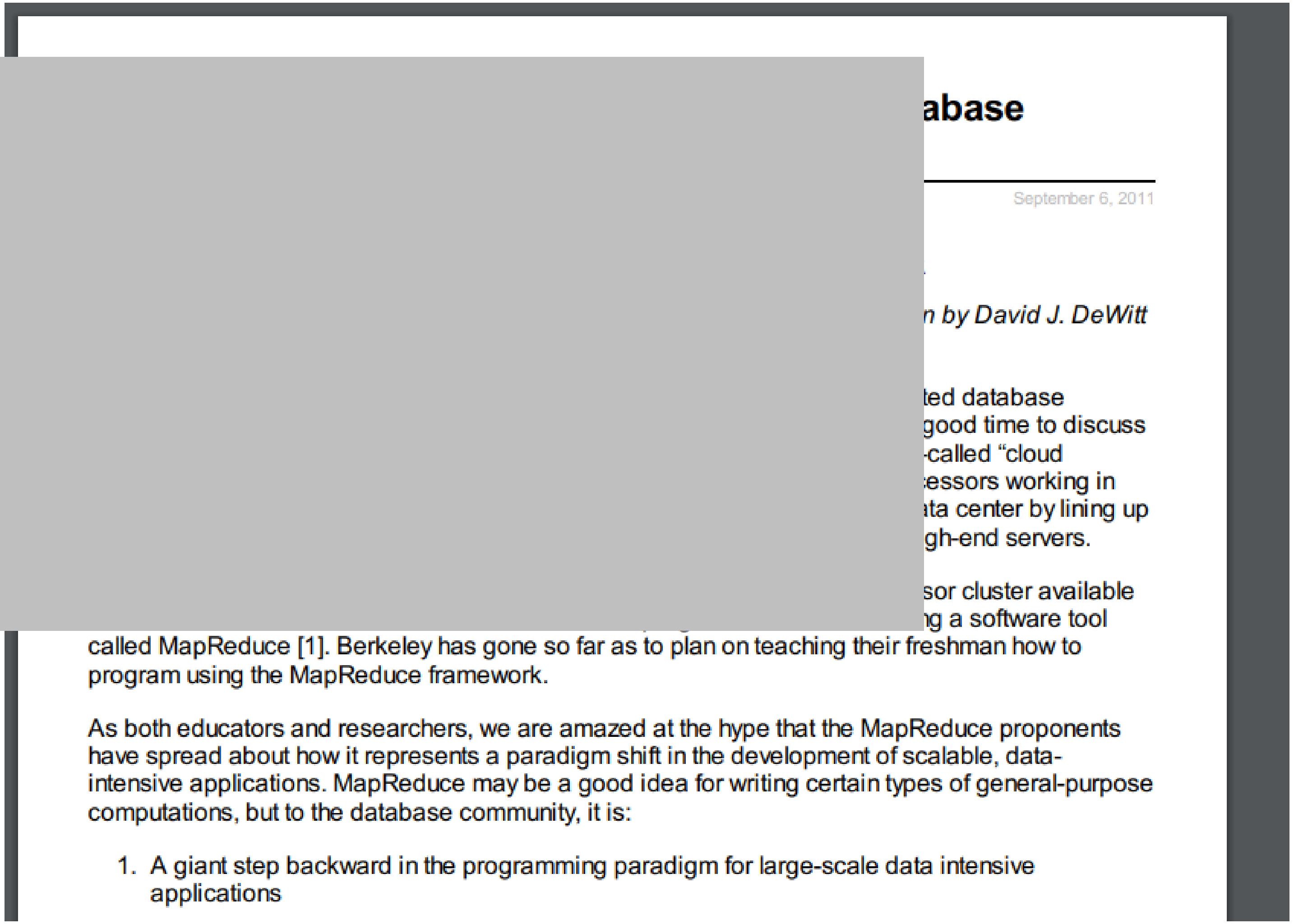
On January 8, a Database Column reader asked for our views on new distributed database research efforts, and we'll begin here with our views on MapReduce. This is a good time to discuss it, since the recent trade press has been filled with news of the revolution of so-called "cloud computing." This paradigm entails harnessing large numbers of (low-end) processors working in parallel to solve a computing problem. In effect, this suggests constructing a data center by lining up a large number of "jelly beans" rather than utilizing a much smaller number of high-end servers.

For example, IBM and Google have announced plans to make a 1,000 processor cluster available to a few select universities to teach students how to program such clusters using a software tool called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications

MapReduce: not without Controversy



abase

September 6, 2011

n by David J. DeWitt

ted database
good time to discuss
-called "cloud
essors working in
ata center by lining up
gh-end servers.

sor cluster available
ng a software tool
called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications

MapReduce: not without Controversy

abase

September 6, 2011

n by David J. DeWitt

ted database
good time to discuss
-called "cloud
essors working in
ta center by lining up
gh-end servers.

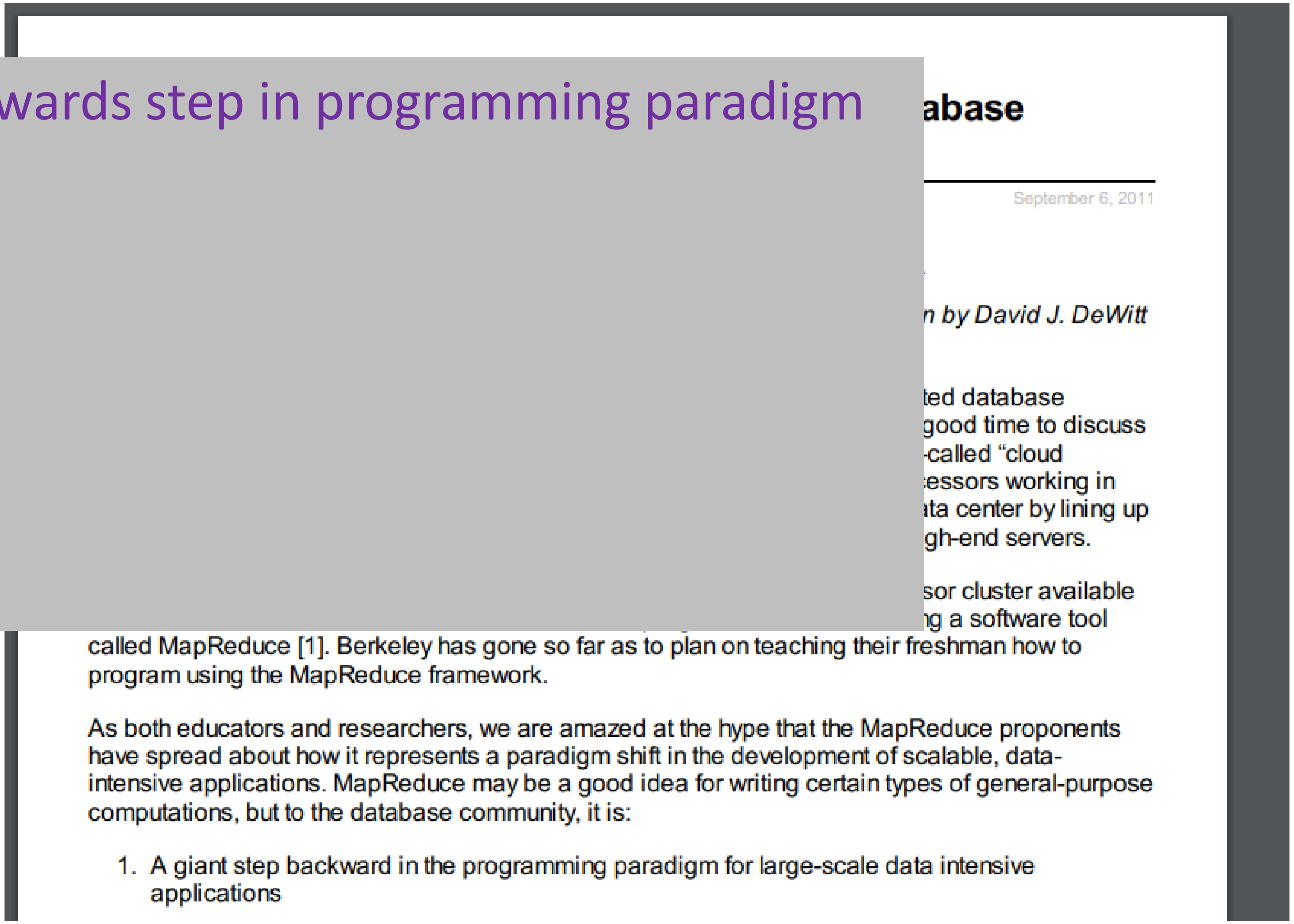
sor cluster available
ng a software tool
called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications

MapReduce: not without Controversy

Backwards step in programming paradigm



abase

September 6, 2011

n by David J. DeWitt

ted database
good time to discuss
-called "cloud
essors working in
ata center by lining up
gh-end servers.

sor cluster available
ng a software tool
called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications

MapReduce: not without Controversy

Backwards step in programming paradigm

Sub-optimal: brute force, no indexing

abase

September 6, 2011

n by David J. DeWitt

ted database
good time to discuss
-called "cloud
essors working in
ta center by lining up
gh-end servers.

sor cluster available
g a software tool
called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications

MapReduce: not without Controversy

Backwards step in programming paradigm

Sub-optimal: brute force, no indexing

Not novel: 25 year-old ideas from DBMS lit

It's just a group-by aggregate engine

abase

September 6, 2011

n by David J. DeWitt

ted database
good time to discuss
-called "cloud
essors working in
ta center by lining up
gh-end servers.

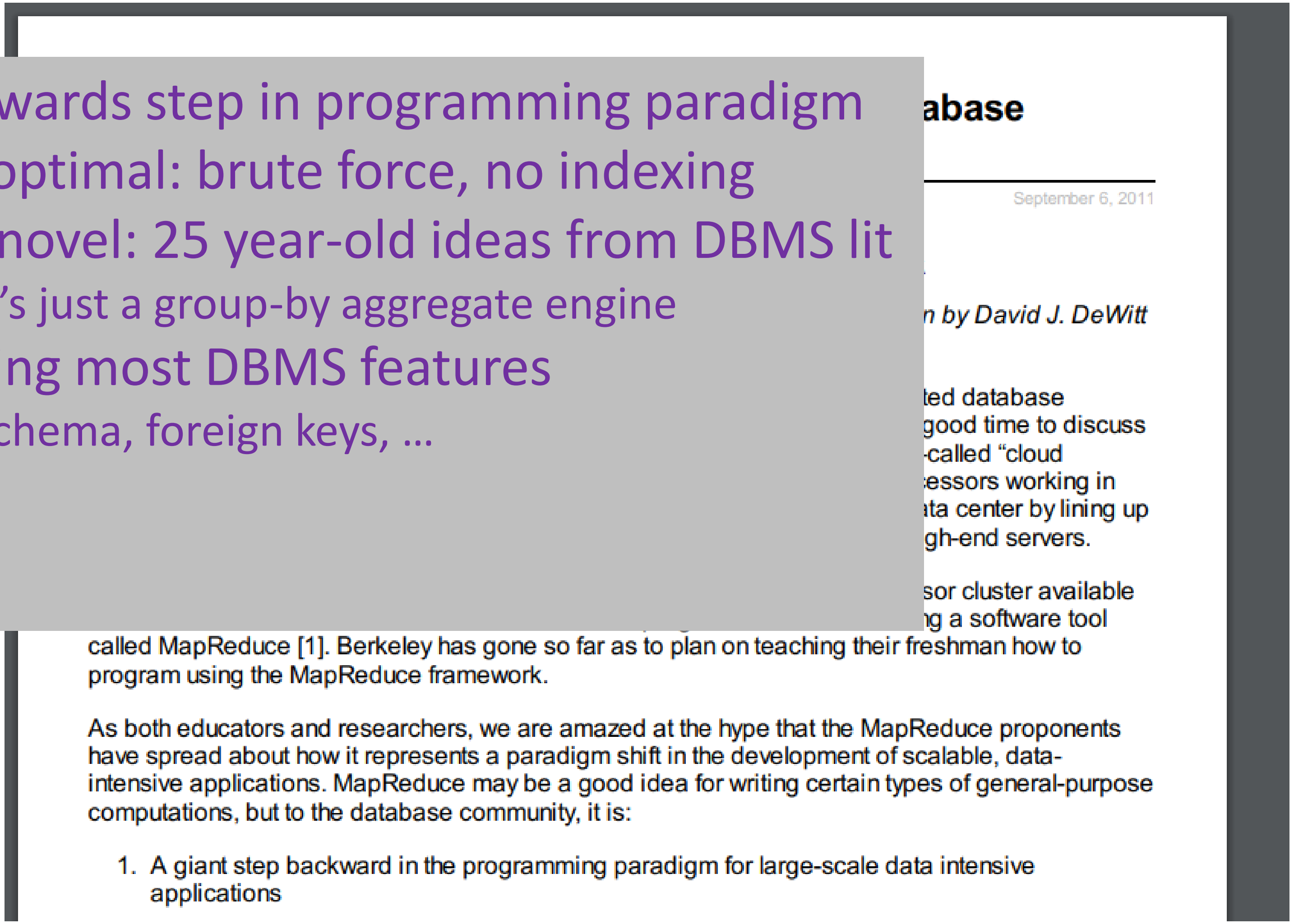
sor cluster available
g a software tool
called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications

MapReduce: not without Controversy

- Backwards step in programming paradigm
- Sub-optimal: brute force, no indexing
- Not novel: 25 year-old ideas from DBMS lit
    - It's just a group-by aggregate engine
- Missing most DBMS features
    - Schema, foreign keys, …


abase

September 6, 2011

n by David J. DeWitt

ted database
good time to discuss
-called "cloud
essors working in
ata center by lining up
gh-end servers.

sor cluster available
ng a software tool
called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications

MapReduce: not without Controversy

Backwards step in programming paradigm

Sub-optimal: brute force, no indexing

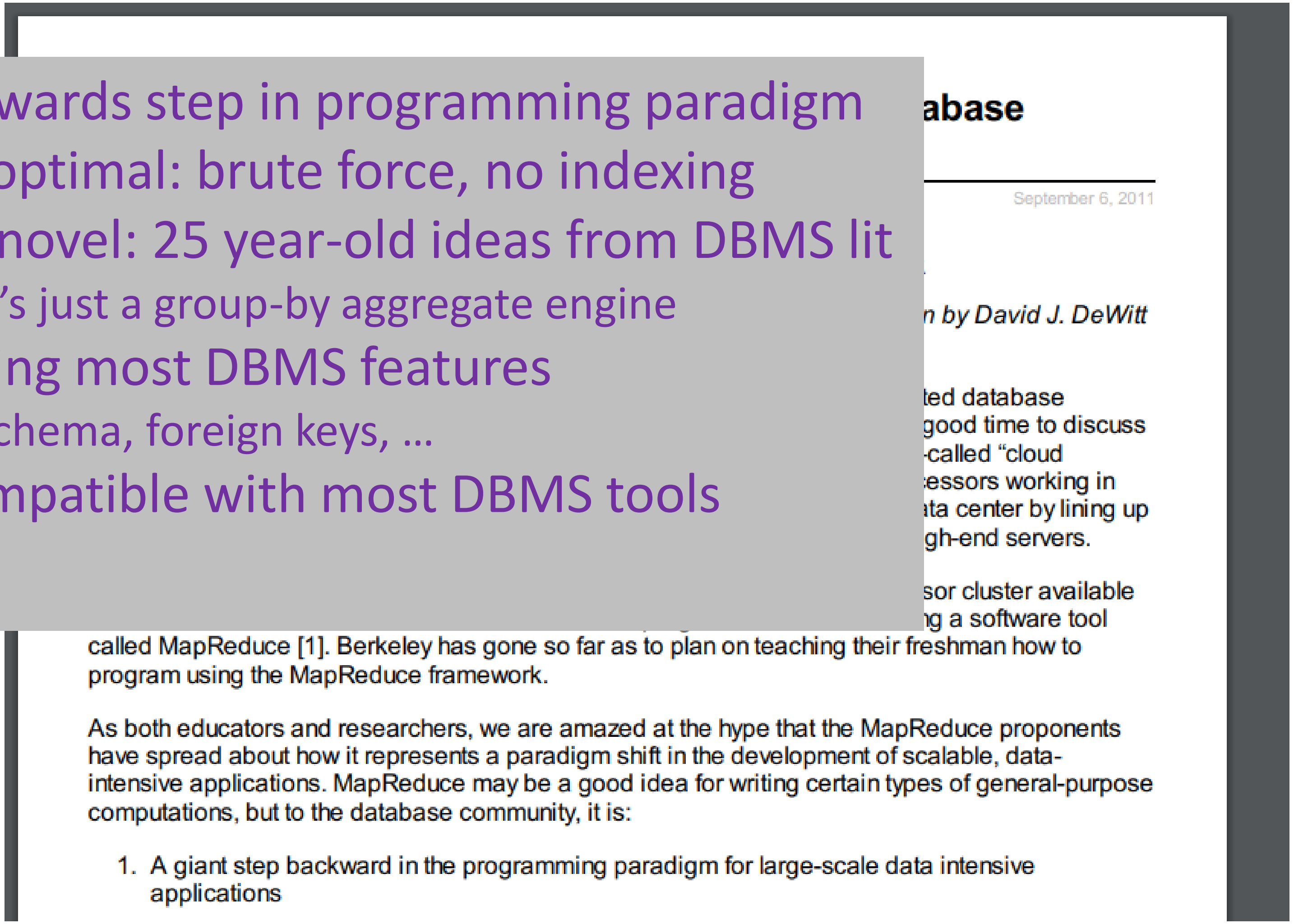
Not novel: 25 year-old ideas from DBMS lit

- It's just a group-by aggregate engine

Missing most DBMS features

- Schema, foreign keys, …

Incompatible with most DBMS tools



abase

September 6, 2011

n by David J. DeWitt

ted database
good time to discuss
-called "cloud
essors working in
ta center by lining up
gh-end servers.

sor cluster available
ng a software tool
called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications

MapReduce: not without Controversy

Backwards step in programming paradigm

Sub-optimal: brute force, no indexing

Not novel: 25 year-old ideas from DBMS lit

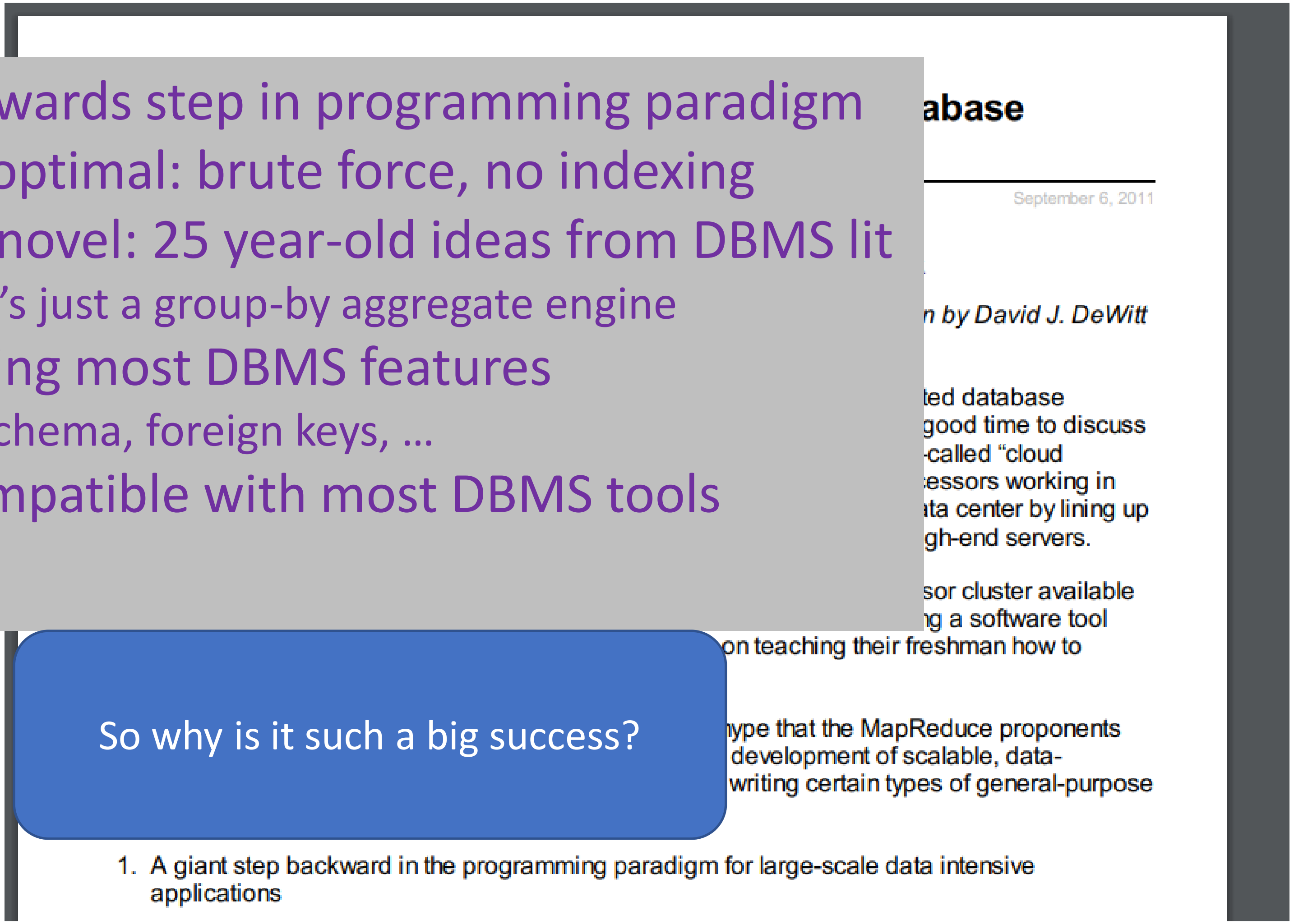
- It's just a group-by aggregate engine

Missing most DBMS features

- Schema, foreign keys, …

Incompatible with most DBMS tools

So why is it such a big success?


abase

September 6, 2011

n by David J. DeWitt

ted database
good time to discuss
-called "cloud
essors working in
ata center by lining up
gh-end servers.

sor cluster available
ng a software tool
on teaching their freshman how to

hype that the MapReduce proponents
development of scalable, data-
writing certain types of general-purpose

1. A giant step backward in the programming paradigm for large-scale data intensive applications

Why is MapReduce backwards?

Why is MapReduce backwards?

Backwards step in programming paradigm

Why is MapReduce backwards?

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing

Why is MapReduce backwards?

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Not novel: 25 year-old ideas from DBMS lit
- It's just a group-by aggregate engine

Why is MapReduce backwards?

- Backwards step in programming paradigm
- Sub-optimal: brute force, no indexing
- Not novel: 25 year-old ideas from DBMS lit
  - It's just a group-by aggregate engine
- Missing most DBMS features
  - Schema, foreign keys, …

Why is MapReduce backwards?

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Not novel: 25 year-old ideas from DBMS lit
- It's just a group-by aggregate engine

Missing most DBMS features
- Schema, foreign keys, …

Incompatible with most DBMS tools

Why is MapReduce backwards?

Backwards step in programming paradigm

Sub-optimal: brute force, no indexing

Not novel: 25 year-old ideas from DBMS lit

- It's just a group-by aggregate engine

Missing most DBMS features

- Schema, foreign keys, …

Incompatible with most DBMS tools

So why is it such a big success?

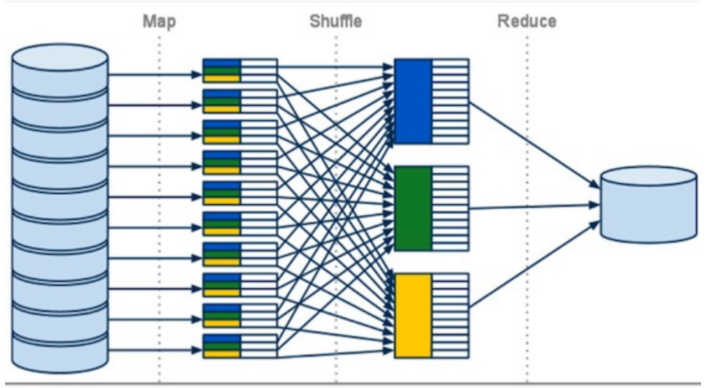
MapReduce and Dataflow

MapReduce and Dataflow

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

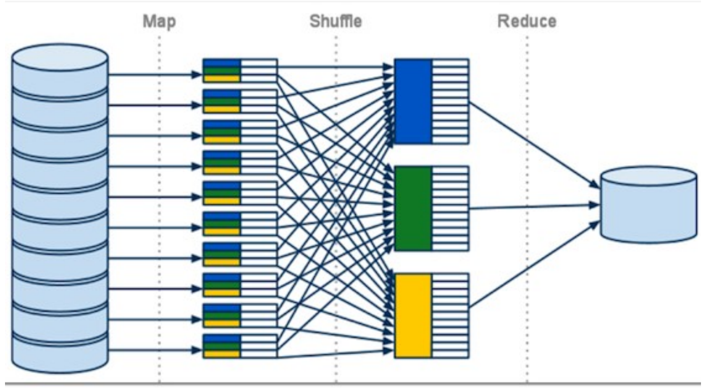
MapReduce and Dataflow

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



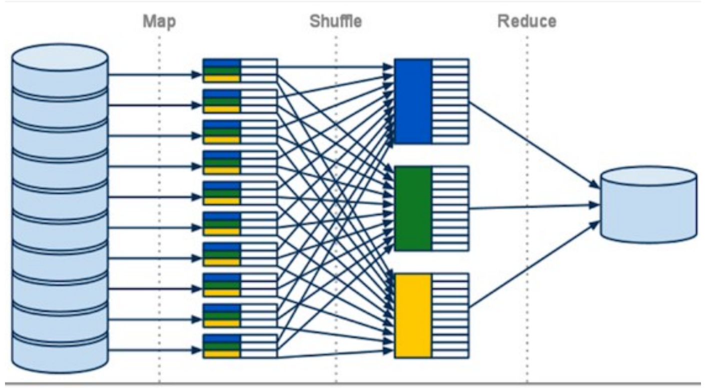
MapReduce and Dataflow

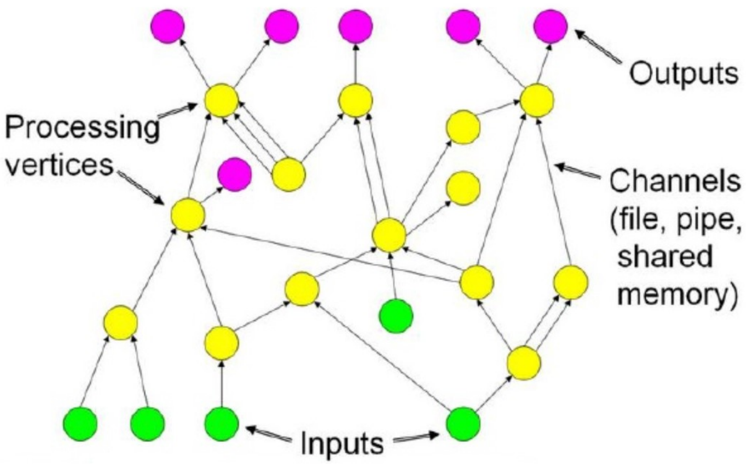
- MR is a ***dataflow*** engine
- Lots of others
  - Dryad
  - DryadLINQ
  - Dandelion
  - CIEL
  - GraphChi/Pregel
  - Spark



MapReduce and Dataflow

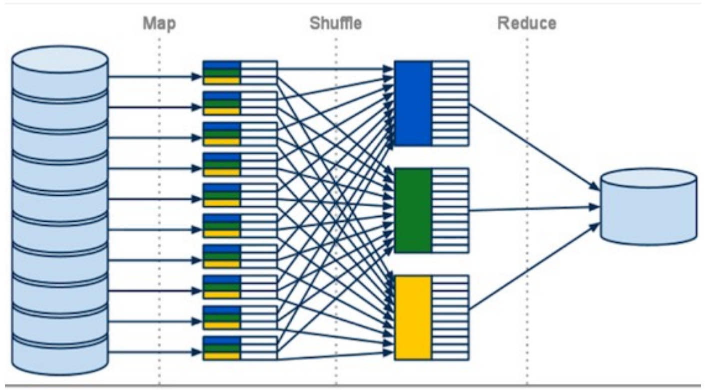
- MR is a ***dataflow*** engine
- Lots of others
  - Dryad
  - DryadLINQ
  - Dandelion
  - CIEL
  - GraphChi/Pregel
  - Spark

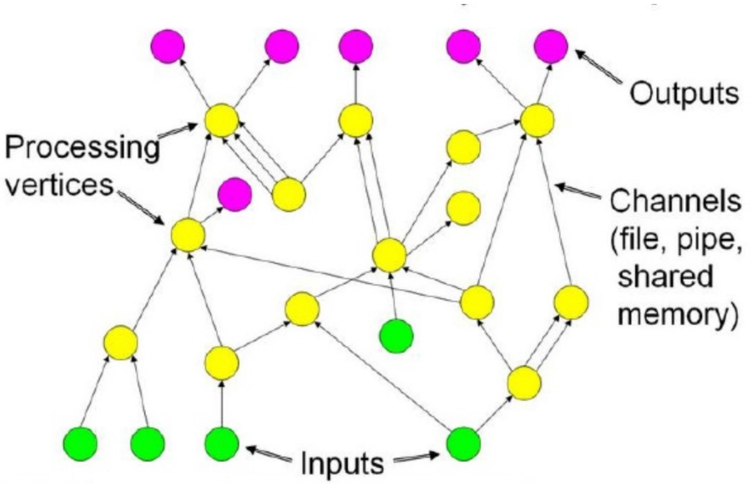




MapReduce and Dataflow

- MR is a ***dataflow*** engine
- Lots of others
  - Dryad
  - DryadLINQ
  - Dandelion
  - CIEL





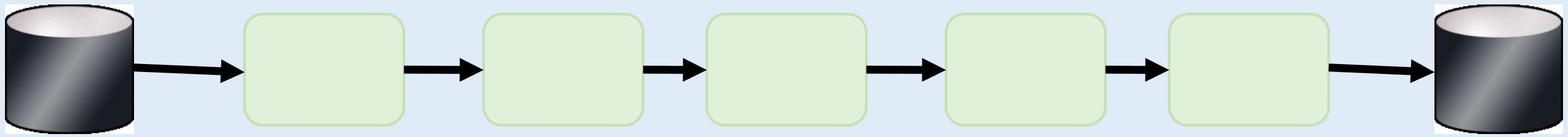
Taxonomies:

- DAG instead of BSP
- Interface variety
  - Memory FIFO
  - Disk
  - Network
- Flexible Modular Composition

Dryad (2007): 2-D Piping

- Unix Pipes: 1-D

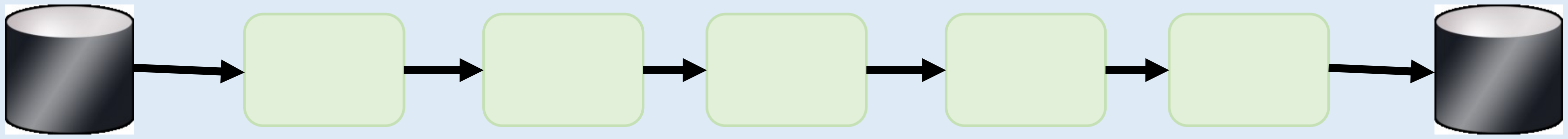
grep | sed | sort | awk | perl



Dryad (2007): 2-D Piping

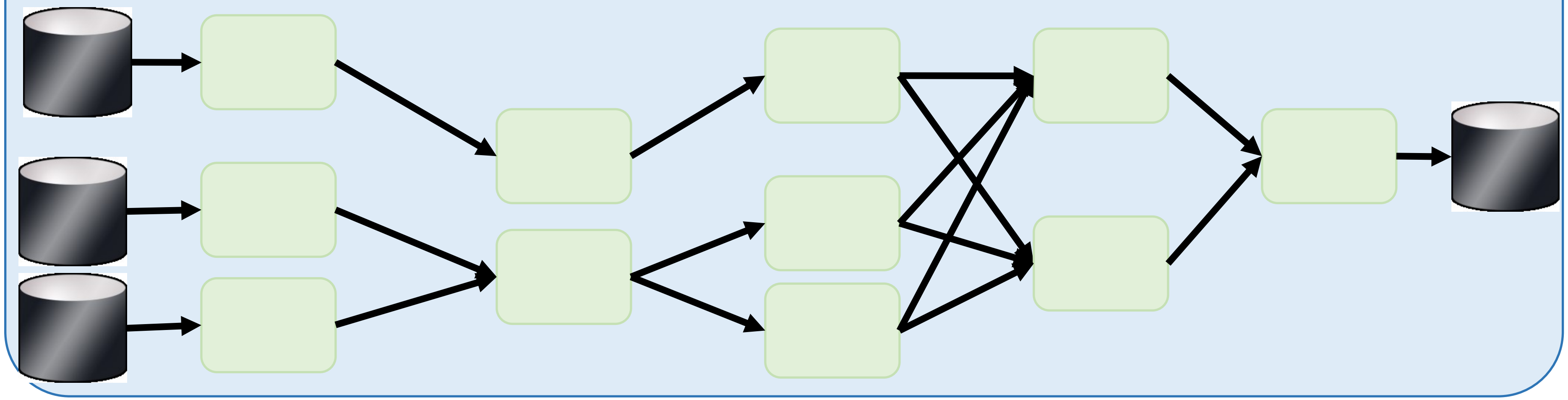
- Unix Pipes: 1-D

grep | sed | sort | awk | perl

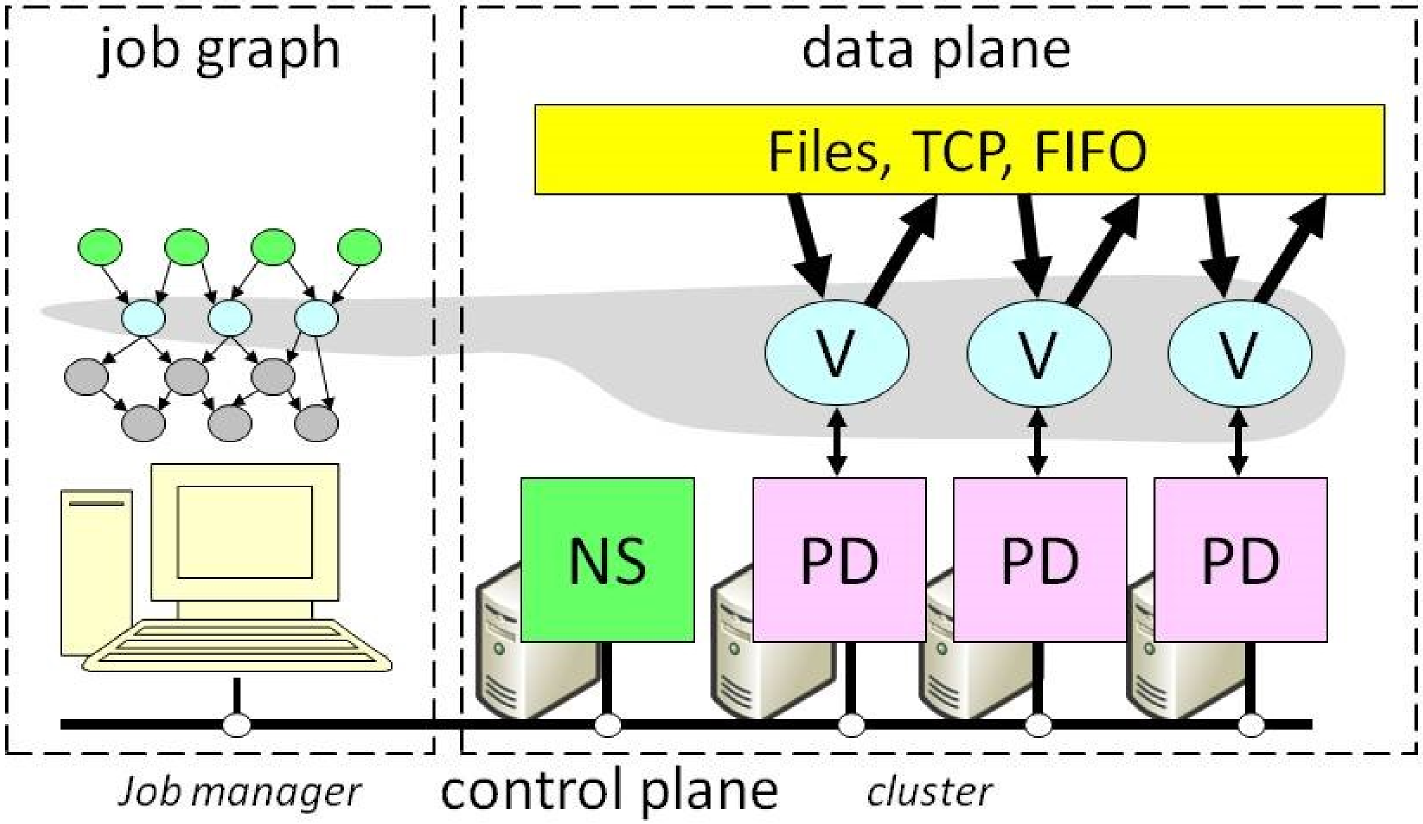


- Dryad: 2-D

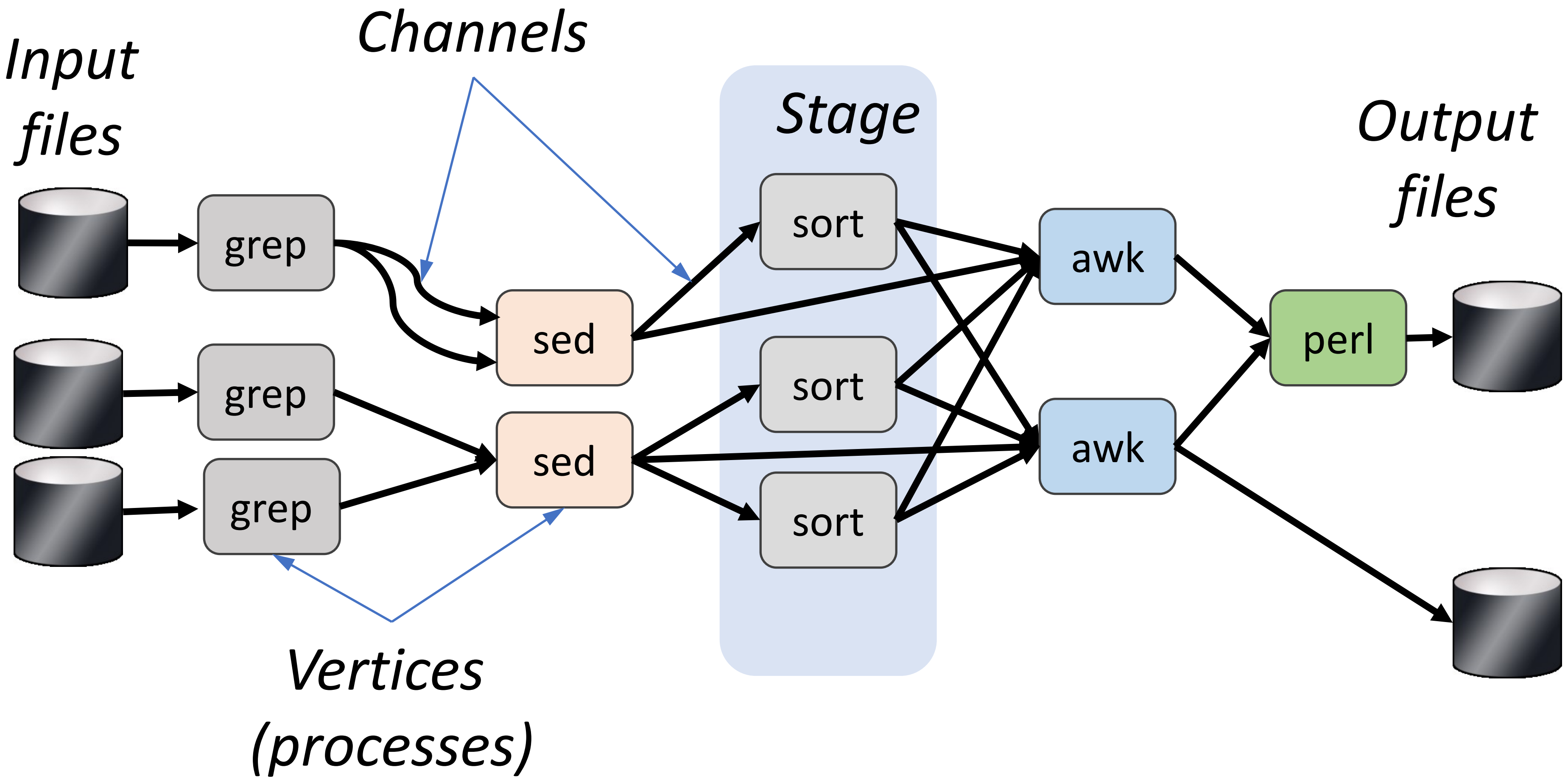
grep^{1000} | sed^{500} | sort^{1000} | awk^{500} | perl^{50}



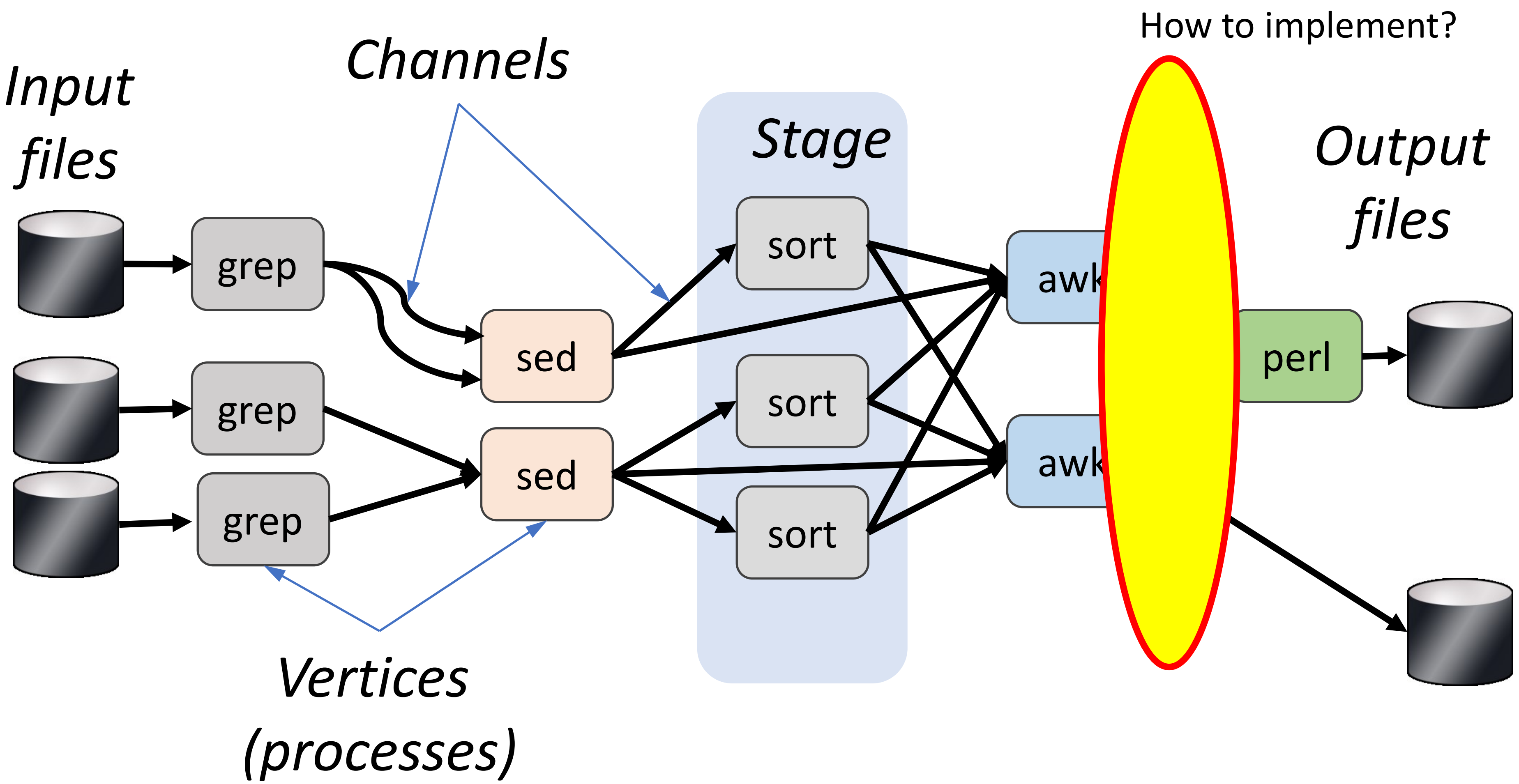
Dataflow Engines



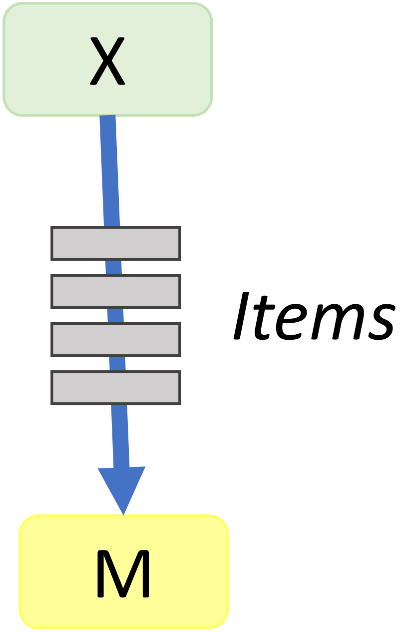
Dataflow Job Structure



Dataflow Job Structure



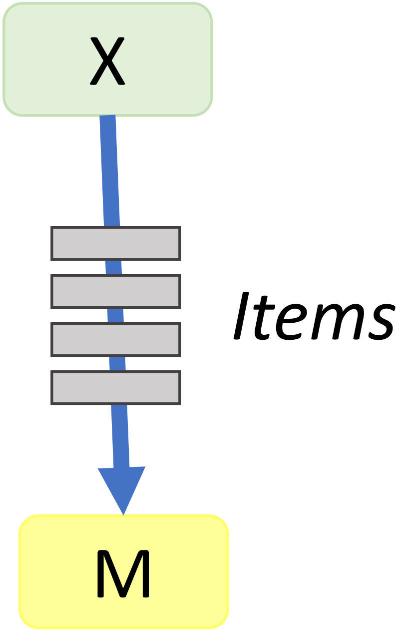
Channels



Finite streams of items

- distributed filesystem files
  (persistent)
- SMB/NTFS files
  (temporary)
- TCP pipes
  (inter-machine)
- memory FIFOs
  (intra-machine)

Channels



Key idea:
Encapsulate data movement behind channel abstraction → gets programmer out of the picture

Finite streams of items

- distributed filesystem files
  (persistent)
- SMB/NTFS files
  (temporary)
- TCP pipes
  (inter-machine)
- memory FIFOs
  (intra-machine)

Spark (2012) Background

Commodity clusters: important platform

- **In industry:** search, machine translation, ad targeting, …
- **In research:** bioinformatics, NLP, climate simulation, …

Cluster-scale models (e.g. MR) de facto standard

- Fault tolerance through replicated durable storage
- Dataflow is the common theme

Spark (2012) Background

Commodity clusters: important platform

- **In industry:** search, machine translation, ad targeting, …
- **In research:** bioinformatics, NLP, climate simulation, …

Cluster-scale models (e.g. MR) de facto standard

- Fault tolerance through replicated durable storage
- Dataflow is the common theme

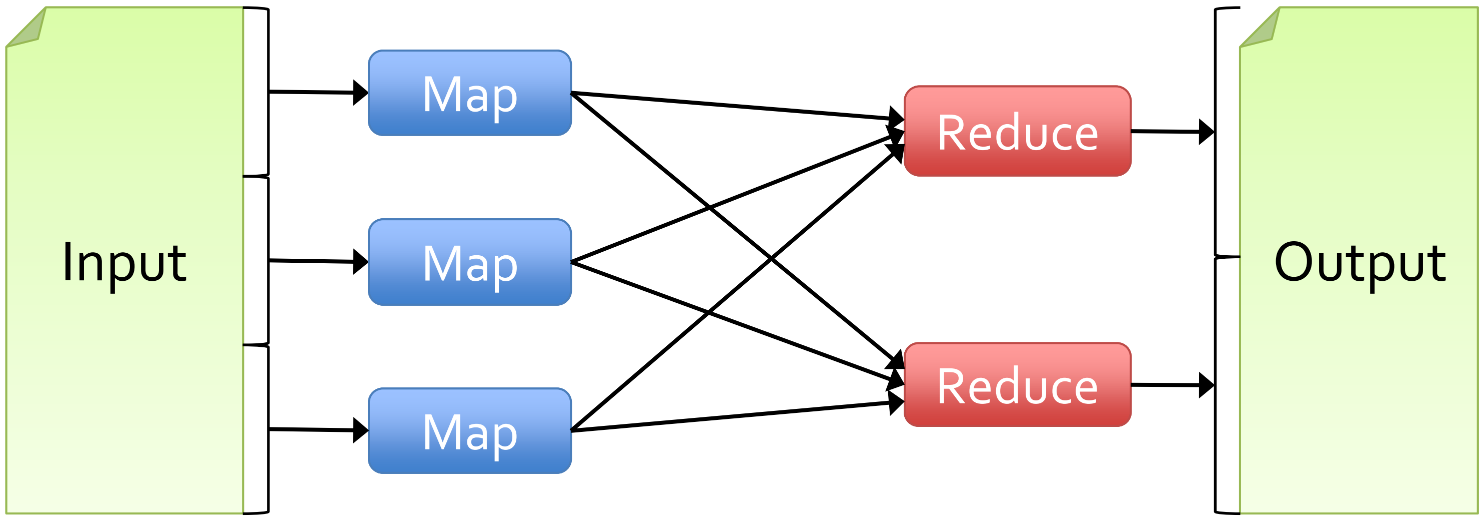
Multi-core

Iteration

Motivation

Programming models for clusters transform data flowing from stable storage to stable storage

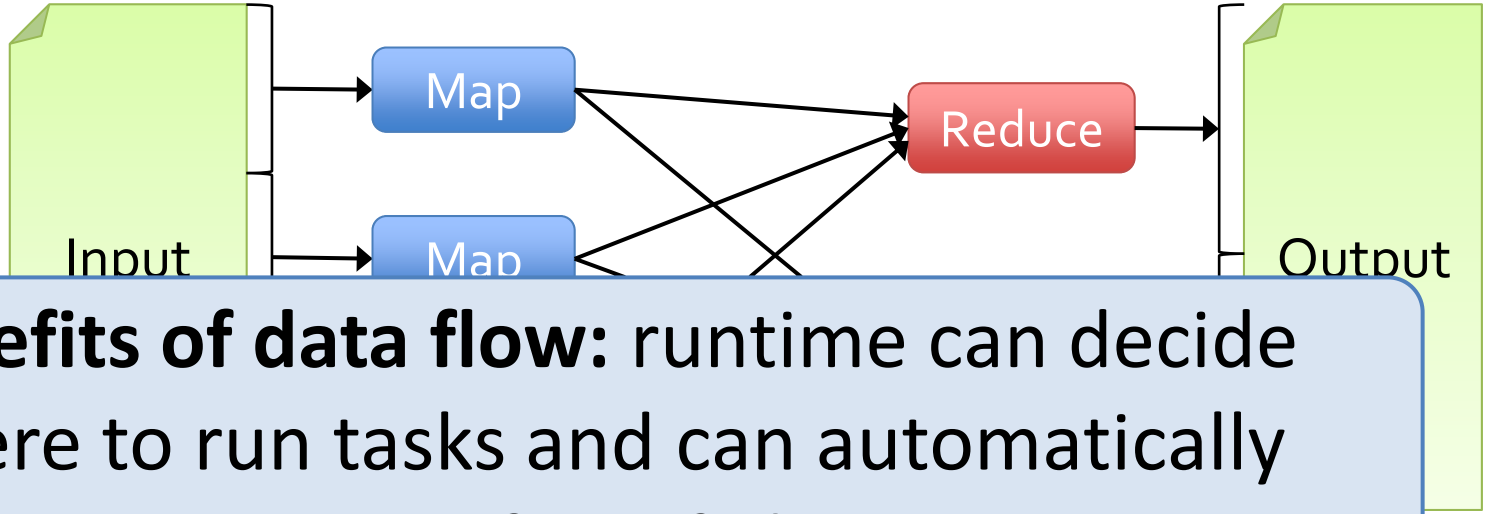
E.g., MapReduce:



Motivation

Programming models for clusters transform data flowing from stable storage to stable storage

E.g., MapReduce:



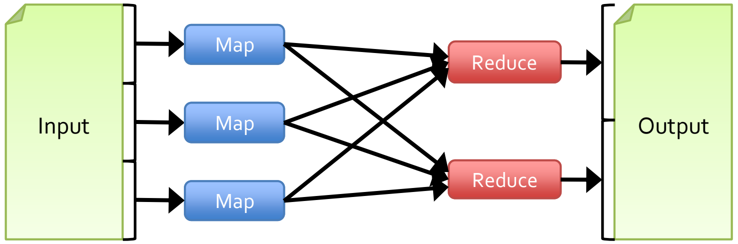
Benefits of data flow: runtime can decide where to run tasks and can automatically recover from failures

Iterative Computations: PageRank

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

$$\Sigma_{i \in neighbors} \text{rank}_i / |\text{neighbors}_i|$$

```
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(_ + _)
}
```

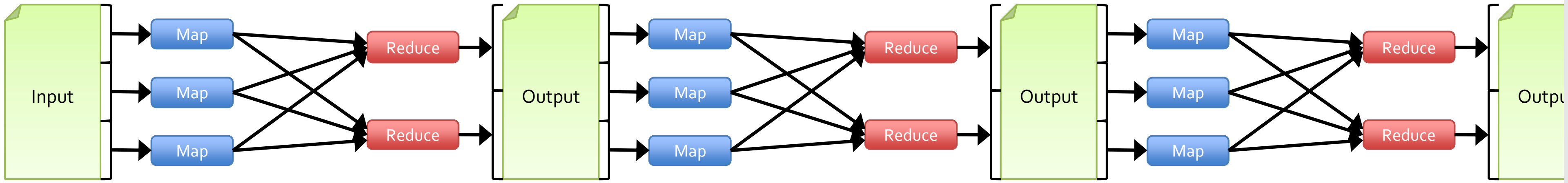


Iterative Computations: PageRank

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(_ + _)
}
```



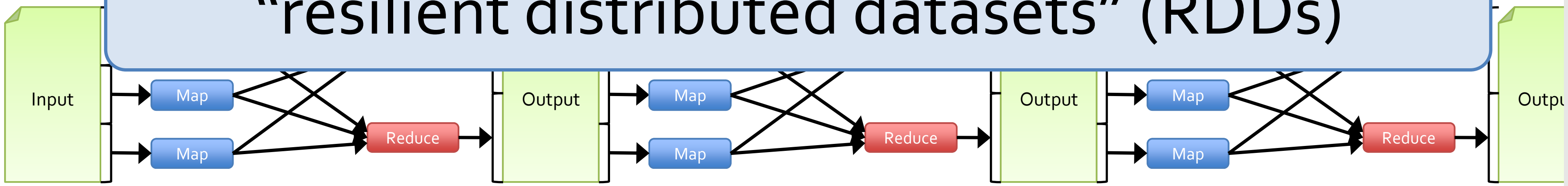
Iterative Computations: PageRank

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(_ + _)
```

Solution: augment data flow model with "resilient distributed datasets" (RDDs)



Programming Model

- Resilient distributed datasets (RDDs)
  - Immutable collections partitioned across cluster that can be rebuilt if a partition is lost
  - Created by transforming data in stable storage using data flow operators (map, filter, group-by, …)
  - Can be *cached* across parallel operations

Programming Model

- Resilient distributed datasets (RDDs)
  - Immutable collections partitioned across cluster that can be rebuilt if a partition is lost
  - Created by transforming data in stable storage using data flow operators (map, filter, group-by, …)
  - Can be *cached* across parallel operations
- Parallel operations on RDDs
  - Reduce, collect, count, save, …

Programming Model

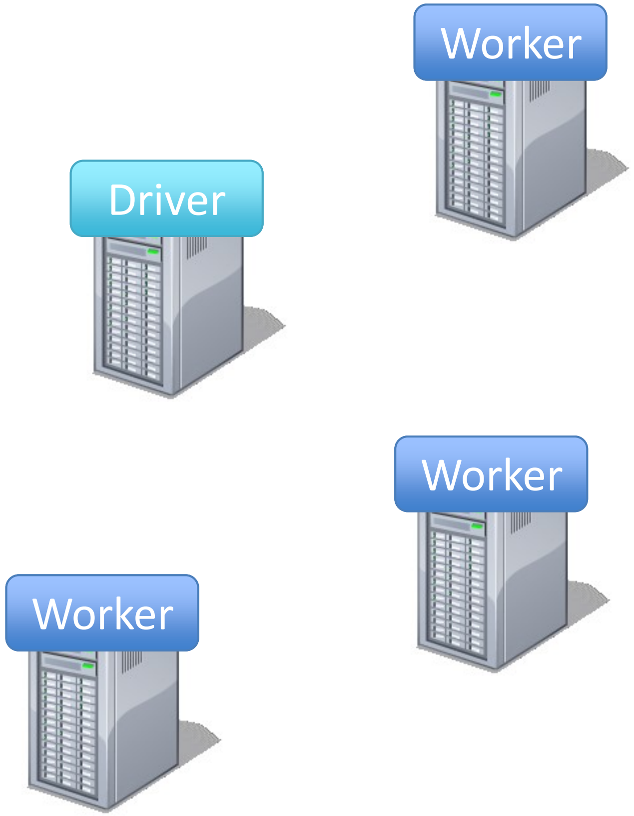
- Resilient distributed datasets (RDDs)
  - Immutable collections partitioned across cluster that can be rebuilt if a partition is lost
  - Created by transforming data in stable storage using data flow operators (map, filter, group-by, …)
  - Can be *cached* across parallel operations
- Parallel operations on RDDs
  - Reduce, collect, count, save, …
- Restricted shared variables
  - Accumulators, broadcast variables

Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

Example: Log Mining

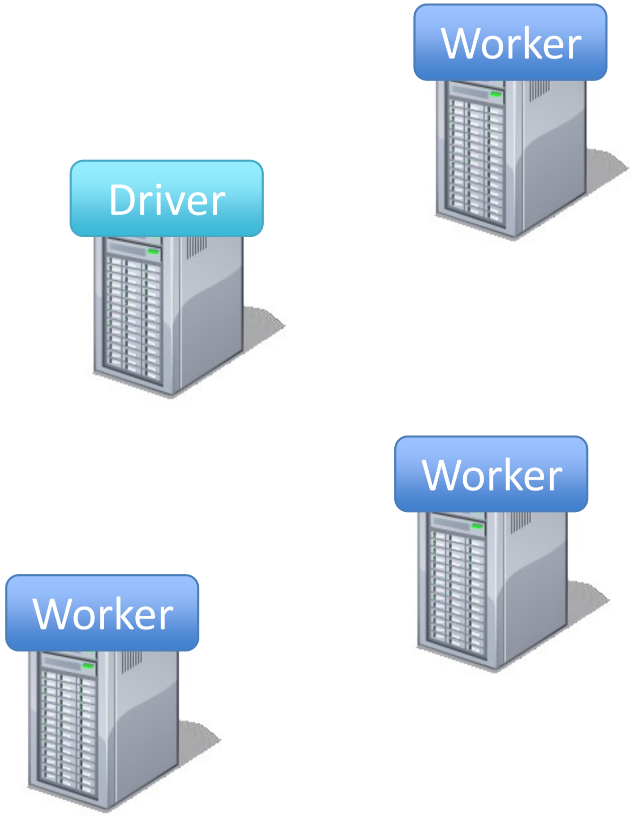
- Load error messages from a log into memory, then interactively search for various patterns



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

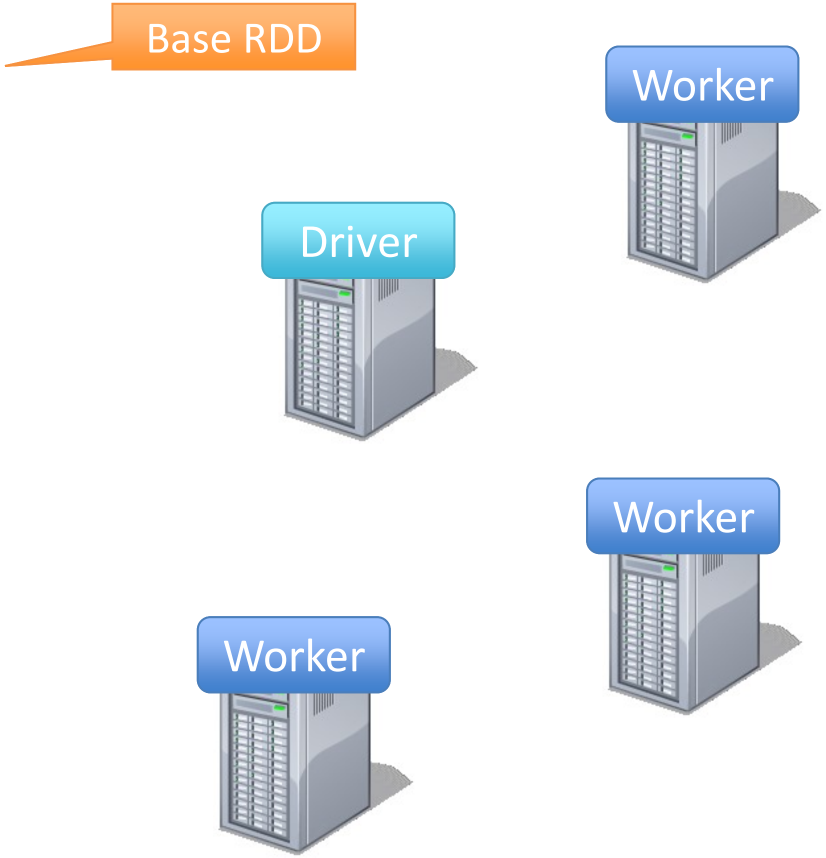
```
lines = spark.textFile("hdfs://...")
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

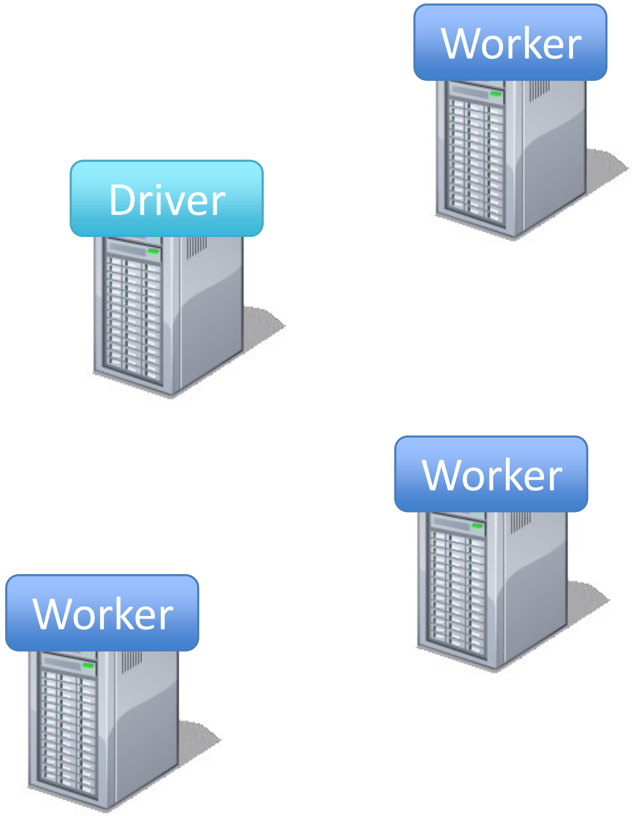
```
lines = spark.textFile("hdfs://...")
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

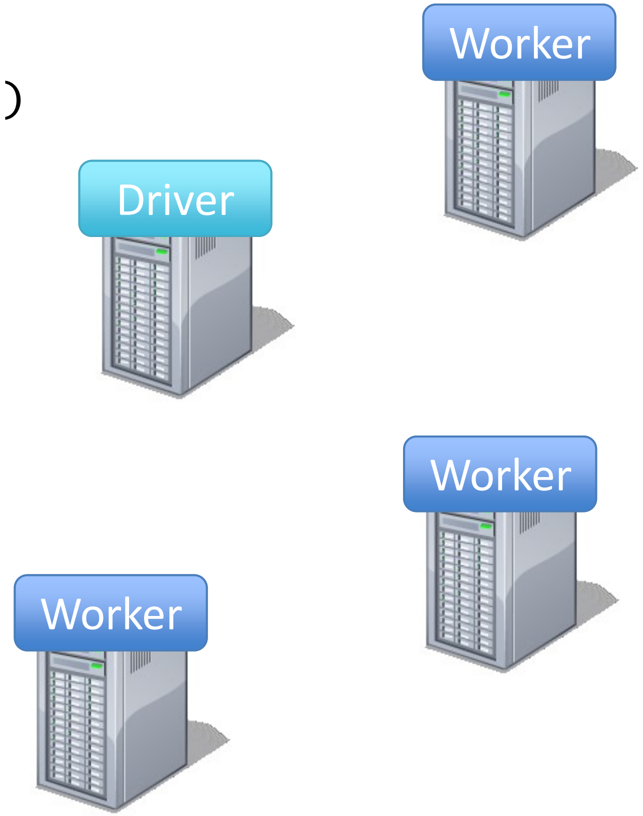
```
lines = spark.textFile("hdfs://...")
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

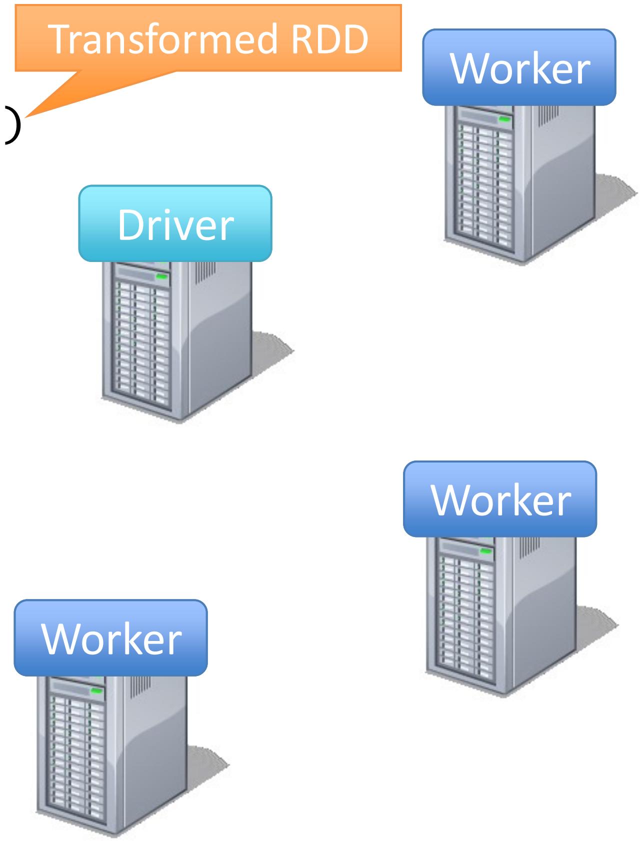
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

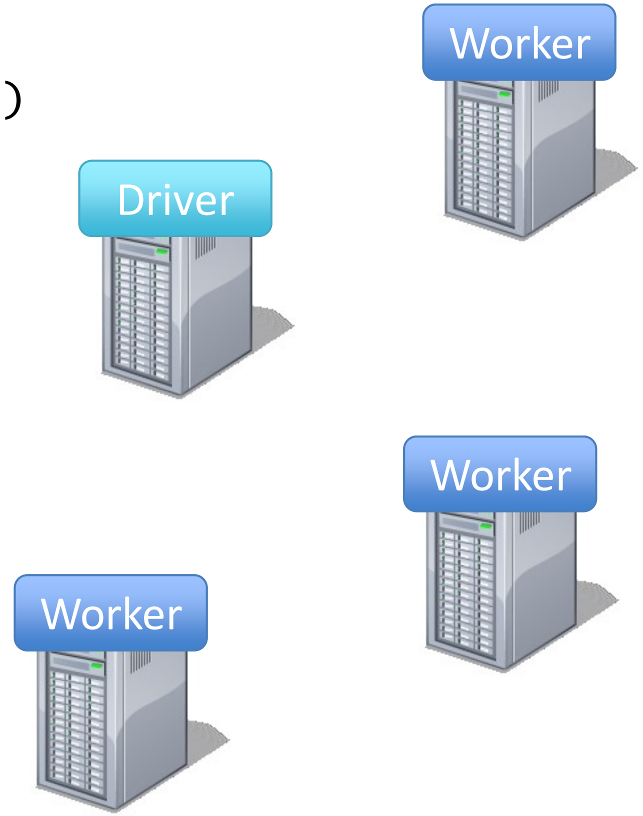
```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(_.startsWith(“ERROR”))
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

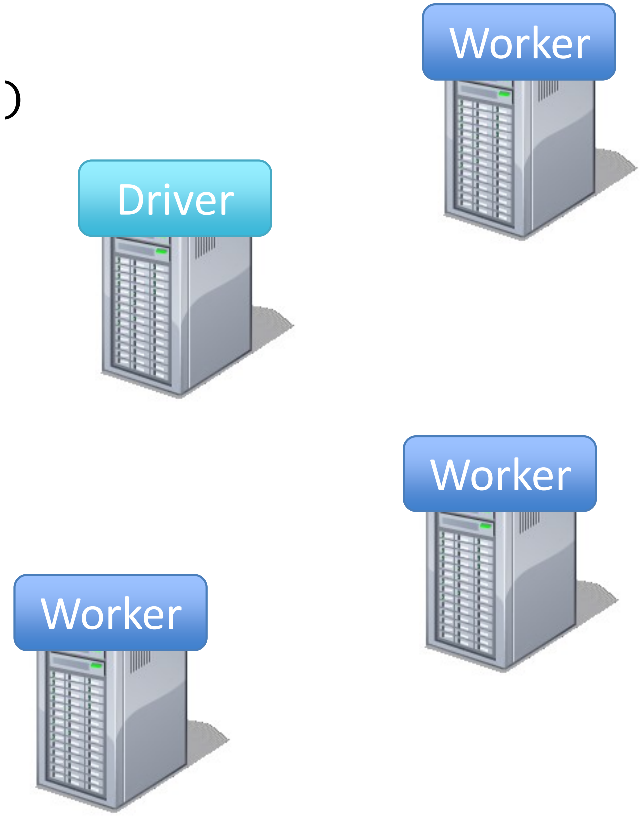
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

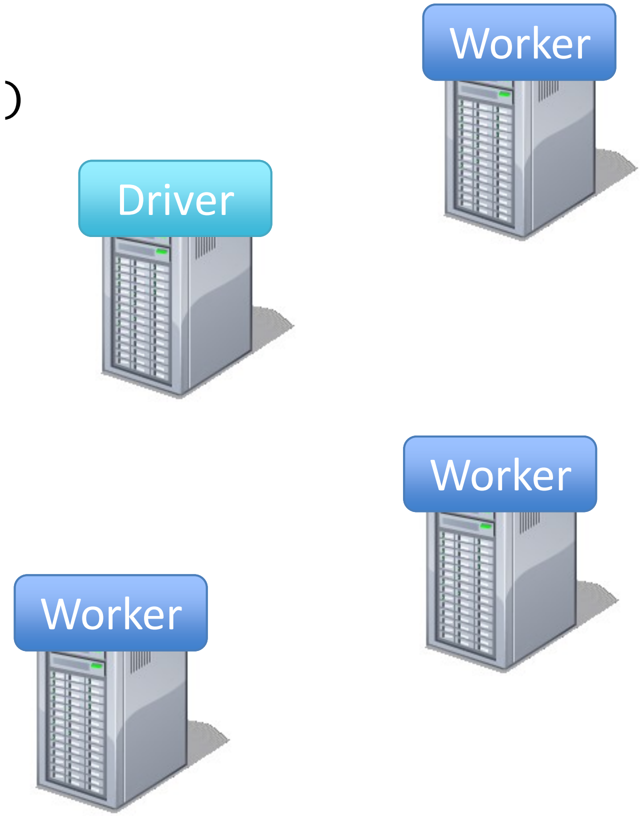
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

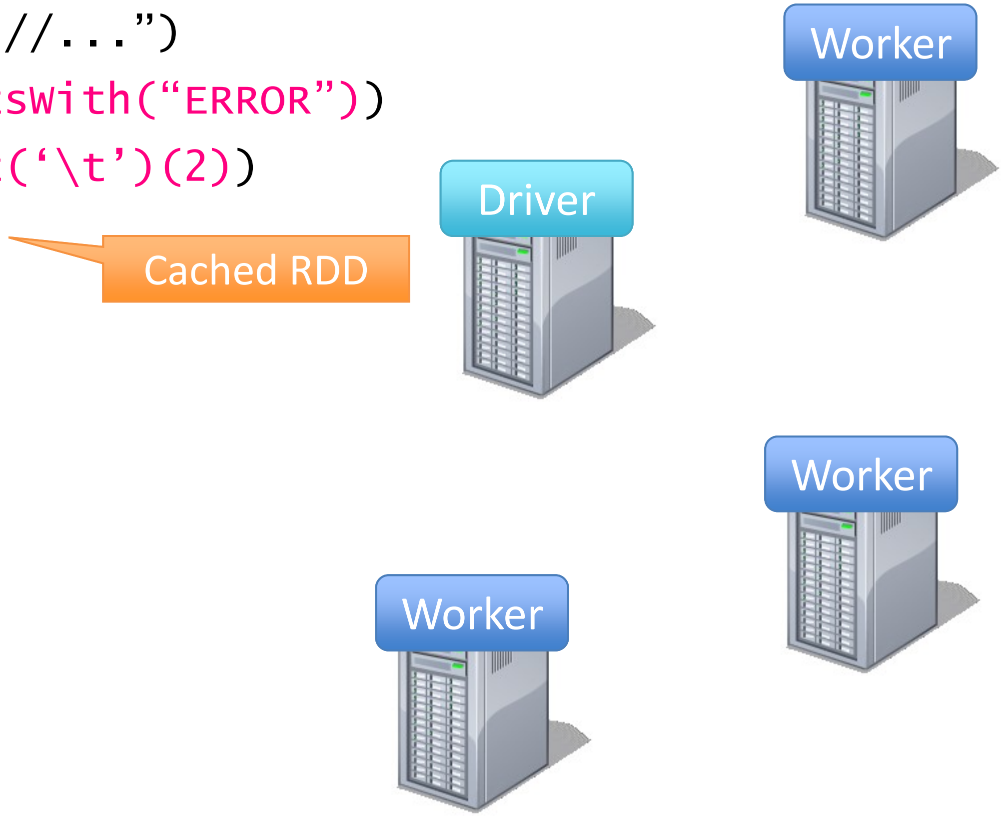
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

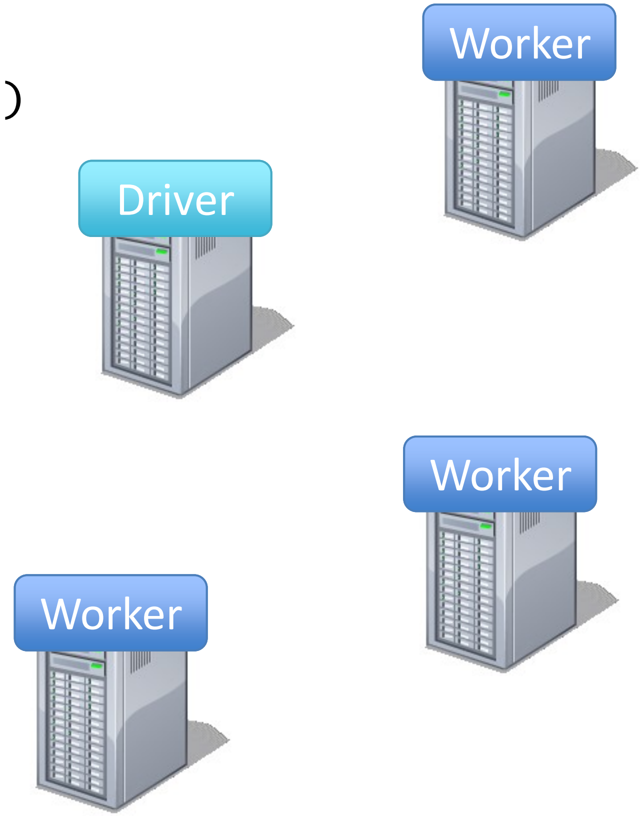
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

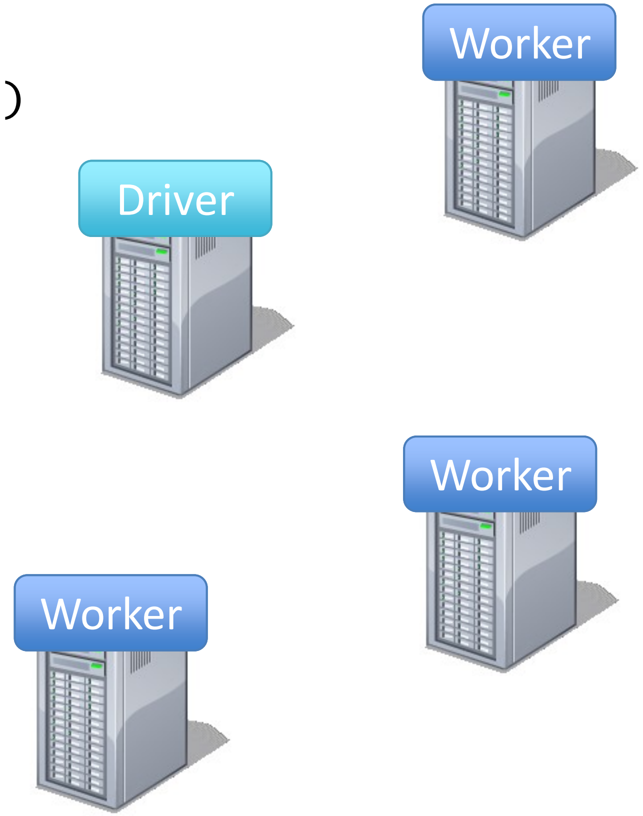
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

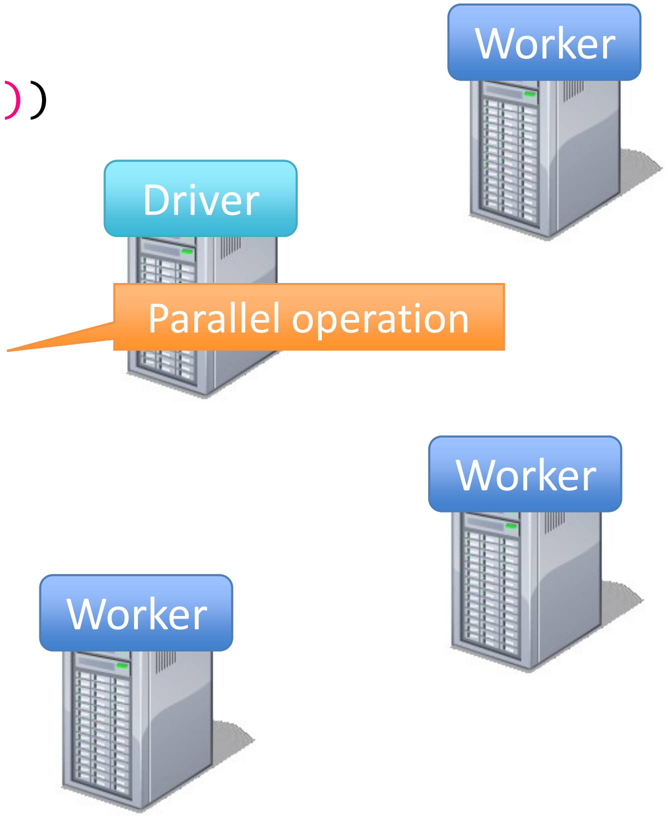
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

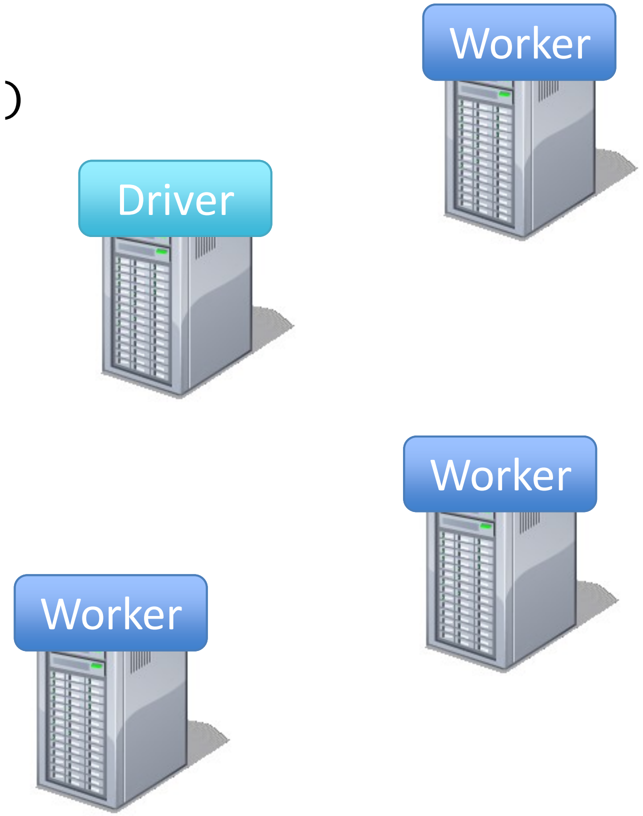
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

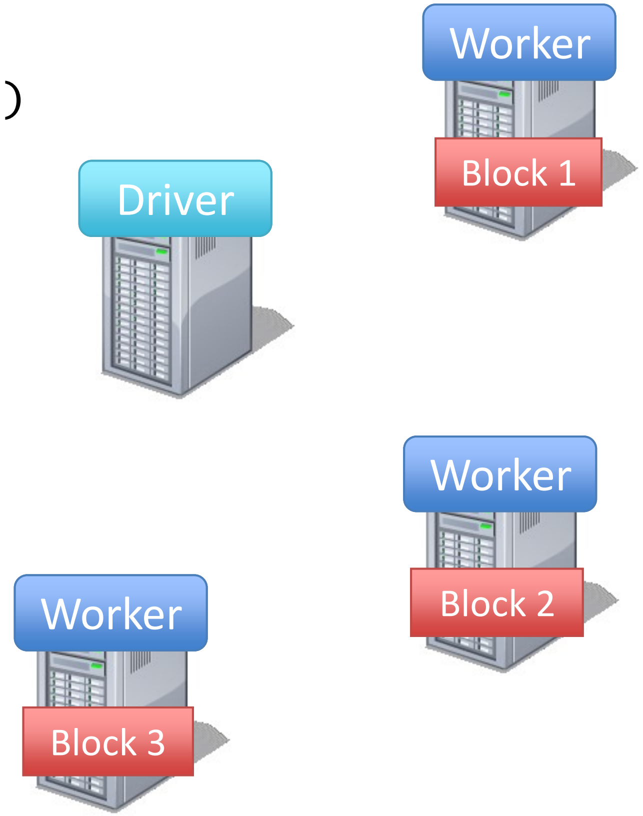
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

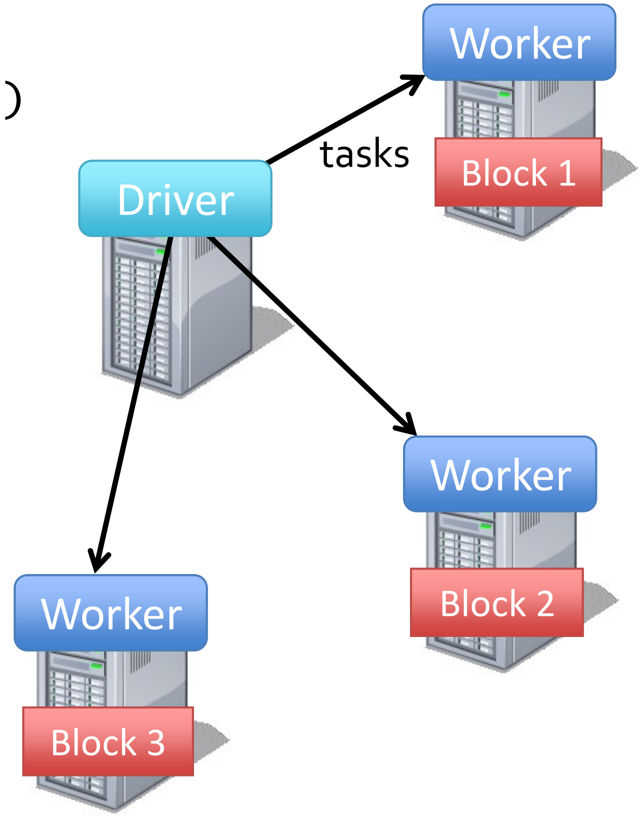
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

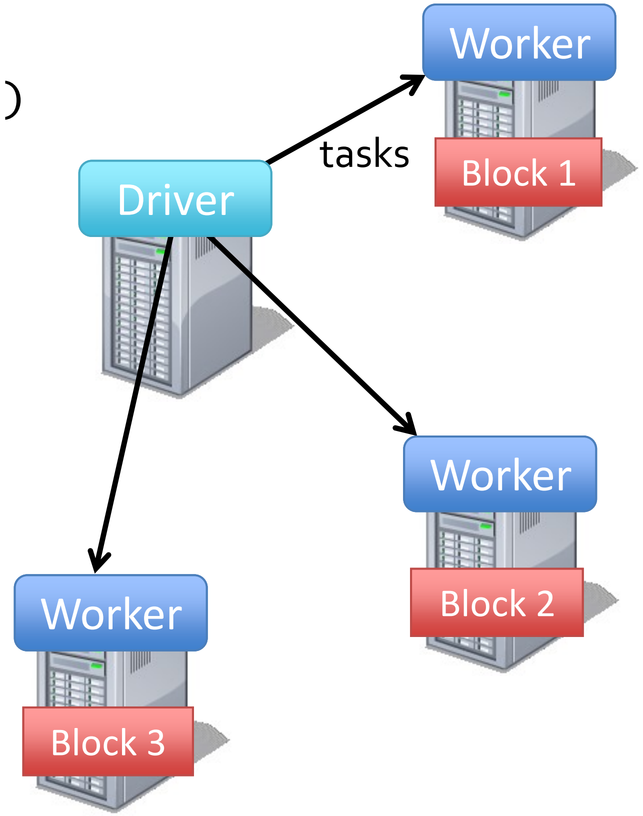
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

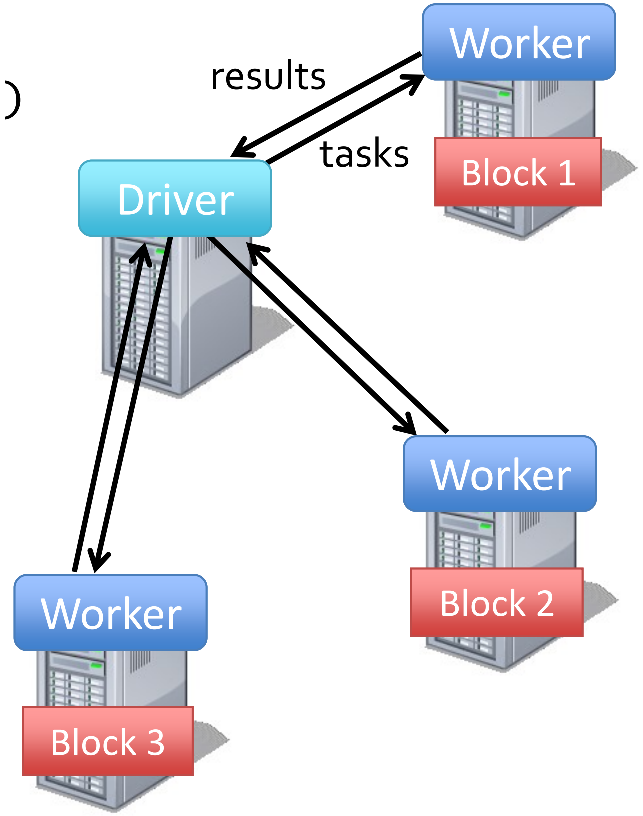
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

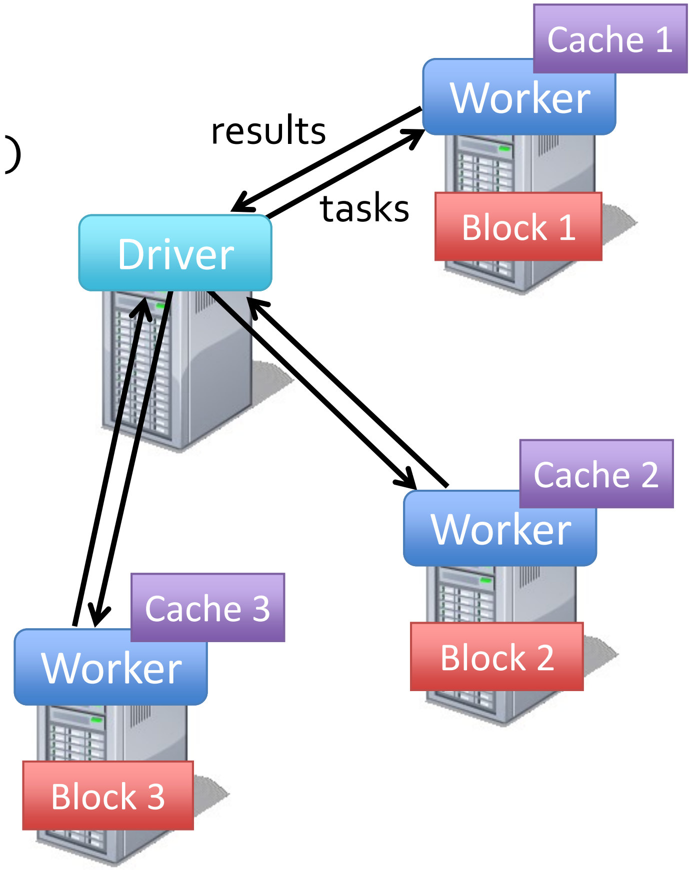
```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(_.startsWith(“ERROR”))
messages = errors.map(_.split(‘\t’)(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains(“foo”)).count
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

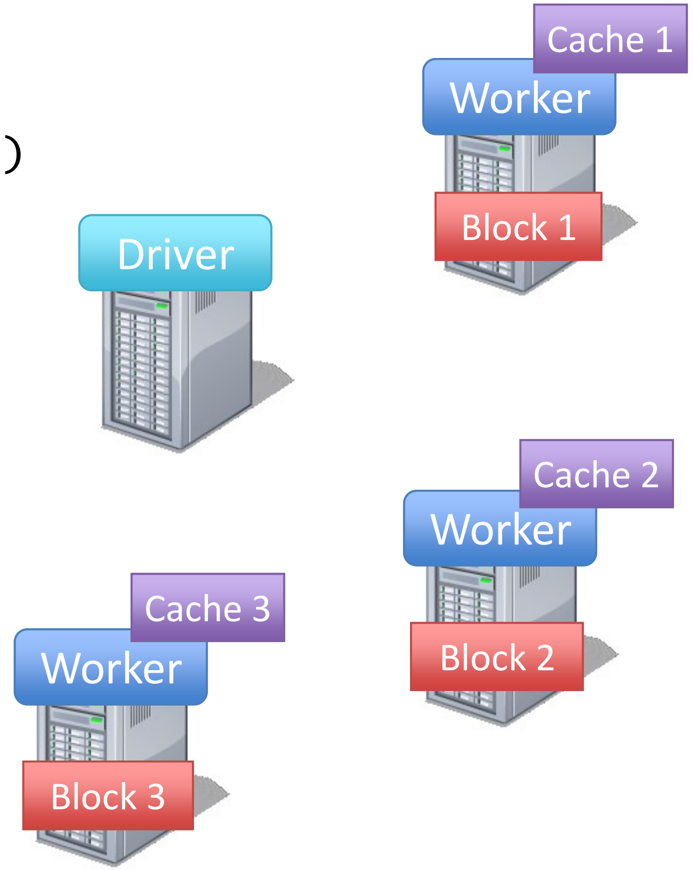
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

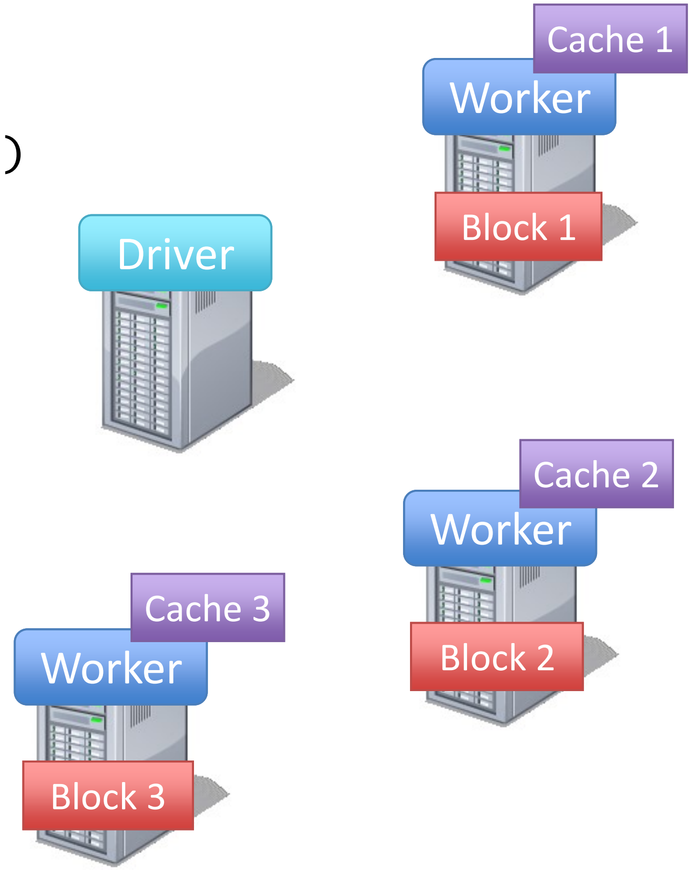
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

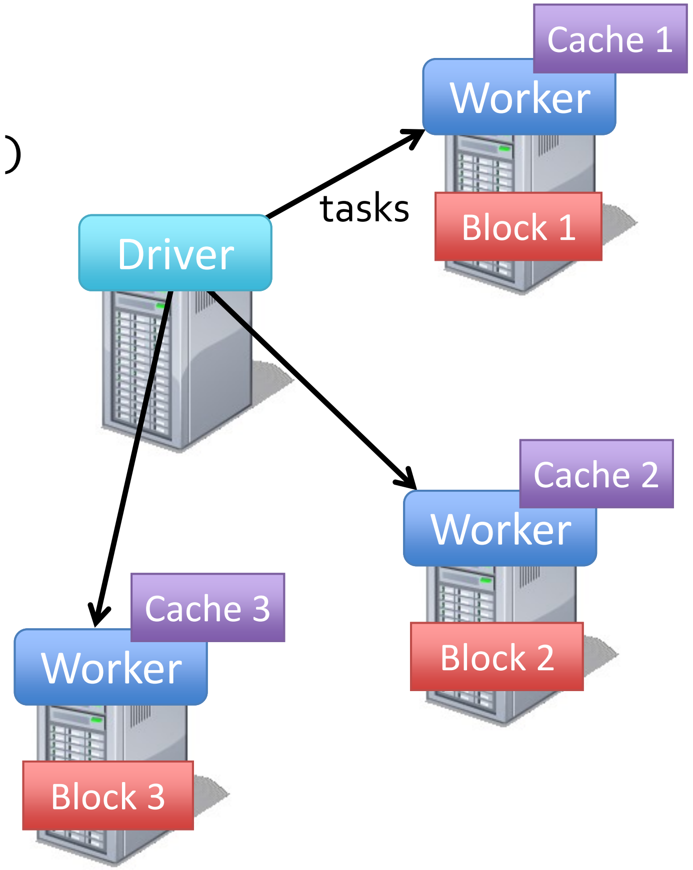
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

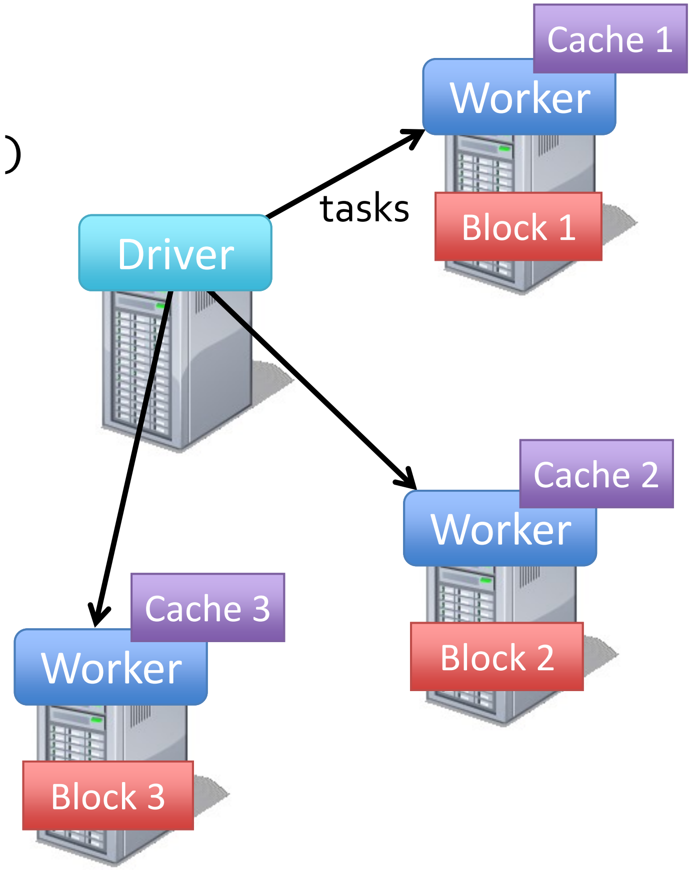
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

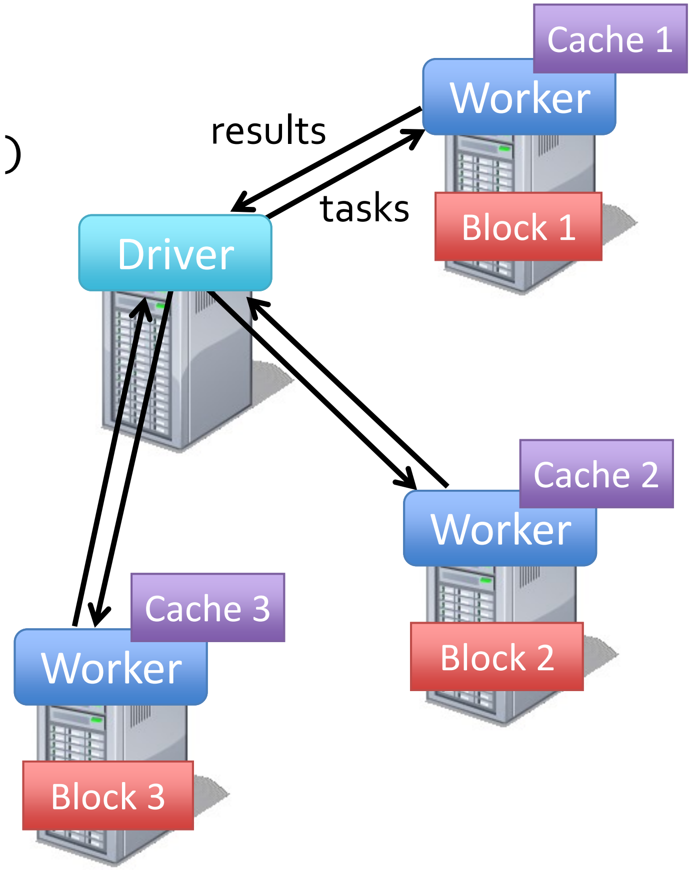
```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(_.startsWith(“ERROR”))
messages = errors.map(_.split(‘\t’)(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains(“foo”)).count
cachedMsgs.filter(_.contains(“bar”)).count
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

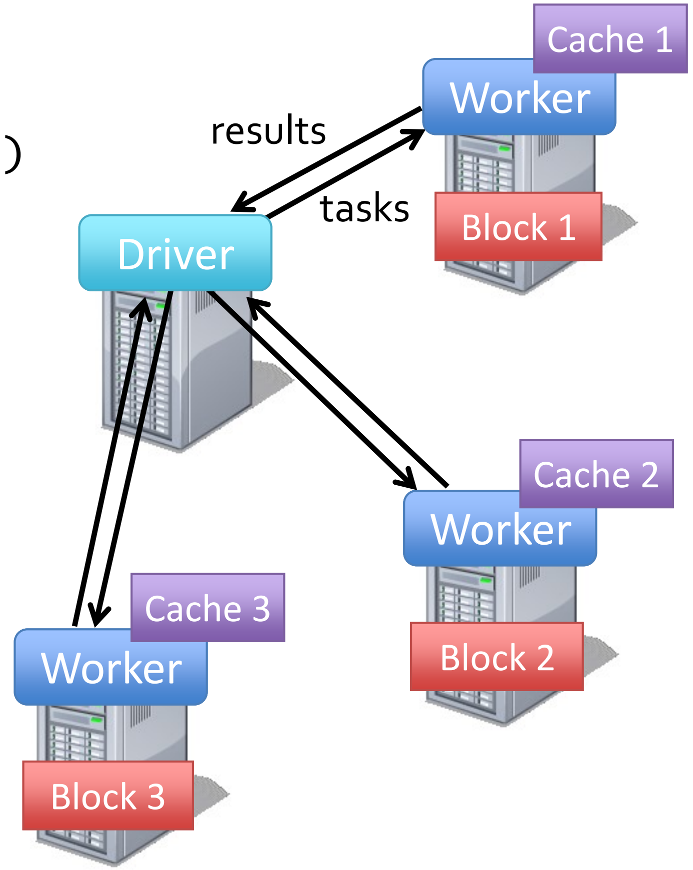
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
```



Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```

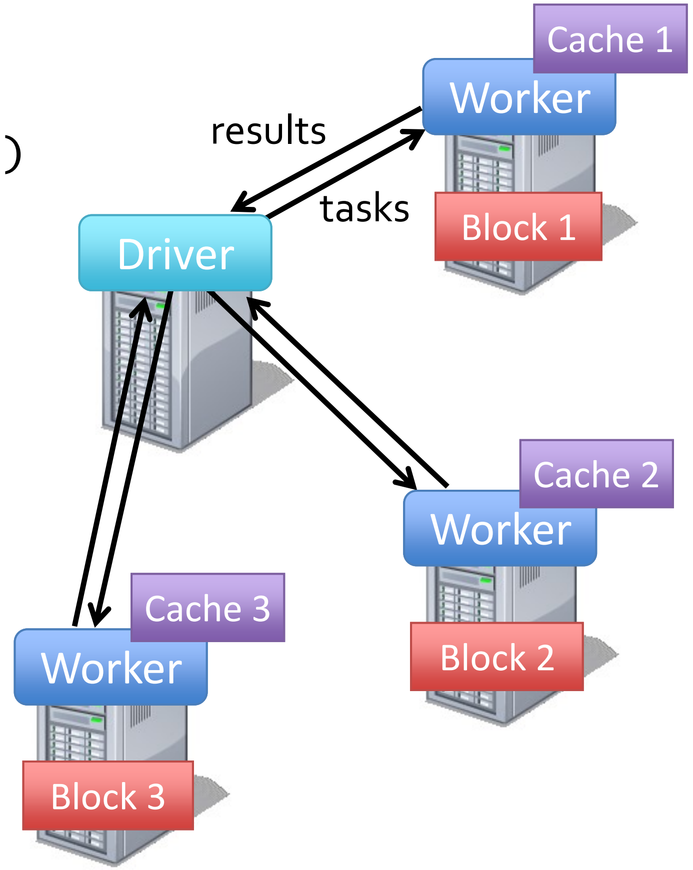


Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```

Result: full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)

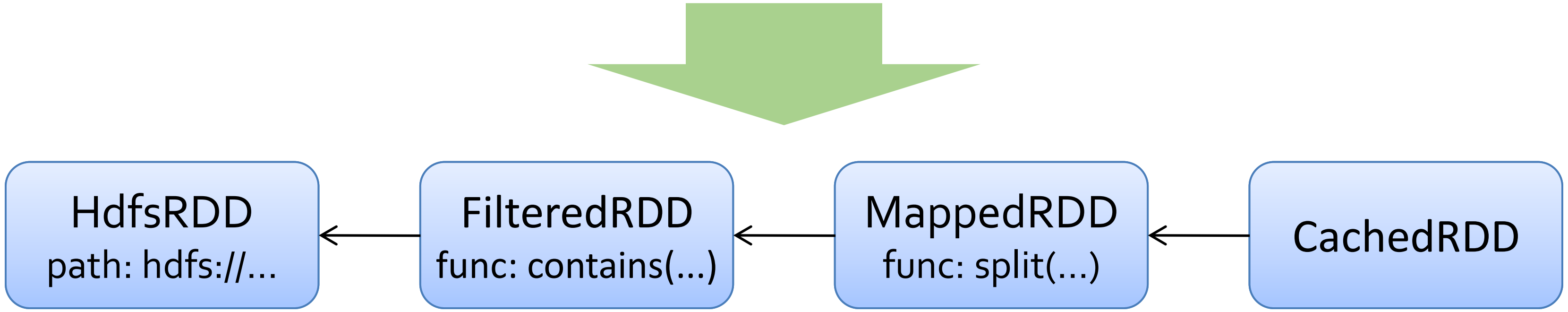


RDD Fault Tolerance

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

- Ex:

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



Data-Parallel Computation Systems

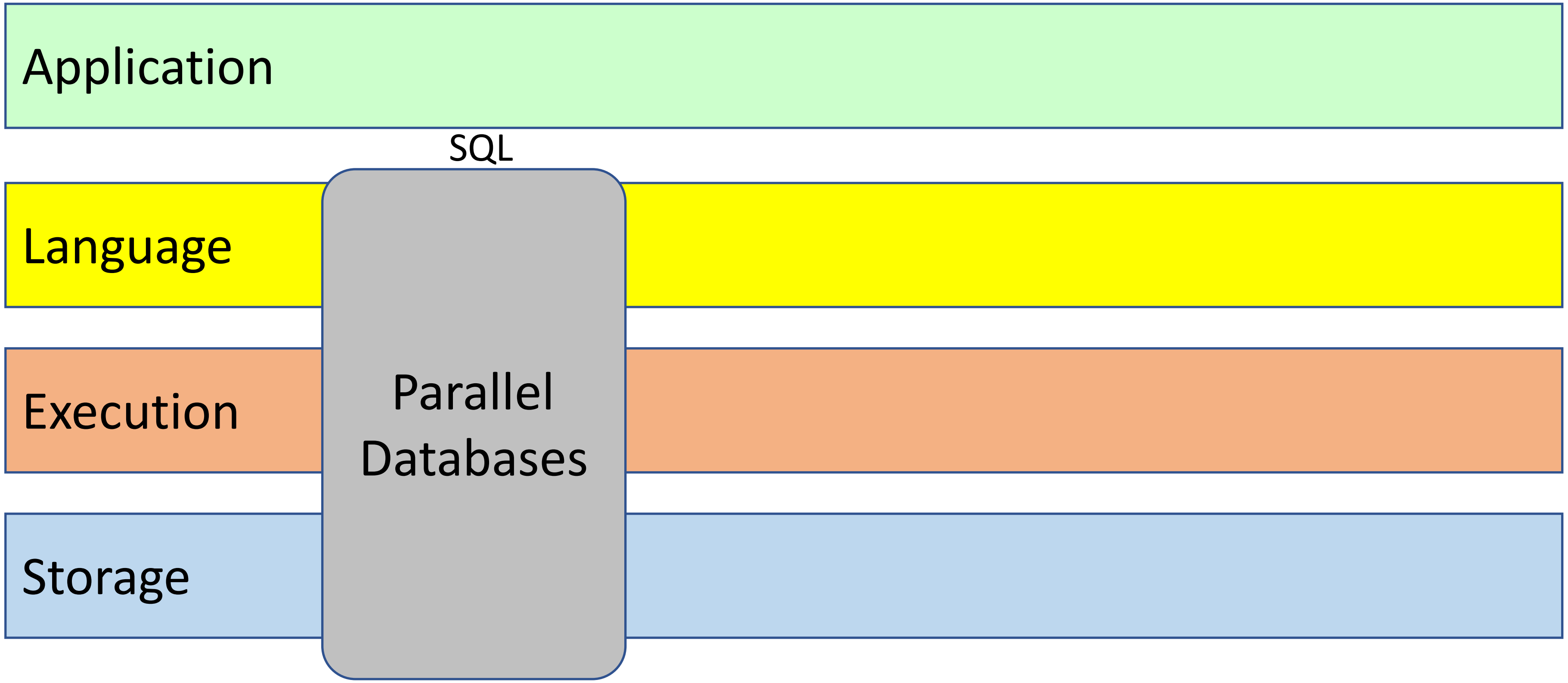
Application

Language

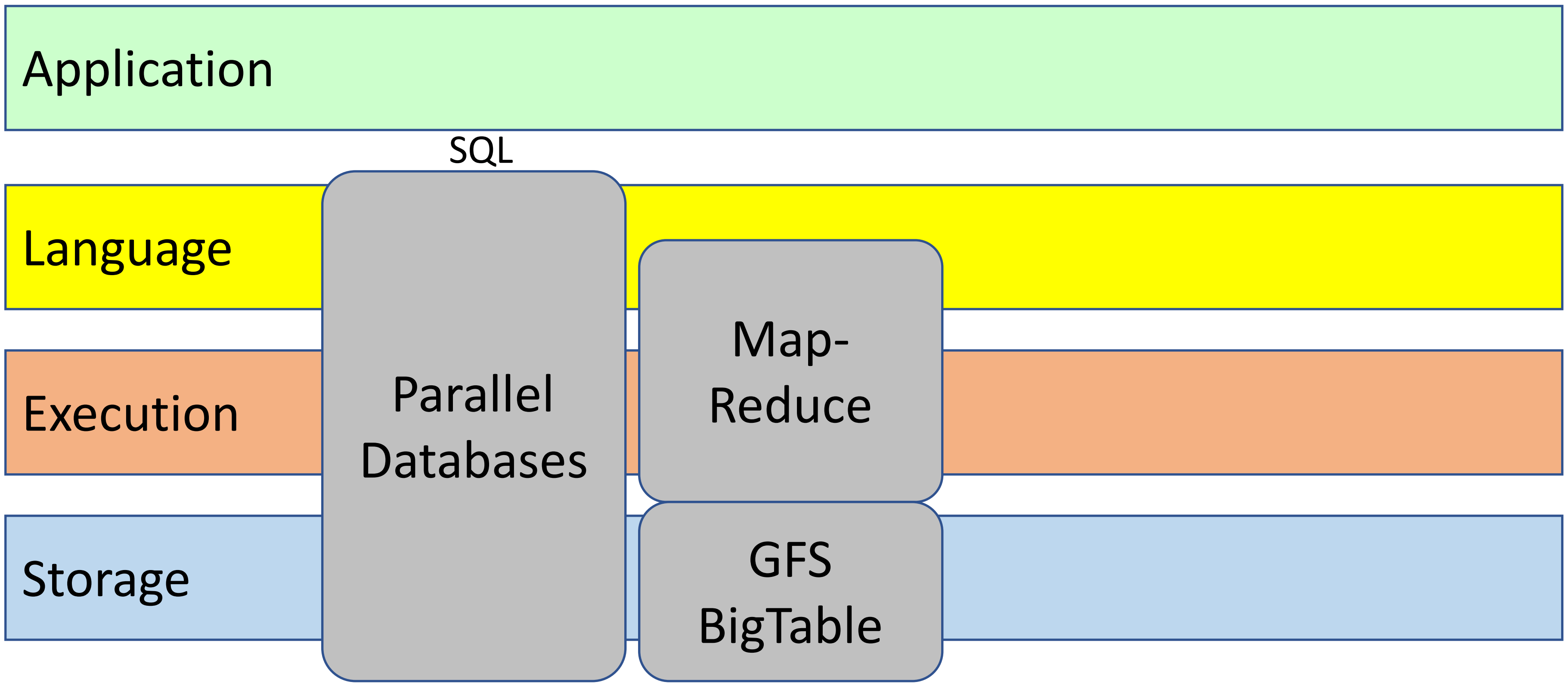
Execution

Storage

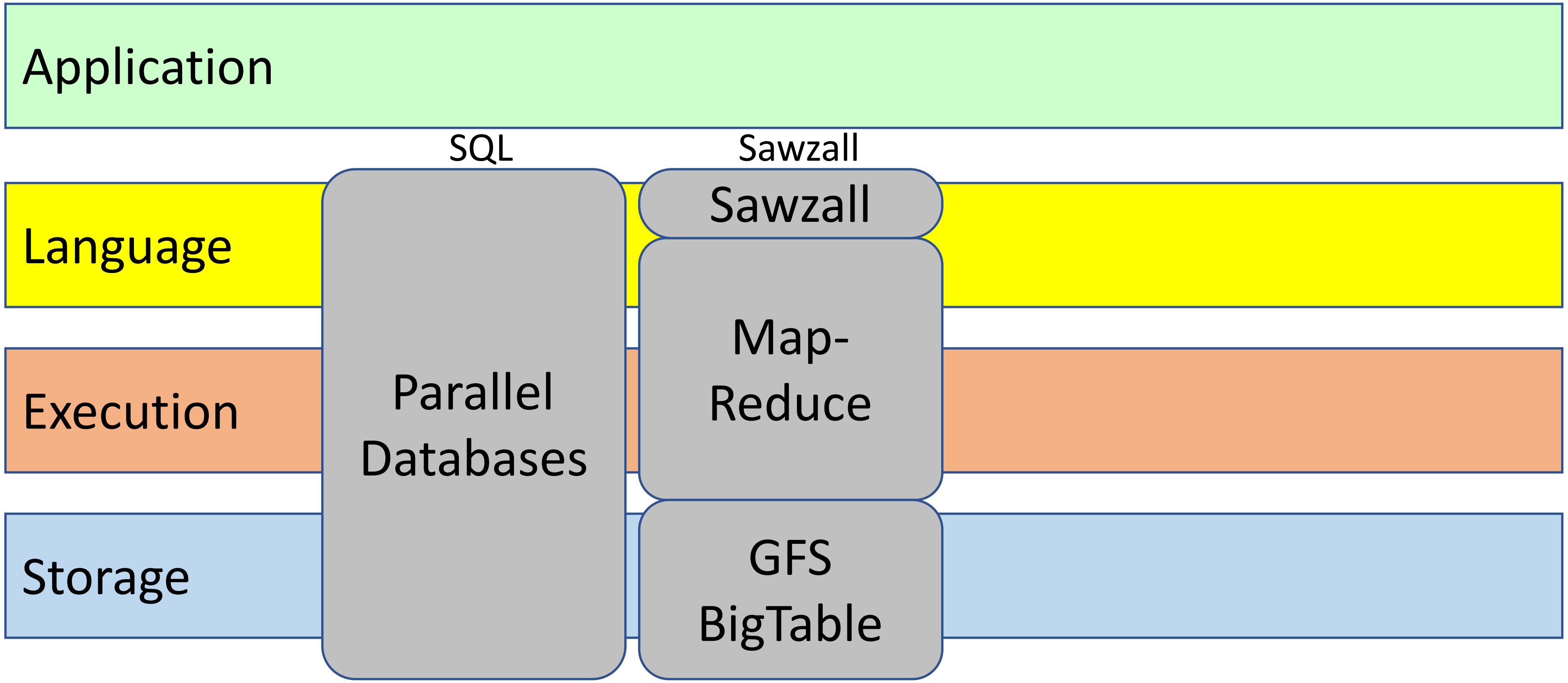
Data-Parallel Computation Systems



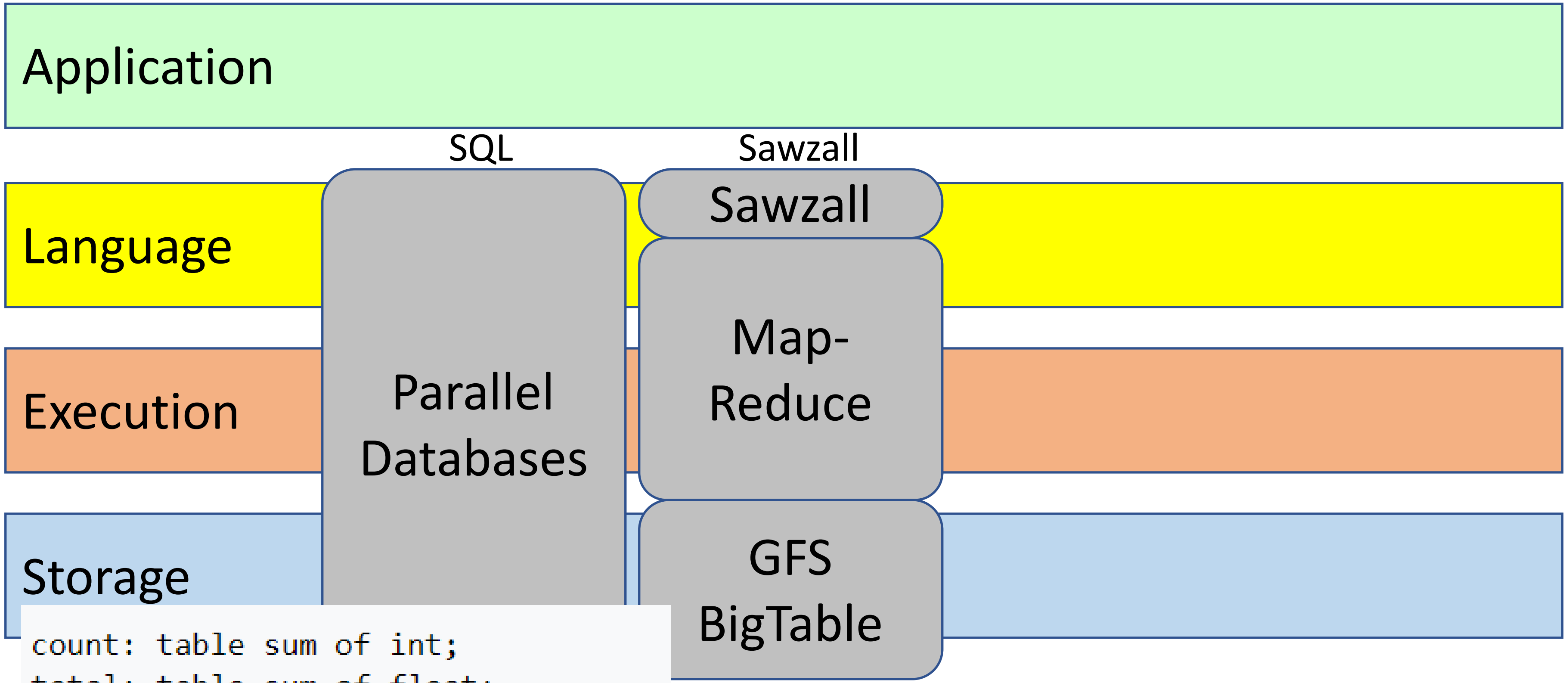
Data-Parallel Computation Systems



Data-Parallel Computation Systems

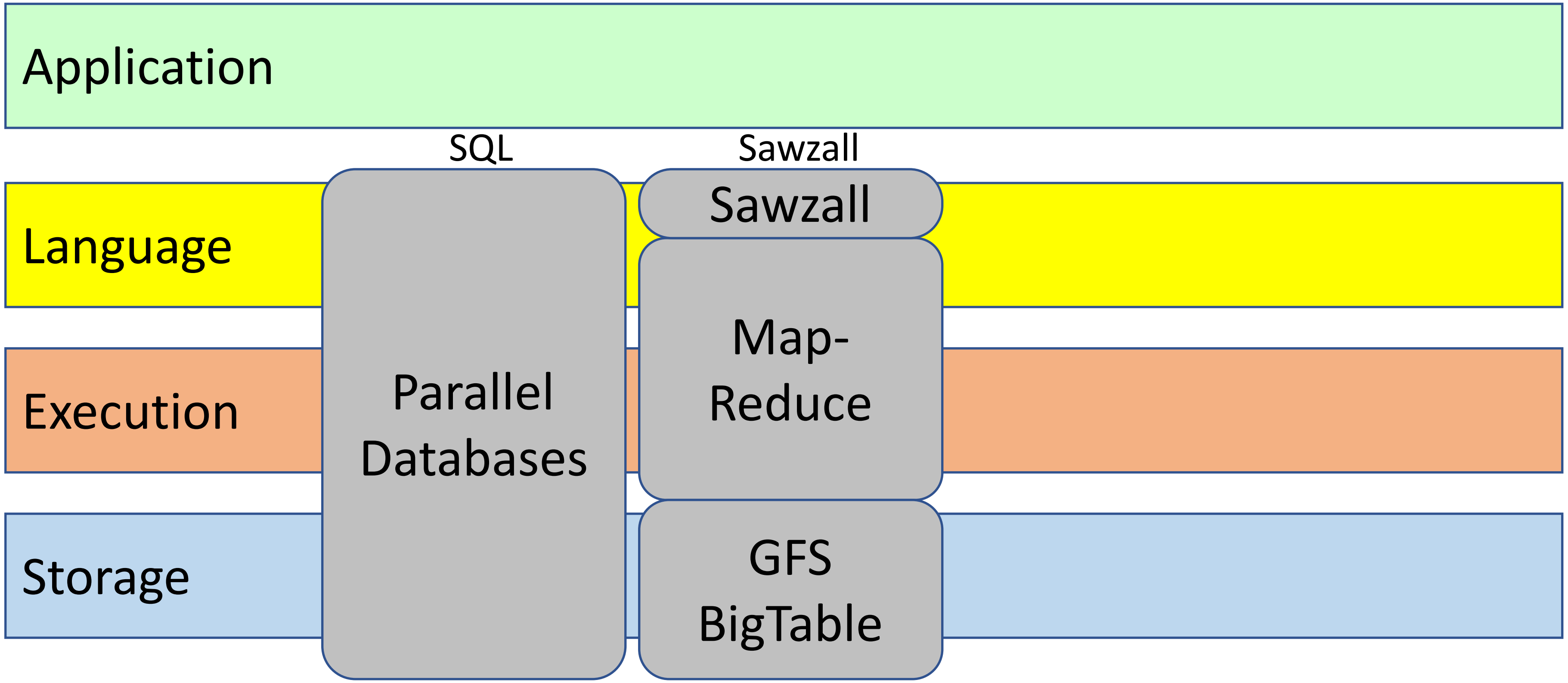


Data-Parallel Computation Systems

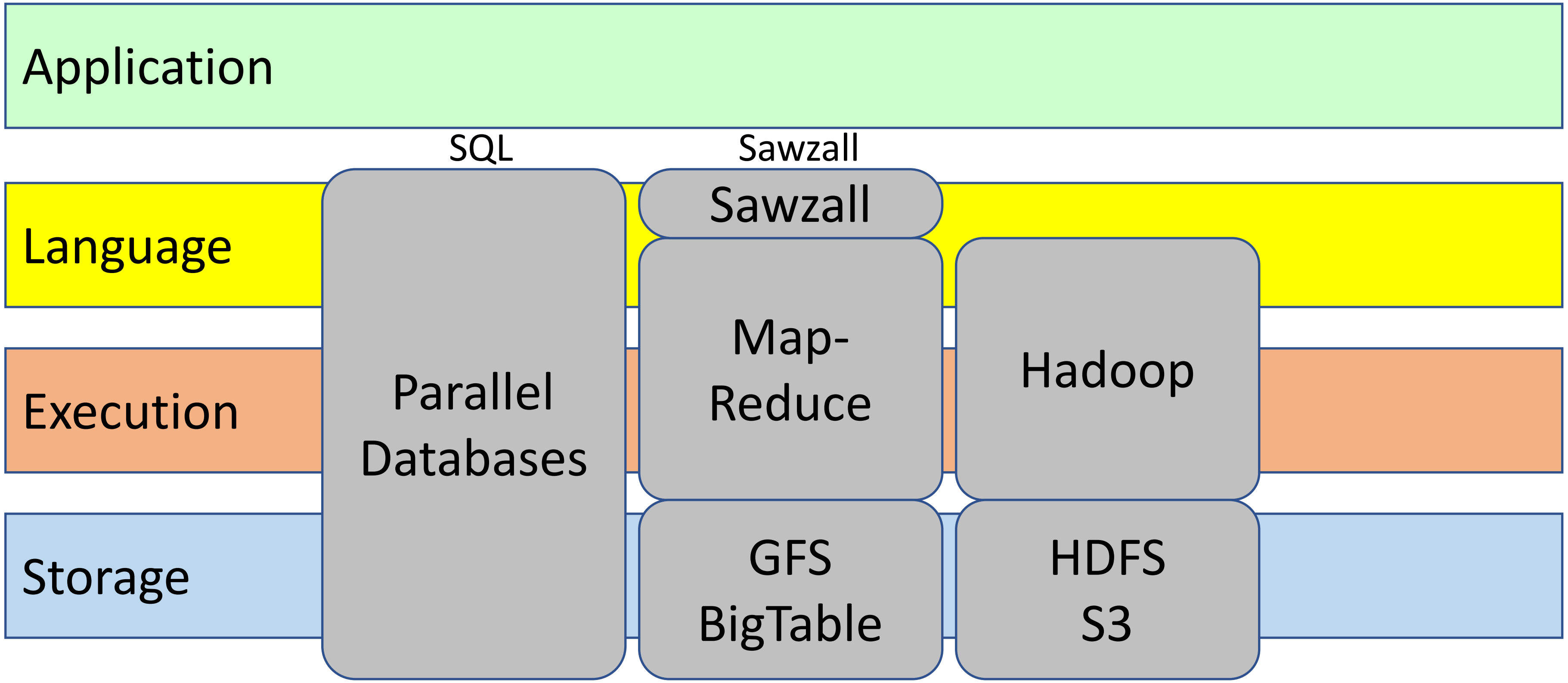



```
count: table sum of int;
total: table sum of float;
sum_of_squares: table sum of float;
x: float = input;
emit count <- 1;
emit total <- x;
emit sum_of_squares <- x * x;
```

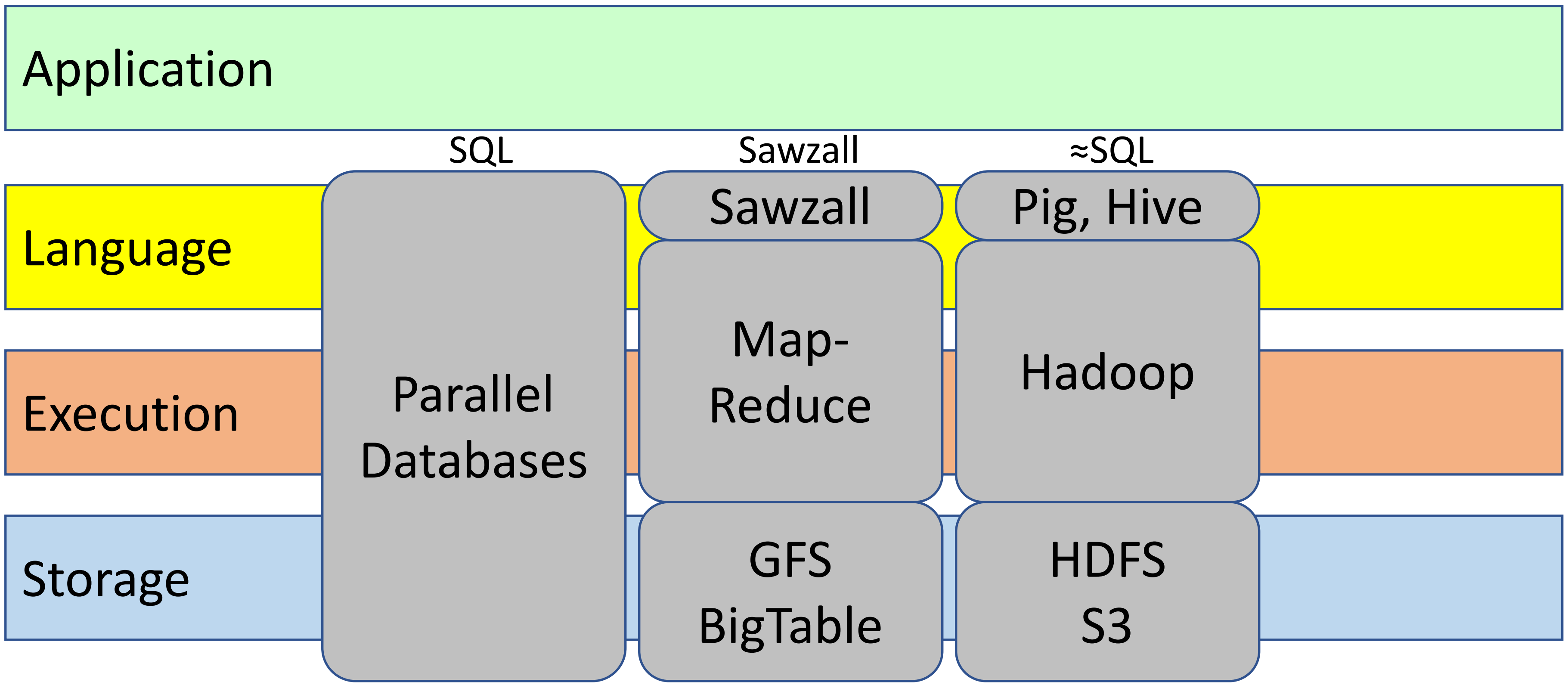
Data-Parallel Computation Systems



Data-Parallel Computation Systems

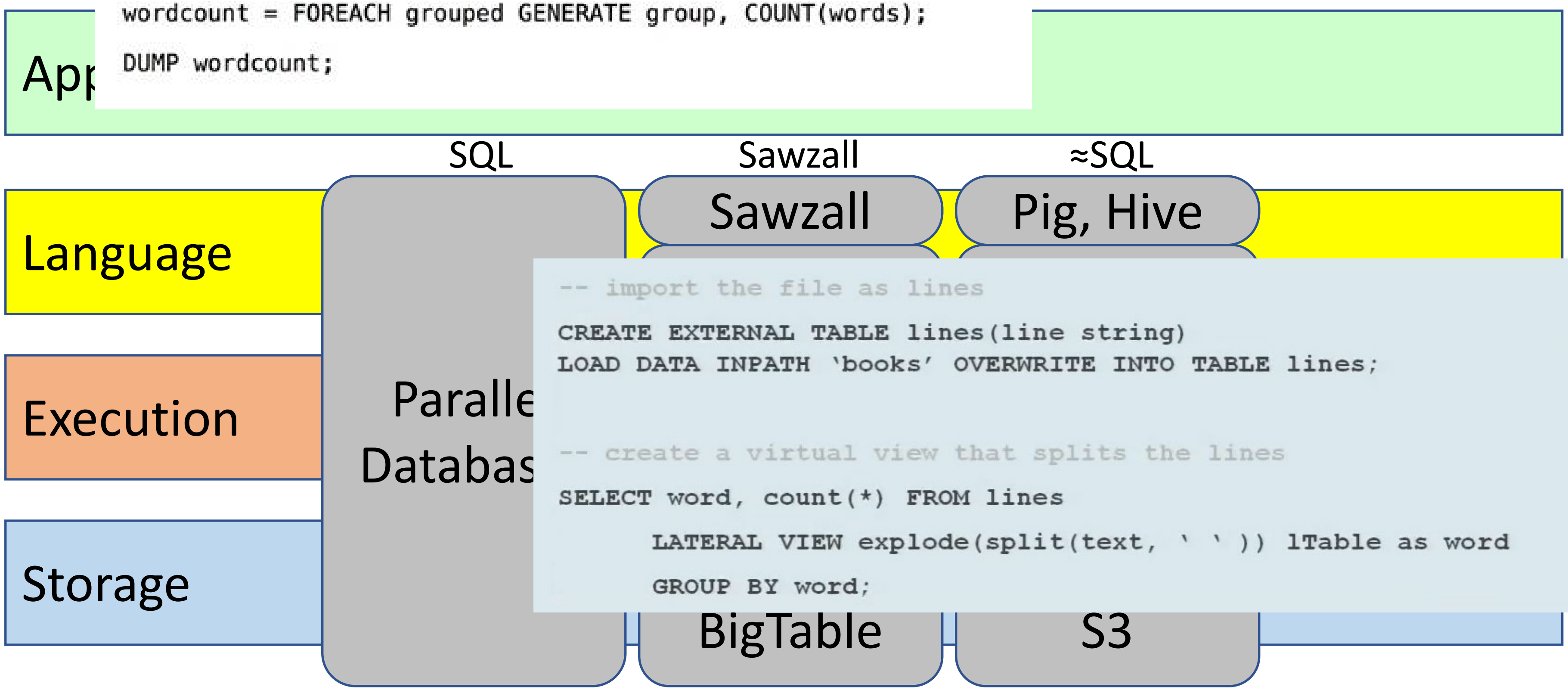


Data-Parallel Computation Systems



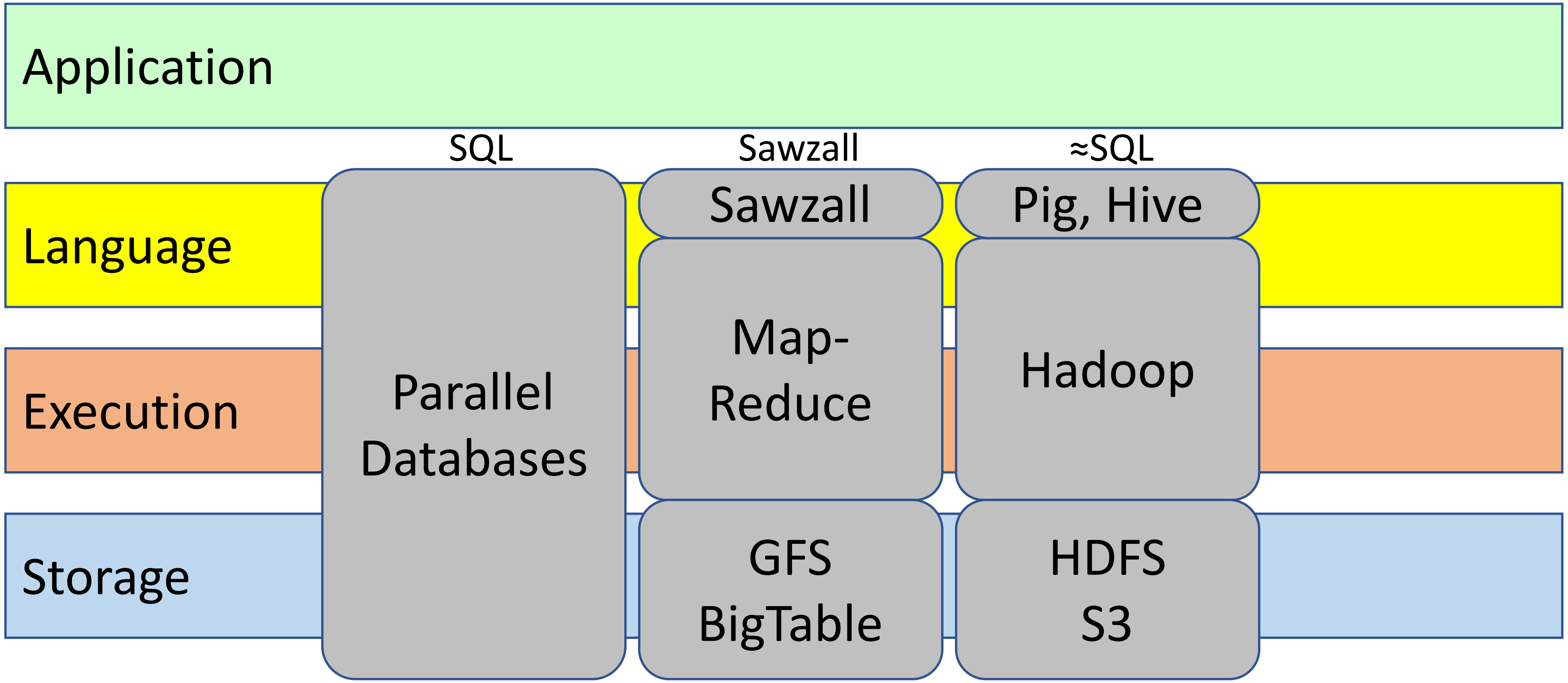
Systems

```
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;
```

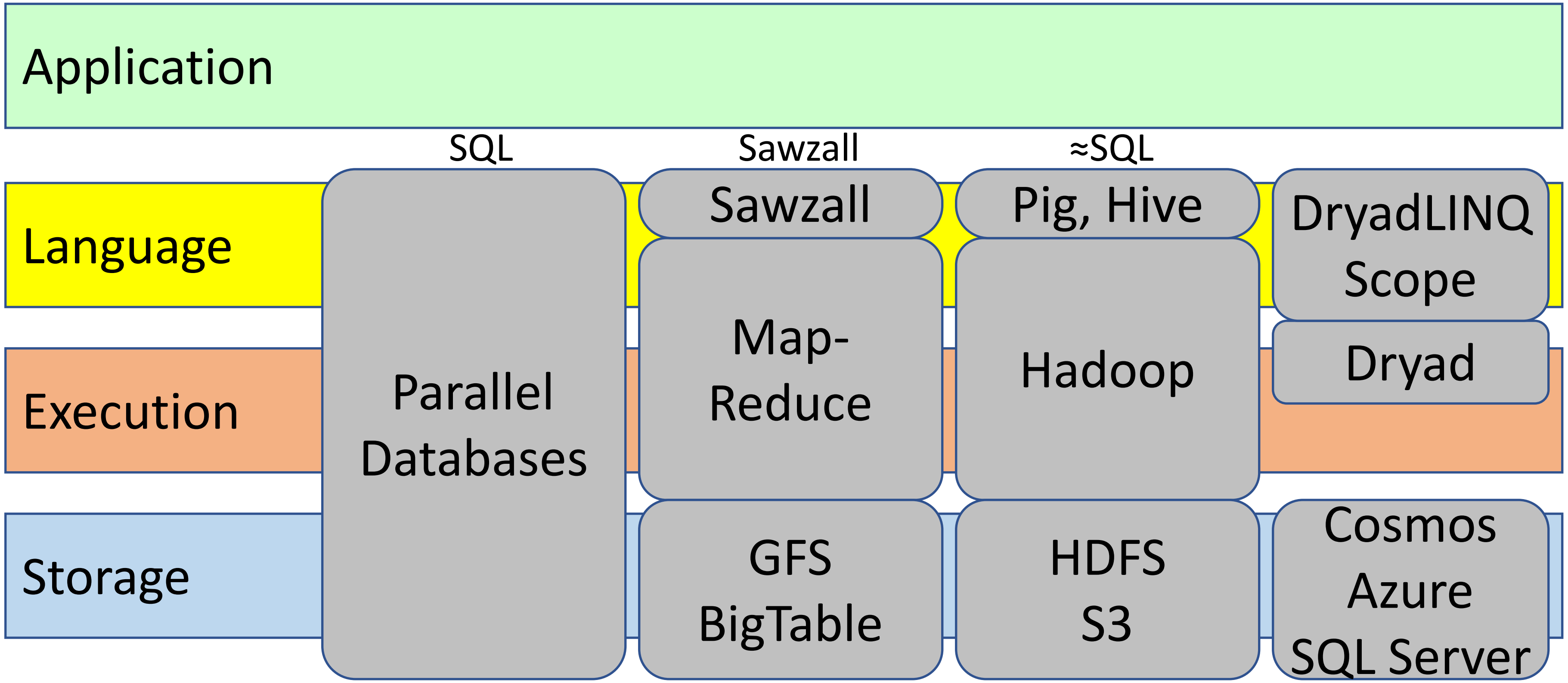



```
-- import the file as lines
CREATE EXTERNAL TABLE lines(line string)
LOAD DATA INPATH 'books' OVERWRITE INTO TABLE lines;

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

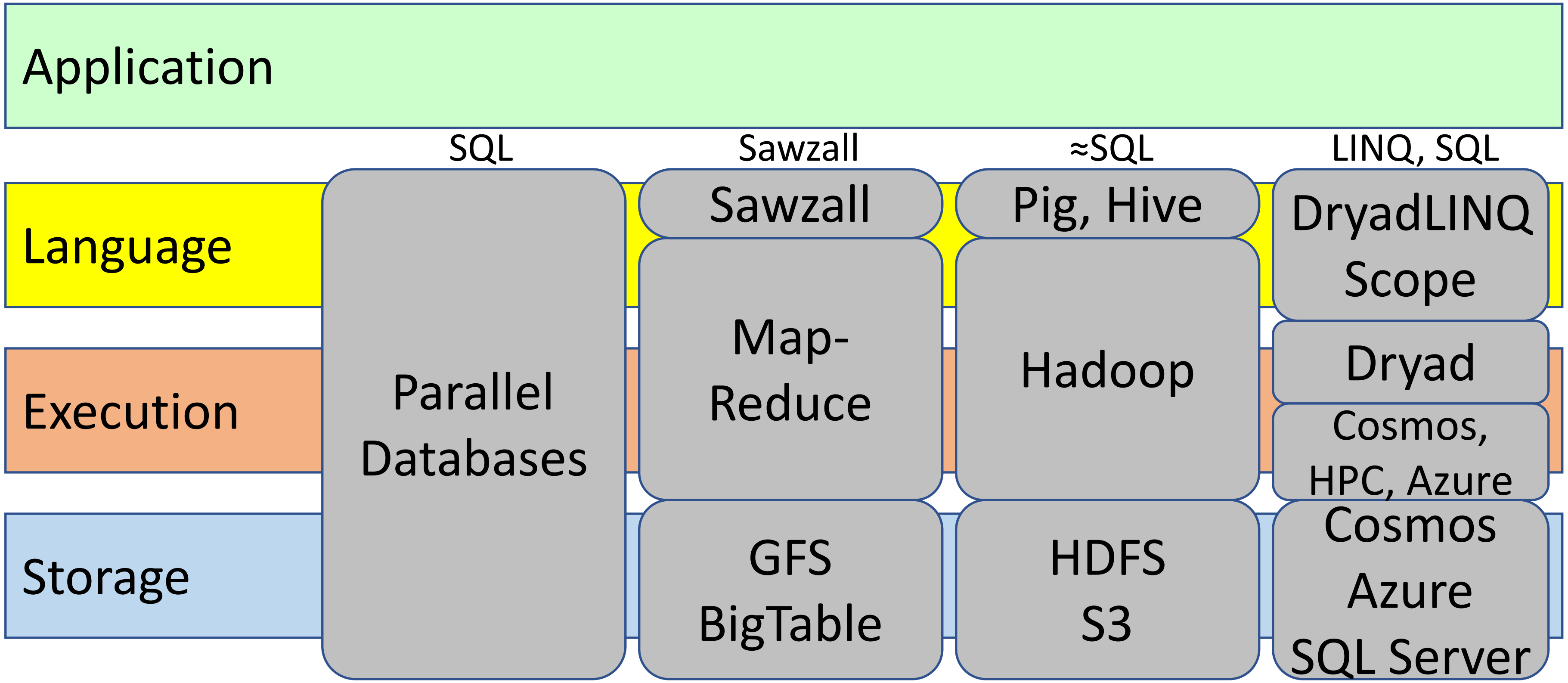
Data-Parallel Computation Systems



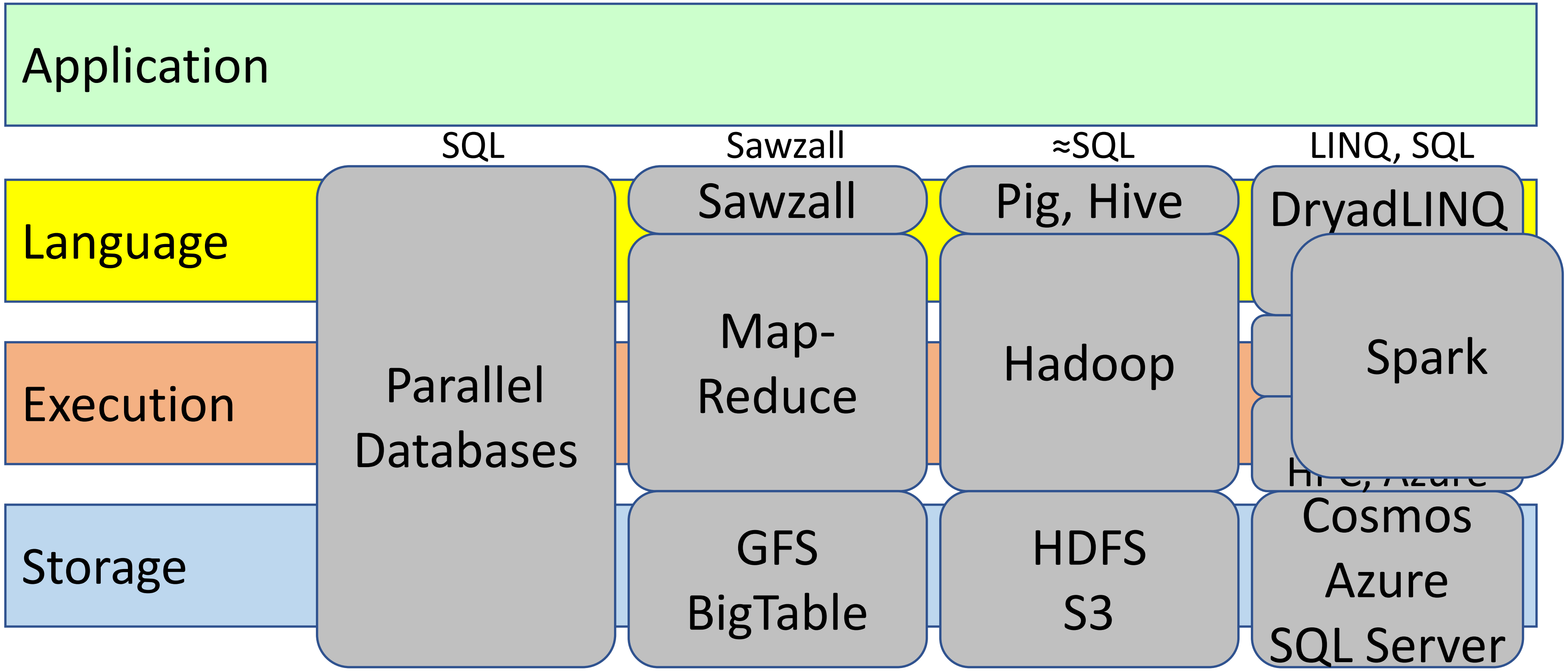
Data-Parallel Computation Systems



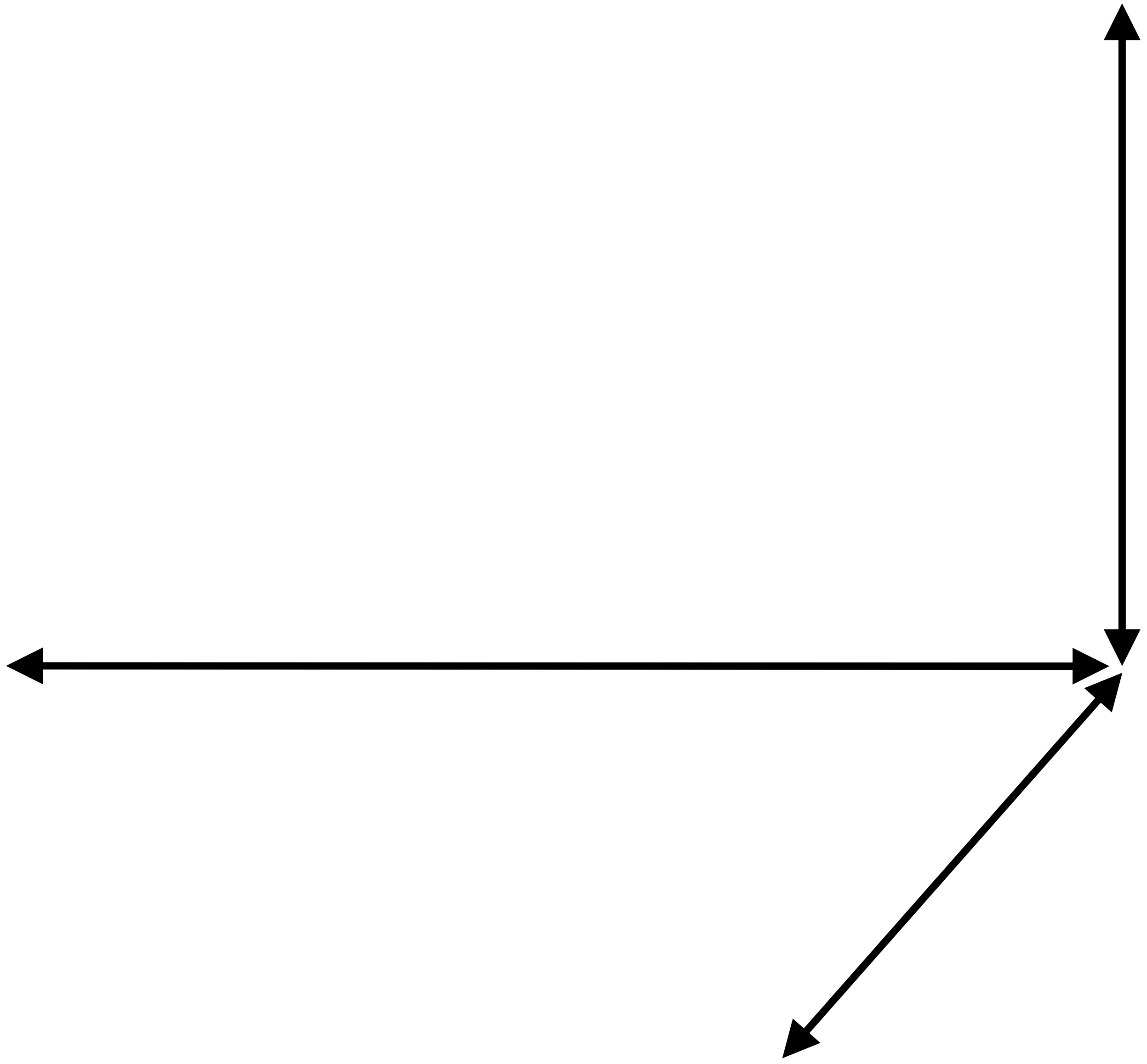
Data-Parallel Computation Systems



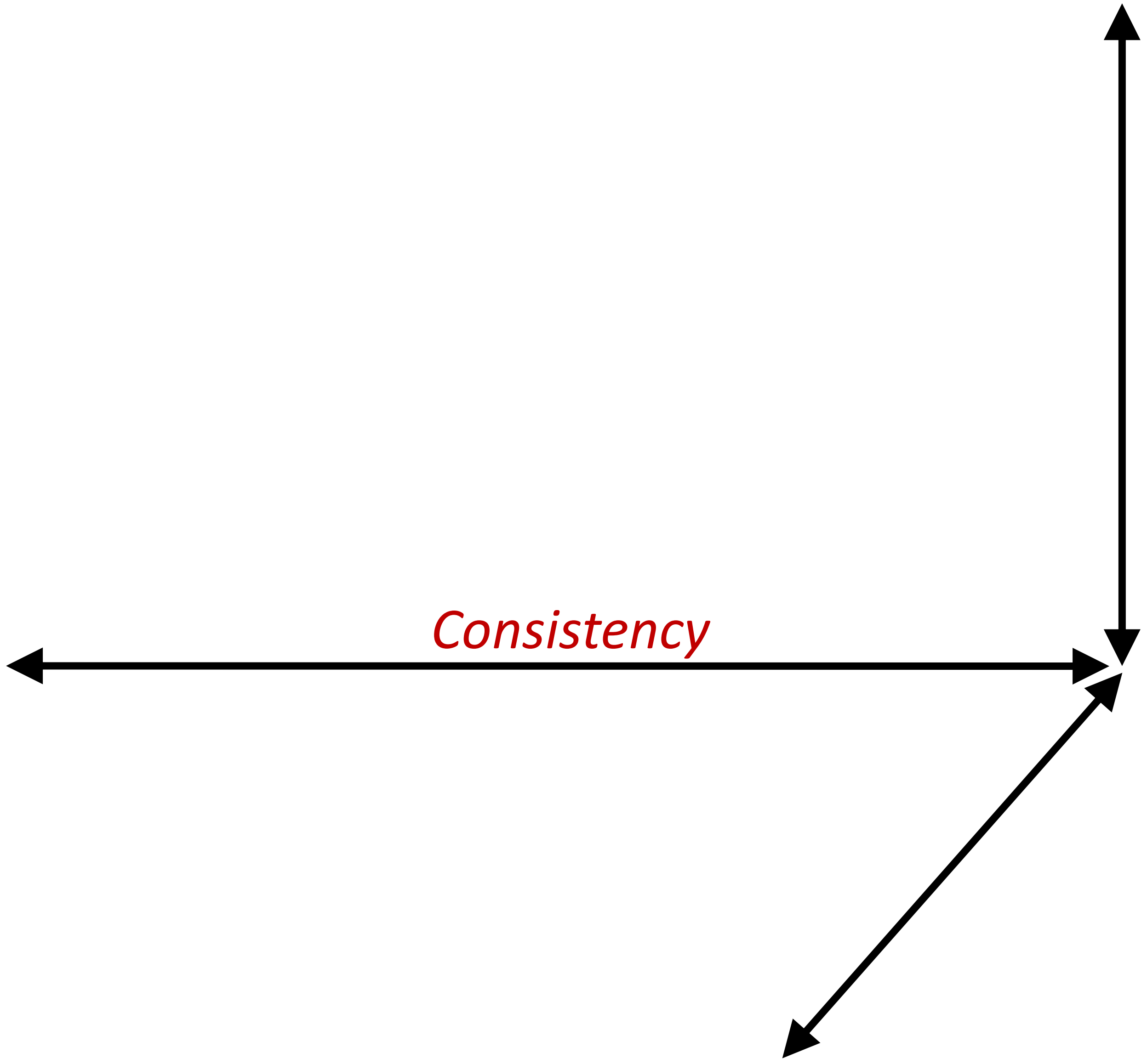
Data-Parallel Computation Systems



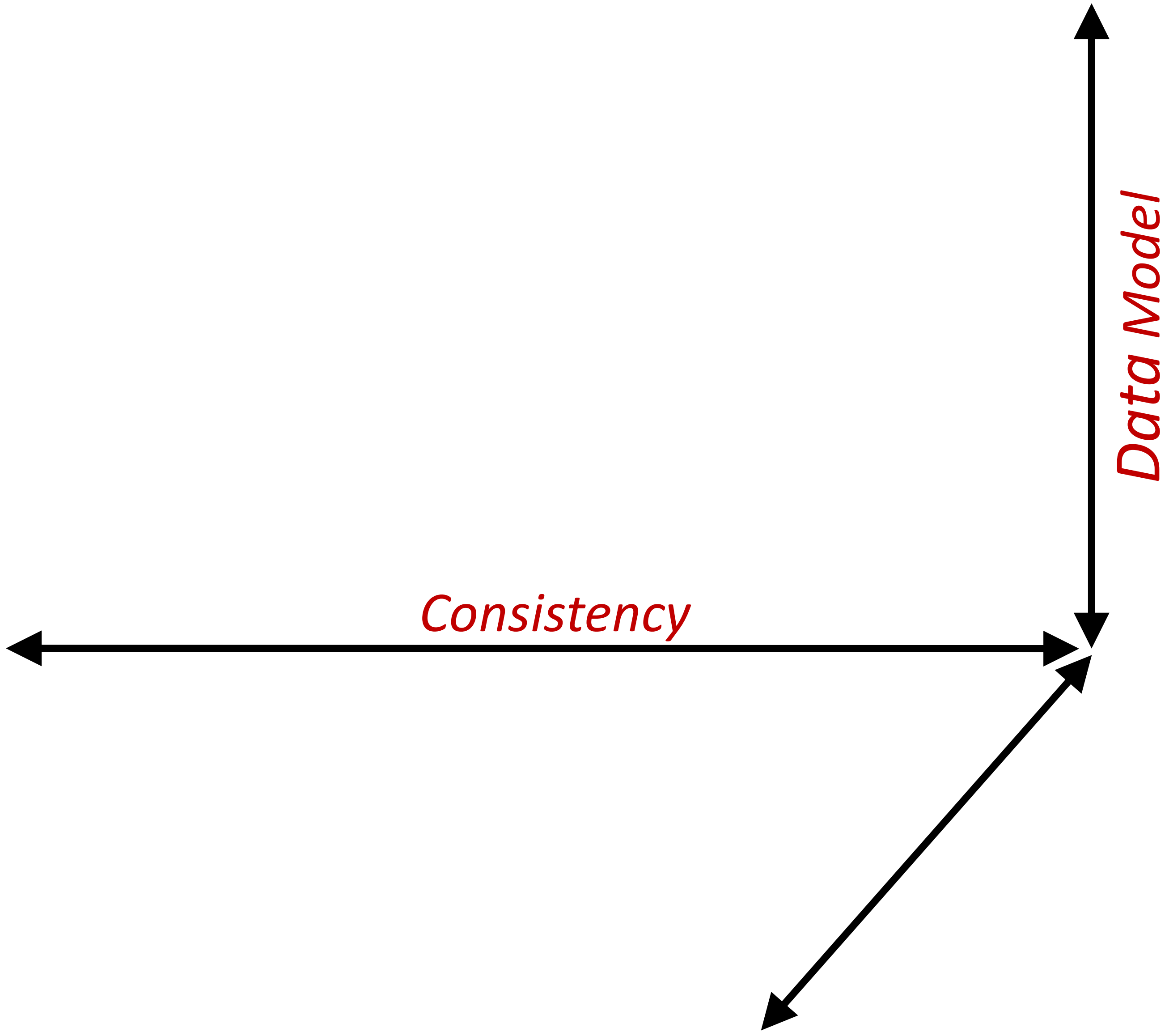
(Yet) Another Framework



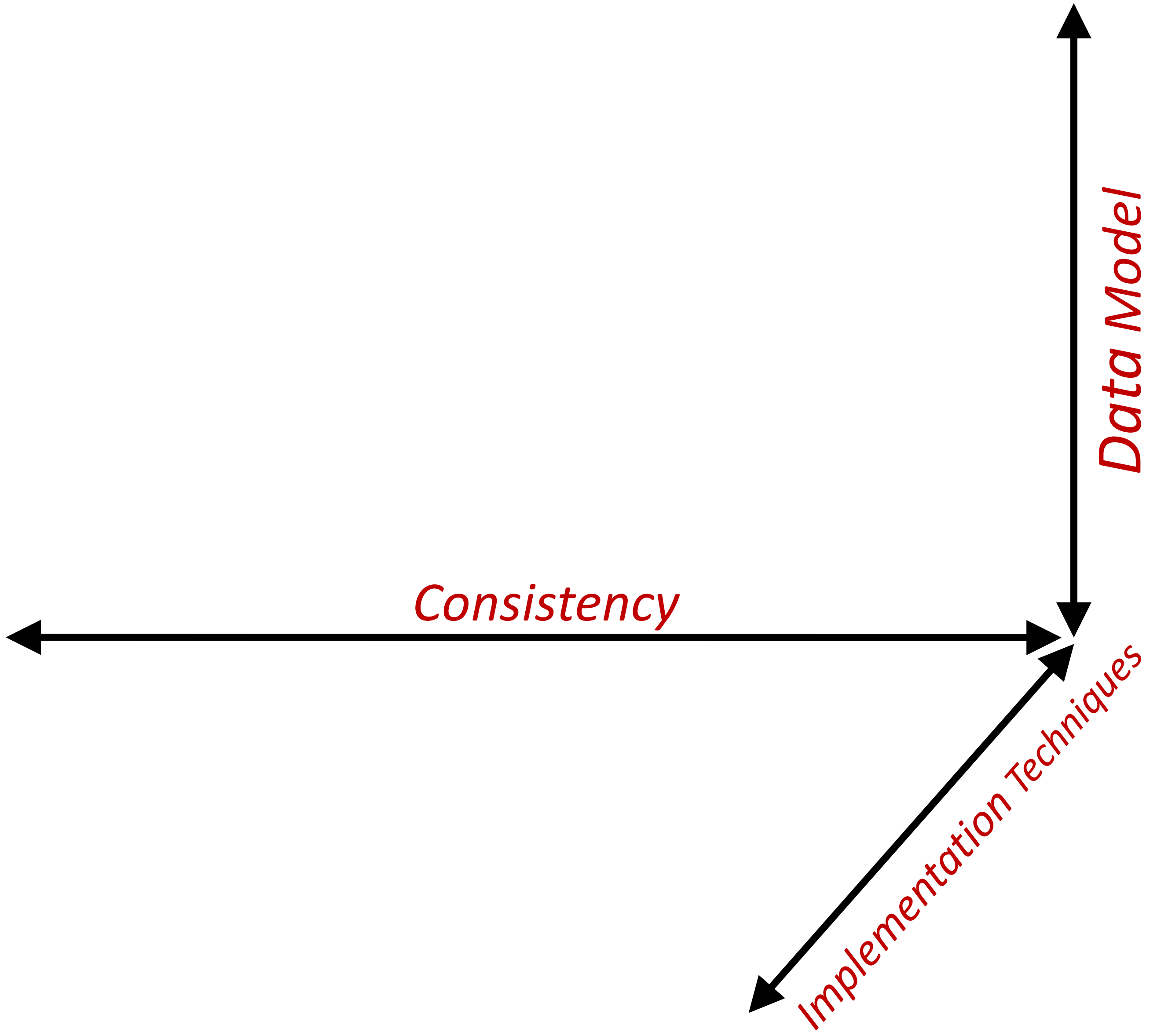
(Yet) Another Framework



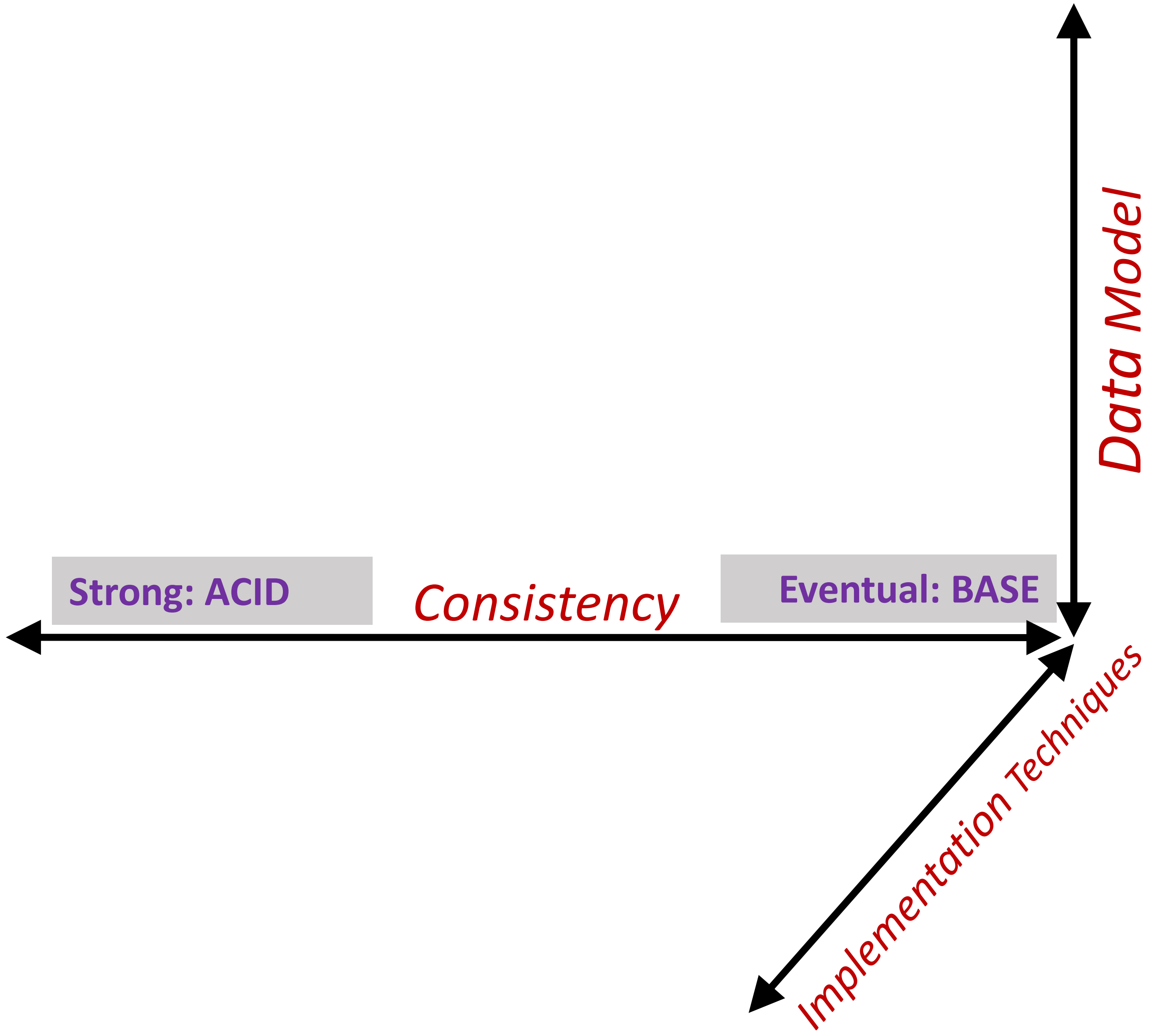
(Yet) Another Framework



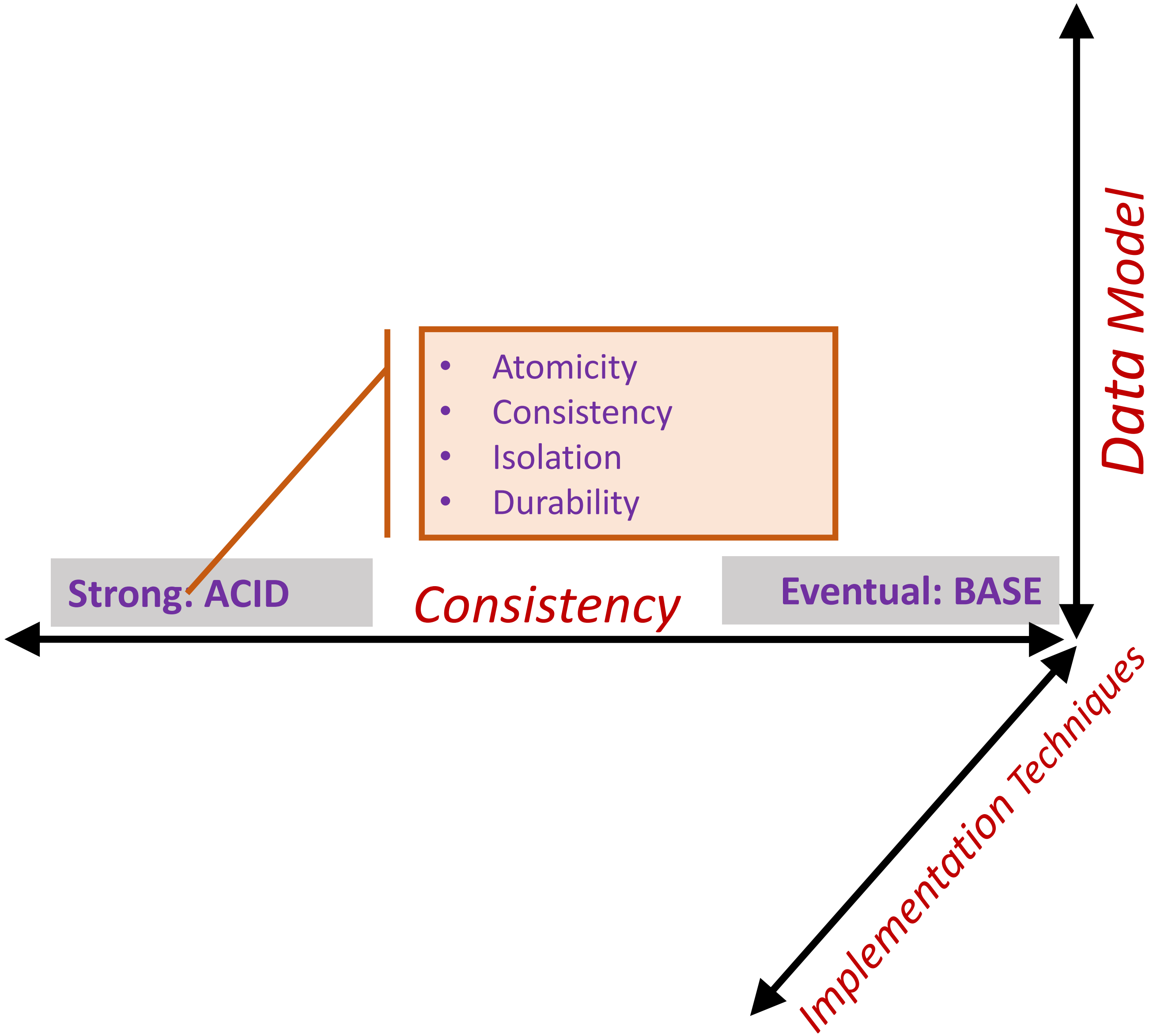
(Yet) Another Framework



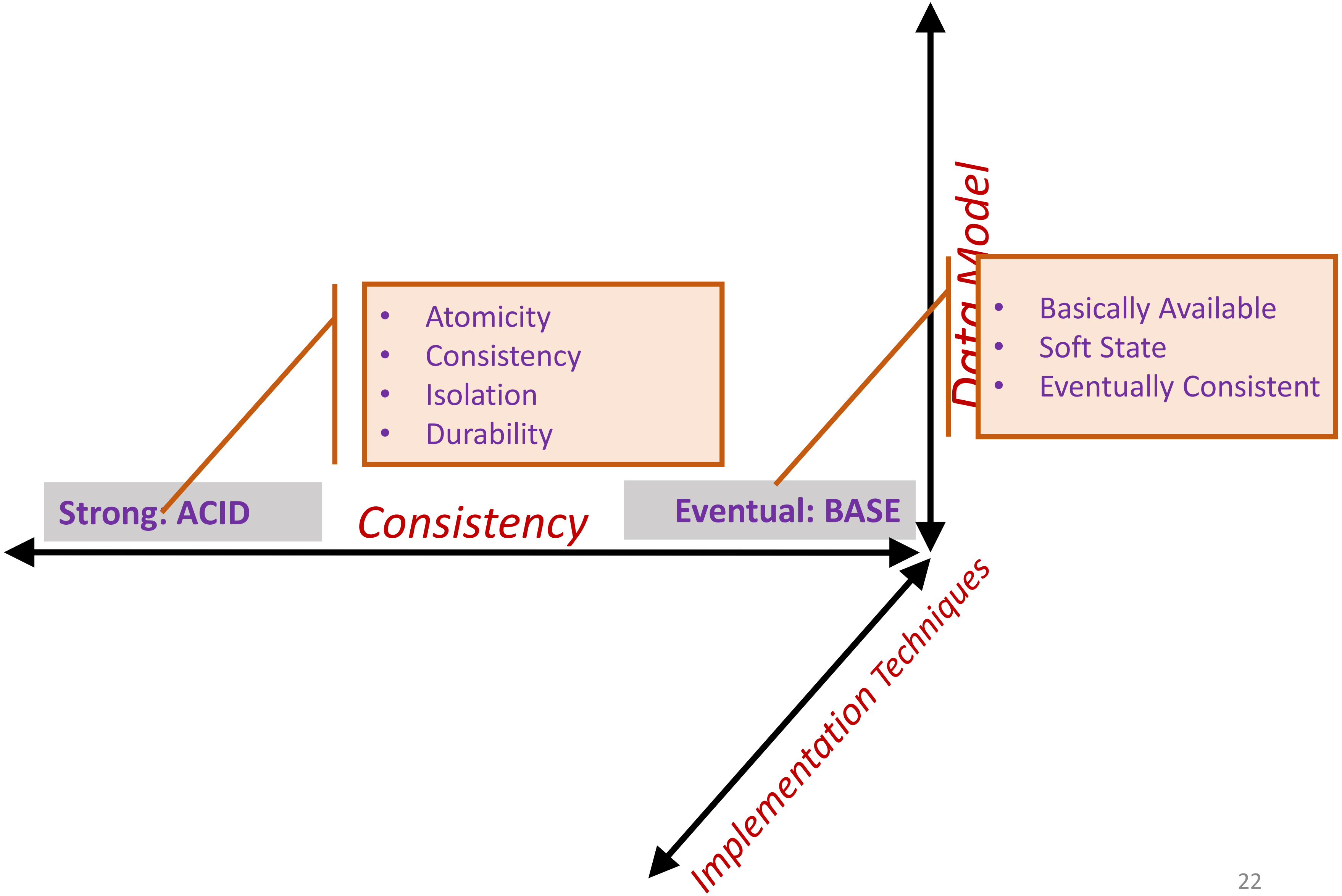
(Yet) Another Framework



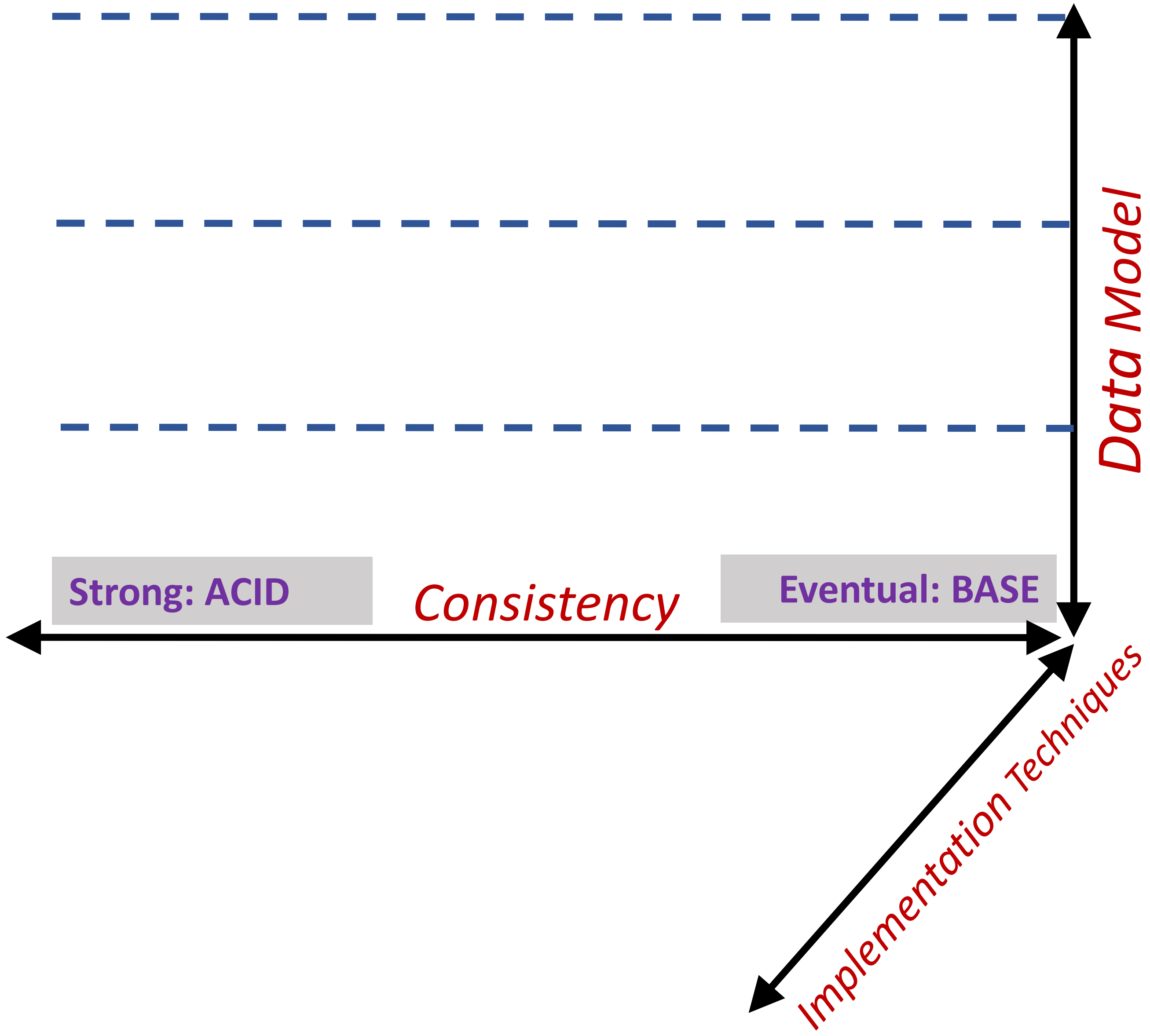
(Yet) Another Framework



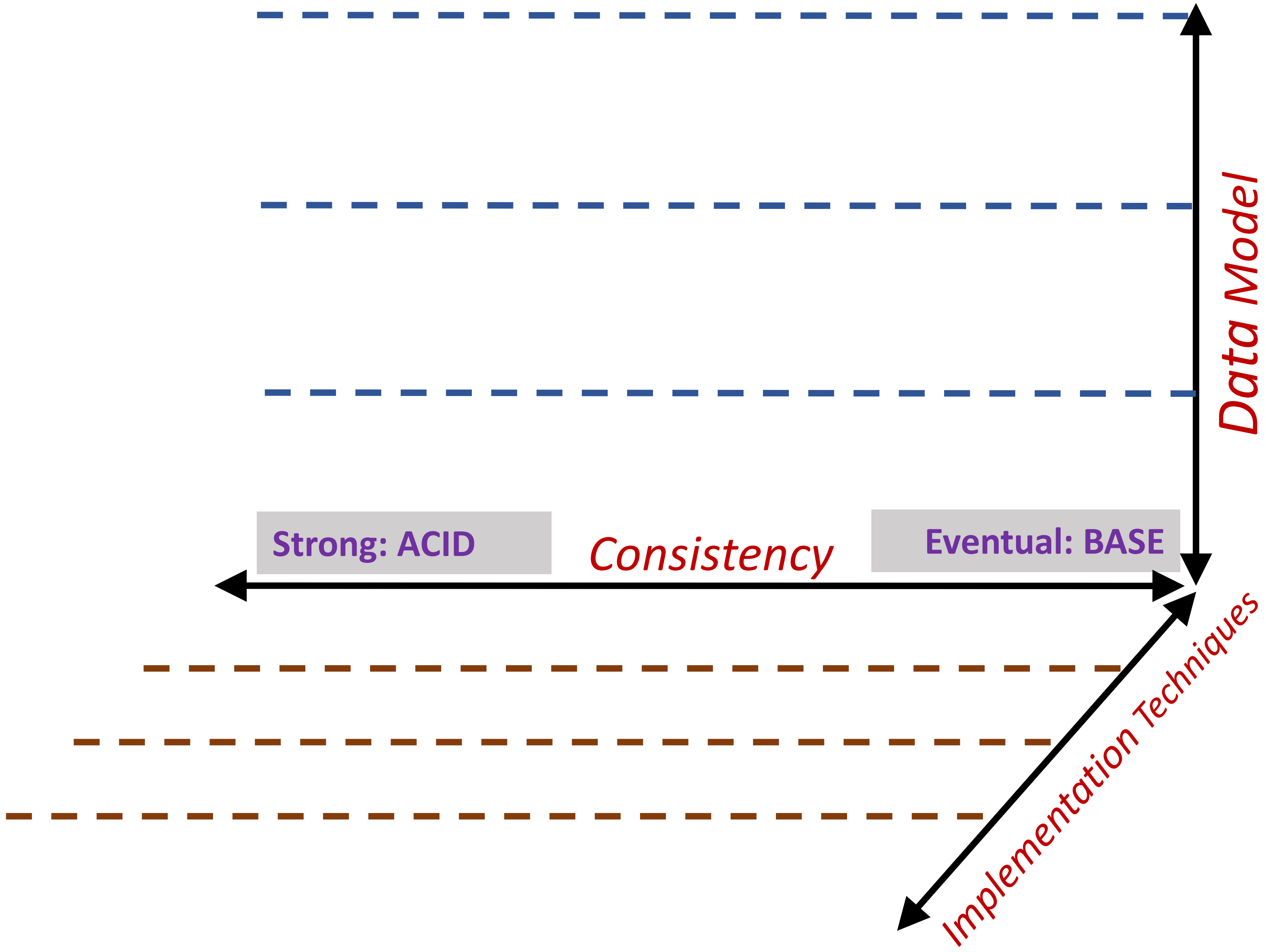
(Yet) Another Framework



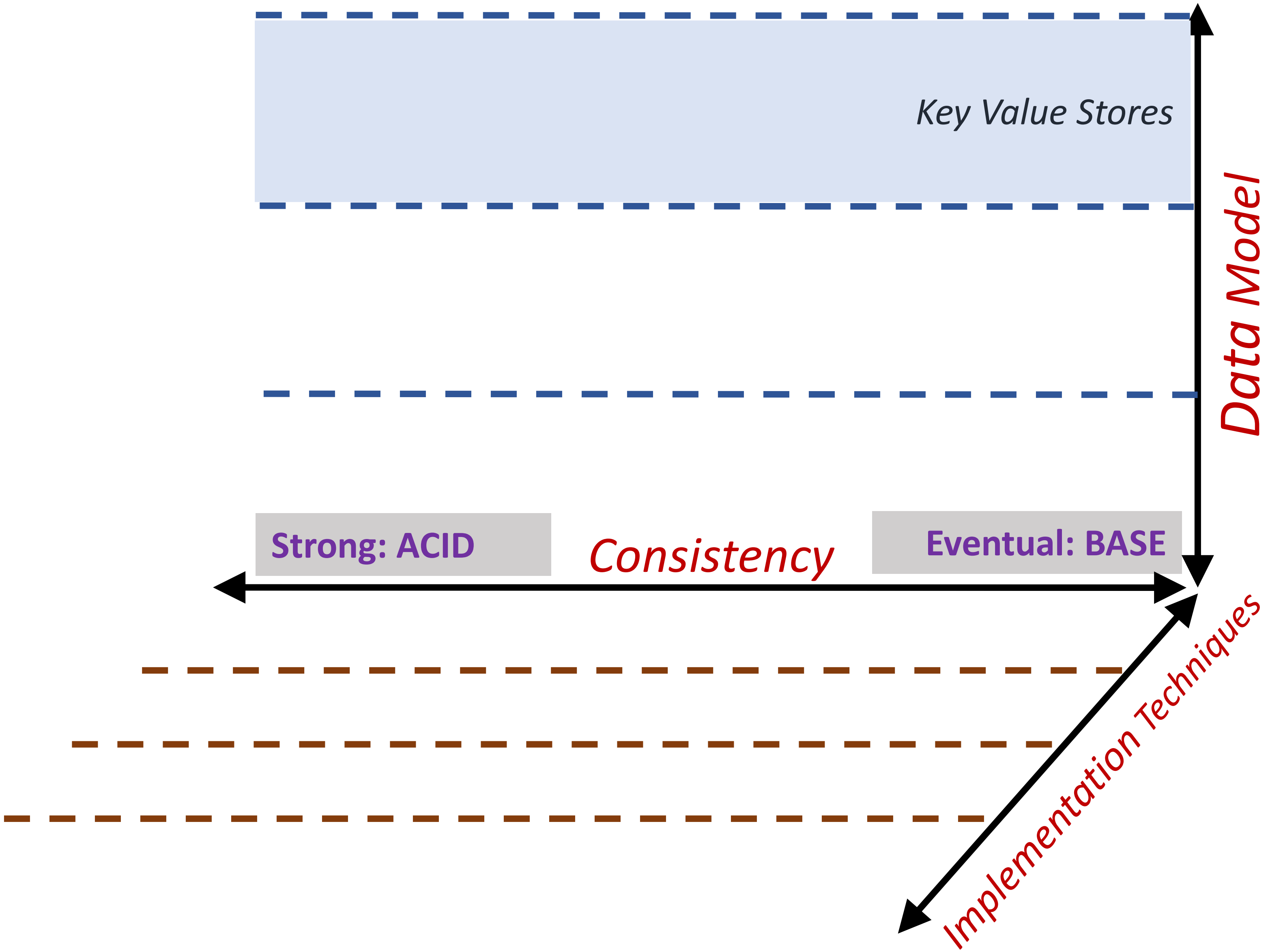
(Yet) Another Framework



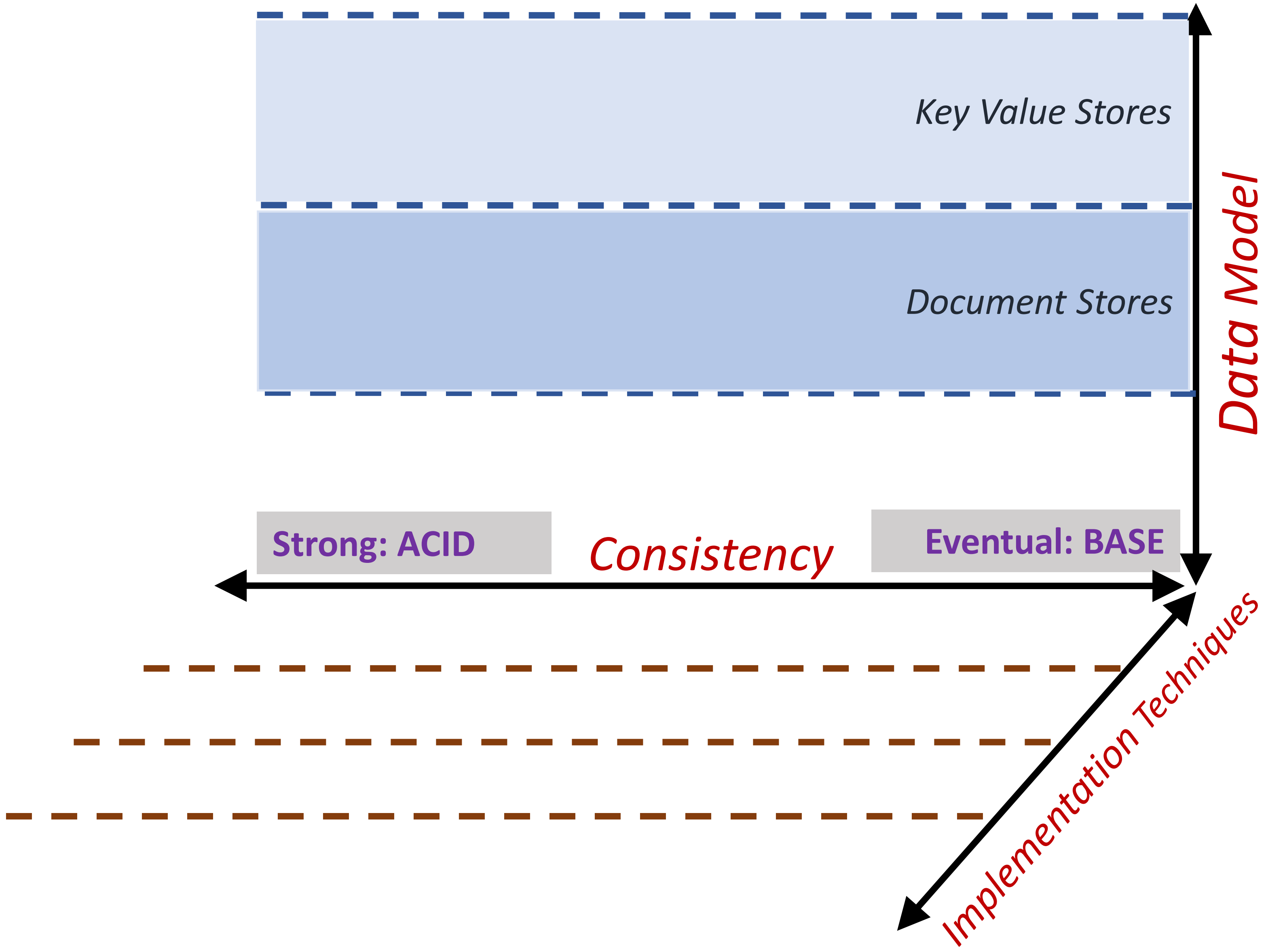
(Yet) Another Framework



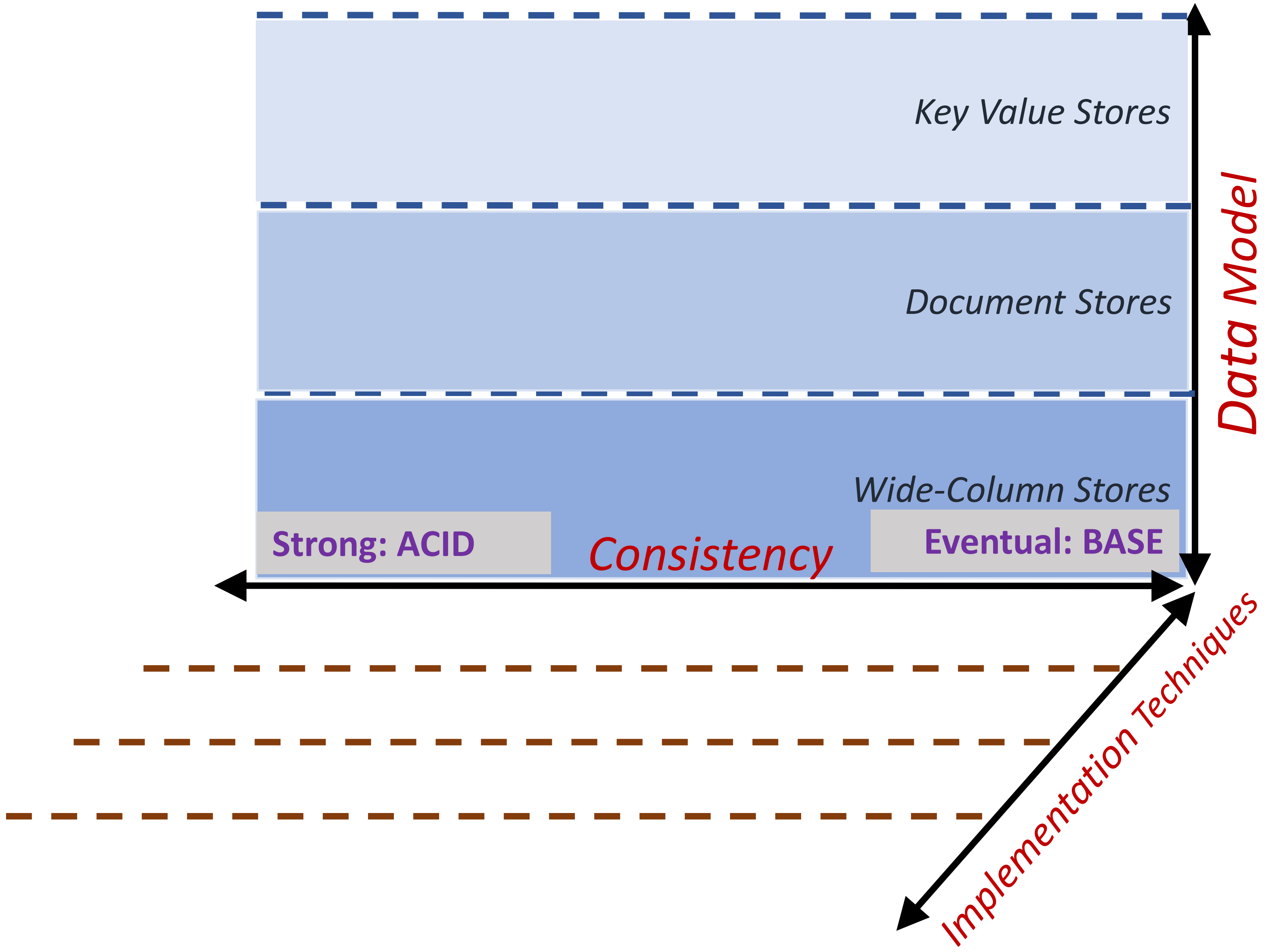
(Yet) Another Framework



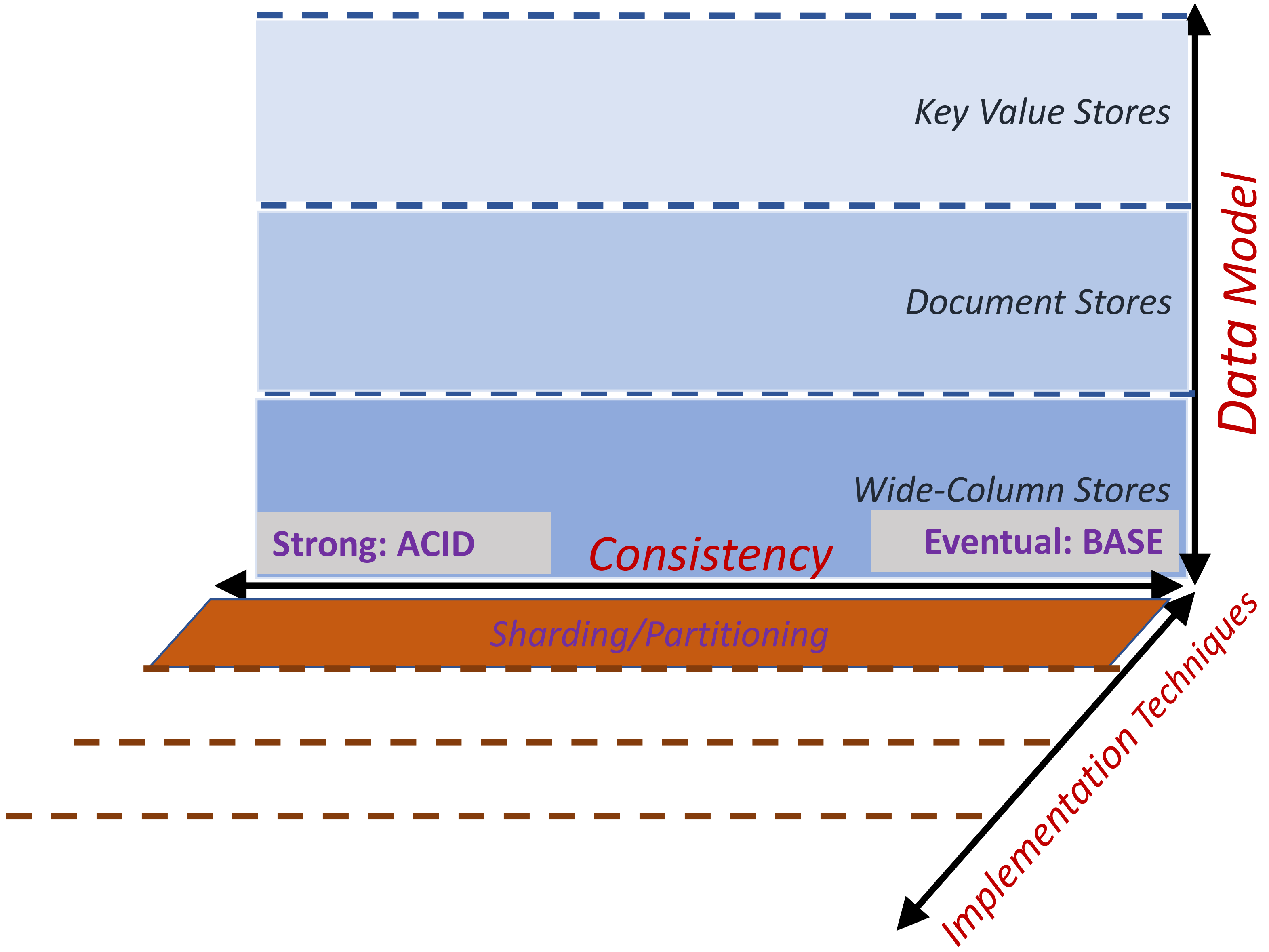
(Yet) Another Framework



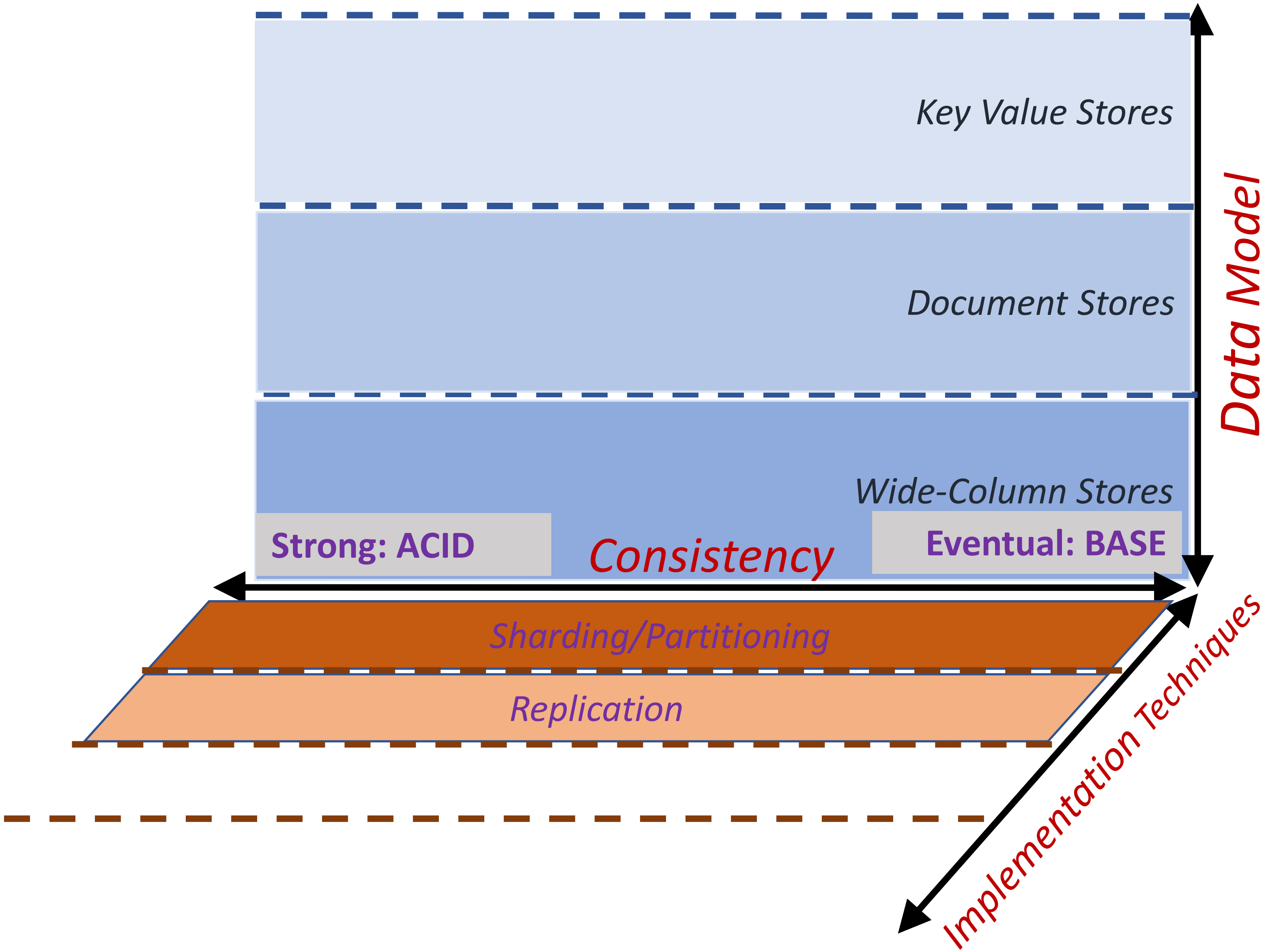
(Yet) Another Framework



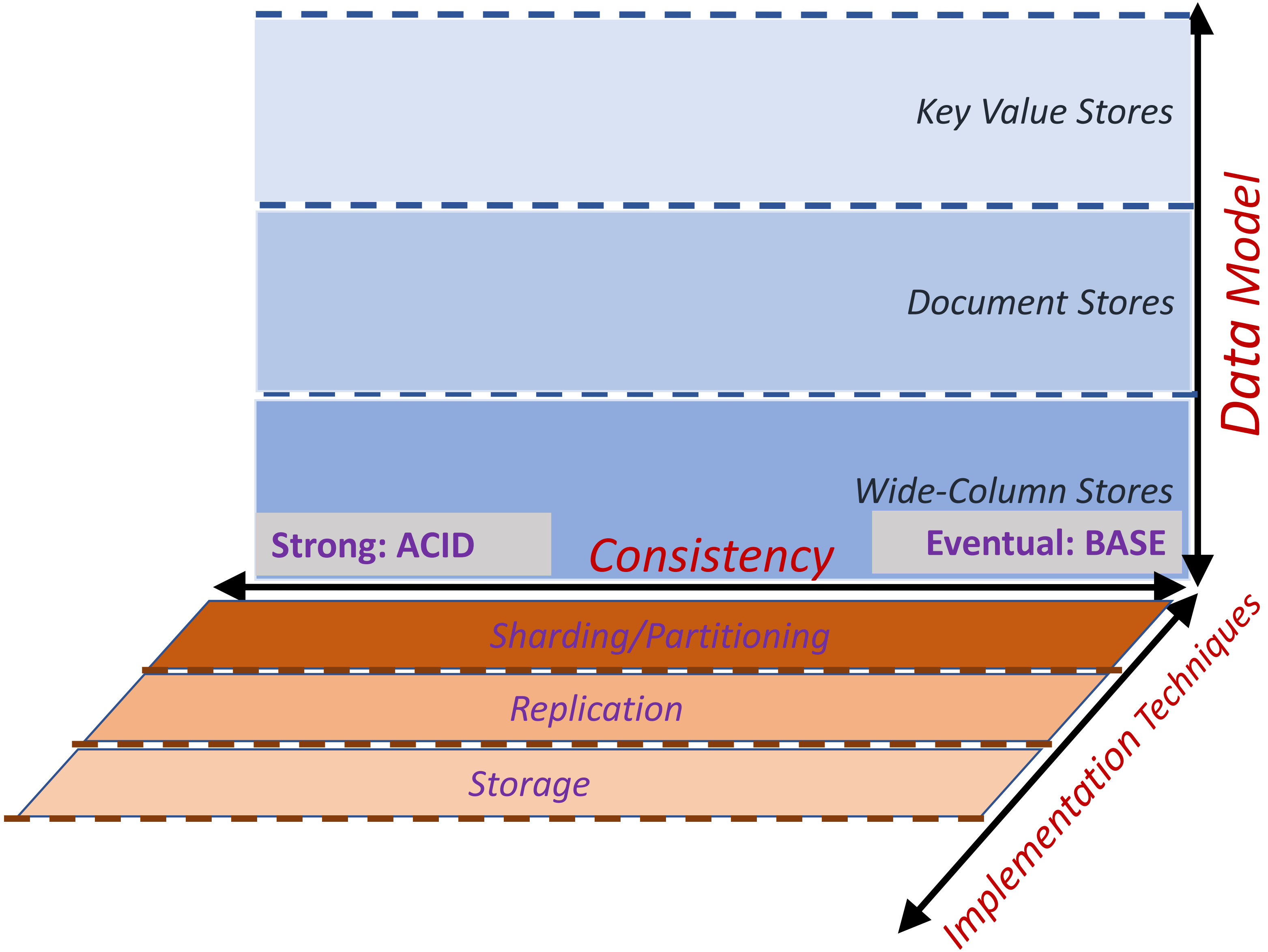
(Yet) Another Framework



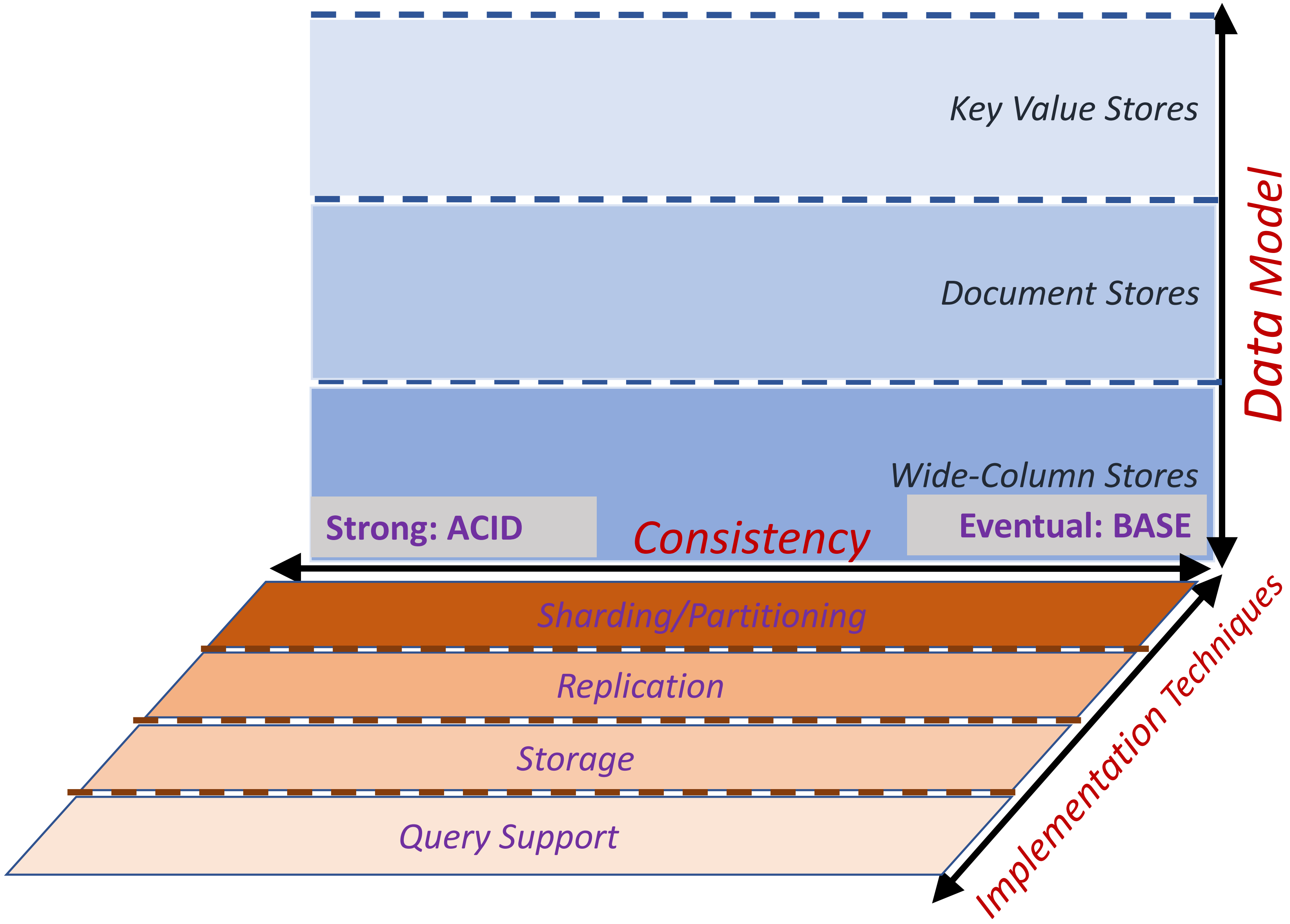
(Yet) Another Framework



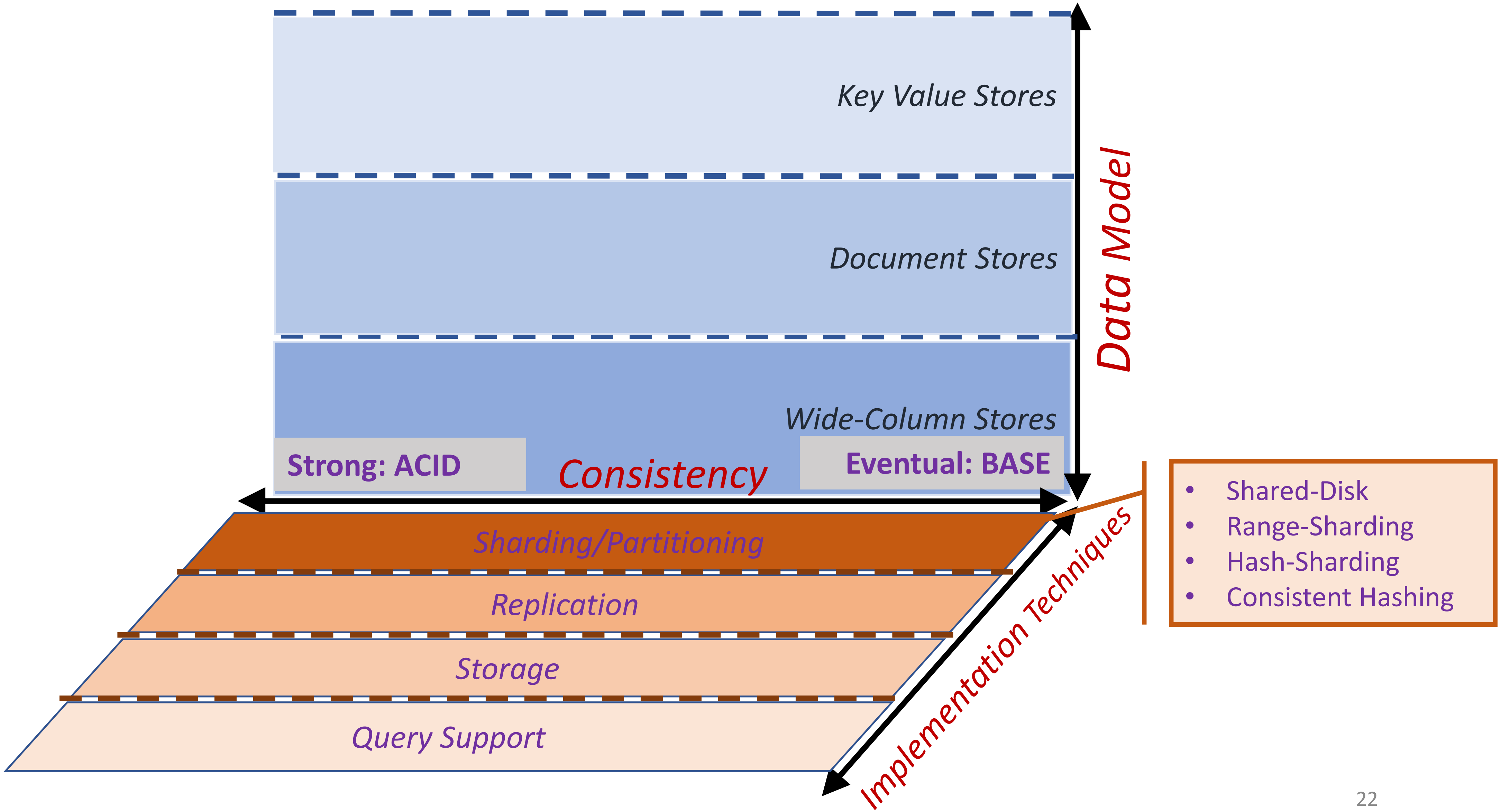
(Yet) Another Framework



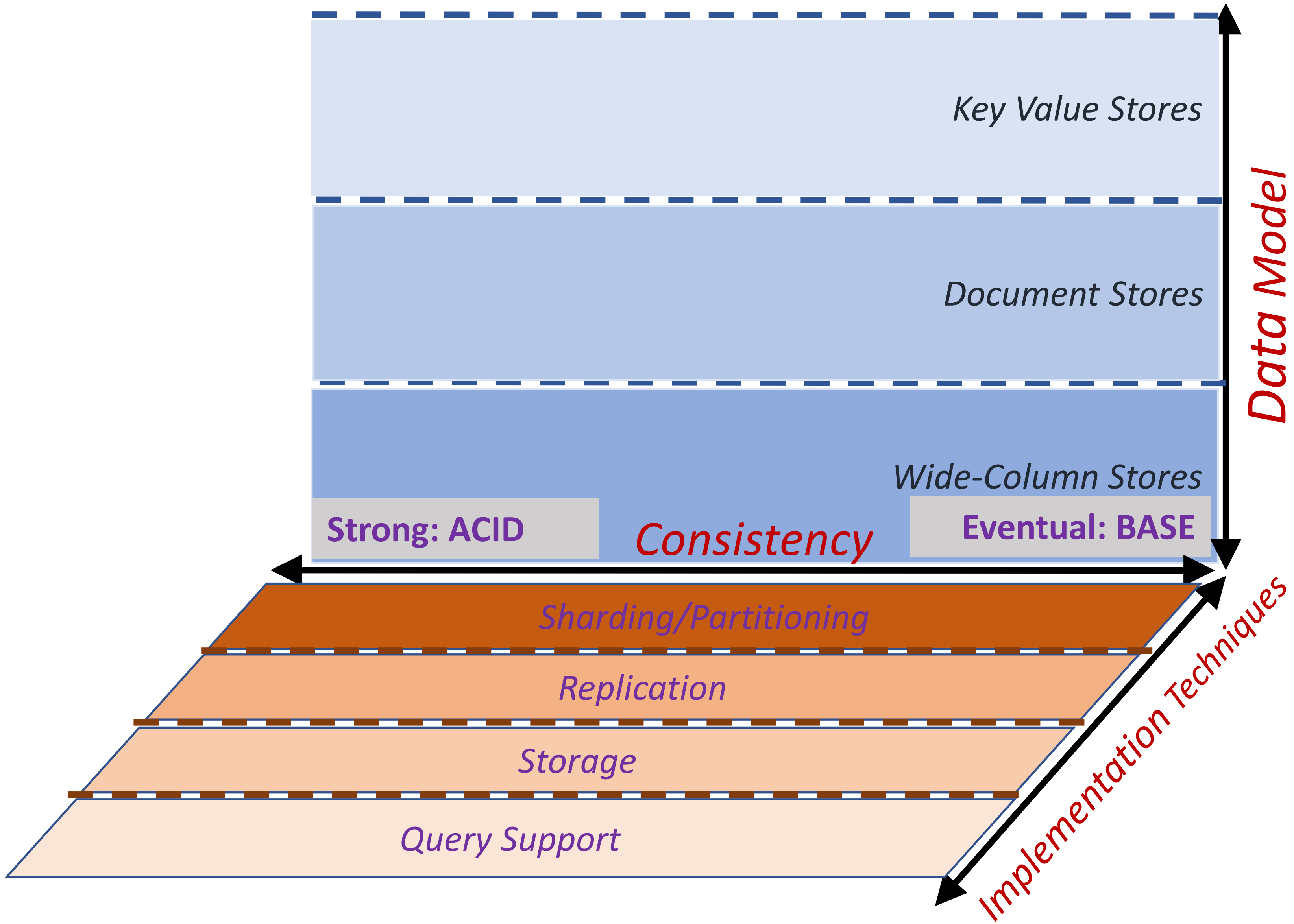
(Yet) Another Framework



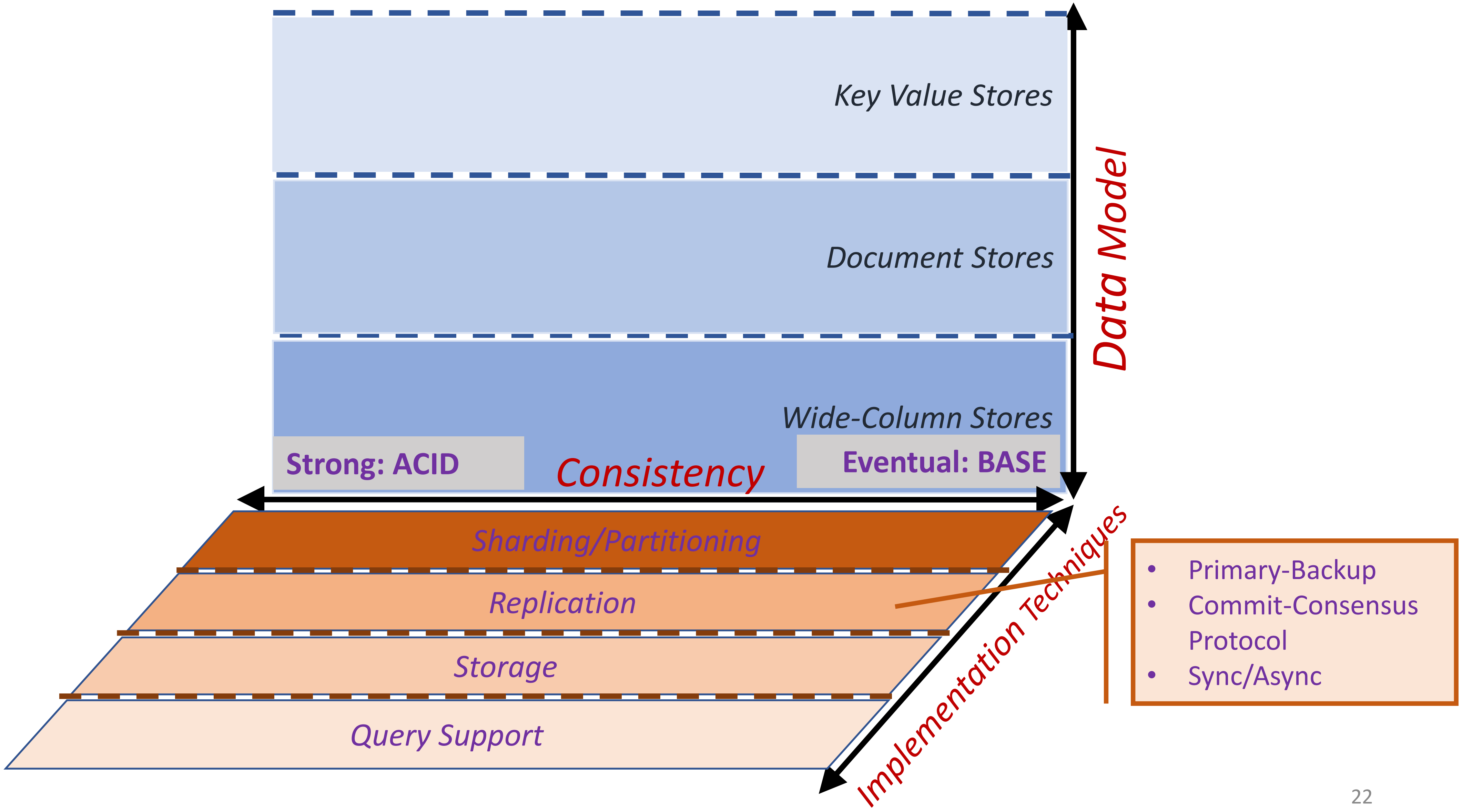
(Yet) Another Framework



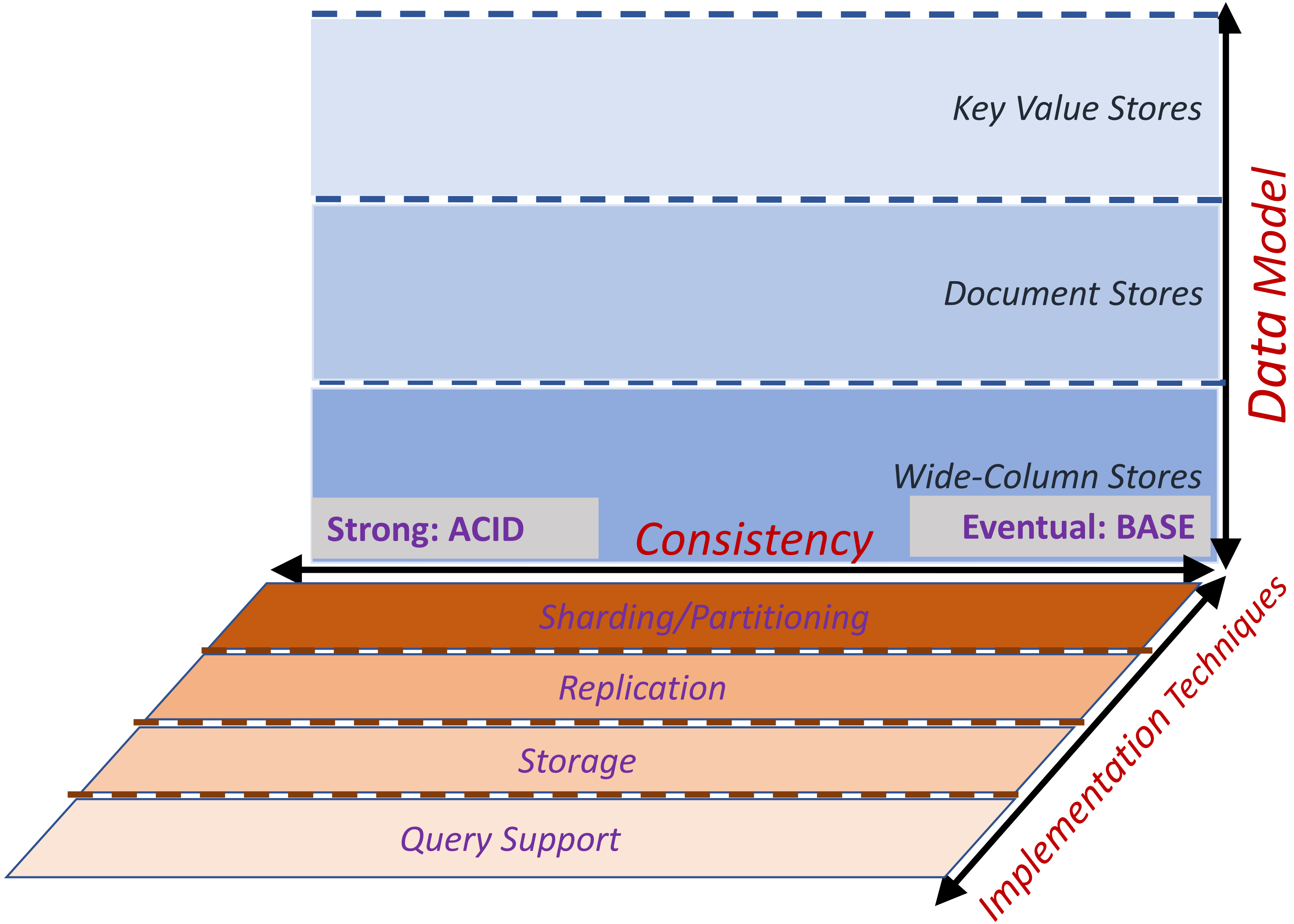
(Yet) Another Framework



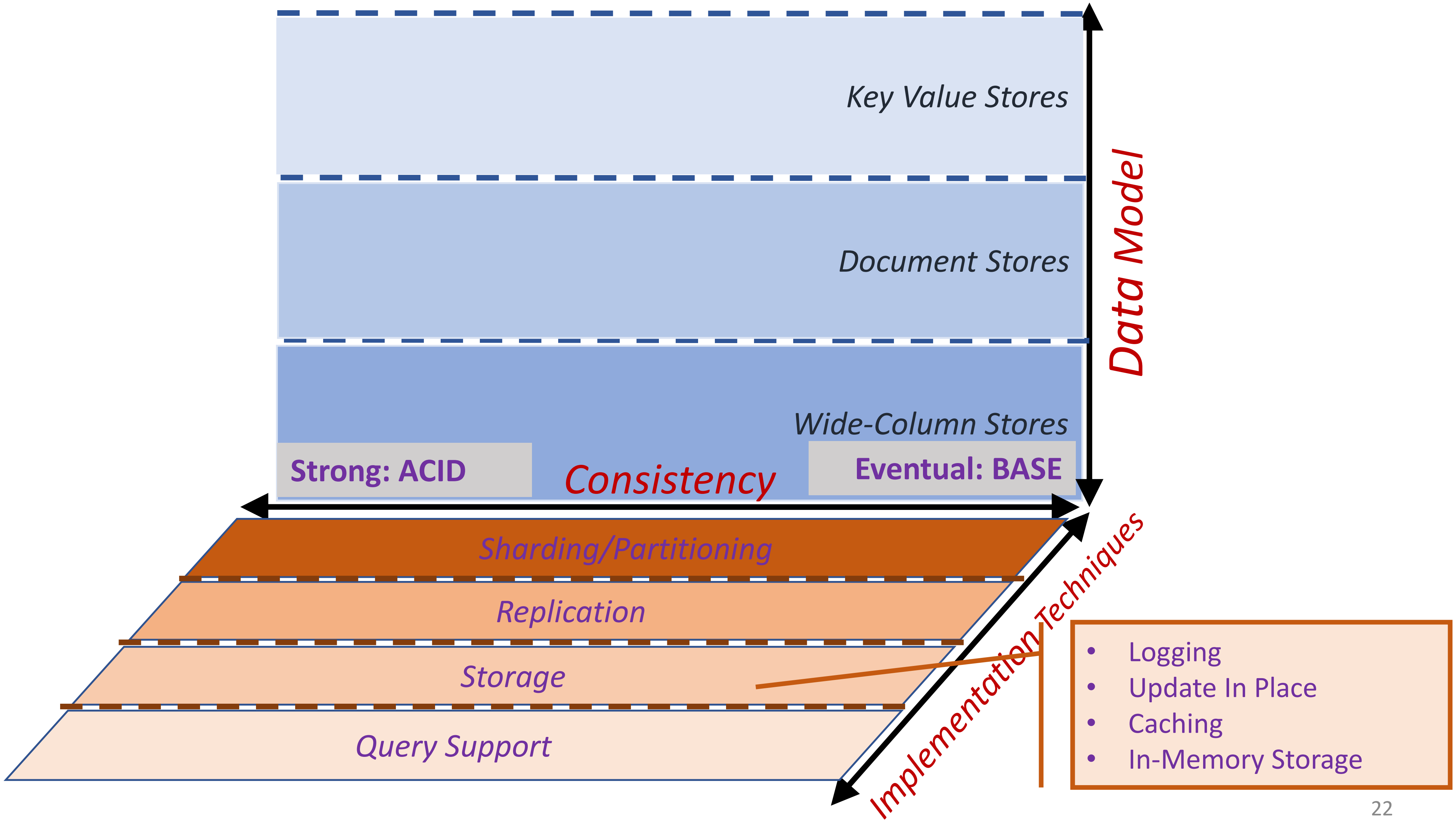
(Yet) Another Framework



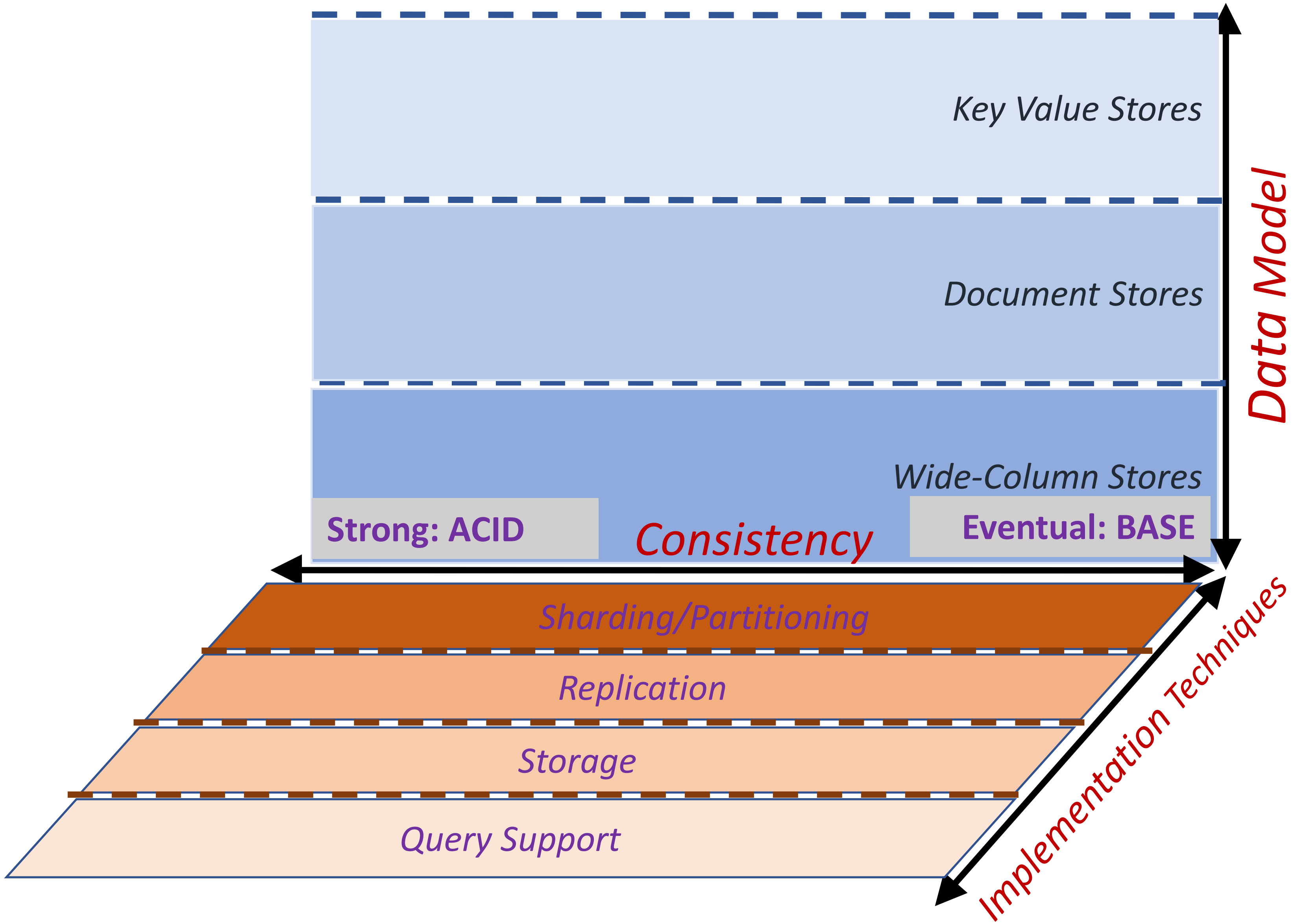
(Yet) Another Framework



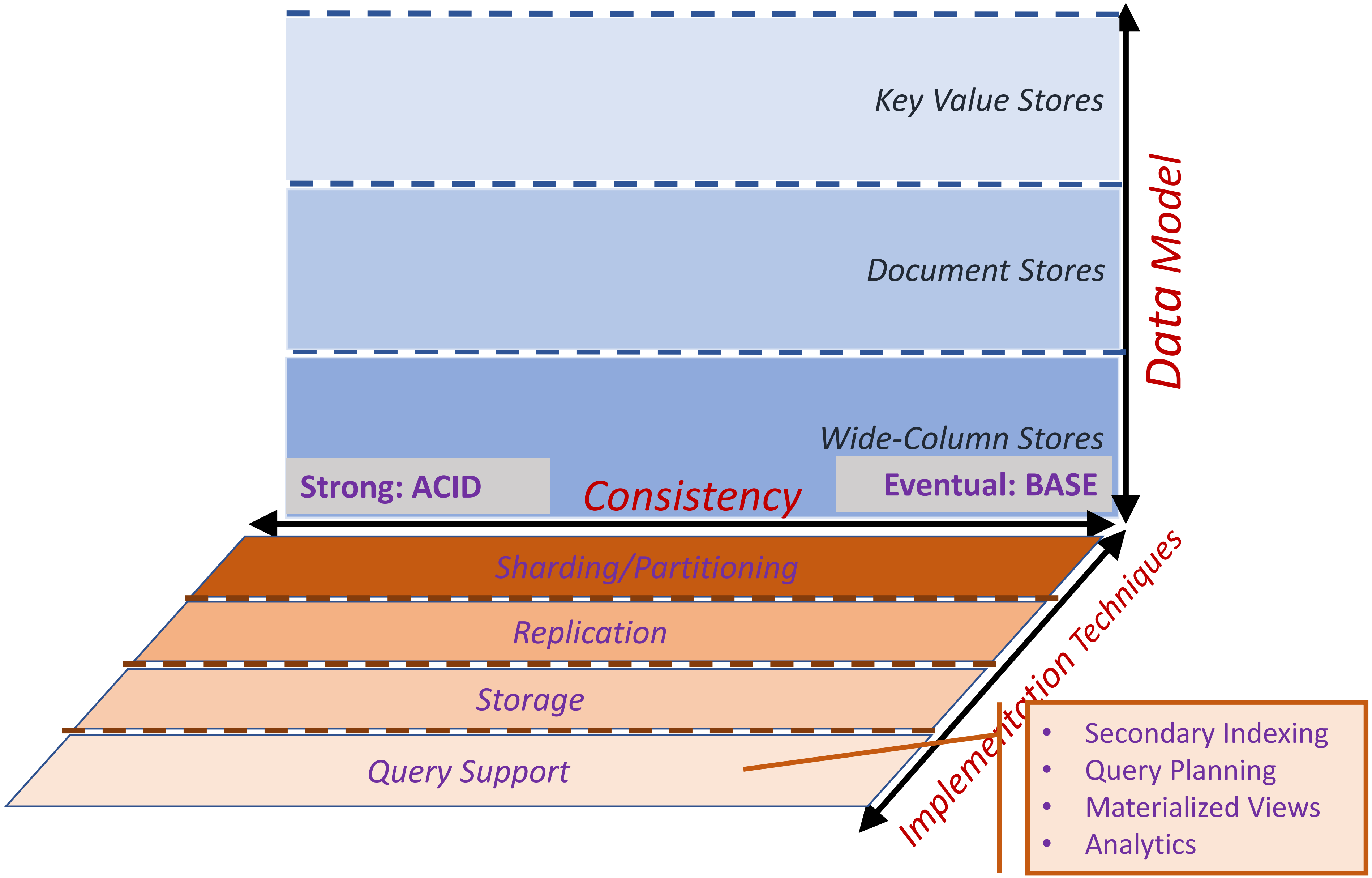
(Yet) Another Framework



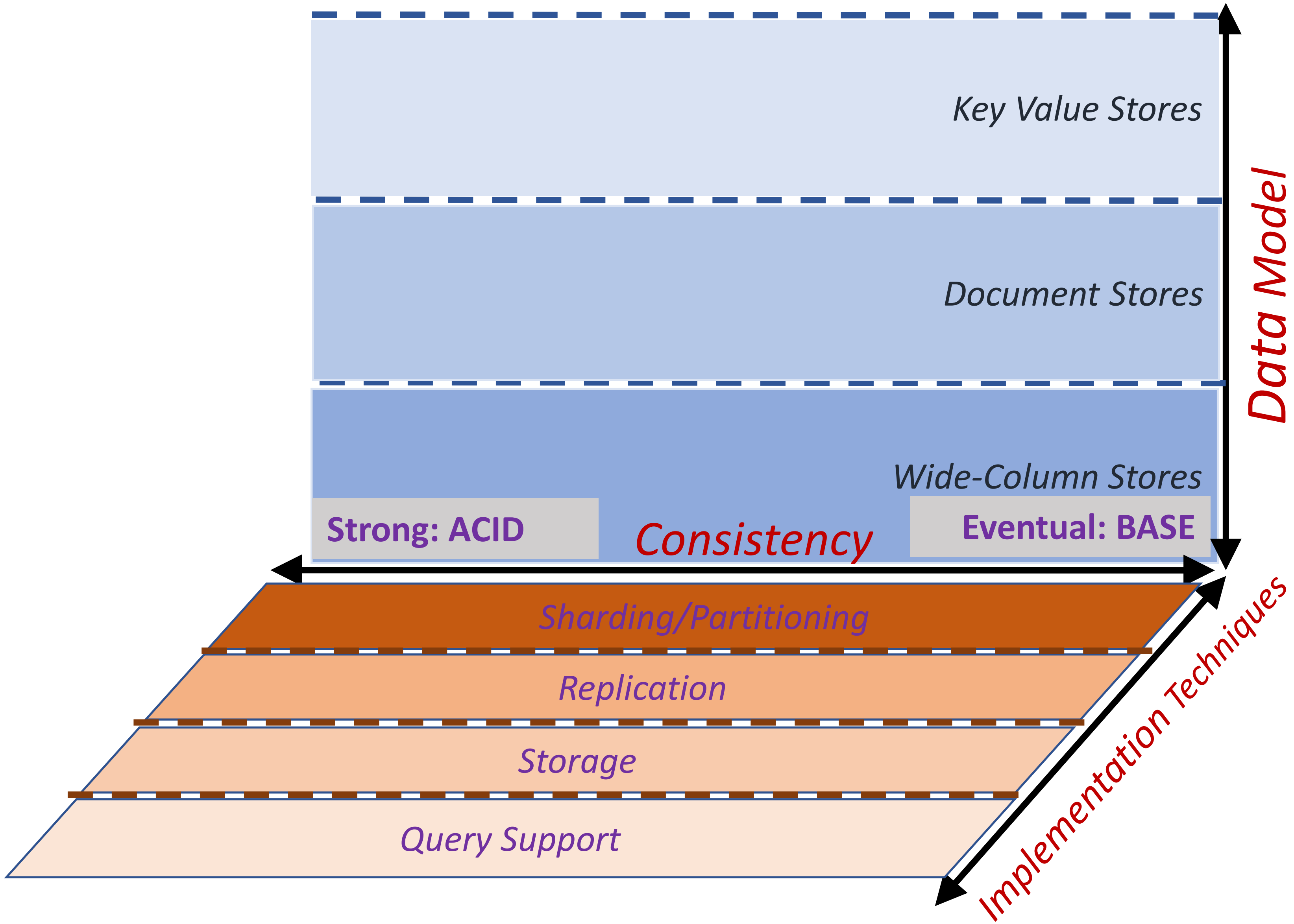
(Yet) Another Framework



(Yet) Another Framework



(Yet) Another Framework



(Yet) Another Framework

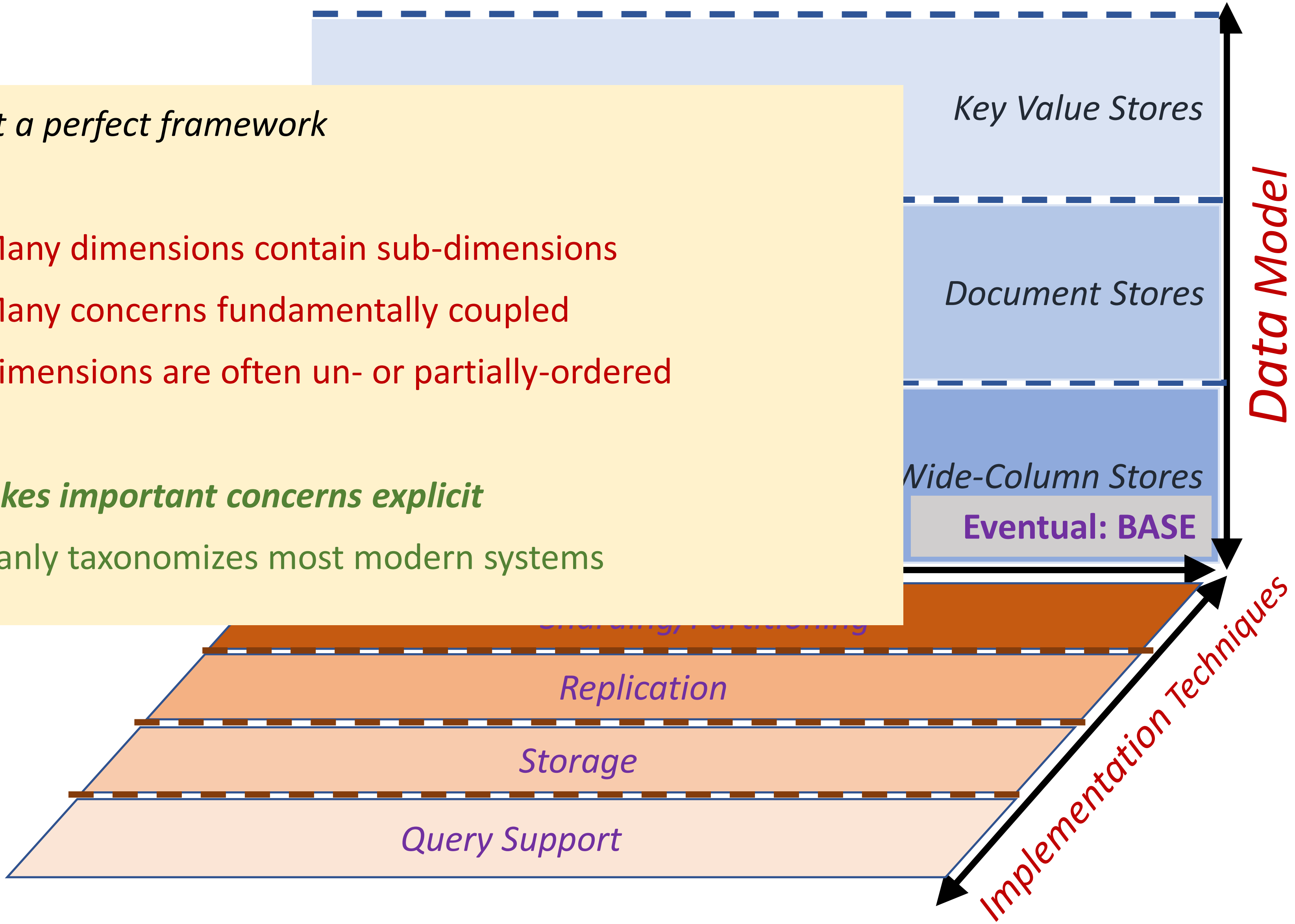
Still not a perfect framework

Cons:

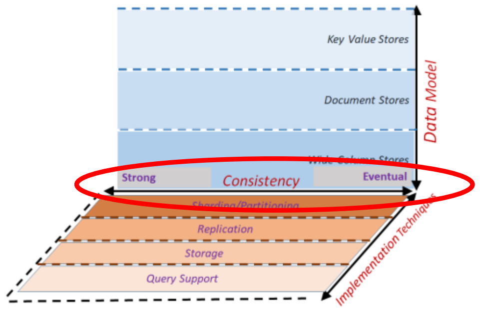
- Many dimensions contain sub-dimensions
- Many concerns fundamentally coupled
- Dimensions are often un- or partially-ordered

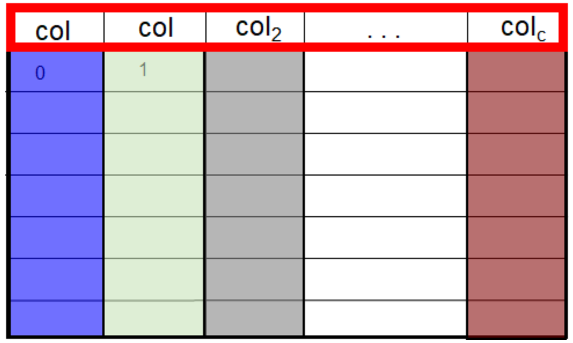
Pros:

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

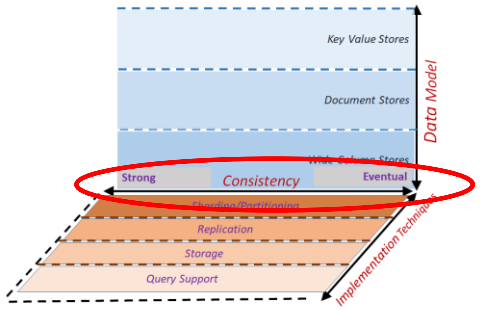


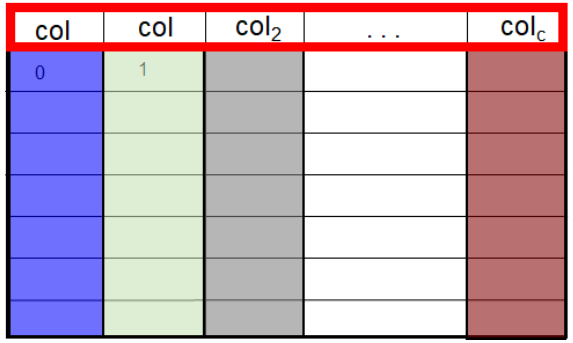
Consistency





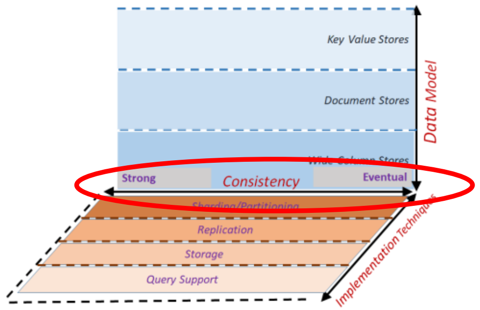
Consistency

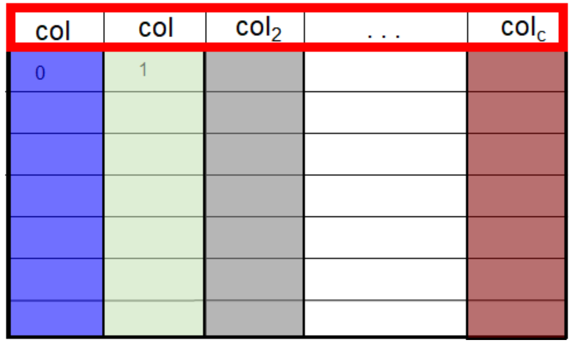




How to keep data in sync?

Consistency

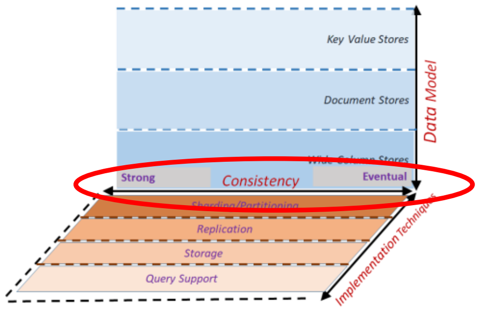


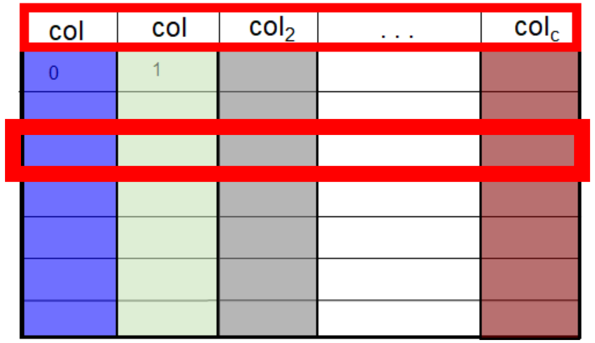


How to keep data in sync?

- Partitioning → single row spread over multiple machines

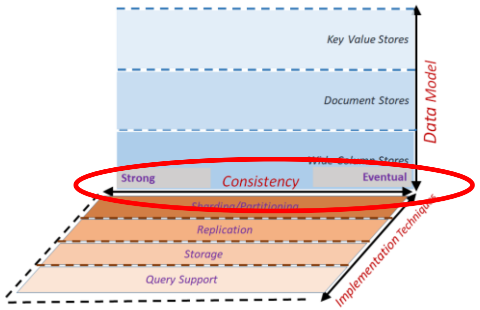
Consistency



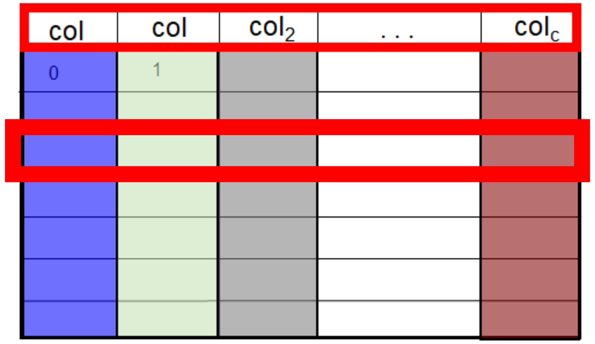


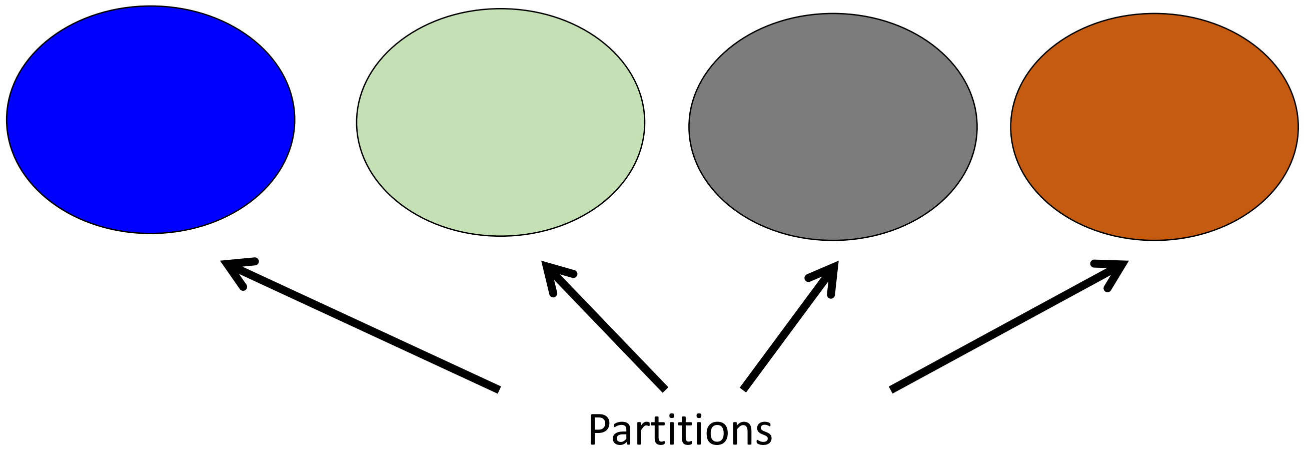
How to keep data in sync?

- Partitioning → single row spread over multiple machines



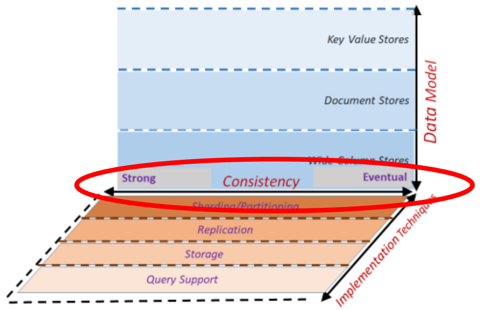
Consistency



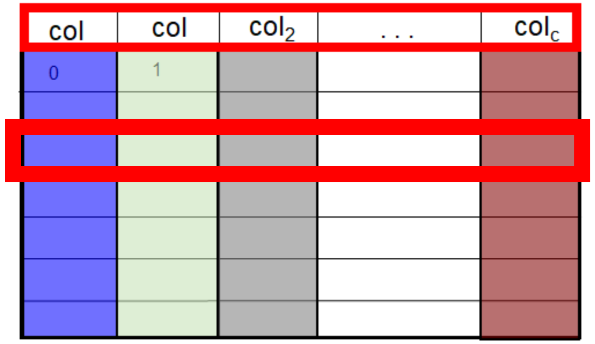


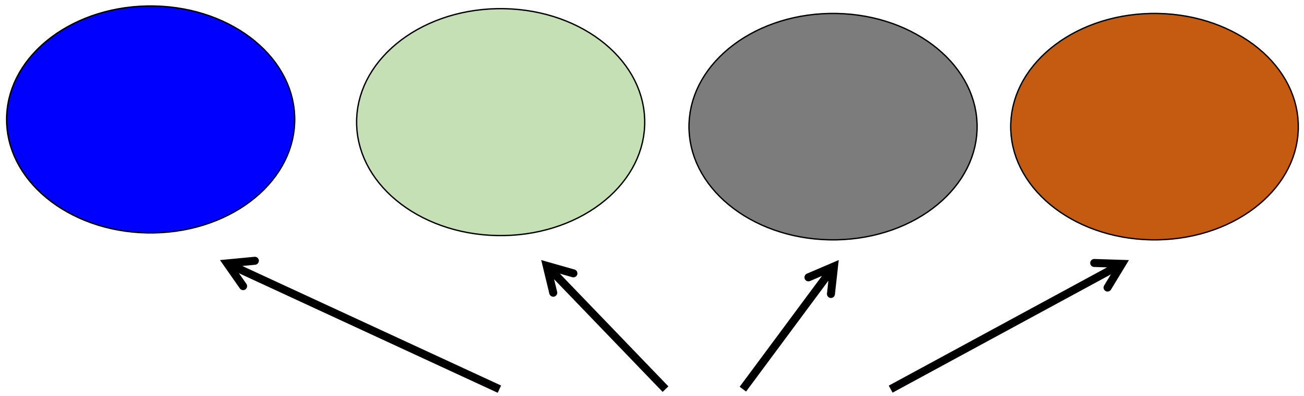
How to keep data in sync?

- Partitioning → single row spread over multiple machines



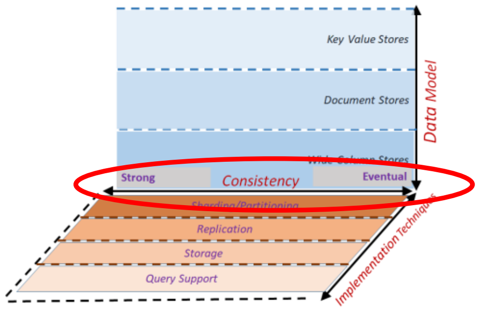
Consistency



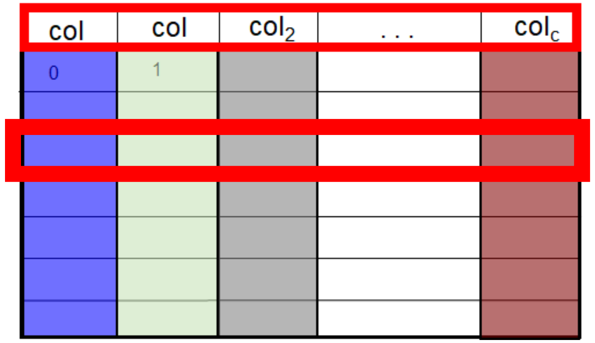


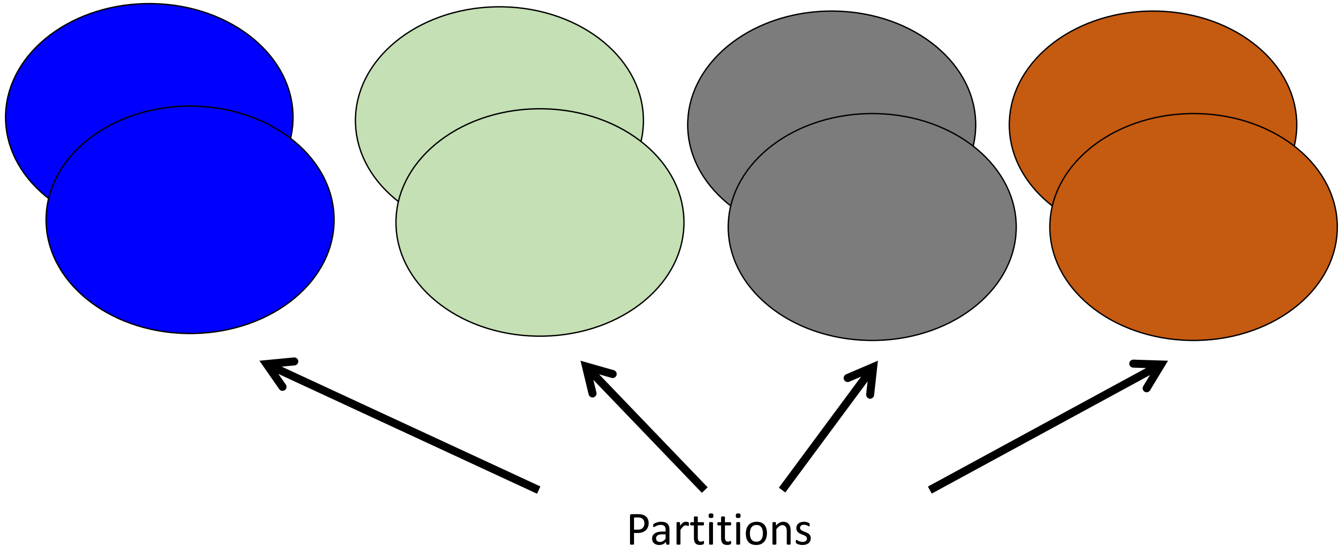
How to keep data in sync?

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



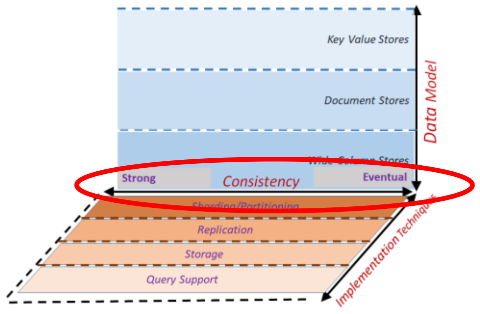
Consistency



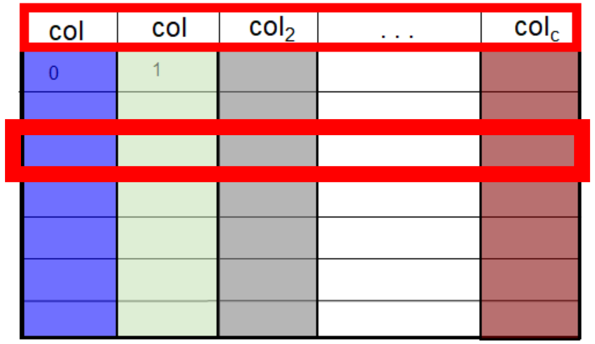


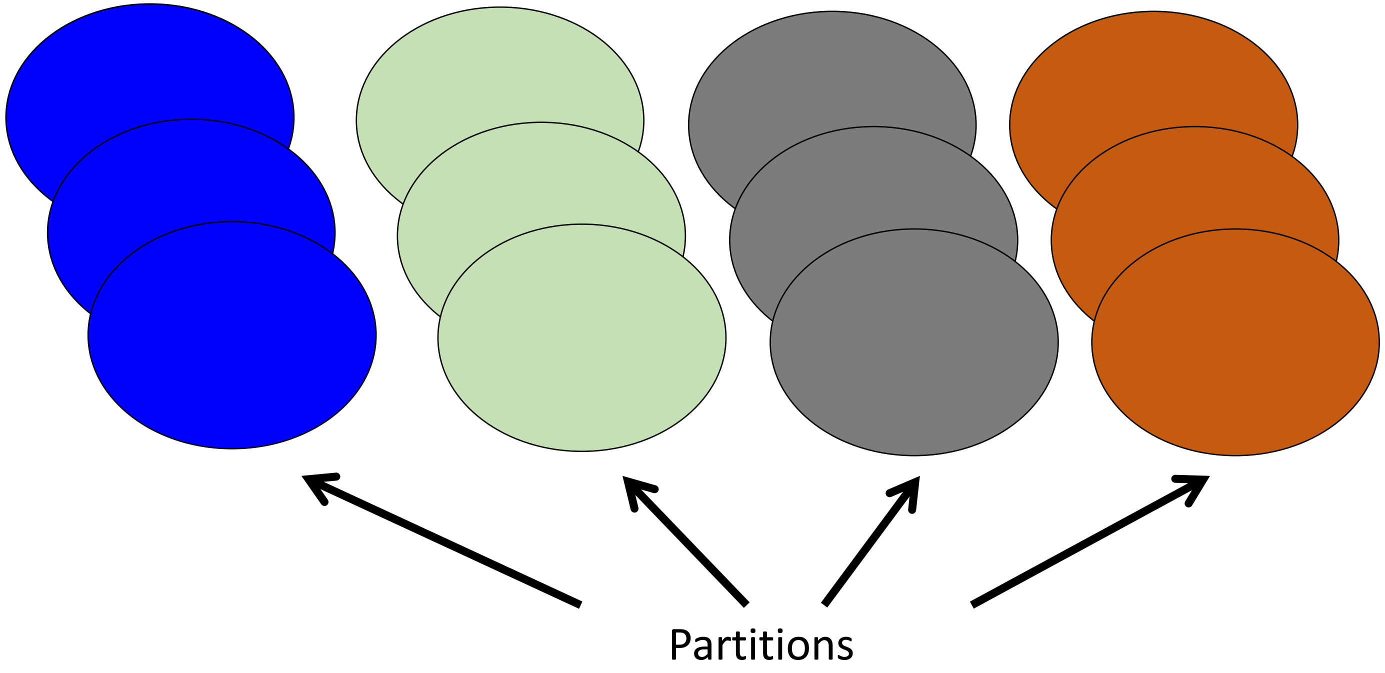
How to keep data in sync?

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



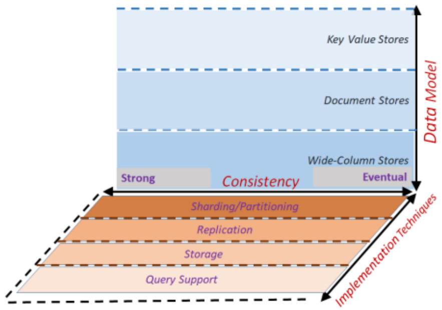
Consistency





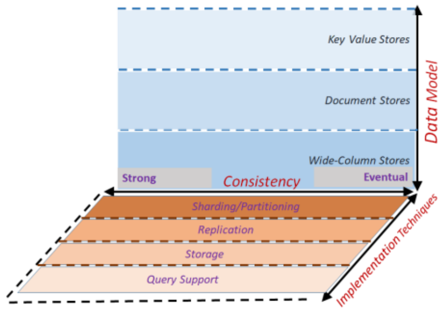
How to keep data in sync?

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

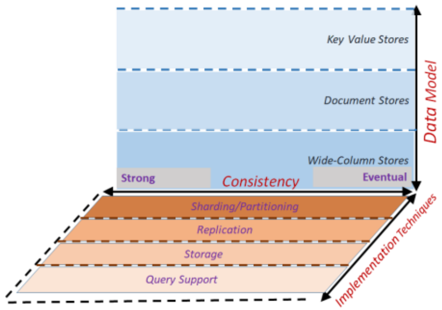


Consistency: the core problem

Consistency: the core problem

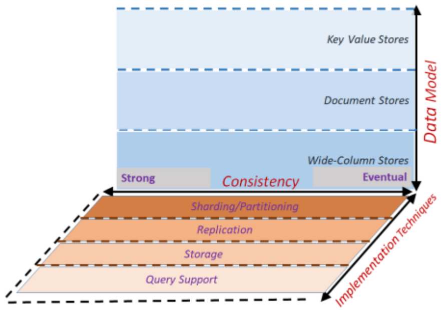


- Clients perform reads and writes



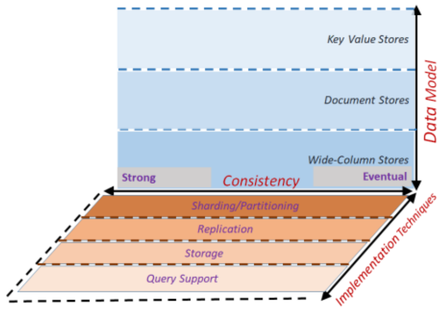
Consistency: the core problem

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



Consistency: the core problem

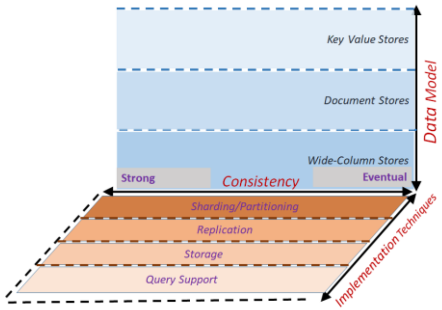
- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers



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

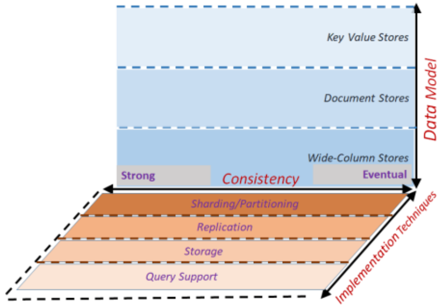
Consistency: the core problem



- 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?

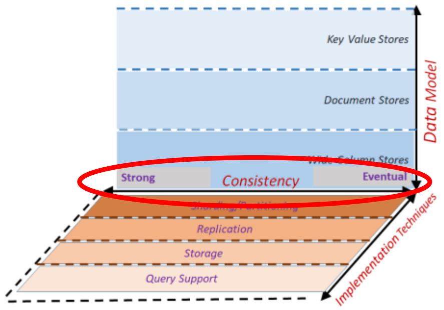
Consistency: the core problem



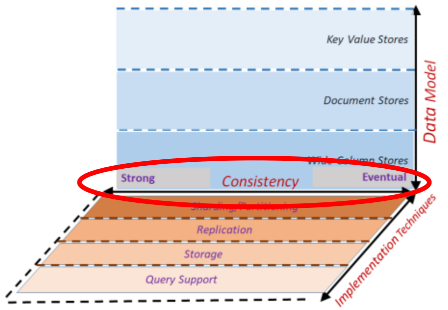
- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes

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

Consistency: CAP Theorem

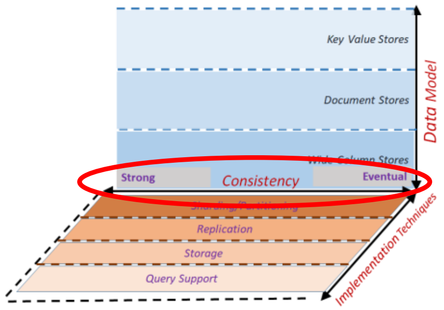


Consistency: CAP Theorem



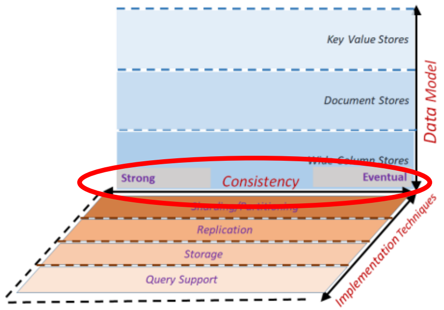
- A distributed system can satisfy at most 2/3 guarantees of:

Consistency: CAP Theorem



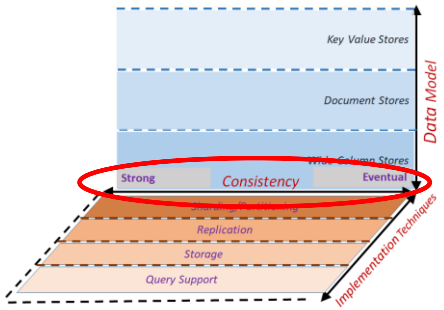
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:

Consistency: CAP Theorem



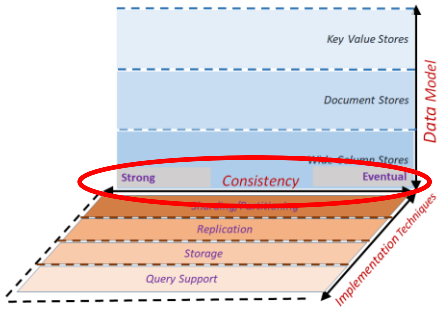
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client

Consistency: CAP Theorem



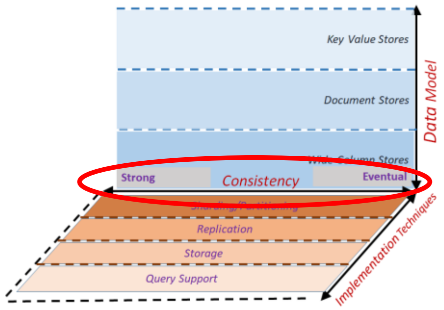
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:

Consistency: CAP Theorem



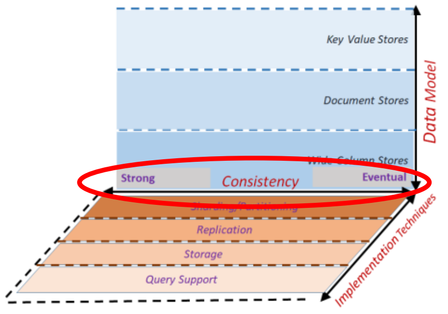
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly

Consistency: CAP Theorem



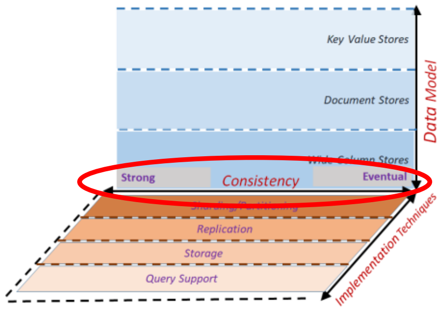
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**:

Consistency: CAP Theorem



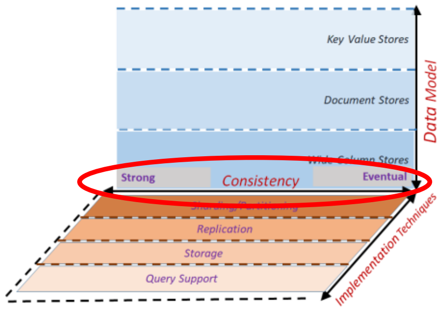
- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**: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

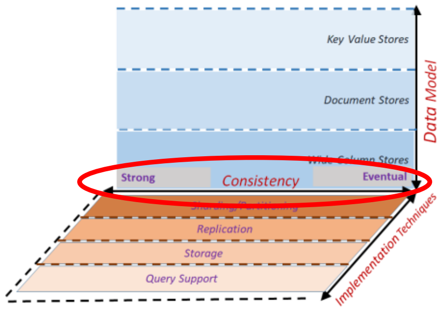
Partitions

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

Consistency

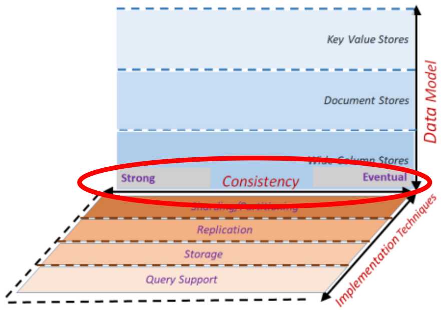
- all nodes see same data at any time, or reads return latest written value by any client.
- ***This basically means correctness!***

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

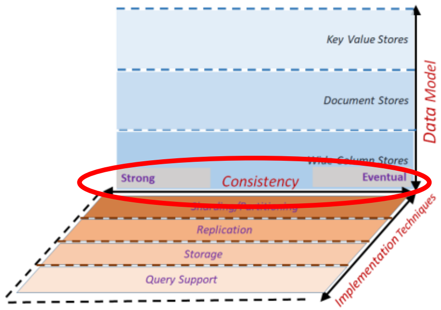
Consistency: CAP Theorem



- 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

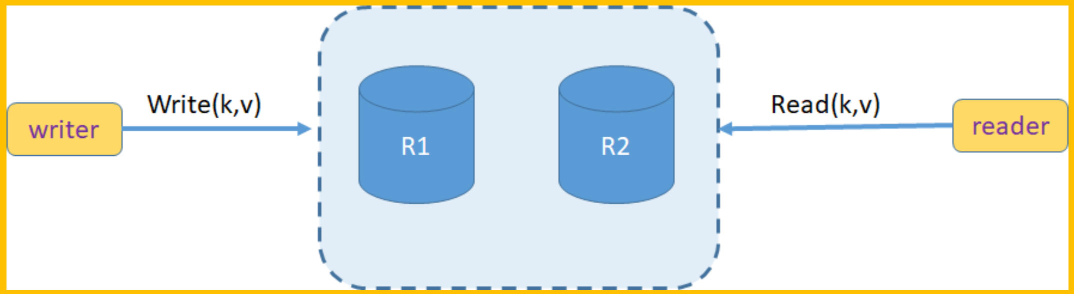
Why is this “theorem” true?

Consistency: CAP Theorem

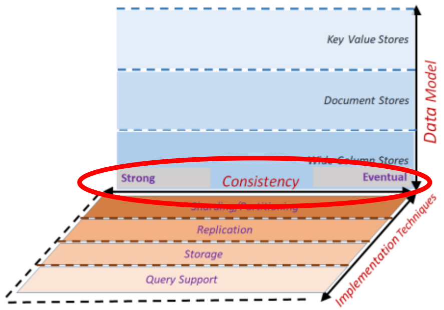


- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**: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

Why is this "theorem" true?

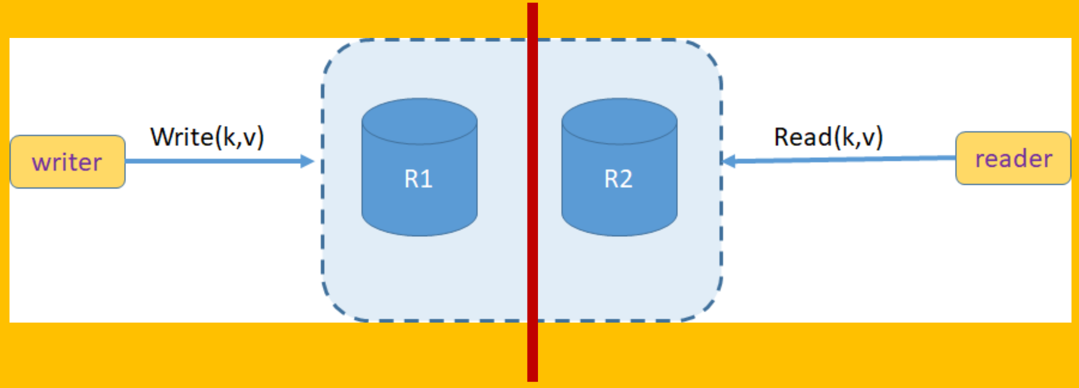


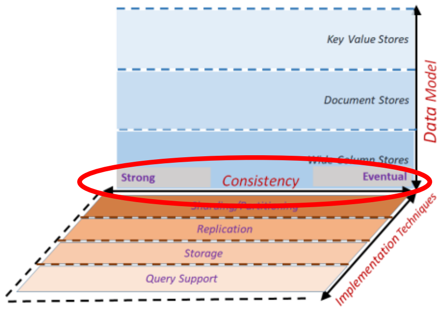
Consistency: CAP Theorem



- 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?

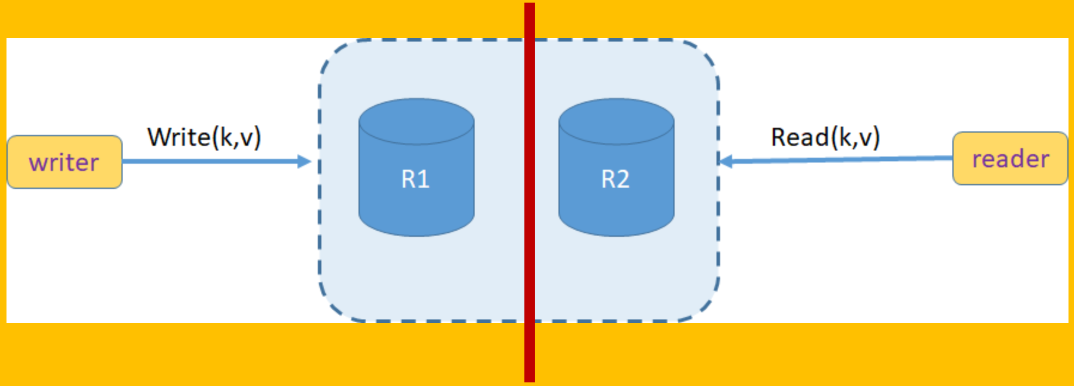




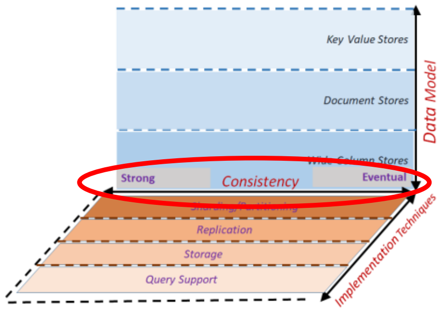
Consistency: CAP Theorem

- 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?



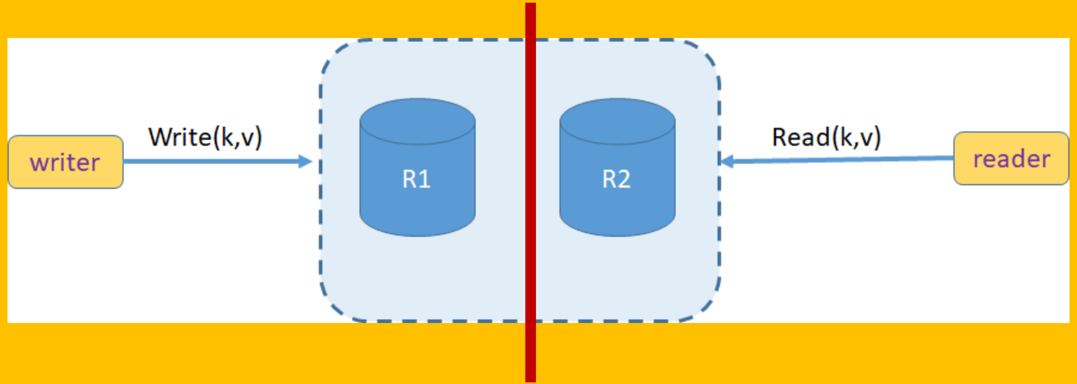
if(partition) { keep going } → !consistent && available



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 networ

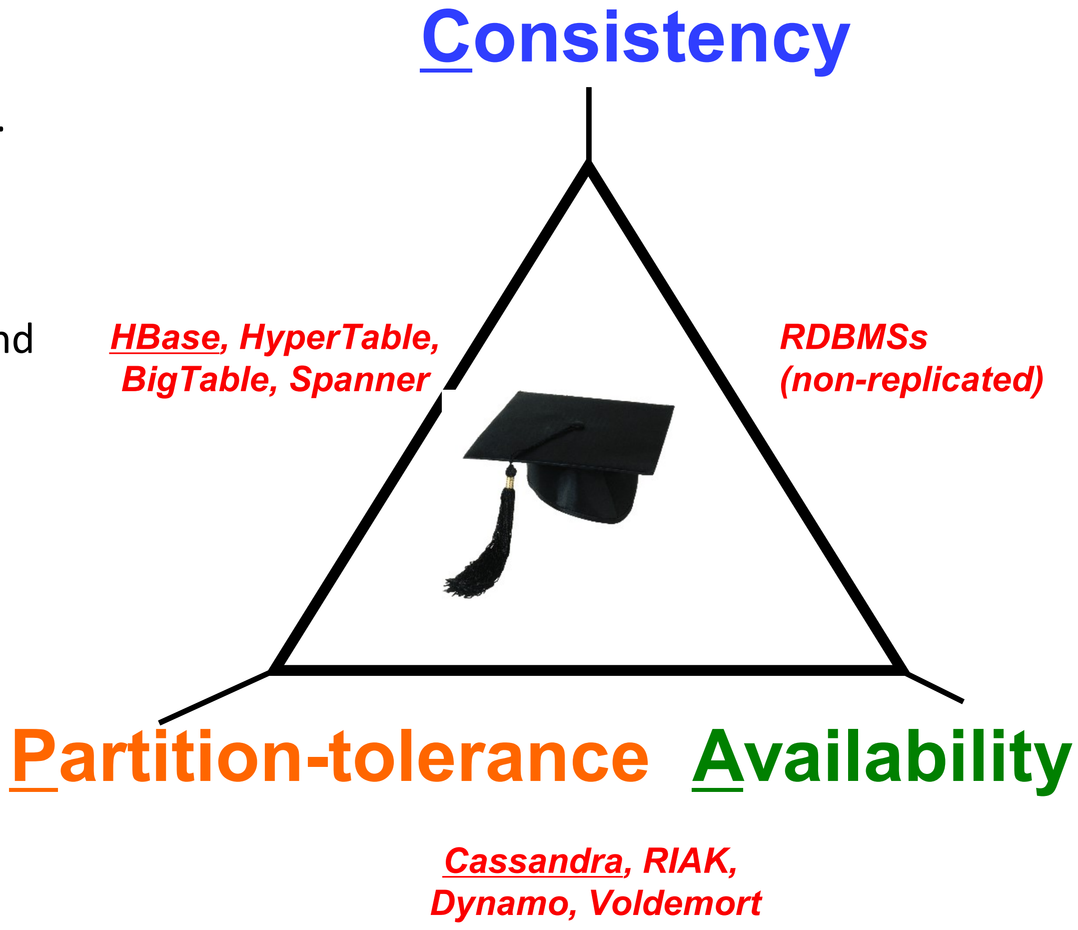
Why is this "theorem" true?



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

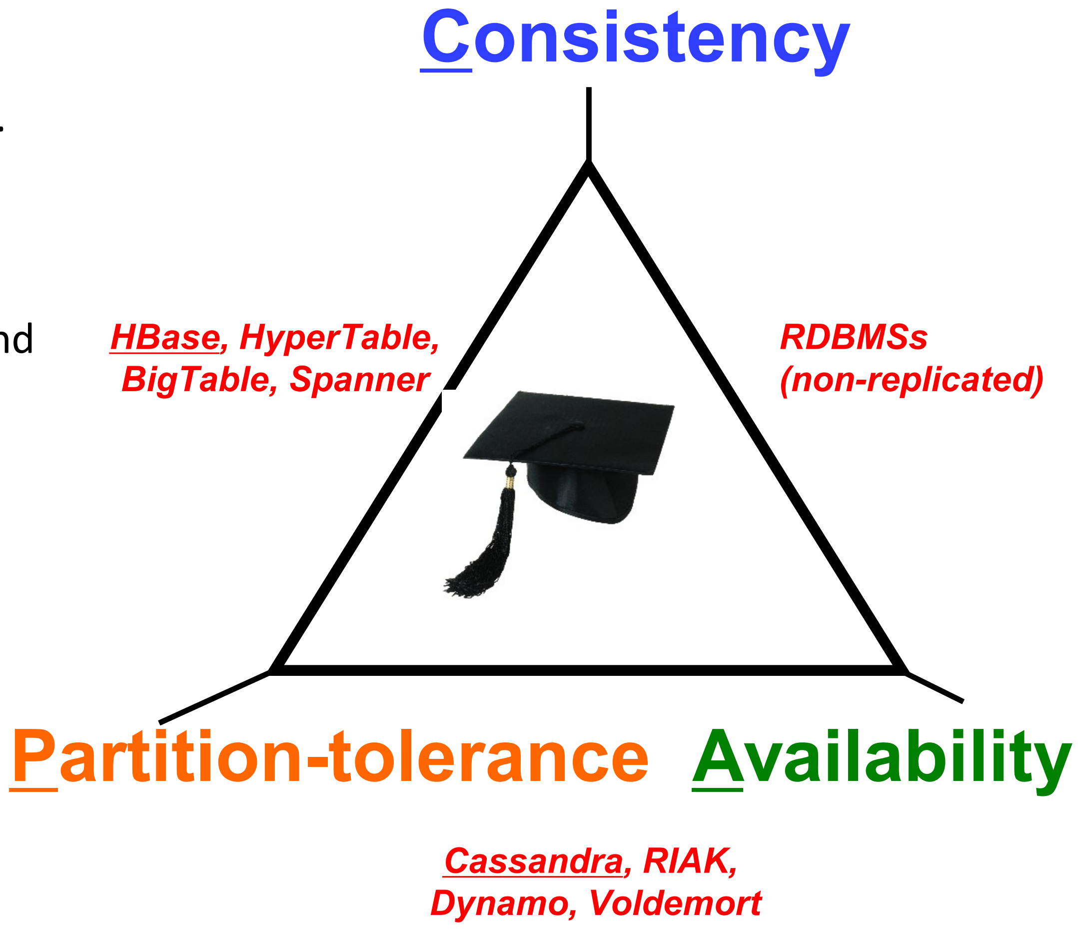
CAP Implications

- 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



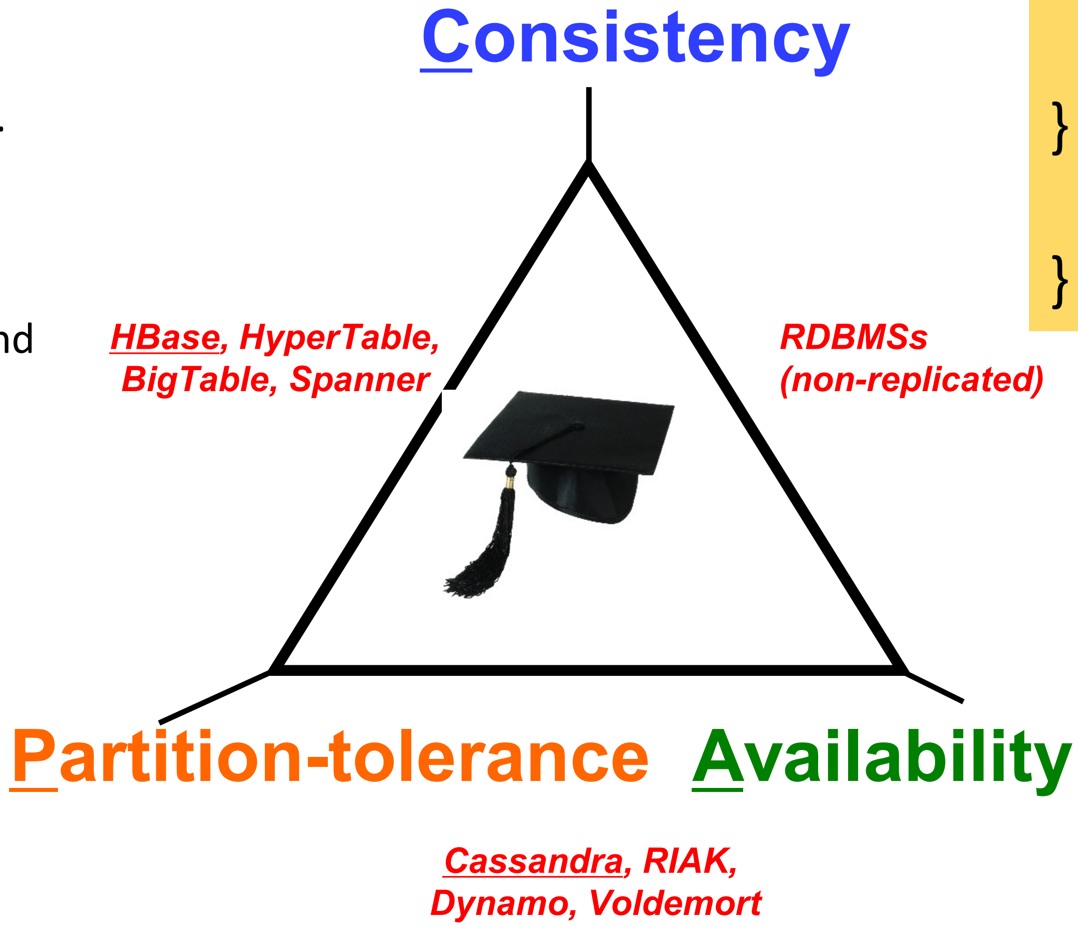
CAP Implications

- 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

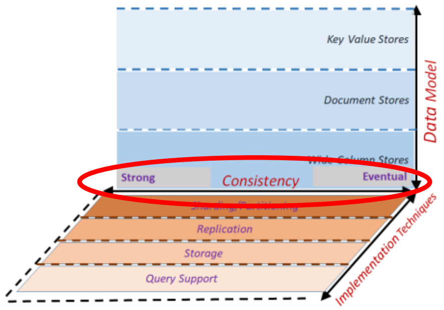
- A distributed storage system can achieve at most two of C, A, and P.
- When partition-tolerance is important, you have to choose between consistency and availability

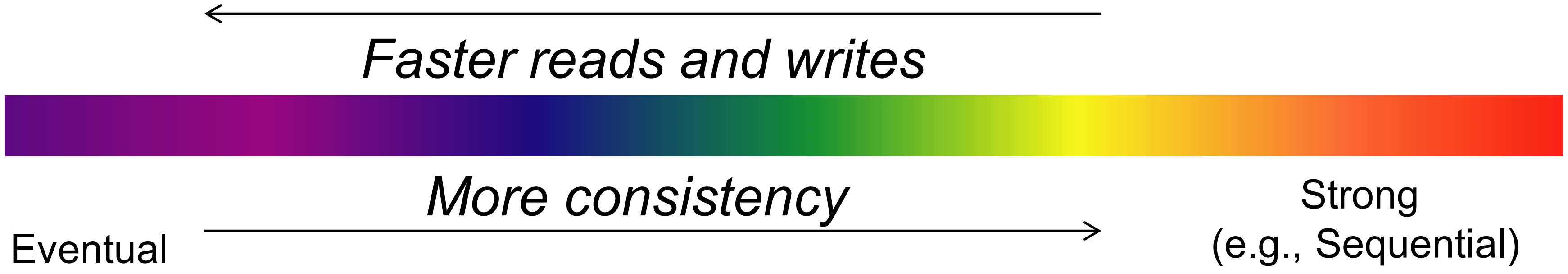


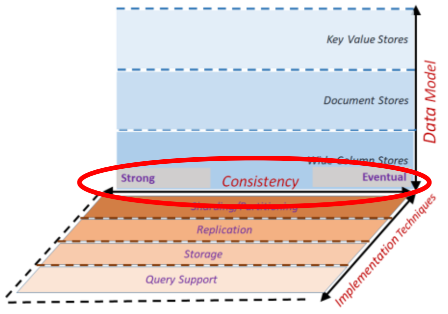
PACELC:

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

Consistency Spectrum

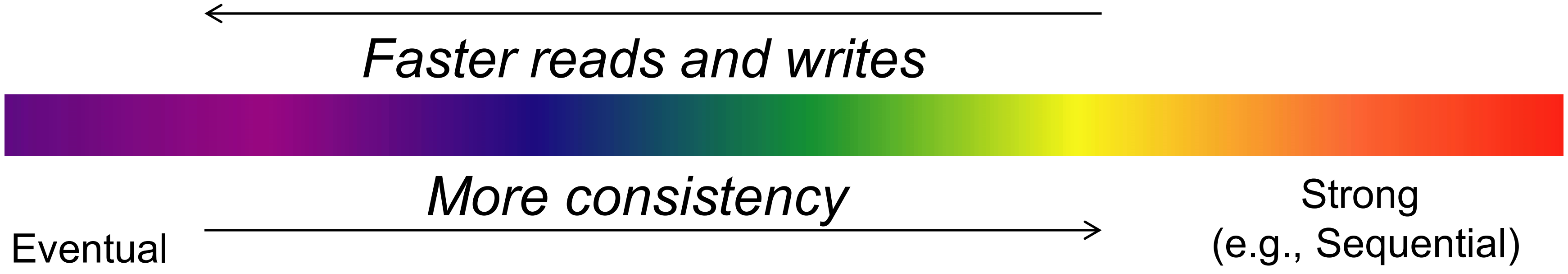




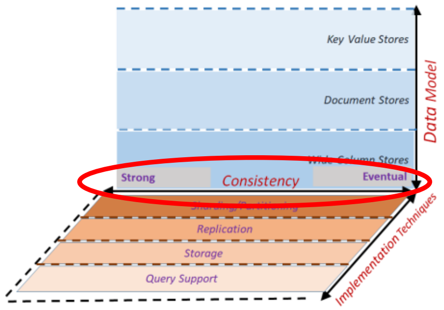


Spectrum Ends: Eventual Consistency

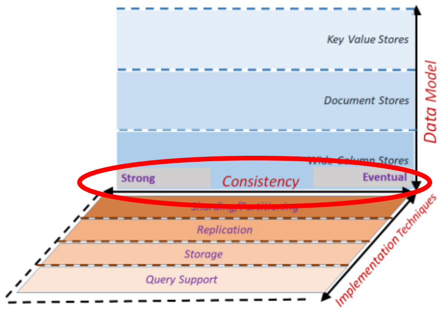
- Eventual Consistency
  - If writes to a key stop, all replicas of key will converge
  - Originally from Amazon's Dynamo and LinkedIn's Voldemort systems



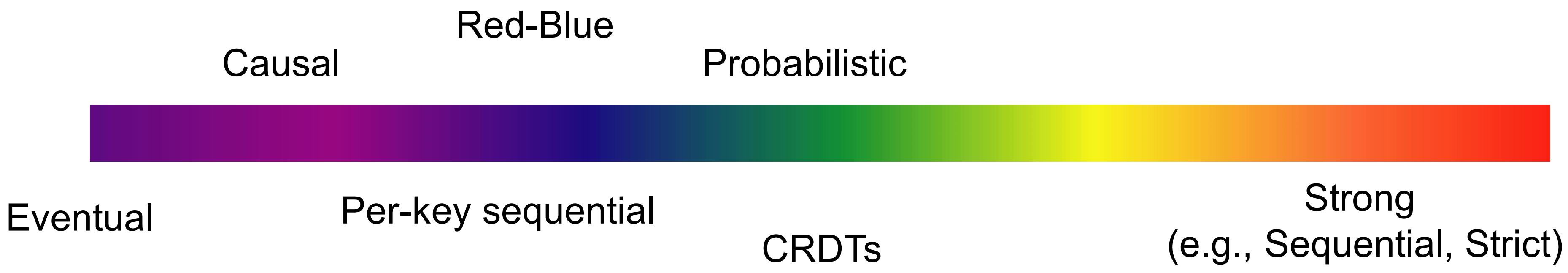
Spectrum Ends: Strong Consistency



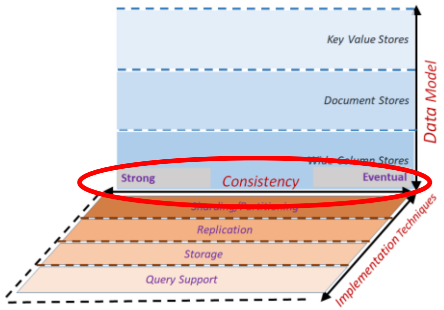
- **Strict:**
  - Absolute time ordering of all shared accesses, reads always return last write
- **Linearizability**:
  - Each operation is visible (or available) to all other clients in real-time order
- **Sequential Consistency** [Lamport]:
  - *"... the result of any execution is the same as if the 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

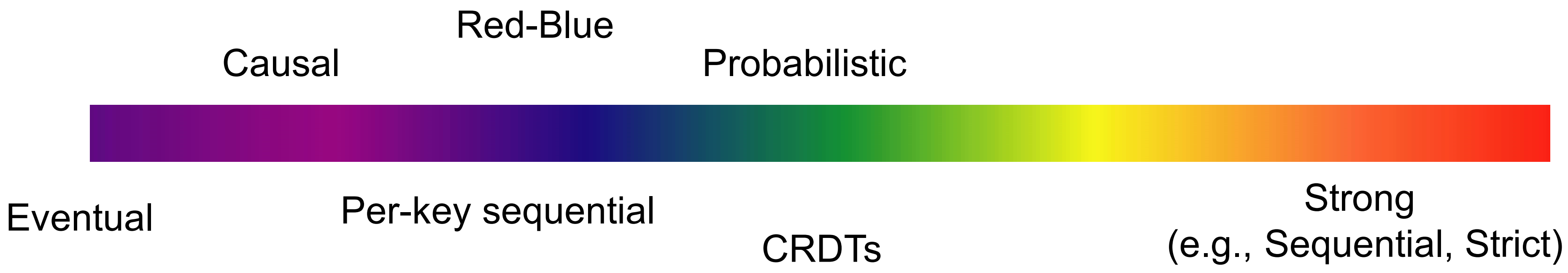


Many *Many* Consistency Models



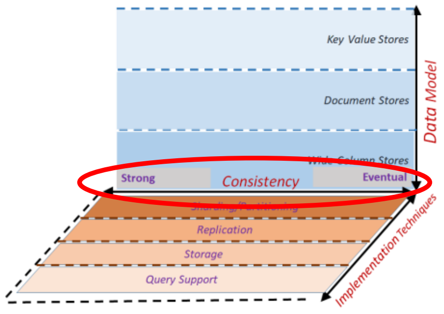
Many *Many* Consistency Models

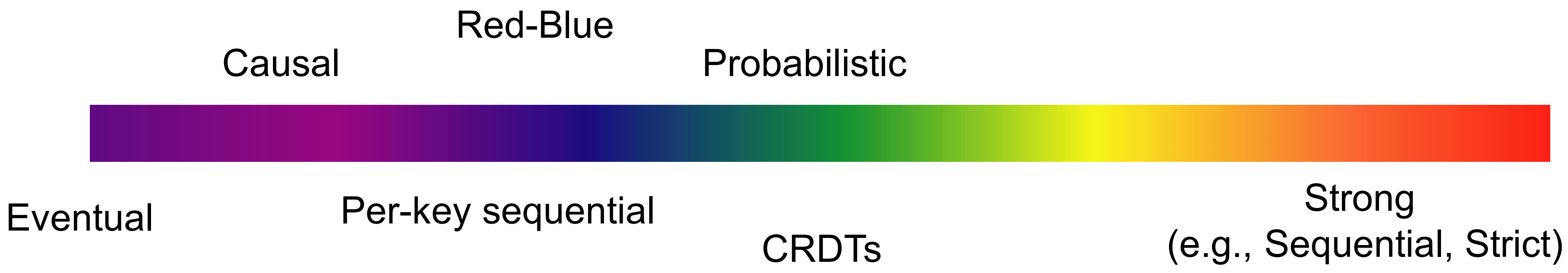




- Amazon S3 – **eventual** consistency
- Amazon Simple DB – **eventual** or strong
- Google App Engine – **strong** or eventual
- Yahoo! PNUTS – **eventual** or strong
- Windows Azure Storage – **strong** (or eventual)
- Cassandra – **eventual** or strong (if R+W > N)
- ...

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)
- …

<u>Question</u>: How to choose what to use or support?

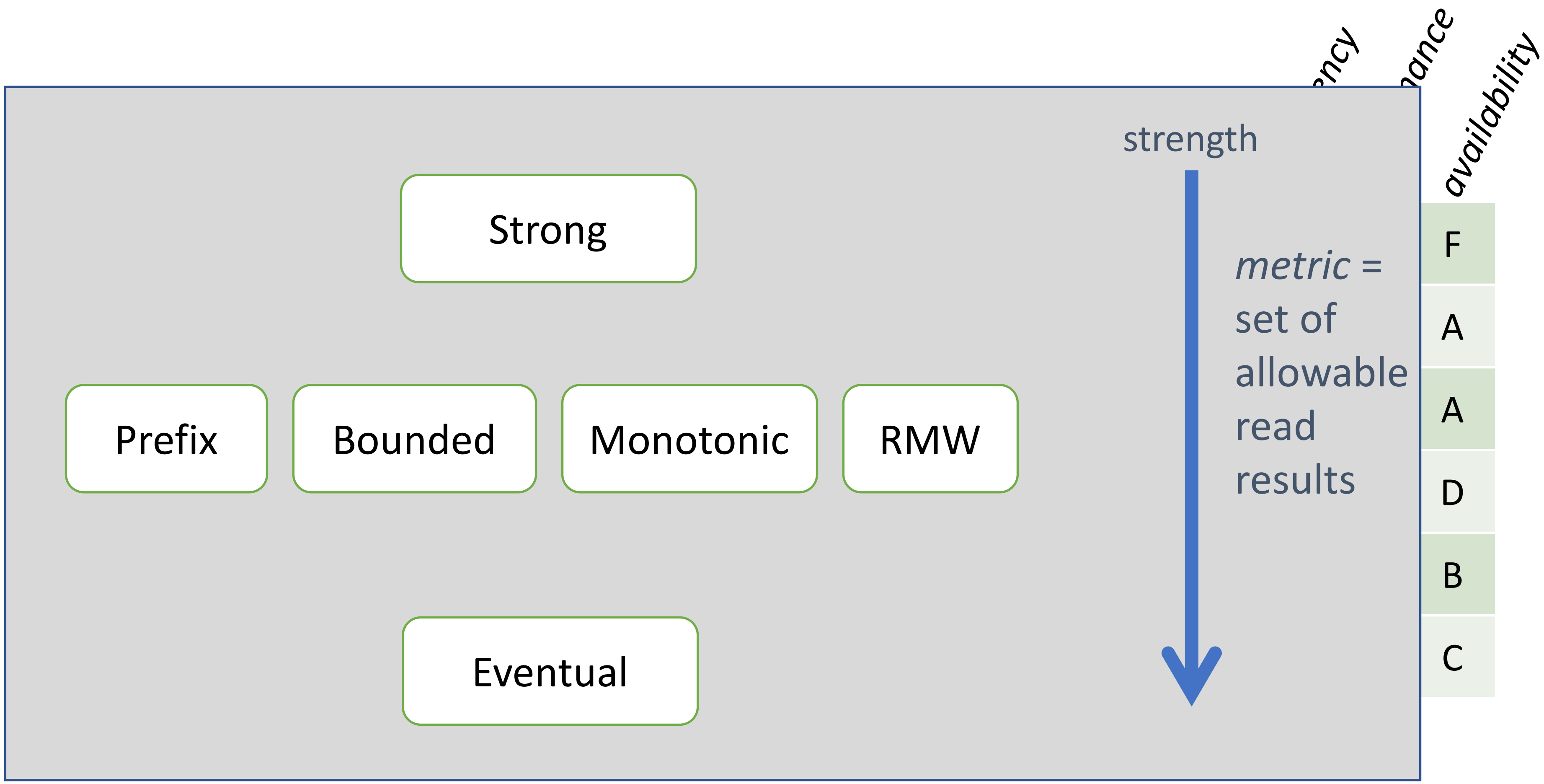
Some Consistency Guarantees

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

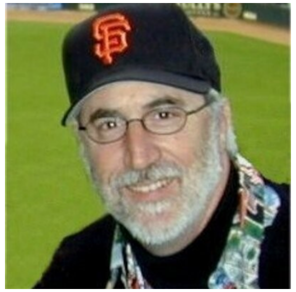
		consistency	performance	availability
Strong Consistency	See all previous writes.	A	D	F
Eventual Consistency	See subset of previous writes.	D	A	A
Consistent Prefix	See initial sequence of writes.	C	B	A
Bounded Staleness	See all “old” writes.	B	C	D
Monotonic Reads	See increasing subset of writes.	C	B	B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

Some Consistency Guarantees



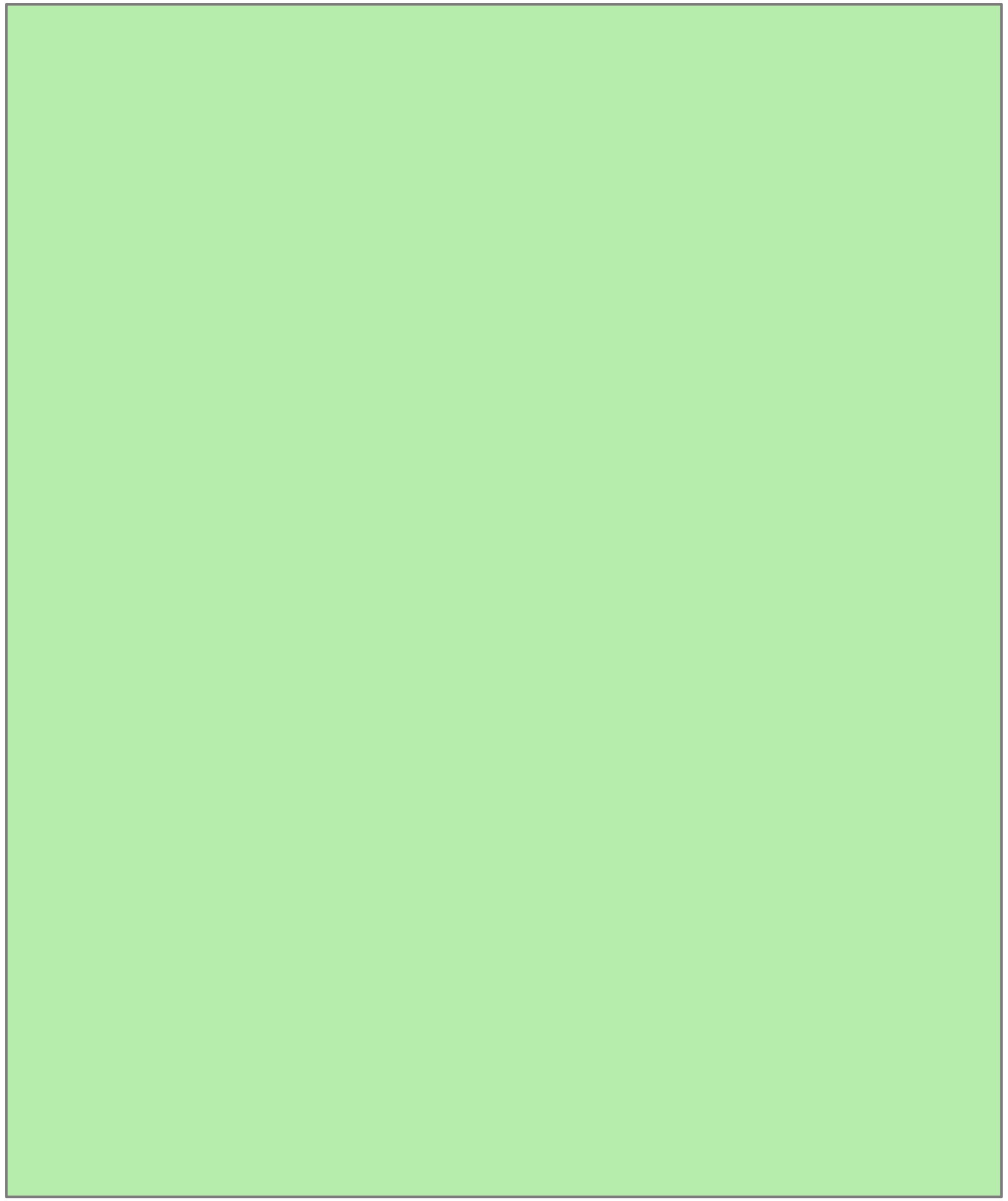
The Game of Soccer

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

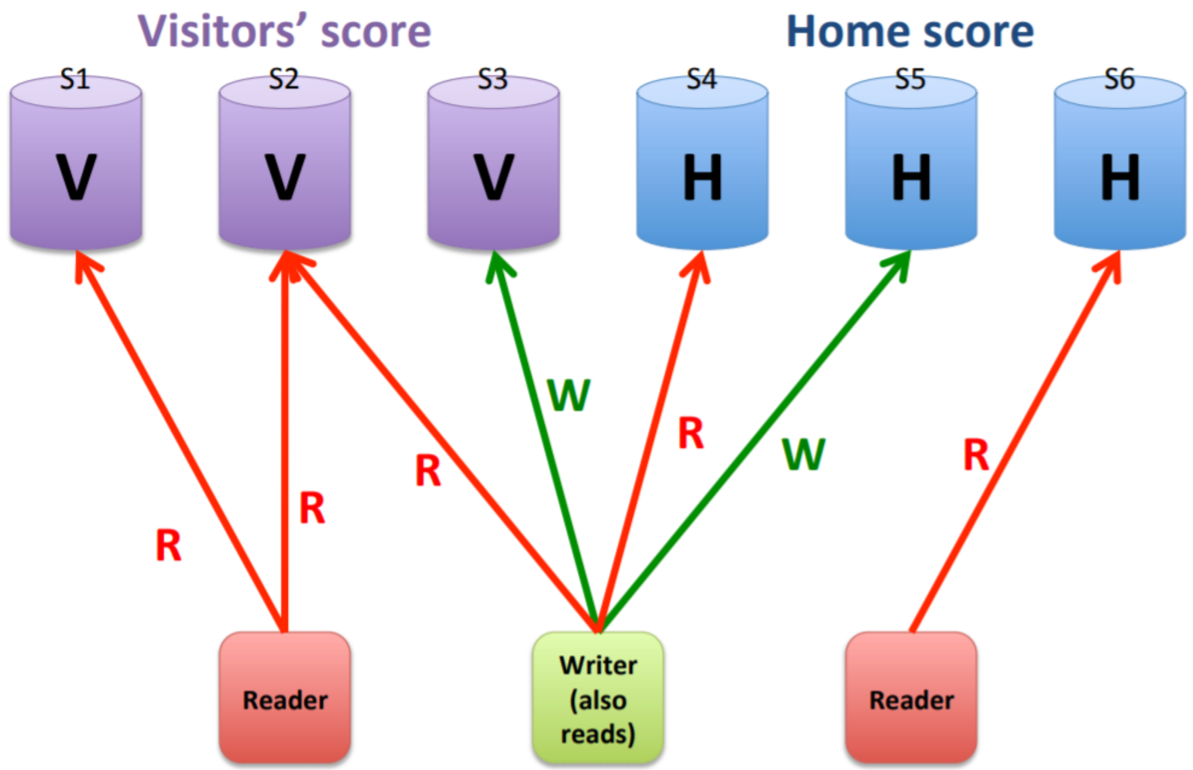
```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
      for each goal {
        if visiting-team-scored {
          score = Read (“visitors”);
          Write (“visitors”, score + 1);
        } else {
          score = Read (“home”);
          Write (“home”, score + 1);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

The Game of Soccer

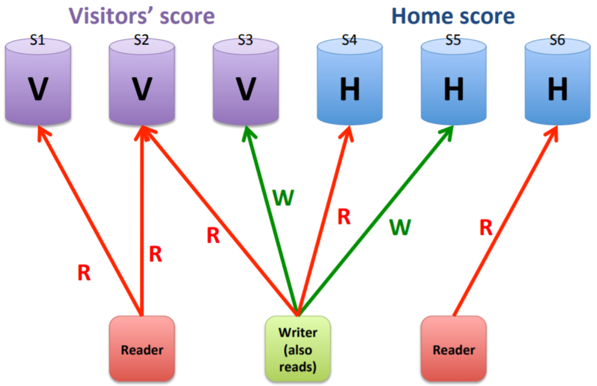
```
for half = 1 .. 2  {
  while half not over {
      kick-the-ball-at-the-goal
      for each goal {
        if visiting-team-scored {
          score = Read ("visitors");
          Write ("visitors", score + 1);
        } else {
          score = Read ("home");
          Write ("home", score + 1);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



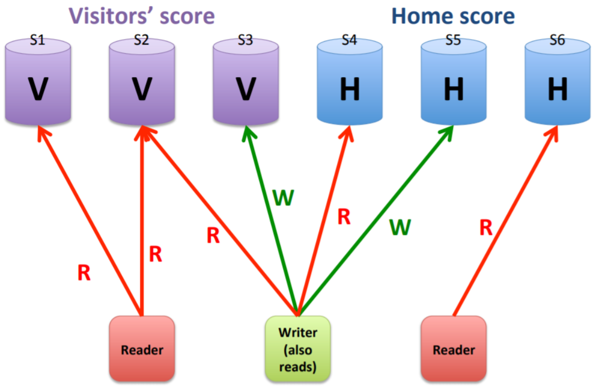
Official Scorekeeper




```
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

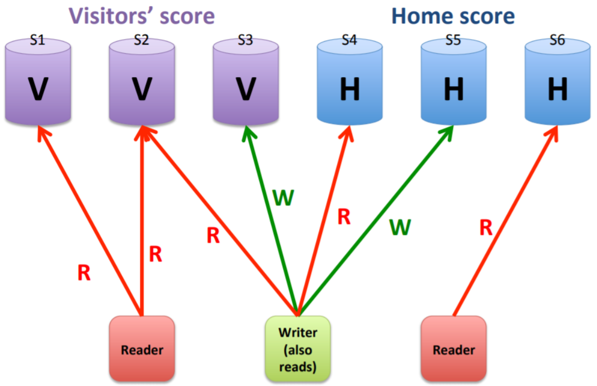



```
score = Read ("visitors");
Write ("visitors", score + 1);
```

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



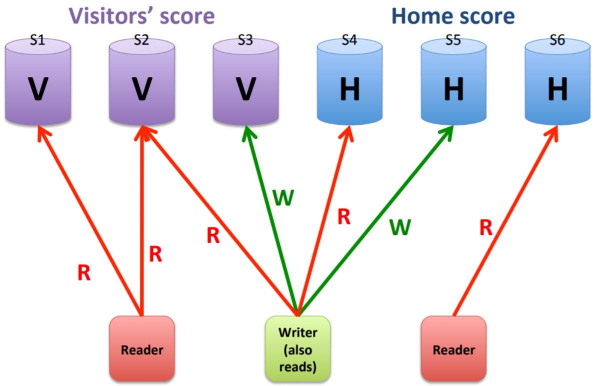

```
score = Read ("visitors");
Write ("visitors", score + 1);
```

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



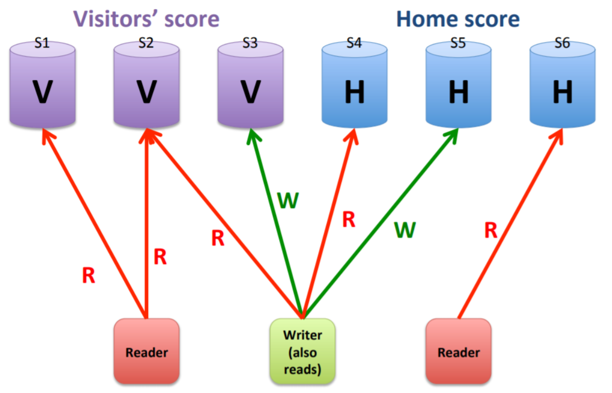

```
score = Read ("visitors");
Write ("visitors", score + 1);
```

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
```

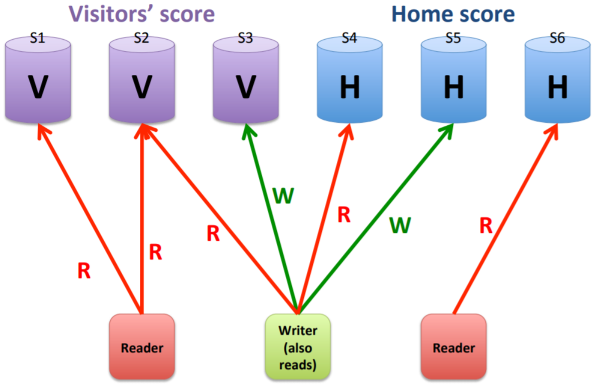
Desired consistency?

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");

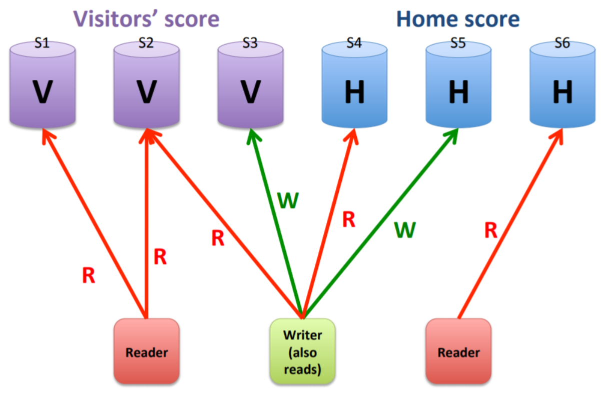
hScore = **Read** ("home");

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

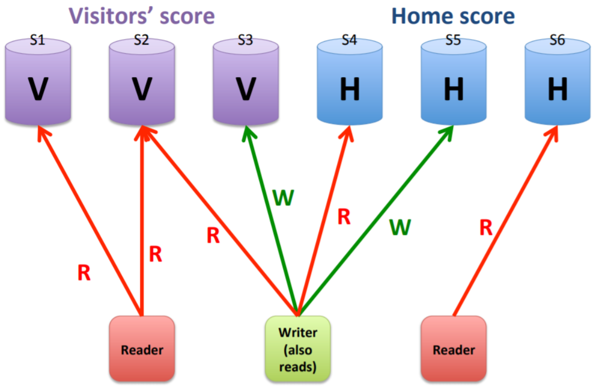



```
vScore = Read ("visitors");
hScore = Read ("home");
if vScore == hScore
    play-overtime
```

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?

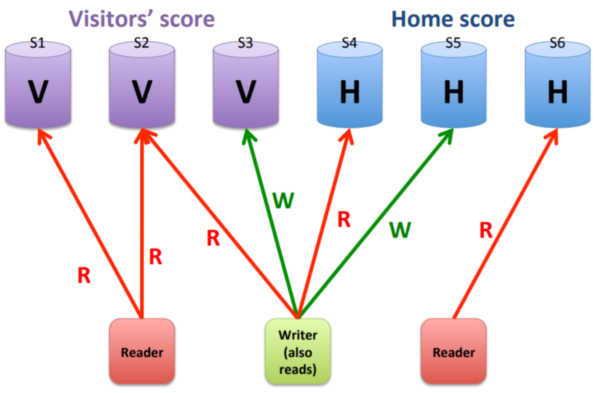
Consistent Prefix

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);
}
```

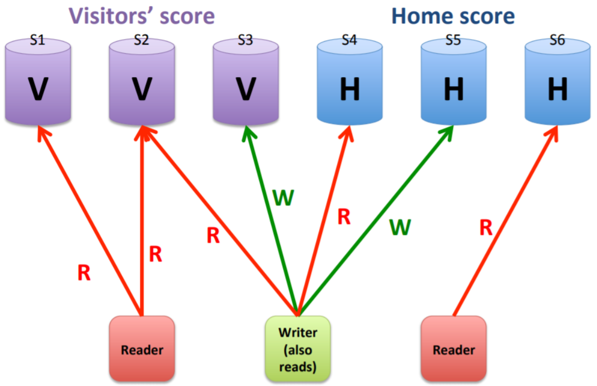
Desired consistency?

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.

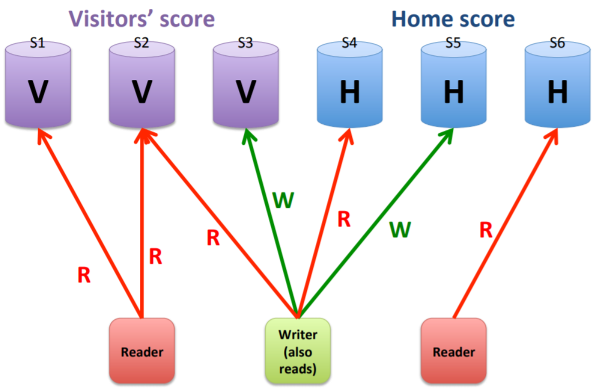
Sportswriter




```
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

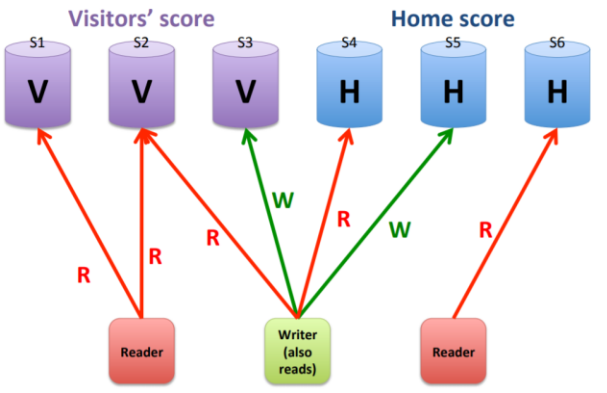



```
While not end of game {
    drink beer;
    smoke cigar;
}
go out to dinner;
vScore = Read ("visitors");
hScore = Read ("home");
write article;
```

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



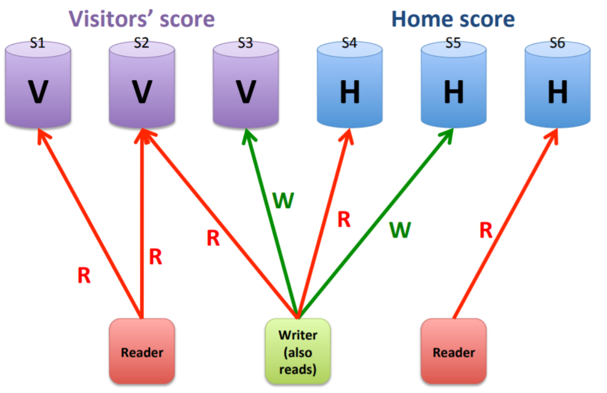

```
While not end of game {
    drink beer;
    smoke cigar;
}
go out to dinner;
vScore = Read ("visitors");
hScore = Read ("home");
write article;
```

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;
```

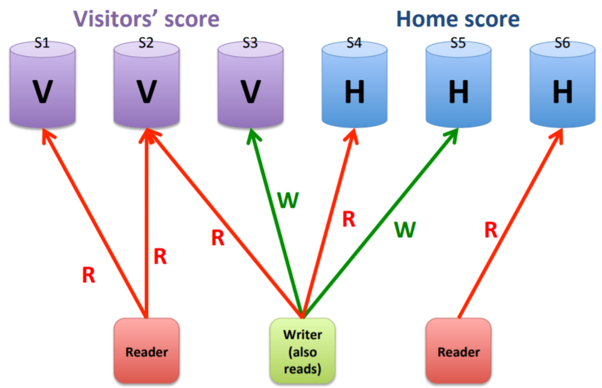
Desired consistency?

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;

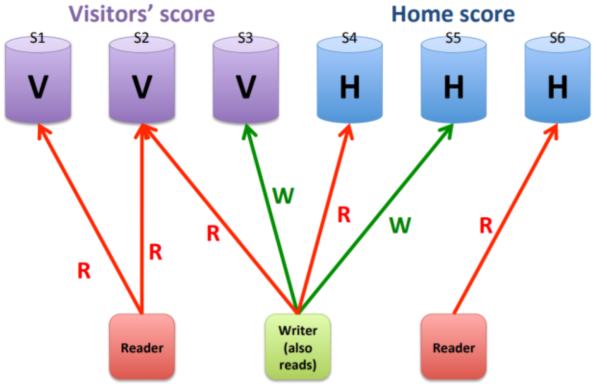
score = **Read** ("home");

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

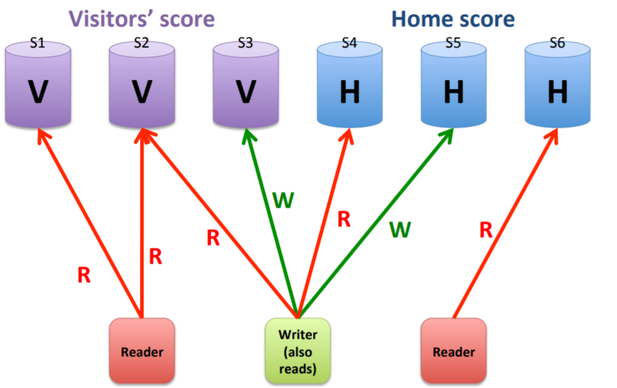


Wait for end of game;
score = **Read** ("home");
stat = **Read** ("season-goals");
Write ("season-goals", stat + score);

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



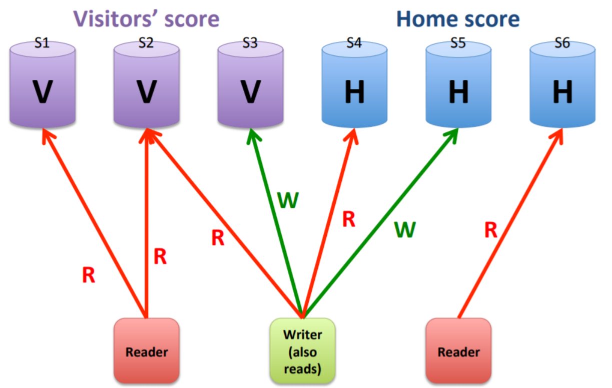

```
Wait for end of game;
score = Read ("home");
stat = Read ("season-goals");
Write ("season-goals", stat + score);
```

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);

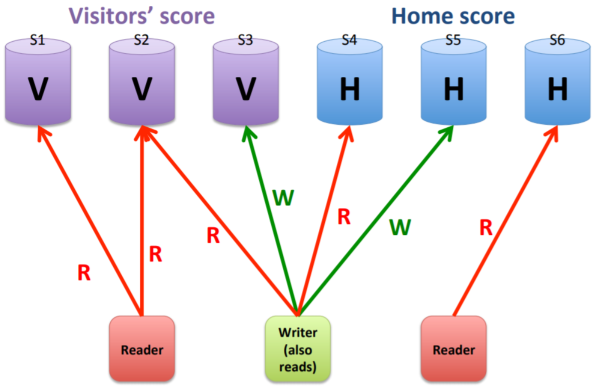
Desired consistency?

Strong Consistency (1st read)

Read My Writes (2nd read)

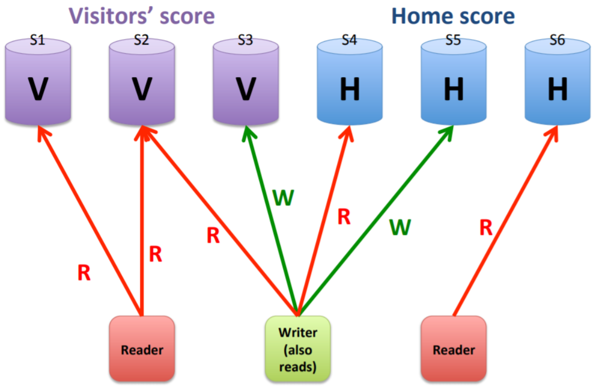
Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.
Bounded Staleness	See all "old" writes.

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.



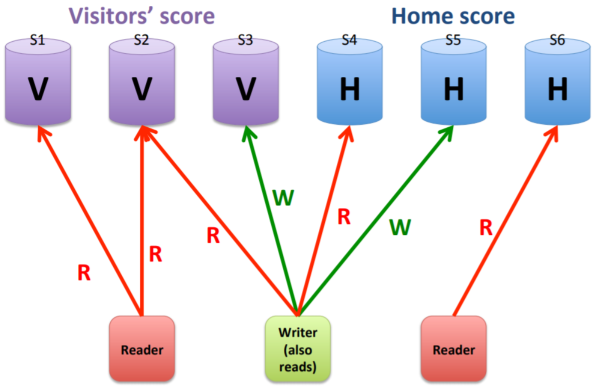
Stat Watcher

```
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** (“visito

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:

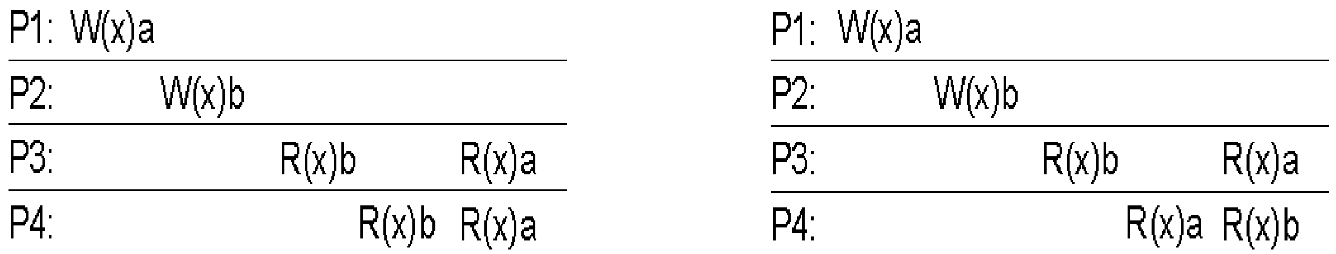
stat = **Read** (“season-runs”)

discuss sta

Eventual Consistency

Sequential Consistency

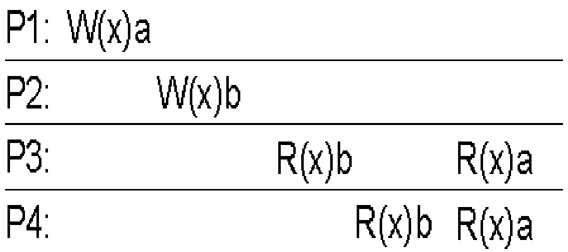
- weaker than strict/strong consistency
  - All operations are executed in *some* sequential order
  - each process issues operations in program order
    - Any valid interleaving is allowed
    - All agree on the same interleaving
    - Each process preserves its program order



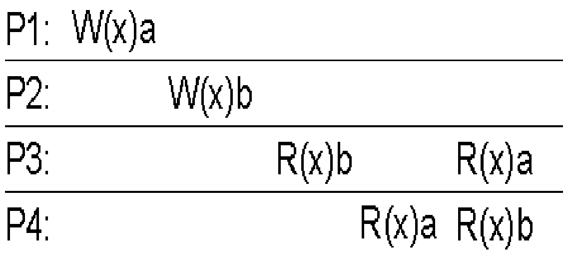
(a) (b)

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



)

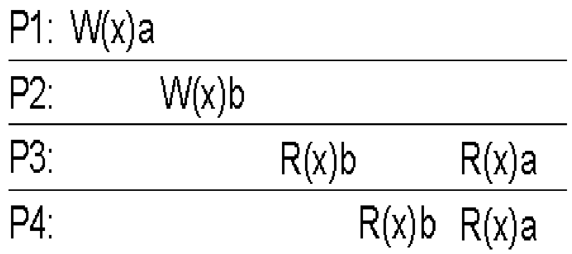


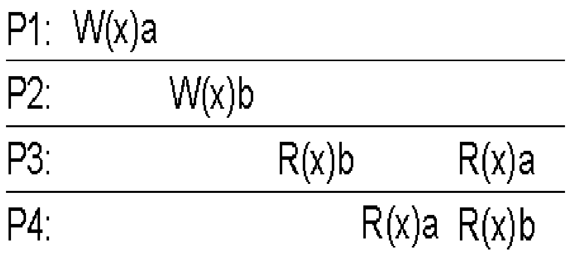
(b)

- ***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





(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 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

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)

Not permitted

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

(b)

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)