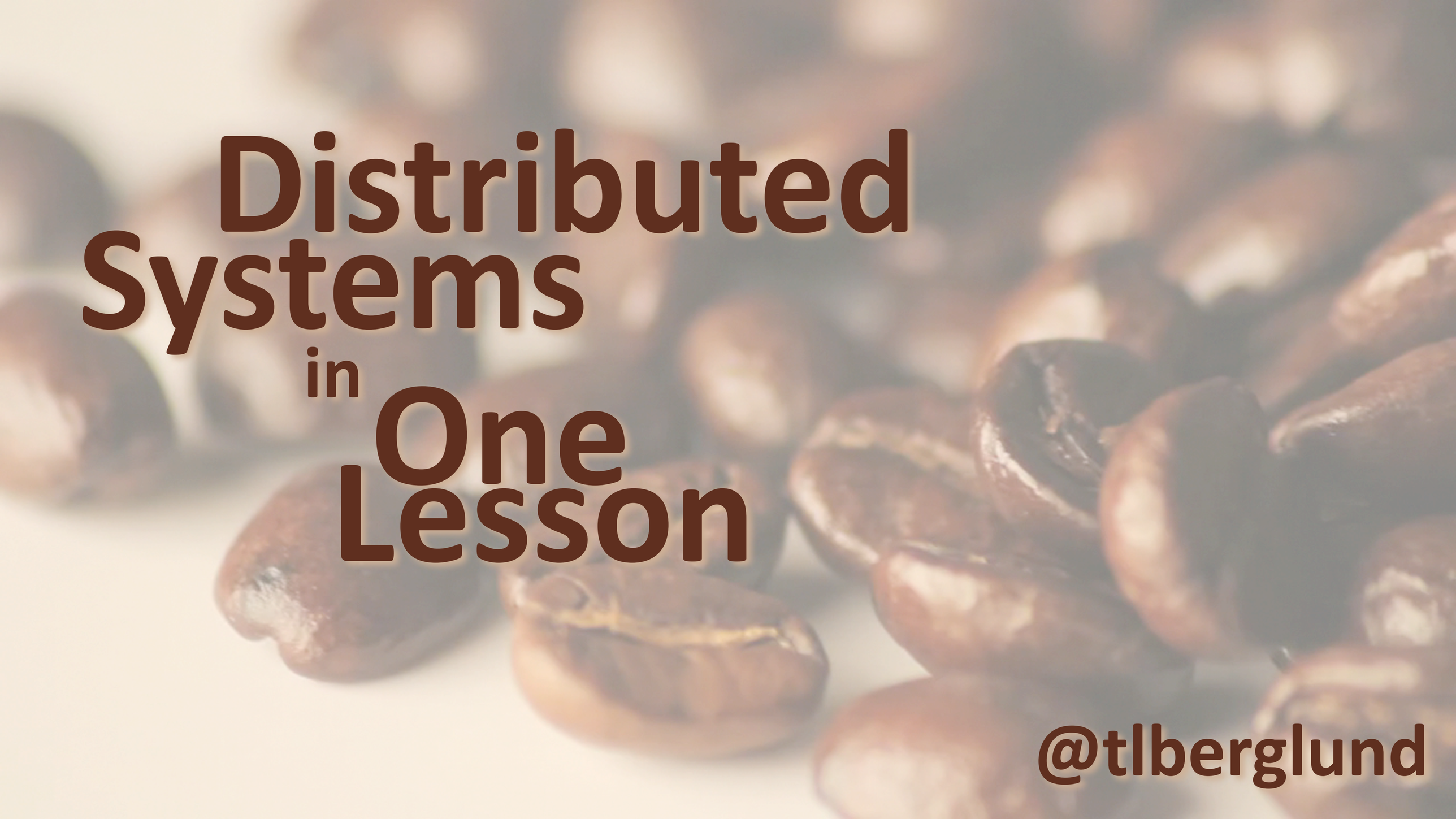

Distributed
Systems
in
One
Lesson
@tlberglund

What are Distributed Systems?

Any system too large to fit
on one computer.

What are Distributed Systems?

- Storage
- Transactions
- Computation
- Coordination

Distributed Storage

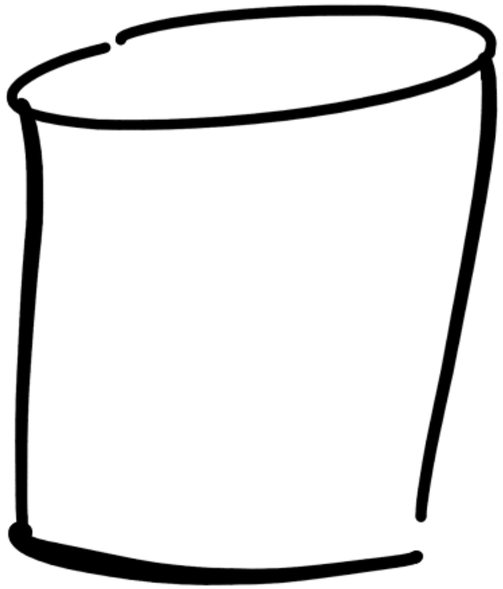
Distributed Storage

- Read replication
- Sharding
- Consistent hashing
- Distributed filesystems

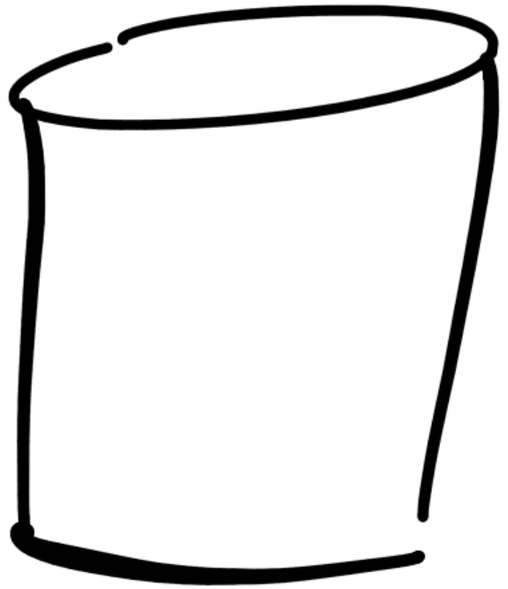
Distributed Storage

- MongoDB
- Cassandra
- HDFS

When Life is Good



When Life is Good

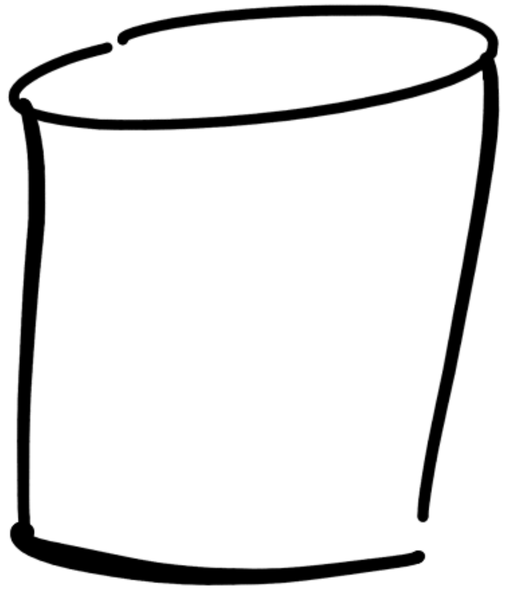


When Life is Good

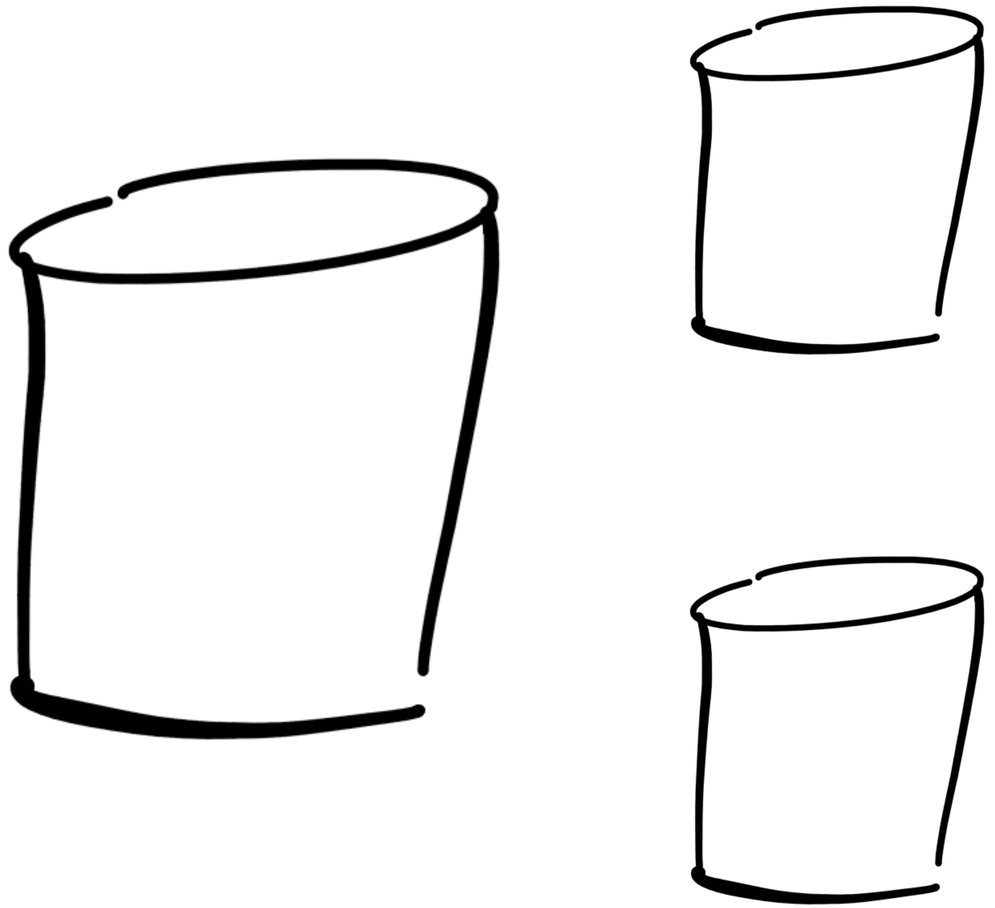
When Life is Good

Tim?

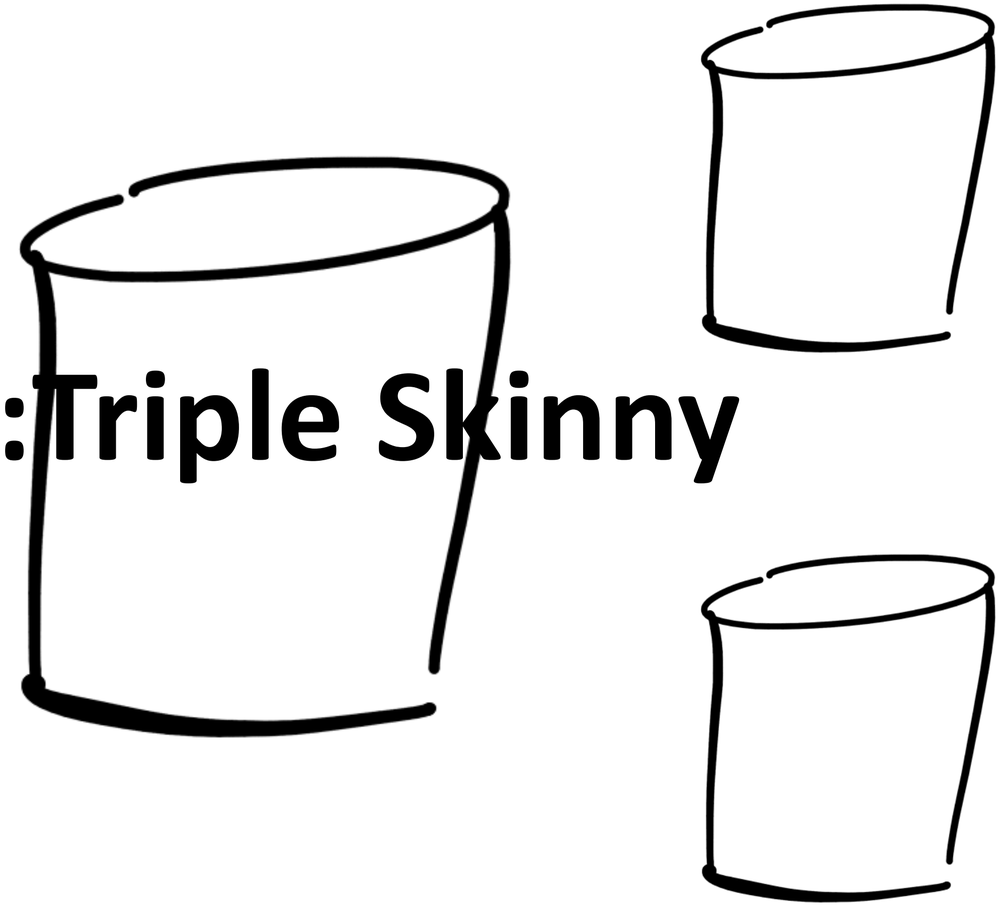
When Life is Good



When Life is Busier

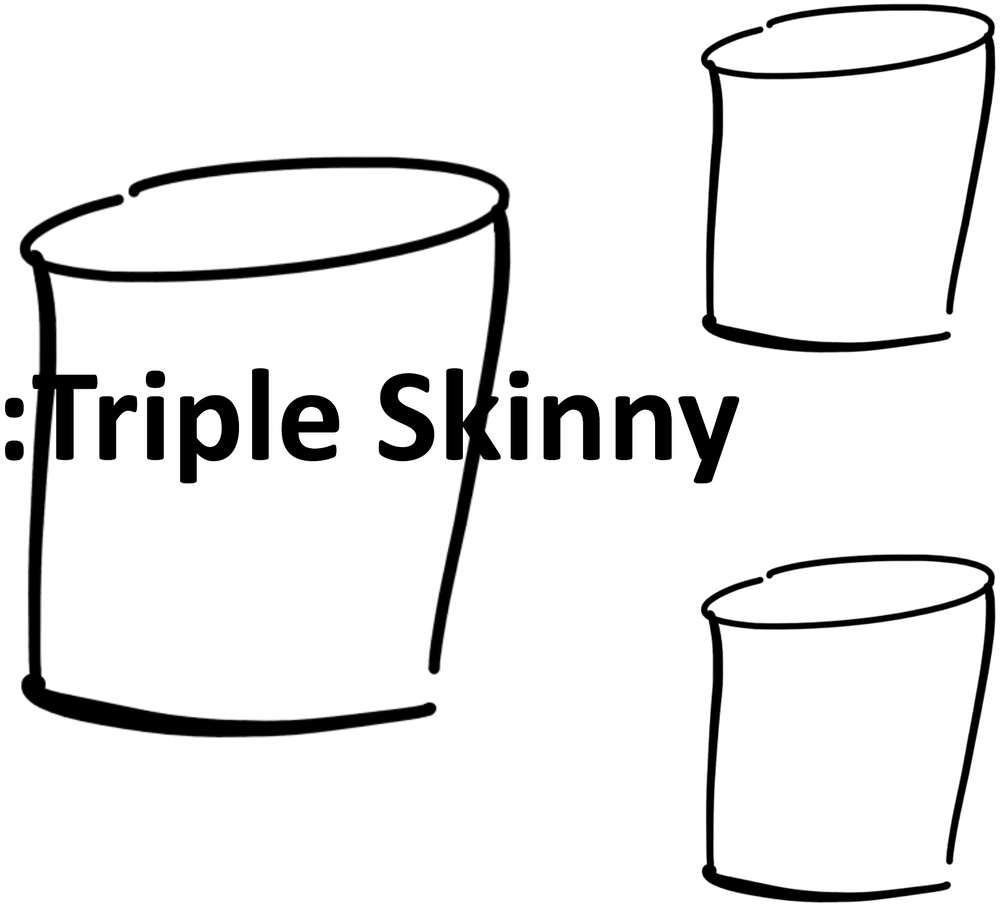


When Life is Busier



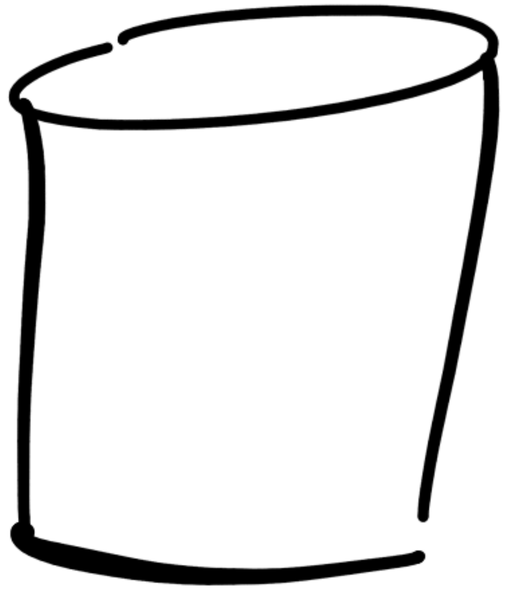
Tim:Triple Skinny

When Life is Busier



Tim:Triple Skinny

When Life is Busier

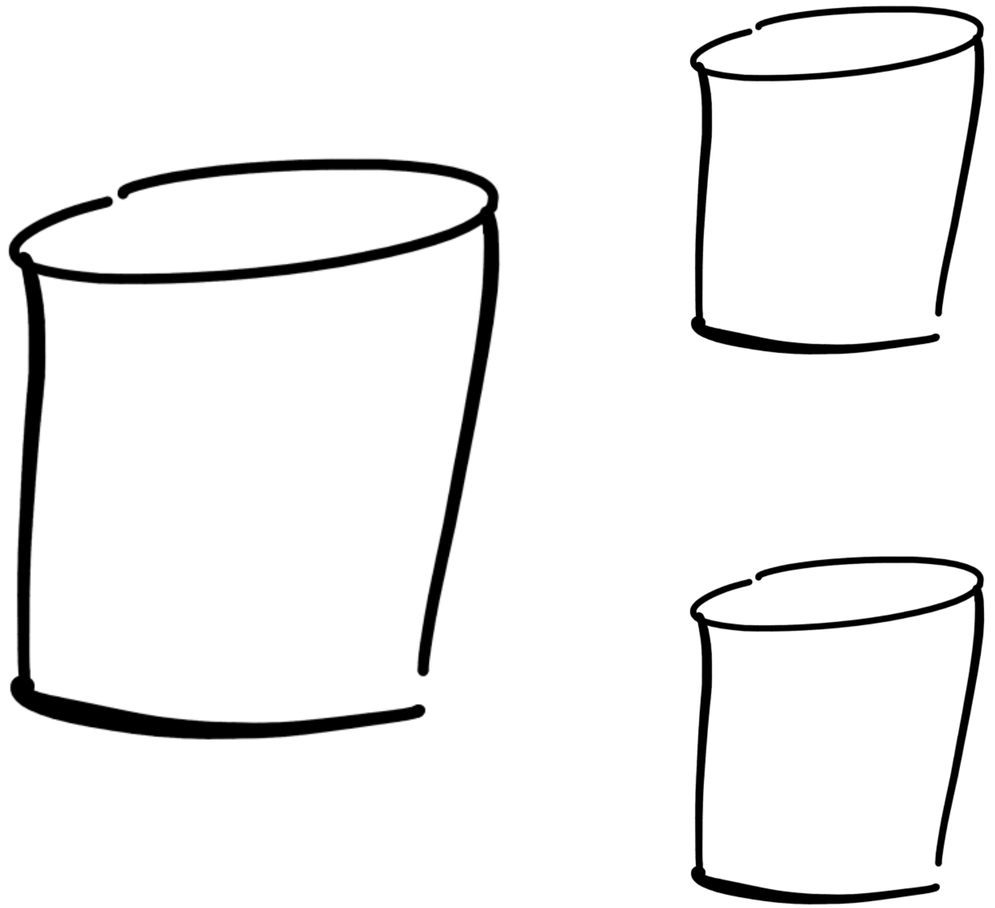


Tim:Triple Skinny

Tim:Triple Skinny

When Life is Busier

Tim?

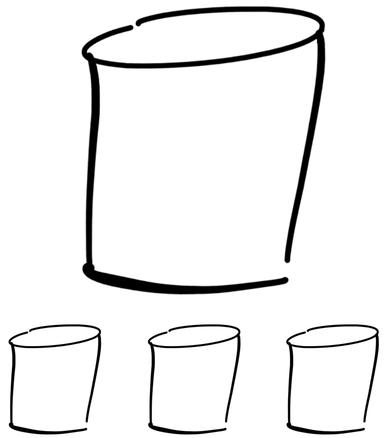


Replication Problems

- Complexity
- Consistency

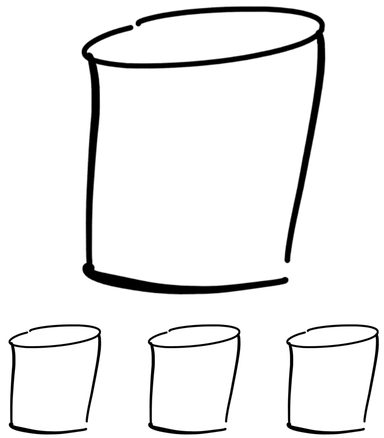
When Life is REALLY busy

Aaron-Faye

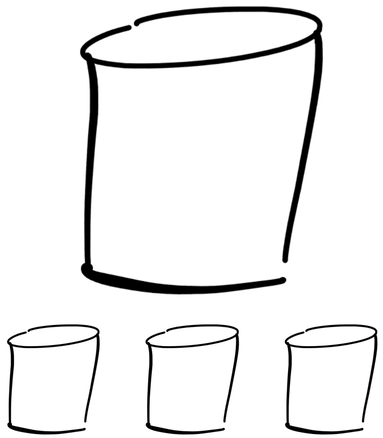


When Life is REALLY busy

Aaron-Faye

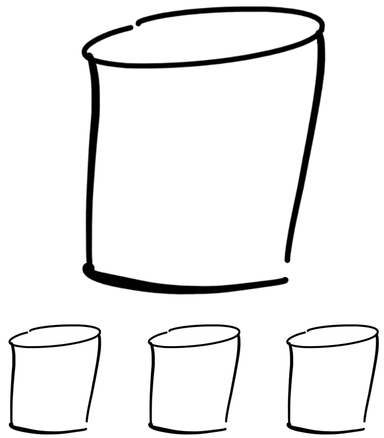


Faye-Nancy

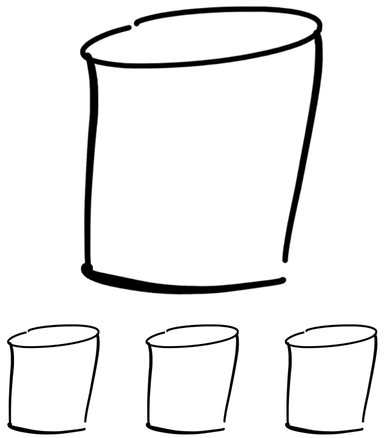


When Life is REALLY busy

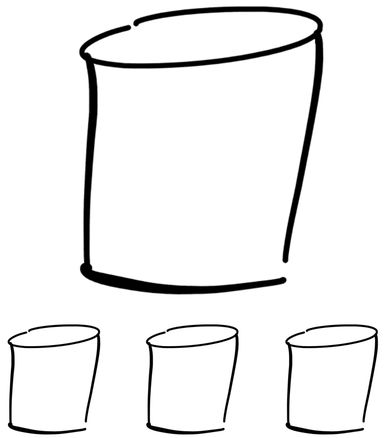
Aaron-Faye



Faye-Nancy



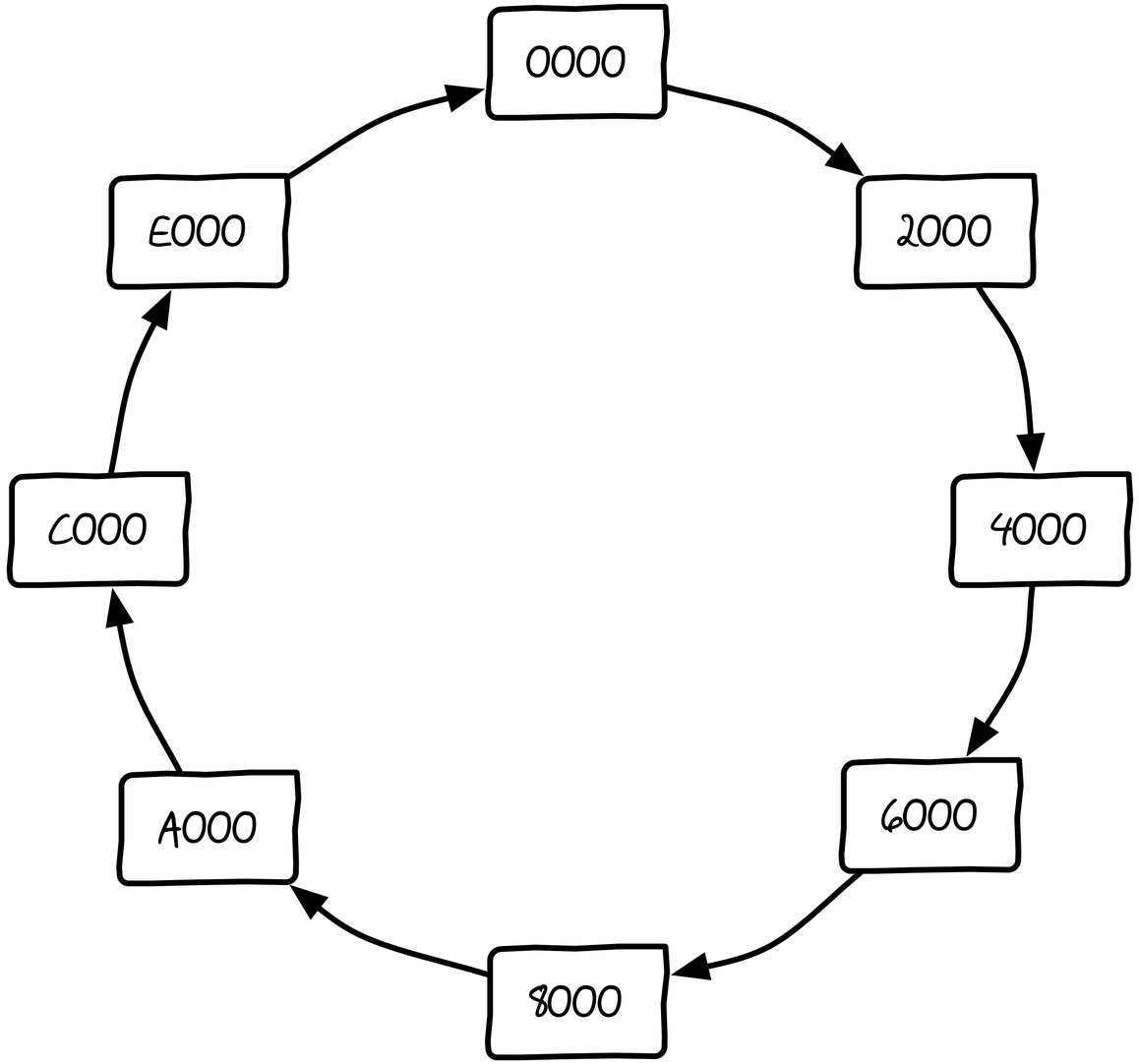
Nancy-Zed



Sharding Problems

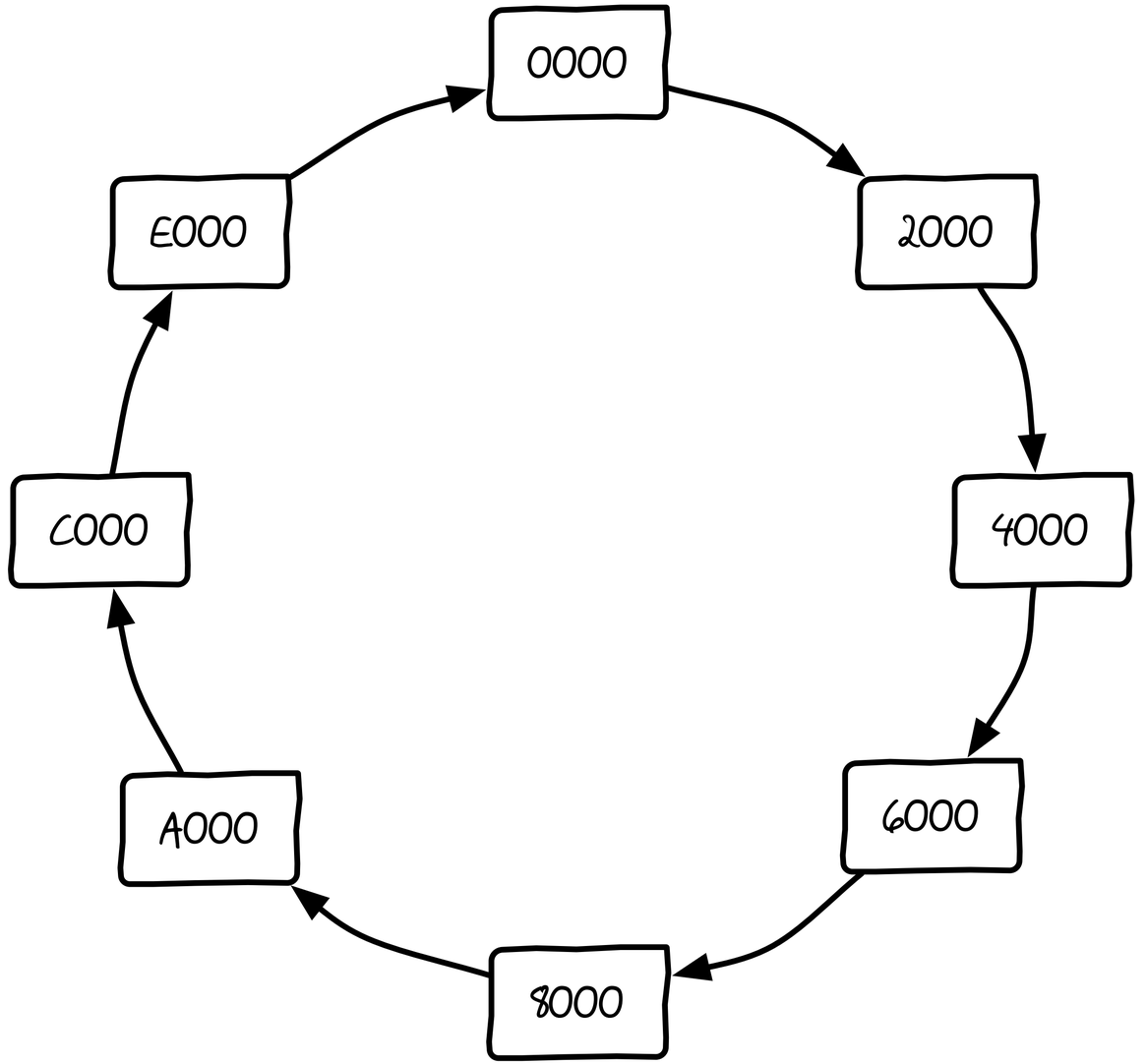
- More Complexity
- Broken data model
- Limited data access patterns

Consistent Hashing



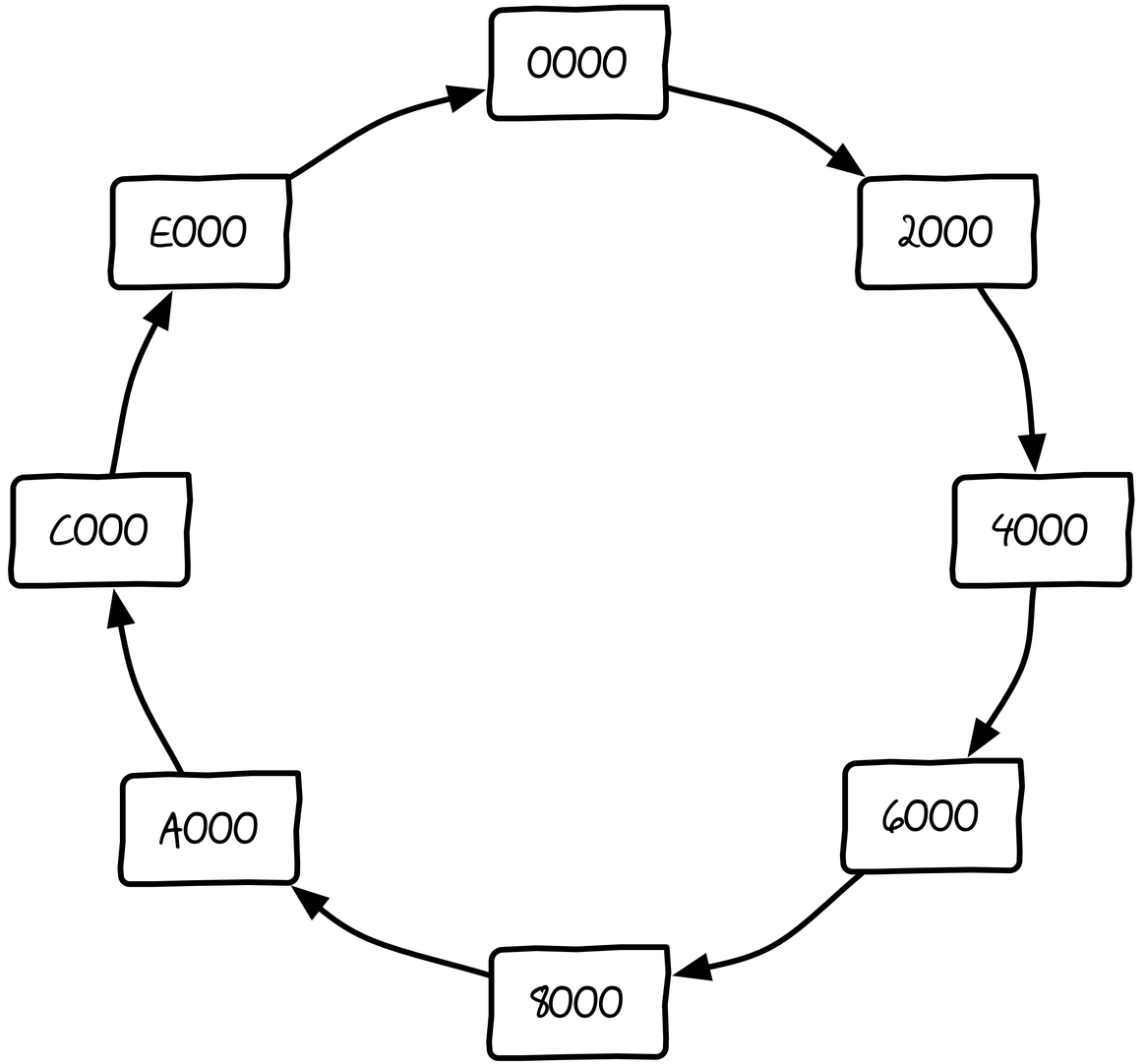
Consistent Hashing

Tim:Americano



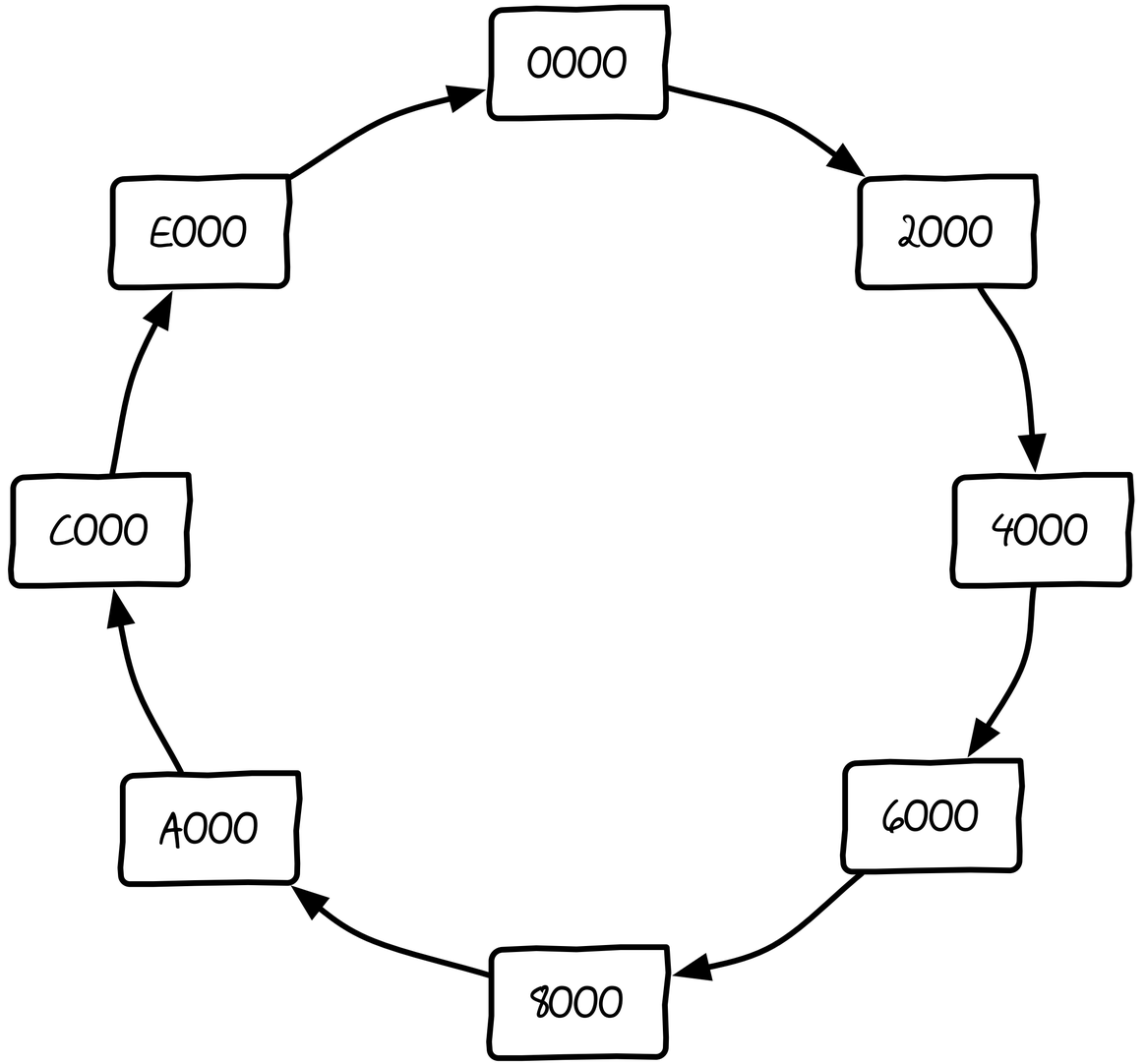
Consistent Hashing

9F72:Americano



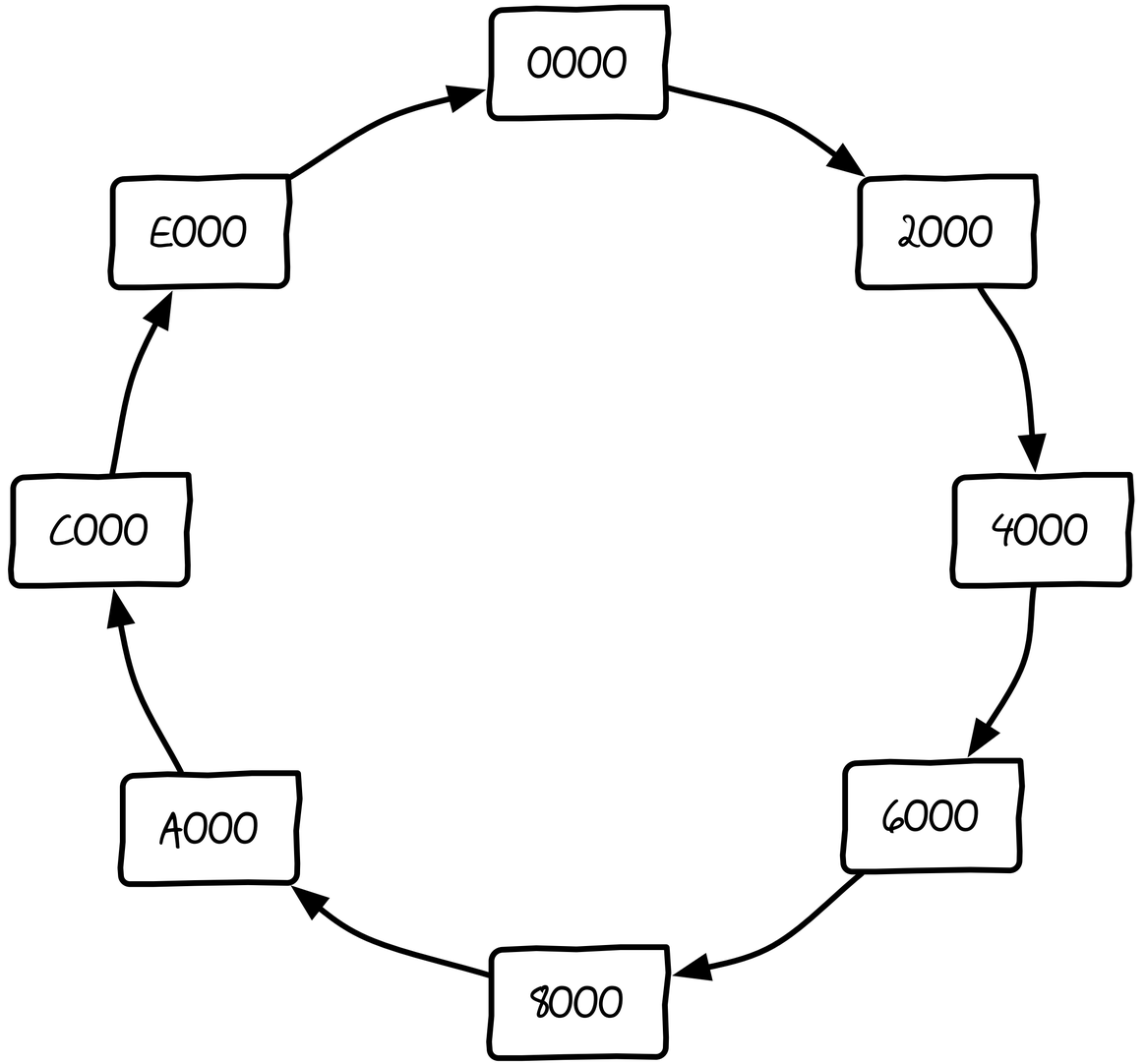
Consistent Hashing

Tim?

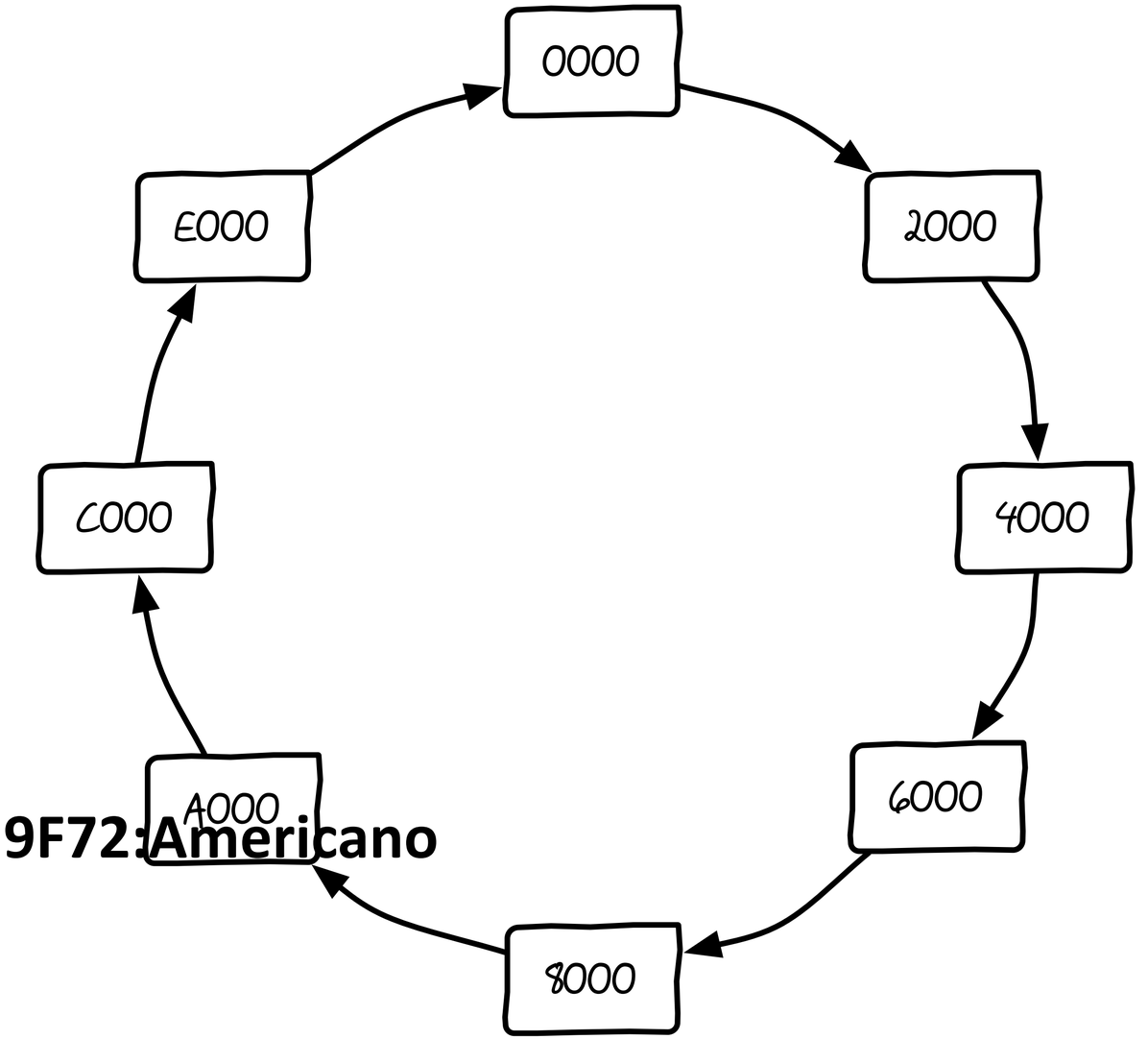


Consistent Hashing

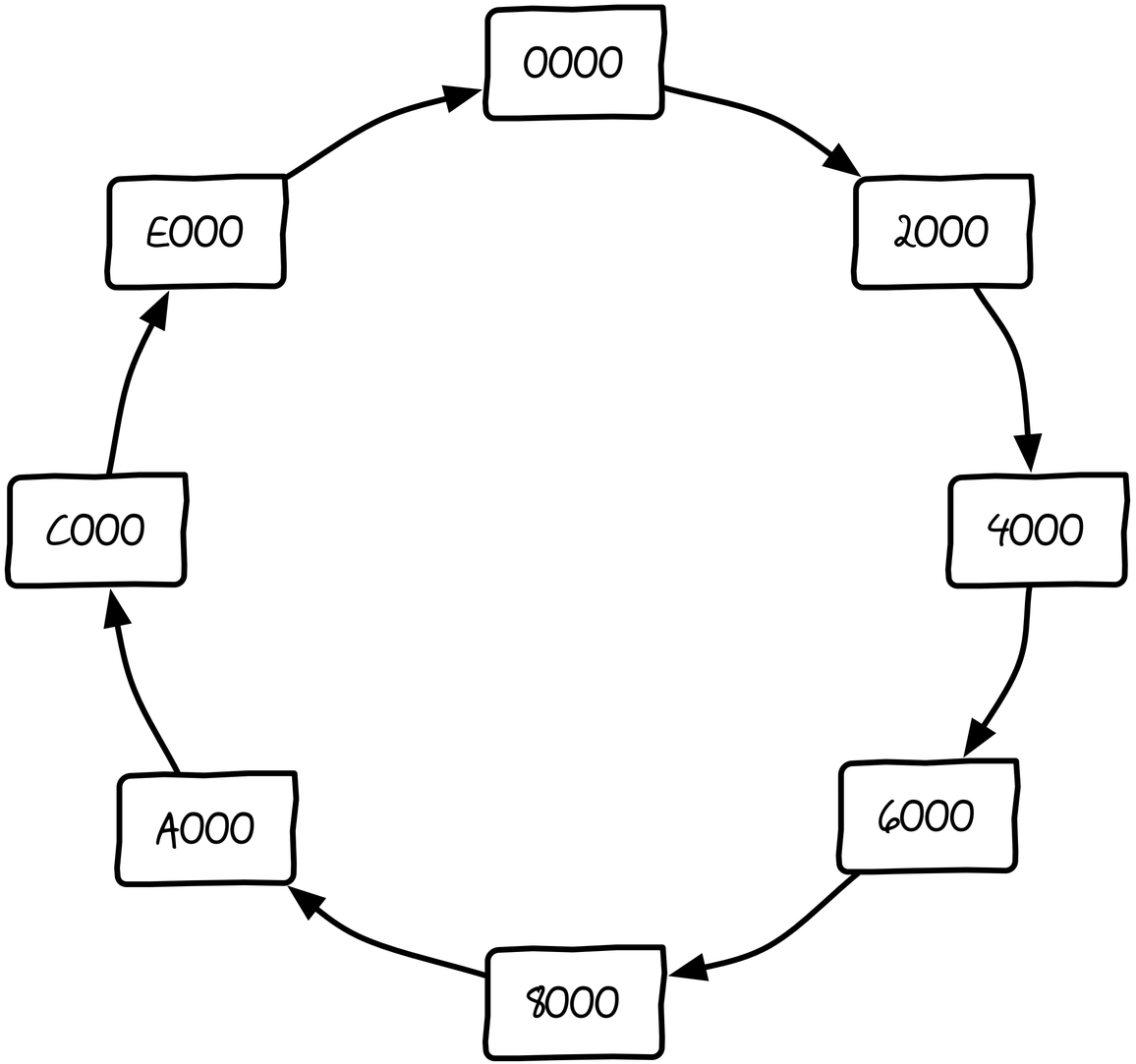
9F72?



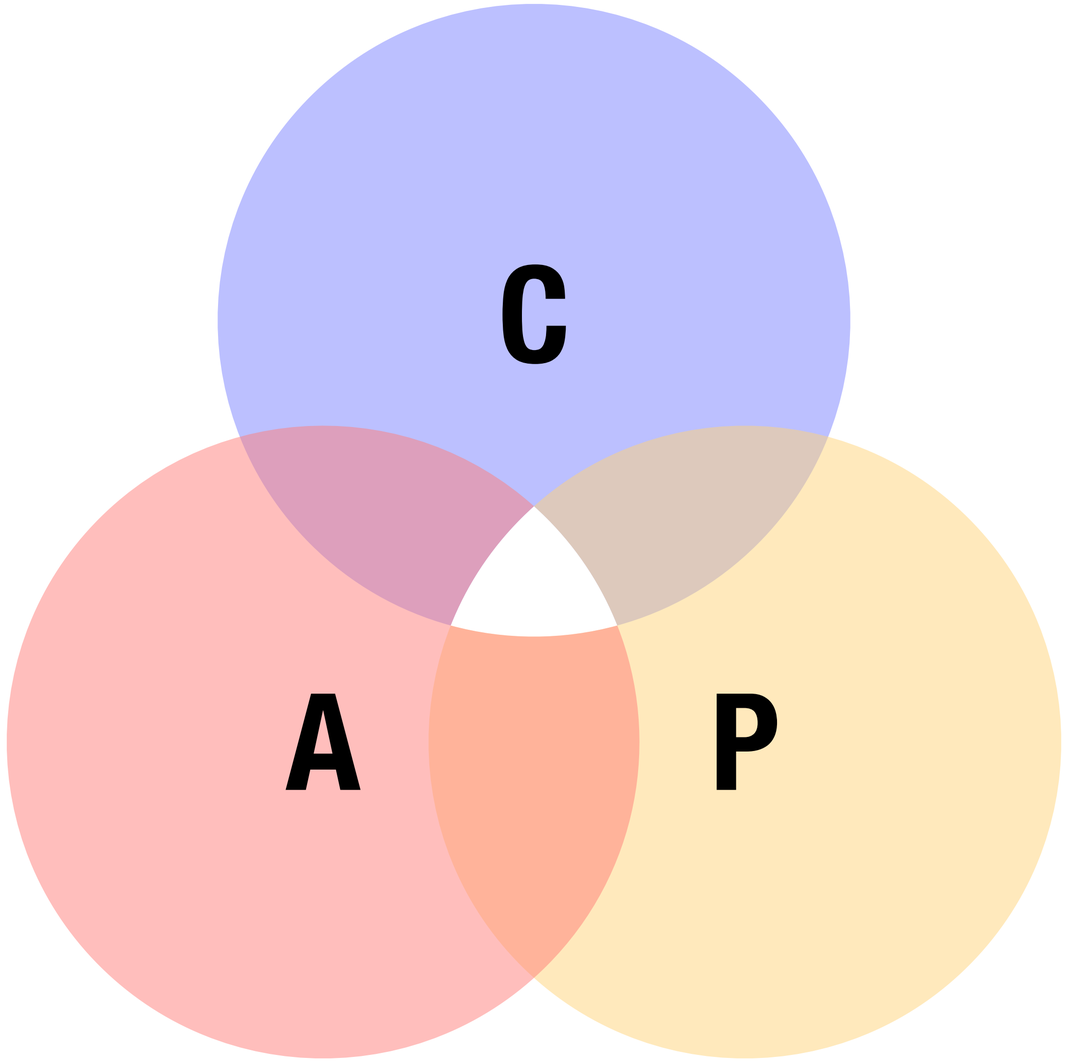
Consistent Hashing



Consistent Hashing



CAP Theorem



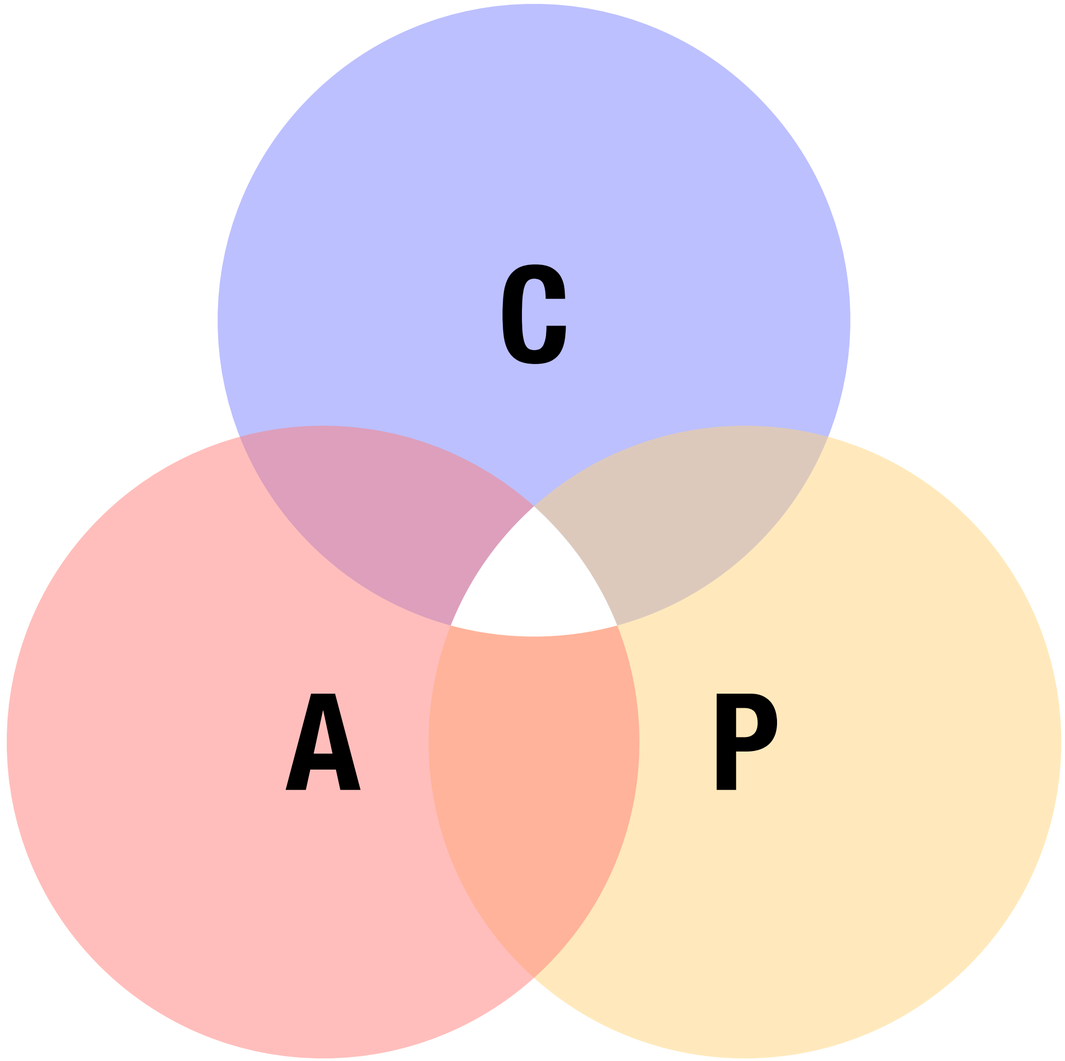
CAP Theorem

- Consistency
- Availability
- Partition Tolerance

CAP Theorem

- Shared writing project
- Coffee shop closes
- Synchronizing over the phone
- Battery dies
- Status report!

CAP Theorem



Distributed Transactions

Distributed Transactions

www.eaipatterns.com/ramblings/18_starbucks.html

Enterprise Integration Patterns

Gregor's Ramblings

HOME • PATTERNS • RAMBLINGS • ARTICLES • TALKS • DOWNLOAD • LINKS • BOOKS • CONTACT

Starbucks Does Not Use Two-Phase Commit

November 19, 2004

Hotto Cocoa o Kudasai

I just returned from a 2 week trip to Japan. One of the more familiar sights was the ridiculous number of Starbucks (スターバックス) coffee shops, especially around Shinjuku and Roppongi. While waiting for my "Hotto Cocoa" I started to think about how Starbucks processes drink orders. Starbucks, like most other businesses is primarily interested in maximizing throughput of orders. More orders equals more revenue. As a result they use asynchronous processing. When you place your order the cashier marks a coffee cup with your order and places it into the queue. The queue is quite literally a queue of coffee cups lined up on top of the espresso machine. This queue decouples cashier and barista and allows the cashier to keep taking orders even if the barista is backed up for a moment. It allows them to deploy multiple baristas in a Competing Consumer scenario if the store gets busy.

Correlation

By taking advantage of an asynchronous approach Starbucks also has to deal with the same challenges that asynchrony inherently brings. Take for example, correlation. Drink orders are not necessarily completed in the order they were placed. This can happen for two reasons. First, multiple baristas may be processing orders using different equipment. Blended drinks may take longer than a drip coffee. Second, baristas may make multiple drinks in one batch to optimize processing time. As a result, Starbucks has a correlation problem. Drinks are delivered out of sequence and need to be matched up to the correct customer. Starbucks solves the problem with the same "pattern" we use in messaging architectures -- they use a Correlation Identifier. In the US, most Starbucks use an explicit correlation identifier by writing your name on the cup and calling it out when the drink is complete. In other countries, you have to

ABOUT ME

Hi, I am Gregor Hohpe, co-author of the book Enterprise Integration Patterns. I like to work on and write about asynchronous messaging systems, service-oriented architectures, and all sorts of enterprise computing and

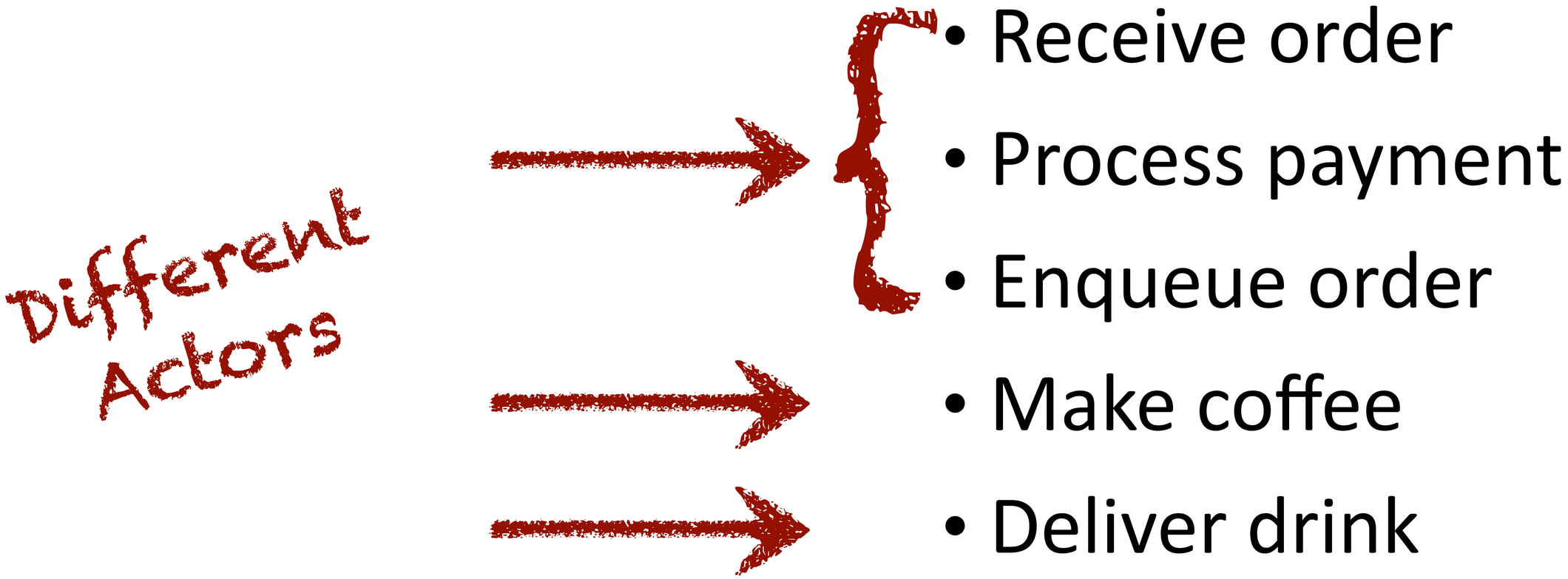
ACID Transactions

- Atomic
- Consistent
- Isolated
- Durable
- One barista
- Multiple baristas

Ordering Coffee

- Receive order
- Process payment
- Enqueue order
- Make coffee
- Deliver drink

Ordering Coffee



Ordering Coffee

- Why split up order processing?
- What can fail?
- What are the consequences of failure?
- How do we repair failure?

Ordering Coffee

- How can we design a coffee shop with atomic transactions?
- How does that limit the business?
- Why give up atomicity?

Distributed Computation

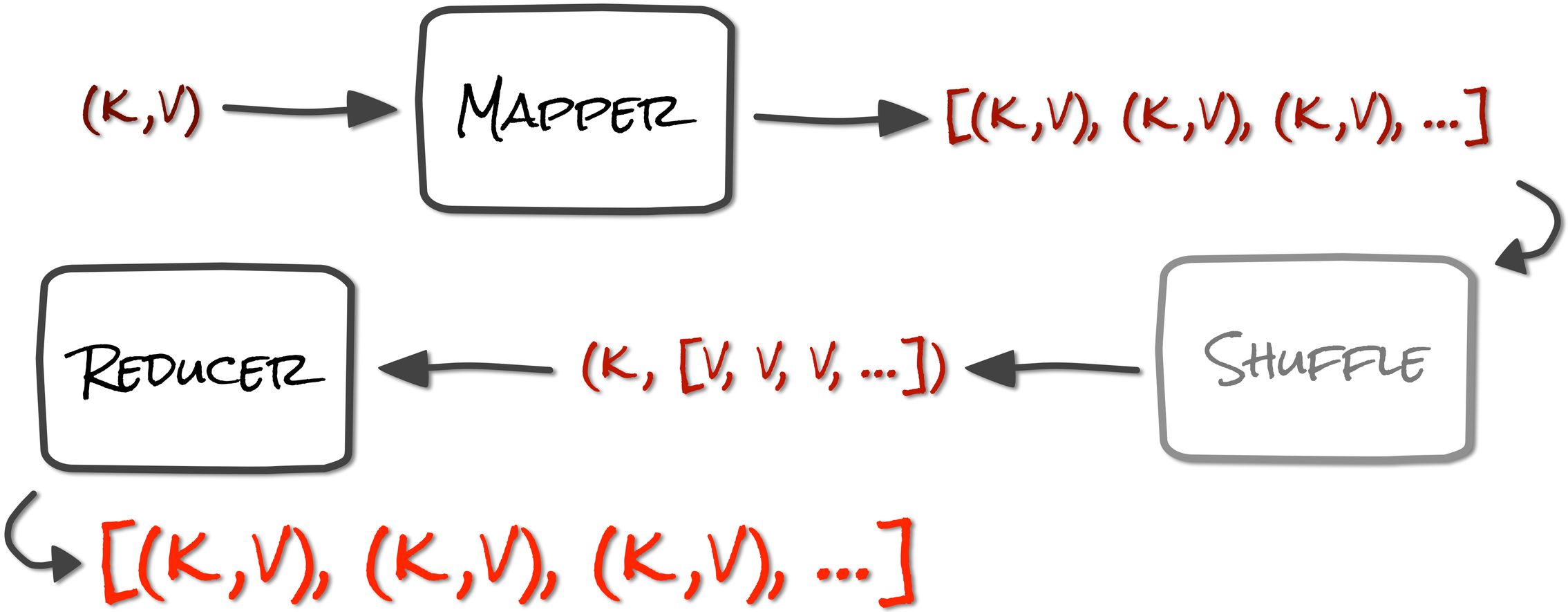
Distributed Computation

- Scatter/Gather
- MapReduce
- Hadoop
- Spark

MapReduce

- All computation in two functions: Map and Reduce
- Keep data (mostly) where it is
- Move compute to data

MapReduce



MapReduce

poems/raven.txt:

Once upon a midnight dreary, while I pondered, weak and weary,
Over many a quaint and curious volume of forgotten lore,
While I nodded, nearly napping, suddenly there came a tapping,
As of some one gently rapping, rapping at my chamber door.
"'Tis some visitor," I muttered, "tapping at my chamber door-
Only this, and nothing more."

Ah, distinctly I remember it was in the bleak December,
And each separate dying ember wrought its ghost upon the floor.
Eagerly I wished the morrow;- vainly I had sought to borrow

MapReduce

poems/raven.txt:

It shall clasp a sainted maiden whom the angels name Lenore-
Clasp a rare and radiant maiden whom the angels name Lenore."
Quoth the Raven, "Nevermore."

"Be that word our sign in parting, bird or fiend," I shrieked, upstarting-
"Get thee back into the tempest and the Night's Plutonian shore!
Leave no black plume as a token of that lie thy soul hath spoken!
Leave my loneliness unbroken!- quit the bust above my door!
Take thy beak from out my heart, and take thy form from off my door!"
Quoth the Raven, "Nevermore."

And the Raven, never flitting, still is sitting, still is sitting
On the pallid bust of Pallas just above my chamber door;
And his eyes have all the seeming of a demon's that is dreaming,
And the lamp-light o'er him streaming throws his shadow on the floor;
And my soul from out that shadow that lies floating on the floor
Shall be lifted- nevermore!

MapReduce

Map

suddenly while upon gently dreary midnight

door came a my pondered rapping tapping

a rapping there Once chamber tapping at

MapReduce

Map

suddenly:1 while:1 upon:1 gently:1 dreary:1 midnight:1

door:1 came:1 a:1 my:1 pondered:1 rapping:1 tapping:1

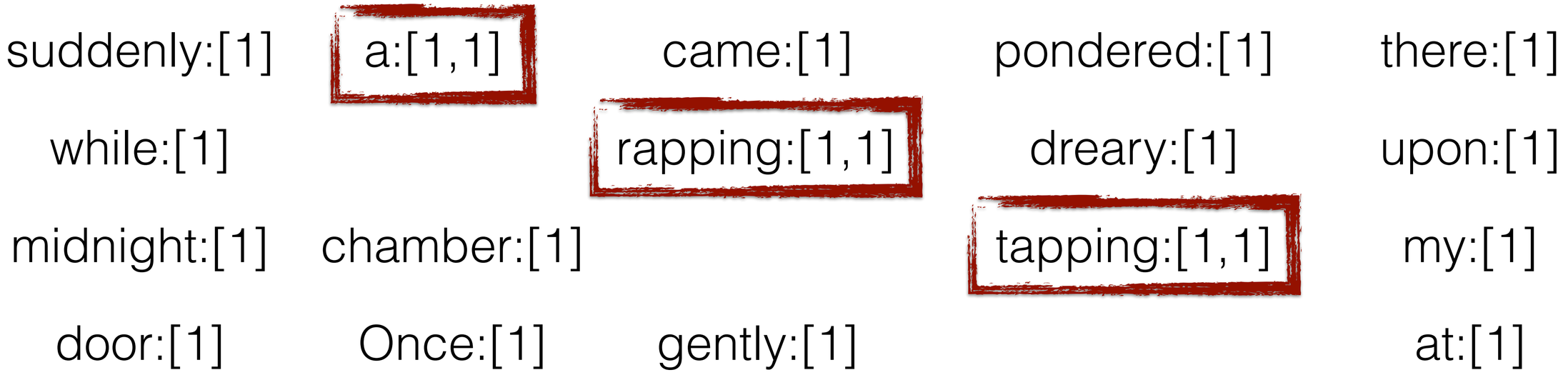
a:1 rapping:1 there:1 Once:1 chamber:1 tapping:1 at:1

MapReduce

Shuffle

MapReduce

Reduce



MapReduce

Reduce

suddenly:1	a:2	came:1	pondered:1	there:1
while:1		rapping:2	dreary:1	upon:1
midnight:1	chamber:1		tapping:2	my:1
door:1	Once:1	gently:1		at:1

Real-World MapReduce

- Hadoop
- Cloudera, Hortonworks, MapR
- Nobody write map, reduce functions
- Hive (SQL-like interface)
- Integration with BI front-ends

Hadoop

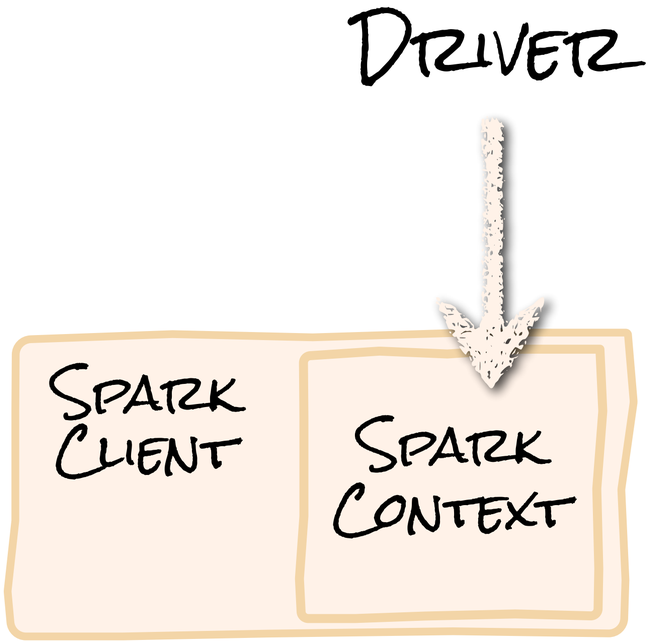
- MapReduce API
- Job Tracker, Task Tracker
- Distributed Filesystem (HDFS)
- Enormous ecosystem

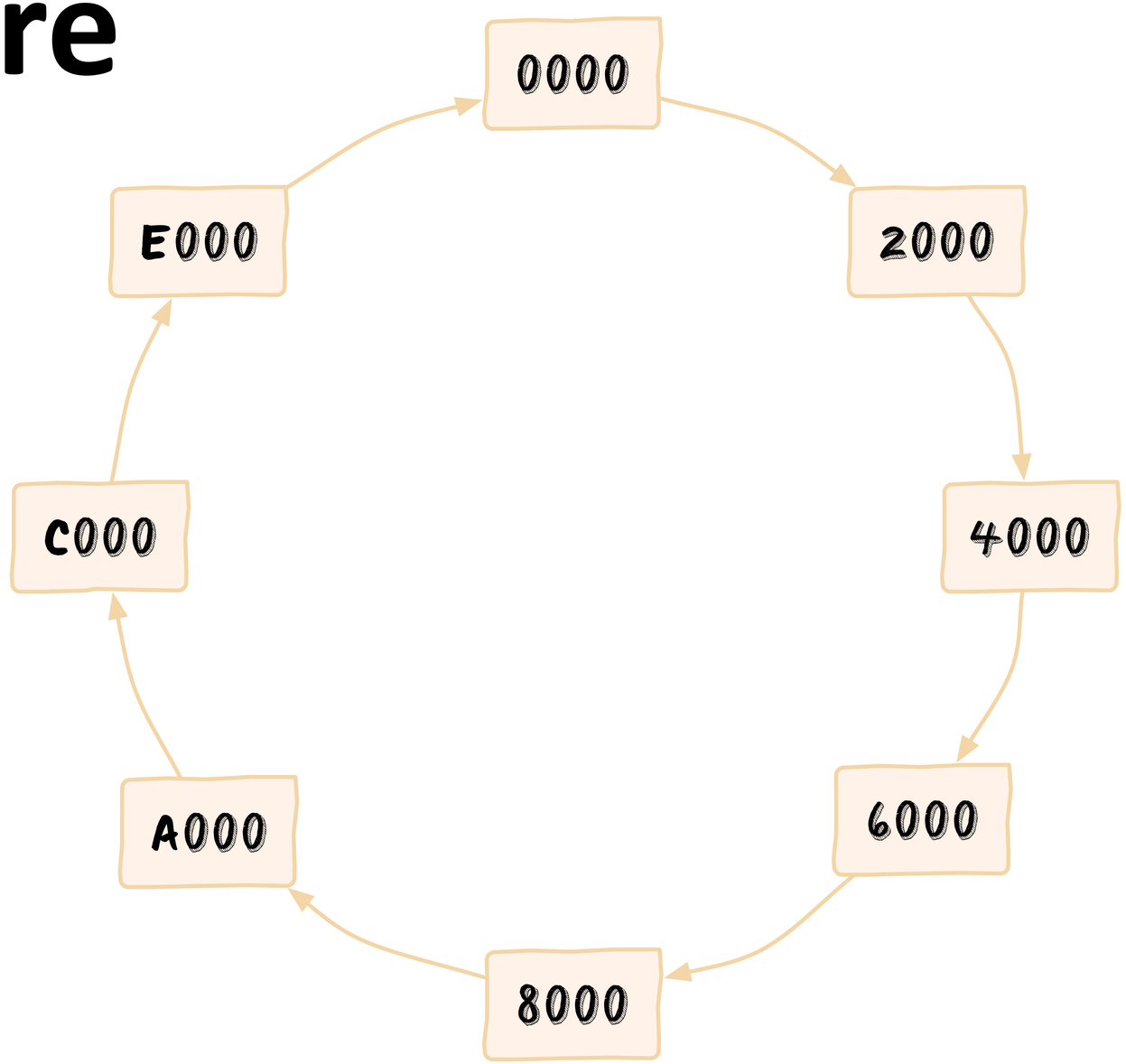
Spark

Spark

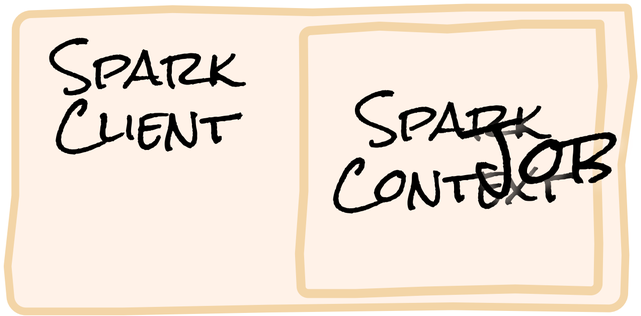
- Scatter/gather paradigm (similar to MapReduce)
- More general data model (RDDs)
- More general programming model (transform/action)
- Storage agnostic

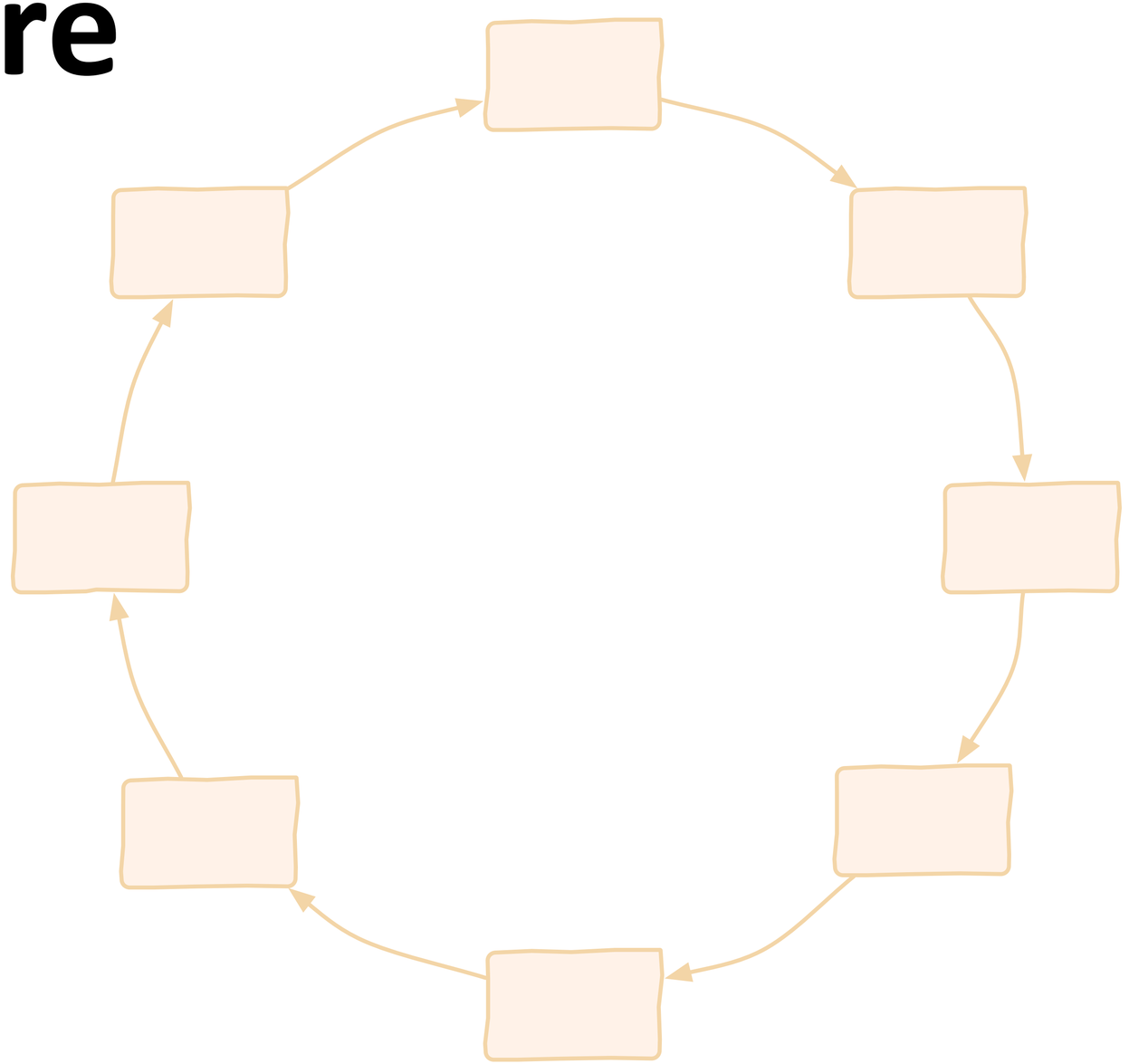
Spark Architecture



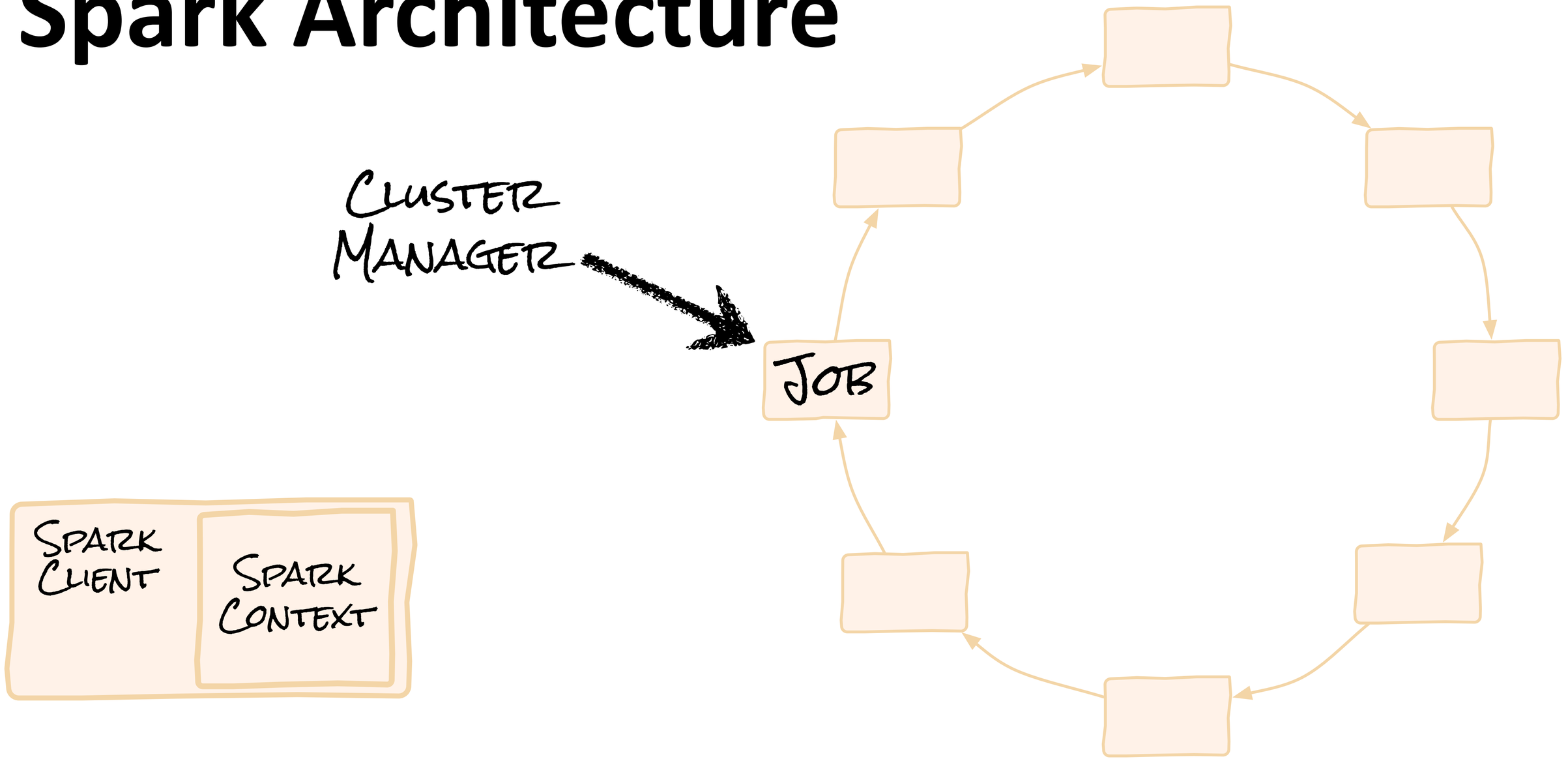


Spark Architecture

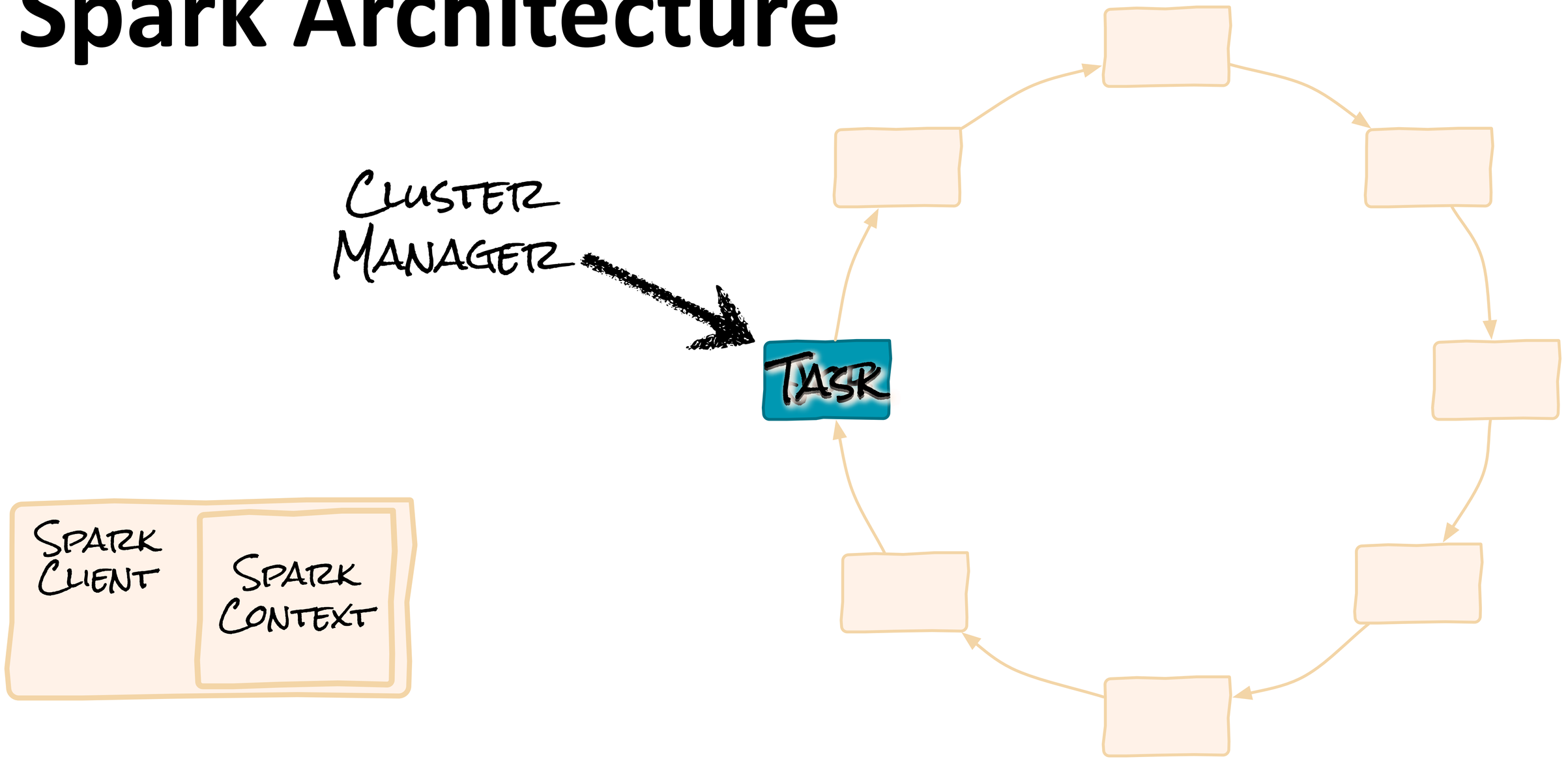




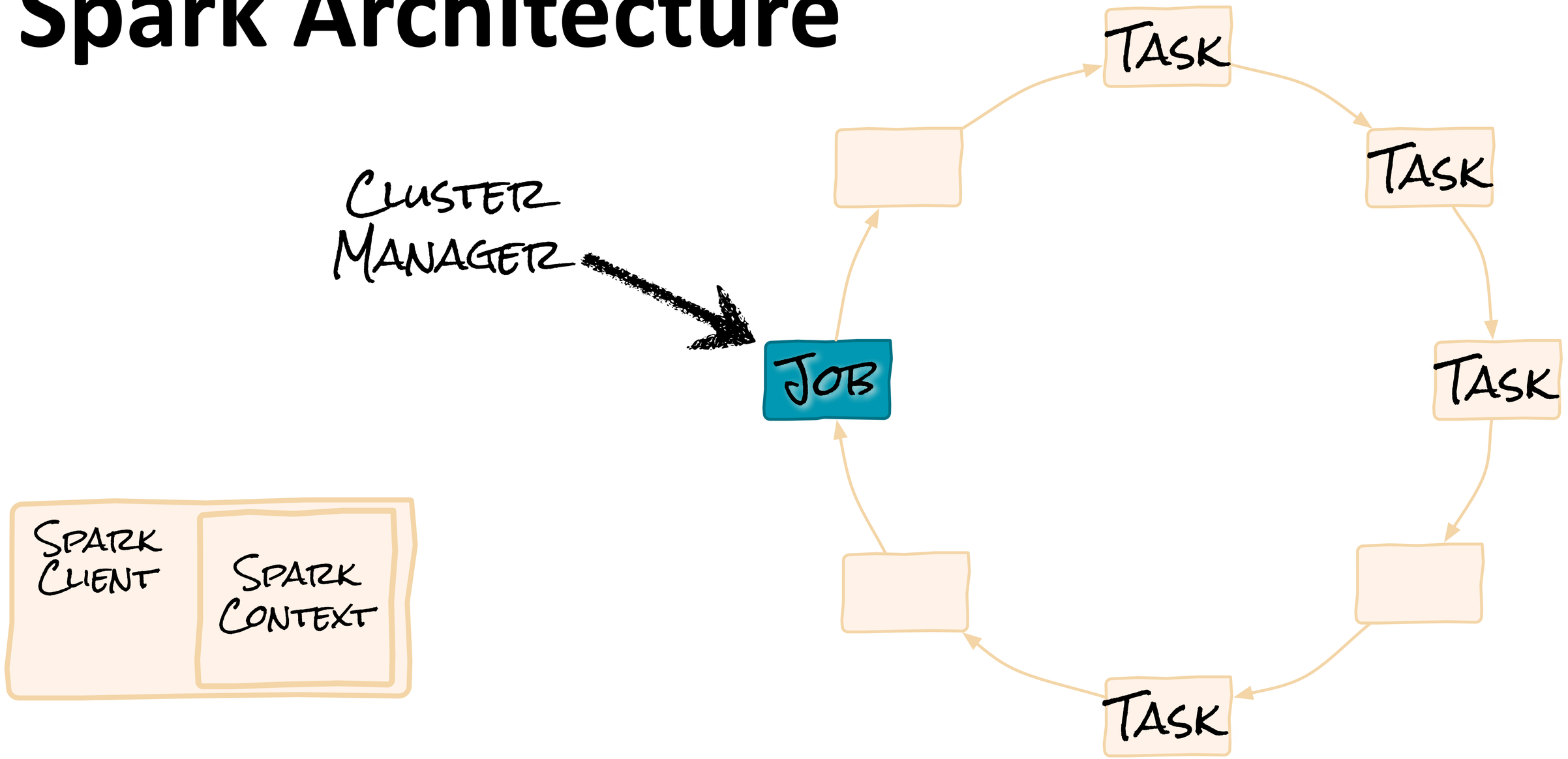
Spark Architecture



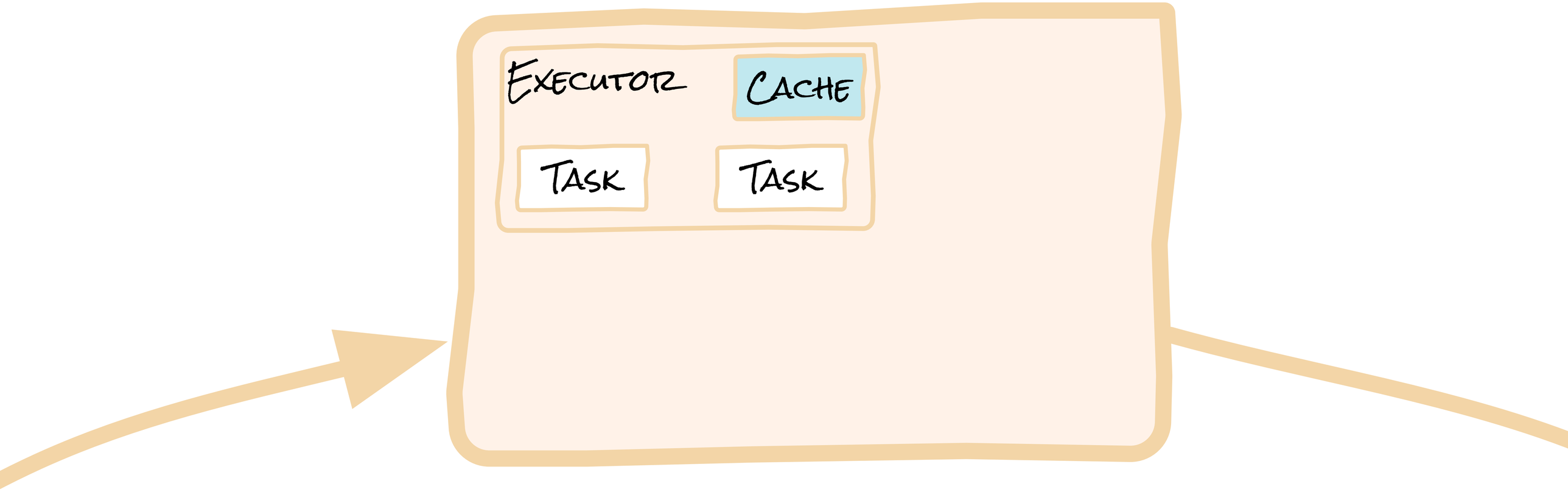
Spark Architecture



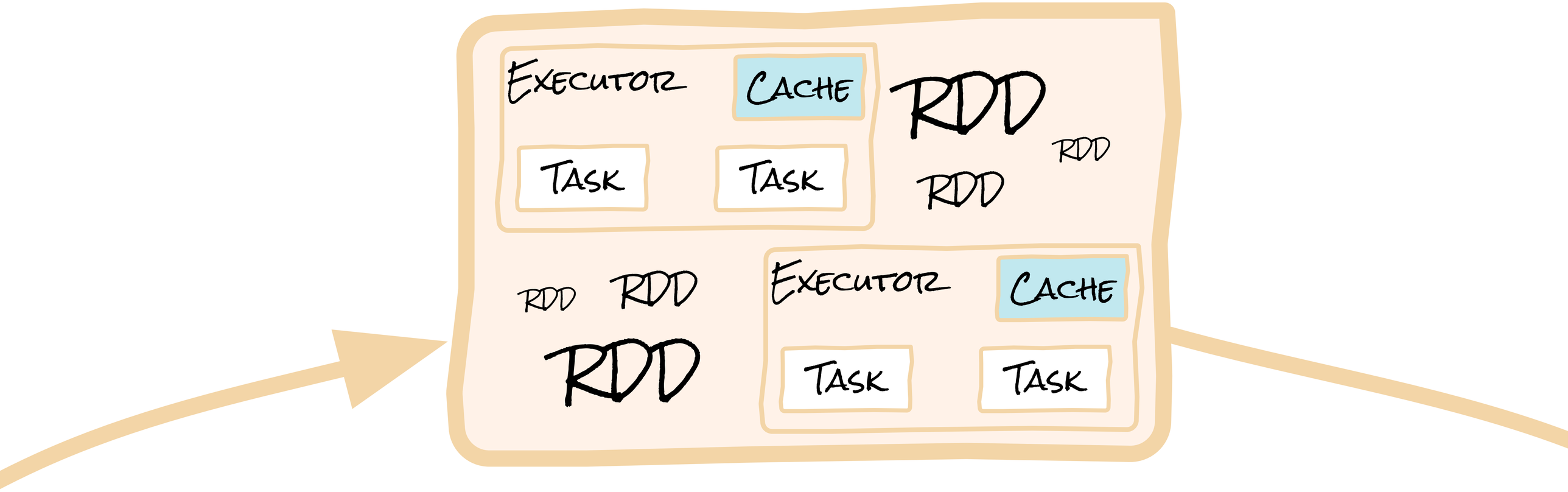
Spark Architecture



Spark Worker Node

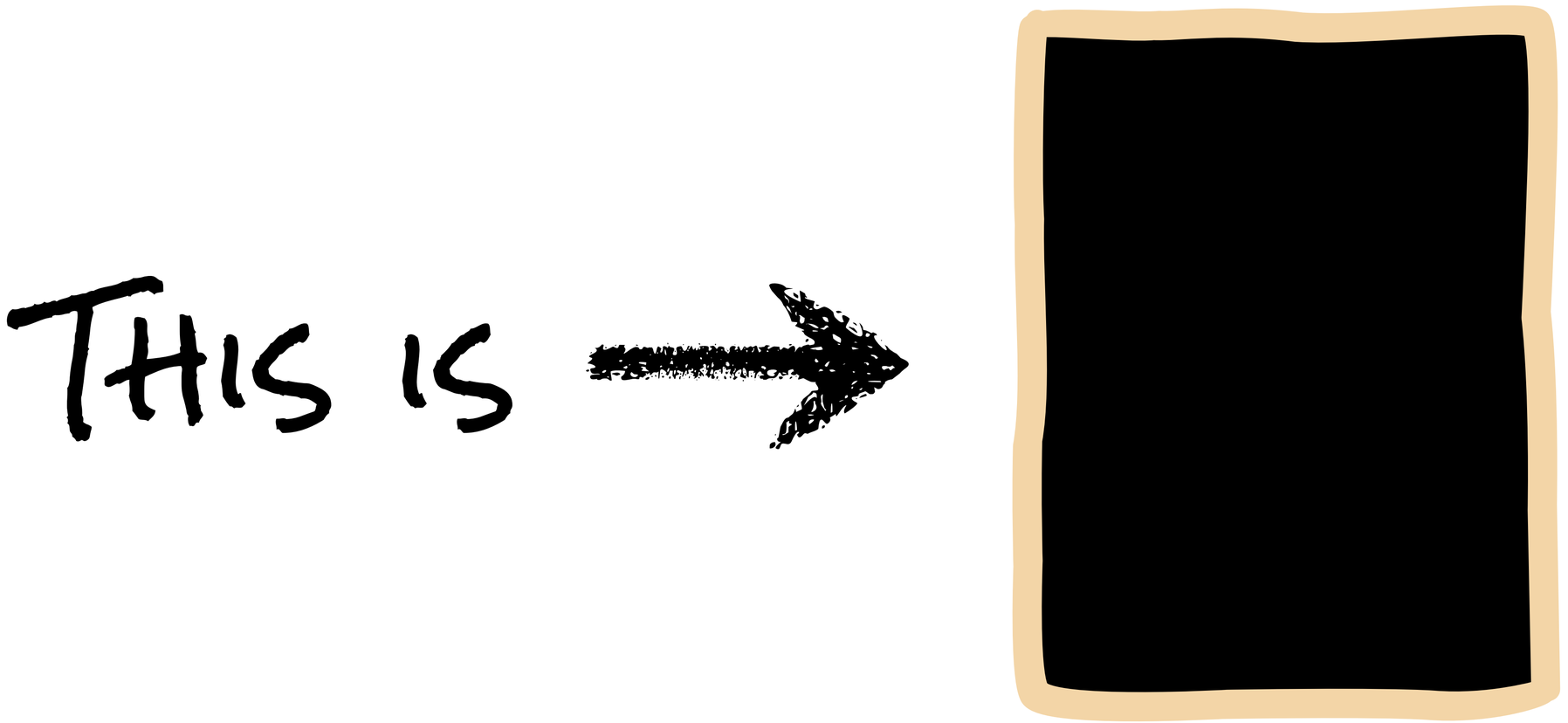


Spark Worker Node



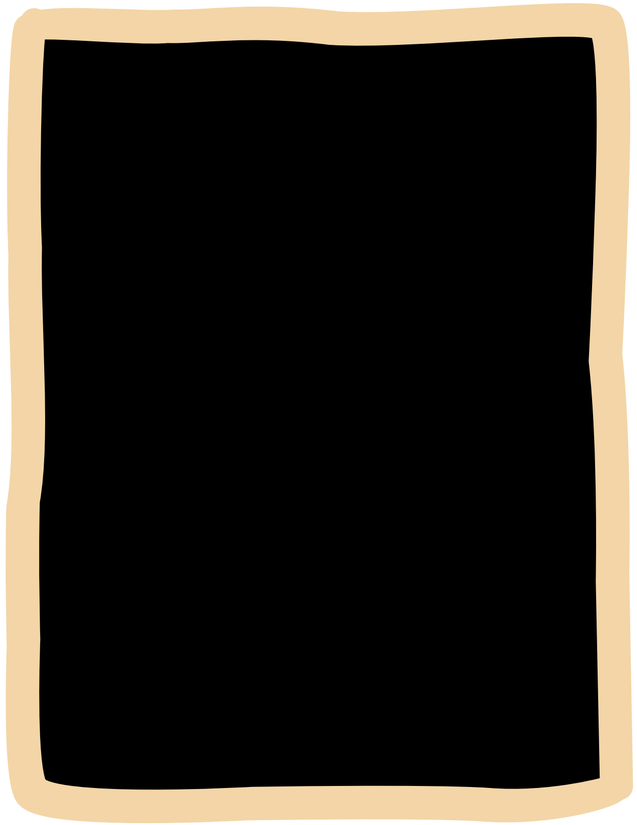
What’s an RDD?

What's an RDD?

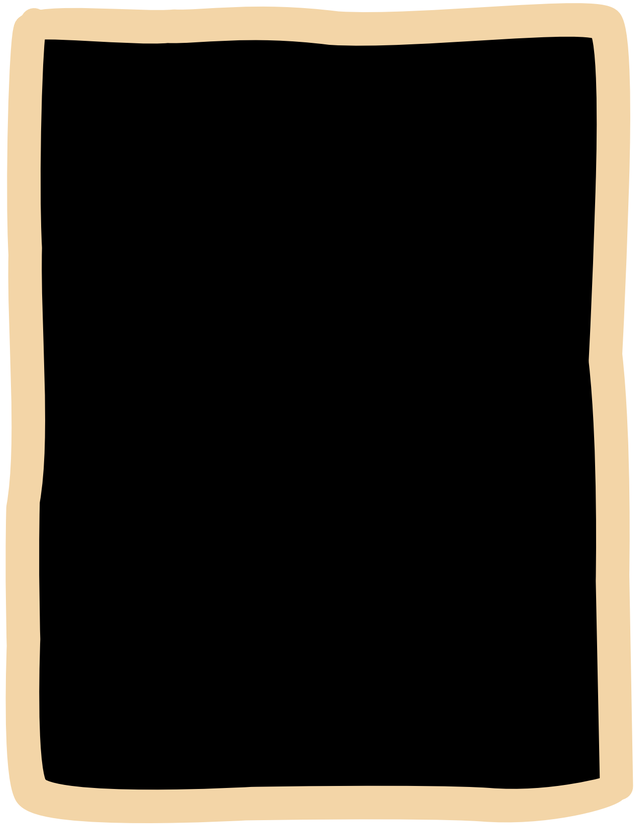


What’s an RDD?

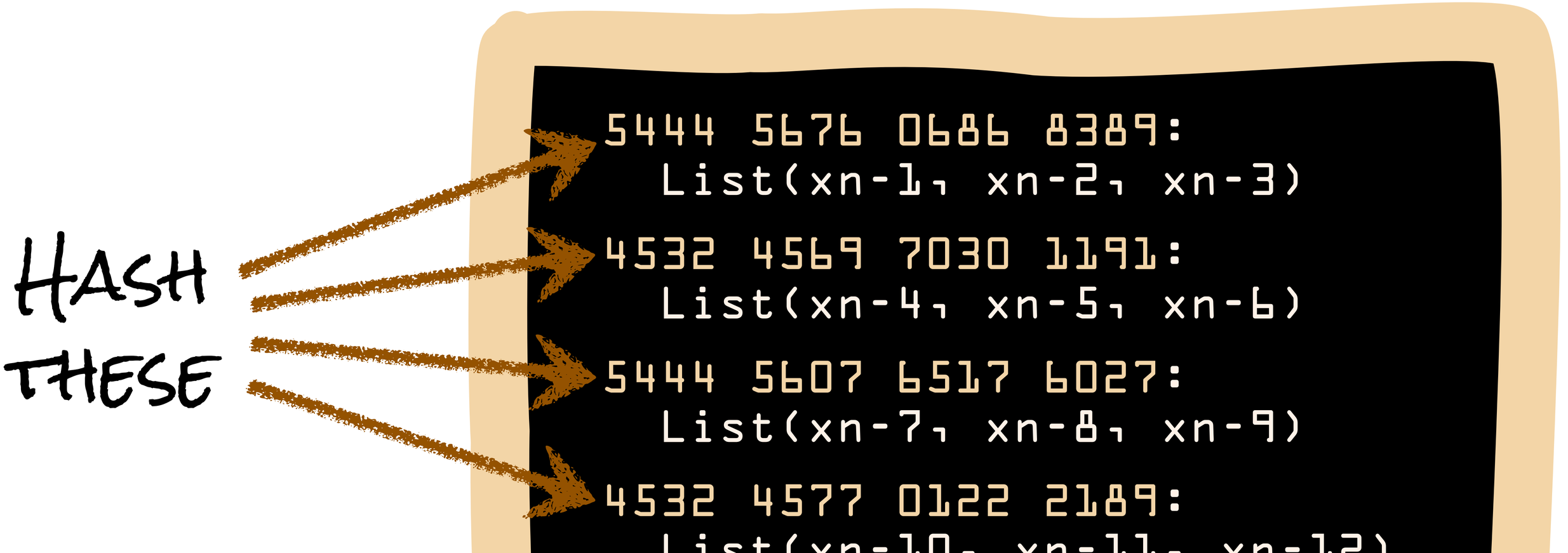
- Bigger than a computer
- Read from an input source
- Output of a pure function
- Immutable
- Typed
- Ordered
- Lazily evaluated
- Partitioned
- Collection of things



Distributed, how?



Distributed, how?



Distributed Coordination

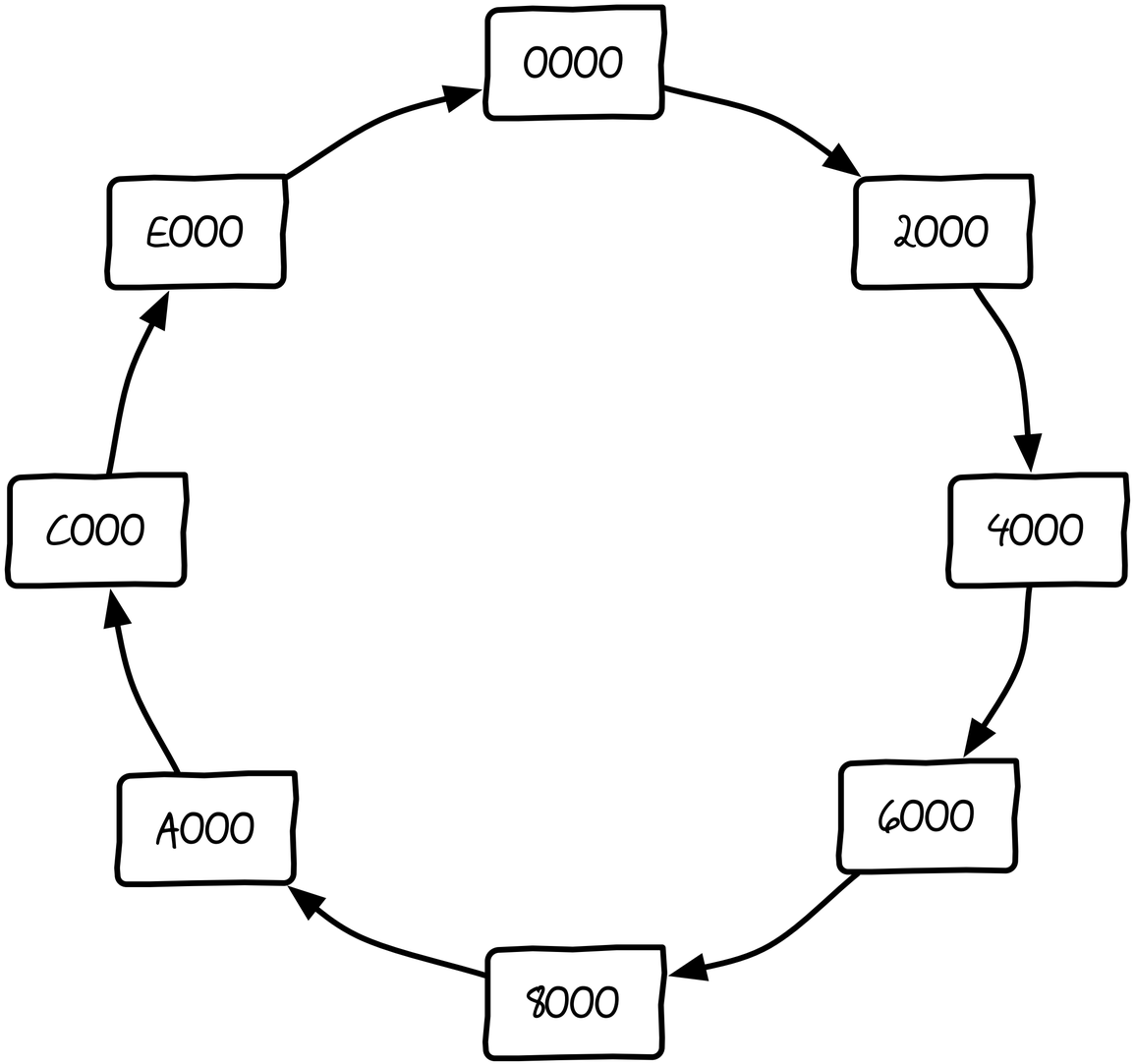
Distributed Coordination

- NTP
- Epidemic Protocols
- Paxos

NTP

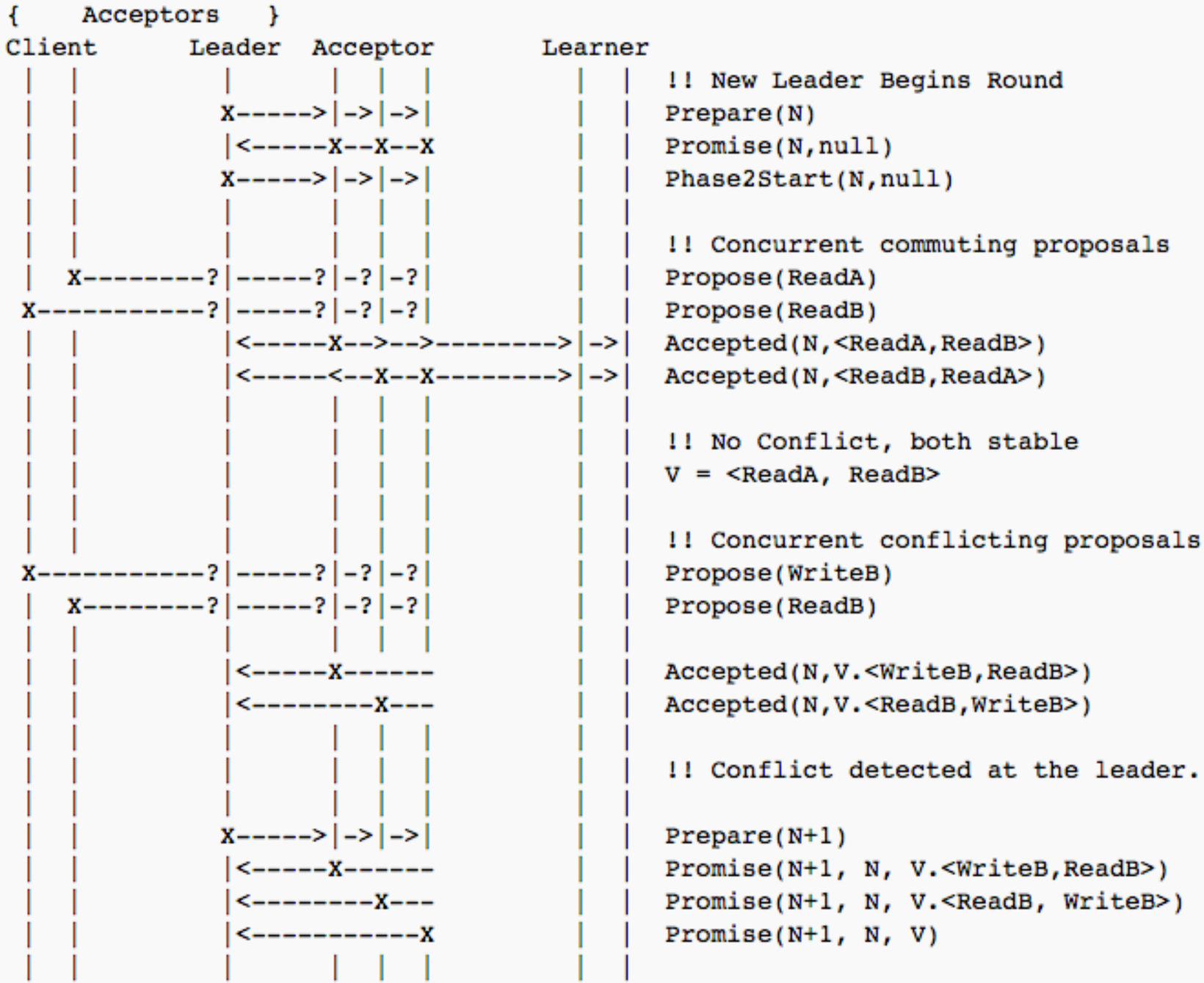
- Distributed time synchronization
- A network of tiered clocks
- Synchronization algorithm
- Often accurate to ±10ms

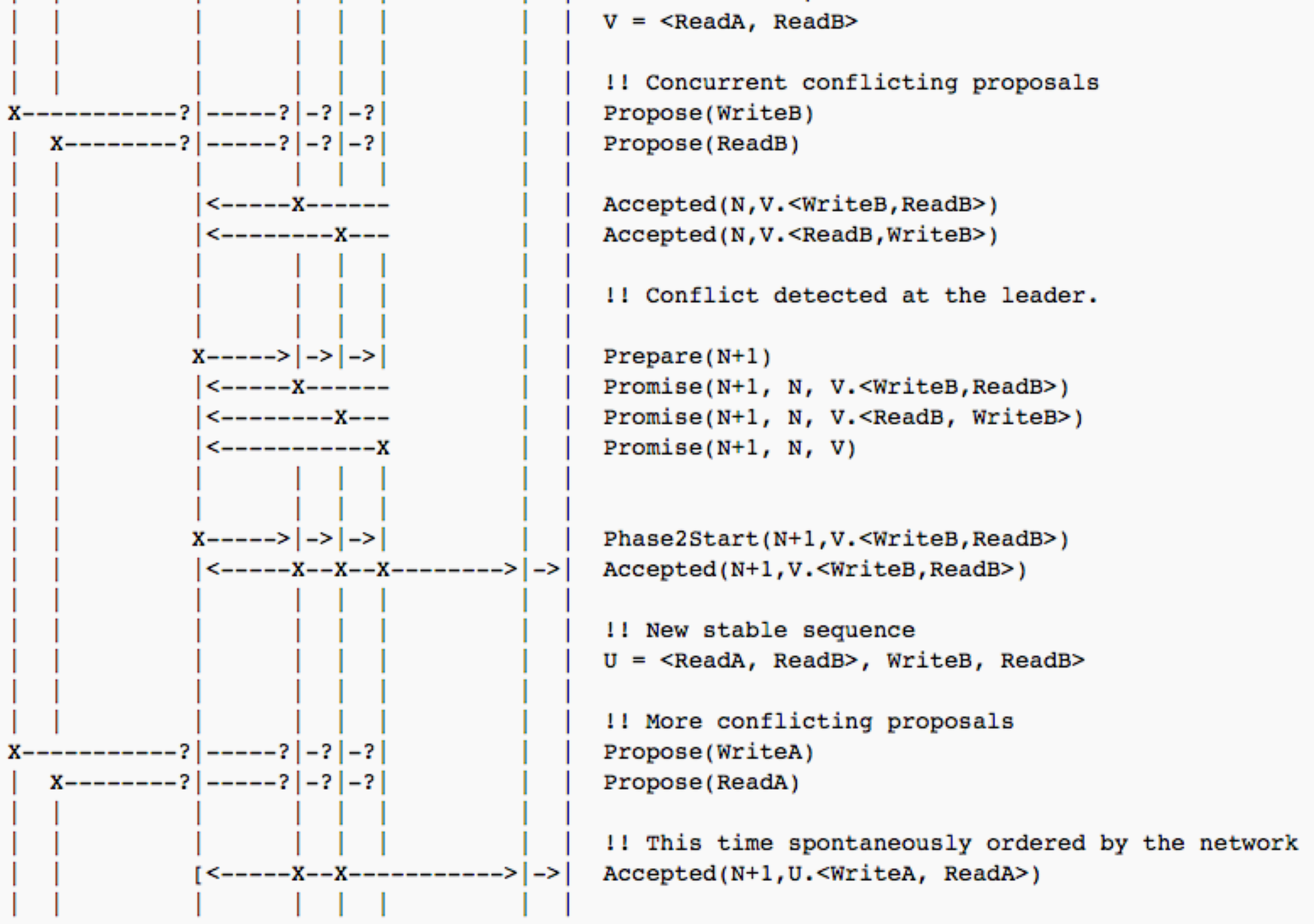
Gossip



Paxos

- Distributed log protocol
- Assumes distributed, unreliable nodes
- Performs leader election


{ Acceptors }
Client
Leader
Acceptor
Learner
!! New Leader Begins Round
Prepare(N)
Promise(N,null)
Phase2Start(N,null)
!! Concurrent commuting proposals
Propose(ReadA)
Propose(ReadB)
Accepted(N,<ReadA,ReadB>)
Accepted(N,<ReadB,ReadA>)
!! No Conflict, both stable
V = <ReadA, ReadB>
!! Concurrent conflicting proposals
Propose(WriteB)
Propose(ReadB)
Accepted(N,V.<WriteB,ReadB>)
Accepted(N,V.<ReadB,WriteB>)
!! Conflict detected at the leader.
Prepare(N+1)
Promise(N+1, N, V.<WriteB,ReadB>)
Promise(N+1, N, V.<ReadB, WriteB>)
Promise(N+1, N, V)


V = <ReadA, ReadB>
!! Concurrent conflicting proposals
Propose(WriteB)
Propose(ReadB)
Accepted(N,V.<WriteB,ReadB>)
Accepted(N,V.<ReadB,WriteB>)
!! Conflict detected at the leader.
Prepare(N+1)
Promise(N+1, N, V.<WriteB,ReadB>)
Promise(N+1, N, V.<ReadB, WriteB>)
Promise(N+1, N, V)
Phase2Start(N+1,V.<WriteB,ReadB>)
Accepted(N+1,V.<WriteB,ReadB>)
!! New stable sequence
U = <ReadA, ReadB>, WriteB, ReadB>
!! More conflicting proposals
Propose(WriteA)
Propose(ReadA)
!! This time spontaneously ordered by the network
Accepted(N+1,U.<WriteA, ReadA>)

Review

- Storage
- Transactions
- Computation
- Coordination

THANK YOU!

@tlberglund