Distributed Commit Log:

Application Techniques for Transaction Processing

@david_mcneil

Strange Loop 2016

Menu amazon web services English My Account Sign Up

Amazon Kinesis Streams

Build your own custom applications that process or analyze streaming data.

Learn More | Get Started

Example: Real-time Content Recommendations for Media Sites

Send clickstream data to Kinesis Streams

Kinesis Streams stores and exposes clickstream data for processing

Custom application built on Kinesis Client Library or Spark Streaming makes real-time content recommendations

Readers see personalized content suggestions

Apache Kafka

A high-throughput distributed messaging system.

Apache Kafka is publish-subscribe messaging rethought as a distributed commit log.

Fast

A single Kafka broker can handle hundreds of megabytes of reads and writes per second from thousands of clients.

Scalable

Kafka is designed to allow a single cluster to serve as the central data backbone for a large organization. It can be elastically and transparently expanded without downtime. Data streams are partitioned and spread over a cluster of machines to allow data streams larger than the capability of any single machine and to allow clusters of co-ordinated consumers

Durable

Messages are persisted on disk and replicated within the cluster to prevent data loss. Each broker can

Goals

Model of distributed commit log

Patterns for transaction processing

Disclaimers

Kinesis vs Kafka
Some details glossed over
Patterns not "best practices"

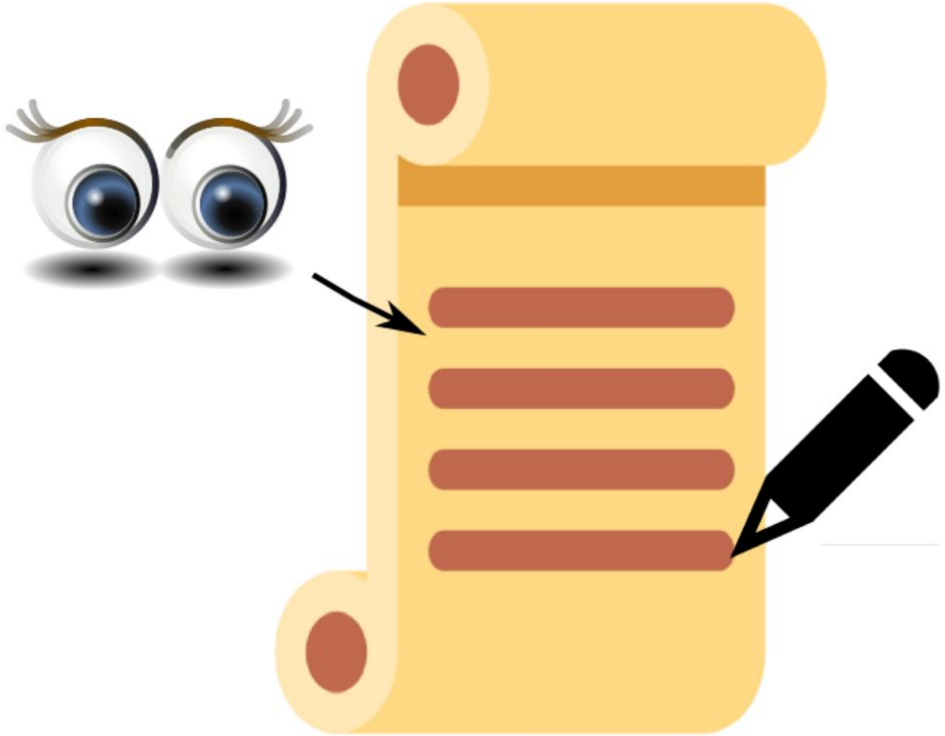
Commit Log

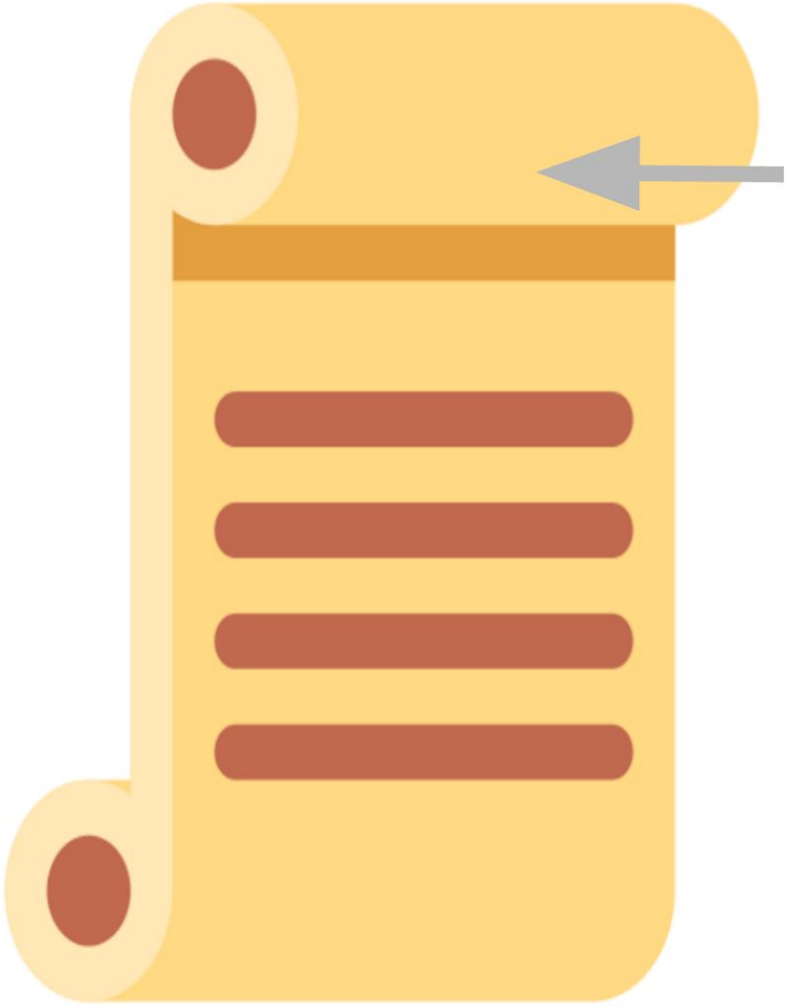
Immutable

Append only

Read in sequence

Non-destructively





Old data "trimmed"

Default 24 hours

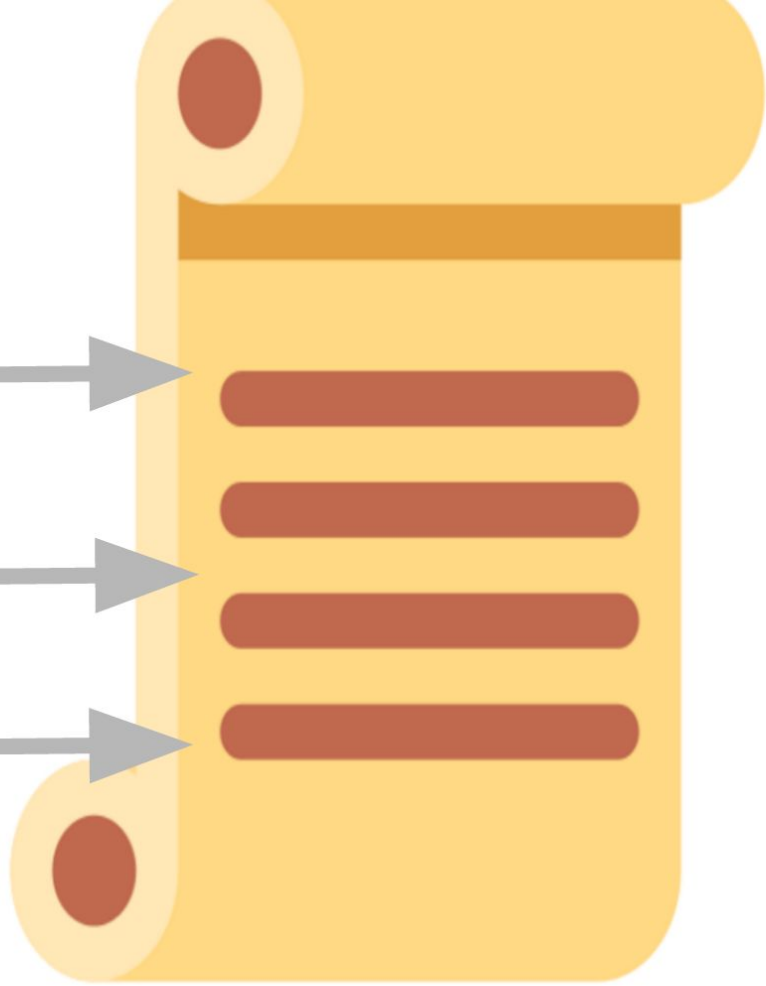
Up to 7 days

Iterator types

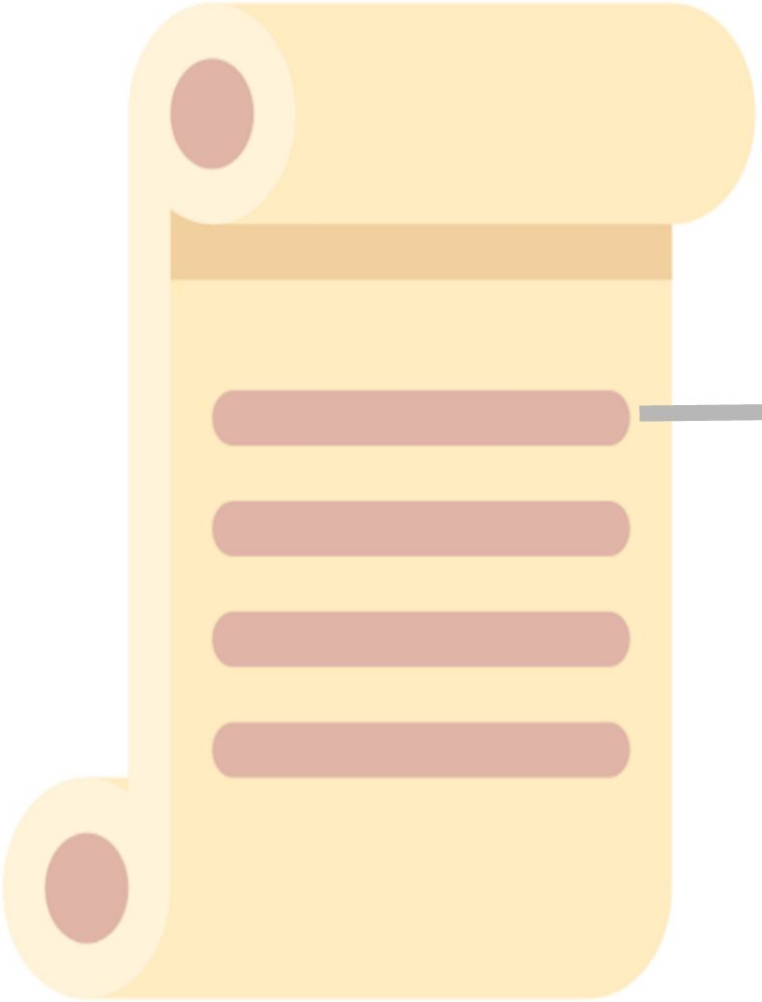
Oldest

Sequence Number

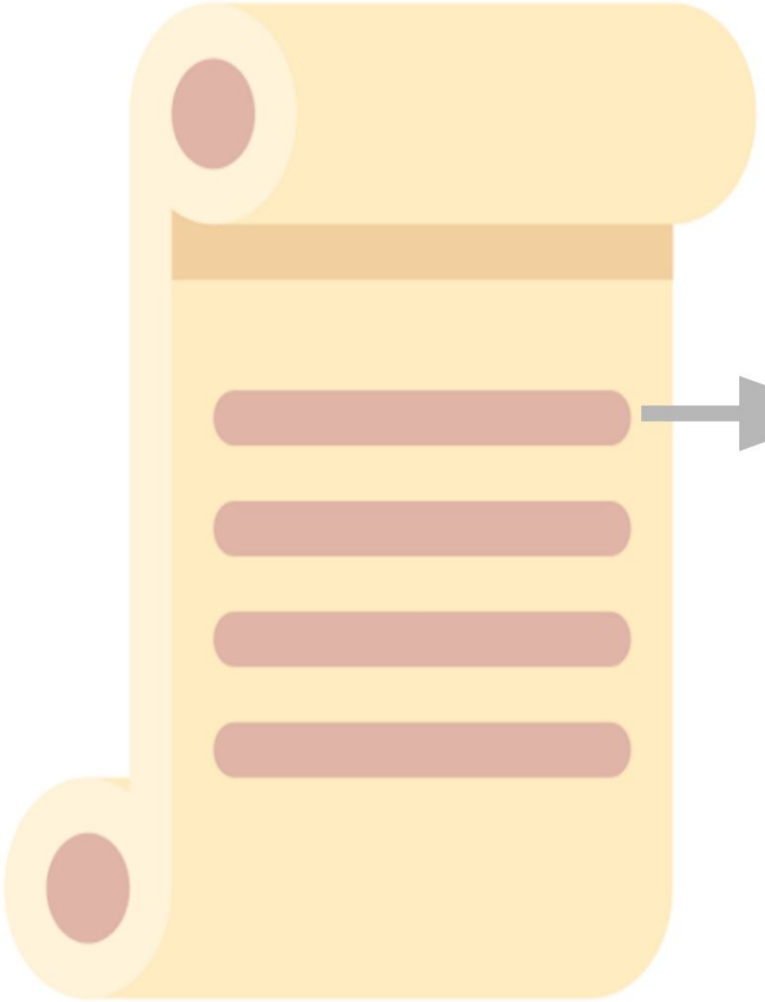
Newest



Record Data

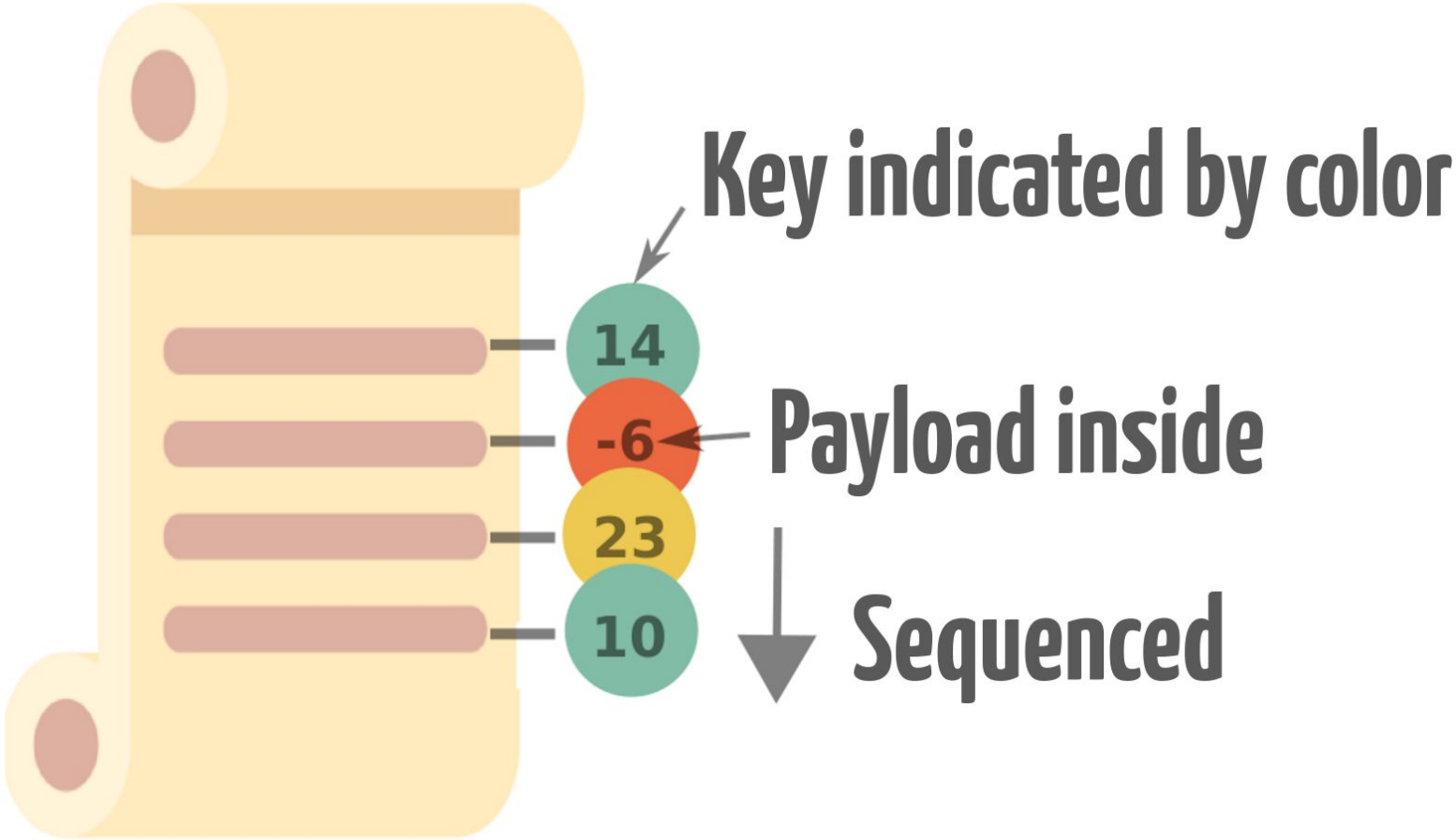


```
{key:   <k>
 data:  <d>}
```



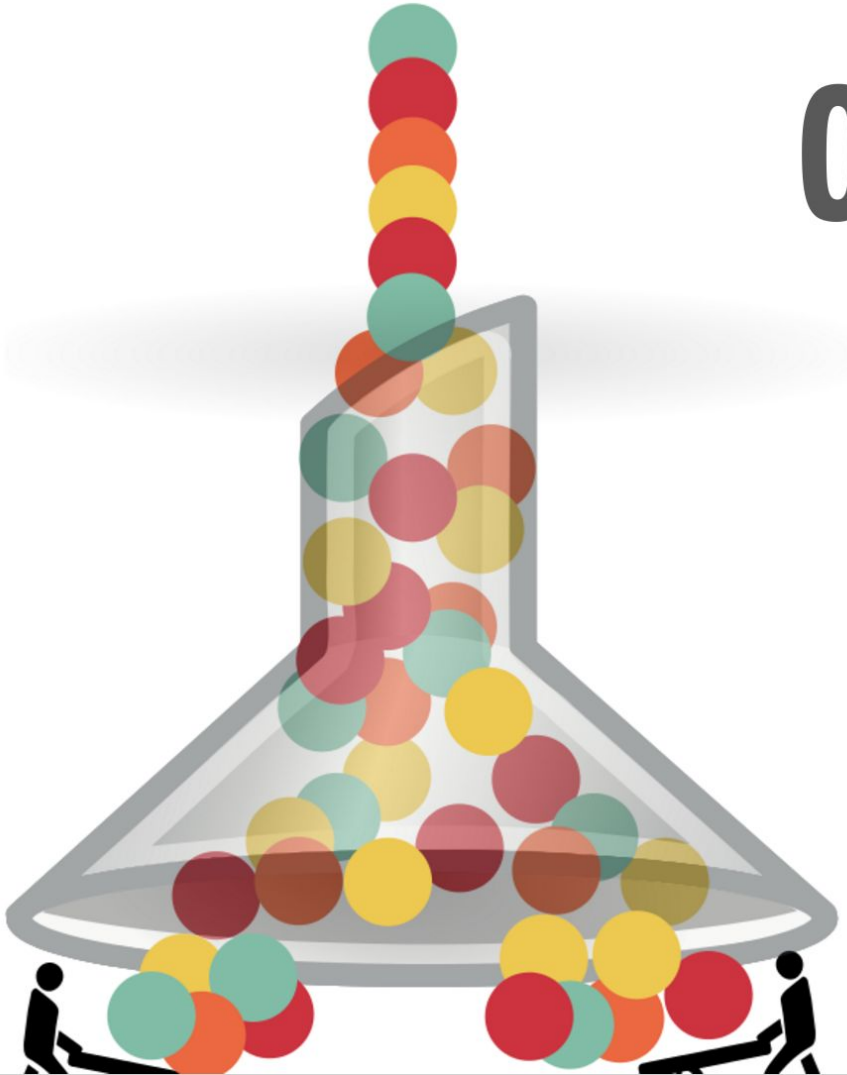
Record Metadata

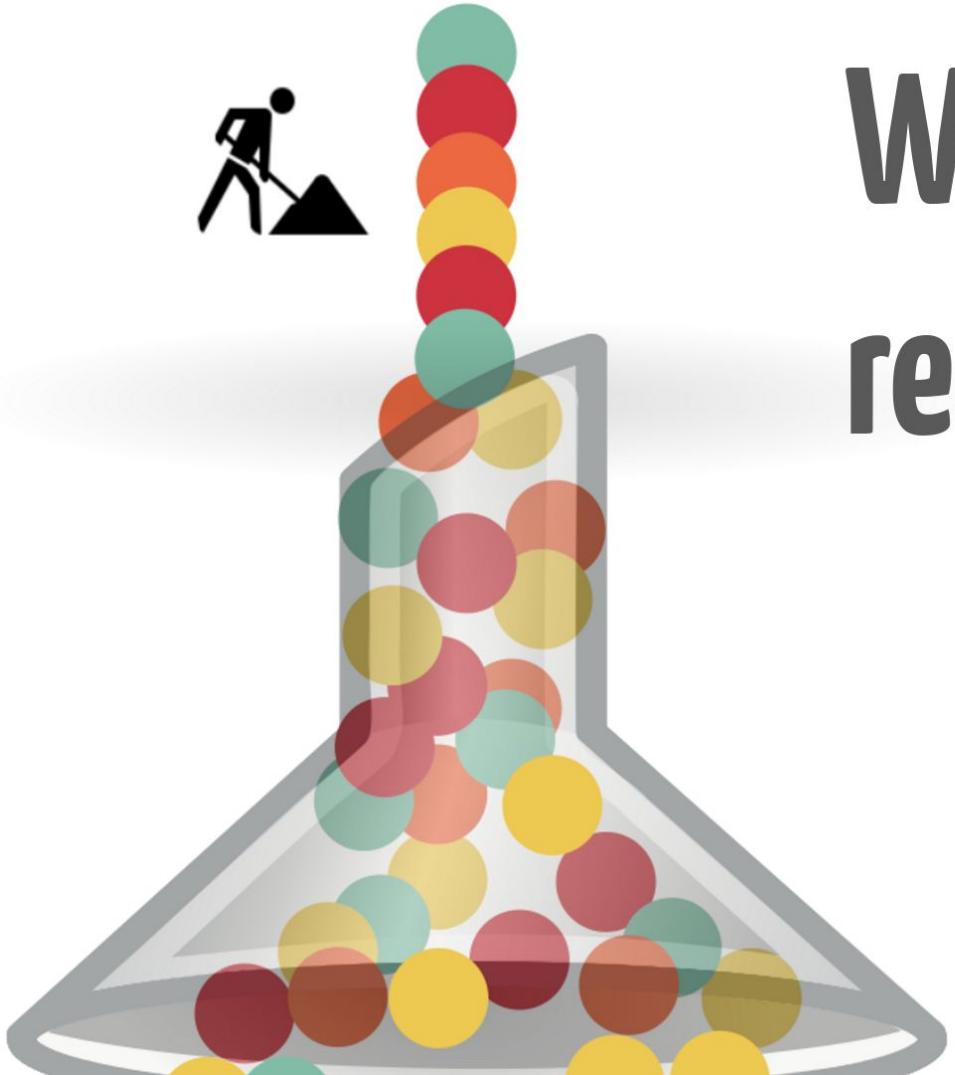
```
{key:   <k>
 data:  <d>
 sequence#:  <s>}
```


Key indicated by color
14
-6
23
10
Payload inside
Sequenced

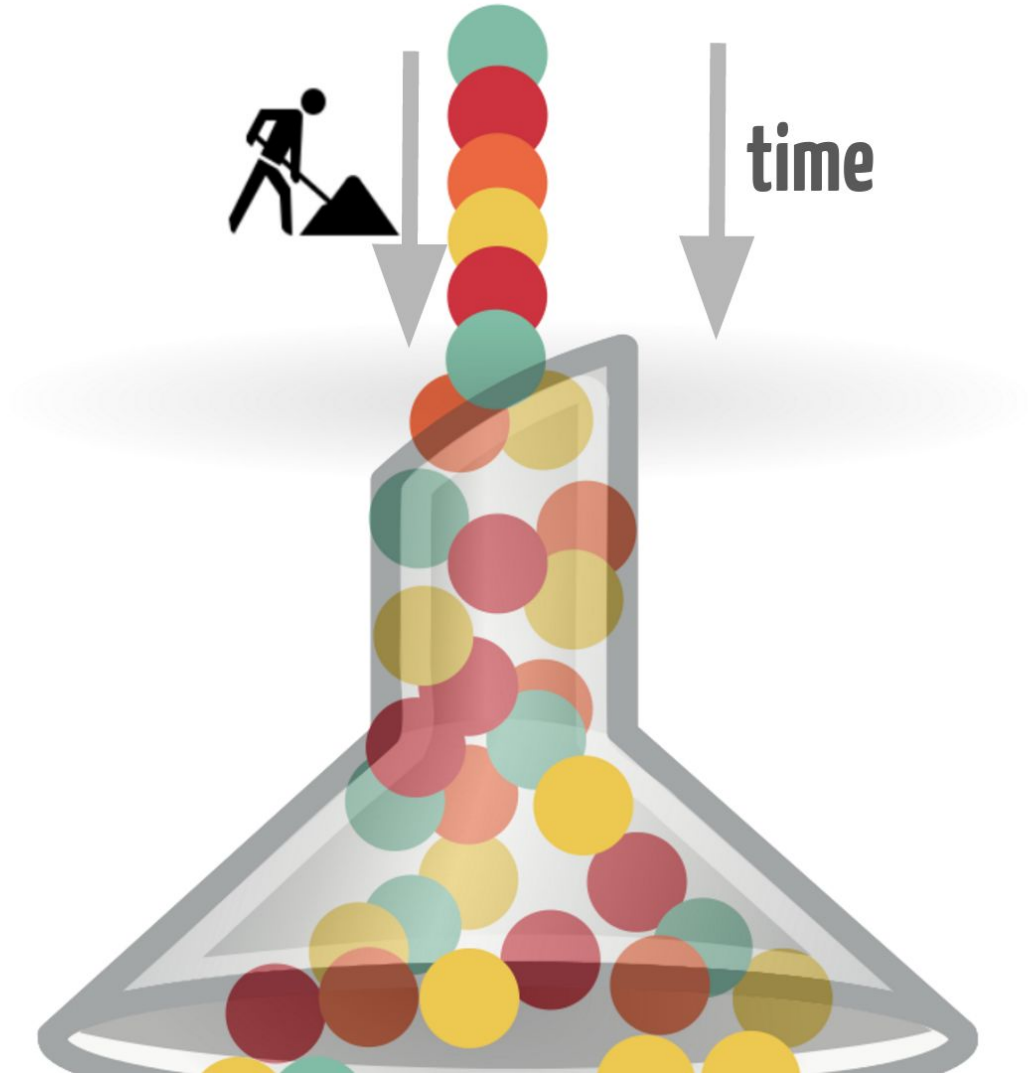
Kinesis Stream

Output serialized

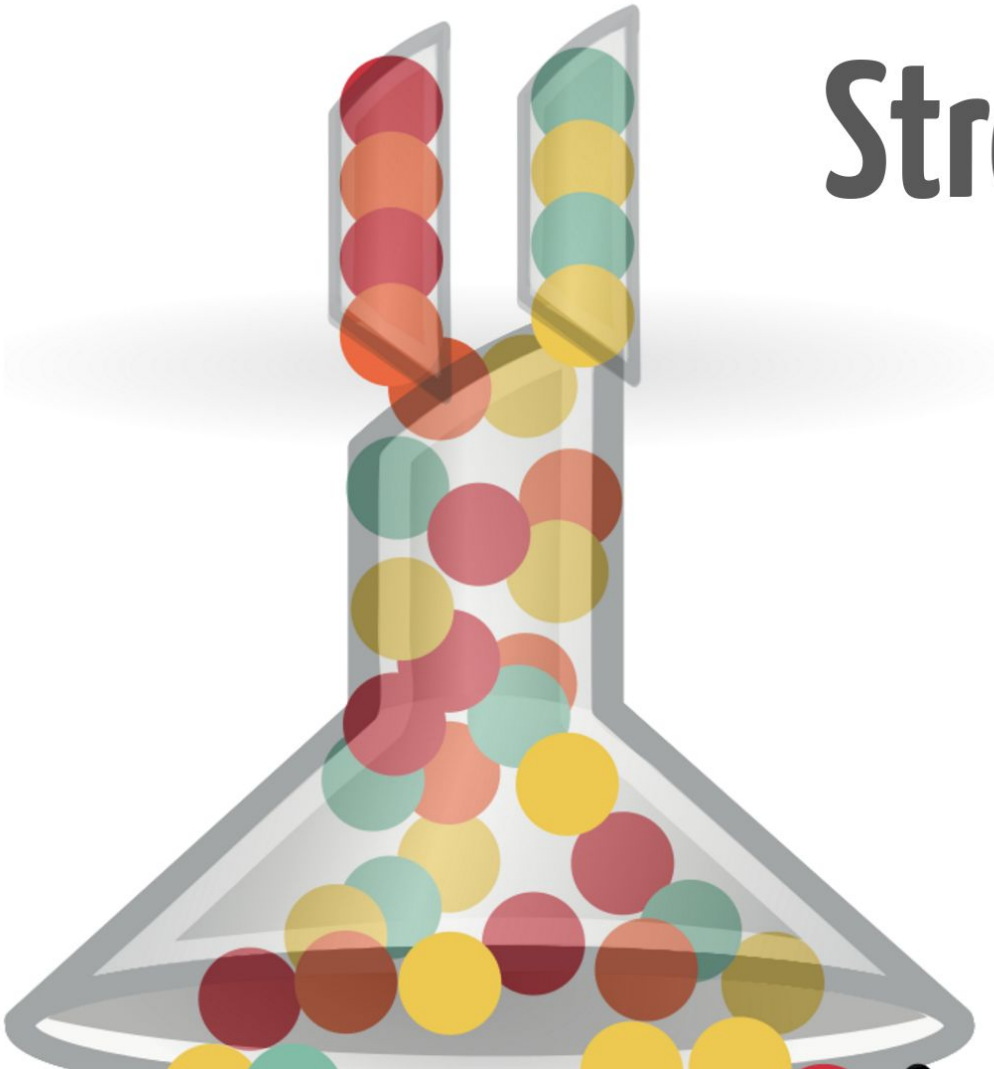




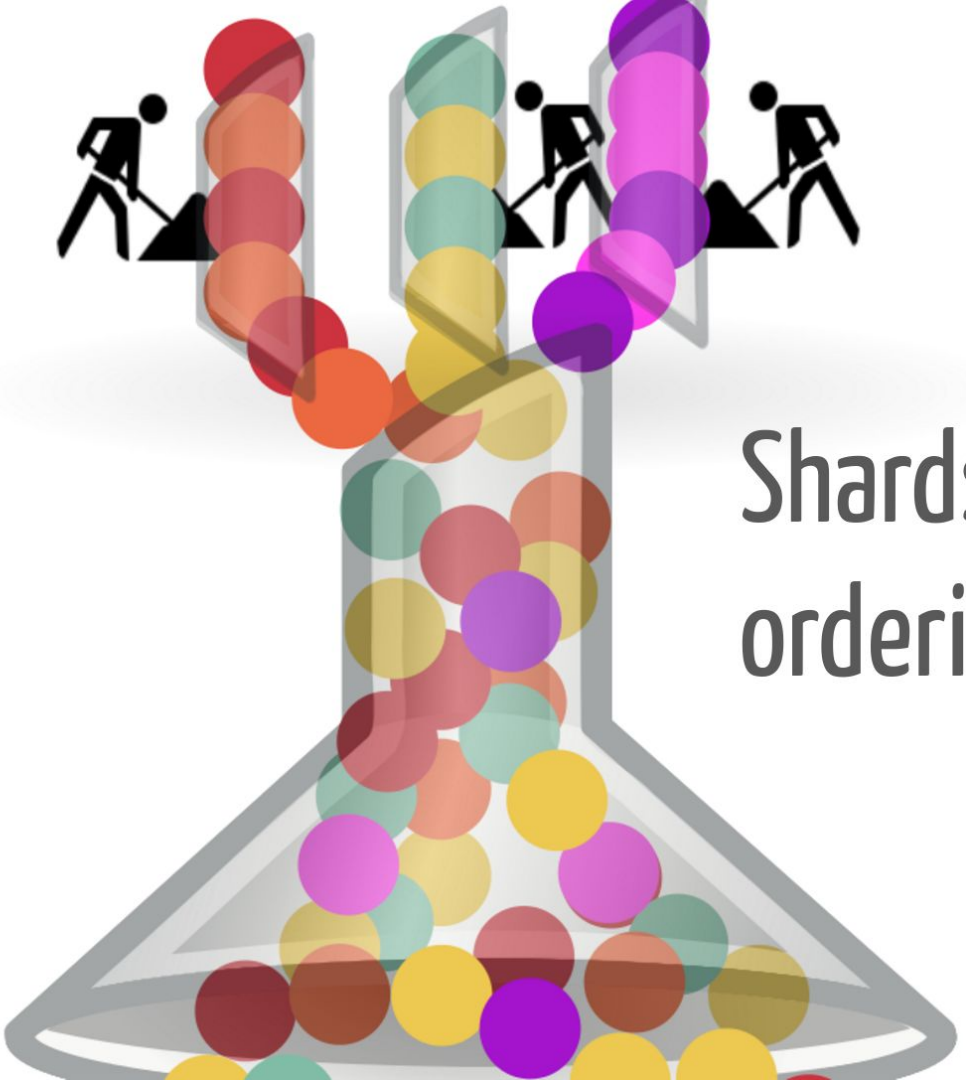
Workers read records in order


time

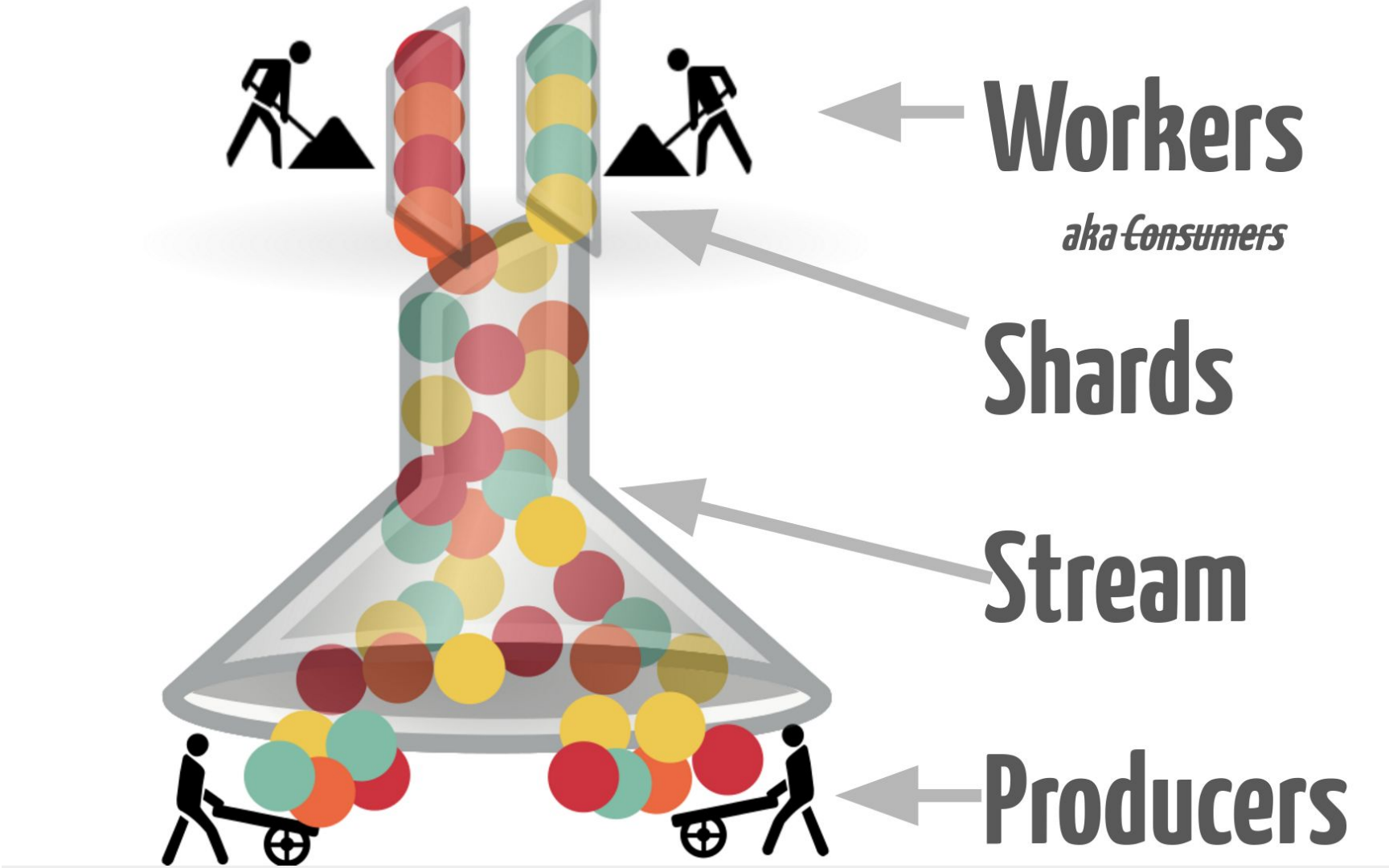
Stream is sharded



Shards preserve
ordering of records by key

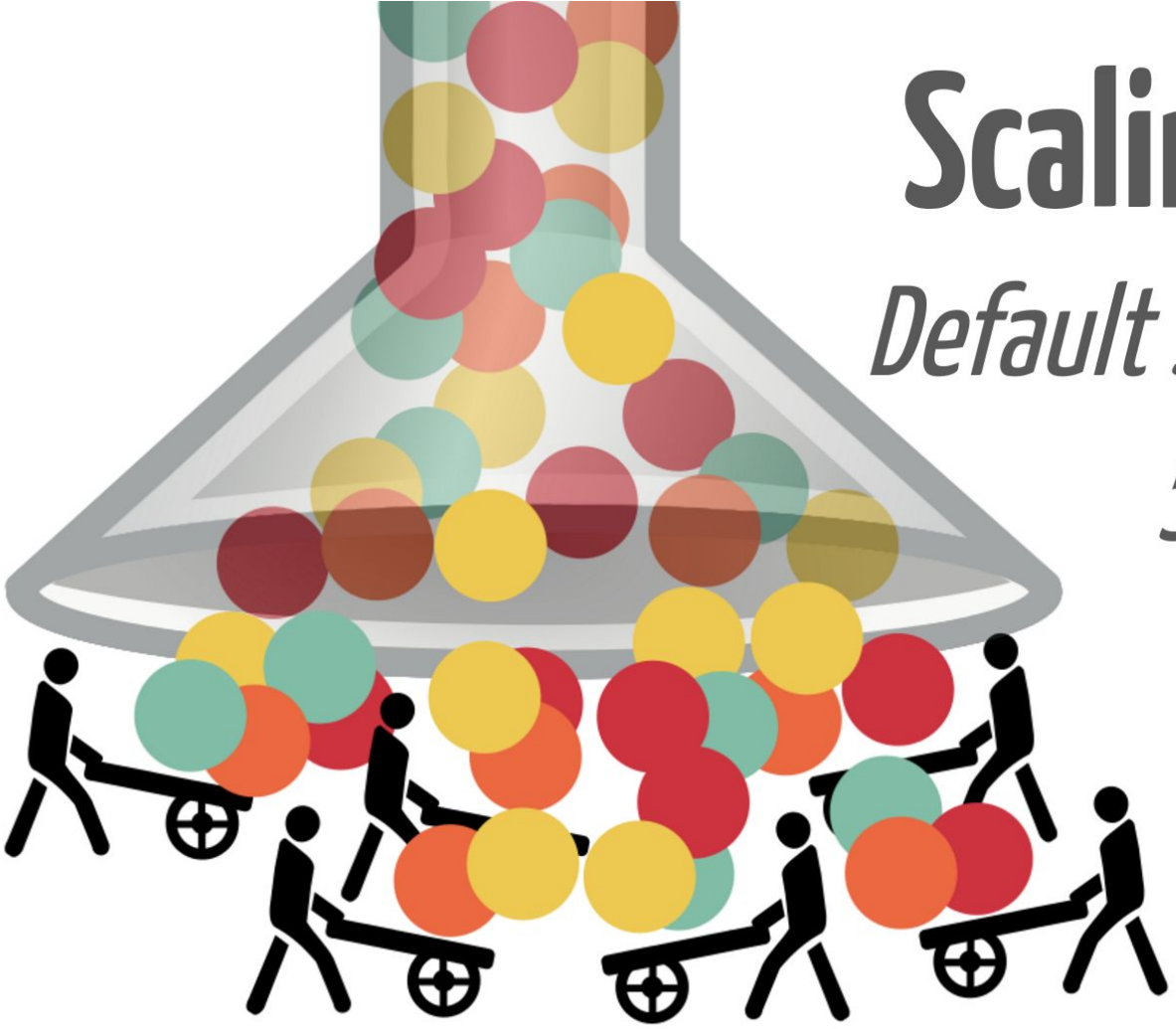


Shards preserve
ordering of records by key


Workers
aka Consumers
Shards
Stream
Producers

Scaling

Scaling Writes

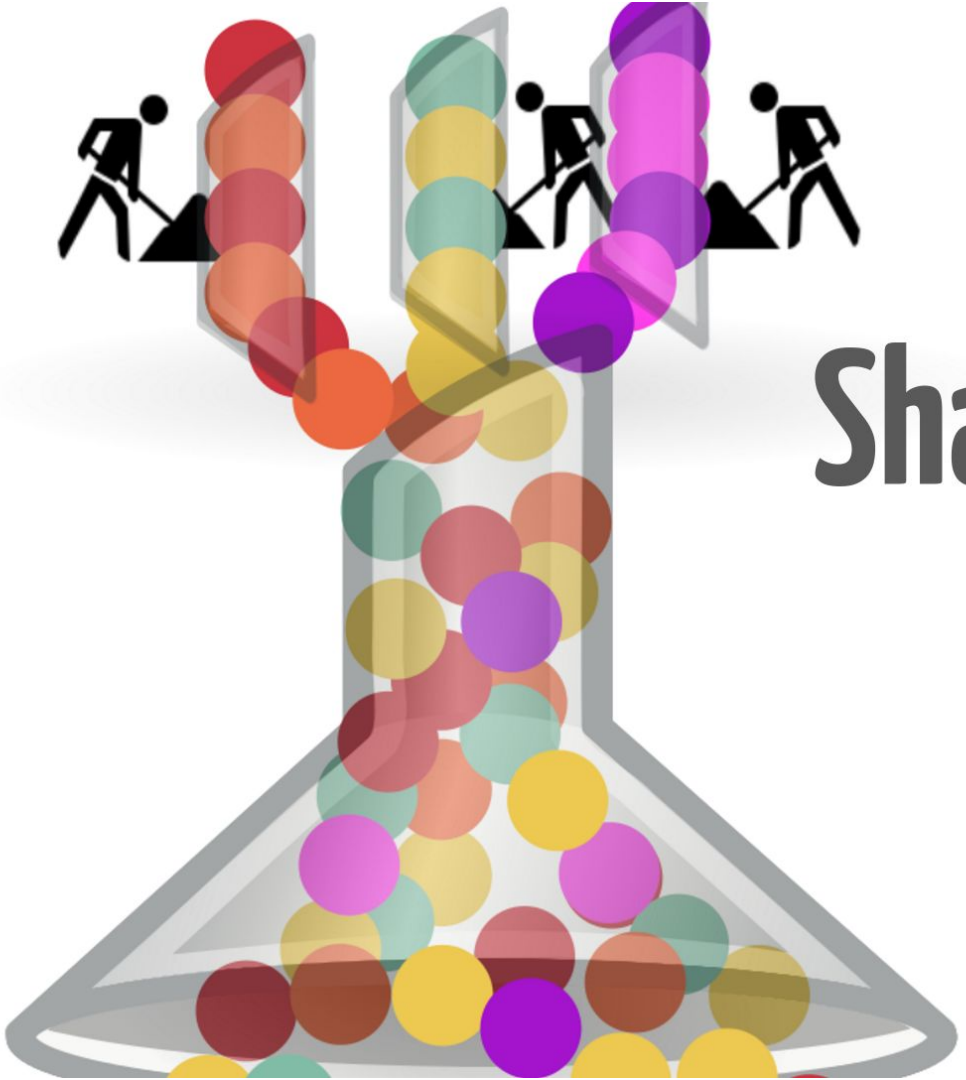


Scaling Writes

Default stream limits:

50k writes / s

50 MBps



Shard read limits:

2 MBps

5 reads / s

Fun & Games

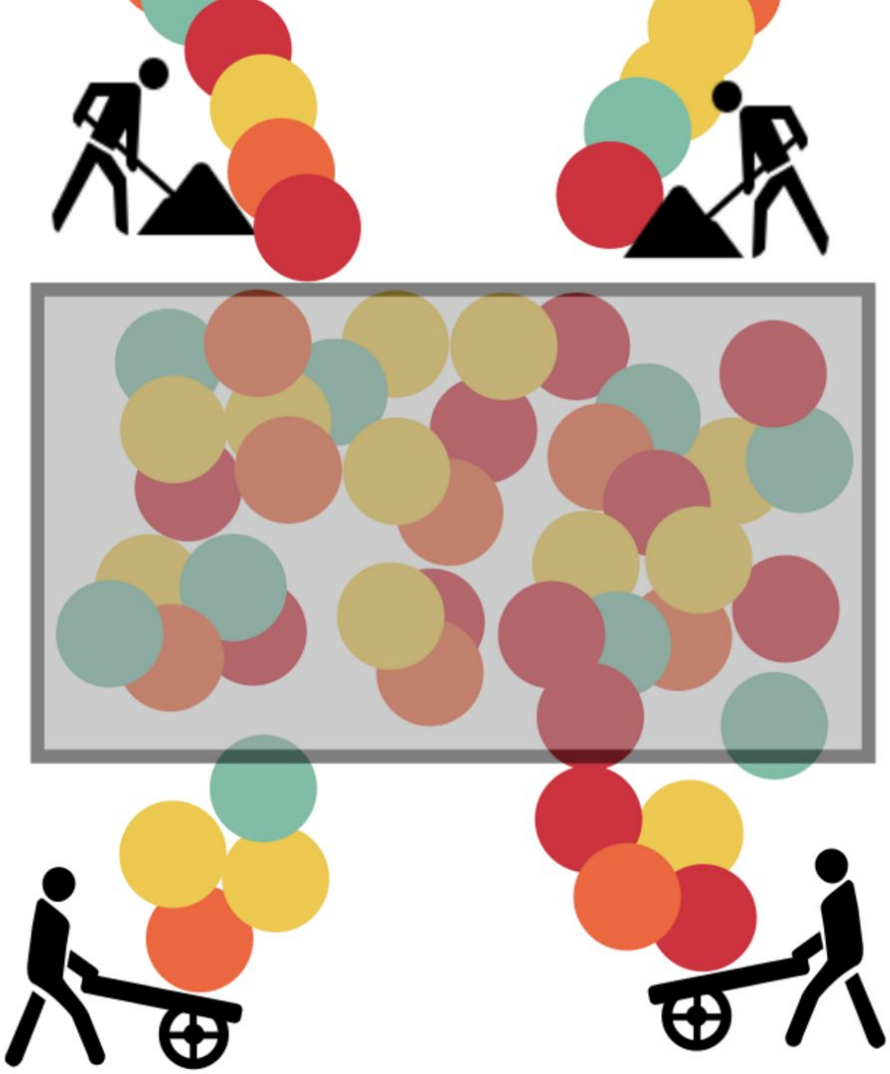
Metered

Pay per shard

```
          Shard $11.00/month
         Writes   1.50/month
Ext Storage  15.00/month
-------------------------
                $27.50/month
```

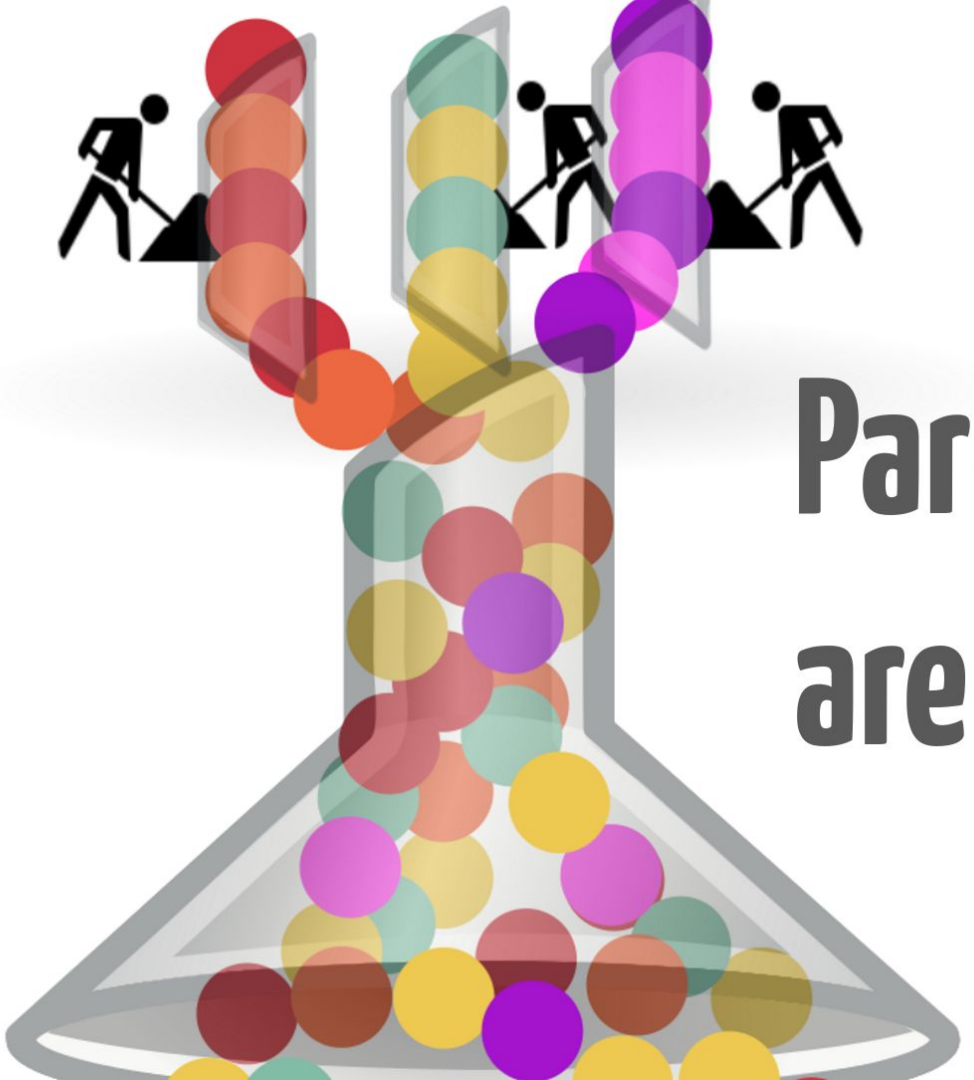
Distributed commit log

- Log is sharded
  - Data read by shard

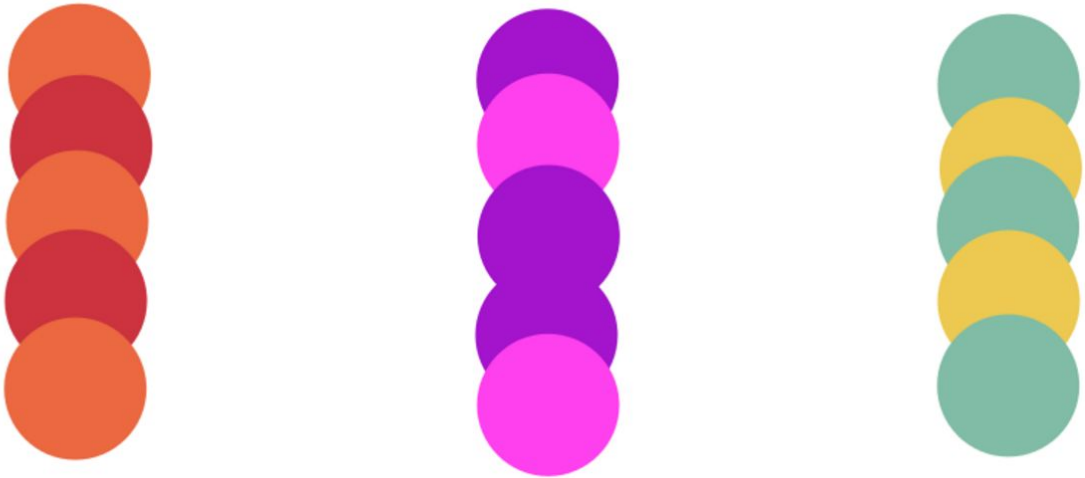


Queue:

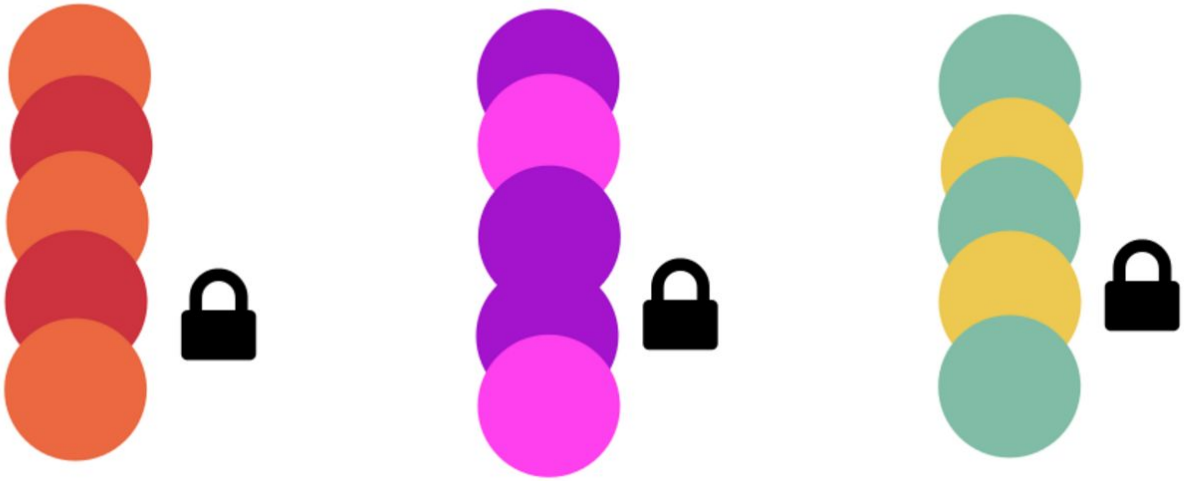
Parallel readers see all keys



Parallel readers are coordinated

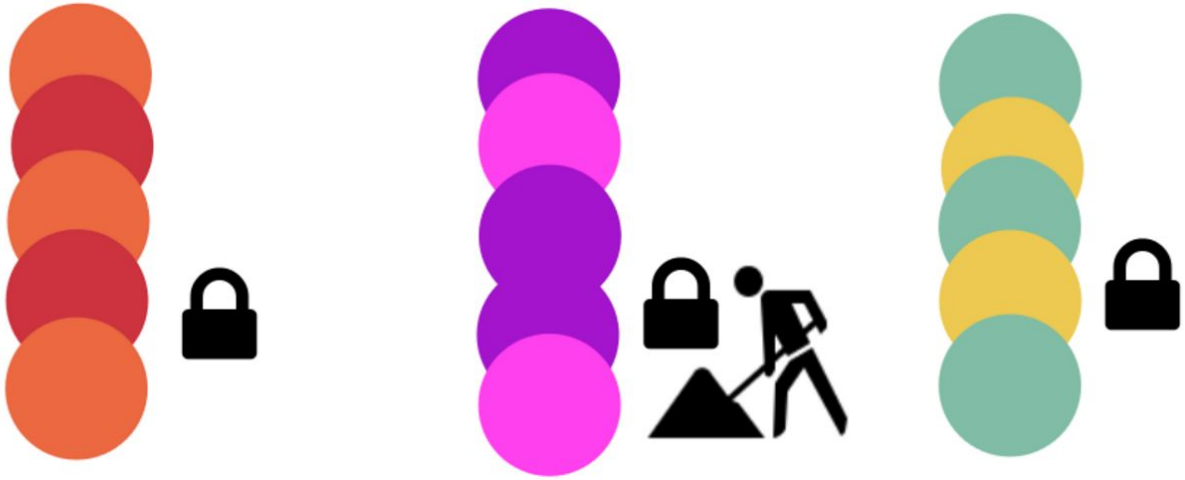


Shards

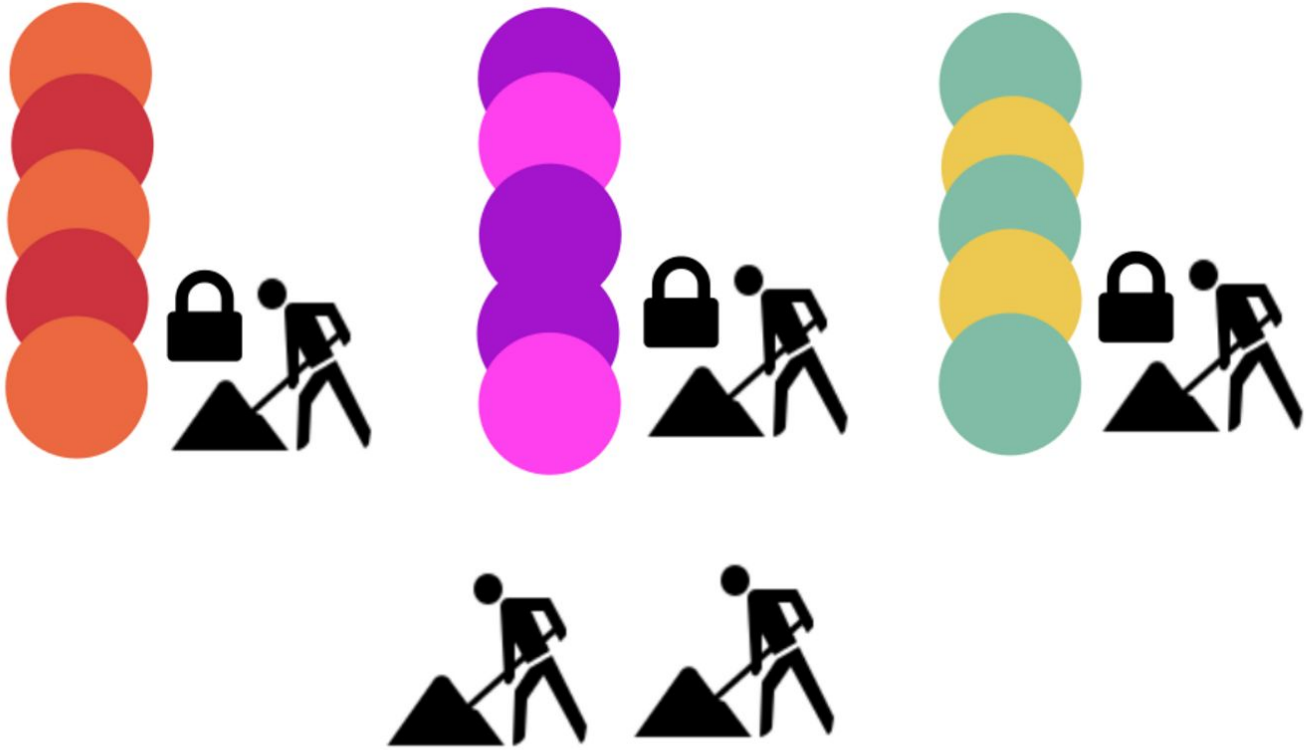


Lease for each shard

Workers obtain leases for shards

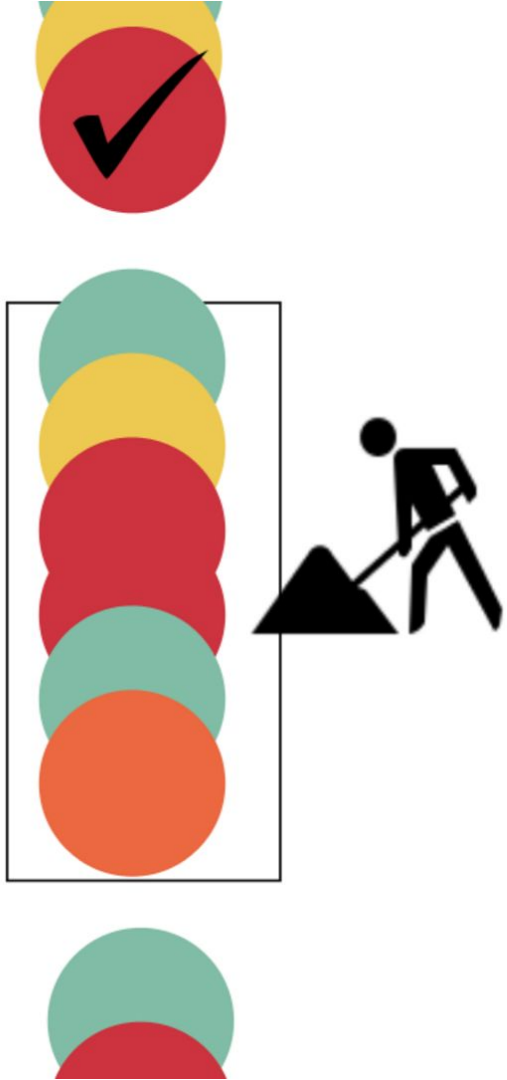


Single worker can hold many leases



"Extra" workers wait for work

Batched Reads

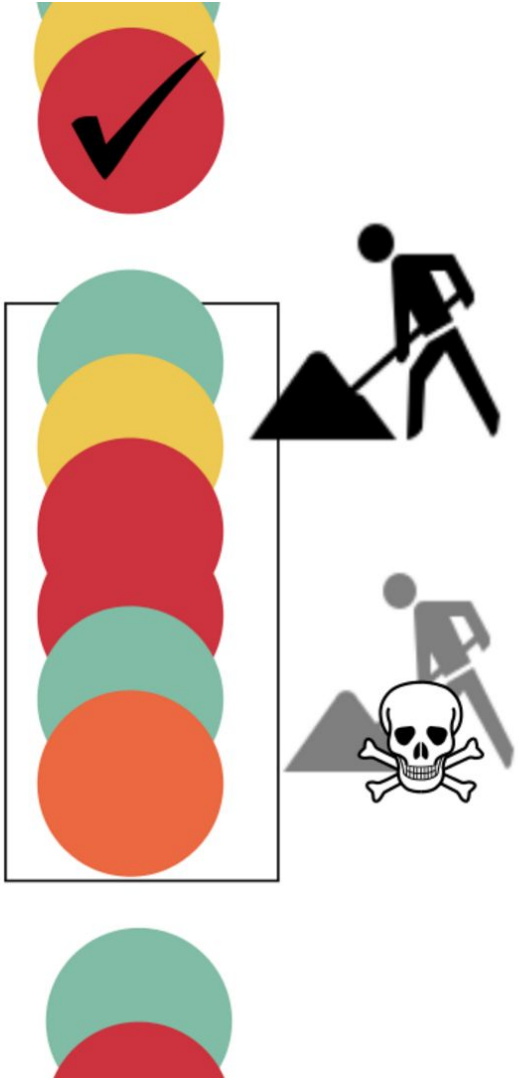


Data read in batches

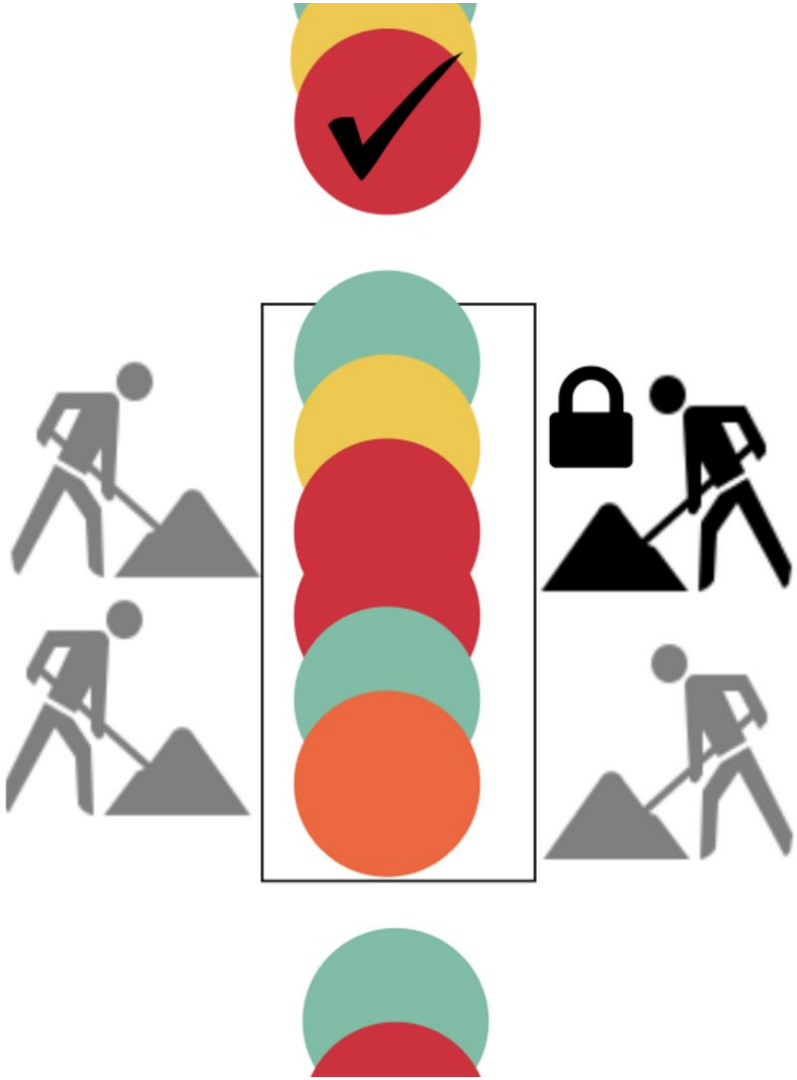
Checkpoints track last record processed

Worker advances checkpoint

"At least once"



Previous worker died before advancing checkpoint



Zombie workers may be processing records

Idempotent record processing

Problem:

Since data records are delivered "at least once" the application must deal with record replay.

Idempotent record processing

Problem:

Since data records are delivered "at least once" the application must deal with record replay.

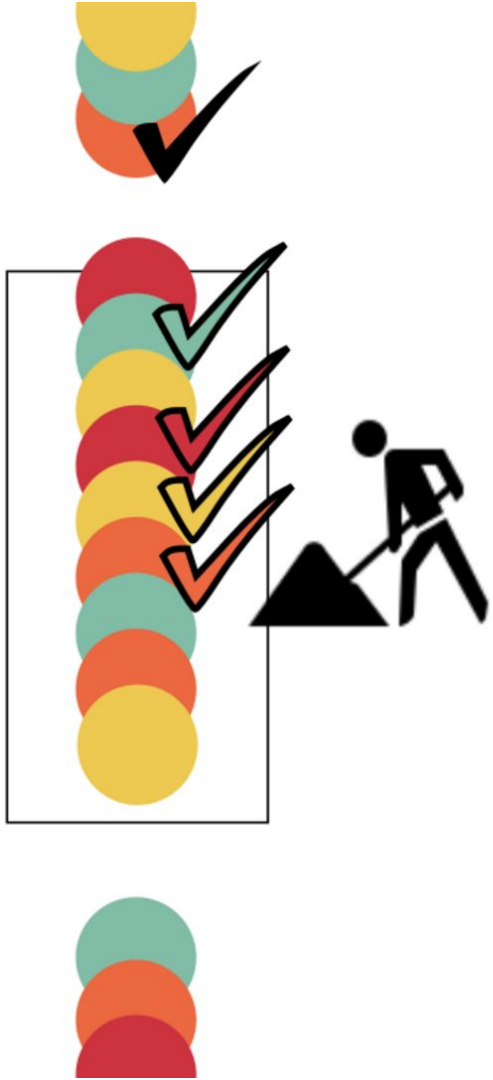
Solution:

Make record processing idempotent.

Idempotent processing

Consequences:

- Record replay does not invalidate results.
- Multiple workers simultaneously processing the same records does no harm.
- Additional, careful design is required to make processing idempotent.



Track last sequence number processed for each key

Track checkpoints by partition key

Problem:

As records are replayed, even if their processing is idempotent, it would be preferable to not replay the processing.

Track checkpoints by partition key

Problem:

As records are replayed, even if their processing is idempotent, it would be preferable to not replay the processing.

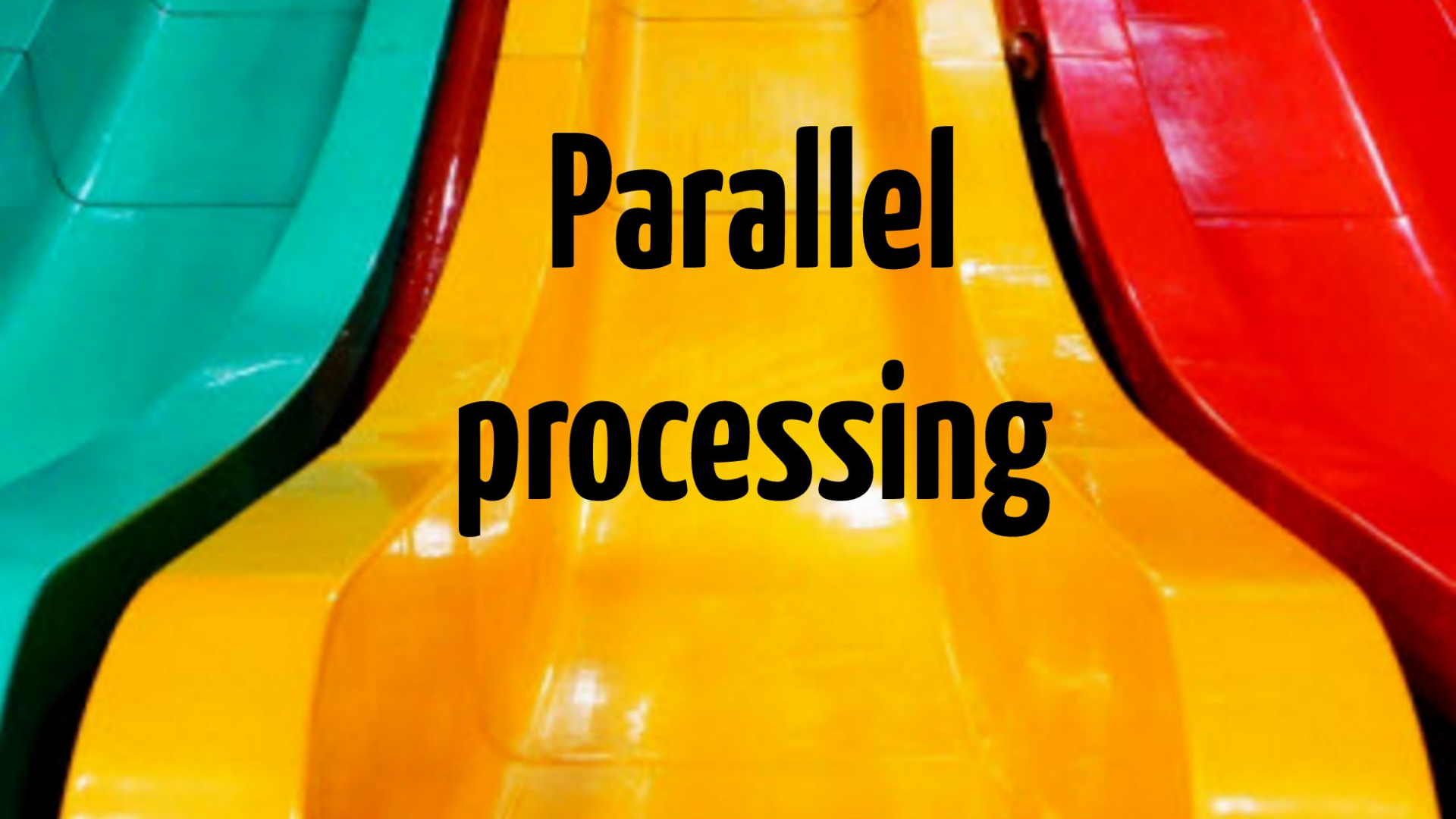
Solution:

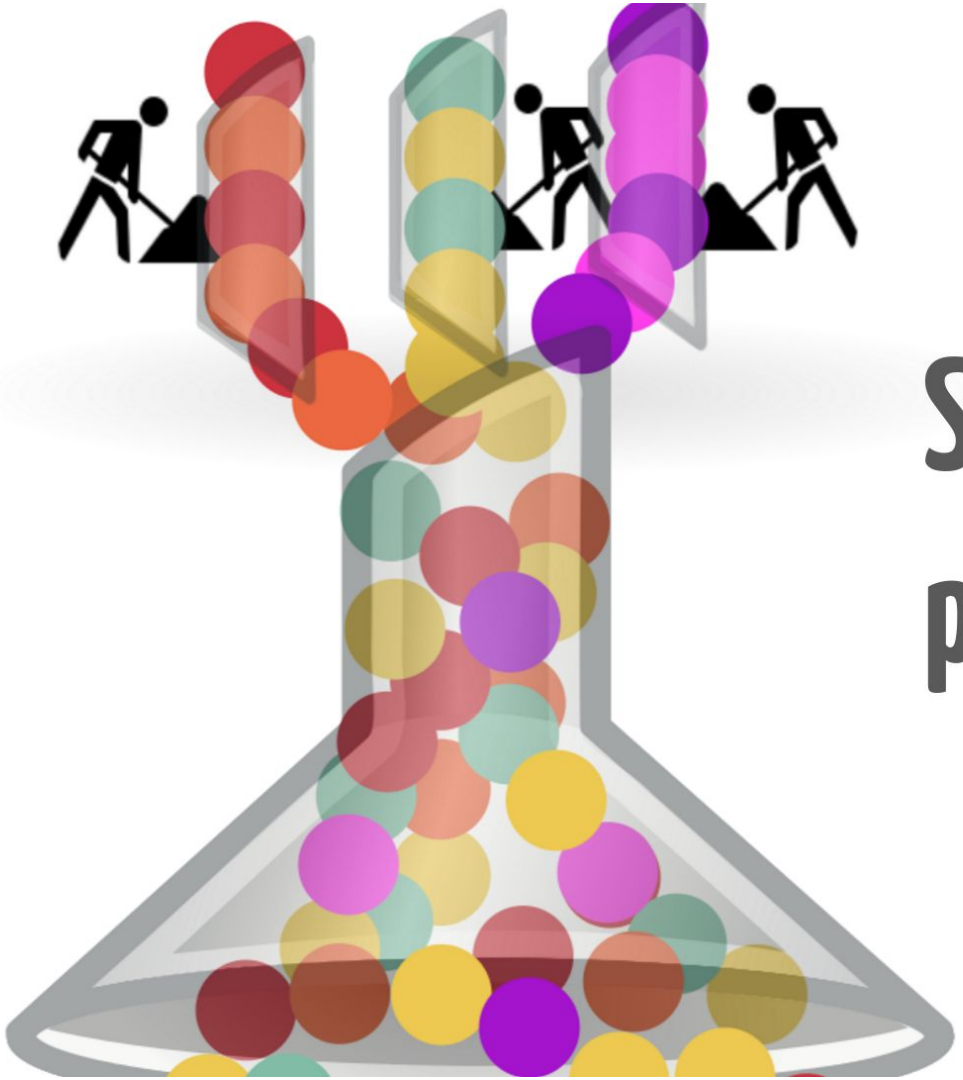
Track the last sequence number processed for each partition key.

Track checkpoints by partition key

Consequences:

- Replayed records can be ignored.
- Requires additional state to be stored for each key.

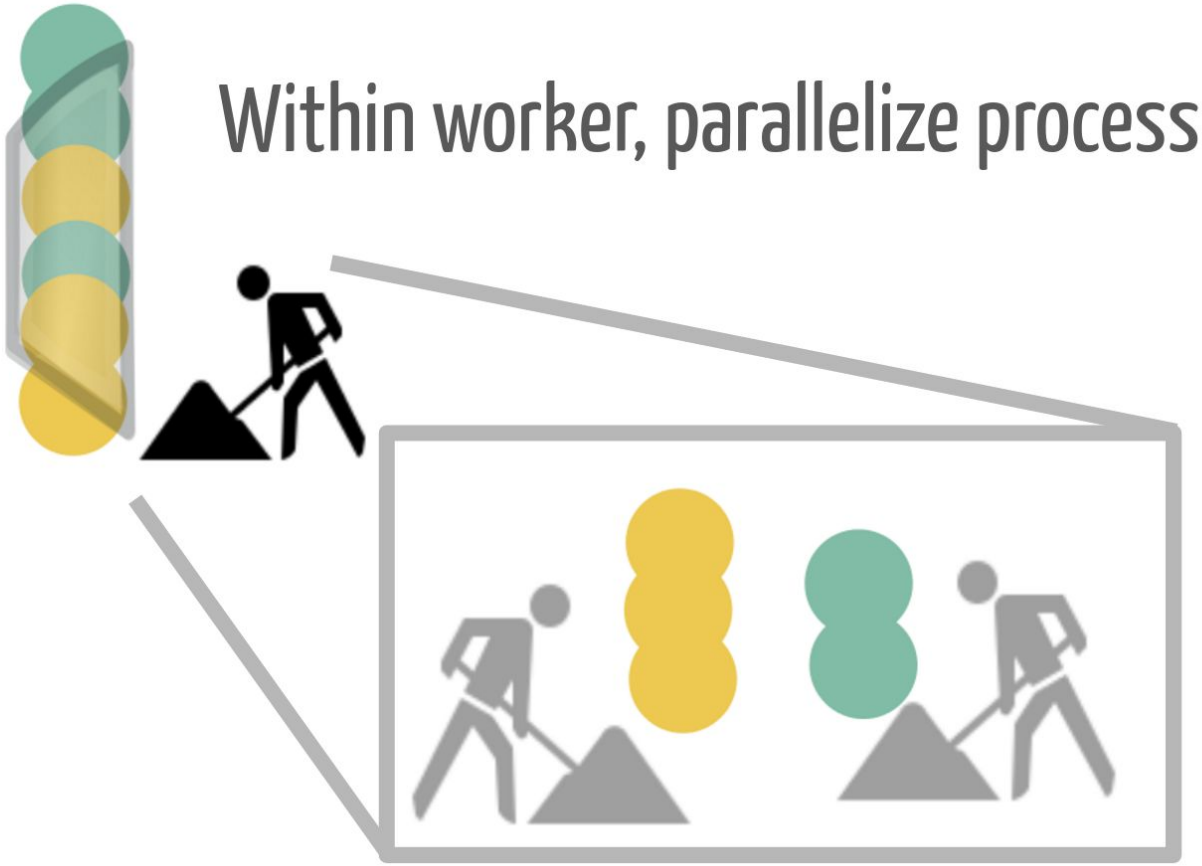

Parallel
processing



Shards enable parallel processing

How to parallelize a shard?

Within worker, parallelize processing by key



Sub-sequence processing

Problem:

Serialized record processing in a worker can be a bottleneck.

Sub-sequence processing

Problem:

Serialized record processing in a worker can be a bottleneck.

Solution:

Within the worker, break the records up by key and process these sub-sequence in parallel in multiple threads.

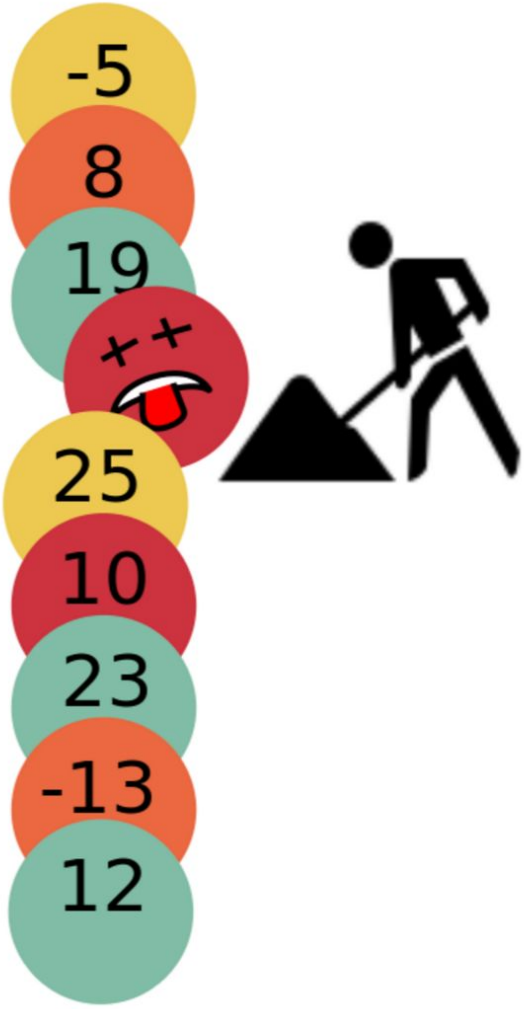
Sub-sequence processing

Consequences:

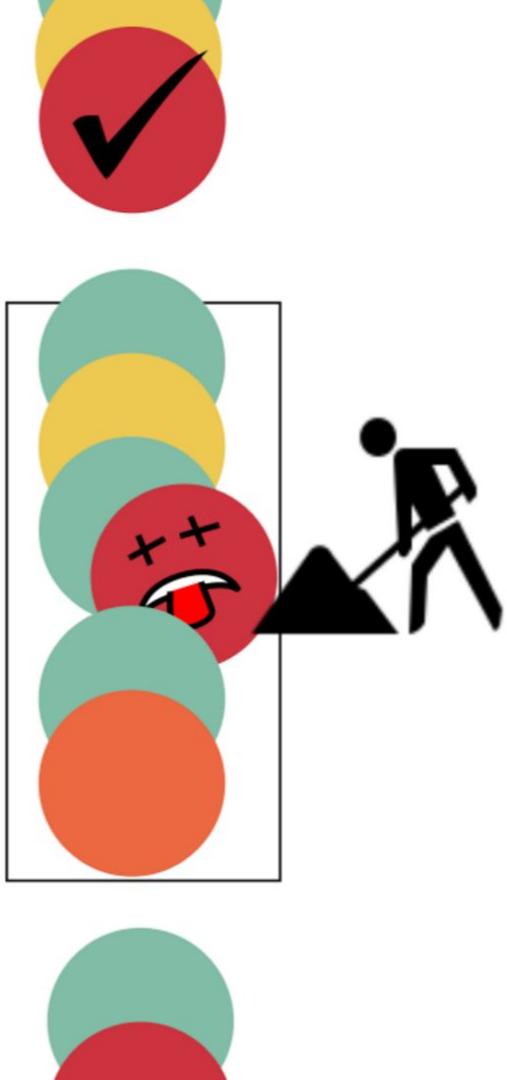
- Parallelizing the work allows the records to be processed in less time, assuming there is available compute capability on the nodes.
- This may in turn motivate scaling the worker nodes up. So it leads to a scale-up vs scale-out tradeoff.
- This approach introduces complexity into the worker logic with respect to checkpointing.
- This approach is not expected to affect the overall "in order" guarantees of the stream because those guarantees can be understood to apply to individual keys.

Errors

In some domains readers can discard records on error



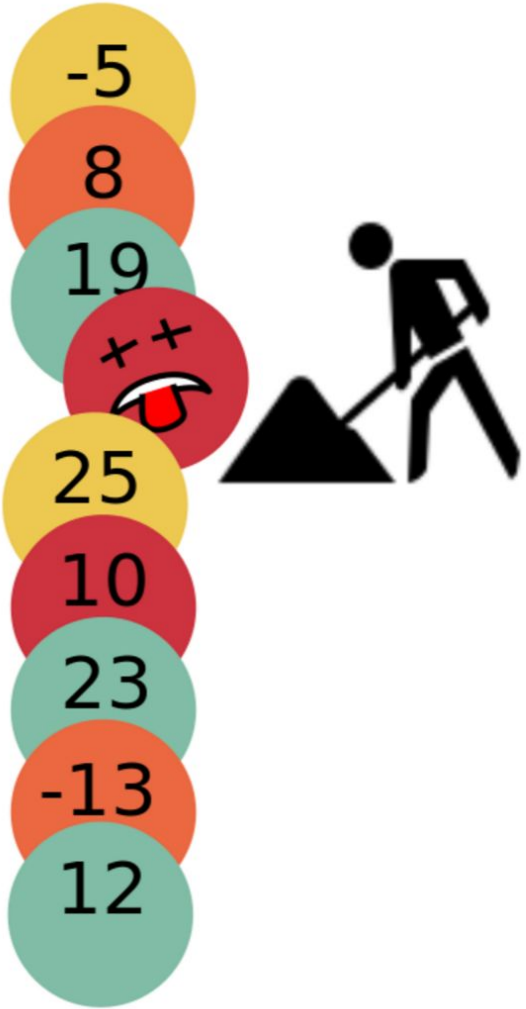
In other domains every record matters



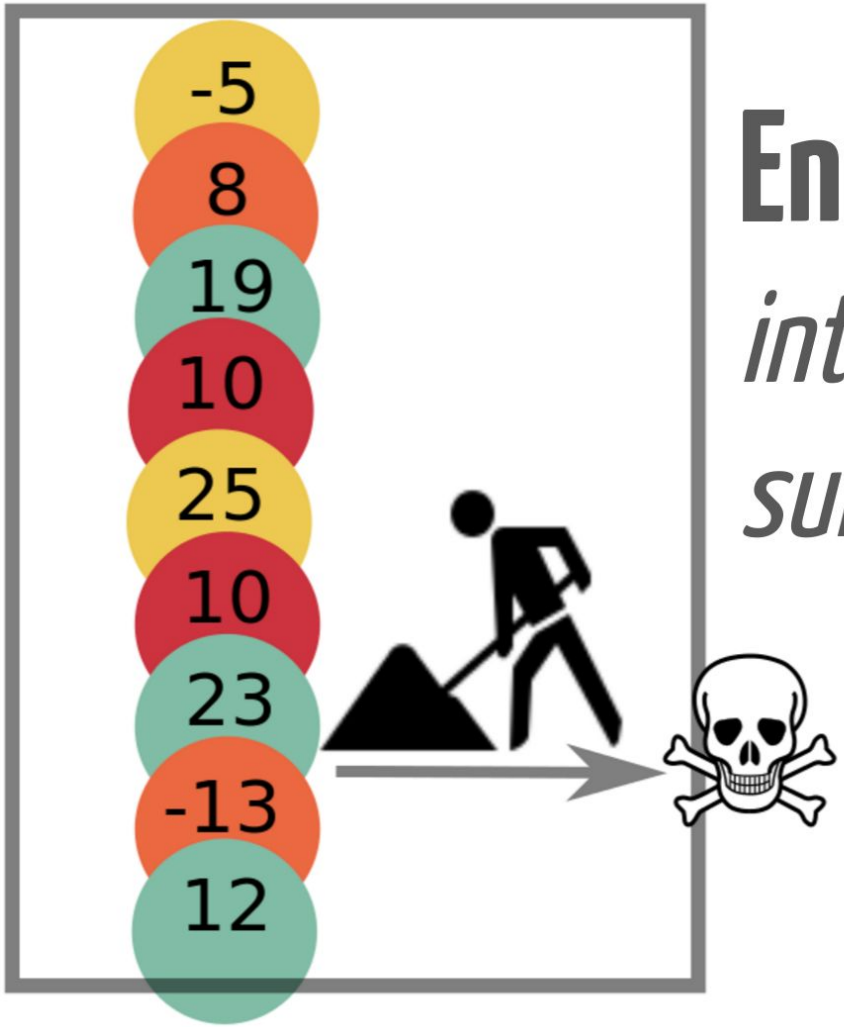
What to do with batch?

How to advance checkpoint?

How to keep up with firehose?



Intrinsic error



Environmental error

interacting with the surrounding systems

Classify exceptions

Problem:

Exceptions occur during batch processing and it is unclear how to handle them.

Classify exceptions

Problem:

Exceptions occur during batch processing and it is unclear how to handle them.

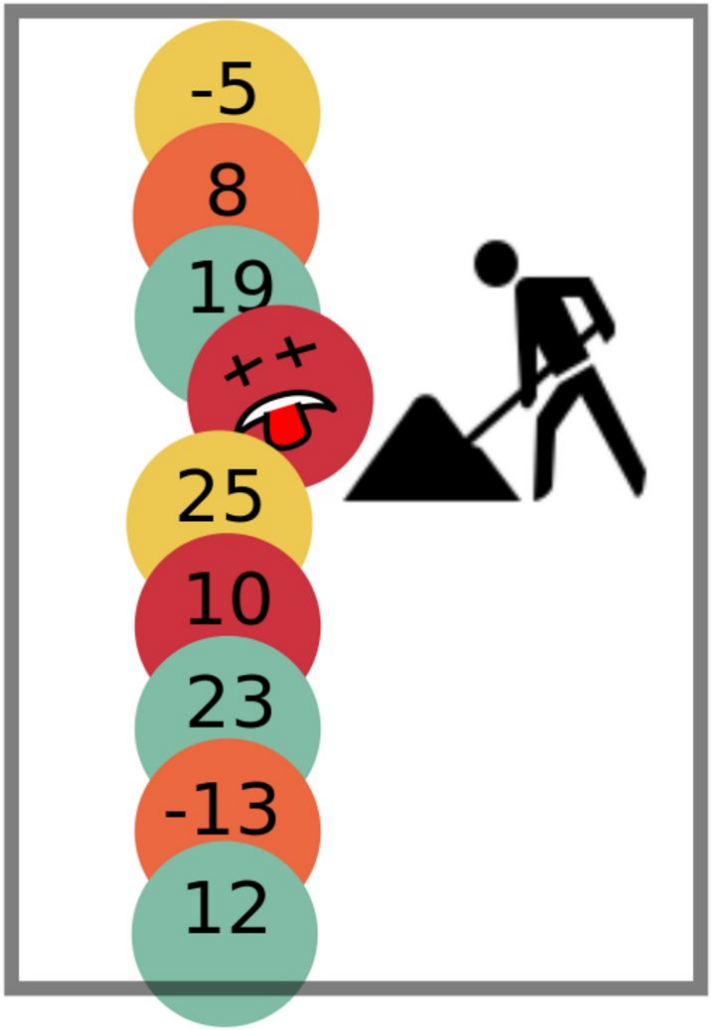
Solution:

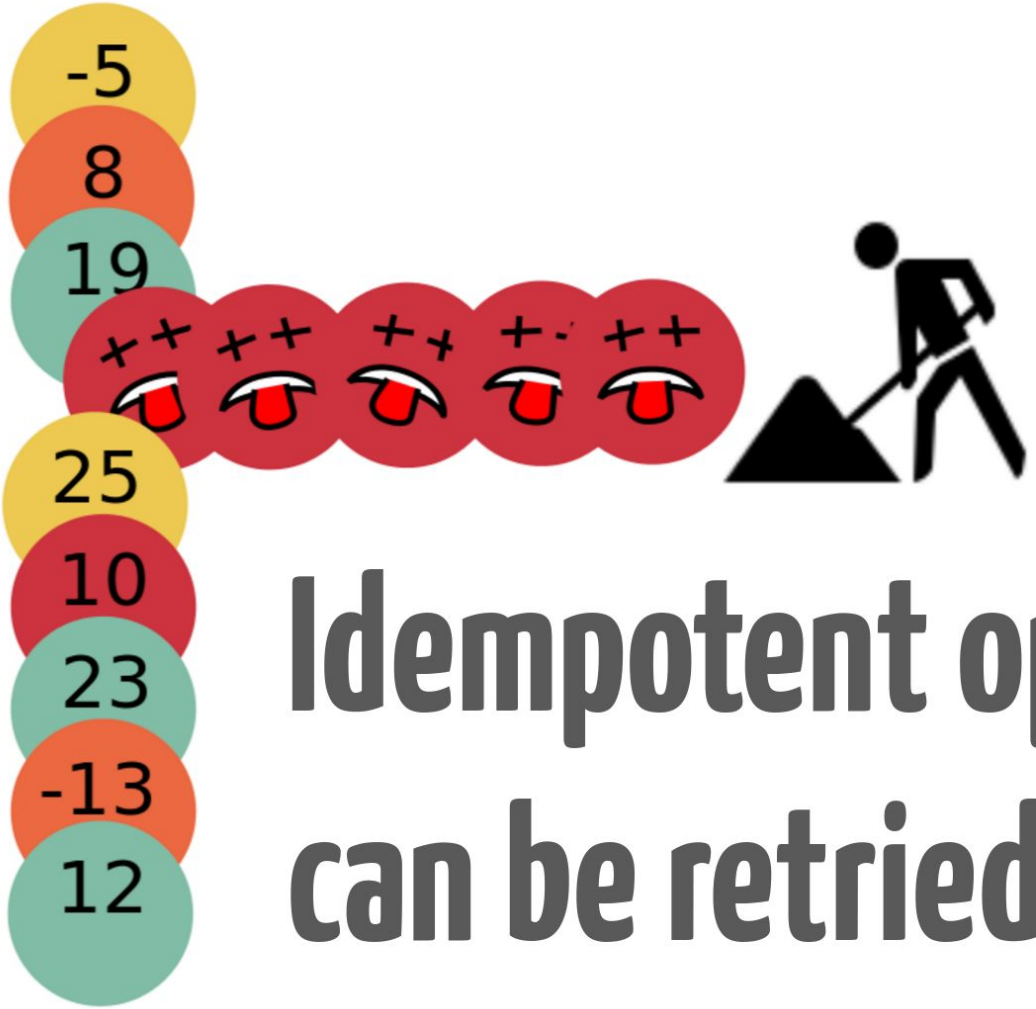
Classify exceptions as indicating an external problem with the environment or an internal problem.

Classify exceptions

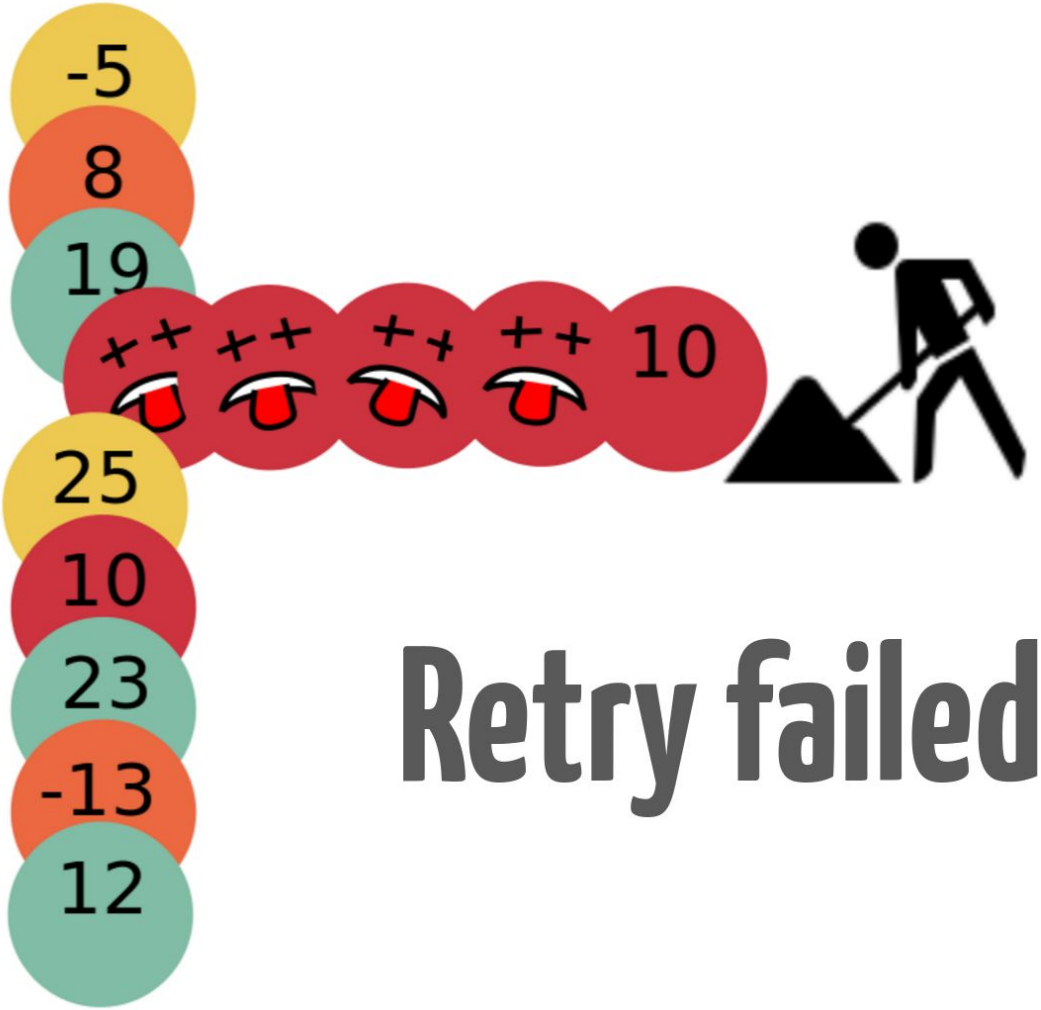
Consequences:

- Enables triggering different error handling procedures based on whether environmental errors are occurring.
- Requires enumerating and classifying various exceptions.
- Some exceptions are ambiguous and defy classification.
- Creates a potentially fragile dependency on details of exceptions.


-5
8
19
++
25
10
23
-13
12



Idempotent operations can be retried



Retry failed records

Retry failed records

Problem:

The cost of failing to process records is high and the cost of retrying them is low.

Retry failed records

Problem:

The cost of failing to process records is high and the cost of retrying them is low.

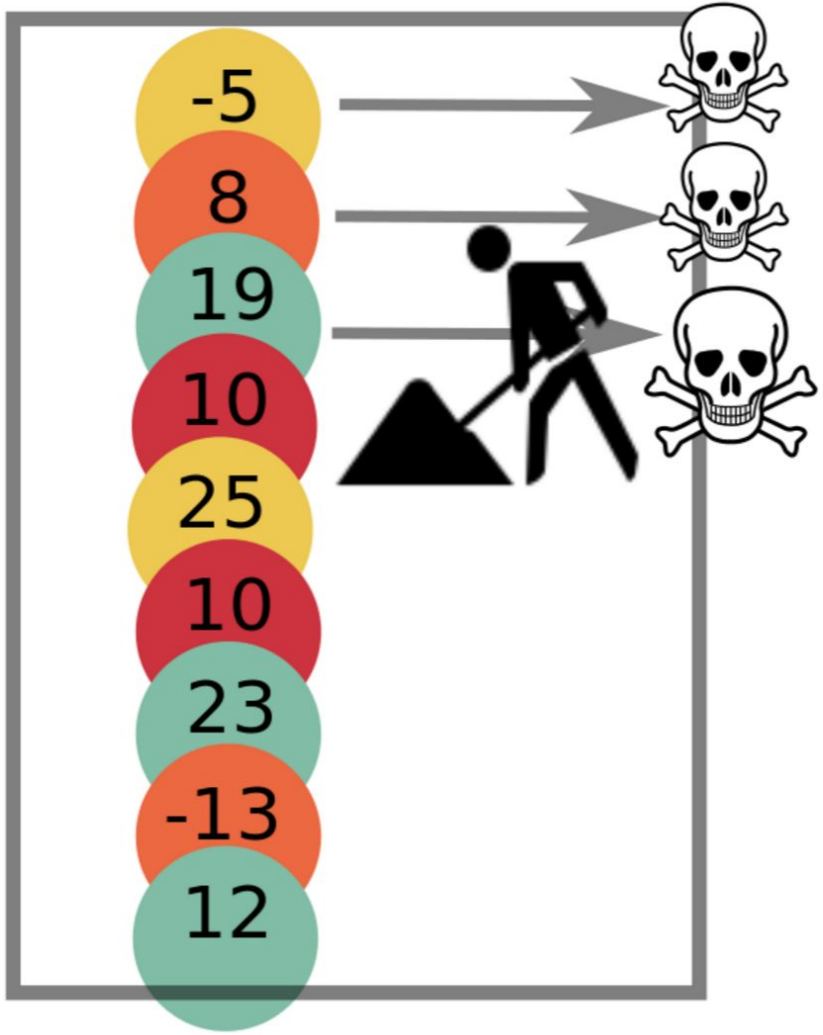
Solution:

Retry failed records <n> times or until they succeed.

Retry failed records

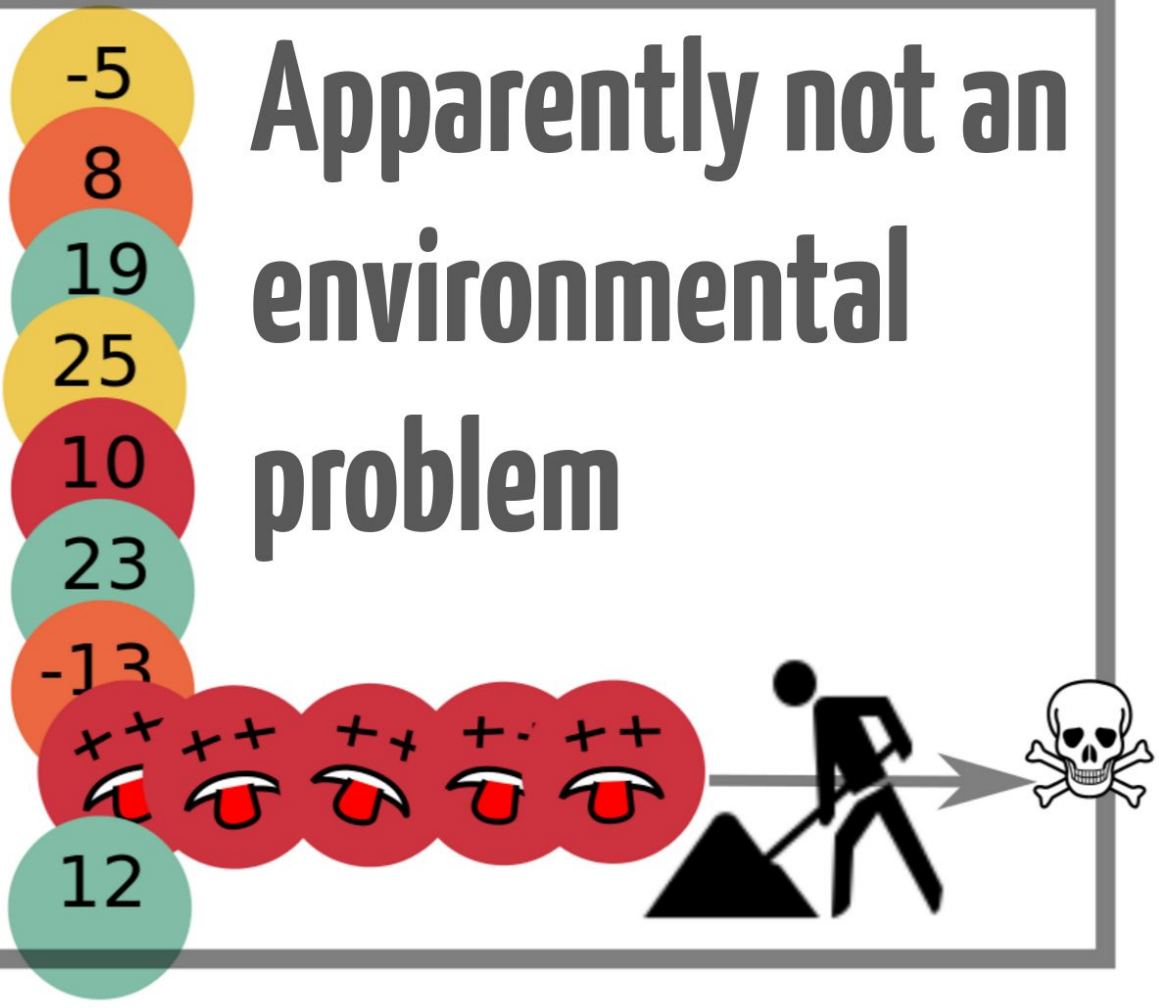
Consequences:

- Some transient failures will be dealt with.
- Assumes the record processing is idempotent.
- This can mask underlying problems.
- Increases the cost of processing a record.
- If the attempted processing of the record leads to problems then this can exacerbate the problem.



Clearly an environmental problem

Apparently not an environmental problem



Count exceptions

Problem:

Different exception handling procedures are warranted based on how widespread failures are.

Count exceptions

Problem:

Different exception handling procedures are warranted based on how widespread failures are.

Solution:

Compute the count and percentages of various exception classes and apply heuristics to determine how to behave.

Count exceptions

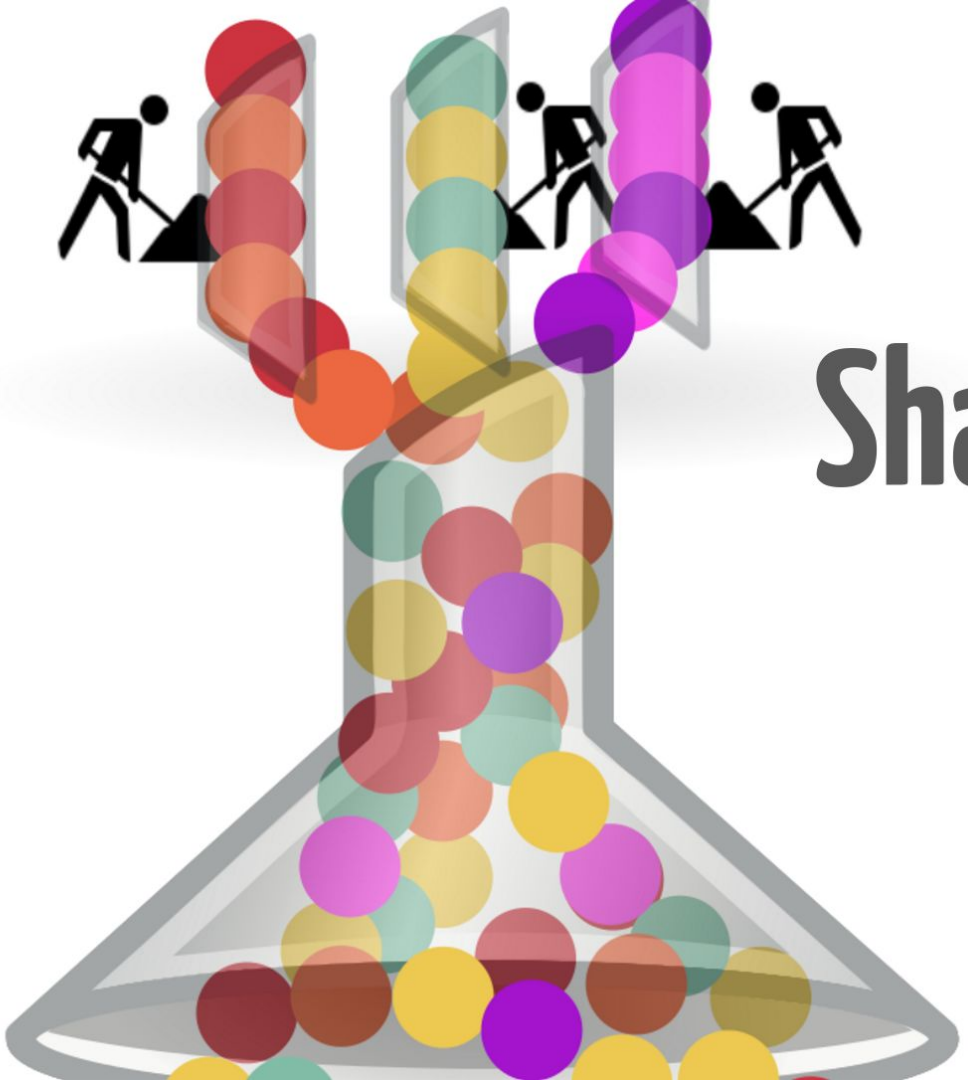
Consequences:

- Allows error handling code to distinguish between widespread system outages vs. failures related to specific records.
- Requires identifying and implementing appropriate heuristics.
- Heuristics may lead to complex and unexpected behaviour.

Multiple Apps

Workers are for a particular "application"

Multiple applications can run in parallel



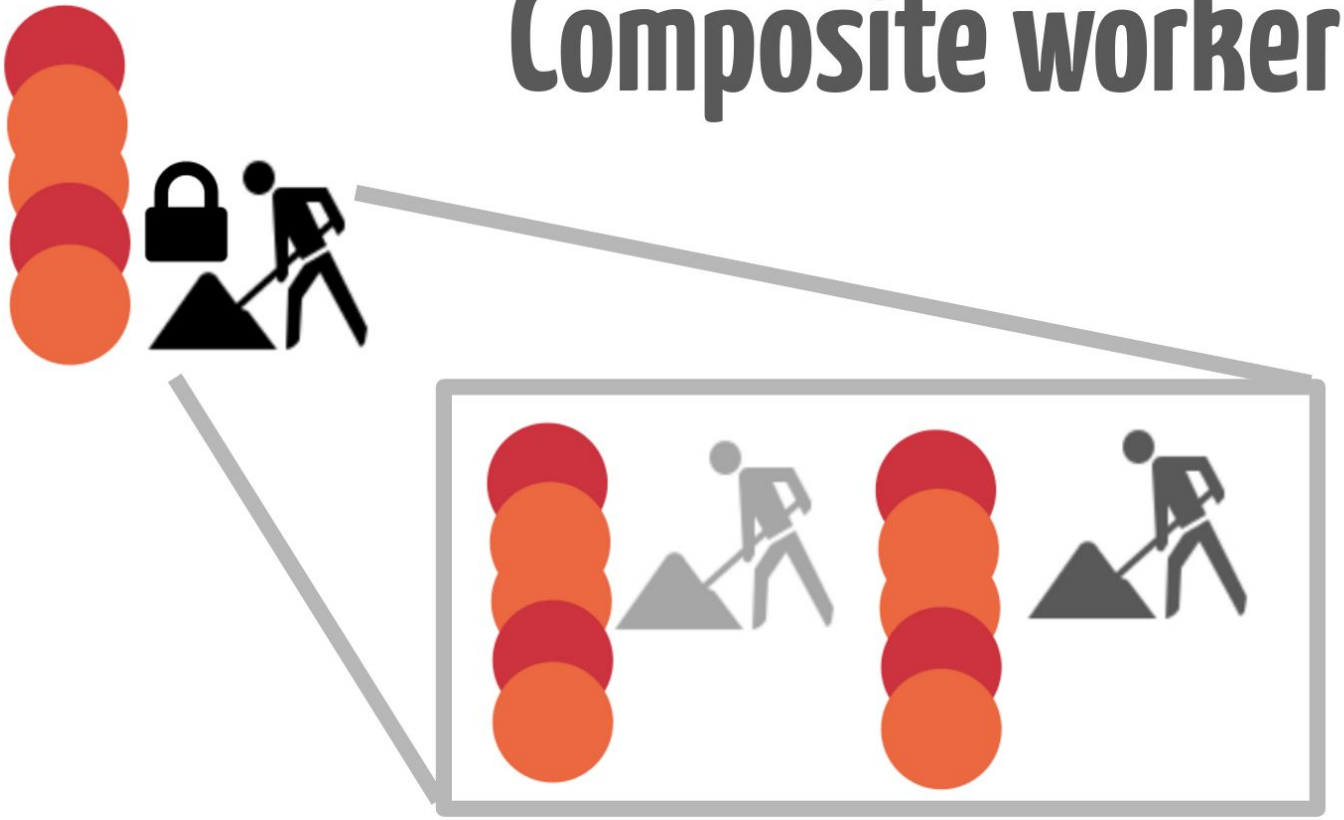
Shard read limits:

2 MBps

5 reads / s

Multiple applications lower application read rate

Composite worker



Composite application

Problem:

Running multiple applications against a stream consumes reduces the rate at which the stream can be read.

Composite application

Problem:

Running multiple applications against a stream consumes reduces the rate at which the stream can be read.

Solution:

Compose multiple applications into a single application.

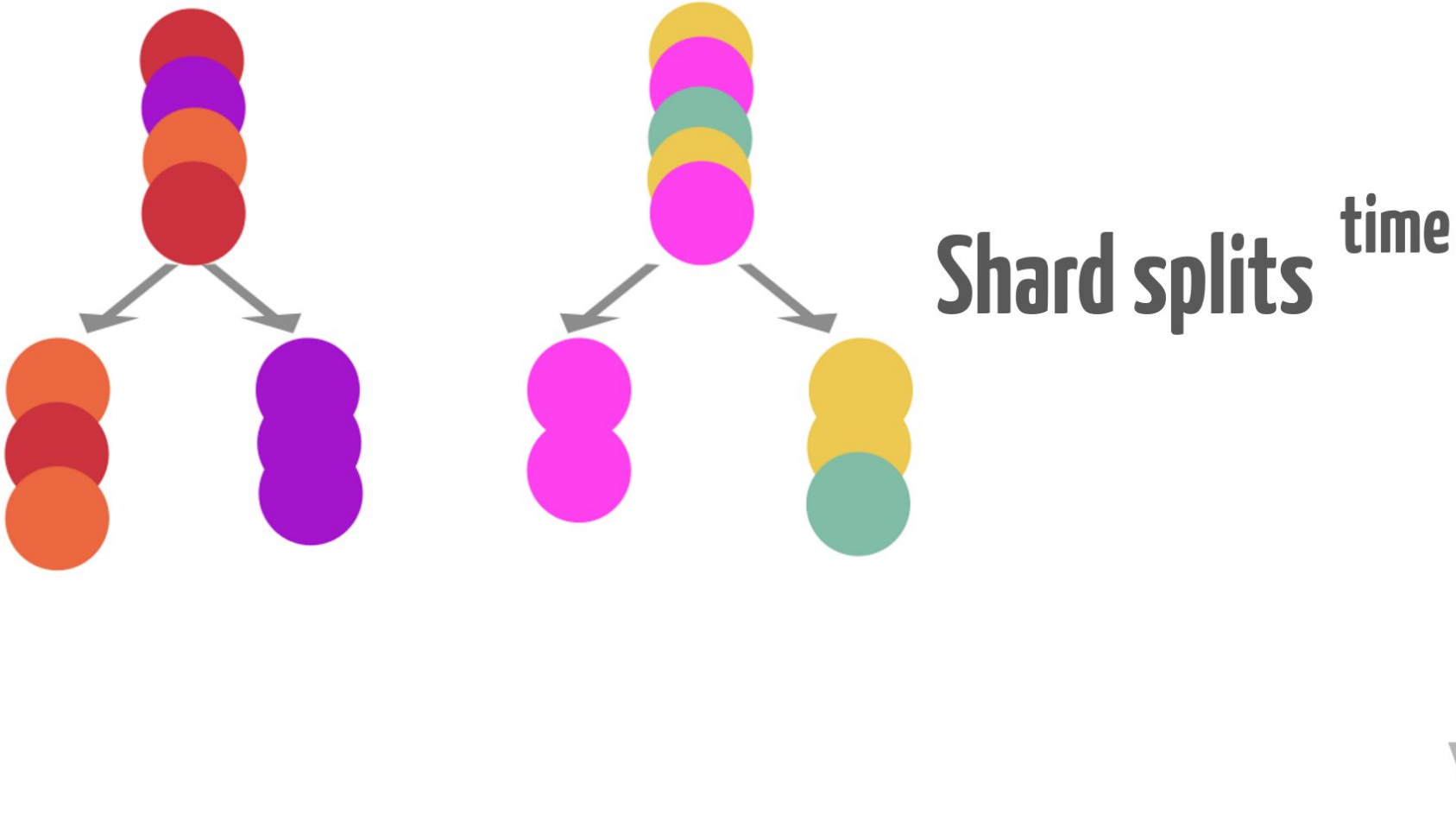
Composite application

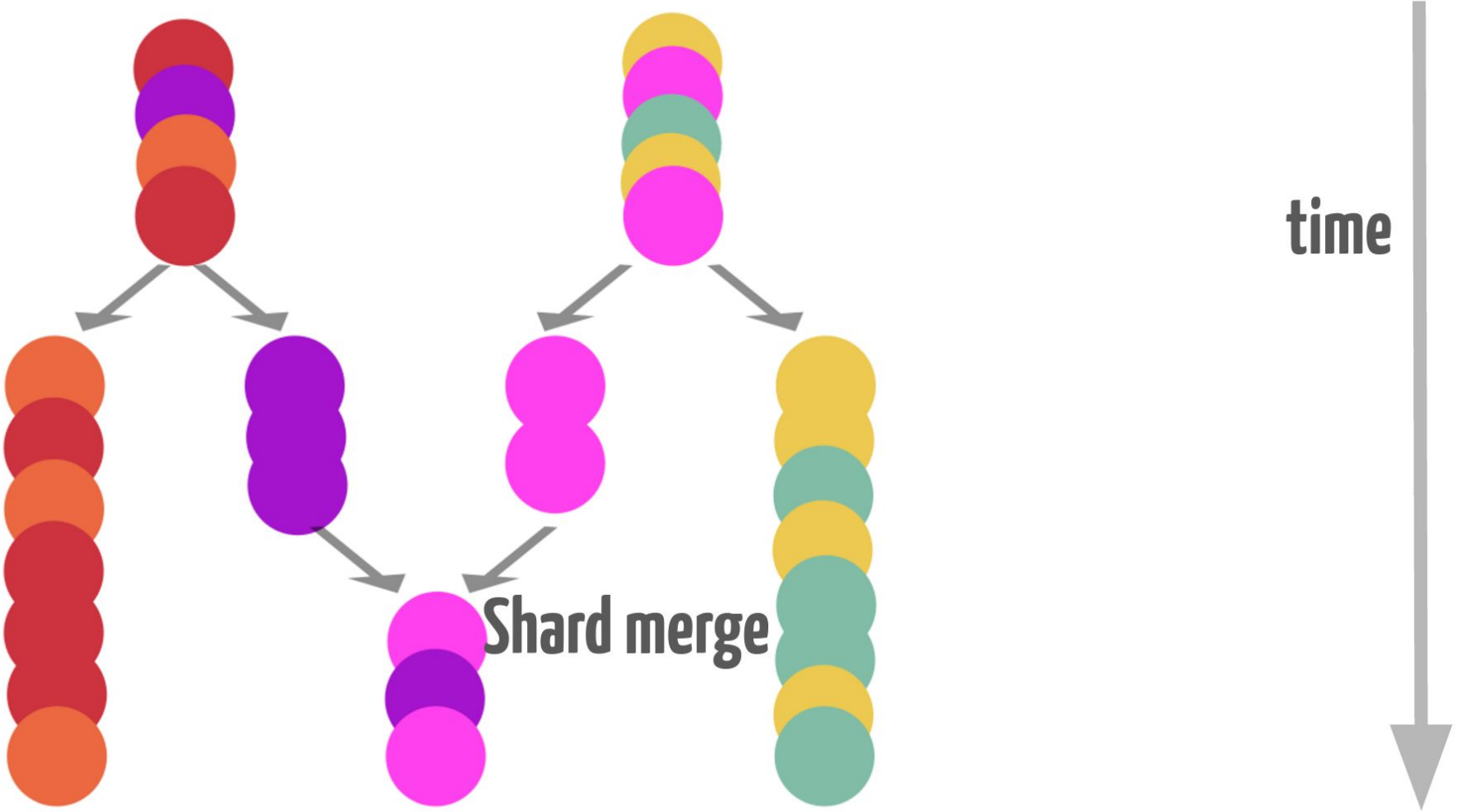
Consequences:

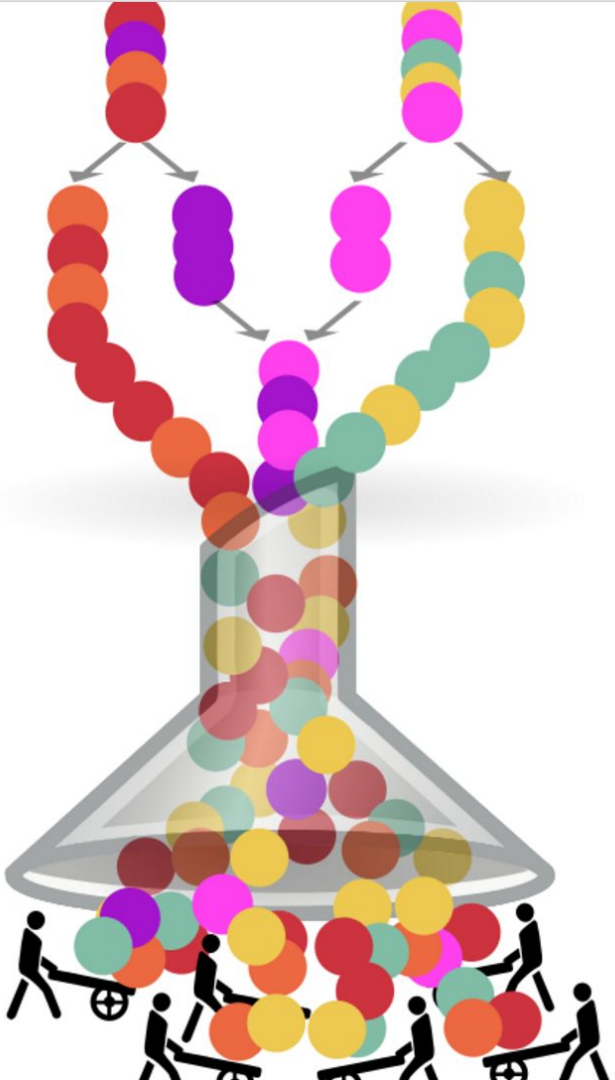
- A single stream read can feed many applications.
- The worker logic is complicated by needing to create a composite.
- The checkpoint logic and replay handling is complicated.
- Exception handling is complicated by the case of a subset of the applications failing.
- The rate at which different applications can process records are now coupled. This could slow down otherwise "fast" applications.

Shard splits / merges

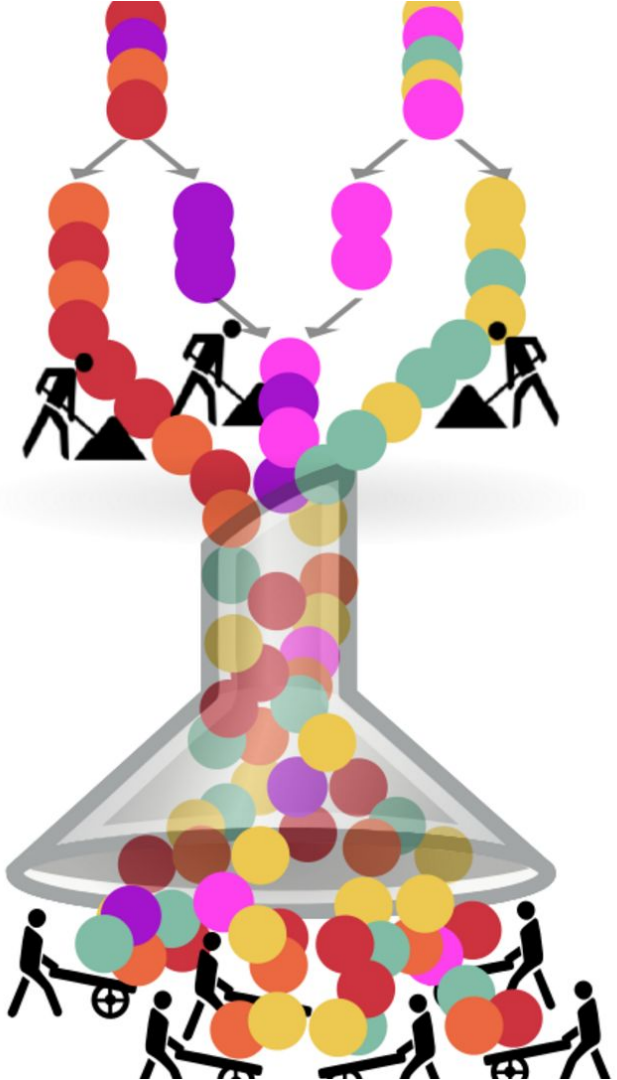
time


Shard splits
time

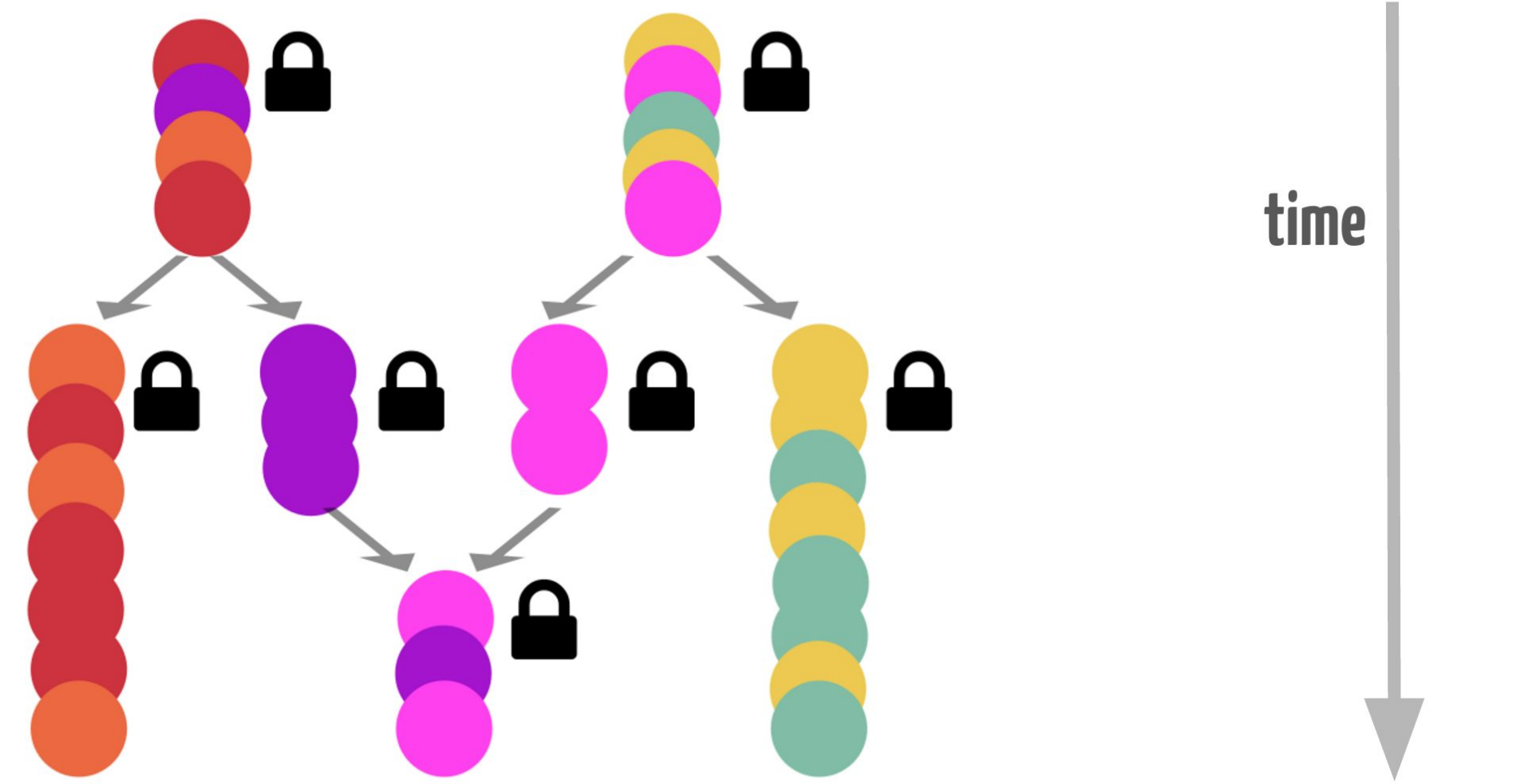

Shard merge
time

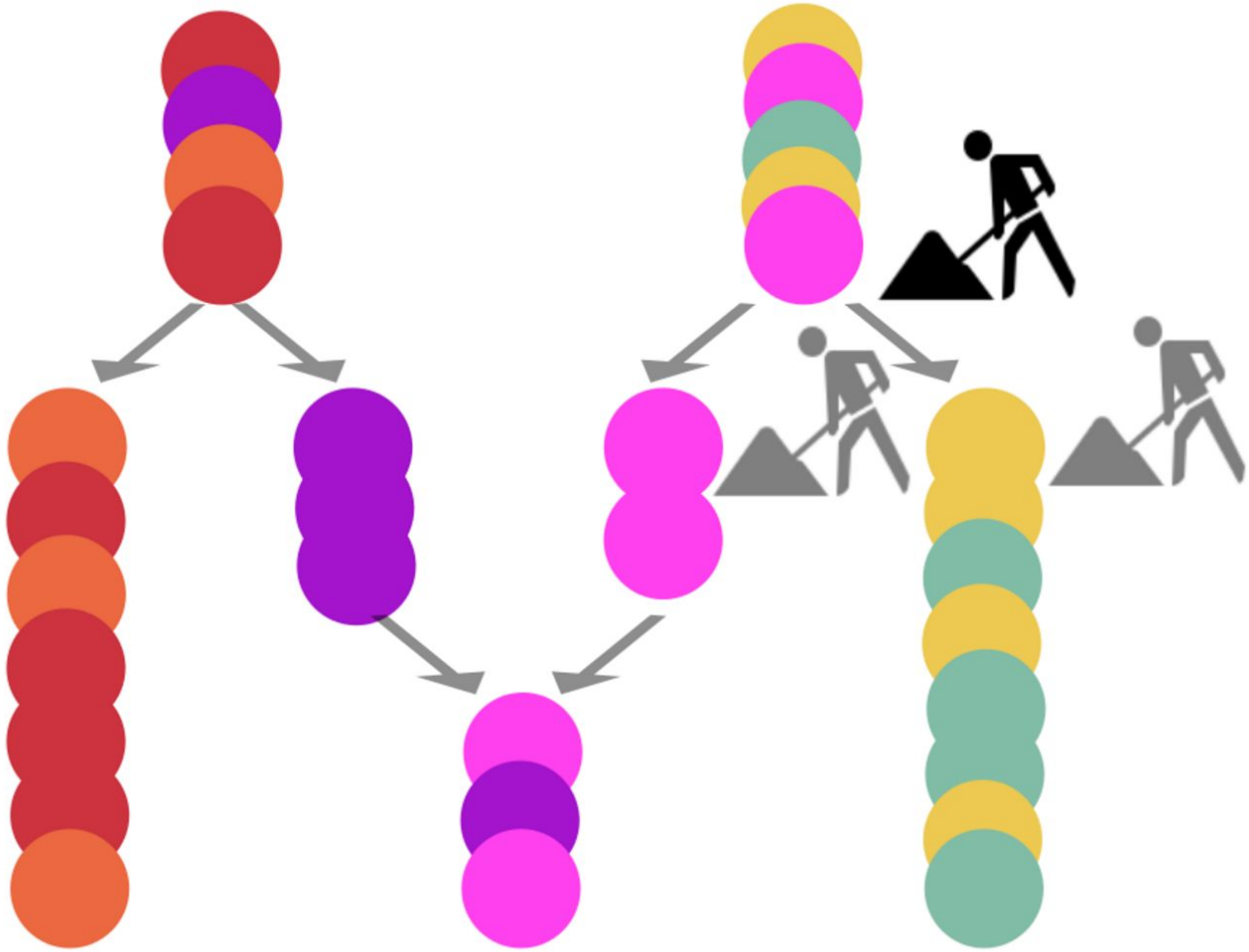


time



time


time



Shard state needs
to be handed off
to new workers

Track shard state by partition key

Problem:

As shards split and merge the corresponding worker shard state needs to be handed split and merged as well.

Track shard state by partition key

Problem:

As shards split and merge the corresponding worker shard state needs to be handed split and merged as well.

Solution:

Break shard state down by partition key so that it can be split and merged in a straightforward fashion.

Track shard state by partition key

Consequences:

- Structure of shard state does not need to change when shards split and merge.
- Requires that the shard state be designed with this in mind.
- Leads to fine-grained handling of state which may increase the overhead.
- Begs the question of how to deal with state that cannot be split by partition key.

Distributed Commit Log: *Patterns*

- Idempotent record processing
- Track checkpoints by partition key
- Sub-sequence processing
- Classify exceptions
- Retry failed records
- Count exceptions
- Composite application
- Track shard state by partition key

Thanks!

Thanks!

https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

Jay Kreps

http://www.thestrangeloop.com/2015/building-scalable-stateful-services.html

Caitie McCaffrey