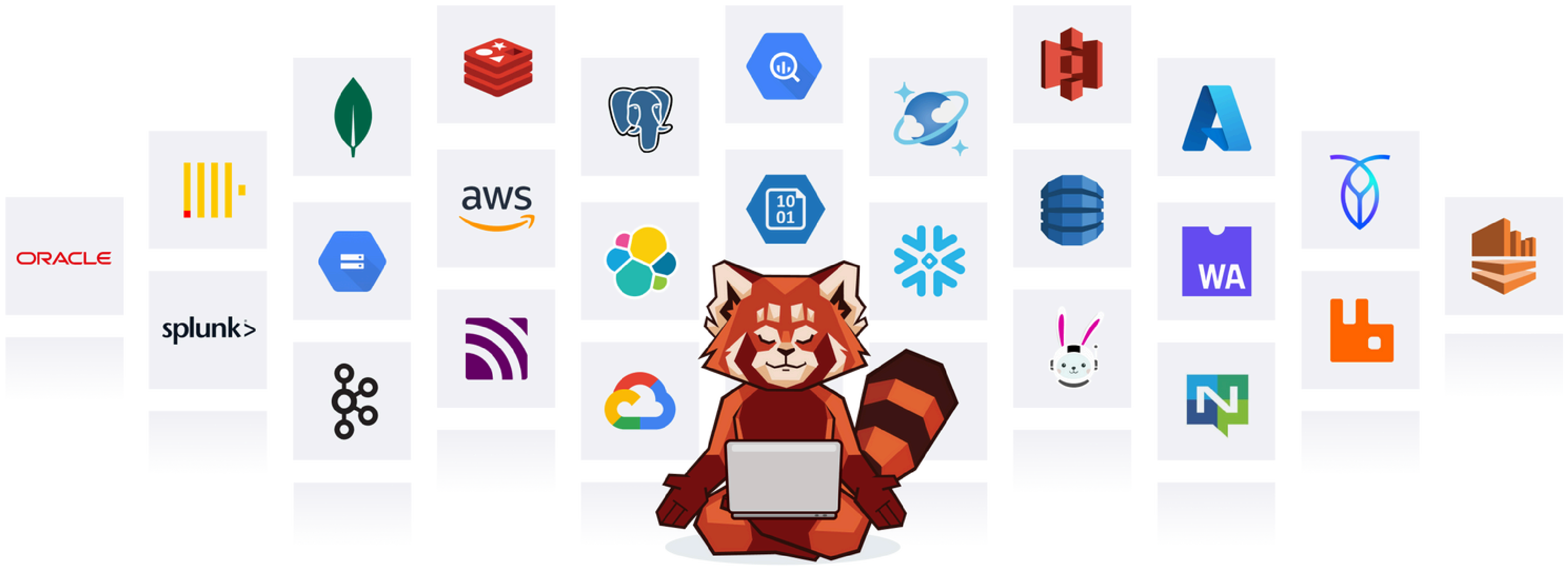
10 USE CASES FOR REDPANDA CONNECT



Redpanda Connect is a declarative data streaming service that solves many data engineering problems with simple, chained, stateless processing steps.

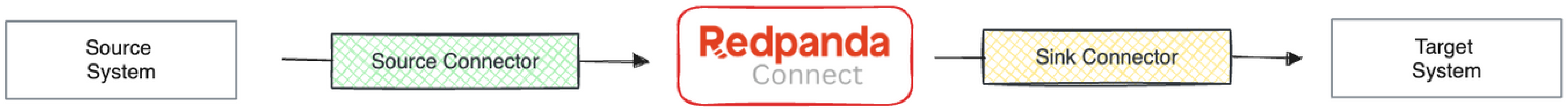
Formerly known as **Benthos**, Redpanda Connect is written in Go and deployed as a static binary with zero external dependencies. Its functionality overlaps with enterprise integration frameworks, log aggregators, and ETL workflow engines.

So, Redpanda Connect can complement these traditional data engineering tools or act as a simpler alternative.

What is a connector in Redpanda Connect?

Redpanda Connect allows you to compose streaming data pipelines that can move and transform data from one system to another. You can declare a pipeline with YAML. A pipeline composition can include one or more **connectors** and **processors**.

Connectors

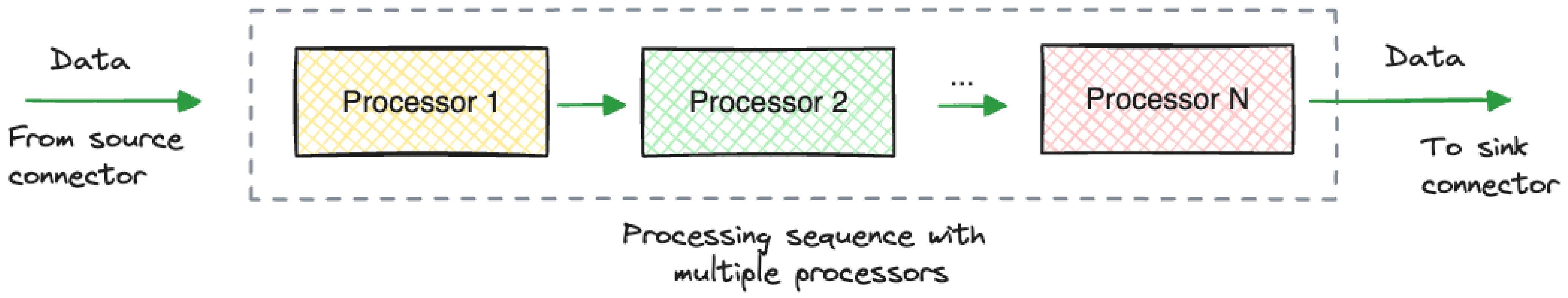


A **connector** integrates your Redpanda data with different data systems. There are two types of connectors: **input** connectors and **output** connectors.

- An **input connector** imports data from a source system into a Redpanda Connect. For example, you can download objects within an Amazon S3 bucket with ***aws_s3*** connector.

- An **output connector** exports data from a Redpanda Connect and transforms it into a format that the target system can use. For example, you can insert a record to a database table with ***sql_insert*** connector.

Processors



A processor is a function applied to messages passing through a pipeline. The function signature allows a processor to mutate or drop messages depending on the content of the message.

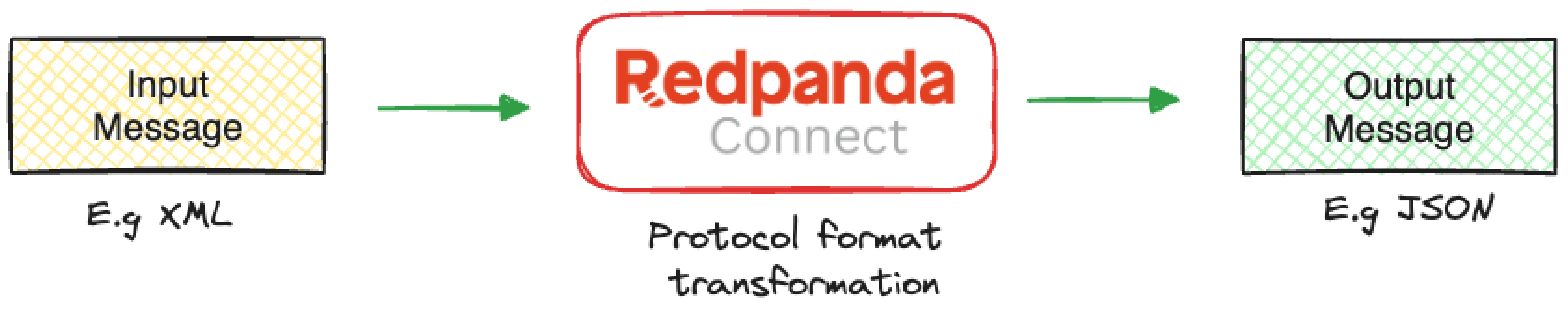
Processors are set via config, and depending on where in the config they are placed, they will be run either immediately after a specific input (set in the input section), on all messages (set in the pipeline section), or before a specific output (set in the output section). Most processors apply to all messages and can be placed in the pipeline section.

Here is an example of how a processor can be configured in the pipeline section:

```
pipeline:
  threads: 1
  processors:
    - label: my_cool_mapping
      mapping: |
        root.message = this
        root.meta.link_count = this.links.length()
```

Now that you got to know Redpanda Connect, let's see some use cases that you can build with Redpanda Connect.

1 Message format transformation



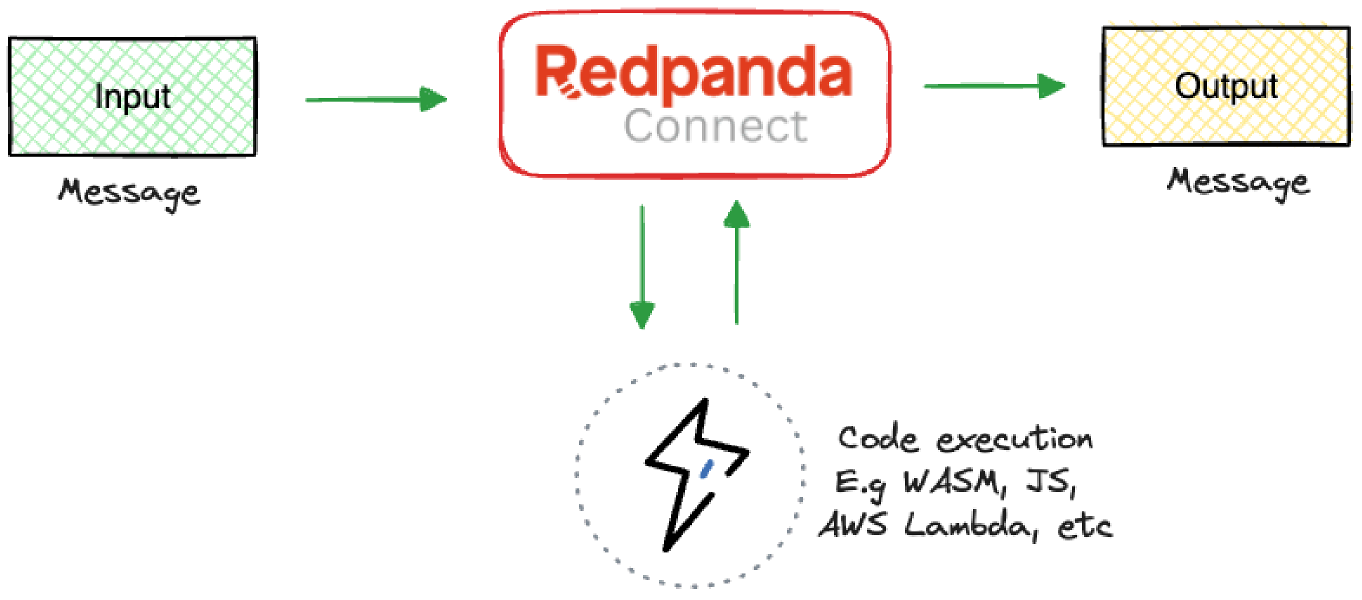
Different processors in Redpanda Connect can decode messages from one format, change their structure, and encode them back into a different format.

This feature can be extremely useful in several scenarios.

Data format normalization - data from different sources in varying formats needs to be converted into a common format for consistent handling and analysis.

System integrations - data from an older system (in an obsolete or less efficient format) needs to be transformed into a format compatible with a new system.
Supported formats Avro, JSON, Protobuf, Parquet, MessagePack, and XML.

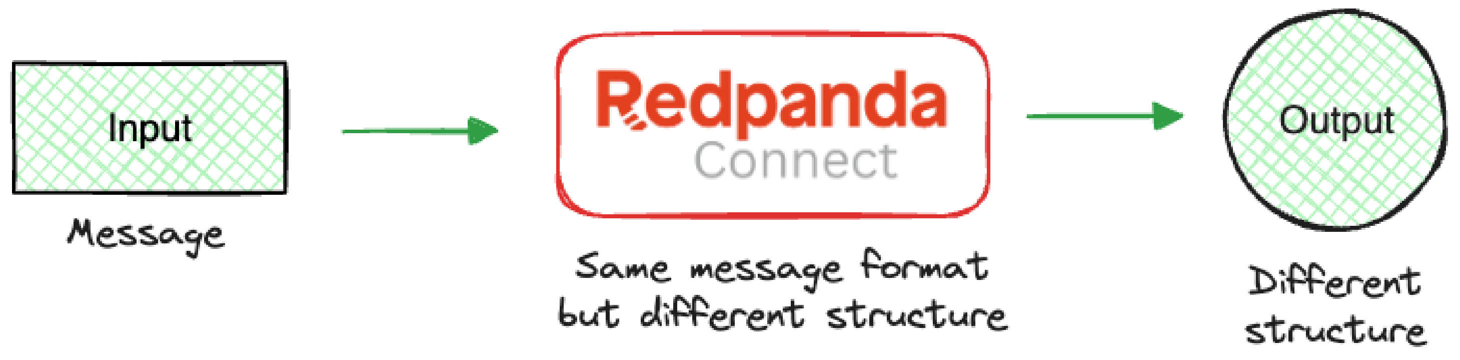
2 Execute a code for every message



Several processors in Redpanda Connect allow you to execute a code snippet or trigger a function for each message. The contents of the message are the payload of the request, and the result of the invocation will become the new contents of the message.

- ***wasm*** - Executes a function exported by a WASM module for each message.
- ***javascript*** - Executes a provided JavaScript code block or file for each message.
- ***command*** - Executes a Linux command for each message.
- ***aws_lambda*** - Invokes an AWS lambda for each message.

3 Message mapping and filtering



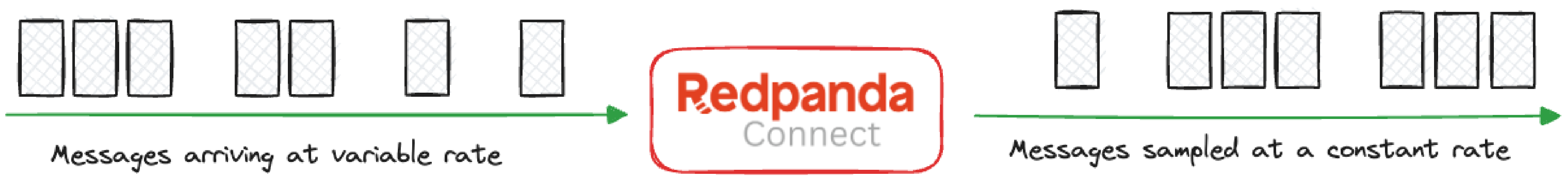
Execute various expressions on the content, transforming it into a different structure. The result of the expression will replace the original message content.

If the expression does not emit any value, the message is filtered, producing no result.

awk - Executes an AWK program on messages, querying and mutating message contents and metadata.
grok - Parses messages into a structured format by attempting to apply a list of Grok expressions.
jmespath - Executes a JMESPath query on JSON documents and replaces the message with the resulting document.
jq - Transforms and filters JSON messages using jq queries.

Additionally, the mapping and mutation processors can do the same with **Bloblang**, a powerful language that enables a wide range of mapping, transformation, and filtering tasks.

4 Message downsampling


Messages arriving at variable rate
Redpanda Connect
Messages sampled at a constant rate

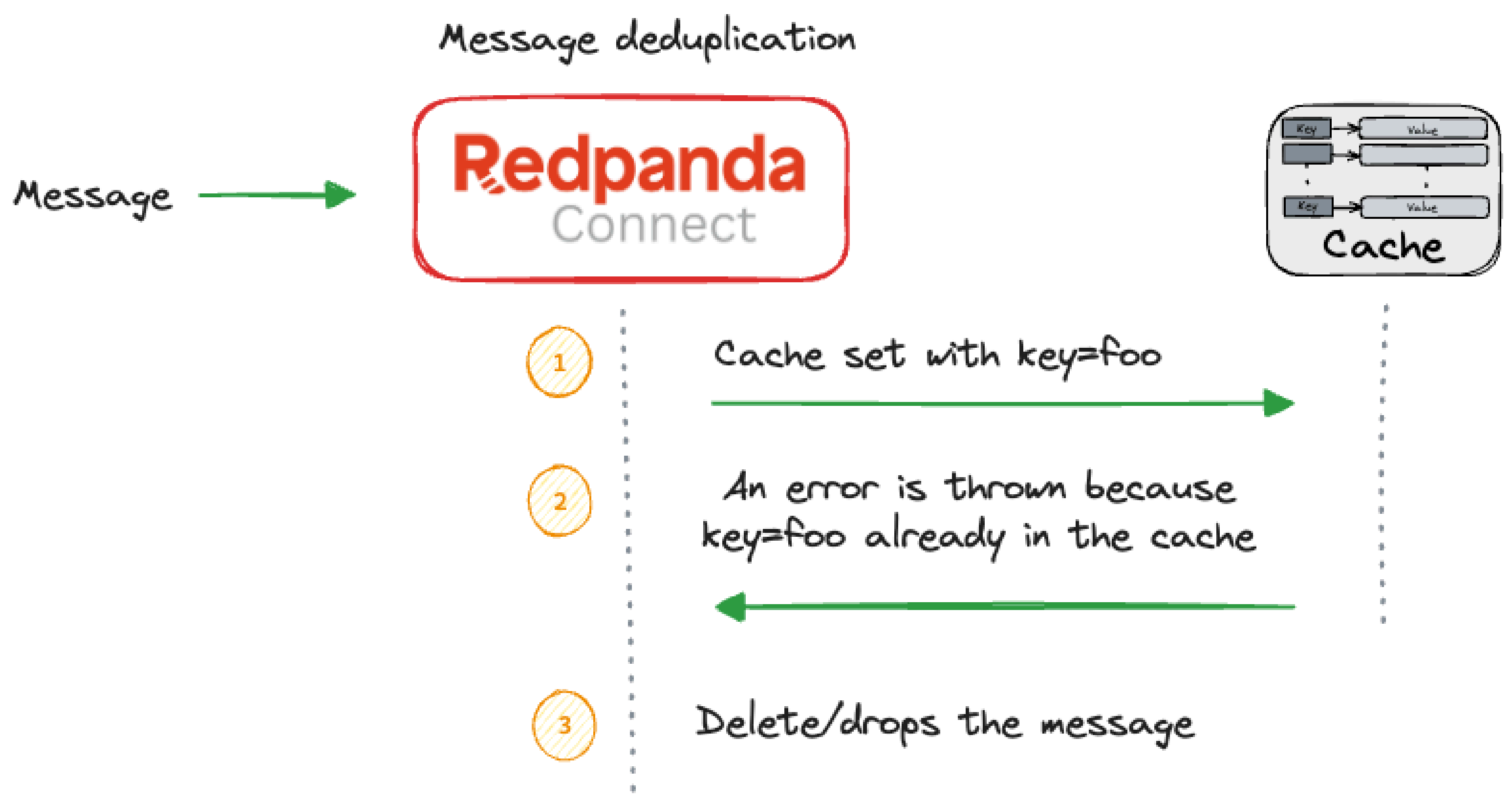
The ***rate_limit*** operator controls the throughput of a pipeline based on a specified limit. This is useful when the downstream system has a lower processing capacity than the data producer.

5 Working with caches

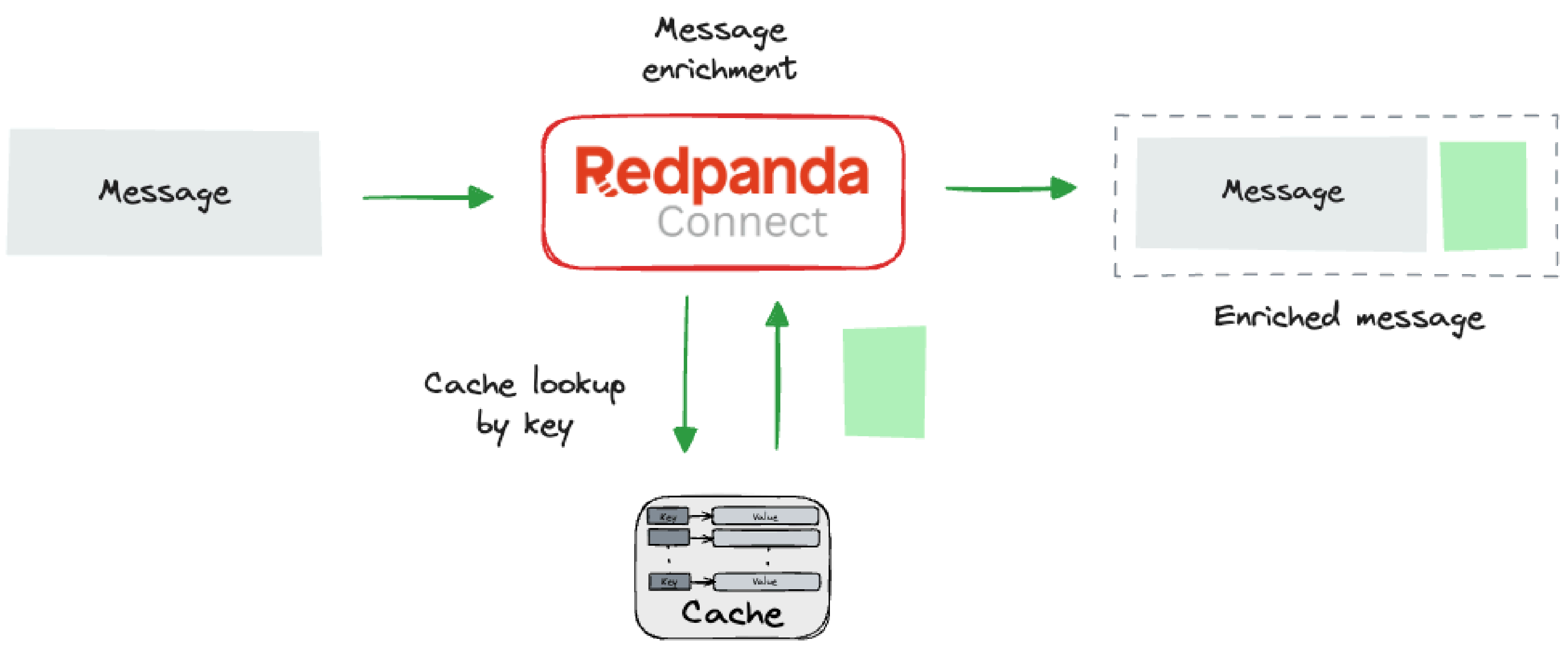
The cache operator performs operations against a cache for each message, allowing you to store or retrieve data within message payloads.

This allows for two important use cases: message deduplication and enrichment.

Deduplication can be achieved by adding a message to the cache with a key, extracted from the message itself. If the add operation fails, it indicates that the key already exists, and we can eliminate the duplicate using the mapping processor.



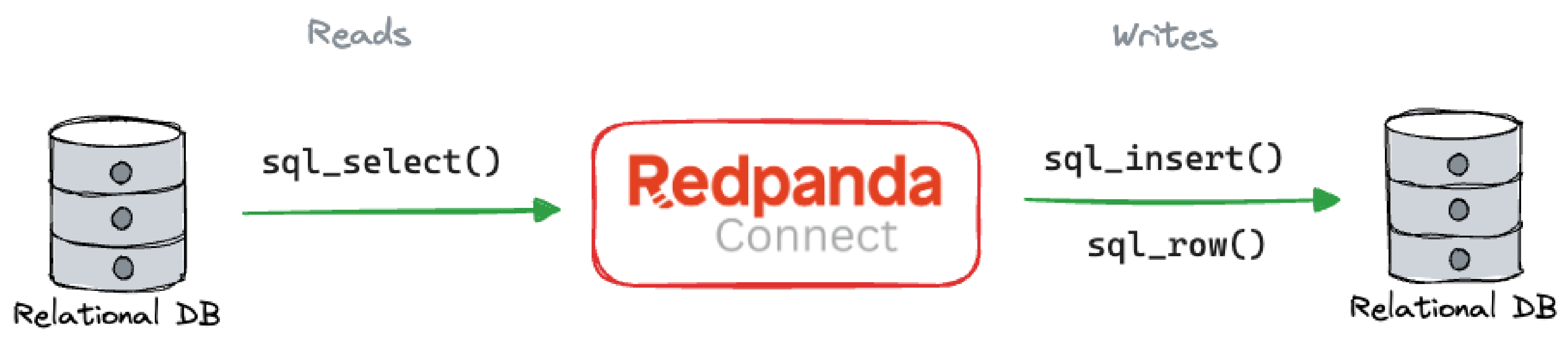
Similarly, you can **enrich** message payloads with content previously stored in a cache.



Redpanda Connect supports cache implementations including but not limited to Redis, AWS S3, AWS DynamoDB, Google Cloud Storage, Memcached, MongoDB, CouchDB, memory, SQL databases, and files.

6 Read from and write to databases

The ***sql_select*** operator executes a select query against a database and creates a message for each row received. For example, you could use this operator to read records from a table that were created within the last hour and forward them to a different processor in the pipeline.

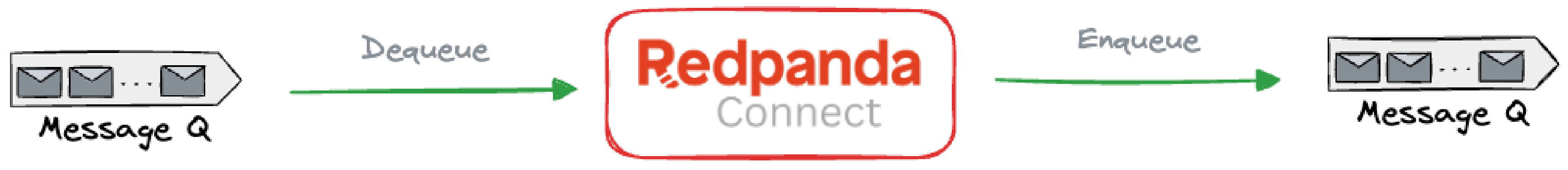


When it comes to writes, the ***sql_insert*** operator inserts rows into an SQL database for each message and leaves the message unchanged.

Additionally, you can use the sql_raw operator to run an arbitrary SQL query against a database, including **SELECT**, **INSERT**, **UPDATE**, and **DELETE** queries. For SELECT queries it returns the result as an array of objects, one for each row returned.

Both operators support databases including MySQL, Postgres, MSSQL, SQLite, Oracle, ClickHouse, Trino, and Azure CosmosDB. Additionally, you can work with other databases via relevant processors, such as couchbase, mongodb, and cassandra.

7 Enqueue and dequeue messages from queues



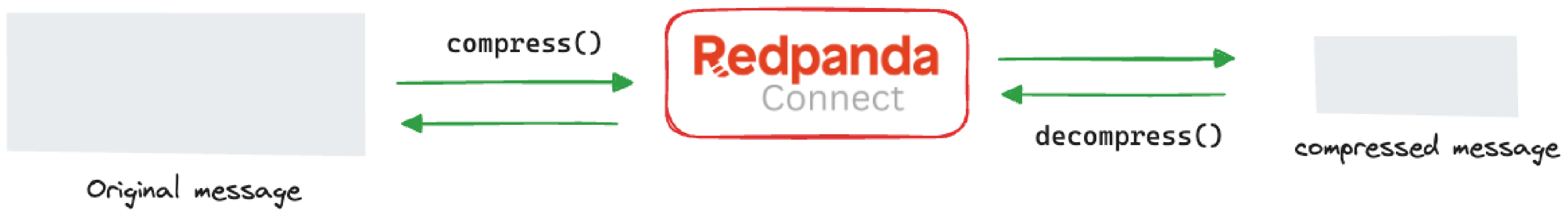
Redpanda Connect has the necessary inputs and outputs to both consume messages from message queues and produce messages to them.

This comes in handy if you want to bridge messages between different message brokers.

Supported implementations include AMQP, MQTT, AWS S3, AWS SNS, AWS Kinesis, Azure Queue Storage, Google Pubsub, NATS, Pulsar, Redis Pubsub, Kafka, and ZeroMQ.

8 Compress/decompress and archive/extract messages

The ***compress*** operator compresses messages using the chosen algorithm. The supported compression algorithms include flate, gzip, lz4, pgzip, snappy, and zlib.



The ***decompress*** operator does the opposite—decompresses messages according to the selected algorithm. Supported decompression algorithms are [bzip2 flate gzip lz4 pgzip snappy zlib]

Message archival

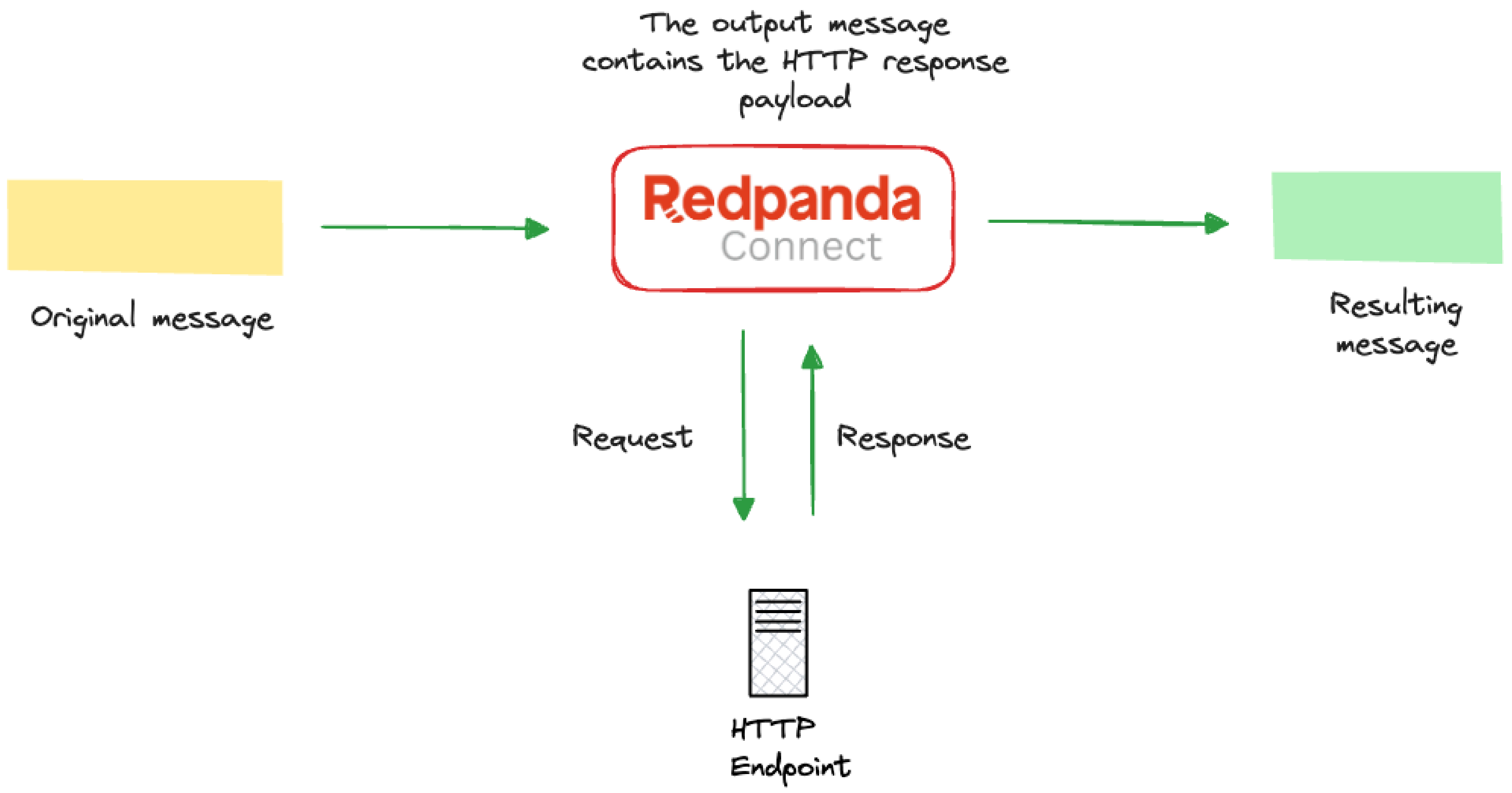
The ***archive*** operator archives all the messages of a batch into a single message according to the selected archive format. The output can be a tar or a zip file, a JSON array, a concatenated text, or a binary message.

The ***unarchive*** operator does the opposite.

9 Retry

The ***retry*** operator attempts to execute a series of child processors until success.
For instance, imagine a child processor attempting to write a message into a database, but it receives an error because the database is offline. After a specified backoff period, the retry operator reattempts the same message at a configured interval until the operation succeeds.

10 Call an HTTP endpoint



The ***https*** operator performs an HTTP request using a message batch as the request body and replaces the original message parts with the body of the response.

This is useful in situations where you need to invoke a Webhook or fetch data from an API.

Want to learn more about Redpanda Connect?

Visit Redpanda Docs

Follow Me @dunithd /in/dunithd/