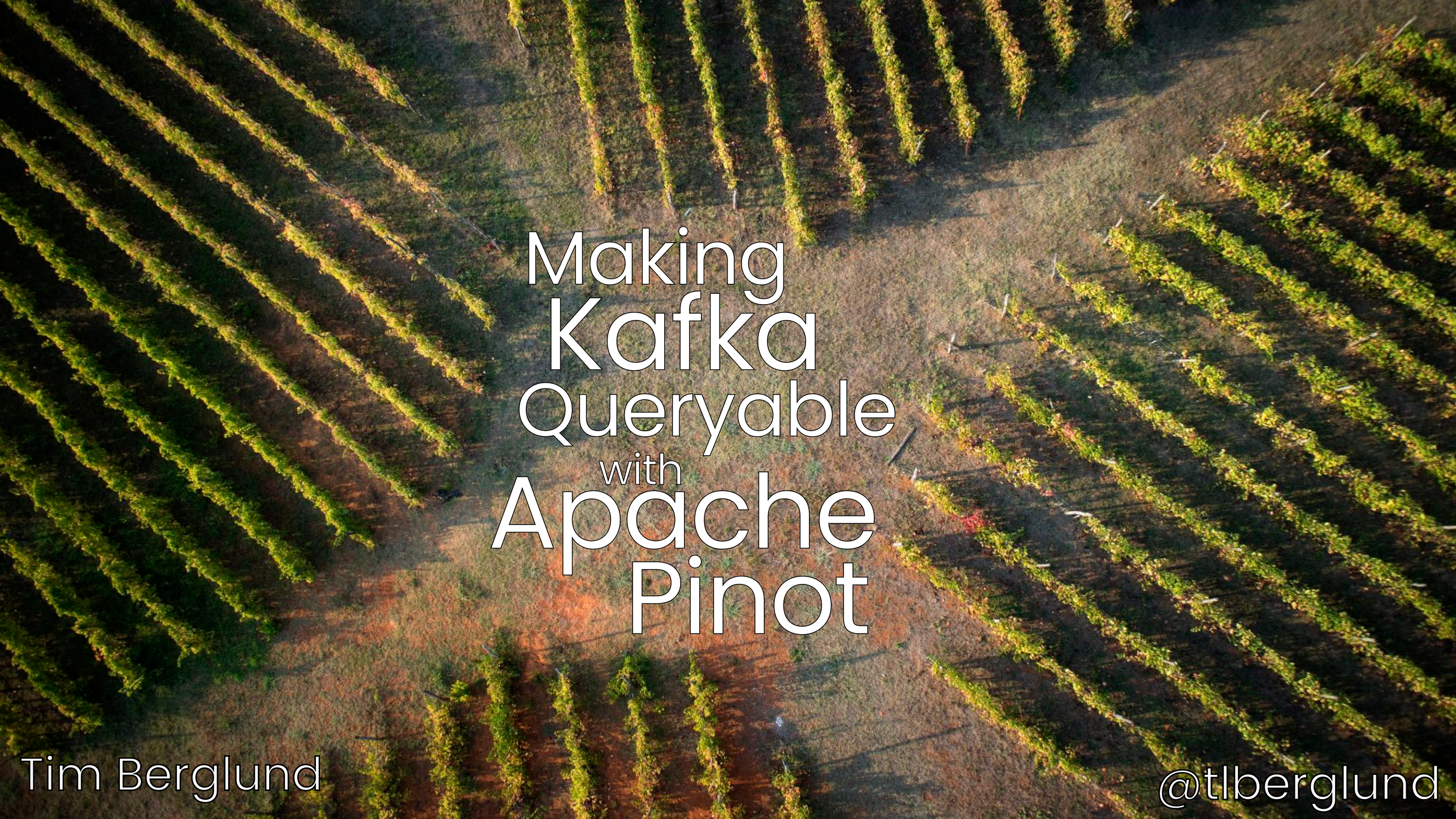


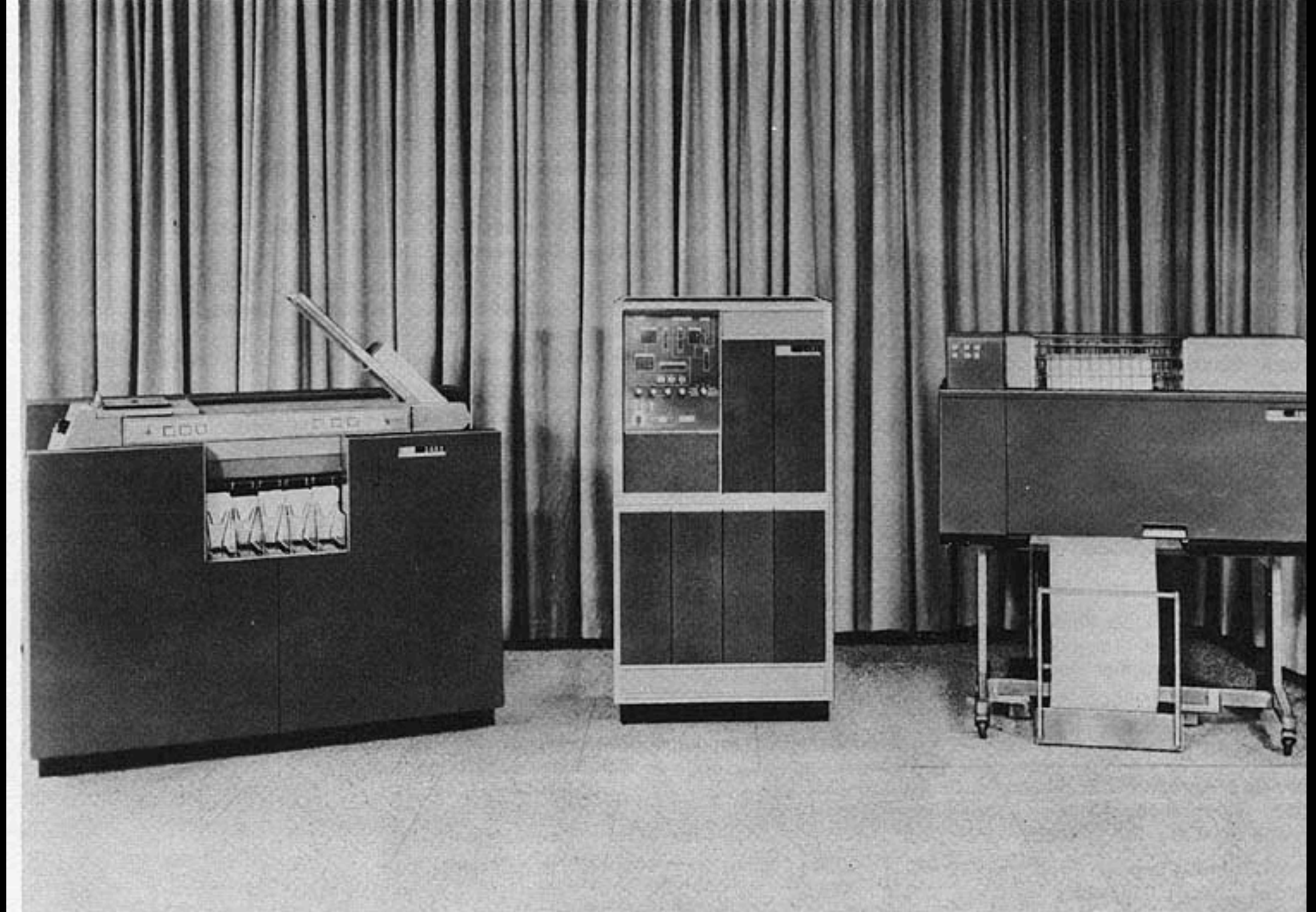
Tim Berglund

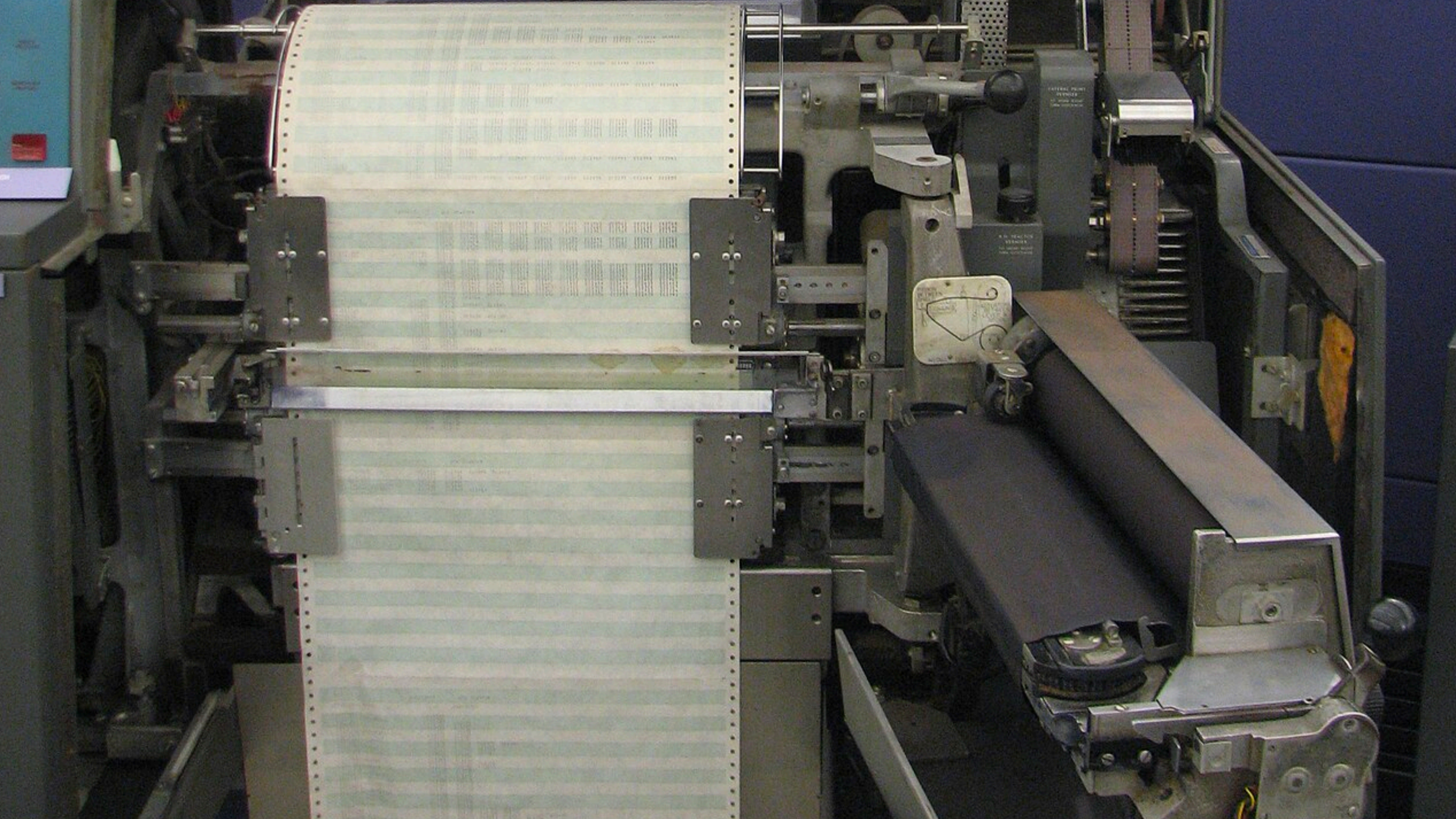
@tlberglund

Swag Challenge

1. Live-tweet this session (ideally in a thread, mention @apachepinot or @tlberglund)
   -or-
2. Write an equivalent summary and post on LinkedIn after the session is over
3. Send me a link to the post or thread in StarTree Community Slack
4. The first five people to respond will get 45 coins in the StarTree Swag Store

a brief history





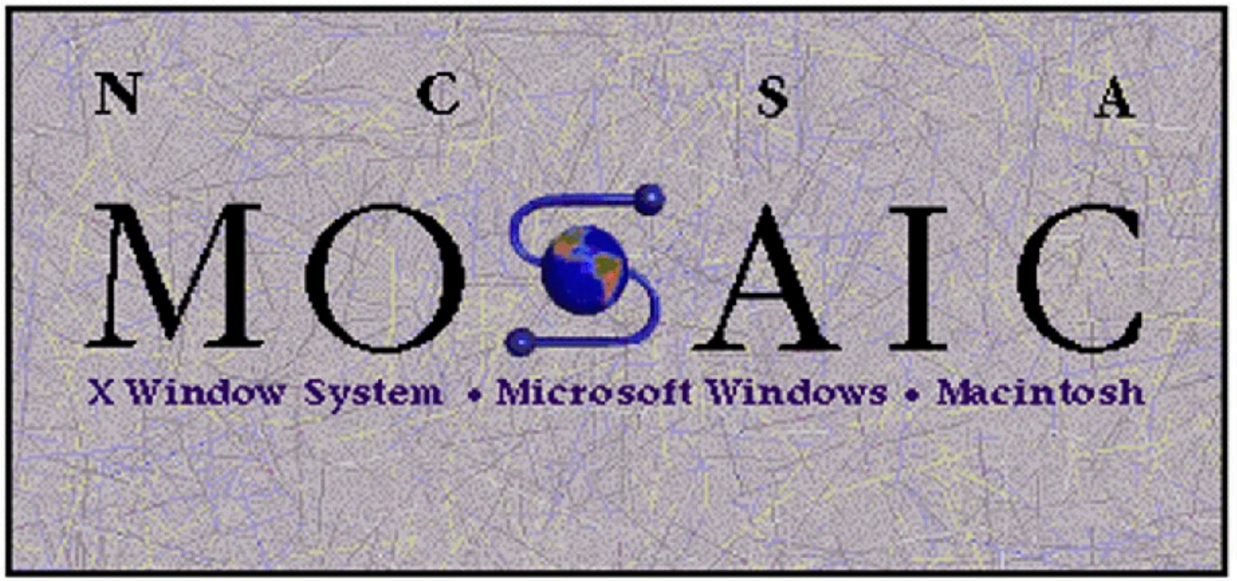
IBM

SWAN

NCSA Mosaic Home Page - NCSA Mosaic

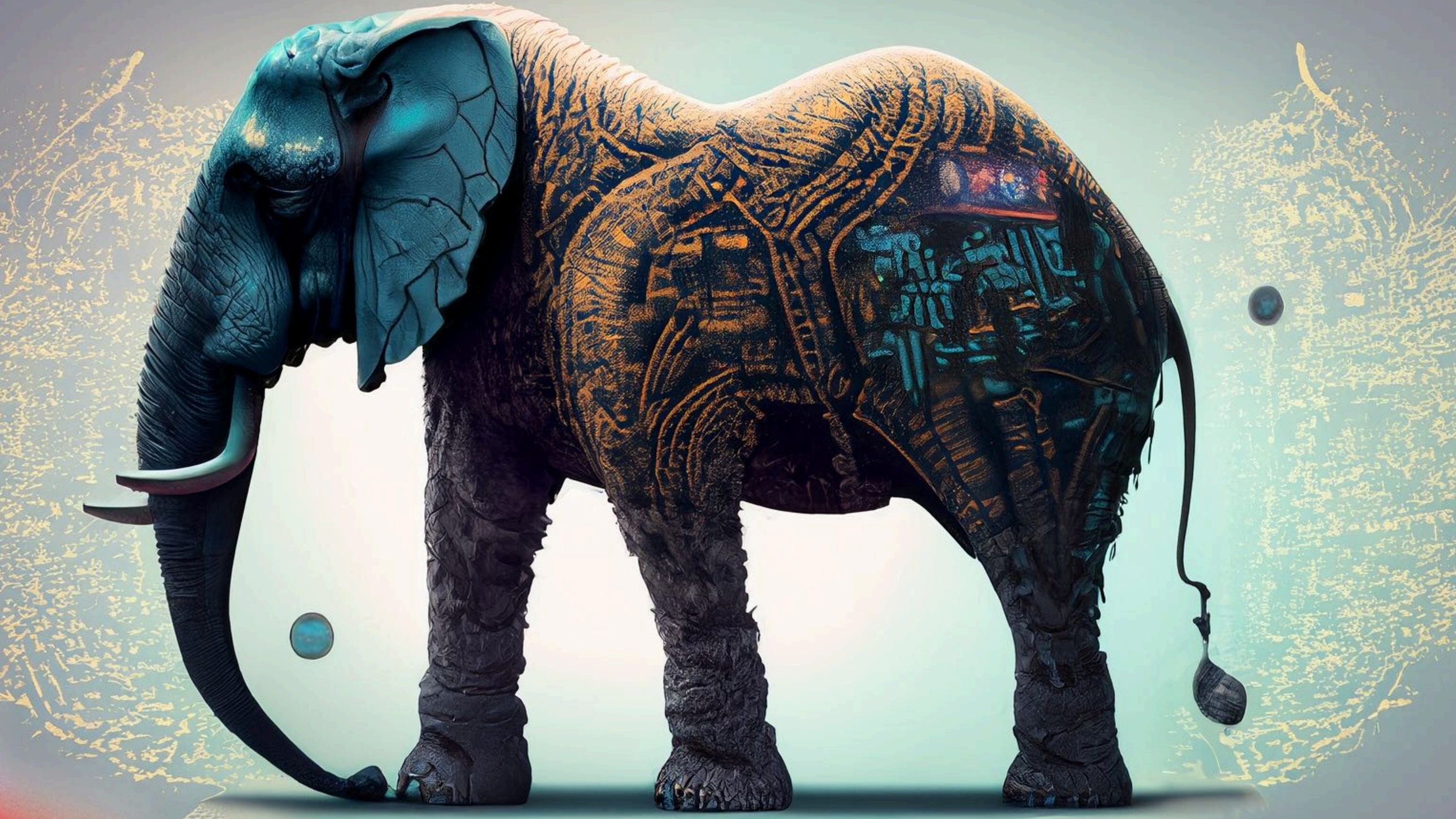
File Edit Source Manager View Navigate Tools Hotlists Help

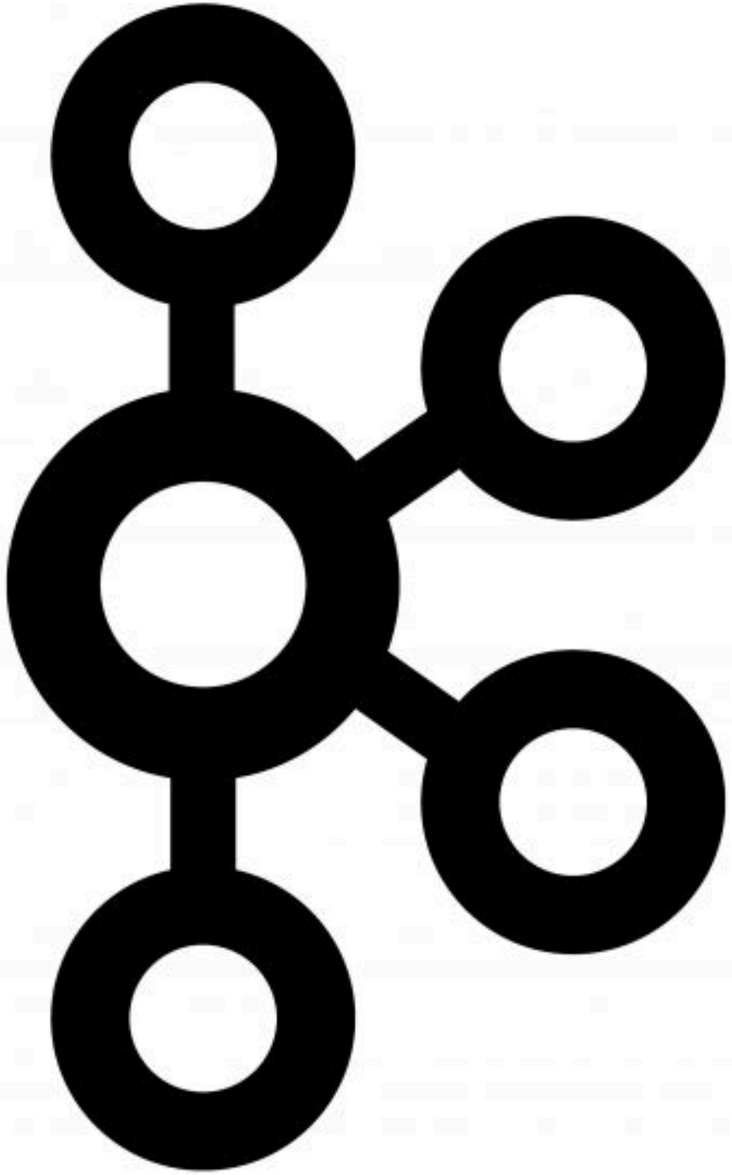
http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/

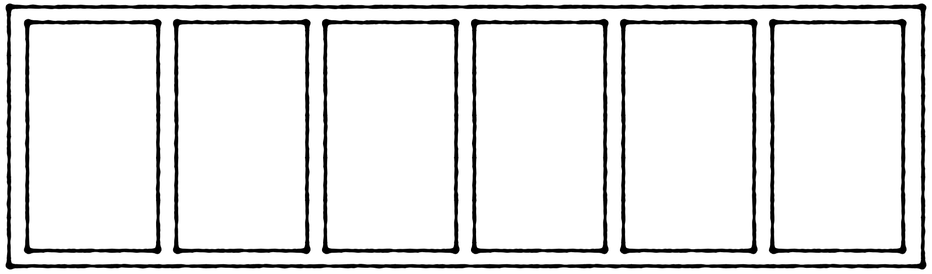


Welcome to NCSA Mosaic, an Internet information browser and World Wide Web client. NCSA Mosaic was developed at the National Center for Supercomputing Applications at the University of Illinois in Urbana-Champaign. NCSA Mosaic software is copyrighted by The Board of Trustees of the University of Illinois (UI), and ownership remains with the UI.

NCSA | Mosaic | Photo CD | Metasearch



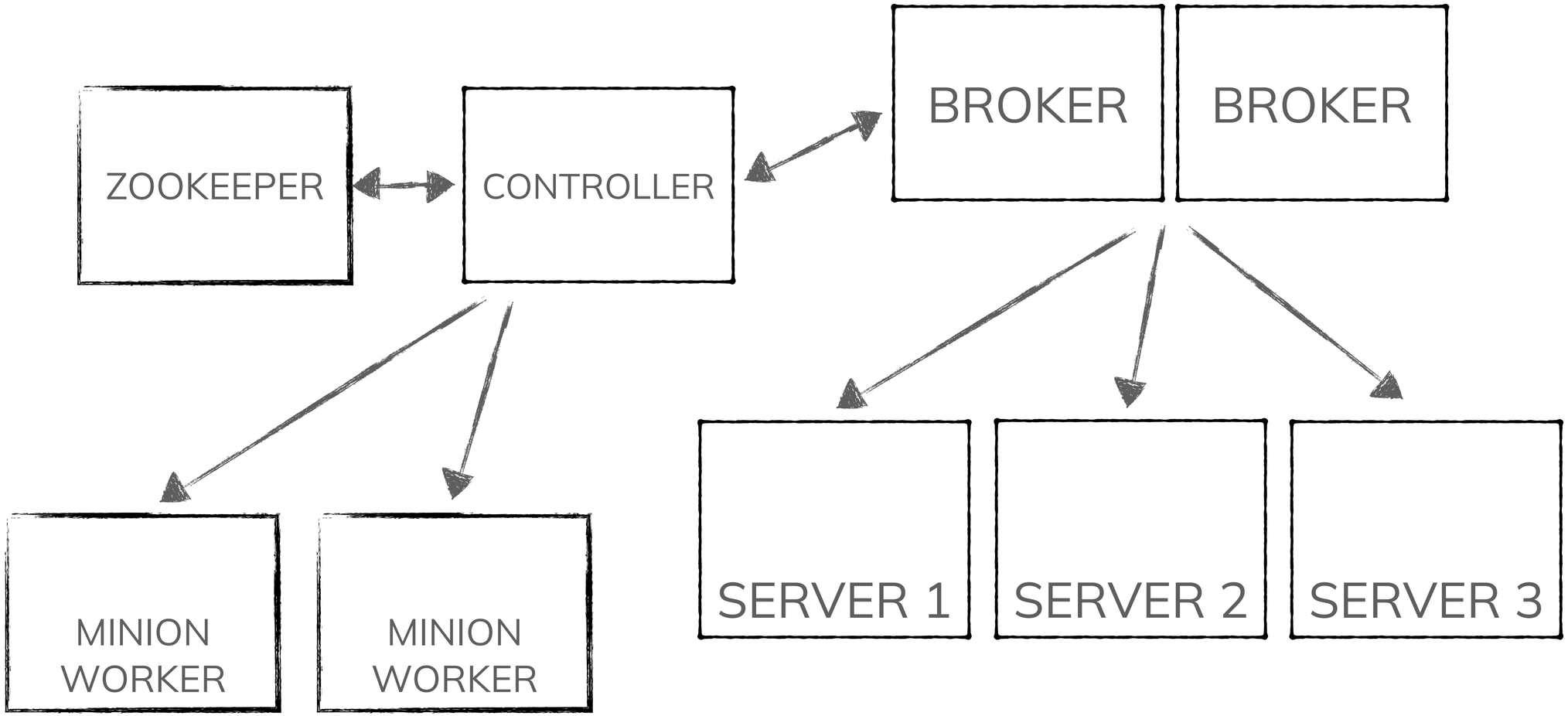




pinot

pinot architecture

Pinot Architecture



Tables

QUERY RESULT

Search…

movieId	rating	releaseYear	title
1	8.199999809265137	1968	Once Upon a Time in the West
2	4.599999904632568	2008	10,000 B.C.
3	7.199999809265137	1955	It's Always Fair Weather
4	6.800000190734863	1999	Tarzan
5	5.900000095367432	1994	Only You
6	6.300000190734863	2011	One Day
7	3.700000047683716	2005	Torrente 3
8	5.900000095367432	2006	The Lake House

Tables

QUERY RESULT

Search...

movieId	rating	ratingTime
802	8.434251841577007	2023-09-25 23:49:00.997
780	3.5665633255624307	2023-09-25 23:49:01.14
780	1.2298736923091265	2023-09-25 23:49:01.141
658	6.0137728823762915	2023-09-25 23:49:01.141
782	4.142424115195889	2023-09-25 23:49:01.141
900	4.875388029793274	2023-09-25 23:49:01.141
782	5.13352750511166	2023-09-25 23:49:01.141
294	9.841956512394209	2023-09-25 23:49:01.141

Tables

TABLE SCHEMA

JSON Format

Search...

Column	Type	Field Type	Multi Value
movieId	INT	Dimension	false
title	STRING	Dimension	false
releaseYear	INT	Dimension	false
rating	DOUBLE	Metric	false

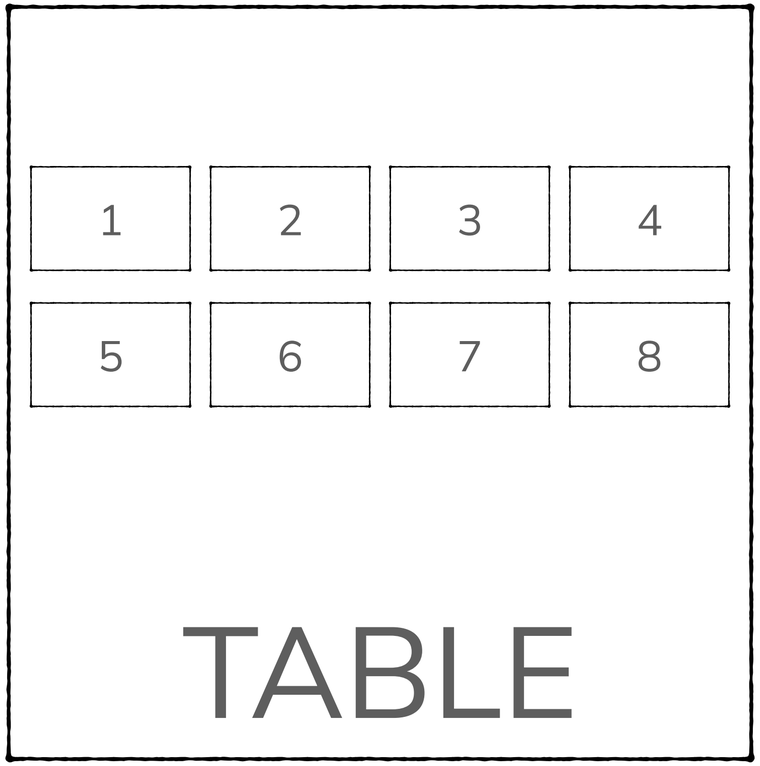
Tables

- The basic unit of data storage in Pinot
- Composed of rows and columns
- Expected to scale to arbitrarily large row counts
- Defined using a `schema` and `tableConfig` JSON file
- Three varieties: *offline*, *real-time*, and *hybrid*
- Every column is either a *metric*, *dimension*, or *date/time*

Segments

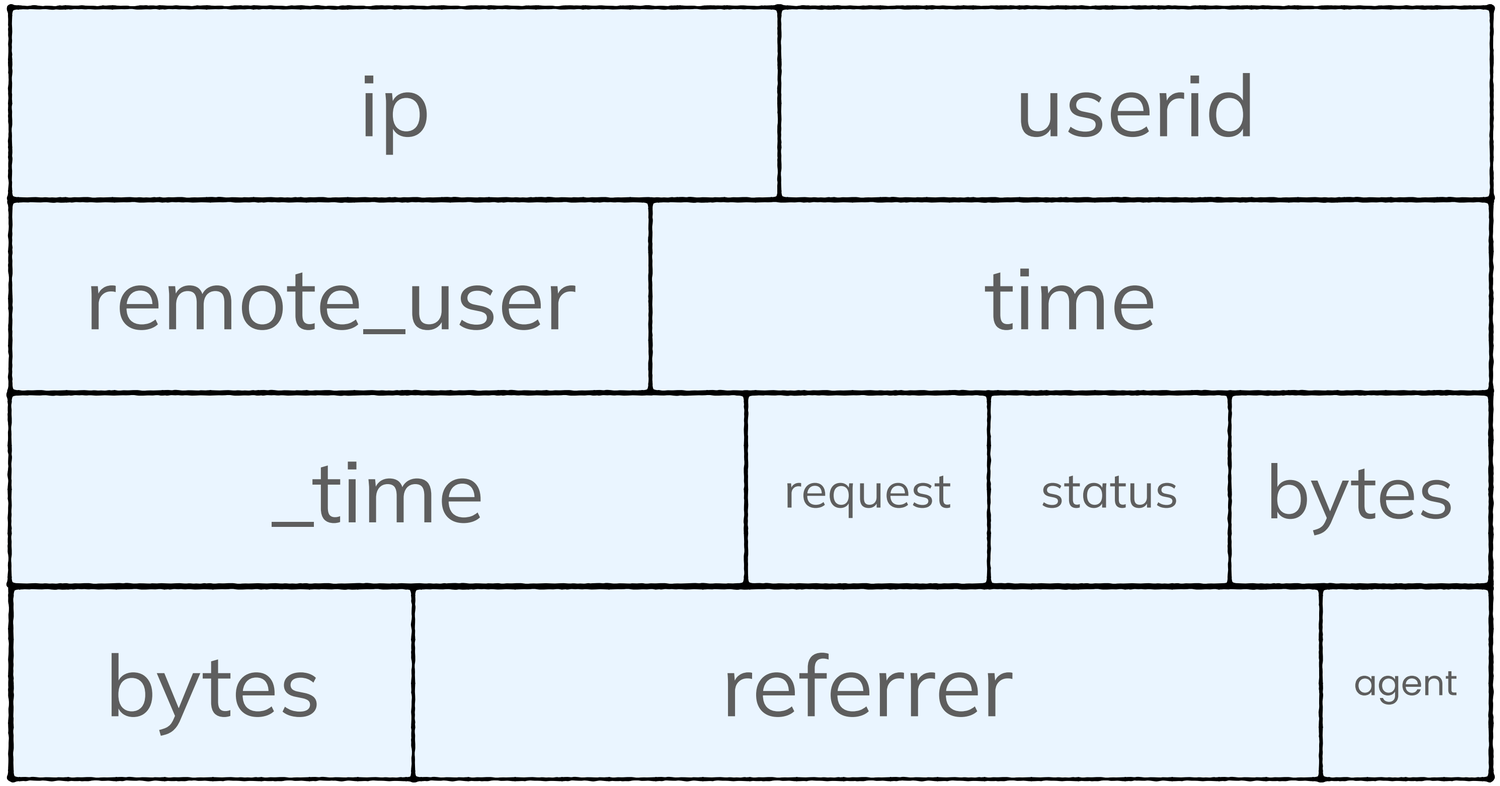
- Tables are split into units of storage called *segments*
- Similar to shards or partitions, but transparent to you, the user
- Ingested from static files or from a streaming source
- Segments are immutable once written
- Segments have a configurable retention period

Segments

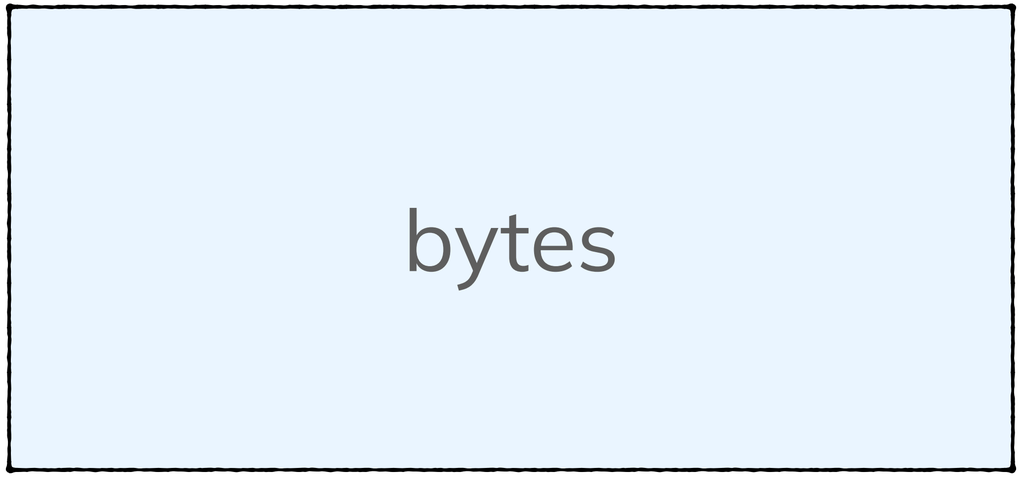


Segment Structure

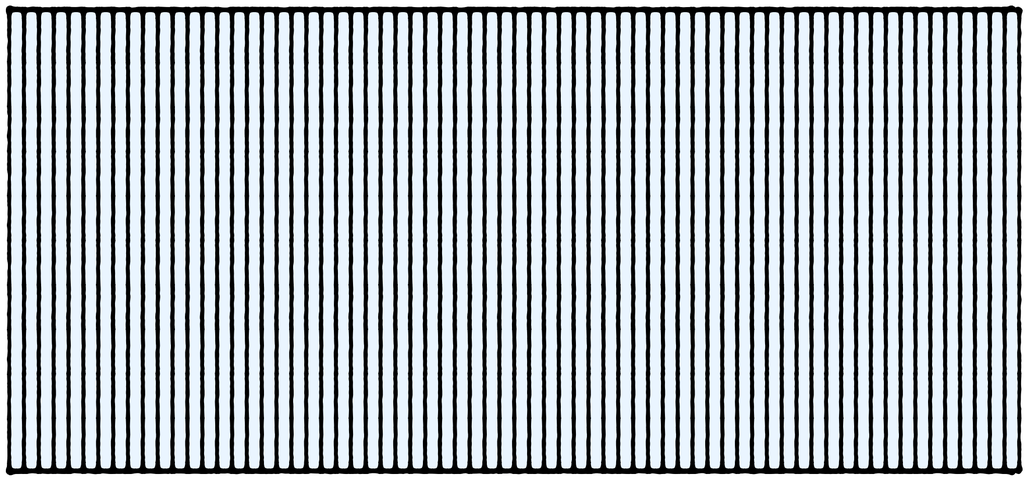
```
...,
{
   "ip": "111.173.165.103",
   "userid": 10,
   "remote_user": "-",
   "time": "3271",
   "_time": 3271,
   "request": "GET",
   "status": "406",
   "bytes": "1289",
   “referrer": "-",
   "agent": "Mozilla/5.0"
},
...,
```



Segment Structure



Segment Structure



S

6538

1294

32534

2914

8625

1343

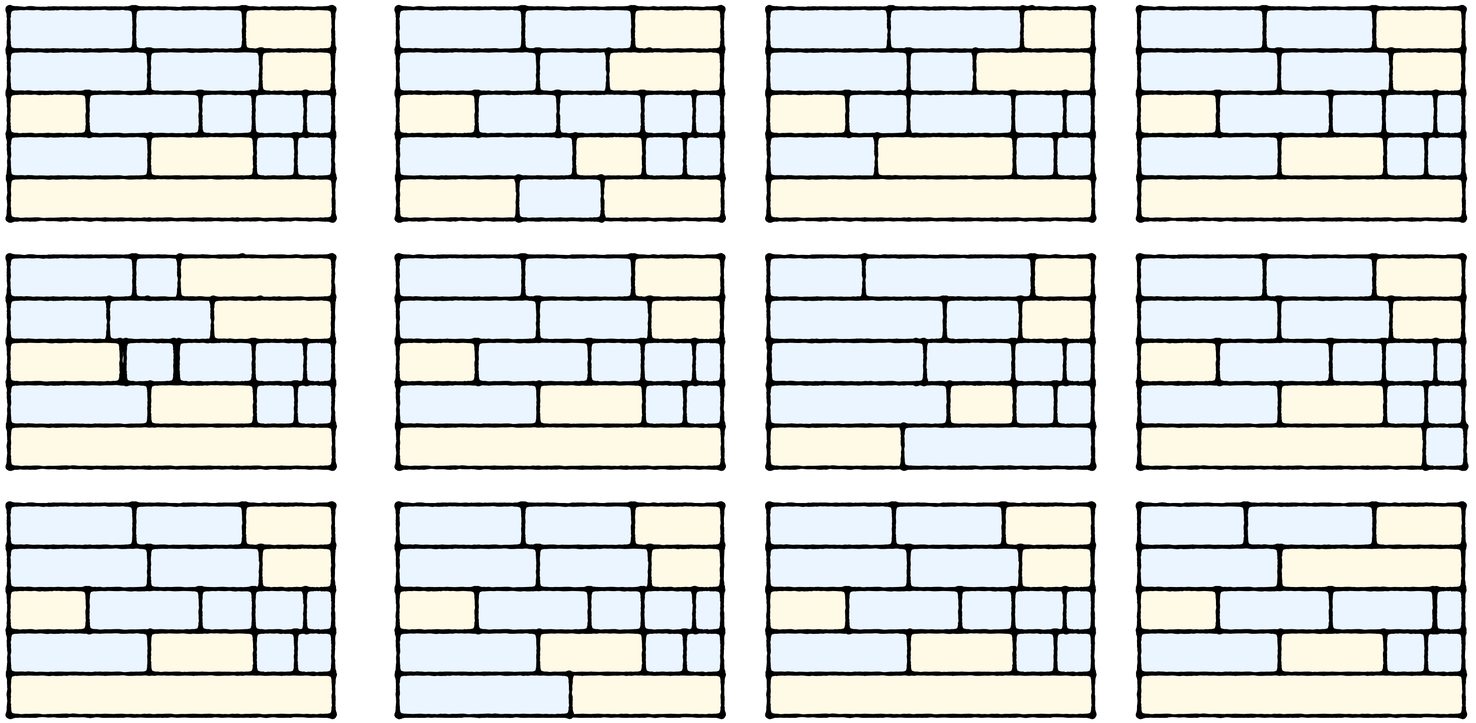
3234

2543

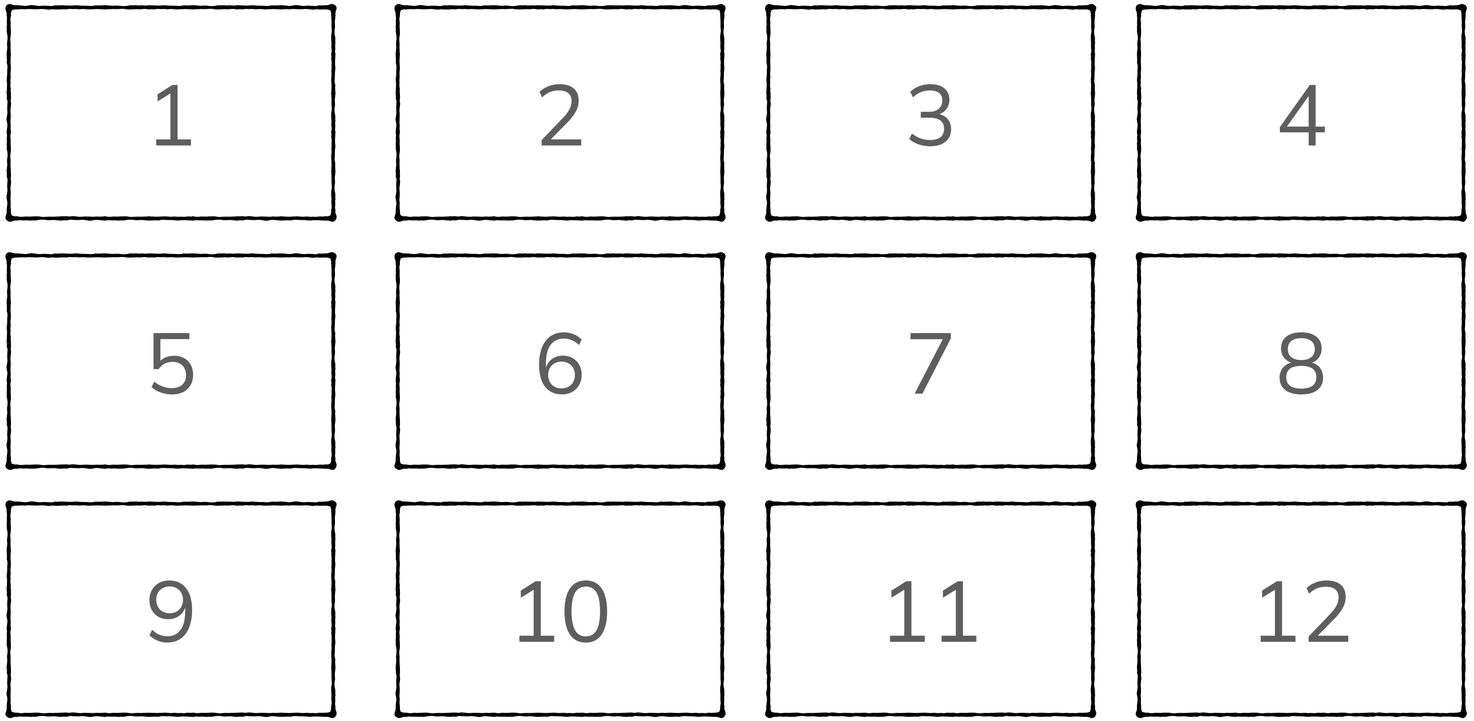
5387

918

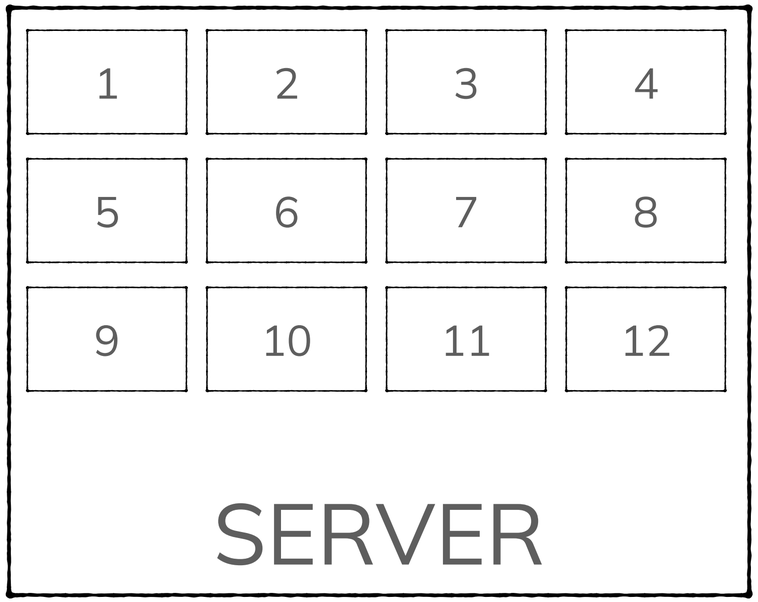
Segments



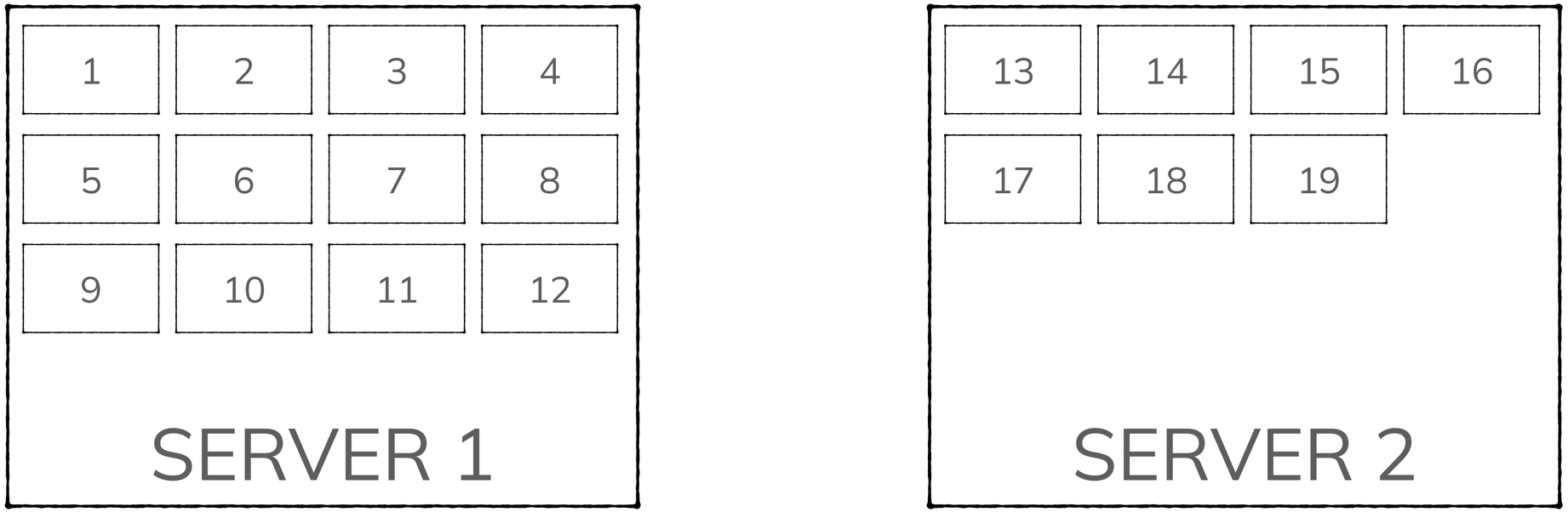
Segments



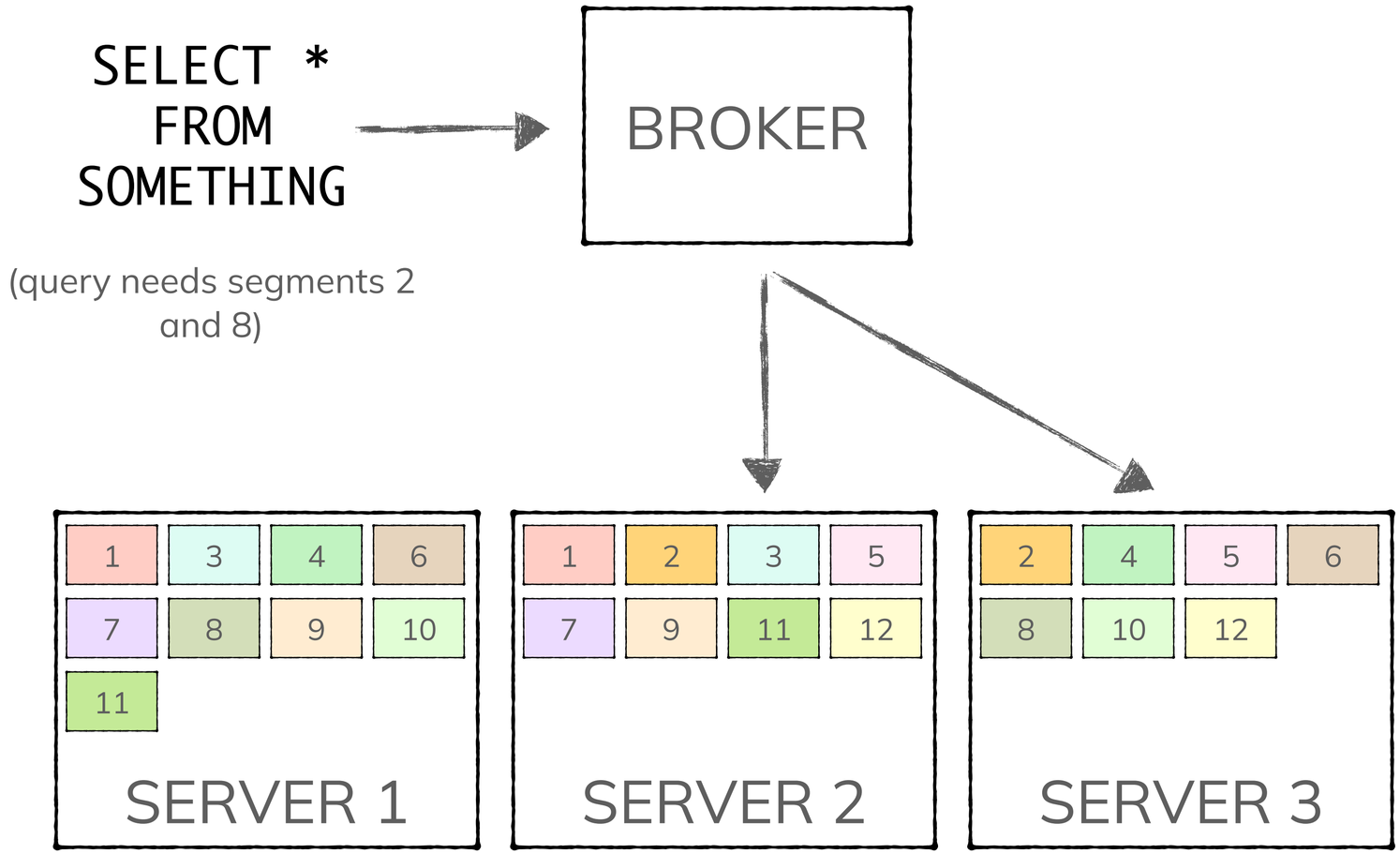
Servers

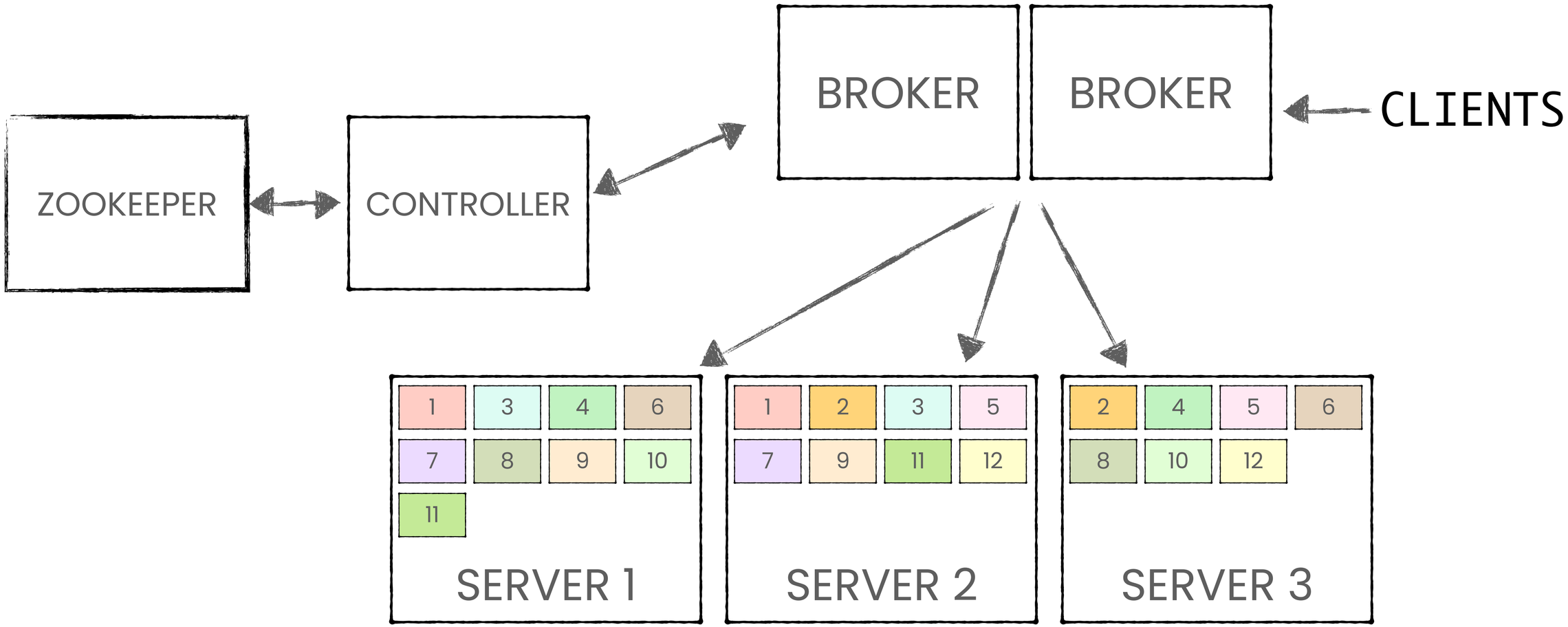


Servers

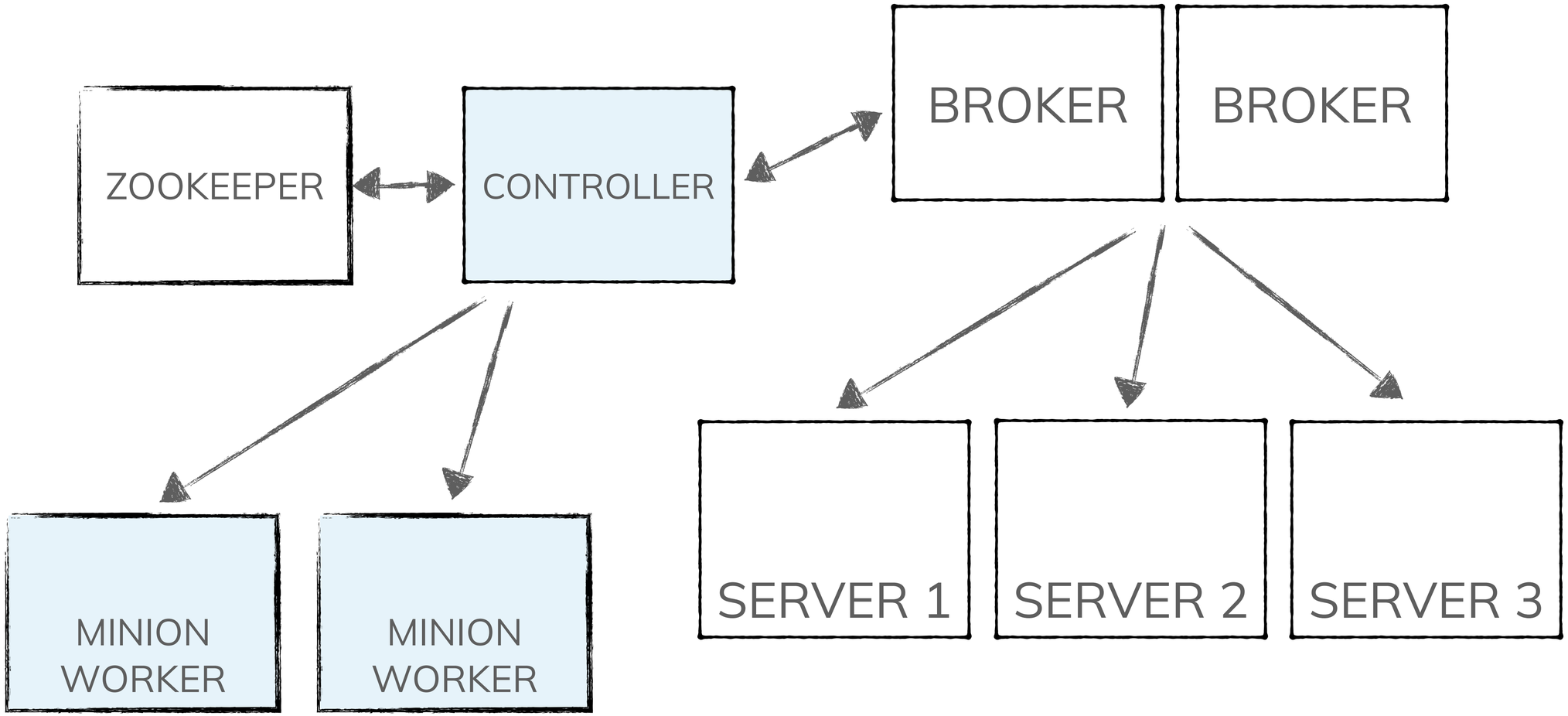


Brokers


SELECT *
FROM
SOMETHING
BROKER
(query needs segments 2 and 8)
1
3
4
6
7
8
9
10
11
SERVER 1
1
2
3
5
7
9
11
12
SERVER 2
2
4
5
6
8
10
12
SERVER 3

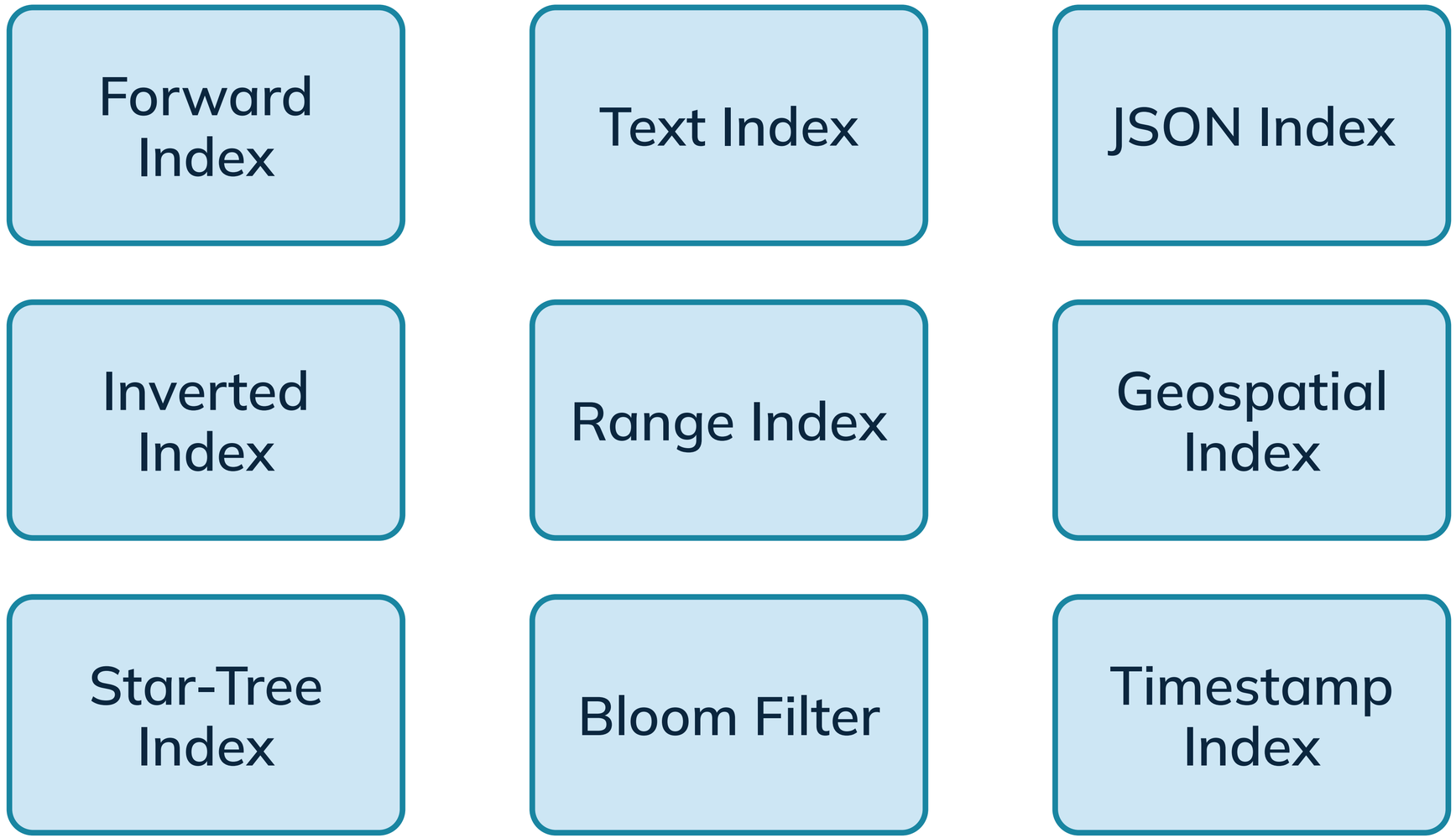

ZOOKEEPER
CONTROLLER
BROKER
BROKER
CLIENTS
1
3
4
6
7
8
9
10
11
SERVER 1
1
2
3
5
7
9
11
12
SERVER 2
2
4
5
6
8
10
12
SERVER 3

Minions


ZOOKEEPER
CONTROLLER
BROKER
BROKER
MINION WORKER
MINION WORKER
SERVER 1
SERVER 2
SERVER 3

indexes

Indexes

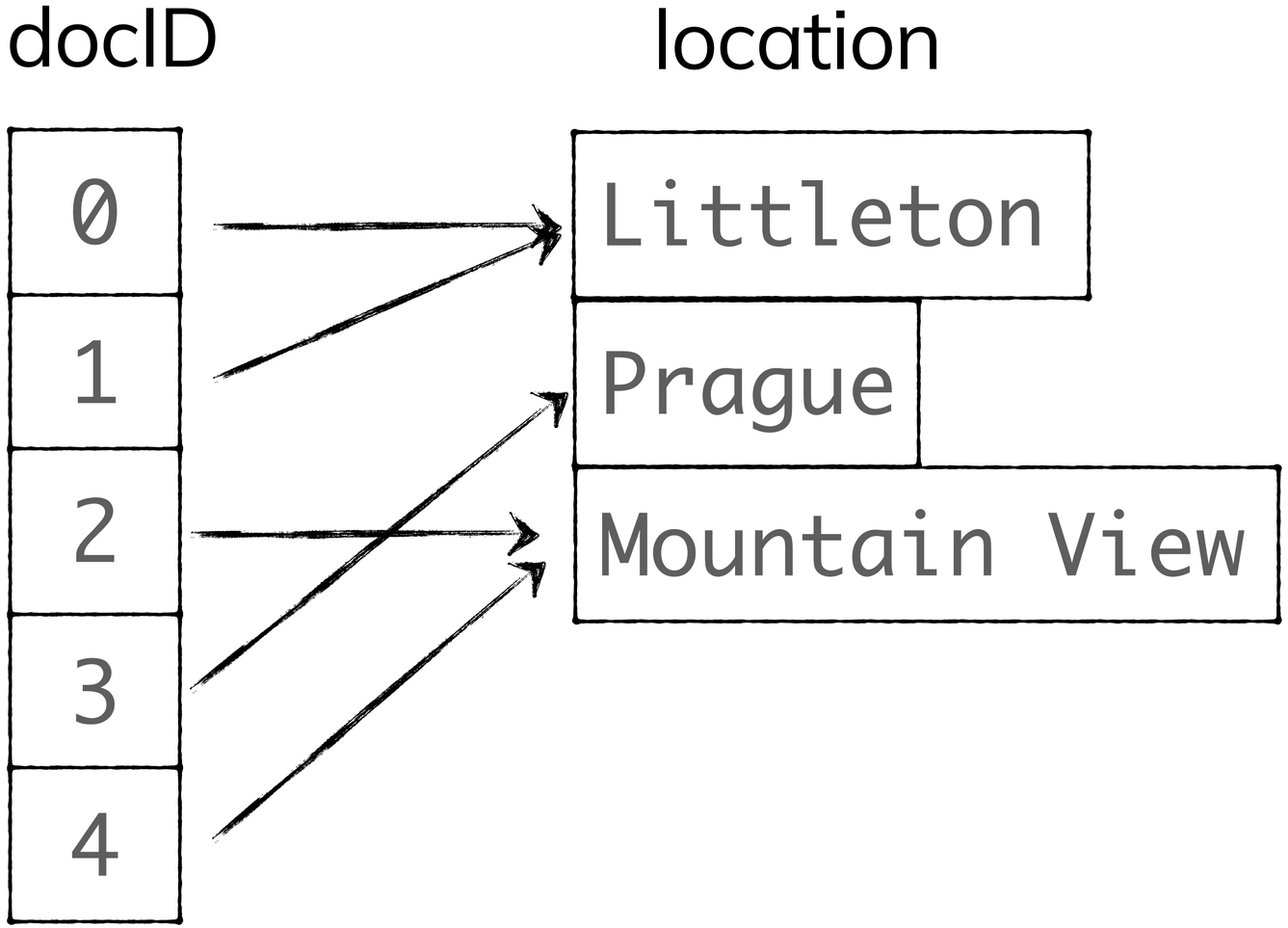

Forward Index
Text Index
JSON Index
Inverted Index
Range Index
Geospatial Index
Star-Tree Index
Bloom Filter
Timestamp Index

Forward Index

- A forward index lists values per document, like a table of contents
- “For a given docID, what is the value of this column?”
- For *dictionary encoded* columns, each unique value is assigned an ID, and the index stores those bit-compressed IDs
- For sorted columns, Pinot adds run-length encoding to the ID
- High-cardinality columns can be stored *raw*

Forward Index

```
docID, location
    0, Littleton
    1, Littleton
    2, Mountain View
    3, Prague
    4, Mountain View
```

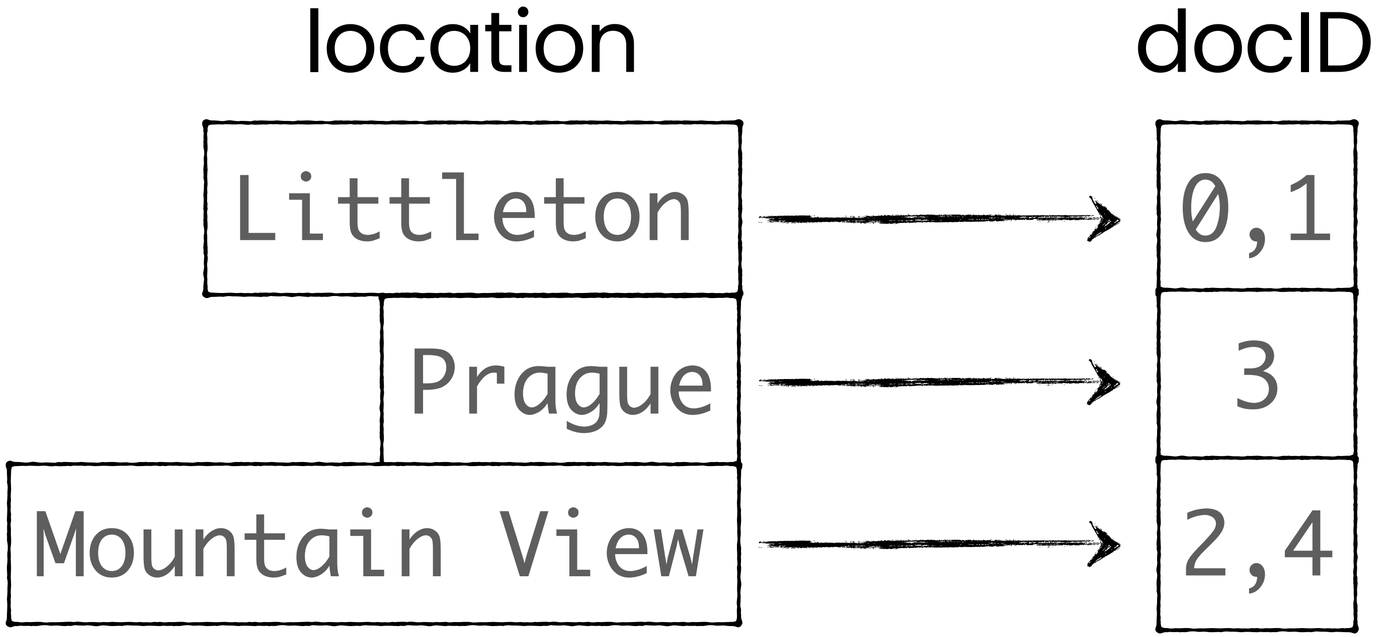


Inverted Index

- An inverted index lists documents per value, like a reference at the end of a book
- “What docIDs have this column value?”
- *Bitmap inverted indexes* maintain a map from each value to a bitmap of rows. Lookups are constant-time.
- *Sorted inverted indexes* (on sorted columns) map each value to start and end docIDs

Inverted Index

```
docID, location
    0, Littleton
    1, Littleton
    2, Mountain View
    3, Prague
    4, Mountain View
```



Range Index

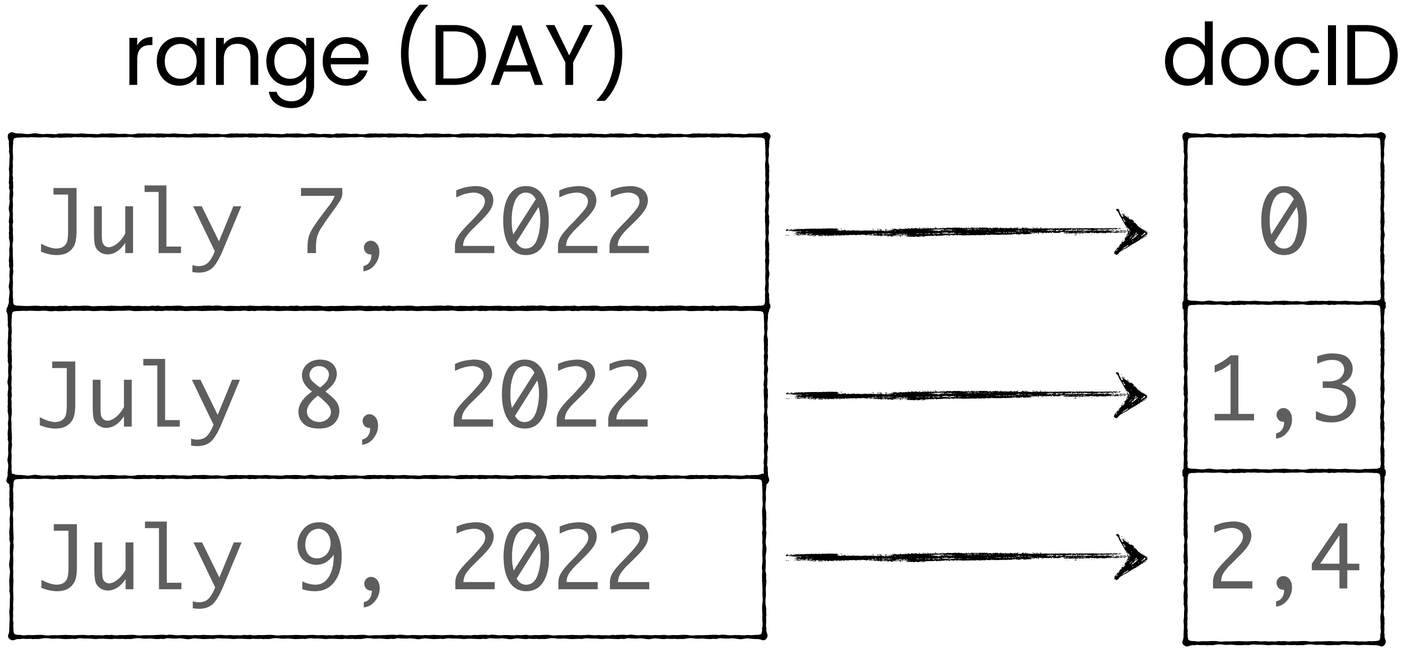
- The range index maps value ranges to docIDs
- Based on the inverted index, but applied to numerical (usually metric) columns
- Numerical columns are typically very high cardinality, and thus not suited to inverted indexes
- Breaking the “continuous” numerical values into discrete ranges reduces cardinality, and makes the inverted index suitable

Timestamp Index

Applies the functionality of a range index to timestamp columns with time grain configurable at index definition time.

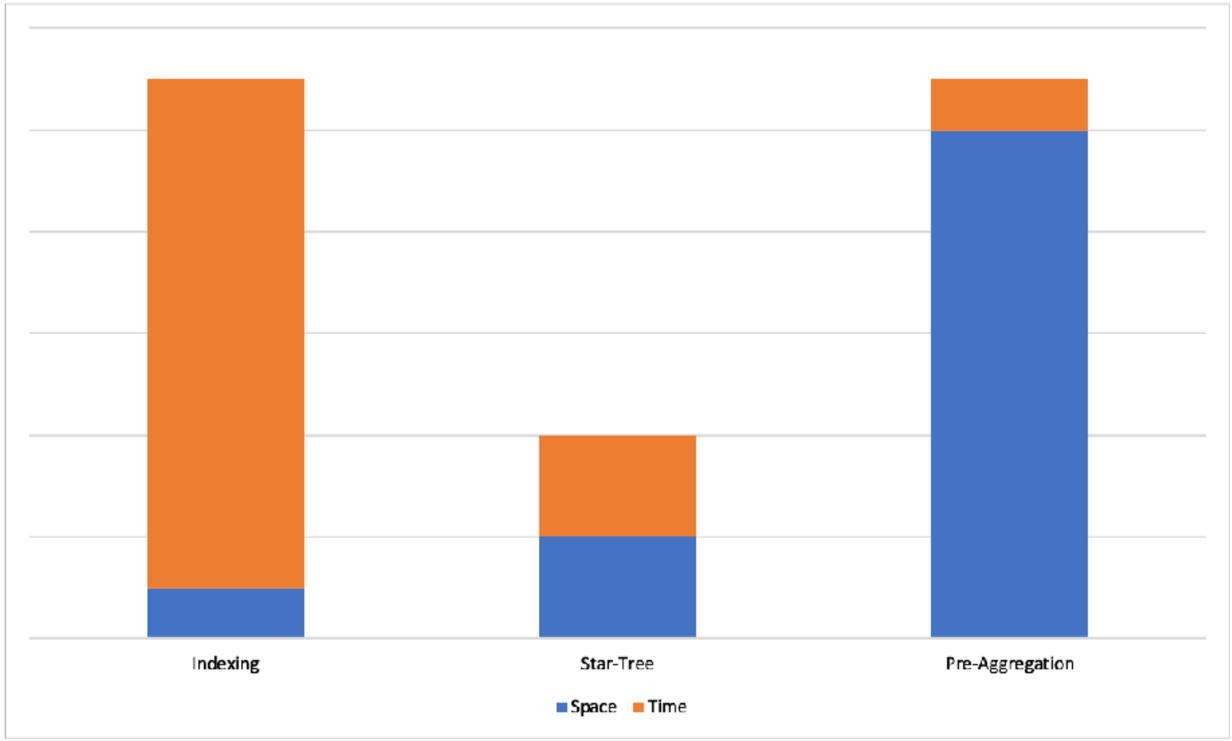
Timestamp Index

```
docID, timestamp
    0, 20220707T1449
    1, 20220708T0543
    2, 20220709T0128
    3, 20220708T2150
    4, 20220709T1507
```



Star-Tree index

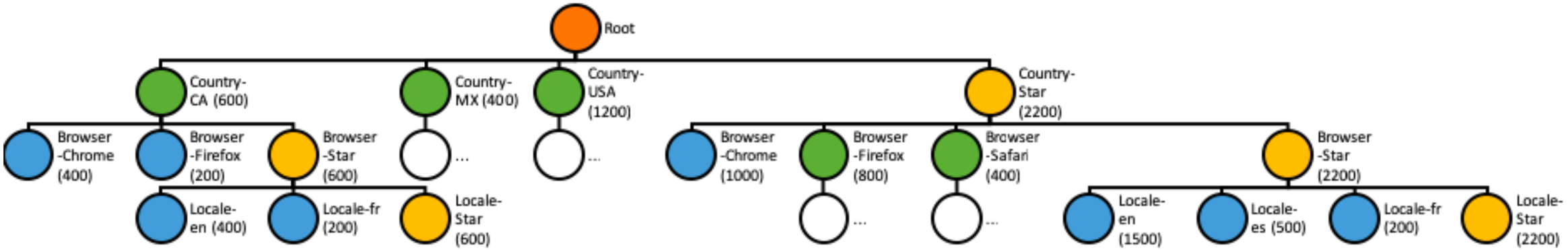
- The star-tree index is uses pre-aggregated results to reduced the number of rows to be scanned
- An optimization for the core filter-and-aggregate query pattern
- Suitable for aggregations on a single metric when several dimensions may appear in the WHERE clause
- The Star Tree index is like writing a pivot table to disk



Star-Tree index

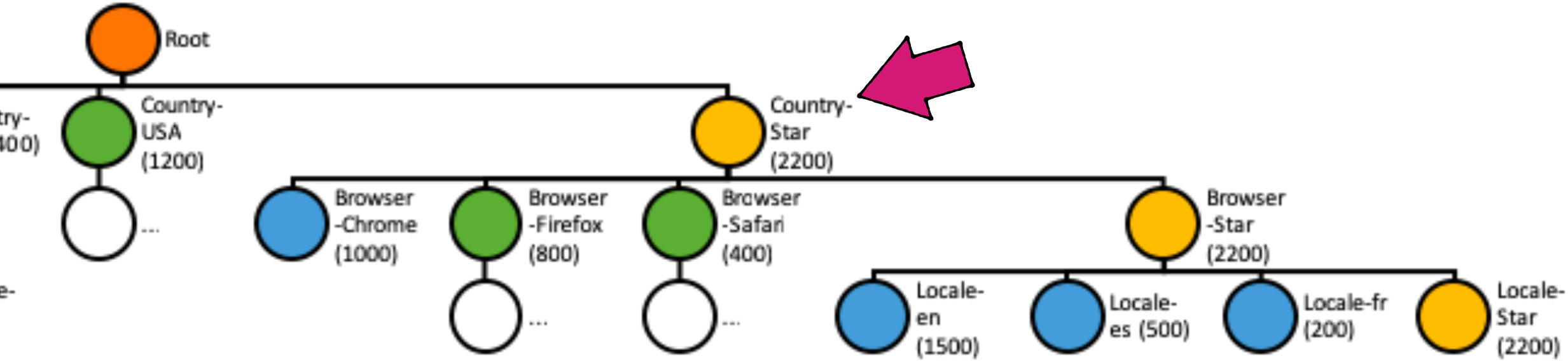
Country	Browser	Locale	Impressions
CA	Chrome	en	400
CA	Firefox	fr	200
MX	Safari	es	300
MX	Safari	en	100
USA	Chrome	en	600
USA	Firefox	es	200
USA	Firefox	en	400

Star-Tree index



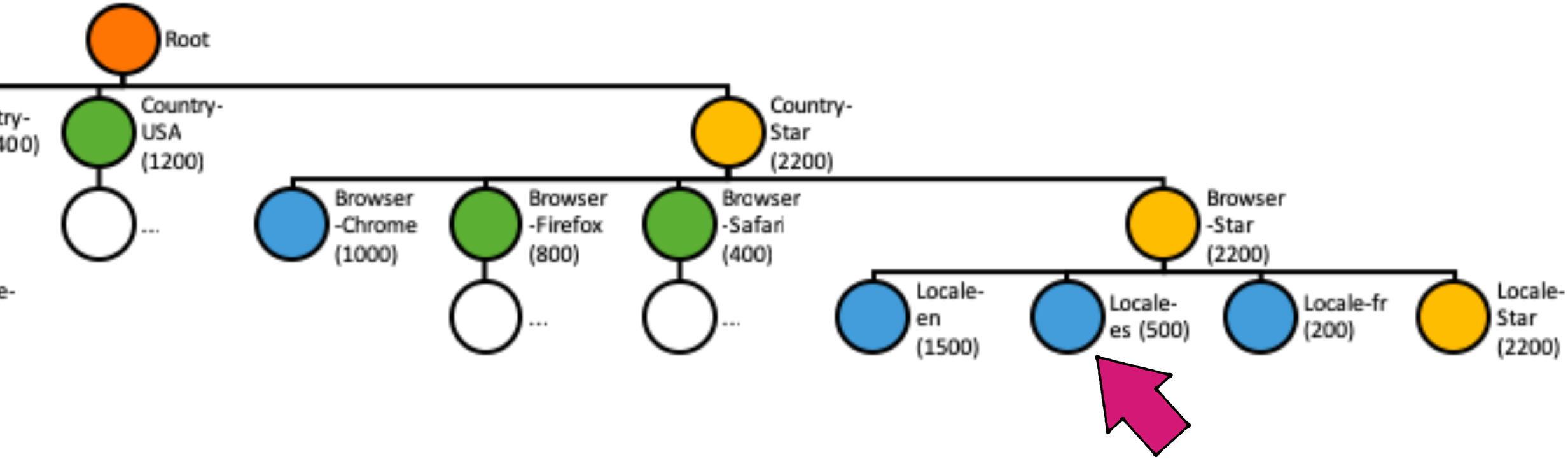
Star-Tree index

Star-Tree index



Star-Tree index

Star-Tree index

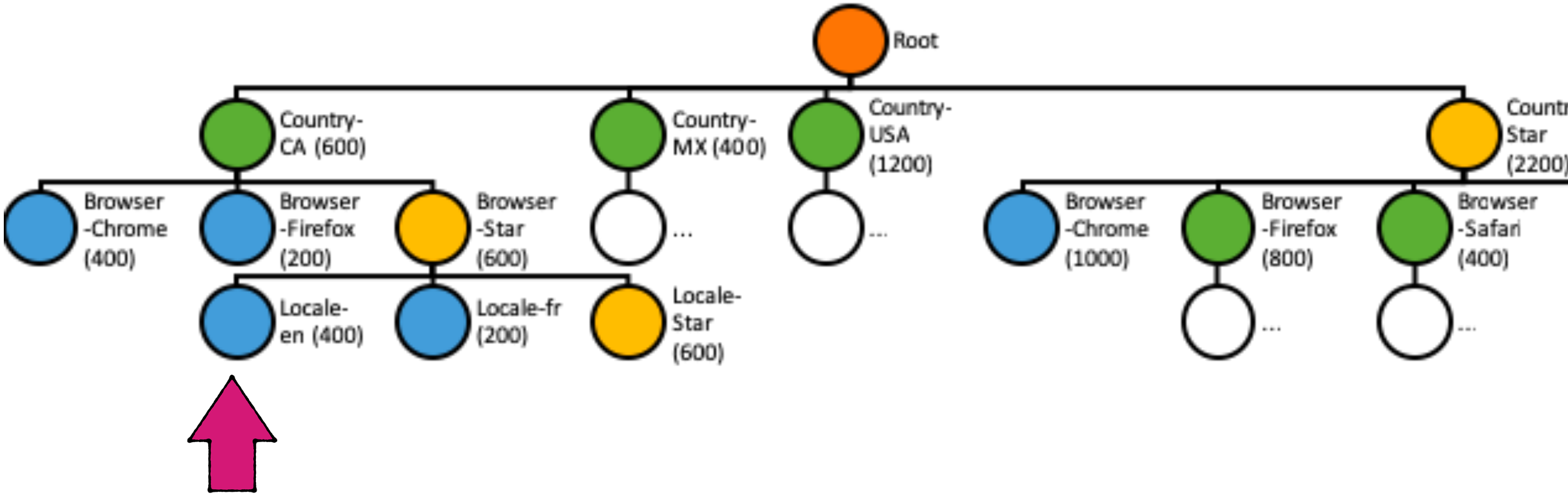


Star-Tree index

Star-Tree index

Star-Tree index

Star-Tree index



Star-Tree index

Country	Browser	Locale	SUM__Impressions
CA	Chrome	en	400
CA	Firefox	fr	200
MX	Safari	en	100
MX	Safari	es	300
USA	Chrome	en	600
USA	Firefox	en	400
USA	Firefox	es	200
CA	*	en	400
CA	*	fr	200
CA	*	*	600
MX	Safari	*	400
USA	Firefox	*	600
USA	*	en	1000
USA	*	es	200

Star-Tree index

USA	Chrome	en	600
USA	Firefox	en	400
USA	Firefox	es	200
CA	*	en	400
CA	*	fr	200
CA	*	*	600
MX	Safari	*	400
USA	Firefox	*	600
USA	*	en	1000
USA	*	es	200
USA	*	*	1200
*	Chrome	en	1000
*	Firefox	en	400
*	Firefox	es	200
*	Firefox	fr	200

ingest

Batch Ingestion

```
studentID,firstName,lastName,gender,subject,score,timestampInEpoch
200,Lucy,Smith,Female,Maths,3.8,1570863600000
200,Lucy,Smith,Female,English,3.5,1571036400000
201,Bob,King,Male,Maths,3.2,1571900400000
202,Nick,Young,Male,Physics,3.6,1572418800000
```

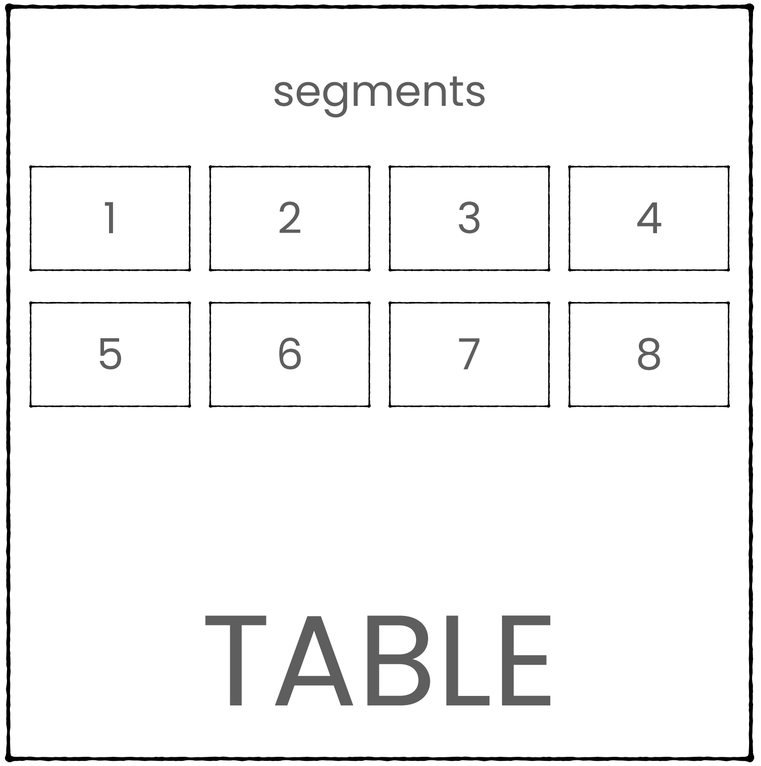
Batch Ingestion

```
studentID,
200,
201,
202,
203,
```

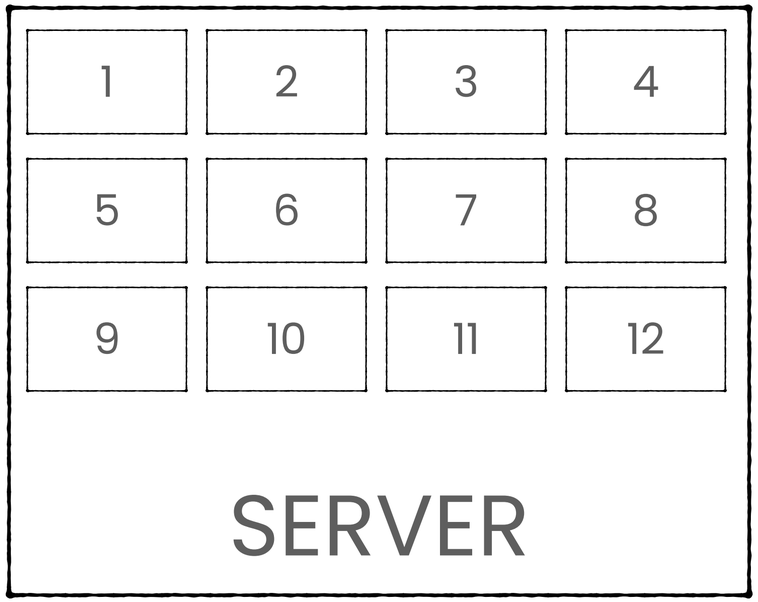
TABLE

Batch Ingestion

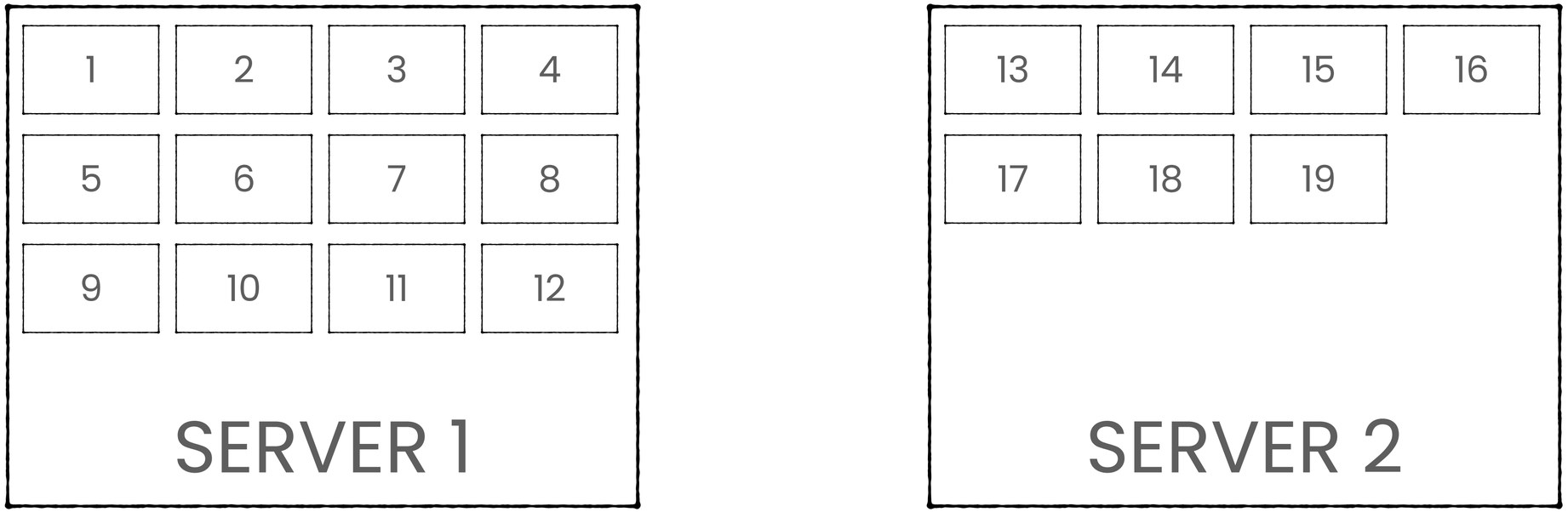
```
studentID,
200,
201,
202,
203,
```



Batch Ingestion



Batch Ingestion



Batch Ingestion: the data

```
studentID,firstName,lastName,gender,subject,score,timestampInEpoch
200,Lucy,Smith,Female,Maths,3.8,1570863600000
200,Lucy,Smith,Female,English,3.5,1571036400000
201,Bob,King,Male,Maths,3.2,1571900400000
202,Nick,Young,Male,Physics,3.6,1572418800000
```

Batch Ingestion: schema

```
{
  "schemaName": "transcript",
  "dimensionFieldSpecs": [
    {
      "name": "studentID",
      "dataType": "INT"
    },
    {
      "name": "firstName",
      "dataType": "STRING"
    },
    {
      "name": "lastName",
      "dataType": "STRING"
    },
    {
      "name": "gender",
      "dataType": "STRING"
```

Batch Ingestion: schema

```
      "dataType": "STRING"
    },
    {
      "name": "gender",
      "dataType": "STRING"
    },
    {
      "name": "subject",
      "dataType": "STRING"
    }
  ],
  "metricFieldSpecs": [
    {
      "name": "score",
      "dataType": "FLOAT"
    }
  ],
  "dateTimeFieldSpecs": [{
    "name": "timestampInEpoch",
```

Batch Ingestion: schema

```
    }
  ],
  "metricFieldSpecs": [
    {
      "name": "score",
      "dataType": "FLOAT"
    }
  ],
  "dateTimeFieldSpecs": [{
    "name": "timestampInEpoch",
    "dataType": "LONG",
    "format" :
"1:MILLISECONDS:EPOCH",
    "granularity": "1:MILLISECONDS"
  }]
}
```

Batch Ingestion: table config

```
{
  "tableName": "transcript",
  "tableType": "OFFLINE",
  "segmentsConfig": {
    "replication": 1,
    "timeColumnName": "timestampInEpoch",
    "timeType": "MILLISECONDS",
    "retentionTimeUnit": "DAYS",
    "retentionTimeValue": 365
  },
  "tenants": {
    "broker":"DefaultTenant",
    "server":"DefaultTenant"
  },
  "tableIndexConfig": {
    "loadMode": "MMAP"
  },
```

Batch Ingestion: table config

```
    "tableIndexConfig": {
      "loadMode": "MMAP"
    },
    "ingestionConfig": {
      "batchIngestionConfig": {
        "segmentIngestionType": "APPEND",
        "segmentIngestionFrequency": "DAILY"
      },
      "continueOnError": true,
      "rowTimeValueCheck": true,
      "segmentTimeValueCheck": false

    },
    "metadata": {}
}
```

Batch Ingestion: create table

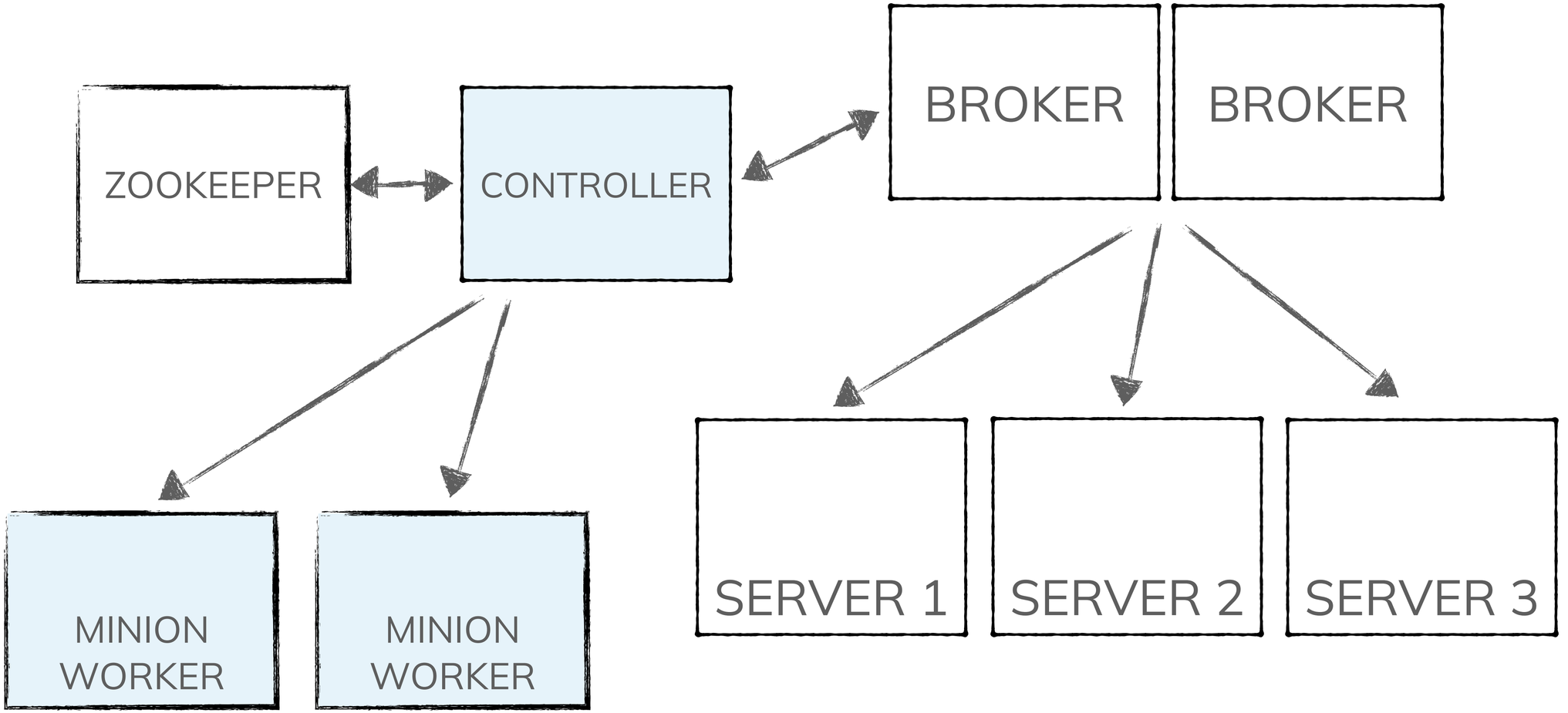
```
bin/pinot-admin.sh AddTable \\
  -tableConfigFile /path/to/table-config.json \\
  -schemaFile /path/to/table-schema.json -exec
```

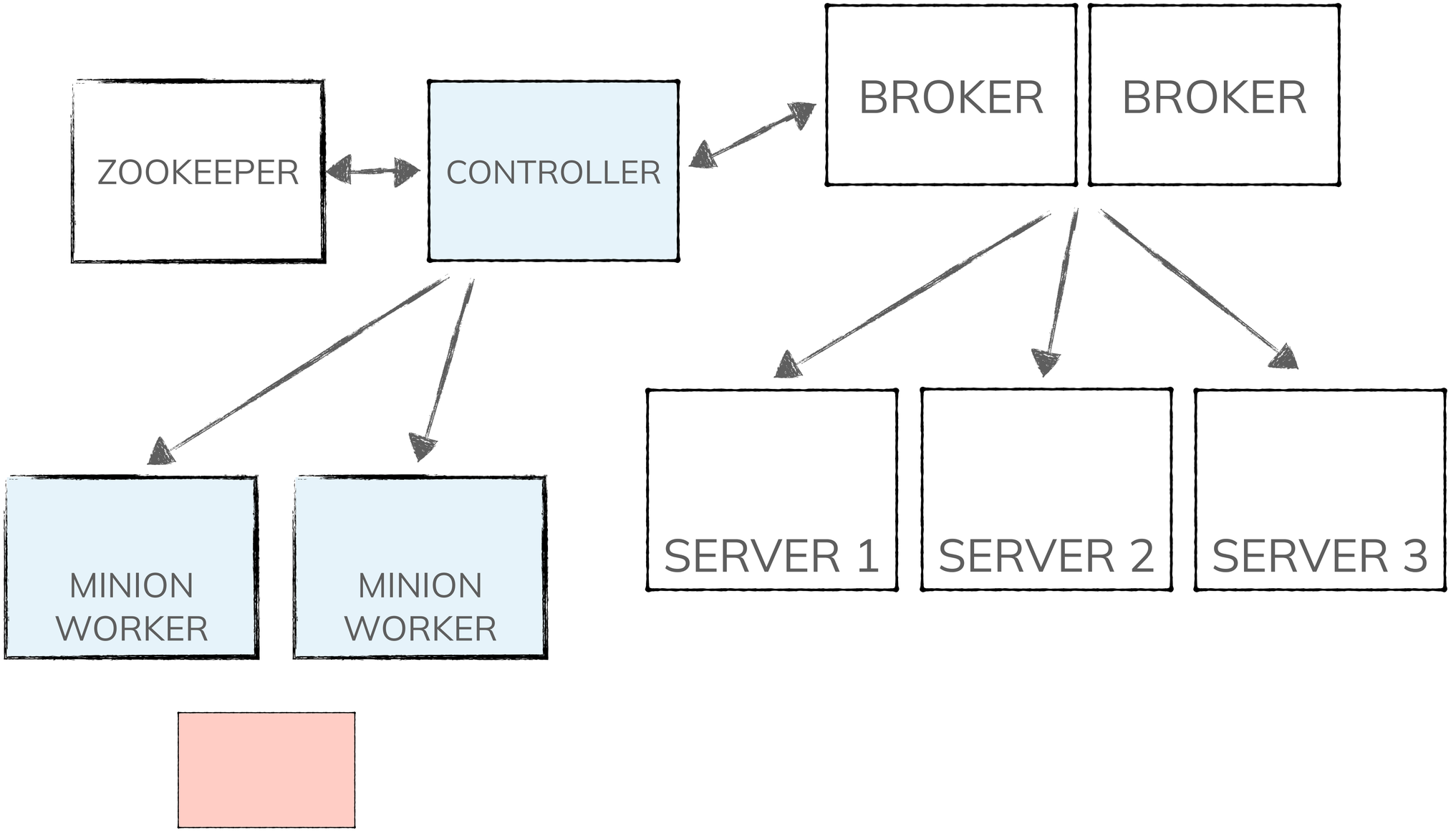
Batch Ingestion: job spec

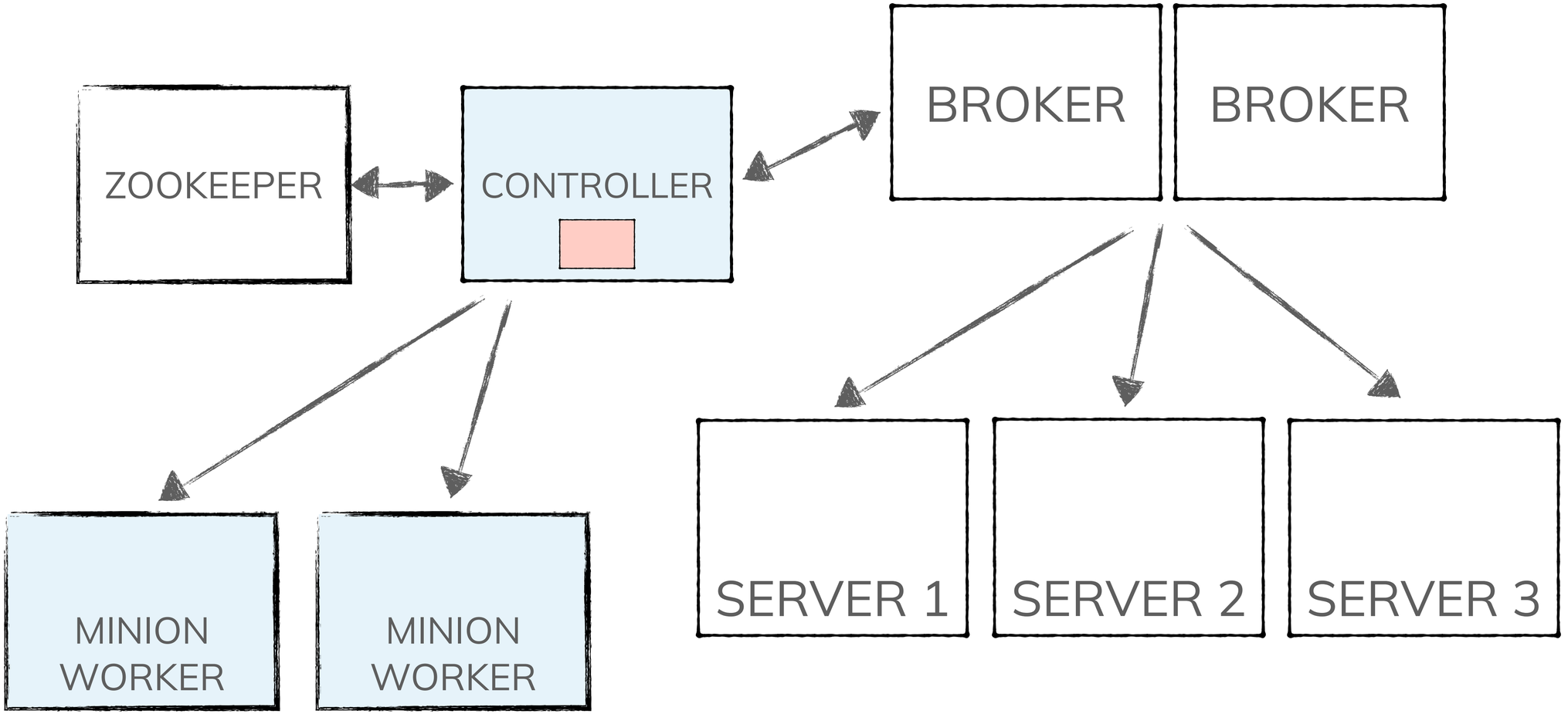
```
executionFrameworkSpec:
  name: 'standalone'
  segmentGenerationJobRunnerClassName:
'org.apache.pinot.plugin.ingestion.batch.standalone.SegmentGenerationJobRunner'
  segmentTarPushJobRunnerClassName:
'org.apache.pinot.plugin.ingestion.batch.standalone.SegmentTarPushJobRunner'
jobType: SegmentCreationAndTarPush
inputDirURI: '/data'
includeFileNamePattern: 'glob:**/import.jsonl'
outputDirURI: '/opt/pinot/data/movies/'
overwriteOutput: true
pinotFSSpecs:
  - scheme: file
    className: org.apache.pinot.spi.filesystem.LocalPinotFS
recordReaderSpec:
  dataFormat: 'json'
  className: 'org.apache.pinot.plugin.inputformat.json.JSONRecordReader'
tableSpec:
  tableName: 'movies'
pinotClusterSpecs:
  - controllerURI: 'http://pinot-controller-json:9000'
pushJobSpec:
  pushParallelism: 2
  pushAttempts: 2
```

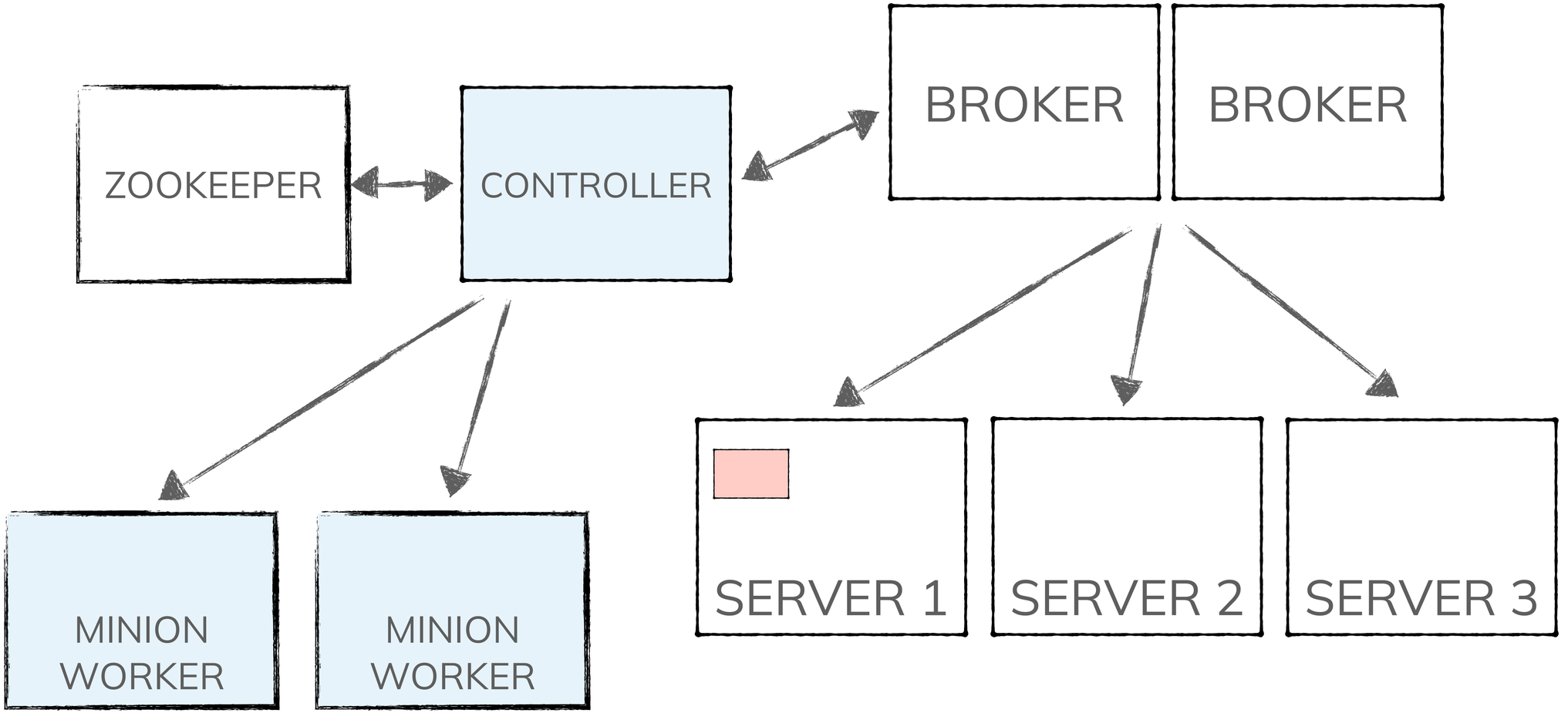
Batch Ingestion: ingest data

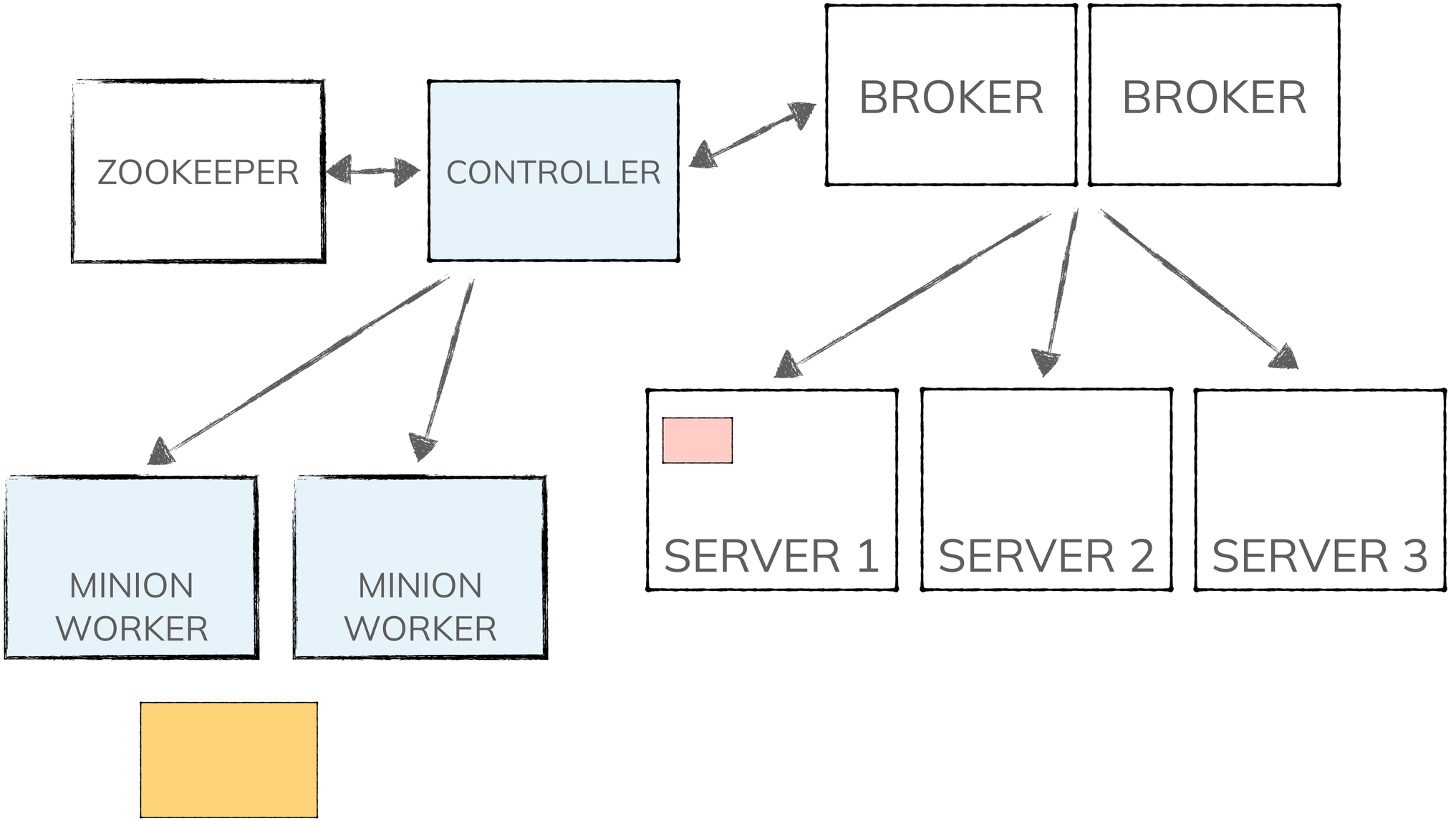
```
bin/pinot-admin.sh LaunchDataIngestionJob \
    -jobSpecFile job-spec.yml
```

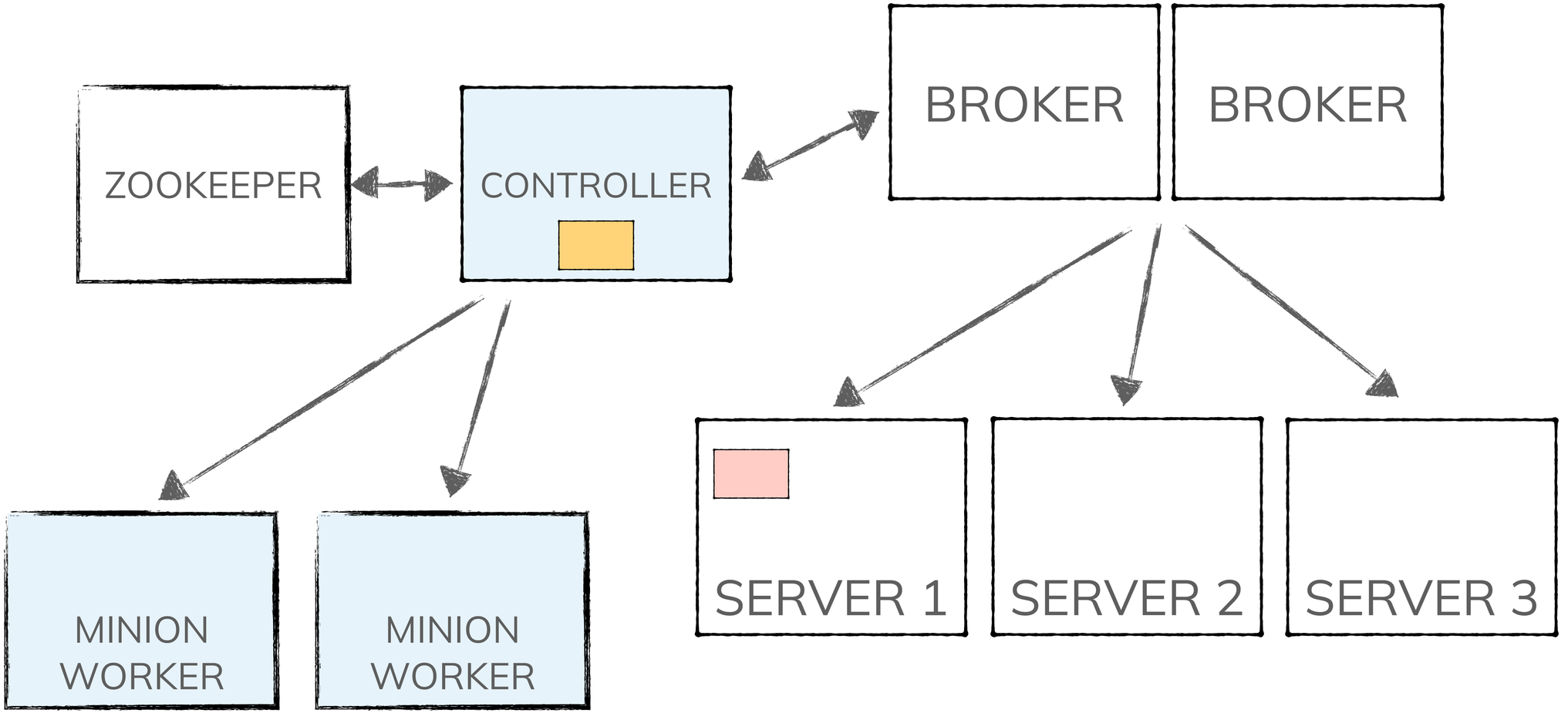

ZOOKEEPER
CONTROLLER
BROKER
BROKER
MINION WORKER
MINION WORKER
SERVER 1
SERVER 2
SERVER 3

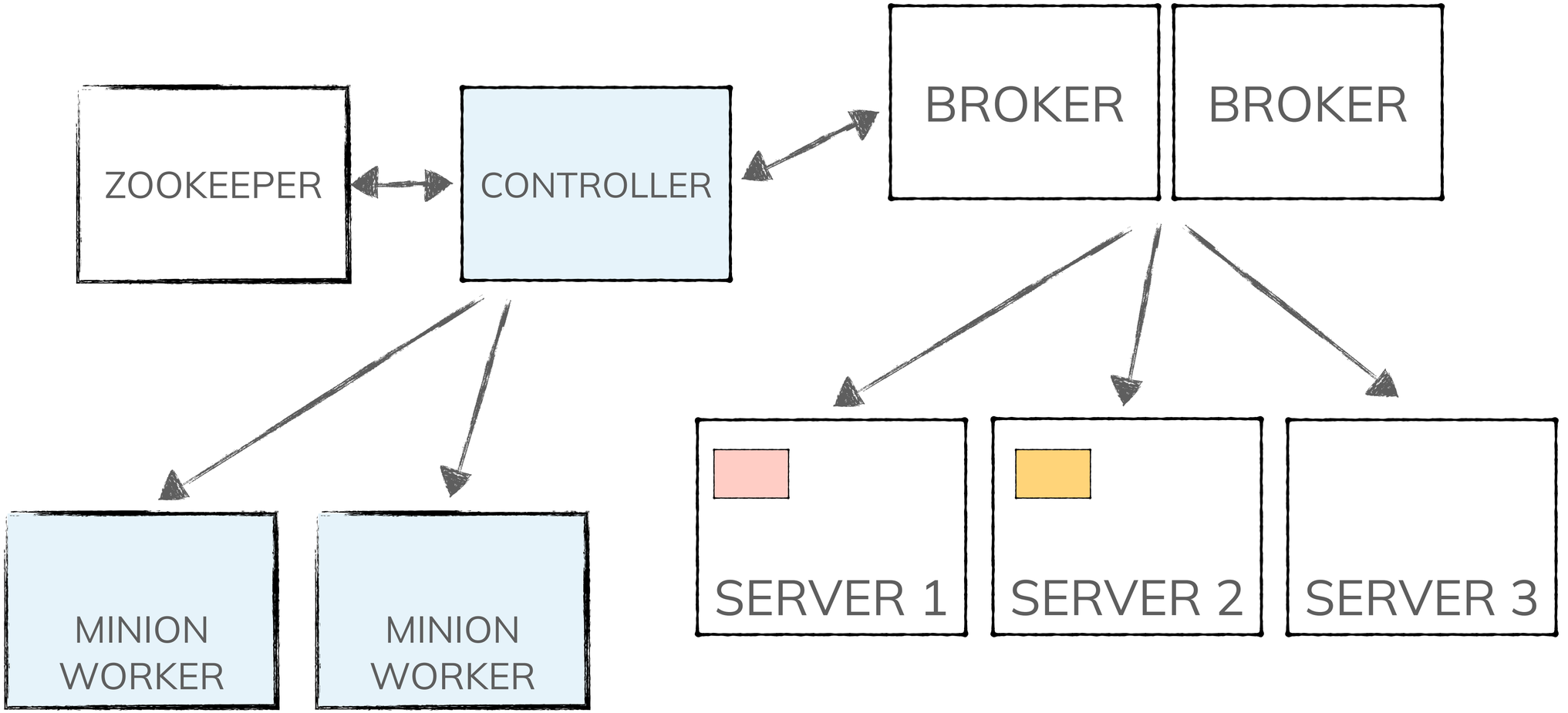

ZOOKEEPER
CONTROLLER
BROKER
BROKER
MINION WORKER
MINION WORKER
SERVER 1
SERVER 2
SERVER 3

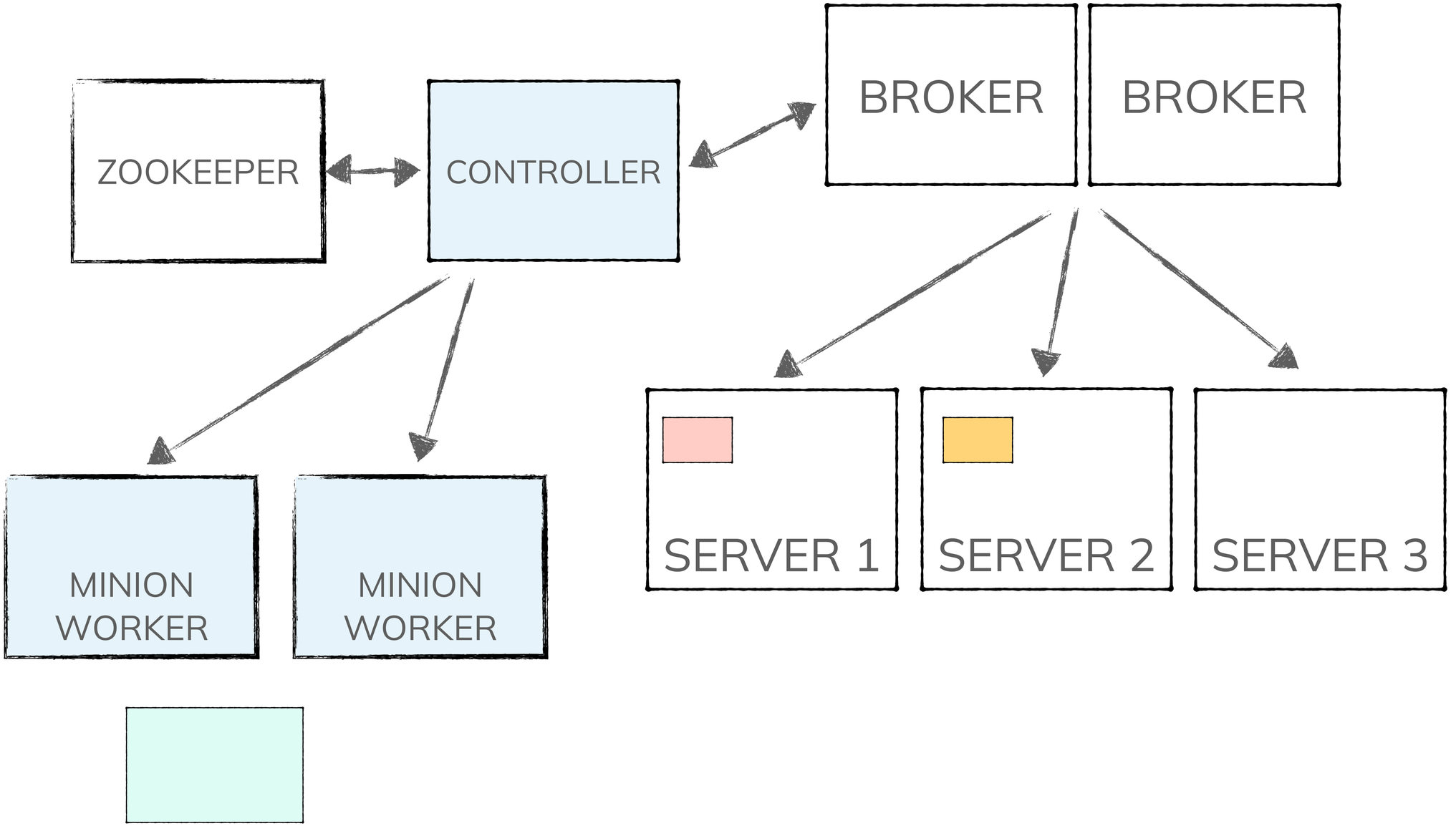

ZOOKEEPER
CONTROLLER
BROKER
BROKER
MINION WORKER
MINION WORKER
SERVER 1
SERVER 2
SERVER 3

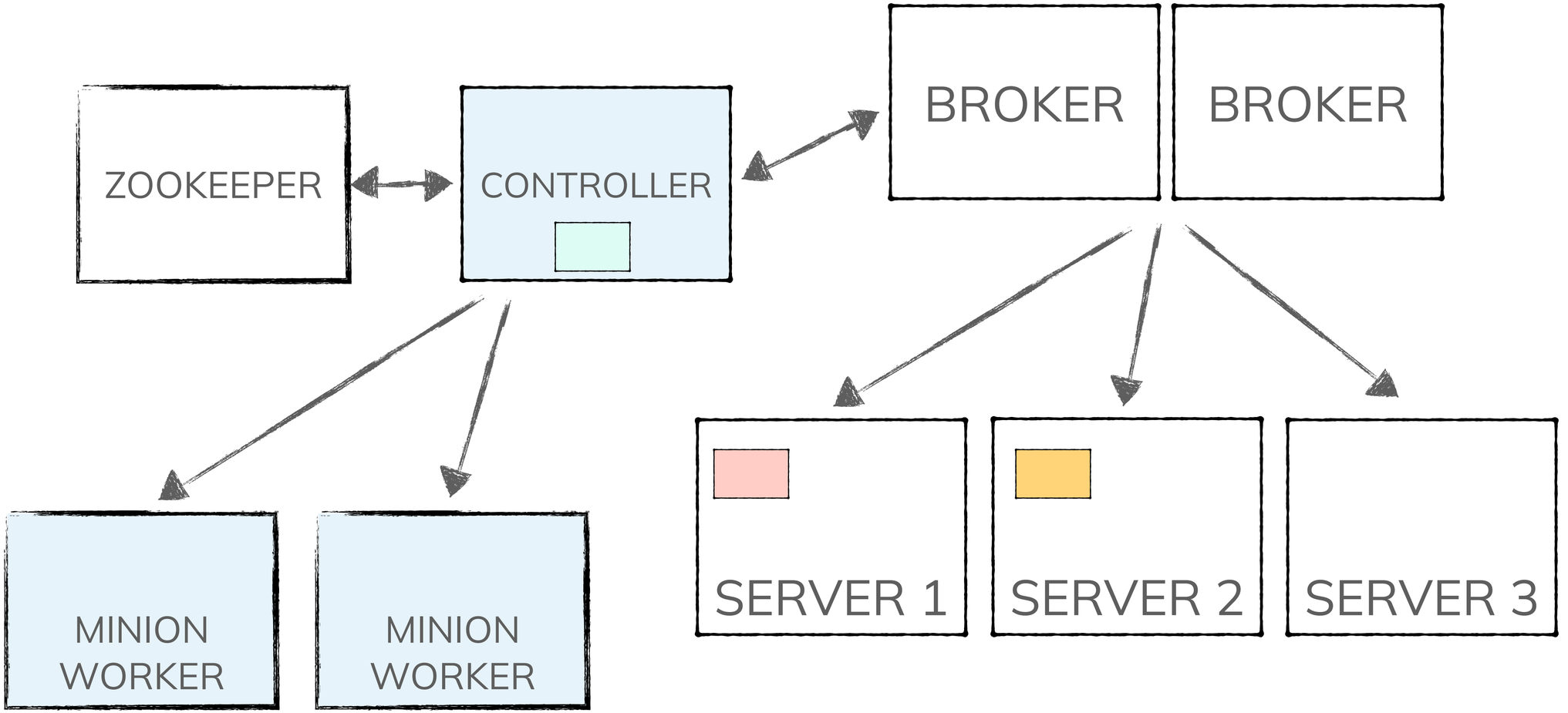

ZOOKEEPER
CONTROLLER
BROKER
BROKER
MINION WORKER
MINION WORKER
SERVER 1
SERVER 2
SERVER 3

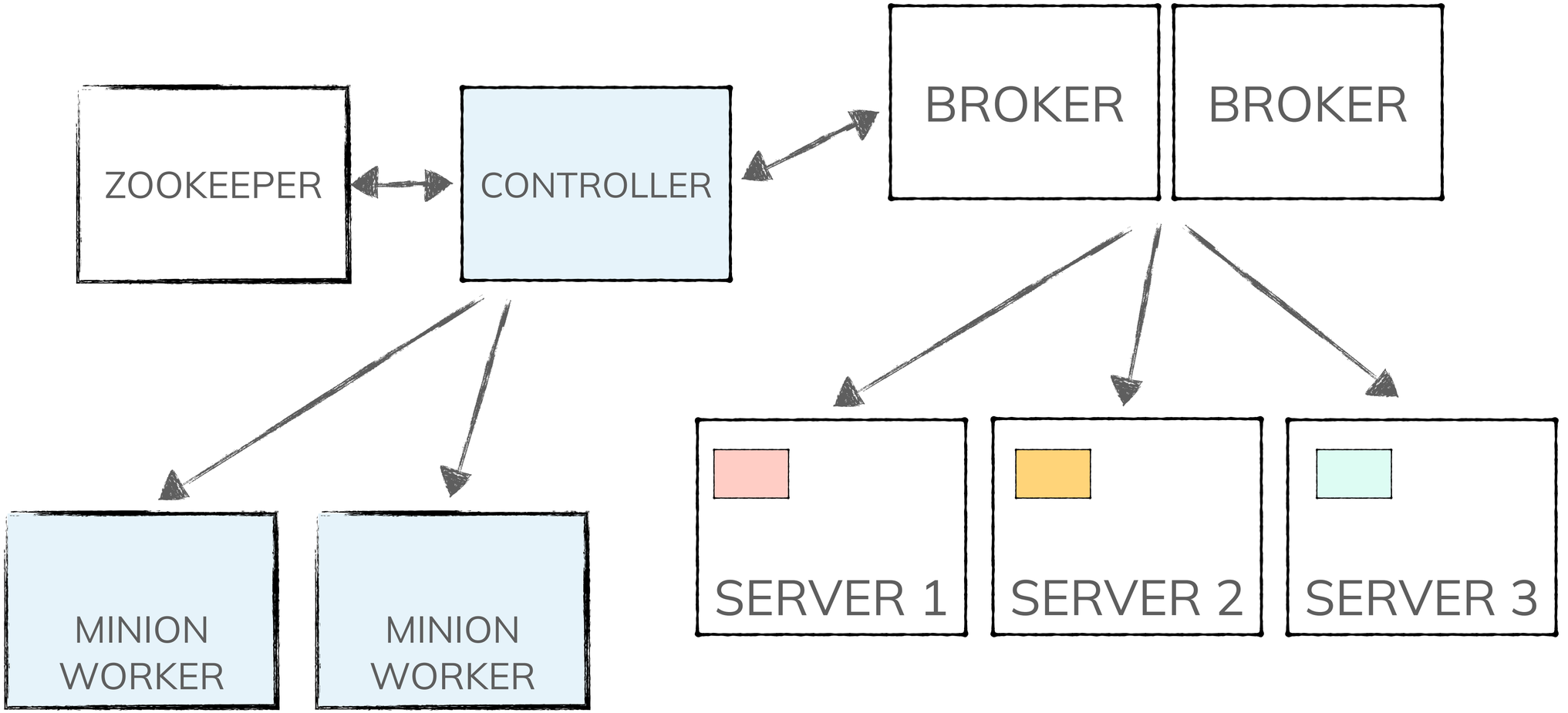

ZOOKEEPER
CONTROLLER
BROKER
BROKER
MINION WORKER
MINION WORKER
SERVER 1
SERVER 2
SERVER 3


ZOOKEEPER
CONTROLLER
BROKER
BROKER
MINION WORKER
MINION WORKER
SERVER 1
SERVER 2
SERVER 3

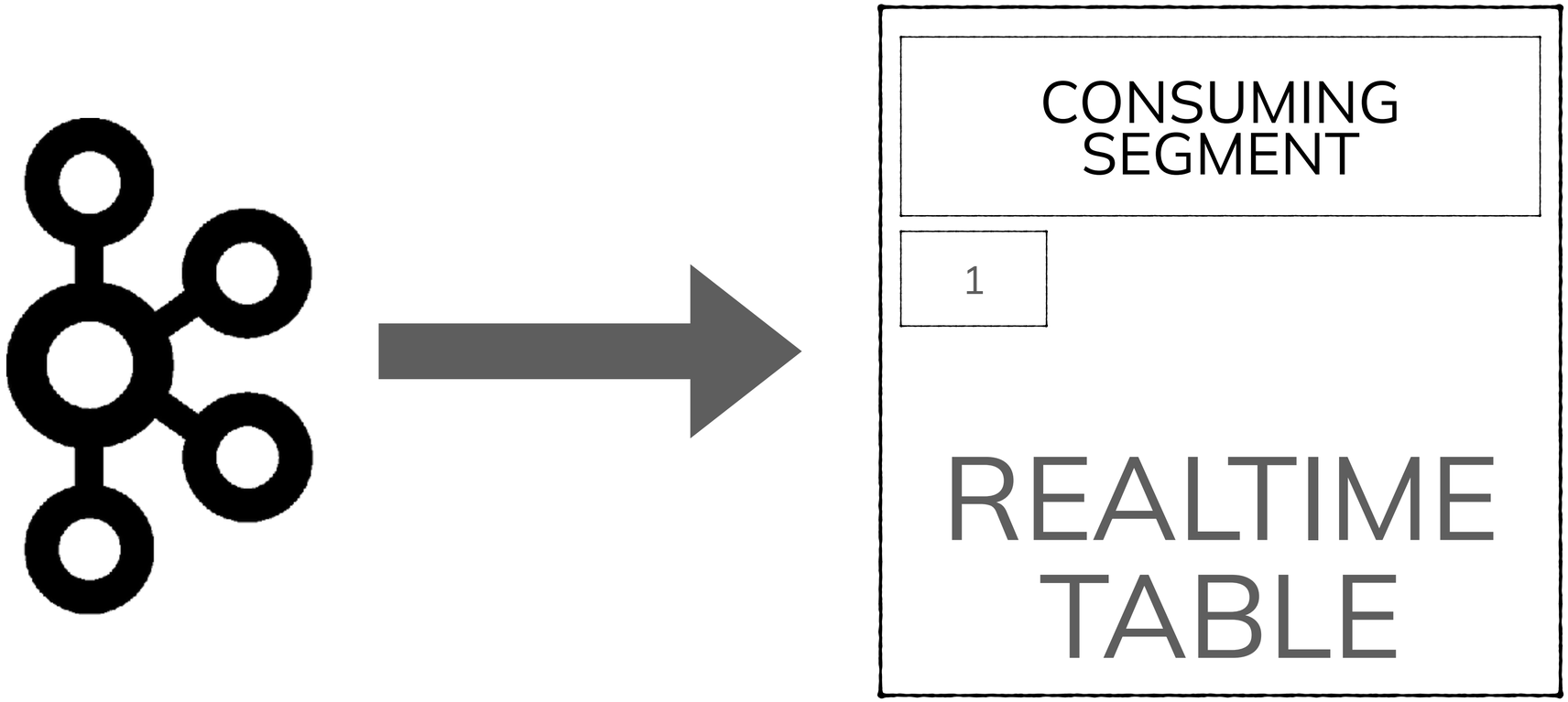

ZOOKEEPER
CONTROLLER
BROKER
BROKER
MINION WORKER
MINION WORKER
SERVER 1
SERVER 2
SERVER 3


ZOOKEEPER
CONTROLLER
BROKER
BROKER
MINION WORKER
MINION WORKER
SERVER 1
SERVER 2
SERVER 3

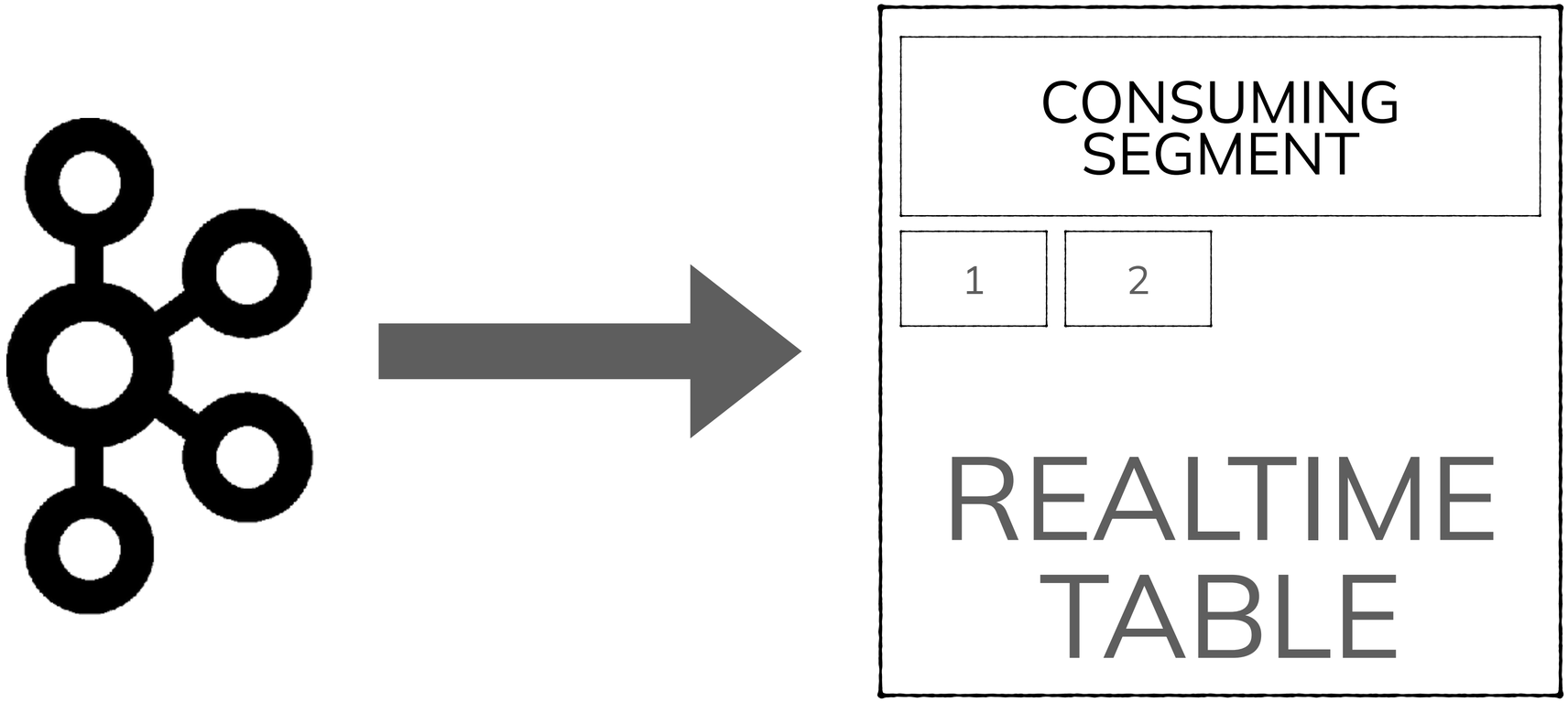

ZOOKEEPER
CONTROLLER
BROKER
BROKER
MINION WORKER
MINION WORKER
SERVER 1
SERVER 2
SERVER 3


ZOOKEEPER
CONTROLLER
BROKER
BROKER
MINION WORKER
MINION WORKER
SERVER 1
SERVER 2
SERVER 3

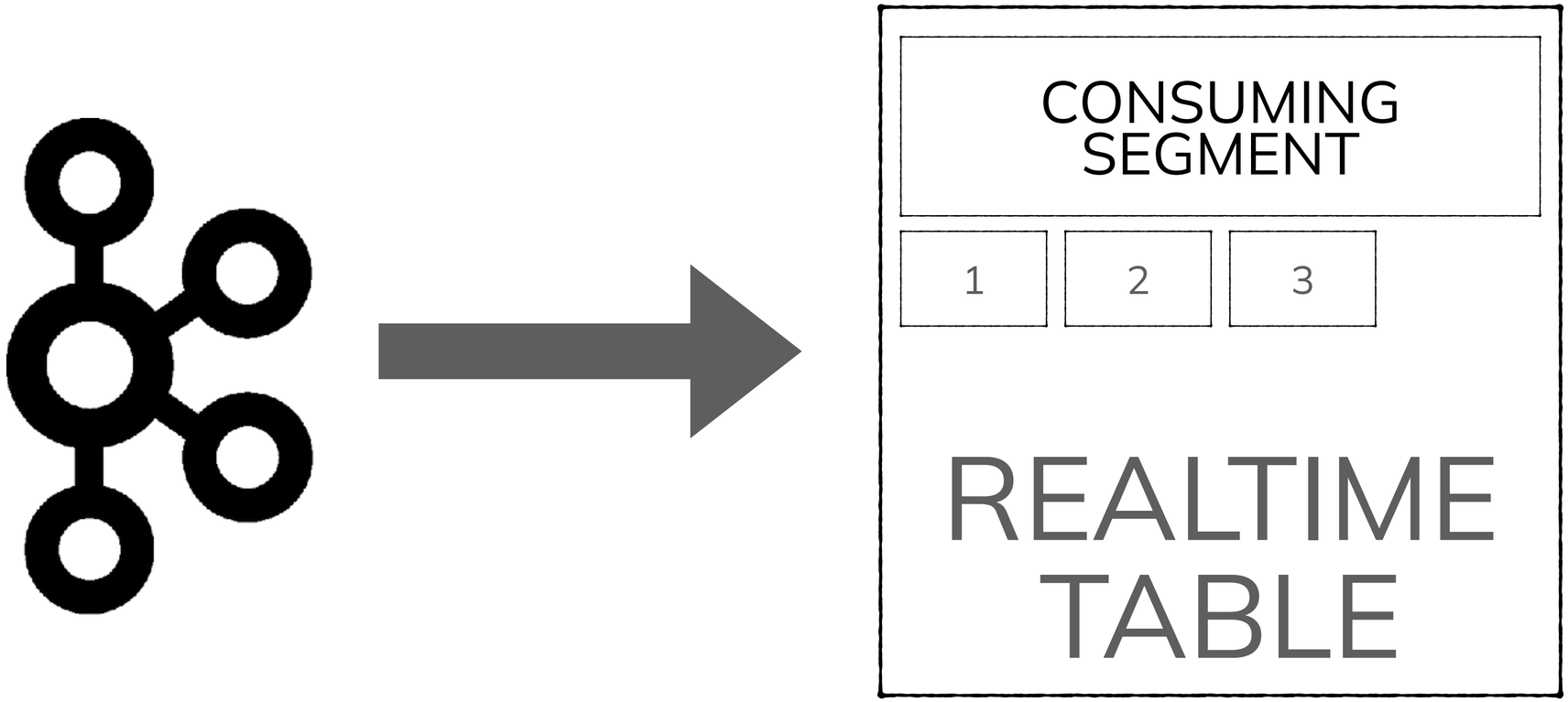
Realtime ingestion



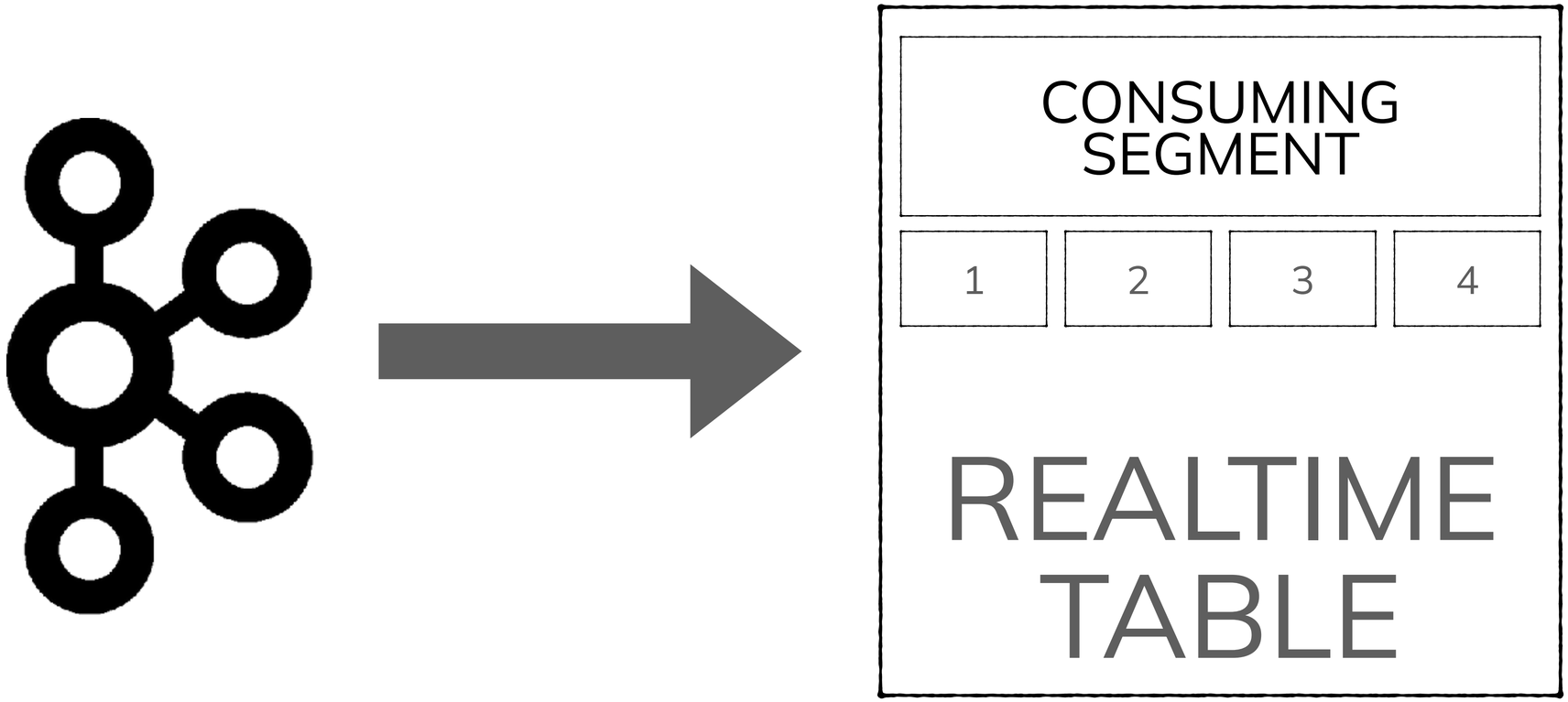
Realtime ingestion



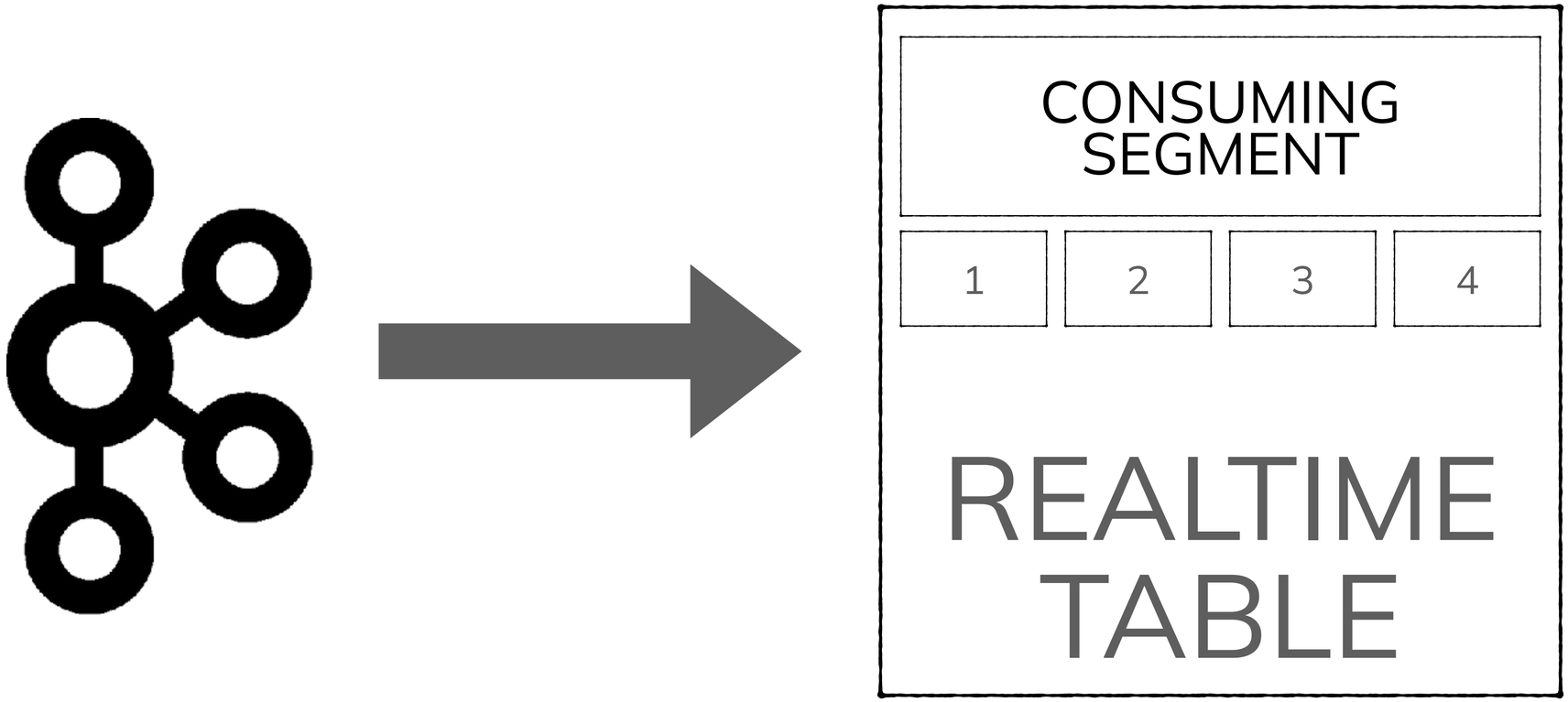
Realtime ingestion



Realtime ingestion



Realtime ingestion



Realtime ingestion: data

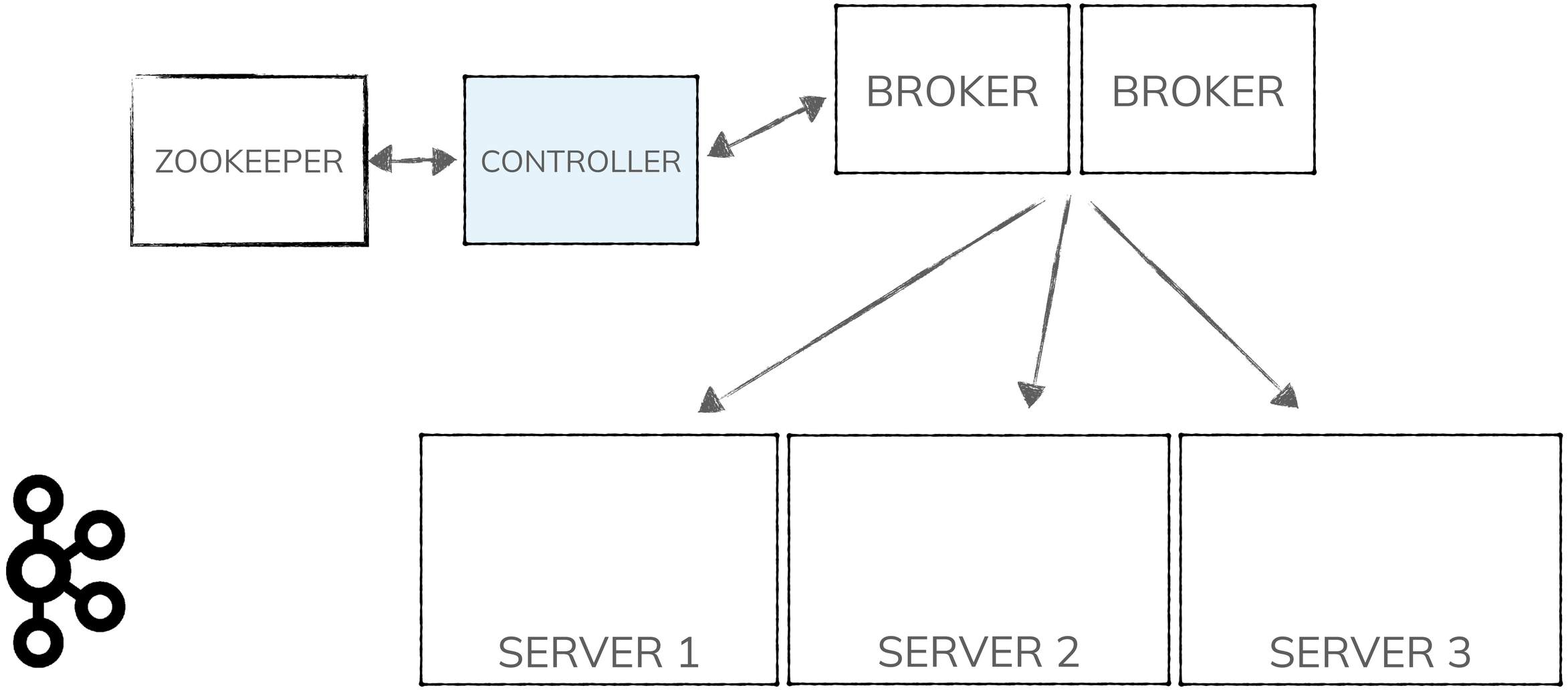

```
{
    "studentID": 205,
    "firstName": "Natalie",
    "lastName": "Jones",
    "gender": "Female",
    "subject": "Maths",
    "score": 3.8,
    "timestamp": 1571900400000
},
{
    "studentID": 207,
    "firstName": "Bob",
    "lastName": "Lewis",
    "gender": "Male",
    "subject": "Maths",
    "score": 3.2,
    "timestamp": 1571900400000
},
{
    "studentID": 209,
    "firstName": "Jane",
    "lastName": "Doe",
```

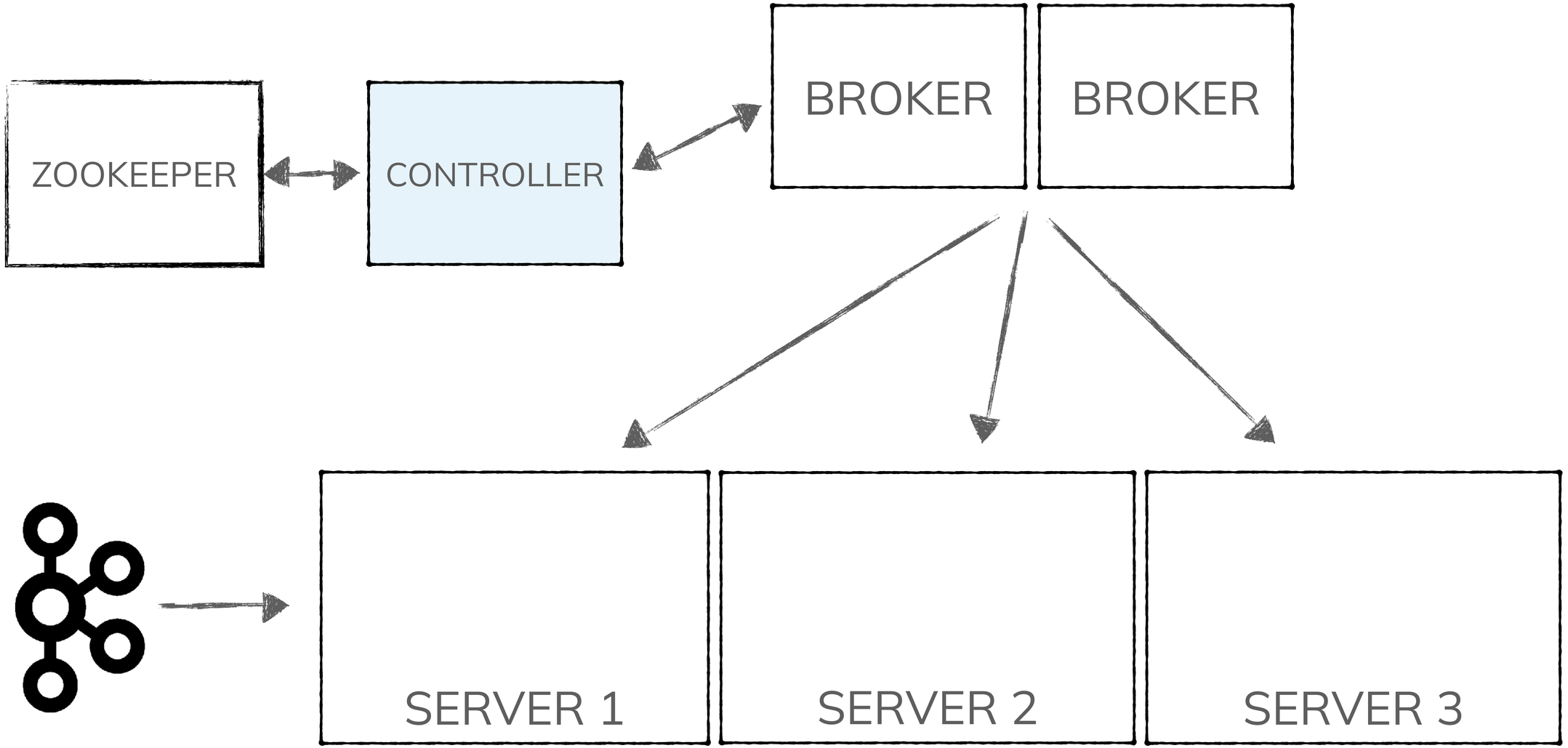
Realtime ingestion: table configuration

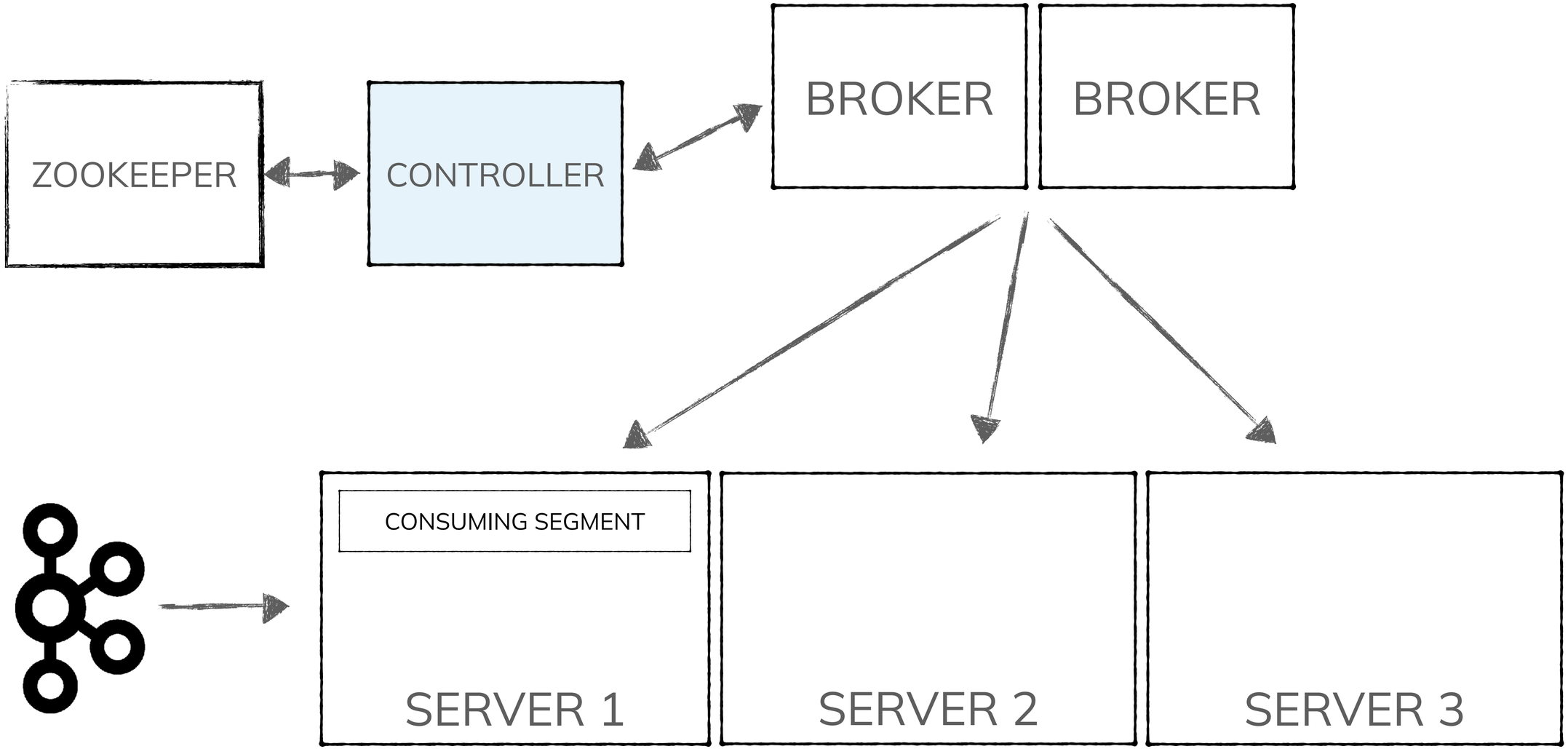
```
{
 "tableName": "transcript",
 "tableType": "REALTIME",
 "segmentsConfig": {
   "timeColumnName": "timestamp",
   "timeType": "MILLISECONDS",
   "schemaName": "transcript",
   "replicasPerPartition": "1"
 },
 "tenants": {},
 "tableIndexConfig": {
   "loadMode": "MMAP",
   "streamConfigs": {
     "streamType": "kafka",
     "stream.kafka.consumer.type": "lowlevel",
     "stream.kafka.topic.name": "transcript-topic",
     "stream.kafka.decoder.class.name":
```

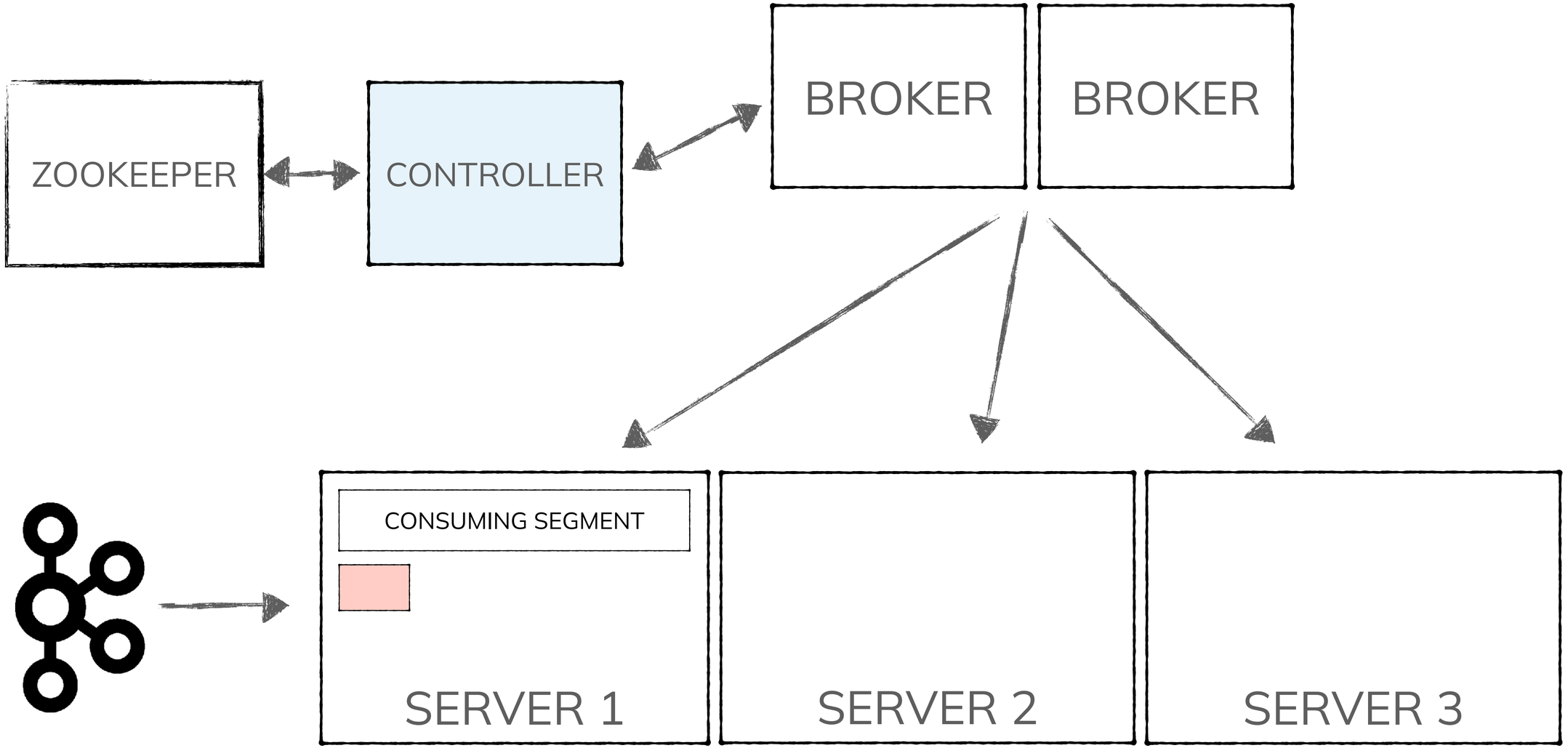
Realtime ingestion: table configuration

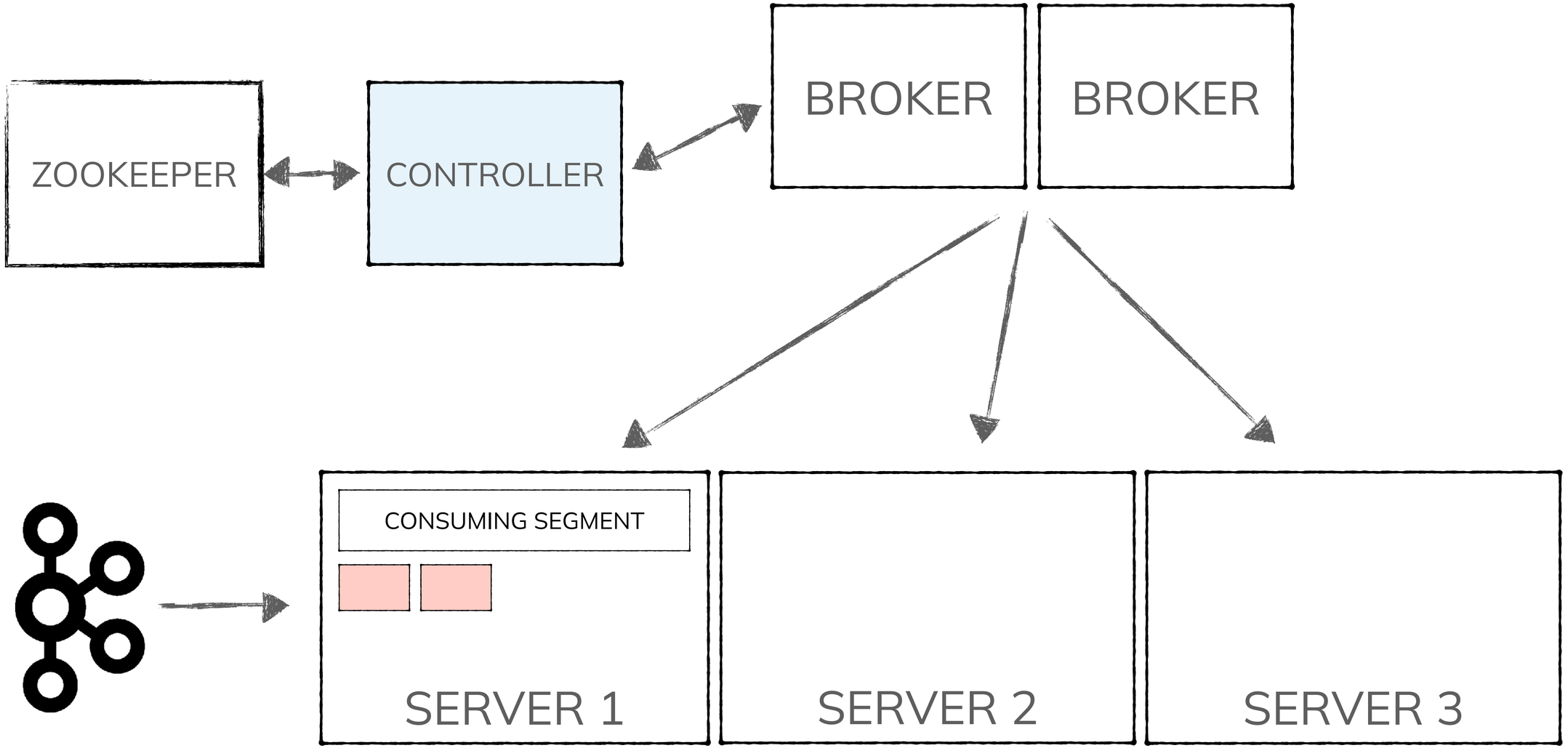
```
  "tenants": {},
  "tableIndexConfig": {
    "loadMode": "MMAP",
    "streamConfigs": {
      "streamType": "kafka",
      "stream.kafka.consumer.type": "lowlevel",
      "stream.kafka.topic.name": "transcript-topic",
      "stream.kafka.decoder.class.name":
"org.apache.pinot.plugin.stream.kafka.KafkaJSONMessageDecoder",
      "stream.kafka.consumer.factory.class.name":
"org.apache.pinot.plugin.stream.kafka20.KafkaConsumerFactory",
      "stream.kafka.broker.list": "kafka:9092",
      "realtime.segment.flush.threshold.time": "3600000",
      "realtime.segment.flush.threshold.rows": "50000",
      "stream.kafka.consumer.prop.auto.offset.reset": "smallest"
    }
  },
  "metadata": {
```

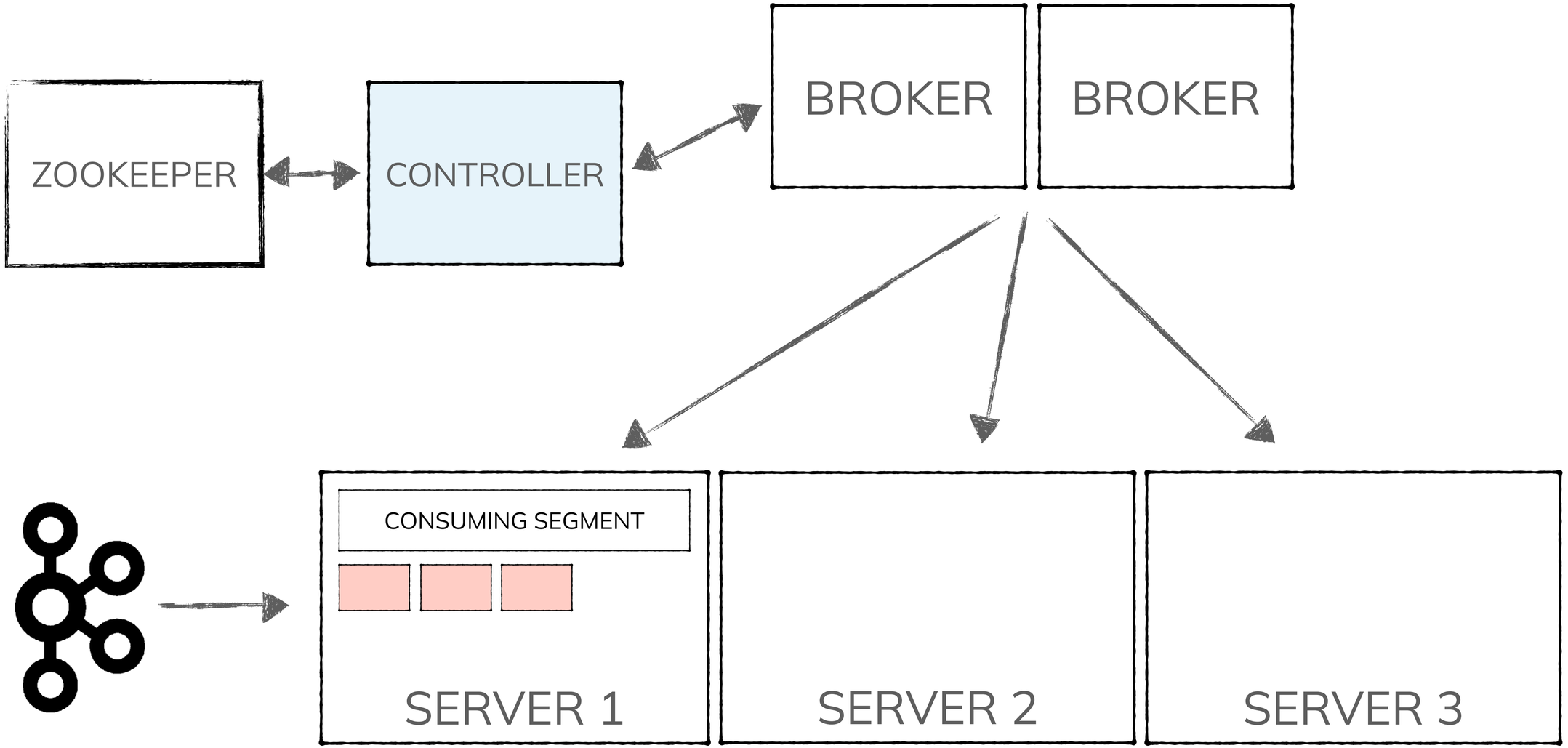

ZOOKEEPER
CONTROLLER
BROKER
BROKER
SERVER 1
SERVER 2
SERVER 3

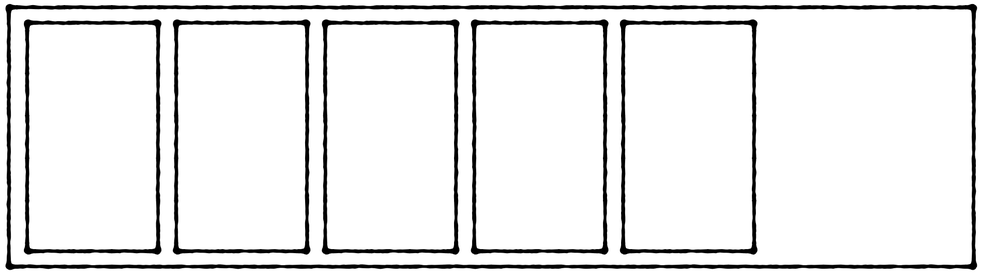

ZOOKEEPER
CONTROLLER
BROKER
BROKER
SERVER 1
SERVER 2
SERVER 3


ZOOKEEPER
CONTROLLER
BROKER
BROKER
CONSUMING SEGMENT
SERVER 1
SERVER 2
SERVER 3

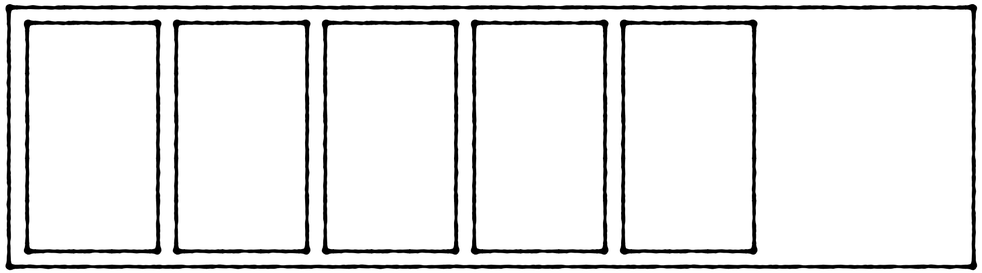

ZOOKEEPER
CONTROLLER
BROKER
BROKER
CONSUMING SEGMENT
SERVER 1
SERVER 2
SERVER 3


ZOOKEEPER
CONTROLLER
BROKER
BROKER
CONSUMING SEGMENT
SERVER 1
SERVER 2
SERVER 3

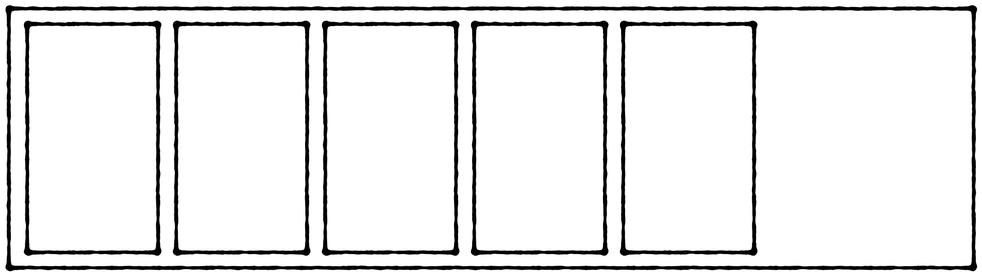

ZOOKEEPER
CONTROLLER
BROKER
BROKER
CONSUMING SEGMENT
SERVER 1
SERVER 2
SERVER 3



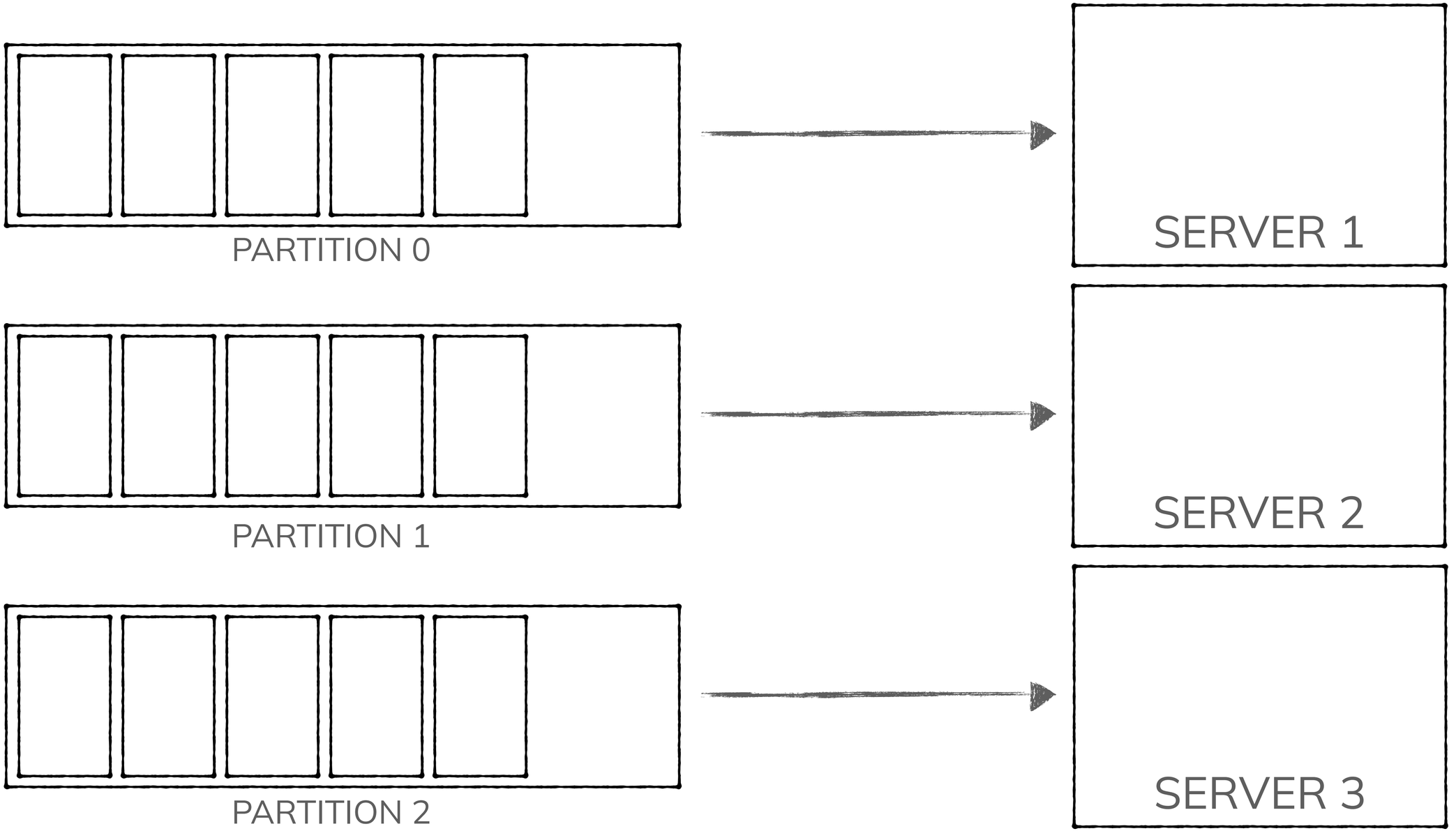
PARTITION 0

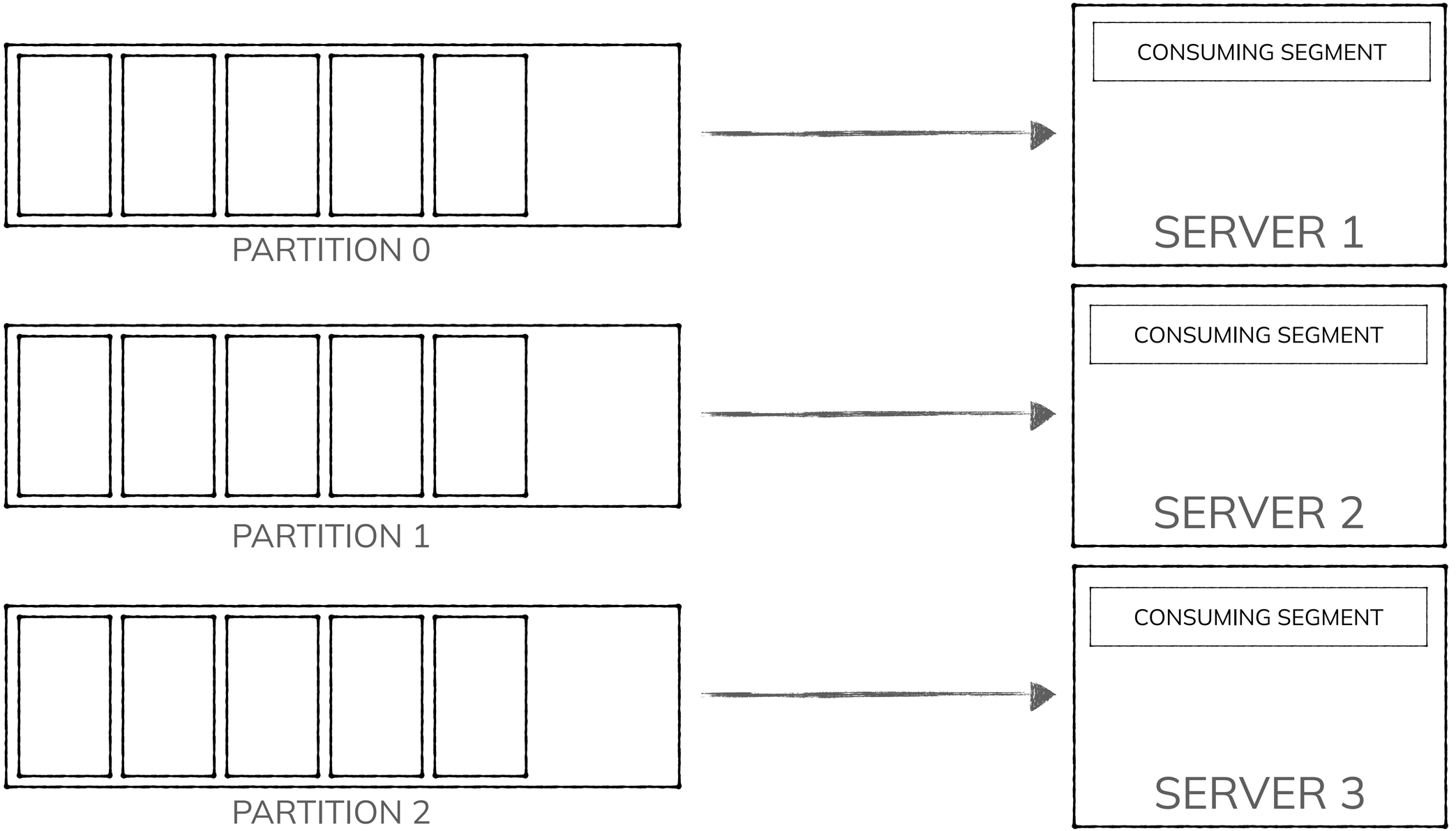


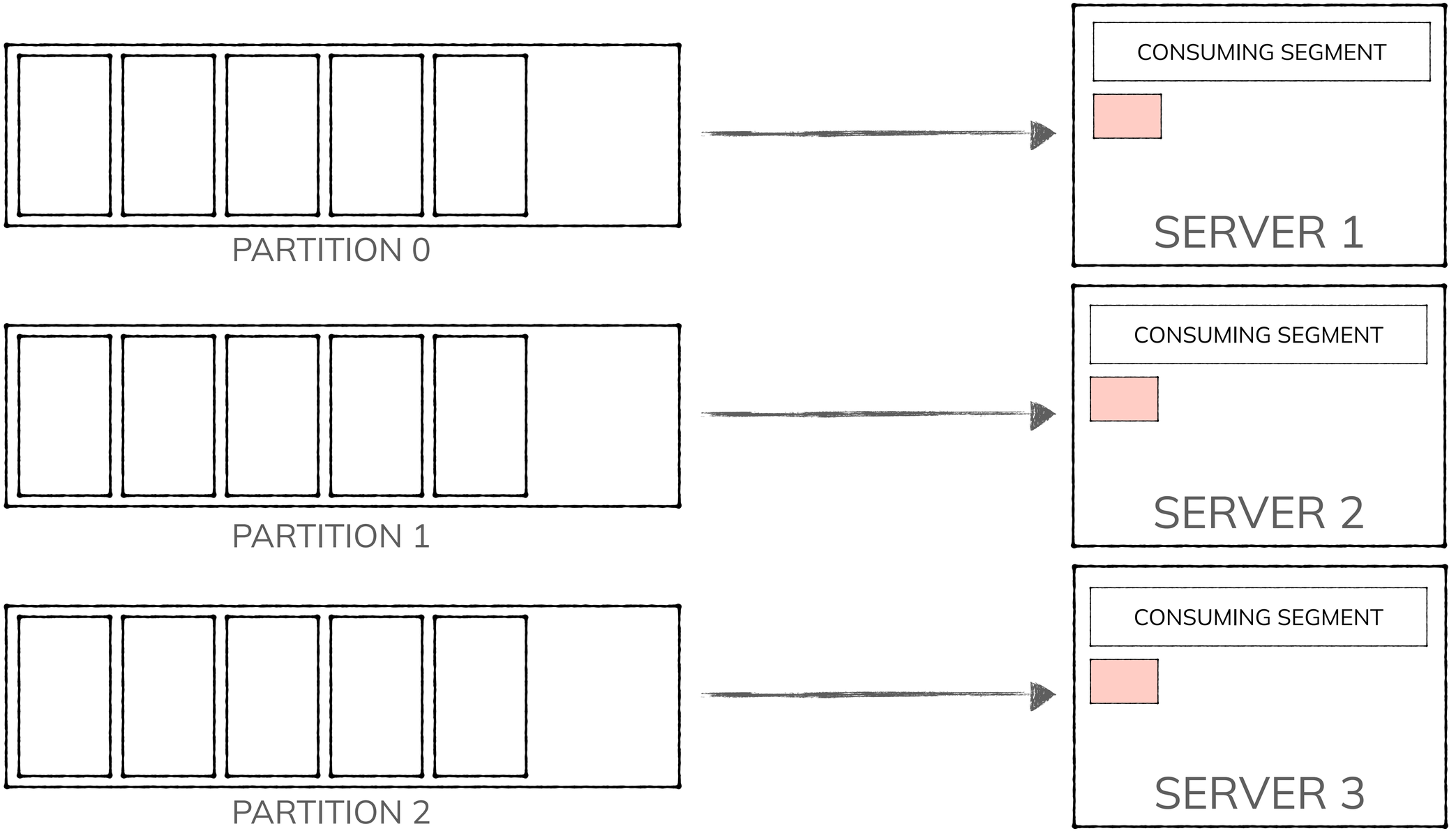
PARTITION 1

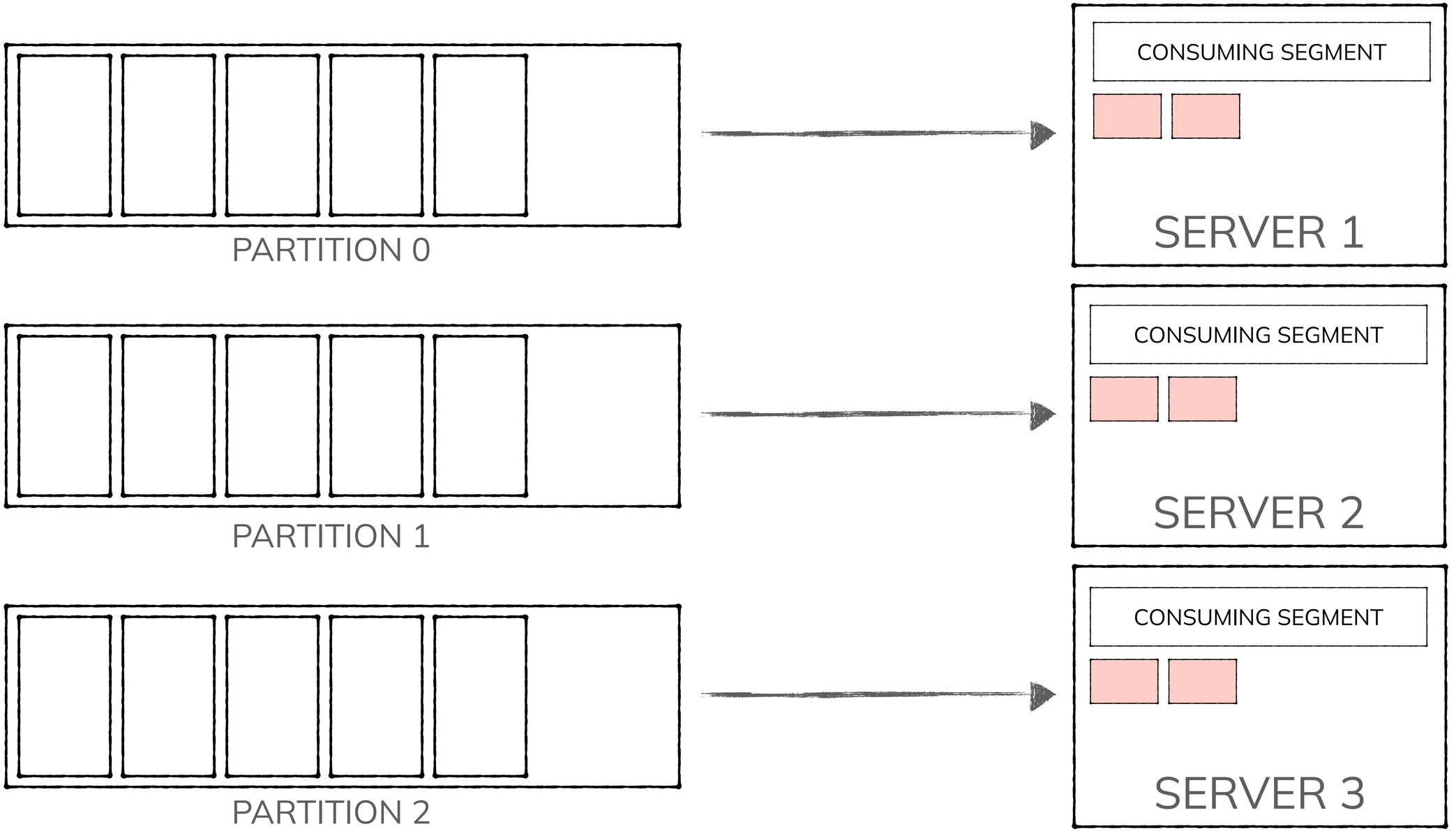


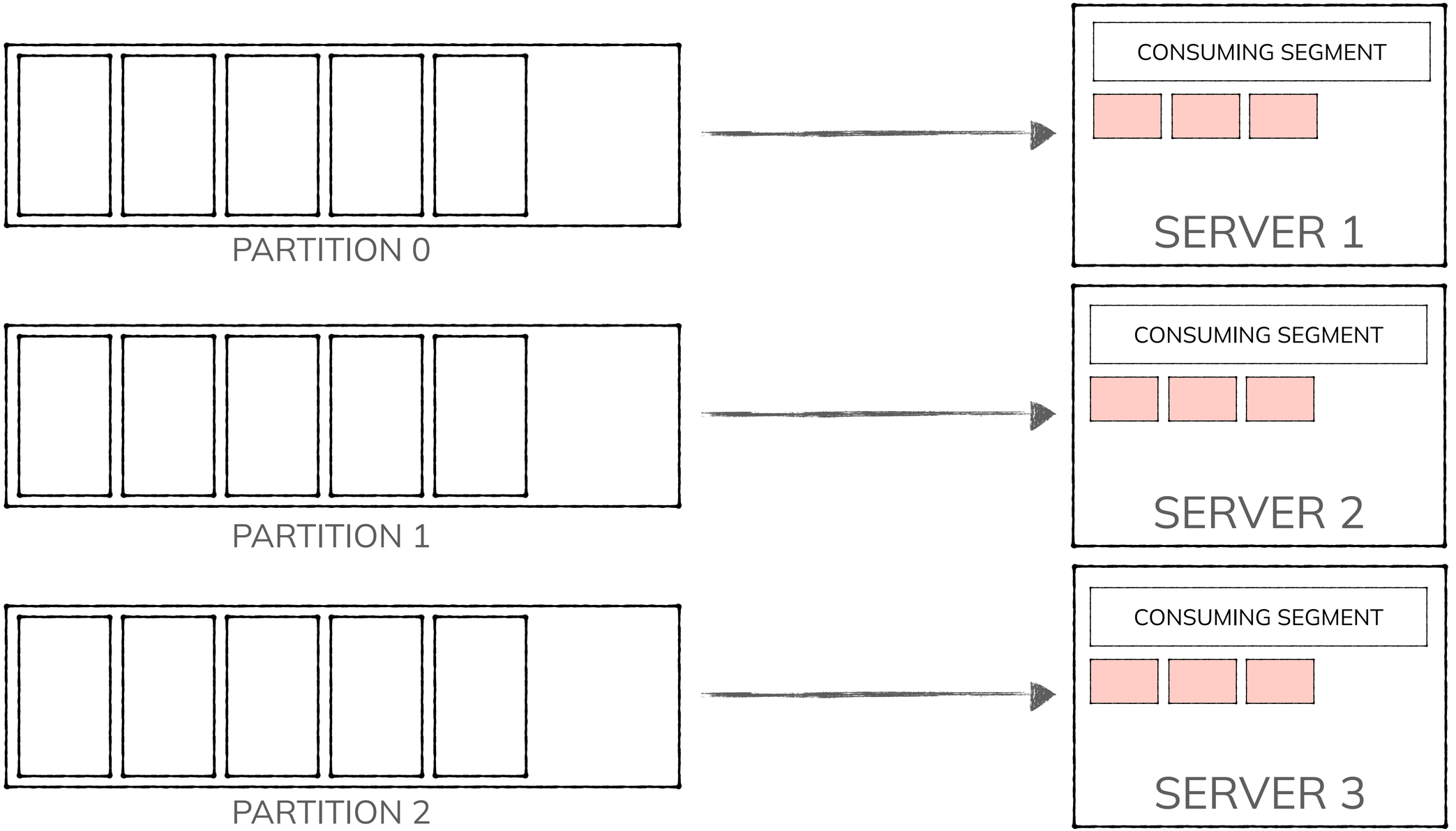
PARTITION 2

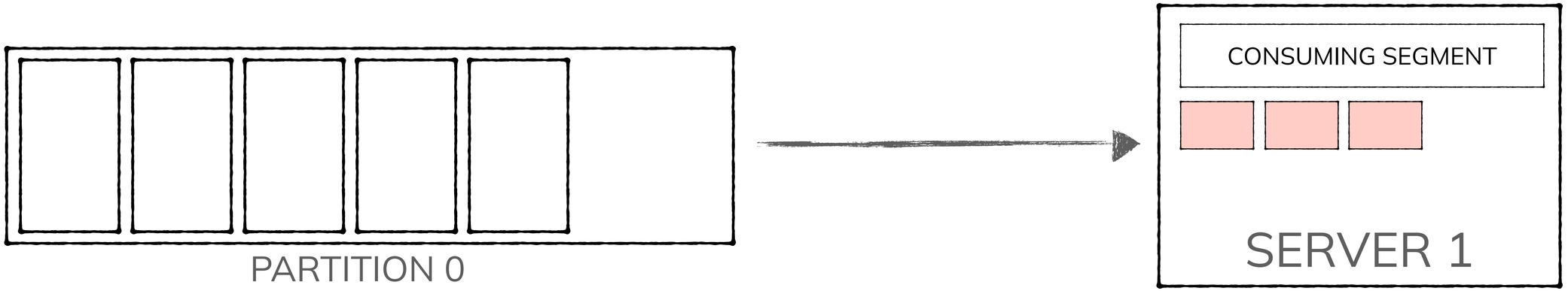

PARTITION 0
PARTITION 1
PARTITION 2
SERVER 1
SERVER 2
SERVER 3


PARTITION 0
PARTITION 1
PARTITION 2
CONSUMING SEGMENT
SERVER 1
CONSUMING SEGMENT
SERVER 2
CONSUMING SEGMENT
SERVER 3

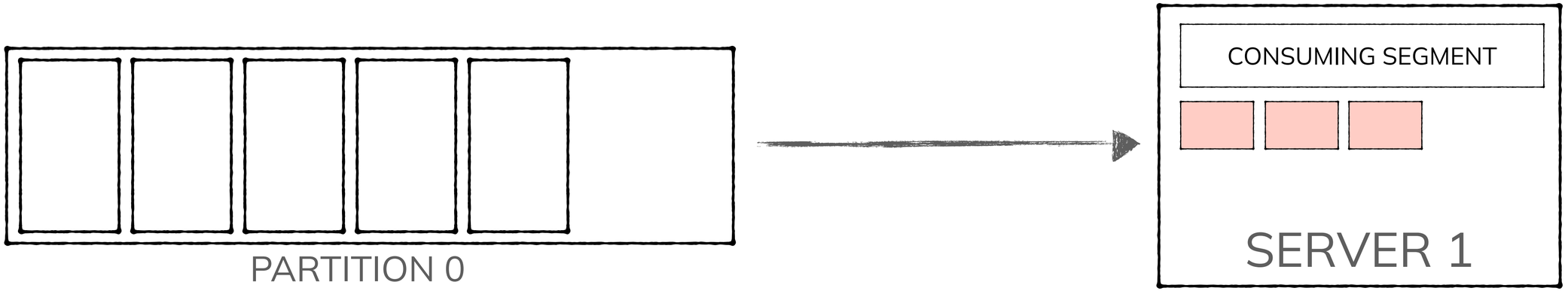

PARTITION 0
PARTITION 1
PARTITION 2
CONSUMING SEGMENT
SERVER 1
CONSUMING SEGMENT
SERVER 2
CONSUMING SEGMENT
SERVER 3


PARTITION 0
PARTITION 1
PARTITION 2
CONSUMING SEGMENT
SERVER 1
CONSUMING SEGMENT
SERVER 2
CONSUMING SEGMENT
SERVER 3


PARTITION 0
PARTITION 1
PARTITION 2
CONSUMING SEGMENT
SERVER 1
CONSUMING SEGMENT
SERVER 2
CONSUMING SEGMENT
SERVER 3

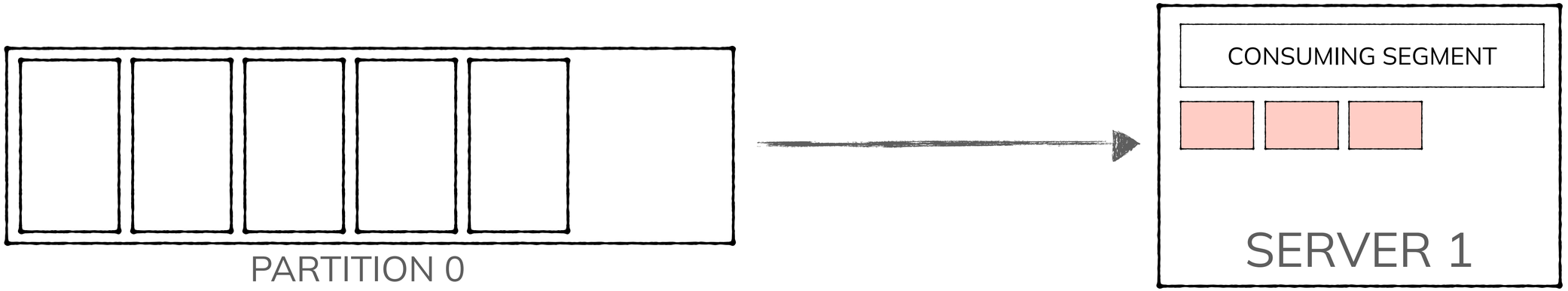



```
{
    "studentID": 205,
    "firstName": "Natalie",
    "lastName": "Jones",
    "gender": "Female",
    "subject": "Maths",
    "score": 3.8,
    "timestamp": 1571900400000
}
```



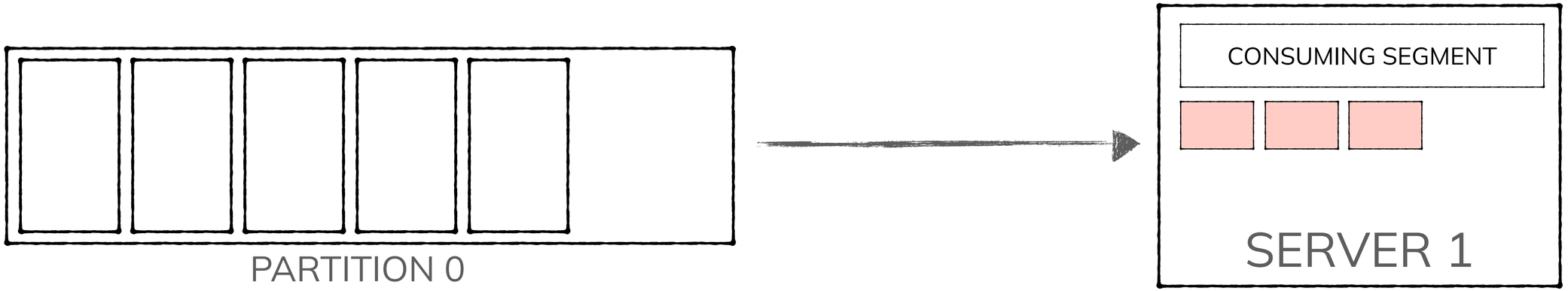
1. Choose a partitioning key

```
{
    "studentID": 205,
    "firstName": "Natalie",
    "lastName": "Jones",
    "gender": "Female",
    "subject": "Maths",
    "score": 3.8,
    "timestamp": 1571900400000
}
```



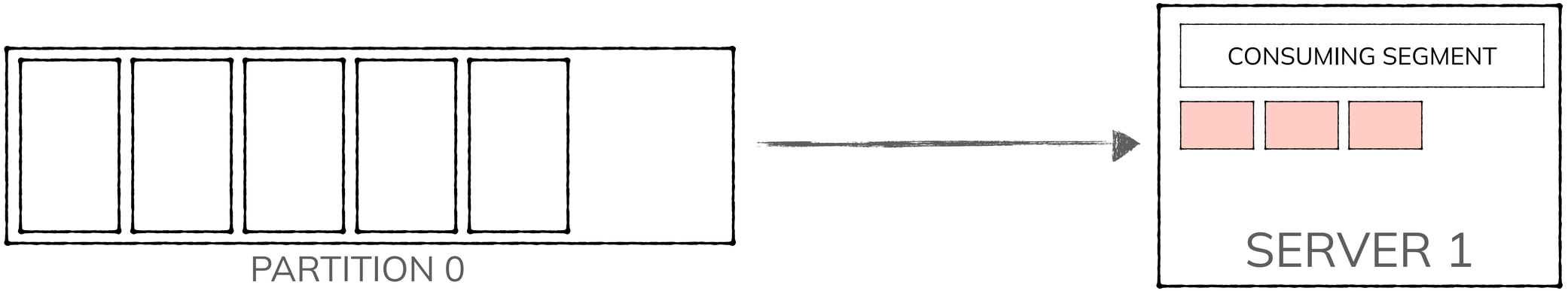

```
{
    "studentID": 205,
    "firstName": "Natalie",
    "lastName": "Jones",
    "gender": "Female",
    "subject": "Maths",
    "score": 3.8,
    "timestamp": 1571900400000
}
```

1. Choose a partitioning key
2. Choose a hash function




```
{
    "studentID": 205,
    "firstName": "Natalie",
    "lastName": "Jones",
    "gender": "Female",
    "subject": "Maths",
    "score": 3.8,
    "timestamp": 1571900400000
}
```

1. Choose a partitioning key
2. Choose a hash function
3. Messages will be assigned to segments by key




```
{
    "studentID": 205,
    "firstName": "Natalie",
    "lastName": "Jones",
    "gender": "Female",
    "subject": "Maths",
    "score": 3.8,
    "timestamp": 1571900400000
}
```

1. Choose a partitioning key
2. Choose a hash function
3. Messages will be assigned to segments by key
4. Brokers can now route queries by partition

```
SELECT AVG(score) WHERE studentID=205
```

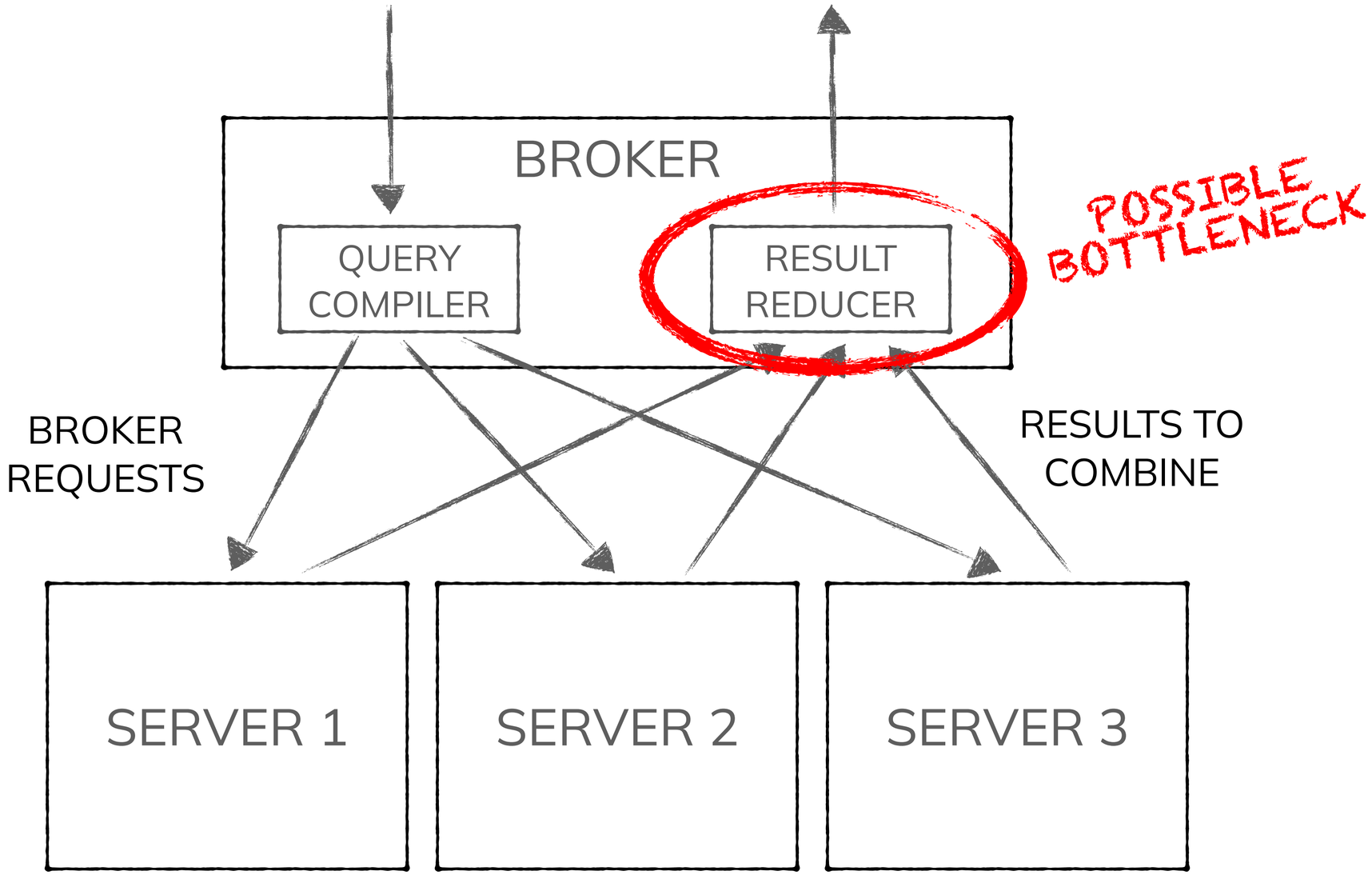
multi-stage queries

Multi-Stage Query Engine

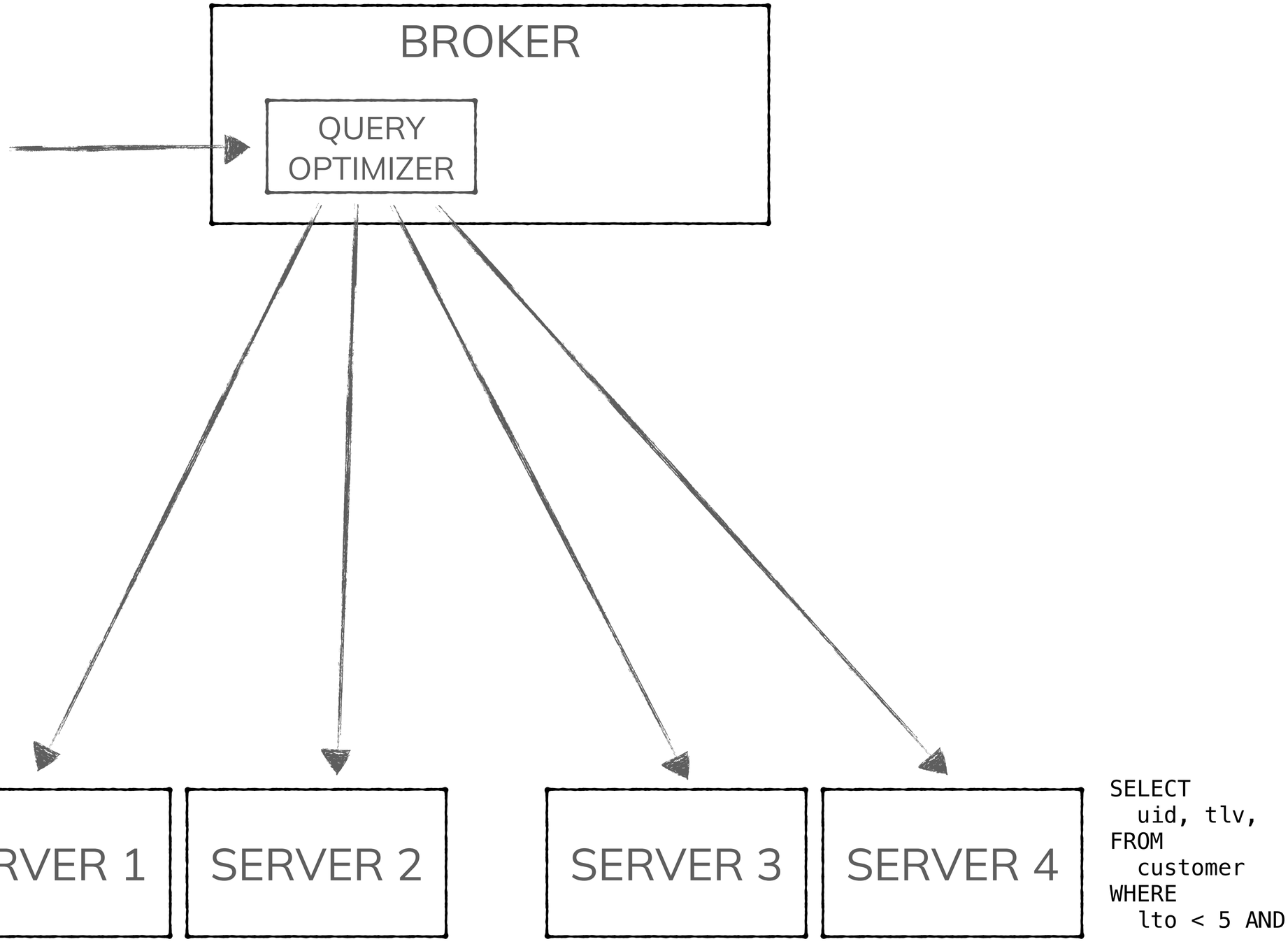
- Pinot’s architecture supports low-latency queries on batch and streaming data at high concurrency
- It is optimized for queries that filter and aggregate
- Query processing occurs on Servers
- Last-mile reducing happens on Brokers—for small data sets
- Brokers cannot process large join data sets

```
SELECT
  AVERAGE(customer.ltv)
FROM
  customer
WHERE
  customer.segment = ‘PREMIUM’
```

16382.46



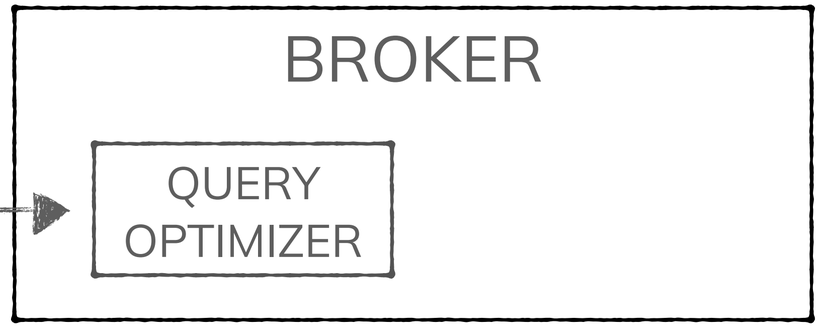
```
SELECT
  status.uid,
  status.rName,
  customer.ltv
FROM
  status INNER JOIN customer
  ON
  status.uid = customer.uid
WHERE
  status.ic < 10 AND
  customer.lto > 5
```

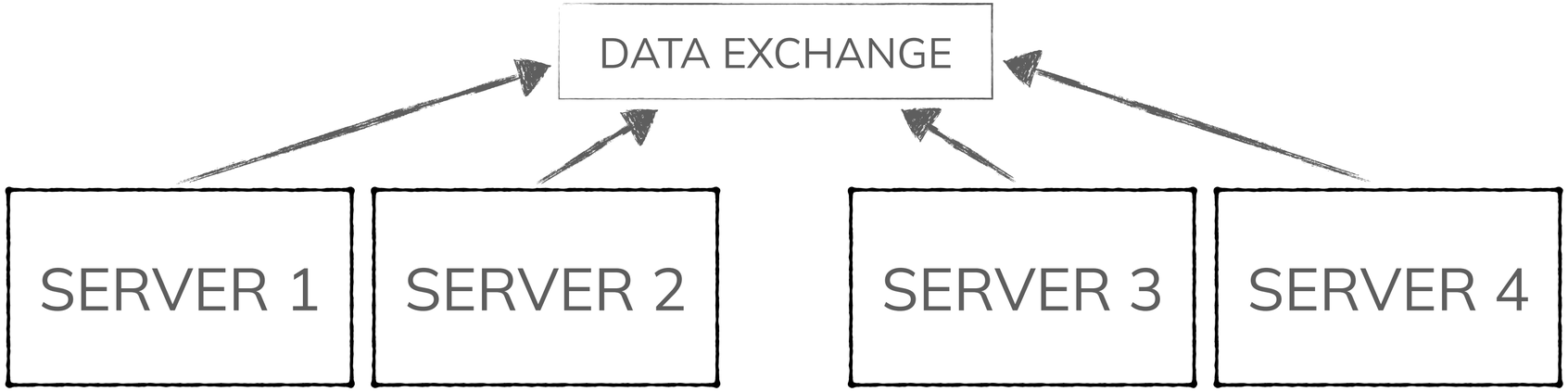



```
SELECT
  uid, rName,
FROM
  status
WHERE
  ic < 10 AND
```

```
SELECT
  uid, tlv,
FROM
  customer
WHERE
  lto < 5 AND
```

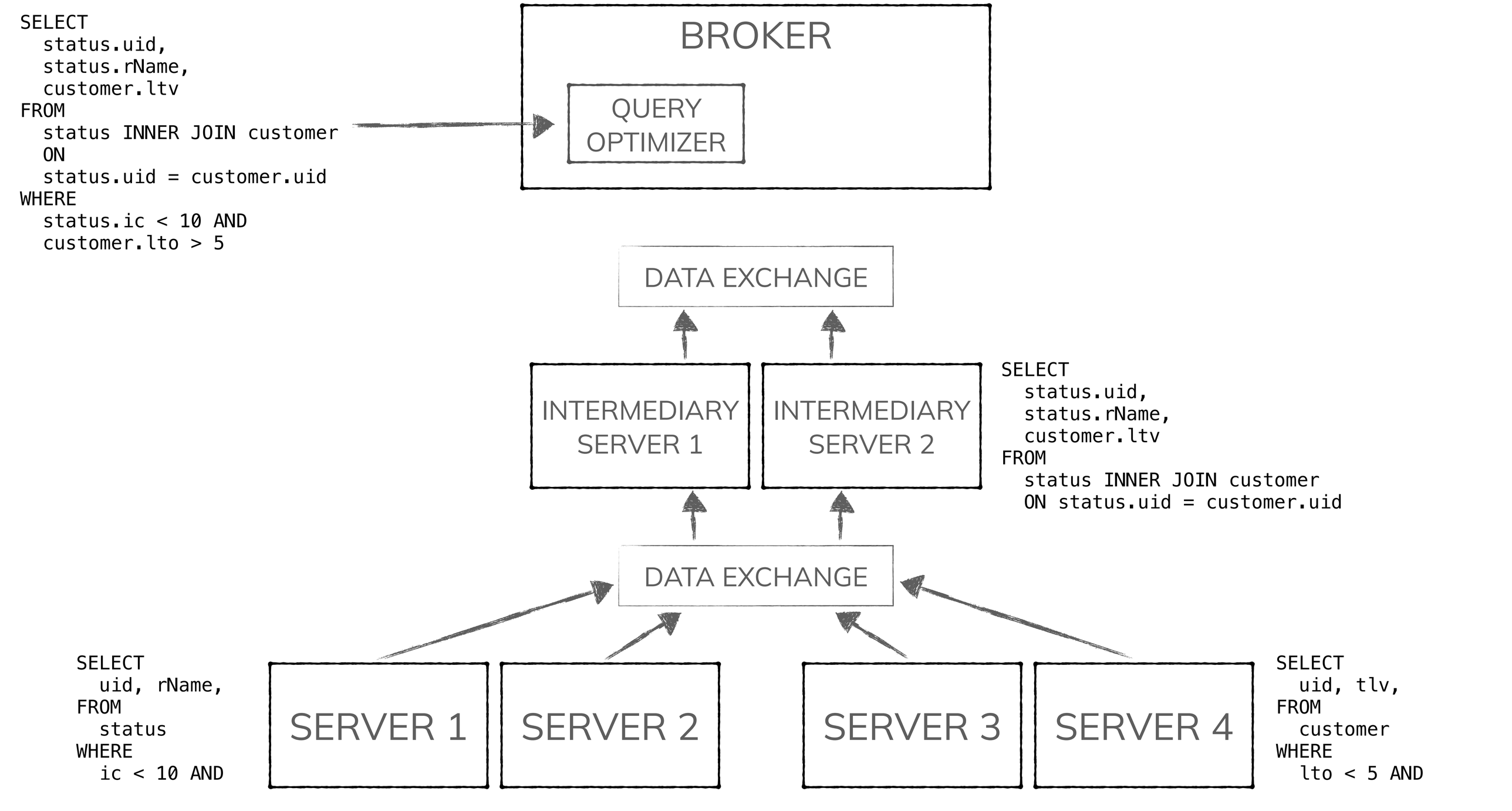
```
SELECT
  status.uid,
  status.rName,
  customer.ltv
FROM
  status INNER JOIN customer
  ON
  status.uid = customer.uid
WHERE
  status.ic < 10 AND
  customer.lto > 5
```



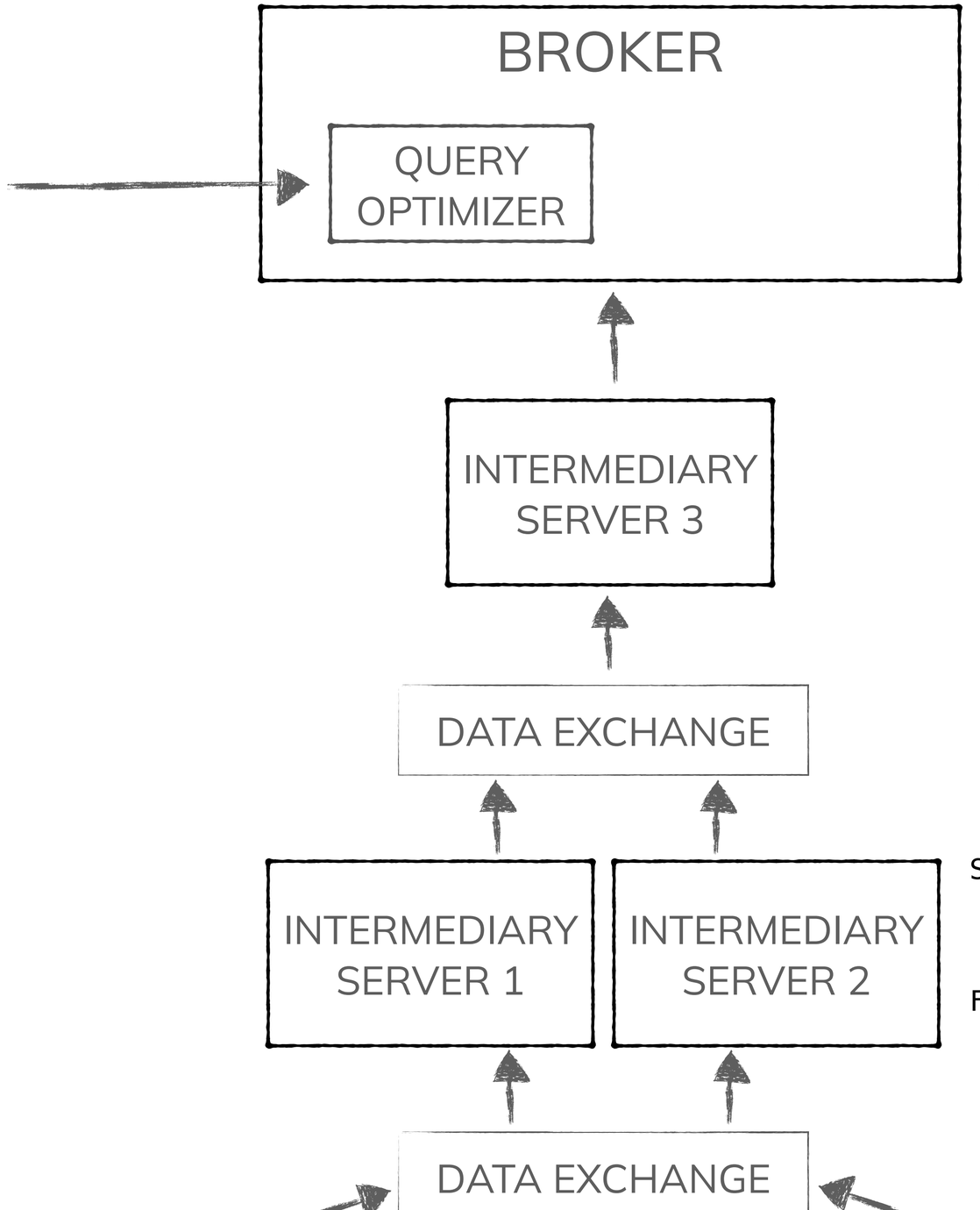



```
SELECT
  uid, rName,
FROM
  status
WHERE
  ic < 10 AND
```

```
SELECT
  uid, tlv,
FROM
  customer
WHERE
  lto < 5 AND
```


SELECT
status.uid,
status.rName,
customer.ltv
FROM
status INNER JOIN customer
ON
status.uid = customer.uid
WHERE
status.ic < 10 AND
customer.lto > 5
BROKER
QUERY OPTIMIZER
DATA EXCHANGE
INTERMEDIARY SERVER 1
INTERMEDIARY SERVER 2
SELECT
status.uid,
status.rName,
customer.ltv
FROM
status INNER JOIN customer
ON status.uid = customer.uid
DATA EXCHANGE
SELECT
uid, rName,
FROM
status
WHERE
ic < 10 AND
SERVER 1
SERVER 2
SERVER 3
SERVER 4
SELECT
uid, tlv,
FROM
customer
WHERE
lto < 5 AND

```
SELECT
  status.uid,
  status.rName,
  customer.ltv
FROM
  status INNER JOIN customer
  ON
  status.uid = customer.uid
WHERE
  status.ic < 10 AND
  customer.lto > 5
```



id	rName	ltv
1	Acme	532099.84
2	Initech	154448.54
3	Umbrella	1322840.35
4	Hooli	9615938.29
5	Globex	1176199.55

```
SELECT
  status.uid,
  status.rName,
  customer.ltv
FROM
  status INNER JOIN customer
  ON status.uid = customer.uid
```

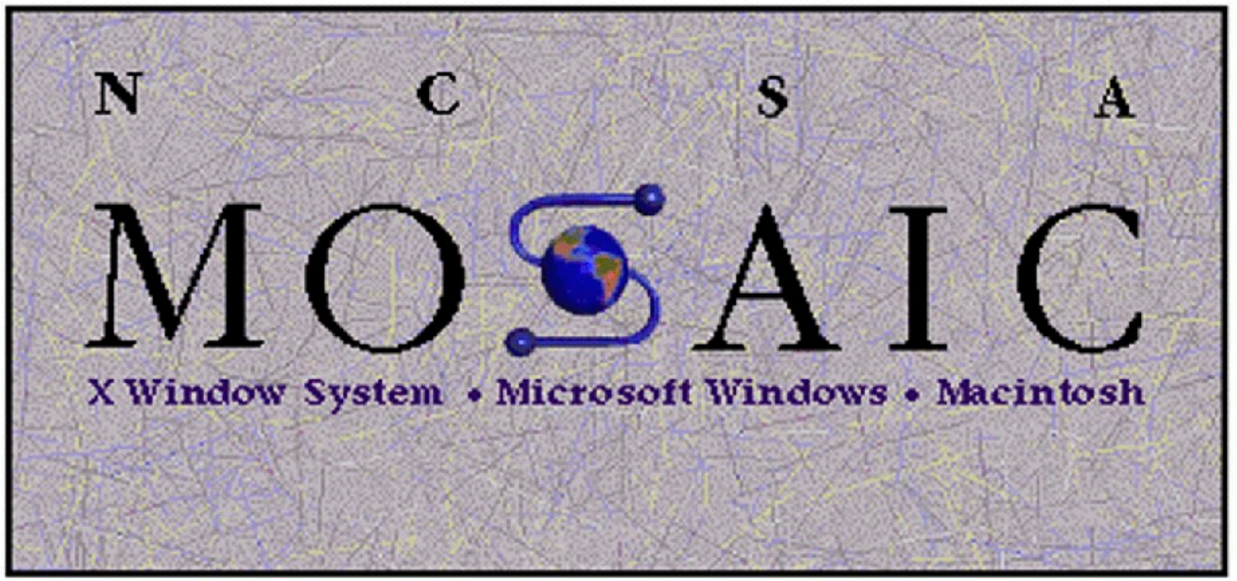
remember our history

IBM

NCSA Mosaic Home Page - NCSA Mosaic

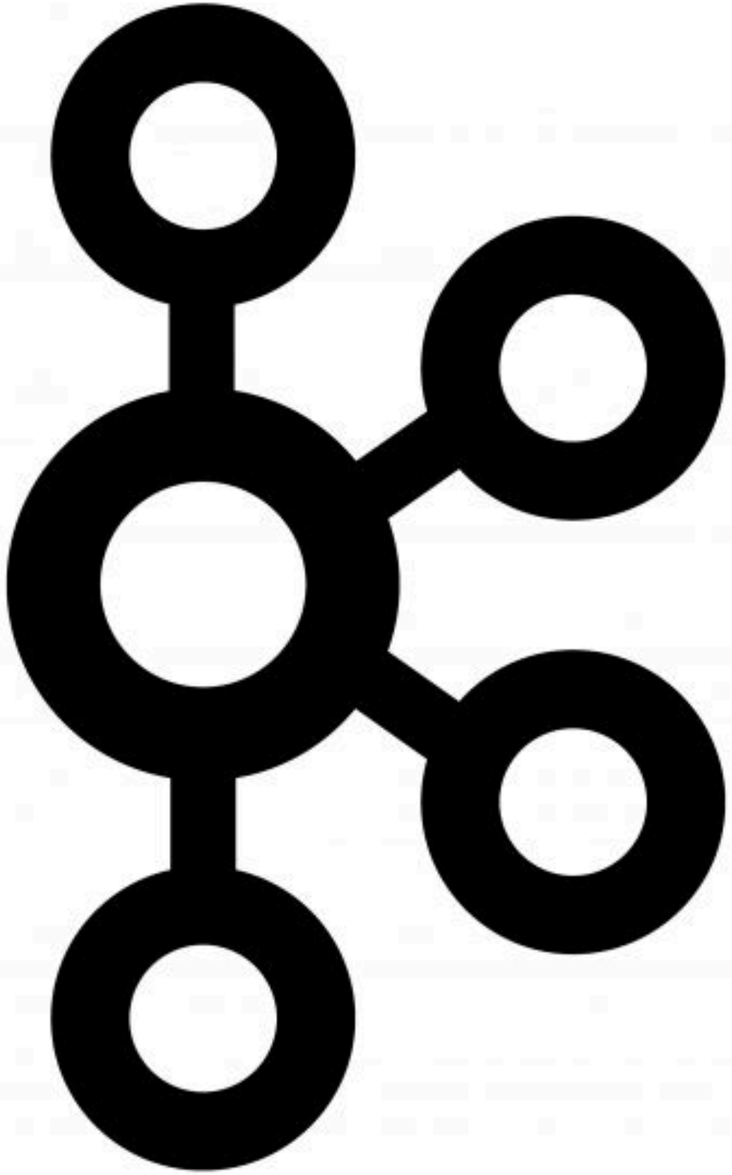
File Edit Source Manager View Navigate Tools Hotlists Help

http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/



Welcome to NCSA Mosaic, an Internet information browser and World Wide Web client. NCSA Mosaic was developed at the National Center for Supercomputing Applications at the University of Illinois in Urbana-Champaign. NCSA Mosaic software is copyrighted by The Board of Trustees of the University of Illinois (UI), and ownership remains with the UI.

NCSA | Mosaic | Photo CD | Metasearch



10:18
◀ Search

burger

Delivery Fees | Offers | Ratings

King's Burger
Fries, Burgers
4.8 ★ (1,600+)
$15 avg per person

2.6 mi • 23 min
$4.99 delivery fee

Smashburger
Sponsored • Burgers
4.6 ★ (20,800+)

1.2 mi • 17 min
$0 delivery fee

Home | Grocery | Retail | Browse | Orders

2.6 mi • 23 min

$4.99 delivery fee

Who is a decision maker?

Everyone.

Thank you!

https://stree.ai/slack

@tlberglund