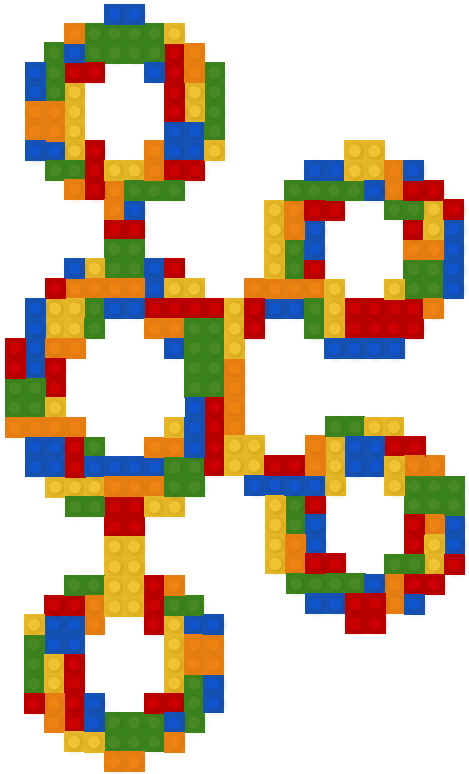
Brick-by-Brick

Exploring the Elements of Apache Kafka®

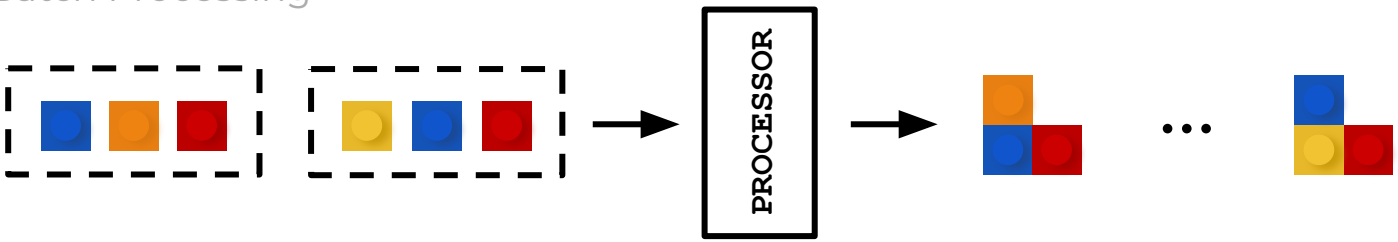
dfine@confluent.io @TheDanicaFine linkedin.com/in/danica-fine/



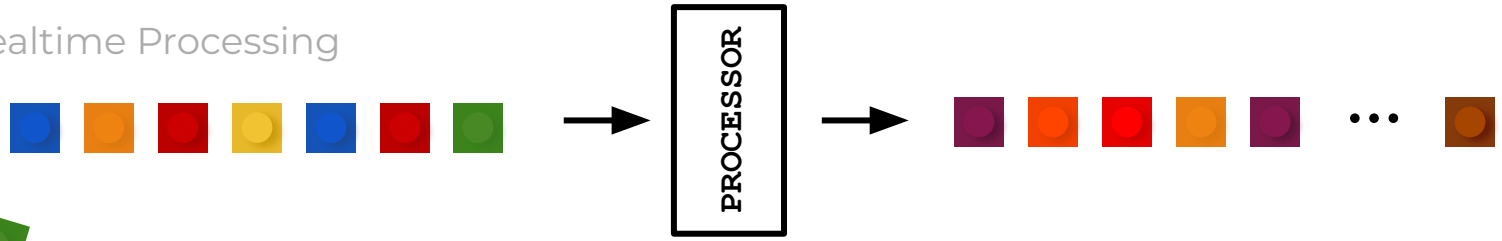
Why Kafka?

… because real-time *makes sense*.

Batch Processing

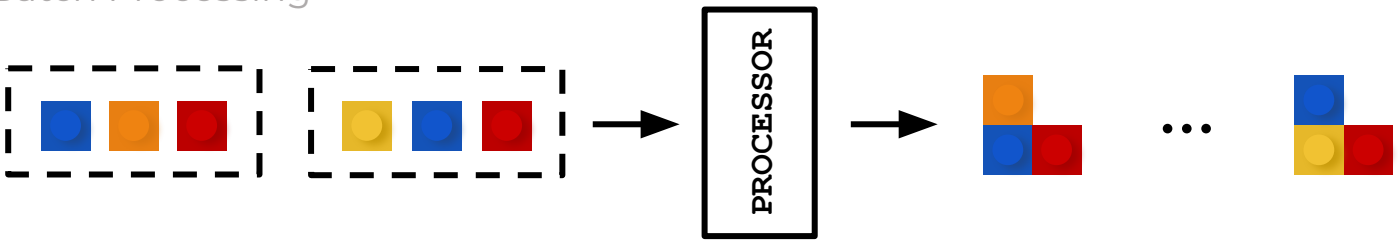


Realtime Processing

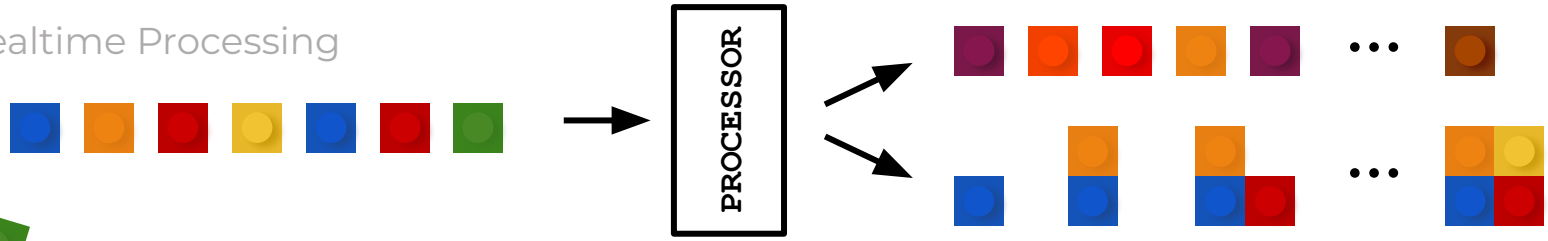


… because real-time *makes sense*.

Batch Processing



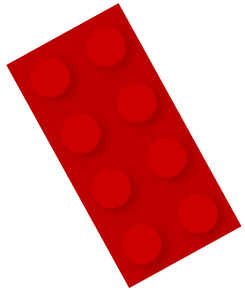
Realtime Processing

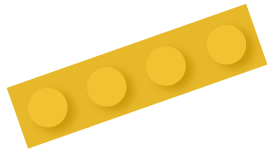


Kafka Components

Events

- Natural way to reason about things
- Indicate that something *has happened*
  - When
  - What/Who
- `event = notification + state`

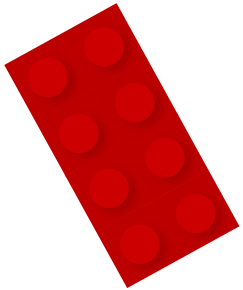


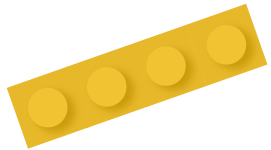
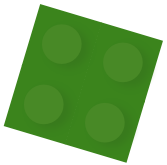




PIZZA
alamy
PIZZA

Events

- Natural way to reason about things
- Indicate that something *has happened*
  - When
  - What/Who
- `event = notification + state`
- Immutable pieces of information

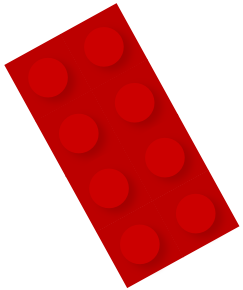


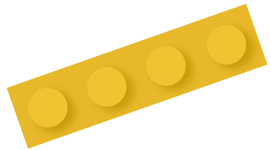
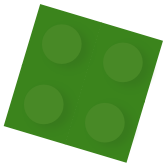




PIZZA
alamy
PIZZA

Events

- Natural way to reason about things
- Indicate that something *has happened*
  - When
  - What/Who
- `event = notification + state`
- Immutable pieces of information



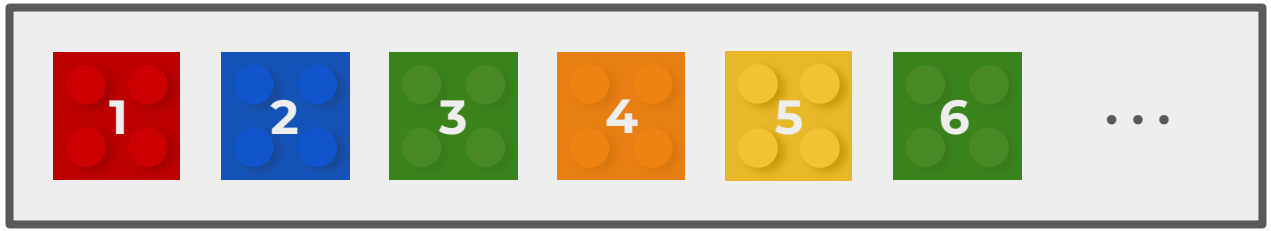
Messages and Records

An aside.

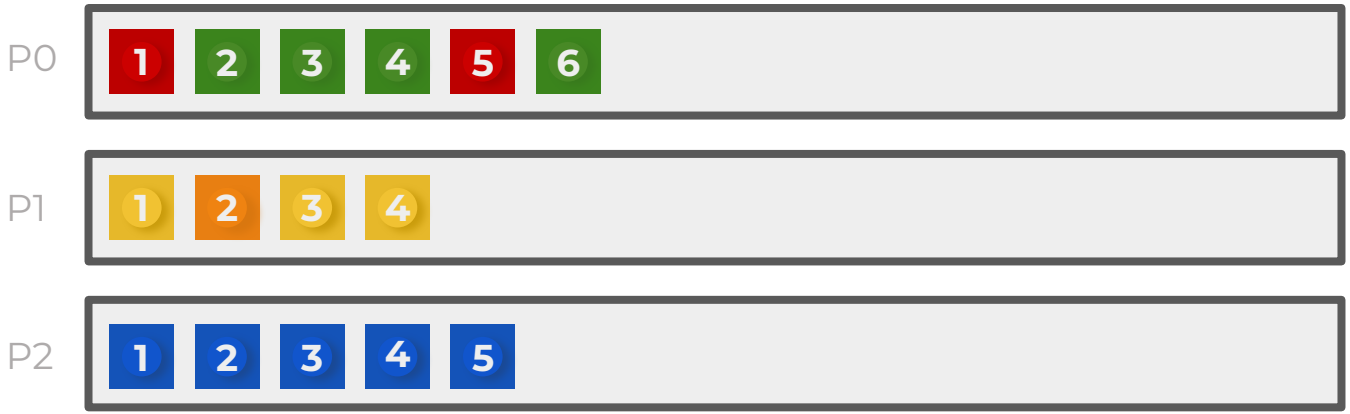
Kafka Storage

The Topic

A durable, immutable, append-only log of events.

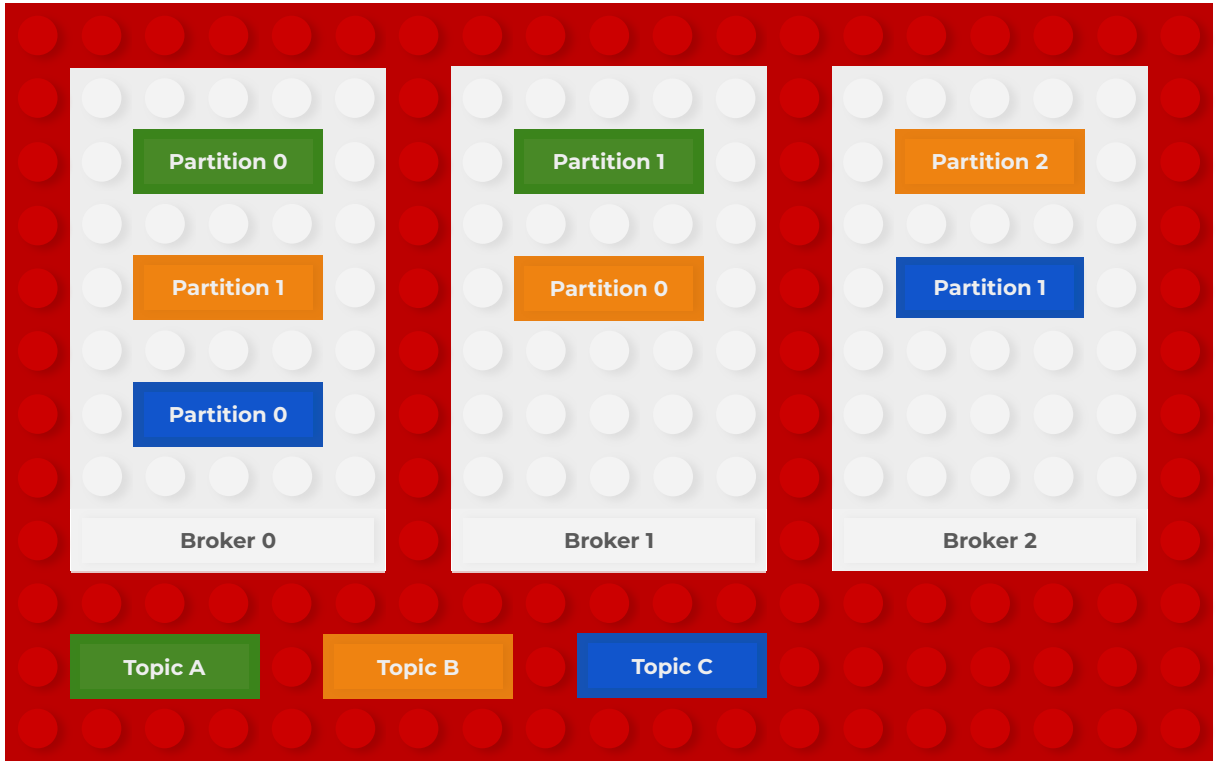


Topic-Partitions

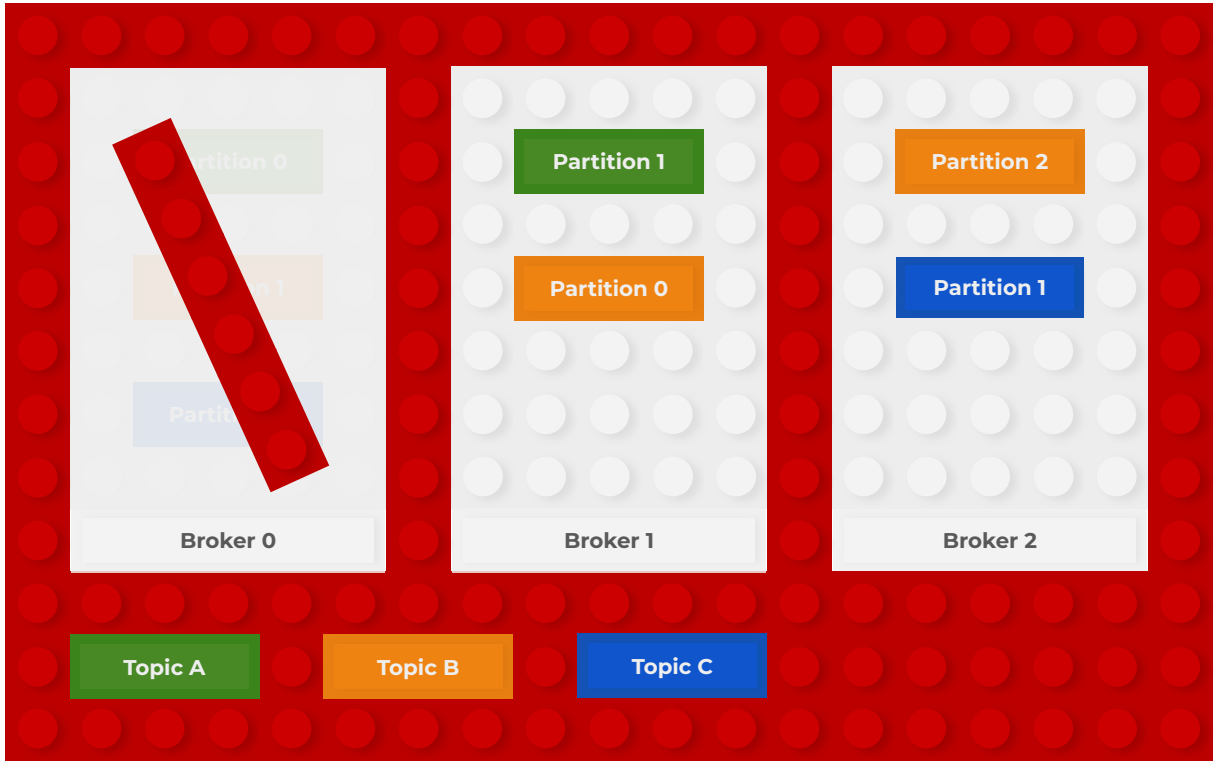


Why partitions?

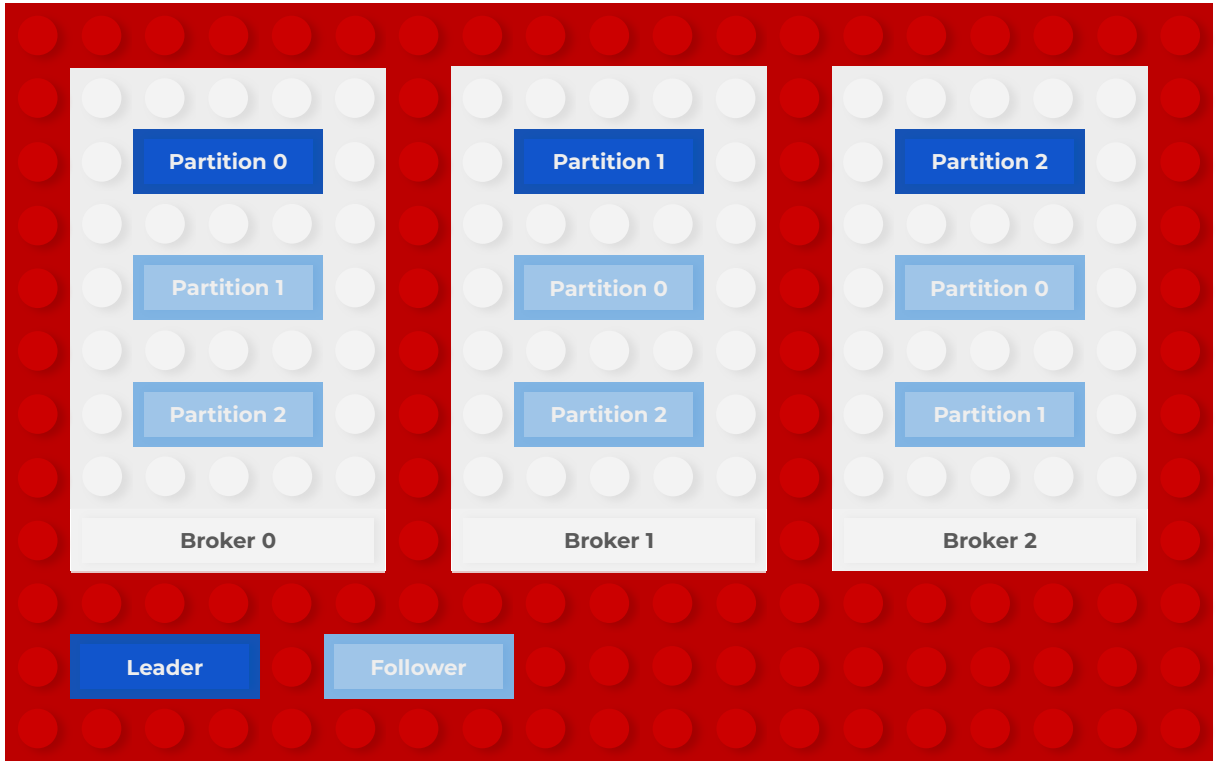
A Kafka Cluster



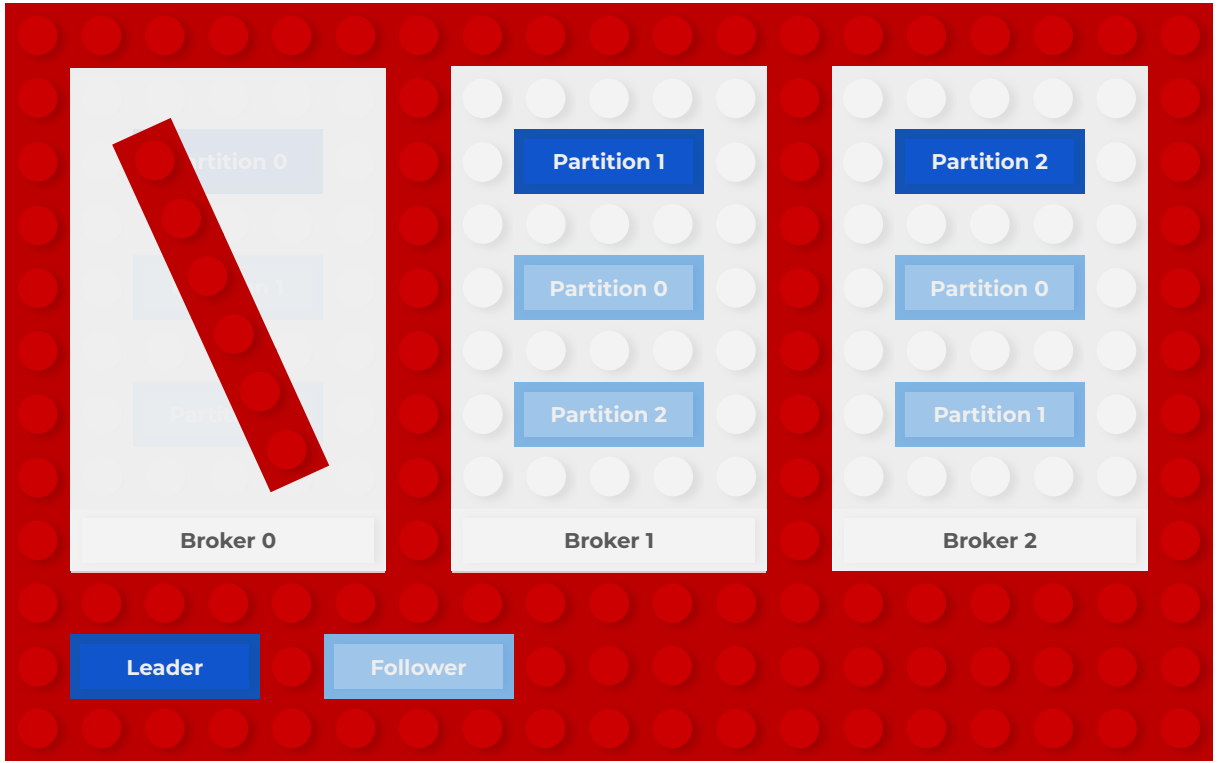
A Kafka Cluster



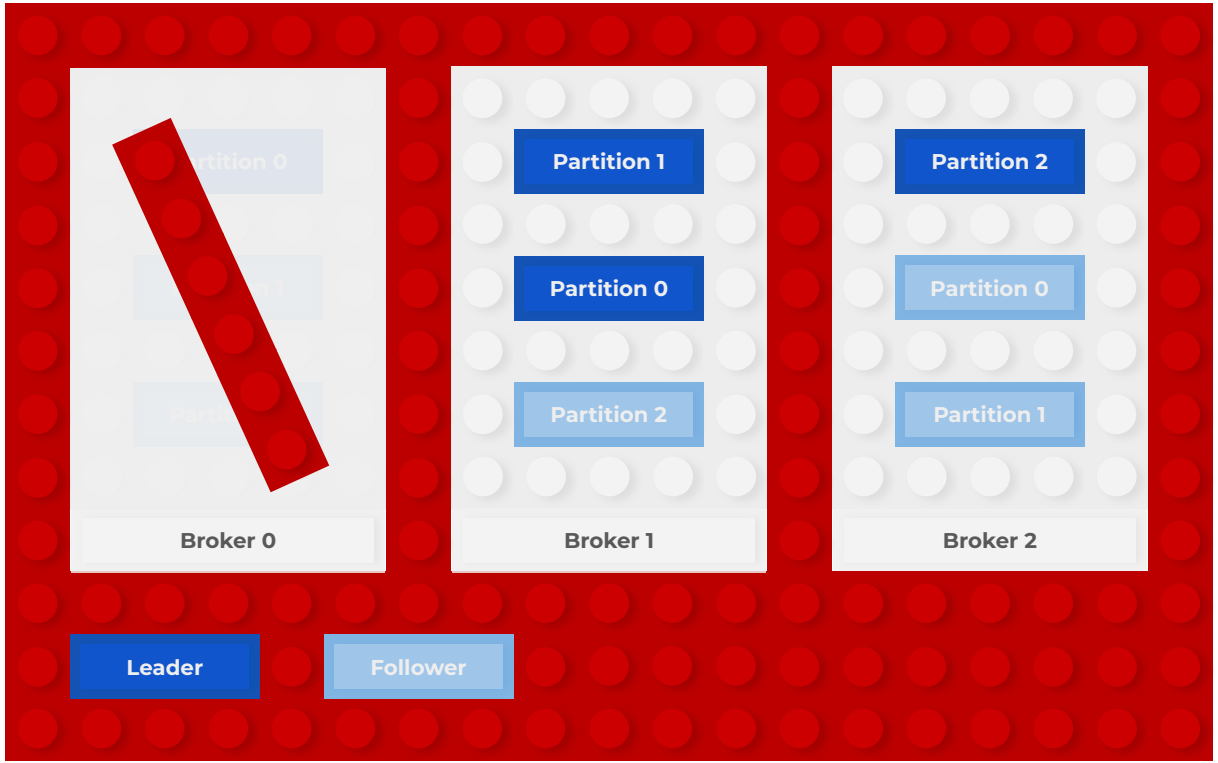
Replication



Replication



Replication



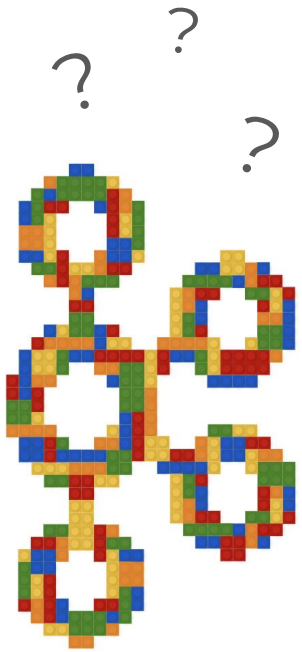
A record

- Required Fields
  - Topic
  - Value
- [Optional] Fields
  - Partition
  - Timestamp
  - Key
  - Headers

A record

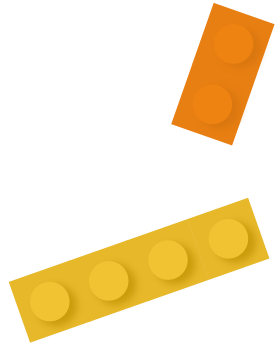
- Required Fields
  - Topic: `'cart-updates'`
  - Value: `'Add 1 Blue 1x1 Plate to Cart 1234'`
- [Optional] Fields
  - Partition: `None`
  - Timestamp: `None`
  - Key: `'Cart 1234'`
  - Headers: `None`

Producer



Brokers and Bytes

- Store bytes is more efficient
- Serialization formats:
  - Avro
  - JSON
  - Protobuf
- Configure producers to serialize the object



Schemas

```
{
  "doc": "Pick a Brick cart update",
  "fields": [
    {
      "name": "cart_id",
      "type": "int"
    },
    {
      "name": "cart_action",
      "type": {
        "name": "Status",
        "type": "cart_action",
        "symbols": ["ADD", "REMOVE",
                    "UPDATE_QTY"]
      }
    },
    {
      "name": "element_id",
      "type": "int"
    },
    {
      "name": "element_desc",
      "type": "string"
    },
    {
      "name": "quantity",
      "type": "int"
    }
  ],
  "name": "cart_updates",
  "namespace": "lego",
  "type": "record"
}
```

Producing a Record: A recap

key: 1234

```
{
  "cart_id": 1234,
  "cart_action": "ADD",
  "element_id": 302423,
  "element_desc": "Bright Blue
                   1x1 Plate",
  "quantity": 1
}
```

Producing a Record: A recap

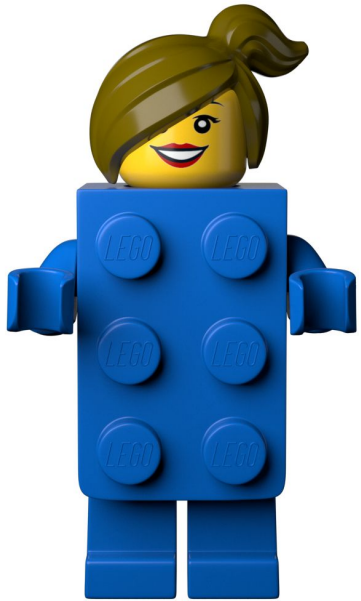
1. Serialize
2. Partition
3. Batch
4. [Compress]
5. Request

Producing a Record: A recap

Cart Updates Topic

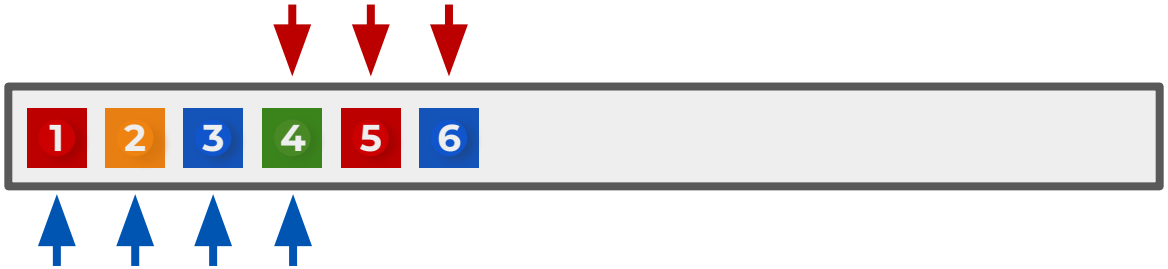
Making it More Interesting

Producer

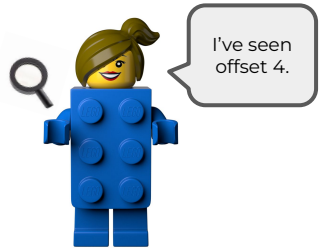


Consumer

Producer



Consumer

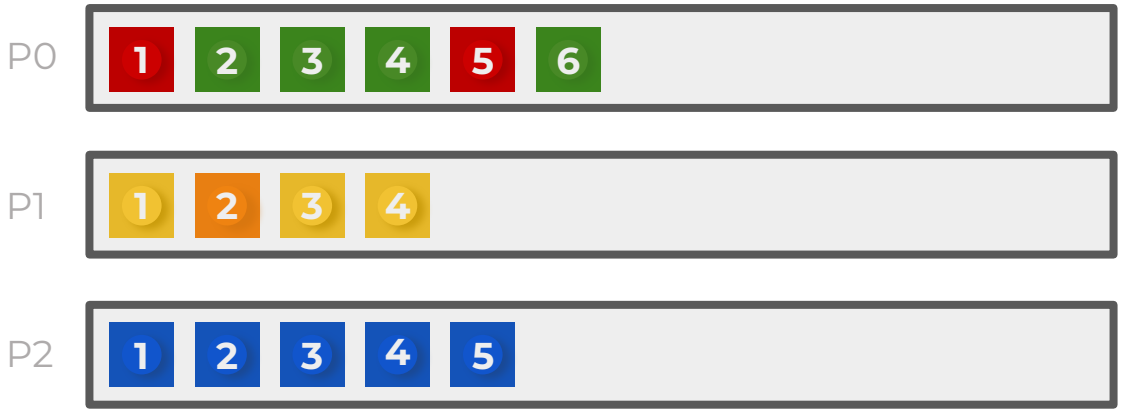


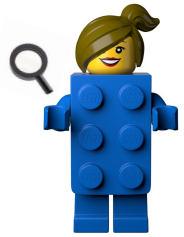
Consumer 1

Consumer 2

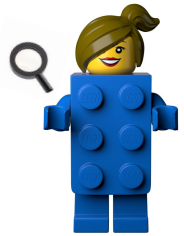
Consumer 3

A Consumer Group

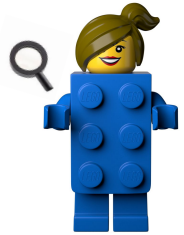




Consumer 1

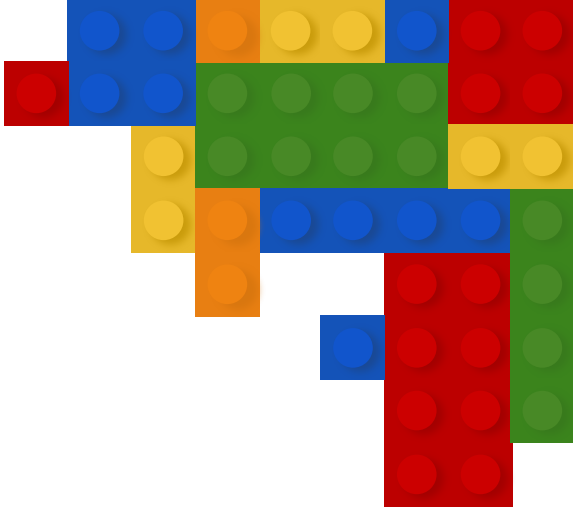


Consumer 2

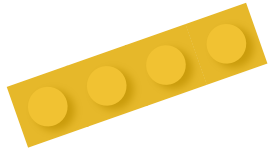


Consumer 3

Consumer 4





Break Time

And a Summary

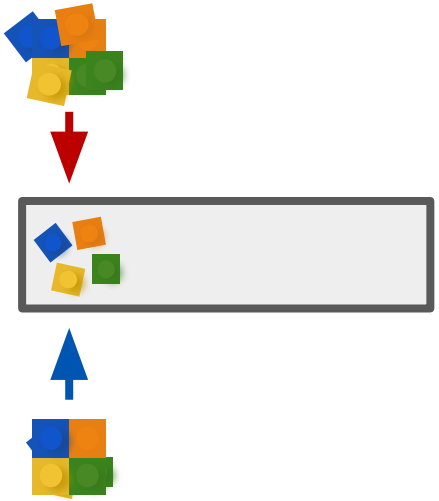
dfine@confluent.io

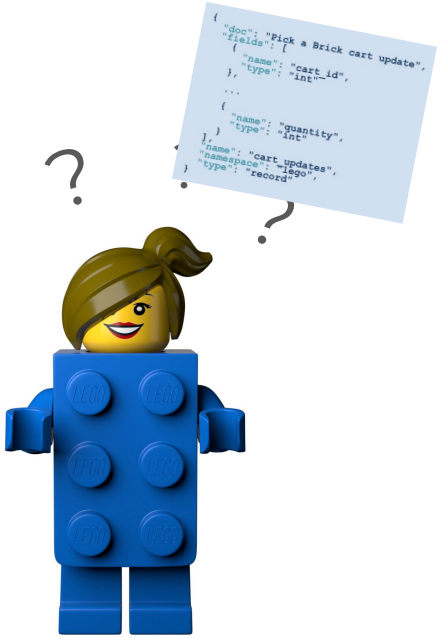
@TheDanicaFine

linkedin.com/in/danica-fine/

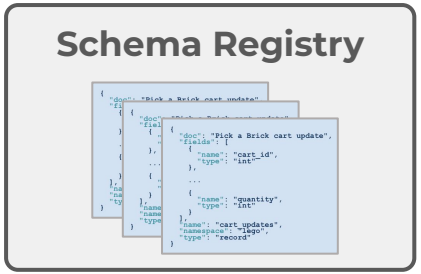
dfine@confluent.io
@TheDanicaFine
linkedin.com/in/danica-fine/

Producer

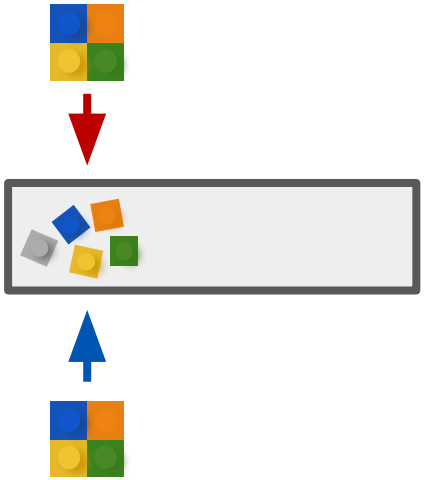


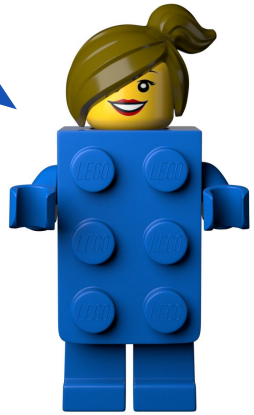


Consumer



Producer





Consumer

Kafka Connect

- A framework for integrating data
- 100+ data sources and sinks
- Low- to no-code option
- Built on consumers and producers

Stream Processing

In and Around Kafka

dfine@confluent.io

@TheDanicaFine

linkedin.com/in/danica-fine/

Processing within Kafka

- Consumer/Producer API
  - Lowest level
  - Consume – Process – Produce
  - Manually define state/fault-tolerance
- Kafka Streams
  - Java library for stream processing
  - Built-in state handling and failover
- ksqlDB
  - SQL syntax
  - Kafka Streams under the hood
  - Cloud based offering

```
subscribe(), poll(), send(),
flush(), beginTransaction(), …
```

```
KStream, KTable,
filter(), map(), flatMap(),
join(), aggregate(), …
```

```
CREATE STREAM, CREATE TABLE,
SELECT, JOIN, GROUP BY, SUM, …
```

Flink

APACHE Spark™

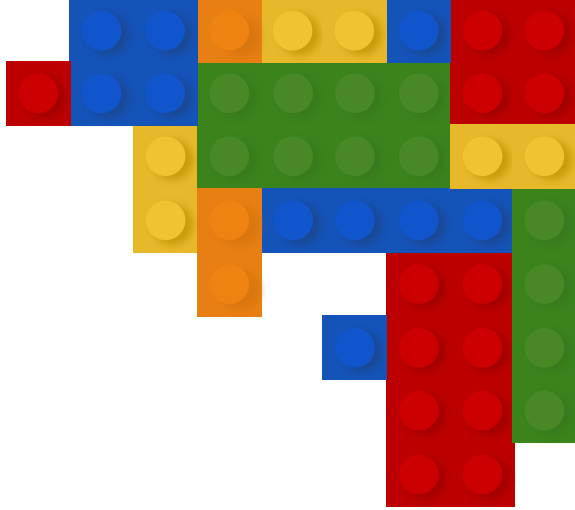
Materialize

bytewax

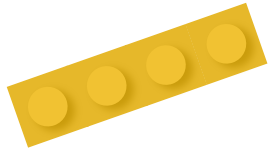
Benthos

FAUST

samza





Now what?

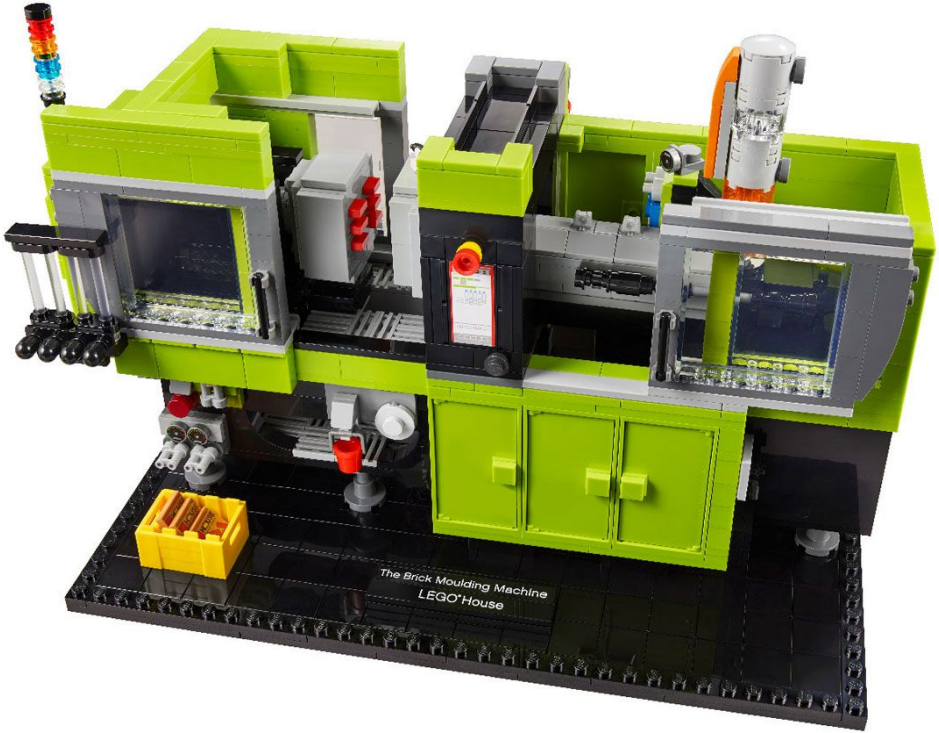
Use Cases and Inspiration

dfine@confluent.io

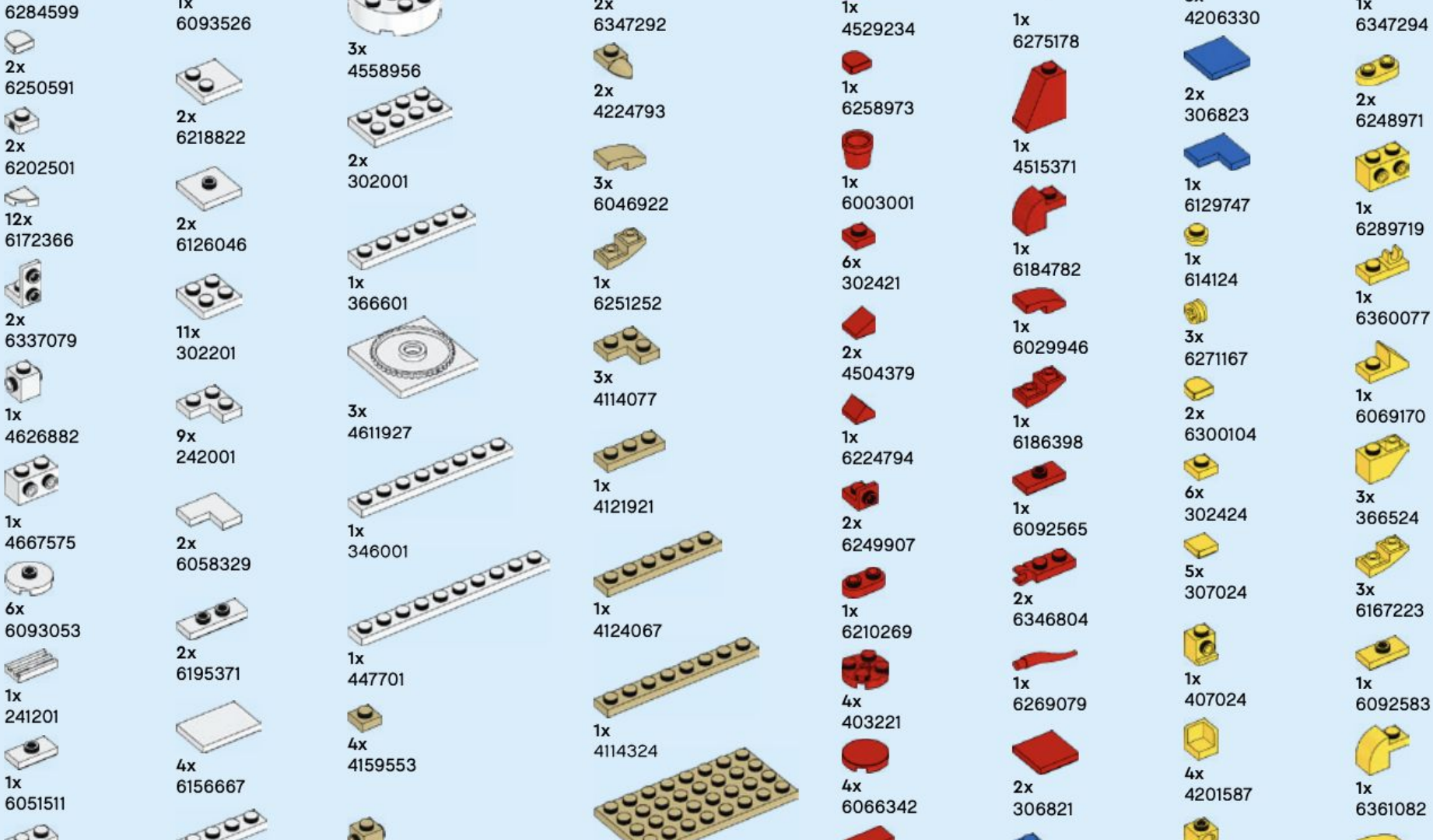
@TheDanicaFine

linkedin.com/in/danica-fine/

dfine@confluent.io
@TheDanicaFine
linkedin.com/in/danica-fine/

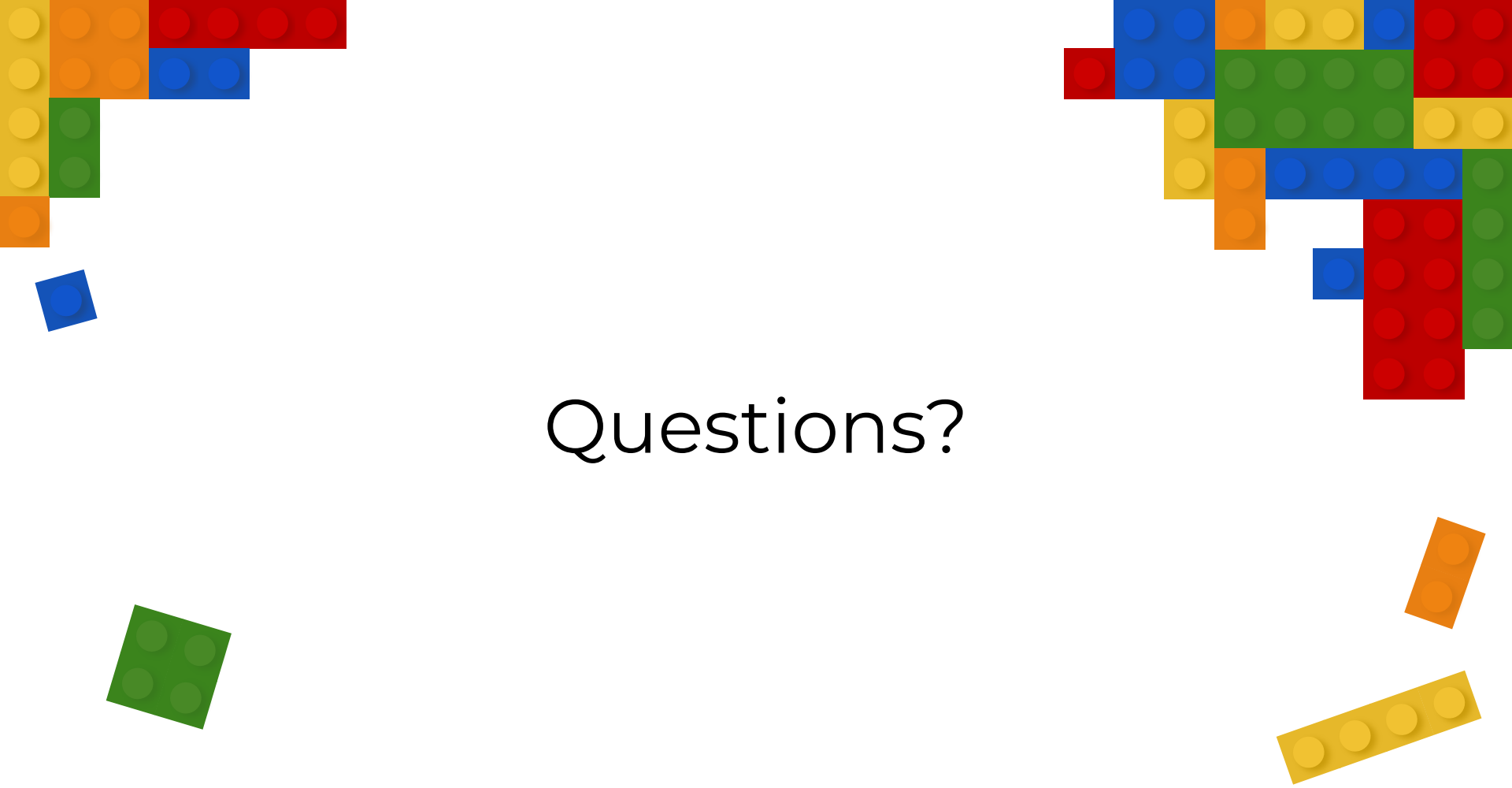


dfine@confluent.io
@TheDanicaFine
linkedin.com/in/danica-fine/

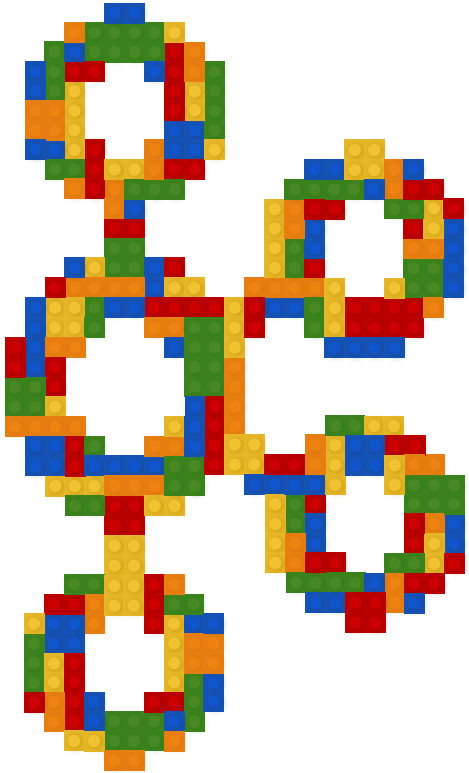

6284599
2x
6250591
2x
6202501
12x
6172366
2x
6337079
1x
4626882
1x
4667575
6x
6093053
1x
241201
1x
6051511
1x
6093526
2x
6218822
2x
6126046
11x
302201
9x
242001
2x
6058329
2x
6195371
4x
6156667
3x
4558956
2x
302001
1x
366601
3x
4611927
1x
346001
1x
447701
4x
4159553
2x
6347292
2x
4224793
3x
6046922
1x
6251252
3x
4114077
1x
4121921
1x
4124067
1x
4114324
1x
4529234
1x
6258973
1x
6003001
6x
302421
2x
4504379
1x
6224794
2x
6249907
1x
6210269
4x
403221
4x
6066342
1x
6275178
1x
4515371
1x
6184782
1x
6029946
1x
6186398
1x
6092565
2x
6346804
1x
6269079
2x
306821
4206330
2x
306823
1x
6129747
1x
614124
3x
6271167
2x
6300104
6x
302424
5x
307024
1x
407024
4x
4201587
1x
6347294
2x
6248971
1x
6289719
1x
6360077
1x
6069170
3x
366524
3x
6167223
1x
6092583
1x
6361082

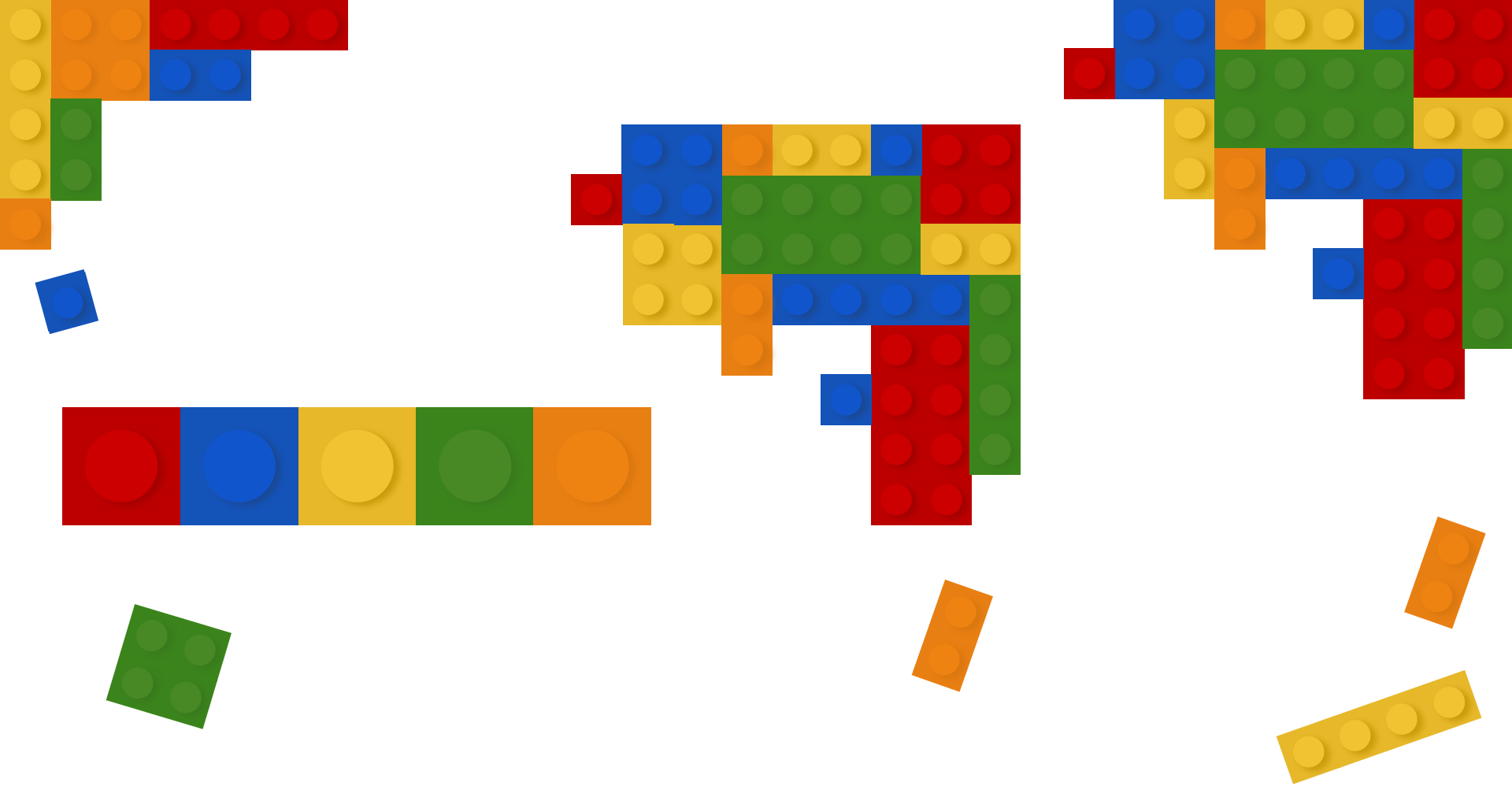
What will YOU build?

LinkTree Resources



dfine@confluent.io
@TheDanicaFine
linkedin.com/in/danica-fine/



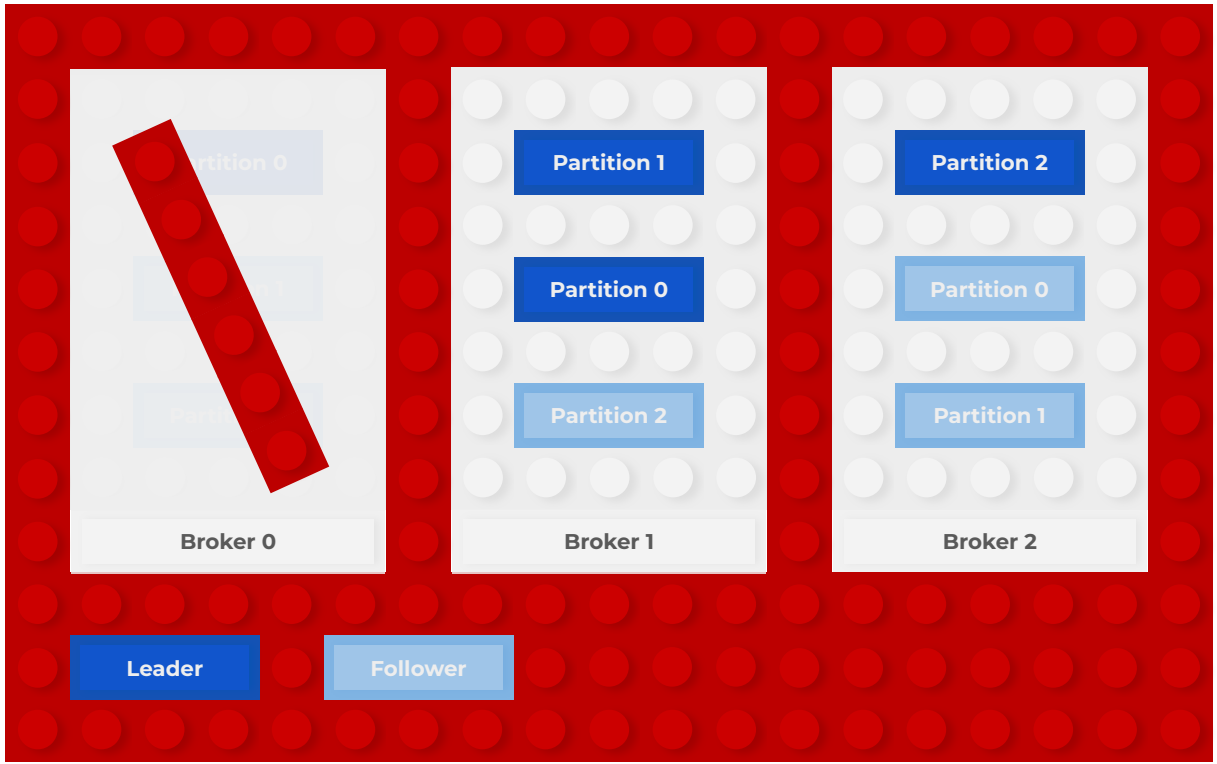


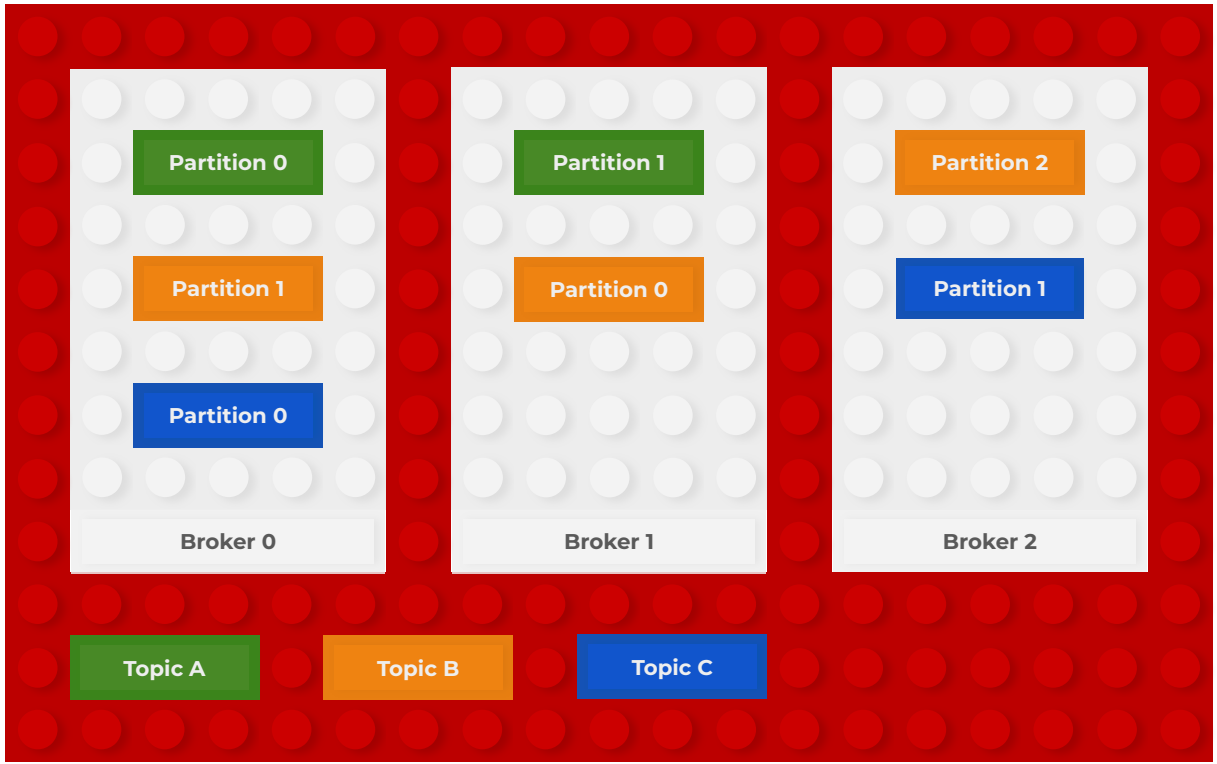
dfine@confluent.io

@TheDanicaFine

linkedin.com/in/danica-fine/

Replication




Partition 0
Partition 1
Partition 0
Broker 0
Partition 1
Partition 0
Broker 1
Partition 2
Partition 1
Broker 2
Topic A
Topic B
Topic C