Streaming
Why should I care?

Christian Trebing
Blue Yonder GmbH
@ctrebing

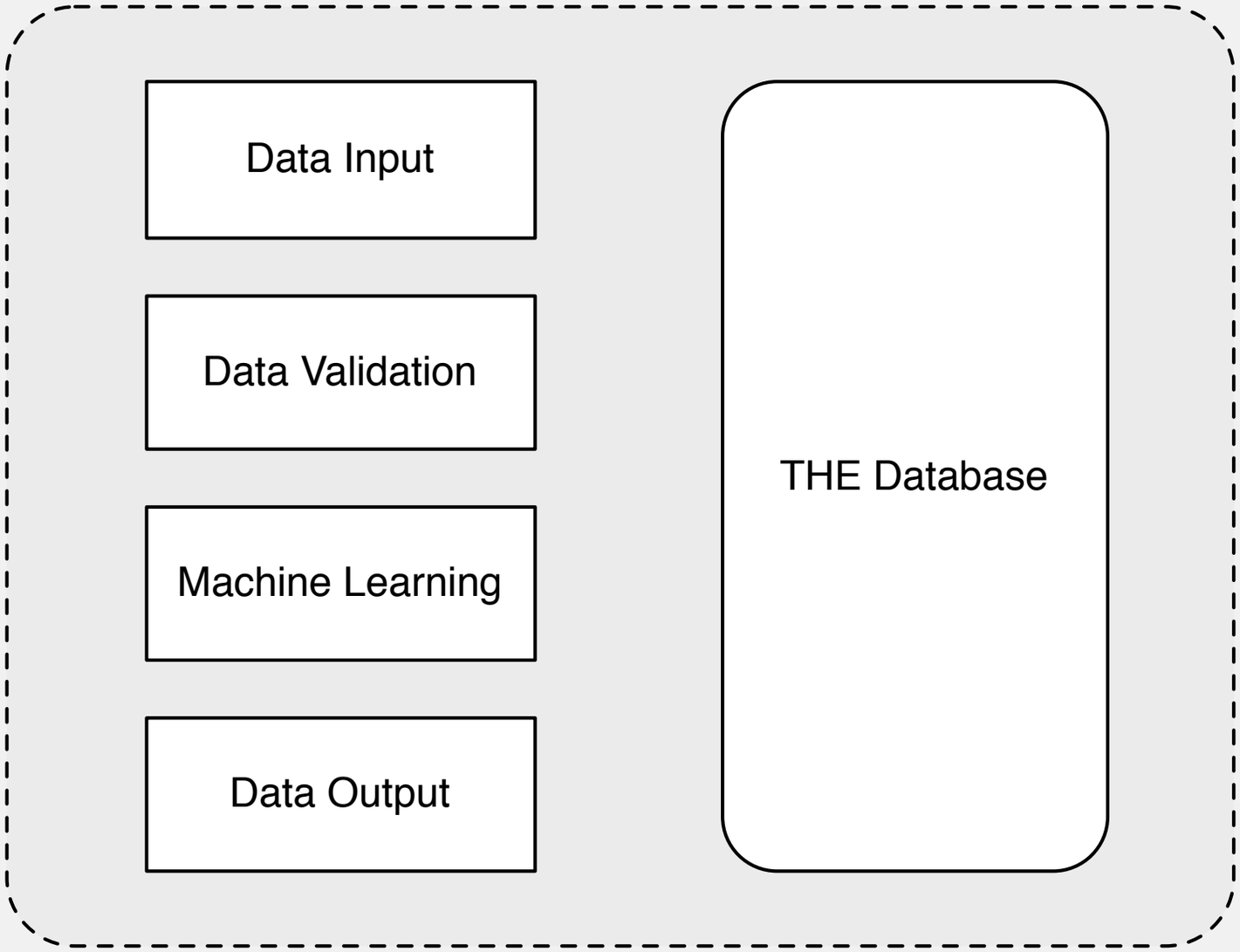
BlueYonder
Best decisions, delivered daily

Agenda

Motivation
Streaming Intro
Implementation
Challenges

BlueYonder
Best decisions, delivered daily

Data Processing - The Monolith



Pain

Several teams are developing this application

- Customer desperately wants new feature in machine learning
- But the data validation team is in the midst of refactoring their database structure (‚will be finished in two weeks‘)
- So you wait…

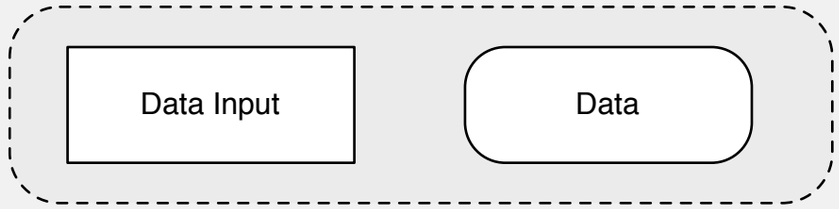
Could Microservices Help?

Great:

- No Dependency on single state
- Independent Development
- Independent Upgrades

Difficult:

- Too much data to transfer
- Too much data to store in each service



Are There Other Possibilities?

Streaming Intro

Databases and Streams

Same information in table and stream:

A=1	B=5	C=3	A=8	A=4	C=2	

A: 1

A: 1
B: 5

A: 1
B: 5
C: 3

A: 8
B: 5
C: 3

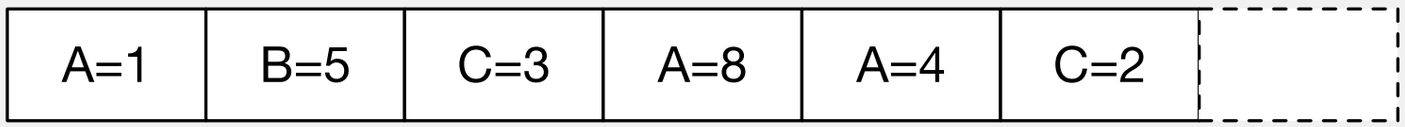
A: 4
B: 5
C: 3

A: 4
B: 5
C: 2

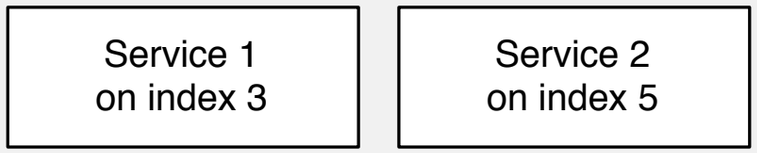
Why does it matter?

Different services can be in different states

- Each service can consume the stream in its own speed
- One service can be updated while the other runs



A: 1	A: 1 B: 5	A: 1 B: 5 C: 3	A: 8 B: 5 C: 3	A: 4 B: 5 C: 3	A: 4 B: 5 C: 2



Partitioned Streams

Sales stream, partitioned by location

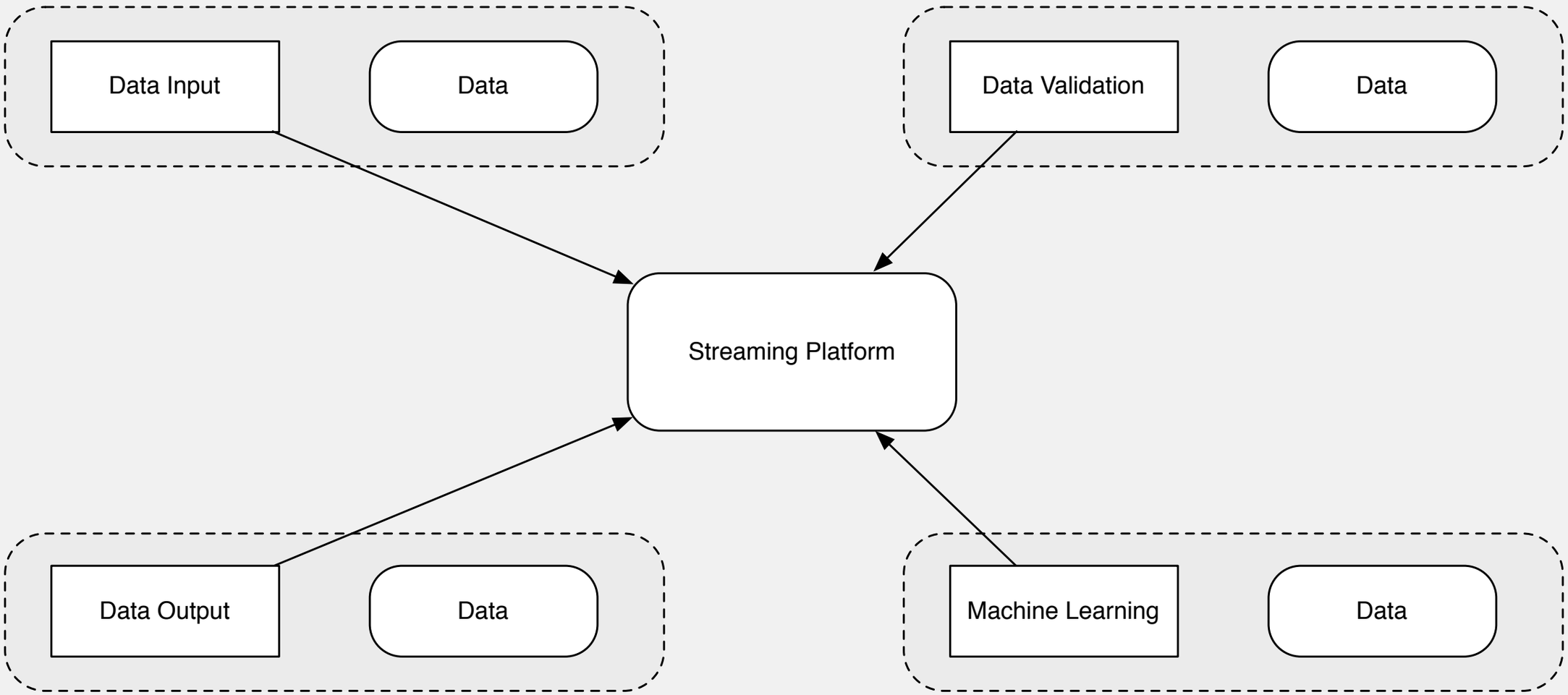
Each partition could be handled by diff. processors

location: Rimini product: Spaghetti sales_date: 2017-07-10 quantity: 5	**location: Rimini** product: Ravioli sales_date: 2017-07-10 quantity: 8	**location: Rimini** product: Pizza sales_date: 2017-07-11 quantity: 1	**location: Rimini** product: Spaghetti sales_date: 2017-07-11 quantity: 7

location: Bilbao product: Pizza sales_date: 2017-07-10 quantity: 3	**location: Bilbao** product: Ravioli sales_date: 2017-07-11 quantity: 9	**location: Bilbao** product: Spaghetti sales_date: 2017-07-11 quantity: 5

location: Karlsruhe product: Pizza sales_date: 2017-07-10 quantity: 8	**location: Karlsruhe** product: Ravioli sales_date: 2017-07-10 quantity: 3	**location: Karlsruhe** product: Ravioli sales_date: 2017-07-11 quantity: 7	**location: Karlsruhe** product: Spaghetti sales_date: 2017-07-11 quantity: 7

Data Processing
- With Streaming



What did we gain?

Independent Development

Independent Upgrade

Scalability

Did we throw out databases completely?

- Let’s see…

Is it magic?

No, it's a tradeoff:

- A database is so powerful: ACID guarantess, SQL language. You can do nearly everything
  - This comes at a price
    - Dependency on single state
    - Scaling is hard

So let's more think what we really need

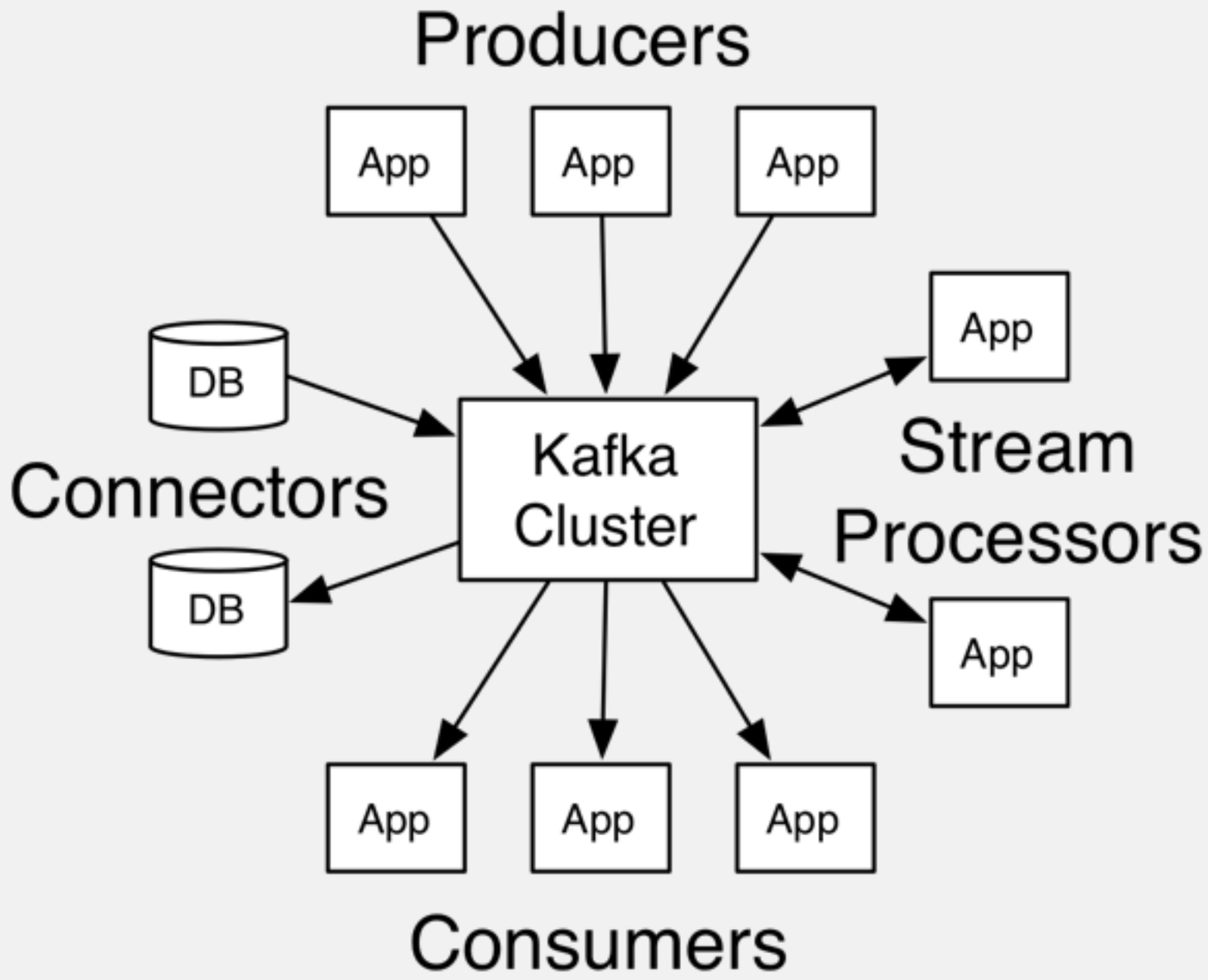
What do we loose?

Database	Stream
ACID	Ordering on stream partition
SQL Queries	Service is responsible of keeping its state

You have to decide whether you can live with that.

Implementation

Apache Kafka



Kafka Clients in Python

- pykafka, python-kafka, confluent-kafka-client
- Nice comparison has been done here:
  - http://activisiongamescience.github.io/2016/06/15/Kafka-Client-Benchmarking/
- Most performant currently is confluent-kafka-client
  - Uses the c library librdkafka

Producer

```
1  import json
2  from confluent_kafka import Producer
3
4  p = Producer({'bootstrap.servers': 'localhost:9092'})
5  data = {
6      'location': 'Rimini', 'product': 'Ravioli',
7      'sales_date': '2017-07-10', 'quantity': 5
8  }
9  p.produce('sales_input', json.dumps(data))
10 p.flush()
```

Consumer

```
from confluent_kafka import Consumer, KafkaError
import uuid

c = Consumer({'bootstrap.servers': 'localhost:9092', 'group.id': uuid.uuid1(),
              'default.topic.config': {'auto.offset.reset': 'smallest'}})
c.subscribe(['sales_input'])
running = True
while running:
    msg = c.poll()
    if not msg.error():
        print('Received message: %s' % msg.value().decode('utf-8'))
    elif msg.error().code() != KafkaError._PARTITION_EOF:
        print(msg.error())
        running = False
c.close()
```

Apache Avro

Data Serialization, Enabling Schema Evolution

Clearly defined schema with:

- Schema evolution
- Schema registry

Writer's schema

Data Type	Field Name
string	location
string	product
string	sales_date
int	quantity

Reader's schema

Data Type	Field Name
string	location
string	product
string	sales_date
int	quantity
int, default=0	delivery_id

Avro Schema

```
import json
from confluent_kafka import avro

value_schema = avro.loads(json.dumps({
    "name": "sales_input_value",
    "type": "record",
    "fields": [
        {"name": "location", "type": ["string"]},
        {"name": "product", "type": ["string"]},
        {"name": "sales_date", "type": ["string"]},
        {"name": "quantity", "type": ["int"]}
    ]
}))
```

AvroProducer

```
from confluent_kafka import avro
from confluent_kafka.avro import AvroProducer
from schema import key_schema, value_schema

avroProducer = AvroProducer({
    'bootstrap.servers': 'localhost:9092',
    'schema.registry.url': 'http://localhost:8081'},
    default_key_schema=key_schema, default_value_schema=value_schema)

key = {'location': 'Rimini'}
value = {
    'location': 'Rimini',
    'product': 'Ravioli',
    'sales_date': '2017-07-10',
    'quantity': 5
}
avroProducer.produce(topic='sales_input', value=value, key=key)
avroProducer.flush()
```

AvroConsumer

```
import uuid
from confluent_kafka import KafkaError
from confluent_kafka.avro import AvroConsumer
from confluent_kafka.avro.serializer import SerializerError

c = AvroConsumer({
    'bootstrap.servers': 'localhost:9092', 'group.id': uuid.uuid1(),
    'schema.registry.url': 'http://localhost:8081',
    'default.topic.config': {'auto.offset.reset': 'smallest'}})
c.subscribe(['sales_input'])
running = True
while running:
    try:
        msg = c.poll(10)
        if msg:
            if not msg.error():
                print(msg.value())
            elif msg.error().code() != KafkaError._PARTITION_EOF:
                print(msg.error())
                running = False
    except SerializerError as e:
        print("Message deserialization failed for %s: %s" % (msg, e))
        running = False

c.close()
```

Example: Data Validation

Separate valid and invalid sales records

```
consumer.subscribe(['sales_input'])

while True:
    msg = self.consumer.poll(1)
    if msg:
        key = msg.key()
        value = msg.value()
        if sales_record_valid(value):
            producer.produce(topic='sales_validated', value=value, key=key)
        else:
            producer.produce(topic='sales_error', value=value, key=key)
```

Additional Processors

Need to evolve your application:

- Add processors for evaluation topics
- Try new variant of validation logic

database

- remember processing state, same for each processor

streaming

- offset in each processor, can work independently

Challenge
- Machine Learning Still Batch

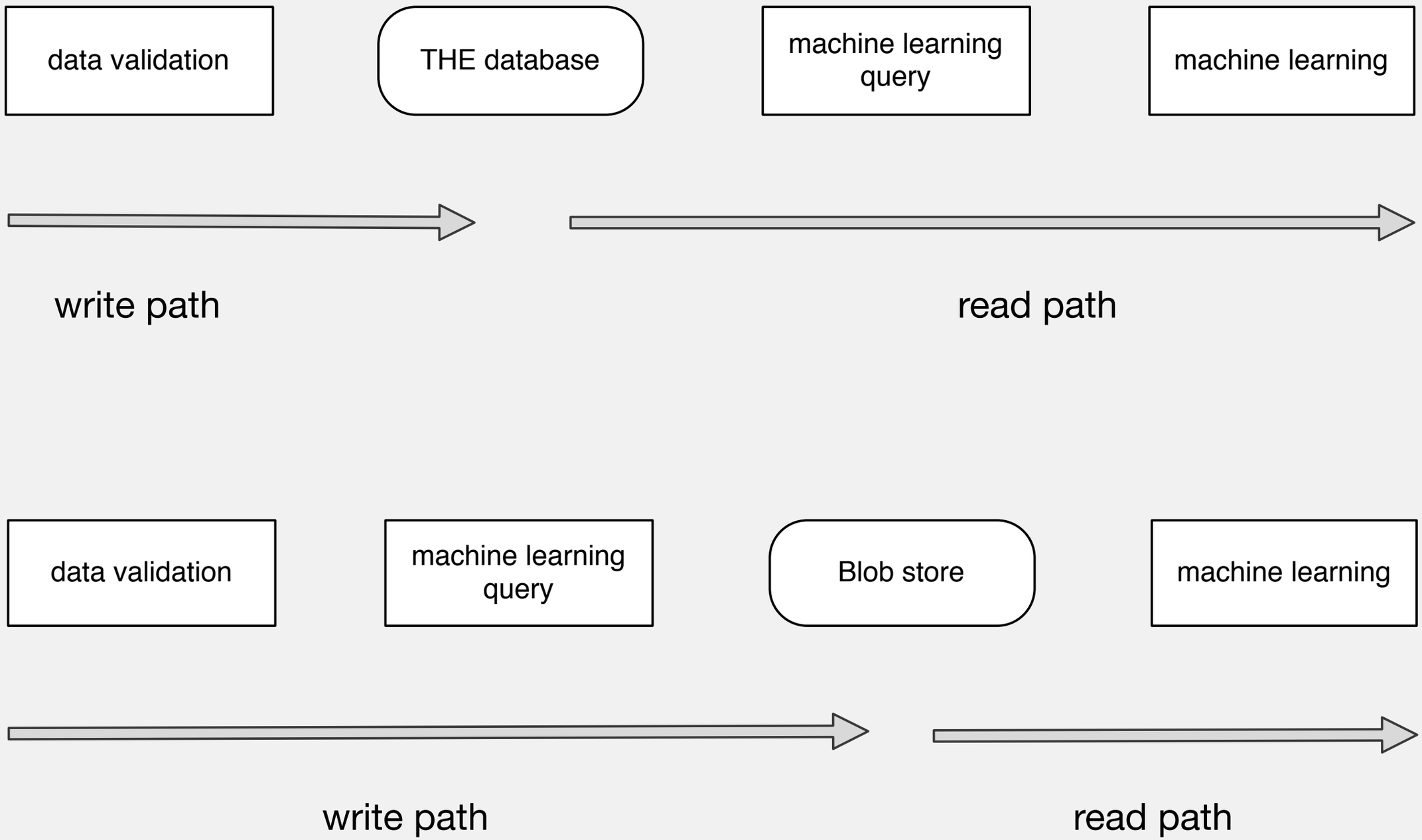
How to get the input data?

Remember: There is no possibility to query a stream.
Somewhere all this data needs to be. Options:

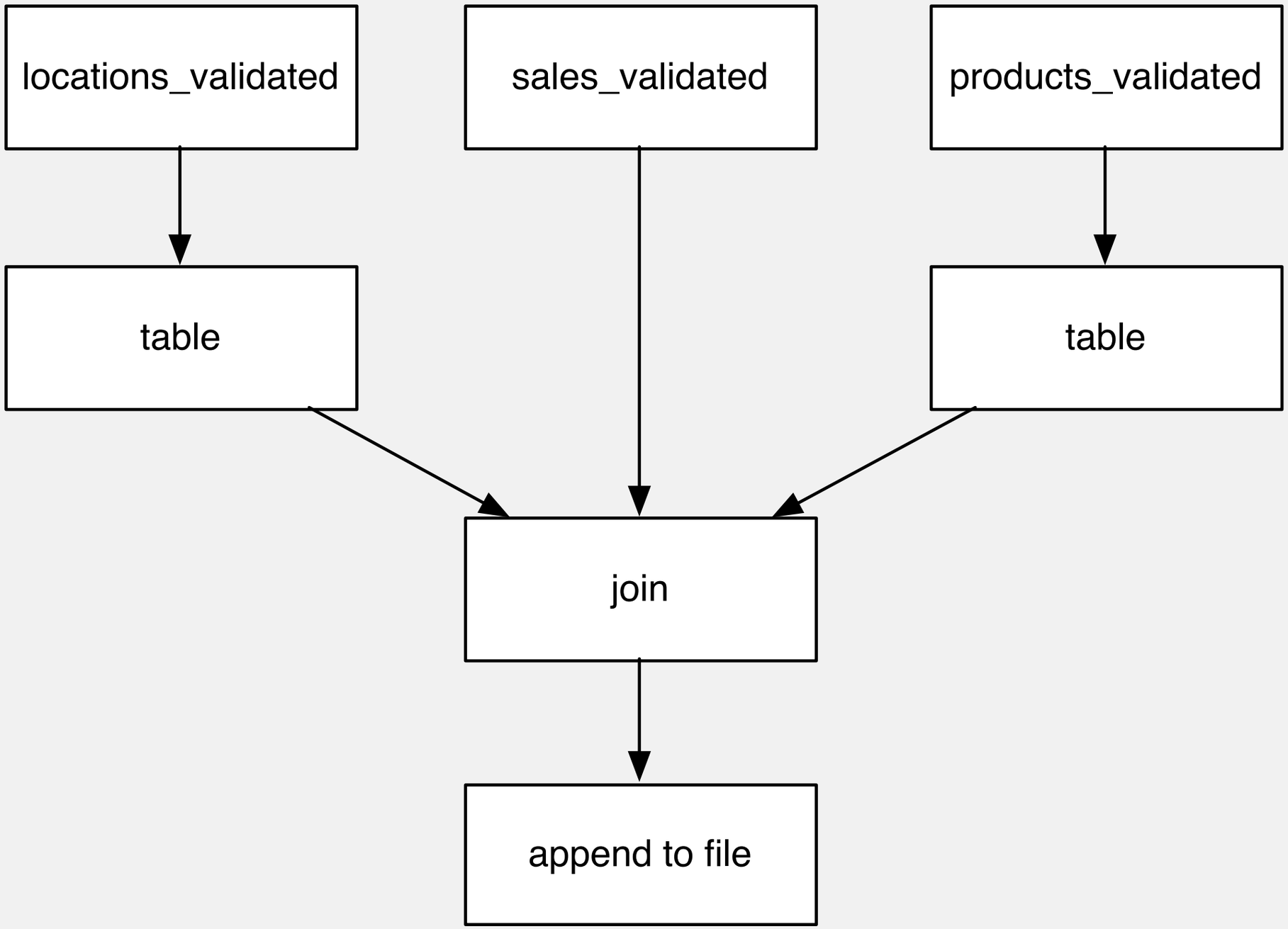
- Memory of the service
- Serving database
- Blob store

Yes, that's duplication. We'll have to live with it.

Write Path / Read Path



Machine Learning Input Data



Challenge
- State in Processors

State - Nightmare of every distributed systems engineer

Streaming: Data just rushes through

Why do we need state?

- Time window processing
- Data you want to join with

Formerly, the database did it for you

State - Some Challenges?

Failure of a processor

Scaling

State in Stream Processors - Possible Solutions

- Just keep in memory. Reprocess stream to warm up
- Each processor to keep its own db
- Save condensed in stream
- Get it from other service

Frameworks exist in other languages:

- for example: Kafka Streams, Apache Samza

Up to yesterday: none in python. Then heard about

https://github.com/wintoncode/winton-kafka-streams

Summary

- You have more options for your data processing applications than you might have thought
- As always, there are some tradeoffs
- You know the challenges

BlueYonder
Best decisions, delivered daily

Questions?

BlueYonder
Best decisions, delivered daily

Blue Yonder
Best decisions, delivered daily

Blue Yonder GmbH
Ohiostraße 8
76149 Karlsruhe
Germany
+49 721 383117 0

Blue Yonder Software Limited
19 Eastbourne Terrace
London, W2 6LG
United Kingdom
+44 20 3626 0360

Blue Yonder Analytics, Inc.
5048 Tennyson Parkway
Suite 250
Plano, Texas 75024
USA

BlueYonder
Best decisions, delivered daily