Top 10 data engineering mistakes

Berlin Buzzwords, 2018-06-11

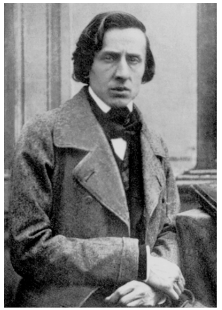
Lars Albertsson
www.mapflat.com, www.mimeria.com

Why this talk?

- Sharing what I have learnt. Please do the same.
- Mistakes mentioned are directly or indirectly experienced.
- "Do this" / "Don't do that" == "Based on what I have seen, I have come to the conclusion that ..."

Learning practices

Mastering practices



Bending practices

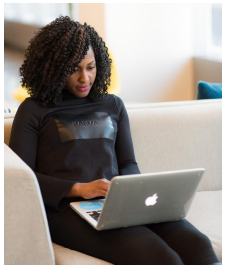
Who’s talking?

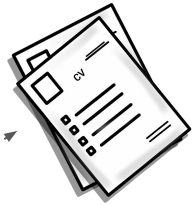
- KTH-PDC Center for High Performance Computing (MSc thesis)
- Swedish Institute of Computer Science (distributed system test+debug tools)
- Sun Microsystems (building very large machines)
- Google (Hangouts, productivity)
- Recorded Future (natural language processing startup)
- Cinnober Financial Tech. (trading systems)
- Spotify (data processing & modelling)
- Schibsted Media Group (data processing & modelling)

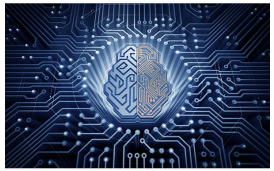
- Mapflat - independent data engineering consultant
  - ~15 clients: Spotify, 3 banks, 3 conglomerates, 4 startups, 5 *tech, misc
- Founder @ Mimeria - data-value-as-a-service

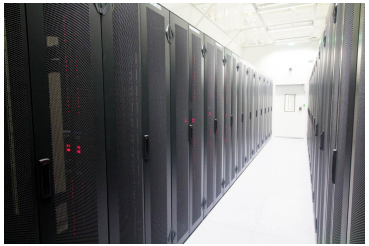
Mistake == obstacle towards end goal

- What is the goal?





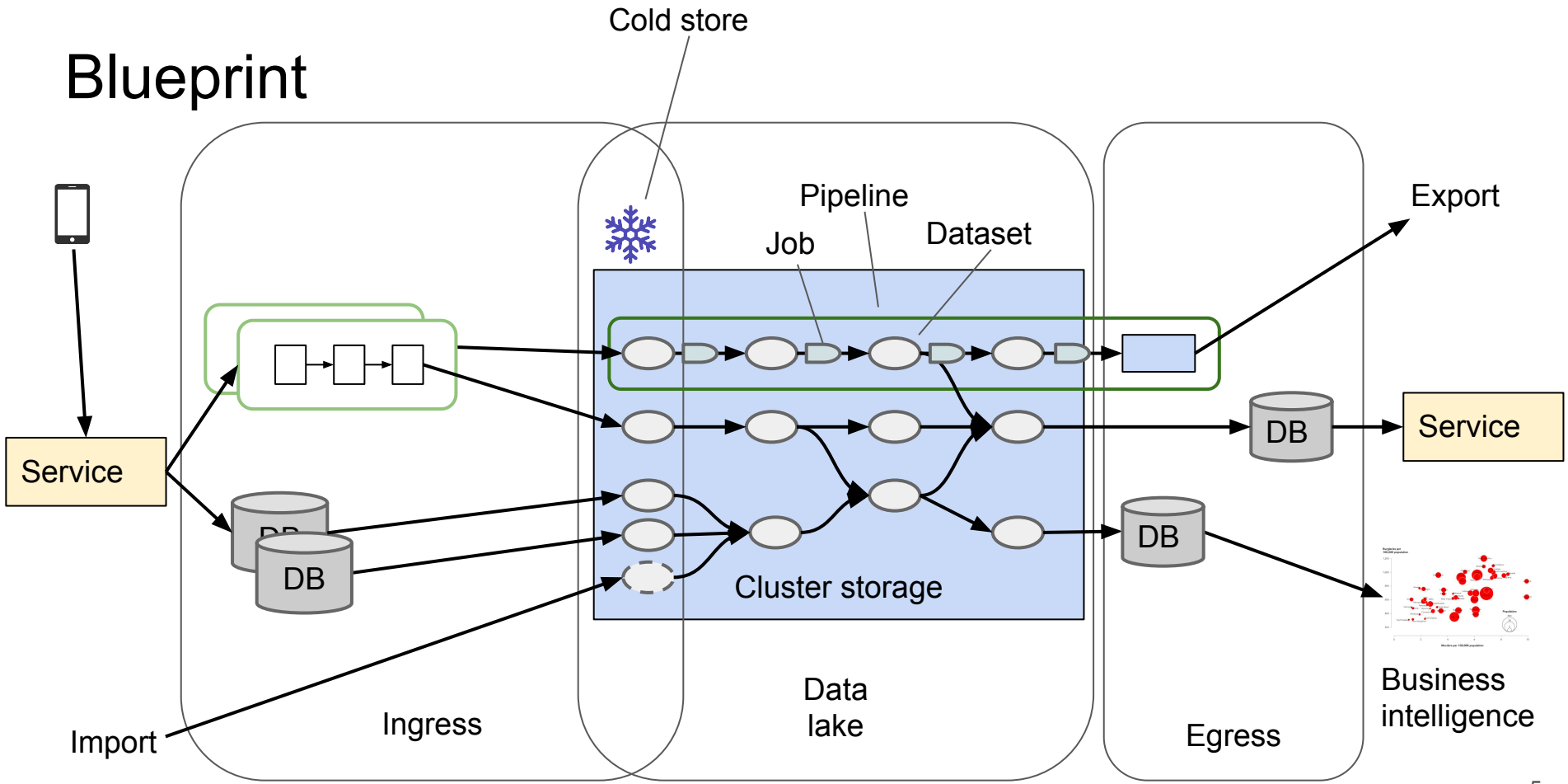




Nick Heudecker @nheudecker · 9 Nov 2017
We were too conservative. The failure rate is closer to 85%. And the problem isn't technology.

Ronda Swaney @RondaSwaney
Gartner says in 2017, 60% of #bigdata projects will fail to move past preliminary stages. Does that mean big data has been a failure? No. But it does mean we have to separate hype from reality. Read my latest written for @Hortonworks. hortonworks.com/article/what-h… via @Hortonworks

Blueprint


Cold store
Pipeline
Job
Dataset
Export
Service
DB
Service
DB
DB
Cluster storage
Import
Ingress
Data
lake
Egress
Business
intelligence

A first pipeline

Cron schedule: 10 * * * *

```
#! /bin/bash

y=$(date +%Y)
m=$(date +%m)
d=$(date +%d)
h=$(date +%h)
page_view="/cold/red/page/v1/$y/$m/$d/$h"
order="/prod/red/order/v1/$y/$m/$d"
order_hour="$order/$h"
session="/prod/red/session/v1/$y/$m/$d"
kpi="/prod/green/kpi/v1/$y/$m/$d"
ss="spark-submit --master spark://spark.acme.ai:7077"

$ss --class ai.acme.FilterOrders kpi.jar $page_view $order_hour
if [ $h == 23 ]; then
  $ss --class ai.acme.FormSession kpi.jar $order $session
  $ss --class ai.acme.Kpi kpi.jar $session $kpi
fi
```

- Input: Page views from www (hourly)
- Output: key performance indexes (daily)
  - Number of orders
  - Web session length

A first pipeline

Cron schedule: 10 * * * *

```
#! /bin/bash

y=$(date +%Y)
m=$(date +%m)
d=$(date +%d)
h=$(date +%h)
page_view="/cold/red/page/v1/$y/$m/$d/$h"
order="/prod/red/order/v1/$y/$m/$d"
order_hour="$order/$h"
session="/prod/red/session/v1/$y/$m/$d"
kpi="/prod/green/kpi/v1/$y/$m/$d"
ss="spark-submit --master spark://spark.acme.ai:7077"

$ss --class ai.acme.FilterOrders kpi.jar $page_view $order_hour
if [ $h == 23 ]; then
  $ss --class ai.acme.FormSession kpi.jar $order $session
  $ss --class ai.acme.Kpi kpi.jar $session $kpi
fi
```

- Works great day 1,2,3,4

Cron schedule: 15 1 * * *

```
#! /bin/bash

y=$(date +%Y)
m=$(date +%m)
d=$(date +%d)
order="/prod/red/order/v1/$y/$m/$d"
campaign="/cold/green/campaign/v1/$y/$m/$d"
impact="/cold/green/impact/v1/$y/$m/$d"
session="/prod/red/session/v1/$y/$m/$d"

ss="spark-submit --master spark://spark.acme.ai:7077"

$ss --class ai.acme.CampaignImpact kpi.jar $order $campaign \
  $session $impact
```

A brittle pipeline

Cron schedule: 10 * * * *

```
#! /bin/bash

y=$(date +%Y)
m=$(date +%m)
d=$(date +%d)
h=$(date +%h)
page_view="/cold/red/page/v1/$y/$m/$d/$h"
order="/prod/red/order/v1/$y/$m/$d"
order_hour="$order/$h"
session="/prod/red/session/v1/$y/$m/$d"
kpi="/prod/green/kpi/v1/$y/$m/$d"
ss="spark-submit --master spark://spark.acme.ai:7077"

$ss --class ai.acme.FilterOrders kpi.jar $page_view $order_hour
if [ $h == 23 ]; then
  $ss --class ai.acme.FormSession kpi.jar $order $session
  $ss --class ai.acme.Kpi kpi.jar $session $kpi
fi
```

Cron schedule: 15 1 * * *

```
#! /bin/bash

y=$(date +%Y)
m=$(date +%m)
d=$(date +%d)
order="/prod/red/order/v1/$y/$m/$d"
campaign="/cold/green/campaign/v1/$y/$m/$d"
impact="/cold/green/impact/v1/$y/$m/$d"
session="/prod/red/session/v1/$y/$m/$d"

ss="spark-submit --master spark://spark.acme.ai:7077"

$ss --class ai.acme.CampaignImpact kpi.jar $order $campaign \
  $session $impact
```

- Works great day 1,2,3,4
- Day 5: FormSession not done by 1:15

Job fails

A corrupt pipeline

Cron schedule: 10 * * * *

```
#! /bin/bash

y=$(date +%Y)
m=$(date +%m)
d=$(date +%d)
h=$(date +%h)
page_view="/cold/red/page/v1/$y/$m/$d/$h"
order="/prod/red/order/v1/$y/$m/$d"
order_hour="$order/$h"
session="/prod/red/session/v1/$y/$m/$d"
kpi="/prod/green/kpi/v1/$y/$m/$d"
ss="spark-submit --master spark://spark.acme.ai:7077"

$ss --class ai.acme.FilterOrders kpi.jar $page_view $order_hour
if [ $h == 23 ]; then
  $ss --class ai.acme.FormSession kpi.jar $order $session
  $ss --class ai.acme.Kpi kpi.jar $session $kpi
fi
```

Cron schedule: 15 1 * * *

```
#! /bin/bash

y=$(date +%Y)
m=$(date +%m)
d=$(date +%d)
order="/prod/red/order/v1/$y/$m/$d"
campaign="/cold/green/campaign/v1/$y/$m/$d"
impact="/cold/green/impact/v1/$y/$m/$d"
session="/prod/red/session/v1/$y/$m/$d"

ss="spark-submit --master spark://spark.acme.ai:7077"

$ss --class ai.acme.CampaignImpact kpi.jar $order $campaign \
  $session $impact
```

- Works great day 1,2,3,4
- Day 5: FormSession not done by 1:15
- Day 8: FilterOrders fails for hour 6
- Day 13: Page views for hour 19 only partially done at 20:10

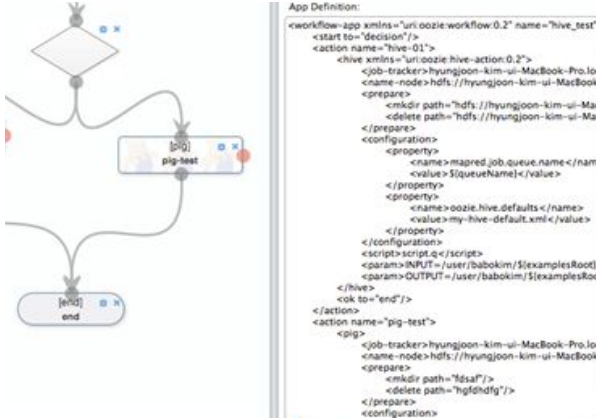
Silent data corruption

Mistake 1: No/weak workflow orchestration

- A *workflow orchestrator* manages the dependencies

- Weak orchestrators use graphical / XML dependency language
  - Not expressive enough

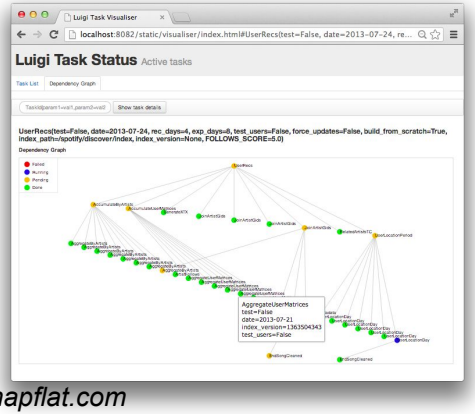
- Hadoop vendors + cloud vendors ship weak orchestrators
  - Except Google



Remedy 1: Python-DSL orchestrators

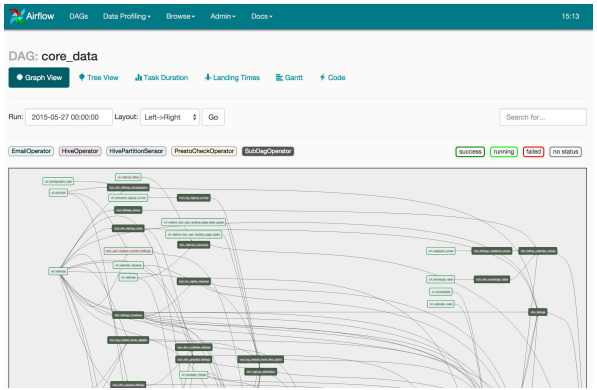
Luigi

- Simple, stable
- Easy to run
- Slim DataOps
- Does one thing well



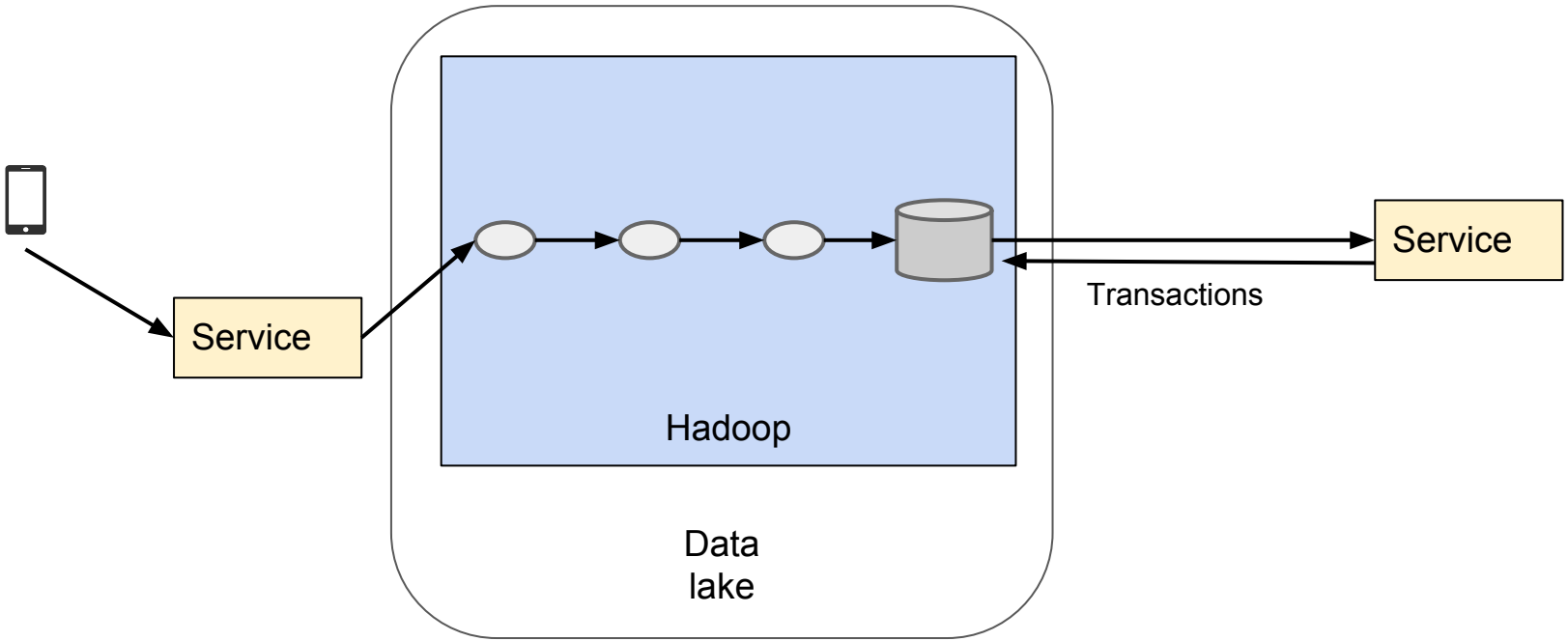
Airflow

- Good momentum
  - Google managed service
- More moving parts
- Complex DataOps
- Includes scheduler & monitoring UI



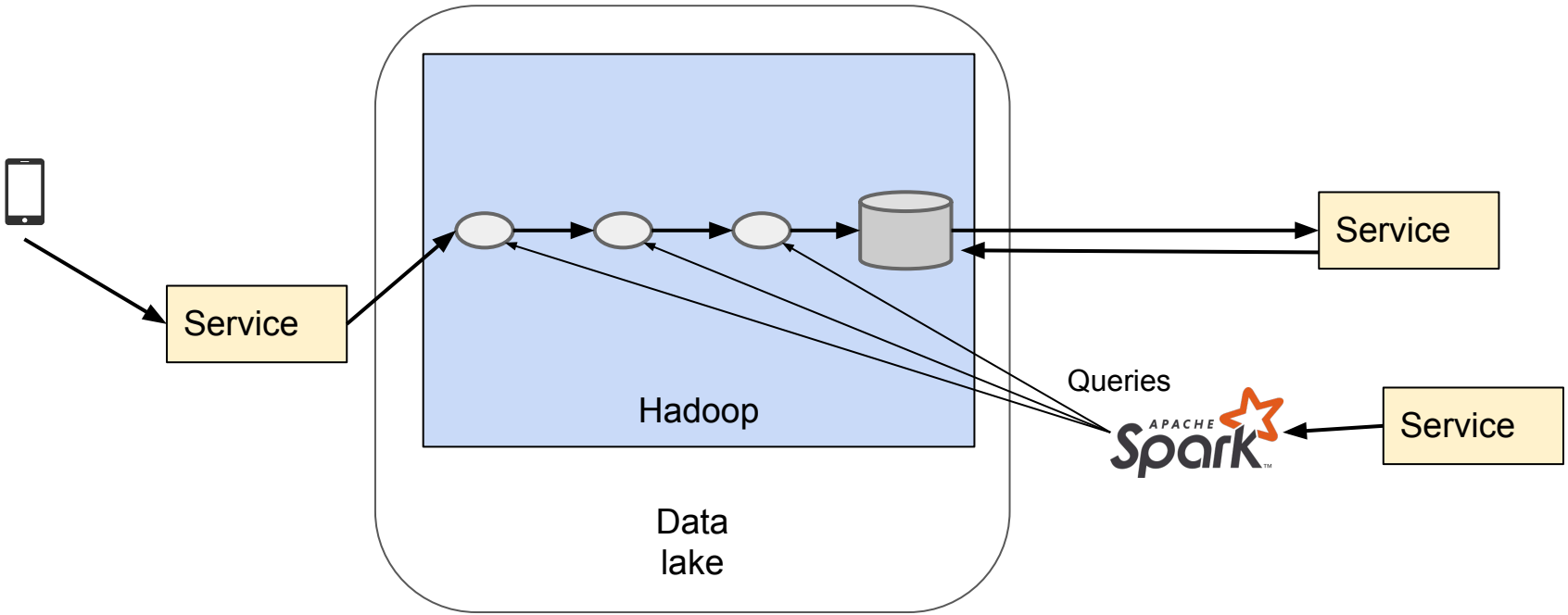
Hadoop has a SQL interface, right?

- Let's pretend it is an RDBMS



Hadoop for everything

- Or a NoSQL database



Mistake 2: New technology, same old patterns

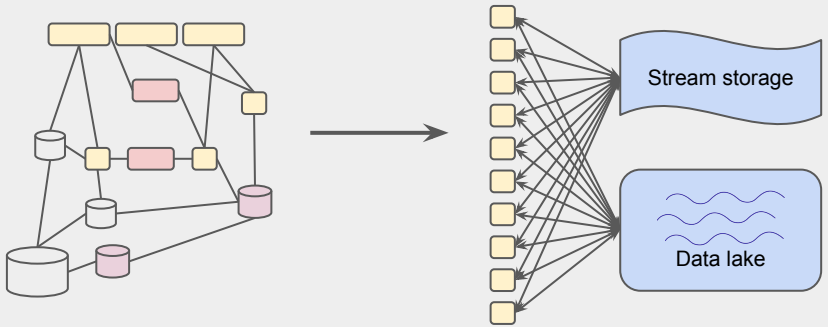
- Distributed technology is single-purpose
  - RDBMS == multi-purpose
- Hadoop + ecosystem = *offline*

Big data components + small data patterns → worst of both worlds

Remedy 2: Work patterns >> technology

- New technology is not the key to success
- Benefits of big data == collaboration and pipelines

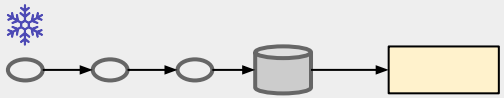
Data democratisation



Offline, forgiving environment

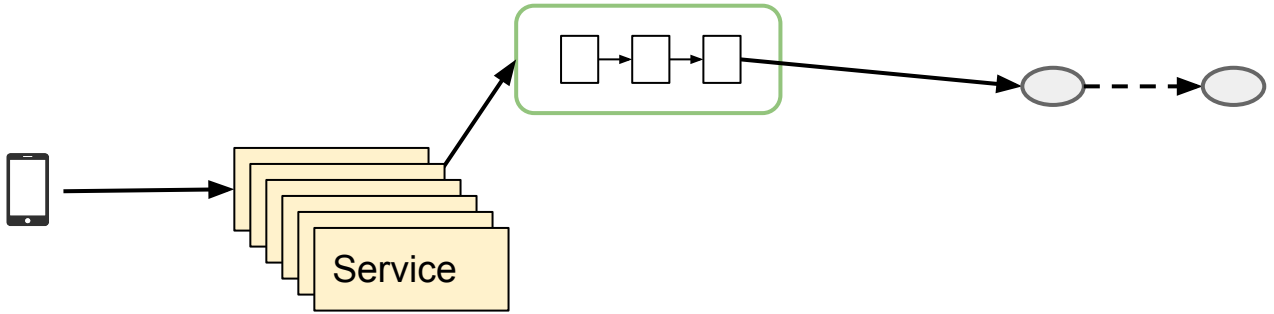
→

iteration speed



We need all the data

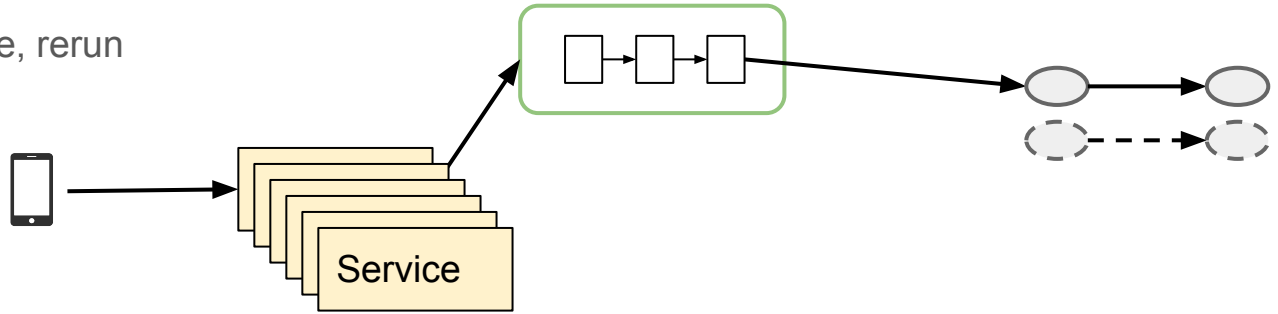
- Wait for all service nodes to send data
  - Then start processing



- For how long?
  - Dashboards?
  - Reports?

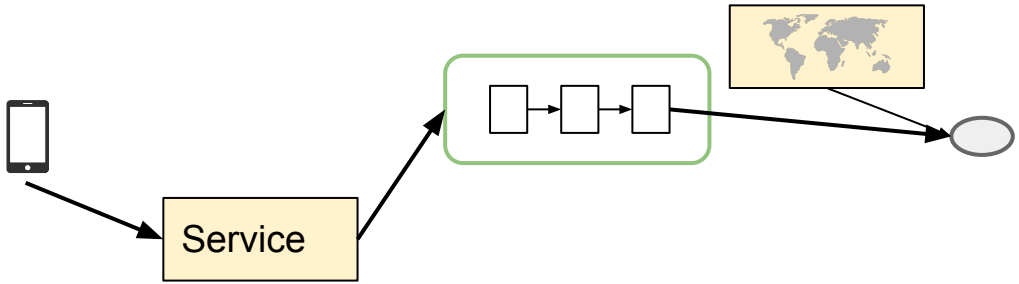
We need it now

- Start right away
- Hope we have most data
- If we get x% more, rerun



Let's improve the events

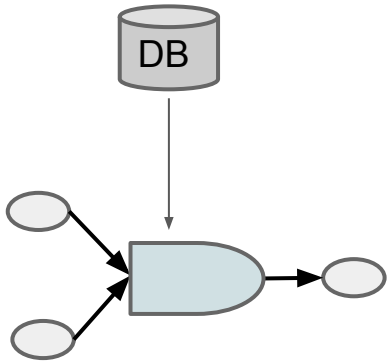
- We have an IP→ geo service
- Decorate incoming events!



- Works great until
  - service fails
  - has low quality geo data
  - has a bug
  - we overload it

Lazy data retrieval

- DB has data. We wants it.
- Just read it from production?



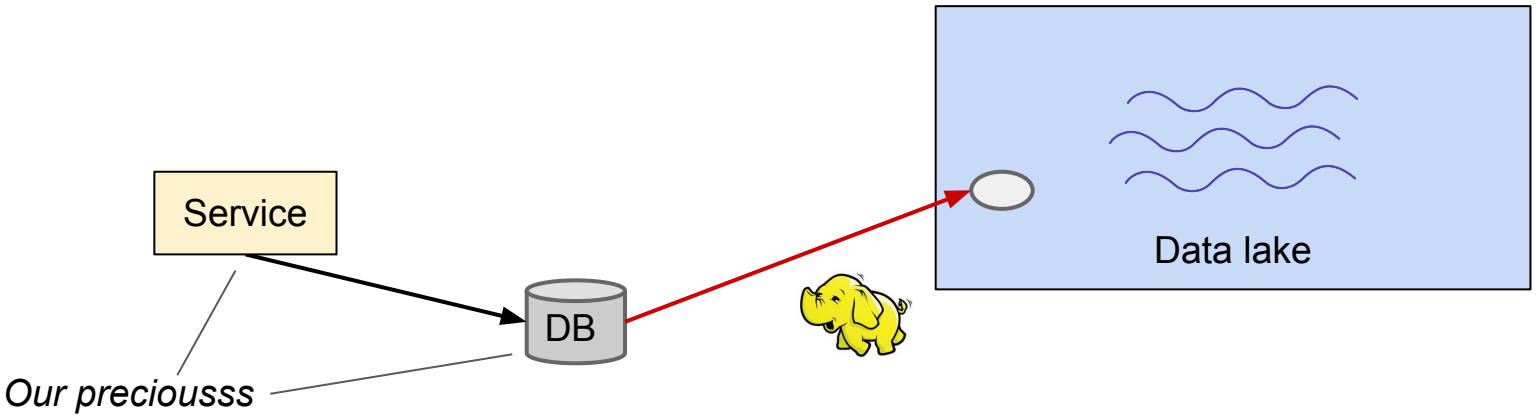
- Works until
  - You overload the database
  - You need to rerun the job later

Mistake 3: Deviate from functional principles

- Immutability
- Idempotency
  - Jobs must be able to rerun without harmful effects
- Reproducibility
  - Necessary for data science
  - Many exceptions - be conscious

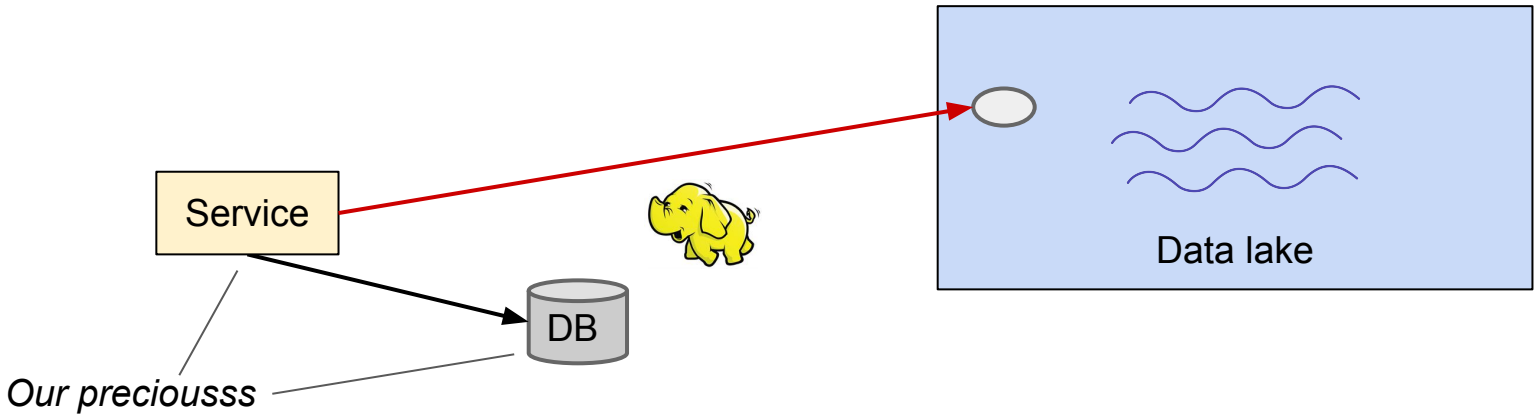
Knock on the back door

- Let's dump the DB to lake
- Spark/Sqoop can speak JDBC
- The more mappers, the merrier?



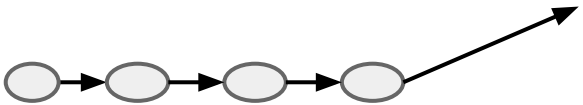
Knock on the front door

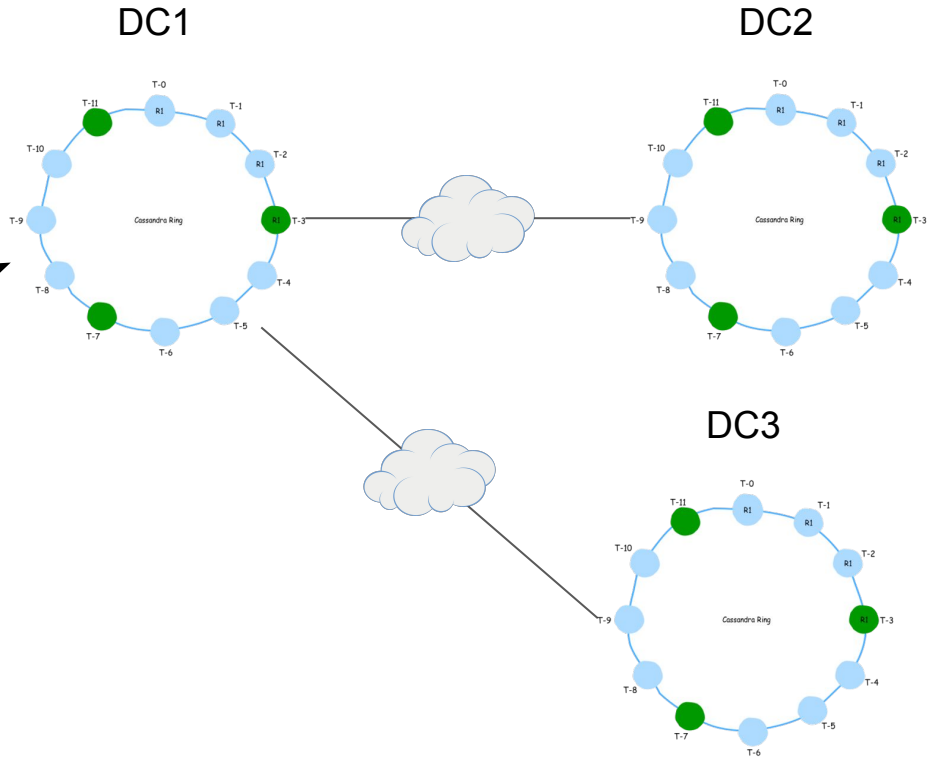
- We can dump from the REST interface
  - Unavailability by elephant



Here is a new ML model for you

- New ML algorithm is much better
- Let's broadcast new recommendation indexes





- Why is the Atlantic link saturated?

Credits: Tuplejump

Mistake 4: Weak online / offline separation

Risk = probability * impact

10000s of customers, imprecise feedback

Need low probability =>
Proactive prevention =>
Low ROI

Mistake 4: Weak online / offline separation

Risk = probability * impact

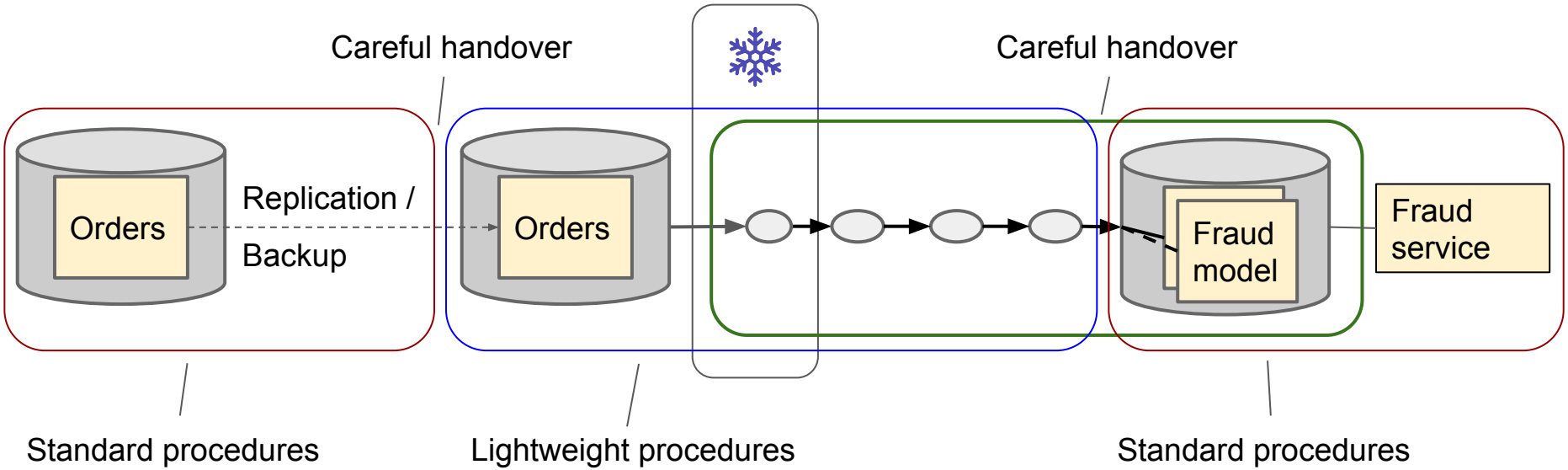
10000s of customers, imprecise feedback

Need low probability =>
Proactive prevention =>
Low ROI

10s of employees, precise feedback

Ok with medium probability =>
Reactive repair =>
High ROI

Remedy 4: Careful handover



Bring on the clusters

- "Our clients have petabytes, and they want to do machine learning."
  Ali Ghodsi, Databricks CEO

100s TBs/day

KBs/day

- Why not use scalable components?
  - In case we get 100M users?

Distributed processing is difficult

Difficult to reuse old code

Problem not parallelisable

Mysteriously slow nodes

Noisy neighbours

Immature frameworks

Excessive shuffling

More parameters to tune

Serialisation problems

Logs are distributed

Errors not reproducible

Data skew

More sysadmin

Limited speedup

Mistake 5: Premature scaling

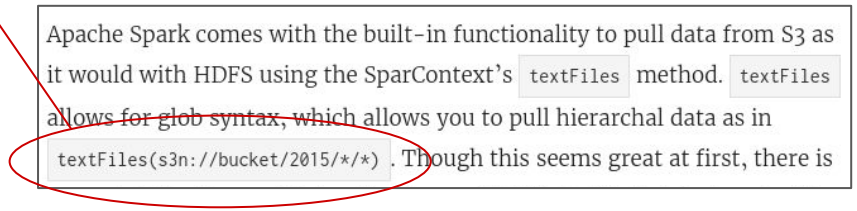
- Most companies have GBs, not PBs
  - Limited by human users
- Largest cloud instances ~4 TB memory.
  - Your data fits
  - Or: One time period of your data fits

Primary value of "Big Data" is not data size, but data sharing and data agility.

Avoid clusters until necessary - use single nodes. E.g. "spark-submit --master=local"

Slower by the day

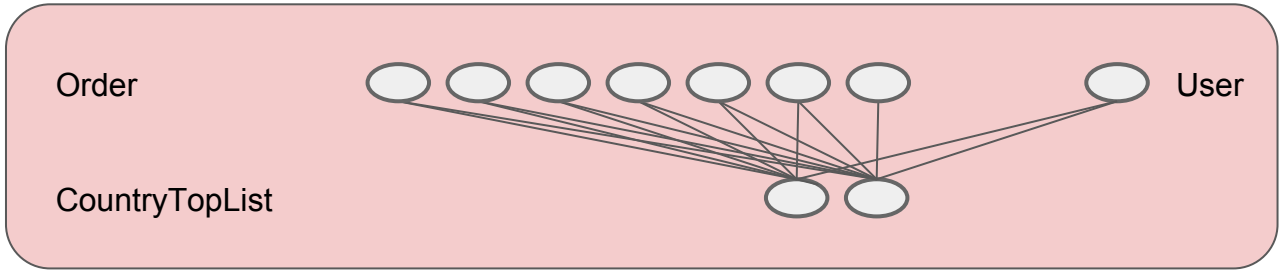
- What if we want top list of last 365 days, every day?


Apache Spark comes with the built-in functionality to pull data from S3 as it would with HDFS using the SparContext's `textFiles` method. `textFiles` allows for glob syntax, which allows you to pull hierarchal data as in `textFiles(s3n://bucket/2015/*/*)`. Though this seems great at first, there is

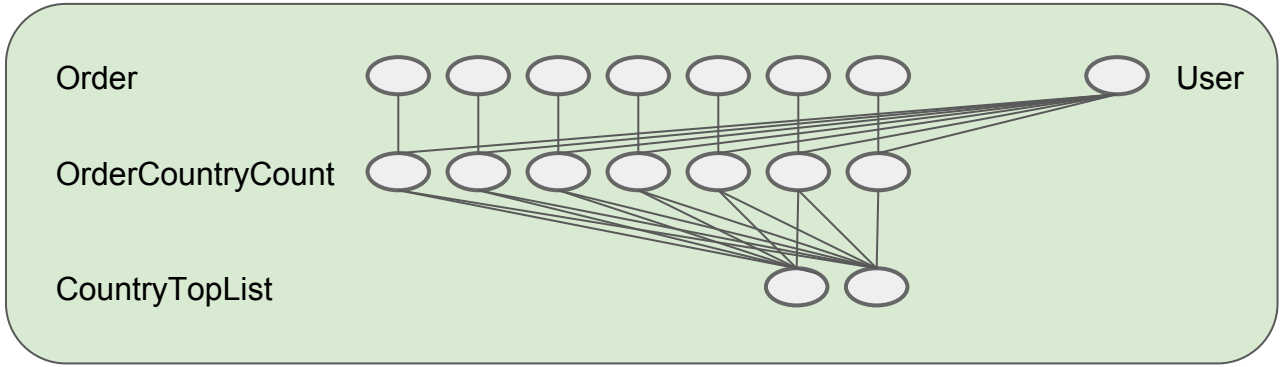
- Why is this team always using 20% of the cluster, every day?

Mistake 6: Read all the data

- Read it all:



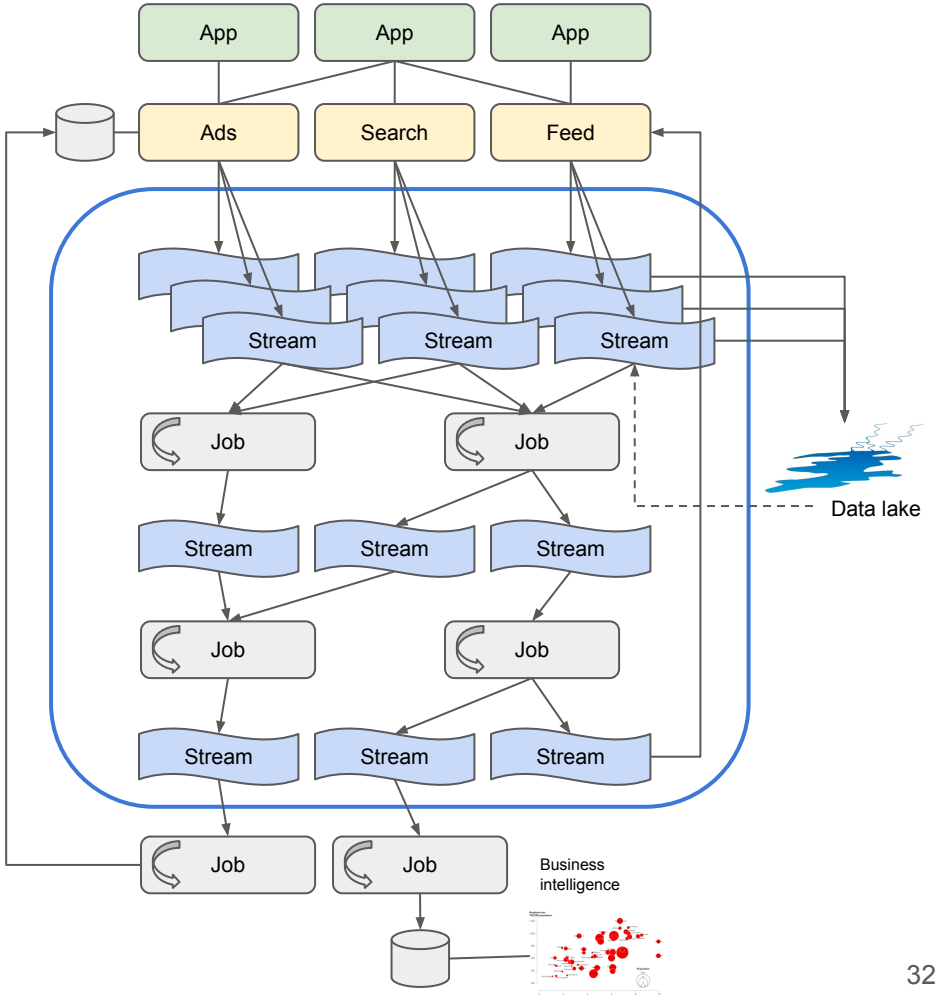
- Incremental counting:



Strive to parallelise workflows, not jobs

Context, streaming

- Data lake has a real-time cousin - the unified log
- Stream storage bus
  - Kafka / cloud service
- Pipelines of stream jobs



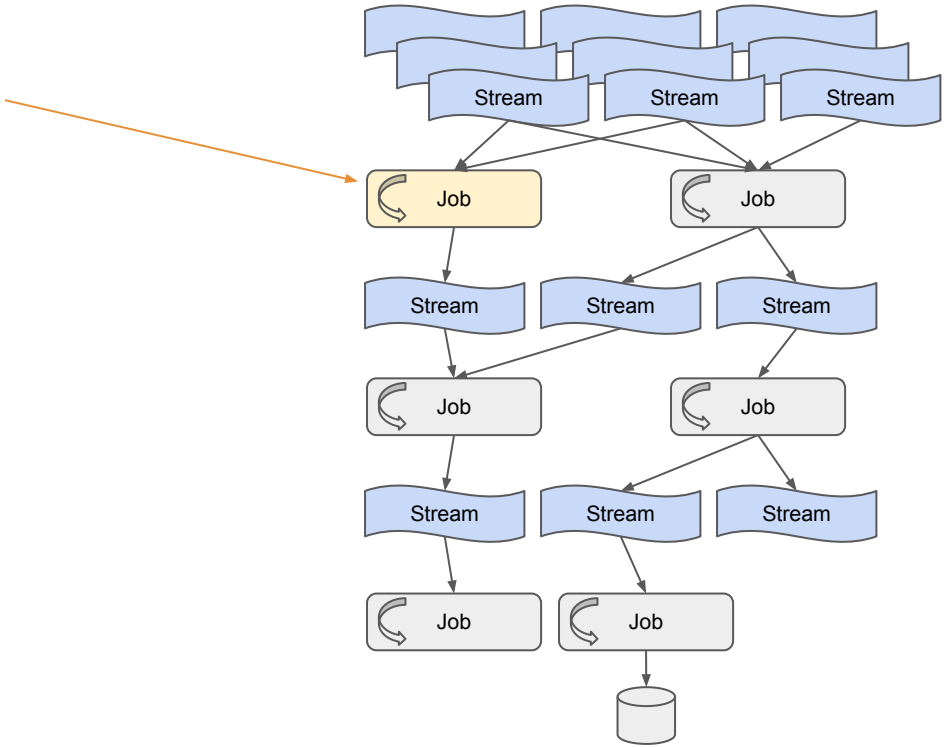
We need real-time everything

- Result now >> result in two hours?
- Today's numbers >> yesterday's?
- Fraud analysis must be quick?
- The demos look great!

"Because life doesn't happen in batch."
- Audi @ Kafka Summit (quoting Ellen Friedman)

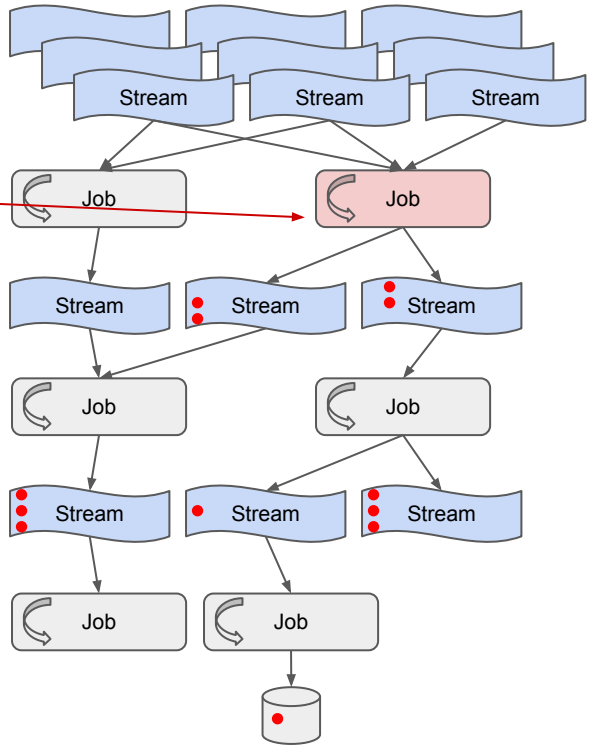
Streaming operations

- Streaming ops is difficult
  - Schema change needed here



Streaming operations

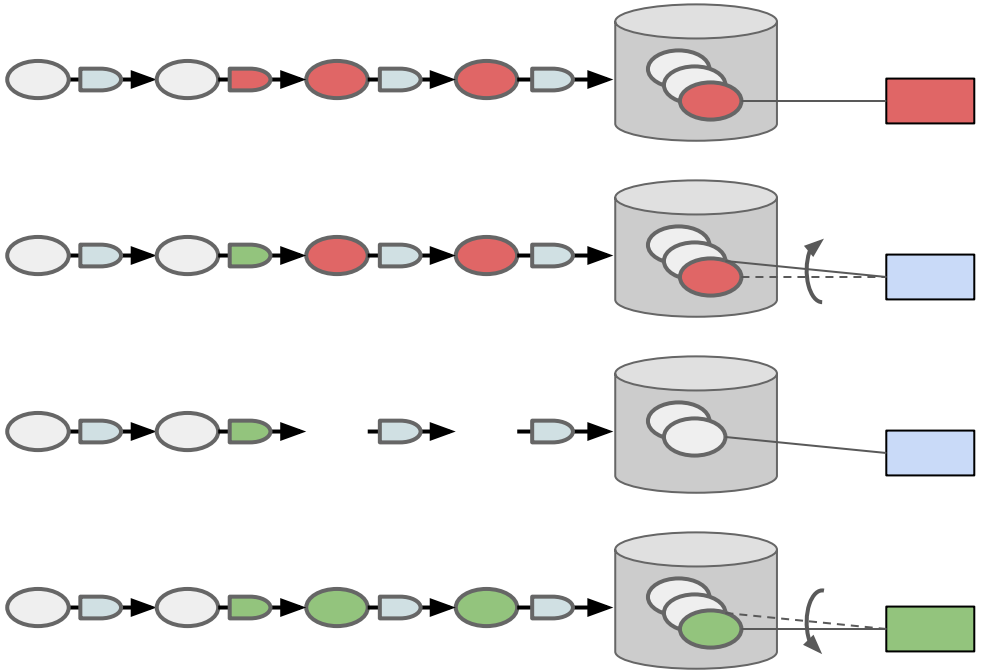
- Streaming ops is difficult
  - Three days of invalid output here



Invalid output, batch operations

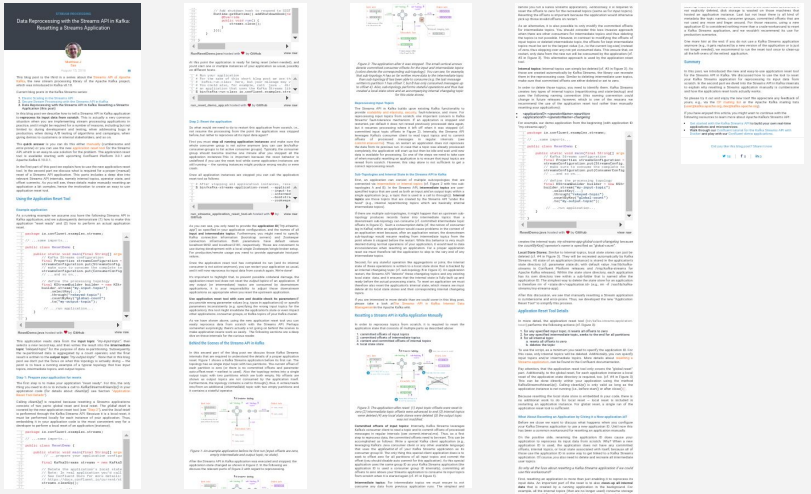
1. Revert serving datasets to old
2. Fix bug
3. Remove faulty datasets
4. Done!

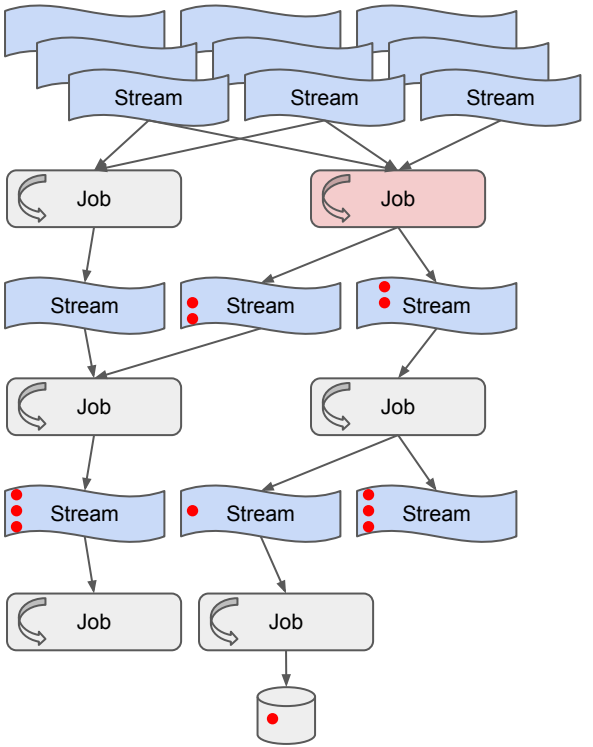
Backfill is automatic (Luigi)



Streaming operations

Reprocessing in Kafka Streams





- Works for a single job, not pipeline. :-(

Mistake 7: Premature streaming

- Stream processing reasonable if one of:
  - Strong use case - worth the cost
  - Single job pipeline
  - Low data quality ok
- Stream tooling is getting better
  - Thanks Confluent, Google!

"Because humans operate on a batch time scale."
- Me @ Berlin Buzzwords.

We see a discrepancy

- These numbers don't add up
  - Have you seen anything suspicious in your pipeline?
  - How many records are invalid?
  - Do these datasets agree?
  - What is your data quality, BTW?
- "Who put 'content-type: bullshit' in these messages?"

"We are the sewer of the company. All garbage dumped upstream ends up here."

Data quality dimensions

- Timeliness
  - E.g. the customer engagement report was produced at the expected time
- Correctness
  - The numbers in the reports were calculated correctly
- Completeness
  - The report includes information on all customers, using all information from the whole time period
- Consistency
  - The customer summaries are all based on the same time period

Mistake 8: Crunching in the dark

- Data products are organic
- Measure odd and unexpected values
  - Spark, Flink, Beam: counters
  - Difficult with SQL processing
- Measure consistency
  - Within records
  - Between records
  - Between datasets
- Be proactive, display in office!

Something broke our pipeline

- "Invalid UTF-8 encoding"
  - It worked yesterday!
- "Field not found"
  - Who changed the schema?

Pipeline fossil

- "Invalid UTF-8 encoding"
  - It worked yesterday!
- "Field not found"
  - Who changed the schema?

- "Yes, we should change that field"
  - But it is too costly
- "We built a new pipeline in a few weeks"
  - And removed the old after 18 months

Mistake 9: Weak end-to-end agility

Data agility measure: Time from client logging field → pipeline → use in feature / BI

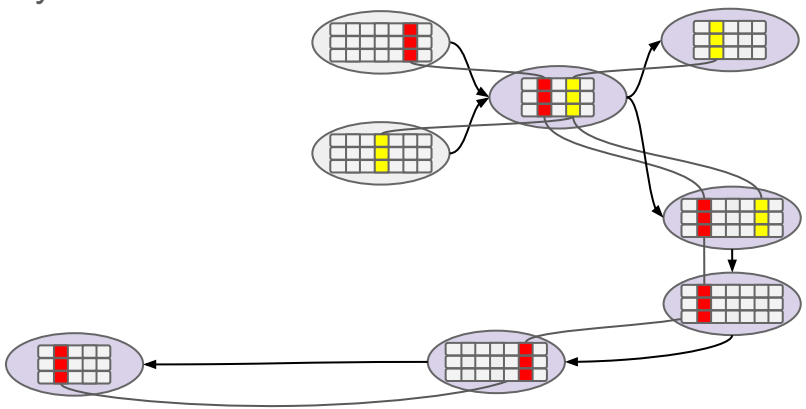
- Siloed company: ~6 months
- Agile company: 1 month
- Coordinated company: days

Remedy: End-to-end testing. Rare, often in conflict with company culture.

Users everywhere

Functional architecture:

- Event-oriented - append only
- Immutability
- At-least-once semantics
- Reproducibility
  - Through 1000s of copies
- Redundancy



GDPR came along

- Please delete me?
- What data have you got on me?
- Please correct this data

- Hold on a second...

Negative value heterogeneity

- We use Avro everywhere, why is this one in Parquet?
  - Beam did not support Parquet. :-(

- Why do we have 25 time formats?
  - ISO 8601, UTC assumed
  - ISO 8601 + timezone
  - Millis since epoch, UTC
  - Nanos since epoch, UTC
  - Millis since epoch, user local time
  - ...
  - ...
  - Float of seconds since epoch, as string. WTF?!?

Mistake 10: Late governance

Data governance for techies:

Proactive things you do in order to avoid later WTF.

Development speed

- Technical processes
- Domain definitions
- Schema evolution
- End-to-end testing

Data democracy

- Data lake rules
- Data quality communication

GDPR compliance

- Erasure
- Anonymisation & pseudonymisation
- Enforced retention
- Access limitations

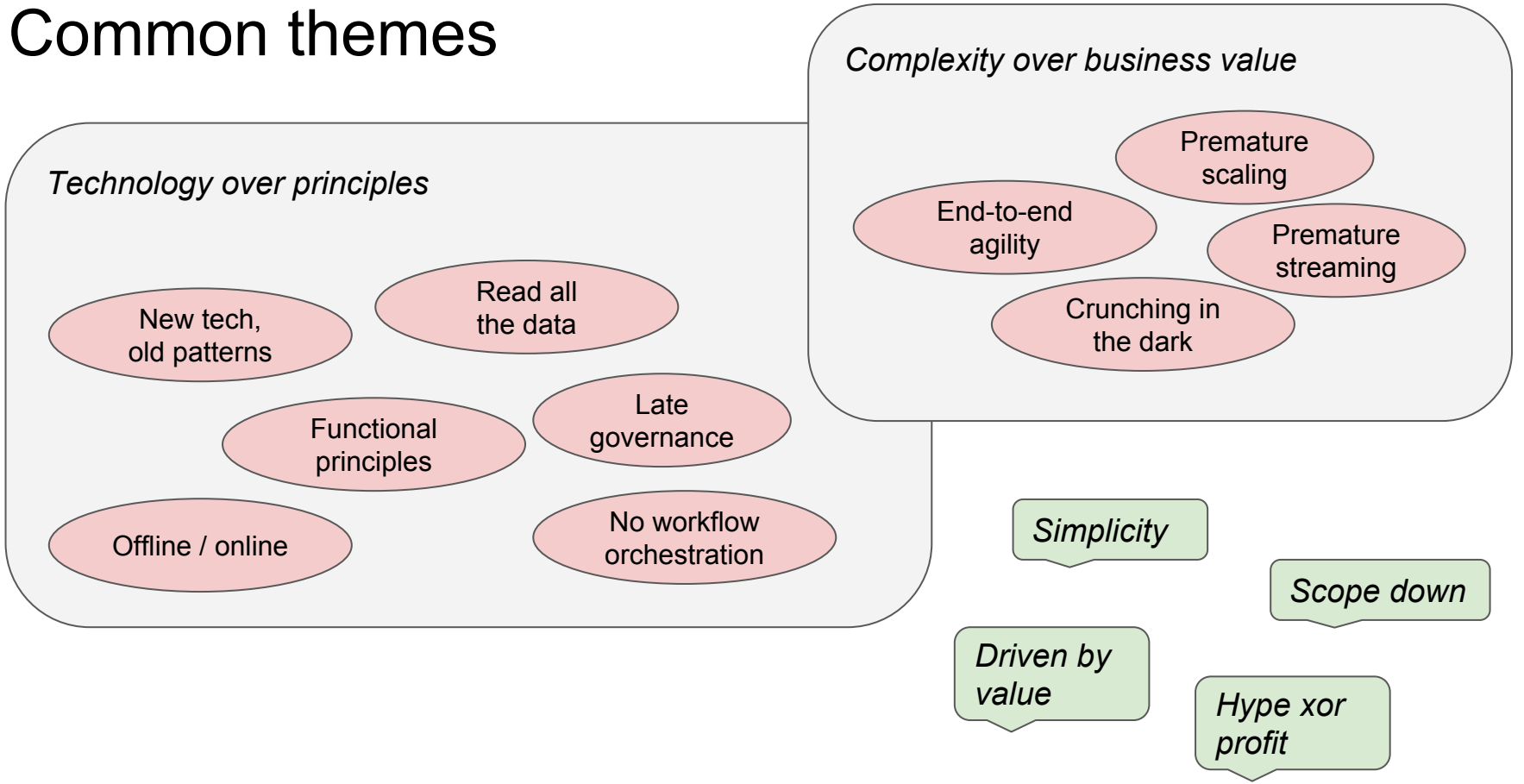
Security

- Blast radius
- Disaster recovery

Financial compliance

- Reproducibility
- Data completeness
- Data consistency

Common themes



Related work

- https://tinyurl.com/mapflat-10-stumble
- Avoiding big data anti-patterns - https://youtu.be/t63SF2UkD0A
- https://www.slideshare.net/JesseAnderson/the-five-dysfunctions-of-a-data-engineering-team

- https://tinyurl.com/mapflat-reading-list
- http://www.mapflat.com/presentations/

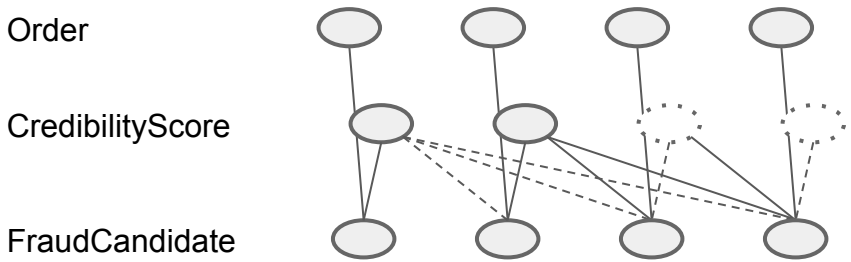
Bonus slides

Dynamic DAGs

```
class FraudCandidate(SparkSubmitTask):
  date = DateParameter()

  def requires(self):
    return [OrderDaily(date=self.date), self._credibility()]

  def _credibility(self):
    max_age_days = 3
    for lookback in range(max_age_days):
      candidate = CredibilityScore(date - timedelta(days=lookback))
      if candidate.output().exists():
        return candidate
    return CredibilityScore(date - timedelta(days=max_age_days))
```



Luigi or Airflow?

Luigi & Kubernetes:

```
myjob.yaml:

kind: CronJob
spec:
  schedule: "17 * * * *"
  jobTemplate:
    spec:
      containers:
        - name: "MyJob"
          image: myregistry/mypipeline:latest
          command: ["luigi"]
          args: ["--module", "mypipeline",
                 "RangeDaily", "--of",
                 "MyJob",
                 "--days-back", "14"]
```

- 1 hour to production

Airflow & Kubernetes:

- AIRFLOW-1314, created 2017-06-16
  - 9/12 sub-tasks resolved
  - 6 PRs
  - 28 commits
  - ~30 files touched
  - ~2000 lines of code

- Airflow monitoring has value, however

Counters

- User-defined
- Technical from framework
  - Execution time
  - Memory consumption
  - Data volumes
  - ...

```
case class Order(item: ItemId, userId: UserId)
case class User(id: UserId, country: String)

val orders = read(orderPath)
val users = read(userPath)

val orderNoUserCounter = longAccumulator("order-no-user")

val joined: C[(Order, Option[User])] = orders
  .groupBy(_.userId)
  .leftJoin(users.groupBy(_.id))
  .values

val orderWithUser: C[(Order, User)] = joined
  .flatMap( orderUser match
    case (order, Some(user)) => Some((order, user))
    case (order, None) => {
      orderNoUserCounter.add(1)
      None
    })
```

SQL: Nope