Data pipelines from zero to solid

Lars Albertsson
www.mapflat.com

Who’s talking?

Swedish Institute of Computer Science (test tools)

Sun Microsystems (very large machines)

Google (Hangouts, productivity)

Recorded Future (NLP startup)

Cinnober Financial Tech. (trading systems)

Spotify (data processing & modelling)

Schibsted (data processing & modelling)

Independent data engineering consultant

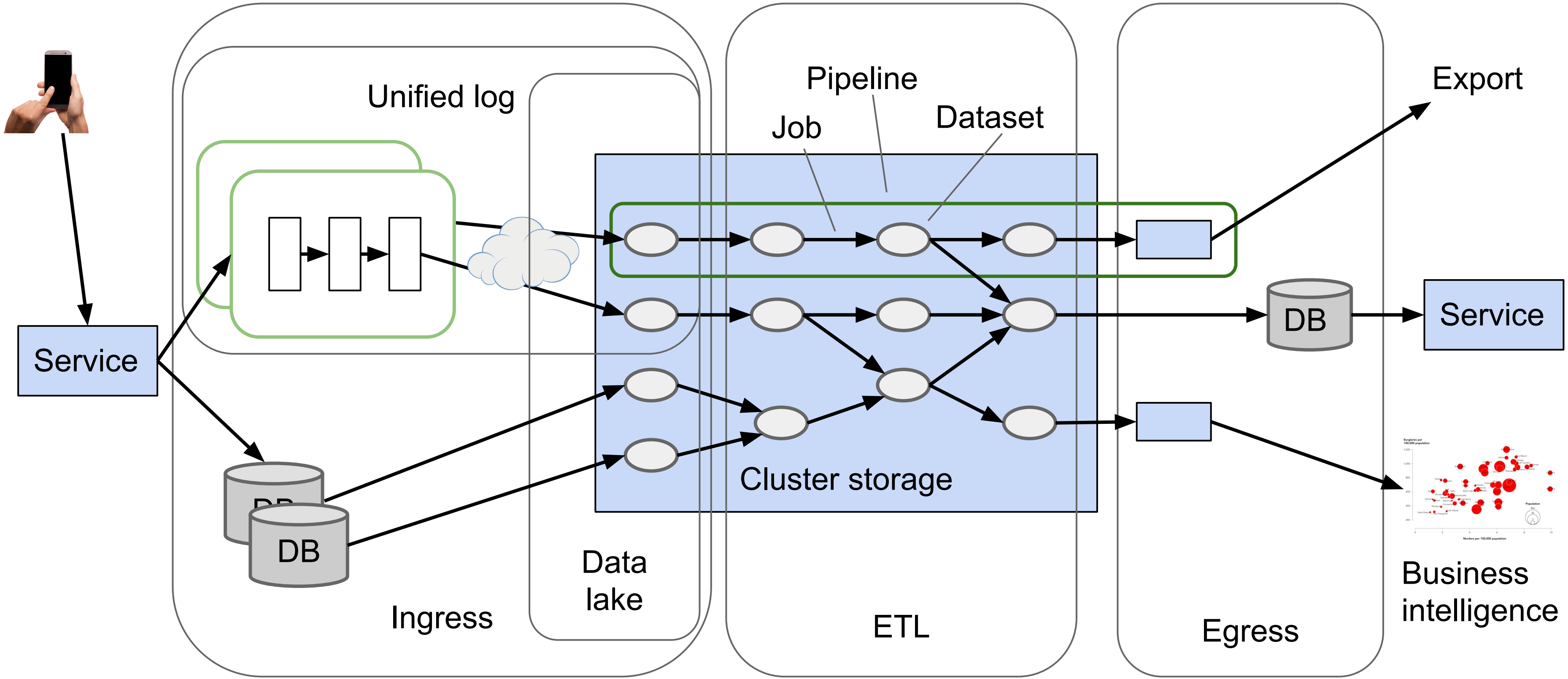
Presentation goals

- Overview of data pipelines for analytics / data products
- Target audience: Big data starters
  - Seen wordcount, need the stuff around
- Overview of necessary components & wiring
- Base recipe
  - In vicinity of state-of-practice
  - Baseline for comparing design proposals
- Subjective best practices - not single truth
- Technology **suggestions**, (alternatives)

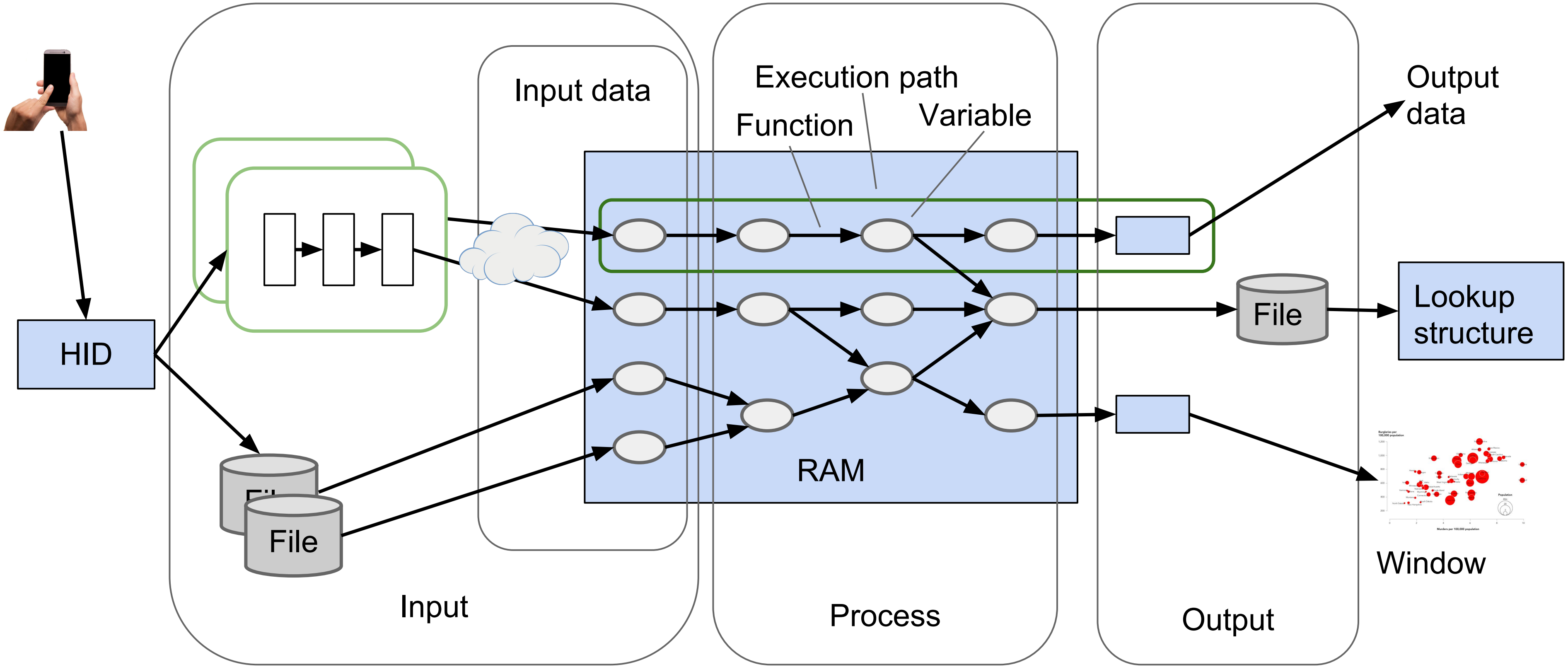
Presentation non-goals

- Stream processing
  - High complexity in practice
  - Batch processing yields > 90% of value
- Technology enumeration or (fair) comparison
- Writing data processing code
  - Already covered en masse

Data product anatomy



Computer program anatomy



Data pipeline = yet another program

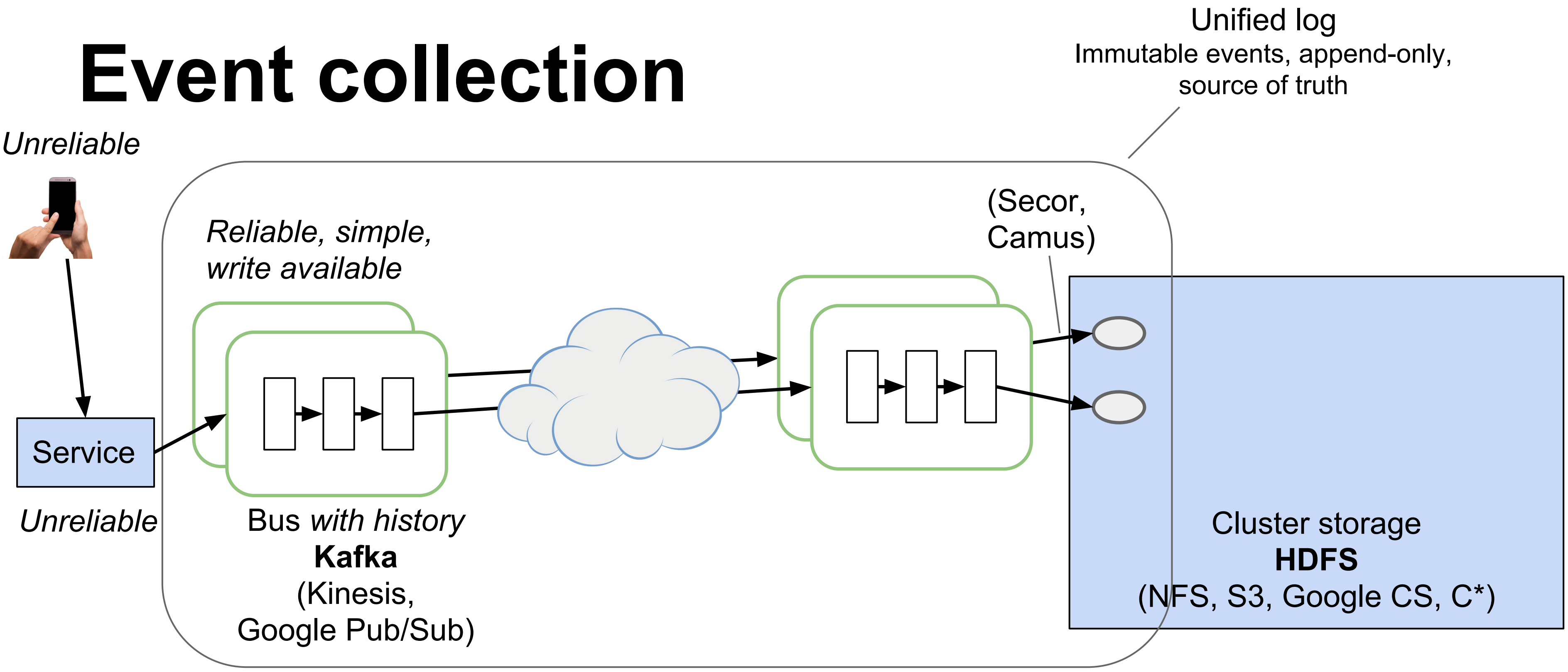
Don’t veer from best practices

- Regression testing
- Design: Separation of concerns, modularity, etc
- Process: CI/CD, code review, lint tools
- Avoid anti-patterns: Global state, hard-coding location, duplication, ...

In data engineering, slipping is the norm... :-(

Solved by mixing strong software engineers with data engineers/scientists. Mutual respect is crucial.

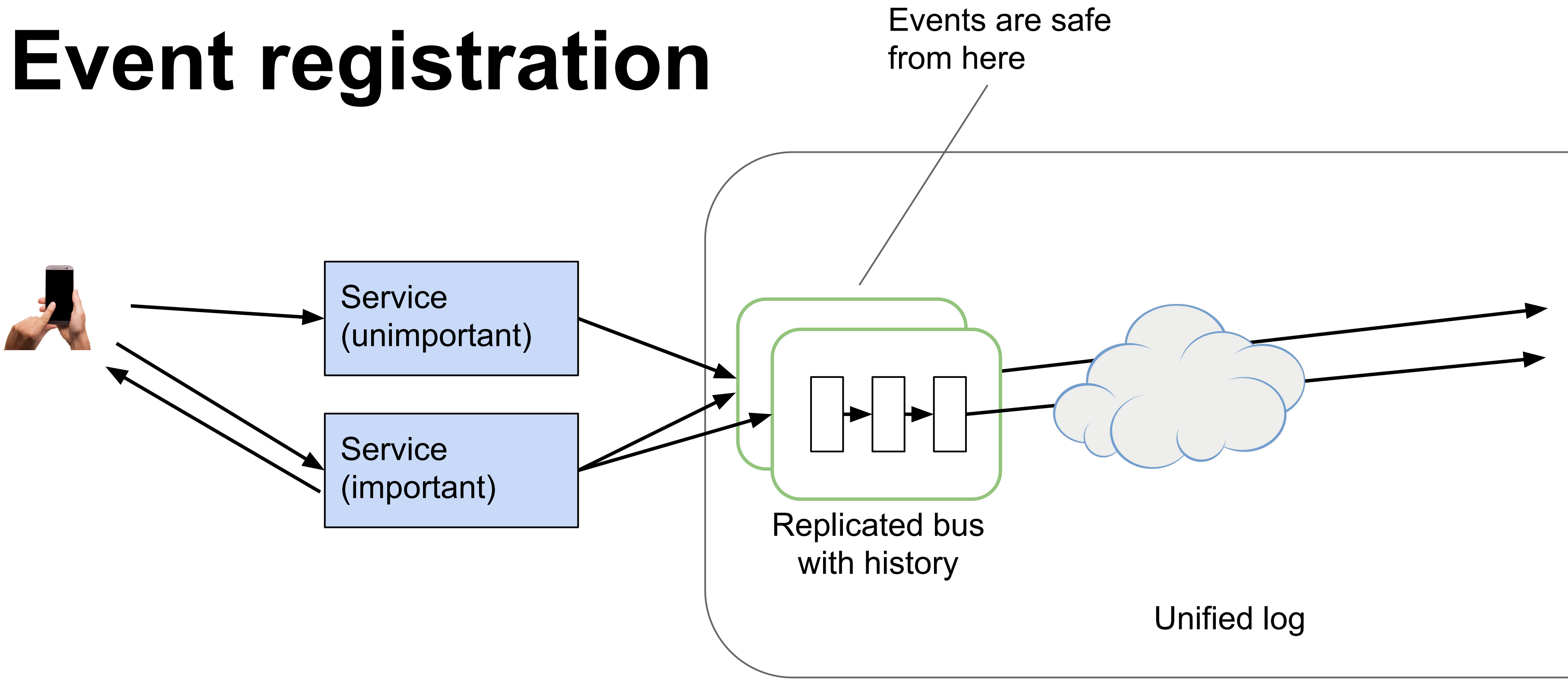
Event collection



Immediate handoff to append-only replicated log.

Once in the log, events eventually arrive in storage.

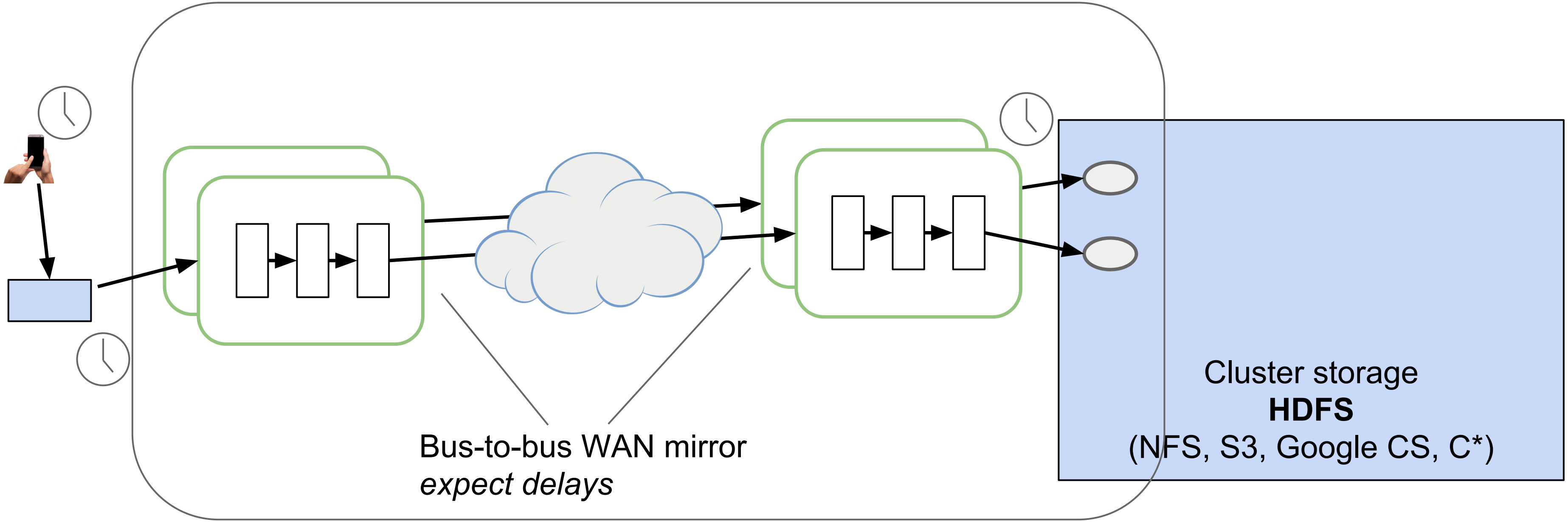
Event registration



Asynchronous fire-and-forget handoff for unimportant data.

Synchronous, replicated, with ack for important data

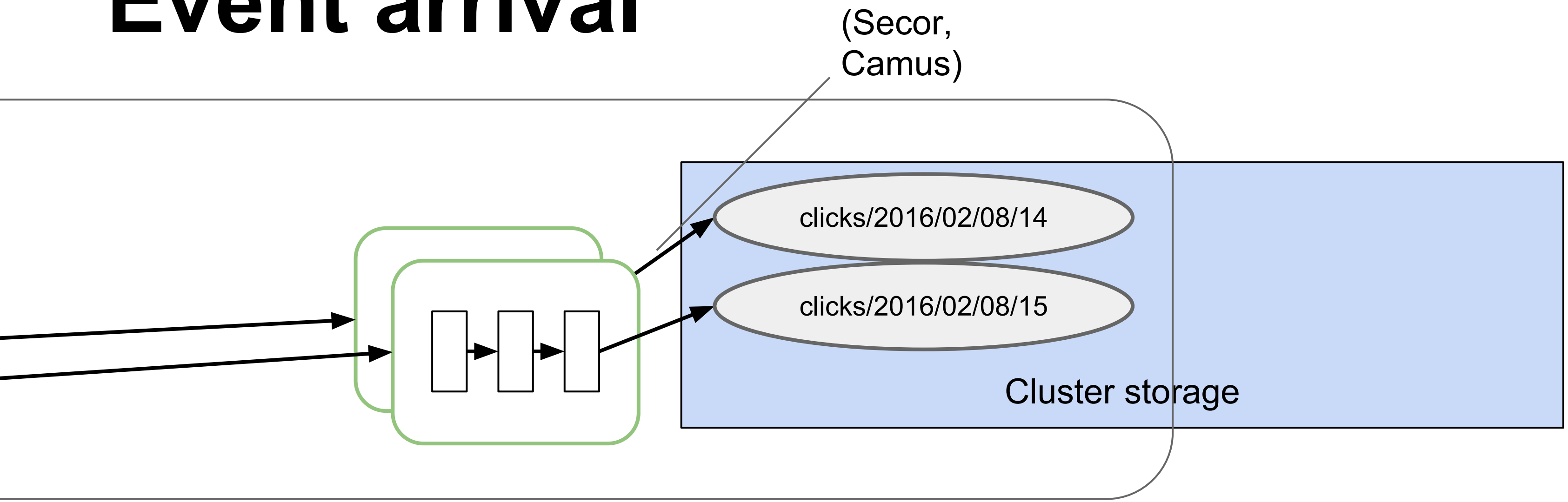
Event transportation



Log has *long history* (months+) => robustness end to end.

Avoid risk of processing & decoration. Except timestamps.

Event arrival



Bundle incoming events into datasets

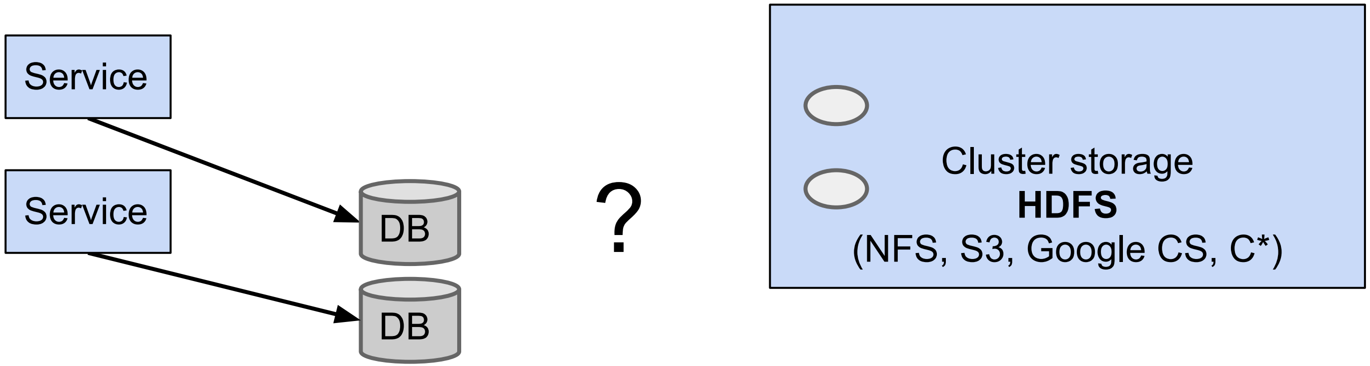
- Sealed quickly, thereafter immutable
- Bucket on arrival / wall-clock time
- Predictable bucketing, e.g. hour

Database state collection

Source of truth sometimes in database.

Snapshot to cluster storage.

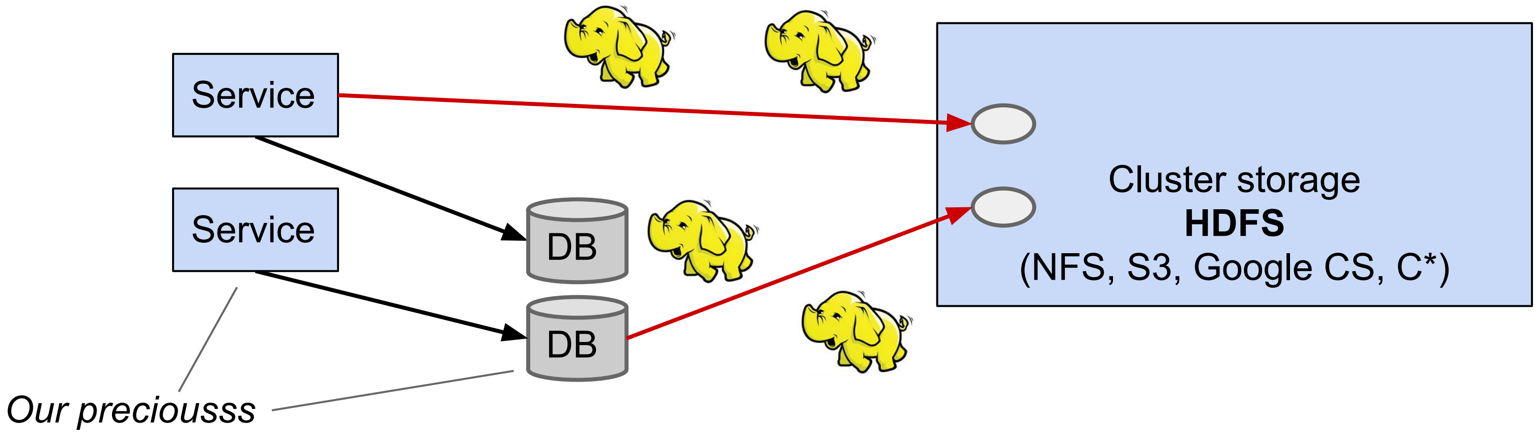
Easy on surface...



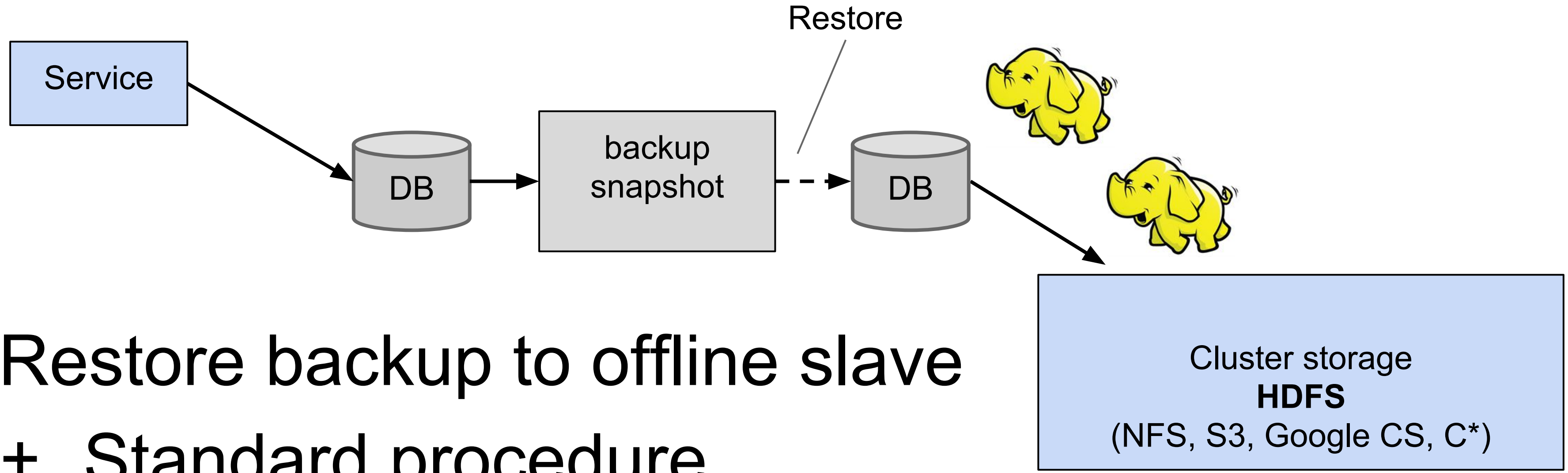
Anti-pattern: Send the oliphants!

- ~~Sqoop (dump with MapReduce) production DB~~
- ~~MapReduce from production API~~

Hadoop / Spark == internal DDoS service



Deterministic slaves

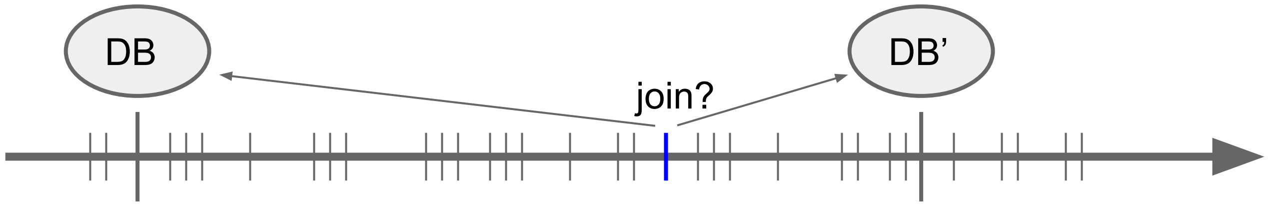


Restore backup to offline slave

\+ Standard procedure

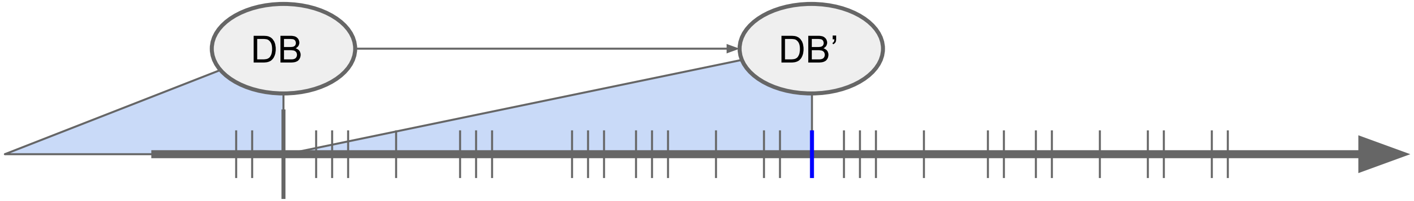
\- Serial or resource consuming

Using snapshots



- join(event, snapshot) => always time mismatch
- Usually acceptable
- Some behaviour difficult to catch with snapshots
  - E.g. user creates, then deletes account

Event sourcing

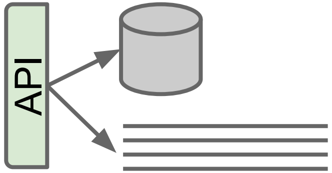


- Every change to unified log == source of truth
- snapshot(t + 1) = sum(snapshot(t), events(t, t+1))
- Allows view & join at any point in time

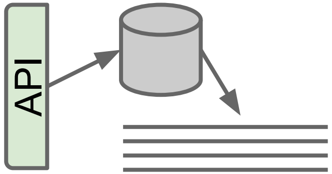
Application services still need DB for current state lookup

Event sourcing, synced database

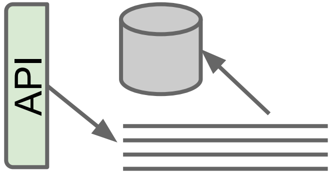
A. Service interface generates events and DB transactions



B. Generate stream from commit log Postgres, MySQL -> Kafka



C. Build DB with stream processing



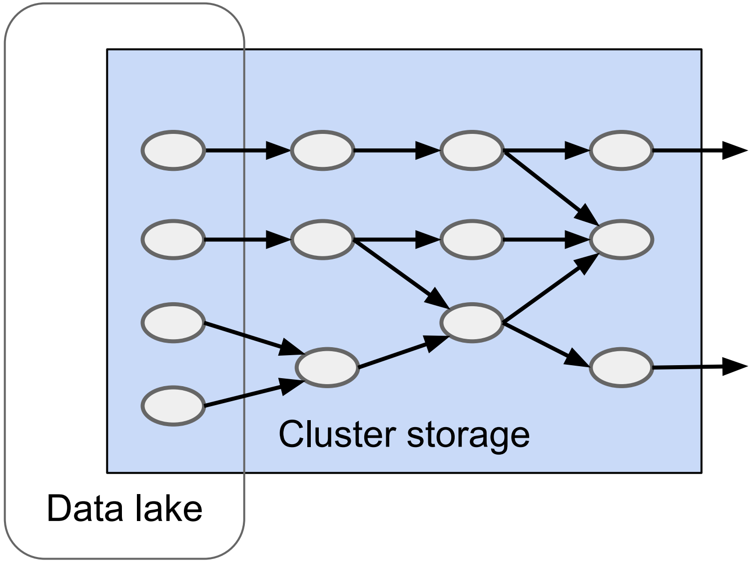
DB snapshot lessons learnt

- Put fences between online and offline components
  - The latter can kill the former
- Team that owns a database/service must own exporting data to offline
  - Protect online stability
  - Affects choice of DB technology

The data lake

Unified log + snapshots

- Immutable datasets
- Raw, unprocessed
- Source of truth from batch processing perspective
- Kept as long as permitted
- Technically homogeneous



Datasets

- Pipeline equivalent of objects
- Dataset class == homogeneous records, open-ended
  - Compatible schema
  - E.g. MobileAdImpressions
- Dataset instance = dataset class + parameters
  - Immutable
  - E.g. MobileAdImpressions(hour=”2016-02-06T13”)

Representation - data lake & pipes

- Directory with multiple files
  - Parallel processing
  - Sealed with _SUCCESS (Hadoop convention)
  - Bundled schema format
    - **JSON lines, Avro, Parquet**
  - Avoid old, inadequate formats
    - ~~CSV, XML~~
  - RPC formats lack bundled schema
    - ~~Protobuf, Thrift~~

Directory datasets

Privacy level | Dataset class | Schema version | Instance parameters, Hive convention | Seal | Partitions

```
hdfs://red/pageviews/v1/country=se/year=2015/month=11/day=4/_SUCCESS
                                                           part-00000.json
                                                           part-00001.json
```

- Some tools, e.g. Spark, understand Hive name conventions

Ingress / egress representation

Larger variation:

- Single file
- Relational database table
- Cassandra column family, other NoSQL
- BI tool storage
- BigQuery, Redshift, ...

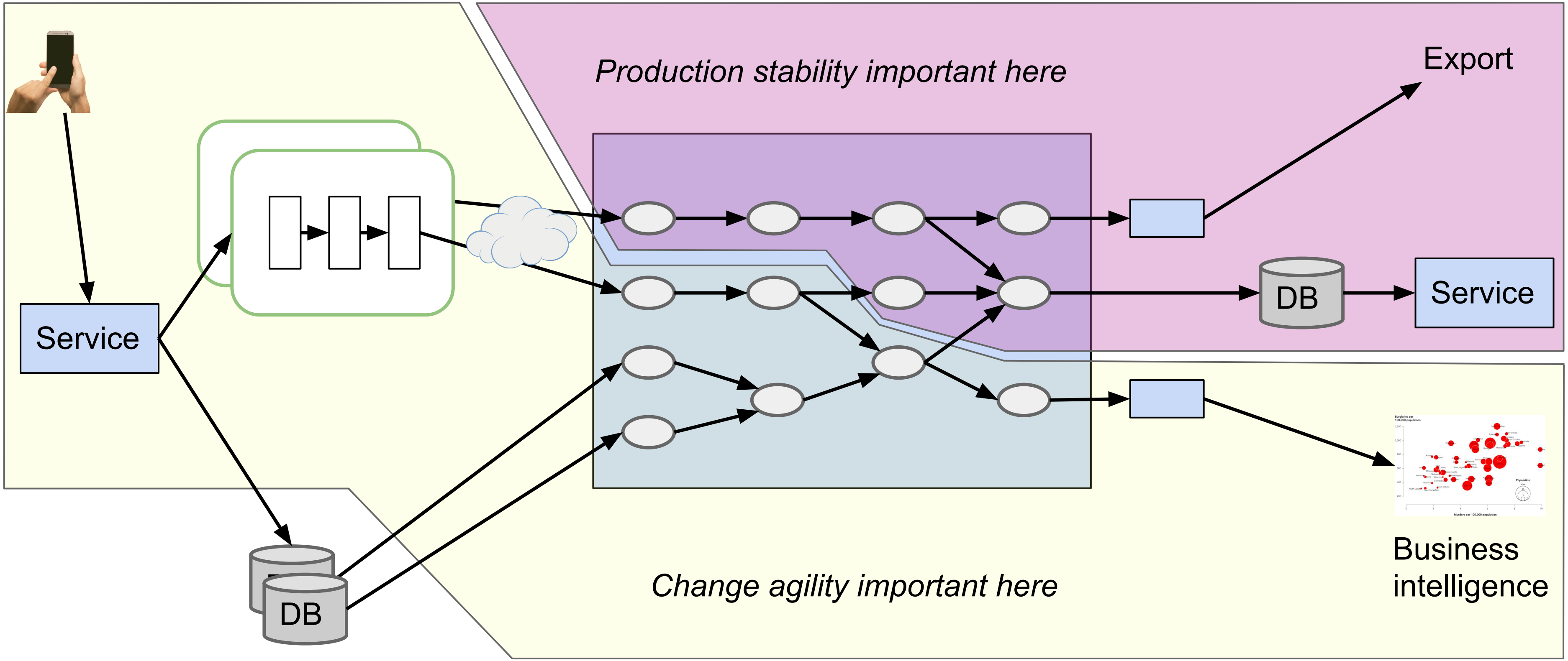
Egress datasets are also atomic and immutable.

E.g. write full DB table / CF, switch service to use it, never change it.

Schemas

- There is always a schema
  - Plan your evolution
- New field, same semantic == compatible change
- Incompatible schema change => new dataset class
- Schema on read - assumptions in code
  - Dynamic typing
  - Quick schema changes possible
- Schema on write - enumerated fields
  - Static typing & code generation possible
  - Changes must propagate down pipeline code

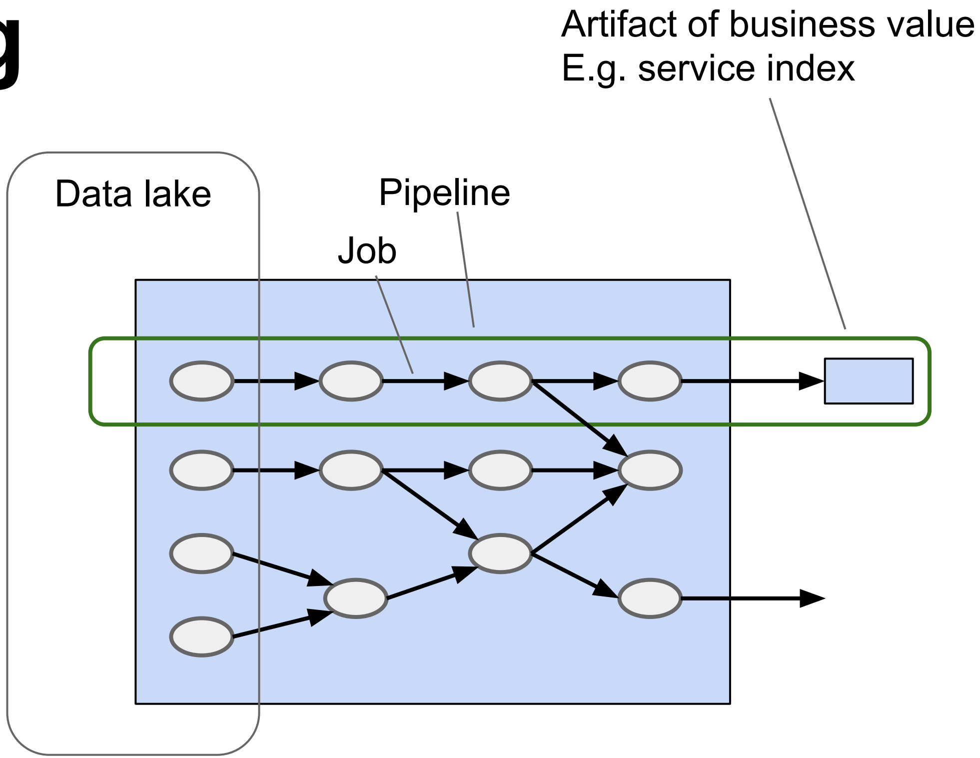
Schema on read or write?



Batch processing

Gradual refinement

1. Wash
   - time shuffle, dedup, ...
2. Decorate
   - geo, demographic, ...
3. Domain model
   - similarity, clusters, ...
4. Application model
   - Recommendations, ...



Batch job code

- Components should scale up
  - **Spark**, (Scalding, Crunch)
- And scale down
  - More important!
  - Component should support local mode
    - Integration tests
    - Small jobs - less risk, easier debugging

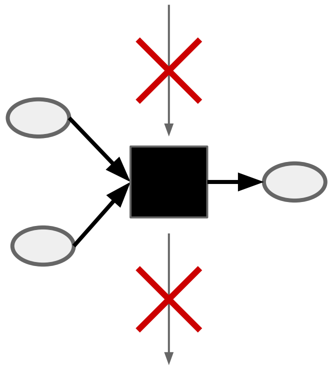
Language choice

- People and community thing, not a technical thing
- Need for simple & quick experiments
  - Java - too much ceremony and boilerplate
- Stable and static enough for production
  - Python/R - too dynamic
- Scala connects both worlds
  - Current home of data innovation
- Beware of complexity - keep it sane and simple
  - Avoid spaceships: <|*|> |@| <**>

Batch job

Job == function([input datasets]): [output datasets]

- No orthogonal concerns
    - ~~Invocation~~
    - ~~Scheduling~~
    - ~~Input / output location~~
- Testable
- No other input factors
- No side-effects
- Ideally: atomic, deterministic, idempotent

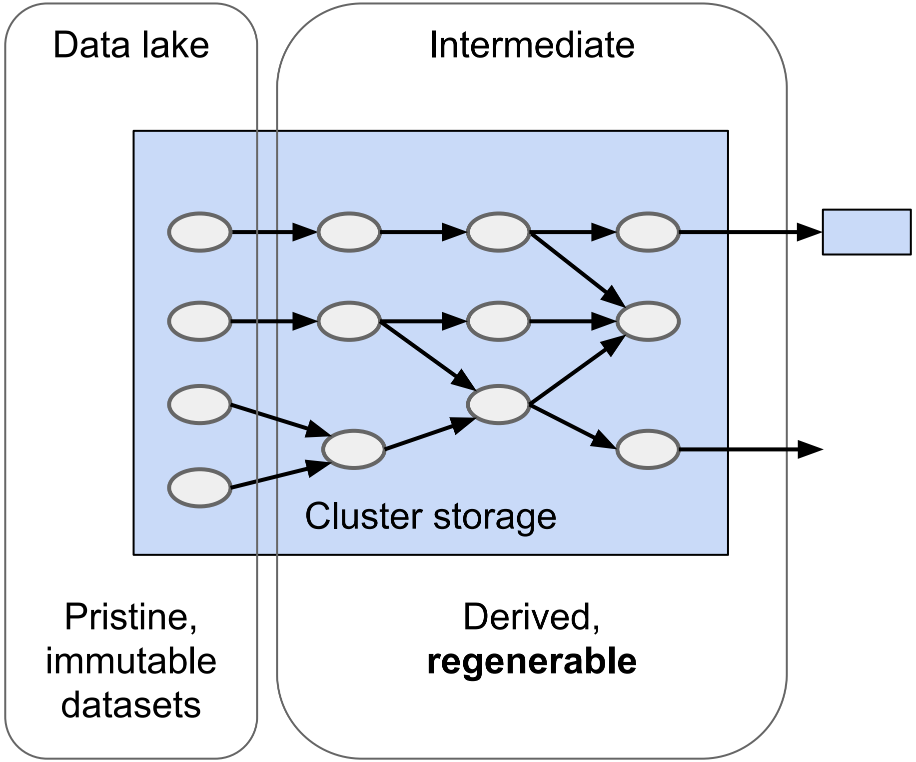


Batch job class & instance

- Pipeline equivalent of Command pattern
- Parameterised
  - Higher order, c.f. dataset class & instance
  - Job instance == job class + parameters
  - Inputs & outputs are dataset classes
- Instances are ideally executed when input appears
  - *Not on cron schedule*

Pipelines

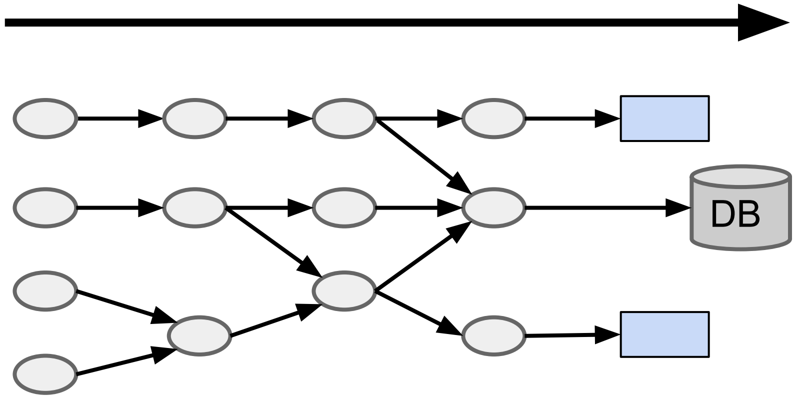
- Things will break
  - Input will be missing
  - Jobs will fail
  - Jobs will have bugs
- Datasets must be rebuilt
- Determinism, idempotency
- Backfill missing / failed
- Eventual correctness



Workflow manager

- Dataset “build tool”
- Run job instance when
    - input is available
    - output missing
    - resources are available
- Backfill for previous failures
- DSL describes DAG
- Includes ingress & egress

Luigi, (Airflow, Pinball)

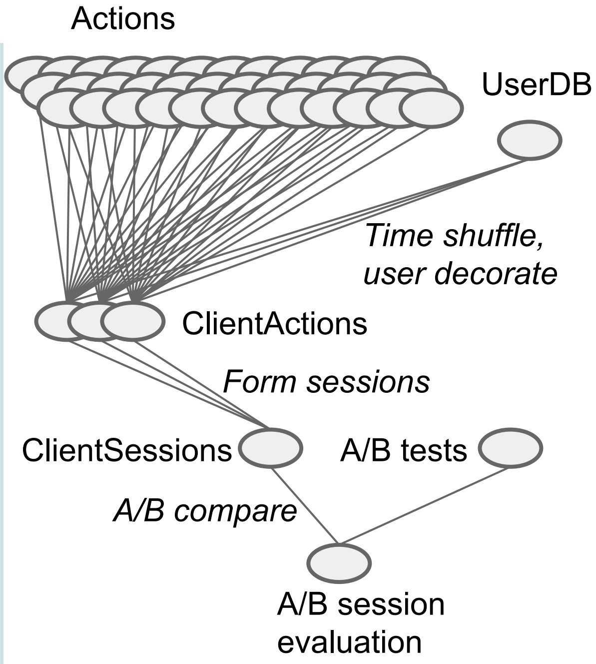


DSL DAG example (Luigi)

```
class ClientActions(SparkSubmitTask):
  hour = DateHourParameter()
  def requires(self):
    return [Actions(hour=self.hour - timedelta(hours=h)) for h in range(0, 12)] + \
      [UserDB(date=self.hour.date)]
  ...

class ClientSessions(SparkSubmitTask):
  hour = DateHourParameter()
  def requires(self):
    return [ClientActions(hour=self.hour - timedelta(hours=h)) for h in range(0, 3)]
  ...

class SessionsABResults(SparkSubmitTask):
  hour = DateHourParameter()
  def requires(self):
    return [ClientSessions(hour=self.hour), ABExperiments(hour=self.hour)]

  def output(self):
    return HdfsTarget("hdfs://production/red/ab_sessions/v1/" +
      "{:year=%Y/month=%m/day=%d/hour=%H}".format(self.hour))

  ...
```

Job (aka Task) classes

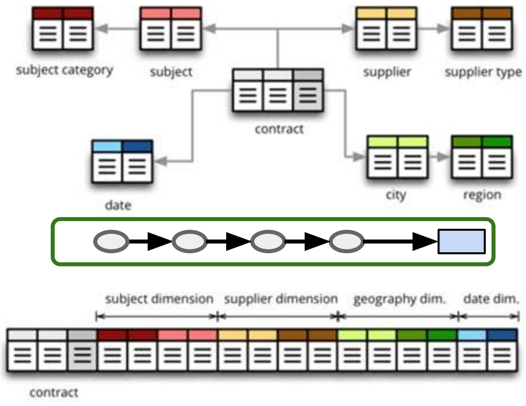
Dataset instance



- Expressive, embedded DSL - a must for ingress, egress
  - Avoid weak DSL tools: ~~Oozie, AWS Data Pipeline~~

Egress datasets

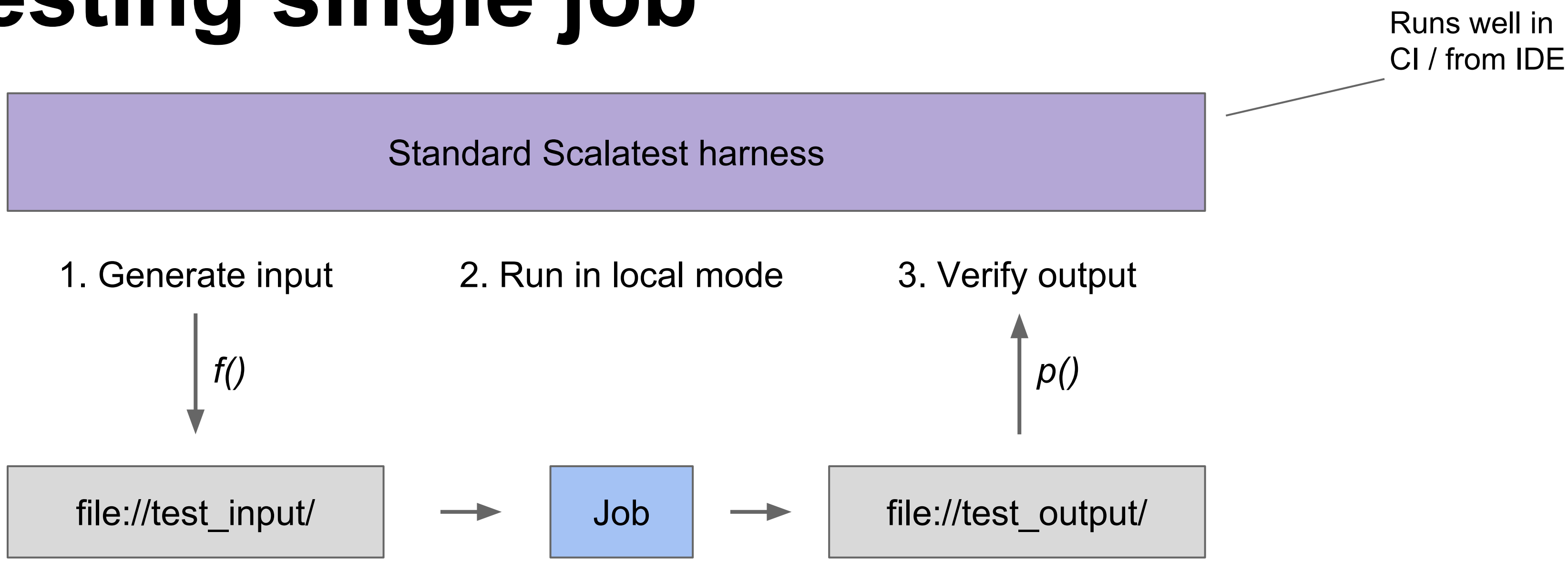
- Serving
  - Precomputed user query answers
  - Denormalised
  - **Cassandra**, (many)
- Export & Analytics
  - **SQL** (single node / Hive, Presto, ..)
  - Workbenches (Zeppelin)
  - (Elasticsearch, proprietary OLAP)
- BI / analytics tool needs change frequently
  - Prepare to redirect pipelines



Test strategy considerations

- *Developer productivity is the primary value of test automation*
- Test at stable interface
    - Minimal maintenance
    - No barrier to refactorings
- Focus: single job + end to end
    - Jobs & pipelines are pure functions - easy to test
- Component, unit - only if necessary
    - Avoid dependency injection ceremony

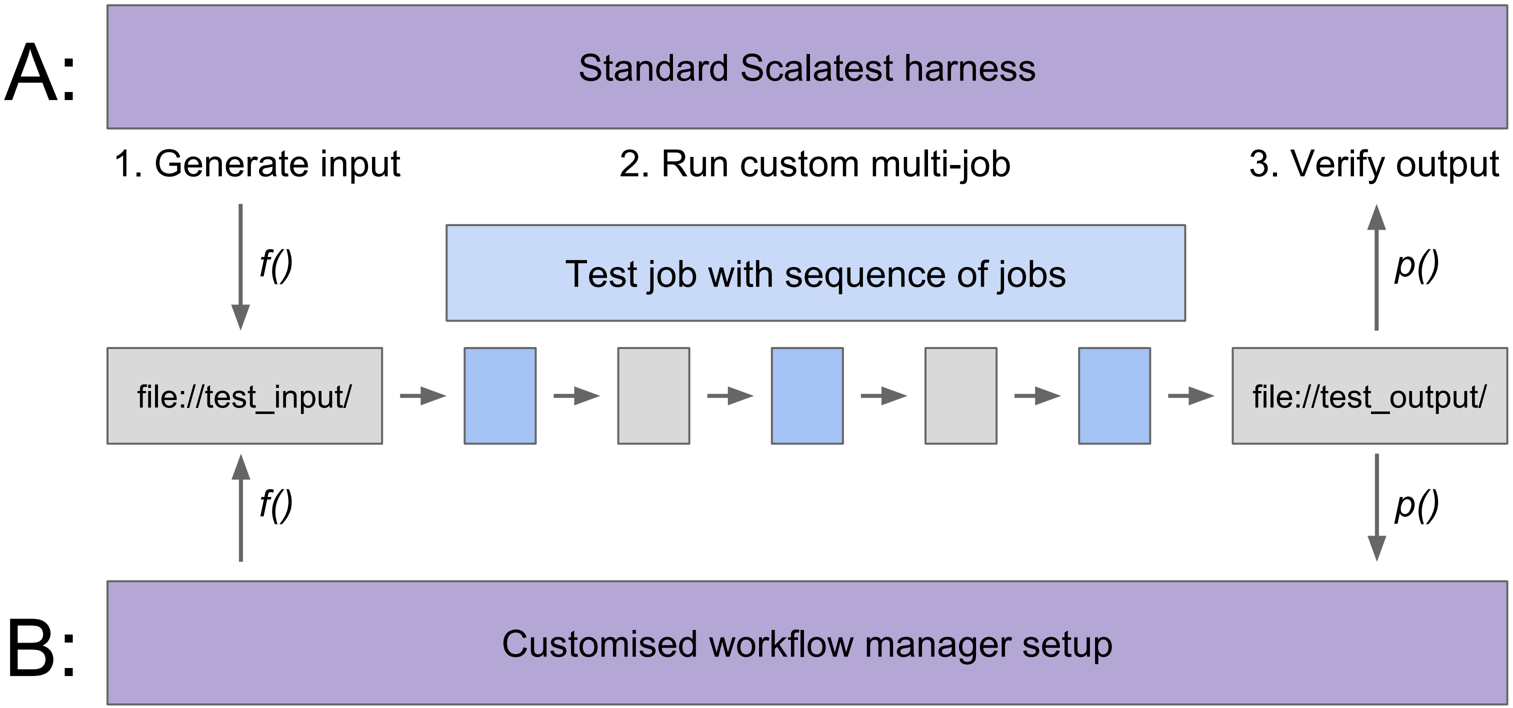
Testing single job



Don't commit - expensive to maintain. Generate / verify with code.

- (Tool-specific frameworks, e.g. for Spark?)
  - Usable, but rarely cover I/O - home of many bugs.
  - Tied to processing technology

Testing pipelines - two options

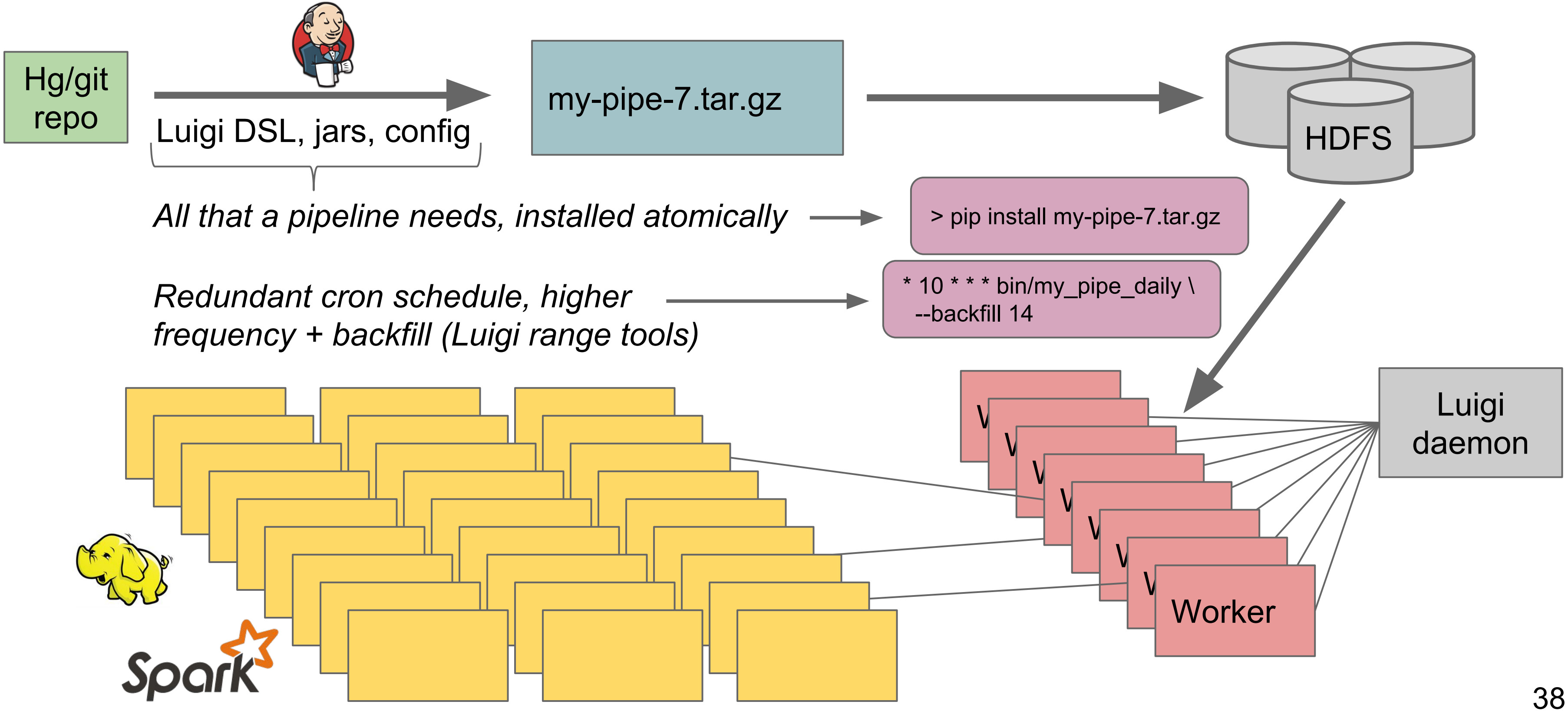


+ Runs in CI
+ Runs in IDE
+ Quick setup
- Multi-job
  maintenance

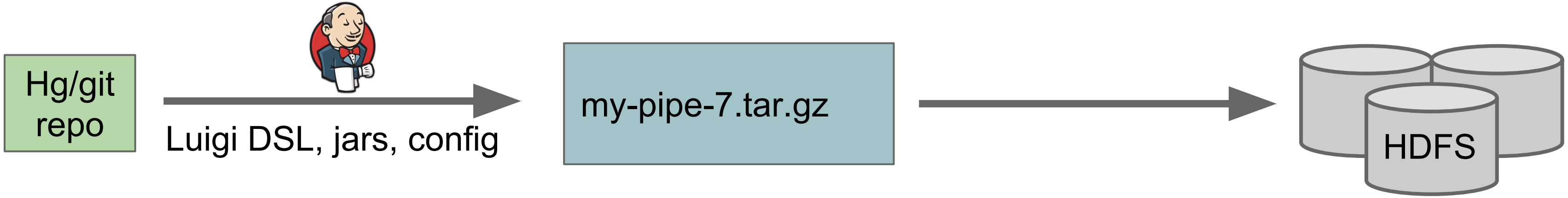
+ Tests workflow logic
+ More authentic
- Workflow mgr setup
  for testability
- Difficult to debug
- Dataset handling
  with Python

- Both can be extended with Kafka, egress DBs

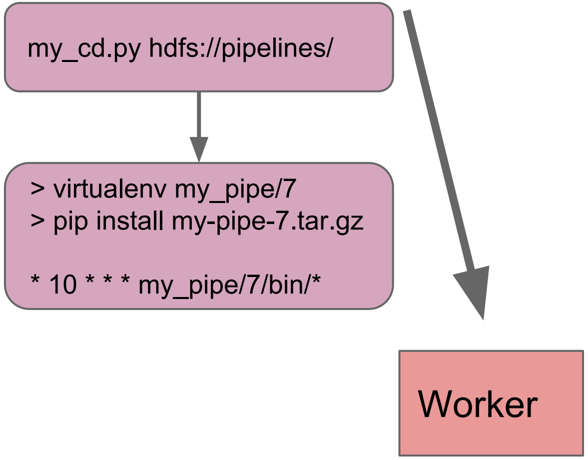
Deployment


Hg/git repo
Luigi DSL, jars, config
my-pipe-7.tar.gz
HDFS
All that a pipeline needs, installed atomically
> pip install my-pipe-7.tar.gz
Redundant cron schedule, higher frequency + backfill (Luigi range tools)
* 10 * * * bin/my_pipe_daily \
--backfill 14
Luigi daemon
Worker
Spark

Continuous deployment



- Poll and pull latest on worker nodes
  - virtualenv package/version
    - No need to sync environment & versions
  - Cron package/latest/bin/*
    - Old versions run pipelines to completion, then exit



Start lean: assess needs

Your data & your jobs:

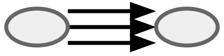
A. Fit in one machine, and will continue to do so
B. Fit in one machine, but grow faster than Moore’s law
C. Do not fit in one machine

- Most datasets / jobs: A
    - Even at large companies with millions of users
- cost(C) >> cost(A)
- Running A jobs on C infrastructure is expensive

Lean MVP

- Start simple, lean, end-to-end
  - *No parallel cluster computations necessary?*
  - Custom jobs or local Spark/Scalding/Crunch
- Shrink data
  - Downsample
  - Approximate algorithms (e.g. Count-min sketch)
- Get workflows running
  - Serial jobs on one/few machines
  - Simple job control (Luigi only / simple work queue)

Scale carefully



- Get end-to-end workflows in production for evaluation
  - Improvements driven by business value, not tech
- Keep focus small
  - Business value
  - Privacy needs attention early
- Keep iterations swift
  - Integration test end-to-end
  - Efficient code/test/deploy cycle
- Parallelise jobs only when forced

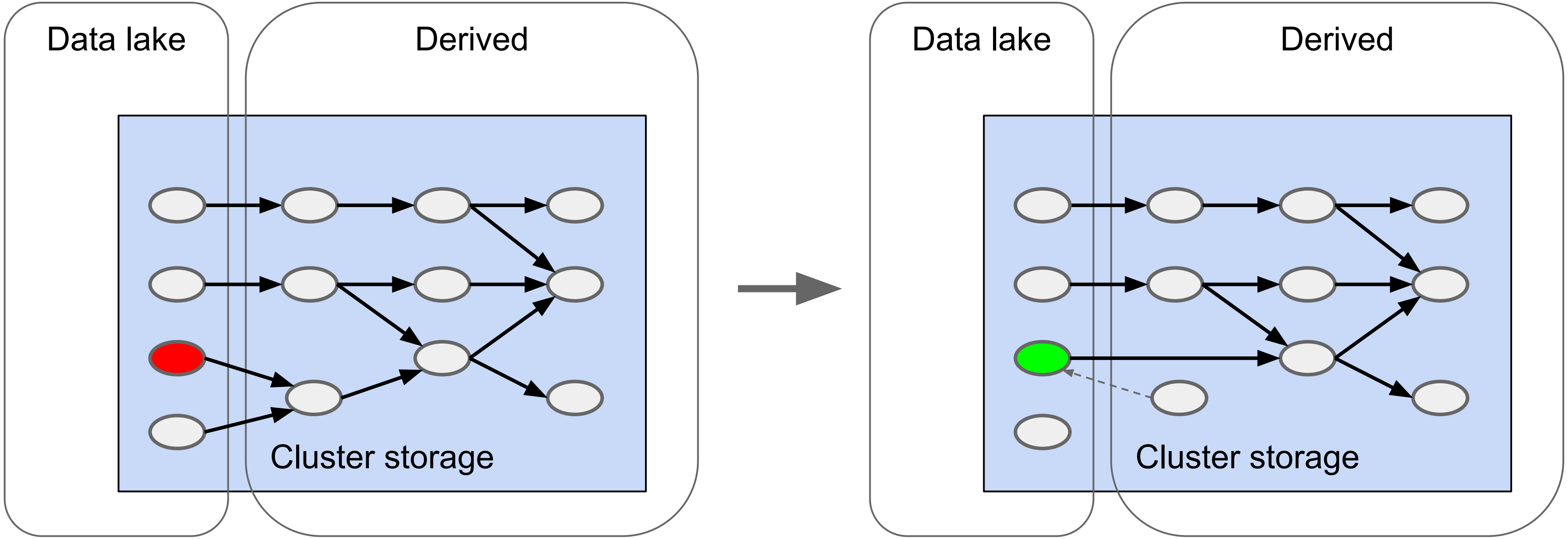
Protecting privacy in practice

- Removing old personal identifiable information (PII)
- Right to be forgotten
- Access control to PII data
- Audit of access and processing

- PII content definition is application-specific
- PII handling subject to business priorities
  - But you should have a plan from day one

Data retention

- Remove old, promote derived datasets to lake



PII removal

Key on PII => difficult to wash

bobwhite,http://site_a/,2015-01-03T
bobwhite,http://site_b/,2015-01-03T
joeblack,http://site_c/,2015-01-03T

bobwhite,Bath,uk
joeblack,Bristol,uk

bobwhite,http://site_a/,2015-01-03T,Bath,uk
bobwhite,http://site_b/,2015-01-03T,Bath,uk
joeblack,http://site_c/,2015-01-03T,Bristol,uk

Split out PII,
wash on user
deletion

34ac,bobwhite
56bd,null

34ac,http://site_a/,2015-01-03T
34ac,http://site_b/,2015-01-03T
56bd,http://site_c/,2015-01-03T

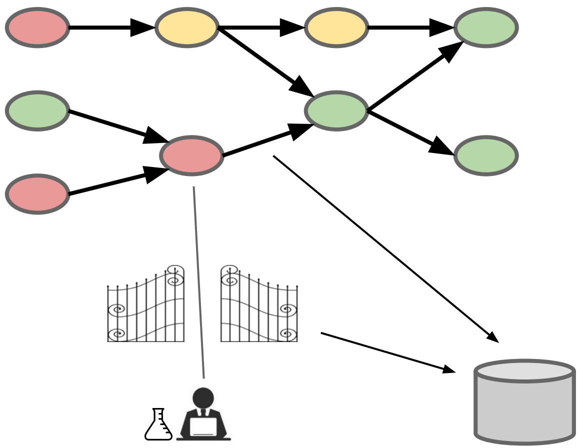
34ac,Bath,uk
56db,Bristol,uk

bobwhite,http://site_a/,2015-01-03T,Bath,
uk
bobwhite,http://site_b/,2015-01-03T,Bath,
uk
null,http://site_c/,2015-01-03T,Bristol,uk

- Must rebuild downstream datasets regularly
    - In order for PII to be washed in x days

Simple PII audit

- Classify PII level
  - Name, address, messages, ...
  - IP, city, ...
  - Total # page views, …
- Tag datasets and jobs in code
- Manual access through gateway tool
  - Verify permission, log
  - Dedicated machines only
- Log batch jobs
  - Deploy with CD only, log hg/git commit hash



Parting words + sales plug

Keep things simple; batch, homogeneity & little state

Focus on developer code, test, debug cycle - end to end

Harmony with technical ecosystems

Little technology overlap with yesterday - follow leaders

Plan early: Privacy, retention, audit, schema evolution

Please give feedback -- mapflat.com/feedback

I help companies plan and build these things

Bonus slides

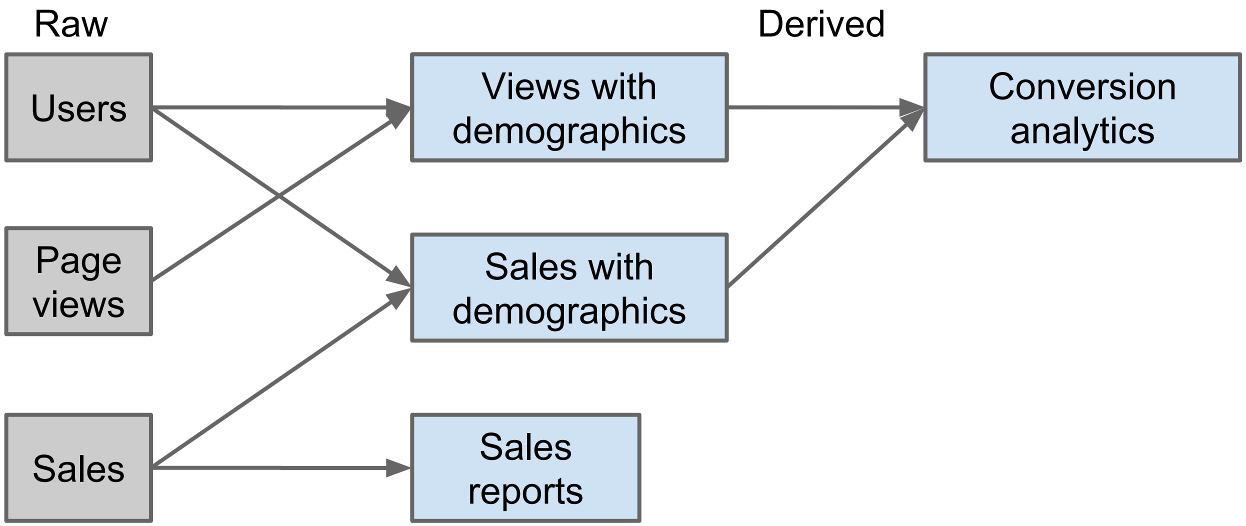
Cloud or not?

+ Operations
+ Security
+ Responsive scaling
- Development workflows
- Privacy
- Vendor lock-in

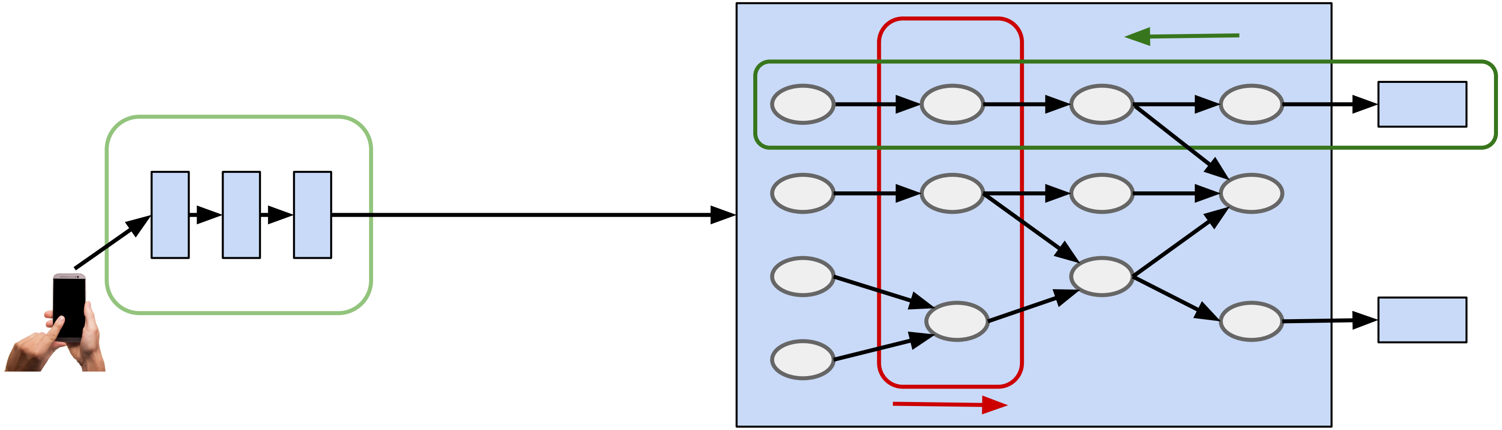
Security?

- Afterthought add-on for big data components
  - E.g. Kerberos support
  - Always trailing - difficult to choose global paradigm
- Container security simpler
  - Easy with cloud
  - Immature with on-premise solutions?

Data pipelines example



Data pipelines team organisation



Form teams that are driven by business cases & need

Forward-oriented -> filters implicitly applied

Beware of: duplication, tech chaos/autonomy, privacy loss

Conway’s law

“Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations.”

Better organise to match desired design, then.

Personae - important characteristics

Architect
- Technology updated
- Holistic: productivity, privacy
- Identify and facilitate governance

Data scientist
- Capable programmer
- Product oriented

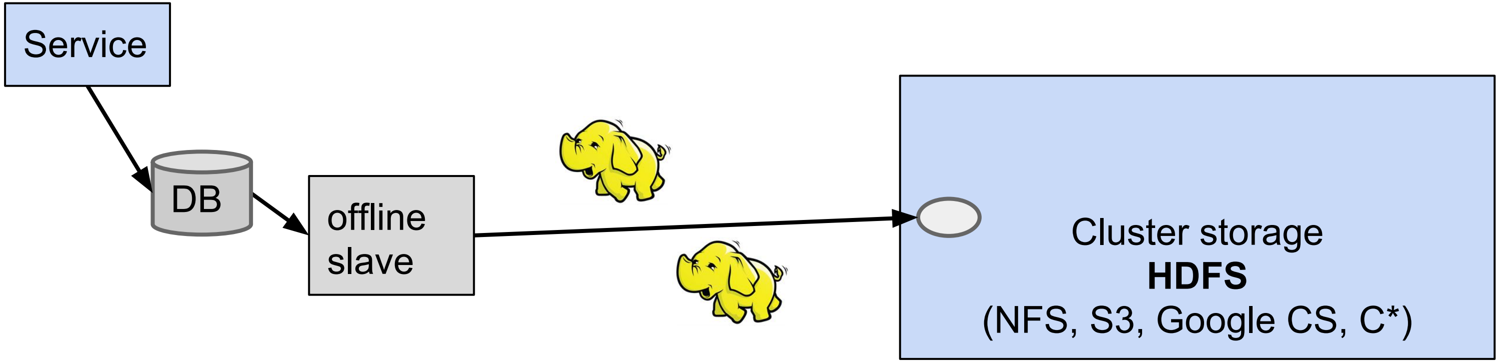
Manager
- Explain what and why
- Facilitate process to determine how
- Enable, enable, enable

Backend developer
- Simplicity oriented
- Engineering practices obsessed
- Adapt to data world

Product owner
- Trace business value to upstream design
- Find most ROI through difficult questions

Devops
- Always increase automation
- Enable, don’t control

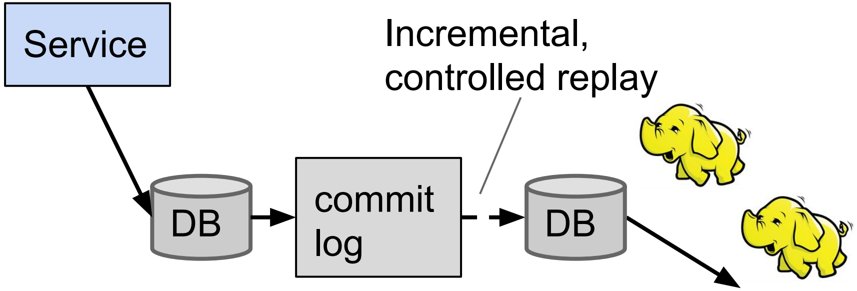
Protect production servers

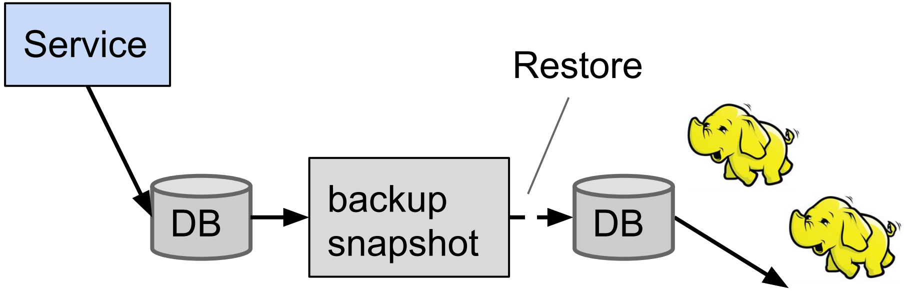


+ Online service is safe
- Replication may be out of sync
- Cluster storage may be write unavailable

=> Delayed, inaccurate snapshot

Deterministic slaves





+ Deterministic
- Ad-hoc solution
- Serial => not scalable

+ Standard procedure
- Serial or resource consuming

PII privacy control

- Simplify with coarse classification (red/yellow/green)
  - Datasets, potentially fields
  - Separate production areas
- Log batch jobs
  - Code checksum -> commit id -> source code
  - Tag job class with classification
    - Aids PII consideration in code review
    - Enables ad-hoc verification

Audit

- Audit manual access
- Wrap all functionality in gateway tool
  - Log datasets, output, code used
  - Disallow download to laptop
  - Wrapper tool happens to be great for enabling data scientists, too - shields them from operations.