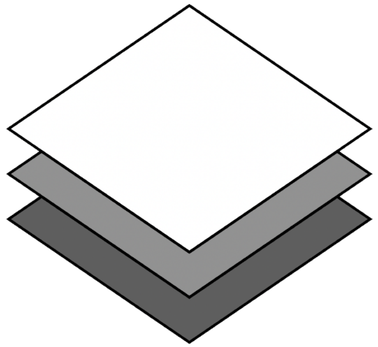


Data Management

Josh Tobin, **Sergey Karayev**, Pieter Abbeel

Amazon SageMaker · Determined AI · Neptune Machine Learning Lab · H2O · FLOYD · DOMINO DATA LAB

"All-in-one"

Data

- Versioning: Quilt, DVC, Pachyderm, LIQUIDATA
- Labeling: figure eight, scale, prodigy
- Processing: PY, Luigi
- Database
- Storage: NFS, S3

or

Development

- Distributed Training: PYTORCH, RAY, HOROVOD
- Hyperparameter Tuning: SIGOPT, tune
- Frameworks: PYTORCH, K, fast.ai
- Experiment Management: comet, Weights & Biases, TensorBoard, LOSSWISE
- Software Engineering: jupyter, git
- Resource Management: Polyaxon, PY, slurm workload manager

or

Training/Evaluation

Deployment

- Monitoring: ?
- Hardware / Mobile: ML, NVIDIA TensorRT, TensorFlow Lite
- Web: ALGORITHMIA, SELDON
- Interchange: ONNX
- CI / Testing: Buildkite, ?

or or

Katherine Scott
@kscottz

One of the biggest failures I see in junior ML/CV engineers is a complete lack of interest in building data sets. While it is boring grunt work I think there is so much to be learned in putting together a dataset. It is like half the problem.

♡ 571 11:50 AM - Feb 1, 2019

mat kelcey
@mat_kelcey

for my last few ML projects the complexity hasn't been in the modelling or training; it's been in input preprocessing. find myself running out of CPU more than GPU & in one project i'm actually unsure how to optimise the python further (& am considering c++ for one piece)

♡ 130 2:01 PM - Feb 11, 2019

Vicki Boykis
@vboykis

Have been extremely curious about this for a while now, so I decided to create a poll.
"As someone titled 'data scientist' in 2019, I spend most of (60%+) my time:"
("Other") also welcome, add it in the replies.

♡ 189 8:17 AM - Jan 28, 2019

6% Picking features/models

67% Cleaning data/Moving data

4% Deploying models in prod

23% Analyzing/presenting data

2,116 votes • Final results

https://veekaybee.github.io/2019/02/13/data-science-is-different/

In this module

1. Sources
   Where training data comes from

2. Labeling
   How to label proprietary data at scale

3. Storage

   Data (images, sound files, etc) and metadata (labels, user activity) should be stored appropriately

4. Versioning

   Dataset is updated through user activity or additional labeling, affects model

5. Processing

   Aggregating and converting raw data and metadata

In this module

1. **Sources**
   Where training data comes from

2. Labeling
   How to label proprietary data at scale

3. Storage
   Data (images, sound files, etc) and metadata (labels, user activity) should be stored appropriately

4. Versioning
   Dataset is updated through user activity or additional labeling, affects model

5. Processing
   Aggregating and converting raw data and metadata

Sources

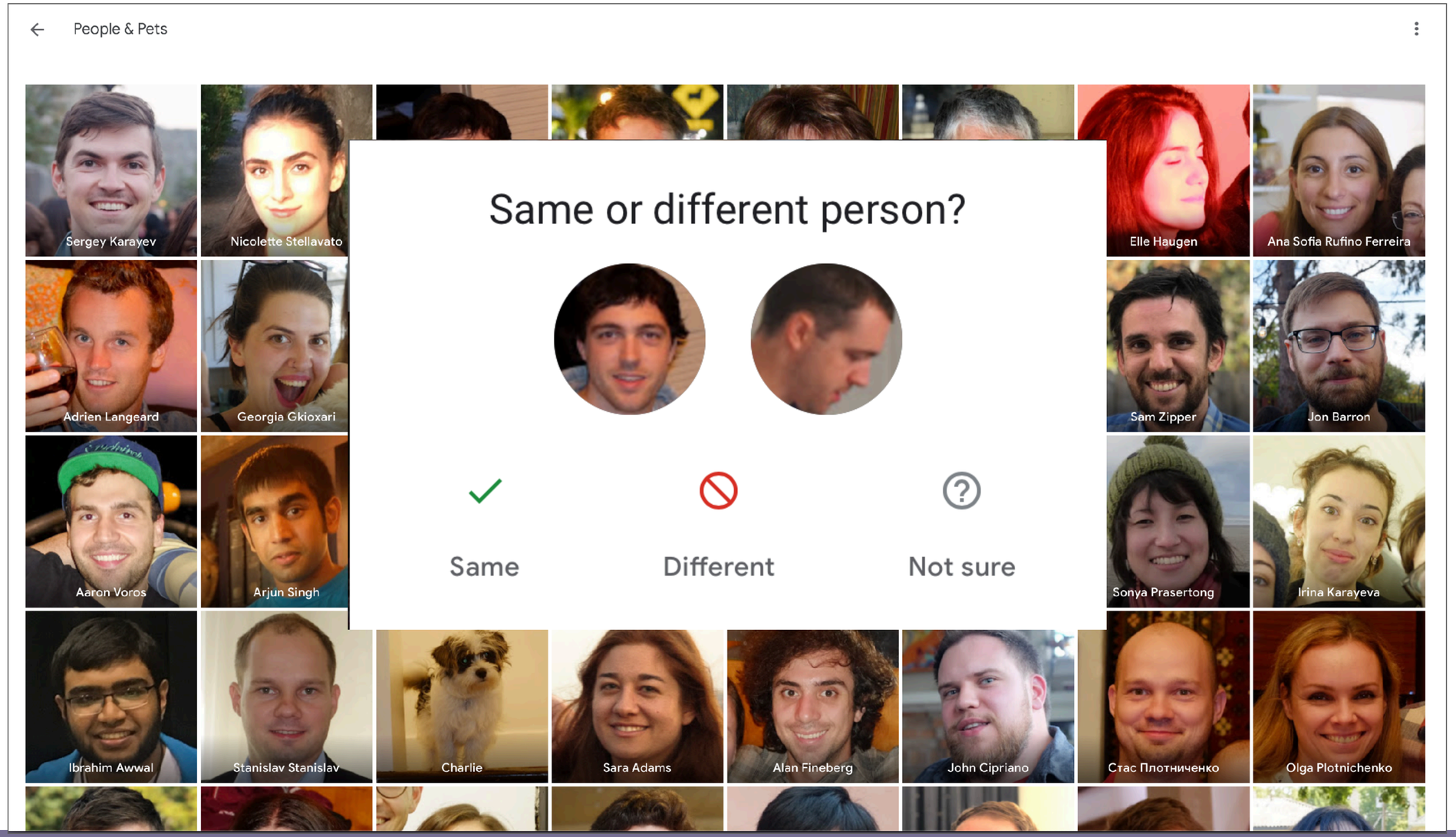
- Most DL applications require lots of labeled data
  - Exceptions: RL, GANs, "semi-supervised" learning
- Publicly available datasets = No competitive advantage
  - But can serve as starting point

Usually: spend $$$ and time to label own data

https://cdn-sv1.deepsense.ai/wp-content/uploads/2017/04/sample_image_from_the_training_set.jpg

Data flywheel

Enables rapid improvement with user labels



Semi-supervised learning

- Predict any part of the input from any other part.
- Predict the future from the past.
- Predict the future from the recent past.
- Predict the past from the present.
- Predict the top from the bottom.
- Predict the occluded from the visible
- Pretend there is a part of the input you don't know and predict that.

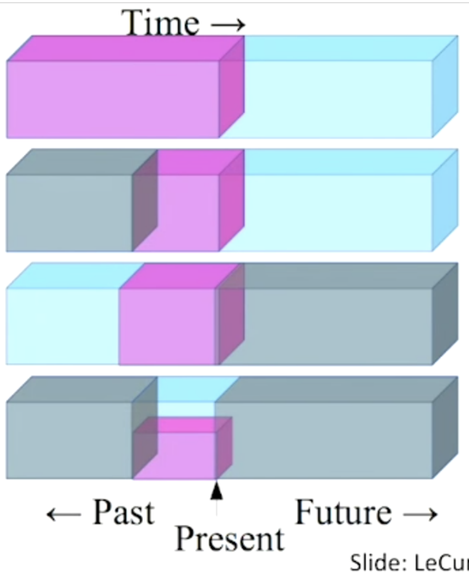
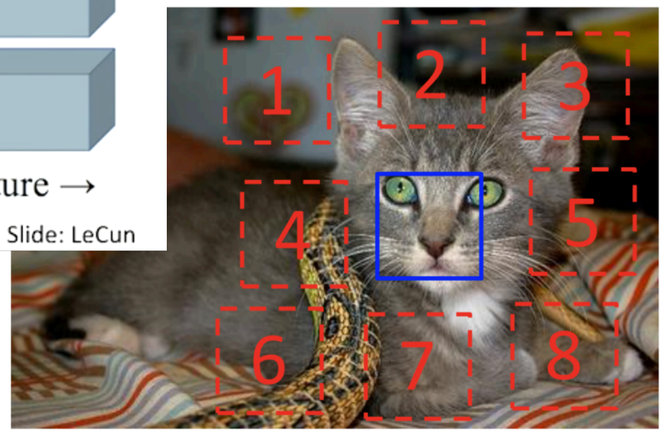
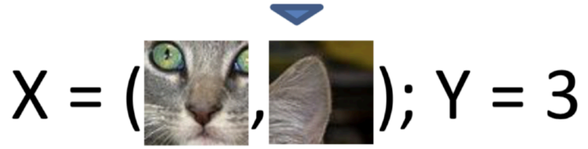


Fig. 1. A great summary of how self-supervised learning tasks can be constructed (Image source: LeCun's talk)

Use parts of data to label other parts





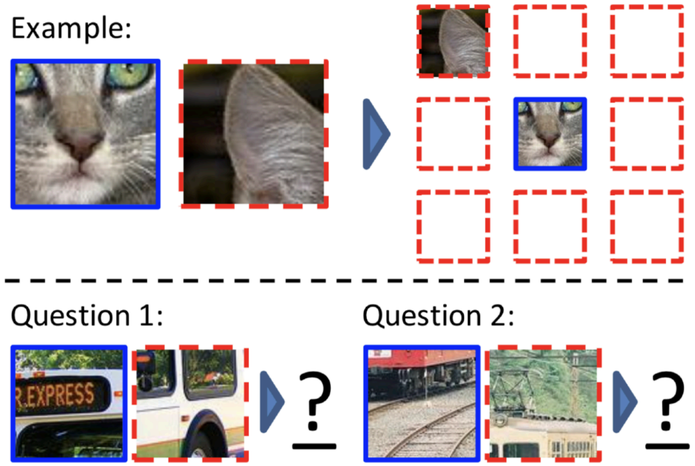


Fig. 4. Illustration of self-supervised learning by predicting the relative position of two random patches. (Image source: Doersch et al., 2015)

https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

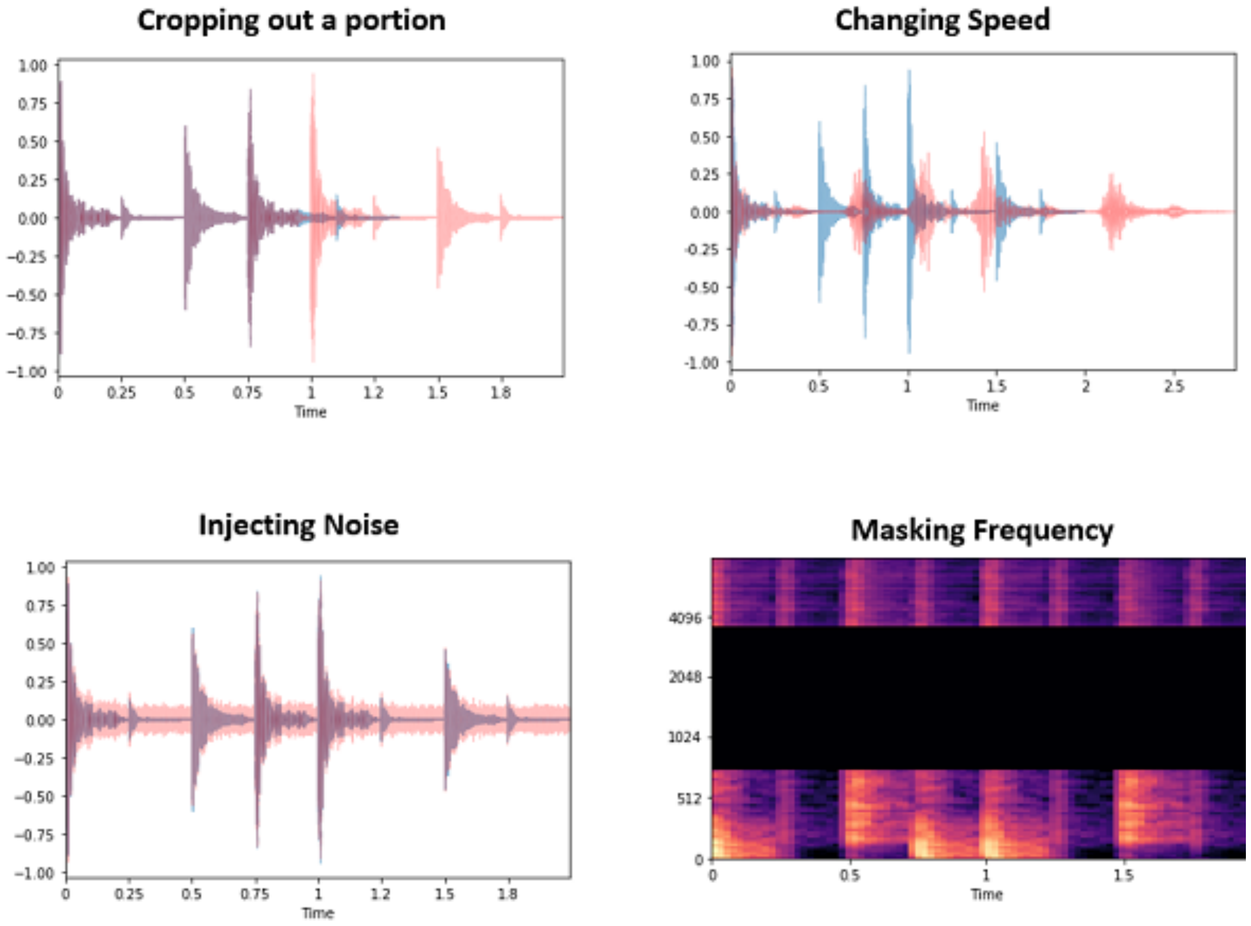
Image data augmentation

- **Must do** for training vision models
- Keras, fast.ai provide functions that do this
- Hook into your Dataset class to do in CPU workers in parallel to GPU training

https://towardsdatascience.com/1000x-faster-data-augmentation-b91bafee896c

Other data augmentation

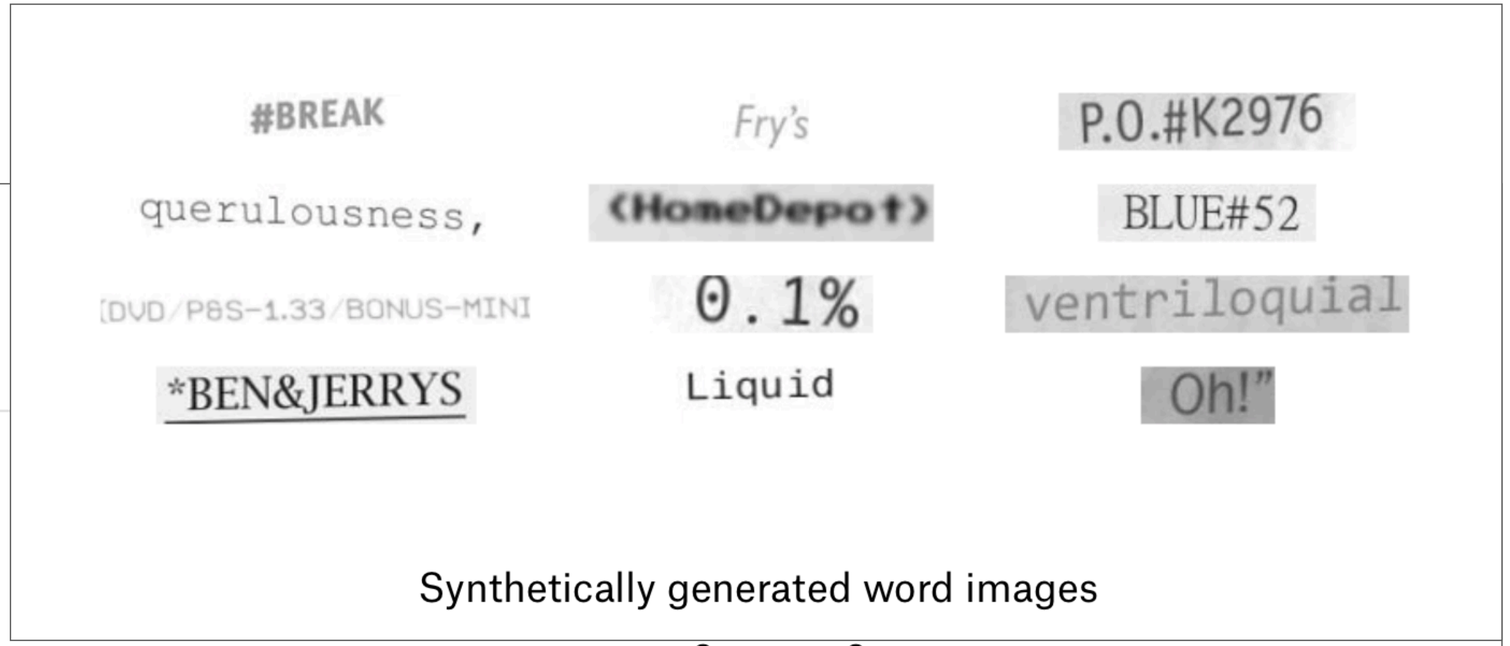
- Tabular
  - Delete some cells to simulate missing data
- Text
  - No well established techniques, but replace words with synonyms, change order of things.
- Speech/video
  - Change speed, inserts pauses, etc



https://github.com/makcedward/nlpaug

Synthetic data

Underrated idea that is almost always worth starting with



Synthetically generated word images

Dropbox

Creating a Modern OCR Pipeline Using Computer Vision and Deep Learning

Brad Neuberg | April 12, 2017

728 0

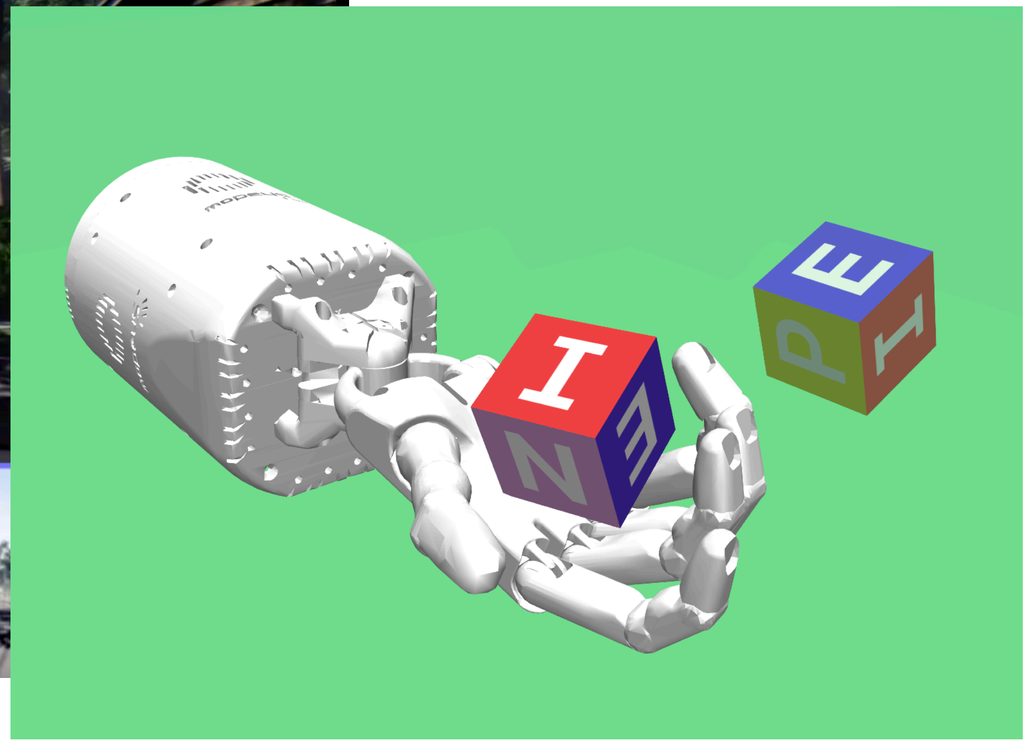
https://blogs.dropbox.com/tech/2017/04/creating-a-modern-ocr-pipeline-using-computer-vision-and-deep-learning/

This can get pretty deep!

- Receipt crinkliness
- Receipt curvature
- Receipt alignment with camera
- Receipt ink fadedness
- Receipt paper glossiness
- Table material
- Camera flash
- Camera location
- Camera direction
- Camera exposure
- Camera focal distance
- Camera aperature size
- Environment ambient brightness
- Primary light location

Andrew Moffat - https://github.com/amoffat/metabrite-receipt-tests

Especially for driving and robotics



https://microsoft.github.io/AirSim/

https://openai.com/blog/ingredients-for-robotics-research/

Questions?

In this module

1. Sources
   Where training data comes from

2. Labeling
   How to label proprietary data at scale

3. Storage
   Data (images, sound files, etc) and metadata (labels, user activity) should be stored appropriately

4. Versioning
   Dataset is updated through user activity or additional labeling, affects model

5. Processing
   Aggregating and converting raw data and metadata

Data Labeling

1. User Interfaces

2. Sources of labor

3. Service companies

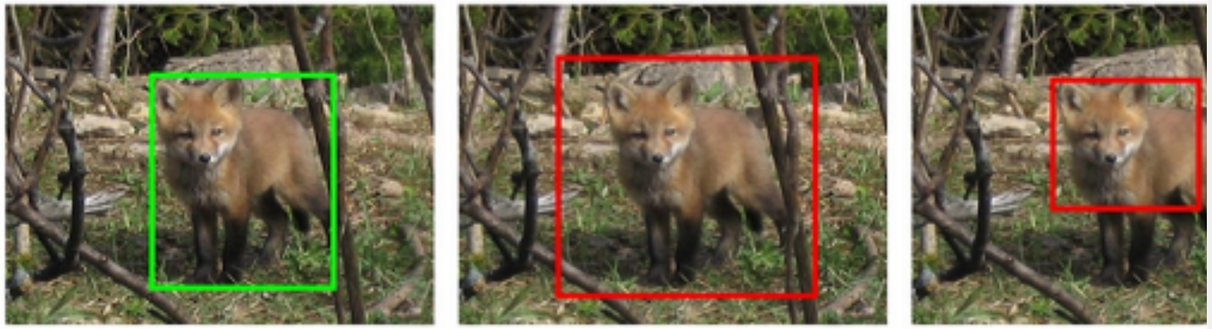
Standard set of features:

- bounding boxes, segmentations, keypoints, cuboids

- set of applicable classes

Training the annotators is crucial

Rule 1: Include all visible part and draw as tightly as possible.



CORRECT

WRONG: must be as tight as possible!

WRONG: must include al parts!

CORRECT

WRONG: occluded parts do not matter as long as all visible parts are included.

Source: cs.stanford.edu

Quality assurance is key

Sources of Labor

- Hire **own annotators**, promote best ones to quality control
  - Pros: secure, fast (once hired), less QC needed
  - Cons: expensive, slow to scale, admin overhead
- ...or, crowdsource (Mechanical Turk)
  - Pros: cheaper, more scalable
  - Cons: not secure, significant QC effort required
- ...or, full-service data labeling companies

Service Companies

- Data labeling requires separate software stack, temporary labor, and quality assurance. **Makes sense to outsource.**
- Dedicate several days to selecting the best one for you:
  - Label gold standard data yourself
  - Sales calls with several contenders, ask for work sample on same data
  - Ensure agreement with your gold standard, and evaluate on value

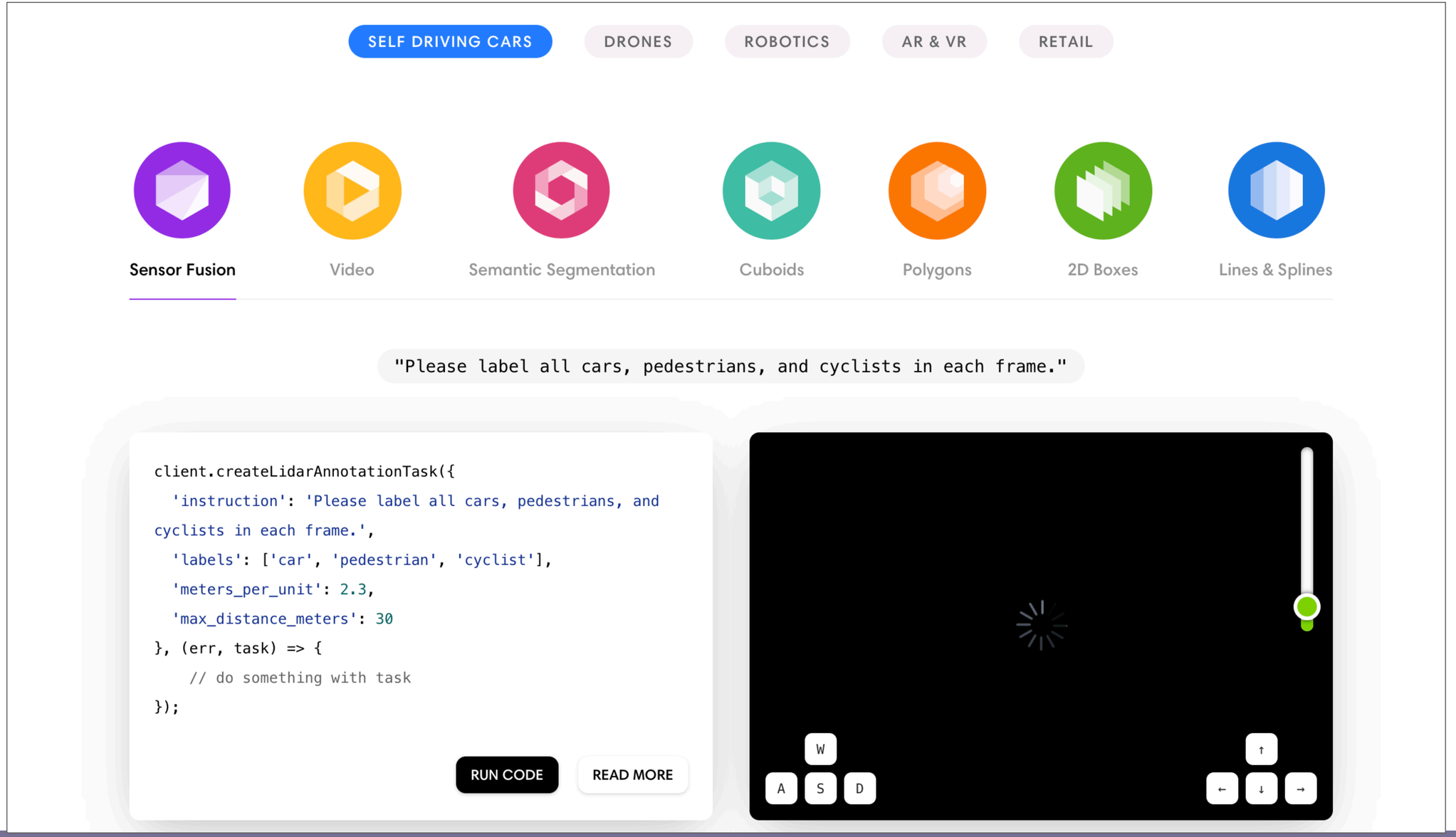
FigureEight is the original AI data labeling company

Over 100 million images labeled

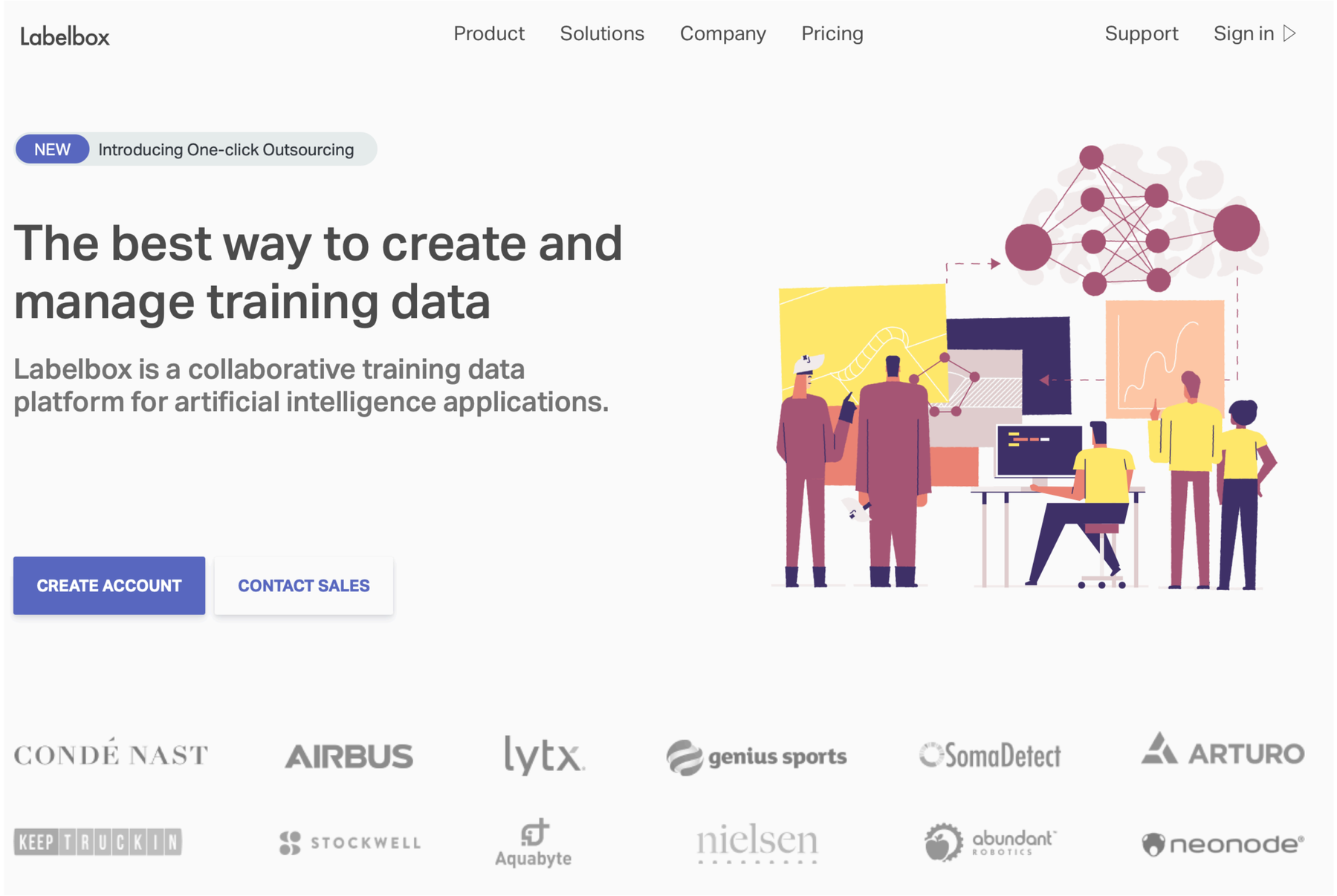
Over 1 billion human judgments in 2018

Over 10 years enabling AI projects

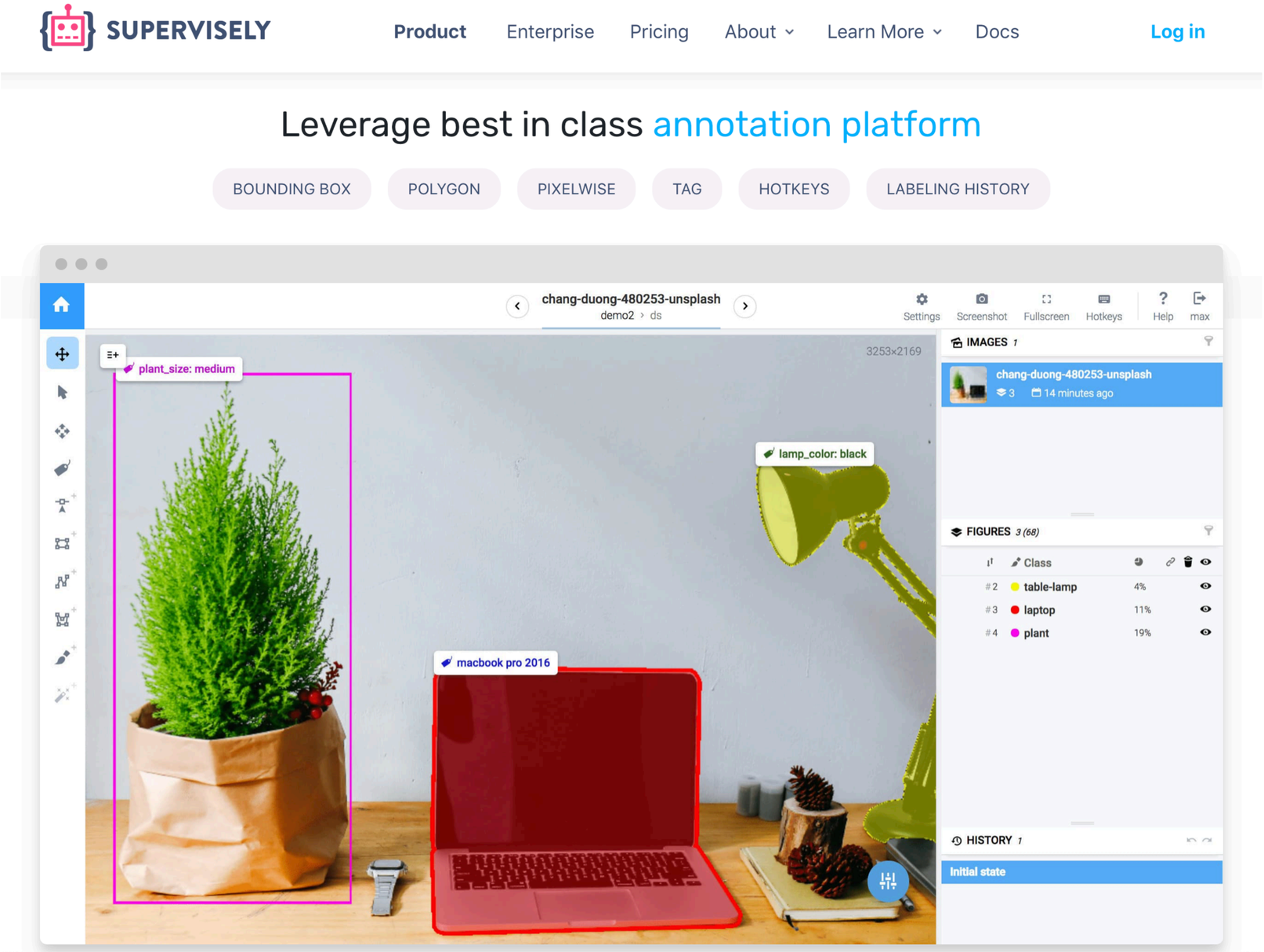
Scale.ai is a dominant up-and-comer



And there are a ton of others



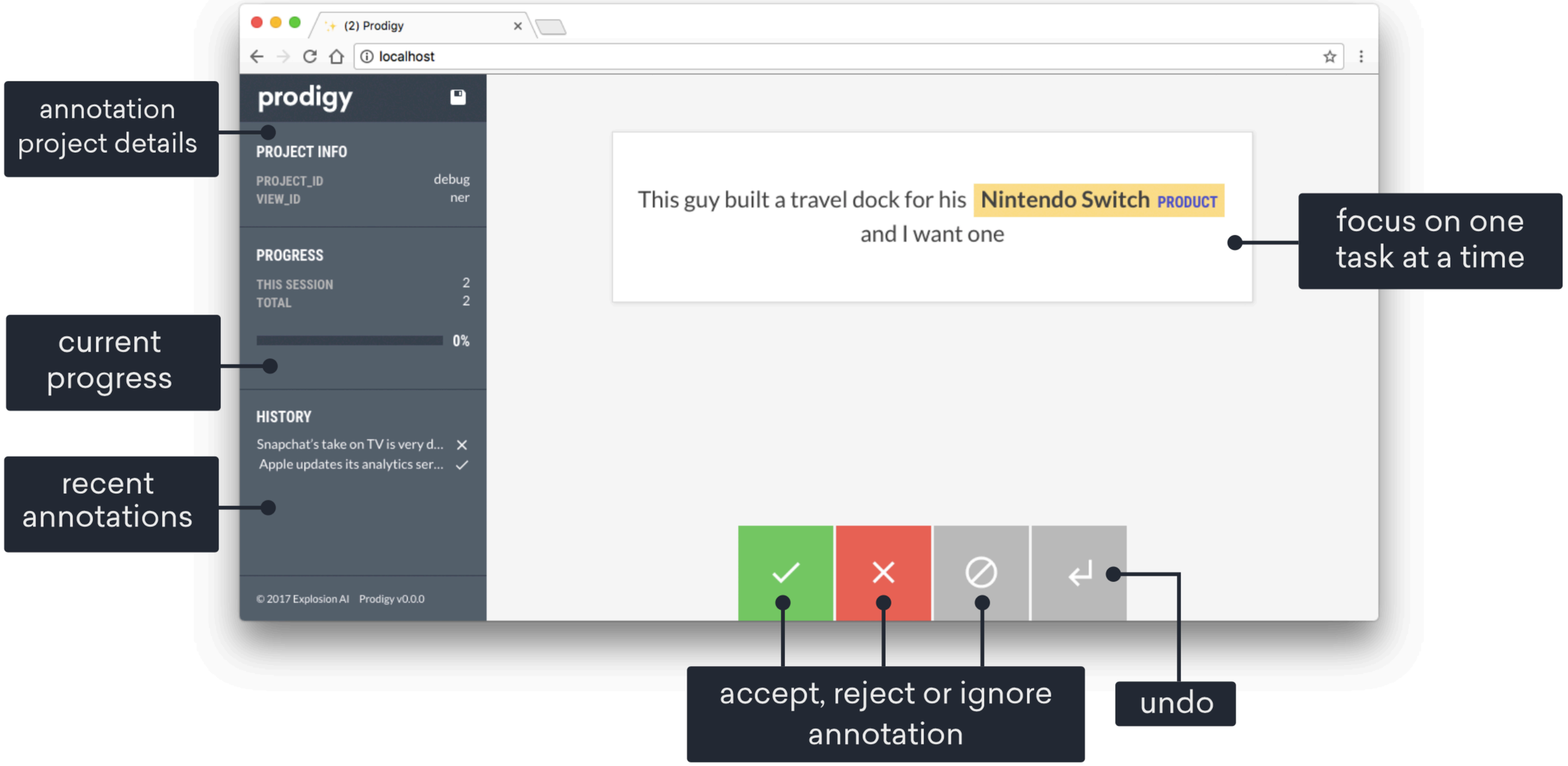
And there are a ton of others



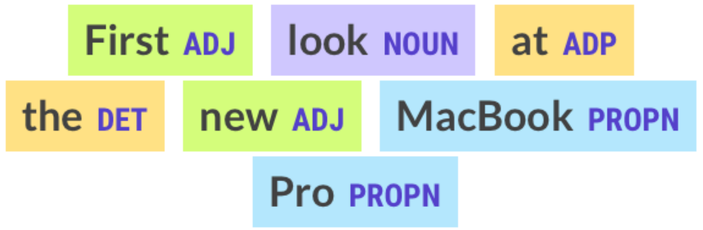
Software

- Full-service data labeling is always pricy
- But some companies offer their software without labor

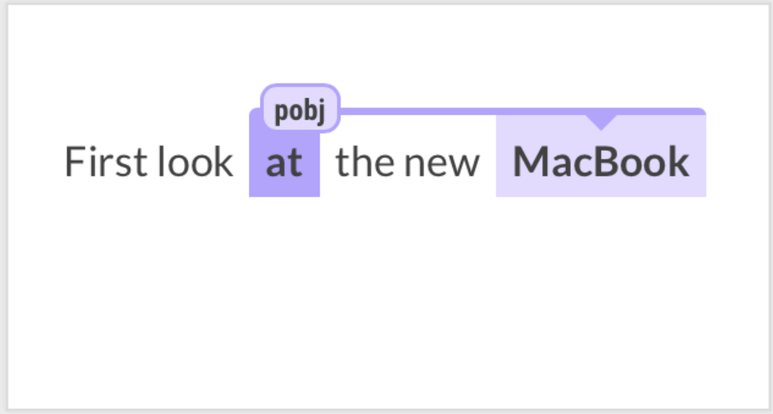
Prodigy


annotation project details
current progress
recent annotations
(2) Prodigy
localhost
prodigy
PROJECT INFO
PROJECT_ID debug
VIEW_ID ner
PROGRESS
THIS SESSION 2
TOTAL 2
0%
HISTORY
Snapchat's take on TV is very d...
Apple updates its analytics ser...
© 2017 Explosion AI Prodigy v0.0.0
This guy built a travel dock for his Nintendo Switch PRODUCT
and I want one
focus on one task at a time
accept, reject or ignore annotation
undo

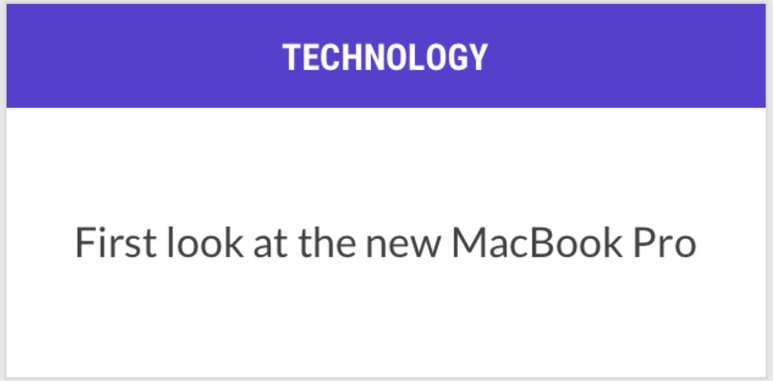
Prodigy



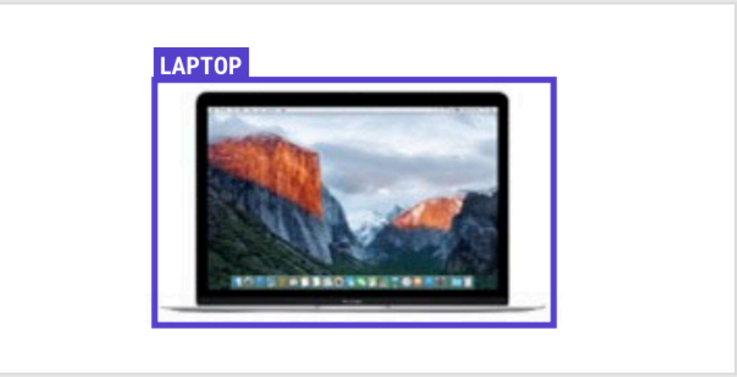
Annotate and correct part-of-speech tags



Annotate syntactic dependencies and semantic relations



Annotate labelled text or images



Annotate images, bounding boxes and image segments

Select one or more answers or pick the odd one out

Compare two annotations

- Conclusions
  - outsource to full-service company if you can afford it
  - if not, then at least use existing software
  - hiring part-time makes more sense than trying to make crowdsourcing work

Questions?

In this module

1. Sources
   Where training data comes from

2. Labeling
   How to label proprietary data at scale

3. Storage

   Data (images, sound files, etc) and metadata (labels, user activity) should be stored appropriately

4. Versioning

   Dataset is updated through user activity or additional labeling, affects model

5. Processing

   Aggregating and converting raw data and metadata

Data Storage

1. Building blocks

    - Filesystem

    - Object Storage

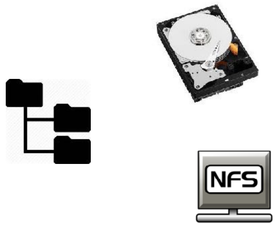
    - Database

    - "Data Lake"

2. What goes where

3. Where to learn more

Filesystem



- Foundational layer of storage.
- Fundamental unit is a "file", which can be text or binary, is not versioned, and is easily overwritten.
- Can be as simple as a locally mounted disk containing all the files you need.
- Can be networked (e.g. NFS): accessible over network by multiple machines.
- Can be distributed (e.g. HDFS): stored and accessed over multiple machines.
- Be mindful of access patterns! Can be fast, but not parallel.

Object Storage

- An API over the filesystem. GET, PUT, DELETE files to a service, without worrying where they are actually stored.
- Fundamental unit is an "object". Usually binary: image, sound file, etc.
- Versioning, redundancy can be built into the service.
- Can be parallel, but not fast.
- Amazon S3 is the canonical example.
- For on-prem setup, Ceph is a good solution.

Database



- Persistent, fast, scalable storage and retrieval of structured data.
- Mental model: everything is actually in RAM, but software ensures that everything is logged to disk and never lost.
- Fundamental unit is a "row": unique id, references to other rows, values in columns.
- Not for binary data! Store references instead.
- Usually for data that will be accessed again (not logs).
- Postgres is the right choice >90% of the time. Best-in-class SQL, great support for unstructured JSON, actively developed.

SQL

- SQL is the right interface for structured data.
- If you avoid using it, you will eventually reinvent a slow and crappy subset of it.

Overview | SQL | Display Table | Average num... | Average fract..

Run | Limit 100 | Format SQL | View History...

```
1  select
2    format('/courses/%s/assignments/%s/submission_batches', a.course_id, a.id) as url,
3    round(sum((b.data->>'confidence')::numeric) / count(b), 2) as fraction_confident,
4    sum(case when (b.data->>'confidence')::numeric = 1
5             and b.data->'suggested_page_indices_list' = b.data->'actual_page_indices_list'
6             then 1 else 0 end) / nullif(sum((b.data->>'confidence')::numeric), 0) as confident_accuracy,
7    max(t.page_count) as template_length,
8    count(b) as num_scans,
9    percentile_cont(0.5) within group (order by p.page_count) as median_scan_length,
10   array_agg(distinct p.page_count) as unique_scan_lengths
11
12 from assignment_submission_batches b
13 inner join assignments a on a.id = b.assignment_id
14 inner join courses c on a.course_id = c.id
15 inner join pdf_attachments t on t.attachable_id = a.id and t.attachable_type = 'Assignment'
16 inner join pdf_attachments p on p.attachable_id = b.id and p.attachable_type = 'AssignmentSubmissionBatch'
17
18 where a.created_at > '2018-05-04'
19 and   a.type = 'PDFAssignment'
20 and   c.demo = false
21
22 group by a.id
```

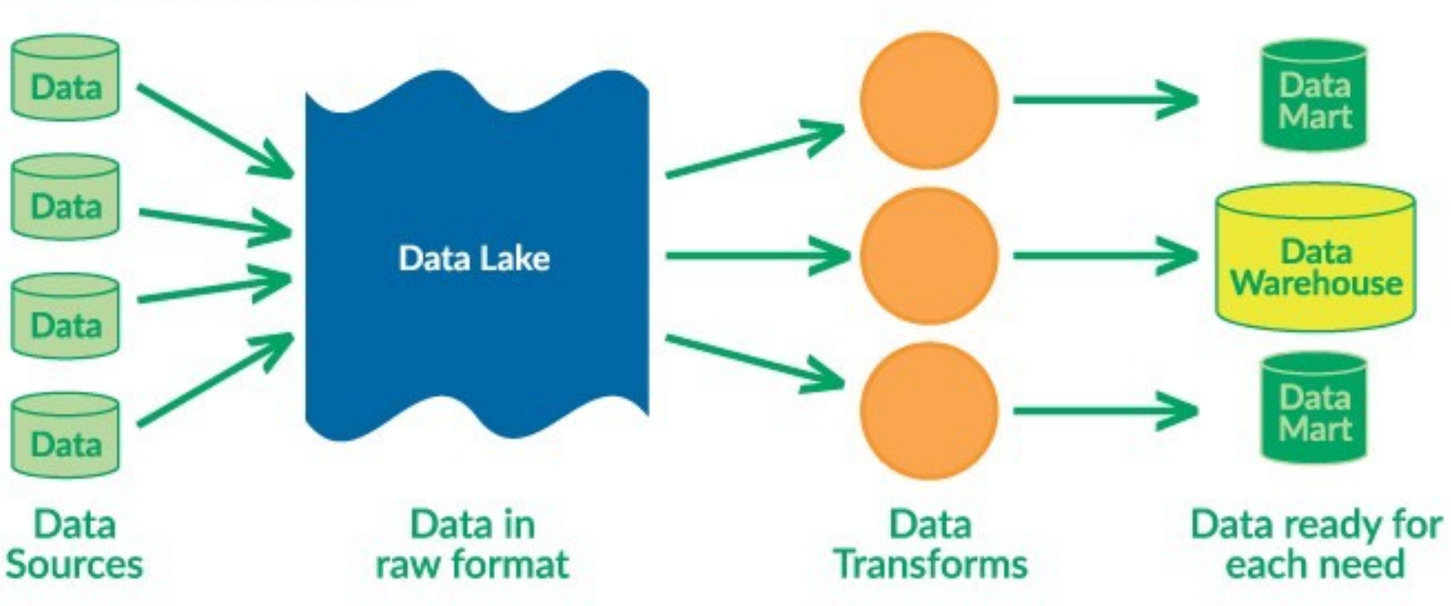
Export | Copy | Chart | Pivot

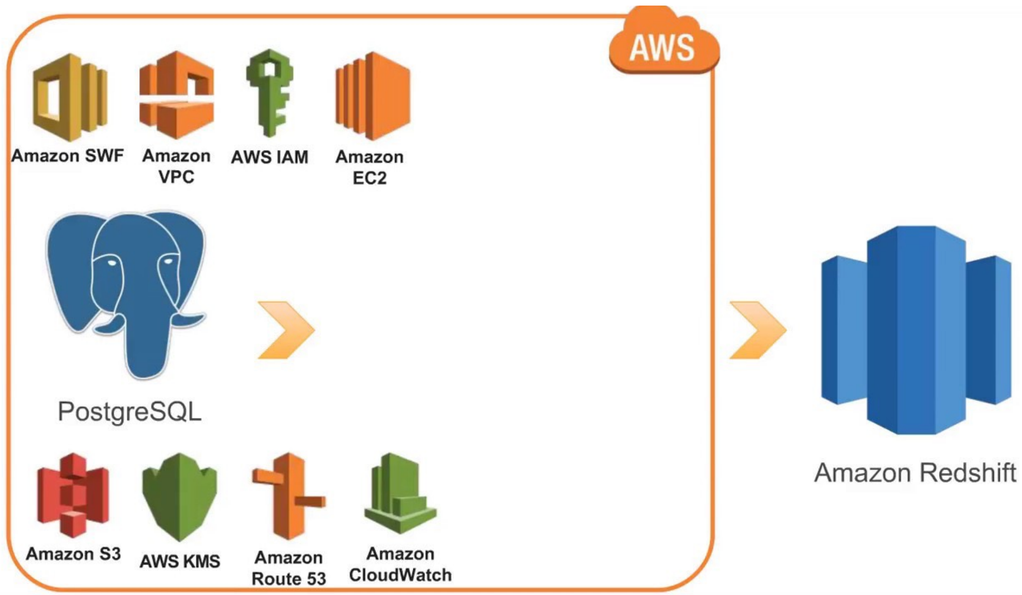
	url	fraction_confident	confident_accuracy	template_length	num_scans	median_scan_length
1	/courses/17773/assignments/76593/submission_batches	1	1	6	7	114
2	/courses/13919/assignments/76596/submission_batches	0.5	0	28	72	28
3	/courses/17503/assignments/76599/submission_batches	1	1	1	1	26

"Data Lake"

- Unstructured aggregation of data from multiple sources, e.g. databases, logs, expensive data transformations.

- "Schema on read": dump everything in, then transform for specific needs later.



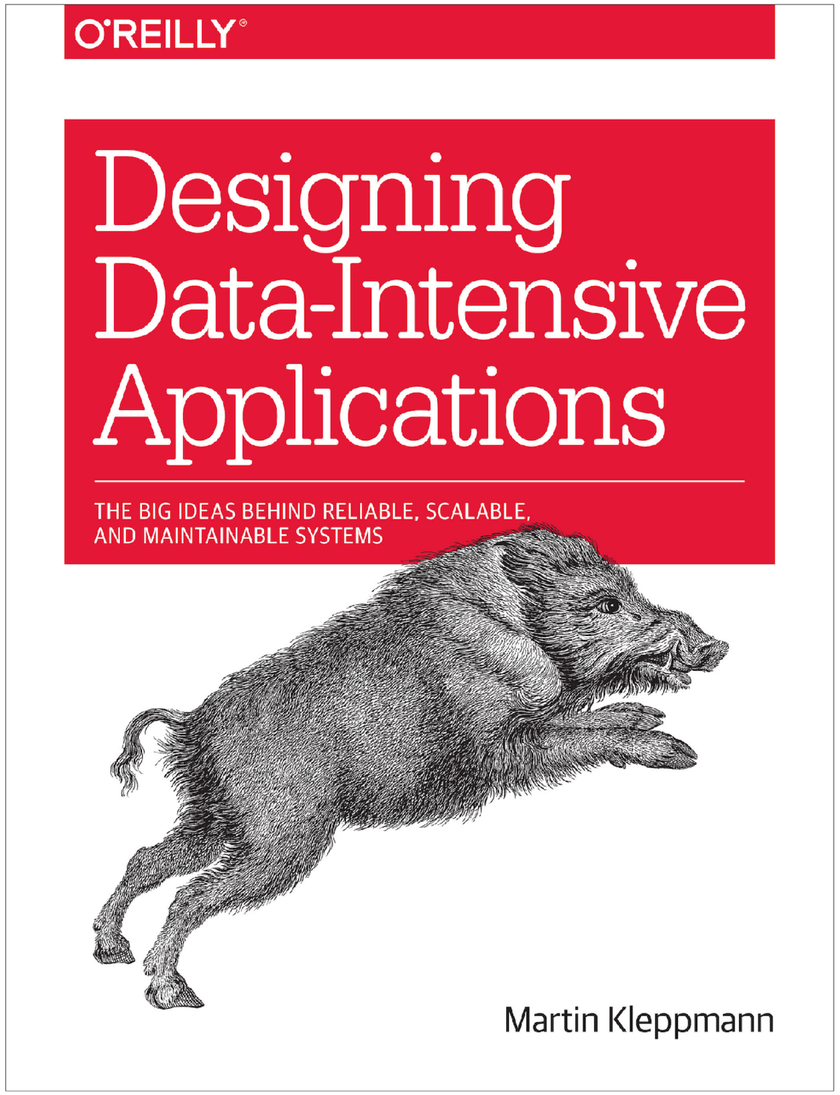


https://medium.com/data-ops/throw-your-data-in-a-lake-32cd21b6de02

What goes where

- **Binary data** (images, sound files, compressed texts) is stored as **objects**.
- **Metadata** (labels, user activity) is stored in **database**.
- If need features which are not obtainable from database (**logs**), set up **data lake** and a process to aggregate needed data.
- At **training time**, copy the data that is needed onto **filesystem** (local or networked).

Where to learn more



Don't just hack it together

NoSQL… Big Data… Scalability… CAP Theorem… Eventual Consistency… Sharding…

Nice buzzwords, but how does the stuff actually *work*?

https://dataintensive.net

Questions?

In this module

1. Sources
   Where training data comes from

2. Labeling
   How to label proprietary data at scale

3. Storage

   Data (images, sound files, etc) and metadata (labels, user activity) should be stored appropriately

4. Versioning

   Dataset is updated through user activity or additional labeling, affects model

5. Processing

   Aggregating and converting raw data and metadata

Data Versioning

Level 0: unversioned

Level 1: versioned via snapshot at training time

Level 2: versioned as a mix of assets and code

Level 3: specialized data versioning solution

Level 0

- Data lives are on filesystem/S3 and database
- **Problem:** Deployments must be versioned. Deployed machine learning models are part code, part data. If data is not versioned, deployed models are not versioned.
- Problem you **will** face: inability to get back to a previous level of performance

Level 1

- Data is versioned by storing a snapshot of everything at training time
- This allows you to version deployed models, and to get back to past performance, but is super hacky.
- Would be far better to be able to version data just as easily as code.

Level 2

- Data is versioned as a mix of assets and code.
- Heavy files stored in S3, with unique ids. Training data is stored as JSON or similar, referring to these ids and include relevant metadata (labels, user activity, etc).
- JSON files can get big, but using git-lfs lets us store them just as easily as code
    - Can improve further with "lazydata": only syncing files that are needed.
- The git signature + of the raw data file defines the version of the dataset
    - Often helpful to add timestamp

Level 3

- Specialized solutions for versioning data.
- Avoid these until you can fully explain how they will improve your project.
- Leading solutions are DVC, Pachyderm, Quill.

DVC

Open-source
Version Control System
for Machine Learning Projects

1

```
$ dvc add data.xml
```

DVC stores information about your data file in a special `.dvc` file, that has a human-readable description and can be committed to Git to track versions of your file:

```
$ git add .gitignore data.xml.dvc
$ git commit -m "add source data to DVC"
```

2

```
$ dvc run \
    -d prepare.py -d data.xml \
    -o data.tsv -o data-test.tsv \
    python prepare.py data.xml
```

3

The first stage, feature extraction:

```
$ dvc run -d featurization.py -d data.tsv \
          -o matrix.pkl \
          python featurization.py data.tsv matrix.pkl
```

The second stage, training:

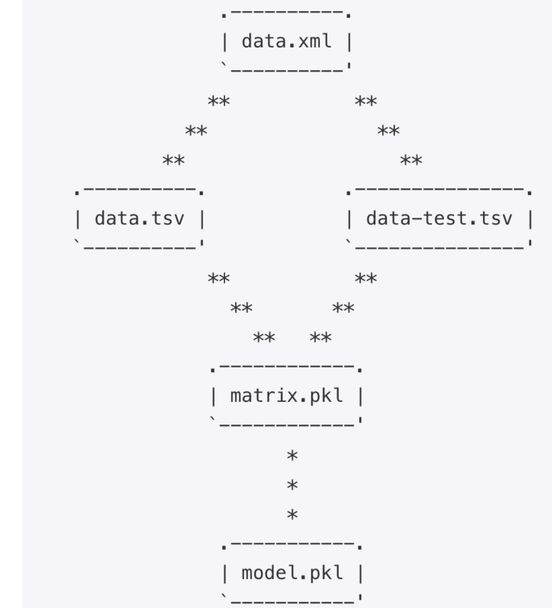
```
$ dvc run -d train.py -d matrix.pkl \
          -o model.pkl \
          python train.py matrix.pkl model.pkl
```

Let's commit meta-files that describe our pipeline:

```
$ git add .gitignore matrix.pkl.dvc model.pkl.dvc
$ git commit -m "add featurization and train steps to the pipe
```

4

```
$ dvc pipeline show --ascii model.pkl.dvc --outs
```



Pachyderm

Version control for data

Pachyderm version controls data, similar to what Git does with code. You can track the state of your data over time, backtest models on historical data, share data with teammates, and revert to previous states of data. **Learn more →**

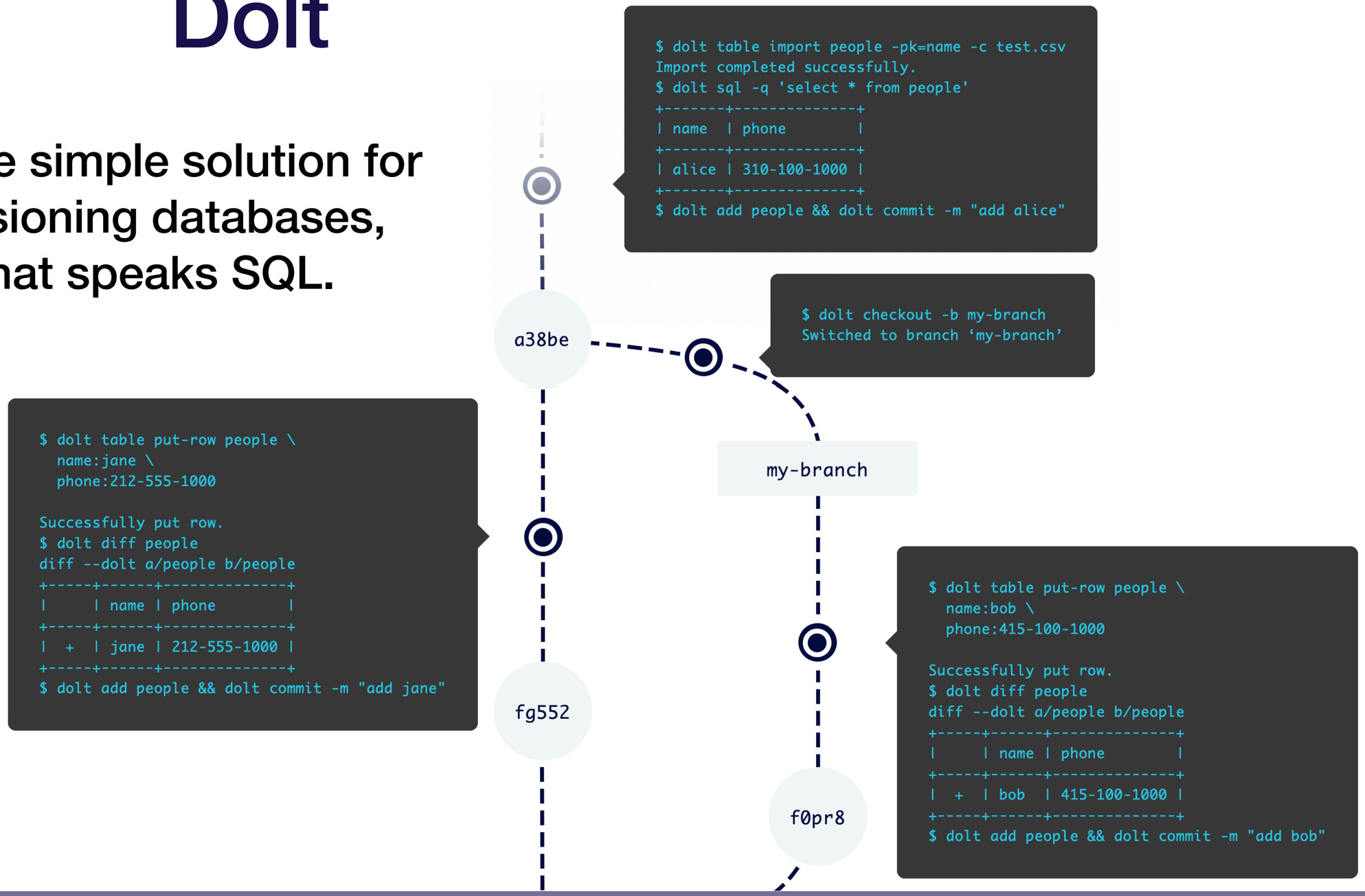
Language-agnostic data pipelines

Pachyderm lets you use the tools and frameworks you need, from bash scripts to Tensorflow. You just declaratively tell Pachyderm what you want to run, and Pachyderm takes care of triggering, data sharding, parallelism, and resource management on the backend. **Learn more →**

LIQUIDATA

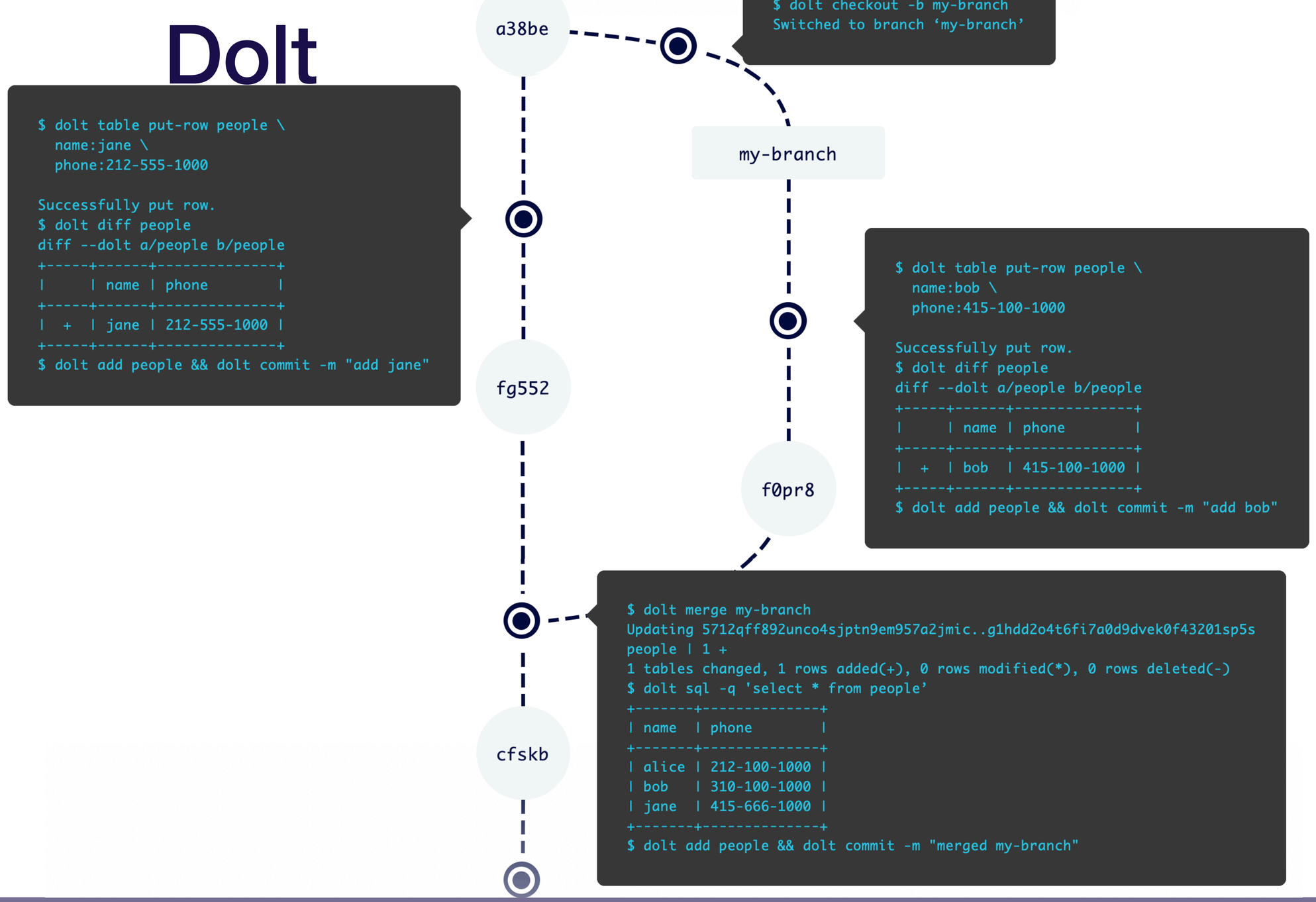
Dolt

A nice simple solution for versioning databases, that speaks SQL.



LIQUIDATA

Dolt



Questions?

In this module

1. Sources
   Where training data comes from

2. Labeling
   How to label proprietary data at scale

3. Storage

   Data (images, sound files, etc) and metadata (labels, user activity) should be stored appropriately

4. Versioning

   Dataset is updated through user activity or additional labeling, affects model

5. Processing

   Aggregating and converting raw data and metadata

Motivational Example

- We have to train a photo popularity predictor every night.
- For each photo, training data must include:
  - Metadata such as posting time, title, location
  - Some features of the user, such as how many times they logged in today.
  - Outputs of photo classifiers (content, style)

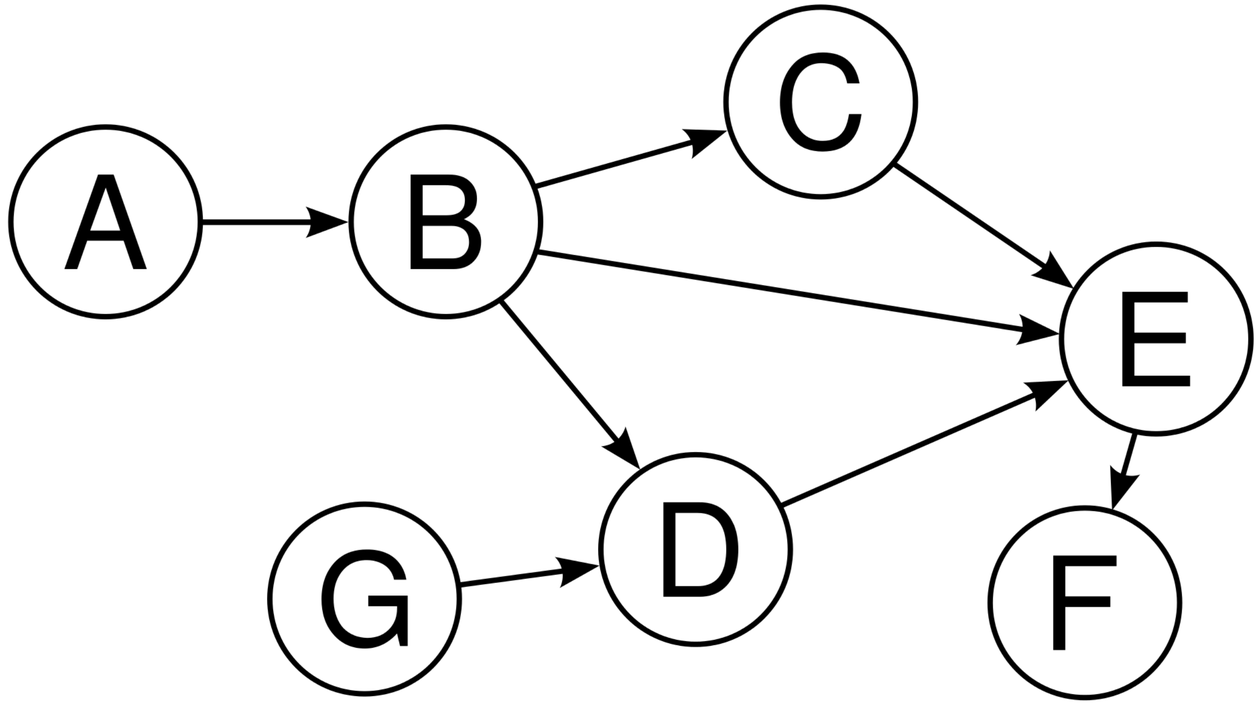
In database

Need to compute from logs

Need to run classifiers

Task Dependencies

- Some tasks can't be started until other tasks are finished.
- Finishing a task should "kick off" its dependencies



- Run everything from scratch. If things exist, no need to recompute.

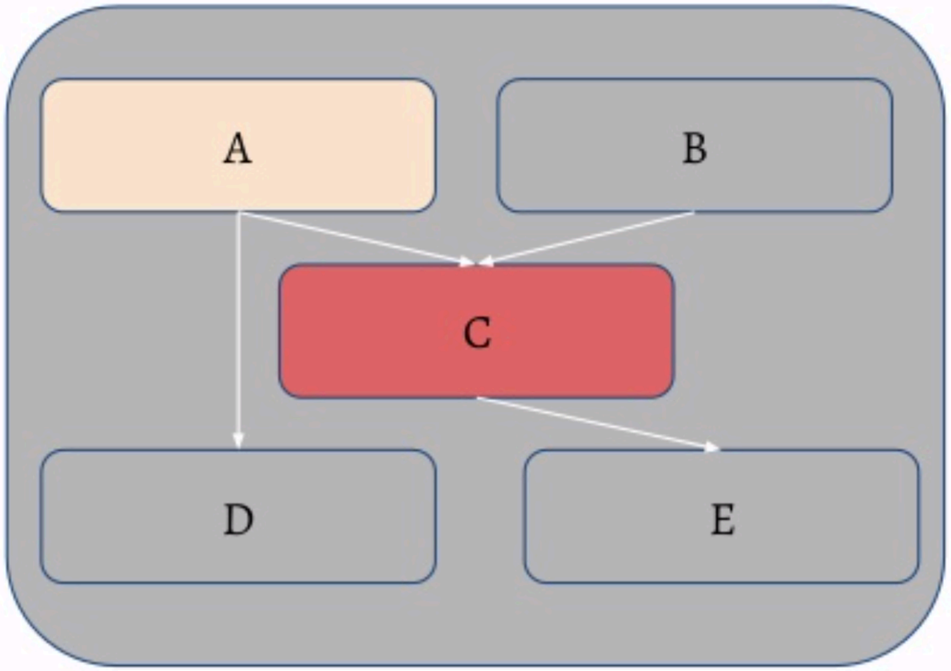

```
#Makefile
wrangled.csv : source1.csv source2.csv
    cat source1.csv source2.csv > wrangled.csv
```

https://www.slideshare.net/PyData/how-i-learned-to-time-travel-or-data-pipelining-and-scheduling-with-airflow-67650418

Makefile limitations

- What if re-computation needs to depend on content, not on date?
- What if the dependencies are not files, but disparate programs and databases?
- What if the work needs to be spread over multiple machines?
- What if there are 100s of "Makefiles" all executing at the same time, with shared dependencies?

Airflow

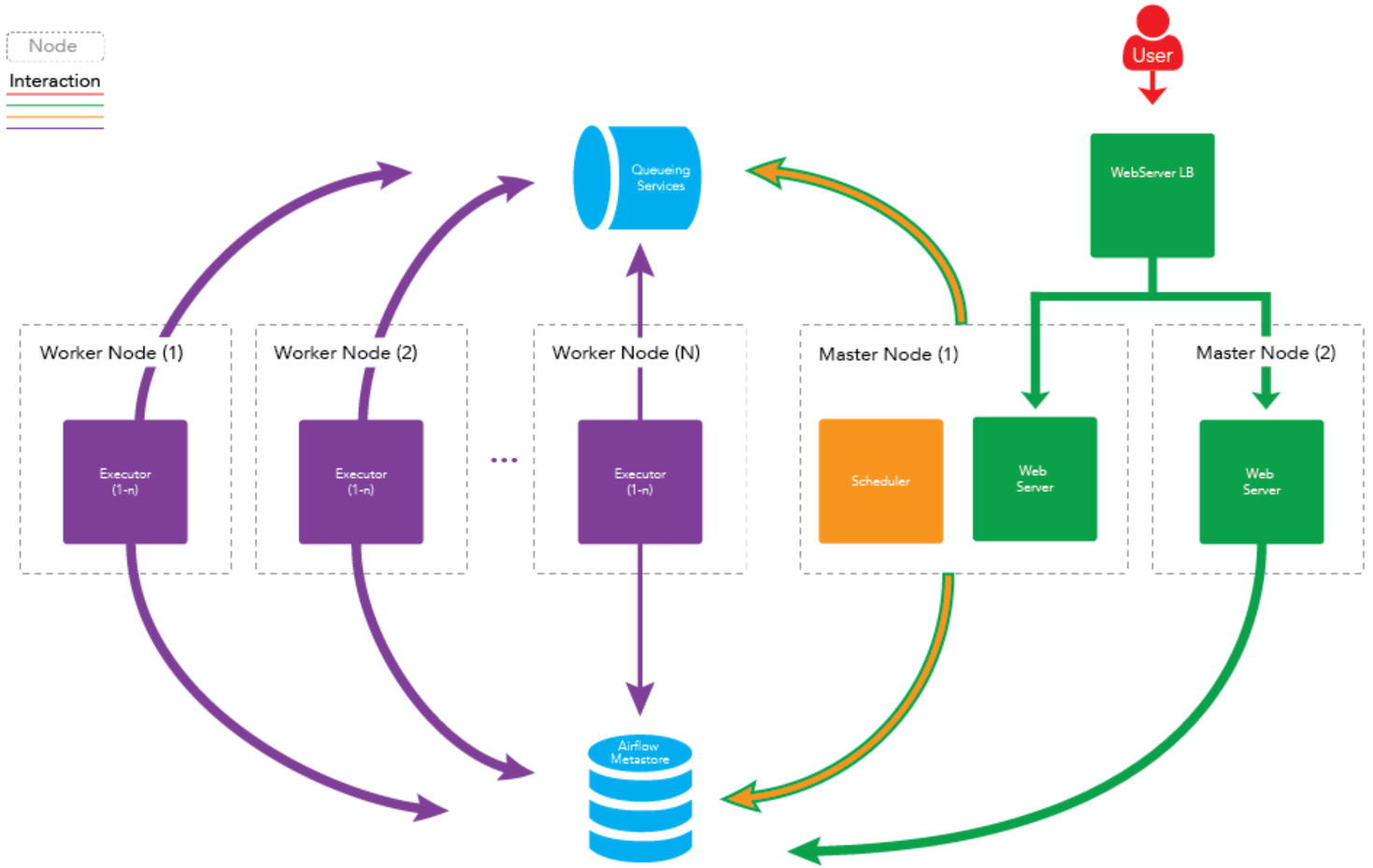



```
# Airflow
dag = DAG(schedule_interval=
              timedelta(days=1),
            start_date=
              datetime(2015,10,6))

a =  PythonOperator(
          task_id="A",
          python_callable=ClassA,
          dag=dag)
c =  MySQLOperator(
          task_id="B",
          sql="DROP TABLE hello",
          dag=dag)

c.set_upstream(a)
```

https://www.slideshare.net/PyData/how-i-learned-to-time-travel-or-data-pipelining-and-scheduling-with-airflow-67650418

Distributing work

- The workflow manager has a queue for the tasks, and manages workers that pull from it, restarting jobs if they fail.



http://site.clairvoyantsoft.com/making-apache-airflow-highly-available/

Try to keep things simple

- Don't overengineer
- e.g. unix: powerful parallelism, streaming, highly optimized

Command-line Tools can be 235x Faster than your Hadoop Cluster

January 18, 2014

26 minutes

in parallel

```
cat *.pgn | grep "Result" | sort | uniq -c
```

70 seconds

in parallel

18 seconds

```
find . -type f -name '*.pgn' -print0 \
  | xargs -0 -n4 -P4 mawk '/Result/ { split($0, a, "-"); res = substr(a[1], length(a[1]), 1); if (
    res == 1) white++; if (res == 0) black++; if (res == 2) draw++ } END { print white+black+draw,
    white, black, draw }' \
  | mawk '{games += $1; white += $2; black += $3; draw += $4; } END { print games, white, black,
    draw }'
```

https://adamdrake.com/command-line-tools-can-be-235x-faster-than-your-hadoop-cluster.html

Questions?

Thank you!