CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

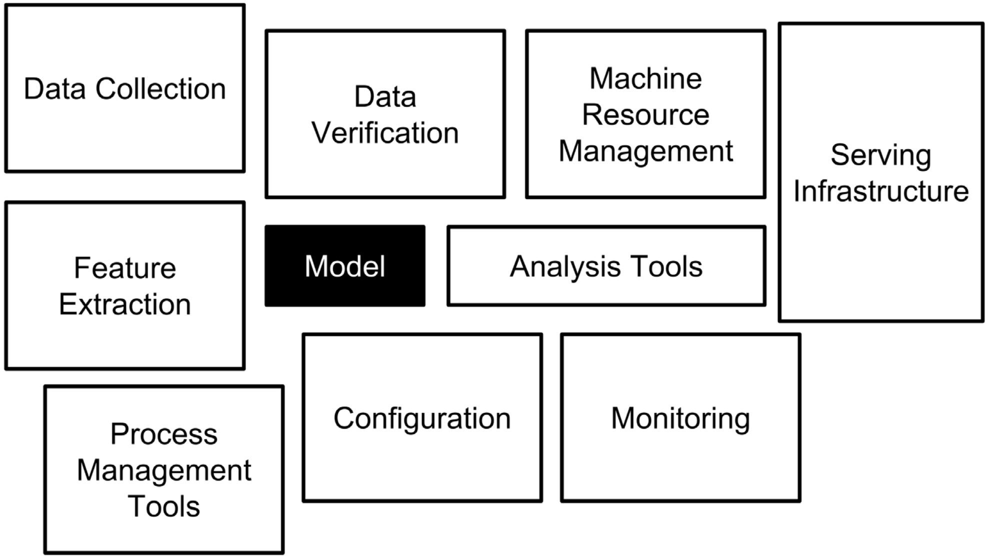
Engineering of ML Systems
Part 2

DIT826

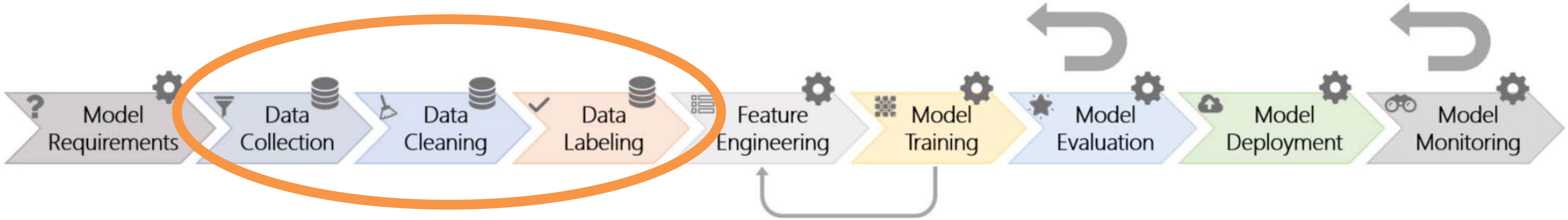
Daniel Strüber

Learning objectives

- **Understand** that building ML systems is more than training a model
- **Understand** practices and challenges of ML systems engineering

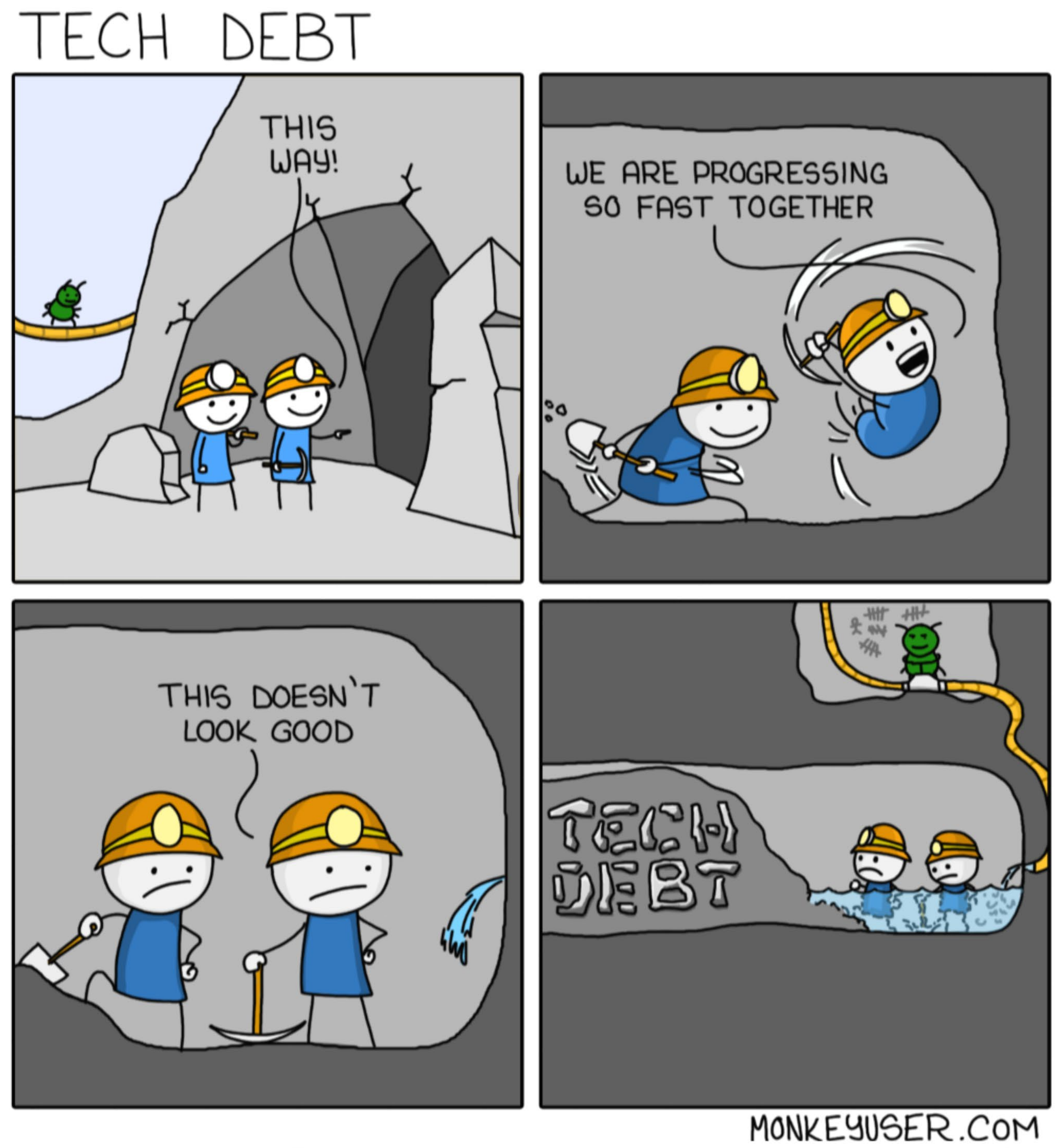


The ML Workflow



- Working with data:
  - More issues
  - Mitigations

CHALMERS
UNIVERSITY OF TECHNOLOGY



SW engineering issues ⇢ data

Unstable dependencies

- SW design principle: *“minimize dependency to unstable modules”*

- Unstable data dependencies
  - Quality or quantity of data source deteriorates
  - Format or schema of data sources changes
- Especially problematic when models are retrained frequently

SW engineering issues ⇢ data

Unstable dependencies

- Mitigation: Create *versioned copy*
  (i.e., decoupling)

- Mitigation: Monitor for upstream instability in features
  - What alert would fire if one datacenter stops sending data?
  - What if an upstream signal provider did a major version upgrade?

CSV validator

Validation of CSV file

- https://pypi.org/project/csvvalidator

```
# import everything from the csvvalidator package
from csvvalidator import *

# Specify which fields (columns) your .csv needs to have
# You should include all fields you use in your dashboard
field_names = ('date',
               'units_sold',
               'store'
               )
# create a validator object using these fields
validator = CSVValidator(field_names)
```

CSV validator

```
# write some checks to make sure specific fields
# are the way we expect them to be

validator.add_value_check(
    'date', # check for a date with the sepecified format
    datetime_string('%Y-%m-%d'),
    'EX1', # code for exception
    'invalid date' # message to report if error thrown
)
validator.add_value_check(
    'units_sold',
    int,
    'EX2',
    'number of units sold not an integer'
)
validator.add_value_check(
     'store',
    enumeration('store1', 'store2'),
    'EX4',
    'store name not recognized’
)
problems = validator.validate(data)
```

SW engineering issues ⇢ data

Unnecessary dependencies

- SW design principle: *“keep it simple”*

- Unnecessary data dependencies
  - Correlated features
  - ϵ-features (small improvement)
  - Forgotten features (from previous experiment)*

*Rule #22 – Clean up features you are no longer using, in “M. Zinkevich, Rules of Machine Learning”

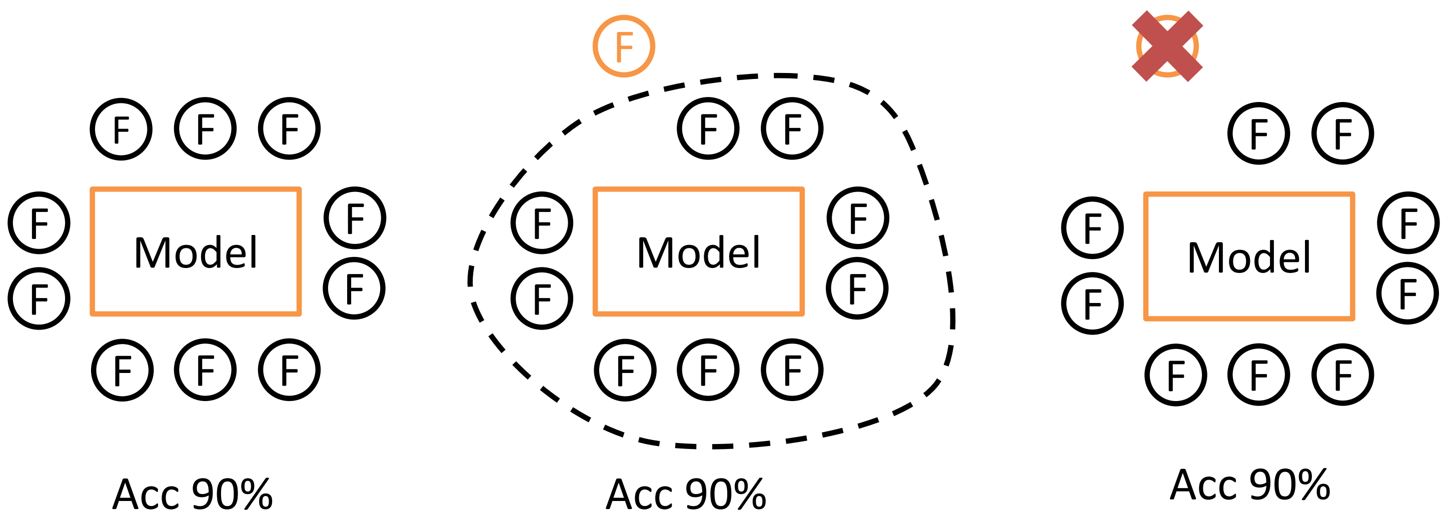
SW engineering issues ⇢ data

Unnecessary dependencies

- Issue of cost
  - Storage cost
    (esp. if data is massively large)
    (esp. if data is versioned)
  - Computational cost
  - Maintenance cost
  - Monitoring cost

SW engineering issues ⇢ data
Unnecessary dependencies

- One Solution:
  - Feature selection via exhaustive '*leave-one-feature-out*' evaluations.



SW engineering issues ⇢ data

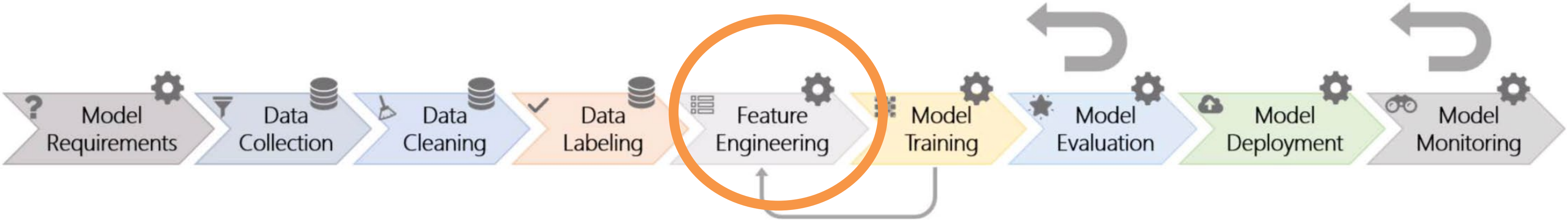
- SW design principle: *"avoid module dependency loops"*

- Hidden feedback loops
  - Output of model A → fed to model B
  - Output of model B → fed *back* to model A

- Mitigation
  - Reduce dependency
  - Better understand the problem that you want to solve
  - Testing

Questions?

The ML Workflow



- Working with data:
  - **Feature Engineering**

Feature Engineering

- The art of converting context into features that *work well* for your problem and with your model structure.

Data + Context → Feature Engineering → Features ☑ Problem

Feature Engineering

- Normalization and standardization: changing numerical features so they are comparable

$$\textit{Standardized } \boldsymbol{f} = \frac{\boldsymbol{f} - \mu}{\sigma}$$

Mean

Std. Deviation

- Expose hidden Information

- Expand the context

Feature Engineering

Feature Selection

- *Which features to use?*
  - Use features that will help your model to generalize.

- *How many features to use?*
  - Approaches:
    - Frequency
    - Accuracy
    - …

Feature Selection

Frequency

- Pick top N most common features in the training set.

Feature with a count > 1000

Feature	Count
to	1745
you	1526
I	1369
a	1337
the	1007
and	758
in	400
…	…

Feature Selection
Accuracy

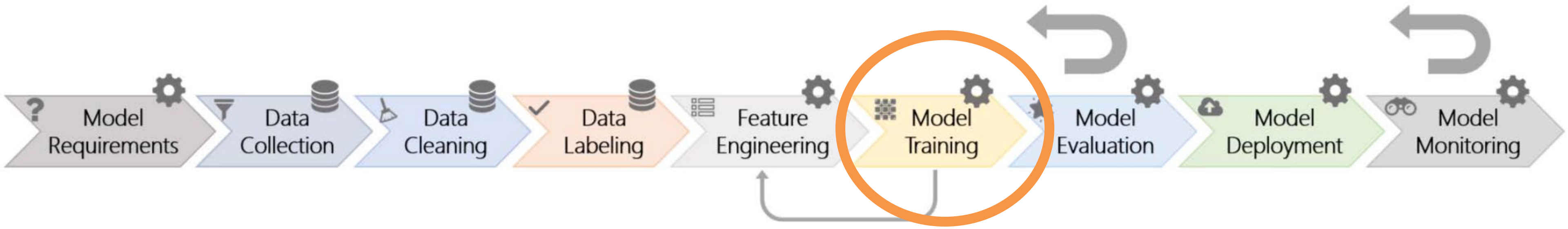
- Pick N that improves the accuracy of the model.
  - Maximize the accuracy gain from using these features.

- Greedy search, adding/**removing** features into/**from** the model:
  - Add (**remove**) a candidate feature into/**from** the model.
  - Build and train the model and evaluate it.
  - Check the accuracy.
  - Add (**remove**) the feature.
  - Repeat until you get n.

Remove	**Accuracy**
<None>	88.2%
to	92.5%
FREE	85.8%
…	…

Questions?

The ML Workflow



- Working with data:
  – Model Training

A Challenge…

- Different expertise is necessary
  - Software development
  - Data management (large data)
  - Machine learning

- Different teams might work in silos
  - Data engineers: building data pipeline
  - Data scientist: selecting and tuning the algorithm
  - Web developers: building the client/server

…consequences

- Models might only work in the lab
  - Never leave the *proof-of-concept* state
- Hard to update and maintain.
- Many are advocating for CD practices applied to production ML systems*

* D. Sato, A. Wider, C. Windheuser, Continuous delivery for machine learning, https://martinfowler.com/articles/cd4ml.html

Static vs. dynamic training

Definition

- **Static training**
  - Inference model trained once on a fixed data set
  - Put in production, does not change

- **Dynamic training**
  - Labeled data continuously coming into the system
  - New data incorporated into the model

Static vs. dynamic training
When to use

- **Static training**
  - Modeled reality is not expected to change (e.g., cat/dog pictures recognition, playing chess)
- **Dynamic training**
  - Modeled reality has trends and changes (e.g., buying trends, recommendations, self-driving cars, etc.)

Static vs. dynamic training

Attention points

- **Static training**
  - Model can degrade slowly.
  - More effort on data collection and labeling
- **Dynamic training**
  - Need monitoring (going off the rail?)
  - Need infrastructure for versioning and roll-back
  - Less effort on data collection, but tricky to get it right.

Reproducibility and auditability

- Keeping track of history is important
- To have reproducible results, need to know the exact version of
  1. Source code
     (e.g. model training and pre-processing)
  2. Training data
     (e.g. input signals and labels)
  3. Platform
     (e.g. OS, GPU, and version of installed packages)

Traceability and versioning

- Data sets should be tagged with metadata
  1. Origin
  2. Freshness (when collected)
  3. Code used to extract it

Traceability and versioning

- Models should be tagged with provenance
  - Which data used for training/testing?
  - Which pipeline generated it?

- Versioned copies should be kept of each <code,data,model,environment> tuple

- At least for models that make it to production

Versioning challenges

- Versioning very large artifacts (e.g., data) is difficult and costly.

Questions?

What is different with ML?

	Data	Model	Code
Traditional software project	A bit of this		A lot of this
ML software project	A lot of this	An this	A bit of this

Code is code, right?

Design anti-patterns in ML systems

- Pipeline jungles
  - no modularization of pipeline code, no refactoring

- *Mitigation*
  - Modularize and organize the code.
  - Testing.

Code is code, right?

Design anti-patterns in ML systems

- Excessive glue code
  - Massive amount of code written to get data into/out of a general-purpose package

- *Mitigation*
  - Create clean native solutions where feasible
  - Carefully consider the benefit and cost of adding (yet another) general-purpose package

Code is code, right?

Design anti-patterns in ML systems

- Dead experimental code paths
  – no clean up discipline

- *Mitigation*
  – Periodical inspection of experimental code.

Example

Blob of code. What does it do?

```
df = pd.DataFrame(...)
del df['column1’]
df = df.dropna(subset=['column2', 'column3’])
df = df.rename({'column2': 'unicorns', 'column3': 'dragons’})
df['new_column'] = ['iterable', 'of', 'items’]
df.reset_index(inplace=True, drop=True)
```

Example

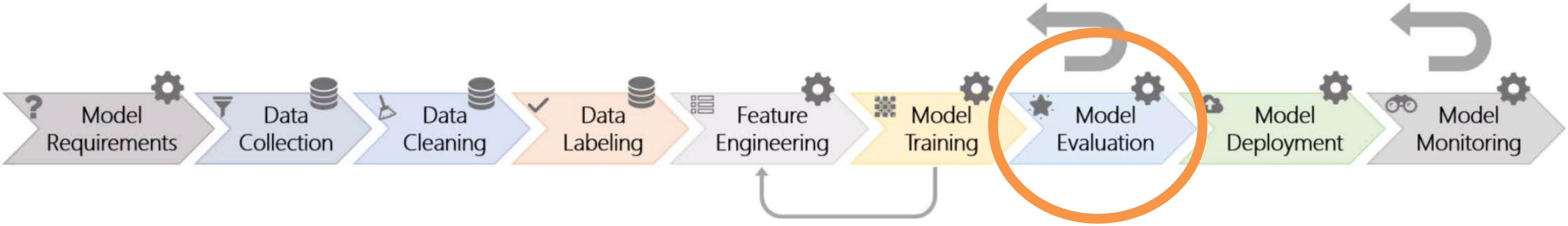
Blob of code. Explained!

```
# create a pandas DataFrame somehow
df = pd.DataFrame(...)
# delete a column from the dataframe
del df['column1']
# drop rows that have empty values in column 2 and 3
df = df.dropna(subset=['column2', 'column3'])
# rename column2 and column3
df = df.rename({'column2': 'unicorns', 'column3': 'dragons'})
# add a new column
df['new_column'] = ['iterable', 'of', 'items']
# reset index to account for the missing row we removed above
df.reset_index(inplace=True, drop=True)
```

Good comments explain **rationale**

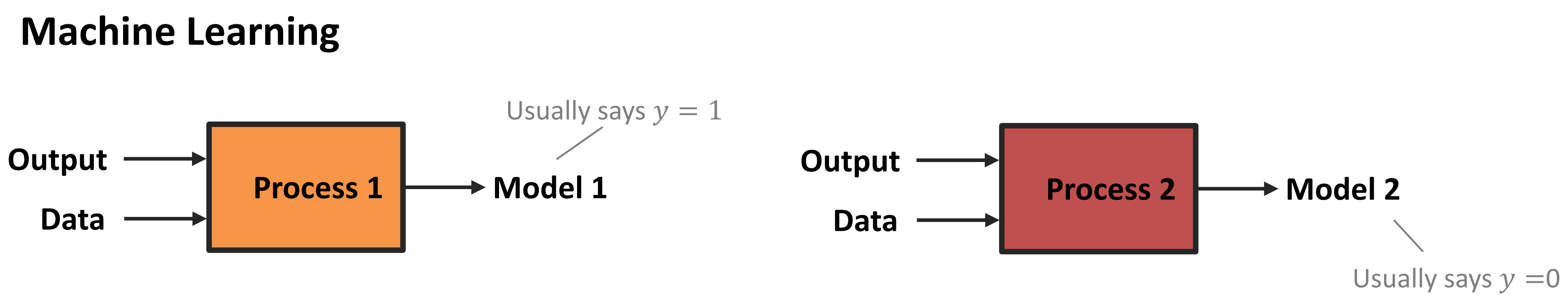
Questions?

The ML Workflow



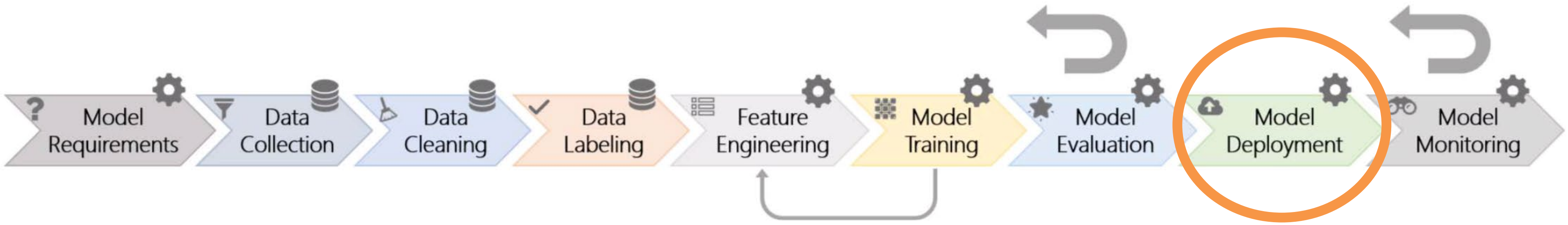
- Working with data:
  - Evaluation is creation

Evaluation



- Does Process 1 do a good job at mapping 'data' to 'output'?
- Is Model 2 better than Model 1?
- Are the mistakes similar or different? Which is better?

The ML Workflow



- Where intelligence lives
- Intelligent experience

- Related to **software architecture** –
  solutions are instances of architectural styles

Where Intelligence Lives

- Static intelligence in the product
- Client-side intelligence
- Server-side intelligence
- Back-end cached intelligence

Where Intelligence Lives

Static intelligence in the product

- Model is packaged with the application
- Pros
  - Cost of operation: Cheap
  - Latency in execution: Excellent
  - Offline operation: Yes
- Cons
  - Latency in updating intelligence: Poor
  - No data to improve the intelligence

Where Intelligence Lives

Client-side intelligence

- Executes completely on the client.
- Pros
  - Latency in execution: Excellent
  - Offline operation: Yes
- It *depends*
  - Latency in updating intelligence: Variable
  - Cost of operation: Based on update rate
- Cons
  - Exposes intelligence to the world

Where Intelligence Lives

Server-centric intelligence

- The intelligence runs in real-time in a service
  - Client sends features to a server; the server executes the intelligence on the features and returns the result.
- Pros
  - Latency in updating intelligence: Good
  - Easier monitoring
- It *depends*
  - Latency in execution: Variable
  - Cost of operation: Variable
- Cons
  - Offline operation: No

Where Intelligence Lives

Back-end cached intelligence

- Involves running the intelligence off-line, caching the results, and delivering these when needed.
- Pros and Cons
  – Latency in execution: Variable
  – Latency in updating intelligence: Variable
  – Cost of operation: Variable
  – Offline operation: Partially

Intelligence Experience

- A connection between the intelligence and users.

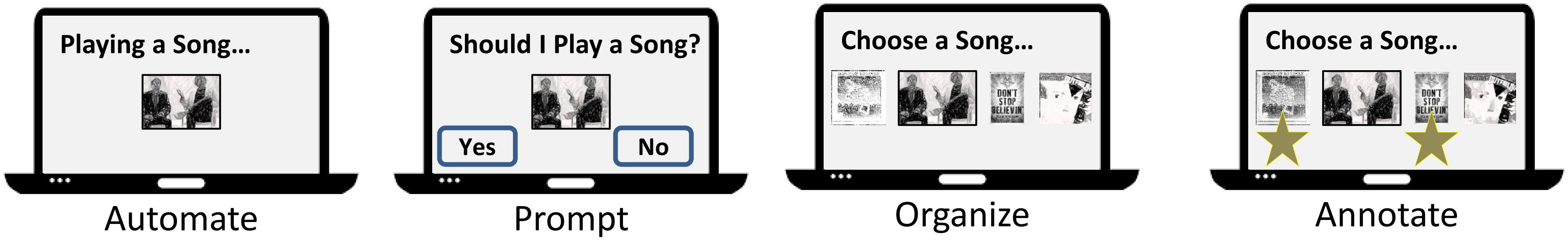
- *Example*: an intelligence determines that the user is in a room where it is a little dark for humans to see comfortably.
  - How should the experience respond?

Intelligence Experience

Techniques

- P (LikeSong |User, History)

- Modes of Intelligent Interaction



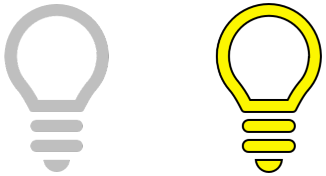
- Which intelligence experience to use? When to Update it?

Intelligence Experience

Goals

- Achieve objectives & increase profit and sales
- Engage users and achieve their objectives
- Get data to improve
- *Example*: Home Light → **P(on | sensors, motion, etc.)**
  - Objective?

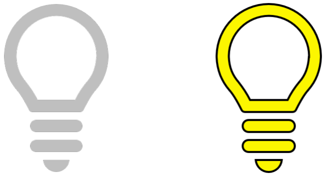
P <= 0.50 **P** > 0.50



Full automation

P > 0.90; turn on light

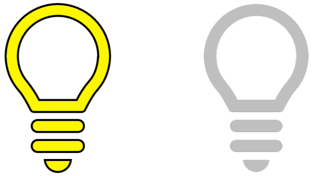
P > 0.75; then prompt user



Save Energy

P < 0.25; turn light off

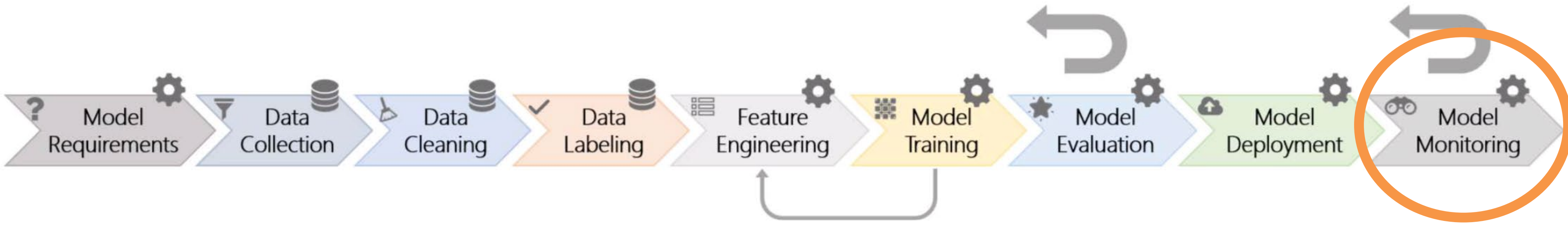
(turn light off after time t)



User Safety

Questions?

The ML Workflow



- Model Monitoring

Monitoring and Telemetry

- **Monitoring** is foundational for producing intelligence that:
  – functions correctly, and
  – improves over time.
- It includes knowing:
  – Context
  – Answers
  – Experiences
  – User behavior

- **Telemetry**: collect data at remote points and transmit them to a central point for monitoring

Monitoring inference output

- Model output
  - Predicted value is within bounds
  - Warning sign of model going off the rail

- Add sanity checks on output
- *Examples*
  - Predicted house not 10x bigger in same area
  - Values are within specified distributions

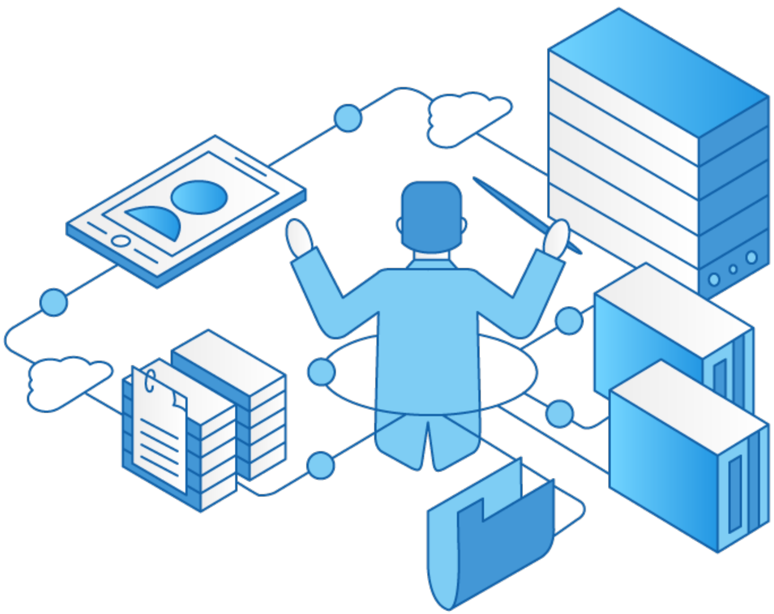
Monitoring: Rolling back

- Model “roll back” procedure: Being able to quickly revert to a previous known-good model

- Test how quickly and safely a model can be rolled back.

Intelligence Orchestration

- Achieve objectives of the ML system
- Controls the monitoring
- Balance experience
- Deals with mistakes



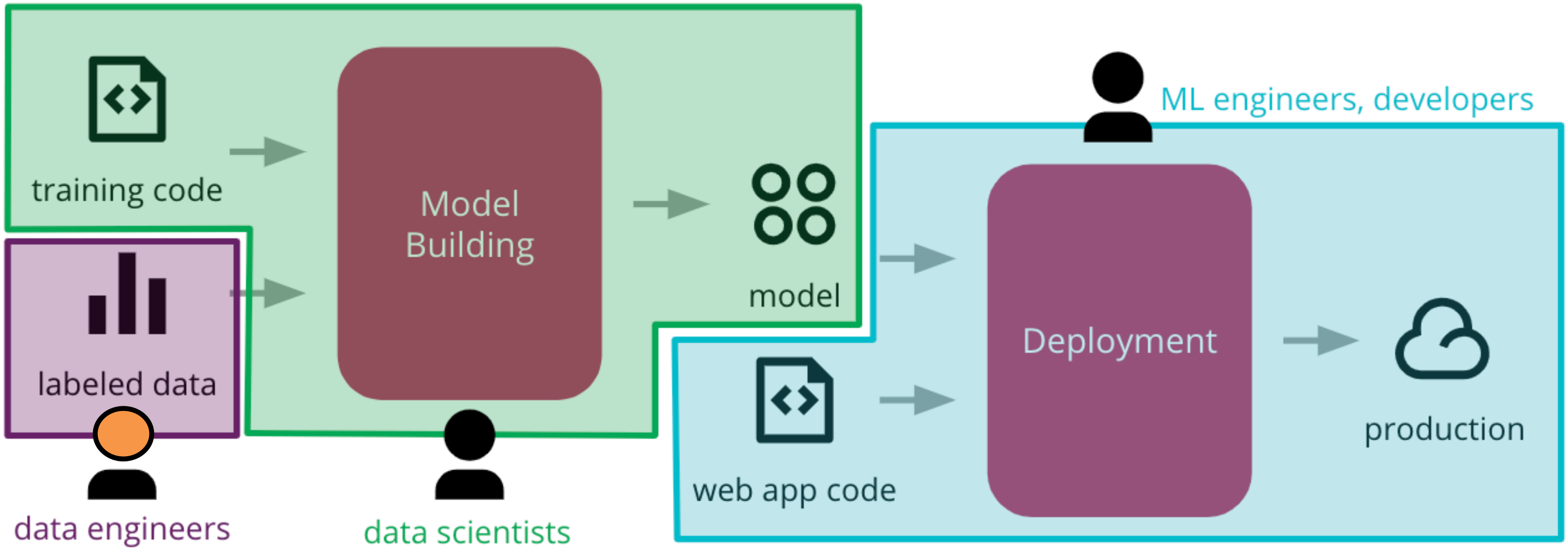
Orchestrating intelligent systems

Why needed?

- Objective changes:
  - Better understand a problem
  - Solved previous objective
- Intelligence changes:
  - More data, better models.
  - More accurate or less accurate model.
- User changes:
  - New users or users leave
- Cost changes:
  - Telemetry costs
  - Mistakes costs

Questions?

Summary: lifecycle of data-intensive AI systems

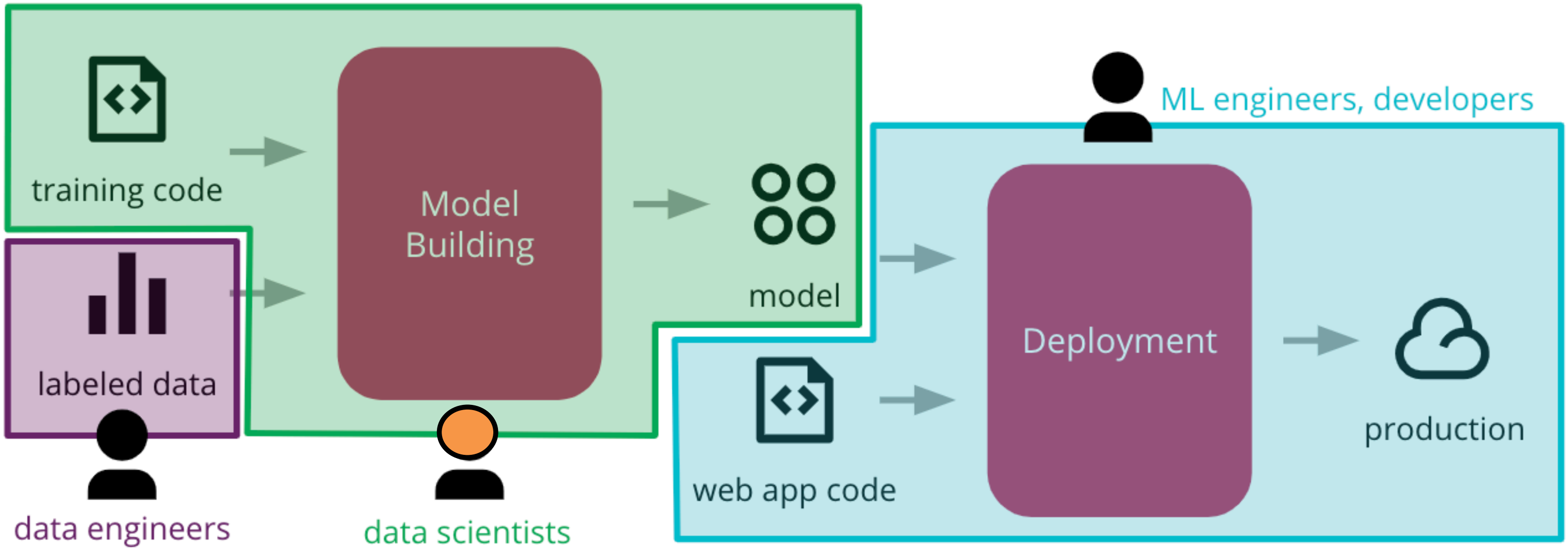


Prepare data

- Ingestion, Labeling, Normalization, Transformation, Validation

Source: D. Sato, A. Wider, C. Windheuser, Continuous Delivery for Machine Learning
https://martinfowler.com/articles/cd4ml.html

Lifecycle of data-intensive AI systems

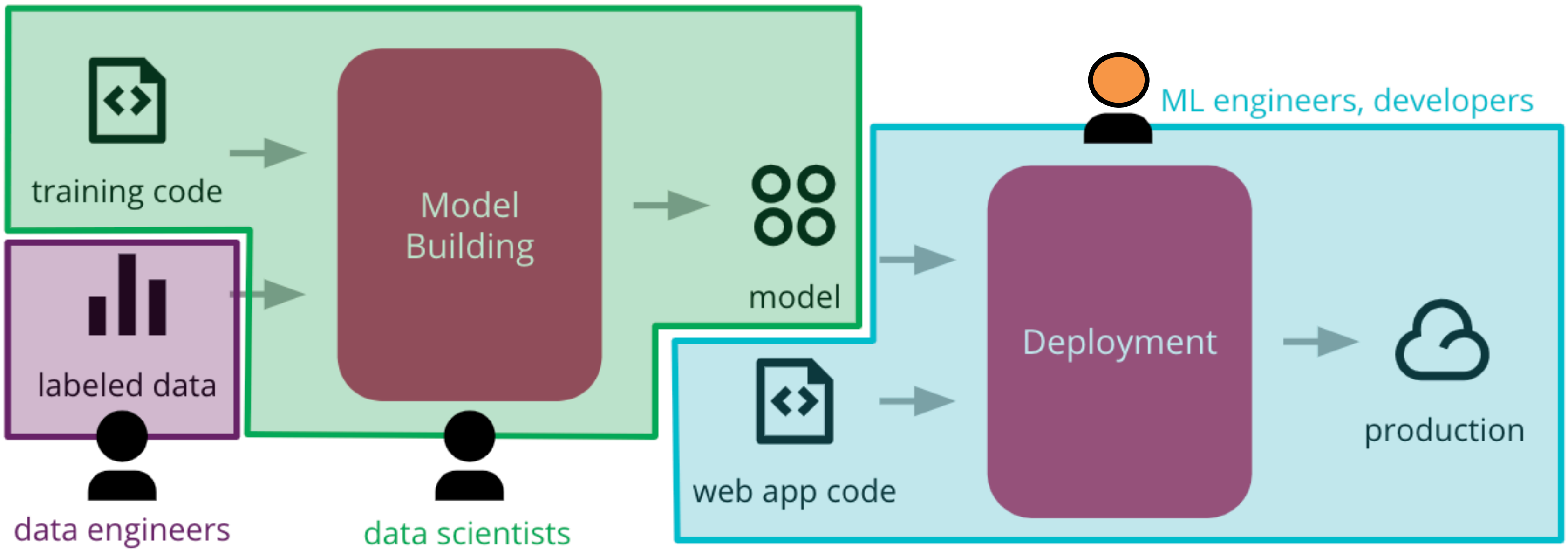


Experiment and build model

- Select features, Select ML algorithm, Tune parameters, perform evaluation and testing

Source: D. Sato, A. Wider, C. Windheuser, Continuous Delivery for Machine Learning

Lifecycle of data-intensive AI systems



Serving the model in production

- Deploy, Predict, Monitor, Update model

Source: D. Sato, A. Wider, C. Windheuser, Continuous Delivery for Machine Learning

Questions?