ML is like nuclear power

It’s one thing to do it in lab…

.. and another to run it in production!

(I have no idea how a nuclear lab looks like)

ML is like nuclear power

This is the part you are most likely familiar with

ML is like nuclear power

Now we focus on this!

ML in Practice with the City of Helsinki

Machine Learning D 2022 spring – Guest Lecture

Nuutti Sten
Data Scientist, the City of Helsinki

Helsinki

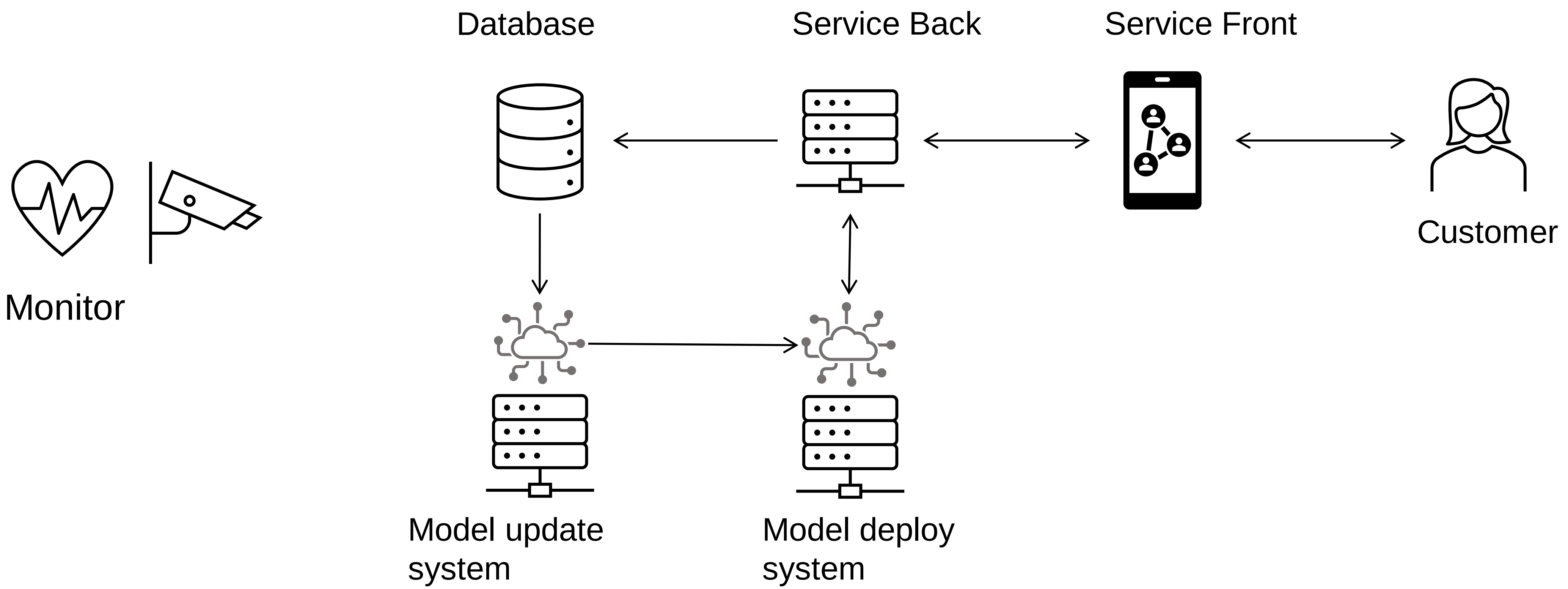
The City of Helsinki

- Why we do ML?

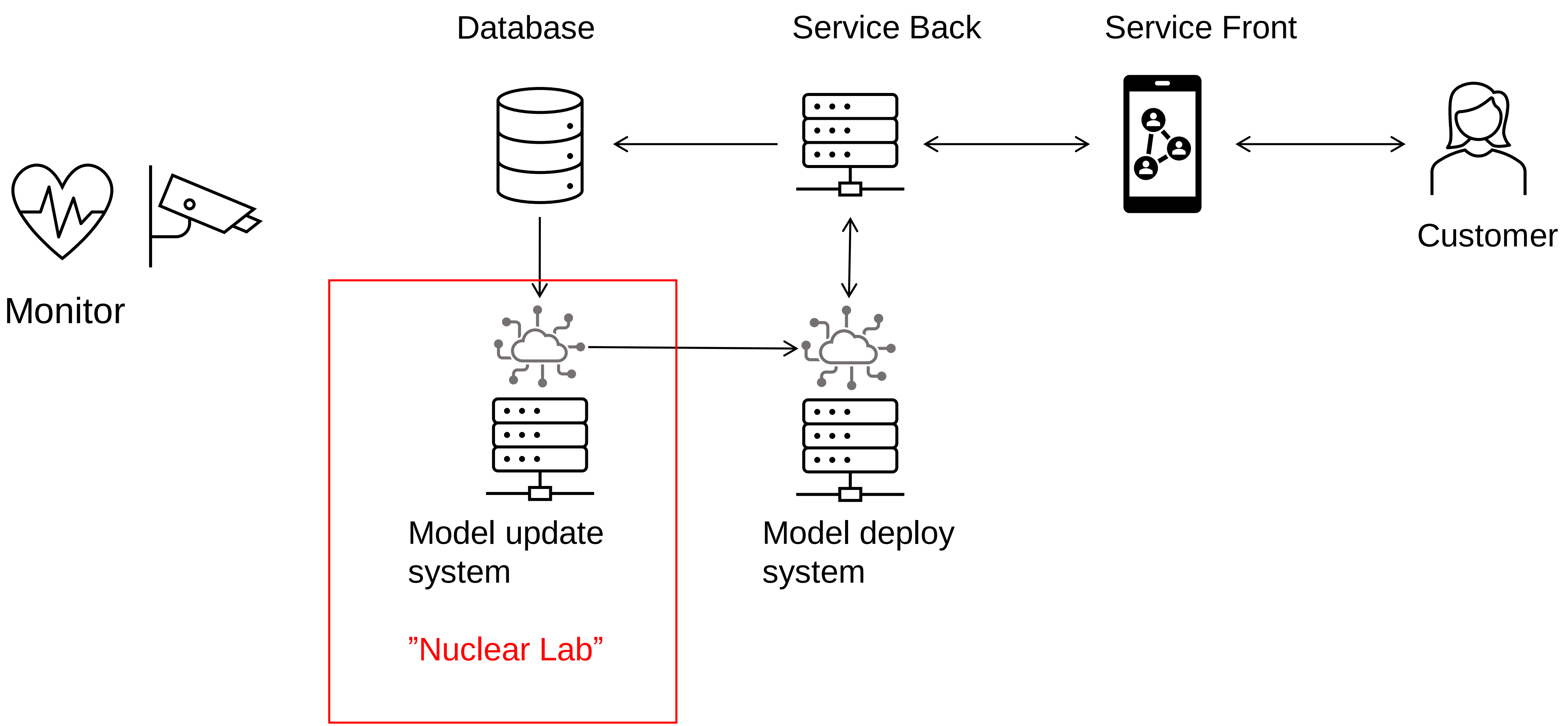
- Big issues:
  - Dependency ratio
  - Environmental crisis

- How ML is of help?
  - Proactive services
  - Descriptive analytics -> predictive analytics

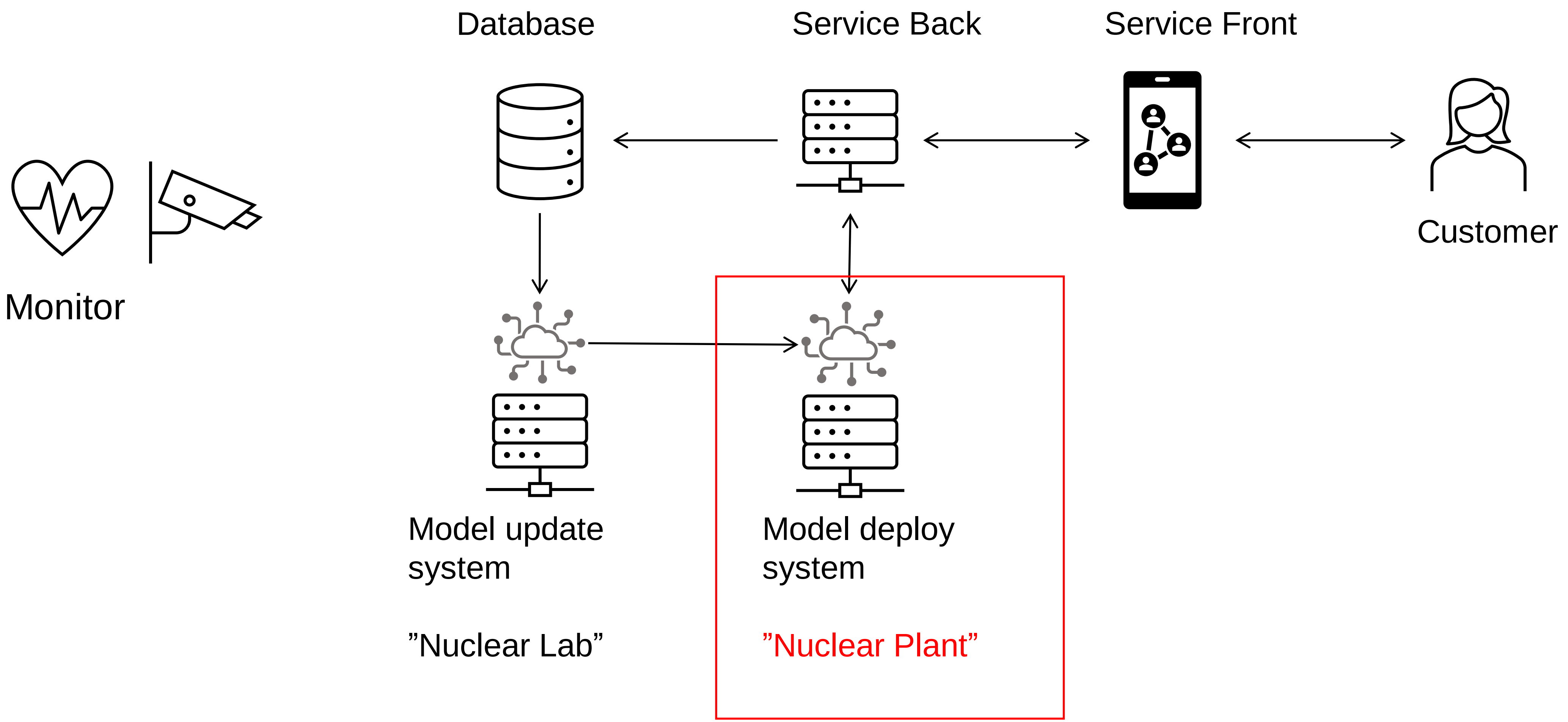
How ML looks like in practice



How ML looks like in practice



How ML looks like in practice



Note!

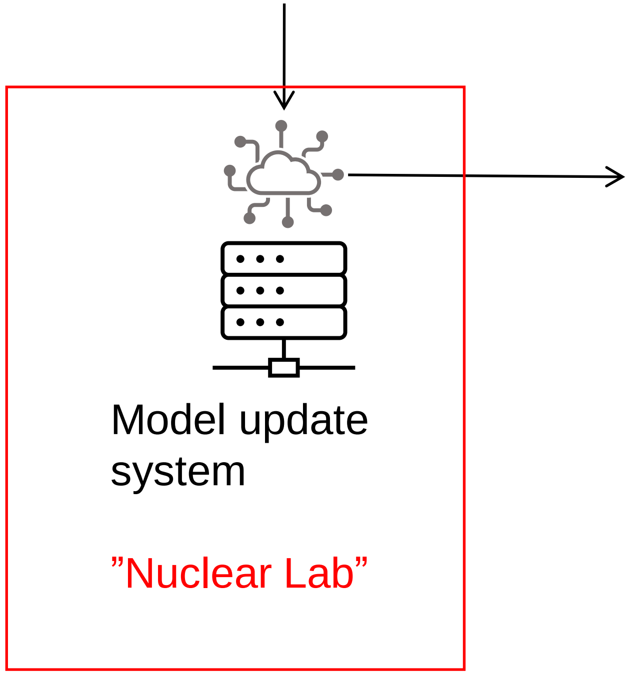
- Perspective: Python

- Everything applies to R & other languages, too

- Different tools, but same principles

Part 1: Creating applicable ML

- Tidy Principles
- Workflow – data, model, loss
- Exploratory programming
- Separate development and evaluation
- Everything is a function!
- Reproducibility
- API
- Monitor



Tidy Principles – Tidy Data

- Tidy Data
  - Each row is a data point
  - Each column is a feature or a label
  - Each cell contains a single value

- Tidy data is easy to process!
- Standard tools assume data is tidy!
  - Numpy, pandas, sklearn…

- Begin by converting your data into tidy format!
  - Exception: kernels!

	features	label
0	{'a': 1, 'b': 3}	2
1	{'a': 4, 'b': 8, 'age': 'young'}	5

	a	b	age	label
0	1.0	3.0	NaN	2
1	4.0	8.0	young	5

Source: Tidy data by Hadley Wickham

Tidy Principles – Tidy Tools

- Tidy Tools
  - Reuse existing data structures
  - Compose simple functions with the pipe
  - ~~Embrase functional programming~~ -> utilize vectorization
  - Design for humans

Code examples:

Source: The tidy tools manifesto by Hadley Wickham

```python
from sklearn.linear_model import LinearRegression as reg
class MessyModel:
    """
    Poorly constructed ML model class
    """


    def __init__(self, data):
        # model accepts data in custom (messy) format
        # and it has to be cleaned before use
        self.data = pd.concat(
            [data["features"].apply(pd.Series), data[["label"]]], axis=1
        )
        self.data = pd.concat(
            [self.data[["a", "b"]].apply(pd.Series), self.data[["label"]]], axis=1
        )
        # in addition model instance creates an unnecessary copy of the data
        self.score = np.nan

    def y_val(self, X):  # unconventional naming, non-verb, difficult to understand
        # a function first does a side-effect
        self.model = reg()
        self.model.fit(self.data.iloc[:, :-1], self.data.iloc[:, -1])
        # and then a transformation
        return self.model.predict(X)

    def score_calculations(self):
        # again, mixing transformations, side-effects and poor naming
        self.score = self.model.score(self.data.iloc[:, :-1], self.data.iloc[:, -1])
        return self

# and we can not pipe the functions!
messymodel = MessyModel(messy_df)
print(f"fit model and predict new values, I guess? {messymodel.y_val(test_x)}")
messymodel.score_calculations()
print(f"score, but whay what score and what the ? {messymodel.score}")
```

[138] ✓ 0.1s

```
fit model and predict new values, I guess? [4.11764706]
score, but whay what score and what the ? 1.0
```

```python
train_df = tidy_df.drop("age", axis=1)
X_train = train_df.iloc[:, :-1]
y_train = train_df.iloc[:, -1]


class TidyModel:
    """
    Fit, predict and evaluate a linear regression model
    """

    def __init__(self):
        self.model = reg()

    def fit(self, X, y):
        self.model.fit(X, y)
        # This function performs a side-effect only (model fit), so it returns self
        return self

    def predict(self, X) -> np.ndarray:
        # and here we do a transformation X -> y, so we return values!
        return self.model.predict(X)  # start noticing something?

    def score(self, X, y):
        return self.model.score(
            X, y
        )  # yup, Sklearn and other standard tools follow tidy principles!

# and so we can pipe simple functions!
tidymodel = TidyModel().fit(X_train, y_train)  # see pipe!
tidymodel.score(X_train, y_train)  # pipe!
TidyModel().fit(X_train, y_train).score(X_train, y_train)  # pipe again!

# can we go any further? Of course!
MAE = np.abs(TidyModel().fit(X_train, y_train).predict(X_train) - y_train).mean()
# wow we can do even longer pipes that are still readable and meaningful and very convenient!!
MAE
```

[175] ✓ 0.1s

... 0.0

Helsinki

Code examples:
github.com/NuuttiSten/tidy_examples

Exploratory programming

- Notebook development
  - Explainability: Code – results – documentation
  - Testing – each running notebook is a test
  - Workflow – each notebook is a workflow component

  - But how about copy-pasting?

- Tools: jupyter, nbdev, papermill

Separate development and evaluation

- Don’t be greedy!
- Dev with toy data, a tiny sample of your data!
- Make sure your algorithm works *as code* before you try it with whole data!
- This will save you so much time and tears!
- It also makes debugging way easier!

Reusable components

- Functions, classes >> scripts

- Parameterize <u>everything</u>
  - Scripts, functions, classes, tests, notebooks, everything together
  - Avoid hard-coding!

- Split code into independet, logical components

  - What are these for ML?

Reusable components

- Functions, classes >> scripts

- Parameterize everything
  - Scripts, functions, classes, tests, notebooks, everything together
  - Avoid hard-coding!

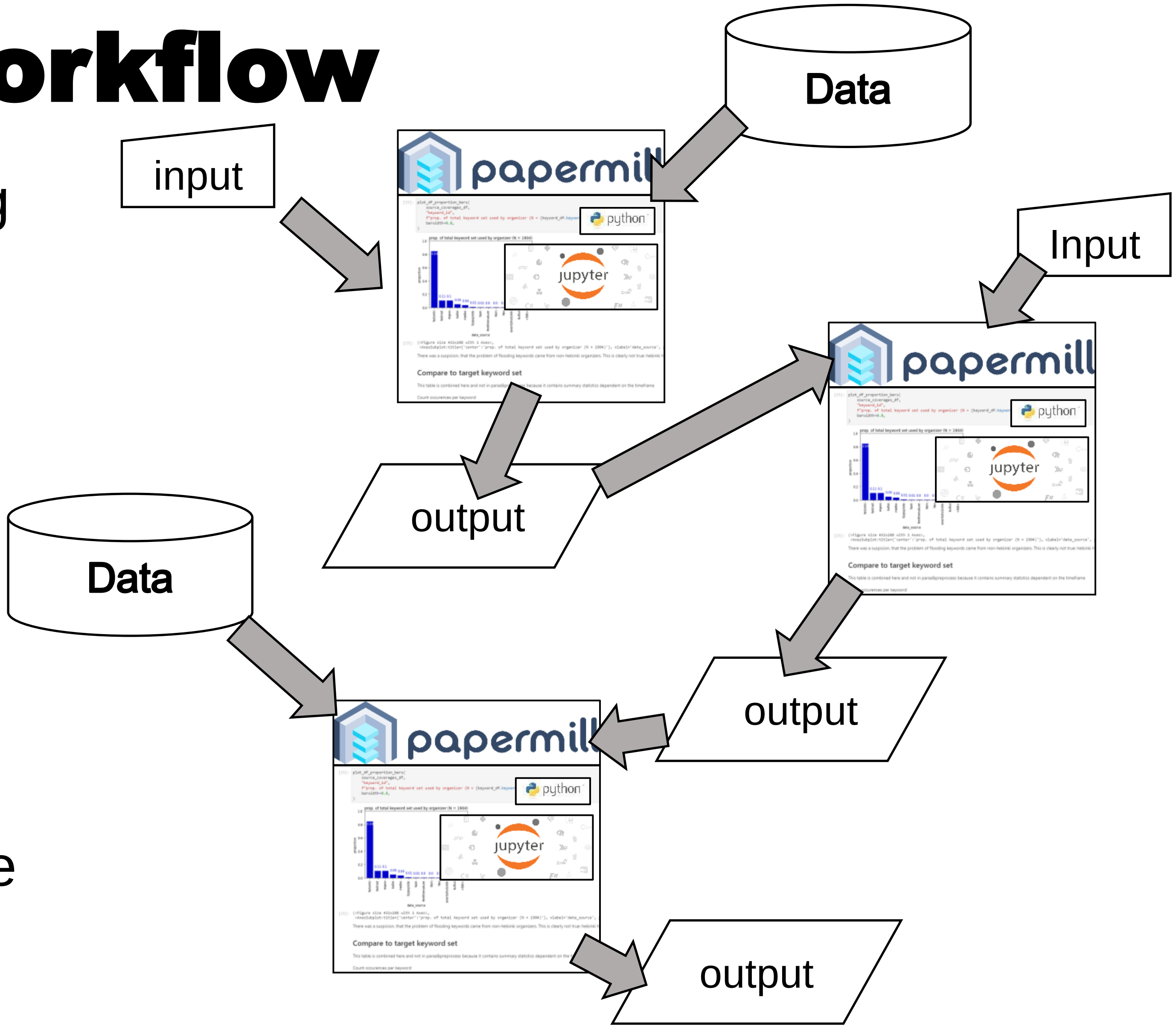
- Split code into independet, logical components

  - What are these for ML? -> data, model, loss

Reproducibily - workflow

- Automatically reproduce everything
  - ’update model’ -button
- Static
  - Light
  - Simple
- Dynamic
  - Only recreate the parts affected by change
  - Avoid unnecessary repetition

- Tools: Papermill, Elyra, Snakemake



Reproducibility - seed

- Seed ensure reproducibility of ’random’ processes

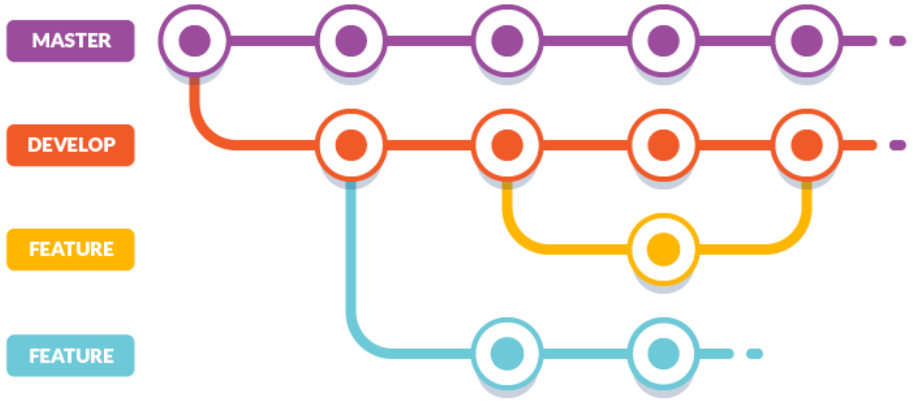
- Use changing seed in production
  - E.g. datetime-based hash

- Not always possible!
  - Weak environment control
  - Multiple runtimes

  - Partial reproducibility!

Reproducibility – version control

- Version everything
  - Code
  - Data
  - Environment
  - Parameters
  - Seeds
- Tools: Git, DVC, flat-data

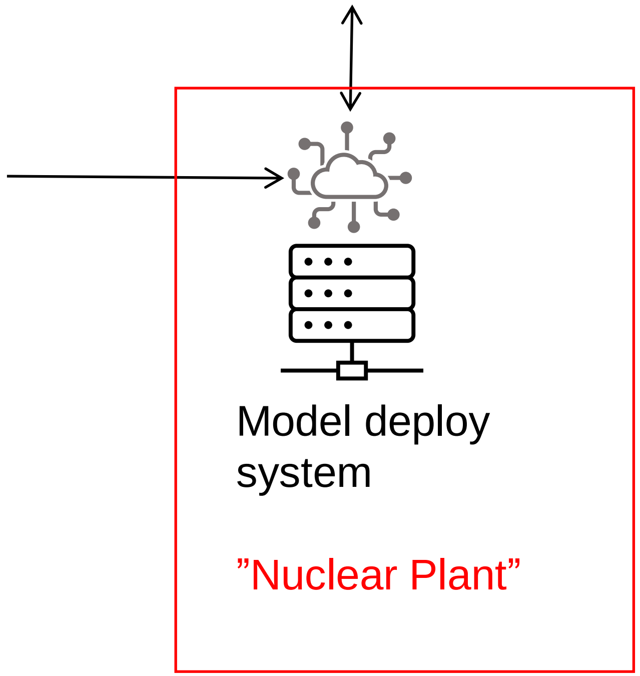


Reproducibility – environment control

- Dependencies
  - Tools: pip, pip-tools, conda etc.
- Environment control
  - Virtual environments (venv, pyenv, conda etc.)
  - Containers (**Docker**, Singularity, gVisor, etc.)
- Mitigate differences between development and deployment environments!

Part 2: Applying ML (ML Ops)

- Continuous delivery of value
- Acceptance tests!
- Deploying ML models
- Monitor
- People
- Best practice & Antipatterns



Change

- Where does change come from?
- Can you control it?
  - Controllable change (business needs)
  - Uncontrollable change (data)

- DevOps – continuous value cretion with SW products

- MLOps – continuous value creation with data products
  - MLOps = ML DevOps = DataOps

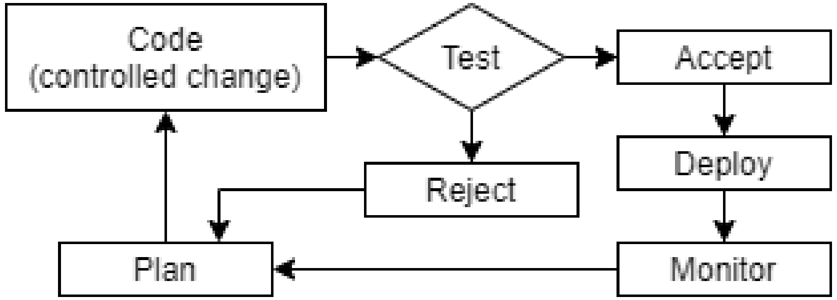


Figure 1: DevOps for software products

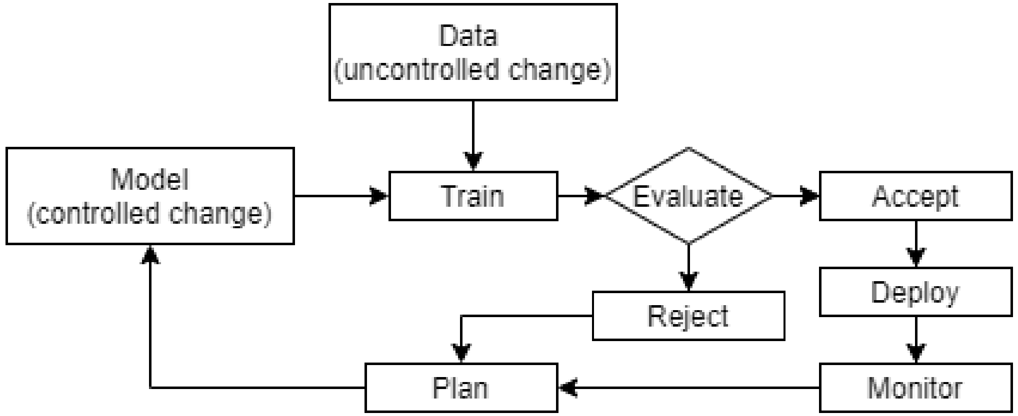


Figure 2: ML Ops for data based products

Acceptance tests

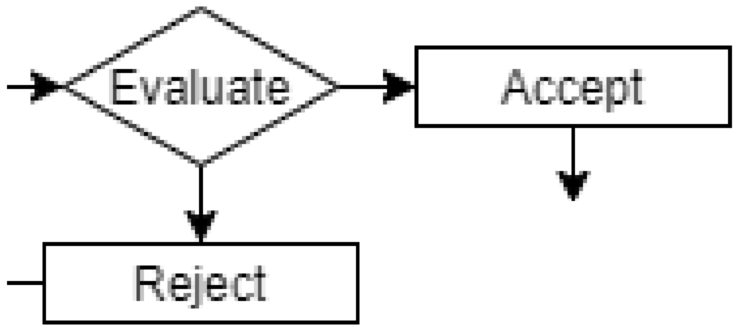
- Only deploy ’good enough’ models!

- Adapt to change, try not control it!

- Train – test – val
  - Validate with data that you or the model have never seen before

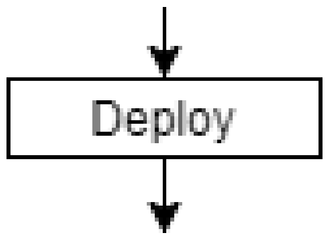
  - Preferably with fresh data, collected after you began development

  - Do not go phishing for good results!



Deploying ML models

- API
  - Tools: Django, Flask, FastAPI



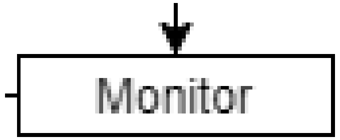
- Computing environment & Scaling up & down

- Tools: commercial products, container orchestration tools, autodeploy tools, cloud platform products

  - However, a DIY open source setup is possible for lightweight demo applications!

Monitor

- Model performance
  - Performance will decay over time
  - Soon the model would not pass the acceptance test
  - Define a critical level, at which the model must be updated!



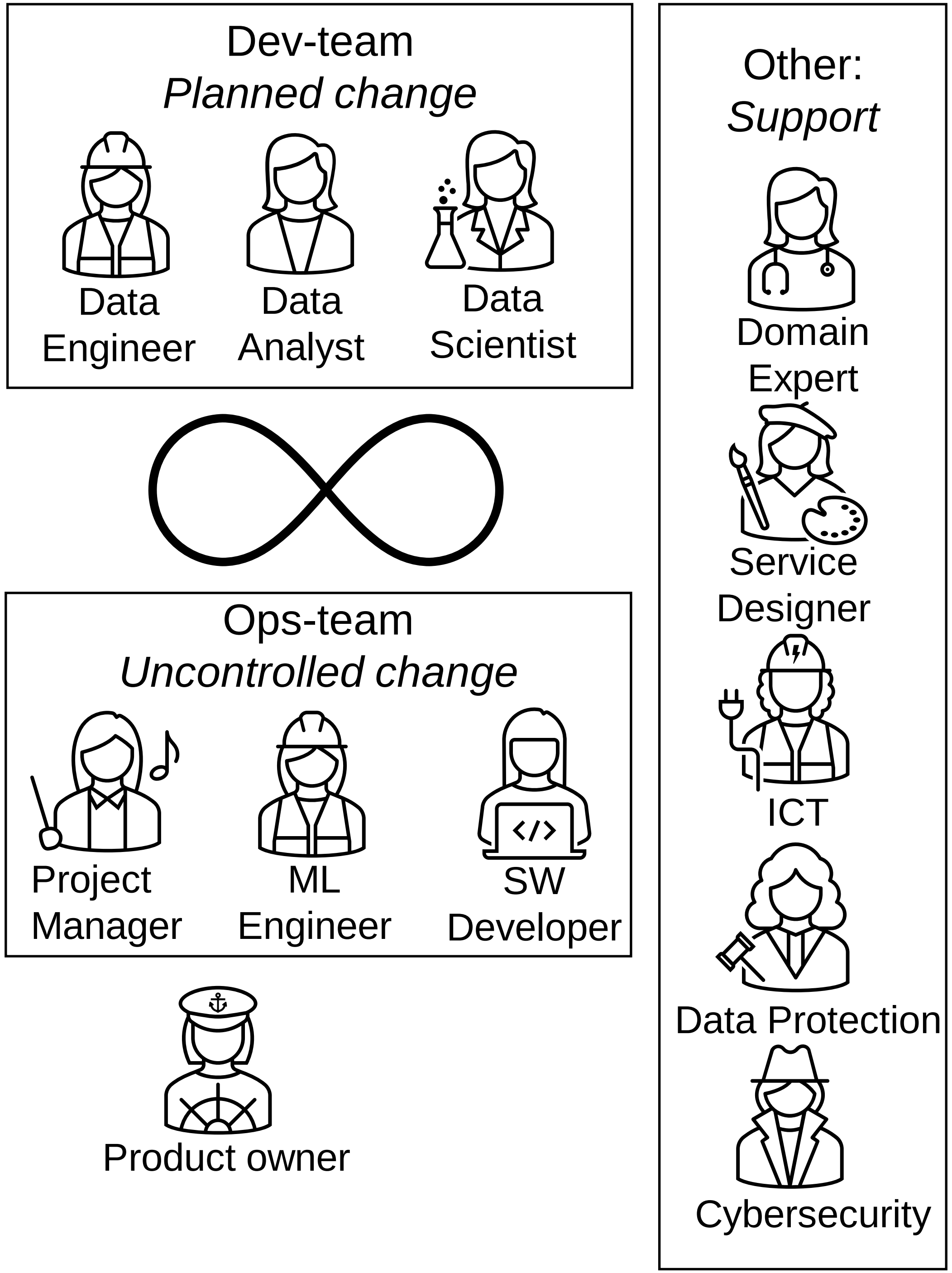
- Business goal metrics
  - Math is irrelevant! Value is what matters!
  - Collect as many metrics as you can!

- Trigger model updates, if changes in:
  - Data, model, model performance, business metrics

- Tools: commercial products and cloud platform tools

People

- It’s a team effort – no superheros!
- It’s important that all responsibility of a ML tool does not lie on one person
- However, how this is solved depends a lot from the organization
  - Roles must be defined!
  - This is worth asking in an interview!



Best practice & Antipatterns

- ’Industry common knowledge’
  - Rarely scientificly proven
  - Beware for hidden agenda

- Best practices: ’we did this and it’s going well’

- Antipatterns: ’we did this and it did not work out at all’

Tips to the end

- The city of Helsinki is all about open source & open data
    - Yes this is the part with the cow stuck in the mud!

Moo!
Use our
things!

Helsinki ML project template

- GitHub repository template
- Easily build reproducible and appicable machine learning tools
- Suitable for any open-source ML project (not sure about the requirements for this course, though)
    - Thesis, Research, Work
    - Build a demo to showcase your ML skills for summer job hunt
    - Found a ML startup, mars is the limit?

    - Happy to see the results if you end up using it!

- github.com/City-of-Helsinki/ml_project_template

Helsinki Tabular Anonymizer

- Open source tool for K-anonymization & pseudonymization
- Implements the Mondrian algorithm

- github.com/Datahel/tabular-anonymizer

Helsinki Region Infoshare (HRI)

- Open data from capital area
  - Maps, economy, healthcare, education, environment – it’s all there

- Project ideas for this course from HRI data:
  - GAN to create fake documents in Finnish administrative language
  - Improve a navigator to avoid routes with high risk of traffic accidents
  - Predict areas with suitable microclimate to relocate endangered species
  - Again, mars is the limit?

  - We are happy to see your results!

- hri.fi

Thank You!

Nuutti Sten

- Data Scientist at the city of Helsinki
- M.Sc. (tech) 2020, Aalto

- Why contact me?
  - Collaborate on applied ML research
  - Get Helsinki data for research purposes
  - Always interested in cool data stuff ☺

- How to contact me?
  - LinkedIn: linkedin.com/in/nuuttisten/
  - Email: nuutti.sten (at) hel.fi
  - github.com/NuuttiSten

Helsinki

Examples of Helsinki ML projects

- Simulating Customer Journey Prediction in a Federated Learning Setup
  - Link: github.com/City-of-Helsinki/ml_federated_customer_journey

- Cluster-based analysis of employment service customers
- Keyword tagging events based on description
- Predicting Metro train maintenance alerts
- Optimizing daycare allocation based on commute route
- Library item classification
- Cluster-based analysis of segregation of zip code areas

ML practices in the industry

- **A good read:**
- **Practices and Infrastructures for ML Systems – An Interview Study**, Dennis Muiruri et al. 2021 (Jukka Nurminen group, Uni. Helsinki)

Want to become a Data Scientist?

- Learn some of these (you don’t need to master all) and showcase them in your CV:
    - Python, R, SQL
    - Git
    - Statistical Hypothesis Testing
    - Software Testing
    - Data Visualization
    - Machine Learning
    - Cloud Concepts
    - DevOps
    - Teamwork: Communications & Task Management
    - Public speaking or teaching
    - Strong Ethics
    - Understand Value
    - A Domain of Expertise (e.g. economics, healthcare, geology etc.)
    - Basic knowledge of Business / Industrial Engineering