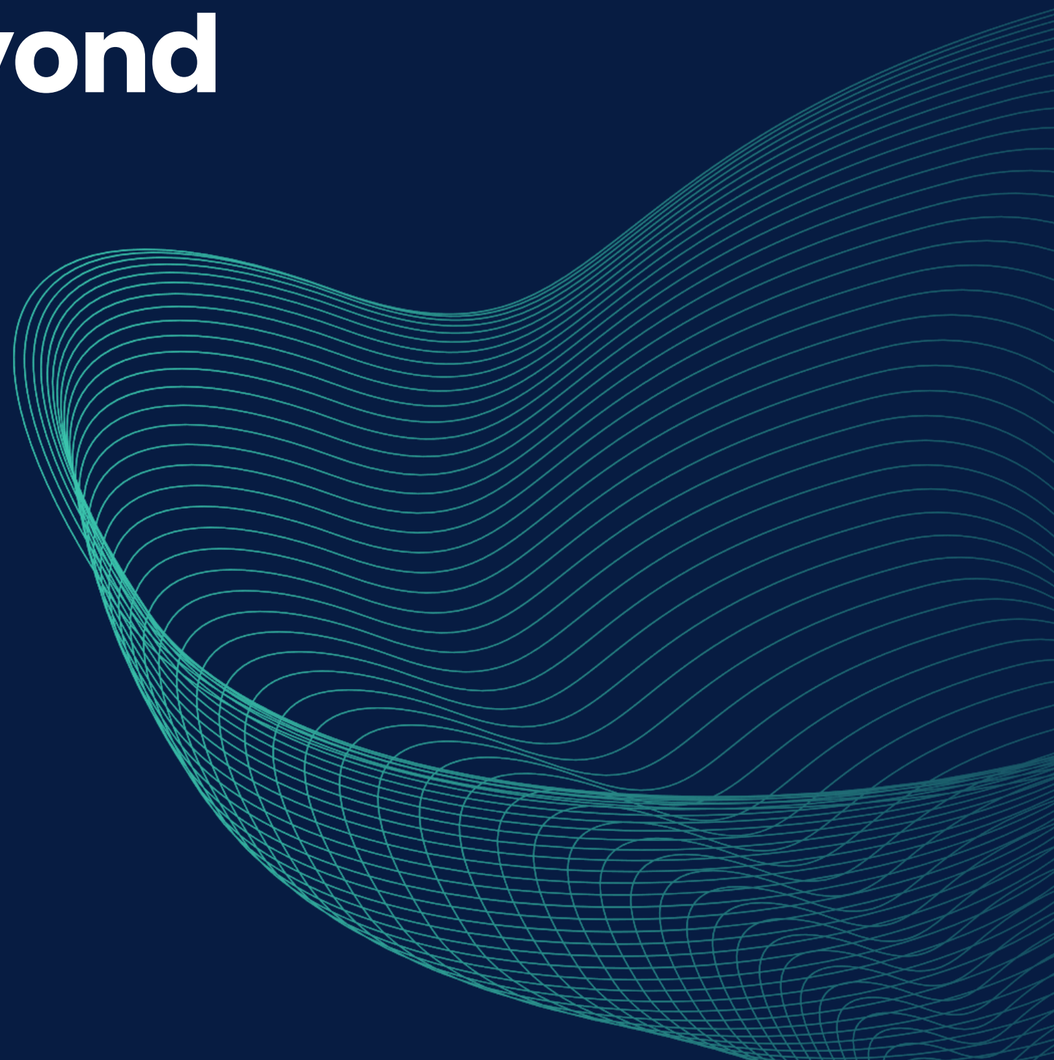
Intro to DevOps and Beyond

Ravindu Nirmal Fernando



Ravindu Nirmal Fernando

https://ravindunfernando.com

About Me

- STL - DevOps @ Sysco LABS - Sri Lanka
- MSc in Computer Science specialized in Cloud Computing (UOM)
- AWS Certified Solutions Architect - Professional
- Certified Kubernetes Administrator (CKA)
- AWS Community Builder

The Era before DevOps

Developers

Focused on Agility

Operators

Focused on Stability

"Destructive downward spiral in IT" - Gene Kim

Act 01 - Operations teams maintaining large fragile applications

Doesn't have any visibility on the application, whether or not its working as expected

Act 02 - The product managers

Larger, unrealistic commitments made to the outside world (client/ investors) without understanding the complexities behind development and operations

Act 03 - The Developers

Developers taking shortcuts and putting more and more fragile code on top of existing ones

Act 04 - Dev and Ops at war

"It worked on my machine" phenomenon

How can we overcome these issues?

“DevOps is the combination of cultural philosophies, practices, and tools that increases an organization’s ability to deliver applications and services at high velocity”
- What is DevOps? [AWS] -

“A compound of development (Dev) and operations (Ops), DevOps is the union of people, process, and technology to continually provide value to customers.”
- What is DevOps? [Azure] -

DevOps allows evolving and improving products at a faster pace than businesses using traditional software development and infrastructure management processes. This speed allows businesses to serve their customers better and compete effectively.

Key Areas in DevOps

Reduce Organizational Silos

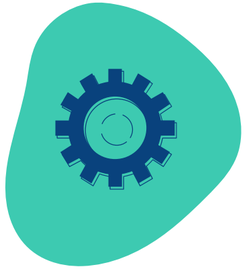
Everyones shares the ownership of production and information is shared among everyone

Accept Failure as Normal

Blameless PMs/ RCA. Risk taking mindset.

Implement Gradual Changes

Frequent deployments, frequent deterministic releases in small chunks which can be rolled back



Leverage Tooling and Automation

Automate and reduce manual work as much as possible

Measure Everything

Application, systems monitoring and metrics etc...

DevOps Practices

- Continuous Integration (CI) - Software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run.

- Continuous Delivery (CD) - Software development practice where code changes are automatically built, tested, and prepared for a release to production (automated code change deployment to staging/ pre-production system).

- Continuous Deployment (CD) - Every change that passes all stages of the pipeline will be deployed into production (released to customers). This practice fully automates the whole release flow without human intervention and only a failed test will prevent a new change being deployed.

- Microservices - The microservices architecture is a design approach to build a single application as a set of small services with each focusing on SRP. Each service can be created, deployed and run independently.

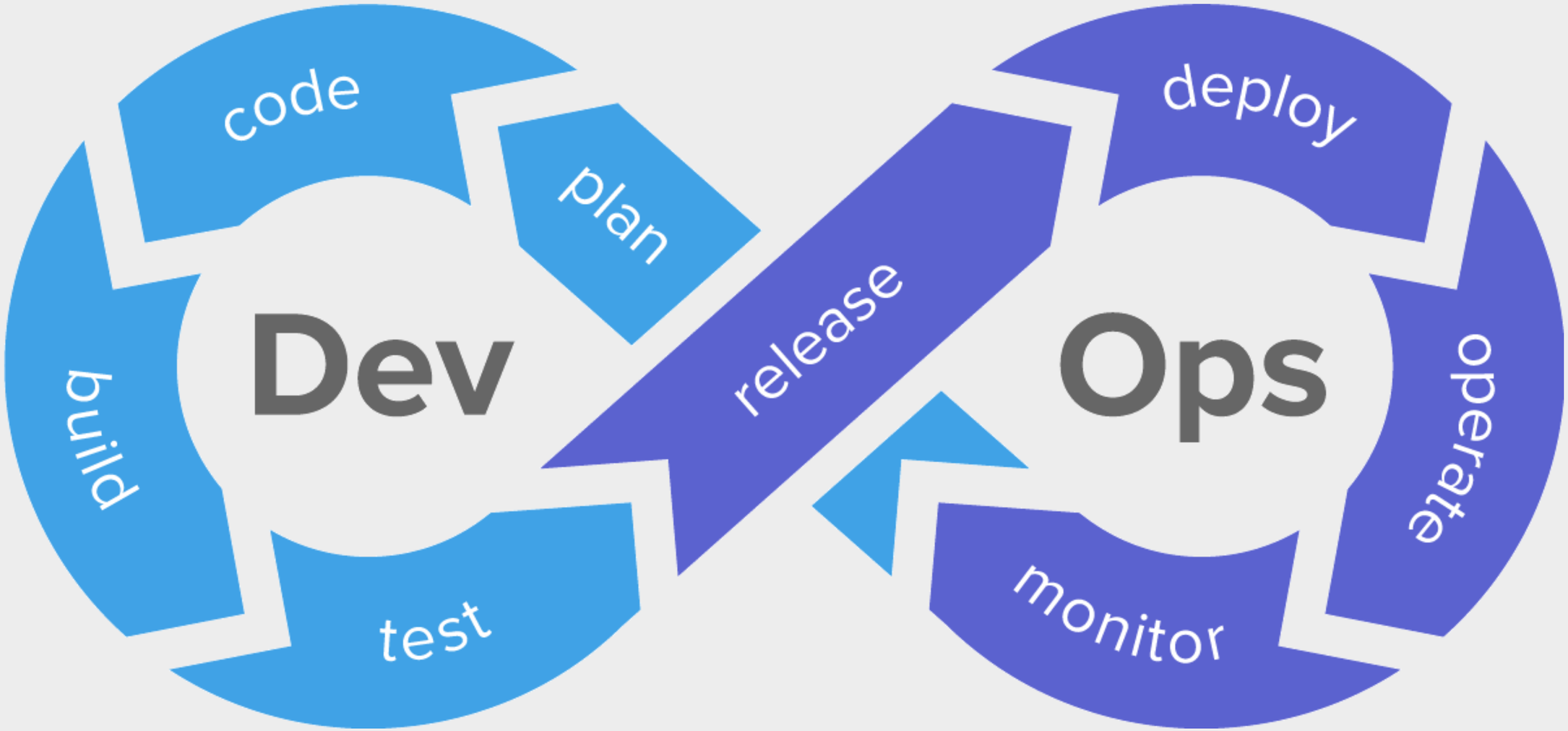
- Infrastructure as Code - A practice in which infrastructure is provisioned and managed using code and software development techniques, such as version control and continuous integration.
  - Configuration Management
  - Policy as Code

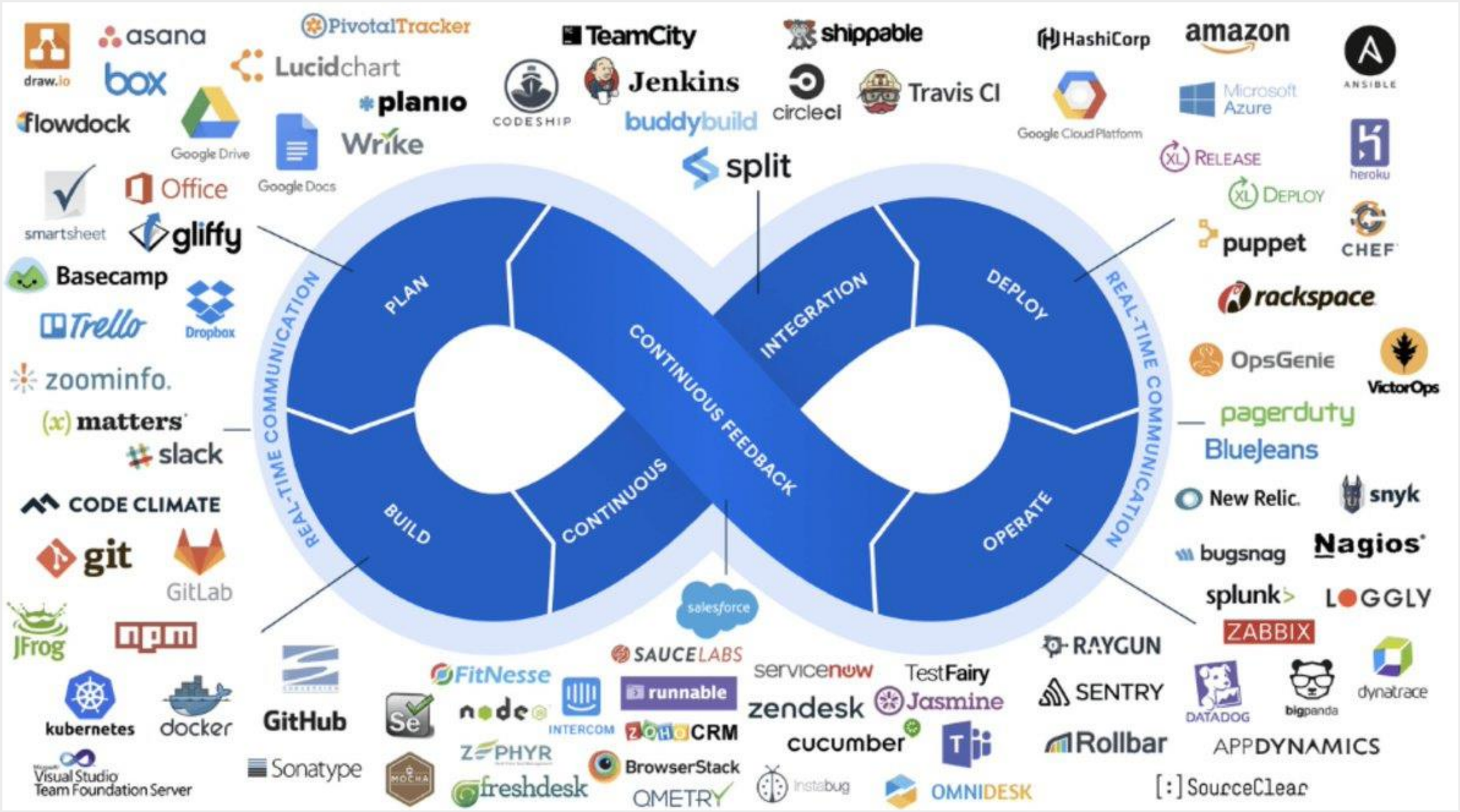
- GitOps - builds on the concept of IaC, incorporating the functionality of Git repositories, merge requests (MRs) and CI/CD to further unify software development and infrastructure operations. GitOps incorporates managing both infrastructure and applications as code.

- Cloud Infrastructure - Cloud provides more flexibility, scalability and toolsets for organizations to implement DevOps culture and practices. Serverless architecture in cloud brings down the efforts of DevOps teams as it eliminates server management operations.

- Continuous Monitoring, Logging and Alerting - Organizations monitor metrics and logs to see how application and infrastructure performance impacts the experience of their product's end user. Combined with real time alerts organizations can do a real time analysis on the application status.

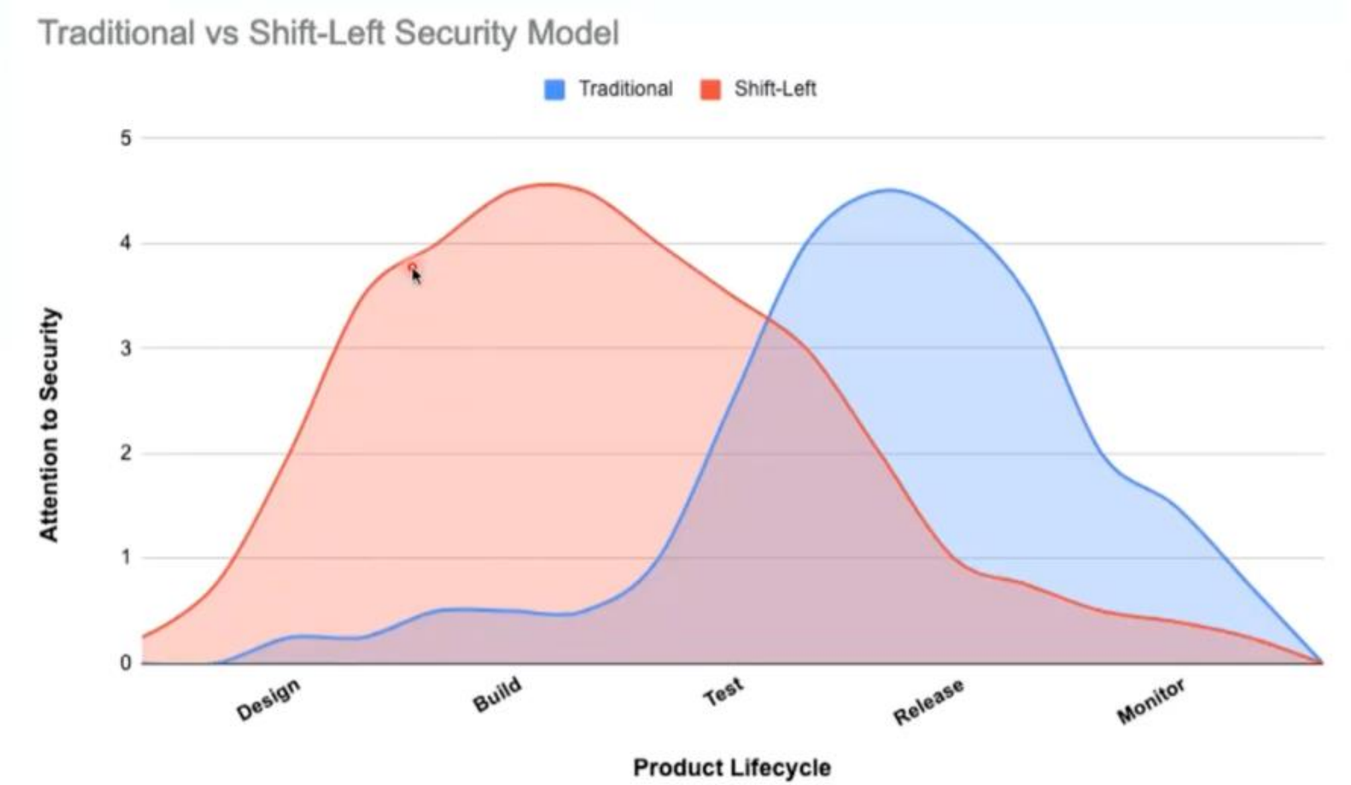
DevOps Tools and Technologies


code
plan
build
test
Dev
release
deploy
operate
monitor
Ops


draw.io
asana
box
flowdock
Google Drive
PivotalTracker
Lucidchart
planio
Wrike
Google Docs
CODESHIP
TeamCity
Jenkins
buddybuild
shippable
circleci
Travis CI
split
HashiCorp
Google Cloud Platform
amazon
Microsoft Azure
ANSIBLE
RELEASE
heroku
DEPLOY
puppet
CHEF
rackspace
smartsheet
Office
gliffy
Basecamp
Trello
Dropbox
zoominfo.
(x) matters
slack
CODE CLIMATE
git
GitLab
JFrog
npm
kubernetes
docker
Visual Studio Team Foundation Server
REAL-TIME COMMUNICATION
PLAN
BUILD
CONTINUOUS
CONTINUOUS FEEDBACK
INTEGRATION
DEPLOY
OPERATE
REAL-TIME COMMUNICATION
OpsGenie
VictorOps
pagerduty
BlueJeans
New Relic.
snyk
bugsnag
Nagios
splunk>
LOGGLY
ZABBIX
RAYGUN
SENTRY
Rollbar
DATADOG
bigpanda
dynatrace
APPDYNAMICS
[:] SourceClear
GitHub
Sonatype
Selenium
MOCHA
FitNesse
node
INTERCOM
ZEPHYR
freshdesk
SAUCELABS
runnable
CRM
BrowserStack
QMETRY
salesforce
servicenow
TestFairy
zendesk
Jasmine
cucumber
Instabug
OMNIDESK

Beyond DevOps

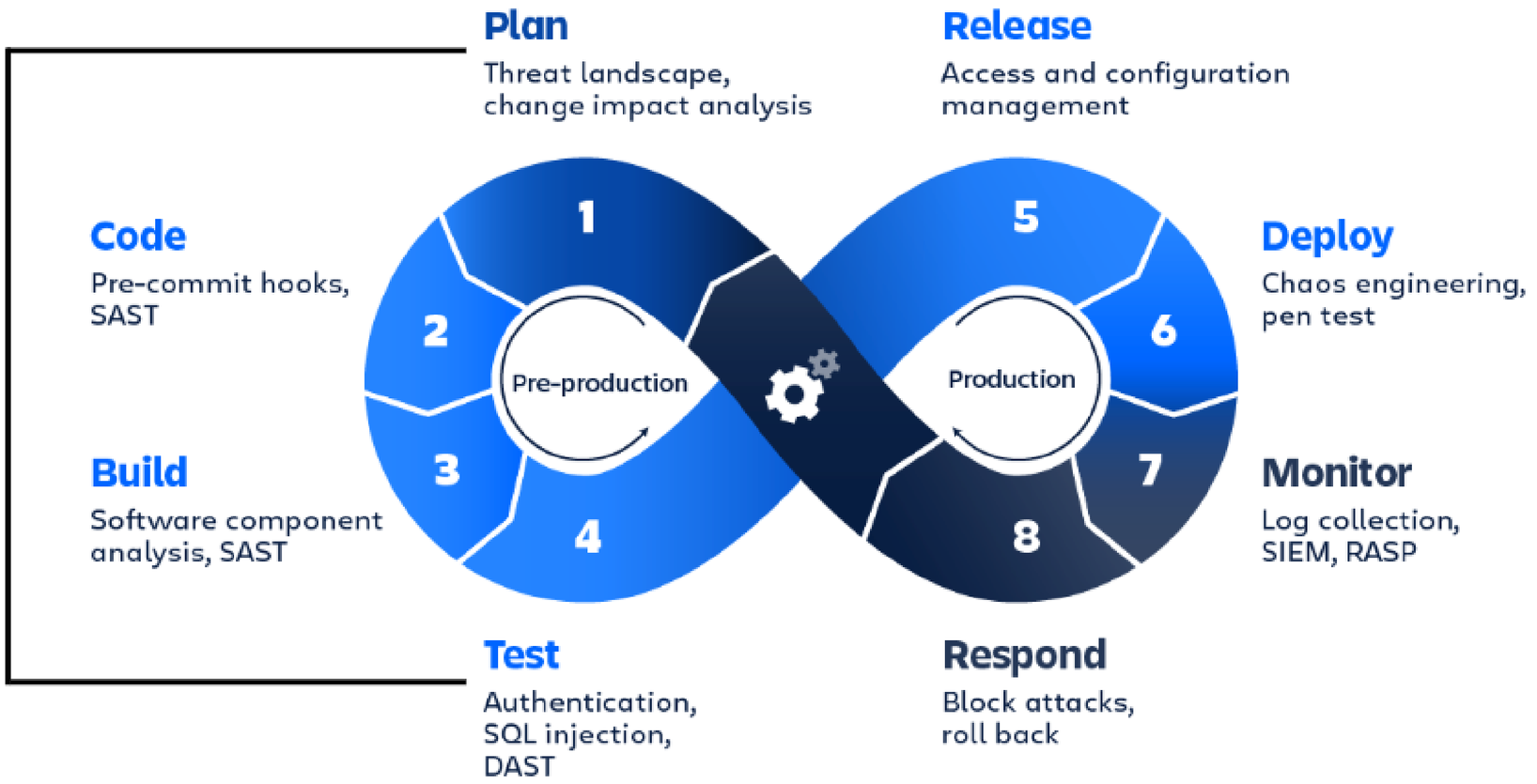
DevSecOps



Idea of moving Security in the early stages of the SDLC pipeline

"the practice of integrating security into a continuous integration, continuous delivery, and continuous deployment pipeline"

DevSecOps


Plan
Threat landscape, change impact analysis
Release
Access and configuration management
Code
Pre-commit hooks, SAST
Deploy
Chaos engineering, pen test
Build
Software component analysis, SAST
Monitor
Log collection, SIEM, RASP
Test
Authentication, SQL injection, DAST
Respond
Block attacks, roll back
1
2
3
4
5
6
7
8
Pre-production
Production

SRE (Site Reliability Engineering)

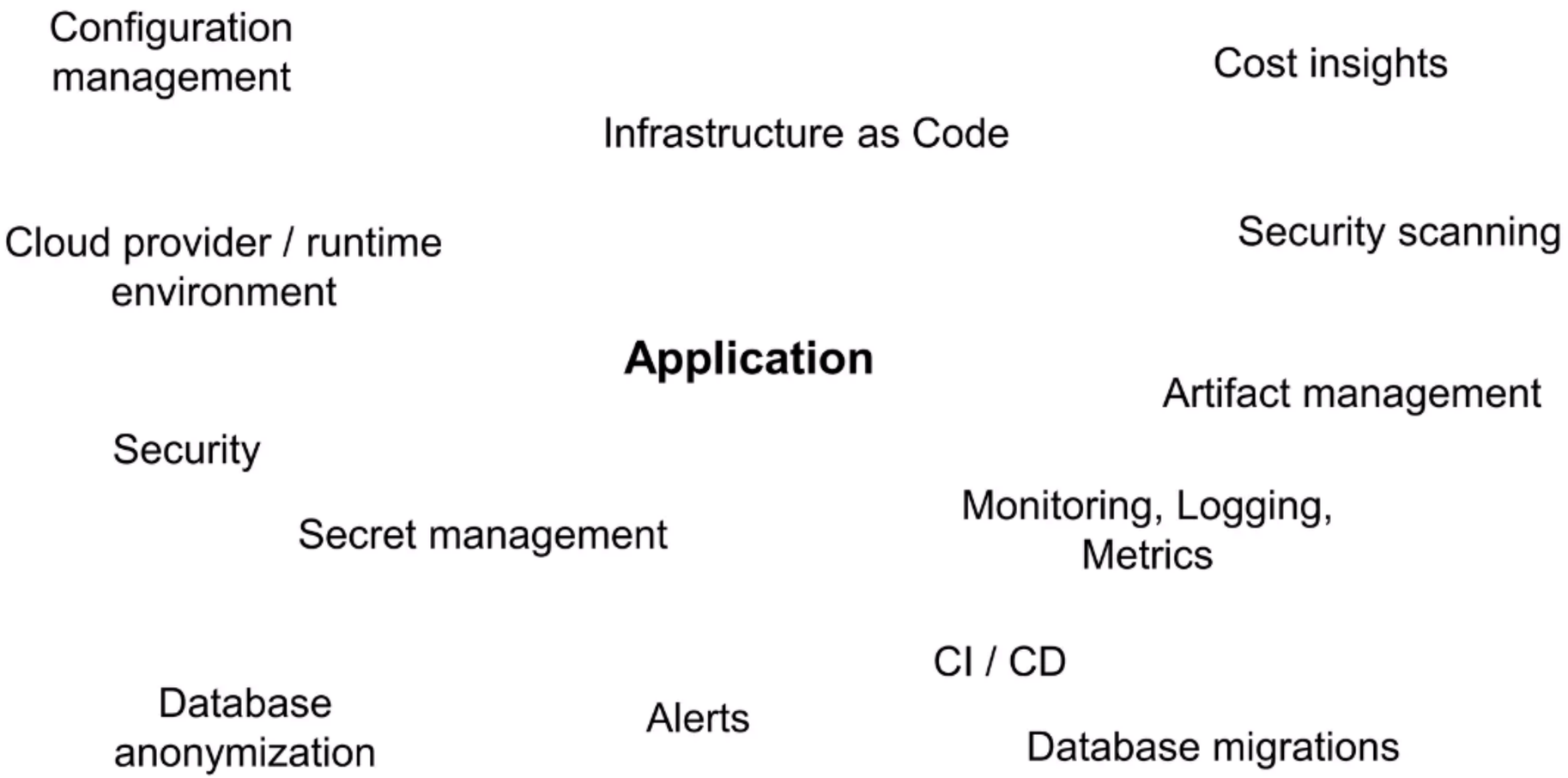
- Not competing with DevOps
- Think that Class SRE implements Interface DevOps
- SRE is a part of the DevOps umbrella

SRE Practices

- Identify and measure **SLI**s, define **SLO**s and agree/ commit to **SLA** for product and service
- Chaos Engineering
- Removing toil
- System designing (DR, Multi-Region, Mult-Cloud)
- Postmortems/ Root Cause Analysis
- Observability

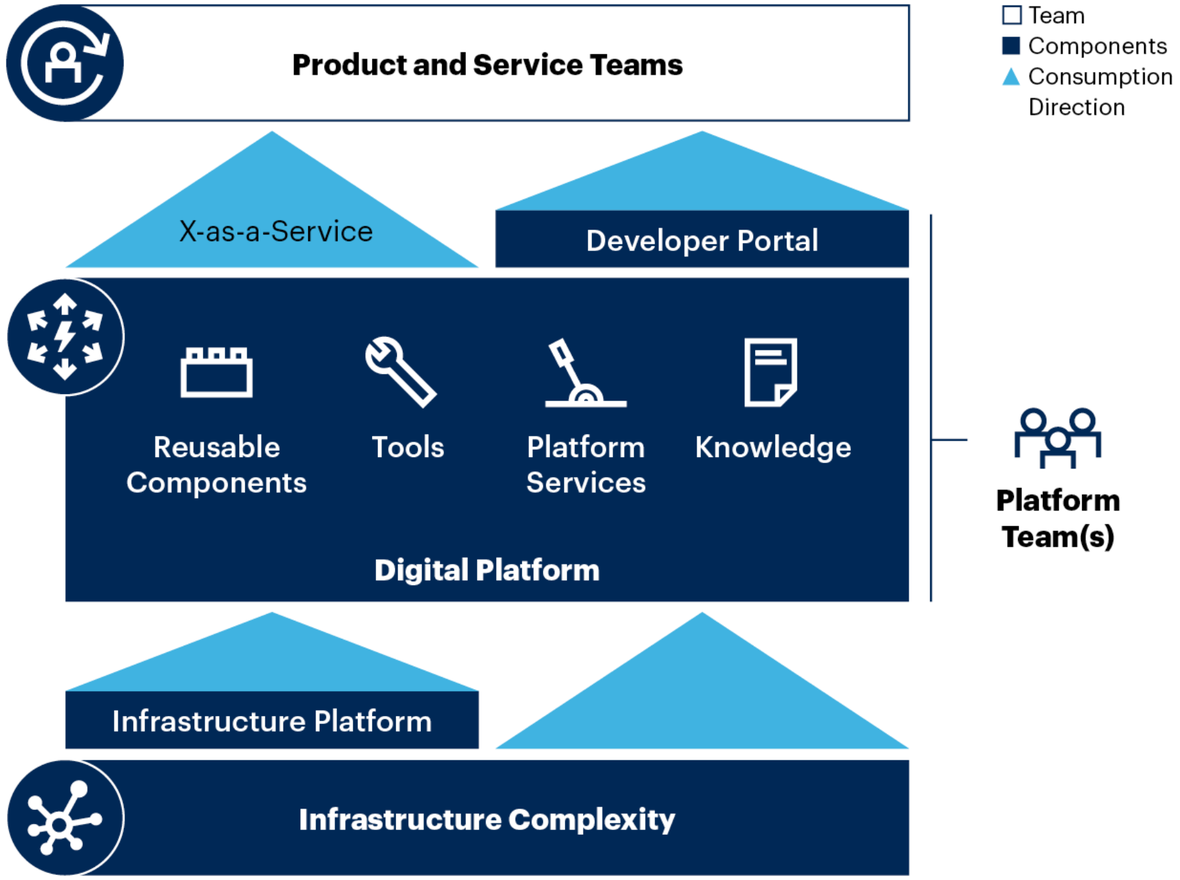
Platform Engineering

Before jumping to definition, let's understand the problem...



“The composition and integration of a set of processes, tools and automation (components) to build a coherent platform with the goal of empowering developers to be able to easily build, maintain and deploy their business logic”

Diagram of Platform Engineering



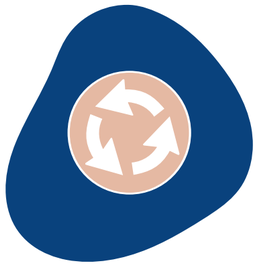
gartner.com

Source: Gartner

Gartner

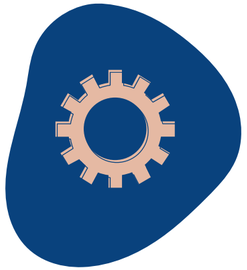
Carrier as a DevOps Engineer

DevOps Engineer Role



CI/ CD Management & Automation

Cloud Deployment and Management



Infrastructure Management

Performance Assessment and Monitoring

Writing Specifications and Documentation

Assisting with DevOps culture apdotion

References

- https://sre.google/sre-book/table-of-contents/
- https://www.gartner.com/en/articles/what-is-platform-engineering
- https://youtu.be/uTEL8Ff1Zvk?si=5QT_LrzedX-BMezt

LinkedIn

https://www.linkedin.com/in/ravindufernando/

X (Twitter)

@ravindunf

Thank You!

Introduction to Microservices

Ravindu Nirmal Fernando

SLIIT | March 2025

Foundations of Modern Software Architecture: Paving the Way for Microservices

- Influential Concepts and Technologies
  - **Domain-Driven Design:** Emphasizing the importance of reflecting real-world complexities in our code for better system modeling.
  - **Continuous Delivery:** Revolutionizing software deployment, making every code check-in a potential release candidate.
  - **Web Communication Advancements:** Enhancing how machines interact, leading to more efficient and robust systems.
- Architectural Shifts
  - **From Layered to Hexagonal:** Moving away from traditional layered architectures to avoid hidden complexities in business logic.
  - **Embracing Virtualization:** Utilizing on-demand provisioning and resizing of resources for greater flexibility with cloud computing.
- Organizational Practices
  - **Small Autonomous Teams:** Inspired by tech giants like Amazon and Google, promoting ownership and lifecycle management of services.
  - **Learning from Netflix:** Building resilient, scalable systems that can withstand and adapt to change.

Microservices: A Natural Progression

- Emergence from Real-World Use: Microservices weren’t pre-planned but evolved as a response to practical needs in software development.
- Responding to Change: Offering the agility and flexibility to adapt to new technologies and market demands.

Monolithic Applications

- **Basic Structure**
  - Single-Tiered Structure: Built as a single, unified unit.
  - Combined Modules: Functional modules like UI, server logic, and database interactions are combined.
- **Design and Construction**
  - Modular Architecture: Follows a modular structure within a single unit, aligning with object-oriented principles.
  - Programming Constructs: Defined using language-specific constructs (e.g., Java packages).
  - Build Artifacts: Built as a single artifact, such as a Java JAR file.
- **Characteristics**
  - Inter-module Dependencies: Modules are tightly coupled and interdependent.
  - Unified Deployment: Deployed as a single entity.
- **Scalability**
  - Scalability Approach: Scaling involves replicating the entire application, not individual components.

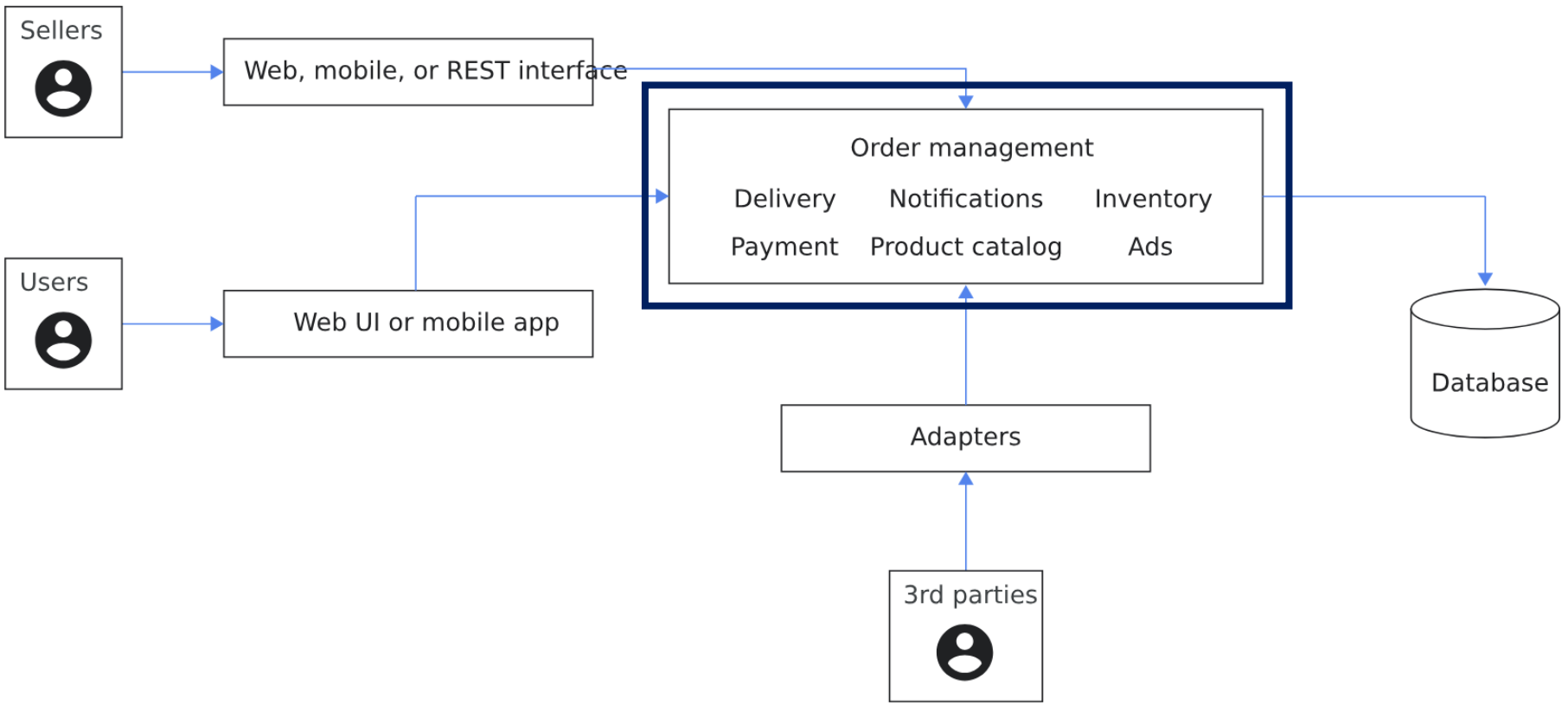


Diagram of a monolithic ecommerce application with several modules using a combination of programming language constructs. (https://cloud.google.com/architecture/microservices-architecture-introduction)

- **Benefits of Monolithic Architecture**
  - **Simplified Testing**: Tools like Selenium enable end-to-end testing of the entire application.
  - **Ease of Deployment**: Deployment involves simply copying the packaged application to a server.
  - **Resource Sharing**: All modules share memory, space, and resources, streamlining cross-cutting concerns like logging, caching, and security.
  - **Intra-Process Communication**: Direct module-to-module calls can offer performance advantages over network-dependent microservices.

- **Challenges of Monolithic Architecture**
  - **Scalability Issues**: Difficulty in scaling when different modules have conflicting resource requirements.
  - **Complexity in Maintenance and Updates**: As the application grows, implementing changes becomes more complicated due to tightly coupled modules.
  - **CI/CD Complications**: Continuous integration and deployment become challenging as any update requires redeploying the entire application.
  - **Vulnerability to System Failures**: A bug in any module, like a memory leak, can crash the entire system.
  - **Technological Rigidity**: Adopting new frameworks or languages is costly and time-consuming, as it often requires rewriting the entire application.

Understanding Microservices

- **Core Characteristics**
  - **Small and Focused**: Aimed at doing one thing well, avoiding sprawling codebases.
  - **Cohesion and Single Responsibility**: Adhering to the principle of grouping related code and separating unrelated functionalities.
- **Size and Scope**
  - **No Fixed Size**: Size varies based on language expressiveness and domain complexity.
  - **Team Alignment**: Ideally sized to be managed by a small team.
  - **Balance in Size**: Smaller services maximize benefits but increase complexity.

- **Autonomy**
  - **Independent Entities**: Deployed separately, can be different technologies, possibly as isolated services on a PAAS or as individual operating system processes.
  - **Network Communication**: Services communicate via network calls, ensuring separation and reducing tight coupling.
- **Deployment and Change Management**
  - **Independent Deployment**: Services can be deployed independently without impacting others.
  - **API-Centric Interaction**: Services expose APIs for interaction, emphasizing decoupled, technology-agnostic interfaces.
- **Decoupling**
  - **Key to Microservices**: Essential for maintaining independence and achieving the benefits of microservices architecture.
  - **Change and Deployment**: Ability to change and deploy a service independently is crucial.

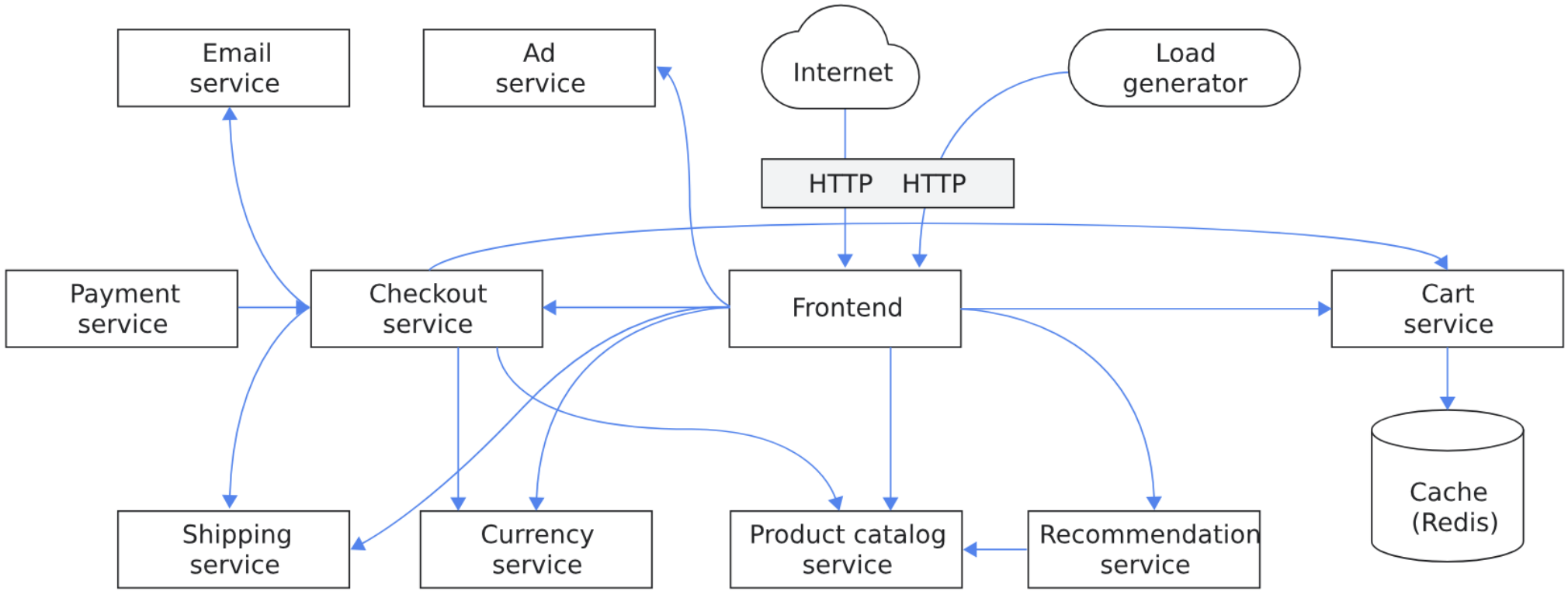
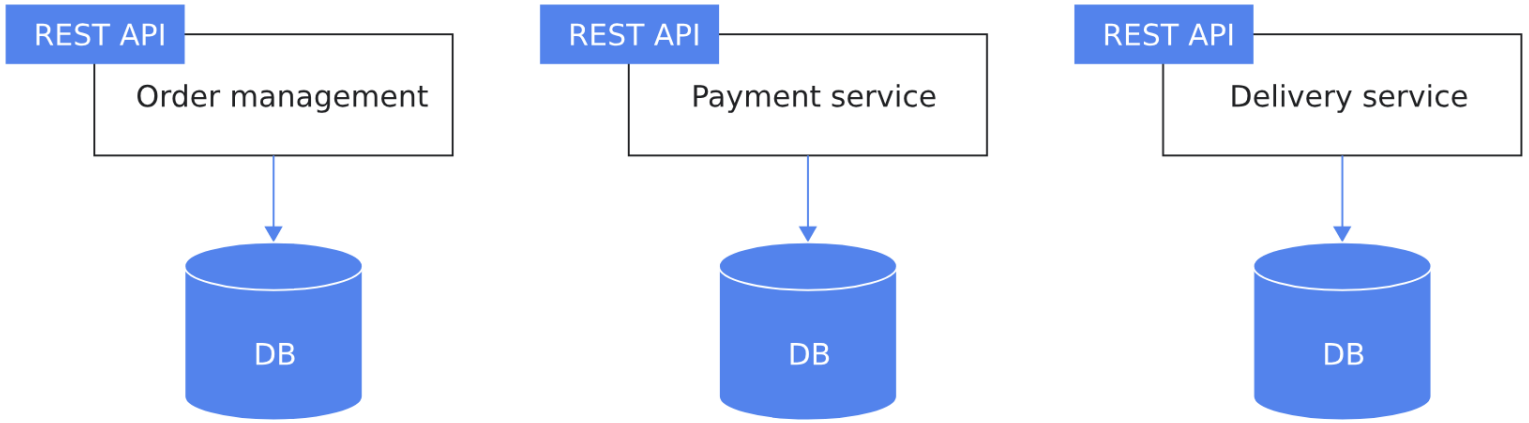


Diagram of an ecommerce application with functional areas implemented by microservices. (https://cloud.google.com/architecture/microservices-architecture-introduction)

- **Database Relationship**
  - **Service-Specific Databases**: Each microservice has its own database tailored to its requirements.
  - **Loose Coupling**: This approach ensures loose coupling by routing data requests through service APIs instead of a shared database.
  - **Independent Data Management**: Each service manages its data independently, enhancing autonomy and reducing interdependencies.



Benefits of Microservices Architecture

Aspect	Benefit Details
Enhanced Development and Maintenance	- Breaks application into smaller, manageable chunks. - Clear boundaries with defined APIs. - Quicker development, easier understanding and maintenance.
Team Autonomy and Efficiency	- Independent development of services by teams. - Full lifecycle ownership of services. - Flexibility to use different programming languages (Polyglot Development).
Improved Scalability and Market Responsiveness	- Independent scaling based on service needs. - Hardware optimization for resource requirements. - Faster product delivery and improved time to market.

Challenges of Microservices Architecture

Challenge Category	Challenge Details
Complexity in Distributed Systems	- Necessity of choosing and implementing inter-service communication mechanisms. - Managing partial failures and service unavailability.
Transaction Management Across Services	- Handling atomic operations across multiple microservices (Distributed Transactions). - Maintaining data consistency during failures (Consistency Issues).
Testing and Deployment Complexities	- Requirement for comprehensive testing across multiple services. - Complexities in managing multiple service deployments and service discovery.
Operational Overhead	- Increased need for monitoring and alerting across more services. - Higher risk of failure due to more points of service-to-service communication. - Challenges in productionizing and maintaining robust operations infrastructure.
Performance and Suitability Considerations	- Potential latency issues due to network calls between services. - Not suitable for all types of applications, especially those requiring real-time data processing. - Importance of clear communication and service boundary planning.

Migrating from Monolithic to Microservices: Key Considerations

Consideration Category	Details
Assessing the Need for Migration	- Evaluate if microservices align with business goals and pain points. - Consider simpler alternatives like autoscaling or enhanced testing.
Starting the Migration Process	- Begin with extracting and deploying one service independently. - Adopt an iterative approach, learning and adapting with each service migration.
Strategic Implementation	- Recognize varying approaches to microservice size and quantity among teams. - Emphasize continuous learning and strategy refinement.
Future Learning and Strategies	- Explore strategies for detailed refactoring from monolithic to microservices. - Plan for ongoing education and adaptation of methods.

References

- https://cloud.google.com/architecture/microservices-architecture-introduction
- Building Microservices, Sam Newman

Introduction to Machine Learning

By Jeewaka Perera

Lesson Objectives

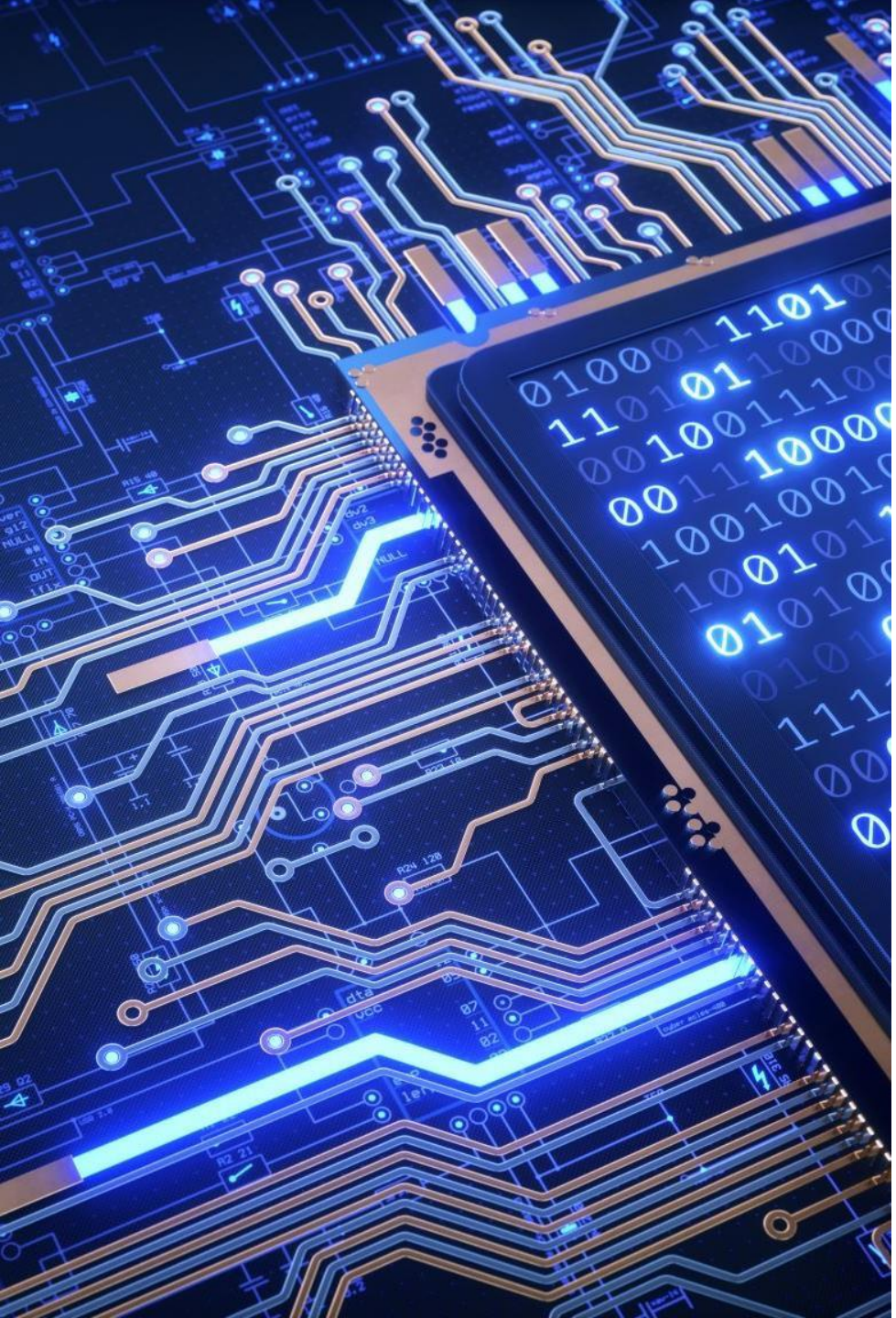
- At the end of the lesson students should be able to explain
  - What is Machine Learning?
  - Why Choose Machine Learning?
  - Different Machine Learning Algorithms and their applications

What is Optimization?

- **Optimization** is the mathematical discipline which is concerned with finding the maxima and minima of functions, possibly subject to constraints.

What is Artificial Intelligence

- " It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.“ by John McCarthy



What is Machine Learning

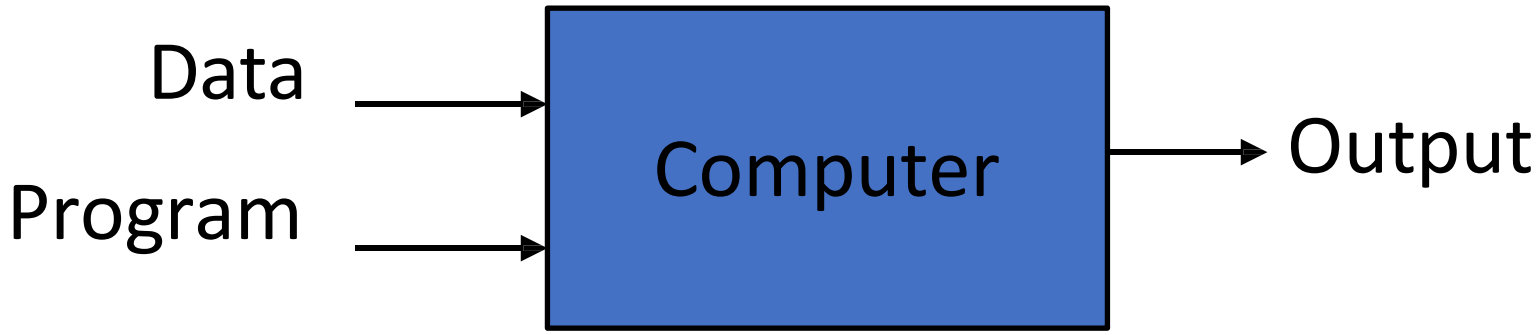
- “field of study that gives computers the ability to learn without explicitly being programmed.” by Arthur Samuel
- Machine learning is a subfield of artificial intelligence, which is broadly defined as the capability of a machine to imitate intelligent human behavior.

Learning in a Machine

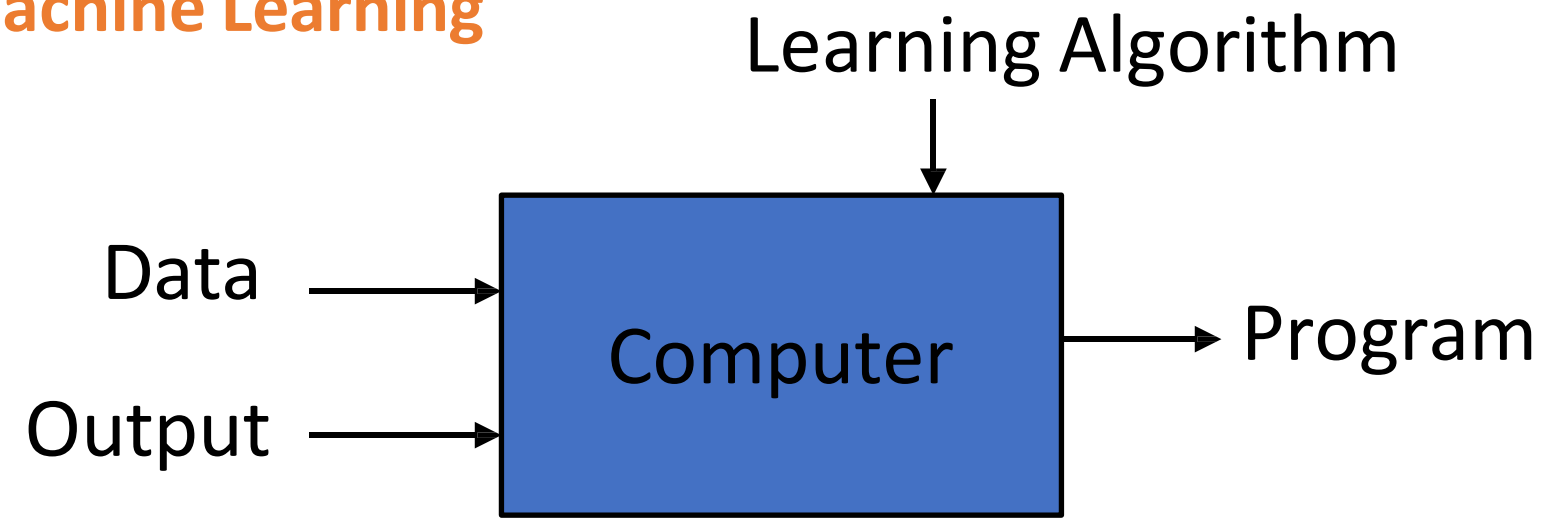
- “A computer program is said to learn from experience (E) with some class of tasks (T) and a performance measure (P) if its performance at tasks in T as measured by P improves with E”

ML vs Traditional programming

Traditional Programming



Machine Learning



Types of Artificial Intelligence Algorithms

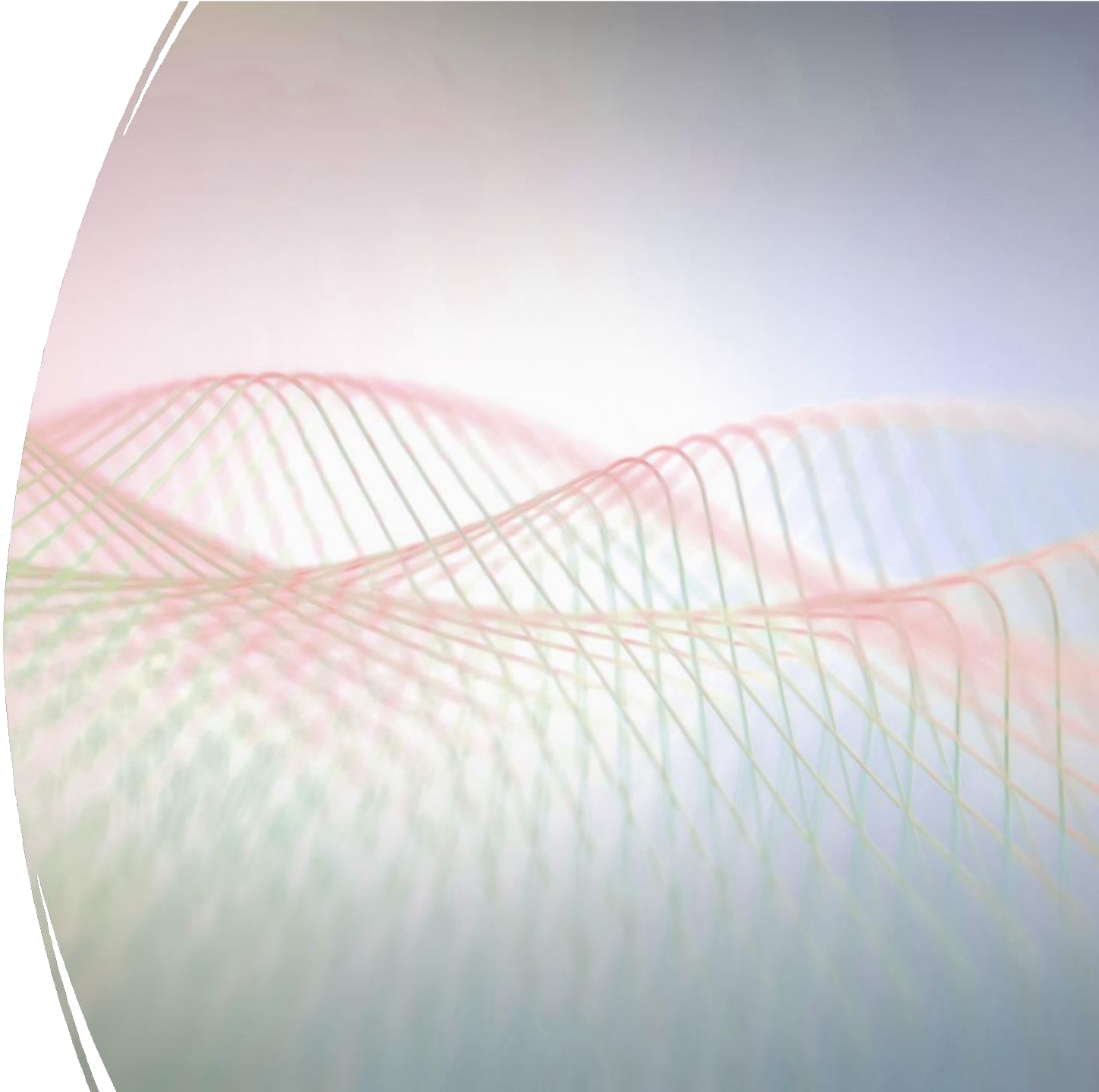
Rule-based expert

Systems

Search Algorithms

Evolutionary Algorithms and Swam Intelligence Algorithms

Machine Learning Algorithms

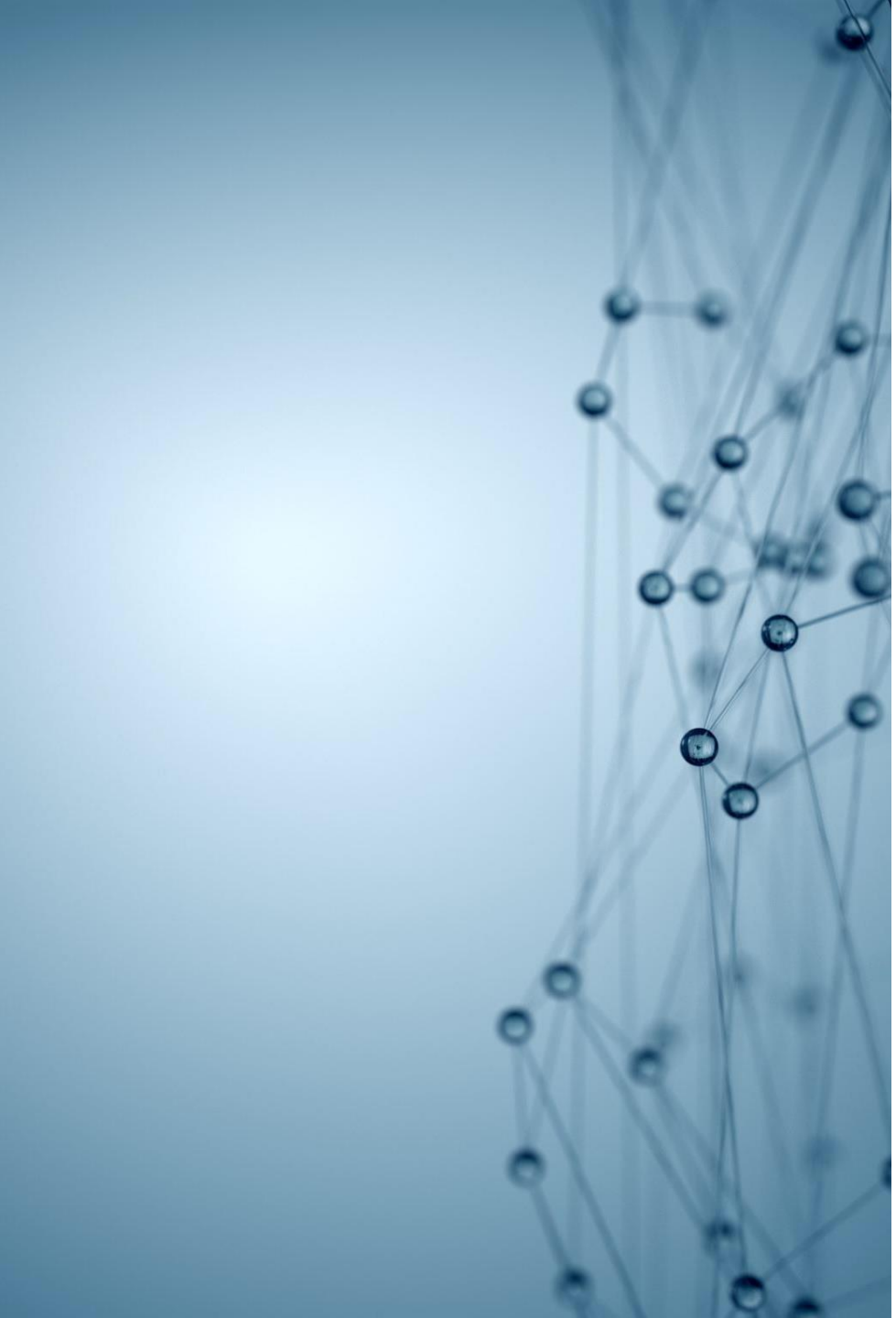


Rule Based Systems

- Expert Systems
- PROSPECTOR
- MYCIN
- Based on pre-defined Rules
- Rules defined based on domain knowledge
- Designed to mimic the decision-making process of human experts

Search Algorithms

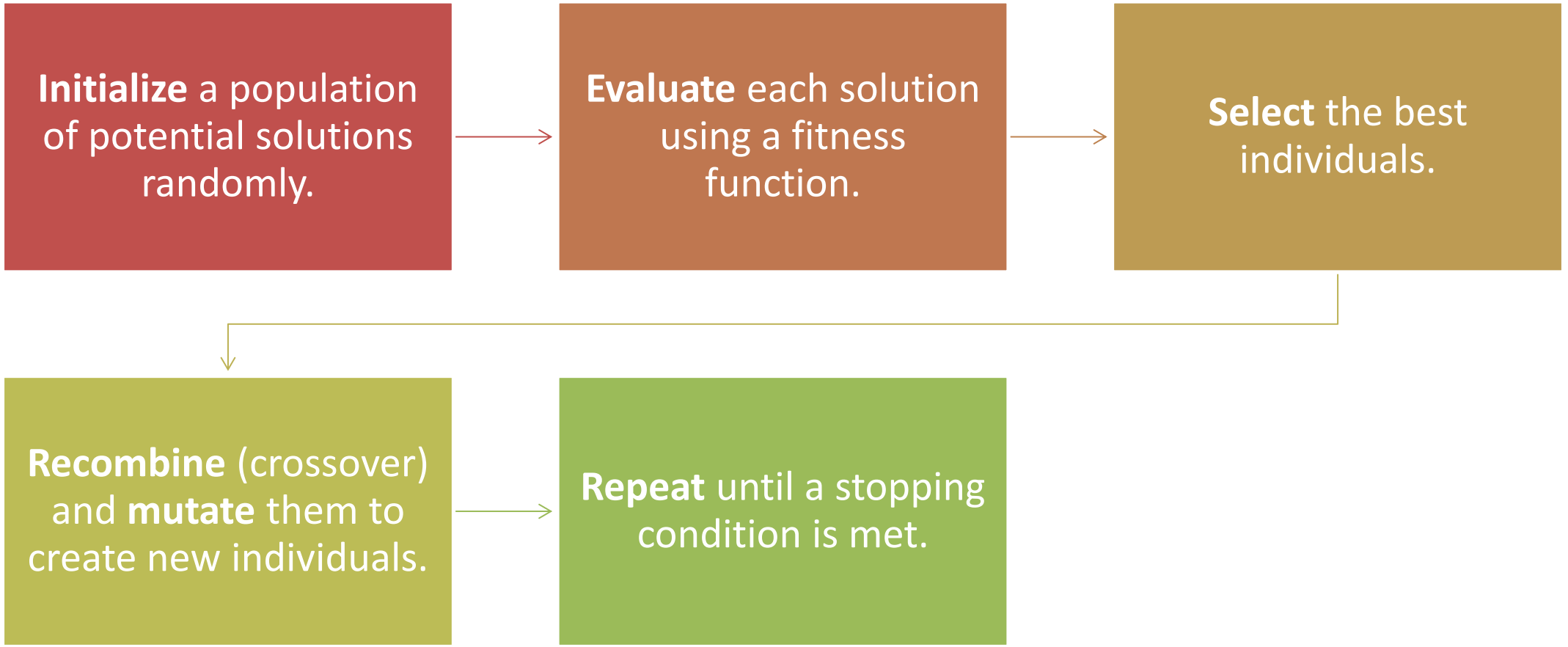
- Breadth-First Search
- Depth First Search
- Iterative Deepening Search
- Uniform Cost Search
- Dijkstra's algorithm
- A* Search



Evolutionary Algorithms

- Evolutionary Algorithms are a family of nature-inspired optimization algorithms that mimic biological evolution—natural selection, mutation, recombination, and survival of the fittest.
  - Genetic Algorithms
  - Particle Swarm Optimization
  - Cultural Algorithms
  - HCA KCA
  - SI Algorithms(ACO, FA)

Steps in a generic Evolutionary Algorithm



Applications of EA

SCHEDULING AIRLINE CREWS

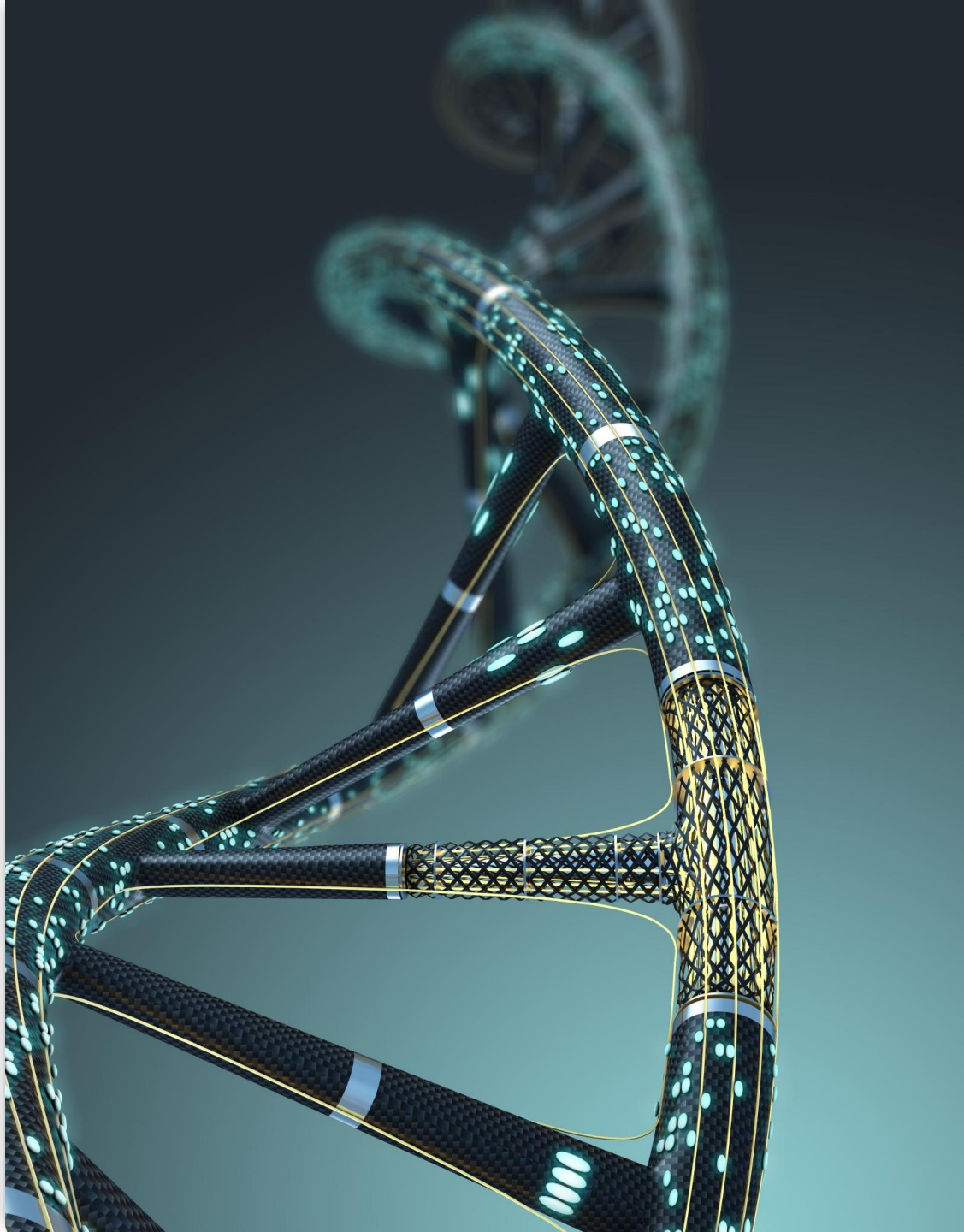
TUNING HYPERPARAMETERS IN ML

DESIGNING ANTENNAS (NASA EXAMPLE!)

PORTFOLIO OPTIMIZATION

Genetic Algorithms

- A Genetic Algorithm (GA) is a search and optimization method inspired by how living things evolve over time through natural selection.
- We represent potential solutions as chromosomes.
- Better solutions (those with higher fitness) are selected
- New solutions are created through crossover and mutation
- Over generations, solutions get better!



Swarm Intelligence Algorithms

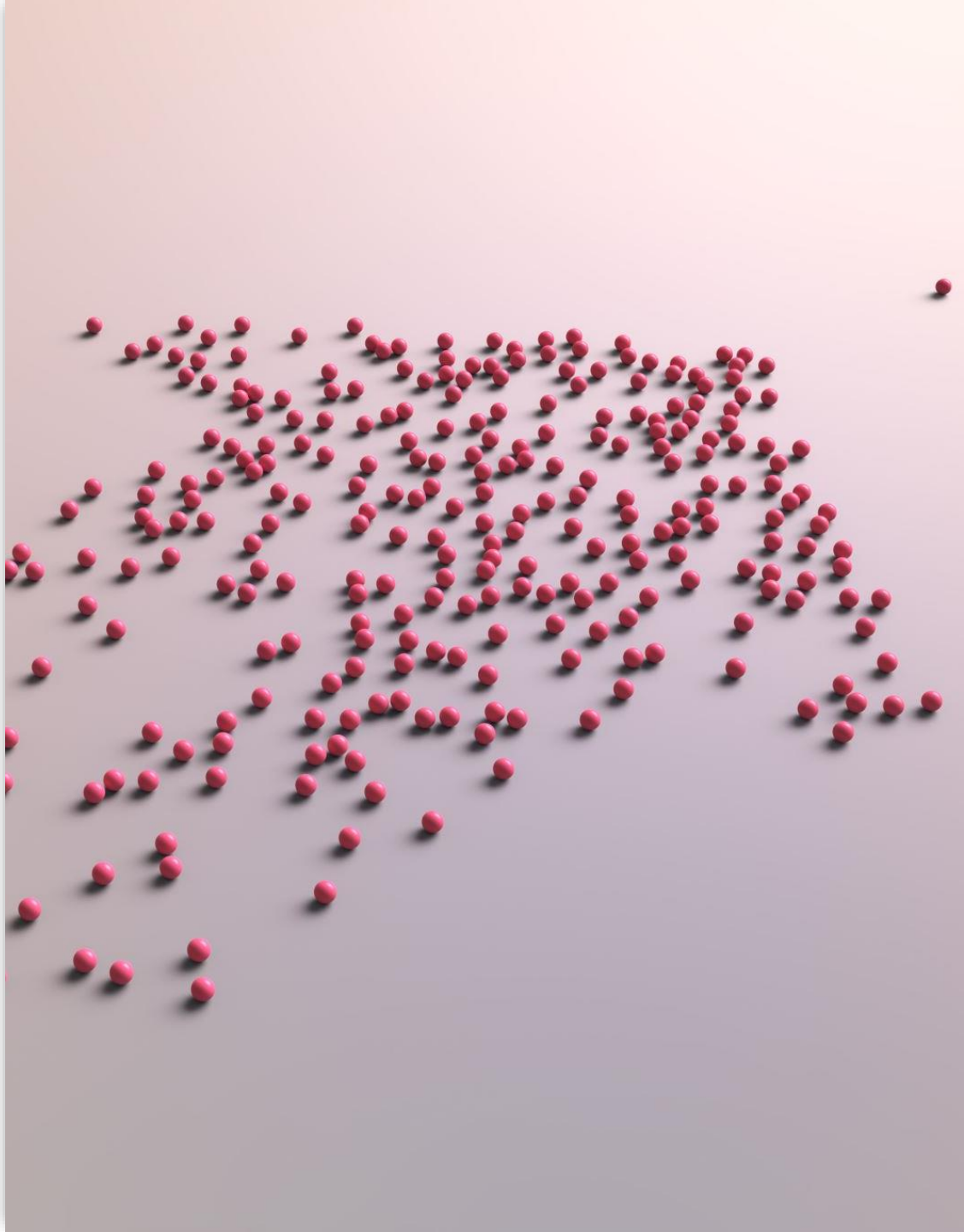
Inspired by: Collective behavior of decentralized, self-organized systems (e.g., birds, ants, fish)

Ant Colony Optimization (ACO)

- Inspired by ants finding shortest paths using **pheromone trails**.
- Good for **discrete path-based problems** like the **Traveling Salesman Problem (TSP)**.

Firefly Algorithm (FA)

- Fireflies are attracted to brighter (better) solutions.
- Uses light intensity and distance for movement.



Types of Machine Learning Algorithms

SUPERVISED LEARNING

UNSUPERVISED LEARNING

SEMI-SUPERVISED LEARNING

REINFORCEMENT LEARNING

Supervised Learning Algorithms

- Learned under supervision.
  - Supervision of what?
  - Humans?
- Supervised by the Labeled data
  - Require labeled data. (Inputs, output)
  - This is the most difficult part of supervised learning.

Types of Supervised Learning Models

- Regression
  - Predicting a Linear value
  - Linear Regression
  - SVR
  - DT
- Classification
- Predicting a class/label
  - Logistic Regression
  - SVM
  - DT
  - NB

Artificial Neural Network Models such as MLP, CNN, RNNs are Considered as supervised Learning Models

Supervised learning examples

A Bank may have borrower details (age, income, gender, etc.) of the past **(features)**

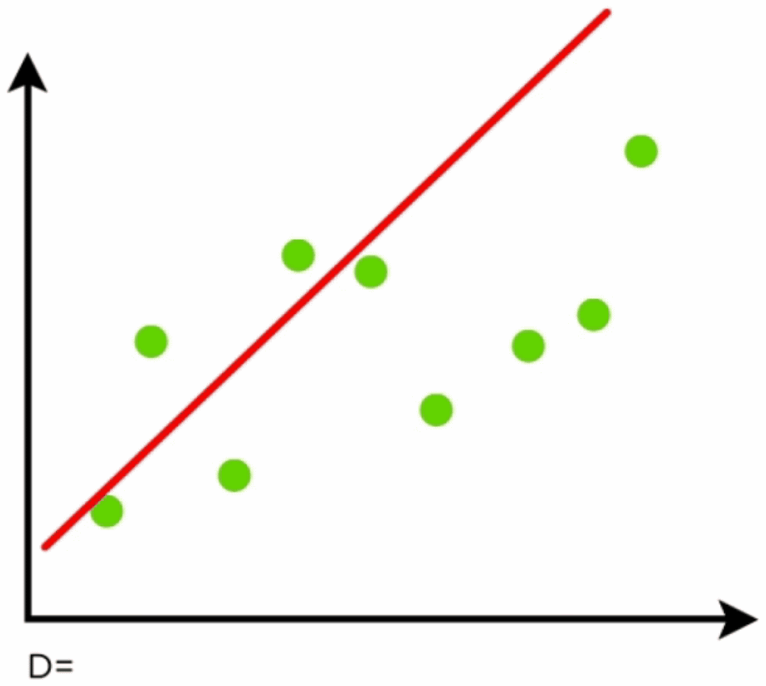
Also it may have details of the borrowers who defaulted in the past **(labels)**

Based on the above, can train a classifier to learn the patterns of borrowers who are likely to default on their payments

Linear Regression

- **Model Explanation:**
  Draws a straight line that best fits the data.
- **Key Concept:**
  Learns the relationship as a **linear equation**:
  $y = mx + c$.
- **What is learnt through training:**
  Finds the best slope (m) and intercept (c) to minimize error.
- **Example Use Cases:**
  - Predicting house prices
  - Salary estimation
  - Sales forecasting
- **Limitations:**
  - Only works well when the relationship is **linear**
  - Not suitable for complex, non-linear patterns
  - Sensitive to **outliers**

Linear regression



“Predictor”:
Evaluate line:

$$r = \theta_0 + \theta_1 x_1$$

return r

Types of Unsupervised Learning Algorithms

- Clustering Algorithms
  - K Means
  - DBSCAN
- Dimensionality Reduction Algorithms
  - PCA
  - MDS (Multidimensional Scaling)
  - LDA (Linear Discriminant Analysis)

- Graph Based Models can be considered as Unsupervised Learning

Unsupervised learning examples

A Supermarket may store each buyer's

basket content details **(features)**

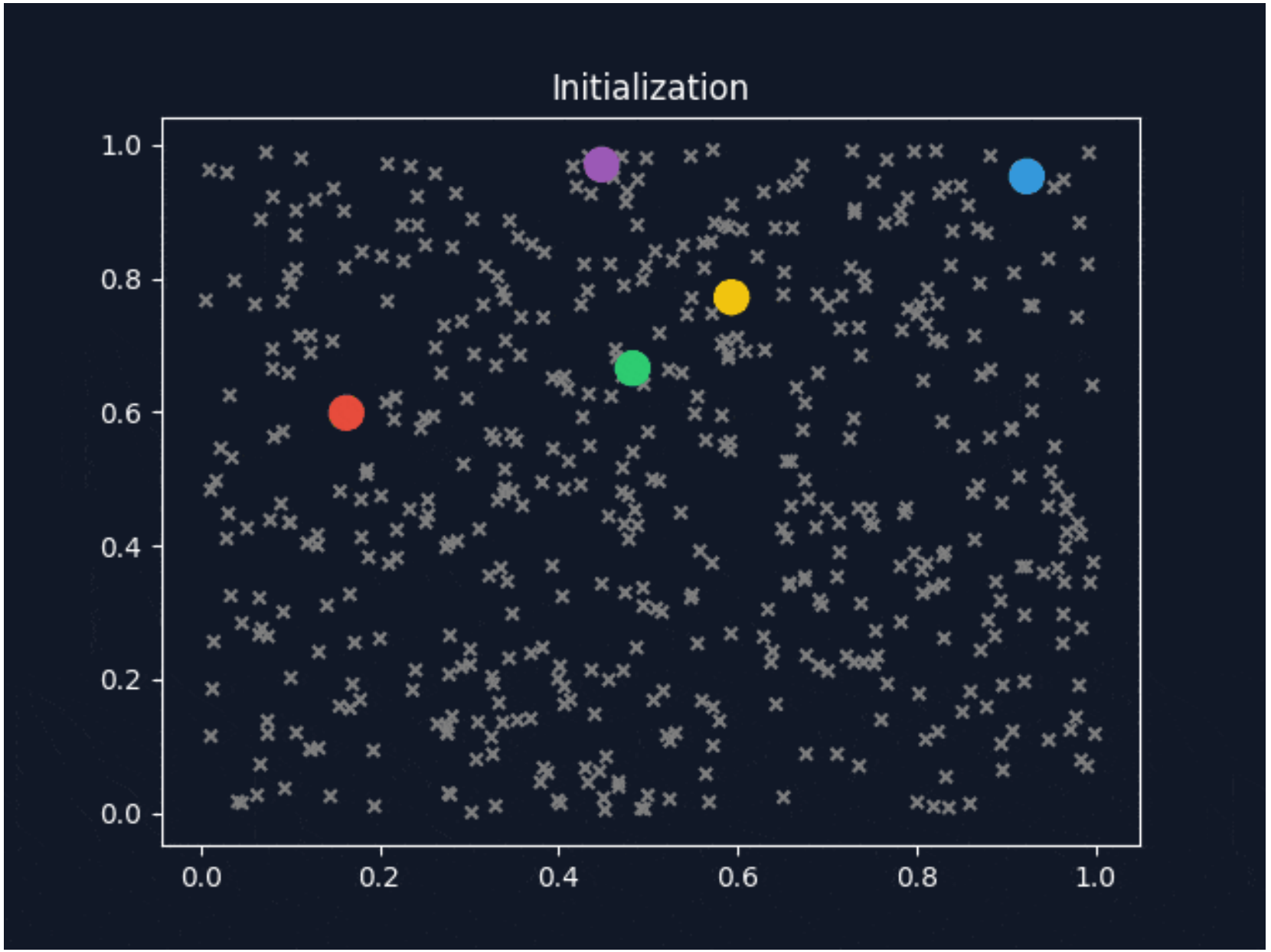
There are **NO** grouping **(labels)**

Need to group the buyers based on their buying patterns in order to best use the shelf space (recommendation)

K- Means Clustering

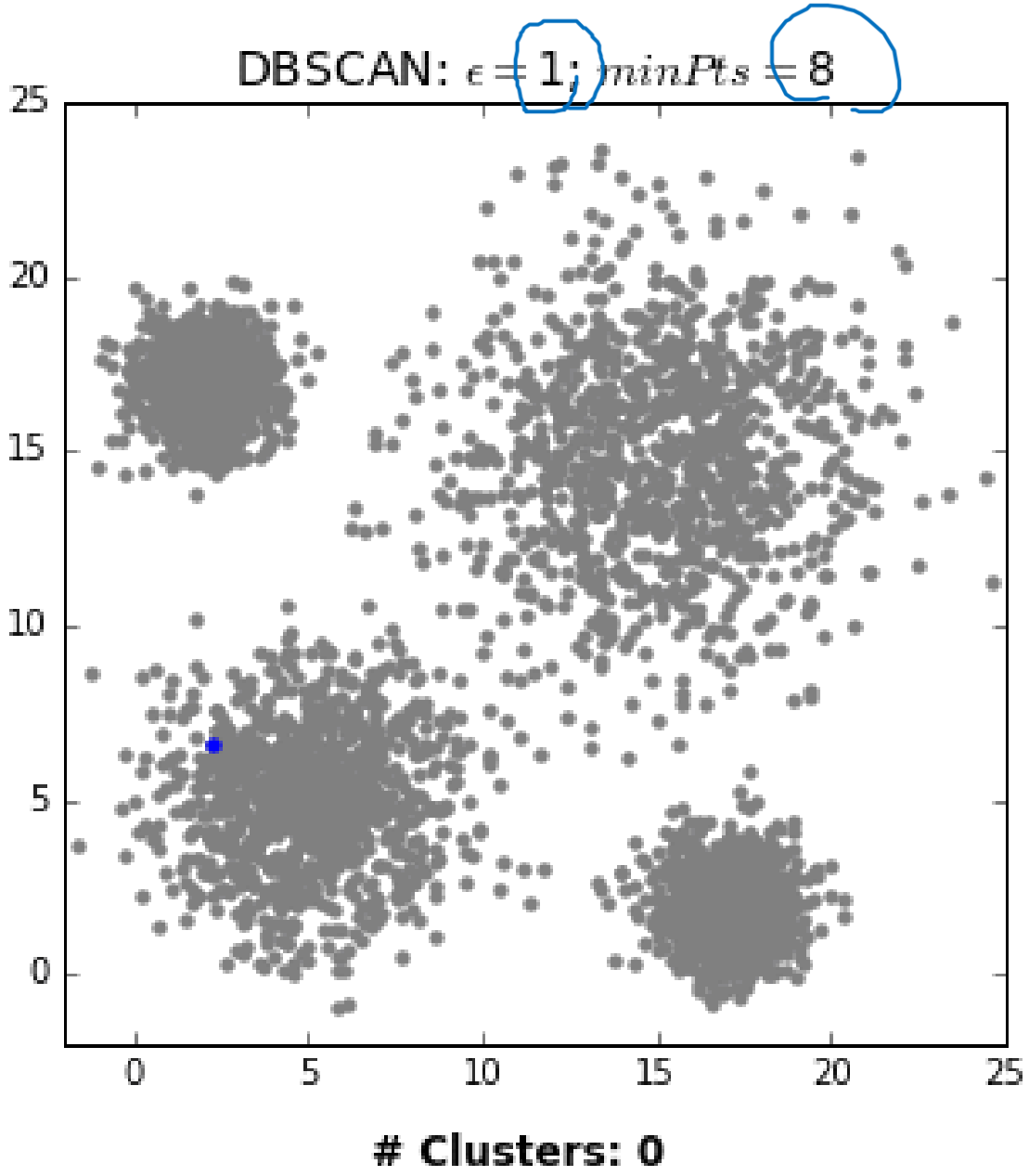
- **Model Explanation:**
  - Groups data into **K clusters** based on similarity.
- **Key Concept:**
  - Finds **cluster centers** and assigns each point to the nearest one.
- **What is learnt through training:**
  - Learns the **position of cluster centers**.
- **Example Use Cases:**
  - Customer segmentation
  - Grouping articles by topic
  - Organizing images
- **Limitations:**
  - You must choose **K (number of clusters)** beforehand
  - Assumes **spherical-shaped clusters**
  - Struggles with **uneven or noisy data**

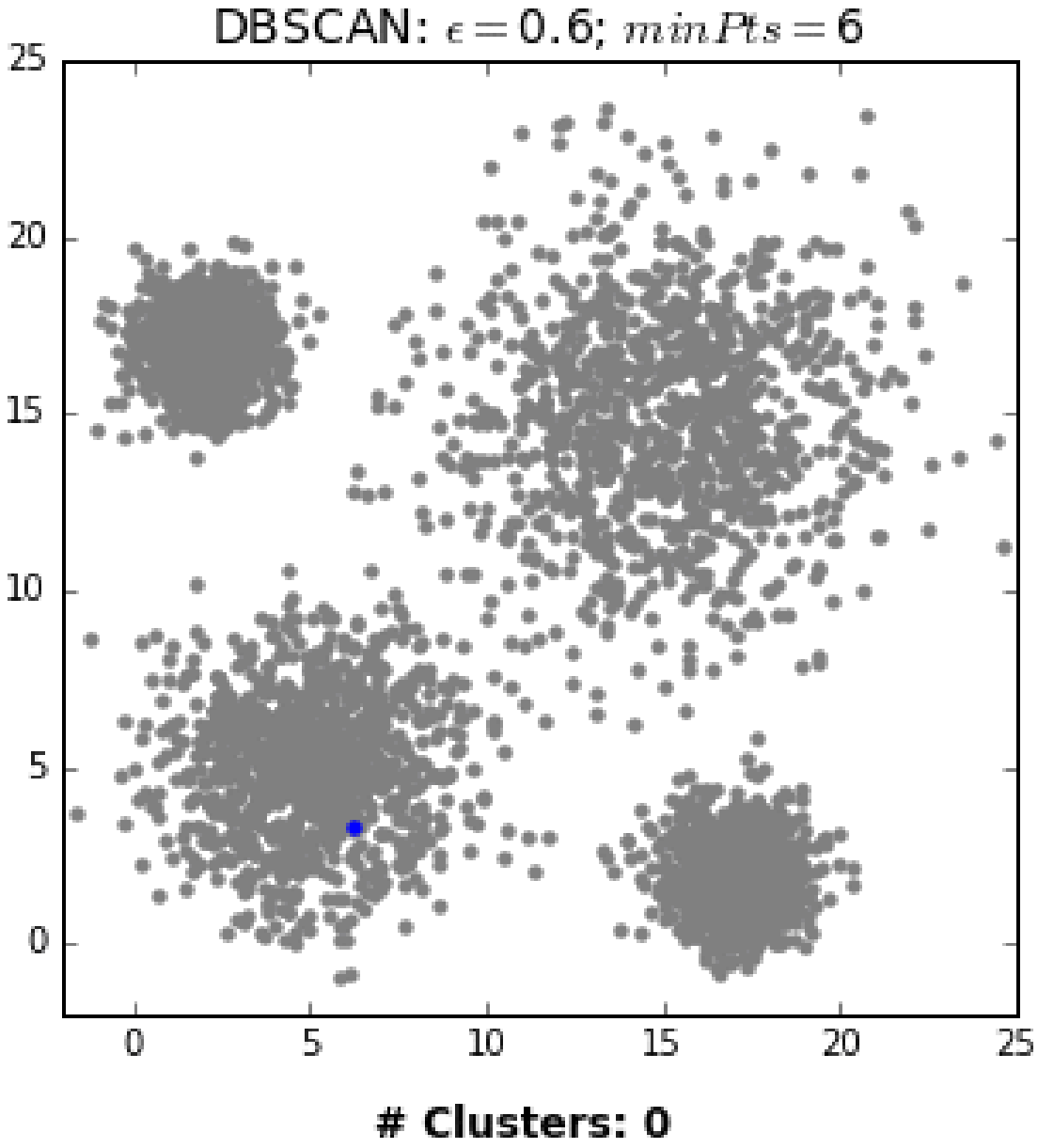
K-means clustering example

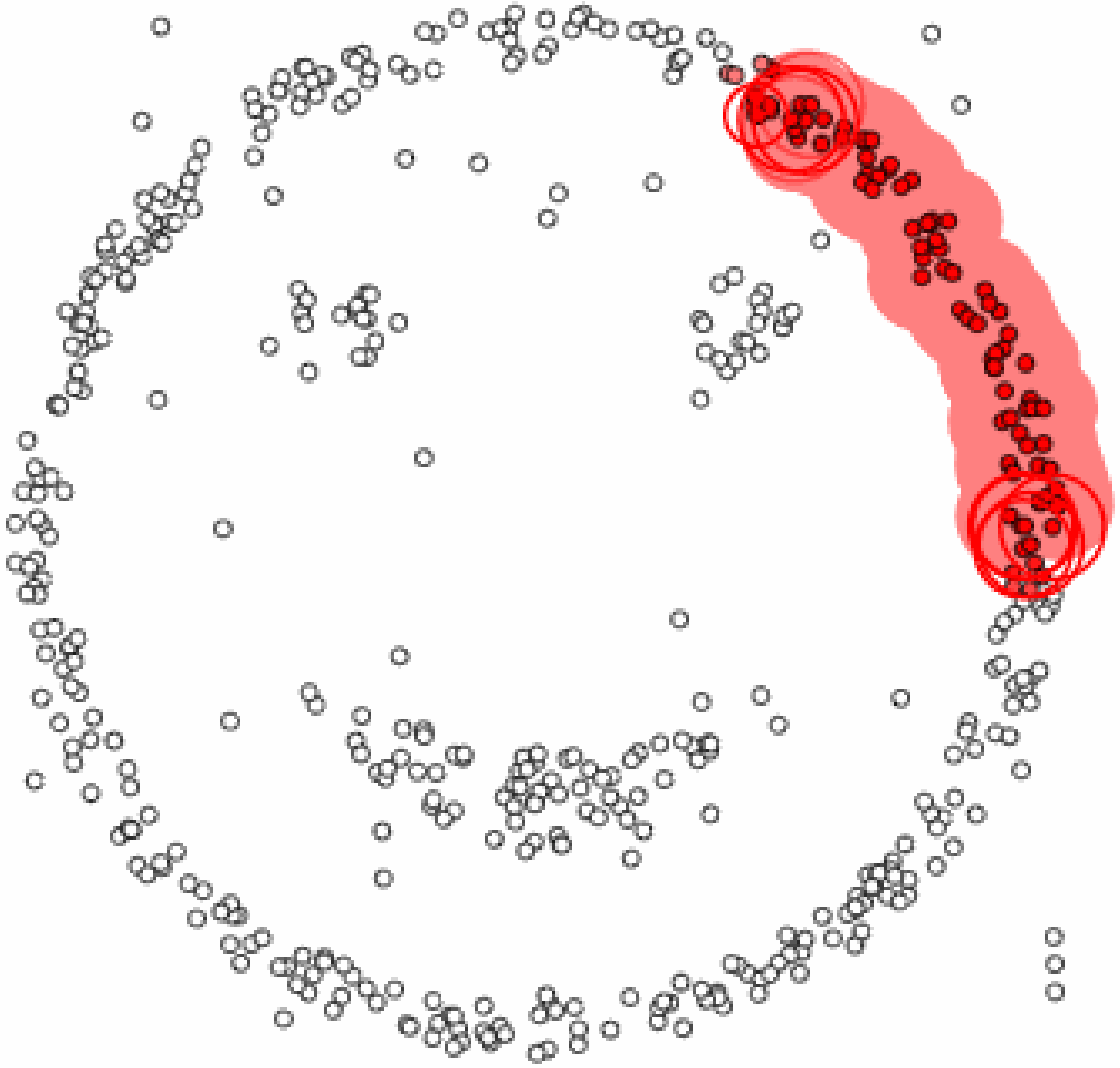


DBSCAN

- **Model Explanation:**
  - Groups together dense areas; labels sparse points as outliers.
- **Key Concept:**
  - Clusters are formed based on **density**, not shape or number.
- **What is learnt through training:**
  - Learns which points belong to **dense clusters** and which are **noise**.
- **Example Use Cases:**
  - Fraud detection
  - Identifying event hotspots
  - Anomaly detection in GPS or sensor data
- **Limitations:**
  - Struggles with **varying densities**
  - Requires tuning of **eps** and **min points**
  - Not ideal for **high-dimensional data**


DBSCAN: $\epsilon = 1$; $minPts = 8$
25
20
15
10
5
0
0
5
10
15
20
25
Clusters: 0


DBSCAN: $\epsilon = 0.6$; $minPts = 6$
25
20
15
10
5
0
0
5
10
15
20
25
Clusters: 0



epsilon = 1.00
minPoints = 4

Restart

Pause

Semi-supervised learning

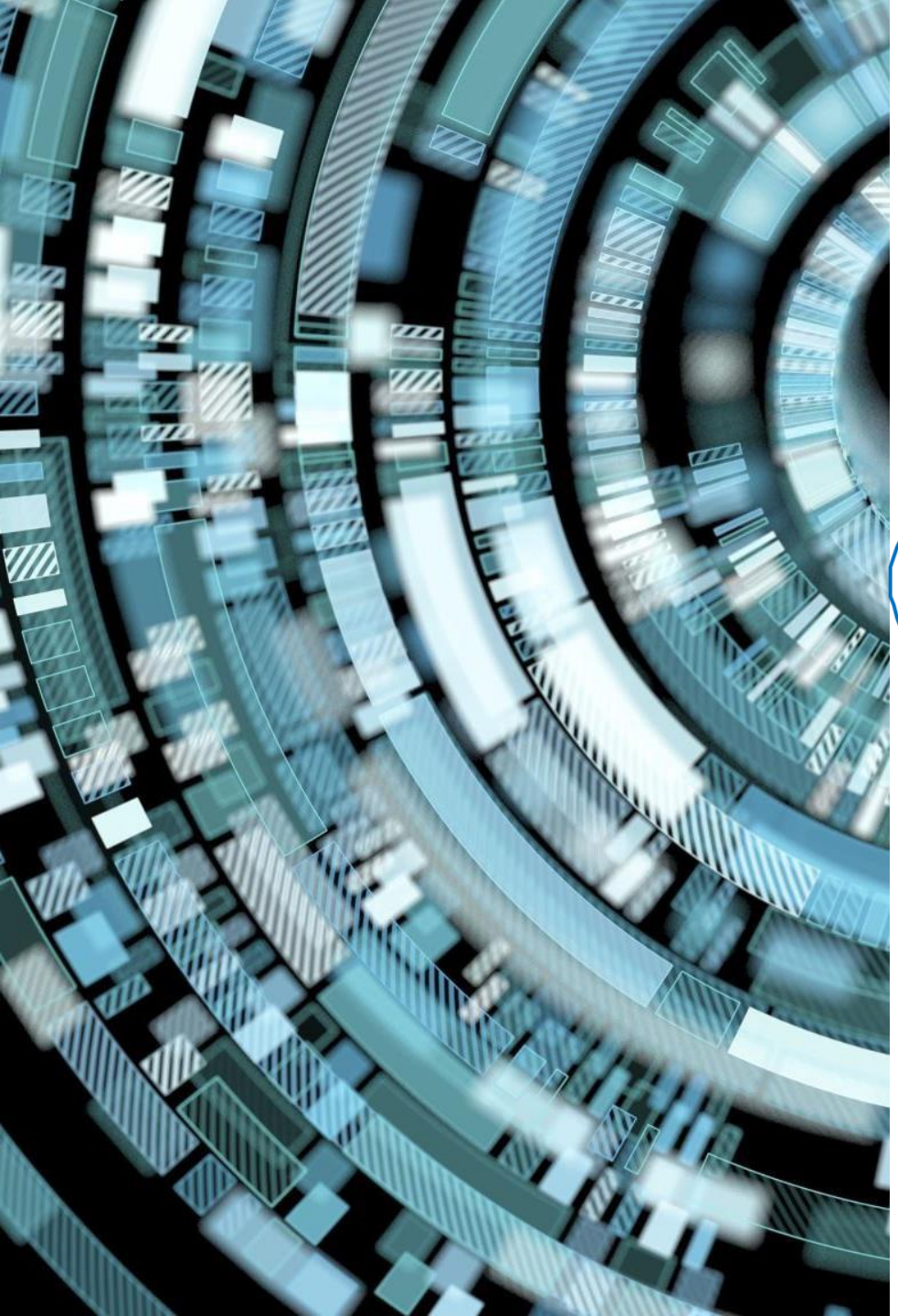
Labeled data is expensive/difficult to get

Unlabeled data is cheap/easier to get

The idea is to use smaller amount of labelled data with larger amount of unlabeled data to creating the training/testing datasets

Algorithms - Self Training, Generative models

- Semi-Supervised Support Vector Machines, etc.

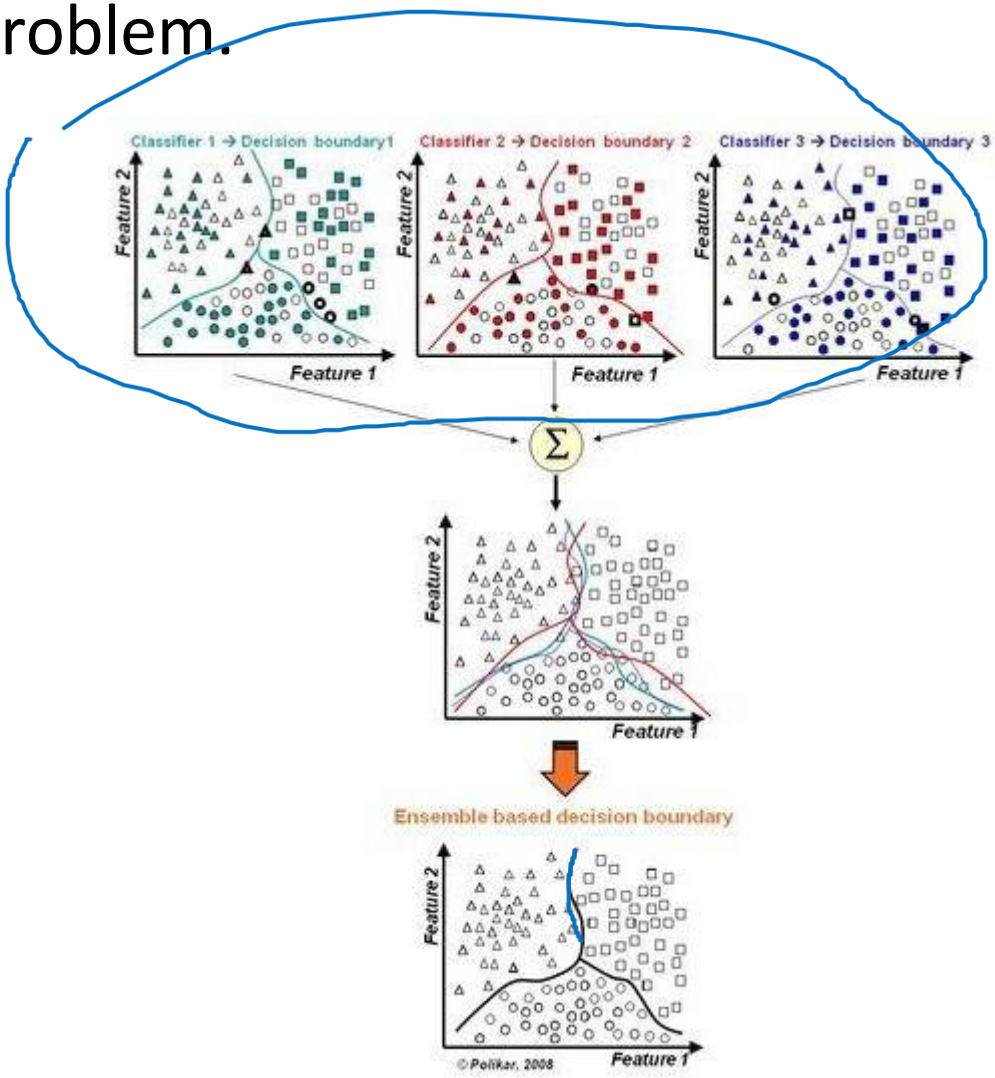


Semi-Supervised Learning Algorithms

- Generative Adversarial Networks
- Auto-encoders
- Variational Auto-encoders

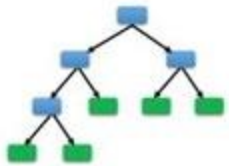
Ensemble Learning

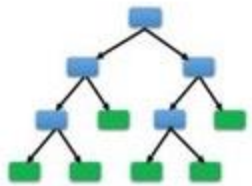
- Often, multiple classifiers need to be combined to solve a real-world problem.

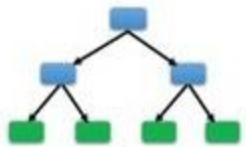


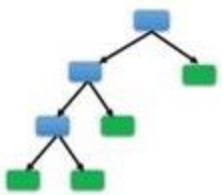
Random Forest

- **Model Explanation:**
  - Combines predictions from many decision trees.
- **Key Concept:**
  - Uses **voting** or **averaging** to make decisions.
- **What is learnt through training:**
  - Learns different rules from subsets of data to make **robust predictions**.
- **Example Use Cases:**
  - Spam detection
  - Credit risk analysis
  - Disease classification
- **Limitations:**
  - Can be **slow** with large datasets
  - Less interpretable than a single decision tree
  - Needs tuning (like number of trees)









Random Forest in Action!!!

Reinforcement Learning

- **Model Explanation:**
  - Learns by interacting with an environment and receiving feedback (rewards or penalties).
- **Key Concept:**
  - Uses **trial-and-error** and **reward maximization** to improve decision-making over time.
- **What is learnt through training:**
  - Learns an **optimal policy** or **action strategy** that maximizes long-term rewards.
- **Example Use Cases:**
  - Game playing (e.g., AlphaGo)
  - Robotics (e.g., navigation or motor control)
  - Dynamic pricing or recommendation systems
- **Limitations:**
  - Requires **many interactions** with the environment (sample inefficiency)
  - Can be **unstable** or hard to converge
  - Needs **careful reward design** to avoid unintended behaviors

Reinforcement learning examples

A group of robots have been deployed in an unknown territory

The objective is for them to collaboratively find the navigation path to reach a particular destination/goal

Can use reinforcement learning where achieving the goal/getting closer to the goal gives a positive reward. Negative reward otherwise

Can share the information among robots (multi-agent system)

Comparing Machine Learning Models

Not all models are created equal — and neither are the ways we evaluate them.

Key Questions to Ask:

Is it a **classification**, **regression**, or **clustering** task?	Do we care more about **correctness**, **fairness**, or **interpretability**?	What are the **costs of wrong predictions**?

Common Evaluation Matrices

Task Type	Metrics Used
Classification	Accuracy, Precision, Recall, F1 Score, ROC-AUC
Regression	MSE, MAE, RMSE, R^2 Score
Clustering	Silhouette Score, Davies-Bouldin Index, Inertia
Ranking/Recommendation	MAP, NDCG, Hit Rate

Classification Matrices Explained

Metric	Use When...	Notes
Accuracy	Classes are balanced, all errors matter	Can be misleading with imbalance
Precision	False positives are costly (e.g., spam)	TP / (TP + FP)
Recall	False negatives are costly (e.g., cancer)	TP / (TP + FN)
F1 Score	Balance between precision and recall	Harmonic mean
ROC-AUC	Need to evaluate ranking ability	Works for probabilistic models

Regression Model

Metric	Use When...	Notes
MSE	Large errors are very bad	Penalizes large errors more
MAE	Equal penalty for all errors	More robust to outliers
RMSE	Like MSE but in original units	Square root of MSE
R^2 Score	Want to explain variability in output	1 = perfect, 0 = no

Clustering Algorithms

Metric	Use When...	Notes
Silhouette Score	Want to measure how distinct clusters are	1 = well-clustered, -1 = wrong
Davies-Bouldin Index	Lower is better (compact & separated)	Good for comparing k-values
Inertia (within-cluster SSE)	Used in K-Means	Lower is better, but not scaled

Practical Evaluation Factors

Factor	Why It Matters
Interpretability	Do we understand how/why it makes predictions?
Training Time	Important for real-time or big data
Fairness	Does it treat all groups equally?
Generalization	Does it perform well on new data?
Explainability	Can we explain decisions to stakeholders?

Things to consider in Selecting a ML Algorithm

- If there's an algorithmic way instead of ML, use it!!! (ML is messy)
- Refer the literature!!!
- Try different ML algorithms (no single algorithm is the best)
- Check the dataset against the usage/strength of each algorithm (e.g. RNNs, ARIMA is good in time-series predictions)
- Be mindful of 'external factors' (e.g. seasonal effects, RL if you don't have data, Clustering if you have unlabeled data, etc.)
- Test your algorithm(s) with test data and select the best performing one for production (include the test results in your thesis/publications)
- No algorithm will be perfect! (There will be an error. The objective is to keep the error at an acceptable rate)

Popular Frameworks/Tools

- Scikit-learn - Python (Anaconda Python Distribution)
- R (R studio)
- Matlab/Octave (can export DLLs)
- Weka (Java based)
- Java OpenNLP/Python NLTK (Natural language processing + ML)
- Apache Spark (part of the Apache Hadoop platform)
- Google Tensorflow (Python library for Deep neural networks)
- Apache Keras (Python library of neural networks)
- Theano (Python library for Multicore processing of DNNs)
- Amazon AWS Services/Microsoft Azure ML (Cloud based ML)

Commonly used python libraries

NumPy

Matrix algebra

Pandas

Data Frames, Series

Matpotlib

Visualization

Summary

- AI is a vast discipline with many varying branches.
- AI attempts to give machine the ability to mimic human decision making/learning capabilities

Thank You

Jeewaka Perera

Jeewaka.p@sliit.lk