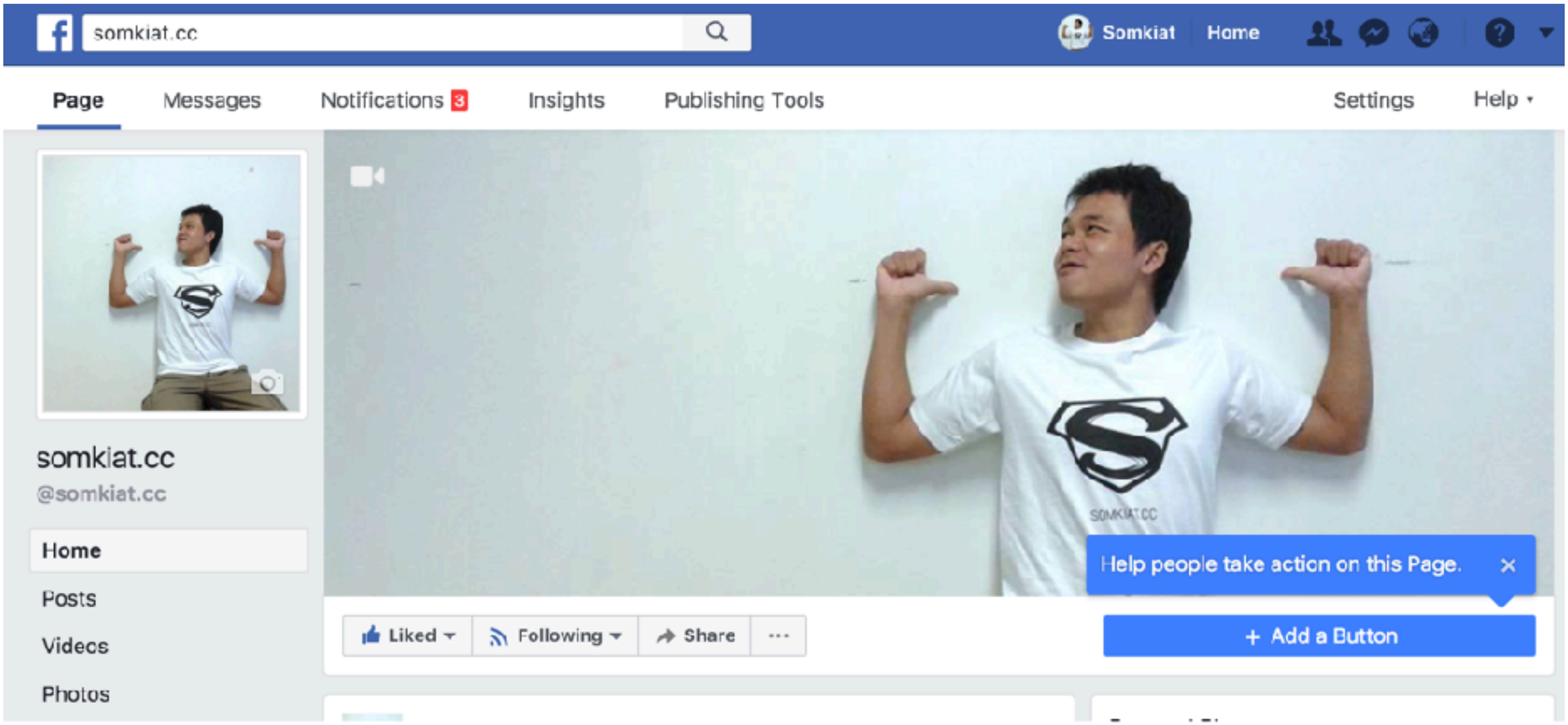
สยามชำนาญกิจ
SIAM CHAMNAN KIT

Basic Python workshop

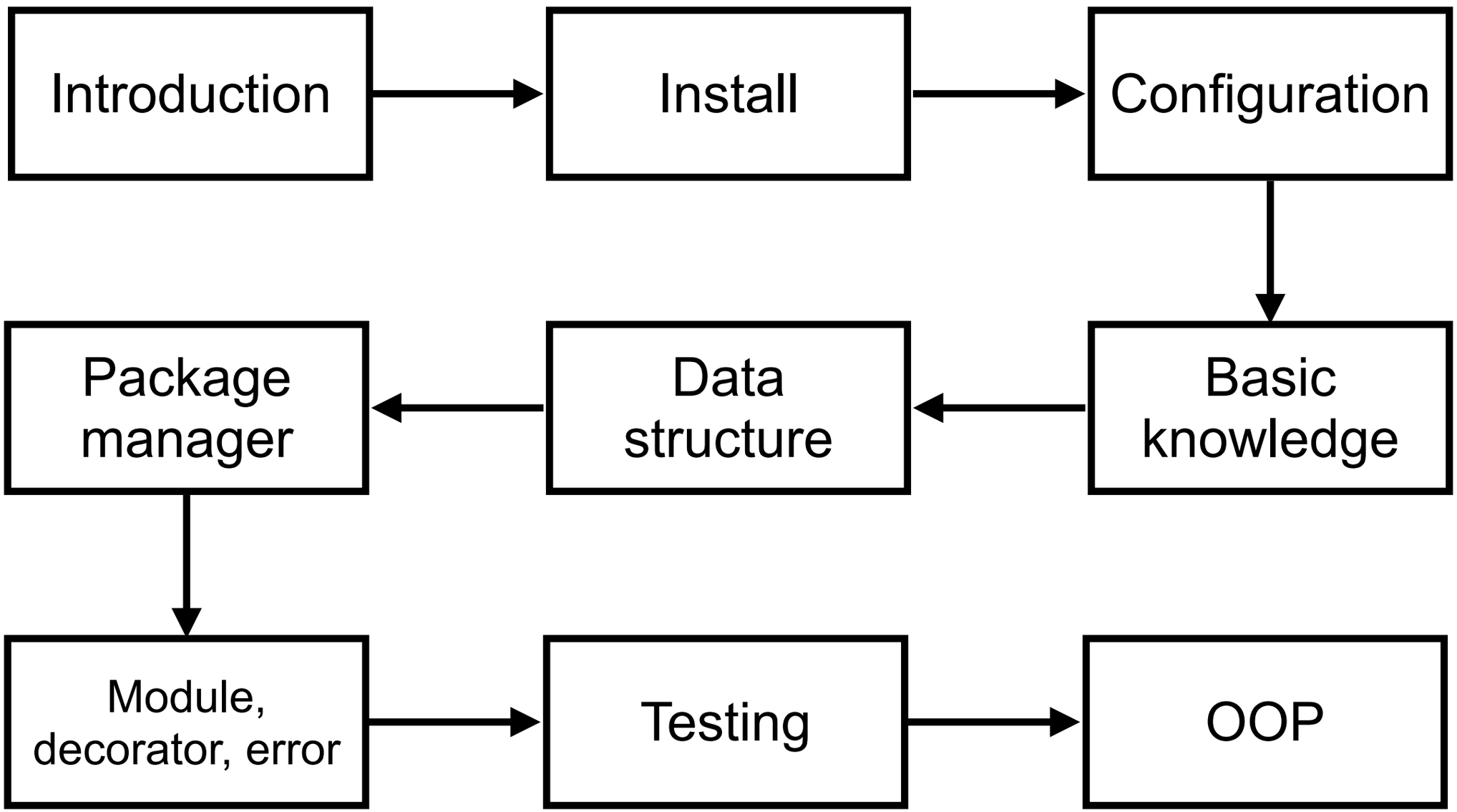


https://github.com/up1/course-python-workshop

Topics

Introduction to Python
Install and configuration
Basic of Python
RAG (Retrieval Augmented Generation)
RAG processes and techniques

Learning Path 1



Introduction to Programming

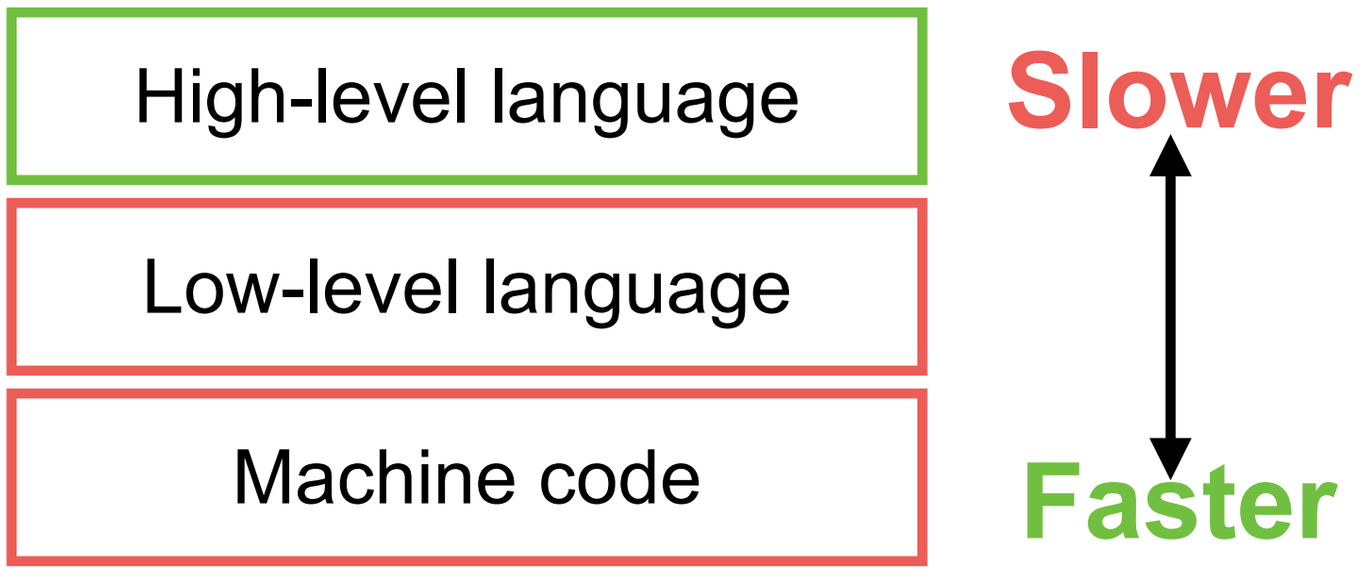
TIOBE Index May 2025

May 2025	May 2024	Change	Programming Language	Ratings	Change
1	1		Python	25.35%	+9.02%
2	3	↑	C++	9.94%	+0.41%
3	2	↓	C	9.71%	-0.27%
4	4		Java	9.31%	+0.62%
5	5		C#	4.22%	-2.27%
6	6		JavaScript	3.68%	+0.66%
7	8	↑	Go	2.70%	+1.10%
8	7	↓	Visual Basic	2.62%	+0.61%
9	11	↑	Delphi/Object Pascal	2.29%	+1.05%
10	9	↓	SQL	1.90%	+0.45%

https://www.tiobe.com/tiobe-index/

Basic of Python

High-level programming language
More human readable
General purpose
Open-source
Interpreter



Python use cases

Data science

Machine learning

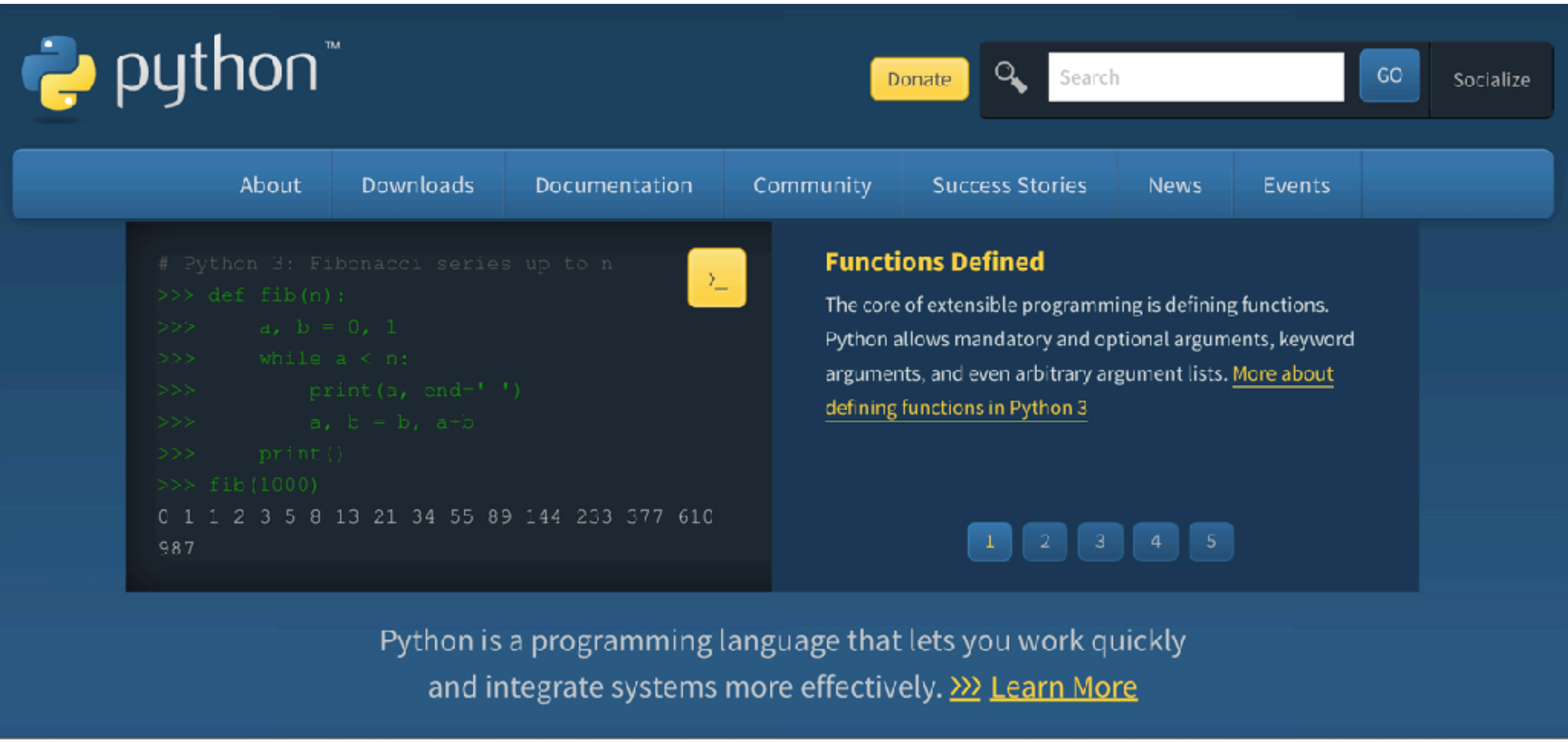
Web development

Desktop application

Task automation

Utility software

Install



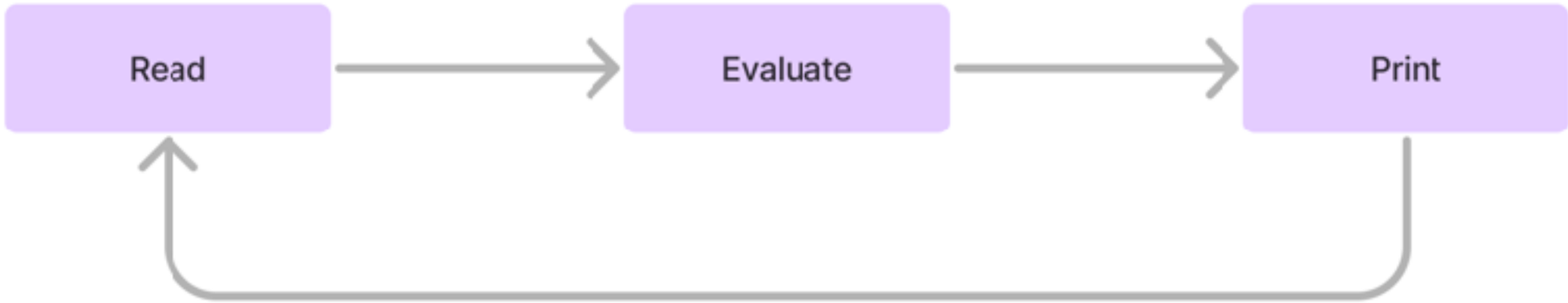
https://www.python.org/

Check your Python

$python -V
$pip -V

REPL mode

Read-Evaluate-Print-Loop
Interactive mode
Easy to code and fast feedback loop



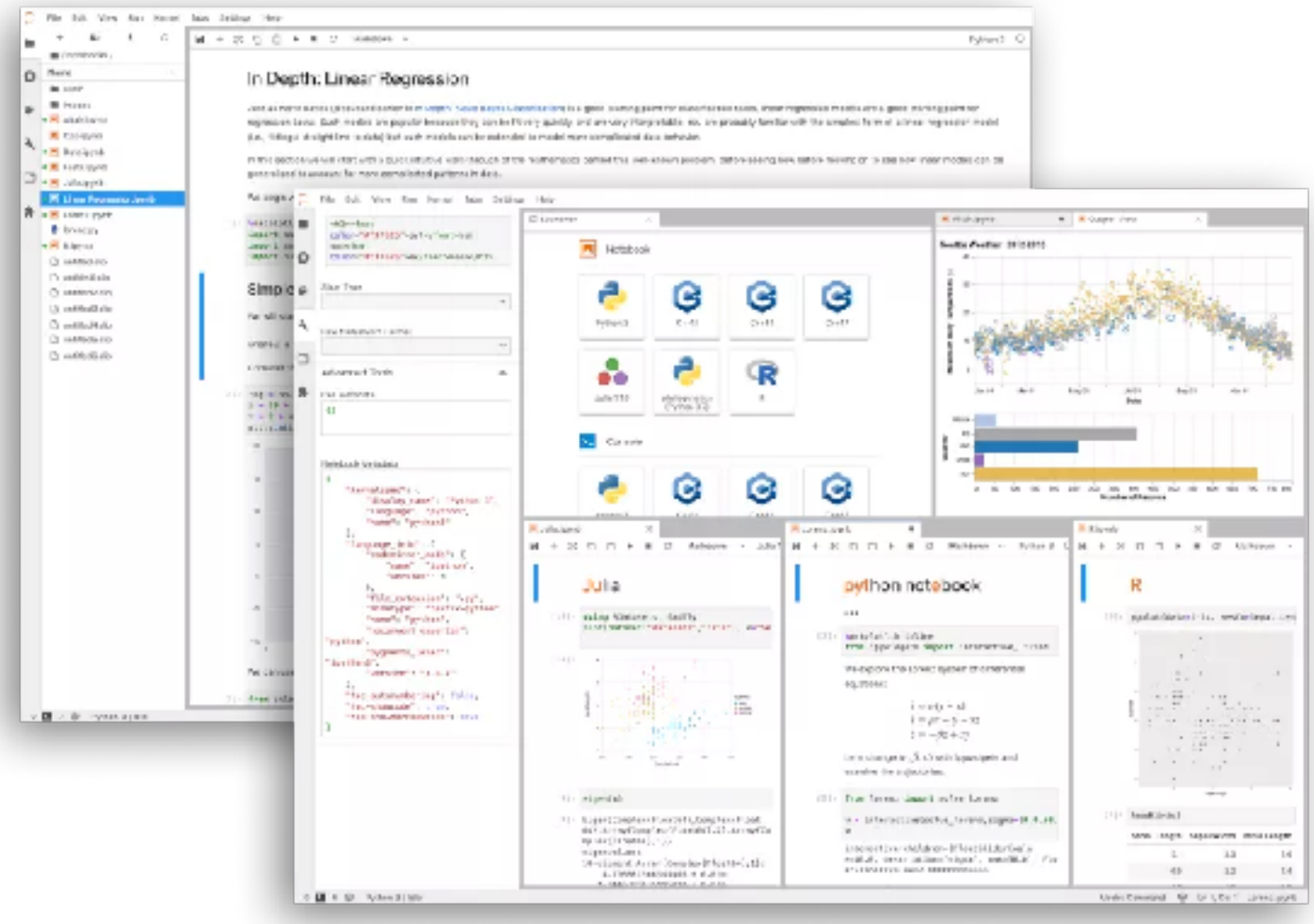
IDE

VSCode
PyCharm
Jupyter Notebook (web-based)

Visual Studio Code

Jupyter Notebook (web-based)

$pip install jupyterlab
$jupyter lab



https://jupyter.org/

First Program

Run as Interpreter process

$python hello.py

Source code
hello.py

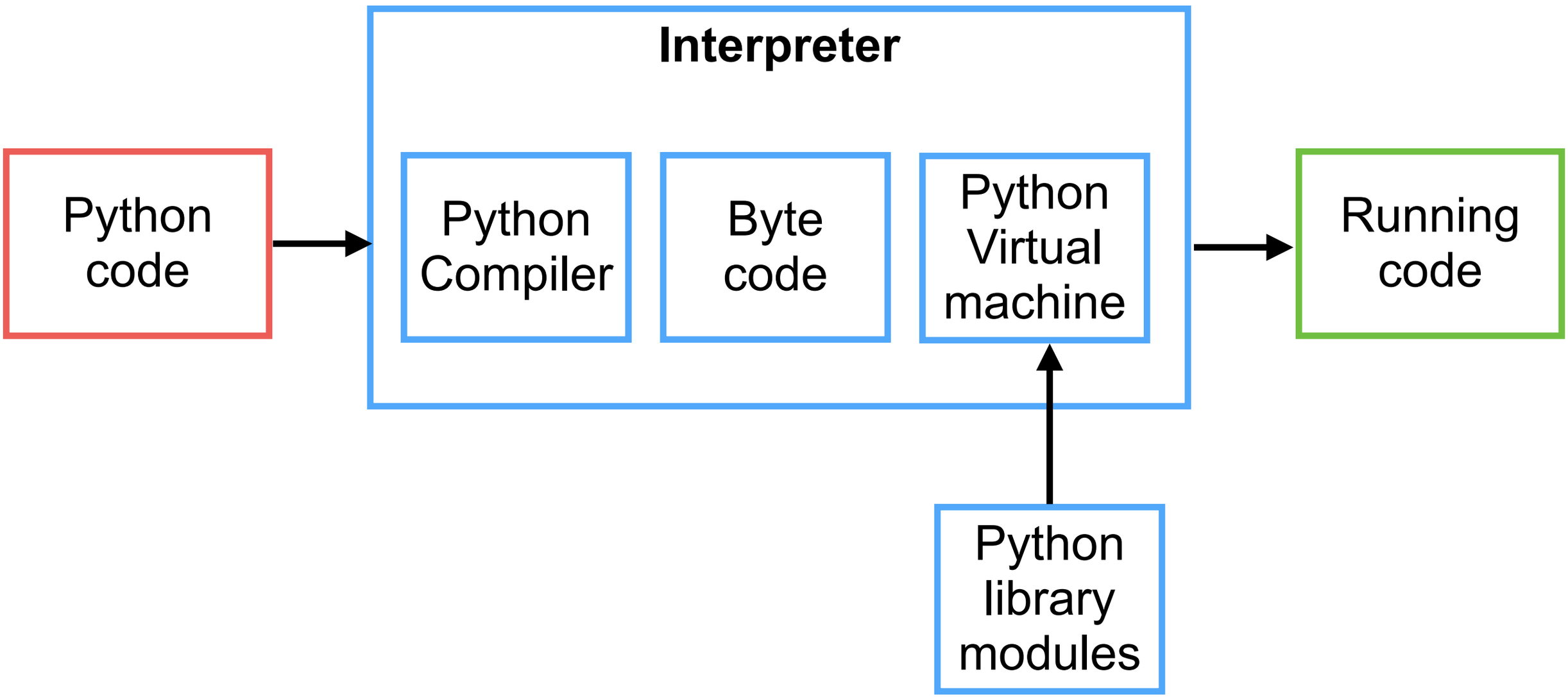
Virtual machine

Machine code

```
print("Hello, World!")
```

Interpreter

Convert code into machine code
when the program is run



Run as Script

$hello.py

```
#!/usr/bin/env python

print("Hello, World!")
```

Run as Module

$python -m hello

Compiled

Source code
hello.py

__pycached__
*.pyc

Machine code

```
print("Hello, World!")
```

Improve performance before calling/import from others

Better structure of program

```python
def say_hello():
    """Prints a greeting message."""
    print("Hello, World!")


if __name__ == "__main__":
    say_hello()
```

Object Oriented Programming

```python
class HelloWorld:
    def __init__(self, name):
        self.name = name

    def say(self):
        return f"Hello, {self.name}!"

if __name__ == "__main__":
    hello = HelloWorld("World")
    print(hello.say())
```

Whitespace !!

```
class HelloWorld:
⟷def __init__(self, name):
    ⟷self.name = name

⟷def say(self):
    ⟷return f"Hello, {self.name}!"

if __name__ == "__main__":
⟷hello = HelloWorld("World")
⟷print(hello.say())
```

Zen of Python

https://en.wikipedia.org/wiki/Zen_of_Python

Zen of Python

$python
>>> import this

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

https://peps.python.org/pep-0020/

Variables and Data Types

Variable Name Rules

Start with letter or underscore character

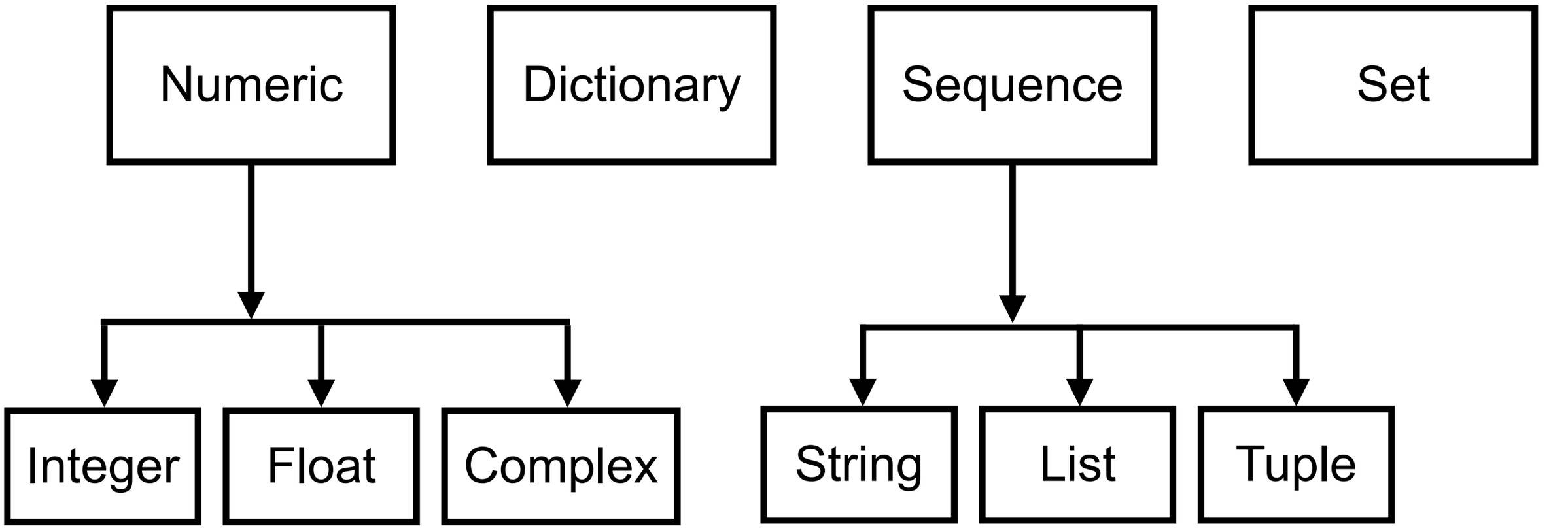
Can’t start with number

Only contain (A-z, 0-9 and _)

Case-sensitive

Python developer use **snack_case**

Data Types

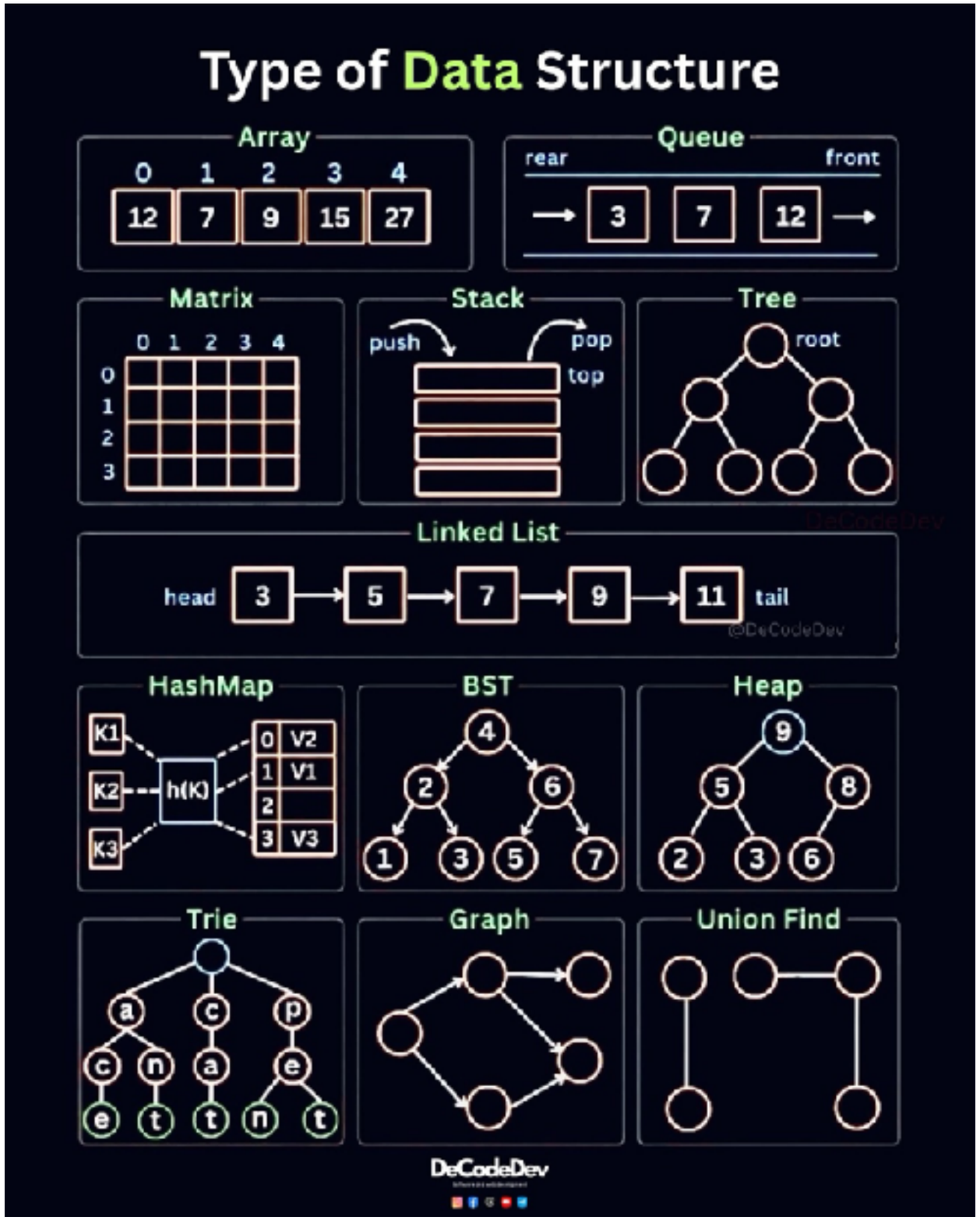


Data Structure in Python

List | Dictionary | Tuple | Set

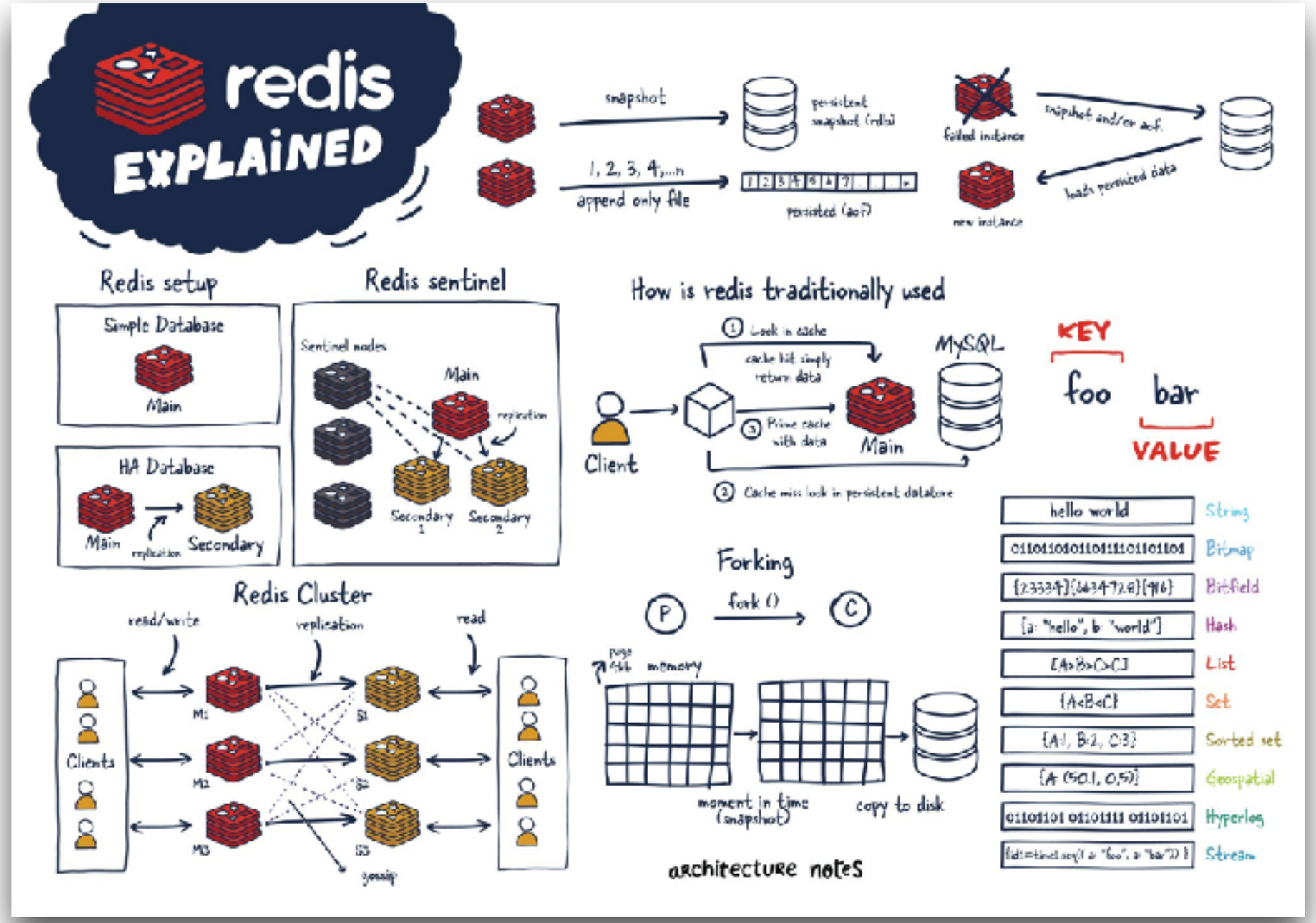
Queue | Linked list | Stack | Tree

Graph | HashMap



https://www.facebook.com/photo/?fbid=1222345108762337 89&set=a.1221211302423378 9

Data Structure Database !!



https://architecturenotes.co/p/redis

Control flows

If	For	Range()	While
Break	Continue	Pass	For-else
Match			

https://docs.python.org/3/tutorial/controlflow.html

Basic build-in functions

type()
dir()
vars()
help()
input()

Working with String

Strings are immutable (can’t change)

Single quote vs Double quote !!

```python
data = "Hello, World!"

print(data) # Print the string
print(len(data)) # Length of the string
print(type(data)) # Type of data is str (string)
print(data[0])  # First character
print(data[-1])  # Last character
print(data[0:5])  # First five characters
print(data[7:])  # From the 8th character to the end
print(data[7:12])  # Characters from index 7 to 11
print(data[7:12:2])  # Characters from index 7 to 11, stepping by 2
print(data[::-1])  # Reverse the string

data[0]= "H"  # This will raise an error because strings are immutable
```

String formatting

```python
a = "Hello"
b = “World"

c = f"{a}, {b}!"  # Using f-string for formatting
print(c)  # Output: Hello, World!

d = "{} {}".format(a, b)  # Using format method for formatting
print(d)  # Output: Hello World!

e = "%s %s!" % (a, b)  # Using old-style formatting
print(e)  # Output: Hello World!

f = "{0}, {1}!".format(a, b)  # Using format method with positional arguments
print(f)  # Output: Hello, World!

g = "{name}, {place}!".format(name=a, place=b)  # Using format method with named arguments
print(g)  # Output: Hello, World!
h = "Hello, {name}!".format(name=b)  # Using format method with a single named argument
print(h)  # Output: Hello, World!
i = "Hello, {0}!".format(b)  # Using format method with a positional argument
print(i)  # Output: Hello, World!
j = "Hello, {name}!".format(name="Alice")  # Using format method with a named argument
print(j)  # Output: Hello, Alice!

k = " ".join([a, b])  # Joining strings with a space
print(k)  # Output: Hello World
```

List vs Tuple

Lists are mutable
Tuples are immutable

[1,2] (1,2)

List’s operation

```python
my_list = [1,2,3]

my_list.append(4)  # Adding an element to the end of the list
my_list.insert(0, 0)  # Inserting an element at the beginning of the list
my_list.remove(2)  # Removing the first occurrence of an element
my_list.pop()  # Removing the last element of the list
my_list.sort()  # Sorting the list in ascending order
my_list.extend([5, 6])  # Extending the list with another list
my_list.reverse()  # Reversing the order of the list
```

Multiply by 2 for all elements

```python
my_list = [1,2,3]


for i in my_list:
    i *= 2
```

```python
my_list = [1,2,3]


for i in range(len(my_list)):
    my_list[i] *= 2
```

```python
my_list = [1,2,3]


# List comprehension to multiply each element by 2
my_list = [x * 2 for x in my_list]
print(my_list)
```

Readability code !!

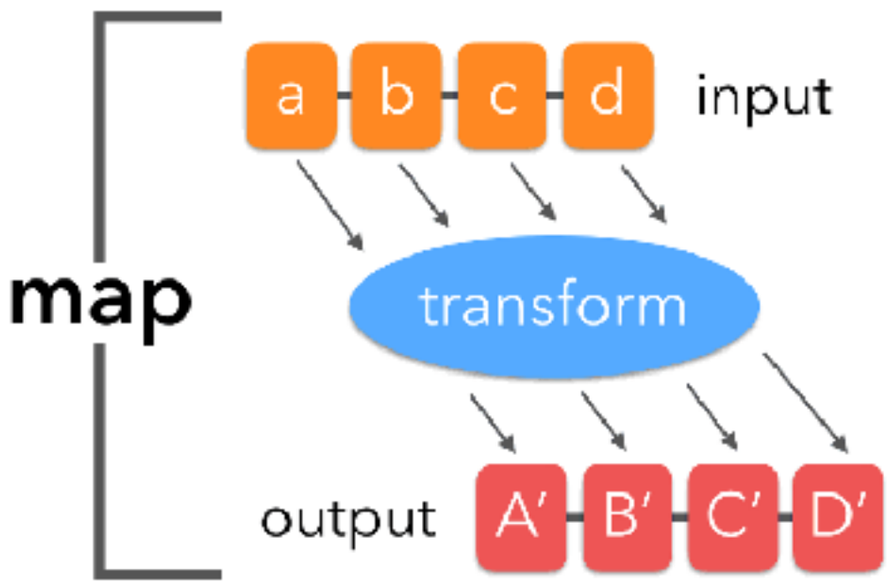
Looping in Python !!

For loop

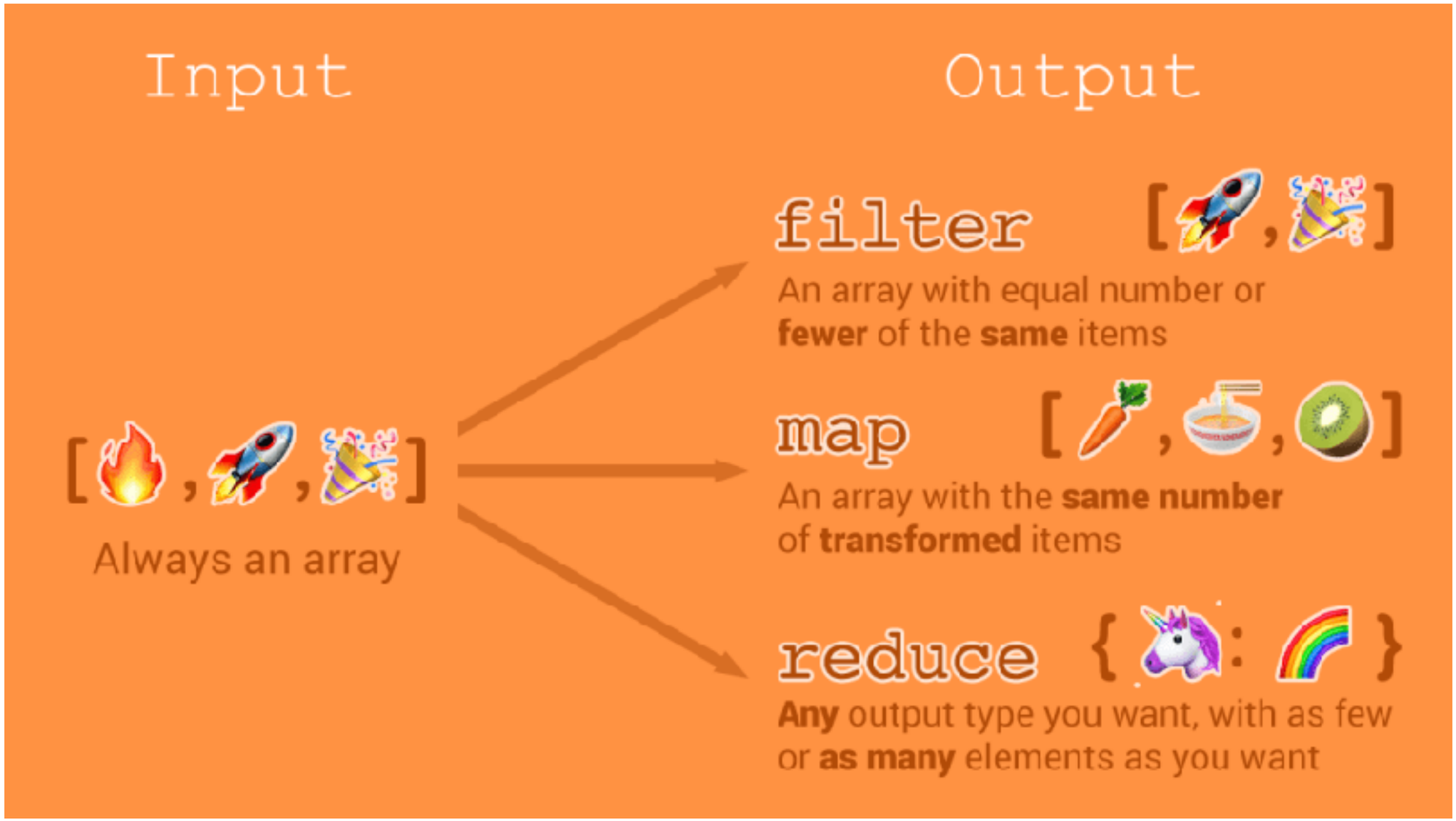
While loop

Functional programming !!

```python
def multiply_by_two(x):
    return x * 2

def is_even(x):
    return x % 2 == 0

if __name__ == "__main__":
    my_list = [1,2,3]

    # Filter and Map with Functions
    result_list = filter(is_even, my_list)
    print(list(result_list))

    # Map with Functions
    result_list = map(multiply_by_two, my_list)
    print(list(result_list))
```

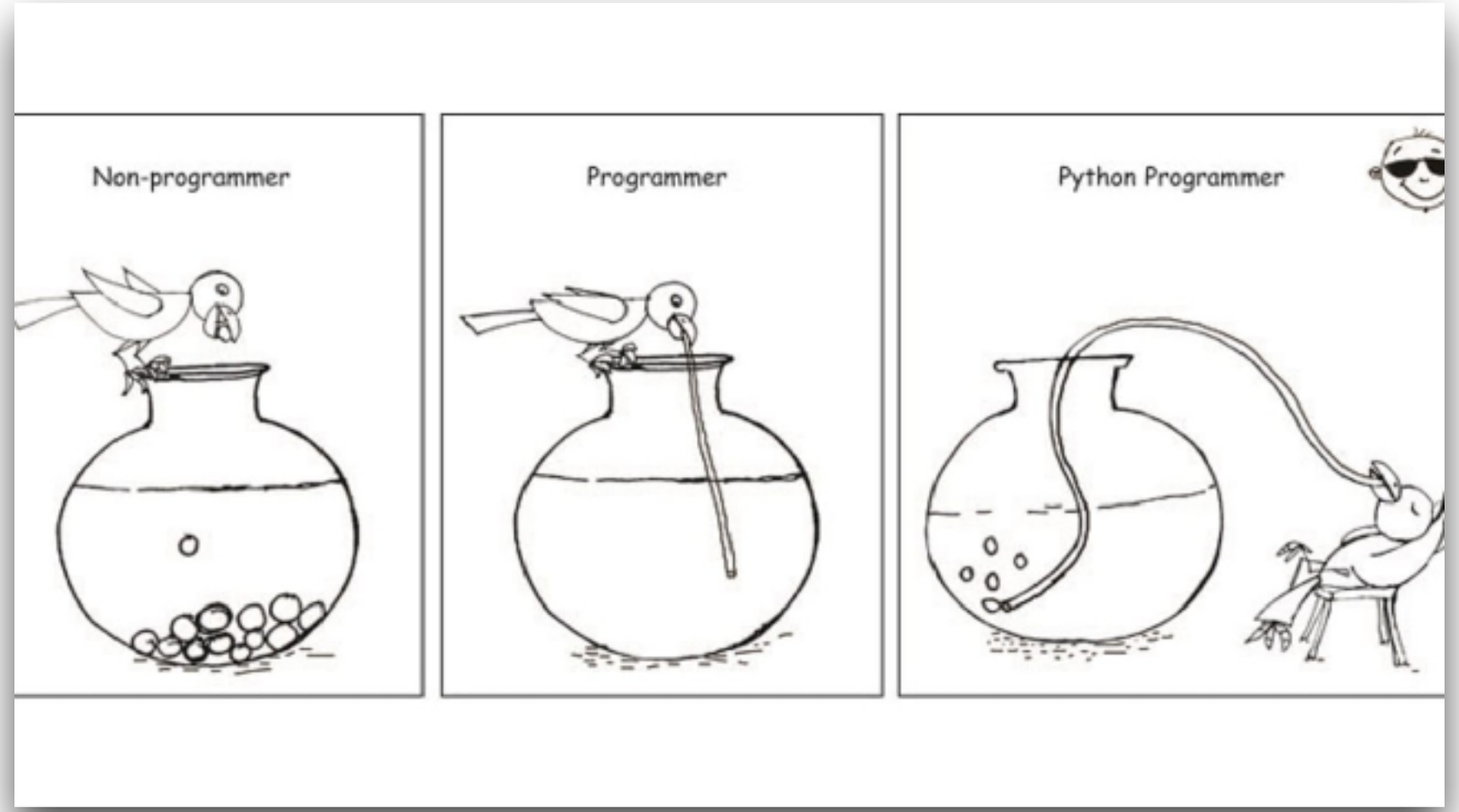


Filter, Map and Reduce



https://www.learnpython.org/en/Map%2C_Filter%2C_Reduce

Python programmer



Tuple

Tuples are immutable
Cannot change !!

Convert List to Tuple ?

Using tuple()
Using loop inside tuple()
Unpack list

```python
data_list = ["Hello", "World", 1, 2, 3.14, True]

# Use tuple()
data_tuple = tuple(data_list)

# Use for loop in tuple
data_tuple = tuple(x for x in data_list)

# Use unpacking list to tuple
data_tuple = (*data_list,)


print(type(data_tuple))
print(data_tuple)
```

Set

Sets are mutable

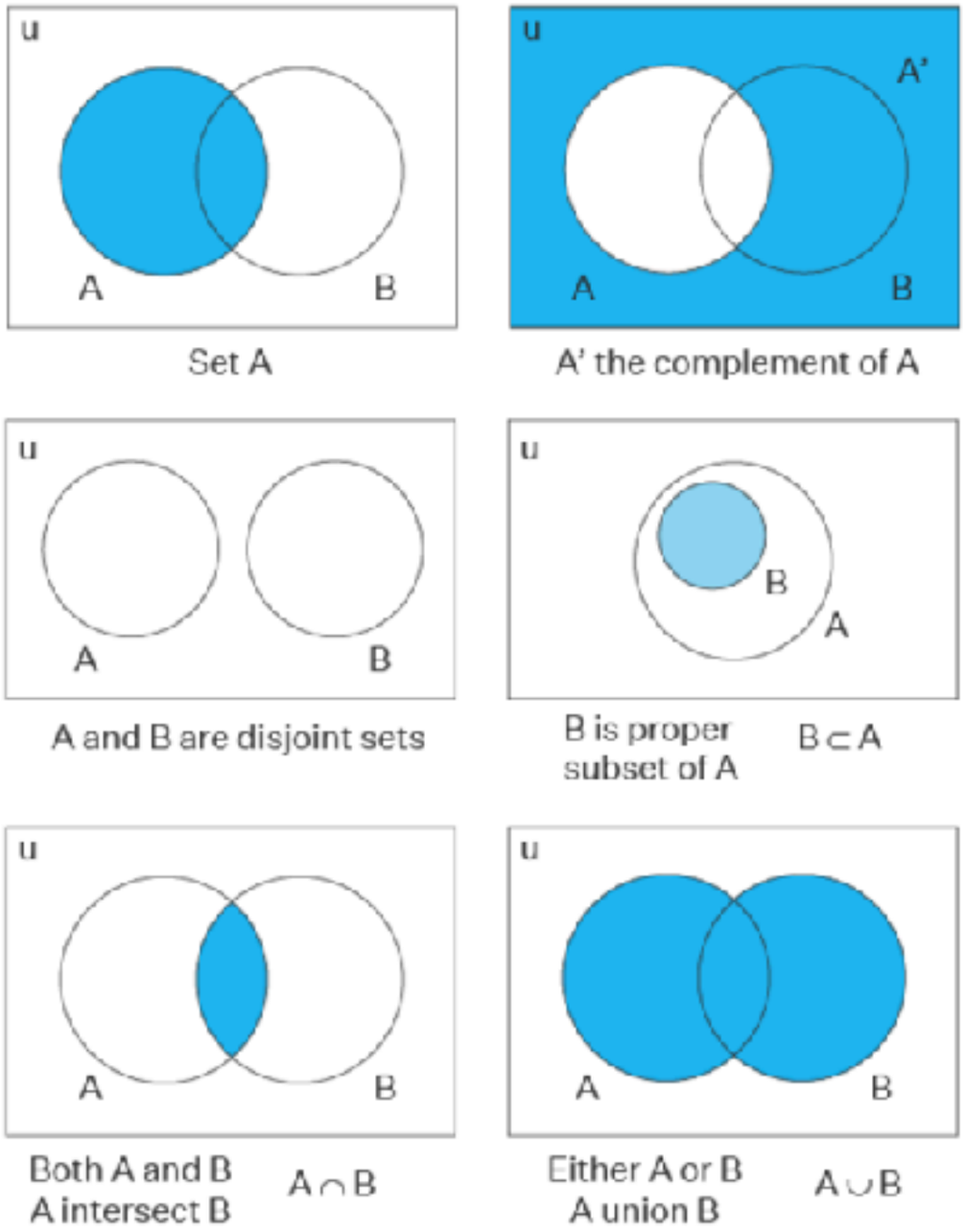
No duplicate elements (unique)

Not maintain the order of the data insertion

Basic of Set

```python
data_list = [1,1,1,2,2,3,4,5,6,7,8,9,10]
data_set = set(data_list)
print(type(data_set))  # <class 'set'>
print(data_set) # {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

# Add elements to the set
data_set.add(11)  # Adding a single element
data_set.update([12, 13])  # Adding multiple elements
print(data_set)  # {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}

# Remove elements from the set
data_set.remove(1)  # Removing a specific element
data_set.discard(2)  # Discarding an element (no error if not found)
data_set.pop()  # Removing an arbitrary element
print(data_set)  # {4, 5, 6, 7, 8, 9, 10, 11, 12, 13}
```

Set's operations



https://realpython.com/python-sets/

Set's operations

```python
# Check membership
print(3 in data_set)  # True
print(14 in data_set)  # False

# Set operations
data_set2 = {5, 6, 7, 8, 9, 10, 11, 12}

# Union
union_set = data_set.union(data_set2)

# Intersection
intersection_set = data_set.intersection(data_set2)

# Difference
difference_set = data_set.difference(data_set2)

# Symmetric Difference
symmetric_difference_set = data_set.symmetric_difference(data_set2)

# Set comprehension
data_set_comp = {x * 2 for x in data_set if x % 2 == 0}
```

List vs Tuple vs Set

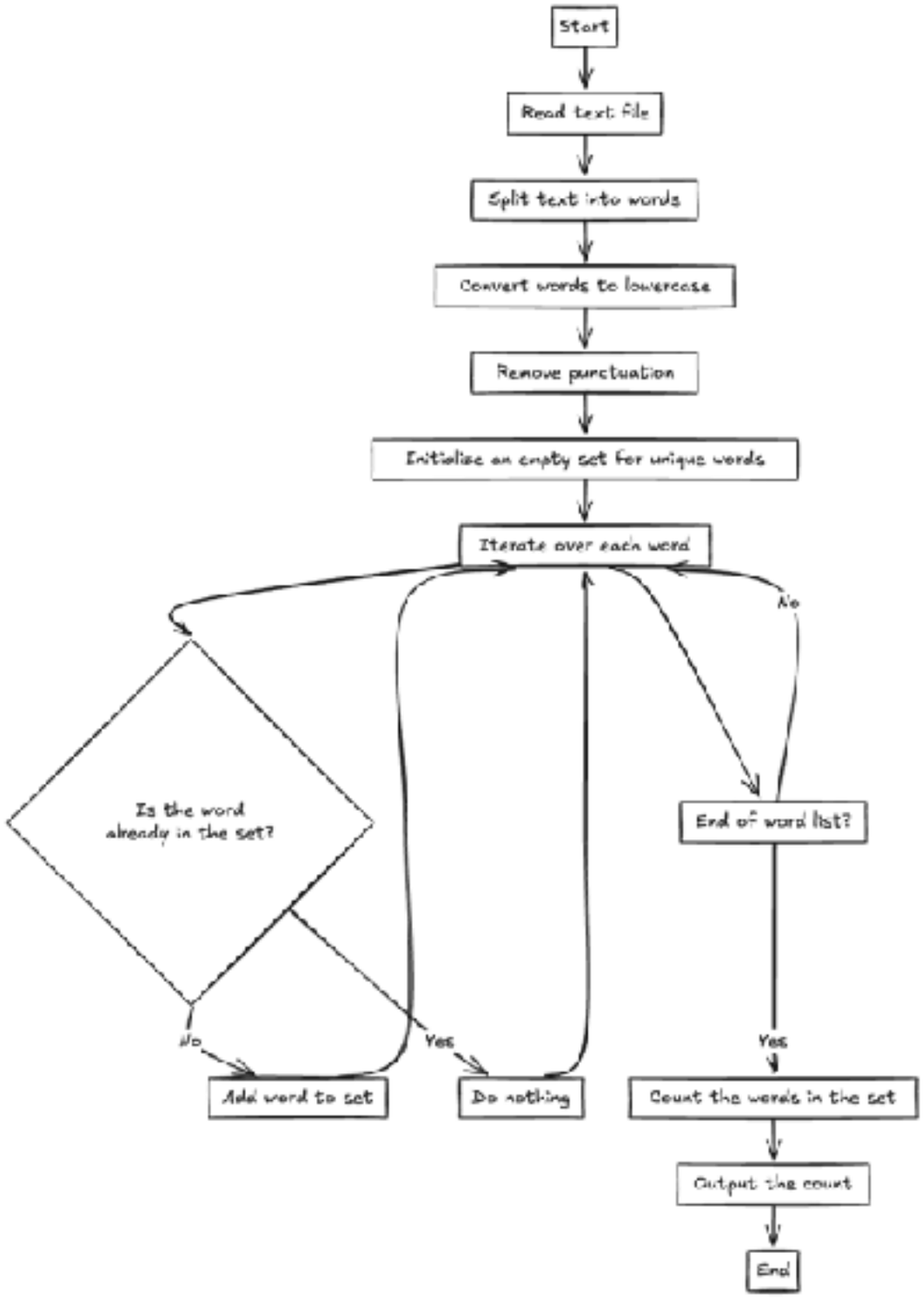
	Mutable	Ordered	Indexing / Slicing	Duplicate Elements
List	✓	✓	✓	✓
Tuple	✗	✓	✓	✓
Set	✓	✗	✗	✗

https://towardsdatascience.com/15-examples-to-master-python-lists-vs-sets-vs-tuples-d4ffb291cf07/

Workshop

Count unique words from text file

Workshop (Flow chart)



Error handling !!

Reliable of program to read data from file

```python
def main():
    file_path = 'data/message.txt'
    try:
        input_text = read_file(file_path)
        unique_word_count = count_unique_words(input_text)
        print(f"Number of unique words: {unique_word_count}")
    except FileNotFoundError:
        print(f"The file {file_path} does not exist.")
    except Exception as e:
        print(f"An error occurred: {e}")
```

https://docs.python.org/3/tutorial/errors.html

Working with logging

Keep stages of application

Unstructured	Structured

https://docs.python.org/3/howto/logging-cookbook.html

Logging !!

```
INFO - User John Doe with ID 123 made a purchase of $200 on a Visa
card with last four digits of 4312 on 2025-05-25.
```

```
{
"severity": "INFO",
"timestamp": “2025-05-25T12:34:56Z",
"userId": 123,
"userName": "John Doe",
"action": "purchase",
"card_type": "Visa",
"last_four_digits": 1414,
"amount": 200
}
```

https://newrelic.com/blog/how-to-relic/python-structured-logging

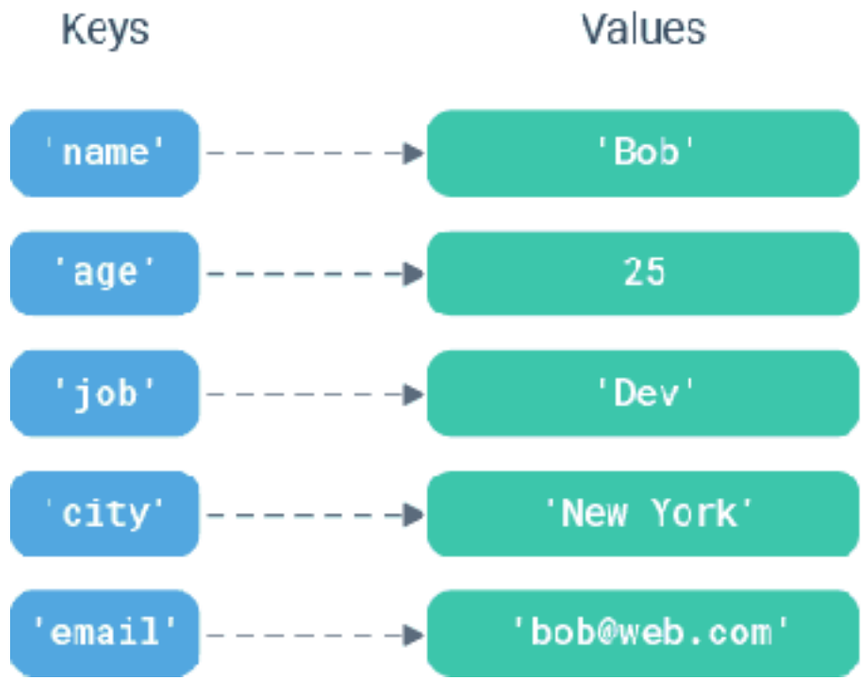
Structured log

```python
import structlog
import logging

# Configure structlog to output structured logs in JSON format
structlog.configure(
    processors=[
        structlog.stdlib.filter_by_level,
        structlog.processors.TimeStamper(fmt="iso"),
        structlog.processors.JSONRenderer()
    ],
    context_class=dict, logger_factory=structlog.stdlib.LoggerFactory()
    )

# Configure the standard logging module to use structlog
logging.basicConfig(
    format="%(message)s",
    level=logging.INFO,
    handlers=[logging.StreamHandler()]
)

# Get a logger
logger = structlog.get_logger()

# Now we can log structured messages!
logger.info("User logged in", user_id="1234", ip="192.0.2.0")
```

https://www.structlog.org/

Dictionary

Ordered collections of unique values (key-value)
Dictionaries are mutable



Basic of Dictionary

```python
data_dict = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
}
# Accessing values
print(data_dict["name"])  # Alice
print(data_dict.get("age"))  # 30

# Adding a new key-value pair
data_dict["country"] = "USA"

# Updating an existing key-value pair
data_dict["age"] = 31

# Removing a key-value pair
data_dict.pop("city")
```

Basic of Dictionary

```python
data_dict = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
}
# Iterating through keys and values
for key, value in data_dict.items():
    print(f"{key}: {value}")

# Checking if a key exists
print("name" in data_dict)  # True
print("city" in data_dict)  # False

# Merging dictionaries
data_dict2 = {"state": "NY", "zip": "10001"}
data_dict.update(data_dict2)
```

Workshop

Merge lists into a list

Names = [A, B, C]

Ages = [10, 20, 30]

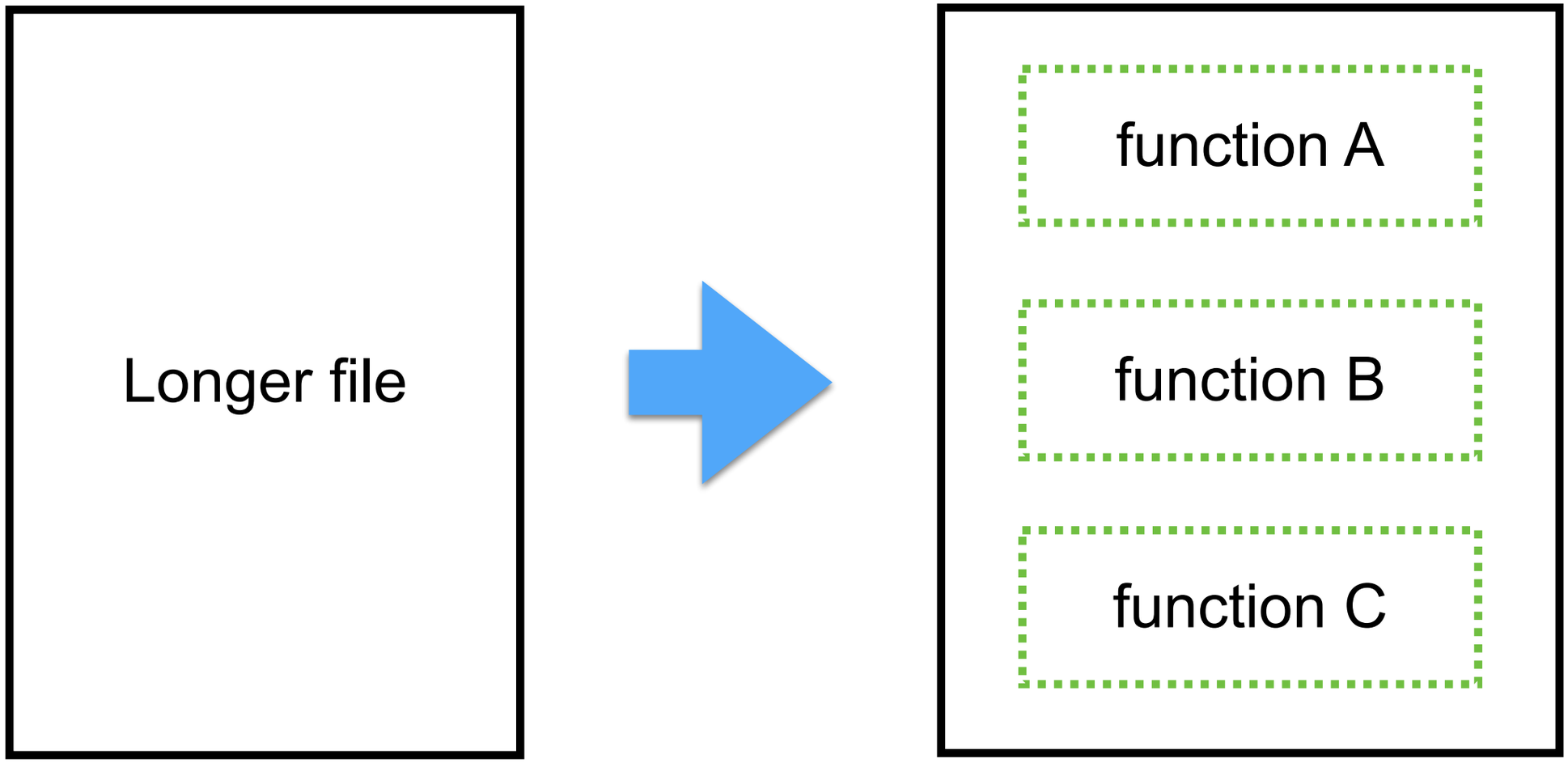
Cities = [X, Y, Z]

```
[
  {‘age': 10, 'city': 'X', 'name': 'A'},
  {'age': 20, 'city': 'Y', 'name': 'B'},
  {‘age': 30, 'city': 'Z', 'name': ‘C'},
]
```

Functions in Python

Functions

Split long script to small groups
Manageable code



Build-in functions

Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

Built-in Functions

A	E	L	R
abs() aiter() all() anext() any() ascii() **B** bin() bool() breakpoint() bytearray() bytes()	enumerate() eval() exec() **F** filter() float() format() frozenset() **G** getattr() globals()	len() list() locals() **M** map() max() memoryview() min() **N** next() **O**	range() repr() reversed() round() **S** set() setattr() slice() sorted() staticmethod() str() sum() super()

https://docs.python.org/3/library/functions.html

Working with function (1)

```python
# Using the Force formula F=ma

while True:
  mass = int(input("Enter the mass value: "))
  if mass > 0:
    break

while True:
  acceleration = int(input("Enter the acceleration: "))
  if acceleration > 0:
    break

print("The Force is", mass * acceleration)
```

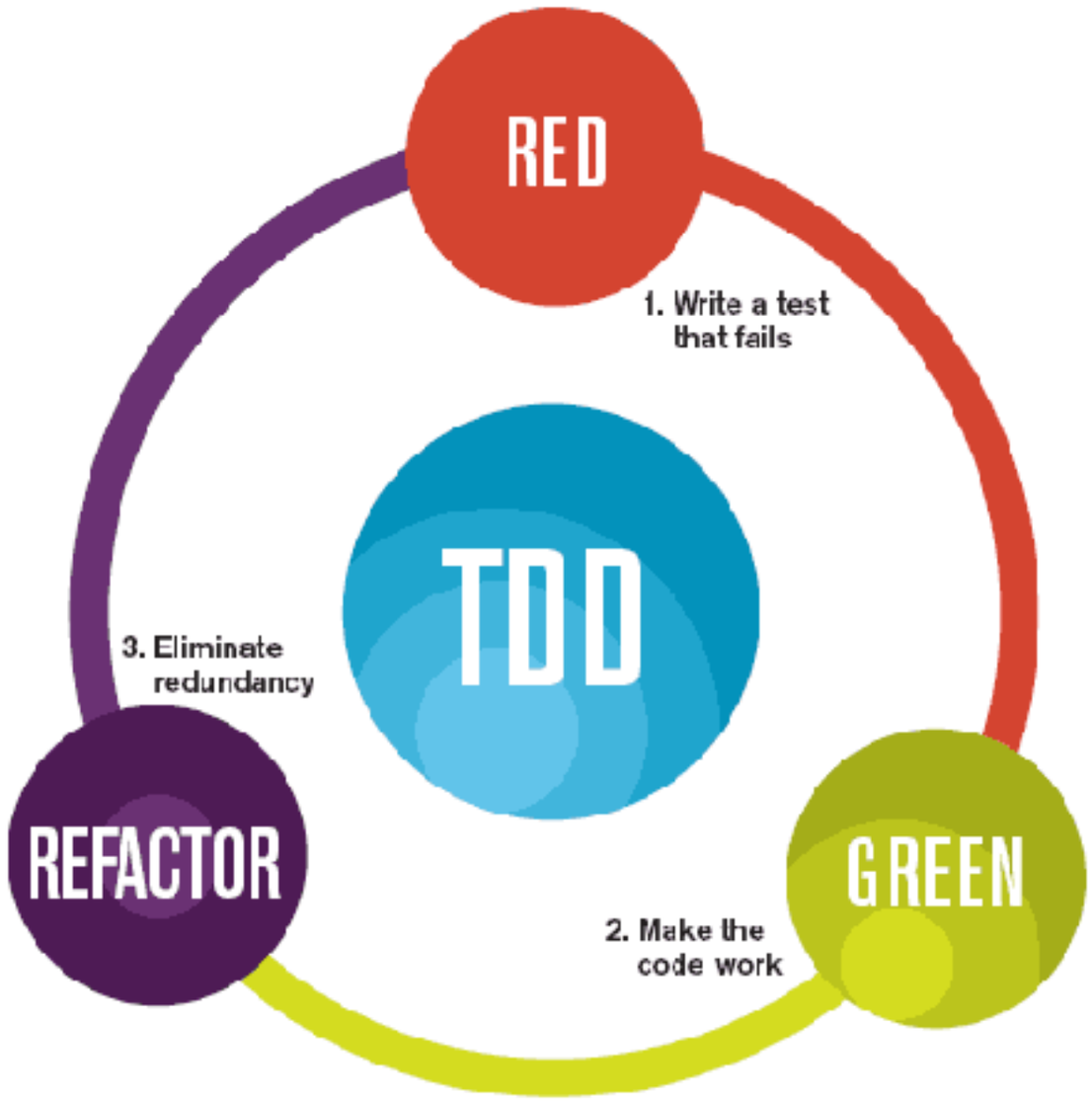
https://www.freecodecamp.org/news/best-practices-for-refactoring-code/

Working with function (2)

```python
# Using the Force formula F=ma

def get_positive_integer(prompt):
    while True:
        try:
            value = int(input(prompt))
            if value > 0:
                return value
        except ValueError:
            print("Please enter a valid positive integer.")

def calculate_force(mass, acceleration):
    return mass * acceleration

if __name__ == "__main__":
    mass = get_positive_integer("Enter the mass value: ")
    acceleration = get_positive_integer("Enter the acceleration: ")

    print("The Force is", calculate_force(mass, acceleration))
```

https://www.freecodecamp.org/news/best-practices-for-refactoring-code/

Trust in your code ?



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Write your first test case

unittest

pytest

https://docs.pytest.org/en/stable/

Run your test

$pytest

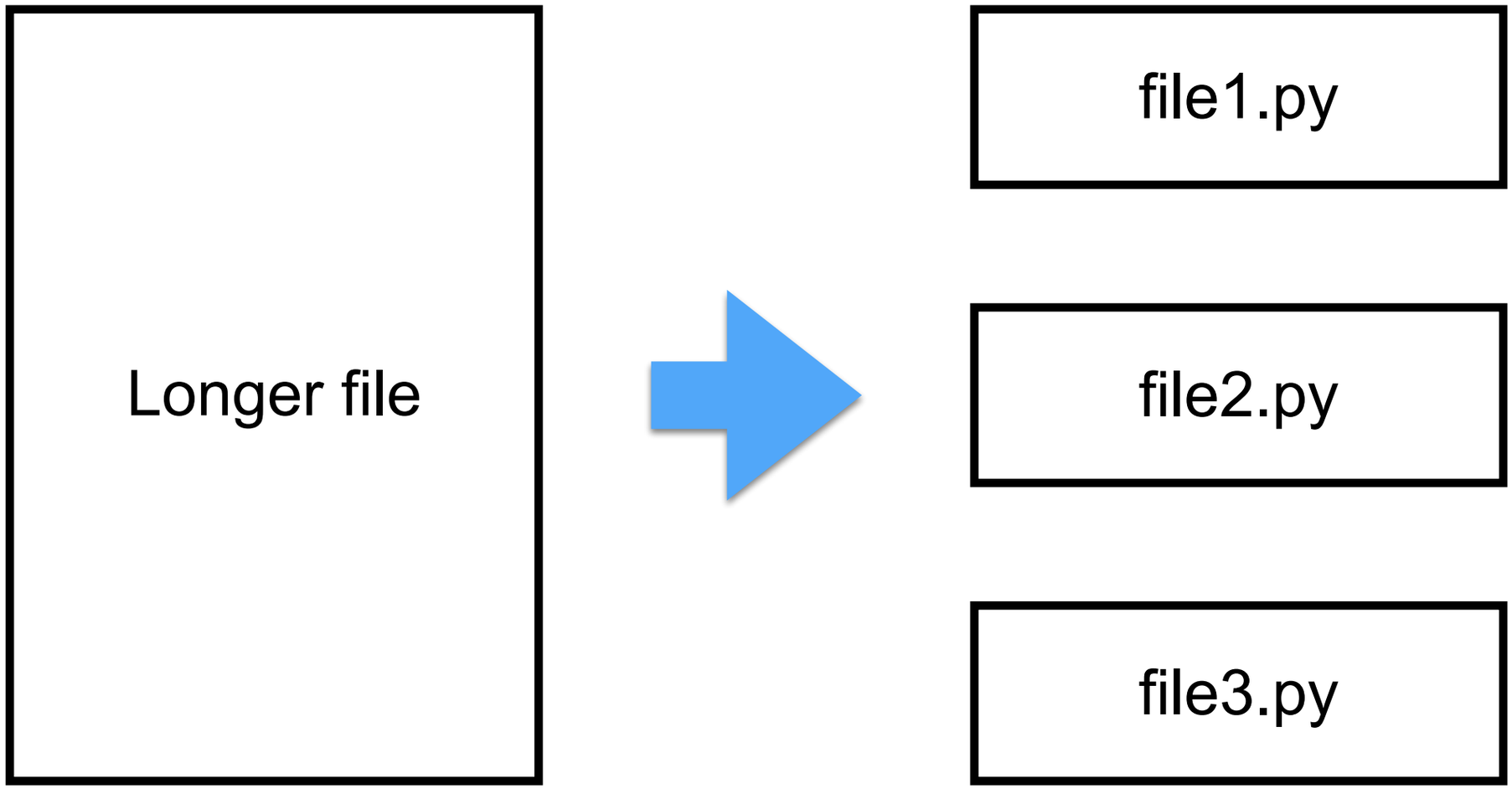
```python
from after import calculate_force

def test_with_valid_inputs_calculate_force():
    # Test with valid inputs
    result = calculate_force(2, 3)
    assert result == 6, f"Expected 6, but got {result}"

    # Test with zero
    result = calculate_force(0, 3)
    assert result == 0, f"Expected 0, but got {result}"

    # Test with negative numbers
    result = calculate_force(-2, -3)
    assert result == 6, f"Expected 6, but got {result}"

    # Test with mixed positive and negative numbers
    result = calculate_force(2, -3)
    assert result == -6, f"Expected -6, but got {result}"
```

Modules in Python

https://docs.python.org/3/tutorial/modules.html

Modules

Split long script to small pieces of files



Example of Modules

Split long script to small pieces of files

hello.py

```python
def say_hi(message: str) -> str:
    return f"Hello, {message}!"
```

use

main.py

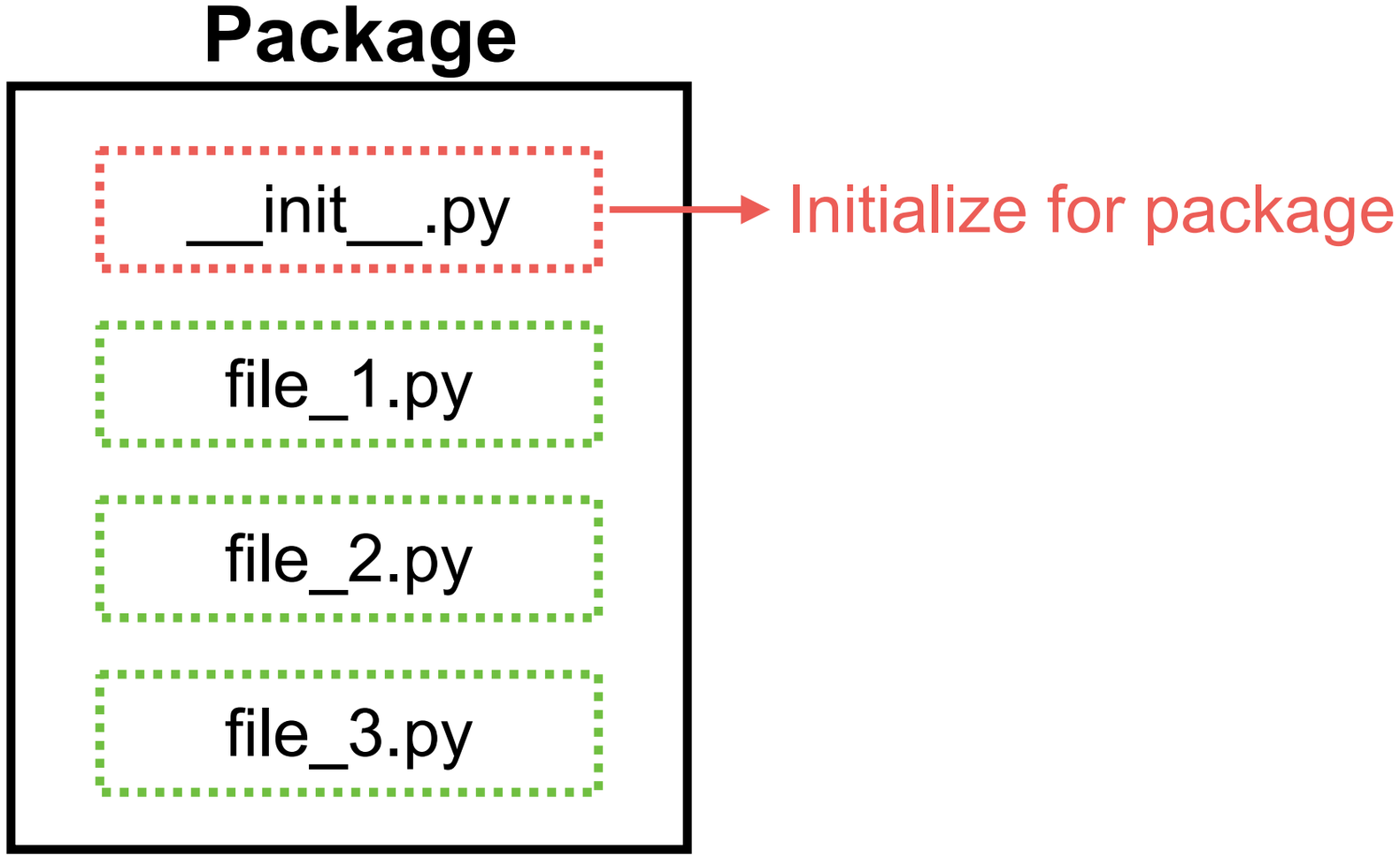
```python
import hello

if __name__ == "__main__":
    result = hello.say_hi("Somkiat")
    print(f'Result: {result}')
```

Modules vs Packages

https://docs.python.org/3/tutorial/modules.html

Packages

Collection of modules that provide a set of related functionalities



https://docs.python.org/3/tutorial/modules.html#packages

Demo package

demo_package

demo.py

```
def say_hi():
    return "Hello from demo_package!"
```

__init__.py

```
__all__ = ["demo"]
```

main.py

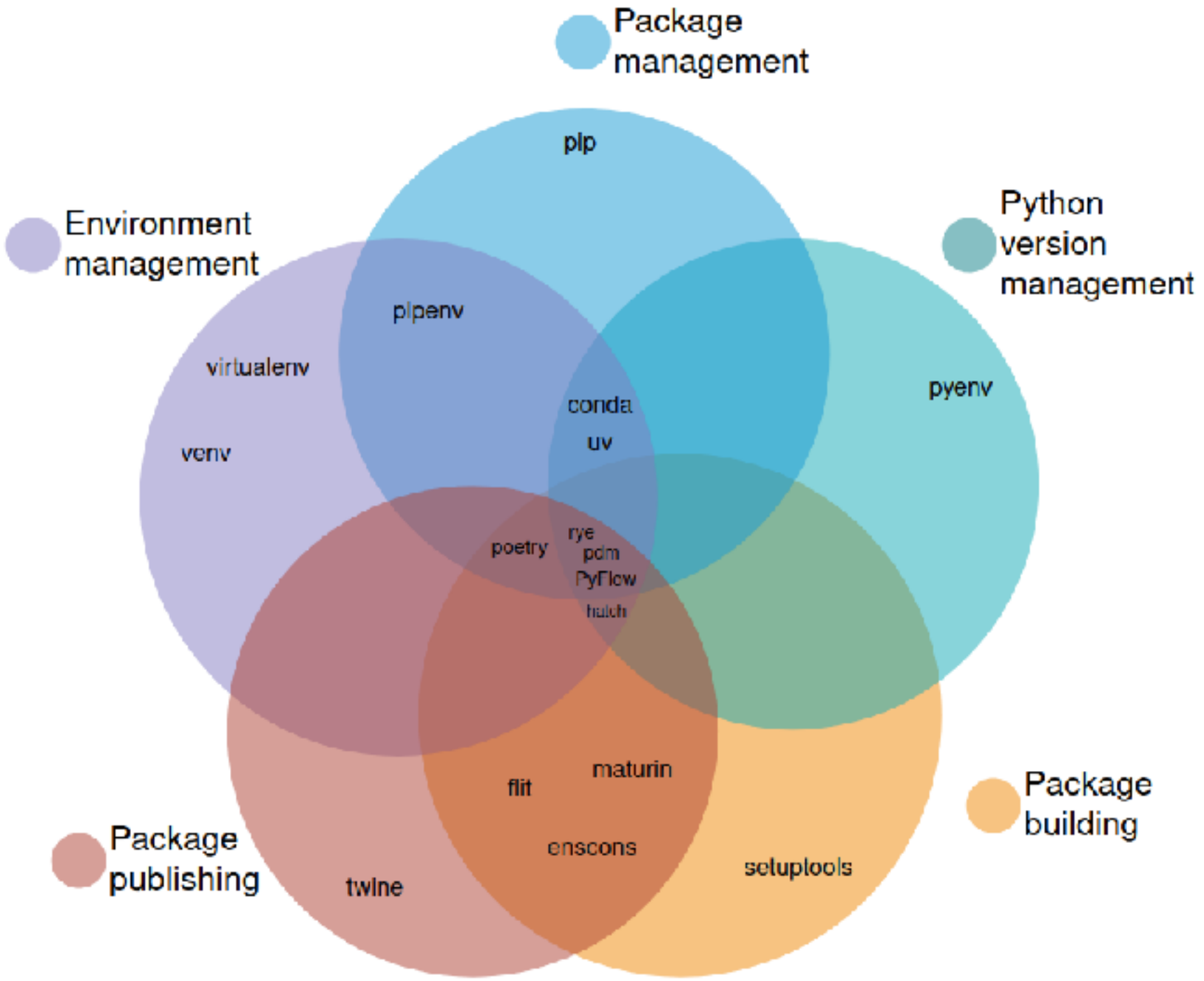
```
import demo_package

if __name__ == "__main__":
    print(dir(demo_package))
    print(demo_package.__file__)
```

```
from demo_package import *

if __name__ == "__main__":
    res = demo.say_hi()
    print(res)
```

https://sentry.io/answers/what-is-init-py-for-in-python/

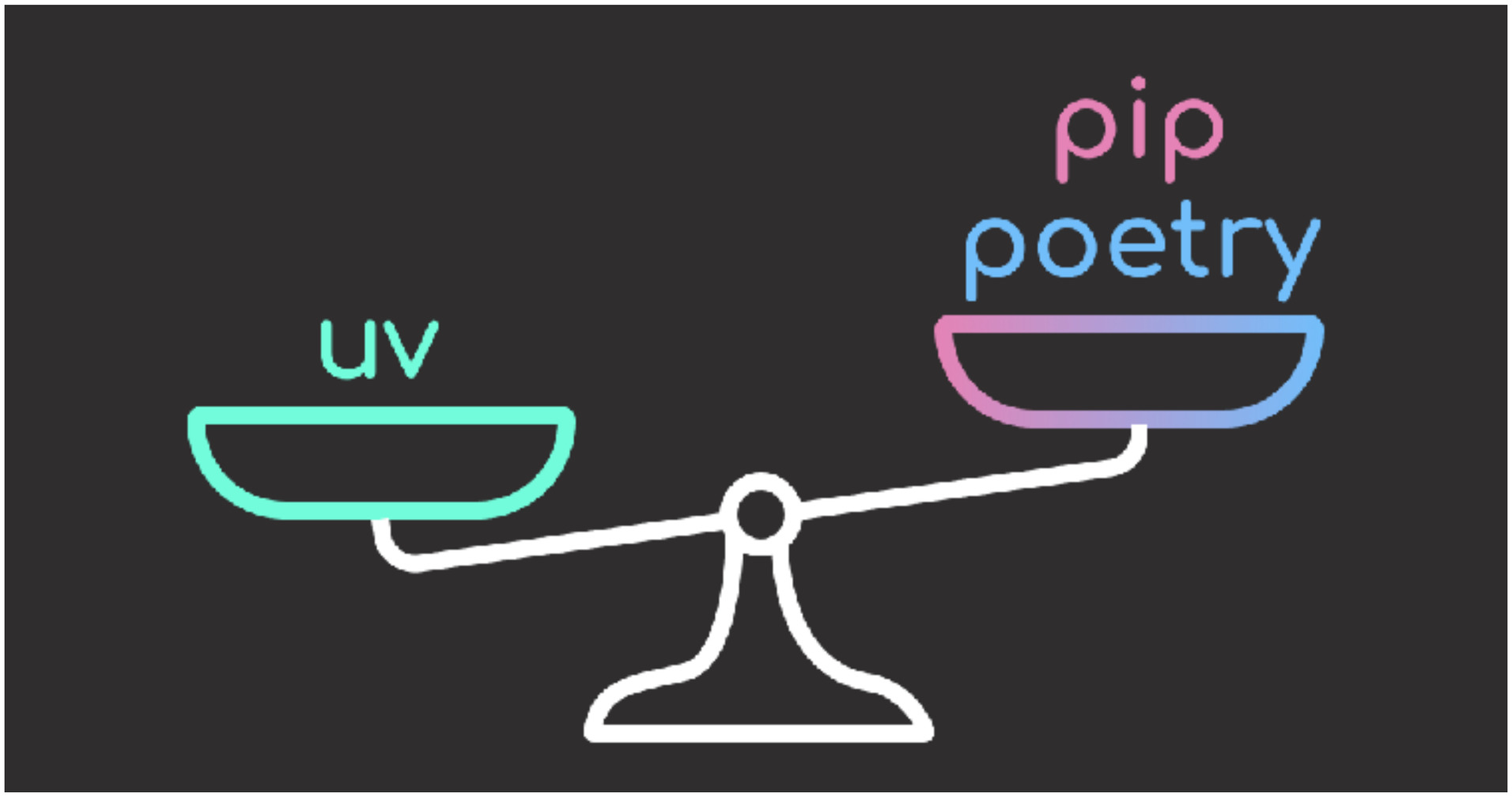
Environment Management

Package Management

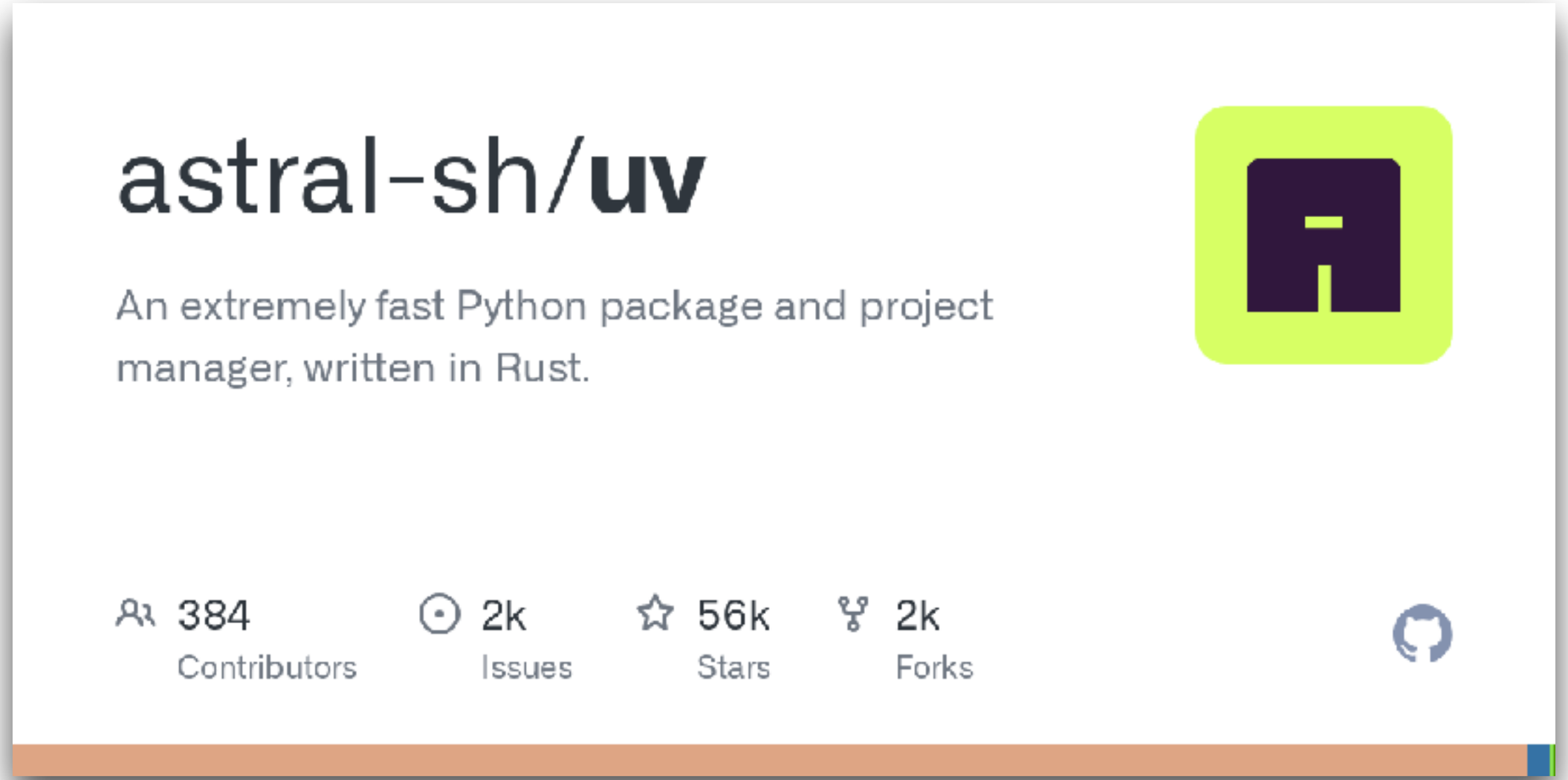


https://alpopkes.com/posts/python/packaging_tools/

Tools ?



UV



https://github.com/astral-sh/uv

Development tools

Code formatting
Linting

Build tools

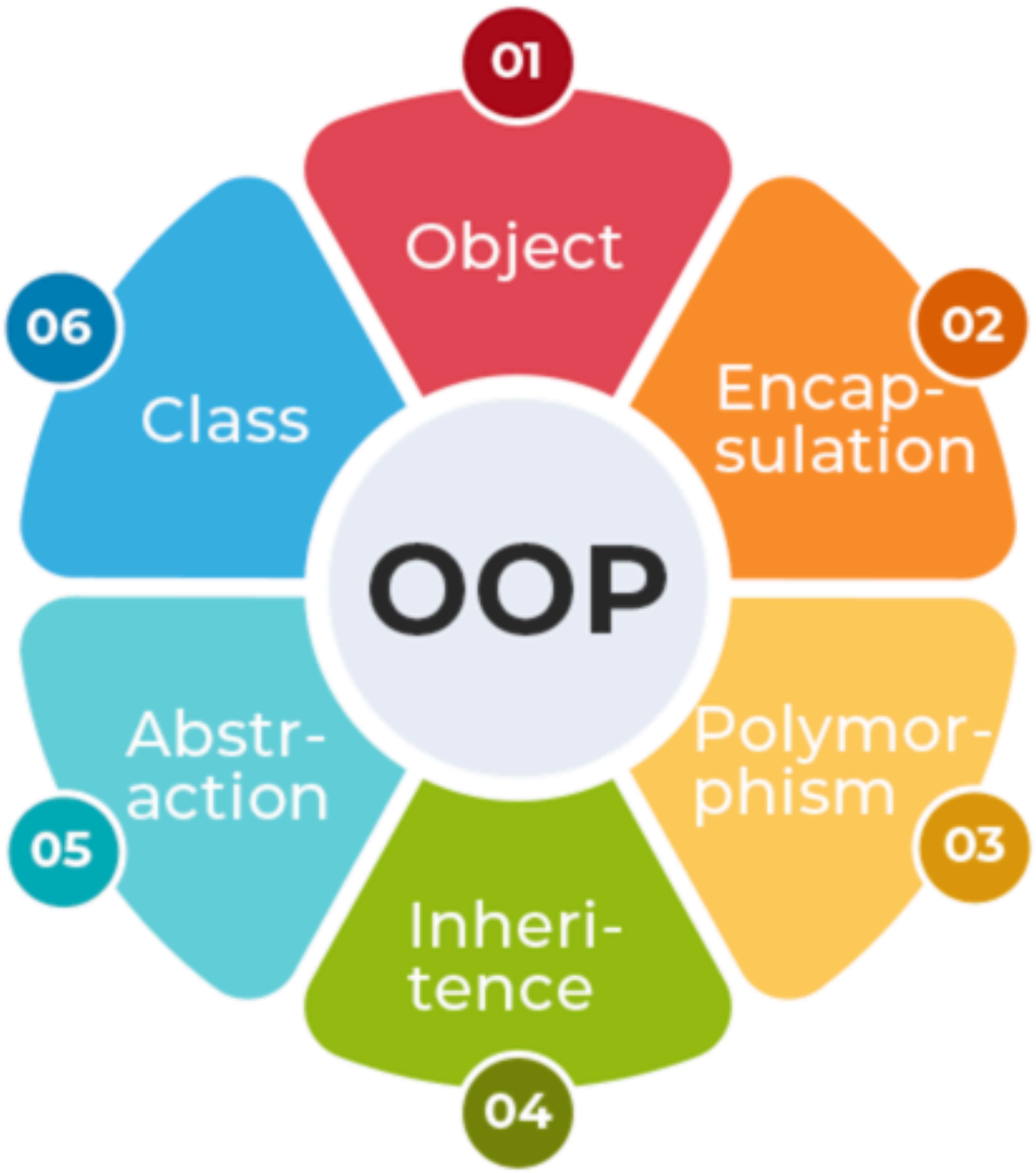
Documentation

Testing

Object Oriented Programming

Classes and Object

OOP concepts



Class and Object

Class name
List of properties/attributes (data members)
List of behaviors/methods
Constructor

Single Responsibility Principle

Workshop
OOP with Harry Potter

https://codingdojo.org/kata/Potter/

Design class and relationship

Book

Order

OrderItem

Book store

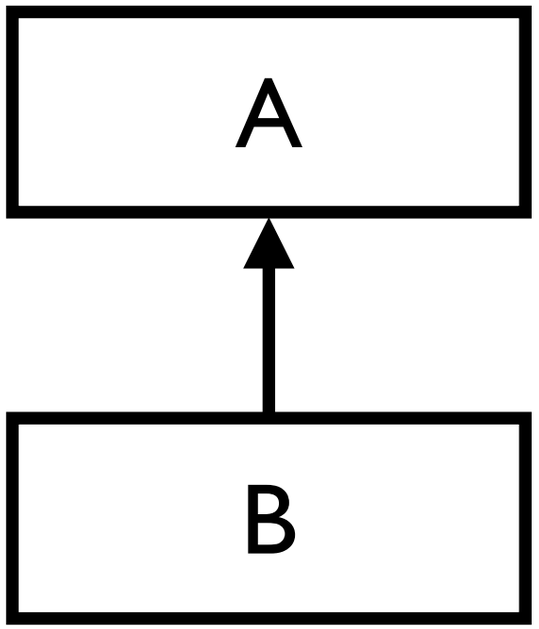
PriceCalculator

DiscountCalculator

Inheritance

Ability to derive a new class from existing class

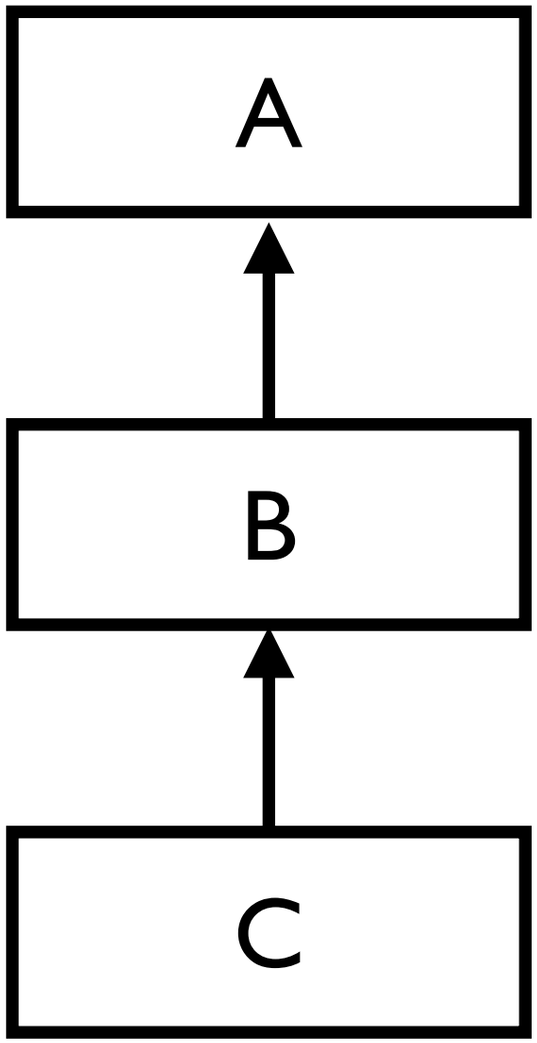
superclass / base class



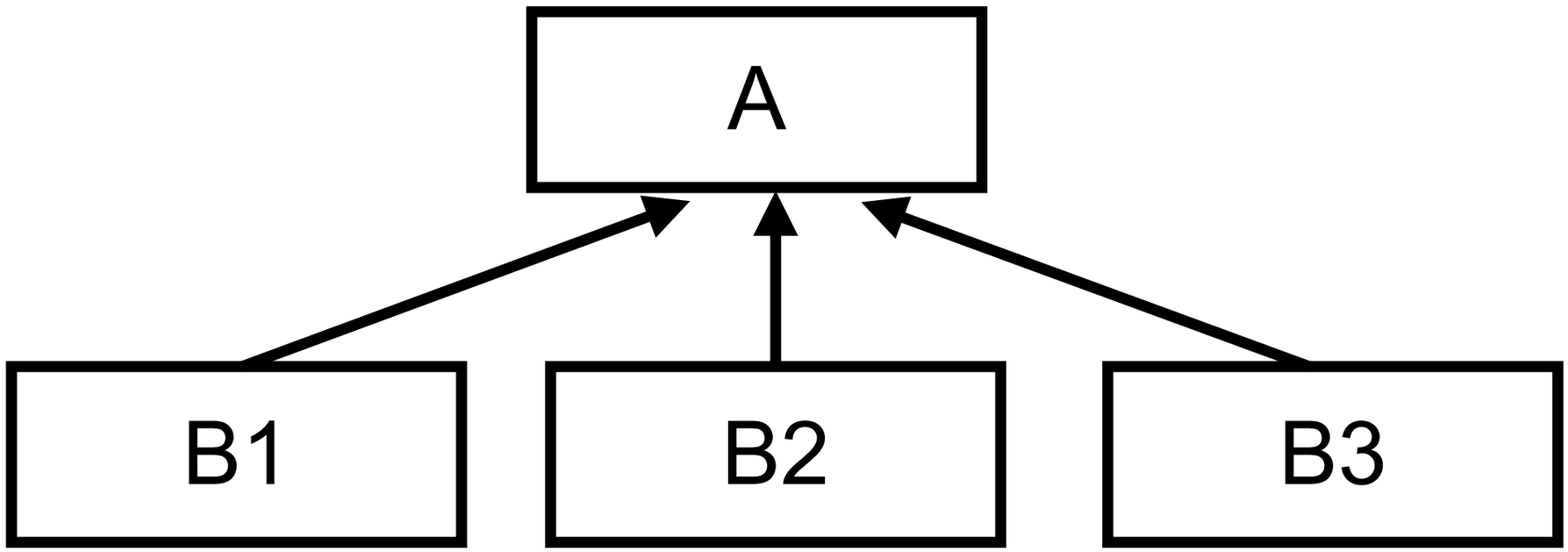
subclass / derived class

Additional properties and behaviors

Multilevel Inheritance

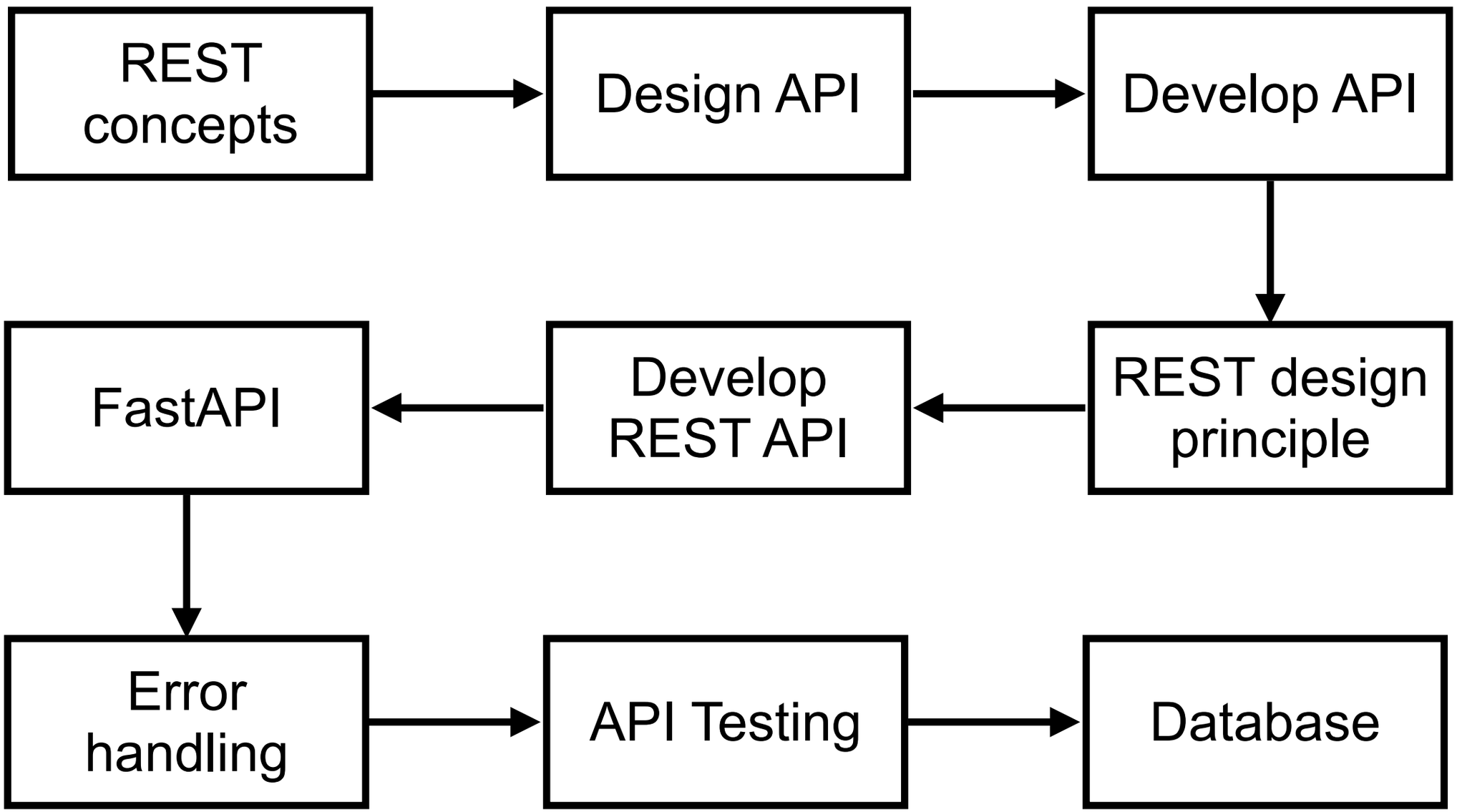


Hierarchical Inheritance

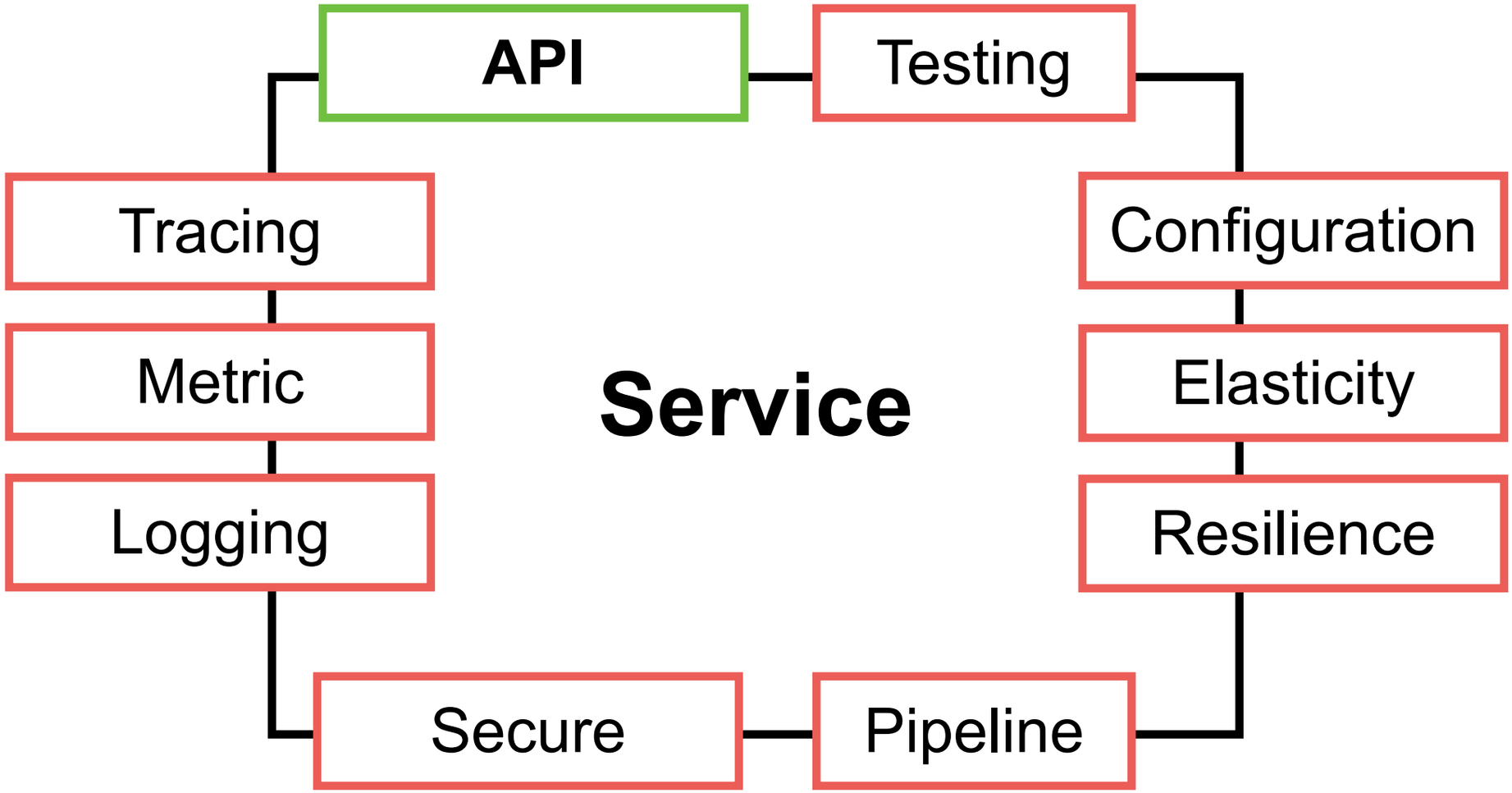


Part 2

Learning Path 2

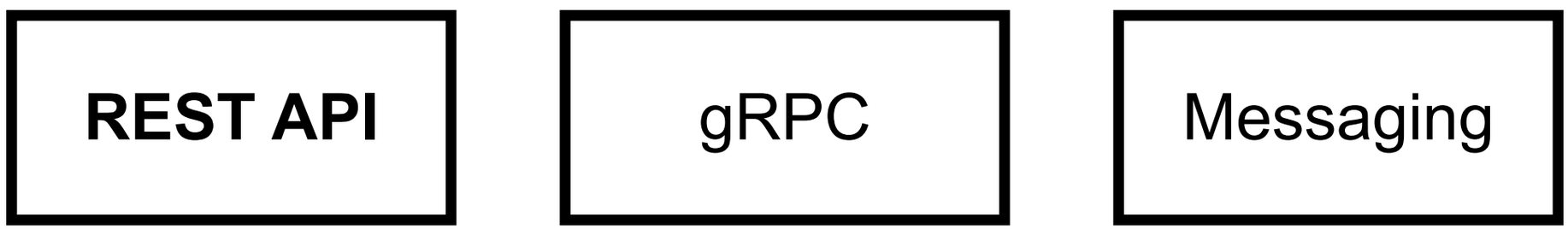


Properties of Service



APIs

Application **P**rogramming **I**nterface



REST

REpresentation **S**tate **T**ransfer

The style of **software architecture** behind RESTful services

Defined in 2000 by Roy Fielding

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Goals

Scalability
Generality of interfaces
Independent deployment of components

REST properties

Performance

Scalability

Simplicity

Modification

Visibility

Portability

Reliability

REST constraints

Client-server

Stateless

Cacheable

Layer system

Uniform interface

Code on demand

eg. JavaScript

RESTful service

Implementation from REST

REST Request & Response

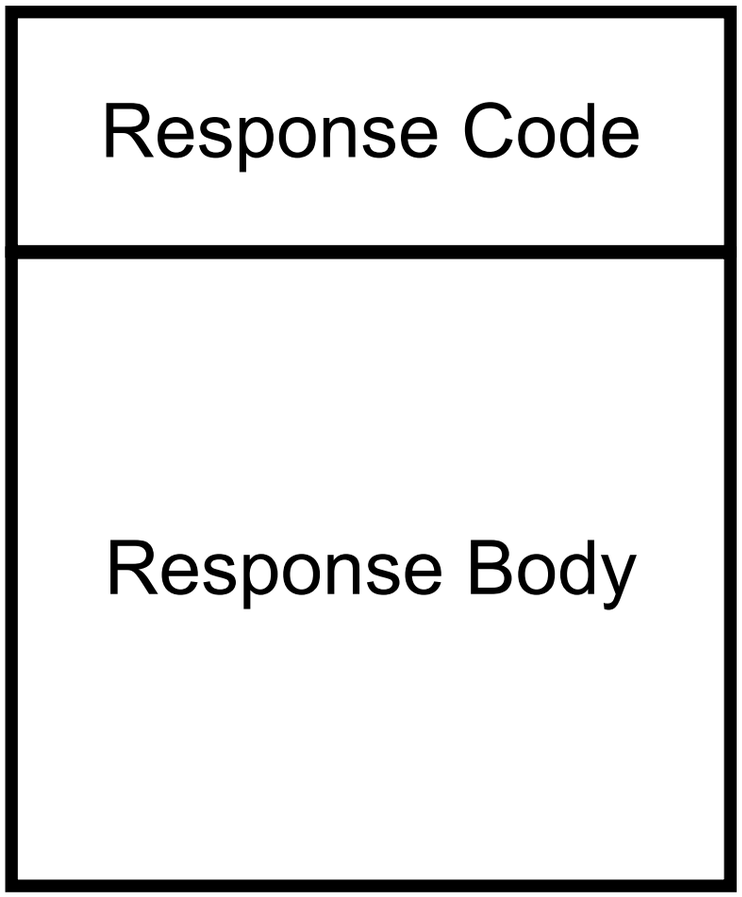
HTTP Request methods

Method	Meaning
GET	Read data
POST	Create/Insert new data
PUT/PATCH	Update data or insert if a new id
DELETE	Delete data

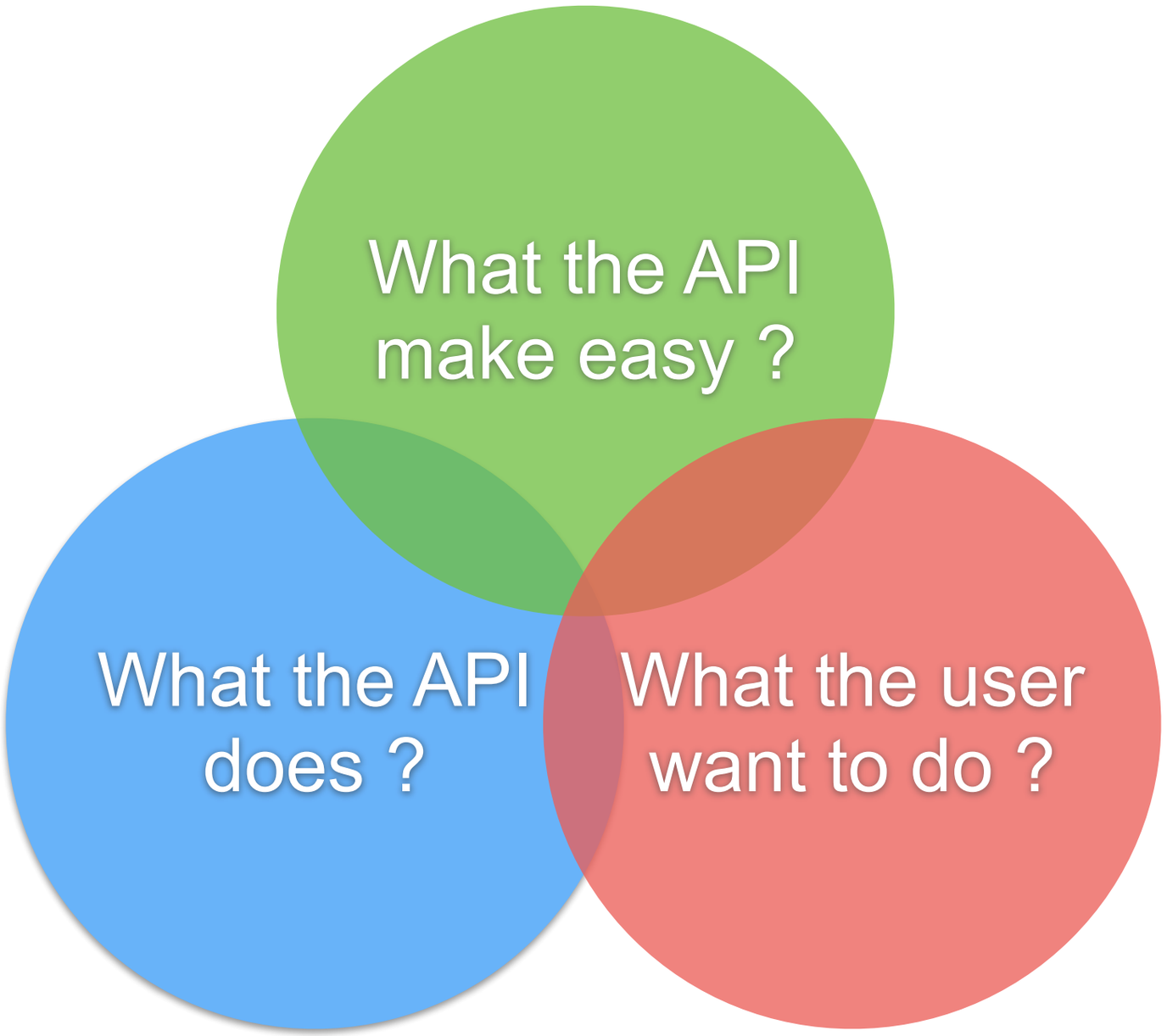
HTTP status codes

Code	Description	Example
1xx	Informational responses	100 = Continue 101 Switching protocols
2xx	Success	200 = OK 201 = Created
3xx	Redirection	300 = Multiple choices 301 = Moved permanently
4xx	Client errors	400 = Bad request 403 = Forbidden 404 = Not found
5xx	Server errors	500 = Internal server error 502 = Bad gateway 503 = Service unavailable

Response format ?



Good APIs ?

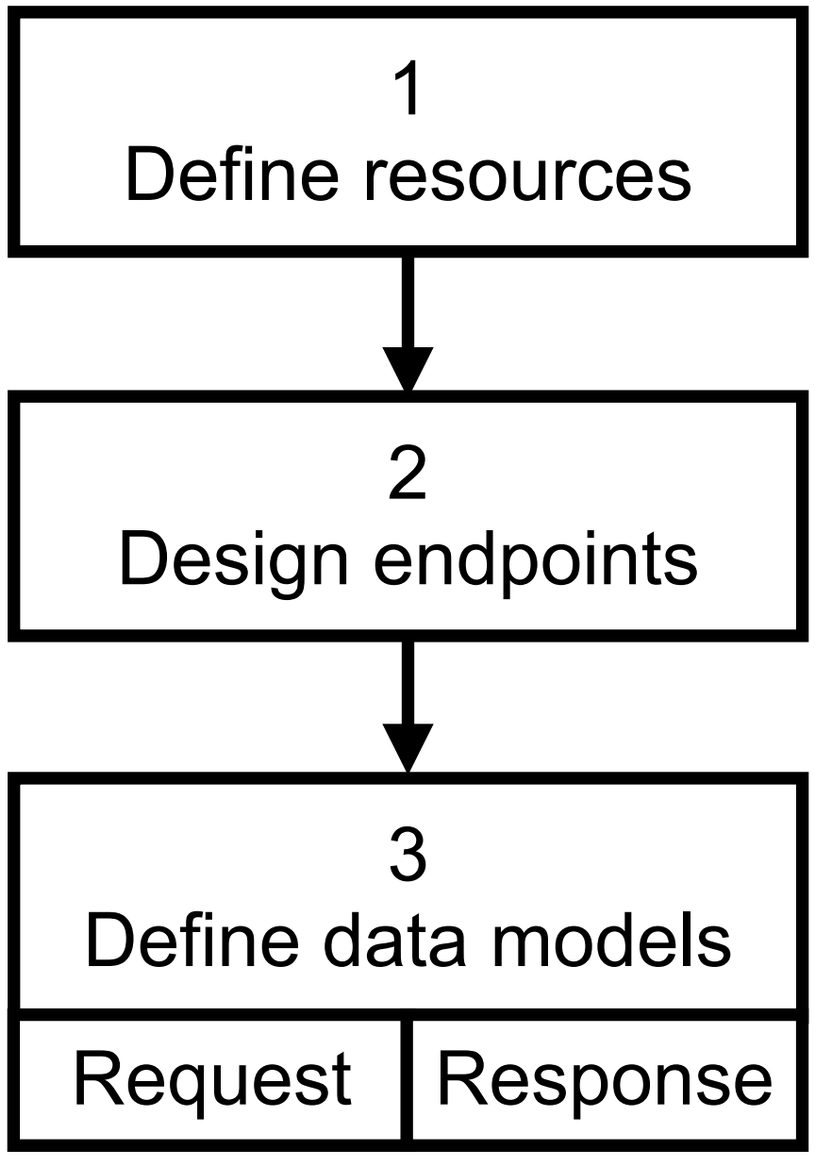


Principles of API Design

Principles of API Design

Consistency	Statelessness	Resource-oriented design
Use standard HTTP methods	Versioning	

Steps to Design APIs



More !!

Authentication
Authorization

Pagination and filtering

Rate Limiting

Pagination and filtering

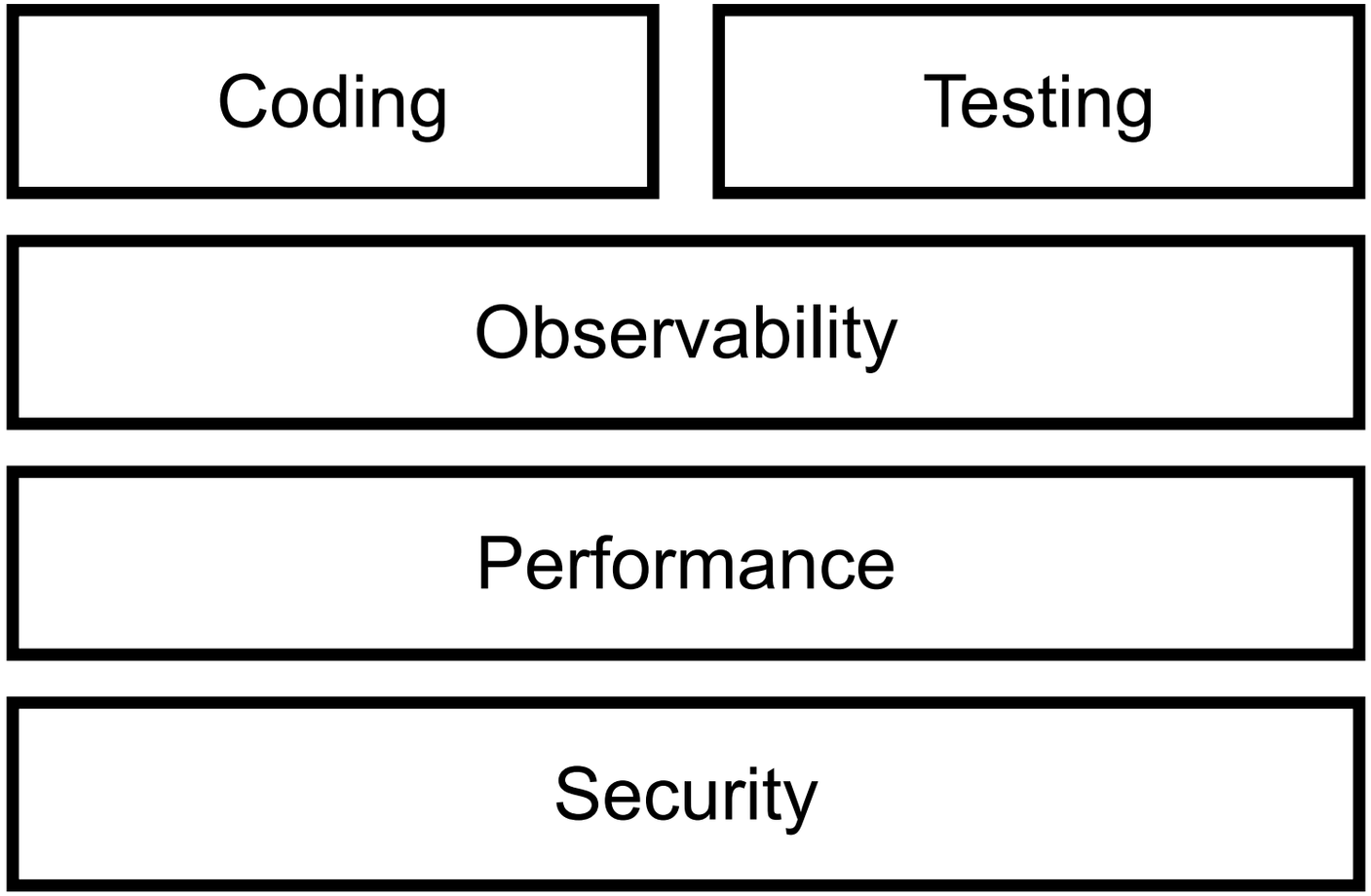
Error handling

Documentation

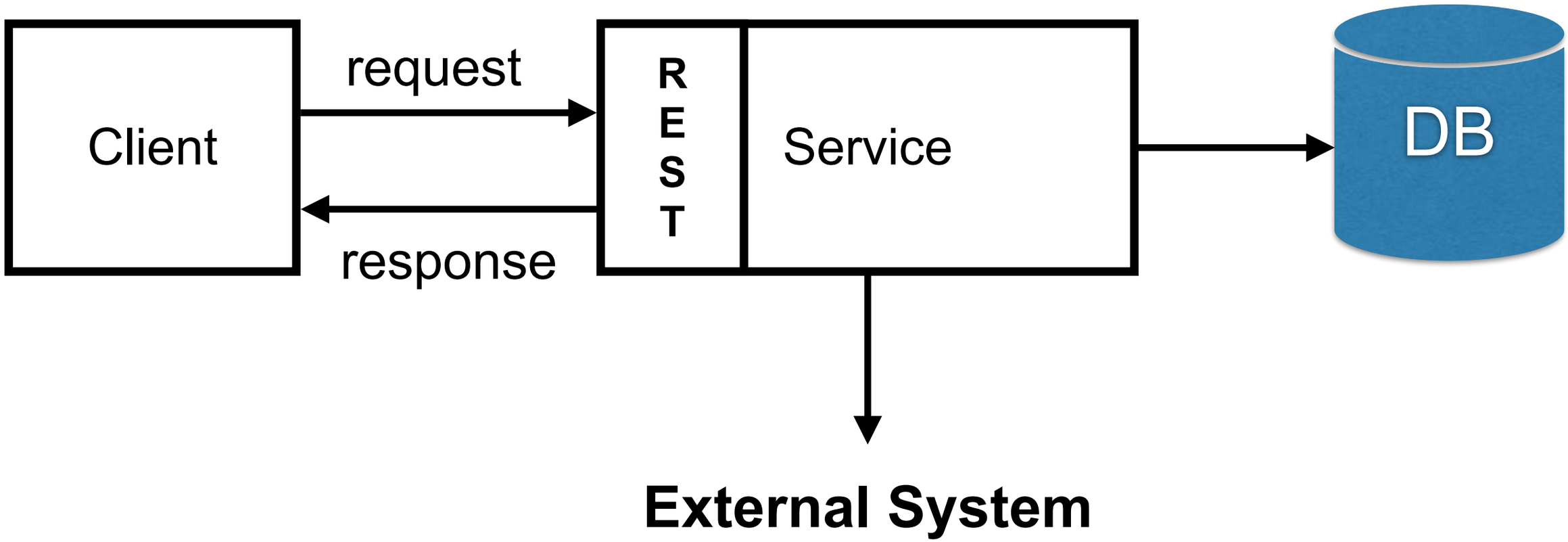
Testing

Monitoring and Observability

Implementation



Structure of Service

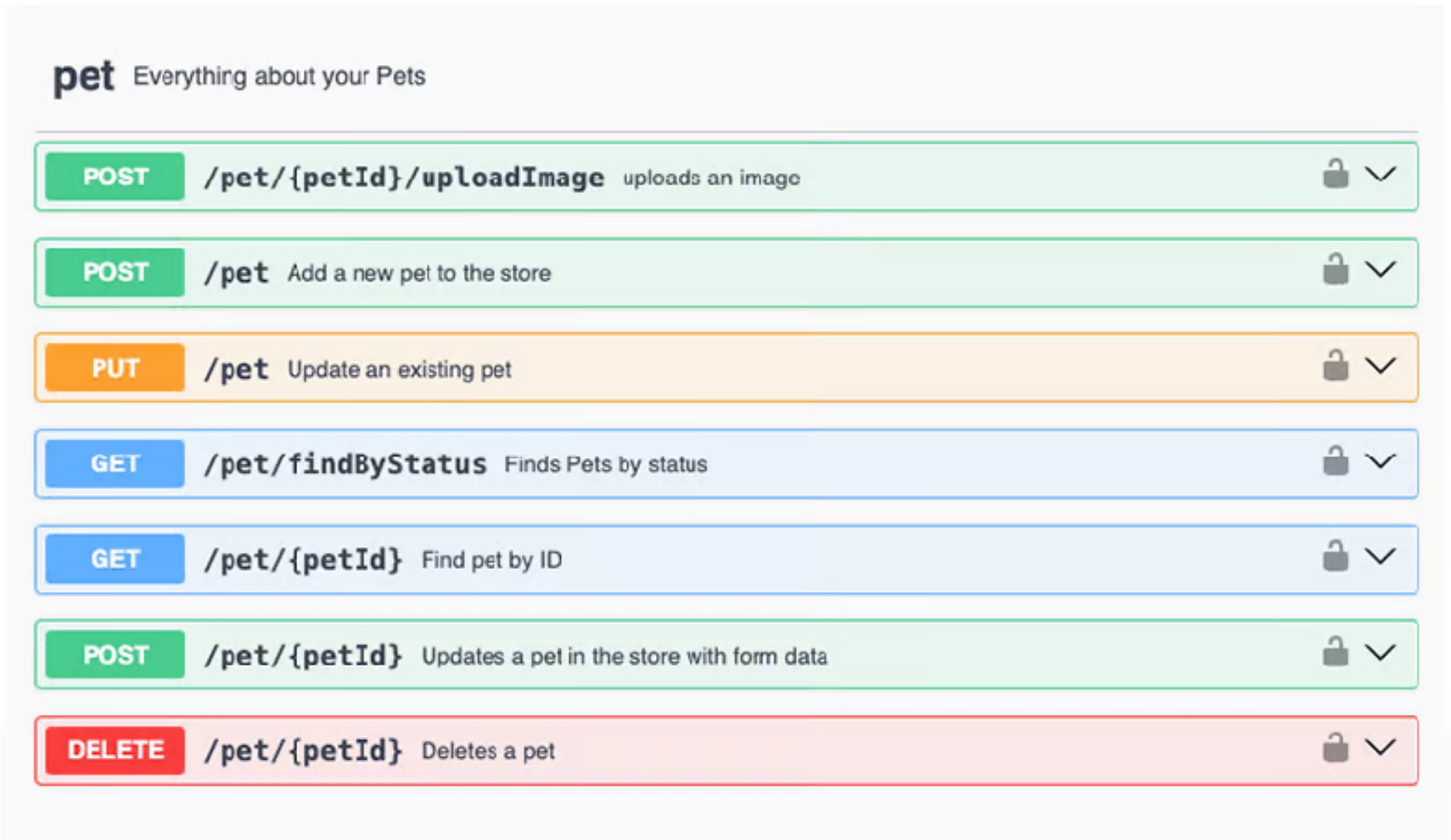


Design RESTFul APIs

Users

Method	Path	Description
GET	/users	Get all users
GET	/users/{id}	Get user by id
POST	/users	Create new user
PUT	/users/{id}	Update user
DELETE	/users/{id}	Delete user by id

API Documentation



https://swagger.io/

Develop RESTFul APIs

RESTFul API with Python

FastAPI

https://fastapi.tiangolo.com/

FastAPI

Fast

Fewer bugs

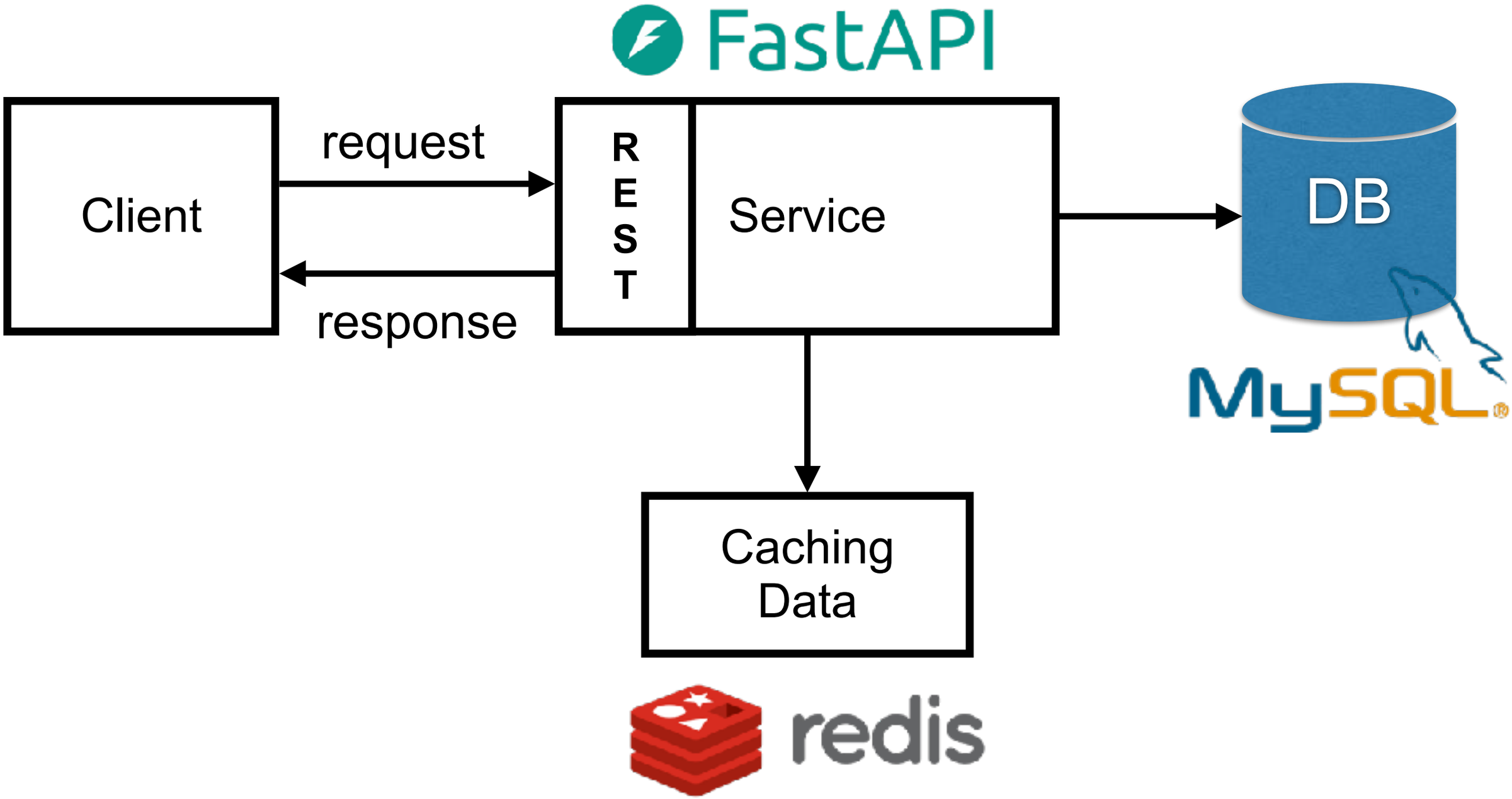
Easy

Production ready

OpenAPI

JSON Schema

Workshop



Working with Database

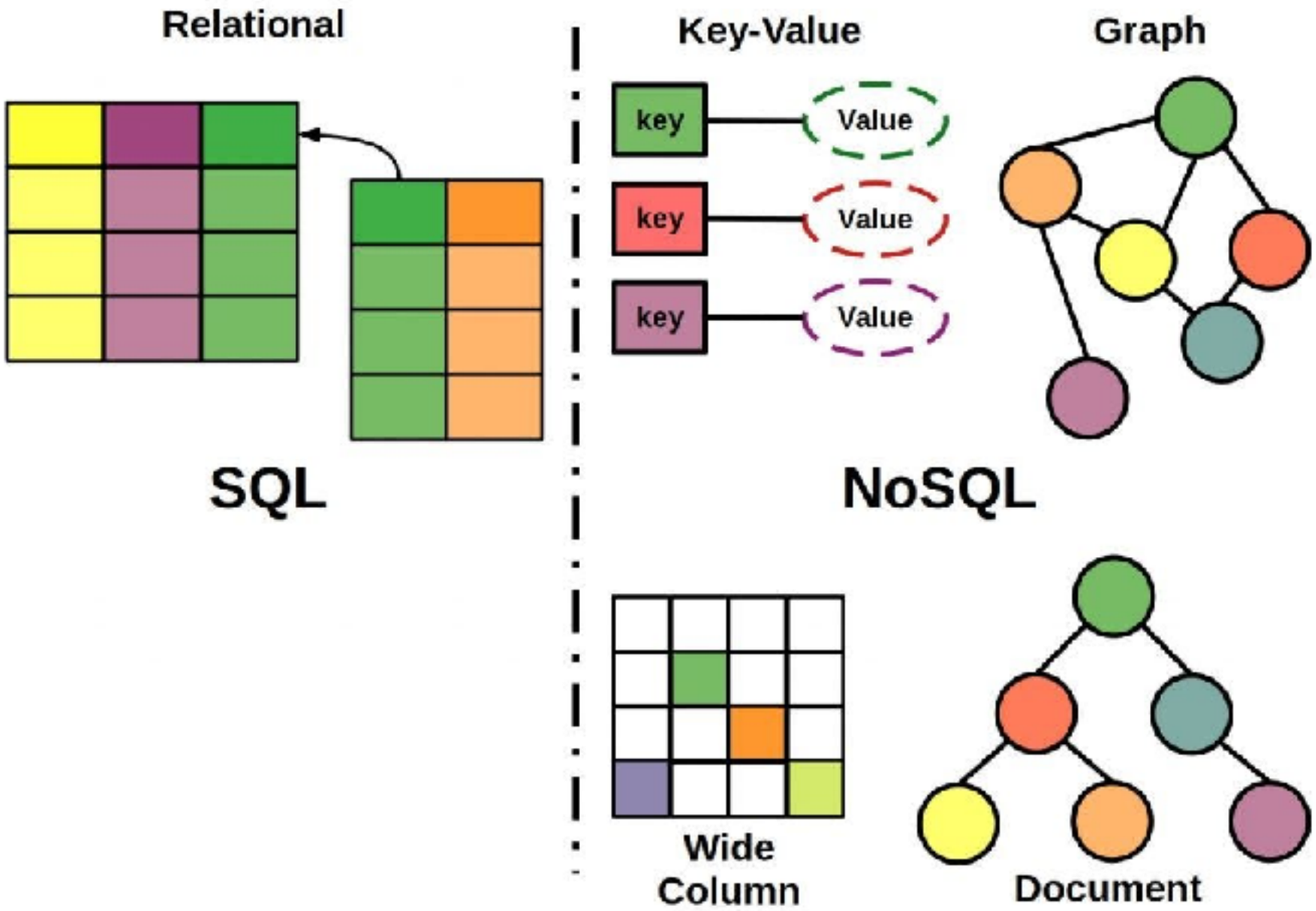
Database Ranking

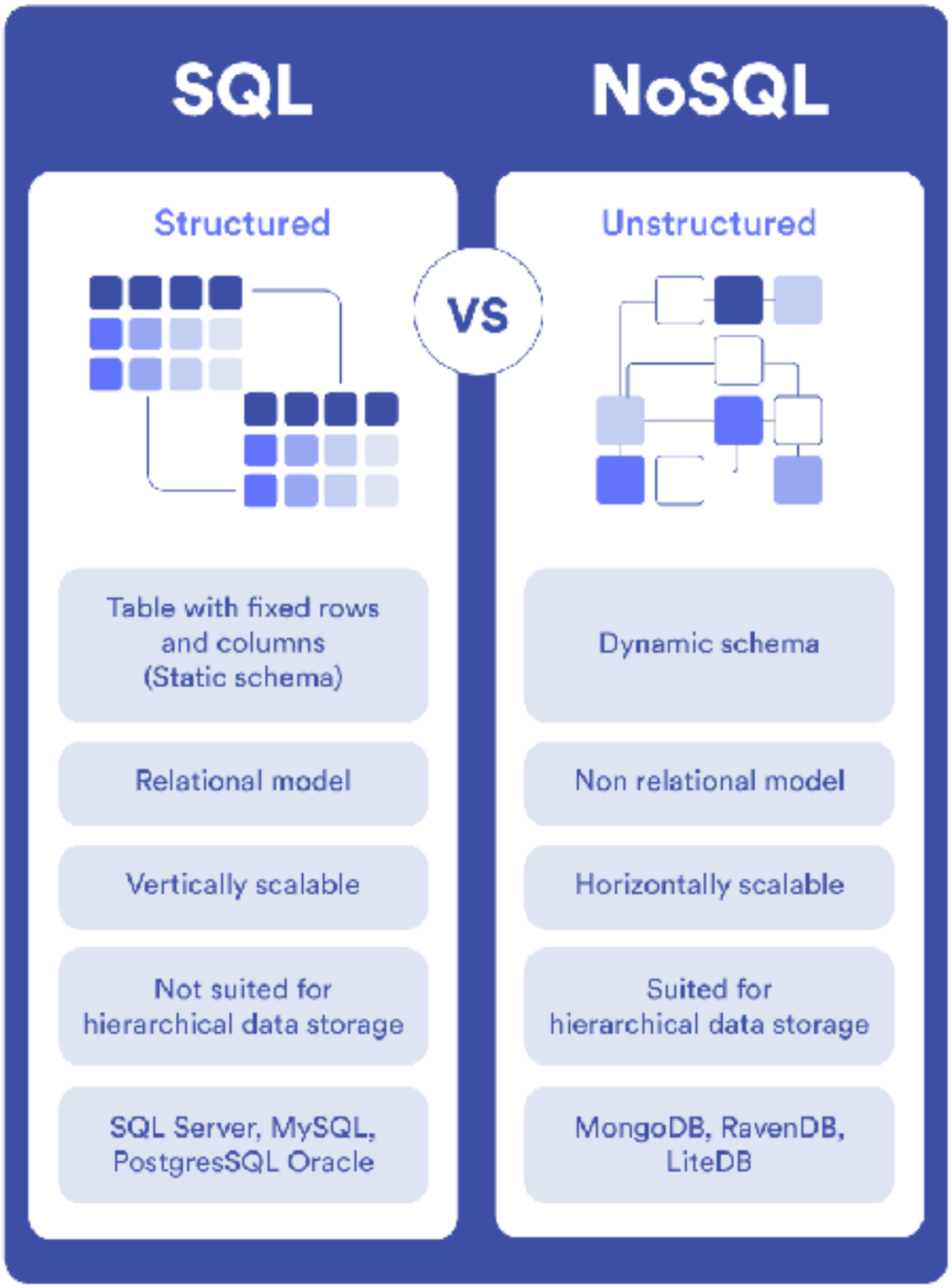
424 systems in ranking, June 2025

Rank Jun 2025	Rank May 2025	Rank Jun 2024	DBMS	Database Model	Score Jun 2025	Score May 2025	Score Jun 2024
1.	1.	1.	Oracle	Relational, Multi-model	1230.38	+3.82	-13.70
2.	2.	2.	MySQL	Relational, Multi-model	953.57	-11.41	-107.77
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	776.75	+1.85	-44.81
4.	4.	4.	PostgreSQL	Relational, Multi-model	680.65	+6.34	+44.41
5.	5.	5.	MongoDB	Document, Multi-model	402.85	+0.33	-18.23
6.	6.	↑ 8.	Snowflake	Relational	174.49	+2.48	+44.13
7.	7.	↓ 6.	Redis	Key-value, Multi-model	151.72	-0.47	-4.22
8.	8.	↑ 9.	IBM Db2	Relational, Multi-model	125.13	-1.27	-0.77
9.	9.	↓ 7.	Elasticsearch	Multi-model	121.28	-2.53	-11.55
10.	10.	10.	SQLite	Relational	117.03	-0.74	+5.63
11.	11.	↑ 12.	Apache Cassandra	Wide column, Multi-model	108.27	+0.22	+9.44
12.	12.	↑ 15.	Databricks	Multi-model	104.67	+2.02	+23.59
13.	↑ 14.	13.	MariaDB	Relational, Multi-model	94.54	+0.93	+3.50
14.	↓ 13.	↓ 11.	Microsoft Access	Relational	88.28	-7.63	-12.88
15.	15.	↑ 17.	Amazon DynamoDB	Multi-model	83.34	+4.04	+8.90

https://db-engines.com/en/ranking

Working with Database

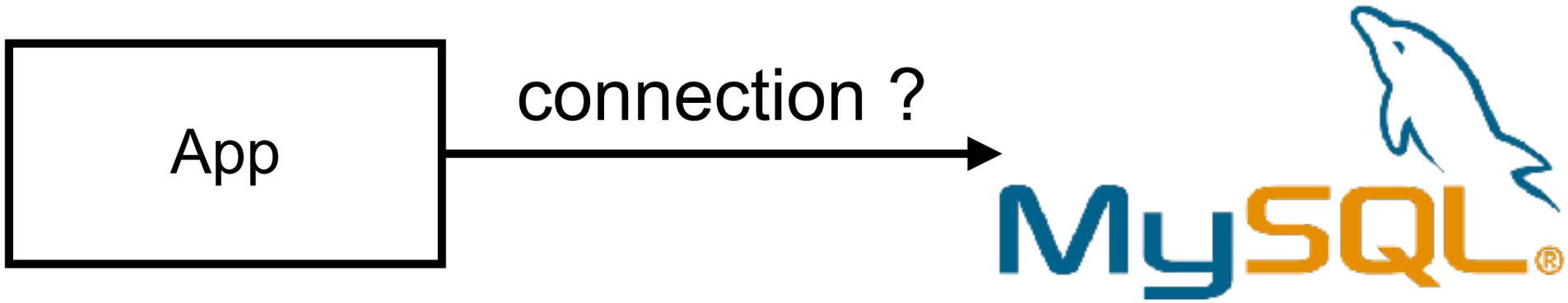




https://pandorafms.com/blog/nosql-vs-sql-key-differences/

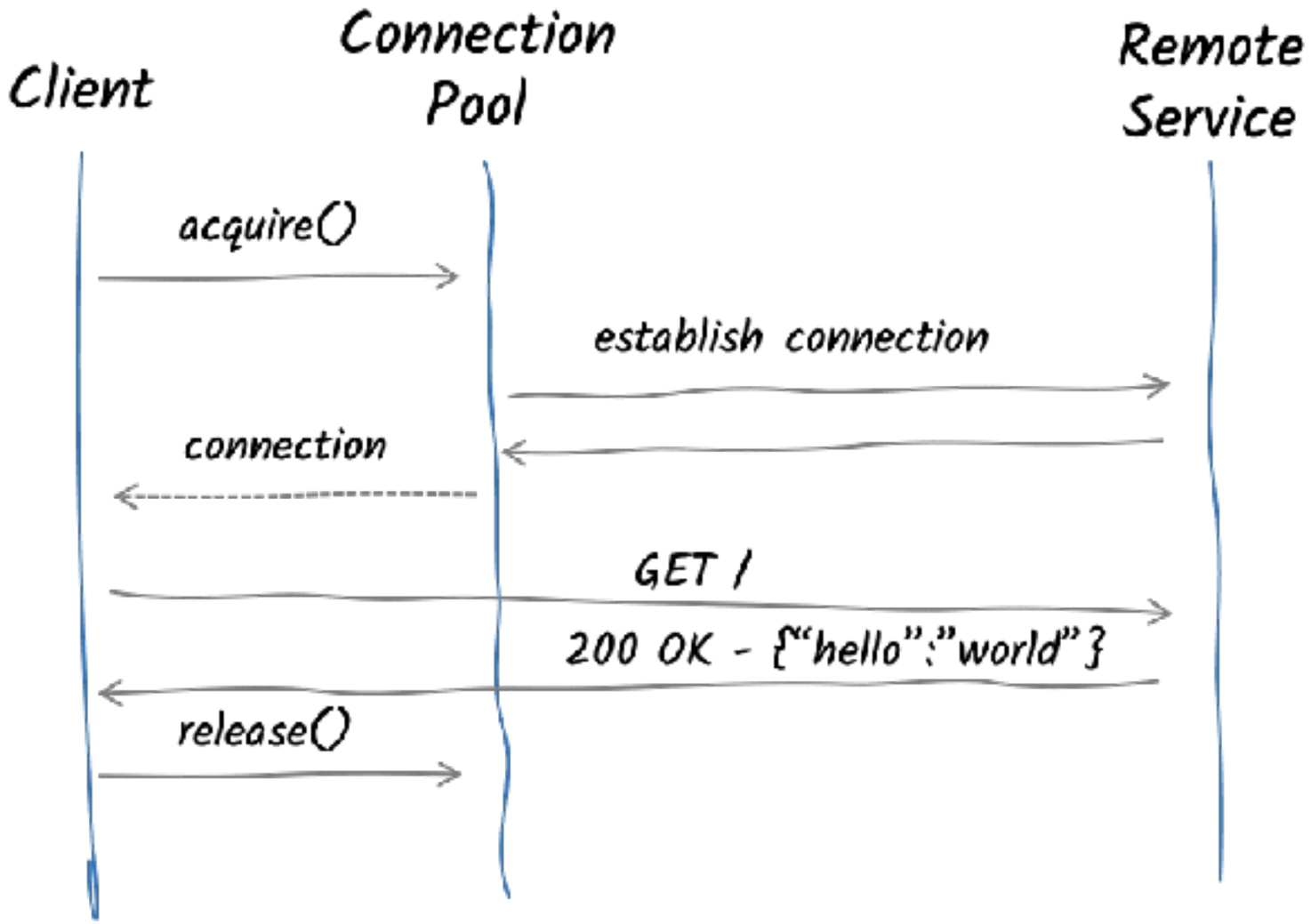
Working with Database

Manage connection ?



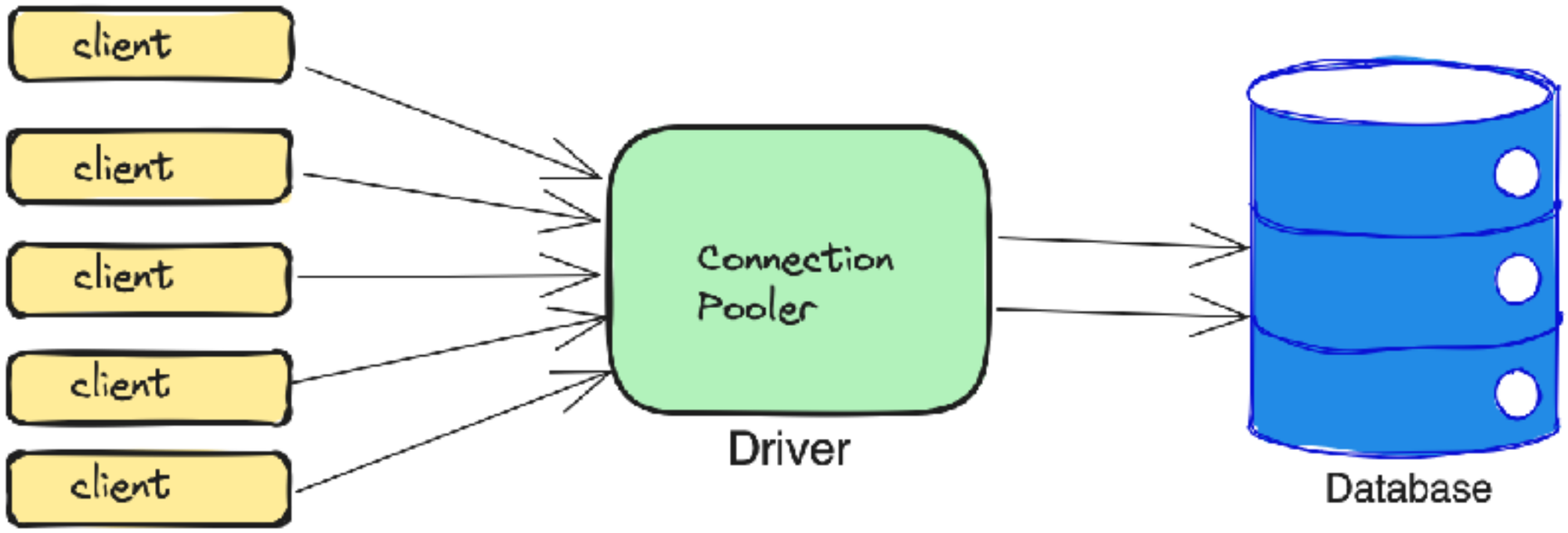
Connection pool

Cache and reuse database connections

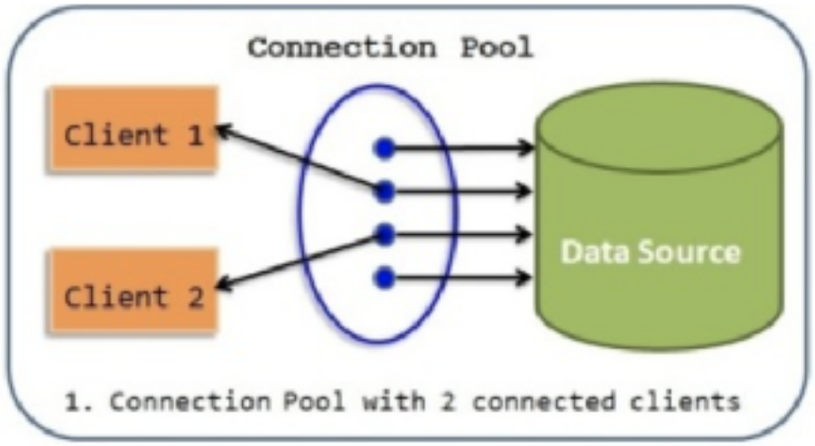


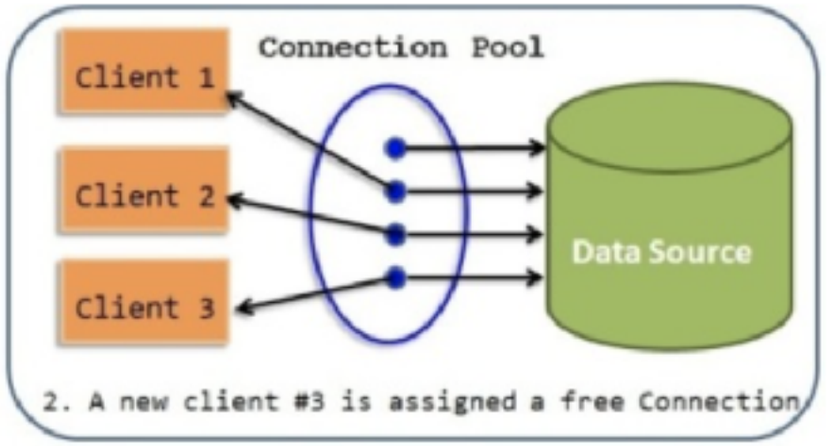
Connection pool

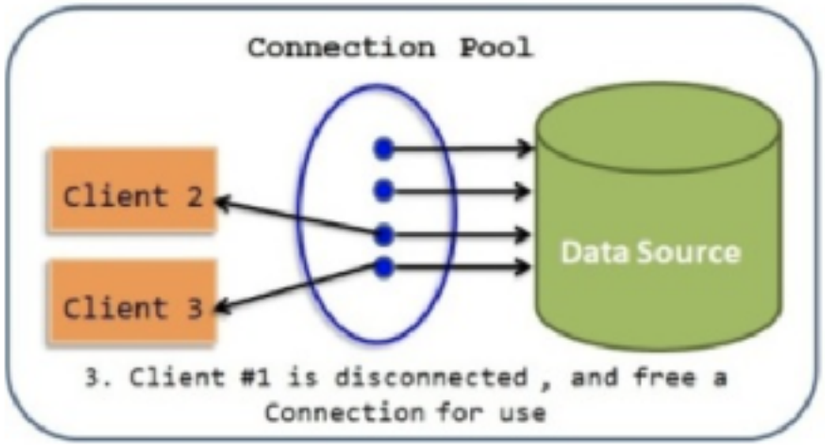
Cache and reuse database connections

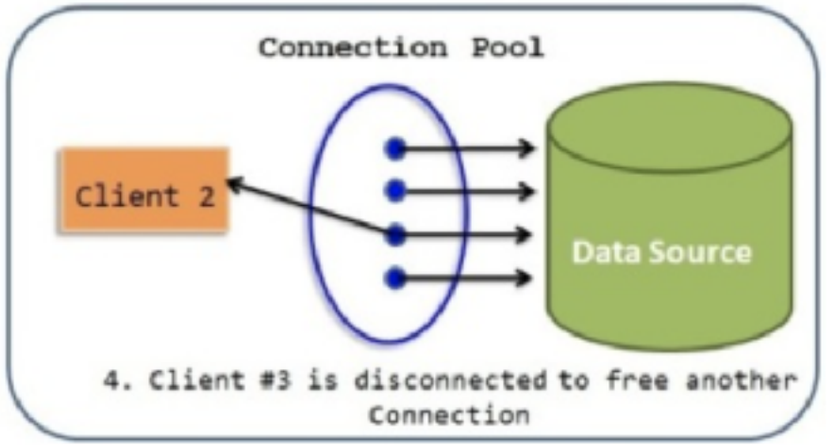


Connection pool





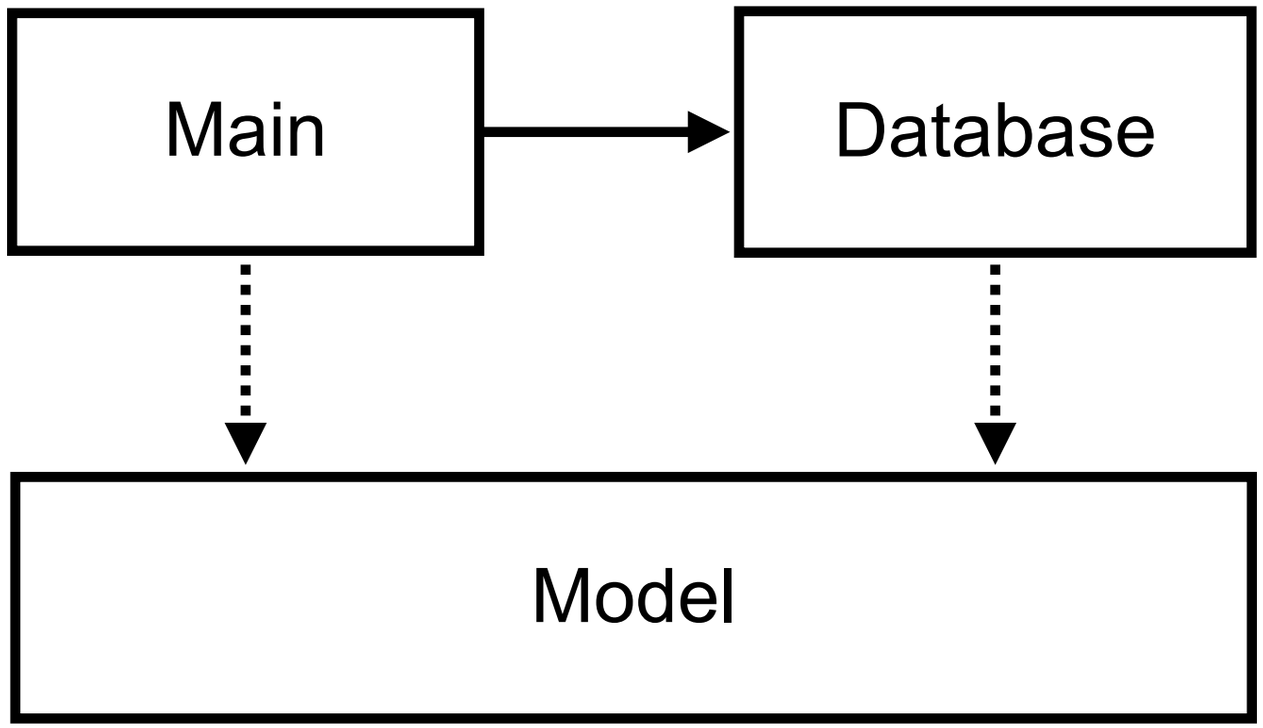




Start workshop

Basic project

CRUD (Create, Read Update, Delete) application

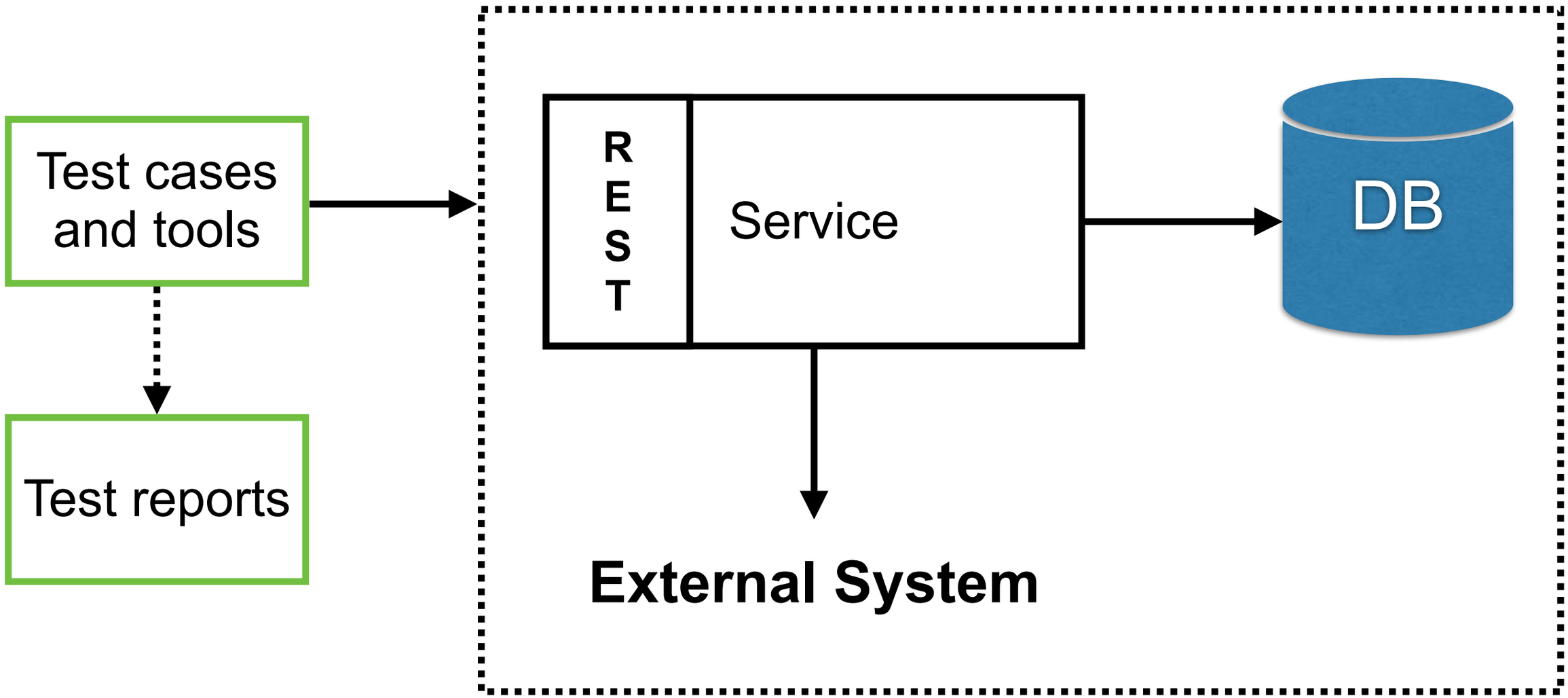


https://github.com/up1/course-python-workshop/wiki/REST-API-with-FastAPI

Testing with APIs

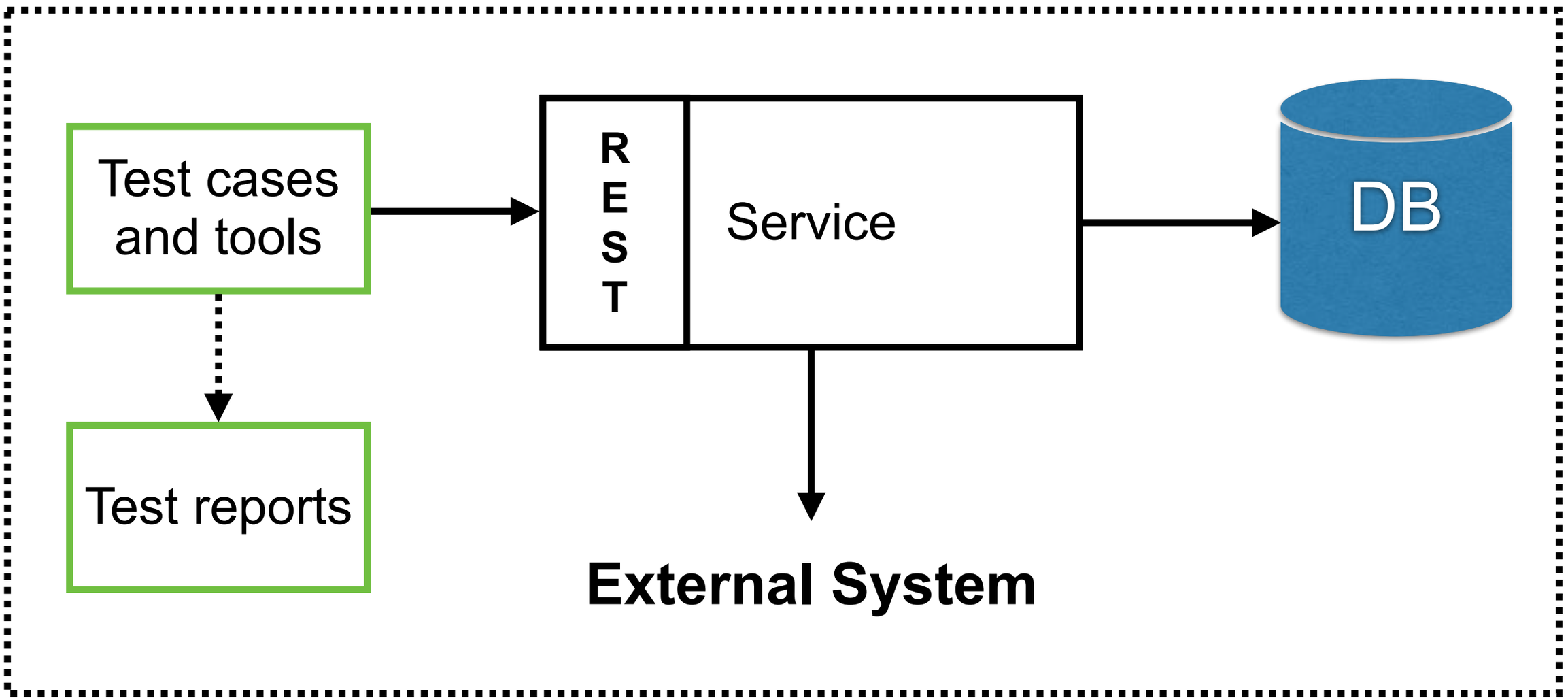
Test strategies

External testing



Test strategies

Internal testing



Testing tools

https://www.postman.com/downloads/

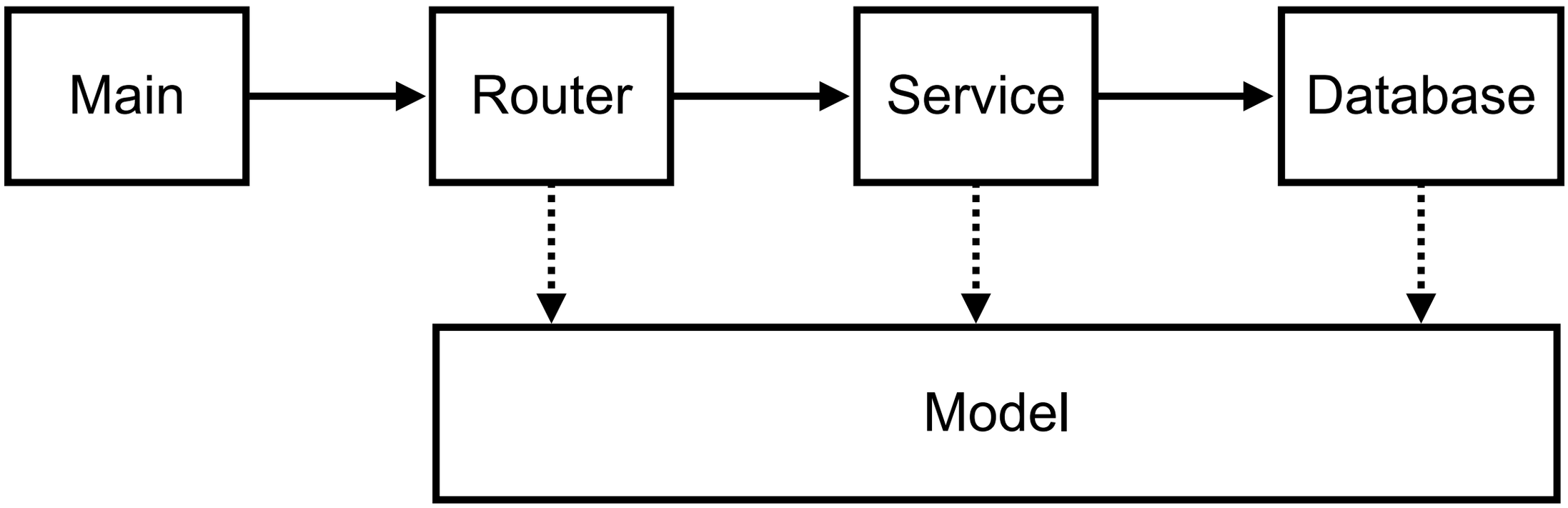
Postman CLI with newman

$npm install -g newman

https://www.npmjs.com/package/newman

Improve structure

Single Responsibility Principle (SRP)
Testability

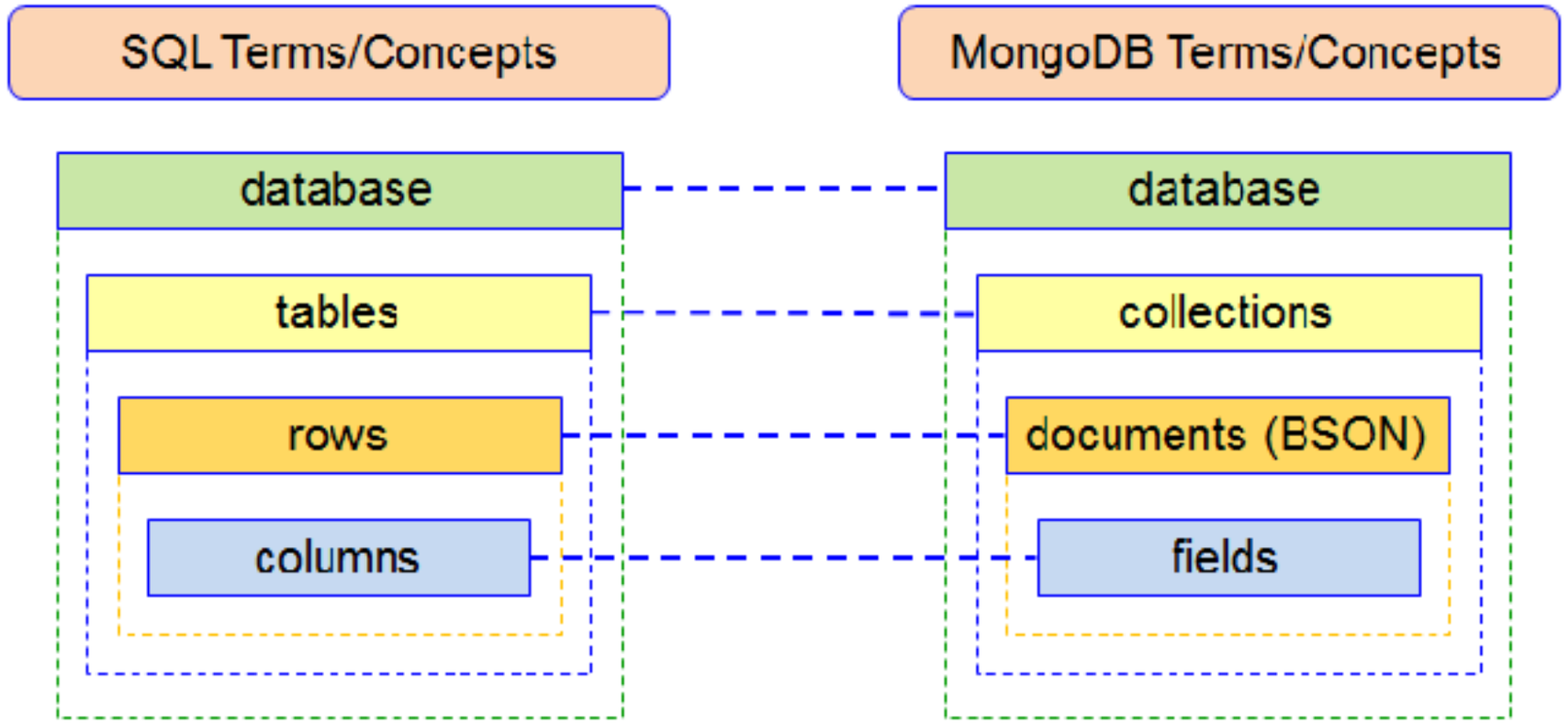


https://github.com/up1/course-python-workshop/wiki/REST-API-with-FastAPI

Working with Document database

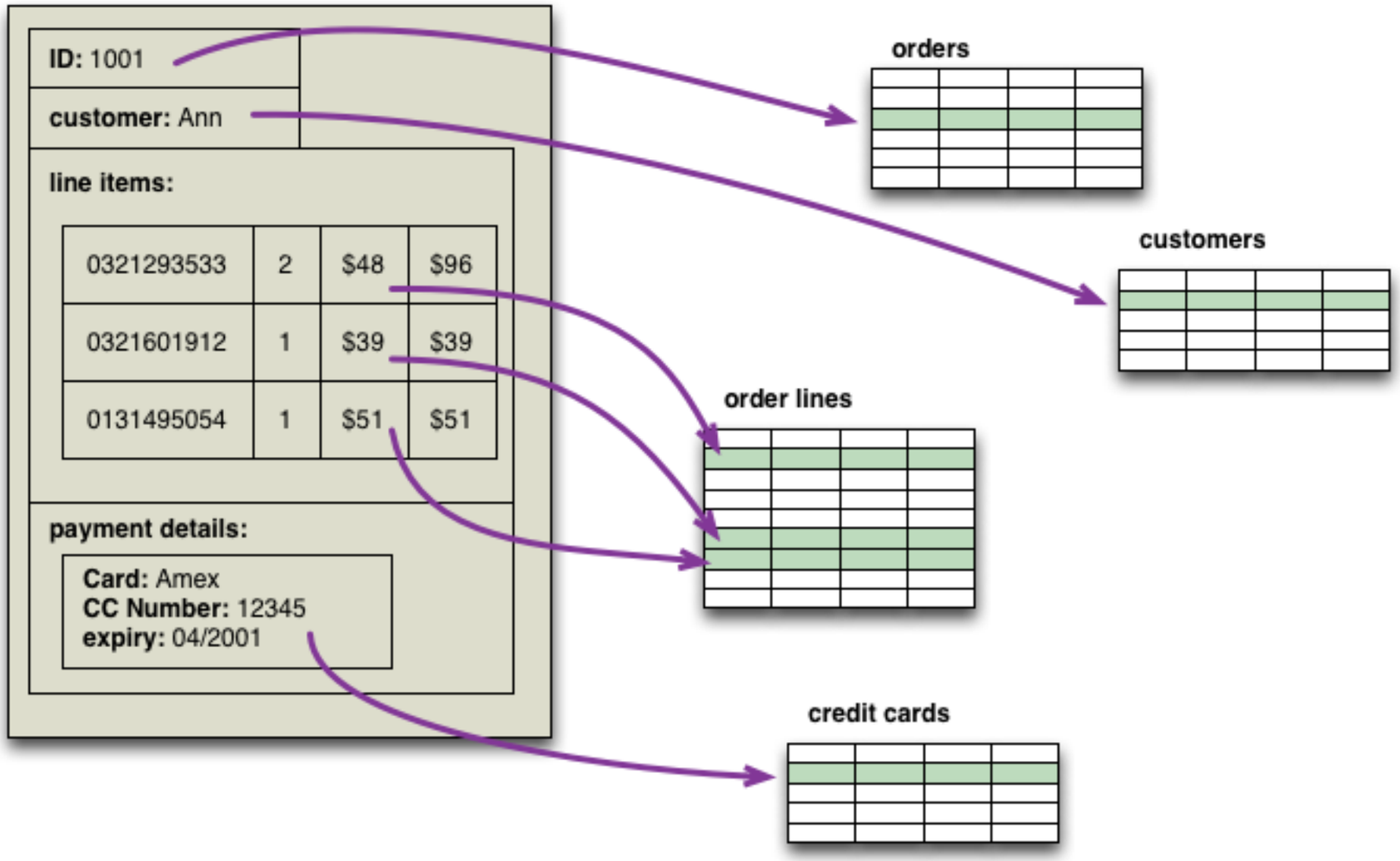
MongoDB

NoSQL database
Document-oriented approach



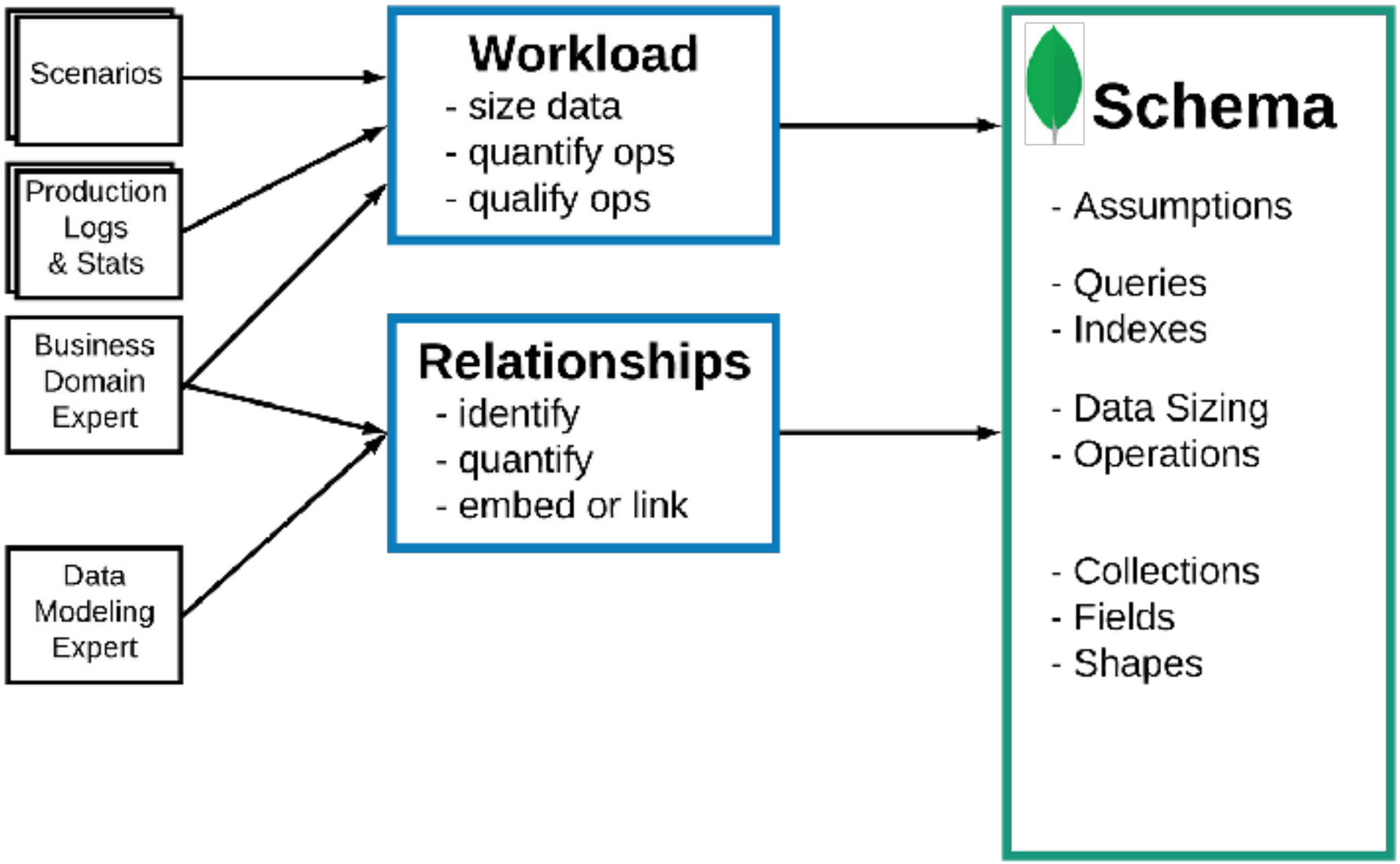
https://www.mongodb.com/

Document-based



https://martinfowler.com/bliki/AggregateOrientedDatabase.html

Data modeling for MongoDB



https://www.mongodb.com/docs/manual/data-modeling/

Working with Key-value database

Redis

REmote **DI**ctionary **S**erver

Key-value database

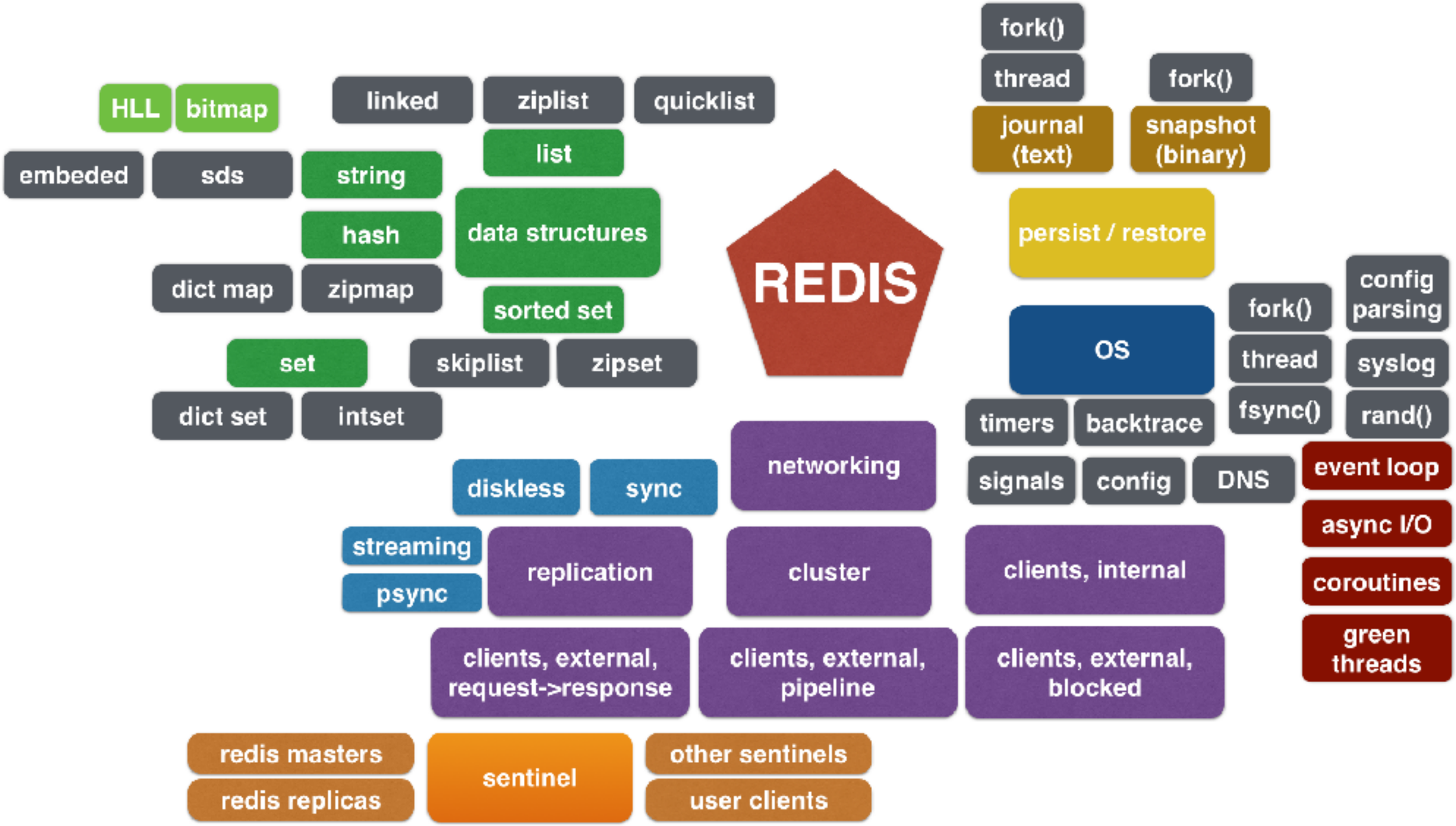
Keep data in memory (save to disk)

Expiration data

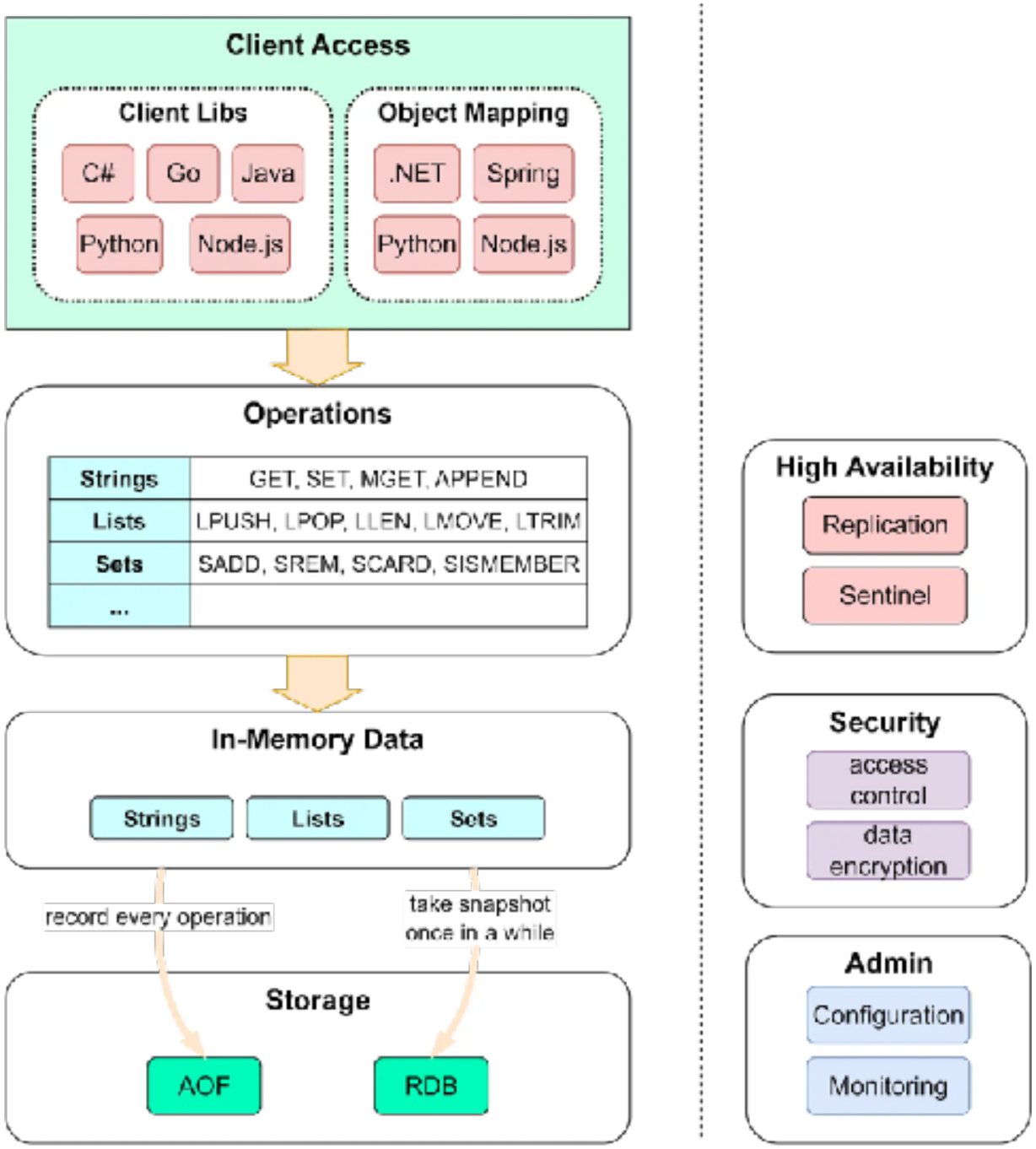
Data structure server

Efficient low-level data structure

https://redis.io/technology/data-structures/



High level view of Reds



https://blog.bytebytego.com/p/a-crash-course-in-redis

Why Redis Fast ?

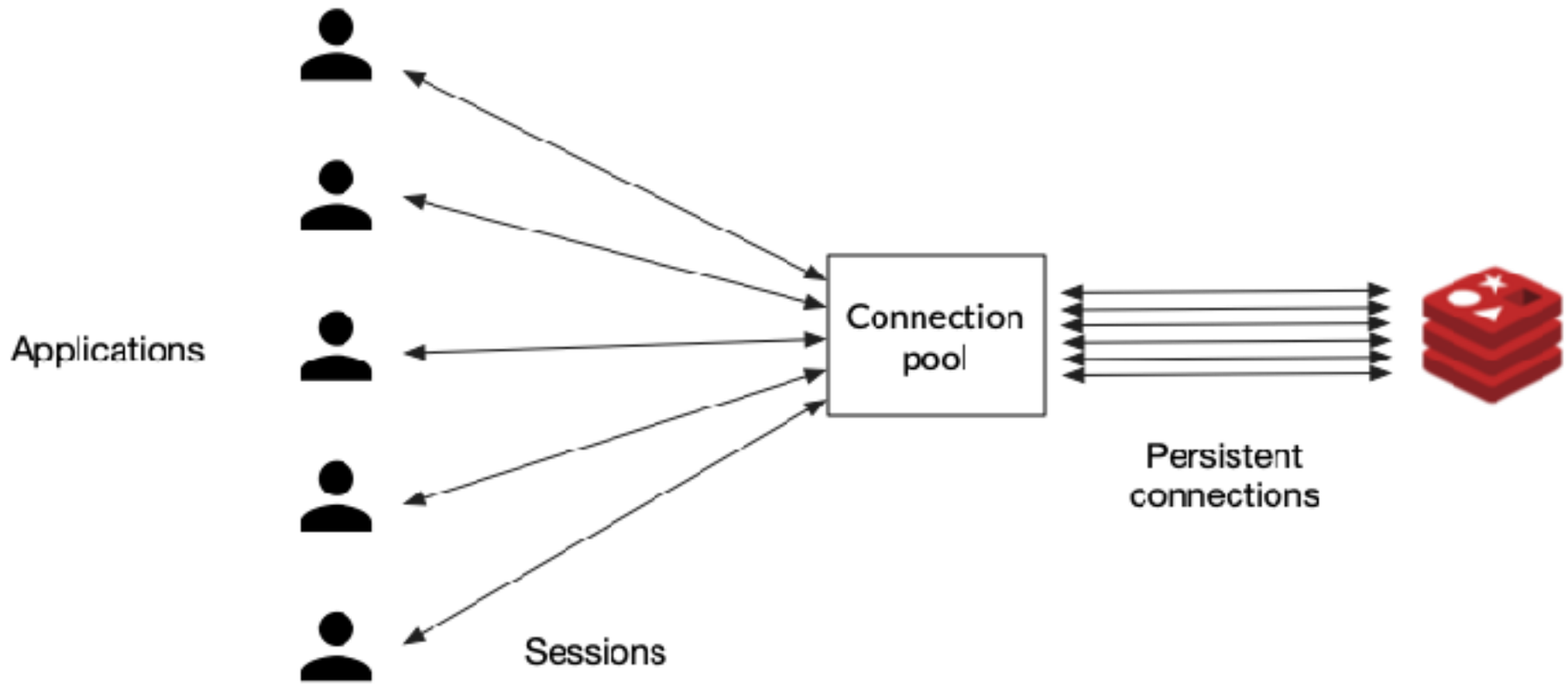
RAM-based database

I/O multiplexing and single-thread read/write

Efficient low-level data structure

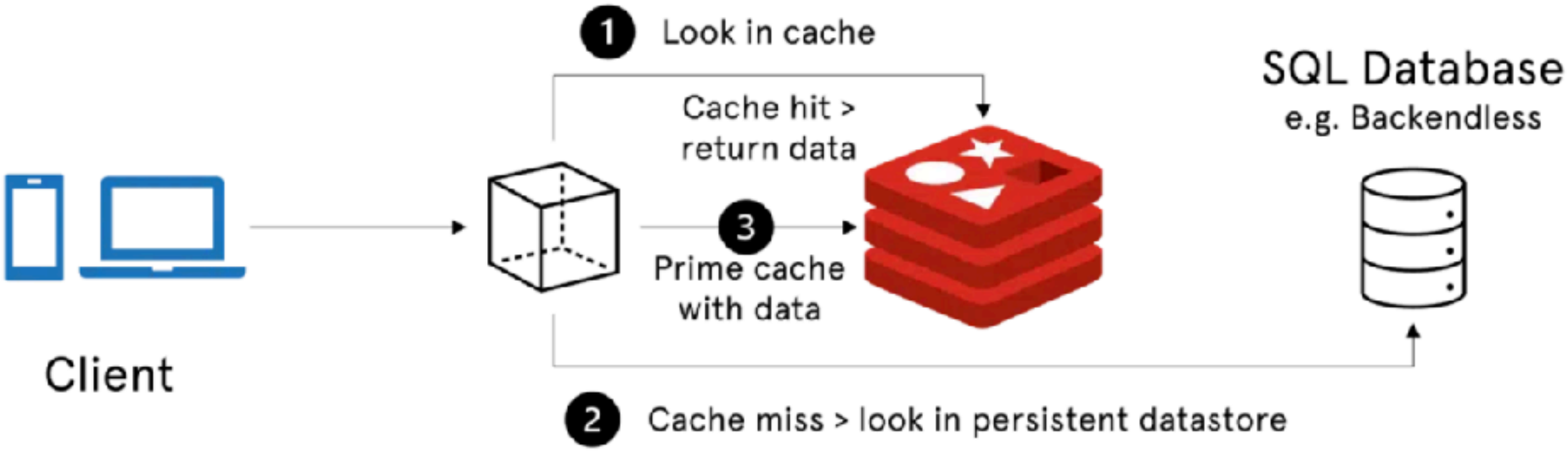
Use connection pool in client

Open and close connections cause overhead !!



Simple Use Case

Caching Data for applications
Reduce workload of database

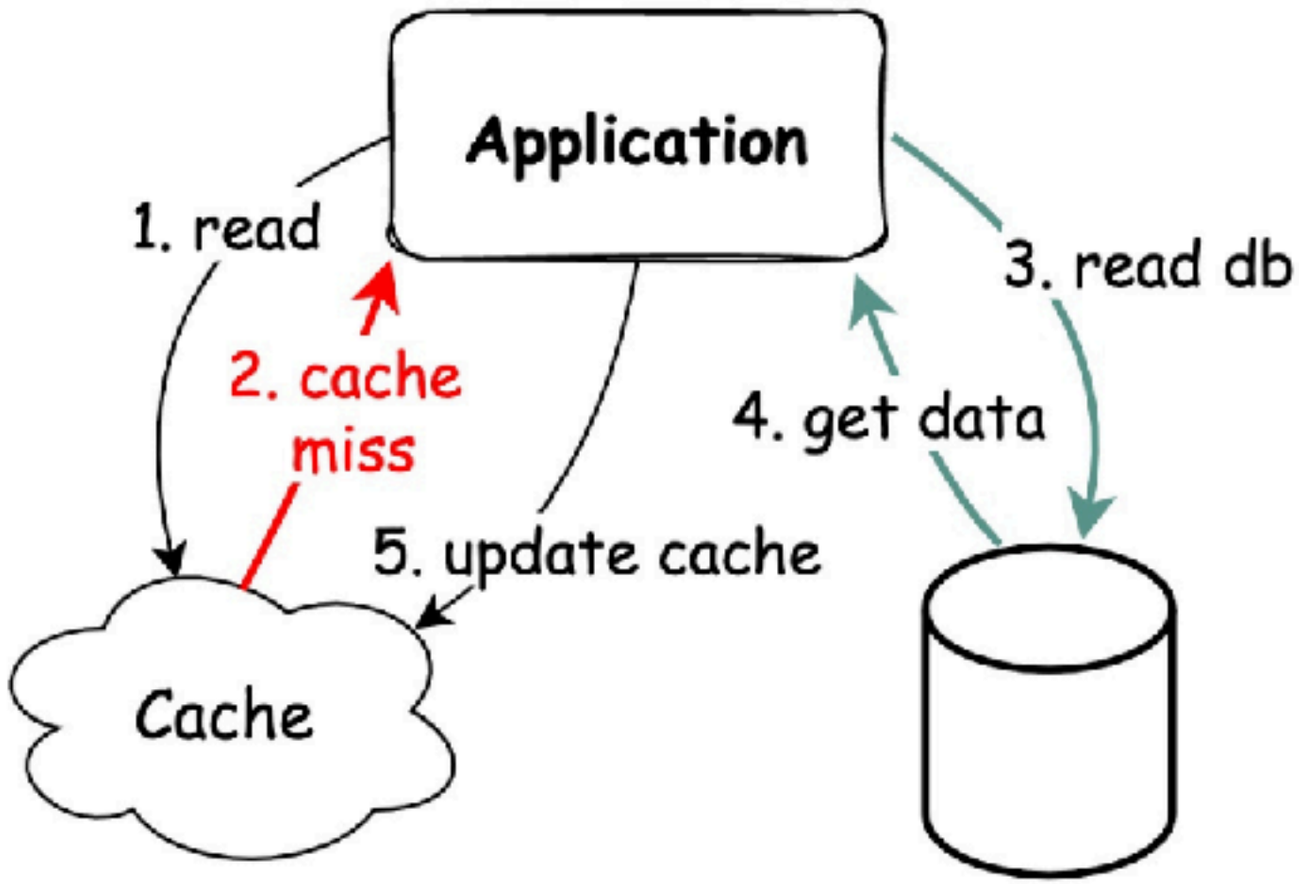


Caching Strategies ?

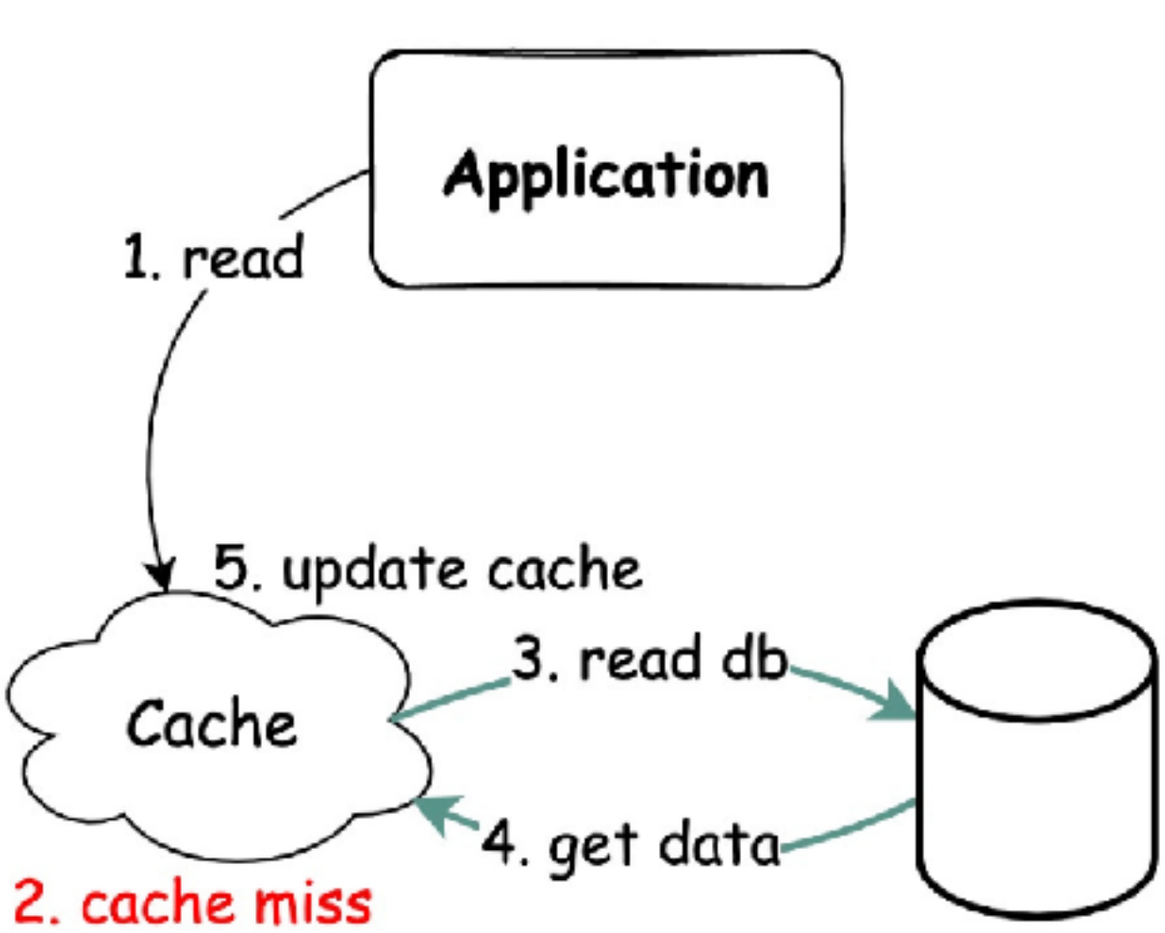
Read

Write

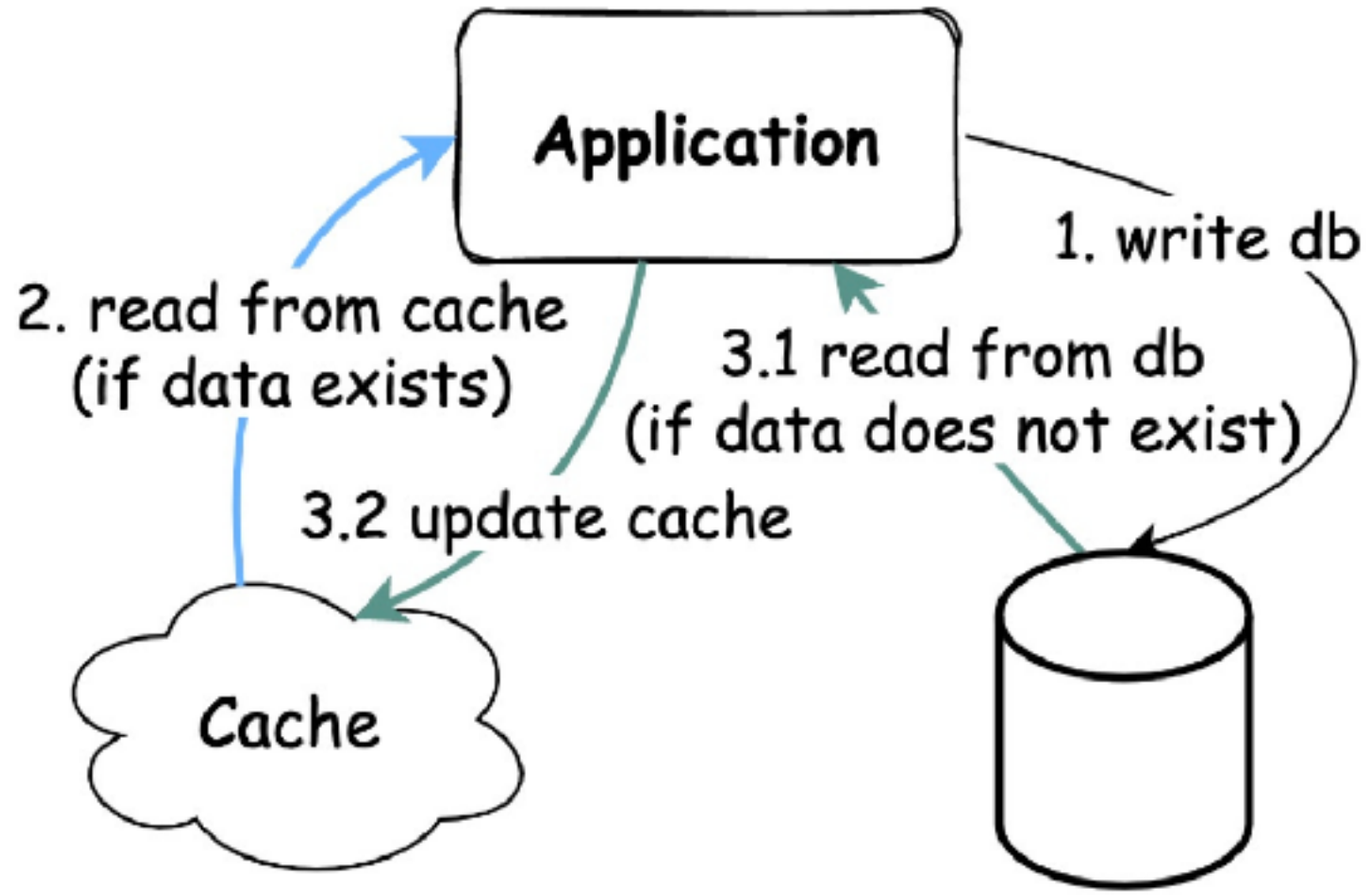
Read strategy - Cache Aside



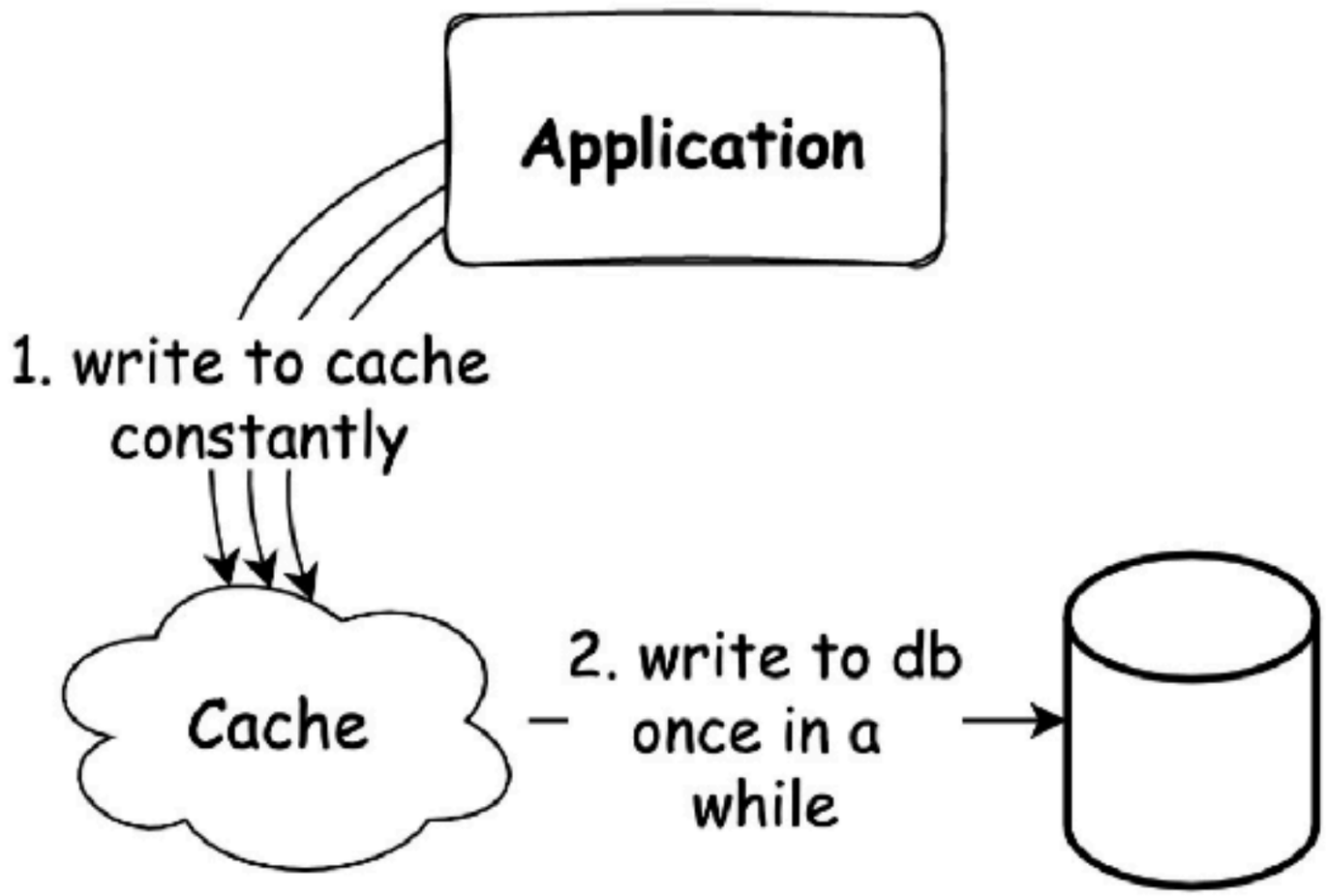
Read strategy - Read through



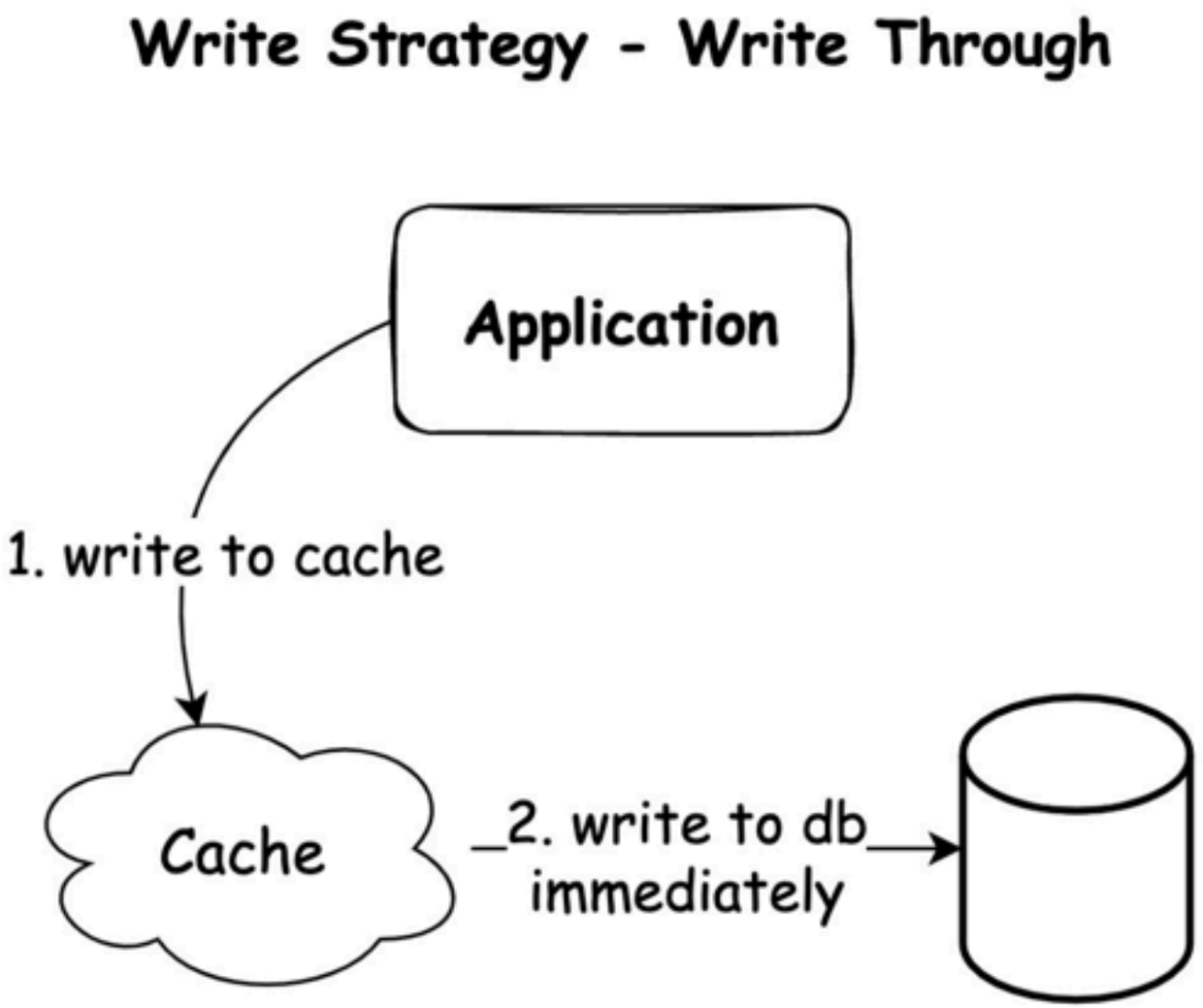
Write strategy - Write around



Write strategy - Write back



Write strategy - Write through

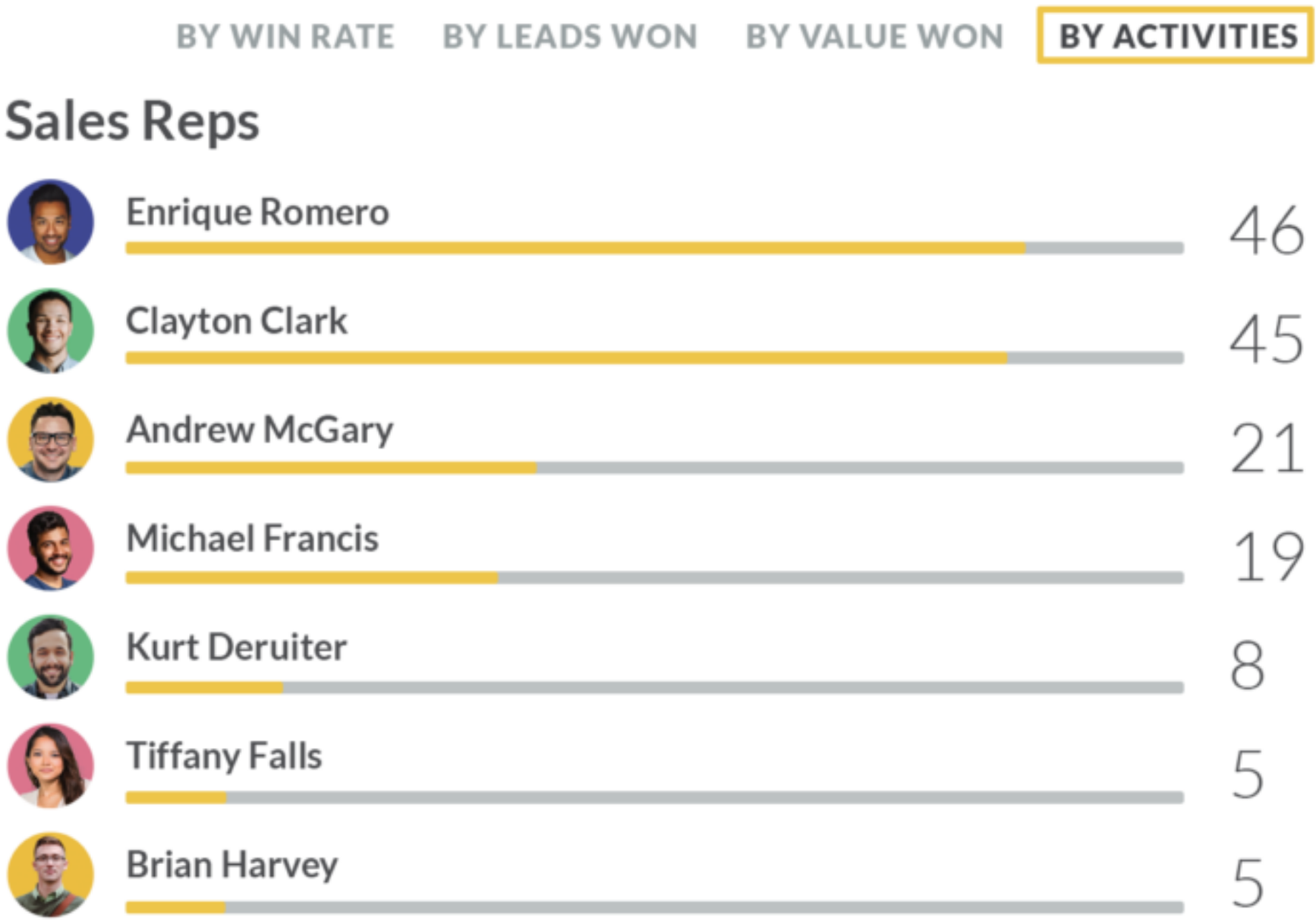


https://redis.io/learn/howtos/solutions/caching-architecture/write-through

More Use Cases

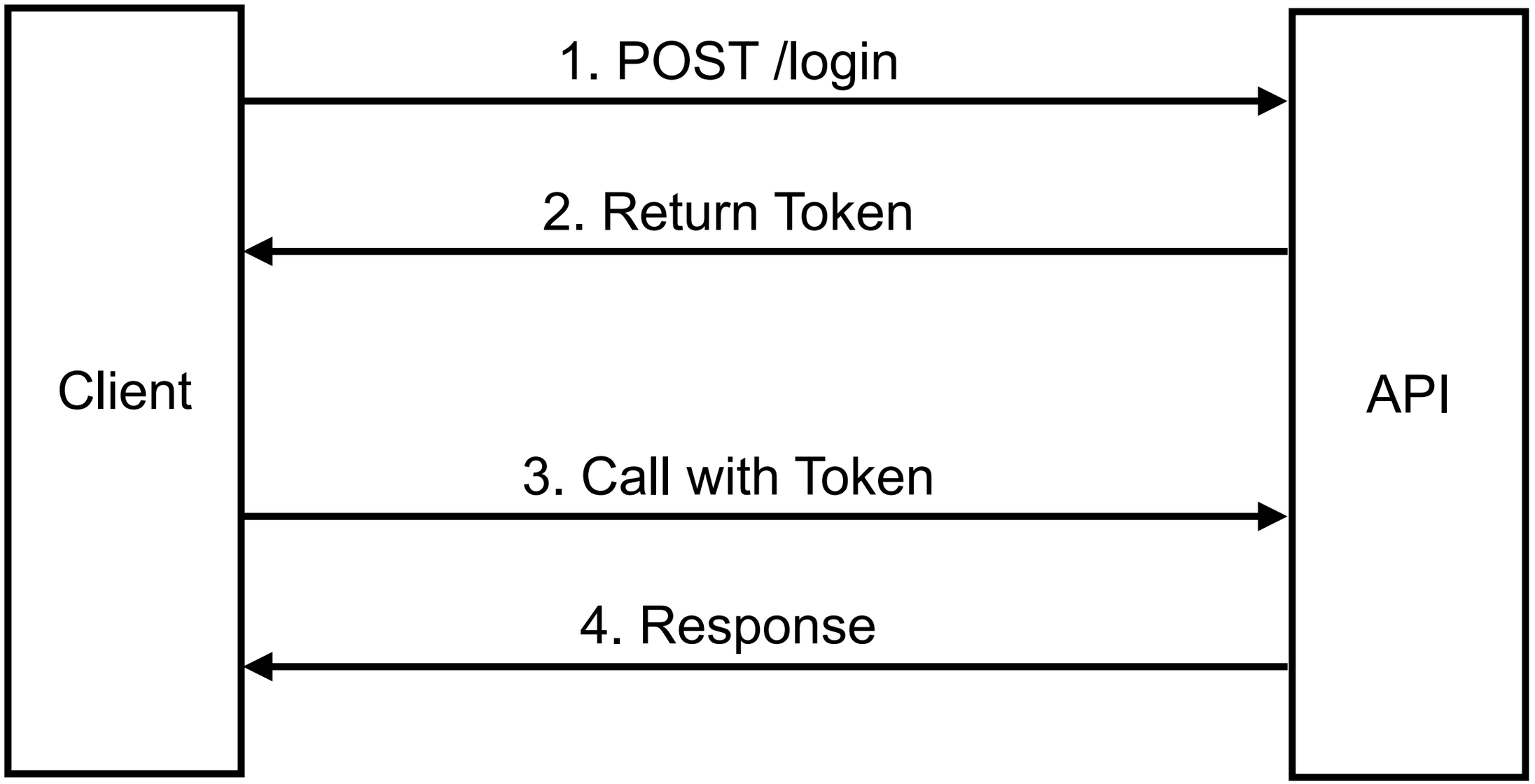
Session or Token data
Message broker (pub/sub)
Job queue
Realtime analytic
Fraud detection
Gaming

Sales leaderboard

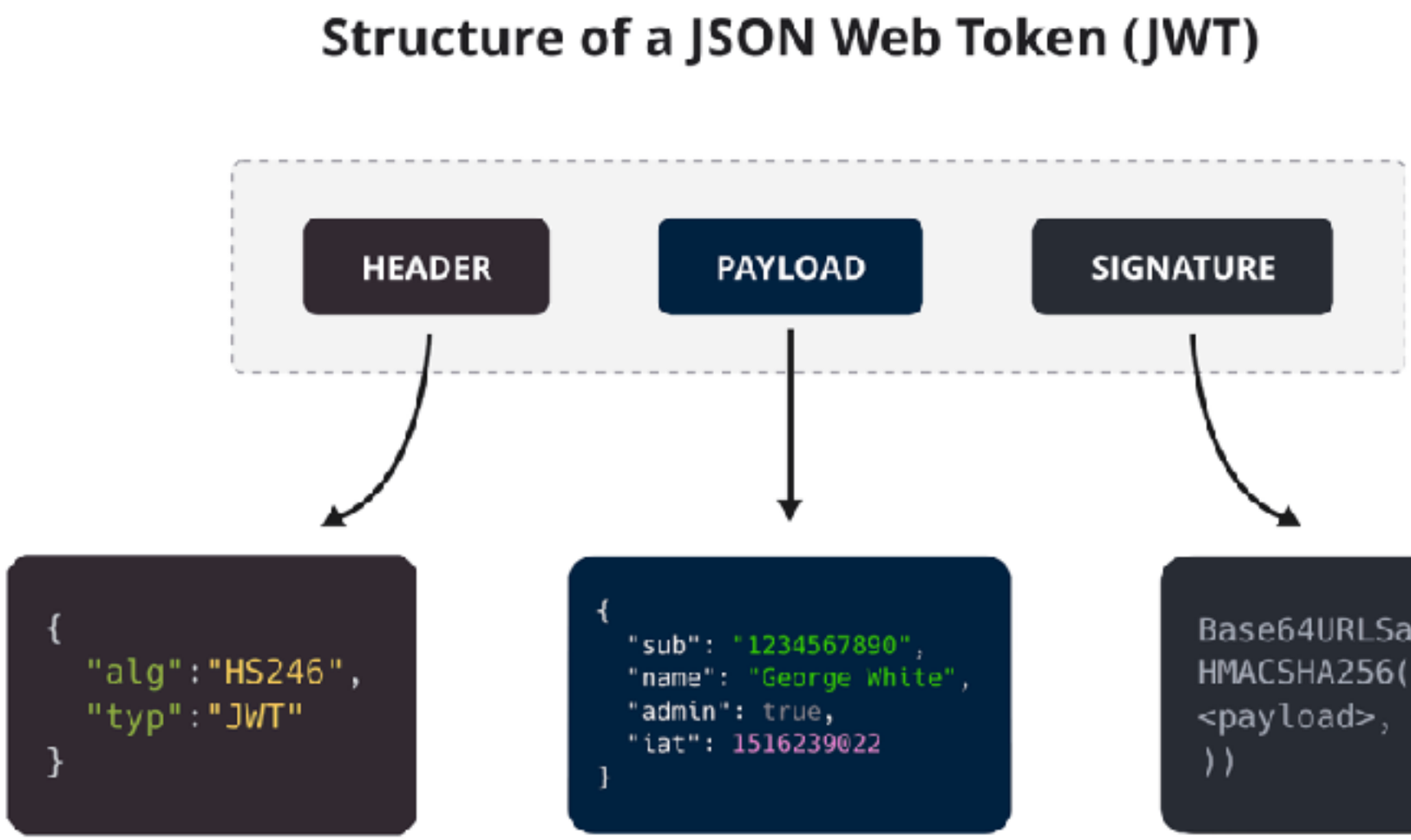


Authentication
Authorization

Simple Flow



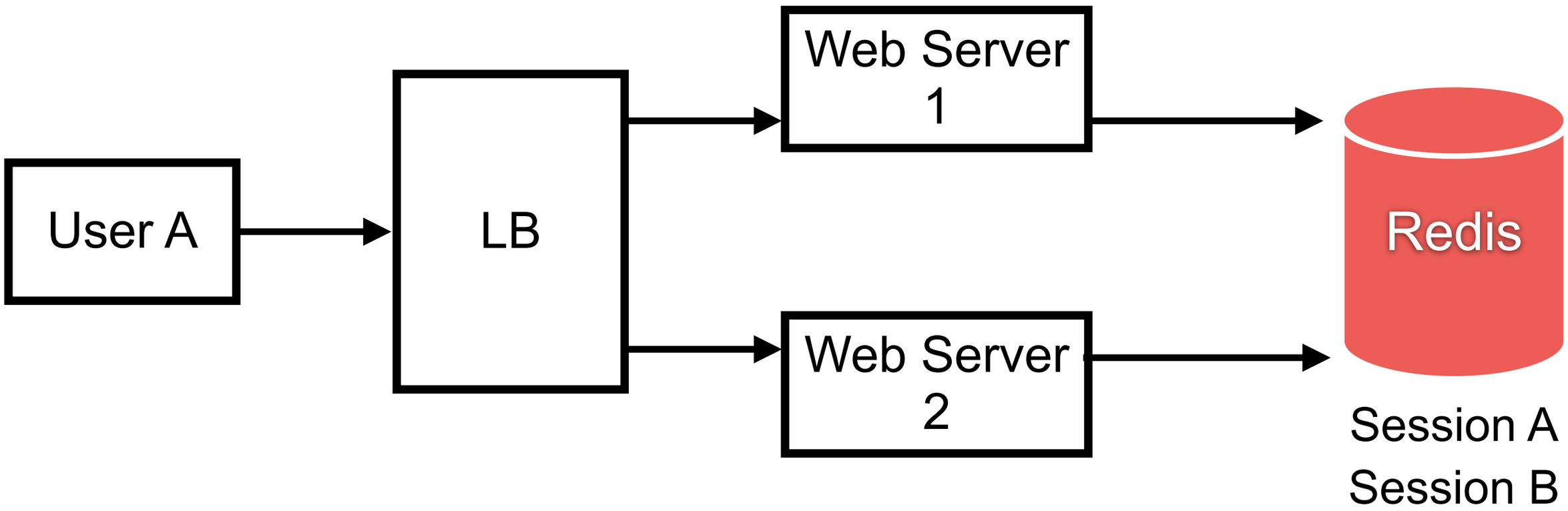
JWT Token



https://jwt.io/

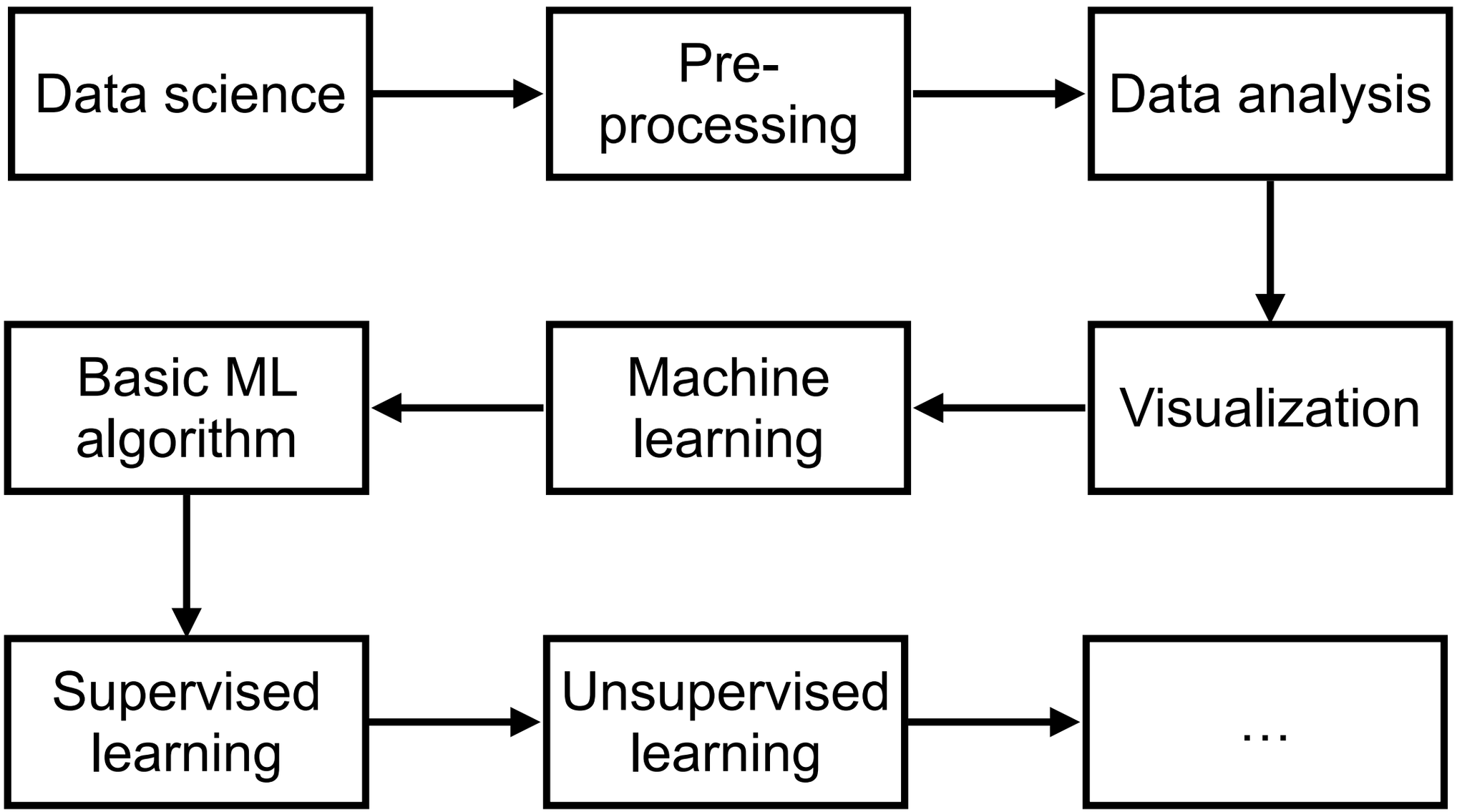
Session or Token data

Data with expire time (automatically delete)
key=value(String)



Part 3

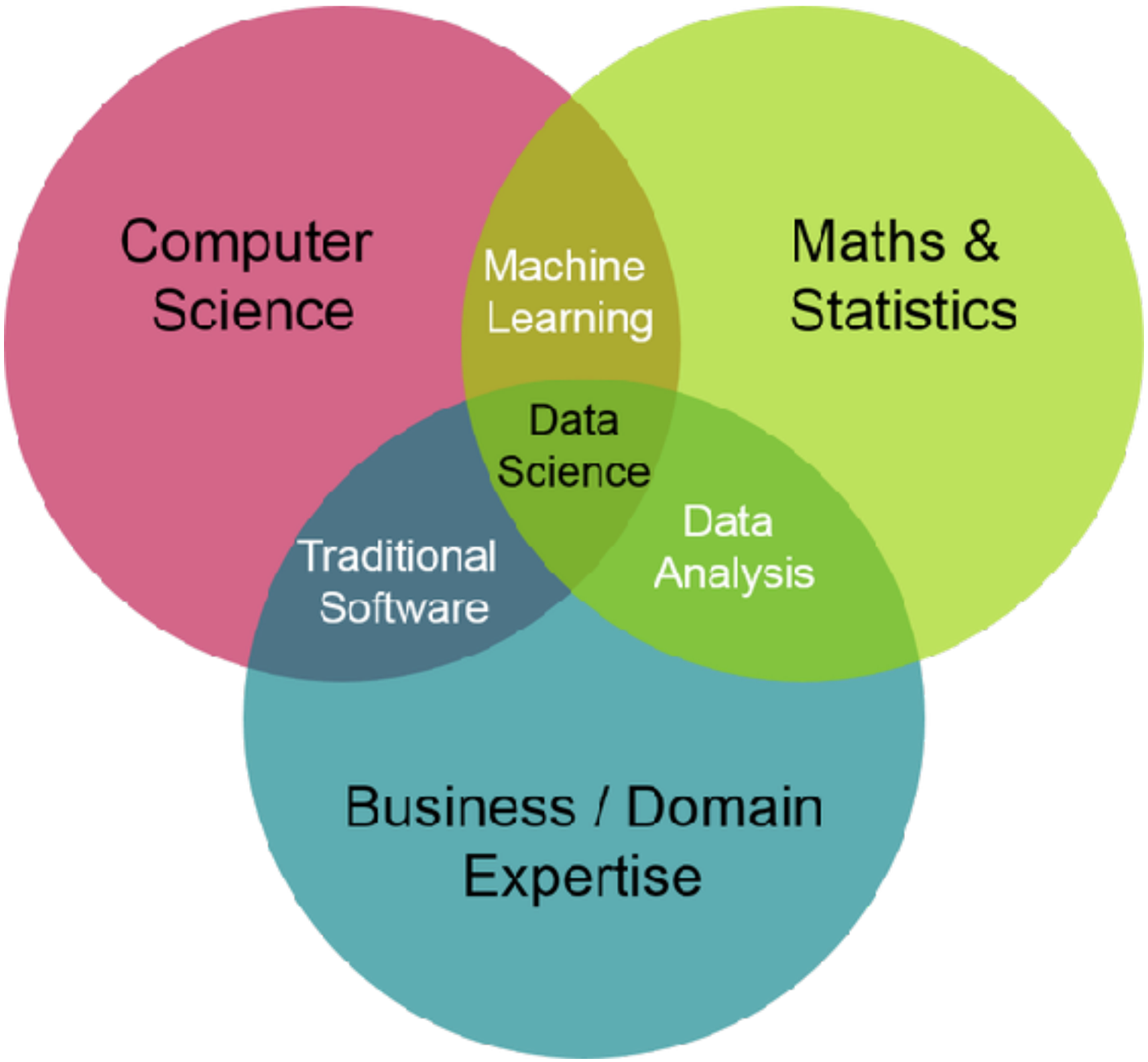
Learning Path 3



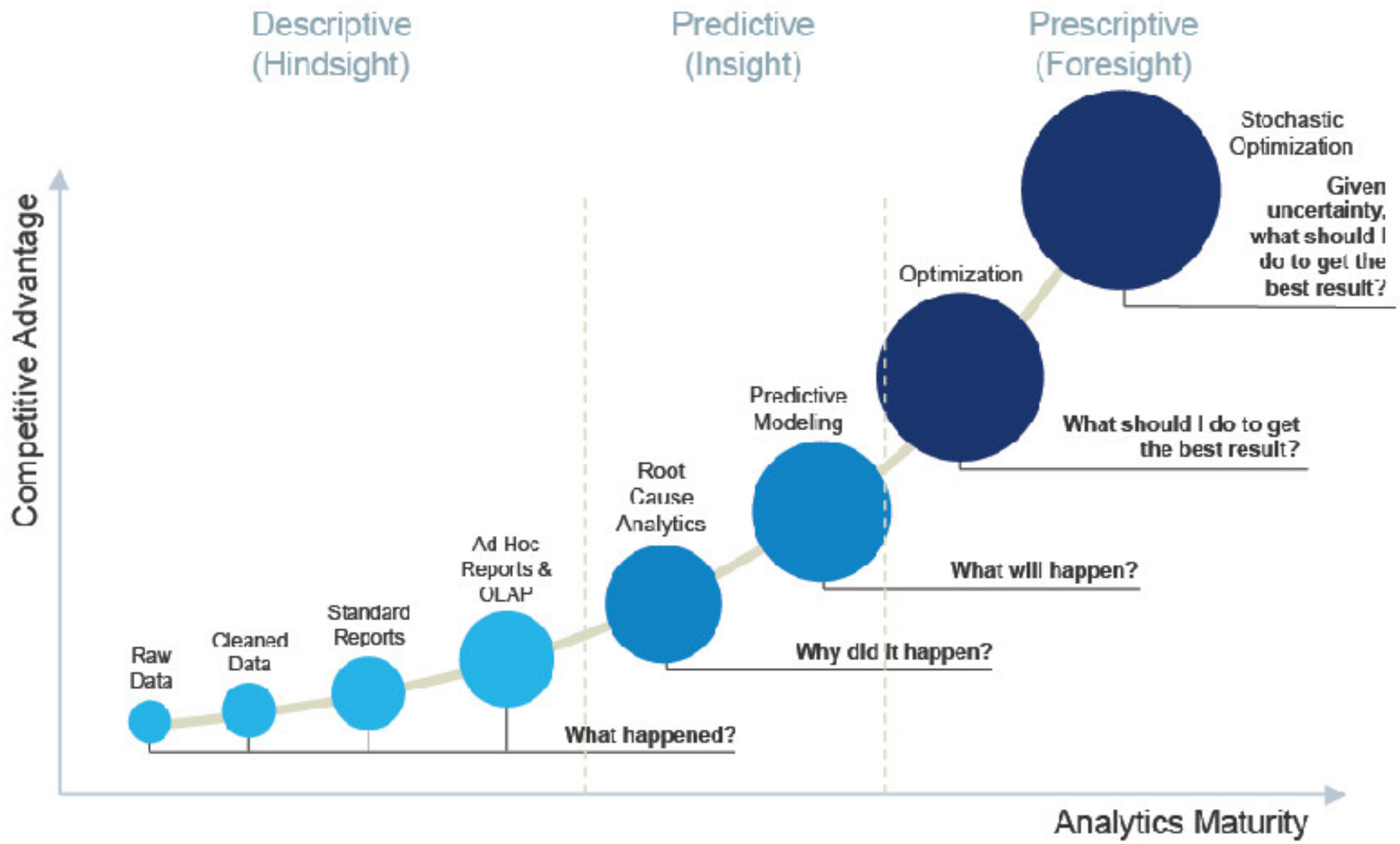
Data Science

The study of formulating and rigorously answering questions using **data-centric process** that emphasizes clarity, reproducibility, effective communication and ethical practices

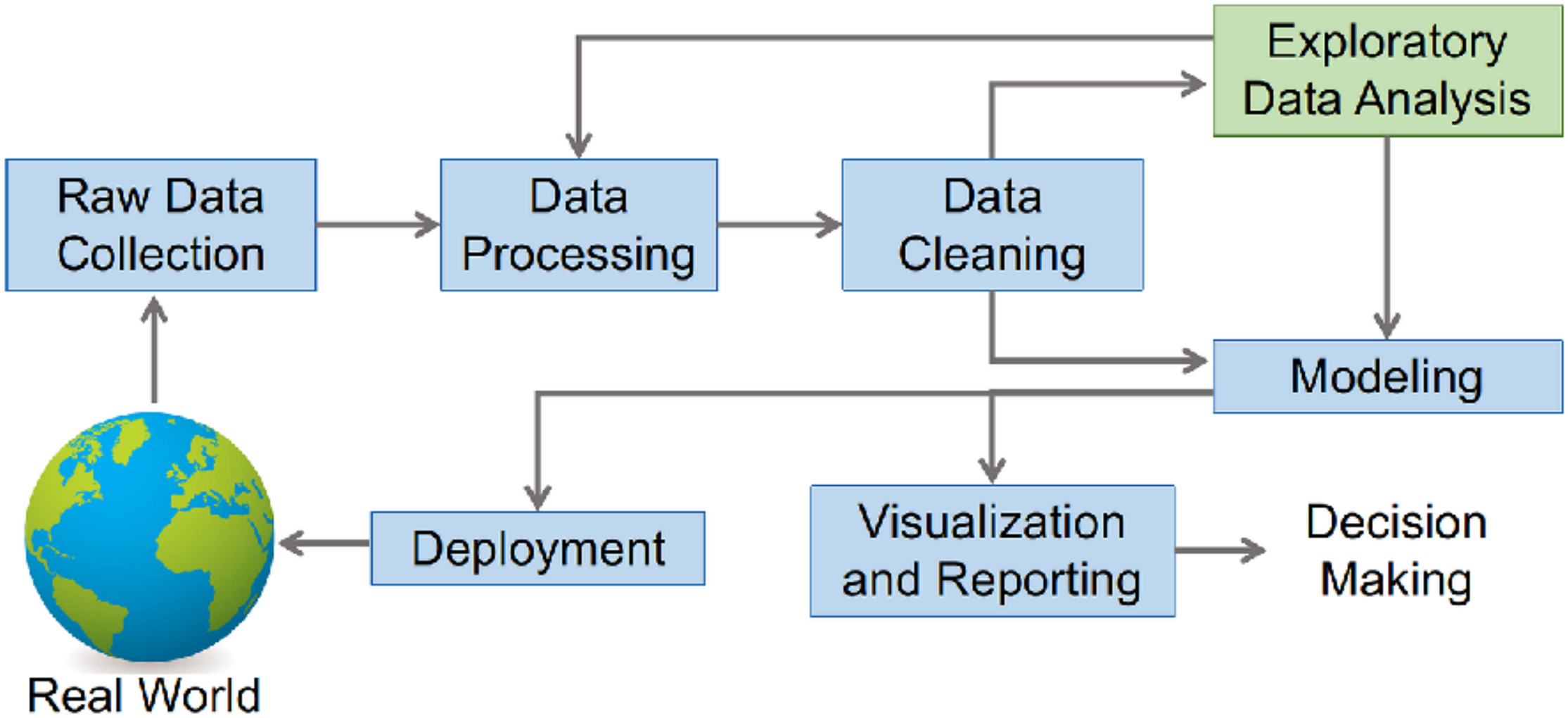
Data Science



Data Science

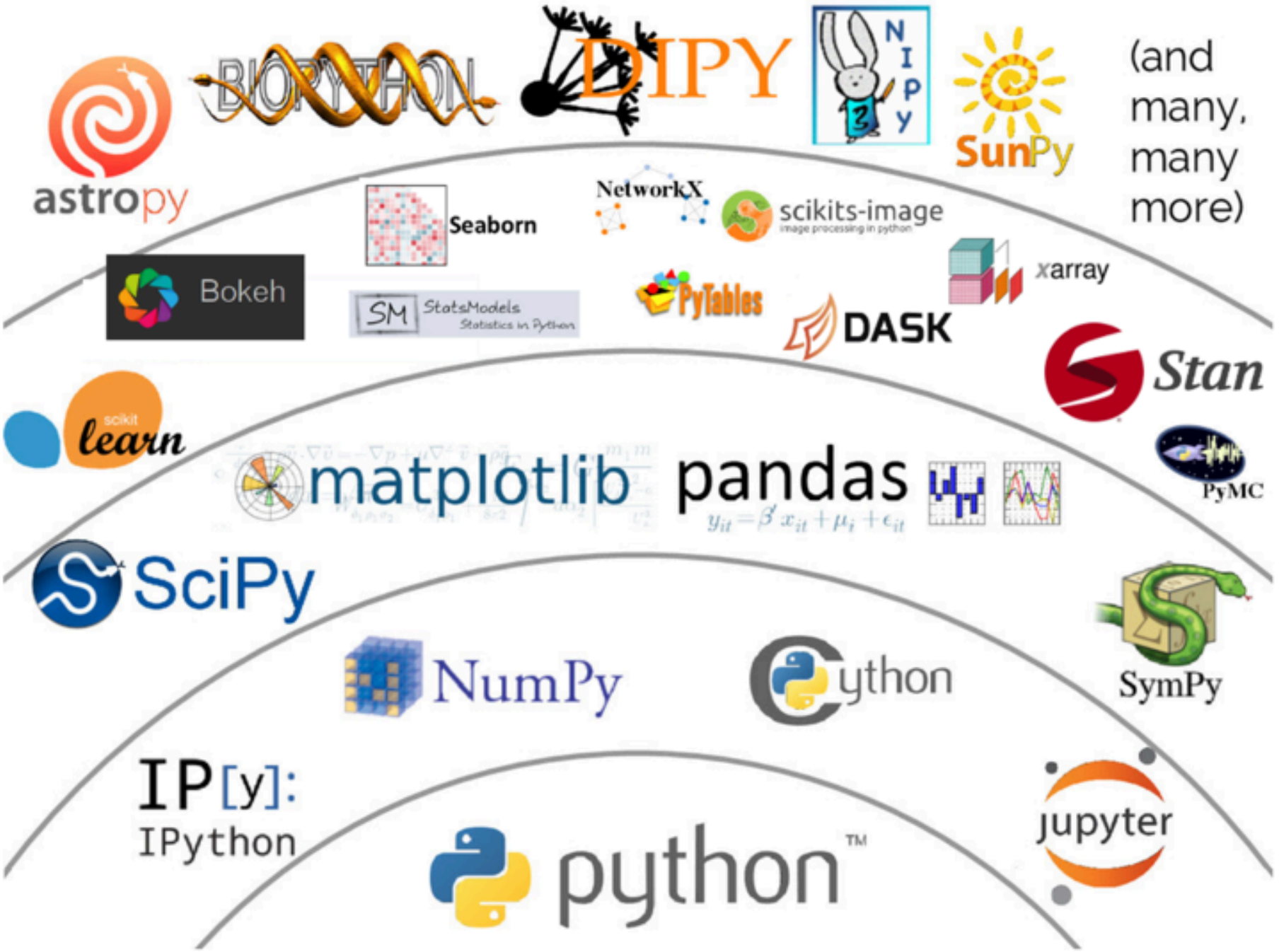


Data Science Process



https://www.markovml.com/blog/exploratory-data-analysis

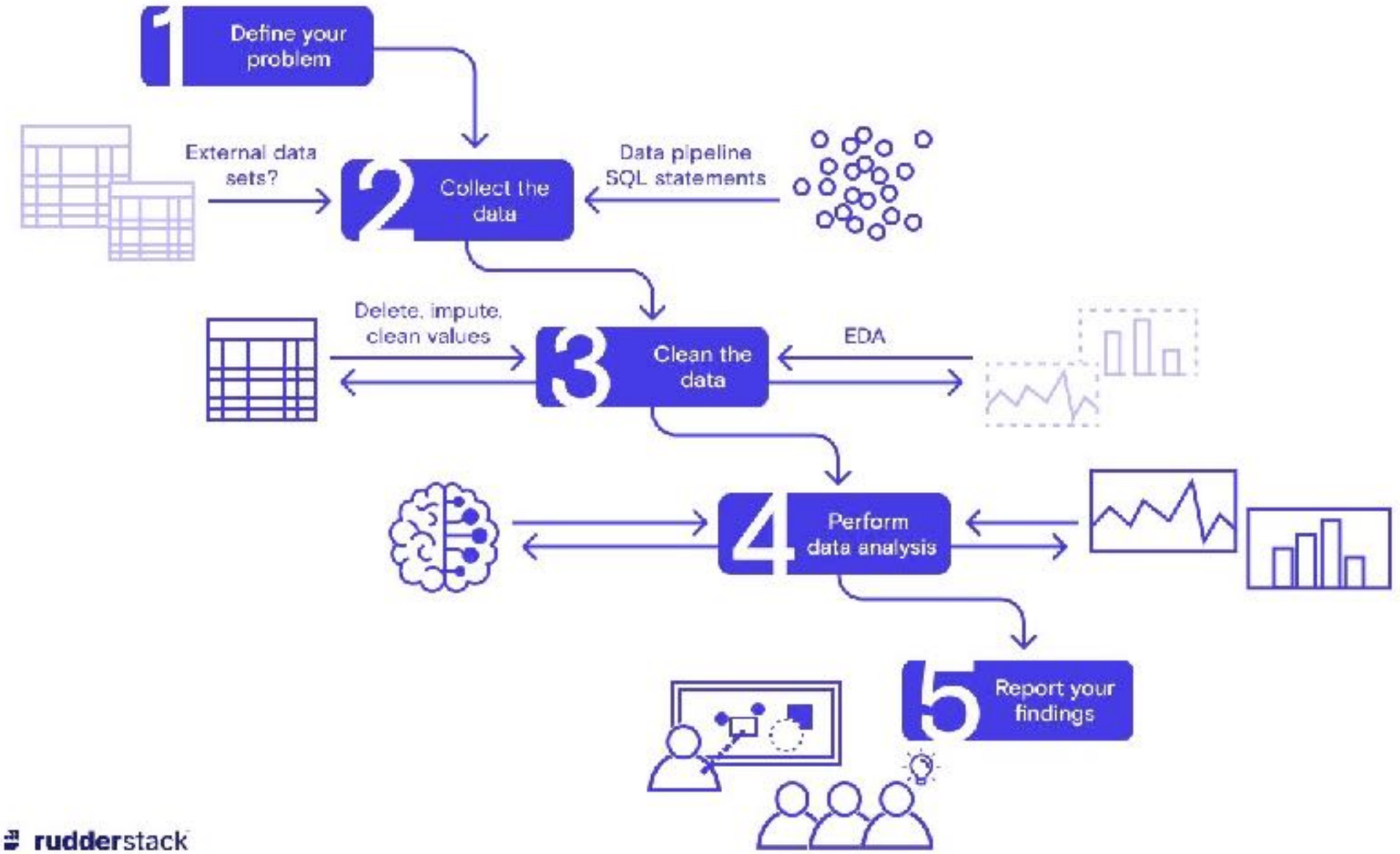
Data Science in Python



Data Analysis

The process of inspecting, cleaning, transforming, and modeling data to discover useful information, draw conclusions, and **support decision-making**

Data analysis process



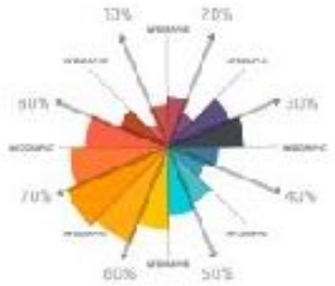
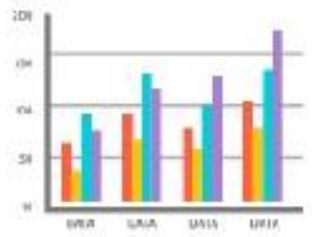
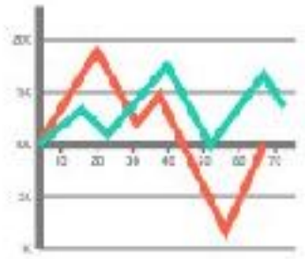
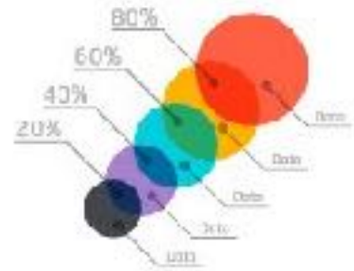
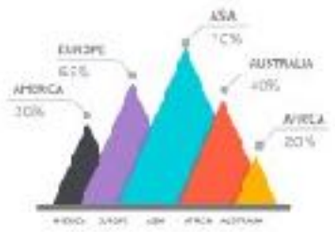
https://www.rudderstack.com/learn/data-analytics/data-analytics-processes/

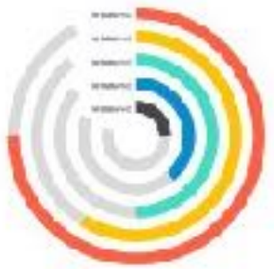
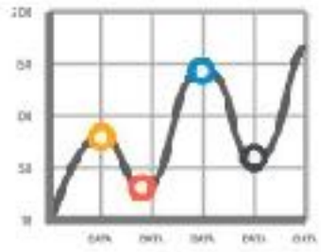
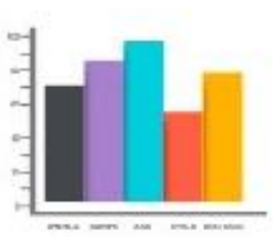
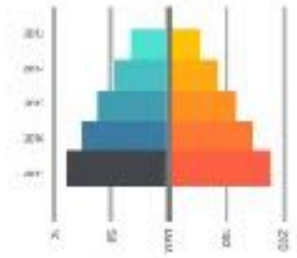
Exploratory Data Analysis

Statistical approach that uses visual
and numerical methods to **understand**
the main characteristics of a dataset, often before
applying more advanced analysis techniques

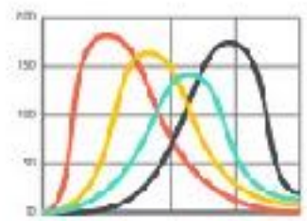
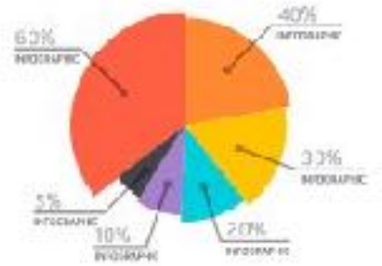
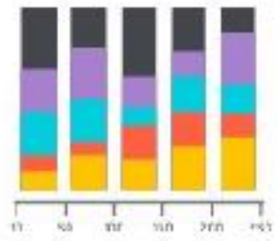
Python for Data analysis

Visualization with Python

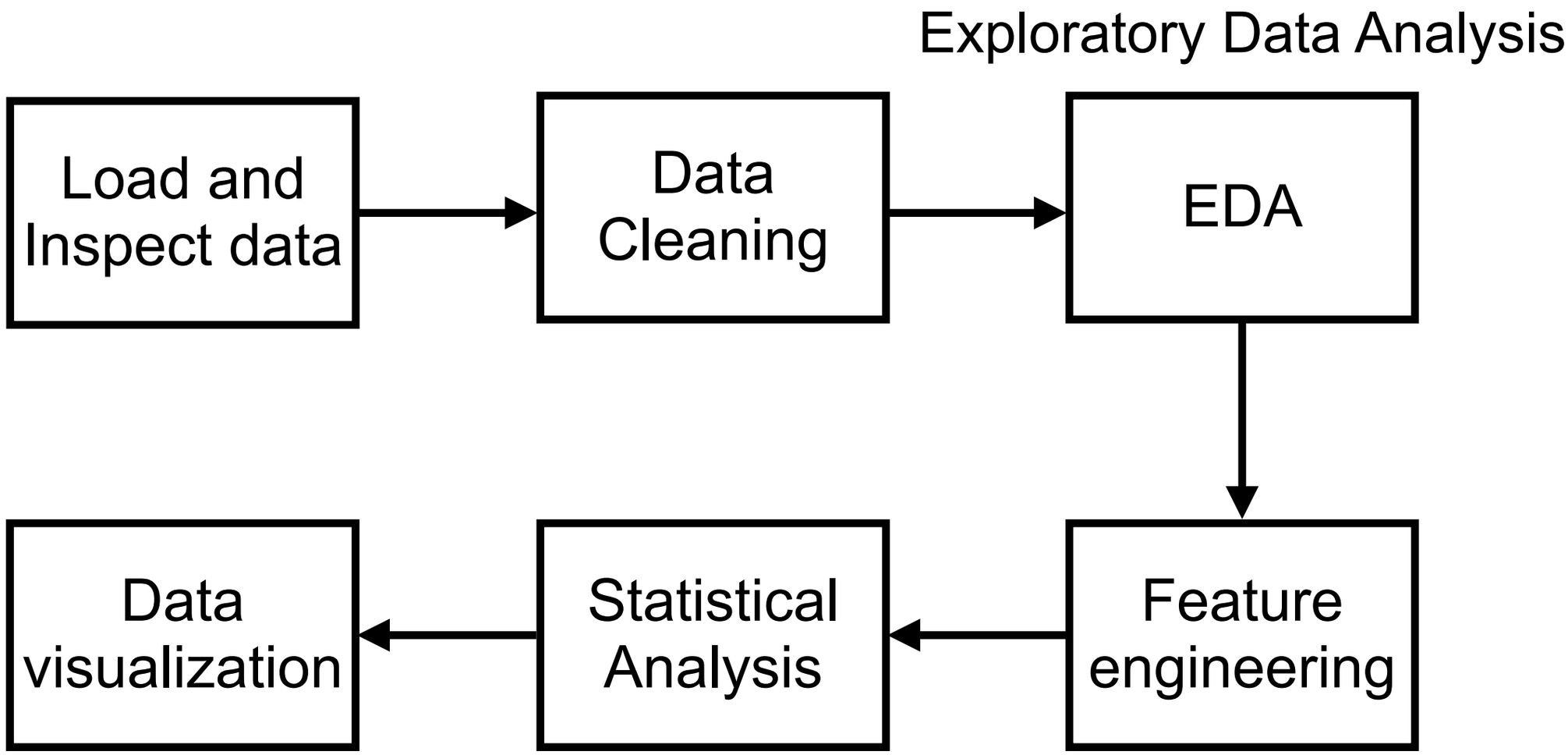







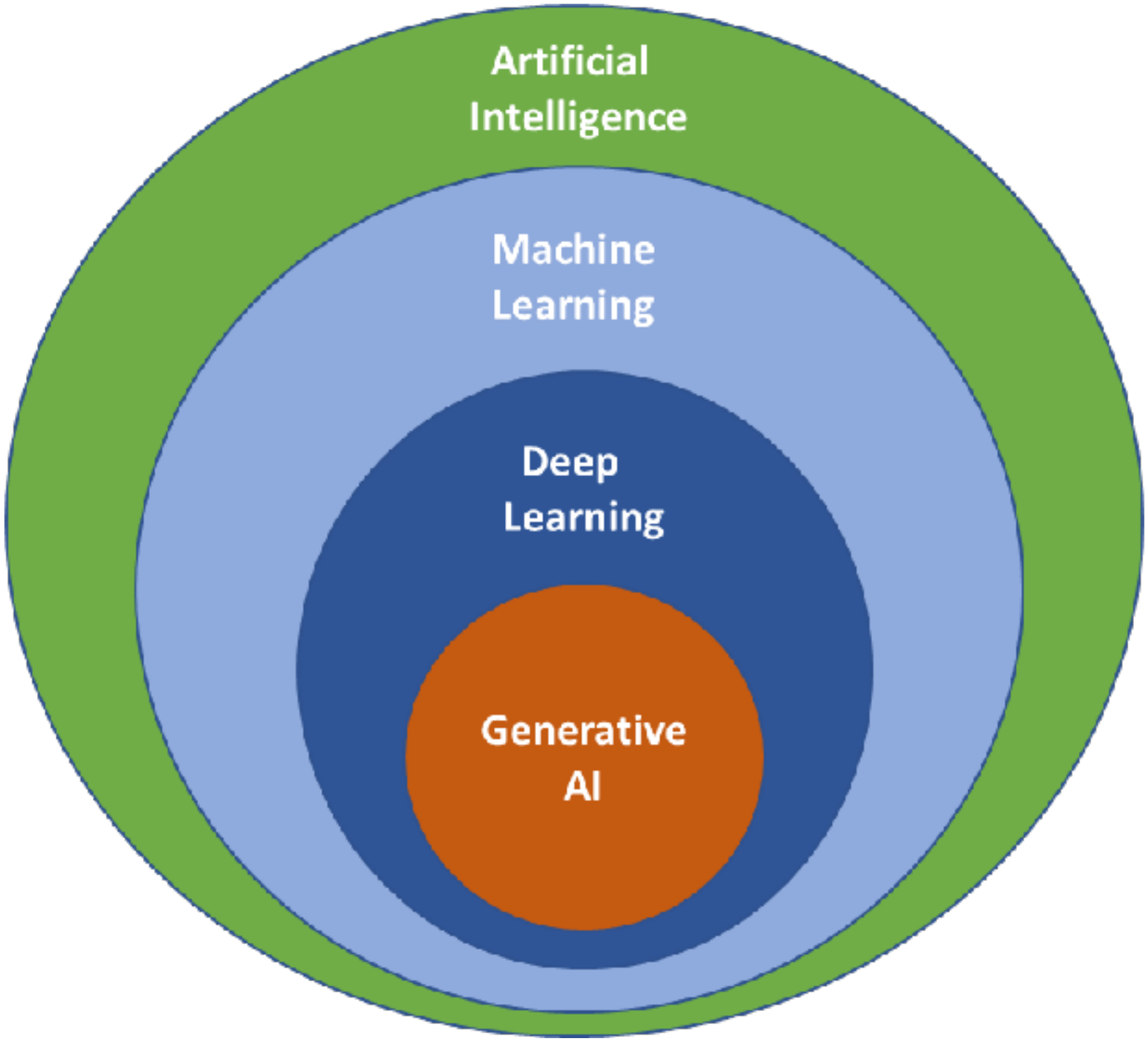
https://matplotlib.org/

Workshop with Data Analysis



Machine Learning

Machine Learning ?



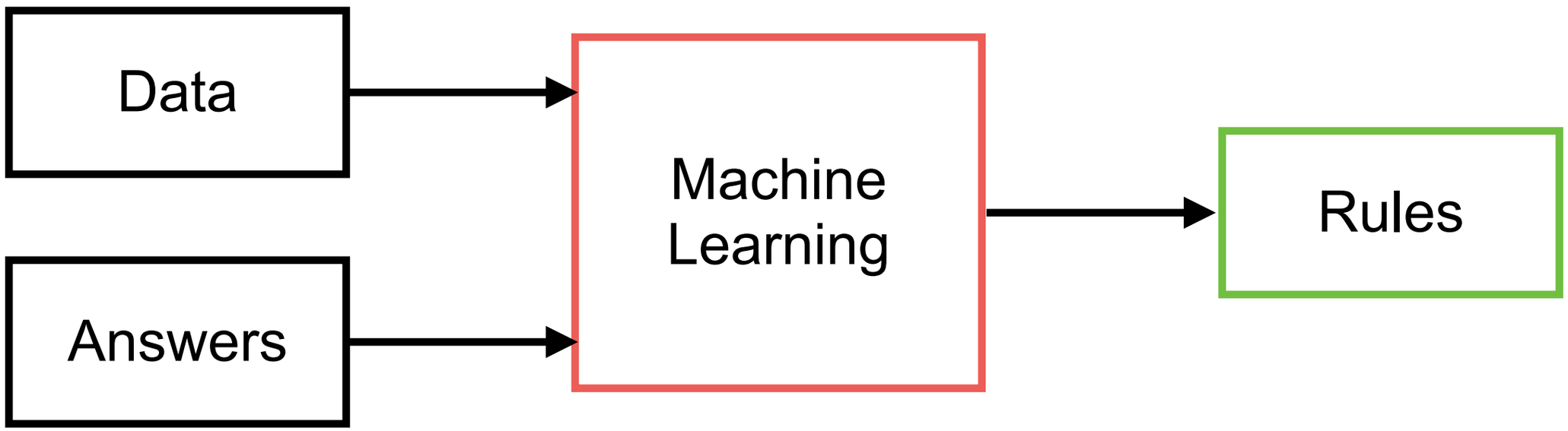
Artificial Intelligence (AI)

The effort to automate intellectual tasks normally performed by humans

Simple and complex tasks

Machine Learning (ML)

A subset of AI that can learn by themselves
ML models take in data and fit data to algorithm



ML Use Cases

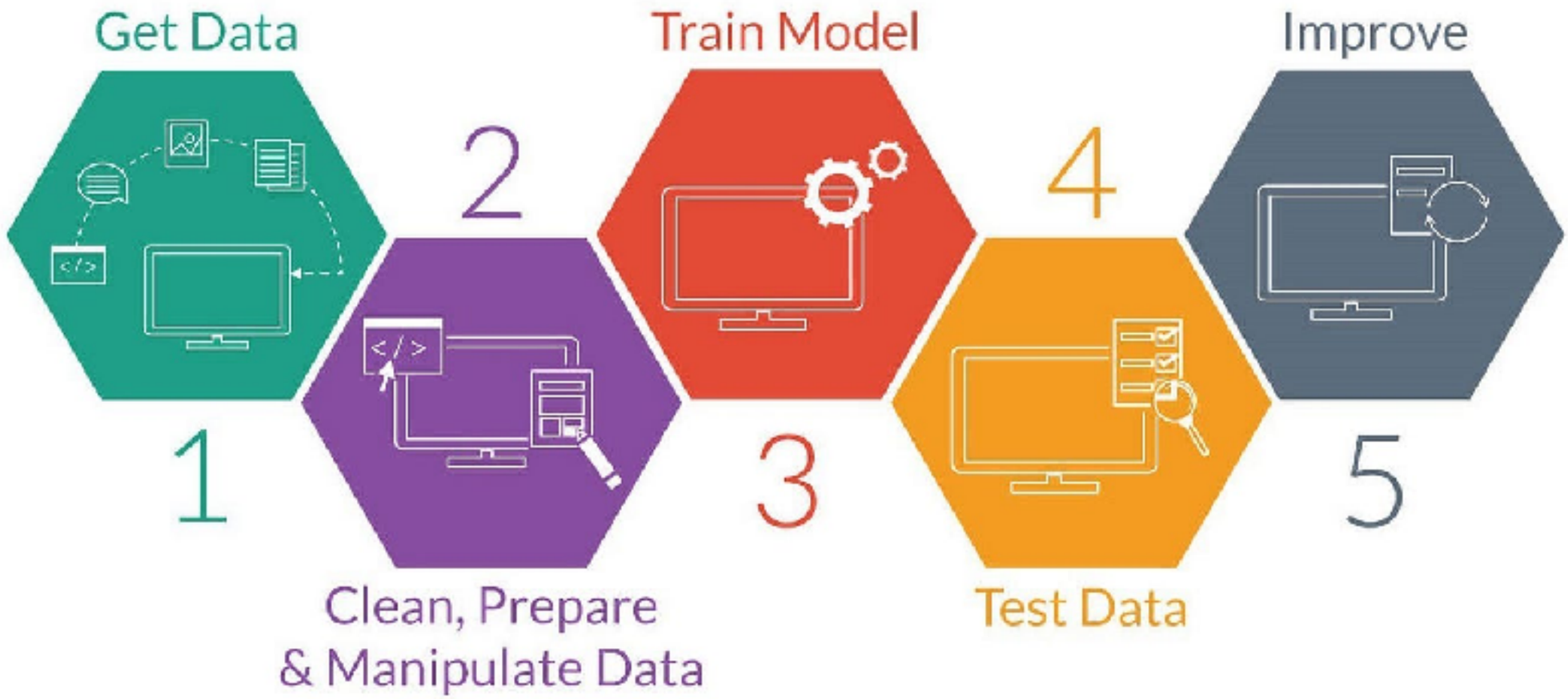
Image recognition

Natural Language Processing

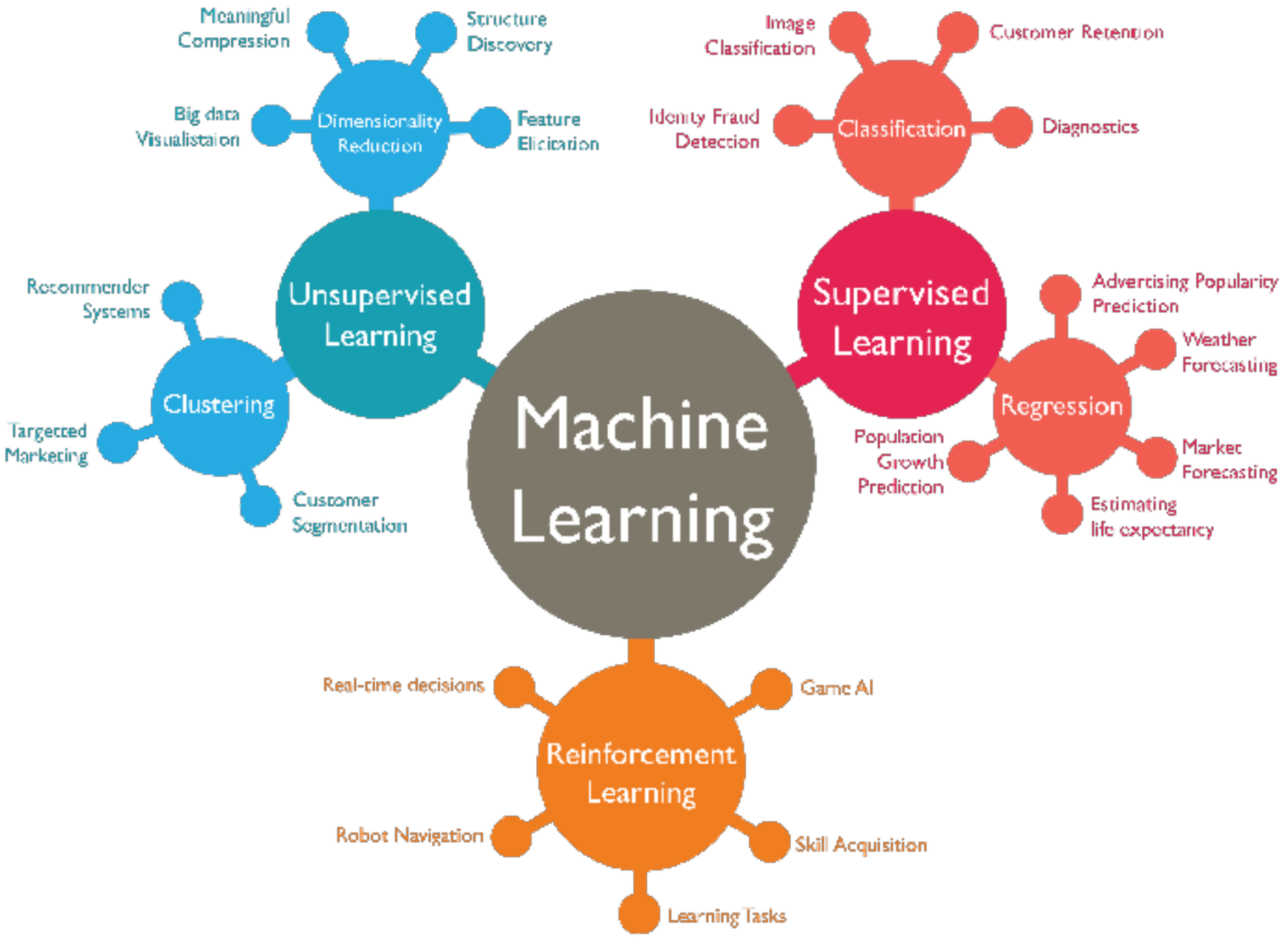
Recommendation System

Predictive Maintenance

Machine learning workflow



Machine Learning



https://www.datacamp.com/blog/what-is-machine-learning

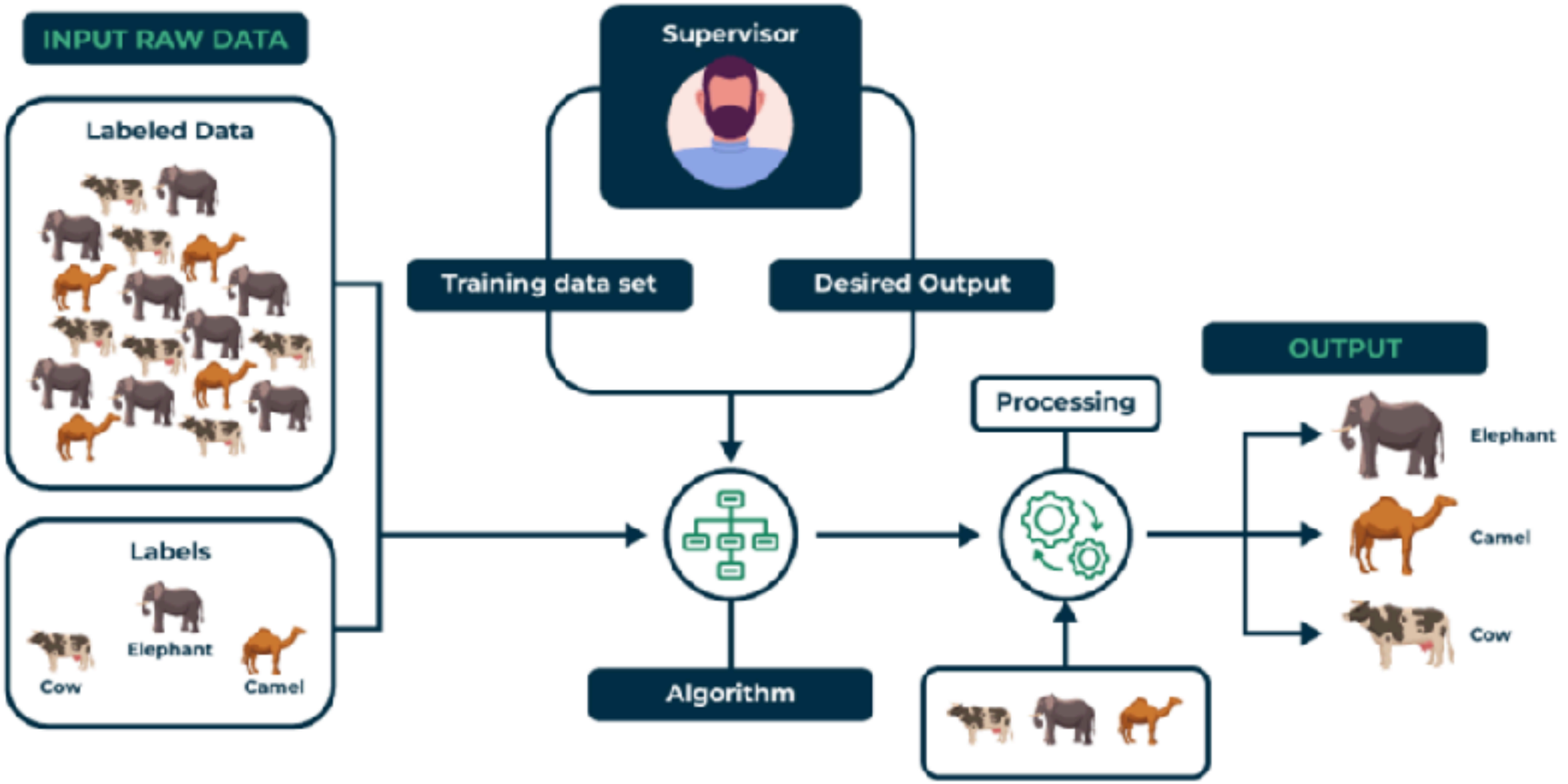
Supervised Learning

Trains models on labeled data to predict
or classify new, unseen data.

Classification

Regression

Supervised Learning



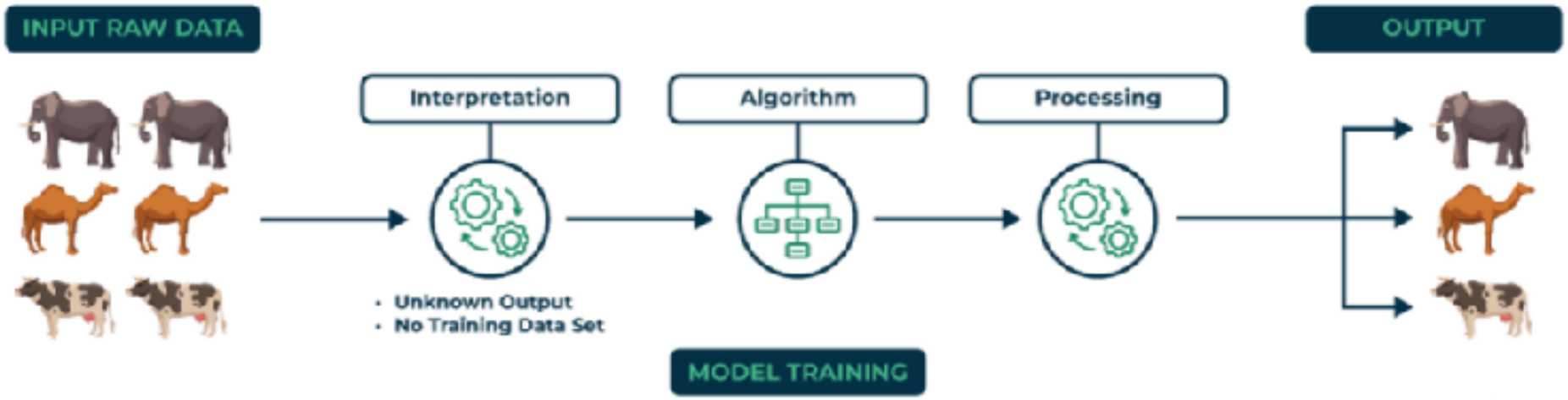
https://www.geeksforgeeks.org/machine-learning/

Unsupervised Learning

Finds patterns or groups in unlabeled data, like clustering or dimensionality reduction.

- Clustering
- Association Rule
- Dimensionality Reduction

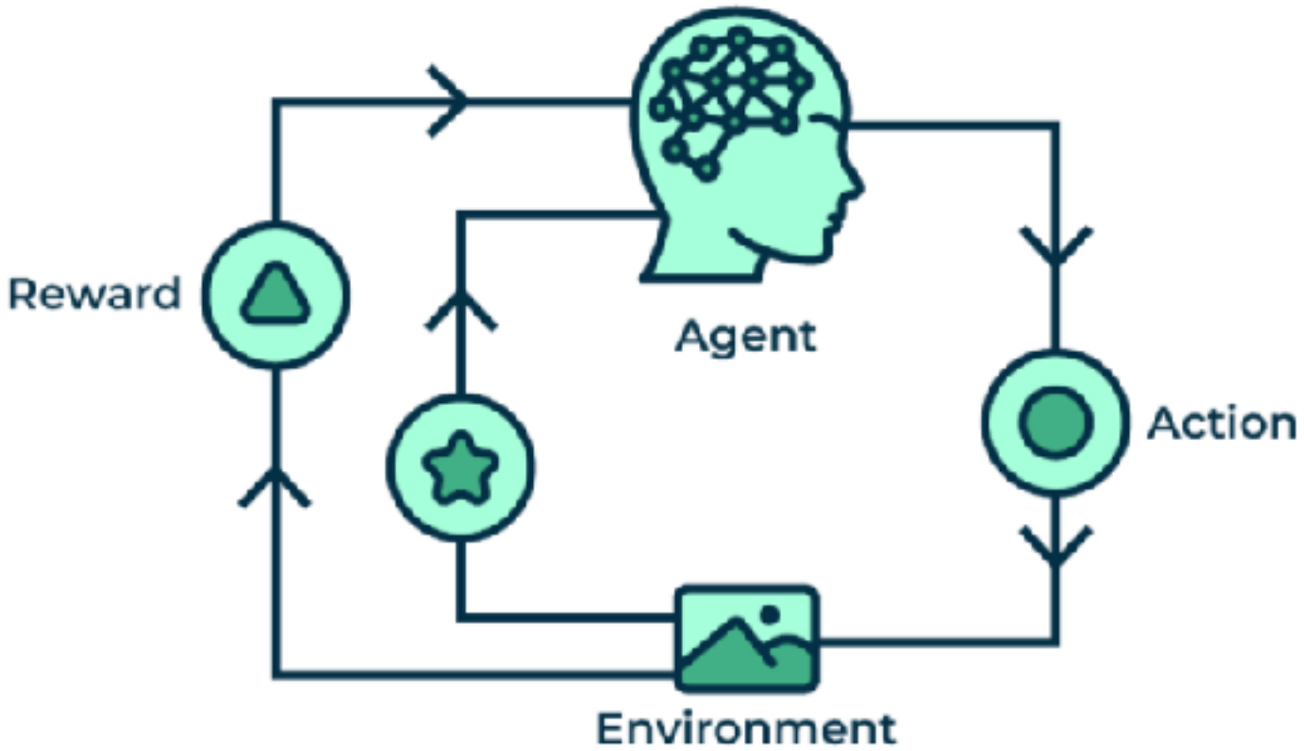
Unsupervised Learning



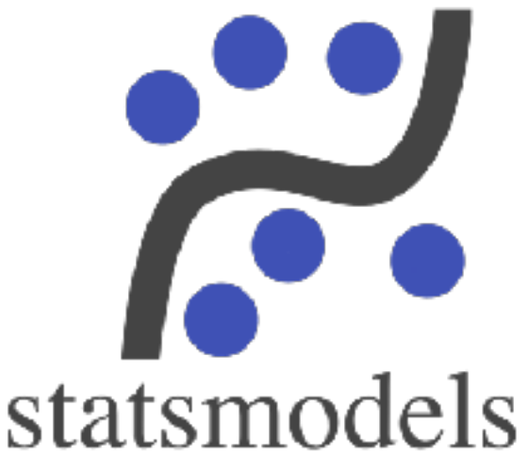
https://www.geeksforgeeks.org/machine-learning/

Reinforcement Learning

Learns through trial and error to maximize rewards, ideal for decision-making tasks.



Python for Machine learning



Let’s go !!