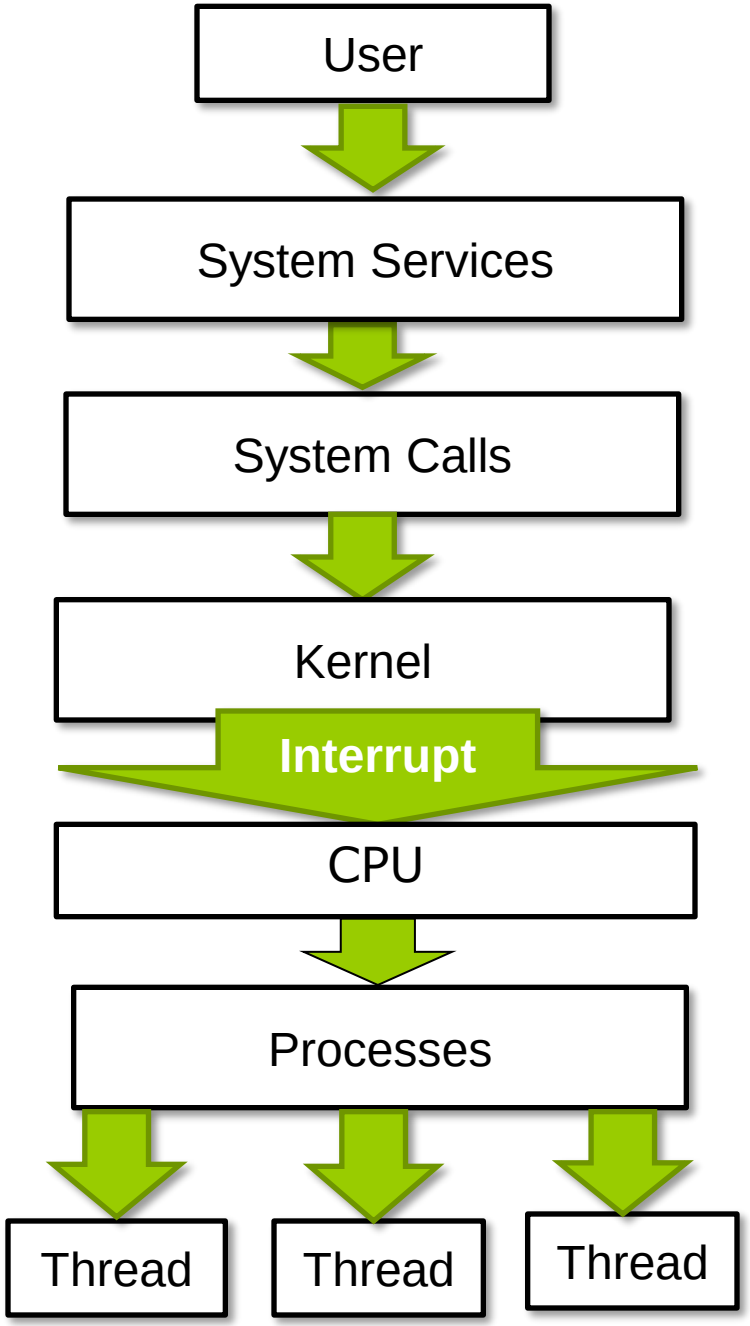
Part 8

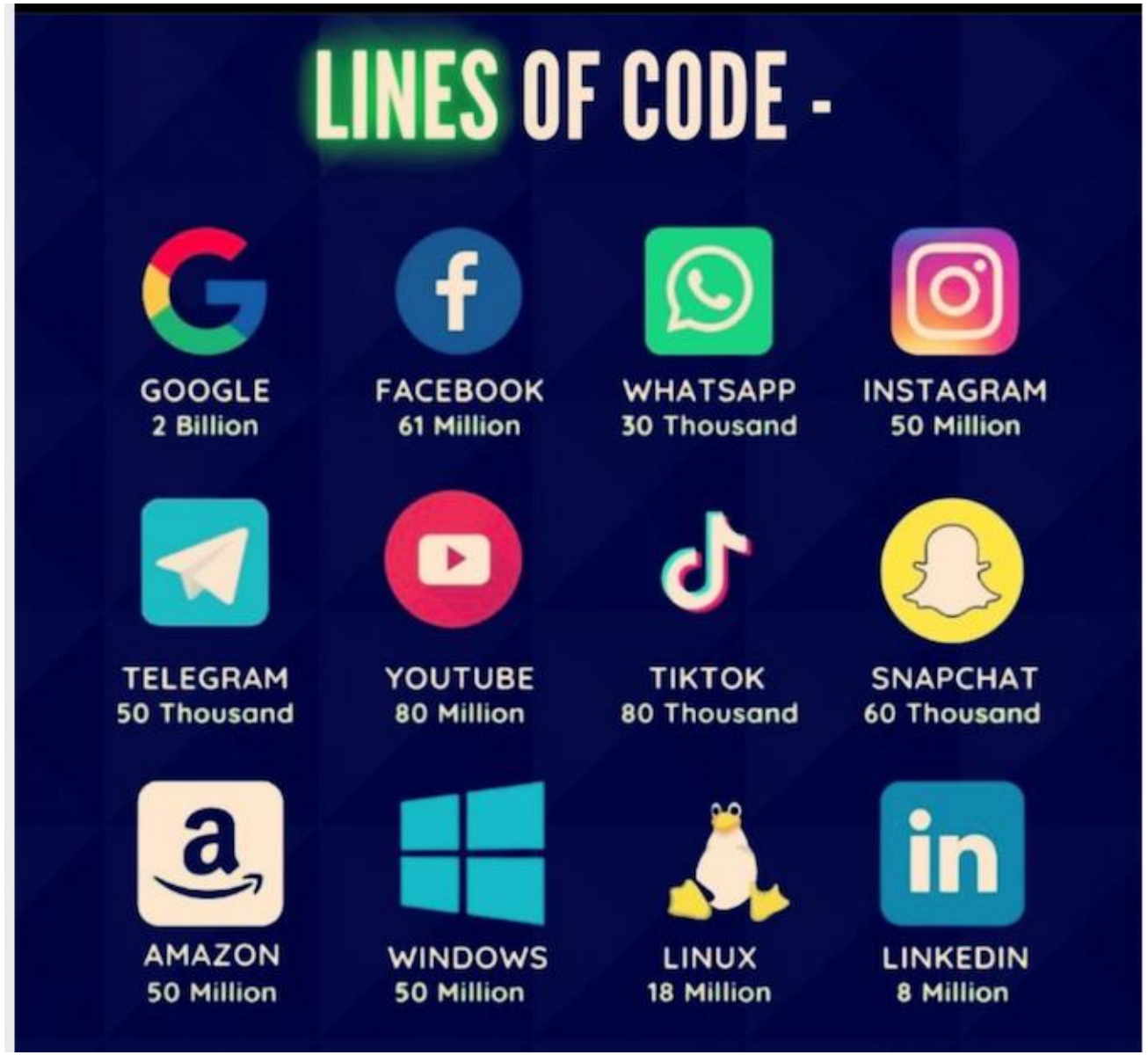
The Big Pic of OS



a simple operating system kernel might look like in C:

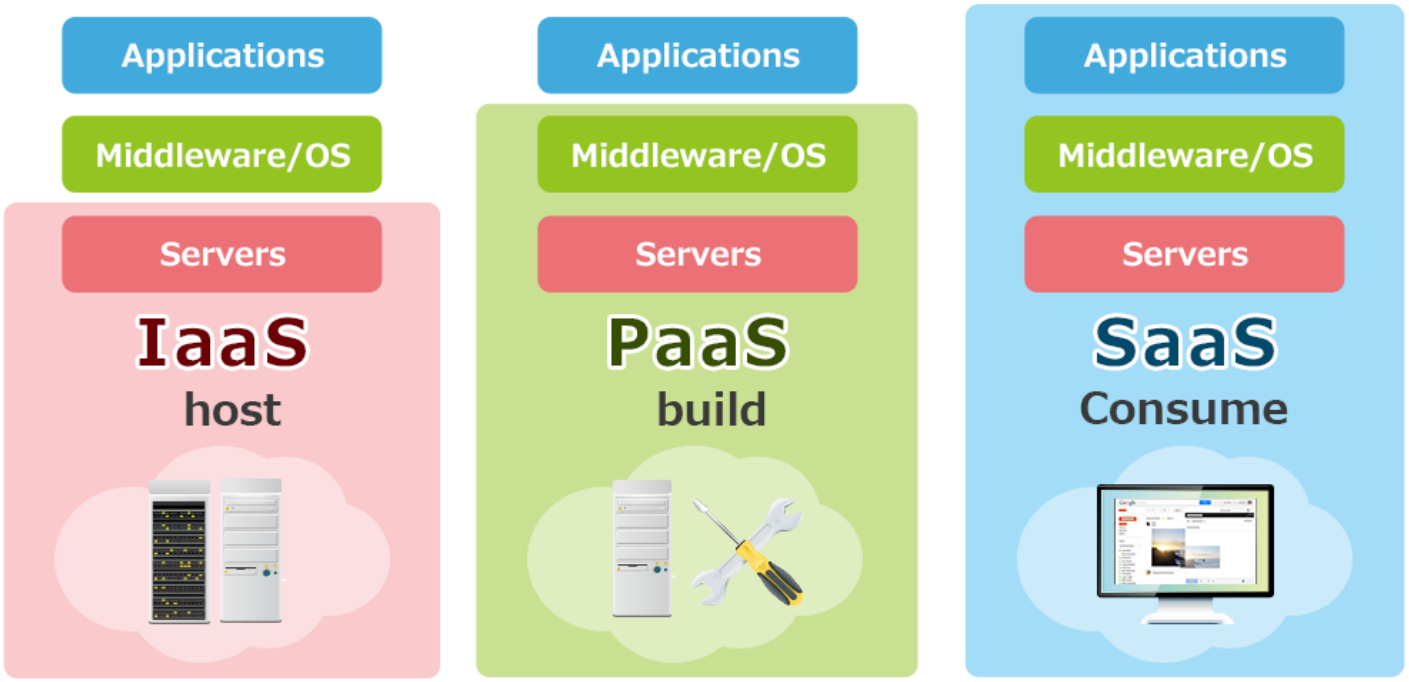
```
scss
#include <stdio.h>
#include <stdlib.h>

void kernel_main(void) {
    /* Initialize hardware */
    init_hardware();

    /* Initialize memory */
    init_memory();

    /* Initialize process manager */
    init_process_manager();

    /* Initialize file system */
    init_file_system();

    /* Start the shell */
    start_shell();
}

void init_hardware(void) {
    /* Initialize hardware devices, such as keyboard, display, and disk */
}

void init_memory(void) {
    /* Allocate memory for the kernel and user processes */
}

void init_process_manager(void) {
    /* Initialize the process table and create the first user process */
}

void init_file_system(void) {
    /* Initialize the file system and mount the root file system */
}

void start_shell(void) {
    /* Run the shell process */
}
```

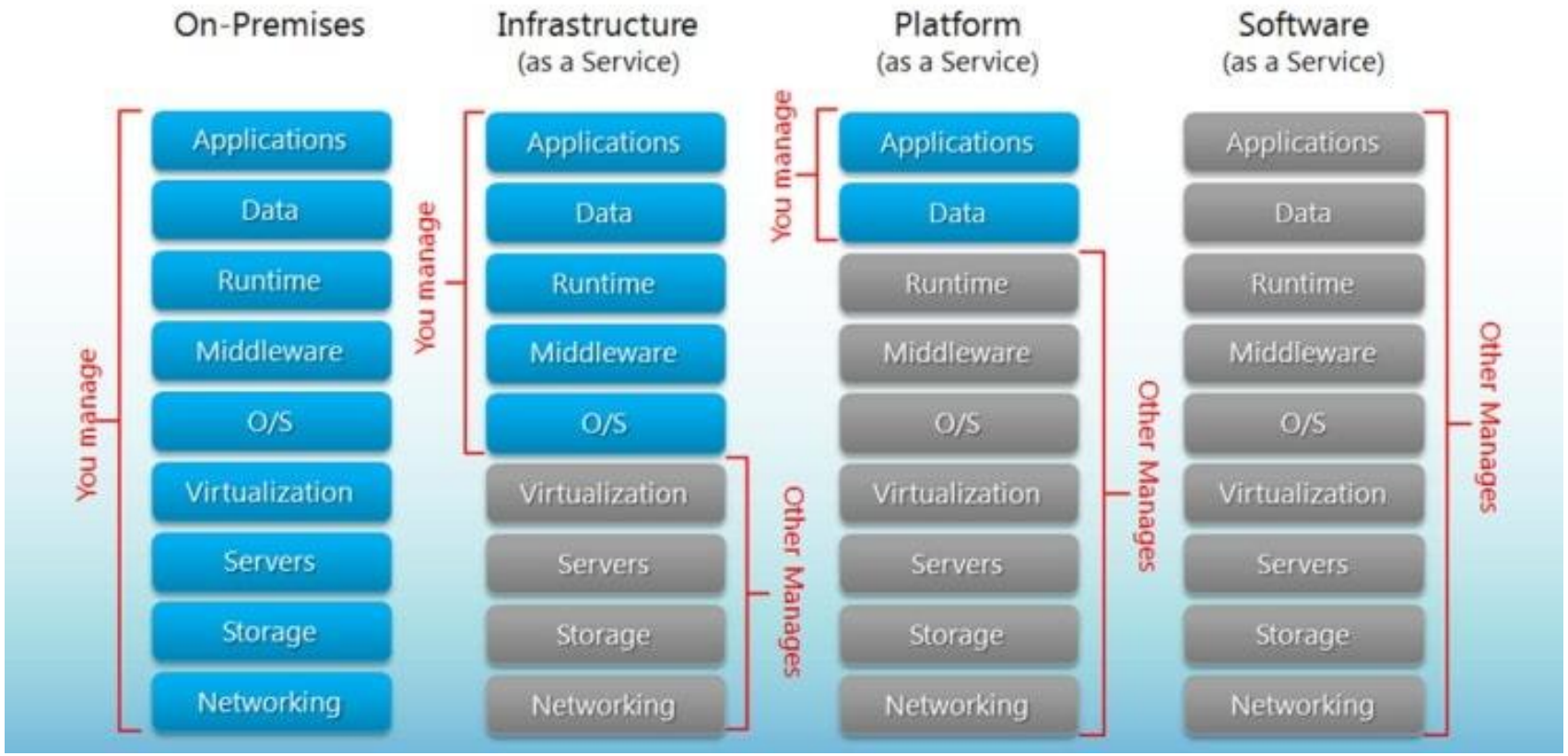
This code represents a basic skeleton for an operating system kernel. It initializes the hardware, memory, process manager, and file system, and then starts the shell process. This is just an example and a real operating system kernel would be much more complex and include many more components and features.


LINES OF CODE -
GOOGLE
2 Billion
FACEBOOK
61 Million
WHATSAPP
30 Thousand
INSTAGRAM
50 Million
TELEGRAM
50 Thousand
YOUTUBE
80 Million
TIKTOK
80 Thousand
SNAPCHAT
60 Thousand
AMAZON
50 Million
WINDOWS
50 Million
LINUX
18 Million
LINKEDIN
8 Million

OpenStack: The Cloud Operating System

XaaS


Applications
Middleware/OS
Servers
IaaS
host
Applications
Middleware/OS
Servers
PaaS
build
Applications
Middleware/OS
Servers
SaaS
Consume


On-Premises
Infrastructure
(as a Service)
Platform
(as a Service)
Software
(as a Service)
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
You manage
You manage
You manage
Other Manages
Other Manages
Other Manages

Types of cloud computing

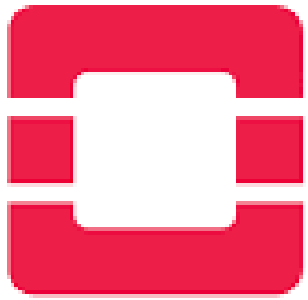
- **Public Cloud**: In Public Cloud the computing infrastructure is hosted by the cloud vendor at the vendor’s premises. (AWS)
- The customer has no visibility and control over where the computing infrastructure is hosted.
- The computing infrastructure is shared between any organizations.

- **Private Cloud**: The computing infrastructure is dedicated to a particular organization and not shared with other organizations.
- Private Clouds are more expensive and more secure when compared to Public Clouds.

- Private Cloud is what used to be called your company network.
- It may be accessed whilst inside the firewall, or outside via some form of secure VPN.
- But it is a resource controlled and consumed by your internal IT department.

- **Hybrid Cloud**: Organizations may **host critical** applications on Private Clouds and **applications with relatively less security** concerns on the Public Cloud.
- The usage of both Private and Public Clouds together is called Hybrid Cloud. A related term is ***Cloud Bursting***.

- In **Cloud Bursting** organization use their own computing infrastructure for normal usage, but access the cloud using services like Salesforce cloud computing for high/peak load requirements.
- This ensures that a sudden increase in computing requirement is handled gracefully.

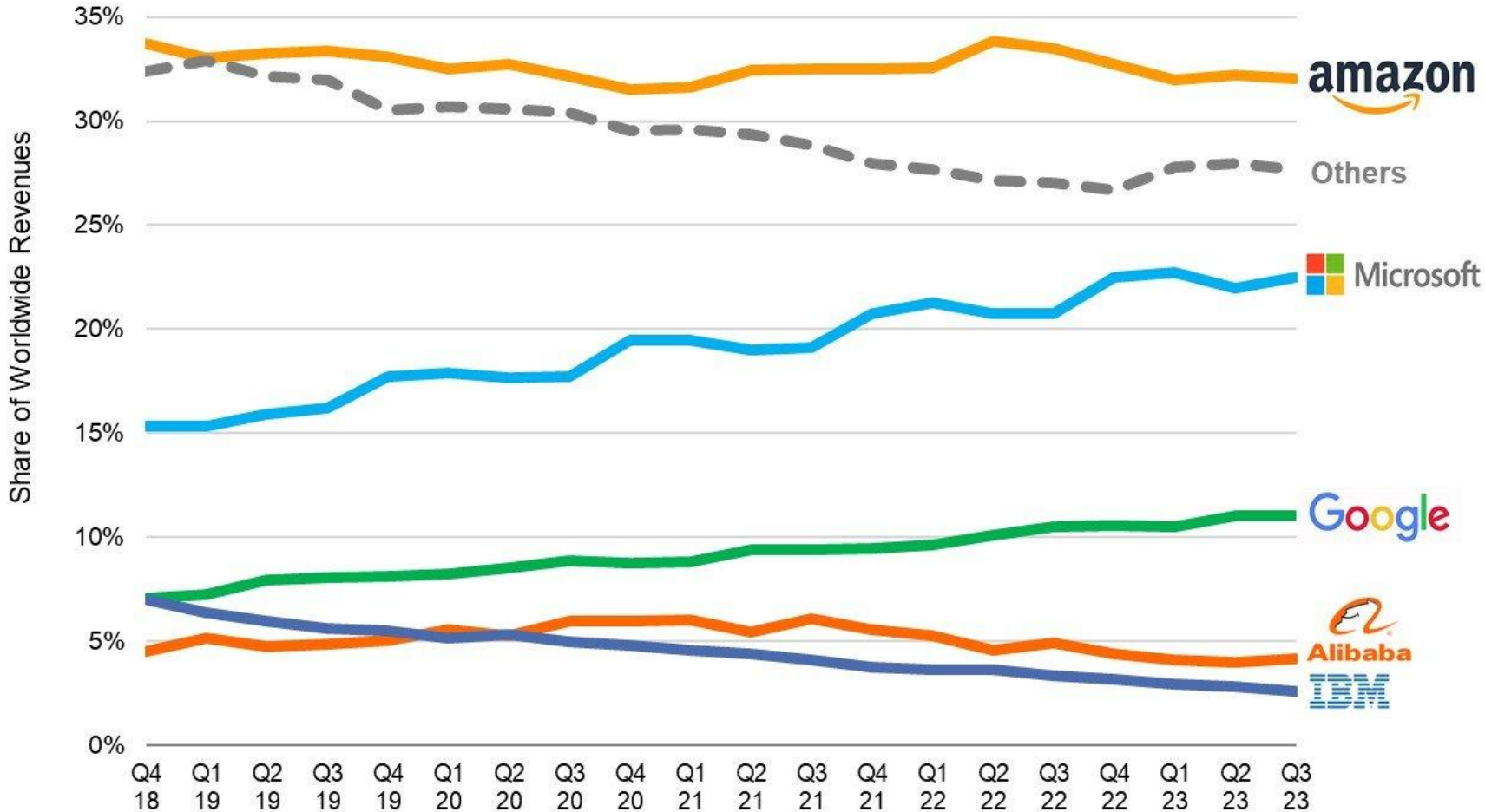


openstack®

OpenStack: The Cloud Operating System

Cloud Provider Market Share Trend

(IaaS, PaaS, Hosted Private Cloud)



Source: Synergy Research Group

Releases Home

Contents

- OpenStack Releases
  - Release Series
  - Series-Independent Releases
  - Teams
  - Cryptographic Signatures
  - References

OpenStack Releases

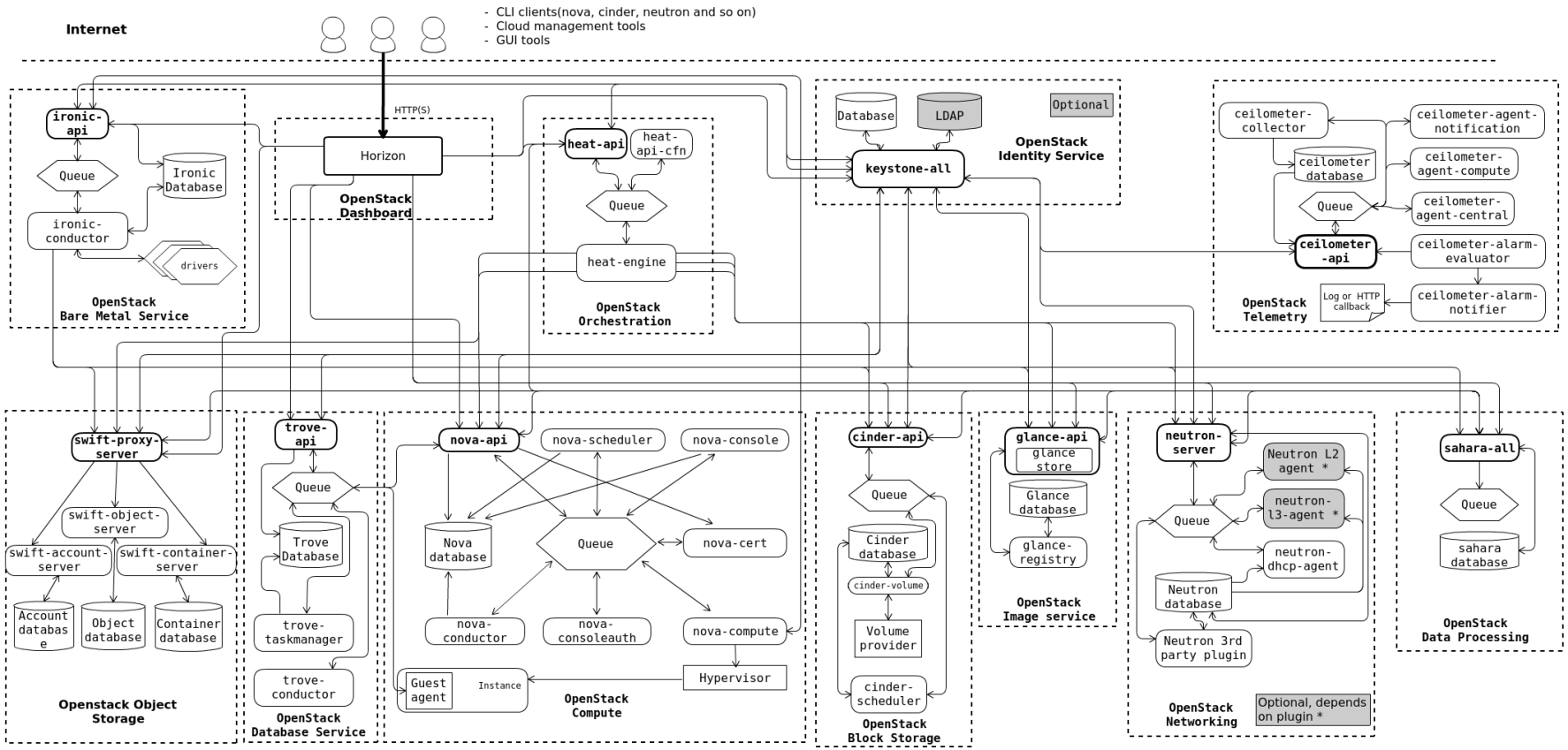
Release Series

OpenStack is developed and released around 6-month cycles. After the initial release, additional stable point releases will be released in each release series. You can find the detail of the various release series here on their series page. Subscribe to the combined release calendar for continual updates.

Series	Status	Initial Release Date	Next Phase	EOL Date
Ussuri	Development	2020-05-13 *estimated* (schedule)	Maintained *estimated 2020-05-13*	
Train	Maintained	2019-10-16	Extended Maintenance *estimated 2021-04-16*	
Stein	Maintained	2019-04-10	Extended Maintenance *estimated 2020-10-10*	
Rocky	Maintained	2018-08-30	Extended Maintenance *estimated 2020-02-24*	
Queens	Maintained	2018-02-28	Extended Maintenance *estimated 2019-10-25*	
Pike	Extended Maintenance	2017-08-30	Unmaintained *estimated TBD*	
Ocata	Extended Maintenance	2017-02-22	Unmaintained *estimated TBD*	
Newton	End Of Life	2016-10-06		2017-10-25
Mitaka	End Of Life	2016-04-07		2017-04-10
Liberty	End Of Life	2015-10-15		2016-11-17
Kilo	End Of Life	2015-04-30		2016-05-02

https://releases.openstack.org/

Logical architecture



OpenStack: The Cloud Operating System

- **OpenStack** is a global collaboration of developers and technologists producing an open source cloud computing platform for public and private clouds.

- OpenStack software delivers a massively **scalable cloud operating system**.
- It is an open source *infrastructure as a service* (IaaS) initiative for creating and managing large groups of virtual private servers in a cloud computing environment.

- The goals of the OpenStack initiative are to support ***interoperability*** between cloud services and allow businesses to build cloud services in their own data centers.

- OpenStack lets users deploy virtual machines and other instances which handle different tasks for managing a cloud environment **on the fly.**
- It makes **horizontal scaling easy**, which means that tasks which benefit from running concurrently can easily serve more or less users on the fly by just spinning up more instances.

Architecture of OpenStack

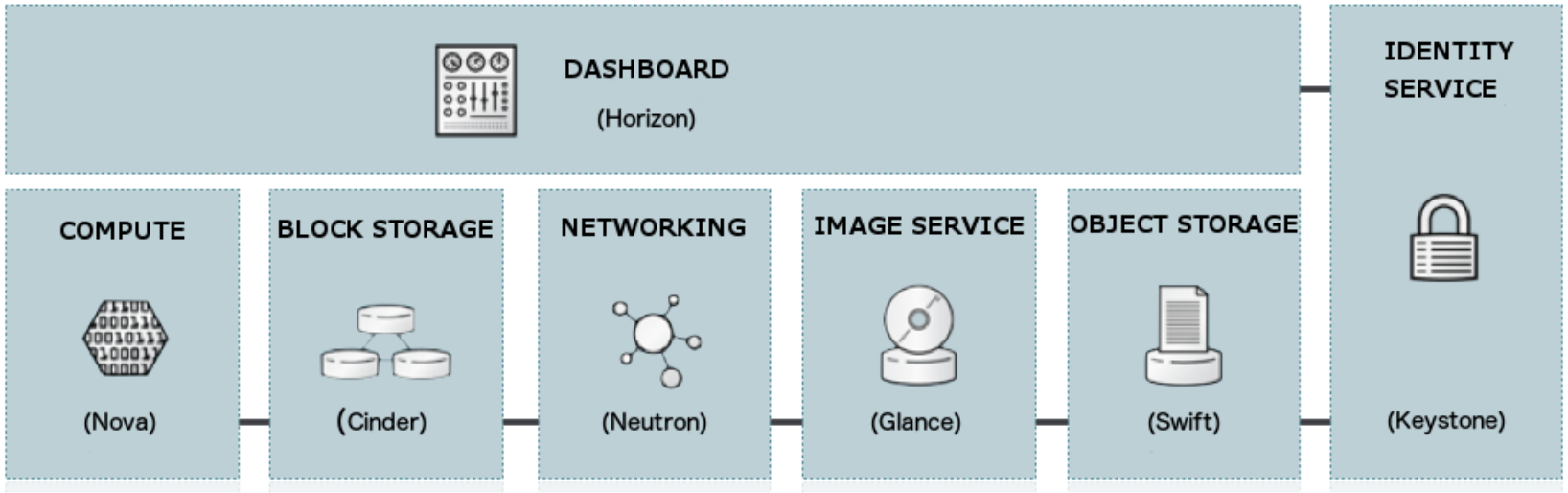
- OpenStack has **a modular architecture** that currently has three components: *compute, storage and image service* :

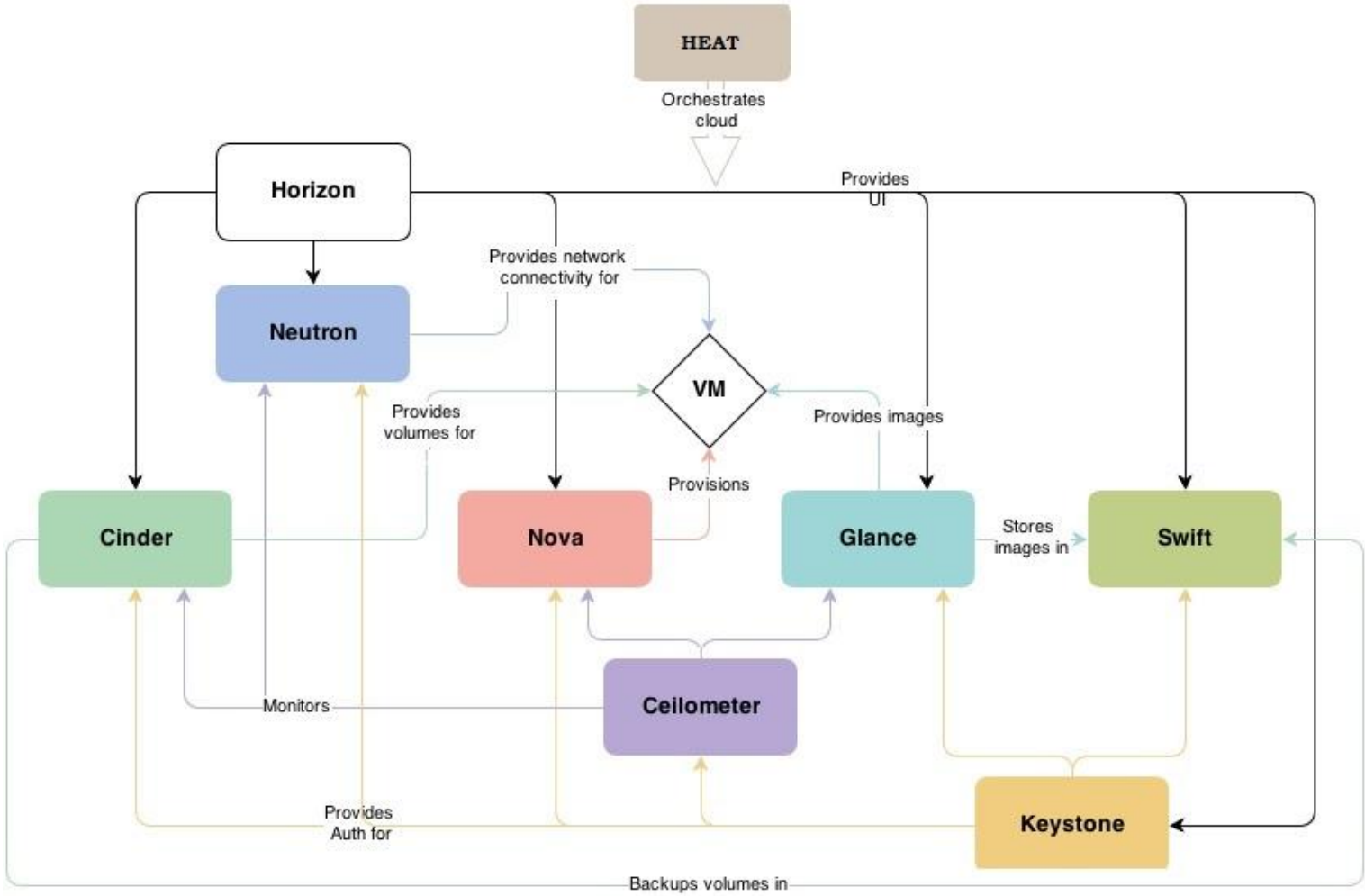
- **Compute:** open source software designed to *provision* and *manage* **large networks of virtual machines**, creating a redundant and scalable cloud computing platform.

- **Storage**: open source software for creating **redundant, scalable object storage** using clusters of standardized servers to store ***petabytes*** of accessible data

- **Image Service**: provides **discovery, registration, and delivery** services for virtual disk images.

Core Components of OpenStack




HEAT
Orchestrates cloud
Horizon
Provides UI
Provides network connectivity for
Neutron
VM
Provides volumes for
Provides images
Provisions
Cinder
Nova
Glance
Stores images in
Swift
Monitors
Ceilometer
Provides Auth for
Keystone
Backups volumes in

- **Nova** is the primary **computing engine** behind OpenStack.
- It is a "fabric controller," which is used for deploying and managing large numbers of virtual machines and other instances to handle computing tasks.

- **Swift** is a **storage system** for objects and files.
- Developers can refer to a unique identifier referring to the file or piece of information and let OpenStack decide where to store this information.

- **Cinder** is **a block storage component**, which is more analogous to the traditional notion of a computer being able to access specific locations on a disk drive.

- **Neutron** provides the **networking capability** for OpenStack.
- It helps to ensure that each of the components of an OpenStack deployment can communicate with one another quickly and efficiently.

- **Horizon** is the **dashboard** behind OpenStack.
- It is the only graphical interface to OpenStack.
- The dashboard provides system administrators a look at what is going on in the cloud, and manage it.

- **Keystone** provides **identity services** for OpenStack.
- It is essentially a central list of all of the users of the OpenStack cloud, mapped against all of the services provided by the cloud which they have permission to use.

- **Glance** provides **image services** to OpenStack.
- In this case, "images" refers to images (or virtual copies) of hard disks.
- Glance allows these images to be used as templates when deploying new virtual machine instances.

- **Ceilometer** provides **telemetry services**, which allow the cloud to provide **billing services** to individual users of the cloud.
- It also keeps a verifiable count of each user’s system usage of each of the various components of an OpenStack cloud.

- **Heat** is the **orchestration** component of OpenStack, which allows developers to store the requirements of a cloud application in a file that defines what resources are necessary for that application.

AI and OS

1. **Automated Code Generation** : AI techniques such as machine learning and natural language processing are used to generate code for various components of the OS.

This includes code for device drivers, kernel modules, and system utilities.

Techniques like program synthesis and code translation are employed to automate the process of generating code from high-level specifications or natural language descriptions.

2. **Dynamic Resource Allocation** : AI algorithms are used for dynamic resource allocation and scheduling in the OS.

This includes techniques like reinforcement learning for task scheduling, neural network-based memory management, and genetic algorithms for processor load balancing.

These techniques help optimize resource utilization and improve system performance.

3. **Power Management** : AI is used for intelligent power management in modern operating systems.

Machine learning models are trained to predict usage patterns and adjust power settings accordingly, leading to better energy efficiency.

Techniques like deep reinforcement learning are used for dynamic voltage and frequency scaling (DVFS) in processors.

4. **Fault Tolerance and Self-Healing** : AI is employed for fault tolerance and self-healing mechanisms in operating systems.

Anomaly detection algorithms based on machine learning are used to identify and isolate system faults.

Techniques like case-based reasoning and expert systems are used for root cause analysis and automated recovery.

5. **User Interface Optimization** : AI is used to improve the user experience in operating systems.

Natural language processing and speech recognition enable voice-based interactions with the OS.

Computer vision and gesture recognition facilitate intuitive touch and motion-based interfaces. Recommendation systems and personalization algorithms tailor the UI to individual user preferences and behavior patterns.

6. **Security and Malware Detection** : Machine learning algorithms are used for security purposes in operating systems.

Techniques like deep learning and ensemble methods are employed for malware detection, intrusion prevention, and identifying security vulnerabilities in the OS codebase.

7. **Performance Optimization** : AI techniques are used for performance optimization in operating systems.

This includes using reinforcement learning for optimizing disk I/O scheduling, neural networks for cache management, and genetic algorithms for memory defragmentation.

8. **Virtual Assistants and Intelligent Agents** : AI-powered virtual assistants and intelligent agents are integrated into modern operating systems.

These agents use natural language processing, knowledge representation, and reasoning techniques to assist users with various tasks, such as file management, application launching, and system configuration.

9. **Predictive Maintenance** : AI is used for predictive maintenance in operating systems.

Machine learning models are trained on system logs and sensor data to predict component failures or performance degradation.

This information is used for proactive maintenance and preventing system downtime.

10. **Automated Testing and Verification** : AI techniques like fuzzing, symbolic execution, and model checking are used for automated testing and verification of operating system code.

These techniques help identify bugs, security vulnerabilities, and ensure compliance with system specifications.

Examples

1. Predictive Resource Management

AI can predict the demand for resources (like CPU, memory) based on usage patterns and pre-allocate resources to improve system performance. Here's a simplified **Python** example using a hypothetical AI model to predict and allocate memory resources:

```python
from sklearn.ensemble import RandomForestRegressor
import numpy as np

# Hypothetical dataset: [hour_of_day, is_weekend, running_apps_count] -> [memory_usag
data = np.array([
    [10, 0, 3, 2.5],
    [12, 0, 5, 3.2],
    [20, 1, 2, 2.8],
    # ... more data
])
X = data[:, :-1]  # Features: hour of day, is_weekend, running_apps_count
y = data[:, -1]   # Target: memory_usage

# Train a model (in a real scenario, this model would be much more complex)
model = RandomForestRegressor()
model.fit(X, y)

def predict_memory_usage(hour_of_day, is_weekend, running_apps_count):
    return model.predict([[hour_of_day, is_weekend, running_apps_count]])

# Example: Predict memory usage for a specific scenario
predicted_memory = predict_memory_usage(15, 0, 4)
print(f"Predicted memory usage: {predicted_memory} GB")

# An OS could use this prediction to pre-allocate memory resources accordingly
```

2. Intelligent File Organization

An AI system can automatically categorize and organize files based on their content, type, or user's past behavior. Here's a pseudo-code example for how such a system might work:

```
from some_ai_text_classification_library import classify_text
from some_ai_image_recognition_library import classify_image
import os

def organize_files(directory):
    for file in os.listdir(directory):
        if file.endswith(".txt"):
            category = classify_text(open(file).read())
        elif file.endswith(".jpg") or file.endswith(".png"):
            category = classify_image(file)
        else:
            category = "misc"

        destination = os.path.join(directory, category)
        if not os.path.exists(destination):
            os.mkdir(destination)

        os.rename(os.path.join(directory, file), os.path.join(destination, file))

# Organize files in a directory
organize_files("/path/to/your/directory")
```

3. Security: Anomaly Detection for Intrusion Detection

AI can enhance security by identifying unusual patterns that may indicate a security breach. Here's an example using a simple anomaly detection algorithm:

```python
from sklearn.svm import OneClassSVM
import numpy as np

# Hypothetical data: [cpu_usage, network_traffic, error_rate]
normal_behavior_data = np.array([
    [20, 0.2, 0.01],
    [15, 0.3, 0.02],
    [25, 0.25, 0.015],
    # ... more data indicating normal operation
])

# Train anomaly detection model
model = OneClassSVM(gamma='auto').fit(normal_behavior_data)

# Anomalous example: High network traffic with usual CPU usage and error rate
new_data = [[20, 0.9, 0.02]]

prediction = model.predict(new_data)
if prediction[0] == -1:
    print("Anomaly detected: Potential security threat or system issue")
```