IV B. TECH I SEMESTER	L	T	P	INTERNAL MARKS	EXTERNAL MARKS	TOTAL MARKS	CREDITS
	3	0	0	30	70	100	3
Code: R20EC4111	**EDGE COMPUTING** ***(Professional Elective – V)***						

COURSE OBJECTIVES:

1. To introduce the concepts of Edge Computing.
2. To understand the IoT Architecture, Implementation of Edge Computing with help of IoT Architecture with Core Modules.
3. To understand the RasberryPi Hardware Layouts and Operating systems and Configuration of Rasberry Pi.
4. To understand the Implementation of Interface of RasberryPi with Micro Computer and Edge to Cloud Protocols and MQTT State transitions.
5. To familiarize with the concepts of Edge Computing with RasberryPi and Industrial and Commercial IoT.

COURSE OUTCOMES:

After completion of this course, the students will be able to

CO1: To illustrate the .Edge Computing use cases and outline Edge computing hardware architecture. **[K2]**

CO2: Make use of IoT architecture and implementation use cases. **[K3]**

CO3: Analyzing the layout and interface, configure of Rasberry Pi. **[K4]**

CO4: List out the relationships of edge computing with Rasberry Pi, with cloud protocols, industrial and commercial IoT and Edge Computing. **[K4]**

SYLLABUS:

UNIT- I: IoT AND EDGE COMPUTING DEFINITION AND USE CASES

Introduction to Edge Computing Scenario's and Use cases- Edge computing purpose and definition, Edge computing use cases, Edge computing hardware architectures, Edge platforms, Edge *vs* Fog Computing, Communication Modles-Edge, Fog and M2M.

UNIT- II: IoT ARCHITECTURE AND CORE IoT MODULES

A Connected ecosystem, IoT versus machine-to-machine versus, SCADA, The value of Network and Metcalfe's and Beckstroms's laws, IoT and Edge Architecture, Role of an architect, Understanding Implementations with examples-Example use case and deployment, Case study-Telemedicine palliative care, Requirements, Implementation, Use case retrospective.

UNIT- III: RASBERRYPi

RasberryPi: Introduction to RasberryPi, About RasberryPi Board, Hardware Layout and Pin outs, Operating systems on RasberryPi, Configuring RasberryPi, Programming RasberryPi, Connecting RasberryPi via SSH, Remote access tools, Interfacing DHT Sensor with Pi, Pi as Webserver, Pi Camera, Image & Video Processing using Pi.

UNIT- IV: INTERFACING RASBERRYPi & MQTT

Implementation of Microcomputer RasberryPi and device Interfacing, Edge to Cloud Protocols-Protocols, MQTT, MQTT publish-subscribe, MQTT Architecture details, MQTT state transitions, MQTT packet structure, MQTT data types, MQTT communication formats, MQTT 3.1.1 working example.

UNIT- V: EDGE COMPUTING WITH RASBERRYPi

Edge Computing with RasberryPi, Industrial and Commercial IoT and Edge, Edge Computing and solutions.

TEXT BOOKS:

1. IoT Edge Computing for Architects –Second Edition, by Perry Lea, Publisher: Packet Publishing, 2020, ISBN: 9781839214806.
2. Raspberry Pi Cookbook, 3rd Edition, by Simon Monk, Publisher: O’Reilly Media, Inc 2019,ISBN: 978149204322.

REFERENCE BOOKS:

1. Fog and Edge Computing: Principles and Paradigms by Rajkumar Buyya, Satish Narayana Srirama, Wiley Publication, 2019, ISBN: 9781119524984.
2. David Jensen, “Beginning Azure IoT Edge Computing: Extending the Cloud to the Intelligent. Edge, Microsoft Azure

UNIT-1: Introduction of IoT and Edge Computing:- Edge Computing Scenario's and Use cases, Edge computing purpose and definition, Edge computing hardware architectures, Edge platforms, Edge vs. Fog Computing, Communication Models - Edge, Fog and M2M.

WHAT IS COMPUTING AND TYPES?

Computing refers to the use of computers and computational systems to process, store, and manage data, perform calculations, and solve problems. It encompasses a wide range of activities involving hardware, software, algorithms, and data.

There are several types of computing that can be categorized based on their characteristics and functionalities. Here are some common types of computing:

1. Cloud Computing: Cloud computing refers to the delivery of computing resources, including processing power, storage, and software applications, over the internet. Users can access and utilize these resources on-demand from remote data centers. Cloud computing offers scalability, flexibility, and cost-efficiency by allowing users to pay for resources they consume.
2. Edge Computing: Edge computing, as discussed earlier, involves processing and analyzing data near the source of its generation, at the edge of the network. It reduces latency, improves response times, and enables real-time or near-real-time decision-making. Edge computing is particularly useful in applications that require immediate processing or operate in environments with limited connectivity.
3. High-Performance Computing (HPC): High-performance computing involves the use of powerful computing systems to solve complex problems or perform large-scale computations. HPC systems typically utilize parallel processing techniques, specialized hardware (such as GPUs or FPGAs), and optimized software algorithms to deliver exceptional computational performance. HPC is used in scientific research,

simulations, weather forecasting, and other computationally intensive tasks.

4. Distributed Computing: Distributed computing involves the use of multiple interconnected computers or nodes to work together as a unified system. The tasks are divided among the nodes, and they collaborate to solve complex problems or perform computations. Distributed computing enables scalability, fault-tolerance, and increased processing power by harnessing the resources of multiple machines.
5. Quantum Computing: Quantum computing utilizes quantum mechanical principles to perform computations. It leverages quantum bits, or qubits, which can represent multiple states simultaneously, allowing for exponential processing power compared to classical computing. Quantum computing has the potential to solve complex problems, optimize algorithms, and advance fields like cryptography and molecular modeling.
6. Mobile Computing: Mobile computing refers to the use of computing devices like smartphones, tablets, and wearables for data processing, communication, and access to applications and services on the go. Mobile computing leverages wireless networks, location awareness, and mobility features to enable portable and personalized computing experiences.
7. Grid Computing: Grid computing involves the pooling of computing resources from multiple geographically distributed and independent sources to work on a common goal. It enables resource sharing, collaboration, and efficient utilization of computing power for large-scale computational tasks.

These are just a few examples of computing types, each serving specific purposes and addressing unique requirements. As technology advances, new computing paradigms and types are continuously emerging, catering to evolving needs and expanding possibilities in the digital world.

INTRODUCTION TO EDGE COMPUTING

Edge computing is a paradigm shift in computing that brings data processing and storage closer to the source of data generation, at the edge of a network. Traditionally, data generated by devices was sent to centralized data centers or the cloud for processing and analysis. However, with the rise of Internet of Things (IoT) devices, autonomous systems, and real-time applications, there is a growing need for faster response times, reduced latency, improved privacy, and more efficient network utilization. This is where edge computing comes into play.

In edge computing, computing resources such as servers, gateways, or edge devices are deployed closer to the edge devices or sensors, where data is generated. These edge nodes have the capability to perform localized data processing, analytics, and storage, reducing the need to transmit large amounts of data to remote data centers or the cloud. By processing data at the edge, latency is minimized, and real-time or near-real-time decision-making becomes possible.

The key advantages of edge computing include:

1. Reduced Latency: By processing data locally at the edge, response times are significantly improved. Applications that require real-time interactions, such as autonomous vehicles or industrial automation, can benefit from immediate data processing without relying on distant servers.
2. Bandwidth Optimization: Edge computing minimizes the need for data transfer over the network. Only relevant or summarized data is transmitted to the cloud, reducing bandwidth requirements and associated costs.
3. Improved Reliability: Edge computing enables local data processing, which means applications can continue to function even in cases of network disruption or intermittent connectivity to the cloud. It enhances system resilience and availability.
4. Enhanced Privacy and Security: Edge computing allows for data to be processed and analyzed locally, reducing the exposure of sensitive information to the cloud or external networks. This can be crucial in industries like healthcare or finance, where data privacy is of utmost importance.
5. Scalability: Edge computing can scale horizontally by deploying additional edge devices or servers as needed. It enables distributed computing capabilities, accommodating varying workloads and increasing computational capacity at the edge.

Edge computing finds applications in a wide range of industries and scenarios. It is particularly relevant in IoT deployments, smart cities, industrial automation, healthcare, autonomous vehicles, retail, and telecommunications, among others.

As technology continues to advance, edge computing is expected to play a pivotal role in enabling the growth of emerging technologies like 5G, AI, and the proliferation of connected devices. It complements and enhances cloud computing, providing a decentralized and distributed computing architecture that meets the demands of today's data-intensive and latency-sensitive applications.

EDGE COMPUTING PURPOSE & DEFINITION

Edge computing refers to the practice of processing, analyzing, and storing data near its source, at the edge of a network, rather than sending it to a centralized data center or cloud. The purpose of edge computing is to enable real-time data processing and reduce the latency, bandwidth, and privacy concerns associated with transmitting data to a remote location for analysis.

In traditional computing models, data generated by devices at the network edge, such as sensors, cameras, or IoT devices, is sent to a central server or cloud infrastructure for processing. However, this approach can result in delays due to the time it takes to transmit the data over the network. Additionally, in scenarios where large amounts of data are generated, transmitting all of it to a remote server may be impractical or costly in terms of network bandwidth.

Edge computing addresses these challenges by bringing computation and data storage closer to the edge devices themselves. By deploying computing resources, such as servers, gateways, or edge devices, in proximity to where the data is generated, processing and analysis can be performed locally. This enables faster response times, reduces the need for extensive data transfers, and can improve overall system efficiency.

Edge computing is particularly useful in scenarios that require real-time or near-real-time processing, such as industrial automation, autonomous vehicles, remote monitoring, smart cities, and healthcare applications. It allows for quicker decision-making, faster response to critical events, and the ability to operate even in situations with limited or intermittent connectivity to the cloud.

Overall, the purpose of edge computing is to distribute computing capabilities and data storage to the network edge, closer to where the data is generated, in order to improve performance, reduce latency, and address the unique requirements of applications that demand real-time processing and low-latency interactions.

EDGE COMPUTING USE CASES

Edge computing has a wide range of use cases across various industries. Here are some examples:

1. Internet of Things (IoT): Edge computing is fundamental to IoT deployments as it allows for real-time data processing, reducing the need for constant data transfers to the cloud. It enables local analytics, decision-making, and automation. Use cases include smart homes, smart cities, industrial IoT, agriculture, and asset tracking.
2. Industrial Automation: Edge computing plays a crucial role in industrial settings where low latency and real-time decision-making are critical. It facilitates real-time monitoring and control of machines and processes, predictive maintenance, and optimization of manufacturing operations.
3. Autonomous Vehicles: Edge computing is vital for autonomous vehicles as they generate massive amounts of data that require immediate processing for real-time decision-making. Edge computing enables on-board analytics, object detection, image recognition, and vehicle-to-vehicle communication.
4. Retail: In retail environments, edge computing enables personalized shopping experiences, inventory management, and real-time analytics. It can power applications like smart shelves, in-store analytics, customer tracking, and dynamic pricing.
5. Telecommunications: Edge computing enhances network performance, reduces latency, and enables new services for telecommunications providers. It supports applications like content delivery networks (CDNs),

mobile edge computing (MEC), virtual network functions (VNFs), and network slicing.

6. Healthcare: Edge computing is valuable in healthcare scenarios where immediate processing of data is crucial, such as remote patient monitoring, real-time analysis of patient data, and wearable health devices. It enables faster diagnoses, timely interventions, and improved patient outcomes.
7. Smart Grids: Edge computing is employed in utility grids for real-time monitoring, fault detection, and load balancing. It facilitates localized analytics, grid management, and enables efficient energy distribution and utilization.
8. Surveillance and Security: Edge computing enables video analytics, facial recognition, and real-time threat detection in security and surveillance applications. It reduces bandwidth requirements and improves response times in critical situations.
9. Edge AI: Edge computing combined with artificial intelligence (AI) allows for localized AI processing and inference. It supports applications like real-time video analytics, natural language processing, and intelligent edge devices.
10. Remote and Harsh Environments: Edge computing is valuable in scenarios where internet connectivity is limited or intermittent, such as offshore operations, mining, and remote monitoring. It enables local data processing, decision-making, and resilience in challenging environments.

These are just a few examples of the diverse use cases for edge computing. As the technology advances, new applications and industries are likely to benefit from its capabilities.

EDGE COMPUTING HARDWARE ARCHITECTURES

Edge computing hardware architectures can vary depending on the specific requirements of the application and the scale of deployment. Here are some common hardware architectures used in edge computing:

1. Edge Servers: Edge servers are deployed at the edge of the network and are similar to traditional servers but with a smaller form factor. These

servers have computing power, storage, and networking capabilities. They are capable of running applications and processing data locally, reducing the need for data transfer to the cloud. Edge servers can be standalone units or rack-mounted devices.

2. Edge Gateways: Edge gateways act as intermediaries between edge devices and the cloud or data center. They provide connectivity, protocol translation, and local data processing capabilities. Edge gateways are often used in scenarios where multiple edge devices need to communicate with the cloud, or where edge devices have limited processing power. They aggregate data from multiple devices, preprocess it, and transmit relevant information to the cloud.
3. Edge Appliances: Edge appliances are specialized hardware devices designed for specific edge computing applications. These appliances are purpose-built for tasks like video analytics, industrial automation, or IoT data processing. They are optimized for low power consumption, compact size, and specific computational requirements. Edge appliances are typically deployed in environments where dedicated processing capabilities are needed.
4. Edge Accelerators: Edge accelerators are specialized hardware devices that focus on accelerating specific types of computations, such as AI inference or machine learning workloads. They can be in the form of dedicated chips, graphics processing units (GPUs), field-programmable gate arrays (FPGAs), or application-specific integrated circuits (ASICs). Edge accelerators enhance the performance and efficiency of edge computing systems, enabling faster and more power-efficient processing of complex workloads.
5. Edge Sensors/Devices: Edge sensors or devices are the endpoints in an edge computing architecture. These devices collect data from the physical world, such as temperature, humidity, motion, or image data. They are typically low-power devices with limited computing resources. Edge sensors/devices are often deployed in large numbers and rely on edge servers, gateways, or appliances for data processing and analysis.
6. Hybrid Cloud-Edge Architectures: In some cases, a hybrid architecture is used, combining elements of both cloud computing and edge computing. Certain processing tasks may be performed in the cloud for scalability or complex analytics, while time-sensitive or critical processing is handled at

the edge. This architecture allows for a balance between local processing and utilizing the scalability and resources of the cloud.

It's important to note that edge computing hardware architectures can be customized and tailored to specific use cases and requirements. The choice of architecture depends on factors such as the volume and nature of data, processing needs, latency requirements, power constraints, and scalability considerations.

EDGE PATFORMS

Edge platforms, also known as edge computing platforms or edge computing frameworks, are software frameworks or platforms designed to facilitate the development, deployment, and management of edge computing applications and services. These platforms provide the necessary tools, libraries, and infrastructure to build and run applications at the edge of the network.

Edge platforms typically offer a combination of the following features:

1. Edge Node Management: Edge platforms enable the management and orchestration of edge nodes, which are the computing devices deployed at the network edge. They provide capabilities for node discovery, provisioning, configuration, and monitoring. This allows for centralized management and control of the distributed edge infrastructure.
2. Application Development and Deployment: Edge platforms provide software development kits (SDKs), APIs, and development frameworks to streamline the development and deployment of edge applications. They often support multiple programming languages and provide libraries and tools for building and packaging edge applications.
3. Data Management: Edge platforms offer mechanisms for data ingestion, storage, and processing at the edge. They provide data management capabilities such as data caching, synchronization, filtering, and aggregation. These features enable efficient and optimized data processing and storage close to the source.

4. Connectivity and Communication: Edge platforms facilitate the connectivity and communication between edge devices, cloud services, and other components of the computing infrastructure. They provide protocols, APIs, and middleware for seamless integration and data exchange across the distributed edge network.
5. Security and Privacy: Edge platforms prioritize security and privacy in edge computing environments. They offer features such as authentication, access control, encryption, and secure communication protocols to protect data and ensure the integrity and confidentiality of edge applications and services.
6. Analytics and Machine Learning: Some edge platforms include built-in analytics and machine learning capabilities, allowing for real-time data analysis, pattern recognition, and intelligent decision-making at the edge. These features enable local data processing and derive actionable insights without relying on cloud-based resources.
7. Integration with Cloud Services: Many edge platforms integrate with cloud services and platforms to enable seamless hybrid cloud-edge deployments. They provide connectors, APIs, and integration frameworks for easy integration with cloud-based applications, services, and data storage.
8. Scalability and Resilience: Edge platforms support horizontal scaling and the ability to handle increasing workloads and edge devices. They provide mechanisms for load balancing, fault tolerance, and failover to ensure high availability and system resilience in edge computing environments.

Examples of popular edge platforms include Microsoft Azure IoT Edge, AWS IoT Greengrass, Google Cloud IoT Edge, and Dell Technologies' Edge Platform. These platforms provide developers and organizations with the necessary tools and infrastructure to harness the power of edge computing and build innovative edge applications across various industries and use cases.

EDGE VS FOG COMPUTING

Edge computing and fog computing are two closely related concepts that both aim to bring computing capabilities closer to the data source in order to address the challenges of latency, bandwidth, and real-time processing. While they share similar goals, there are subtle differences between the two:

Edge Computing:

- Edge computing focuses on processing and analyzing data at or near the edge devices themselves, typically within the local network.
- The emphasis is on minimizing latency, reducing data transfer, and enabling real-time or near-real-time decision-making.
- In edge computing, processing is typically performed on individual edge devices or localized servers deployed at the edge of the network.
- Edge computing is more decentralized, with each edge device or node capable of performing some level of data processing and analysis independently.

Fog Computing:

- Fog computing extends the concept of edge computing by introducing an intermediate layer between edge devices and the cloud.
- The fog layer, also known as the edge cloud or fog nodes, consists of a distributed network of servers or gateways placed closer to the edge devices.
- The fog layer acts as an intermediary for processing and analyzing data from edge devices before forwarding relevant information to the cloud.
- Fog computing enables local data aggregation, filtering, and preprocessing, reducing the amount of data sent to the cloud and improving efficiency.
- Fog computing provides a hierarchical architecture, with the fog layer acting as an additional tier between edge devices and the cloud, facilitating more complex computations and coordination across multiple edge devices.

In summary, edge computing focuses on processing data at or near the edge devices themselves, while fog computing adds an intermediate layer of fog nodes between the edge devices and the cloud. Fog computing provides a more hierarchical and scalable architecture, enabling more advanced processing capabilities, coordination, and management at the edge. Both approaches aim to address the challenges of latency, bandwidth, and real-time processing in distributed computing environments, but with slightly different architectural considerations.

The basic difference between edge computing and fog computing lies in the architectural placement of computing resources and where data processing occurs

In summary, edge computing focuses on processing data at the edge devices themselves, while fog computing adds an intermediate layer (fog layer) between the edge devices and the cloud. The fog layer facilitates local data processing and acts as an intermediary for forwarding relevant information to the cloud, providing a hierarchical architecture for more advanced computations and coordination at the edge.

COMMUNICATION MODULES-EDGE, FOG AND M2M

Communication modules play a crucial role in enabling connectivity and data exchange within edge computing, fog computing, and machine-to-machine (M2M) systems. Here's an overview of communication modules in each context:

Edge Computing: In edge computing, communication modules facilitate communication between edge devices and other components in the computing infrastructure. These modules can include:

1. Wireless Communication: Edge devices often use wireless communication technologies such as Wi-Fi, Bluetooth, Zigbee, or cellular networks (e.g., 4G, 5G) to connect with each other and communicate with other networked devices.
2. Ethernet: Wired Ethernet connections are commonly used to establish reliable and high-speed communication between edge devices, edge servers, and other network elements.

3. MQTT: The Message Queuing Telemetry Transport (MQTT) protocol is commonly used in edge computing scenarios for lightweight and efficient communication between edge devices and the cloud or other devices. It is well-suited for resource-constrained environments.
4. Protocols and APIs: Various communication protocols and APIs are employed to facilitate data exchange between edge devices and cloud services or other systems. These can include HTTP/REST APIs, WebSockets, OPC UA (for industrial automation), CoAP (Constrained Application Protocol), or custom protocols specific to the edge computing environment.

Fog Computing: Fog computing builds upon edge computing and extends the communication capabilities to include the fog layer or fog nodes. Communication modules in fog computing can include:

1. Fog-to-Edge Communication: Communication modules enable efficient and reliable data exchange between the fog layer and the edge devices. This can involve protocols such as MQTT, CoAP, or custom protocols specifically designed for fog computing environments.
2. Inter-Fog Communication: Fog nodes within the fog layer need to communicate with each other for coordination, resource sharing, and data aggregation. Communication modules such as messaging queues, pub-sub systems, or custom protocols facilitate inter-fog communication.
3. Cloud-to-Fog Communication: Communication modules enable communication between the fog layer and cloud services or data centers. This can involve standard protocols such as HTTP/REST APIs or message queues like MQTT, as well as specific fog-to-cloud integration frameworks provided by fog computing platforms.

Machine-to-Machine (M2M) Communication: M2M communication refers to direct communication between machines or devices without human intervention. Communication modules in M2M systems include:

1. Wireless Communication: M2M devices often utilize wireless communication technologies such as cellular networks, Wi-Fi, Bluetooth, or Zigbee to establish connections and exchange data with other devices in the network.

2. IoT Protocols: M2M systems commonly employ protocols specifically designed for IoT communication, such as MQTT, CoAP, or AMQP (Advanced Message Queuing Protocol). These protocols enable efficient and lightweight communication between devices and facilitate interoperability.
3. Industrial Protocols: In industrial settings, M2M communication often involves industry-specific protocols such as Modbus, OPC UA, or BACnet, which are used for device-to-device communication in areas like industrial automation, building management systems, or smart grids.
4. APIs and Integration: M2M systems may utilize APIs and integration frameworks to enable communication between devices and backend systems, cloud platforms, or other applications. These APIs can be RESTful APIs, custom APIs, or platform-specific integration mechanisms.

In summary, communication modules in edge computing, fog computing, and M2M systems encompass a range of wireless and wired technologies, protocols, and APIs that facilitate efficient data exchange and connectivity between devices, nodes, layers, and external systems.

2

IoT Architecture and Core IoT Modules

The edge computing and IoT ecosphere starts with the simplest of sensors located in the remotest corners of the earth and translates analog physical effects into digital signals (the language of the Internet). Data then takes a complex journey through wired and wireless signals, various protocols, natural interference, and electromagnetic collisions, before arriving in the ether of the Internet. From there, packetized data will traverse various channels arriving at a cloud or large data center. The strength of IoT is not just one signal from one sensor, but the aggregate of hundreds, thousands, potentially millions of sensors, events, and devices.

This chapter starts with a definition of IoT versus machine-to-machine architectures. It also addresses the architect's role in building a scalable, secure, and enterprise IoT architecture. To do that, an architect must be able to speak to the value the design brings to a customer. The architect must also play multiple engineering and product roles in balancing different design choices.

This chapter provides an outline to how the book is organized and how an architect should approach reading the book and performing their role as an architect. The book treats architecture as a holistic exercise involving many systems and domains of engineering. This chapter will highlight:

- **Sensing and power**: We cover the transformation of physical to digital sensing, power systems, and energy storage.
- **Data communication**: We delve into the communication of devices using near-meter, near-kilometer, and extreme-range communication systems and protocols as well as networking and information theory.
- **Edge computing**: Edge devices have multiple roles from routing, to gateways, edge processing and cloud-edge (fog) interconnect. We examine the role of the edge and how to successfully build and partition edge machines. We also look at communication protocols from the edge to the cloud.
- **Compute, analytics and machine learning**: We then examine dataflow through cloud and fog computing, as well as advanced machine learning and complex event processing.
- **Threat and security**: The final content investigates security and the vulnerability of the largest attack surface on earth.

A connected ecosystem

Nearly every major technology company is investing or has invested heavily in IoT and the edge computing space. New markets and technologies have already formed (and some have collapsed or been acquired). Throughout this book, we will touch on nearly every segment in information technology, as they all have a role in IoT.

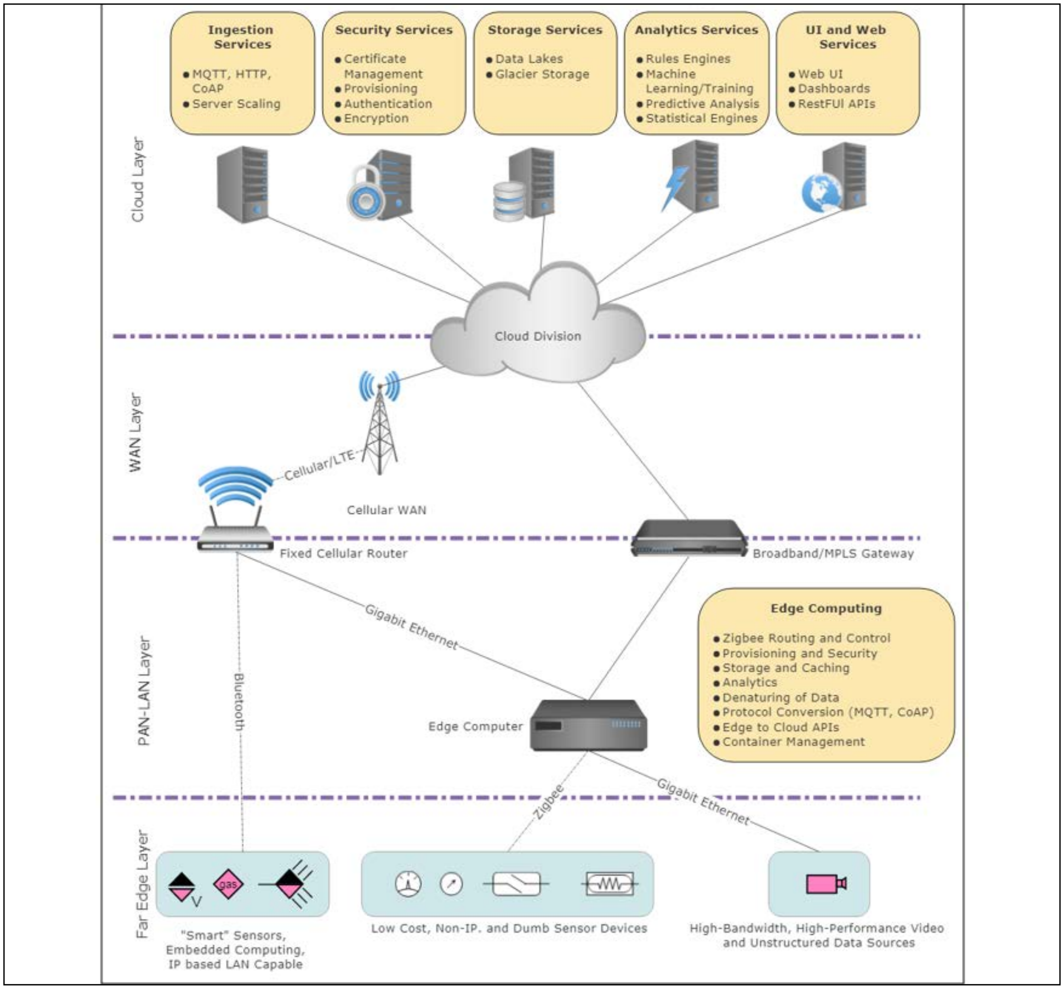


Figure 1: Example of the architectural layers of an IoT/edge computing system. This is one of the many potential configurations that must be considered by the architect. Here we show the sensor-to-cloud routes through direct communication and through edge gateways. We also highlight the functionality provided by the edge compute nodes and the cloud components.

As illustrated in the preceding figure, here are some of the components within an IoT/edge solution that we will study:

- **Sensors, actuators, and physical systems**: Embedded systems, real-time operating systems, energy-harvesting sources, micro-electro-mechanical systems (MEMs).
- **Sensor communication systems**: **Wireless personal area networks** (**WPANs**) reach from 0 cm to 100 m. Low-speed and low-power communication channels, often non-IP based, have a place in sensor communication.
- **Local area networks (LANs)**: Typically, IP-based communication systems such as 802.11 Wi-Fi used for fast radio communication, often in peer-to-peer or star topologies.
- **Aggregators, routers, gateways**: Embedded systems providers, cheapest vendors
- **Wide area networks (WANs)**: Cellular network providers using LTE or Cat M1, satellite network providers, low-power **wide-area network (LPWAN)** providers like Sigfox or LoRa. They typically use Internet transport protocols targeted for IoT and constrained devices like MQTT, CoAP, and even HTTP.
- **Edge computing**: Distributing computing from on-premise data centers and cloud to closer to the sources of data (sensors and systems). This is to remove latency issues, improve response time and real-time systems, manage the lack of connectivity, and build redundancy of a system. We cover processors, DRAM, and storage. We also study module vendors, passive component manufacturers, thin client manufacturers, cellular and wireless radio manufacturers, middleware providers, fog framework providers, edge analytics packages, edge security providers, certificate management systems, WPAN to WAN conversion, routing protocols, and software-defined networking/software-defined perimeters.
- **Cloud**: Infrastructure as a service provider, platform as a service provider, database manufacturers, streaming and batch processing manufacturers, data analytics packages, software as a service provider, data lake providers, and machine learning services.
- **Data analytics**: As the information propagates to the cloud en masse, dealing with volumes data and extracting value is the job of complex event processing, data analytics, and machine learning techniques. We study different edge and cloud analytic techniques from statistical analysis and rules engines to more advanced machine learning techniques.

- **Security**: Tying the entire architecture together is security. End-to-end security from edge hardening, protocol security, to encryption. Security will touch every component from physical sensors to the CPU and digital hardware to the radio communication systems to the communication protocols themselves. Each level needs to ensure security, authenticity, and integrity. There cannot be a weak link in the chain, as the IoT will form the largest attack surface on earth.

This ecosystem will need talents from the body of engineering disciplines, such as:

- Device physics scientists developing new sensor technologies and many-year batteries
- Embedded system engineers working on driving sensors at the edge
- Network engineers capable of working in a personal area network or wide area network, as well as on a software-defined networking
- Data scientists working on novel machine learning schemes at the edge and in the cloud
- DevOps engineers to successfully deploy scalable cloud solutions as a well as *fog* solutions

IoT will also need service vendors such as solution provision firms, system integrators, value-added resellers, and OEMs.

IoT versus machine-to-machine versus SCADA

One common area of confusion in the IoT world is what separates it from the technologies that define **machine to machine** (**M2M**). Before IoT became part of the mainstream vernacular, M2M was the hype. Well before M2M, **SCADA** (**supervisory control and data acquisition**) systems were the mainstream of interconnected machines for factory automation. While these acronyms refer to interconnected devices and may use similar technologies, there are differences. Let's examine these more closely:

- **M2M**: It is a general concept involving an autonomous device communicating directly to another autonomous device. *Autonomous* refers to the ability of the node to instantiate and communicate information with another node without human intervention. The form of communication is left open to the application. It may very well be the case that an M2M device uses no inherent services or topologies for communication. This leaves out typical Internet appliances used regularly for cloud services and storage. An M2M system may communicate over non-IP based channels as well, such as a serial port or custom protocol.

- **IoT**: IoT systems may incorporate some M2M nodes (such as a Bluetooth mesh using non-IP communication), but they aggregate data at an edge router or gateway. An edge appliance like a gateway or router serves as the entry point onto the Internet. Alternatively, some sensors with more substantial computing power can push the Internet networking layers onto the sensor itself. Regardless of where the Internet *on-ramp* exists, the fact that it has a method of tying into the Internet fabric is what defines IoT.
- **SCADA**: This term refers to supervisory control and data acquisition. These are industrial control systems that have been used in factory, facility, infrastructure and manufacturing automation since the 1960s. They typically involve **programmable logic controllers** (**PLCs**) that monitor or controls various sensors and actuators on machinery. SCADA systems are distributed and only recently have been connected to Internet services. This is where Industry 2.0 and the new growth of manufacturing is taking place. These systems use communication protocols such as ModBus, BACNET, and Profibus.

By moving data onto the Internet for sensors, edge processors, and smart devices, the legacy world of cloud services can be applied to the simplest of devices. Before cloud technology and mobile communication became mainstream and cost-effective, simple sensors and embedded computing devices in the field had no good means of communicating data globally in seconds, storing information for perpetuity, and analyzing data to find trends and patterns. As cloud technologies advanced, wireless communication systems became pervasive, new energy devices like lithium-ion became cost-effective, and machine learning models evolved to produce actionable value. This greatly improved the IoT value proposition. Without these technologies coming together when they did, we would still be in an M2M world.

The value of a network and Metcalfe's and Beckstrom's laws

It has been argued that the value of a network is based on Metcalfe's law. Robert Metcalfe in 1980 formulated the concept that the value of any network is proportional to the square of connected *users* of a system. In the case of IoT, "users" may mean sensors or edge devices with some form of communication.

Generally speaking, Metcalfe's law is represented as:

$$V \propto N^2$$

Where:

- V = Value of the network
- N = Number of nodes within the network

A graphical model helps to understand the interpretation as well as the crossover point, where a positive **return on investment** (**ROI**) can be expected:

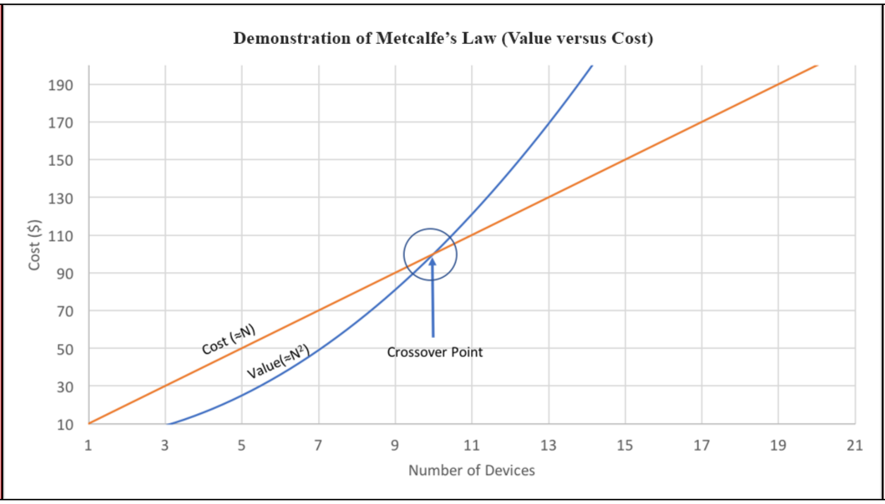


Figure 2: Metcalfe's law: The value of a network is represented as proportional to N^2. The cost of each node is represented as kN where k is an arbitrary constant. In this case, k represents a constant of $10 per IoT edge sensor. The key takeaway is the crossover point occurs rapidly due to the expansion of value and indicates when this IoT deployment achieves a positive ROI.

An example validating Metcalfe's law to the value of blockchains and cryptocurrency trends was recently conducted. We will go much deeper into blockchains in the security chapter.

A recent white paper by Ken Alabi finds that blockchain networks also appear to follow Metcalfe's law, *Electronic Commerce Research and Applications*, Volume 24, C (July 2017), page number 23-29.

Metcalfe's law does not account for service degradation in cases in which service degrades as the number of users and/or data consumption grows, but the network bandwidth does not. Metcalfe's law also doesn't account for various levels of network service, unreliable infrastructure (such as 4G LTE in a moving vehicle), or bad actors affecting the network (for example, denial of service attacks).

To account for these circumstances, we use Beckstrom's law:

$$\sum_{i=1}^{n} V_{i,j} = \sum_{i=1}^{n} \sum_{k=1}^{m} \frac{B_{i,j,k} - C_{i,j,k}}{(1 + r_k)^{t_k}}$$

Where:

- $V_{i,j}$: Represents the present value of the network for device *i* on network *j*
- *i*: An individual user or device on the network
- *j*: The network itself
- *k*: A single transaction
- $B_{i,j,k}$: The benefit that value *k* will bring to device *i* on network *j*
- $C_{i,j,k}$: The cost of a transaction *k* to a device *i* on network *j*
- r_k: The discount rate of interest to the time of transaction *k*
- t_k: The elapsed time (in years) to transaction *k*
- *n*: The number of individuals
- *m*: The number of transactions

Beckstrom's law teaches us that to account for the value of a network (for example, an IoT solution), we need to account for all transactions from all devices and sum their value. If the network *j* goes down for whatever reason, what is the cost to the users? This is the impact an IoT network brings and is a more representative real-world attribution of value. The most difficult variable to model in the equation is the benefit of a transaction *B*. While looking at each IoT sensor, the value may be very small and insignificant (for example, a temperature sensor on some machine is lost for an hour). At other times, it can be extremely significant (for example, a water sensor battery died, and a retailer basement is flooded, causing significant inventory damage and insurance adjustments).

An architect's first step in building an IoT solution should be to understand what value they are bringing to what they are designing. In the worst case, an IoT deployment becomes a liability and actually produces negative value for a customer.

IoT and edge architecture

The coverage in this book will span many technologies, disciplines, and levels of expertise. As an architect, one needs to understand the impact that choosing a certain design aspect will have on scalability and other parts of the system. The complexities and relationships of IoT technologies and services are significantly more intercoupled than traditional technologies not only because of the scale but also due to the disparate types of architecture. There is a bewildering number of design choices. For example, as of this writing, we counted over 700 IoT service providers alone offering cloud-based storage, SaaS components, IoT management systems, middleware, IoT security systems, and every form of data analytics one can imagine. Add to that the number of different PAN, LAN, and WAN protocols that are constantly changing and varying by region. Choosing the wrong PAN protocol could lead to poor communications and significantly low signal quality that can only be resolved by adding more nodes to complete a mesh. The role of an architect should ask and provide solutions for problems that span the system as a whole:

- The architect needs to consider interference effects in the LAN and WAN—how will the data get off the edge and on the Internet?
- The architect needs to consider resiliency and how costly the loss of data is. Should resiliency be managed within the lower layers of the stack, or in the protocol itself?
- The architect must also make choices of Internet protocols such as MQTT versus CoAP and AMQP, and how that will work if he or she decides to migrate to another cloud vendor.

Choices also need consideration with regards to where processing should reside. This opens up the notion of edge/fog computing to process data close to its source to solve latency problems, but more importantly to reduce bandwidth and costs of moving data over WANs and clouds. Next, we consider all the choices in analyzing the data collected. Using the wrong analytic engine may result in useless noise or algorithms that are too resource-intensive to run on edge nodes. How will queries from the cloud back to the sensor affect the battery life of the sensor device itself? Add to this litany of choice, and we must layer on security as the IoT deployment we have built is now the largest attack surface in our city. As you can see, the choices are many and have relationships with one another.

There are many choices to consider. When you account for the number of edge computing systems and routers, PAN protocols, WAN protocols, and communication, there are over 1.5 million different combinations of architectures to choose from:

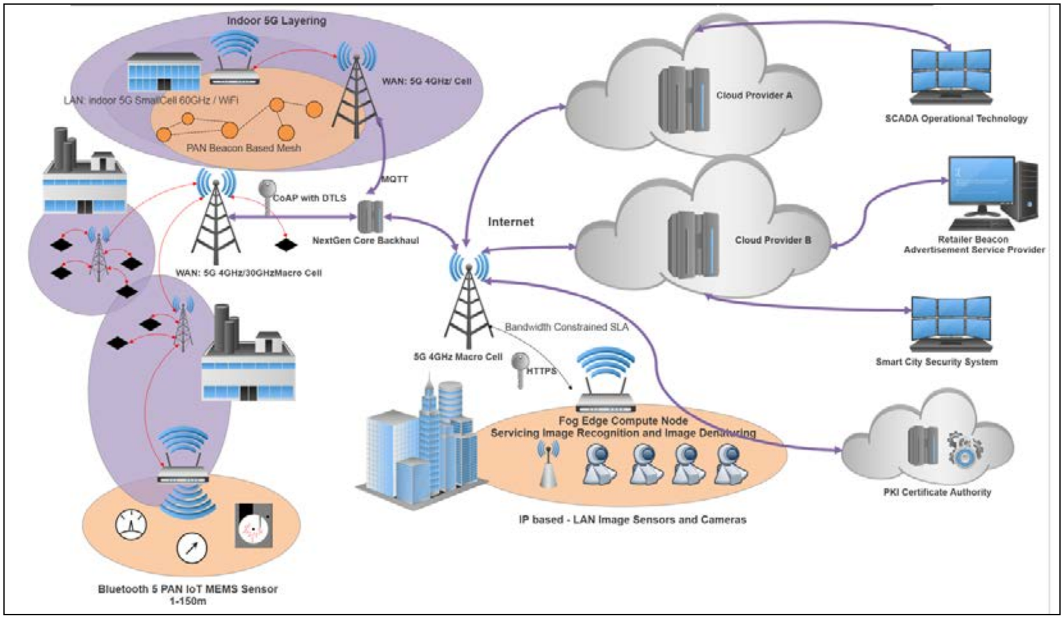

Indoor 5G Layering
WAN: 5G 4GHz/ Cell
LAN: indoor 5G SmallCell 60GHz / WiFi
PAN Beacon Based Mesh
MQTT
CoAP with DTLS
NextGen Core Backhaul
WAN: 5G 4GHz/30GHzMacro Cell
Internet
Cloud Provider A
SCADA Operational Technology
Cloud Provider B
Retailer Beacon
Advertisement Service Provider
Smart City Security System
Bandwidth Constrained SLA
5G 4GHz Macro Cell
HTTPS
Fog Edge Compute Node
Servicing Image Recognition and Image Denoising
PKI Certificate Authority
IP based - LAN Image Sensors and Cameras
Bluetooth 5 PAN IoT MEMS Sensor
1-150m

Figure 3: IoT design choices: The full spectrum of various levels of IoT architecture from the sensor to the cloud and back.

Role of an architect

The term **architect** is often used in technical disciplines. There are software architects, system architects, and solution architects. Even within specific domains, such as computer science and software engineering, you may see people with the title SaaS architect, cloud architect, data science architect, and so on. These are individuals who are recognized experts with tangible skills and experience in a domain. These types of specialized vertical domains cross several horizontal technologies. In this book, we are targeting the IoT architect.

This is a horizontal role, meaning it will touch a number of these domains and bring them together for a usable, secure, and scalable system.

> We will go as deep as necessary to understand an entire IoT system and bring a system together. At times, we will go into pure theory, such as information and communication theory. Other times, we will brush on topics that are on the periphery of IoT systems or are rooted in other technologies. By reading and referencing this book, the architect will have a go-to guide on different aspects of IoT that are all needed to build a successful system.

Whether you are disciplined in electrical engineering or computer science, or have domain expertise in cloud architectures, this material will help you understand a holistic system—which should be, by definition, part of the role of an architect.

This book is also intended for geographically global and massive scaling. It is one thing to build a proof of concept with one or two endpoint devices. It is by far a different challenge to build an IoT solution that stretches multiple continents, different service providers, and thousands of endpoints. While every topic can be used for hobbyist and maker movements, this is intended to scale to global enterprise systems on the order of thousands to millions of edge devices.

The architect will ask questions for the full stack of connected systems. He or she will be aware of how optimizing for one solution may in fact deliver a less than desirable effect in another part of the system.

For example:

- Will the system scale and to what capacity? This will affect decisions on wide area networking, edge-to-cloud protocols, and middleware-to-cloud provisioning systems.
- How will the system perform with loss of connectivity? This will impact the choices of edge systems, storage components, 5G service profiles, and network protocols.
- How will the cloud manage and provision edge devices? This will affect decisions on edge middleware and fog components, and security services.
- How will my customer's solution work in a noisy RF environment? This will affect decisions on PAN communications and edge components.
- How will software be updated on a sensor? This will affect decisions around security protocols, edge hardware and storage, PAN network protocols, middleware systems, sensor costs and resources, and cloud provisioning layers.

- What data will be useful to improving my customer's performance? This will affect decisions on what analytics tools to use, where to analyze the data, how data will be secured and denatured, and edge/cloud partitioning.
- How will devices, transactions, and communication be secured from end to end?

Part 1 – Sensing and power

An IoT transaction starts or ends with an event: a simple motion, a temperature change, perhaps an actuator moving on a lock. Unlike many IT devices in existence, IoT in a large part is about a physical action or event. It responds to affect a real-world attribute. Sometimes this involves considerable data being generated from a single sensor, such as auditory sensing for preventative maintenance of machinery. Other times, it's a single bit of data indicating vital health data from a patient. Whatever the case may be, sensing systems have evolved and made use of Moore's law in scaling to sub-nanometer sizes and significantly reduced costs. Part 1 explores the depths of MEMs, sensing, and other forms of low-cost edge devices from a physical and electrical point of view. The part also details the necessary power and energy systems to drive these edge machines. We can't take power for granted at the edge. Collections of billions of small sensors will still require a massive amount of energy in total to power. We will revisit power throughout this book, and how innocuous changes in the cloud can severely impact the overall power architecture of a system.

Part 2 – Data communication

A significant portion of this book surrounds connectivity and networking. There are countless other sources that dive deep into application development, predictive analytics, and machine learning. This book too will cover those topics, but an equal amount of emphasis is given to data communications. The IoT wouldn't exist without significant technologies to move data from the remotest and most hostile environment to the largest data centers at Google, Amazon, Microsoft, and IBM. The acronym IoT contains the word *Internet*, and because of that, we need to dive deep into networking, communications, and even signal theory. The starting point for IoT isn't sensors or the application; it's about connectivity, as we will see throughout this book. A successful architect will understand the constraints of Internetworking from a sensor to a WAN and back again.

This communication and networking section starts with theory and mathematical foundations of communication and information. Preliminary tools and models are needed by a successful architect not only to understand why certain protocols are constrained, but also to design future systems that scale successfully at IoT levels.

These tools include wireless radio dynamics like range and power analysis, signal-to-noise ratio, path loss, and interference. Part 2 also details foundations of information theory and constraints that affect overall capacity and quality of data. The foundations of Shannon's law will be explored. The wireless spectrum is also finite and shared, so an architect deploying a massive IoT system will need to understand how the spectrum is allocated and governed.

Theory and models explored in this part will be reused in other parts of the book.

Data communication and networking will then build up from the near-range and near-meter communication systems known as **personal area networks** (**PANs**), typically using non-Internet protocol messages. The chapter on PAN will include the new Bluetooth 5 protocol and mesh, as well as Zigbee and Z-Wave in depth. These represent the plurality of all IoT wireless communication systems. Next, we explore wireless local area networks and IP-based communication systems including the vast array of IEEE 802.11 Wi-Fi systems, thread, and 6LoWPAN. The chapter also investigates new Wi-Fi standards such as 802.11p for in-vehicle communication.

The part concludes with long-range communication using cellular (4G LTE) standards, and dives deep into the understanding and infrastructure to support 4G LTE and new standards dedicated to IoT and machine-to-machine communication, such as Cat-1 and Cat-NB. The last chapter also covers the 5G standard and publicly licensed cellular (MulteFire) to prepare the architect for future long-range transmissions where every device is connected in some capacity. A proprietary protocol like LoRaWAN and Sigfox are also explored to understand the differences between architectures.

Part 3 – Edge computing

Edge computing brings nontraditional computing power close to the sources of data. While embedded systems have existed in devices for the last 40 years, edge computing is more than a simple 8-bit microcontroller or analog-to-digital converter circuit used to display temperature. Edge computing attempts to solve critical problems as the number of connected objects and the complexity of use cases grows in the industry. For example, in IoT areas we need the following:

- Accumulate data from several sensors and provide an entry point to the Internet.
- Resolve critical real-time responses for safety-critical situations like remote surgery or automated driving.

- Solutions that can manage an overwhelming amount of processing of unstructured data like video data or even streaming of video to save on costs of transporting the data over wireless carriers and cloud providers.

Edge computing also comes in layers as we will examine with 5G infrastructure, multiaccess edge computing, and fog computing.

We will closely examine the hardware, operating systems, mechanics, and power that an architect must consider for different edge systems. For example, an architect may need a system that delivers on a constraining cost and power requirement but may forgo some processing ability. Other designs may need to be extremely resilient as the edge computer may be in a very remote region and essentially need to manage itself.

To bridge data from sensors to the Internet, two technologies are needed: gateway routers and supporting IP-based protocols designed for efficiency. This part explores the role of router technologies at the edge for bridging sensors on a PAN network to the Internet. The role of the router is especially important in securing, managing, and steering data. Edge routers orchestrate and monitor underlying mesh networks and balance and level data quality. The privatization and security of data is also critical. Part 3 will explore the router role in creating virtual private networks, virtual LANs, and software-defined wide area networks. There literally may be thousands of nodes serviced by a single edge router, and in a sense, it serves as an extension to the cloud, as we will see in the *Chapter 11*, *Cloud and Fog Topologies*.

This part continues with the protocols used in IoT communication between nodes, routers, and clouds. The IoT has given way to new protocols rather than the legacy HTTP and SNMP types of messaging used for decades. IoT data needs efficient, power-aware, and low-latency protocols that can be easily steered and secured in and out of the cloud. This part explores protocols such as the pervasive MQTT, as well as AMPQ and CoAP. Examples are given to illustrate their use and efficiency.

Part 4 – Compute, analytics, and machine learning

At this point, we must consider what to do with the data streaming in from edge nodes into a cloud service. First, we begin by talking about the aspects of cloud architectures such as SaaS, IaaS, and PaaS systems. An architect needs to understand the data flow and typical design of cloud services (what they are and how they are used). We use OpenStack as a model of cloud design and explore the various components from ingestor engines to data lakes to analytics engines.

Understanding the constraints of cloud architectures is also important to make a good judgment on how a system will deploy and scale. An architect must also understand how latency can affect an IoT system. Alternatively, not everything belongs in the cloud. There is a measurable cost in moving all IoT data to a cloud versus processing it at the edge (edge processing) or extending cloud services downward into an edge computing device (fog computing). This part dives deep into new standards of fog computing such as the OpenFog architecture.

Data that has been transformed from a physical analog event to a digital signal may have actionable consequences. This is where the analytics and rules engines of the IoT come in to play. The level of sophistication for an IoT deployment is dependent on the solution being architected. In some situations, a simple rules engine looking for anomalous temperature extremes can easily be deployed on an edge router monitoring several sensors. In other situations, a massive amount of structured and unstructured data may be streaming in real time to a cloud-based data lake, and require both fast processing for predictive analytics and long-range forecasting using advanced machine learning models, such as recurrent neural networks in a time-correlated signal analysis package. This part details the uses and constraints of analytics from complex event processors to Bayesian networks to the inference and training of neural networks.

Part 5 – Threat and security in IoT

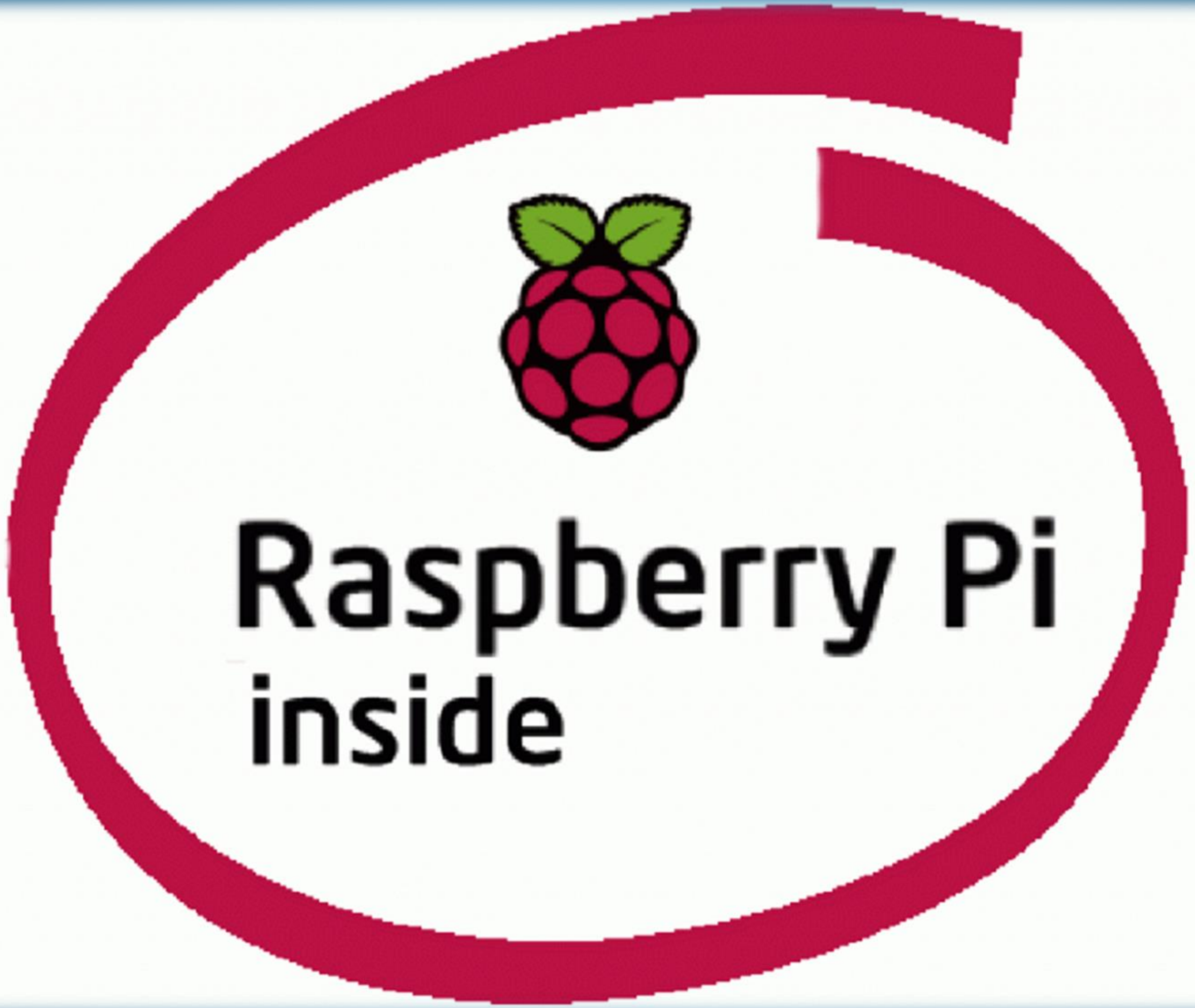
We conclude the book with a survey of IoT compromises and attacks. In many cases, IoT systems will not be secured in a home, or in a company. They will be in public, in very remote areas, in moving vehicles, or even inside a person. The IoT represents the single biggest attack surface for any type of cyberattack. We have seen countless academic hacks, well-organized cyber assaults, and even nation-state security breaches with IoT devices being the target. Part 5 will detail several aspects of such breaches and the types of remediation any architect must consider when making a consumer or enterprise IoT deployment a good citizen of the Internet. We explore the proposed congressional act to secure the IoT and understand the motivation and impact of such a government mandate.

This part will checklist the typical security provisions needed for IoT, or any network component. Details of new technologies such as blockchains and software-defined perimeters will also be explored to provide insight into future technologies that will be needed to secure the IoT.

Summary

This book will bridge the spectrum of technologies that comprise edge computing and the IoT. In this chapter, we summarized the domains and topics covered in the book. An architect must be cognizant of the interactions between these disparate engineering disciplines to build a system that is scalable, robust, and optimized. An architect will also be called upon to provide supporting evidence that the IoT system provides a value to the end user or the customer. Here, we learned about the application of Metcalfe's and Beckstrom's laws as tools for supporting an IoT deployment.

In the next chapters, we will learn about communication from sensors and edge nodes to the Internet and cloud. First, we will examine the fundamental theory behind radio signals and systems and their constraints and limits, and then we will dive into near-range and long-range wireless communication.


Raspberry Pi
inside

Connecting Raspberry pi via SSH

Connecting to your Raspberry Pi via SSH

The Raspberry Pi can be controlled like any other Desktop computer using a keyboard, mouse, and monitor. However, there are also various ways to command the Raspberry Pi remotely, of which SSH is one of the best and often used.

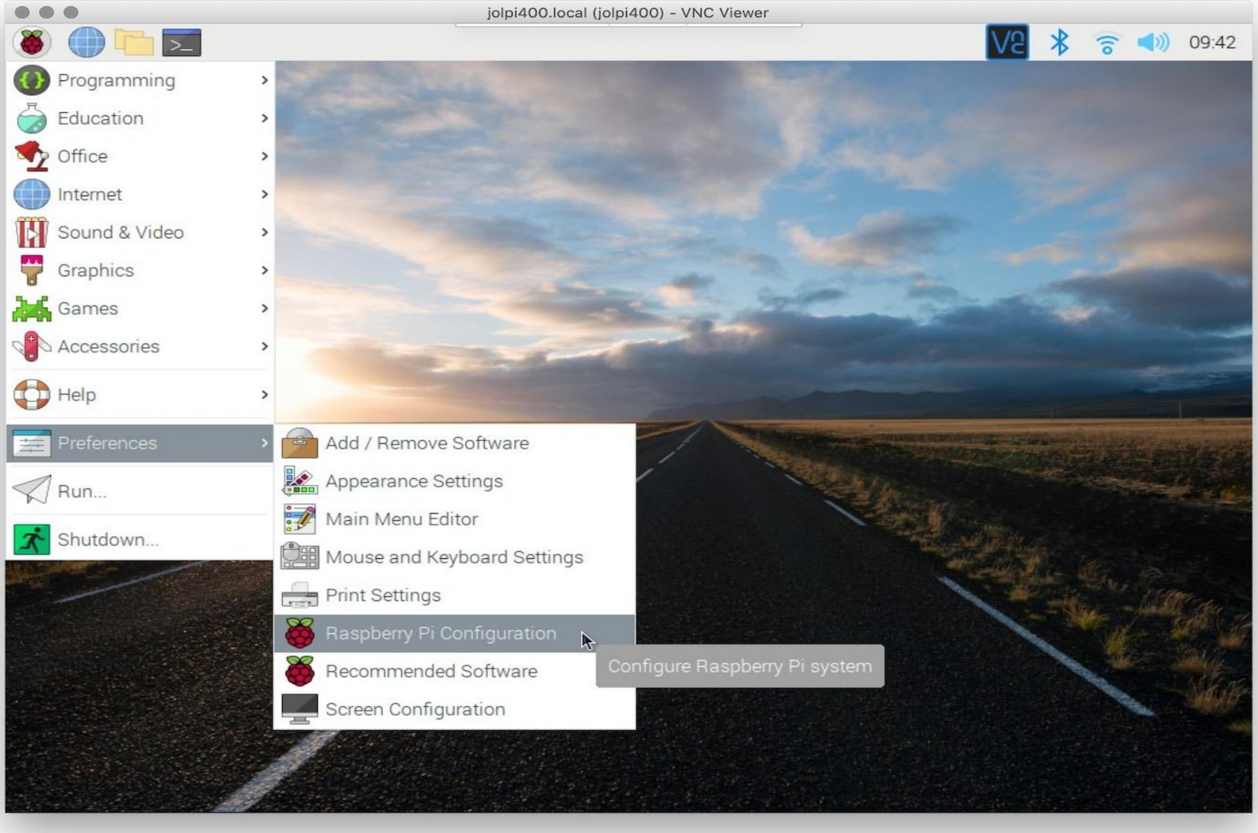
TABLE OF CONTENTS

What is SSH

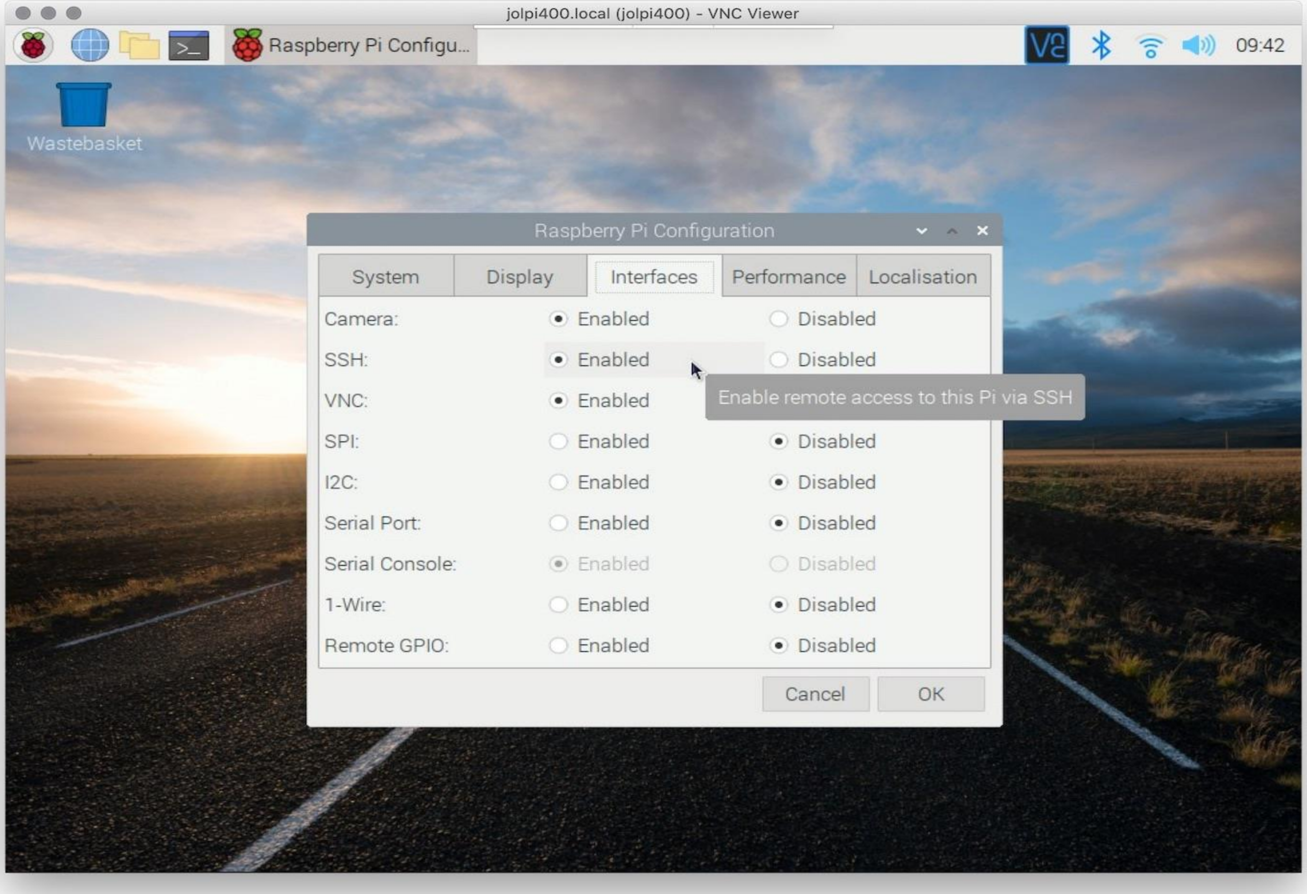
- Secure Shell (SSH) enables you to access the command line of a Raspberry Pi from another computer or device on the same network. This is very handy for quickly installing software or editing configuration files. SSH is pre-installed on Linux, Mac and some Windows operating systems and can also be installed on mobile devices.

Enable SSH On the Raspberry pi

By default , SSH disabled on the raspberry pi .It is however very easy to enable it, both using the desktop and via the terminal .To enable SSH via desktop ,go to start menu,> preferences > raspberry pi configuration.


jolpi400.local (jolpi400) - VNC Viewer
09:42
Programming
Education
Office
Internet
Sound & Video
Graphics
Games
Accessories
Help
Preferences
Run...
Shutdown...
Add / Remove Software
Appearance Settings
Main Menu Editor
Mouse and Keyboard Settings
Print Settings
Raspberry Pi Configuration
Recommended Software
Screen Configuration
Configure Raspberry Pi system

No



Raspberry Pi Configuration

Camera: ● Enabled ○ Disabled

SSH: ● Enabled ○ Disabled

VNC: ● Enabled

Enable remote access to this Pi via SSH

SPI: ○ Enabled ● Disabled

I2C: ○ Enabled ● Disabled

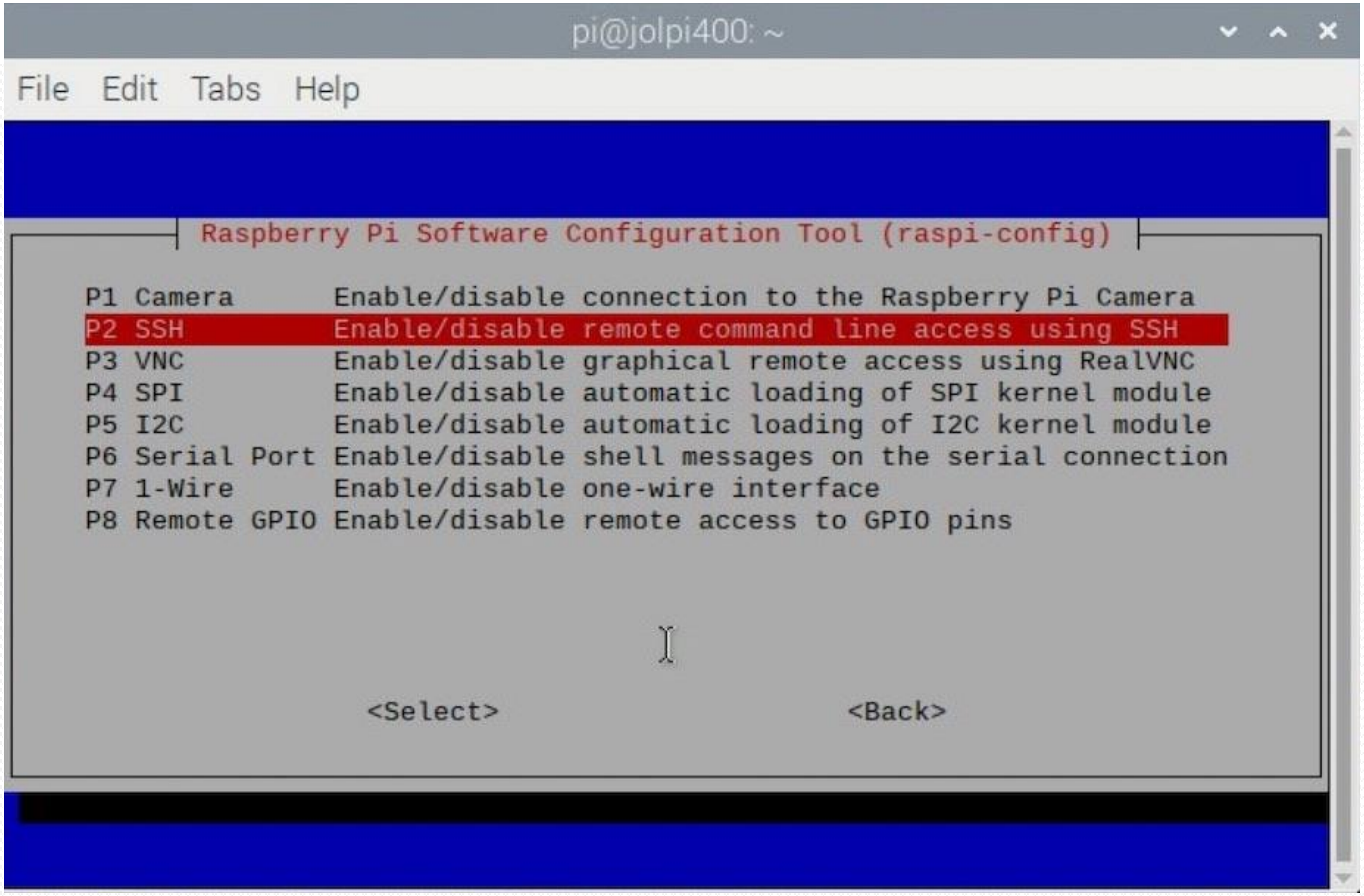
Serial Port: ○ Enabled ● Disabled

Serial Console: ● Enabled ○ Disabled

1-Wire: ○ Enabled ● Disabled

Remote GPIO: ○ Enabled ● Disabled

To enable SSH via the terminal, open a terminal window and enter sudo raspi-config. Now with the arrows select Interfacing Options, navigate to and select SSH, choose Yes, and select Ok.



```
Raspberry Pi Software Configuration Tool (raspi-config)

P1 Camera      Enable/disable connection to the Raspberry Pi Camera
P2 SSH         Enable/disable remote command line access using SSH
P3 VNC         Enable/disable graphical remote access using RealVNC
P4 SPI         Enable/disable automatic loading of SPI kernel module
P5 I2C         Enable/disable automatic loading of I2C kernel module
P6 Serial Port Enable/disable shell messages on the serial connection
P7 1-Wire      Enable/disable one-wire interface
P8 Remote GPIO Enable/disable remote access to GPIO pins

             <Select>                     <Back>
```

Connecting via SSH

- Now SSH is enabled, we need to know the hostname of the Raspberry Pi or use its IP address to connect to it.
- Once enabled, you can connect to your raspberry pi from your computer.

- Here are a few tools you can use to do this depending on your operating system.

Enable SSH on the host device

- SSH is standard available on Linux distributions and on Mac so should work automatically.
- To enable SSH on a Windows 10 computer, make sure it has Update or later and go to settings>Apps > features >Manage optional features>Add a feature, and choose to install OpenSSH Client.

Passwordless SSH

When using SSH, each time you connect you will be asked for the password of your Raspberry Pi.

In some cases it may be preferable to access your Raspberry Pi from another computer without a password, such as to (automatically) send files using rsync To enable password-less access with SSH you will need to generate an SSH key.

What we are going to discuss

- What is raspberrypi?
- History of raspberrypia?
- Founder of raspberrypia?
- Goal of raspberrrypia?
- Hardware layout?
- Versions of Raspberry pi?
- Advantages
- Applications

What is raspberrypi?

- The Raspberry Pi is **a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse**.
- Pi project was initiated by Eben Upton and his colleagues at the University of Cambridge's Computer Laboratory
- **2008:** The first prototype was developed

History of Raspberry Pi Single-Board Computers

February 2012

Raspberry Pi Model B

- Broadcom BCM2835 SoC
- 700 MHz single-core ARM CPU
- 512 MB RAM

The first Raspberry Pi single-board computer to enter production.

February 2013

Raspberry Pi Model A

- Broadcom BCM2835 SoC
- 700 MHz single-core ARM CPU
- 256 MB RAM

Designed to be more affordable and compact, the Model A didn't include network adapters.

July 2014

Raspberry Pi Model B+

- Broadcom BCM2835 SoC
- 700 MHz single-core ARM CPU
- 512 MB RAM

GPIO pins increased from 26 to 40, switched from SD to microSD cards, and added HDMI.

November 2014

Raspberry Pi Model A+

- Broadcom BCM2835 SoC
- 700 MHz single-core ARM CPU
- 256 MB RAM

GPIO pins increased from 26 to 40, switched from SD to microSD cards, and added HDMI.

February 2015

Raspberry Pi 2 Model B

- Broadcom BCM2835 SoC
- 900 MHz quad-core ARM CPU
- 1 GB RAM

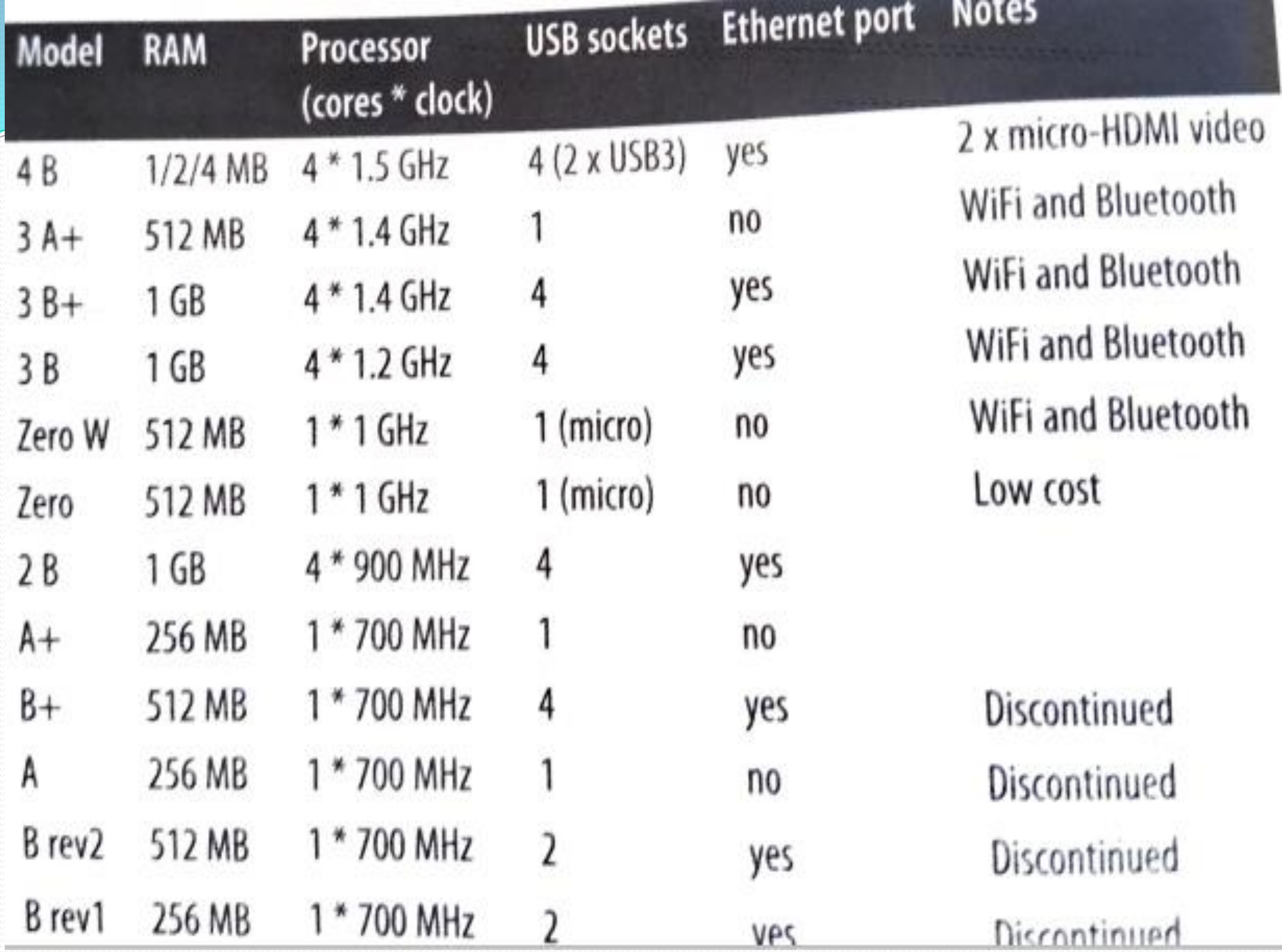
Upgraded to a A 900MHz quad-core ARM CPU and 1 GB of RAM.

November 2015

Raspberry Pi Zero

- Broadcom BCM2835 SoC
- 1 GHz single-core ARM CPU
- 512 MB RAM

At half the size of the Model A form factor, the Raspberry Pi designed the Zero for small devices.



Model	RAM	Processor (cores * clock)	USB sockets	Ethernet port	Notes
4 B	1/2/4 MB	4 * 1.5 GHz	4 (2 x USB3)	yes	2 x micro-HDMI video
3 A+	512 MB	4 * 1.4 GHz	1	no	WiFi and Bluetooth
3 B+	1 GB	4 * 1.4 GHz	4	yes	WiFi and Bluetooth
3 B	1 GB	4 * 1.2 GHz	4	yes	WiFi and Bluetooth
Zero W	512 MB	1 * 1 GHz	1 (micro)	no	WiFi and Bluetooth
Zero	512 MB	1 * 1 GHz	1 (micro)	no	Low cost
2 B	1 GB	4 * 900 MHz	4	yes	
A+	256 MB	1 * 700 MHz	1	no	
B+	512 MB	1 * 700 MHz	4	yes	Discontinued
A	256 MB	1 * 700 MHz	1	no	Discontinued
B rev2	512 MB	1 * 700 MHz	2	yes	Discontinued
B rev1	256 MB	1 * 700 MHz	2	yes	Discontinued

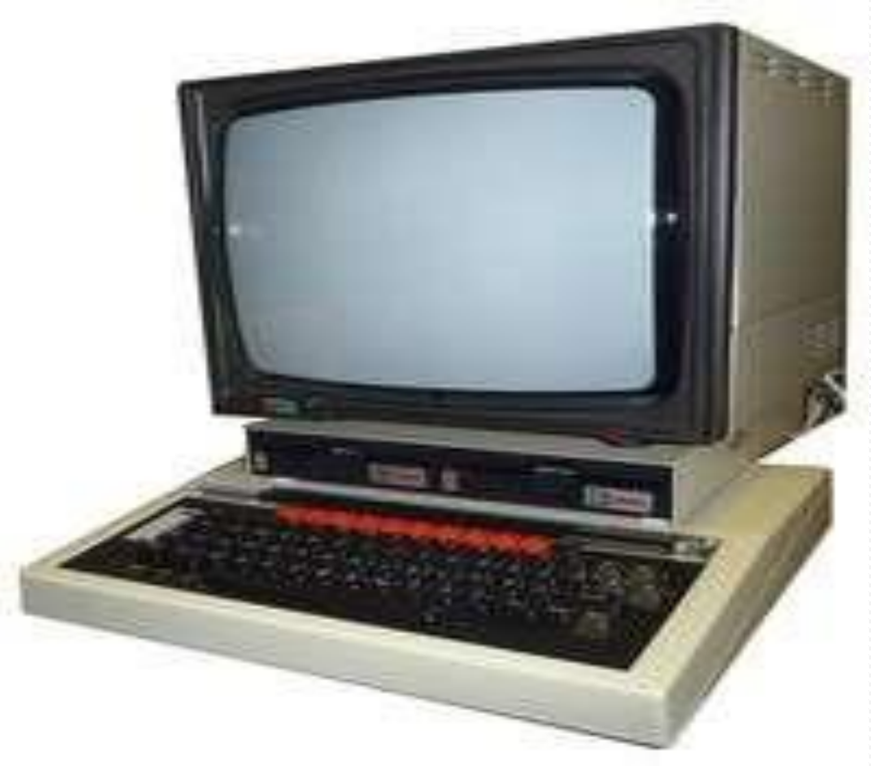
Present version of raspberry pi

- The latest version of Raspberry pi is pi 5 was announced on September 28, 2023
- Improvements in hardware and software reportedly make the Pi 5 more than twice as powerful as the Pi 4.
- Comes with an I/O-controller designed in-house, a power button.

Who invented Raspberry pi and when?

- **Eben Upton is a British engineer, creator of the Raspberry Pi and the Raspberry Pi Foundation**. He studied physics and engineering at the Cambridge university before working for prestigious companies, like Broadcom, Intel and IBM.
- **The Raspberry Pi story started in 2006 with the creation of the first prototypes inspired from the BBC Micro.**

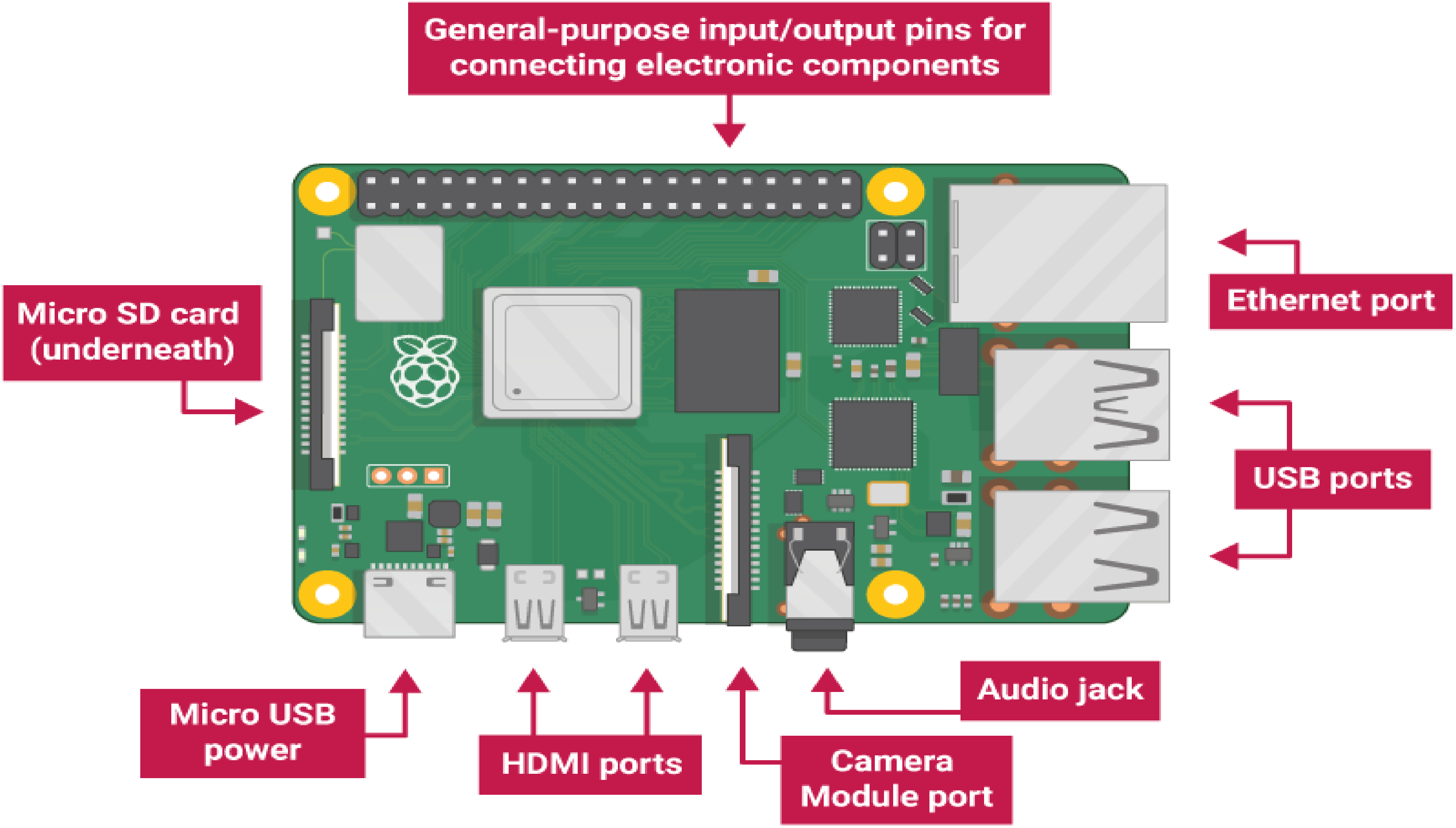
This is BBC Micro picture



Goal of raspberrypia?

The main goal was to help young people to discover computers at low cost

Raspberrypia-diagram



General-purpose input/output pins for connecting electronic components

Micro SD card (underneath)

Ethernet port

USB ports

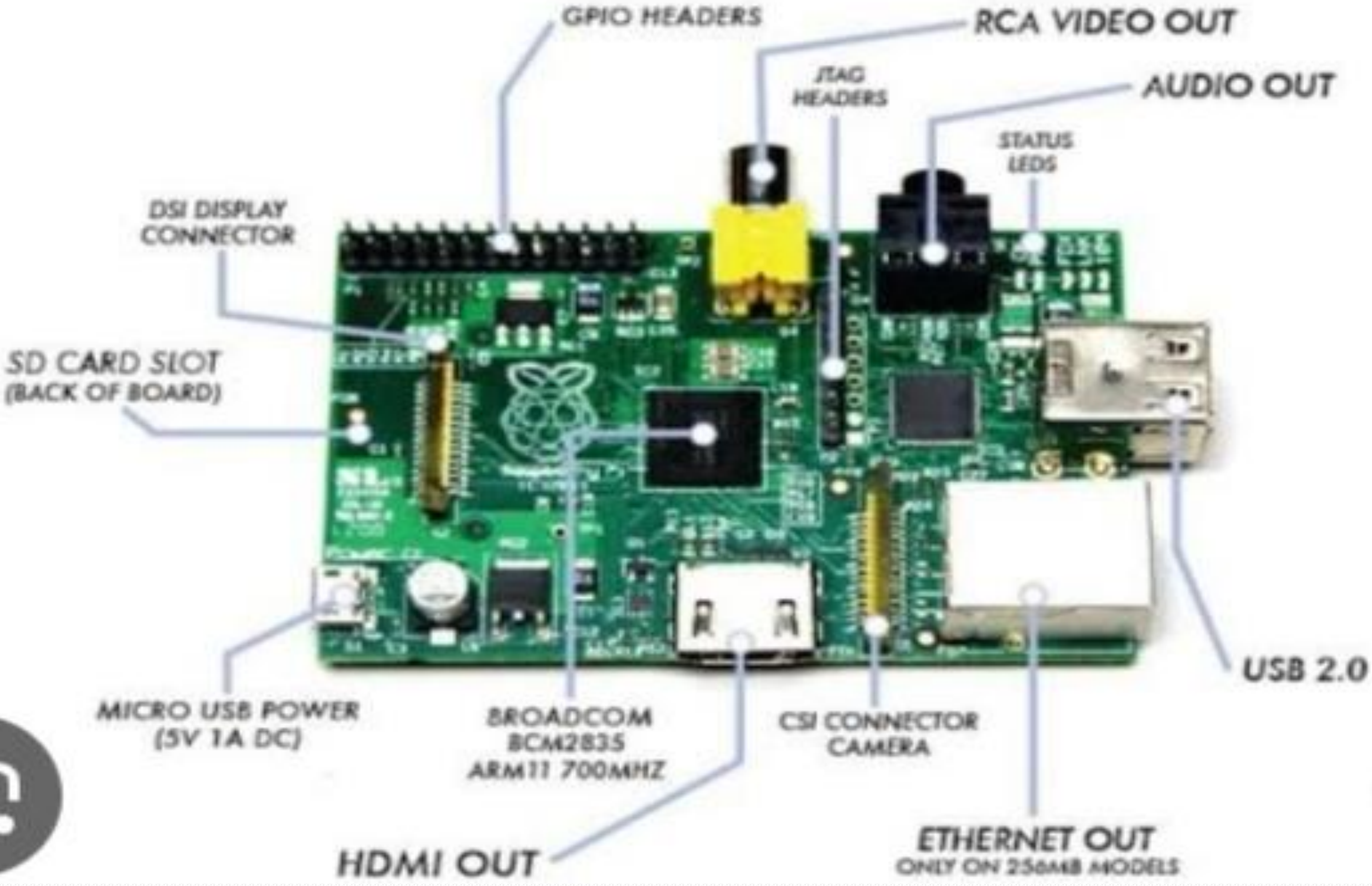
Micro USB power

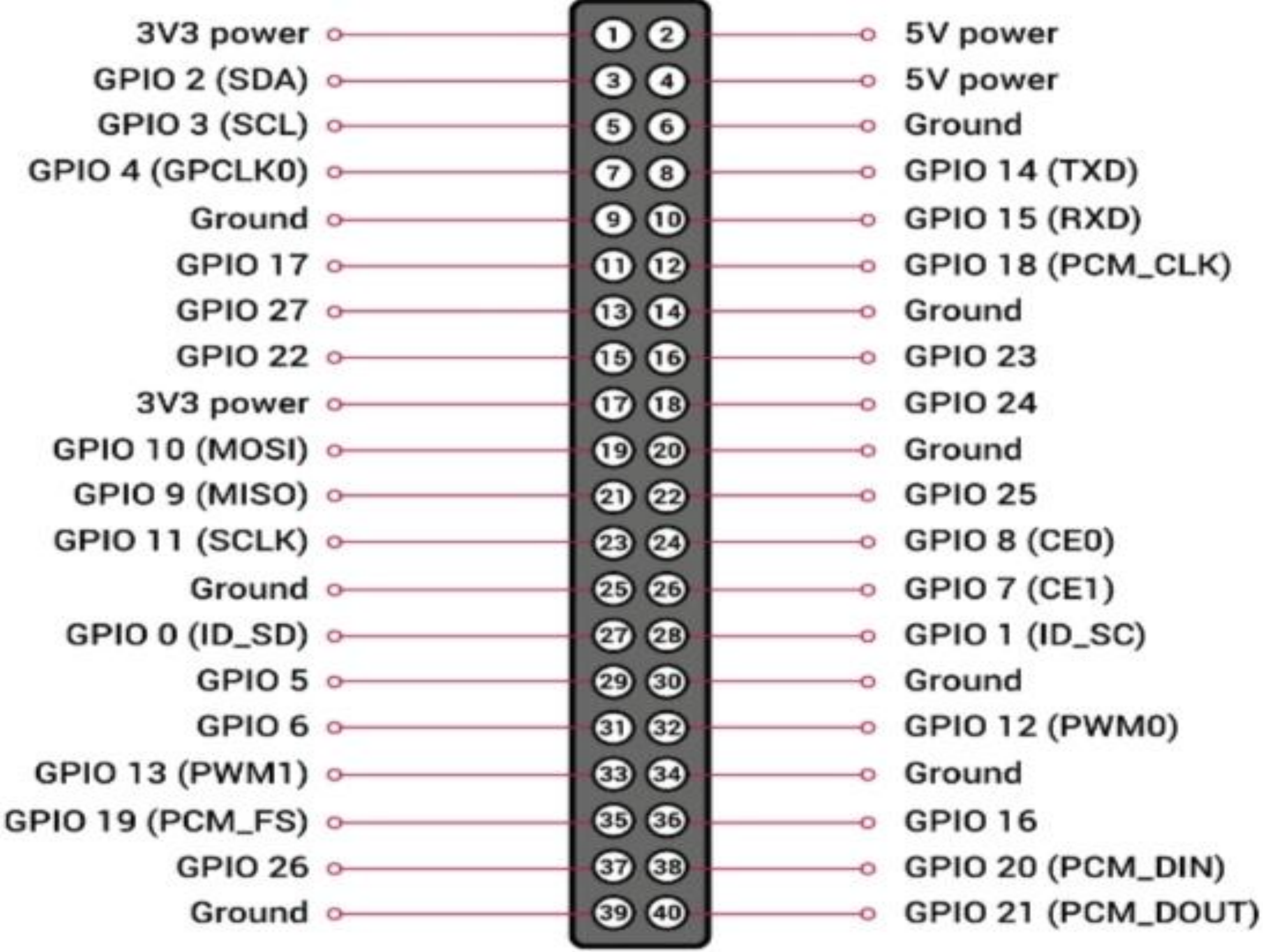
HDMI ports

Camera Module port

Audio jack

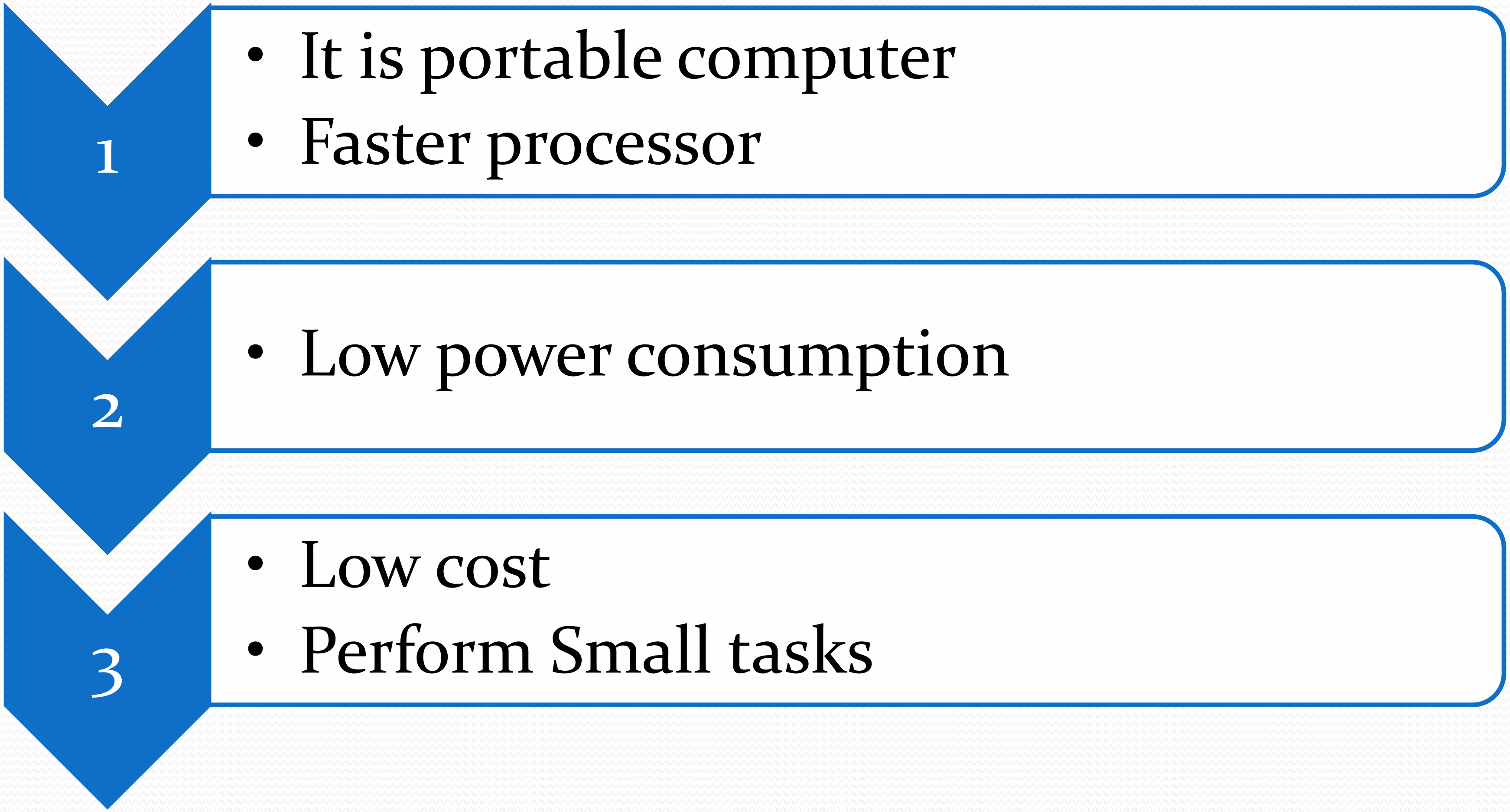
Raspberry Pi hardware


GPIO HEADERS
RCA VIDEO OUT
JTAG HEADERS
AUDIO OUT
STATUS LEDS
DSI DISPLAY CONNECTOR
SD CARD SLOT (BACK OF BOARD)
USB 2.0
MICRO USB POWER (5V 1A DC)
BROADCOM BCM2835 ARM11 700MHZ
CSI CONNECTOR CAMERA
HDMI OUT
ETHERNET OUT ONLY ON 256MB MODELS



3V3 power	1	2	5V power
GPIO 2 (SDA)	3	4	5V power
GPIO 3 (SCL)	5	6	Ground
GPIO 4 (GPCLK0)	7	8	GPIO 14 (TXD)
Ground	9	10	GPIO 15 (RXD)
GPIO 17	11	12	GPIO 18 (PCM_CLK)
GPIO 27	13	14	Ground
GPIO 22	15	16	GPIO 23
3V3 power	17	18	GPIO 24
GPIO 10 (MOSI)	19	20	Ground
GPIO 9 (MISO)	21	22	GPIO 25
GPIO 11 (SCLK)	23	24	GPIO 8 (CE0)
Ground	25	26	GPIO 7 (CE1)
GPIO 0 (ID_SD)	27	28	GPIO 1 (ID_SC)
GPIO 5	29	30	Ground
GPIO 6	31	32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33	34	Ground
GPIO 19 (PCM_FS)	35	36	GPIO 16
GPIO 26	37	38	GPIO 20 (PCM_DIN)
Ground	39	40	GPIO 21 (PCM_DOUT)

Advantages of raspberrypi



1

- It is portable computer
- Faster processor

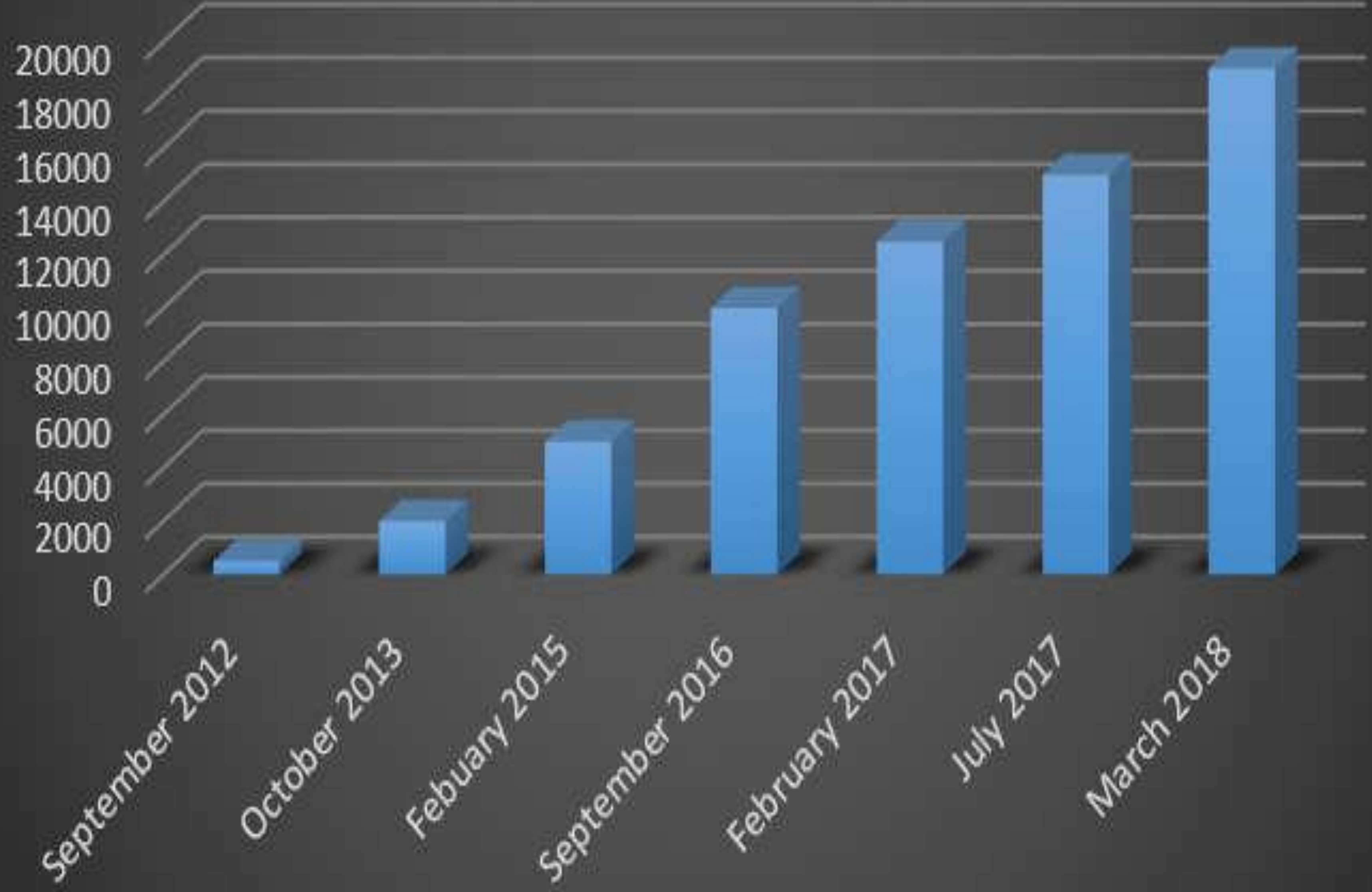
2

- Low power consumption

3

- Low cost
- Perform Small tasks

Raspberry Pi sales in thousands



Applications

- Low cost survelliance system
- Super computer using raspberrypi
- Pi pad tablet
- Pi coding websites

Implementing a project using a Raspberry Pi involves several steps, from setting up the hardware to writing and running code. Here's a general outline of how to implement a project using a Raspberry Pi as a microcomputer:

1. **Gather Hardware:** Obtain the necessary hardware components for your project. This can include a Raspberry Pi board, power supply, microSD card, peripherals (keyboard, mouse, monitor), sensors, actuators, and any other components specific to your project.
2. **Prepare the MicroSD Card:** Download the official Raspberry Pi OS (formerly known as Raspbian) from the Raspberry Pi website. Use the Raspberry Pi Imager tool to write the OS image to the microSD card.
3. **Initial Setup:** Insert the microSD card into the Raspberry Pi and power it up. Follow the on-screen instructions to complete the initial setup, including configuring language, time zone, and setting up a password. You can also enable SSH and VNC for remote access.
4. **Connect Peripherals:** If your project requires peripherals like a monitor, keyboard, and mouse, connect them to the Raspberry Pi. Alternatively, you can set up remote access via SSH or VNC.
5. **Software Updates:** After the initial setup, it's a good idea to update the software packages on the Raspberry Pi. Open a terminal and run the following commands:
6. **Install Libraries and Dependencies:** Depending on your project requirements, you might need to install additional libraries or software packages. Use package managers like `apt` or `pip` to install the necessary tools.
7. **Write and Run Code:** Develop your project's code using a programming language of your choice (Python is commonly used for Raspberry Pi projects due to its ease of use). Write scripts to interact with sensors, control actuators, and perform any desired functionalities.
8. **GPIO Programming:** Raspberry Pi's General Purpose Input/Output (GPIO) pins allow you to interface with external components like sensors, LEDs, and motors. Libraries like RPi.GPIO make it easy to control these pins. Here's an example to toggle an LED connected to GPIO pin 17:
9. **Testing and Debugging:** Test your code and hardware components thoroughly. Use print statements, debugging tools, and error handling to identify and fix issues.
10. **Deployment:** Once your project is working as expected, you can deploy it in its intended environment. This could involve securing components, setting up power management, and ensuring the project runs reliably.
11. **Monitoring and Maintenance:** Monitor the performance of your project over time. Make sure to keep software and libraries updated, especially if security updates are released.

12. **Documentation:** Document your project's setup, code, wiring diagrams, and any other relevant information. This will be helpful for future reference and sharing your project with others.

Remember that the specific steps and details will vary depending on the nature of your project. Whether it's a home automation system, a weather station, a robot, or any other application, the Raspberry Pi's versatility and community support can be invaluable resources throughout your implementation process.

Device Interfacing using Rasberry Pi:

Interfacing devices with a Raspberry Pi involves connecting external hardware components such as sensors, actuators, and displays to the GPIO pins of the Raspberry Pi. Here's a general guide on how to interface devices with a Raspberry Pi:

1. **Choose Your Device:** Decide on the device you want to interface with your Raspberry Pi. This could be a sensor (e.g., temperature sensor, motion sensor), an actuator (e.g., LED, motor), a display (e.g., LCD screen), or any other electronic component.
2. **Understand GPIO Pins:** The Raspberry Pi has a set of GPIO pins that can be used for digital input and output. Make sure you understand the pin numbering and capabilities of these GPIO pins. The pins can be used for both digital (on/off) and analog (variable) signals.
3. **Check Pinout Diagram:** Refer to the official Raspberry Pi pinout diagram or use online resources to understand the layout of the GPIO pins and their functions.
4. **Wiring and Connection:** Wire your device according to its specifications and the Raspberry Pi's GPIO pinout. You might need jumper wires, a breadboard, and possibly resistors or level shifters, depending on the voltage levels and requirements of your device.
5. **GPIO Libraries:** To control and read data from the GPIO pins, you'll need to use programming libraries. The `RPi.GPIO` library is commonly used for Python programming. Install it if not already installed:
6. **Write Code:** Create a Python script to interact with your device using the GPIO pins. Here's a simple example of toggling an LED connected to GPIO pin 17:
7. **Test and Debug:** Run your code and test the device's behaviour. Debug any issues that arise, such as incorrect wiring, incorrect pin configuration, or errors in your code.
8. **Interface with Sensors:** Interfacing with sensors typically involves reading data from their output pins. For example, if you're using a digital temperature sensor:

9. **Interface with Displays:** Interfacing with displays, such as LCD screens, might require additional libraries specific to the display type. Follow the documentation provided by the display manufacturer to correctly wire and program the display.
10. **Advanced Interfacing:** Depending on your project's complexity, you might need to use analog-to-digital converters (ADCs) for analog sensors, motor driver circuits for controlling motors, and other hardware components.
11. **Safety Considerations:** When working with electronic components, especially devices that involve higher voltages or currents, prioritize safety. Double-check your wiring, use appropriate components like resistors, and avoid short circuits.
12. **Documentation:** Document your wiring diagrams, code, and any modifications you make. This documentation will be helpful for troubleshooting and future reference.

Remember that the specifics of interfacing devices will depend on the devices themselves and your project's requirements. Always refer to the datasheets and documentation of the components you're using for accurate and detailed information.

Edge to Cloud Protocols:

Edge-to-cloud communication protocols are crucial for connecting devices at the edge of a network (like sensors and IoT devices) with cloud-based services and applications. These protocols facilitate the transfer of data, commands, and information between the edge devices and cloud platforms. Here are some commonly used edge-to-cloud communication protocols:

1. **MQTT (Message Queuing Telemetry Transport):** MQTT is a lightweight publish-subscribe messaging protocol designed for IoT applications. It is well-suited for scenarios where devices need to send data to the cloud or receive commands from the cloud. MQTT uses a broker-based architecture, allowing devices to publish data to topics and subscribe to topics to receive data.
2. **CoAP (Constrained Application Protocol):** CoAP is another lightweight protocol designed for constrained devices and networks. It is specifically optimized for resource-constrained environments and supports both request-response and publish-subscribe models. CoAP is often used in scenarios where devices need to communicate over low-power networks.
3. **HTTP/HTTPS (Hypertext Transfer Protocol/Secure):** While not specifically designed for IoT, HTTP and HTTPS are widely used for communication between edge devices and the cloud. HTTP can be used for sending RESTful API requests and receiving responses, making it suitable for applications that require cloud-based data retrieval and control.

4. **AMQP (Advanced Message Queuing Protocol):** AMQP is a versatile protocol that provides a standardized way for different applications to communicate using a message-oriented paradigm. It supports both publish-subscribe and queuing models, making it suitable for various edge-to-cloud communication scenarios.
5. **DDS (Data Distribution Service):** DDS is a protocol used in real-time and mission-critical systems where reliable and high-performance data exchange is essential. It supports data-centric publish-subscribe communication and is often used in applications such as industrial automation and aerospace.
6. **WebSocket:** WebSocket is a communication protocol that provides full-duplex, bidirectional communication channels over a single TCP connection. It's particularly useful for applications requiring real-time data updates or interactive communication between edge devices and the cloud.
7. **AMT (Asynchronous Messaging Transport):** AMT is a protocol designed to efficiently transport large volumes of data between edge devices and cloud services. It's optimized for handling streaming data and is suitable for applications such as multimedia streaming and remote monitoring.
8. **DDS (Data Distribution Service):** DDS is a protocol that focuses on real-time data distribution and communication in distributed systems. It's commonly used in scenarios where high-performance and low-latency communication are critical, such as industrial automation and military applications.
9. **Sigfox and LoRaWAN:** These are proprietary low-power, wide-area (LPWA) network protocols designed for IoT devices with long-range connectivity. They are often used for edge-to-cloud communication in applications requiring extended battery life and communication over large areas.
10. **Bluetooth and BLE (Bluetooth Low Energy):** For short-range communication between edge devices and cloud gateways, Bluetooth and BLE are commonly used. They are suitable for applications like home automation, wearables, and proximity-based interactions.

When selecting an edge-to-cloud communication protocol, consider factors such as data volume, latency requirements, power constraints, security, and compatibility with your cloud platform. Your choice will depend on the specific needs of your IoT project and the technology stack you're working with.

Edge-to-cloud communication protocols:

Edge-to-cloud communication protocols are crucial for connecting devices at the edge of a network (like sensors and IoT devices) with cloud-based services and applications. These protocols facilitate the transfer of data, commands, and information between the edge devices and cloud platforms. Here are some commonly used edge-to-cloud communication protocols:

1. **MQTT (Message Queuing Telemetry Transport):** MQTT is a lightweight publish-subscribe messaging protocol designed for IoT applications. It is well-suited for scenarios where devices need to send data to the cloud or receive commands from the cloud. MQTT uses a broker-based architecture, allowing devices to publish data to topics and subscribe to topics to receive data.
2. **CoAP (Constrained Application Protocol):** CoAP is another lightweight protocol designed for constrained devices and networks. It is specifically optimized for resource-constrained environments and supports both request-response and publish-subscribe models. CoAP is often used in scenarios where devices need to communicate over low-power networks.
3. **HTTP/HTTPS (Hypertext Transfer Protocol/Secure):** While not specifically designed for IoT, HTTP and HTTPS are widely used for communication between edge devices and the cloud. HTTP can be used for sending RESTful API requests and receiving responses, making it suitable for applications that require cloud-based data retrieval and control.
4. **AMQP (Advanced Message Queuing Protocol):** AMQP is a versatile protocol that provides a standardized way for different applications to communicate using a message-oriented paradigm. It supports both publish-subscribe and queuing models, making it suitable for various edge-to-cloud communication scenarios.
5. **DDS (Data Distribution Service):** DDS is a protocol used in real-time and mission-critical systems where reliable and high-performance data exchange is essential. It supports data-centric publish-subscribe communication and is often used in applications such as industrial automation and aerospace.
6. **WebSocket:** WebSocket is a communication protocol that provides full-duplex, bidirectional communication channels over a single TCP connection. It's particularly useful for applications requiring real-time data updates or interactive communication between edge devices and the cloud.
7. **AMT (Asynchronous Messaging Transport):** AMT is a protocol designed to efficiently transport large volumes of data between edge devices and cloud services. It's optimized for handling streaming data and is suitable for applications such as multimedia streaming and remote monitoring.
8. **DDS (Data Distribution Service):** DDS is a protocol that focuses on real-time data distribution and communication in distributed systems. It's commonly used in scenarios where high-performance and low-latency communication are critical, such as industrial automation and military applications.
9. **Sigfox and LoRaWAN:** These are proprietary low-power, wide-area (LPWA) network protocols designed for IoT devices with long-range connectivity. They are often used for edge-to-cloud communication in applications requiring extended battery life and communication over large areas.
10. **Bluetooth and BLE (Bluetooth Low Energy):** For short-range communication between edge devices and cloud gateways, Bluetooth and BLE

are commonly used. They are suitable for applications like home automation, wearables, and proximity-based interactions.

When selecting an edge-to-cloud communication protocol, consider factors such as data volume, latency requirements, power constraints, security, and compatibility with your cloud platform. Your choice will depend on the specific needs of your IoT project and the technology stack you're working with.

MQTT (Message Queuing Telemetry Transport):

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol designed for efficient communication between devices in IoT (Internet of Things) and other resource-constrained environments. It follows a publish-subscribe messaging pattern, allowing devices to exchange data and commands via a central message broker. MQTT is well-suited for scenarios where low bandwidth, low power consumption, and reliable communication are important.

Key features of MQTT include:

1. **Publish-Subscribe Model:** In MQTT, devices communicate through topics. Publishers send messages to specific topics, and subscribers receive messages from topics they are interested in. This decoupled approach allows devices to communicate without knowing each other's identities.
2. **Quality of Service (QoS) Levels:** MQTT supports different levels of message delivery assurance:
    - QoS 0: The message is delivered at most once, without any acknowledgment. This level is suitable for scenarios where message loss is acceptable.
    - QoS 1: The message is delivered at least once, and the sender waits for an acknowledgment (PUBACK) from the receiver. If no acknowledgment is received, the message is resent.
    - QoS 2: The message is delivered exactly once by using a four-step handshake process. This ensures the message is received only once.
3. **Retained Messages:** A retained message is a message that is stored on the broker and sent to new subscribers immediately upon subscribing. This is useful for sending status updates or configuration information to new devices joining the network.
4. **Last Will and Testament (LWT):** Clients can specify a "last will" message that the broker will send if the client unexpectedly disconnects. This can be used to notify other devices of a client's offline status.
5. **Low Bandwidth Usage:** MQTT messages have a small overhead, making it efficient for low-bandwidth and high-latency networks.

6. **Keep-Alive Mechanism:** Clients periodically send "ping" messages to the broker to keep the connection alive. If the broker doesn't receive a ping within a specified time, it can assume the client has disconnected.
7. **Security:** MQTT supports various authentication and security mechanisms, including username/password authentication, SSL/TLS encryption, and more.
8. **Wildcards:** MQTT allows the use of wildcards to subscribe to multiple topics with a single subscription. The two wildcards are '+' (matches a single level of a topic) and '#' (matches multiple levels).

To use MQTT, you typically need three components:

- **Client:** The device or application that publishes or subscribes to MQTT messages.
- **Broker:** The central server that acts as an intermediary between clients. It receives messages from publishers and forwards them to subscribers.
- **Topic:** A string identifier that specifies the category of the message. Clients can publish messages to topics and subscribe to topics to receive messages.

Overall, MQTT's lightweight design and support for various QoS levels make it a popular choice for IoT applications, home automation, real-time monitoring, and other scenarios where efficient and reliable communication between devices is crucial.

MQTT (Message Queuing Telemetry Transport) follows a client-server or broker-based architecture, where clients (devices or applications) communicate with each other through a central MQTT broker. Here are the key components and details of MQTT architecture:

1. **Client:** MQTT clients are the endpoints that send or receive messages. Clients can be devices, sensors, applications, or any system capable of MQTT communication. Clients can be categorized into two types: publishers and subscribers.
    - **Publisher:** A publisher is an MQTT client that sends messages to the MQTT broker. Publishers specify a topic for each message they publish. Topics are used to categorize messages.
    - **Subscriber:** A subscriber is an MQTT client that receives messages from the MQTT broker. Subscribers specify the topics they are interested in and subscribe to those topics. When a message is published to a subscribed topic, the broker forwards it to all interested subscribers.

2. **Broker:** The MQTT broker is the central server that acts as an intermediary between publishers and subscribers. It is responsible for routing messages from publishers to subscribers based on topic subscriptions. The broker manages the MQTT communication and ensures the proper delivery of messages. Some key functions of the MQTT broker include:
    - **Topic Routing:** The broker keeps track of which clients are subscribed to which topics. When a message is published to a topic, the broker forwards it to all clients that have subscribed to that topic.
    - **Quality of Service (QoS) Handling:** The broker handles the QoS levels specified by clients during message publication and ensures that messages are delivered according to the desired QoS level.
    - **Retained Messages:** The broker can retain the last message sent on a specific topic, allowing new subscribers to receive the last known value for that topic.
    - **Last Will and Testament (LWT):** The broker manages LWT messages, which are sent to a specified topic if a client disconnects unexpectedly.
    - **Security:** Brokers can enforce security mechanisms such as authentication, authorization, and encryption when clients connect to them. Some brokers support TLS/SSL for secure communication.
    - **Session Management:** MQTT brokers can manage client sessions, including keeping track of active client connections and handling client reconnections.
3. **Topic:** Topics are used to categorize and organize messages in MQTT. They are hierarchical in nature and allow for structured messaging. Topics are represented as strings, and clients can subscribe to and publish to specific topics. For example, a topic hierarchy might look like "home/living-room/temperature."
4. **Quality of Service (QoS):** MQTT supports three levels of QoS, which define the message delivery guarantees between publishers and subscribers:
    - **QoS 0 (At Most Once):** Messages are delivered at most once, with no acknowledgment or guarantee of delivery. This is the fastest but least reliable QoS level.
    - **QoS 1 (At Least Once):** Messages are guaranteed to be delivered at least once, but duplicates may occur. This level ensures message reliability at the cost of potential duplicates.
    - **QoS 2 (Exactly Once):** Messages are guaranteed to be delivered exactly once. This level ensures both reliability and no duplicates but involves the most communication overhead.

In summary, MQTT architecture consists of clients (publishers and subscribers), an MQTT broker, topics for message categorization, and QoS levels for message delivery guarantees. This architecture is designed to be lightweight, efficient, and well-suited for IoT and other applications where efficient messaging is crucial.

MQTT (Message Queuing Telemetry Transport) follows a specific set of state transitions to manage the communication between MQTT clients (publishers and subscribers) and the MQTT broker. Here are the key MQTT state transitions:

1. **Disconnected State (Initial State):**
   - **Description:** When an MQTT client is not connected to the broker, it is in the Disconnected state.
   - **Transitions:** The client can transition to the Connecting state to establish a connection with the broker.
2. **Connecting State:**
   - **Description:** In this state, the MQTT client initiates a connection to the MQTT broker.
   - **Transitions:**
     - If the connection attempt is successful, the client transitions to the Connected state.
     - If the connection attempt fails, the client can retry or may transition back to the Disconnected state.
3. **Connected State:**
   - **Description:** Once the MQTT client successfully connects to the broker, it enters the Connected state. In this state, the client can perform various MQTT operations, such as publishing and subscribing to topics.
   - **Transitions:**
     - The client can initiate a disconnection, which transitions it back to the Disconnected state.
     - If the connection is unexpectedly terminated (e.g., due to a network issue or broker disconnection), the client can transition to the Disconnected state.
4. **Publishing State (Optional):**
   - **Description:** When an MQTT client wants to publish a message, it enters the Publishing state temporarily to send the message to the broker.
   - **Transitions:** After publishing the message, the client returns to the Connected state.
5. **Subscribing State (Optional):**
   - **Description:** When an MQTT client subscribes to one or more topics, it may enter a Subscribing state temporarily to send the subscription request to the broker.
   - **Transitions:** After subscribing to the topics, the client returns to the Connected state.
6. **Disconnected (Clean Disconnect) State:**
   - **Description:** When an MQTT client decides to disconnect gracefully, it transitions to the Disconnected state.

- **Transitions:** The client can initiate a new connection by moving to the Connecting state or remain in the Disconnected state until it decides to reconnect.

7. **Disconnected (Ungraceful Disconnect) State:**
    - **Description:** If the connection is unexpectedly terminated (e.g., due to network issues), the MQTT client transitions to the Disconnected state.
    - **Transitions:** The client typically initiates a reconnection attempt by moving to the Connecting state, allowing it to reestablish communication with the broker.

These state transitions illustrate how MQTT clients manage their connection states and handle reconnections when necessary. MQTT brokers also play a role in managing client sessions and ensuring the appropriate handling of messages and subscriptions during client disconnections and reconnections. Additionally, MQTT supports features like the "Last Will and Testament" (LWT) to handle unexpected client disconnects gracefully.

MQTT (Message Queuing Telemetry Transport) uses a specific packet structure for communication between clients (publishers and subscribers) and the MQTT broker. MQTT packets are used to convey various types of information, including connection requests, published messages, subscriptions, acknowledgments, and more. Here is an overview of the MQTT packet structure:

1. **Fixed Header:** The fixed header appears at the beginning of every MQTT packet and contains essential information about the packet type, flags, and remaining length. It consists of the following components:
    - **Packet Type:** Indicates the type of packet, such as CONNECT, PUBLISH, SUBSCRIBE, etc.
    - **Flags:** Various flags that provide additional control over the packet behavior, depending on the packet type.
    - **Remaining Length:** Specifies the length of the variable header and payload.
2. **Variable Header:** The structure and content of the variable header depend on the specific packet type. It may include information such as the protocol name and version, message identifier, topic name, QoS level, and more.
3. **Payload:** The payload carries the actual data associated with the MQTT packet. The content and structure of the payload vary based on the packet type. For example:
    - **CONNECT Packet:** The payload contains information about the client's identifier, clean session flag, username, password, and more.

    - **PUBLISH Packet:** The payload holds the message topic, message body, QoS level, and retain flag.
    - **SUBSCRIBE Packet:** The payload specifies the list of topics and their corresponding QoS levels that the client wants to subscribe to.

4. **Checksum/CRC (Cyclic Redundancy Check):** Some MQTT packets may include a checksum or CRC to ensure data integrity. This is typically seen in MQTT over WebSocket implementations.

Here's a breakdown of the structure of a few common MQTT packets:

- **CONNECT Packet:** This packet is used to establish a connection between the client and the broker.
    - **Fixed Header:** Packet Type (CONNECT), Flags, Remaining Length
    - **Variable Header:** Protocol Name, Protocol Level, Connect Flags, Keep Alive, Client Identifier, Will Topic, Will Message, Username, Password
    - **Payload:** None
- **PUBLISH Packet:** This packet is used to publish a message to a specific topic.
    - **Fixed Header:** Packet Type (PUBLISH), Flags (QoS, Retain), Remaining Length
    - **Variable Header:** Topic Name, Message Identifier (if QoS > 0)
    - **Payload:** Message Data
- **SUBSCRIBE Packet:** This packet is used by the client to subscribe to one or more topics.
    - **Fixed Header:** Packet Type (SUBSCRIBE), Flags, Remaining Length
    - **Variable Header:** Message Identifier
    - **Payload:** List of Topic Filters and QoS Levels
- **SUBACK Packet:** This packet is sent by the broker to acknowledge the SUBSCRIBE request.
    - **Fixed Header:** Packet Type (SUBACK), Flags, Remaining Length
    - **Variable Header:** Message Identifier
    - **Payload:** List of QoS Levels (acknowledging the requested QoS for each subscribed topic)
- **PUBACK Packet:** This packet is used to acknowledge the receipt of a PUBLISH packet with QoS level 1.
    - **Fixed Header:** Packet Type (PUBACK), Flags, Remaining Length
    - **Variable Header:** Message Identifier
    - **Payload:** None
- **DISCONNECT Packet:** This packet is used to disconnect from the broker.
    - **Fixed Header:** Packet Type (DISCONNECT), Flags, Remaining Length
    - **Variable Header:** None
    - **Payload:** None

The specific packet structure may vary slightly depending on the MQTT version being used (e.g., MQTT v3.1.1 or MQTT v5.0) and the presence of optional fields like user authentication and will messages. The fixed header and variable header together provide the necessary information for processing and routing MQTT packets correctly.

MQTT (Message Queuing Telemetry Transport) is a lightweight and efficient messaging protocol often used in IoT (Internet of Things) and other scenarios where low overhead and reliable communication are essential. MQTT packets are the fundamental units of communication in MQTT, and they have a specific structure. MQTT uses a publish-subscribe model, where clients can publish messages to topics, and other clients can subscribe to topics to receive those messages.

MQTT packets can be classified into four types:

1. **Fixed Header**: This is a mandatory part of every MQTT packet and contains information about the packet type, the QoS (Quality of Service) level, whether the packet is a duplicate, and the length of the remaining variable header and payload.

   The fixed header consists of the following fields:

   - **Packet Type**: A 4-bit field that specifies the type of packet (e.g., CONNECT, PUBLISH, SUBSCRIBE, etc.).
   - **Flags**: Various flags specific to each packet type.
   - **Remaining Length**: A variable length field (1-4 bytes) encoding the length of the variable header and payload.
2. **Variable Header**: This part of the packet structure varies in content and length depending on the packet type. It typically contains information such as protocol version, client ID, topic name, message ID, etc.
3. **Payload**: The payload is optional and carries the actual data of the message, such as the message text in a PUBLISH packet.
4. **Checksum (Optional)**: Depending on the packet type and QoS level, an optional checksum (CRC) may be present to ensure message integrity.

Here's a more detailed breakdown of MQTT packet structure for some common packet types:

- **CONNECT**: This is the initial packet sent by the client to establish a connection with the MQTT broker. The variable header contains information like the protocol name and version, the client's connection parameters, and the client ID.

- **PUBLISH**: This packet is used to send a message from a publisher to subscribers. The variable header contains the topic name, message ID (optional), QoS level, and retain flag. The payload carries the actual message.
- **SUBSCRIBE**: Sent by the client to subscribe to one or more topics. The variable header contains a message ID, and the payload contains a list of topic filter and QoS level pairs that the client wants to subscribe to.
- **SUBACK**: Sent by the broker to acknowledge the SUBSCRIBE request. The variable header contains the message ID, and the payload contains a list of granted QoS levels for each requested subscription.
- **PUBACK, PUBREC, PUBREL, PUBCOMP**: These packets are used in the QoS level 1 and 2 message delivery flows, providing acknowledgment and message flow control.
- **UNSUBSCRIBE**: Sent by the client to unsubscribe from one or more topics. The variable header contains a message ID, and the payload contains a list of topic filters to unsubscribe from.
- **UNSUBACK**: Sent by the broker to acknowledge the UNSUBSCRIBE request. The variable header contains the message ID.
- **PINGREQ**: Sent by the client to the broker as a keep-alive mechanism to maintain the connection.
- **PINGRESP**: Sent by the broker in response to a PINGREQ to acknowledge the client's presence.

These are the basic MQTT packet types and their structures. The specific format and contents of variable headers and payloads can vary depending on the packet type and options used in the MQTT protocol version being used (e.g., MQTT 3.1.1 or MQTT 5.0).

UNIT- V: EDGE COMPUTING WITH RASBERRYPi

Industrial IoT

IoT leverages connected devices, sensors, data analytics, and real-time communication to transform traditional industries and improve operational efficiency, productivity, and decision-making. Here are key aspects and components of Industrial IoT:

1.Connected Devices and Sensors:

- IIoT relies on a network of sensors, actuators, and connected devices that collect data from various points in an industrial environment.
- These devices can include temperature sensors, pressure sensors, RFID tags, cameras, and more.

2. Data Collection and Monitoring:

•IIoT systems gather data in real-time from industrial assets and processes. This data includes information on equipment status, performance metrics, environmental conditions, and more.

•Industrial IoT platforms use protocols like MQTT and CoAP to efficiently transmit data to centralized systems or the cloud.

3. Cloud Computing and Edge Computing:

1. IIoT systems often employ cloud computing for data storage, analysis, and processing. Cloud platforms provide scalability and accessibility.
2. Edge computing is used to perform real-time processing and decision-making closer to the data source, reducing latency and enabling quick responses in critical applications

4. Data Analytics and Machine Learning:

1. Advanced analytics and machine learning algorithms are applied to IIoT data to gain insights, detect anomalies, predict failures, and optimize processes.
2. Predictive maintenance is a common use case, where machine learning models help predict when industrial equipment is likely to fail, allowing for proactive maintenance.

5. Security and Cybersecurity:

1. Security is a paramount concern in IIoT, as industrial systems are critical and vulnerable to cyberattacks.
2. Security measures include data encryption, access controls, intrusion detection systems, and secure communication protocols.

6. **Interoperability and Standards**:

   1. Interoperability is crucial in IIoT to ensure that devices and systems from different manufacturers can communicate and work together seamlessly.
   2. Standards like OPC UA (Unified Architecture) and MQTT help facilitate interoperability.

7. **Remote Monitoring and Control**:

   IIoT enables remote monitoring and control of industrial processes and equipment, allowing operators to make adjustments, diagnose issues, and optimize operations from anywhere.

8. **Energy Efficiency**:

   IIoT can help improve energy efficiency in industrial settings by monitoring energy consumption, optimizing machine operation, and identifying areas for improvement.

9. **Supply Chain and Inventory Management**:

   IIoT can be used for tracking inventory, managing the supply chain, and improving logistics processes in manufacturing and distribution.

10. **Predictive Analytics**:

    Predictive analytics in IIoT allows businesses to anticipate demand, optimize production schedules, and reduce waste by making data-driven decisions.

11. **Quality Control**:

1. IIoT sensors and cameras can be used for real-time quality control in manufacturing processes, ensuring that products meet quality standards.

12. **Safety and Compliance**:

1. IIoT technologies can help improve workplace safety by monitoring conditions and providing alerts in hazardous environments.
2. Compliance with industry regulations and standards is also facilitated through IIoT data collection and reporting.

Industrial IoT has the potential to transform a wide range of industries, including manufacturing, energy, agriculture, healthcare, and transportation, by making processes more efficient, cost-effective, and data-driven.

Commercial IoT (Internet of Things) refers to the deployment of IoT technologies and solutions for commercial and business purposes.
It involves the use of connected devices, sensors, data analytics, and automation to improve operational efficiency, reduce costs, enhance customer experiences, and drive revenue growth in various commercial sectors.
Here are some key aspects and applications of commercial IoT:

Smart Buildings and Facilities Management:

- Commercial IoT is used to create smart buildings and facilities.
- IoT sensors monitor and control various aspects of building operations, including lighting, security, and energy consumption.
- Smart building systems can optimize energy usage, improve occupant comfort, and reduce maintenance costs.

2. Retail and Customer Engagement:

1. In the retail sector, IoT enables retailers to gather data on customer behavior and preferences. This data can be used for personalized marketing, inventory management, and optimizing store layouts.
2. Beacons and RFID technology are often used for in-store customer engagement and tracking.

3. Supply Chain Management:

1. Commercial IoT is employed to monitor and track goods throughout the supply chain. RFID tags, GPS trackers, and sensors provide real-time visibility into the movement and condition of products.
2. This helps improve inventory management, reduce loss, and enhance the accuracy of delivery and logistics.

4. Fleet Management:

- IoT solutions are used in commercial fleets to monitor vehicle location, driver behavior, fuel consumption, and maintenance needs.
- Fleet management systems can optimize routes, reduce fuel costs, and enhance driver safety.

5. **Manufacturing and Industrial Automation**:
    1. IoT technologies are integral to Industry 4.0, enabling smart factories and industrial automation.
    2. Sensors and connected devices collect data from machines and processes, facilitating predictive maintenance, quality control, and production optimization.
6. **Healthcare and Telemedicine**:
    1. In the healthcare sector, commercial IoT is used for remote patient monitoring, wearable health devices, and telemedicine.
    2. IoT devices help healthcare providers collect patient data, make diagnoses, and improve patient outcomes.
7. **Agriculture**:
    1. Precision agriculture relies on IoT sensors and automation to monitor soil conditions, weather, crop health, and equipment performance.
    2. IoT in agriculture improves crop yields, reduces resource waste, and promotes sustainable farming practices.

8. **Asset Tracking and Management**:

   1. Many businesses use IoT to track and manage valuable assets, such as equipment, vehicles, and tools.
   2. Asset tracking solutions help prevent theft, loss, and improve asset utilization.

9. **Environmental Monitoring**:

   1. Commercial IoT is used for environmental monitoring, including air quality, water quality, and pollution control.
   2. Real-time data from sensors helps businesses comply with regulations and mitigate environmental risks.

10. **Security and Access Control**:

    1. IoT-based security systems are widely used for access control, surveillance, and alarm systems in commercial properties.
    2. These systems enhance security and provide remote monitoring capabilities.

11. **Energy Management**:
    IoT-based energy management solutions help commercial buildings and industries optimize energy consumption, reduce costs, and lower carbon footprints.
12. **Financial Services**:
    In the financial sector, IoT is used to monitor and track assets, automate processes, and enhance customer services through smart banking and payment solutions.

Commercial IoT solutions are diverse and can be tailored to meet the specific needs of different industries and businesses

They often involve the integration of various IoT devices, cloud computing, data analytics, and machine learning to drive efficiency and innovation in commercial operations.

Industrial and Commercial Edge Computing

When it comes to edge computing in industrial and commercial settings, Raspberry Pi can be a valuable tool with its features tailored to meet the requirements of these environments.

1. **Real-Time Data Processing:** Raspberry Pi's processing power and low latency make it suitable for real-time data processing in industrial automation and control systems.
   It can handle data from sensors, analyze it locally, and take appropriate actions in real-time.

2. **Industrial Protocols Support:** Raspberry Pi can be equipped with additional hardware or software to support industrial communication protocols such as Modbus, Profibus, or OPC UA.

This allows seamless integration with industrial equipment and machines.

3. IoT Device Integration: Raspberry Pi's GPIO pins and support for various communication protocols enable easy integration with a wide array of IoT devices commonly used in industrial and commercial applications.
It can serve as a gateway for managing and monitoring IoT devices at the edge.

4. Edge AI and Machine Learning: Raspberry Pi can run lightweight machine learning models and AI algorithms at the edge.
This is valuable in applications like predictive maintenance, quality control, and anomaly detection, enhancing operational efficiency in industries.

5. Remote Monitoring and Control: Raspberry Pi, combined with suitable software and connectivity options, enables remote monitoring and control of industrial processes and equipment.
This is crucial for efficient operations and timely decision-making.

6. Secure Data Processing: Security features and capabilities of Raspberry Pi can be enhanced using encryption, secure boot, and access control mechanisms to ensure data security in sensitive industrial and commercial environments.

7. Custom Industrial Applications: Developers can create custom applications tailored to specific industrial and commercial use cases, leveraging the flexibility and customization options of Raspberry Pi.

8. Edge Data Storage and Retrieval: Raspberry Pi can store and manage data at the edge, reducing the need for constant data transfers to centralized servers.

This is particularly useful for applications with intermittent or limited connectivity.

9. Scalability: Raspberry Pi solutions can be easily scaled by deploying multiple devices across a facility or network, providing a scalable edge computing infrastructure.

10.Integration with Existing Systems: Raspberry Pi can integrate with existing legacy systems and equipment in industrial and commercial environments, extending the lifespan and functionality of these systems.

11. Diagnostics and Monitoring: Raspberry Pi can be utilized for monitoring the health and performance of equipment, predicting failures, and scheduling maintenance, thus reducing downtime and increasing efficiency.

Raspberry Pi's features make it a versatile and cost-effective solution for implementing edge computing in industrial and commercial domains, enhancing operational efficiency, data processing capabilities, and enabling innovative applications in these sectors.

Edge Computing & Solutions

Edge computing involves processing and analyzing data closer to its source, typically on devices located at the "edge" of a network, rather than sending all data to centralized data centers.
This approach offers benefits like reduced latency, improved data privacy, bandwidth savings, and the ability to operate in low-connectivity environments.

Here are some key concepts and solutions related to edge computing:

1. **Edge Devices:** Edge devices are the physical devices (e.g., sensors, IoT devices, industrial controllers) responsible for collecting data at the source. These devices have computing capabilities to process data locally.
2. **Edge Computing Infrastructure:** This includes the hardware and software that facilitate data processing and analysis at the edge. It may involve edge servers, gateways, or specialized edge computing hardware.

3. Edge Analytics: Analytical processes performed at the edge to extract meaningful insights from data in real-time.
This can involve data filtering, aggregation, machine learning inference, and other analytics tasks.

4. Edge AI (Artificial Intelligence): Running AI algorithms and machine learning models at the edge devices for immediate decision-making and reduced reliance on centralized cloud AI.
This is crucial for applications like image recognition, predictive maintenance, and natural language processing.

5. **Edge Data Storage:** Local storage of data at the edge to support immediate access and reduce the need for constant communication with central data centers. It's important for applications that require quick data retrieval.

6.Edge-to-Cloud Integration: Efficiently managing data flow and processing between edge devices and central cloud infrastructure. Some data may be processed at the edge, while relevant information is sent to the cloud for further analysis or long-term storage.

7.Edge Security: Implementing security measures to protect data at the edge, which can include encryption, access controls, secure boot processes, and other cybersecurity practices to ensure data integrity and confidentiality.

8.Edge Networking: Networking solutions and protocols optimized for edge environments, ensuring reliable communication between edge devices and centralized systems, especially in scenarios with intermittent connectivity.

9.Edge Application Development: Creating software applications specifically designed to run on edge devices, utilizing the processing power and capabilities available at the edge for improved performance and efficiency.

10.Fog Computing: A related concept to edge computing, fog computing extends the capabilities of edge devices by providing additional computing resources and services in the local network.

11.Use Cases: Edge computing finds applications in various domains including industrial automation, healthcare (e.g., remote patient monitoring), retail (e.g., smart shelves, inventory management), transportation (e.g., autonomous vehicles), smart homes, agriculture, and more.

12.Edge Management and Orchestration: Tools and platforms that assist in **managing** and **orchestrating edge devices**, applications, and resources, ensuring optimal performance, security, and scalability.

Effective implementation of edge computing solutions requires careful consideration of the specific use case, the nature of the data, the processing requirements, and the connectivity available at the edge.

Integration of edge computing with centralized cloud services creates a powerful hybrid computing environment that can address a wide range of business and technological needs.