blockchained
{'india'}

presents

60 Days of Code with

Polkadot

Instructions:

1. Ensure that the task is completed on the very same day it is assigned. If you face any problem, do share it with us in the group at the same moment, so that the doubts are cleared at the earliest.

2. Every week we will complete one module. Each module is further divided into 5 days. We have kept the weekends free so you can catch up on the work over the weekend and attend the doubts session/workshops which will be organised.

3. If you are not able to complete the task, ensure that they are covered in the very week so that the process is not delayed much.

4. We will have live workshops every week. The schedule of the same will be shared at the start of the week and the same is added to your weekly curriculum. Do ensure you are attending these workshops as they are live sessions by industry experts. It is going to help you only in your journey. If by any reason you are not able to attend, inform the team beforehand.

5. Please submit your assignment as these will be necessary to participate in the hiring track.

6. Please update your daily progress by posting on twitter and sending a message on the group When posting on your twitter accounts please tag us **@blockchainedind** and **@Polkadot**, along with the use of following hashtags:
#60DaysofCode #BuildInPublic #Polkadotdevbootcamp

If you have any questions or doubts, please reach out to us on the group or email us at hriday@blocumen.com and manav@blocumen.com.

Content

Module 3: Introduction to Polkadot & Substrate
(June 18 to June 24)

Module 3: Introduction to Polkadot & Substrate

Day 1

Introduction to Polkadot

Polkadot is a multi-chain platform that enables different blockchains to interoperate and communicate with each other.Polkadot was the first fully sharded blockchain. It was developed by the Web3 Foundation, founded by Dr. Gavin Wood, one of the co-founders of Ethereum. Polkadot aims to address the challenges of scalability, security, and interoperability that exist in many existing blockchain networks

Why Polkadot?

In the early 2000s, the internet started with basic, static web pages known as Web 1.0. As social media and online businesses emerged, the internet evolved into Web 2.0, featuring interactive pages and user-generated content. However, Web 2.0 raised concerns about data control and privacy. This is where Web 3.0, or Web3, comes in.

Web3 aims to decentralize the internet, shifting from centralized platforms to trust-free protocols. It envisions a web where users have control over their data and identity, eliminating the need for intermediaries. The goal is to build a decentralized and trustless infrastructure for a more secure and user-centric online experience.

Differences between Bitcoin,Ethereum and Polkadot

Key Points	Bitcoin	Ethereum	Polkadot
Purpose and Functionality	Bitcoin was the first decentralized cryptocurrency and is primarily designed as a digital currency for peer-to-peer transactions and store of value. Its main focus is on secure and censorship-resistant payments	Ethereum is a decentralized platform that enables the creation of smart contracts and decentralized applications (dApps). It allows developers to build and deploy their own applications on the Ethereum blockchain, utilizing its native cryptocurrency called Ether (ETH).	Polkadot is a multi-chain platform that facilitates interoperability between different blockchains. It aims to overcome the scalability and connectivity limitations of single-chain networks by enabling parallel processing and interchain communication. Polkadot allows for the creation of specialized blockchains called parachains that can run in parallel and connect to external networks.
Blockchain Architecture	Bitcoin operates on a single-chain blockchain architecture. It utilizes a proof-of-work (PoW) consensus algorithm, where miners compete to solve complex mathematical problems to validate transactions and add blocks to the chain.	Ethereum also operates on a single-chain blockchain architecture, but it introduced the concept of smart contracts. Ethereum initially used a PoW consensus algorithm but is transitioning to a proof-of-stake (PoS) consensus mechanism called Ethereum 2.0, which relies on validators instead of miners	Polkadot utilizes a multi-chain architecture, where multiple parallel blockchains called parachains run in parallel. The network's central chain, known as the Relay Chain, provides security and coordination for the parachains. Polkadot employs a nominated proof-of-stake (NPoS) consensus algorithm where stakeholders nominate validators to secure the network.
Interoperability	Bitcoin is primarily focused on its own	Ethereum has limited	Polkadot is specifically designed for

	blockchain and does not natively support interoperability with other blockchains. However, there are solutions like atomic swaps and wrapped tokens that enable limited cross-chain functionality.	interoperability through solutions like bridges and token standards (such as ERC-20 and ERC-721), allowing for the exchange of assets between Ethereum and other blockchains. Ethereum's upcoming upgrade, Ethereum 2.0, aims to enhance scalability and interoperability.	interoperability. It enables parachains to communicate with each other and with external networks through its interchain messaging system. This allows for seamless asset transfers and data sharing between different blockchains.
Governance and Upgradability	Bitcoin has a decentralized governance model where changes require community consensus. Upgrades to the Bitcoin network typically require a majority of miners and users to adopt the proposed changes through a process known as a "soft fork" or "hard fork."	Ethereum is transitioning to a more decentralized governance model through Ethereum Improvement Proposals (EIPs) and community voting. Upgrades and improvements are proposed, discussed, and implemented through a governance process.	Polkadot implements an on-chain governance mechanism where token holders can participate in decision-making processes. Changes to the network, including protocol upgrades, parameter changes, and the addition or removal of parachains, are decided through a voting mechanism.

Key Concepts

1. Multi-chain Network: Polkadot is designed as a heterogeneous multi-chain network, consisting of multiple parallel blockchains called "parachains" that run in parallel.Each parachain is said to be a sharded chain. These parachains can have different characteristics, such as their own consensus mechanisms, token economies, and governance models.

2. Shared Security: Polkadot employs a shared security model where all parachains in the network benefit from the security provided by the Polkadot Relay Chain. The Relay Chain acts as the central chain of Polkadot and is responsible for validating transactions, coordinating consensus, and ensuring the security of the entire network.

3. Cross-Chain Interoperability: Polkadot enables cross-chain communication and interoperability through its innovative interchain messaging system. Parachains can send messages to each other or to external networks like Ethereum or Bitcoin, allowing for the transfer of assets and information between different blockchains.

4. Governance and Upgradability: Polkadot has a built-in governance mechanism that allows stakeholders to participate in the decision-making process for protocol upgrades, parameter changes, and the addition or removal of parachains. This on-chain governance model ensures that the network can evolve and adapt over time in a decentralized manner.

5. Scalability and Performance: By utilizing a sharded architecture, Polkadot can achieve high scalability by processing multiple transactions in parallel across different parachains. This helps to alleviate the issues of network congestion and high transaction fees commonly associated with single-chain blockchain platforms.

6. Polkadot's Native Token: The native token of the Polkadot network is called DOT. DOT holders have various roles and responsibilities within the network, including participating in the governance process, staking DOT for block production, and bonding DOT to support the security of parachains.

Benefits of Polkadot

1. Interoperability: Polkadot's interoperability framework allows different blockchains to seamlessly communicate and share data, opening up new possibilities for decentralized applications (dApps) and cross-chain transactions.

2. Scalability: The sharded architecture of Polkadot enables horizontal scaling, where the network can process multiple transactions in parallel across different parachains, significantly increasing the network's capacity.

3. Security: By sharing security among all parachains through the Relay Chain, Polkadot provides a robust security model that helps protect against attacks and vulnerabilities that may exist in individual blockchains.

4. Governance: Polkadot's on-chain governance system empowers token holders to have a say in the decision-making process, fostering a more decentralized and community-driven ecosystem.

5. Future-proofing: Polkadot's modular design and upgradability make it well-positioned to adapt to new technological advancements and accommodate future innovations in the blockchain space.

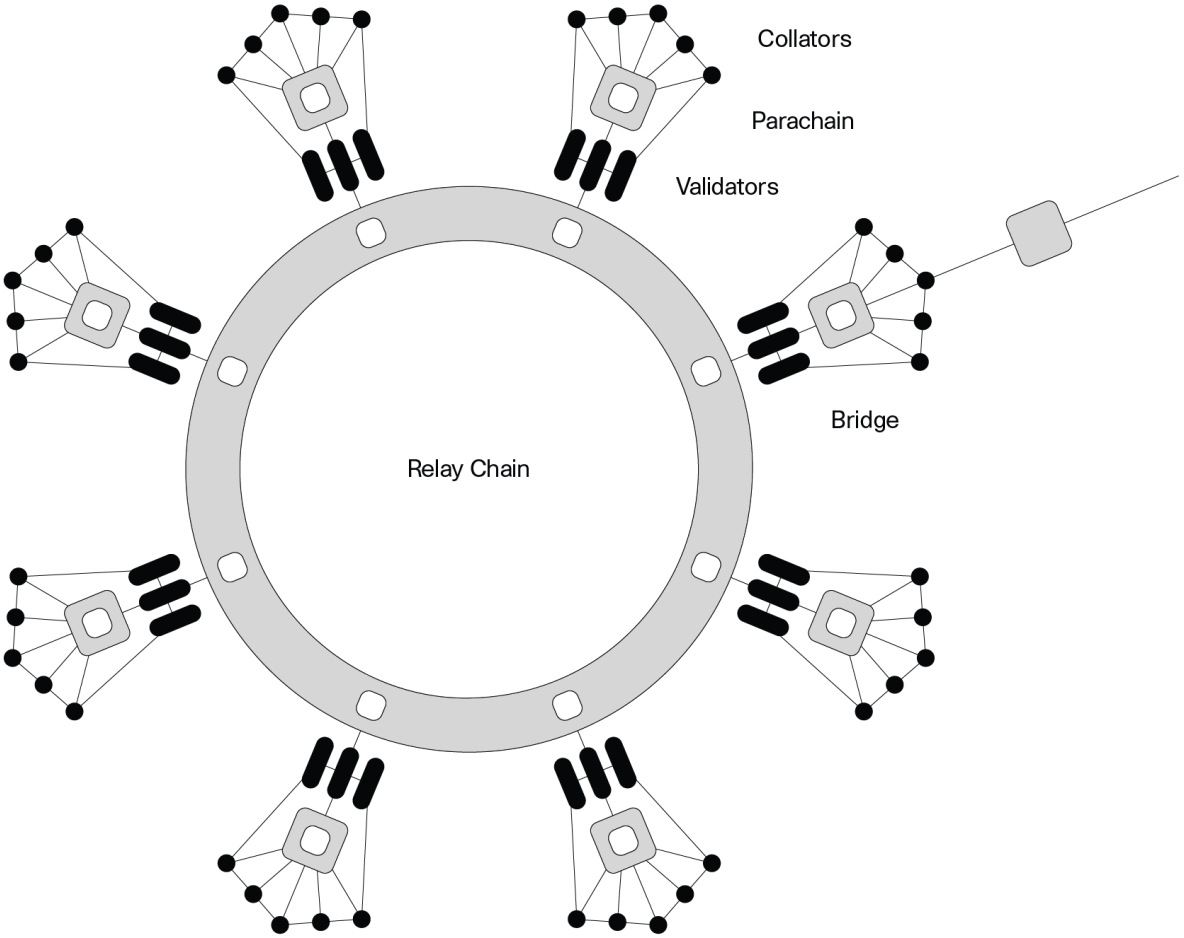
In summary, Polkadot is a multi-chain platform that addresses the challenges of scalability, security, and interoperability in blockchain networks. Its sharded architecture, shared security model, and governance mechanism make it a promising framework for building scalable and interconnected decentralized applications.

Day 2

Architecture of Polkadot

Videos:

What is Polkadot? Parachains, Parathreads, and the Polkadot Ecosystem Explained in…



RelayChains:

In Polkadot's architecture, the Relay Chain is the heart of the network and ensures the overall security and coordination between parachains. It achieves consensus through the nominated Proof-of-Stake (NPoS) mechanism, where DOT token holders

select a set of validators who are responsible for block production and finalizing the chain.

Parachains:

Parachains are independent chains that run in parallel to the Relay Chain. They can have their own governance models, consensus mechanisms, and rules tailored to specific use cases or applications. Each parachain has its own set of validators that secure the chain and participate in block production. The validators are selected through a transparent and decentralized mechanism to ensure the security and integrity of the network.

Collators:

Collators, which are full nodes specific to each parachain, collect transactions and other data for inclusion in the parachain's blocks. Collators are responsible for constructing parachain blocks and submitting them to the validators on the Relay Chain for validation and inclusion in the final chain. Once the parachain block is validated and finalized by the validators, it becomes a part of the Relay Chain's history.

Shared Security:

Polkadot uses a technique called "shared security" to provide security to parachains. Since the security of the network is derived from the Relay Chain, each parachain benefits from the overall security of the network. This shared security model allows parachains to focus on their specific functionalities and use cases without compromising on security.

Interoperability

XCM:

XCM, short for cross-consensus message, is a format and not a protocol. The format does not assume anything about the receiver or senders consensus mechanism, it only cares about the format in which the messages must be structured in. The XCM format is how parachains will be able to communicate with one another. Different from XCMP, which is short for cross-chain messaging protocol, XCM is what gets delivered, and XCMP is the delivery mechanism.

Bridge:

The bridge parachains in Polkadot enable interoperability between Polkadot and other external blockchains or networks. These bridges facilitate the transfer of assets and data between different chains, allowing Polkadot to connect with other ecosystems and leverage their functionalities. For example, bridges can enable the transfer of tokens between Polkadot and Ethereum, enabling cross-chain compatibility and liquidity.

Governance:

Polkadot also has a built-in governance mechanism that allows token holders to participate in the decision-making process of the network. Through on-chain governance, stakeholders can propose and vote on important network upgrades, parameter changes, and the addition or removal of parachains.

Overall, Polkadot's architecture enables scalability, interoperability, and flexibility by utilizing parallel processing, shared security, and a modular approach to parachains. It provides a framework for building and connecting specialized blockchains, creating a robust and interconnected ecosystem of decentralized applications.

Actors in Polkadot Ecosystem:

Validators:

Validators are responsible for producing blocks on the Relay Chain if they are elected to the validator set. They also accept proofs of valid state transitions from collators. In return for their work, validators receive staking rewards.

To ensure the reliability of the network, validators must store enough parachain blocks locally. This allows other participants to retrieve the blocks when needed to verify the validity of parachain statements. Upholding these responsibilities involves following the Availability & Validity (AnV) protocol, which consists of multiple steps.

Nominators:

Nominators bond their stake to specific validators to help them become part of the active validator set. By doing so, nominators contribute to the production of blocks on the chain. In return for their support, nominators typically receive a portion of the staking rewards from the validator they nominated.

Collators:

Collators are full nodes that operate on both a parachain and the Relay Chain. Their main tasks include collecting transactions for the parachain and producing state transition proofs for the validators on the Relay Chain. Collators can also send and receive messages from other parachains using XCMP.

While collators are responsible for creating parachain blocks, relay chain validators focus on verifying the validity of these blocks, including their availability at later stages. This separation of tasks ensures efficient processing and validation within the network.

Video:

Whiteboard Series with NEAR | Ep: 14 Alistair Stewart from Polkadot |

Day 3

Consensus Mechanism in Polkadot

Consensus in Polkadot

Consensus is a crucial component in blockchain networks as it enables nodes to agree on a shared state. Without consensus, nodes in a decentralized network would not be able to synchronize with each other. Consensus ensures that all nodes in the network have a common understanding of the blockchain's state. It allows nodes to communicate, reach agreement, and build new blocks.

Proof of work and Proof of Stake

Proof of Work (PoW) and Proof of Stake (PoS) are two commonly used consensus mechanisms in blockchain systems. However, these terms are often used inaccurately to refer to the broader consensus mechanisms. PoW is a method used to determine the block author and is part of the Nakamoto consensus, which also includes a chain selection algorithm (such as the longest chain rule in Bitcoin). PoS, on the other hand, is a set of rules for selecting validators but does not specify a chain selection rule or how a chain achieves finality. PoS algorithms are typically combined with Byzantine fault-tolerant algorithms for coming to an agreement between nodes.

Although PoW is simple and effective in achieving decentralized consensus, it consumes a significant amount of energy, lacks economic or provable finality, and is not resistant to cartels.

Nominated Proof of Stake

Nominated Proof of Stake (NPoS) is a mechanism used by Polkadot to select its validators. In traditional PoS systems, block production participation is based on token holdings, which can lead to centralization where only a limited number of wealthy validators have full participation rights. To address this, Polkadot allows token holders to vote for validators using their stake. Validators play a role in producing new blocks, validating parachain blocks, and ensuring finality. Nominators can choose to back certain validators with their stake, approving candidates they trust.

Probabilistic finality

Blockchain networks that rely solely on PoW achieve probabilistic finality and eventual consensus. Probabilistic finality means that if a few blocks are built on a given block, there is a probability that it is final. Eventual consensus refers to the point in the future when all nodes agree on the truthfulness of a set of data. However, finality gadgets like GRANDPA or Ethereum's Casper FFG aim to provide stronger and quicker guarantees on block finality. These gadgets ensure that once a certain process of Byzantine agreements takes place, the consensus on blocks becomes provable and irreversible.

Hybrid Consensus

Polkadot uses a hybrid consensus approach by combining two protocols: GRANDPA and BABE. GRANDPA is responsible for achieving finality, while BABE handles block production. This hybrid consensus allows for the benefits of both probabilistic finality and provable finality. It ensures rapid block production while finalizing blocks through a separate process to avoid transaction processing delays or stalling.

Block Production

BABE is the block production mechanism in Polkadot that assigns block production slots to validators based on their stake and using Polkadot randomness. Validators participate in a lottery for each slot, determining whether they are candidates for block production. Slots are discrete units of time, approximately 6 seconds long. Multiple validators can be candidates for the same slot, leading to a race where the block produced and disseminated first wins. If no validator qualifies for block production in a slot, a secondary, round-robin style validator selection algorithm runs to ensure a block is produced.

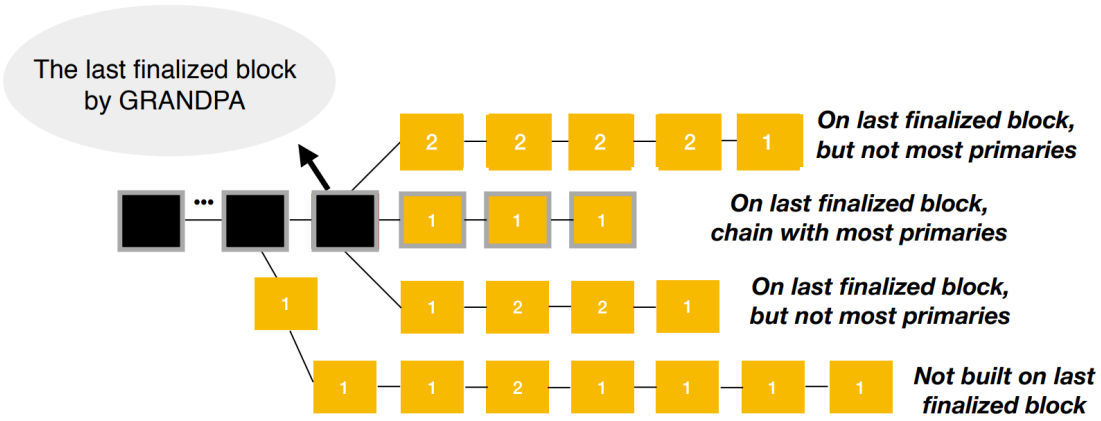
Finality

GRANDPA is the finality gadget implemented in the Polkadot Relay Chain. It achieves finality through consecutive rounds of voting by validator nodes. Once more than 2/3 of validators attest to a chain containing a certain block, all blocks leading up to that block are finalized. GRANDPA provides quicker finalization by reaching agreements on chains rather than individual blocks.

Bridging

BEEFY is a secondary protocol in the Polkadot consensus mechanism that works alongside GRANDPA to facilitate efficient bridging between the Polkadot network (relay chain) and external blockchains like Ethereum. BEEFY enables participants in the remote network to verify finality proofs generated by the validators of the Polkadot relay chain. This allows clients in the Ethereum network, for example, to confirm the specific state of the Polkadot network.

Storing all the information required to verify the state of the remote chain, such as block headers, can be resource-intensive. BEEFY addresses this challenge by storing the information in a space-efficient manner. Clients can request additional information through the BEEFY protocol as needed.
The combination of BABE and GRANDPA in Polkadot determines the fork choice of the network. BABE always builds on the chain that has been finalized by GRANDPA. In the event of forks occurring after the finalized head, BABE provides probabilistic finality by selecting the chain with the most primary blocks.



Longest chain with most primaries on last finalized GRANDPA block

In the fork choice diagram provided, the black blocks represent finalized blocks, while the yellow blocks are non-finalized. The blocks marked with "1" are primary blocks, and those marked with "2" are secondary blocks. In this example, even though the topmost chain is the longest chain from the latest finalized block, it does not qualify because it has fewer primary blocks compared to the chain below it at the time of evaluation.

Comparison of Various Consensus Mechanisms

When comparing Polkadot's consensus mechanisms to other existing ones, some notable differences arise:

1. Nakamoto Consensus: Nakamoto consensus, used in Bitcoin, relies on the longest chain rule and proof of work for Sybil resistance and leader election. It offers probabilistic finality, where the safety of a block depends on the number of confirmations it has. However, it does not guarantee the permanence of a chain, as an actor with substantial computational resources could create a competing chain.

2. PBFT / Tendermint: Polkadot's GRANDPA consensus can be compared to the Practical Byzantine Fault Tolerance (PBFT) consensus used in Tendermint. However, there are differences in the ability for different voters to cast simultaneous votes for blocks at different heights. GRANDPA's fork-choice rule relies on finalized blocks to influence the underlying block production mechanism.

3. Casper FFG: Another comparison can be drawn between Polkadot's GRANDPA and Ethereum's Casper FFG. The main differences lie in the simultaneous voting capability of different voters at different block heights in GRANDPA. Additionally, GRANDPA's fork-choice rule is impacted by finalized blocks.

By incorporating various consensus mechanisms and protocols like BABE, GRANDPA, and BEEFY, Polkadot aims to provide a scalable, interoperable, and secure platform for multiple blockchains to coexist and interact with each other.

Workshop Details

Fourth Workshop by Radha Krishnan from Web 3 Foundation at 7:00 pm IST on 21st June. Joining Link:

https://us02web.zoom.us/j/85958300827?pwd=dTVreXJLQXFzdVNrTUp0aVpZUzdJUT09

Day 4

Introduction to Substrate

Substrate is an open-source framework developed by Parity Technologies that allows developers to build custom blockchain networks. It is primarily written in the Rust programming language and provides a flexible foundation for creating decentralized applications (dApps) and blockchains, including Polkadot.

Architecture of Substrate:

Substrate's architecture is modular and follows a layered design. It consists of several components that work together to provide a robust blockchain development framework:

1. Core:

The core module contains the fundamental building blocks for creating blockchains, including the runtime, consensus, networking, and storage.

2. Runtime:

The runtime defines the logic and behavior of the blockchain. It is the customizable part of Substrate that developers can modify to create their own blockchain's specific functionality. The runtime is implemented using the Rust programming language and is compiled to WebAssembly (Wasm) for execution on the Substrate platform.

3. Pallets:

Pallets are reusable modules that developers can use to add predefined functionality to their blockchain. Pallets provide a way to extend the runtime with features like balances, governance, staking, oracles, and more. They can be developed and integrated by the blockchain developers or reused from existing libraries.

4. Substrate Node:

The Substrate Node is the executable that runs the blockchain network. It provides the infrastructure to connect peers, validate blocks, execute runtime logic, and handle other network-related tasks.

Rust Libraries in Substrate:

Substrate leverages various Rust libraries to enhance its functionality and provide a robust development environment. Some important Rust libraries used in Substrate include:

1. Libp2p

A modular networking library that enables peer-to-peer communication and connectivity in Substrate networks.

2. Tokio:

A runtime for asynchronous programming that provides concurrent and non-blocking execution of tasks, enhancing the performance of Substrate.

3. Serde: A serialization and deserialization library that allows easy conversion of data structures to and from various formats like JSON, binary, or SCALE (Substrate Customizable Arbitrary-Length Encoding).

4. RocksDB: A high-performance embedded database used for efficient storage of blockchain state data.

What can be built using Substrate:

Substrate provides developers with a powerful framework to build various types of blockchain-based applications, including:

1. Custom Blockchains: Substrate allows developers to create their own standalone blockchain networks with customized runtime logic and built-in features.

2. Parachains: Parachains are independent blockchains that connect to the Polkadot network. Substrate enables the development of parachains with specific functionalities that can interact and share security with the Polkadot relay chain.

3. dApps and Smart Contracts: Substrate allows the development of decentralized applications and smart contracts with its customizable runtime. Developers can create their own application-specific logic and integrate it with the blockchain network.

Runtime Development in Substrate:

The runtime is the most important part of a Substrate-based blockchain as it defines the state transition logic and runtime modules. Runtime development in Substrate involves the following steps:

1. Define the Runtime: Developers define the runtime by creating and configuring runtime modules. These modules define the state and behavior of the blockchain, such as account balances, staking, governance, oracles, etc.

2. Implement Pallets: Developers can implement custom pallets by extending or reusing existing pallets to add specific functionality to their blockchain. Pallets encapsulate related runtime logic and provide reusable components.

3. Compile to Wasm: The runtime logic, written in Rust, is compiled to WebAssembly (Wasm). Substrate uses the wasm32-unknown-unknown target to generate the Wasm bytecode.

4. Integration and Testing: The compiled runtime is integrated into the Substrate framework, and developers can test the runtime using the Substrate Node environment or the Substrate

Testkit.

Transaction and Block Basics in Substrate:

In Substrate, transactions represent operations or actions performed by users or dApps. These transactions are bundled into blocks, which are then added to the blockchain. Here's an overview of their lifecycle and related concepts:

1. Transaction Lifecycle:
- Creation: Users or dApps create transactions to perform actions on the blockchain.
- **Verification:** Transactions are validated for correctness, signature verification, and other custom checks defined in the runtime.
- Inclusion: Valid transactions are included in a block by block producers (validators or miners).
- **Execution:** When a block is finalized, its transactions are executed, updating the blockchain state based on the transaction logic defined in the runtime.

2. Block Lifecycle:

- **Block Proposal:** Validators or miners propose new blocks by including a set of valid transactions.
- **Block Validation:** Proposed blocks are validated by other validators to ensure correctness and adherence to consensus rules.
- **Finalization:** Consensus mechanisms (e.g., GRANDPA in Polkadot) determine the finality of a block, marking it as part of the canonical chain.
- **State Transition:** Once a block is finalized, its transactions are executed, and the blockchain state is updated based on the transaction logic.

State Transitions and Storage:

Substrate's state transition function, also known as the "execute_block" function, is responsible for executing the transactions in a block and updating the blockchain state. It follows these steps:

1. Validation: Transactions within a block are validated individually and collectively to ensure correctness, signature verification, and adherence to runtime-specific rules.

2. Execution: Valid transactions are executed in order, modifying the blockchain state according to the logic defined in the runtime modules and pallets.

3. Storage Changes: The execution of transactions may result in changes to the storage state of the blockchain. Substrate manages storage through a key-value store, allowing efficient and secure storage and retrieval of data.

Accounts, Access, and Keys in Substrate:

Substrate provides a rich account model for users and dApps interacting with the blockchain. Some key aspects include:

1. Accounts: Accounts represent entities on the blockchain and store balances and other data associated with them.

2. Keys and Signatures: Accounts are associated with cryptographic key pairs (public and private keys). Transactions are signed using the account's private key to ensure authenticity and integrity.

3. Access Control: Substrate provides a flexible access control mechanism, allowing developers to define custom permission schemes and control who can execute specific runtime functions or access certain resources.

Developers can leverage these account-related features to implement secure and permissioned interactions within their Substrate-based blockchains.

In summary, Substrate is a powerful and flexible framework for building custom blockchains, parachains, dApps, and smart contracts. With its modular architecture, Rust libraries, and runtime development capabilities, developers can create scalable and interoperable blockchain solutions tailored to their specific use cases while leveraging the infrastructure and security provided by Polkadot.

Rust for Substrate:

Rust is the primary programming language used for developing Substrate. Rust is chosen for its focus on memory safety, performance, and concurrency. It provides a strong type system and advanced features like ownership and borrowing, which help prevent common programming errors. Rust's characteristics make it well-suited for building secure and efficient blockchain applications.

Offchain Computation:

Offchain computation refers to performing computations outside the blockchain's consensus process. In Substrate, offchain workers (OCWs) enable developers to execute complex computations off the blockchain while utilizing the blockchain's data and security. OCWs can access external APIs, perform heavy computations, and provide results to the blockchain as lightweight proofs. This approach improves scalability and reduces the burden on the consensus mechanism.

Light Clients in Substrate:

Light clients allow users or nodes to interact with a blockchain without fully downloading and validating the entire blockchain history. Substrate supports light clients, which can verify the authenticity and integrity of transactions and blocks using a reduced set of data called "headers." Light clients are useful for applications that require low resource consumption, such as mobile wallets or lightweight nodes.

Cryptography:

Cryptography plays a crucial role in Substrate for securing the blockchain network and protecting sensitive data. Substrate employs various cryptographic algorithms and protocols, including:

- **Public-Key Cryptography:** Substrate uses cryptographic key pairs (public and private keys) for account identities, digital signatures, and encryption. Popular algorithms like Elliptic Curve Cryptography (ECC) are commonly used for key generation and signature verification.

- **Hash Functions:** Hash functions like SHA-256 or BLAKE2 are utilized for data integrity checks, storing passwords securely, and constructing Merkle trees for efficient storage and verification.

- **Encryption:** Substrate can utilize symmetric or asymmetric encryption algorithms for secure communication and data protection.

Consensus:

Consensus algorithms determine how agreement is reached among network participants on the state of the blockchain. Substrate provides various consensus mechanisms that can be selected or customized based on the specific requirements of the blockchain, including:

- **Aura:** Aura is a simple, Byzantine Fault-Tolerant (BFT) consensus algorithm used in Substrate's single-chain networks.

- **BABE (Blind Assignment for Blockchain Extension):** BABE is a probabilistic, leader-based consensus algorithm used in Substrate's Polkadot-compatible networks. It provides finality and enhances the scalability of the network.

- **GRANDPA (GHOST-based Recursive Ancestor Deriving Prefix Agreement):** GRANDPA is a finality gadget used in Substrate's Polkadot-compatible networks. It ensures block finality by recursively selecting the longest chain and enables interoperability among parachains.

Cross-Consensus Messaging:

In Substrate, cross-consensus messaging allows different blockchains, such as parachains in the Polkadot ecosystem, to communicate and exchange information. It enables interoperability and data transfer between chains, facilitating complex decentralized applications. Substrate provides mechanisms like XCMP (Cross-Chain Message Passing) and SPREE (State Provisioning and Receipt Execution) to achieve cross-consensus messaging.

Day 5

Installing Substrate

Linux development environment

To set up your development environment for Substrate on Linux and compile a Substrate node, follow these steps:

1. Install Required Packages:

- Check your operating system's documentation to see if any additional software dependencies are needed for your specific distribution and version.
- On Ubuntu, you can use the following command to install some necessary packages: `sudo apt install build-essential clang curl git make libssl-dev`

2. Install Rust:

- Open a terminal shell and run the following command to download and install the Rust toolchain: `curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh`
- Follow the prompts to proceed with the default installation.
- Update your current shell to include Cargo, the Rust package manager, by running: `source $HOME/.cargo/env`
- Verify your Rust installation by running: `rustc --version`

3. Configure Rust Toolchain:

- Set the default Rust toolchain to the latest stable version with: `rustup default stable`
- Update the toolchain by running: `rustup update`
- Add the nightly release and the nightly WebAssembly (Wasm) targets with: `rustup update nightly` and `rustup target add wasm32-unknown-unknown --toolchain nightly`
- Verify the configuration of your development environment by running: `rustup show` and `rustup +nightly show`

4. Clone the Substrate Node Template:

- Clone the Substrate node template repository by running: `git clone https://github.com/substrate-developer-hub/substrate-node-template`
- Change to the root directory of the cloned repository: `cd substrate-node-template`

- Optionally, create a new branch to save your changes: `git switch -c my-wip-branch`

5. Compile the Node Template:

- Build the Substrate node template by running: `cargo build --release`
- The compilation may take several minutes due to the required packages.
- After a successful build, your local computer is ready for Substrate development.

By following these steps, you will have a properly set up development environment for Substrate on Linux and a compiled Substrate node to work with.

macOS Development Environment Setup

To install Rust and set up a Substrate development environment on macOS, follow these steps:

Before You Begin

Ensure that your computer meets the following requirements:
- macOS version 10.7 Lion or later.
- Processor speed of at least 2GHz (3GHz recommended).
- Memory of at least 8GB RAM (16GB recommended).
- At least 10GB of available storage space.
- Broadband internet connection.

Support for Apple Silicon

Before the build process, install Protobuf by running the following command:
```
brew install protobuf
```

Install Homebrew

Homebrew is recommended for package management on macOS. If you don't have Homebrew installed, follow these steps:

1. Open the Terminal application.
2. Run the command below to download and install Homebrew:
```
```

/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
```
3. Verify the installation by running:
```
brew --version
```
You should see output similar to this:
```
Homebrew 3.3.1
Homebrew/homebrew-core (git revision c6c488fbc0f; last commit 2021-10-30)
Homebrew/homebrew-cask (git revision 66bab33b26; last commit 2021-10-30)
```

## **Installation**

To install openssl and the Rust toolchain on macOS, follow these steps:

1. Open the Terminal application.
2. Update Homebrew with the command:
```
brew update
```
3. Install the openssl package using Homebrew:
```
brew install openssl
```
4. Download and install Rust toolchain (rustup) by running:
```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```
5. Follow the prompts to complete the installation.
6. Update your current shell to include Cargo (Rust package manager):
```
source ~/.cargo/env
```
7. Verify the installation by running:
```
rustc --version
```
8. Set the default Rust toolchain to the latest stable version:
```
rustup default stable

```
rustup update
rustup target add wasm32-unknown-unknown
```

9. Add the nightly release and nightly WebAssembly (wasm) targets:

```
rustup update nightly
rustup target add wasm32-unknown-unknown --toolchain nightly
```

10. Verify the configuration of your development environment:

```
rustup show
rustup +nightly show
```

You should see output similar to this:

```
# rustup show

active toolchain
----------------

stable-x86_64-apple-darwin (default)
rustc 1.61.0 (fe5b13d68 2022-05-18)

# rustup +nightly show

active toolchain
----------------

nightly-x86_64-apple-darwin (overridden by +toolchain on the command line)
rustc 1.63.0-nightly (e71440575 2022-06-02)
```

11. Install cmake using Homebrew:

```
brew install cmake
```

Compile a Substrate Node

To compile the Substrate node template, follow these steps:

1. Clone the Substrate node template repository:

```
git clone https://github.com/substrate-developer-hub/substrate-node-template
```

```
2. Change to the root directory of the cloned repository:
```
cd substrate

-node-template
```
3. Optionally, create a new branch to save your changes:
```
git switch -c my-wip-branch
```
4. Compile the node template:
```
cargo build --release
```
The compilation process may take several minutes due to the required packages.

Once the build is successful, your macOS computer is ready for Substrate development.

These instructions provide a clear and concise guide to setting up a Substrate development environment on macOS.

## **Windows Development Environment Setup**

You can install Rust and set up a Substrate development environment on Windows computers.

## **Before You Begin**

Before installing Rust and setting up your development environment on Windows, ensure that your computer meets the following requirements:

- Operating system: Windows 7 or later.
- Processor: 1.5 GHz or faster.
- Memory: At least 4 GB RAM.
- Storage: At least 20 GB available space.
- Broadband internet connection.
```

Install Visual Studio Build Tools

To compile native code, you'll need to install the Visual Studio Build Tools. Follow these steps:

1. Download Visual Studio Build Tools from the official Microsoft website: https://visualstudio.microsoft.com/downloads/.
2. Run the installer and select the "C++ build tools" workload.
3. Proceed with the installation, selecting the default options.

Installation

To install Rust and set up your development environment on Windows, follow these steps:

1. Open a web browser and go to the Rust website: https://www.rust-lang.org/tools/install.
2. Click the "Download" button to download the Rust installation executable (rustup-init.exe).
3. Run the downloaded executable file (rustup-init.exe).
4. Follow the prompts to proceed with the installation. Use the default options unless you have specific requirements.
5. During the installation, select the default installation directory (usually "C:\Users\<your_username>\.cargo").
6. After the installation is complete, open a Command Prompt or PowerShell window.

Update Environment Variables

To ensure the Rust compiler and cargo (Rust package manager) are available in your command prompt, follow these steps:

1. Open the Start menu and search for "environment variables."
2. Click on "Edit the system environment variables."
3. In the System Properties window, click the "Environment Variables" button.
4. In the "User variables" section, select the "Path" variable and click the "Edit" button.
5. Click the "New" button and add the following two paths:
   - C:\Users\<your_username>\.cargo\bin
   - C:\Program Files (x86)\Microsoft Visual Studio\2019\BuildTools\VC\Tools\MSVC\<version>\bin\Hostx64\x64 (replace `<version>` with the appropriate version number, e.g., 14.29.30037)
6. Click "OK" to save the changes and close all the windows.

Verify the Installation

To verify that Rust is installed correctly, open a new Command Prompt or PowerShell window and run the following command:
```
rustc --version
```
You should see output similar to this:
```
rustc x.y.z (abcabcabc yyyy-mm-dd)
```

Compile a Substrate Node

To compile the Substrate node template, follow these steps:

1. Open a Command Prompt or PowerShell window.
2. Change the current directory to the location where you want to clone the node template repository, using the `cd` command.
3. Clone the Substrate node template repository by running the following command:
```
git clone https://github.com/substrate-developer-hub/substrate-node-template
```
4. Change to the root directory of the cloned repository:
```
cd substrate-node-template
```
5. Compile the node template:
```
cargo build --release
```
The compilation process may take several minutes due to the required packages.

Once the build is successful, your Windows computer is ready for Substrate development.

These instructions provide a clear and concise guide to setting up a Substrate development environment on Windows.