Polkadot | Blockchain Academy

Introduction to FRAME

What is FRAME?

FRAME is a Rust framework for more easily building Substrate runtimes.

Explaining FRAME Concisely

Writing the Sudo Pallet:

Without FRAME: 2210 lines of code.

With FRAME: 310 lines of code.

7x Smaller.

Goals of FRAME

- Make it easy and concise for developers to do development.
- Provide maximum flexibility and compatibility for pallet developers.
- Provide maximum modularity for runtime developers.
- Be as similar to vanilla Rust as possible.

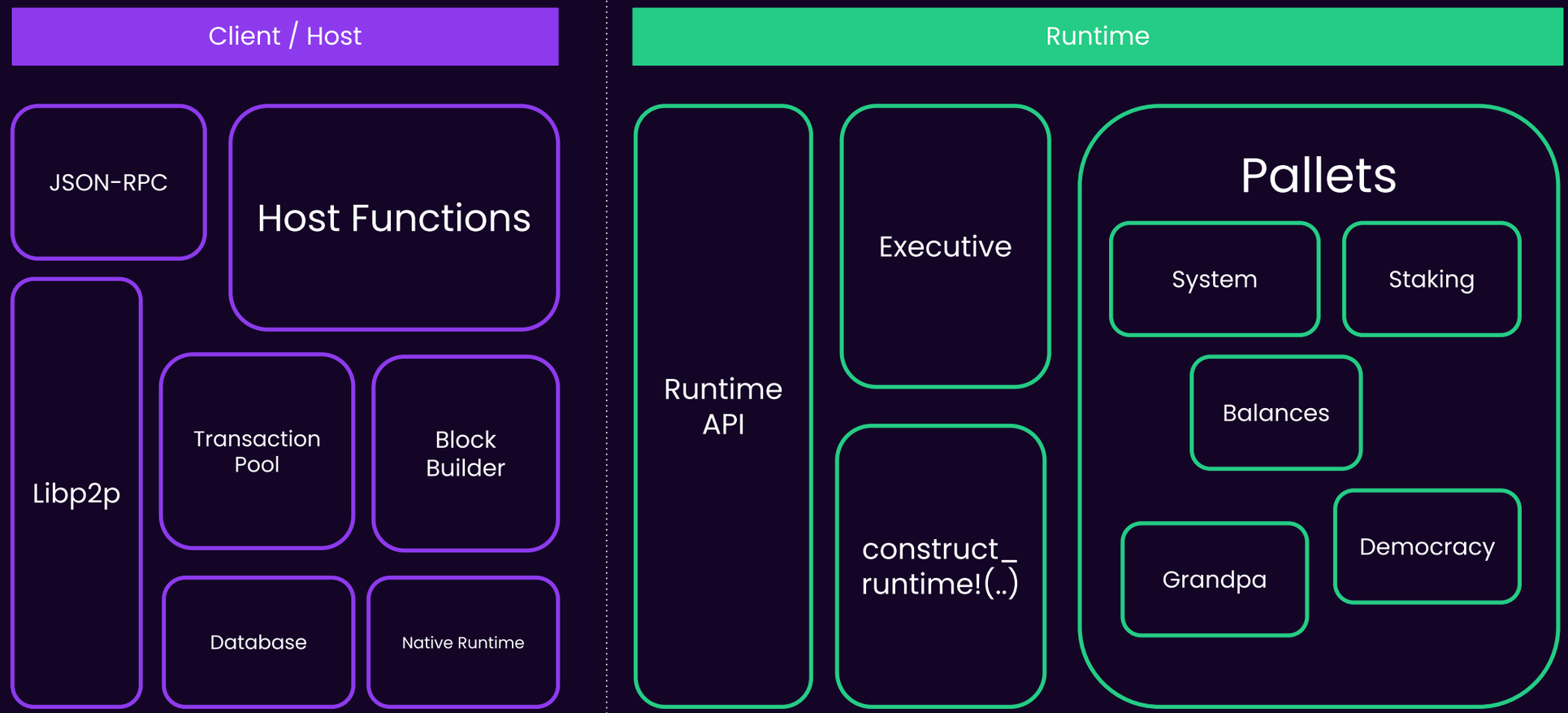
Building Blocks of FRAME

- FRAME Development
  - Pallets
  - Macros
- FRAME Coordination
  - FRAME System
  - FRAME Executive
  - Construct Runtime

Polkadot | Blockchain Academy

Pallets

FRAME takes the opinion that the blockchain runtime should be composed of individual modules. We call these Pallets.



Building Blocks of Pallets

Pallets are composed of multiple parts common for runtime development:

- Dispatchable extrinsics
- Storage items
- Hooks for:
  - Block initialization,
  - Finalizing block (*!= block finality i.e. GRANDPA*)

More Building Blocks of Pallets

And some less important ones:

- Events
- Errors
- Custom validation/communication with tx-pool
- Offchain workers
- A lot more! but you will learn about them later.

"Shell" Pallet

```rust
pub use pallet::*;

#[frame_support::pallet]
pub mod pallet {
  use frame_support::pallet_prelude::*;
  use frame_system::pallet_prelude::*;

  #[pallet::pallet]
  #[pallet::generate_store(pub(super) trait Store)]
  pub struct Pallet<T>(_);

  #[pallet::config]  // snip
  #[pallet::event]   // snip
  #[pallet::error]   // snip
  #[pallet::storage] // snip
  #[pallet::call]    // snip
}
```

FRAME Macros

Rust allows you to write Macros, which is code that generates code.

FRAME uses Macros to simplify the development of Pallets, while keeping all of the benefits of using Rust.

We will look more closely at each attribute throughout this module.

See For Yourself

- `wc -l` will show the number of lines of a file.
- `cargo expand` will expand the macros to "pure" Rust.

```
→  substrate git:(master) ✗ wc -l frame/sudo/src/lib.rs
     310 frame/sudo/src/lib.rs

→  substrate git:(master) ✗ cargo expand -p pallet-sudo | wc -l
     2210
```

FRAME System

The FRAME System is a Pallet which is assumed to always exist when using FRAME. You can see that in the `Config` of every Pallet:

```
#[pallet::config]
pub trait Config: frame_system::Config { ... }
```

It contains all the most basic functions and types needed for a blockchain system. Also contains many low level extrinsics to manage your chain directly.

- Block Number
- Accounts
- Hash
- etc...

- `BlockNumberFor<T>`
- `frame_system::Pallet::<T>::block_number()`
- `T::AccountId`
- `T::Hash`
- `T::Hashing::hash(&bytes)`

FRAME Executive

The FRAME Executive is a "coordinator", defining the order that your FRAME based runtime executes.

```rust
/// Actually execute all transitions for `block`.
pub fn execute_block(block: Block) { ... }
```

- Initialize Block
  - `on_runtime_upgrade` and `on_initialize` hooks
- Initial Checks
- Signature Verification
- Execute Extrinsics
  - `on_idle` and `on_finalize` hooks
- Final Checks

Construct Runtime

Your final runtime is composed of Pallets, which are brought together with the `construct_runtime!` macro.

```
// Create the runtime by composing the FRAME pallets that were previously configured.
construct_runtime!(
    pub struct Runtime {
        System: frame_system,
        RandomnessCollectiveFlip: pallet_randomness_collective_flip,
        Timestamp: pallet_timestamp,
        Aura: pallet_aura,
        Grandpa: pallet_grandpa,
        Balances: pallet_balances,
        TransactionPayment: pallet_transaction_payment,
        Sudo: pallet_sudo,
        // Include the custom logic from the pallet-template in the runtime.
        TemplateModule: pallet_template,
    }
);
```

Pallet Configuration

Before you can add a Pallet to the final runtime, it needs to be configured as defined in the `Config`.

In the Pallet:

```
/// The timestamp pallet configuration trait.
#[pallet::config]
pub trait Config: frame_system::Config {
  type Moment: Parameter + Default + AtLeast32B

  type OnTimestampSet: OnTimestampSet<Self::Mom

  #[pallet::constant]
  type MinimumPeriod: Get<Self::Moment>;

  type WeightInfo: WeightInfo;
}
```

In the Runtime:

```
/// The timestamp pallet configuration.

impl pallet_timestamp::Config for Runtime {
  type Moment = u64;
  type OnTimestampSet = Aura;


  type MinimumPeriod = ConstU64<{ SLOT_DURATIO

  type WeightInfo = ();
}
```