sui

Intro to **Dynamic Fields**

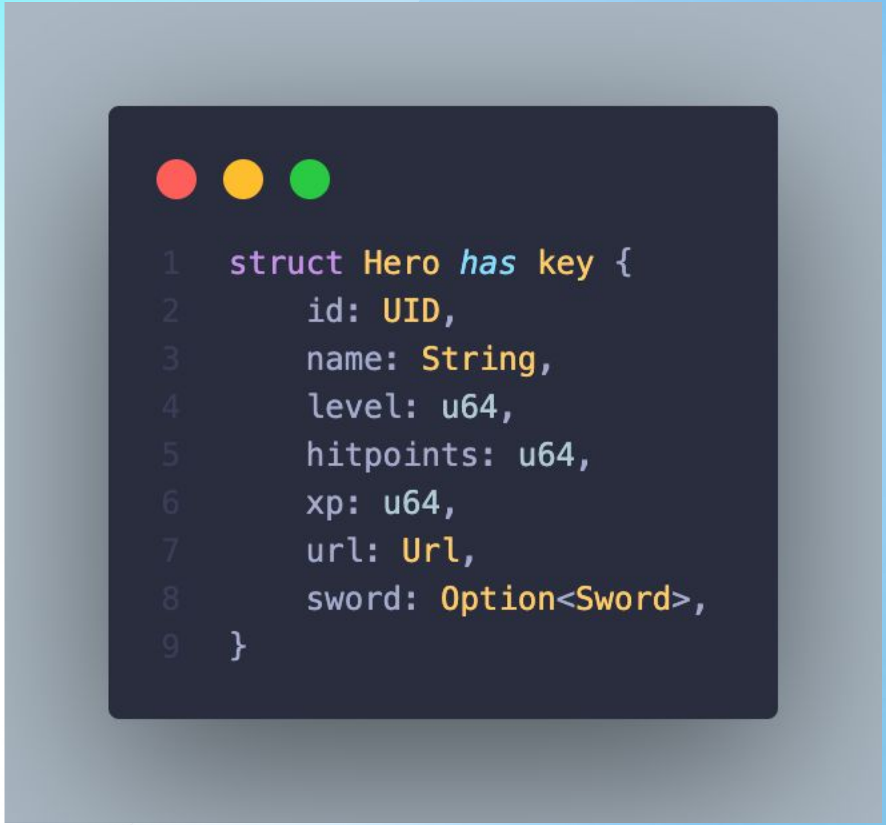
March 8, 2023

Agenda

Why do we need **Dynamic Fields**?

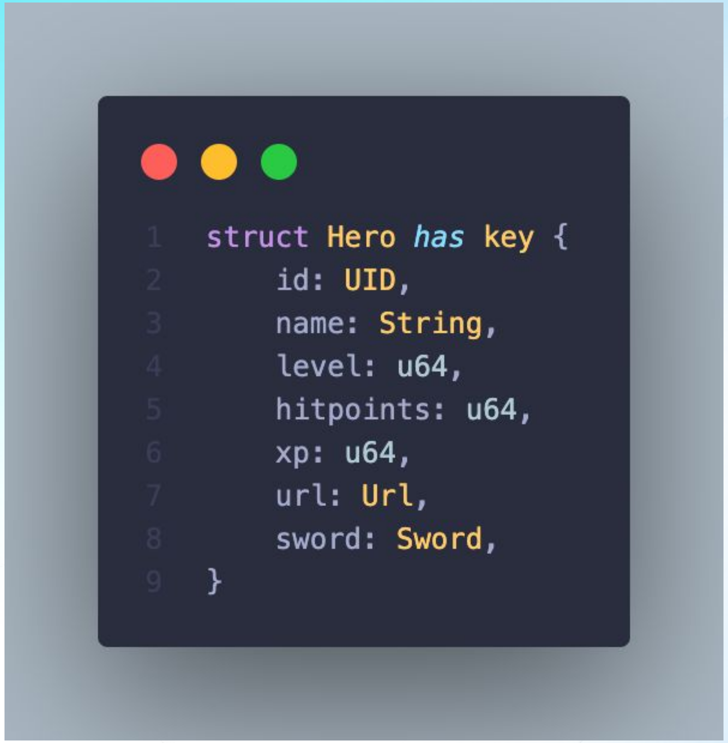
Review:
Object wrapping

sui

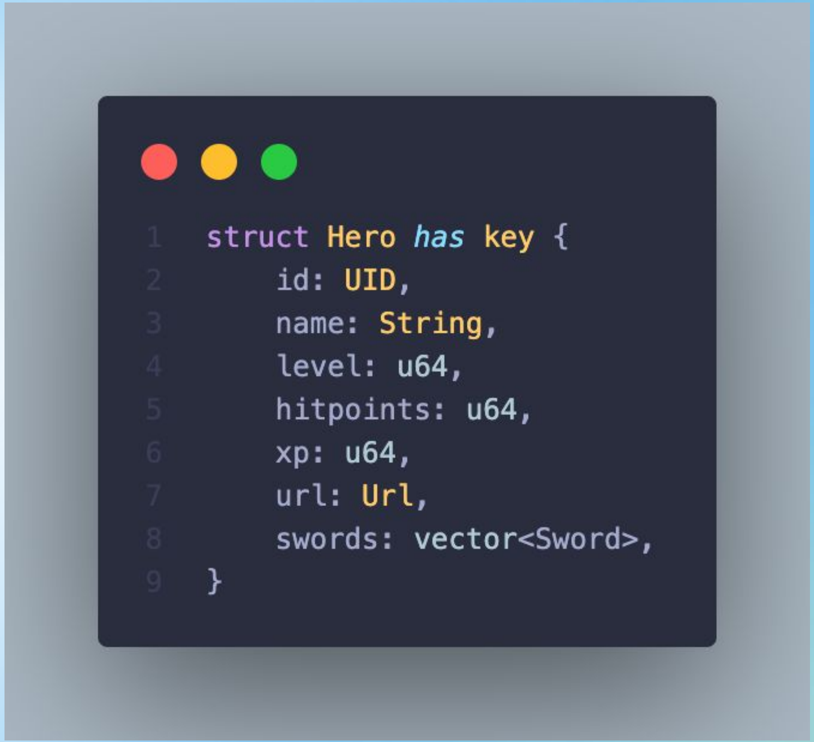


```
struct Hero has key {
    id: UID,
    name: String,
    level: u64,
    hitpoints: u64,
    xp: u64,
    url: Url,
    sword: Option<Sword>,
}
```

sui



```
struct Hero has key {
    id: UID,
    name: String,
    level: u64,
    hitpoints: u64,
    xp: u64,
    url: Url,
    sword: Sword,
}
```



```
struct Hero has key {
    id: UID,
    name: String,
    level: u64,
    hitpoints: u64,
    xp: u64,
    url: Url,
    swords: vector<Sword>,
}
```

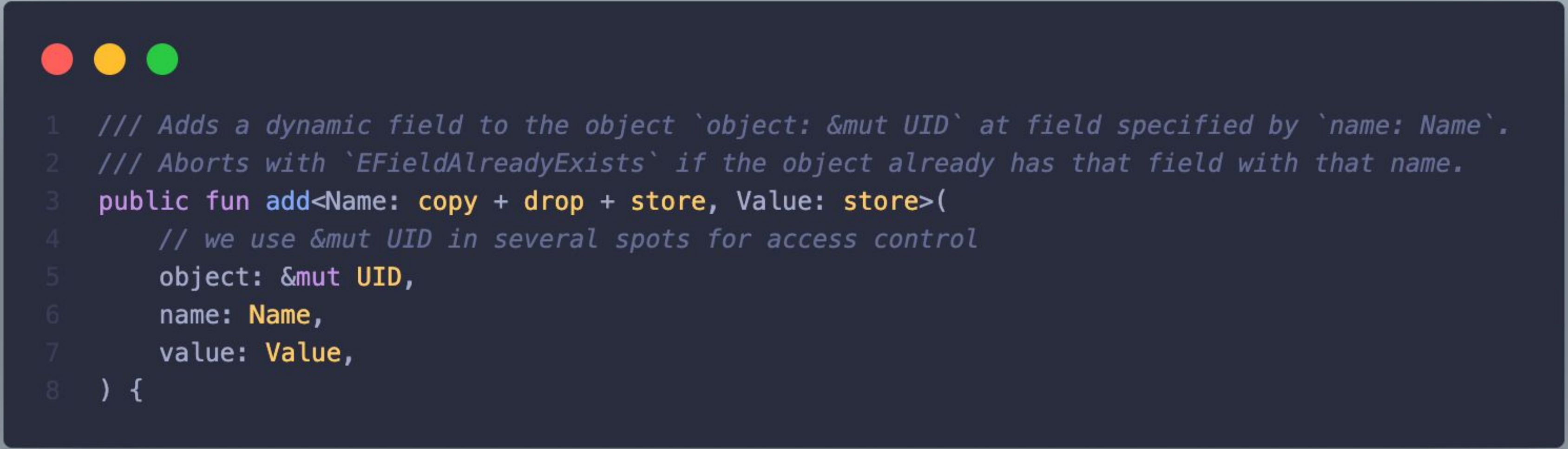
sui

What's wrong with this approach?

Thus **Dynamic Fields** were born.

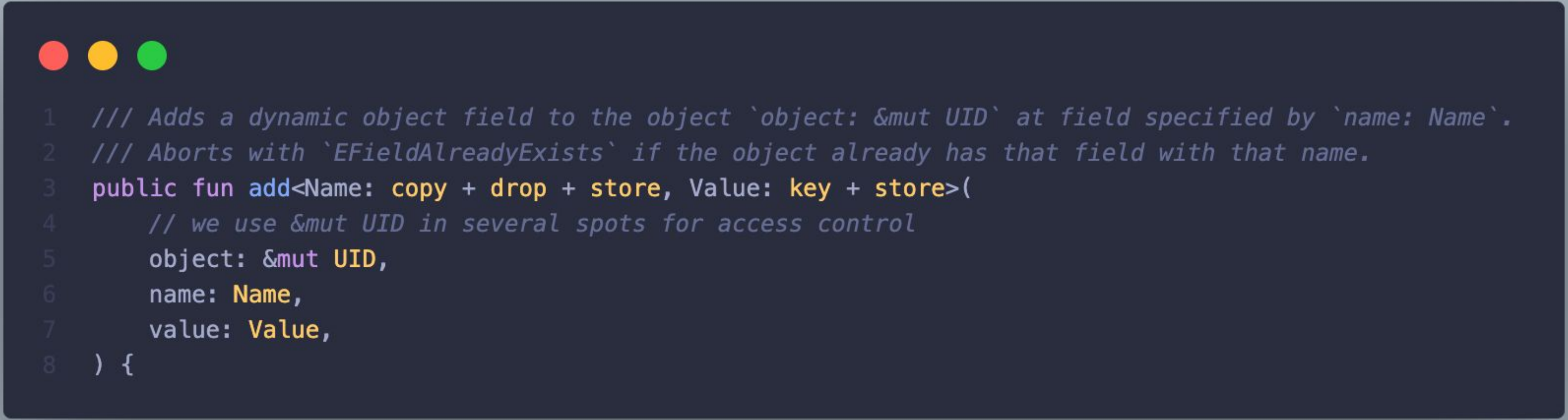
Dynamic Fields
VS.
Dynamic Object Fields

sui



```
/// Adds a dynamic field to the object `object: &mut UID` at field specified by `name: Name`.
/// Aborts with `EFieldAlreadyExists` if the object already has that field with that name.
public fun add<Name: copy + drop + store, Value: store>(
    // we use &mut UID in several spots for access control
    object: &mut UID,
    name: Name,
    value: Value,
) {
```

sui



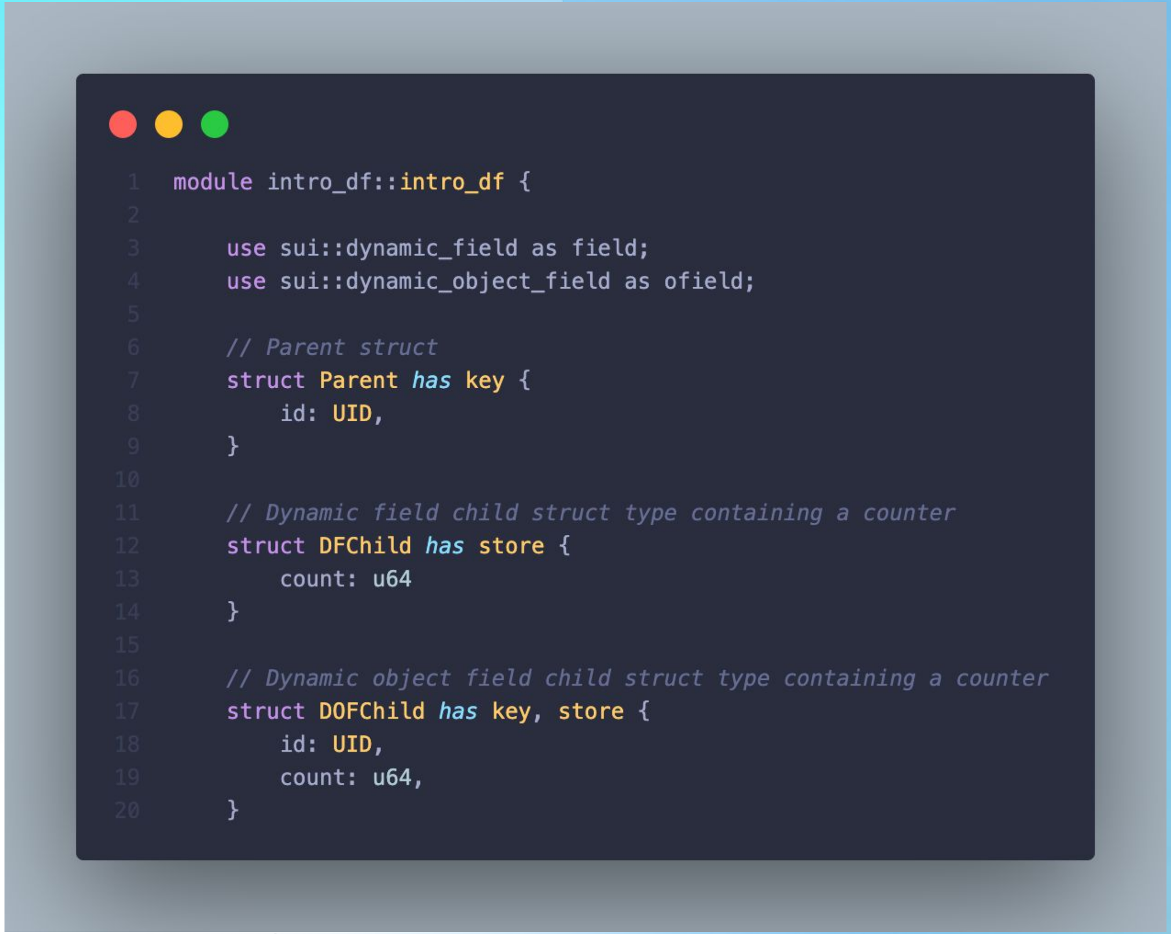
```
/// Adds a dynamic object field to the object `object: &mut UID` at field specified by `name: Name`.
/// Aborts with `EFieldAlreadyExists` if the object already has that field with that name.
public fun add<Name: copy + drop + store, Value: key + store>(
    // we use &mut UID in several spots for access control
    object: &mut UID,
    name: Name,
    value: Value,
) {
```

Why do we have both?

sui

Dynamic Fields

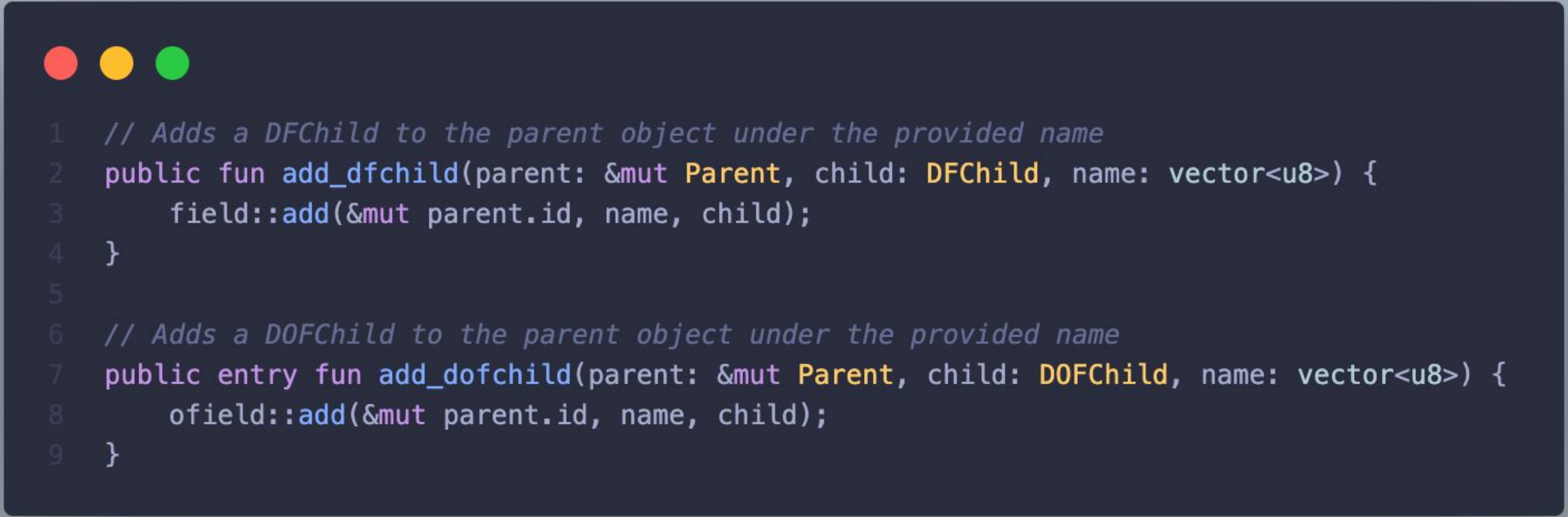
```
sui move new intro_df
```



```
module intro_df::intro_df {

    use sui::dynamic_field as field;
    use sui::dynamic_object_field as ofield;

    // Parent struct
    struct Parent has key {
        id: UID,
    }

    // Dynamic field child struct type containing a counter
    struct DFChild has store {
        count: u64
    }

    // Dynamic object field child struct type containing a counter
    struct DOFChild has key, store {
        id: UID,
        count: u64,
    }
```

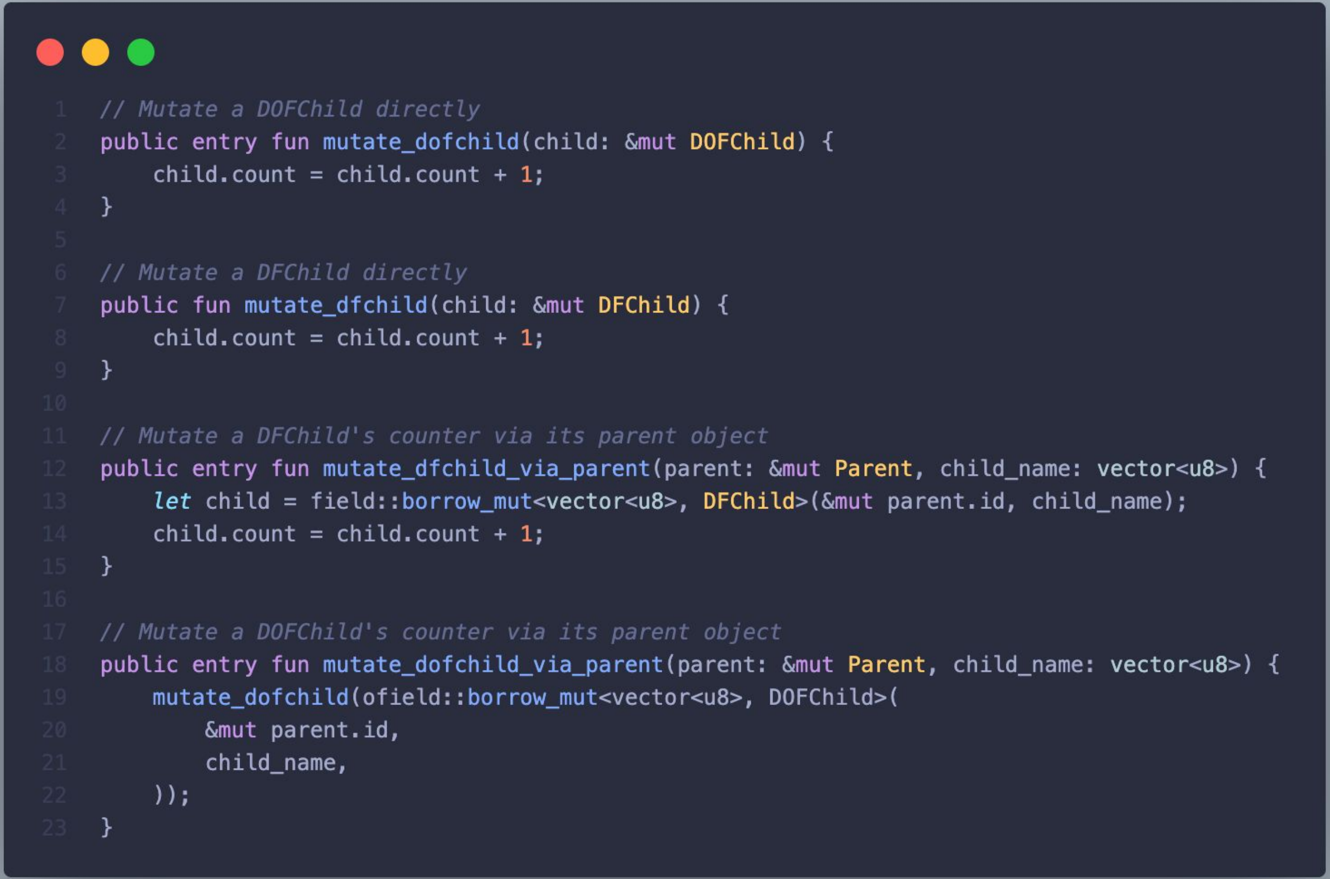
sui

Adapted from: github.com/sui-foundation/sui-move-intro-course



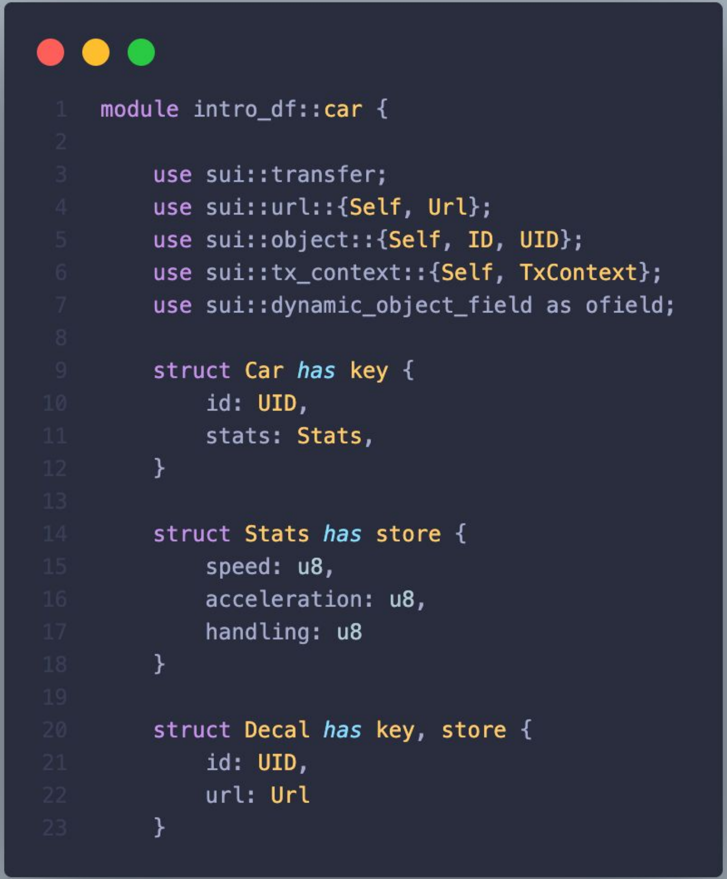
```
// Adds a DFChild to the parent object under the provided name
public fun add_dfchild(parent: &mut Parent, child: DFChild, name: vector<u8>) {
    field::add(&mut parent.id, name, child);
}

// Adds a DOFChild to the parent object under the provided name
public entry fun add_dofchild(parent: &mut Parent, child: DOFChild, name: vector<u8>) {
    ofield::add(&mut parent.id, name, child);
}
```

sui

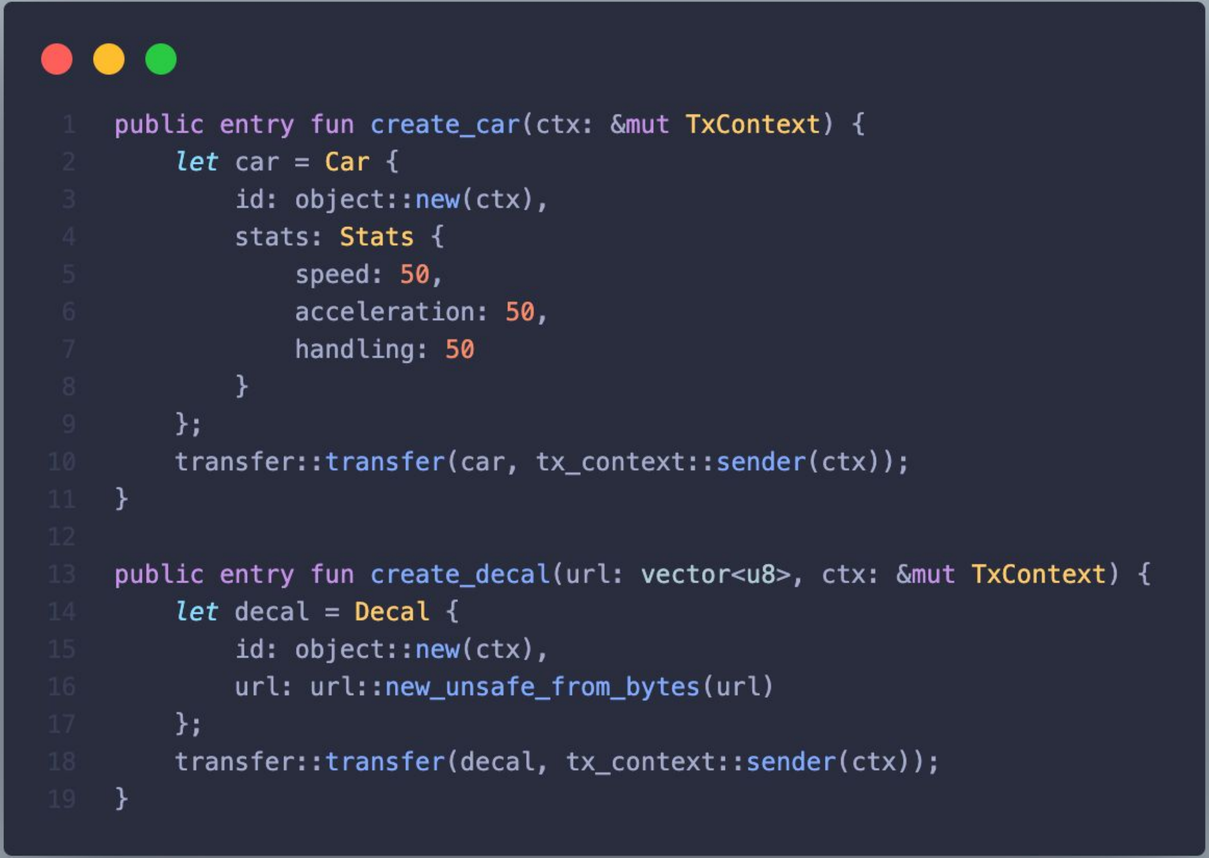


```
// Mutate a DOFChild directly
public entry fun mutate_dofchild(child: &mut DOFChild) {
    child.count = child.count + 1;
}

// Mutate a DFChild directly
public fun mutate_dfchild(child: &mut DFChild) {
    child.count = child.count + 1;
}

// Mutate a DFChild's counter via its parent object
public entry fun mutate_dfchild_via_parent(parent: &mut Parent, child_name: vector<u8>) {
    let child = field::borrow_mut<vector<u8>, DFChild>(&mut parent.id, child_name);
    child.count = child.count + 1;
}

// Mutate a DOFChild's counter via its parent object
public entry fun mutate_dofchild_via_parent(parent: &mut Parent, child_name: vector<u8>) {
    mutate_dofchild(ofield::borrow_mut<vector<u8>, DOFChild>(
        &mut parent.id,
        child_name,
    ));
}
```

sui

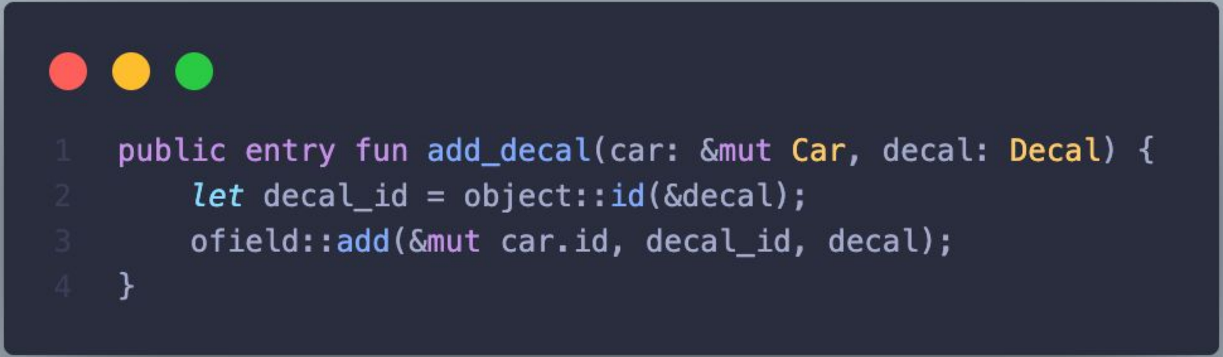


```
module intro_df::car {

    use sui::transfer;
    use sui::url::{Self, Url};
    use sui::object::{Self, ID, UID};
    use sui::tx_context::{Self, TxContext};
    use sui::dynamic_object_field as ofield;

    struct Car has key {
        id: UID,
        stats: Stats,
    }

    struct Stats has store {
        speed: u8,
        acceleration: u8,
        handling: u8
    }

    struct Decal has key, store {
        id: UID,
        url: Url
    }
```

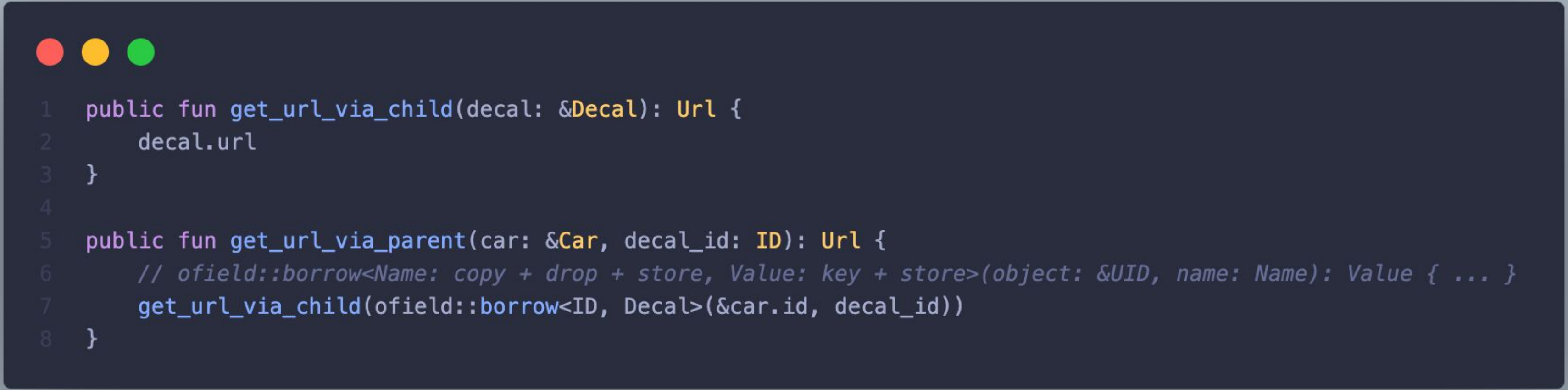


```
public entry fun create_car(ctx: &mut TxContext) {
    let car = Car {
        id: object::new(ctx),
        stats: Stats {
            speed: 50,
            acceleration: 50,
            handling: 50
        }
    };
    transfer::transfer(car, tx_context::sender(ctx));
}

public entry fun create_decal(url: vector<u8>, ctx: &mut TxContext) {
    let decal = Decal {
        id: object::new(ctx),
        url: url::new_unsafe_from_bytes(url)
    };
    transfer::transfer(decal, tx_context::sender(ctx));
}
```

sui



```
public entry fun add_decal(car: &mut Car, decal: Decal) {
    let decal_id = object::id(&decal);
    ofield::add(&mut car.id, decal_id, decal);
}
```



```
public fun get_url_via_child(decal: &Decal): Url {
    decal.url
}

public fun get_url_via_parent(car: &Car, decal_id: ID): Url {
    // ofield::borrow<Name: copy + drop + store, Value: key + store>(object: &UID, name: Name): Value { ... }
    get_url_via_child(ofield::borrow<ID, Decal>(&car.id, decal_id))
}
```

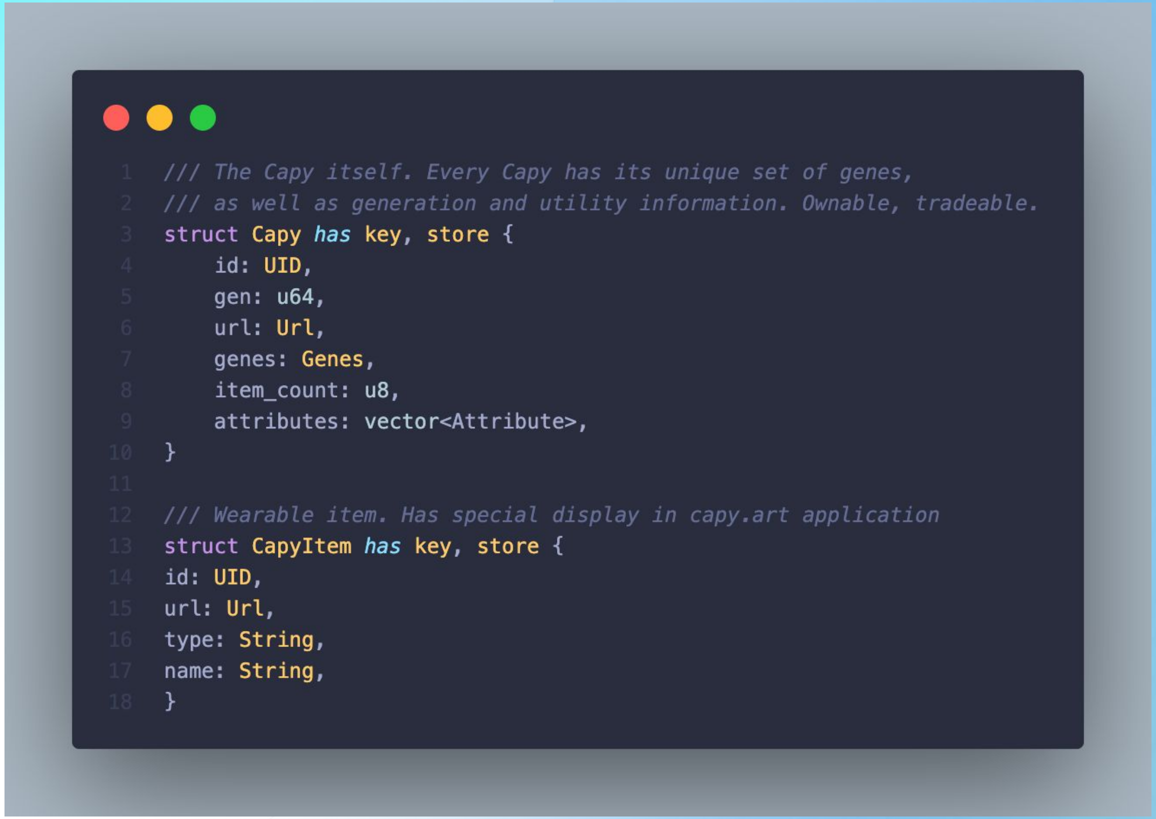
sui

How can we utilize **Dynamic Fields** to take **composability** to the next level?

sui

capy.art

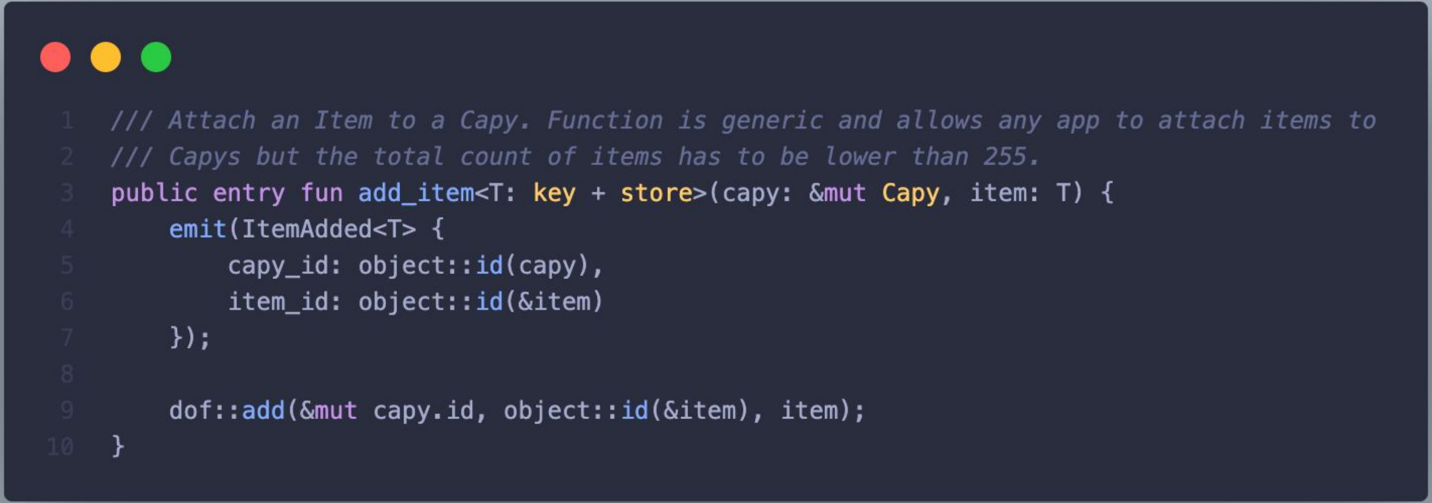
sui



```move
/// The Capy itself. Every Capy has its unique set of genes,
/// as well as generation and utility information. Ownable, tradeable.
struct Capy has key, store {
    id: UID,
    gen: u64,
    url: Url,
    genes: Genes,
    item_count: u8,
    attributes: vector<Attribute>,
}

/// Wearable item. Has special display in capy.art application
struct CapyItem has key, store {
id: UID,
url: Url,
type: String,
name: String,
}
```

capy.move

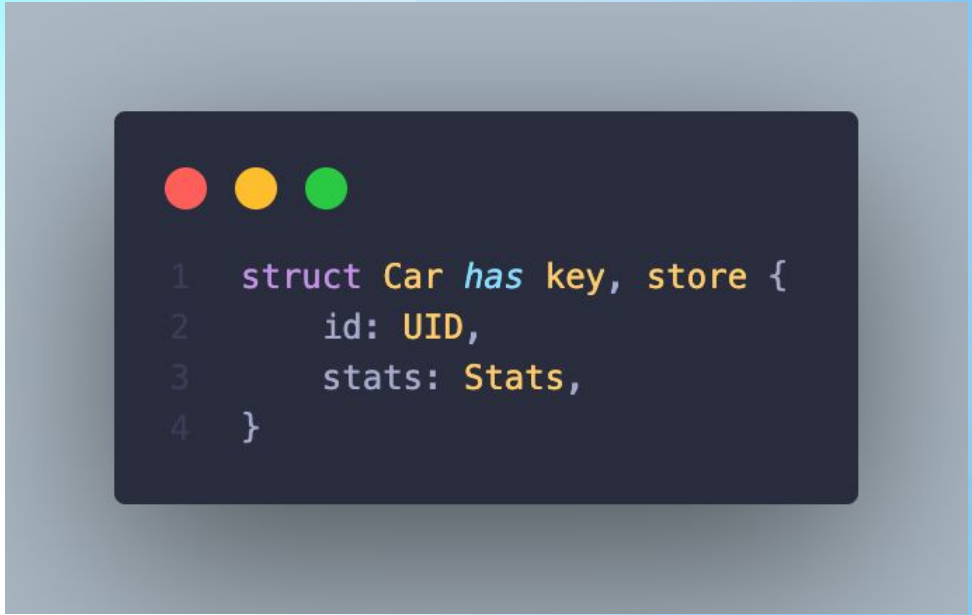
sui



```
/// Attach an Item to a Capy. Function is generic and allows any app to attach items to
/// Capys but the total count of items has to be lower than 255.
public entry fun add_item<T: key + store>(capy: &mut Capy, item: T) {
    emit(ItemAdded<T> {
        capy_id: object::id(capy),
        item_id: object::id(&item)
    });

    dof::add(&mut capy.id, object::id(&item), item);
}
```

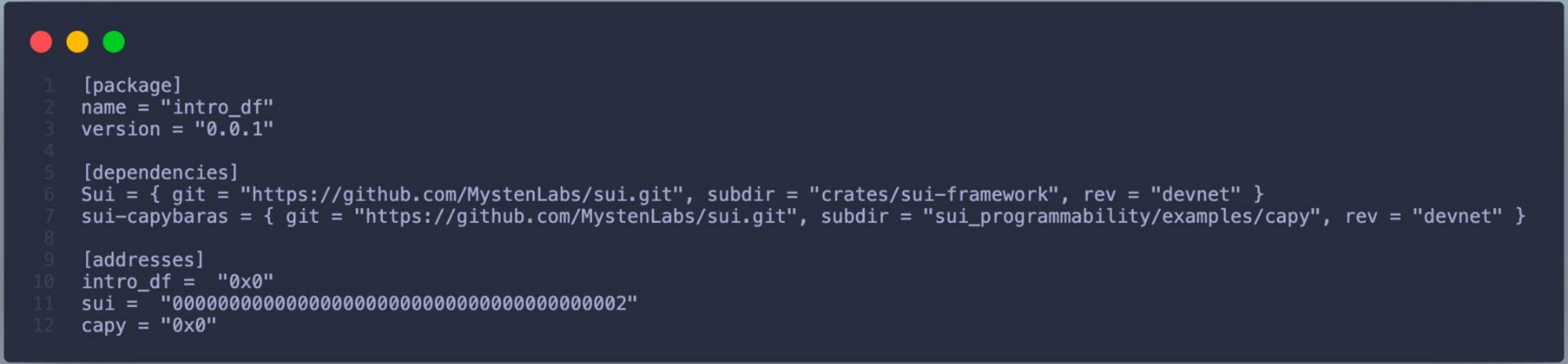
capy.move

sui



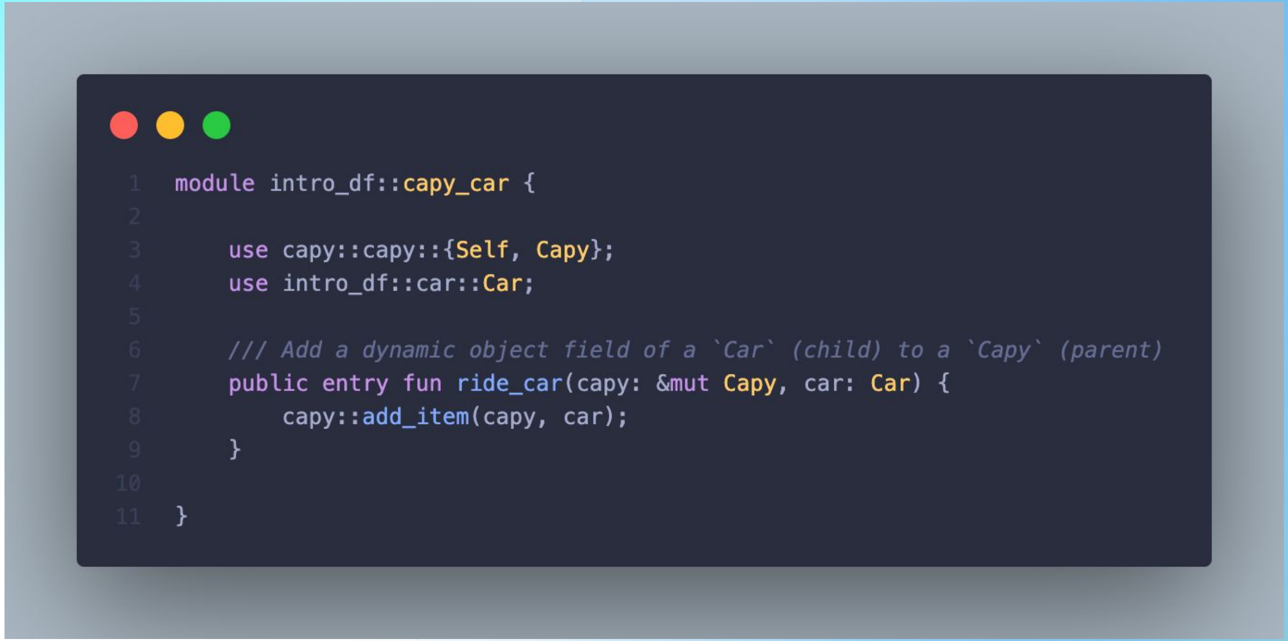
```
struct Car has key, store {
    id: UID,
    stats: Stats,
}
```

car.move



```toml
[package]
name = "intro_df"
version = "0.0.1"

[dependencies]
Sui = { git = "https://github.com/MystenLabs/sui.git", subdir = "crates/sui-framework", rev = "devnet" }
sui-capybaras = { git = "https://github.com/MystenLabs/sui.git", subdir = "sui_programmability/examples/capy", rev = "devnet" }

[addresses]
intro_df =  "0x0"
sui =  "0000000000000000000000000000000000000002"
capy = "0x0"
```

Move.toml



```
module intro_df::capy_car {

    use capy::capy::{Self, Capy};
    use intro_df::car::Car;

    /// Add a dynamic object field of a `Car` (child) to a `Capy` (parent)
    public entry fun ride_car(capy: &mut Capy, car: Car) {
        capy::add_item(capy, car);
    }

}
```

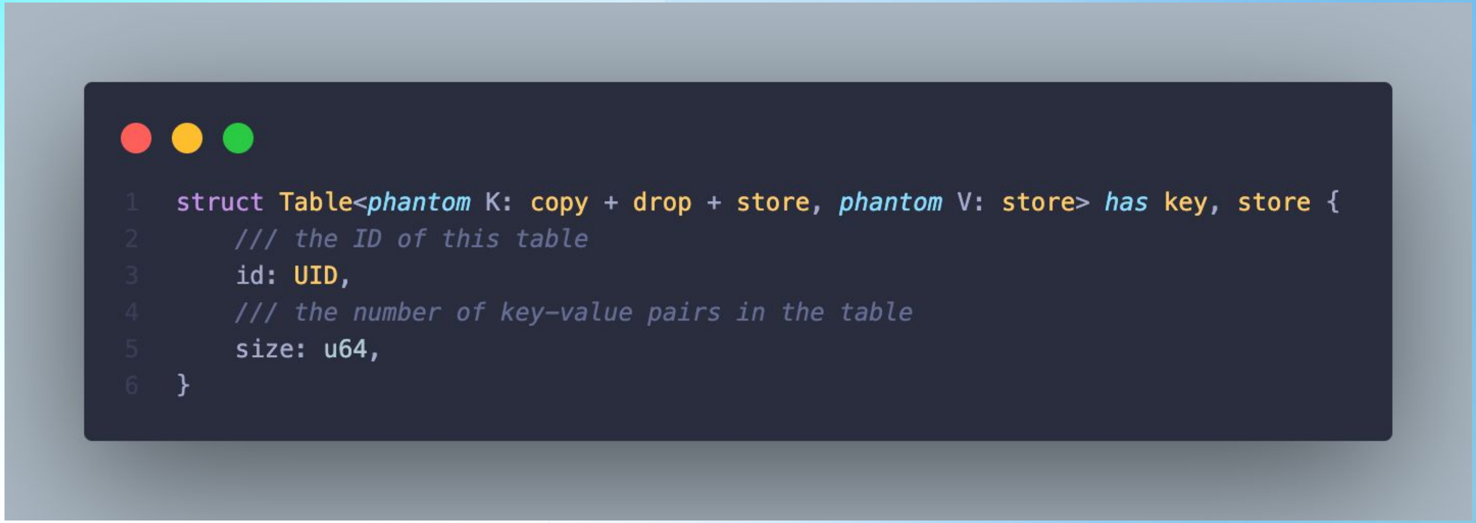
Biggest Takeaway:

You can create composable dApps in only 1 line of code!

sui

Collections

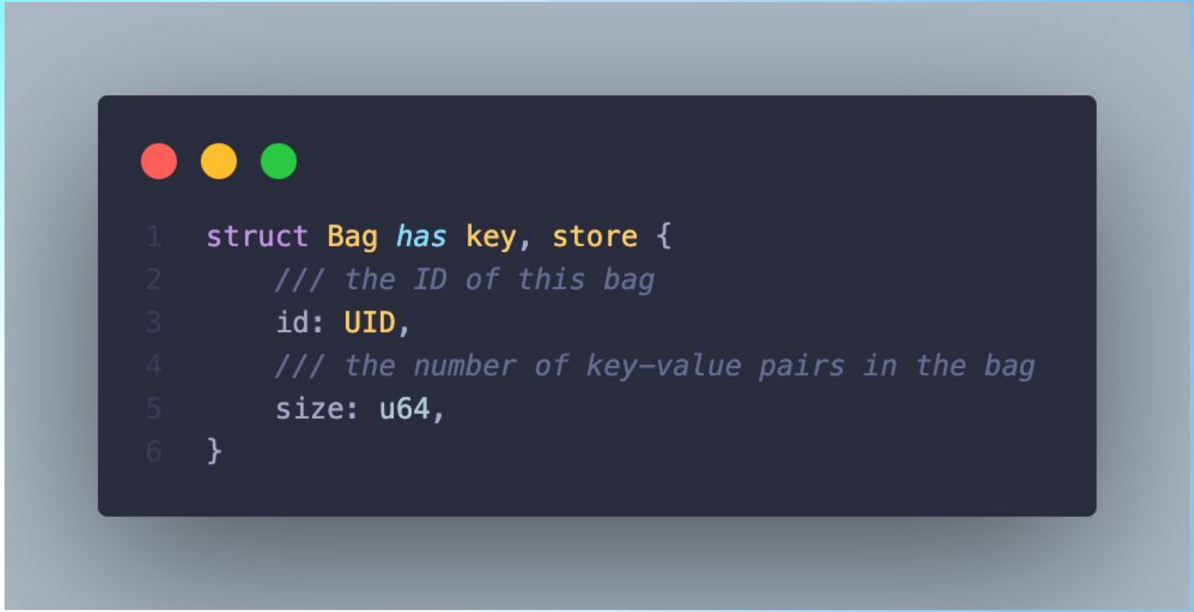
sui



```
struct Table<phantom K: copy + drop + store, phantom V: store> has key, store {
    /// the ID of this table
    id: UID,
    /// the number of key-value pairs in the table
    size: u64,
}
```

table.move

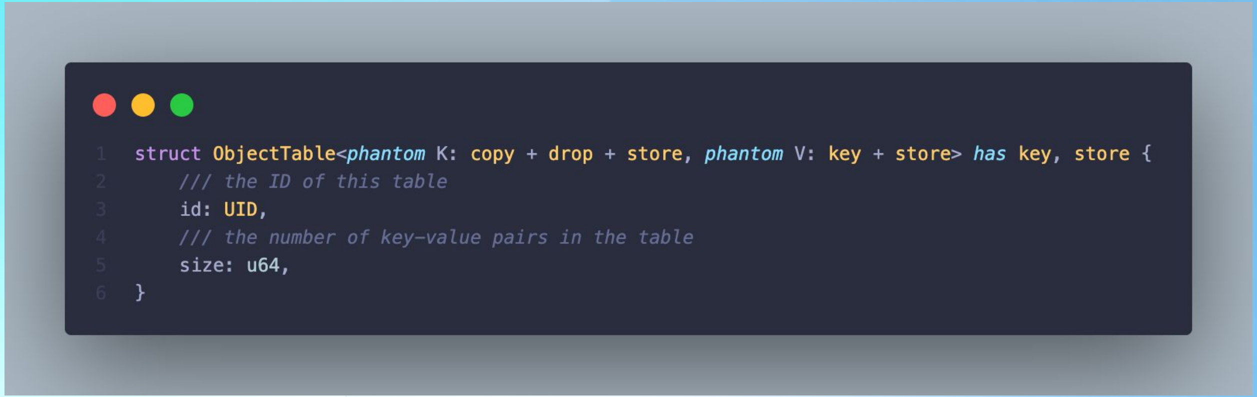
sui



```
struct Bag has key, store {
    /// the ID of this bag
    id: UID,
    /// the number of key-value pairs in the bag
    size: u64,
}
```

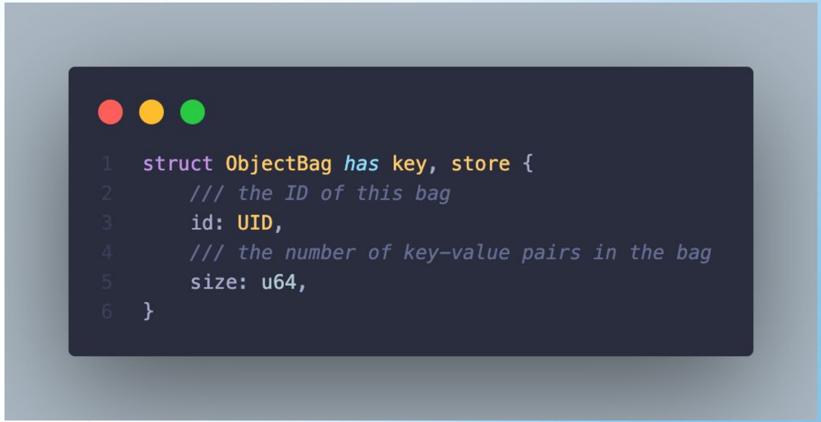
bag.move

sui



```
struct ObjectTable<phantom K: copy + drop + store, phantom V: key + store> has key, store {
    /// the ID of this table
    id: UID,
    /// the number of key-value pairs in the table
    size: u64,
}
```

object_table.move



```
struct ObjectBag has key, store {
    /// the ID of this bag
    id: UID,
    /// the number of key-value pairs in the bag
    size: u64,
}
```

object_bag.move

sui

Recap:

	Non-Object Values	Object Values (has key)
Heterogeneous Map	bag	object_bag
Homogeneous Map	table	object_table

sui

Bibliography/ Further Reading

docs.sui.io/build/programming-with-objects/ch5-dynamic-fields

github.com/sui-foundation/sui-move-intro-course

github.com/MystenLabs/sui/blob/main/sui_programmability/examples/nfts/sources/marketplace.move

forums.sui.io/t/dynamicfield-vs-dynamicobjectfield-why-do-we-have-both/2095

forums.sui.io/t/does-dynamic-field-access-cost-grow-with-the-number-of-fields/2301

What’s Next!

Next Workshop:
Project Showcase: **On-chain RPG**
(Part 1)

March 13, 2023

sui

Sui Vietnam Builder House!

lu.ma/sui.vietnam

Survey + Questions?

Twitter: @0xShayan

sui

sui