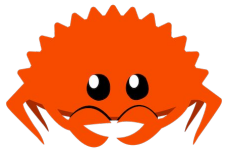
A Distributed ECommerce Software

implemented by **_undefined** group



una producción de

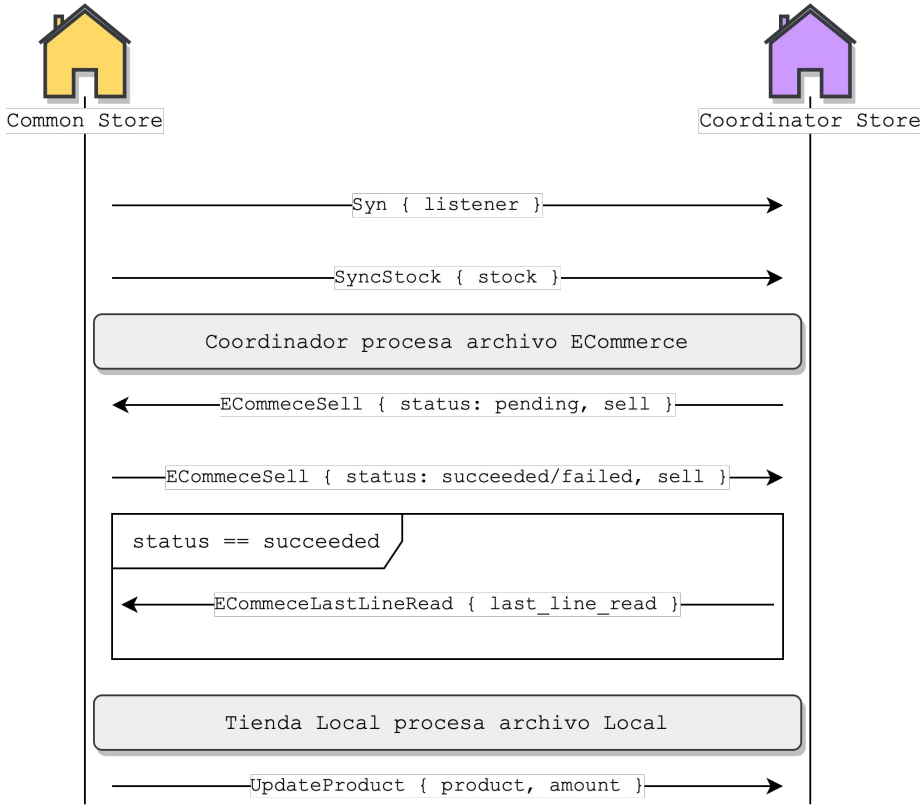
Francisco O. Lorda

&

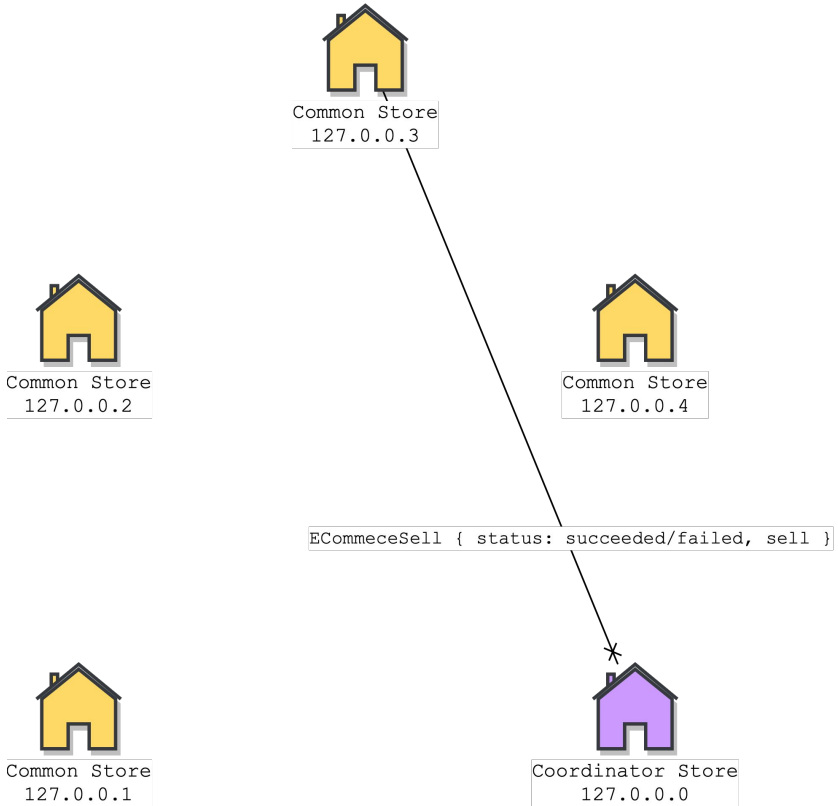
Carolina Di Matteo

Definición del Protocolo de Envío de Mensajes

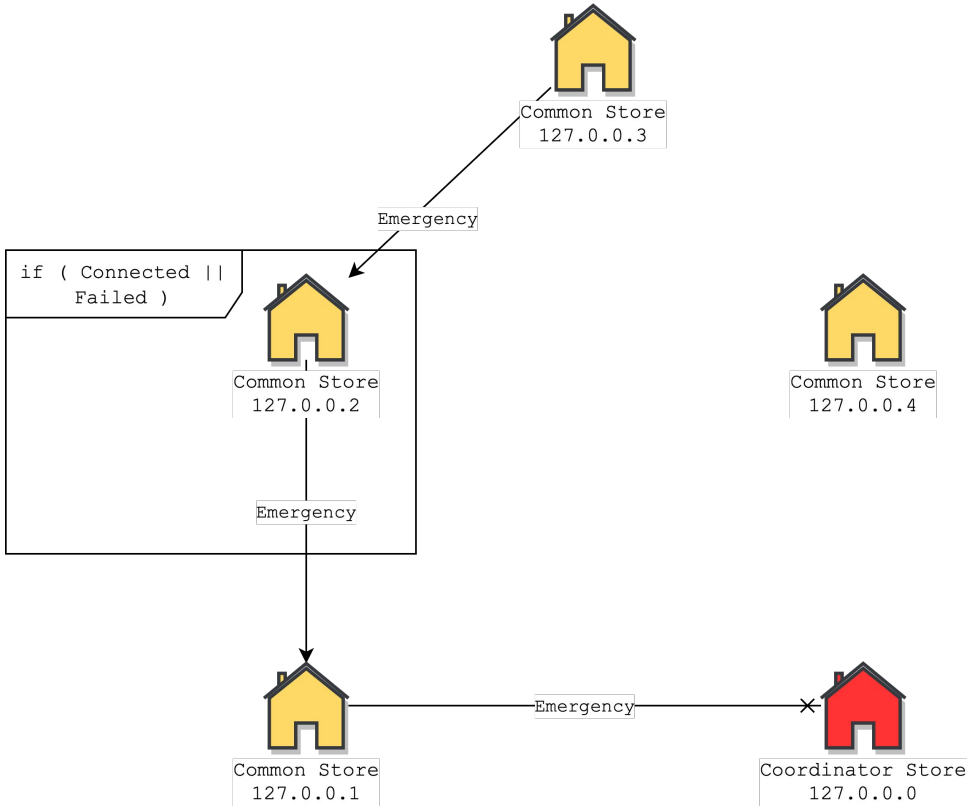
Conexión Inicial entre una Tienda Local y el Coordinador



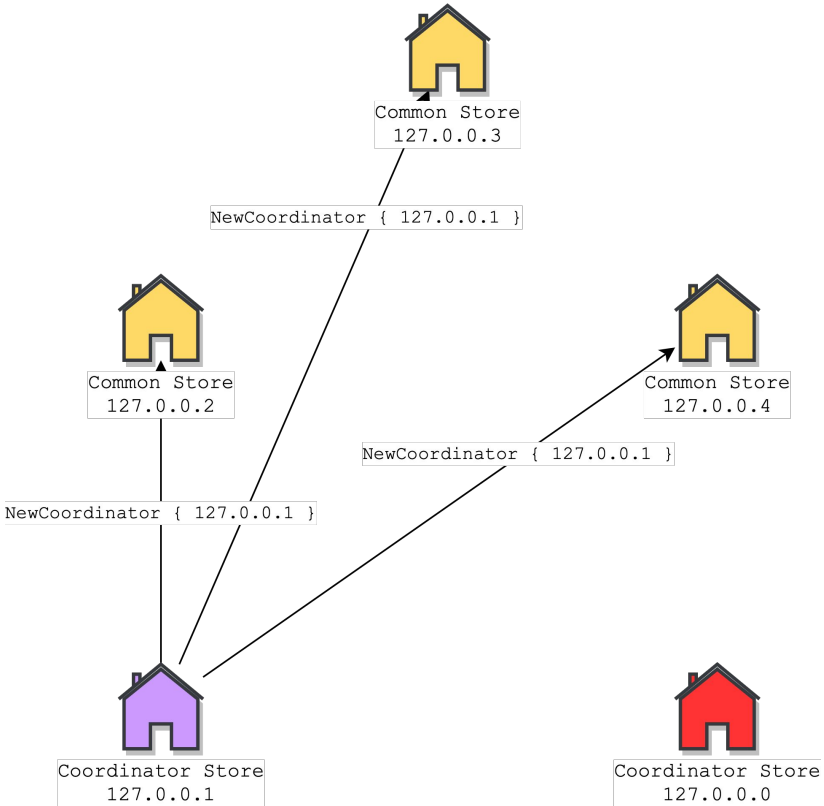
Caída del Coordinador y Reconstrucción de la Red



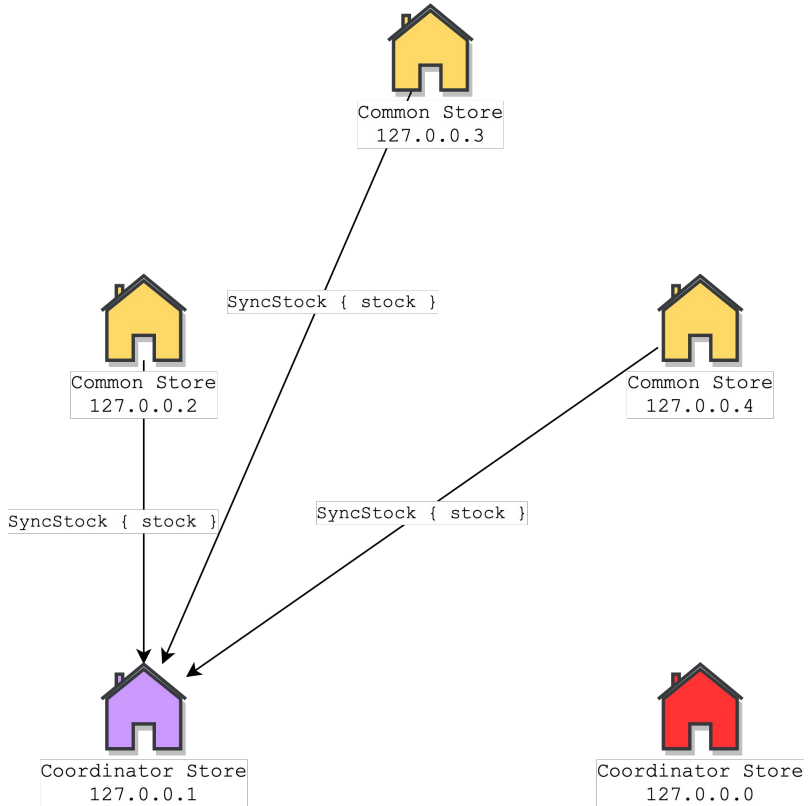
Caída del Coordinador y Reconstrucción de la Red



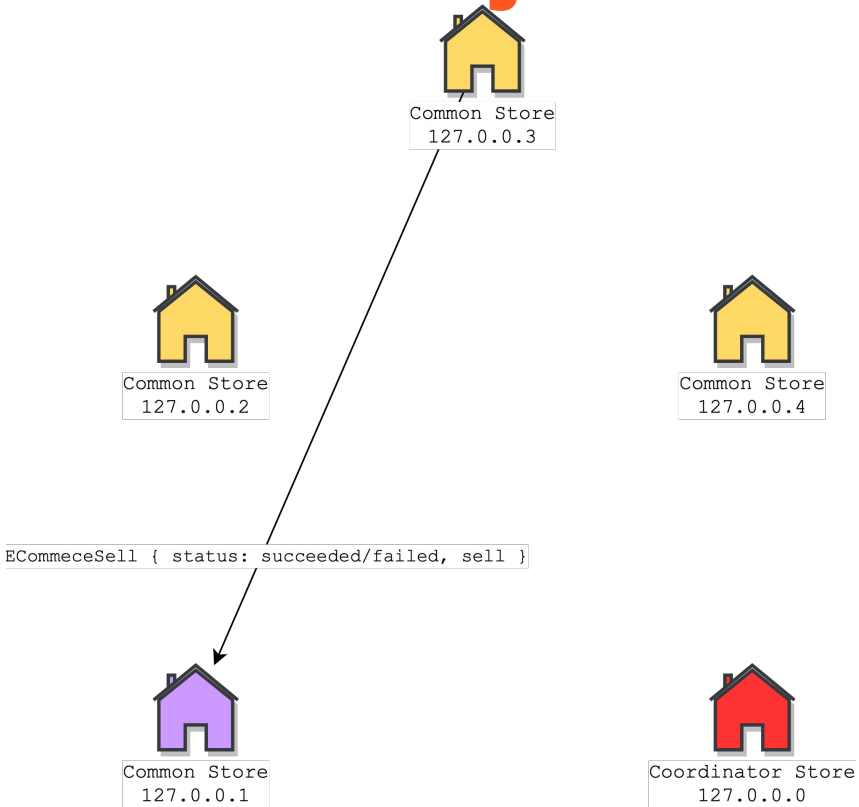
Caída del Coordinador y Reconstrucción de la Red



Caída del Coordinador y Reconstrucción de la Red

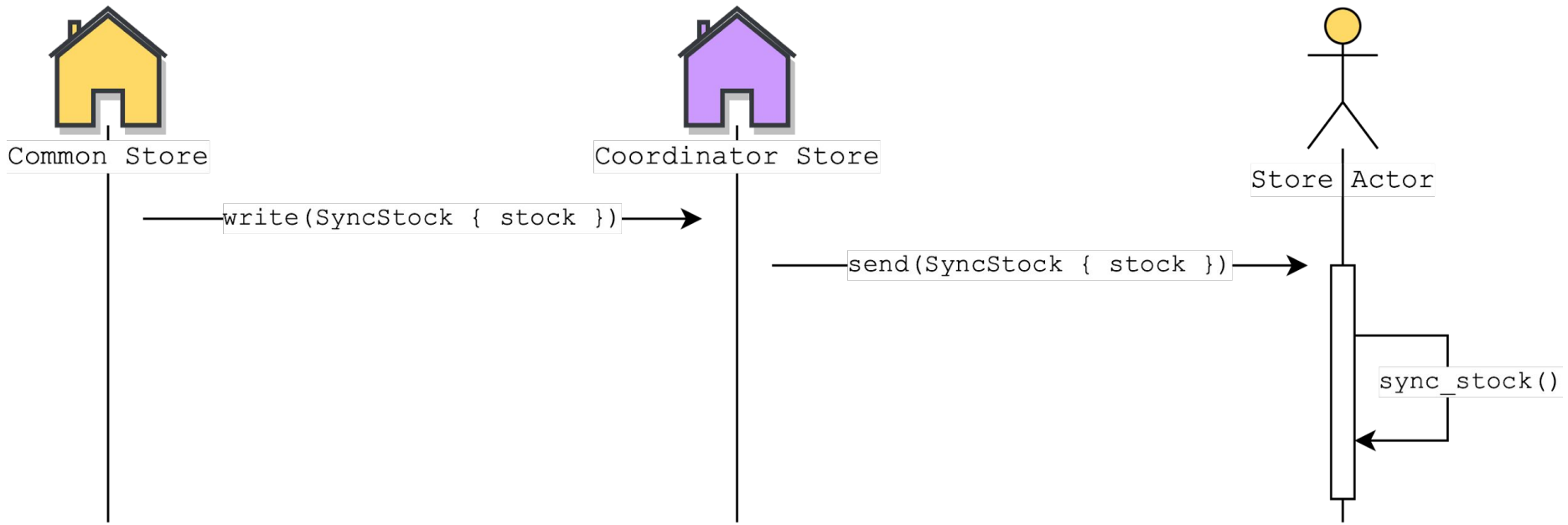


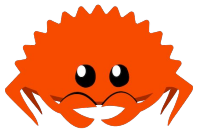
Caída del Coordinador y Reconstrucción de la Red - Reenvío del Mensaje Perdido



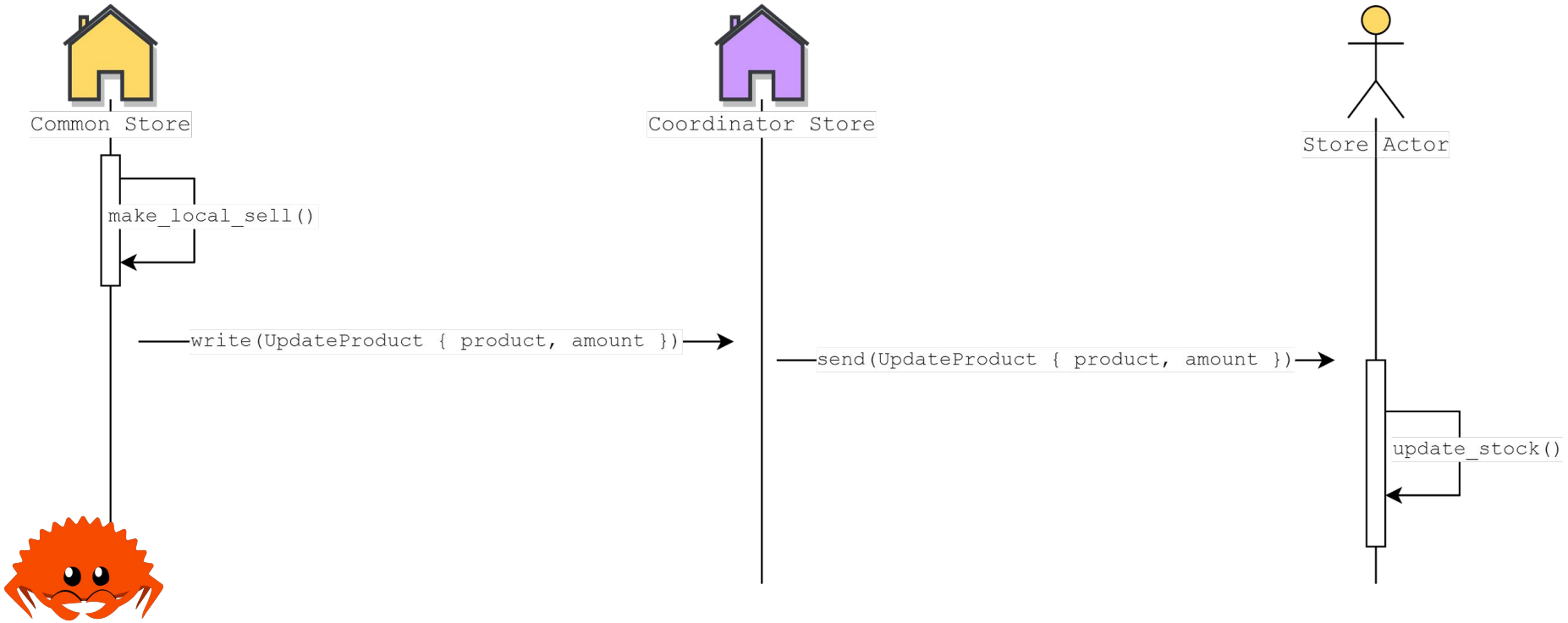
Utilización de Actores

Sincronización Inicial de Stock

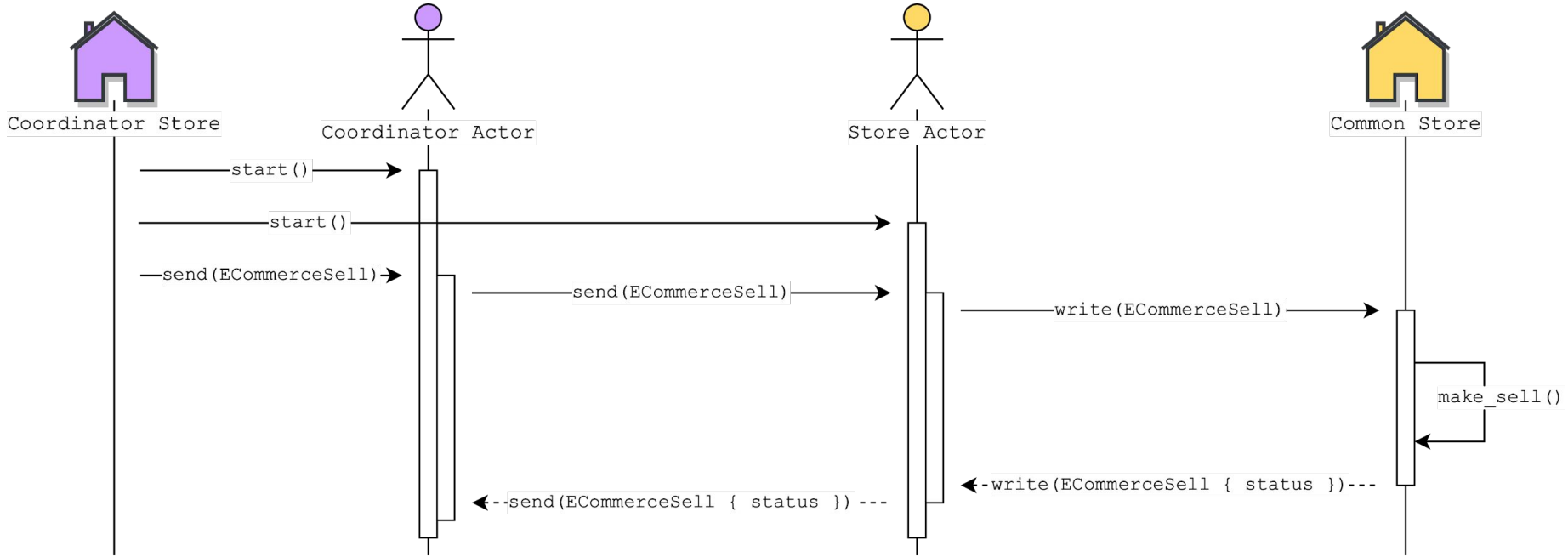




Ejecución de una Venta Local y Actualización de Stock



Ejecución de una Venta ECommerce



Código Fuente

Estructuras Principales

```rust
pub struct CoordinatorStore {
    store: Store,
    actor: Addr<CoordinatorActor>,
    stores_streams: Arc<Mutex<Vec<TcpStream>>>,
    tcp_listener: TcpListener,
}
```

```rust
pub struct Store {
    id: StoreId,
    life: Arc<Mutex<StoreLife>>,
    lost_ecommerce_message: Arc<Mutex<Vec<SocketMessage>>>,
    ecommerce_last_line: Arc<Mutex<usize>>,
    local_last_line: Arc<Mutex<usize>>,
    stock: Arc<Mutex<Stock>>,
    listener: Listener,
    neighbours: Arc<Mutex<Vec<Listener>>>,
}
```

```rust
pub struct CommonStore {
    store: Store,
    coordinator_stream: Option<Arc<Mutex<TcpStream>>>,
    connection_state: Arc<Mutex<ConnectionState>>,
}
```

Actor del Coordinador y sus Mensajes

```rust
pub struct CoordinatorActor {
    stores: Vec<Addr<StoreActor>>,
    stock: Arc<Mutex<Stock>>,
    rejected_sells: HashMap<Sell, Vec<Addr<StoreActor>>>,
    stores_taken: HashSet<Addr<StoreActor>>, // fairness
}
```

```rust
pub enum CoordinatorGenericMessage {
    CoordinatorIncomingStore {
        addr: Addr<StoreActor>,
    },
    CoordinatorECommerceSell {
        sell: Sell,
    },
    CoordinatorECommerceSellFailed {
        sell: Sell,
        sender: Addr<StoreActor>,
    },
    CoordinatorECommerceSellSucceeded {
        sell: Sell,
        sender: Addr<StoreActor>,
    },
    StoreDied {
        addr: Addr<StoreActor>,
    },
}
```

Actor de la Tienda y sus Mensajes

```rust
pub struct StoreActor {
    stock: Stock,
    stream: TcpStream,
    coordinator: Addr<CoordinatorActor>,
    pending_sells: Vec<Sell>,
}
```

```rust
pub enum StoreGenericMessage {
    AskSell { sell: Sell },
    ECommerceSell { status: ECommerceStatus, sell: Sell },
    SyncStock { stock: Stock },
    UpdateProduct { sell: Sell },
    ECommerceLastLineRead { last_line_read: usize },
    StoreDied,
}
```

Ejecución Inicial

```rust
let system = System::new();

let mut store_builder = StoreBuilder::new(id);

store_builder.set_default_stock();

if id == CONST_FIRST_COORDINATOR_ID {
    system.block_on(async { store_builder.build_coordinator().await.unwrap().run().await })?;
} else {
    system.block_on(async { store_builder.build_common().await.unwrap().run().await })?;
}

match system.run() {
    Ok(_v) => {
        log::info!("--- EXECUTION FINISHED ---");
    }
    Err(_e) => return Err(ECommerceError::ThreadJoinError),
};

System::current().stop();
```

¡Demo Time!