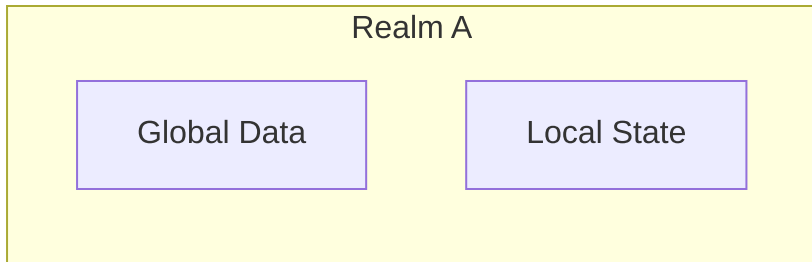# Interrealm in Gno 🌐

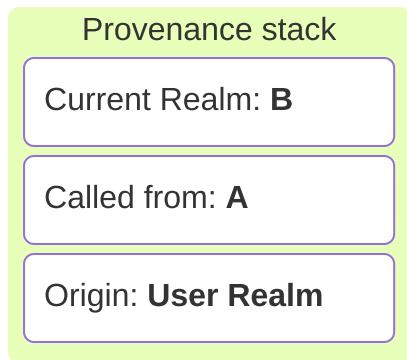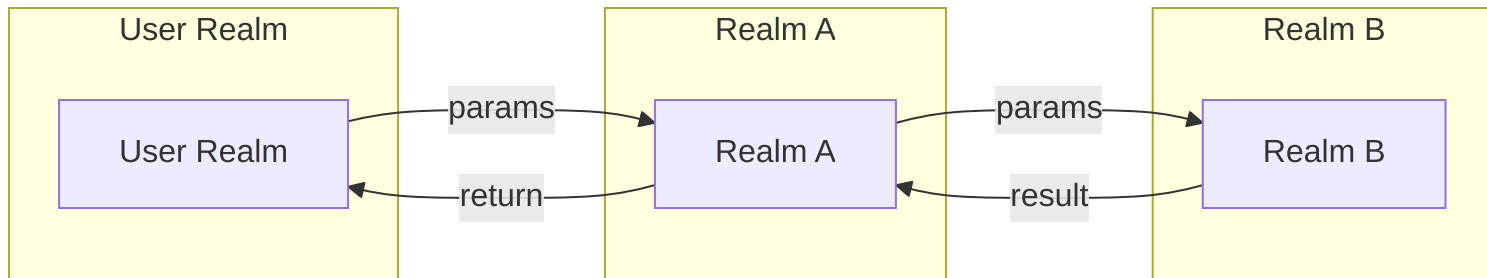A deep dive into Gno's realm system and rules

What is a Realm?

- A **realm** is an isolated execution and storage context 🌍

- Each realm has:

  - Its own **state and authority**

  - Rules for **entry** and **data modification**

- Realms enforce:

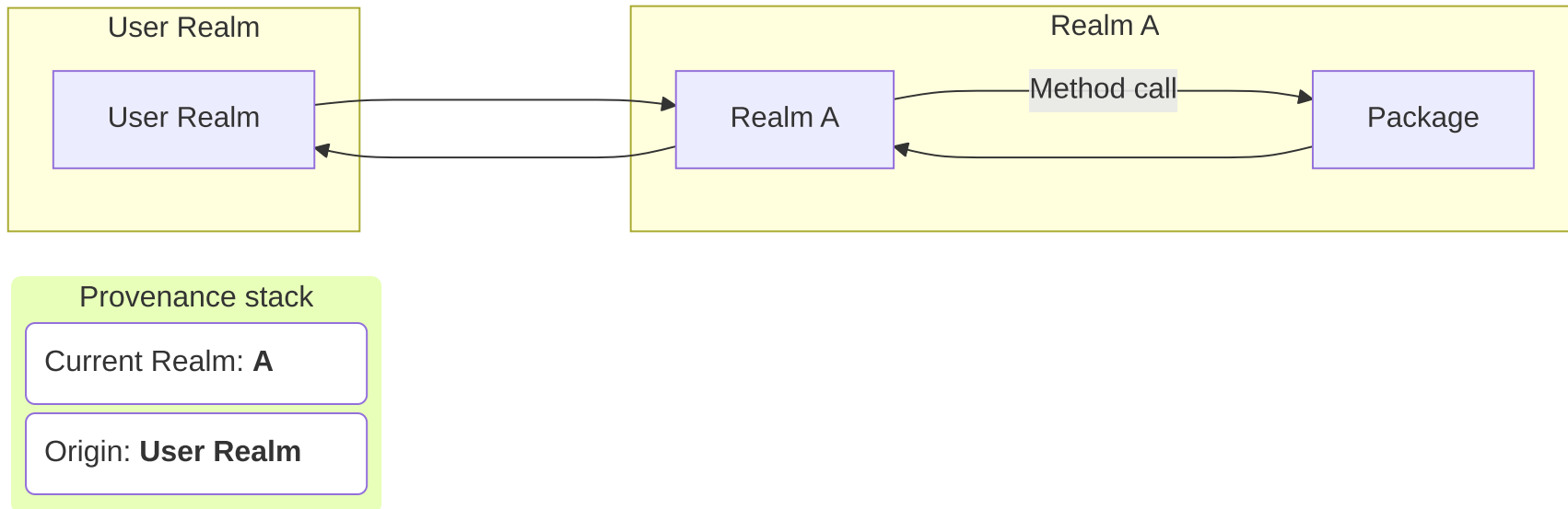 - **Security**, **Traceability** and **Isolation**

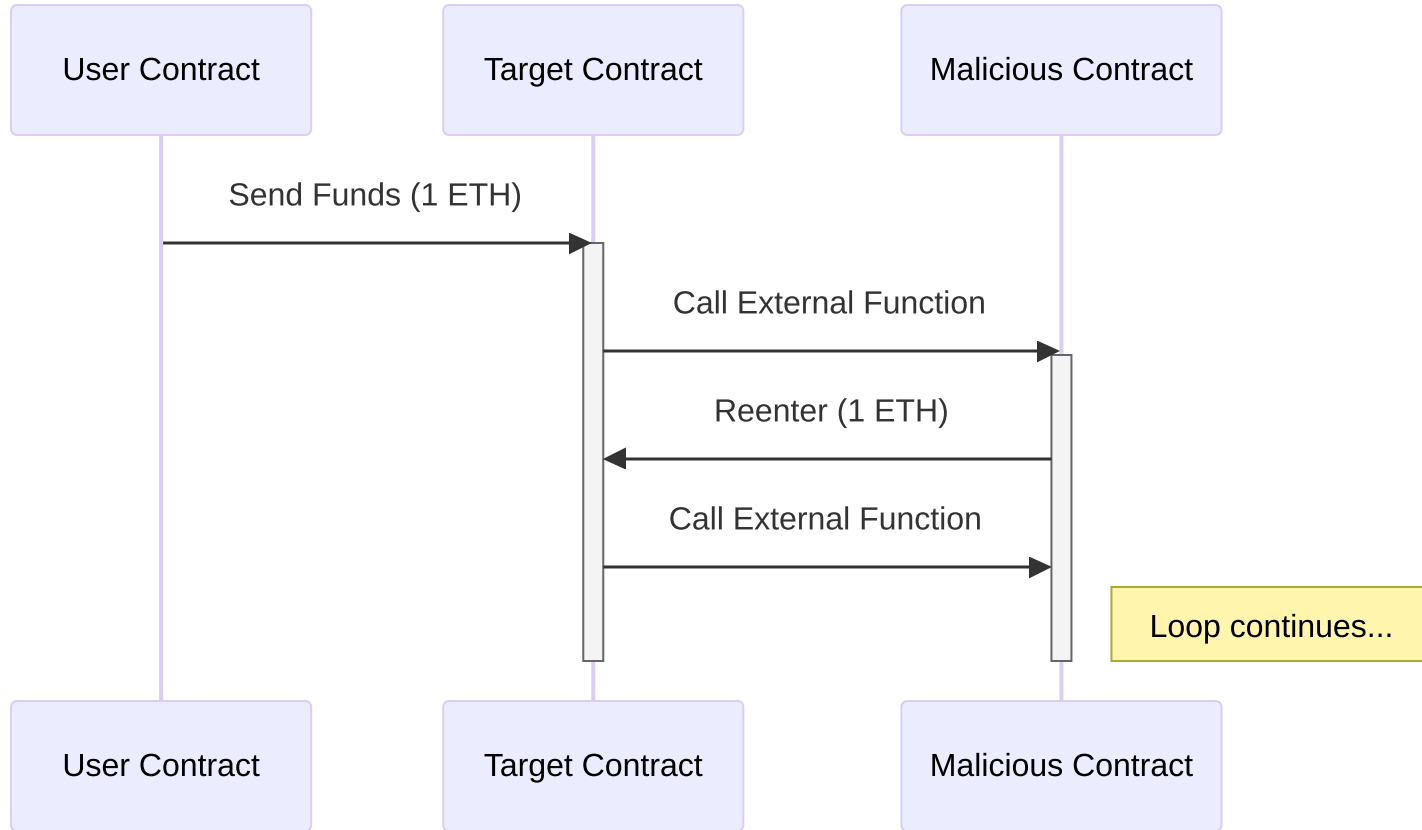# Interrealm Flow

Realm method call

Interrealm Flow

Package method call

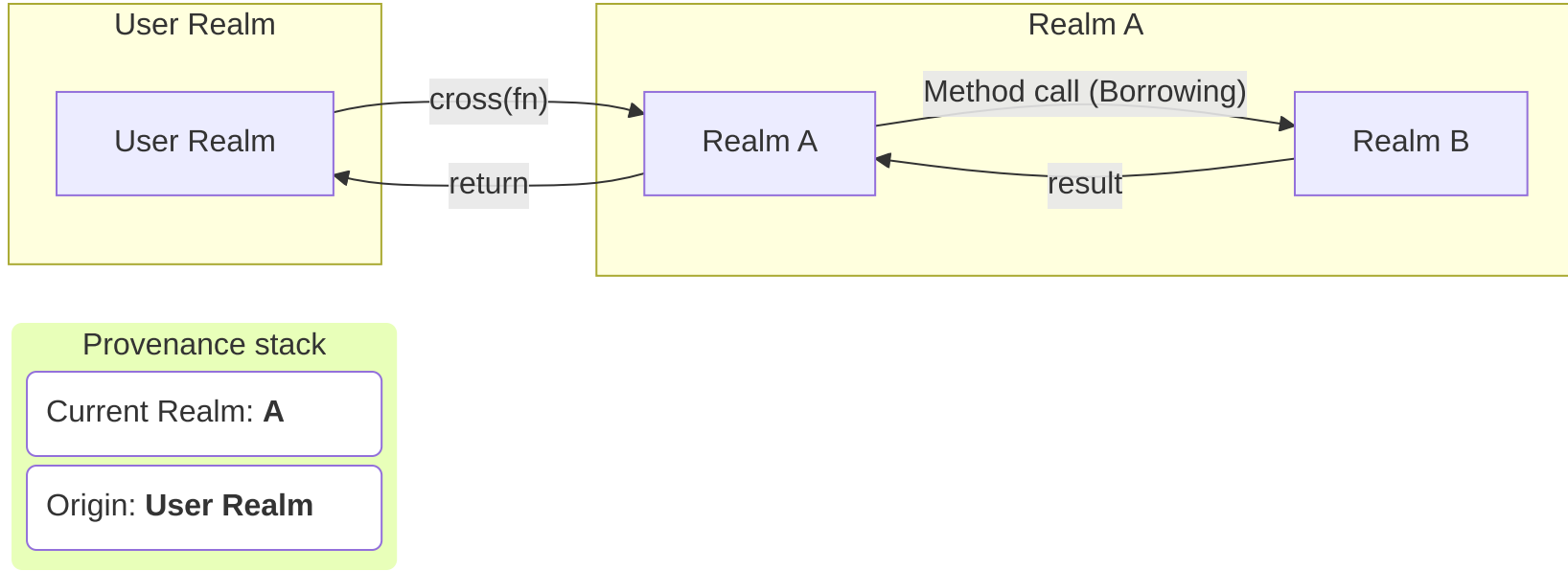# But it instaure crucial flaw - Reentrency attack

Let's introduce Borrowing

Less permissive call for a more secure environment.

- **Borrowing** = calling a method on an object in another realm.

- You **implicitly visit** the object's realm for that method call.

- You can **read and modify** the receiver and its reachable state.

- **Limit**: You cannot freely create new root-level objects.

Interrealm Flow (Borrowing)

User Realm

Realm A

User Realm

Realm A

Realm B

cross(fn)

return

Method call (Borrowing)
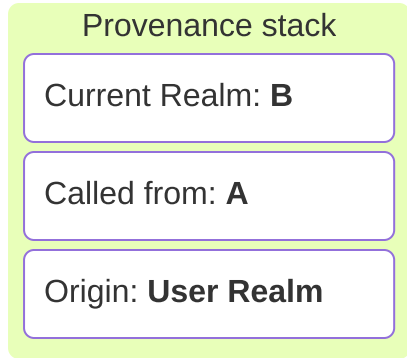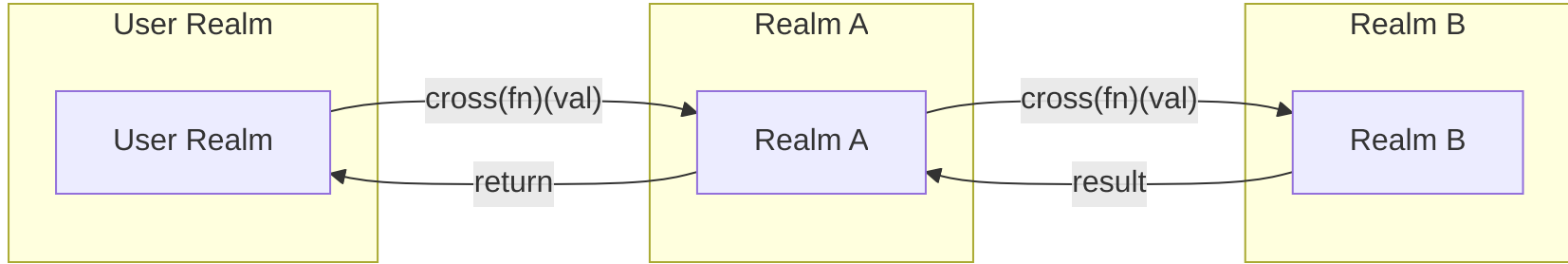
result

Provenance stack

Current Realm: **A**

Origin: **User Realm**

Crossing

- **Explicitly switch** into another realm.

- **Crossing** = `cross(fn)(...)` or functions marked with `crossing()` .

- Gain full **write access** to the realm's global storage like it used to.

- Use for **creating new objects** or performing realm-specific logic.

Interrealm Flow (Crossing)

User Realm

User Realm

cross(fn)(val)

return

Realm A

Realm A

cross(fn)(val)

result

Realm B

Realm B

Provenance stack

Current Realm: **B**

Called from: **A**

Origin: **User Realm**

Inter-realm Flow (Crossing)

Realm A

```
func CallCreatePost(title, content string) {
    cross(CreatePost)(title, content)
}
```

Realm B

```
func CreatePost(title, content string) {
    newPost := Post{
        Title: title,
        Content: content,
    }
    Posts = append(Posts, *newPost)
}
```

Rules Summary

Action	Borrowing	Crossing
Modify existing object	✅ via method call	✅
Create new unattached object	❌	✅
Implicit realm context change	No (temporary for method)	Yes (permanent inside fn)
Method syntax	`obj.Method()`	`cross(fn)(...)` + `crossing()`

Code Example: Borrowing

```go
// In realmB
func (b *Book) SetTitle(new string) {
  b.Title = new  // Allowed: borrowing b's realm
}

// In realmA
book := &realmB.Book{}
book.SetTitle("Hello Gno")
```

Code Example: Crossing

```
// In realmB
func CreateUser(name string) {
  crossing()
  user := &User{Name: name}
  users[name] = user  // Global storage in realmB
}

// In realmA
cross(realmB.CreateUser)("alice")
```

Best Practices

1. Default to **non-crossing** methods.

2. Use **borrowing** for object-specific ops.

3. Reserve **crossing** for realm-level state changes.

4. Always mark public crossable functions with `crossing()` .

Summary

- **Realms** = isolated worlds.

- **Borrowing** = temporary, object-scoped access.

- **Crossing** = explicit realm switch for full access.

- Choose the right approach for **security** and **clarity**.