Theseus: a State Spill-free Operating System

Kevin Boos and Lin Zhong

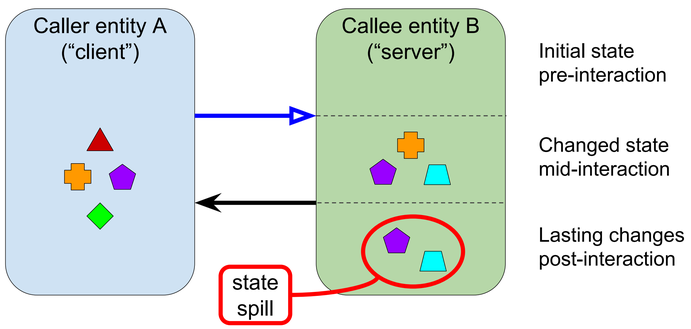
OSes are complex and entangled

- Existing OSes are a web of entangled entities
  - Cannot treat entities independently
- OS components should be easily interchangeable at runtime, for fluid system evolution
  - Goal: **Runtime Composability**
- Prior decoupling strategies are insufficient; entanglement remains between system entities
  - Modularization
  - Encapsulation (OOP)
  - Privilege-level separation (µkernel)
  - Hardware-driven (Barrelfish)

For disentanglement, we focus *only* on **states** and how they propagate throughout entities in the system.

State Spill is the root cause

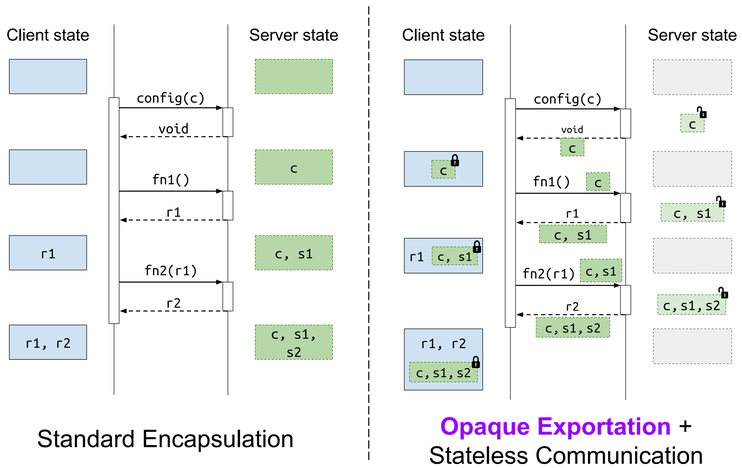
- Scenario: source entity A ("client" role) communicates with destination B ("server role).
- State spill occurs when B's state undergoes a **lasting change** after handling an interaction from A.



Main goals of Theseus

Directive 1: no state spill (above all else)
Directive 2: elementary modules

Encapsulation causes state spill



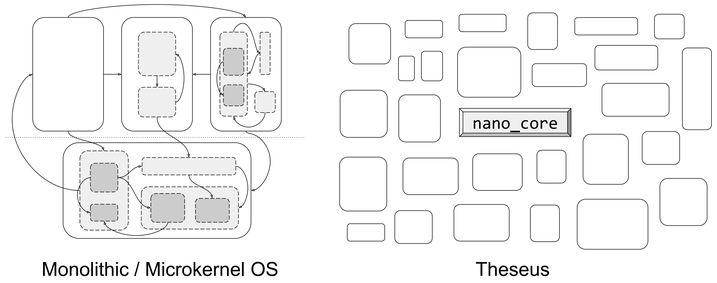
Theseus design principles

1. **No traditional encapsulation**
   - Client A should maintain the state representing its progress with server B, instead of B
   - Preserve information hiding: A cannot inspect or modify state from B
2. **Stateless interactions**
   - An interaction from A → B must include everything B needs to handle it
   - Implication: B can be practically stateless
3. **Universal, connectionless communication**
   - All entities are accessible in a uniform way
   - Do not assume ongoing existence of interfaces
4. **Re-use of generic, spill-free patterns**
   - Implement common OS design patterns once in a spill-free way, then re-use across system

Design & implementation decisions

Flat Module Architecture

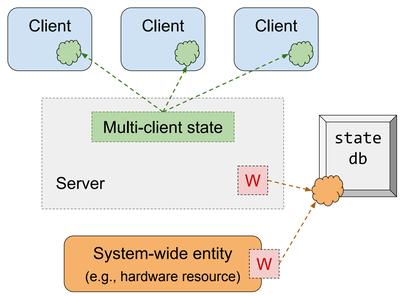
- Submodules contribute to complex entanglement
  - Extract submodules into first-order modules



- Simplifies module logic → `nano_core` manages all
- Permits *communication* and *compositional hierarchy*

State Management

- At *some* point, some entities must hold some state



- Export multi-client state as data blob jointly owned by all clients
- Clientless states are owned by `state_db` metamodule
  - Entity caches a weak reference to it

Software-only Isolation and Safety

- Modules are separate binaries: namespace isolation
- Augment Rust compiler to permit minimal subset of unsafe code necessary for basic OS functionality
- Error handling is mandatory, using `Option` & `Result`
  - Panics are disallowed and transformed into errors

Current status and future work

- Done: baseline OS from scratch, all in Rust
- Now: analyze & rethink modules and interfaces to remove state spill
- Far: no user/kernel distinction: "bag of modules"

RICE Unconventional Wisdom

NSF