JANUARY 2026 SERIES

FROM GO BUILD
TO GO RUN

GOLANG 2026 - NIV RAVE

#18

INTERFACES - BASED MOCKING & DESIGN

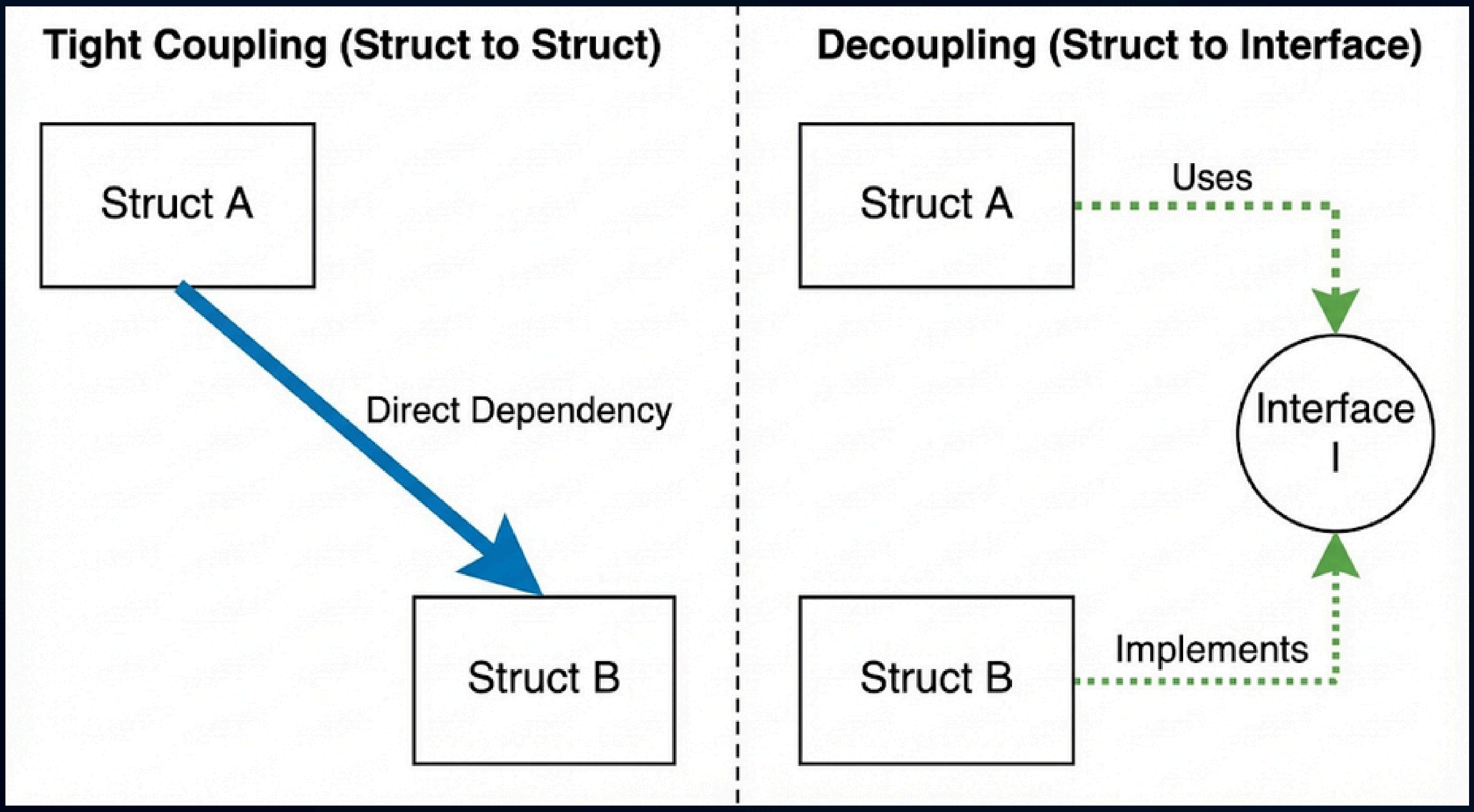
DECOUPLING FOR TESTABILITY

The Dependency Problem

Why Concrete Types Kill Testing

If your business logic depends on a concrete **sql.DB* or **aws.S3Client*, you cannot test it without a database or a cloud connection.

The Senior Fix: We define an interface that represents only the behavior our logic needs from that dependency.



Consumer-Defined Interfaces

Define it where you use it

In many languages, interfaces belong to the provider. In Go, the Consumer defines the interface.

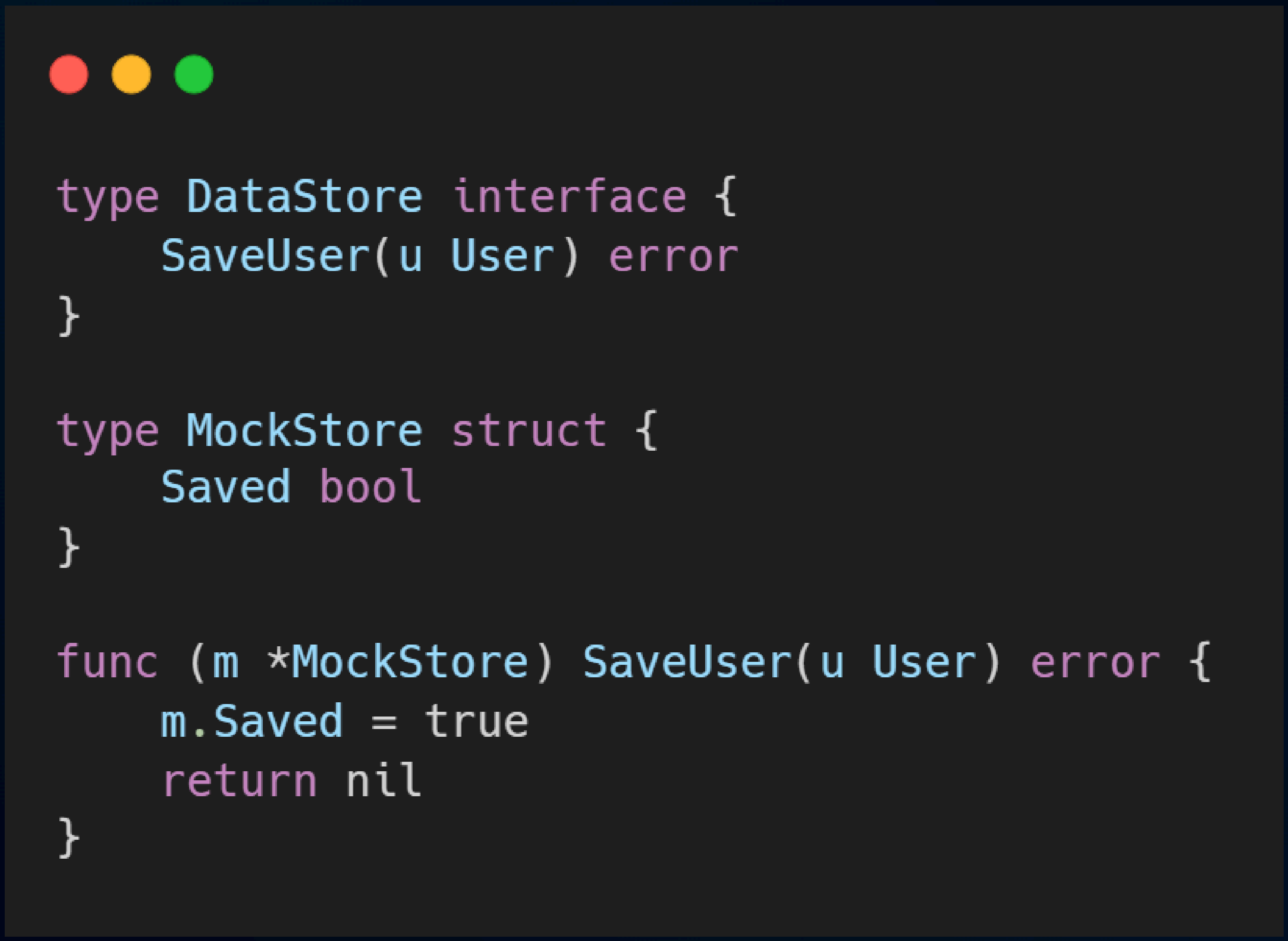
```go
// The "Logic" package defines what it needs
type DataStore interface {
    SaveUser(u User) error
}

type UserService struct {
    store DataStore // The dependency is the interface
}
```

This keeps the *UserService* totally unaware of the database implementation (Postgres, Mock, etc.).

Hand-written Mocks (The Go Way)

You don't always need a Mocking Library

Because interfaces are implicit, writing a mock is just creating a struct that matches the signature.



```go
type DataStore interface {
    SaveUser(u User) error
}

type MockStore struct {
    Saved bool
}

func (m *MockStore) SaveUser(u User) error {
    m.Saved = true
    return nil
}
```

The Benefit: No complex DSLs or code generation needed for simple tests. It’s readable, standard Go code.

Testing the "Error Path"

Simulating Failure with Ease

The hardest things to test are network failures and DB timeouts. Interfaces make this trivial.

```go
type DataStore interface {
    SaveUser(u User) error
}

type FailStore struct{}

func (f FailStore) SaveUser(u User) error {
    return errors.New("database connection lost")
}
```

By passing *FailStore* into your service, you can verify exactly how your system handles a crash without actually crashing a server.

The Mocking Trap

Don't Mock Your Own Logic

A common pitfall is mocking every internal component. If you mock your own business logic, your tests stop checking behavior and start checking implementation. If you change the code, the test breaks, even if the result is still correct.

The Strategy:

Mock the Boundaries: Only use interfaces to mock things you don't control or things that are slow/nondeterministic (External APIs, Databases, Clock/Time, Randomness).

Verify the Core: For internal logic, use the concrete structs. It's faster, safer, and ensures your tests actually validate that the code works.

Pro tips:

Brittle Tests: If your test setup requires 50 lines of *mock.On("Something").Return(...)*, your architecture is too fragmented.

The "I/O" Rule: If a function doesn't perform I/O (disk, network, etc.), it probably doesn't need an interface for testing. Use the real thing.

Confidence vs. Coverage: High test coverage with mocks is a false sense of security. Integration tests with real (or faked) data are where true confidence is built.

```go
// THE TRAP: Mocking an internal calculator
// You are now testing the MOCK, not the MATH.
type MathService interface { Add(a, b int) int }
```

Interface Embedding for Mocks

Mocking Large Interfaces

If you need to mock a large interface but only care about one method, use Anonymous Embedding.

```go
type MockReader struct {
    io.Reader // Embed the interface
}

// Only override what you need
func (m MockReader) Read(p []byte) (n int, err error) {
    return 0, io.EOF
}
```

This allows your mock to "satisfy" the large interface without you having to manually implement every single method.

The "Mock" as Documentation

Design as a Contract

When you design with interfaces and mocks, your tests become the living documentation of how your components interact.

Your code becomes a set of interchangeable LEGO blocks rather than a single, monolithic block of marble.

Recap:

- Consumers should define the interfaces they need.
- Use mocks to test error paths and boundaries.
- Interfaces are your primary tool for Dependency Injection.

Do you prefer using a mocking library like *gomock* or *testify*, or do you write your mocks by hand? Let's debate! 👇