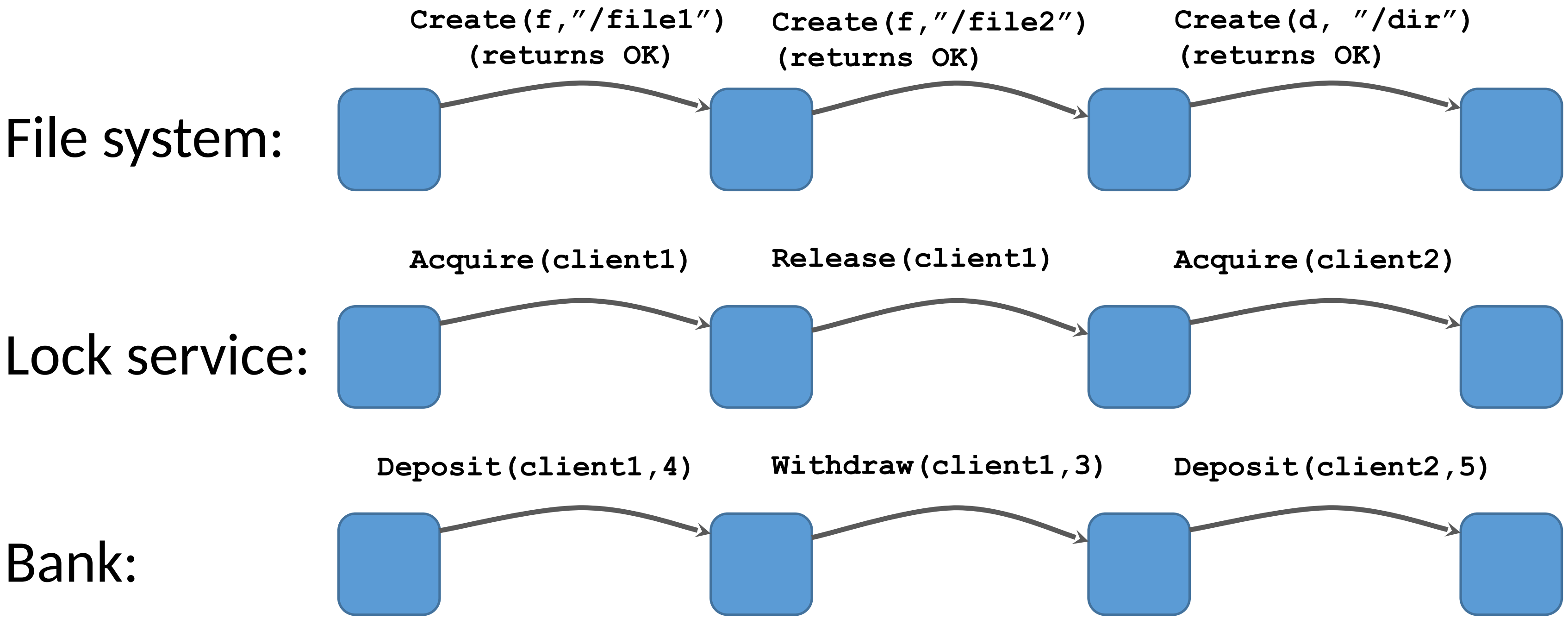
EECS498-003 Formal Verification of Systems Software

Material and slides created by

Jon Howell and Manos Kapritsos

Events define correctness

One should be able to evaluate the correctness of the system by inspecting a behavior (sequence) consisting of world-visible events



File system:

`Create(f,”/file1”) (returns OK)` `Create(f,”/file2”) (returns OK)` `Create(d, ”/dir”) (returns OK)`

Lock service:

`Acquire(client1)` `Release(client1)` `Acquire(client2)`

Bank:

`Deposit(client1,4)` `Withdraw(client1,3)` `Deposit(client2,5)`

A refinement proof

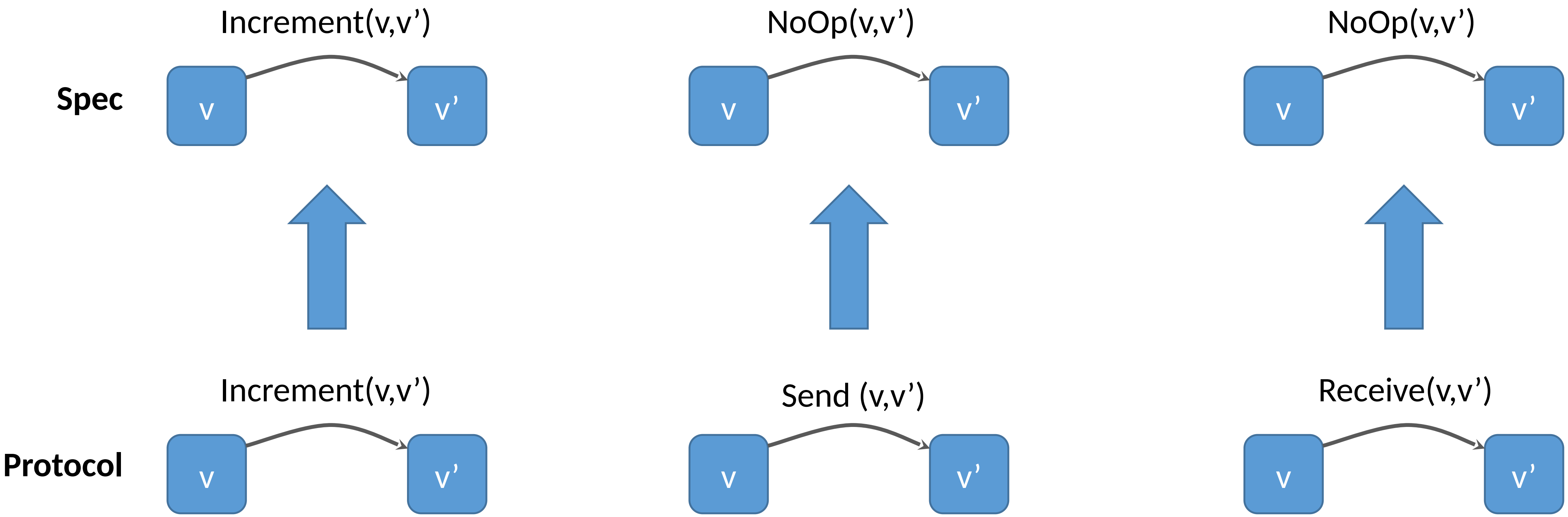
```
ghost function Abstraction(v:Variables) : Spec.Variables
predicate Inv(v:Variables)

lemma RefinementInit(v:Variables)
    requires Init(v)

    ensures Spec.Init(Abstraction(v))  // Refinement base case

lemma RefinementNext(v:Variables, v':Variables)
    requires Next(v, v’, evt)


    ensures Spec.Next(Abstraction(v), Abstraction(v’), evt) // Refinement
inductive step
        || Abstraction(v) == Abstraction(v’) && evt == NoOp // OR stutter step
```

Case study: a moving counter

- Hosts pass a counter around
- They can increment it or send it to someone else
  - Three types of protocol steps: Increment, Send, Receive
- No duplicates in the network

- Spec: a counter

Case study: a moving counter



VSCode transition