

# **EECS498-008**

# **Formal Verification**

# **of Systems Software**

Material and slides created by  
Jon Howell and Manos Kapritsos

# A distributed system is composed of multiple hosts, a **network** and **clocks**

Distributed system: attempt #3

```
module DistributedSystem {
```

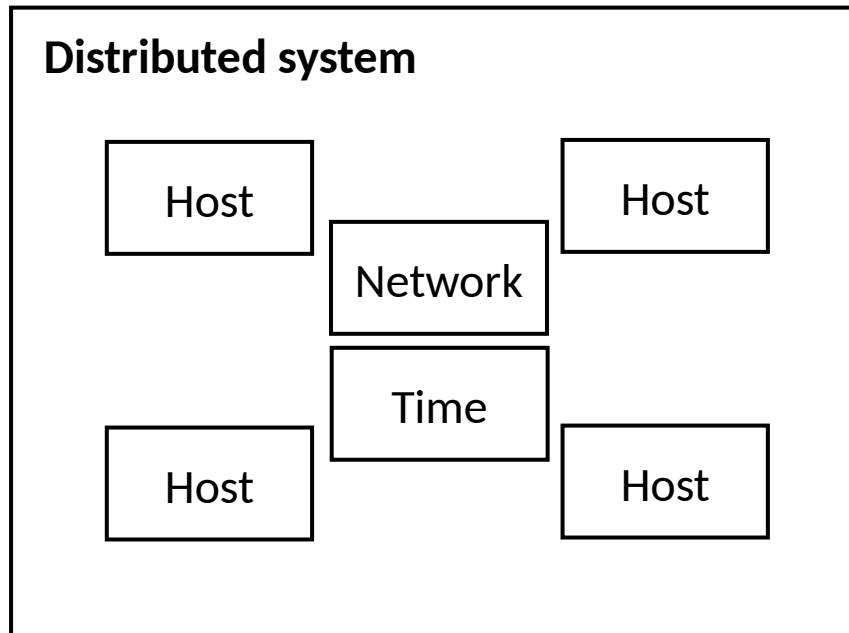
```
  datatype Variables =
```

```
    Variables(hosts:seq<Host.Variables>,
              network: Network.Variables,
              time: Time.Variables)
```

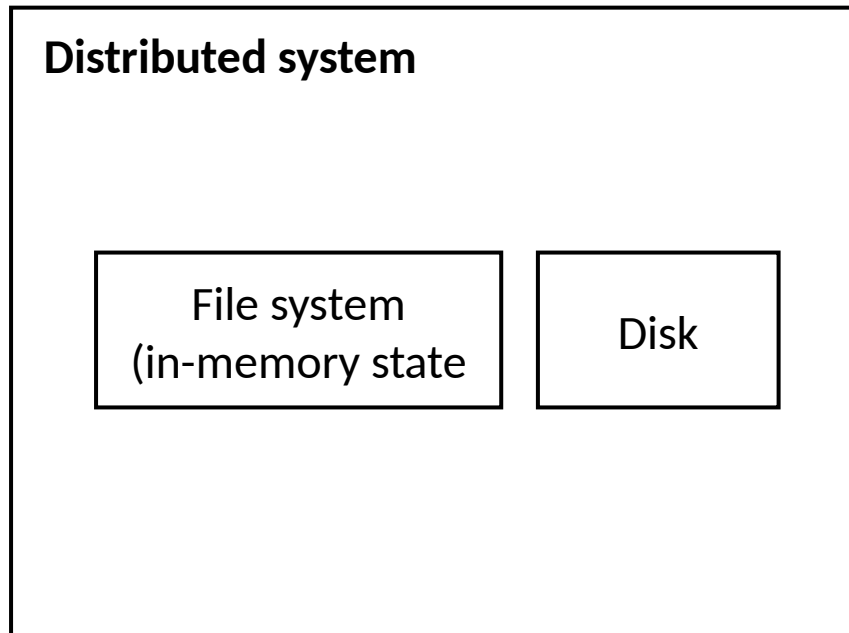
```
  predicate Next(v, v', hostid, msgOps: MessageOps, clk:Time) {
    || (&& HostAction(v, v', hostid, msgOps)
       && Network.Next(v, v', msgOps)
       && Time.Read(v.time, clk))
    || (&& Time.Advance(v.time, v'.time)
       && v'.hosts == v.hosts
       && v'.network == v.network)
```

```
  }
```

```
}
```



# A “distributed” system



```

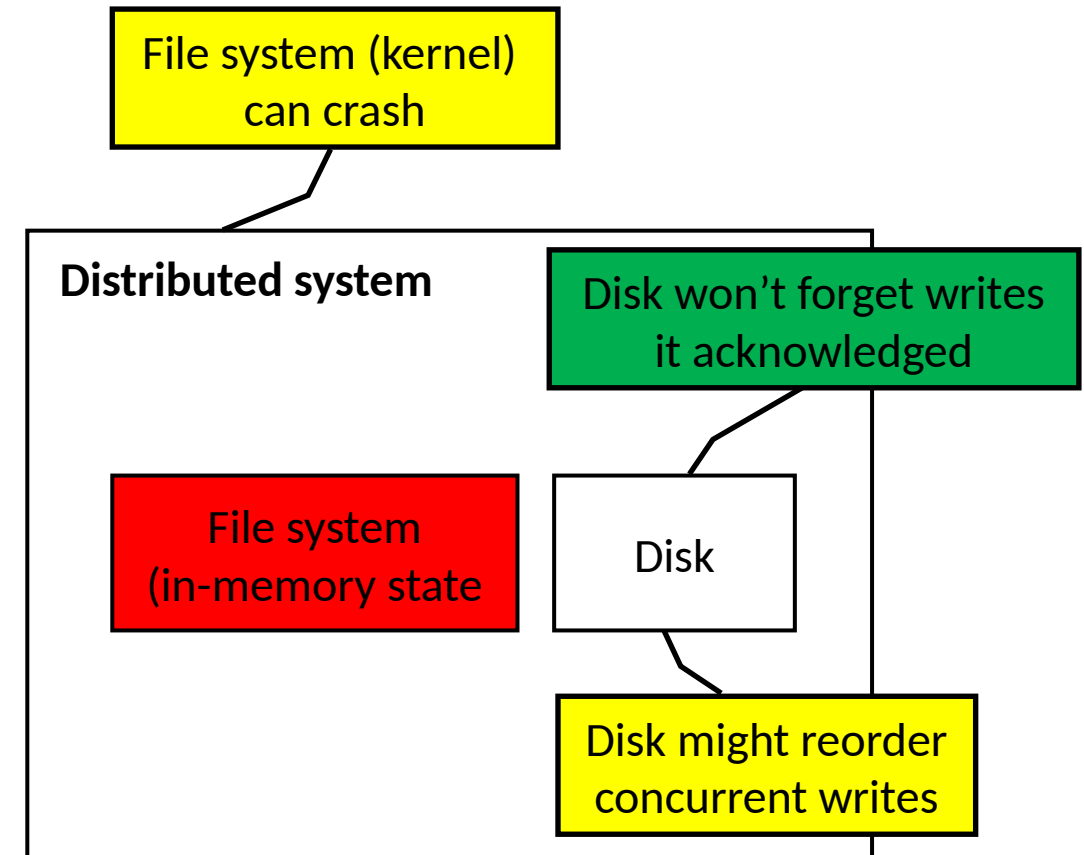
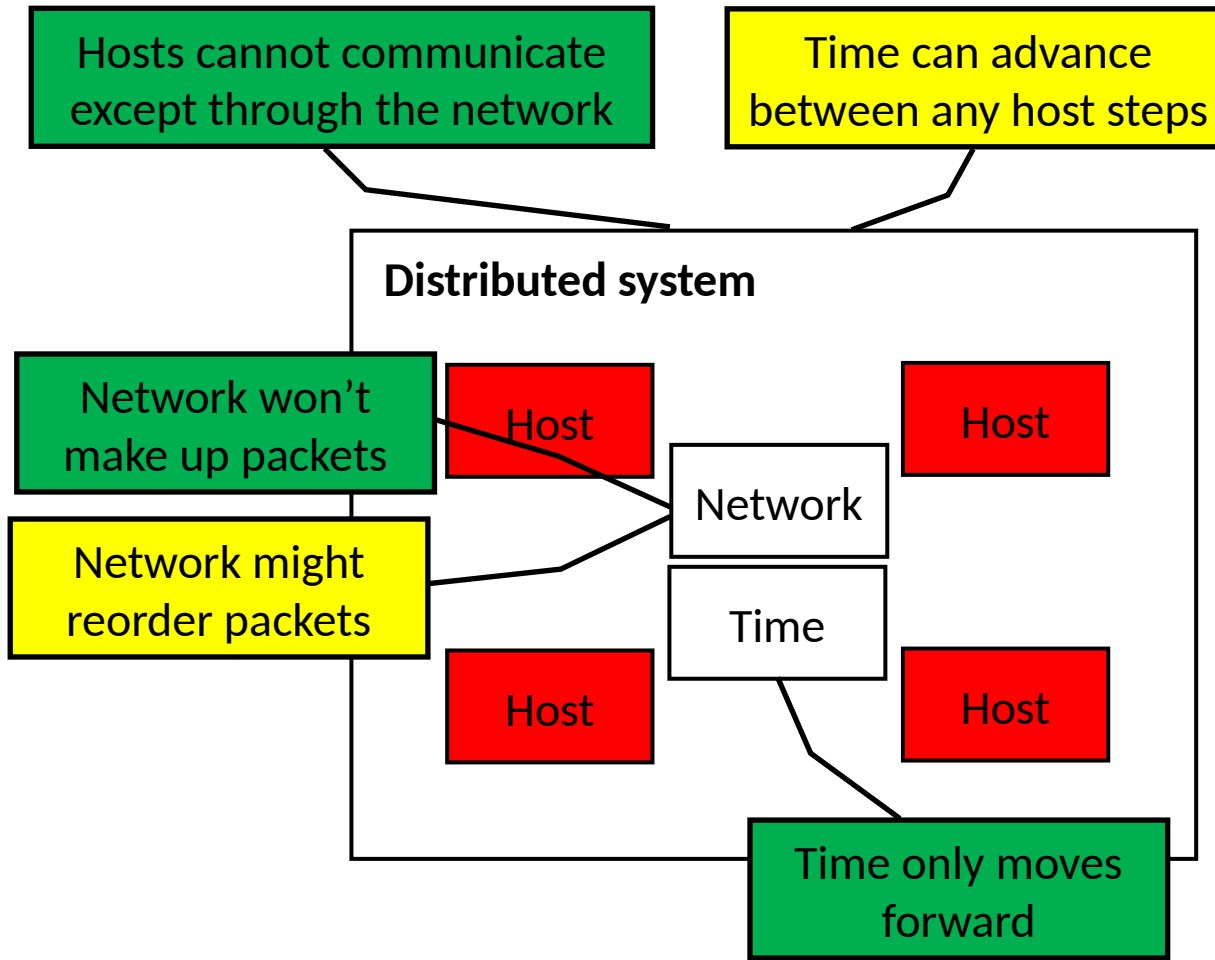
module DistributedSystem {
  datatype Variables =
    Variables(fs: FileSystem.Variables,
              disk: Disk.Variables)

  predicate Next(v, v') {
    || (exists io ::
        && FileSystem.Next(v.fs, v'.fs, io)
        && Disk.Next(v.disk, v'.disk, io)
    || ( // Crash!
        && FileSystem.Init(v'.fs)
        && v'.disk == v.disk
    )
  }
}

```

Binding variable

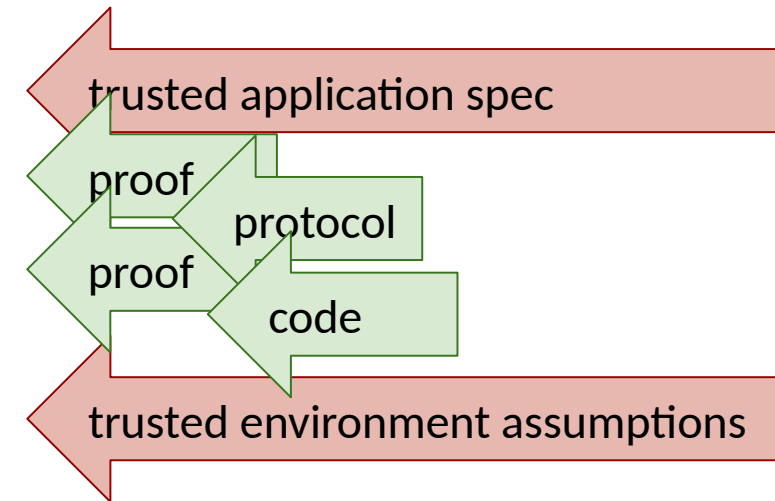
# Trusted vs proven



# **SPECIFICATION** : the systems specification sandwich



image: pixabay

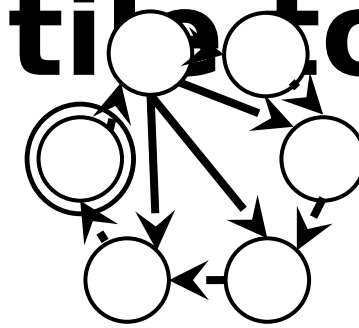


# Administrivia

- Midterm went great, congratulations!
- Problem set 3 due on Friday
- Project 1 will be released on Monday
  - Let me know if you can't find teammates
- Midterm evaluations due today!
  - They are **really, really important**
  - If we reach 66%, I'll design custom course stickers
- I'm giving another lecture right after this one, so I have to skip IOH

# Chapter 6: Refinement

# State machines: a versatile tool



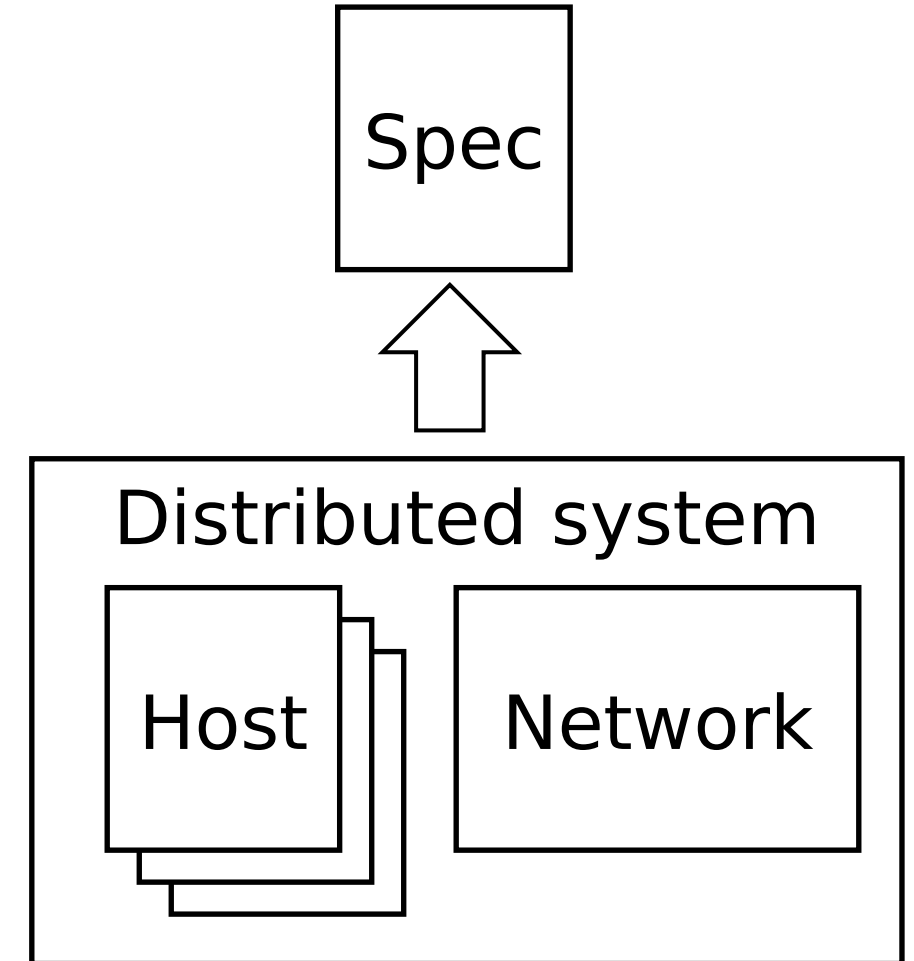
State machines can be used to

- Model the program
- Model environment components
- Model how the system (program+environment) fits together
- Specify the system behavior



# Different ways to specify behavior

- C-style assertions
- Postconditions
- Properties/invariants
- Refinement to a state machine



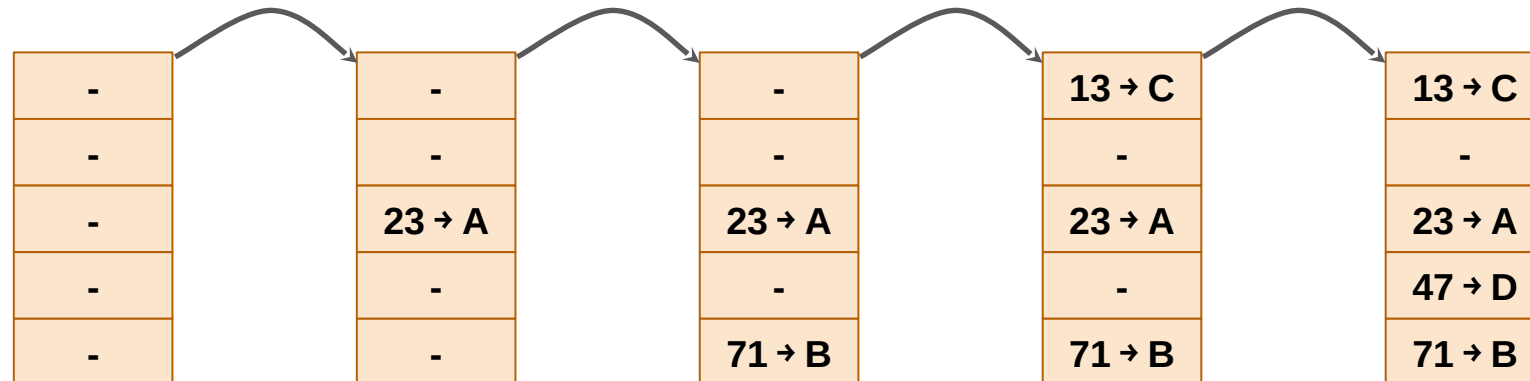
# Example: hashtable

```

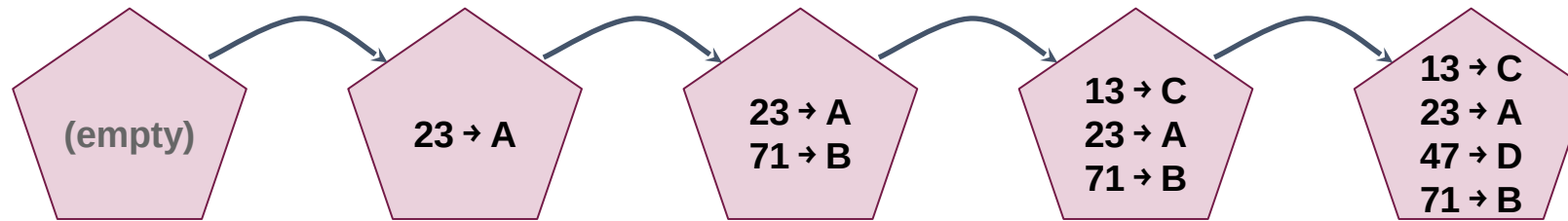
module HashTable {
  datatype Variables = Variables(tbl:seq<Pair<int, string>>)

  predicate Insert(v:Variables, v':Variables, key:int,
val:string) {
    var free := Probe(v.tbl, key);
    && free.Some?
    && v'.tbl == v.tbl[free.value := Pair(key, val)]
  }
}

```

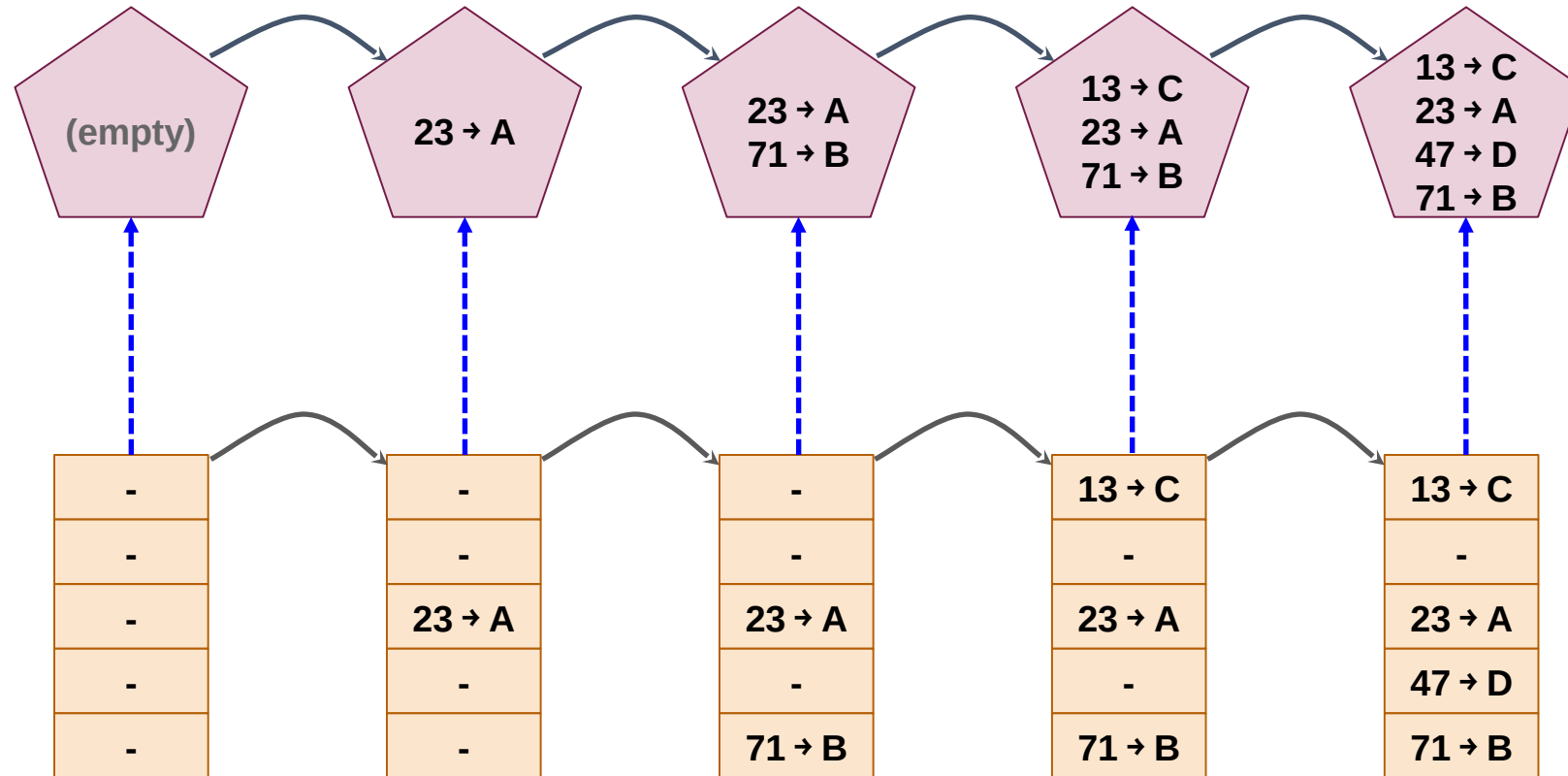


# The spec: a simple map



```
module MapSpec {  
  datatype Variables = Variables(mapp:map<Key, Value>)  
  
  predicate InsertOp(v:Variables, v':Variables, key:Key,  
value:Value) {  
    && v'.mapp == v.mapp[key := value]  
  }  
}
```

# Refinement



# The benefits of refinement

Refinement allows for good specs

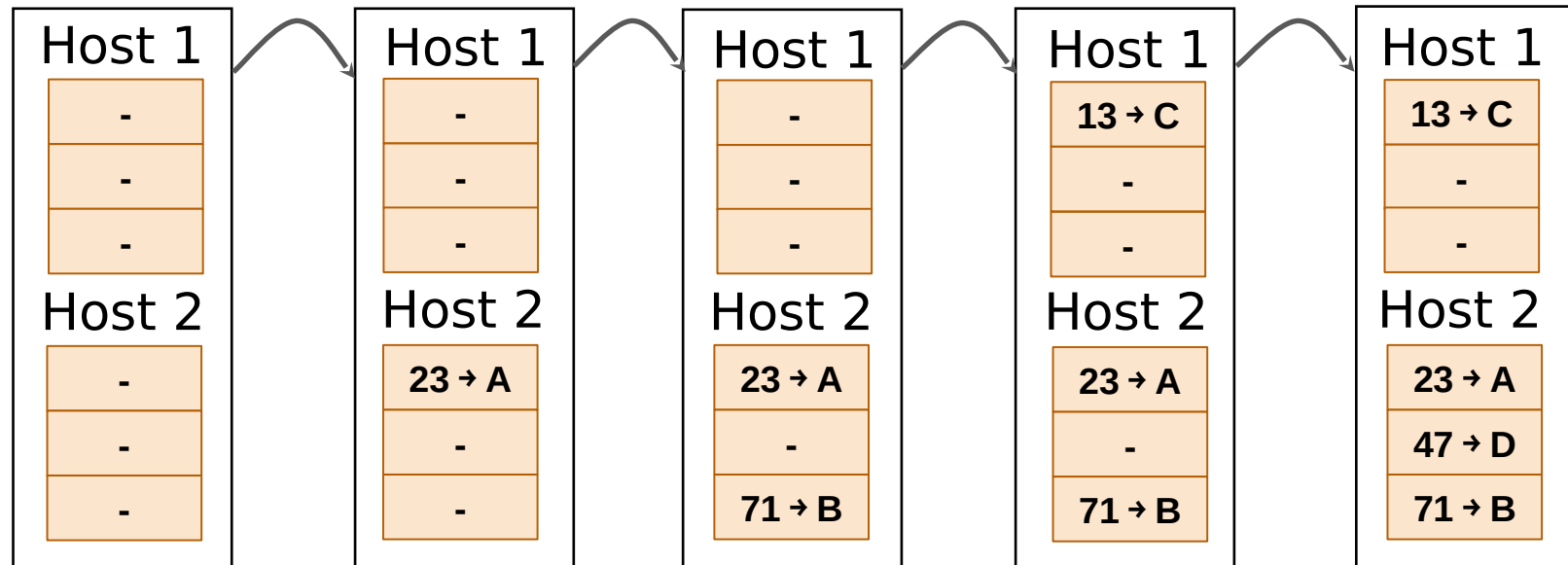
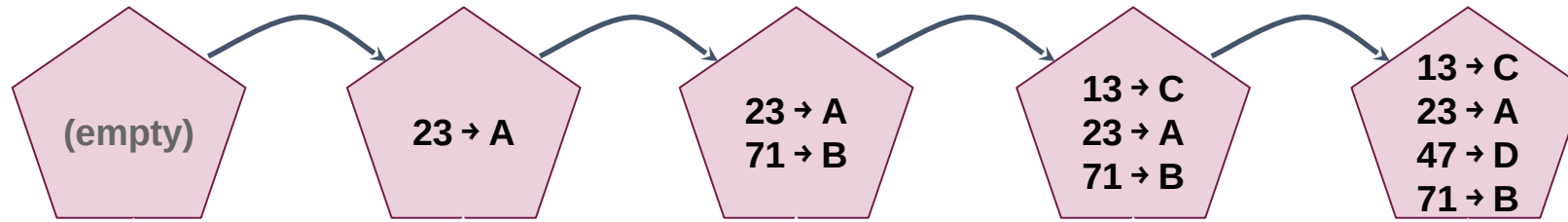
- Abstract: elide implementation details
- Concise: simple state machine
- Complete: better than a “bag of properties”
  - But if you want, you can prove properties about the spec

Refinement is very powerful

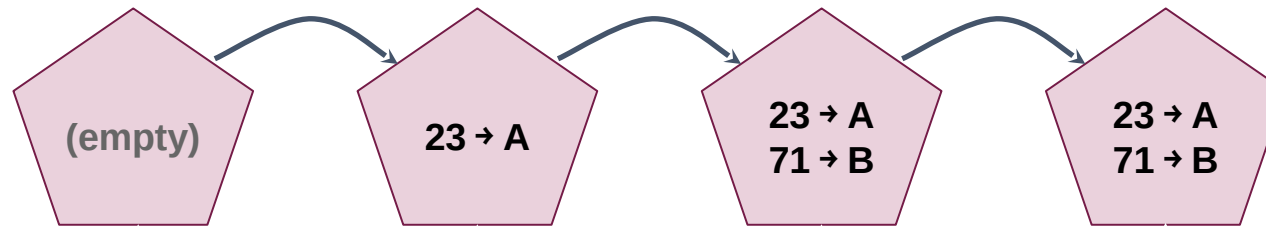
- Can specify systems that are hard to specify otherwise
  - E.g. linearizability

# A sharded key-value store

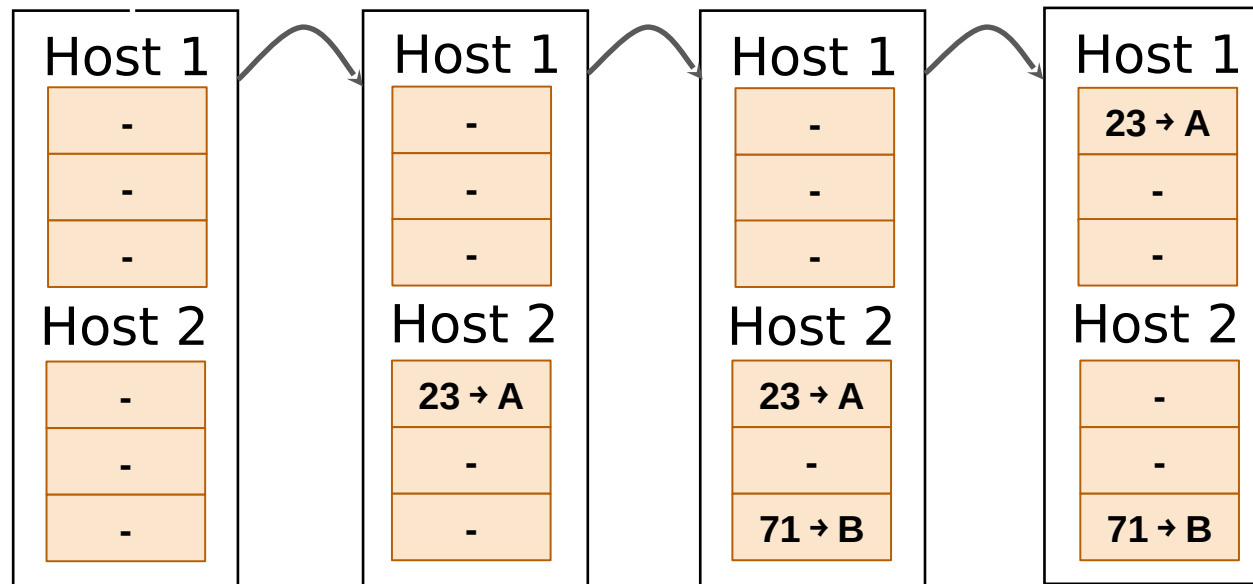
Logically centralized, physically distributed



# Stutter steps



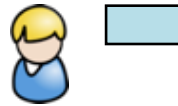
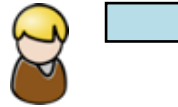
One  
“stutter”  
step for the  
spec



One normal  
step for the  
implementati  
on

# A primary-backup protocol

Clients

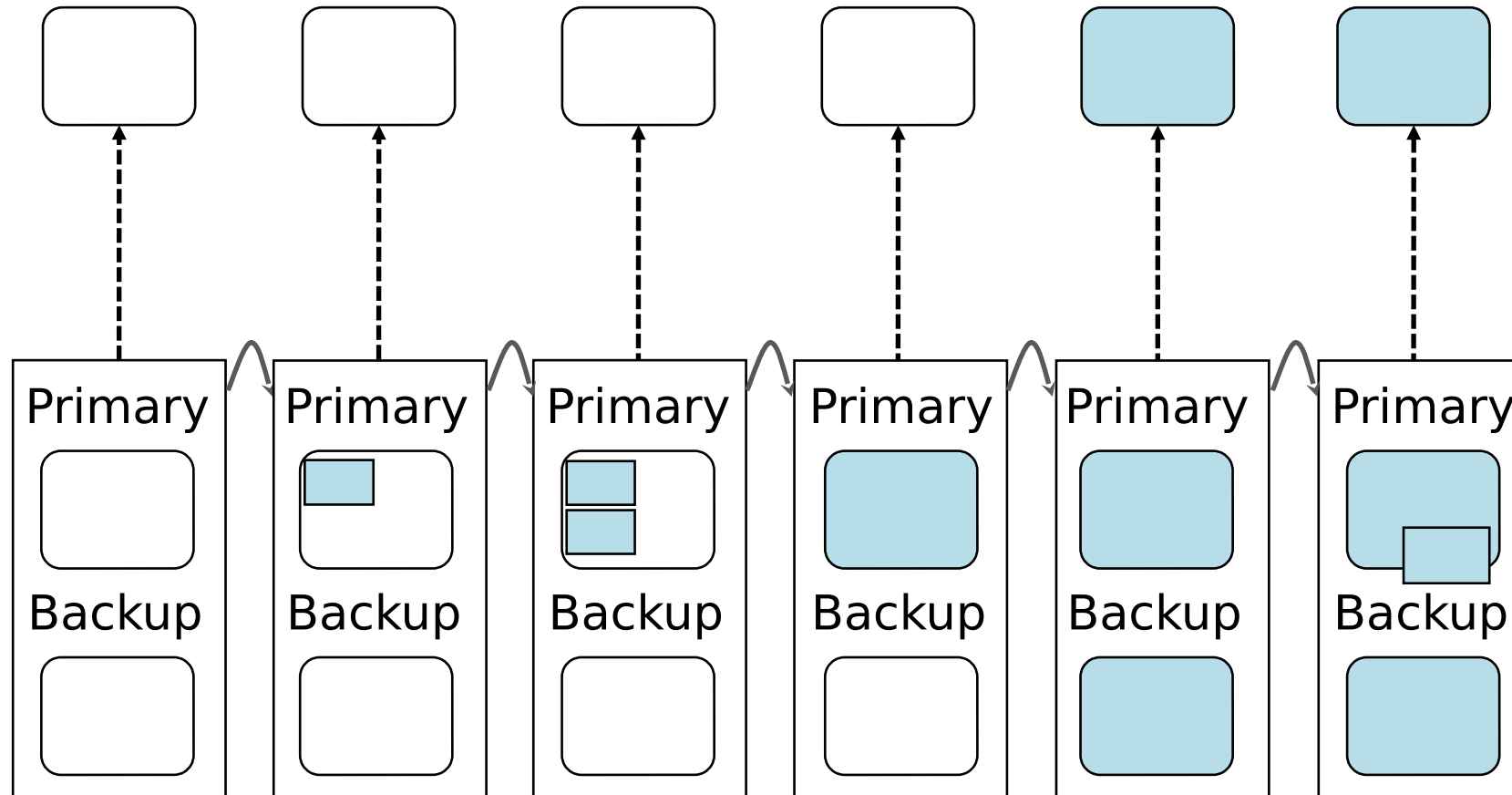


Primary

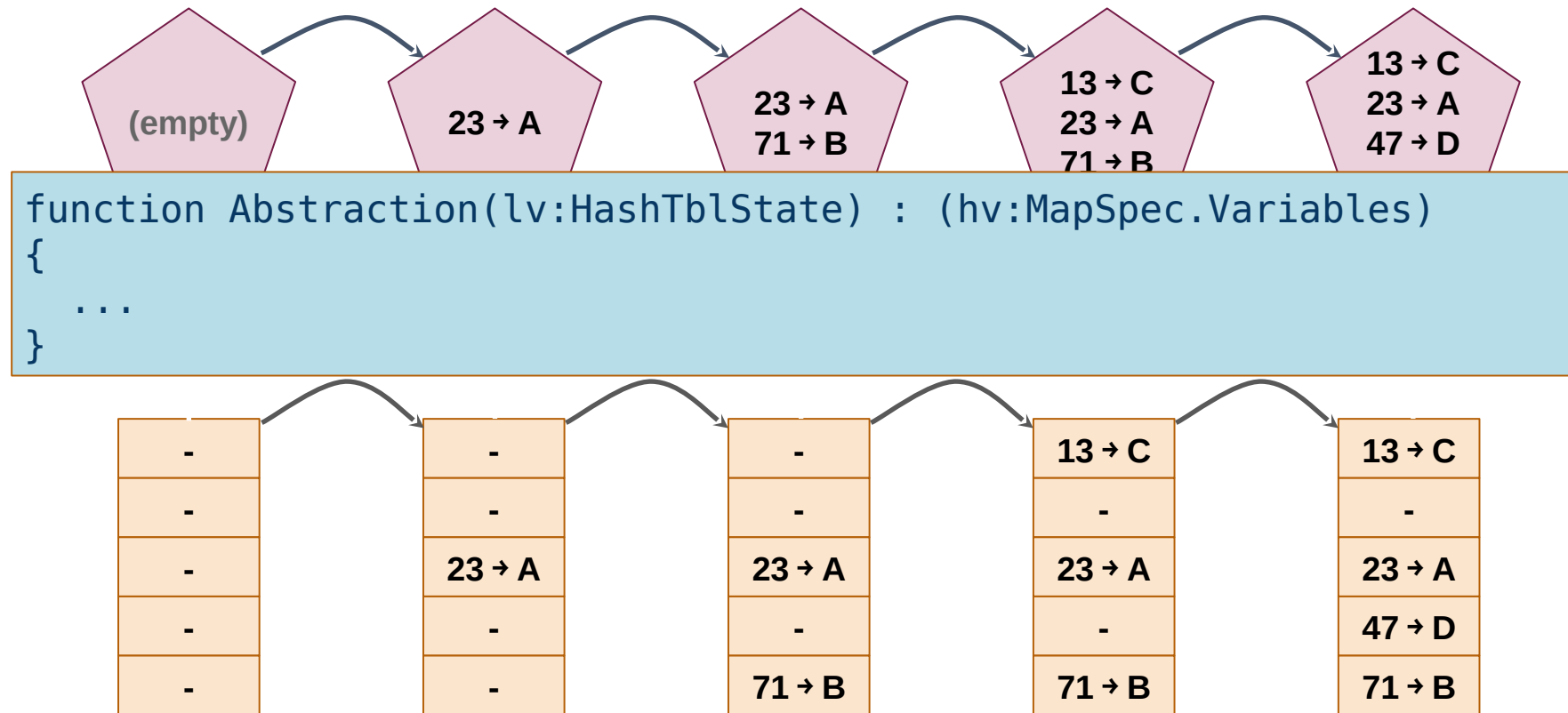
Backup



# A primary-backup protocol



# The interpretation (Abstraction) function



# A refinement proof

```
function Abstraction(v:Variables) : MapSpec.Variables
predicate Inv(v:Variables)

lemma RefinementInit(v:Variables)
  requires Init(v)
  ensures Inv(v) // Inv base case
  ensures MapSpec.Init(Abstraction(v)) // Refinement base case

lemma RefinementNext(v:Variables, v':Variables)
  requires Next(v, v')
  requires Inv(v)
  ensures Inv(v') // Inv inductive step
  ensures MapSpec.Next(Abstraction(v), Abstraction(v')) // Refinement inductive
step
  || Abstraction(v) == Abstraction(v') // OR stutter
step
```