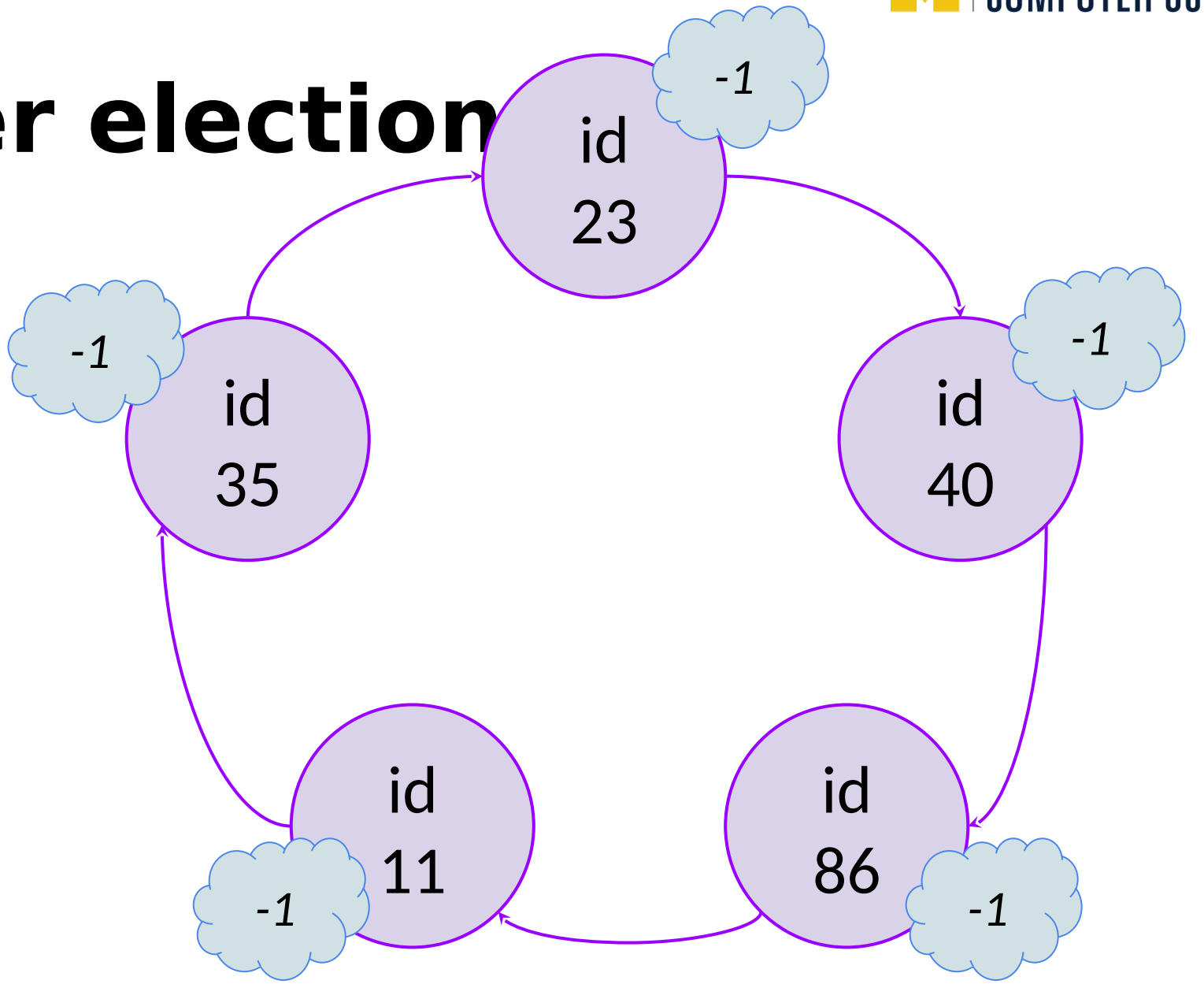
EECS498-008
Formal Verification of Systems Software

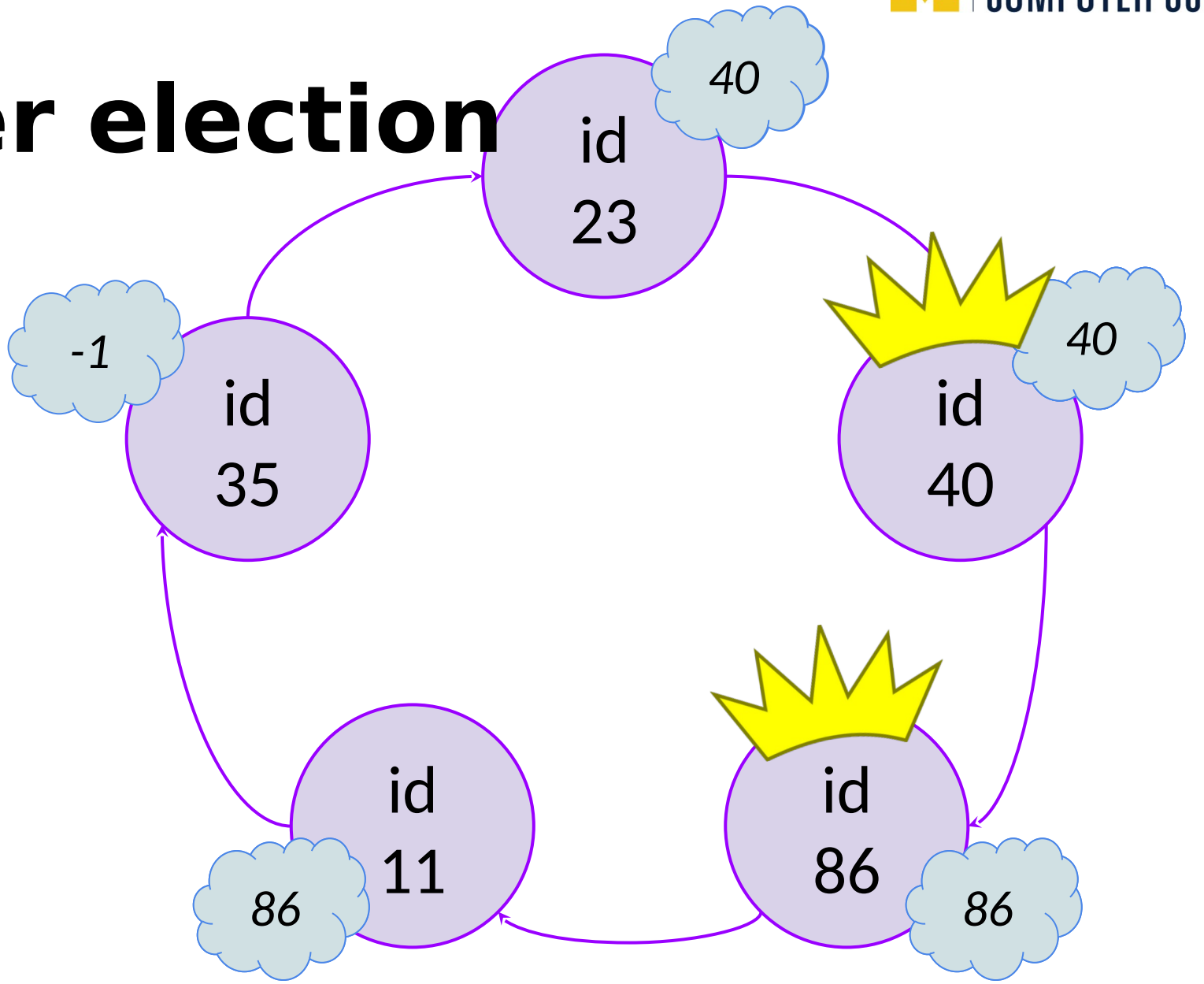
Material and slides created by

Jon Howell and Manos Kapritsos

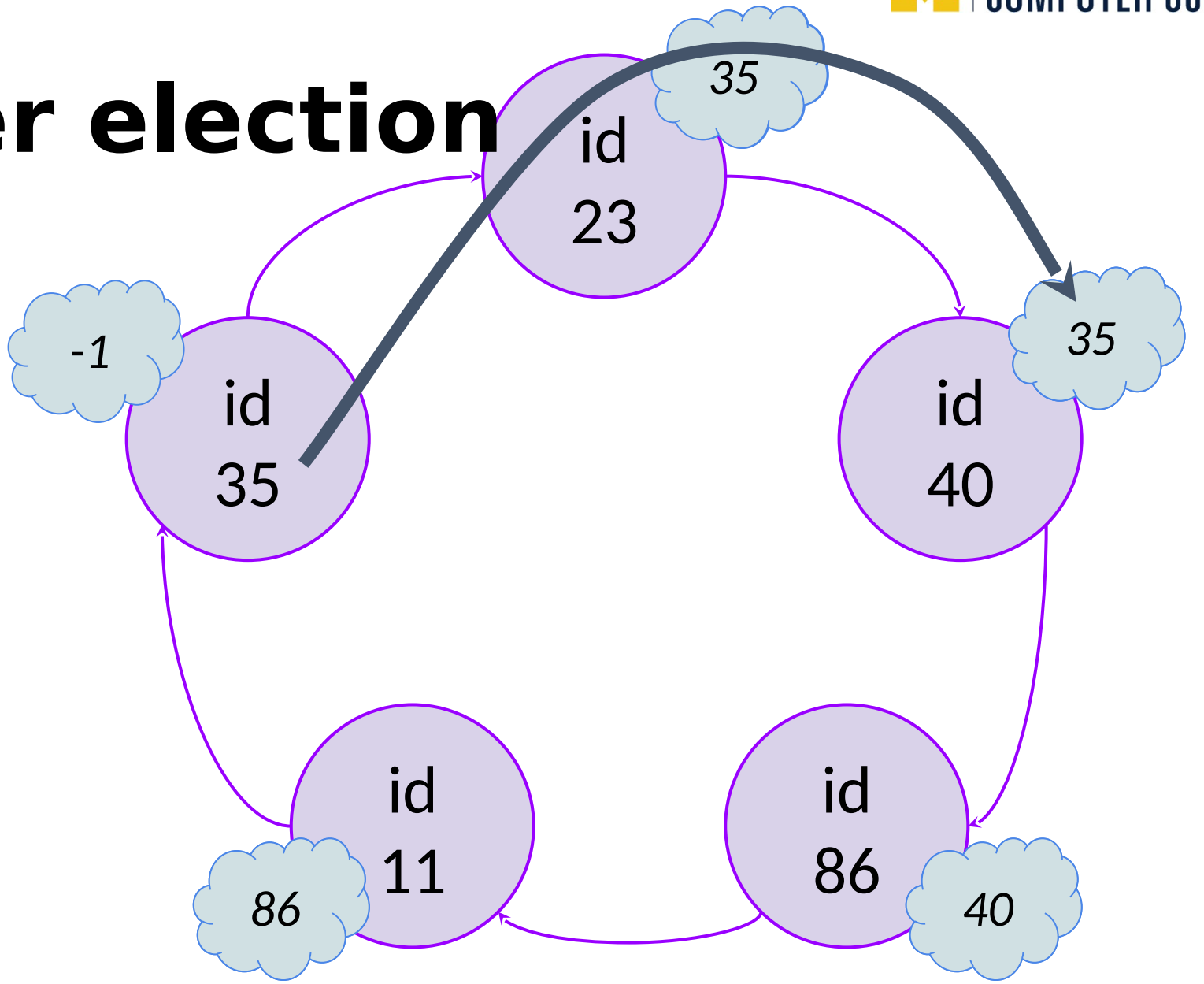
Leader election



Leader election



Leader election



Administrivia

- Midterm exam next Wednesday, 10/12
  - 6-8pm, EECS1303
  - No lecture that day
- Closed books
  - Allowed one double-sided cheat-sheet, 10pt minimum
- Covers everything up to Chapter 4 (i.e. excluding distributed systems)

- Problem set 3 (Chapter 5) will be released on Monday, 10/10

Introduction to distributed systems

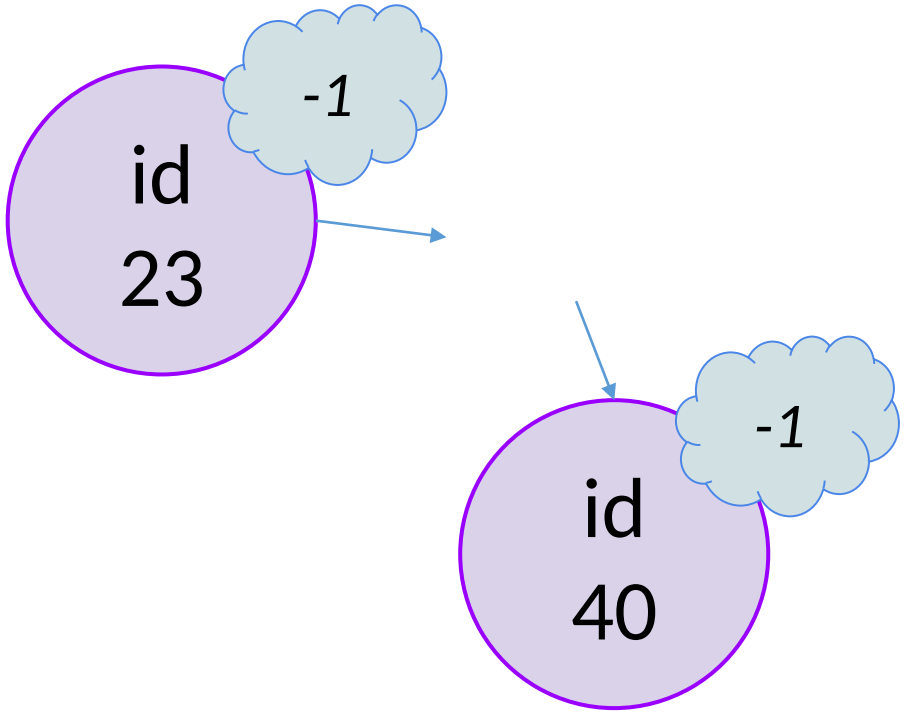
What is a distributed system?

A collection of distinct processes that:

- are spatially separated
- communicate with one another by exchanging messages
- have non-negligible communication delay
- do not share fate
- have separate, imperfect, unsynchronized physical clocks

Leader election

…as a distributed, asynchronous system



New Dafny syntax: modules

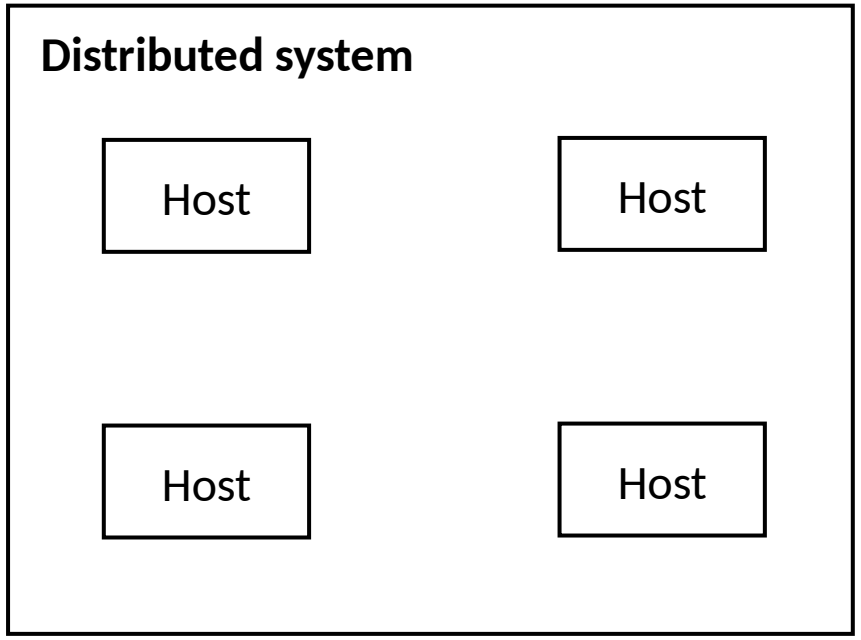
Modules allow us to break up our code into multiple parts

```
module A {
      predicate MyPredicate() { ... }
}

module B {
      import A
      predicate MySecondPredicate() { A.MyPredicate() }
}
```

Modeling distributed systems

A distributed system is composed of multiple hosts

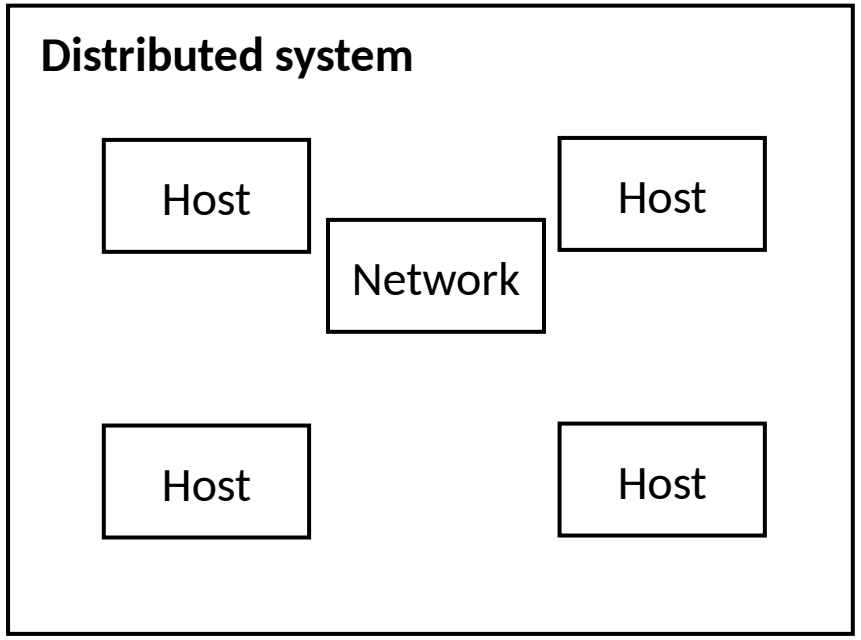


Distributed System: attempt #1

```
module DistributedSystem {
  datatype Variables =
    Variables(hosts:seq<Host.Variables>)

  predicate Next (v:Variables, v’:Variables, hostid: nat) {
    && Host.Next(v.hosts[hostid],v'.hosts[hostid]))
    && forall otherHost:nat | otherHost != hostid ::
        v’.hosts[otherHost] == v.hosts[otherHost]
  }
}
```

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



Distributed system: attempt #2

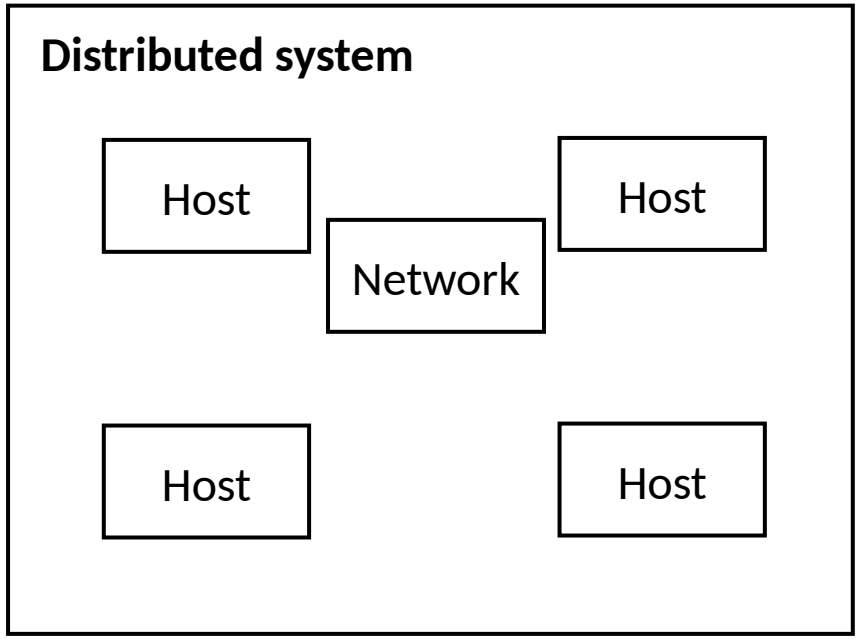
```
module DistributedSystem {
  datatype Variables =
    Variables(hosts:seq<Host.Variables>,
              network: Network.Variables)

  predicate HostAction(v, v', hostid, msgOps) {
    && Host.Next(v.hosts[hostid],v'.hosts[hostid],msgOps))
    && forall otherHost:nat | otherHost != hostid ::
        v'.hosts[otherHost] == v.hosts[otherHost]
  }

  predicate Next(v, v', hostid, msgOps: MessageOps) {
    && HostAction(v, v', hostid, msgOps)
    && Network.Next(v, v', msgOps)
  }
}
```

Binding variable

Defining the network



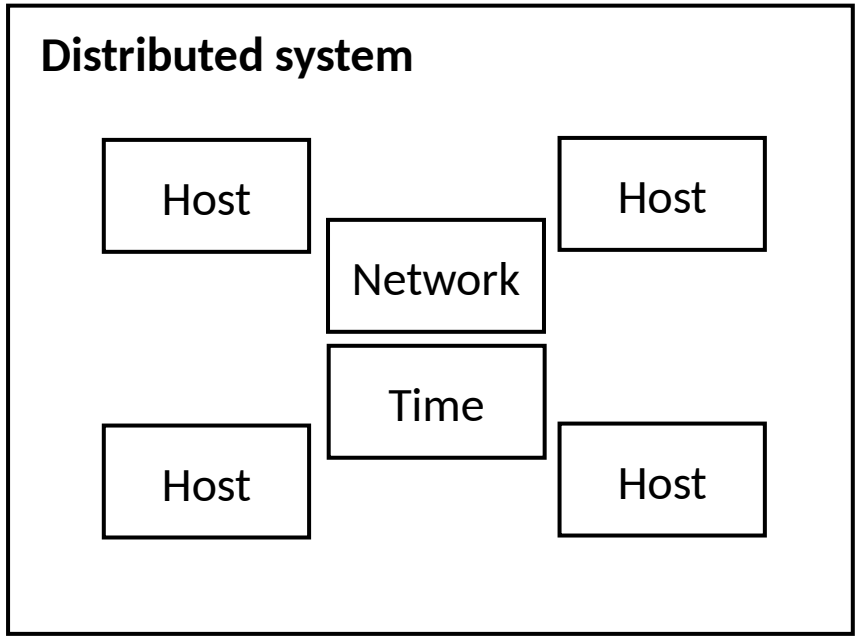

```
datatype Option<T> = Some(value:T) | None
datatype MessageOps = MessageOps(
                    recv:Option<Message>,
                    send:Option<Message>)
```

Network module

```
module Network {
  datatype Variables =
    Variables(sentMsgs: set<Message>)

  predicate Next(v, v', msgOps:MessageOps) {
    // can only receive messages that have been sent
    && (msgOps.recv.Some? ==> msgOps.recv.value in
v.sentMsgs)
    // Record the sent message, if there was one
    && v'.sentMsgs ==
       v.sentMsgs + if msgOps.send.None? then {}
                    else {msgOps.send.value}
  }
}
```

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

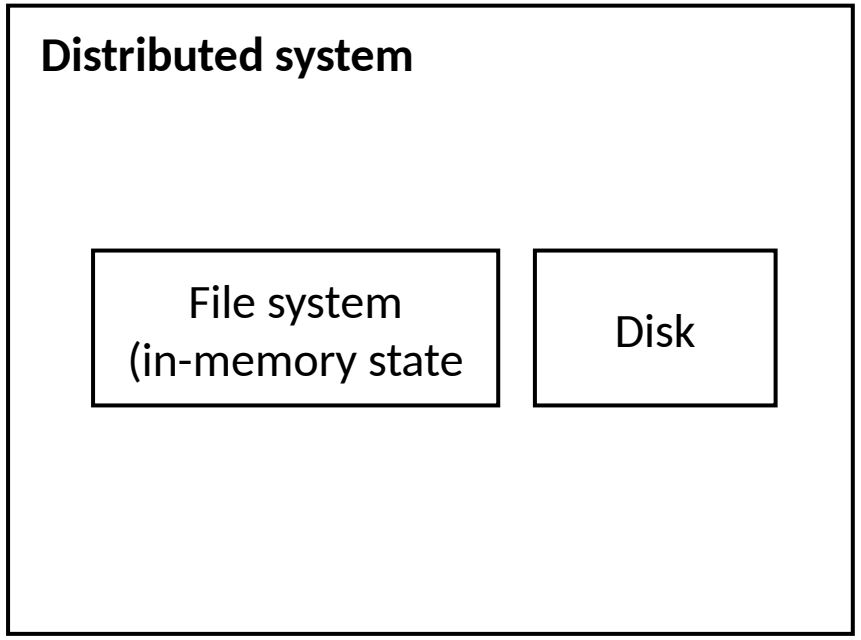


Distributed system: attempt #3

```
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

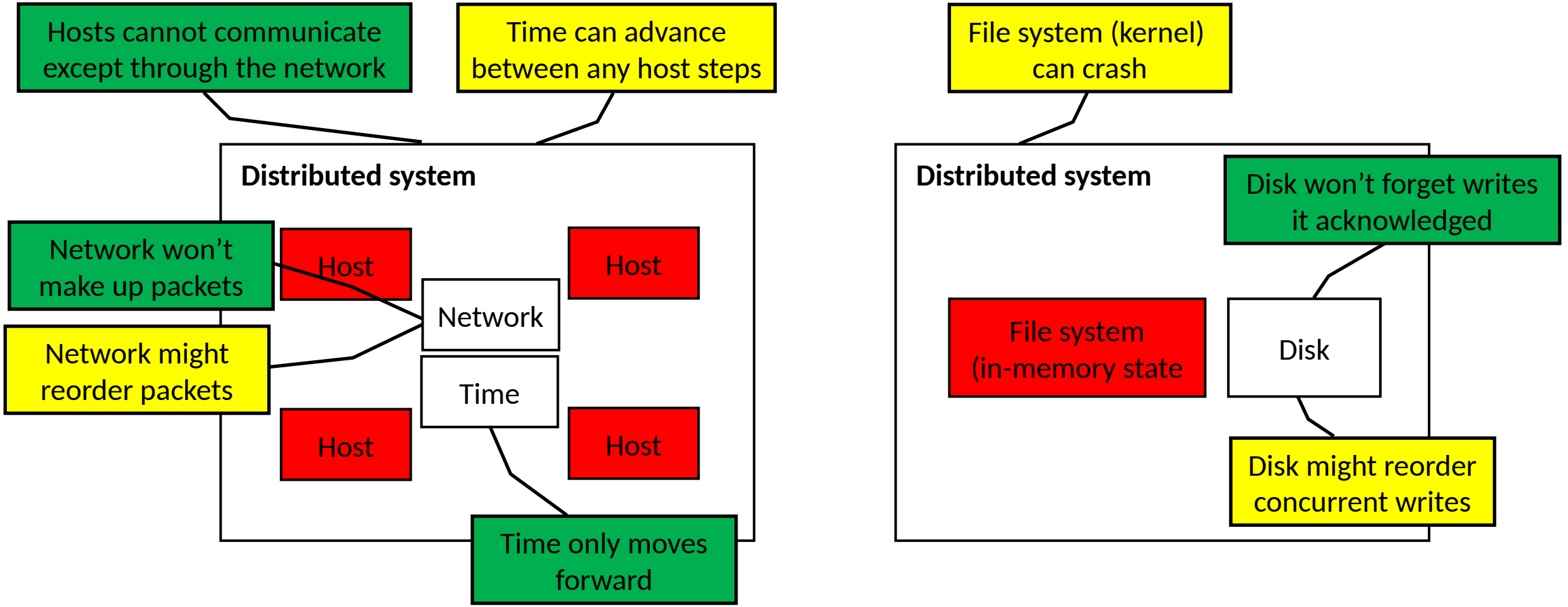



```
module DistributedSystem {
  datatype Variables =
    Variables(fs: FileSystem.Variables,
              disk: Disk.Variables)

  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