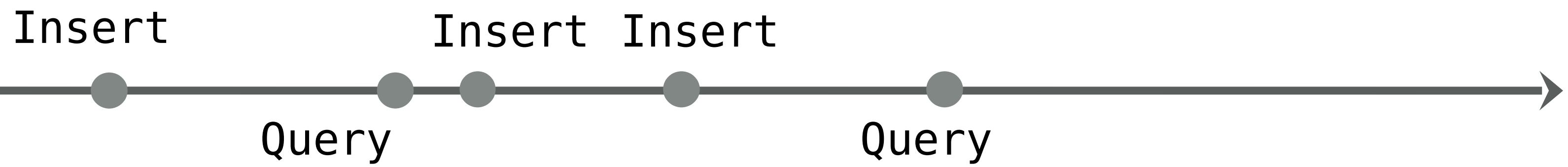
EECS498-008 Formal Verification of Systems Software

Material and slides created by

Jon Howell and Manos Kapritsos

Synchronous specs

```
module MapSpec {
  datatype Variables = Variables(mapp:map<Key, Value>)

  predicate InsertOp(v:Variables, v':Variables, key:Key,
value:Value) {
    ...
  }

  predicate QueryOp(v:Variables, v':Variables, key:Key,
output:Value) {
    ...
  }
}
```



Insert

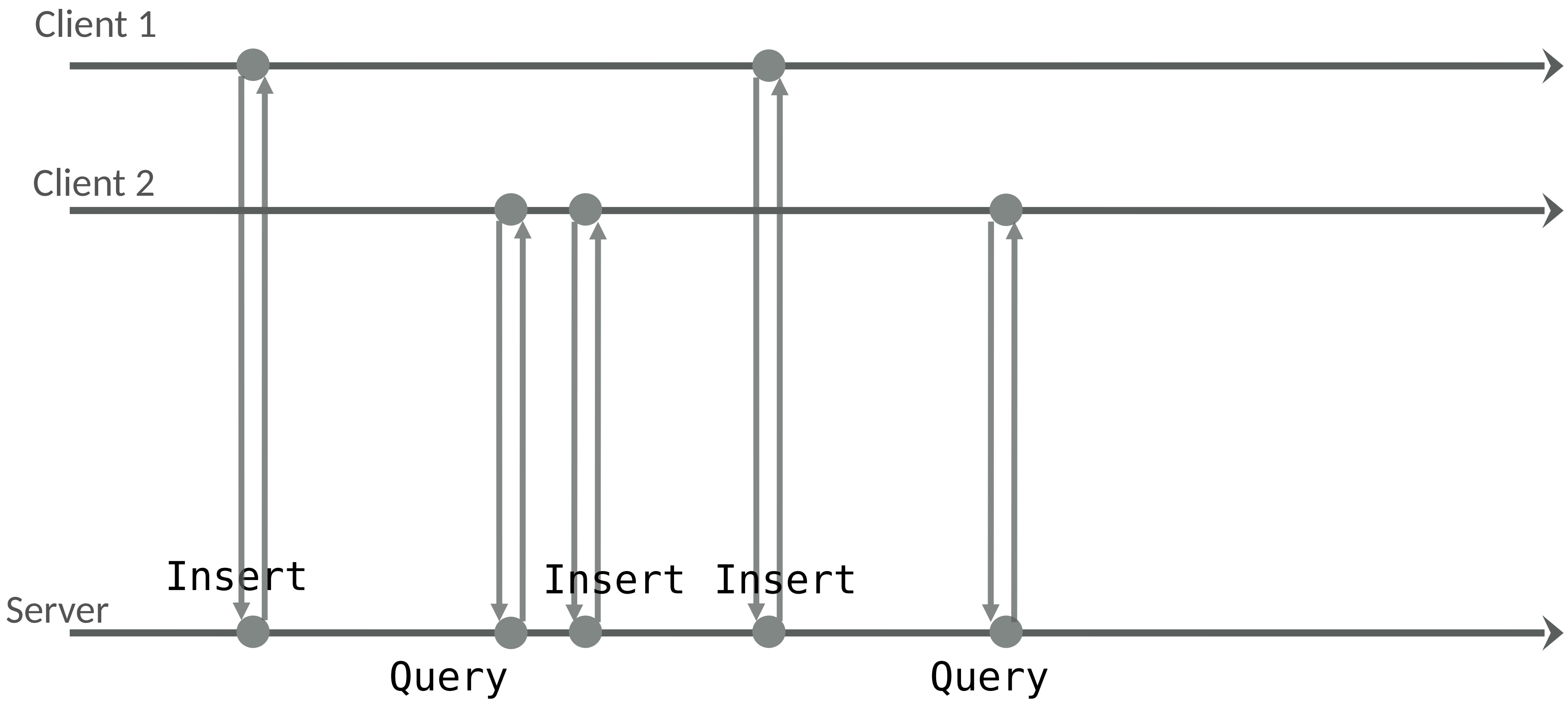
Query

Insert

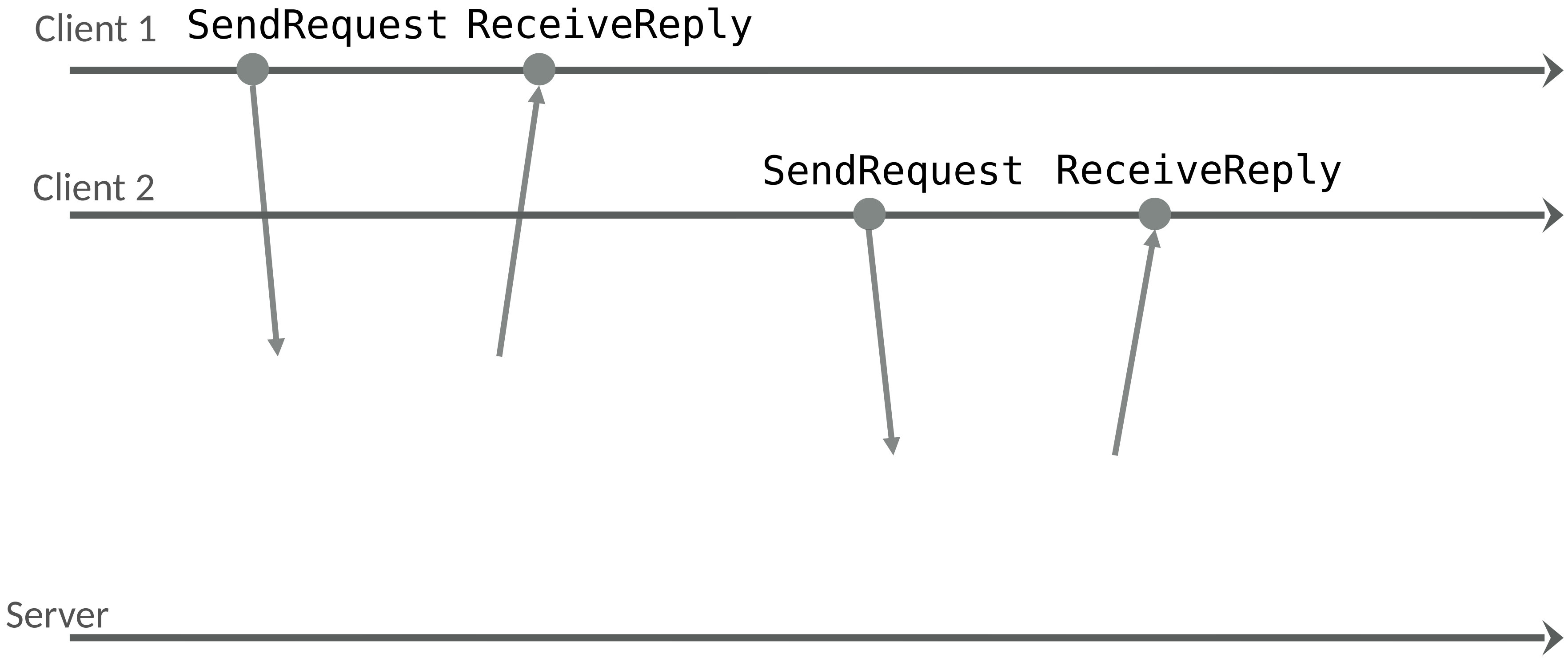
Insert

Query

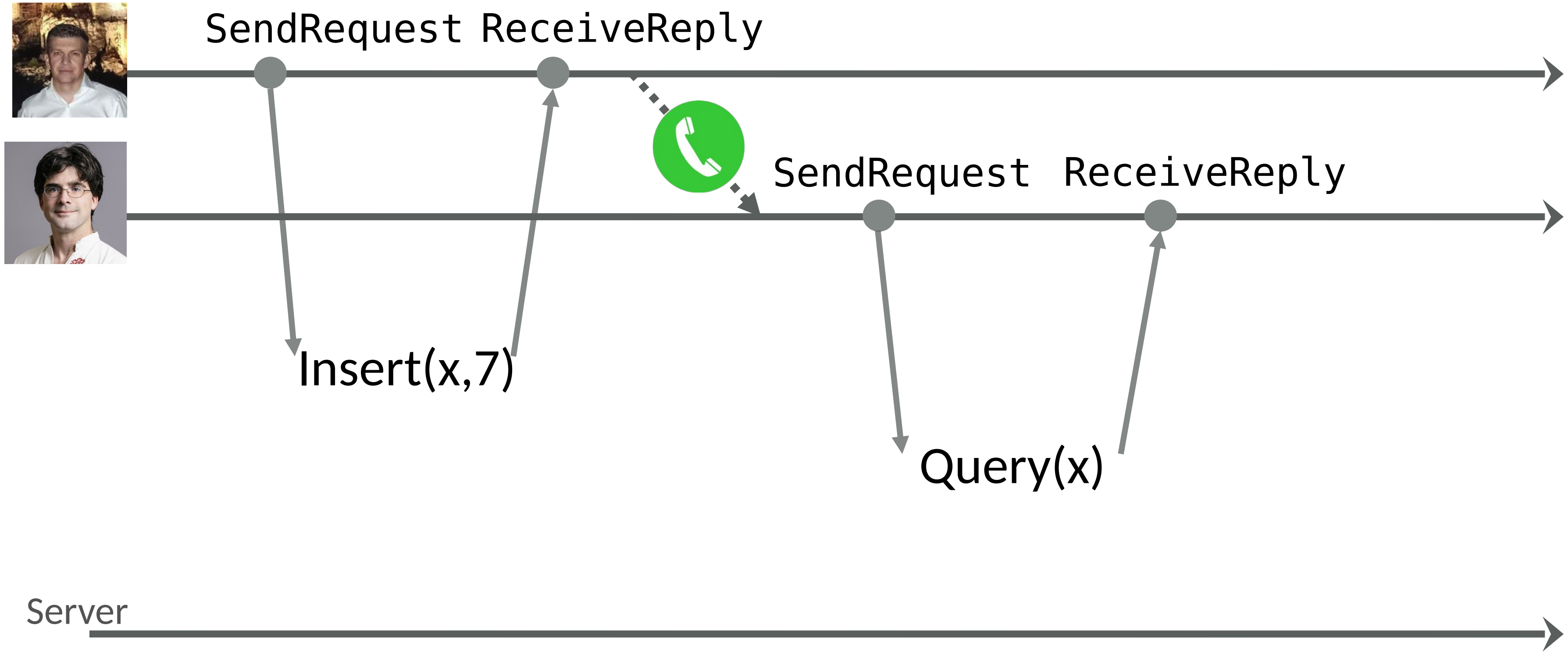
Synchronous specs



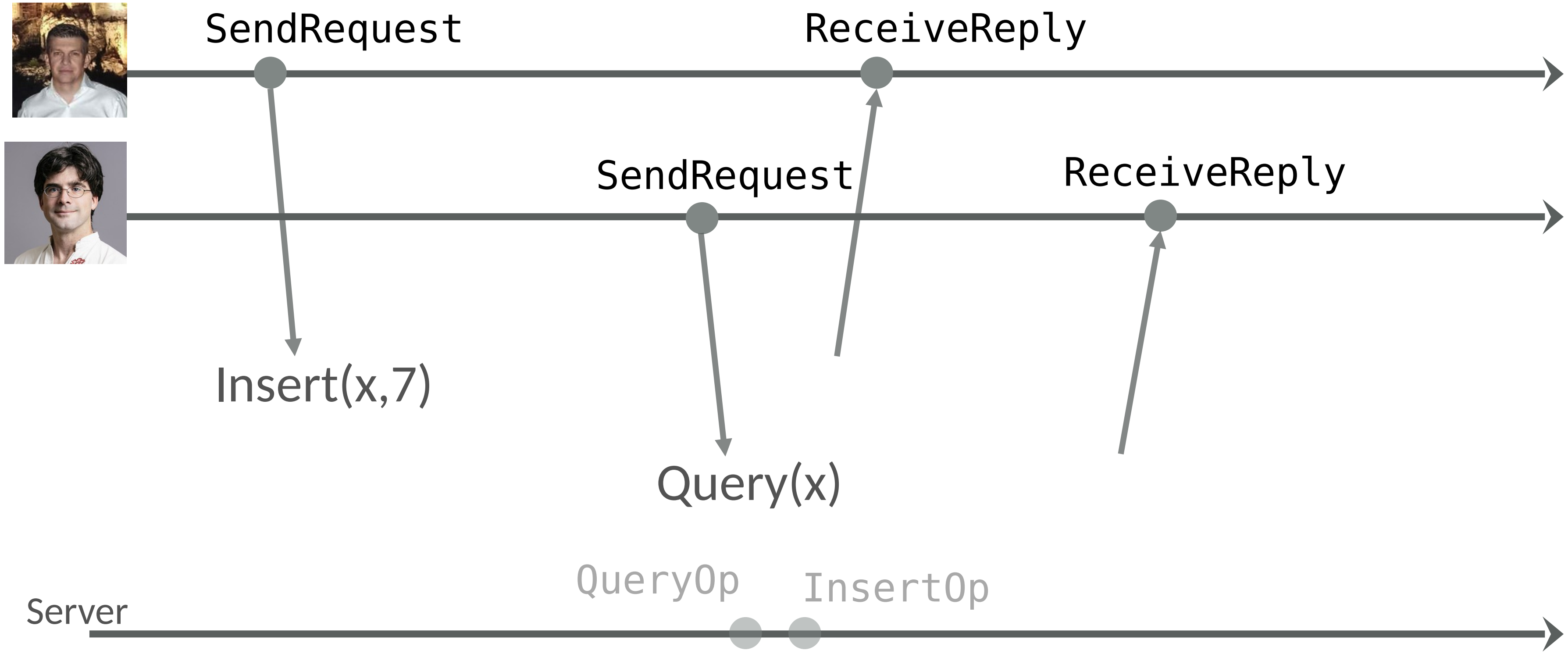
Asynchrony in real life



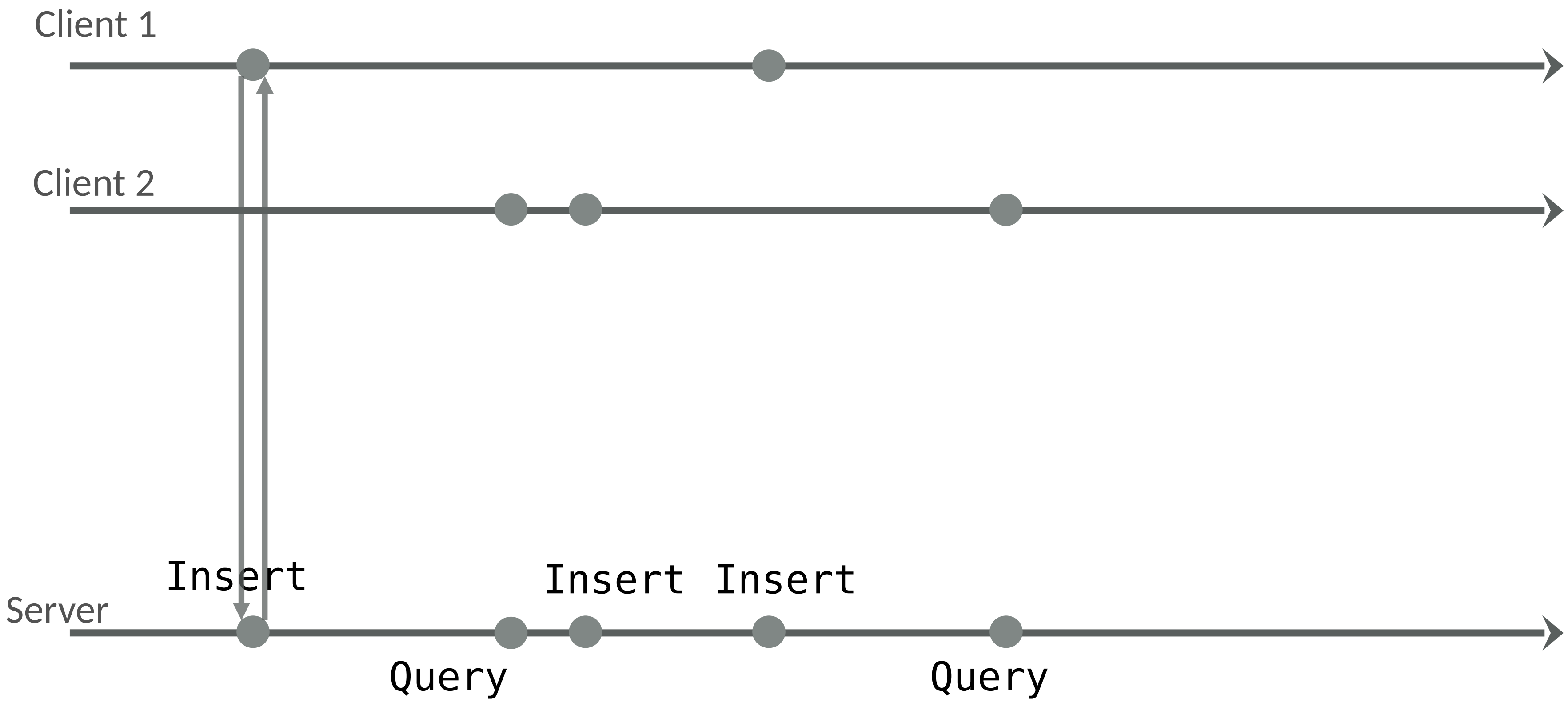
Linearizability



Linearizability

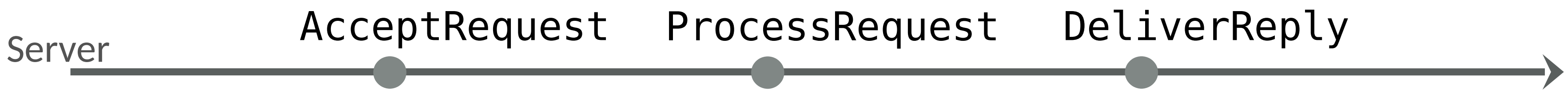


The limitation of Synchronous specs

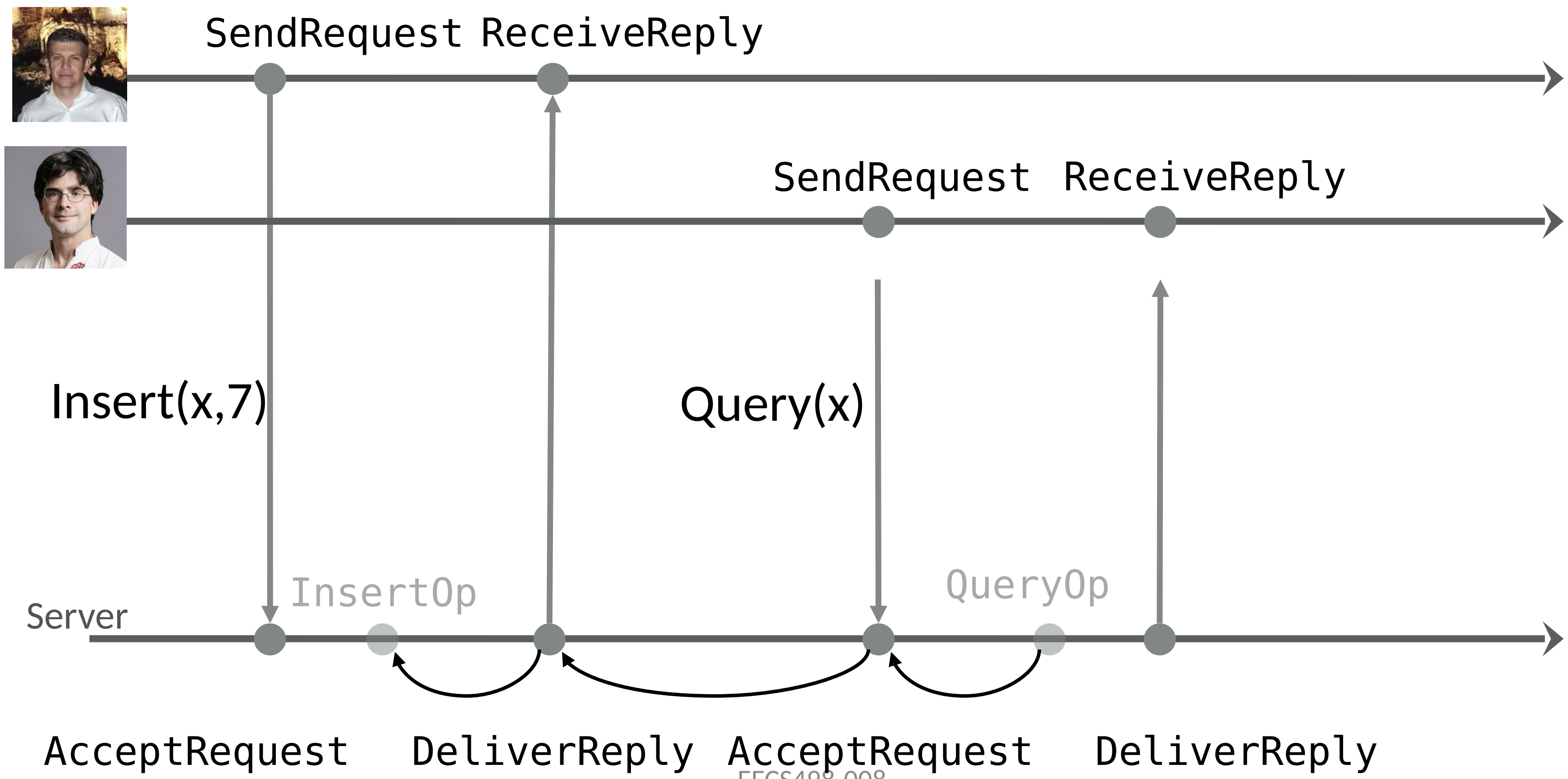


Defining an asynchronous interface

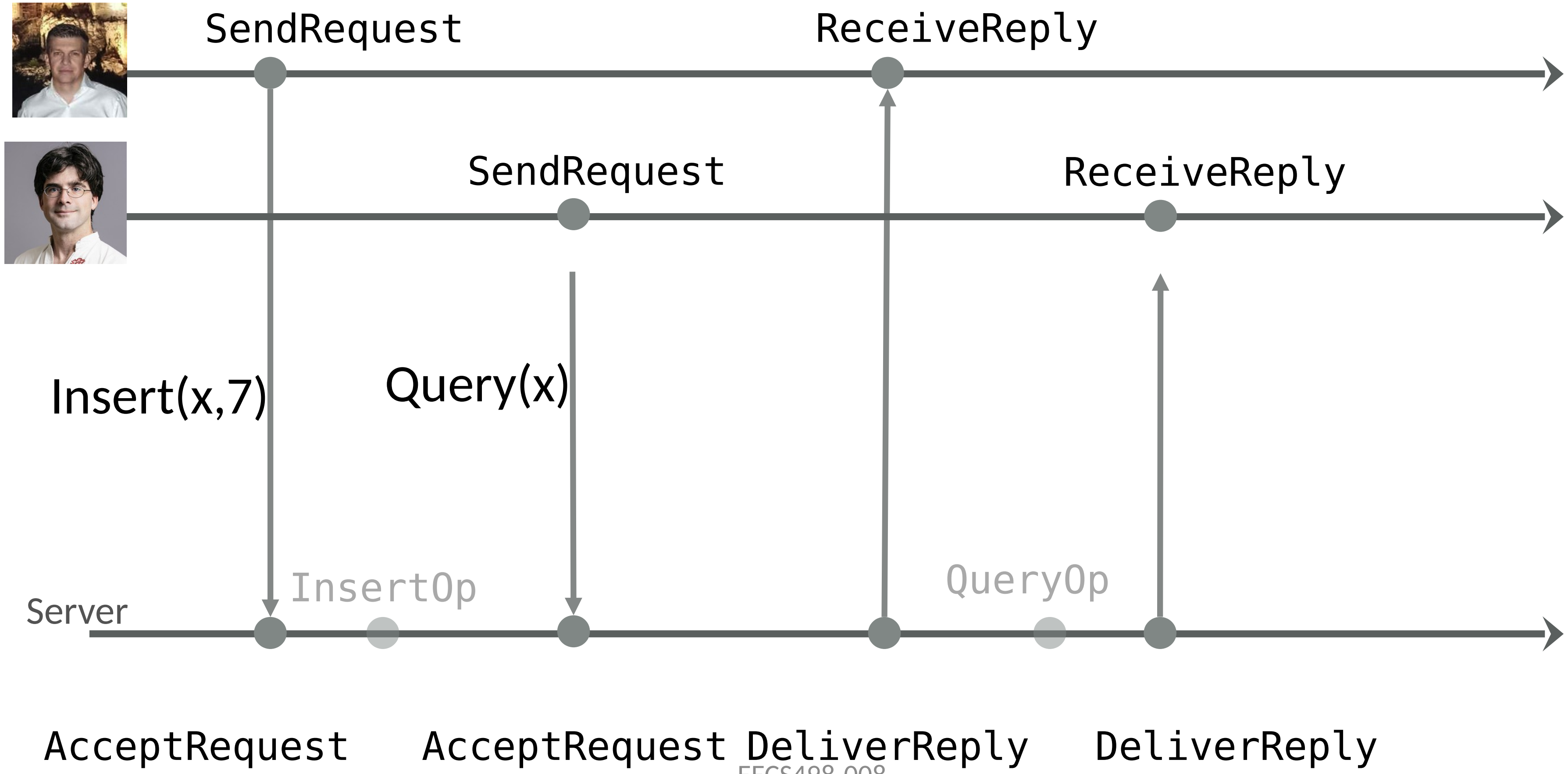
```
module MapSpec {
  datatype Variables = Variables(mapp:map<Key, Value>,
                                   requests:set<Input>, replies:set<Output>)

  predicate InsertOp(v:Variables, v':Variables, request: Input) {...}
  predicate QueryOp(v:Variables, v':Variables, request: Input, output:Value)
{...}
  predicate AcceptRequest(v:Variables, v':Variables, request: Input) {
    // add request to requests, if it’s not there already
  }
  predicate DeliverReply(v:Variables, v':Variables, reply: Output) {
    // remove reply from replies
  }
}
```



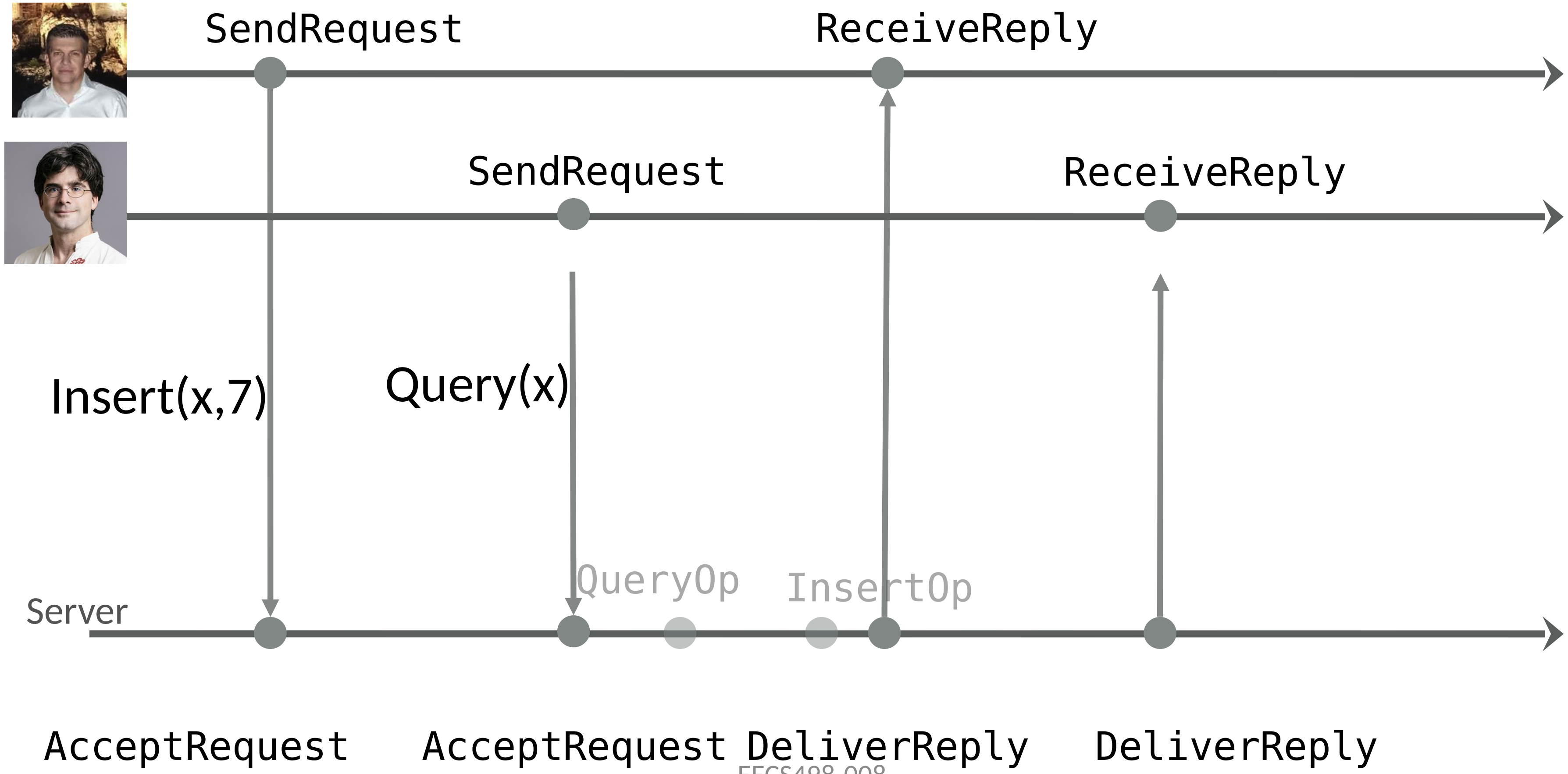
Example run


SendRequest
ReceiveReply
SendRequest
ReceiveReply
Insert(x,7)
Query(x)
Server
InsertOp
QueryOp
AcceptRequest
DeliverReply
AcceptRequest
DeliverReply

Example run #2

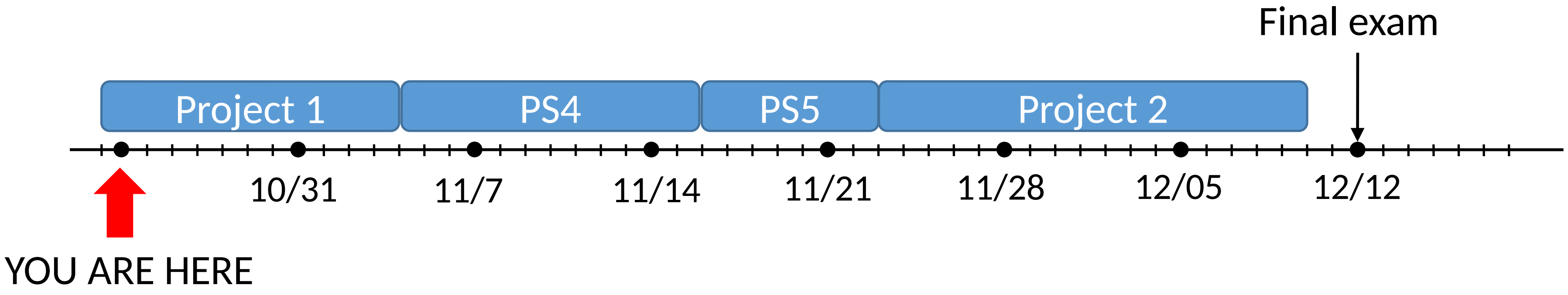


Example run #2



Administrivia

- Project 1 has been released
  - Deadline: Nov 4
  - Groups of (up to 2)
- We will have Jon live with us next Monday!
- No lecture on Nov 2 and Nov 14
- Assignment timeline



Dafny: finite set heuristics

```
predicate IsEven(x:int) {
  x/2*2==x
}

predicate IsModest(x:int) {
 0 <= x < 10
}

lemma IsThisSetFinite() {
  var modestEvens := set x | IsModest(x) &&
IsEven(x);
  assert modestEvens == {0,2,4,6,8};
}
```

Error: the result of a set comprehension must be finite, but Dafny's heuristics can't figure out how to produce a bounded set of values for 'x'

Dafny: finite set heuristics

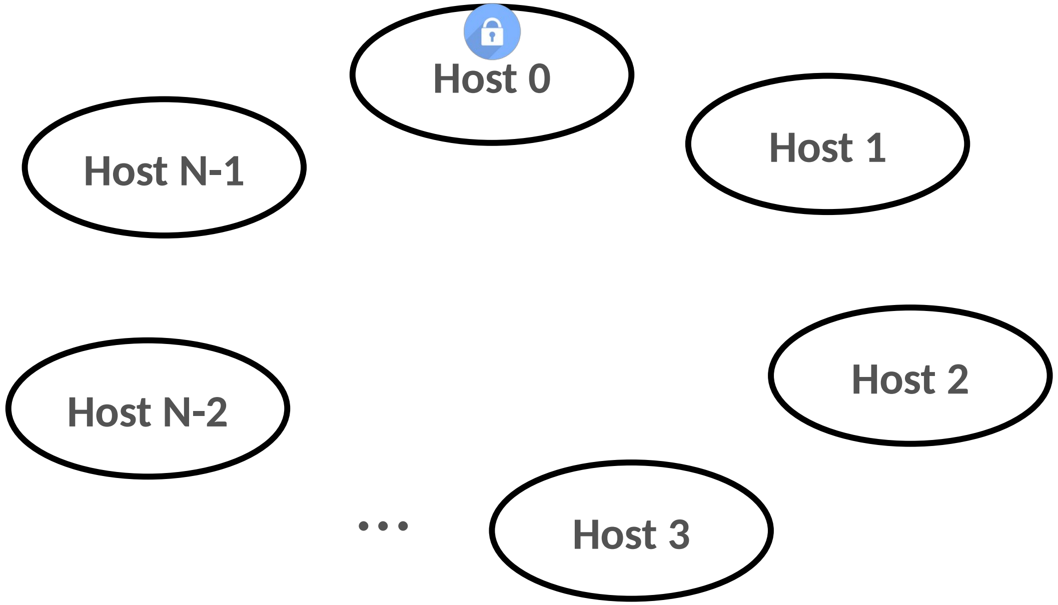
```
predicate IsEven(x:int) {
  x/2*2==x
}

predicate IsModest(x:int) {
  0 <= x < 10
}

function ModestNumbers() : set<int> {
  set x | 0 <= x < 10
}

lemma IsThisSetFinite() {
  var modestEvens := set x | x in ModestNumbers() &&
IsEven(x);
  assert modestEvens == {0,2,4,6,8};
}
```

Distributed lock service

Differences from centralized lock server

- **No centralized server** that coordinates who holds the lock
  - The hosts pass the lock amongst themselves
- The hosts communicate via **asynchronous messages**
  - A single state machine transition **cannot** read/update the state of two hosts

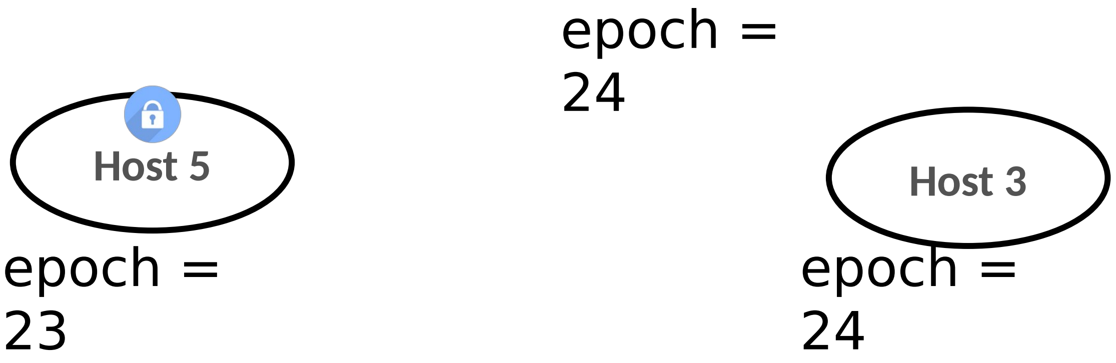
Distributed lock server



- N = numHosts, defined in network.t.dfy
- Messages are asynchronous (i.e. sending and receiving are two separate steps)

Distributed lock server

The lock is associated with a monotonically increasing epoch number



Accept an incoming message only if it has a higher epoch number than your current epoch

Distributed lock server

Safety property:

The desirable property is the same as the centralized lock server: at most one node holds the lock at any given time

Project files

Framework files
(trusted/immutable)

network.t.dfy

distributed_system.t.dfy

Host and proof files
(for you to complete)

host.v.dfy

exercise01.dfy