EECS498-003
Formal Verification of Systems Software

Material and slides created by

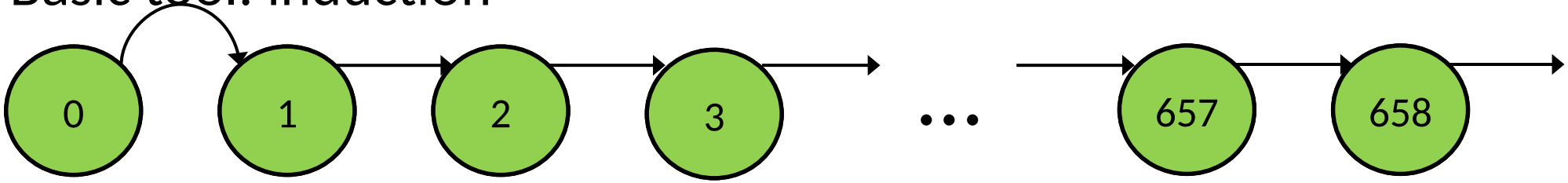
Jon Howell and Manos Kapritsos

Chapter 4: Proving properties

Expressing a system as a state machine allows us to prove that it has certain properties

- We will focus on safety properties
  - i.e. properties that hold throughout the execution

Basic tool: induction



- Show that the property holds on state 0
- Show that if the property holds on state k, it must hold on state k+1

Proving a safety invariant

```
predicate Safety(v:Variables) {
  true // TBD
}

lemma SafetyProof()
  ensures forall v :: Init(v) ==> Safety(v)
  ensures forall v, v' :: Safety(v) && Next(v, v') ==> Safety(v')
{
}
```

Base case

VSCode transition

Inductive Step

Jay Normal Form

As you begin writing more interesting specs, proofs will be nontrivial.

Pull all the nondeterminism into one place, and get a receipt.

image: flickr/afagen CC-by-nc-sa

Jay Normal Form

```
datatype Step =
  | Action1Step( <parameters> )
  | Action2Step( <parameters> )
  ...

predicate NextStep(v: Variables, v': Variables, step:Step)
{
  match step
    case Action1Step(<parameters>) => Action1(v, v', <parameters>)
    case Action2Step(<parameters>) => Action2(v, v', <parameters>)
    ...
}
predicate Next(v: Variables, v': Variables)
{
    exists step :: NextStep(v, v', step)
}
```

VSCode transition

Administrivia

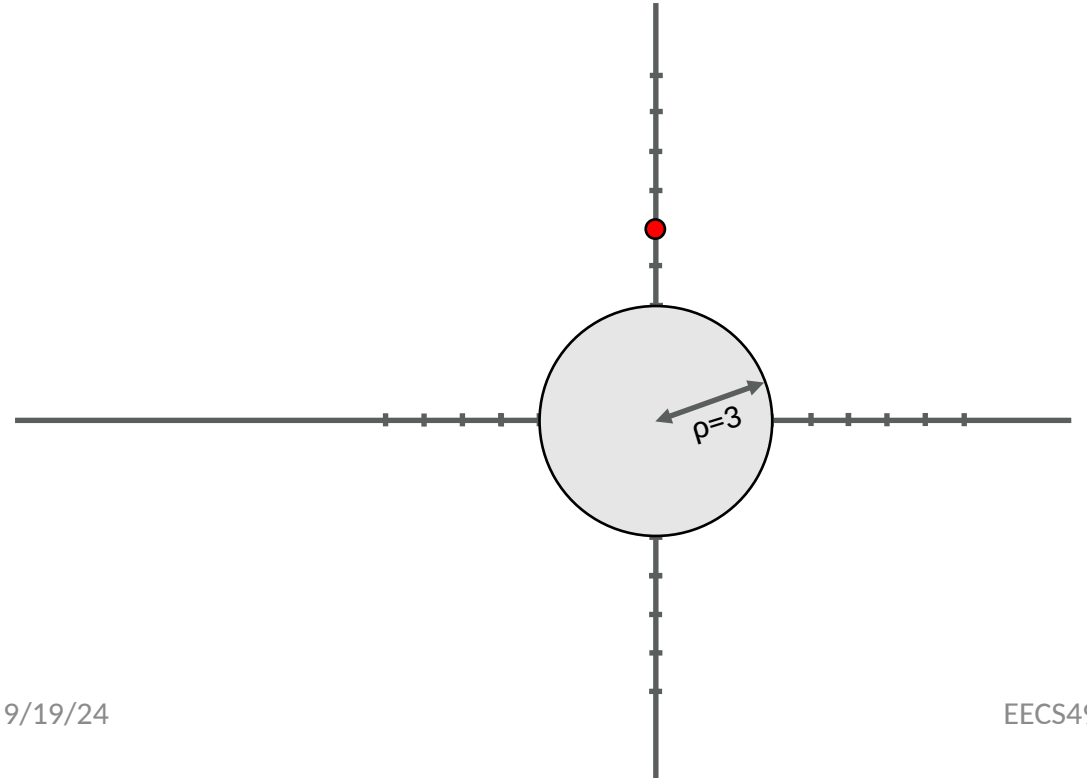
- I have to leave right after class today

- Problem Set 1 due today
- Problem Set 2 will be released tomorrow
  - Chapters 3 and 4
    - Due October 3, 11:59pm
- Reminder: assignment timeline on Piazza

- Not too early to start finding a partner for Project 1

A simple application: Crawler

- Crawler starts at (0,5)
- It can move 1 step north or 1 step south-east
- Can it ever fall in the hole?

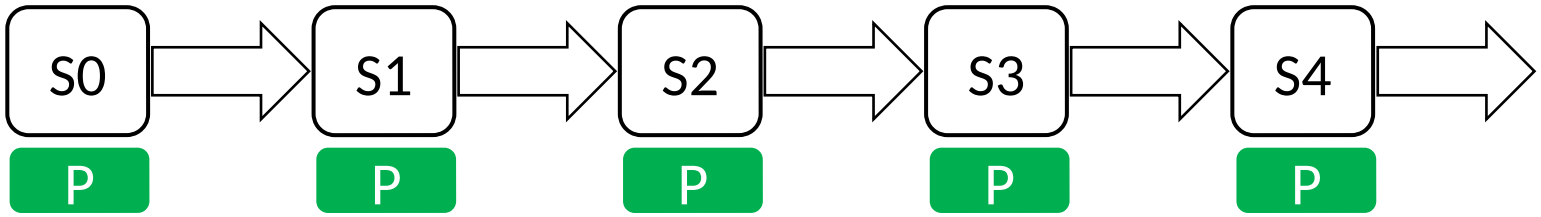



```
predicate Init(v:Variables) {
   && v.x == 0
   && v.y == 5
 }

 predicate MoveNorth(v:Variables, v':Variables) {
   && v'.x == v.x
   && v'.y == v.y + 1
 }

 predicate MoveSouthEast(v:Variables, v':Variables) {
   && v'.x == v.x + 1
   && v'.y == v.y - 1
 }
```

Proving invariants

Proof by induction

- Prove it holds on the first state
- Prove it holds during a transition



Init(v) ==> P(v)

P(v) && Next(v, v') ==> P(v')

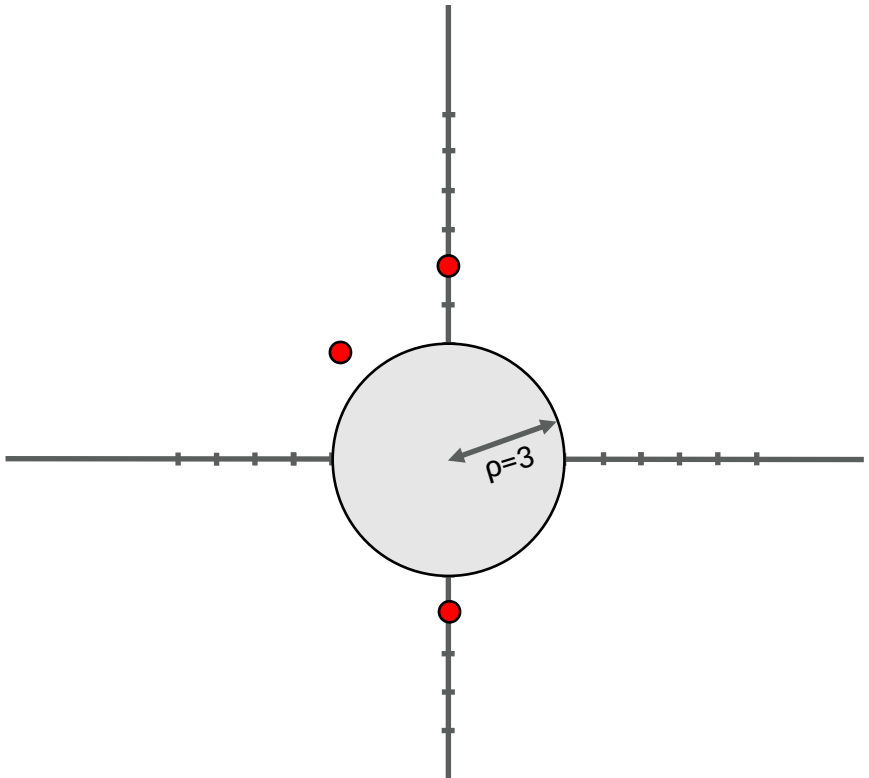
Proving the Crawler

```
predicate Init(v:Variables) {
   && v.x == 0
   && v.y == 5
 }

 predicate MoveNorth(v:Variables, v':Variables) {
   && v'.x == v.x
   && v'.y == v.y + 1
 }

 predicate MoveSouthEast(v:Variables, v':Variables) {
   && v'.x == v.x + 1
   && v'.y == v.y - 1
 }
```

```
predicate Safety(v:Variables) {
    v.x*v.x + v.y*v.y > 3*3
}
```

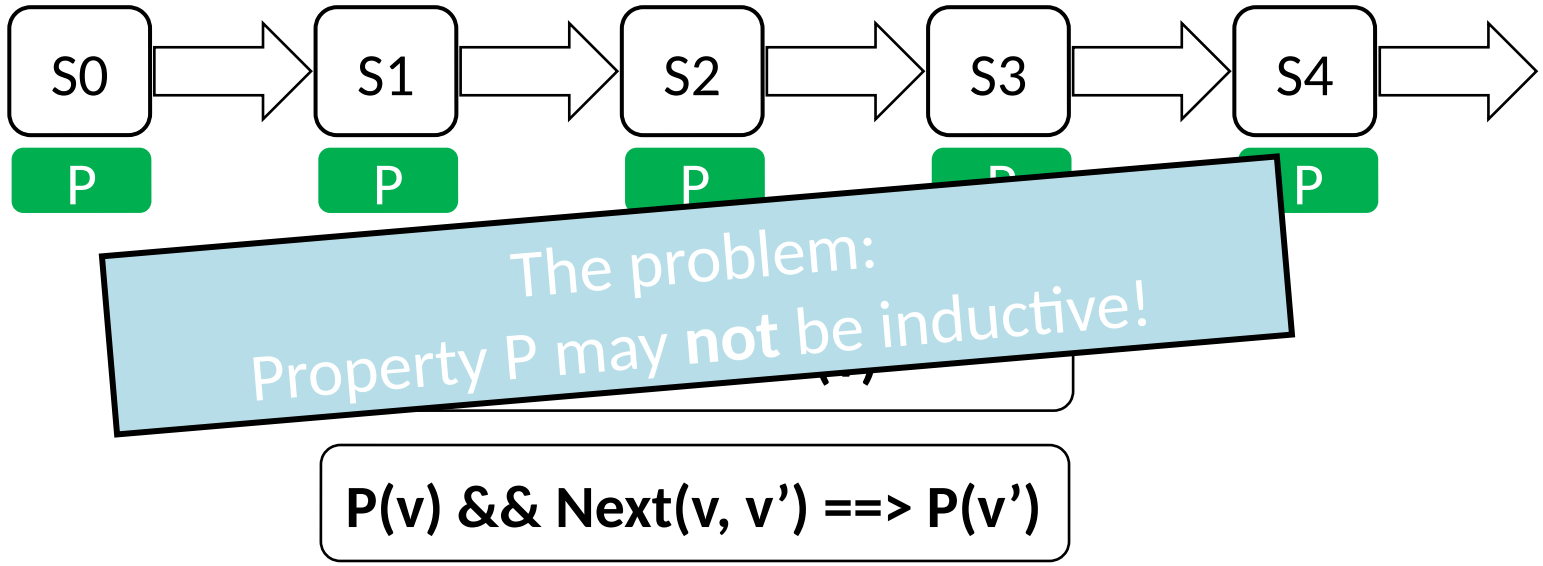
Init(v) ==> P(v) ✓

P(v) && Next(v, v') ==> P(v') ✗

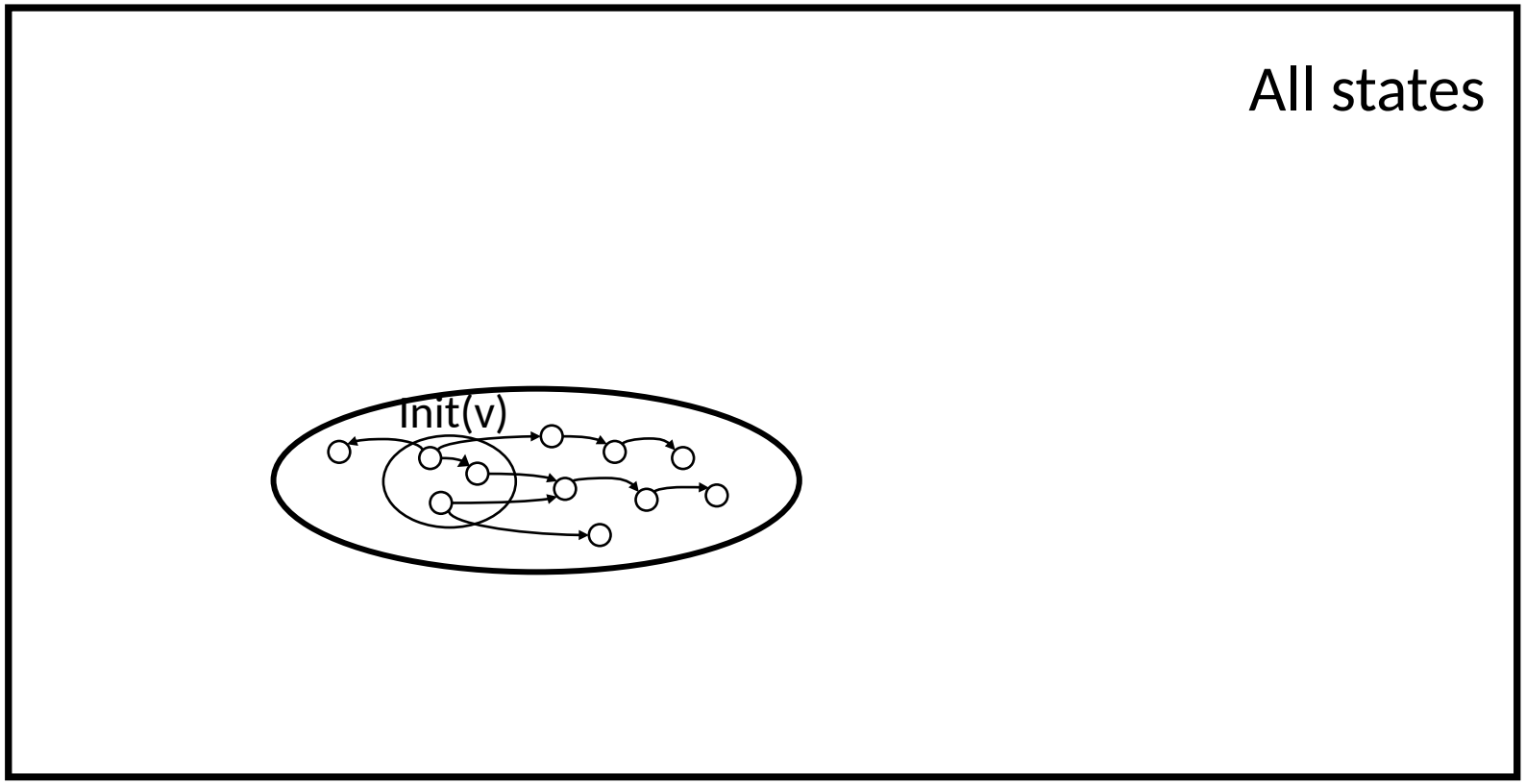


Inductive invariants

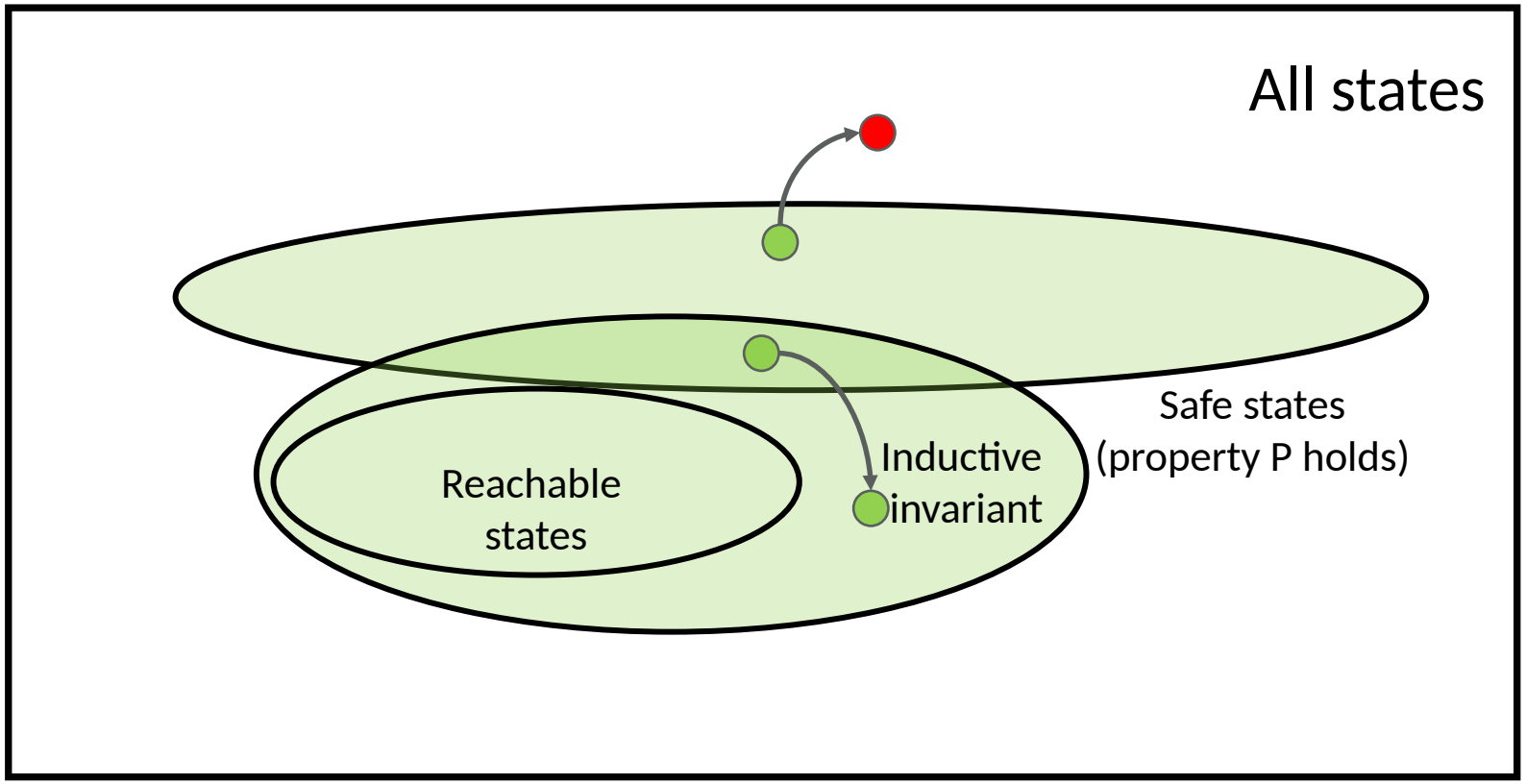
Safety property (a.k.a. invariant):
a property that **always** holds



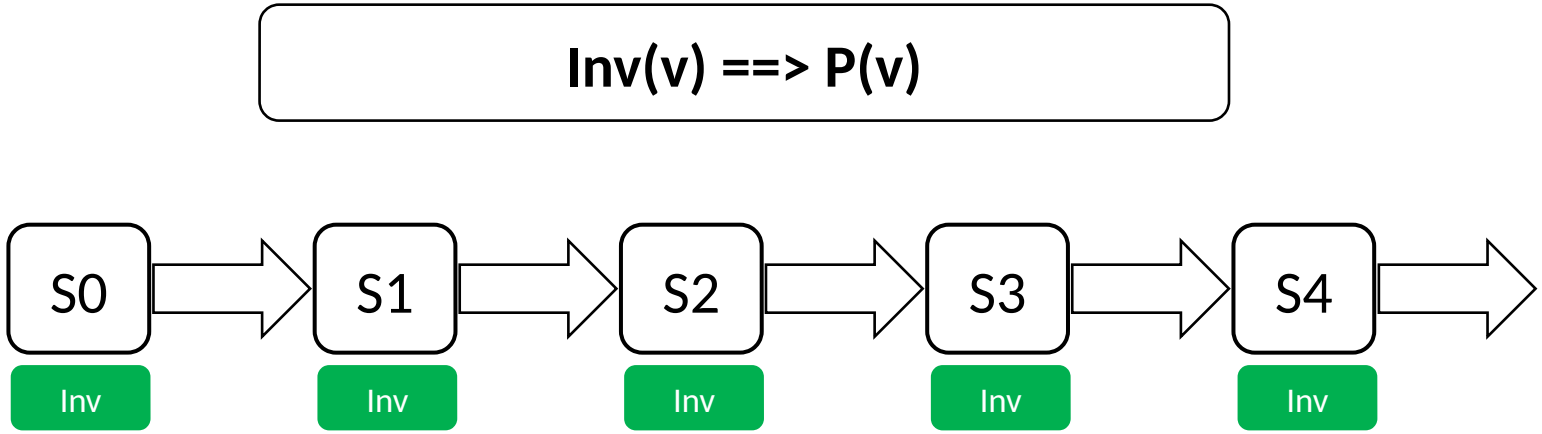
Invariants vs Inductive invariants



Invariants vs Inductive invariants



Proving safety with Inductive invariants

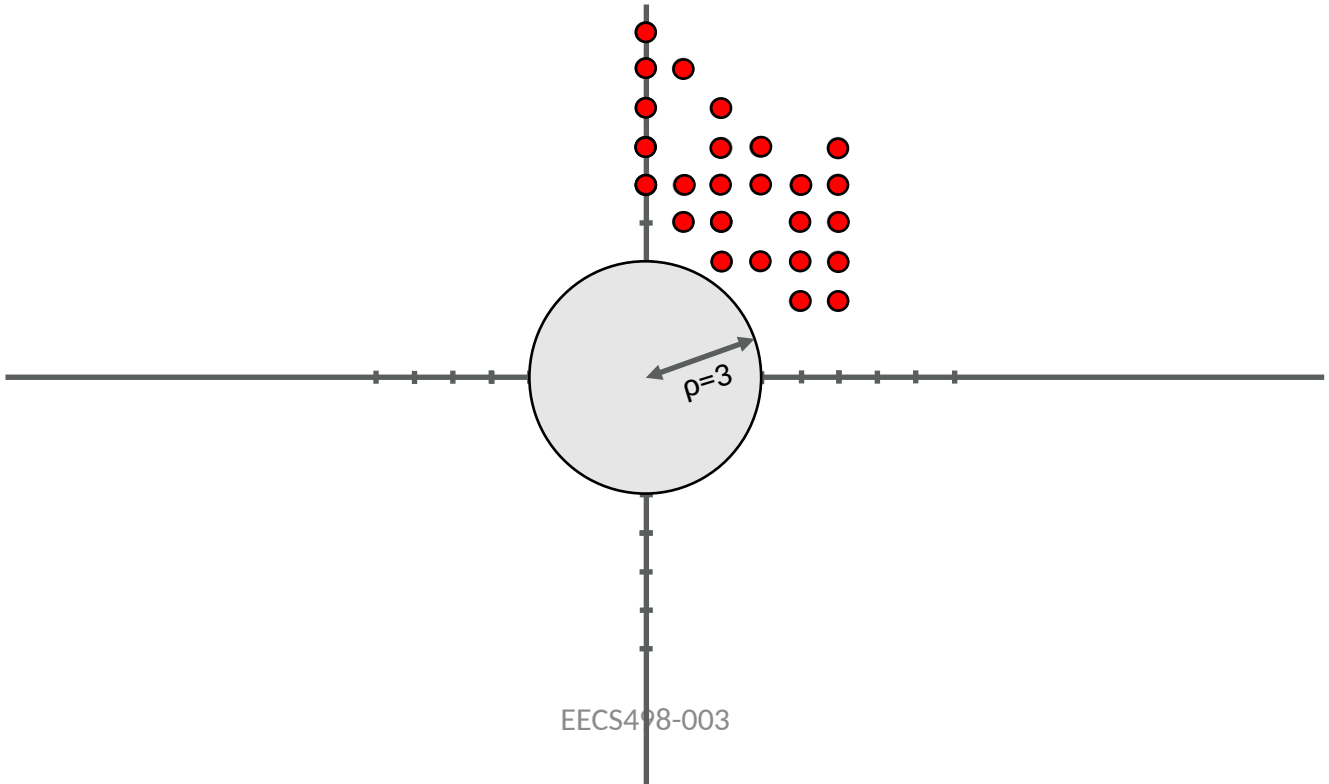


Init(v) ==> Inv(v)

Inv(v) && Next(v, v') ==> Inv(v')

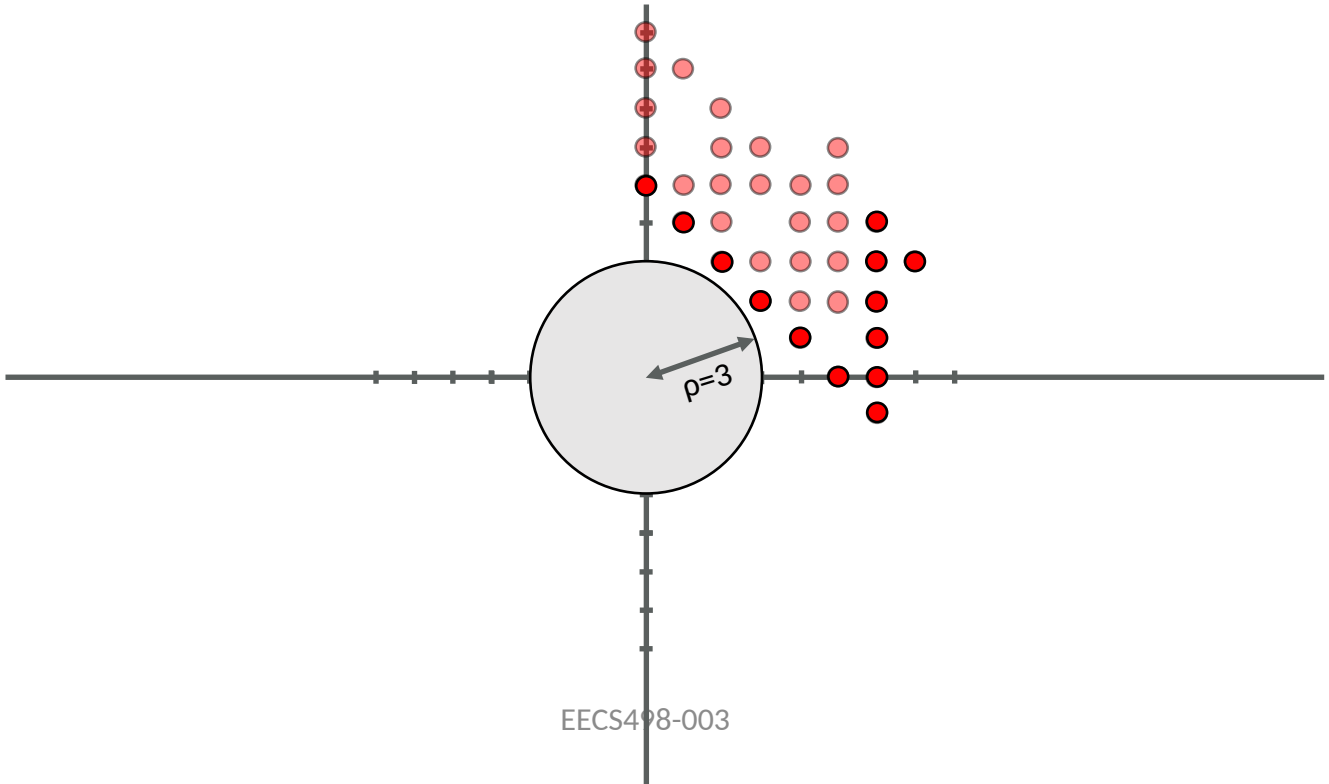
Proving the Crawler

Can the crawler ever fall in the hole?



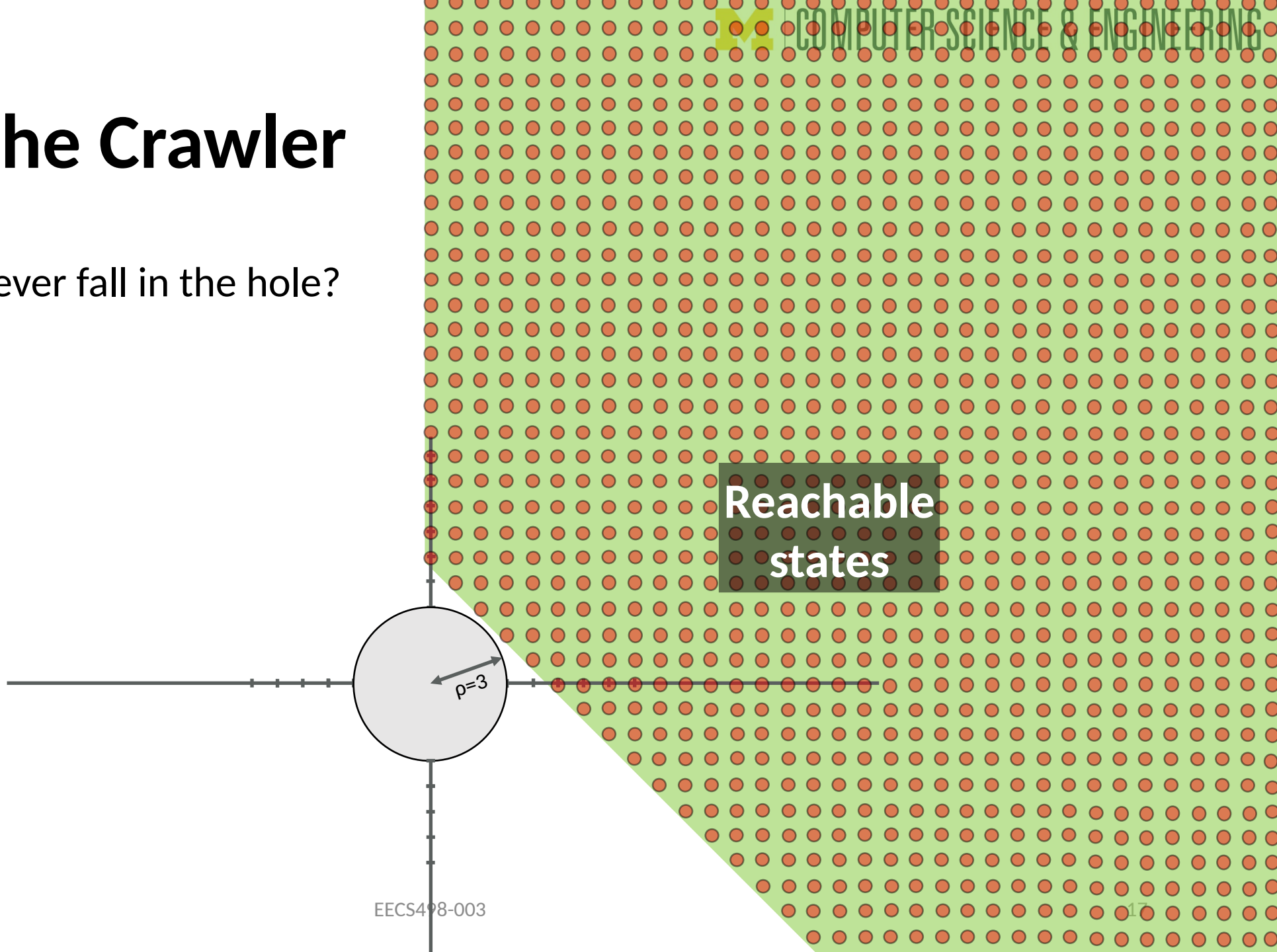
Proving the Crawler

Can the crawler ever fall in the hole?



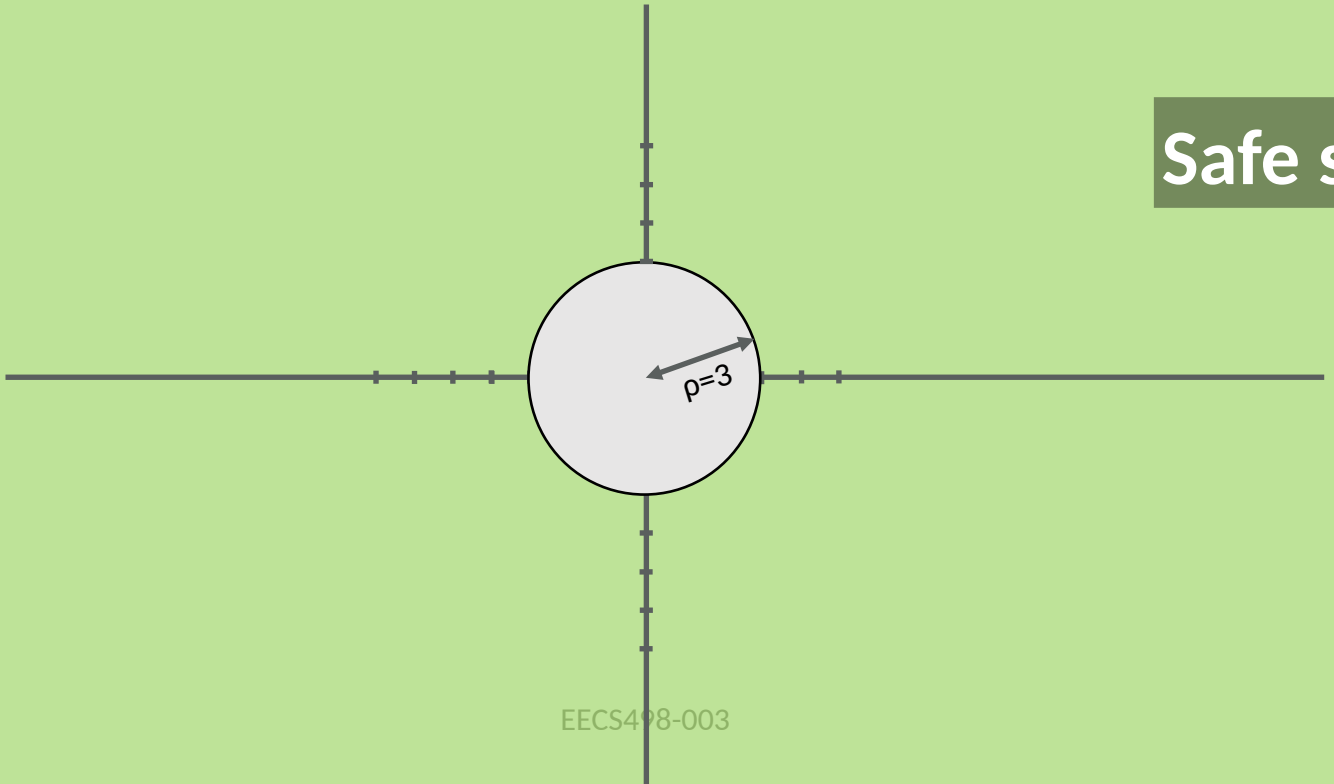
Proving the Crawler

Can the crawler ever fall in the hole?



Naïve safety proof

Safe(v) && Next(v, v') ==> Safe(v')

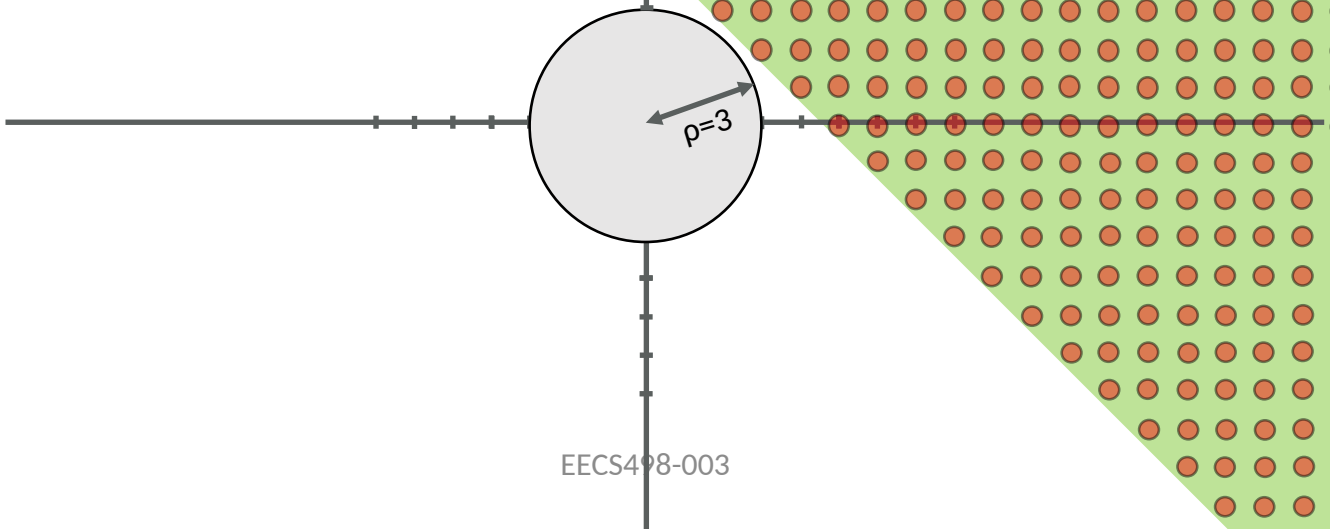


Safe states

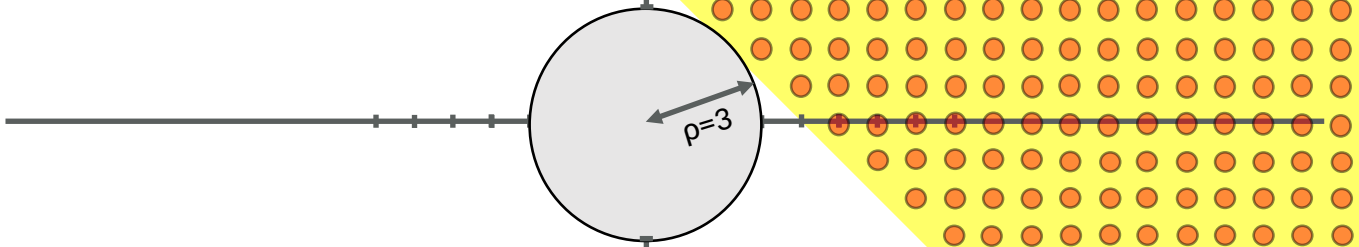
Safety proof using a stronger invariant

InGreenRegion(v) && Next(v, v') ==>
InGreenRegion(v')

The set of reachable states is always an inductive invariant



A simpler inductive invariant



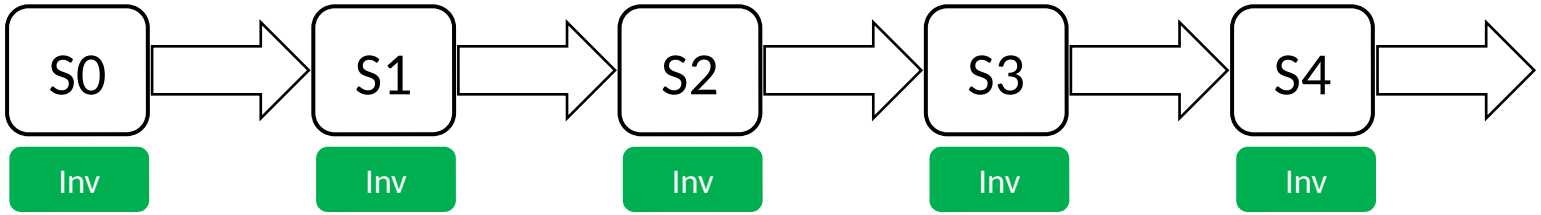
InYellowRegion(v) && Next(v, v') ==>
InYellowRegion(v')

Proving safety with Inductive invariants

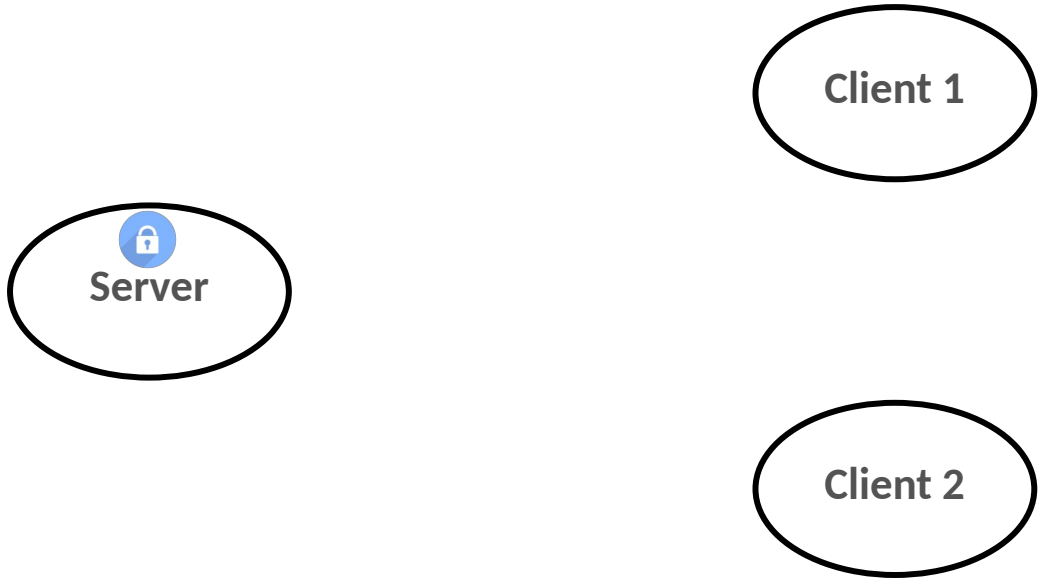
Inv(v) ==> Safety(v)

Init(v) ==> Inv(v)

Inv(v) && Next(v, v') ==> Inv(v')



Example: lock server



datatype Variables = Variables(S: bool, C1: bool, C2:bool)

ghost predicate Safety(v) { !(v.C1 && v.C2) }

Both clients cannot hold the lock at the same time

Example: lock server