COMPUTER SCIENCE & ENGINEERING

EECS498-003 Formal Verification of Systems Software

Material and slides created by

Jon Howell and Manos Kapritsos

Chapter 3: State Machines

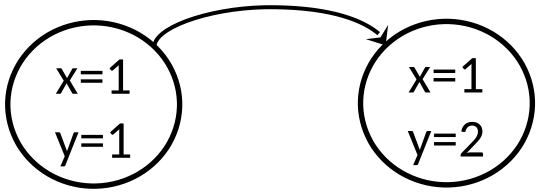
Building state machines

A state is an assignment of values to variables

An action is a transition from one state to another

An execution is a sequence of states

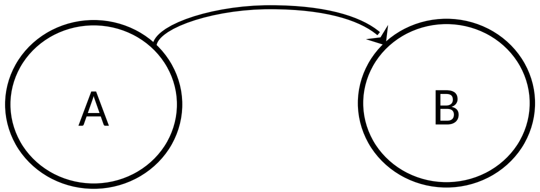
We will capture executions with state machines



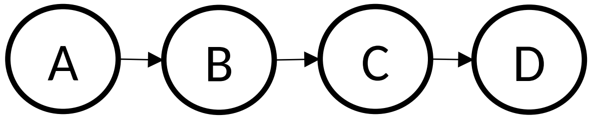
Building state machines

A state is an assignment of values to variables

An action is a transition from one state to another

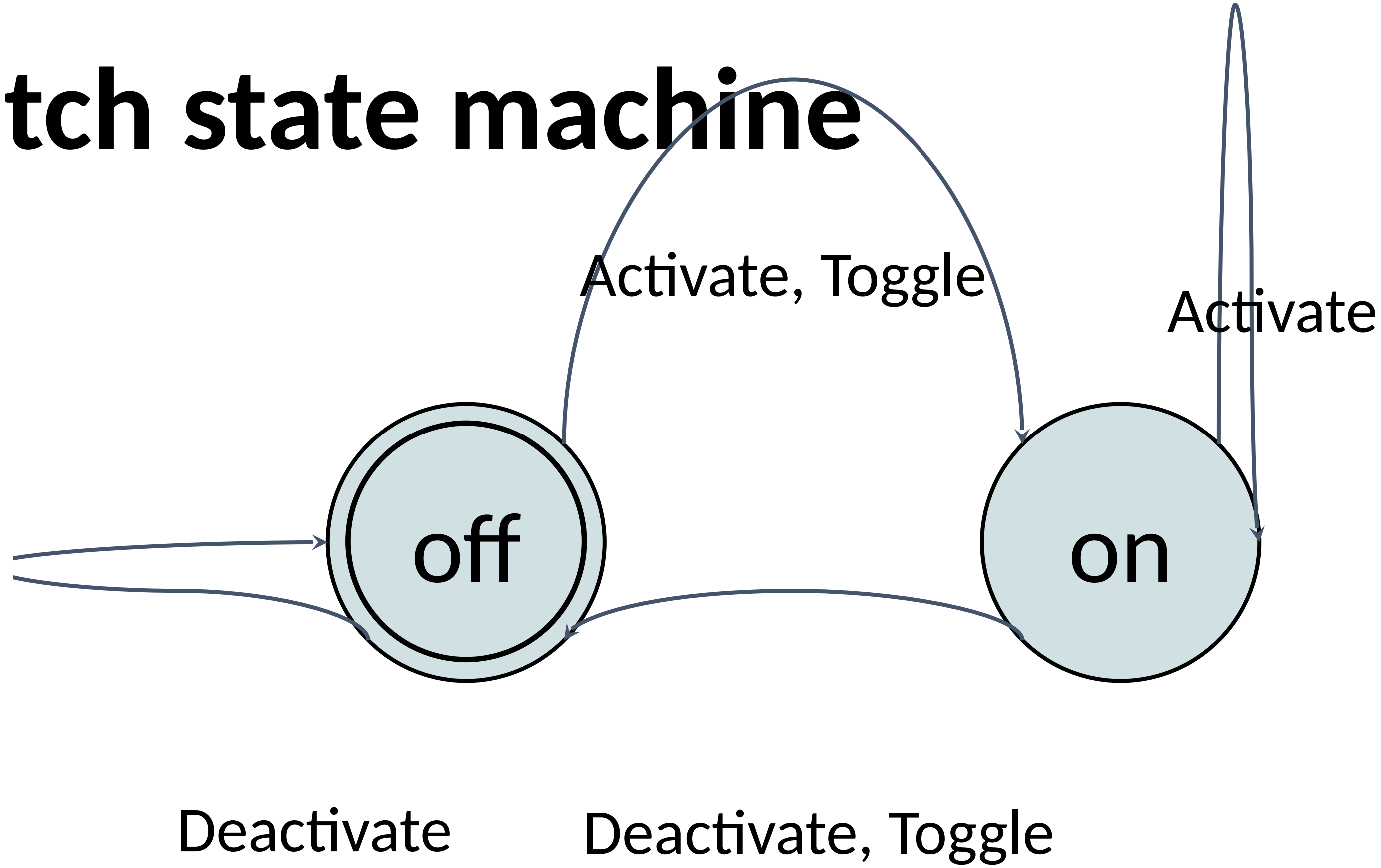


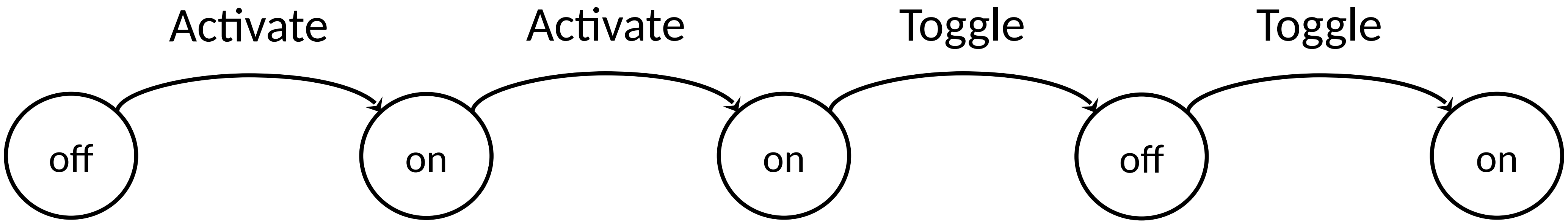
An execution is a sequence of states



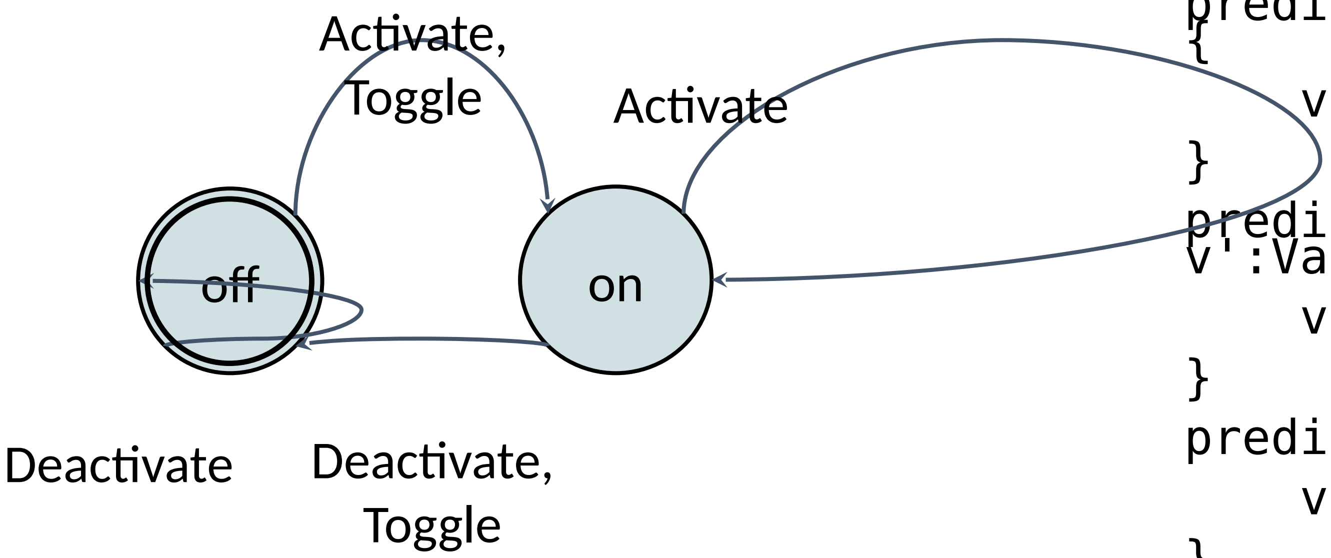
We will capture executions with state machines

The Switch state machine





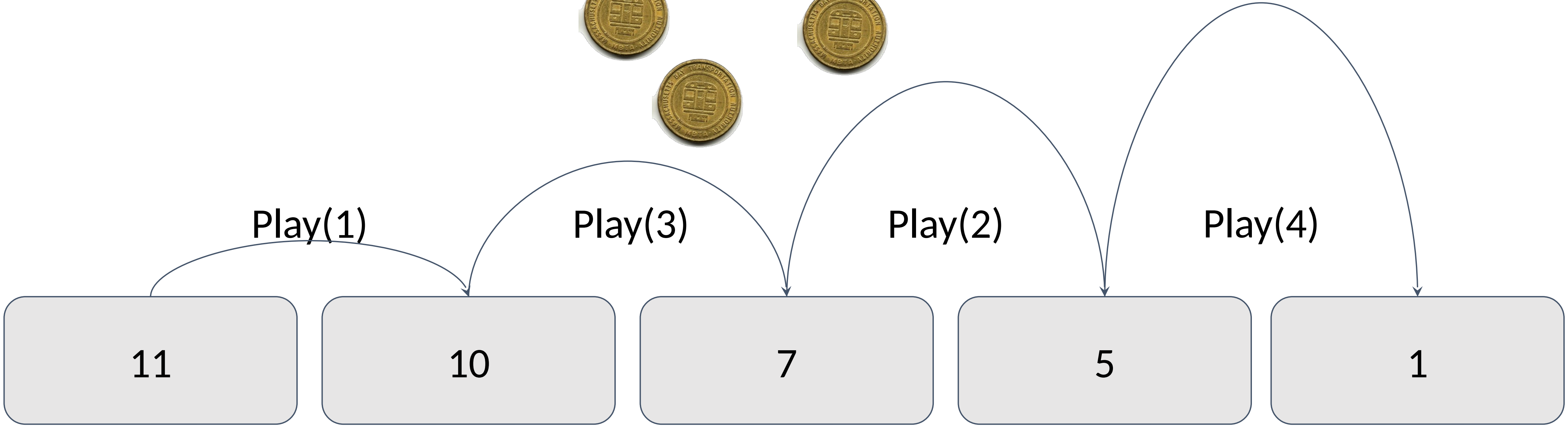
The Switch state machine




```
predicate Activate(v:Variables, v':Variables) {
    v'.switch == On
}
predicate Deactivate(v:Variables, v':Variables) {
    v'.switch == Off
}
predicate Toggle(v:Variables, v':Variables) {
    v'.switch != v.switch
}
predicate Next(v:Variables, v':Variables) {
    || Activate(v, v')
    || Deactivate(v, v')
    || Toggle(v, v')
}
```

```
datatype SwitchState = On | Off
datatype Variables =
        Variables(switch:SwitchState)
predicate Init(v:Variables) {
    v.switch == Off
}
```

The Game of Nim

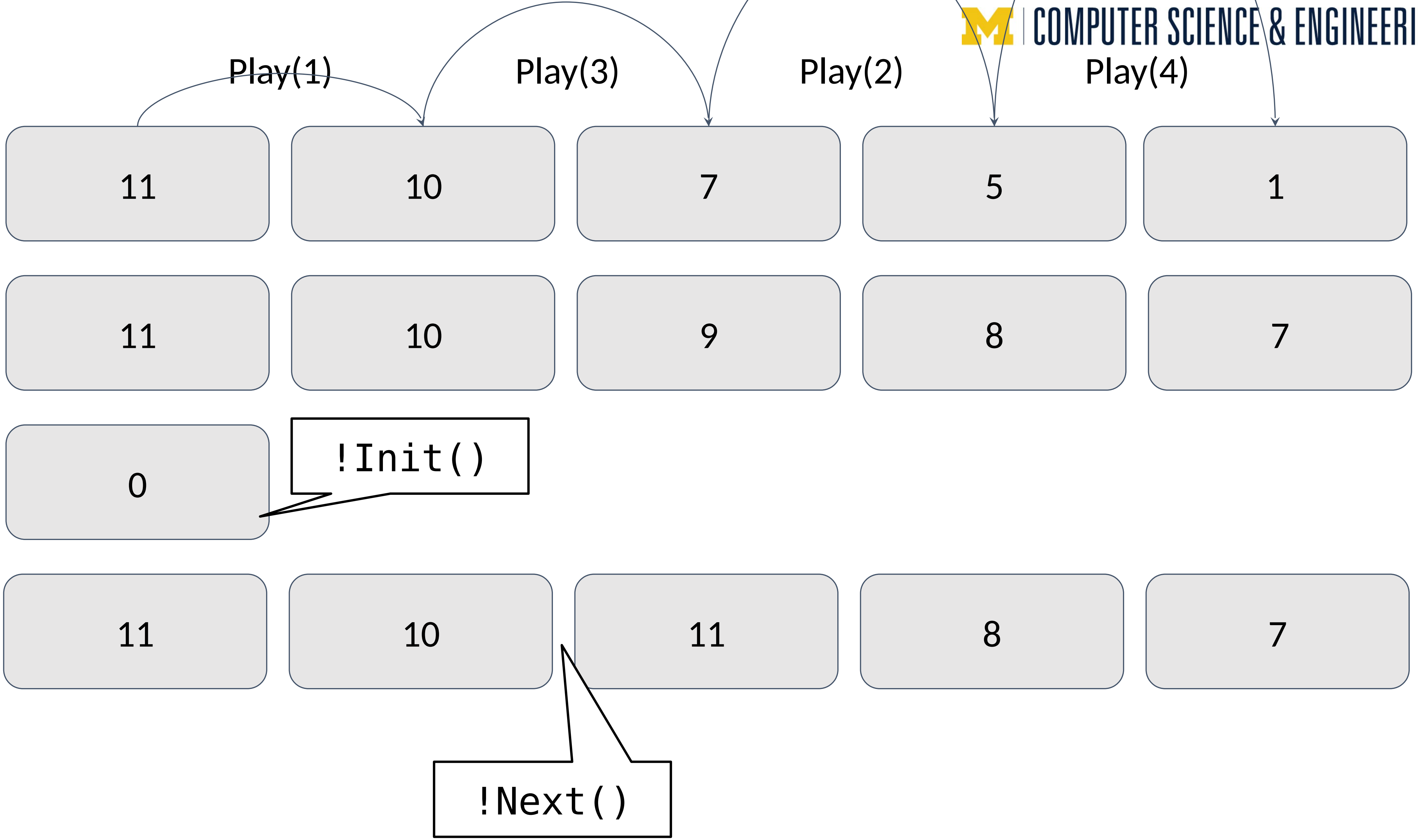


The Nim state machine

```
datatype Variables = Variables(tokens:nat)
predicate Init(v:Variables) {
    v.tokens > 0
}

predicate Play(v:Variables, v':Variables, take:nat) {
    && 1 <= take <= 4                       } enabling condition
    && v'.tokens == v.tokens - take         } "update"
}

predicate Next(v:Variables, v':Variables)
{
    exists take :: Play(v, v', take)
}
```


Play(1)
Play(3)
Play(2)
Play(4)
11
10
7
5
1
11
10
9
8
7
0
!Init()
11
10
11
8
7
!Next()

Administrivia

- Remember that Problem Set 1 is due next Thursday, September 19
- My office hours today were moved to 5-6pm

A simple library app

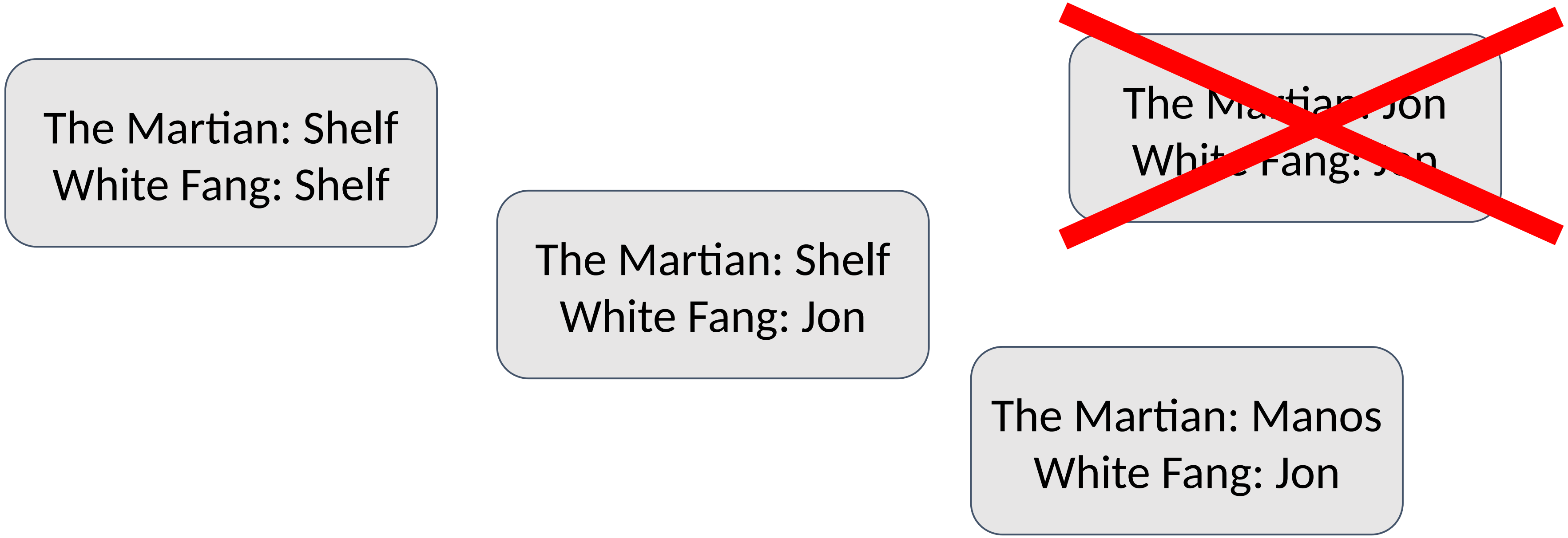
```
datatype Card = Shelf | Patron(name:
string)
datatype Book = Book(title: string)
type Variables = map<Book, Card>
```

Small-library rule: each patron can have *at most one book* checked out

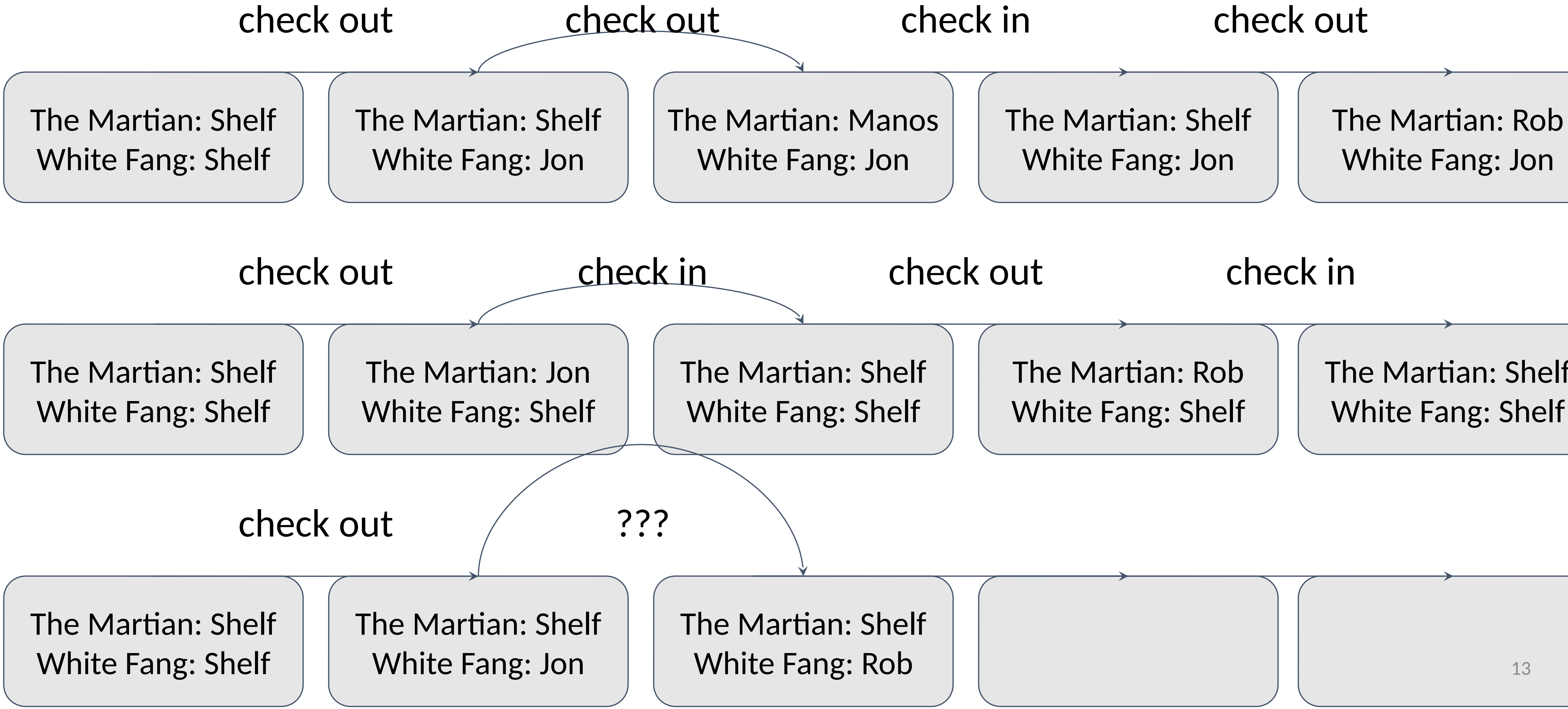
A state is an assignment of values to variables

```
datatype Card = Shelf | Patron(name:
string)
datatype Book = Book(title: string)
type Variables = map<Book, Card>
```

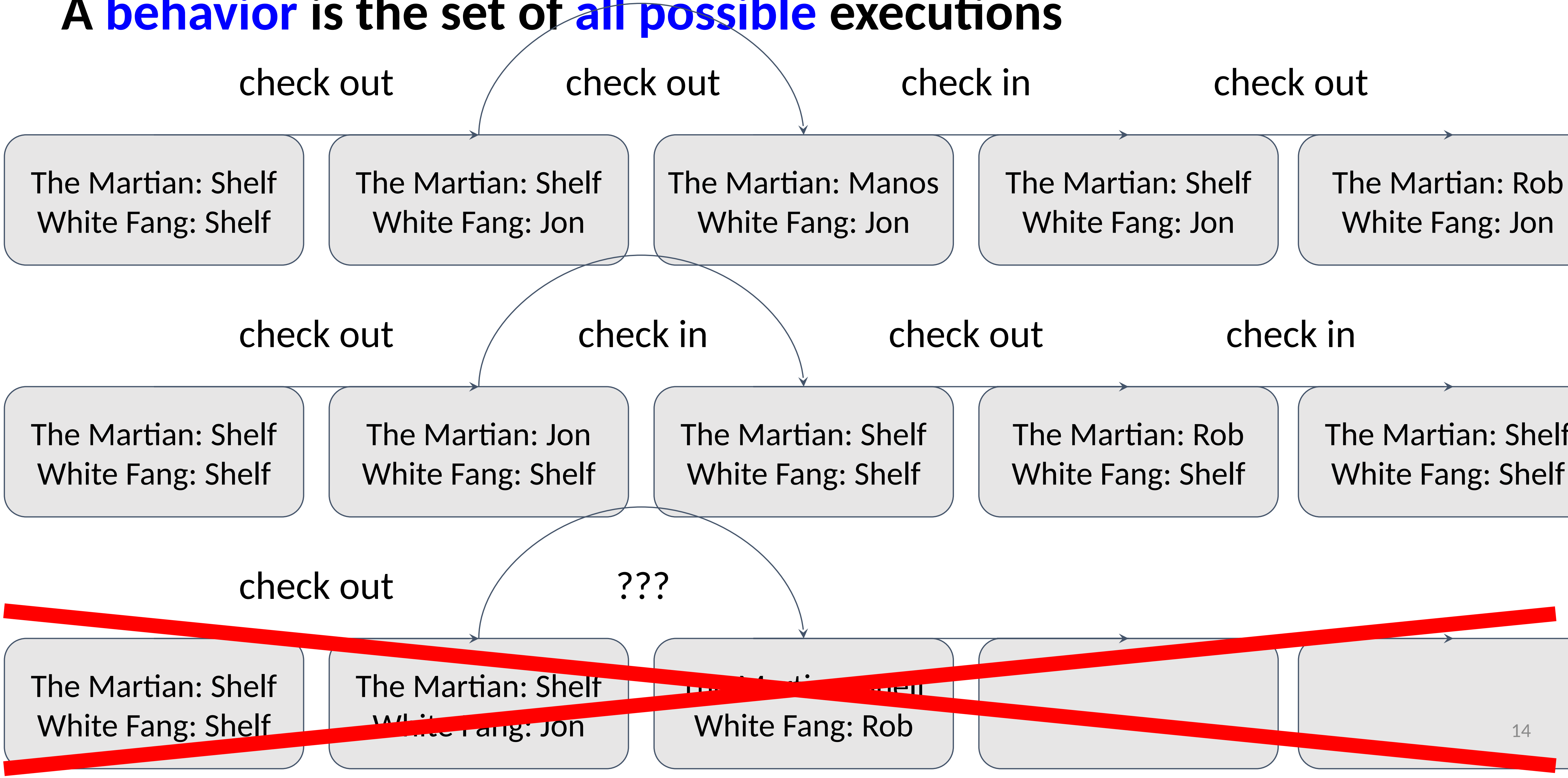
The state space is the set of possible assignments.



An execution is an infinite sequence of states



A behavior is the set of all possible executions



A state machine definition

```
datatype Card = Shelf | Patron(name: string)
datatype Book = Book(title: string)
type Variables = map<Book, Card>
```

```
predicate Init(v: Variables) {
  forall book | book in v :: v[book] == Shelf
}
predicate CheckOut(v : Variables, v' : Variables, book: Book, name: string) {
  && book in v
  && v[book] == Shelf
  && (forall book | book in v :: v[book] != Patron(name))
  && v' == v[book := Patron(name)]
}
predicate CheckIn(v : Variables, v' : Variables, book: Book, name: string) {
  && book in v
  && v[book] == Patron(name)
  && v' == v[book := Shelf]
}
predicate Next(v: Variables, v': Variables) {
  || (exists book, name :: CheckOut(v, v', book, name))
  || (exists book, name :: CheckIn(v, v', book, name))
}
```

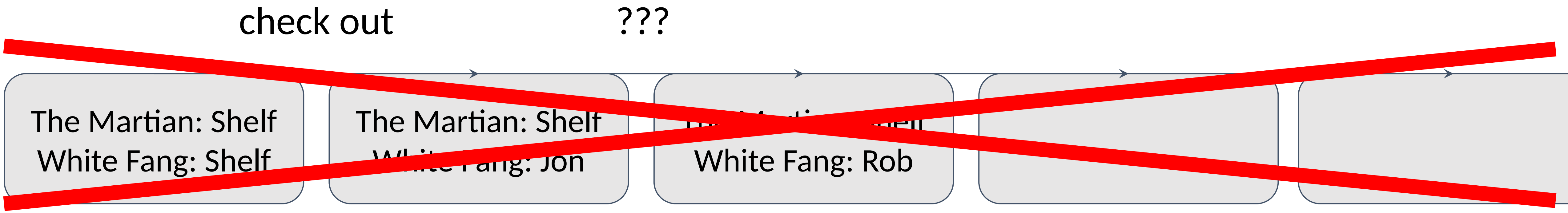
} enabling condition

} “update”

} Nondeterministic definition

A behavior is the set of all possible executions

```
predicate CheckOut(v, v', book, name) {
  && book in v
  && v[book] == Shelf
  && (forall book | book in v :: v[book] !=
Patron(name))
  && v' == v[book := Patron(name)]
}
predicate CheckIn(v, v', book, name) {
  && book in v
  && v[book] == Patron(name)
  && v' == v[book := Shelf]
}
```



How should we define a behavior?

With a **program**?

Its variables define its state space
Its executions define its behavior

Weaknesses:

- concreteness
- nondeterminism
- asynchrony
- environment

How should we define a behavior?

With a **state machine**

Its type defines its state space
Its initial states and transitions define its behavior

State machine strengths

- Abstraction
  - States can be abstract
    - Model an infinite map instead of an efficient pivot table
  - Next predicate is nondeterministic:
    - Implementation may only select some of the choices
    - Can model Murphy’s law (e.g. crash tolerance) or an adversary

State machine strengths

- Abstraction
- Asynchrony
  - Each step of a state machine is conceptually atomic
  - Interleaved steps capture asynchrony: threads, host processes, adversaries
  - Designer decides how precisely to model interleaving; can refine/reduce

State machine strengths

- Abstraction
- Asynchrony
- Environment
  - Model a proposed program with one state machine (verified)
  - Model (adversarial) environment with another (trusted)
  - Compound state machine models their interactions (trusted)