Changing State in the

Language

Jonathan Aldrich, Karl Naden,
Sven Stork, Joshua Sunshine

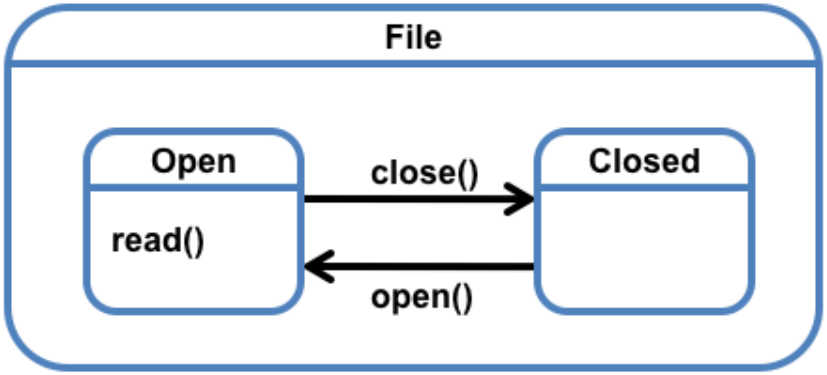
OOPSLA 2011 Demonstration

Carnegie Mellon University
School of Computer Science

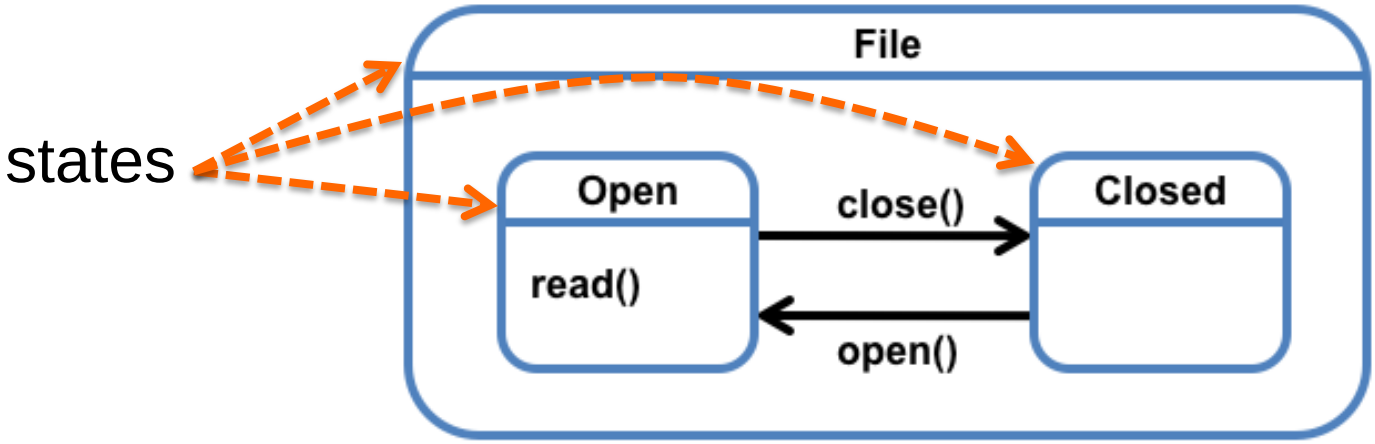
Object Protocols

- An **Object Protocol** dictates an **order** on **method calls**:
  - Has a **finite number** of **abstract states** in which different method calls are valid;
  - Specifies **transitions** between abstract states that occur as a part of some method calls.

- File state chart [harel 87]

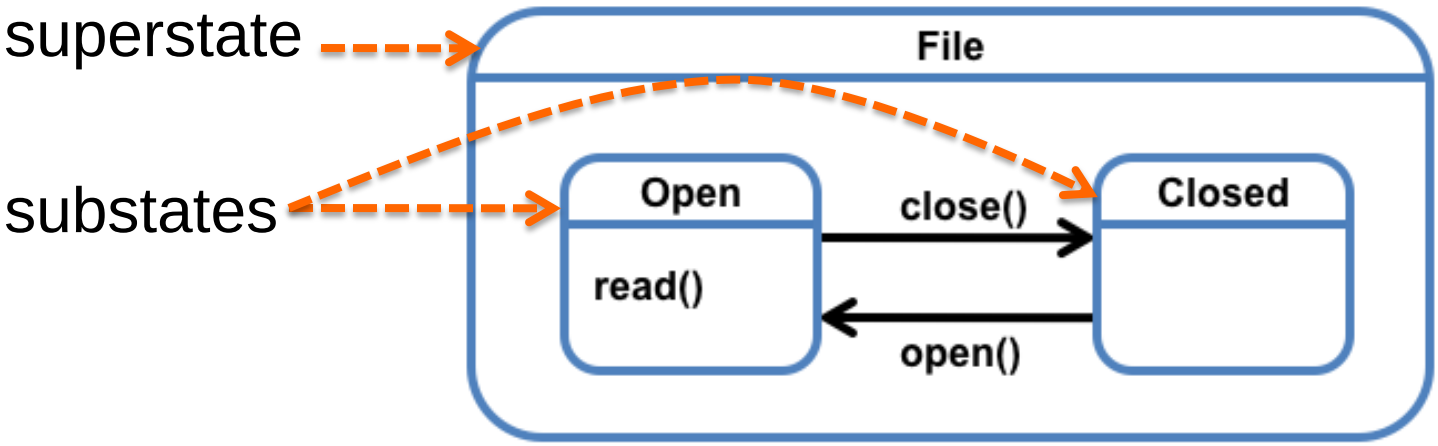


File States



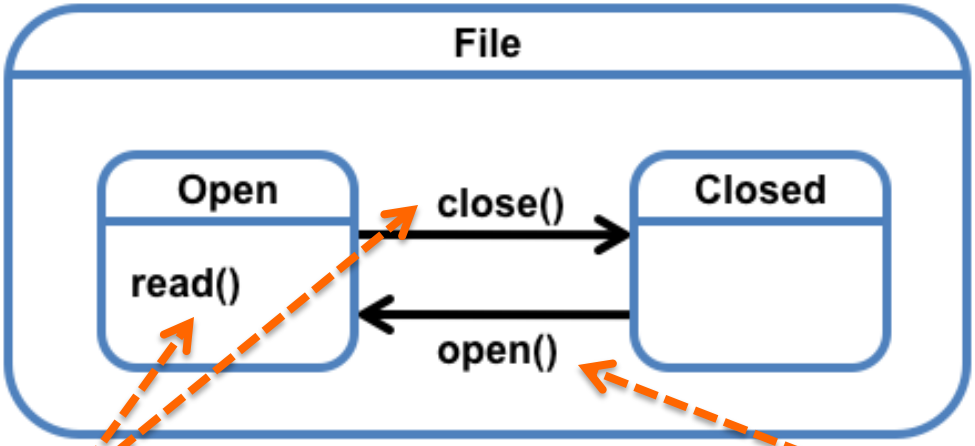
PLAID

File States



PLAID

File States

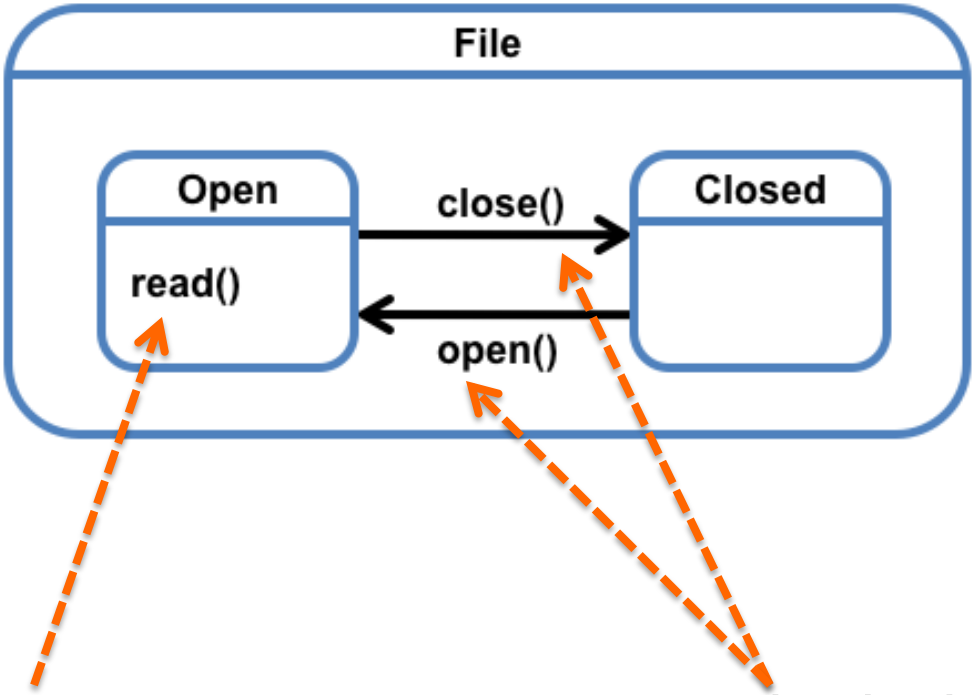


Methods available
in the Open state

Method available
in the Closed state

PLAID

File States

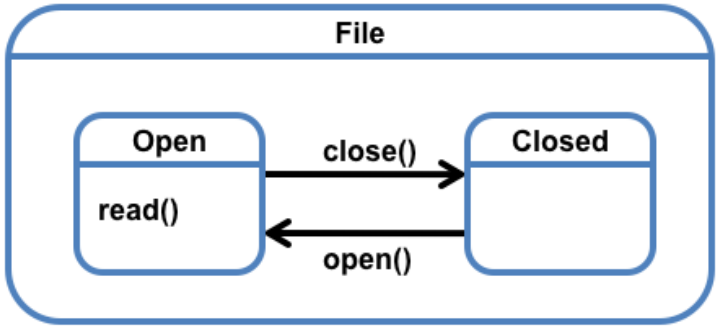


Method that keeps the object in the same state

Methods that transition the state of the object

PLAID

Plaid Syntax

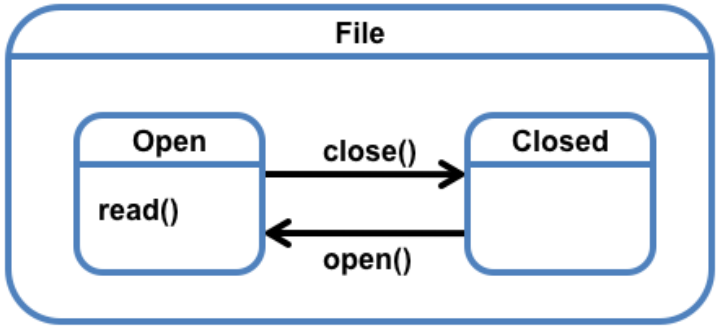


- Need to encode:

PLAID

Code from File.plaid

Plaid Syntax



- Need to encode:
  - States and dimensions

```
state File {

}
```

```
state Open case of File {


}
```

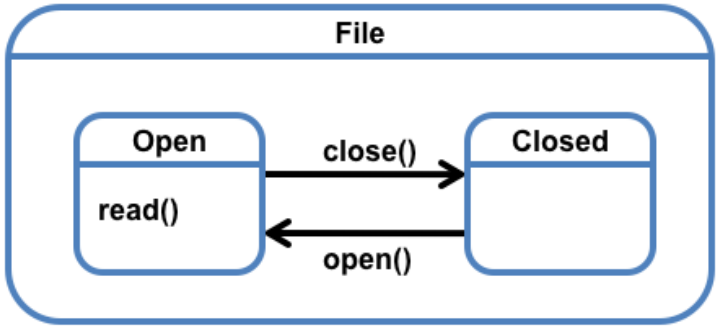
```
state Closed case of File {


}
```

PLAID

Code from File.plaid

Plaid Syntax



- Need to encode:
  - States and dimensions
  - Actions and supporting representation

```
state File {
   val filename;
}
```

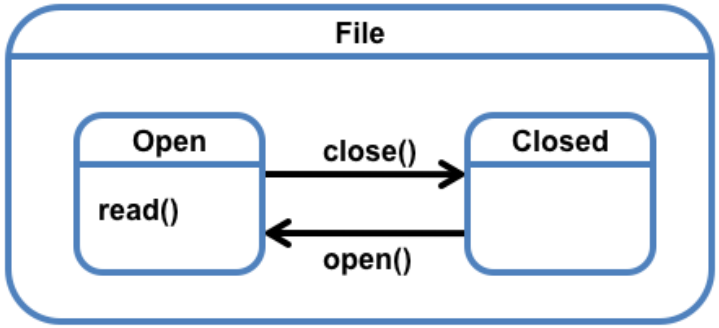
```
state Open case of File {
   val filePtr;
   method read() {…}
   method close() {…}
}
```

```
state Closed case of File {
   method open() {
      …
   }
}
```

PLAID

Code from File.plaid

Plaid Syntax



- Need to encode:
  - States and dimensions
  - Actions and supporting representation
  - Transitions

```
state File {
   val filename;
}
```

```
state Open case of File {
   val filePtr;
   method read() {…}
   method close() { this ← Closed; }
}
```

```
state Closed case of File {
   method open() {
      this ← Open { val filePtr = … };
   }
}
```

PLAID

Code from File.plaid

Using Files

- Instantiation:

```
val fs = new Closed { val filename = “path.to.file”; };
```

PLAID

Code from File.plaid

Using Files

- Matching:

```
method readAndCloseFile(theFile) {

}
```

PLAID

Code from File.plaid

Using Files

- Matching:

```
method readAndCloseFile(theFile) {
  match (theFile) { /* make sure the file is Open */
    case Closed { theFile.open() }
    case Open { /* no op – already Open */ }
  };

}
```

PLAID

Code from File.plaid

Using Files

- Matching:

```
method readAndCloseFile(theFile) {
  match (theFile) { /* make sure the file is Open */
    case Closed { theFile.open() }
    case Open { /* no op – already Open */ }
  };
  val ret = theFile.read();


}
```

PLAID

Code from File.plaid

Using Files

- Matching:

```
method readAndCloseFile(theFile) {
  match (theFile) { /* make sure the file is Open */
    case Closed { theFile.open() }
    case Open { /* no op – already Open */ }
  };
  val ret = theFile.read();
  theFile.close();

}
```

PLAID

Code from File.plaid

Using Files

- Matching:

```
method readAndCloseFile(theFile) {
  match (theFile) { /* make sure the file is Open */
    case Closed { theFile.open() }
    case Open { /* no op – already Open */ }
  };
  val ret = theFile.read();
  theFile.close();
  ret
}
```

PLAID

Code from File.plaid

Conditionals

- If functions: (

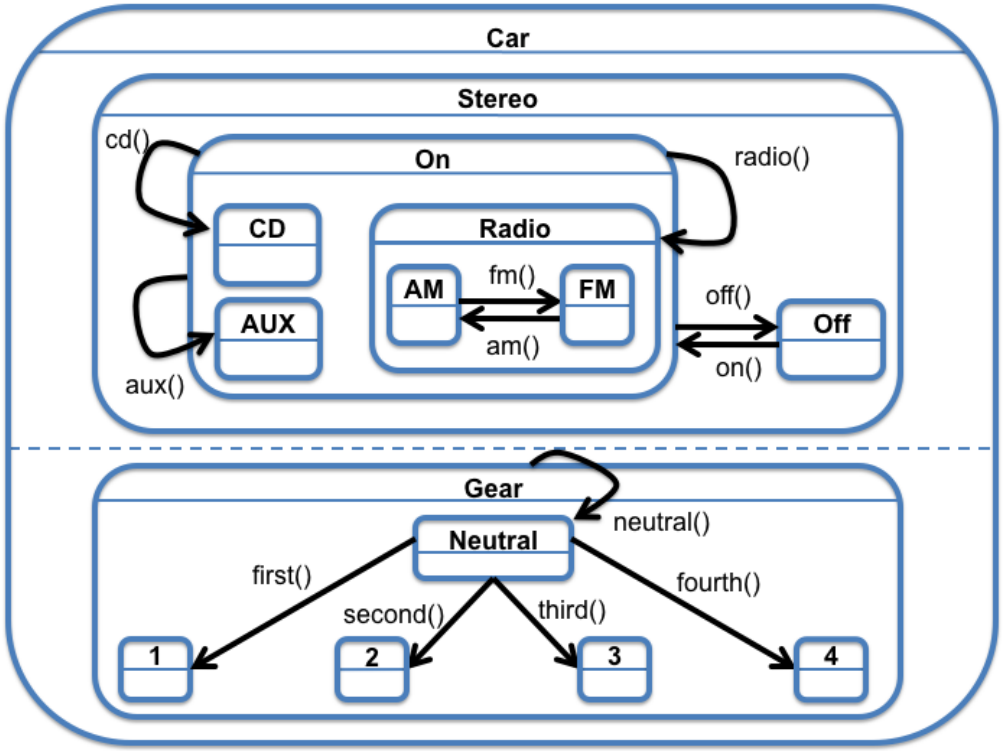
```
method readTwiceAndAdd(openFile) {
  var result = openFile.read();
  val second = openFile.read();
  if (second != -1) { //only add if not EOF symbol
    result = result + second;
  };
  result
}
```

PLAID

Code from File.plaid

And-states and Composition

- A Car has a more complicated state chart



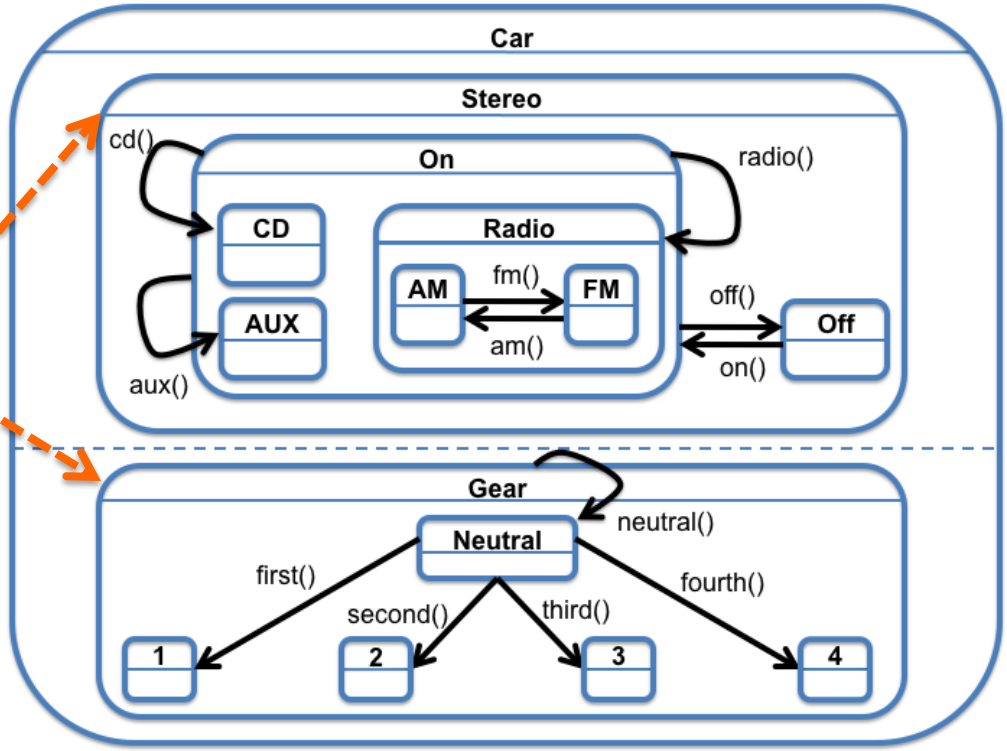
PLAID

And-states and Composition

- A Car has a more complicated state chart

Two independent dimensions within Car

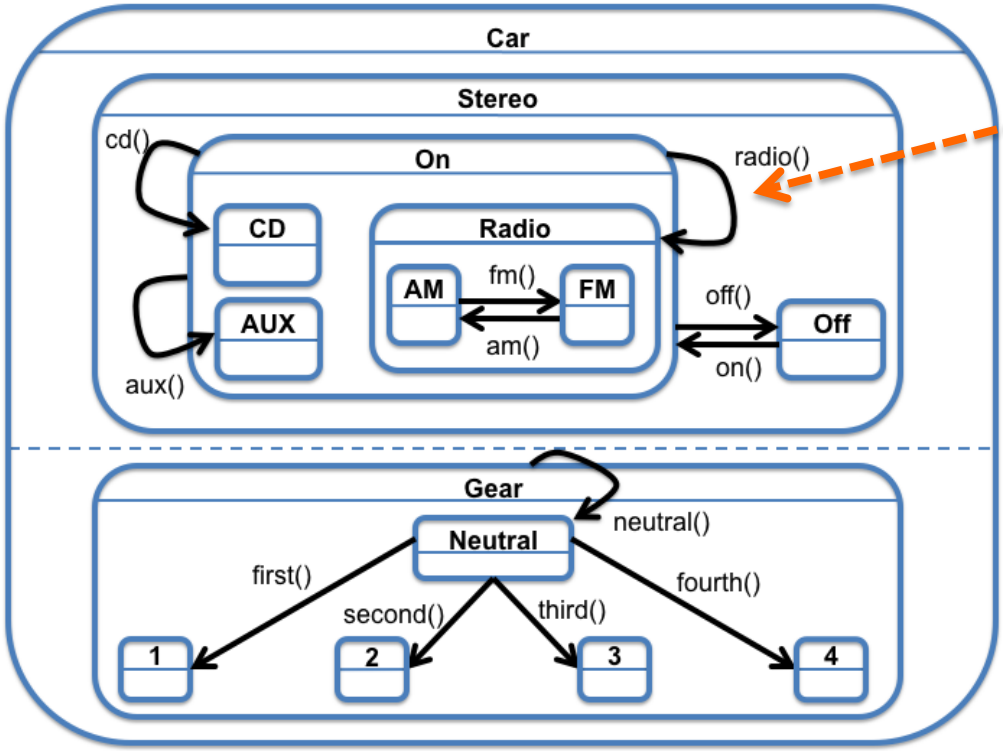
Changing a Gear does not impact the stereo state



PLAID

And-states and Composition

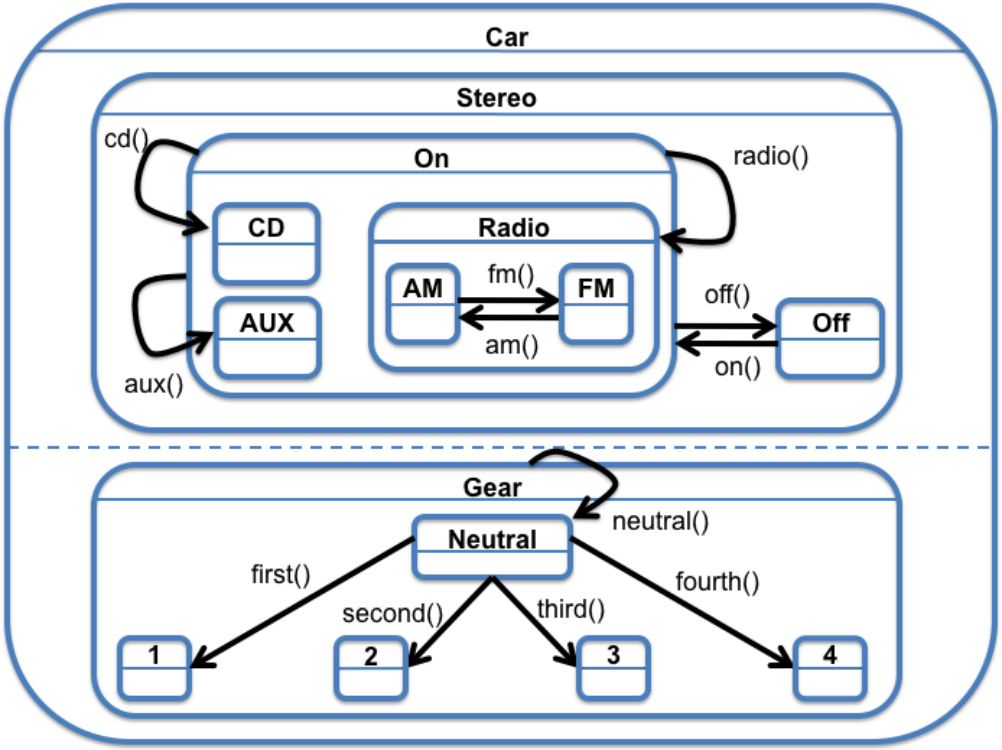
- A Car has a more complicated state chart



radio() method available in all substates of On. Always ends in the Radio state.

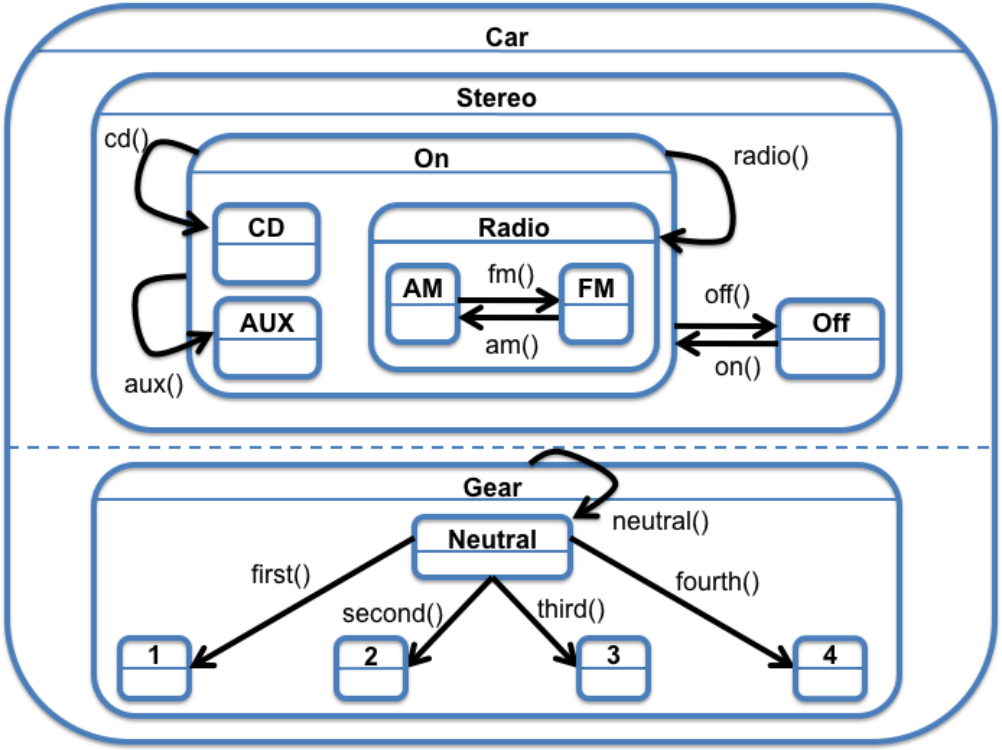
PLAID

Plaid Car



Code from Car.plaid

Plaid Car

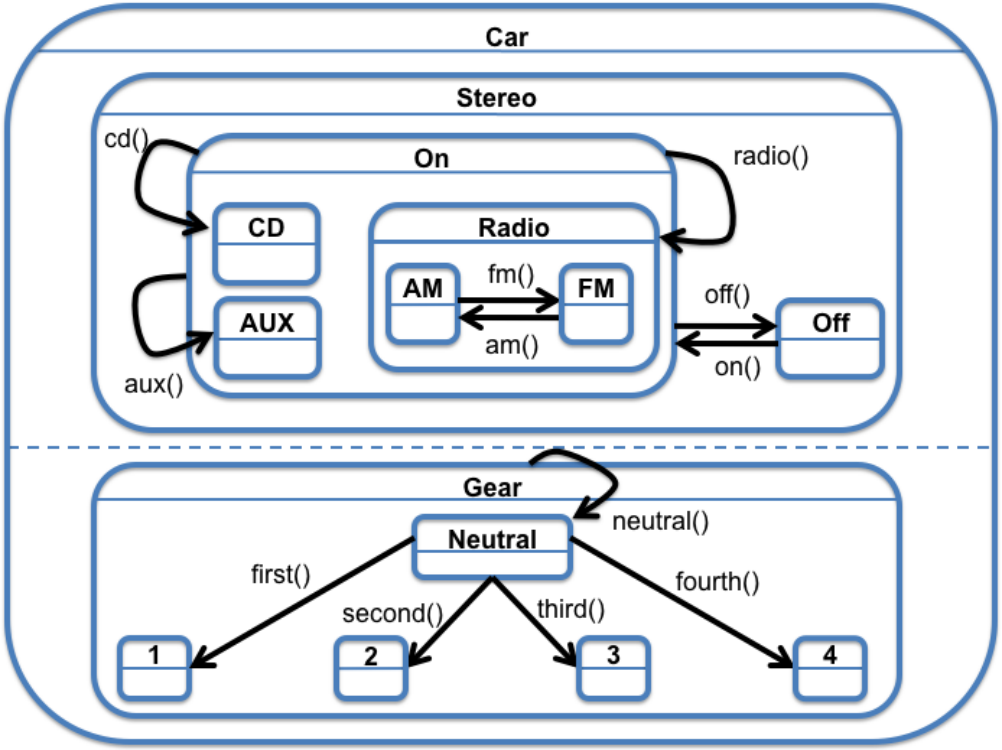



```
state Gear {
  method neutral() { this ← Neutral }
}
```

PLAID

Code from Car.plaid

Plaid Car



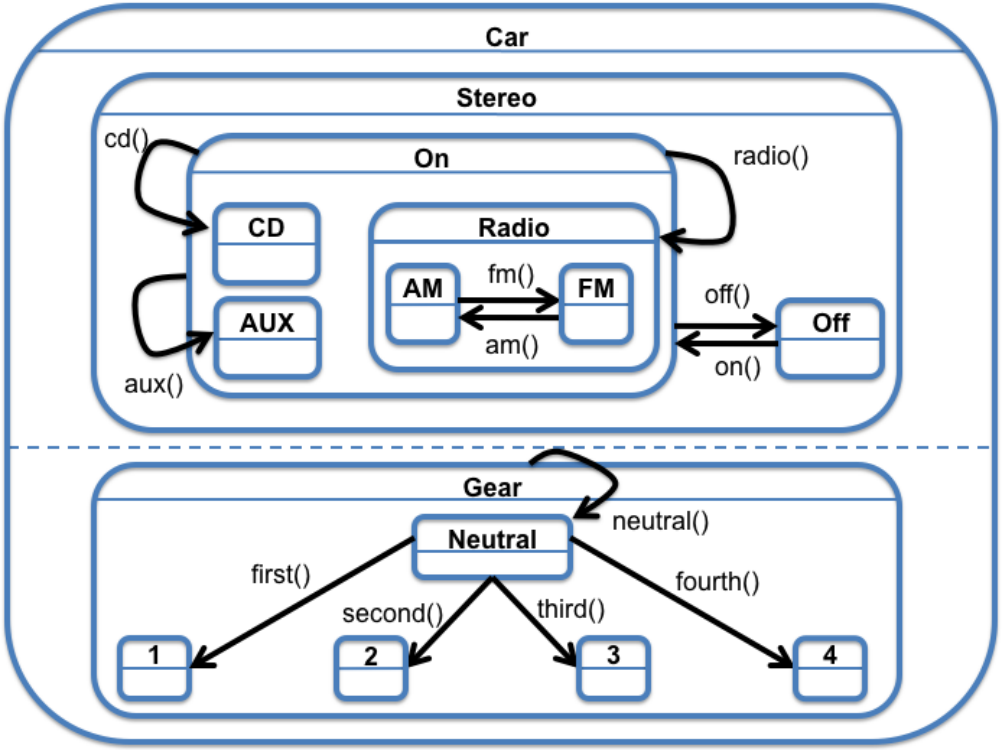

```
state Gear {
  method neutral() { this ← Neutral }
}
```

```
state 1 case of Gear { }
state 2 case of Gear { }
...
```

PLAID

Code from Car.plaid

Plaid Car




```
state Gear {
  method neutral() { this ← Neutral }
}
```

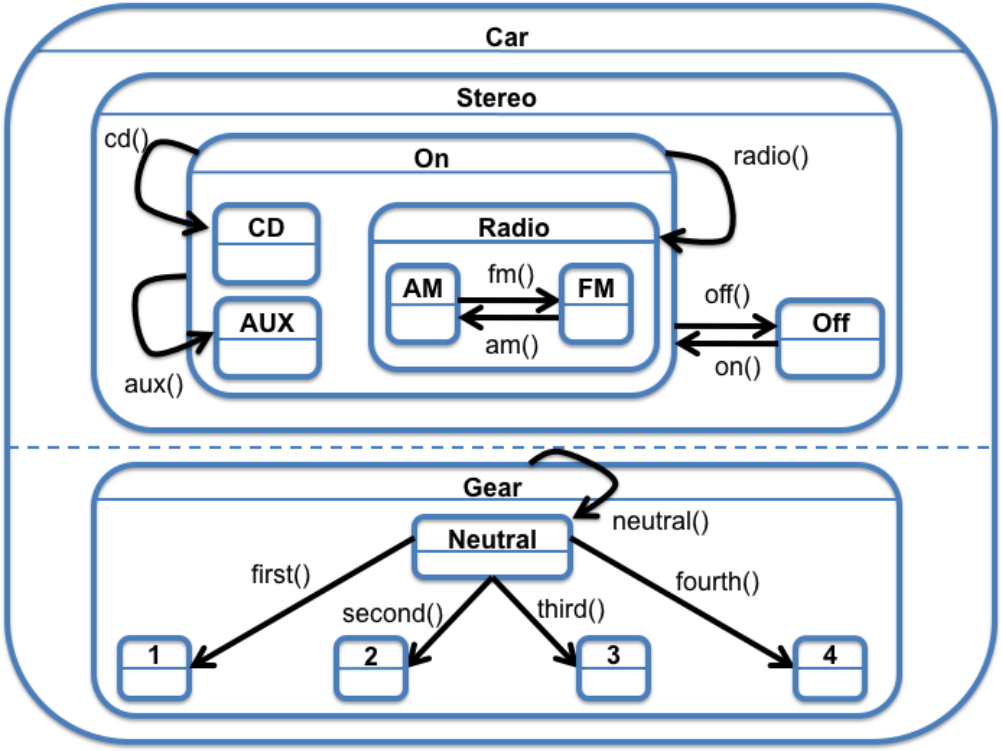
```
state 1 case of Gear { }
state 2 case of Gear { }
...
```

```
state Neutral case of Gear {
  method first() { this ← 1 } ...
}
```

PLAID

Code from Car.plaid

Plaid Car




```
state Gear {
  method neutral() { this ← Neutral }
}
```

```
state 1 case of Gear { }
state 2 case of Gear { }
...
```

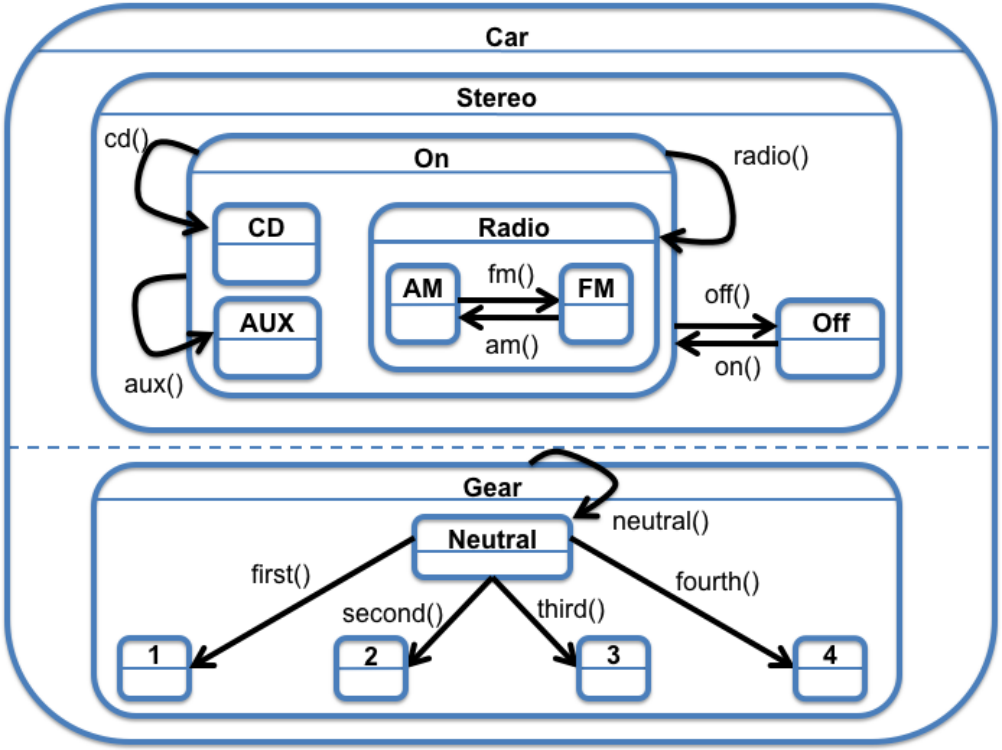
```
state Neutral case of Gear {
  method first() { this ← 1 } ...
}
```

```
state Car = Stereo with Gear;
```

PLAID

Code from Car.plaid

Plaid Car




```
state Gear {
  method neutral() { this ← Neutral }
}
```

```
state 1 case of Gear { }
state 2 case of Gear { }
...
```

```
state Neutral case of Gear {
  method first() { this ← 1 } ...
}
```

```
state Car = Stereo with Gear;
```

Composition

Code from Car.plaid

PLAID

Questions?

Let’s write some code!

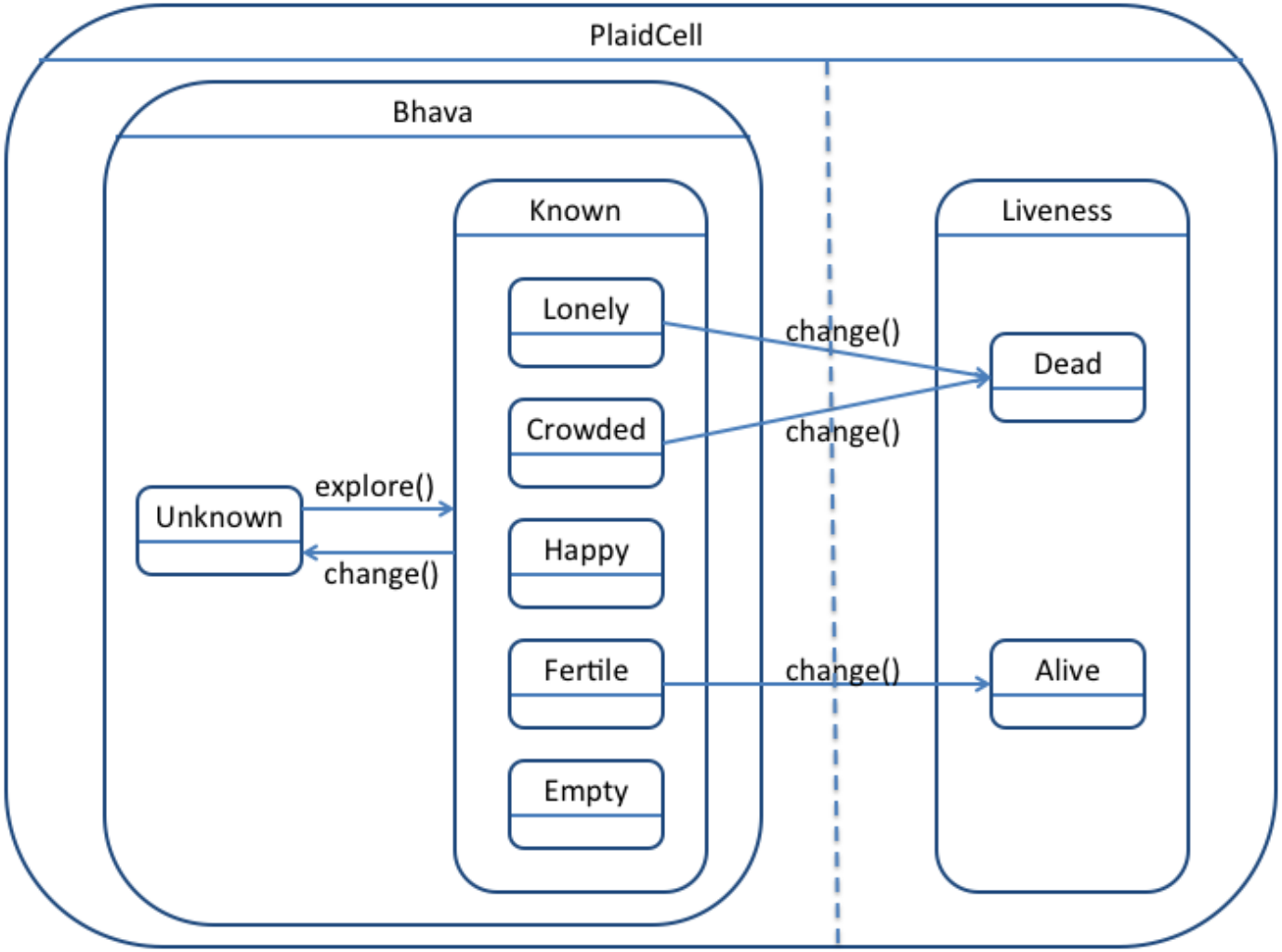
PLAID

Game Of Life Rules

For each step, iterate over each cell and

- If the cell is alive and
  - Has 0 or 1 Alive neighbors, it dies from Loneliness
  - Has 2 or 3 Alive neighbors, it is Happy and stays alive
  - Has 4 or more Alive neighbors, it dies from Overcrowding
- If the cell is dead and
  - Has 2 or fewer Alive neighbors, it remains Dead
  - Has 3 or more Alive neighbors, it is Fertile and becomes Alive

Game Of Life State Chart



PLAID