

# First-Class State Change in

# PLAiD

**Karl Naden**

with Joshua Sunshine, Sven Stork

Jonathan Aldrich, and Éric Tanter

OOPSLA 10/27/2011



**Carnegie Mellon University**  
School of Computer Science

# States and State Change

- Things all around us are changing state
  - Butterfly: Egg → Caterpillar → Chrysalis → Imago

# States and State Change

- Things all around us are changing state
  - Butterfly: Egg → Caterpillar → Chrysalis → Imago
  - You: Awake ↔ Asleep

# States and State Change

- Things all around us are changing state
  - Butterfly: Egg → Caterpillar → Chrysalis → Imago
  - You: Awake ↔ Asleep
- In Programming, too
  - File: Open ↔ Closed

# States and State Change

- Things all around us are changing state
  - Butterfly: Egg → Caterpillar → Chrysalis → Imago
  - You: Awake ↔ Asleep
- In Programming, too
  - File: Open ↔ Closed
  - Java Exception: Cause Not Set → Cause Set

# States and State Change

- Things all around us are changing state
  - Butterfly: Egg → Caterpillar → Chrysalis → Imago
  - You: Awake ↔ Asleep
- In Programming, too
  - File: Open ↔ Closed
  - Java Exception: Cause Not Set → Cause Set
- Different abilities depending on the state
  - Butterfly can only fly as an Imago
  - read() only available in the Open state of File

# States and State Change

- Things all around us are changing state
  - Butterfly: Egg → Caterpillar → Chrysalis → Imago
  - You: Awake ↔ Asleep
- In Programming, too
  - File: Open ↔ Closed
  - Java Exception: Cause Not Set → Cause Set
- Different abilities depending on the state
  - Butterfly can only fly as an Imago
  - read() only available in the Open state of File
- Error when the current state does not support the action

# Describing Objects with State

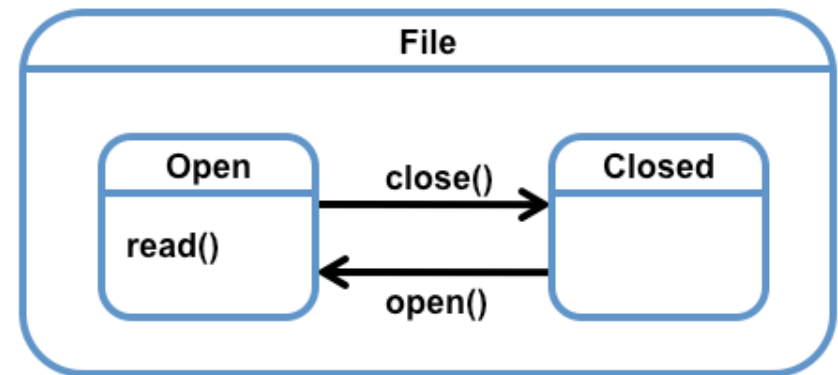


# Describing Objects with State

- An **Object Protocol** [Strom, Yemini '86] 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.

# Describing Objects with State

- An **Object Protocol** [Strom, Yemini '86] 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.
- State Charts [Harel '87]: **File**
  - States: File, Open, Closed
  - Methods: read(), open(), closed()
  - Transitions: close(), open()



# Existing Support for States

# Existing Support for States

- States exists

- In documentation

```
/** @throws IllegalStateException if task was already scheduled or  
 *      cancelled, timer was cancelled, or timer thread terminated.  
 */  
private void sched(TimerTask task, long time, long period) { ... }
```

# Existing Support for States

- States exists

- In documentation

```
/** @throws IllegalStateException if task was already scheduled or  
 *      cancelled, timer was cancelled, or timer thread terminated.  
 */  
private void sched(TimerTask task, long time, long period) { ... }
```

- Encoded as lower-level constructs

```
if (task.state != TimerTask.VIRGIN) throw new IllegalStateException(...);
```

# Existing Support for States

- States exists

- In documentation

```
/** @throws IllegalStateException if task was already scheduled or  
 *     cancelled, timer was cancelled, or timer thread terminated.  
 */  
private void sched(TimerTask task, long time, long period) { ... }
```

- Encoded as lower-level constructs

```
if (task.state != TimerTask.VIRGIN) throw new IllegalStateException(...);
```

- Problems

- States from design are obfuscated

- Code difficult to understand

- If checks forgotten, results difficult to debug, e.g.

- Non-specific **NullPointerException**

- Data corruption (overwritten **TimerTask**)

# Common and Complex

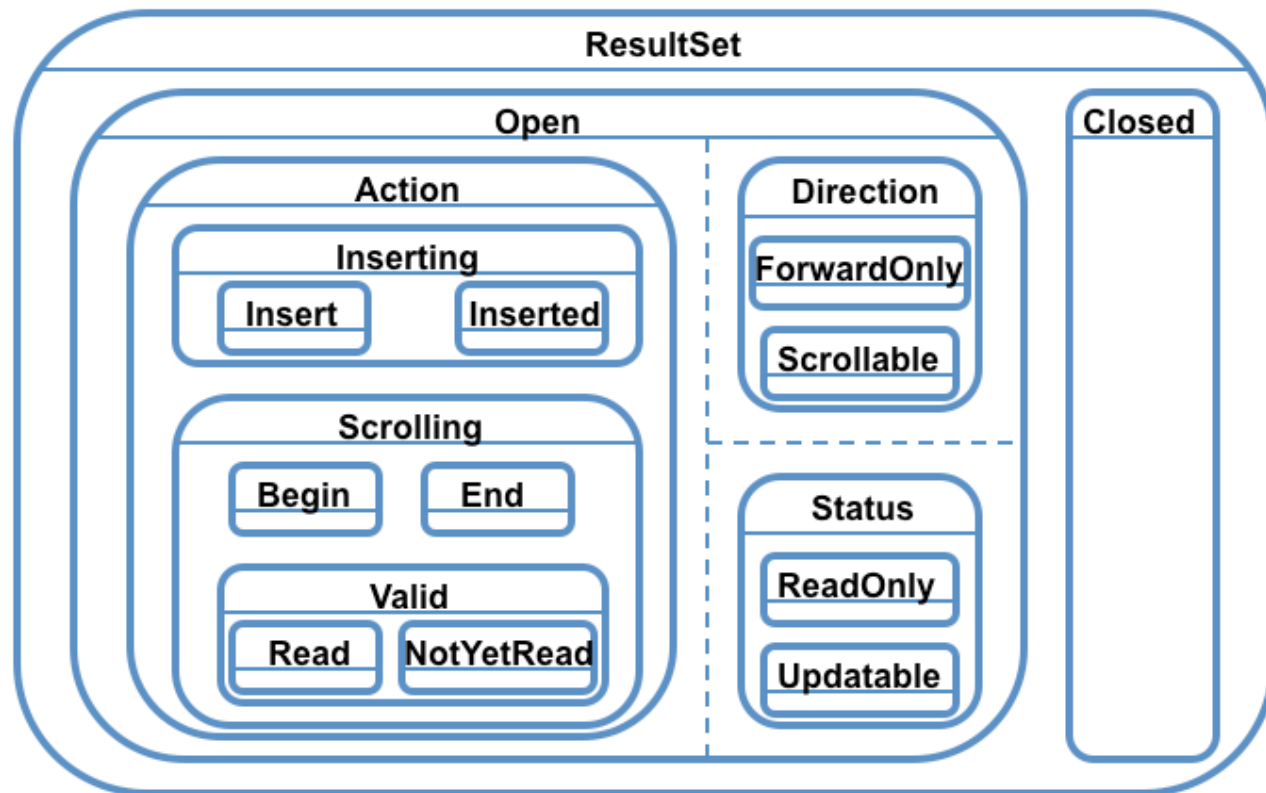
# Common and Complex

- Common [Beckman '11]
  - 7% of Java classes define object protocols
    - 3x as many as define generics
  - 13% use them



# Common and Complex

- Common [Beckman '11]
  - 7% of Java classes define object protocols
    - 3x as many as define generics
  - 13% use them
- Complex:



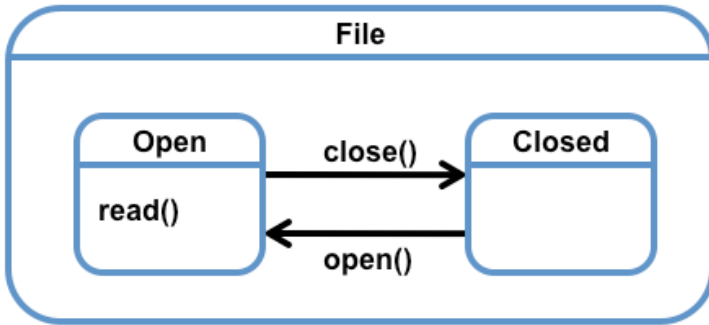
# Overview

- **PLAiD** programming language
  - First class states and transitions

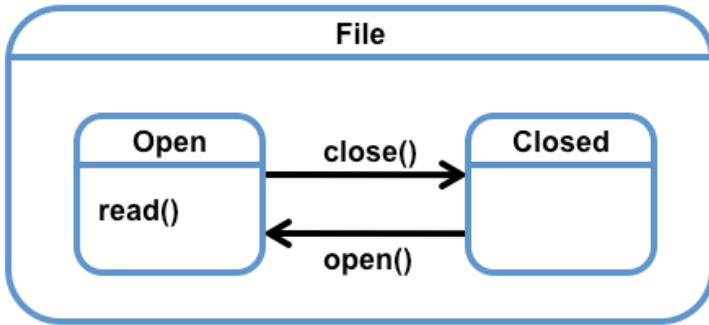
# Overview

- **PLAiD** programming language
  - First class states and transitions
- Overview
  - Syntax and semantics of states and transitions
  - Trait-based reuse

# Two Encodings of File



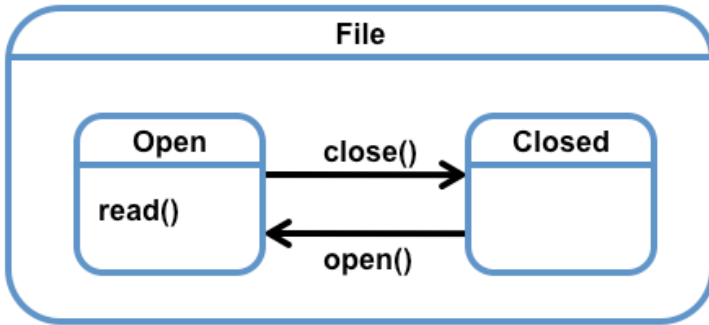
# Two Encodings of File



## 1) States

```
class File {  
  
    private FileResource filePtr = null;  
  
}
```

# Two Encodings of File



## 1) States

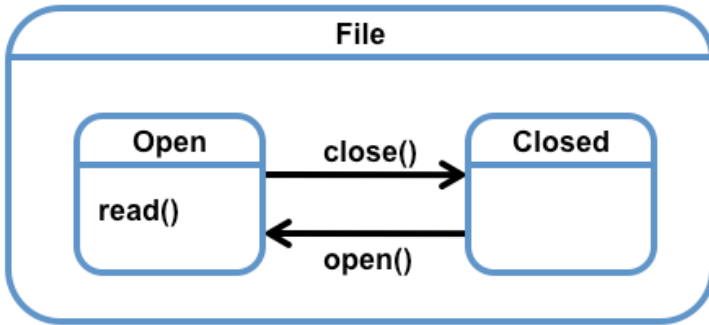
```
class File {
```

```
    private FileResource filePtr = null;
```

substate of file determined  
by **null**-ness of filePtr field

```
}
```

# Two Encodings of File



```
class File {  
  
    private FileResource filePtr = null;  
  
}
```

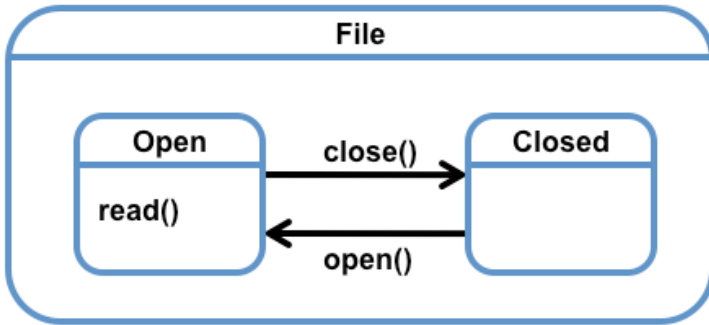
## 1) States

```
state File {  
  
}
```

```
state Open case of File {  
  
}
```

```
state Closed case of File {  
  
}
```

# Two Encodings of File



```
class File {  
  
    private FileResource filePtr = null;  
    public int read() {...}  
  
    public void close() {...}  
    public void open() {...}  
  
}
```

## 2) Methods

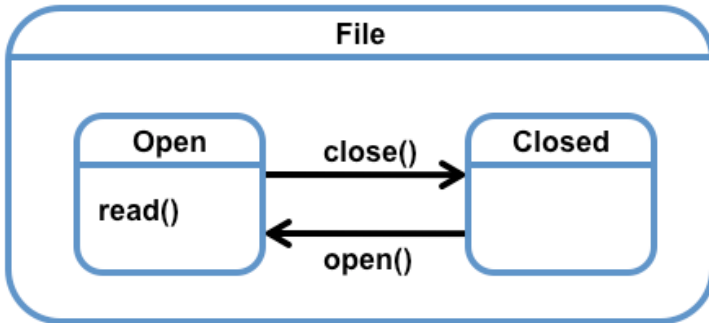
```
state File {  
  
}
```

```
state Open case of File {  
  
}
```

```
state Closed case of File {  
  
}
```



# Two Encodings of File



```
class File {  
  
    private FileResource filePtr = null;  
    public int read() {...}  
  
    public void close() {...}  
    public void open() {...}  
  
}
```

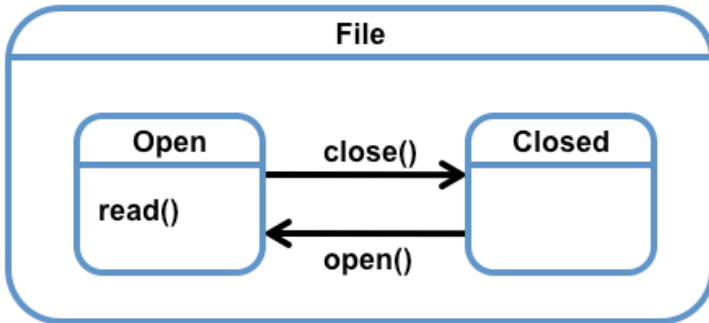
## 2) Methods

```
state File {  
  
}
```

```
state Open case of File {  
  
    method read() {...}  
    method close() {...}  
  
}
```

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

# Two Encodings of File



```
class File {  
  
    private FileResource filePtr = null;  
    public int read() {  
        if (filePtr == null)  
            throw new IOException  
        else ...  
    }  
    public void close() {...}  
    public void open()  
        { if (filePtr == null) {...} }  
}
```

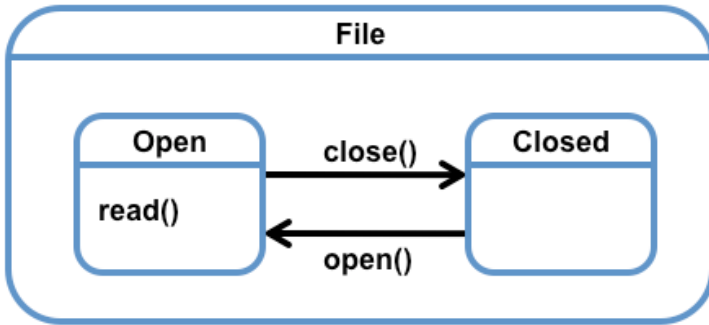
## 2) Methods

```
state File {  
  
}
```

```
state Open case of File {  
  
    method read() {...}  
    method close() {...}  
}
```

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

# Two Encodings of File



```
class File {  
  private String filename;  
  private FileResource filePtr = null;  
  public int read() {  
    if (filePtr == null)  
      throw new IOException  
    else ...  
  }  
  public void close() {...}  
  public void open()  
    { if (filePtr == null) {...} }  
}
```

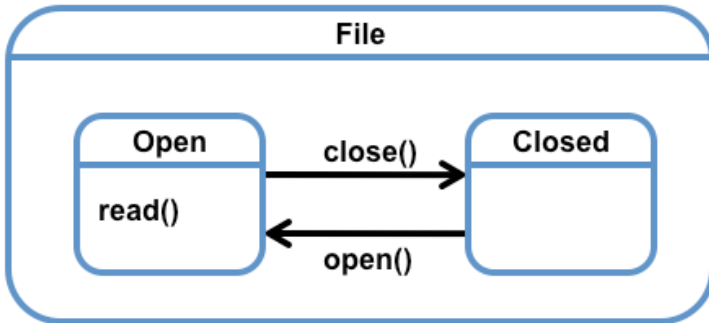
## 3) Representation

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

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

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

# Two Encodings of File



```
class File {  
  private String filename;  
  private FileResource filePtr = null;  
  public int read() {  
    if (filePtr == null)  
      throw new IOException  
    else ...  
  }  
  public void close() {...; filePtr = null;}  
  public void open()  
    { if (filePtr == null) { filePtr = ... } }  
}
```

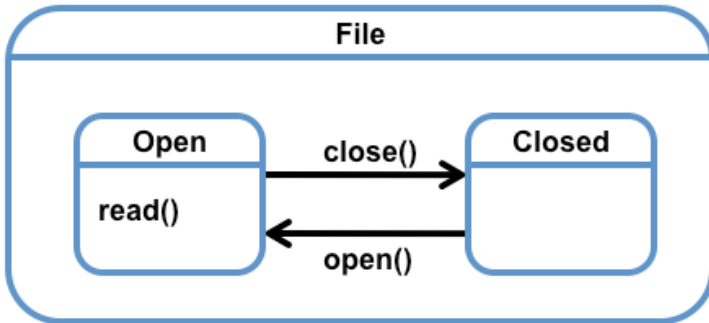
## 4) Transitions

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

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

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

# Two Encodings of File



```
class File {  
  private String filename;  
  private FileResource filePtr = null;  
  public int read() {  
    if (filePtr == null)  
      throw new IOException  
    else ...  
  }  
  public void close() {...; filePtr = null;}  
  public void open()  
    { if (filePtr == null) { filePtr = ... } }  
}
```

## 4) 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 = ... };  
  }  
}
```

# Comparison

- **PLAiD** encoding advantages:
  - Design salient in the code
  - Succinct: fewer explicit checks
  - Errors handled safely and informatively

# Reuse and Composition

- **Goal:** flexibility of Traits [Ducasse '06] with benefits of protocols
  - Break protocols up into elemental protocols
  - Compose them

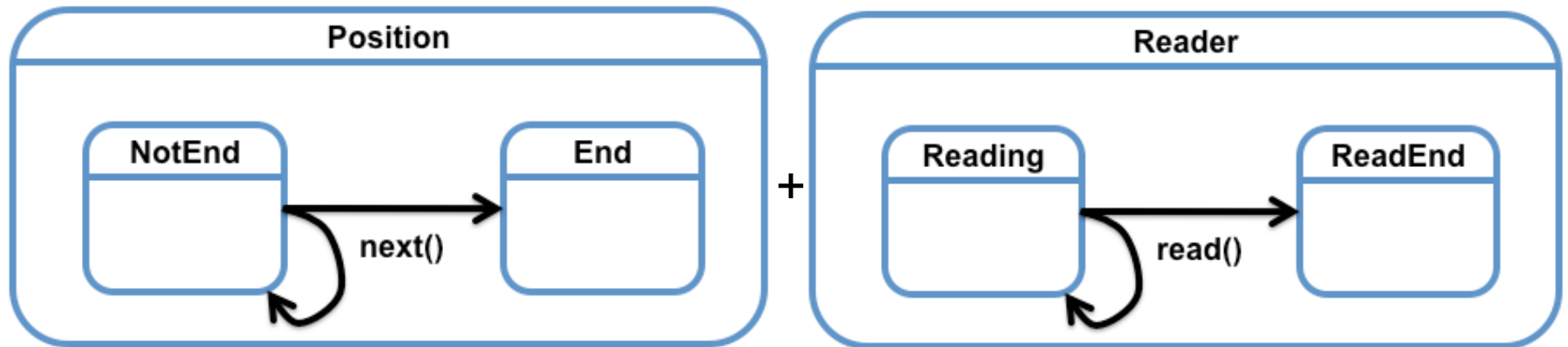
# Reuse and Composition

- `ReadStream = Position + Reader`



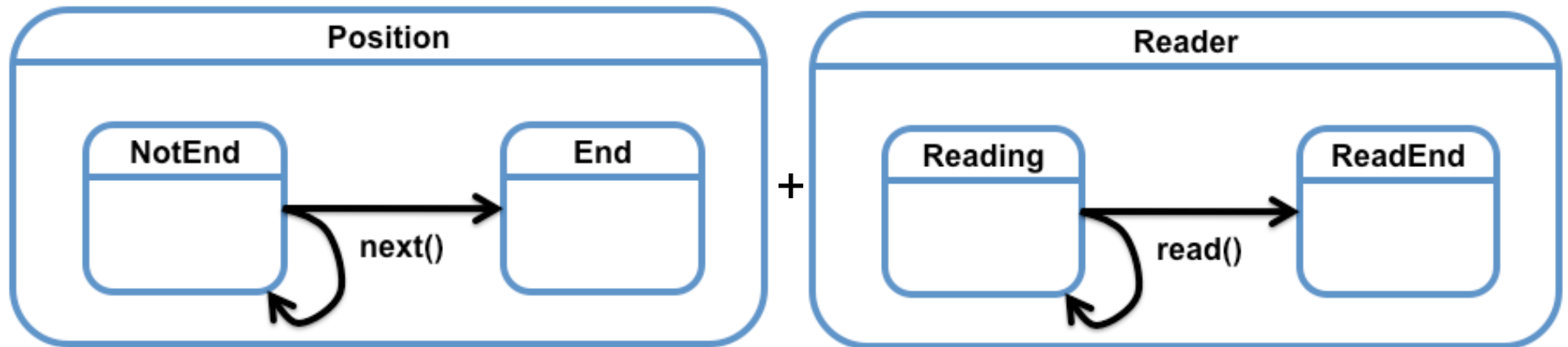
# Reuse and Composition

- ReadStream = Position + Reader



# Reuse and Composition

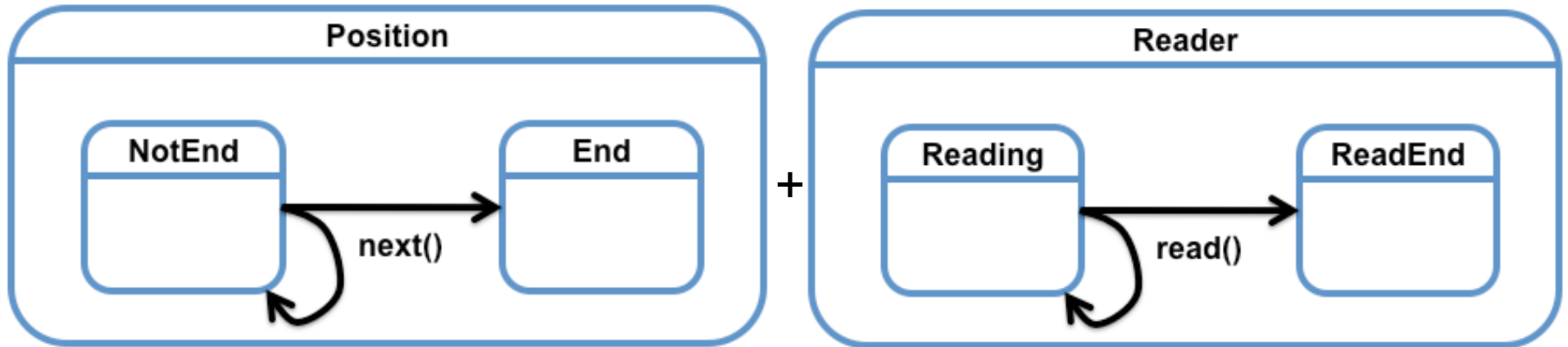
- ReadStream = Position + Reader



Protocols in two separate **dimensions**

# Reuse and Composition

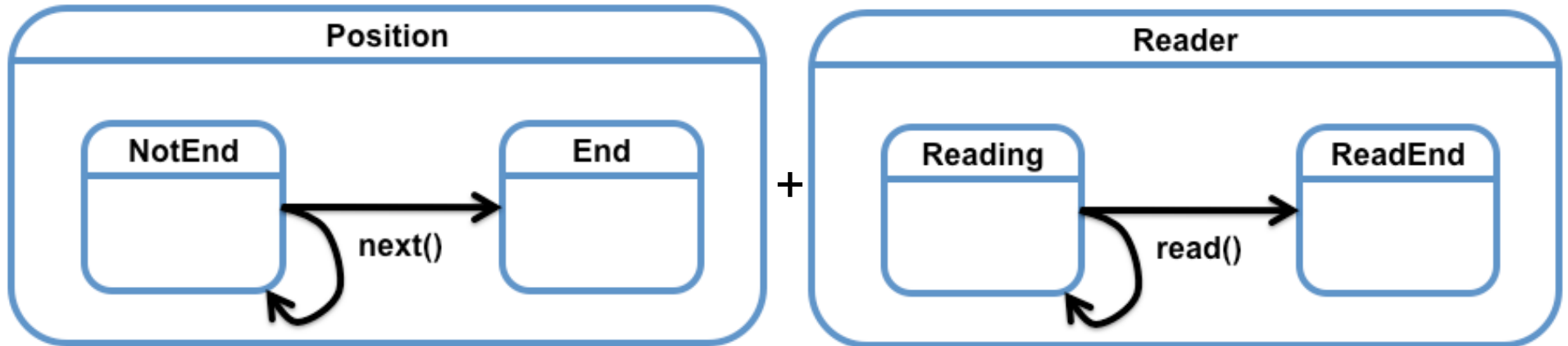
- ReadStream = Position + Reader



```
state Reading {  
  method read() {  
    /* read character */  
    this.next();  
  }  
}
```

# Reuse and Composition

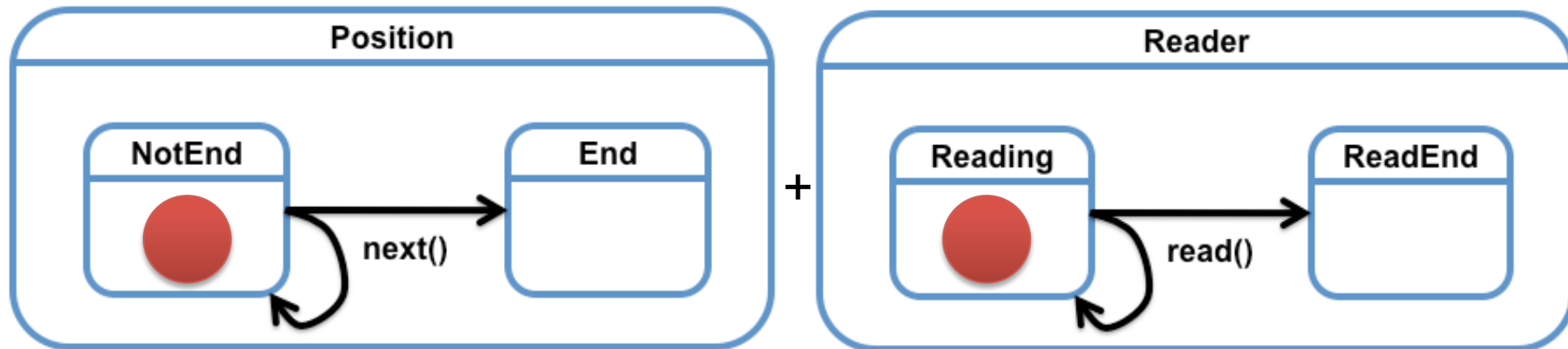
- ReadStream = Position + Reader



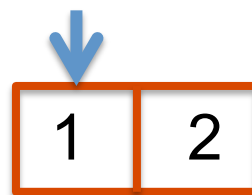
```
state NotEnd case of Position {  
  method next() {  
    /* move position forward */  
    if (/* at the end */) {  
      this ← End  
    }  
  }  
}
```

```
state Reading {  
  method read() {  
    /* read character */  
    this.next();  
  }  
}
```

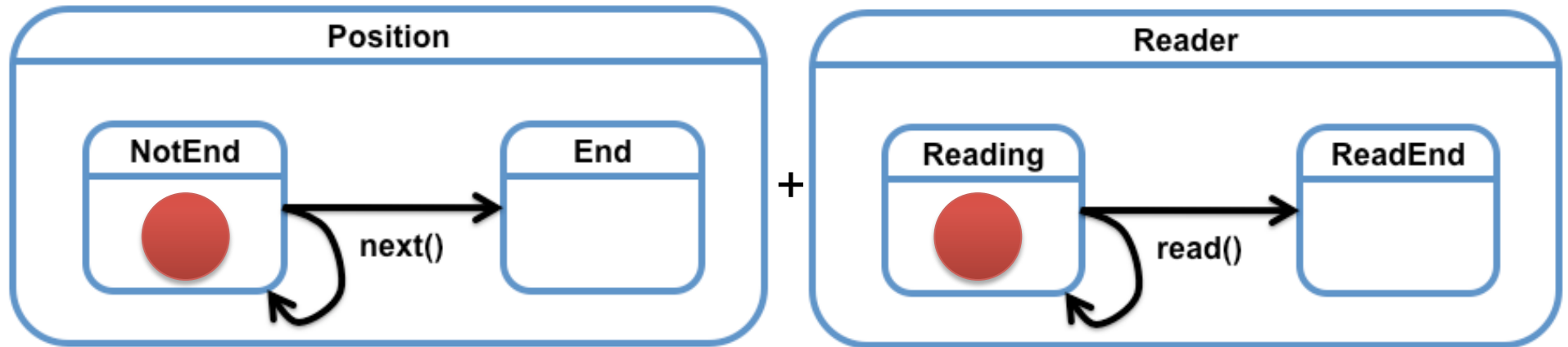
# Reuse and Composition



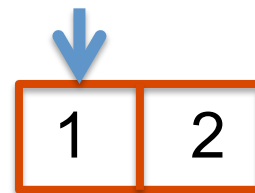
```
val rs = new Reading with  
  NotEnd { val coll = [1,2]; ... };
```



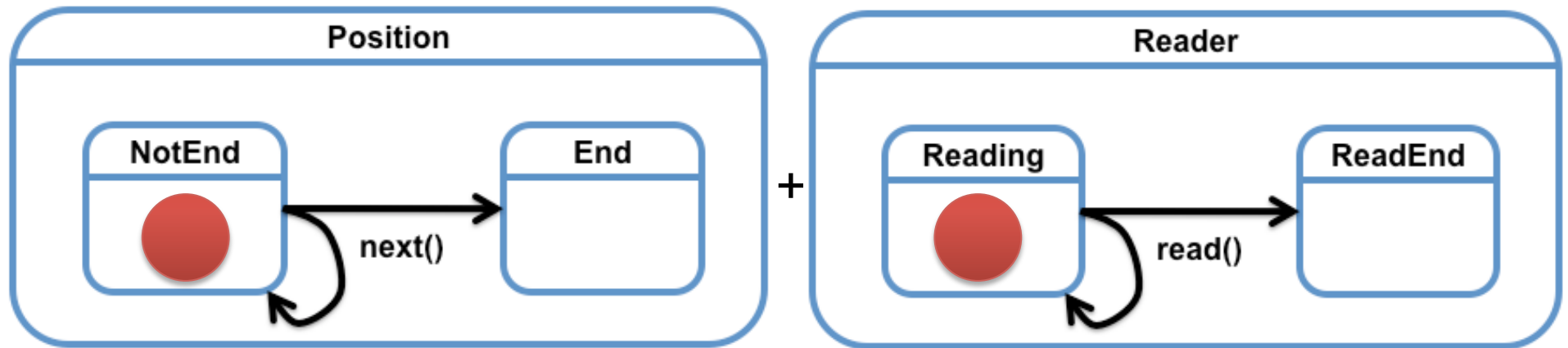
# Reuse and Composition



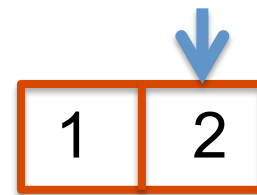
```
val rs = new Reading with  
    NotEnd { val coll = [1,2]; ... };  
rs.read();
```



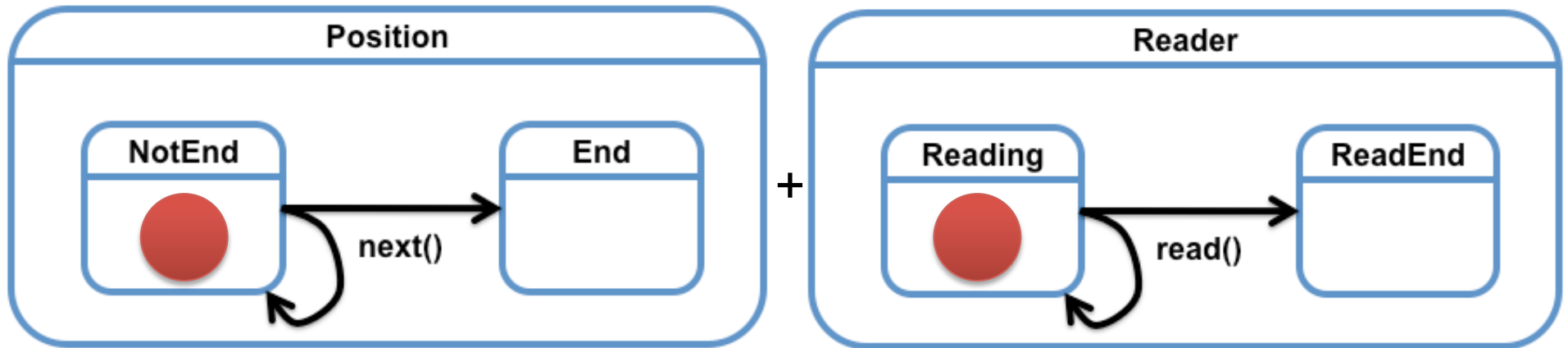
# Reuse and Composition



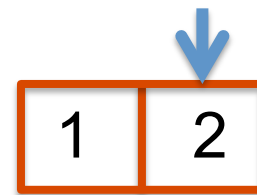
```
val rs = new Reading with  
  NotEnd { val coll = [1,2]; ... };  
rs.read(); //no state change
```



# Reuse and Composition

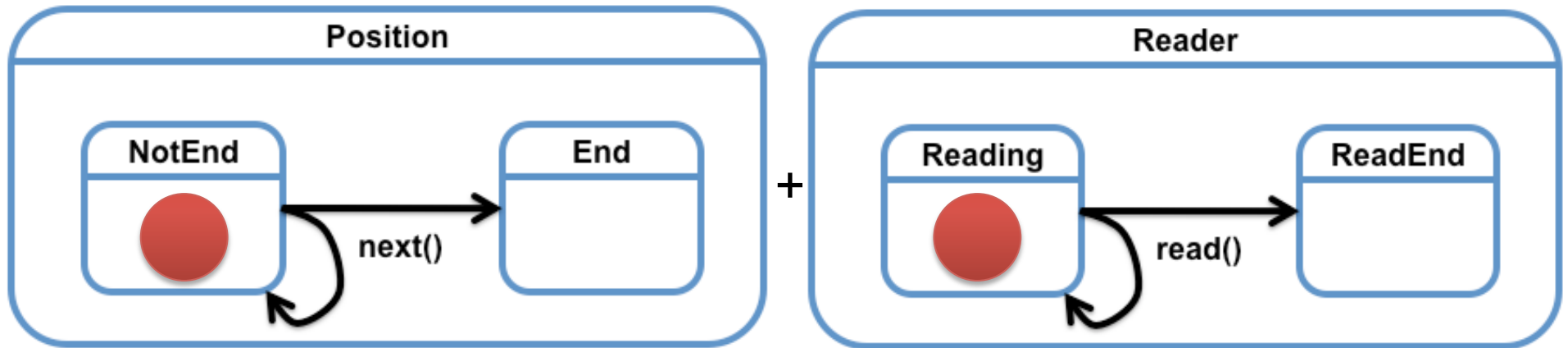


```
val rs = new Reading with  
    NotEnd { val coll = [1,2]; ... };  
rs.read(); //no state change  
rs.read();
```

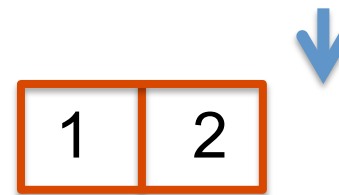




# Reuse and Composition

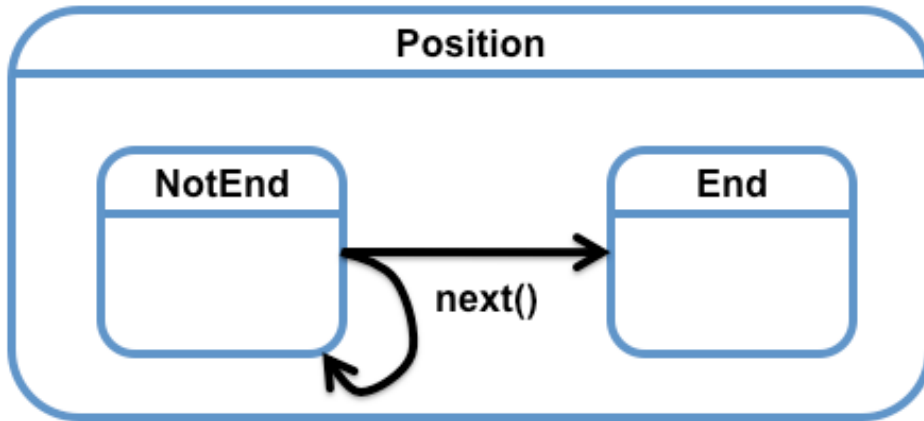


```
val rs = new Reading with  
    NotEnd { val coll = [1,2]; ... };  
rs.read(); //no state change  
rs.read(); // this ← End
```



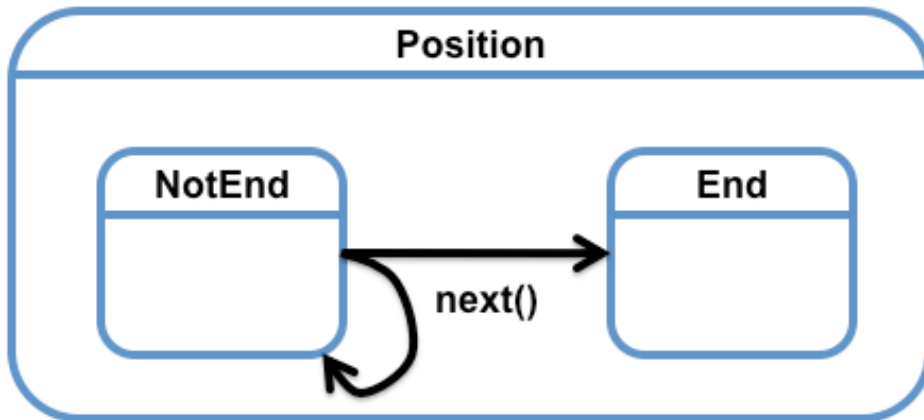
No change in the Reader dimension!

# State Members



**Solution:**  
State Members allow the incoming state to be determined dynamically

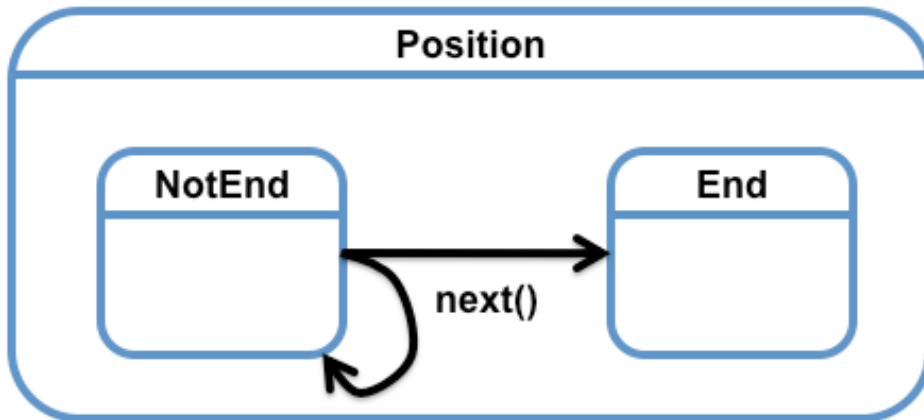
# State Members



```
state Position {  
  val endState = End;  
  /* ... */  
}
```

**Solution:**  
State Members allow the incoming state to be determined dynamically

# State Members

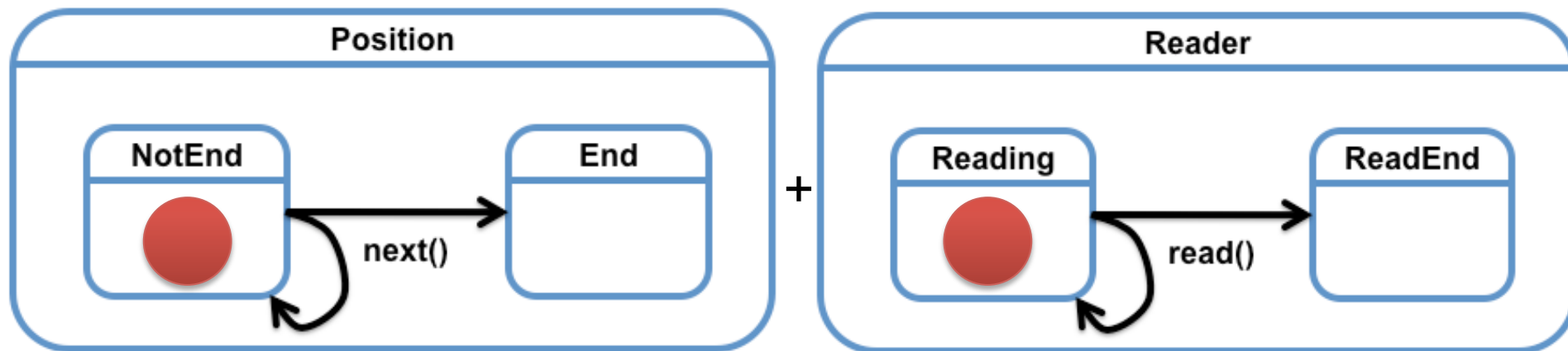


```
state Position {  
  val endState = End;  
  /* ... */  
}
```

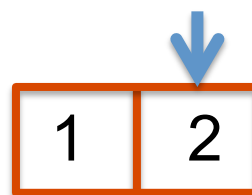
**Solution:**  
State Members allow the incoming state to be determined dynamically

```
state NotEnd case of Position {  
  method next() {  
    /* ... */  
    if (/* at the end */) {  
      this ← this.endState  
    }  
  }  
}
```

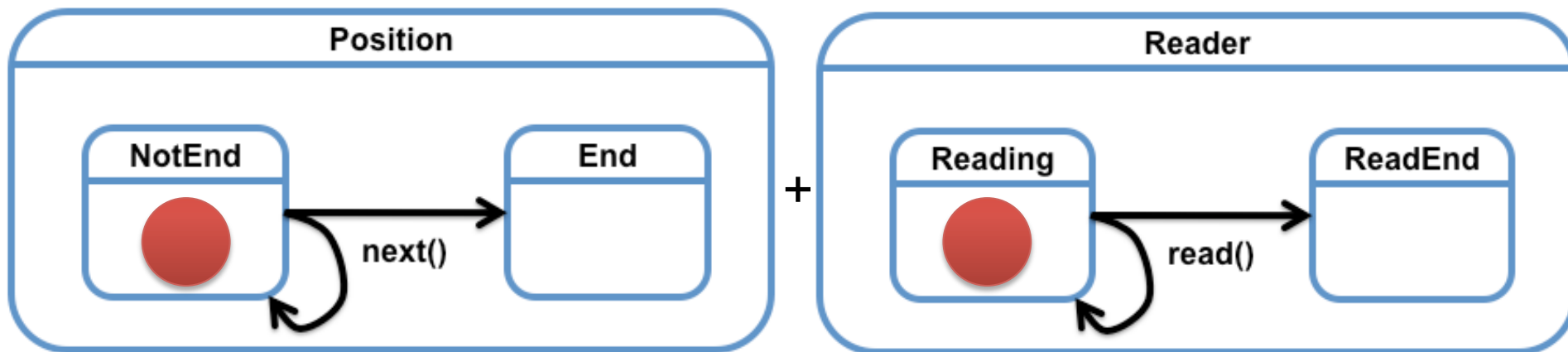
# Reuse and Composition



```
val rs = new Reading with
  NotEnd { val coll = [1,2]; ...;
    val endState = ReadEnd with End; };
rs.read(); //no state change
rs.read();
```



# Reuse and Composition



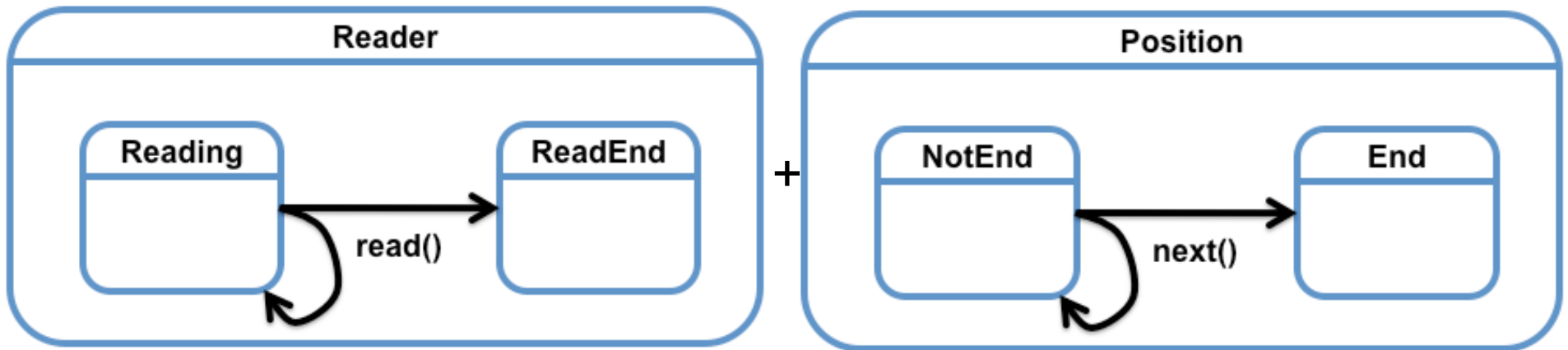
```
val rs = new Reading with  
  NotEnd { val coll = [1,2]; ...;  
    val endState = ReadEnd with End; };  
rs.read(); //no state change  
rs.read(); // this ← ReadEnd with End
```



Both dimensions change together!

# Reuse

- ReadStream = Reader + Position

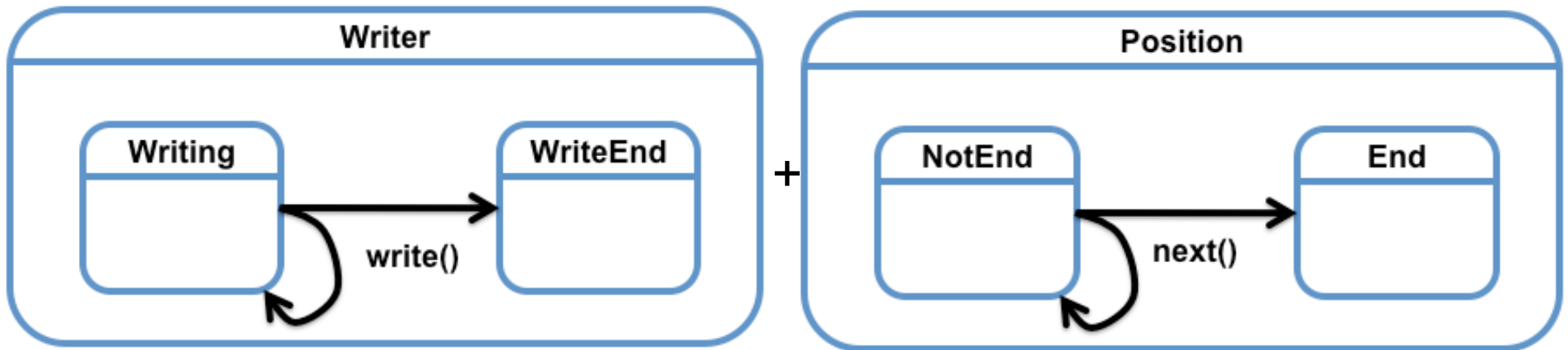


```
state Reading {  
  method read() {  
    /* read character */  
    this.next();  
  }  
}
```

```
state ReadStream = Position {  
  val endState = ReadEnd with  
    End;  
} with Reader
```

# Reuse

- Reuse as a WriteStream = Writer + Position



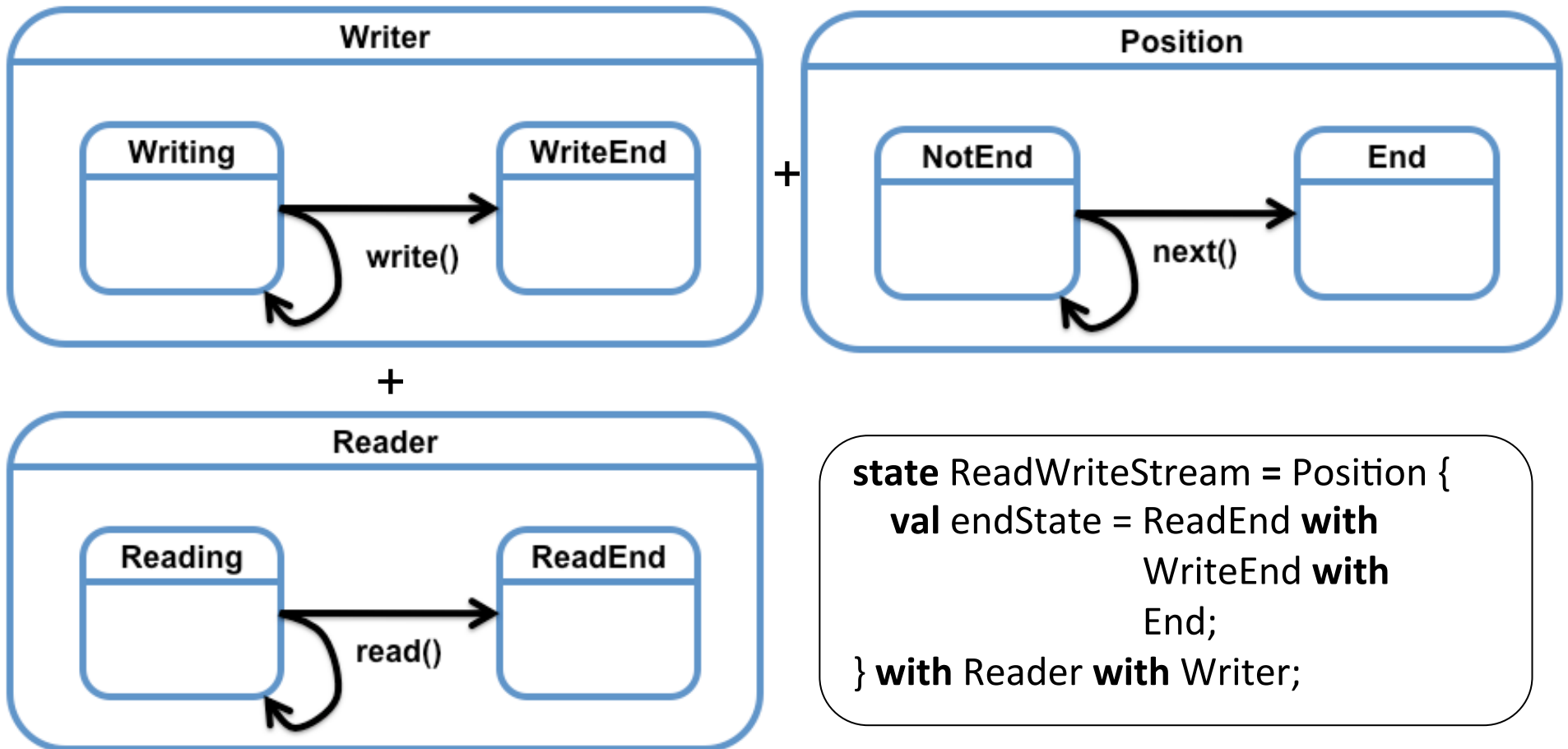
```
state Writing {  
  method write() {  
    /* write character */  
    this.next();  
  }  
}
```

```
state WriteStream = Position {  
  val endState = WriteEnd with  
    End;  
} with Writer
```



# Reuse

- Or as a ReadWriteStream = Reader + Writer + Position



# Conclusion

- Object Protocols in Plaid are
  - More concise, understandable, and safer
  - Reusable and composable

# Conclusion

- Object Protocols in Plaid are
  - More concise, understandable, and safer
  - Reusable and composable
- Open up new possibilities
  - Visualization tools
  - More helpful error messages
  - Static checking

# Conclusion

- Object Protocols in Plaid are
  - More concise, understandable, and safer
  - Reusable and composable
- Open up new possibilities
  - Visualization tools
  - More helpful error messages
  - Static checking

**PLAID** Demo, 2pm Galleria III