Swift × Northwest, October 27, 2017

These are a Few of My Stateful Machines

Curt Clifton
The Omni Group
Twitter: @curtclifton
Web: www.curtclifton.net

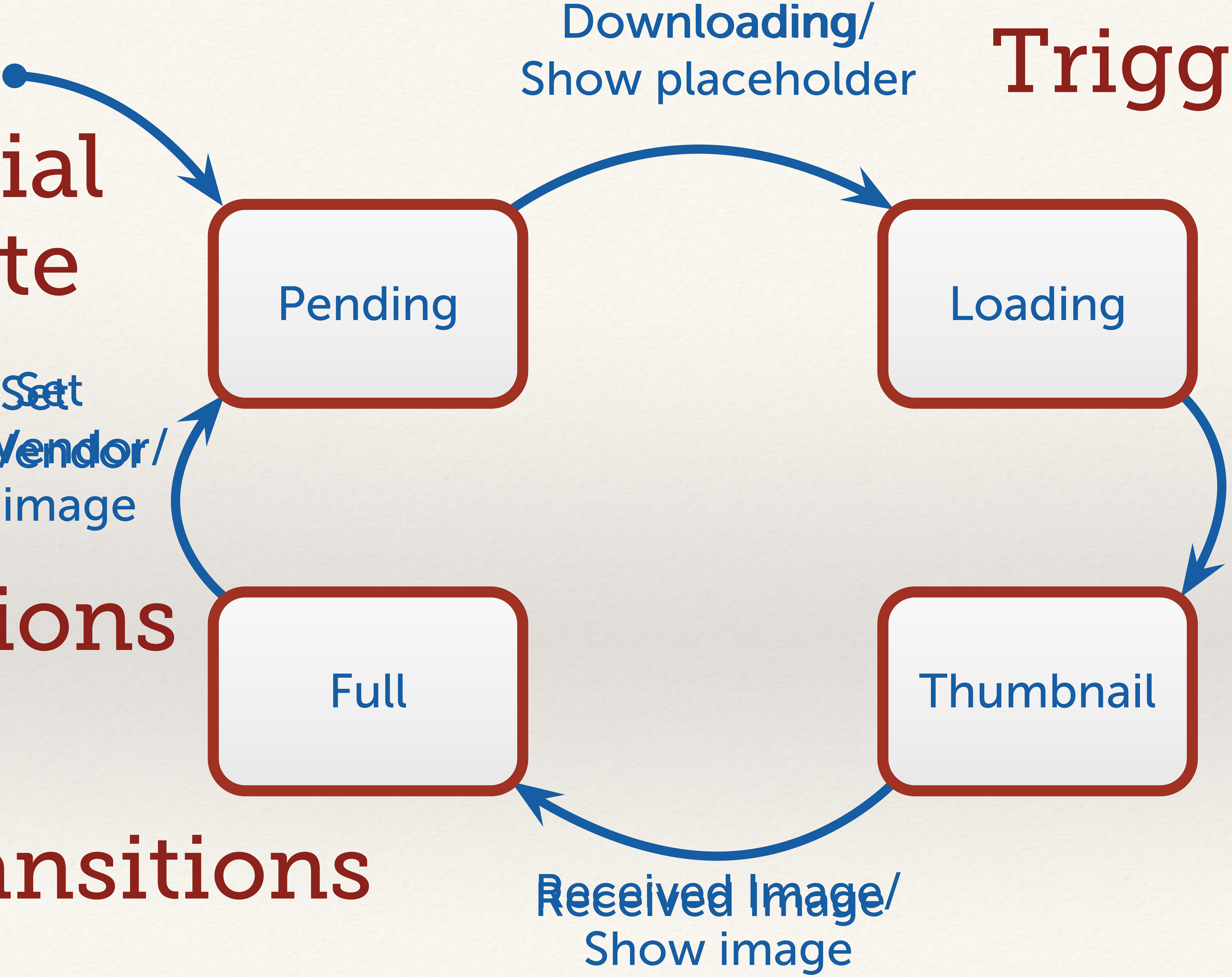
Goals

- Understand the basics of state machines
- Recognize when one is appropriate
- Quickly create state machines in Swift

What are State Machines?

Demo

Terms



Triggers

States

Initial State

Downloading/
Show placeholder

Pending

Loading

Received
Thumbnail/
Show
thumbnail

Received
Thumbnail/

Thumbnail

Received Image/
Show image

Received Image/

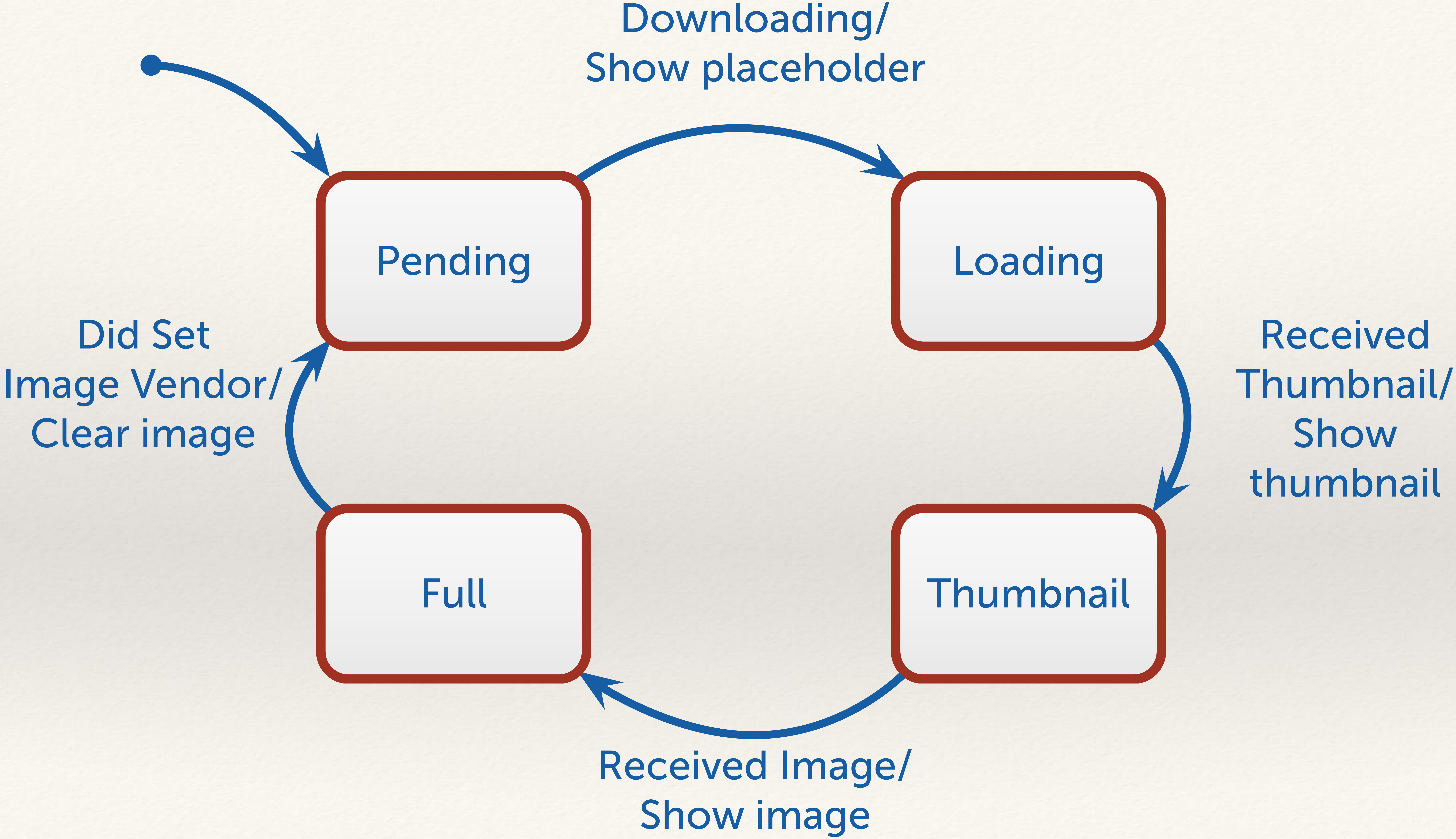
Full

Did Set
Image Vendor/
Clear image

Did Set
Image Vendor/

Actions

Transitions



Downloading/
Show placeholder

Pending

Loading

Received
Thumbnail/
Show
thumbnail

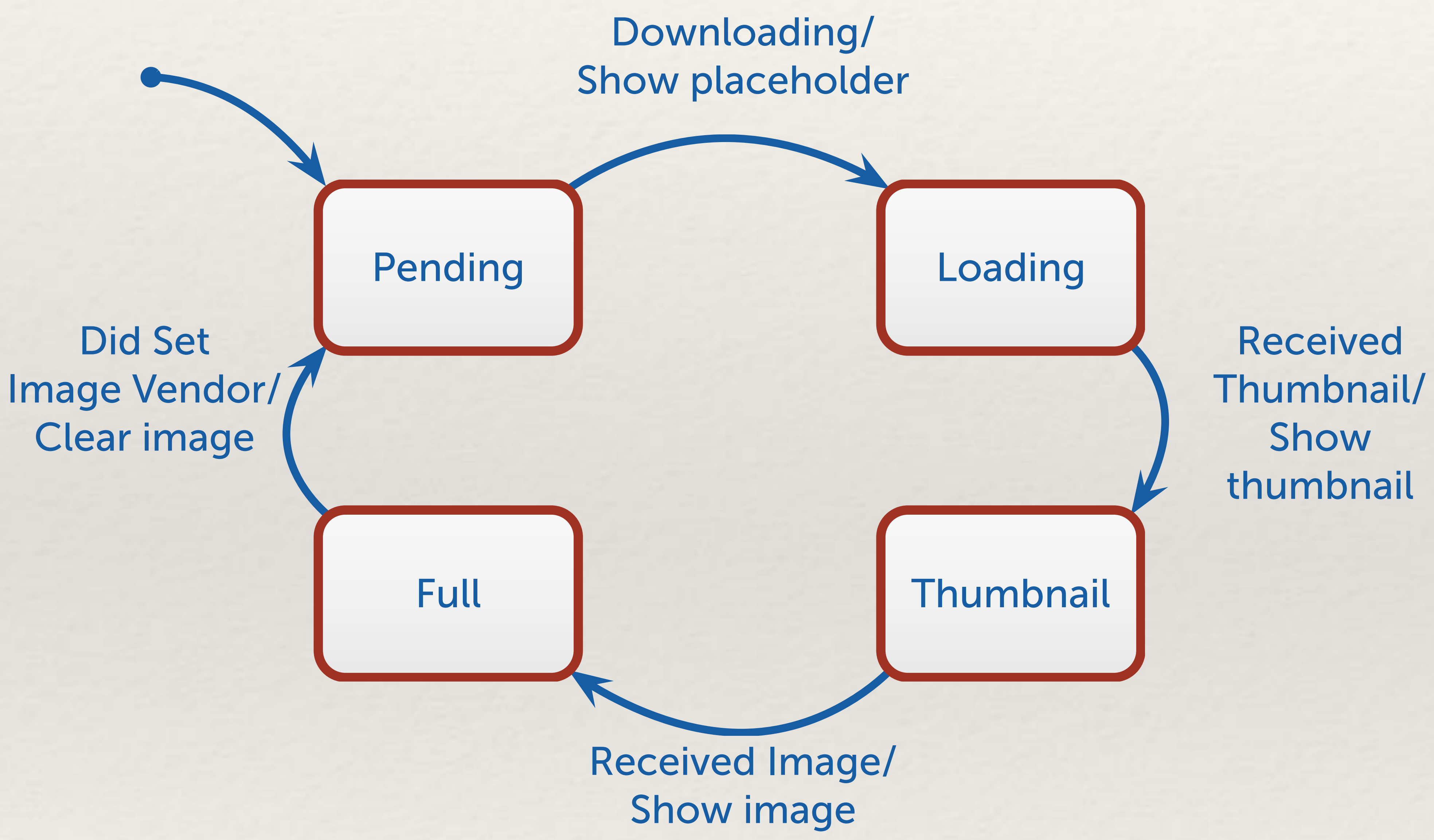
Thumbnail

Received Image/
Show image

Full

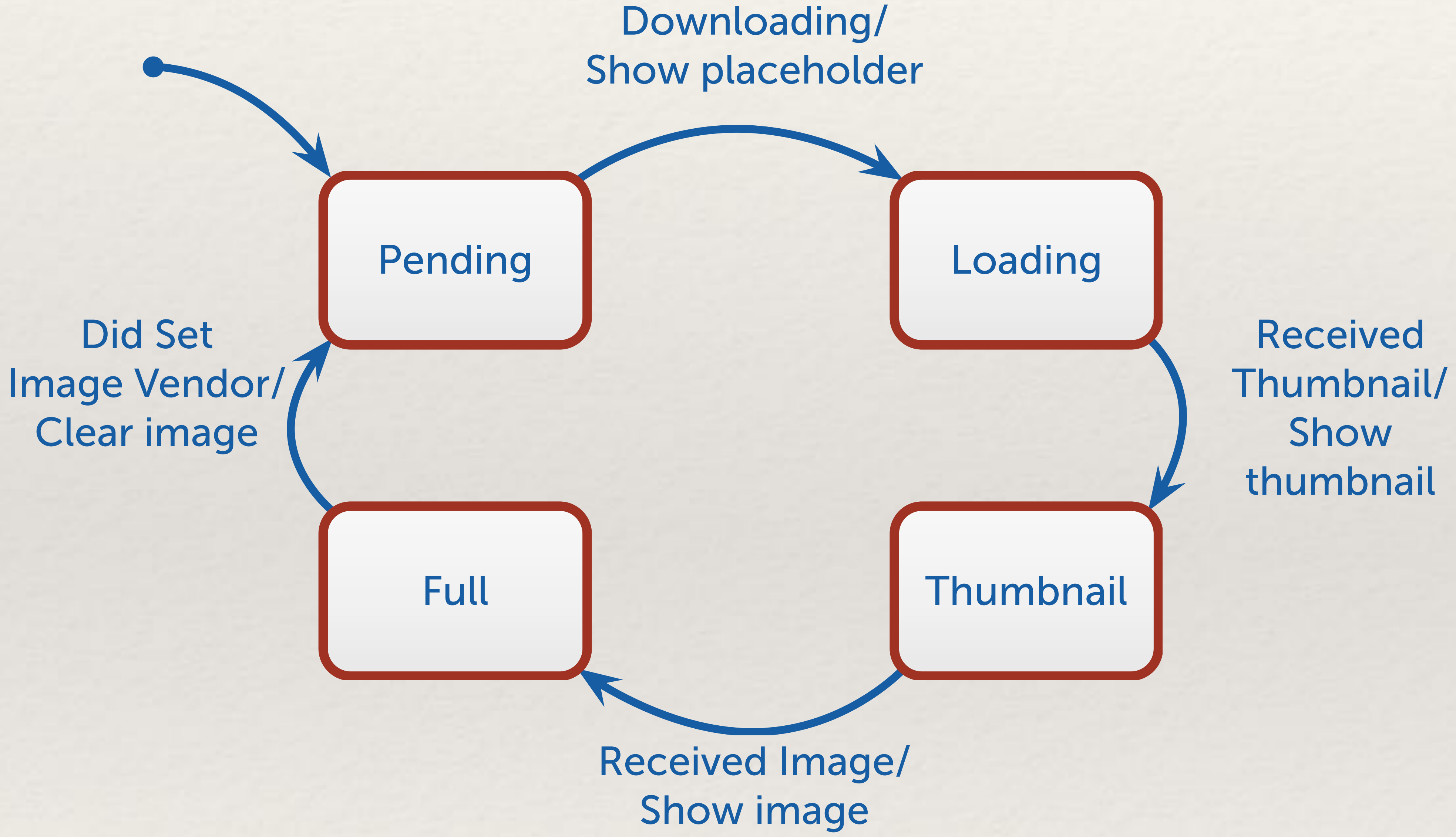
Did Set
Image Vendor/
Clear image

Swift Enumerations



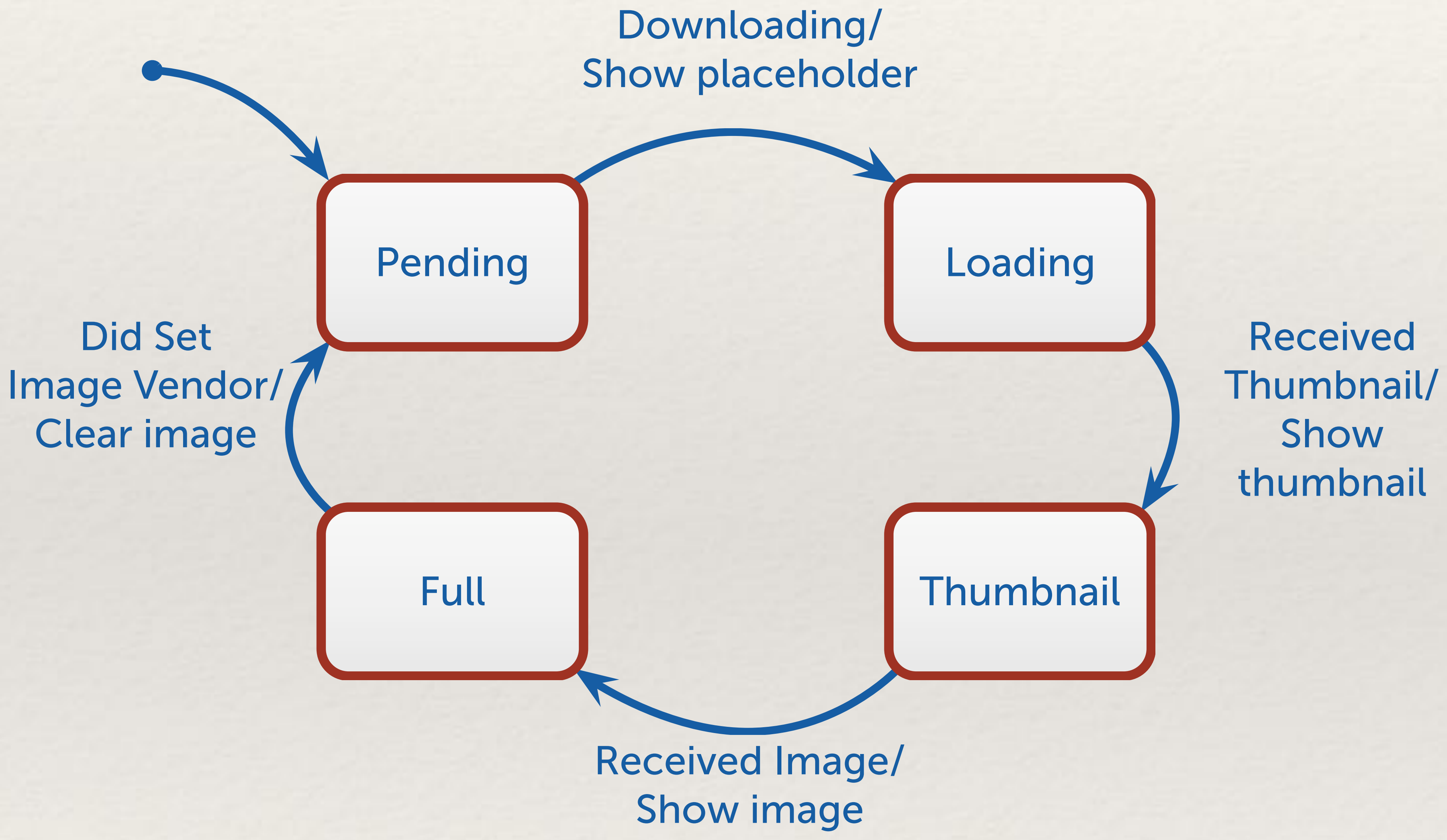
```
class ImageCollectionViewCell: UICollectionViewCell {
    @IBOutlet weak var imageView: UIImageView!
    private var state: State
    ...
}

private enum State {
    case pending
    case loading
    case thumbnail
    case full
}
```

Swift Enumerations



```
class ImageCollectionViewCell: UICollectionViewCell {
    @IBOutlet weak var imageView: UIImageView!
    private var state: State = .pending
    ...
}

private enum State {
    case pending
    case loading
    case thumbnail
    case full
}
```

Swift Enumerations



Downloading/
Show placeholder

Pending

Loading

Received
Thumbnail/
Show
thumbnail

Thumbnail

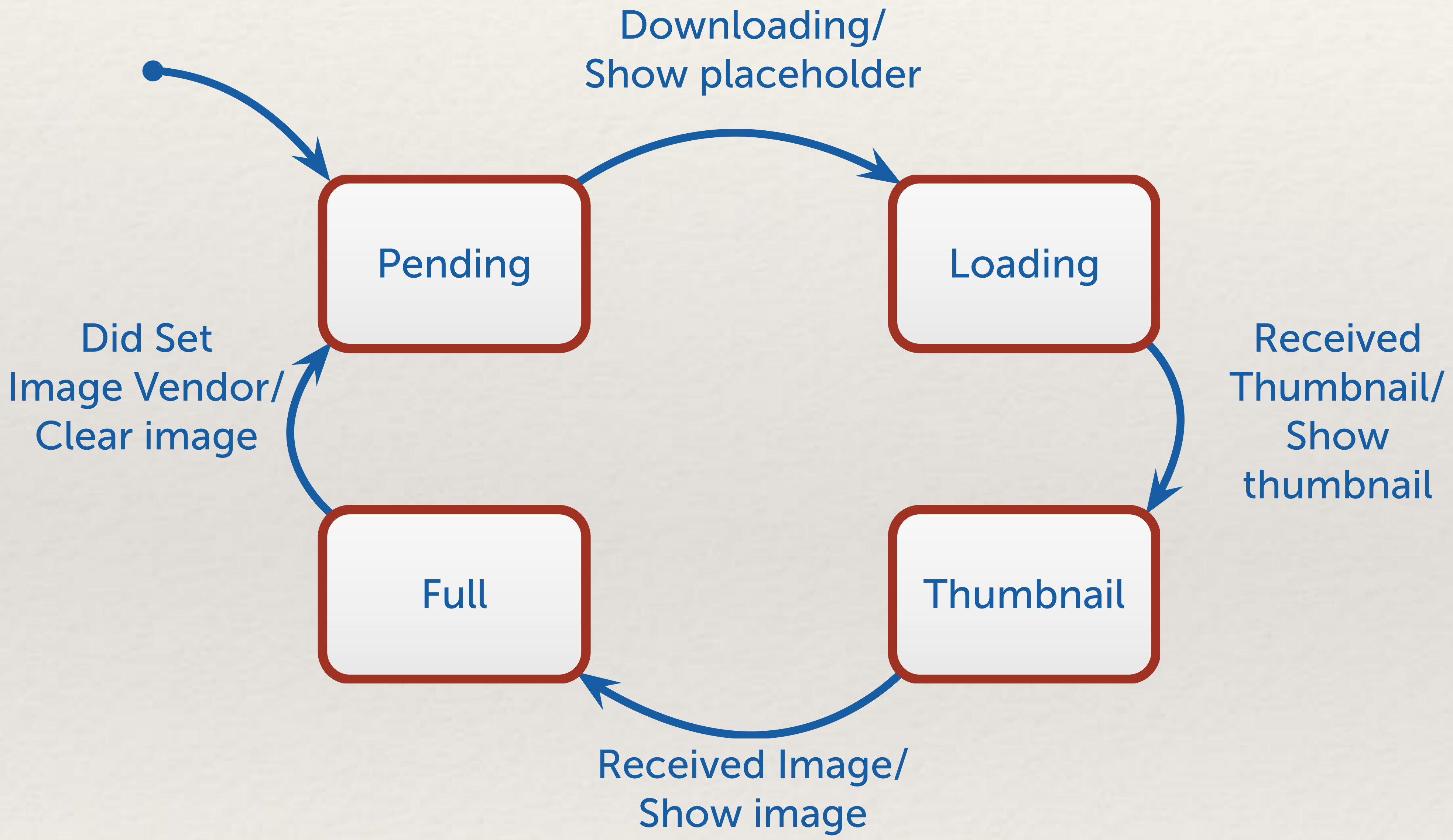
Received Image/
Show image

Full

Did Set
Image Vendor/
Clear image

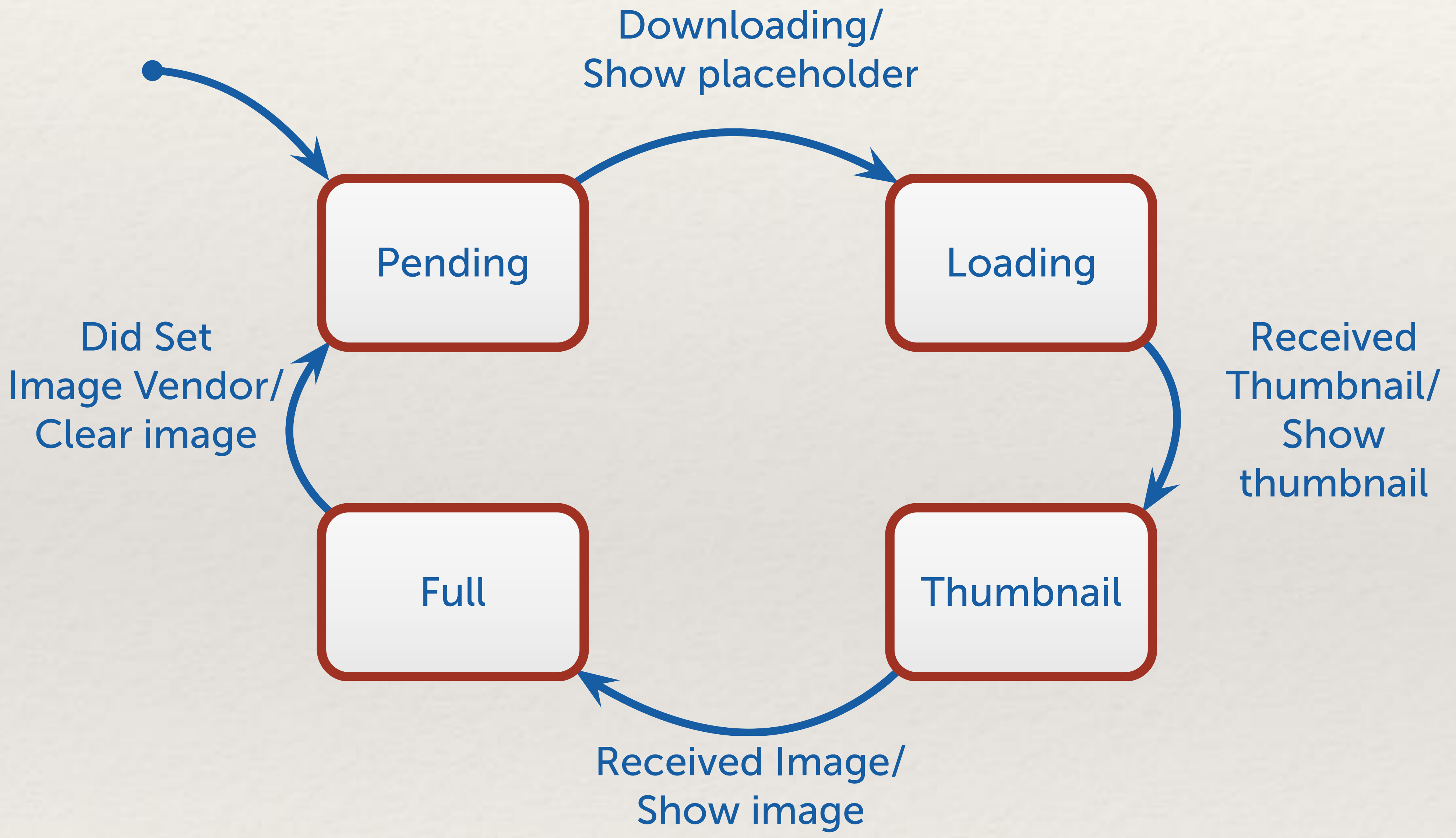
```
class ImageCollectionViewCell: UICollectionViewCell {
    @IBOutlet weak var imageView: UIImageView!
    private var state: State = .pending
    ...
}

private enum State {
    case pending
    case loading
    case thumbnail
    case full
}
extension ImageCollectionViewCell: ImageVendorDelegate
{
    func downloading(id: ImageID) {
        ...
    }

    func receivedThumbnail(_ thumbnail: UIImage
```

Swift Enumerations



```
    case full
}

extension ImageCollectionViewCell: ImageVendorDelegate
{
    func downloading(id: ImageID) {
        state = .loading
        ...
    }

    func receivedThumbnail(_ thumbnail: UIImage,
                           for id: ImageID) {
        state = .thumbnail
        ...
    }

    func receivedImage(_ fullResolutionImage: UIImage,
                       for id: ImageID) {
        state = .full
        ...
```

Swift Enumerations



Downloading/
Show placeholder

Pending

Loading

Did Set
Image Vendor/
Clear image

Received
Thumbnail/
Show
thumbnail

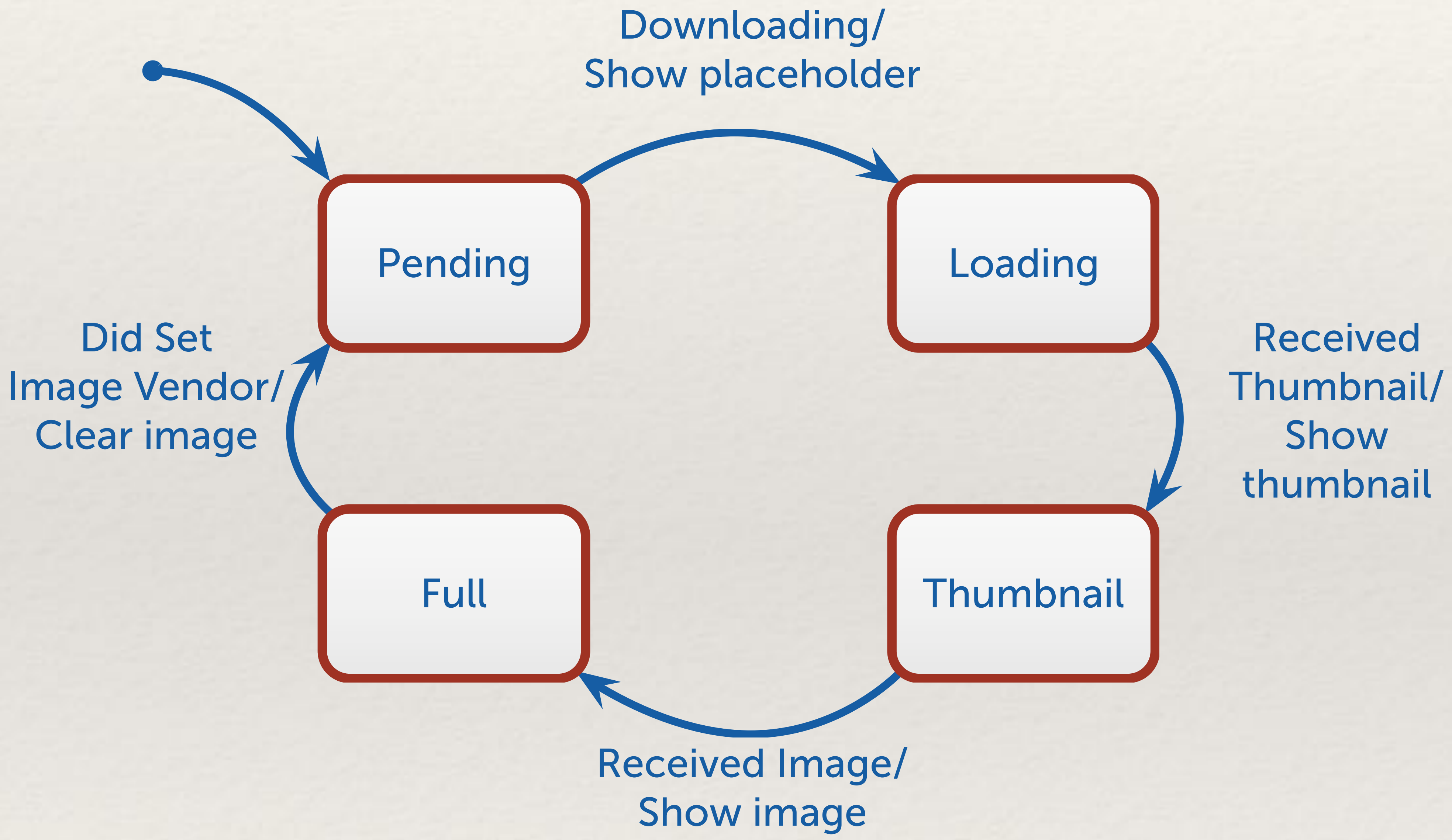
Full

Thumbnail

Received Image/
Show image

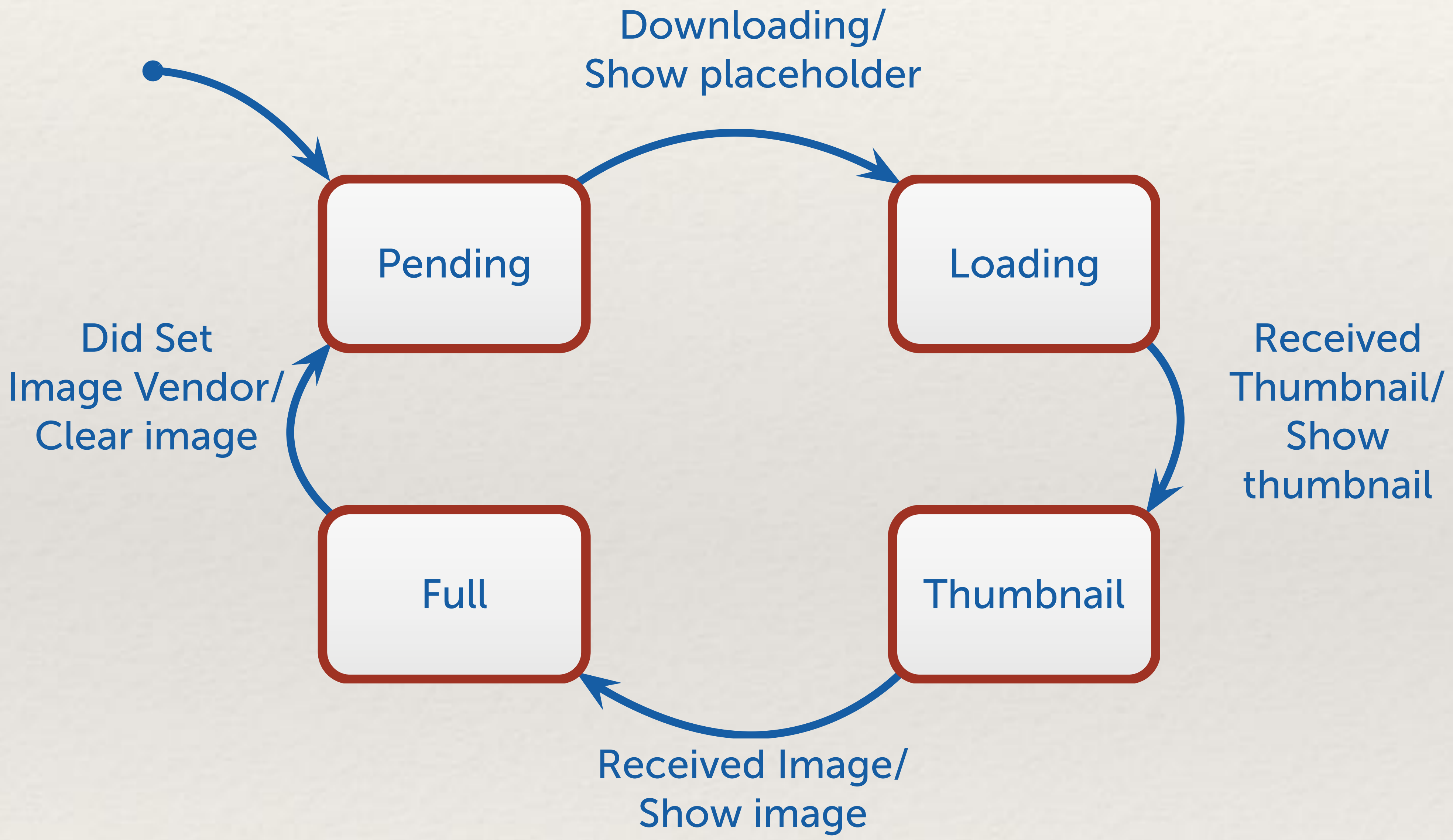
```
extension ImageSelectionViewCell: ImageVendorDelegate
{
    func downloading(id: ImageID) {
        state = .loading
        imageView.image = placeholder
    }

    func receivedThumbnail(_ thumbnail: UIImage,
                           for id: ImageID) {
        state = .thumbnail
        ...
    }

    func receivedImage(_ fullResolutionImage: UIImage,
                       for id: ImageID) {
        state = .full
        ...
    }
}
```

Swift Enumerations



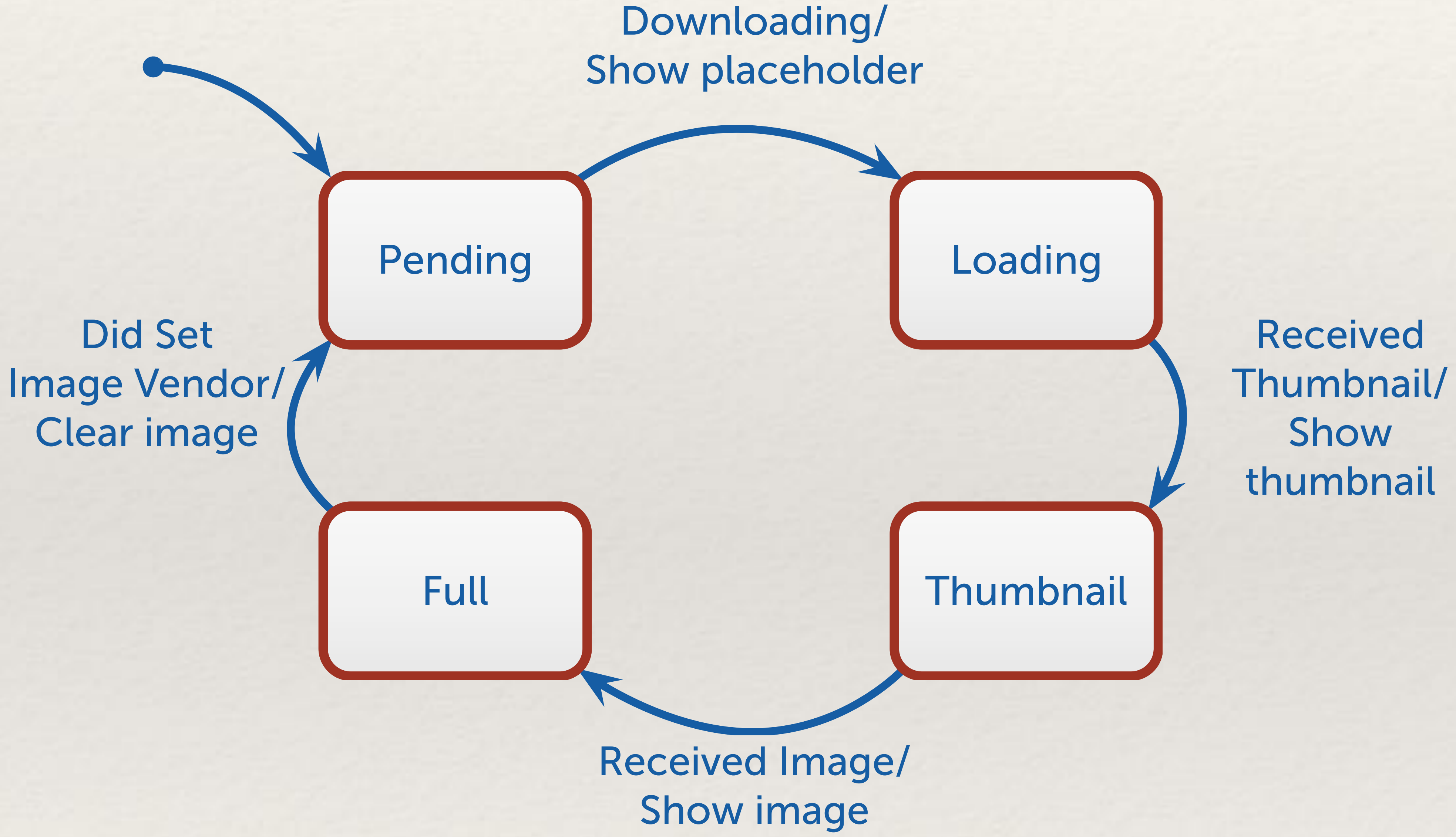
```
{
    func downloading(id: ImageID) {
        state = .loading
        imageView.image = placeholder
    }

    func receivedThumbnail(_ thumbnail: UIImage,
                                        for id: ImageID) {
        state = .thumbnail
        imageView.image = thumbnail
    }

    func receivedImage(_ fullResolutionImage: UIImage,
                               for id: ImageID) {
        state = .full
        ...
    }
}
```

Swift Enumerations



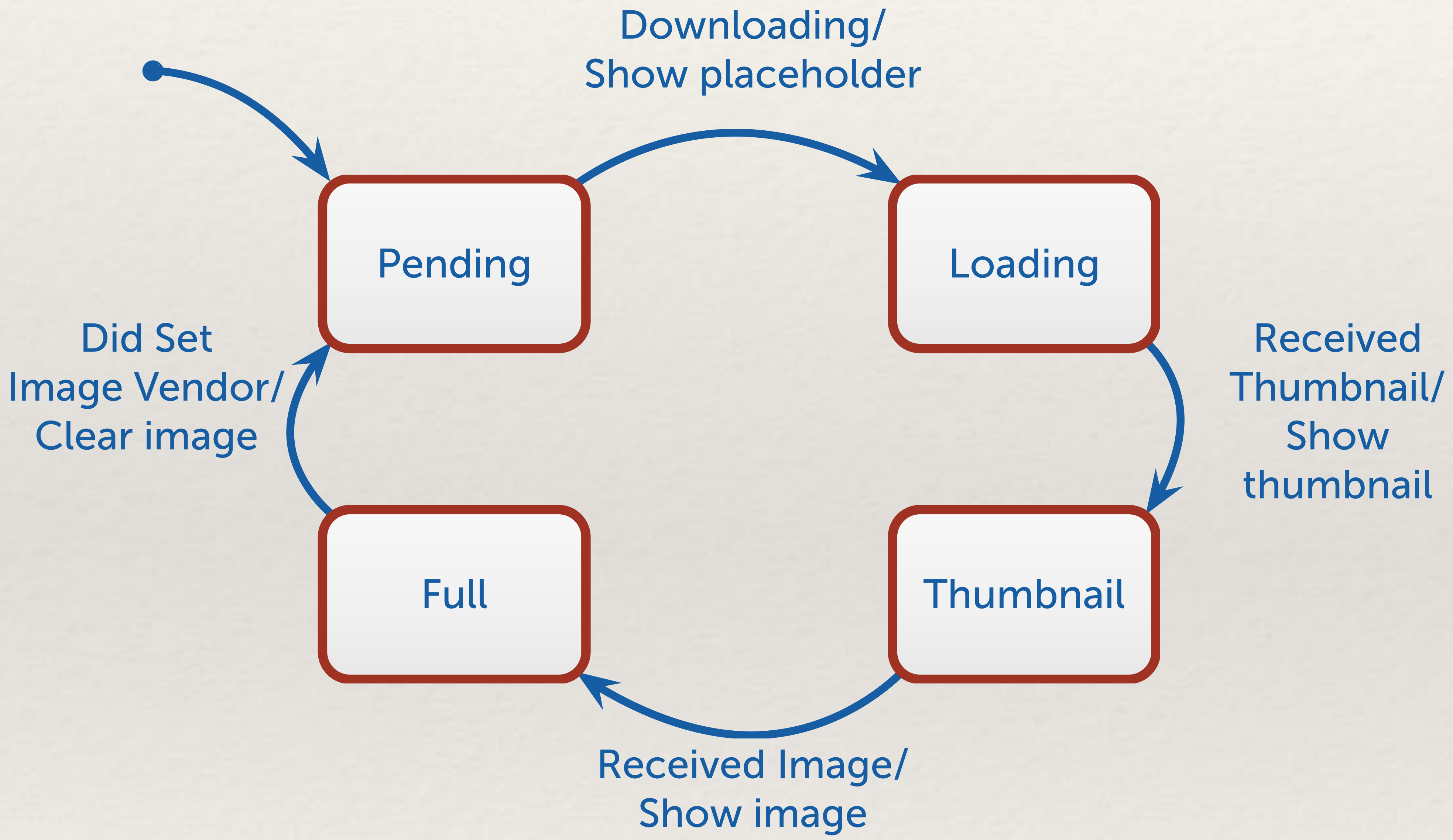
```
extension ImageCollectionViewCell: ImageVendorDelegate
{
    func downloading(id: ImageID) {
        state = .loading
        imageView.image = placeholder
    }

    func receivedThumbnail(_ thumbnail: UIImage,
                           for id: ImageID) {
        state = .thumbnail
        imageView.image = thumbnail
    }

    func receivedImage(_ fullResolutionImage: UIImage,
                       for id: ImageID) {
        state = .full
        imageView.image = fullResolutionImage
    }
}
```

Swift Enumerations



```
{
    func downloading(id: ImageID) {
        state = .loading
        imageView.image = placeholder
    }

    func receivedThumbnail(_ thumbnail: UIImage,
                                        for id: ImageID) {
        state = .thumbnail
        imageView.image = thumbnail
    }

    func receivedImage(_ fullResolutionImage: UIImage,
                                for id: ImageID) {
        state = .full
        imageView.image = fullResolutionImage
    }
}
```

Swift Enumerations



Downloading/
Show placeholder

Pending

Loading

Did Set
Image Vendor/
Clear image

Received
Thumbnail/
Show
thumbnail

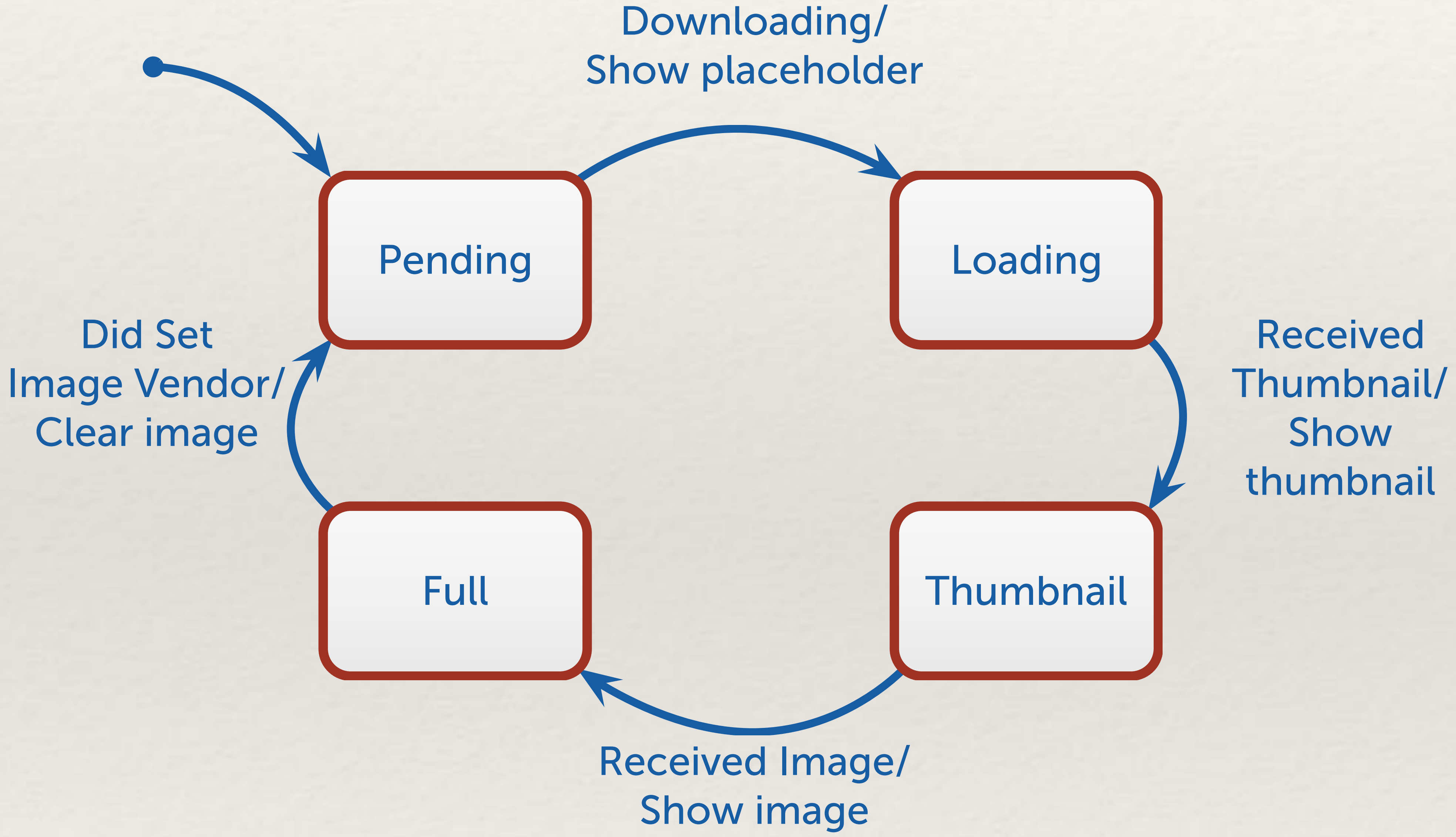
Full

Thumbnail

Received Image/
Show image

```
class ImageCollectionViewCell: UICollectionViewCell {
    @IBOutlet weak var imageView: UIImageView!
    private var state: State = .pending
    …
    override func prepareForReuse() {
        super.prepareForReuse()
        imageVendor = nil
    }
}

private enum State {
    case pending
    case loading
    case thumbnail
    case full
}

extension ImageCollectionViewCell: ImageVendorDelegate
{
    func downloading(id: ImageID) {
```

Swift Enumerations



Downloading/
Show placeholder

Pending

Loading

Did Set
Image Vendor/
Clear image

Received
Thumbnail/
Show
thumbnail

Full

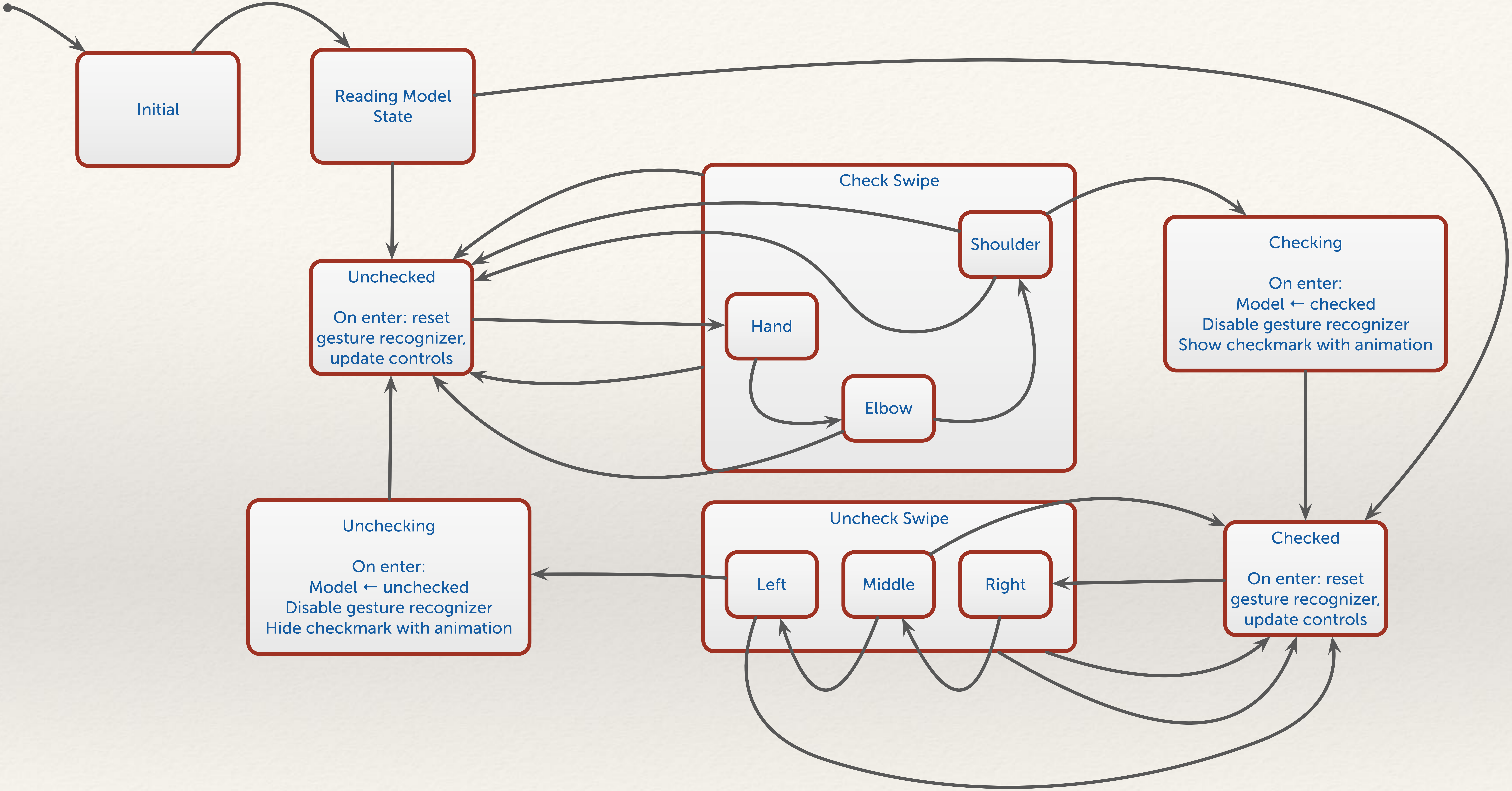
Thumbnail

Received Image/
Show image

```
class ImageCollectionViewCell: UICollectionViewCell {
    @IBOutlet weak var imageView: UIImageView!
    private var state: State = .pending
    var imageVendor: ImageVendor? {
        didSet {
            state = .pending
            ...
        }
    }

    override func prepareForReuse() {
        super.prepareForReuse()
        imageVendor = nil
    }
}

private enum State {
    case pending
    case loading
```

Gesture Recognizer Example

Demo

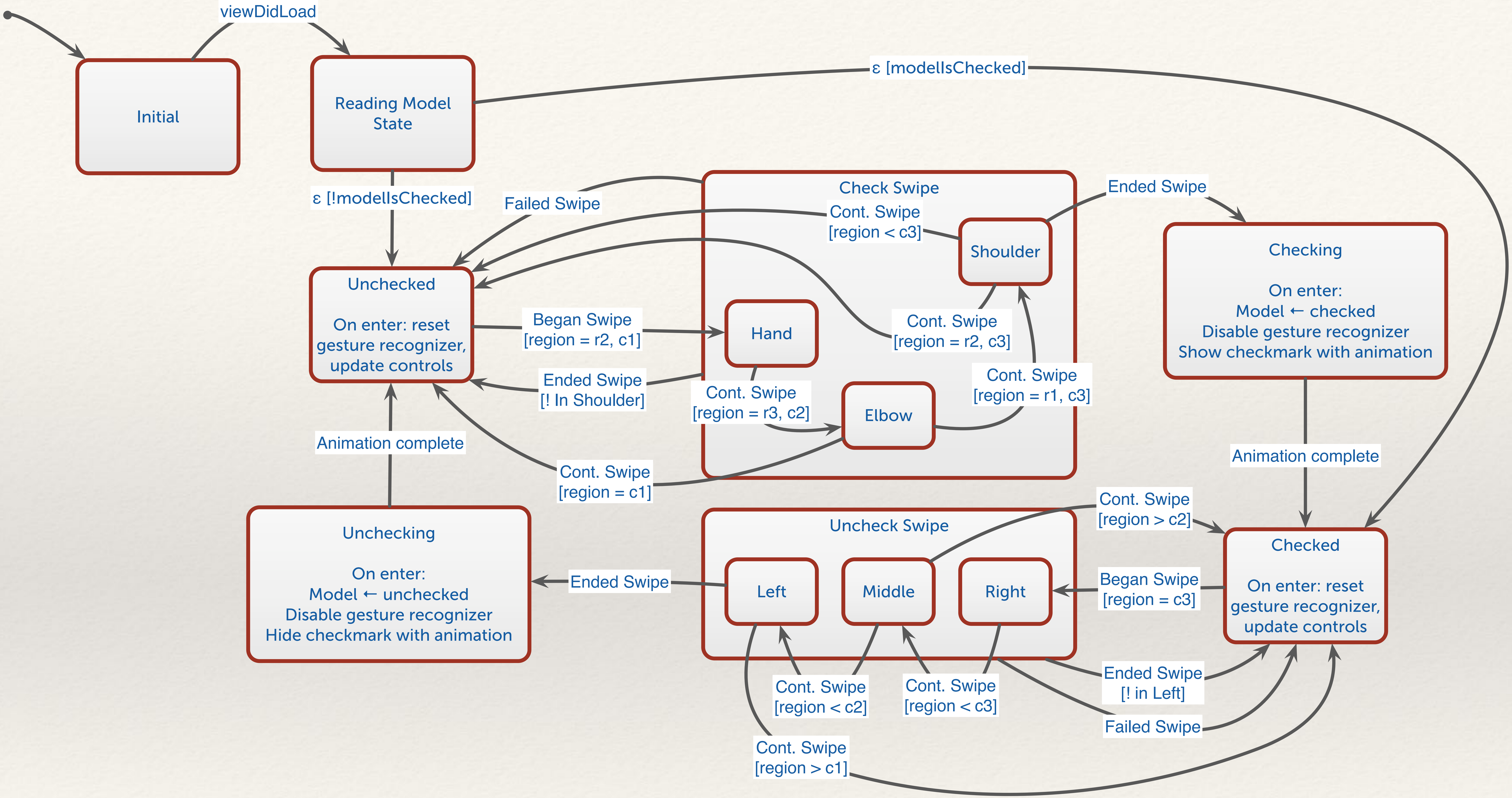

Initial
Reading Model State
Check Swipe
Shoulder
Hand
Elbow
Checking

On enter:
Model ← checked
Disable gesture recognizer
Show checkmark with animation
Unchecked

On enter: reset gesture recognizer, update controls
Unchecking

On enter:
Model ← unchecked
Disable gesture recognizer
Hide checkmark with animation
Uncheck Swipe
Left
Middle
Right
Checked

On enter: reset gesture recognizer, update controls



viewDidLoad

Initial

Reading Model State

ε [modelIsChecked]

ε [!modelIsChecked]

Unchecked

On enter: reset gesture recognizer, update controls

Failed Swipe

Began Swipe [region = r2, c1]

Ended Swipe [! In Shoulder]

Animation complete

Cont. Swipe [region = c1]

Check Swipe

Cont. Swipe [region < c3]

Shoulder

Hand

Cont. Swipe [region = r2, c3]

Cont. Swipe [region = r3, c2]

Elbow

Cont. Swipe [region = r1, c3]

Ended Swipe

Checking

On enter:
Model ← checked
Disable gesture recognizer
Show checkmark with animation

Animation complete

Unchecking

On enter:
Model ← unchecked
Disable gesture recognizer
Hide checkmark with animation

Ended Swipe

Uncheck Swipe

Left

Middle

Right

Cont. Swipe [region > c2]

Began Swipe [region = c3]

Checked

On enter: reset gesture recognizer, update controls

Cont. Swipe [region < c2]

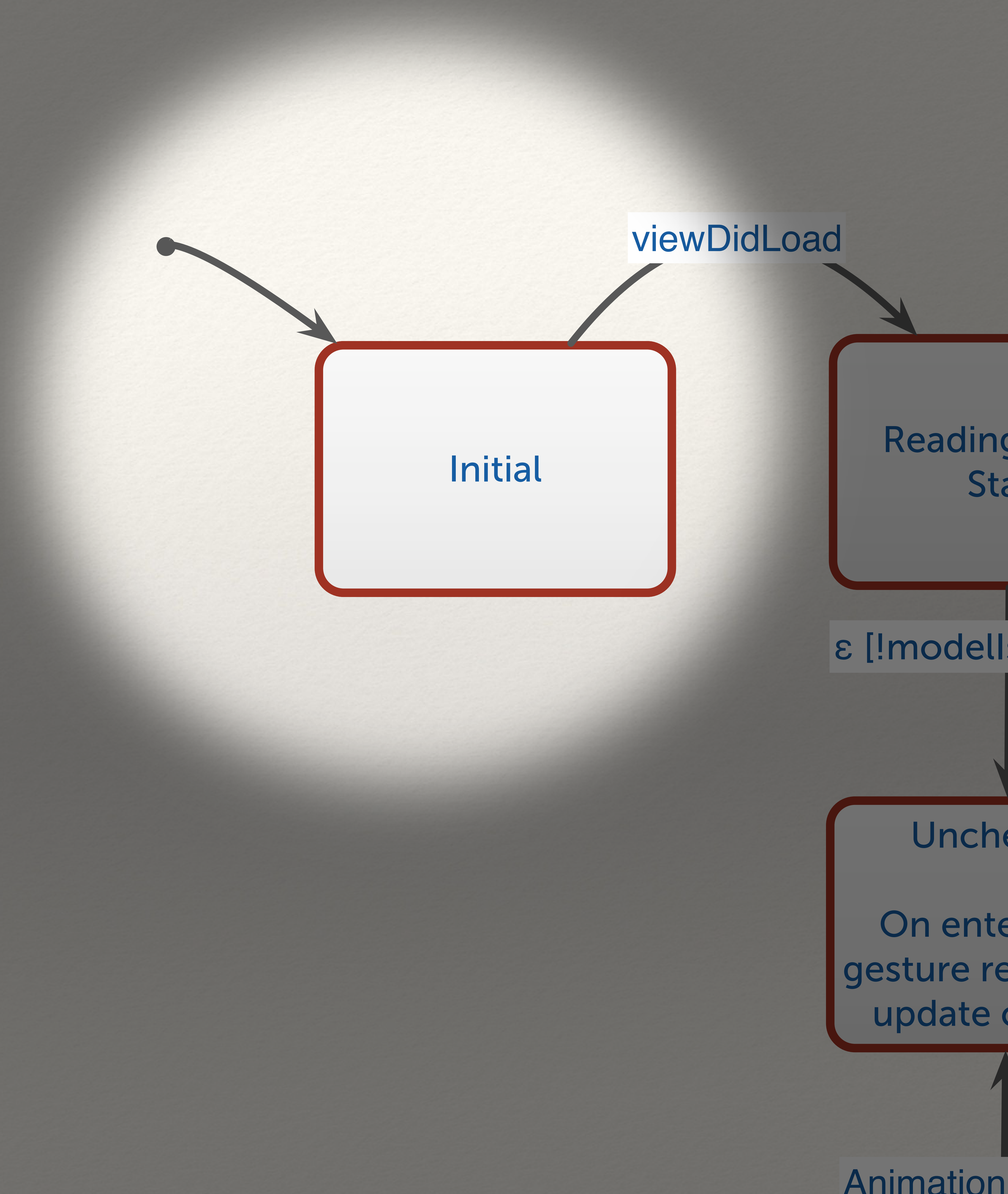
Cont. Swipe [region < c3]

Ended Swipe [! in Left]

Failed Swipe

Cont. Swipe [region > c1]

```
private var state: State = .initial { ... }
```



viewDidLoad

Initial

Reading
Sta

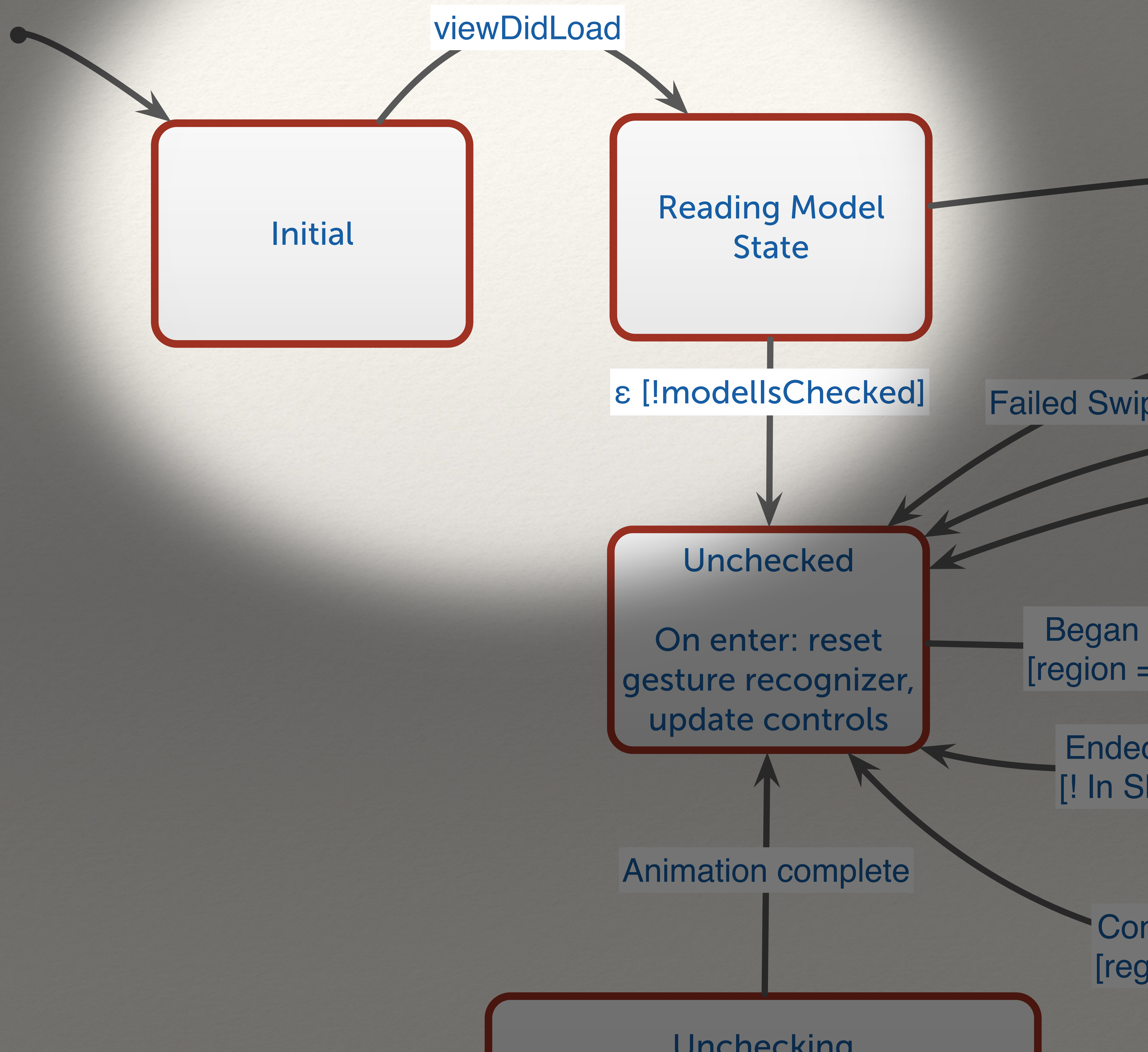
ε [!modell

Unche

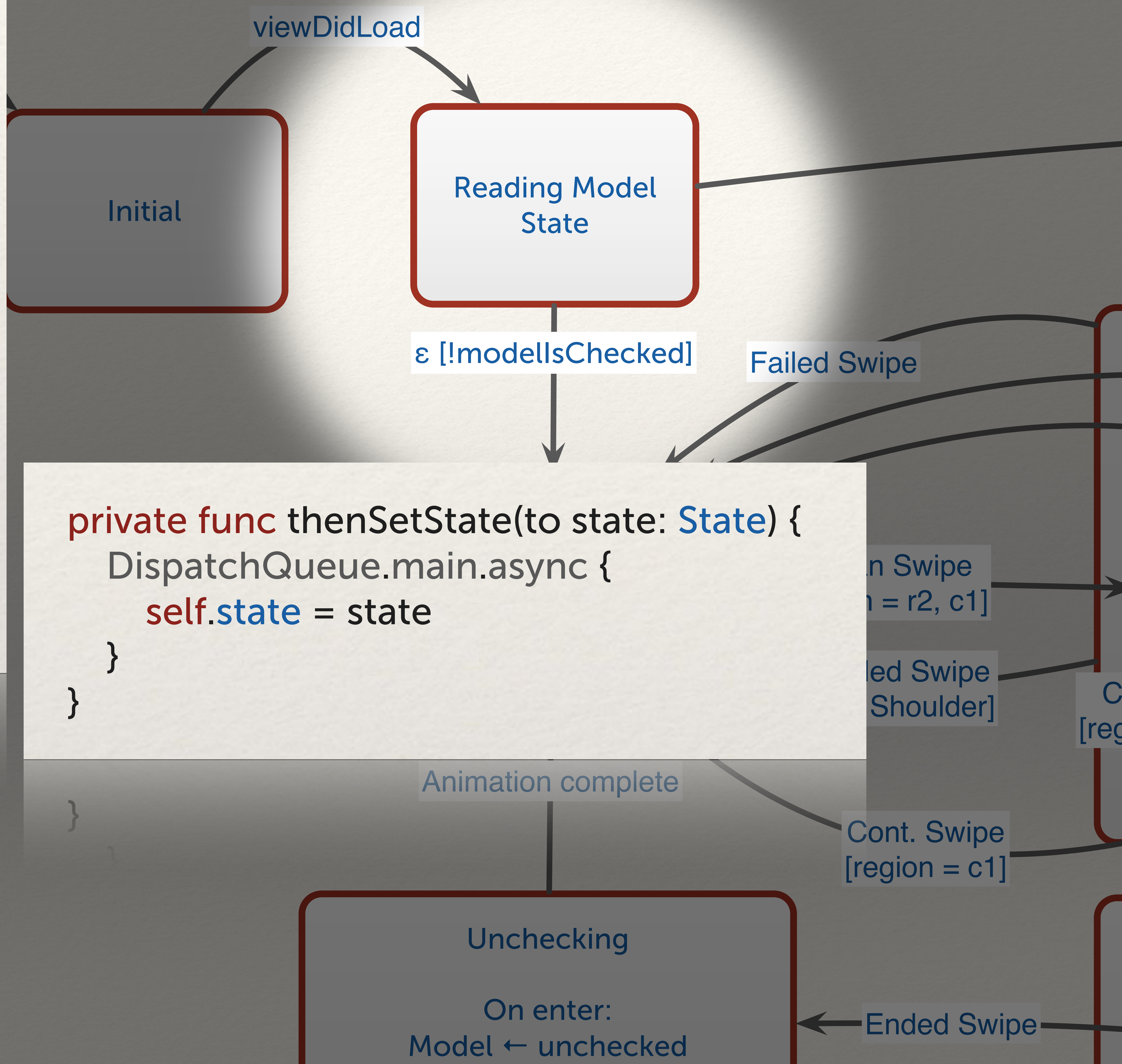
On ente
gesture re
update

Animation

```
override func viewDidLoad() {
    super.viewDidLoad()
    ...
    state = .readingModelState
}
```

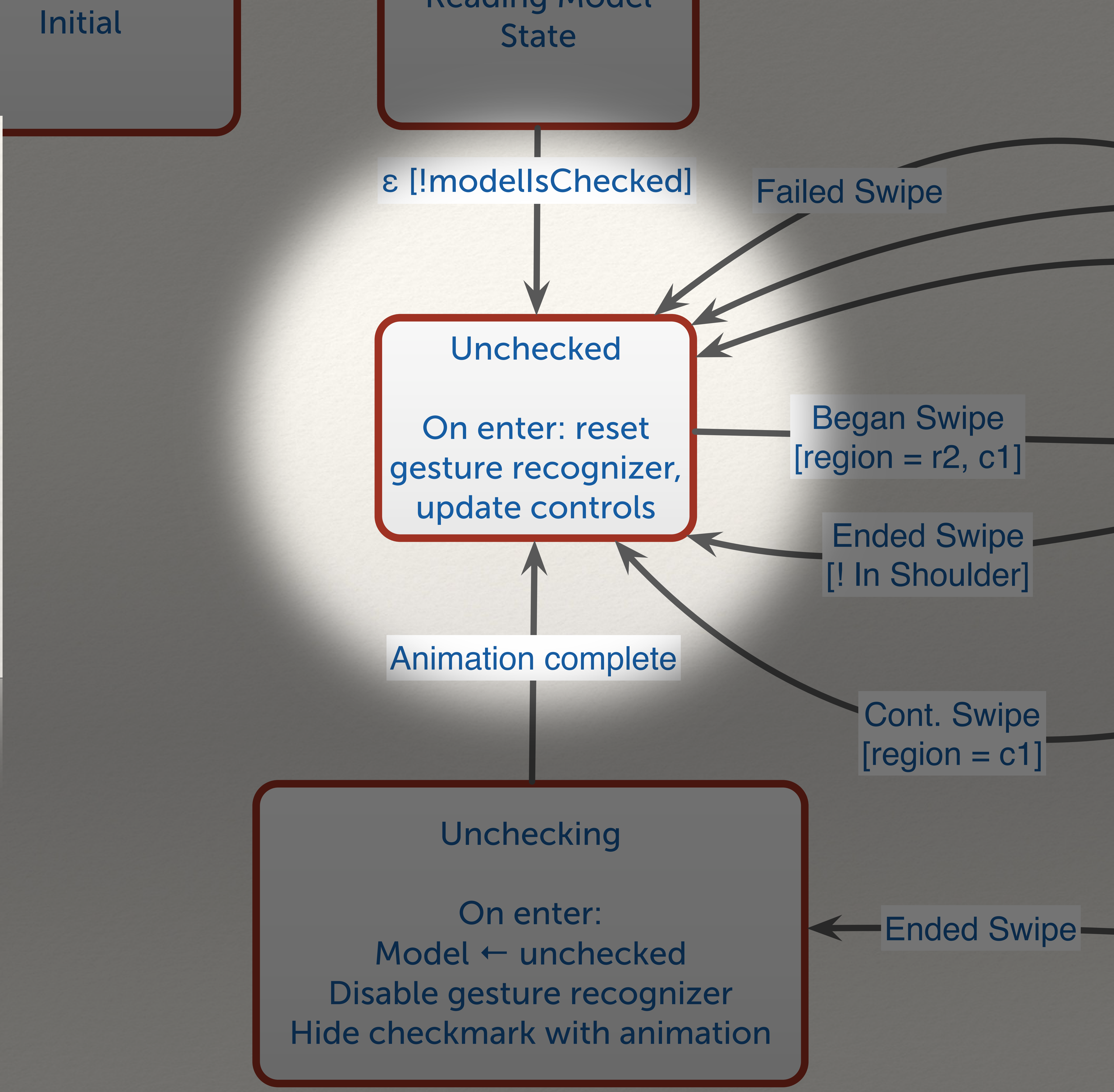


```
private var state: State = .initial {
    didSet {
        guard state != oldValue else { return }

        switch state {
        case .readingModelState:
            if (modelIsChecked) {
                self.thenSetState(to: .checked)
            } else {
                self.thenSetState(to: .unchecked)
            }
        }
        ...
    }
```



```
private func thenSetState(to state: State) {
    DispatchQueue.main.async {
        self.state = state
    }
}
```

```
private var state: State = .initial {
  didSet {
    guard state != oldValue else { return }

    switch state {
    ...
    case .unchecked:
      resetGestureRecognizer()
      updateControls()
    ...
  }
```



Initial

Reading Model State

ε [!modelIsChecked]

Failed Swipe

Unchecked

On enter: reset gesture recognizer, update controls

Began Swipe [region = r2, c1]

Ended Swipe [! In Shoulder]

Animation complete

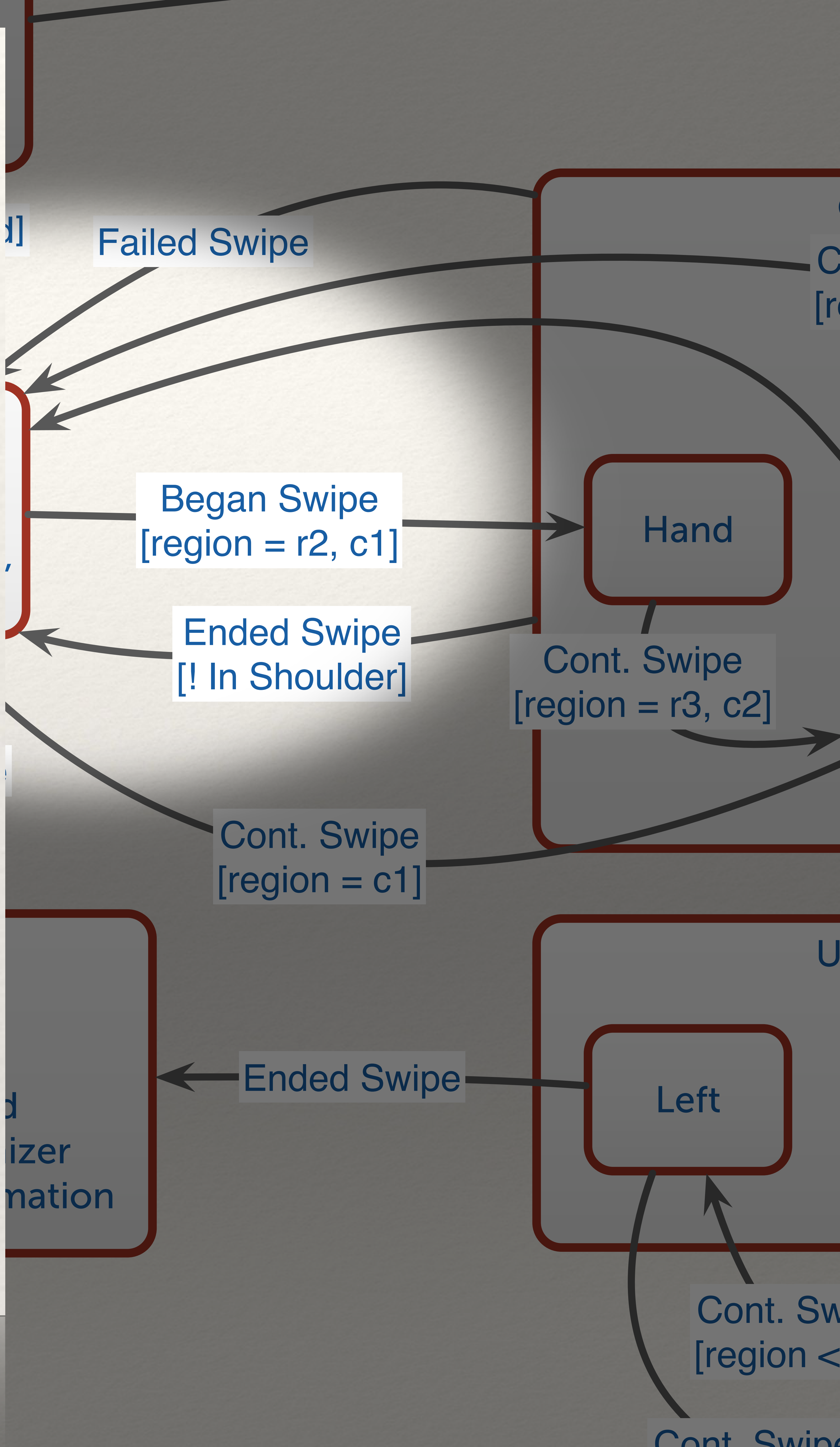
Cont. Swipe [region = c1]

Unchecking

On enter:
Model ← unchecked
Disable gesture recognizer
Hide checkmark with animation

Ended Swipe

```
@IBAction func gestureDidUpdate(_ pan: UIPanGestureRecognizer) {
    do {
        let region = try PanRegion(panRecognizer: pan)
        switch pan.state {
        case .began:
            beganSwipe(in: region)
        case .changed:
            continuedSwipe(in: region)
        case .ended:
            endedSwipe()
        case .failed:
            failedSwipe()
        case .possible, .cancelled: // cancel ourselves to reset
            break
        }
    } catch is RangeError {
        endedSwipe()
   } catch {
        failedSwipe()
   }
}
```



Reading Model
Initial
Check Swipe
Cont. Swipe [region < c3]
Shoulder
Ended Sw
Hand
Cont. Swipe [region = r2, c3]
Cont. Swipe [region = r1, c3]
Cont. Swipe [region = r3, c2]
Elbow
Unchecking
On enter:
Model ← unchecked
Disable gesture recognizer
Hide checkmark with animation
Ended Sw
Cont. Swipe [region < c2]
Cont. Swipe [region < c3]
Ended Swi [! in Left

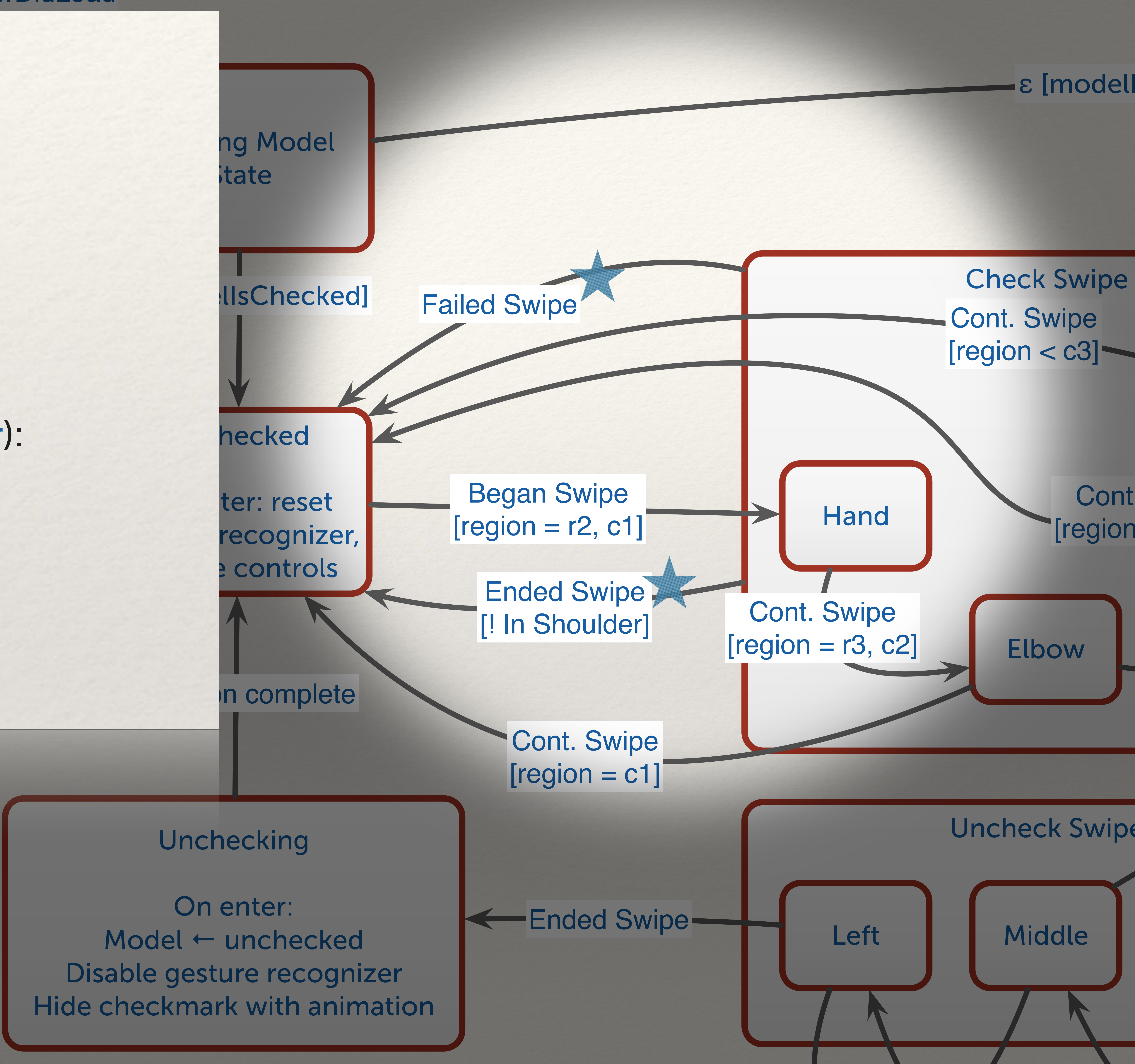
```swift
private enum State {
    case initial
    case readingModelState

    case unchecking
    case unchecked

    case checking
    case checked

    case checkSwipe(substate: CheckSwipeState)
    case uncheckSwipe(substate: UncheckSwipeState)
}
```

```swift
private enum CheckSwipeState {
    case hand
    case elbow
    case shoulder
}
```

```
private func beganSwipe(in region: PanRegion) {
   switch state {
   case .unchecked
      where region == PanRegion(row: 2, column: 1):
      state = .checkSwipe(substate: .hand)
   ...
}
```

Initial
Reading Model
State
ε [!modelIsChecked]
Failed Swipe
Check Swipe
Cont. Swipe
[region < c3]
Should
Began Swipe
[region = r2, c1]
Hand
Cont. Swipe
[region = r2, c3]
Ended Swipe
[! In Shoulder]
Cont. Swipe
[region = r3, c2]
Elbow
Cont
[region
Animation complete
Cont. Swipe
[region = c1]
Unchecking
On enter:
Model ← unchecked
Disable gesture recognizer
Hide checkmark with animation
Ended Swipe
Uncheck Swipe
Left
Middle
Right
Cont. Swipe
[region < c2]
Cont. Swipe
[region < c3]
Cont. Swipe

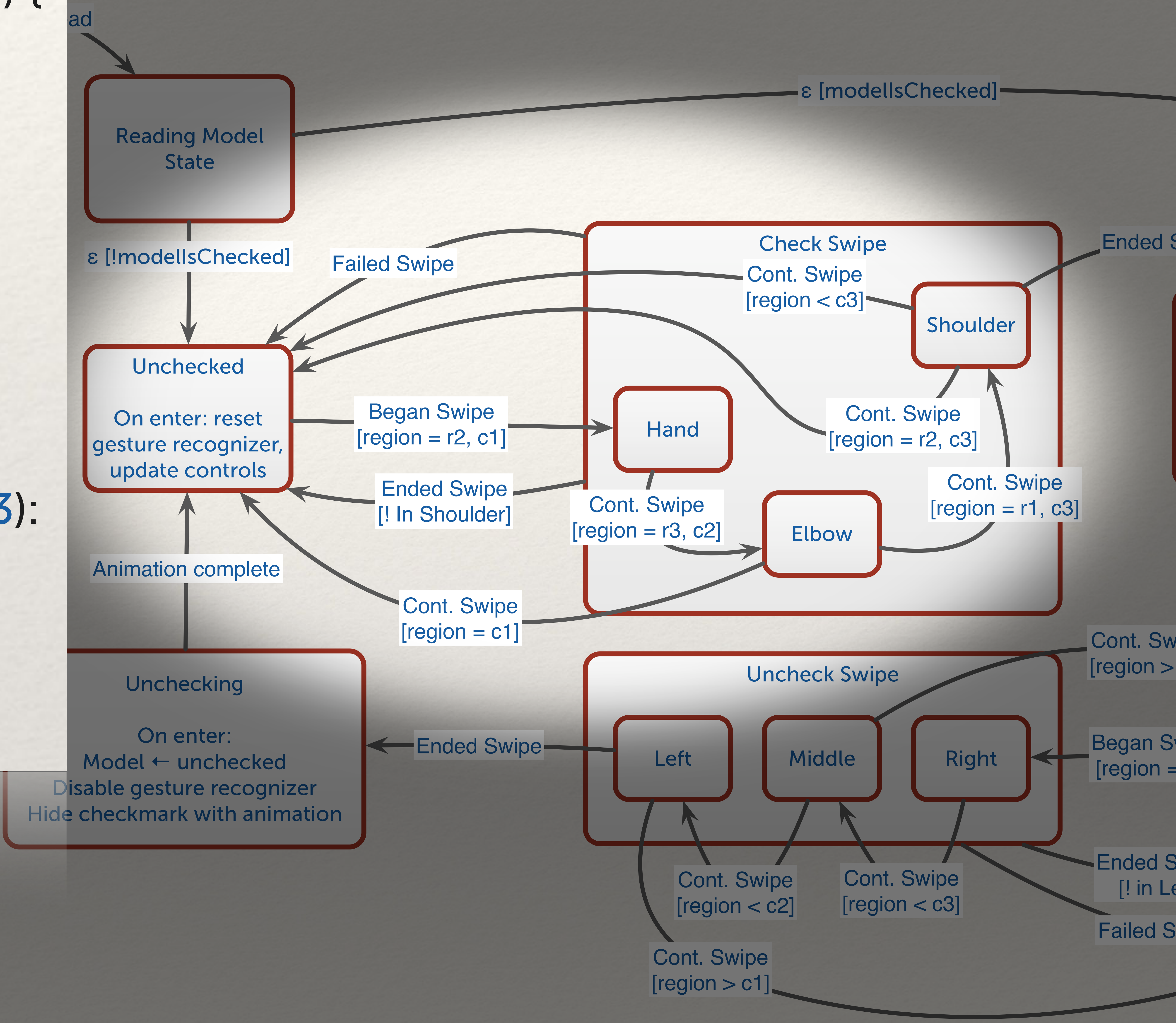
```
private func failedSwipe() {
   switch state {
   case .checkSwipe:
      state = .unchecked
   ...
   }
}

private func endedSwipe() {
   switch state {
   case .checkSwipe(substate: .shoulder):
      ...
   case .checkSwipe:
      state = .unchecked
   ...
   }
}
```



```
private func continuedSwipe(in region: PanRegion) {
    switch state {
    case .checkSwipe(substate: .hand)
        where region == PanRegion(row:3, column: 2):
        state = .checkSwipe(substate: .elbow)
    case .checkSwipe(substate: .elbow)
        where region == PanRegion(row: 1, column: 3):
        state = .checkSwipe(substate: .shoulder)
    ...
    }
}
```

Initial
Reading Model State
c [modelIsChecked]
Check Swipe
Cont. Swipe [region < c3]
Shoulder
Ended S
Hand
Cont. Swipe [region = r2, c3]
ipe
, c1]
wipe
ulder]
Cont. Swipe [region = r3, c2]
Elbow
Cont. Swipe [region = r1, c3]
Cont. Swipe [region = c1]
Unchecking
On enter:
Model ← unchecked
Disable gesture recognizer
Hide checkmark with animation
Ended Swipe
Uncheck Swipe
Left
Middle
Right
Cont. Sw
[region >
Began S
[region =
Cont. Swipe [region < c2]
Cont. Swipe [region < c3]
Ended S
[! in Le

```swift
private func continuedSwipe(in region: PanRegion) {
  switch state {
  ...
  case .checkSwipe(substate: .elbow)
    where region.column == 1:
    state = .unchecked
  case .checkSwipe(substate: .shoulder)
    where region.column < 3:
    state = .unchecked
  case .checkSwipe(substate: .shoulder)
    where region == PanRegion(row: 2, column: 3):
    state = .unchecked
  ...
  }
}
```



ad
Reading Model State
ε [modelIsChecked]
ε [!modelIsChecked]
Failed Swipe
Check Swipe
Cont. Swipe [region < c3]
Ended S
Shoulder
Unchecked
On enter: reset gesture recognizer, update controls
Began Swipe [region = r2, c1]
Hand
Cont. Swipe [region = r2, c3]
Ended Swipe [! In Shoulder]
Cont. Swipe [region = r3, c2]
Cont. Swipe [region = r1, c3]
Elbow
Animation complete
Cont. Swipe [region = c1]
Cont. Sw [region >
Unchecking
On enter: Model ← unchecked Disable gesture recognizer Hide checkmark with animation
Uncheck Swipe
Ended Swipe
Left
Middle
Right
Began S [region =
Cont. Swipe [region < c2]
Cont. Swipe [region < c3]
Ended S [! in Le
Failed S
Cont. Swipe [region > c1]

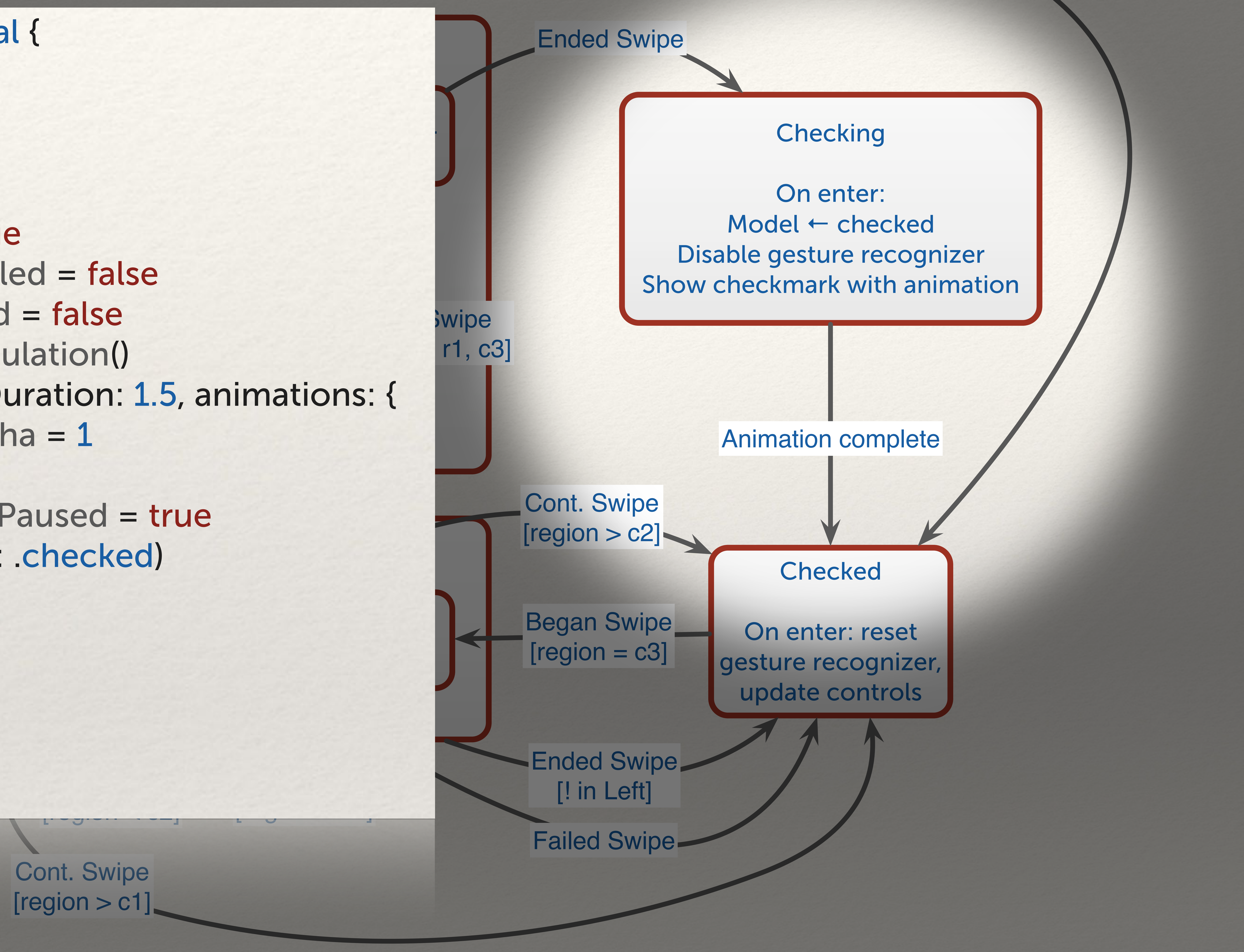
Reading Model State
ε [modelIsChecked]
Ended Swipe
Shoulder
Checking
On enter:
Model ← checked
Disable gesture recognizer
Show checkmark with animation
t. Swipe
n = r2, c3]
Cont. Swipe
[region = r1, c3]
[region = r3, c2]
Elbow
Animation complete
Animation complete
Cont. Swipe
[region = c1]
Cont. Swipe
[region > c2]
Uncheck Swipe
Checked
On enter: reset gesture recognizer, update controls
Unchecking
On enter:
Model ← unchecked
Disable gesture recognizer
Hide checkmark with animation
Ended Swipe
Left
Middle
Right
Began Swipe
[region = c3]
Cont. Swipe
[region < c2]
Cont. Swipe
[region < c3]
Ended Swipe
[! in Left]
Failed Swipe
Cont. Swipe
[region > c1]

```
private func endedSwipe() {
    switch state {
    case .checkSwipe(substate: .shoulder):
        state = .checking
    ...
    }
}
```

```
private var state: State = .initial {
    didSet {
        ...
        switch state {
        case .checking:
            modelIsChecked = true
            panRecognizer.isEnabled = false
            particleScene.isPaused = false
            particleNode.resetSimulation()
            UIView.animate(withDuration: 1.5, animations: {
                self.checkImage.alpha = 1
            }, completion: ...)
        ...
        }
    }
}
```

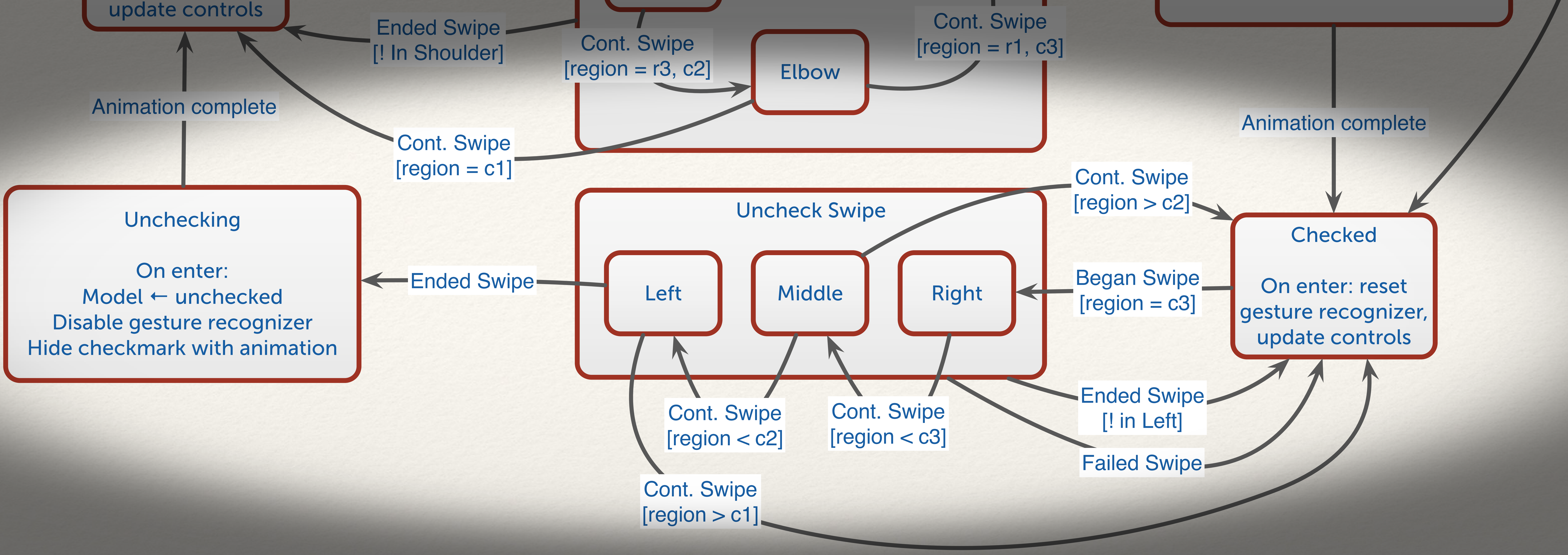
d]
ε [
ge
Ar
lder
nt. Swipe
on = r1, c3]
Ended Swipe
Checking
On enter:
Model ← checked
Disable gesture recognizer
Show checkmark with animation
Animation complete
Cont. Swipe
[region > c2]
Checked
On enter: reset gesture recognizer, update controls
Began Swipe
[region = c3]
On enter:
Model ← unchecked
Disable gesture recognizer
de checkmark with animation
Ended Swipe
Left
Middle
Right
Cont. Swipe
[region < c2]
Cont. Swipe
[region < c3]
Ended Swipe
[! in Left]
Failed Swipe
Cont. Swipe
[region > c1]

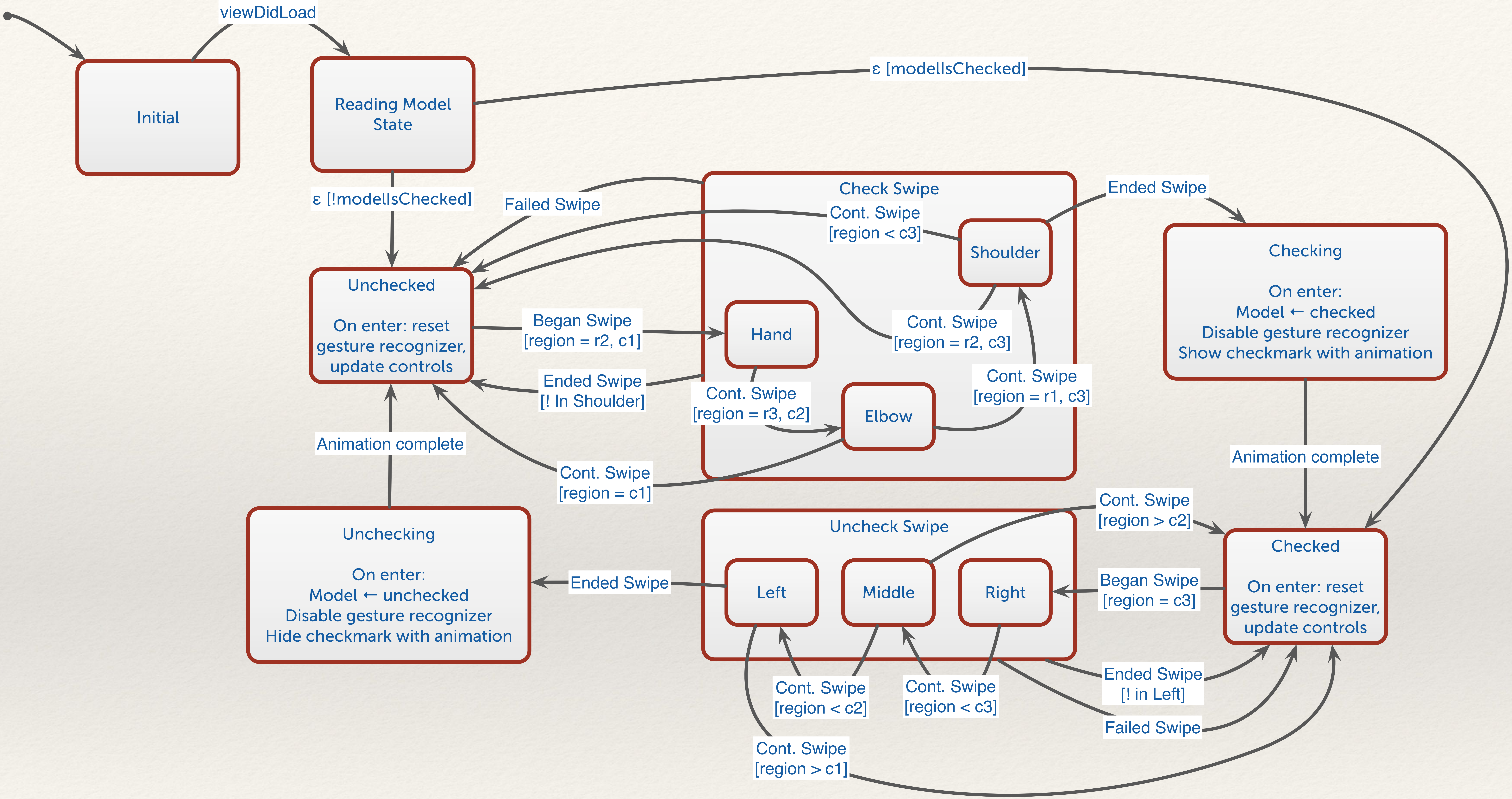
```
private var state: State = .initial {
    didSet {
        ...
        switch state {
        case .checking:
            modelIsChecked = true
            panRecognizer.isEnabled = false
            particleScene.isPaused = false
            particleNode.resetSimulation()
            UIView.animate(withDuration: 1.5, animations: {
                self.checkImage.alpha = 1
            }, completion: { _ in
                self.particleScene.isPaused = true
                self.thenSetState(to: .checked)
            })
        ...
        }
    }
}
```



```
private var state: State = .initial {
    didSet {
        ...
        switch state {
        ...
        case .checked:
            resetGestureRecognizer()
            updateControls()
        ...
        }
    }
}
```

Ended Swipe
[! In Shoulder]
Cont. Swipe
[region = r3, c2]
Elbow
Cont. Swipe
[region = r1, c3]
on
Animation complete
Cont. Swipe
[region > c2]
Checked
On enter: reset gesture recognizer, update controls
Began Swipe
[region = c3]
Ended Swipe
[! in Left]
Failed Swipe
Cont. Swipe
[region > c1]

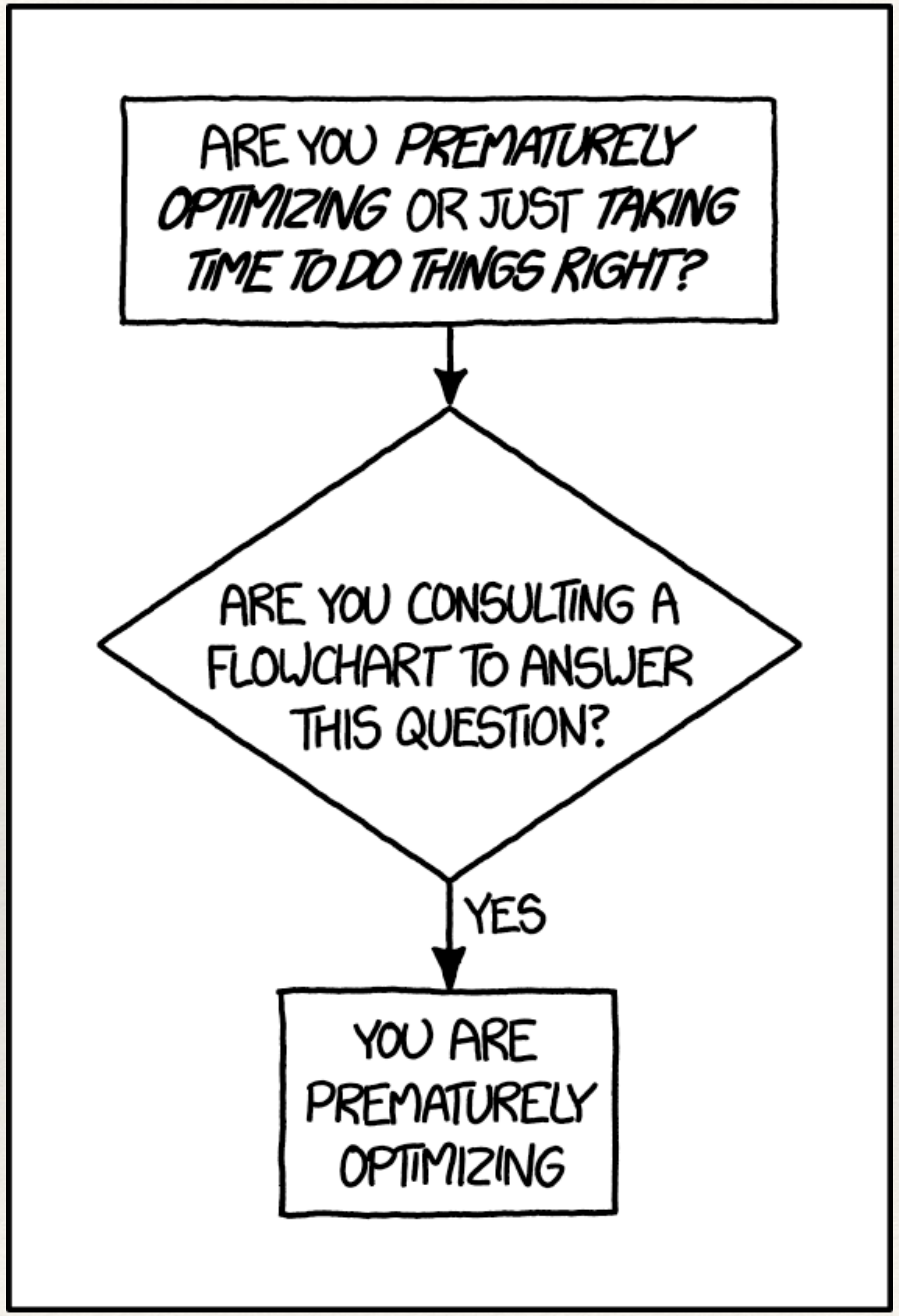

update controls
Ended Swipe
[! In Shoulder]
Cont. Swipe
[region = r3, c2]
Elbow
Cont. Swipe
[region = r1, c3]
Animation complete
Cont. Swipe
[region = c1]
Animation complete
Cont. Swipe
[region > c2]
Unchecking
On enter:
Model ← unchecked
Disable gesture recognizer
Hide checkmark with animation
Ended Swipe
Uncheck Swipe
Left
Middle
Right
Began Swipe
[region = c3]
Checked
On enter: reset
gesture recognizer,
update controls
Ended Swipe
[! in Left]
Cont. Swipe
[region < c2]
Cont. Swipe
[region < c3]
Failed Swipe
Cont. Swipe
[region > c1]


viewDidLoad
Initial
Reading Model State
ε [modelIsChecked]
ε [!modelIsChecked]
Failed Swipe
Check Swipe
Cont. Swipe [region < c3]
Shoulder
Ended Swipe
Checking
On enter:
Model ← checked
Disable gesture recognizer
Show checkmark with animation
Unchecked
On enter: reset gesture recognizer, update controls
Began Swipe [region = r2, c1]
Hand
Cont. Swipe [region = r2, c3]
Ended Swipe [! In Shoulder]
Cont. Swipe [region = r3, c2]
Elbow
Cont. Swipe [region = r1, c3]
Animation complete
Cont. Swipe [region = c1]
Animation complete
Cont. Swipe [region > c2]
Unchecking
On enter:
Model ← unchecked
Disable gesture recognizer
Hide checkmark with animation
Uncheck Swipe
Checked
On enter: reset gesture recognizer, update controls
Ended Swipe
Left
Middle
Right
Began Swipe [region = c3]
Cont. Swipe [region < c2]
Cont. Swipe [region < c3]
Ended Swipe [! in Left]
Failed Swipe
Cont. Swipe [region > c1]

Optimization

Cartoon of the Day

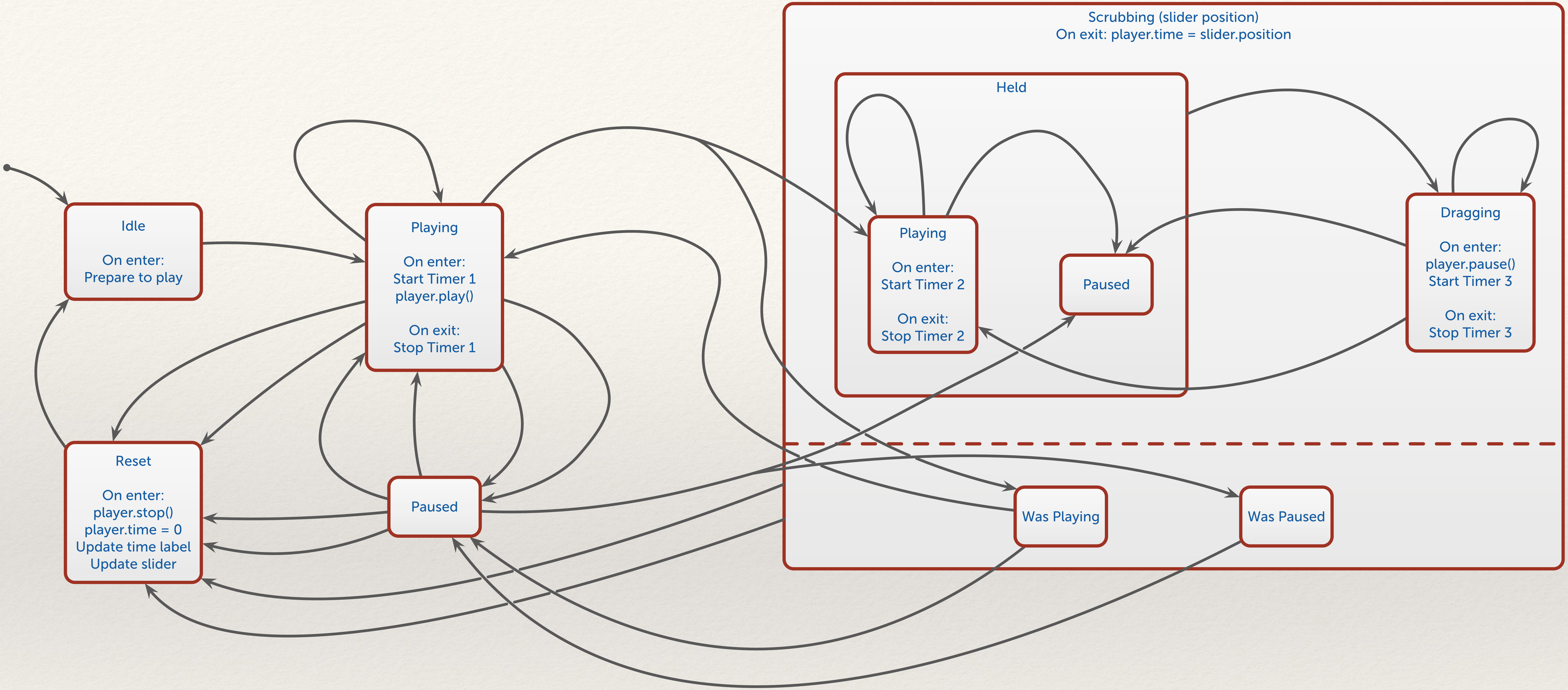
Premature optimization is the root of all evil, so to start this project I'd better come up with a system that can determine whether a possible optimization is premature or not.

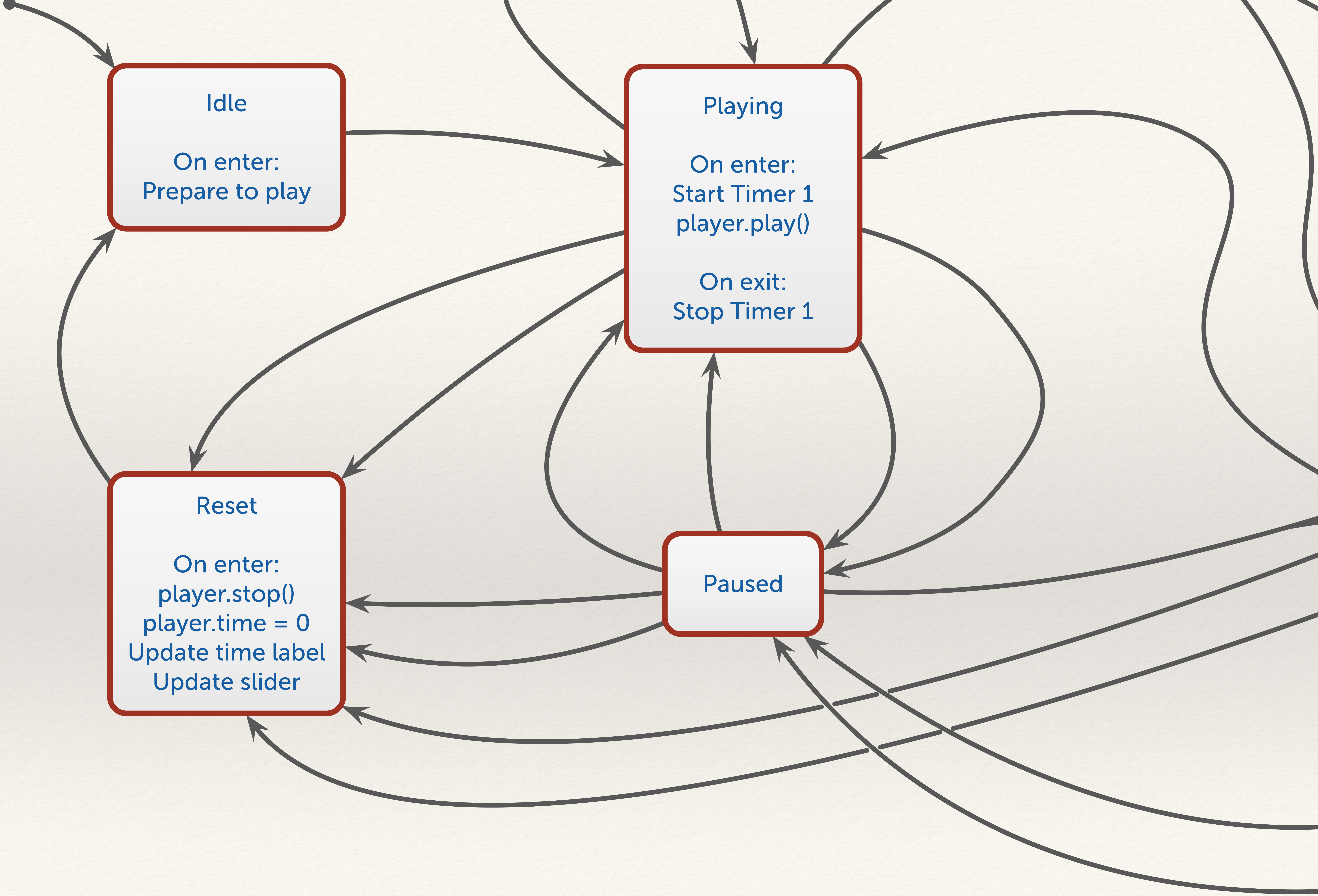

ARE YOU PREMATURELY OPTIMIZING OR JUST TAKING TIME TO DO THINGS RIGHT?
ARE YOU CONSULTING A FLOWCHART TO ANSWER THIS QUESTION?
YES
YOU ARE PREMATURELY OPTIMIZING

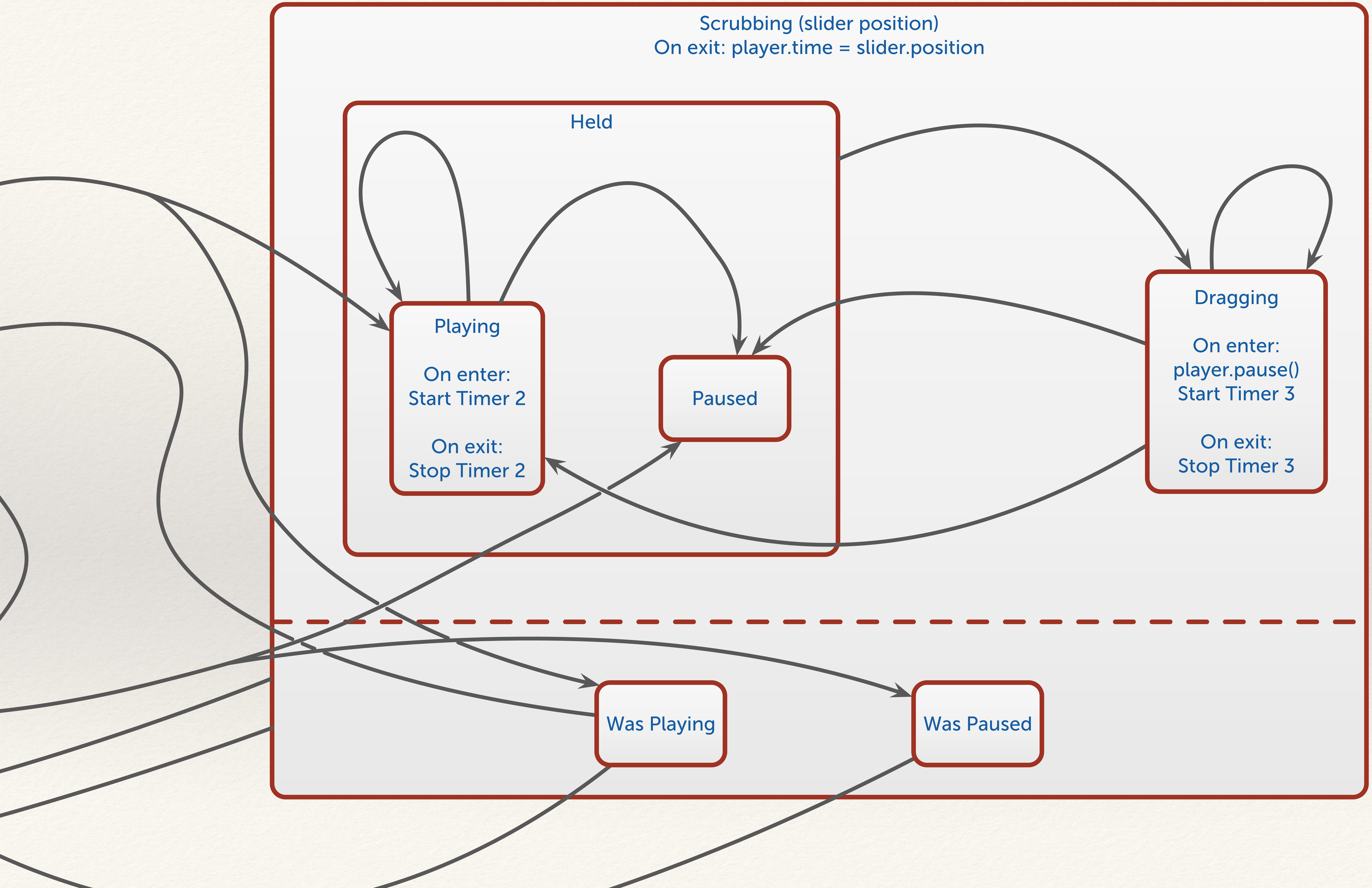
https://xkcd.com/1691/

Audio Player Example

Demo

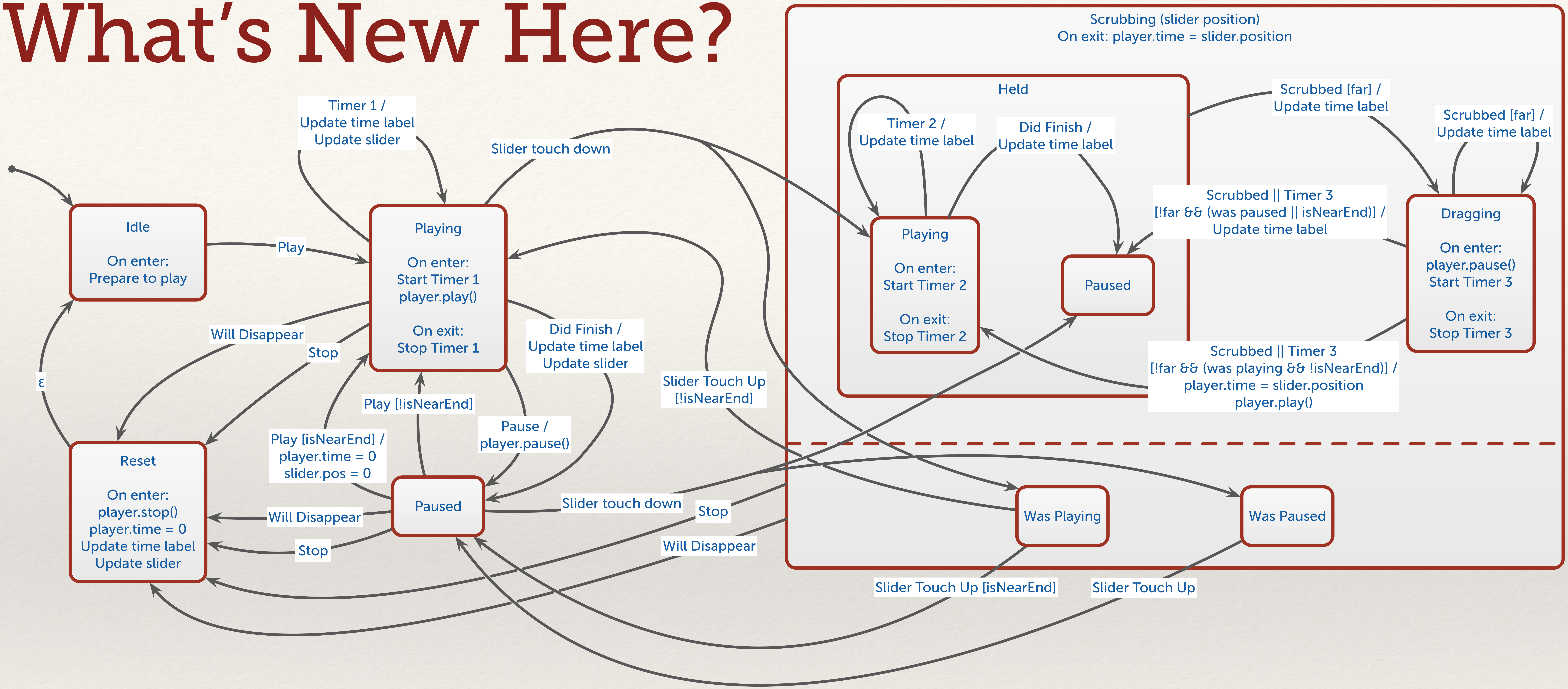

Scrubbing (slider position)
On exit: player.time = slider.position
Held
Idle
On enter:
Prepare to play
Playing
On enter:
Start Timer 1
player.play()
On exit:
Stop Timer 1
Playing
On enter:
Start Timer 2
On exit:
Stop Timer 2
Paused
Dragging
On enter:
player.pause()
Start Timer 3
On exit:
Stop Timer 3
Reset
On enter:
player.stop()
player.time = 0
Update time label
Update slider
Paused
Was Playing
Was Paused


Idle
On enter:
Prepare to play
Playing
On enter:
Start Timer 1
player.play()
On exit:
Stop Timer 1
Reset
On enter:
player.stop()
player.time = 0
Update time label
Update slider
Paused

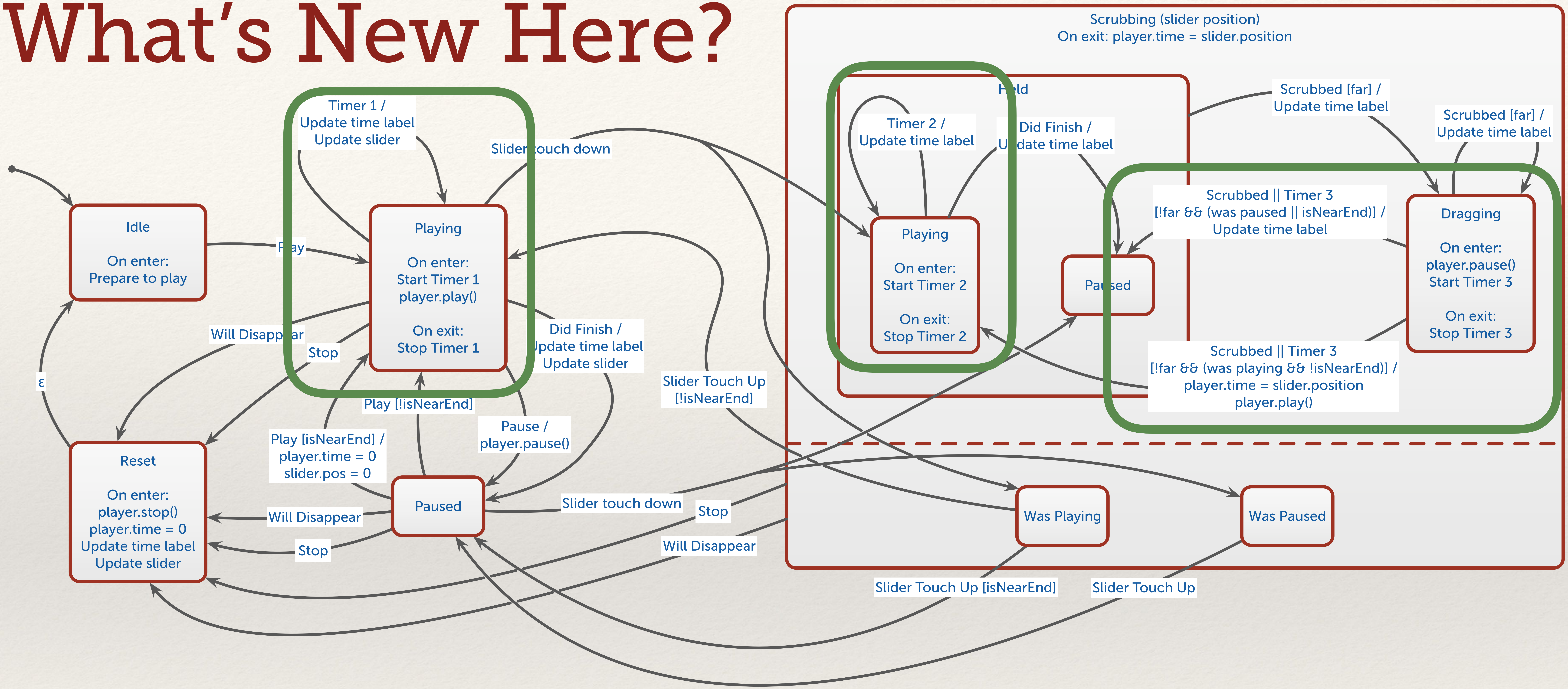


Scrubbing (slider position)
On exit: player.time = slider.position
Held
Playing
On enter:
Start Timer 2
On exit:
Stop Timer 2
Paused
Dragging
On enter:
player.pause()
Start Timer 3
On exit:
Stop Timer 3
Was Playing
Was Paused

What’s New Here?



What’s New Here?



Scrubbing (slider position)
On exit: player.time = slider.position

Held

Idle

On enter:
Prepare to play

Play

Timer 1 /
Update time label
Update slider

Playing

On enter:
Start Timer 1
player.play()

On exit:
Stop Timer 1

Slider touch down

Did Finish /
Update time label
Update slider

Will Disappear

Stop

ε

Play [!isNearEnd]

Play [isNearEnd] /
player.time = 0
slider.pos = 0

Pause /
player.pause()

Reset

On enter:
player.stop()
player.time = 0
Update time label
Update slider

Paused

Will Disappear

Stop

Slider touch down

Stop

Will Disappear

Slider Touch Up
[!isNearEnd]

Timer 2 /
Update time label

Playing

On enter:
Start Timer 2

On exit:
Stop Timer 2

Did Finish /
Update time label

Paused

Scrubbed [far] /
Update time label

Scrubbed [far] /
Update time label

Scrubbed || Timer 3
[!far && (was paused || isNearEnd)] /
Update time label

Dragging

On enter:
player.pause()
Start Timer 3

On exit:
Stop Timer 3

Scrubbed || Timer 3
[!far && (was playing && !isNearEnd)] /
player.time = slider.position
player.play()

Was Playing

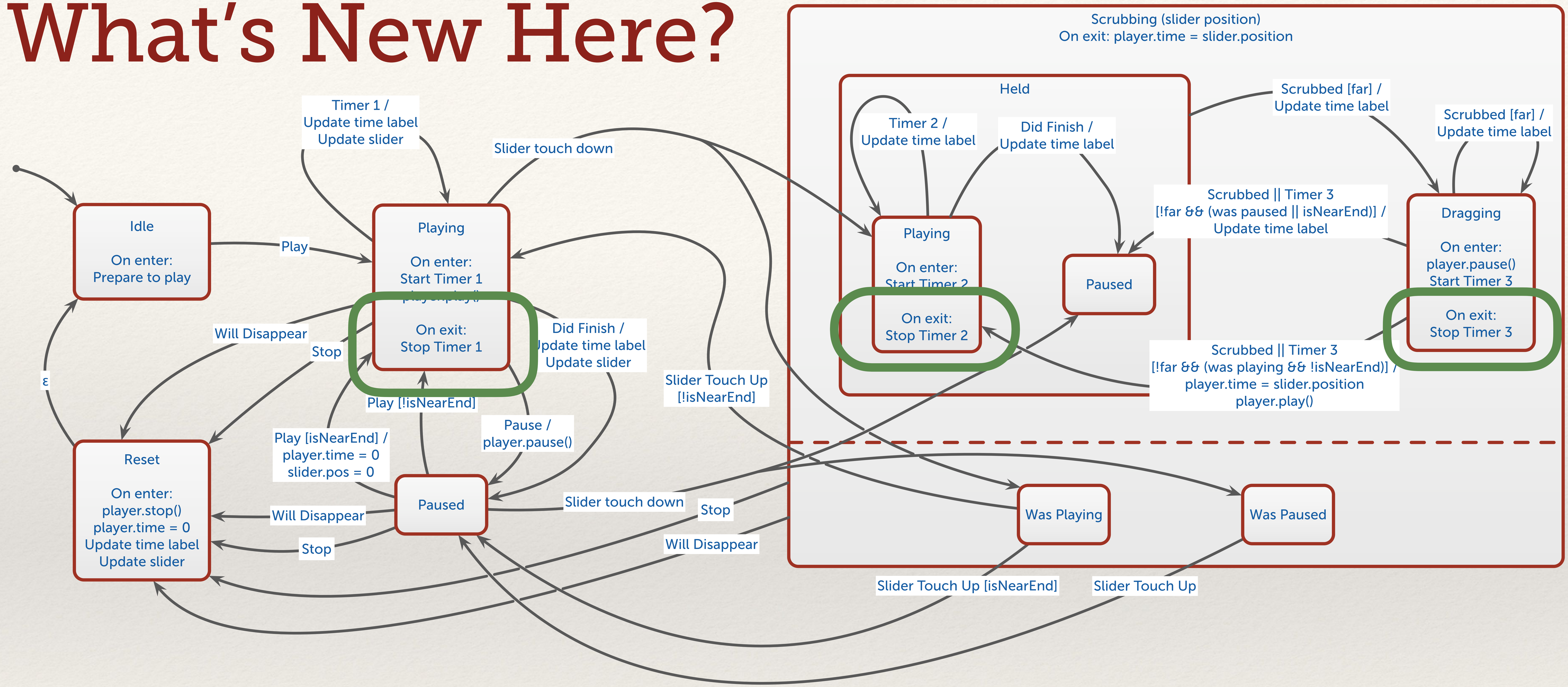
Was Paused

Slider Touch Up [isNearEnd]

Slider Touch Up

Timers

What's New Here?



Idle

On enter:
Prepare to play

Play

Timer 1 /
Update time label
Update slider

Slider touch down

Playing

On enter:
Start Timer 1
player.play()

On exit:
Stop Timer 1

Will Disappear

Stop

ε

Did Finish /
Update time label
Update slider

Slider Touch Up
[!isNearEnd]

Play [!isNearEnd]

Play [isNearEnd] /
player.time = 0
slider.pos = 0

Pause /
player.pause()

Reset

On enter:
player.stop()
player.time = 0
Update time label
Update slider

Will Disappear

Stop

Paused

Slider touch down

Stop

Will Disappear

Scrubbing (slider position)
On exit: player.time = slider.position

Held

Timer 2 /
Update time label

Did Finish /
Update time label

Playing

On enter:
Start Timer 2

On exit:
Stop Timer 2

Paused

Scrubbed [far] /
Update time label

Scrubbed [far] /
Update time label

Scrubbed || Timer 3
[!far && (was paused || isNearEnd)] /
Update time label

Dragging

On enter:
player.pause()
Start Timer 3

On exit:
Stop Timer 3

Scrubbed || Timer 3
[!far && (was playing && !isNearEnd)] /
player.time = slider.position
player.play()

Was Playing

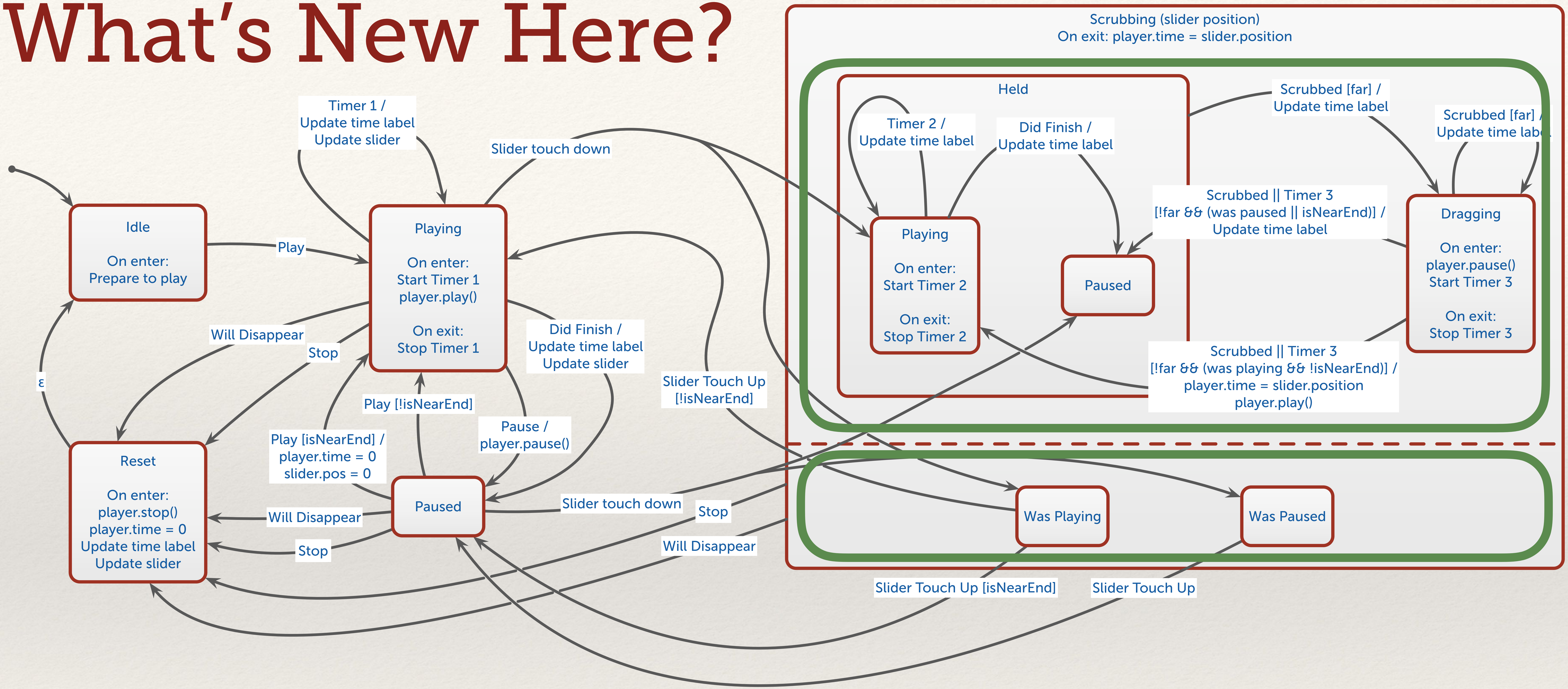
Was Paused

Slider Touch Up [isNearEnd]

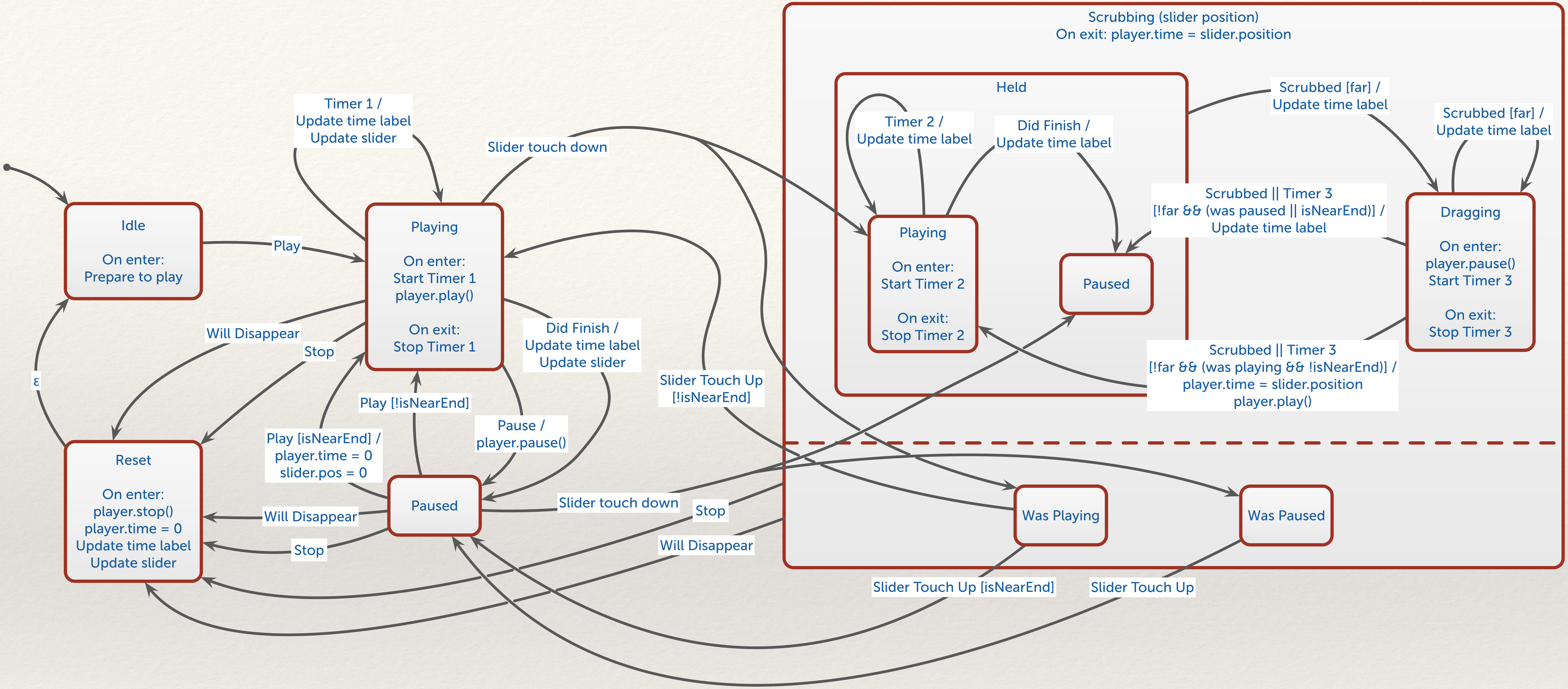
Slider Touch Up

On-exit Actions

What’s New Here?



Parallel Substates



Idle

On enter:
Prepare to play

Play

Playing

On enter:
Start Timer 1
player.play()

On exit:
Stop Timer 1

Timer 1 /
Update time label
Update slider

Slider touch down

Will Disappear

Stop

ε

Reset

On enter:
player.stop()
player.time = 0
Update time label
Update slider

Did Finish /
Update time label
Update slider

Play [!isNearEnd]

Pause /
player.pause()

Play [isNearEnd] /
player.time = 0
slider.pos = 0

Paused

Will Disappear

Stop

Slider touch down

Stop

Will Disappear

Slider Touch Up
[!isNearEnd]

Scrubbing (slider position)
On exit: player.time = slider.position

Held

Timer 2 /
Update time label

Did Finish /
Update time label

Playing

On enter:
Start Timer 2

On exit:
Stop Timer 2

Paused

Scrubbed [far] /
Update time label

Scrubbed [far] /
Update time label

Scrubbed || Timer 3
[!far && (was paused || isNearEnd)] /
Update time label

Dragging

On enter:
player.pause()
Start Timer 3

On exit:
Stop Timer 3

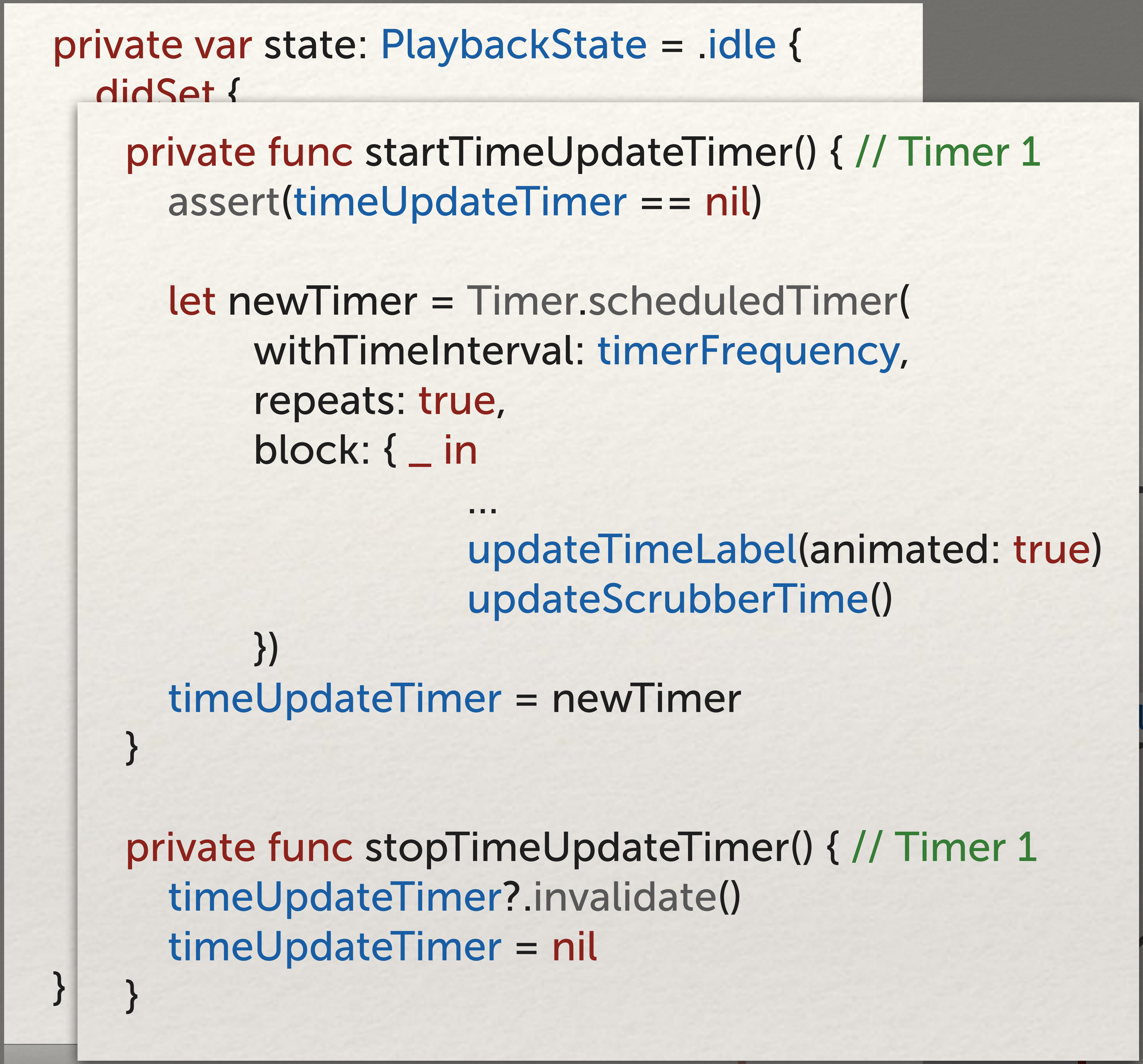
Scrubbed || Timer 3
[!far && (was playing && !isNearEnd)] /
player.time = slider.position
player.play()

Was Playing

Was Paused

Slider Touch Up [isNearEnd]

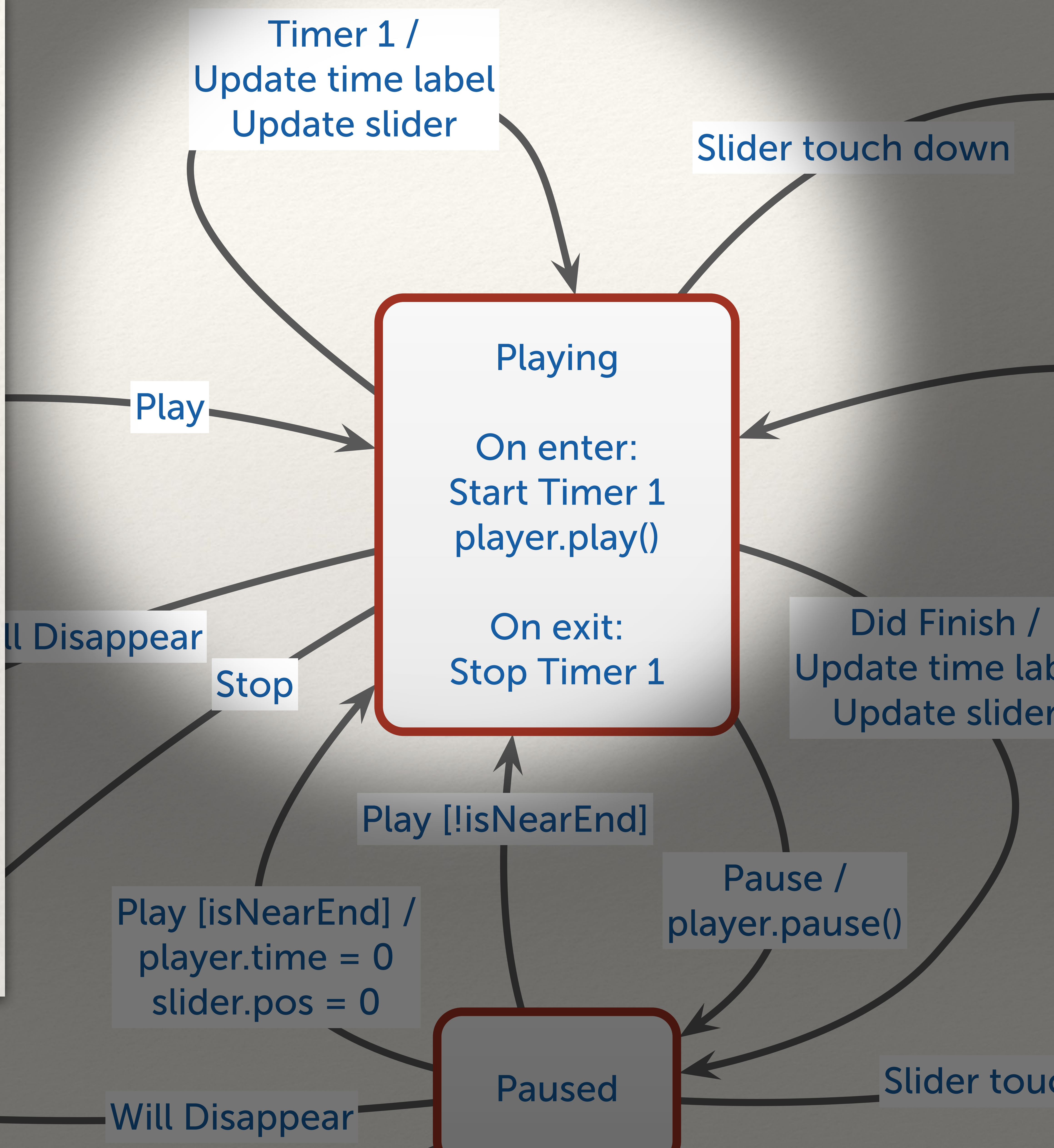
Slider Touch Up

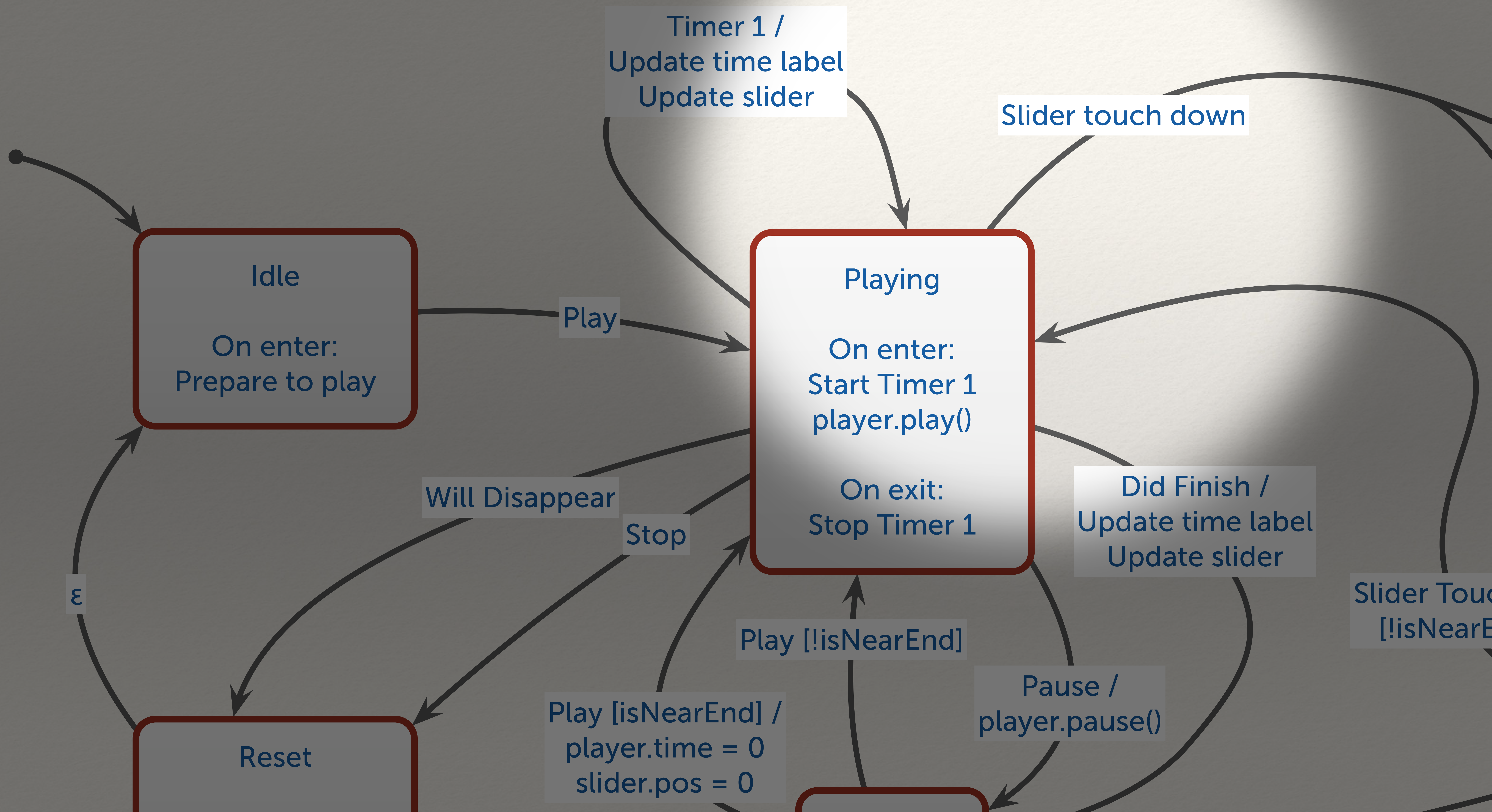


```
private var state: PlaybackState = .idle {
    didSet {
```

```
private func startTimeUpdateTimer() { // Timer 1
    assert(timeUpdateTimer == nil)

    let newTimer = Timer.scheduledTimer(
        withTimeInterval: timerFrequency,
        repeats: true,
        block: { _ in
                    ...
                    updateTimeLabel(animated: true)
                    updateScrubberTime()
        })
    timeUpdateTimer = newTimer
}

private func stopTimeUpdateTimer() { // Timer 1
    timeUpdateTimer?.invalidate()
    timeUpdateTimer = nil
}
```

```
}
```





Idle

On enter:
Prepare to play

Play

Playing

On enter:
Start Timer 1
player.play()

On exit:
Stop Timer 1

Timer 1 /
Update time label
Update slider

Slider touch down

Did Finish /
Update time label
Update slider

Slider Touc
[!isNearE

Will Disappear

Stop

ε

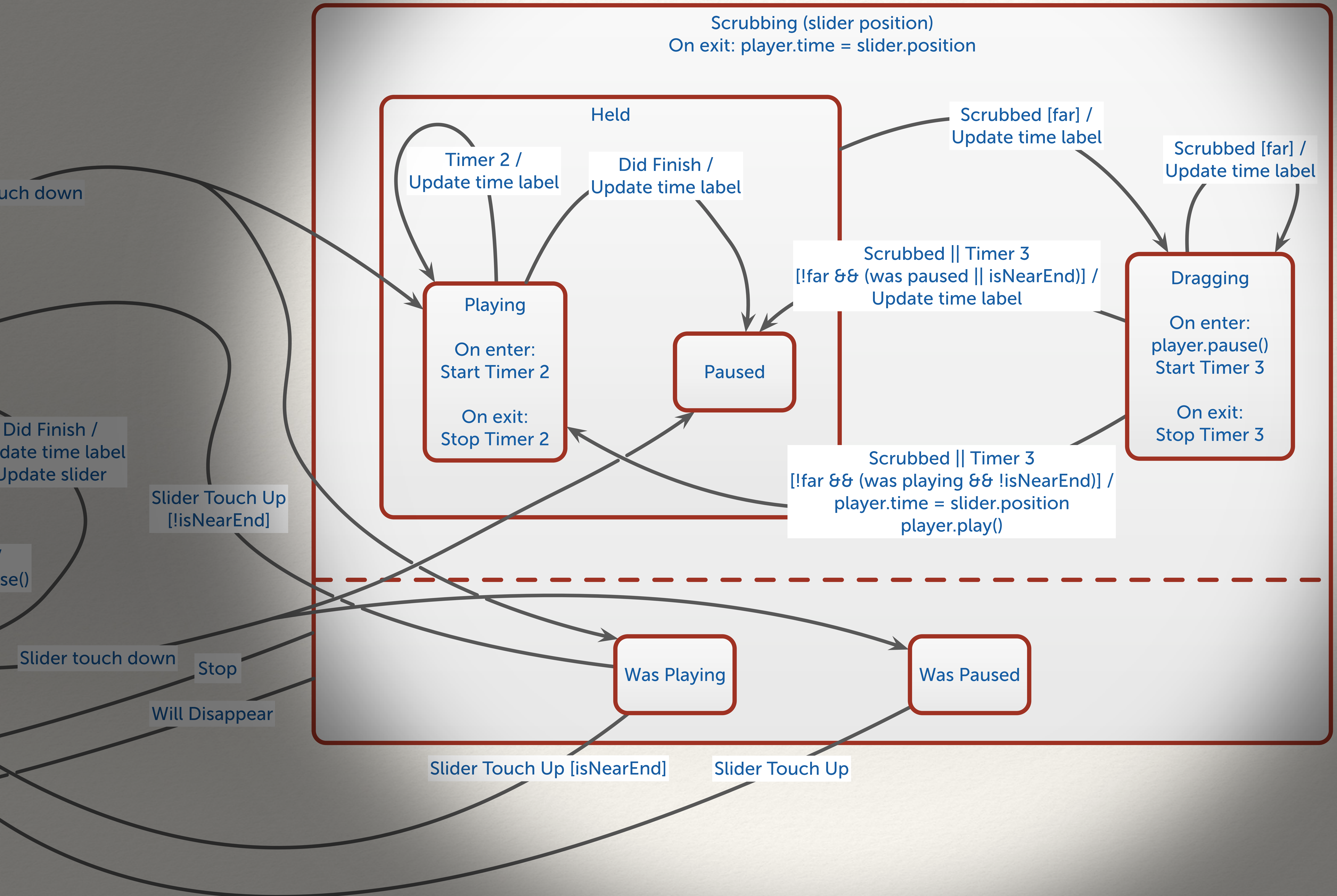
Play [!isNearEnd]

Play [isNearEnd] /
player.time = 0
slider.pos = 0

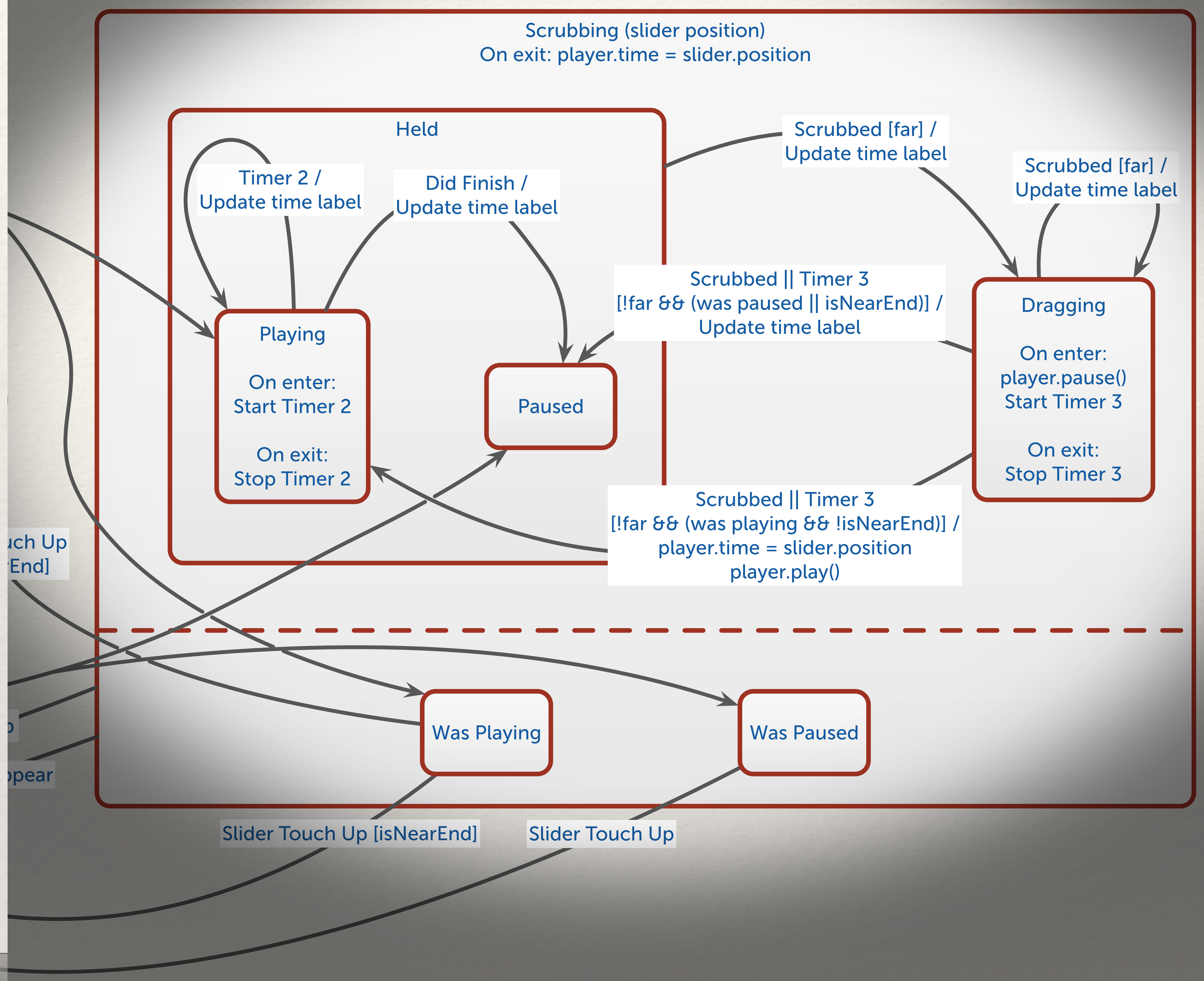
Pause /
player.pause()

Reset

On exit: playe
Held
Timer 2 /
Update time label
Did Finish /
Update time label
Playing
On enter:
Start Timer 2
On exit:
Stop Timer 2
Paused
Timer 1 /
Update time label
Update slider
Slider touch down
Idle
On enter:
Prepare to play
Play
Playing
On enter:
Start Timer 1
player.play()
On exit:
Stop Timer 1
Will Disappear
Stop
ε
Did Finish /
Update time label
Update slider
Slider Touch Up
[!isNearEnd]
Play [!isNearEnd]
Pause /
player.pause()
Play [isNearEnd] /
player.time = 0
slider.pos = 0
Reset
On enter:
player.stop()
player.time = 0
Update time label
Update slider
Paused
Will Disappear
Stop
Slider touch down
Stop
Will Disappear
Was Playing
Slider Touch Up [isNearEnd]
Slider To


Scrubbing (slider position)
On exit: player.time = slider.position
Held
Timer 2 /
Update time label
Did Finish /
Update time label
Playing
On enter:
Start Timer 2
On exit:
Stop Timer 2
Paused
Scrubbed [far] /
Update time label
Scrubbed [far] /
Update time label
Scrubbed || Timer 3
[!far && (was paused || isNearEnd)] /
Update time label
Dragging
On enter:
player.pause()
Start Timer 3
On exit:
Stop Timer 3
Scrubbed || Timer 3
[!far && (was playing && !isNearEnd)] /
player.time = slider.position
player.play()
uch down
Did Finish /
date time label
Update slider
Slider Touch Up
[!isNearEnd]
/
se()
Slider touch down
Stop
Will Disappear
Was Playing
Was Paused
Slider Touch Up [isNearEnd]
Slider Touch Up

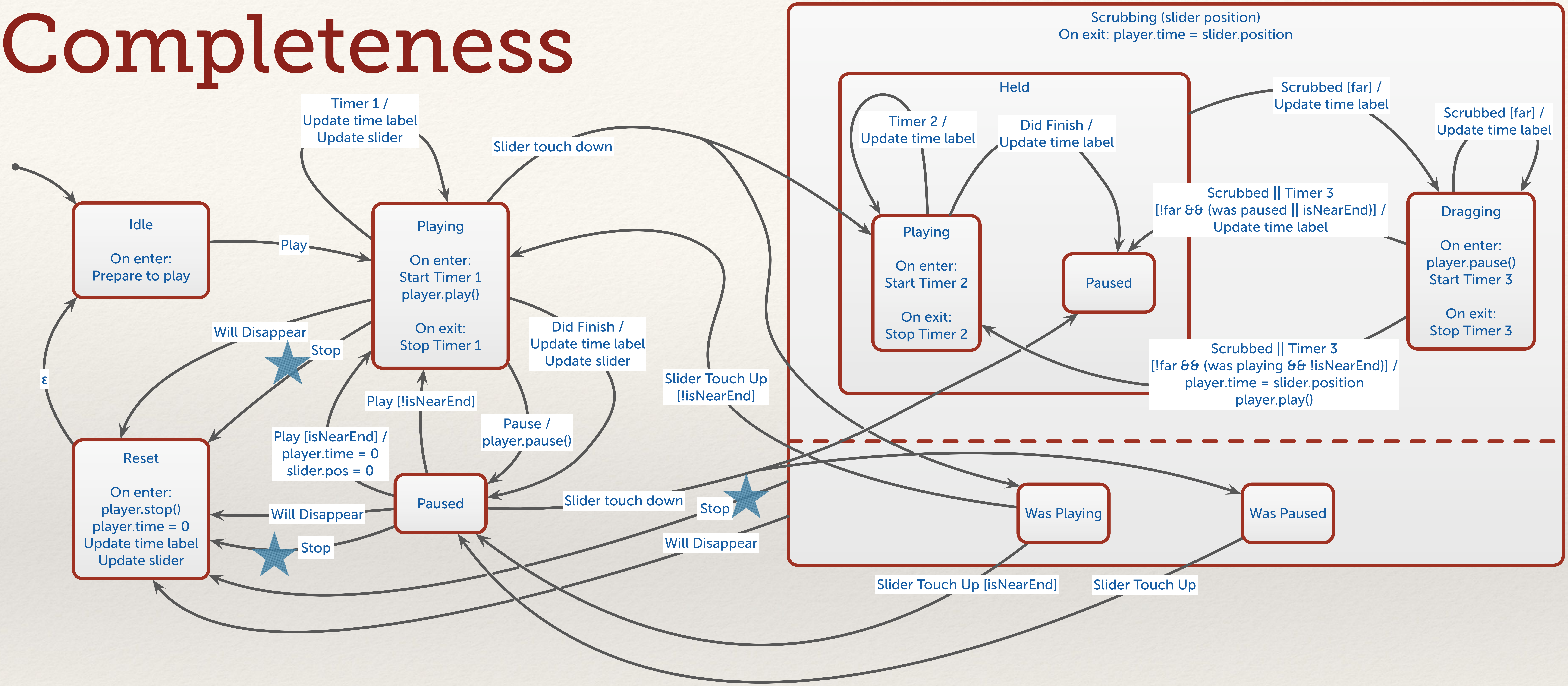
```
private enum PlaybackState {
    case reset
    case idle
    case playing
    case paused
    case scrubbing(motion: ScrubbingMotion,
                   history: PlayingSubstate,
                   position: PlaybackPosition)
}

private enum ScrubbingMotion {
    case held(PlayingSubstate)
    case dragging
}

private enum PlayingSubstate {
    case playing
    case paused
}
```



```
private func sliderTouchDown() {
    switch state {
    case .playing:
        state = .scrubbing(motion: .held(.playing),
                           history: .playing,
                           position: PlaybackPosition(slider:
                                                      playbackPosition))
    ...
    }
}
```

On exit: playe
Held
Timer 2 /
Update time label
Did Finish /
Update time label
Playing
On enter:
Start Timer 2
On exit:
Stop Timer 2
Paused
Timer 1 /
Update time label
Update slider
Slider touch down
Slider Touch Up
[!isNearEnd]
Was Playing
player.stop()
player.time = 0
Update time label
Update slider
Will Disappear
Stop
Paused
Slider touch down
Stop
Will Disappear
Slider Touch Up [isNearEnd]
Slider To

Completeness



Scrubbing (slider position)
On exit: player.time = slider.position

Held

Timer 1 /
Update time label
Update slider

Slider touch down

Idle

On enter:
Prepare to play

Play

Playing

On enter:
Start Timer 1
player.play()

On exit:
Stop Timer 1

Will Disappear

Stop

ε

Reset

On enter:
player.stop()
player.time = 0
Update time label
Update slider

Did Finish /
Update time label
Update slider

Slider Touch Up
[!isNearEnd]

Play [!isNearEnd]

Play [isNearEnd] /
player.time = 0
slider.pos = 0

Pause /
player.pause()

Paused

Will Disappear

Stop

Slider touch down

Stop

Will Disappear

Timer 2 /
Update time label

Did Finish /
Update time label

Playing

On enter:
Start Timer 2

On exit:
Stop Timer 2

Paused

Scrubbed [far] /
Update time label

Scrubbed [far] /
Update time label

Scrubbed || Timer 3
[!far && (was paused || isNearEnd)] /
Update time label

Dragging

On enter:
player.pause()
Start Timer 3

On exit:
Stop Timer 3

Scrubbed || Timer 3
[!far && (was playing && !isNearEnd)] /
player.time = slider.position
player.play()

Was Playing

Was Paused

Slider Touch Up [isNearEnd]

Slider Touch Up

Recap

Recap

- States

```
class ImageCollectionViewCell: UICollectionViewCell {
    private var state: State = .pending
    ...
}

private enum State {
    case pending
    case loading
    case thumbnail
    case full
}
```

Recap

- States
- Transitions

```
override func viewDidLoad() {
    super.viewDidLoad()
    ...
    state = .readingModelState
}
```

Recap

- States
- Transitions
- On enter/exit

```
private var state: State = .initial {
    didSet {
        guard state != oldValue else { return }

        switch state {
        ...
        case .unchecked:
            resetGestureRecognizer()
            updateControls()
        ...
    }
```

Recap

- States
- Transitions
- On enter/exit
- Epsilon transitions

```
private var state: State = .initial {
    didSet {
        ...
        switch state {
        case .readingModelState:
            ...
                self.thenSetState(to: .checked)
            ...
        }
        ...
    }
}
private func thenSetState(to state: State) {
    DispatchQueue.main.async {
        self.state = state
    }
}
```

Recap

- States
- Transitions
- On enter/exit
- Epsilon transitions
- Timers

```
private func startTimeUpdateTimer() { // Timer 1
    assert(timeUpdateTimer == nil)

    let newTimer = Timer.scheduledTimer(
            withTimeInterval: timerFrequency,
            repeats: true,
            block: { _ in
                        ...
                        updateTimeLabel(animated: true)
                        updateScrubberTime()
            })
    timeUpdateTimer = newTimer
}

private func stopTimeUpdateTimer() { // Timer 1
    timeUpdateTimer?.invalidate()
    timeUpdateTimer = nil
}
```

Recap

- States
- Transitions
- On enter/exit
- Epsilon transitions
- Timers
- Depth

```
private enum PlaybackState {
    case reset
    case idle
    case playing
    case paused
    case scrubbing(motion: ScrubbingMotion,
                   history: PlayingSubstate,
                   position: PlaybackPosition)
}

private enum ScrubbingMotion {
    case held(PlayingSubstate)
    case dragging
}

private enum PlayingSubstate {
    case playing
    case paused
}
```

Recap

- States
- Transitions
- On enter/exit
- Epsilon transitions
- Timers
- Depth
- Completeness

by Ostrich Farm, https://www.flickr.com/photos/ostrichfarm/8193063467

Thank you!

These are a Few of My Stateful Machines

Curt Clifton
The Omni Group
Twitter: @curtclifton
Web: www.curtclifton.net