UNIVERSITY OF
Southampton

UNIVERSITY OF Southampton

REST in Practice

COMP3220 Web Infrastructure

Dr Nicholas Gibbins – nmg@ecs.soton.ac.uk

Web Services as state machines

Consider a hypothetical online bookseller: Orinoco Books

When we create an order, the order may be in one of a number of discrete states:

- Open: we can add or remove items to our order
- Paid: we have successfully sent payment to Orinoco, and can no longer change our order
- Shipping: Orinoco is preparing and dispatching our order
- Delivered: we have received our order

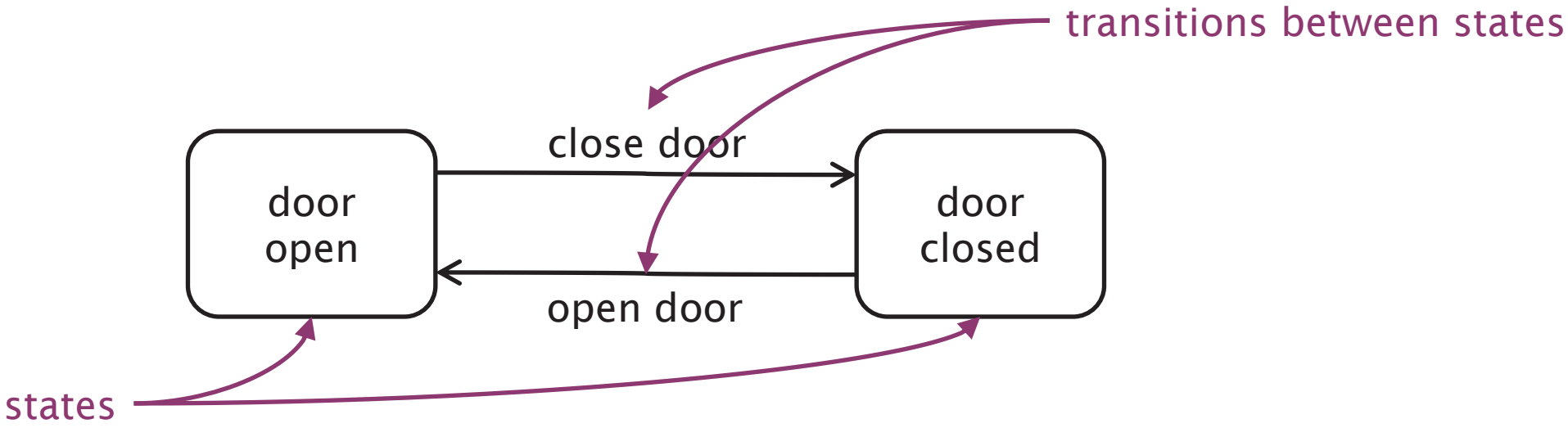
The order moves between states in response to our interactions with Orinoco

UML Statecharts: states and transitions

Common graphical notation for describing state machines

- Object-oriented extension to Harel's statechart
- (you'll need this for your coursework!)

Tip: label states with nouns or adjectives and transitions with verbs or verb phrases

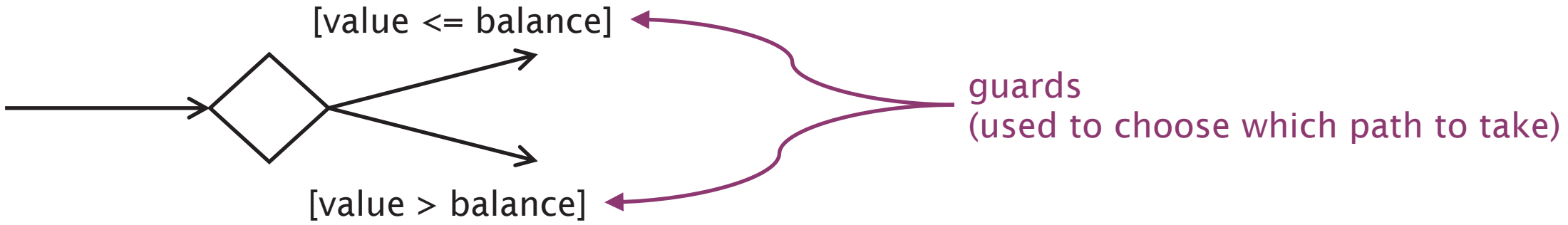


UML Statecharts: pseudostates

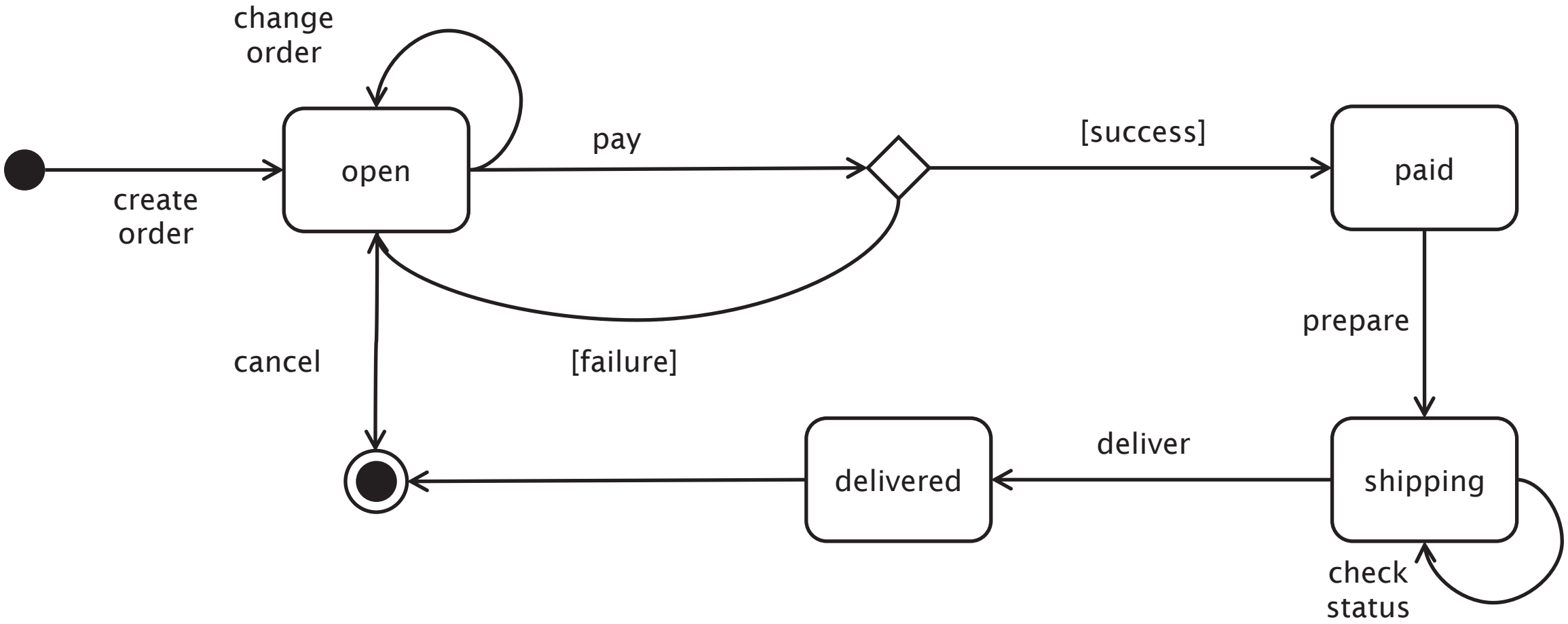
Two distinguished pseudostates:

- Initial state 
- Final state 

Choice pseudostate:



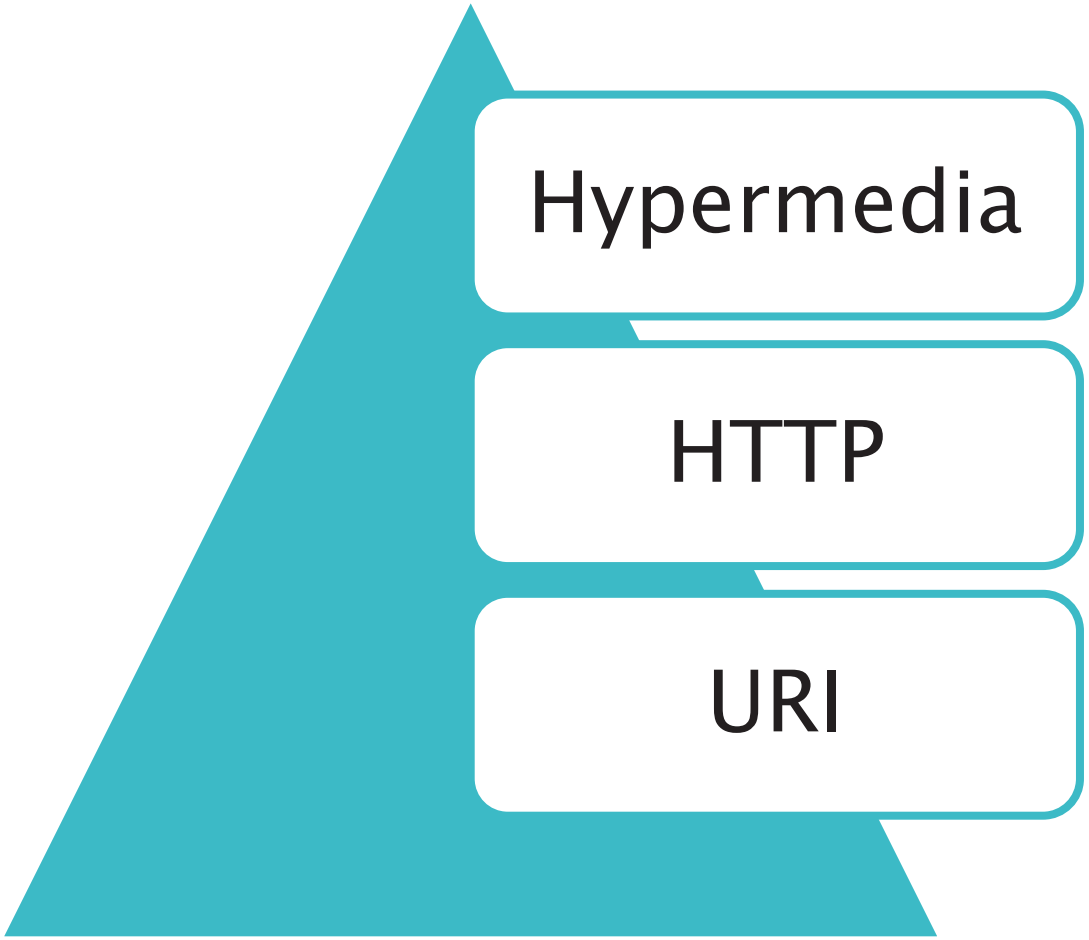
Orinoco Workflow



UNIVERSITY OF Southampton

Revisiting the Richardson Maturity Model

Richardson Maturity Model



Richardson Level 1

Multiple URIs used for resources

Key resource type from the workflow is an *order*

- http://orinoco.com/order/{order_id}

Richardson Level 2

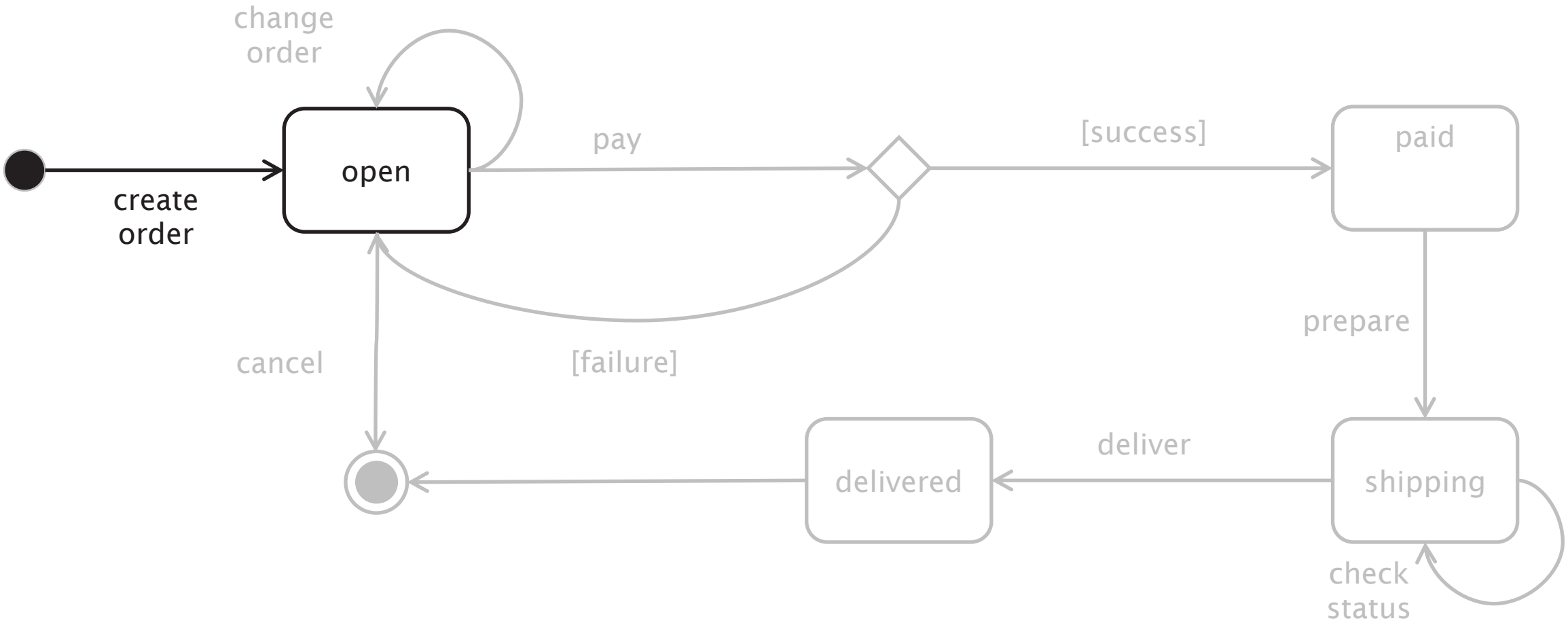
We have different URIs for each order (resource)

How do we interact with the orders?

- create a new order
- change order (add/remove items)
- cancel an order
- checkout and payment (submit order)
- check order status

Use appropriate HTTP methods!

Create an order



Create an order

Can use either PUT or POST:

PUT to a new URI

- new URI: http://orinoco.com/order/{order_id}
- client chooses order id

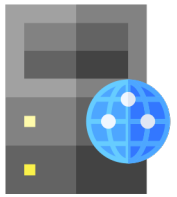
POST to an existing URI

- existing URI: http://orinoco.com/order/
- server chooses order id

PUT to a new URI

```
PUT /order/1234 HTTP/1.1
Host: orinoco.com
Content-Type: application/xml
Content-Length: 107

<order xmlns="http://schema.orinoco.com/order">
  <items>
  </items>
</order>
```

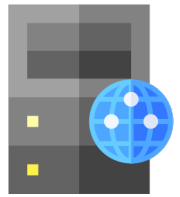
```
HTTP/1.1 201 Created
Date: Tue, 29 Oct 2019 17:10:00 GMT
Content-Length: 0
```



POST to an existing URI

```
POST /order/ HTTP/1.1
Host: orinoco.com
Content-Type: application/xml
Content-Length: 107

<order xmlns="http://schema.orinoco.com/order">
  <items>
  </items>
</order>
```

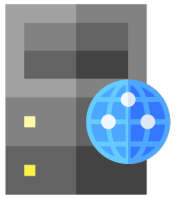
```
HTTP/1.1 201 Created
Location: /order/1234
Date: Tue, 29 Oct 2019 17:10:00 GMT
```



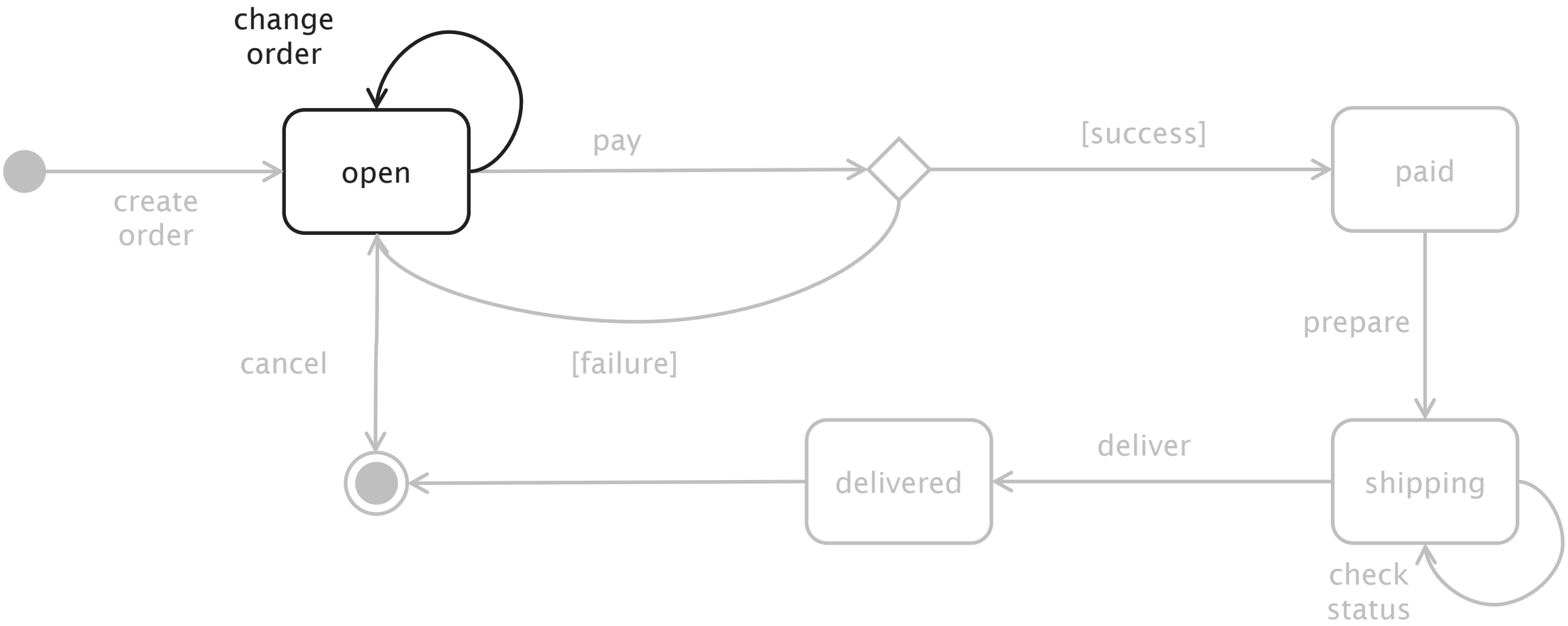
POST to an existing URI

```
POST /order/ HTTP/1.1
Host: orinoco.com
Content-Type: application/xml
Content-Length: 107

<order xmlns="http://schema.orinoco.com/order">
  <items>
  </items>
</order>
```

```
HTTP/1.1 201 Created
Content-Location: /order/1234
Date: Tue, 29 Oct 2019 17:10:00 GMT

<order xmlns="http://schema.orinoco.com/order">
  <items>
  </items>
</order>
```



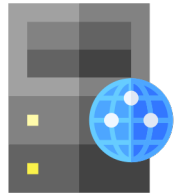
Change order



PUT to an existing URI

```
PUT /order/1234 HTTP/1.1
Host: orinoco.com
Content-Type: application/xml
Content-Length: 134

<order xmlns="http://schema.orinoco.com/order">
  <items>
    <item quantity="1" isbn="1234567890"/>
  </items>
</order>
```

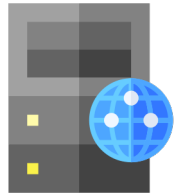
```
HTTP/1.1 200 OK
Date: Tue, 29 Oct 2019 17:15:00 GMT
```



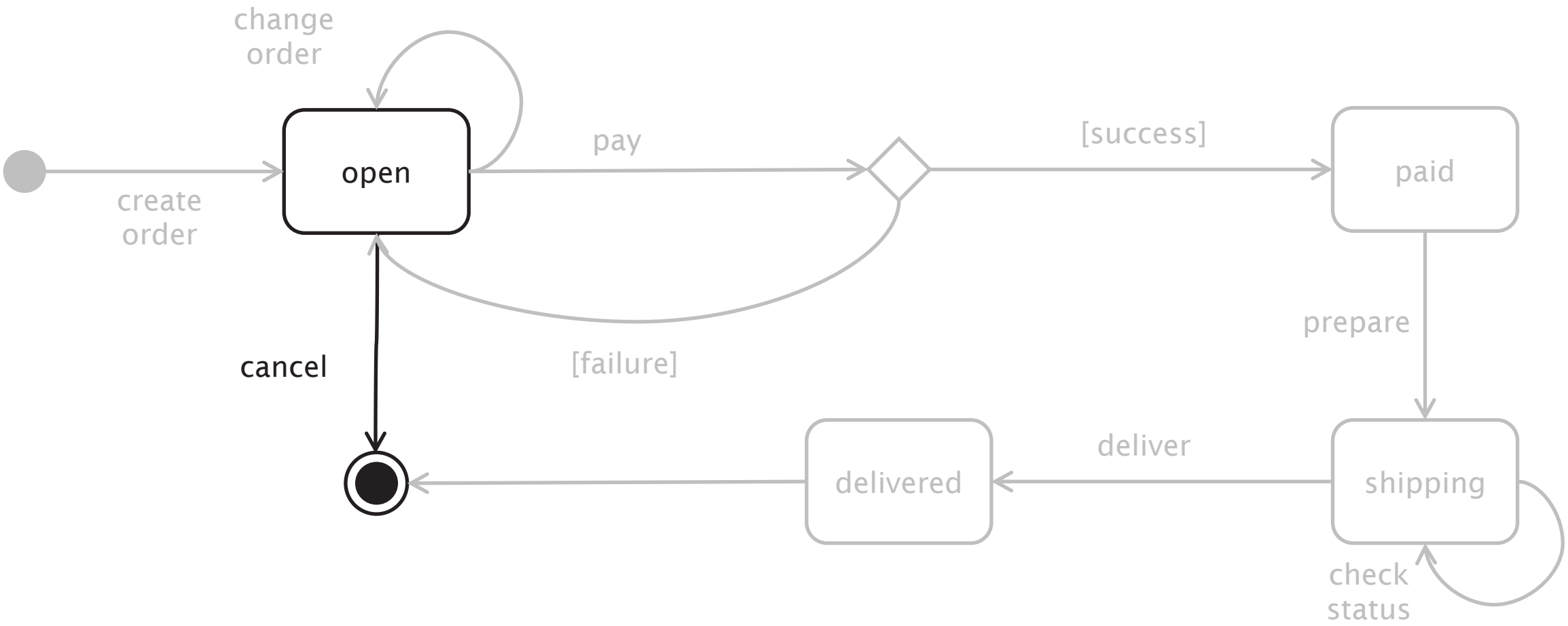
Conditional PUT

```
PUT /order/1234 HTTP/1.1
Host: orinoco.com
Content-Type: application/xml
Content-Length: 134
If-Unmodified-Since: Tue, 29 Oct 2019 17:15:00 GMT

<order xmlns="http://schema.orinoco.com/order">
  <items>
    <item quantity="1" isbn="1234567890"/>
  </items>
</order>
```

```
HTTP/1.1 412 Precondition Failed
Date: Tue, 29 Oct 2019 17:20:00 GMT
Content-Length: 0
```



Cancel an order



Cancel an order

Use DELETE

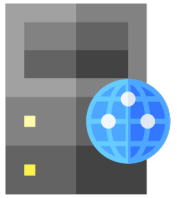
DELETE is idempotent

- Repeated DELETEs have the same effect as a single DELETE
- Status codes may change (e.g. 404 for subsequent DELETEs)

DELETE

```
DELETE /order/1234 HTTP/1.1
Host: orinoco.com
```

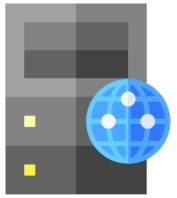
```
HTTP/1.1 204 No Content
Content-Length: 0
Date: Tue, 29 Oct 2019 17:25:00 GMT
```



DELETE

```
DELETE /order/1234 HTTP/1.1
Host: orinoco.com
```

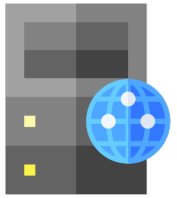
```
HTTP/1.1 404 Not Found
Content-Length: 0
Date: Tue, 29 Oct 2019 17:25:00 GMT
```



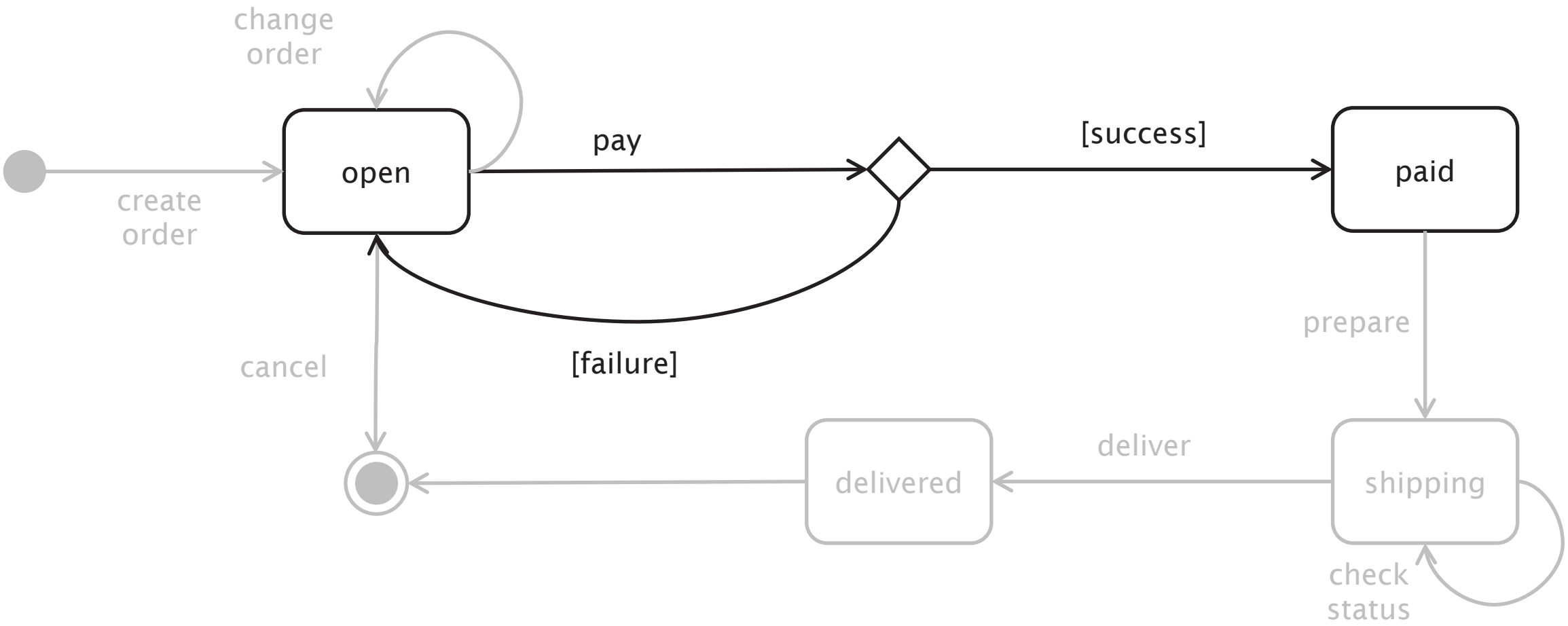
DELETE

```
DELETE /order/1234 HTTP/1.1
Host: orinoco.com
```

```
HTTP/1.1 410 Gone
Content-Length: 0
Date: Tue, 29 Oct 2019 17:25:00 GMT
```



Payment


change order
open
pay
[success]
paid
create order
prepare
cancel
[failure]
deliver
delivered
shipping
check status

Richardson Level Three

CRUD isn't everything!

- Limited application model
- In our scenario, payment doesn't fit cleanly into the CRUD model
- Encourages tight coupling through URI templates
- Simple pattern

Use hypertext links to indicate protocols

- What are the next steps that you can take?
- What are the next resources?

Where are the links?

```
<order xmlns="http://schema.orinoco.com/order">
  <items>
    <item quantity="1" isbn="1234567890"/>
  </items>
  <status>open</status>
</order>
```

What can you do next?

Media Types

application/xml doesn't have specific link semantics

Can adopt standard hypermedia format (HTML, Atom, etc)

- Widely understood by software agents
- Needs to be adapted to domain

Can create domain-specific format that supports application

- Direct supports domain
- Maintains visibility of messages at the protocol level
- Not widely understood

Use link types to define protocols

text/html

Use OPTIONS to determine the right HTTP method to use with links

- `Allow:` header in response lists allowed methods (for payment, PUT?)

Need to define link types for use with `rel`: microformats, RDF, etc

```
<html xmlns="http://www.w3.org/1999/xhtml”>
  <body>
    <div class="order”>
      <ul class="items”>
        <li class="item”>
          <p class=”isbn">1234567890</p>
          <p class="quantity">1</p>
        </li>
      </ul>
      <a href="https://orinoco.com/payment/1234” rel="payment">payment</a>
    </div>
  </body>
</html>
```

application/vnd.orinoco+xml

Proprietary (vendor-specific) media type

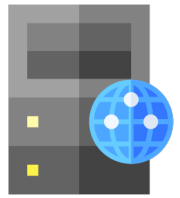
- Uses POX for business data
- Uses (e.g.) Atom link elements for hypermedia control

```
<order xmlns="http://schema.orinoco.com/order">
  <items>
    <item quantity="1" isbn="1234567890"/>
  </items>
  <link href="https://orinoco.com/payment/1234" rel="payment"/>
  <status>open</status>
</order>
```

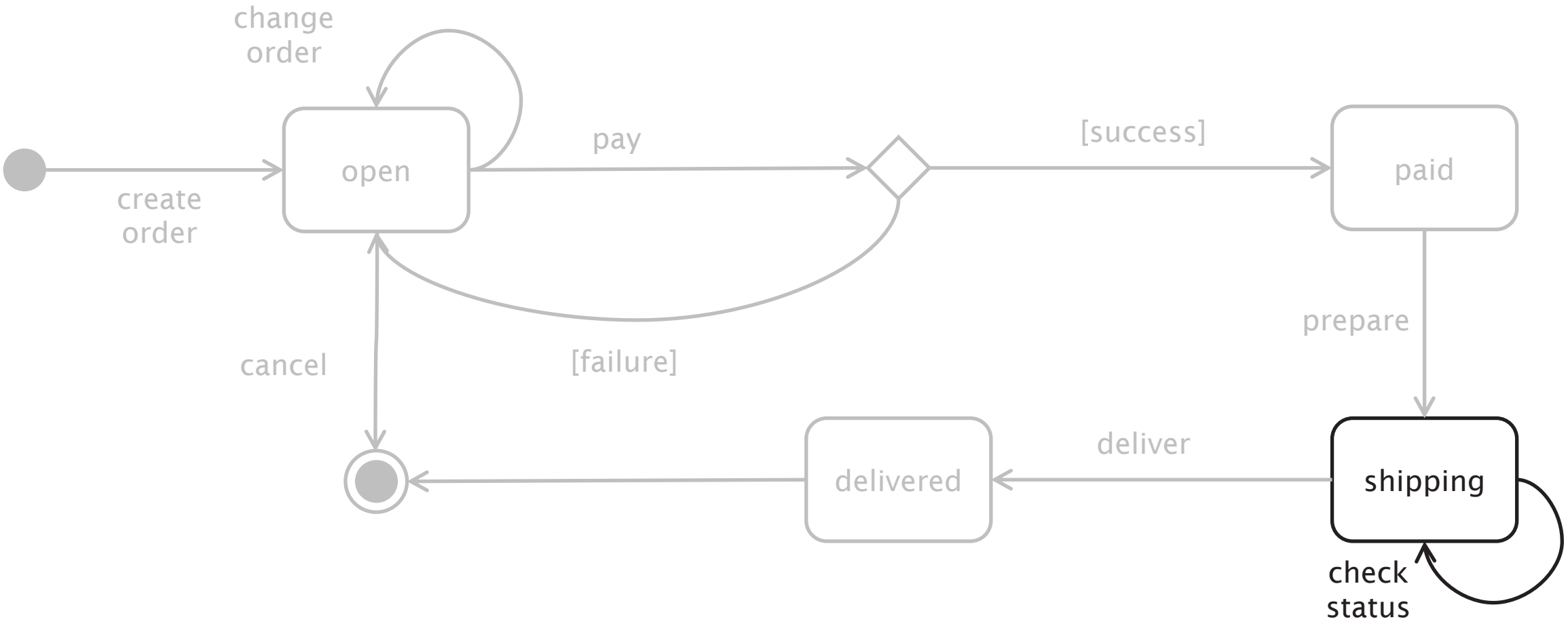
Link: header

```
GET /order/1234 HTTP/1.1
Host: orinoco.com
```

```
HTTP/1.1 200 OK
Content-Type: application/vnd.orinoco+xml
Link: <https://orinoco.com/payment/1234>; rel="payment"

<order xmlns="http://schema.orinoco.com/order">
  <items>
    <item quantity="1" isbn="1234567890"/>
  </items>
</order>
```



Check order status



Check order status

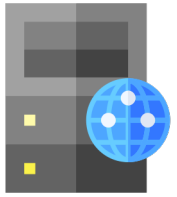
Use GET

- GET is idempotent
- GET has no side-effects!

GET

```
GET /order/1234 HTTP/1.1
Host: orinoco.com
```

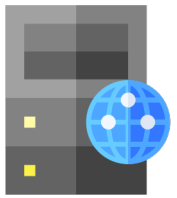
```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 107
Date: Tue, 30 Oct 2018 16:30:00 GMT

<order xmlns="http://schema.orinoco.com/order">
  <items>
  </items>
  <status>open</status>
</order>
```



GET

```
GET /order/9999 HTTP/1.1
Host: orinoco.com
```

```
HTTP/1.1 404 Not Found
Content-Length: 0
Date: Tue, 30 Oct 2018 16:30:00 GMT
```



Collections and Elements

Extra conventions for talking about collections of elements

- An order can be considered to be a collection
- An item in the order is an element of that collection

Some consensus of semantics of HTTP methods for these

In our case:

- http://orinoco.com/order/ is a collection
- http://orinoco.com/order/{order_id} is an element

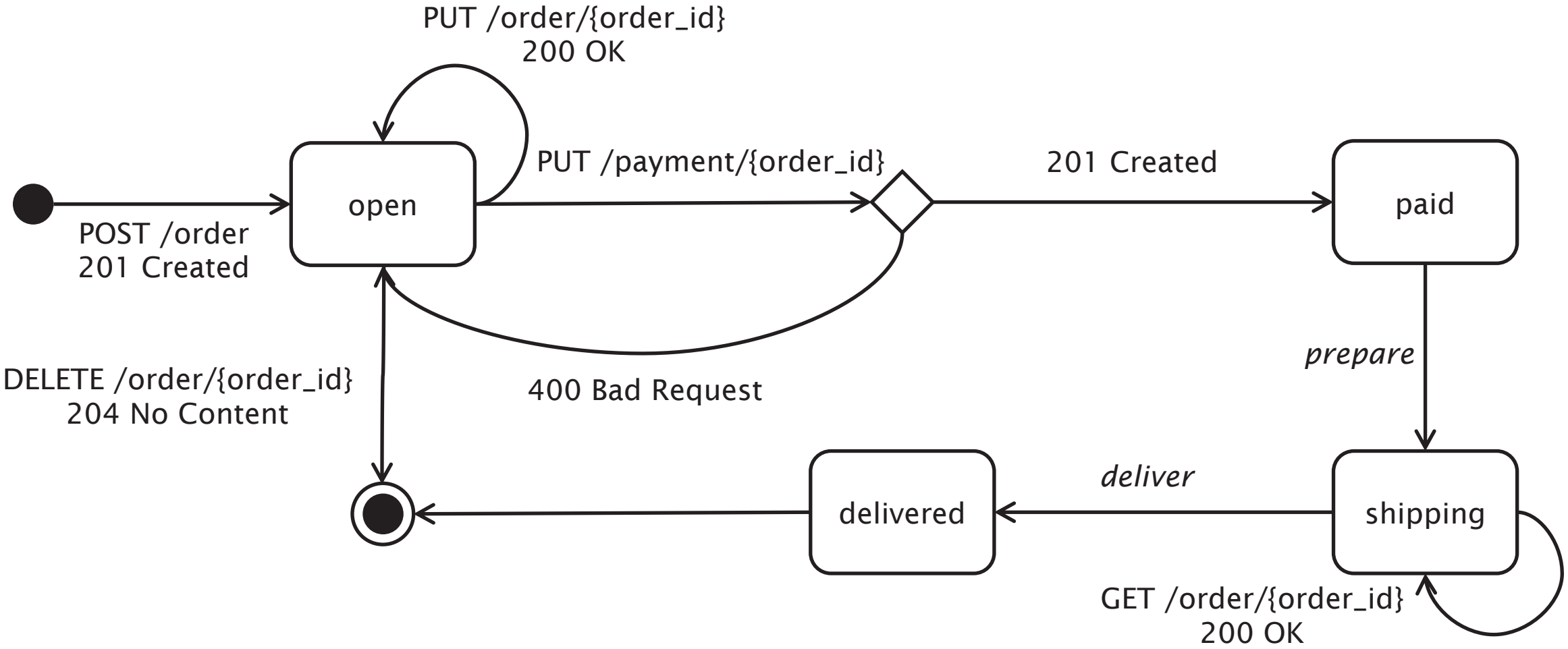
RESTful Methods for Collections

Method	Behaviour
GET	List the members of the collection (list of URIs)
PUT	Replace the entire collection with another collection
POST	Create a new member in the collection and automatically assign it a URI
DELETE	Delete the entire collection

RESTful Methods for Collection Elements

Method	Behaviour
GET	Retrieve a representation of the specified element
PUT	Replace the specified element of the collection, or if it doesn't exist create it
POST	Treat the specified member as a collection and create a new element in it
DELETE	Delete the specified member of the collection

Orinoco Workflow



Further Reading

Further Reading

REST in Practice tutorial slides

- http://www.slideshare.net/guilhermecaelum/rest-in-practice

Webber et al (2010) REST in Practice. Sebastopol, CA: O'Reilly Media

UNIVERSITY OF Southampton

Next Lecture: REST Documentation