Empirical Modelling and the challenge of enterprise architecture

Charlie Care

BT

THE UNIVERSITY OF WARWICK

About me

- BSc Computer Science (Warwick, 2004)
- Ph.D. Computer Science (Warwick, 2008)
- Software Engineer at BT since 2007
  - Graduate Software Engineer/Analyst, 2007-2009
  - Senior Software Engineer. 2009-Present

- Things that interest me
  - Integration patterns, REST web services, simple integrations
  - Java, Python, Ruby, Service/Client side JavaScript
  - Convention over Configuration (frameworks and approaches)

This Lecture – outline

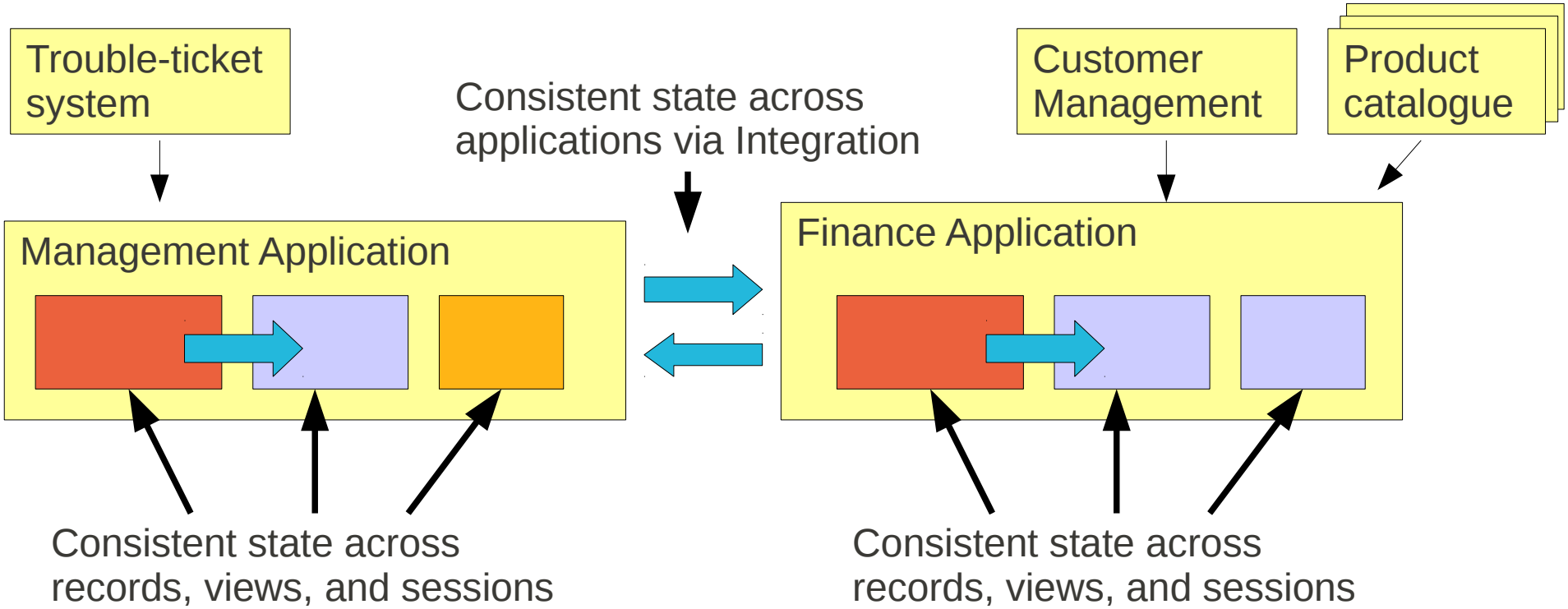
- **Introduction**
- Section 1: Scene setting
- Section 2: Maintaining state within the application
- Section 3: Maintaining state between applications
- Section 4: Blue sky
- Conclusions

This Lecture – outline

- Introduction
- **Section 1: Scene setting**
- Section 2: Maintaining state within the application
- Section 3: Maintaining state between applications
- Section 4: Blue sky
- Conclusions

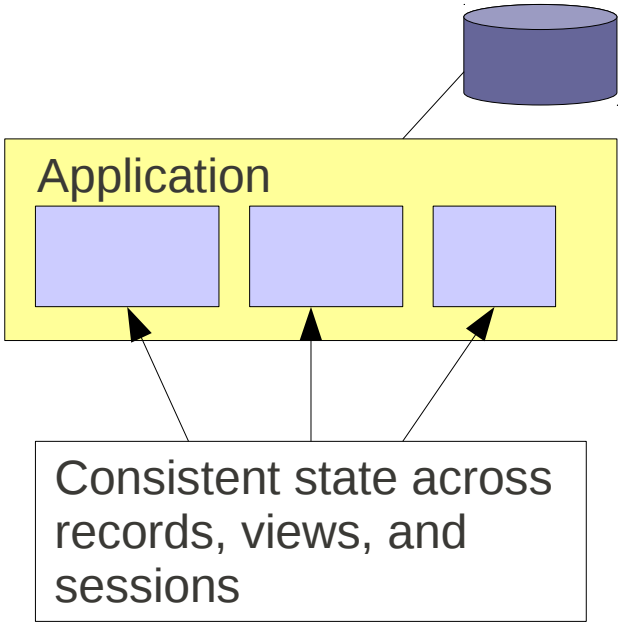
State management in Enterprise Systems

- Management of state within applications
- Management of state between applications



Intra-application state management

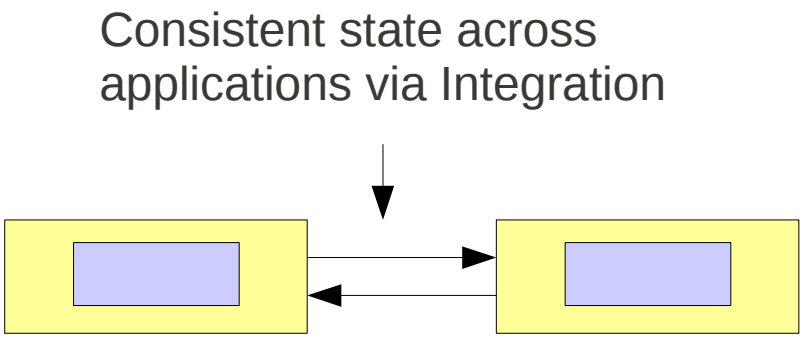
- Traditionally the role of the database
- Views offer dependency
- Triggers are an agency mechanism
- That's fine when applications pull data from db...



- What about clustering and caching outside the db?
- What about no-sql solutions
- What about distributed apps?

Inter-application state management

- Variety of ways of integrating
  - File transfer
  - Shared Database
  - RPC
  - Messaging
- Not as much EM thinking here...



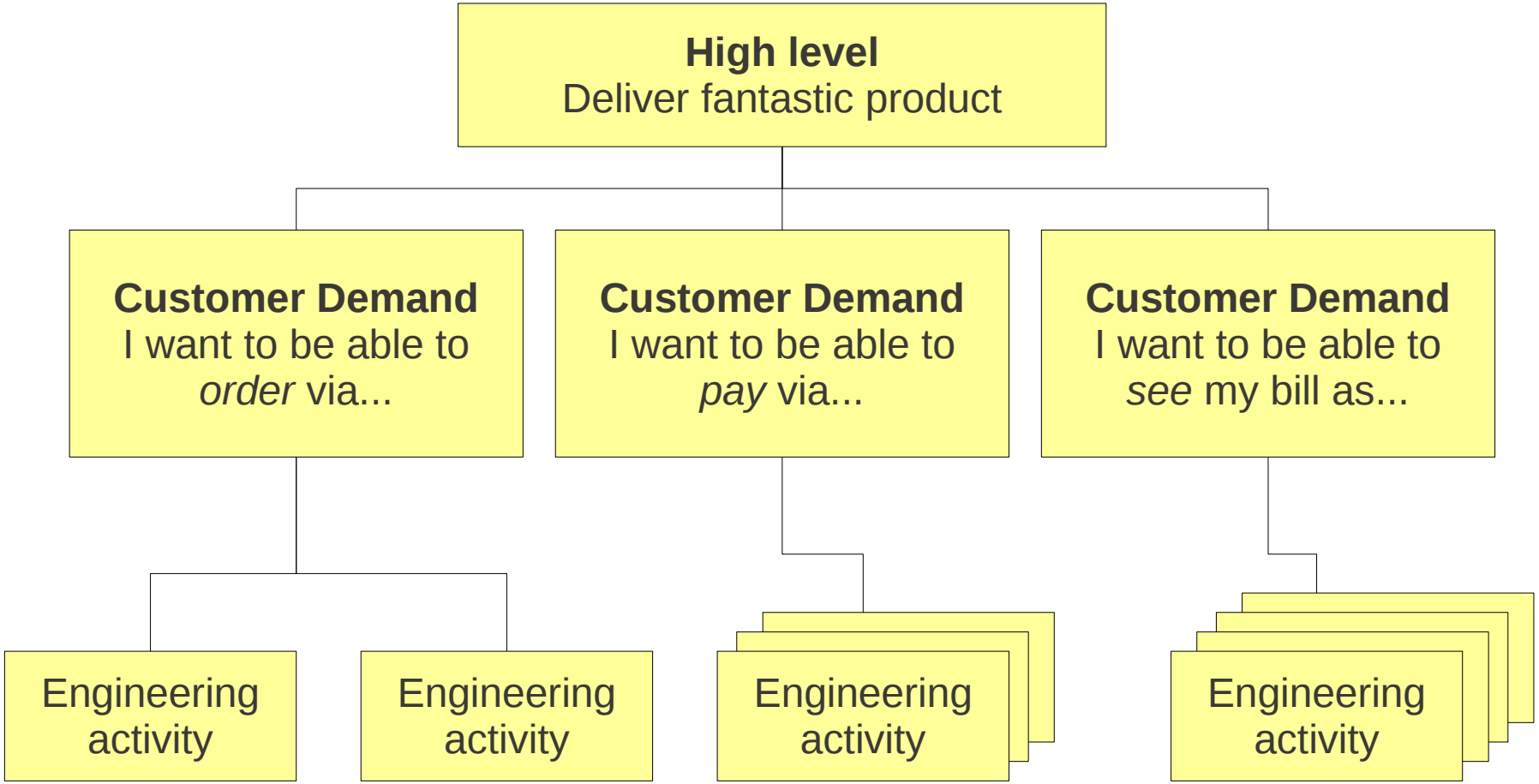
What does EM have to say about integration...?

- LSD Notation
- *Oracles*, *Handles*
- *Derivates* tell you about master data etc.

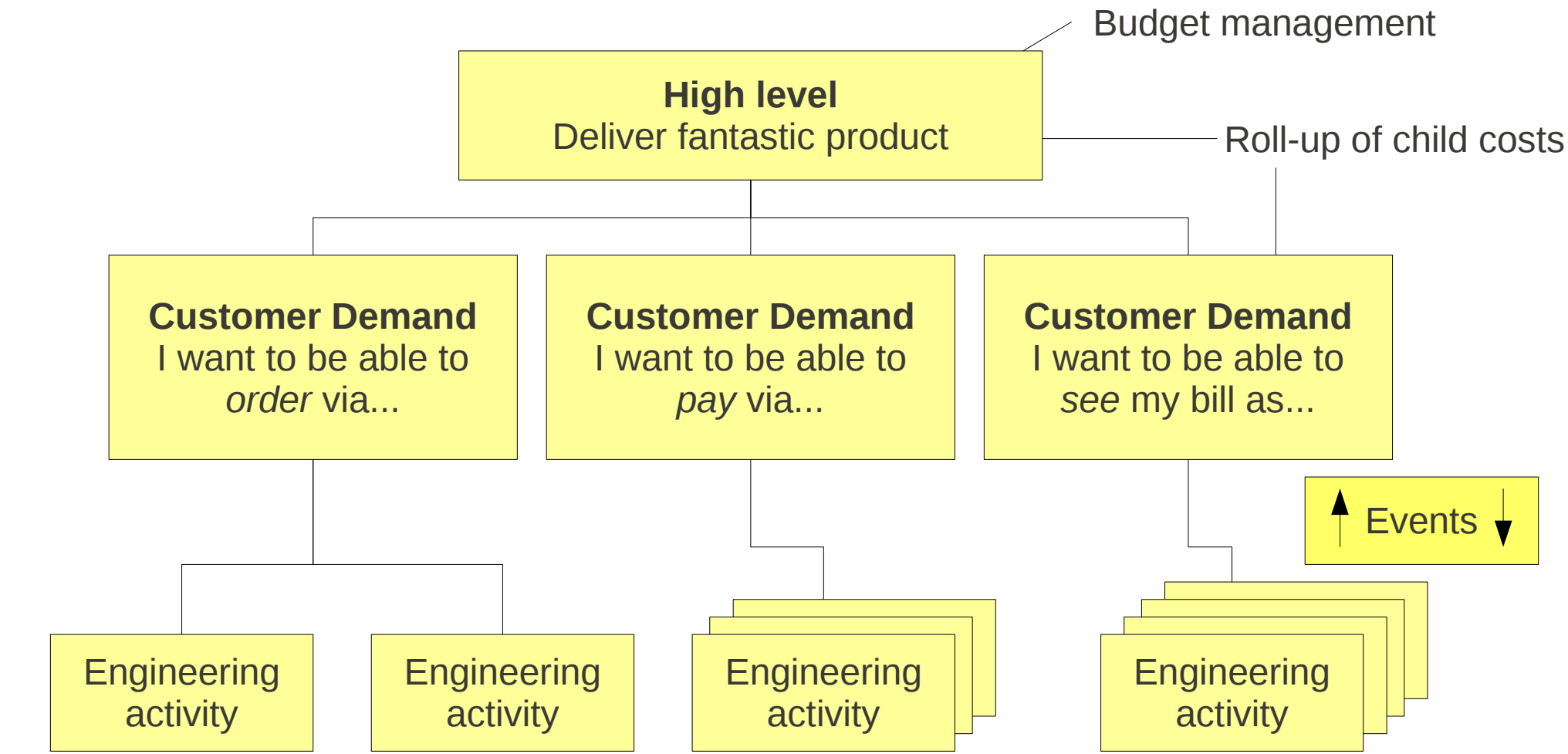
Real-world example

- Agile system for managing demand
- Supports decomposition of 'user stories'
- Reporting provided in a separate data warehouse

Example: Hierarchical State



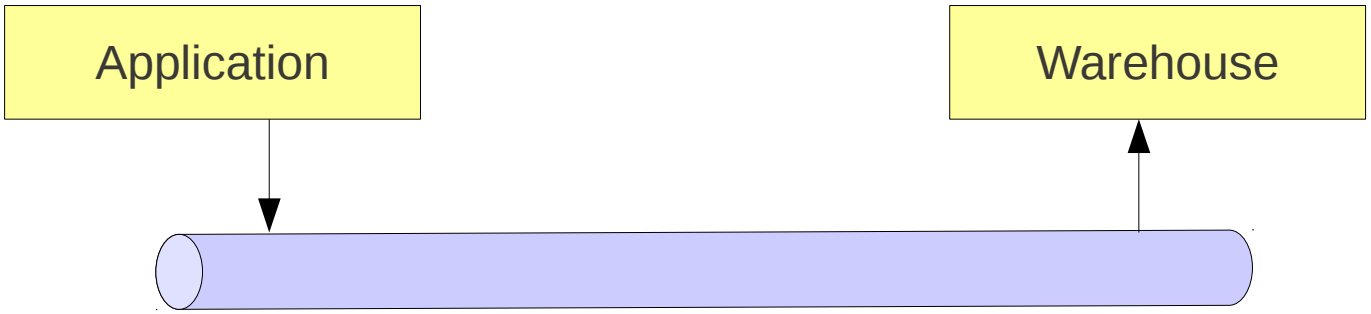
Inside the application – example hierarchy



Propagate information up and down tree on editing of a record

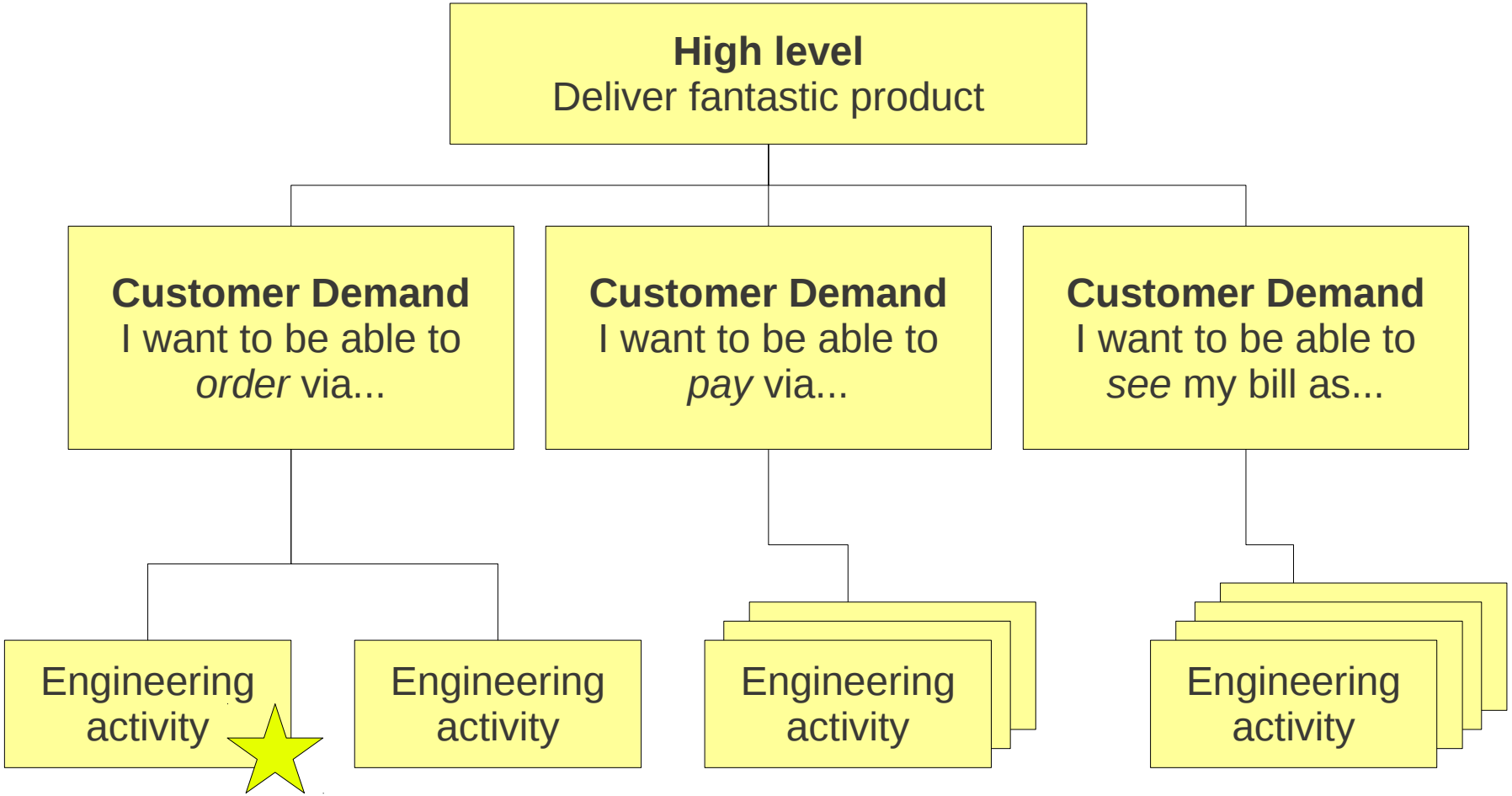
Inter-Application state management

- Application doesn't exist on it's own
  - e.g. ship messages to data warehouse

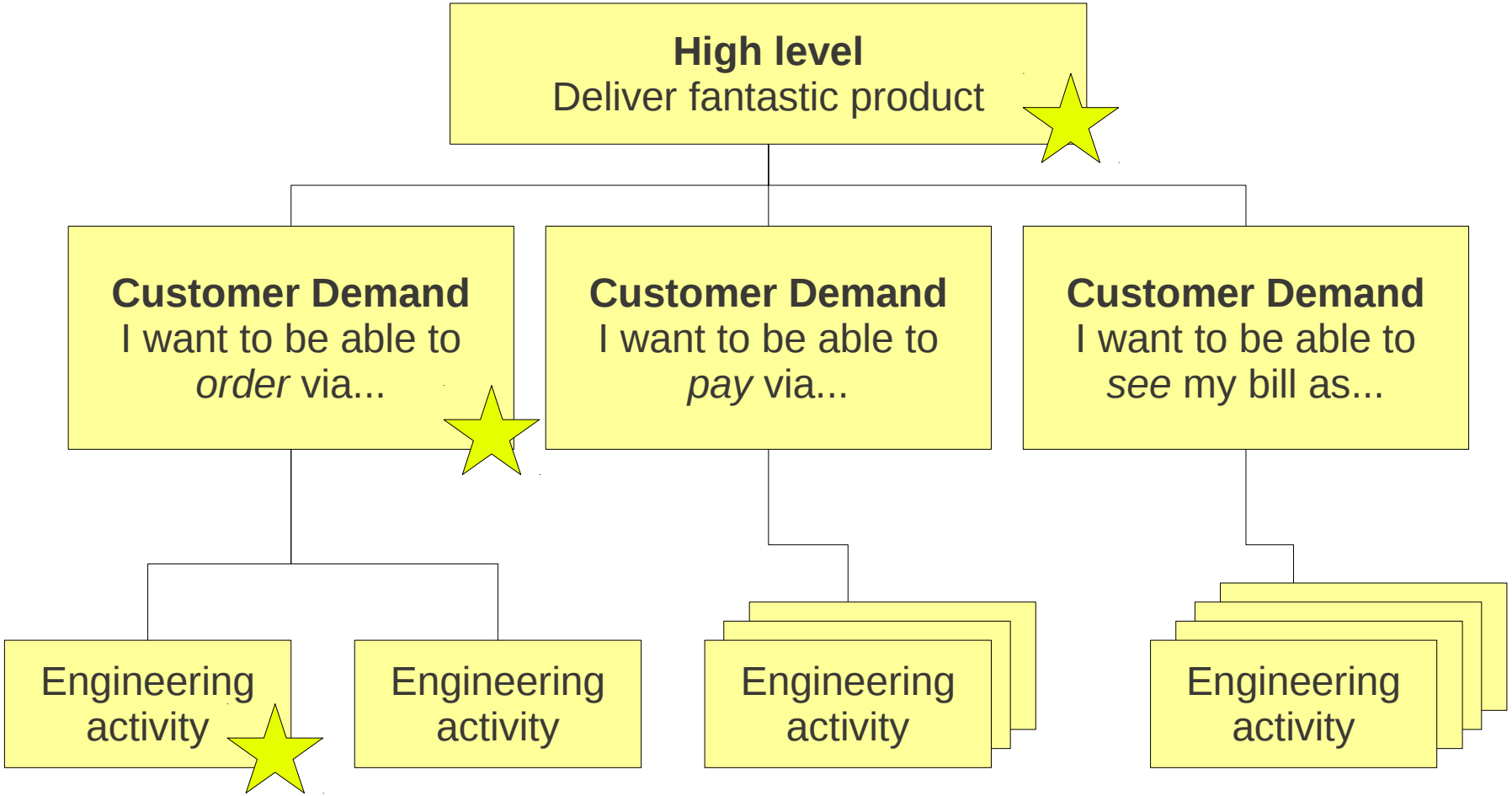


- What do we send?

Example: Hierarchical State

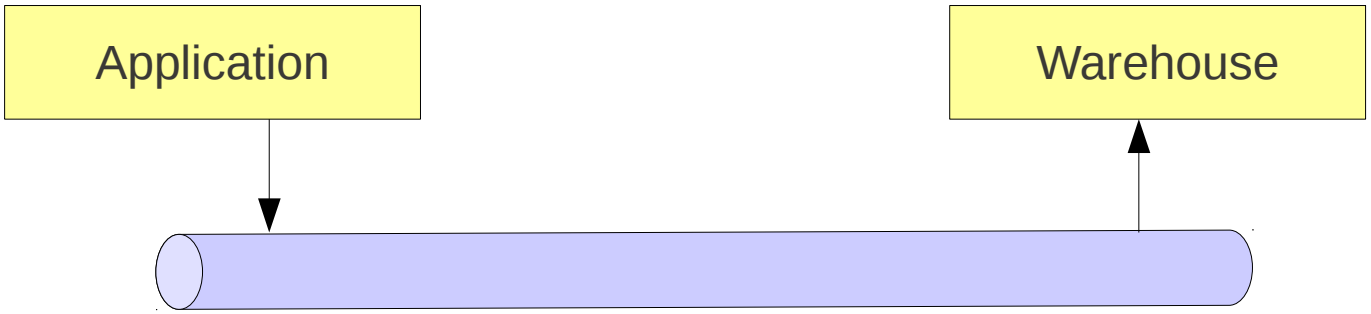


Example: Hierarchical State



Inter-Application state management

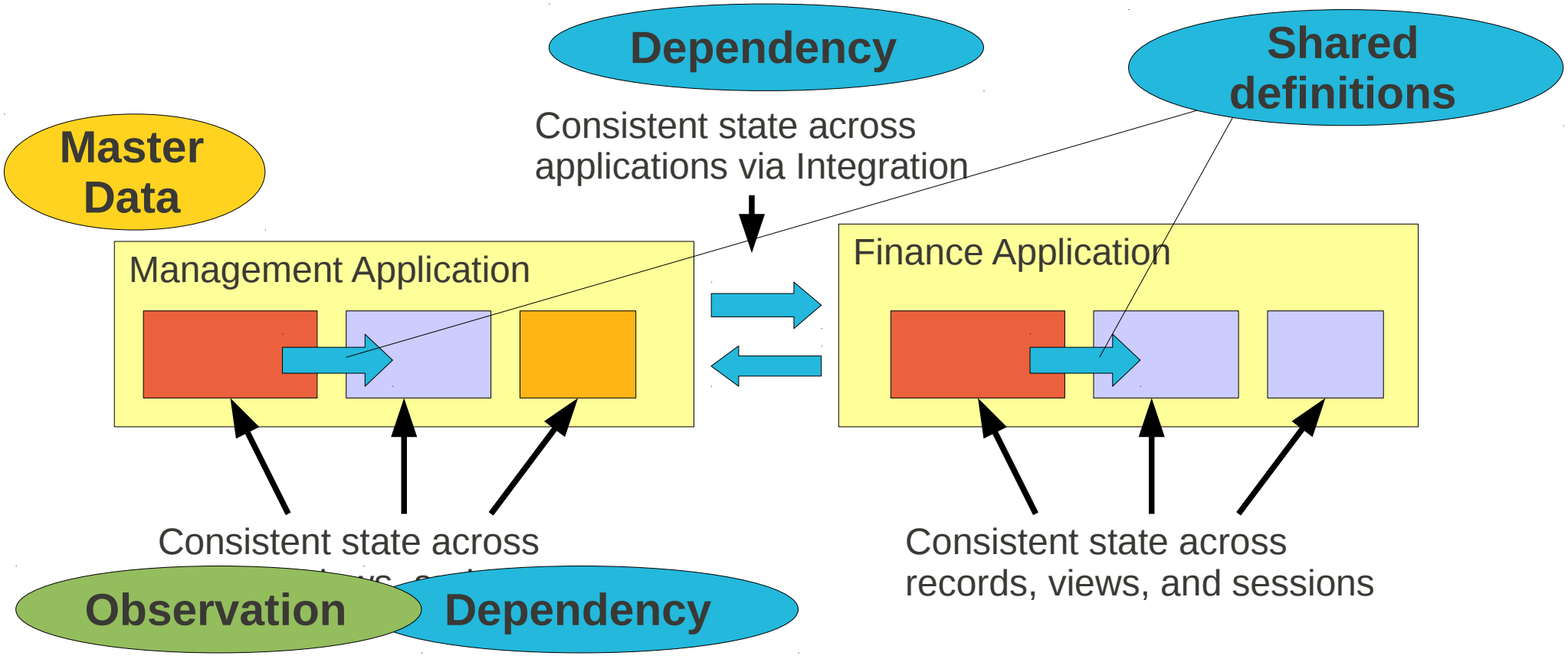
- Application doesn't exist on it's own
  - e.g. ship messages to data warehouse



- What do we send?
  - One message for each change?
  - What about dependent changes?
  - Do we replicate the dependency and derivation logic on the far end?
    - Who masters this logic? What about upgrades?

State management in Enterprise Systems

- Management of state between applications
- Management of state within applications



This Lecture – outline

- Introduction
- Section 1: Scene setting
- **Section 2: Maintaining state within the application**
- Section 3: Maintaining state between applications
- Section 4: Blue sky
- Conclusions

The need for state maintainers

- Dependency (as you've seen it in EDEN) is a really useful tool
  - Simplicity of state relationships...
  - ...in a declarative way
  - Avoids the classic updateState() function !
  - We're not the only ones to realise the benefits of declarative state!

- What's the general pattern here?

The Observer Pattern

- One of the classic patterns in the GOF book
  - *Design Patterns: Elements of Reusable Object-Oriented Software*.

    Gamma, Helm, Johnson, Vlissides (1994)

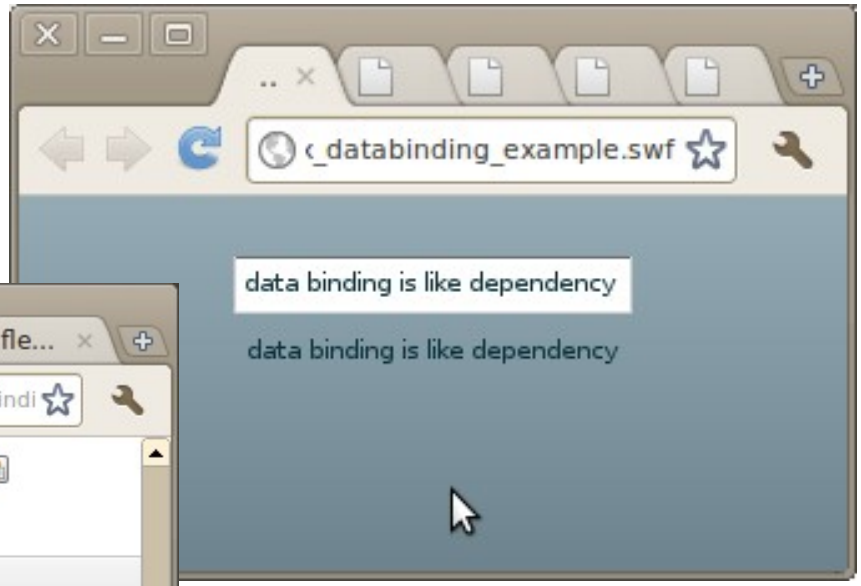
  - “defines a one-to-many dependency between objects so that when one object changes state, all it's dependents are notified and updated automatically” (p. 293)
  - Sounds good... but is this *dependency*?

Observer (cont...)

- Adobe Flex, Microsoft Xaml
  - data-binding expressions are implementations of **Observer**.
  - General support via events dispatch mechanism
- C#.net
  - Support for observer via event pub/sub
  - Provides **IObservable** interface
- Java
  - Even Java has had **Observer** and **Observable** interfaces support since JDK 1.0

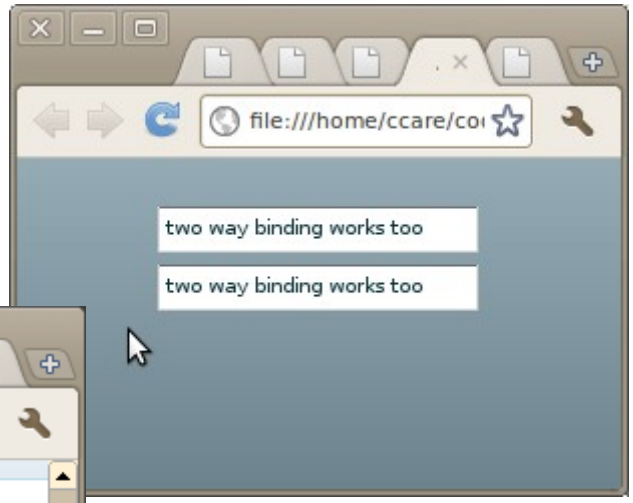
Demo 1 – Flex data-binding

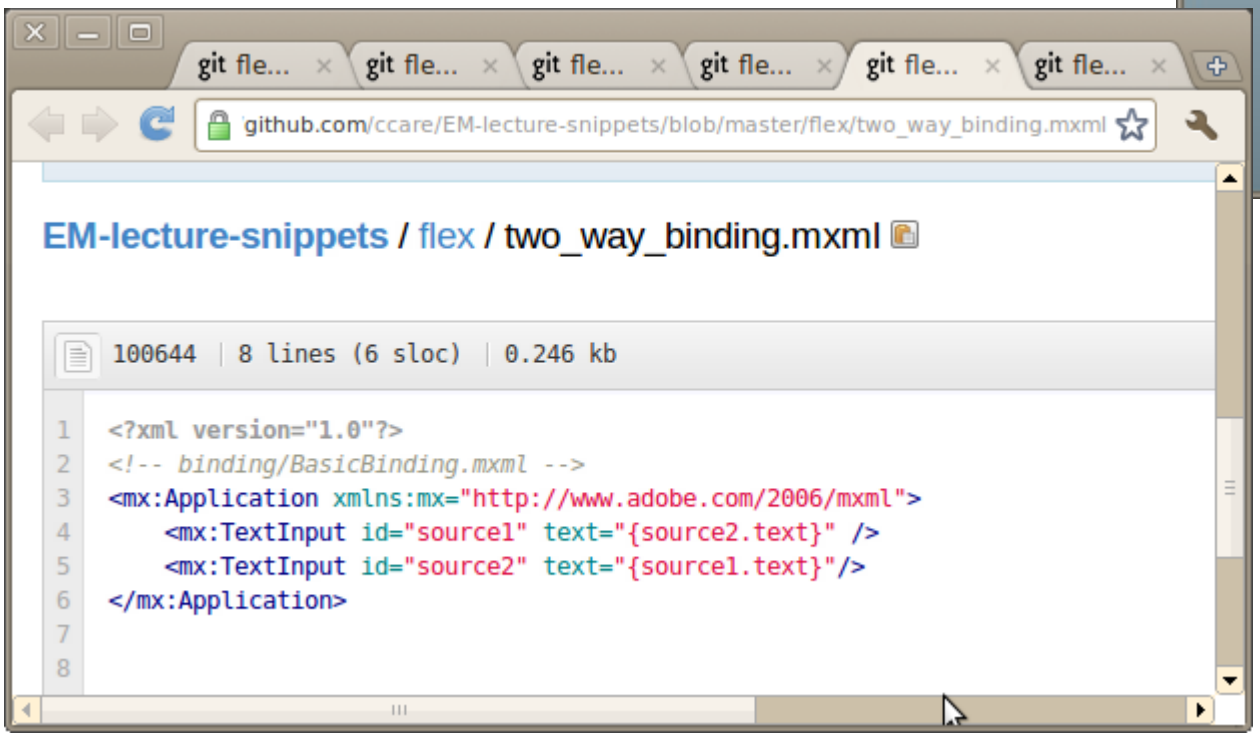
- Data-binding is like dependency
- Declarative



Demo 2 – two-way binding

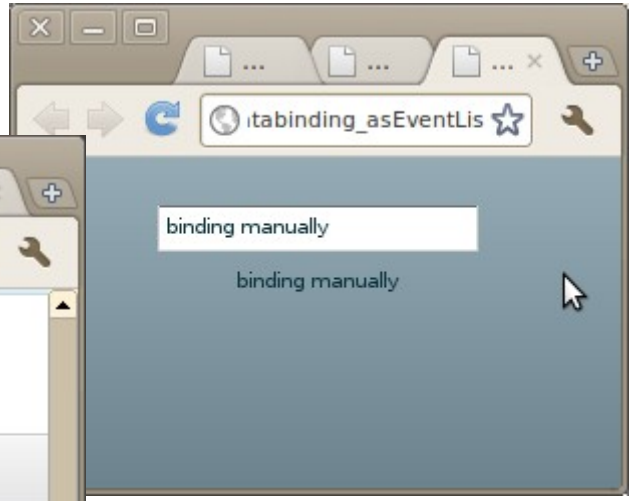
- Declarative two way binding is possible too
- Need to use agency for this in EDEN

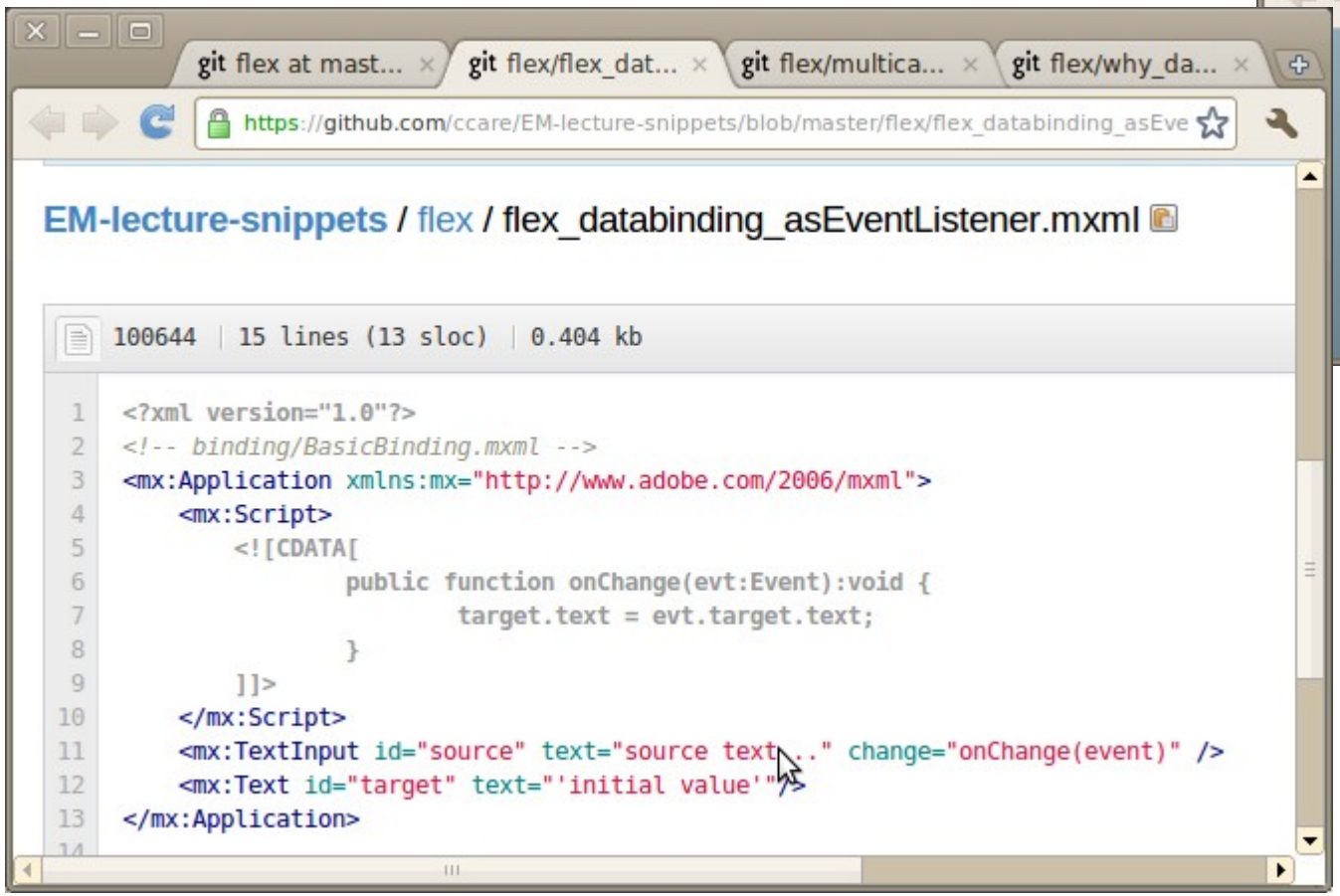




Demo 3 – binding by event listeners

- Of course, you can implement this yourself with event listeners





Demo 4 – Multi-target agency

- Custom event listeners will allow you to break dependency semantics

git flex at master fro... | git flex/multicast_eve... | git flex/why_databind...

https://github.com/ccare/EM-lecture-snippets/blob/master/flex/multicast_event_bin

EM-lecture-snippets / flex / multicast_event_binding.mxml

100644 | 19 lines (17 sloc) | 0.699 kb

```
<?xml version="1.0"?>
<!-- binding/BasicBinding.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
                public function onChange(evt:Event):void {
                        target.text = evt.target.text;
                }
        ]]>
    </mx:Script>
    <mx:TextInput id="source1" text="source 1" change="onChange(event)" />
    <mx:TextInput id="source2" text="source 2" change="onChange(event)" />
    <mx:TextInput id="source3" text="source 3" change="onChange(event)" />
    <mx:TextInput id="source4" text="source 4" change="onChange(event)" />
    <mx:TextInput id="source5" text="source 5" change="onChange(event)" />
    <mx:Text id="target" text="'initial value'"/>
</mx:Application>
```

why... | mult...

ıst_event_binding.swf

val1

val2

source 3

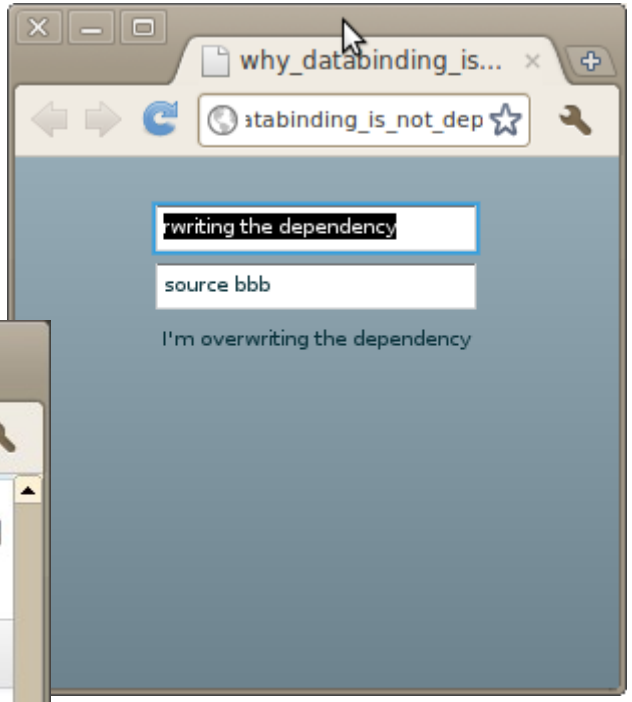
source 4

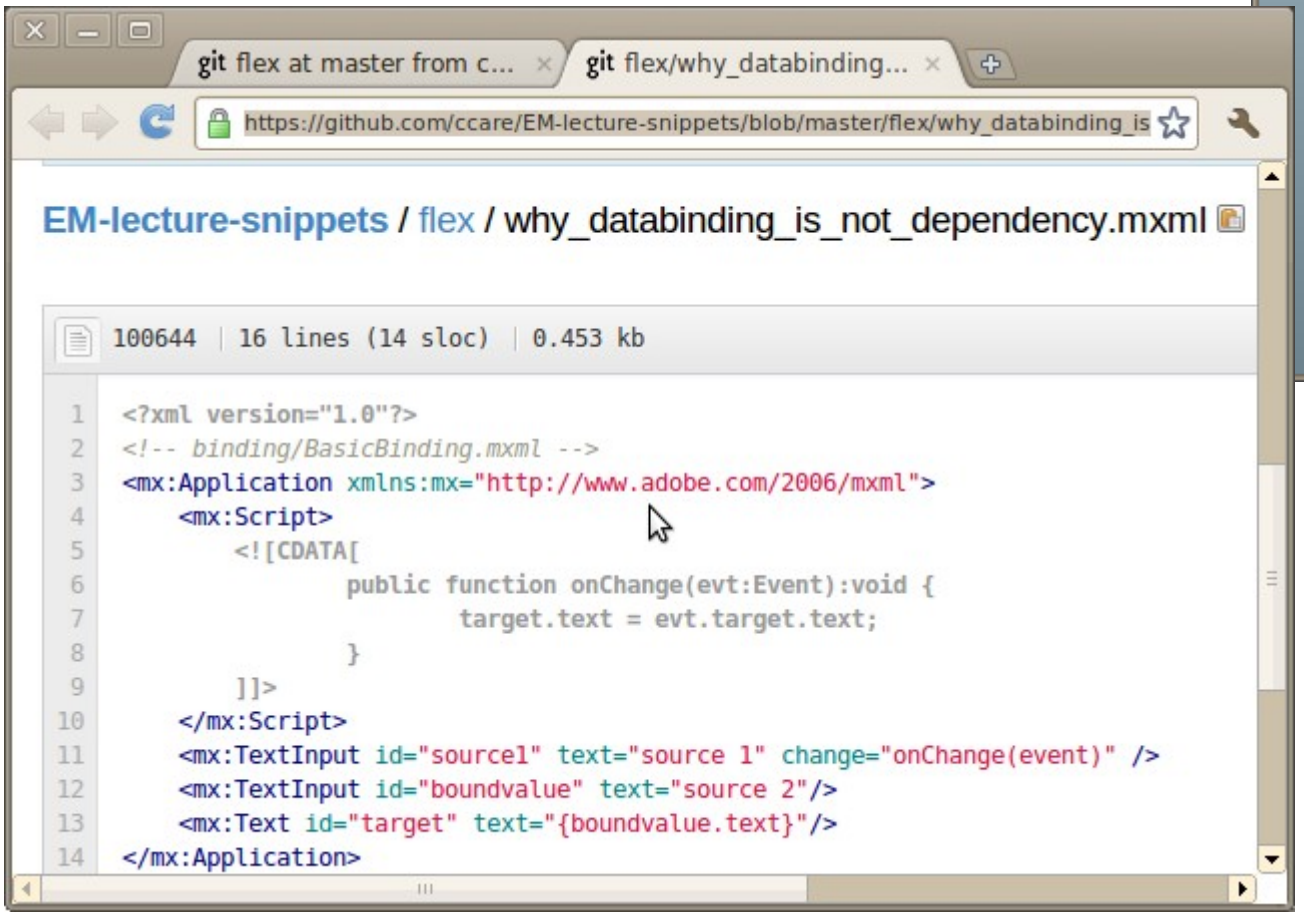
source 5

val2

Demo 5 – Data-binding is NOT definitive

- And a simple investigation shows that data-binding expression do not give you definitive behaviour either.





Data-binding reviewed

- Data-binding in Flex is really handy
- You get
  - The convenience of dependency
  - Declarative expression of state update
  - The benefits of being able to do more than **EDEN** (two way, multicast etc.)
- However, you don't get
  - atomic state change
  - introspection of definition
  - Clear dependency graph
  - Protection from redefinition...

EDEN: Two types of Observer

- In EDEN, Both dependencies and triggered procs are types observers.

```
a is b + c;
```

```
proc update_A : b, c {
      a = b + c;
 })
```

- Can implement data-binding as agency, but without the semantic guarantees of dependency.
- Modern languages are happy to give you the sugared syntax without the guarantees...
- What behaviour makes sense?
- Might be the compromise in a traditional language

Some types of **observer**

- Dependency – guaranteed not to observe inconsistency
  - a **is** b + c
- Triggered updates (agency), trigger on change, no guarantees when executed
  - **proc** f : a {}
- Some others we might consider
  - Definitive state – like dependency – no immediate update
    - a **is calculated by** b + c (dtkeden kind of gives you this)
  - Triggered updates that are guaranteed to run before observation – e.g. like database triggers
    - **eager_proc** trigger : a {}

This Lecture – outline

- Introduction
- Section 1: Scene setting
- Section 2: Maintaining state within the application
- **Section 3: Maintaining state between applications**
- Section 4: Blue sky
- Conclusions

Observation and enterprise integration

- We want applications to observe state in other applications

- If we're going to expose a tree of observables...
  - …why not do it over http?
  - Can a simple RESTful exposure can provide the observable semantics we want?

REST Example: Publication service

- Representational State Transfer (REST)
- Every resource is a url
- Use standard http verbs to manipulate resources
- Access representation of resource **5**
  - HTTP GET /pubdata/dcs/papers/5
- Create or Update resource **5** (submit message body)
  - HTTP PUT /pubdata/dcs/papers/5
- Create a new resource (submit message body)
  - HTTP POST /pubdata/dcs/papers/
- Delete an entry
  - HTTP DELETE /pubdata/dcs/papers/5

REST publications service

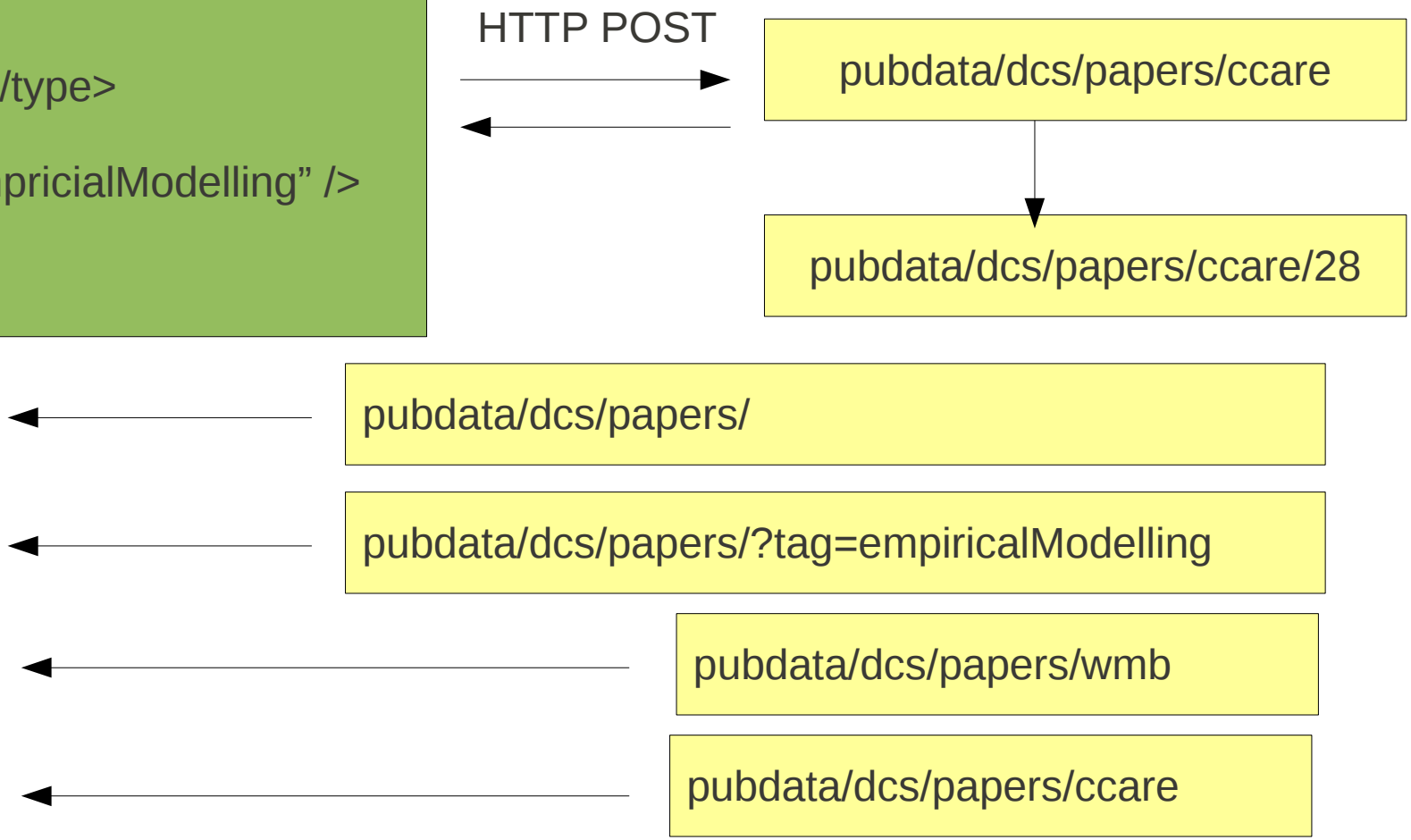
- Use case: Access publications within dcs
- Access via http GET

All dcs publications	pubdata/dcs/papers/
All EM publications	pubdata/dcs/papers/?tag=empiricalModelling
Meurig's publications	pubdata/dcs/papers/wmb
Charlie's publications	pubdata/dcs/papers/ccare

Example: publications repository

- Use case: I add this lecture... what's changed?

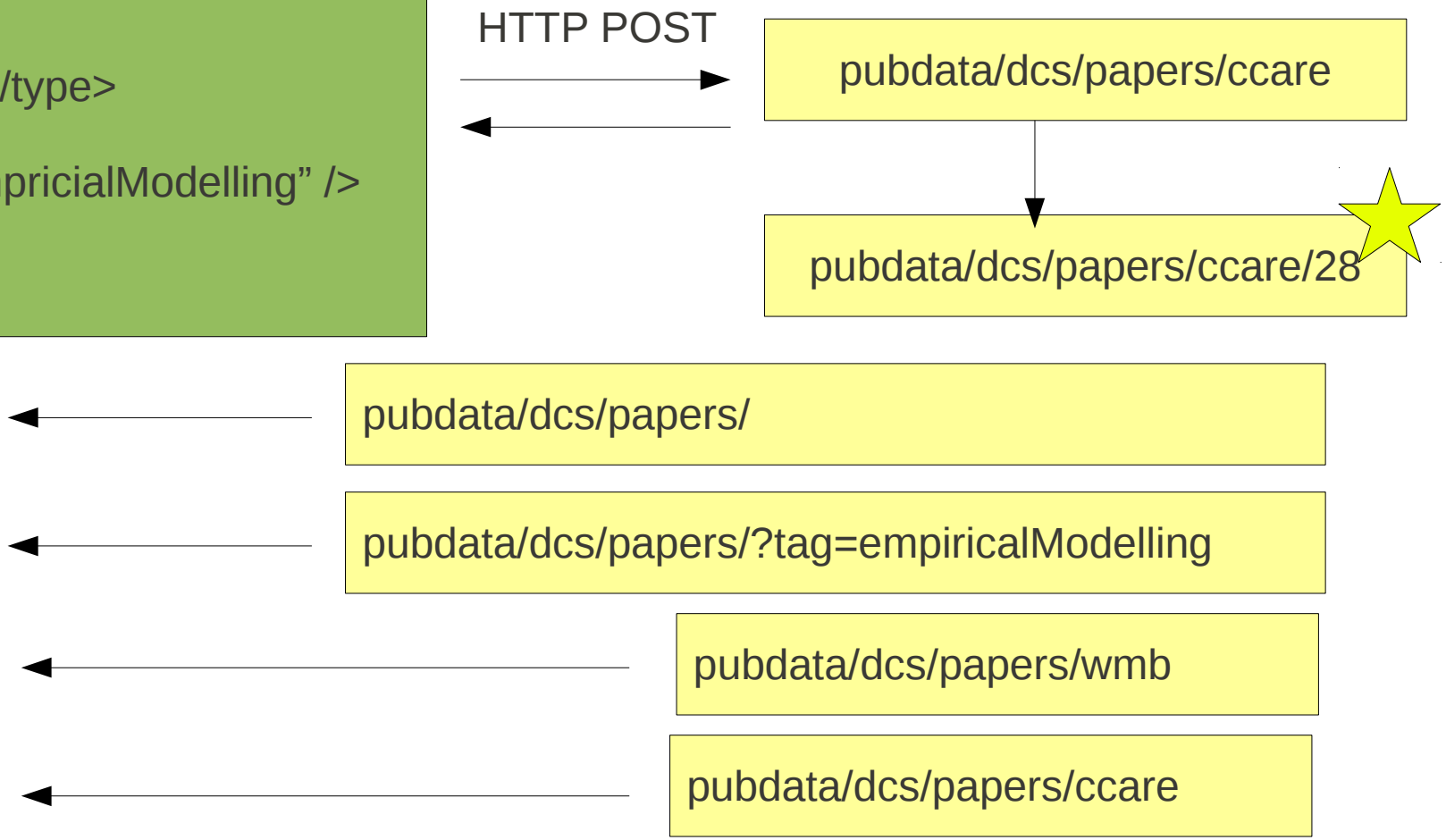
```
<publication>
   <title>...</title>
   <type>lecture</type>
   <tags>
      <tag id="empricialModelling" />
   </tags>
</publication>
```



Example: publications repository

- Use case: I add this lecture... what's changed?

```
<publication>
    <title>...</title>
    <type>lecture</type>
    <tags>
        <tag id="empricialModelling" />
    </tags>
</publication>
```



Example: publications repository

- Use case: I add this lecture... what's changed?

```
<publication>
    <title>...</title>
    <type>lecture</type>
    <tags>
        <tag id="empricialModelling" />
    </tags>
</publication>
```

HTTP POST

pubdata/dcs/papers/ccare

pubdata/dcs/papers/ccare/28

pubdata/dcs/papers/

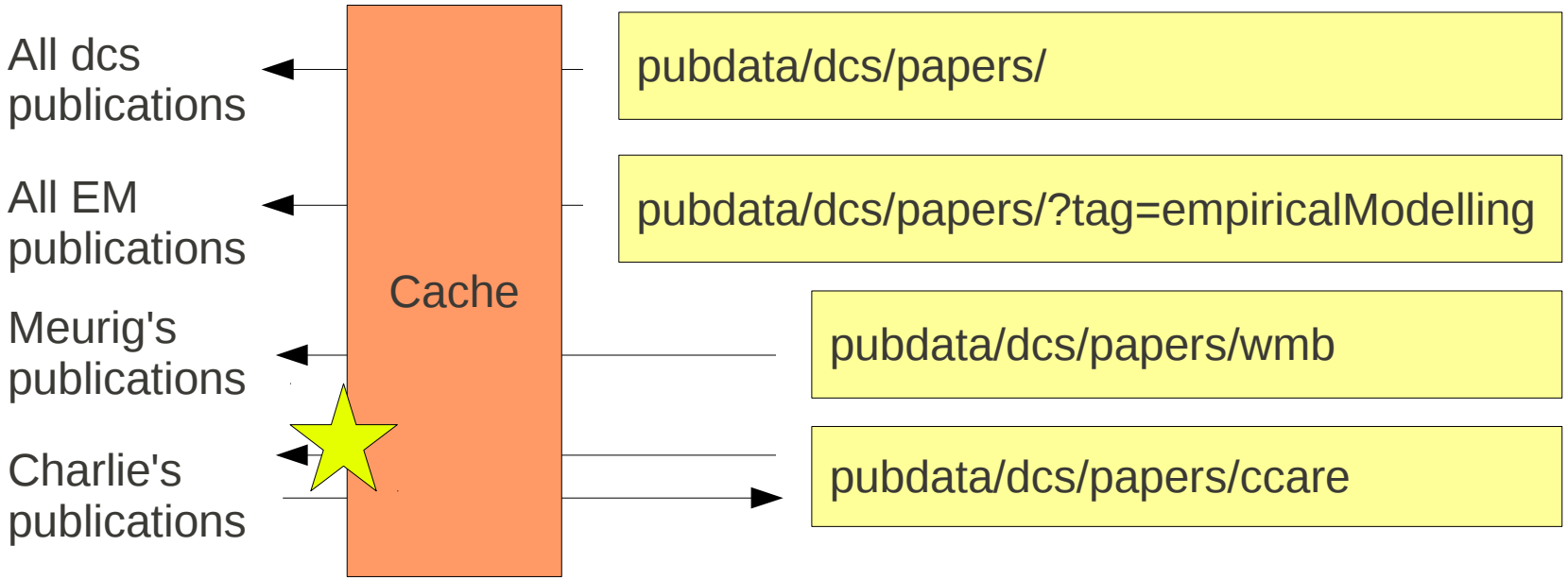
pubdata/dcs/papers/?tag=empiricalModelling

pubdata/dcs/papers/wmb

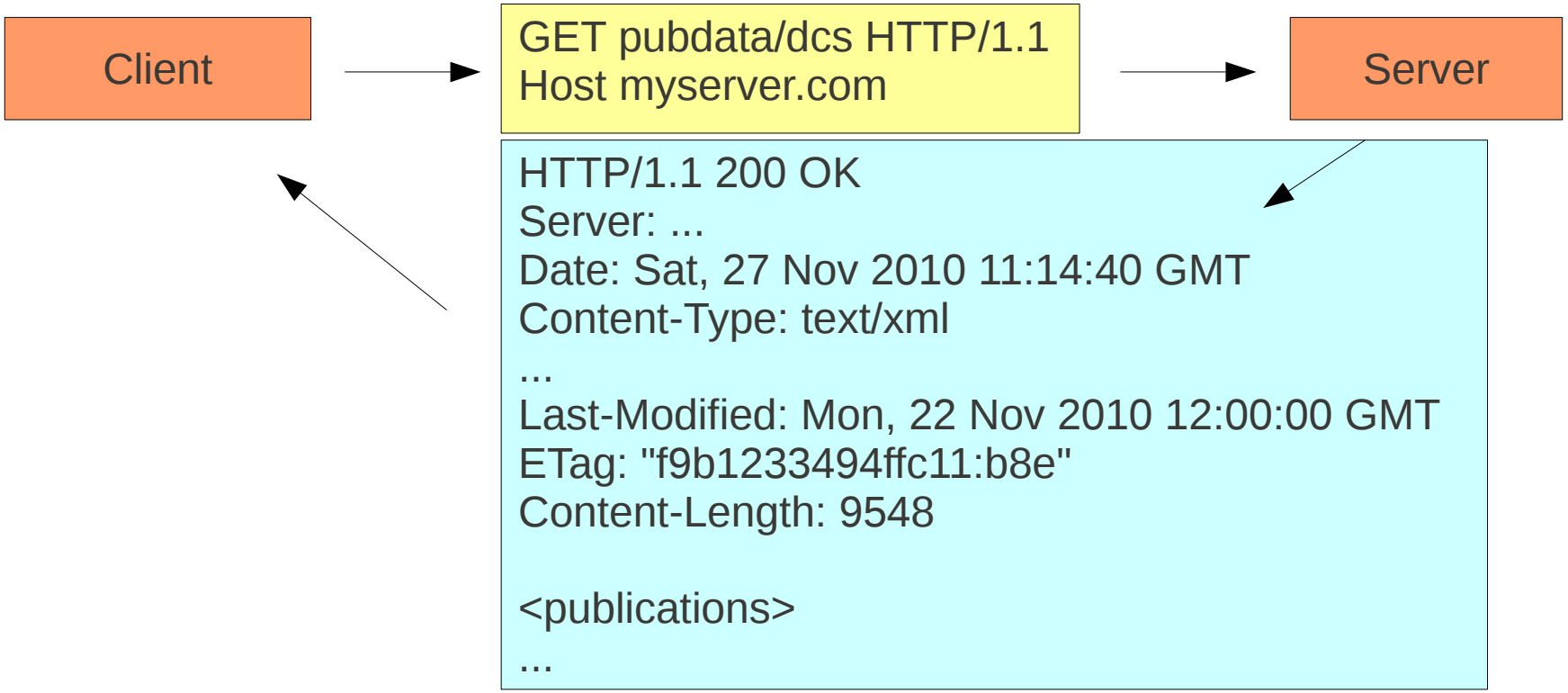
pubdata/dcs/papers/ccare

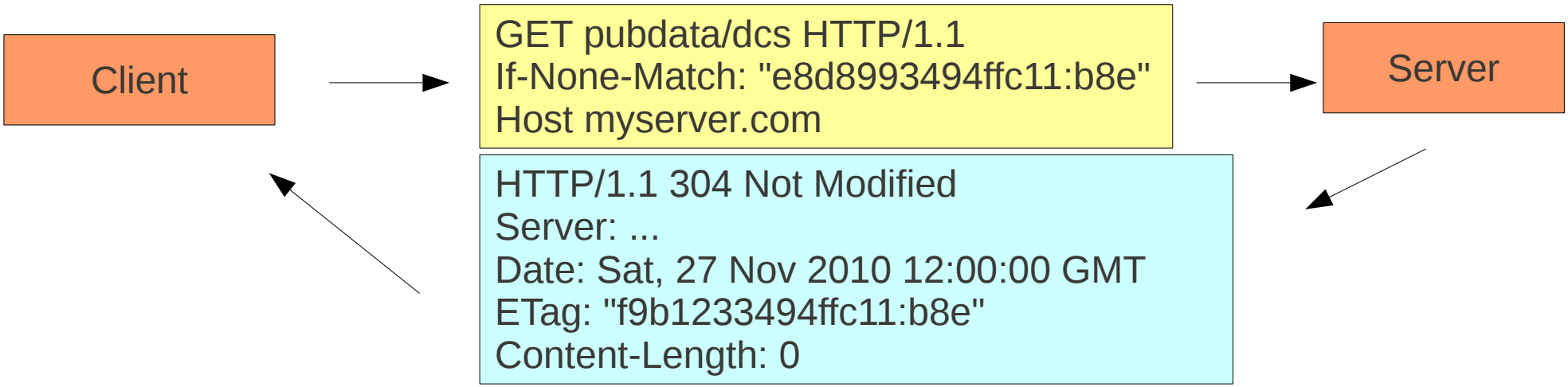
Example: publications repository

- The nice thing about REST is that I can use standard cache technology
- But what about cache expiry?



HTTP Etags – Entity tags





HTTP ETags

- ETags provide a nice way of interrogating the application for value expiry
- So now we're back to an intra-app problem
- Which we can solve with
  - Events
  - Observers
  - Custom controllers
  - Or possibly dependency
  - Or one of our other observer types...

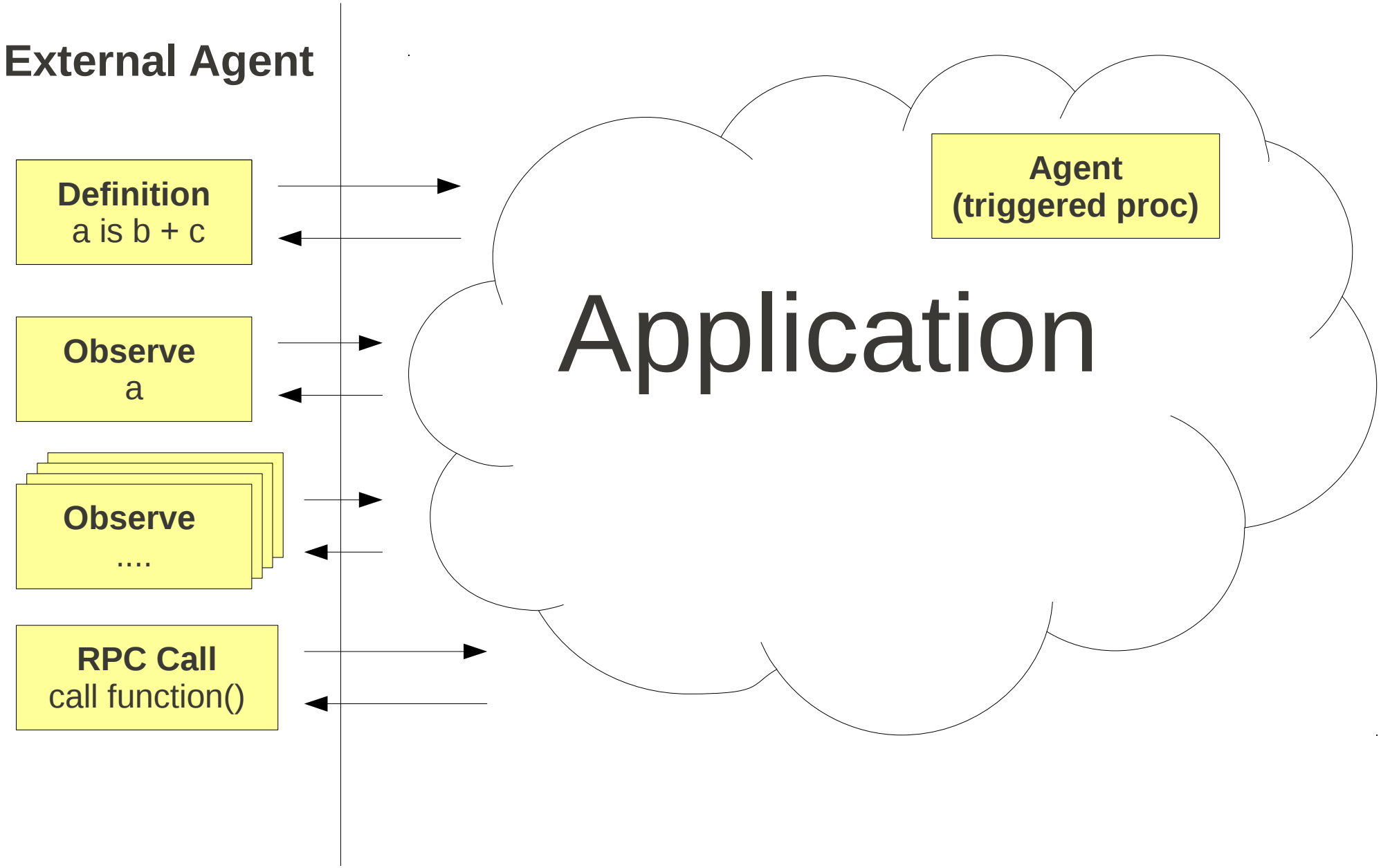
This Lecture – outline

- Introduction
- Section 1: Scene setting
- Section 2: Maintaining state within the application
- Section 3: Maintaining state between applications
- **Section 4: Blue sky**
- Conclusions

EM in the cloud

- We have webeden, but this is really a web-enabled exposure of tkeden.
- Scalable? Depends what you mean.
- What about a web server with built in dependency?
- Web Service with Multiple dependency 'worlds' or instances.
- Modellers/Programmers submit definitions to a 'world' to change state
- Http subscribers can observe the state

EM over HTTP


External Agent
Definition
a is b + c
Observe
a
Observe
....
RPC Call
call function()
Agent
(triggered proc)
Application

What's out there already?

- What's the relationship with other technologies out there?
- Hadoop – Distributed document DB with map reduce.
- CouchDB – provides JavaScript views
  - Incremental map-reduce is a scalable way of maintaining dependencies
  - although no lock down of state (which, for CouchDB, is a good thing)
- MongolDB – more like traditional DB
  - Also uses JavaScript in map-reduce
- Persevere-framework
  - Server side JavaScript store
  - Can call functions via RPC

High level Implementation

- Are we really talking about dependency enabled caches?
  - V. Fast (and transactional) marking of out of date state
  - Parallel recalculation of state using thread pool
  - No self management of expiry
  - Re-use etags mechanism...
- Definitions should be defined using common language... e.g. EDEN or JavaScript with extensions
- Don't forget triggered procs – **agency**

eden-ws

- Not sure about name :-)
- Beginnings of a reference implementation
- Restful web service fronting a definitive machine
- Implemented in Java
- Definitive scripts based on JavaScript (Rhino)
- Work in progress
  - Not completed decided about integrations
  - Not completed decided about client side stuff

cURL – a very quick primer

- Standard command line utility to automate web requests
- Easily interact with http endpoint

Http GET

```
curl http://..../resource
```

Http PUT

```
curl -X PUT http://..../resource
```

Http PUT or POST with data

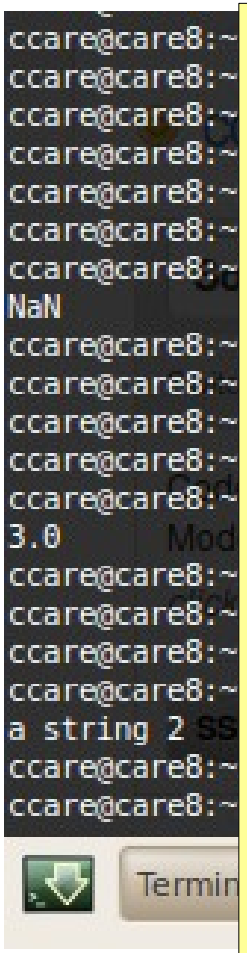
```
curl -X PUT http://..../resource -d 'request body'
curl -X POST http://..../resource -d 'request body'
```

eden-ws – beginnings of a reference implementation

```
ccare@care8:~$
ccare@care8:~$ # create a new dependency world
ccare@care8:~$ curl -X PUT http://localhost:8080/services/spaces/myspace -d ""
ccare@care8:~$ # Define: a is b + c
ccare@care8:~$ curl -X PUT http://localhost:8080/services/spaces/myspace/a -d "#b + #c"
ccare@care8:~$ # Read observable a
ccare@care8:~$ curl http://localhost:8080/services/spaces/myspace/a -w "\n"
NaN
ccare@care8:~$ # define b and c
ccare@care8:~$ curl -X PUT http://localhost:8080/services/spaces/myspace/b -d "1"
ccare@care8:~$ curl -X PUT http://localhost:8080/services/spaces/myspace/c -d "2"
ccare@care8:~$ # Read observable a
ccare@care8:~$ curl http://localhost:8080/services/spaces/myspace/a -w "\n"
3.0
ccare@care8:~$ # update b
ccare@care8:~$ curl -X PUT http://localhost:8080/services/spaces/myspace/b -d "'a string '"
ccare@care8:~$ # Read observable a
ccare@care8:~$ curl http://localhost:8080/services/spaces/myspace/a -w "\n"
a string 2
ccare@care8:~$ # terminate my app
ccare@care8:~$ curl -X DELETE http://localhost:8080/services/spaces/myspace -d ""
```

Terminal 0 Terminal 1

eden-ws – beginnings of a reference implementation




```
# create a new dependency world
curl -X PUT http://localhost:8080/services/spaces/myspace -d ""
# Define: a is b + c
curl -X PUT http://localhost:8080/services/spaces/myspace/a -d "#b + #c"
# Read observable a
curl http://localhost:8080/services/spaces/myspace/a -w "\n"
> NaN
# define b and c    ( b = 1, c = 2)
curl -X PUT http://localhost:8080/services/spaces/myspace/b -d "1"
curl -X PUT http://localhost:8080/services/spaces/myspace/c -d "2"
# Read observable a
curl http://localhost:8080/services/spaces/myspace/a -w "\n"
> 3
# update b ( b = ' a string' )
curl -X PUT http://localhost:8080/services/spaces/myspace/b -d "'a string '"
# Read observable a
curl http://localhost:8080/services/spaces/myspace/a -w "\n"
> a string 2
```

Instance name

Definition

Observable name

What about function invocation?

- PUT to define a function

```
• curl -X PUT http.../services/myspace/f
  -d="function() { return 'hello world' }"
```

- GET returns function representation

```
curl http.../services/myspace/f
> Function<function() { return 'hello world' }>
```

- POST invokes the function

```
curl -X POST http.../services/myspace/f
> hello world
```

Eden-ws architecture

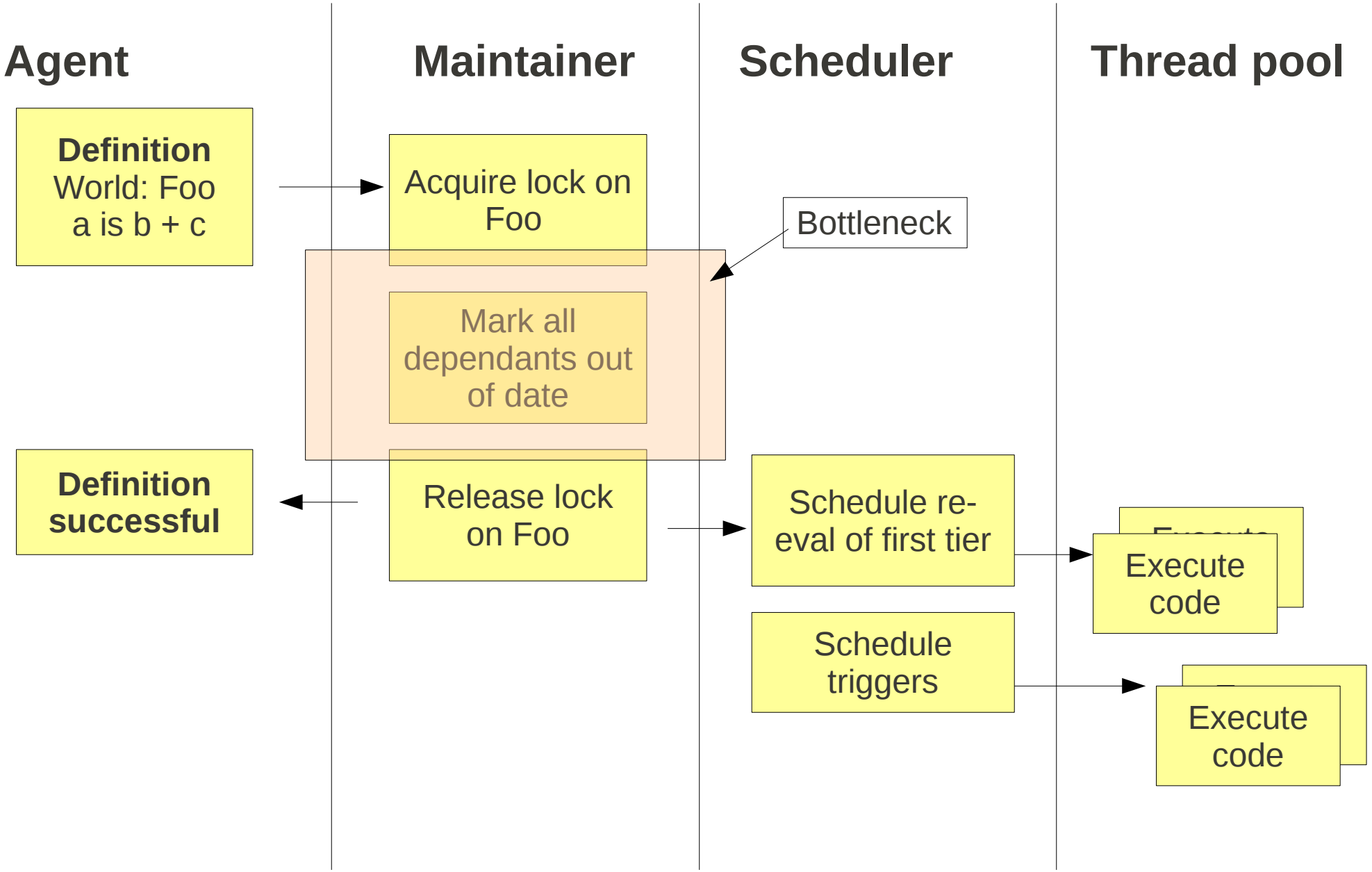
- Definitive principles on the server
- Can use these to provide the intra-app state management in front of traditional storage
- Other eden-ws apps can integrate via HTTP as agents using LSD style semantics
- Other applications can integrate over standard HTTP
  - Either routine polling of Etags via HTTP
  - Or via message-driven middleware driven by HTTP exposure
  - Or another eden-ws app could implement a triggered proc to enqueue a message

Will it scale?

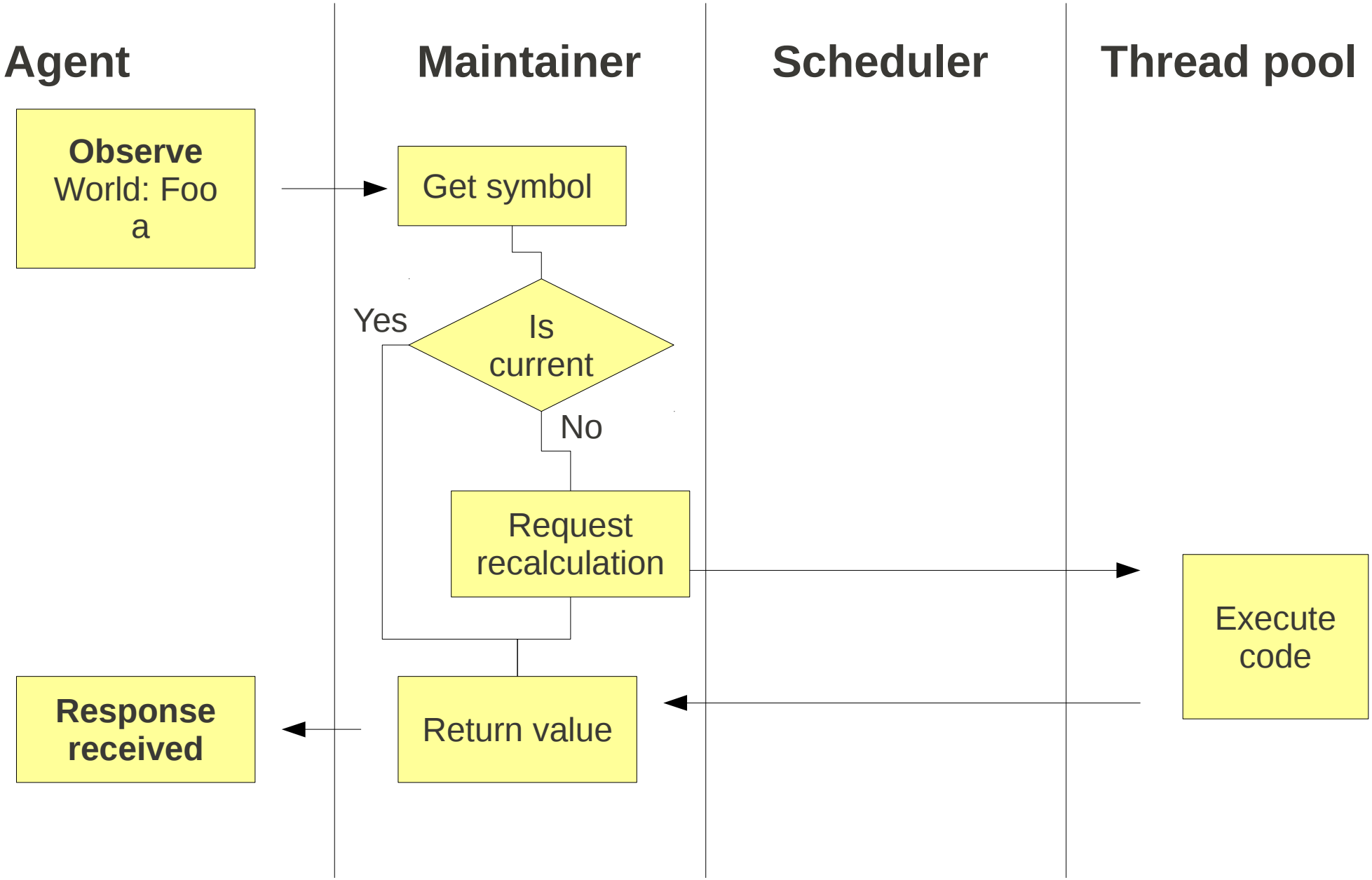
- Simple to provide dependency/agency in single thread
- Can imagine **k** threaded operation
  - Single definition thread with worker pool?

- But what about **n** threaded operation?
- What about enterprise grade
- What about 25,000 redefinitions a minute?

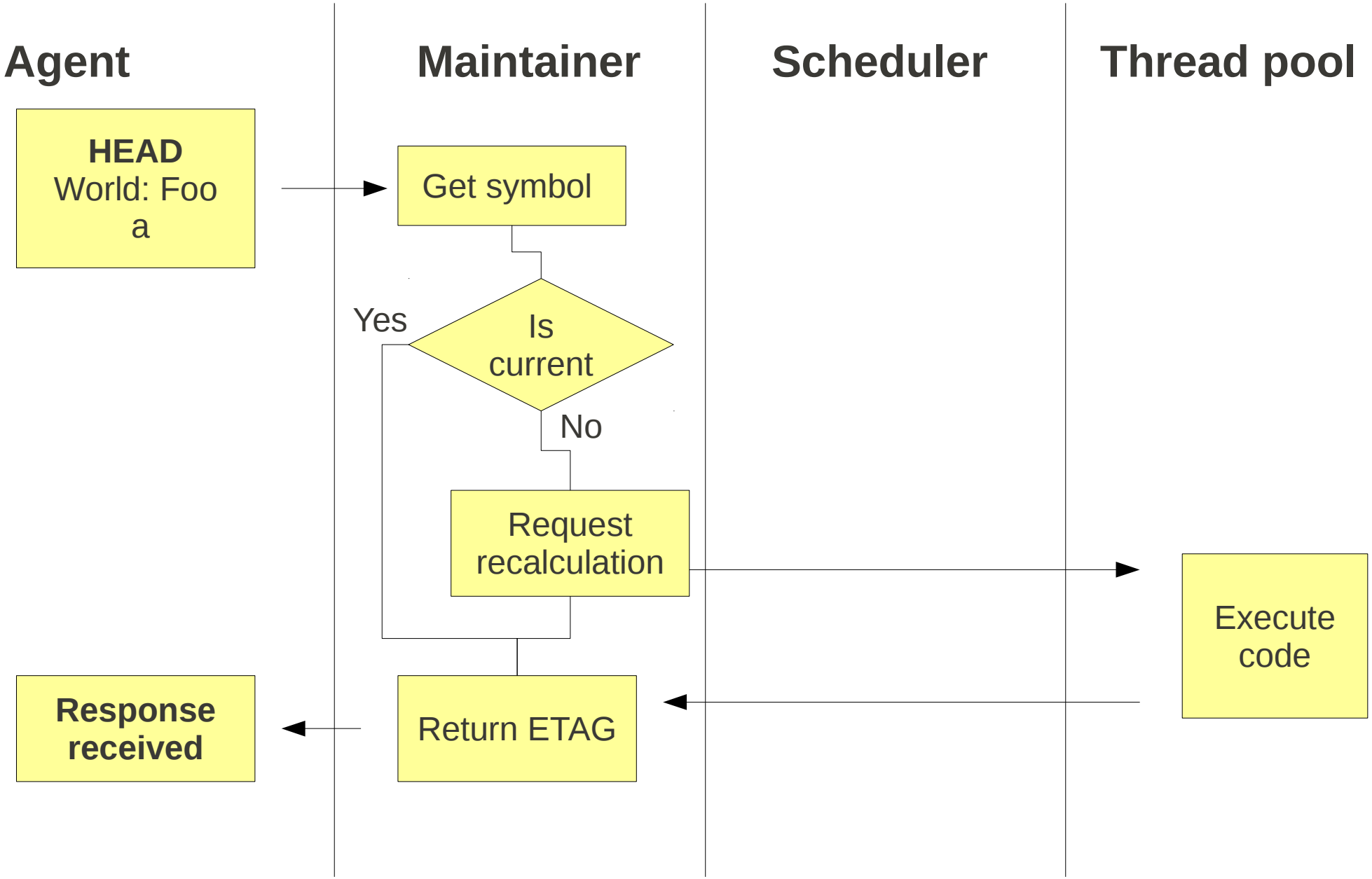
EM over HTTP – redefinition



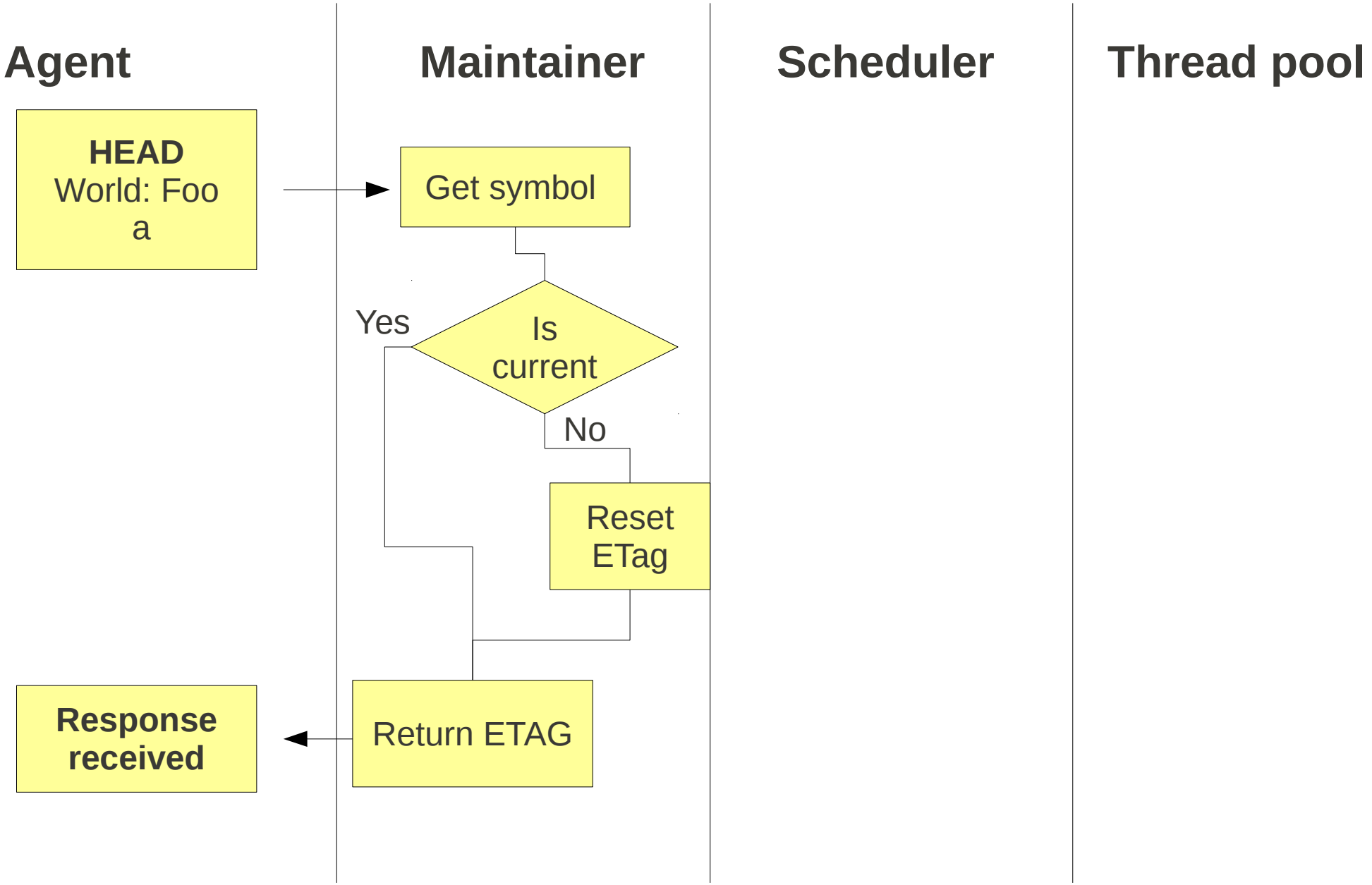
EM over HTTP – observation



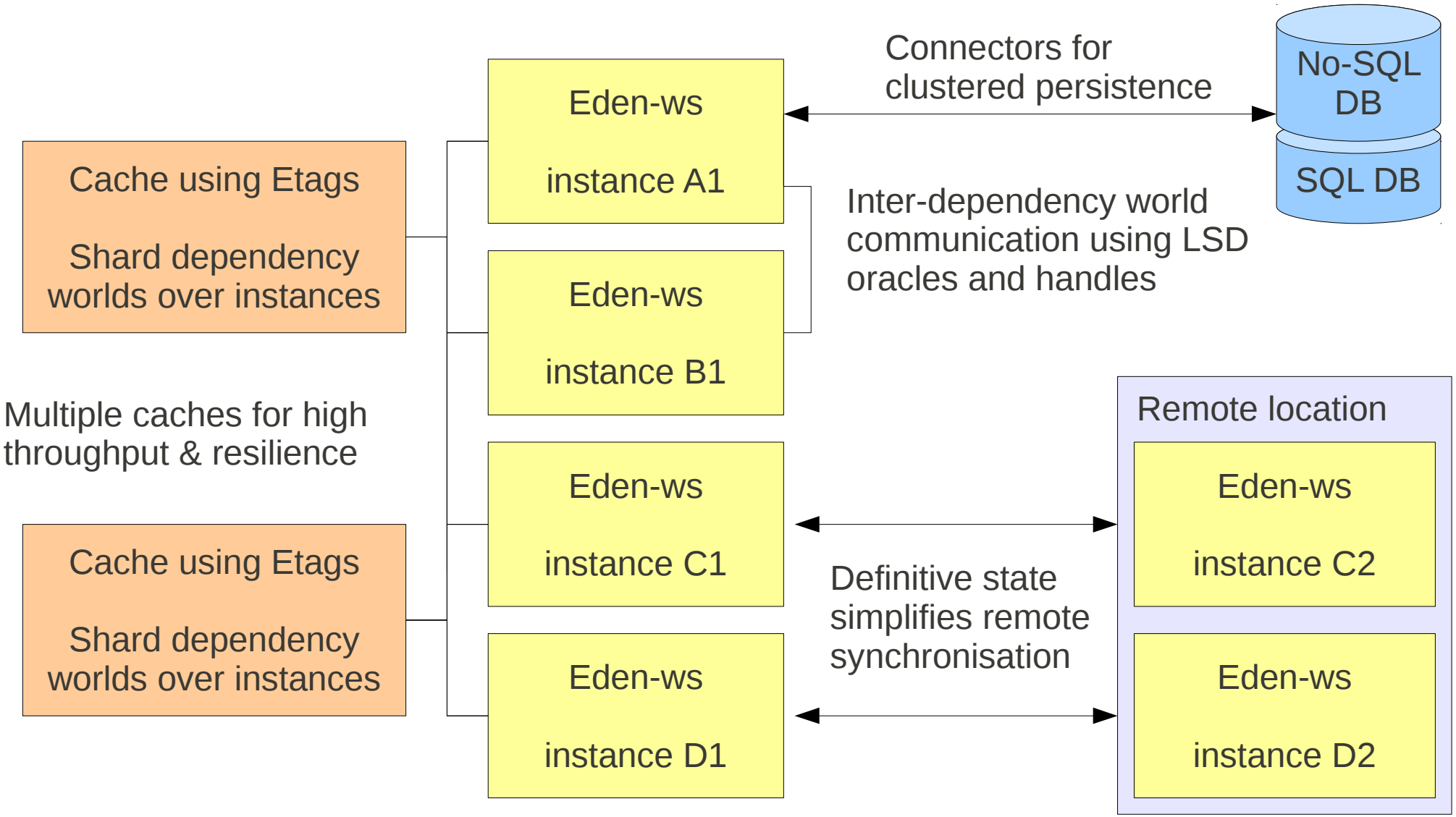
Observe whether up to date? (HEAD)



EM over HTTP – Fast HEAD

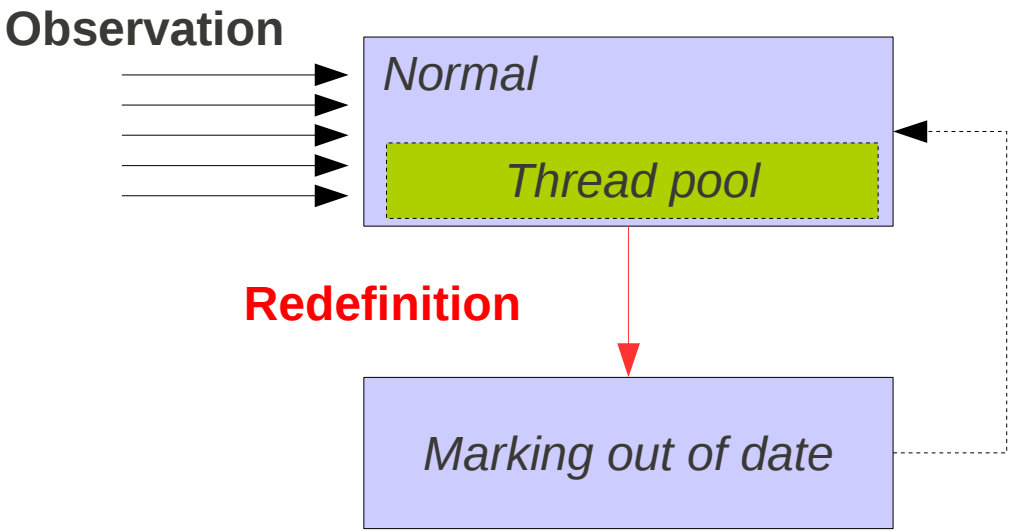


Scalable architecture?



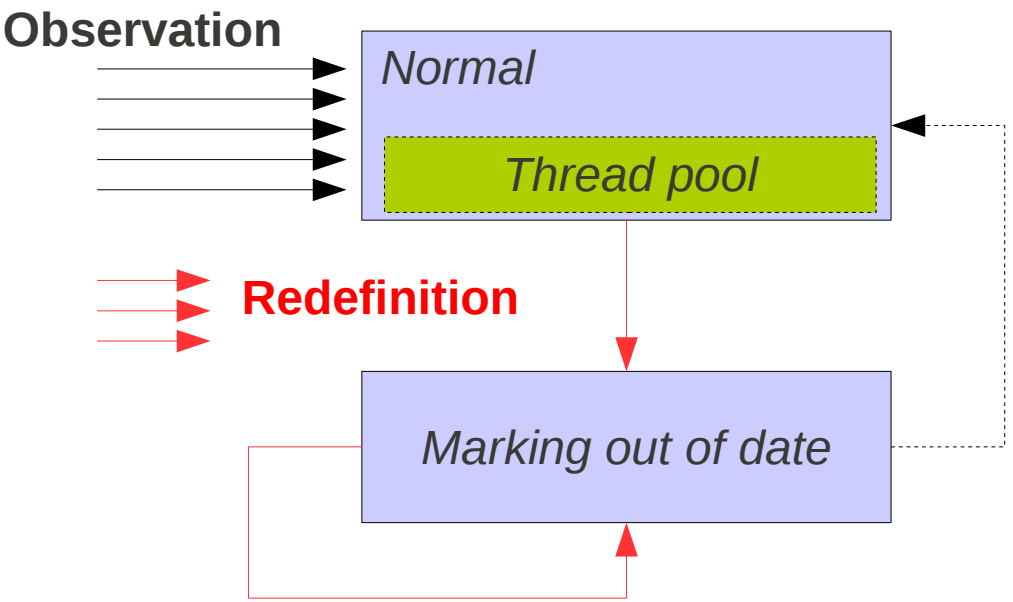
EM over HTTP – high throughput

- Simple two-state model
  - Synchronise on redefinition
  - Fast for reads
  - Slow for writes
  - *But can we do better?*



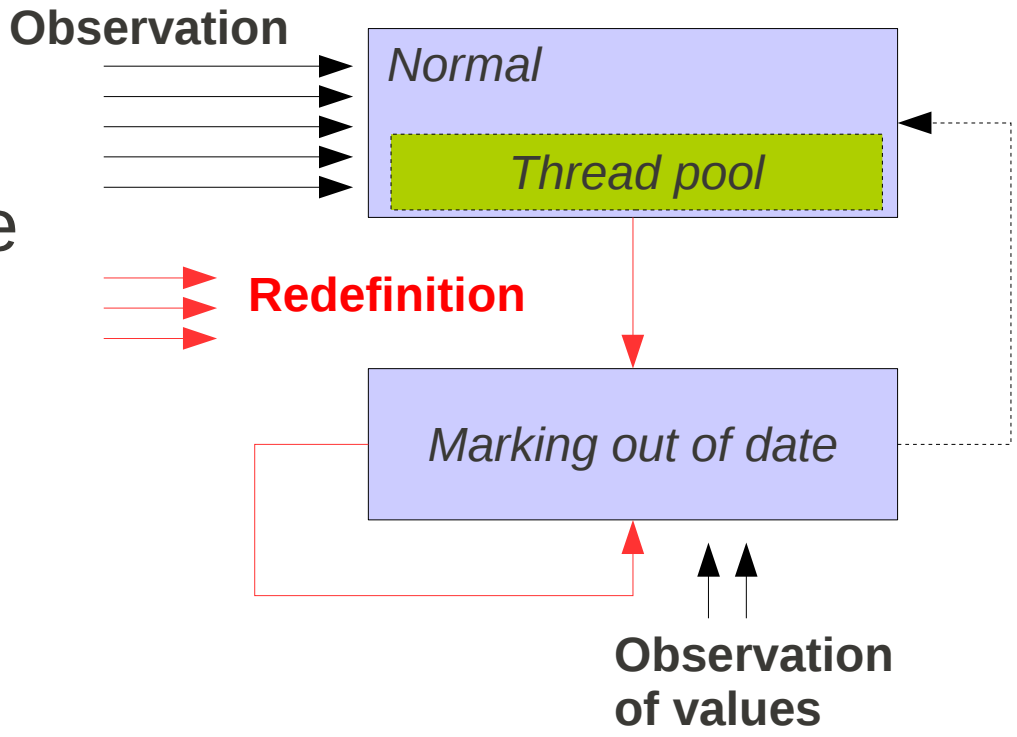
EM over HTTP – high throughput

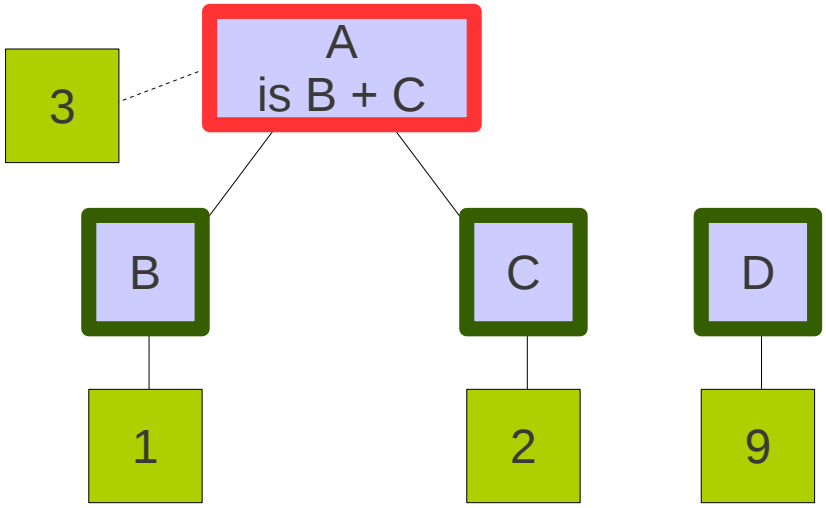
- Multiple redefinitions at once?
  - Synchronise on redefinition
  - Fast for reads
  - Slow for writes
  - *But can we do better?*



EM over HTTP – high throughput

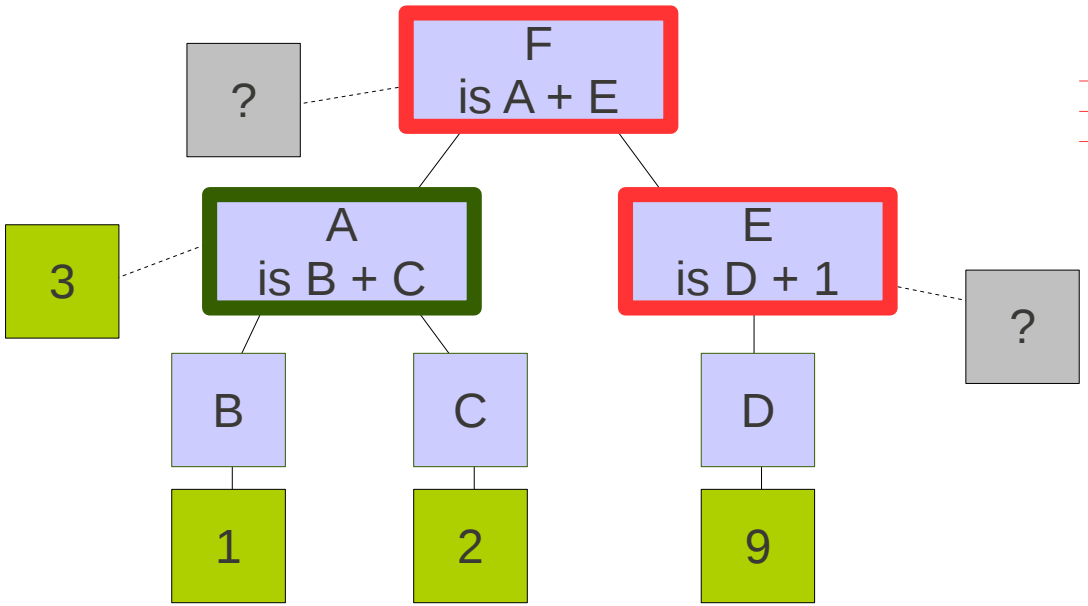
- Observe during redefinition?
  - Values can be observed at any time

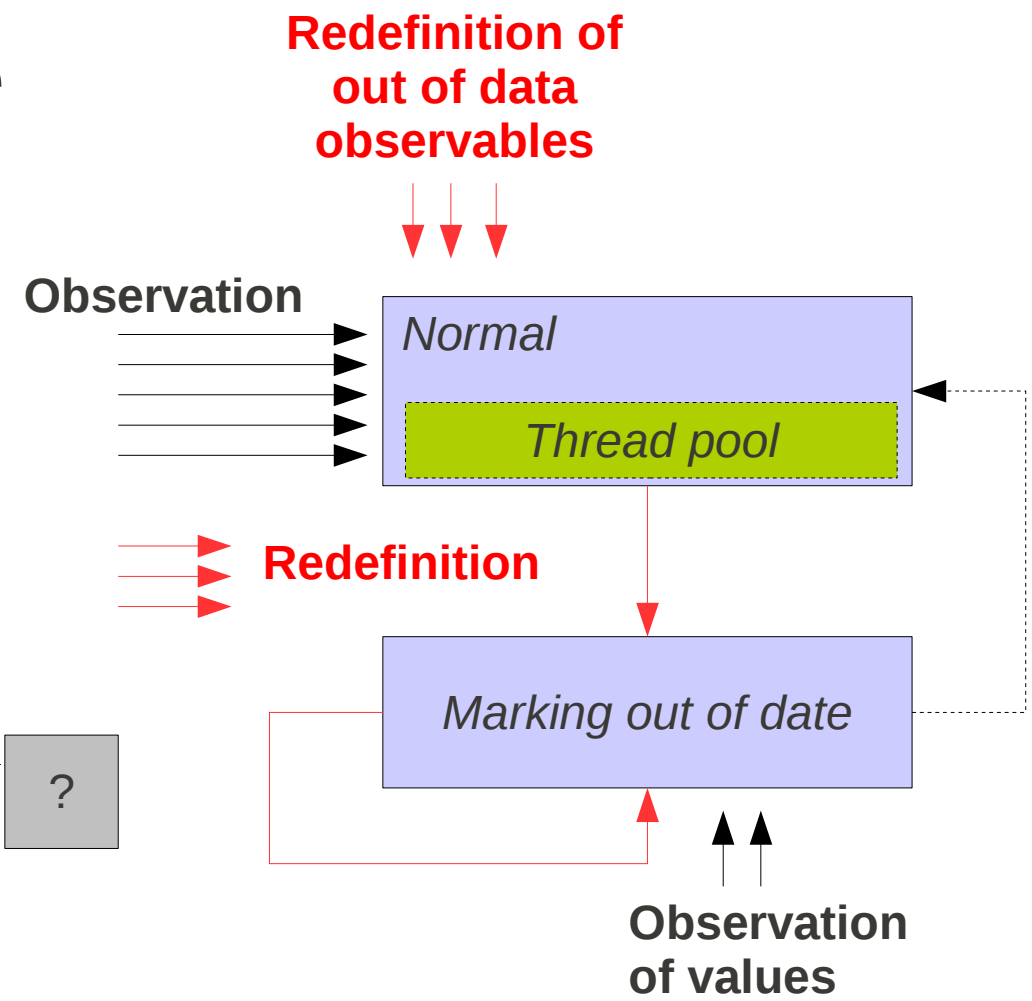




EM over HTTP – high throughput

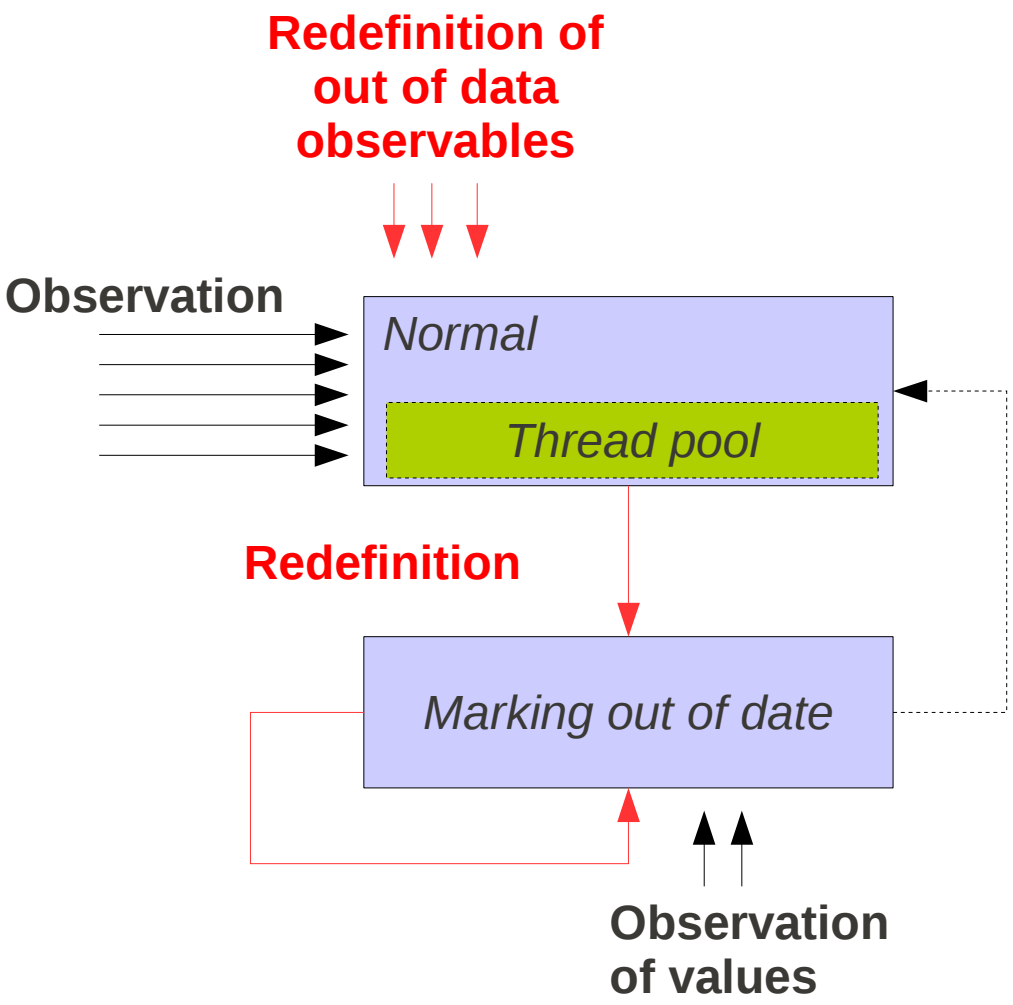
- Does redefinition require state change
  - *if it's already out of date... then it's possible to redefine.*





EM over HTTP – high throughput

- Repeated reads
- Repeated writes
- Lazy calculation
- Definition-driven caching
- Pre-emptive calculation?
  - Generational?
- Speculative calculation



This Lecture – outline

- Introduction
- Section 1: Scene setting
- Section 2: Maintaining state within the application
- Section 3: Maintaining state between applications
- Section 4: Blue sky
- **Conclusions**

Conclusions

- Within the Application
  - We can be pragmatic, follow patterns, and when coding UIs, use features like data-binding in an EM way
  - Data-binding is nice, but need to understand it's not dependency - can create Event overload
- Between applications
  - Dependency and agency provide an interesting caching technology
  - Performance benefit of not constantly walking hierarchical data.
  - Key thing is that the engine can do intelligent expiry
- Some things will always be done with traditional integrations
- But an Rest exposure over http opens a lot of doors
  - Caching, mash-ups, cheap integrations

And final thoughts...

- Events are difficult to debug
- Dependency is easy to debug
- IllegalStateException should not happen
- Observing state over http makes a lot of sense

Links to code

- Code snippets from this lecture
  - https://github.com/ccare/EM-lecture-snippets
- To grab eden-ws code
  - https://github.com/ccare/eden-ws