In-process REST

Marcel Weiher

@mpweiher

In-process REST

Simplify code by applying REST principles without distribution

In-process REST

- BBC SportStats
- Polymorphic Identifiers / Objective-Smalltalk
- Wunderlist

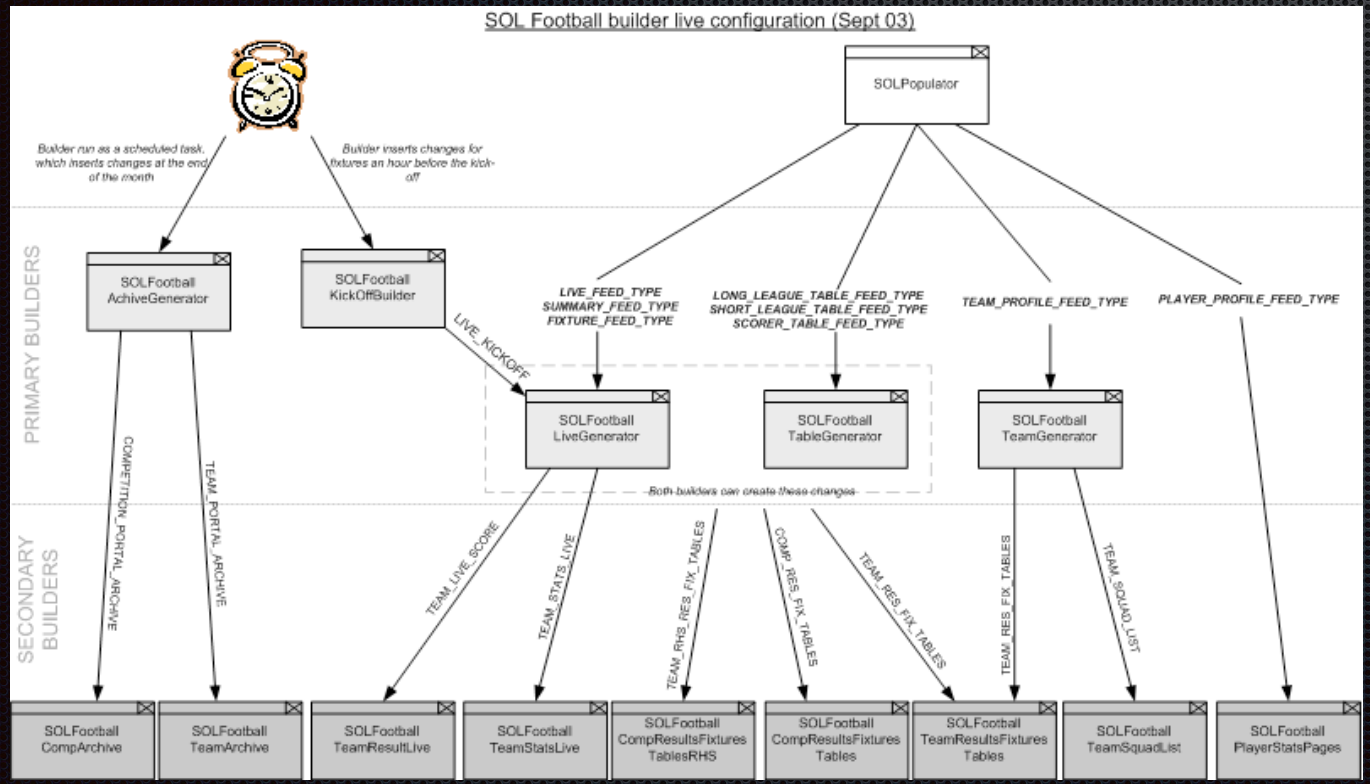
In-process REST

- BBC **SportStats**
- Polymorphic Identifiers / Objective-Smalltalk
- Wunderlist

BBC SportStats

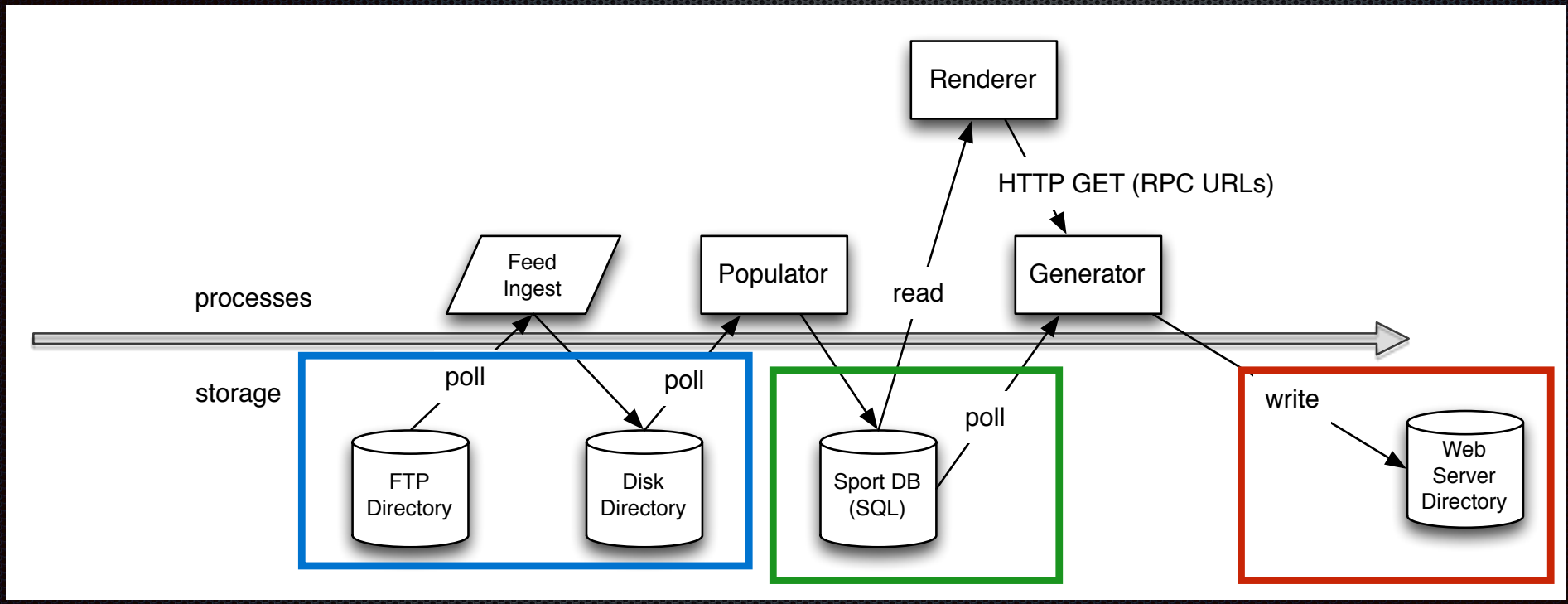
- XML Provider → BBC web-site (Ceefax, Mobile)
- Fixtures, Live Scores/Coverage, Results, Tables
- Replace existing system

BBC SportStats: Before



- Slow
- Lots of machines
- >100 processes
- Fragile
- No traceability

BBC SportStats
v1 Architecture Diagram (SOA)



BBC SportStats Generator URLs

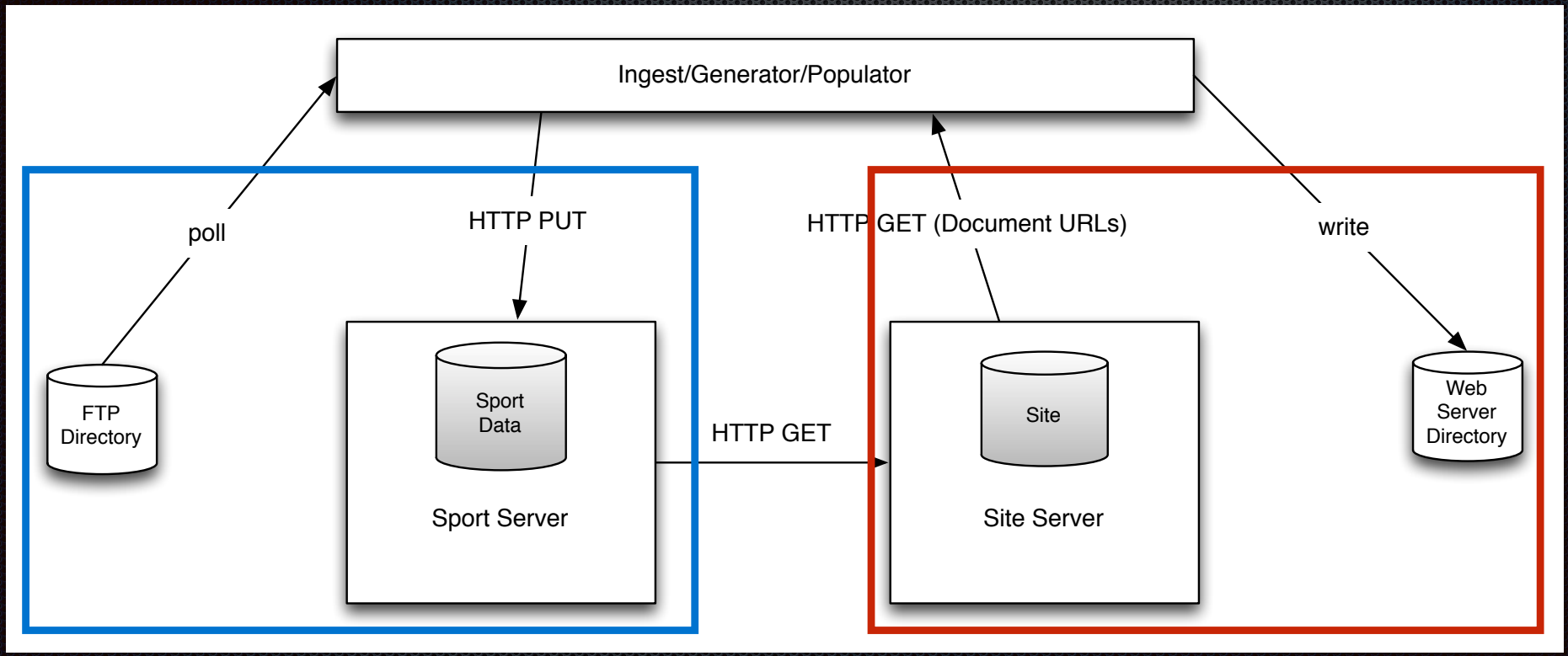
```
http://nolparser07/scripts/WebObjects.exe/SOLPageRenderer.woa
```

machine | process

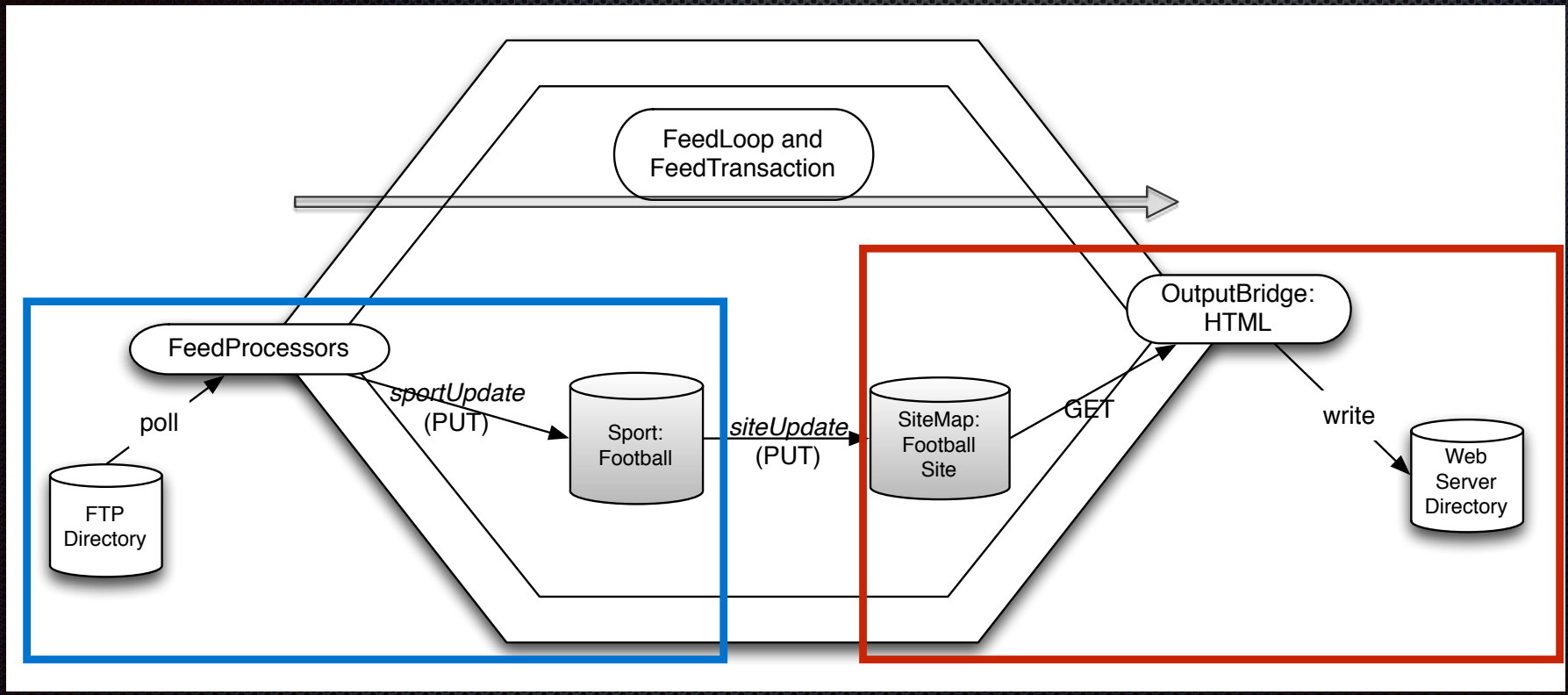
```
/wa/SOLRugbyCompResultsSmall?Competition=7331&type=Competition
```

page-type | arguments

BBC SportStats
RESTful re-imagining (μ-services)

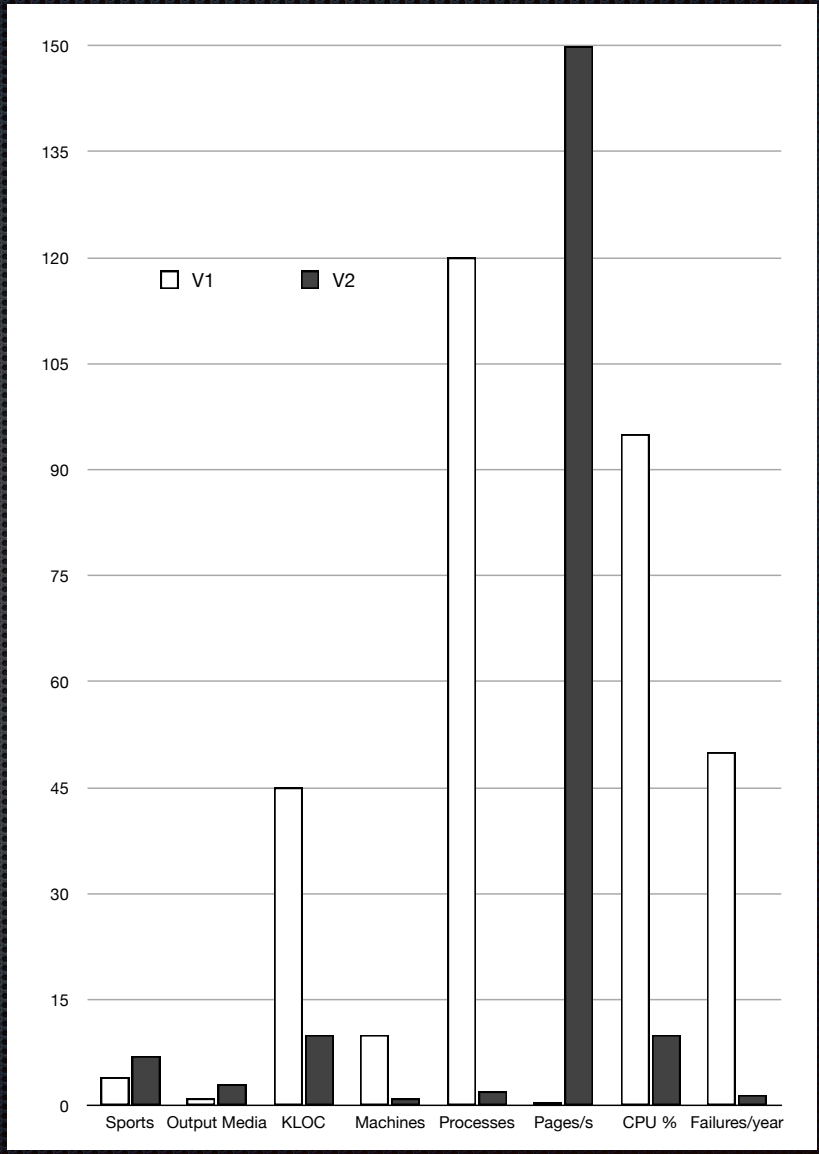


BBC SportStats
v2 In-process REST



BBC SportStats After

- Code (KLOC): 45 vs 12
- Sports: 4 vs 7
- Output Media: 1 vs 3
- Machines: 10 vs 1
- Pages/s: 0.5 vs 150
- Failuers/year: 52 vs 1



BBC SportStats Reflection

- Simplified: “forgot” to implement the DB
- Simplified: from SOA to REST
- Simplified: REST servers brought In-process
- Simplicity, performance, reliability, …
- But…

BBC SportStats
Stringly referenced

```
public void buildTheActualSite( Sport aSport ) {
    super.buildTheActualSite( aSport );
    football=(Football)aSport;
    footballDir = sportBase();
    sportStaticBase = root().mkdirs(staticSportPath());
    players = sportStaticBase().mkdir( "players" );
    teamsStatic = sportStaticBase().mkdir( "teams" );
    playerRaterData = root().mkdirs("/sol/shared/spl/hi/playerrater/data/games");
    playerRaterIncludes = root().mkdirs("/sol/shared/spl/hi/playerrater/inc/games");
```

In-process REST

- BBC SportStats
- **Polymorphic Identifiers / Objective-Smalltalk**
- Wunderlist

Objective-Smalltalk & Polymorphic Identifiers

- Objective-Smalltalk: Architectural Programming
- Stringly-referenced programming: everywhere
- Polymorphic Identifiers: custom identifiers / URIs in programs

Objective-Smalltalk

- Custom connectors instead of fixed meta-level
- Messaging → Higher Order Messages
- Assignment → Dataflow and Constraints (Friday!)
- Identifiers → Polymorphic Identifiers
- http://objective.st

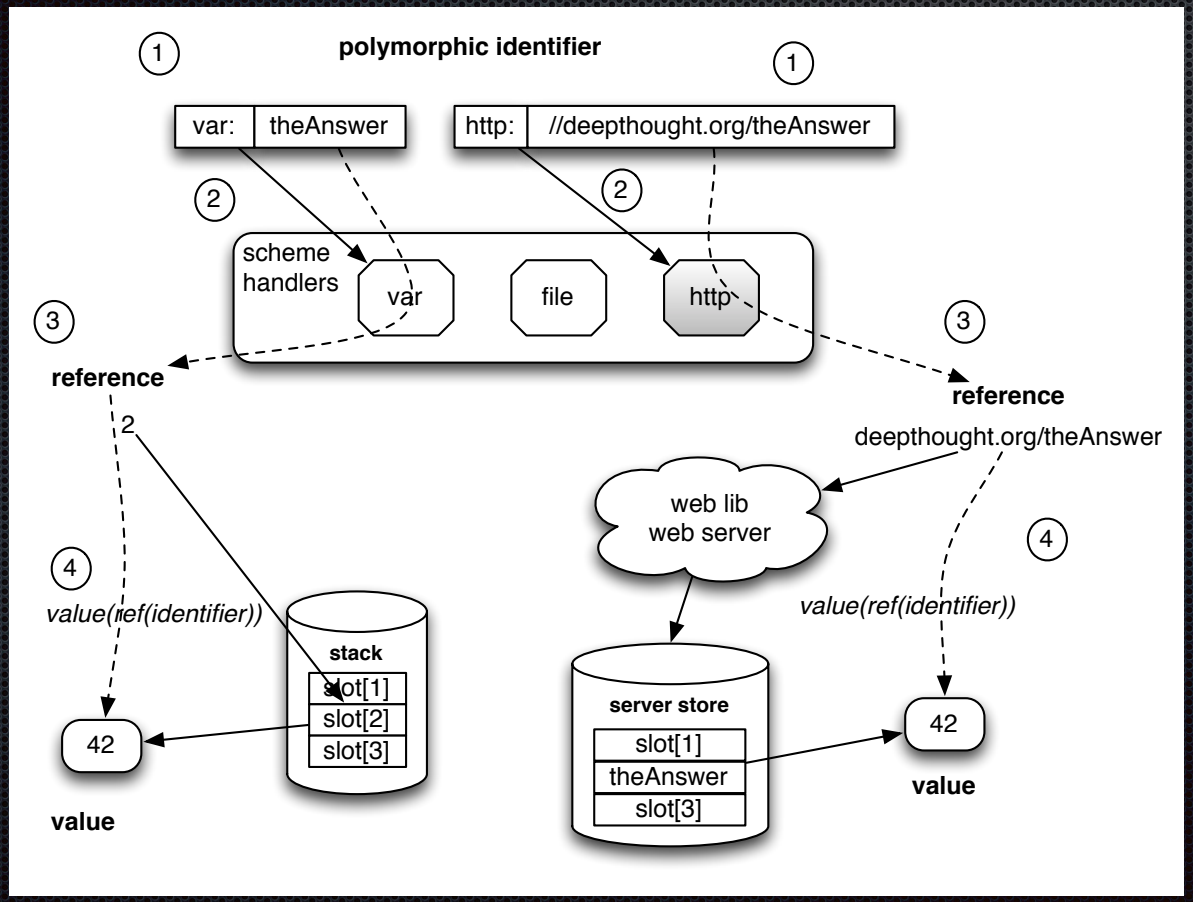
Stringly-Referenced

```
html = [NSData dataWithContentsOfURL:[NSURL
URLWithString:@“http://www.amazon.com"];

[@“Hello World” writeToFile:[@“/tmp/
stringByAppendingPathComponent:name];

textField.intValue = [[NSDefaults
standardUserDefaults] integerForKey:@“celsius”];
```

Polymorphic Identifiers

```
html ← http://www.amazon.com/
file:/tmp/{name} ← “Hello World”

textField/intValue ← defaults:celsius
```

Polymorphic Identifiers



Polymorphic Identifiers
Http Server Example 1

```
#!/usr/local/bin/stsh

scheme:site ← MemoryStore scheme.
server ← SchemeHttpServer serverOnPort:8081
site:/hi ← 'Hello World!'.

(scheme:site → server) start:nil.
```

Polymorphic Identifiers
$HOME HTTP Server

```
#!/usr/local/bin/stsh

server ← SchemeHttpServer serverOnPort:8081
source ← ref:file:{env:HOME}/Sites asScheme.


(source →          server) start:nil.
```

Polymorphic Identifiers
$HOME HTTP Server, Cached

```
#!/usr/local/bin/stsh

server ← SchemeHttpServer serverOnPort:8081
source ← ref:file:{env:HOME}/Sites asScheme.
cache  ← MemoryStore scheme.

(source → cache → server) start:nil.
```

Polymorphic Identifiers Storage Combinators

- Cache: caches one scheme using another
- Sequential: searches a list of schemes
- Filter: transforms identifiers and/or values
- Switcher: use one of a set of schemes

In-process REST

- BBC SportStats
- Polymorphic Identifiers / Objective-Smalltalk
- **Wunderlist**

In-process REST Wunderlist

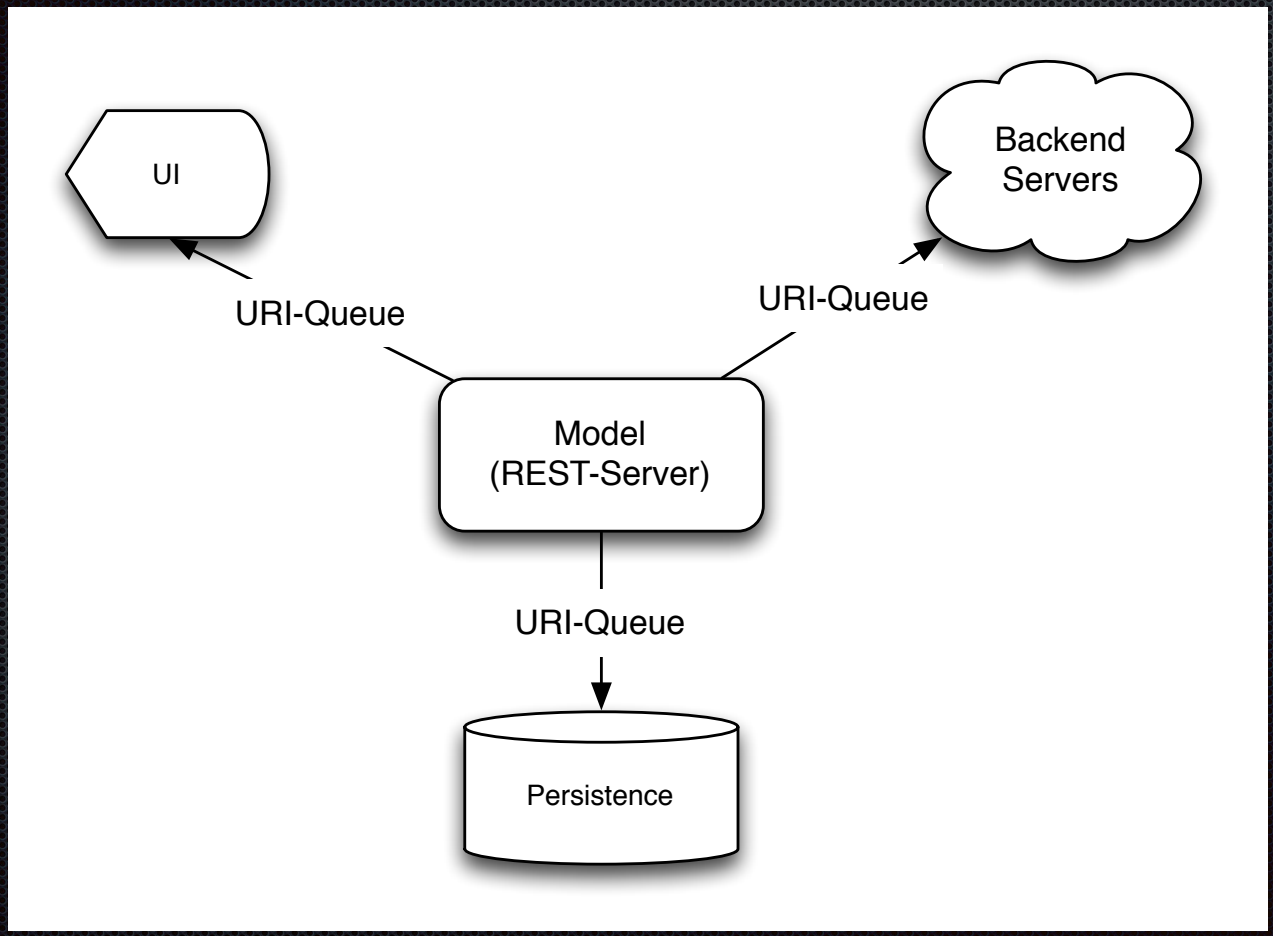
- Task management application
- Native clients for iOS, Mac, Android, Windows + Web
- 5 million active users
- Acquired 2015 by Microsoft

In-process REST Wunderlist

- URIs (ObjectReference) refer to model objects
- URIs can reference non-existent objects
- URIs can be persisted
- URIs are structured:

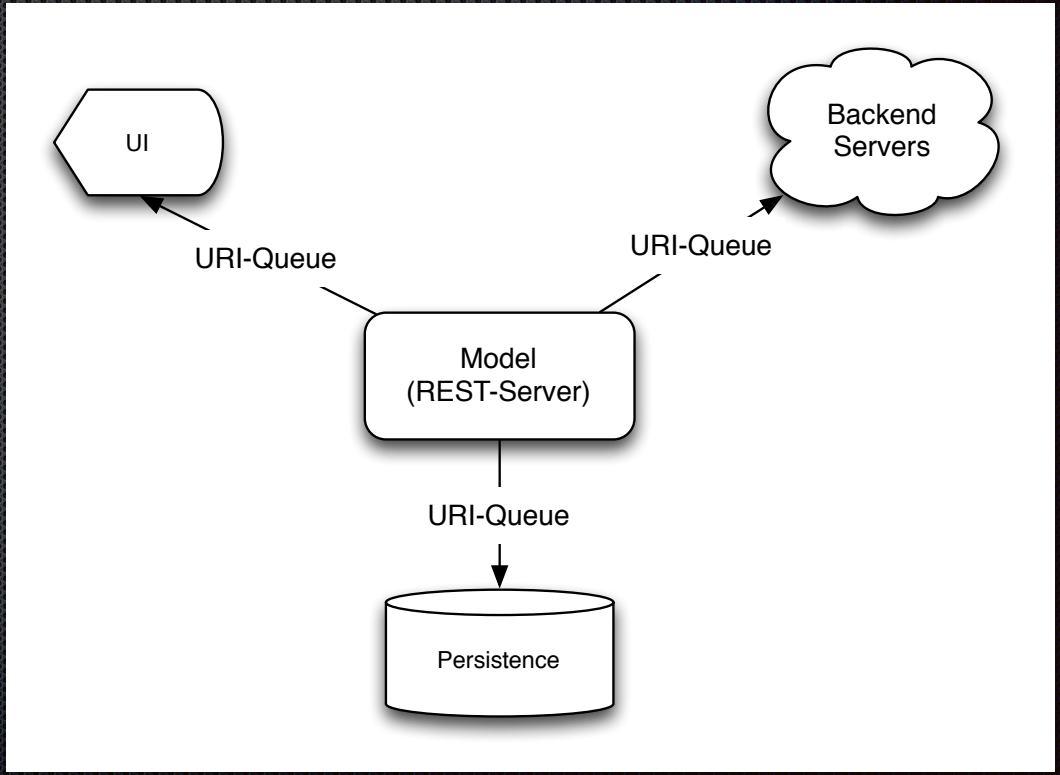
  wlstore:task/container/1/object/1
  wlstore:task/container/1

Wunderlist


UI
Backend Servers
URI-Queue
URI-Queue
Model (REST-Server)
URI-Queue
Persistence

URI-Queues

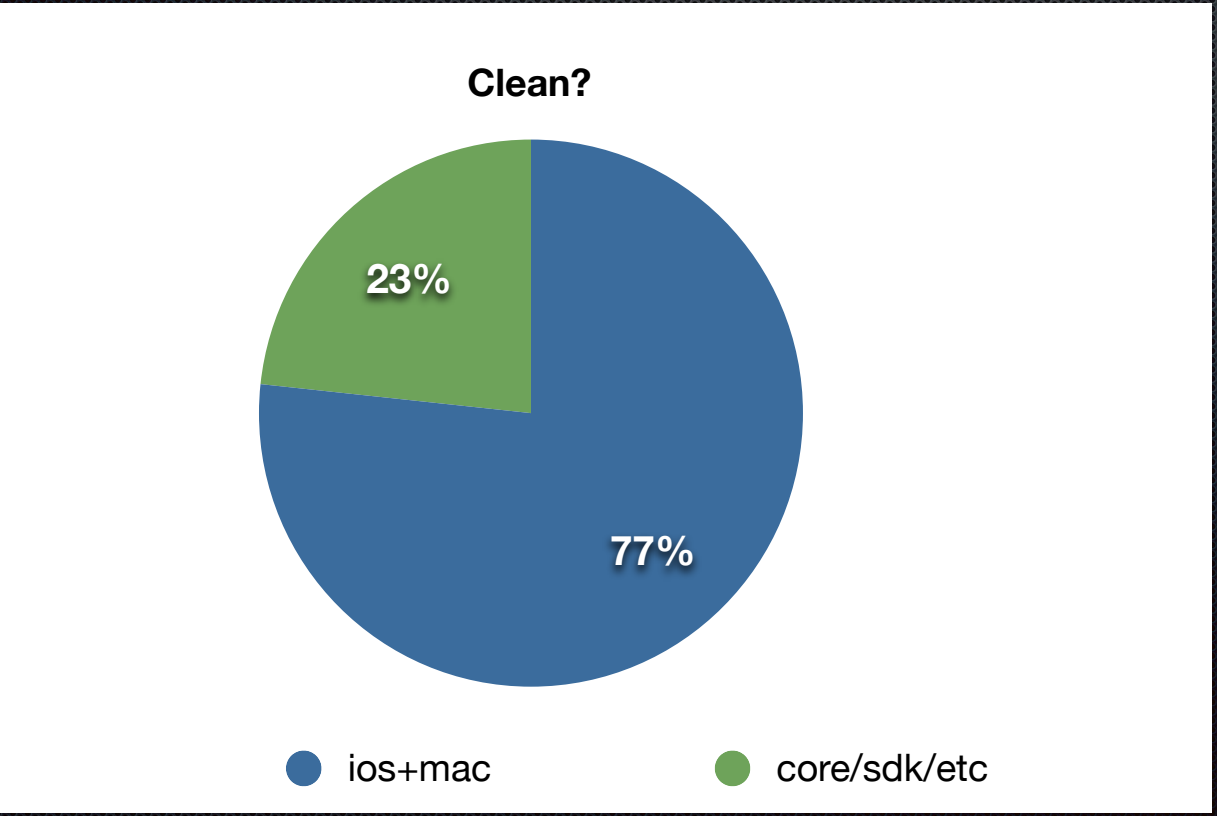
- A queue of references
- Asynchronous
- Persistable (network)
- Uniqued (network)
- “Bucketized” (storage)
- Dynamic coalescing/throttling (UI)



In-process REST Wunderlist 3

- Reliability: 10x
- Performance: animations, 60 fps under heavy load
- Simplicity
- But…

In-process REST Wunderlist UI Code



In-process REST Wunderlist UI

- Apple MVC: Massive ViewController
- Glue code: C is unique for every combination of V + M

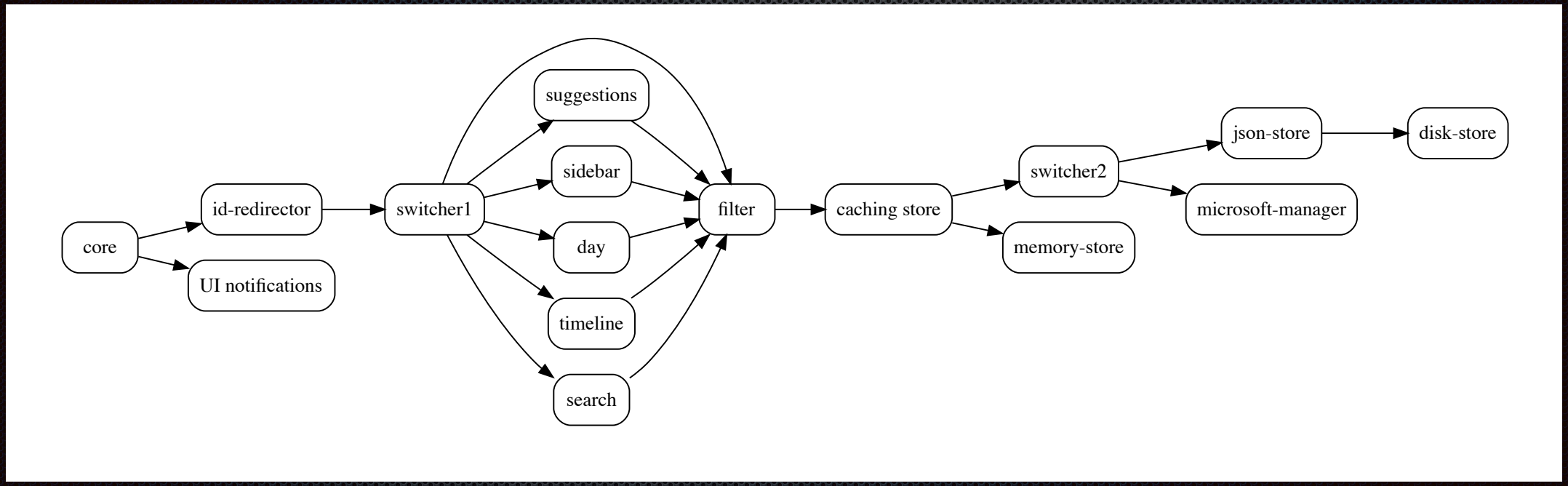
In-process REST Wunderlist UI

- Generic views: parametrised with URI
- Controllers
- Model: composite stores evaluate URIs

In-process REST Wunderlist UI

- Generic views: parametrised with URI
- Model: composite stores evaluate URIs

In-process REST Wunderlist Store Hierarchy



In-process REST Wunderlist UI

- Client-specific UI code reduced by at least 10x
- Virtually no glue code
- MVc

In-process REST: Q&A

- URIs + storage
- Reduces code (10x-20x)
- Language support
- Friday 15:50: Reactive Dataflow Constraints

- http://objective.st/ @mpweiher