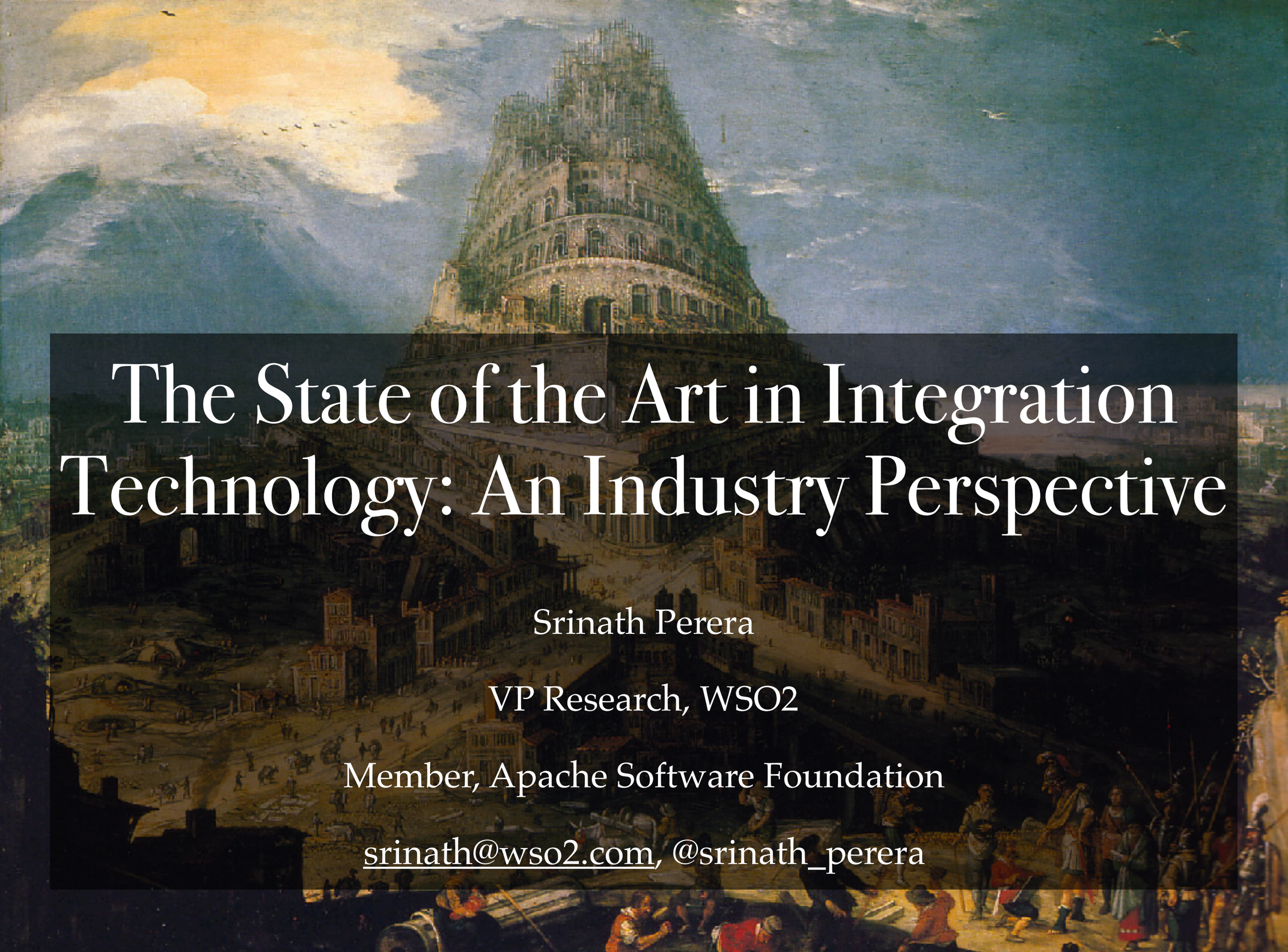


The State of the Art in Integration Technology: An Industry Perspective

Srinath Perera

VP Research, WSO2

Member, Apache Software Foundation

srinath@wso2.com, @srinath_perera

Part 1: Walk down the Integration Memory Lane

My Memory Lane (15 years)

@WSO2: Has helped design, worked, debugged

- Workflow Engine (Apache Ode)
- Identity Server
- Application Server
- Message Broker
- Stream Processor
- ESB (Apache Synapse)

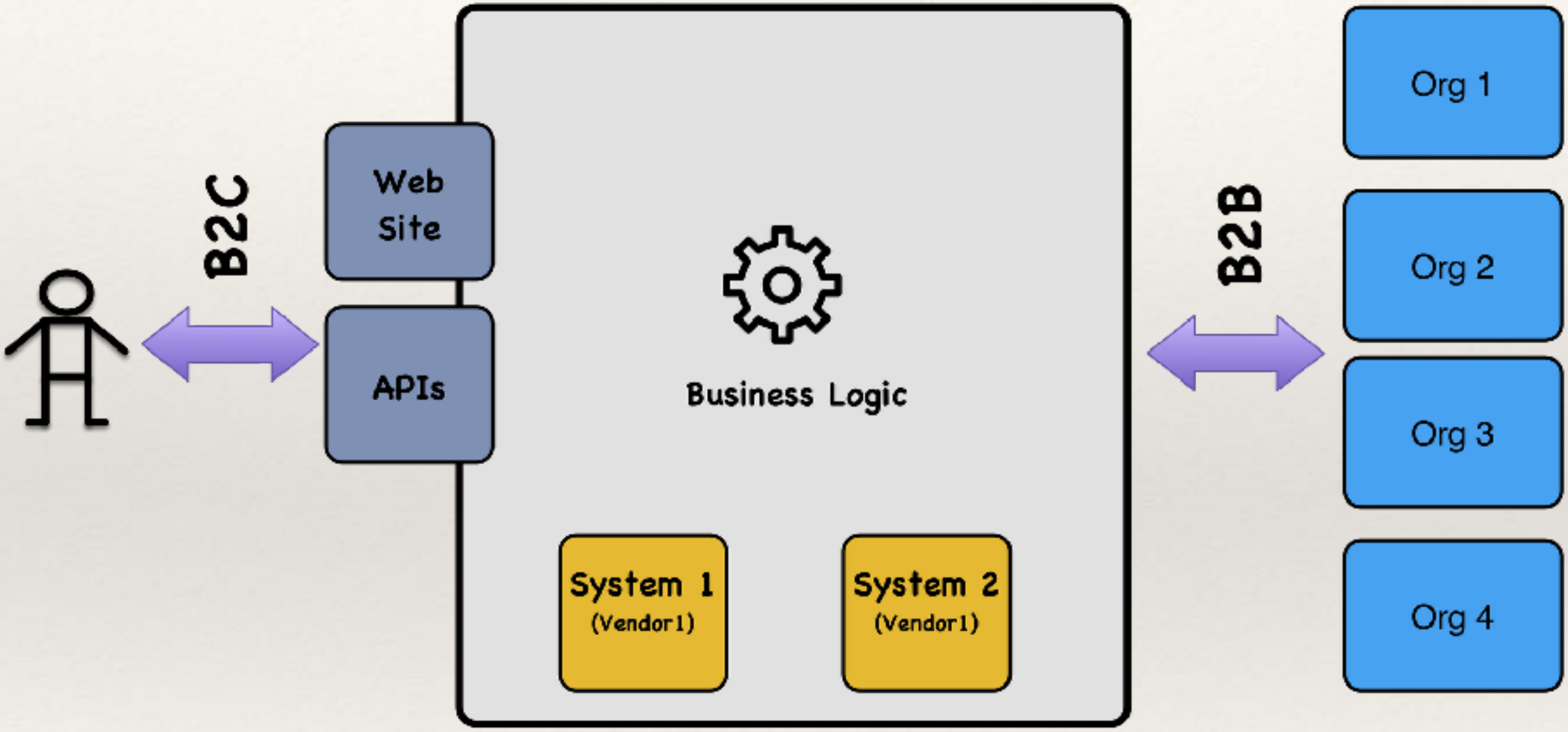
5-10 release each

Worked in a J2EE Server

Wrote two SOAP Engines and worked in one more

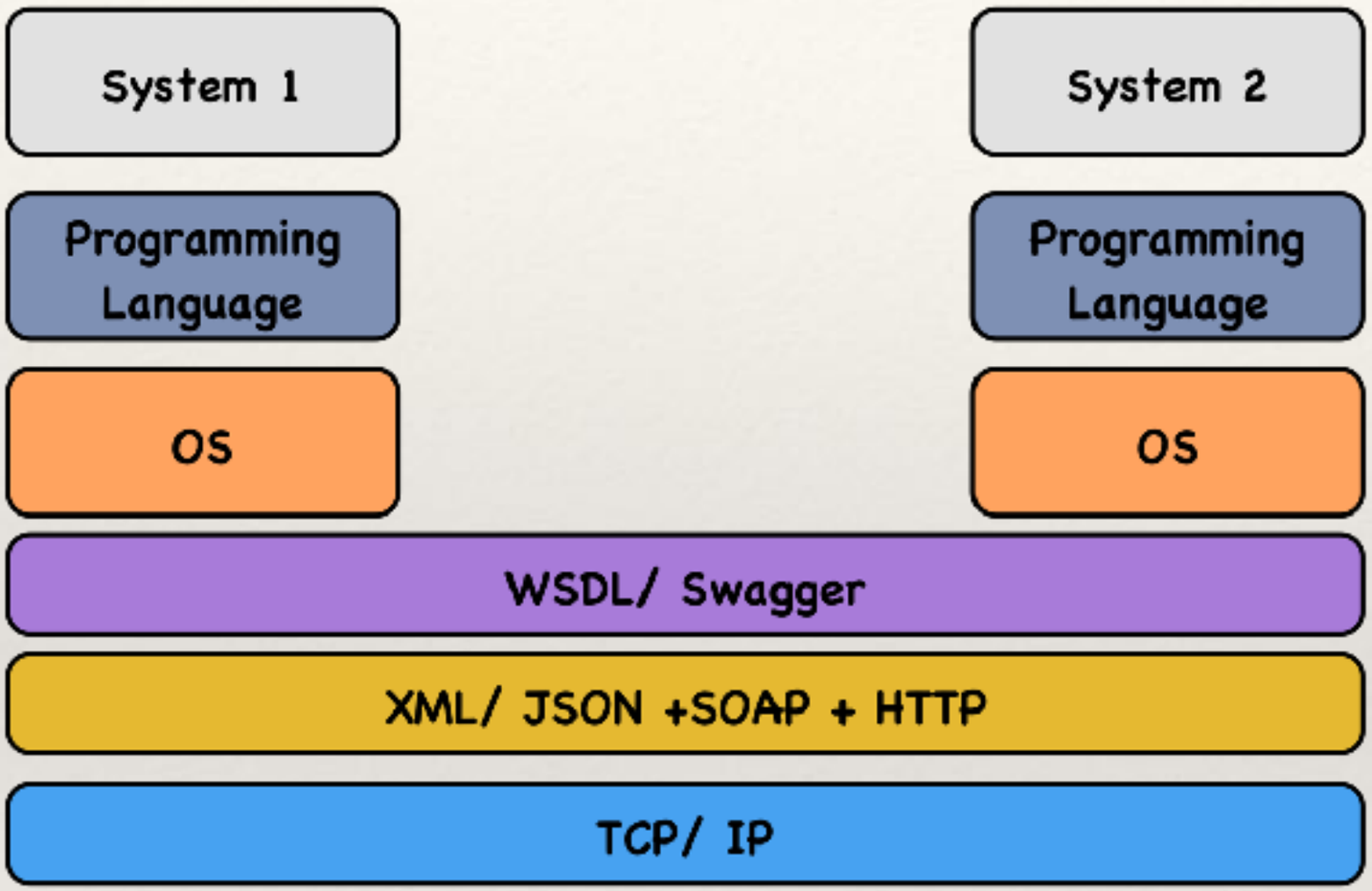
WSO2 (Opensoure middleware company 500+ employees, 300+ customers including BNY, TFL, Boeing, Fidelity, Ebay ..)

Enterprise Systems



(e.g. Book a flight that involve two airlines)

Interoperability



1st try: Proprietary protocols (RMI, COM)

2nd Try: CORBA

3rd Try: Web Service

Agreeing on the wire via XML worked

Back to binary protocols? Thrift, Protobuf, Avro

Lessons

Simplicity Wins

(as you need developers to understand it)

UX can decide
technology fate

Middleware: SOAP Engines

Service authoring and Hosting

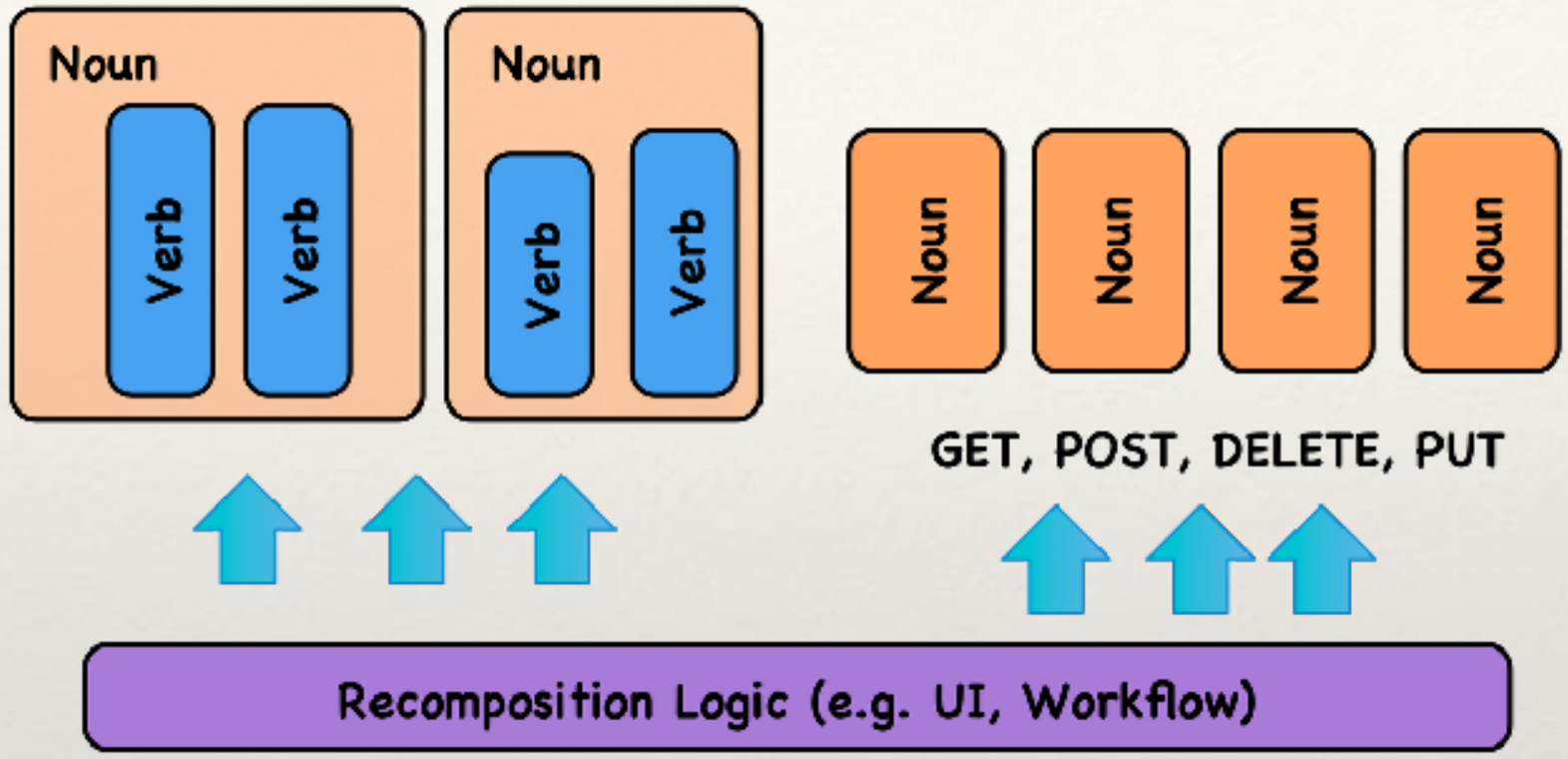
Code first vs. contract (WSDL) first

e.g. **GSoap, Apache Axis, Apache Axis2, Apache CXF, .NET**

Later JAX-WS Specification for Java

REST / Mircoservices Frameworks (DropWizard, SpringBoot, WSO2 MSS4J

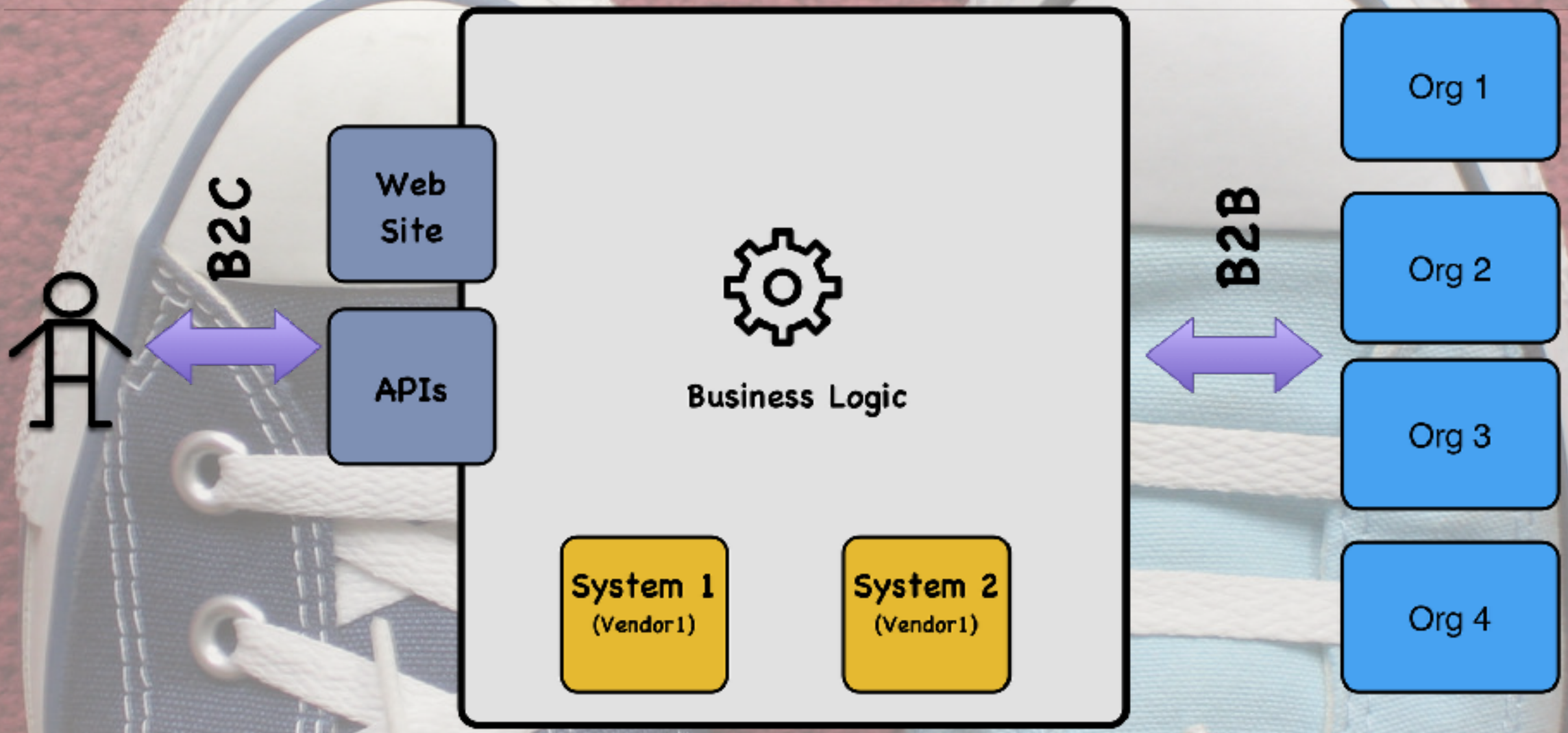
SOA vs. ROA



Build the architecture by recomposing loosely coupled Units

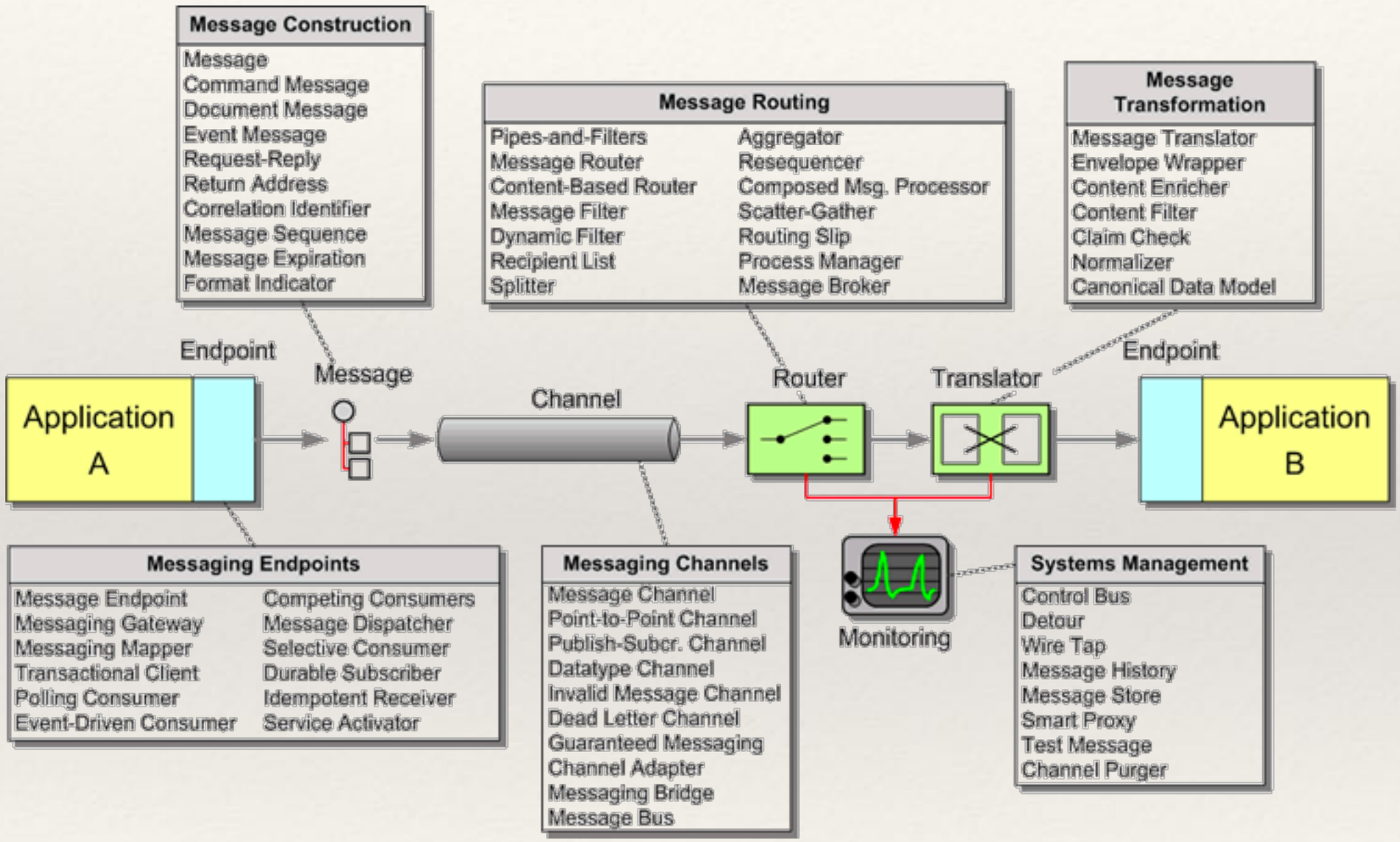
After lot of battles we kind of agreed to use one that is natural to the use case

Enterprise Integration



Integration is mediation

Enterprise Integration



(See EIP patterns (http://www.enterpriseintegrationpatterns.com/patterns/) for details)

CC BY Gregor Hohpe and Bobby Woolf

Middleware: ESB

Abstract service via proxies

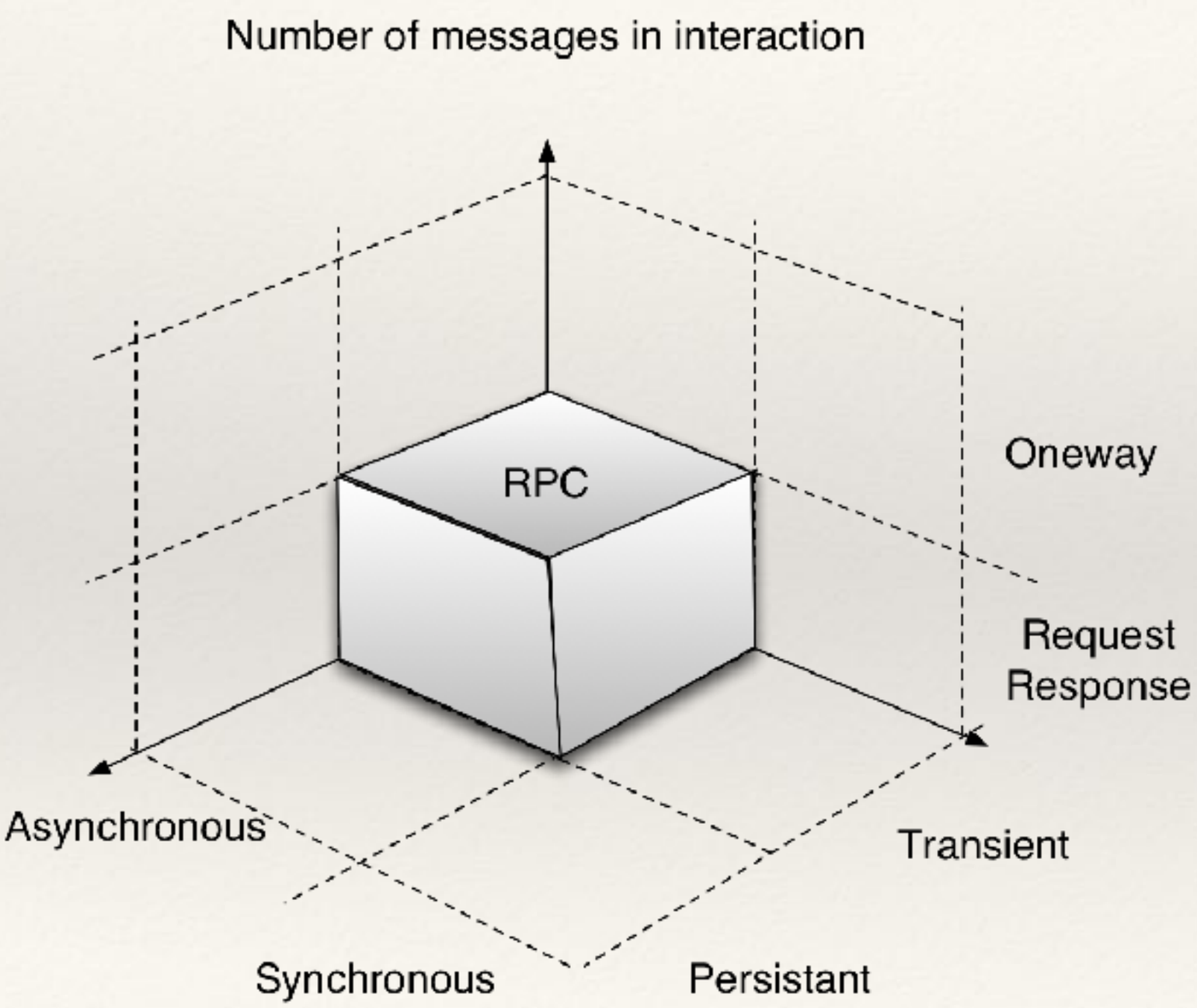
One place to all integration Logic (Avoid Spaghetti)

Simple script language to code mediation logic

Keep backend simple by terminating security etc

e.g. **WSO2 ESB, Mule, Boomi, Apache Camel**

Async Persistent Messaging



Decoupling: Time, Space, Synchronization

ESBs learnt to work with Message brokers (mainly JMS) and added operators like message stores and processors

Composition

Recomposition part of SOA/ROA

Most done via UI Code

Others done via Code

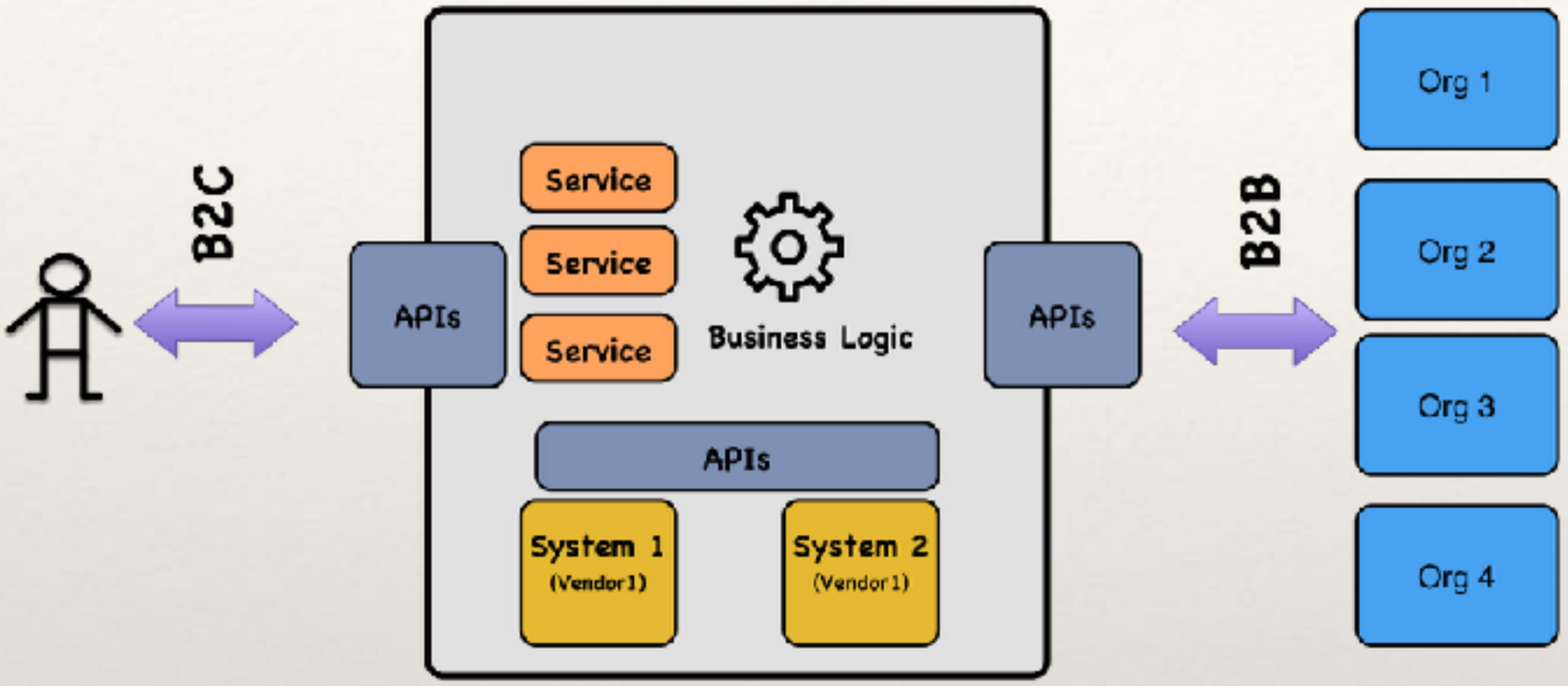
ESB can do it also

Short running

We use workflows for long running

Think a home loan, which will last longer than servers it will run on

APIs/ API Management



Crossing Trust boundaries adds complexity

- Secure the endpoint
- User provisioning

API management provided a packaged solutions

Also it acts as the API Registry

Enterprise Architecture in 2014

Had lot of services

Most service has more than one client

Services are recomposed at higher levels

Services are developed as one big system, or several big systems

They shared data bases

Deployment was complicated. Services couldn't evolve independently (although SOA always thought about them that way).

Microservices

simple, lightweight, loosely coupled services that can be developed and released independently of each other.

It is mostly set of good practices to build large systems

No shared database

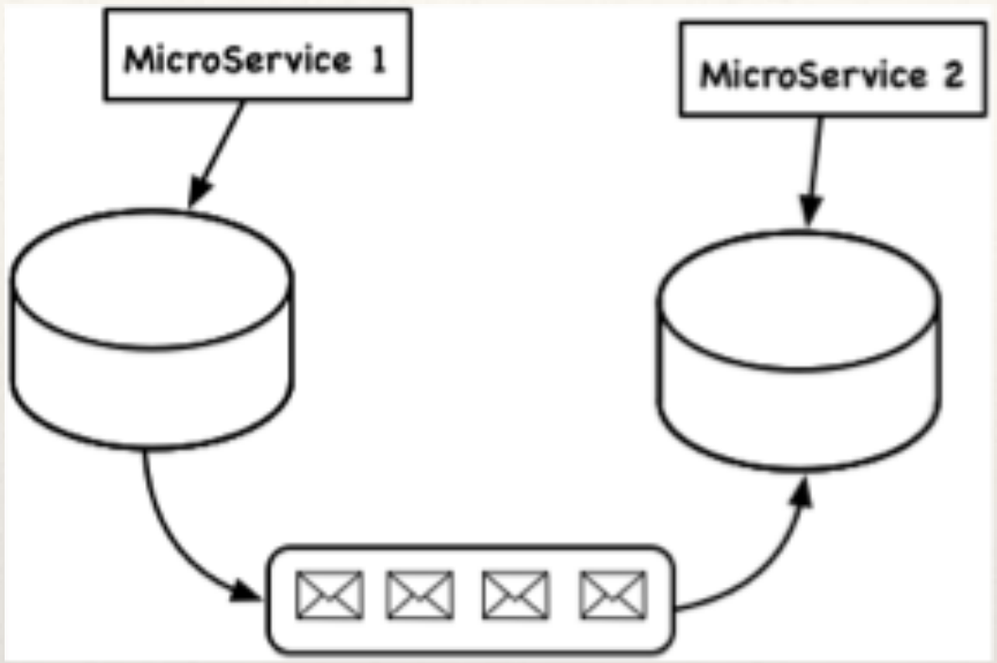
Services are versioned and backward

Avoid dependency hell

- Composition
- Security

I have written about this topic in details in https://medium.com/systems-architectures/walking-the-microservices-path-towards-loose-coupling-few-pitfalls-4067bf5e497a

No Shared Databases



To remove coupling between service implementations

What if two microservices share data?

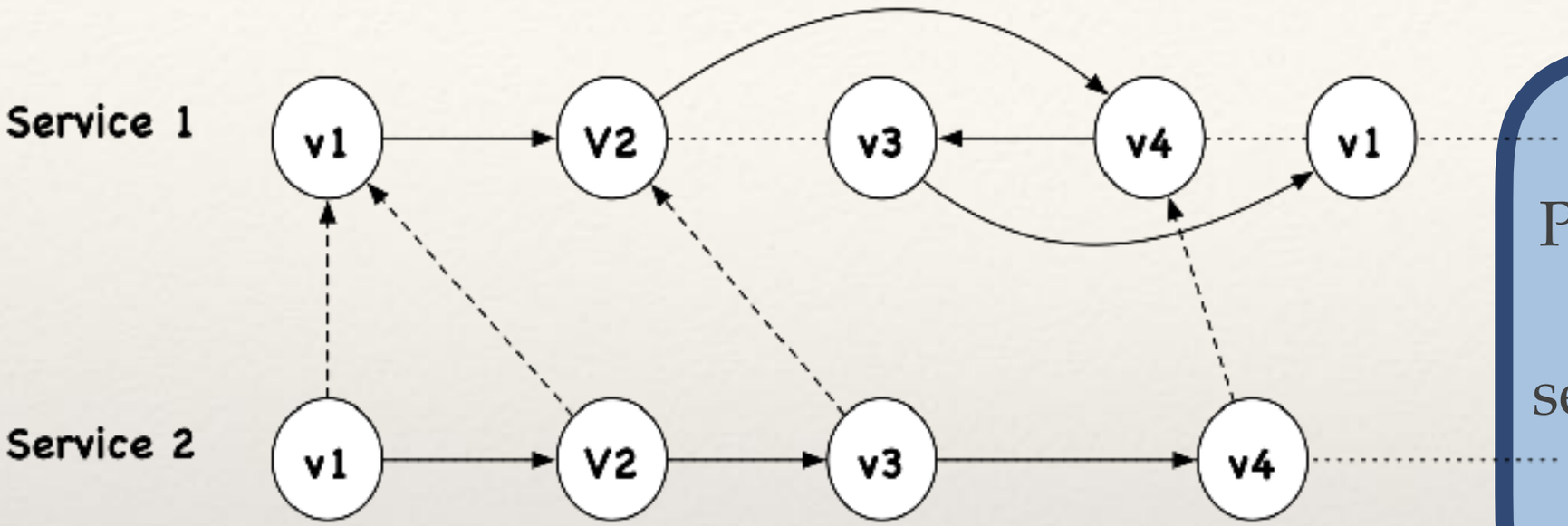
Use asynchronous messaging

Merge two Microservices

Use two and deal with Consistency

- Use distributed transactions
- Use compensation or lesser guarantees

Service Evolution



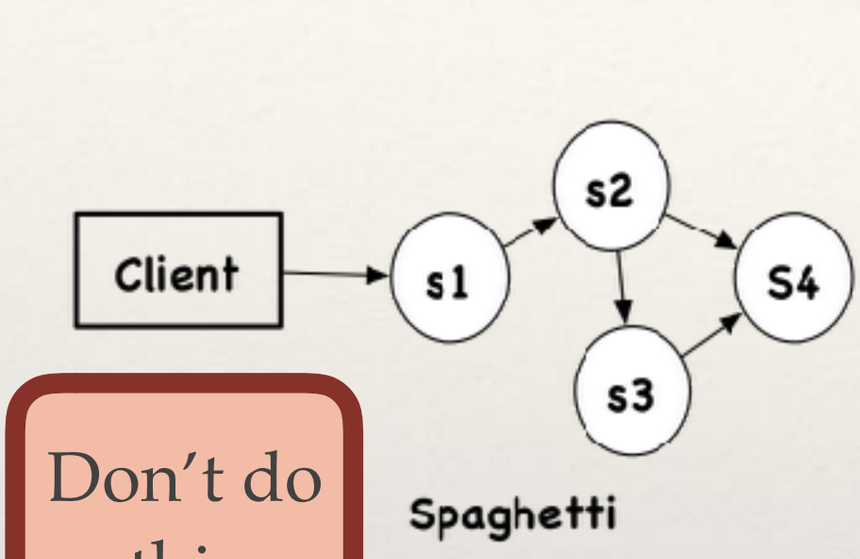
Point of microservices is new release of a service must not break other services

Services should be versioned

Backward and forward compatible (time bounded)

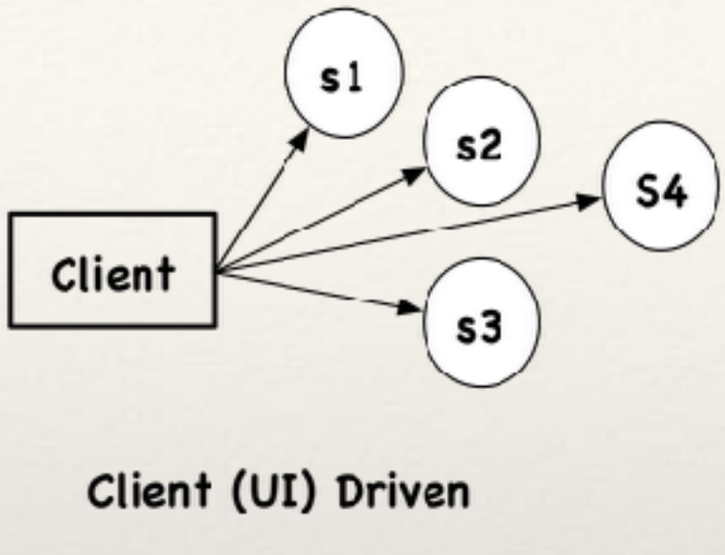
In most cases this is a matter of adding optional parameters and never renaming or removing existing parameters.

Avoid Dependency Hell



Spaghetti

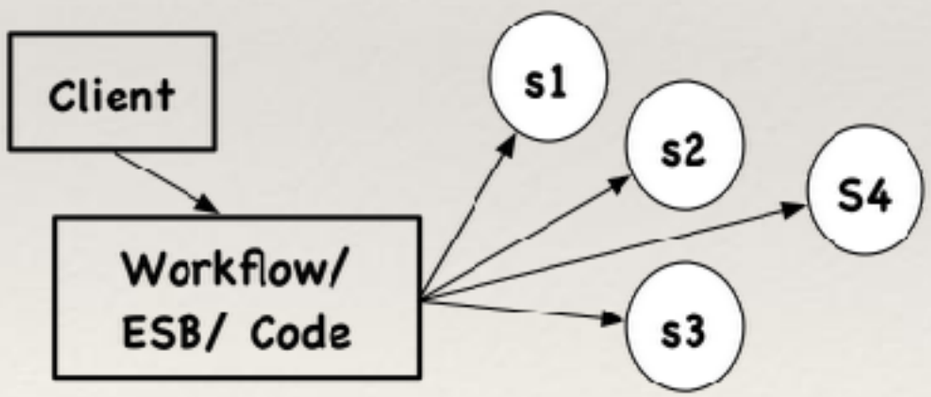
Don't do this



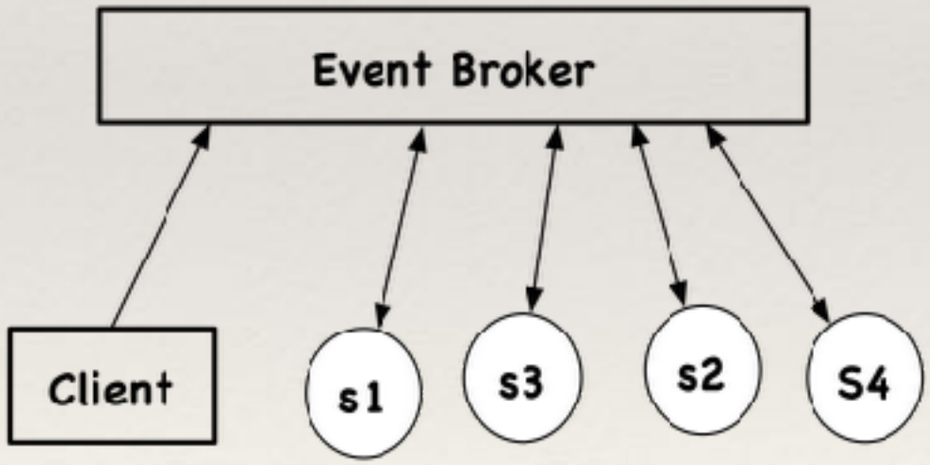
Client (UI) Driven

Not always possible

Perf and security concerns



Centerlized



Choreography

e.g. RXJava, node.js model

Hard to get it right

No ESB allowed??

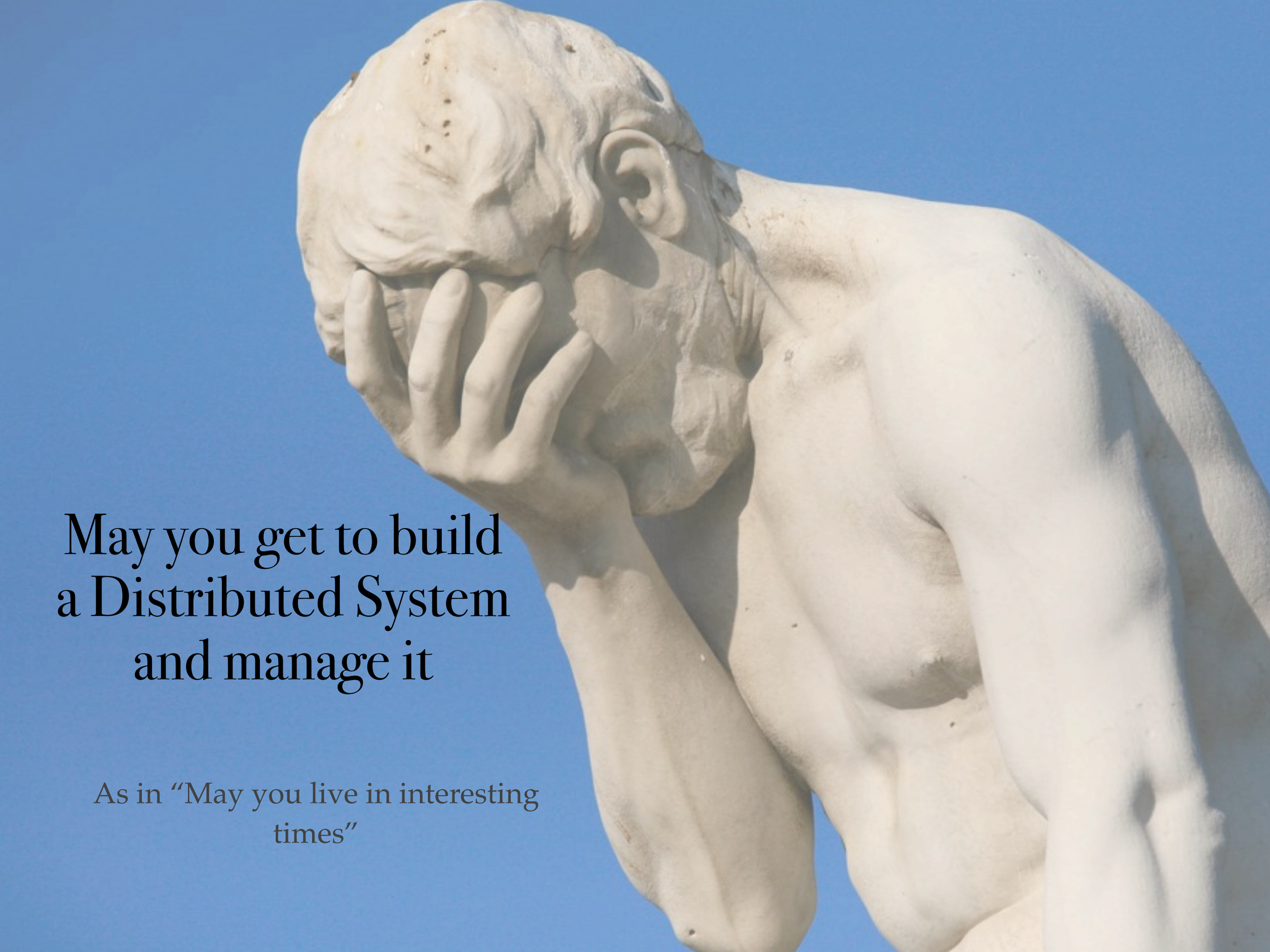
"A rallying cry" in Microservices movement

Not sure what is the solution

Client driven Model

Mico Integration

(One integration per server)



May you get to build a Distributed System and manage it

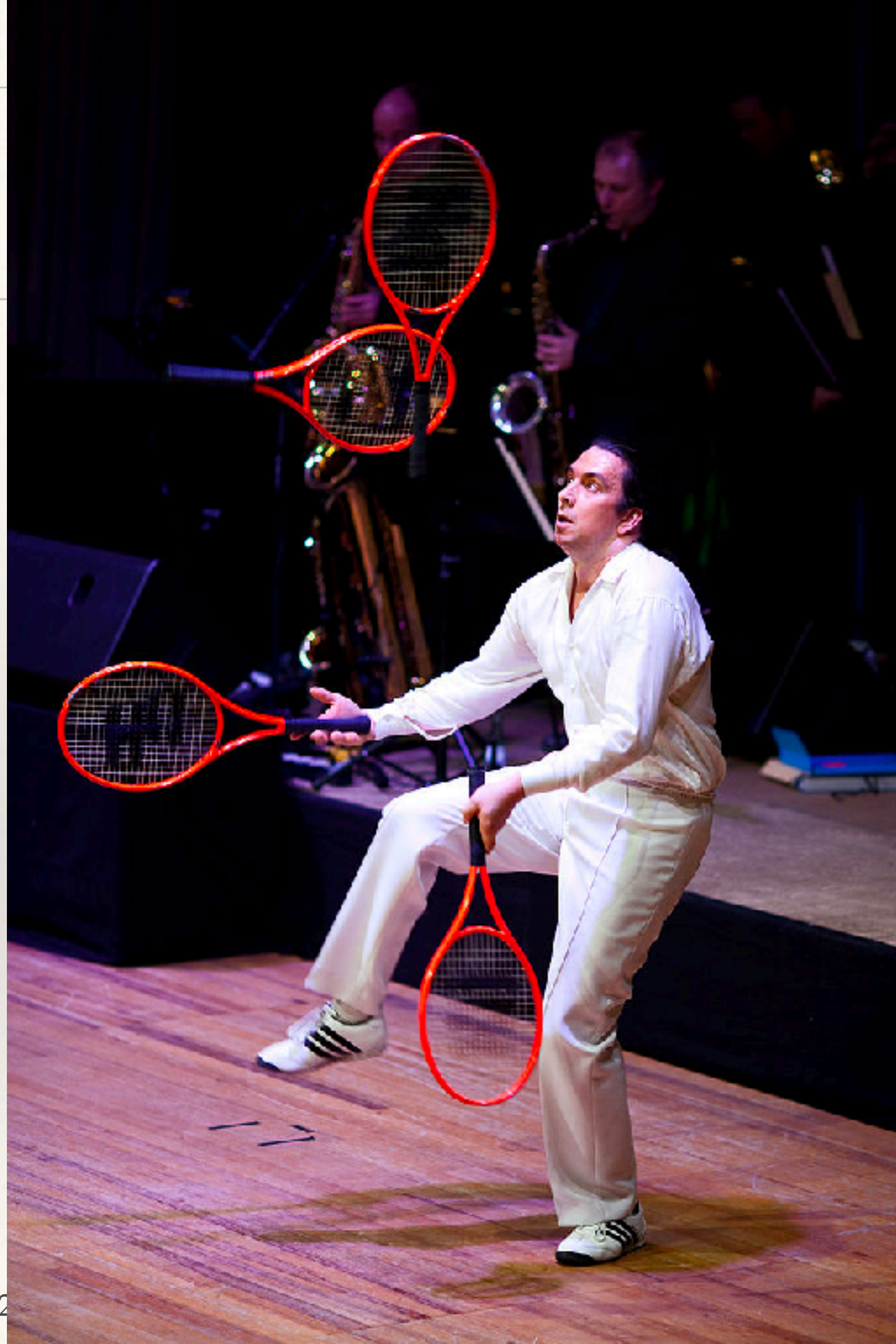
As in “May you live in interesting times”

Now you have truly built a distributed System

- You can only justify this if
  - Team > Two Pizza team
  - If you do > 1000s TPS
- If not may be you should go back to small web app you had not stop making your life hard

If yes, now you have new problems

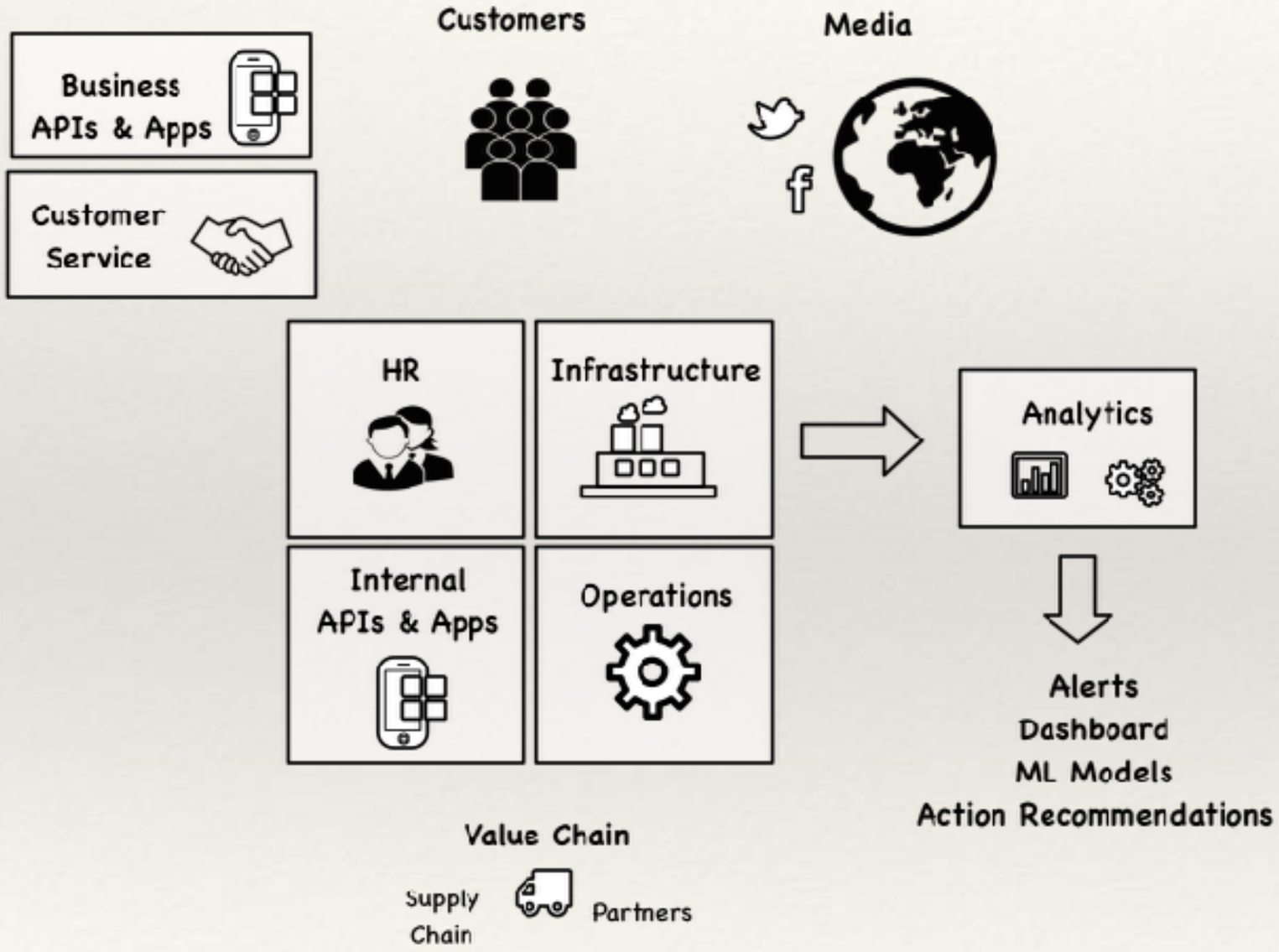
- Keep it running
- Find and debug problems



Tracing

- Need a distributed tracing infrastructure to find out when things go wrong
- One option is Elasticsearch, that let you collect logs, search them via text indexing, and visualize them
- Can use analytics infrastructure as we will discuss next

APIs are Great Data Collection Points



- In a architecture done around APIs, all key functionalities captured by APIs
- Data collected at these points can be used to understand business

Part 2: Can we rethink middleware for Integration?

We are entering the age of APIs

- Future Apps are built (mostly) with API composition
- We must make it easier to produce and consume networked services

Integration

XML / JSON based

(as that only format that work across systems)

Parallel

Program by people with limited programming knowledge

Inherently distributed

- Built with messages over the network
- Needs security

Glued with data manipulations and mapping

Integration is often done in a Second Class Way

Parallelism need to be explicitly handled

Concurrency is poorly handled

Need lot of plumbing for testing

Users need to think through the security model

Network communications need to be explicitly handled

XML / JSON conversions manually done

What is Wrong with DSLs?

Poor performance in non obvious scenarios

Type system is weak

Concurrency is poorly handled

Painful when actual complex logic is needed

Editors and tooling are not native

Debugging is painful

Ballerina

Textual and graphical parity on sequence diagram metaphor

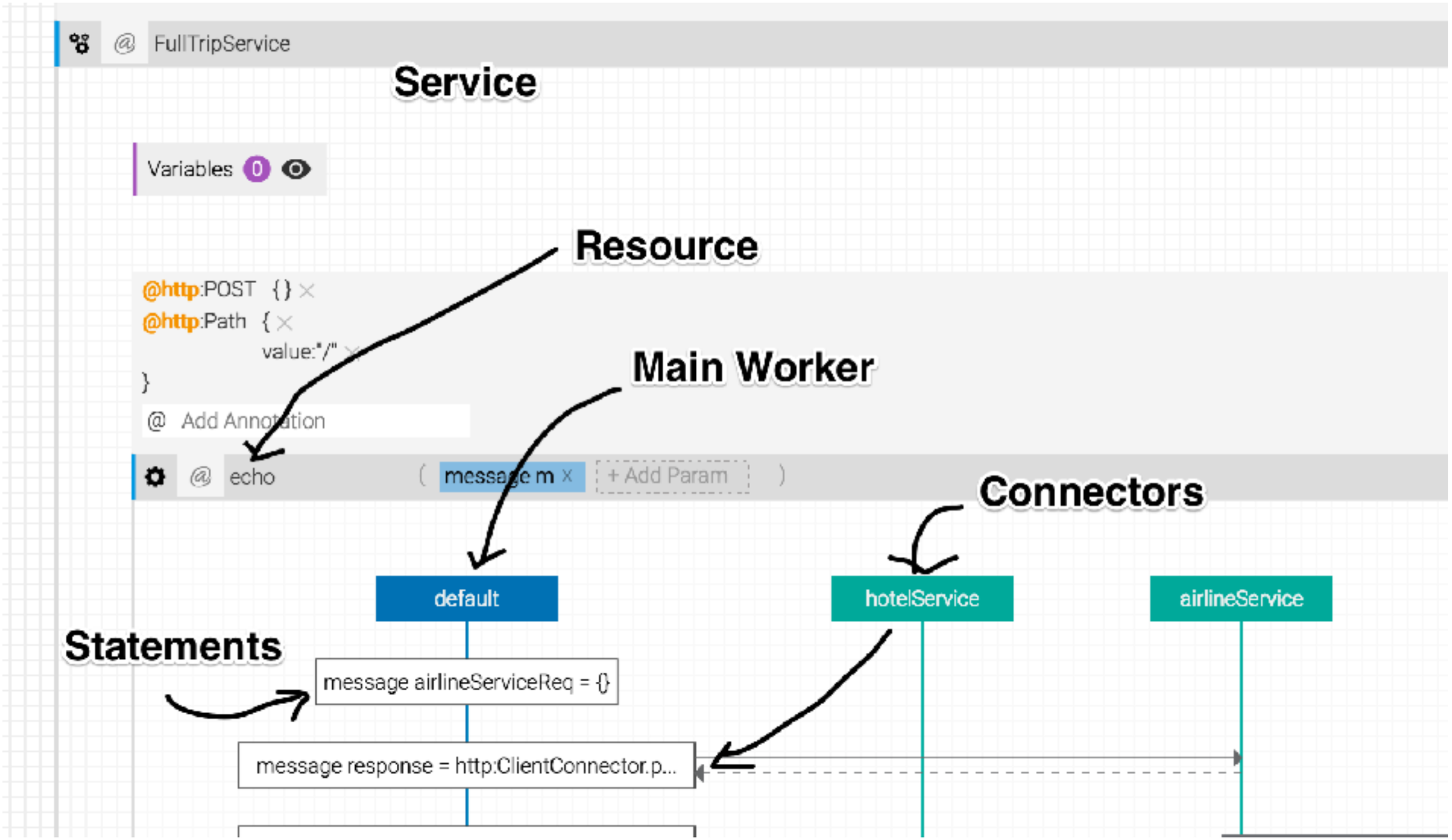
Strongly and statically typed with powerful type system

First class programming language with tools, debugging etc

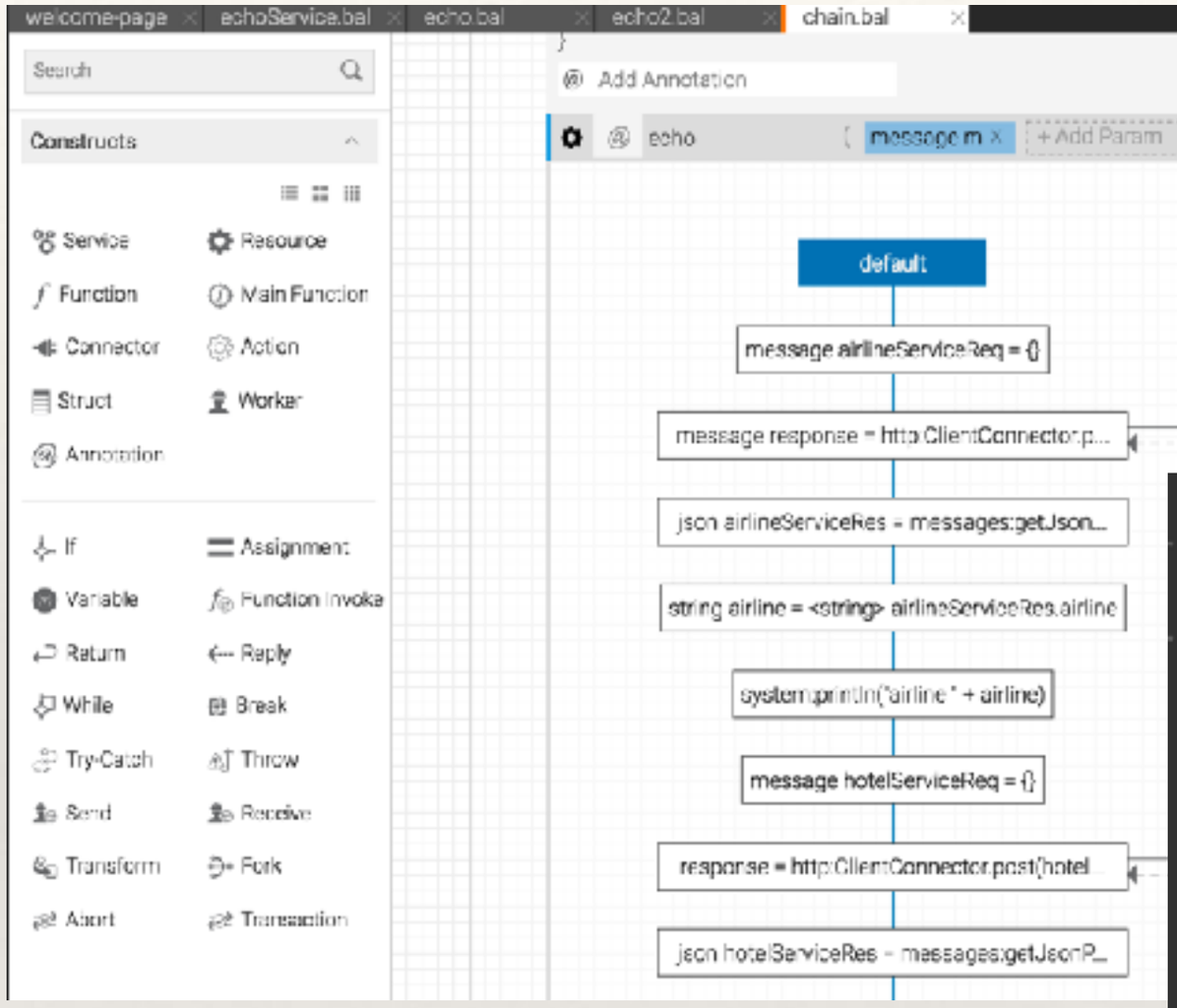
Designed for network with JSON, XML, SQL, MIME with HTTP, JMS

Natively parallel

Key Concepts



Visual and Textual Parity




```
@http:BasePath {value:"/fulltrip"}
service FullTripService {
    @http:POST{}
    @http:Path {value:"/"}
    resource echo (message m) {
        http:ClientConnector hotelService = create http:ClientConnector ("http://l
        http:ClientConnector airlineService = create http:ClientConnector ("http:/

        message airlineServiceReq = {};
        message response = http:ClientConnector.post(airlineService, "", airlineS
        json airlineServiceRes = messages:getJsonPayload(response);
        string airline = <string> airlineServiceRes.airline;
        system:println("airline " + airline);

        message hotelServiceReq = {};
        response = http:ClientConnector.post(hotelService, "", hotelServiceReq);
        json hotelServiceRes = messages:getJsonPayload(response);
        string hotel = <string> hotelServiceRes.hotel;
        system:println("hotel " + hotel);

        message fulltripRes = {};
        json tripData = {"airline":"","hotel":""};
        tripData.airline =airline;
        tripData.hotel =hotel;
        messages:setJsonPayload(fulltripRes,tripData);

        reply fulltripRes;
    }
}
```

Functions and Annotation

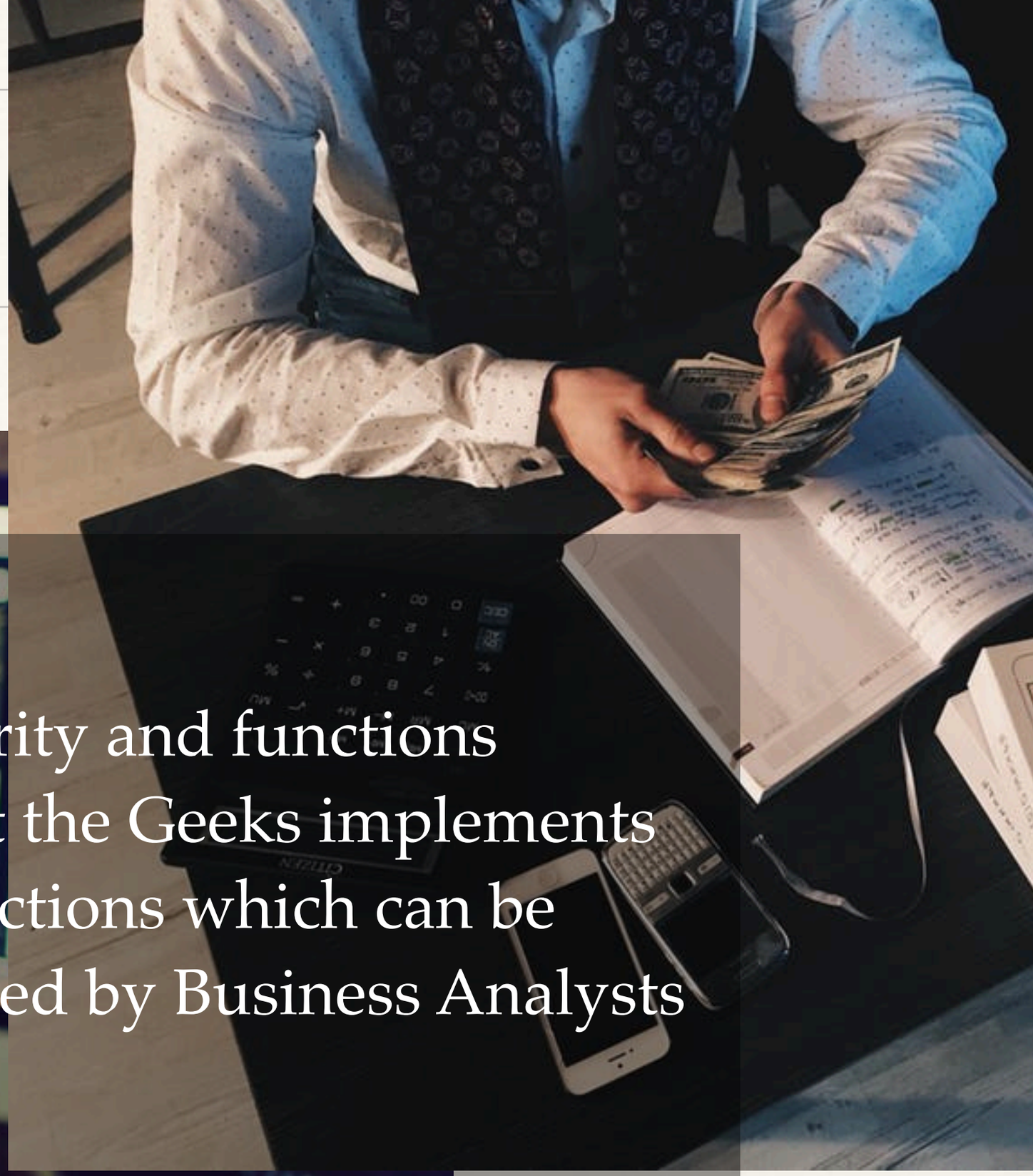
```
function TransformAddress (Address add)(Address) {
    ...
}
```

- Let you compose the program in term of decomposable units which is the key to scale the visual representation

```
@http:GET {}
@http:Path {value:"/"}
resource EchoResource (message m) {
    message m1 = {};
    messages:setStringPayload(m1,"hello2");
    reply m1;
}
```

- Annotations let you
  - Separate configurations from logic
  - Externalize configurations to files without code

Code vs. Biz



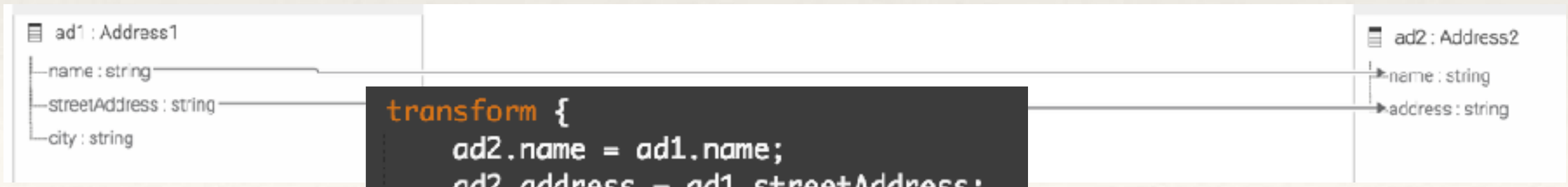
Visual Parity and functions
should let the Geeks implements
detail functions which can be
recomposed by Business Analysts

Ballerina knows XML and JSON

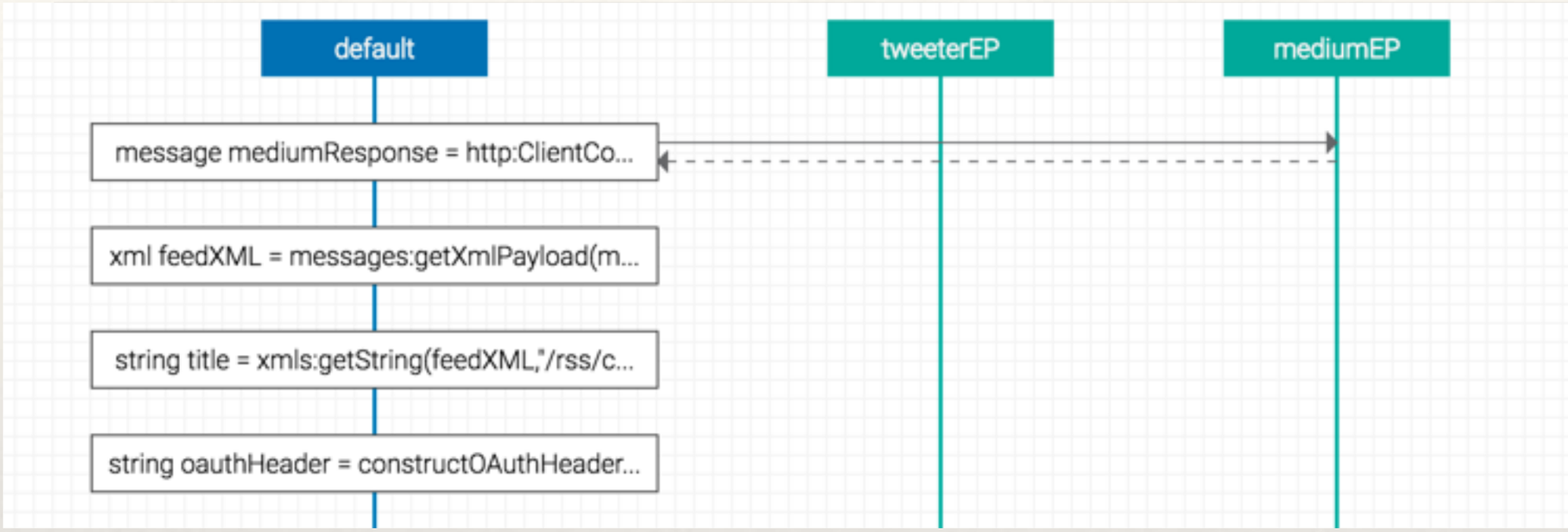
- Include XML and JSON as native types that can be casted to and from ballerina structs

```
message m1 = {};
json response = { "hotel":"Hilton Colombo" };
messages:setJsonPayload(m1,response);
reply m1;
```

- Language include native type mapping support, with type mapping drag and drop editor



Connectors



Support HTTP, Web Socket, JMS etc built into the language

```
http:ClientConnector mediumEP = create http:ClientConnector ("https://medium.com");
message mediumResponse = http:ClientConnector.get(mediumEP, "/feed/@wso2", {});
xml feedXML = messages:getXmlPayload(mediumResponse);
```

Call external APIs via Connectors

Security: BasicAuth, OAuth, AmazonAuth

Web APIs: Twitter, GMail, LinkedIn, Facebook, Lambda Functions

Tool: Swagger -> connector make recomposing ballerina services

receive events via Resources

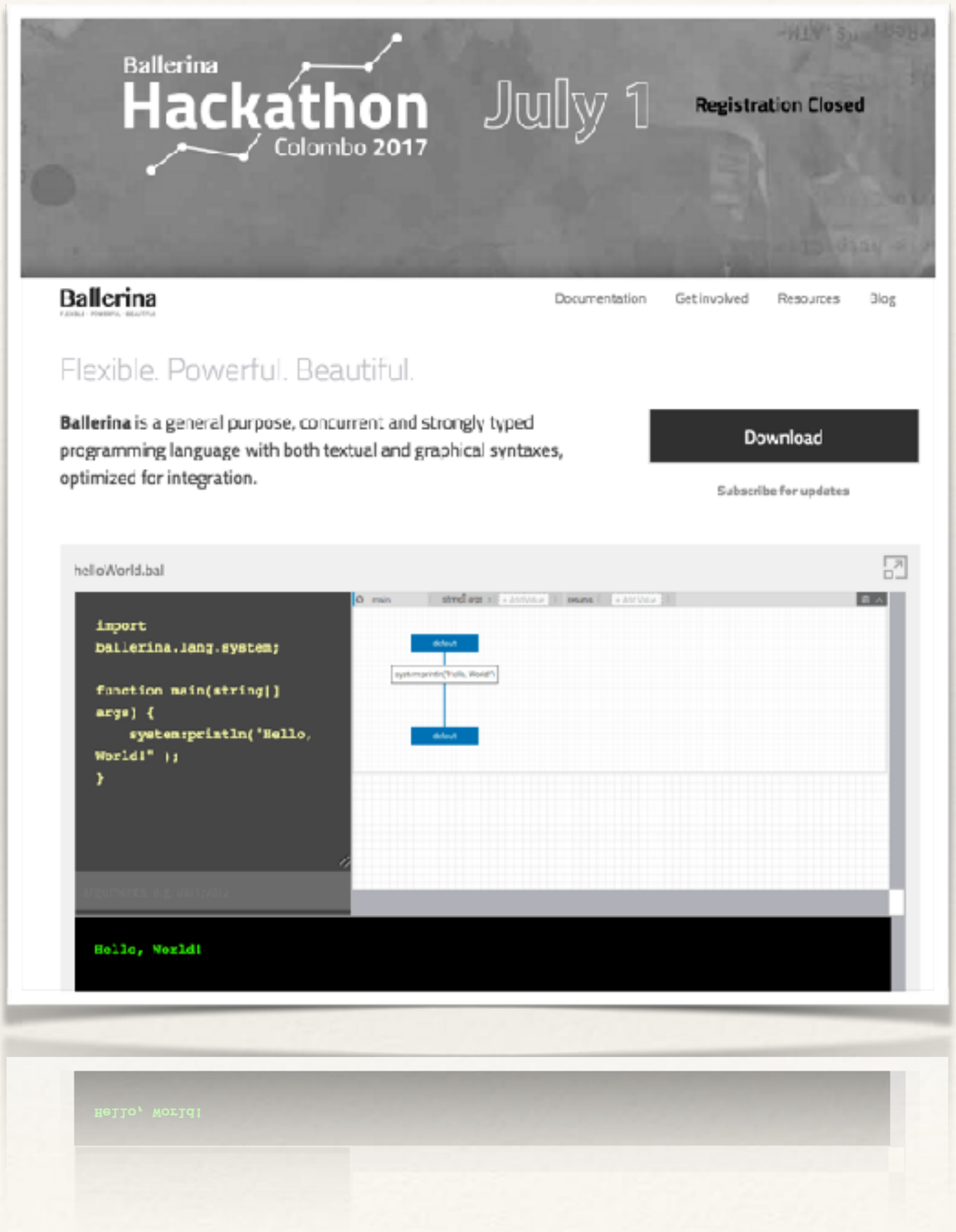
When to use Ballerina?

- Write micro (integration) services:80-20 rule
  - 80%. - work with other services
  - 20% - your logic
- Recompose existing services to an API backend (very similar to central ESB
- Integration scripts, scripts for Web API programming (main)

It is open source under Apache License

Find us through

- Users: ballerina-user@googlegroups.com
- Slack: #ballerinalang
- Twitter: @ballerinalang
- StackOverflow: #ballerinalang
- Developers: ballerina-dev@googlegroups.com



Questions?