

A Field Guide to your very own Guerrilla SOA Campaign

Dr. Jim Webber
Professional Services Director, ThoughtWorks

A decorative graphic in the bottom right corner consisting of several overlapping, wavy lines in shades of teal and green.

Fundamental Premise

There are two things money cannot buy:

1. *Love*
(Lennon/McCartney)
2. *An SOA*
(Webber)

Roadmap

- Enterprise Application Integration Approaches
- Enterprise Architecture, now and future
- The Appealing Rationale for ESB...
- Enterprise Architecture
- SOA and the Web
- What this means for you
- Conclusions
- Q&A

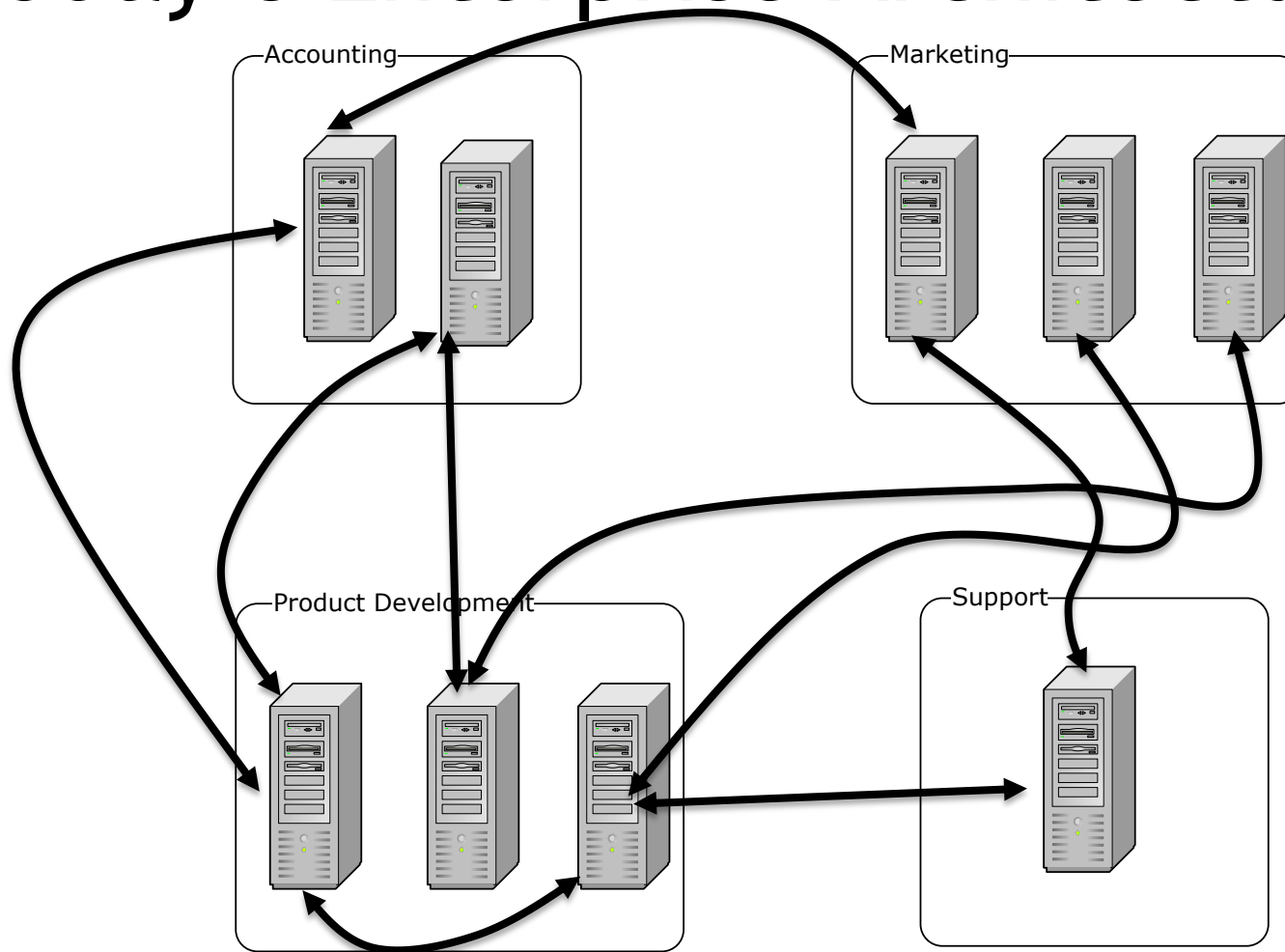
Integration Approaches

- Data integration
 - Extract, transform, route, inject data
- Application level
 - Re-use application APIs, or I/O mechanisms
- EAI implementation
 - Queues etc
- Business domain tier
 - Integration at the object level, as typified by CORBA, DCOM etc
- User interface
 - Screen scraping, revamping, etc.
 - Last resort, when an application offers no other hooks
- Web Services
 - Our first foray into protocol-centric integration!

To ESB or not to ESB, that is the question

- Product vendors are keen to provide product solution for everything
 - Or to supply “consultantware” solutions
- The Enterprise Service Bus is the latest incarnation of EAI technology that supports a number of useful functions:
 - Transformations; adapters; choreography; reliability; security etc
- Seems like a good idea...

Today's Enterprise Architecture

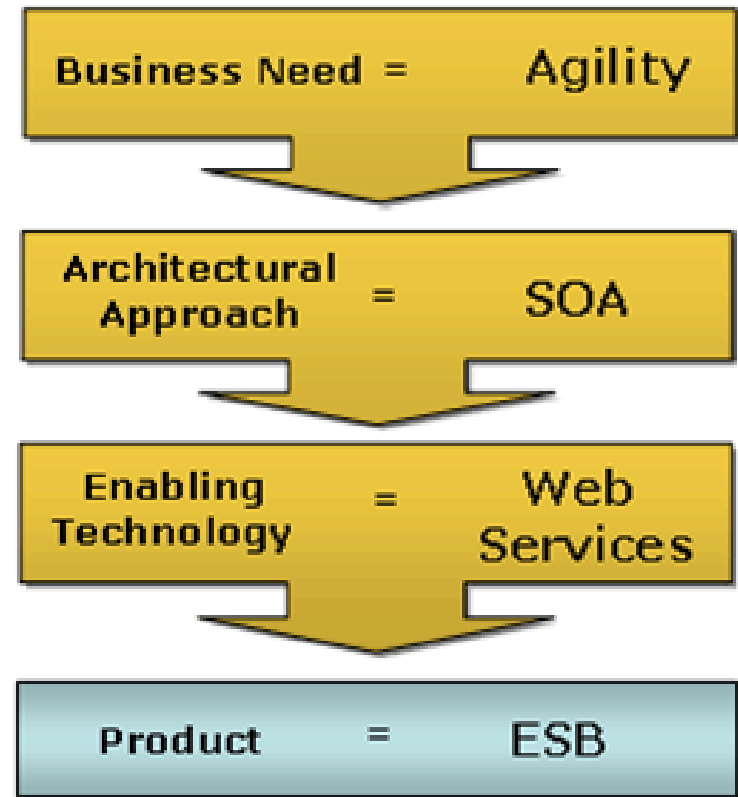


How did we get here?

- Tactical decisions
- Time and technology pressures
- Path of least resistance for individual applications
- This is the thin end of the wedge, technical debt can only increase from here
- Help!

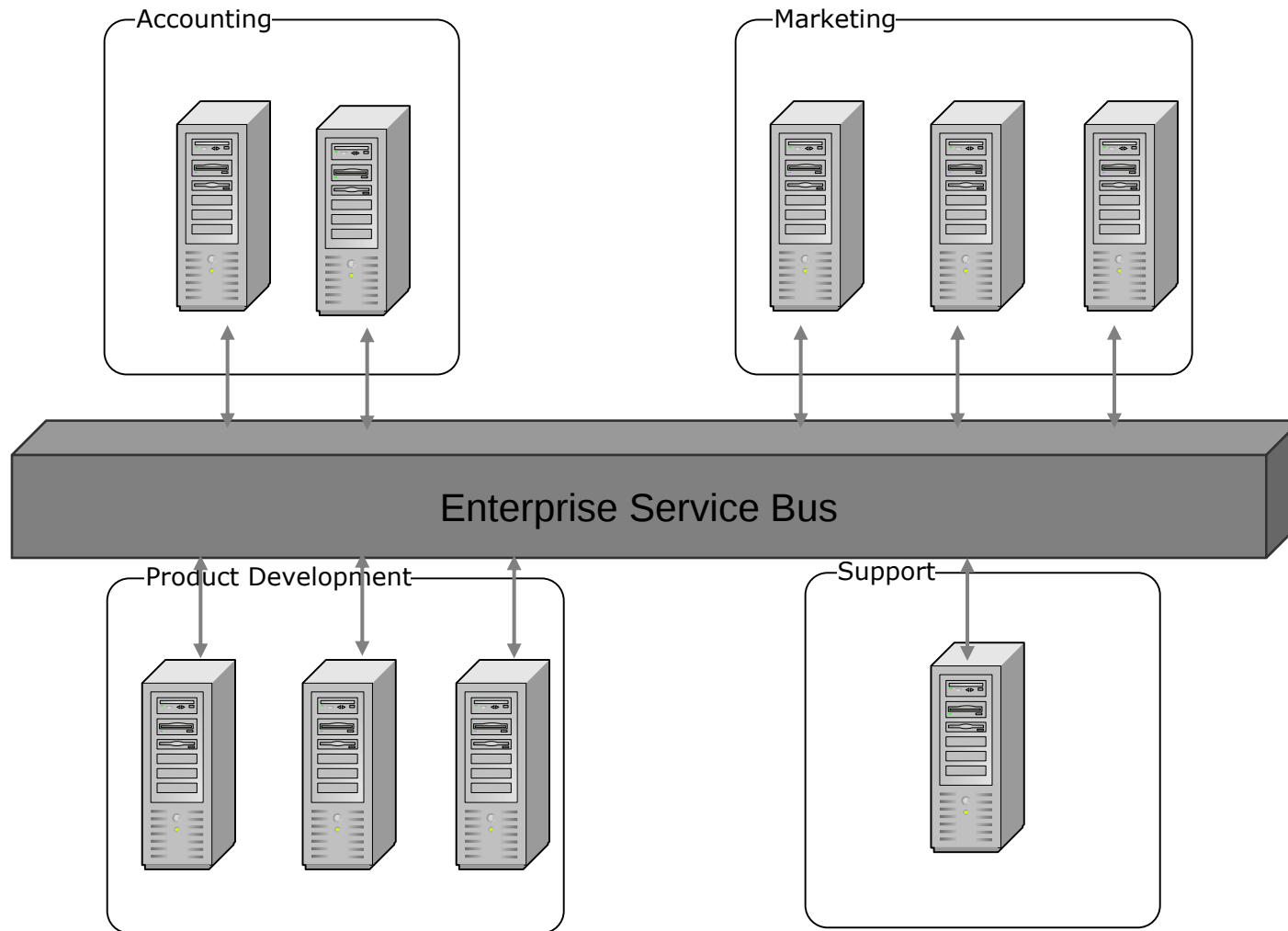
Vendor Solutions Appear

- Business needs to compete
 - IT needs to be responsive
- SOA gives IT a business process focus
- Pick a technology, and...
- More proprietary middleware is the answer!
 - $2 + 2 = 5$

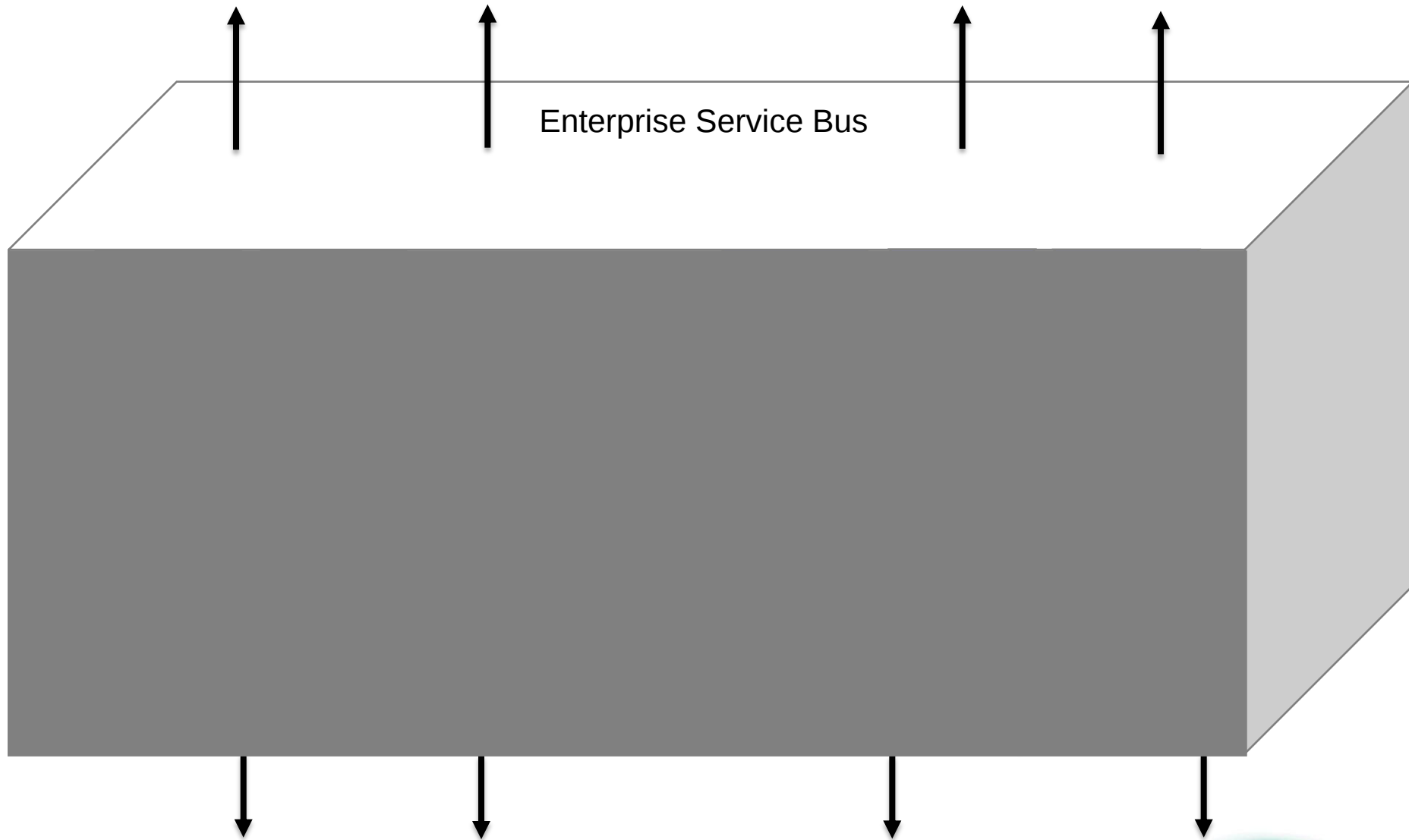


<http://www.capeclear.com/technology/index.shtml>

Integration Two Years Later



Skeletons in the Closet...



The Appealing Rationale for ESB...

- Perceived single framework for all integration needs
- Perceived simple connectivity between systems
- Some features for security, reliable delivery, etc.
- All you have to do is agree to lock yourself into a ESB and all this can be yours...

...And the Reality

- The mess is swept under the carpet hidden inside a vendor box
 - Mixing business rules, transformations, QoS etc with connectors
- Vendor lock-in of the whole network!
 - ESBs are proprietary, so no guarantees that the messages transmitted across the bus are actually based on any open protocol
- Held to ransom by the ESB vendor!
 - Can only easily integrate systems for which the ESB vendor provides specific adaptors
 - Or invest your money into extending their product

Intelligent Networks, Dumb Idea?

- Isn't this precisely what we're trying to get away from?
- Integration should happen on the wire by default, not inside some server
- The ESB approach eschews the dumb network
 - Smart endpoints underpin scalable, robust systems
 - Smart networks are failure points

More plumbing gets built



SOA “experts” grow powerful



And ESB software grows...



... the wrong way



On a rich diet



BPM



Reliability



Rules
Engine



Transformations



Security



GUI Tools

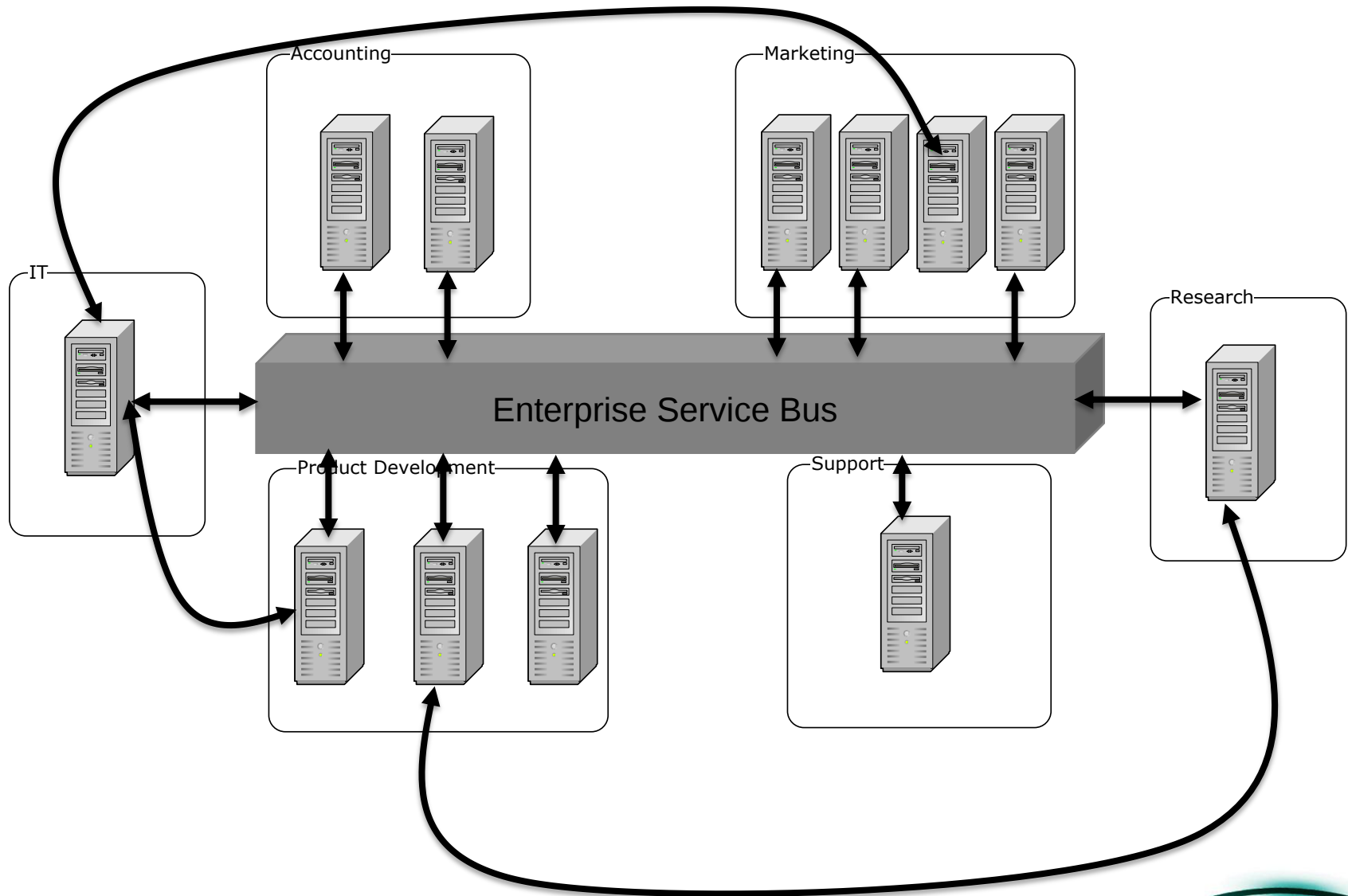


Low
Latency

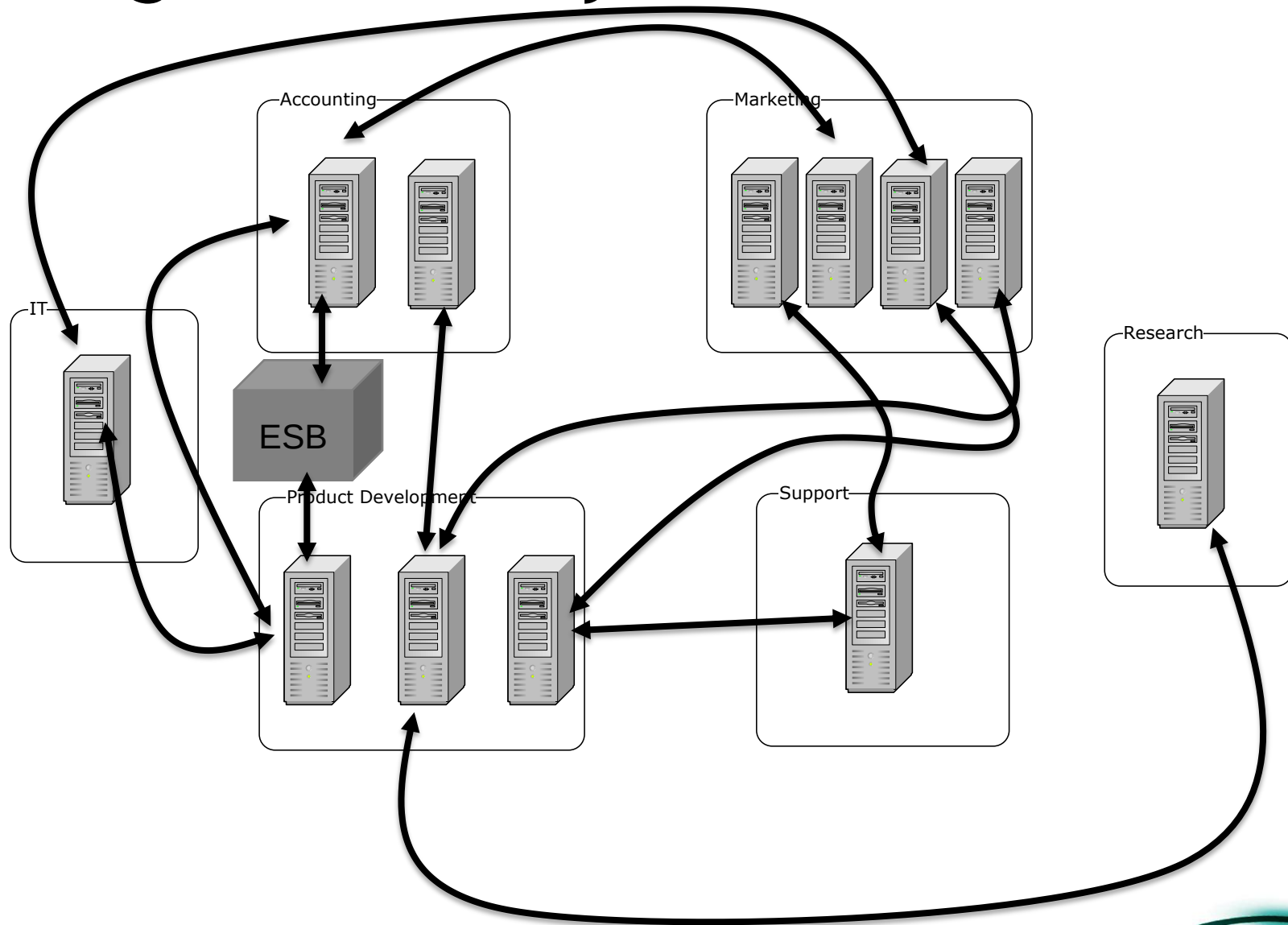
Adapters



Integration five years from now



Integration ten years from now



Architectural Fantasy



Ungovernable



Doesn't Scale



How did this happen?

- Same old story:
 - Tactical decisions
 - Time and technology pressures
 - Path of least resistance for individual applications
- Centralised ownership of the ESB sometimes is an inhibitor
 - Too much effort to get on the bus, technically, politically
 - Individuals always mean to redress hacked integrations
 - But seldom do - it's too hard when systems are live

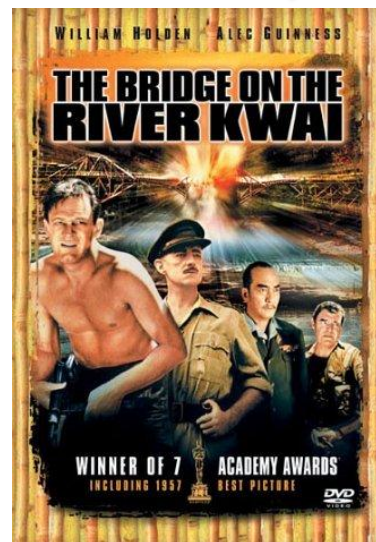
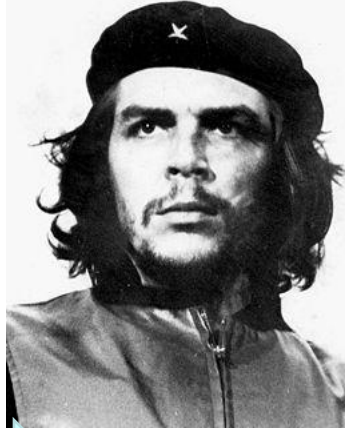
BDUF Trench Warfare



*"If nothing else works,
a total pig-headed
unwillingness
to look facts in the face
will see us through."*



Guerrilla SOA



Spaghetti is a fact of life

- Businesses change
- Processes change
- Applications change
- Integration changes
- Need an enterprise computing strategy that:
 - Reflects the changing structure of the business;
 - Is spaghetti-friendly;
 - Commoditised;
 - Robust, secure, dependable, etc.

Business-Led Integration

- ESBs integrate with whatever existing systems expose
 - Green screen, web pages, CORBA objects, XML, etc
- Integration happens at a low level
 - Mapping of bits and bytes of one variety onto bits and bytes of another format
- This makes it hard to engage business in such projects
 - Without business benefit no software has value
- Integration is currently opaque to the business
- Business must be involved in integration projects - not just initiate them
 - The integration domain must use the same vocabulary as the business domain

Spaghetti-Oriented Architecture

- Fighting against spaghetti is usually unsuccessful
 - This does not mean integration should be undertaken without diligence!
- SOA is an approach which is spaghetti-agnostic
- Services are designed for integration with any consumer
 - Integration is decentralised
- In Web-based SOA, we model key business entities as Web resources
- Result:
 - Loosely coupled, re-usable services
 - Focus on business-meaningful artifacts

Web Characteristics

- Scalable
 - Fault-tolerant
 - Recoverable
 - Secure
 - Loosely coupled
-
- Precisely the same characteristics we want in SOA!

Tenets for Web-based Services

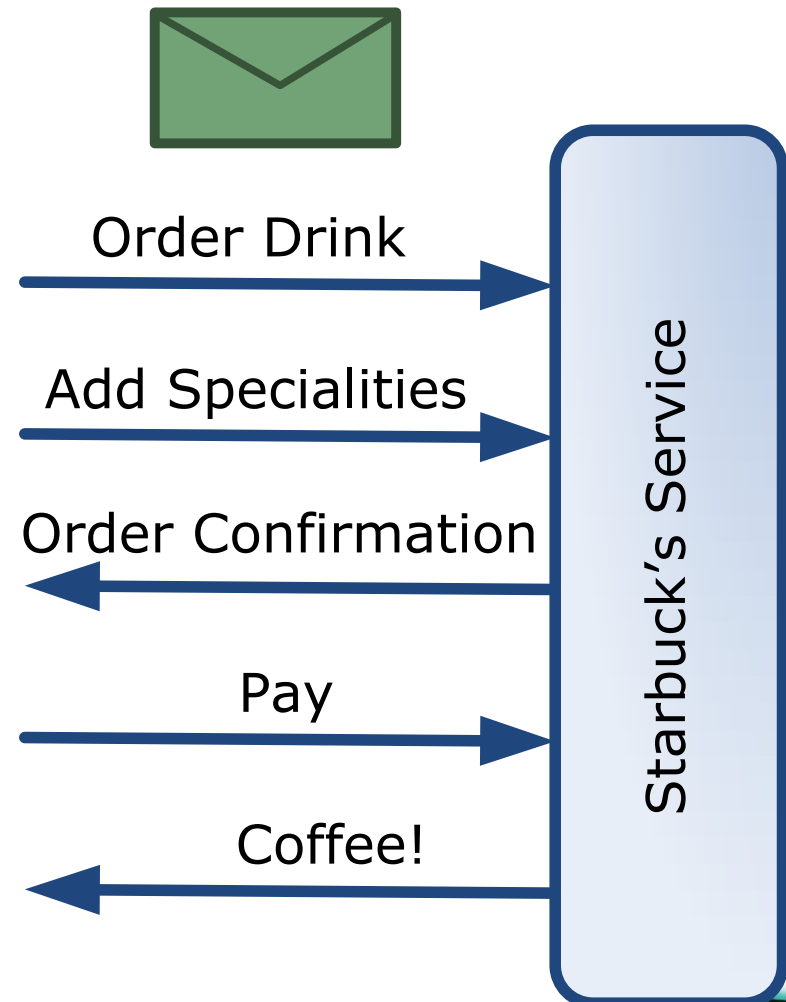
- Resource-based
 - Rather than service-oriented (the Web is not MOM!)
- Addressability
 - Interesting things should have names
- Statelessness
 - No stateful conversations with a resource
- Representations
 - Resources can be serialised into representations
- Links
 - Resources
- Uniform Interface
 - No plumbing surprises!

Workflow

- How does a typical enterprise workflow look when it's implemented in a Web-friendly way?
- Let's take Starbuck's as an example, the happy path is:
 - Make selection
 - Add any specialities
 - Pay
 - Wait for a while
 - Collect drink

Workflow and MOM

- With Web Services we exchange messages with the service
- Resource state is hidden from view
- Conversation state is all we know
 - Advertise it with SSDL, BPEL, WS-Chor
- Uniform interface, roles defined by SOAP
 - No “operations”

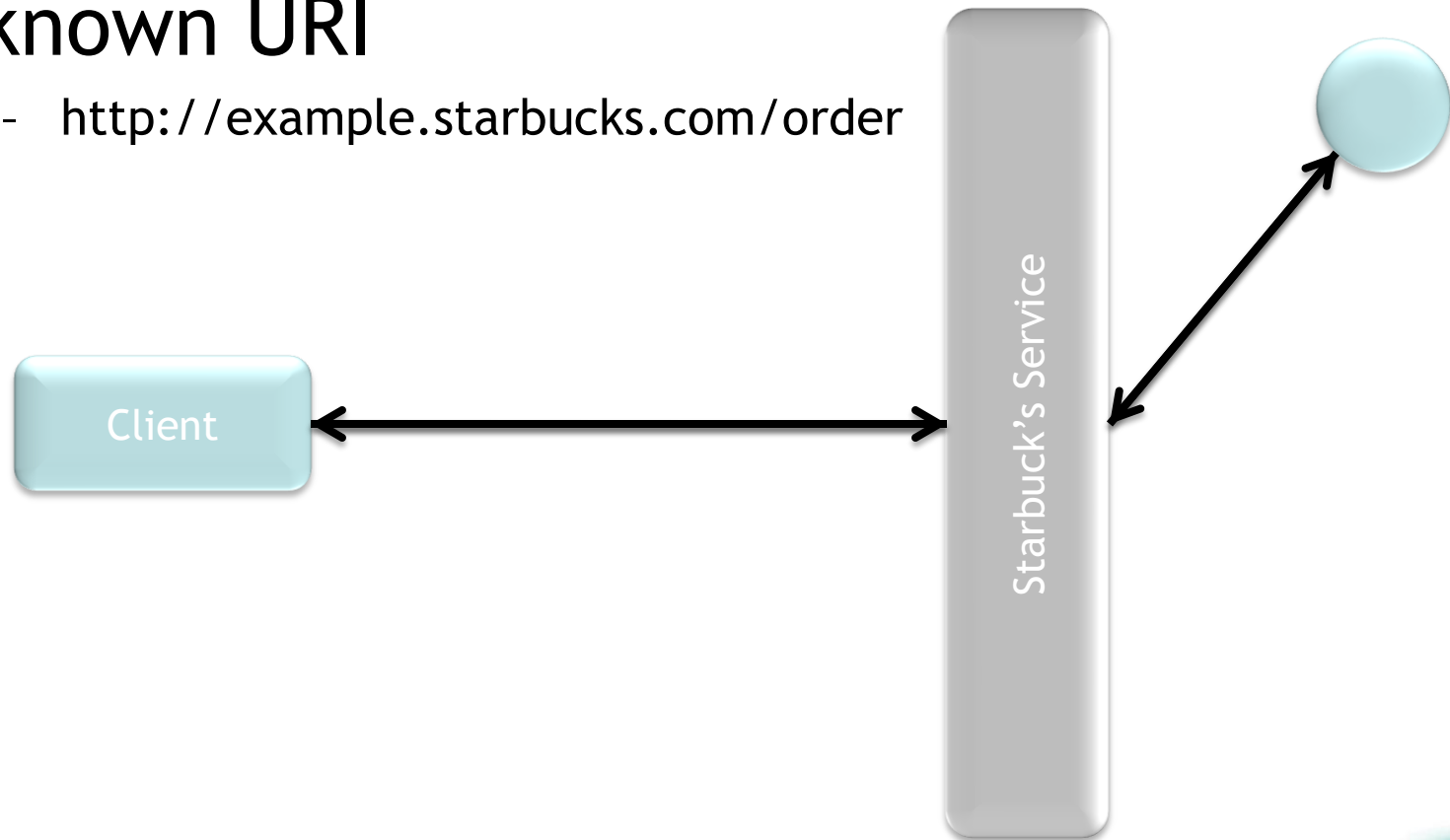


Web-friendly Workflow

- What happens if workflow stages are modelled as resources?
- And state transitions are modelled as hyperlinks or URI templates?
- And events modelled by traversing links and changing resource states?
- Answer: we get Web-friendly workflow
 - With all the quality of service provided by the Web

Placing an Order

- Place your order by POSTing it to a well-known URI
 - `http://example.starbucks.com/order`



Placing an Order: On the Wire

- Request

```
POST /order HTTP 1.1
Host: starbucks.example.org
Content-Type: application/xml
Content-Length: ...
```

```
<order xmlns="urn:starbucks">
<drink>latte</drink>
</order>
```

If we have a (private) microformat, this can become a neat API!

- Response

```
201 Created
Location:
    http://starbucks.example.org/
    order?1234
Content-Type: application/xml
Content-Length: ...
```

```
<order xmlns="urn:starbucks">
<drink>latte</drink>
<link rel="payment"
      href="https://starbucks.example.org/payment/order?1234"
      type="application/xml"/>
</order>
```

Whoops! A mistake

- I like my coffee to taste like coffee!
- I need another shot of espresso
 - What are my OPTIONS?

Request

```
OPTIONS /order?1234 HTTP 1.1
```

```
Host: starbucks.example.org
```

Response

```
200 OK
```

```
Allow: GET, PUT
```

Phew! I can
update my order,
for now

Optional: Look Before You Leap

- See if the resource has changed since you submitted your order
 - If you're fast your drink hasn't been prepared yet

◉ Request

```
PUT /order?1234 HTTP 1.1  
Host: starbucks.example.org  
Expect: 100-Continue
```

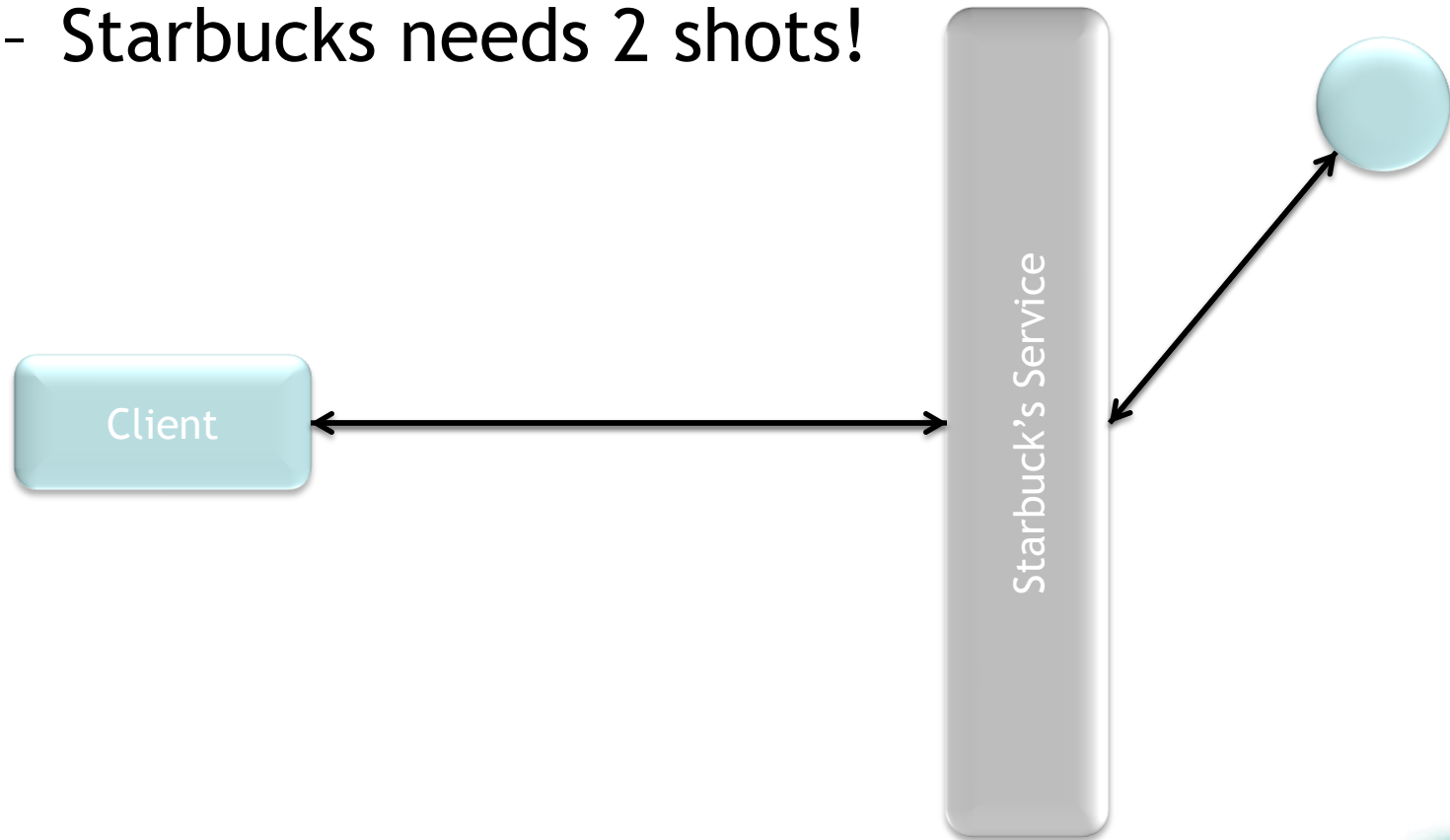
◉ Response

```
100 Continue
```

I can still PUT this resource, for now.
(417 Expectation Failed otherwise)

Amending an Order

- Add specialities to you order via PUT
 - Starbucks needs 2 shots!



Amending an Order: On the Wire

- Request

```
PUT /order?1234 HTTP 1.1
Host: starbucks.example.org
Content-Type: application/xml
Content-Length: ...

<order xmlns="urn:starbucks">
  <drink>latte</drink>
  <additions>shot</additions>
  <link rel="payment"
    href="https://starbucks.example.org/payment/order?1234"
    type="application/xml"/>
</order>
```

- Response

```
200 OK
Location:
  http://starbucks.example.org/order?1234
Content-Type: application/xml
Content-Length: ...

<order xmlns="urn:starbucks">
  <drink>latte</drink>
  <additions>shot</additions>
  <link rel="payment"
    href="https://starbucks.example.org/payment/order?1234"
    type="application/xml"/>
</order>
```

Statelessness

- Remember interactions with resources are stateless
- The resource “forgets” about you while you’re not directly interacting with it
- Which means race conditions are possible
- Use `If-Unmodified-Since` on a timestamp to make sure
 - Or use `If-Match` and an ETag
- You’ll get a `412 PreconditionFailed` if you lost the race
 - But you’ll avoid potentially putting the resource into some inconsistent state

Warning: Don't be Slow!

- Can only make changes until someone actually makes your drink
 - You're safe if you use `If-Unmodified-Since` or `If-Match`
 - But resource state can change without you!

Request

```
PUT /order?1234 HTTP 1.1
Host: starbucks.example.org
...
```

Request

```
OPTIONS /order?1234 HTTP 1.1
Host: starbucks.example.org
```

Response

409 Conflict

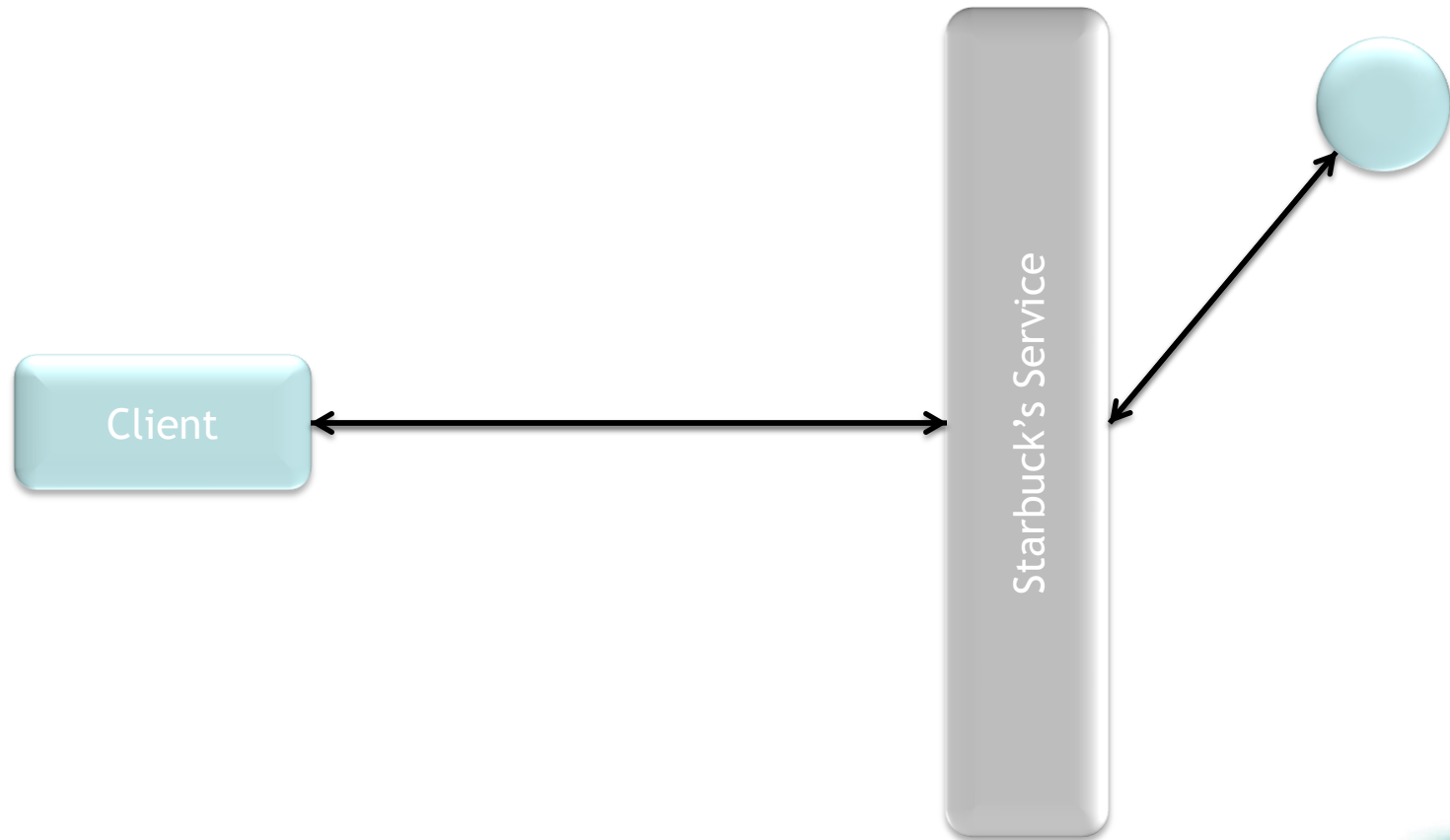
Too slow! Someone else has changed the state of my order

Response

Allow: GET

Order Confirmation

- Check your order status by GETing it



Order Confirmation: On the Wire

- Request

```
GET /order?1234 HTTP 1.1
Host: starbucks.example.org
Content-Type: application/xml
Content-Length: ...
```

- Response

```
200 OK
Location:
    http://starbucks.example.org/order?1234
Content-Type: application/xml
Content-Length: ...
```

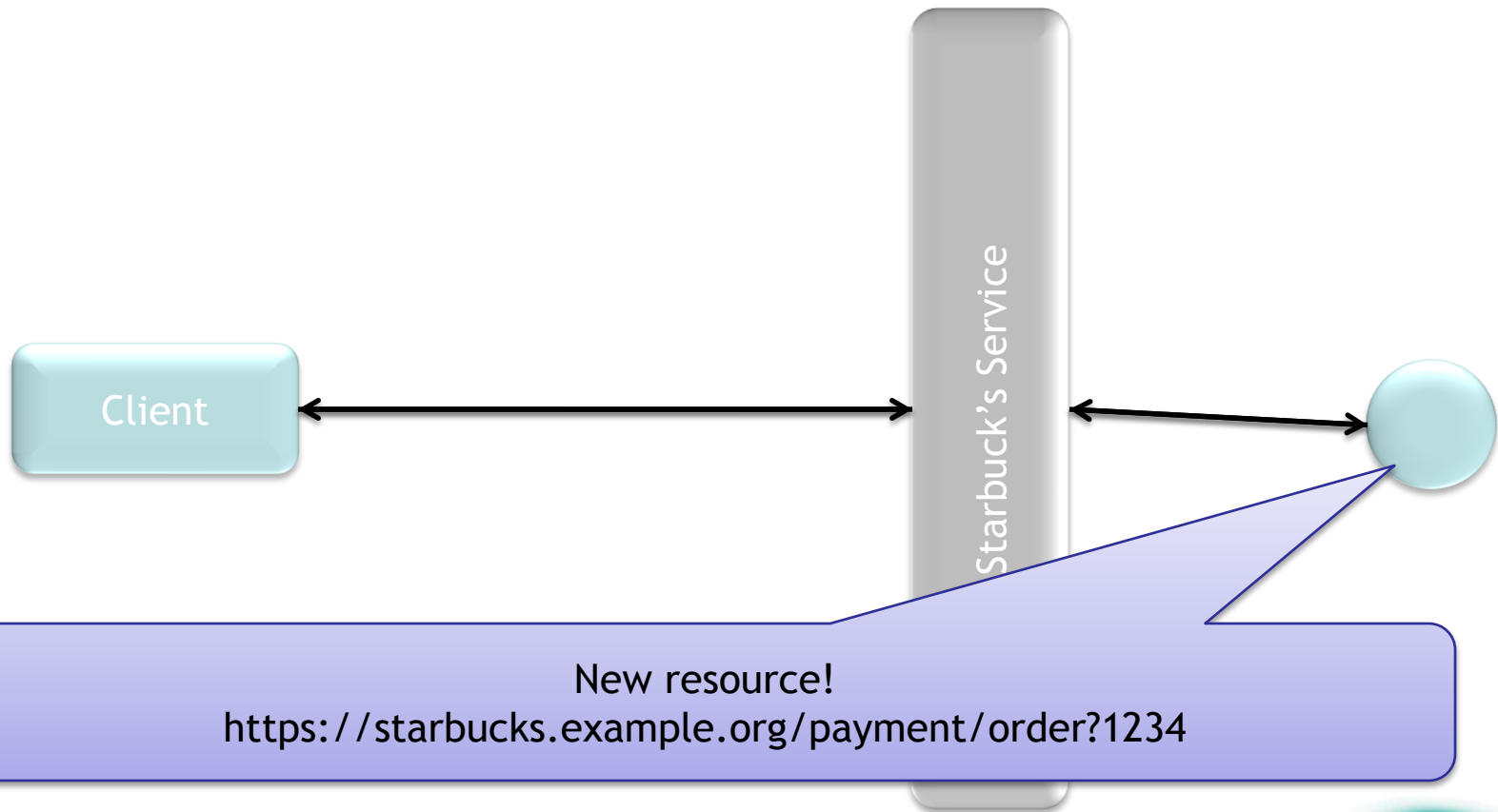
```
<order xmlns="urn:starbucks">
<drink>latte</drink>
<additions>shot</additions>
<link rel="payment"
    href="https://starbucks.example.org/payment/order?1234"
    type="application/xml"/>
</order>
```

Are they trying to tell me something with hypermedia?

Order Payment

- PUT your payment to the order resource

`https://starbucks.example.org/payment/order?1234`



How did I know to PUT?

- The client knew the URI to PUT to from the link
 - PUT is also idempotent (can safely re-try) in case of failure
- Verified with OPTIONS
 - Just in case you were in any doubt 😊

◉ Request

`OPTIONS /payment/order?1234 HTTP 1.1`

`Host: starbucks.example.org`

◉ Response

`Allow: GET, PUT`

Order Payment: On the Wire

- Request

```
PUT /payment/order?1234 HTTP 1.1
Host: starbucks.example.org
Content-Type: application/xml
Content-Length: ...
```

```
<payment xmlns="urn:starbucks">
<cardNo>123456789</cardNo>
<expires>07/07</expires>
<name>John Citizen</name>
<amount>4.00</amount>
</payment>
```

- Response

```
201 Created
Location:
    https://starbucks.example.org/payment/order?1234
Content-Type: application/xml
Content-Length: ...
```

```
<payment xmlns="urn:starbucks">
<cardNo>123456789</cardNo>
<expires>07/07</expires>
<name>John Citizen</name>
<amount>4.00</amount>
</payment>
```

Check that you've paid

- Request

```
GET /order?1234 HTTP 1.1
Host: starbucks.example.org
Content-Type: application/xml
Content-Length: ...
```

My "API" has changed,
because I've paid enough
now

- Response

```
200 OK
Content-Type: application/xml
Content-Length: ...
```

```
<order xmlns="urn:starbucks">
  <drink>latte</drink>
  <additions>shot</additions>
</order>
```

What Happened Behind the Scenes?

- Starbucks can use the same resources!
- Plus some private resources of their own
 - Master list of coffees to be prepared
- Authenticate to provide security on some resources
 - E.g. only Starbucks are allowed to view payments

Payment

- Only Starbucks systems can access the record of payments
 - Using the URI template: `http://.../payment/order/{order_id}`
- We can use HTTP authorisation to enforce this

Request

```
GET /payment/order?1234 HTTP 1.1
Host: starbucks.example.org
```

Request

```
GET /payment/order?1234 HTTP 1.1
Host: starbucks.example.org
Authorization: Digest username="jw"
realm="starbucks.example.org"
nonce="..."
uri="payment/order?1234"
qop=auth
nc=00000001
cnonce="..."
reponse="..."
opaque="..."
```

Response

```
401 Unauthorized
WWW-Authenticate: Digest
realm="starbucks.example.org",
qop="auth", nonce="ab656...",
opaque="b6a9..."
```

Response

```
200 OK
Content-Type: application/xml
Content-Length: ...

<payment xmlns="urn:starbucks">
  <cardNo>123456789</cardNo>
  <expires>07/07</expires>
  <name>John Citizen</name>
  <amount>4.00</amount>
</payment>
```

Master Coffee List

- `/orders` URI for all orders, only accepts GET
 - Anyone *can* use it, but it is only *useful* for Starbucks's
 - It's not identified in any of our public APIs anywhere, but the back-end systems know the URI

Request

GET `/orders` HTTP 1.1

Host: `starbucks.example.org`

Atom feed!

Response

```
200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0" ?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Coffees to make</title>
  <link rel="alternate"
    href="http://example.starbucks.com/order.atom"/>
  <updated>2007-07-10T09:18:43Z</updated>
  <author><name>Johnny Barrista</name></author>
  <id>urn:starkbucks:45ftis90</id>

  <entry>
    <link rel="alternate" type="application/xml"
      href="http://starbucks.example.org/order?1234"/>
    <id>urn:starbucks:a3tfpfz3</id>
  </entry>
  ...
</feed>
```

Finally drink your coffee...



Source: http://images.businessweek.com/ss/06/07/top_brands/image/starbucks.jpg

What did we learn from Starbuck's?

- HTTP has a header/status combination for every occasion, including failures
 - And well-defined semantics for crash recovery!
- APIs are expressed in terms of links, and links are great!
 - APP-esque APIs
- APIs can also be constructed with URI templates and inference
 - Trade off for tighter coupling
- XML is fine
 - Can also use formats like Atom, JSON or even XHTML as a middle ground
- State machines (defined by links) are important
 - Just as in Web Services...

But we still need middleware for non-functional requirements,
right?

WRONG!

In the bottom right corner, there are decorative wavy lines in shades of teal and green, curving upwards and to the right.

Scalability

- Stateless model
- Caching
 - Excellent for read-mostly applications
 - Allows the Web to trade latency for massive scalability
- Conditionals (Etag and friends)

Reliability

- Safe, idempotent behaviours for some verbs
 - GET, HEAD, OPTIONS
 - Not monadic though!
- Idempotent behaviours for some verbs
 - Just re-try in the event of failure
 - PUT, DELETE
- Lots of status codes and metadata to help in failure scenarios

Security

- Don't underestimate HTTPs!
- But longer term we have:
 - OpenID (or maybe not!)
 - SAML
 - OpenAuth
 - Etc

Transactions

- Not a good idea in large distributed systems anyway
 - Eventual consistency preferred
 - Be loose with your definition of durable
- HTTP is a coordination framework anyway
 - Status codes give you an idea of what to do in failure cases
 - More like workflow transactions than ACID transactions

Loose Coupling

- Our services share protocol only
 - No shared middleware
- Intermediaries are transparent
 - E.g. caches
- Degree of coupling becomes a design decision, rather than arising through accidental complexity

Same Old Architects

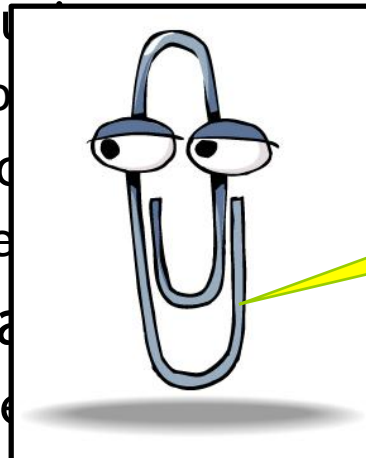
- Business and IT people collaborate around automating business processes and key business artifacts
- Service architects and developers build services
 - RESTlet, NetKernel, ASP.Net MVC, Rails, etc
 - Or even just the Servlet API!
- Enterprise architects spread best practices
 - and undertake necessary governance roles

ESB xor SOA?

- Investing in proprietary integration systems now is investing in future legacy
- ESB is not the solution
 - It's oh-so 1990's integration glue
- SOA is the solution
 - Because it focuses on supporting business processes
- The Web is robust platform for SOA

Conclusions

- SOA is the right integration architecture going forward
 - SOA should be implemented incrementally
 - Drive SOA from a business process
 - Most valuable process
 - Commoditisation and standardisation
 - Servers, development tools, existing software, etc
- Migrating towards a service-oriented architecture is not always easy
 - Learning to build distributed systems can be difficult
 - ESBs and Wizards cannot help - you need service-savvy geeks and process-aware business people
- No centralised integration middleware needed!



It looks like you're trying to build an SOA...

Quote of the Day

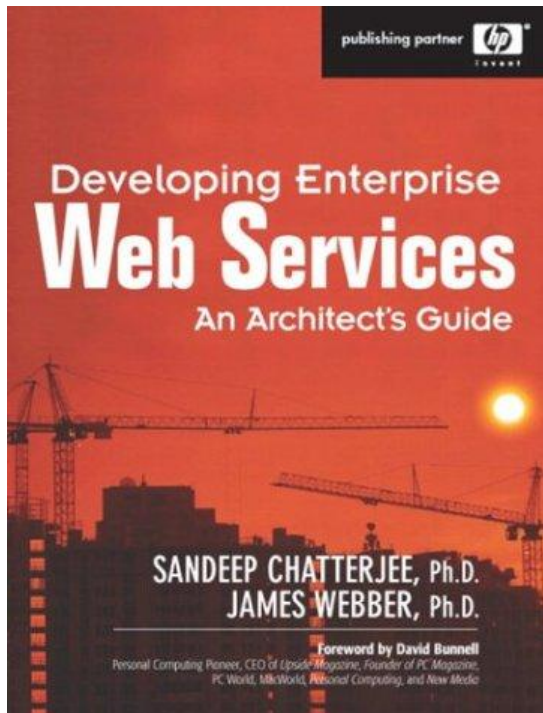
“...the idiots that are running around yelling "guerrilla SOA" have to be put in their place.”

Quoted on InfoQ:

<http://www.infoq.com/news/2007/11/soa-long>



Questions?



GET /Connected
(working title)

Jim Webber
Savas Parastatidis
Ian Robinson

Expected 2009

Blog:

<http://jim.webber.name>