Session INO-2739

Differentiating between web APIs, SOA, & integration …and why it matters

Kim Clark (kim.clark@uk.ibm.com)
Brian Petrini (petrini@us.ibm.com)

Impact2014

Be First.

April 27 – May 1 | The Venetian – Las Vegas, NV

#ibmimpact

Please Note

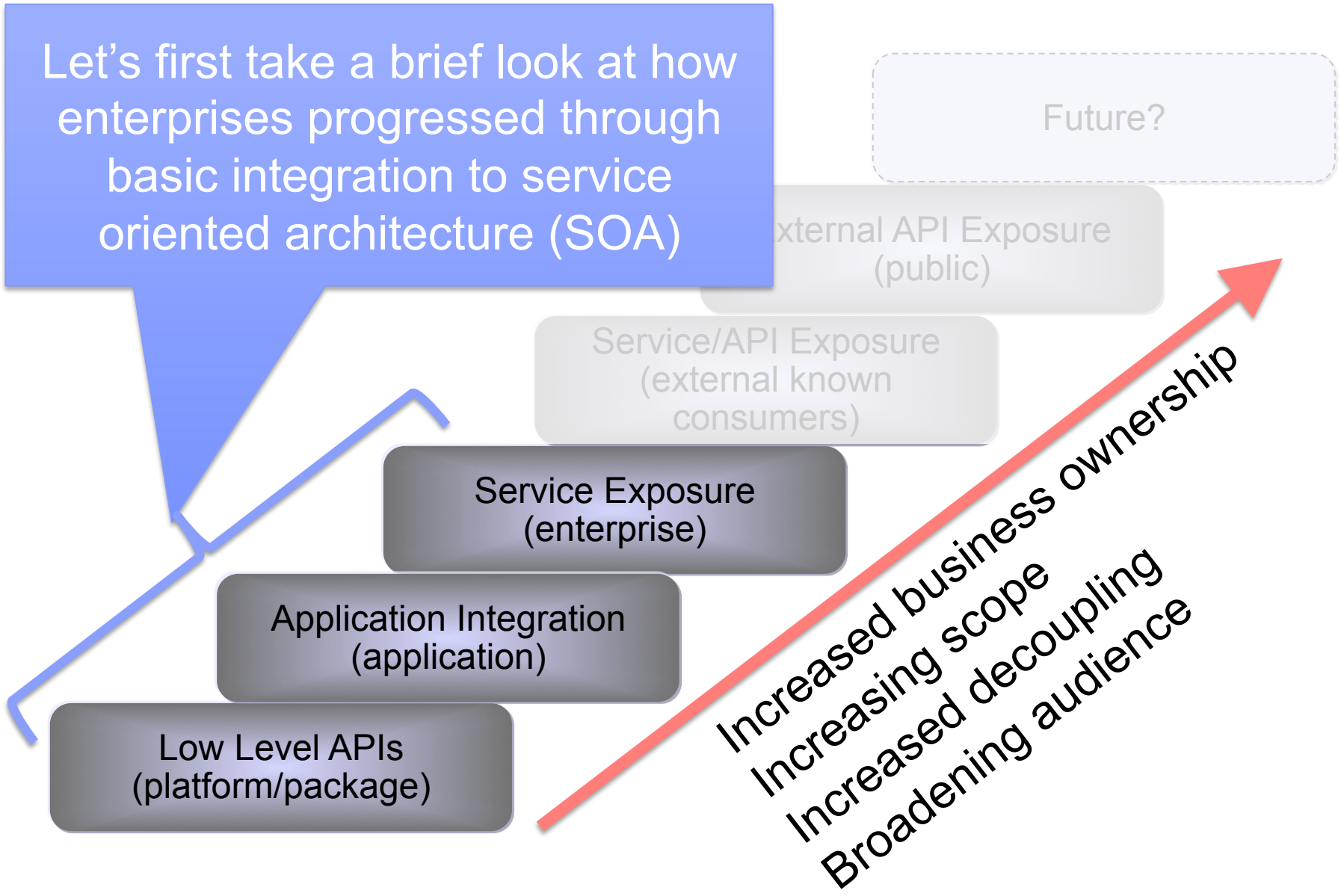
IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Evolving exposure of business function

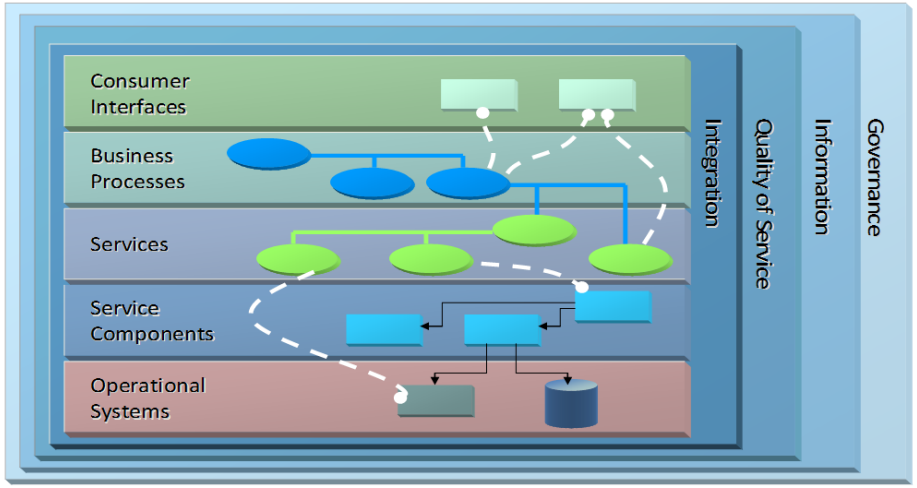


What was SOA originally

- Inwardly focused, based on understanding of core business capabilities.
- Often driven from business top down exercises such as Component Business Modelling.
- Focused on improving core business processes – organising/streamlining people/teams processes, audit tracking, status tracking, MI.

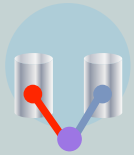
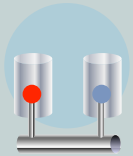
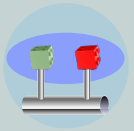
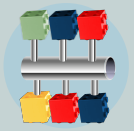
	Business Administration	New Business Development	Relationship Management	Servicing & Sales	Product Fulfillment	Financial Control and Accounting
directing	Business Planning	Sector Planning	Account Planning	Sales Planning	Fulfillment Planning	Portfolio Planning
controlling	Business Unit Tracking	Sector Management	Relationship Management	Sales Management	Fulfillment Planning	Compliance
	Staff Appraisals	Product Management	Credit Assessment			Reconciliation
executing	Staff Administration	Product Directory	Credit Administration	Sales	Product Fulfillment	Customer Accounts
		Marketing Campaigns		Customer Dialogue	Document Management	
	Production Administration			Contact Routing		General Ledger

Component Business Map



[SOA Reference Architecture](https://collaboration.opengroup.org/projects/soa-ref-arch)
https://collaboration.opengroup.org/projects/soa-ref-arch

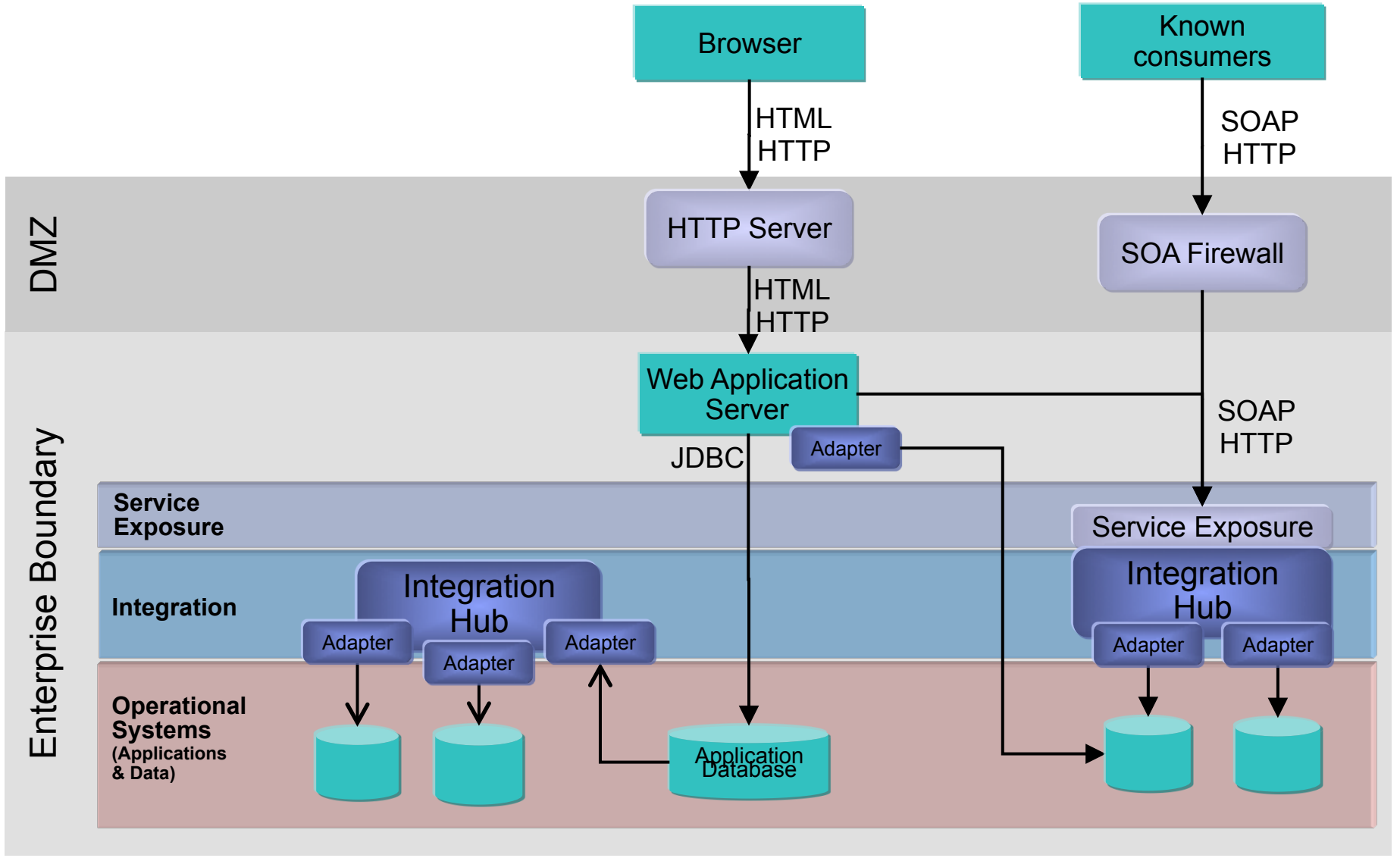
Open Service Integration Maturity Model (OSIMM)

	Silo	Integrated	Componentized	Services	Composite Services	Virtualized Services	Dynamically Re-Configurable Services
							
Business View	Function Oriented	Function Oriented	Function Oriented	Service Oriented	Service Oriented	Service Oriented	Service Oriented
Organization	Ad hoc IT Governance	Ad hoc IT Governance	Ad hoc IT Governance	Emerging SOA Governance	SOA and IT Governance Alignment	SOA and IT Governance Alignment	SOA and IT Governance Alignment
Methods	Structured Analysis & Design	Object Oriented Modeling	Component Based Development	Service Oriented Modeling	Service Oriented Modeling	Service Oriented Modeling	Grammar Oriented Modeling
Applications	Modules	Objects	Components	Services	Process Integration via Services	Process Integration via Services	Dynamic Application Assembly
Architecture	Monolithic Architecture	Layered Architecture	Component Architecture	Emerging SOA	SOA	Grid Enabled SOA	Dynamically Re-Configurable Architecture
Infrastructure	Platform Specific	Platform Specific	Platform Specific	Platform Specific	Platform Specific	Platform Neutral	Dynamic Sense & Respond
	Level 1	**Level 2**	**Level 3**	**Level 4**	**Level 5**	**Level 6**	**Level 7**

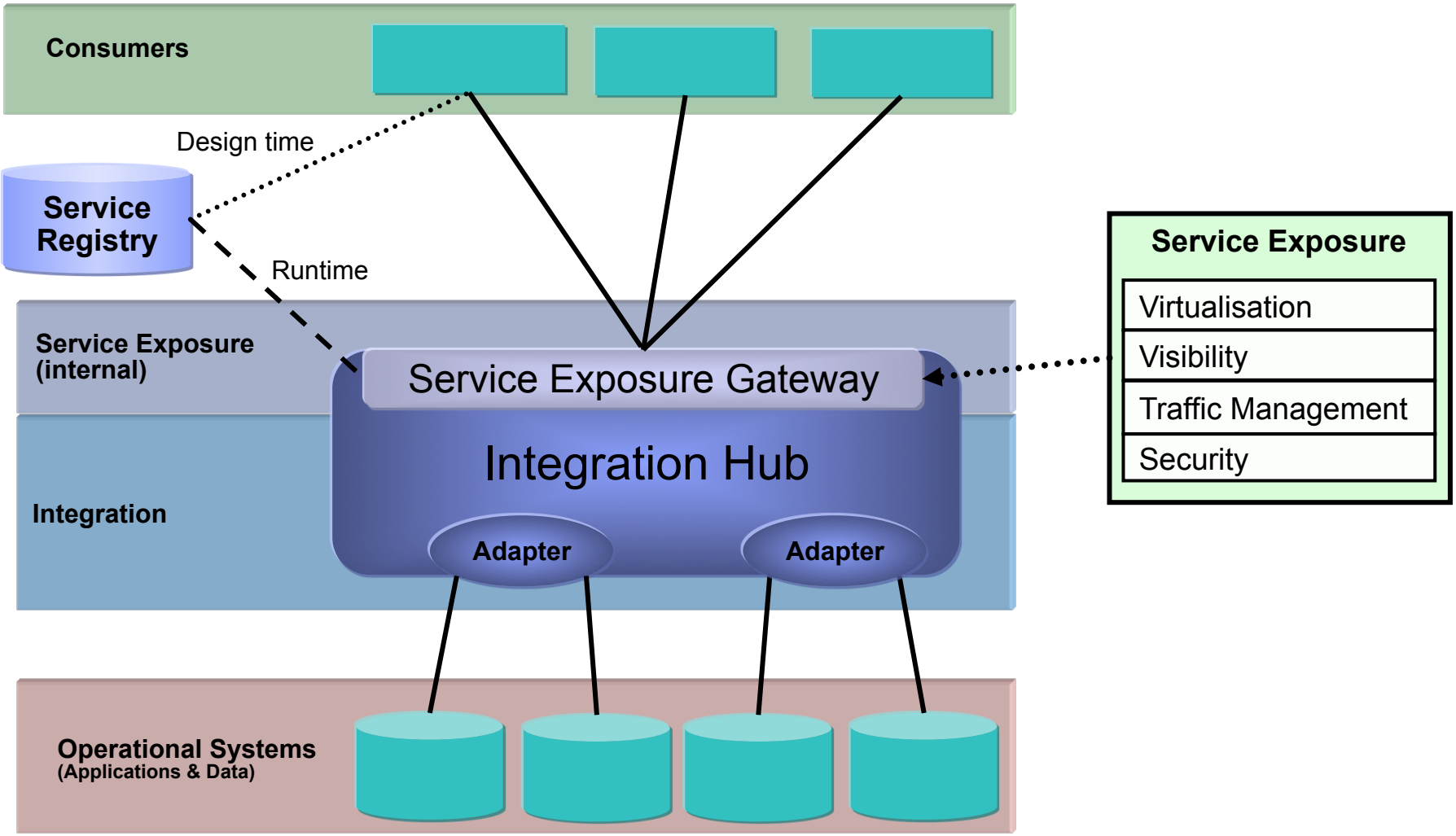
Primary focus of this presentation

https://collaboration.opengroup.org/projects/osimm

Web “1.0” – From Hub and Spoke to SOA



The role of the Service Exposure Gateway

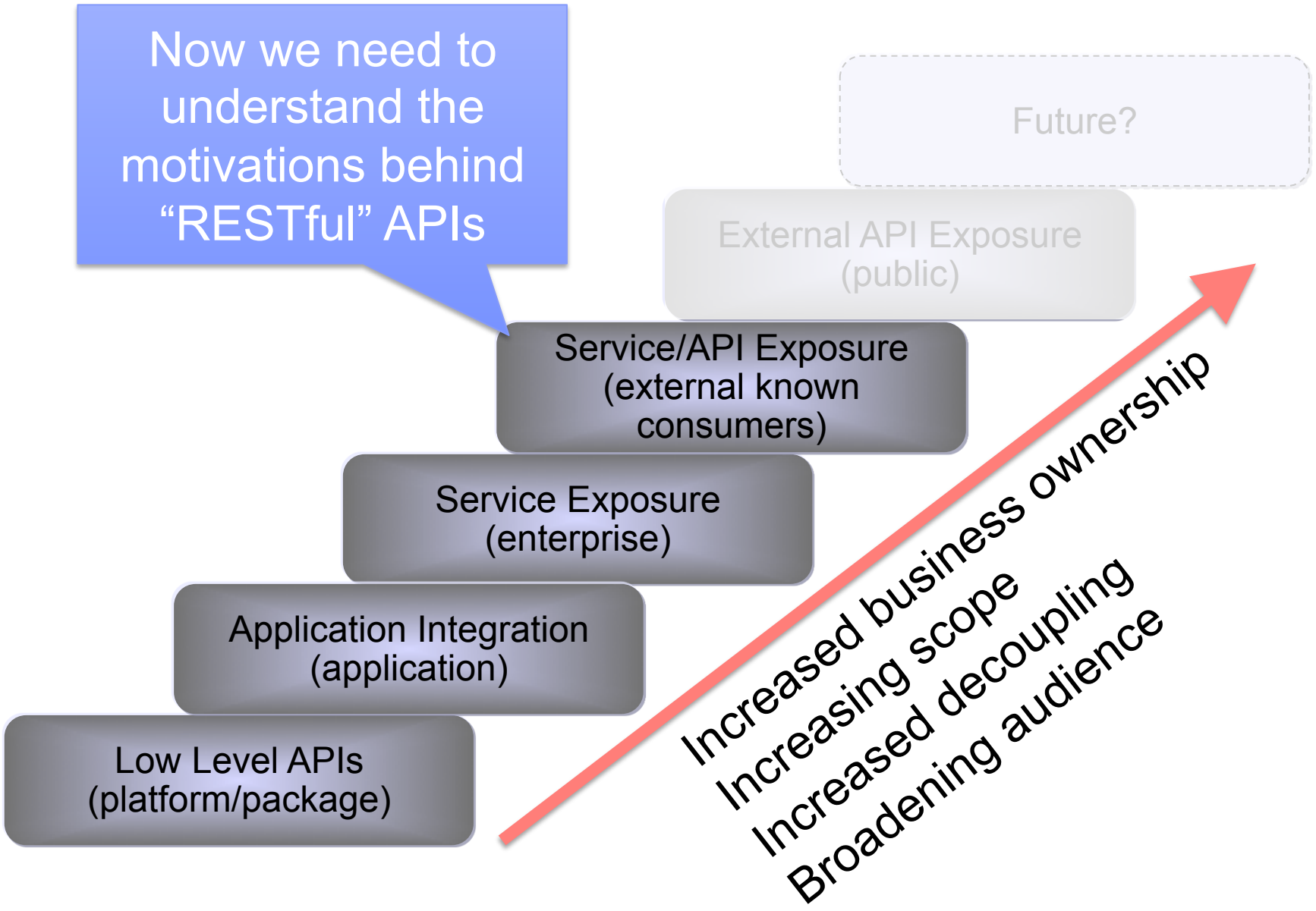


Common challenges with early SOA maturity

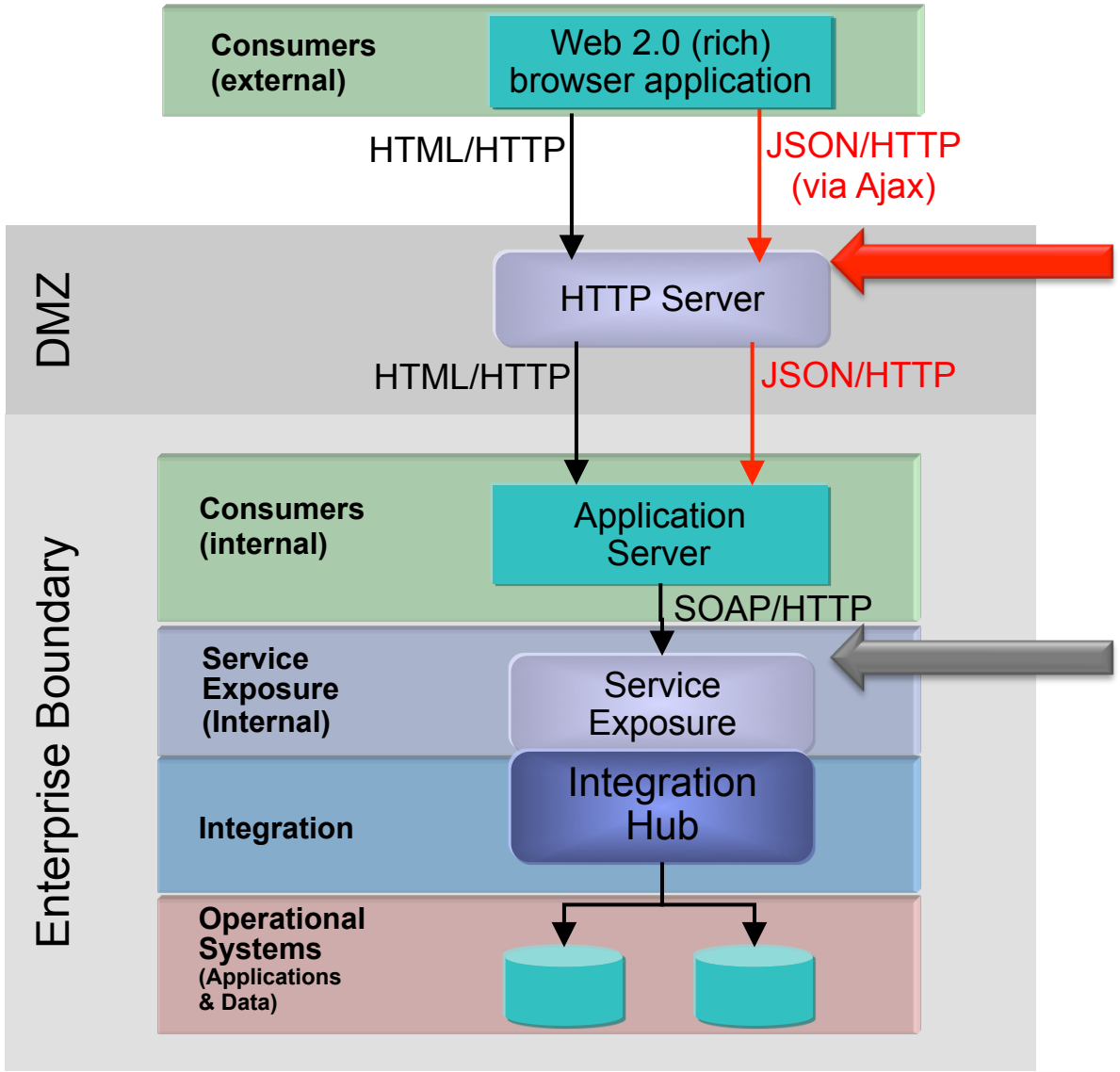
- Issues with SOAP and XML
  - Multiple different flavors of SOAP. Very late consolidation on Document/literal wrapped WSDL over “RPC/encoded”, “RPC/literal” and “Document/literal”
  - No first class representation of action/verb/method. Even now you have to parse to the SOAP body element name.
  - XML provides too many ways to represent the same thing. Attributes vs. Elements, representations of namespace, arrays etc.
  - XML had no native language bindings. DOM trees are ugly to navigate.
  - XML Schema took the eXtensible out of XML. Rigidity of complex data typing combined with early XML parser inflexibility made versioning complex quickly.
- Organizational issues
  - Most services were created by projects. Services were only good for first use.
  - Funding for SOA was hard to come by, but because of immaturity SOA was expensive.
  - Metrics for re-use in the wrong place. Most re-use was happening not at the service exposure level, but in the underlying integration layer.
  - Services were often just a façade on a set of still mis-aligned components. No re-engineering budget.
- Other technical issues
  - A whole new realm of security mechanisms to support, with little if any prior history. E.g. WS-Security, SAML.
  - Tools were immature for common needs e.g. mapping, logging/auditing/monitoring frameworks.

However, the tools and techniques may have been immature, but the fundamental premise of SOA, to make functions and data available in a re-usable way, **is still essential today**.

Evolving exposure of business function



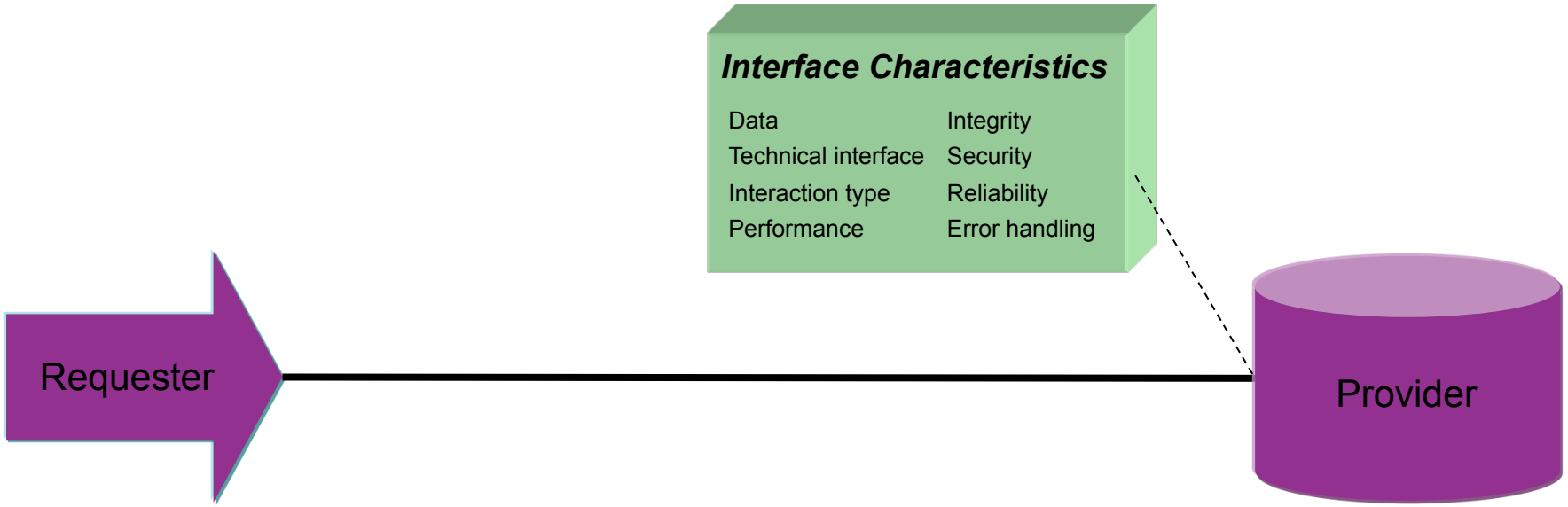
Web 2.0 interaction patterns
Introducing HTTP/JSON interactions



Fine grained
application scoped
"RESTful API" exposure

Course grained
enterprise scoped
"web service" exposure

Characterising the interface



Capturing integration complexity
http://www.ibm.com/developerworks/websphere/techjournal/1112_clark/1112_clark.html

Protocol comparison

	SOAP web services	"REST" APIs
Protocol	SOAP	HTTP
Transport	HTTP typically (but JMS/MQ common)	TCP/IP
Data format	XML schema	JSON or XML, or request URL parameters
Interface definition	WSDL file	By inspection/ documentation
Action/operation	Inferred by XML element name	URL and HTTP verb
Response types	XML and MIME attachments	JSON, XML, and other MIME types

***Note**: "REST" is NOT a formalised protocol in the same way that SOAP is. More on this later in the presentation.*

What is a RESTful interaction? (REST = Representational State Transfer)

For a more formal definition of REST see here:
http://www.ics.uci.edu/~taylor/documents/2002-REST-TOIT.pdf

- A component interaction style that ensures *independence*, *performance*, *scalability* and *consumability*.

- Some key constraints of a RESTful architecture
  - **Uniform Interface** (*consumability*, *independence*)
    - Identification of resources via unique references
    - Manipulation via these unique references
    - Self-descriptive messages
  - **Cachable** (*performance*, *independence*)
  - **Stateless** (*scalability*, *independence*)

- HTTP 1.0/1.1 were designed using RESTful principles
- Rejuvenated recently for “Web APIs” typically using HTTP/JSON or HTTP/XML and using the core HTTP verbs for “(S)CRUD”

“RESTful” interfaces – important observations

- There is **no official standard for RESTful interactions**
  - However, a set of style guidelines around how to write interfaces that play to the strengths of HTTP’s RESTful nature have reached communal consensus.
  - These principles take advantage of HTTP’s existing infrastructure for security, caching, resource identifiers (URI/URL), standard verbs (GET, PUT, POST, DELETE), media type negotiations etc.
  - The JSON data format is popular for payloads on RESTful interfaces due to its simplicity and native javascript support, but XML is also common.
- **“SOAP” is not well suited to RESTful interfaces**
  - SOAP circumvents many of the core RESTful properties of HTTP. It plays purely within the payload so its standard cannot reach into the transport (URIs, HTTP headers, authentication, mime types for example).

Data Format differences

XML

First class namespaces but multiple representations
Rich data types, but requires a “schema”
Validation, but requires a “schema”
Code heavy to navigate DOM
Most bindings required schema

Header required

Attribute or element for values?

```
<?xml version="1.0" encoding="UTF-8"?>
<order orderid=”123456” xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=”order.xsd">
  <orderperson>Joe Bloggs</orderperson>
  <deliveryaddress>
    <name>Fred Smith</name>
    <address>High Street, London</address>
  </deliveryaddress>
  <itemlist>
    <item>
      <title>Romeo and Juliet</title>
      <quantity>1</quantity>
      <price>9.99</price>
    </item>
    <item>
      <title>Pride and Prejudice</title>
      <quantity>1</quantity>
      <price>10.99</price>
    </item>
  </itemlist>
</order>
```

Arrays implied

Numbers are text

All tags duplicate text

JSON

Native parsing in Javascript
Single way to represent data values
No “schema” to define the structure
Slimmer (“low fat xml”)
Unburdened by namespaces
First class representation of numbers vs text
First class representation of arrays

```
{
  “orderid” : ”123456”,
  “orderperson” : “Joe Bloggs”,
  “deliveryaddress” : {
    “name” : “Fred Smith”,
    “address” : “High Street, London”,
  },
  “itemList” : [
    {
      “title” : “Romeo and Juliet”,
      “quantity” : 1,
      “price” : 9.99
    }
    {
      “title” : “Pride and Prejudice”,
      “quantity” : 1,
      “price” : 10.99
    }
  ]
}
```

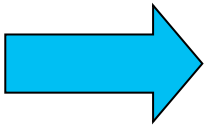
Explicit arrays

Number representation

“Non-extensible” e**X**stensible **M**arkup **L**anguage

*The issues with **response** data structure changes*

```
<response>
  <order>
    <reference>123<reference>
    <quantity>2</quantity>
    <productId>KI987</productId>
  </order>
</response>
```



```
<response>
  <order>
    <reference>123</reference>
    <quantity>2</quantity>
    <productId>KI987</productId>
    <status>SHIPPED</status>
  </order>
</response>
```

- Formally not backward compatible as consumer is passed unexpected data not present in their schema
  However…
  - Most common interface change due to incremental evolution of interfaces
  - Treating as non-backwardly compatible is very expensive in refactoring/versioning
  - Many (but unfortunately not all) consumers **can** accommodate it without any refactoring, but can you be sure?

- REST/JSON interfaces do not suffer from this problem.
  - JSON parsing requires no schema
  - The data is completely self defining
  - Consumers simply read the data they need
  - Can still get tripped up with deep data comparisons etc.

Protocol – Examples

SOAP/HTTP

```
POST /ordermanagement HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://…"
soap:encodingStyle="http://…">
<soapenv:Header>
  <wsse:Security soapenv:mustUnderstand="1"
      xmlns:wsse="http://...xsd">
    <wsse:UsernameToken>
      <wsse:Username>John</wsse:Username>
      <wsse:Password
        Type="http://…">Doe</wsse:Password>
    </wsse:UsernameToken>
  </wsse:Security>
</soapenv:Header>
<soap:Body xmlns:m="http://www.example.org/
ordermanagement">
  <m:AddOrderItem>
    <m:order orderid="123456">
      <m:item>
        <m:title>Romeo and Juliet</m:title>
        <m:note>Special Edition</m:note>
        <m:quantity>1</m:quantity>
        <m:price>9.99</m:price>
      </m:item>
    </m:order>
  </m:AddOrderItem>
</soap:Body>
</soap:Envelope>
```

A C B D E

“REST”/HTTP

With XML

```
POST /orders/123456/item HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
Authorization: …
<?xml version="1.0"?>
<m:item
  xmlns:m="http://www.example.org/ordermanagement">
    <m:title>Romeo and Juliet</m:title>
    <m:note>Special Edition</m:note>
    <m:quantity>1</m:quantity>
    <m:price>9.99</m:price>
</m:item>
```

B C D E A

With JSON

```
POST /orders/123456/item HTTP/1.1
Host: www.example.org
Content-Type: application/json; charset=utf-8
Content-Length: nnn
Authorization: …

{
  “title” : “Romeo and Juliet”,
  “note” : “Special Edition”,
  “quantity : 1,
  “price” : 9.99
}
```

B C D E A

Protocol – Pros/cons

SOAP/HTTP

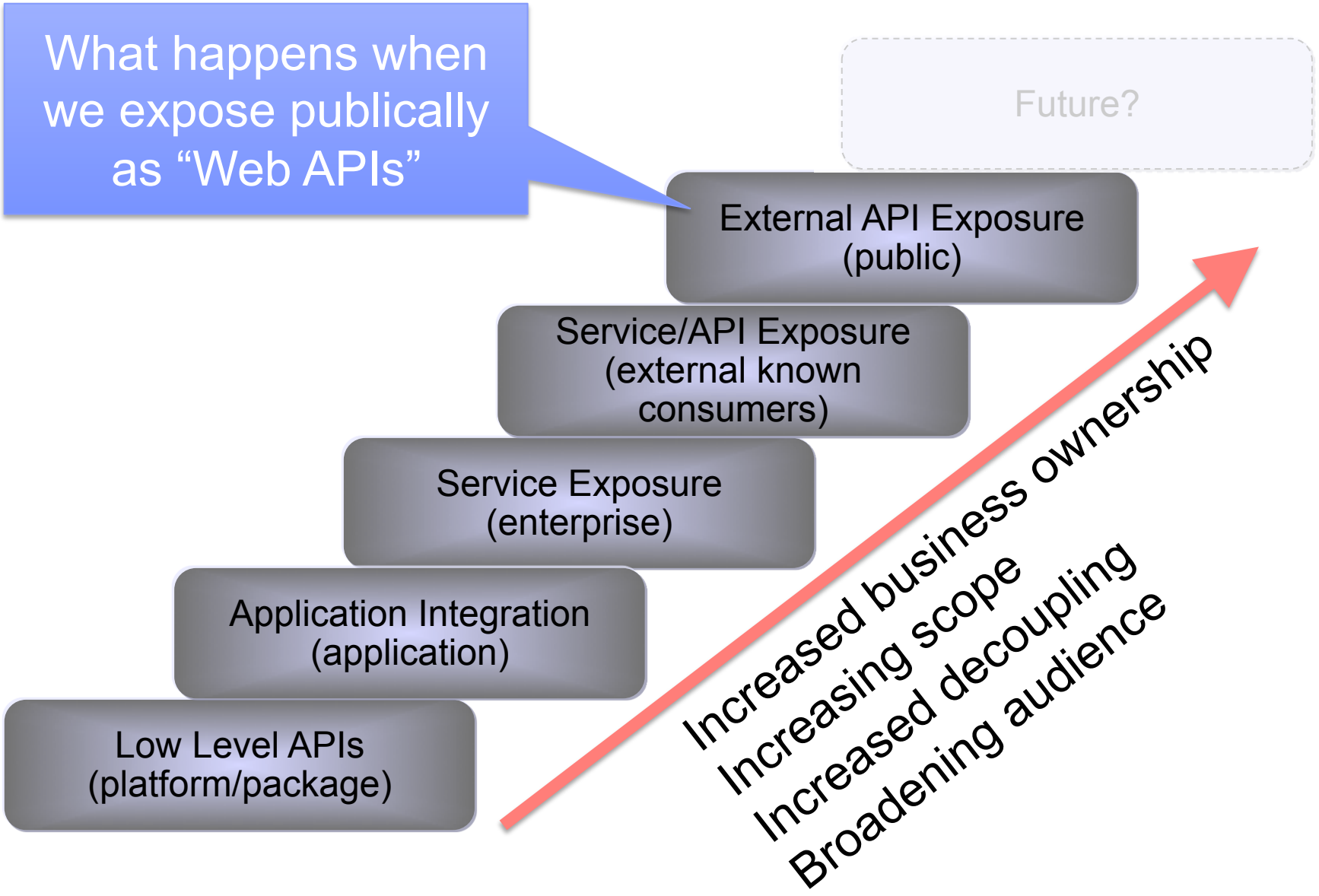
- Pros
  - Enables object oriented verb style
  - Support for transactionality
  - Many standards for security
  - Can use any transport, even asynchronous messaging.
  - Flexible routing patterns
  - Can be used over any transport
- Cons
  - Complex to get simple things done
    - Minimum structure for a request is full SOAP envelope
  - Forcibly XML data format
    - Less natural to parse and navigate in browser side javascript
  - Transport agnostic
    - Cannot benefit from inherent capabilities of the underlying transport

“REST”/HTTP

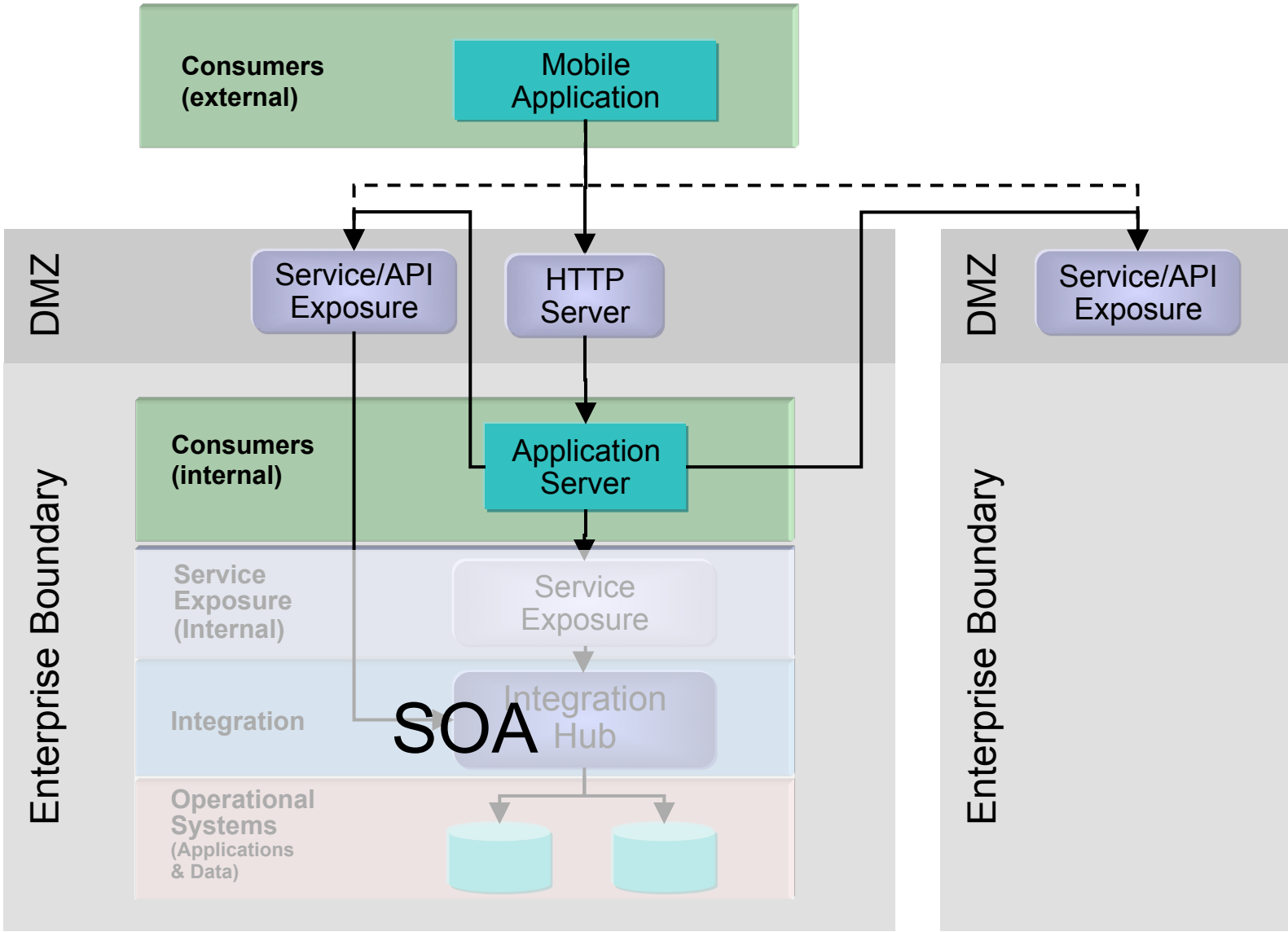
- Pros
  - Simple things are simple
  - Fully leverages existing mature HTTP infrastructure – URLs, security, verbs, caching etc.
  - Can use any data format, though JSON and XML most common.
  - Provides unique references to resources.
  - Simplifies design of the service model to (S)CRUD
- Cons
  - Bound to HTTP. Harder to use over other transports.
  - Can be chatty – multiple requests to achieve one action
  - Highly “functional” operations can be hard to represent

But ultimately, the protocol is only a small part of the story!

Evolving exposure of business function



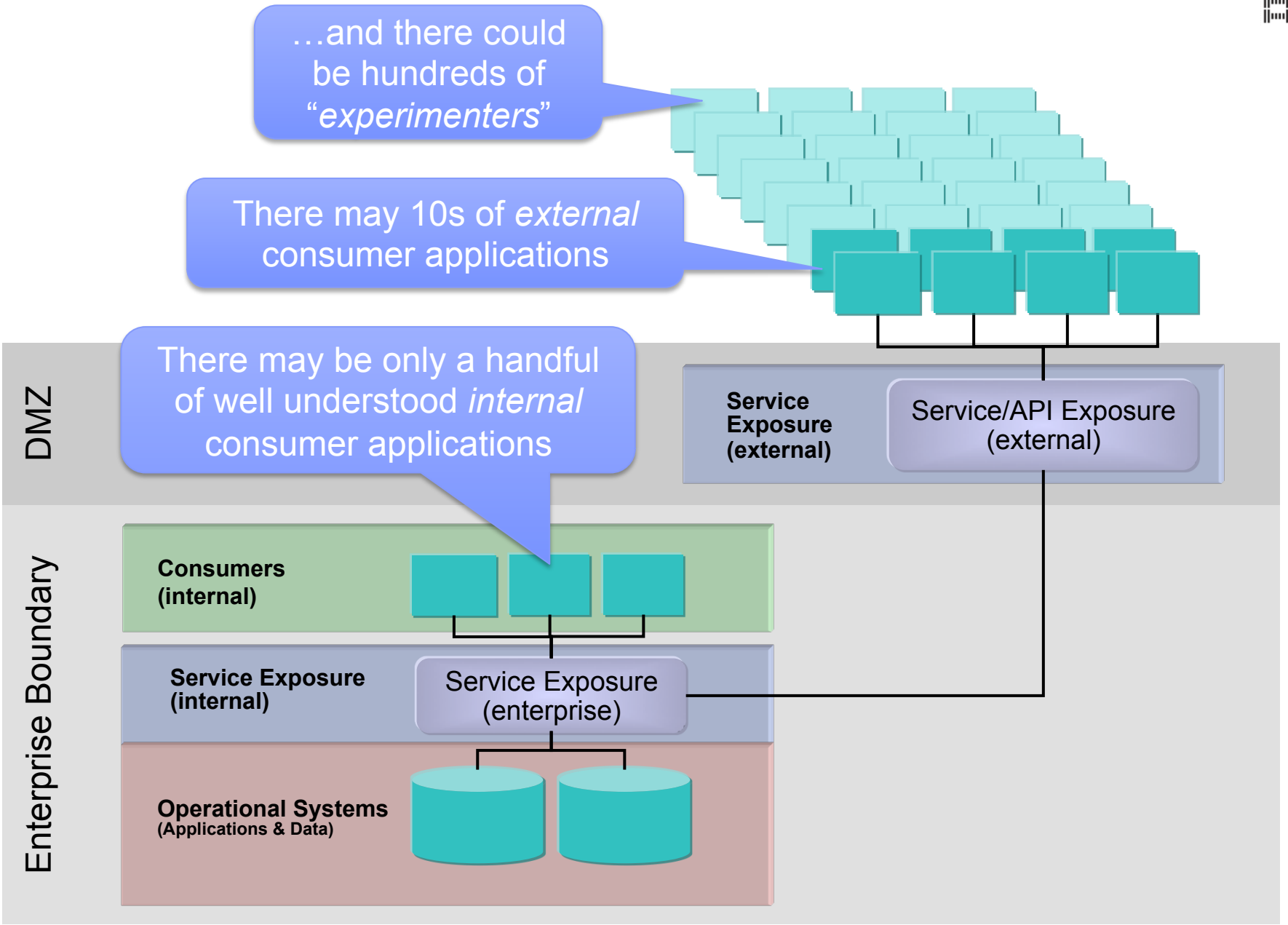
Mobile device side “app” interaction styles
Formalising of API exposure



Application Programming Interface (API)
An old term re-invented

- API
  - A well defined interface to enable one component to talk to another, programmatically, without understanding it's implementation.
  - Examples would include
    - Database drivers
      - provide an API that allows programming languages with a mechanism to talk to databases without having to understand the details of how the communication is done (e.g. how JDBC actually works).
    - Enterprise Java Beans – Home/Remote interfaces
      - provide remove interfaces that hide the complexities of the underlying RMI requests to enable you to call java applications located in another server.
- "Web APIs"
  - The new term currently used to describe HTTP/JSON, HTTP/XML APIs that are often publicly accessible.
  - HTTP/SOAP *could* be used to expose a web API, but for reason we will discuss later, typically not the preferred method.

Different between the internal and external service consumer



Architectural style differences

“Web Services”

- Coarse grained
- Function oriented
- Optionally transactional
- Numerous maturing security options
- Sophisticated data format
- Typically SOAP/HTTP

- Best suited to
  - Internal (within enterprise)
  - System to system
    - E.g. Business process automation

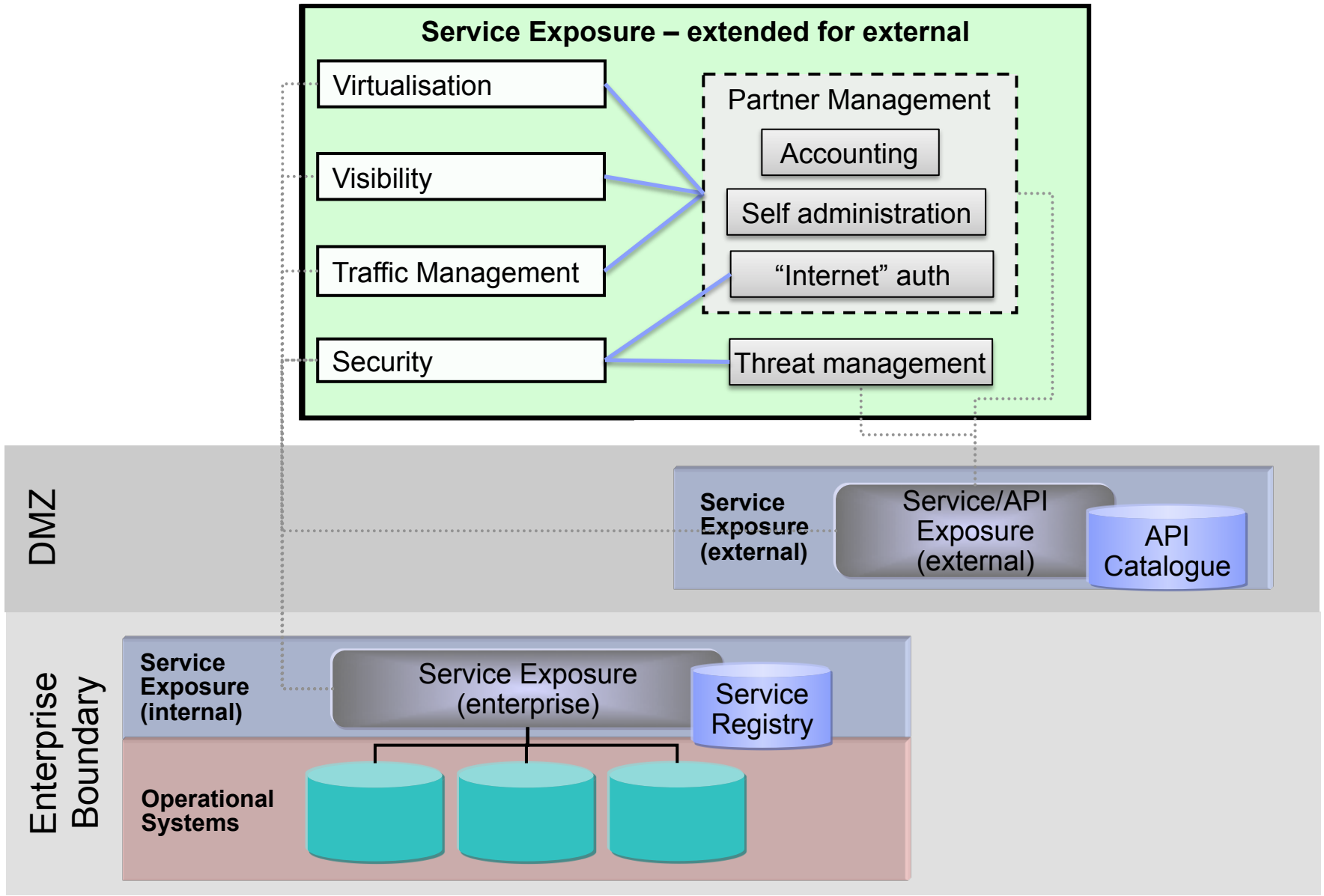
“Web APIs”

- Fine grained/chatty
- Resource/data oriented
- Non-transactional
- Limited but mature security options
- Simple lightweight data format
- Typically “REST” – e.g. HTTP/JSON

- Best suited to
  - Internal or external
  - User interface to system
    - E.g. Mobile app to multiple APIs

Note: “Web Service” and “Web API” are not formal terms with agreed definitions. They are just two of the most common terms used to explain styles in use today.

Additional requirements for external exposure - Introducing “API management”



Core conceptual differences with Web APIs

- Your *audience* is different
  - Web APIs offer radically new business models. New ways of making money takes broad innovation. Your audience are now “app developers”
    - No longer just inward looking for innovation. We are now crowd sourcing new business ideas externally.
  - The app market is an loosely controlled sand pit.
    - Recall the pervasiveness of “unofficial” office applications?
  - Web APIs are the public persona of your organisation
    - For some, not having a Web API is like not having a web page 10 years ago.
  - Did we know where Web 1.0 was going?
    - Initially brochure-ware. Suddenly eCommerce and monetisation of the web.

- Web APIs “*look and feel*” different
  - Web APIs cannot make assumptions about the types of applications that will be created.
  - You cannot guess the usage of situational applications and mobile applications.
  - The API is “resource” based, meaning it is closely aligned with the data model.

- Web APIs are a “*product*” and must be treated like one
  - Your Web APIs are fighting for survival alongside your competitors Web APIs.
  - API Management Portal needs to be attractive
  - They simply make the data as accessible and the interface as consumable as possible.

Dispelling some myths about Web APIs

- ~~Web APIs are always HTTP/JSON~~
  - Many are HTTP/JSON, but there are a good proportion that are HTTP/XML.
  - Note that one of the benefits of using raw HTTP interactions is that you can use the in built mime type support. A response could even be pdf!
- ~~Web APIs are always RESTful~~
  - Most web APIs comply with some of the tenets of RESTful interfaces, such as being stateless. However, desirable though RESTfulness may be, few employ all RESTful recommendations.
  - Some Web APIs derive from older HTTP/SOAP web services, and are little more than a translation of that into HTTP/JSON. Again, REST is a paradigm, HTTP/JSON is just a protocol choice – it depends how you use it
- ~~Web APIs don't use SOAP~~
  - Using SOAP/XML doesn't disqualify an interface from being classed as a Web API. It might however suggest that it is less RESTful.

Thought experiments

- Consider Web API first?
  - Why expose APIs internally at all. If they're useful, they're useful externally too (maybe over VPN)

- What is a “test environment” for an API based application?
  - How will we provide test environments for our API consumers. Are we really talking about massive multi-tenancy rather than shared systems.
  - How would you provide data into those environments.

- Do Web APIs simplify versioning?
  - Web APIs still need to be governed – they’re a product remember…
  - Neither SOAP, or REST/JSON have first class versioning mechanisms
  - REST/JSON tolerates changes in responses slightly better. Everything else is still hard!

- How do we manage reliability and data integrity over Web APIs?
  - Do we need to design differently to ensure web interactions over non-transactional protocols result in safe state changes? Do we need to introduce idempotence?
  - More fine grained, so may have multiple updates.

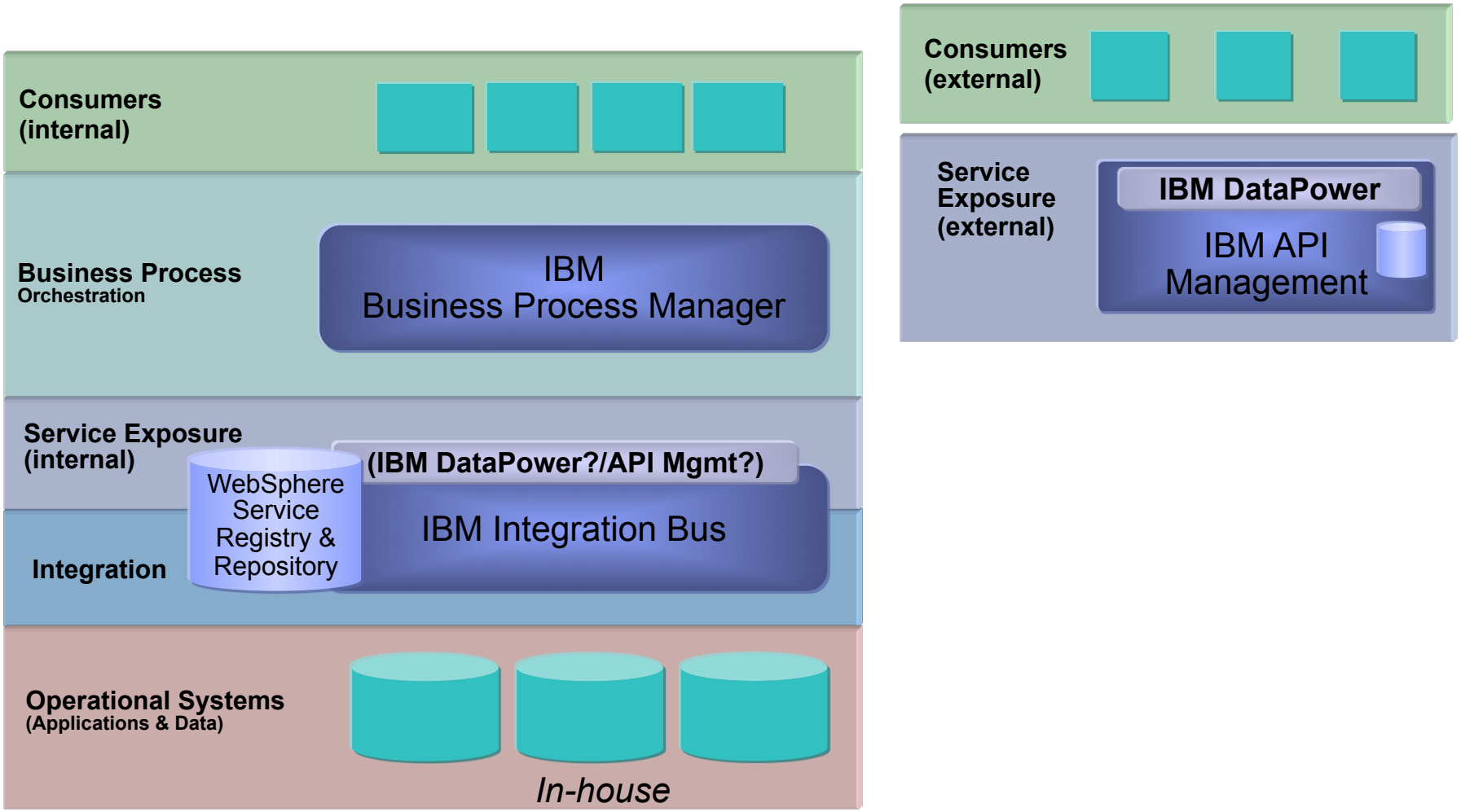
- How much do we expose?
  - Many SOAs initiatives stalled during over-enthusiastic top down analysis of the organization.
  - You’re unlikely expose your entire business over APIs. Focus in on a business domain or value stream.

Beware!
"Web API" actually has two completely different definitions

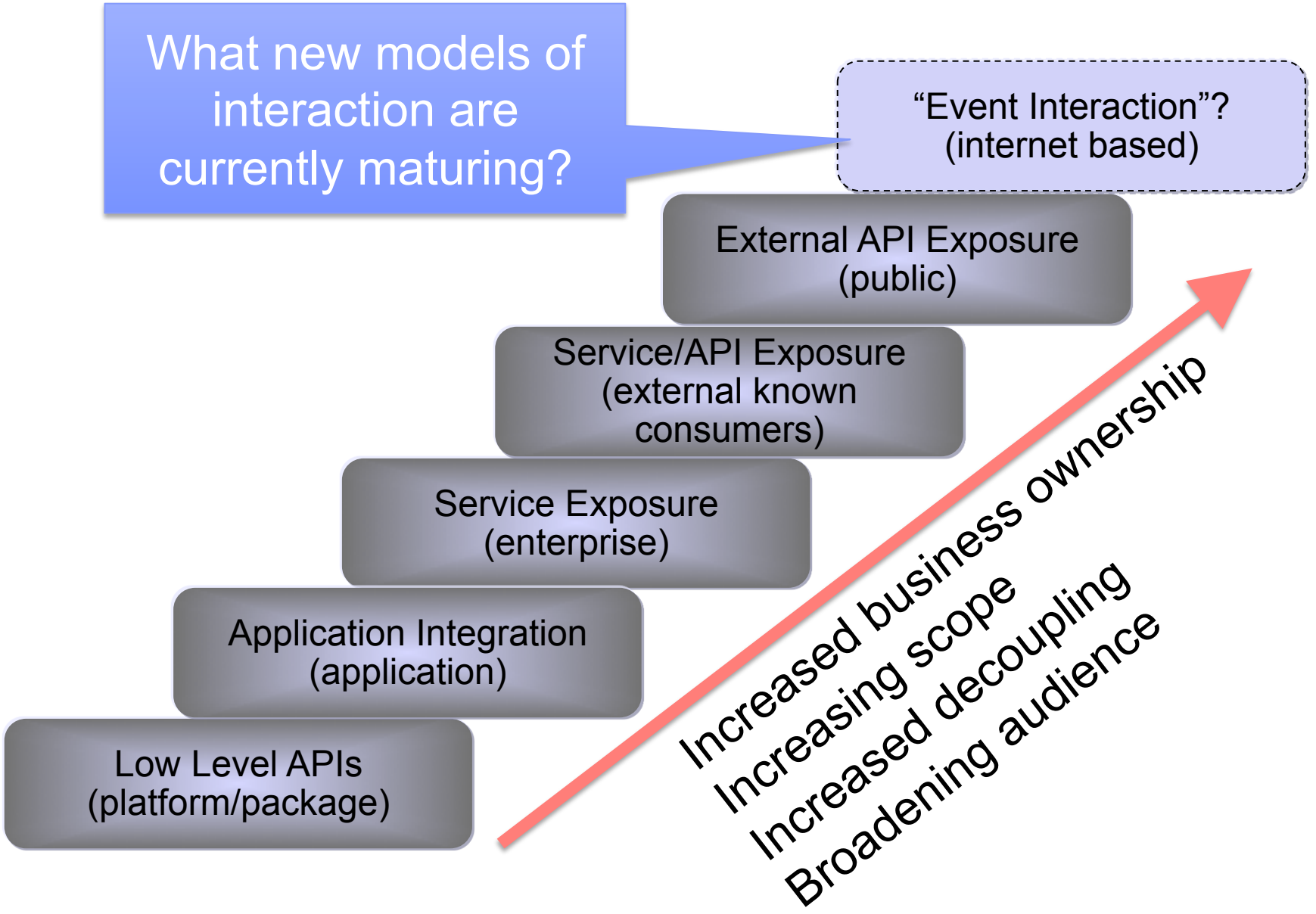
- Client side (an API within web browsers)
  - **This is NOT what we are talking about today!**
  - Browser side programming interfaces, typically using javascript libraries, to enable end-user interfaces to interact more powerfully with their environment.
  - For example to create safer more sandboxed browser side applications (Google "Native Client"), or to enable browser based applications to access device capabilities without the need for a native wrapper (Mozilla WebAPI).

- Server side (an API used to expose services on the web)
  - **This IS what we are referring to in this presentation!**
  - Programming interfaces used to expose functionality in back end operational systems over the internet by offering lightweight "RESTful" HTTP/JSON or HTTP/XML interfaces.
  - Google, Amazon, Facebook, Twitter all provide Web APIs to enable applications to easily access their data from mobile and other web appliations. More examples on http://www.programmableweb.com

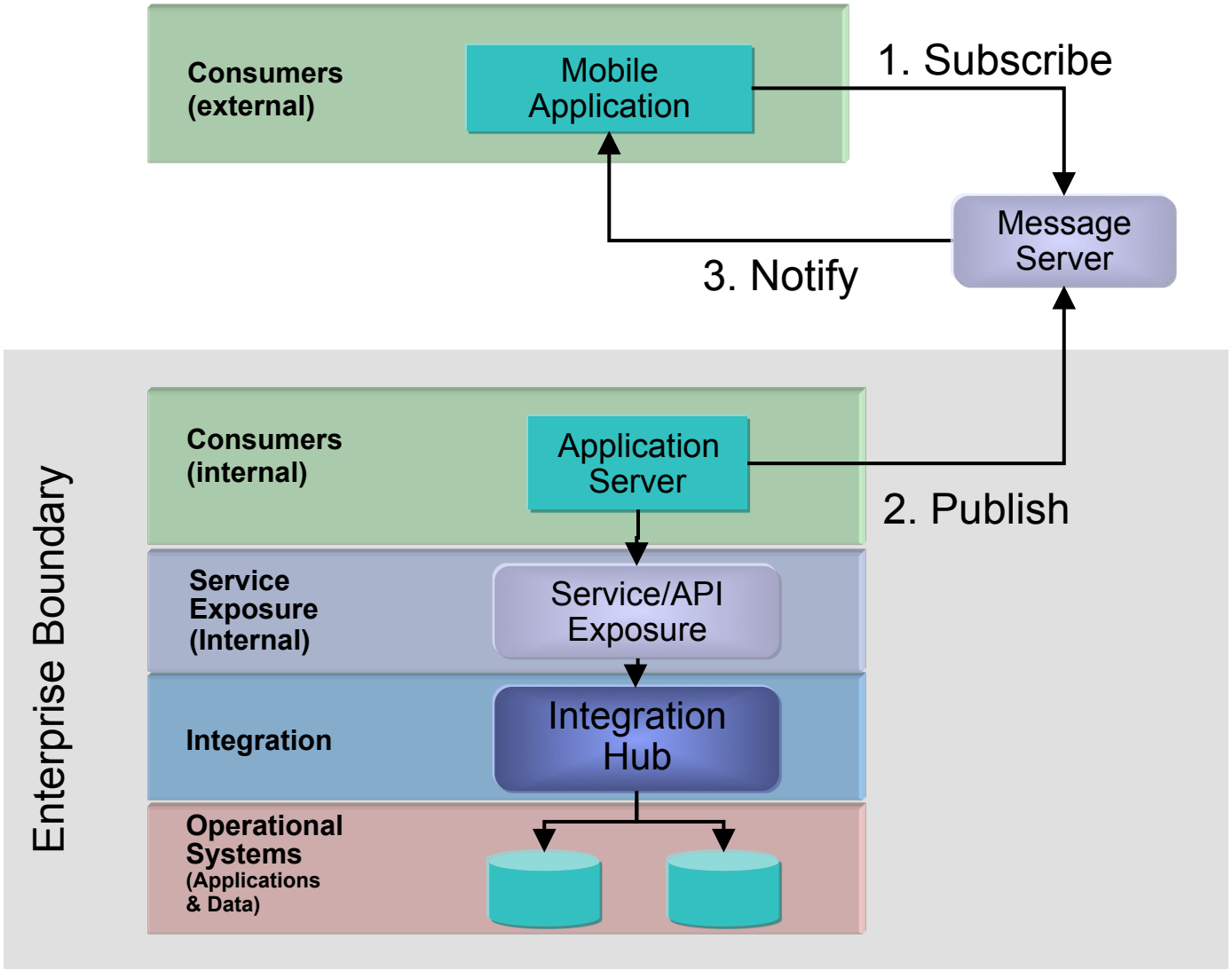
Extended reference architecture showing some relevant IBM products



Evolving exposure of business function



Mobile device event interactions – Publish/Subscribe



Events, messages, notifications
The re-invention of asynchronous interaction!

- New forces driving resurgence of asynchronous interaction
  - Internet of Things (IoT), mass notification requirements, disconnectable mobile apps, Event Driven Architecture (EDA)
- Asynchronous patterns are well established
  - e.g. Messaging, store/forward publish/subscribe etc.
- Messaging technologies are mature
  - e.g. WebSphere MQ, IBM Integration Bus (was Broker)

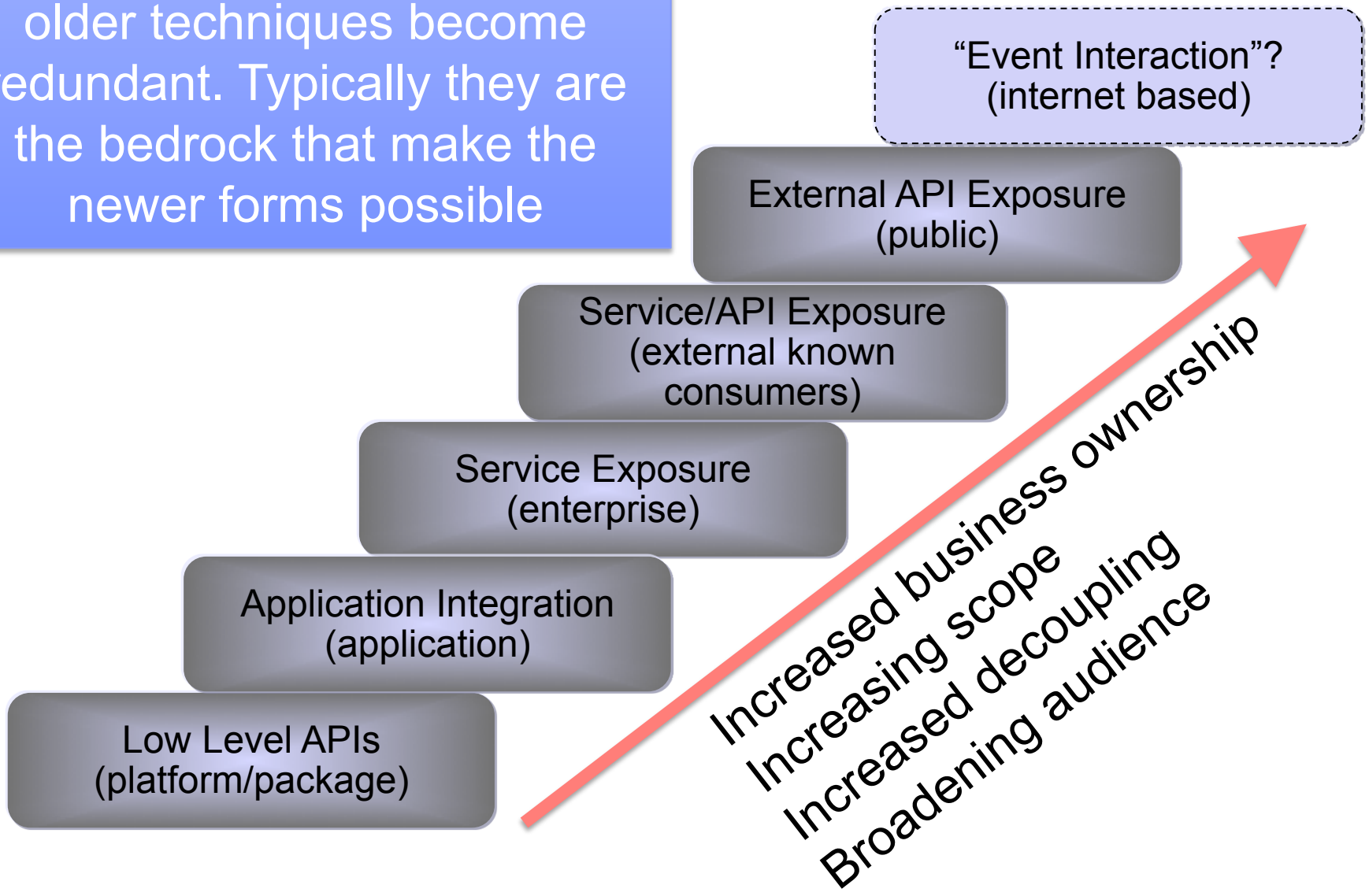
- Driving simplicity and efficiency
  - Why request information, when you could just be given it (notification)
  - Why send information to people who don't need it (publish/subscribe)
  - Why should a system have to be present/available when you want to communicate with it (store/forward)

- Messaging and notification patterns are now baked into mobile application infrastructure. Unfortunately they vary by platform and even by app, but standards are starting to emerge.
  - Tools such as IBM Worklight provide agnosticism across these platforms whilst the standards settle.

Note: Ajax although terms "asynchronous" is actually fully thread blocking from the browser down. The sense in which it is asynchronous that requests are not done on the users UI thread.

Evolving exposure of business function

Evolution does NOT mean the older techniques become redundant. Typically they are the bedrock that make the newer forms possible



What did SOA ever do for us?

SOA initiatives to date have provided:

- Rationalization of components around business functionality
- Standardized vocabularies (data models, service models)
- Simplified data models and functions
- Better documentation of available interfaces
- Support for synchronous real-time interaction
- Feedback from consumers on ease of use
- Innovation around what can be exposed and how it can be used
- Improvements in integration tooling, and infrastructure. e.g.
    - E.g. Adapters, meta-data discovery, data formatting, graphical data mapping, industry data models, registries, API management, gateway appliances,

SOA was simply about good architectural principles around a layered architecture. Web APIs are just an example of one of the ways an SOA can mature.

Questions?

Impact 2014

Be First.

#ibmimpact

We Value Your Feedback

- Don't forget to submit your Impact session and speaker feedback! Your feedback is very important to us – we use it to continually improve the conference.
- Use the Conference Mobile App or the online Agenda Builder to quickly submit your survey
    - Navigate to "Surveys" to see a view of surveys for sessions you've attended

Thank You

Impact2014

#ibmimpact

Be First.

Legal Disclaimer

Interface Characteristics

Data
- Principal data objects
- Operation/function
- Read or change
- Request/response objects

Technical Interface
- Transport
- Protocol
- Data format

Interaction type
- Request-response or fire-forget
- Thread-blocking or asynchronous
- Batch or individual

Performance
- Response times
- Throughput
- Volumes
- Concurrency
- Message size

Integrity
- Validation
- Transactionality
- Statefullness
- Event Sequence
- Idempotence

Security
- Identity/Authentication
- Authorisation
- Data Ownership
- Privacy

Reliability
- Availability
- Delivery assurance

Error Handling
- Error Management capabilities
- Known exception conditions

http://www.ibm.com/developerworks/websphere/techjournal/1112_clark/1112_clark.html

What are the fundamental questions that drive design

- Who owns it?
  - Who creates and prioritises the requirements?
- How long does it last?
  - What is the runtime lifespan of its instances?
- How often does it change?
  - How frequently do its requirements change?