

# **SERVICE ORIENTED ARCHITECTURE AND THE LEGACY ORGANIZATION**

David Koenig, Brookline Technology  
Michael Galarneau, Liberty Mutual

May 21, 2007

# Introduction

---

***Service Oriented Architecture (SOA) offers companies a trifecta of more modular applications, reduced development costs, and faster time-to-market. Recent advances are making it easier-and-easier to implement SOA across a broad range of systems.***

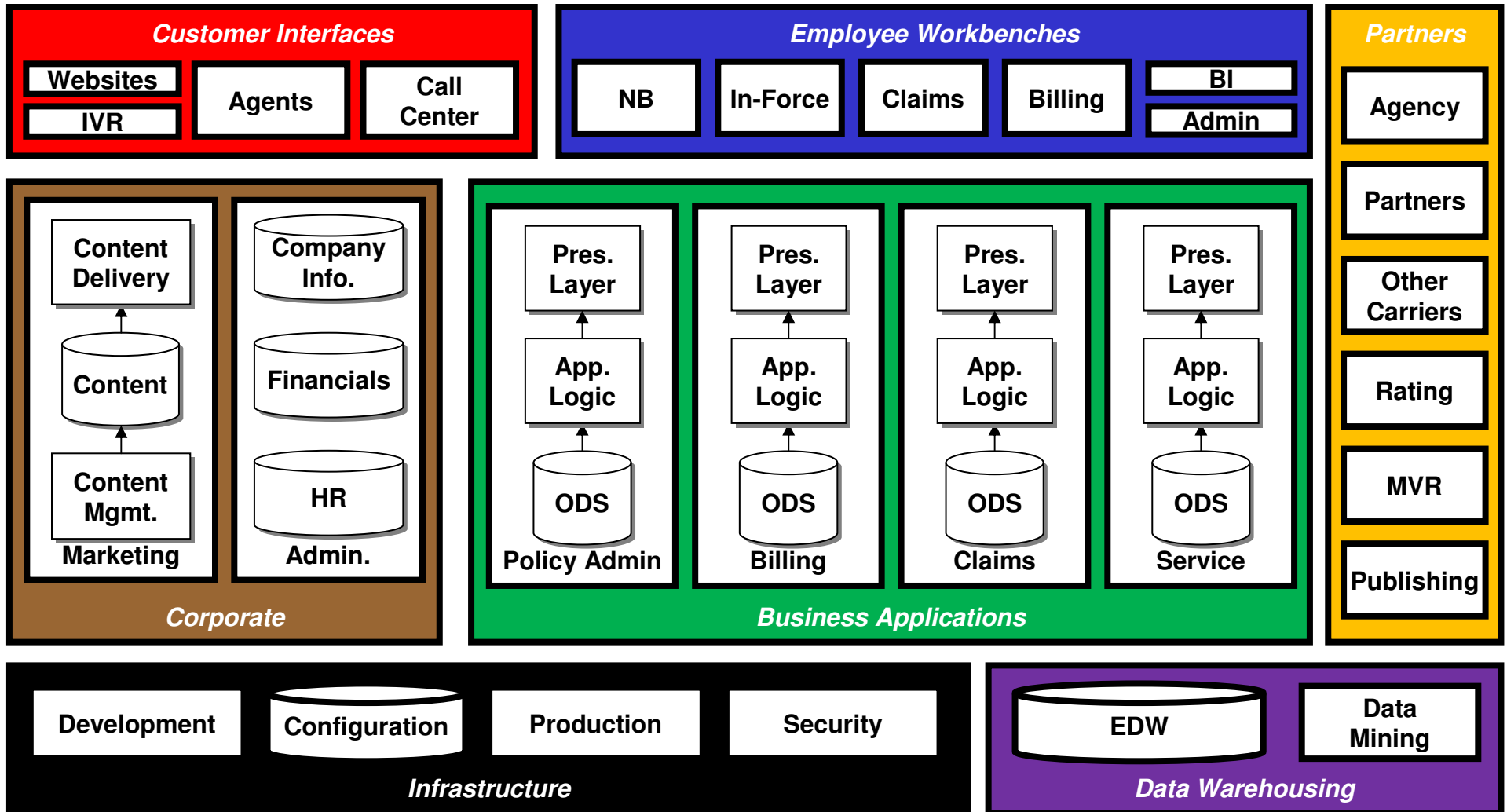
***However, SOA is not a silver bullet, especially for organizations that have large investments in older technologies. SOA initiatives often fail in legacy organizations, not because of poor design or lack of technical capabilities, but because of failure to govern how services are built once the design is complete. Introducing services incrementally, and learning from less-than-ideal experiences, are the first steps toward integrating SOA with existing legacy systems.***

***In this session, we will present a strategy for introducing enterprise-class SOA technology into the patchwork that constitutes the operating environment for many large organizations. We will also explore several examples of SOA initiatives that failed to realize their lofty goals, and how we used these experiences to improve on later iterations of our SOA strategy.***

***These examples span different applications at three different companies that were struggling to marry SOA with their legacy platforms. Each implementation achieved moderate success, yet under-delivered in ways that taught us where we needed to change our strategy and, more importantly, our expectations of Service Oriented Architecture.***

# Patchwork Legacy Architecture

Everyone has legacy stuff to deal with, often to the tune of \$100MM's in historical investment. You can't just start building a whole new set of applications from scratch. Nor would you want to abandon years of development, testing, and production experience. Nor could the organization afford the time, cost, and risk of such an approach.



# Getting Beyond the Hype

---

- From the vendor point-of-view:
  - *“The combination of SOA and Web Services is very close to being the ‘silver bullet’ that companies have been looking for to:*
    1. *Realize IT’s long-promised potential*
    2. *Justify IT expenses and capital outlays*
    3. *Provide non-technical people a clear understanding of what IT does, how they do it and their intrinsic value”*
- From the business point-of-view:
  - *“It always seems that this year’s next big thing is just a way for IT to get out of finishing last year’s next big thing.”*
- What does SOA mean to a legacy IT organization?
  - Constructing applications from components, services, and workflows
  - Design for interoperability and reusability
  - “Loosely-coupled” services and development teams (division of responsibilities)
  - Repurposing previous investment in existing, stable code
  - Strong standards and governance
- In other words, SOA is just good, basic n-tier development with focus on interoperability and standards

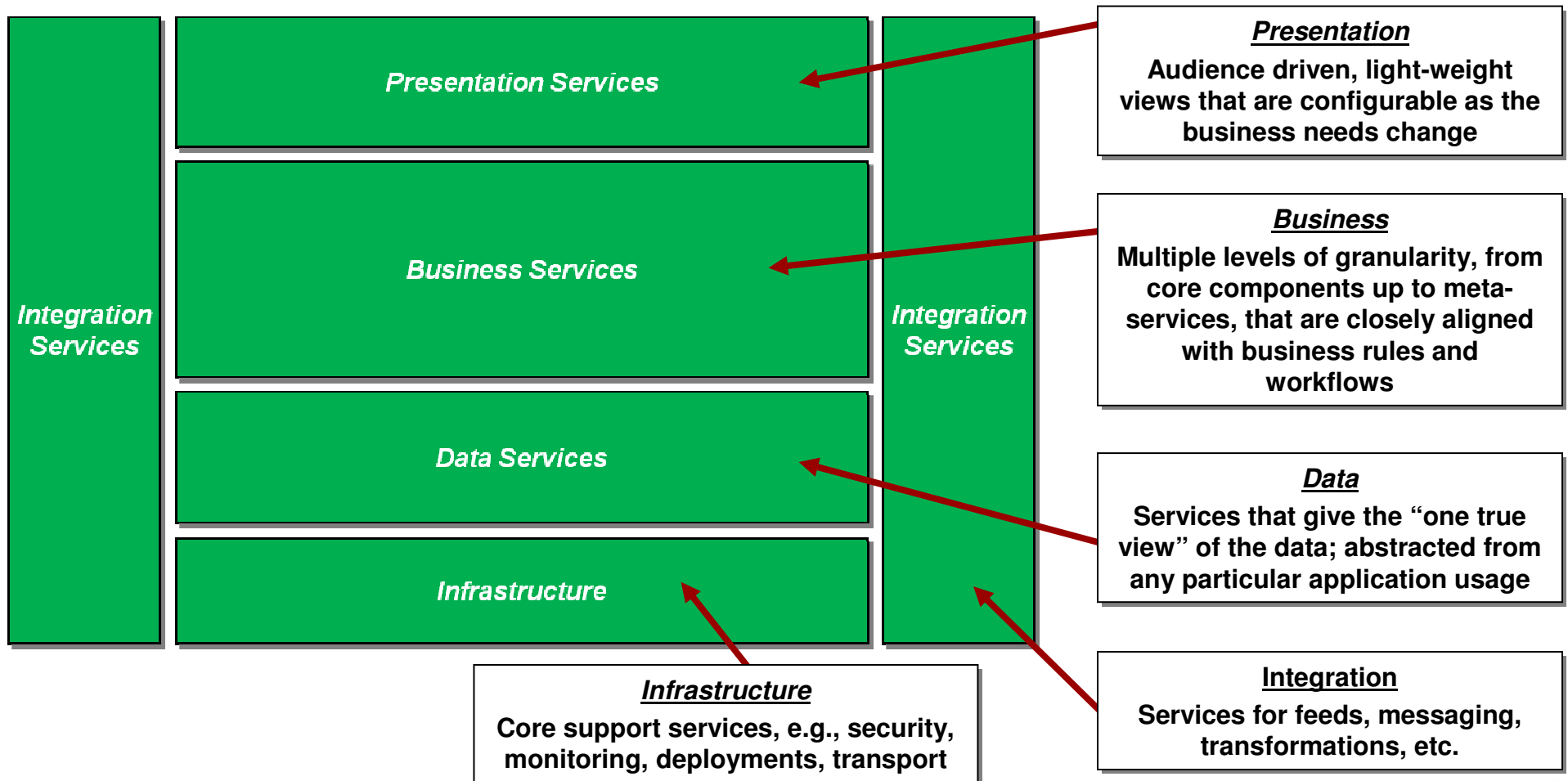
# Legacy Architecture and SOA

---

- **For the legacy organization, SOA provides a structure and a set of tools for delivering the new platform in parallel with current production systems**
  - Chaining services together into business processes
  - Reusing existing functionality where possible
  - Building new services only where necessary to advance business needs
  - One set of processes, one application, one technology component at a time
- **Done right, the benefits of SOA are enormous:**
  - Better business alignment, more targeted functionality
  - Faster time-to-market, without cutting corners and without “big bang” deliverables
  - More flexible and more configurable, to support changing business needs
  - More reuse, more return on investment
  - More scalable, more manageable, more monitorable infrastructure
- **Done wrong, and SOA is “yet another example of where IT didn’t live up to its promises” to the business**
  - Too much focus on technology and not enough on business functionality
  - Further fragmentation of the operating environment
  - New legacy code and new vendor products to support
  - Increased “lights on” maintenance and processing costs

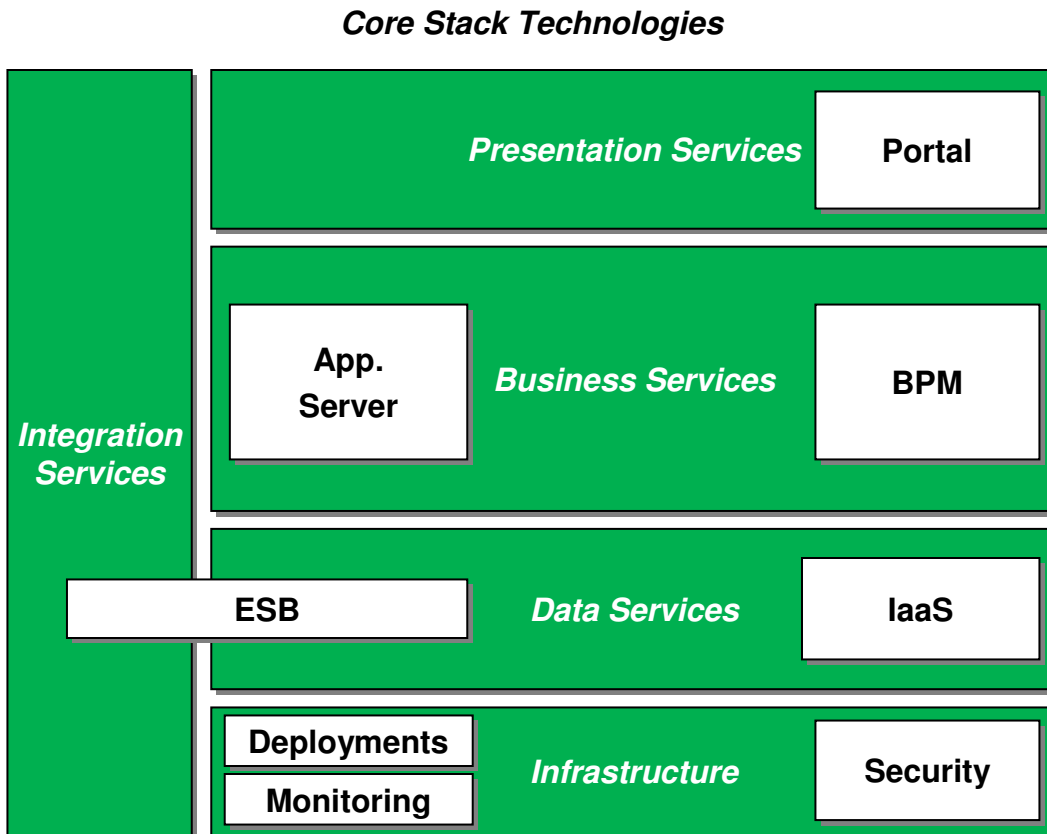
# Service Oriented Architecture – A Framework

SOA is an architectural framework that enables applications to be built as a set of granular services, and to interact with one another in a loosely-coupled, platform- and organization-independent manner. Facilitates reuse by allowing applications running on one platform to consume resources running on other platforms.



# The SOA “Stack”

With the acceleration of SOA adoption, vendors are offering better-and-better products at each tier of the platform. Key to making SOA work is respect for “The Stack”, an integrated set of tools that work together to support each tier. However, even with industry-standard products, successful implementation is no slam-dunk.



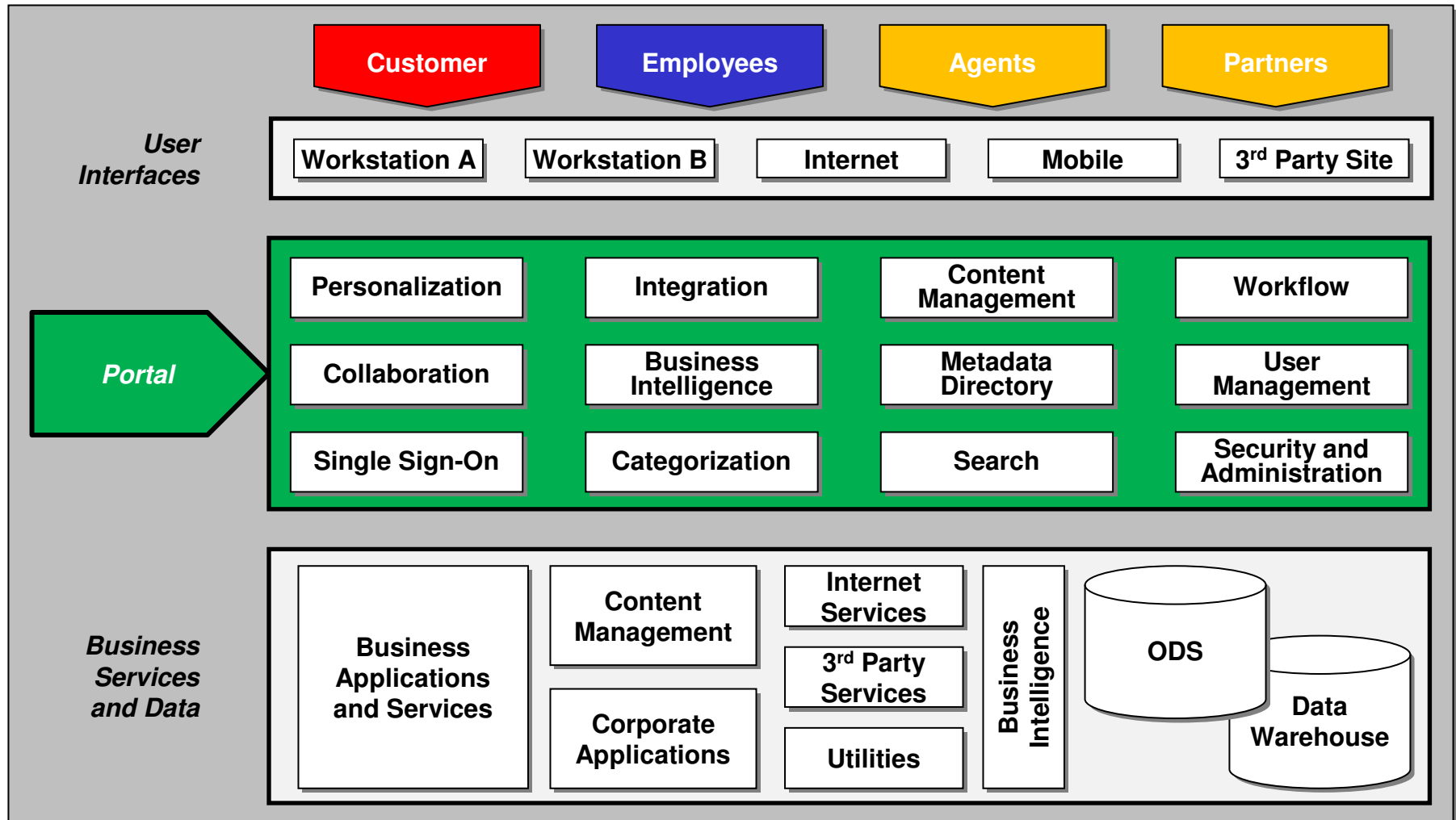
## Respect the Stack

- **Leading SOA vendors:**
  - BEA: WebLogic, AquaLogic
  - IBM: WebSphere
  - Microsoft: .NET, Sharepoint
  - Oracle: SOA Suite
  - SAP: Enterprise SOA, NetWeaver
  - Sun: Java System
  - Plus a multitude of specialty vendors providing niche solutions
- **“Good enough is good enough”**
  - Benefits of single-vendor far outweigh risks of integrating best-in-class products piecemeal
  - If you’re a \_\_\_\_\_ shop, you go with the \_\_\_\_\_ stack

**The one thing you absolutely don’t want to do is roll your own SOA tools**

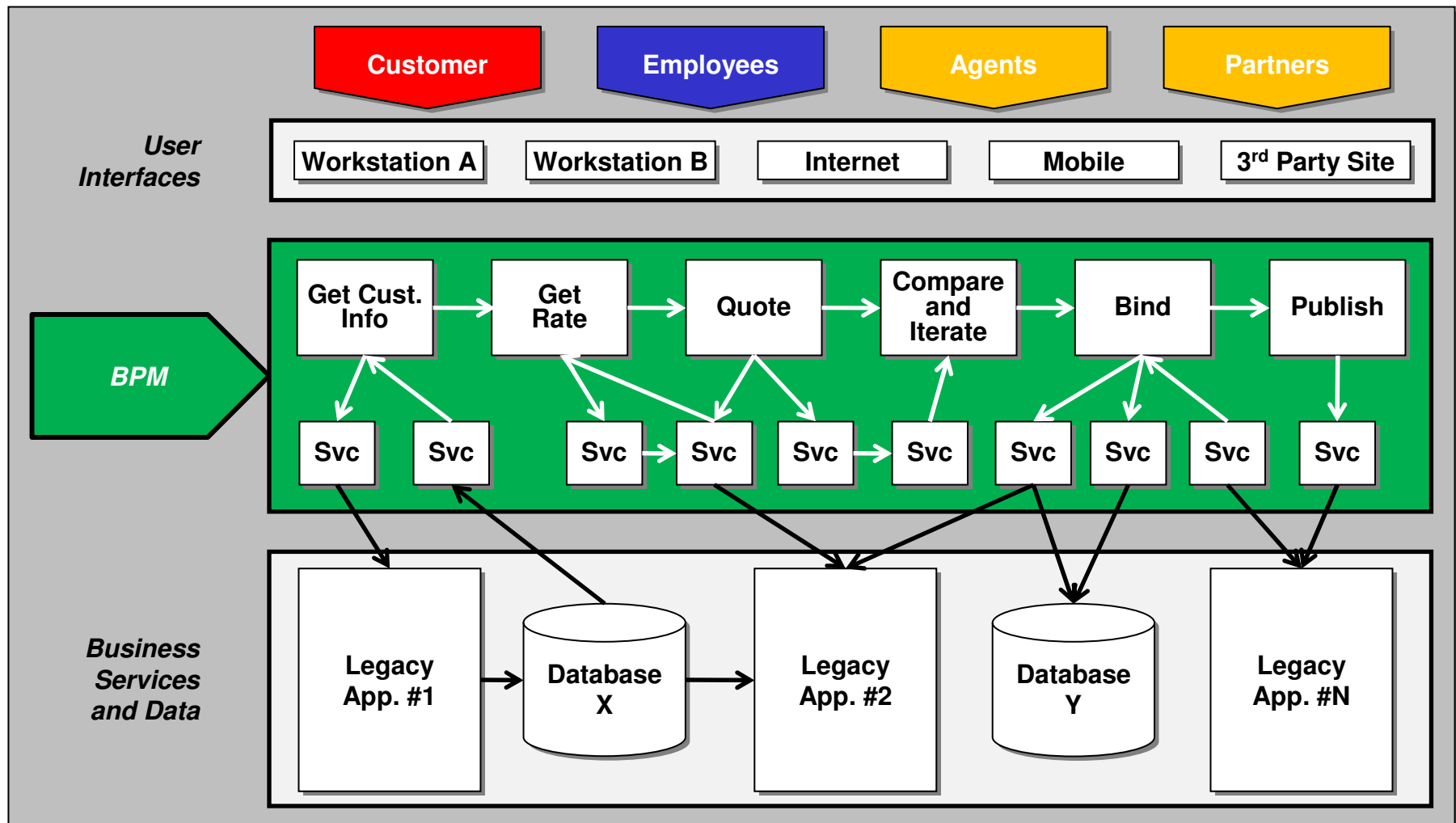
# SOA Stack – Portal

Portal technology forms the core of the SOA presentation tier. Portal provides the development framework and infrastructure for managing user interfaces across a broad range of websites, business applications, databases, and other information resources.



# SOA Stack – Business Process Management (BPM)

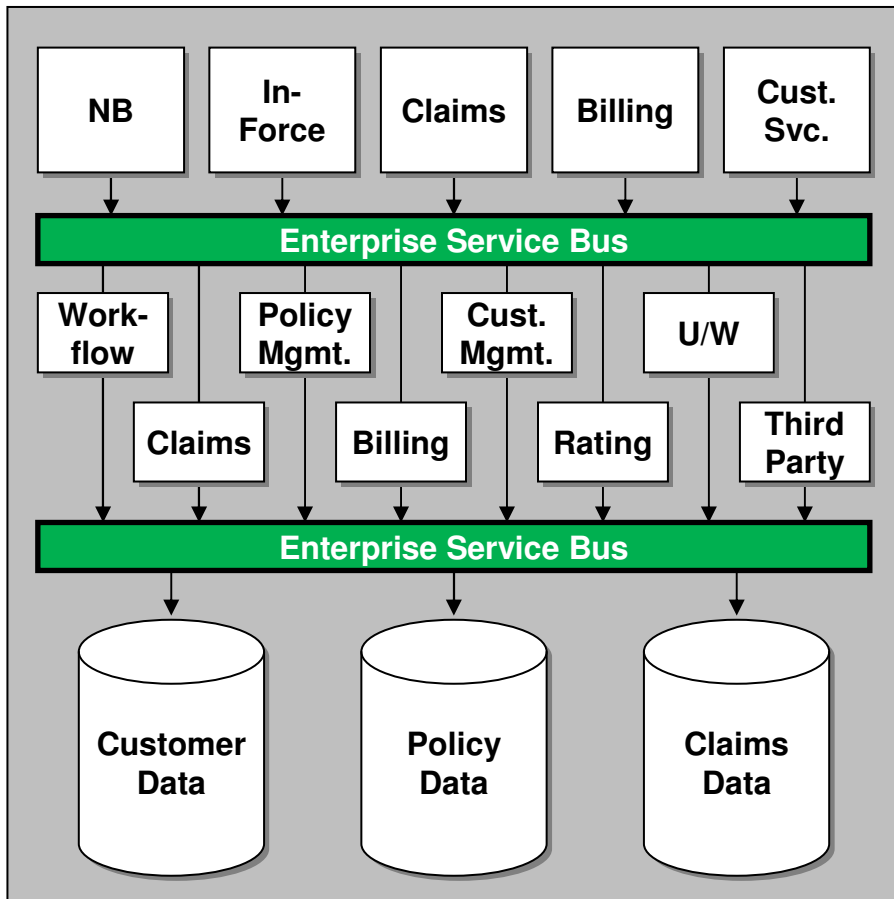
More than just workflow, BPM operates at the Application Server level to deliver robust business services that chain together individual components, back-end services, databases, legacy applications, and more—without the need to code workflow logic directly in Java or other language.



# SOA Stack – Enterprise Service Bus (ESB)

Enterprise Service Bus (ESB) is a protocol transformation gateway for enabling standard interactions between business and data services. Core functionality is centered on *process choreography, message routing, event monitoring, and data security*, allowing for a less-complex and more manageable services architecture.

*ESB Architecture*



	<i>Core Components</i>
<i>ESB Platform</i>	<ul style="list-style-type: none"> <li>• Intelligent Routing and Addressing</li> <li>• Transformation and Mediation</li> <li>• Service Registry and Publishing</li> <li>• Communication Adapters</li> </ul>
<i>Business / Workflow</i>	<ul style="list-style-type: none"> <li>• Development Tools</li> <li>• Business Modeling</li> <li>• Business Process Choreography</li> <li>• Business Activity Monitoring</li> <li>• Event Logging</li> </ul>
<i>Infrastructure</i>	<ul style="list-style-type: none"> <li>• Security</li> <li>• Environment Monitoring</li> <li>• Administration and Maintenance</li> </ul>

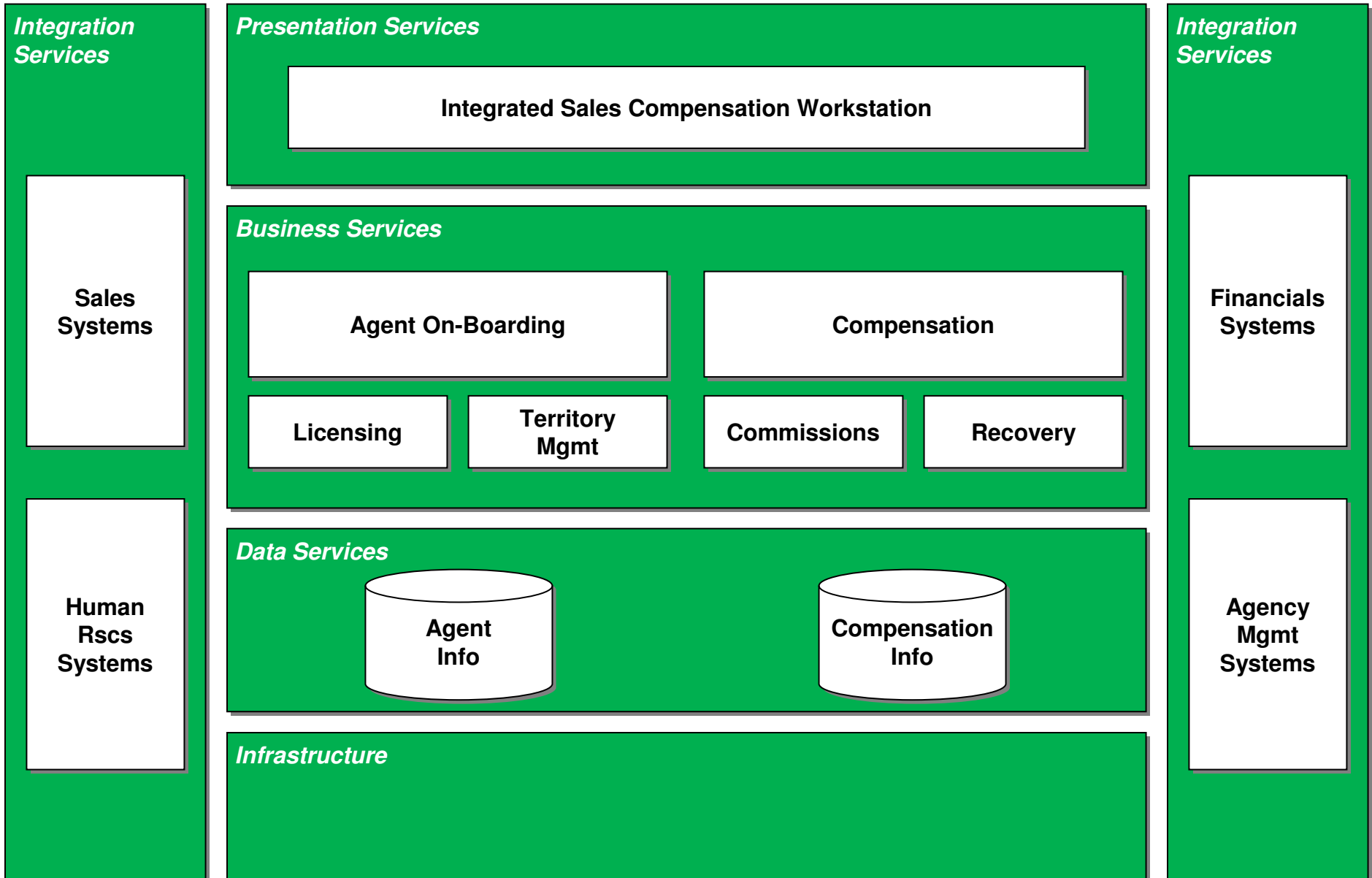
## **SOA – THREE LEARNING EXPERIENCES**

# Overview of the Three Learning Experiences

---

- **Different companies, different industries, different teams...same problems**
  - Each experience is drawn from a composite of several projects
  - Management and lead architect same across projects
  - Each system is still active today and evolving as teams gain more experience
  
- **Sales Compensation System: “SOA As Implemented By Mainframe Folks”**
  - Crude first implementation
  - Classic legacy “rip-and-replace” approach
  
- **Back Office Administration System: “Indigestion From Too Much SOA At Once”**
  - Better understanding of component and service design
  - Alignment around business functionality
  
- **Real-Time Pricing Engine: “Right Architecture, Wrong Governance”**
  - Limited only to functionality that warranted SOA
  - Integration and data flow core to service design

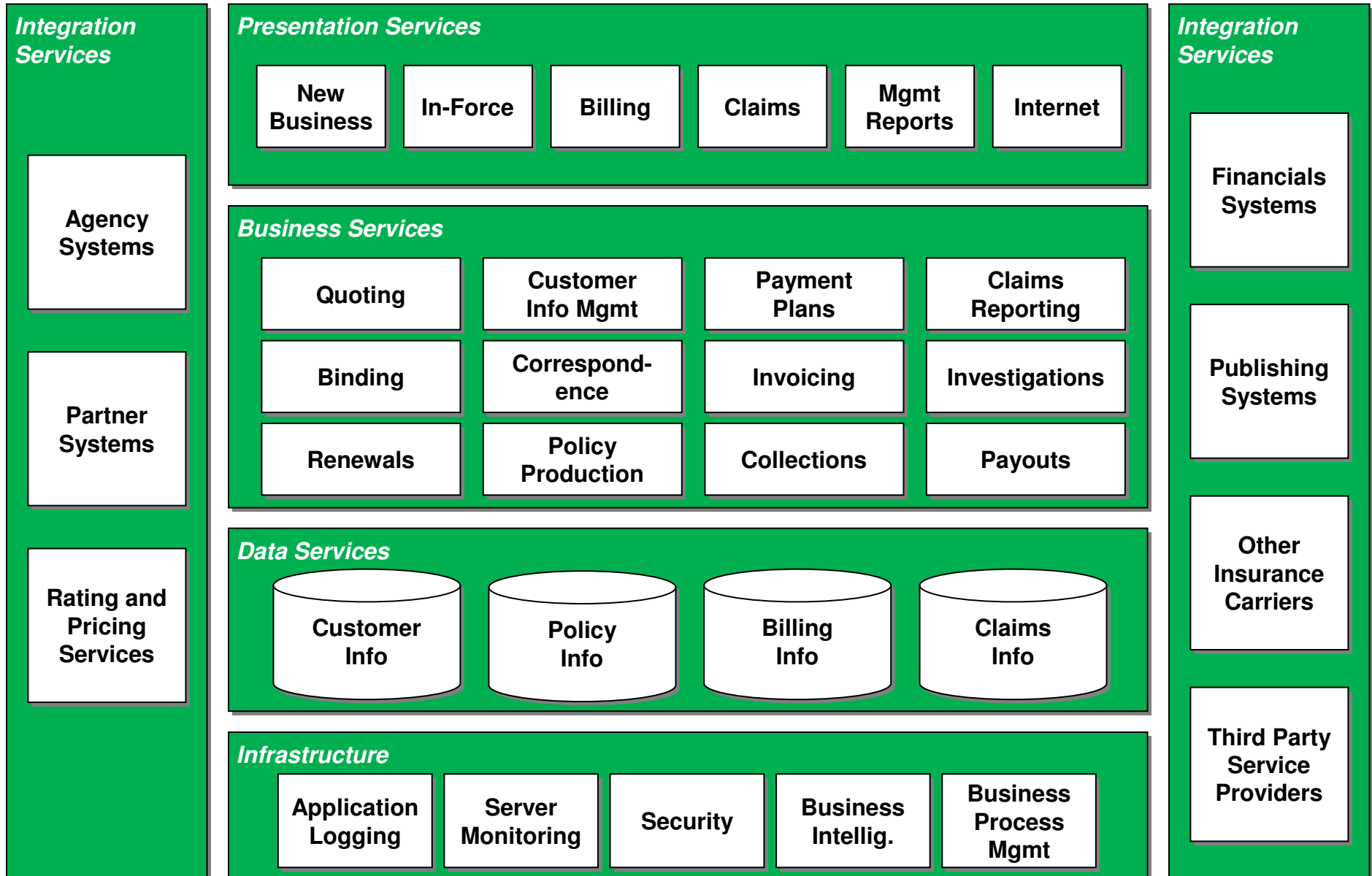
# Experience #1 – Sales Compensation System



## Experience #1 – “SOA As Implemented By Mainframe Folks”

<i>Services</i>	<i>Good</i>	<i>Bad</i>
<b><i>Presentation</i></b>	<ul style="list-style-type: none"> <li>• Well-defined business workflows led to quick user acceptance</li> </ul>	<ul style="list-style-type: none"> <li>• Classic fully-integrated, rigid interface</li> <li>• Heavy-weight client difficult to manage</li> <li>• Backlash from false configurability</li> </ul>
<b><i>Business</i></b>	<ul style="list-style-type: none"> <li>• Good modular design of <u>components</u></li> <li>• Stayed faithful to J2EE standards</li> </ul>	<ul style="list-style-type: none"> <li>• Still a closed architecture; components embedded within monolithic app</li> <li>• Services not granular, not interoperable</li> </ul>
<b><i>Data</i></b>	<ul style="list-style-type: none"> <li>• Data services isolated from app-specific functionality</li> </ul>	<ul style="list-style-type: none"> <li>• Monolithic structure led to isolated data</li> </ul>
<b><i>Integration</i></b>	<ul style="list-style-type: none"> <li>• Designed for integration with up- and down-stream apps</li> </ul>	<ul style="list-style-type: none"> <li>• No ESB; relied on point-to-point and batch interfaces</li> </ul>
<b><i>Infrastructure</i></b>	<ul style="list-style-type: none"> <li>• Vendor independent; focus on Java standard and portability; not tied to O/S or app server</li> </ul>	<ul style="list-style-type: none"> <li>• Locked into legacy standards (e.g., CORBA); unable to migrate to newer standards as they were introduced</li> </ul>

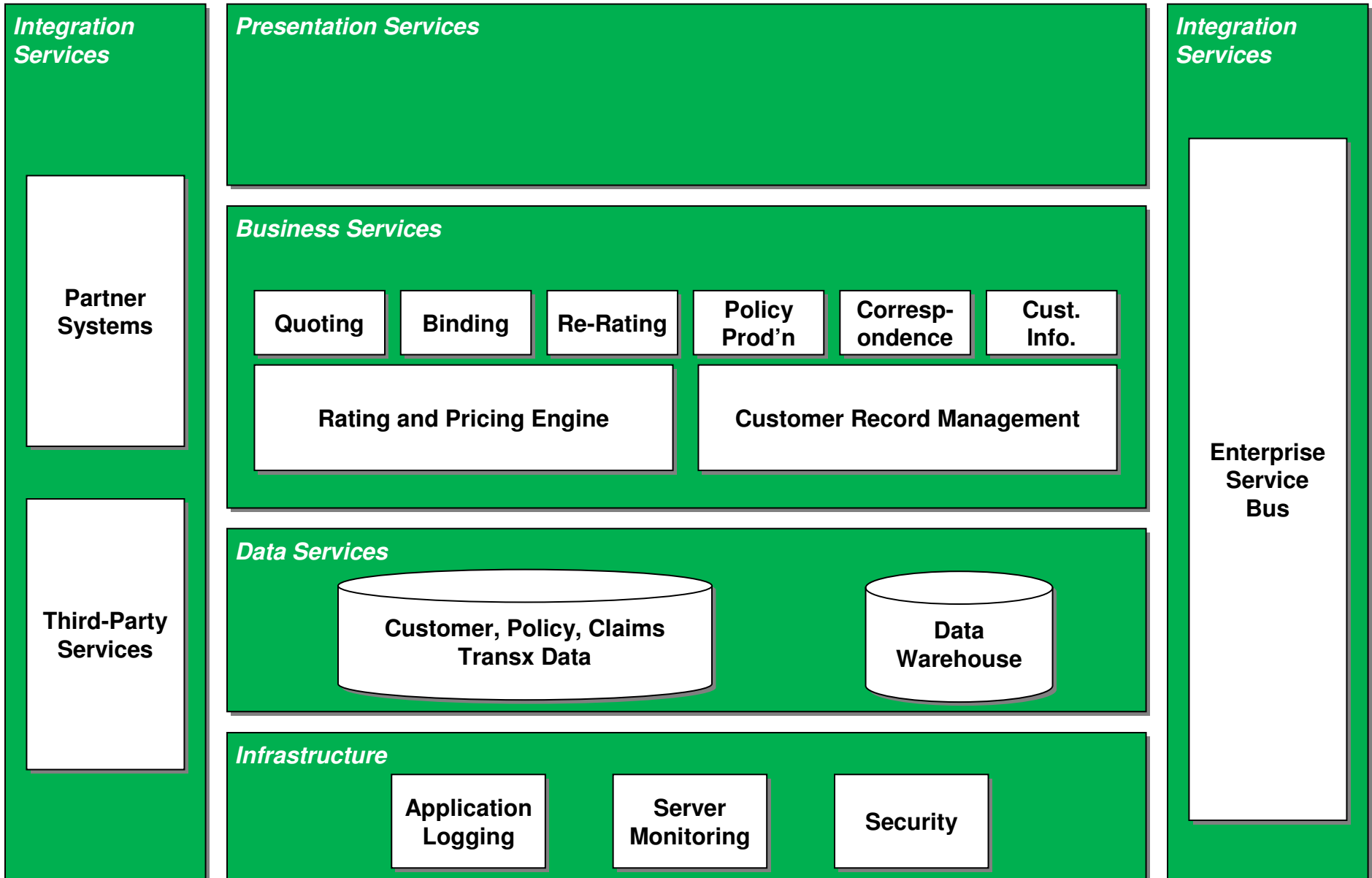
# Experience #2 – Back Office Administration System



## Experience #2 – “Indigestion From Too Much SOA At Once”

<i>Services</i>	<i>Good</i>	<i>Bad</i>
<b><i>Presentation</i></b>	<ul style="list-style-type: none"> <li>• Introduced portal capability, giving business visibility into services</li> </ul>	<ul style="list-style-type: none"> <li>• Home-brewed portal and content management framework</li> <li>• Late binding of loosely-defined services led to complex testing</li> </ul>
<b><i>Business</i></b>	<ul style="list-style-type: none"> <li>• Strong business alignment of components</li> <li>• Designed for maximum reuse</li> <li>• BPM tool for configuration of meta-services</li> <li>• Leveraged legacy mainframe code</li> </ul>	<ul style="list-style-type: none"> <li>• Too ambitious; too much SOA at once</li> <li>• Weak service governance, leading to sprawl</li> <li>• Services too granular, too loosely-define</li> </ul>
<b><i>Data</i></b>	<ul style="list-style-type: none"> <li>• Good metadata across many databases</li> <li>• Focus on efficient data access for services</li> </ul>	<ul style="list-style-type: none"> <li>• Fragmented data model difficult to manage</li> <li>• No thought given to integration with data warehouse</li> </ul>
<b><i>Integration</i></b>	<ul style="list-style-type: none"> <li>• MQ-based connectivity to host via message broker hub; easy to connect pieces</li> </ul>	<ul style="list-style-type: none"> <li>• Too many point-to-point interfaces</li> <li>• Non-standard message formats</li> </ul>
<b><i>Infrastructure</i></b>	<ul style="list-style-type: none"> <li>• Logging and tracking of business events</li> <li>• BI baked into the architecture</li> </ul>	<ul style="list-style-type: none"> <li>• Little security</li> <li>• Many basic services home-built (e.g., synch of databases)</li> </ul>

# Experience #3 – Real-Time Pricing Engine



## Experience #3 – “Right Architecture, Wrong Governance”

<i>Services</i>	<i>Good</i>	<i>Bad</i>
<b><i>Presentation</i></b>	<ul style="list-style-type: none"> <li>• No effort wasted on presentation for essentially back-end services</li> </ul>	<ul style="list-style-type: none"> <li>• No thought put into how to integrate with other presentation services</li> </ul>
<b><i>Business</i></b>	<ul style="list-style-type: none"> <li>• Strong business alignment of services</li> <li>• Right level of granularity</li> <li>• Mixed technology (C, Java, COBOL) with minimal rewrite</li> </ul>	<ul style="list-style-type: none"> <li>• No industry framework (e.g., open source, spring, struts)</li> <li>• Not designed for reuse (still app focused)</li> </ul>
<b><i>Data</i></b>	<ul style="list-style-type: none"> <li>• Single database for customer and policy data</li> <li>• Direct application access; well-tuned for transactions</li> </ul>	<ul style="list-style-type: none"> <li>• Overbuilt data model used by many apps; not abstracted from app tier</li> <li>• Not tuned for query or data mining</li> <li>• Large extracts to bolt-on data warehouse</li> </ul>
<b><i>Integration</i></b>	<ul style="list-style-type: none"> <li>• Designed to interact with variety of applications and front-ends</li> <li>• Tuned for speed (DTD-based)</li> </ul>	<ul style="list-style-type: none"> <li>• Home-brewed ESB</li> <li>• Too much app logic in the bus (so tempting for “time to market”)</li> </ul>
<b><i>Infrastructure</i></b>	<ul style="list-style-type: none"> <li>• Interfaces entirely XML</li> <li>• Fairly platform independent</li> </ul>	<ul style="list-style-type: none"> <li>• Wrote own security, monitoring, transformations</li> </ul>

# What We Learned

---

Each implementation achieved moderate success, yet under-delivered in ways that taught us where we needed to change our strategy and, more importantly, our expectations of Service Oriented Architecture.

## *Design Issues*

- **Didn't engage business early enough in design**
- **Slow to embrace industry standards**
- **Had trouble defining right level of granularity for given platform**
- **Not everything had to be a service, some could have been just components**
- **Too much focus on the service and not the underlying components**
- **Confused "repurposing" with "reuse"**

## *Management Issues*

- **Introduced SOA in the wrong order: revolutionary vs. "rip and replace"**
- **Abandoned governance when time-to-market became urgent**
- **Failed to embrace "open source" management practices**
- **Allowed business functionality to trump architecture**
- **Didn't realize that people want SOA to be more than what it is...but it is just basic n-tier architecture with ever-improving standards**

# **IMPLEMENTING SOA IN THE LEGACY ORGANIZATION**

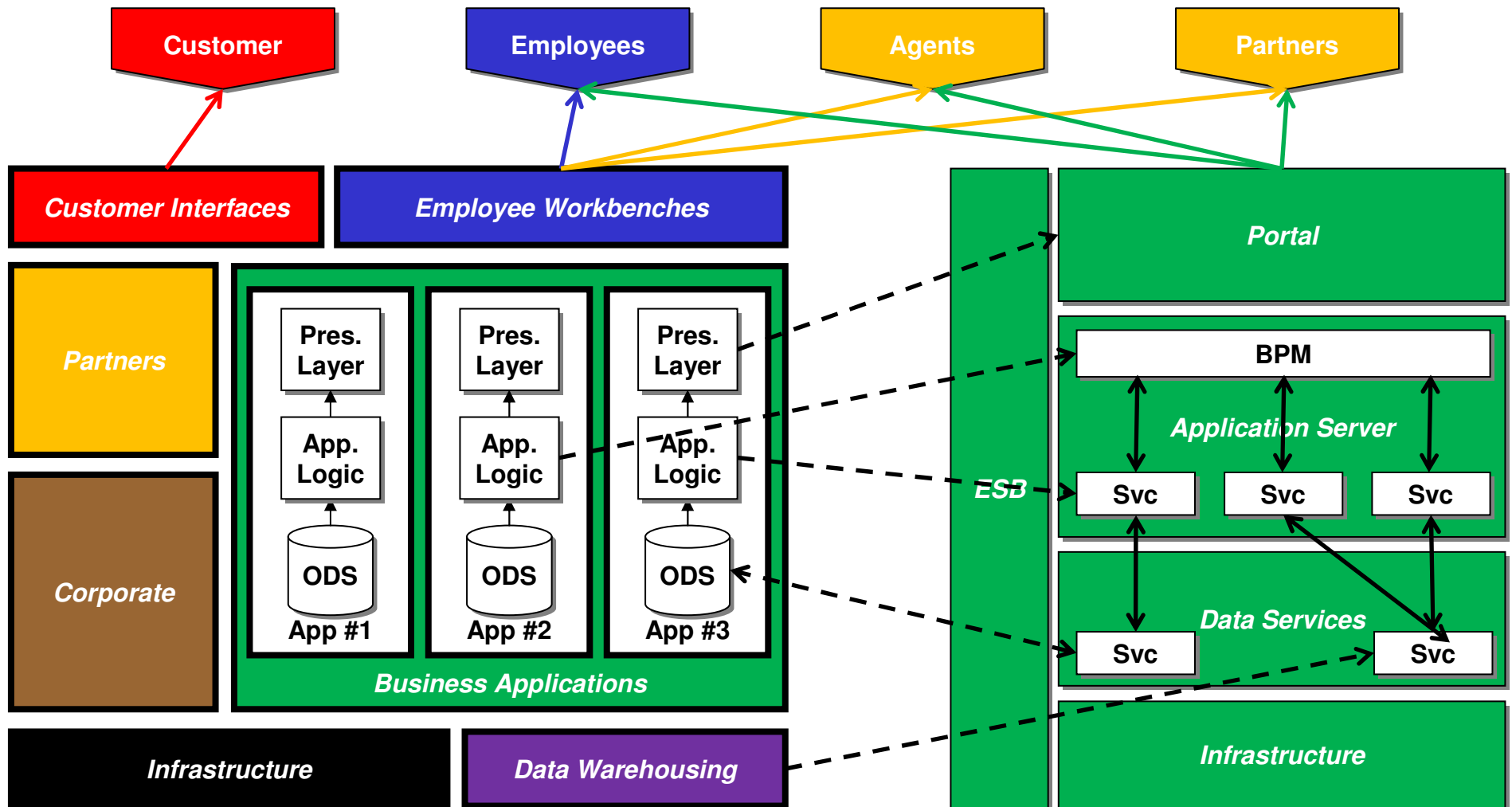
# Implementing SOA in the Legacy Organization

---

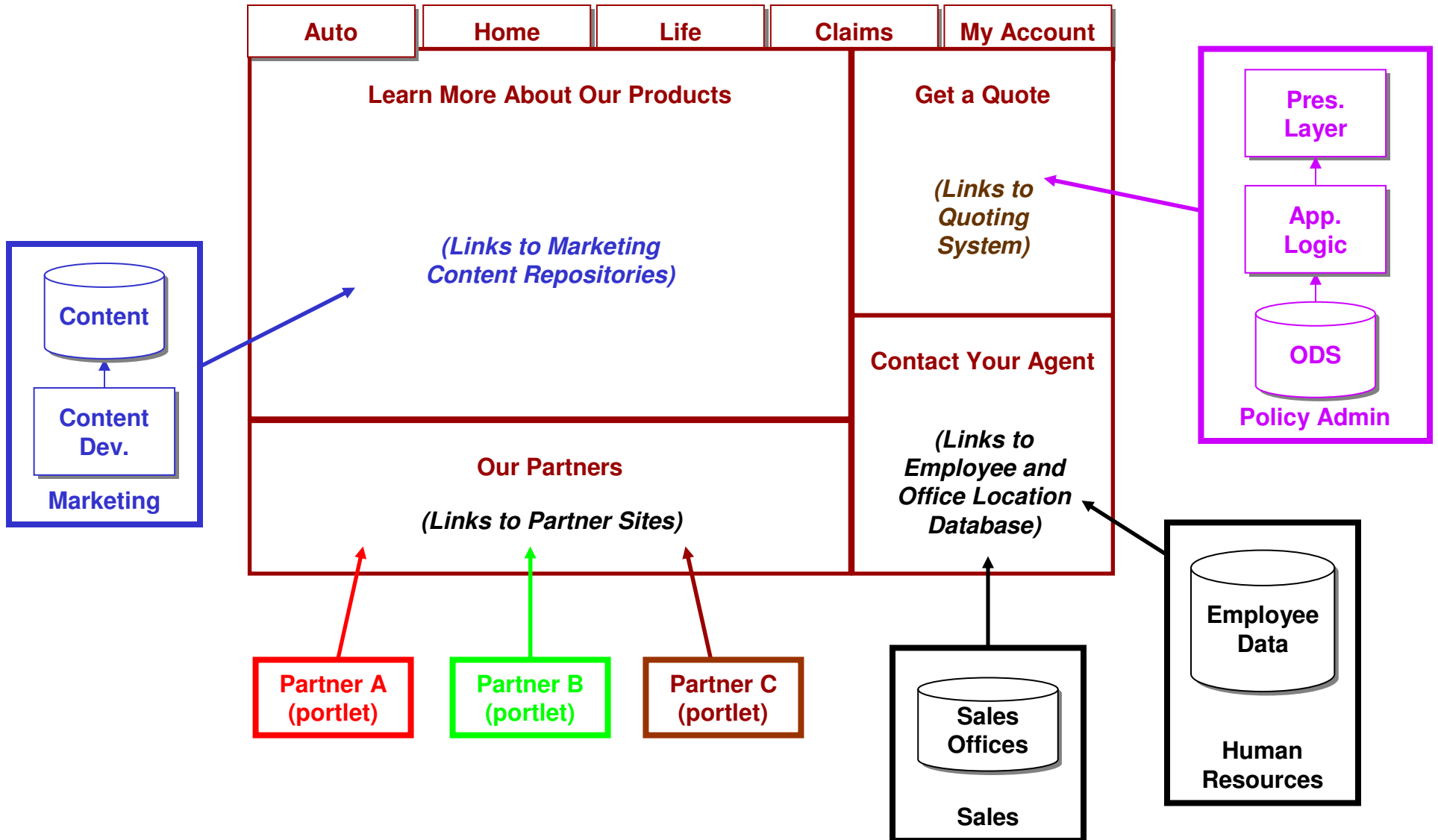
- **Design a meta-architecture to guide SOA implementation over time and across teams**
  - Must be able to support legacy code in parallel with SOA for a long time
- **Create a set of well-defined enterprise standards, and stick with them**
  - SOA requires a small, but inviolate, set of standards for all teams and all applications
  - Development teams can have flexibility on other aspects, as long as they respect the core standards
- **Build the go-forward infrastructure in parallel with current operating environment**
  - Don't overbuild the infrastructure; aim for horizontal scalability
  - Plan for a multi-year transition, and implement incrementally
  - One business process, one set of services, one technology, one organization at a time
- **Choose the right team to implement the SOA strategy**
  - Strong business knowledge and architecture
  - Strong development, testing, and respect for standards
  - SOA is strategic, interesting, impactful; people should see it as a "privilege"
- **Cherry-picking an initial set of services to be built using SOA technology**
  - Use Portal to build an enterprise "gateway"
  - Use ESB to build a set of efficient data services
  - Use BPM to begin migrating high-priority business services to SOA

# Building the “Go Forward” Infrastructure in Parallel

Like building new Yankee Stadium in the lot next to the old one. Implement a robust, horizontally-scalable platform in parallel, then migrate business processes, services, technologies, applications one-by-one. Key is to make the infrastructure *enterprise-class*. ..you’re starting with a clean slate, so don’t skimp on getting it right.



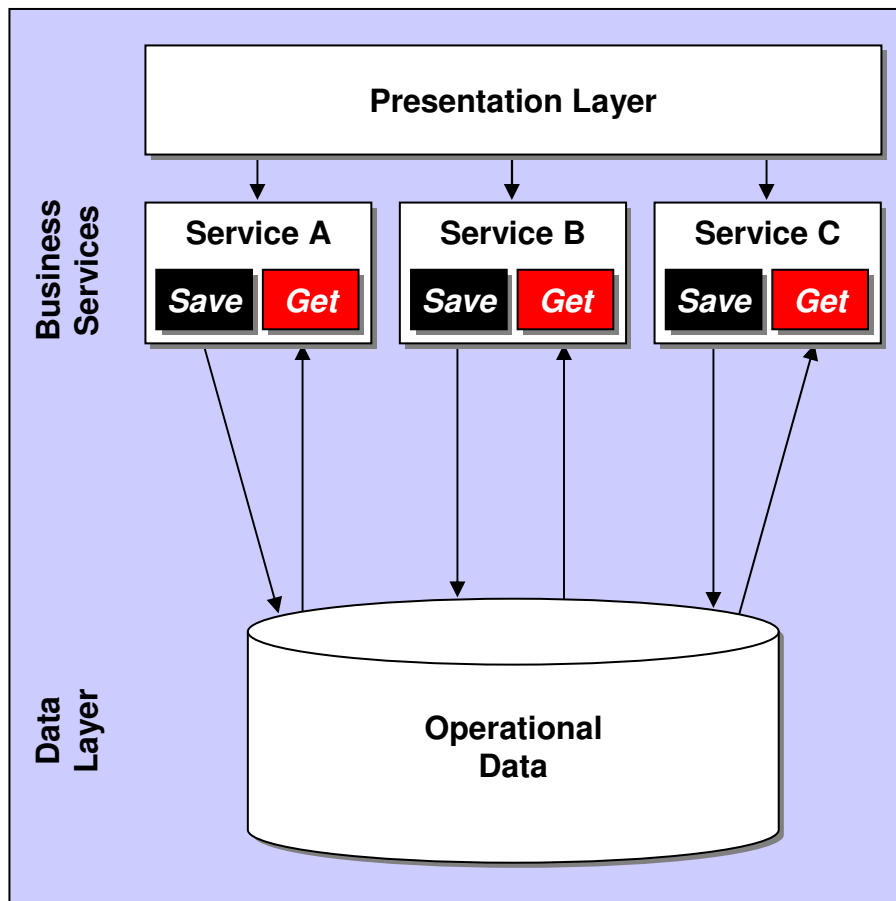
# Build a "Portal Gateway" for Presenting Your Services



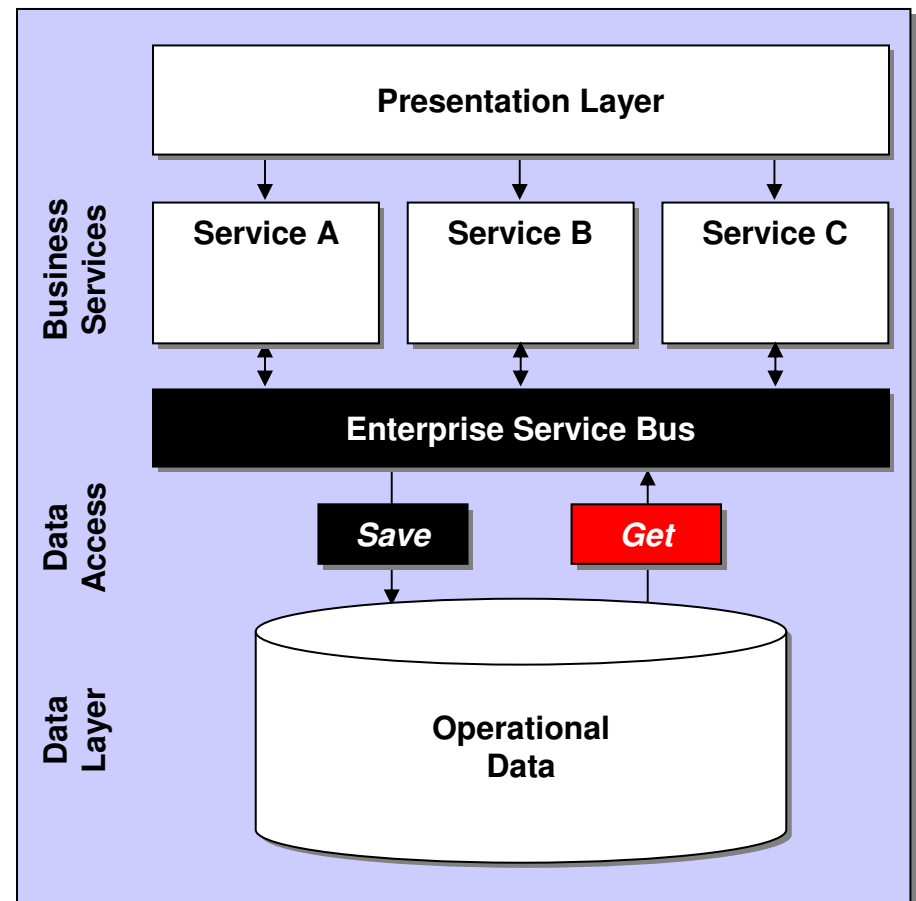
# Use ESB to Get Control Over Data Management

Trends toward *SOA* and *Distributed Development* are in conflict with efforts to get data management under control. Centralizing core services through ESB forces developers to use enterprise I/O standards, and to think more in terms of flow across services, encouraging them to separate the “get”, “process”, and “save” aspects of their applications.

*Fragmented Data Access Services*



*Centralized Data Services Using ESB*



# **SUMMARY**

# Summary

---

- **Engage the business early and often**
  - Always maintain process point-of-view when designing services
- **Don't skimp on infrastructure – Build a robust platform to handle growth**
  - Horizontal scalability – add capacity as you grow
  - Respect “The Stack”
  - Include security, monitoring, and environment management
- **Plan for heterogeneity of applications, technologies, processes**
  - SOA is the glue to bring together disparate legacy systems
  - “Live and let live” if systems don't need migration
  - “Cut bait and migrate” when rewrite is the best solution
- **Small and incremental – Migrate functionality one step at a time**
  - SOA is not a “Big Bang” initiative
- **Success = Business Focus Plus Governance**
  - Stay the course, maintain standards, work through design and org. issues
  - If not, SOA will end up just another competing legacy technology

***Jump in with both feet; there is no perfect project to start with; SOA is a discipline, not a killer app, and good enough is good enough***

# **SERVICE ORIENTED ARCHITECTURE AND THE LEGACY ORGANIZATION**

David Koenig, Brookline Technology  
Michael Galarneau, Liberty Mutual

May 21, 2007