

Designing e-Business Systems

Using a loosely coupled adaptive architecture methodology

By **Joe Ellsworth** - joe@pybiz.com (408) 364-1741



CEO & Lead Architect

PyBiz, Inc.

(Silicon Valley)

Our Focus


Applying advanced Internet techniques that make it easier to cope with rapid change.

Our Mission

Enable rapidly evolving & collaboration centric businesses to manage the impact of diversity and business changes on IT infrastructure.

Techniques stressed during this Class

- **Open / Cross Platform Techniques**
 - http, XML, XSL, etc
- **Designing for Scale**
- **XSLT to isolate Business functionality**
- **Design field customization in.**
- **Think in Evolution from the beginning.**



My History

Why I am qualified to comment

- XDisect & XDisect Forms
 - E-Speak Broker
 - Dynamic Security Cells
 - ESN – Enterprise Class Extranet Portal
 - QOA – Web Configuration.
 - Watson Server
 - RPG to C++ translator.
 - Distributed workman's compensation System.
 - Municipal Billing Systems
 - Books Store POS
 - 3d robotic control
- History starting in 1982 software Development.
 - 5 years self employed building fixed price solutions.
6 years as a preiminant Object oriented Architect Consultant.
 - 6 Years as manager and thought leader at H.P.
 - CEO – PyBiz, Incorporated.



Keywords & Audience

■ Watch Words

- Internet E-Business
- Enterprise Scalability
- Mission Critical
- E-Commerce
- Dynamic Provisioning
- XML / XSL / XSLT
- B2B Architecture
- Loosely-Coupled Methodology
- Evolution
- Rapid Change
- Partnership oriented portals

■ Who Should Attend

- Technologists responsible for delivering & maintaining next generation mission critical e-business
- IT Development Managers & Senior architects responsible for deploying enterprise class e-business applications.

This course is semi-technical and hence does not require a deep programming background. It does require a background in designing e-business applications

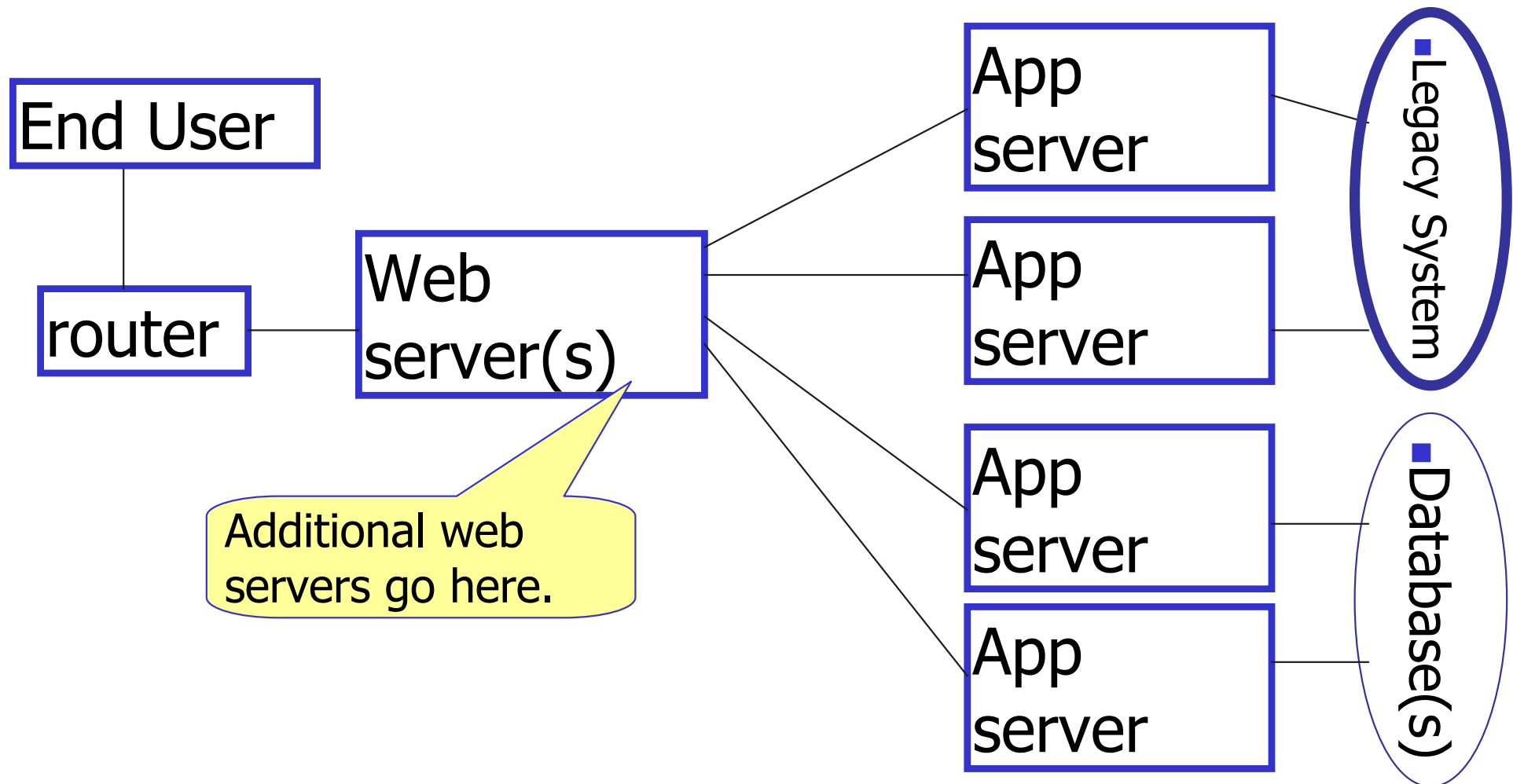


Goals - What you will Learn

- ⑩ Problems with current generation e-business applications & systems
- ⑩ General technology architecture concepts
- ⑩ Issues to keep in mind when designing next generation e-business applications.
- ⑩ Designing for change using loosely coupled adaptive architectures
- ⑩ Designing e-business applications that allow independent evolution
- ⑩ How to integrate your e-business application with other systems at low cost
- ⑩ How to integrate with other systems with minimum disruption to the systems involved

An Example E-Business System

- Assuming the heavy domain processing is done on app servers.





Today's Environment

- Rapid Change
- Partnership Driven
- Monolithic systems are too slow.
- Centralized planning is too slow.
- Lots of external integrators Needed.
- Independent Evolution is occurring in parallel.
- Global Shift
- Lots of systems working together collaboratively to deliver value to the end user.
- Requirements change every 3 months.
- No single application provides everything.
- Best of breed integration is mandatory.

Architectural principles



Rules of the Road

- Deliver early value and improve rapidly
- Design for Change
 - Business requirements are changing faster now than ever.
 - Strong partnering requires special focus on change.
 - A common CEO complaint is that IT infrastructure is preventing effective decisions.
 - Design to allow horizontal scaling
- Use Loosely coupled tactics
- Design for Horizontal Scaling



General Principles

■ NO BIG BANG

- Technology is driven by fads. You have to follow them but do not become dependant on them.
- Adapt a series of working systems and incrementally improve them over time.
- Major Releases every three months.

- Use open / vendor agnostic integration protocols.
- Force engineers to design for the boundary.
- Remember WAN latency is 100 to 500 times slower than LAN.
- Design for Integration
- Loosely Couple
- Build to facilitate Adapting
- A lot of value is still in old legacy systems so designs much accommodate them in their world view.



Architectural problems faced by current generation e-business apps

- Huge Scale with Widely fluctuating loads
- Lots more integration.
- Rapidly Changing Requirements.
- FAD based decision making.
- Lots of partnerships
- Users are demanding more capability.
- Need to extract more value.
- 100% uptime is needed.
- Vendors with Monolithic designs who claim to be open.
- World wide deployment slow & unreliable internet network.

Company Document Management Life Cycle

Growth towards Complexity

1 Documents stored on local PC hard disk.

2 Common Document Share

3 Version & Change Control..

4 Simple Content tools to make some documents available on Web.

5 Hire external consultants as ability of single user simple tool exceeded.

6 **Use eContent Mgr for distributed authorship and advanced capabilities**

7 Invest over 300K and 3 months for average Broad vision or Documentum



Changing Business Requirements

```
<person>
  <sex>female</sex>
  <name>
    <first>Wendy</first>
    <last>Dodger</last>
  </name>
  <email>wendy_dodger@hotmail.com</email>
  <email>wendy@coolguys.com</email>
  <skill>cobol</skill>
  <skill>accounting</skill>
  <skill>java X C++</skill>
  <meta>
    <uri>people/wend_dodger.xml</uri>
    <expires>2001-09-23</expires>
    <id>wdodger</id>
  </meta>
</person>
```

```
<person id="cpatel">
  <sex>male</sex>
  <name>
    <first>Chetan</first>
    <last>Patel</last>
  </name>
  <rate>000249.43</rate>
  <id>493596</id>
  <contact>
    <email>chetan_patel@hotmail.com</email>
    <email>chean@coolguys.com</email>
    <phone>
      <home>408-733-8463</home></phone>
  </contact>
  <skill>
    <business>
      <networking>business</networking>
    </business>
    <technical>
      <cef>collaborative admin for web sites</cef>
      <lang>java</lang>
      <lang>c</lang>
    </technical>
    <hobby>scuba</>
    <certified id="cpr"/>
  </skill>
  <skill>talking</skill>
</person>
```



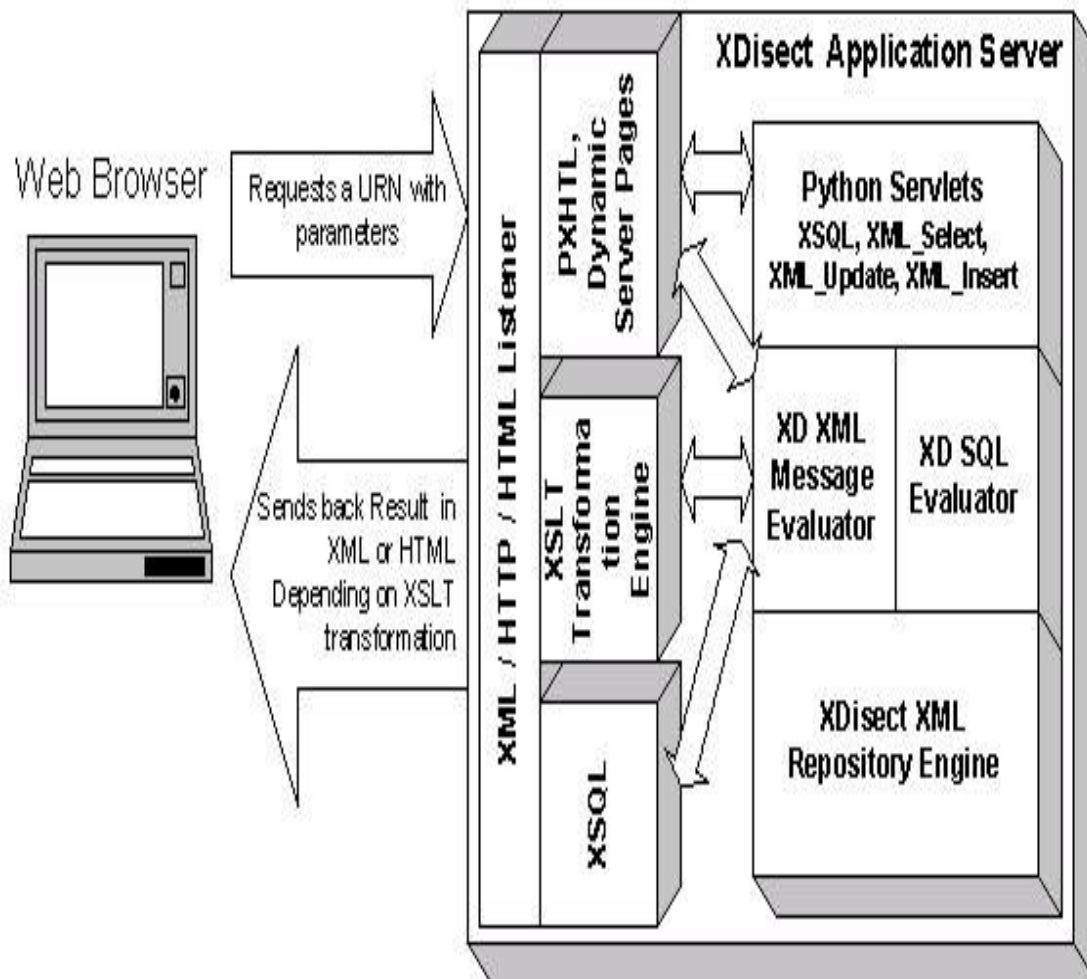
Design Strategies

CGI, XSL/XSLT, N-Tier

- See.: http://www.xdfind.com/read_me/concept_overview.html
- See.: http://www.pybiz.com/products/xdisect/xd_technical_architecture.pdf

XSL Architectural View

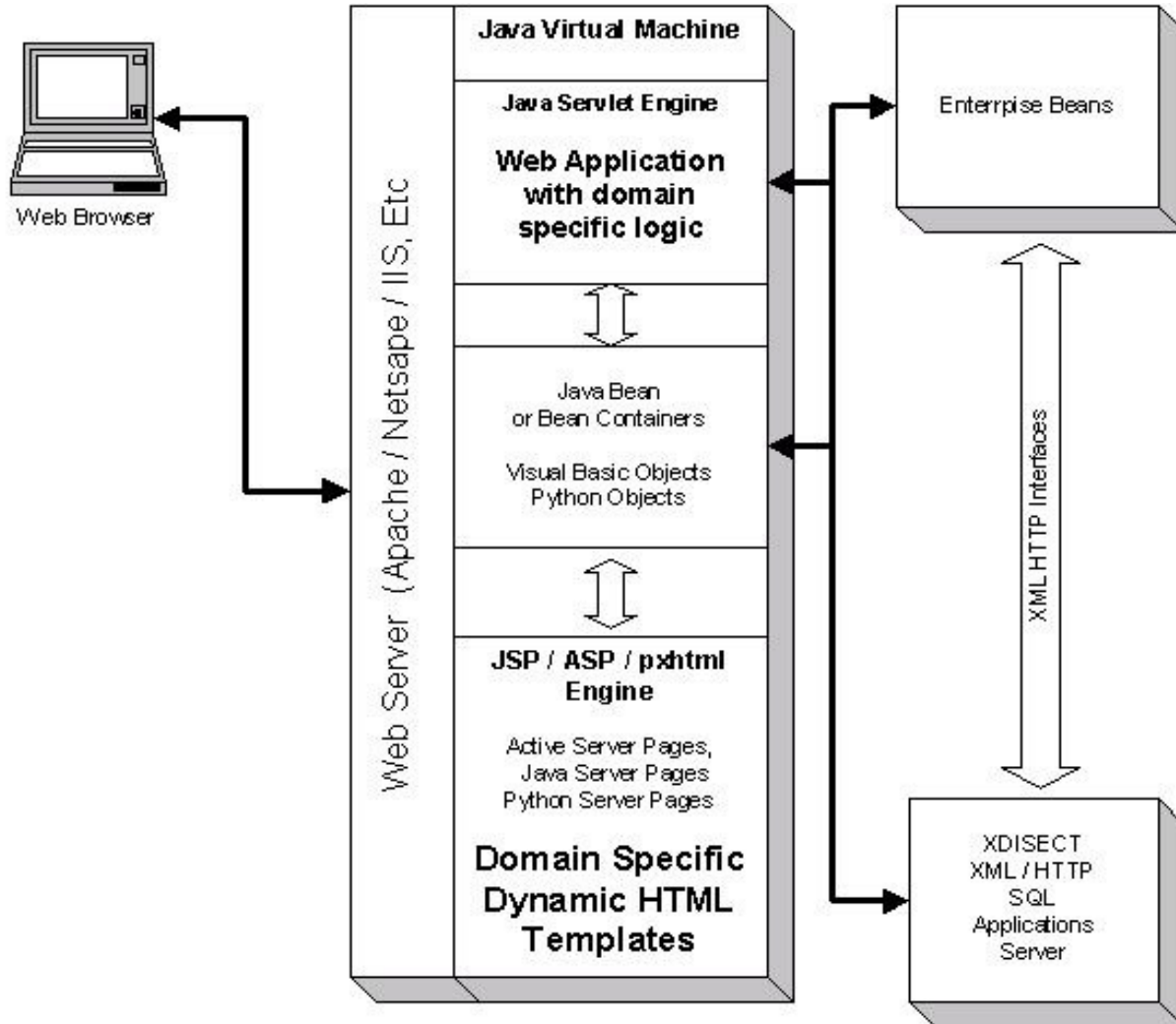
XSQL & XSLT



- XSQL – Server side stored XML queries. A standard published by Oracle.
- Use XSQL to query XDisect.
- Use XSLT to transform query results into different XML or html formats.
- On the front-end an application server like BlueStone or BEA-WebLogic is used rather than direct connection to the browser.
- For security a reverse proxy between public internet & XDisect application server is recommended.

JSP/J2EE Architectural View

XDisect can be accessed from JSP & EJ beans



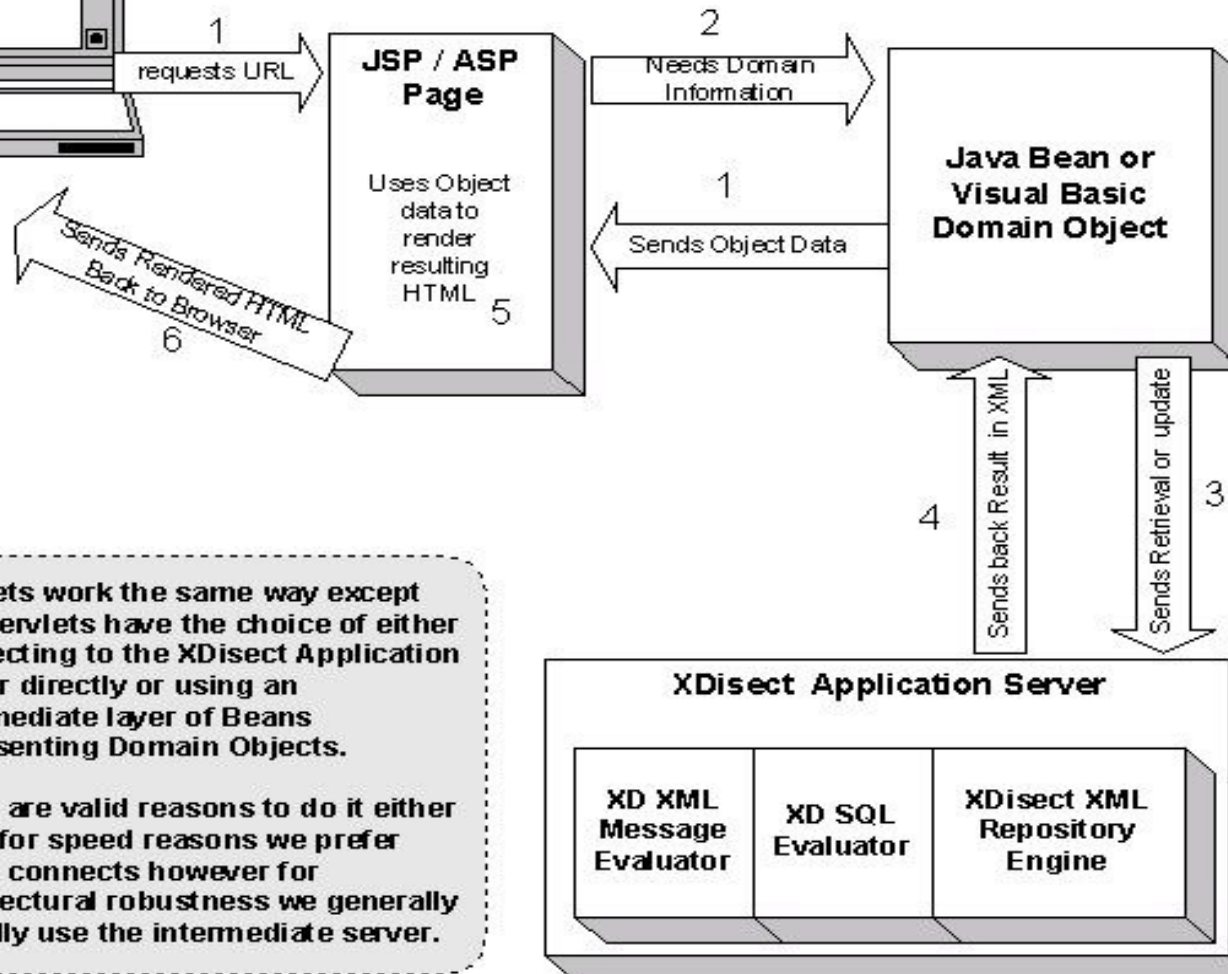
- XDisect can be accessed from any J2EE platform
- Open XML/HTTP protocol
- The front-end JSP & beans can also access XDisect.
- The app server view of this is more complicated but most JSP&EJB components will flawlessly move into the app server platform.
- Some advanced app server capabilities are not adequately utilized in this simplified view.

Traditional EJB Message Flow

Web Browser



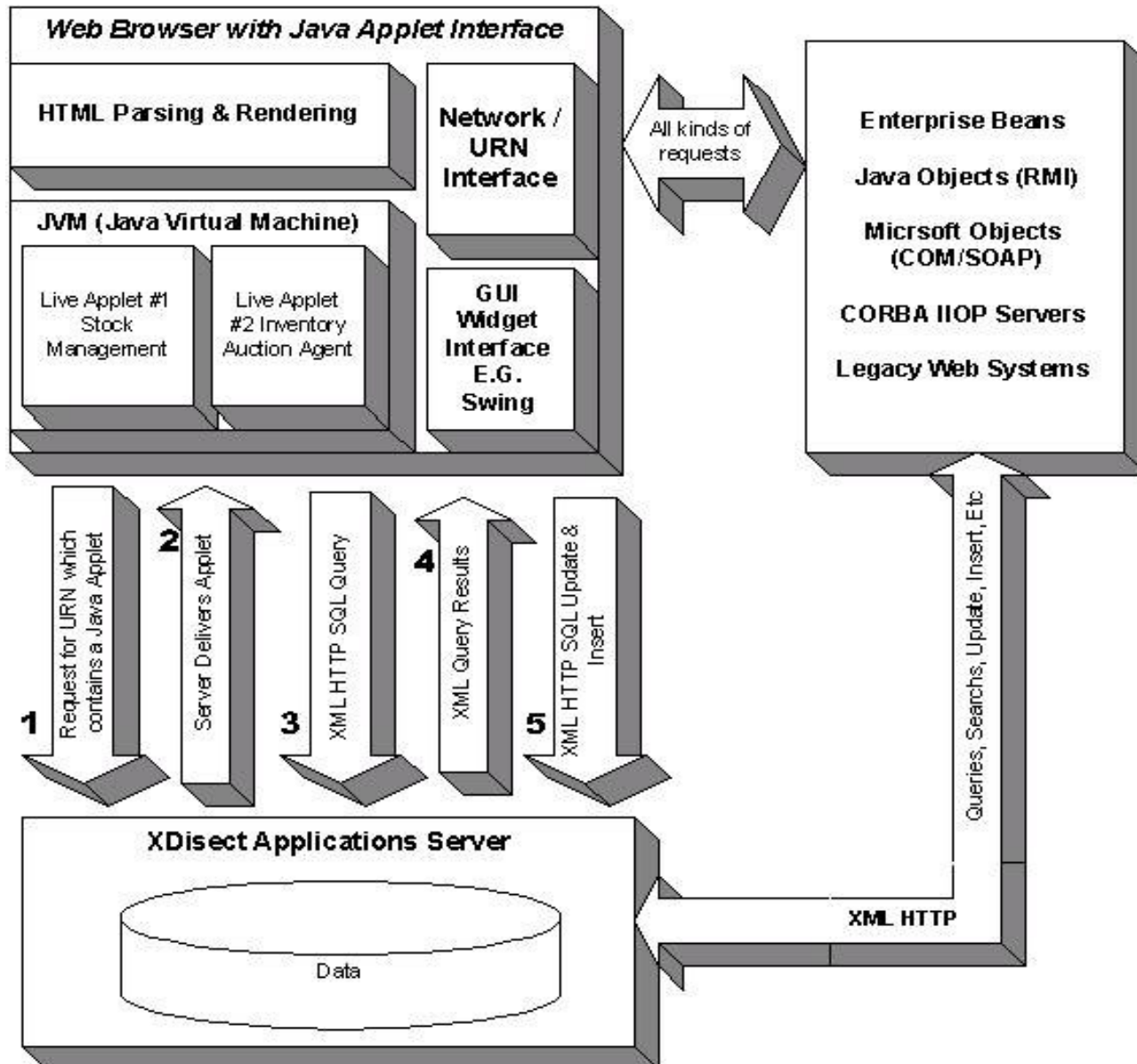
Message Flow with JSP



Servlets work the same way except that servlets have the choice of either connecting to the XDisect Application Server directly or using an intermediate layer of Beans representing Domain Objects.

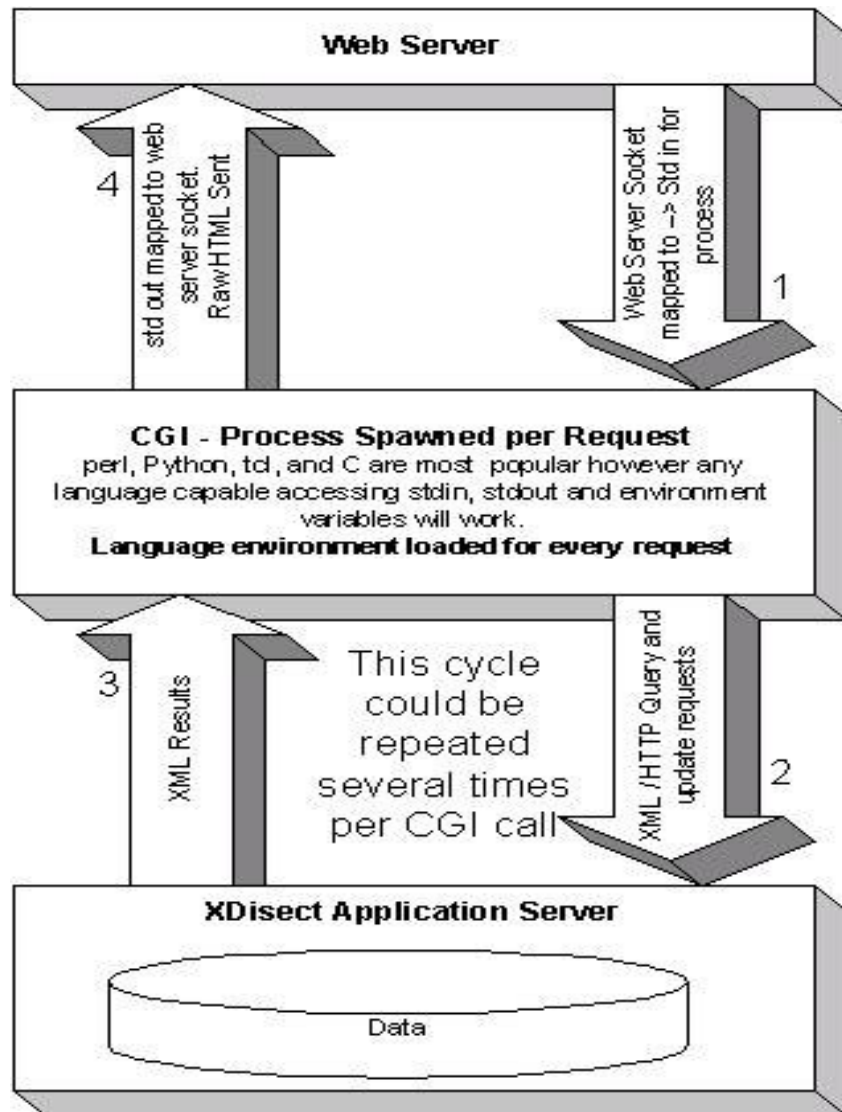
There are valid reasons to do it either way. for speed reasons we prefer direct connects however for architectural robustness we generally actually use the intermediate server.

Java Applet & Intelligent Clients



- Java applets can connect to remote URL's as long as they are on the same server from which they were served. They can make IIOP, RMI and EJB calls to access enterprise resources. This flexibility makes them ideal to interact directly with the XDisect engine using the XML / HTTP interface.

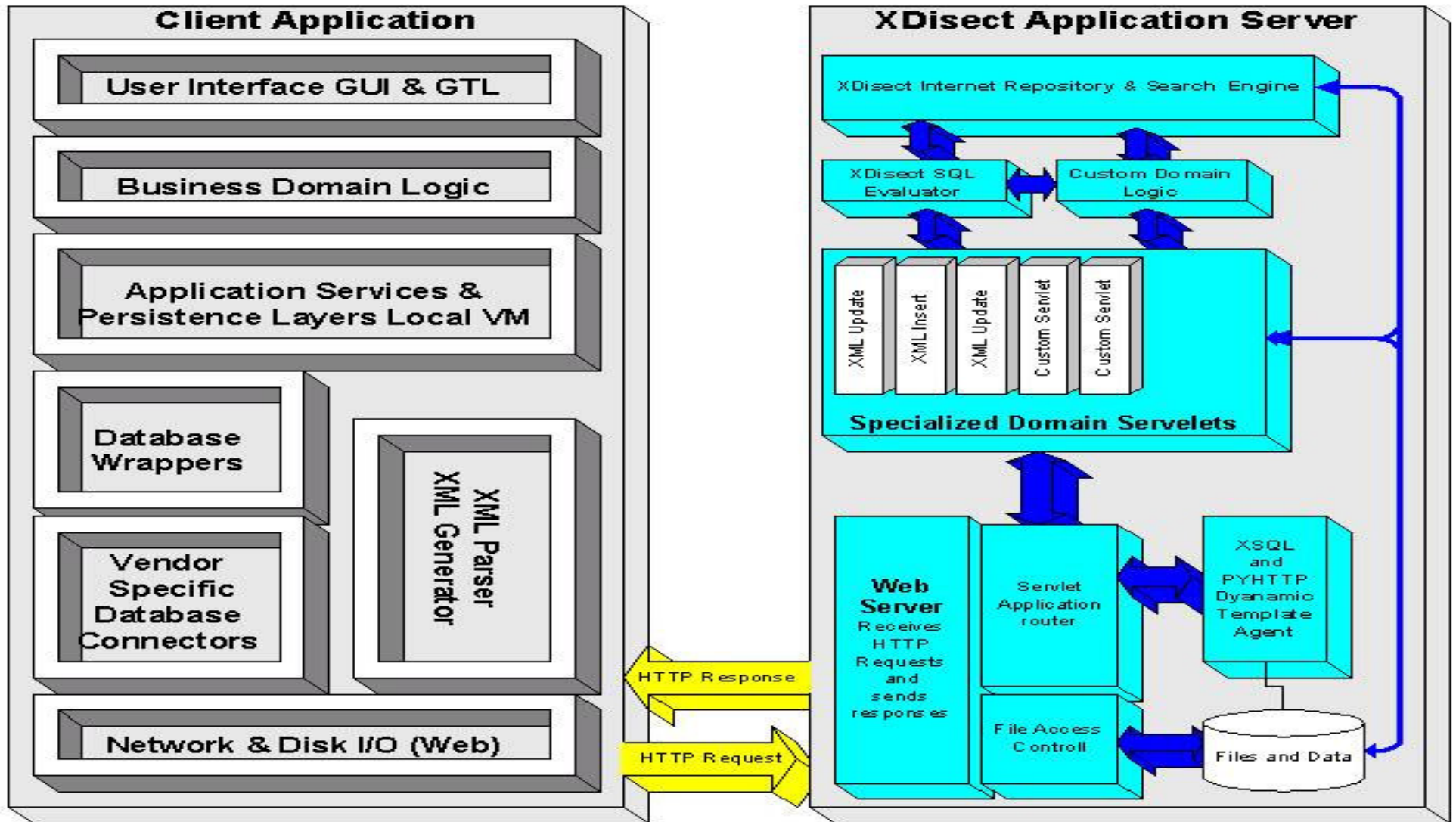
CGI based Approach



- **CGI remains popular because of its resilience.**
 - If the application script request crashes, it only takes down that one process and leaves the web server in a stable state.
 - This can make it easier to ensure that most users get correct behavior even when there may be a bug or two in the software.

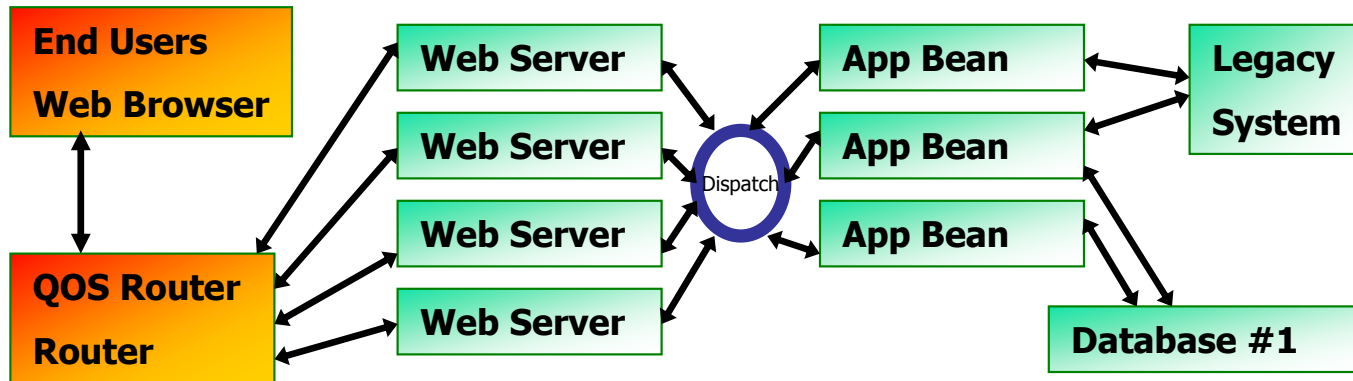
Detailed E-Biz Application Component Breakdown

Basic Client & XDisect Component View



General N tiered architecture design (Software)

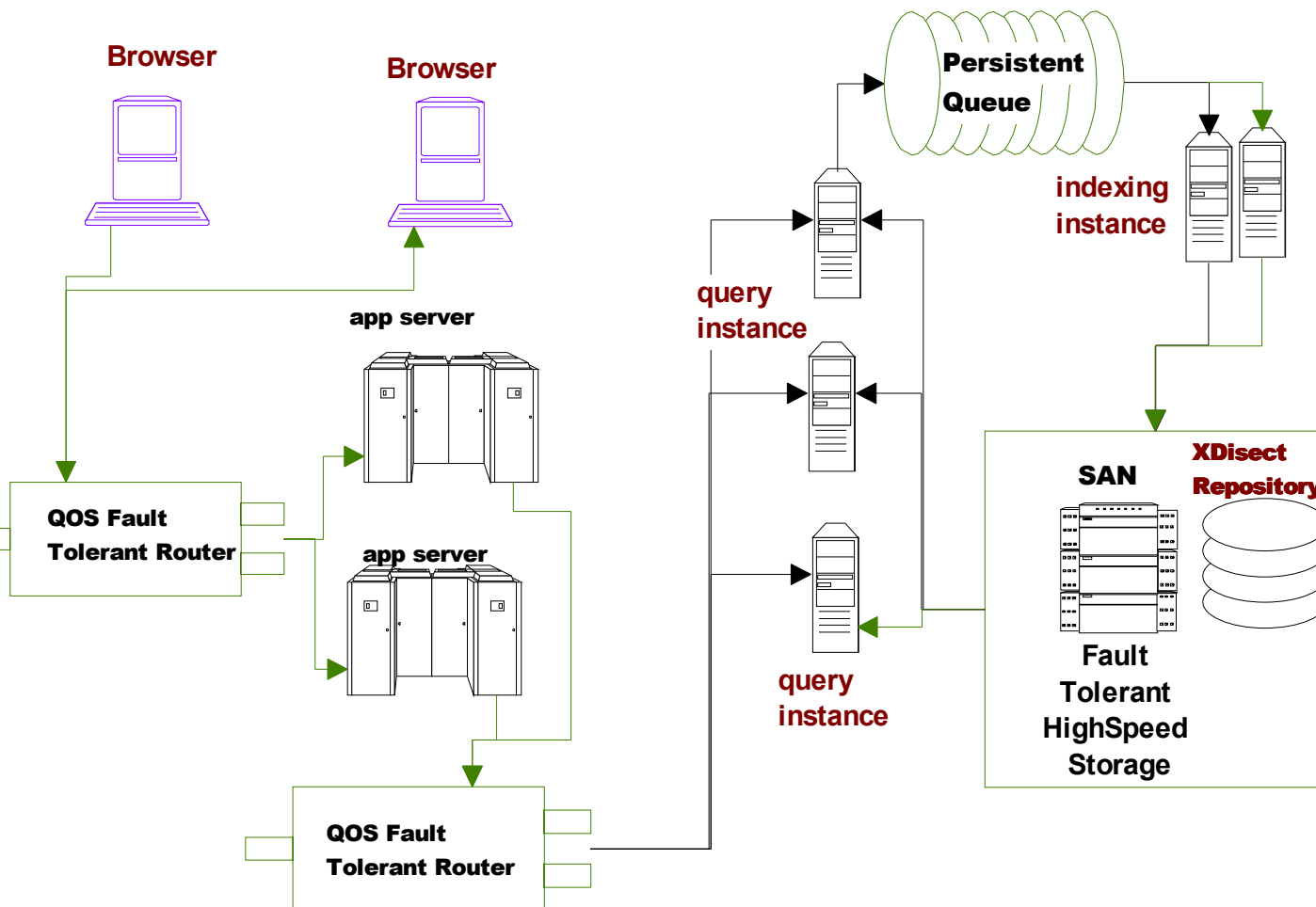
3 Tier Software Architecture



- App Object Instances "Beans" are any 2nd tier application server such as EJB, COM, XDisect, Etc.
- There is commonly a broker that sits between the web server and app bean. It provides fault tolerance and load balancing.
- In newer systems a Persistent Queue is used to add robustness.

Fault Tolerance

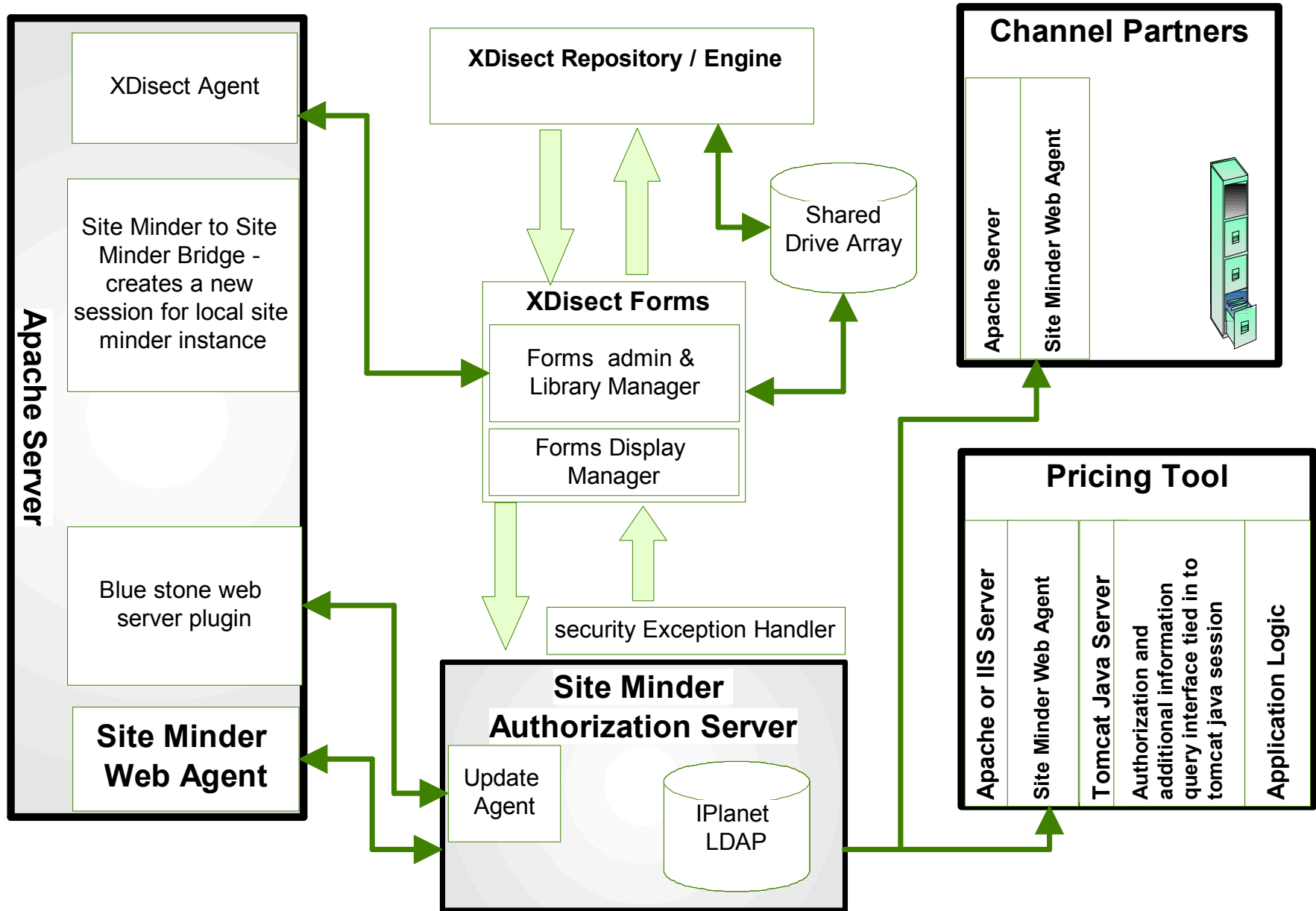
utilizes standard QOS Routers



- Any query instance can receive requests for insert/update
- Requests are stored in a persistently queue.
- No single point of failure in the architecture.
- Total failure would require failure of all components
- Second router functionality can be done using software but we prefer the h/w router for performance.
- Only one instance of the XDisect Indexer is active at a time.

Prototype Software Architecture

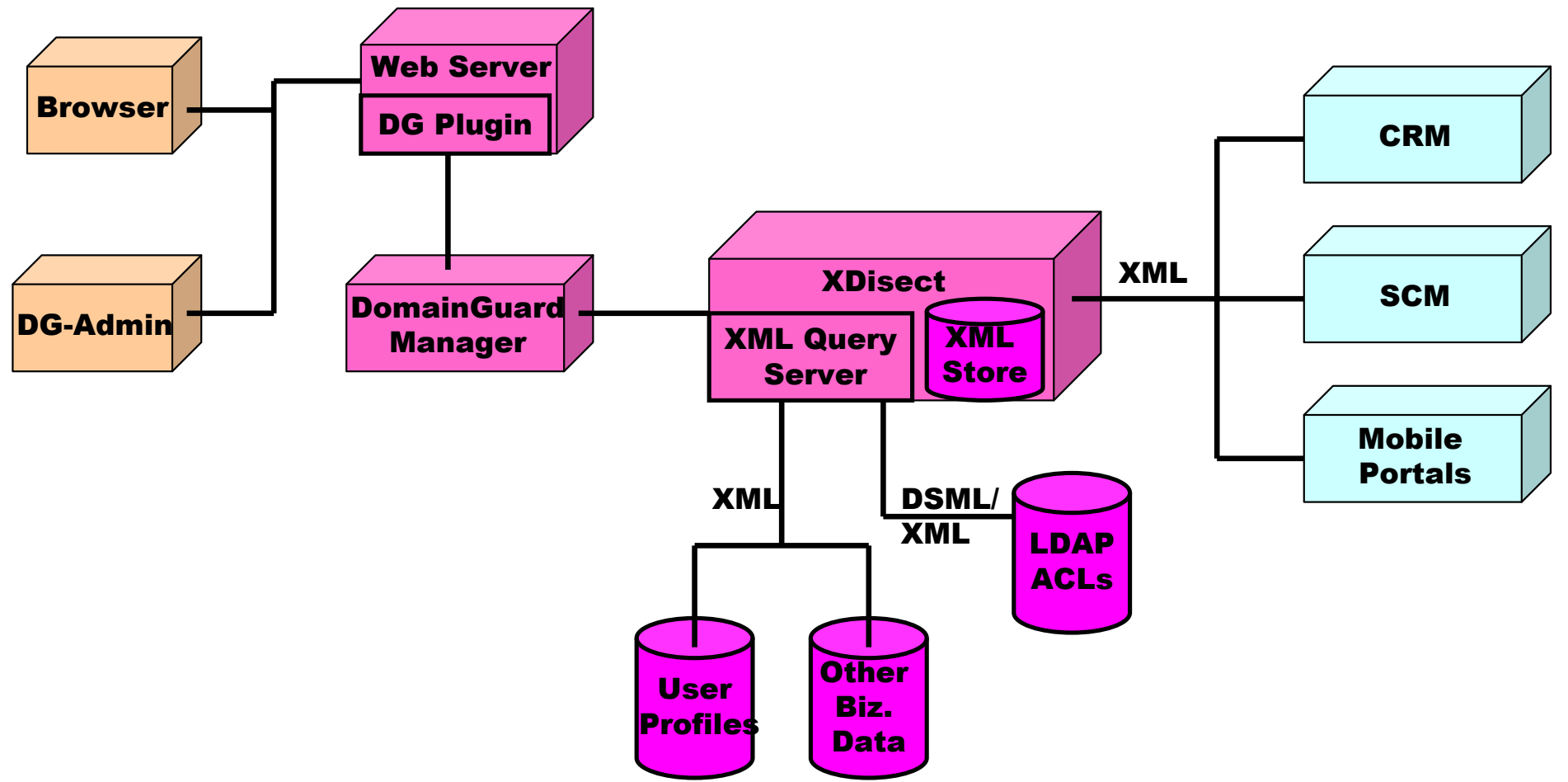
Site Minder Based



Domain Guard enabled Proxy

Clients

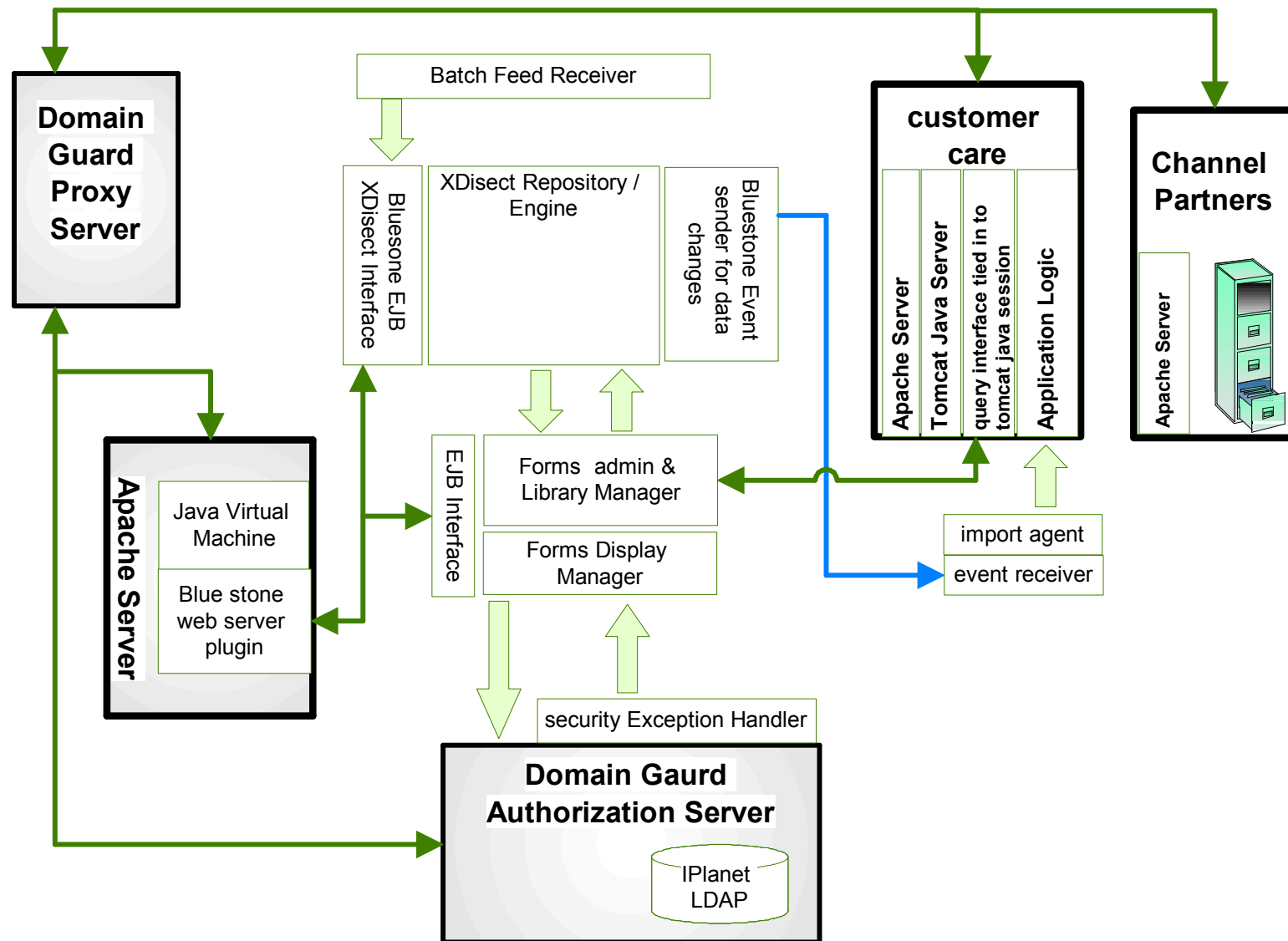
Corporate Business Apps



Corporate Databases

Prototype Software Architecture

Domain Guard Proxy Based

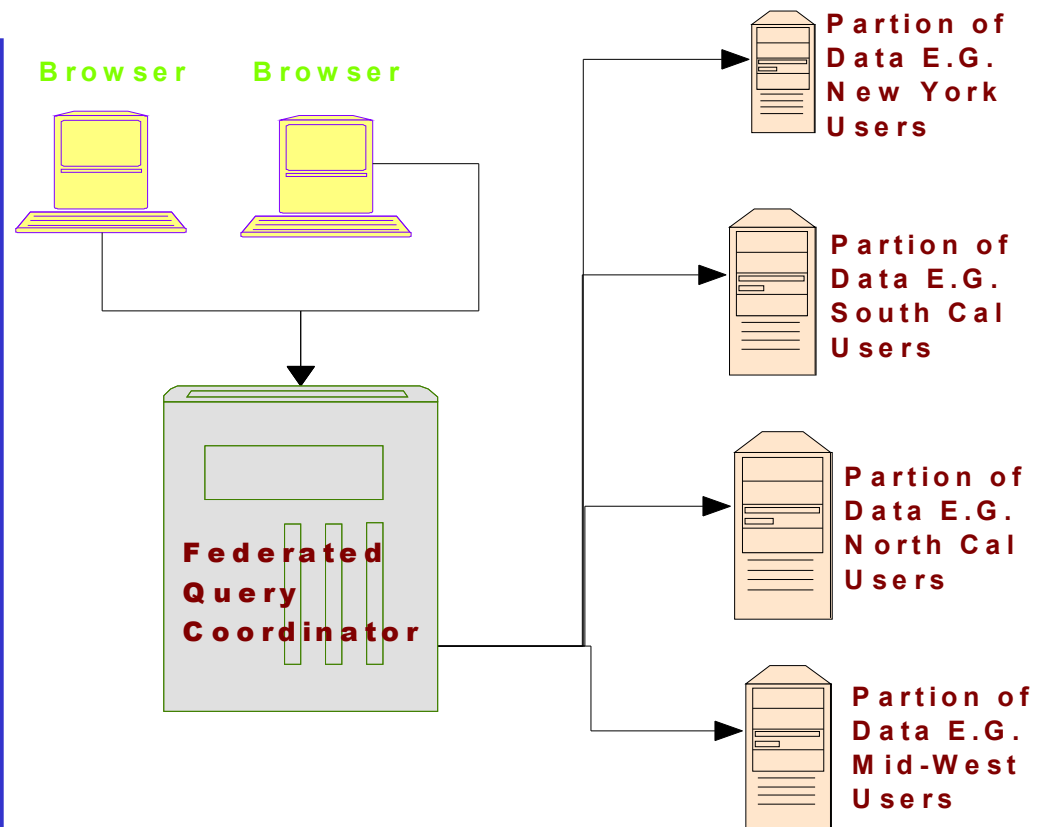


Designing Data Partitions for Scale

Federated query management

Federated Query Partitions Data for scaling optimal Query Response

- Each partition can store different data.
- Each partition internally may have load balanced QOS infrastructure.
- Coordinator aggregates results from different partitions.
- Coordinator manages paging for clients.
- Partitions do not need to be XDisect Based.

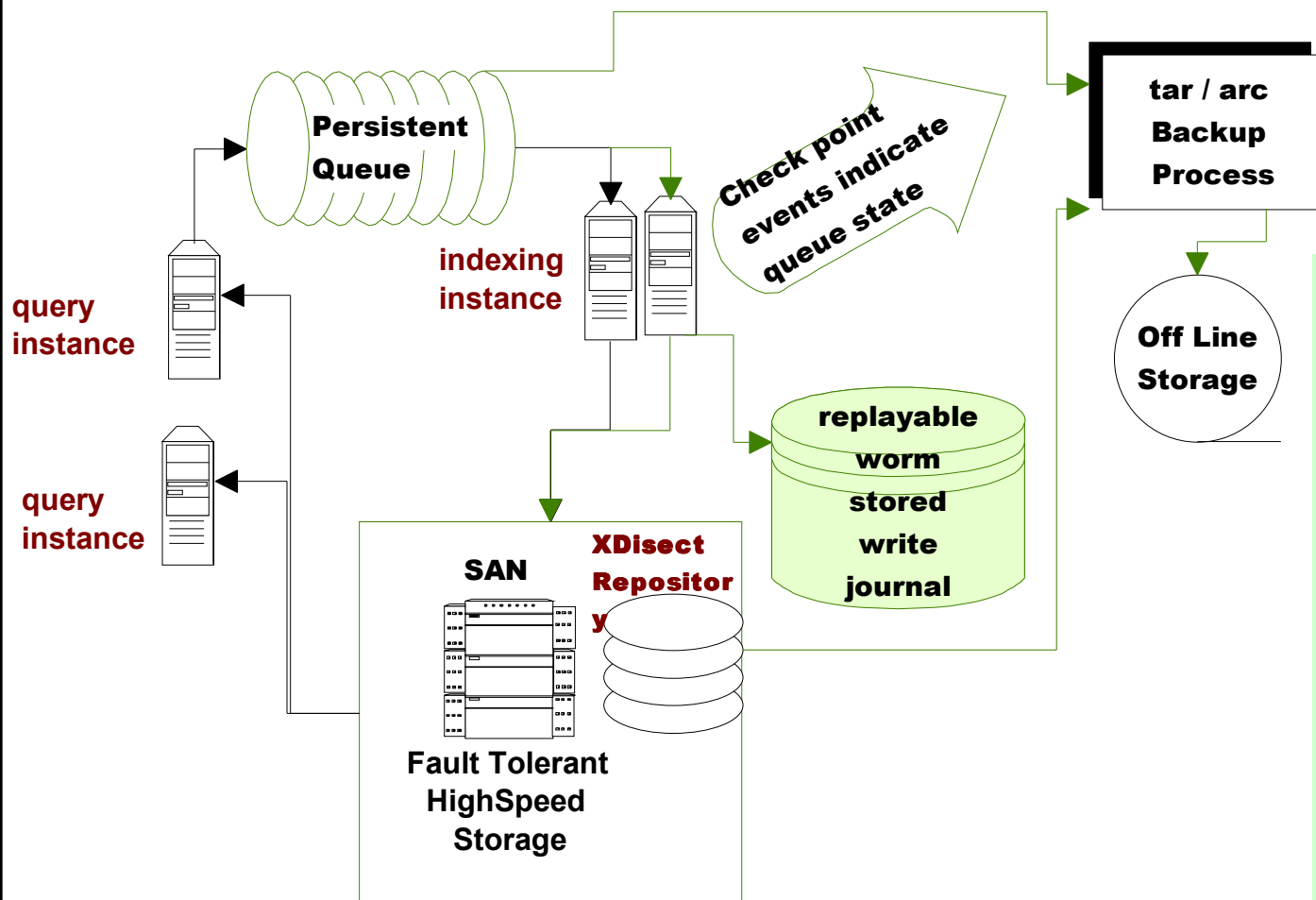


Addition of a Load balancing router can add fail over and horizontal scaling to the query instances

Intermediate application servers such as Bluestone not shown for simplicity.

Hot Backup & Recovery

backup any time, ASCII write journaling



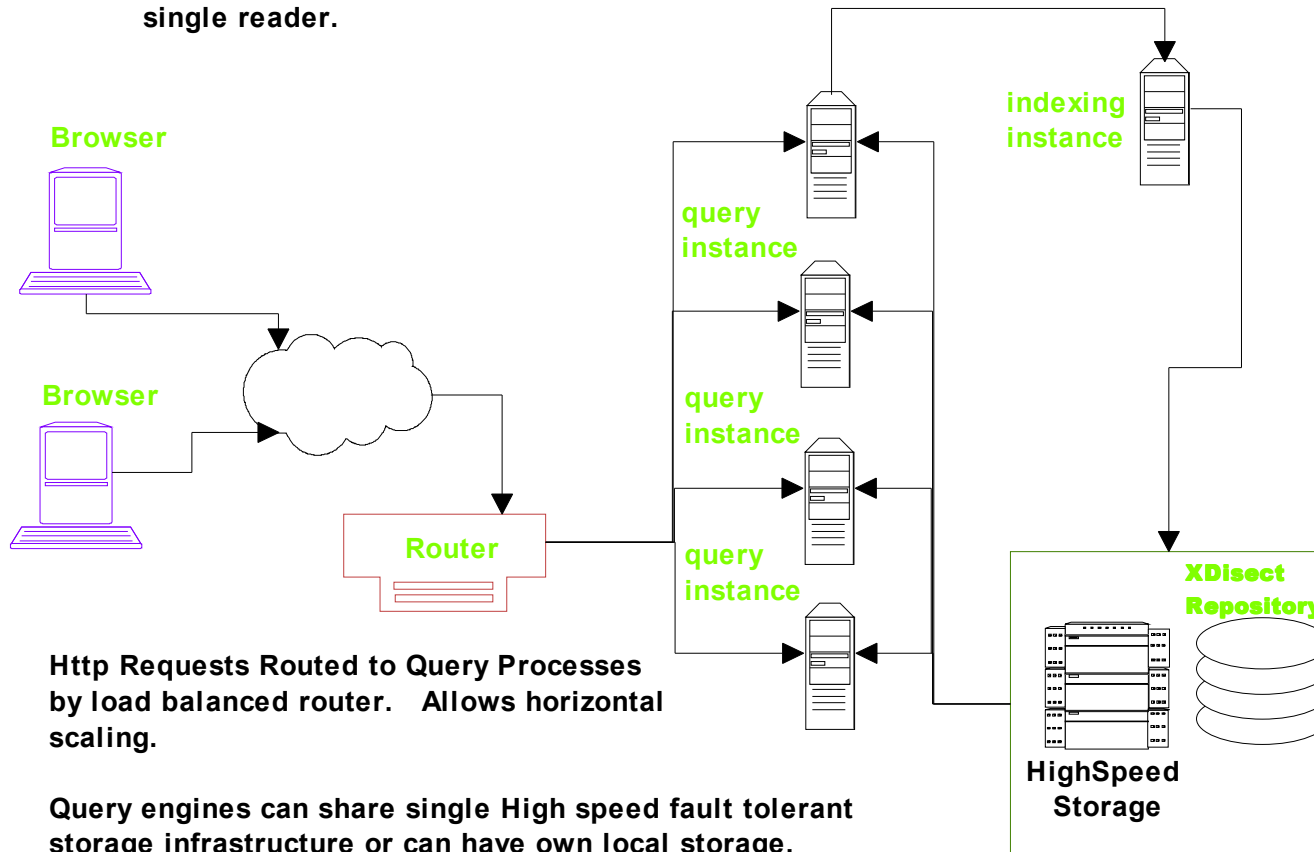
- Safe to backup at any time.
- Recovery involves
 - shut down
 - restore from tape
 - Replay journal into command queue for restoring since backup.
 - Restart processes.
 - Wait for command queue to empty
- Backup leverages standard IT backup tools & processes.

General N tiered architecture design (Hardware)

Scalability

Horizontal scaling using multiple reader agents - Shared Storage

Any query instance can receive update commands but they are all processed by single reader.



Http Requests Routed to Query Processes by load balanced router. Allows horizontal scaling.

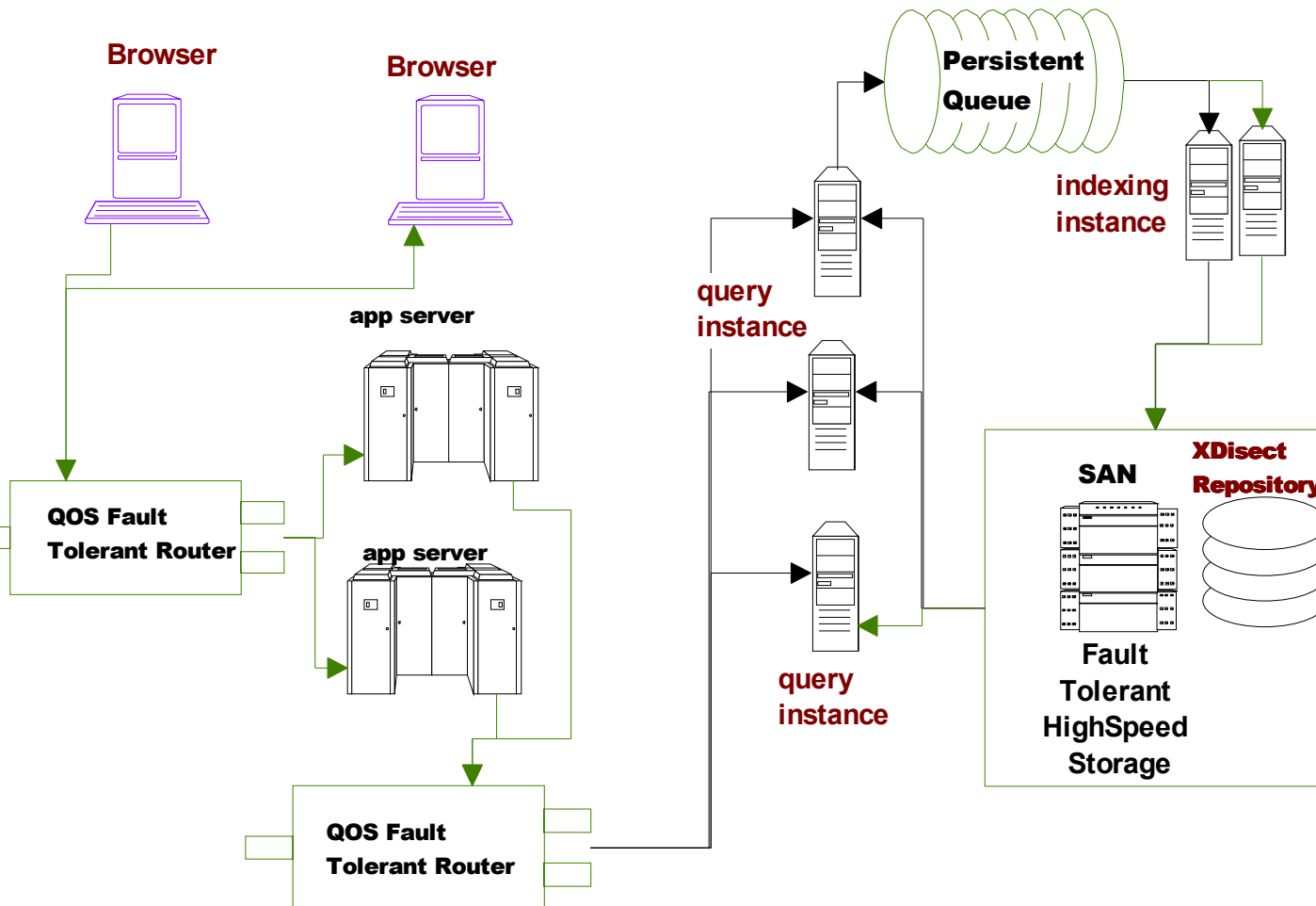
Query engines can share single High speed fault tolerant storage infrastructure or can have own local storage.

Many reader single writer design makes query scaling easy.

Support for synchronizing multiple data stores when query machines are using local storage.

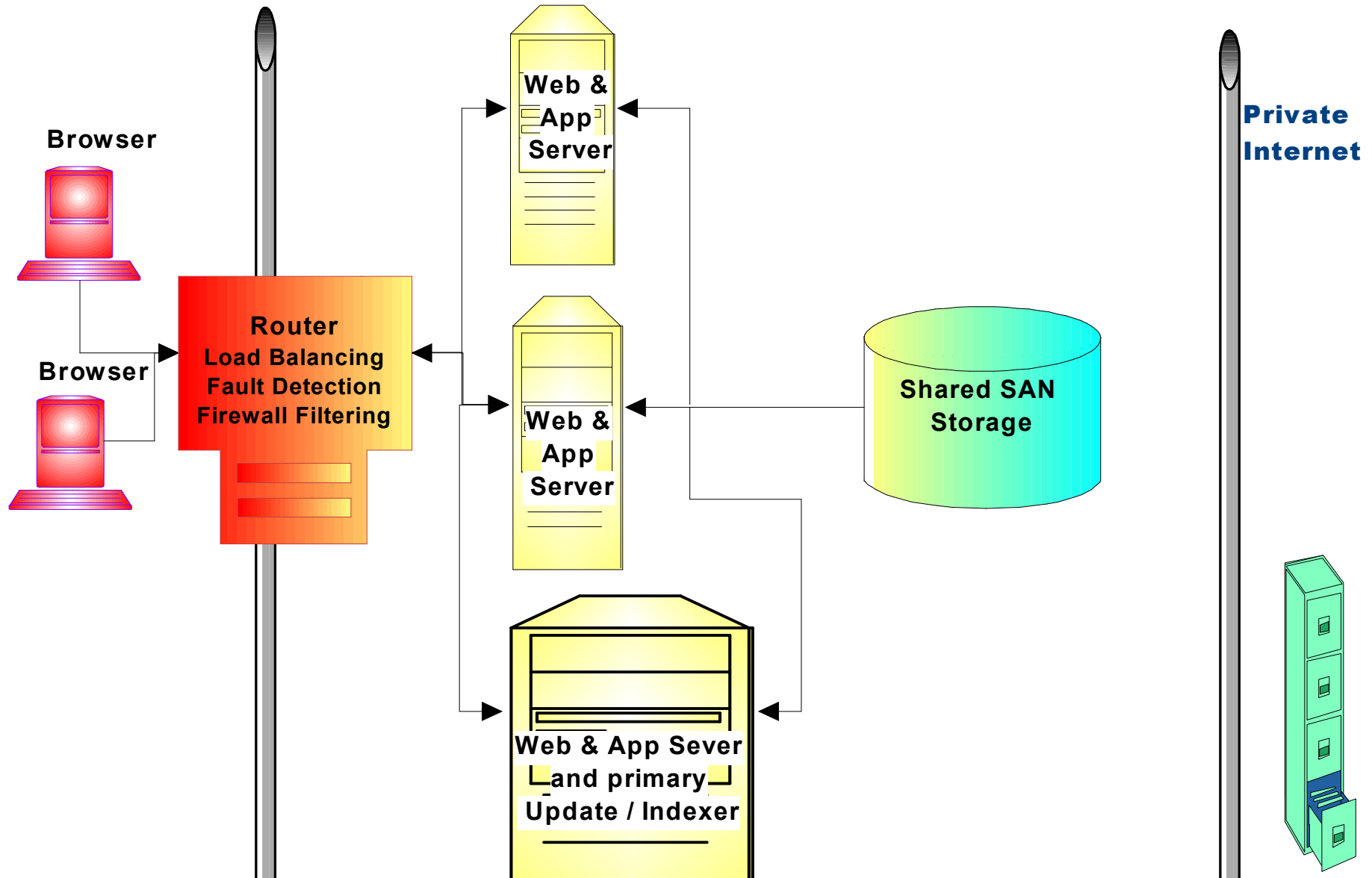
Fault Tolerance

utilizes standard QOS Routers

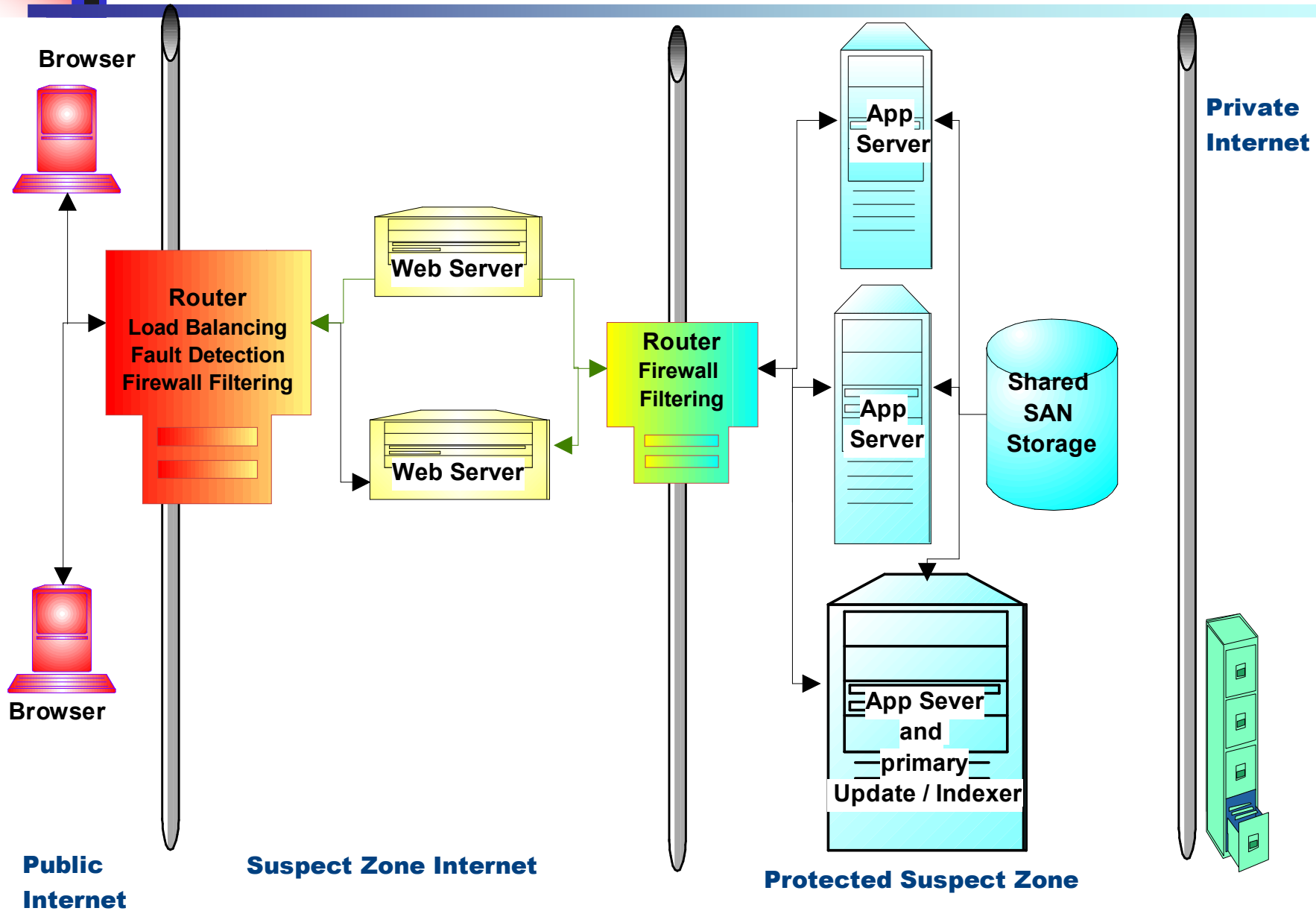


- Any query instance can receive requests for insert/update
- Requests are stored in a persistently queue.
- No single point of failure in the architecture.
- Total failure would require failure of all components
- Second router functionality can be done using software but we prefer the h/w router for performance.
- Only one instance of the XDisect Indexer is active at a time.

Minimal Hardware Architecture

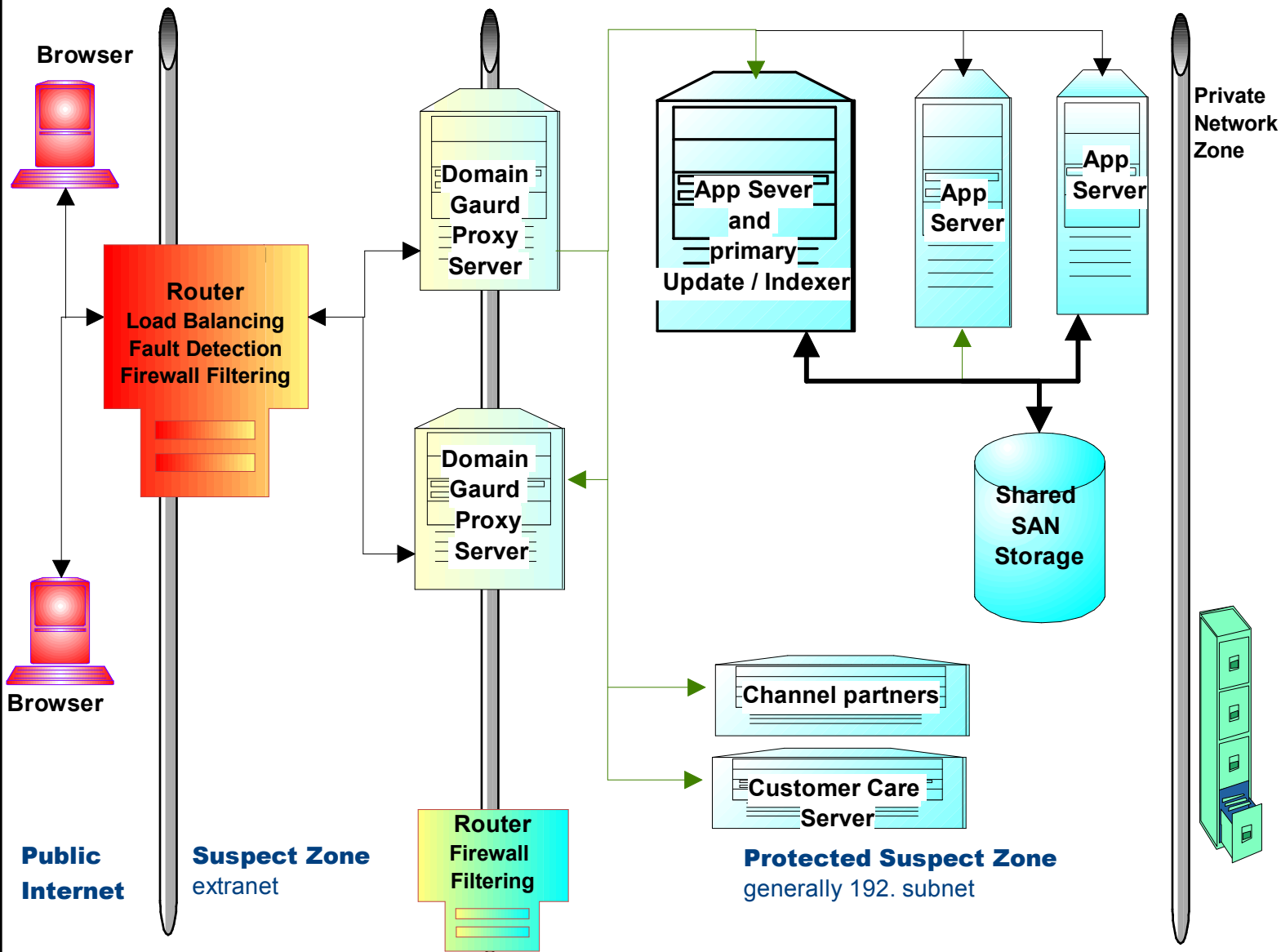


Typical Hardware Architecture



Hardware Architecture

when using Domain Guard Proxy



- Requires moving web servers into protected zone.
- The domain guard proxy server machine(s) can be relatively small.
- Any non http connections required by outside world must be configured in router.
- Outbound connections must be configured in router



Introduction of loosely coupled architectures

A Loosely coupled system is one in which any of the loosely coupled components can be replaced with an entirely new technology without having any impact on the components in the system with which it interacts.

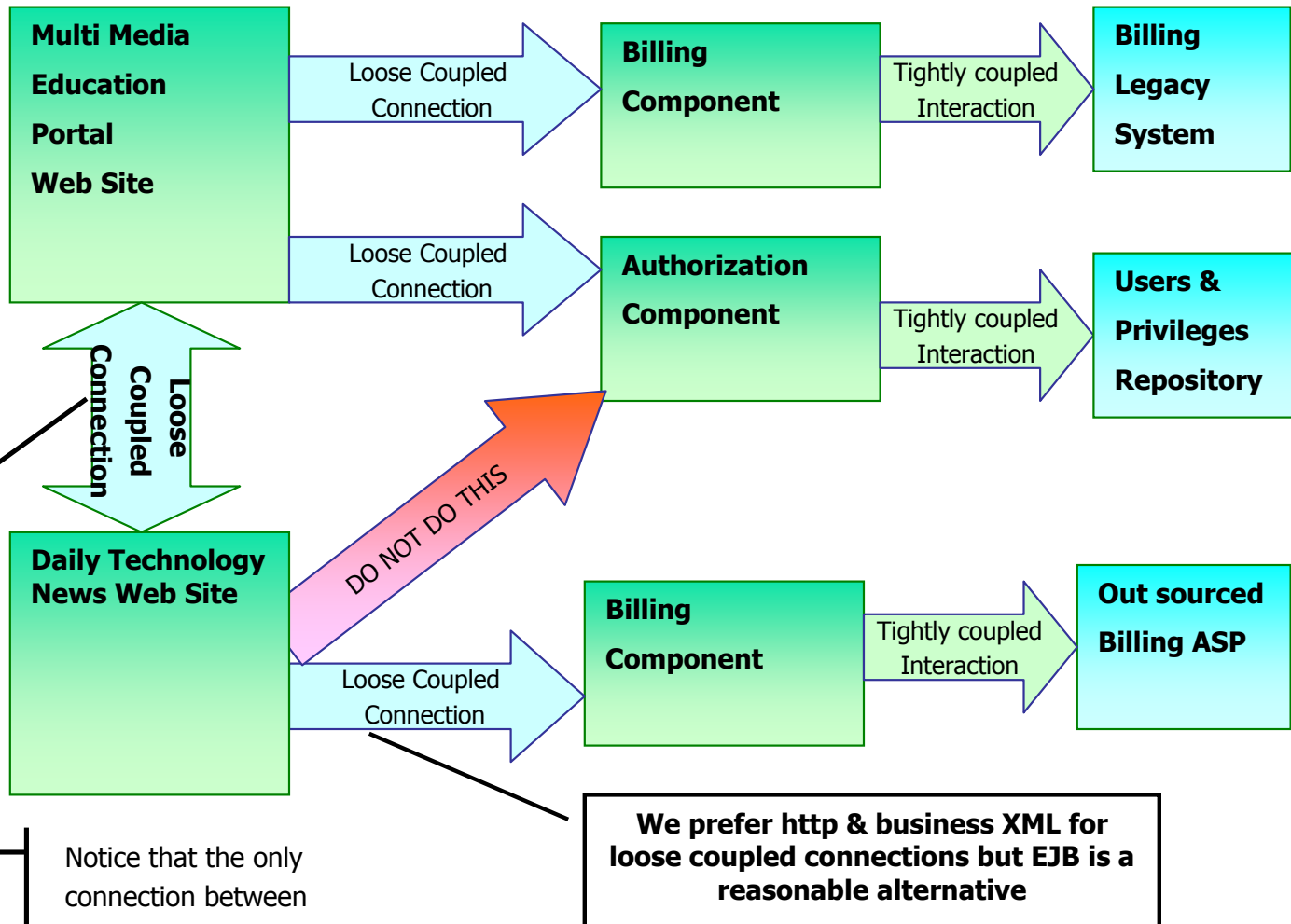
- Each component may be built with a different technology.
- Components generally may run on different or the same systems.
- A strong focus on the data interaction between components is essential to allow for narrow network pipes and effective isolation.



Design for Change

- A Loosely coupled component should be able to accept more data than it was originally designed without error.
- Adding new loosely coupled components should be relatively easy.
- Components should be able to dynamically register with in framework.
- Auction related techniques can be used to find best fit for current need.
- Use tools that can tolerate rapid evolution of underlying data.

What is loose coupling



Notice that the only connection between the two web sites is at the site level. Never expose the internal components across teams.

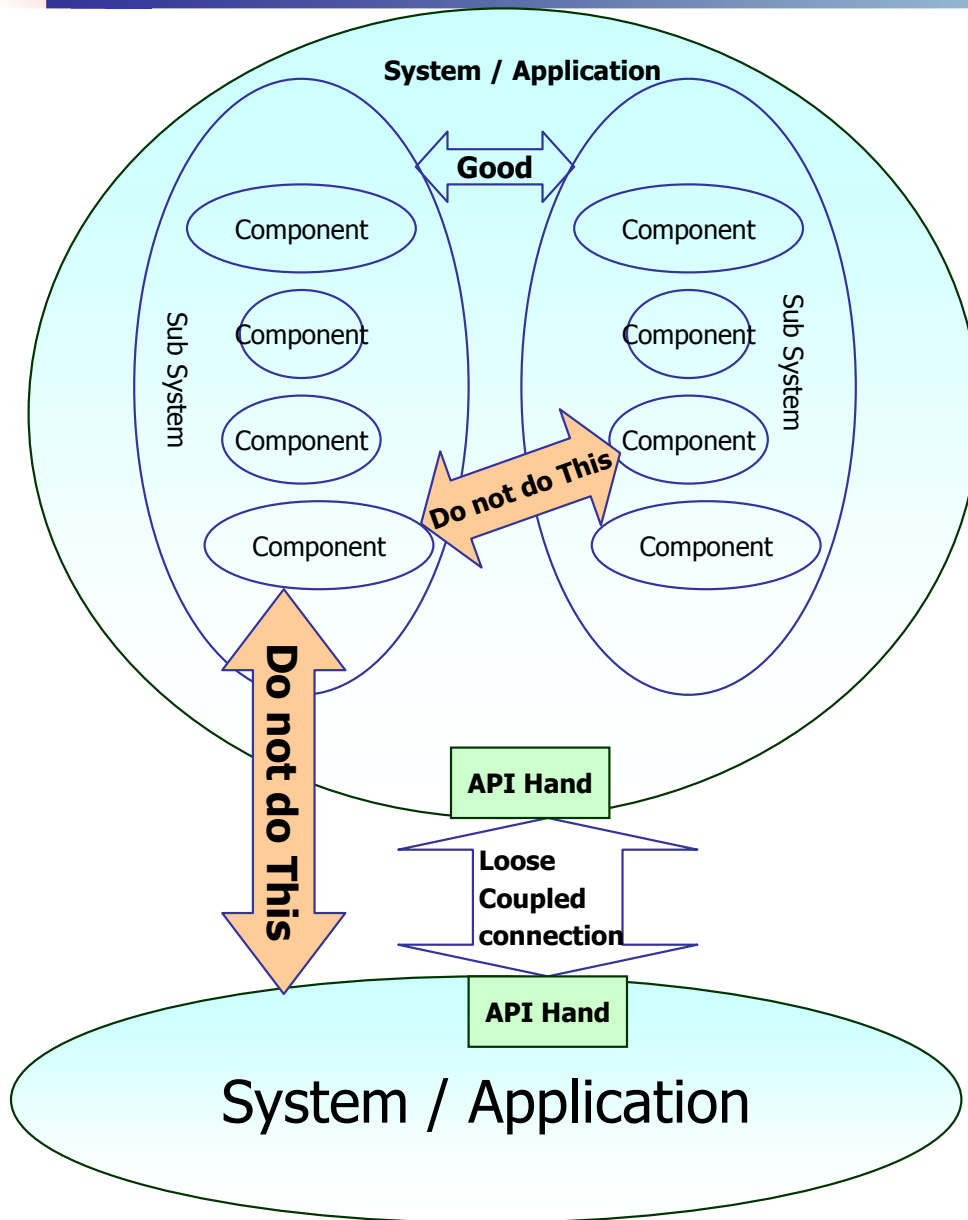
- It is important to spend more time on the yellow connection since it is likely to cross team boundaries.
- The blue connections are important but they can generally be fixed within the context of the team.
- The green connections are done in using any appropriate proprietary techniques as necessary.



Why loose coupling

- Technology trends changes every 3 to 5 years. This means that underlying components will be replaced but the system must live on.
- Much of the required functionality already exists in existing systems. Loose coupling allows those systems to be adapted to serve a broader audience.
- Small independently acquired teams of 4 to 8 people are the most productive. Allowing each team the freedom to work with their preferred tools while still integrating into a larger cohesive customer experience improves lower
- New components can be added at a relatively low cost rather than requiring changes of large monolithic systems.
- New functionality can be added with less problems overcoming entrenched organizational inertia.
- Purchasing decisions are made in non optimal ways. The true architect needs to keep things running even when popular vendors do not deliver as promised. Loose coupling is a good failure avoidance strategy.

Isolating Interfaces



- Always publish the highest level interface possible.
- Do not expose smaller internal components any further than mandatory.
- This was the Great issue with Smalltalk. It is also a problem with most Java systems.



Loose Coupling Trade off.

Positives

- Makes it easy to partition large projects across a series of smaller independent teams.
- Makes it feasible to incorporate many different technologies to obtain best of breed.
- Architecturally well suited to enterprise scaling.
- Works well in a strong partnering & collaboration environment with limited trust.

Drawbacks

- Standard GUI tools do not support it.
- Requires up front engineering to design data API.
- Junior programmers can muck it up really bad.
- Can take a little bit longer to show visible results at the front of the project.
- Project sponsors do not understand until late in the life cycle when then get to see the advantages.



Design trade off for loose coupling

- In Class Exercise



Adaptive architectures.

- An adaptive architecture is one which uses loosely coupled techniques to make it feasible to include existing systems as components in a larger system even when those components were not built to operate as such.

- The focus is on the plumbing.
- Figure out the API to the existing system first.
- Standardize the API where possible.
- Build minimally intrusive wrappers that talk the API on one side and the proprietary technology to the legacy system on the other.
- Additional functionality can be provided in the component wrapper.



Why use adaptive methodology

- Much of your existing Information value is tied up in existing systems.
 - It is generally faster than replacing the legacy system.
 - It avoids charter and authority battles since you are still using the legacy system.
- It provides a layer of abstraction so the legacy system can be transparently replaced in the future.
 - Bring more sophisticated systems out faster and with fewer negative impacts on existing teams.
 - With a little care the API can be reused by other systems so they can talk to the wrapper.



Building a E-business application using this approach.

1. Identify the systems that need to participate.
2. Define the API at the system level.
3. Get the players to agree on the API.
4. Build the Wrappers for each system that implement the API.
5. Test the Wrappers for each system.
6. Build the actual new application.
7. Deploy the new application.



Integrating new applications

Integrating new technologies

- Integrate at the open protocol level http & XML.
- Require a minimum of changes to the new applications.
- Do not require single vendor run times on every server.
- Design for integration in less than two months by a 3rd year programmer.
- Contrary to marketing press the whole world is not Java.
- Cold Fusion, IIS, ASP, PHP, Perl, Python, VB, C++, etc.
- Oracle, Broad Vision, Vignette, Interwoven, Documentum



Relative merits and problems of this approach

Disadvantages

- Most popular vendors take exactly the opposite strategy so they do not directly support his activity.
- Popular GUI development tools such as VB do not effectively support it.
- Requires a relatively large system before the paybacks become obvious.
- The current generation of programmers do not understand and will not like this approach.

Advantages

- Allows you to bring best of breed functionality to users Fast.
- Allows you to change as fast as the business changes.
- Allows you to deploy a large number of smaller teams.
- Makes it feasible to integrate functionality from external vendors.



What is Business XML.

- Business XML is statement of the business object being described using the highest level semantics feasible for the task.
- It is the opposite of exposing system and design details of the component systems.



Business XML Example

Always use the highest level semantics reasonable for the task at hand. (Very Common mistake in Junior and Intermediate XML programmers).

Business XML

```
<person>
  <name>
    <first>joe</first>
    <last>ellsworth</last>
  </name>
  <member>
    <since>2001-10-30</since>
    <expires>2002-03-15</expires>
  </member>
</person>
```

Programmers XML

```
<record name="person" type="single">
  <element type="hash-container" name="name">
    <field type="string" length="40"
      name="first">joe</field>
    <field type="string"
      length = "40"   name = "last">ellsworth
    </field>
  </element>
  <element type="hash-container" name="member">
    <field type="date" storage="julian" size="4"
      format="ccyy-mm-dd"
      name="since">2001-10-30</field>
    </field>
    .....
  </element>
</record>
```



Why is Business XML essential.

- **Robustness** – We do not want to have somebody else's minor change in design affect us or visa versa.
- **Change** - The internal design details of any system is likely to completely change every 24 to 60 months. The external API is the only thing that will remain in place.
- **Force Early Thinking** – In the process of forcing programmers & designers to boil up to the business level XML engineers have to truly understand their business problem.



Where XML fits this approach.

- Using XML and innovative tools such as XDisect components can be designed such that they can transparently handle evolving data structures with little or no changes to the underlying software. Will improve the components ability to meet business needs under times of rapid change.

- **Any place where a loosely coupled connection is implemented . Business XML should be strongly considered as the data mechanism for the API implementation.**
- HTTP is our default communication layer however EJB and messaging strategies work just as well and have their own unique advantages.



Detailed design of and how to use XML for loose coupling.

- This will be done as an In Class Exercise



Integration considerations

- In Class Exercise



Design trade off between adaptive and non adaptive techniques.

- In Class Exercise



Current XML enabling loose coupling.

Tightly Coupled

- WIDDL
- XML/RPC
- Soap – can go either way but tends to encourage tight coupling.
- ICE

Enable Loose Coupling

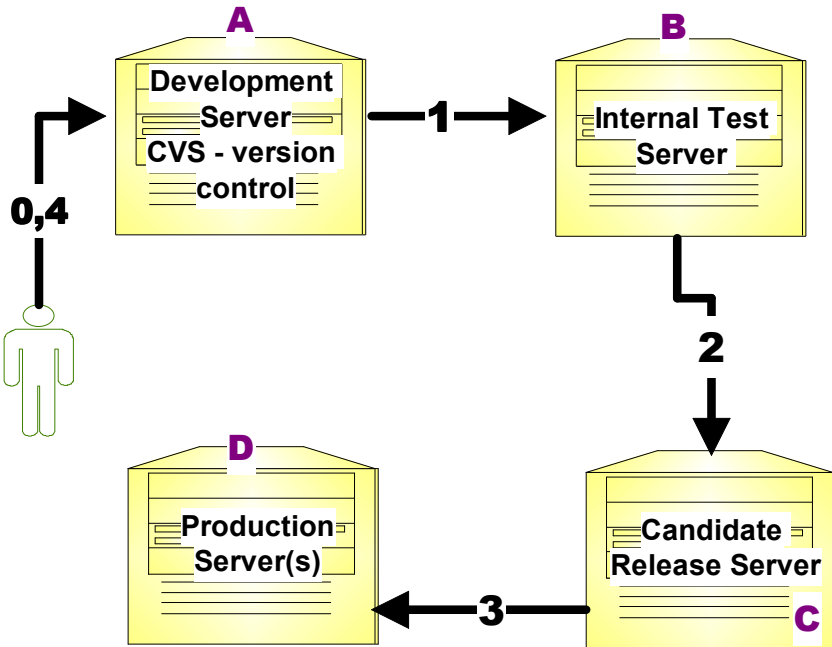
- RDF – Resource Description format.
- Xlink – Link and topography description
- CDF – Channel Description format.
- WebDav – Distributed documentation synchronization & management.
- eBXML – business to business
- BizTalk – started out O.K. but seems to be going towards tighter coupling.



Limitations of current XML based approaches.

- Current XML users tend towards low level XML / RPC specification which is fragile.
- Many programmers tend to create mega XML documents rather than sending only what is appropriate to the current business context.

Release & Promotion Flow



- In some instances server A and Server B can be same machine but it is less reliable and more likely to cause delays due to additional configuration overhead.
- Server C & D can be same machine but it is not recommended.
- Servers D should be large powerful systems. Server A should be almost as powerful as D. Servers C & D can be smaller.

- Users provide requirements and Feedback on line 0,4 and work is done to meet these requests on server A.
- Movement from server A to server B is approved by unanimous agreement of developers and is normally pre-arranged as code freeze. Typical Target every 2 weeks. Normal basic script testing investment 3 to 5 hours.
- HP Internal (core team) testers review and QA software while on server B. Core testers and developers agree on movement from B to C. If core team testers reject build then activity moves back to A. Typical Target every 4 weeks. Internal tester is responsible for providing end user script for use in next step. Normal testing investment 12 to 18 hours. Script update 10 to 20 hours.
- Extended team testers and customer manager do extensive testing while on server C. This is to validate the entire scope of application and that all enhancements satisfy original business requirements. On approval release is moved from C to D. This move must be scheduled with IT and field implementation staff so they can warn customers about upcoming changes. Typical Target once a quarter. Normal investment 4 people 2 hours each.

Case Studies.

- Secure Extranet Portals
- Market Makers
- Front End Search Cache
- Search Aggregation
- Customer Chosen



Secure Portals

Case Study #1

- Designing an Enterprise Personalized corporate portal.
 - Overview
 - Challenges & design considerations.
 - Examples of designing business-business apps.
 - Benefits, risks and limitations
 - A concrete example of a new business to business app. that can withstand change.
 - How to manage change in the application.

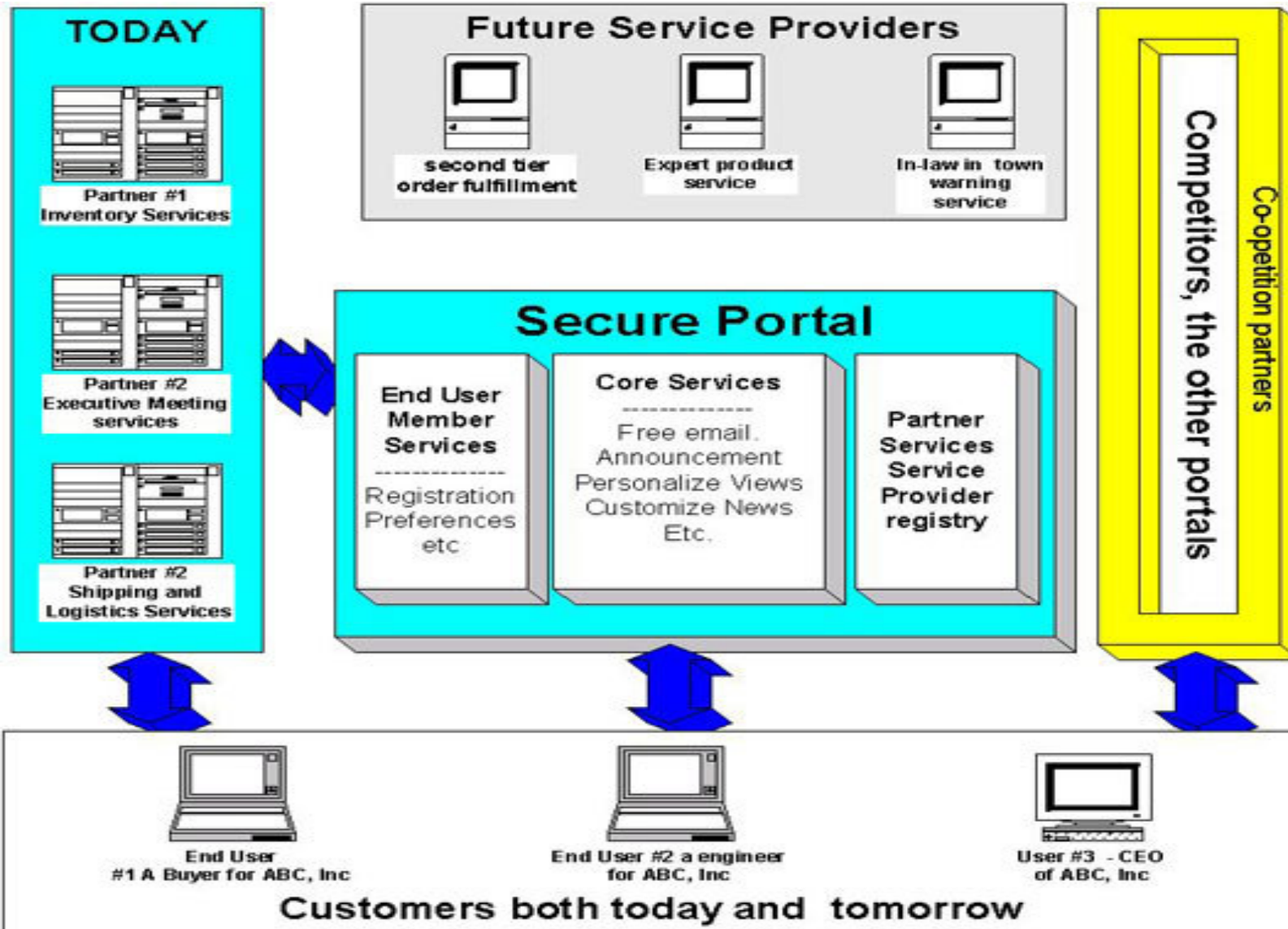
- Adapting existing business to business applications.
 - How to adapt and existing applications to this methodology.
 - What are the challenges.
 - A concrete example of designing an application to adapt to change.
 - Benefits, risks and limitations.



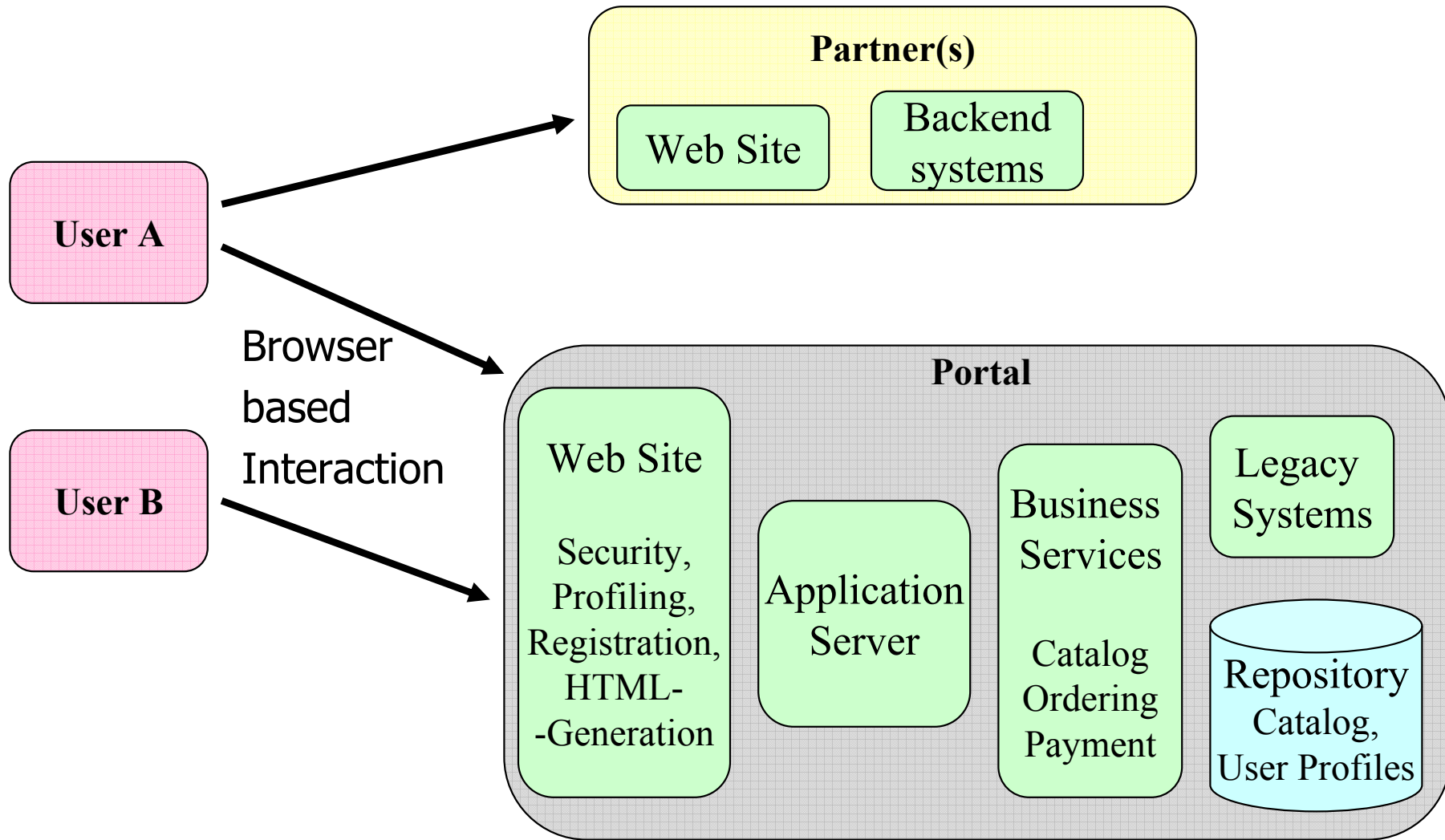
Hub and Spoke View of the World

- There are really multiple hubs.
- The total number of hubs can be in the thousands.
- Each Hub can add new data requirements.

A Typical Secure Portals



Example Portal / E-Business App

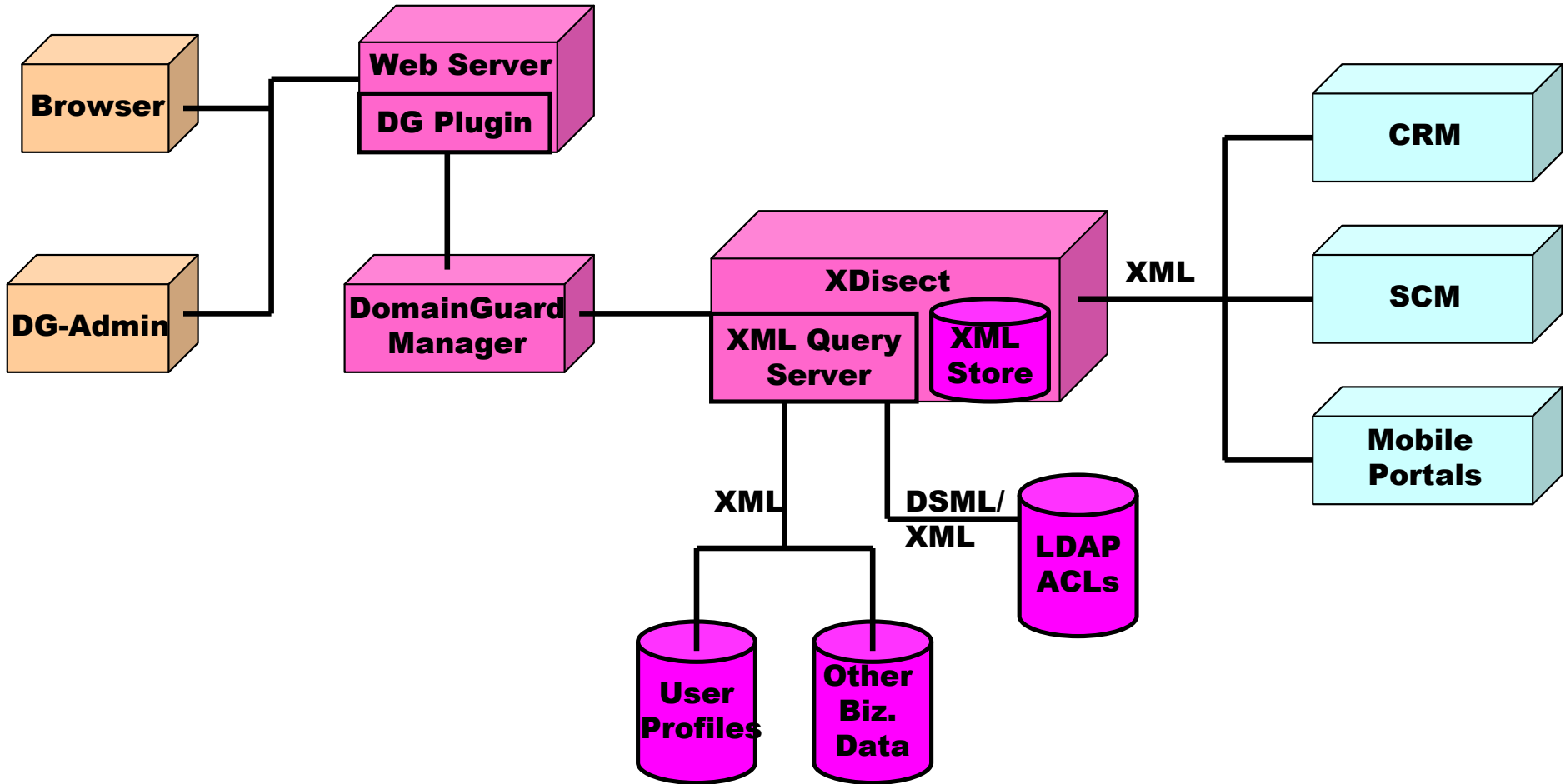




Secure Extranet Portals

Clients

Corporate Business Apps



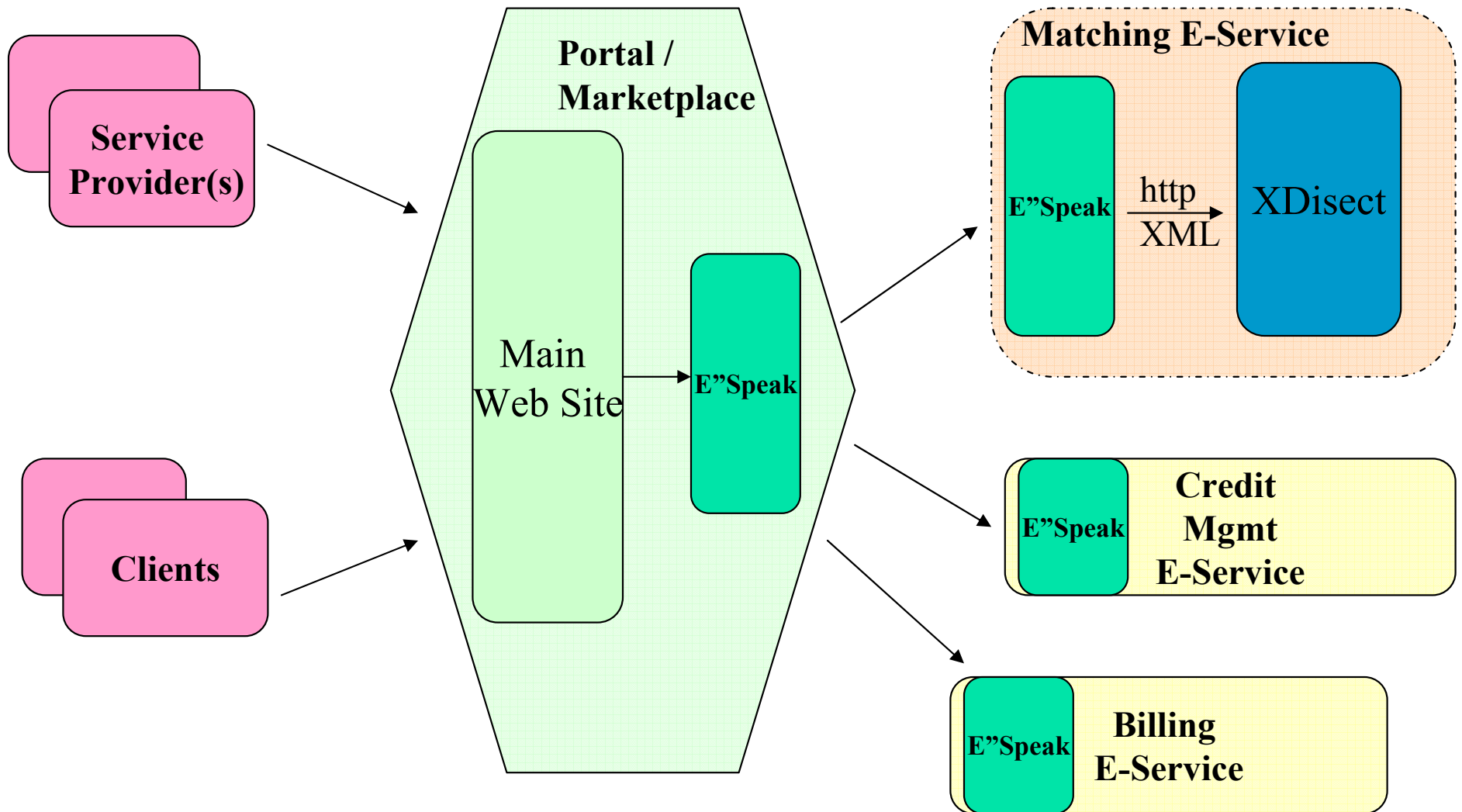
Corporate Databases



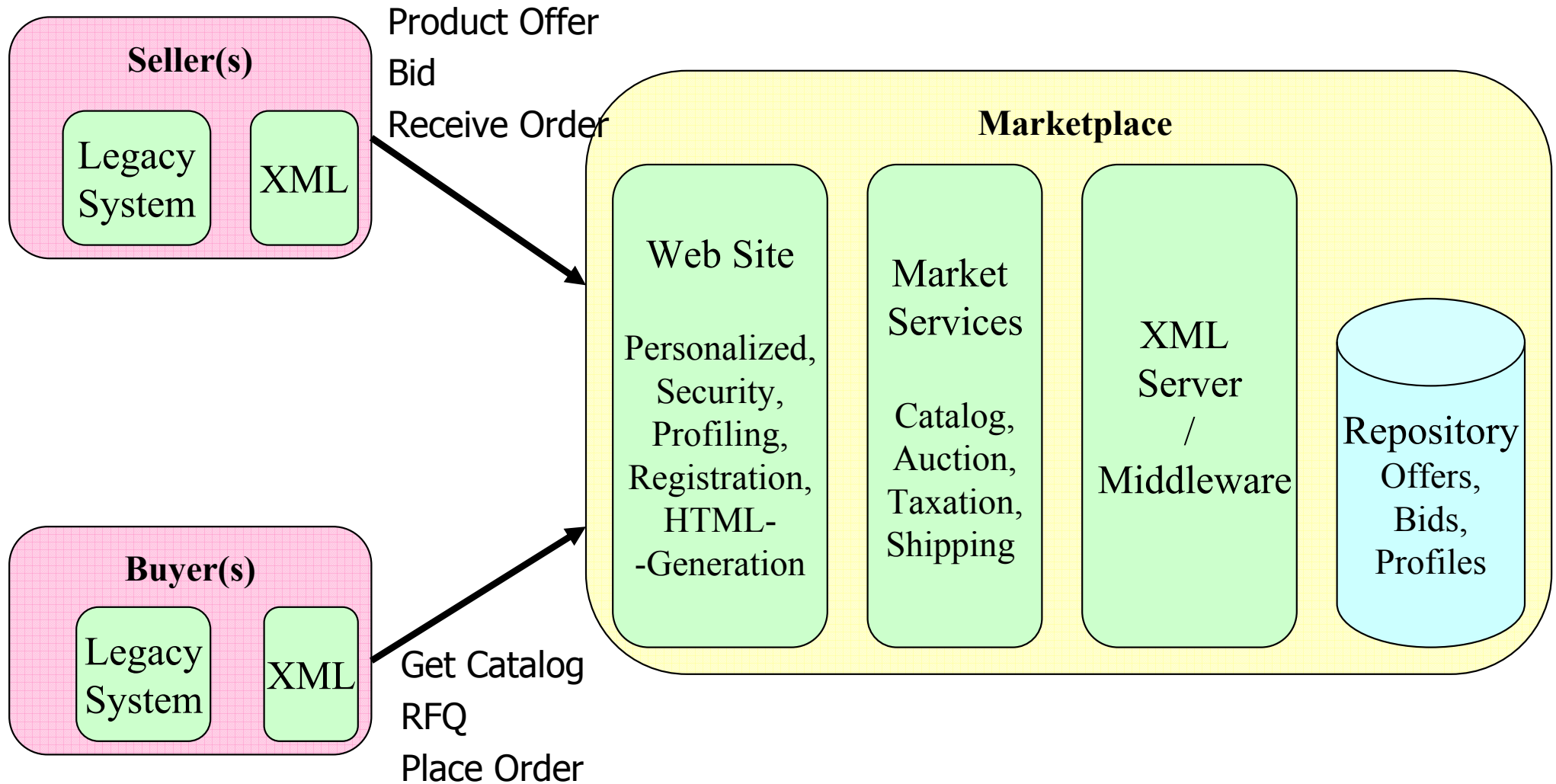
Case Study#2

- Market Makers
- Brokers
- Auctions

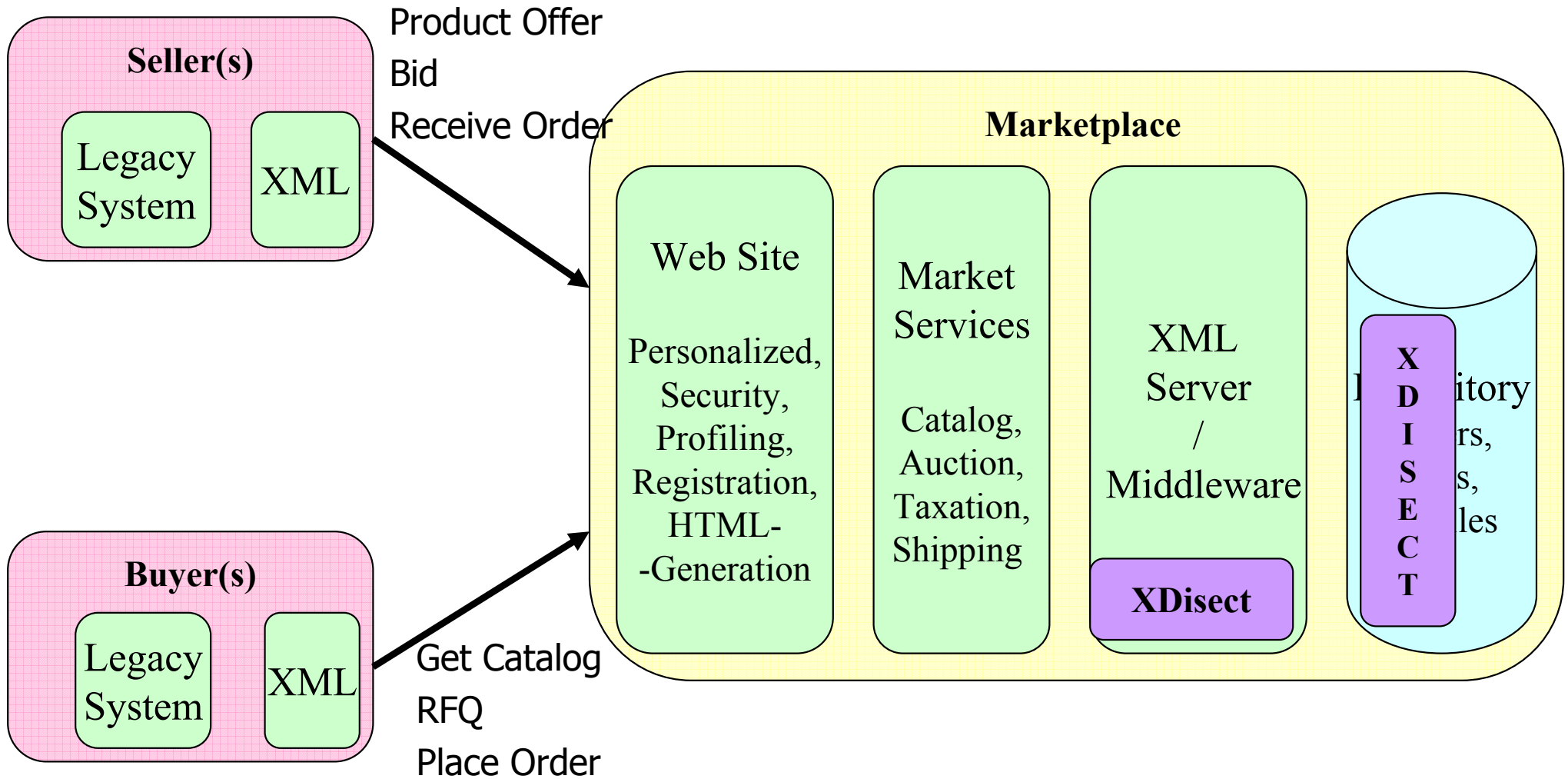
Matching E-Service - Architecture



Example - Current Market Makers



Next Generation Market Makers

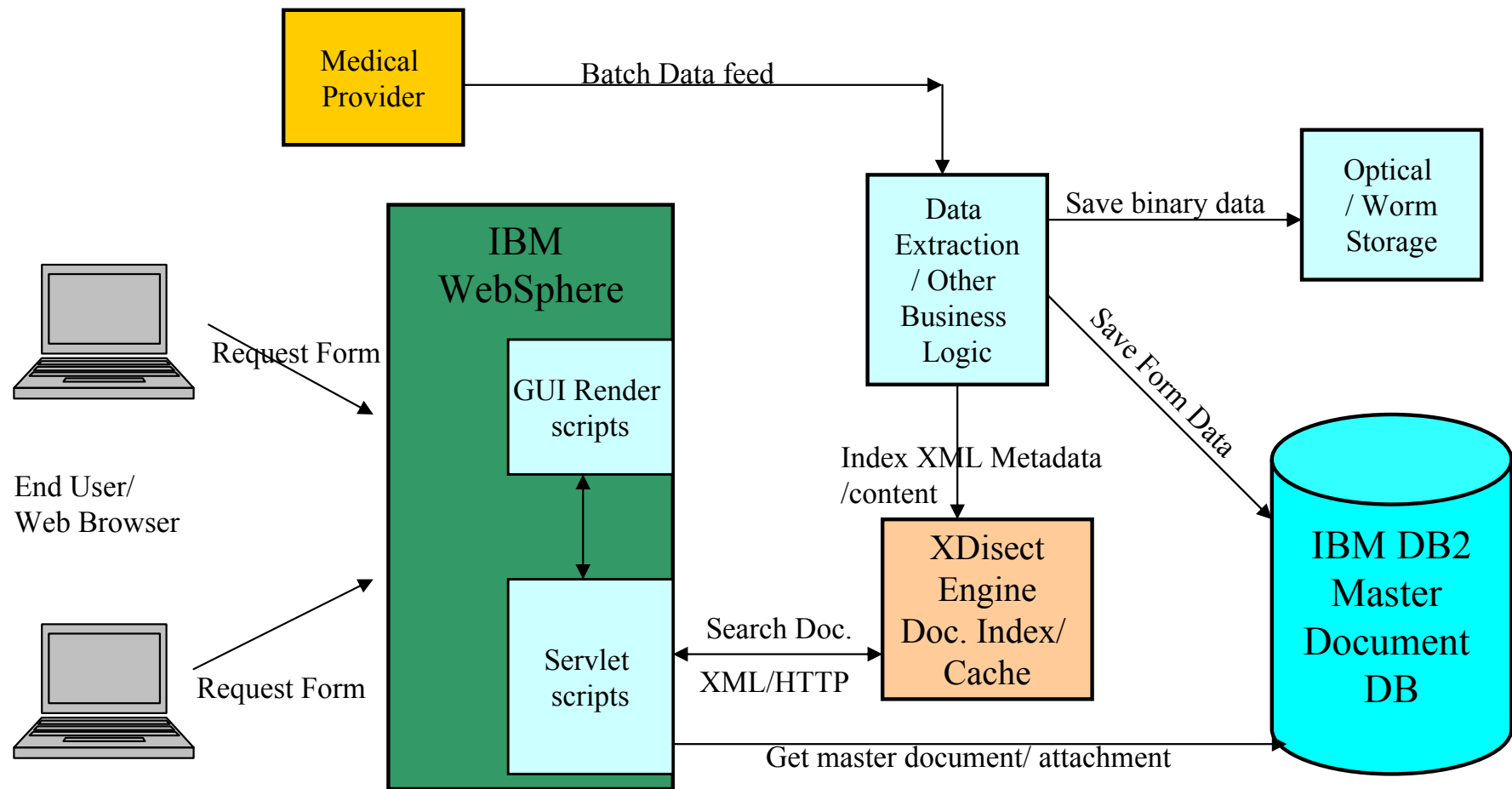




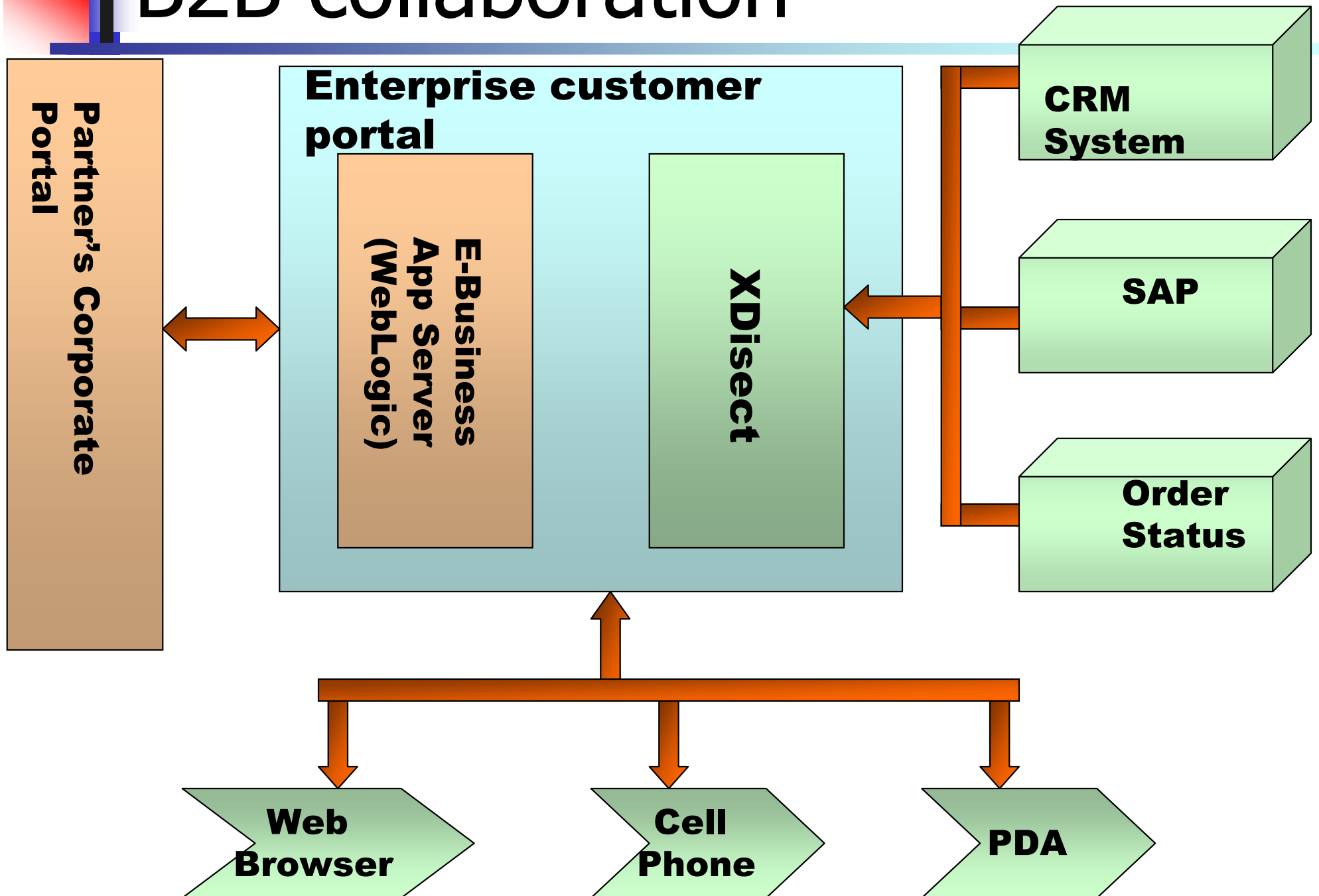
Case #3

Front
End
Search
Cache

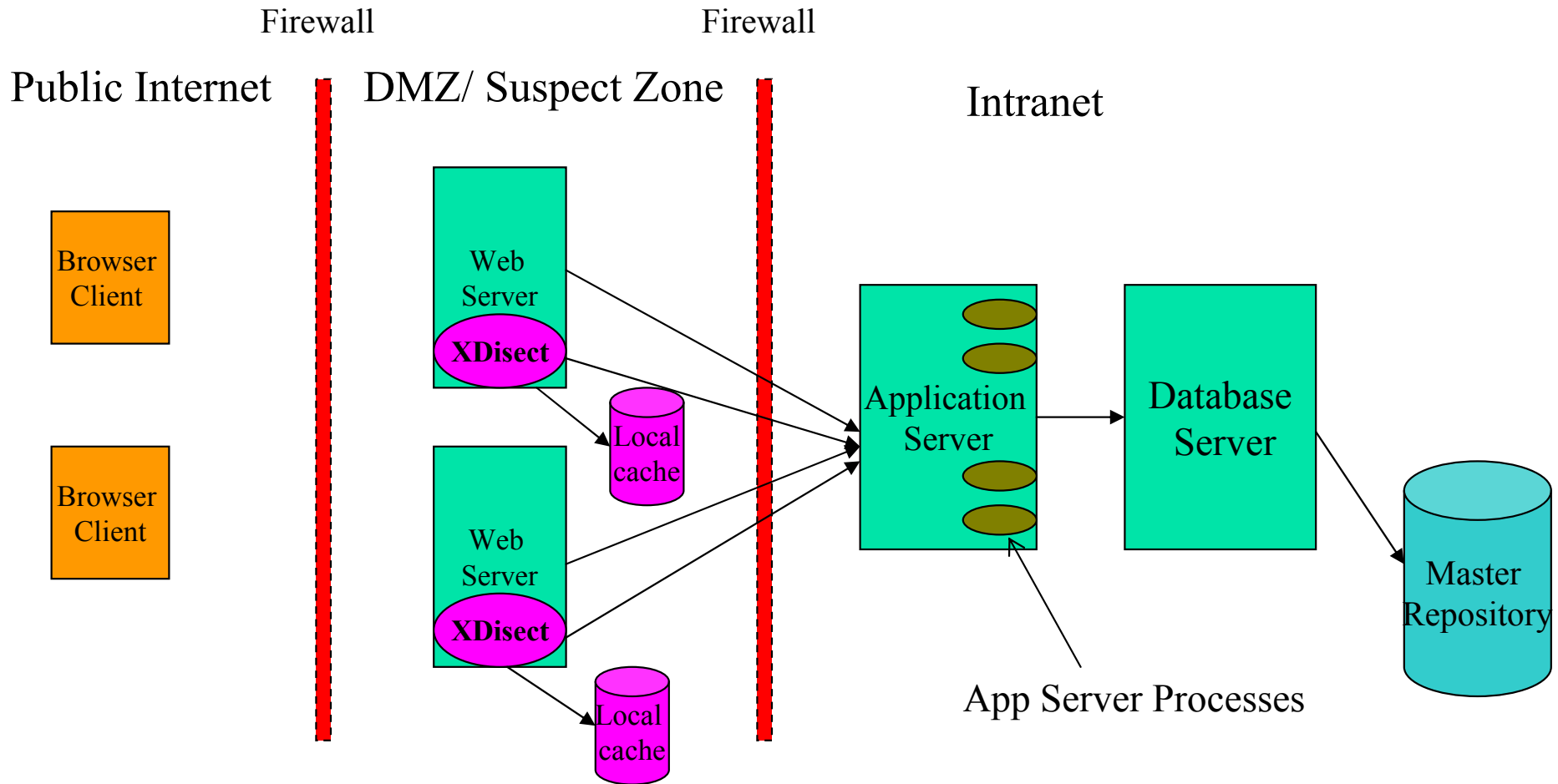
Flexible Search Cache



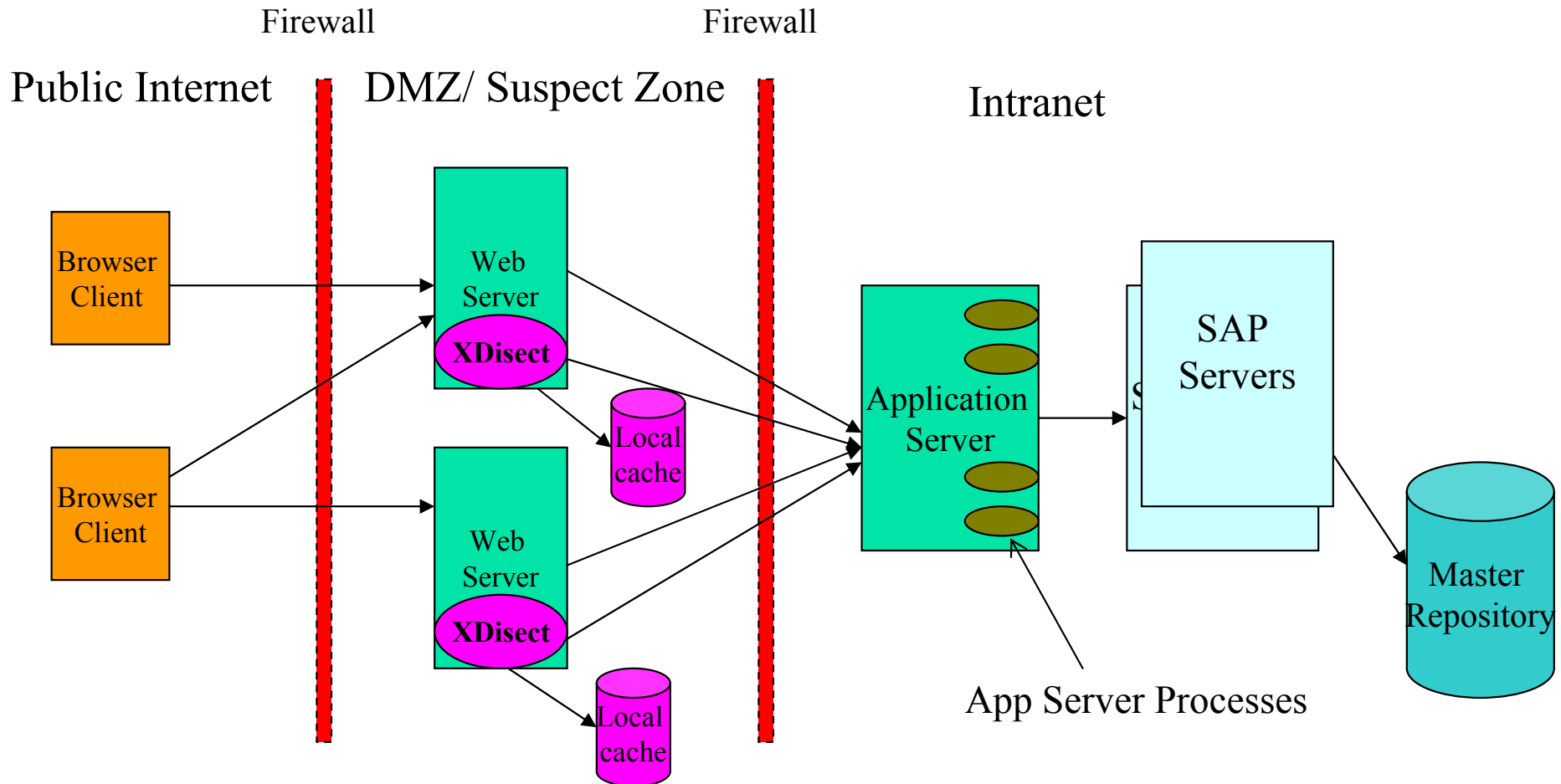
B2B collaboration



N-Tier Front end Database Cache



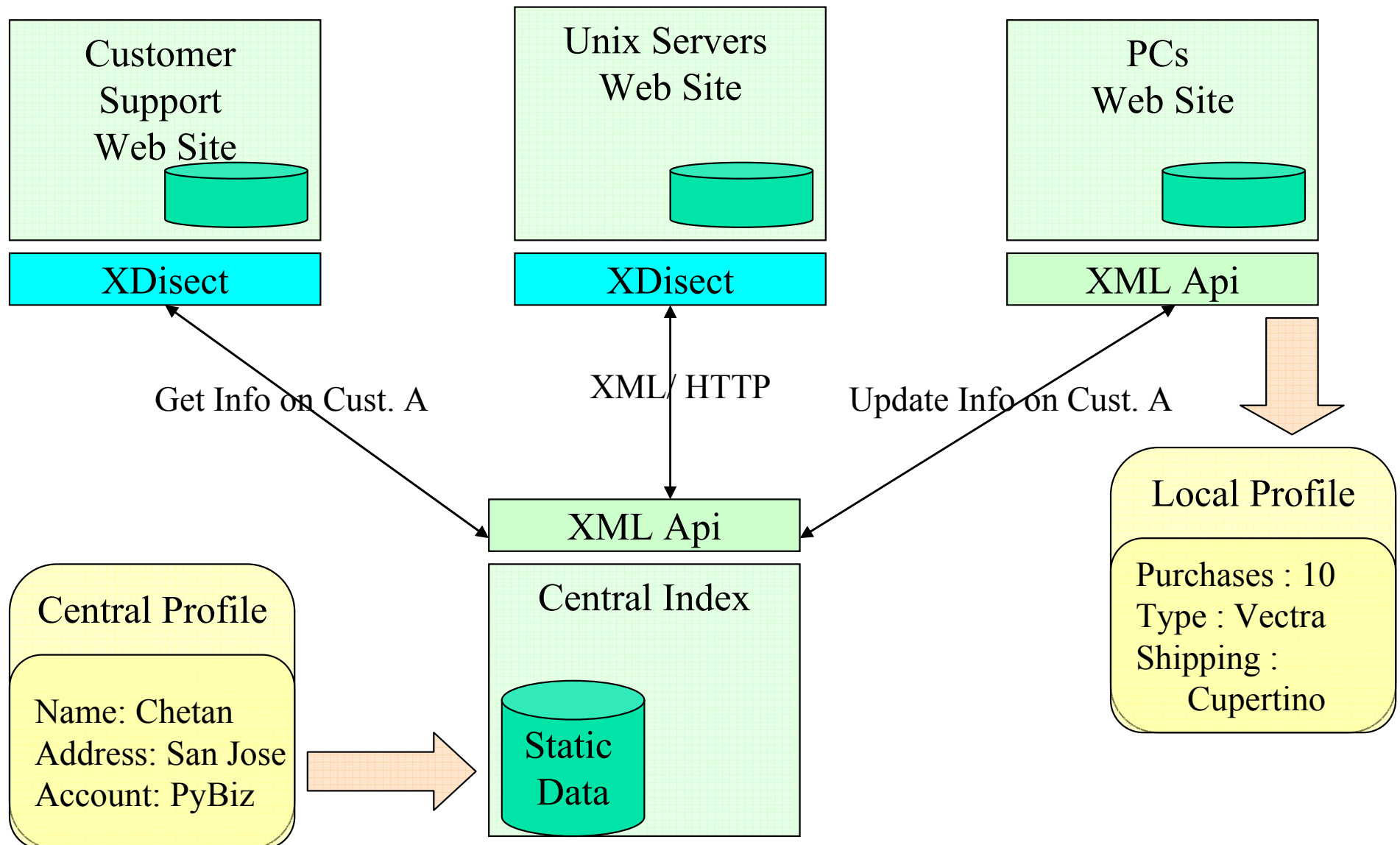
N-Tier Legacy System Search Cache



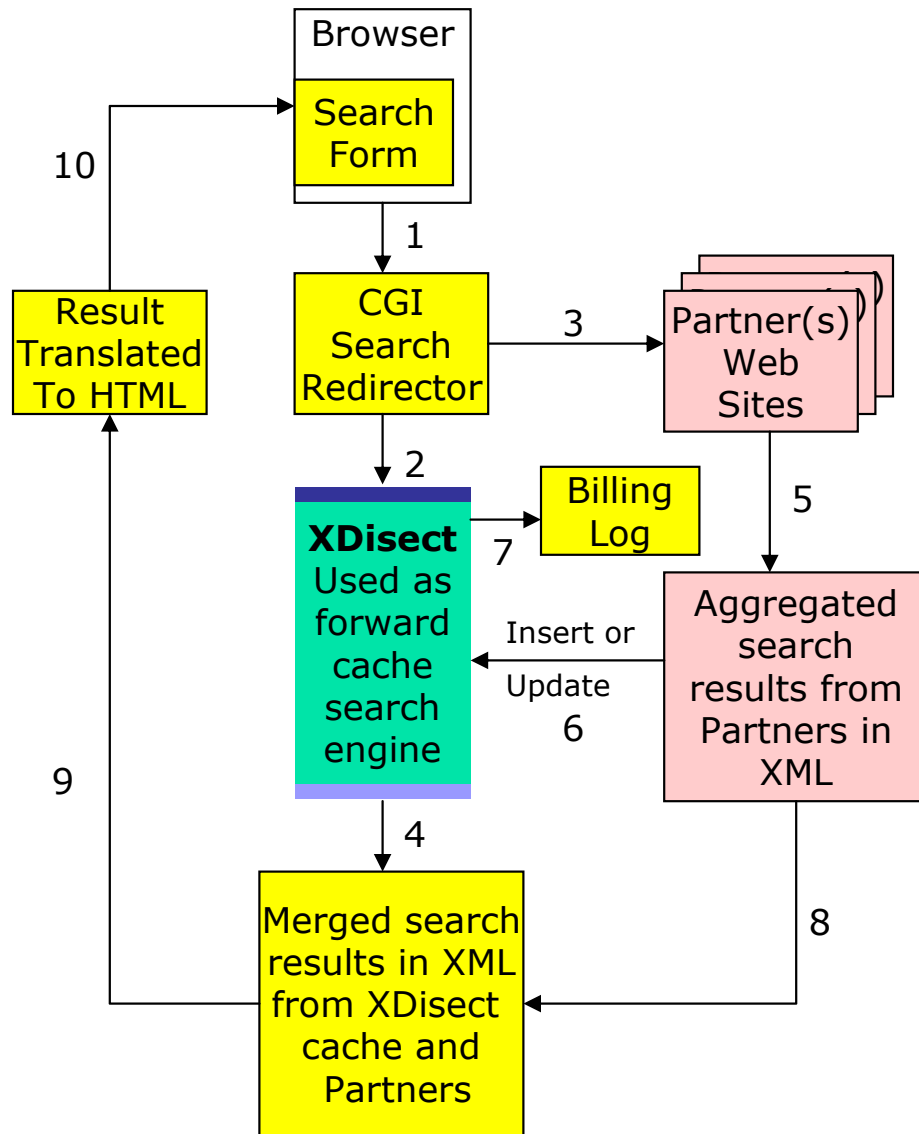
Search Aggregation

CRM Global Customer View

Fortune 1000 customers

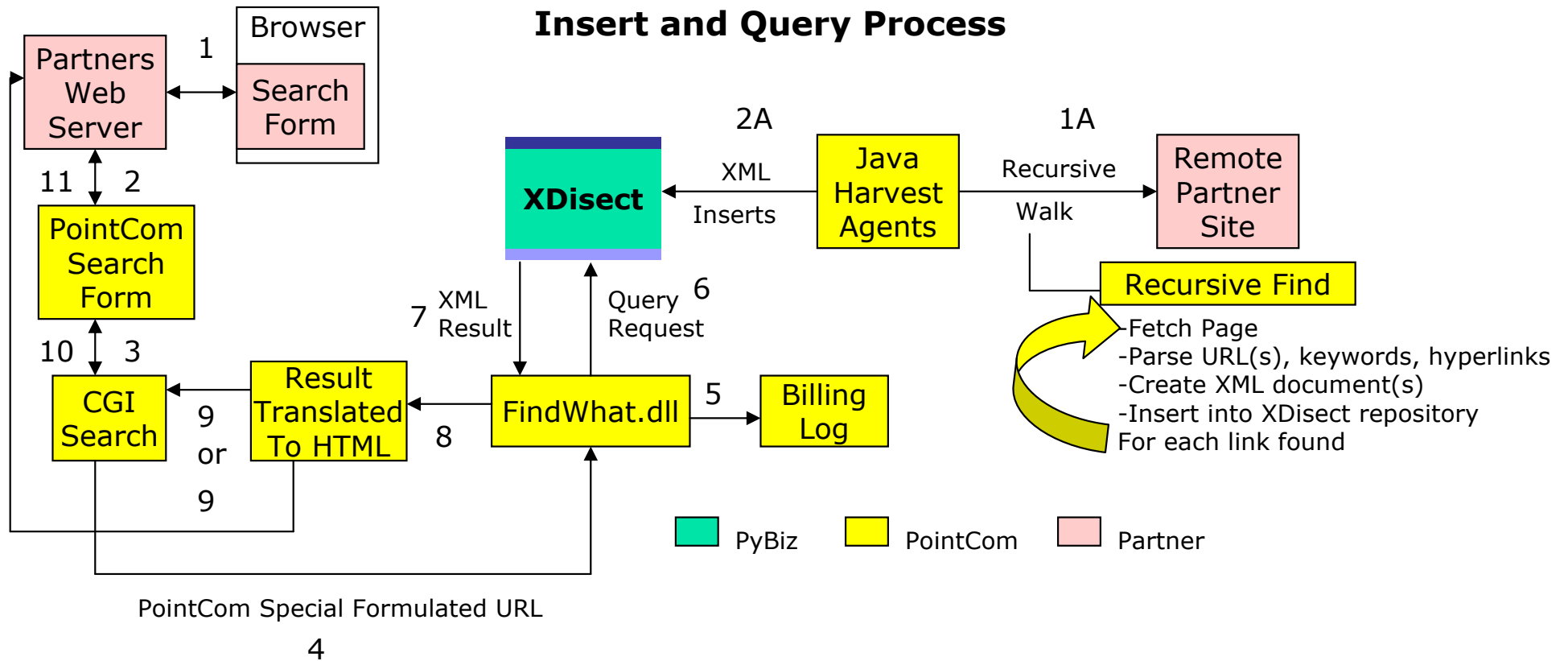


Forward Search Cache Federated Queries



Federated Search Continued

Insert and Query Process





Case #5 selected by seminar participants.

- Defining basic requirements.
- Defining business ness architecture.
- Defining the technical architecture.
- Defining the hardware architecture.
- Defining the software architecture.
- Loose coupling considerations.
- Adaptability considerations.



Case Study Components

- Scenario Definition
 1. Defining the basic requirements
 2. Defining the Business Architecture
 3. Defining the Technical Architecture
 1. Basic user or screen flow
 2. Defining the Hardware Architecture
 3. Defining the Software Architecture
 4. Scalability Considerations



Conclusion

- **Loosely coupled and adaptive techniques can improve time to market while increasing total functionality delivered to the end user.**
- **These techniques will save money while preparing organizations to effectively deal with change.**
- They will not work for all teams and they require a project(s) of a certain size before the value delivered becomes apparent.
- The biggest issue in deploying these techniques will be one of training and forcing the use of the mind set until the organization matures sufficiently enough to see the pay off of using these techniques.