Java is a trademark of Sun Microsystems, Inc.

Java

JavaOne℠

TS-5154: XTP: Patterns for Scaling SOA, WOA, and REST Predictably with a Java™ Technology-Based Data Grid

Dave Chappell
VP & Chief Technologist, SOA
Oracle

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

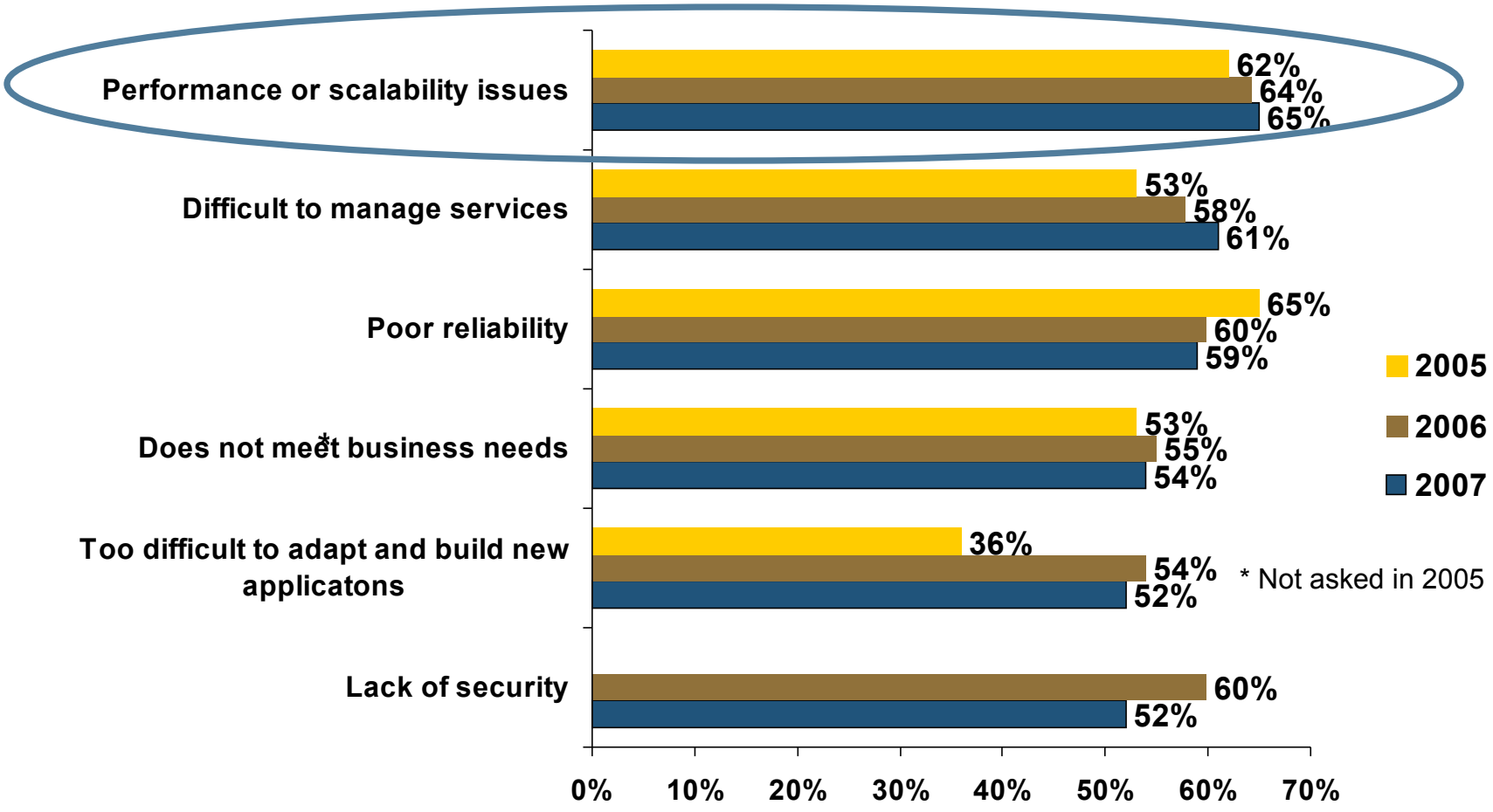
Agenda

- SOA and REST on Application Grid
- Intro to REST
- XTP and Application Grid
- Challenges to Scalability
- Use Cases and Patterns:
  - Shared Web Container State
  - Service-Result-Cache
  - Service State Cache
  - Not Your MOM's Bus
  - XML Grid

"*If you want a 3 second response time, you get 1 sec in the presentation layer, 1 sec in the middle tier, and 1 sec in the backend. Period*!"*

*commenting on the need for middle tier caching and in-memory state management

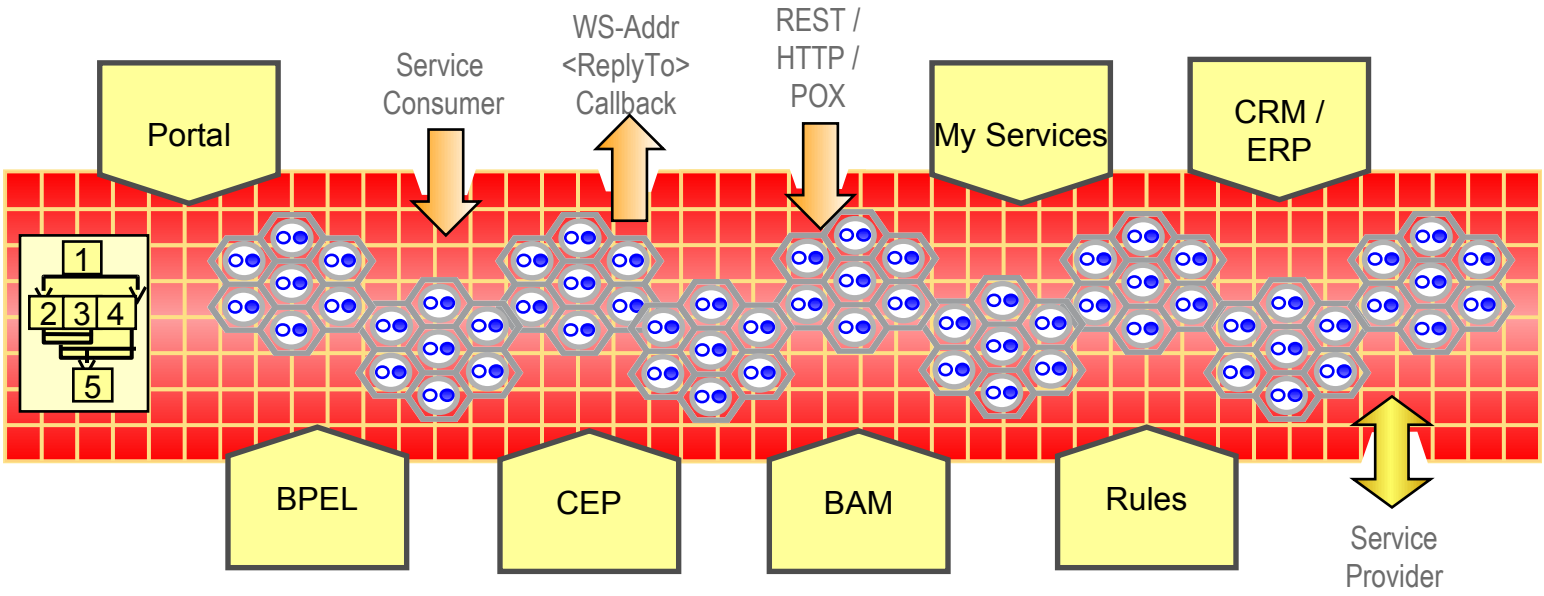
Concerns linger about lack of performance, scalability



Q: What would constitute SOA failure? 2007 Base: 549 / 2006 Base: 473 / 2005 Base: 167 (Among current deployers)

Next Generation SOA and REST on App Grid

Combines Orchestration, Mediation, Service State Caching, EDA,Service Result Caching, Compute Grid, Deterministic GC



State-aware continuous availability for service infrastructure, services, application data, and processing logic

=Service state data

=Stateful Service Orchestration

What is REST

Introducing REST

- **Nouns:** All resources, or nouns, are network-addressable with a global URL. Resources have "href" references to other related resources. The only way to address a resource is with a URL.

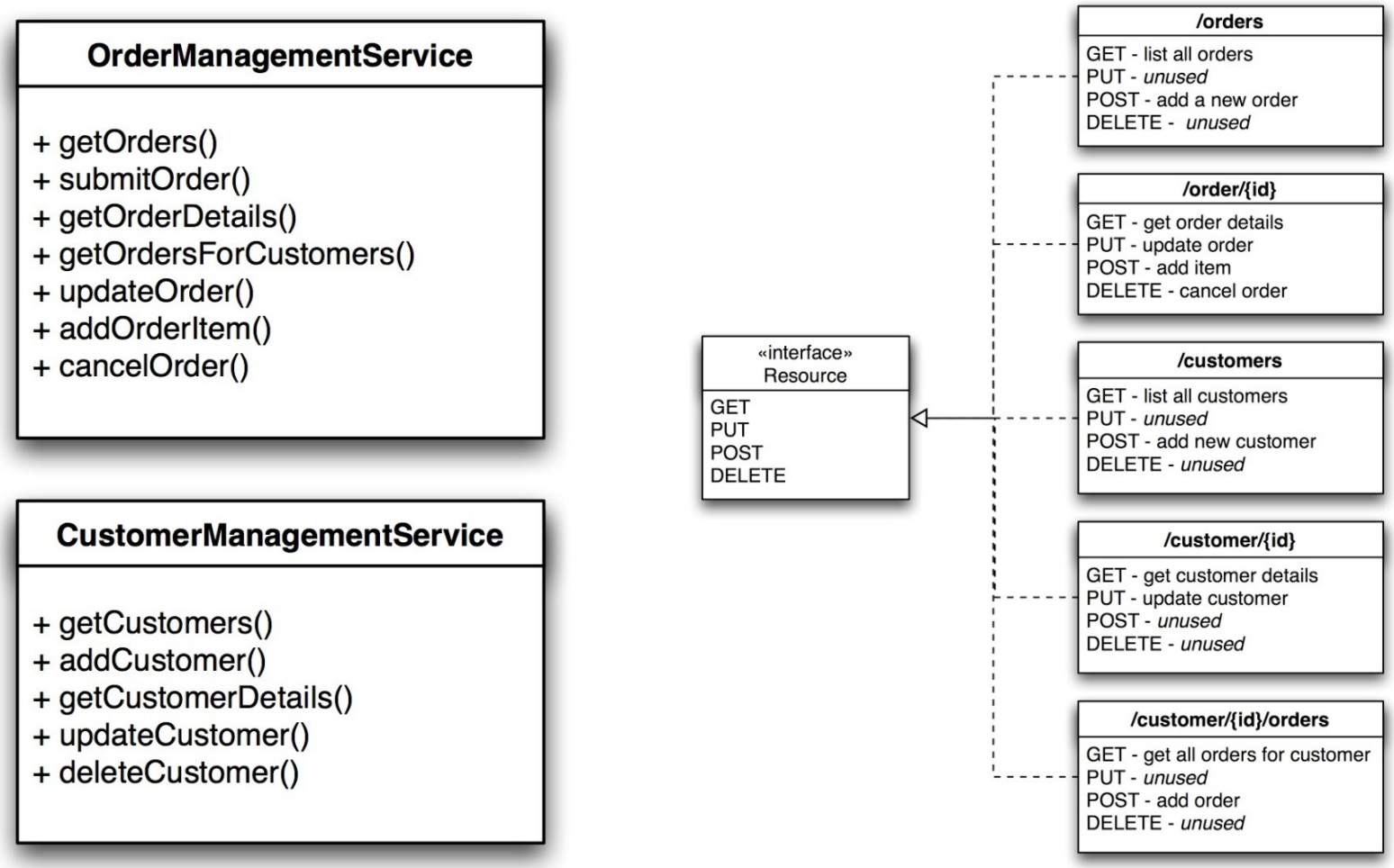
- **Verbs:** A common set of verbs are used to access all resources. The HTTP methods GET, PUT, POST, and DELETE map to retrieve, update, create, and delete (CRUD) functions.

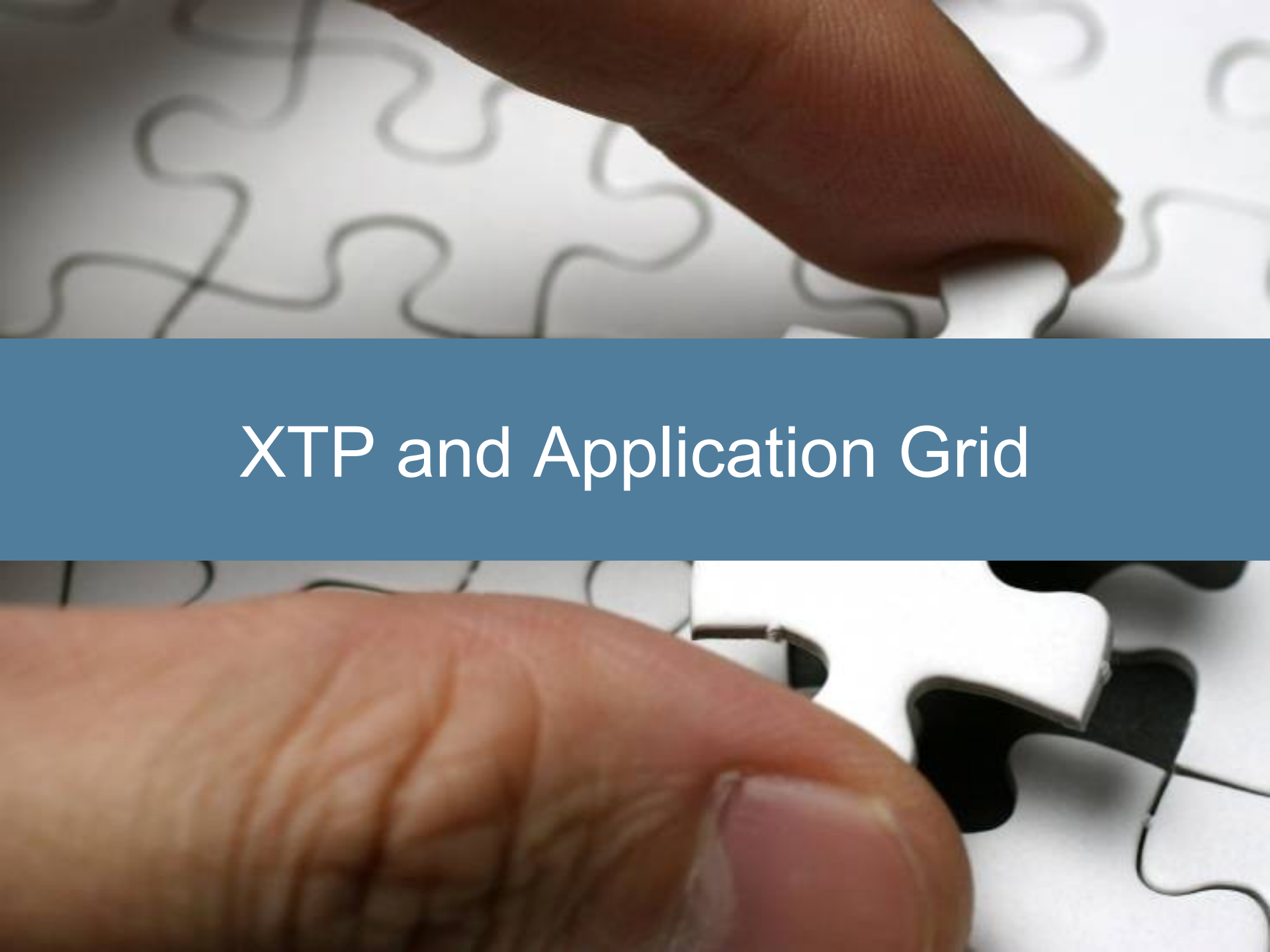
- **Representations:** A resource might be represented as XML, JSON, HTML, text, image, and so on. Instantiating representation from or to the resource is handled on the server. A client can pick its preferred representation via a URI parameter or the "Accept:" header.

REST Patterns 2/2

- Resources with multiple representations
  - GET /customers/1234 HTTP/1.1
    Host: example.com
    Accept: application/vnd.mycompany.customer+xml
  - GET /customers/1234 HTTP/1.1
    Host: example.com
    Accept: text/x-vcard
- Communicate statelessly
  - Cookies shouldn't be used to encode information that can be transferred by other, more standardized means

Processes and Resources





XTP and Application Grid

eXtreme Transaction Processing (XTP)

- Telco call setup and billing
- Online gaming
- Securities trading
- Risk management
- Online travel booking
- Traffic sensor data

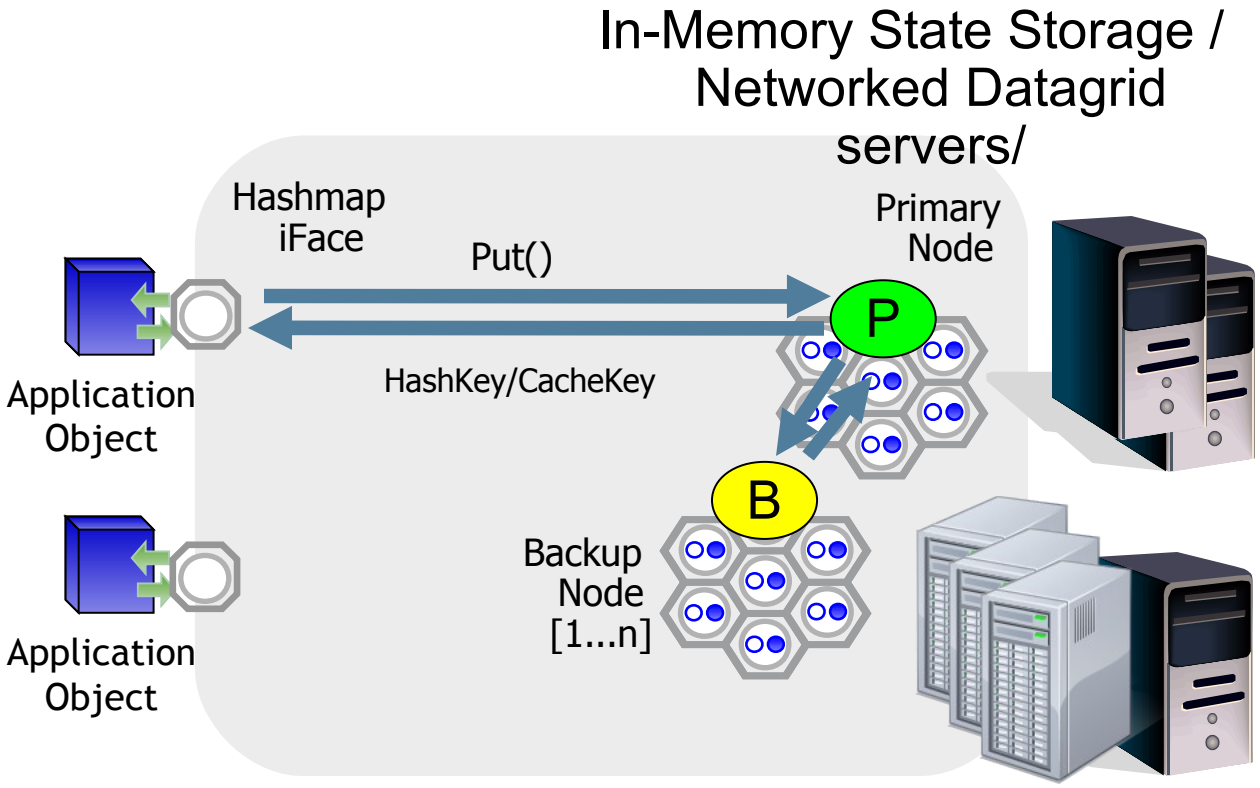
Rest of the World That Needs to Scale

- Portal apps that need to scale up to thousands of customers
- Backend systems never designed to handle load
- Peak processing times at certain times of year
  - Retail – holiday shopping
  - Health Insurance – Enrollment time
  - Product launch
  - Marketing programs

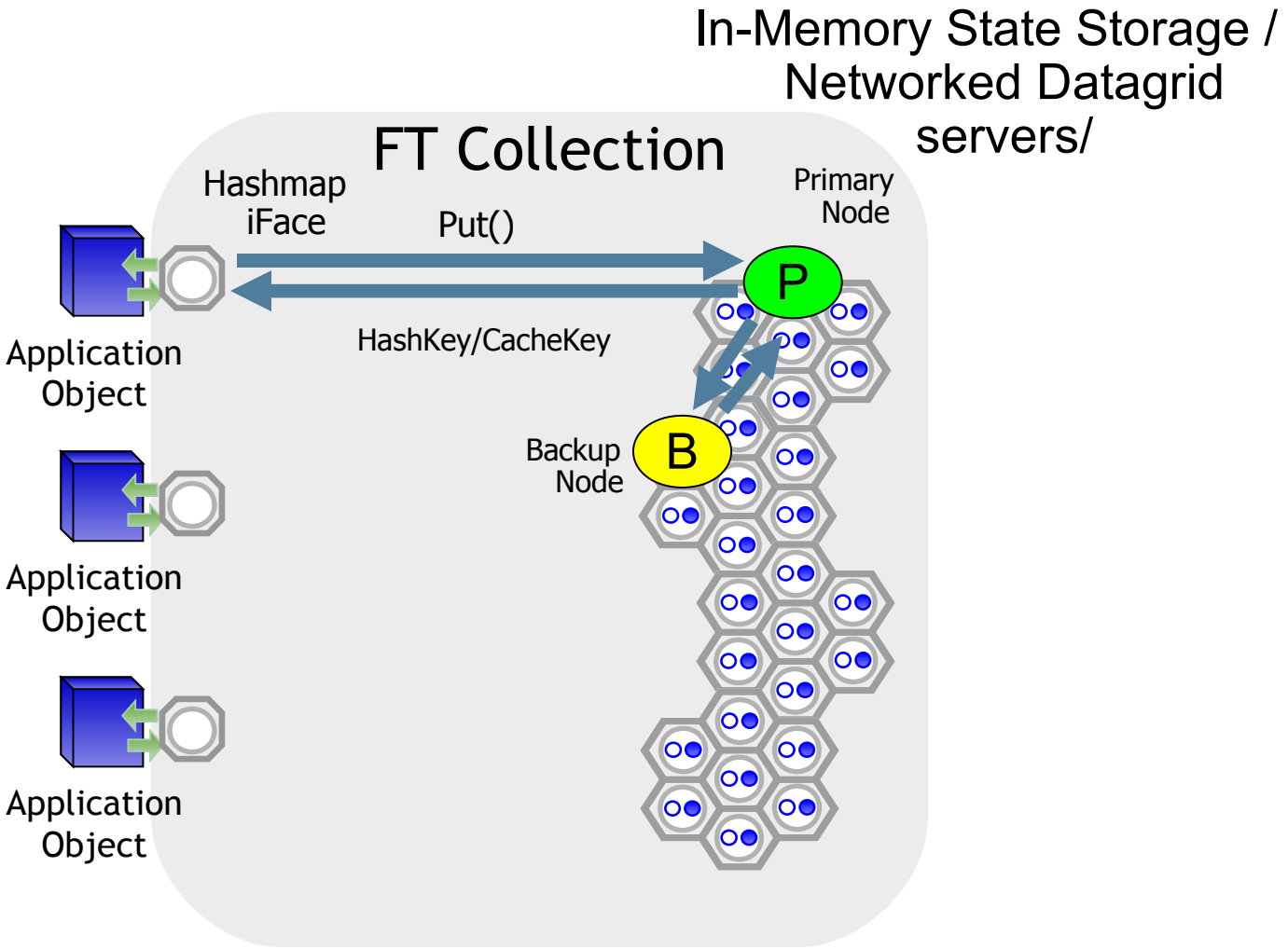
Application Grid

- Data Grid and Compute Grid
  - Linearly scalable shared memory and logic
  - Intelligent co-location and affinity between processing logic and Grid storage
  - Parallel Processing and Queries against in-memory native object data

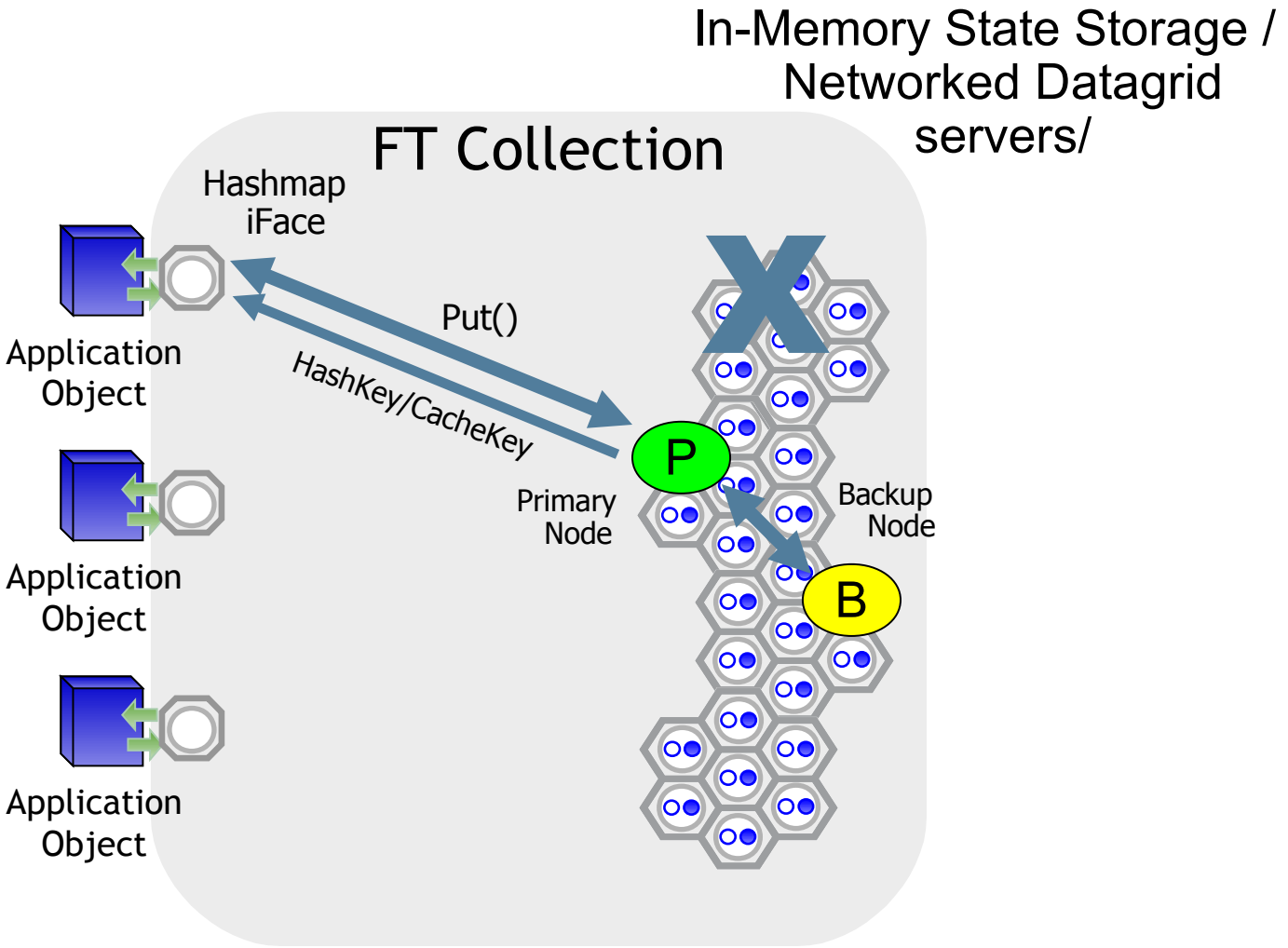
Fault-Tolerant Collection - Primary/Backup Synchronization



Fault-Tolerant Collection - Primary/Backup Synchronization

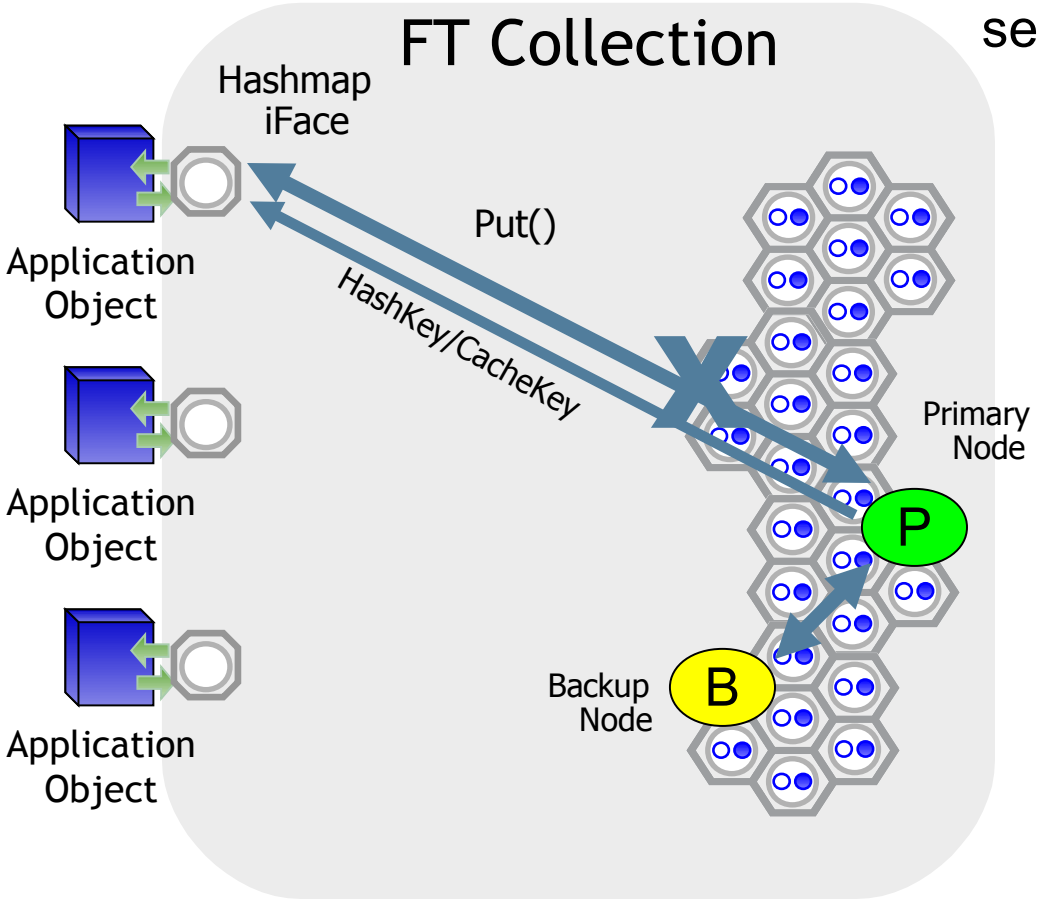


Fault-Tolerant Collection - Primary/Backup Synchronization



Fault-Tolerant Collection - Primary/Backup Synchronization

In-Memory State Storage / Networked Datagrid servers/



SOA and REST on App Grid Value Proposition

- Reduce cost of hitting back-end systems multiple times over the life of a session
- Improve response times for service requests and web apps while keeping load off backend systems
- Improved fault tolerance without sacrificing performance
- Ensuring predictable latency under increased sustained loads
- Achieving dramatic overall increase in performance and throughput
- Predictable scalability for **XTP**
  - Based on advanced software clustering techniques
  - Scales out linearly, whether 2 or 2,000 servers
  - Heterogeneous Environment
  - High-end / low-cost commodity hardware

Key Capabilities Highlighted in Case Studies and Patterns

- **Case Study: Shared Web Session State Across Multiple Portal Environments**
- **Seamless State Management for Stateful Services**
  - Without the need to enforce “state passing” design patterns across the organization
  - Reduced/Eliminate dependency on disk persistence and State Repositories
  - Without sacrificing HA
- **Pattern: SOA Caching in the Application Grid**
  - Hold data reliably in-memory
  - DB interaction Patterns: sync write thru, asynch write behind, multi-level cached read
- **Pattern: Inline Service State Cache**
  - Sharing state data in-memory across service boundaries
- **Case Study and Pattern: Service-Result-Cache**
  - To offload loads on backend services and SOA Infrastructure use side cache to cache results of service calls, c.f. Web cache
  - Without sacrificing consistency and programmability
- **Case Study: Multi-Dimensional graphs of data in the Grid**
  - Eliminate Bulky Nested XML
- **Pattern: XML Grid –**
  - Efficiently Handle / Process XML Payloads within SOAs
  - Increasing throughput and decreasing latency
- **Pattern: Not Your MOM’s Bus**
  - Reliable and sharable state caching ensures QoS without pushing large JMS, SOAP, or REST messages across the bus

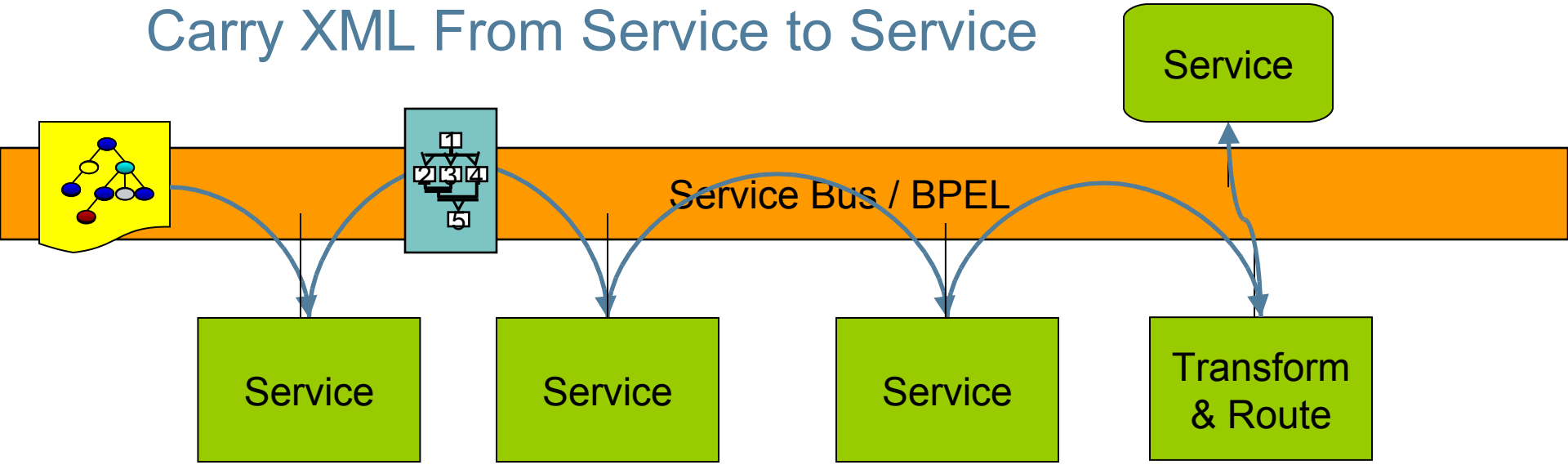
Challenges for Scalability

Challenges for SOA Scalability

- Performance tuning of Monolithic Applications is a relatively known science
- Performance throughput testing of individual services can also be done in isolation
- Breaking up monolithic applications into loosely coupled service level components and putting them together in new ways to form composite apps introduces new challenges.
  - Boundary costs
  - Sharing data reliably

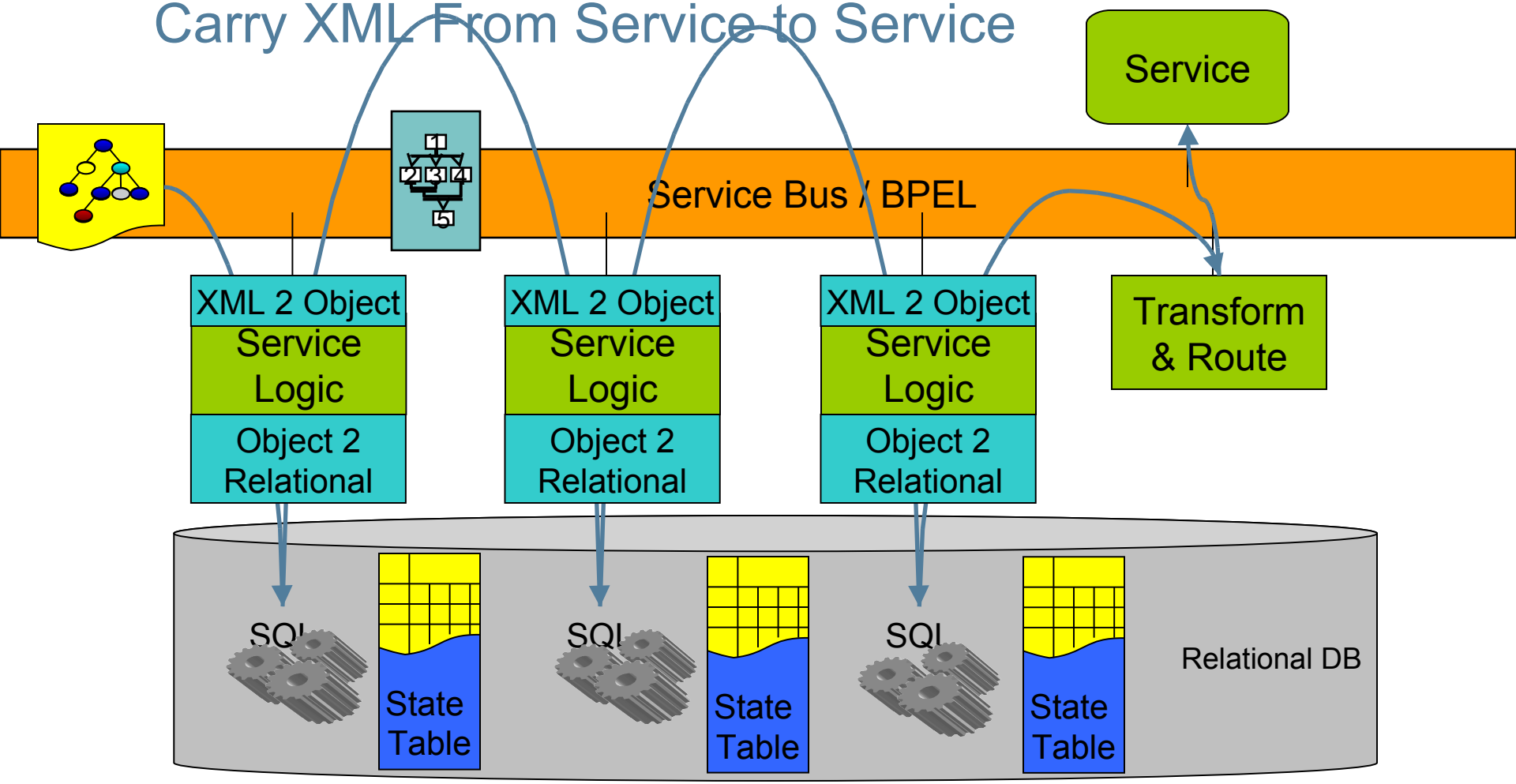
Traditional SOA Data Payload Model

Carry XML From Service to Service



Traditional SOA Data Payload Model

Carry XML From Service to Service

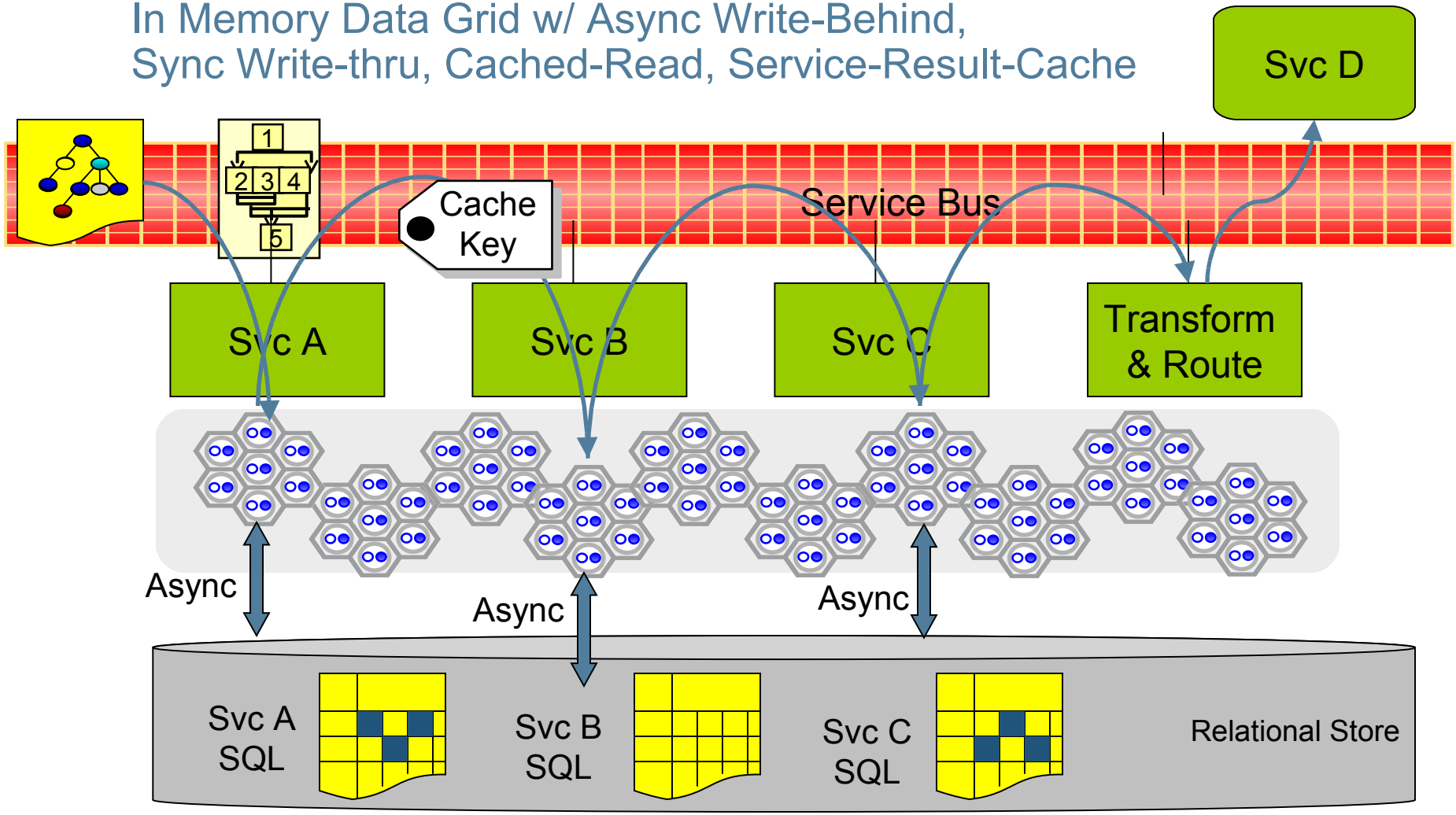


What does the scalable architecture need to ensure

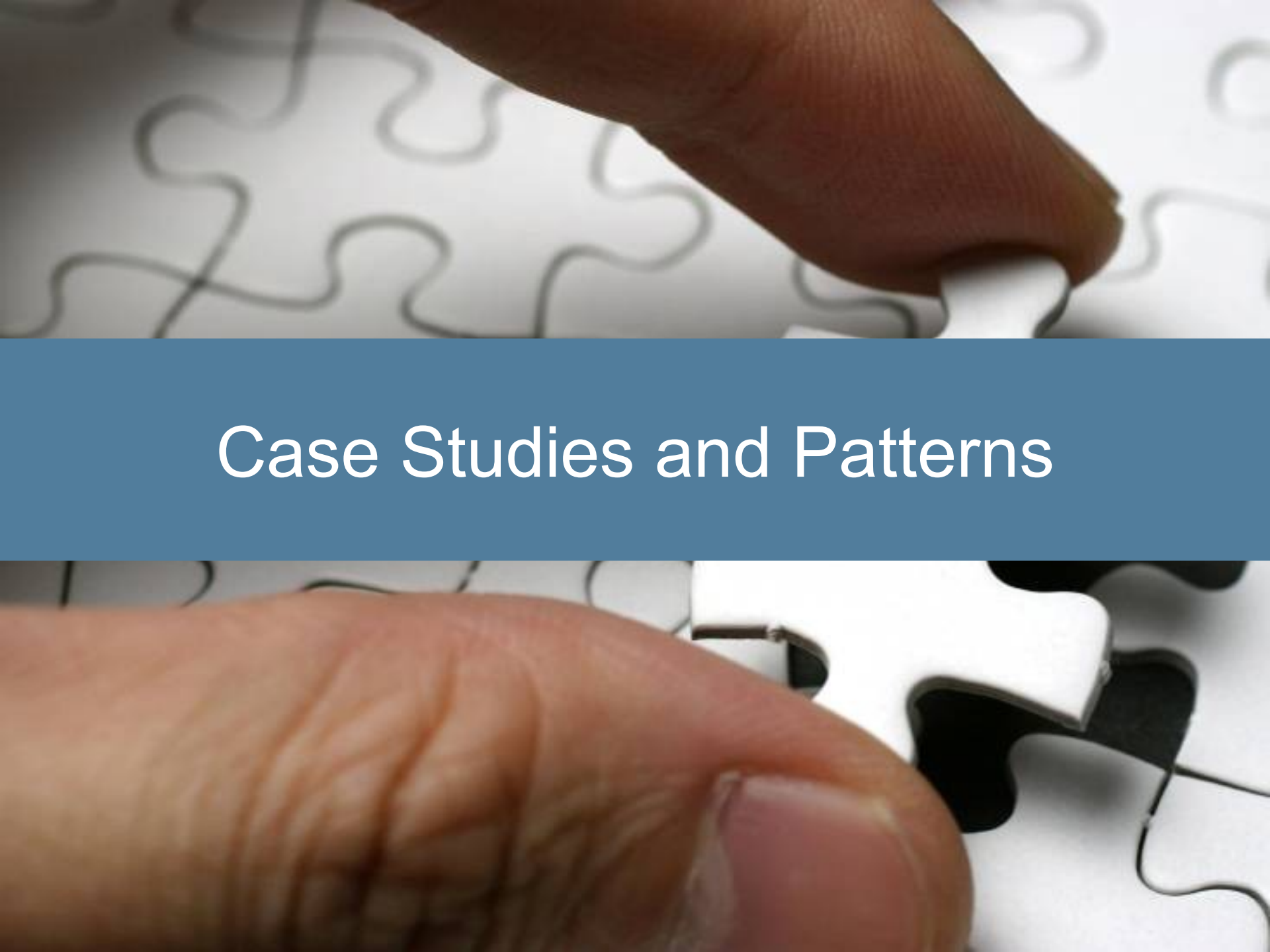
- Avoid processing where possible
- Avoid I/O where possible
- Avoid serialization and de-serialization
- Avoid sending large documents over network
- Avoid distributed file schemes as your app scales up
- Avoid driving increasing traffic to backend systems
- Architect for performance from the beginning aka *not* throwing hardware at the problem!

SOA Caching Using a Datagrid

In Memory Data Grid w/ Async Write-Behind, Sync Write-thru, Cached-Read, Service-Result-Cache



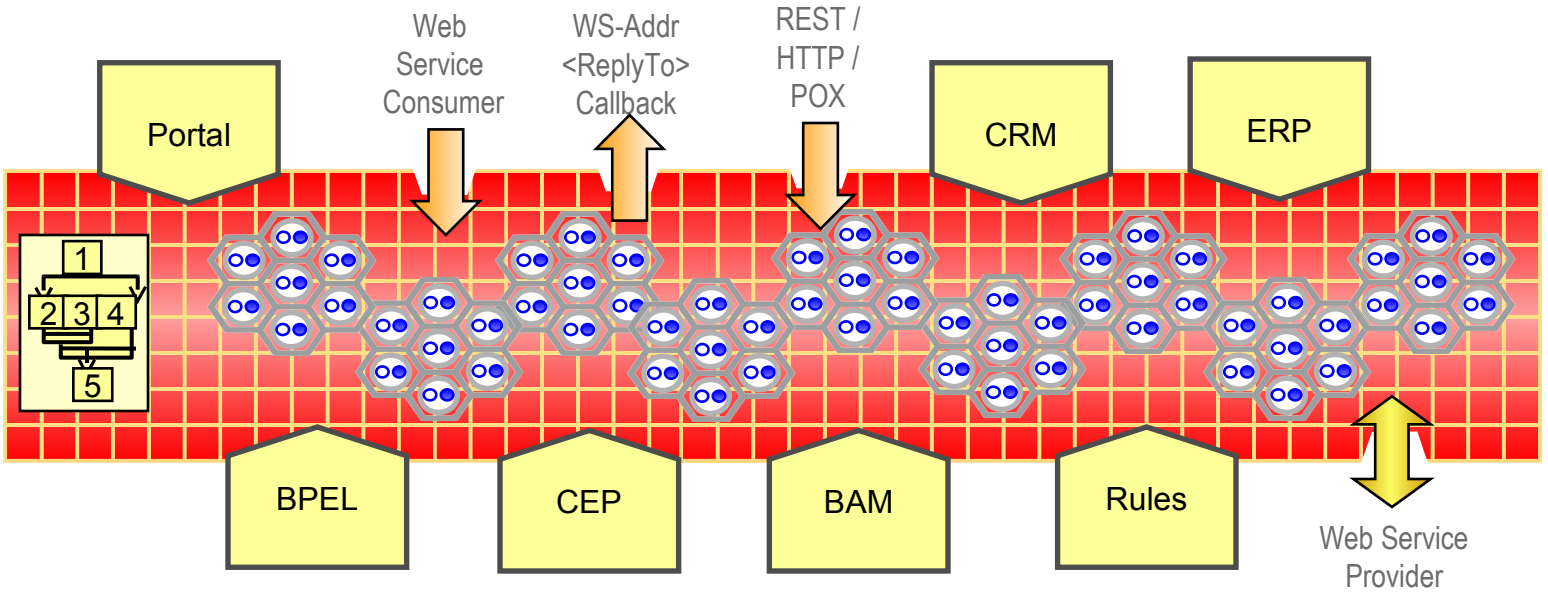
Cache Key = "Claim check" carrying the key to data cached in the grid



Case Studies and Patterns

Case Study: Not Your MOM's Bus

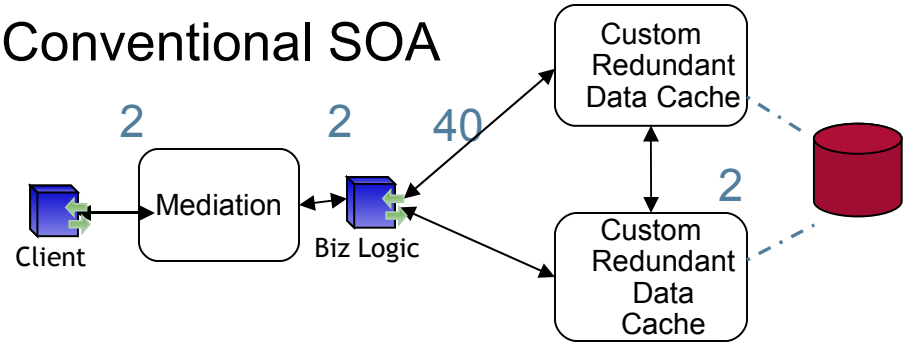
Why send it when its already there?

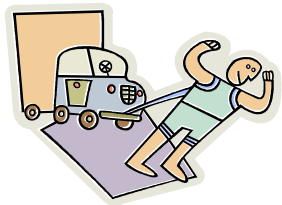


Reliable and sharable state caching ensures QoS without pushing large JMS or SOAP messages across the bus

Case Study: Not Your MOM's Bus

Conventional SOA

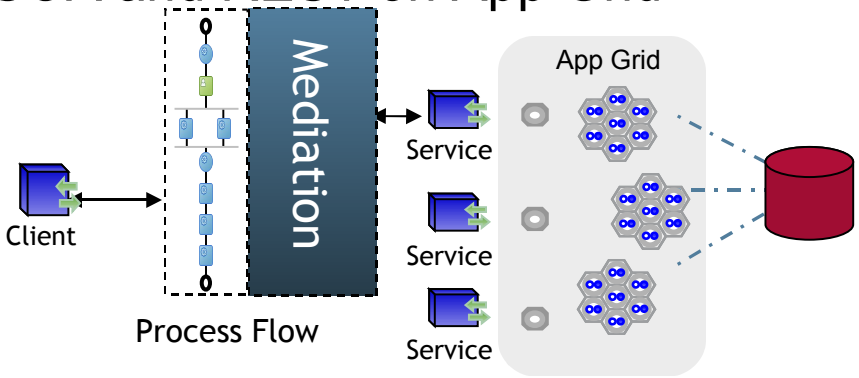


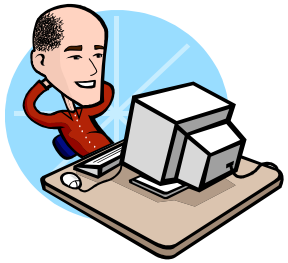


46 network hops/data serialization steps

46 Boundary Crossings

SOA and REST on App Grid



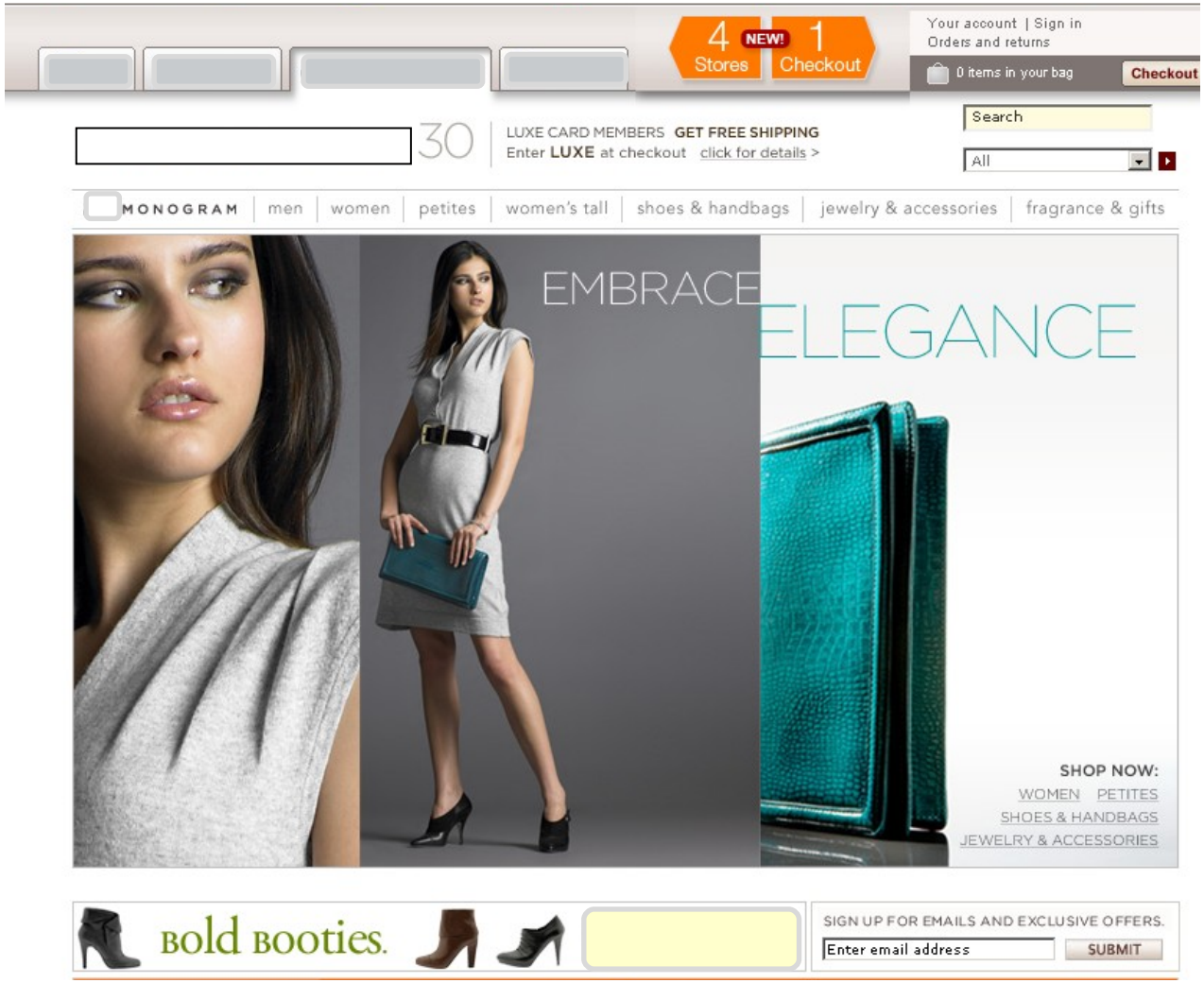


5 - 7 network hops/data serialization steps

Horizontally scalable

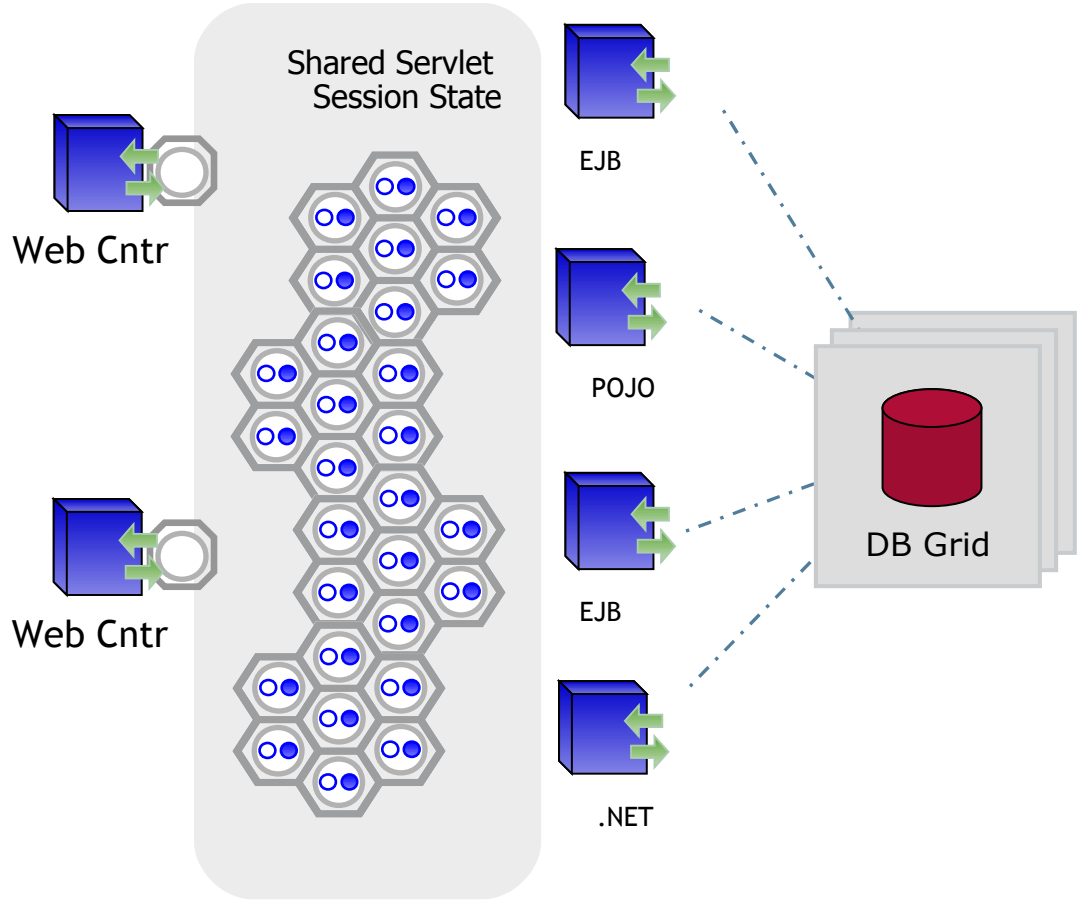
5-7

Case Study: Multiple Store Fronts - Sister Tabs



*OOW2008 S299392 - Beyond Performance: Pushing Transaction…

Pattern: State Management using App Grid: Shared Web Session State



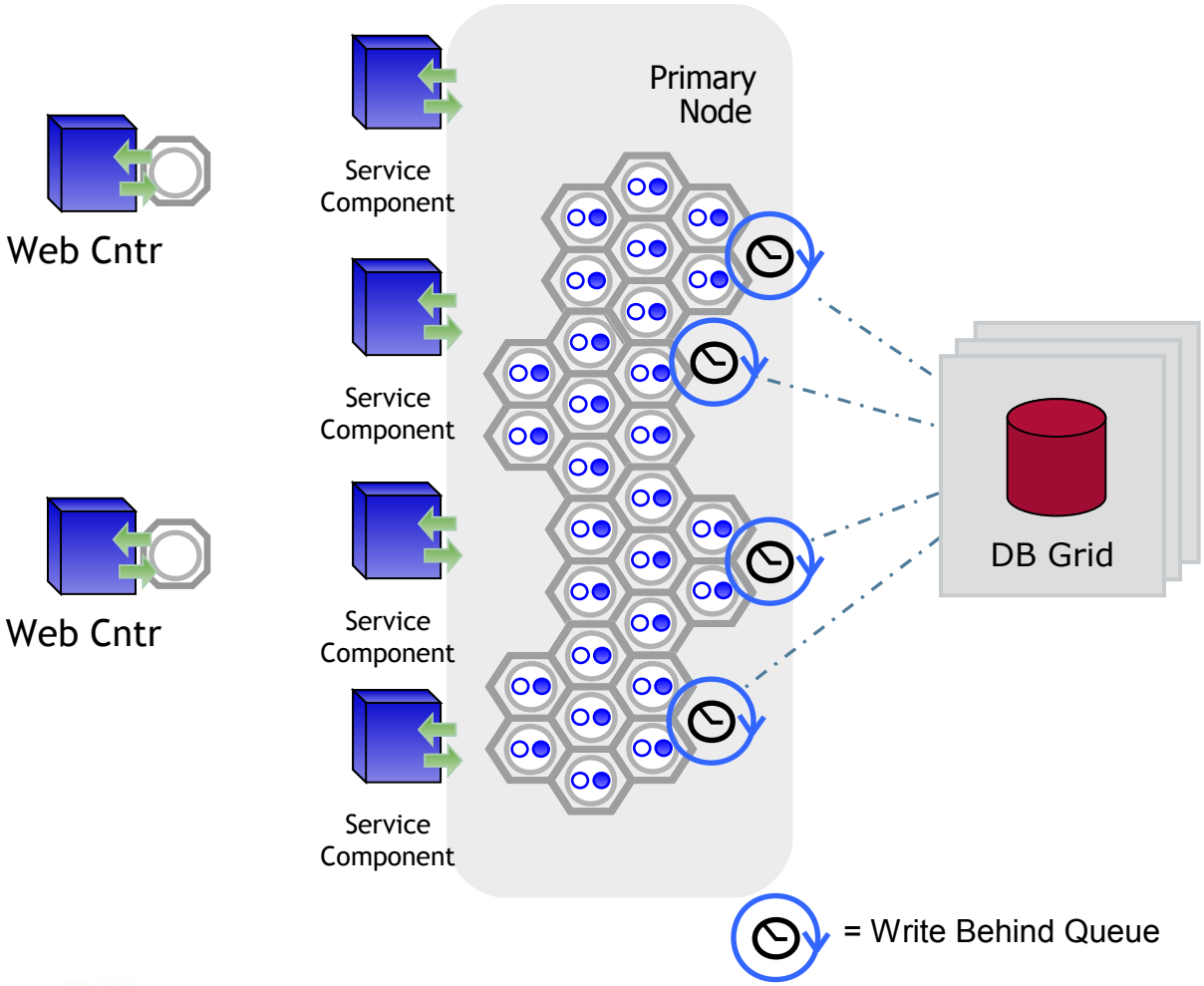
Business logic: Traditional middle tier EJB

Data Grid: Web Containers share servlet session state via data grid

Database:

Use for storing, managing transactions

State Management with RESTful SOA on Application Grid: Shared Service State Data



Business logic:

Services manage their state in the data grid

Data Grid:

Use for managing transient state (read and write)

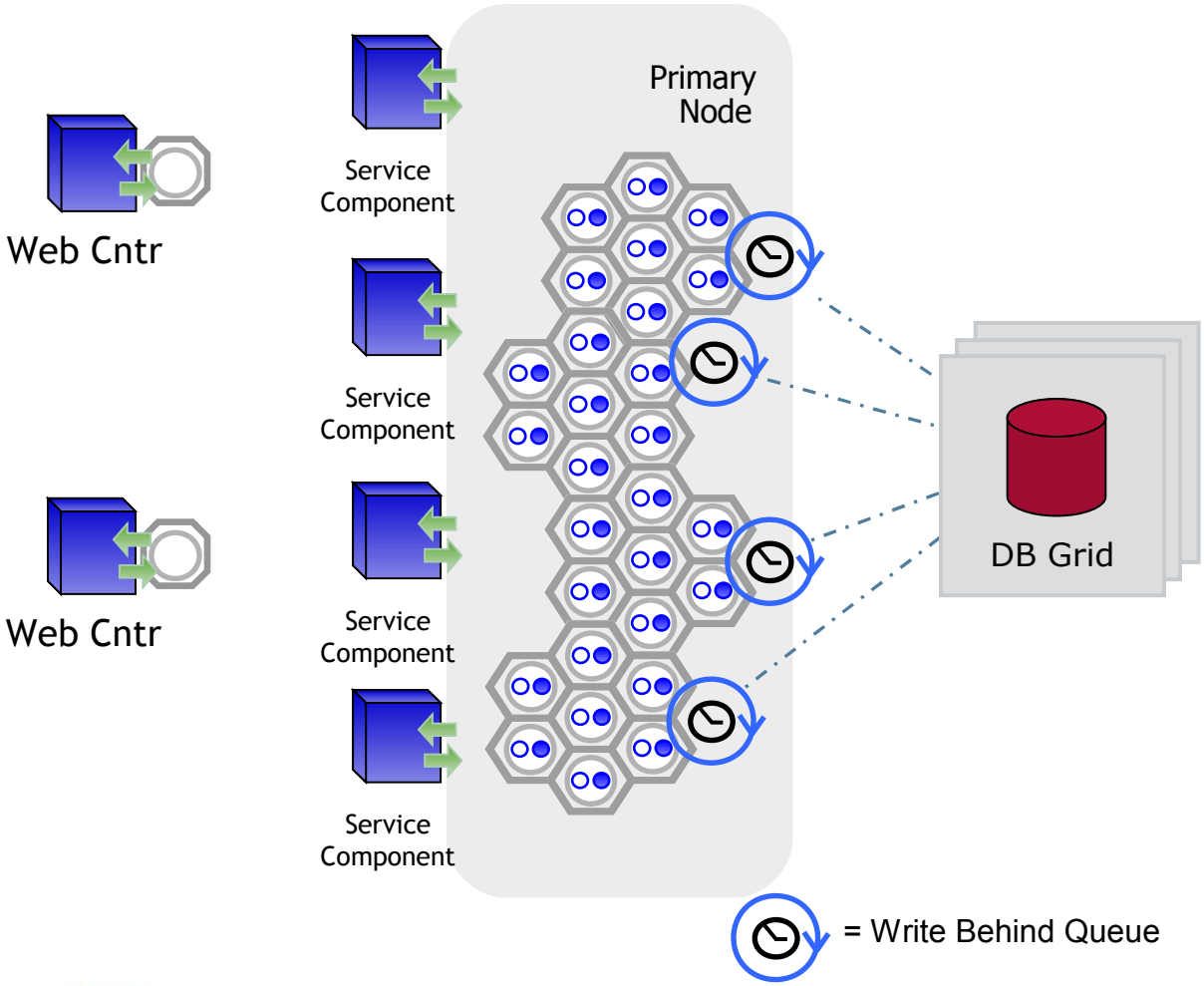
Database:

Use for storing, managing transactions

Write through for state information from data grid

Similar architecture, similar benefits!

DB Interaction: Asynchronous Write-behind



Business logic:

Services manage their state in the data grid

Data Grid:

Use for managing transient state (read and write)

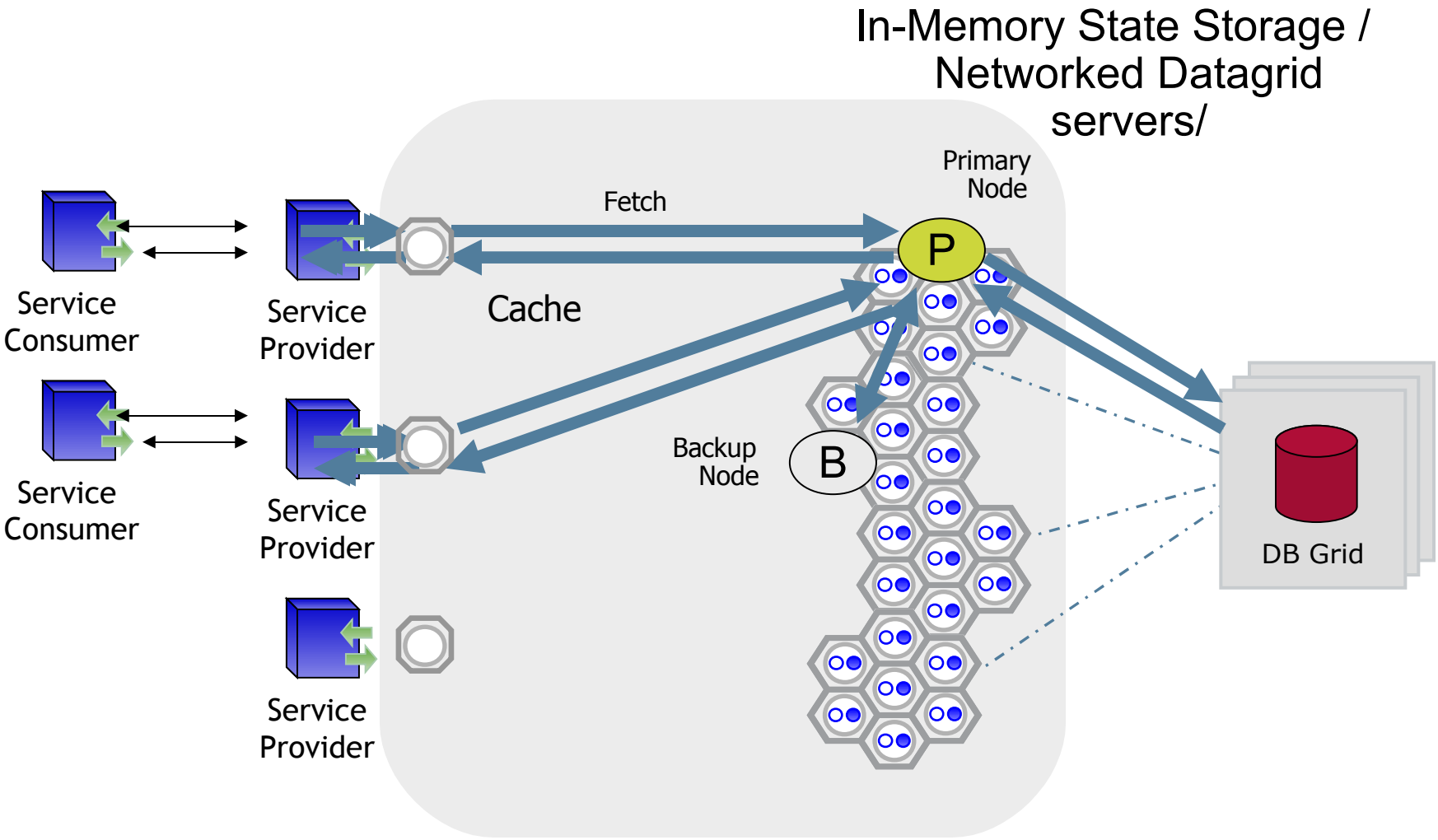
Database:

Use for storing, managing transactions

Write through for state information from data grid

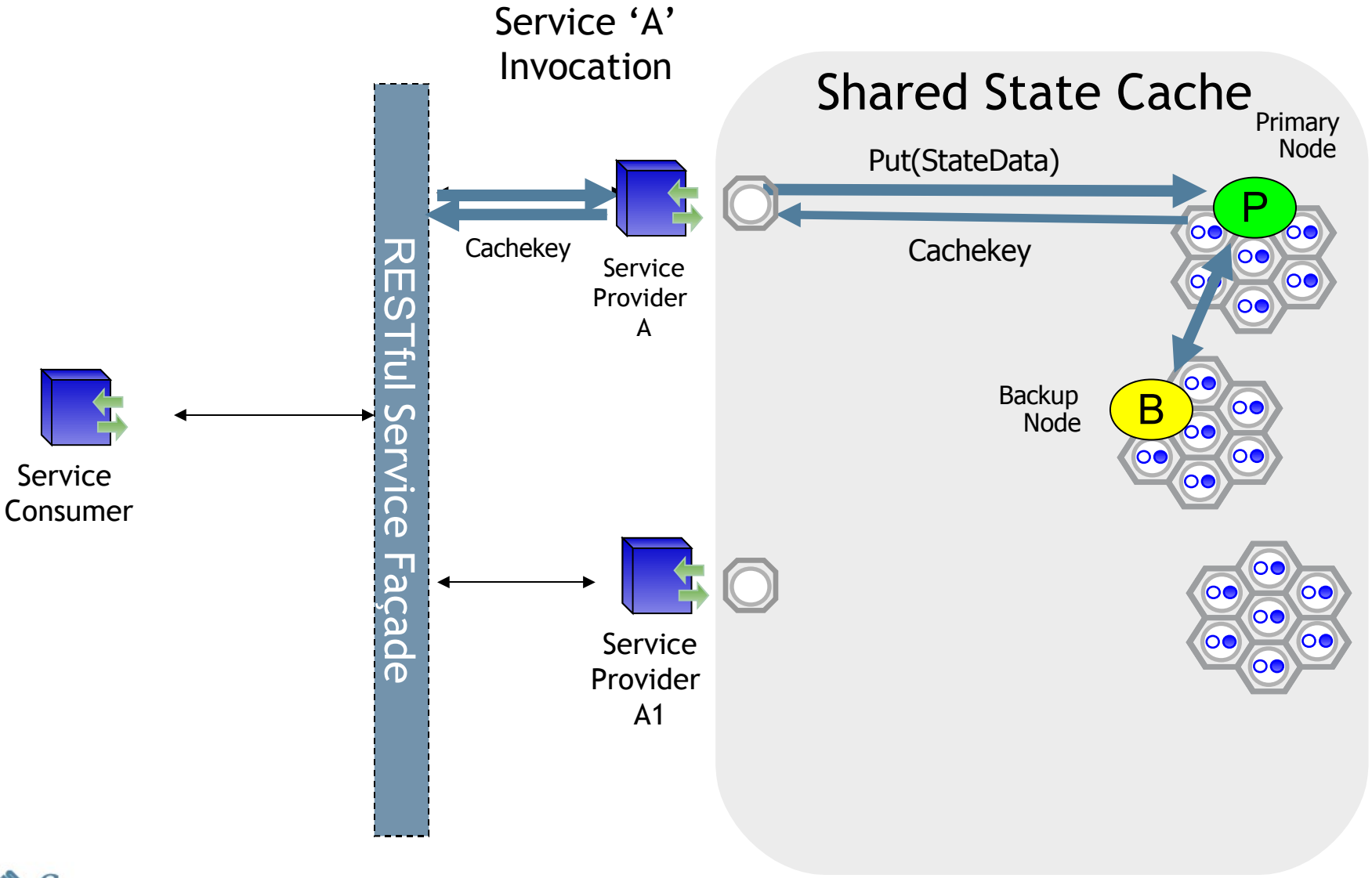
Similar architecture, similar benefits!

DB Interaction: Near-Cache Optimization



400X Improvement on Reads, 40X Improvement on Writes

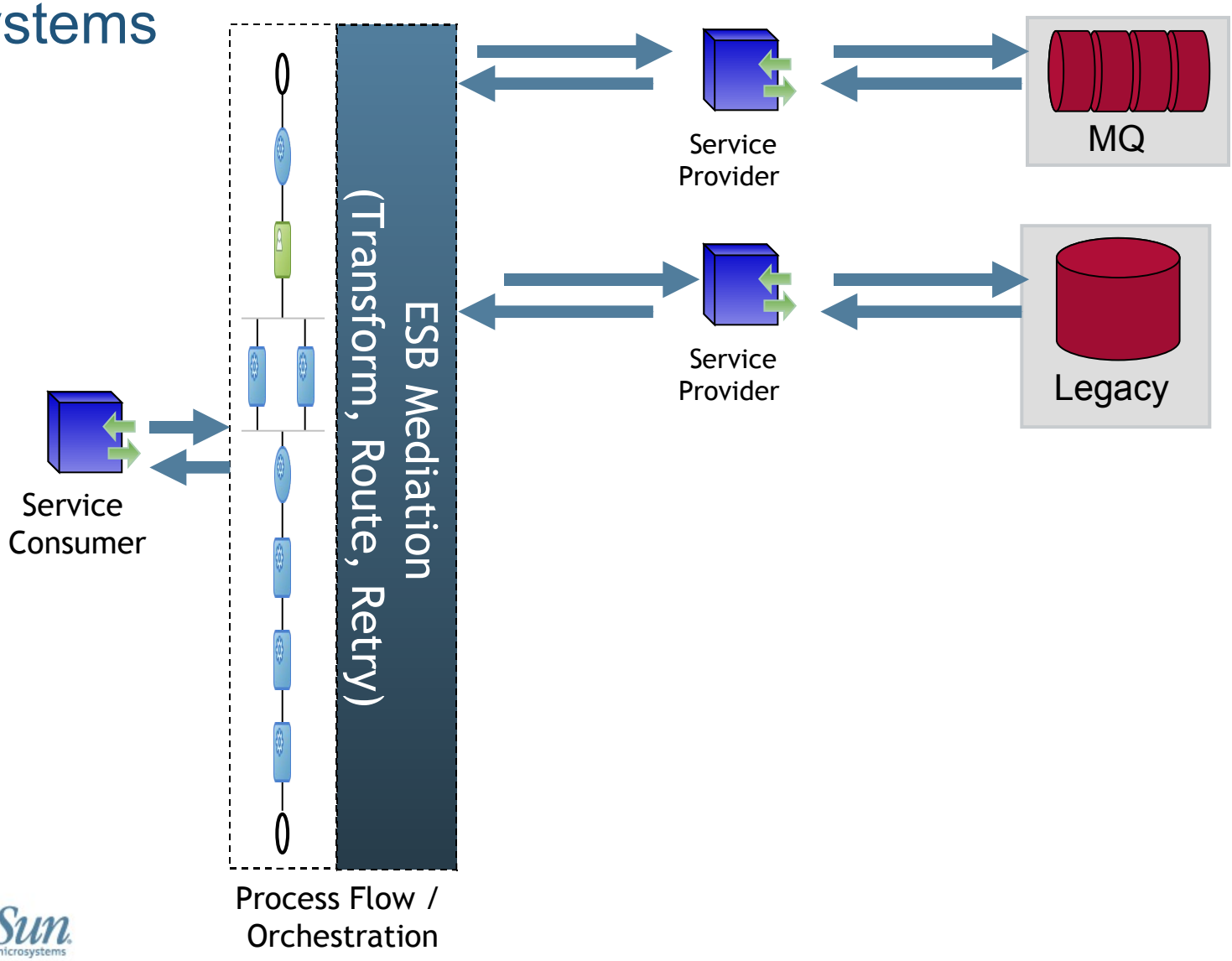
Service Pattern: Inline Shared State Cache



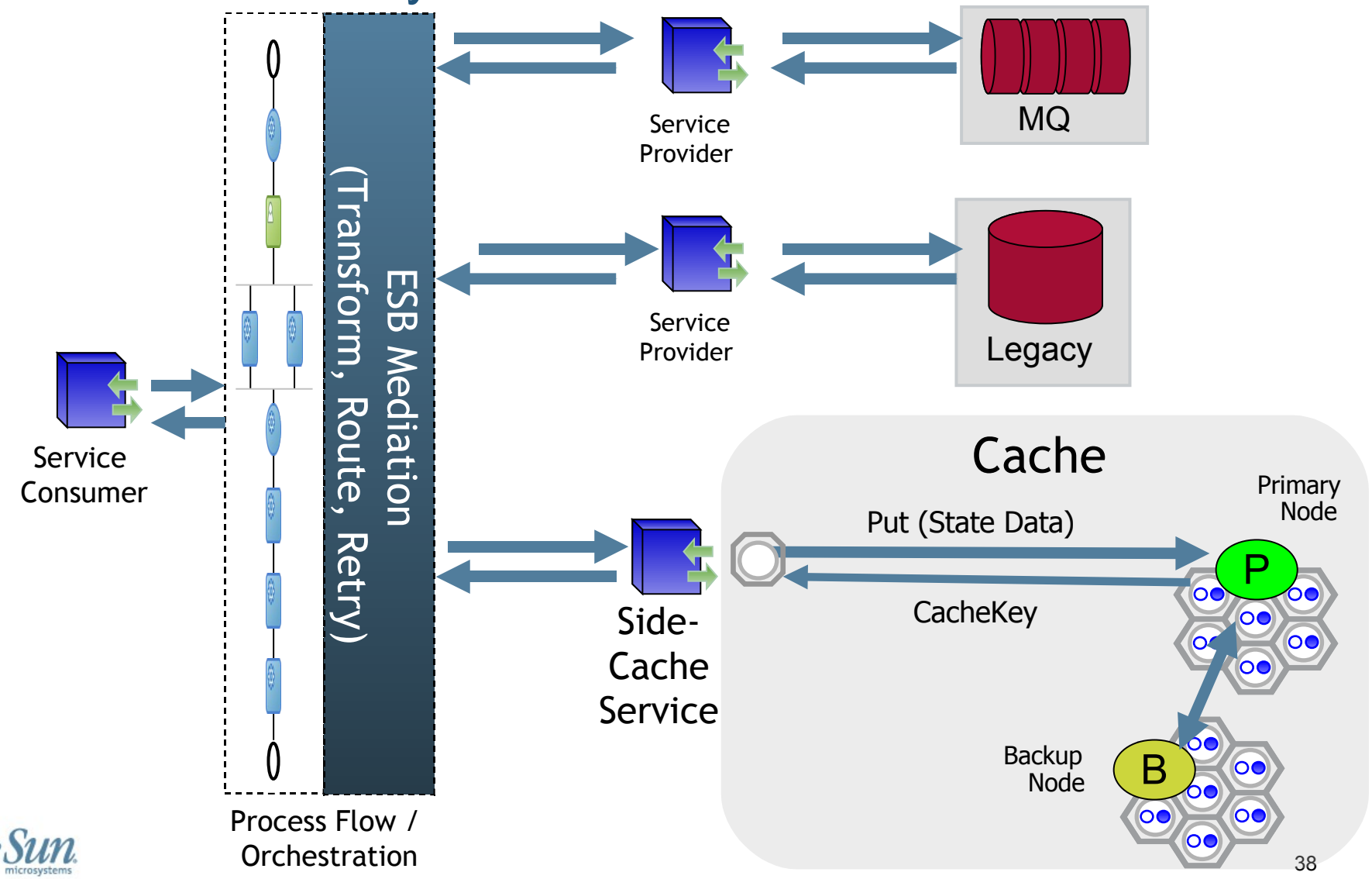
Case Study: Pharmaceutical Intranet Portal

- Scenario
- Internal sales portal
  - Gathers information about physicians to sell to
  - Requests to backend systems sometimes yields10 web pages of data
  - 3 seconds on a single backend service call thru an MQ queue to a mainframe
  - Solution
- Cache Results of first call using Service Result Cache pattern
  - Offloading expensive backend calls
  - Reduction in mainframe traffic
  - Improved user experience

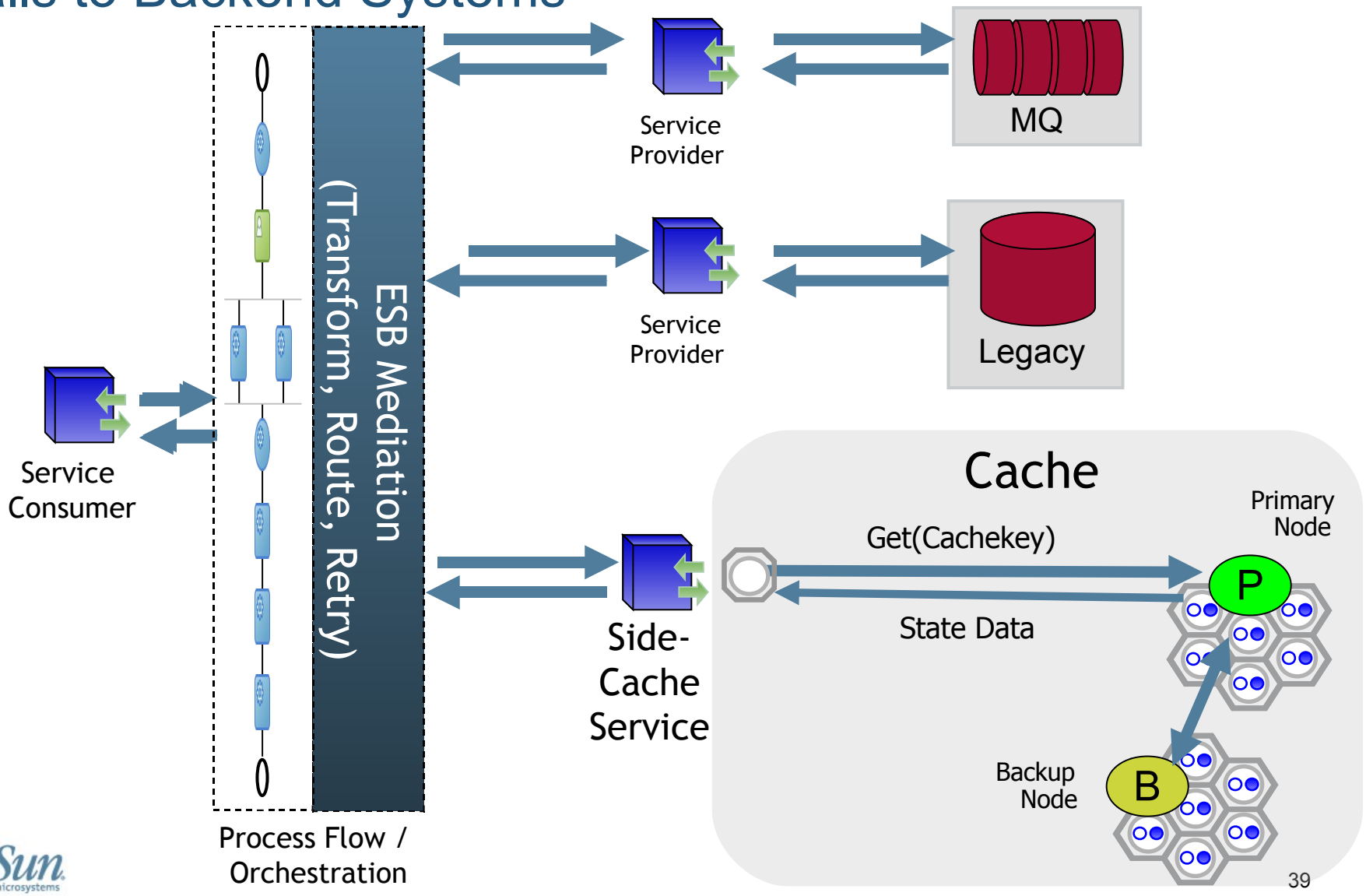
Before | Caching Results to Service Calls to Backend Systems



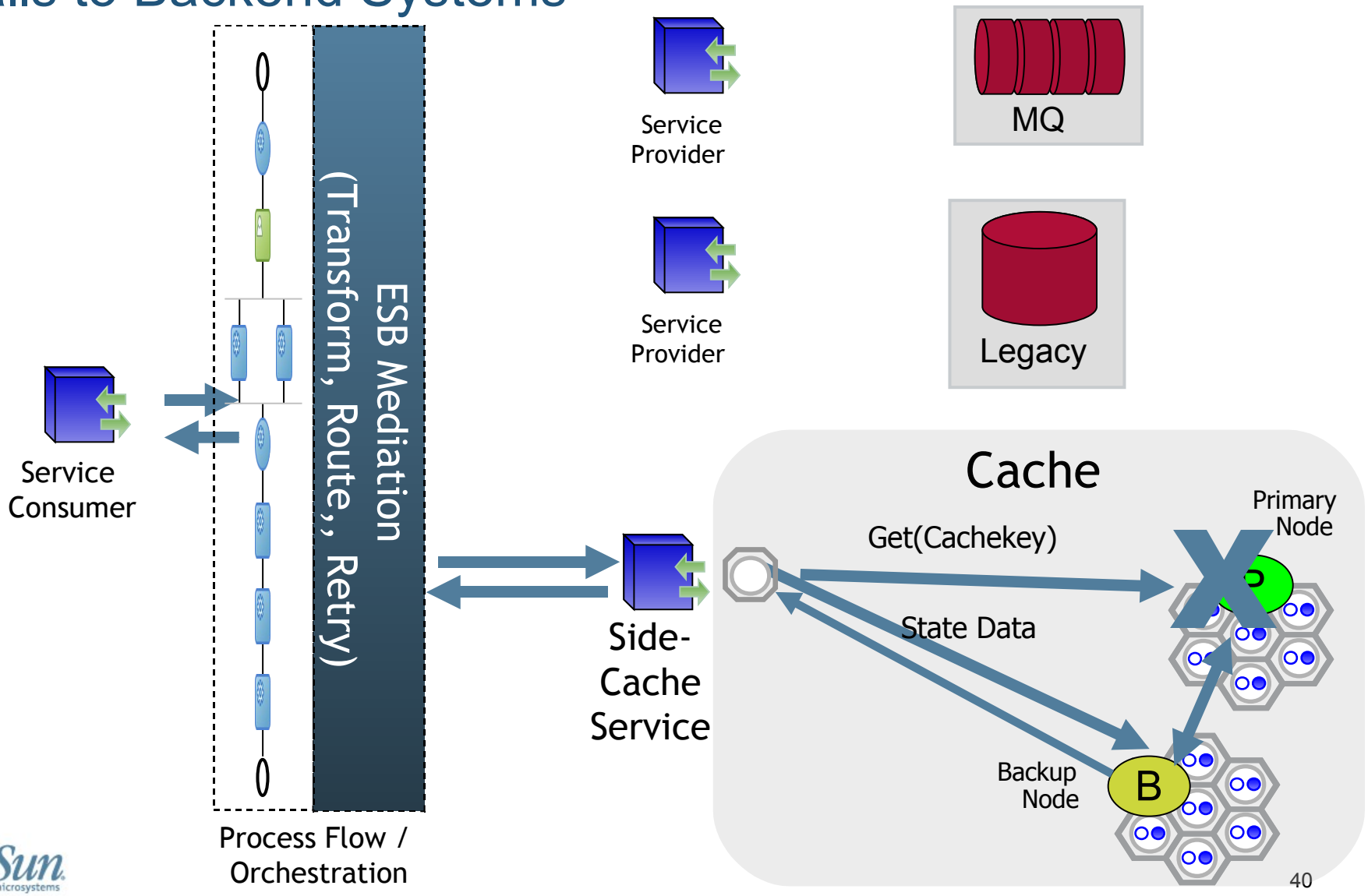
Service-Result-Cache | Caching Results to Service Calls to Backend Systems



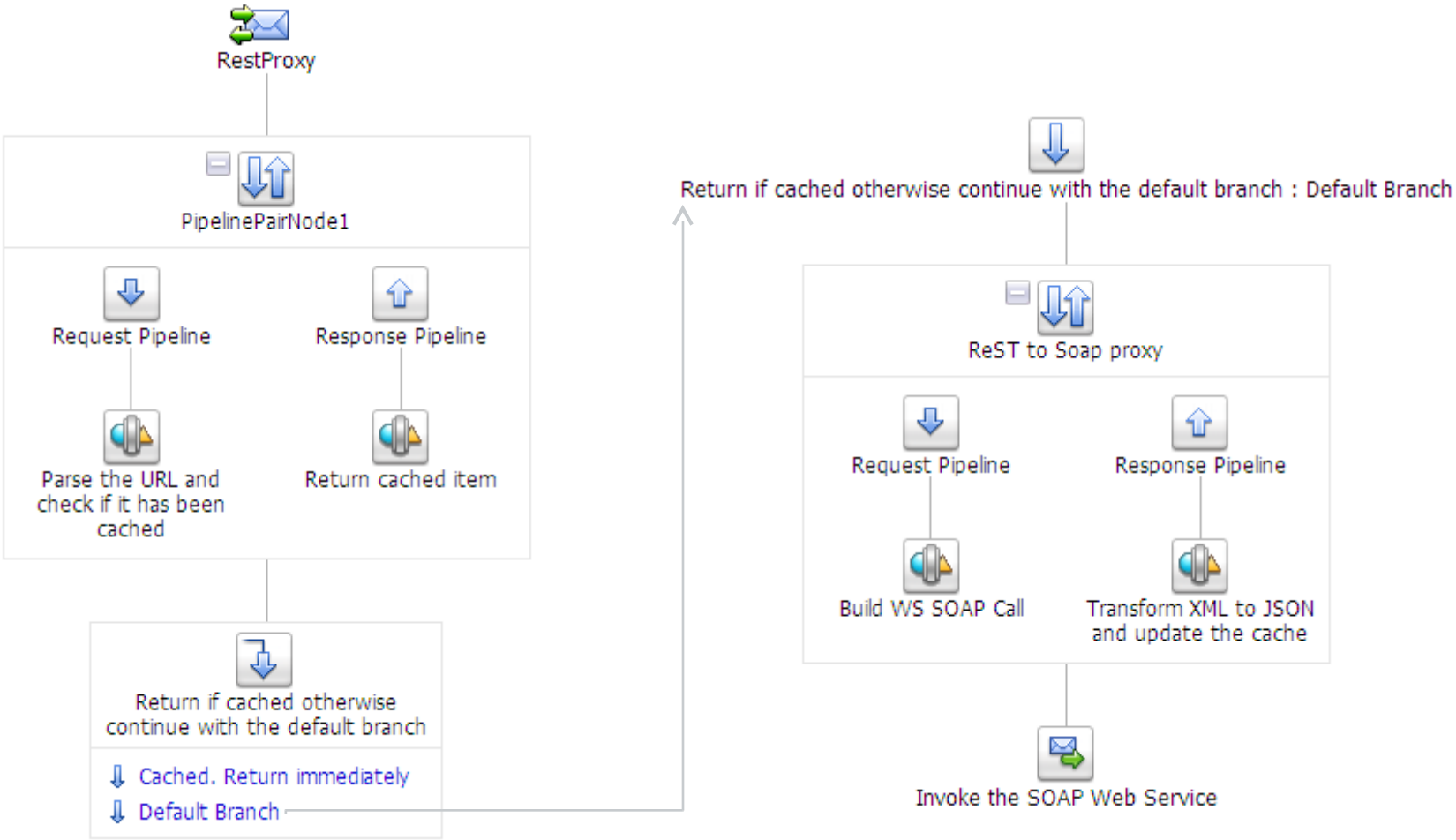
Service-Result-Cache | Caching Results to Service Calls to Backend Systems



Service-Result-Cache | Caching Results to Service Calls to Backend Systems

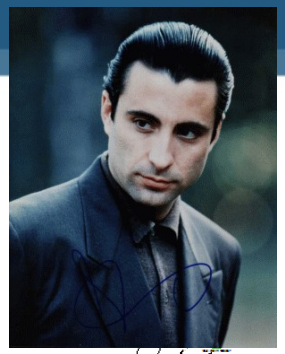


Variation: Restful SOA Datagrid - ESB pipeline



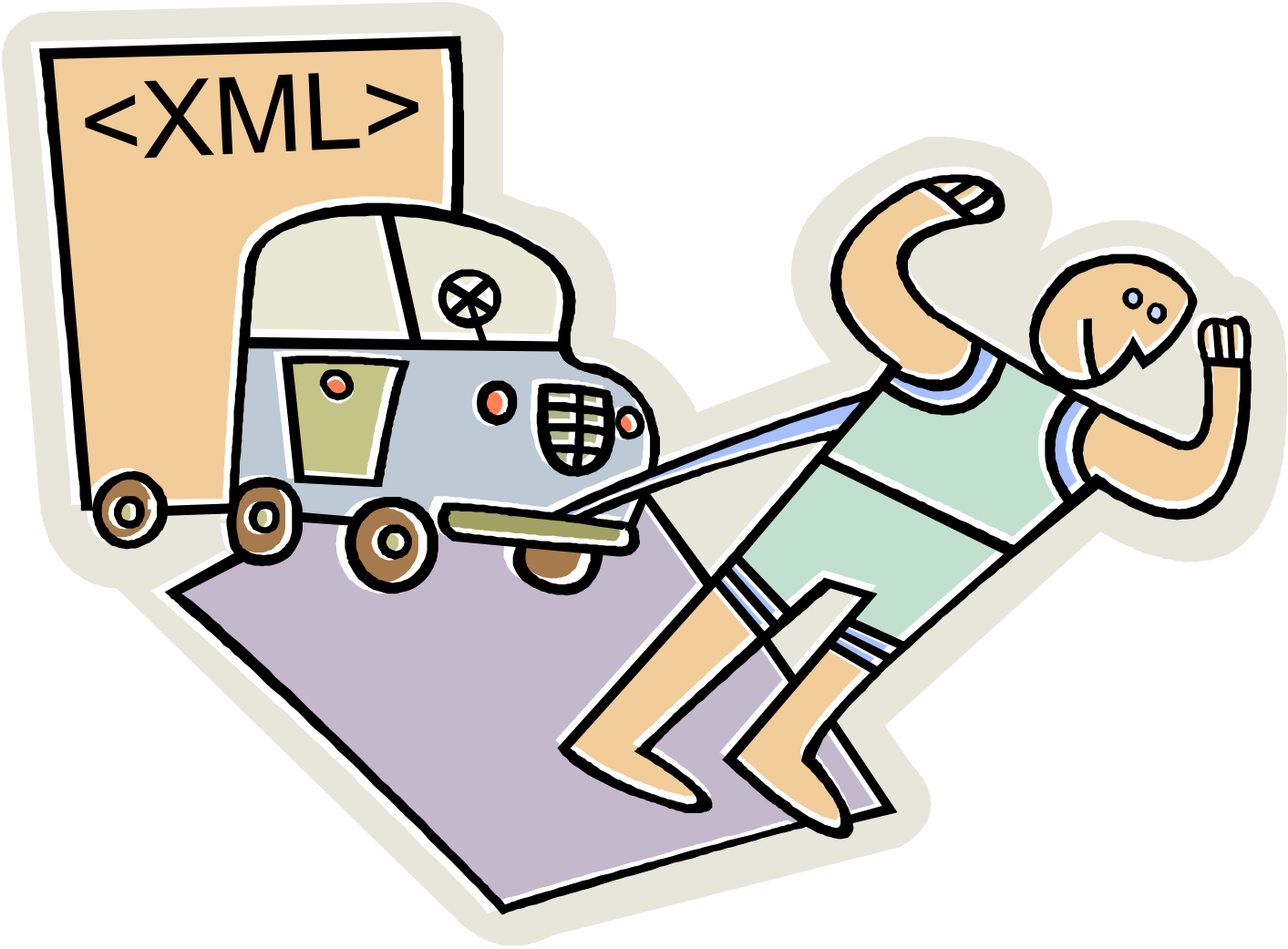
Case Study: Ministry of Justice Anti Money Laundering System

- Use Case
  - Track illicit money laundering activities
  - Build Network of people, criminal organizations, police records
  - Identify patterns of suspicious cash transactions
- Problem
  - Build large dependency Graphs
    - Crime families, associations
  - 10K – 20K XML messages
- Solution: SOA on App Grid
  - BPEL for Orchestration, Coherence for State
  - Caching results of WS calls
    - Reference data from external agencies, banks, etc
- XML Nodes fragmented across Coherence
  - Converted to java objects
  - Can be serialized on demand




AL CAPONE

Pattern: XML Grid



Pattern: XML Grid

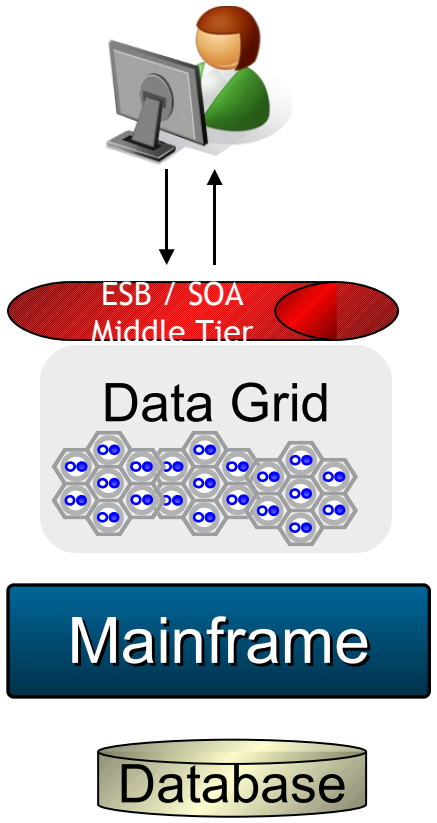
- Problem statement - “Large” XML
  - Serializing, transporting, consuming, parsing, manipulating of large XML documents is costly
    - Consumes resources
    - Introduces latency due to bottlenecks
    - Inhibits scalability
- Potential solutions
  - Streaming solves part of the problem, however
    - Still may need to fully materialize document in memory
    - Still need to serialize, deserialize, and transport
    - No easy way to do CRUD operations

Pattern: XML Grid

- Solution: XML in the App Grid
  - Use streaming parse to read document into the grid
  - Break it up into its constituent parts
  - Use JAXB to convert to Java Objects
  - Use distributed, parallel query and aggregation capabilities of App Grid to query and update objects in the grid
  - App Grid scales out horizontally simply by adding new JVMs to the cluster.
  - No limit to the size of the XML document or the number of them
  - Article in SOA Journal this month http://soa.sys-con.com/node/976701

The Benefits, Quality of Service Goes Up, Costs Go Down!

- **Improved response time**
  - In-memory Cache provides sub-second response time
- **Improved scalability**
  - RESTful SOA on an App Grid will scale linearly with increased demand
  - As demand increases, new grid servers are easily added
- **Improved availability**
  - Linear scalability protects against unexpected shutdowns from sudden spikes in volume
  - If the backend systems go down, the middle tier can still service transactions against cached data
- **Reduced mainframe costs**
  - Eliminate millions in annual mainframe costs



For More Information

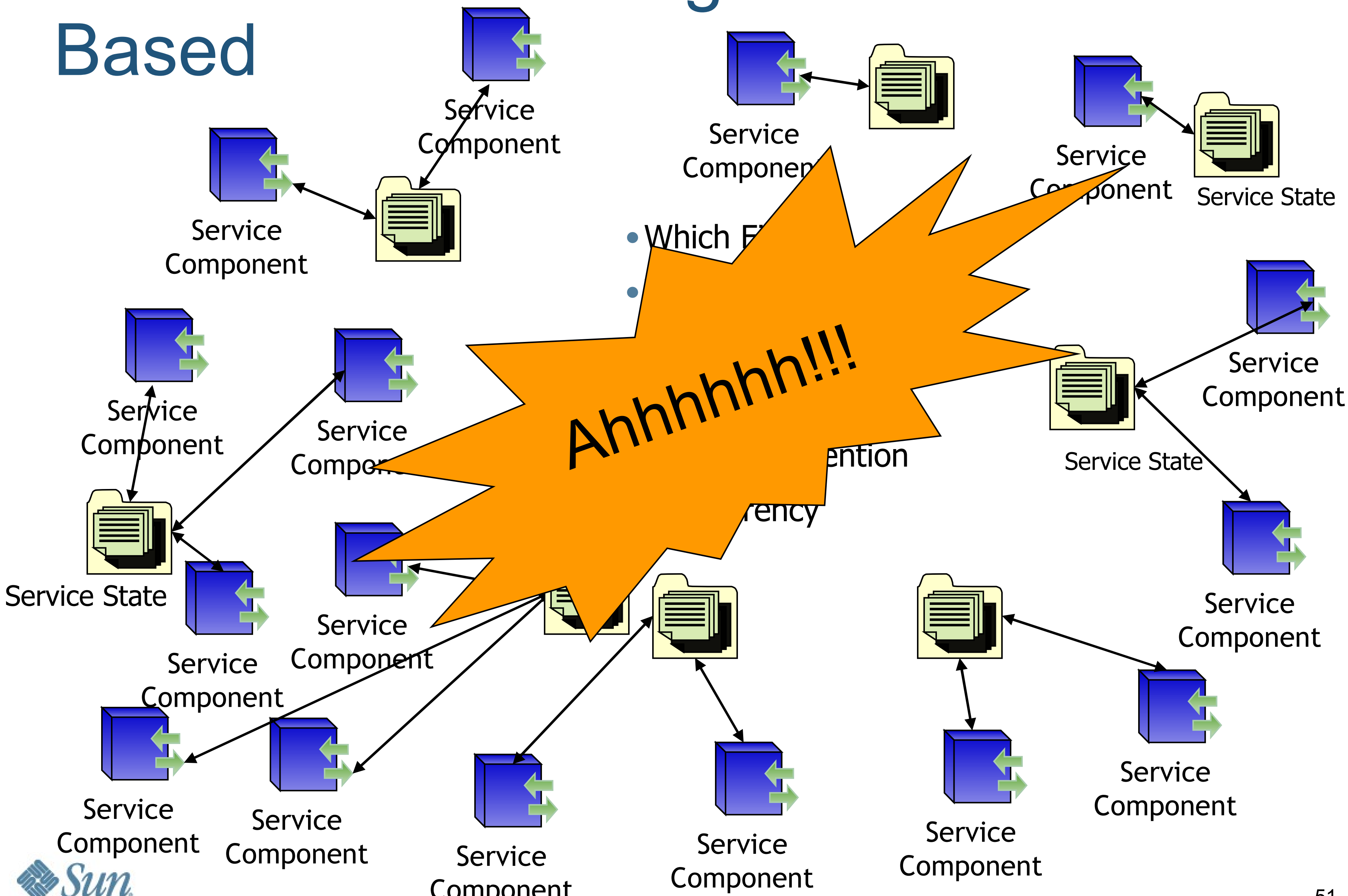
- XML Grid
  - http://soa.sys-con.com/node/976701
- Making SOA Leaner and Cleaner to Maximize Products
  - http://www.ebizq.net/topics/soa/features/10393.html
- David Chappell Blog
  - http://blogs.oracle.com/davidchappell
- Restful SOA Datagrid
  - http://www.slideshare.net/EmilianoPecis/the-restful-soa-datagrid-with-oracle

Appendix

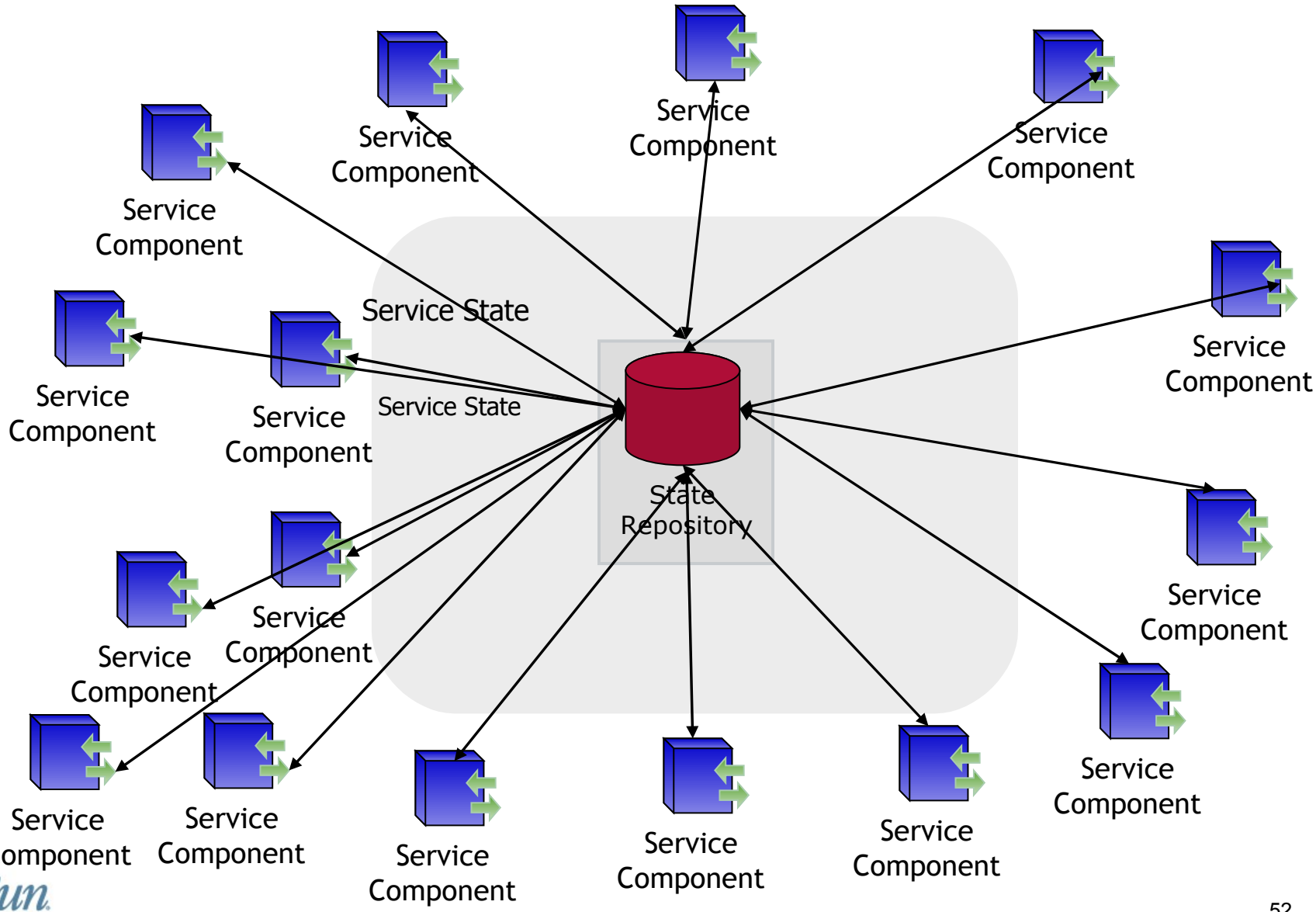
SOA Grid Patterns

- SOA Grid: Basic Patterns for State Management Using Middle Tier Caching
  - Fault Tolerant Collection
  - Inline Service State Cache
  - Side Cache
  - Stateful Service Load Balancing
  - Stateful Service Availability/Failover
- Advanced State Management Patterns
  - State Based Notification
  - Asynch DB Update
  - Near Cache Optimization
  - Sticky Load Balancers and Cached Service State
  - Business Logic Affinity
  - Claim Check/Pass-by-ref/Deferred State

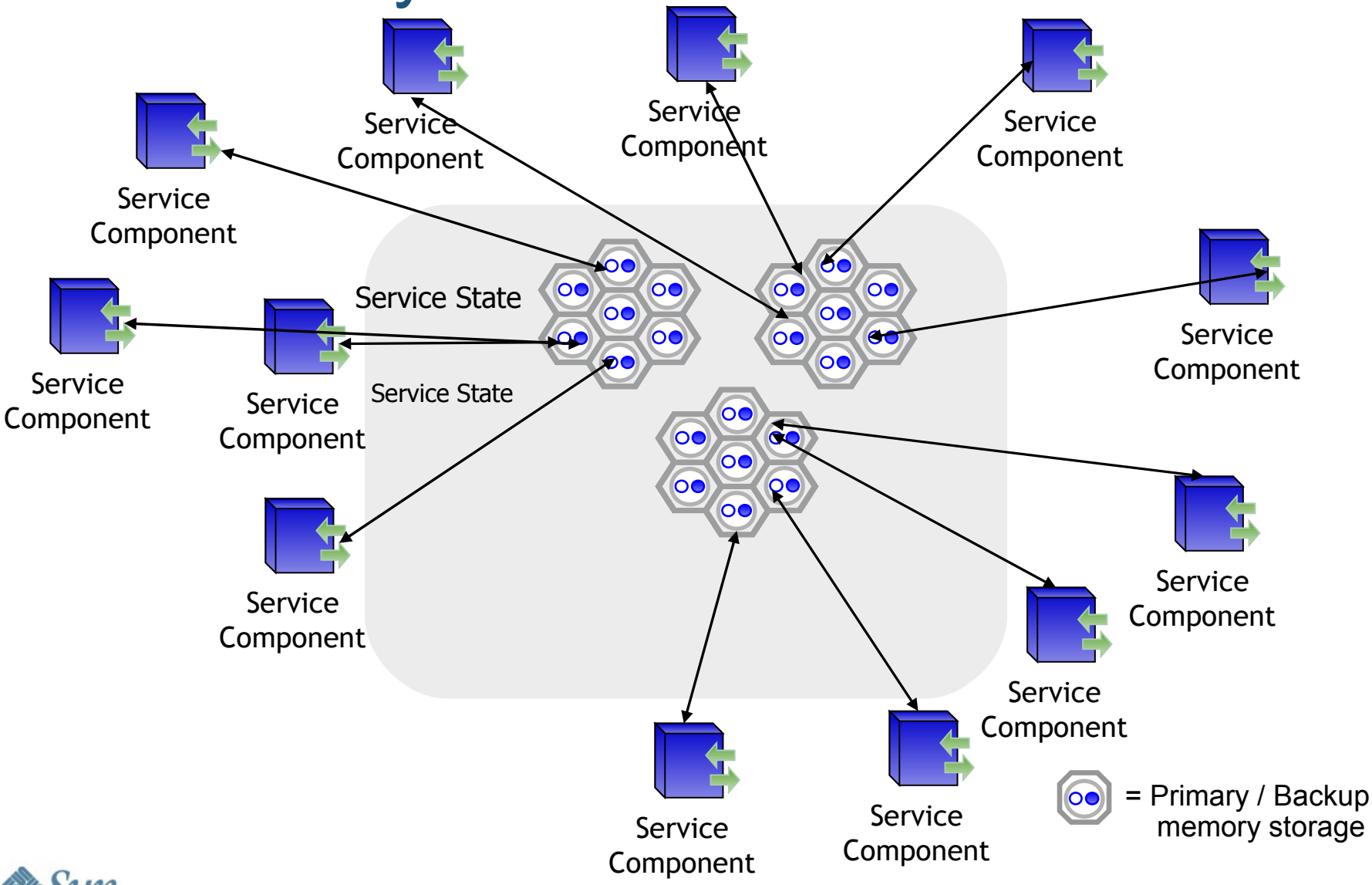
Service State Management – File Based



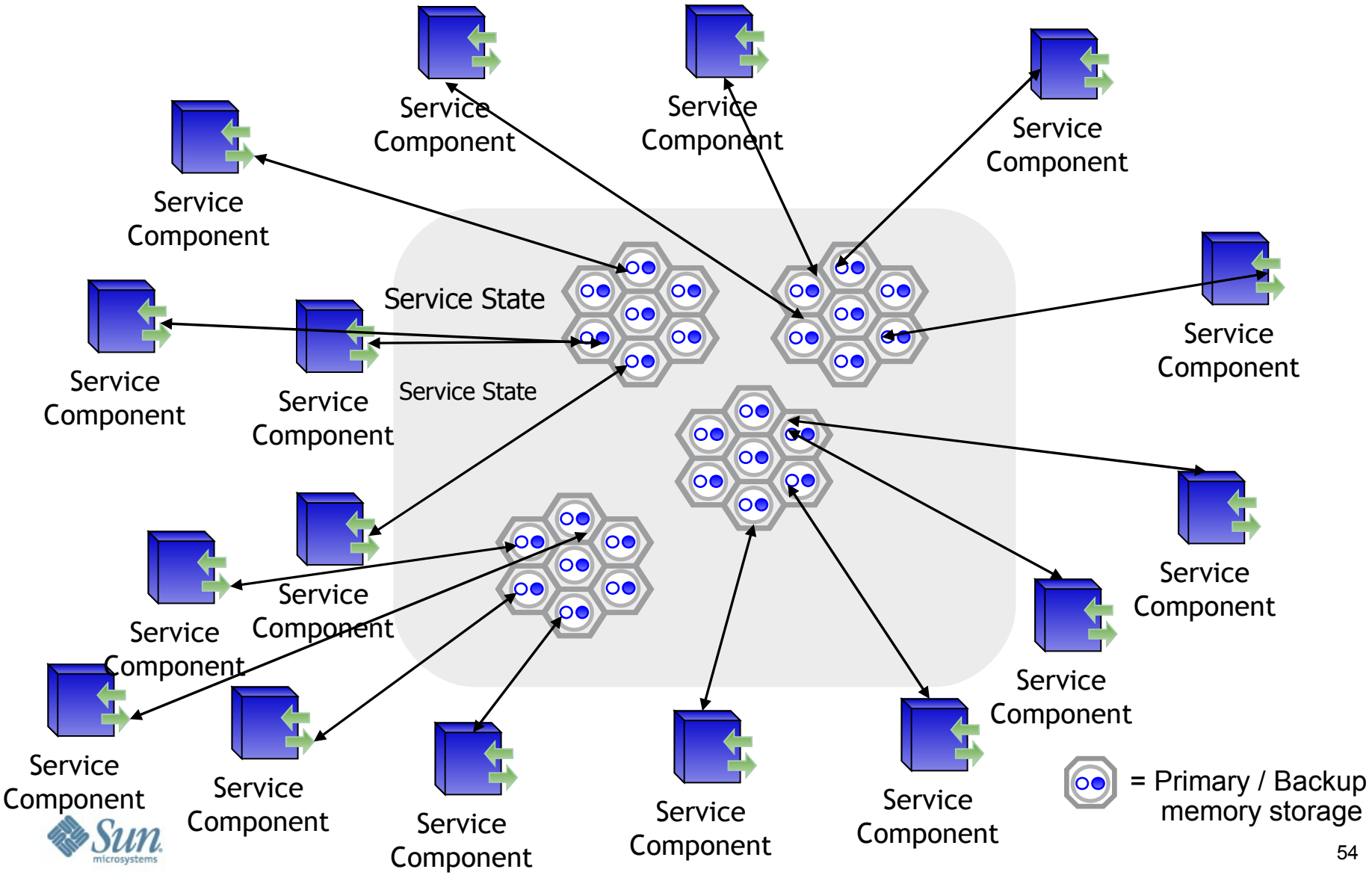
Service State Repository



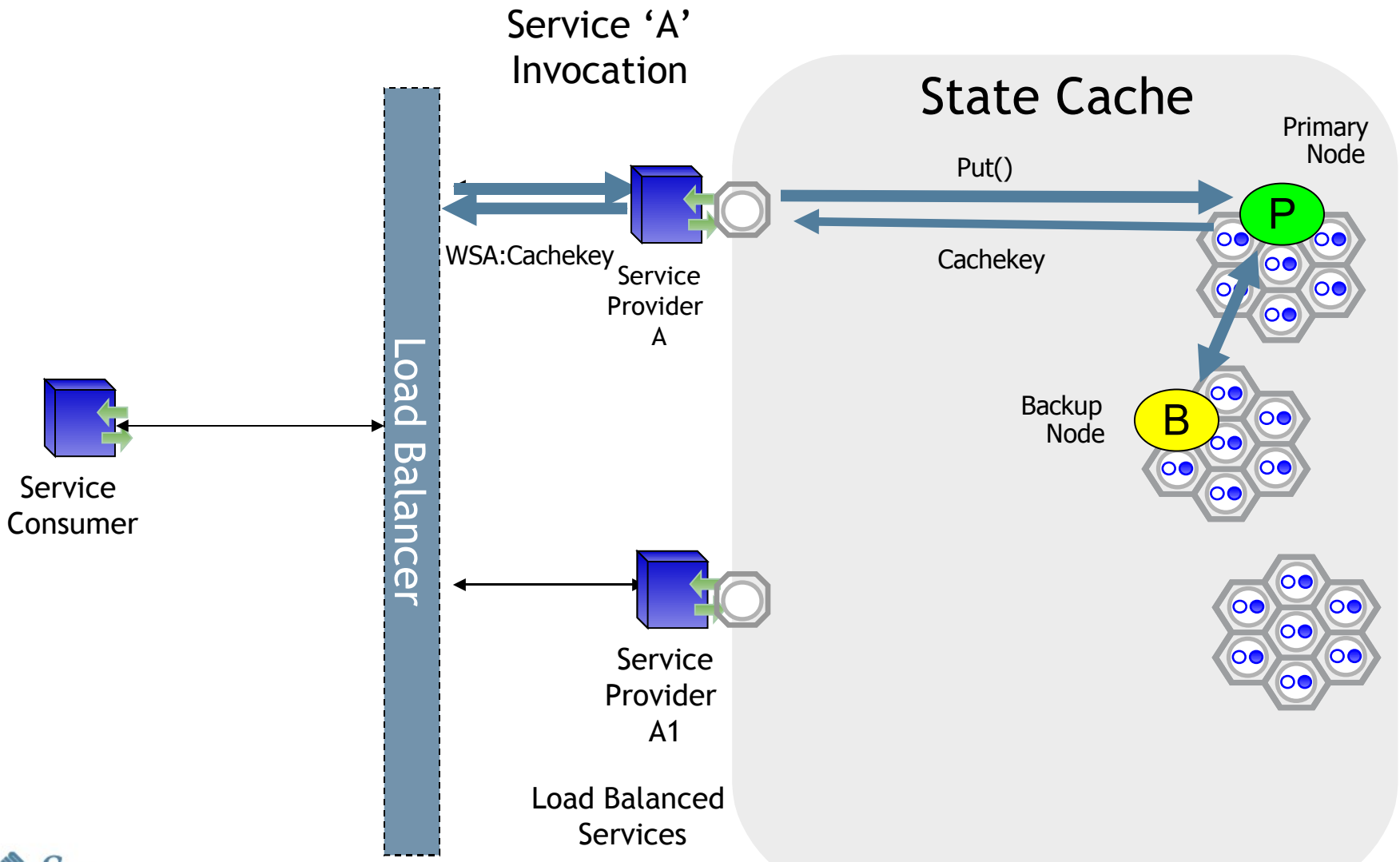
In-Memory FT Service State



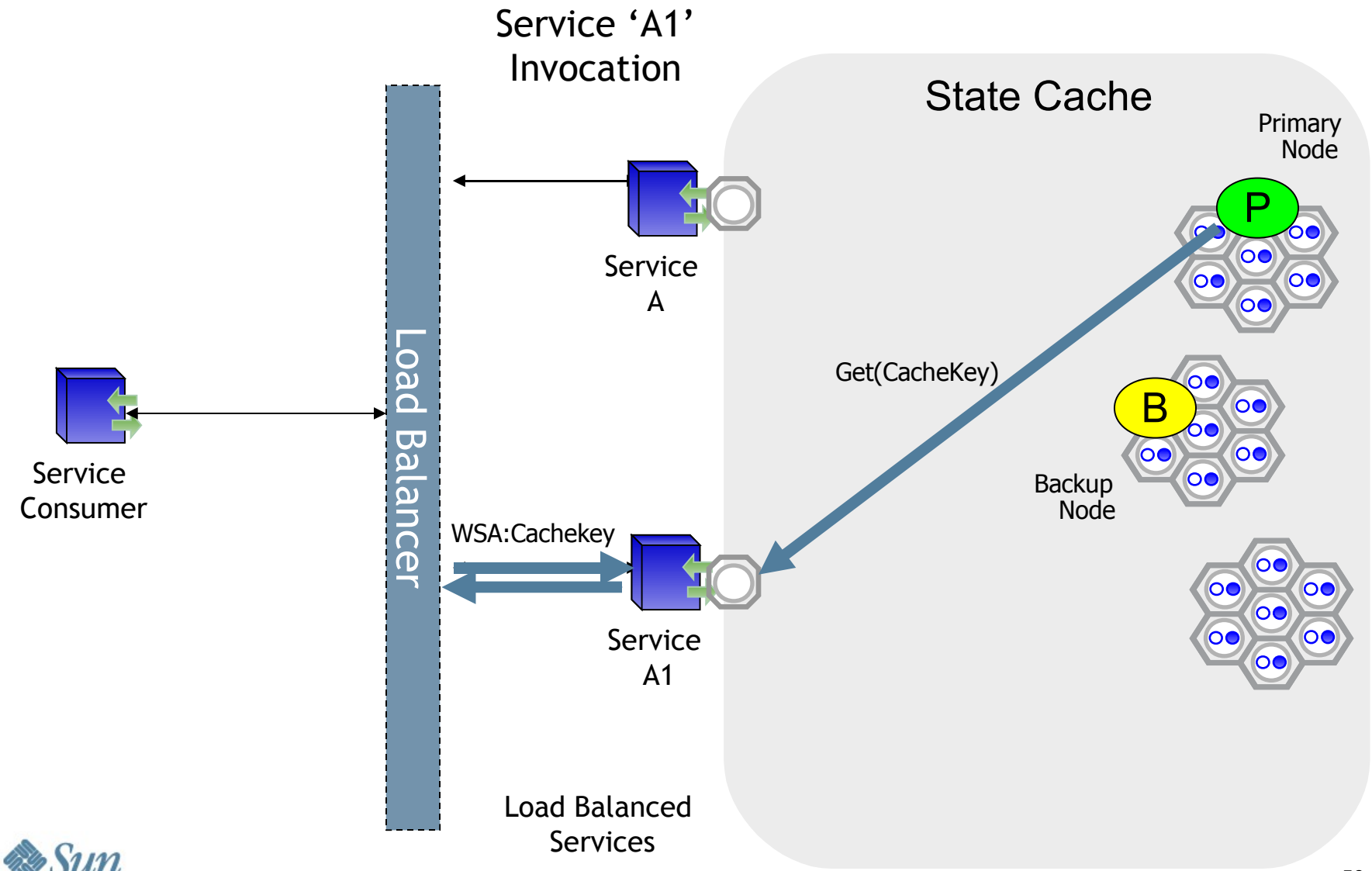
In-Memory FT Service State



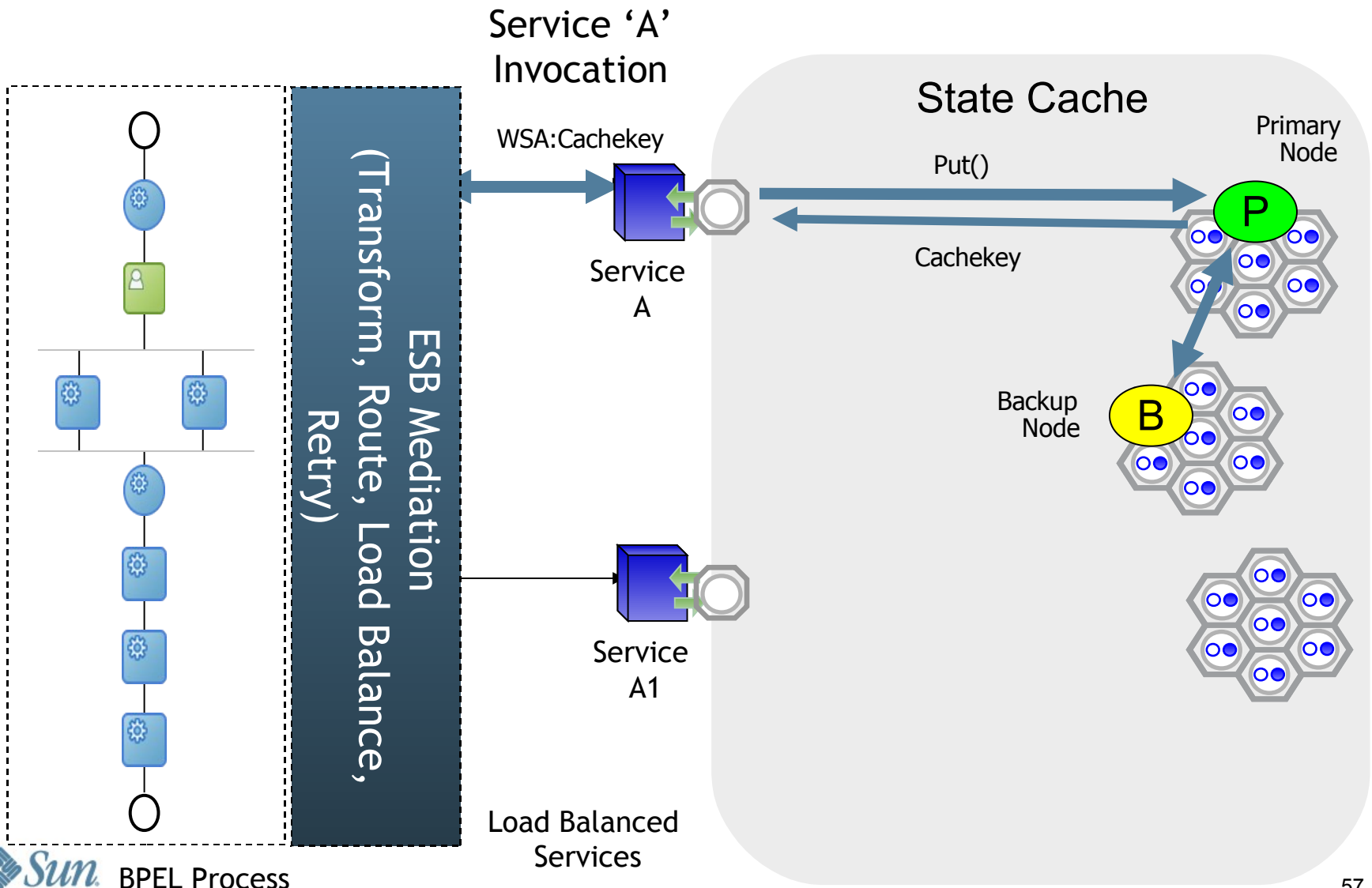
Stateful Service load balancing



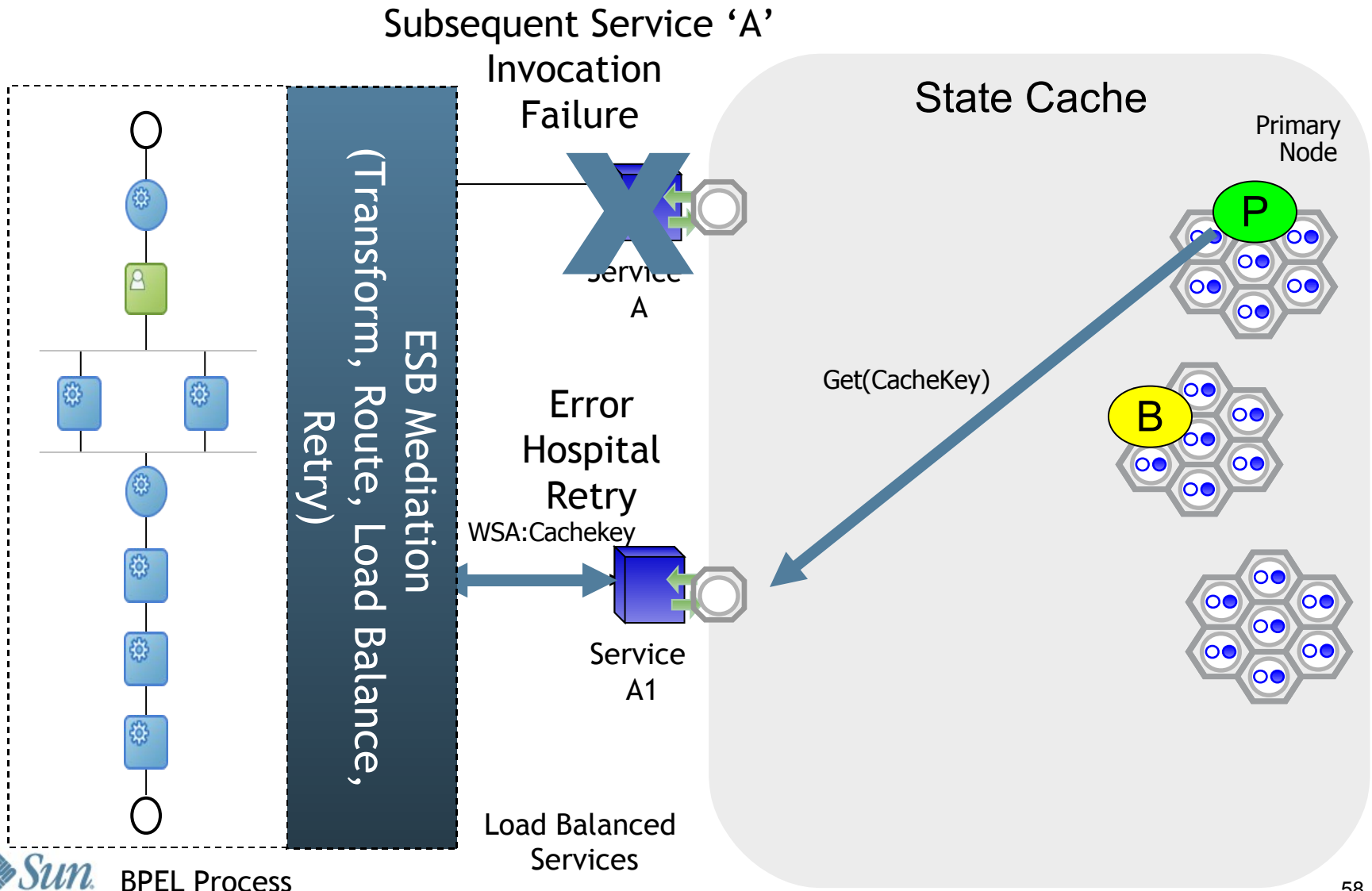
Stateful Service load balancing



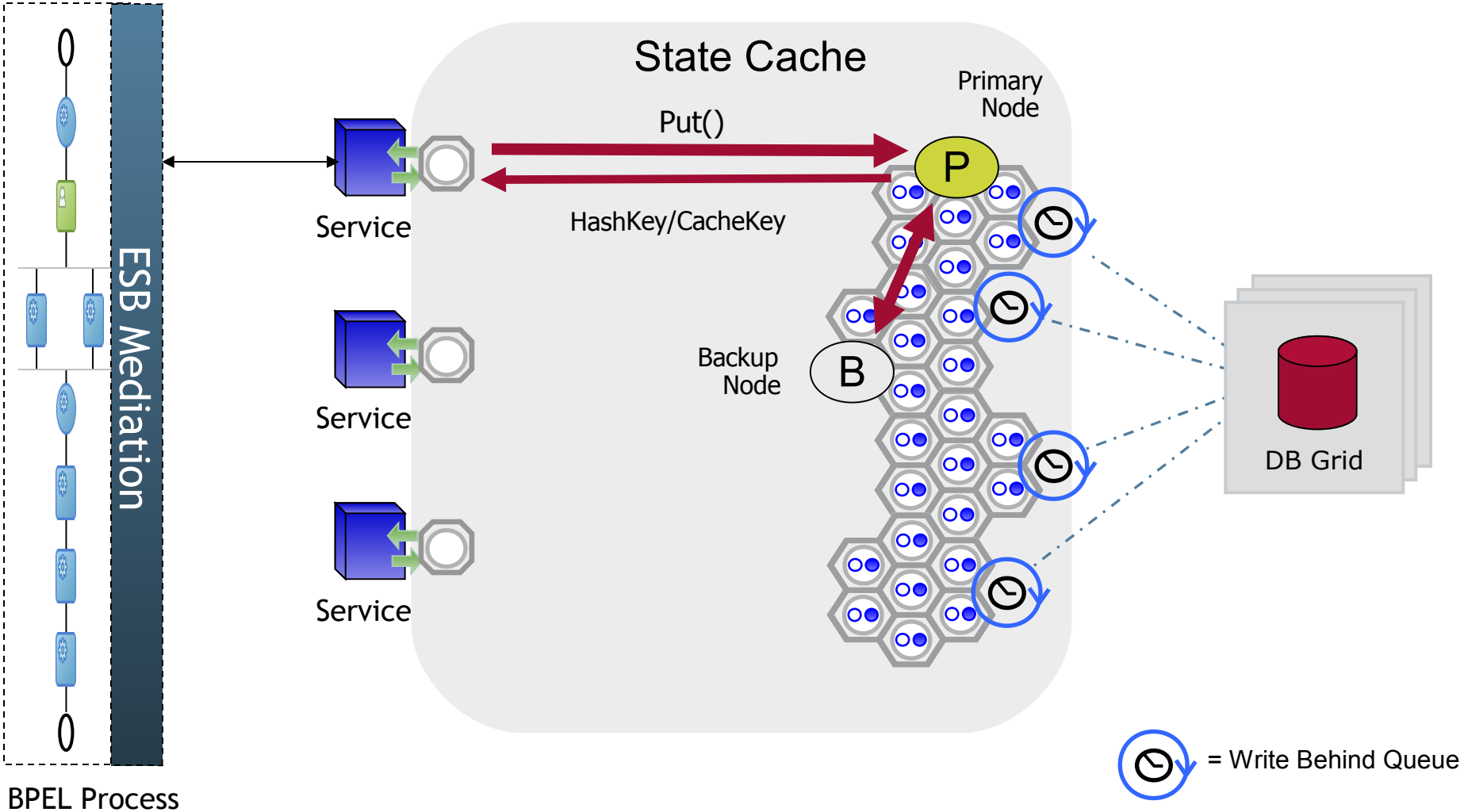
Stateful Service availability/failover



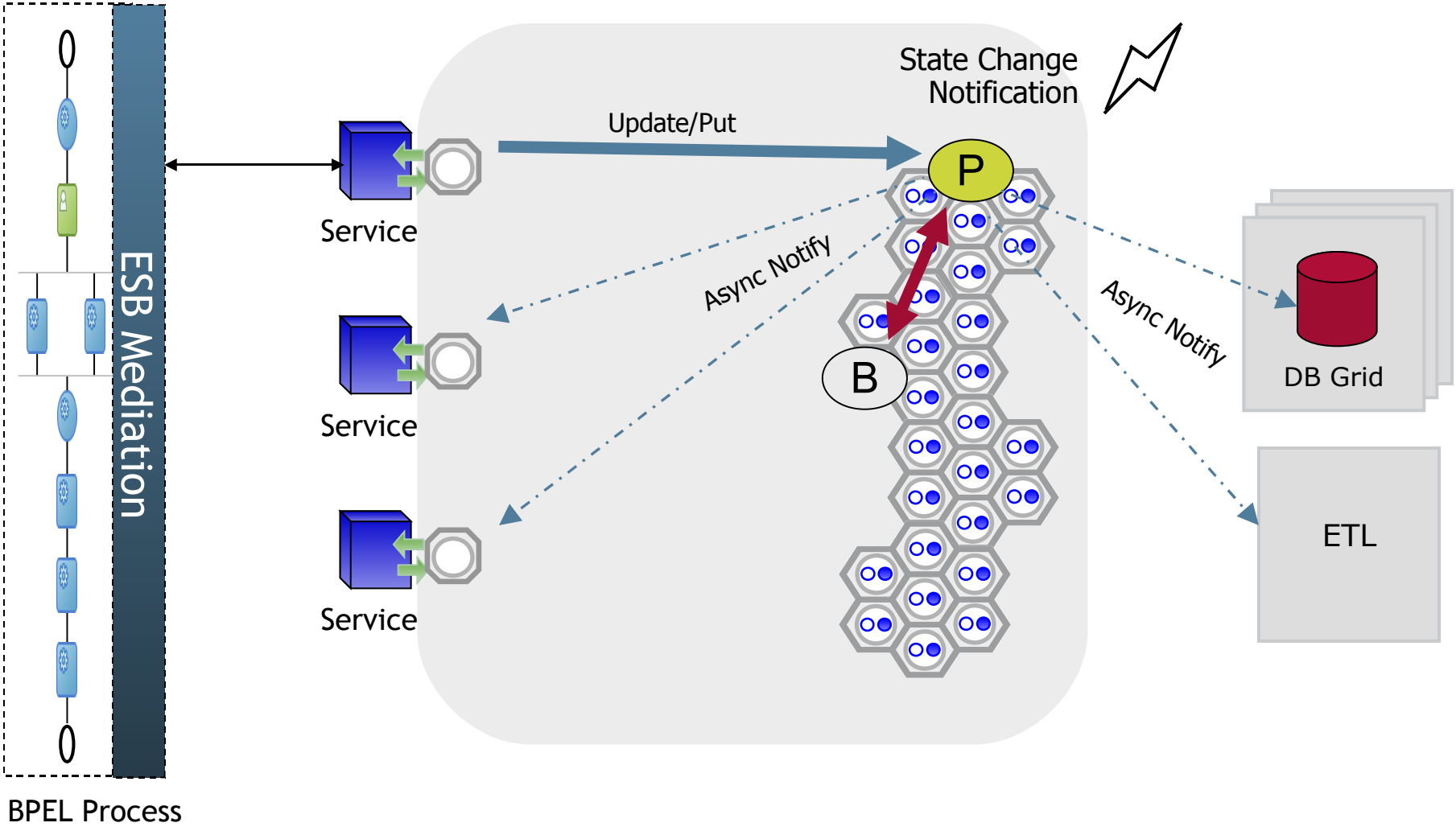
Stateful Service availability/failover



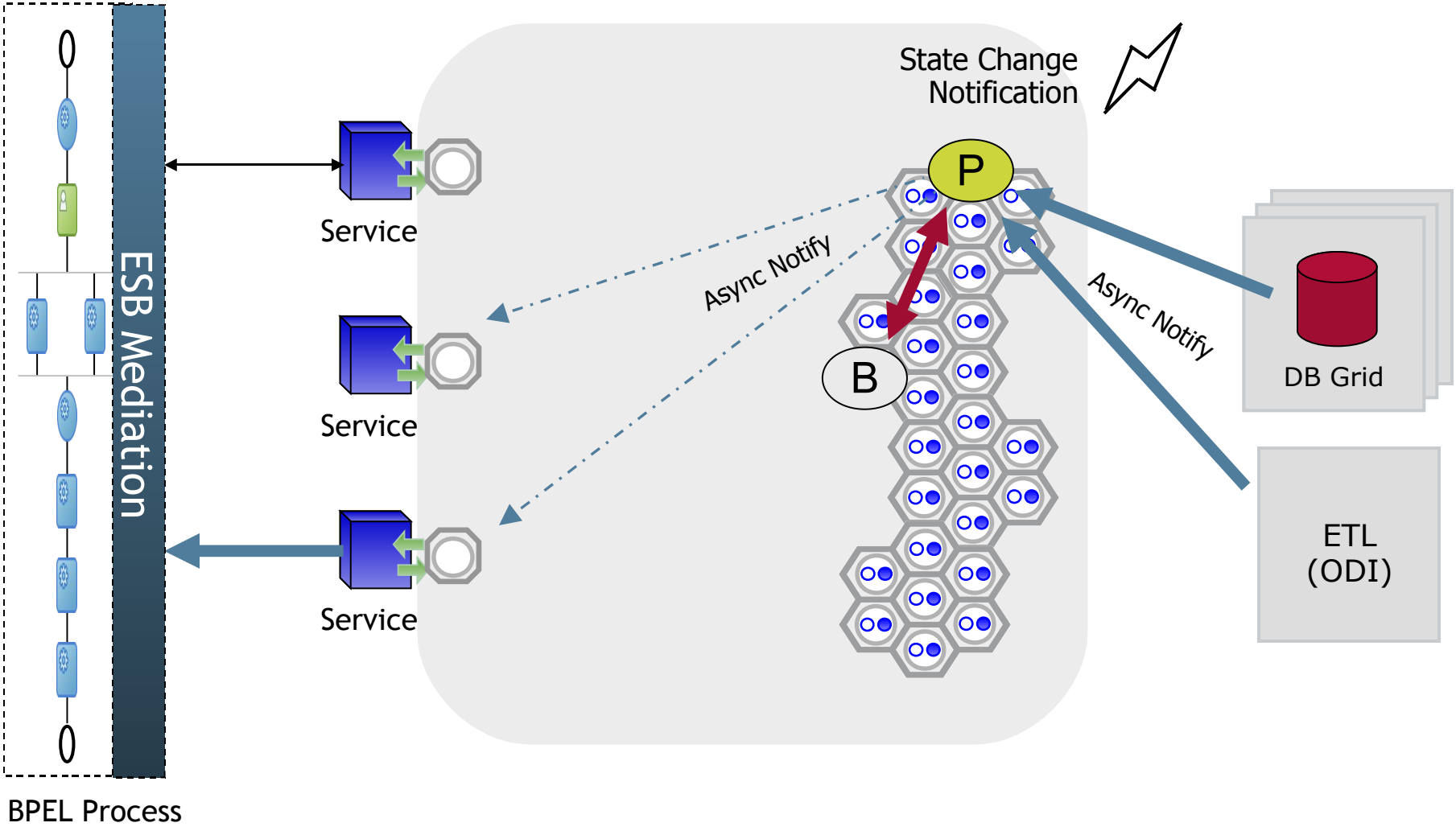
Asynchronous DB Updates



State-Based Notifications



State-Based Notifications



REST Patterns

- Give every “thing” an ID (URI):
  - http://example.com/orders/2007/11
- Link things together (Hypermedia links):
  - <order self='http://example.com/customers/1234' >
    <amount>23</amount>
    <product ref='http://example.com/products/4554' />
    <customer ref='http://example.com/customers/1234' />
    </order>
- Use standard methods (HTTP verbs)
  - GET is safe and idempotent
  - PUT and DELETE are not safe but are idempotent
  - POST is neither safe nor idempotent