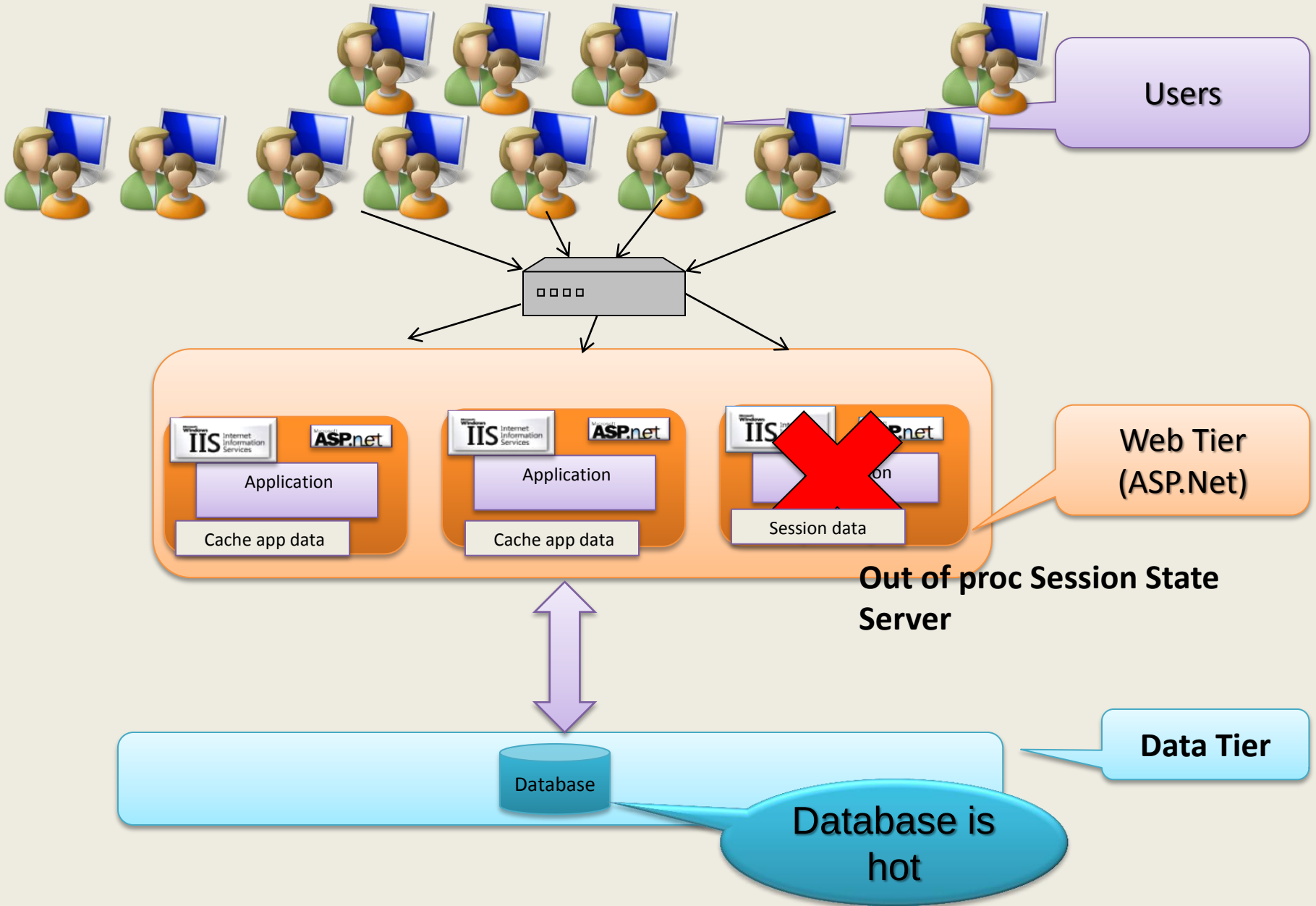
Distributed Caching Platforms

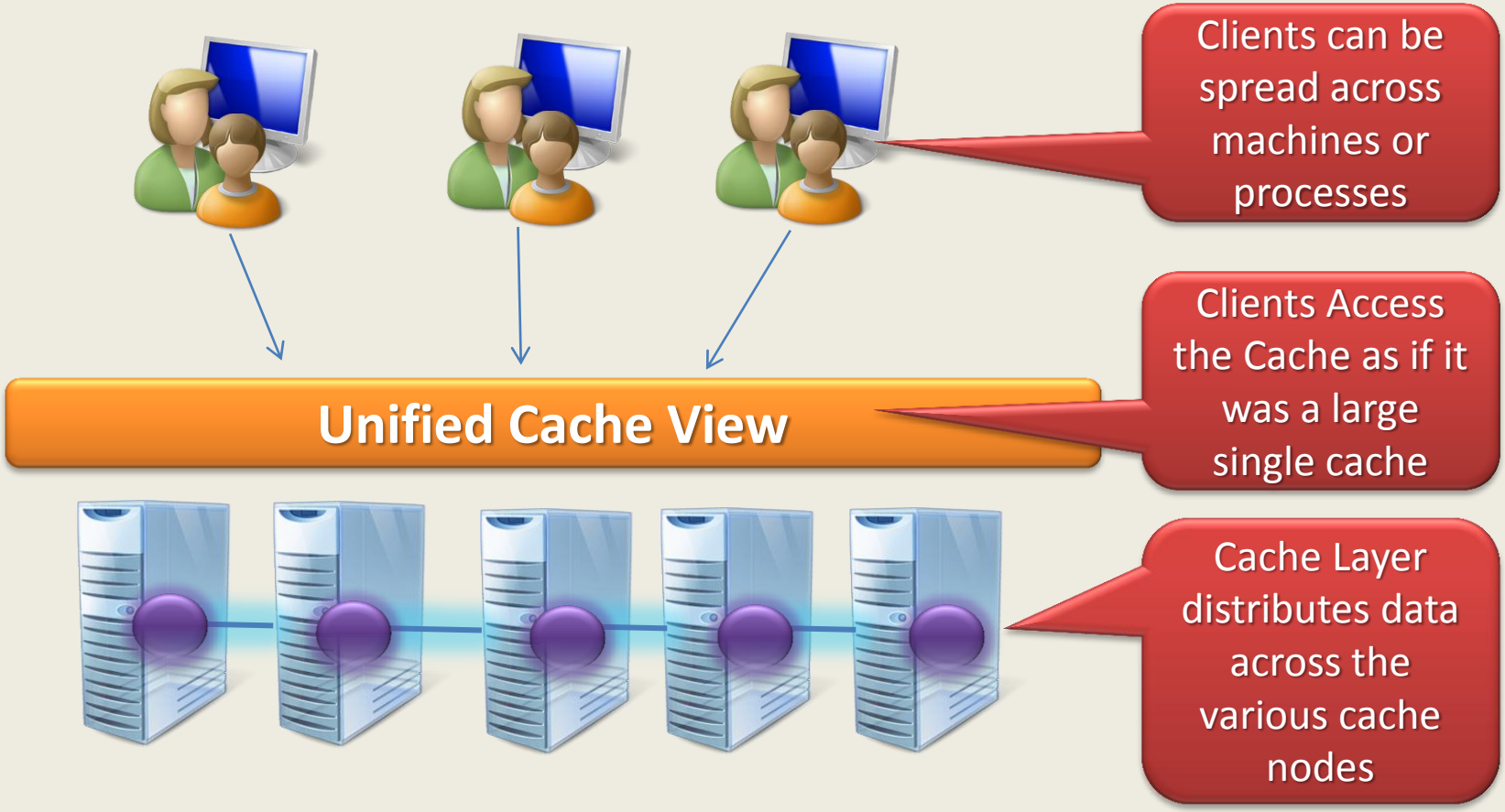
Anil Nori
anilnori@microsoft.com
September 14, 2010
VLDB 2010

Typical Web Applications

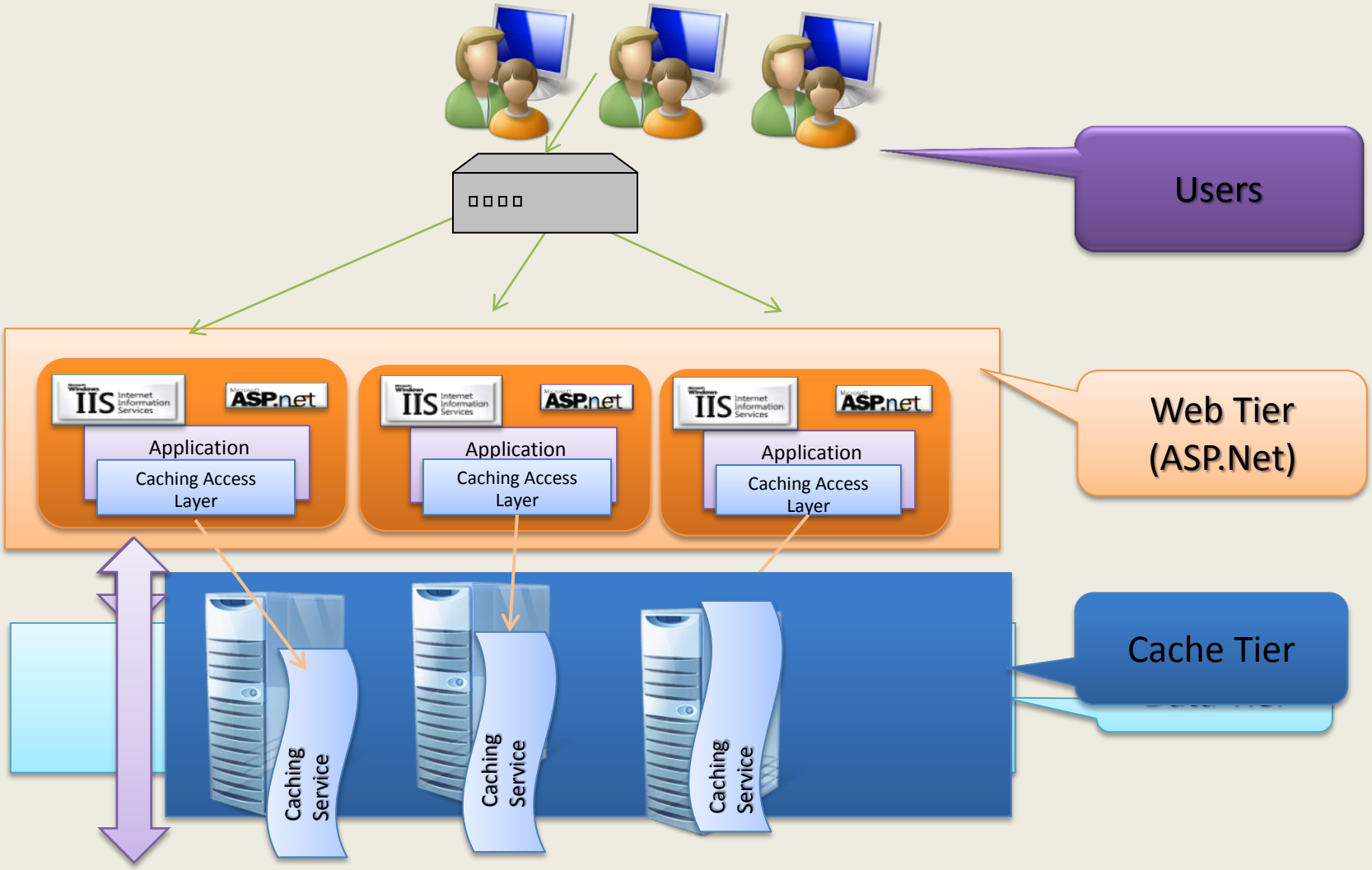


What is "Distributed Caching"?

- **An explicit, distributed, in-memory application cache for all kinds of data (Java/.Net objects, rows, XML, Binary data etc.)**
  - Fuse "memory" across machines into a unified cache



Where does it fit?

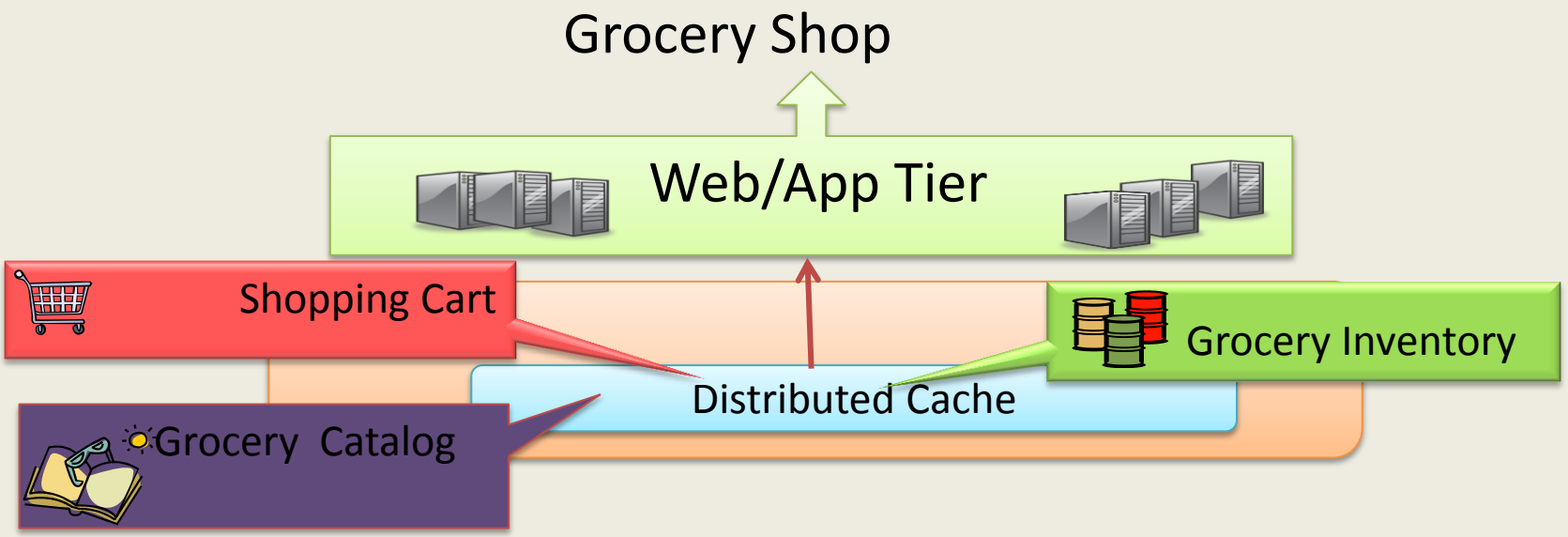


Distributed Cache Usage

	Scenario	
Horizontal	Web	• User-specific HTTP session and shared state across web farm • In-flight shopping carts for web retail • Enabling online self-service applications • Explicit storage of pre-computed or highly-accessed data
Horizontal	LOB	• Enterprise-wide product catalog for POS, analytics • Caching frequently used reference data for a ERP application
Verticals	Telco	• Cellular/VOIP: compute utilization, prepay charges, call routing and session info • SMS: message content / notification / receipt, billing
Verticals	Travel	• Aggregated flight pricing / availability retrieved from airlines
Verticals	Defense	• Sensor network data processing and threat detection
Verticals	Financial	• Per-user portfolio data and delayed quote storage for trading
Verticals	Financial	• Aggregate and process ticker stream for algorithmic trading

Types of Application Data

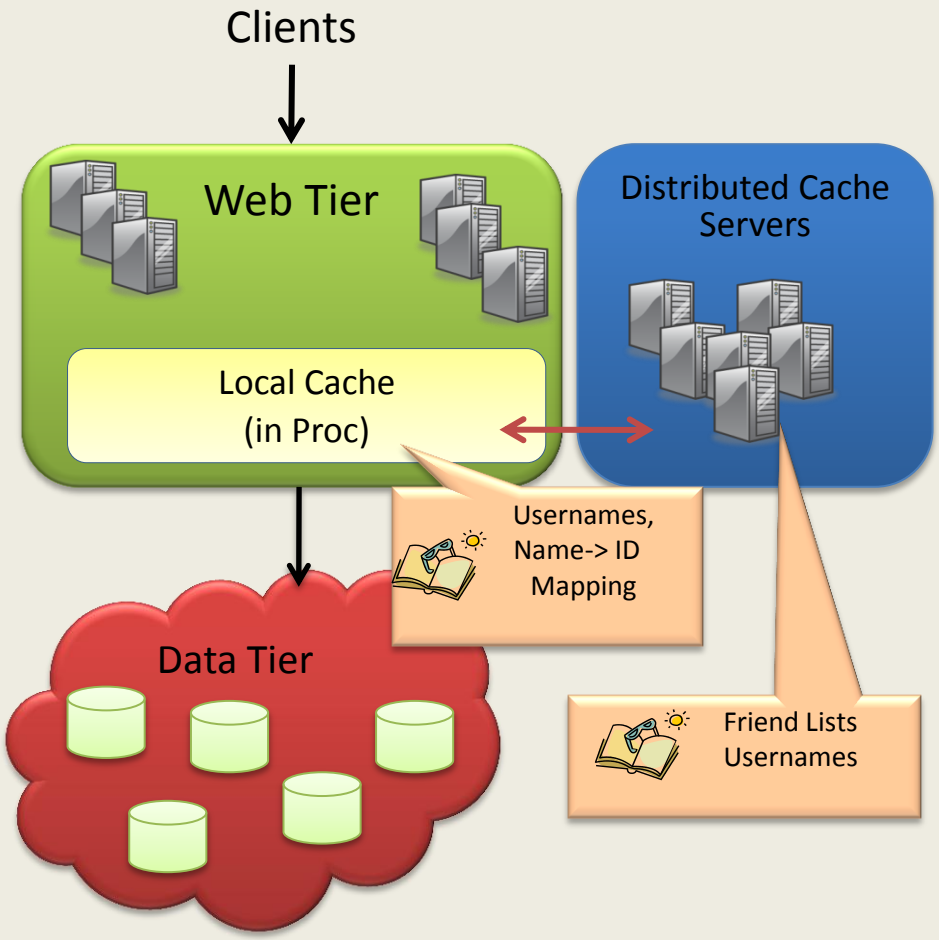
Reference	Activity	Resource
Primary Read Only	Read-Write Not shared	Read-Write, Shared
Catalog Data	Shopping Cart	Auction Data/Seat Assignment



Caching Reference Data

- A version of the authoritative data
  - Aggregated or transformed
- Each version is unique
- Refreshed periodically
- Examples
  - Web and Enterprise (Product) Catalogs
  - User, Employee data
- Access pattern
  - Mostly read
  - Shared & Concurrent Access
- Scale
  - Large number of accesses
- Functionality
  - Key based Access
  - Simple Query & Filtering
  - Loading

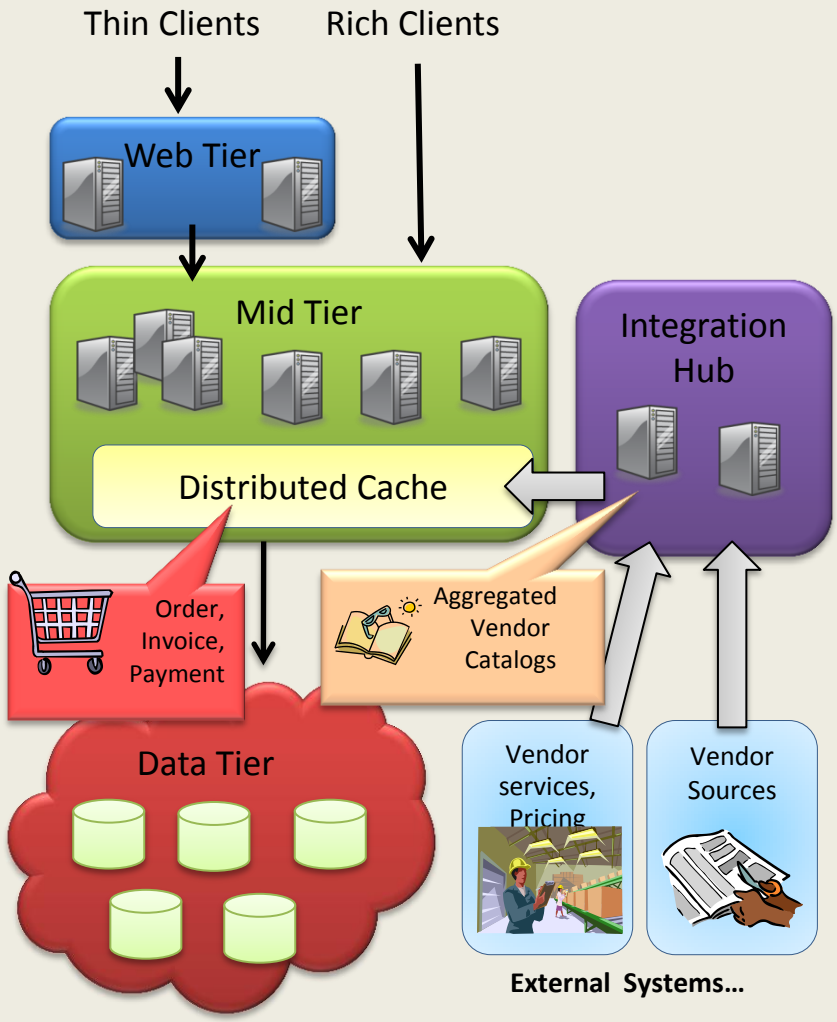
Scenario: Social Networking



Caching Activity-oriented Data

- Data typically generated as part of the application activity
- Active during business transactions
  - Typically logged to a backend data source
  - Historical data
- Examples
  - Shopping Cart
  - Session State
  - Enterprise LOB app (Purchase Order)
- Access pattern
  - Read and write
  - Primarily exclusive access
- Scale
  - High data (and access) scale
- Functionality
  - Key based access
  - Transactions (Grouping)

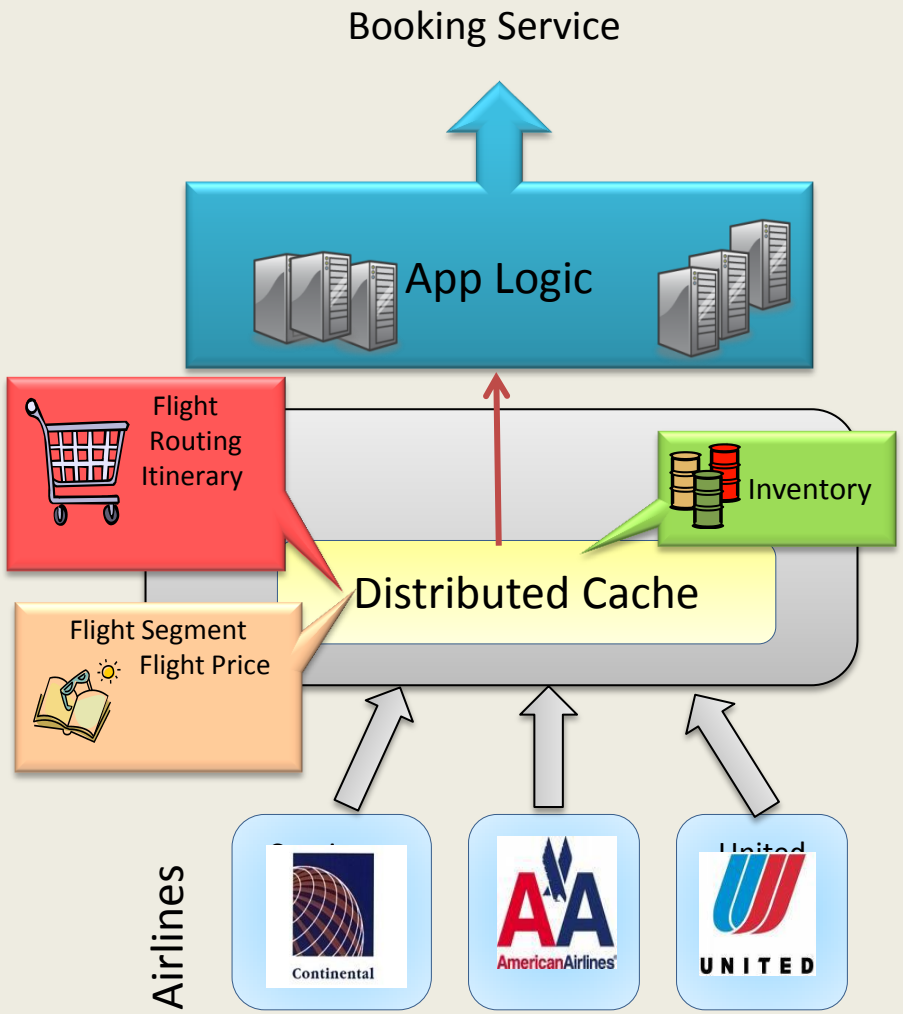
Scenario: Enterprise LOB Application



Caching Resource-oriented Data

- Authoritative data
- Modified by transactions; spans transactions
- Examples
  - Flight Inventory
- Access pattern:
  - Read and write
  - Shared access
- Functionality
  - Key based access
  - Transactions
- Scale
  - Large number of concurrent accesses
  - Relaxed consistency for scale

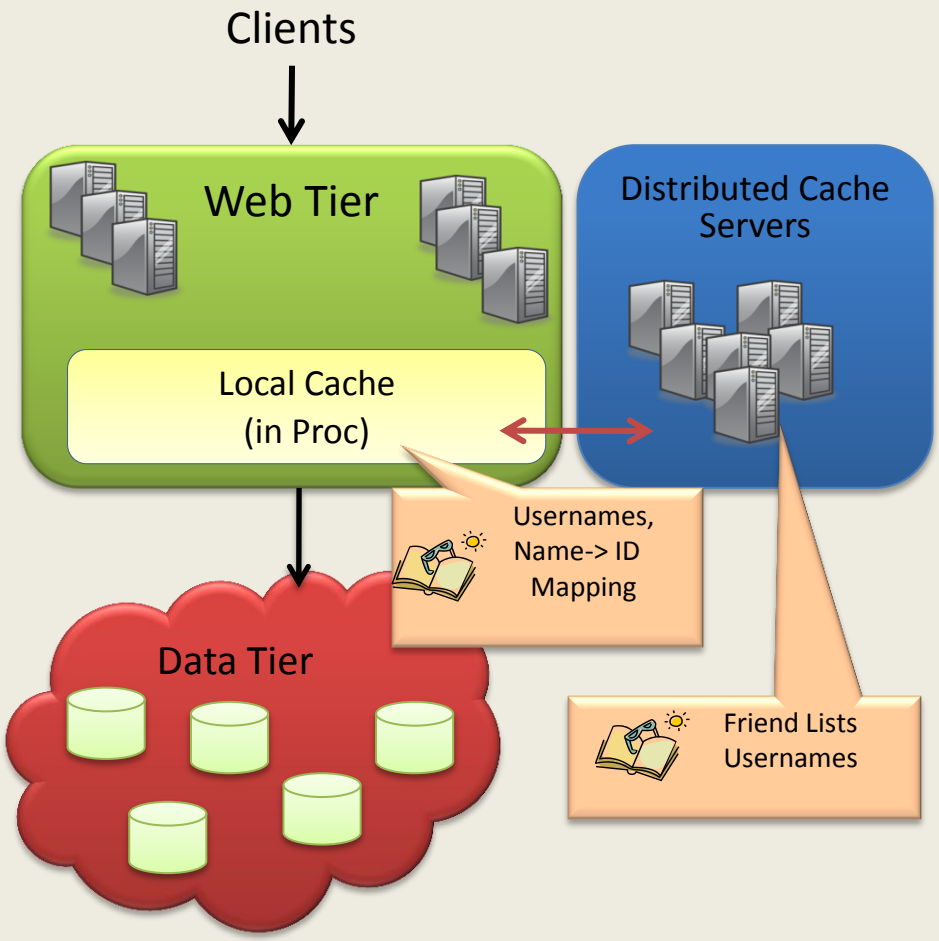
Scenario: Flight Inventory and Pricing



The Facebook Scenario

- A version of the authoritative data
  - Aggregated or transformed
- Several TBs on 100s of Memcached Servers
- Examples
  - User data, friend data, pictures
- Most accesses hit the cache
- Access pattern
  - Mostly read
  - Shared & Concurrent Access
- Scale
  - Large number of accesses
- Functionality
  - Key based Access
  - Simple query/Filtering

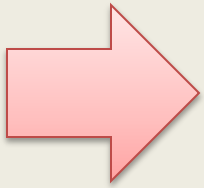
Scenario: Social Networking



Extreme Transaction Processing

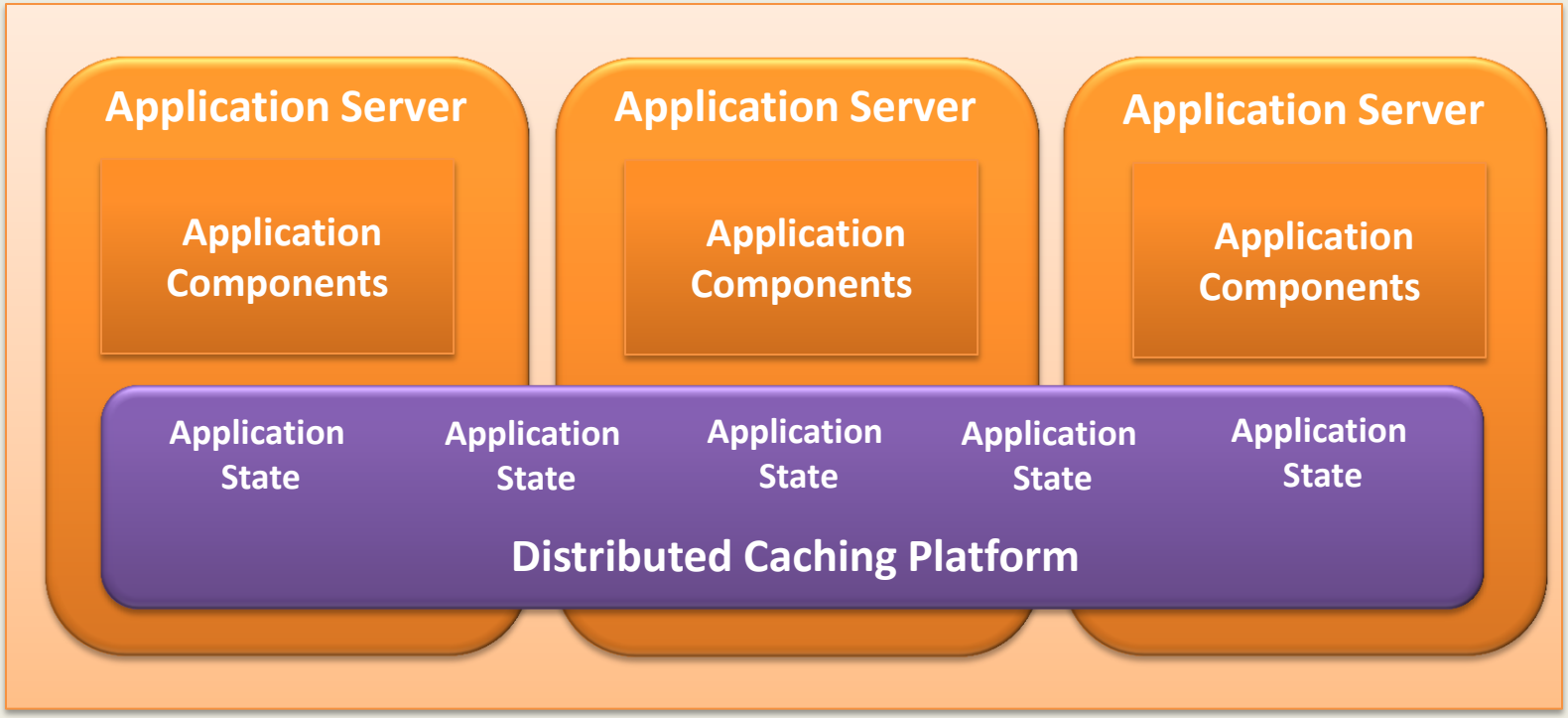
- Distributed TP applications with exceptionally demanding performance, scalability, availability
- Real-time, business critical, secure, and manageable

- Traditional TP monitors
- Enterprise Application Servers
- Traditional Integration Brokers
- Message Servers



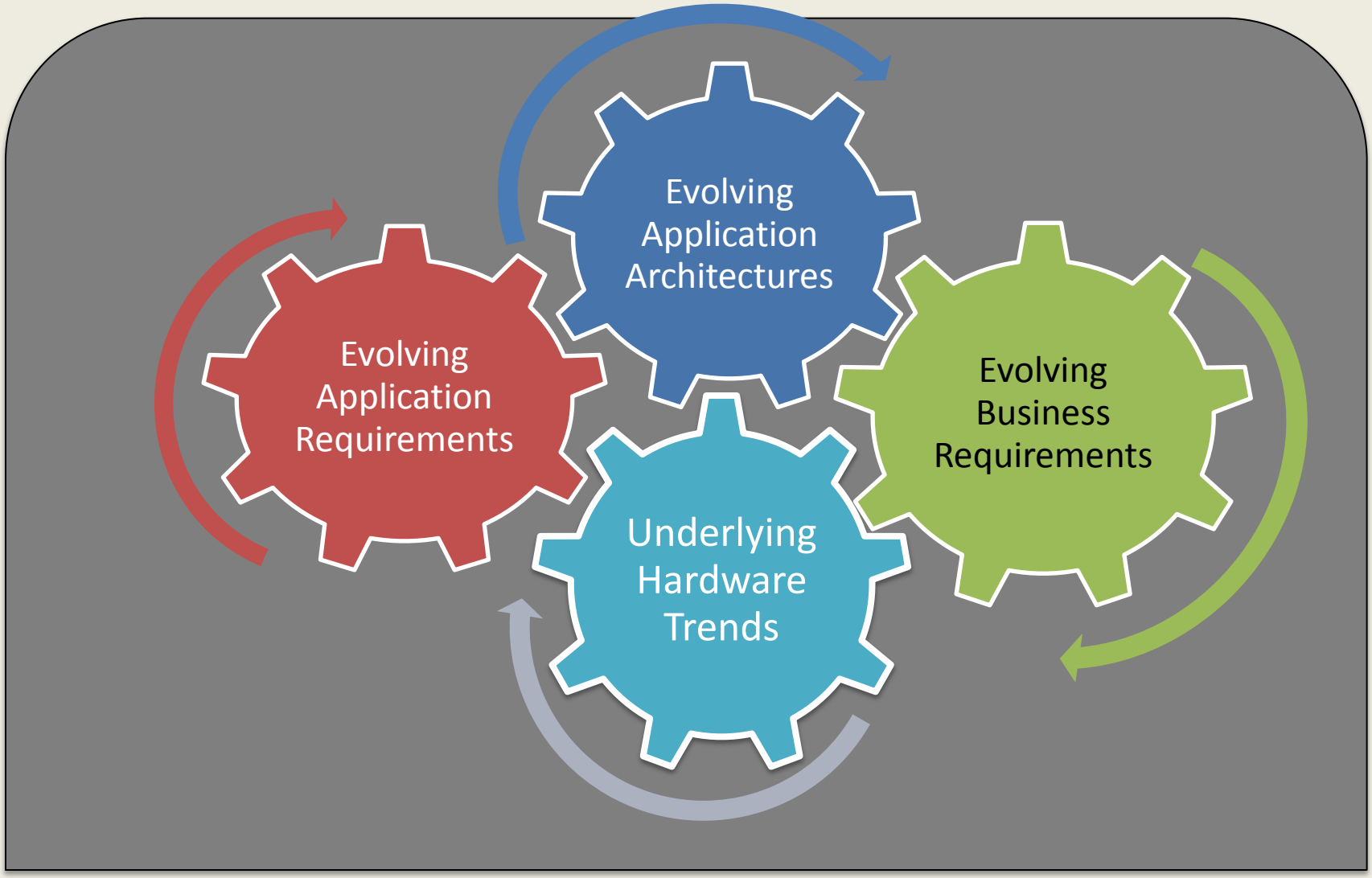
- Event Driven Messaging
- Enterprise/Internet Service Bus
- Grid/Fabric based Application Servers
- Low latency platform

Grid/Fabric based Application Servers



- Integrated distributed caching platform
- Application State Management
- Partitioned and Replicated application state
- Co-located logic and state
- Data aware routing
- Extreme low latency routing and access
- Durability and Persistence

Next generation applications – distributed, loosely-coupled, even-driven requiring high scale, performance and availability.



Application Requirements

- Efficient (Application) State management
- Performance
  - Millisecond/microsecond access
  - 100s of 1000s of accesses
- Scale
  - 10s – 100s of nodes in enterprise
  - 100s – 1000s in cloud applications
- Availability
  - Always available
- Consistency
  - Different degrees: Strong, Weak, Eventual, . . .
- Access
  - Key based and simple query based access
  - Transactions, Optimistic concurrency control
  - Invalidations

Caching API

```
// Create instance of cachefactory (reads appconfig)
DataCacheFactory fac = new DataCacheFactory();

// Get a named cache from the factory
DataCache catalog = fac.GetCache("catalogcache");
```

```
// Simple Get/Put
catalog.Put("toy-101", new Toy("Puzzle", .,.));

// From the same or a different client
Toy toyObj = (Toy)catalog.Get("toy-101");
```

```
// Region based Get/Put
catalog.CreateRegion("toyRegion");

// Both toy and toyparts are put in the same region
catalog.Put("toy-101", new Toy( .,.), “toyRegion”);
Catalog.Put("toypart-100", new ToyParts(…), “toyRegion”);

Toy toyObj = (Toy)catalog.Get("toy-101“,"toyRegion");
```

Access APIs – Tagging Items

- Add Tags to Items
  - Tag Search on Default Regions

```
Tag hotItem  = new Tag("hotItem");

catalog.Put("toy-101", new Toy("Puzzle"),
             new Tag[]{hotItem}, “toyRegion”);

catalog.Put("toy-102", new Toy("Bridge"), “toyRegion”);

// From the same or a different client
List<KeyValuePair<string, object>> toys =
          catalog.GetAnyMatchingTag("toyRegion", hotItem);
```

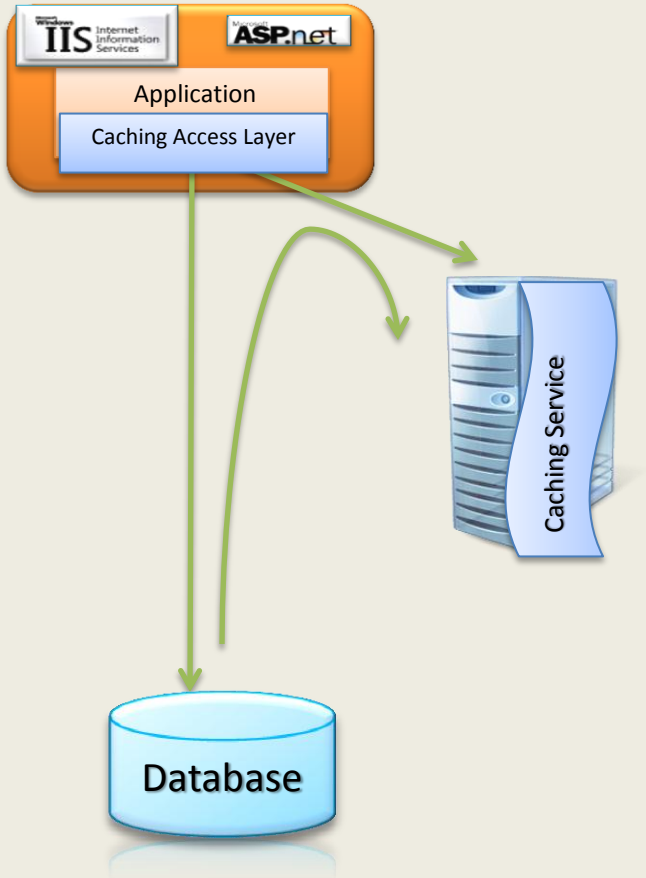
Usage Pattern – Cache Aside (Explicit Caching)

```
//  Read from Cache
Toy toyObj = (Toy)
    catalog.Get("toy-101");
```

```
// If Not present in the cache
if (toyObj == null)
{
   // Read from backend..
   toyObj = ReadFromDatabase();

   // Populate Cache
   catalog.Put("toy-101", toyObj);

    return toyObj;
}
```



Features

API
CRUD Operations (Create, Read, Update and Delete)
Any Object type
Multiple Client Languages
Concurrency APIs
Async and Batch APIs
Transactions
Query & Continuous Query
Cache Notifications
Eviction
Persistence
Session State (.NET, Java)
IDE support

Other
Administration & Monitoring
Security
Co-location of logic & data in cache

Supported Topologies
Partitioned
Replicated
Failover Support (High Availability)
Multiple Backups
Local Caching
Explicit Data Affinity
Embedded Cache
Geo-replicated

Extensibility
Custom Eviction
Custom Persistence
Custom Query
Triggers

Persistence
Read Through
Refresh Ahead
Write Through
Write Behind

IMDB vs. Distributed Caching Platforms (DCPs)

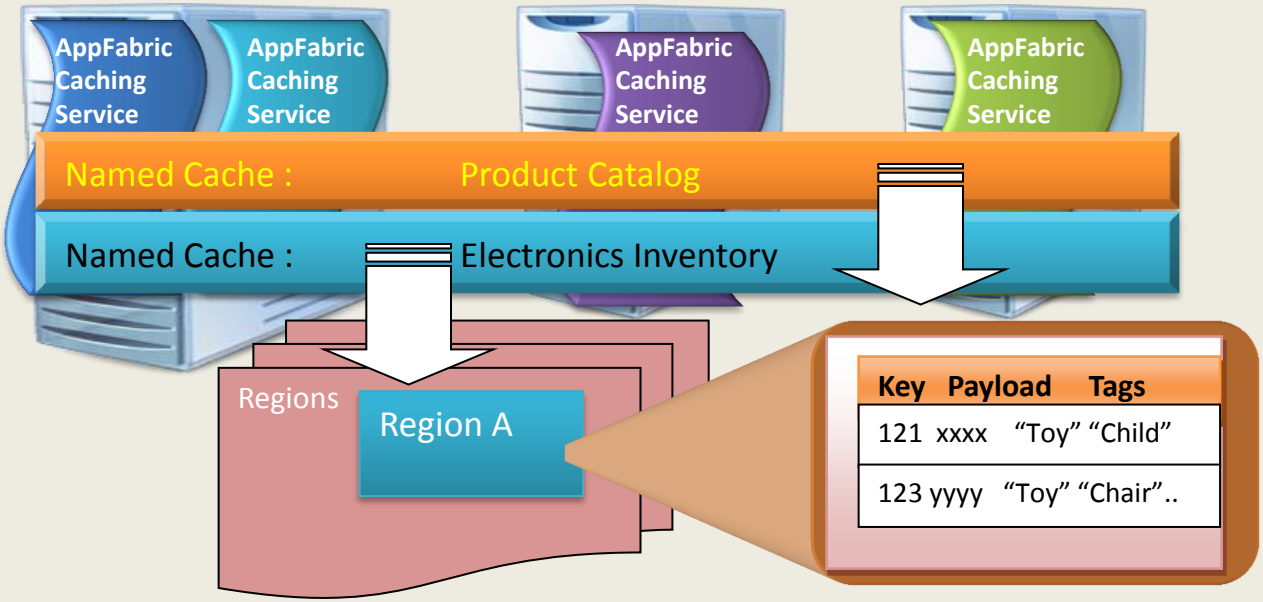
IMDB	DCP
Primarily relational store	Object store – any form of object
DB-specific representation	Application-specific representation
Only SQL query	Object/relational query (e.g. Linq, SQL)
Set-oriented access	Key based, Navigational, set-oriented access (e.g. GET, PUT, simple query)
Centralized	Distributed
Performance acceleration	Performance, Scale, and Failover
Server deployments	Embedded or server deployments
Niche, vertical markets (e.g. Telco)	General purpose (e.g. Web, LOB)
e.g. TimesTen, Solid DB, ANTS	e.g. memcacheD, Gemstone, Oracle Coherence, Gigaspaces, IBM extremeScale, AppFabric Caching etc..

DCP Players

- Memcached (open source)
- VMWare (Gemstone) Gemfire
- Gigaspaces Extreme Application Platform
- IBM WebSphere Extreme Scale Cache
- Microsoft AppFabric Caching
- Oracle Coherence
- Terracotta's Terracotta Server (open source)

Distributed Caching Platform Concepts

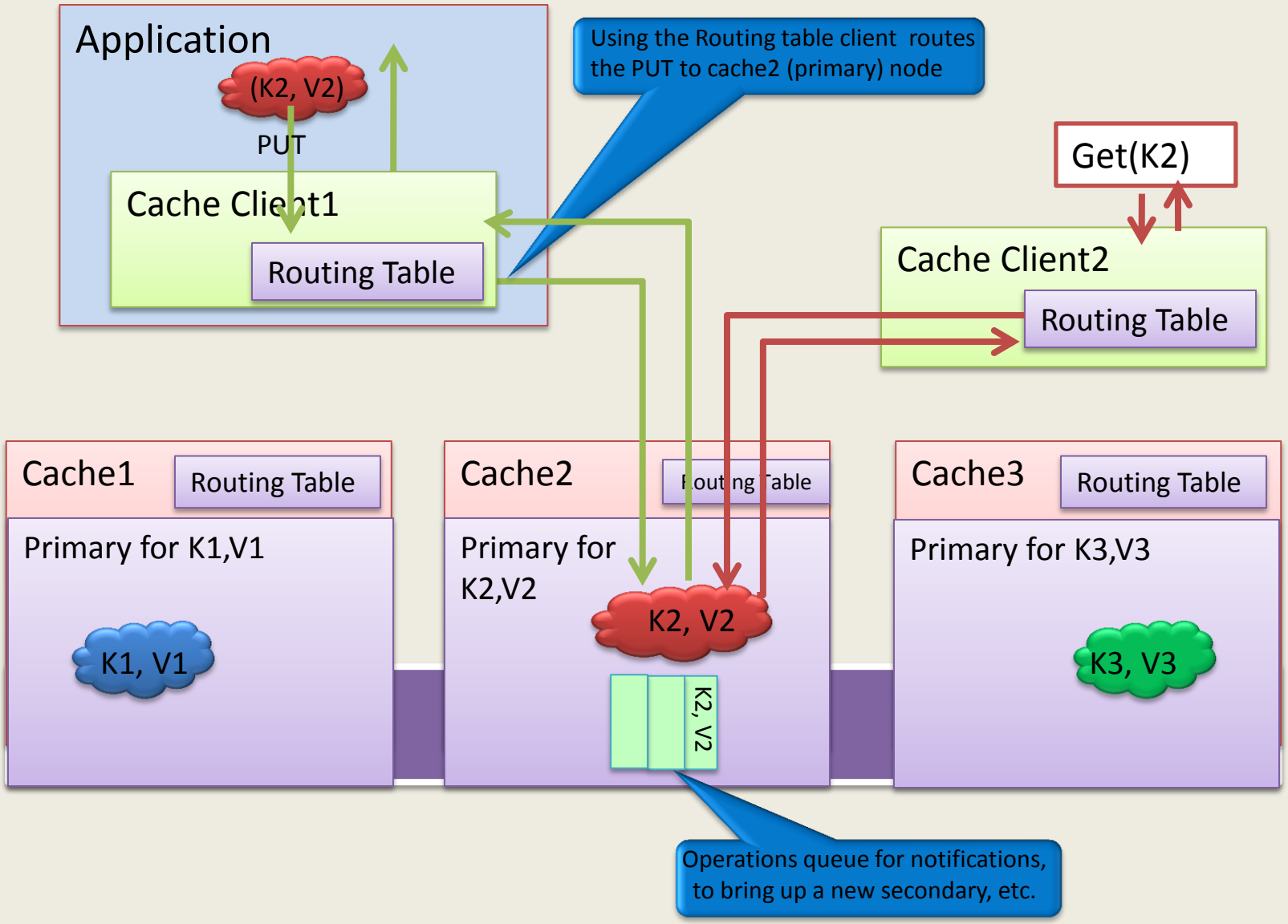
AppFabric Caching Logical Hierarchy



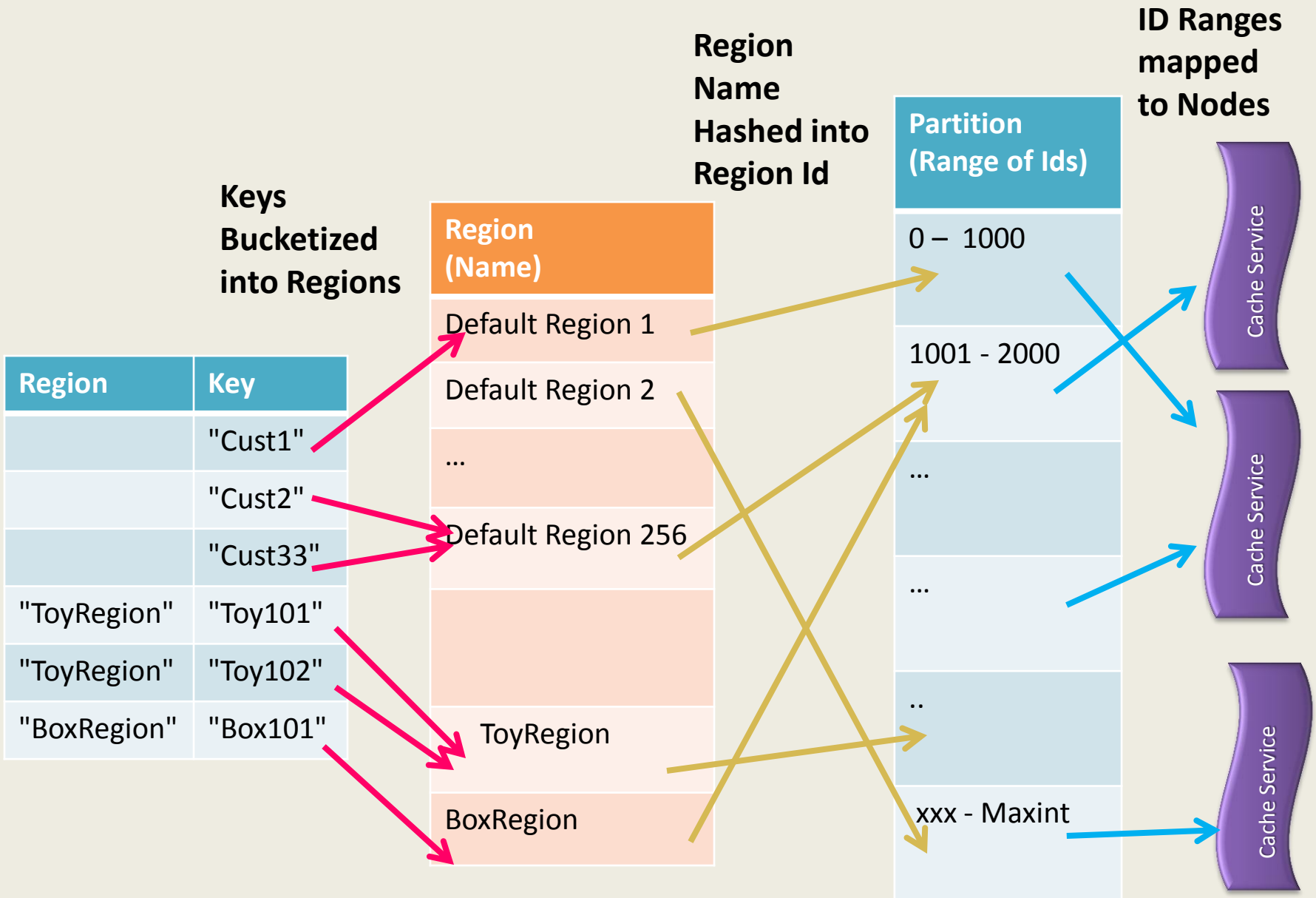
Machine -> Cache Host -> Named Caches -> Regions -> Cache Items -> Objects

- Host
  - Physical processes hosting AppFabric Caching instance.
- Named Caches
  - Can span across machines
  - Defined in the configuration file
- Cache Item
  - Key, Payload (Object), Tags, TTL, Timestamps, Version
- Regions
  - Physically co-located Container of Cache Items
  - May be implicit or explicitly created

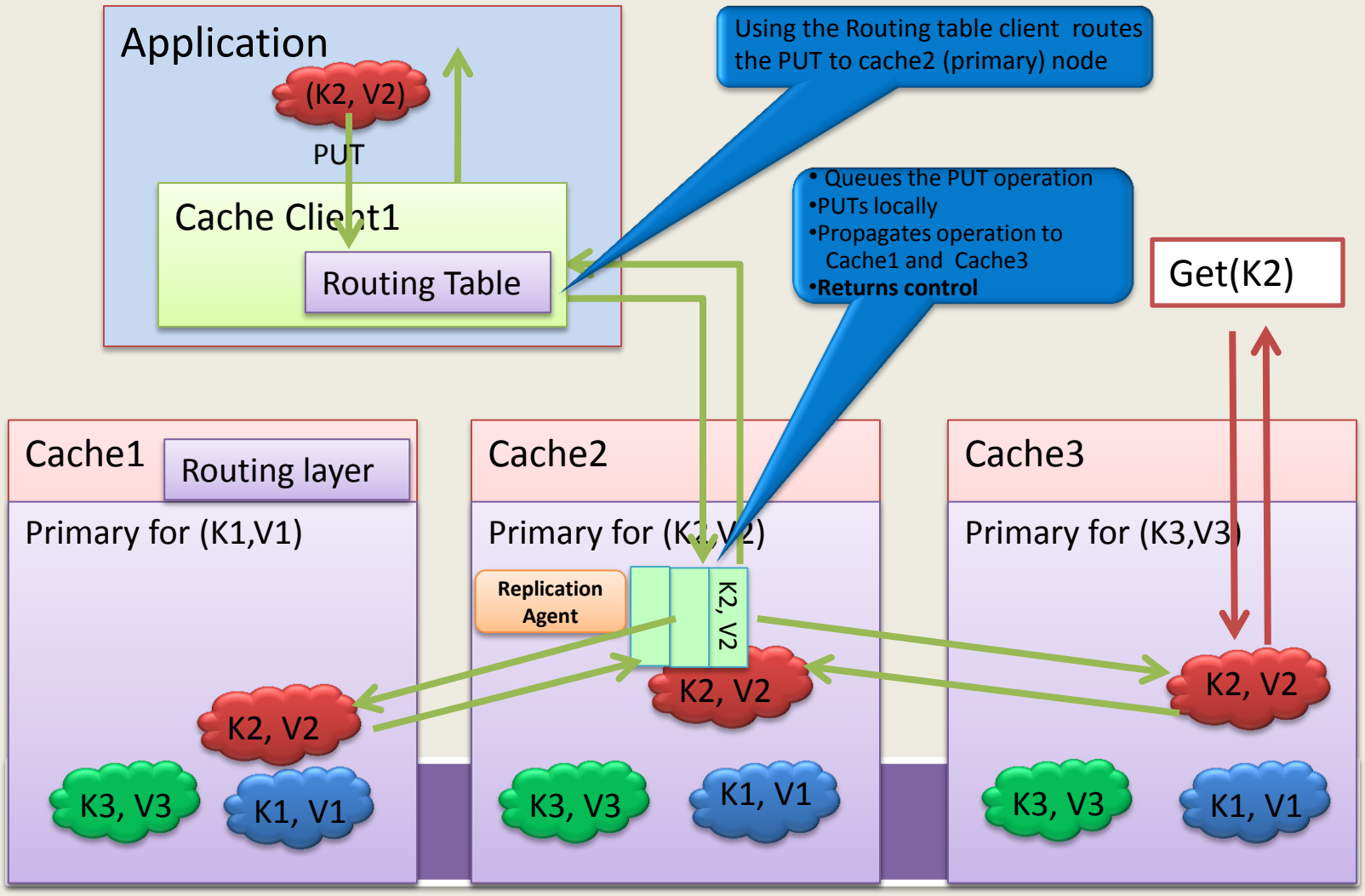
Scale: Partitioned Cache



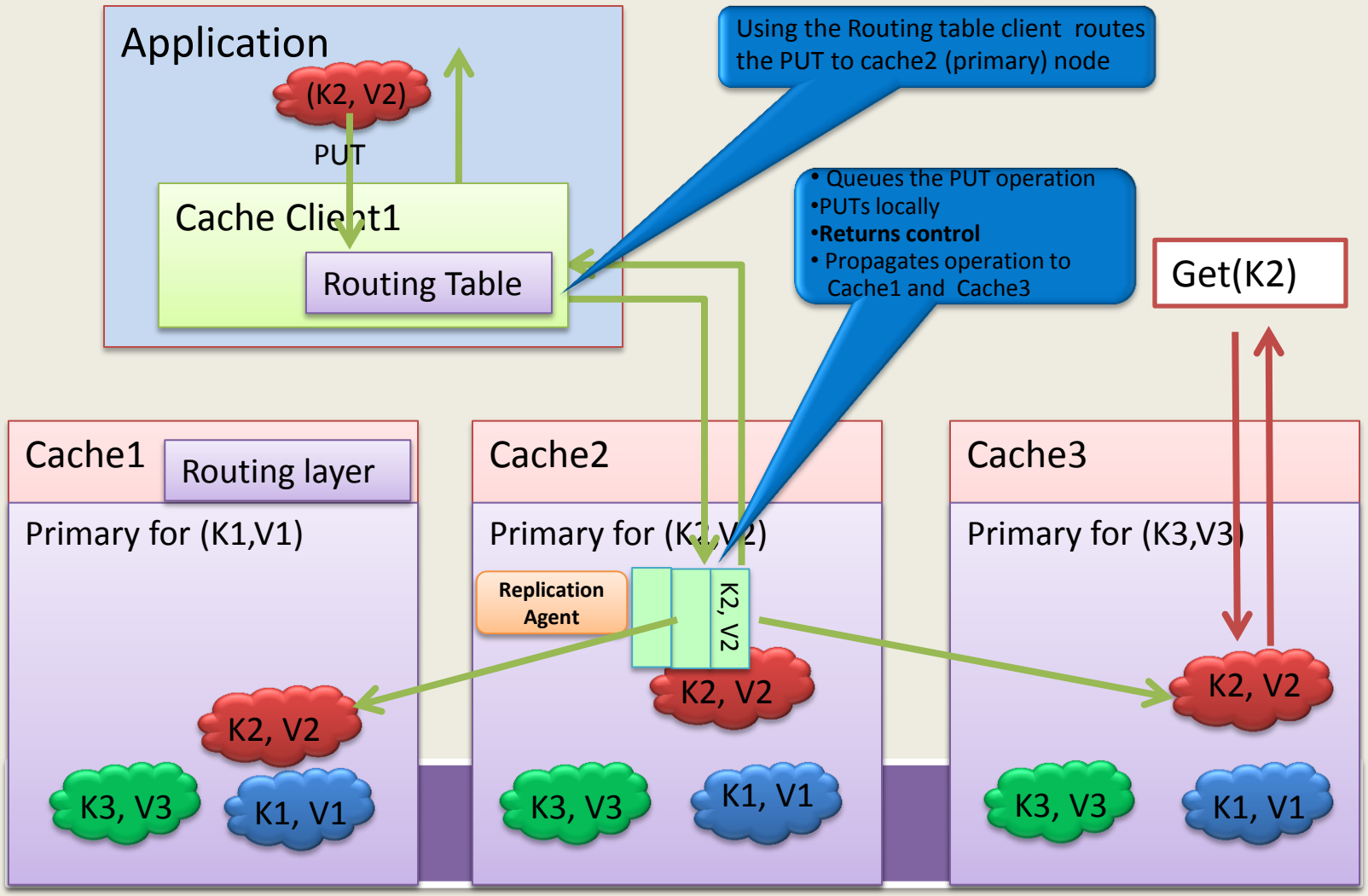
Key Mapping


Keys Bucketized into Regions
Region Name Hashed into Region Id
ID Ranges mapped to Nodes
Region
Key
"Cust1"
"Cust2"
"Cust33"
"ToyRegion"
"Toy101"
"ToyRegion"
"Toy102"
"BoxRegion"
"Box101"
Region (Name)
Default Region 1
Default Region 2
...
Default Region 256
ToyRegion
BoxRegion
Partition (Range of Ids)
0 – 1000
1001 - 2000
...
...
..
xxx - Maxint
Cache Service
Cache Service
Cache Service

Scale: Replicated Cache (Synchronous)

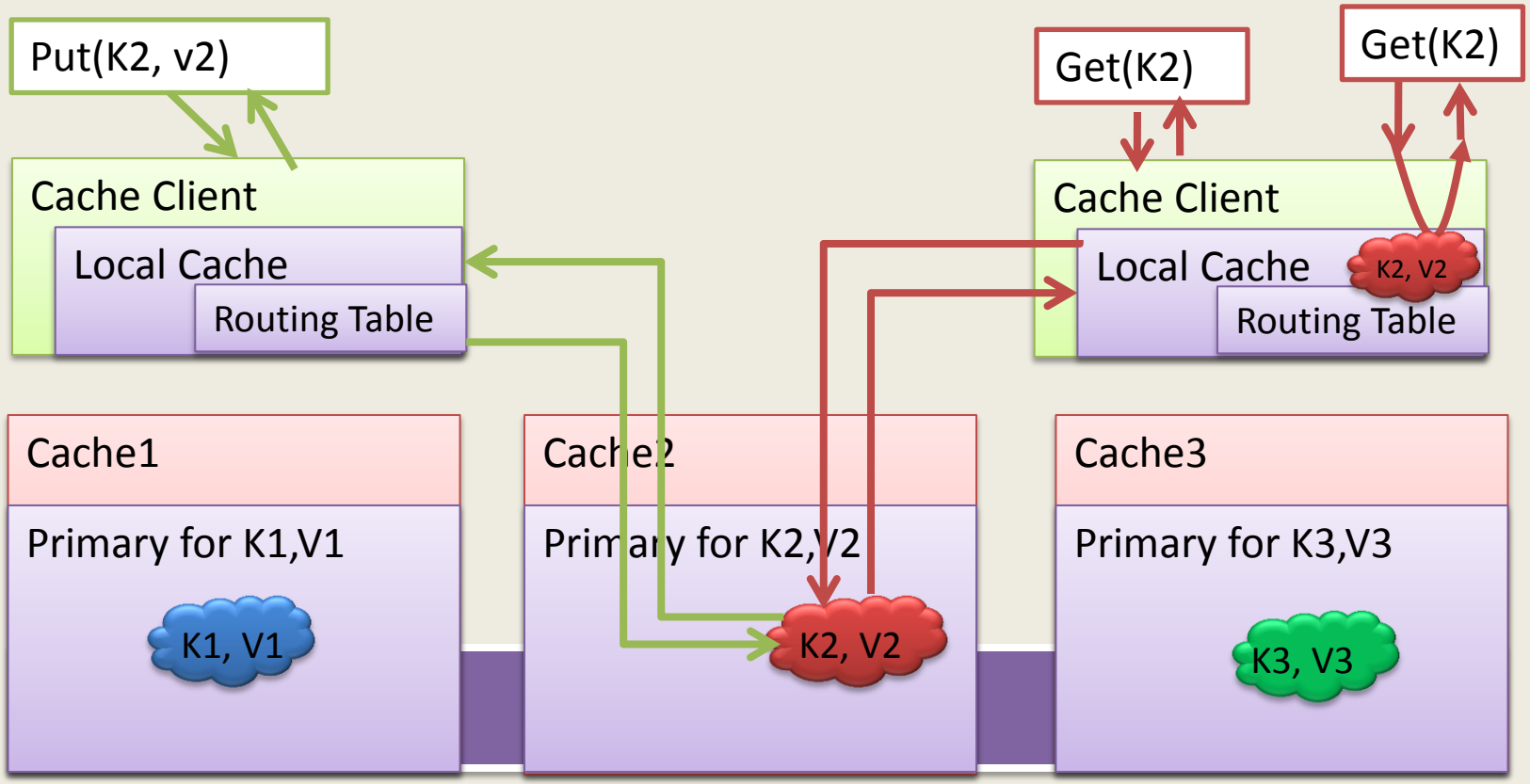


Scale: Replicated Cache (Async)

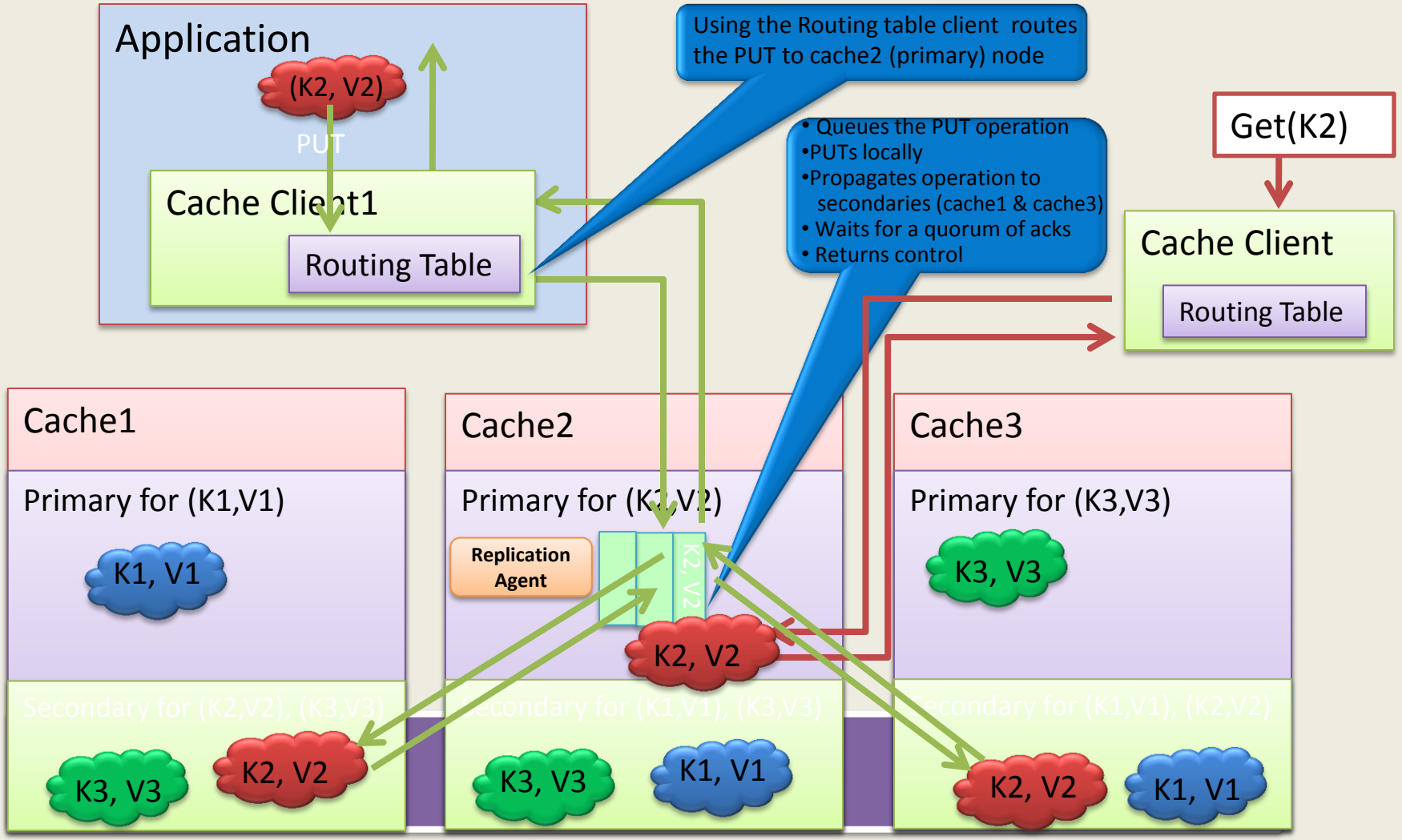


Local Cache

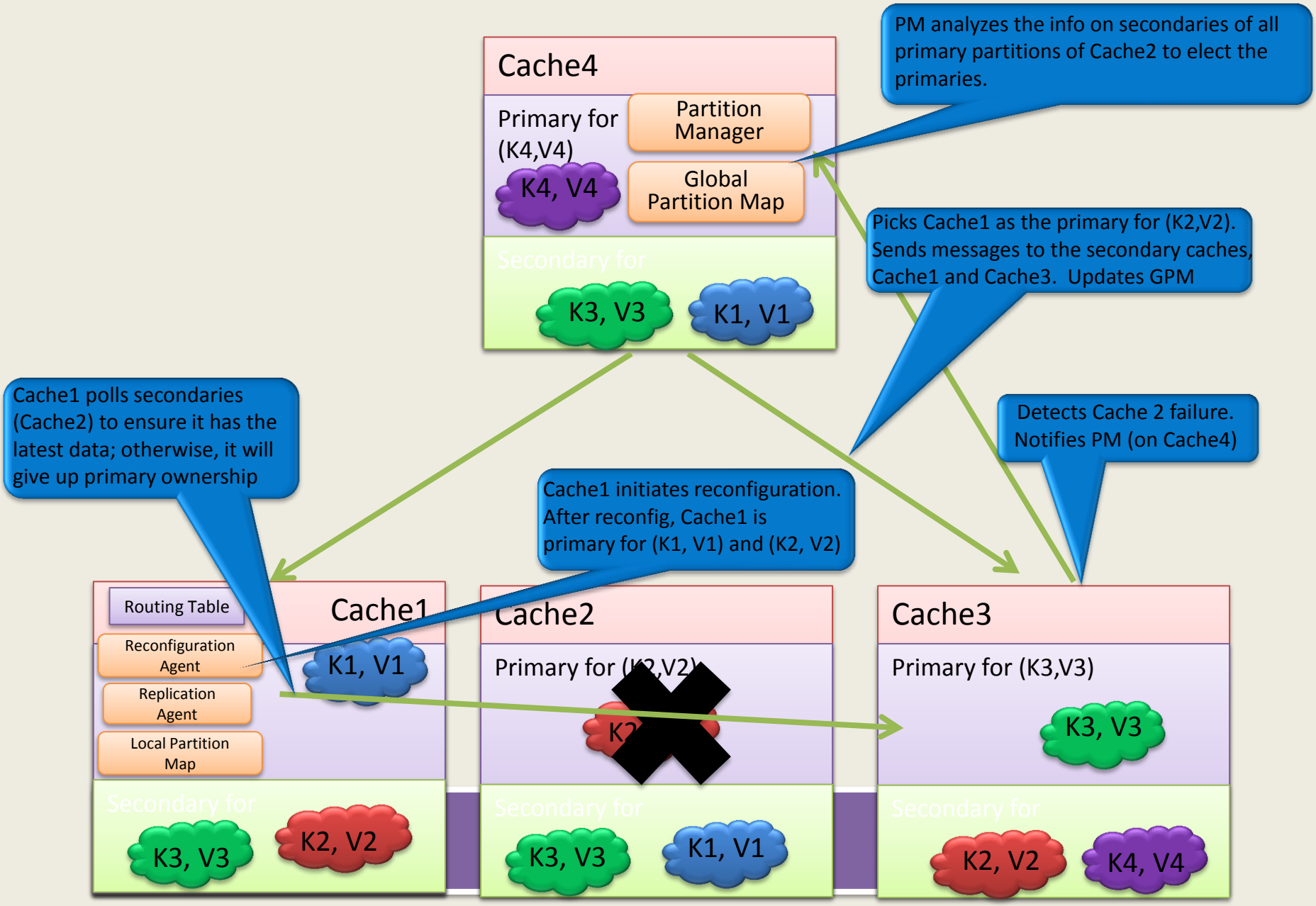
- Local Cache can help speed up access on clients
- Uses notification mechanism to refresh the cache on cache item changes



Availability

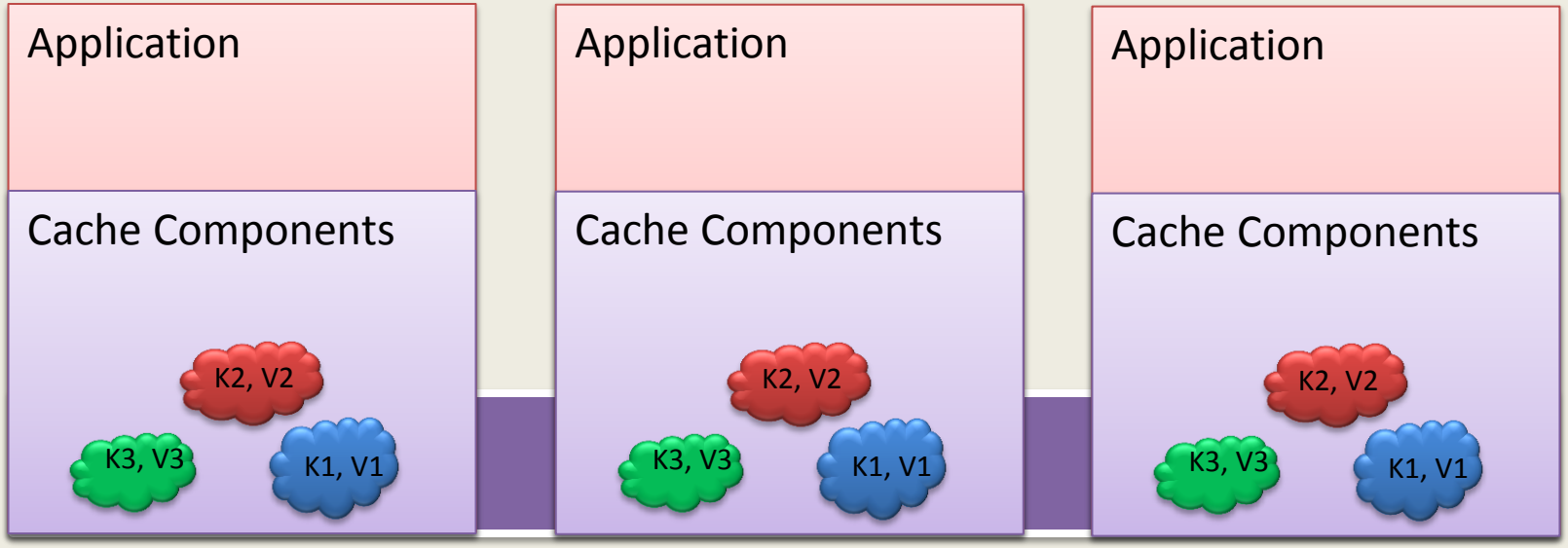


Failover


Cache4
Primary for (K4,V4)
K4, V4
Partition Manager
Global Partition Map
Secondary for
K3, V3
K1, V1
PM analyzes the info on secondaries of all primary partitions of Cache2 to elect the primaries.
Picks Cache1 as the primary for (K2,V2). Sends messages to the secondary caches, Cache1 and Cache3. Updates GPM
Cache1 polls secondaries (Cache2) to ensure it has the latest data; otherwise, it will give up primary ownership
Detects Cache 2 failure. Notifies PM (on Cache4)
Cache1 initiates reconfiguration. After reconfig, Cache1 is primary for (K1, V1) and (K2, V2)
Routing Table
Reconfiguration Agent
Replication Agent
Local Partition Map
Cache1
K1, V1
Secondary for
K3, V3
K2, V2
Cache2
Primary for (K2,V2)
Secondary for
K3, V3
K1, V1
Cache3
Primary for (K3,V3)
K3, V3
Secondary for
K2, V2
K4, V4

Embedded Cache

- Cache client and server components run as part of the application process
- Avoids serialization and network costs
- Provides high performance, low latency access
- Guaranteeing locality and load balancing is tricky
- Better suited for replicated caches



Optimistic Version-based Locking

- GetCacheItem returns a version object
- Every update to an object internally increments it's version
- Supply the version obtained along with the Put/Remove
- Put/Remove will succeed only if the passed in version matches the version in the cache

Version Based Update		
Time	Client1	Client2 (Different Thread or process)
T0	CacheItem item = catalog.GetCacheItem("PlayerRegion", "Zune");	CacheItem item = catalog.GetCacheItem("PlayerRegion", "Zune");
T1	((ZuneObject)item.Object).inventory --;	((ZuneObject)item.Object).inventory--;
T2		catalog.Put("PlayerRegion", "Zune", item.Object, item.Version);

T3	catalog.Put("PlayerRegion", "Zune", item.Object, item.Version);
	// Version mismatch
	// Client must retry again	

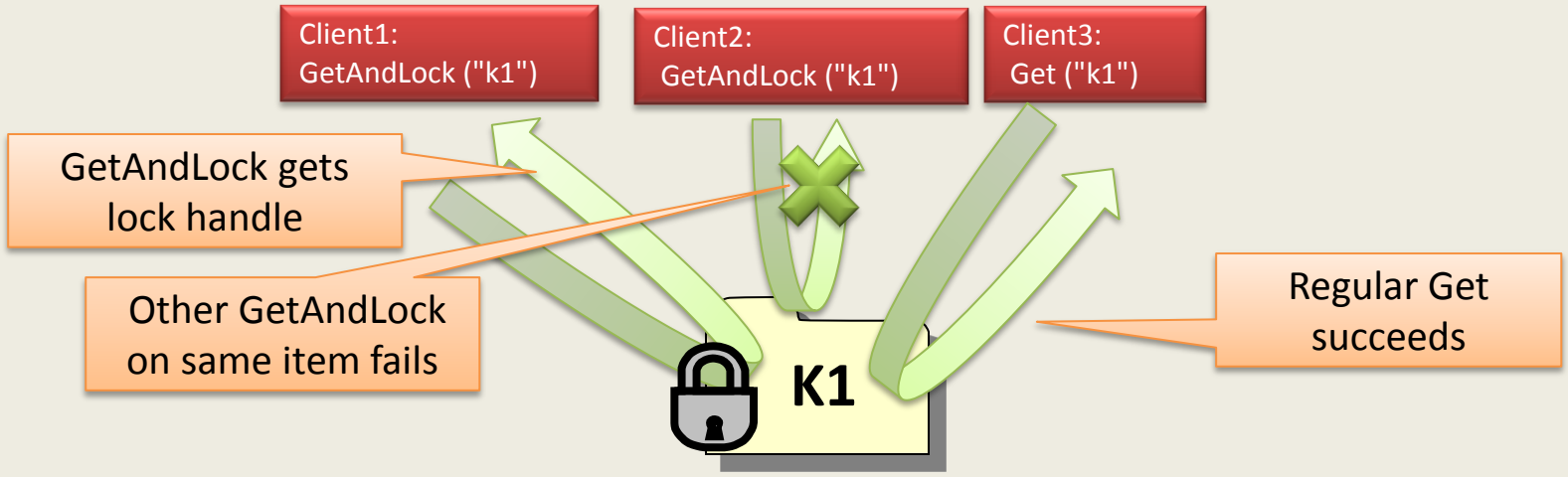
Two clients access the same item

Both update the item

Second Client gets in first; put succeeds because item version matches; atomically increments the version

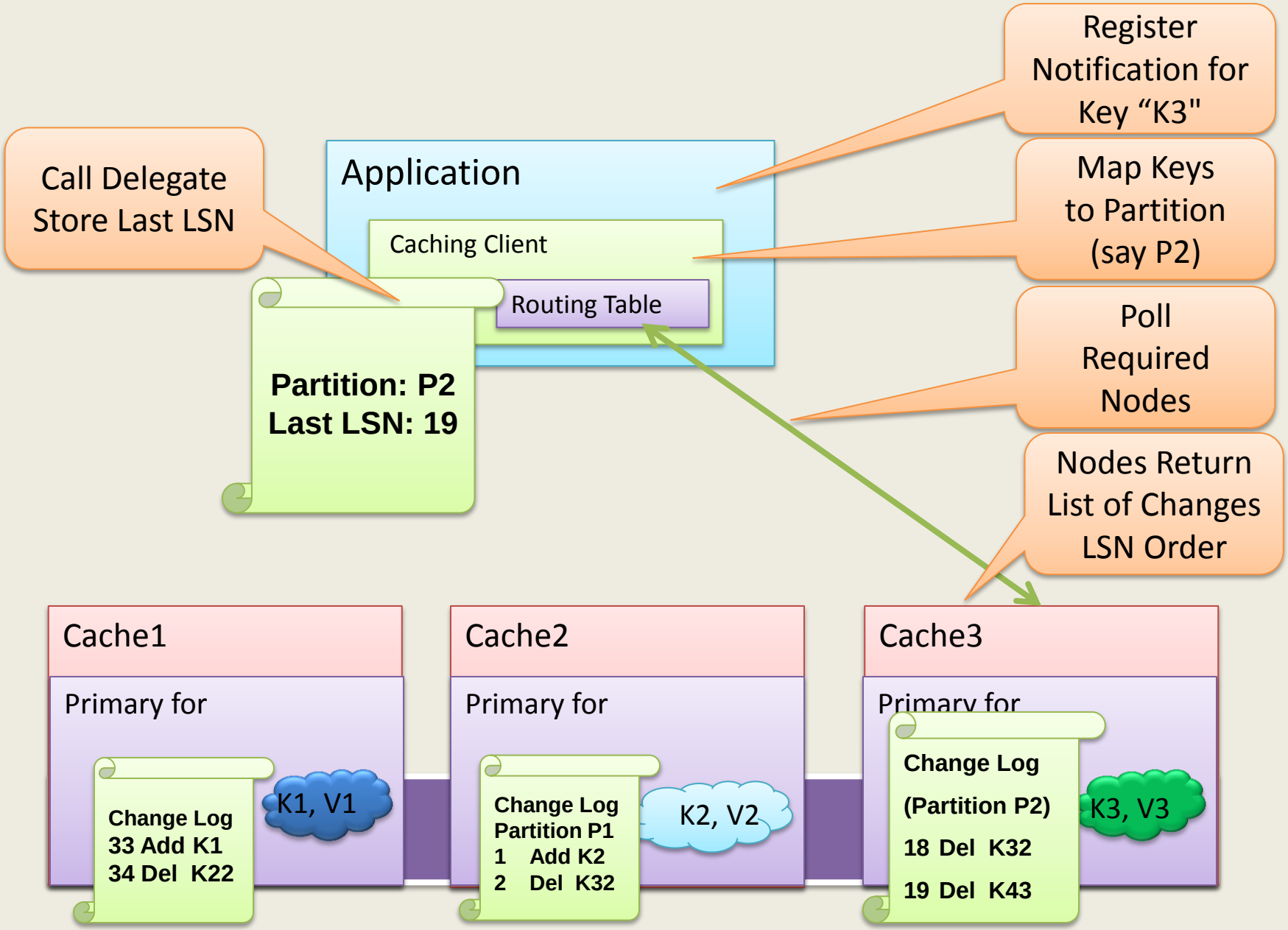
First client tries put; Fails because the versions don't match

Pessimistic Locking



- Take locks on non-existent keys
- Allows you to co-ordinate creating new object amongst multiple clients

Scalable Notifications



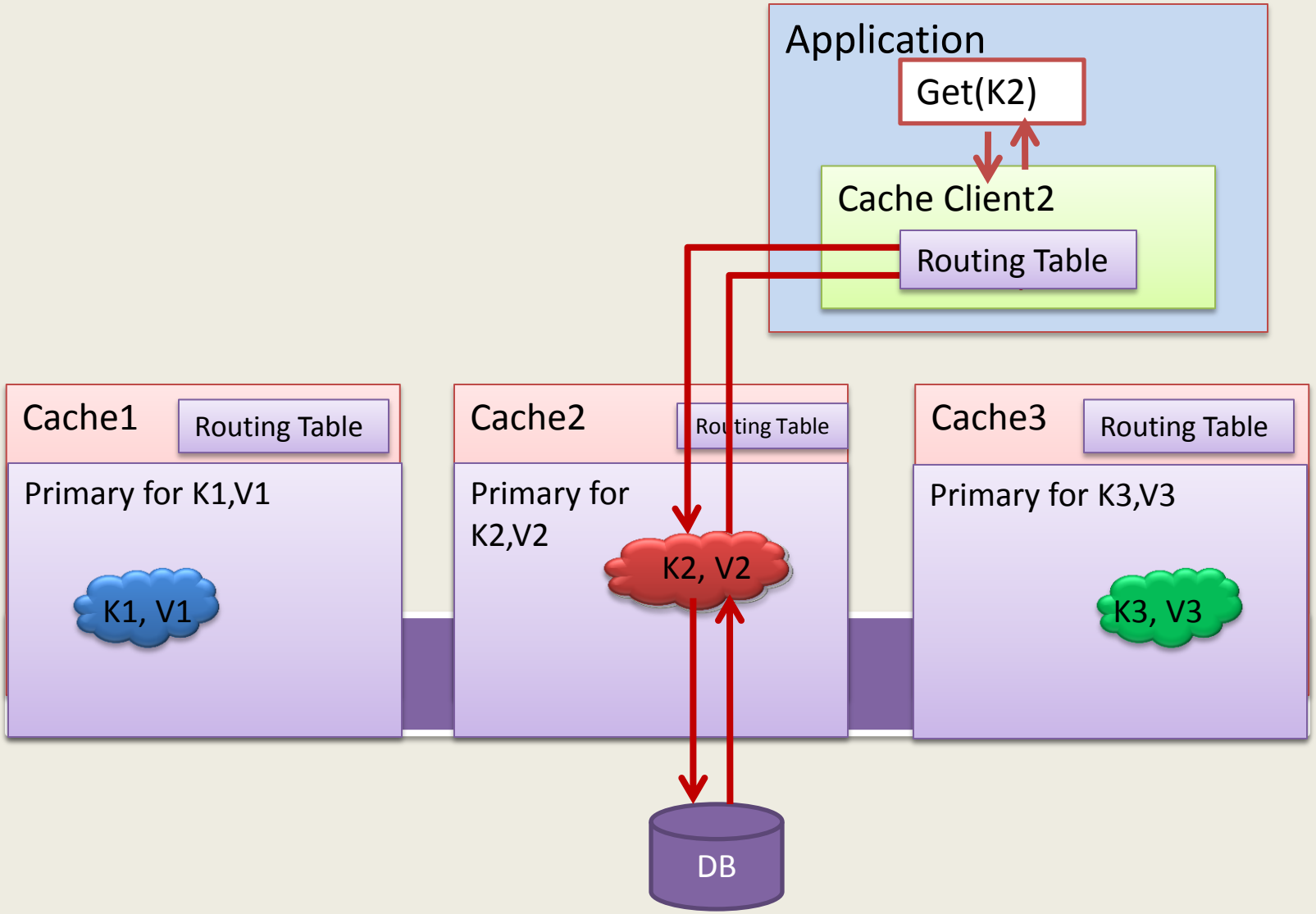
Eviction

- Expiry only eviction which
  - Evicts expired items alone
  - Periodic
  - Per partition
- Hard-eviction (Data > Allocated Cache Size)
  - Evicts expired items + non-expired items (in LRU order)
  - Per request
  - Can be turned off
- Memory pressure based eviction
  - A thread for detecting memory pressure (polling per second)
  - Avoids paging
  - Triggers hard-eviction (mentioned above) at 85% system memory usage and asks for releasing 5% of system memory

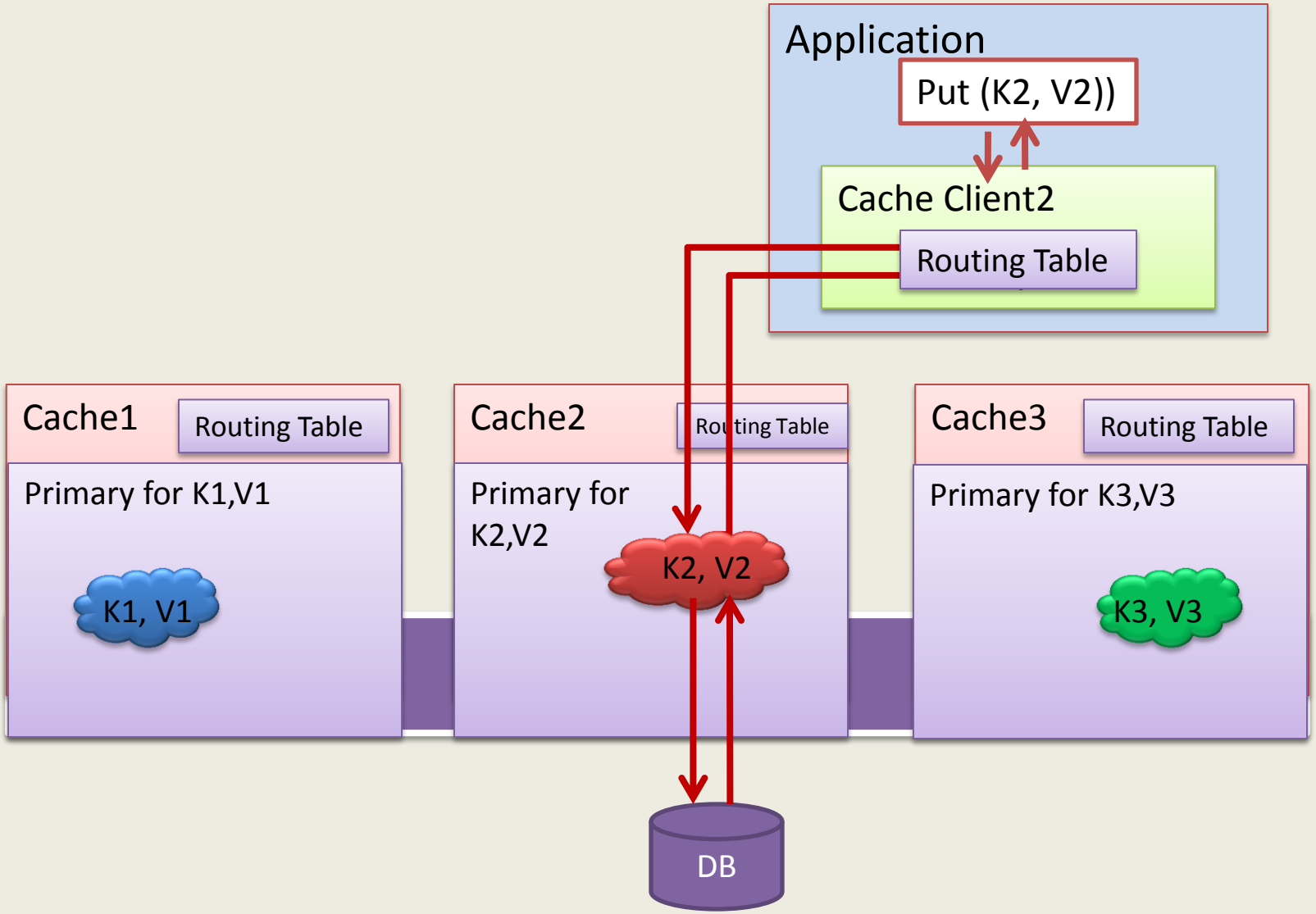
Persistence – Cache Through

- Callback for read-through, write-through, write-behind
- Specified at Named Cache Level
- Read-Through
  - Called when item not present in cache
  - Callback returns the object/serialized bytes
- Write-Through
  - Called when item is put
- Write-Behind
  - Writes to cache are queued
  - Callback called asynchronously in batches
  - Re-tries upon failure
- Bulk Access APIs

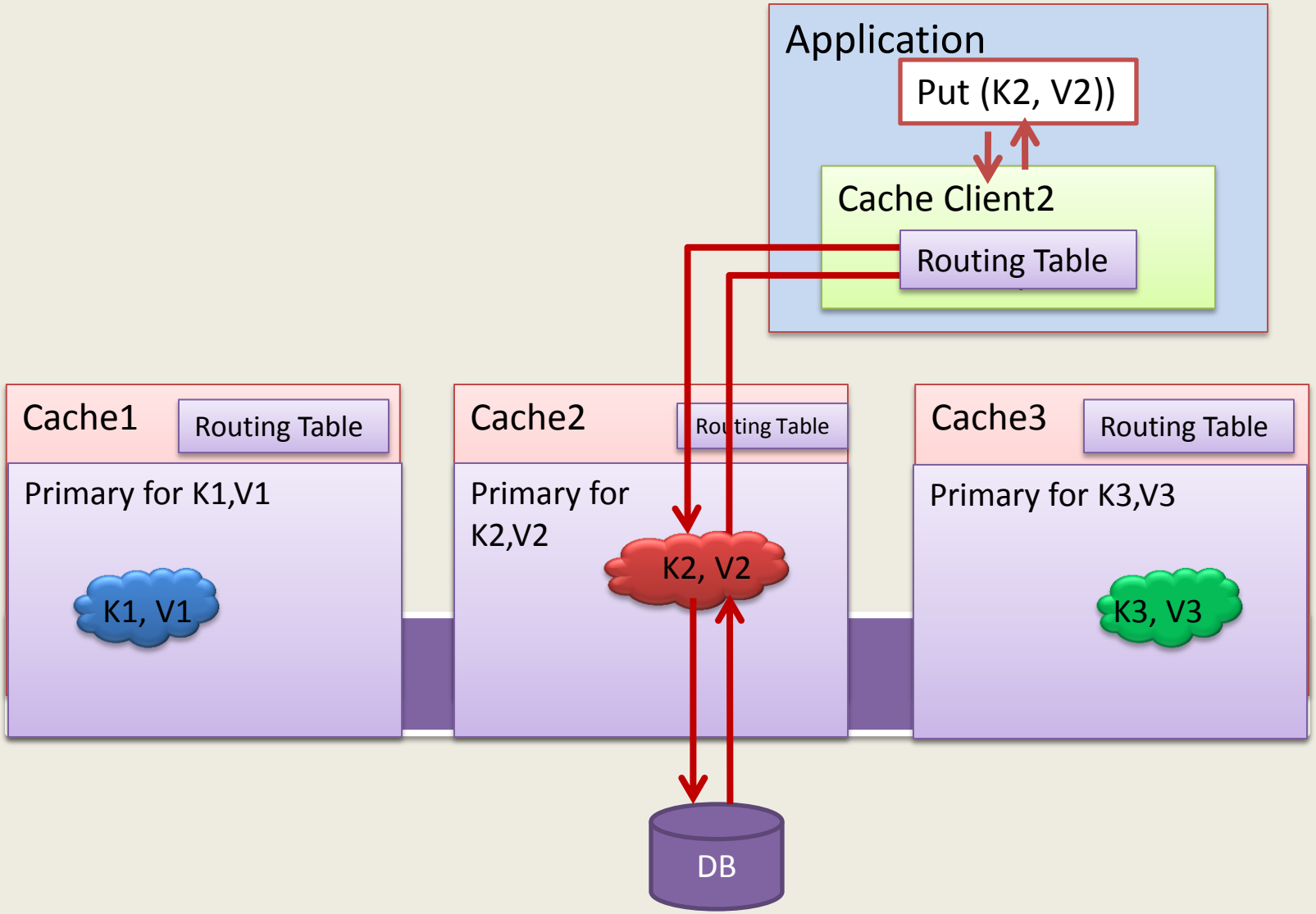
Read-Through Cache



Write-Through Cache



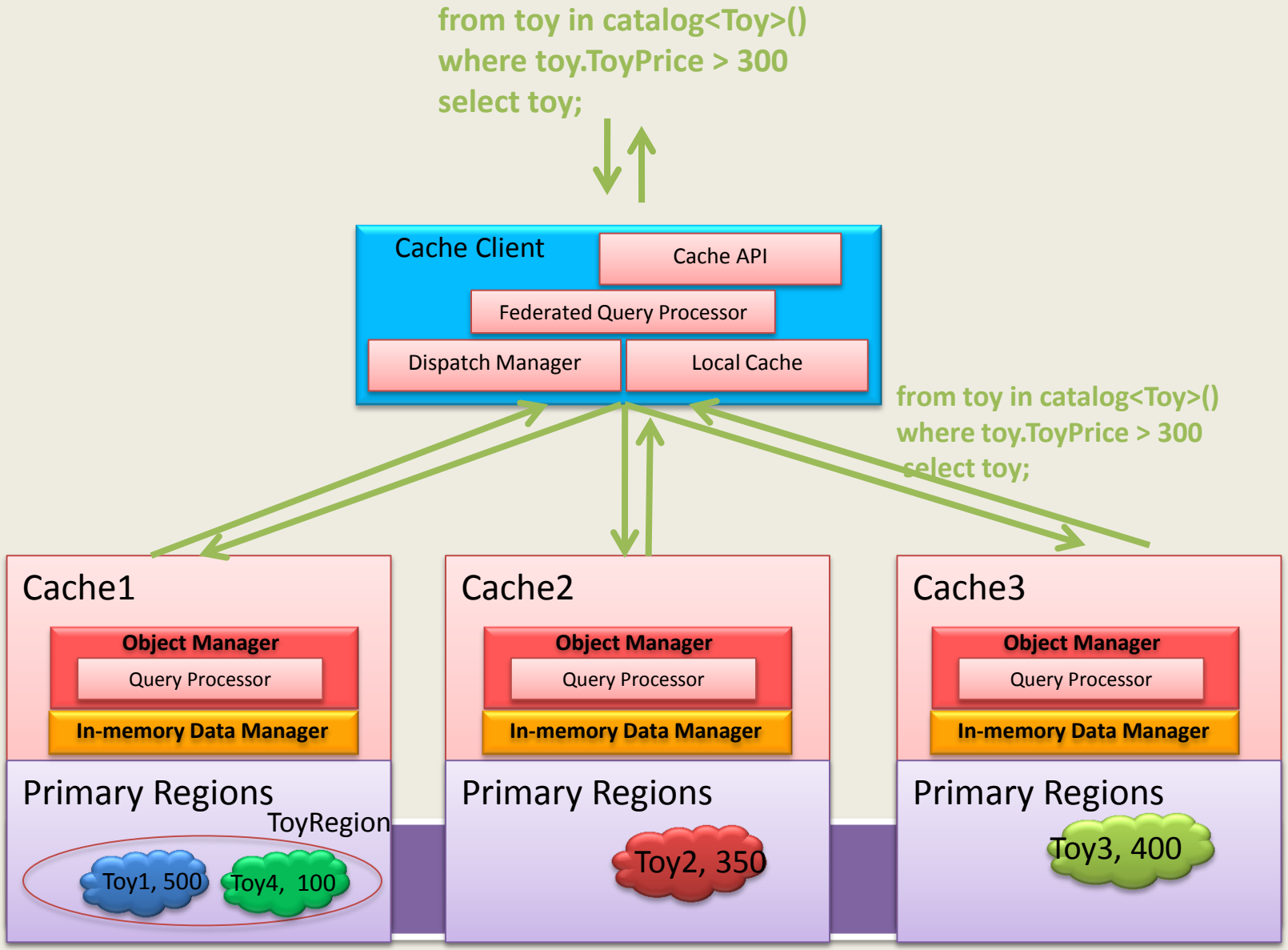
Async Write-Back Cache



Async Write Back (Write Behind) Cache

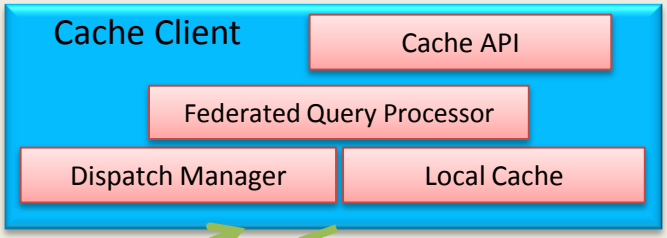
- Specified at Named Cache Level
- Write-Back
  - Asynchronously written to disk (e.g. database)
  - Physical write done via callbacks
  - Writes to cache are queued
  - Callback called asynchronously in batches
  - Re-tries upon failure

Executing A Query

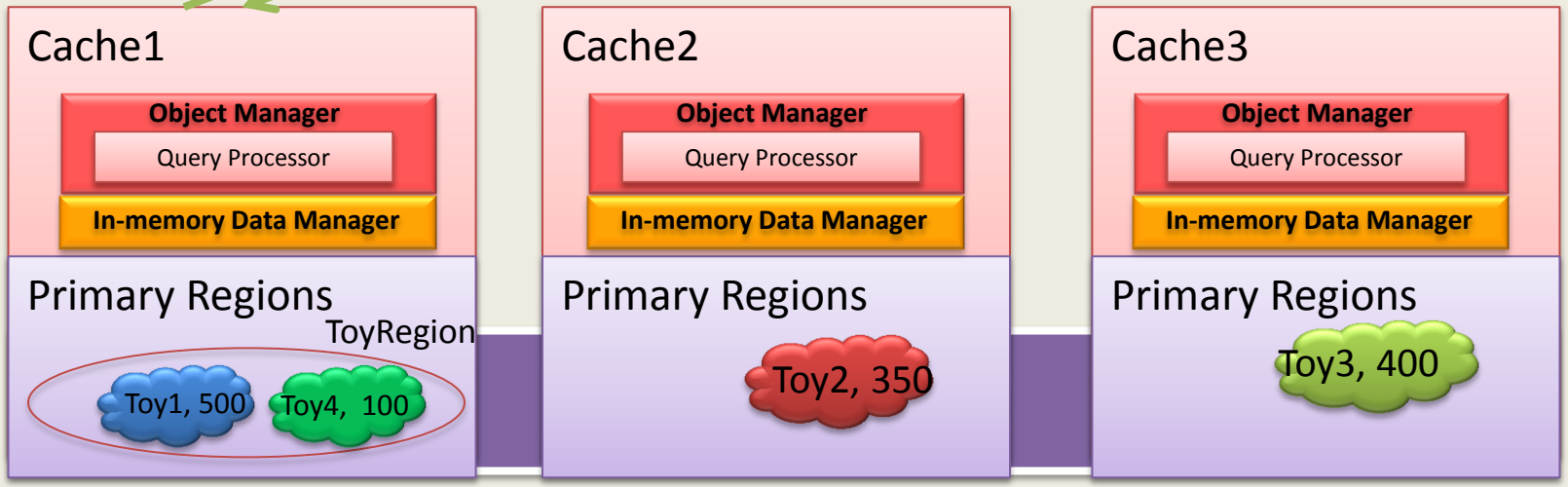


Executing A Query

```
from toy in catalog.GetRegion<Toy>("ToyRegion")
where toy.ToyPrice > 300
select toy;
```

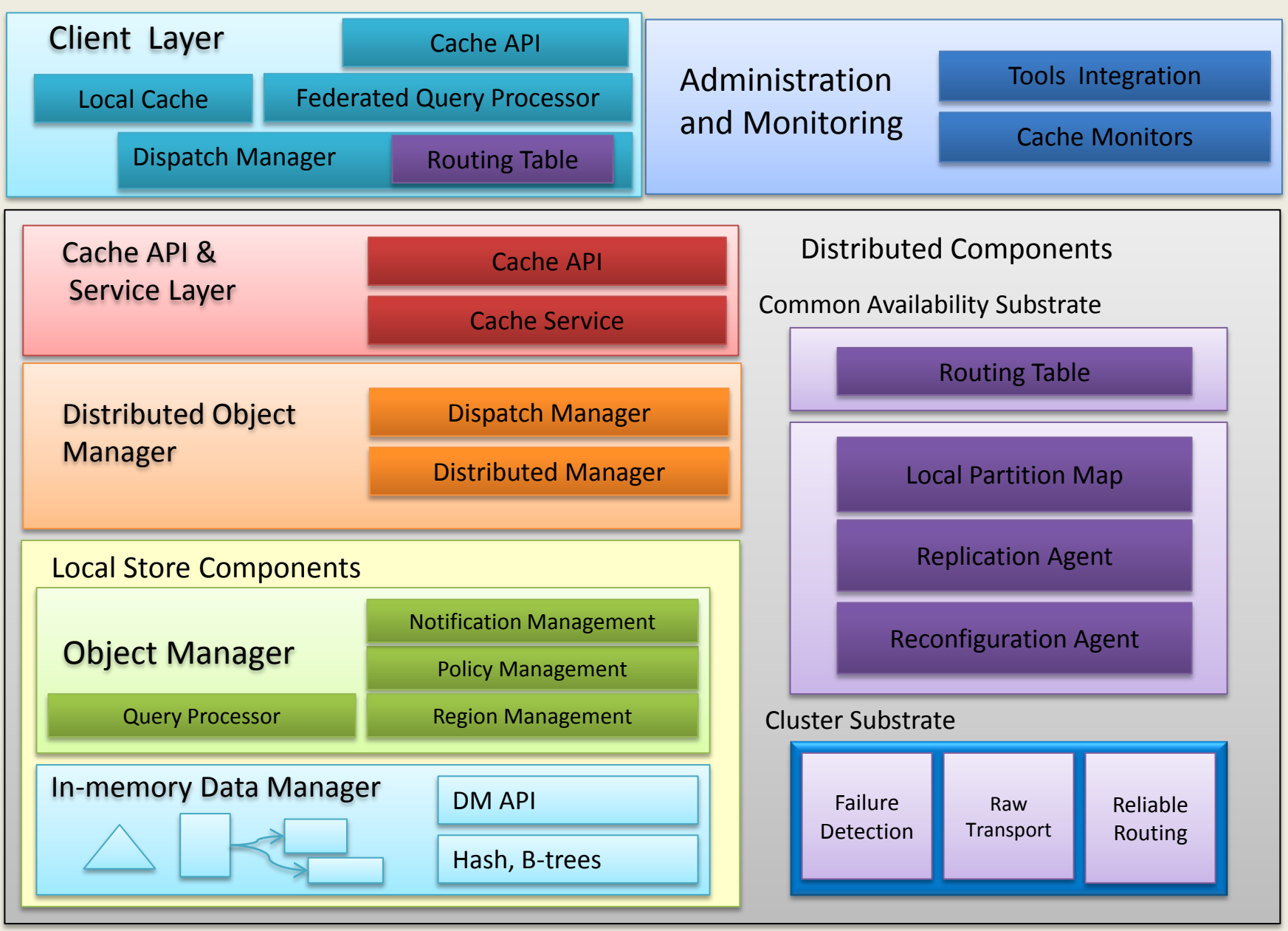



```
from toy in catalog.GetRegion<Toy>("ToyRegion")
where toy.ToyPrice > 300
select toy;
```



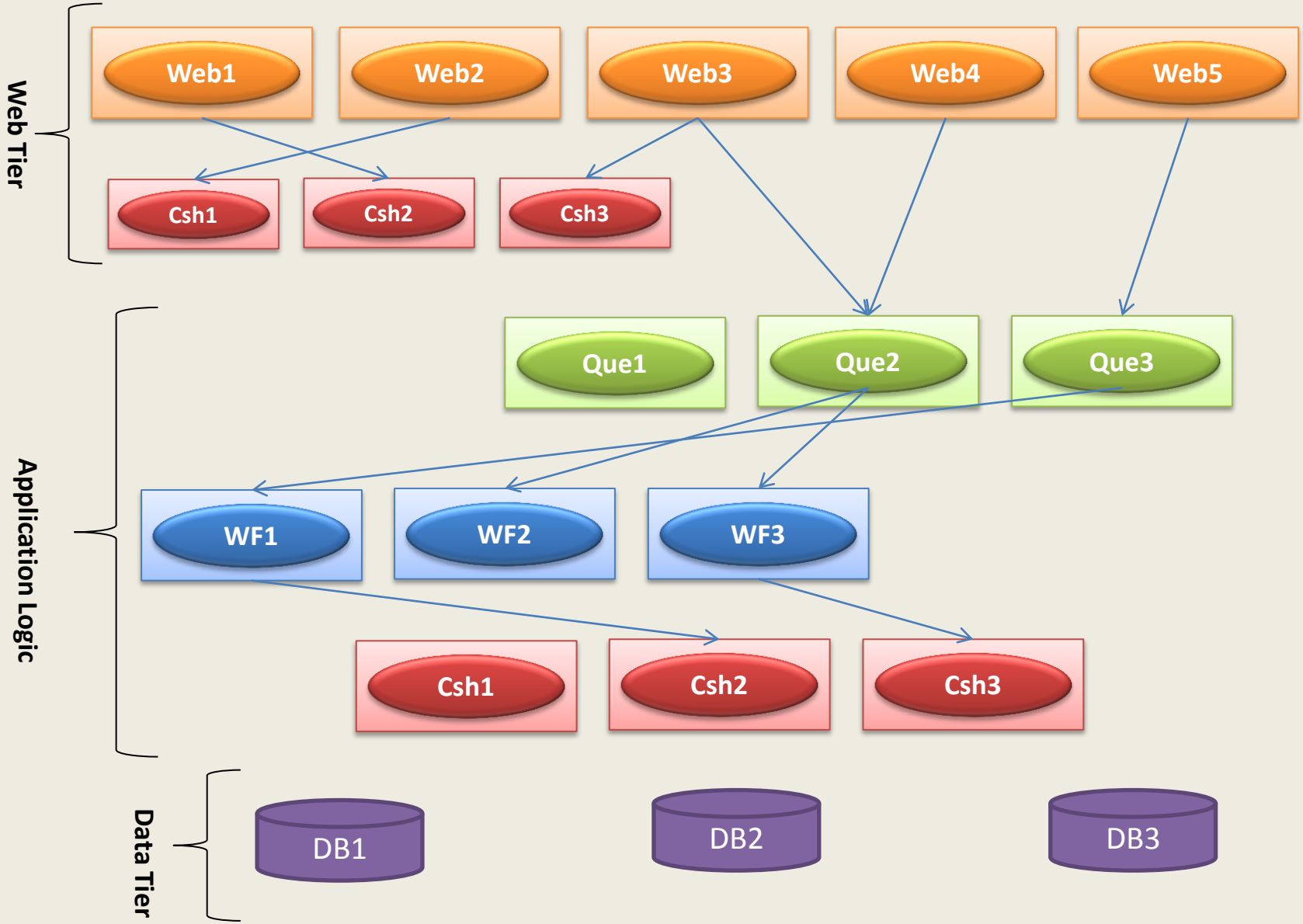
DCP Architecture

Microsoft's AppFabric Caching Architecture

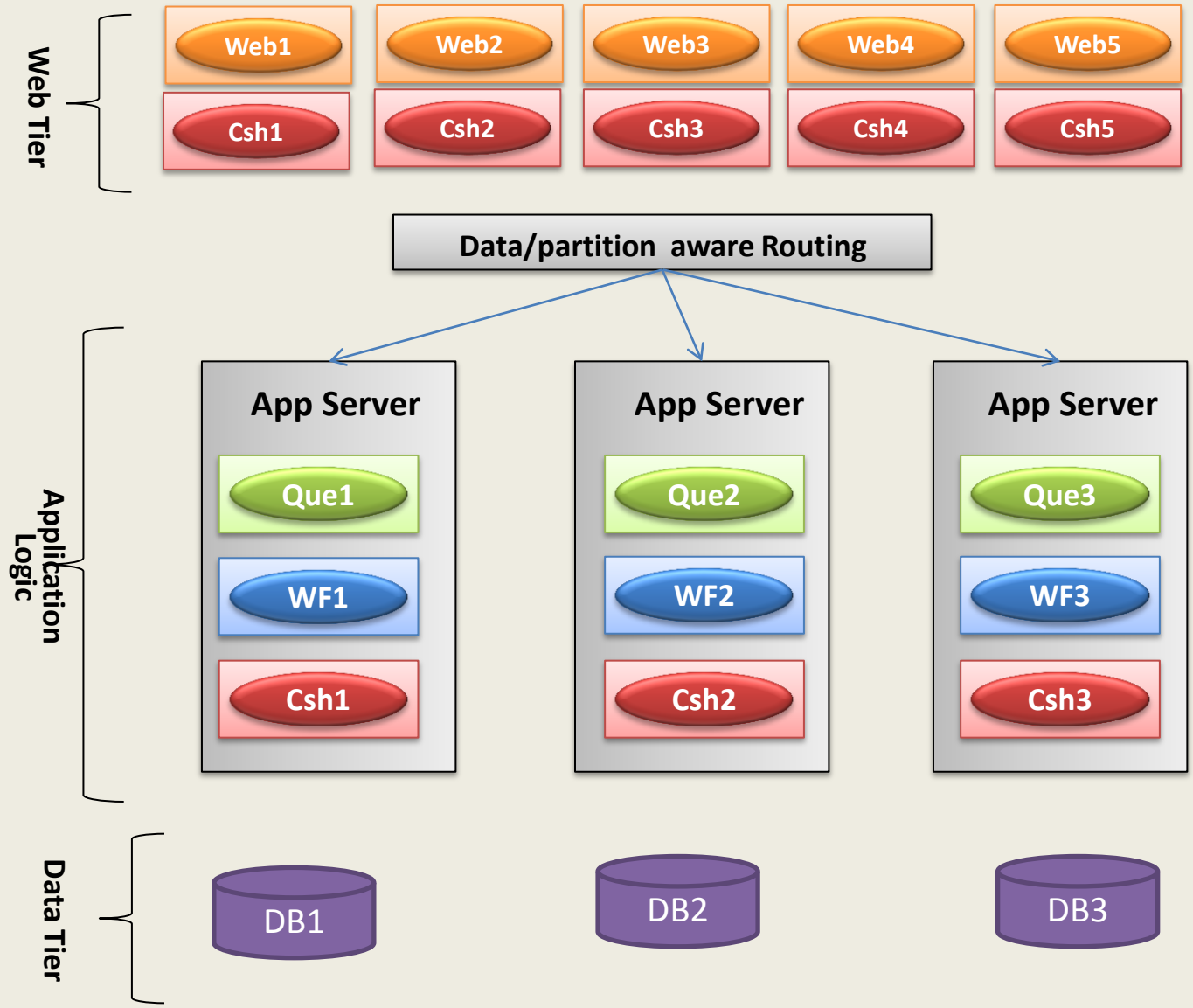


Customer & Usage Trends

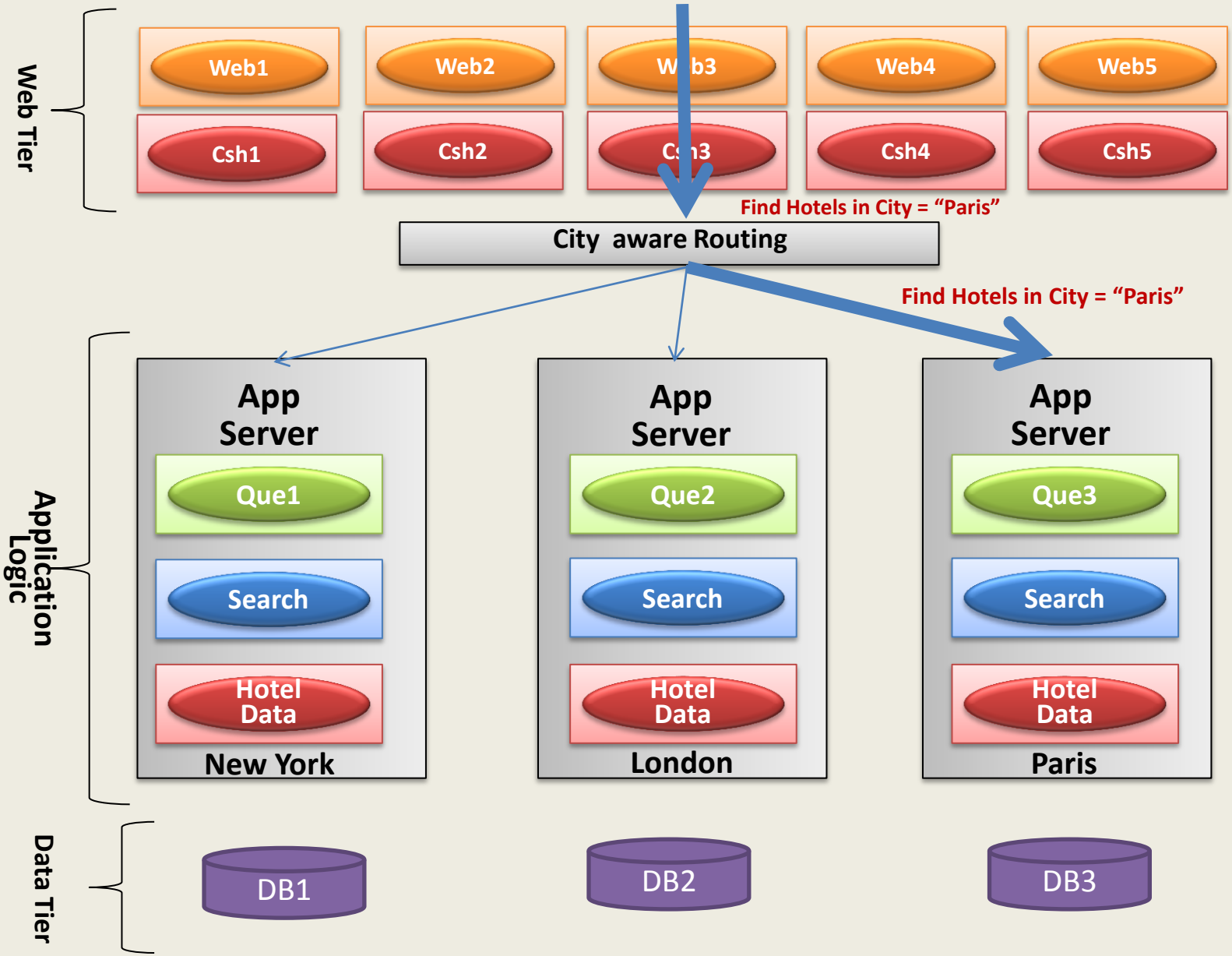
Cache in Multi-tiered Application



Tier Merging – Co-locating Caches



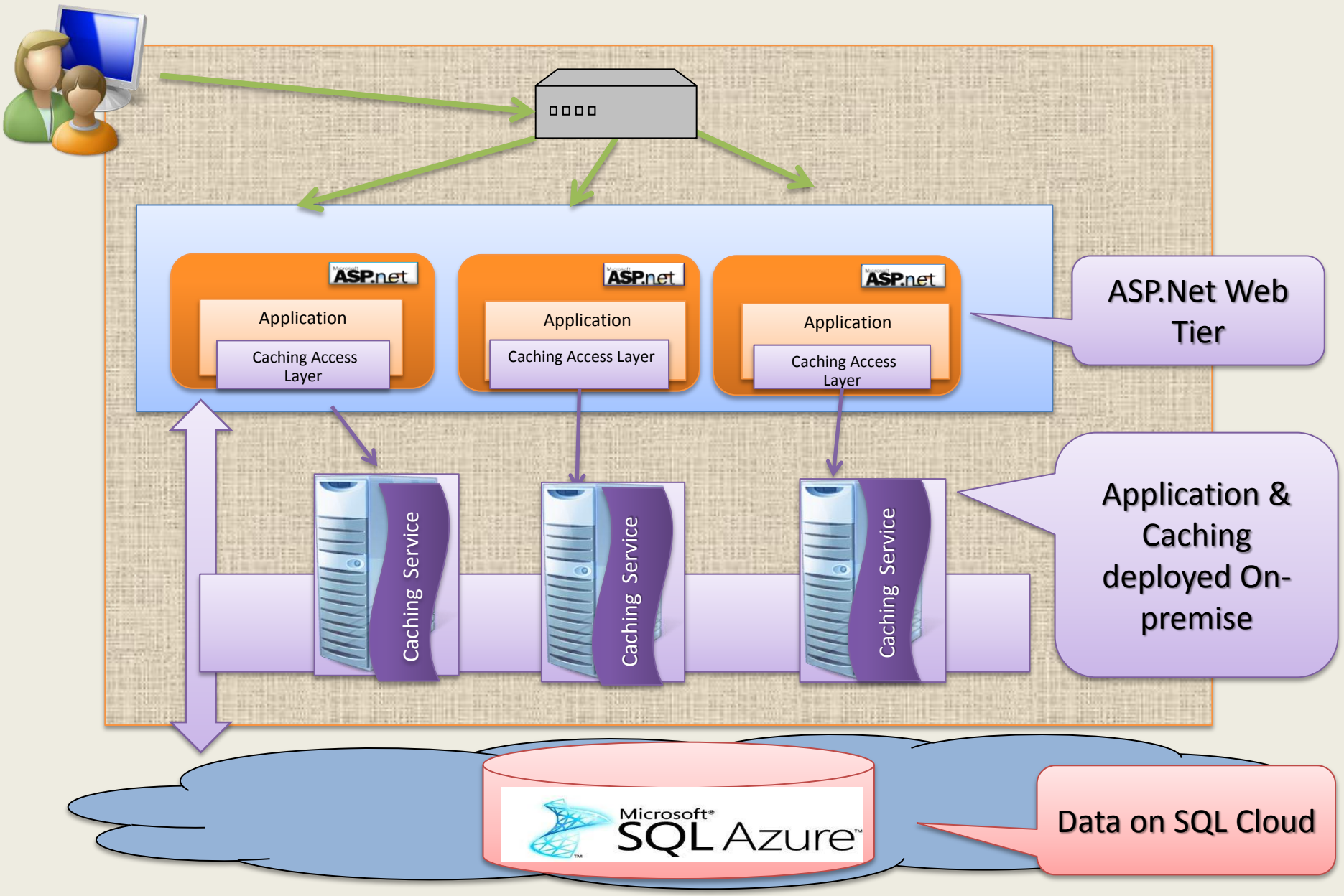
Hotel Search



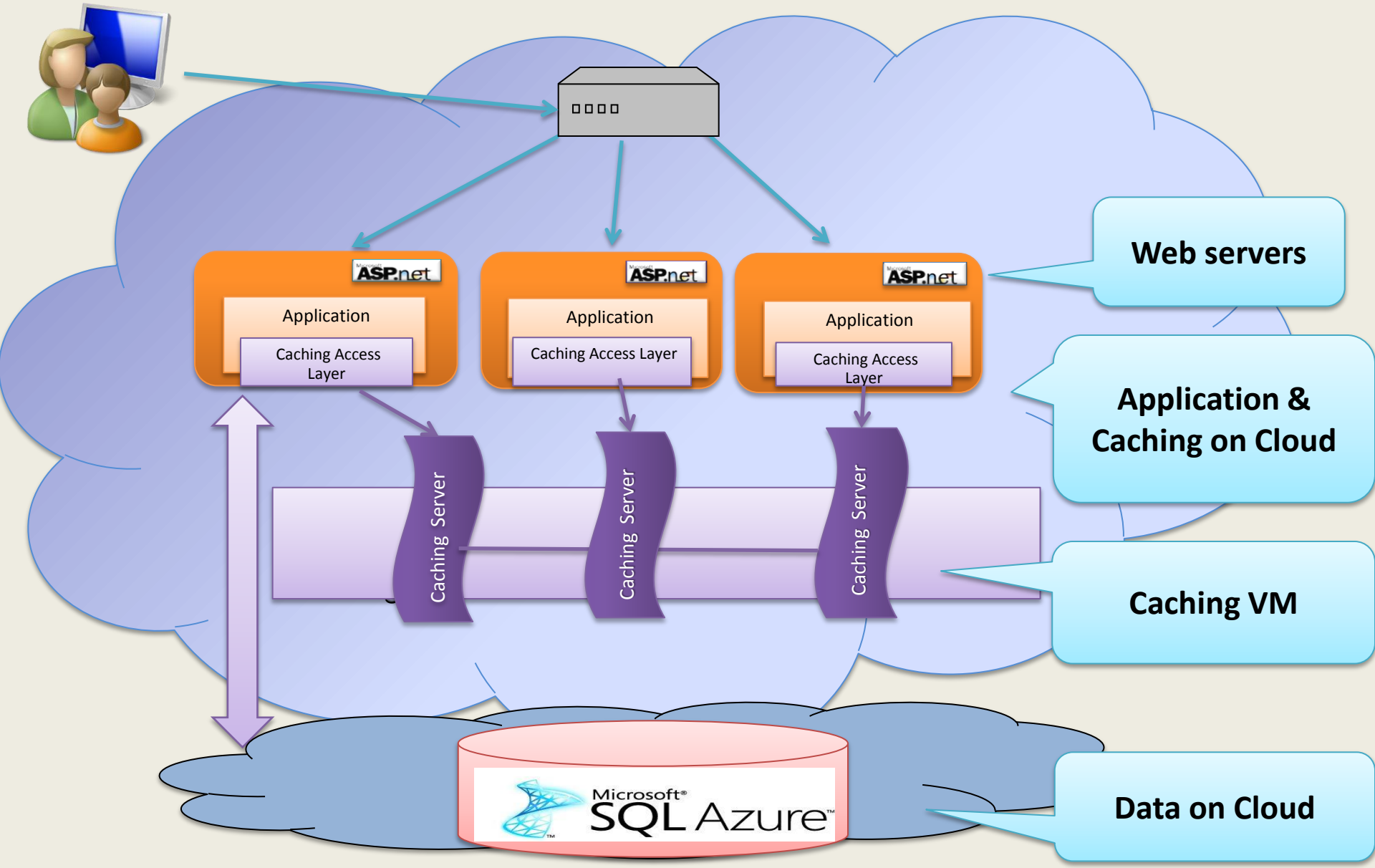
Cloud Applications and Caching

- Application (and cache) on-premises and Data on Cloud
- Application and Data on Cloud
  - Cache as a service
  - Cache co-located with App
- Application on Cloud and Data on-premises

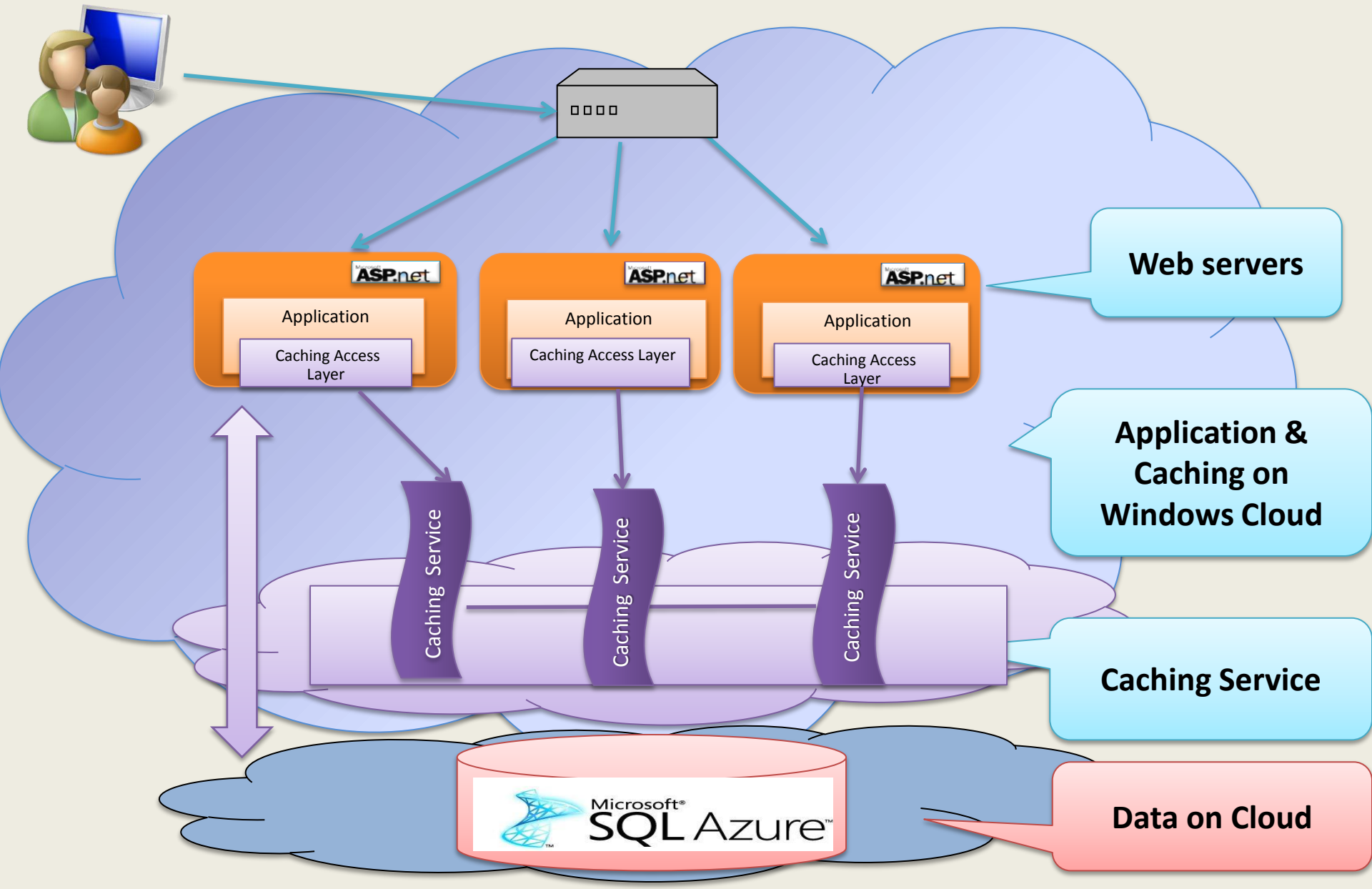
App on-premises; Data on Cloud



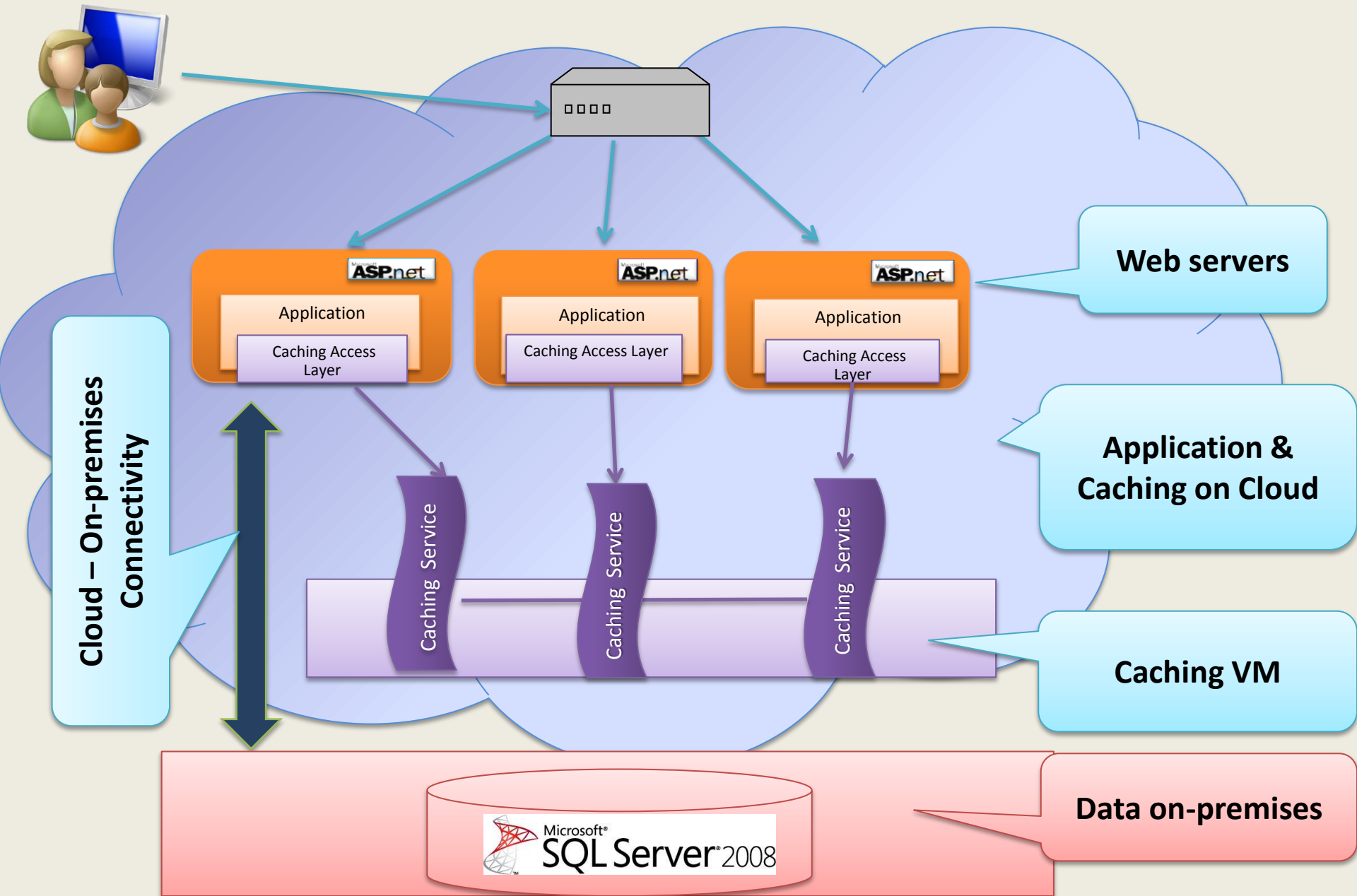
App on Cloud; Data on Cloud; Cache on a VM



App on Cloud; Data on Cloud; Cache as a Service



App on Cloud; Data on-premise



DCP Vendors

- Memcached (open source)
- VMWare (Gemstone) Gemfire
- Gigaspaces Extreme Application Platform
- IBM WebSphere Extreme Scale Cache
- Microsoft AppFabric Caching
- Oracle Coherence
- Terracotta's Terracotta Server (open source)

Distributed Caching Hard Problems

- Large caches
- Extreme Low Latency
- Impact of NVRAM technologies
  - PCM?
- Cache as the Truth?
- Durability?, Persistence?
- DBMS Capabilities?

Q/A?