GIGASPACES

WRITE ONCE.
SCALE ANYWHERE.

Scale-out your Tier-Based Systems in 3 steps Using Spring

Nati Shalom

CTO GigaSpaces

Agenda

- Drivers for scalability
- Tier based approach and its inherent bottlenecks
- A three-steps approach for achieving scalability
- Transparent migration using Spring-based abstractions
- Comparing both approaches
- Summary

The Business and Technology Drivers

- Business driver: Must process an increasing volume of information faster in a global marketplace
- Technology challenge: Need a cost-effective solution to scale distributed applications easily while maintaining high performance and resiliency

Capital Markets:

Algorithmic trading Market Data Risk Analysis Portfolio Analysis Surveillance/Compliance

Telecom:

Real-time billing, Order Management, VOIP, Location-based services, Mobile device content

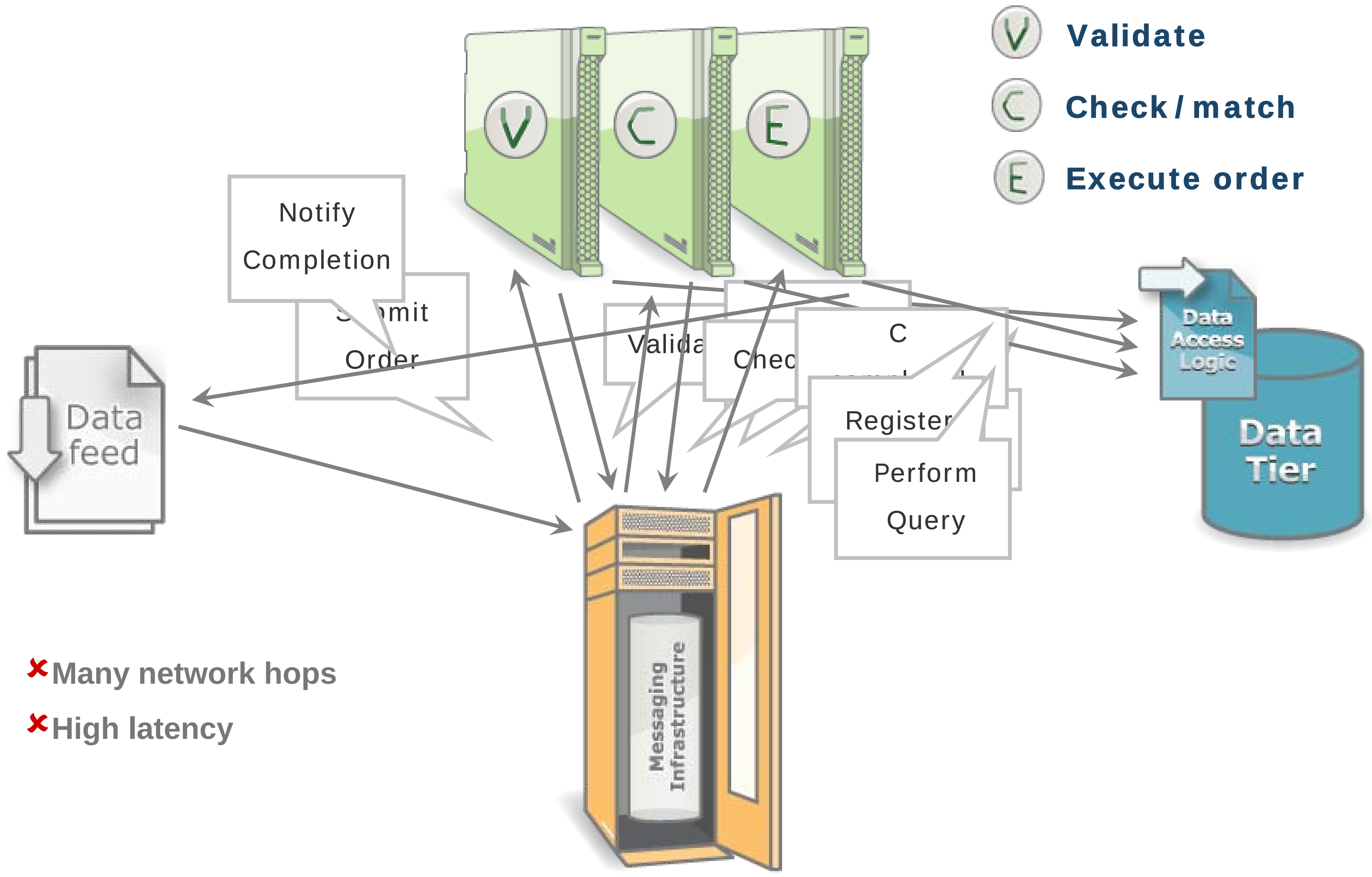
On-Line:

Gaming, Travel, Advertising/Marketing, Commerce, Consumer portals, Search engines

Defense

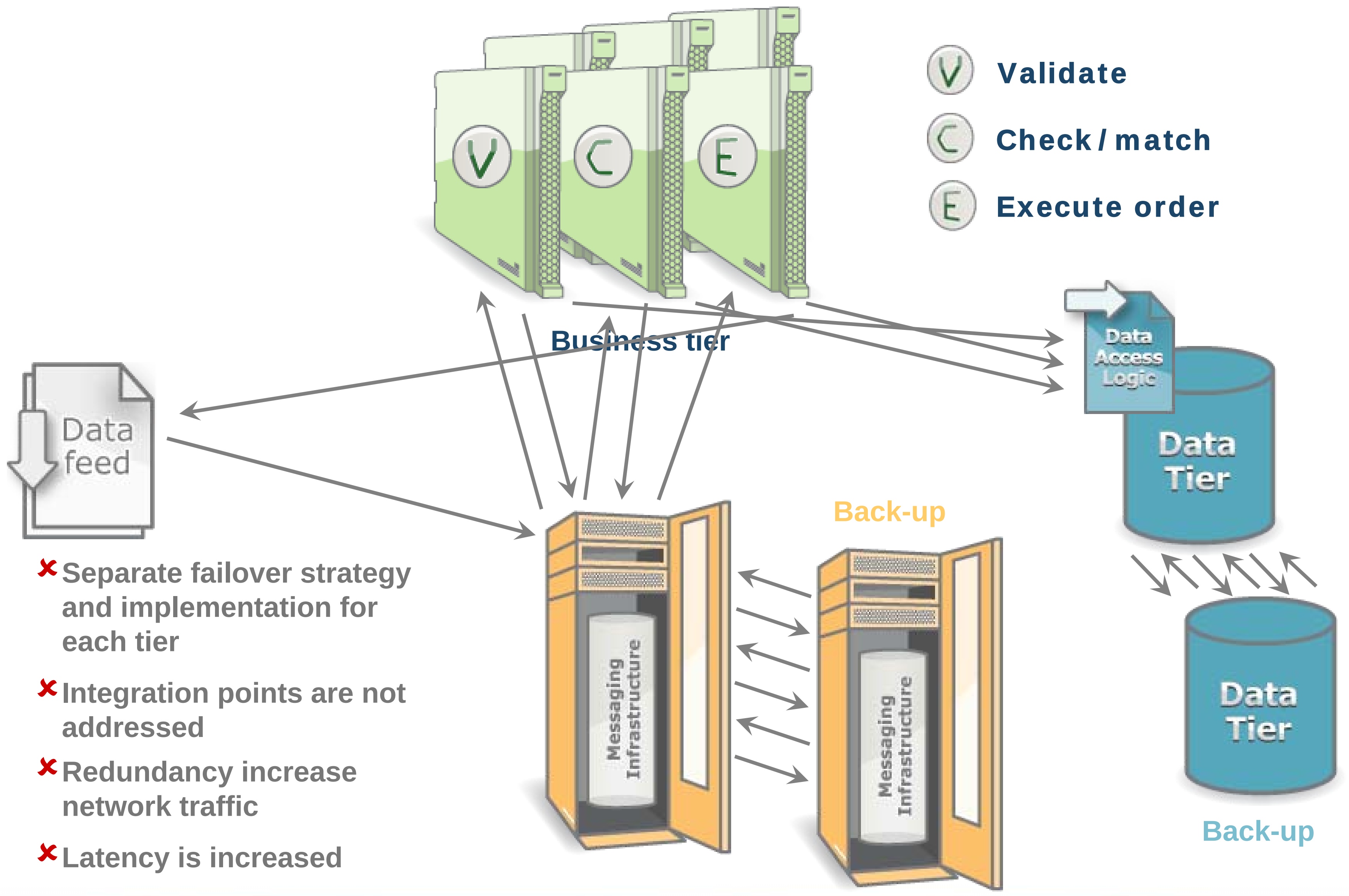
Real-time intelligence, Pattern Analysis

A Transaction Flow Example - Order Management



- ✗ Many network hops
- ✗ High latency

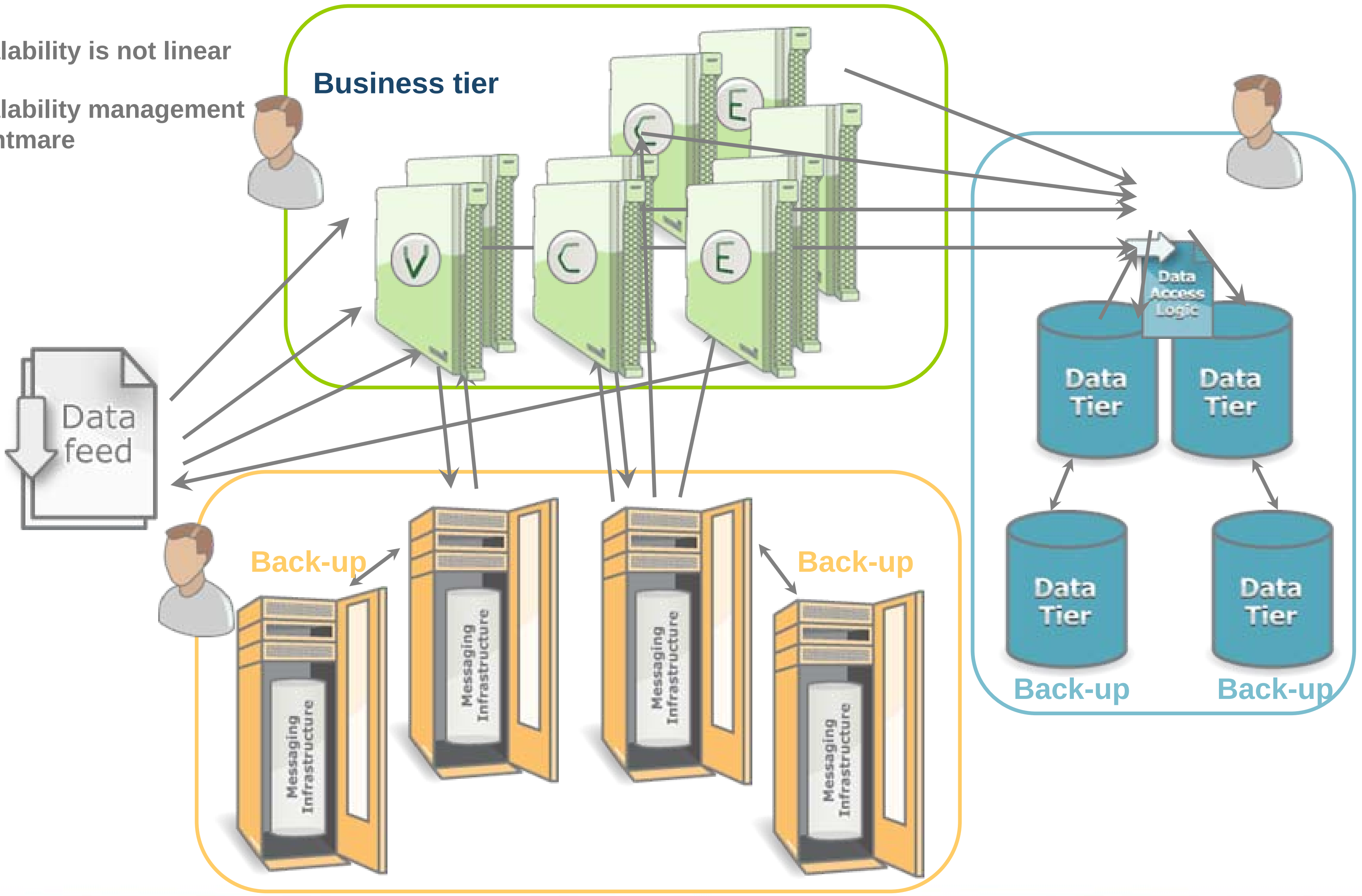
Maintaining Resiliency in a Traditional Tiered Application



- ✗ Separate failover strategy and implementation for each tier
- ✗ Integration points are not addressed
- ✗ Redundancy increase network traffic
- ✗ Latency is increased

Scaling and Managing a Traditional Tiered Application

- ✗ Scalability is not linear
- ✗ Scalability management nightmare



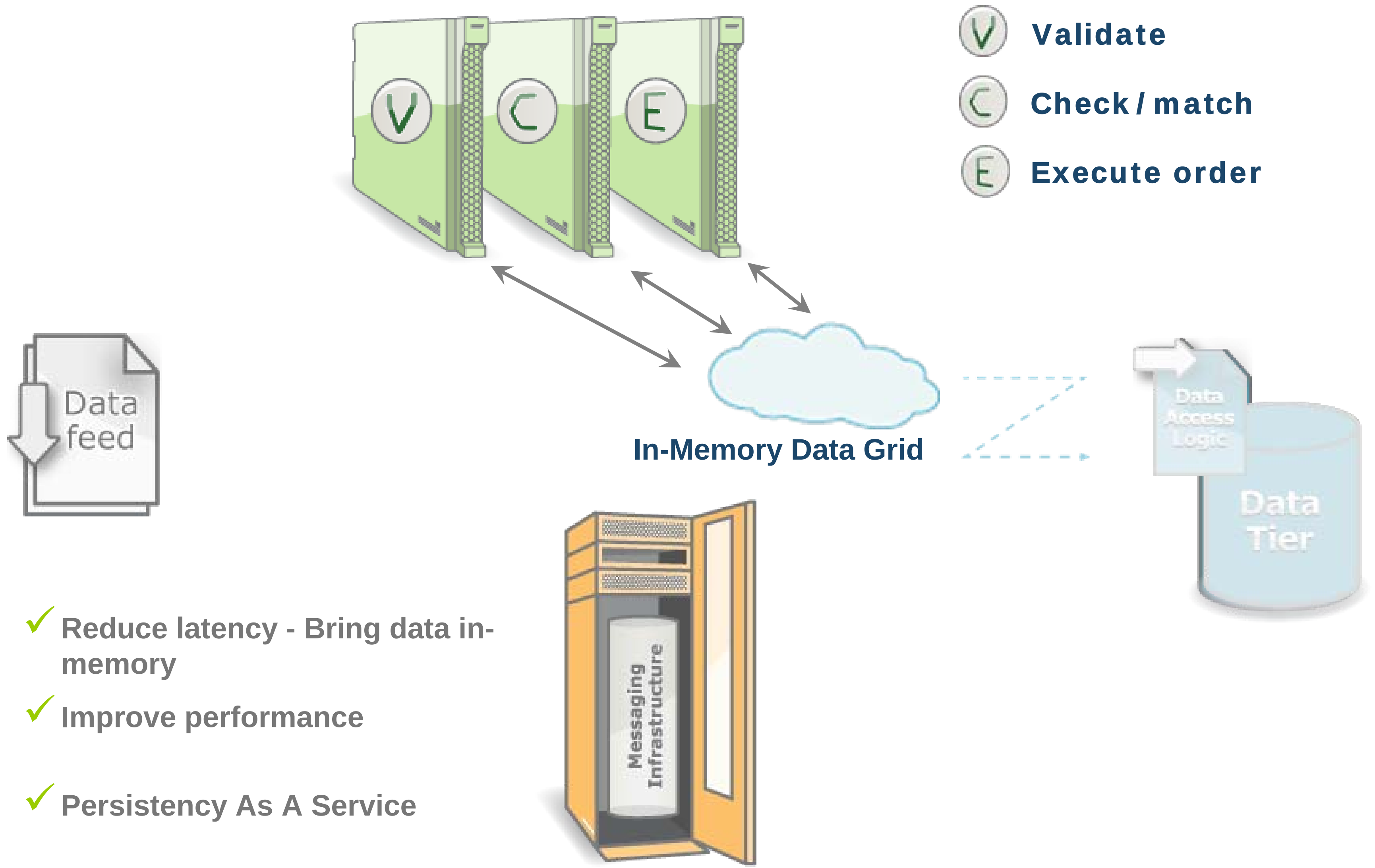
Simple Scale-out of a Tiered Application in 3 Steps

1. Reduce I/O Bottleneck using an In-Memory Data Grid
   - Bring data in-memory
   - Improve performance
   - Persistency As A Service – persist only for compliance & reporting purposes

2. Consolidate the ESB and Data
   - Address data affinity between the messaging infrastructure and the data tier
   - Reduce the number of moving parts
   - Single cluster – reduce redundancy

3. Assemble the business logic together with the data and messaging
   - Create a single, efficient process to scale your application
   - Ensure a single built-in failover/redundancy investment strategy
   - Simplify the process of scaling and deployment

Step 1:
Reduce I/O Bottleneck using In-Memory-Data Grid



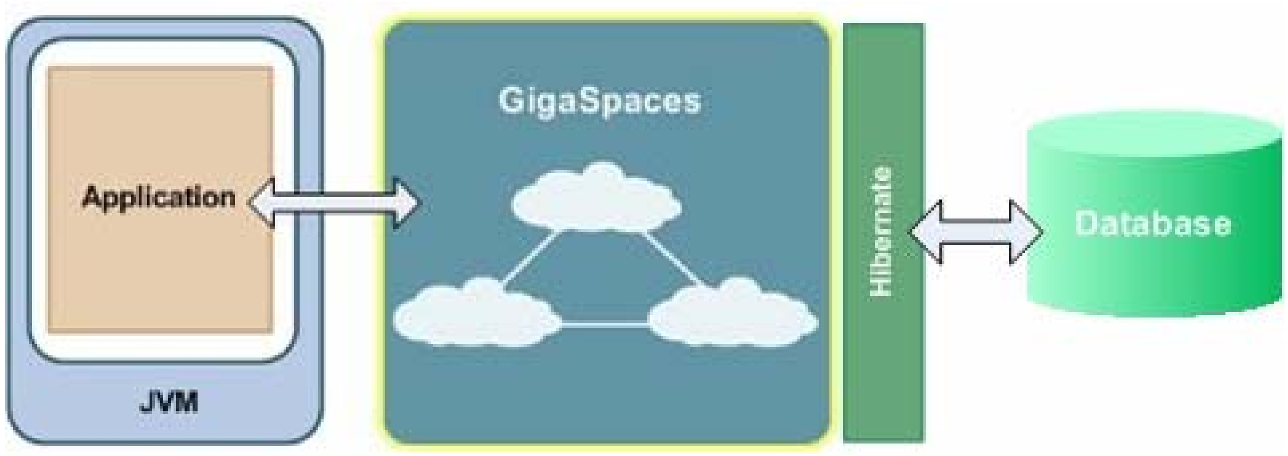
✓ Reduce latency - Bring data in-memory

✓ Improve performance

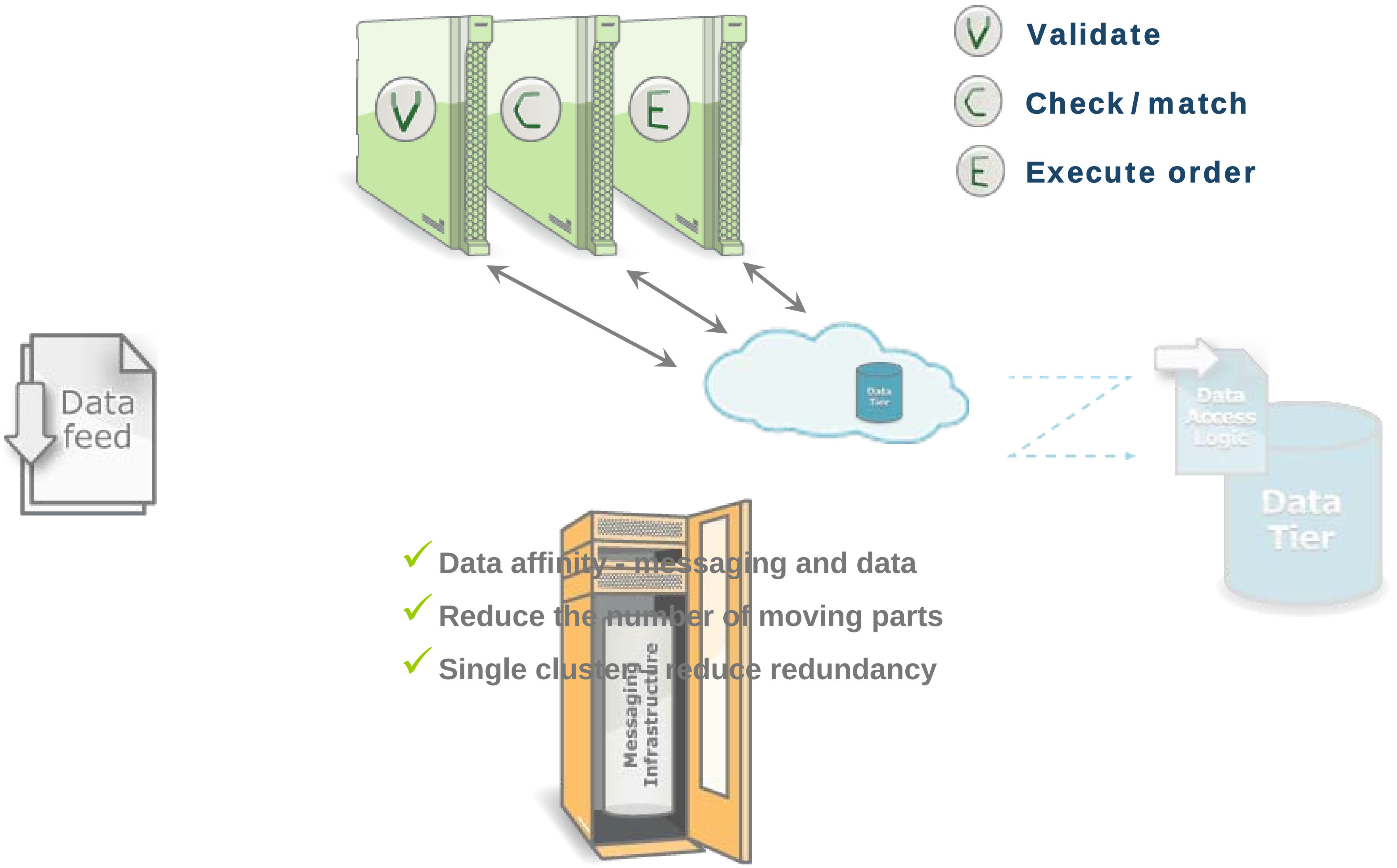
✓ Persistency As A Service

Persistency As a Service

- Moving the database to the backend
  - In-Memory Data Grid is used as the front-end data store
  - Synchronization with the database is done in the background
  - Reliable asynchronous replication is used to ensure no data-loss
  - Hibernate can be used to provide transparent mapping

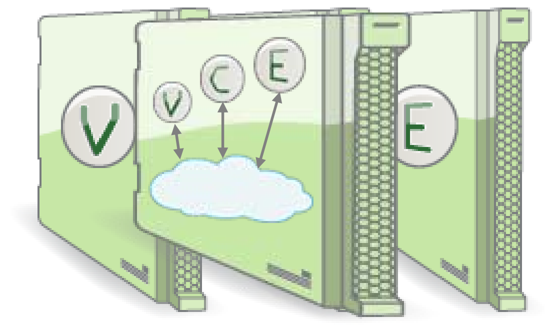


Step 2:
Consolidate the ESB and Data Together



- ✓ Data affinity - messaging and data
- ✓ Reduce the number of moving parts
- ✓ Single cluster – reduce redundancy

Step 3:
Assemble the Business Logic, Data, and Messaging



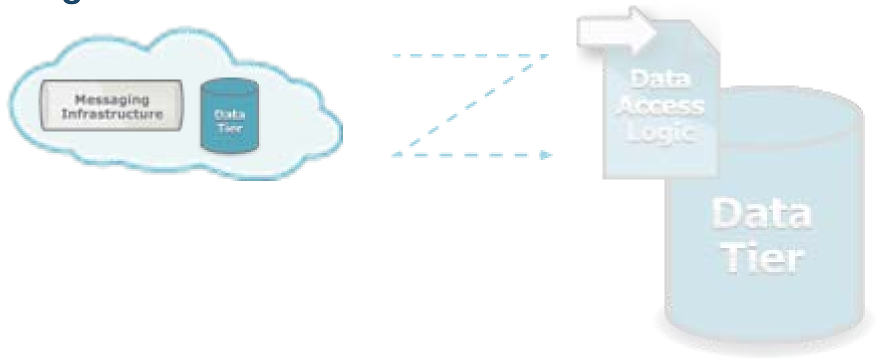
Ⓥ **Validate**

Ⓒ **Check / match**

Ⓔ **Execute order**

Processing Unit

Business tier

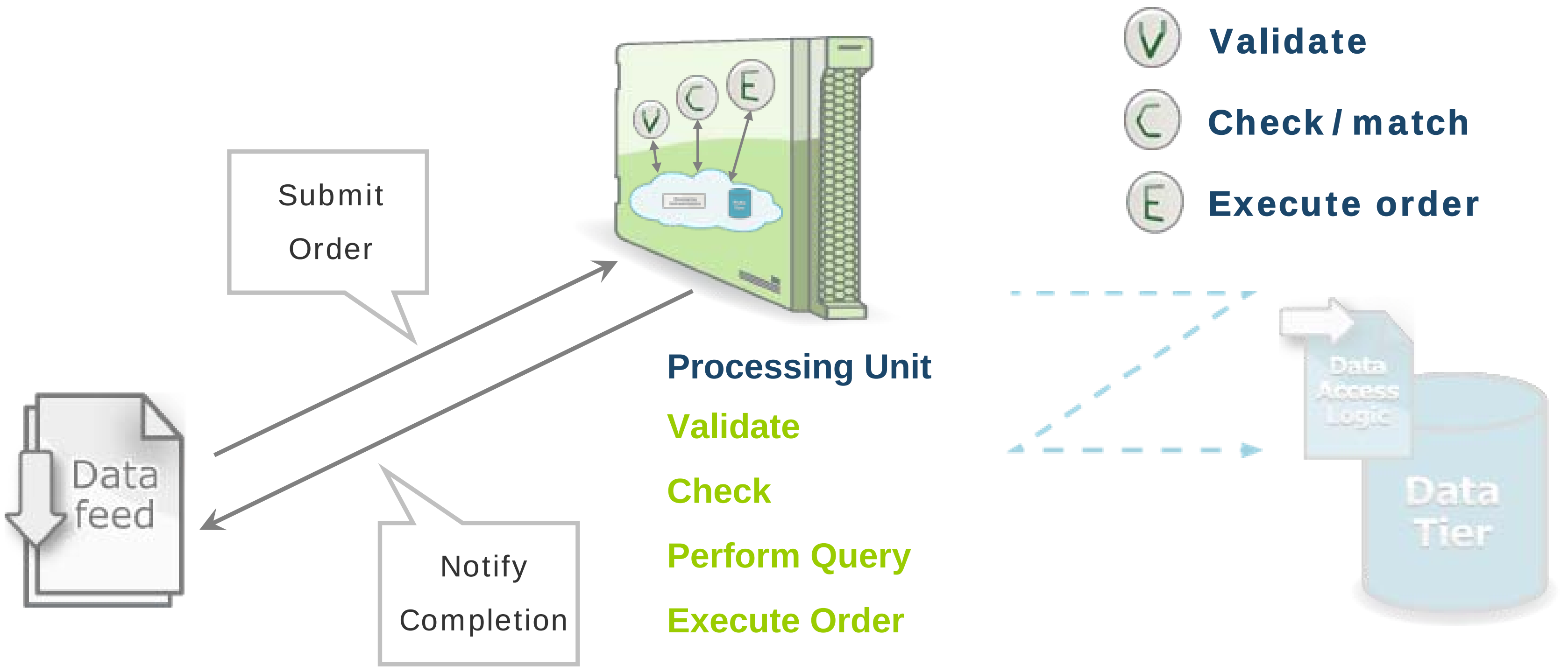


✓ **Single model for:**

- ✓ **Design**
- ✓ **Development**
- ✓ **Testing**
- ✓ **Implementation**
- ✓ **Deployment**
- ✓ **Management**

✓ **No integration effort**

Putting it all together..



✓ Collocation of all tiers enables transactions to occur in process with minimal network hops

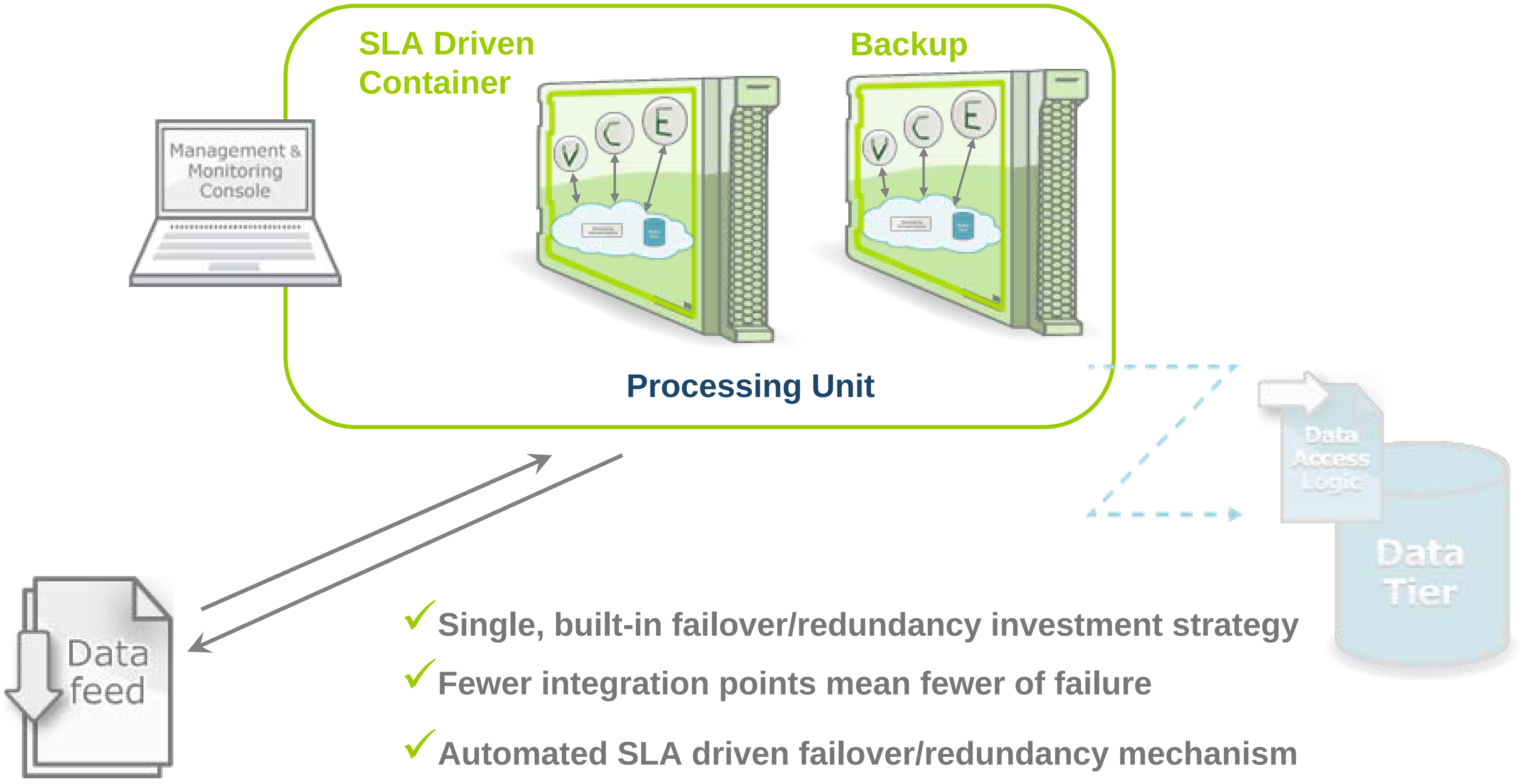
✓ Minimum latency and maximum throughput

✓ Unparalleled End-To-End Transaction Performance

Persist for Compliance & Reporting purposes:

- Storing State
- Register Orders
- etc.

SLA Driven Deployment



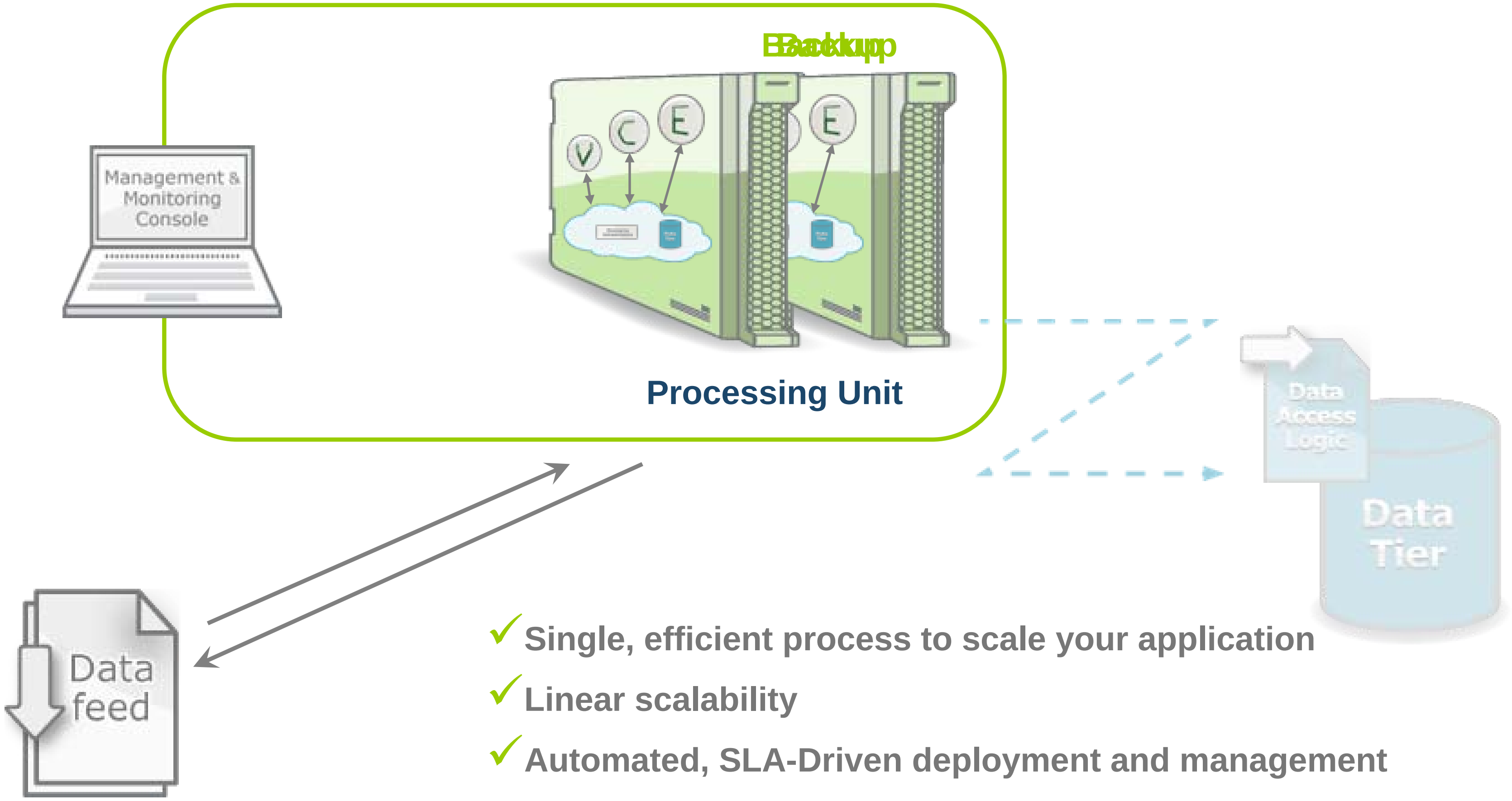
✓ Single, built-in failover/redundancy investment strategy

✓ Fewer integration points mean fewer of failure

✓ Automated SLA driven failover/redundancy mechanism

✓ Continuous Availability

Scaling …. made simple!



- ✓ Single, efficient process to scale your application
- ✓ Linear scalability
- ✓ Automated, SLA-Driven deployment and management
  - Scaling policy, System requirements, Space cluster topology

SBA - Space Based Architecture

- What is Space Based Architecture?
  - A holistic architecture for scaling out stateful applications
  - Provides details on how to combine the three steps in the most optimal manner
  - Can be implemented in various ways and products:
    - Using Combinations of products – Messaging, Distributed Caching and integrate them together.
    - Using single virtual implementation for all of the above:
      - This is currently supported by GigaSpaces
      - Google refers to a similar model called *“Cloud Computing”*
      - Other vendors seem to follow that direction: Amazon EC2, eBay, etc.

- **See Wikipedia for further details:**
  - **http://en.wikipedia.org/wiki/Space_based_architecture**

Transparent Transition to SBA using Spring

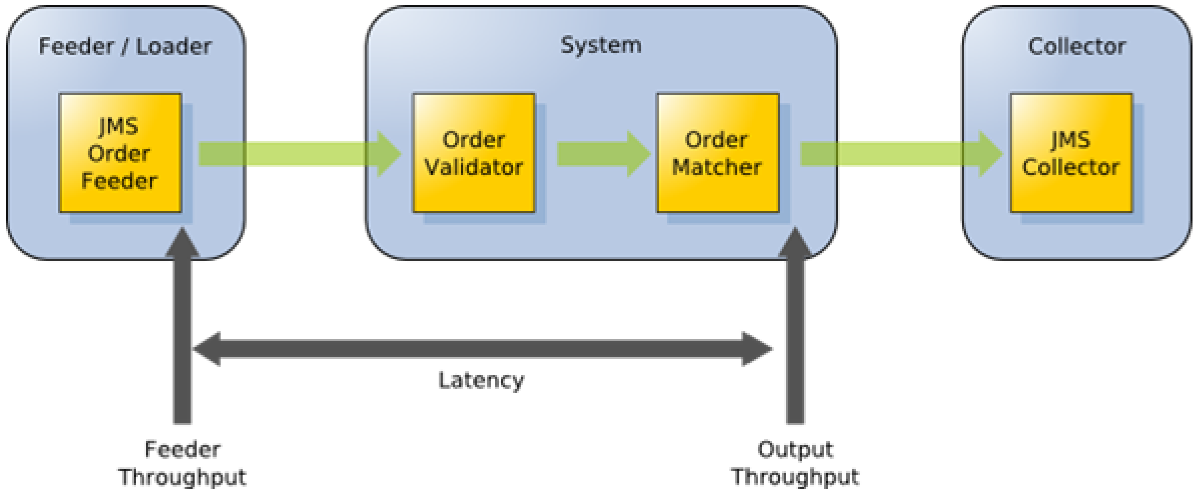
- Spring abstraction is a good starting point for separation between the applications code and the underlying runtime middleware through the use of abstractions:
  - Abstract the Data Tier
    - DAO
      - Abstraction from the underlying data implementation (database or another caching solution)
    - Declarative transaction
      - Abstract the transaction semantics from our code
  - Abstract the Messaging Tier
    - JMS Façade
    - Remoting
    - Event handlers
  - Abstract the deployment, configuration and packaging
    - Use of XML namespace enable simple extension of the existing configuration
    - OSGI provides packaging and deployment model tuned for high performance SOA

How seamless the transition to SBA can be?

- Applications written with the mentioned abstractions can easily migrate to the new model; those that don’t will require development effort.
- Not every application can be transformed to the new model
  - The majority of applications can handle step1-2
  - Step 3 relies on partitioning, which may require re-architecture/design.

Comparing SBA and TBA

Reference Application



Main Requirements:

-Hot failover – no data loss

-Full consistency

Measures:

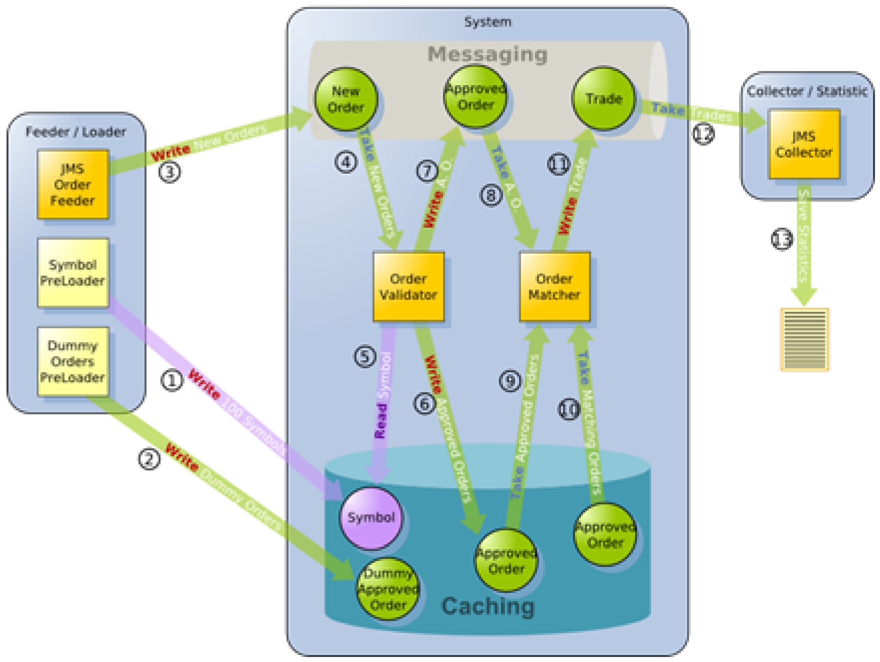
-Latency

-Scalability

Implementation

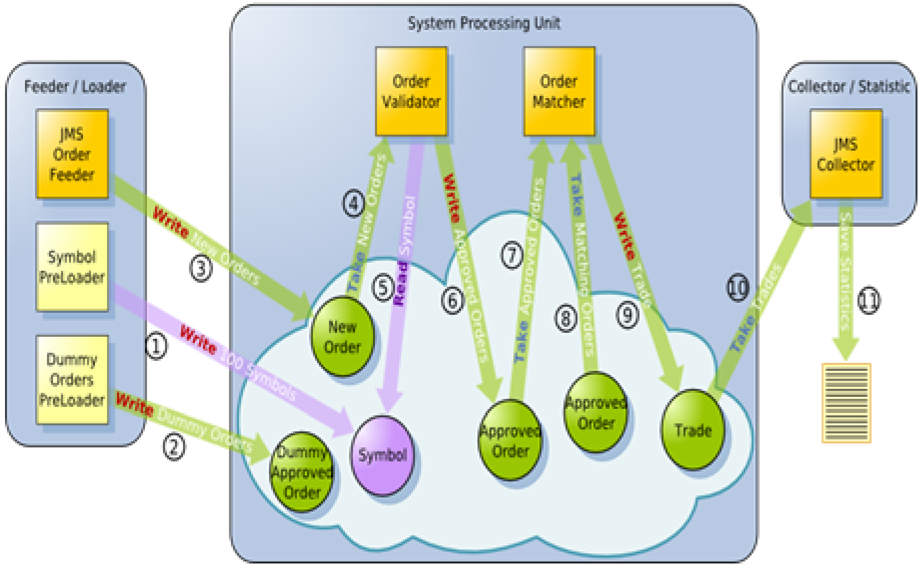
Tier Based Implementation

Space Based Architecture versus Tiers Based Architecture: TBA WorkFlow



Space Based Implementation

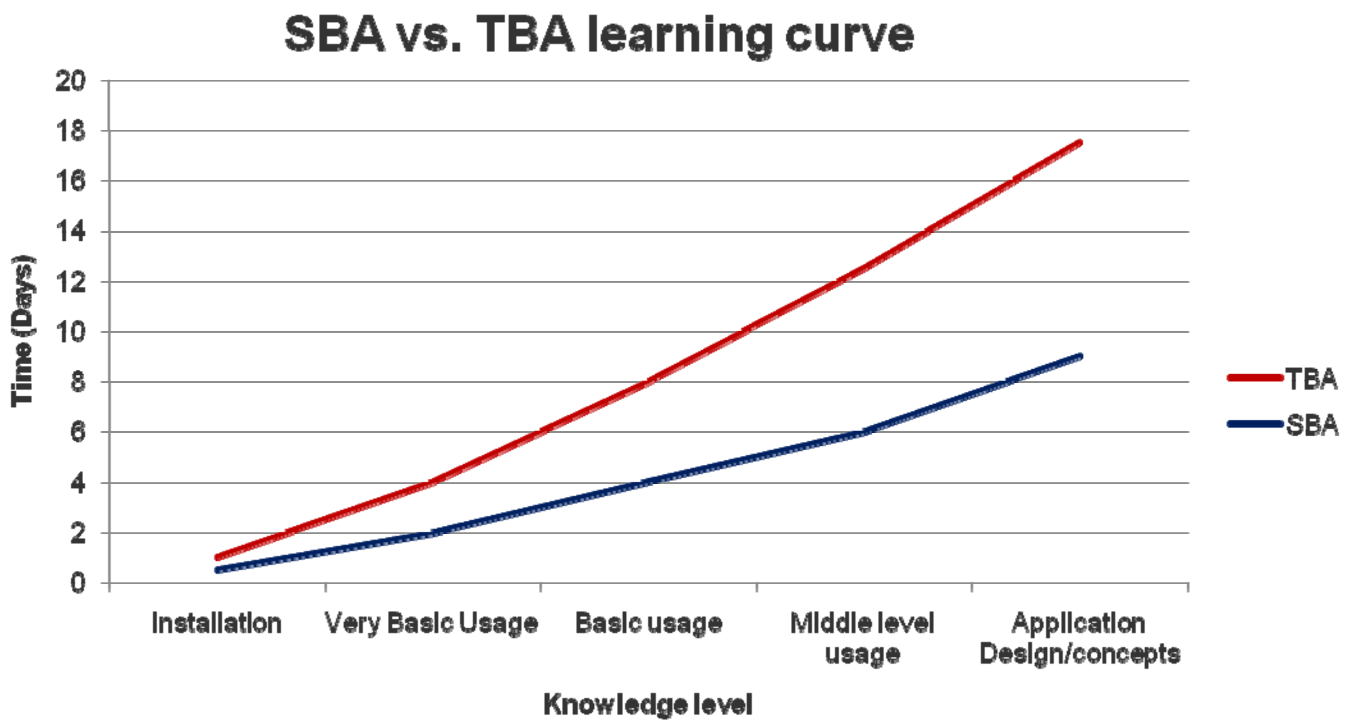
Space Based Architecture versus Tiers Based Architecture: SBA WorkFlow



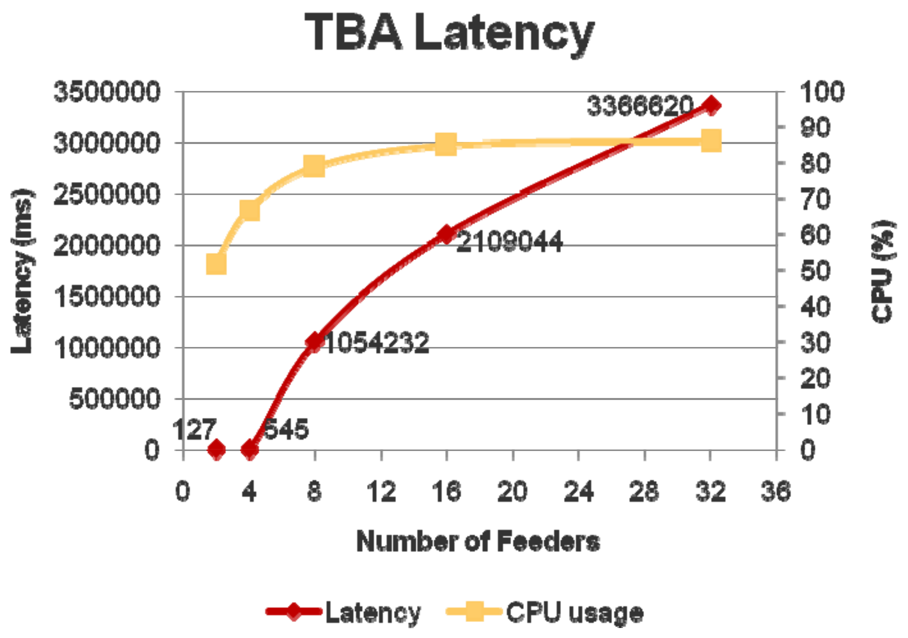
SBA vs. TBA: Context

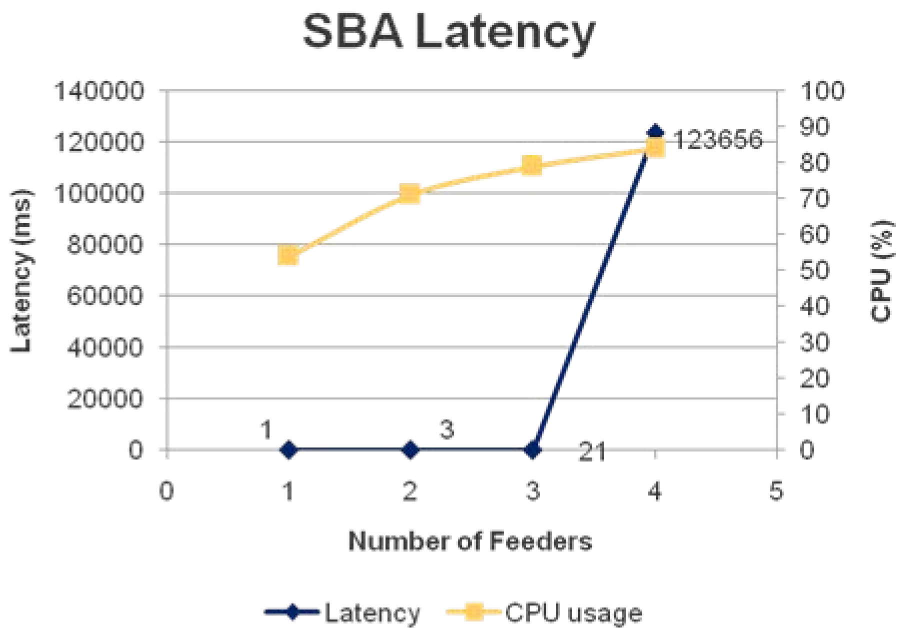
- Development approach
  - 2 teams; SBA & TBA
  - Native approach for each TBA product
    - Leading application server and a caching vendor
  - TBA team had more than one product expert

Learning curve

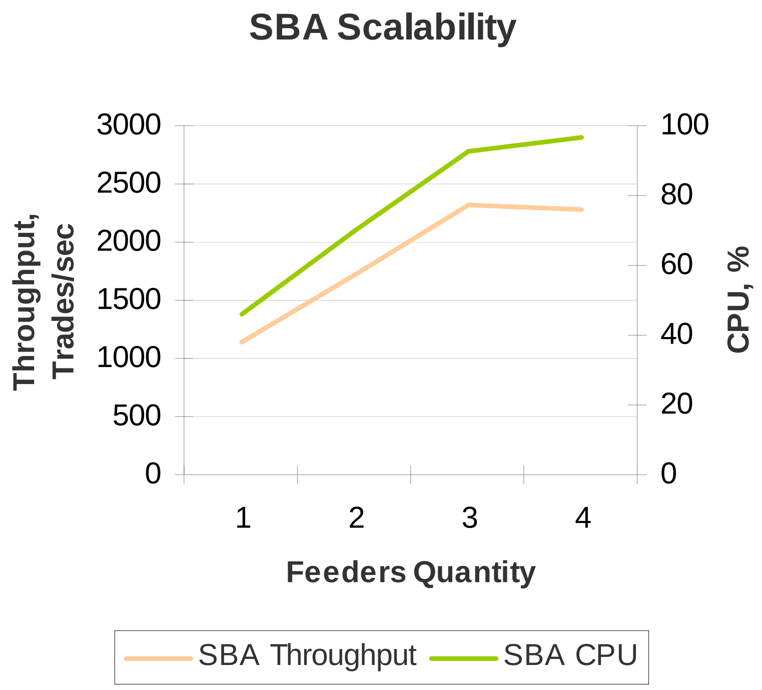


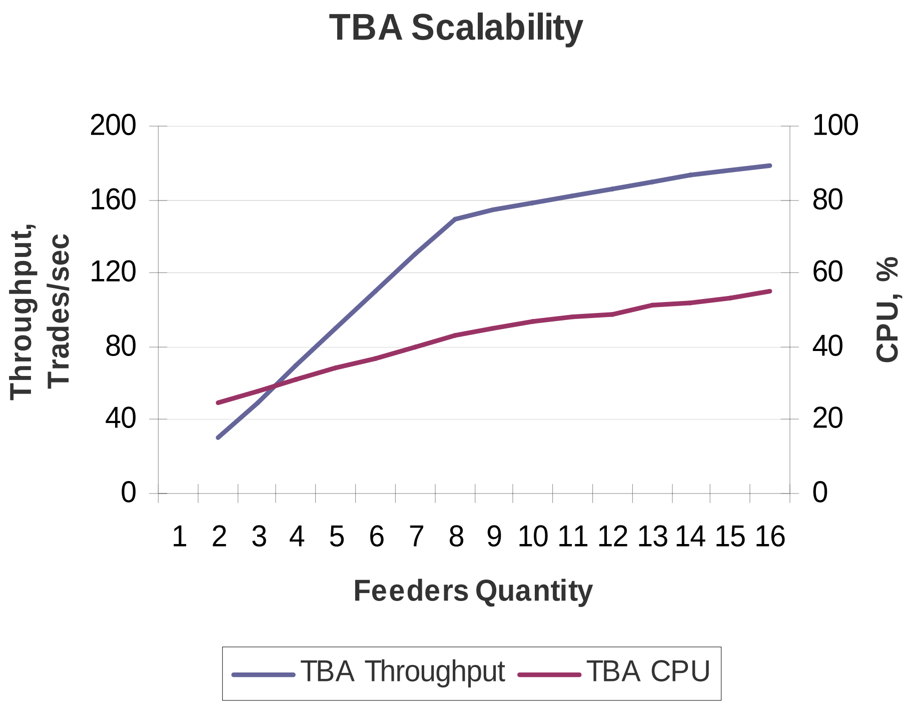
Latency measurement





Results - Feeding scalability





TBA Results Analysis

- Queues persistency
  - High availability is required for the messaging tiers
  - Test without persistency enabled is **4 times faster**
  - Requires specific HW for ensuring no data-loss.
- Distributed transactions
  - Required to ensure no message-loss between the tiers
  - Tests without transactions is **4** and **5 times faster**.
- Additional network calls due to lack of consistent data affinity
  - As the workflow and the cache layer are in separate tiers, network calls occur in each step in the workflow.

- **Conclusion**
  - Caching can only **improve** performance and scalability but doesn't enable linear scalability

Summary: Benefits of SBA vs. TBA

- Performance
  - Eliminate/reduce network hops per business transaction
  - Based on in-memory approach
- Scalability
  - True End to End linear scalability
- Resilience
  - Fewer points of failure (less moving parts)
  - Designed for hot fail-over
- Complexity
  - Enable agile development (no need to change the code or configuration when moving from a standalone development to a cluster environment).
- TCO
  - Hardware purchases
  - Eliminate efforts required to integrate tiers
  - Single, built-in failover/redundancy investment and strategy
  - Single monitoring and management strategy
  - Automated, SLA-Driven deployment and management
  - Shorter and more efficient development process