WRITE ONCE.
SCALE ANYWHERE.

Scaling out in three steps or From TBA to SBA

Guy Nirpaz

EVP R&D, GigaSpaces

guyn@gigaspaces.com

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

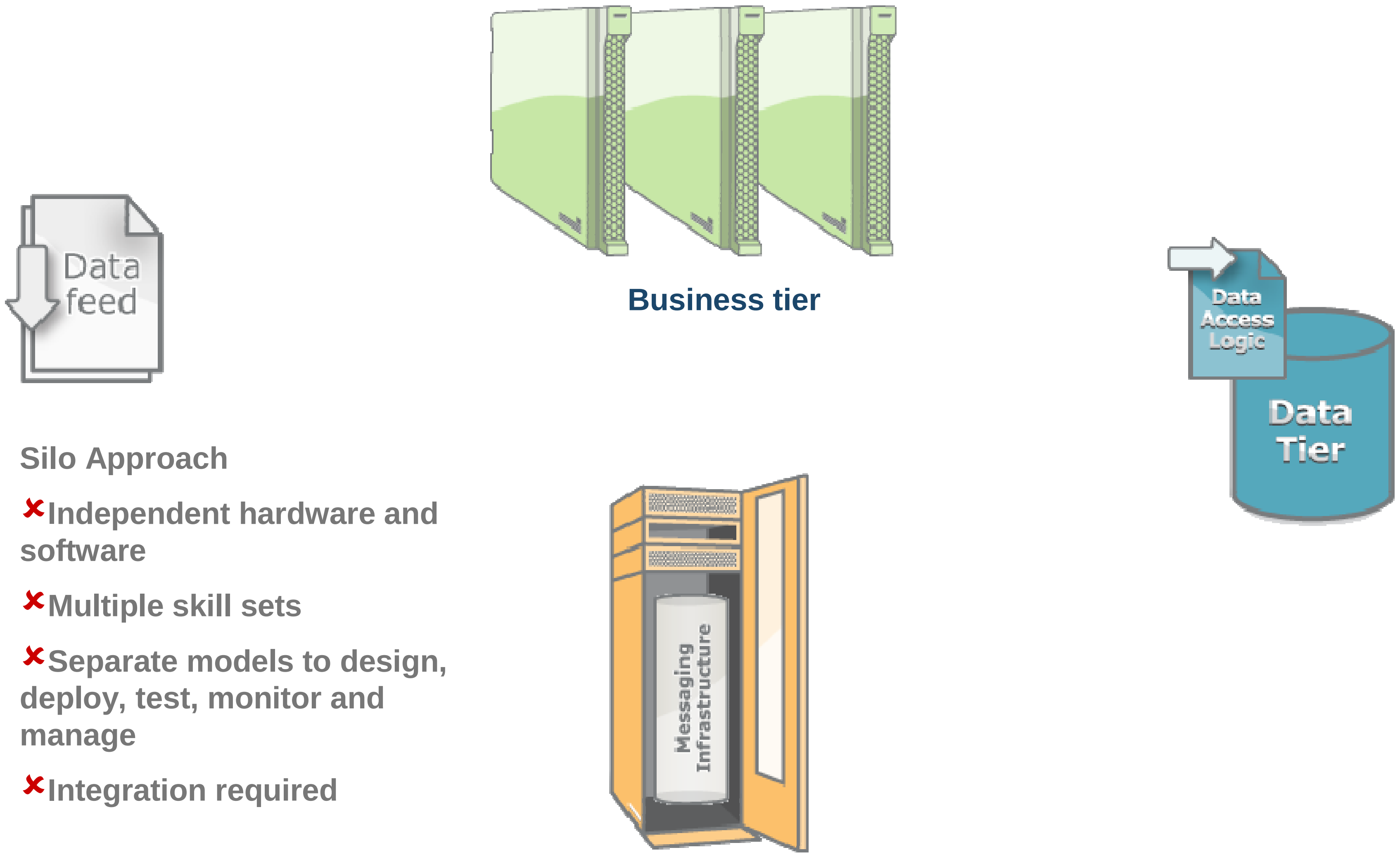
On-Line:

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

Defense

Real-time intelligence, Pattern Analysis

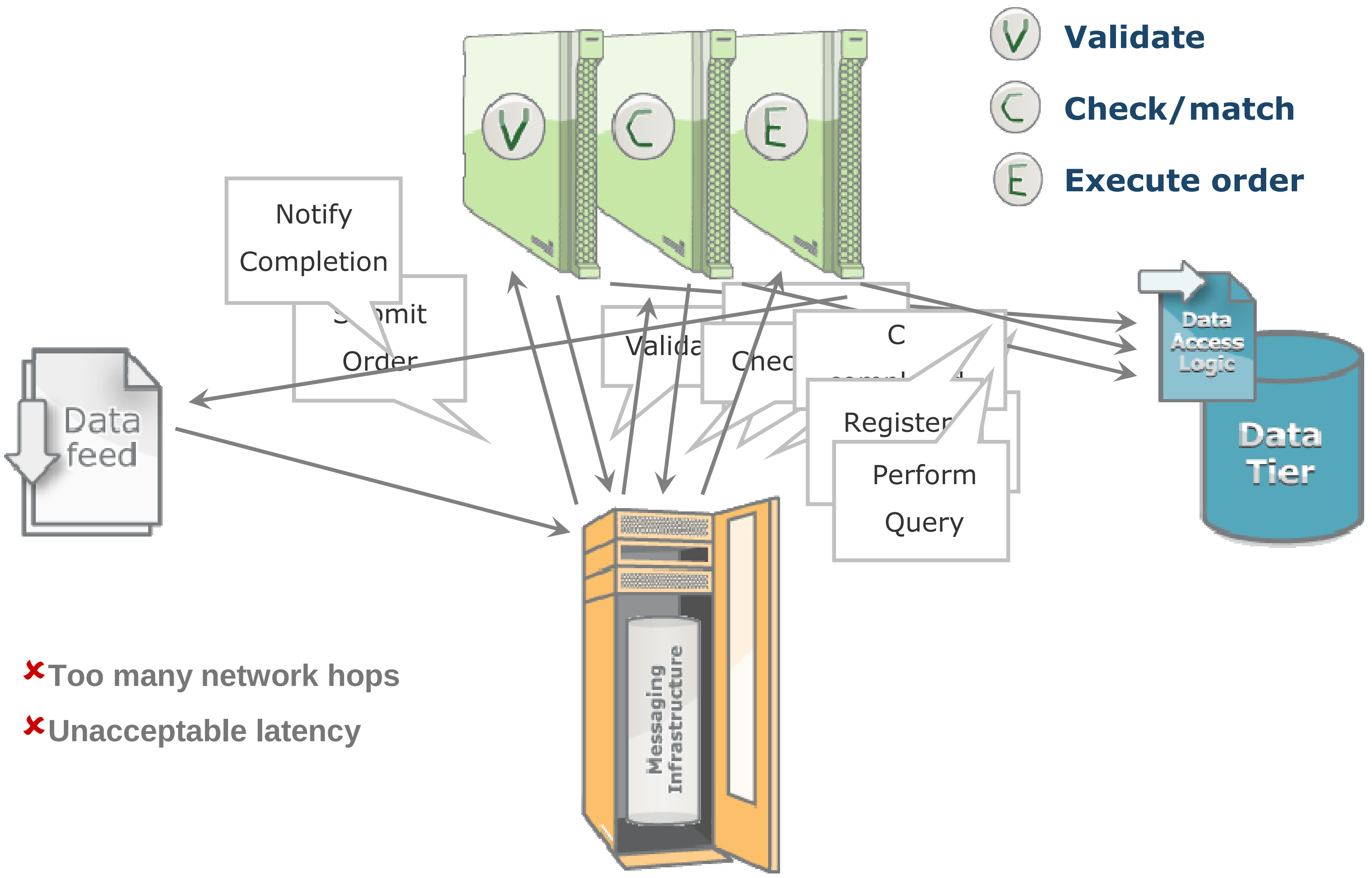
Traditional Tier-Based Architecture



Silo Approach

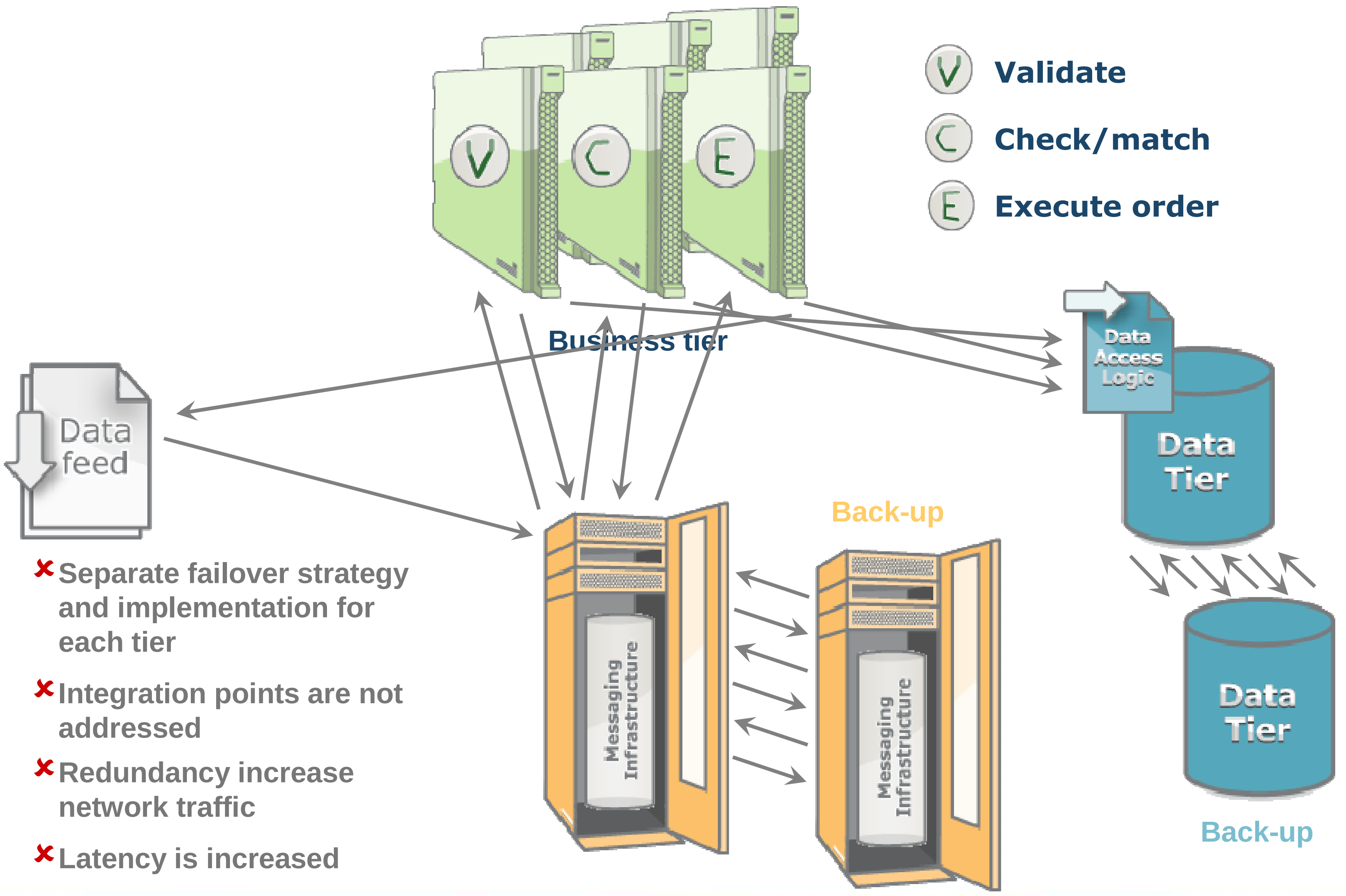
- ✗Independent hardware and software
- ✗Multiple skill sets
- ✗Separate models to design, deploy, test, monitor and manage
- ✗Integration required

A Transaction Flow Example - Order Management



- ✘Too many network hops
- ✘Unacceptable latency

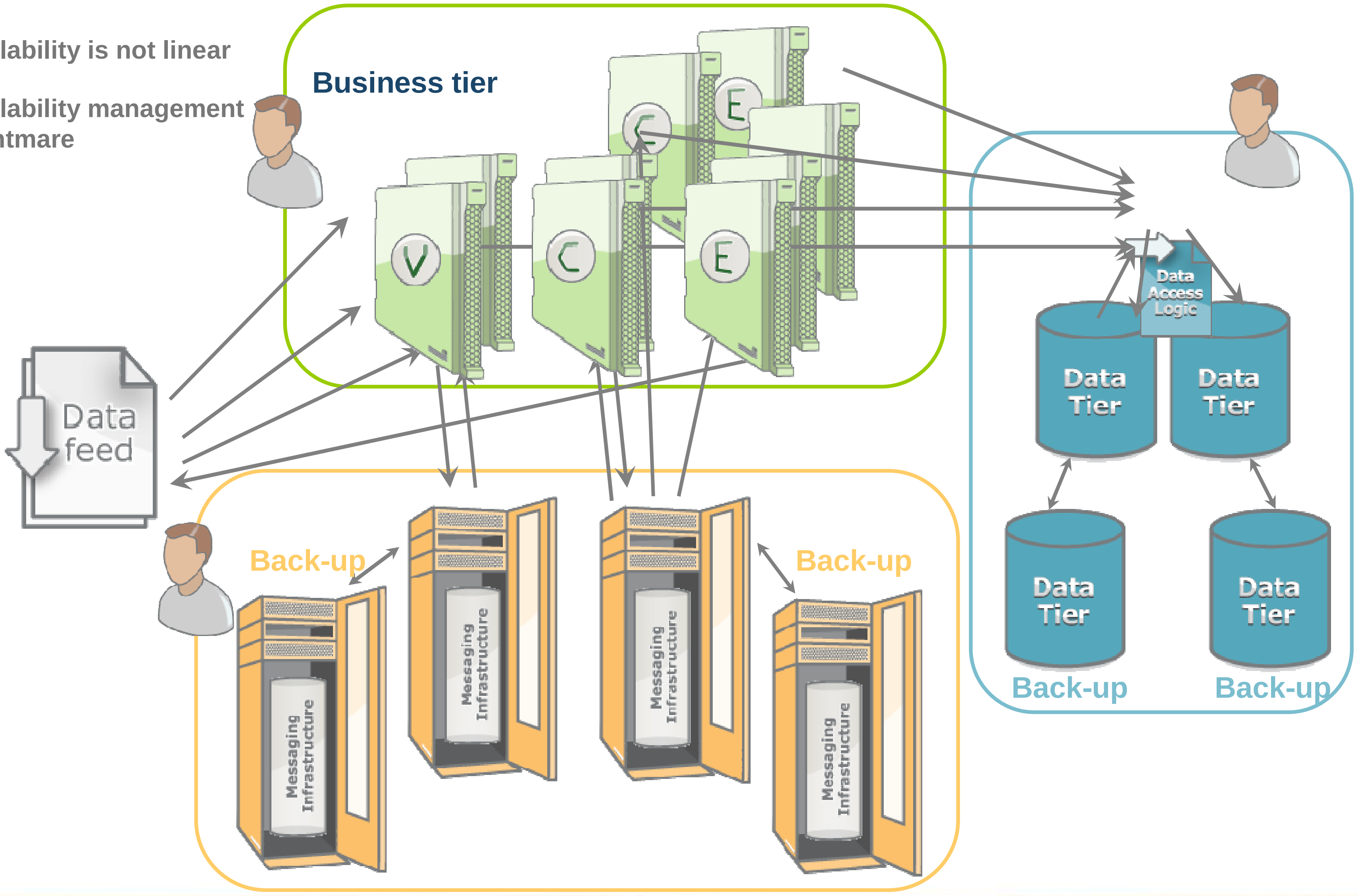
Maintaining Resiliency in a Traditional Tiered Application



- ✗ 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

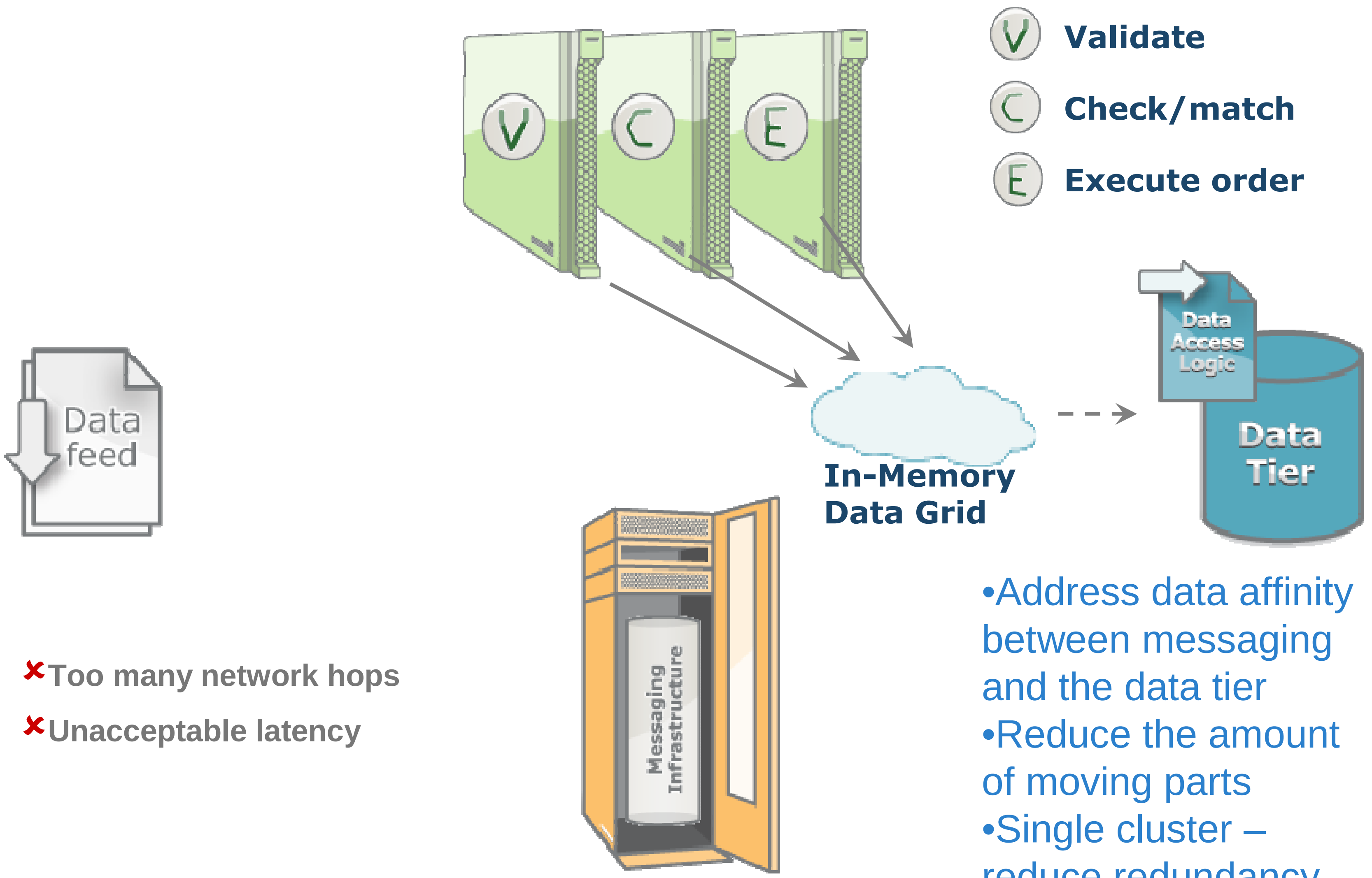
- ✗ Scalability is not linear
- ✗ Scalability management nightmare



Three steps to (high performance) SOA

- Reduce I/O Bottleneck using In-Memory-Data Grid
  - Reduce I/O bottleneck
  - Improve the scaling on each individual unit
  - Persistency As A Service – move the persistency a step behind
- Consolidate the ESB and Data together
  - Address data affinity between messaging and the data tier
  - Reduce the amount of moving parts
  - Single cluster – reduce redundancy
- Assemble the business logic togather with the data and messaging
  - Create a single unit of scale and fail-over
  - Reduce the latency
  - Simplify the scaling and deployment

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

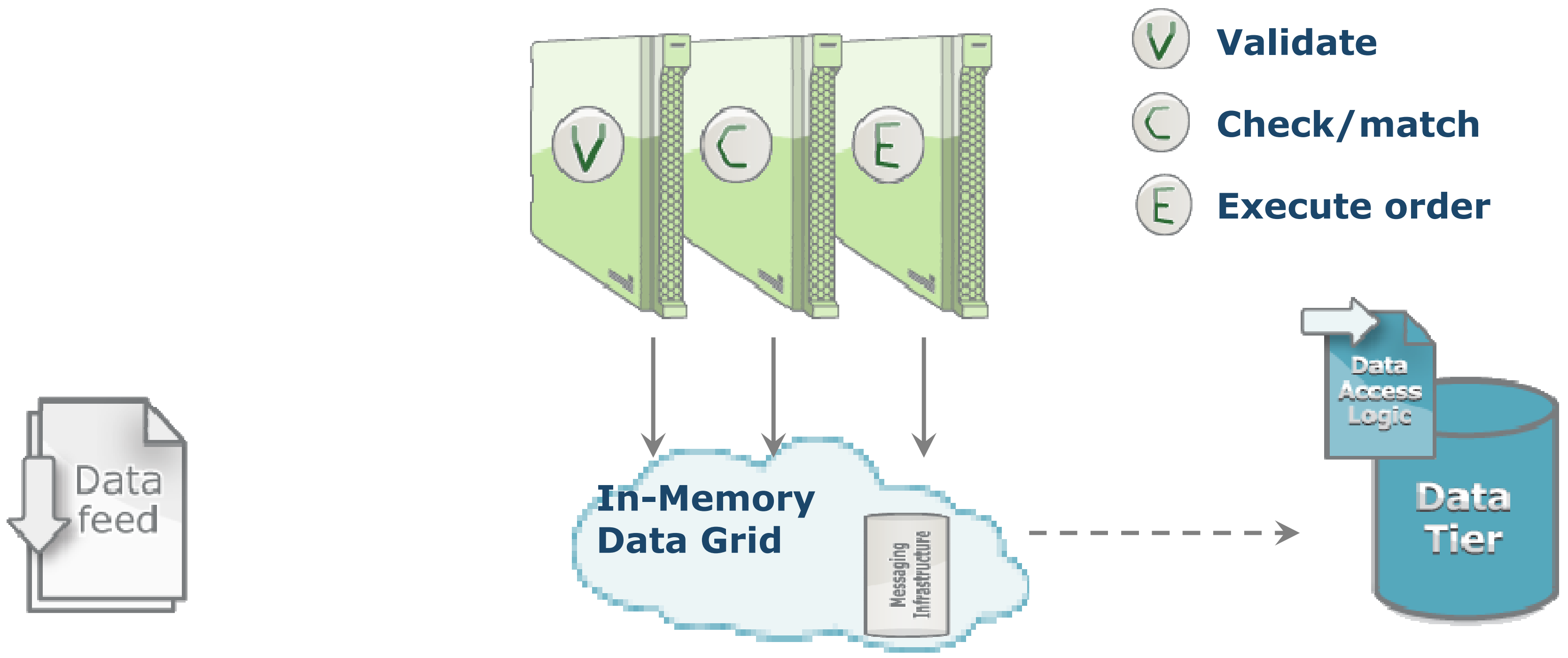


✘Too many network hops

✘Unacceptable latency

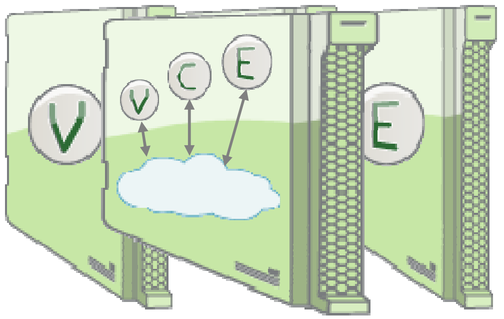
- Address data affinity between messaging and the data tier
- Reduce the amount of moving parts
- Single cluster – reduce redundancy

Step 2: Consolidate the ESB and Data together



- Address data affinity between messaging and the data tier
- Reduce the amount of moving parts
- Single cluster – reduce redundancy

Step 3: Assemble the business logic together with the data and messaging



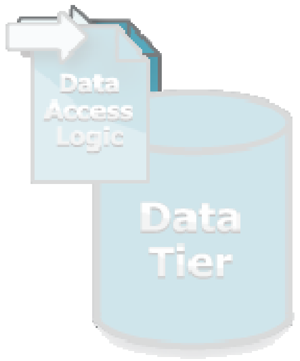
Processing Unit

Business tier

V **Validate**

C **Check/match**

E **Execute order**

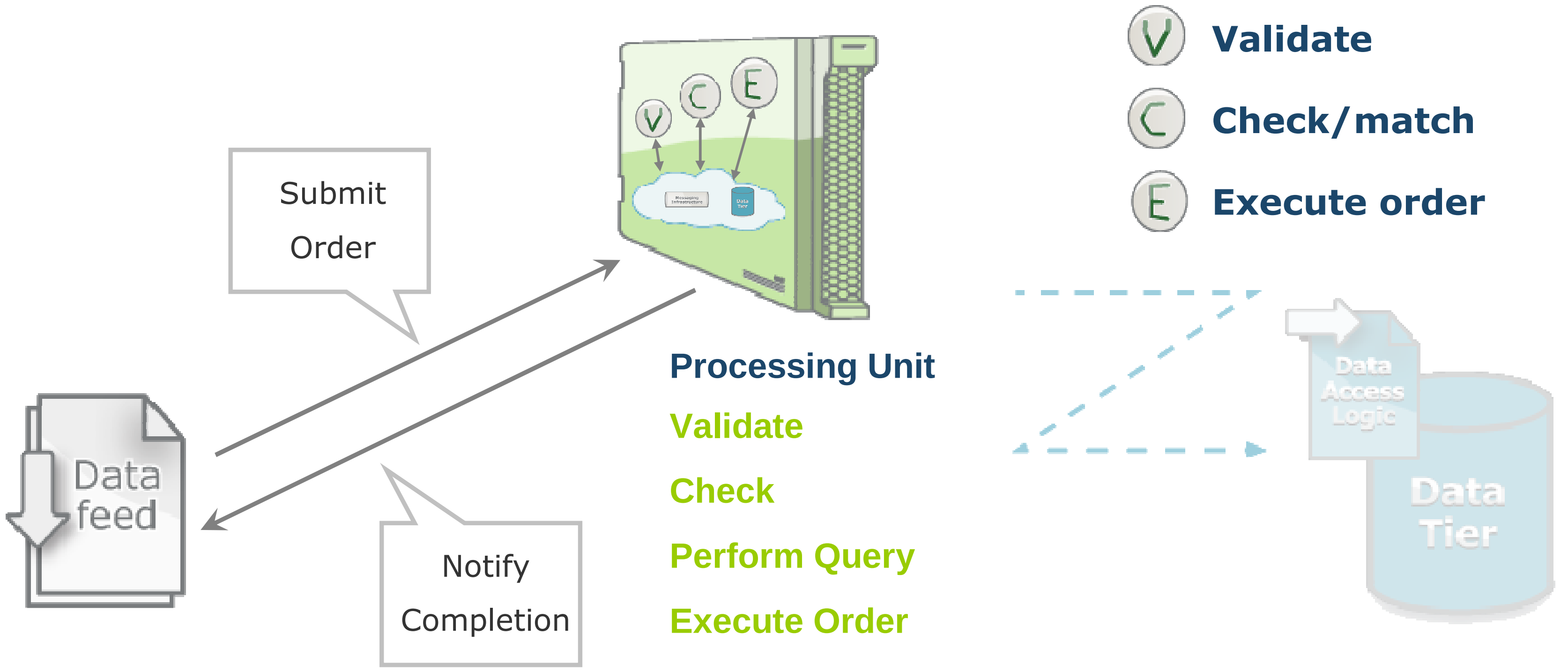


✓ Single model for:

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

✓ No integration effort

Putting it all together..



Processing Unit

Validate

Check

Perform Query

Execute Order

✓Memory based for maximum performance

✓Collocation of data, messaging and services enable transactions to occur in process with minimal network hops

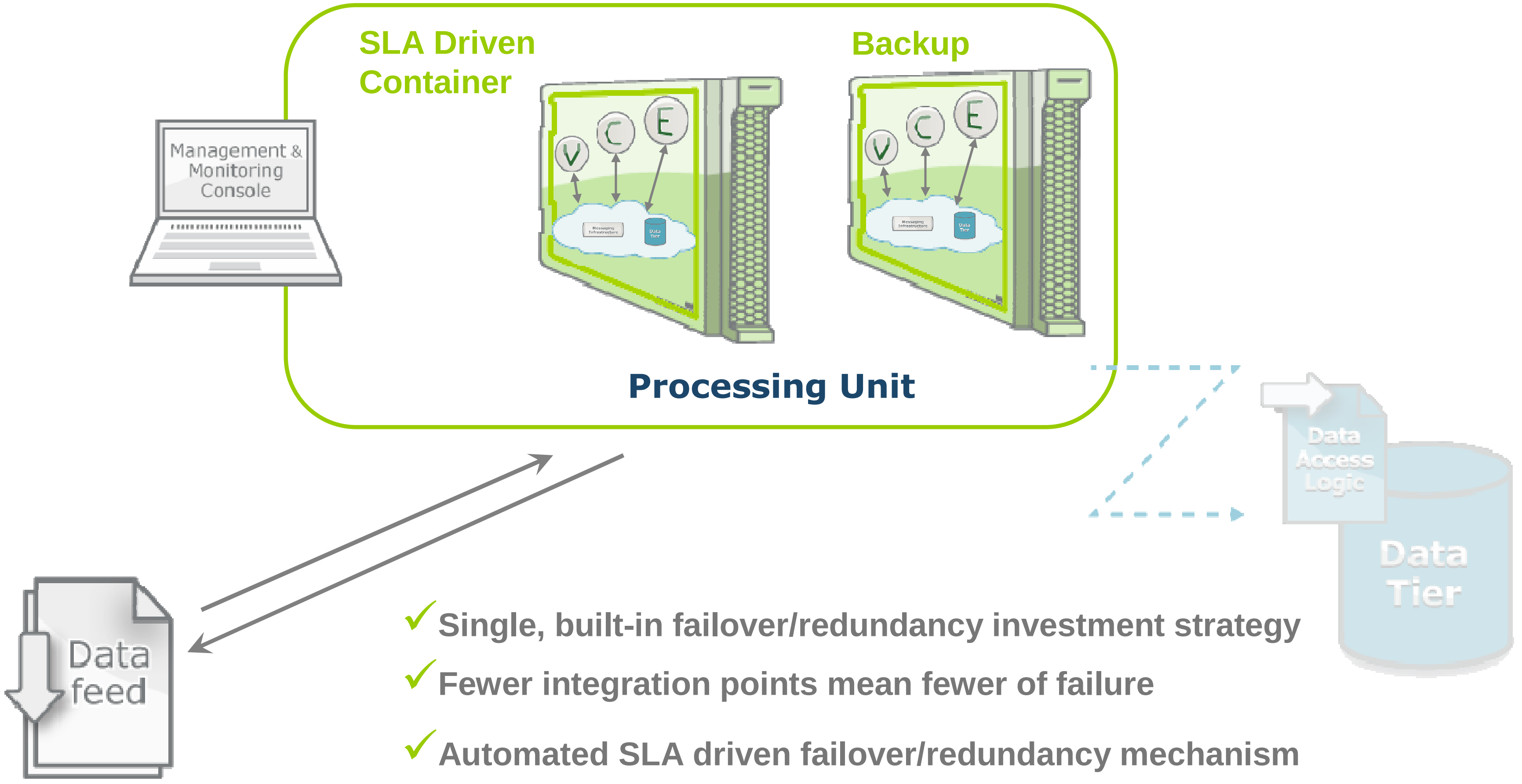
✓Minimum latency and maximum throughput

✓Unparalleled End-To-End Transaction Performance

Persist for Compliance & Reporting purposes:

- Storing State
- Register Orders
- etc.

SLA Driven Deployment



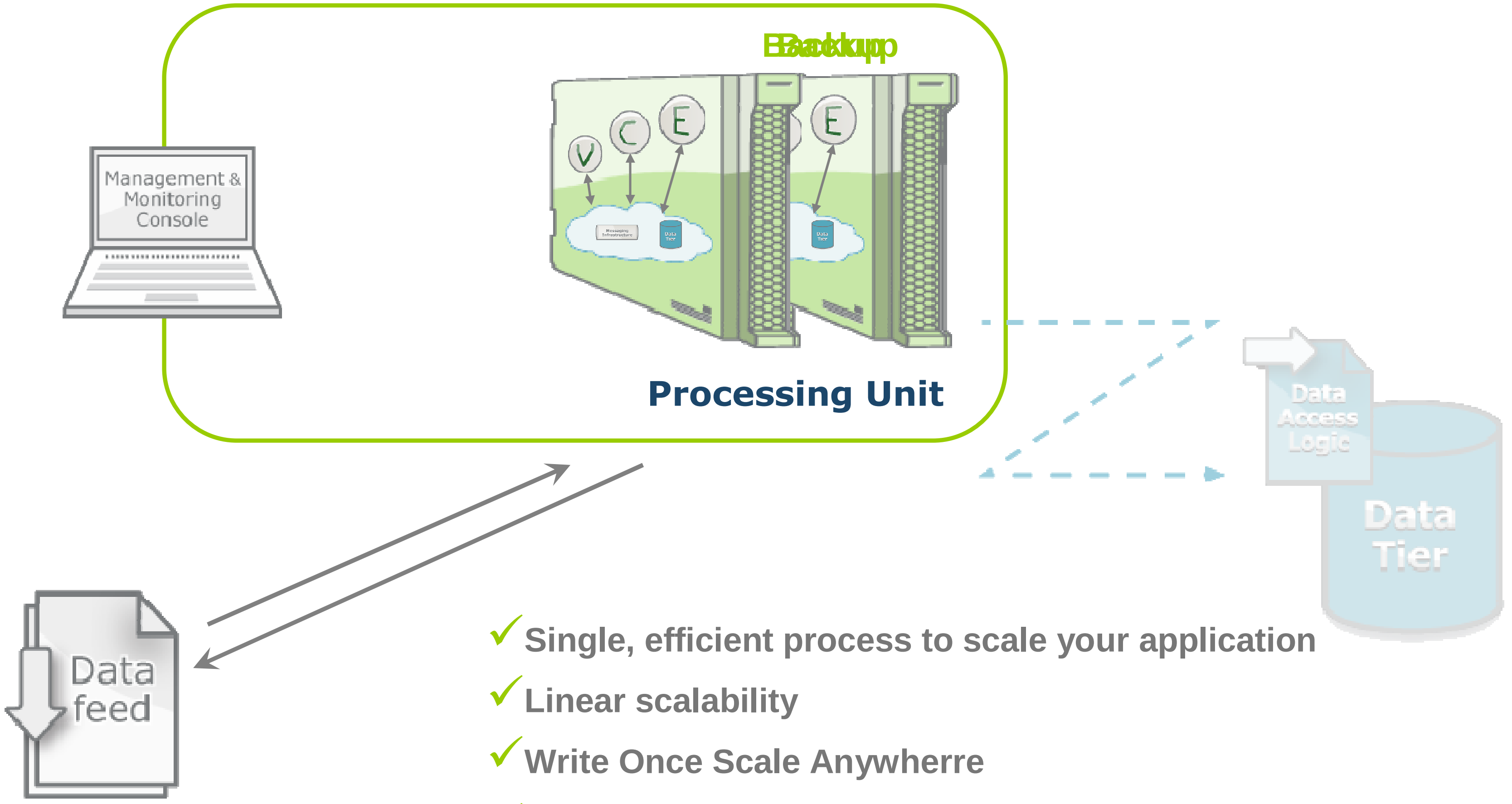
✓Single, built-in failover/redundancy investment strategy

✓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
- ✓Write Once Scale Anywherre
- ✓Automated, SLA-driven engine for deployment, monitoring and management engine

SBA - Space Based Architecture

- What Space Based Architecture?
  - 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
      - Other vendors seem to follow that direction

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

Making the transition transparent through:

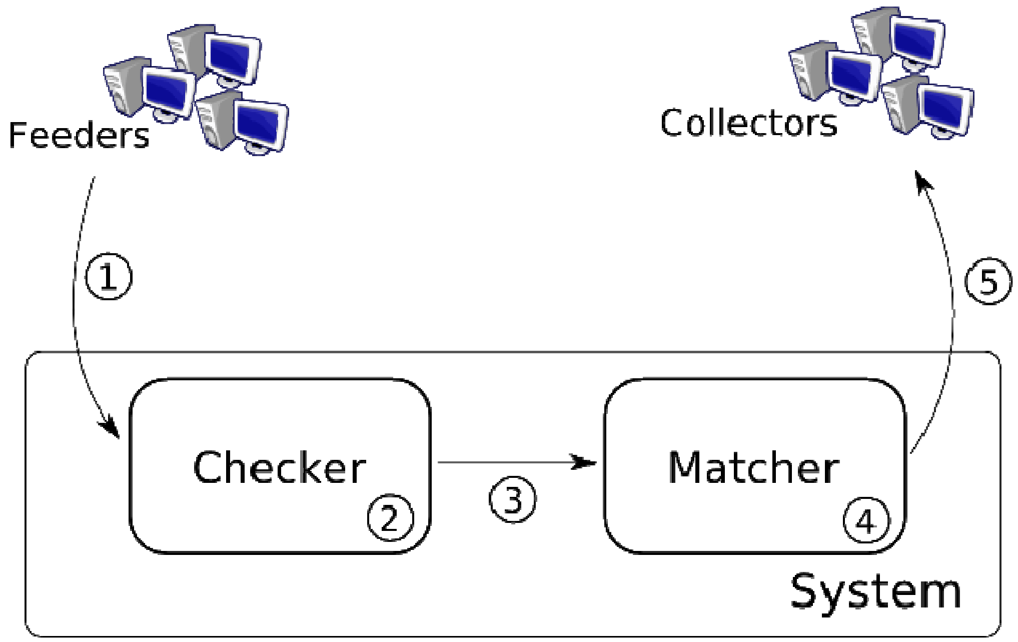
- Spring abstraction
  - Spring provide a good starting point for separation between our implementation and the underlying runtime middleware through the use of abstractions:
  - Abstract the Data Tier
    - DAO
      - Abstraction from the underlying data implementation (data base or other caching solution).
    - 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

Making the transition transparent through (Cont)

- How seamless can it be?
  - Not every application can be transformed to the new model
  - Majority can handle step1-2,
  - Step 3 relies on partitioning which may require re-architecture/design.
  - Application written with the mentioned abstractions can easily migrate to the new model, those that don't will require development effort.

Comparing SBA and TBA

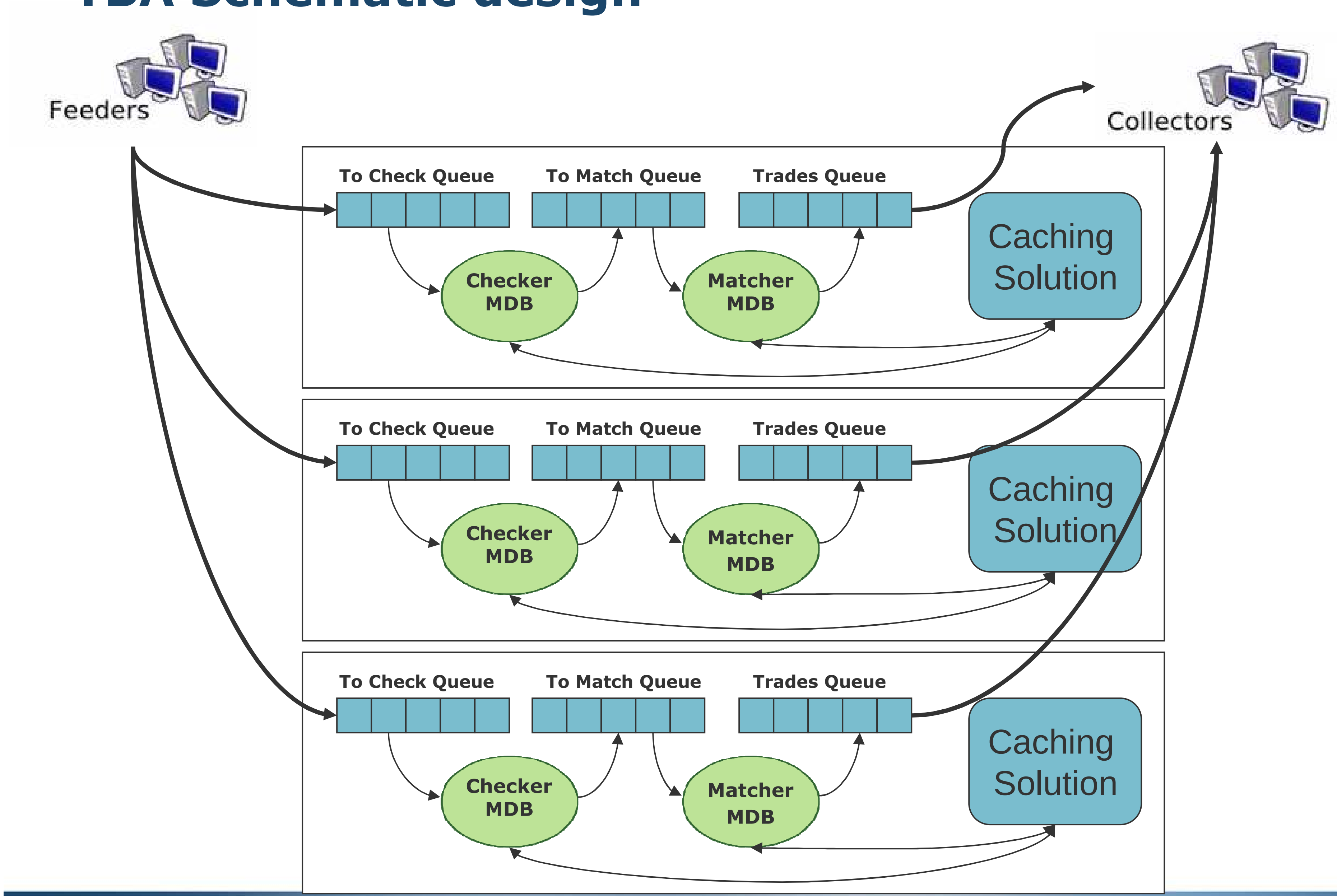
- **Guidelines**
  - **Clients use JMS**
  - **System is highly available**
  - **Transactions are measured end-to-end**



SBA vs TBA: Context

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

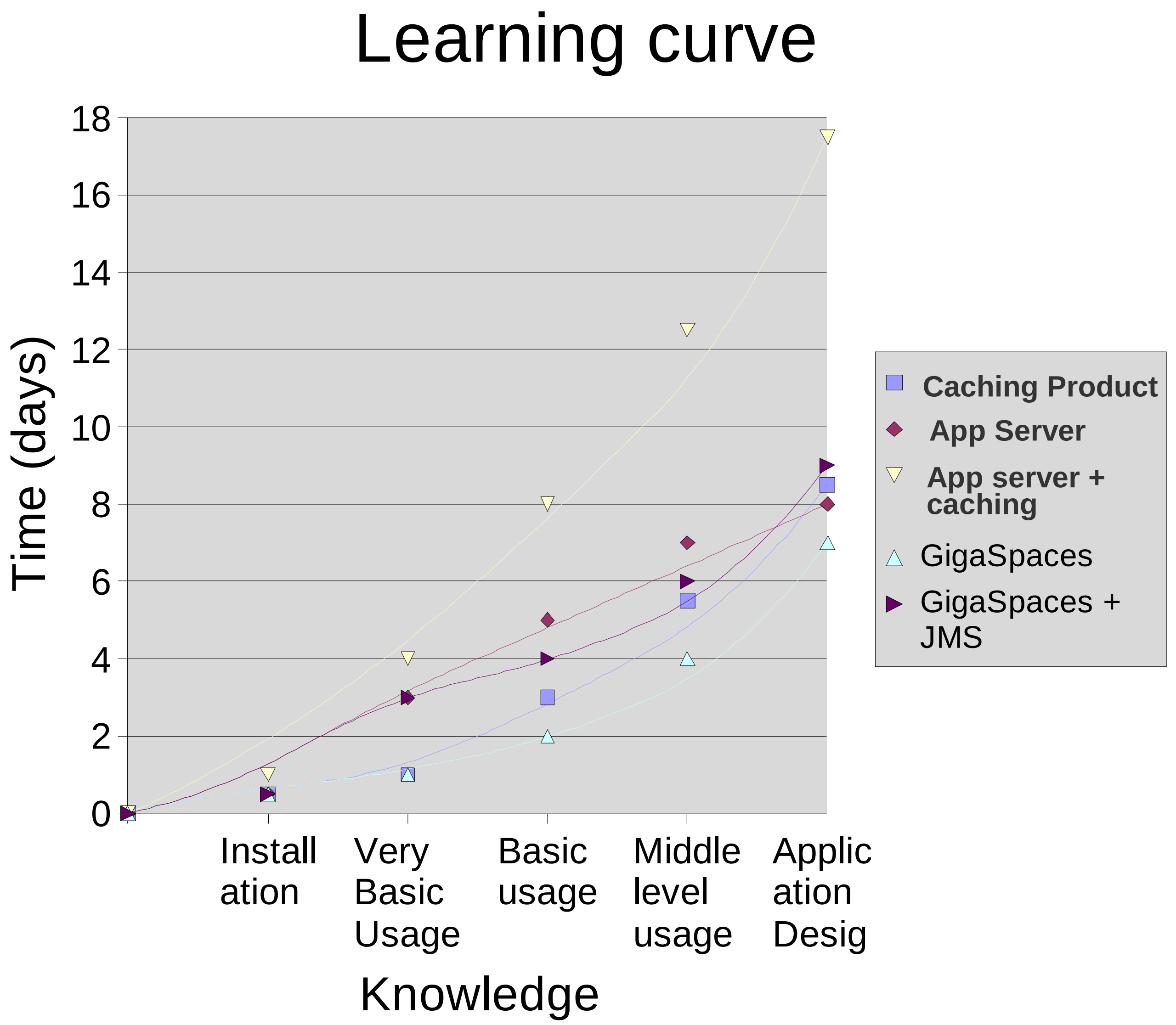
TBA Schematic design



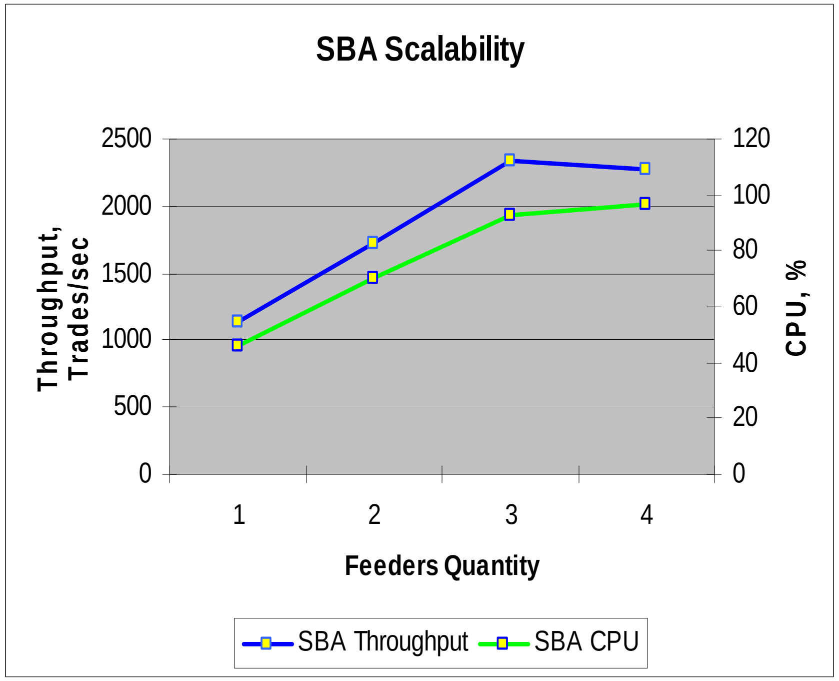
TBA Schematic design

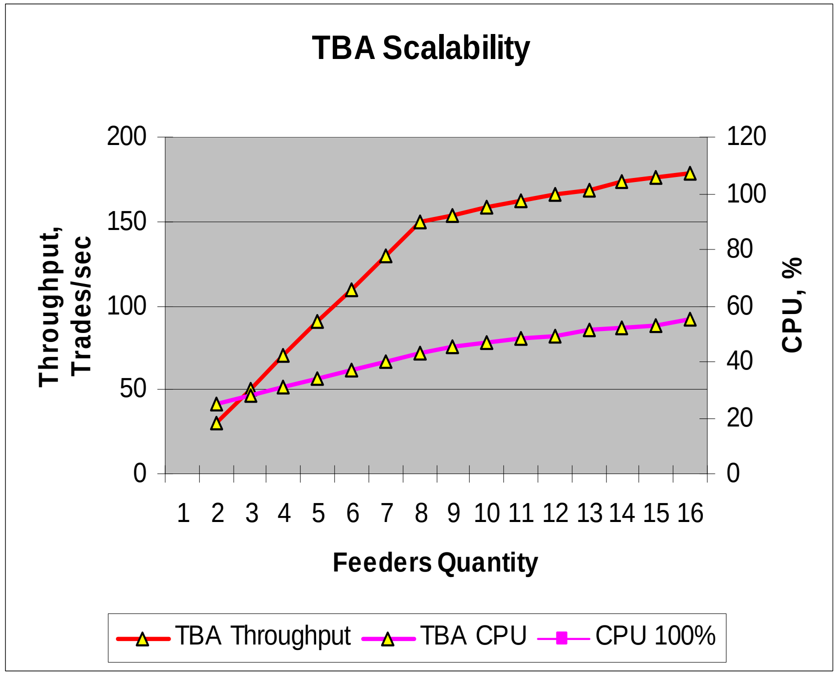
- This architecture requires 2 licenses per machine : 1 for app server and 1 for caching product
- The business logic and data is not collocated in the same process (due to affinity complexity)
- Persistent Queues (introducing I/O bottleneck) are the only way to handle high availability in the leading application server JMS implementation

SBA vs TBA


Learning curve
18
16
14
12
10
8
6
4
2
0
Time (days)
Install ation
Very Basic Usage
Basic usage
Middle level usage
Applic ation Desig
Knowledge
Caching Product
App Server
App server + caching
GigaSpaces
GigaSpaces + JMS

SBA vs TBA: Results: Feeding scalability





Benefits of SBA vs. a Tier-Based Architecture

- Performance
  - Eliminate/reduce network hops per business transaction
- Scalability
  - Enable application growth through a single, consistent…
- Resilience
  - Fewer points of failure
  - Inherent replication eliminates the need to failover
- TCO
  - A single software purchase
  - 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

Questions, please?

If you would like to join us …

jobs@gigaspaces.com