
# Top 10 Patterns for Scaling Out Java™ Applications

Cameron Purdy, Vice President, Oracle Corp.

TS-6339

ORACLE

Java

Sun microsystems

Learn about the most common patterns for scaling-out Java applications, and how to ensure that those patterns work for you.

GOAL

"But if I ran IT,"
Said young Gerald McGrew,
"I'd make a few upgrades.
That's just what I'd do . . ."

Those heavy app-servers and that kind of stuff
They have in use now are not *quite* good enough.
You use things like these for just any old site.
They're awfully old-fashioned. I want something *lite!*

Text adapted from "If I Ran the Zoo" by Dr. Seuss

# Introduction

*"I believe in an open mind, but not so open that your brains fall out."*

*-Arthur Hays Sulzberger*

# About the Speaker

- Cameron Purdy is the Vice President of Development for Oracle Fusion Middleware, responsible for the Coherence Data Grid product
- Cameron was a founder and the CEO of Tangosol, acquired by Oracle in 2007
- Cameron has been working in Java since 1996, Java EE since 1999, and PowerPoint since 2004

ORACLE

# Disclaimer

Religious application of principles learned from a presentation attended at an industry tradeshow is not a substitute for common sense.

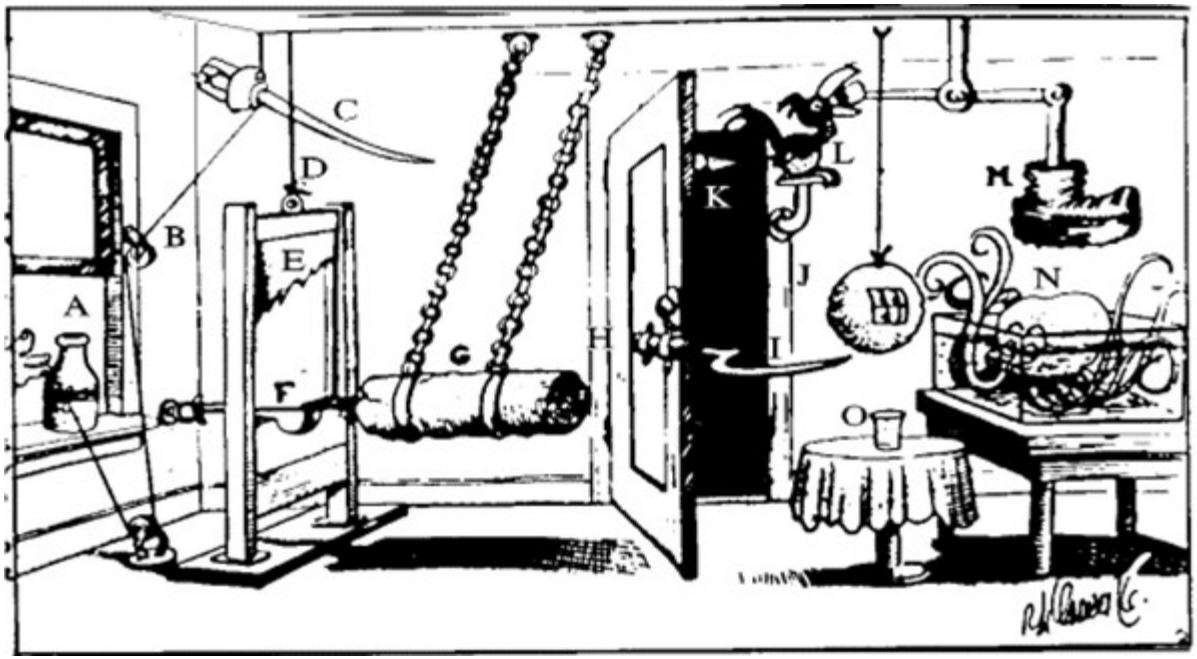
# 10. Understand the Problem

*"The hardest thing to understand in the world is the income tax."*

*-Albert Einstein*

# Initial Expectations are Crucial

- Building for Scalability cannot speed up an application
  - Building in Scalability always increases complexity
  - Complexity always increases path length
  - A scalable system will always be slower in single-user mode



# Trust me, the problem is scalability
*(Since the audience is self-selecting.)*

- Performance or Scalability?
  - If the system is fast in single-user mode, but slow under heavy load, then the problem is scalability, **not** performance
  - Queue theory almost always explains bad performance under load
  - Building a scalable system does not absolve you of the responsibility to build an efficient system

# Common Sense

- Horizontal scalability is the only answer
  - If there's too much work for one server to do, there's only one possible answer: You need more than one server!
  - Network: This is a distributed system by definition
  - Availability and Reliability must be baked in

# 9. Define the Requirements

*"A stupid man's report of what a clever man says is never accurate because he unconsciously translates what he hears into something he can understand."*

*-Bertrand Russell*

# The ARSPMS Requirements Model

- Availability
- Reliability
- Scalability
- Performance
- Manageability
- Serviceability


Acronym is from the PIOOMA Guide to Made Up Acronyms, 2008 Edition

# Goal

- Understand the theoretical achievable performance
  - Defined under single user load scenario
- Assumption: Availability, Reliability and Scalability are always required
- Understand costs of achieving -ilities
- Work toward ensuring linear systemic scalability

# Gotcha: Predictable Latency

- Distribution introduces more moving pieces, each of which degrades predictability of latency
  - Garbage Collection
  - Thread switches
  - Network I/O
  - Switches and routers

# Gotcha: Read Consistency

- Second law of distributed systems: The only things you can distribute are state (information) and behavior
- Two fundamental challenges:
  - Distributing state almost certainly creates multiple copies of the same information; how do you keep them all in sync?
  - How do you guarantee transactional consistency when you read from two different servers, services or systems?

# Gotcha: Durability

- Achieving availability for stateful systems requires some form of durability
  - Synchronous replication
  - Disks, SANs
  - Transaction logging
  - Durable message queues
- Durability presents three challenges
  - Latency: To guarantee that state is written
  - Shared access between each writer and any potential failover or recovery systems
  - Coordinating readers and writers

# 8. Architecture trumps technology

*"If you can't beat your computer at chess, try kickboxing."*

*-Anon.*

# Fundamental Scalability Limiters

- Natural domain-specific limitations
  - Sequential high-latency operations
  - Globally ordered operations
  - Data "Hot Spots"

- Data Consistency and Reliability
  - Immediacy of reliable state access
  - Durability of state change

# Red Herrings

- Current Whipping Boys
  - Enterprise JavaBeans™ (EJB™)
  - Tiered Architectures
  - RPC
  - XML
  - Web Services
  - SOAP
  - HTTP
- Programming Language
- Operating System

# Legitimate Language & Platform Concerns

- Predictability of Latency
  - Unpredictable scheduling and duration of full-GC pauses
  - Lack of control over thread scheduling
  - Lack of asynchronous I/O
- Ease of Development
  - Languages differ in effectiveness for specialized tasks
- Availability of specialized libraries & components
  - We're not all rocket scientists …

# 7. Understand the Basics

*"The scientific theory I like best is that the rings of Saturn are composed entirely of lost airline luggage."*

*-Mark Russell*

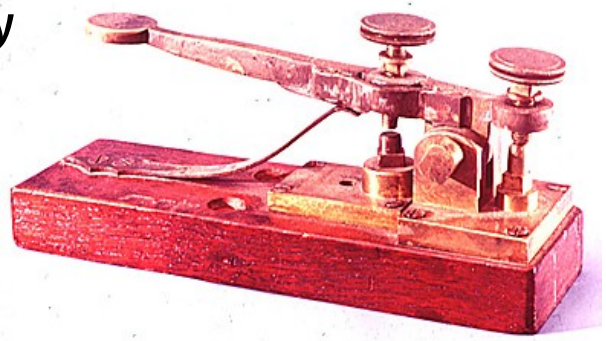
# Distribution Instruments

- Packet-Oriented (UDP/IP)
- Stream-Oriented (TCP/IP)
- Message-Oriented (JMS)
- Enterprise Service Bus (ESB)
- Request/Response (HTTP)
- Task-Oriented (Master/Worker)
- Process-Oriented (BPM, BPEL)
- State-Oriented (JCache, Data Grid)
- Procedure-Oriented (RPC, Stateless Session EJB)
- Object-Oriented (Java RMI, JINI, Stateful Session EJB)
- Event Correlation /Complex Event Processing (CEP)

# Boiling it down

- UDP/IP – Packet-Oriented
  - Unicast or Multicast
  - Relatively predictable latency
  - Lacking reliability – packets can be lost
  - Limited size payload (1.4KB or 8KB)

- TCP/IP – Stream-Oriented
  - Reliable
  - Lacks predictability of latency – latency increases sharply with load
  - TLS/SSL – adds security



- HTTP – Request/Response Model
  - Built on top of TCP/IP
  - Non-Persistent, Persistent, HTTPS

# Messaging Concepts

- Delivery Guarantees
  - None ("Unreliable")
  - Reliable
  - Guaranteed (Durable)
- Ordering Guarantees
  - None
  - Approximate Ordering
  - Point-to-Point Ordering
  - Global Ordering
- Latency
  - FIFO
  - Prioritized
  - Bounded
- Processing Guarantees
  - Best Effort
  - At Most Once
  - At Least Once
  - Once And Only Once

# Programming Models versus Messaging

- Efficient Programming Models
  - Single threaded
  - Reliable
  - In order
  - Zero latency invocation
  - Once and only once processing

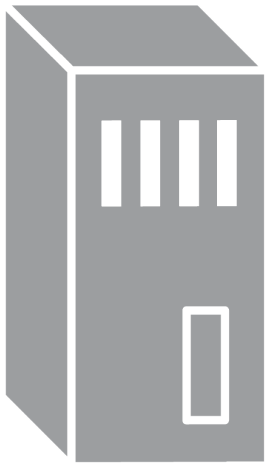
- Messaging makes you pay
  - Ordering limits scalability
  - Reliability adds costs
  - Introduces latency
  - Once-and-only-once is hard

# 6. Visualize the Network

*"Every man takes the limits of his own field of vision for the limits of the world."*

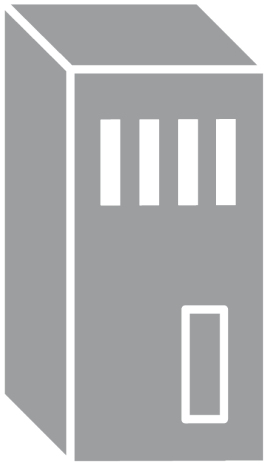
*-Arthur Schopenhauer*

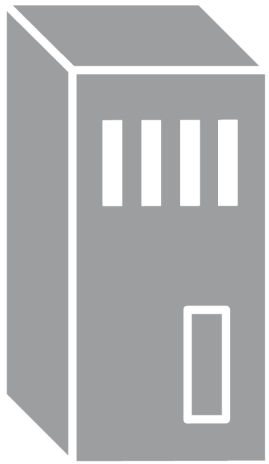
# One server, all alone ..



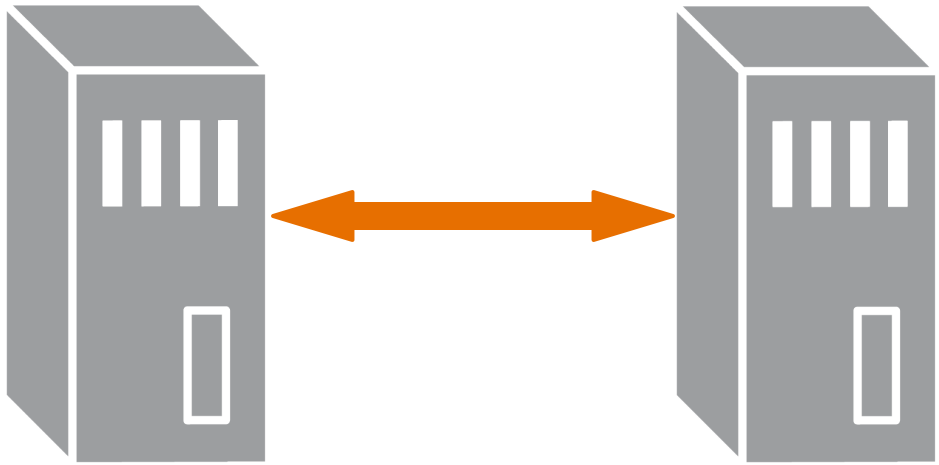
Text adapted from “Hippos Go Berserk” by Sandra Boynton

# Calls two servers ..

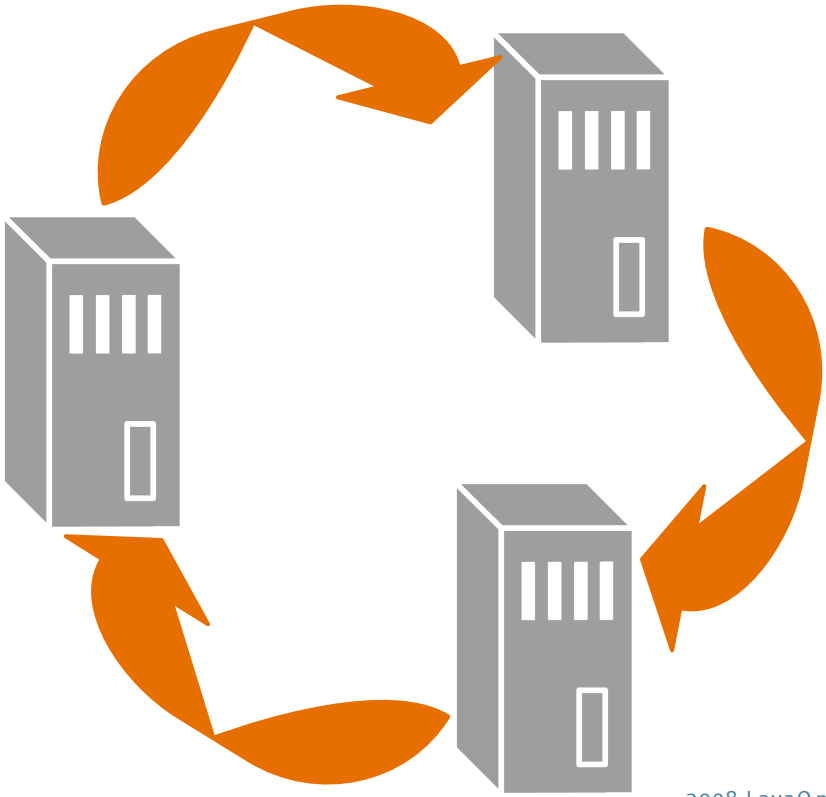




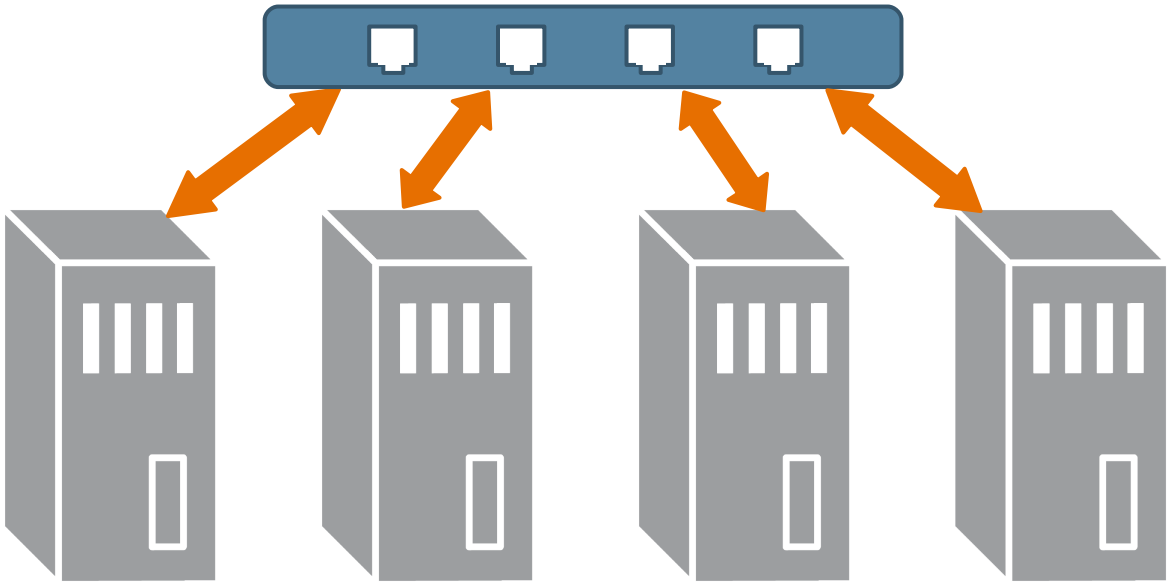
# On the phone



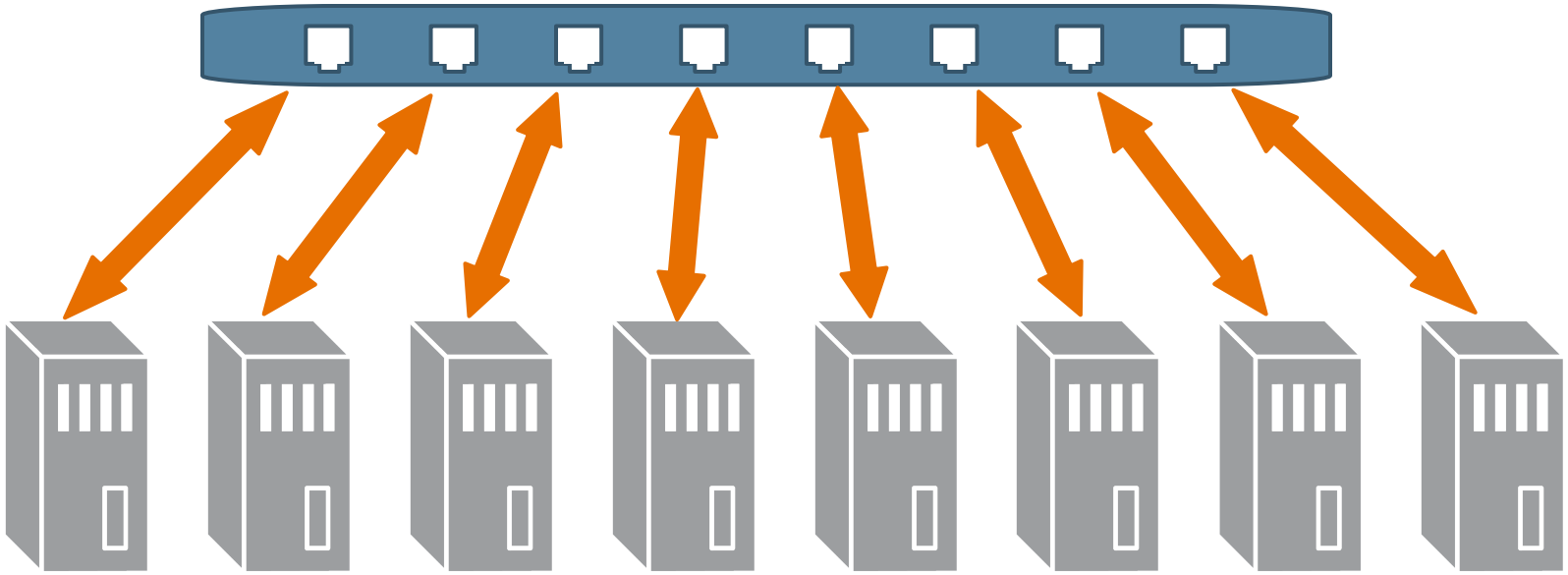
# Three servers at the door



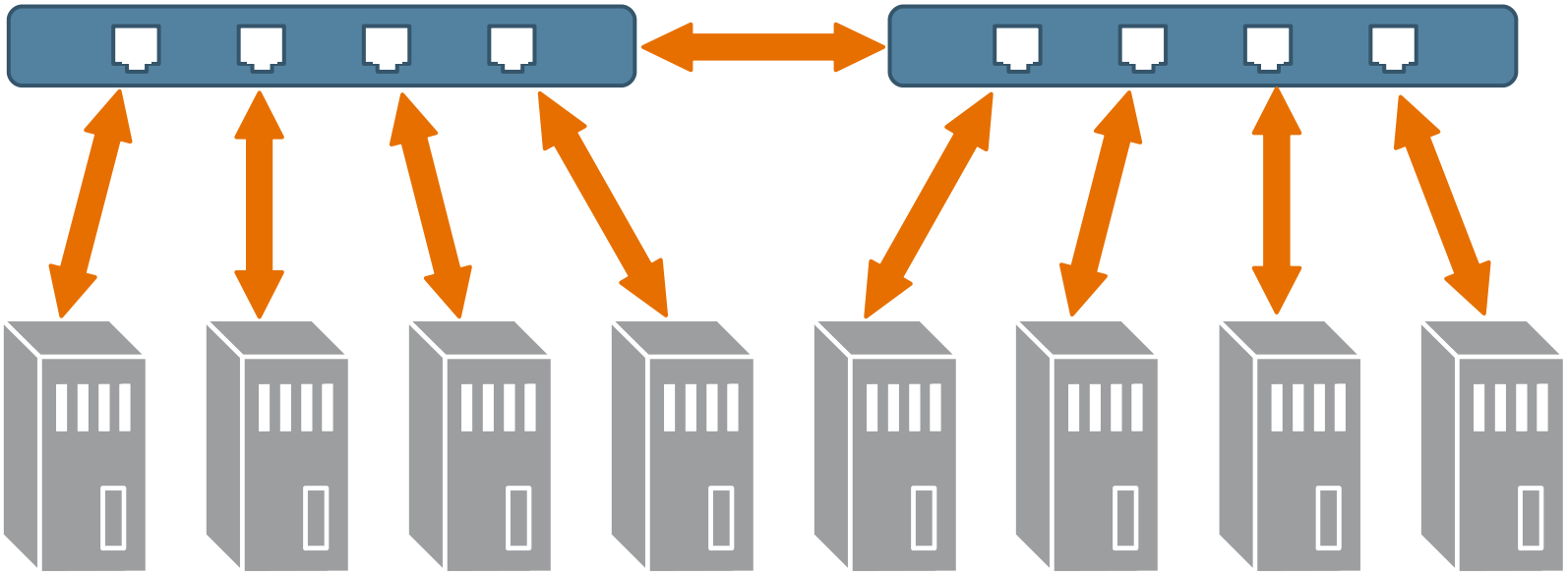
# Bring along another four



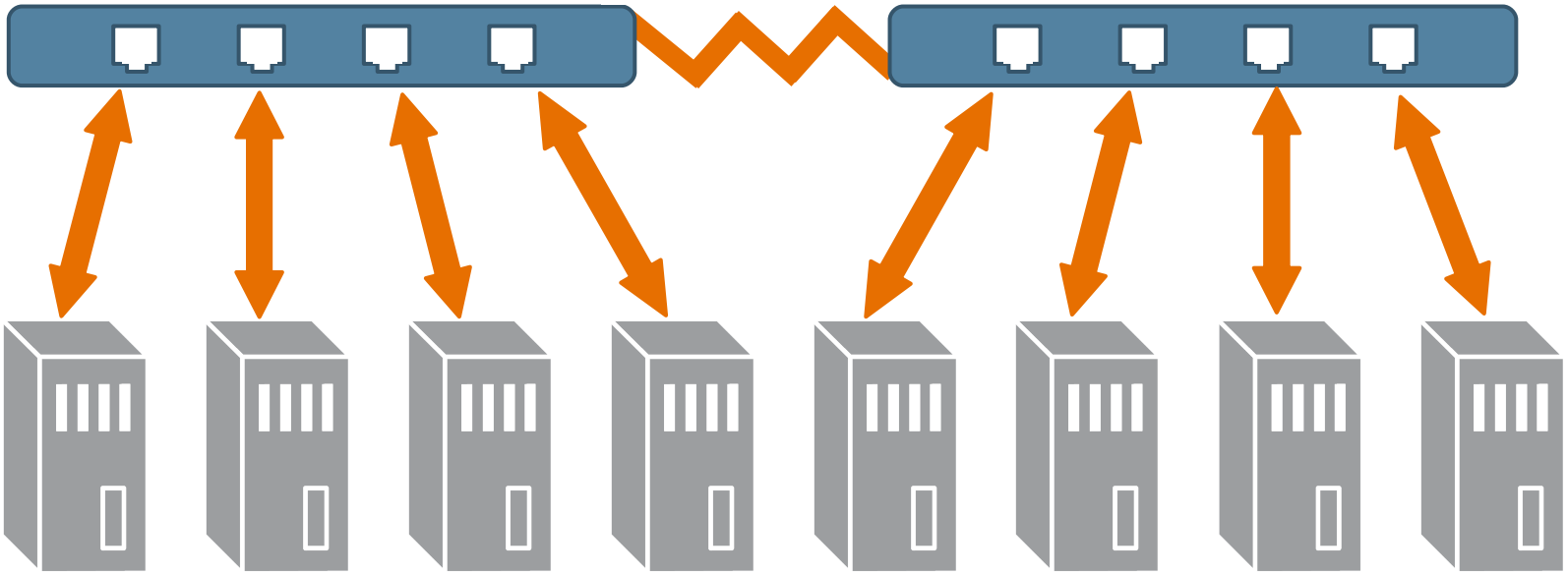
# Eight servers sneak in the back



# All the servers ..



# .. GO ACK/NACK !!!

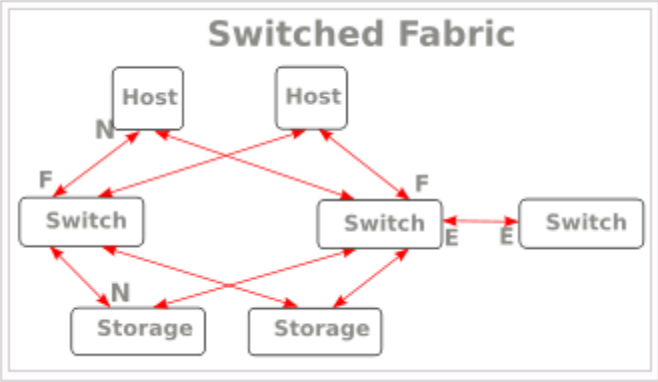


# The Wire

- Ethernet is the predominant standard
  - 10Gb emerging, 1Gb ubiquitous (cheap!), 100Mb phasing out
  - Relatively Low Latency, Relatively High Throughput
  - Predictability of Latency degrades rapidly under load
- Infiniband
  - Potentially Very Low Latency and Very High Throughput
  - Supports a variety of models, including RDMA, messaging (MPI)
  - Still exotic, but popular with some large HPC grids
- Token Passing: Token Ring, IEEE 802.5, FDDI
  - Predictable Latency
  - Low-level Ordering Guarantees are natural
  - Low Throughput

# Switched Fabric

- Fault Tolerance
- Scalability
- Direct point-to-point connections




Example of a simple switched fabric

*"A switched-fabric bus allows all nodes on the bus to inter-connect with all other nodes on the bus. All nodes connecting to all other nodes is the 'Fabric'... . To form larger networks or nodes the switches may be cascaded together to form any size [fabric] network. Switched fabric networks are used to provide Fault tolerance ...."*

Quote from: http://www.interfacebus.com/Design_Connector_SwitchedFabric_Networks.html
Image from: http://en.wikipedia.org/wiki/Switched_fabric

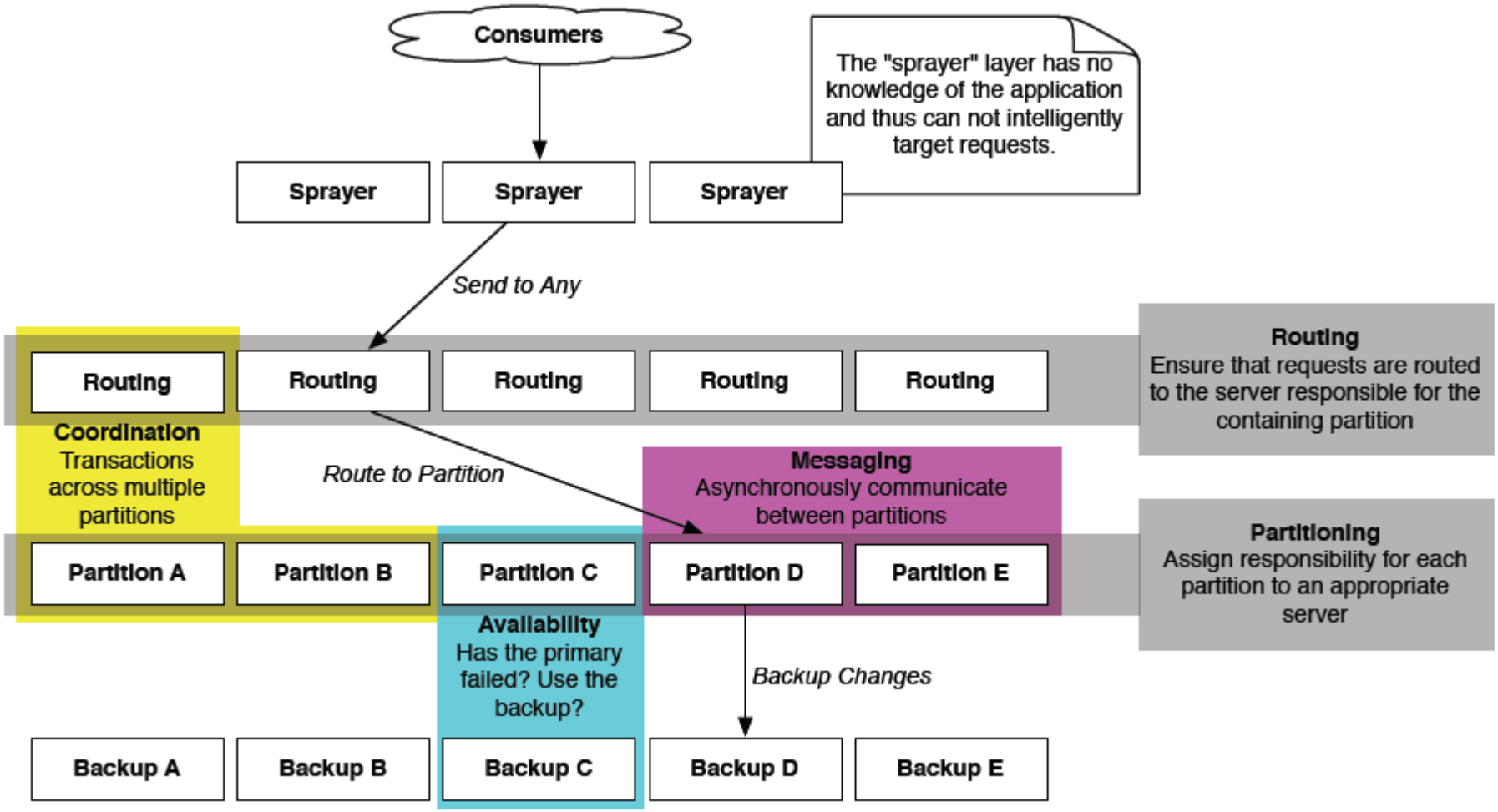
# 5. Visualize the Design

*"Un croquis vaut mieux qu'un long discours."*

*-Napoleon*

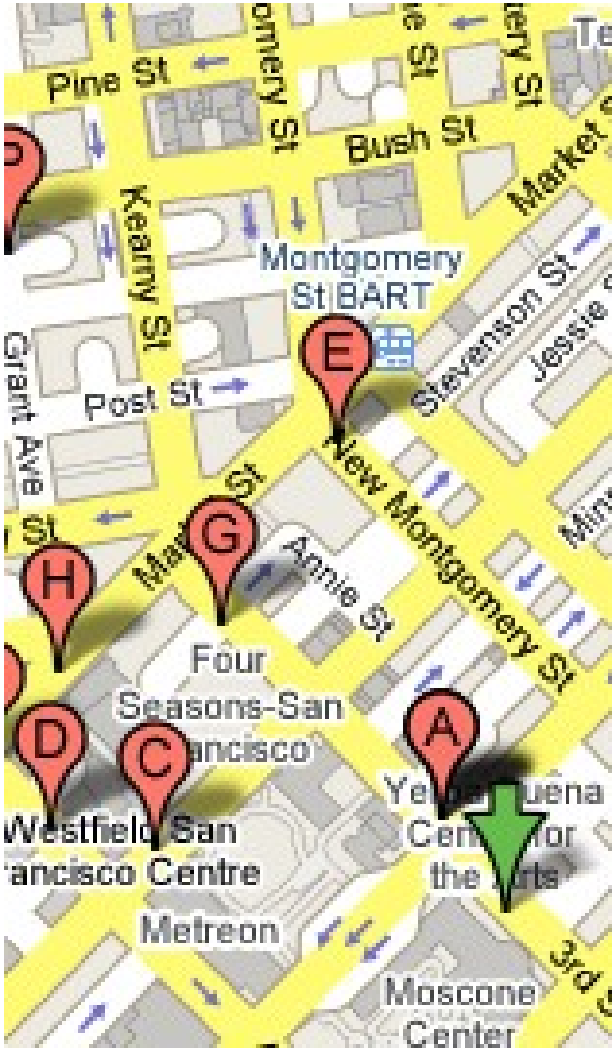
*("A sketch is better than a long speech.")*

# The Five Patterns of Stateful Scale-Out



# Routing

- In a stateful system, routing moves operations to servers where there is locality of state
- Reliable routing is the fundamental enabler of stateful scale-out
- Supports Partitioning and Replication



# Partitioning

- Partitioning designates ownership for specific state and behavior responsibilities within a distributed system
- Reduces risk by spreading responsibility
- Enables scalability
- Requires Routing, supports Replication

# Replication (Availability)

- Redundancy for surviving component failure
- Relies on synchronization of state transitions, which requires *ownership* within a distributed system
- Enabled by Partitioning

# Coordination

- Coordination conducts state change across multiple partitions within a distributed system
- Generally difficult to scale, and can work against availability
  - Widens scope of durability requirements
  - Involves multiple components in an orchestrated state-change process
- Benefits:
  - Simplifies programming model
  - Supports synchronous behavior

# Messaging

- Messaging enables synchronous and asynchronous coupling across components
- Queues support:
  - Reliability: hand-off
  - Durability: survivability
  - Ordering: sequence
  - Collection: batching

# 4a. Plan for Overload

*"It's tough to make predictions, especially about the future."*

*-Yogi Berra*

# Scalability: How to Deal with Load

- There are only so many ways to deal with load
  - Load Balancing: An arbitrary spreading of load (spraying)
  - Partitioning: Slicing a stateful domain and routing work accordingly
  - Queue: Collect work that can be performed more efficiently in bulk
  - Parallelization: Concurrent processing by spreading horizontally

# Overloaded?

- Turn away load
- Queue load and degrade latency
- Add capacity dynamically
- Increase batch size or queue delay for batching
- Relax read and/or write consistency

# 4b. Partition for Scalability

*"The way a team plays as a whole determines its success. You may have the greatest bunch of individual stars in the world, but if they don't play together, the club won't be worth a dime."*

*-Babe Ruth*

# Partition for Scale

- Partitioning is the only way to provide significant scale in a stateful system
  - Dynamic partitioning supports dynamic capacity
- Introduces challenges for distributed read consistency
- Even with custom affinity of data within a distributed system, complete data locality is simply impossible for some applications

# 3a. Plan for Failure

*"The most likely way for the world to be destroyed, most experts agree, is by accident. That's where we come in; we're computer professionals. We cause accidents."*

*-Nathaniel Borenstein*

# Fault Tolerance

- The three R's
  - Redundancy
  - Reserve
  - Recoverability
- Plan for failure at two levels
  - Component
  - Systemic
- When possible, use failover to transparently handle component failure
- Use recoverability to handle everything else
  - Requires durability


Image from: http://failblog.wordpress.com/

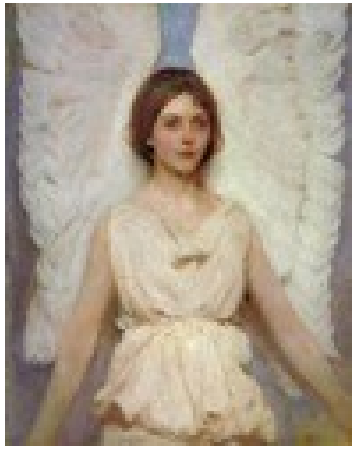
# 3b. Replicate for Availability

*"When all men think alike, no one thinks very much."*

*-Walter Lippmann*

# Replication for Availability

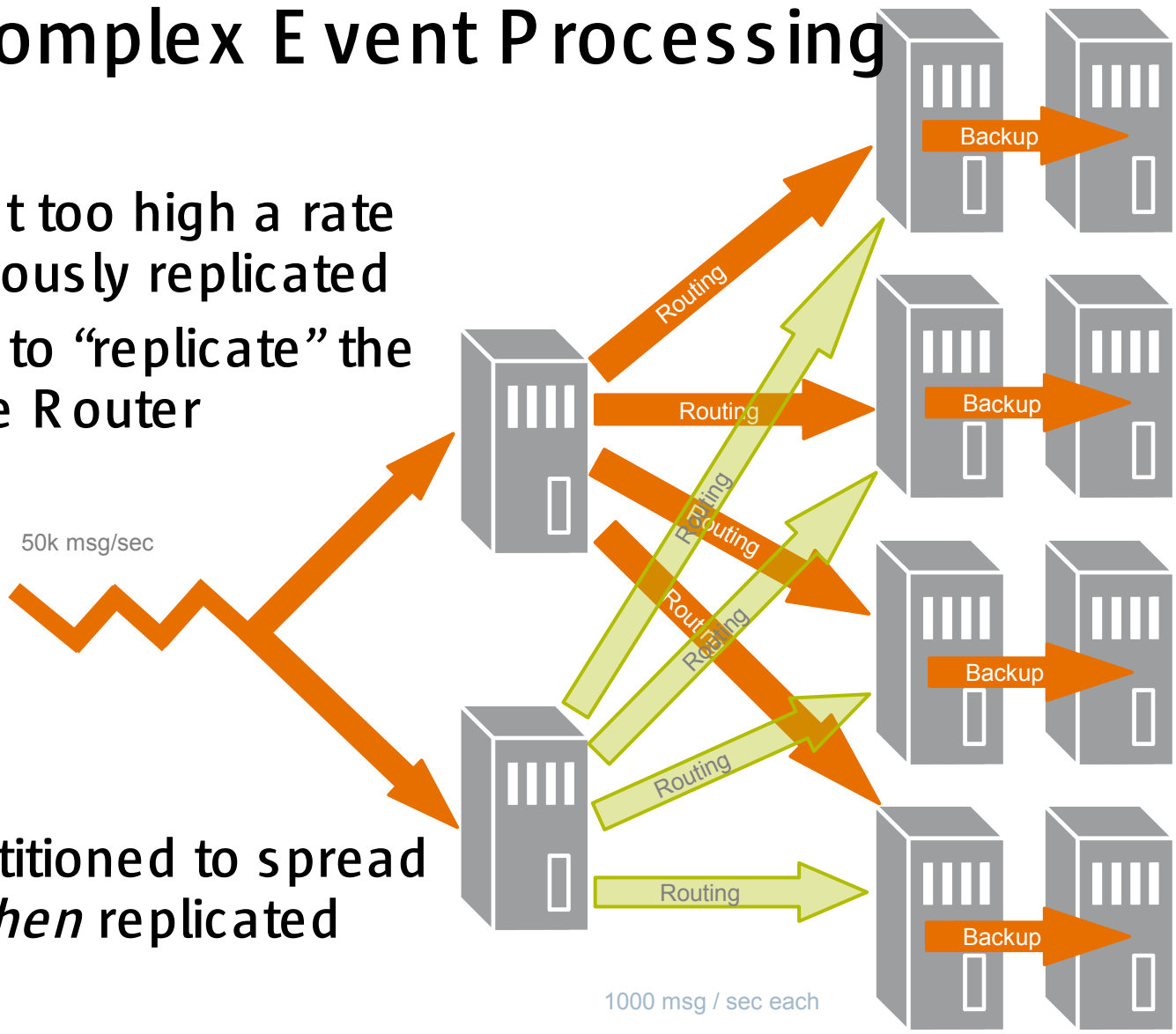
- Do:
  - Synchronously replicate for durability
  - Replicate at levels that are responsible for failover
  - Asynchronously replicate for increased locality (e.g. caching), or for recoverable data



- Do not:
  - Replicate synchronously inside the "big O" or a critical section *(... unless you have to)*
  - Replicate information that can be easily reconstructed or efficiently recovered

# Example: Complex Event Processing

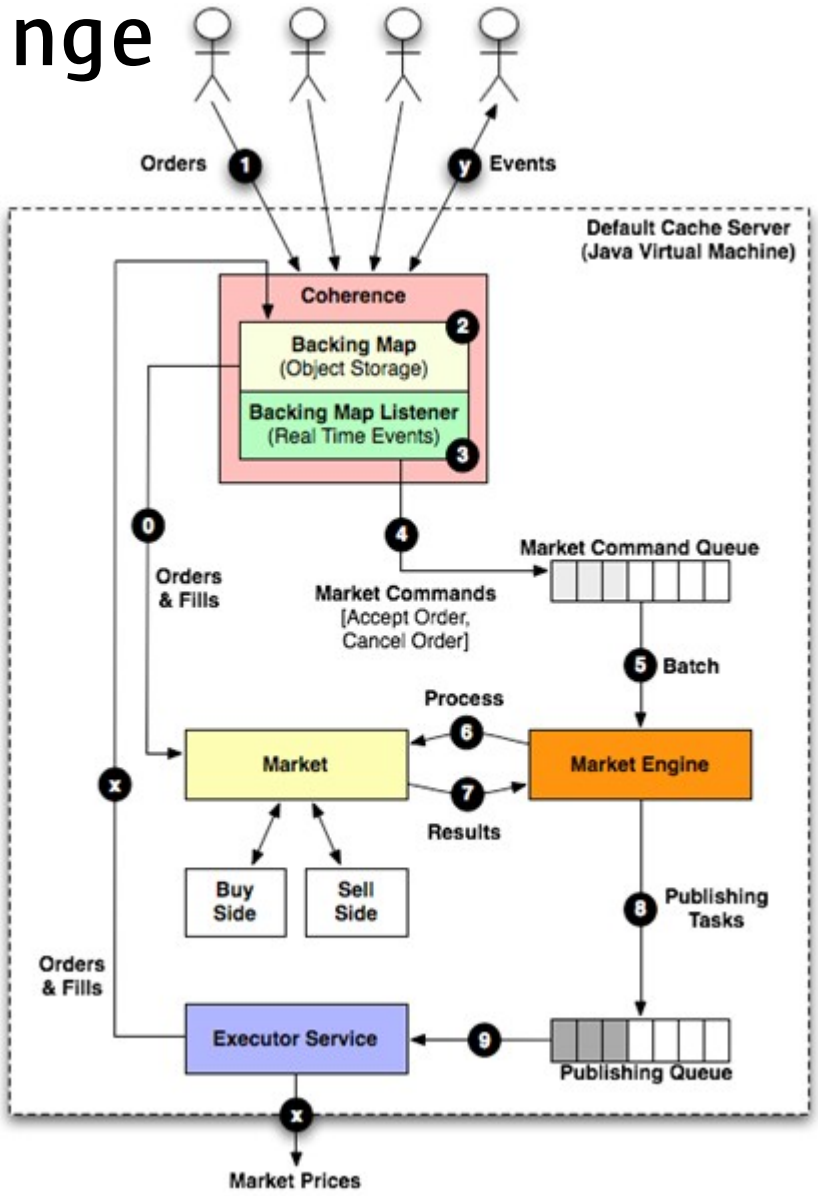
- Events arrive at too high a rate to be synchronously replicated
- The solution is to "replicate" the entire Message Router




- Events are partitioned to spread the load, and *then* replicated

# Example: Foreign Exchange
## *An Event-Driven FX System*

- Architecture
  - Partitioned and Load-Balanced
  - System is a Finite State Machine
  - Data Affinity: Markets, Orders, Fills
  - Process Affinity: Markets & Matching Engines
  - Only Orders and Fills are replicated
  - Market state and Matching Engine is reconstructed on failover
- Results
  - 1000x improvement in throughput
  - 95% hardware cost savings by using a commodity hardware grid
  - 99%+ Orders processed in <3ms

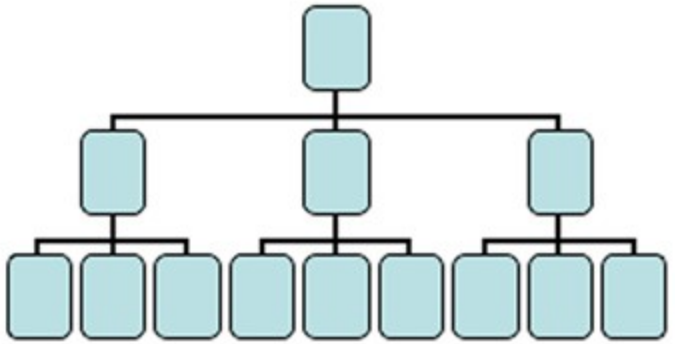


# 2. Tier where it makes sense

*"The point of philosophy is to start with something so simple as not to seem worth stating, and to end with something so paradoxical that no one will believe it."*

*-Bertrand Russell*

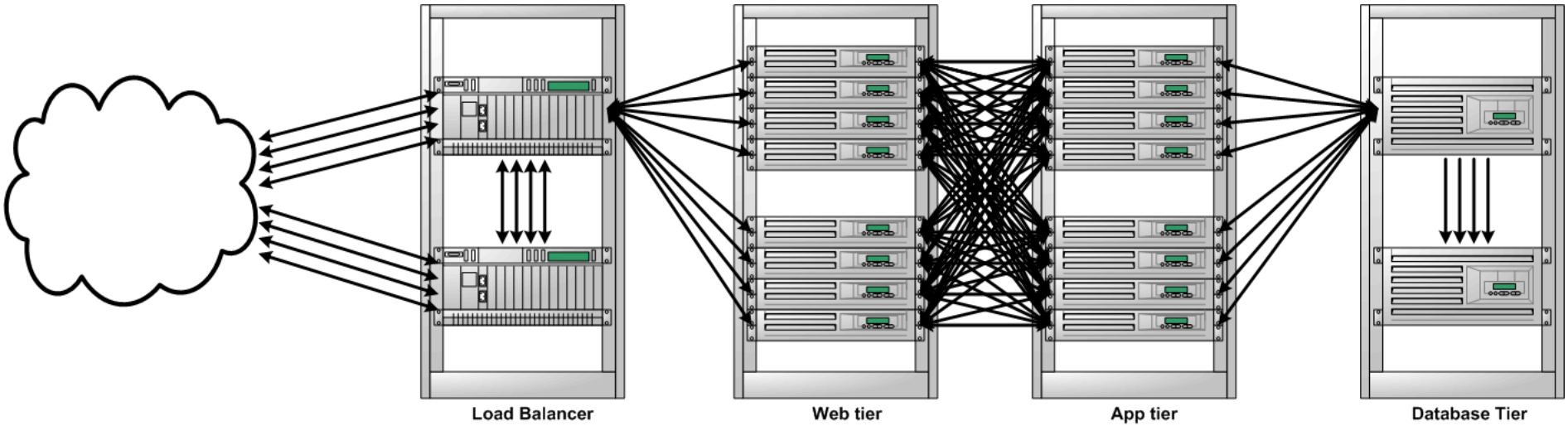
# Why Tier?



- Tiering is *not* horizontal scalability
- Pros
  - Tiering *can* be used to increase scalability
    - Each tier provides an opportunity to partition (and thus co-locate) responsibilities in a different manner
    - Localization can eliminate the need for Coordination, improving Scalability
    - Tier traversal may create opportunities for queuing for batch efficiency
- Cons
  - Added complexity
  - Potential for significantly reduced efficiency
    - Tier traversal should be on a constant order
- There is no reason to tier stateless systems for scalability

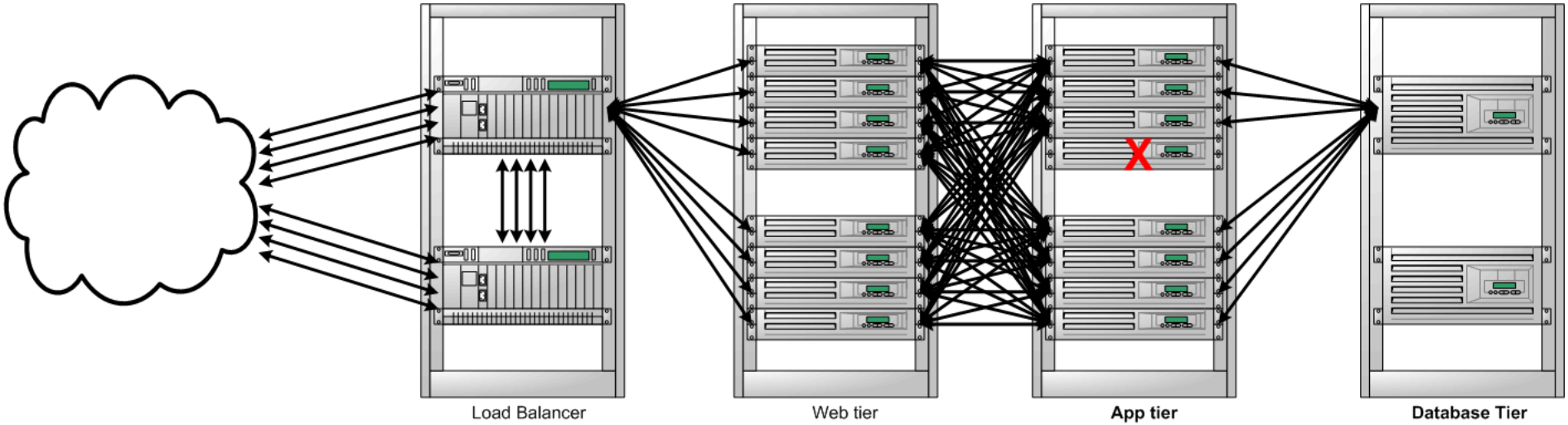
# Traditional Three Tier Web Architecture

- Spraying ("load balancing") across the web tier
- Routing ("sticky load balancing") to the application tier using HTTP Session-based Partitioning
- Replication (for Availability) of HTTP Session data



# Traditional Three Tier Web Architecture

- Failure of a stateful server in the application tier changes the routing to go to the backup servers for the affected sessions
- Requests that were in flight can be transparently resubmitted to the backup if they were idempotent



# To Tier or not to Tier?

- Latency
  - Tiering may facilitate increased locality
  - Compare the cost of tier traversal versus the cost of accessing shared services (coordination)
- Scaling
  - Does scaling one tier require scaling the others?
  - Is there an opportunity to introduce parallelism?
- Availability
  - Is availability independently maintained for each tier?
- Efficiency
  - Are CPU and network resources wasted on communication?
  - Does the reduced coordination and increased locality compensate?

# 1. Simplify

*"Plain question and plain answer make the shortest road out of most perplexities."*

*-Mark Twain*

# Live the Simple Life

- Complexity is the enemy of Reliability
- Complex distributed systems must be modeled as Finite State Architectures
  - It is the *only* way to prove correctness
- Partitioning comes with hidden treasures
  - Localized ordering is almost free
  - Idempotency of operations can be managed by the partition owner
- Change your programming model: Use an Event Driven Architecture to achieve the highest levels of concurrency

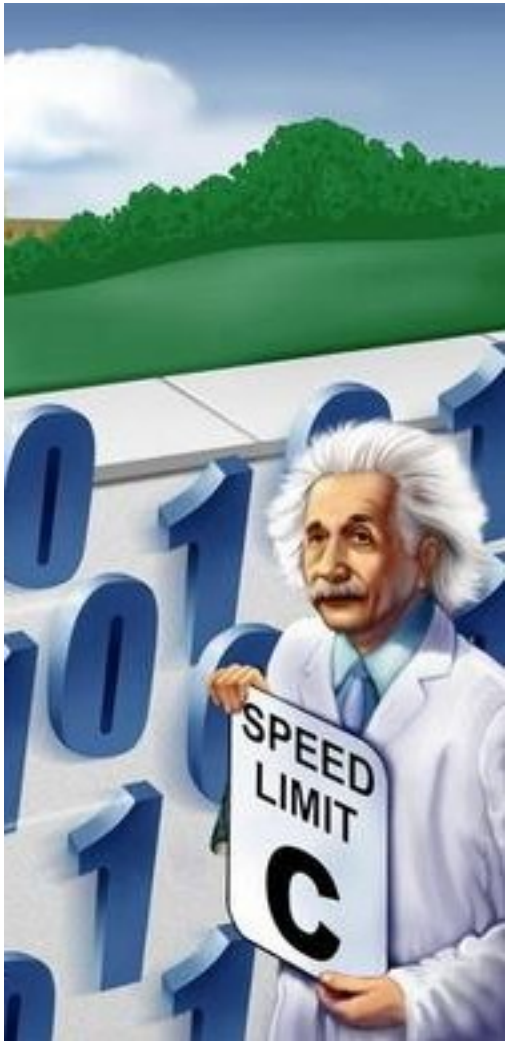
# Conclusion

*"'Thou shalt not' is soon forgotten, but 'Once upon a time' lasts forever."*

*-Philip Pullman*

# It's *not* Rocket Science
## *(.. it's just physics)*

- For stateful systems, the challenge is to achieve Availability, Reliability, Scalability, Performance – including predictability – in systems that can provide both Manageability and Serviceability
- Effective solutions are composed from some or all of: Routing, Partitioning, Replication, Coordination and Messaging
- If you can't show it working on the white board, you can be certain that it won't work in production



# THANK YOU

Cameron Purdy, Vice President, Oracle Corp.

TS-6339