Designing for Scalability

Patrick Linskey
pcl@apache.org

Patrick Linskey

Apache OpenJPA Committer

JPA 1, 2 EG Member

EJB3, EJB3.1 EG Member

Agenda

- Define and discuss scalability
  - Vertical
  - Horizontal
- Examine ways to make software scale
  - Code / Algorithms
  - Asynchronous Libraries
  - Other Languages

Scalability

- Ability to increase the total number of operations performed in a unit of time
- Vertical Scalability:
  - “Make the machine bigger”
- Horizontal Scalability
  - “Add more machines”

Bottlenecks

- Limit the scalability of a system
- Intrinsic bottlenecks
- Artificial bottlenecks

Example Problem Domain

- Financial fund management
- Multiple in-house engineering needs
  - Trade Execution
  - Trade Settlement
  - Strategy Definition
  - Strategy Simulation
  - Portfolio Risk Analysis

Vertical Scalability

Translated into Java:

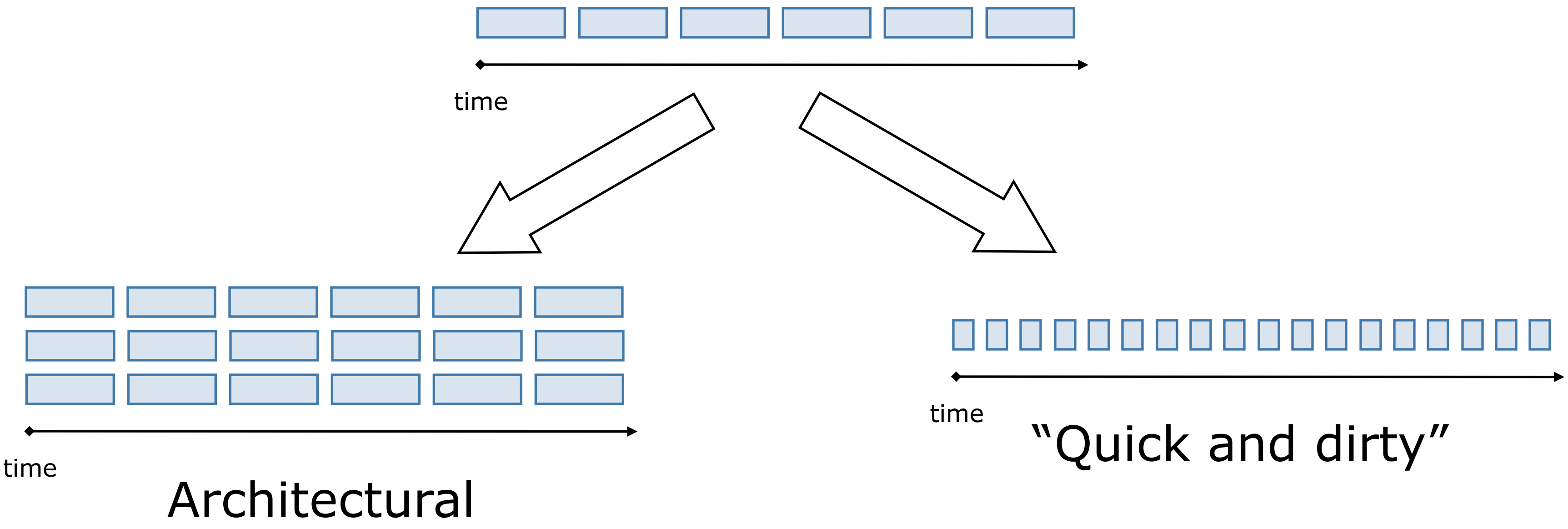
Scaling Within a Machine

Vertical Scale Factors In Your Control

- **Improve code efficiency**
  - **Memory**
  - **CPU**
- **Optimize I/O between physical tiers**
  - **Web 2.0: beware!**
- **Make code scale across multiple cores / CPUs**

Code Optimization Possibilities

- **Performance and scalability are linked**
- **Scalability: more operations per time unit**



“Scale” Vertically via Code Optimization

- Reduce copying, looping, etc.
  - “Write good code”
- SQL statement batching
  - `PreparedStatement.addBatch()`
  - ORM frameworks
- Transaction batching
  - Especially powerful in XA environments
  - JMS message batching

Synchronization

- `synchronized` is for *asynchronous execution*
  - "Execute this block of code in its entirety before others that share this lock"
- Modern computers handle high* concurrency
  - `synchronized` is often a bottleneck
  - Avoid synchronization at runtime *at all costs*
    - uncontended synchronization is cheap

Write-Once Shared Memory

```
class SlowTradeManager {
  private Set types;
  public synchronized Set
    getTradeTypes() {
    if (types == null)
      types = loadTypeData();
    return types;
  }
}
```

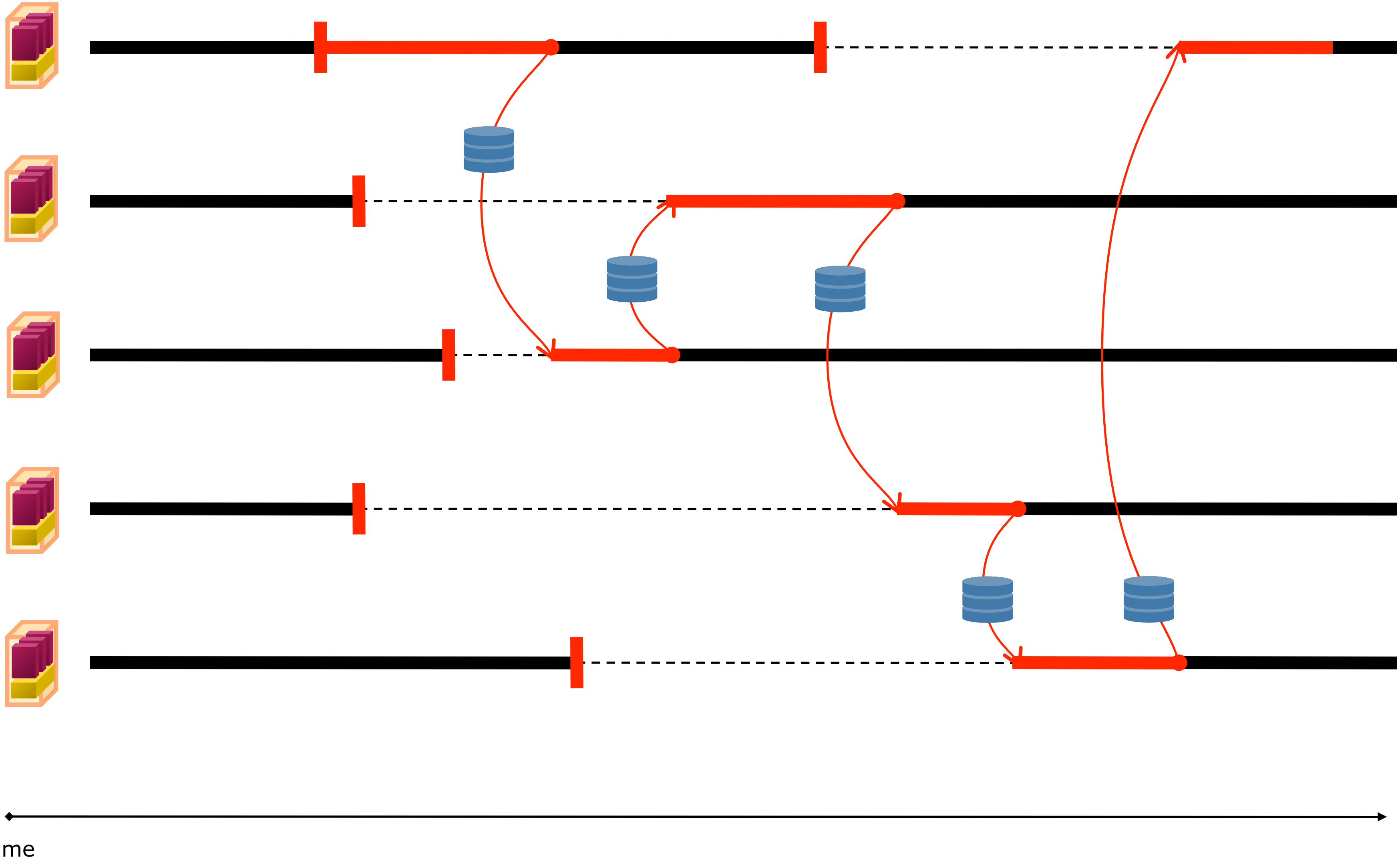
```
class FastTradeManager {
  private Set types;
  public Set getTradeTypes() {
    if (types == null)
      types = loadTypeData();
    return types;
  }
}
```

loadTypeData() might be called more than once

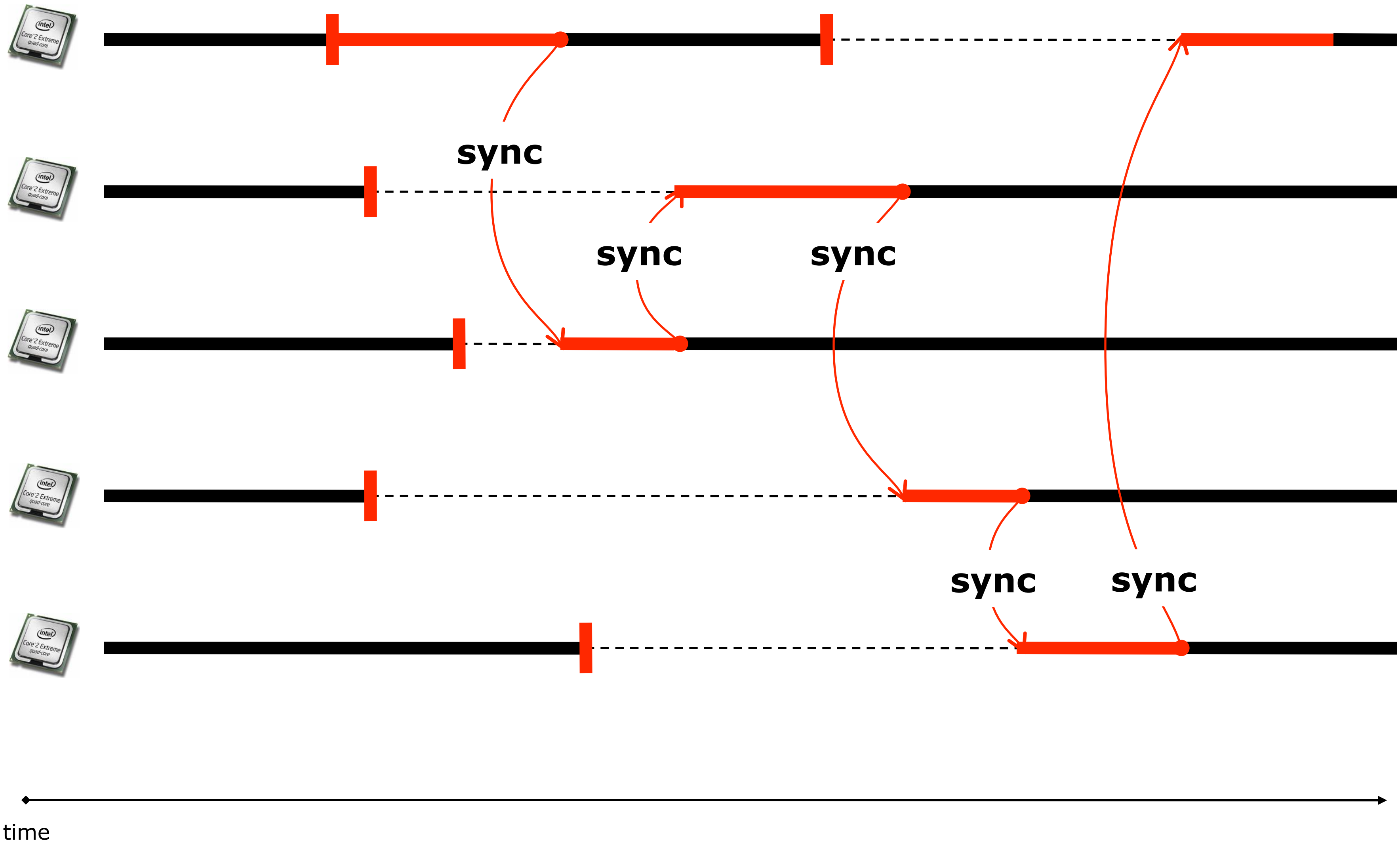
Fund Risk Balancing

- Problem
  - Multiple traders act on the same security
- Solution
  - Maintain fund-global position data
  - Mutable shared state!

Multi-machine solution (circa 1998)



Multi-core / CPU synchronization

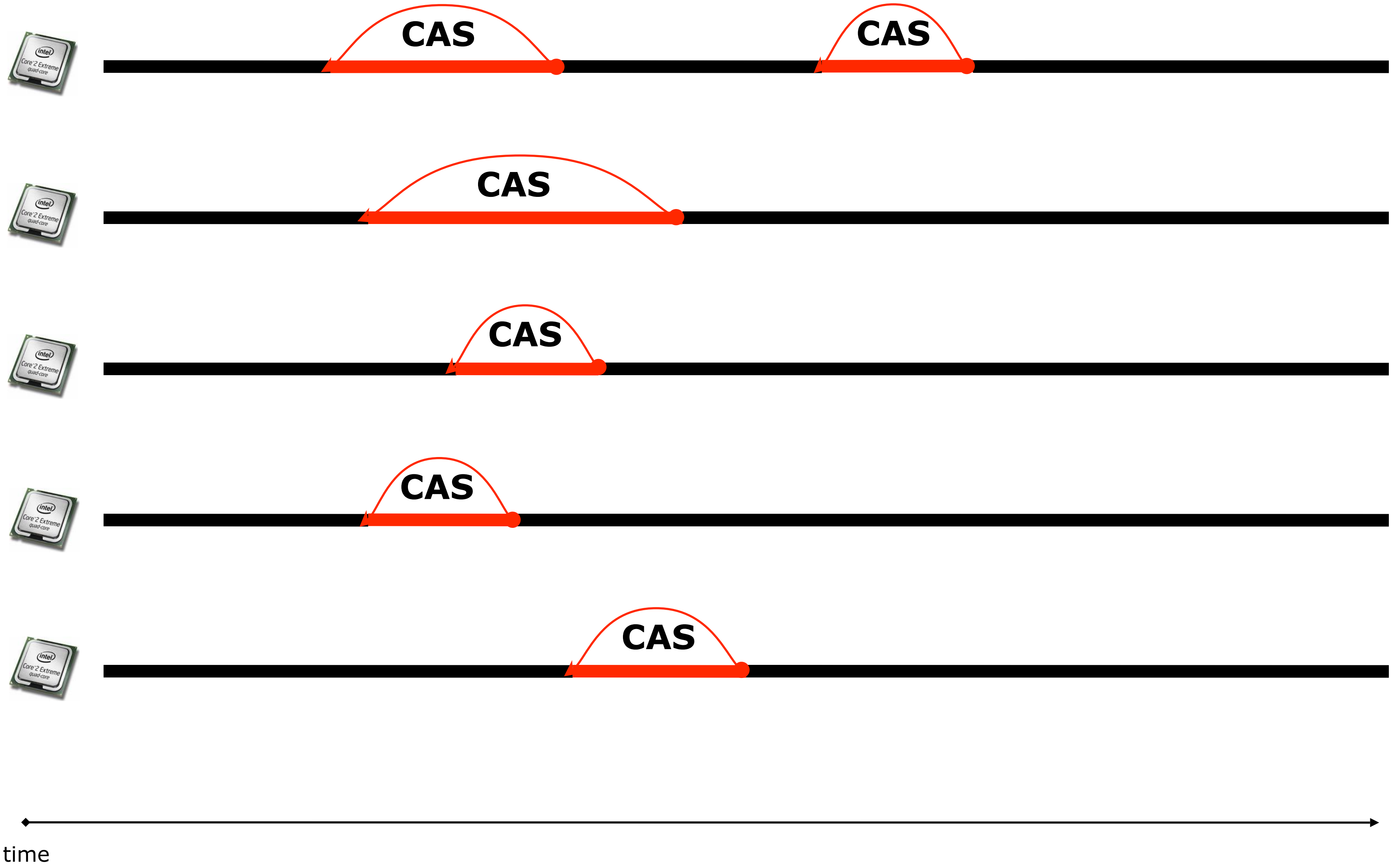


Mutable Shared Memory

```java
import java.util.concurrent.atomic.AtomicDouble;

class AggregateFundPosition {
    private AtomicDouble totalExposure = new AtomicDouble(0);
    public double incrementBy(double amount) {
        while (true) {
            double old = totalExposure.get();
            double next = old + amount;
            if (counter.compareAndSet(old, next))
                return next;
        }
    }
}
```

Synchronization-free shared state



Horizontal Scalability

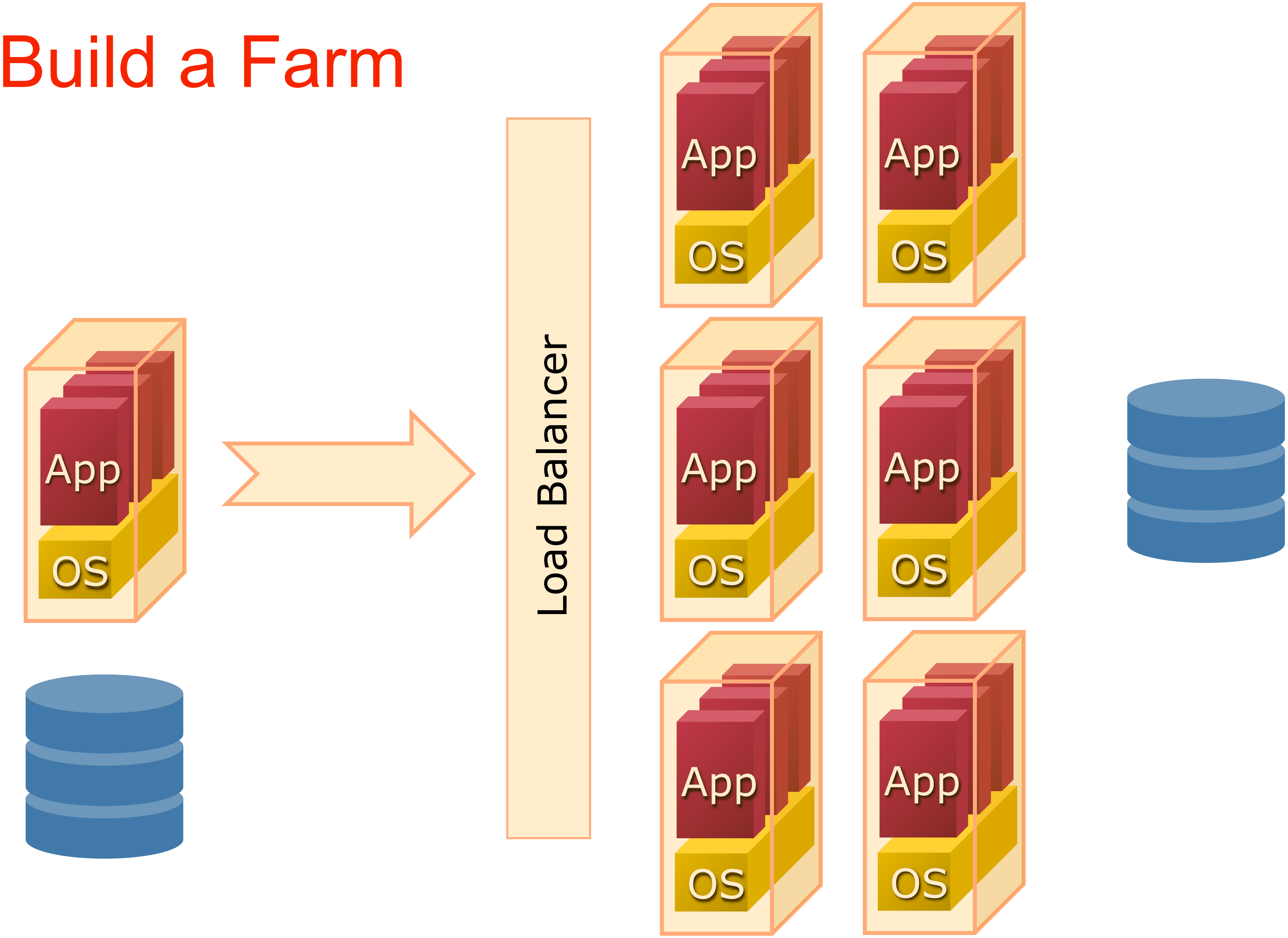
Translated into Java:

Scaling Across Machines

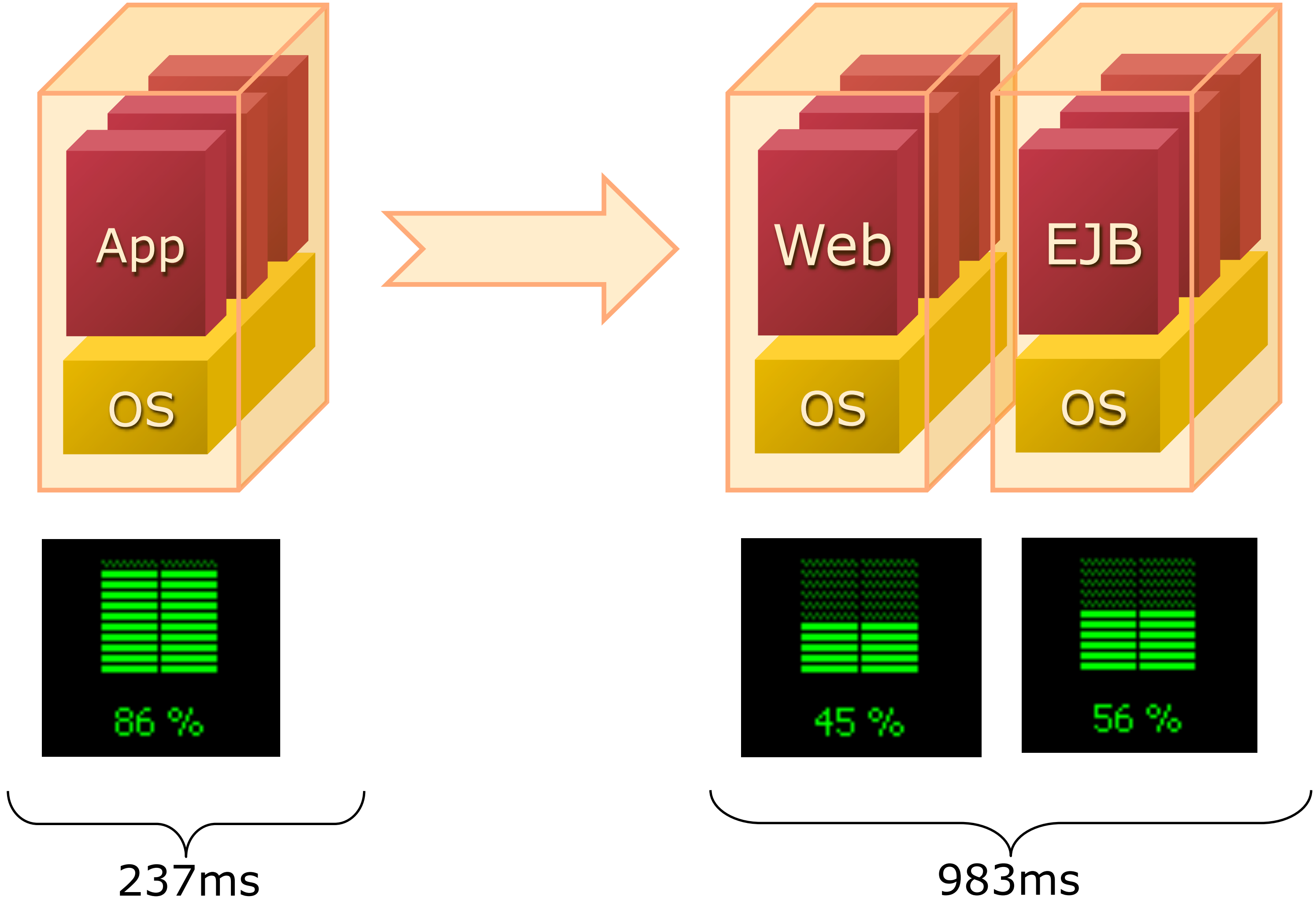
Horizontal Scaling: Add More Servers

- **All doing the same thing**
- **Partitioned by infrastructure layer**
- **Partitioned by application role**
- **Partitioned along data graph boundaries**

Build a Farm



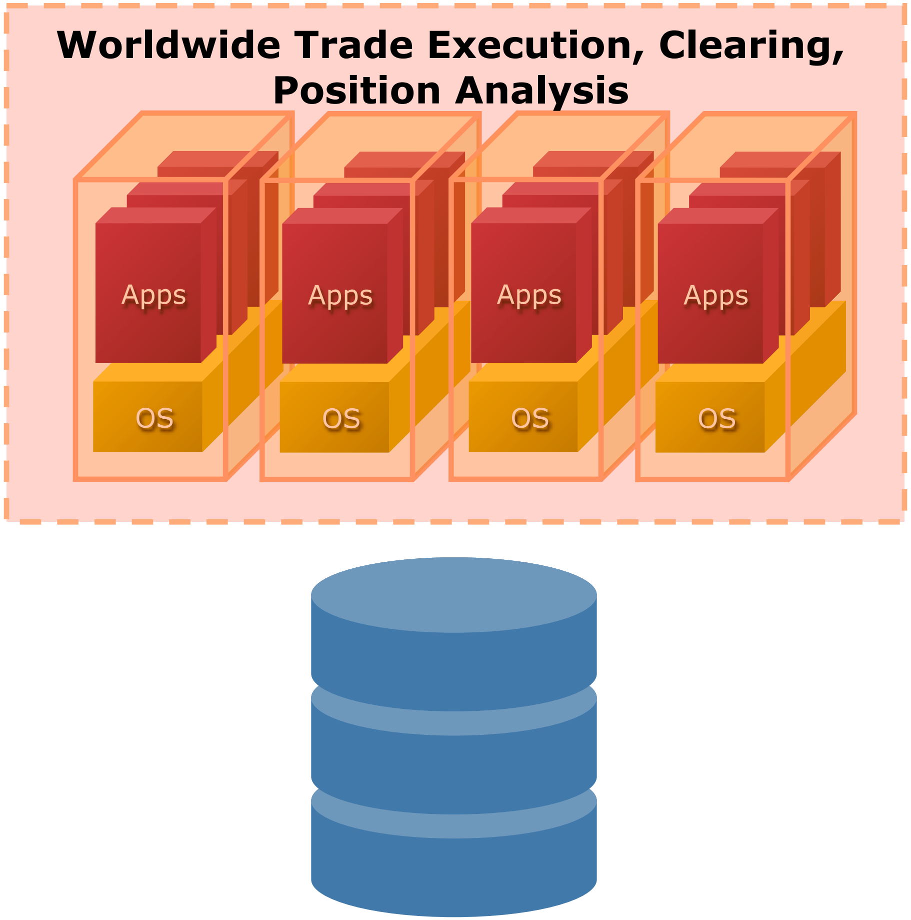
Slow Down



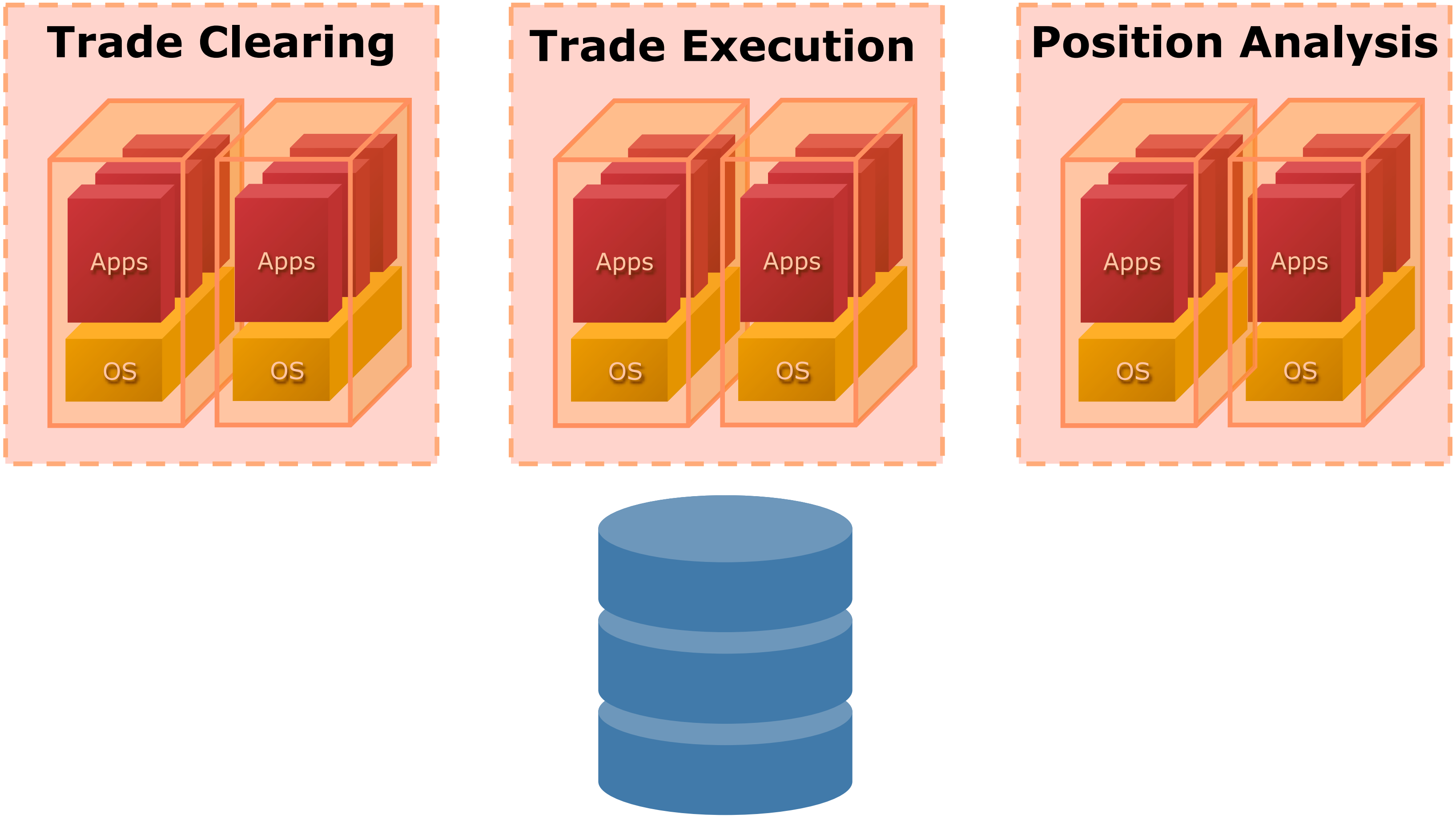
Divide and Conquer

- Old as `` `time` `` itself
  - mail, news, telnet all on different servers
- You use partitioning every day
  - Telephone call routing
  - ATM card transactions
  - Stock markets
  - Elevator banks

Break Up Stateful Services



Partition Along Application Boundaries

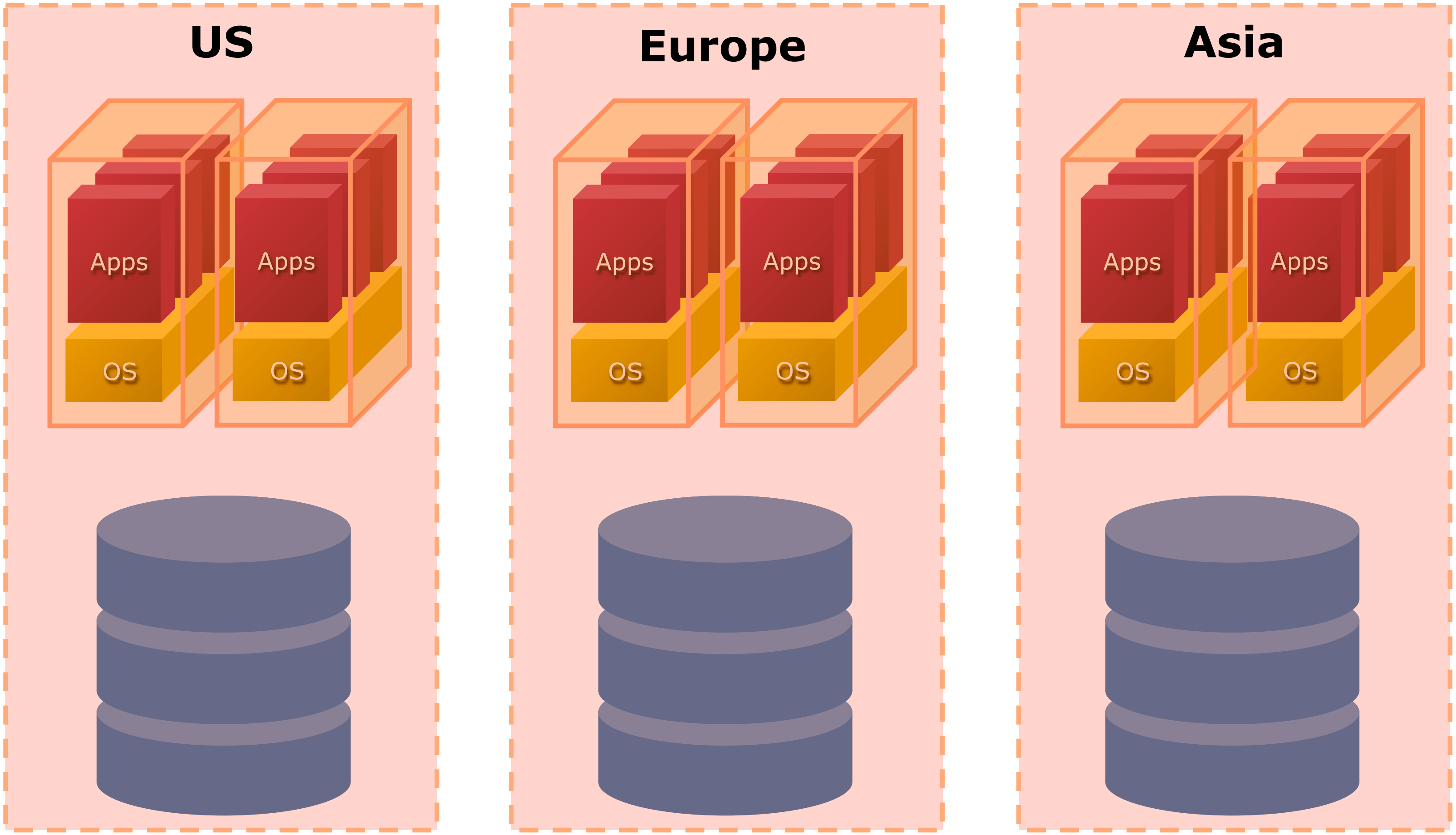


Trade Clearing

Trade Execution

Position Analysis

Partition along data set “fault lines”

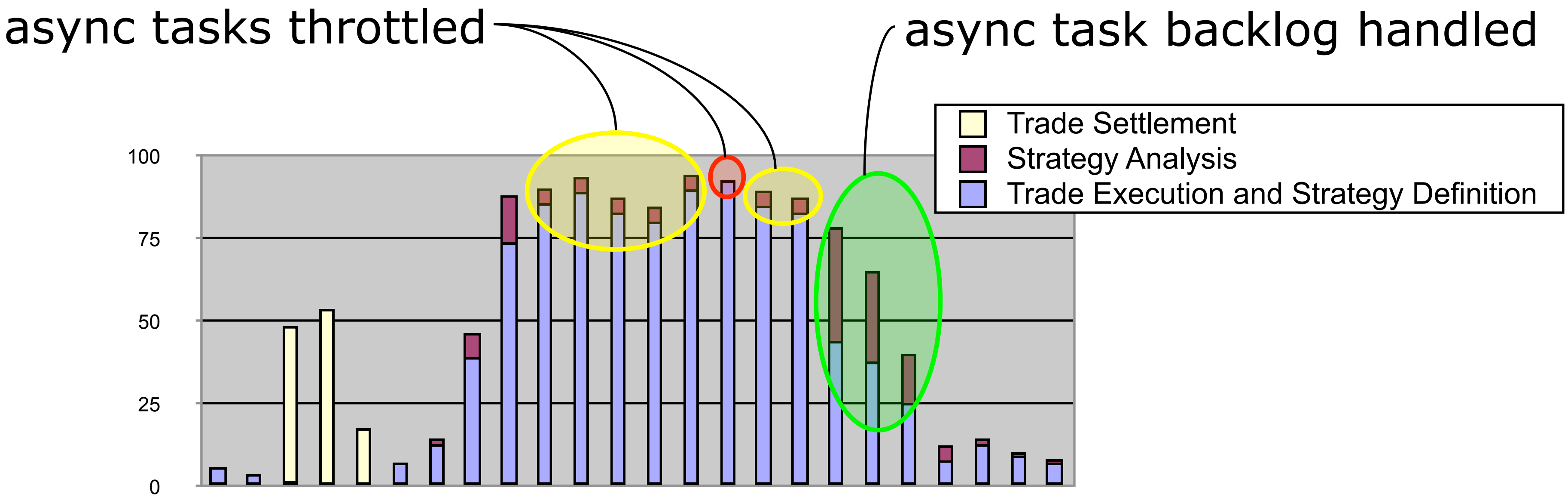


Asynchrony in Java

- Java is a mostly synchronous environment
- Business algorithms often aren’t
- Take advantage of this where possible
  - JMS message queues
  - java.util.concurrent.ExecutorService
  - commonj.work.WorkManager
  - Scheduled jobs

Async Tasks and Resource Utilization

- **Good JMS servers / ExecutorServices / WorkManagers do resource tuning and optimization**
  - **Limit threads allocated to async processing**
  - **Configure priority of async vs. sync (i.e., HTTP request)**



async tasks throttled

async task backlog handled

Adapt Requirements to Concurrency

- Identify slow-running / expensive parts of the user experience
- Work with requirements team to replace these with asynchronous processes
    - Website usage statistics generated nightly instead of on-demand
    - Dynamic PDF delivery via email instead of embedded web content

Starting from Scratch

Choose Your Toolset

- **Java makes synchronization easy**
  - **... but synchronization != scalability**
- **Other languages avoid shared state**
  - **Rely on message-passing instead**

Erlang: Functional, Asynchronous, Mature

- Designed for concurrency *in the language*
  - Parallel execution
  - Intrinsic hot-redeploy
  - State can only be assigned once
- Communication happens via message-passing between actors
  - No threads → no shared state!
  - JMS-like behavior; language-native syntax

Scala: Functional Programming for the JVM

- **Java-integrated**
  - **Designed by Java stalwart Martin Odersky**
- **JVM-optimized**
- **Supports Erlang-style concurrency**

Compute Grids

- Federate your data around a cluster
- Decompose your algorithm into serializable work items
- Let the compute grid send your work items to the data

Decision Factors

- What are your application requirements?
  - How many concurrent operations?
  - How big of a workload?
  - What sorts of SLAs?
- Tolerance of deployment complexity?
  - How about your operations, QA teams?

Recap

- Concepts
  - Scalability
  - Bottlenecks
  - Synchronization
  - Asynchrony vs. concurrency
  - Compare-and-set
  - Application Partitioning
  - Synchronous tasks vs. asynchronous tasks

- Technology
  - java.util.concurrent
  - j.u.concurrent.atomic
  - Operation batching
    - Transactions
    - SQL
  - JMS; Executor; WorkManager
  - Scala and Erlang
  - Hibernate Shards
  - OpenJPA Slice

Questions

Patrick Linskey
pcl@apache.org