




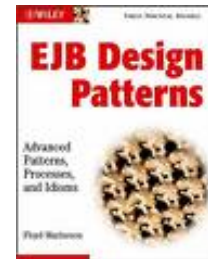
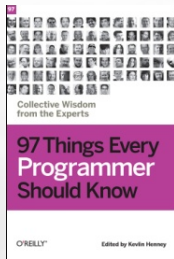
G A M E S

League of Legends: Scaling to  
Millions of Ninjas, Yordles, and  
Wizards



# Speaker Introduction

- Scott Delap
  - Scalability Architect, Riot Games, Inc.
  - [sdelap@riotgames.com](mailto:sdelap@riotgames.com)
  - @scottdelap 
- Randy Stafford
  - Consulting Architect, Coherence Product Dev.
  - Formerly Chief Architect, IQNavigator



# Introducing Riot Games



# Introducing Riot Games

- Tripled in headcount over the last 12 months
- Ranked #47 on the Business Insider Digital Startup 100
- US and Europe Currently
- Expanding internationally
  - China, Philippines
- Agile development
- Release every 2 weeks





# Introducing Riot Games

- Launched in Oct of 2009
- MOBA
  - Multiplayer
  - Online
  - Battle
  - Arena
- Battles happen online in real time
- 5x5 or 3x3 Matches
- Typical game is 30-45m in length
- #3 Most Played Online PC Game
  - Xfire.com
  - Gamespot.com

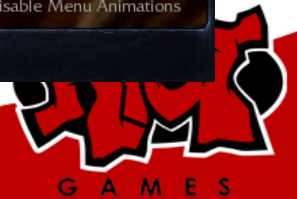
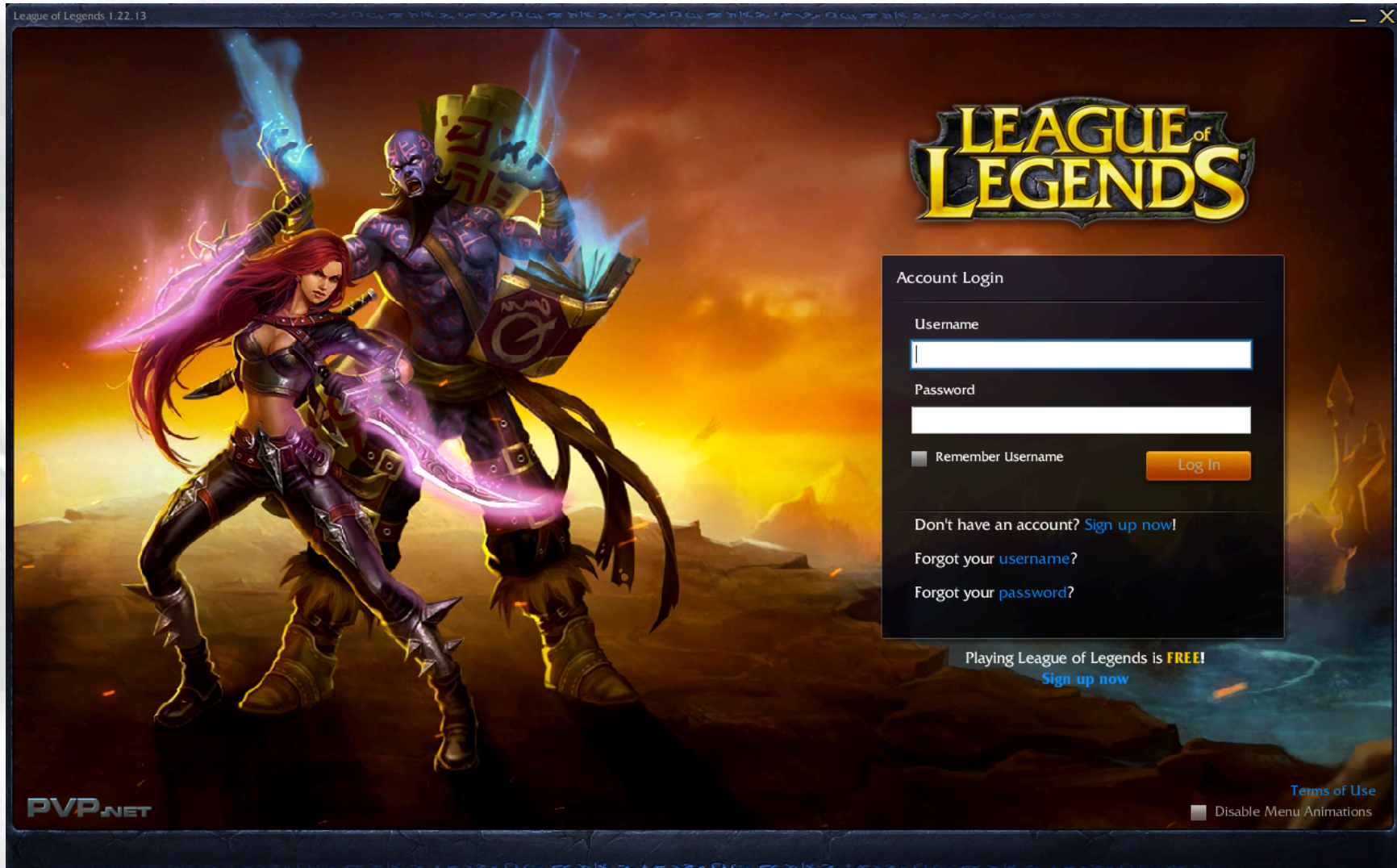


# Introducing Riot Games





# Introducing Riot Games



# Introducing Riot Games

League of Legends 1.22.13

LEAGUE OF LEGENDS

Play

Scott D 69055 x 10173

Home

**SKINS SALE**  
50% OFF  
Guerilla Tristana, Emerald Taric, and Jack of Hearts are available for 50% off until November 8th!  
Click here to purchase and unlock this discounted skin.

My Account | Purchase Riot Points

**On Sale 85:59:41 Until Sale Ends**

Guerilla Tristana	975
50% OFF	487

**On Sale 12:59:41 Until Sale Ends**

Unmasked Kayle	975
50% OFF	487

**What's New**

<b>LeBlanc</b> The Deceiver 975 OR 3150	<b>LeBlanc Double Bundle</b> 1722	<b>Wicked LeBlanc Bundle</b> 1462	<b>Perseus Pantheon</b> 975
---	--------------------------------------	--------------------------------------	--------------------------------

**Top Sellers**

- LeBlanc The Deceiver
- Wicked LeBlanc *Requires Champion*
- Lux The Lady of Luminosity *Already Owned*

Home Champions Skins Boosts Runes Bundles Other Codes Riot Points

(0)





# Introducing Riot Games



# Introducing Riot Games



# 2010 Awards

*“Our Favorite Free Game” – PC Gamer*





# Critical Acclaim

“One hell of a great multiplayer game” - Gamespy

“A satisfying strategy game that's not for the faint of heart” – IGN

“I can't stop playing League of Legends.” -Kotaku

“One of the most refreshing strategy games in years.” – GameTrailers



# A Unique Scale Challenge

- Planning for Fuzzy Growth
- New concurrency highs regularly
- 2 million downloads as of June
- No Barrier of Entry
  - League of Legends is free to download and play.
  - Highly viral growth

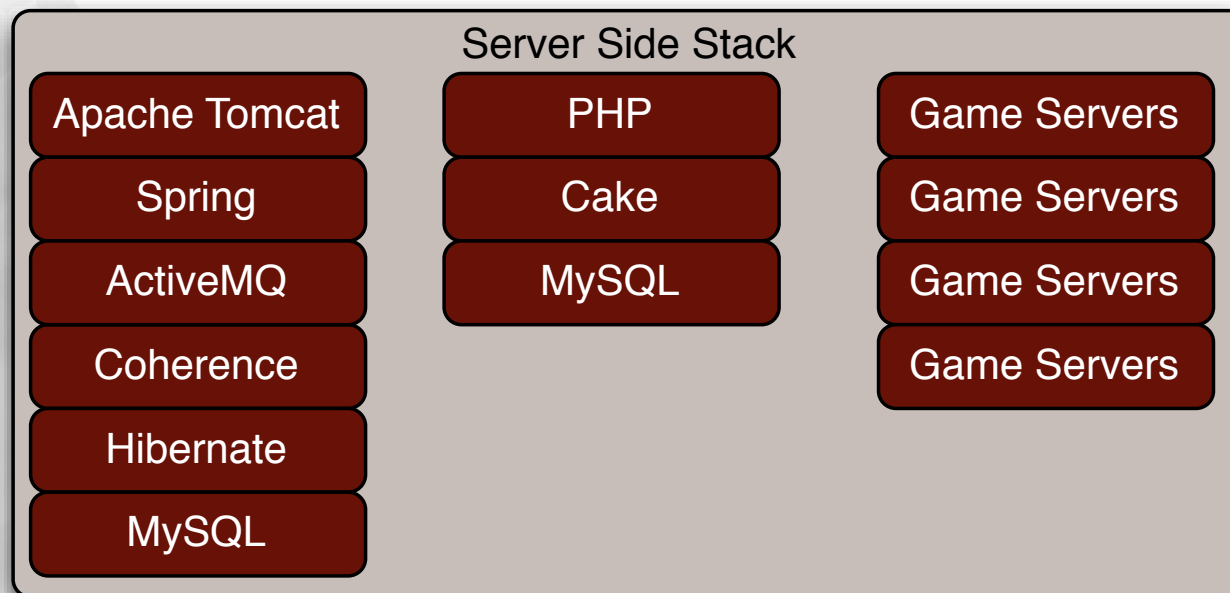
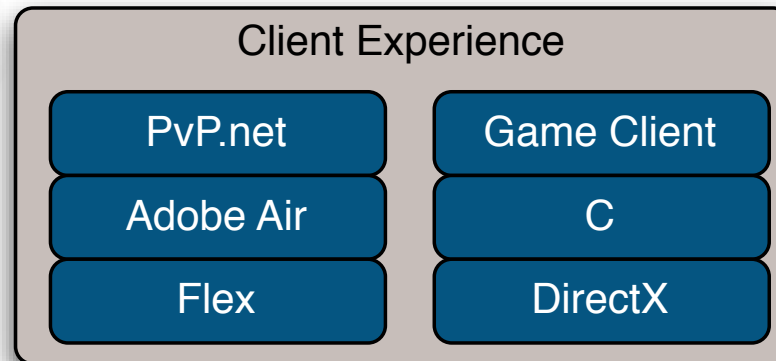


# A Unique Scale Challenge

- Game features do not always support traditional architecture decomposition
  - Social elements require uniform access
  - QOS is “soft” real time
  - Crafting a enjoyable user experience
    - I want to be able to invite friends to a game instantly not 20m later
    - Incremental roll out while technically possible creates negative experiences



# A Quick Architectural Overview



# Today We Are Focusing On...

Client Experience

Server Side Stack

Apache Tomcat

Spring

ActiveMQ

Coherence

Hibernate

MySQL



# A Quick Architectural Overview

Apache Tomcat

Apache Tomcat

Apache Tomcat

Spring

Spring

Spring

Coherence

Coherence

Coherence

Coherence

Hibernate

Hibernate

Hibernate

Hibernate

MySQL

# Coherence in a Nutshell

- Can be considered a NoSQL solution
- 1.0 was released in 2001
- Depending what your definitions are it is a key/value or document oriented data store.
- Data is stored in caches by key



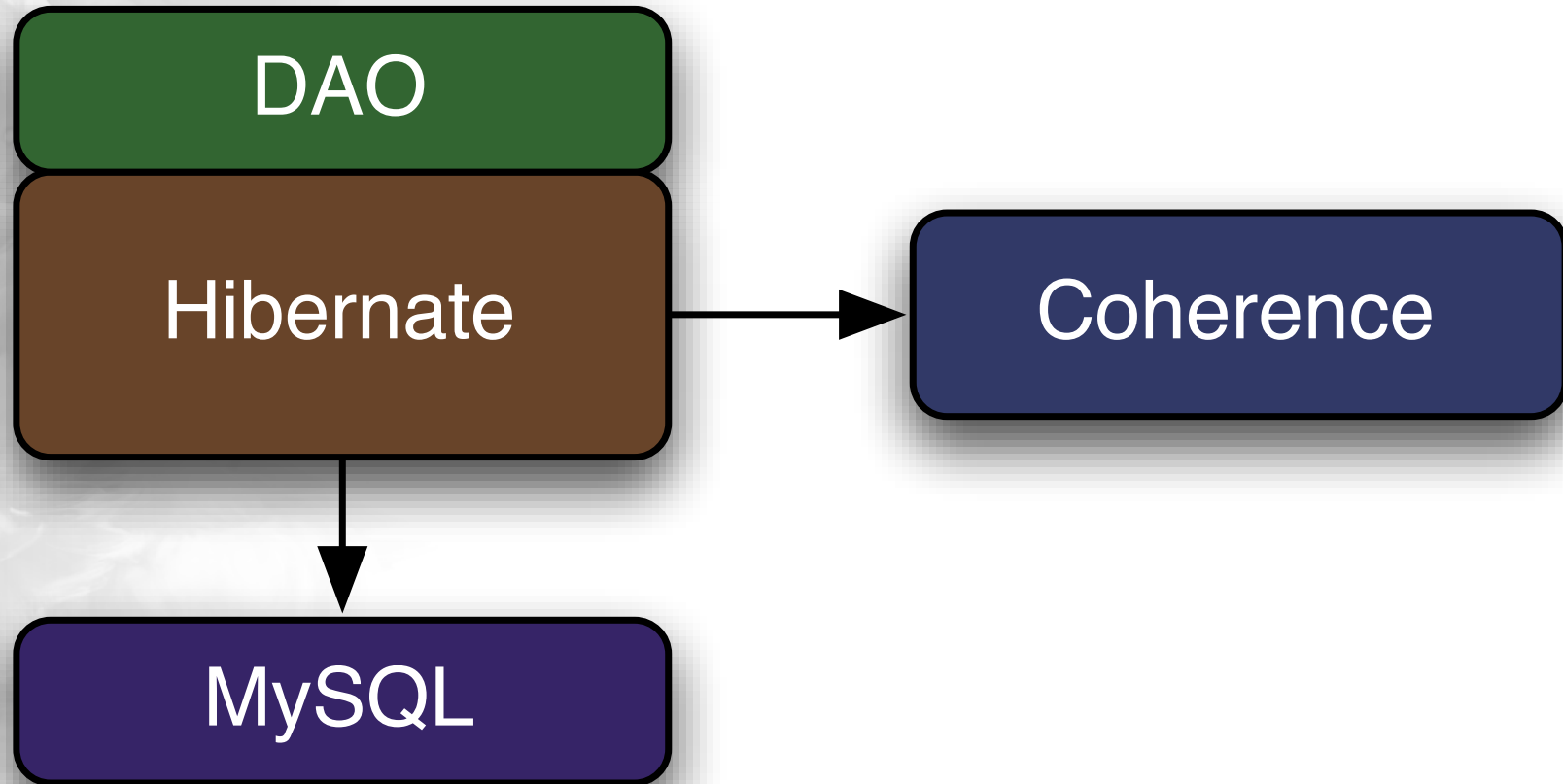


# Coherence in a Nutshell

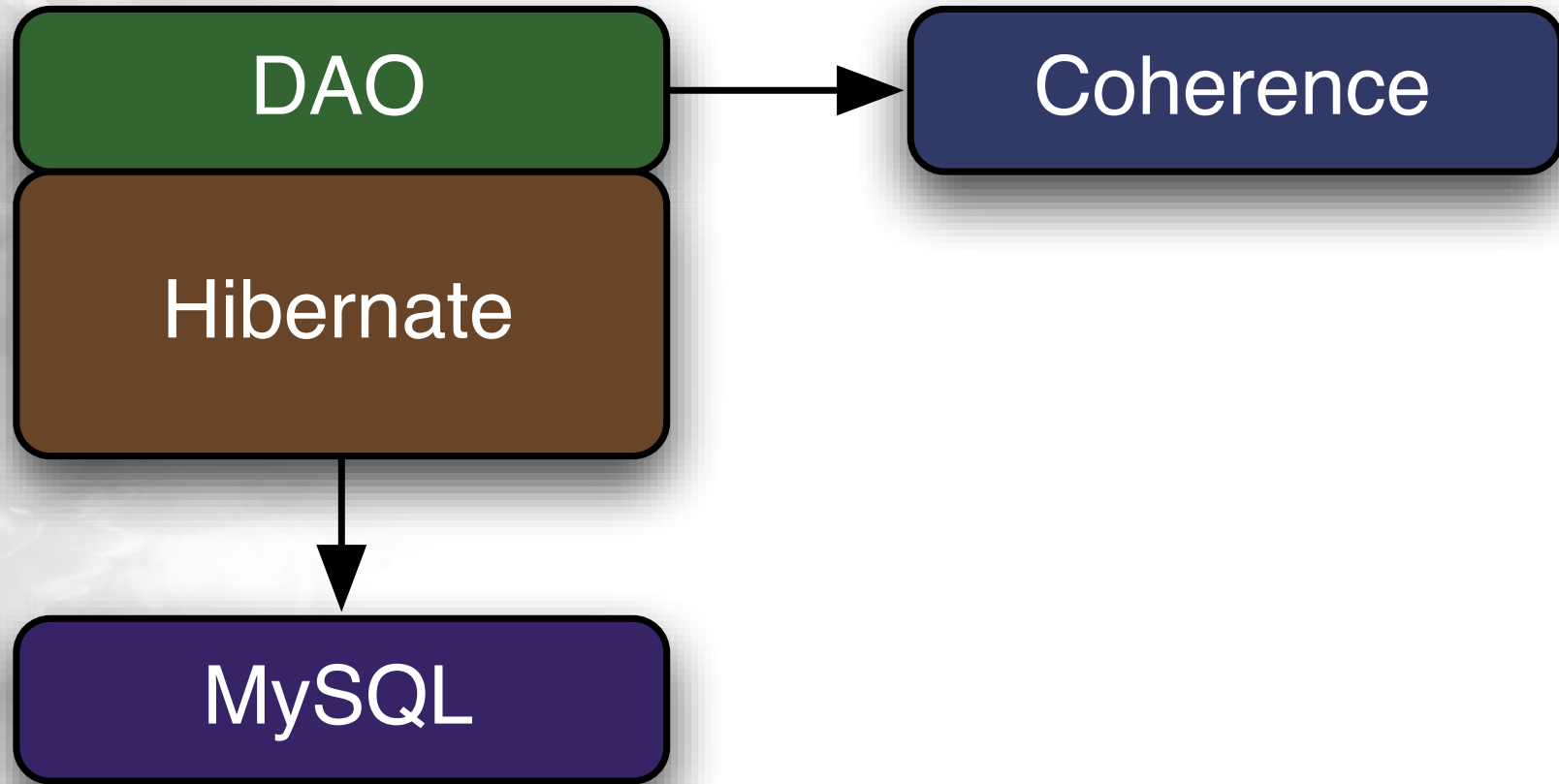
- Clustering
- Caching
- Grid computing
- Dynamically horizontally scalable



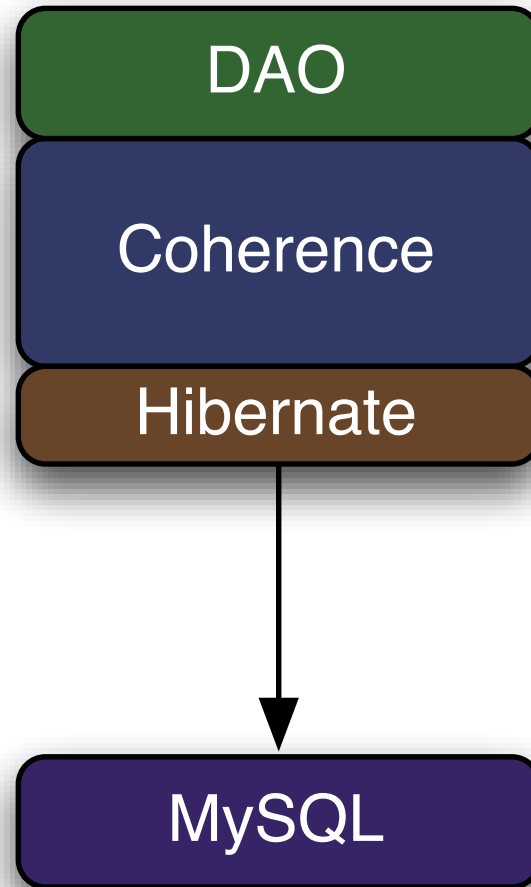
# A Quick Architectural Overview



# A Quick Architectural Overview



# A Quick Architectural Overview



# Coherence in Front

- Supports atomic operations at the key level
- Send the work to the data
  - Work is smaller than data
  - Work is more easily parallelizable
- Root objects /caches essentially define operation and query scope
- Queries becomes distributed merge searches
  - Blazing fast with 100s of cores and memory access



# Functional Uses

- Transient platform data
  - Matchmaking
  - Game Instances
  - Game End Statistics
- Caching provides flexibility
  - Failover
  - Distribution of workload
- Example cache usage - Tomcat control logic allocating users to game servers



# Agenda

- Simple is Best
- Don't Over Use Your Toolbox
- Scaling Best Practices Have Consequences
- Monitor Everything
- Code a Dynamic System







# Simple is Best

Everyone thinks they have a difficult problem  
that needs a difficult solution.



# Simple is Best

- Overheard at an engineering meeting...

“There is no way one box can handle the load for this new feature. We need caching, 4 boxes, and lots of threads to implement it right!”



# Simple is Best

- Complex solutions are hard to get right and often buggy.
- Has anyone tried the simple approach?
  - Modern CPUs are fast
    - 3 billion instructions a second
  - Memory is fast
  - Java is fast
  - Not everything needs NIO and the latest open source library



# Simple is Best

- Step 1 – Don't Over Design
  - Write a basic implementation
  - Test with real world data
  - Assert performance
    - This is not the same as premature optimization
  - Is it fast enough for today and +6 months?
  - Will it be faster in production on a 16 core box with lots of memory and a fast network?
  - If not go to step 2

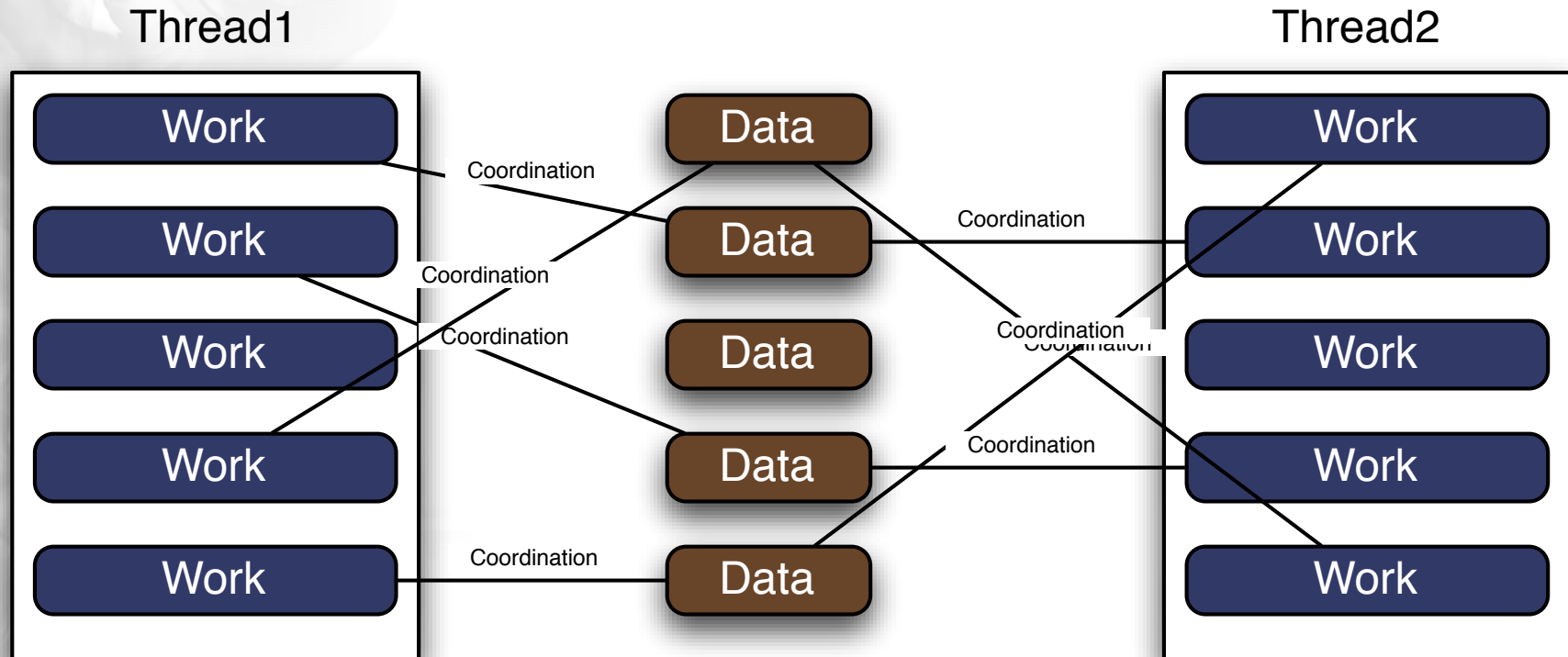


# Simple is Best

- Step 2 – Rig The Game
  - Coordination is hard and expensive.
  - Can you partition the algorithm before hand?
  - It is easier to divide the inputs of an algorithm and then parallel process than it is to continually coordinate while processing
    - Concurrency becomes infrastructure
    - Algorithm goes back to single threaded



# Simple is Best



# Simple is Best

Data

Data

Data

Data

Data

Thread1



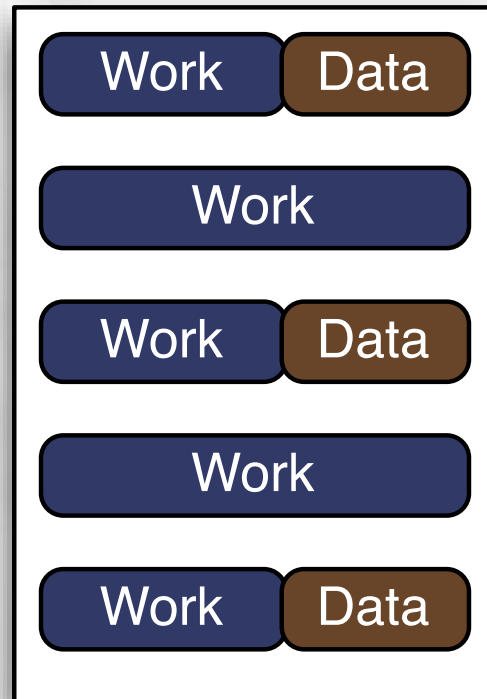
Thread2



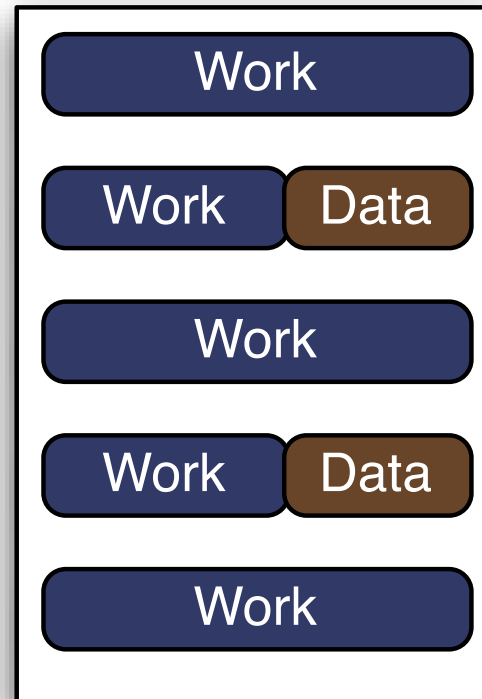


# Simple is Best

Thread1



Thread2



# Agenda

- Simple is Best
- Don't Over Use Your Toolbox
- Scaling Best Practices Have Consequences
- Monitor Everything
- Code a Dynamic System





# Don't Over Use Your Toolbox

If you have a hammer everything is a nail.

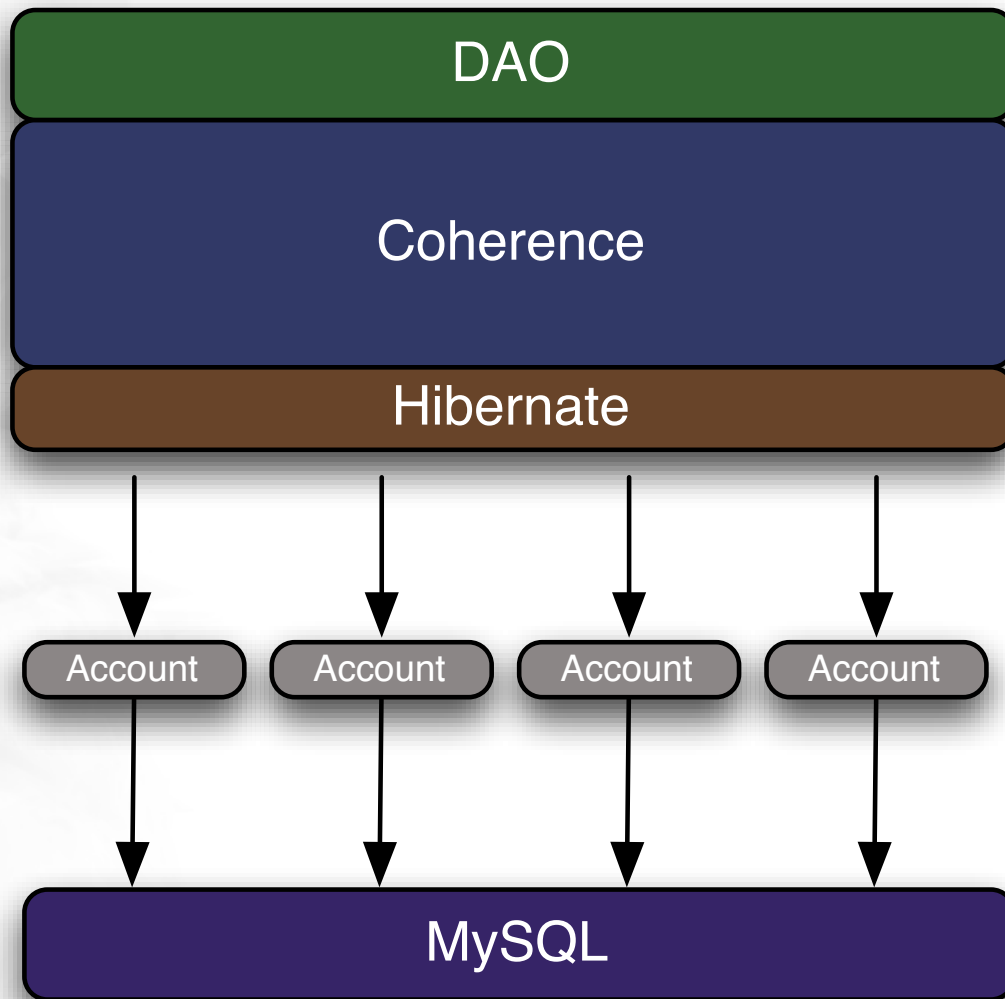


# Don't Over Use Your Toolbox

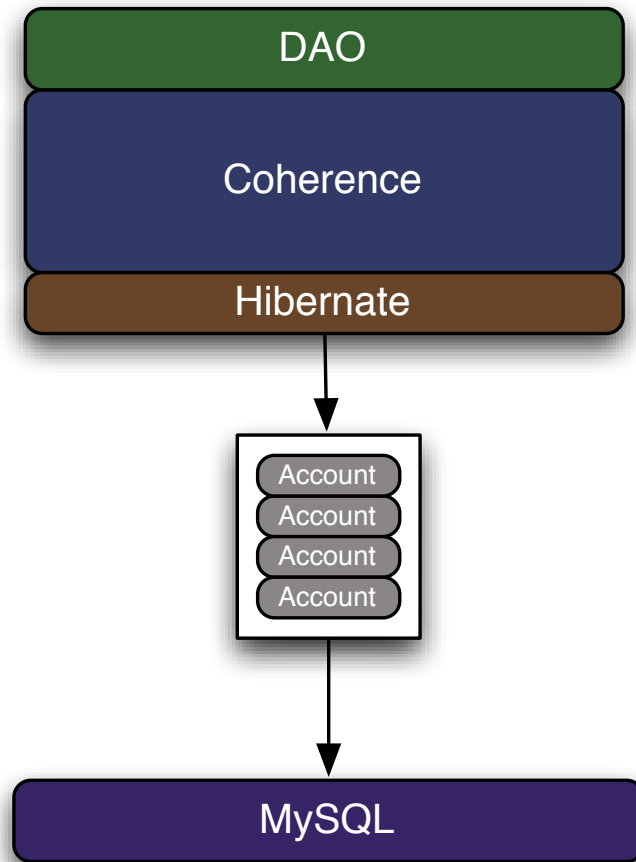
- Coherence caches support write behind
- Reduces db load
- Uniqueness is enforced by key.
- Lets look at an example...



# Don't Over Use Your Toolbox



# Don't Over Use Your Toolbox





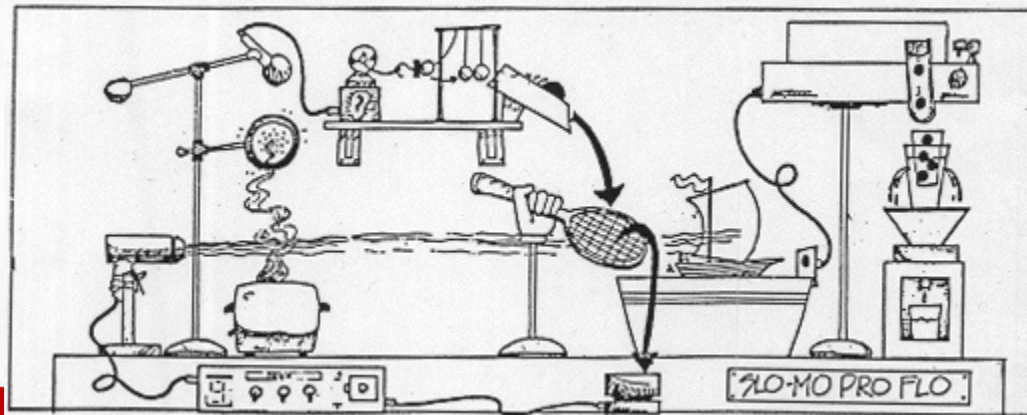
# Don't Over Use Your Toolbox

- We cache accounts by id
- We need to enforce unique user names
- Problem: user names aren't our key
- Let's query the db in our application logic!
- Wait a minute isn't there a race condition with that write behind thing?
- Hmm...



# Don't Over Use Your Toolbox

- What can we do?
  - Cache all accounts
  - Add a distributed semaphore to handle the race
  - Query DB from Coherence storage component
  - Add and use a cross-reference cache
  - Create a Rube Goldberg machine

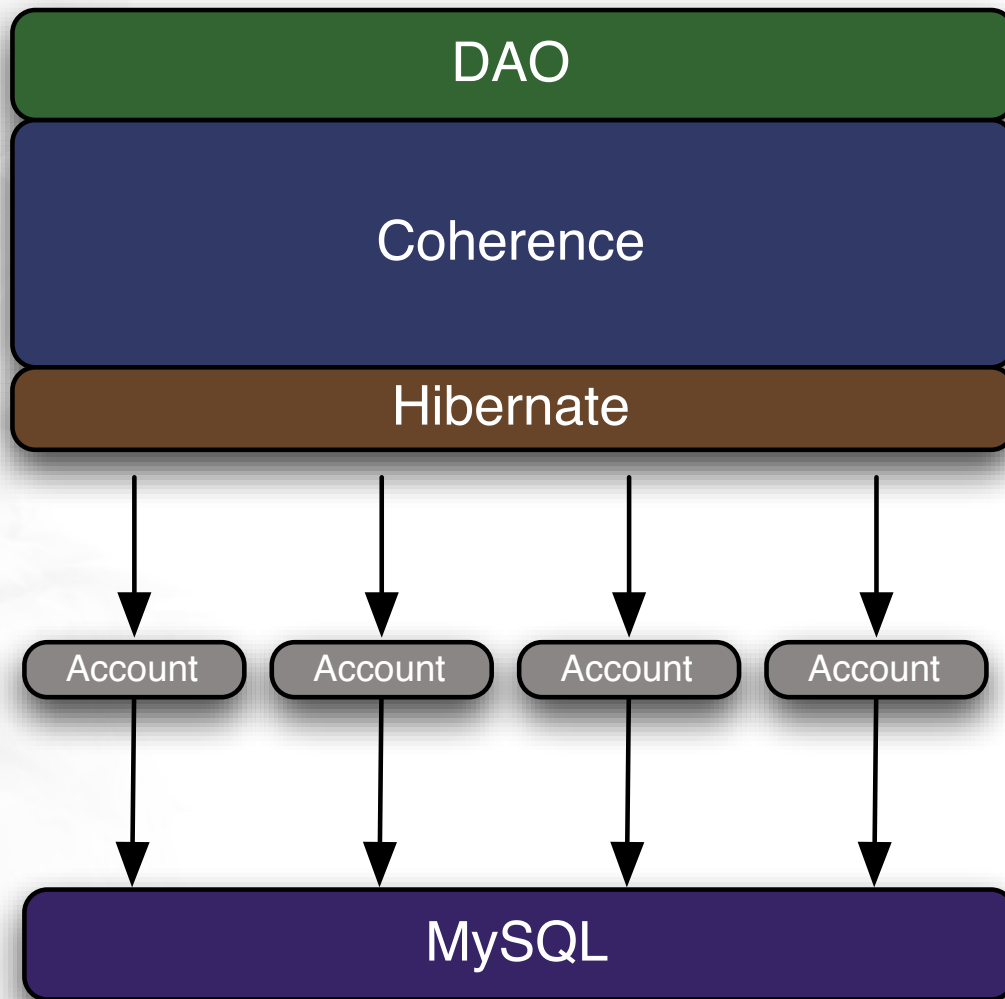


# Solution – Remove Write Behind

- There were many ways to solve our account uniqueness issue.
- Many involved custom Coherence additions
- Easiest solution: write through when putting account by id, let DB enforce uniqueness
- Account creation is relatively infrequent, so write through latency is acceptable



# Don't Over Use Your Toolbox



# Agenda

- Simple is Best
- Don't Over Use Your Toolbox
- Scaling Best Practices Have Consequences
- Monitor Everything
- Code a Dynamic System





# Scaling Best Practices Have Consequences





# Modern Scaling Technology 101

1. Scaling is hard.
2. Lets get rid of some things so we can do this easier.
3. What do we get rid of? I can't decide.
4. Plan B ... instead of telling you what you can't do I will tell you what you can.
5. Follow these X rules and everything will be fine.



# Examples

- Map Reduce
  - If all problems can be written with a map step and a reduce step...
- NoSQL
  - I am taking away your joins...
- CAP
  - Pick Two...



# Consequences

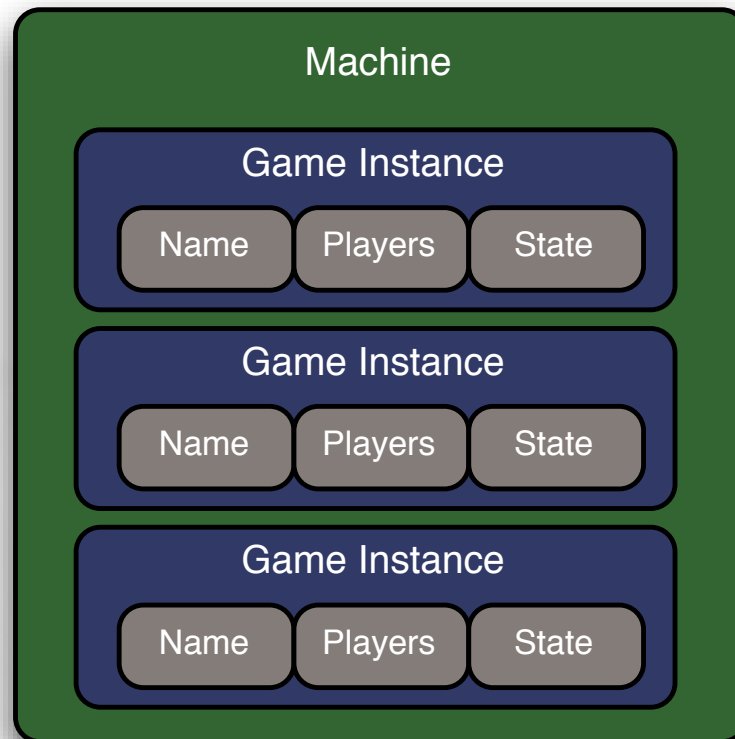
- Atomic operations often become scoped by entry values and root objects
- Blog Entry
  - Comment
  - Comment
  - Comment
- What happens with a really popular post with 300 comments?

# An Example of a Mismatch

- Riot runs multiple games per server
- A root object represents the server
- As games are allocated child objects are added to this object
- As we've grown these child objects have become more complex
- We also run more games per server than at launch



# Root Objects and Child Objects



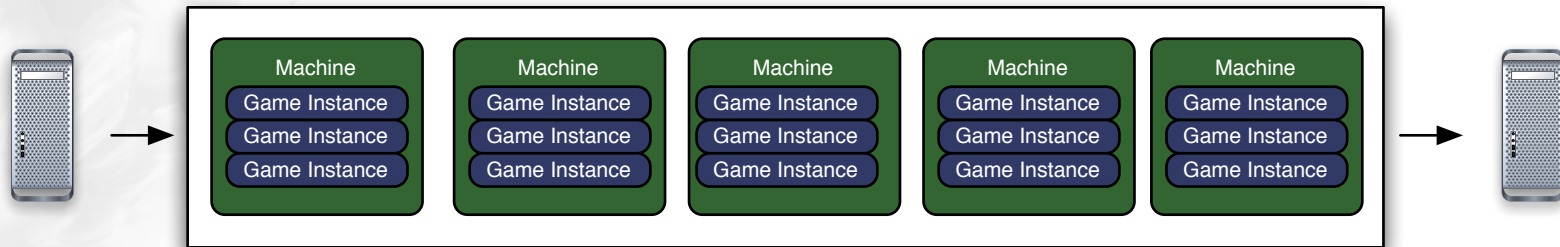
# Evolution of an Anti-Pattern

- A full Machine object has many child objects
- Each approached 2-50k in size
- As a result the Machine object went from <20k to >500k in size
- Network transfer fast became a bounding factor
- Object serialization became a bounding factor

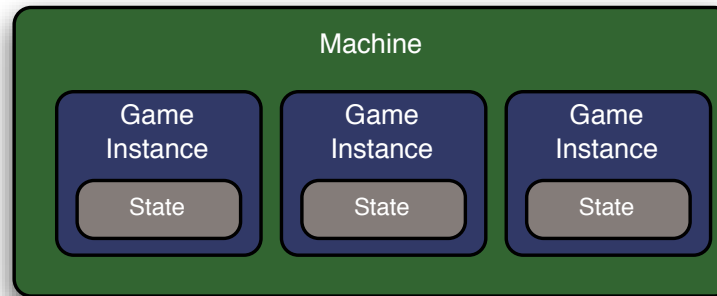
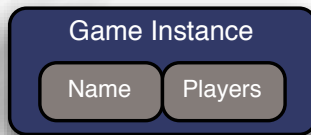




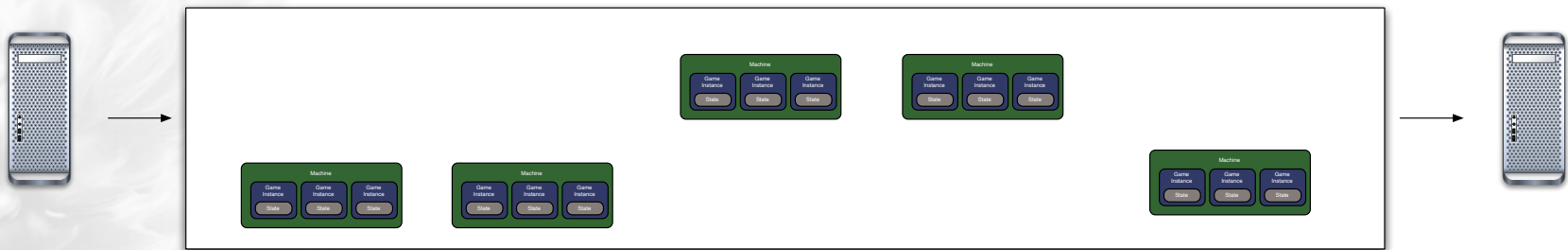
# The Pipe is Full



# Do we really have one object?

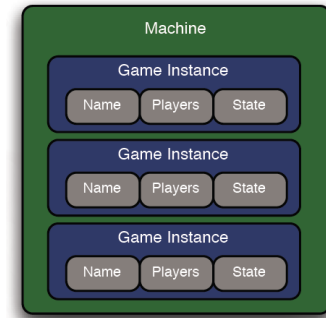


# Smaller is better!



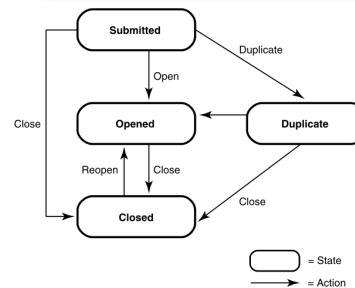
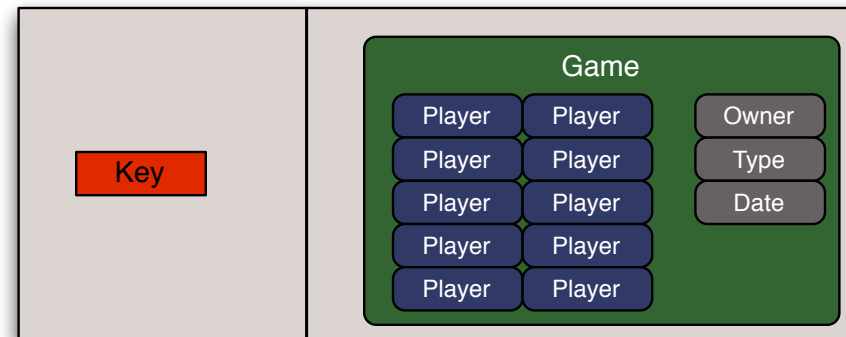
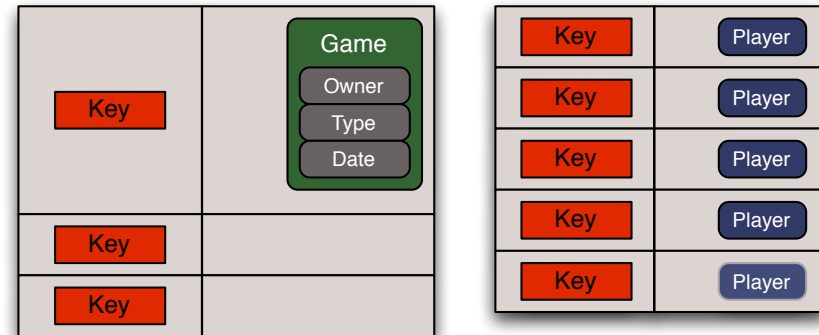
# Domain Driven Design for the Grid

- DDD is based on domain objects
  - Some are Entities (have identifiers)
  - Some Entities are Aggregate Roots
- DDD is mute on serialization (impl. detail)
- Coherence based on Named Cache (Map<K, V>)
- Coherence requires serialization
- What should be serialized together?
- What are the consequences?



# Named Cache Usage Patterns

- Named Cache per Entity Type
- Named Cache per Aggregate Root Type
- Named Cache per Object State



# Usage Patterns and Consequences

- Selected usage pattern has substantial pervasive impact on application codebase
- Related patterns and refactorings
  - Disconnected Domain Model
  - Reference By Identifier
  - Replace Field Access with Cache Get
  - Replace Collection Access with Cache Query
  - Indexed Queries

# Agenda

- Simple is Best
- Don't Over Use Your Toolbox
- Scaling Best Practices Have Consequences
- Monitor Everything
- Code a Dynamic System







# Monitor Everything

A picture is worth a thousand words.



# Monitor Everything

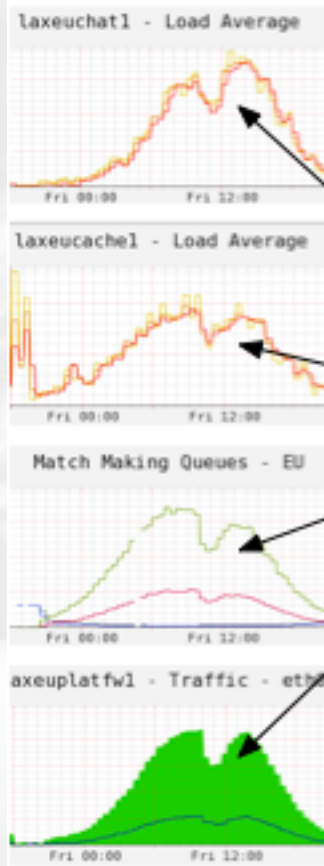
- 1 box = predictable
- N boxes = organic
  - In a large system everything interacts
    - Network
    - CPU load
    - Hard disk failure
    - Player behavior
  - The trick is understanding the average case and when something changes

# Monitor Everything

- Graph everything.
- Pictures are easier to understand than logs with millions of operations per day.
- Even with grep you will often not be able to find many trends.



# Monitor Everything



- What happened here?
- Networking issue!

# Monitor Everything

- Automate metrics gathering
- Spring performance monitoring interceptor
  - Log out call stack on external calls
  - Sample internal calls
  - Automate reporting
- Isn't that slow?
  - Less than 1% overhead.
- Trivial cost vs the benefit



# Monitor Everything

- Data is useless without an easy way to view it.

Top 50 EXTERNAL calls by volume

Service	Method	Current Import					Previous Import			
		Num Calls	Avg Call Time	% of Total Calls	Baseline Factor	% diff	Previous Num Calls	Previous Avg Call Time	Previous % of Total Calls	Previous Baseline Factor
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	3465105	27	7.3648%	2.6471	-4.5827%	3572459	24	7.5898%	2.7742
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	3090954	7	6.5695%	2.3612	-3.6516%	3155920	7	6.7049%	2.4507
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	2406357	0	5.1145%	1.8383	+3.3200%	2291151	0	4.8676%	1.7792
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	2406357	7	5.1145%	1.8383	+3.3200%	2291151	7	4.8676%	1.7792
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	2138788	2	4.5458%	1.6339	+0.1456%	2100940	2	4.4635%	1.6315
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	2031441	24	4.3176%	1.5519	-4.7465%	2097979	24	4.4572%	1.6292
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	1552590	0	3.2999%	1.1861	-4.3052%	1596050	0	3.3909%	1.2394
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	1493027	6	3.1733%	1.1406	-4.4092%	1536489	6	3.2643%	1.1932
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	1358414	20	2.8872%	1.0377	-4.0802%	1393162	20	2.9598%	1.0819
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	1309035	0	2.7822%	1.0000	0.0000%	1287743	0	2.7359%	1.0000
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	1106124	23	2.3510%	0.8450	-0.1746%	1090036	22	2.3158%	0.8465
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	1100421	7	2.3388%	0.8406	+1.4989%	1066536	7	2.2659%	0.8282
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	1073141	17	2.2809%	0.8198	-4.1639%	1101554	17	2.3403%	0.8554
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	1072878	3	2.2803%	0.8196	-4.1650%	1101296	3	2.3397%	0.8552
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	1069397	1	2.2729%	0.8169	+0.1457%	1050472	1	2.2318%	0.8157
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	1069391	12	2.2729%	0.8169	+0.1455%	1050468	12	2.2318%	0.8157
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	1069390	0	2.2729%	0.8169	+0.1454%	1050469	0	2.2318%	0.8157
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	1059574	2	2.2520%	0.8094	-3.3607%	1078588	2	2.2915%	0.8376
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	893122	61	1.8983%	0.6823	-2.4315%	900490	59	1.9131%	0.6993
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	846345	73	1.7988%	0.6465	+1.2110%	822617	68	1.7477%	0.6388
com.rigence.portal.gam...Service	com.rigence.portal.gam...Service	776295	1	1.6499%	0.5930	-4.3052%	798025	1	1.6954%	0.6197



# Monitor Everything

- Data is useless without an easy way to view it.
- ...lets grep to research the red item...

Top 50 EXTERNAL calls by volume

Service	Method	Current Import					Previous Import			
		Num Calls	Avg Call Time	% of Total Calls	Baseline Factor	% diff	Previous Num Calls	Previous Avg Call Time	Previous % of Total Calls	Previous Baseline Factor
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	3465105	27	7.3648%	2.6471	-4.5827%	3572459	24	7.5898%	2.7742
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	3090954	7	6.5695%	2.3612	-3.6516%	3155920	7	6.7049%	2.4507
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	2406357	0	5.1145%	1.8383	+3.3200%	2291151	0	4.8676%	1.7792
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	2406357	7	5.1145%	1.8383	+3.3200%	2291151	7	4.8676%	1.7792
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	2138788	2	4.5458%	1.6339	+0.1456%	2100940	2	4.4635%	1.6315
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	2031441	24	4.3176%	1.5519	-4.7465%	2097979	24	4.4572%	1.6292
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	1552590	0	3.2999%	1.1861	-4.3052%	1596050	0	3.3909%	1.2394
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	1493027	6	3.1733%	1.1406	-4.4092%	1536489	6	3.2643%	1.1932
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	1358414	20	2.8872%	1.0377	-4.0802%	1393162	20	2.9598%	1.0819
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	1309035	0	2.7822%	1.0000	0.0000%	1287743	0	2.7359%	1.0000
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	1106124	23	2.3510%	0.8450	-0.1746%	1090036	22	2.3158%	0.8465
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	1100421	7	2.3388%	0.8406	+1.4989%	1066536	7	2.2659%	0.8282
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	1073141	17	2.2809%	0.8198	-4.1639%	1101554	17	2.3403%	0.8554
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	1072878	3	2.2803%	0.8196	-4.1650%	1101296	3	2.3397%	0.8552
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	1069397	1	2.2729%	0.8169	+0.1457%	1050472	1	2.2318%	0.8157
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	1069391	12	2.2729%	0.8169	+0.1455%	1050468	12	2.2318%	0.8157
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	1069390	0	2.2729%	0.8169	+0.1454%	1050469	0	2.2318%	0.8157
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	1059574	2	2.2520%	0.8094	-3.3607%	1078588	2	2.2915%	0.8376
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	893122	61	1.8983%	0.6823	-2.4315%	900490	59	1.9131%	0.6993
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	846345	73	1.7988%	0.6465	+1.2110%	822617	68	1.7477%	0.6388
com.hogwarts.zellion.gems.ServicerService	com.hogwarts.zellion.gems.ServicerService	776295	1	1.6499%	0.5930	-4.3052%	798025	1	1.6954%	0.6197





# Monitor Everything

- Automate the next 5 questions
- Why should they be manual?

[0, 100)	542603
[100, 200)	90834
[200, 300)	15576
[300, 400)	3176
[400, 500)	642
[500, 600)	154
[600, 700)	48
[700, 800)	16
[800, 900)	15
[900, 1000)	5
[1000, 1100)	1
[1100, 1200)	1
[1200, 1300)	0
[1300, 1400)	0
[1400, 1500)	0
[1500, 1600)	2
[1600, 1700)	0
[1700, 1800)	6
[1800, 1900)	45
[1900, 2000)	57
[2000,)	113

# Agenda

- Simple is Best
- Don't Over Use Your Toolbox
- Scaling Best Practices Have Consequences
- Monitor Everything
- Code a Dynamic System





# Code a Dynamic System

We all think we have planned for everything. Often we are wrong. At this point operational flexibility becomes useful.



# Code a Dynamic System

- A large system will change while it is running
  - Spikes in Users
  - Hardware failures
  - New user behavior
- The next release or during a downtime are not viable strategies for a response



# Code a Dynamic System

- Design features so they can be turned off
  - Most things can be set to OFF if you plan for the use case
- Design algorithms that can be adjusted on the fly

# Code a Dynamic System

- Choose technologies that have elastic properties
  - Dynamic cluster recomposition
  - Stateless growth patterns
- Not every piece of your stack has to be elastic
  - You are only as fast as you slowest point.  
However....
  - One or two building blocks can be a huge force multiplier



# Code a Dynamic System

- League of Legends caches all relevant configuration properties.
- Coherence near caches are used to propagate changes to nodes dynamically.
- It is preferred that algorithms are written so they are aware that their variables may change while running.
  - This can be scoped in seconds or minutes
  - This can often be done without complexity





# Code a Dynamic System

- Thread pools are dynamically configurable
- Machines can assume new roles on the fly
- Algorithms can dynamically shard based on volume





# Don't Build a System Based on Hope

Certainty is a far better principle to build a business on.



# Don't Build a System Based on Hope

- <http://www.quora.com/Is-Cassandra-to-blame-for-Digg-v4s-technical-failures>
- " Really it comes down to the fact that we should have load tested better."

# Don't Build a System Based on Hope

- Load testing is an extremely valuable development step
- There is no substitute for reality
- Continually refine the model of testing
  - Matchmaking
  - Game Starts
- Found countless bottlenecks and fixed them before production



# Don't Build a System Based on Hope

- Able to upgrade stack components with confidence
  - Spring
  - Coherence
  - ActiveMQ
- Instant regression analysis



# Don't Build a System Based on Hope

- How do we test?
  - Started with Jmeter
  - Not flexible enough for our needs
  - Developed custom load testing solution
  - Use EC2 extensively
  - Run 1000s of threads per machine
    - Easier programming model for simulating users



# Don't Build a System Based on Hope

- Load test using realistic servers both in speed and quantity.
  - Don't use your desktop
  - Scale testing environment has >50 machines
- Gathering load test data is as important as production data
  - Logs are worthless with thousands of simulated clients
  - There will be failures, the question is what %



# Don't Build a System Based on Hope

- Load tests are tricky
  - Is your network reliable
    - I'm talking about you EC2...
  - Are you being fair?
    - Test too light and bad things happen
    - Test too hard and you are fixing problems that will never occur
  - Load testing is a partnership between production, engineers, and forecasting







# Obligatory Plug

<http://www.riotgames.com/careers>



# Join the Riot!

- Java Engineers
  - Groovy, Grails, Scale, NoSQL Wizards
- Operations
- Data Warehouse
  - Hadoop Ninjas
- MySQL DBAs





# Obligatory Plug

<http://www.riotgames.com/careers>

