SCALING

TO MILLIONS OF SUMMONERS

GDC 12

SCOTT DELAP
SCALABILITY ARCHITECT
GDC 2012

ABOUT ME – SCOTT DELAP

- Scalability Architect
- Joined Riot in 2008
- About a year before beta
- @scottdelap
- sdelap@riotgames.com

ABOUT RIOT GAMES

FOUNDED
SEPT.2006

500+
EMPLOYEES

OFFICES IN
SANTA MONICA, ST. LOUIS, DUBLIN, SEOUL

OUR MISSION

TO BE THE MOST
PLAYER-FOCUSED
GAME COMPANY
IN THE WORLD.


LEAGUE of LEGENDS

LEAGUE OF LEGENDS: INTRO

July 2011	November 2011
15 MIL REGISTERED	**32.5 MIL** REGISTERED
4 MIL MONTHLY	**11.5 MIL** MONTHLY
1.4 MIL DAILY	**4.2 MIL** DAILY
0.5 MIL PEAK CCU	**1.3 MIL** PEAK CCU
3.7 MIL DAILY HRS	**10.5 MIL** DAILY HRS

A UNIQUE SCALING CHALLENGE

GAME FEATURES
DO NOT ALWAYS SUPPORT
TRADITIONAL DECOMPOSITION

Social elements require uniform access

Crafting an enjoyable user experience

HOW DO WE CREATE A SYSTEM THAT

MEETS THESE NEEDS?

AGENDA

- EMBRACING JAVA AND NoSQL
- SIMPLE IS BEST
- CODE A DYNAMIC SYSTEM
- SCALING BEST PRACTICES
- MONITOR EVERYTHING

PROBLEM #1:

HOW DO WE DEVELOP A SYSTEM **RAPIDLY**…

…WHILE PLANNING FOR **FUTURE CAPACITY NEEDS?**

LEAGUE OF LEGENDS: TECH OVERVIEW

CLIENT EXPERIENCE

PvP.net | Adobe Air | Flex | Game Client | C | DirectX

SERVER SIDE STACK

Apache Tomcat	PHP	Game Servers
Spring	Cake	Game Servers
ActiveMQ	MySQL	Game Servers
Coherence		Game Servers
Hibernate		
MySQL		

TODAY'S FOCUS

CLIENT EXPERIENCE

PvP.net | Adobe Air | Flex | Game Client | C | DirectX

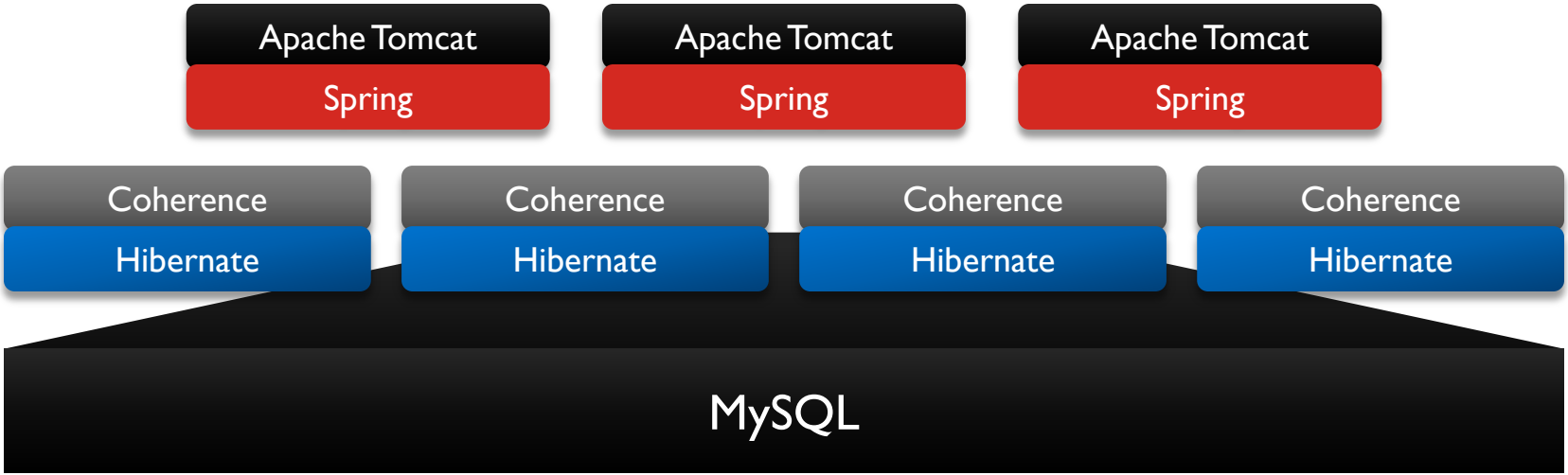
SERVER SIDE STACK

- Apache Tomcat
- Spring
- ActiveMQ
- Coherence
- Hibernate
- MySQL

- PHP
- Cake
- MySQL

- Game Servers
- Game Servers
- Game Servers
- Game Servers

A TECH STACK WITH NEW AND OLD ELEMENTS



BENEFITS OF TRADITIONAL JAVA

- ☑ MATURE OPEN SOURCE ECOSYSTEM
- ☑ ESTABLISHED TOOLS
- ☑ LARGE POOL OF TALENTED DEVELOPERS

ACCELERATING THE FOUNDATION WITH NoSQL

DATA STORED IN CACHES BY KEY

NUMEROUS USES

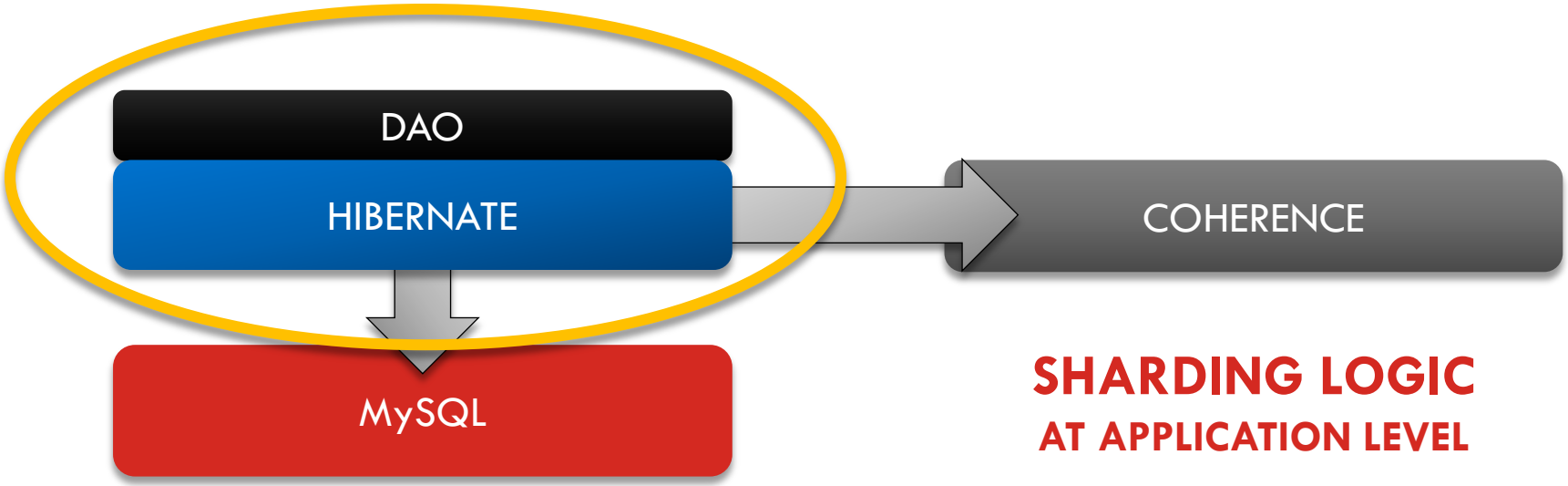
PROVIDES ELASTICITY

NoSQL ENABLING RAPID GROWTH

1. Horizontal scaling of Coherence greatly simplified absorbing CCU growth over time

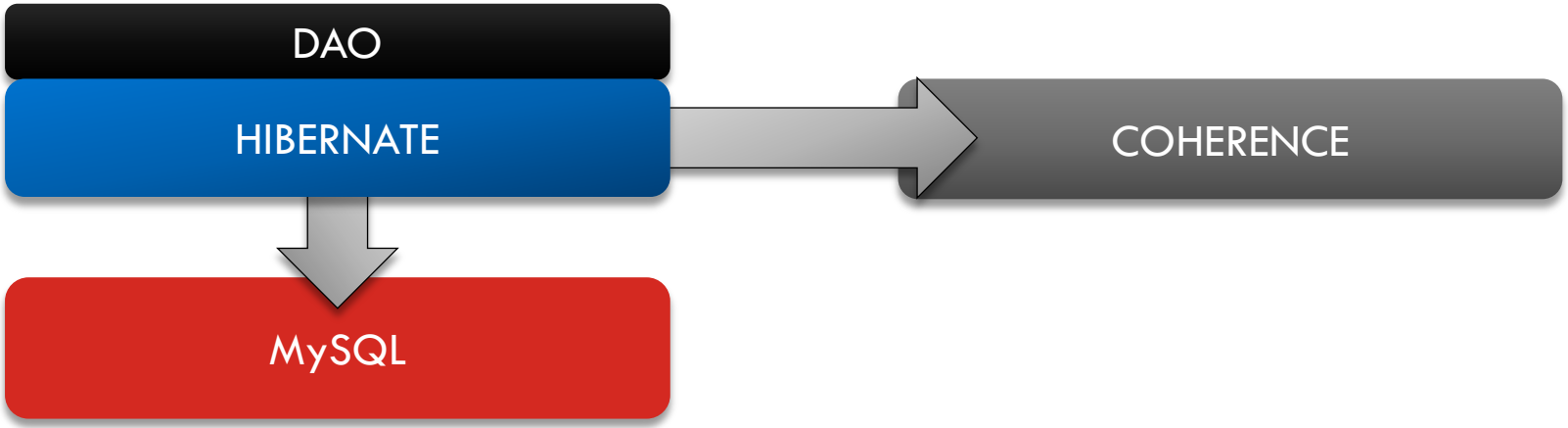
2. Design patterns enforced by Coherence promoted feature level scaling as well

CACHING IN DETAIL

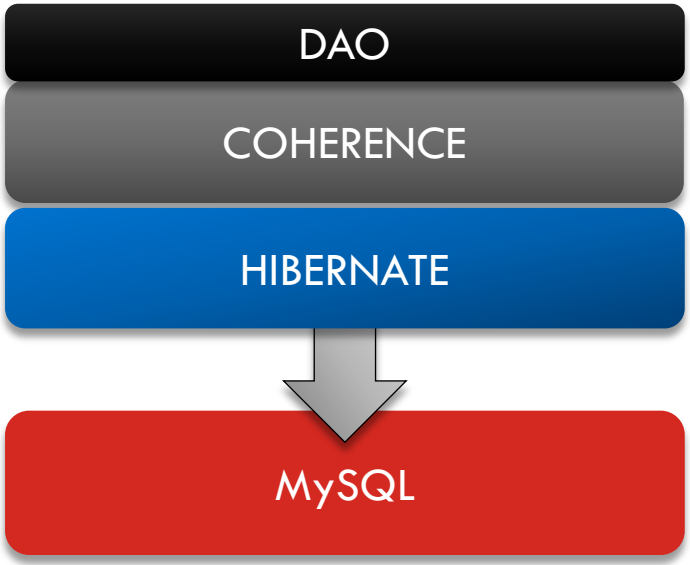


SHARDING LOGIC
AT APPLICATION LEVEL

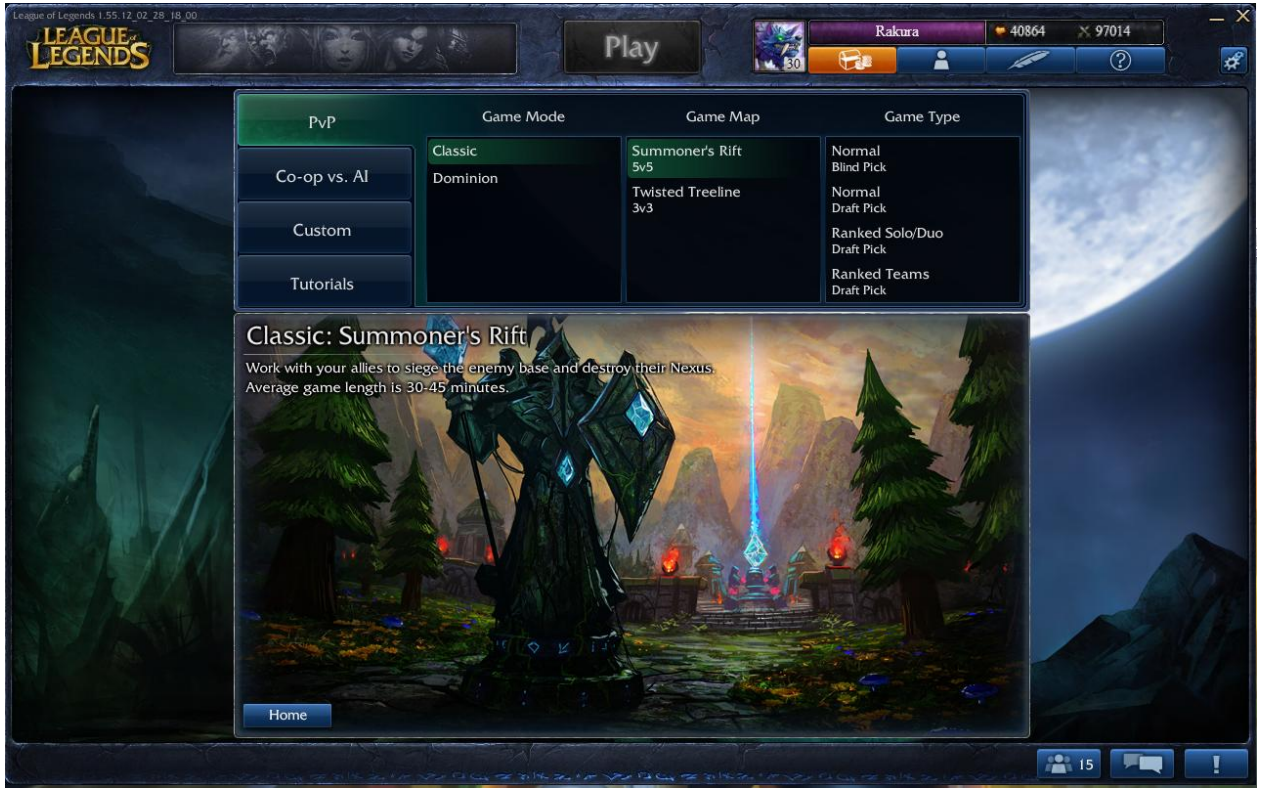
EMBRACING CACHE ADVANTAGES



EMBRACING CACHE ADVANTAGES



LEVERAGING ADVANTAGES



GRID COMPUTING

TRANSPARENT PARTITIONING

AGENDA

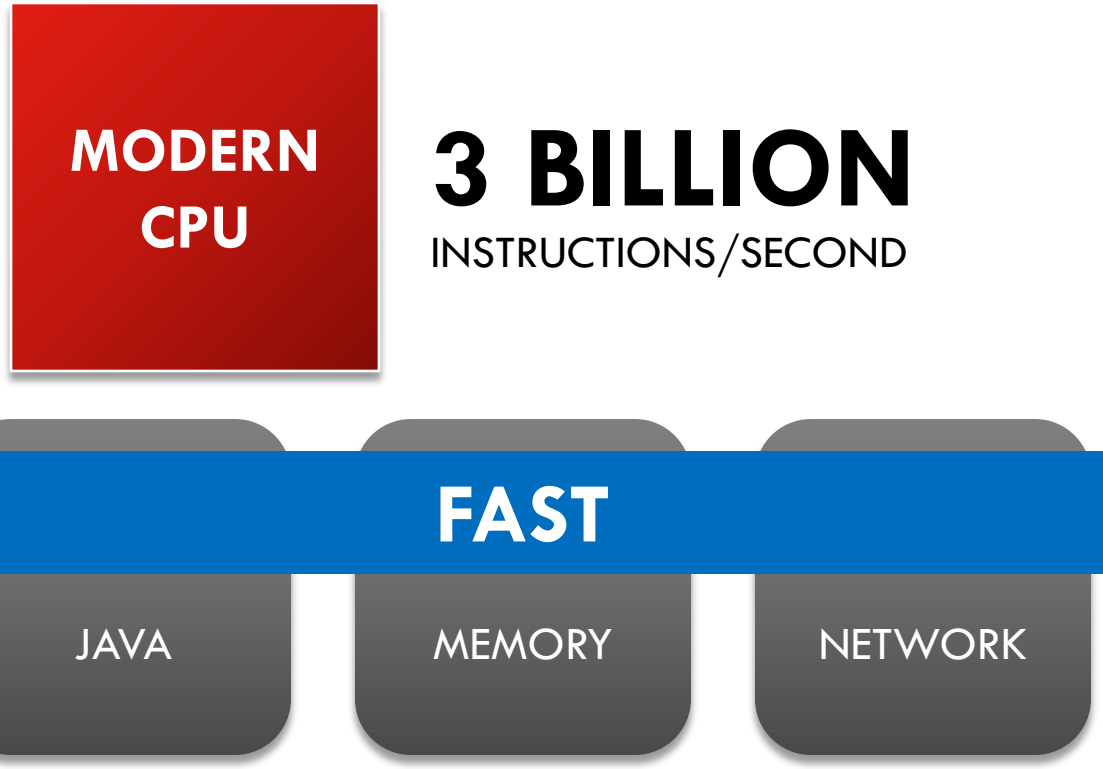
- EMBRACING JAVA AND NoSQL
- SIMPLE IS BEST
- CODE A DYNAMIC SYSTEM
- SCALING BEST PRACTICES
- MONITOR EVERYTHING

PROBLEM #2:

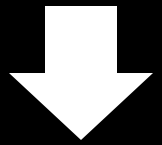
HOW DO WE QUICKLY
DEVELOP **NEW FEATURES**…

…WHILE **LIMITING BUGS?**

SIMPLE IS BEST



Complexity is the enemy of quality



DON'T OVER DESIGN

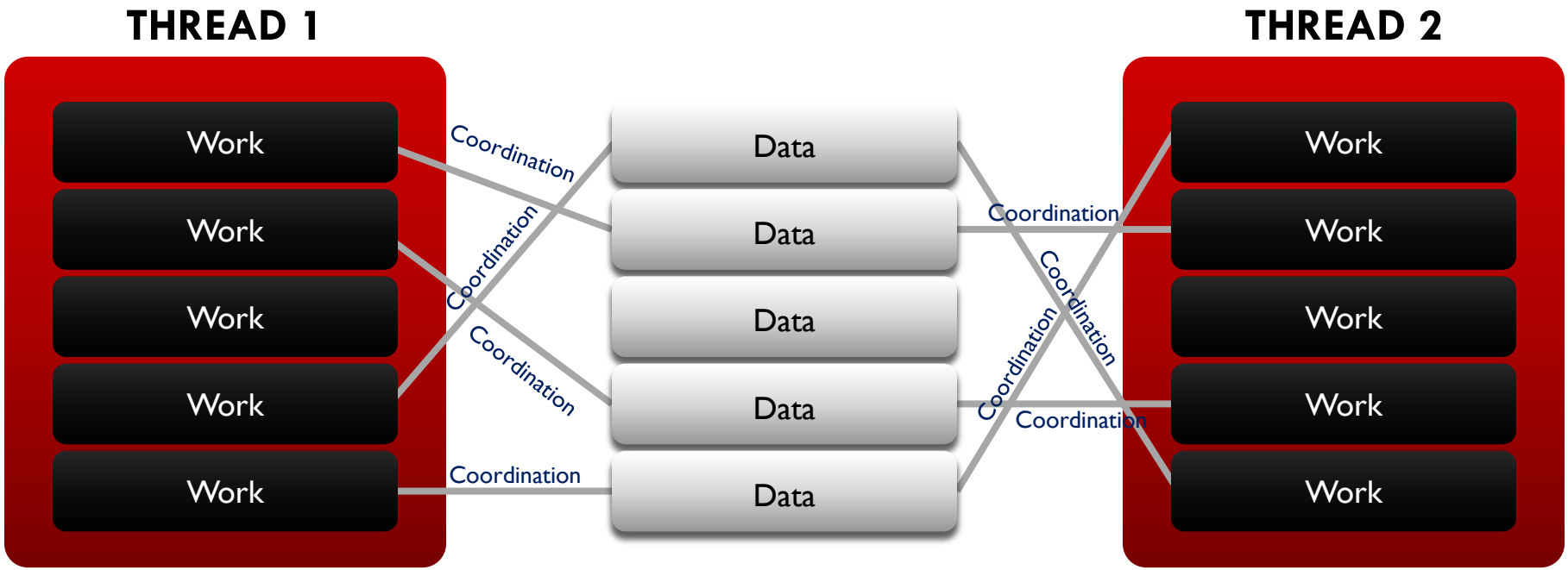
RIG THE GAME

Divide inputs of algorithm, then parallel process

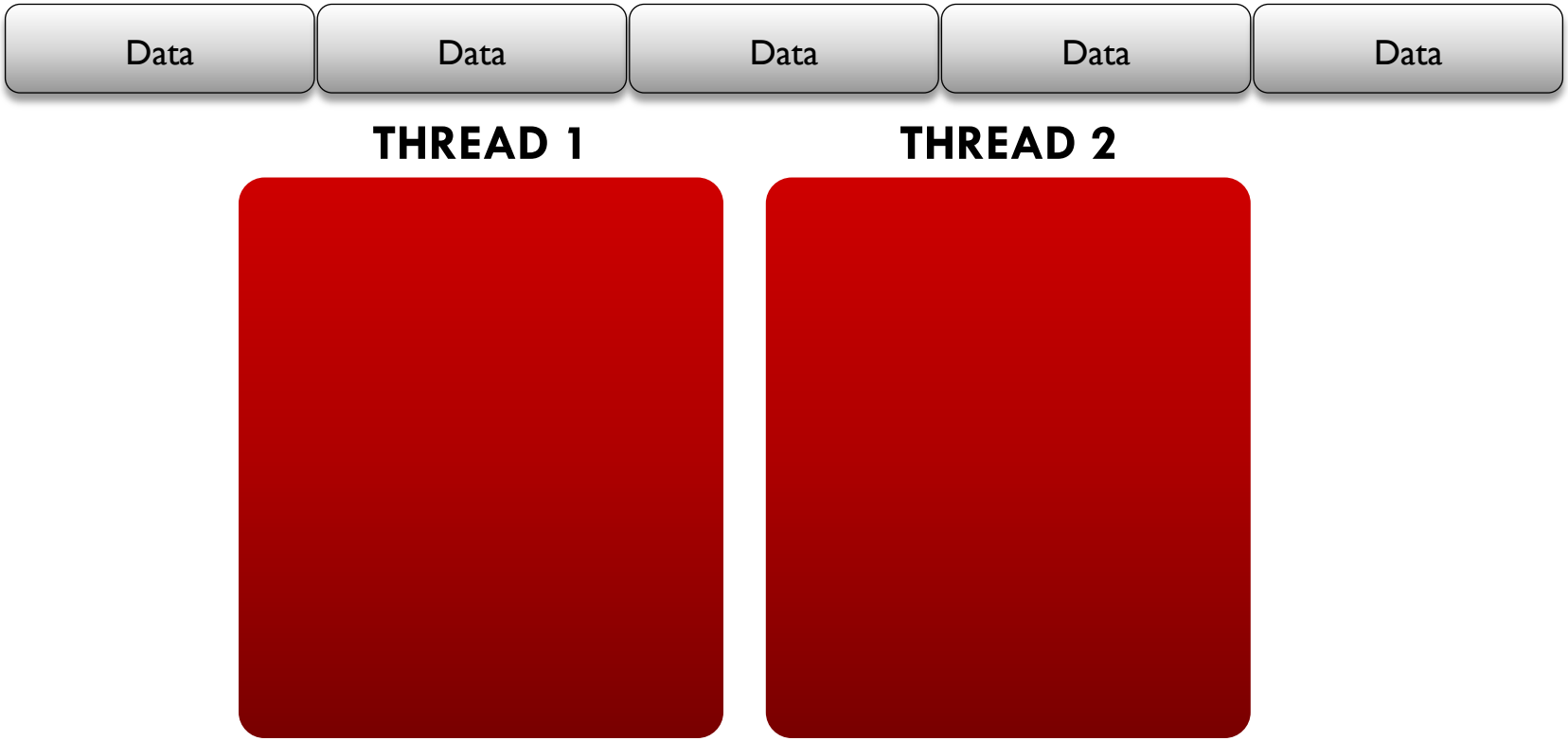
EASIER

Continually coordinate

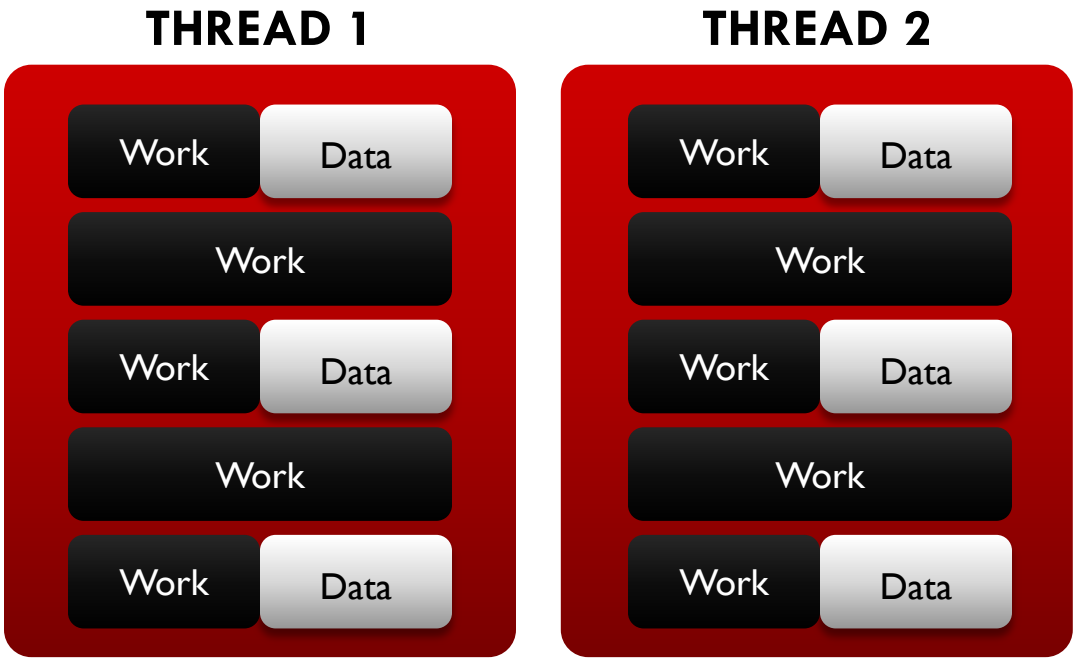
RIG THE GAME


THREAD 1
THREAD 2
Work
Work
Work
Work
Work
Work
Work
Work
Work
Data
Data
Data
Data
Data
Coordination
Coordination
Coordination
Coordination
Coordination
Coordination
Coordination
Coordination
Coordination

RIG THE GAME



RIG THE GAME


THREAD 1
Work
Data
Work
Work
Data
Work
Work
Data
THREAD 2
Work
Data
Work
Work
Data
Work
Work
Data

AGENDA

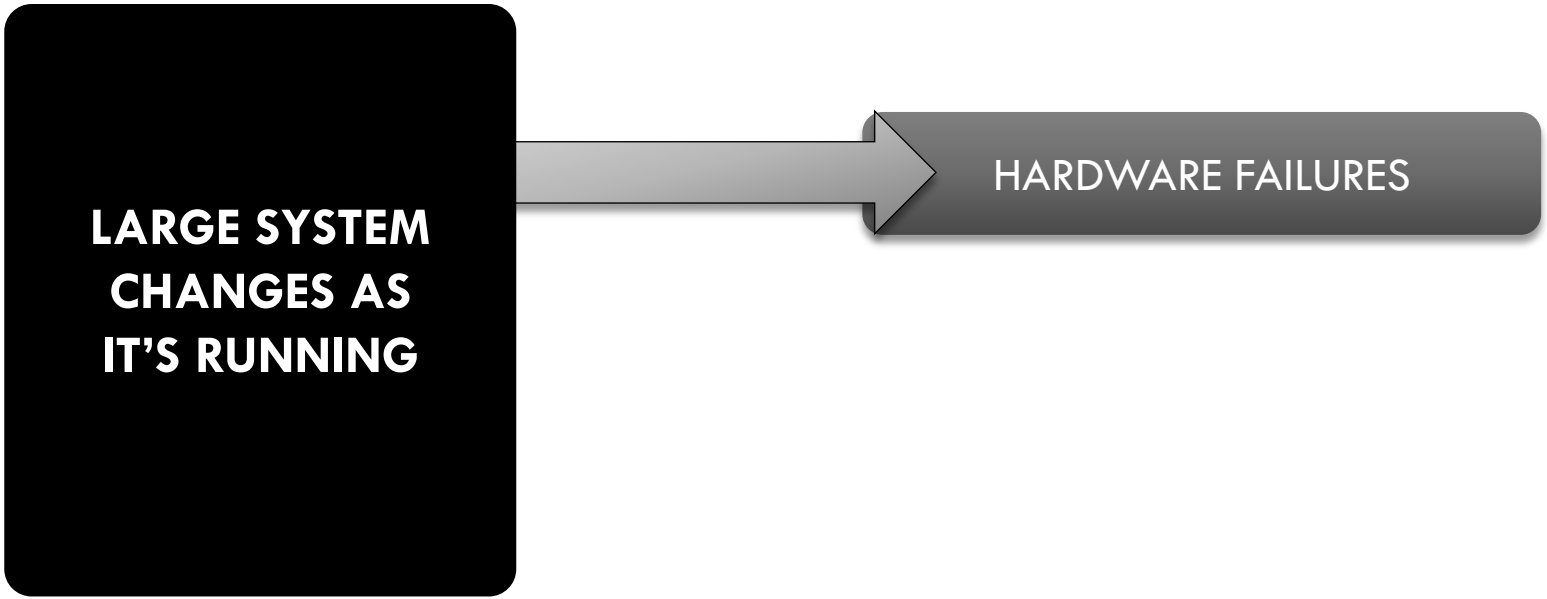
- EMBRACING JAVA AND NoSQL
- SIMPLE IS BEST
- CODE A DYNAMIC SYSTEM
- SCALING BEST PRACTICES
- MONITOR EVERYTHING

PROBLEM #3:

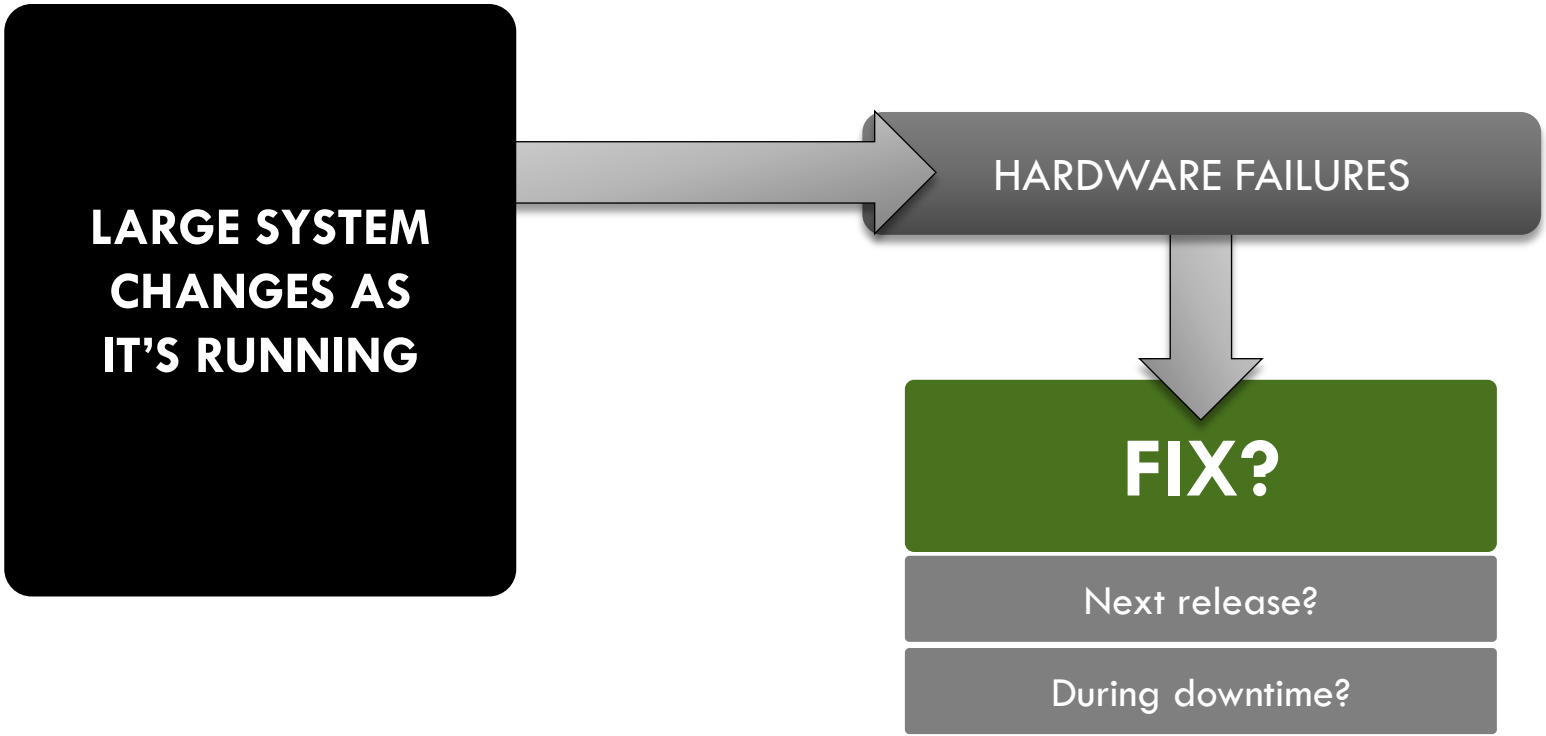
HOW DO WE HANDLE NOT JUST **MONTHLY CHANGE**…

…BUT **HOURLY CHANGE?**

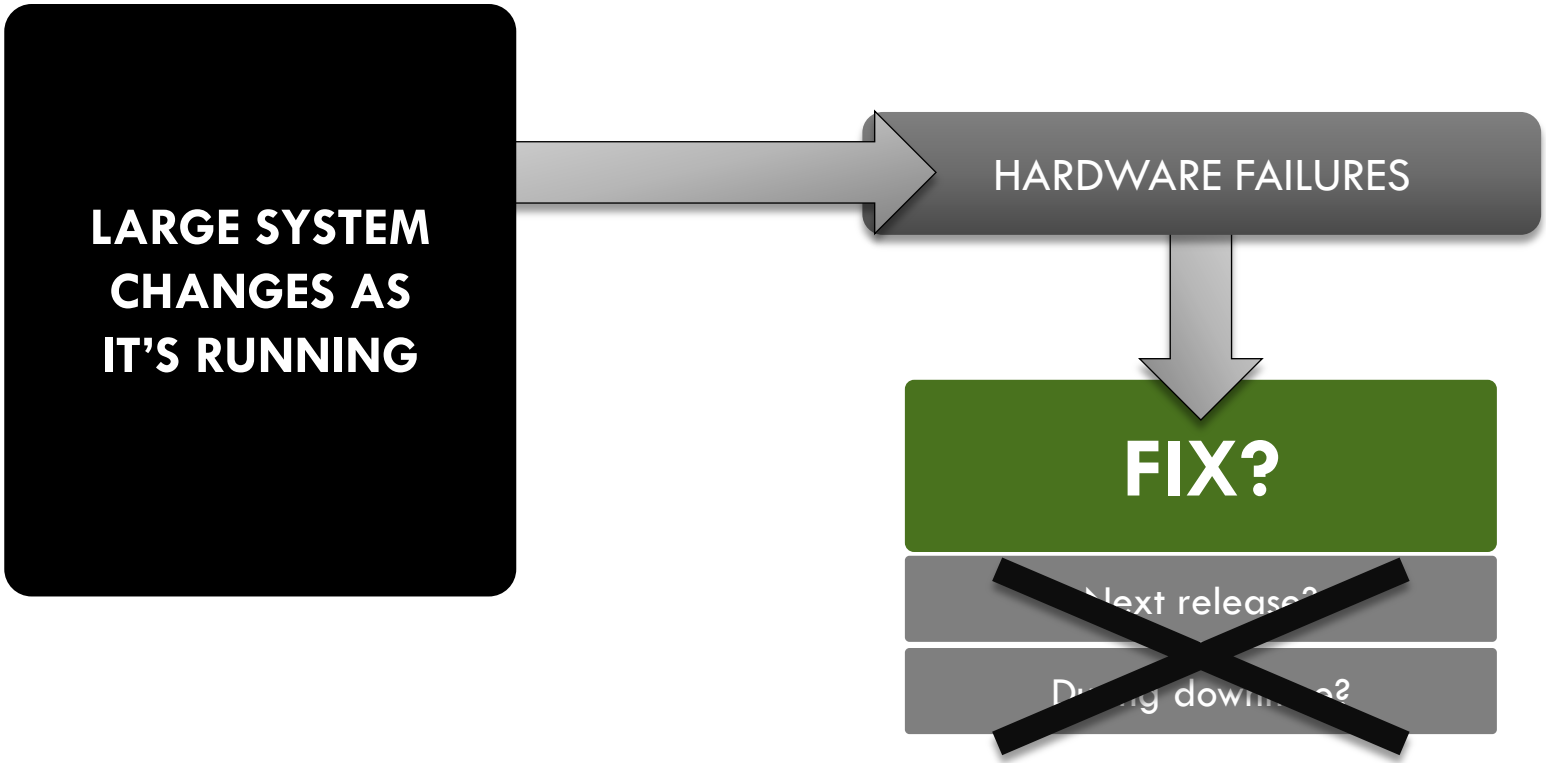
CODE A DYNAMIC SYSTEM



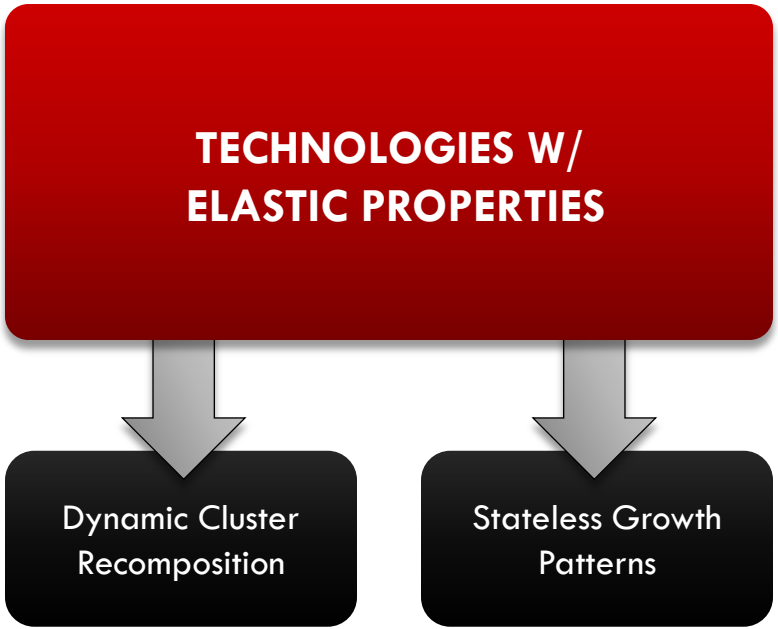
CODE A DYNAMIC SYSTEM



CODE A DYNAMIC SYSTEM



CODE A DYNAMIC SYSTEM



NOT EVERY PIECE OF YOUR STACK HAS TO BE ELASTIC

CODE A DYNAMIC SYSTEM

1. All relevant configuration properties are dynamic

2. Coherence near caches used to propagate changes to nodes dynamically

3. Algorithms written so they are aware their variables may change while running

LARGER EXAMPLES OF DYNAMIC BEHAVIOR

THREAD POOLS

=

DYNAMICALLY CONFIGURABLE

Entire machine/feature combinations can be deployed & updated

Hotfixes require less downtime

Features can be deployed in advance of release windows

AGENDA

- EMBRACING JAVA AND NoSQL
- SIMPLE IS BEST
- CODE A DYNAMIC SYSTEM
- SCALING BEST PRACTICES
- MONITOR EVERYTHING

PROBLEM #4:

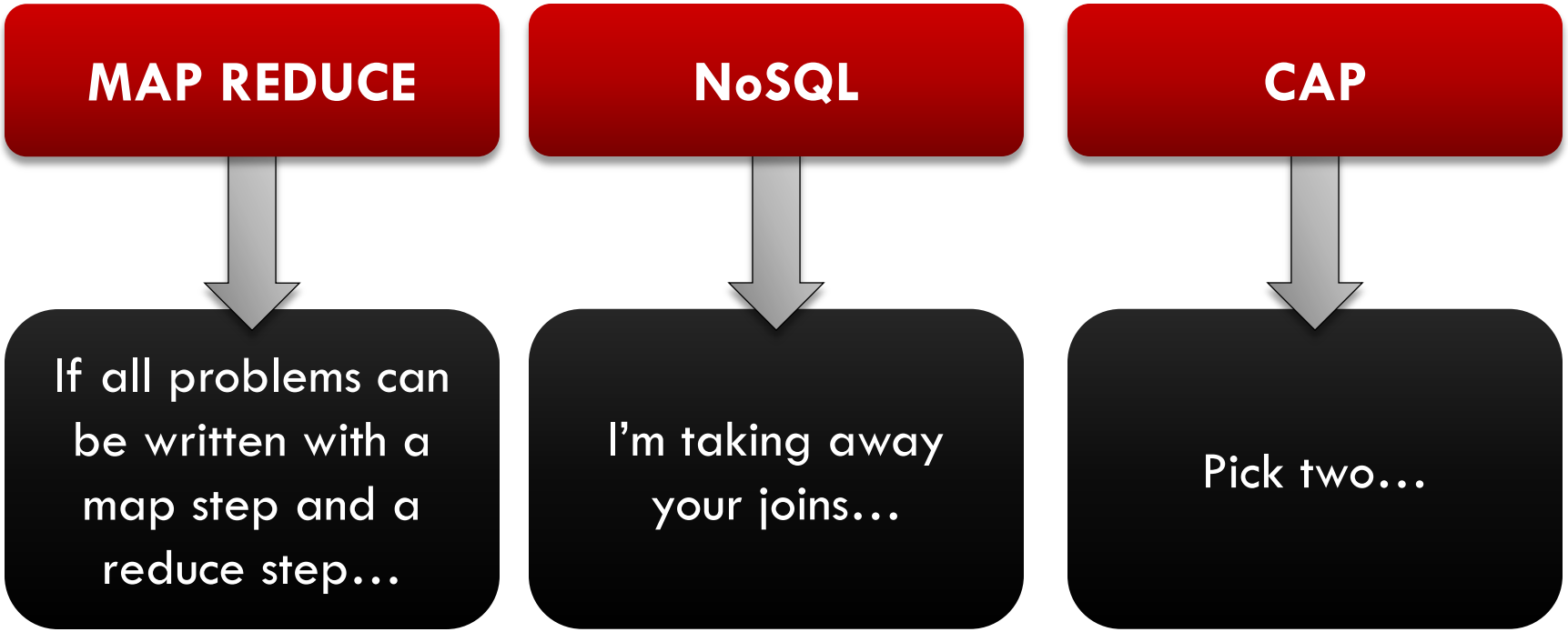
WHAT HAPPENS WHEN WE
FOLLOW ALL THE RULES...

...AND **STILL RUN INTO ISSUES?**

SCALING BEST PRACTICES HAVE CONSEQUENCES

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

SCALING BEST PRACTICES HAVE CONSEQUENCES



CONSEQUENCES

ATOMIC OPERATIONS OFTEN BECOME SCOPED BY ENTRY VALUES AND ROOT OBJECTS

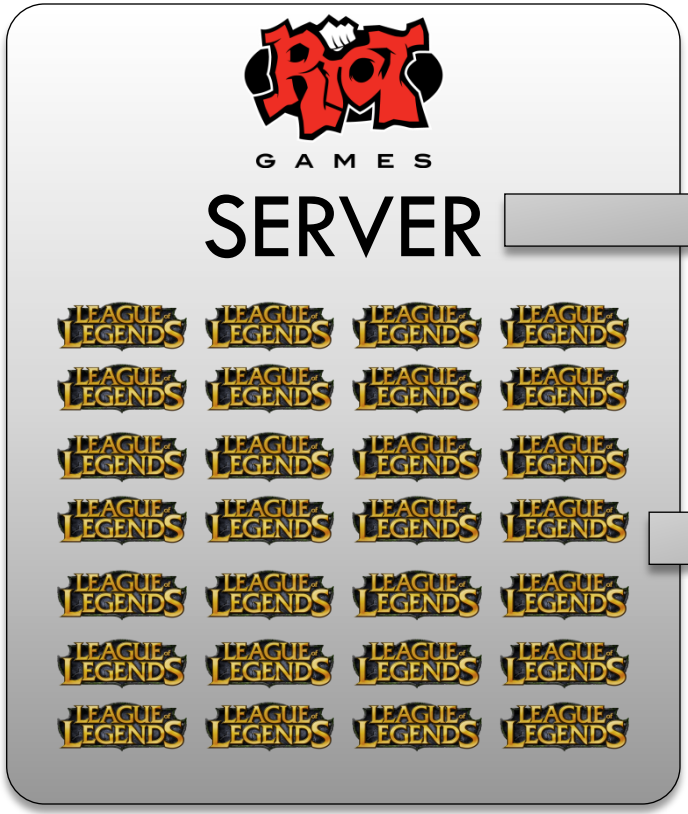
CONSEQUENCES

ATOMIC OPERATIONS OFTEN BECOME SCOPED BY ENTRY VALUES AND ROOT OBJECTS

CONSEQUENCES

ATOMIC OPERATIONS OFTEN BECOME SCOPED BY ENTRY VALUES AND ROOT OBJECTS

AN EXAMPLE OF A MISMATCH



ROOT OBJECT

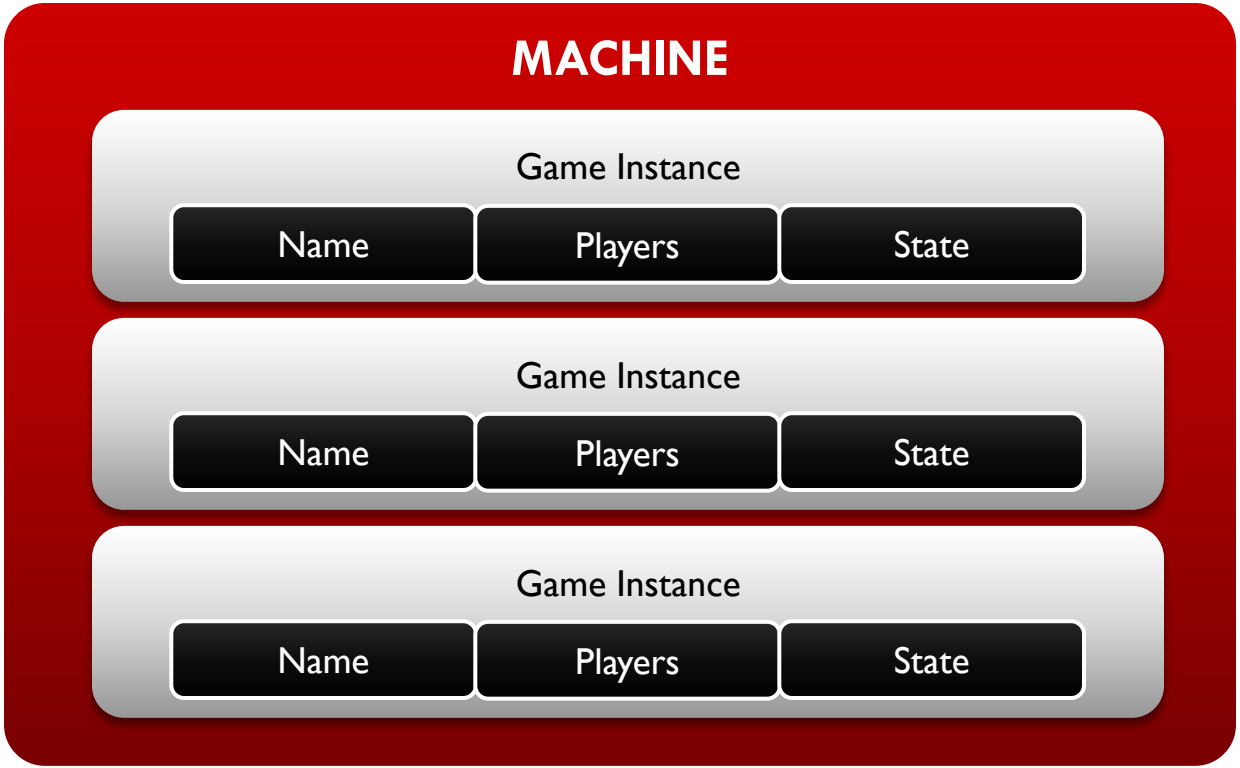
AS GAMES ARE ALLOCATED, CHILD OBJECTS ARE ADDED

AN EXAMPLE OF A MISMATCH

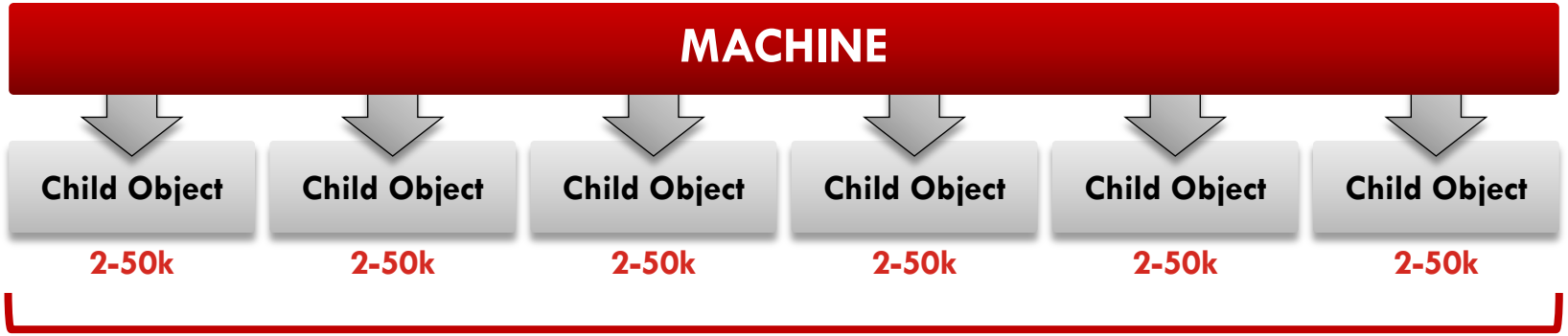
COMPLEXITY OF CHILD OBJECTS

GAMES PER SERVER

ROOT OBJECTS AND CHILD OBJECTS



EVOLUTION OF AN ANTI-PATTERN



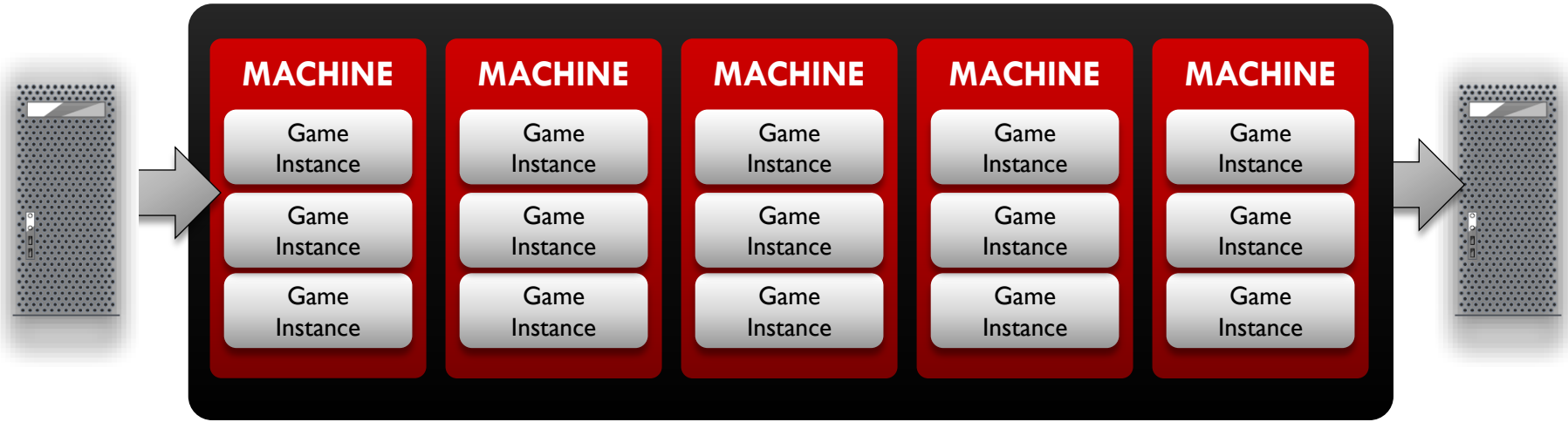
<20k → >500k

NETWORK TRANSFER FAST

OBJECT SERIALIZATION

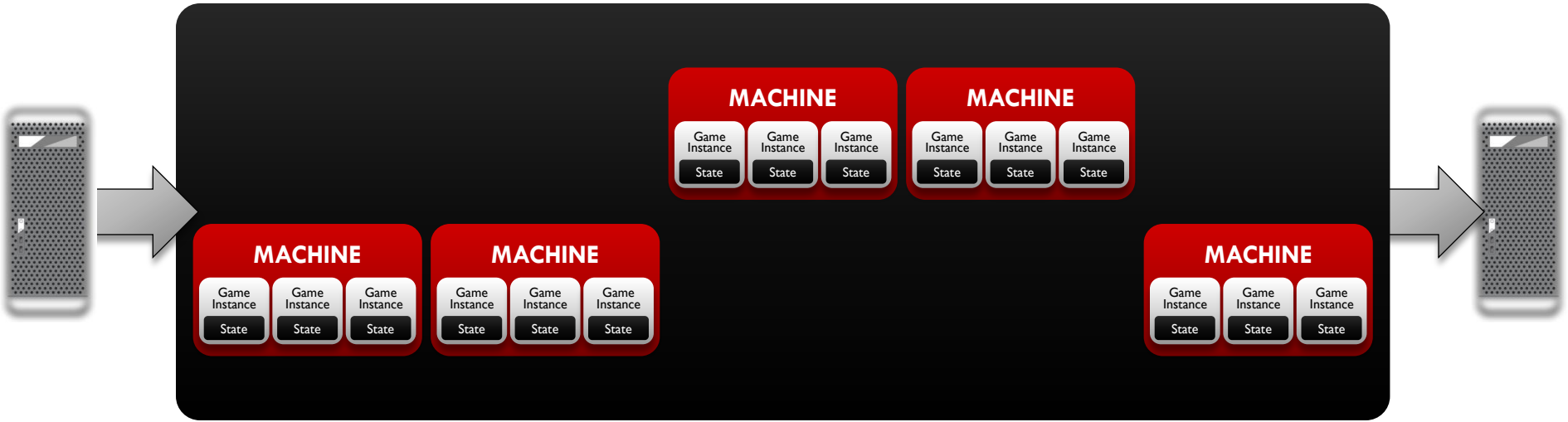
BOUNDING FACTORS

THE PIPE IS FULL



DO WE REALLY HAVE ONE OBJECT?

SMALLER IS BETTER!



AGENDA

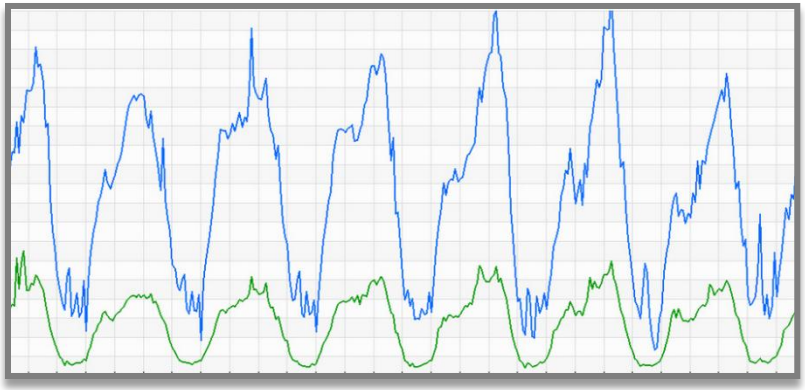
- EMBRACING JAVA AND NoSQL
- SIMPLE IS BEST
- CODE A DYNAMIC SYSTEM
- SCALING BEST PRACTICES
- MONITOR EVERYTHING

PROBLEM #5:

HOW DO WE **KNOW**…

…WHEN WE HAVE **A PROBLEM?**

MONITOR EVERYTHING



VS.

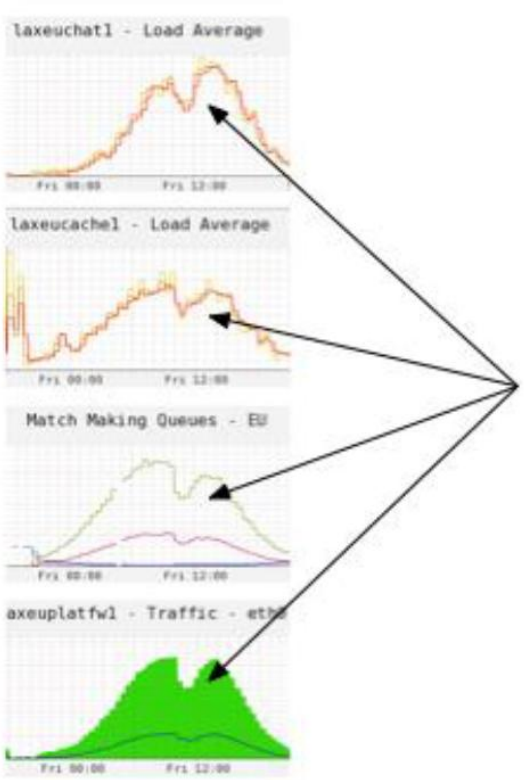
LOGS WITH MILLIONS OF OPERATIONS/DAY

MONITOR EVERYTHING

VS.

LOGS WITH MILLIONS OF OPERATIONS/DAY

MONITOR EVERYTHING



WHAT HAPPENED HERE?

Networking issue!

MONITOR EVERYTHING

1. Automate metrics gathering
2. Spring performance monitoring interceptor
3. Log out call stack on external calls
4. Sample internal calls
5. Automate reporting
6. Trivial cost vs. benefit

MONITOR EVERYTHING

DATA IS **USELESS** WITHOUT AN EASY WAY TO VIEW IT.

Top 50 EXTERNAL calls by volume

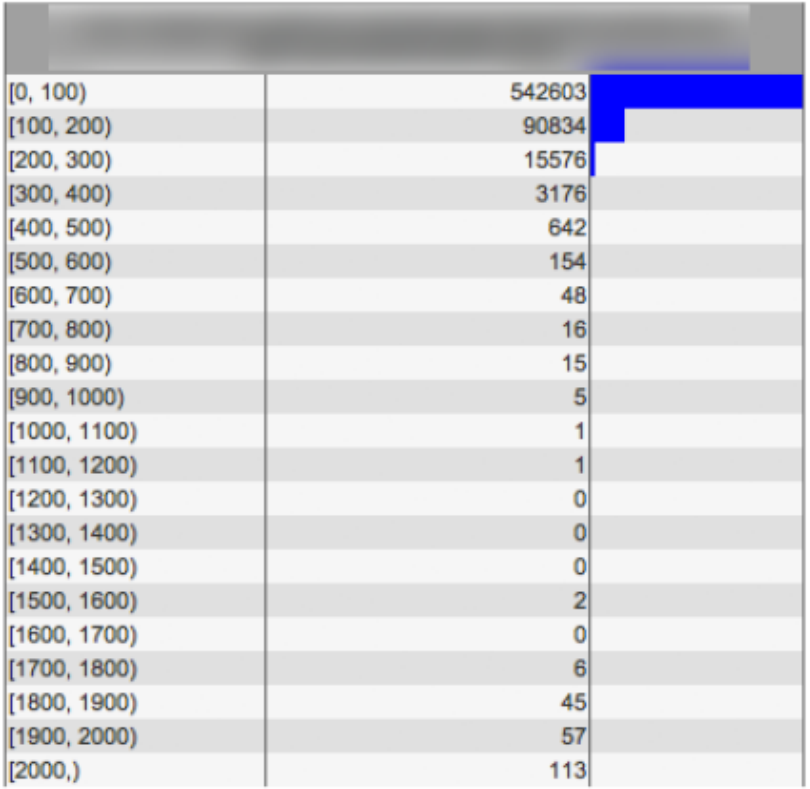
Service	Method	Current Import						Previous Import			
		Num Calls	Avg Call Time	% of Total Calls	Baseline Factor	% diff		Previous Num Calls	Previous Avg Call Time	Previous % of Total Calls	Previous Baseline Factor
[redacted]	[redacted]	3465105	27	7.3648%	2.6471	-4.5827%		3572459	24	7.5898%	2.7742
[redacted]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

...LETS GREP THE RED ITEMS...

MONITOR EVERYTHING

AUTOMATE NEXT
5 QUESTIONS/ANSWERS

(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

RECAP

- EMBRACING JAVA AND NoSQL
- SIMPLE IS BEST
- CODE A DYNAMIC SYSTEM
- SCALING BEST PRACTICES
- MONITOR EVERYTHING

QUESTIONS?

www.riotgames.com/careers

(We're also in the Career Pavilion at booth #CP1813)

SCOTT DELAP
SCALABILITY ARCHITECT
sdelap@riotgames.com
GDC 2012