

# HELLO DEVDAY

# Justin Marney

## HELLO DEVDAY

# Viget Labs

## Justin Marney

### HELLO DEVDAY

**GMU BCS '06**

**Viget Labs**

**Justin Marney**

**HELLO DEVDAY**

**Ruby '07**

**GMU BCS '06**

**Viget Labs**

**Justin Marney**

**Viget '08**

**Ruby '07**

**GMU BCS '06**

**Viget Labs**











# DISTRIBUTING YOUR DATA

**WHY?**

**DISTRIBUTING  
YOUR DATA**



**web applications are judged  
by their level of availability**

**WHY?**

**DISTRIBUTING  
YOUR DATA**

**ability to continue operating  
during failure scenarios**

web applications are judged  
by their level of availability

**WHY?**

**DISTRIBUTING  
YOUR DATA**



**ability to **manage availability**  
during failure scenarios**

ability to continue operating  
during **failure scenarios**

web applications are judged  
by their level of availability

**WHY?**

**increase throughput**

ability to **manage availability**  
during node failure

ability to continue operating  
during **failure scenarios**

web applications are judged  
by their level of availability

**WHY?**



**increase durability**

increase throughput

ability to **manage availability**  
during node failure

ability to continue operating  
during **failure scenarios**

web applications are judged  
by their level of availability

**increase scalability**

increase durability

increase throughput

ability to **manage availability**  
during node failure

ability to continue operating  
during **failure scenarios**

# SCALABILITY

"I can add twice as much **X** and get twice as much **Y**."

**X** = processor, RAM, disks, servers, bandwidth

**Y** = throughput, storage space, uptime



# SCALABILITY

scalability is a **ratio**.

**2:2 = linear scalability ratio**

**scalability ratio** allows you to predict how much it will cost you to grow.

# SCALABILITY

UP/DOWN/VERTICAL/  
HORIZONTAL/L/R/L/R/**A**/  
**B**/START



# SCALABILITY

**UP**

**grow your infrastructure**

**multiple data centers**

**higher bandwidth**

**faster machines**



# SCALABILITY

**DOWN**

**shrink your infrastructure**

**mobile**

**set-top**

**laptop**

# SCALABILITY

## VERTICAL

add to a single node

CPU

RAM

RAID

# SCALABILITY

## HORIZONTAL

**add more nodes**

**distribute the load**

**commodity cost**

**limited only by capital**

**@gary\_hustwit:** Dear  
**Twitter: when a World Cup  
match is at the 90th  
minute, you might want to  
turn on a few more servers.**



# ASYNCHRONOUS

**A distributed transaction is bound by availability of all nodes.**

# ASYNCHRONOUS

**A distributed transaction is bound by availability of all nodes.**

$$(.99^1) = .99$$

$$(.99^2) = .98$$

$$(.99^3) = .97$$

# ASYNCHRONOUS

**Asynchronous systems operate without the concept of **global state**.**

**The concurrency model more accurately reflects the real world.**

# ASYNCHRONOUS

**Asynchronous systems operate without the concept of **global state**.**

**The concurrency model more accurately reflects the real world.**

**What about my **ACID**!?**



# ACID

## **A**tomic

Series of database operations either all occur, or nothing occurs.

## **C**onsistent

Transaction does not violate any integrity constraints during execution.

## **I**solated

Cannot access data that is modified during an incomplete transaction.

## **D**urable

Transactions that have committed will survive permanently.

# ACID

**Defines a set of characteristics that aim to **ensure consistency**.**

**What happens when we realize that in order scale we need to distribute our data and handle asynchronous operations?**

# ACID

**Without global state, no Atomicity.**

**Without a linear timeline, no transactions and no Isolation.**

**The canonical location of data might not exist, therefore no D.**

**Without A, I, or D, Consistency in terms of entity integrity is no longer guaranteed.**

# CAP Theorem

**Eric Brewer @ 2000 Principles of Distributed Computing (PODC).**

**Seth Gilbert and Nancy Lynch published a formal proof in 2002.**



# CAP Acronym

**Consistency:** Multiple values for the same piece of data are not allowed.

**Availability:** If a non-failing node can be reached the system functions.

**Partition-Tolerance:** Regardless of packet loss, if a non-failing node is reached the system functions.

# CAP Theorem

**Consistency, Availability, Partition-Tolerance: Choose One...**

# CAP Theorem

**Single node systems bound by CAP.**

**100% Partition-tolerant**

**100% Consistent**

**No Availability Guarantee**

# CAP Theorem

Multi-node systems bound by CAP.

CA : DT, 2PC, ACID

CP : Quorum, distributed databases

AP : Dynamo, no ACID

# CAP Theorem

**CAP** doesn't say **AP** systems are the solution to your problem.

Not an **absolute decision**.

Most systems are a hybrid of **CA**, **CP**, & **AP**.



# CAP Theorem

**Understand the trade-offs and use that understanding to build a system that fails predictably.**

**Enables you to build a system that degrades gracefully during a failure.**

# BASE

**Dan Pritchett**

**BASE: An ACID Alternative**

**Associate for Computing Machinery  
Queue, 2008**

# BASE

**BASE: An ACID Alternative**

**Basically Available**

**Soft State**

**Eventually Consistent**

# BASE

**BASE: An ACID Alternative**

**Basically Available**

**Soft State**

**Eventually Consistent**

# Eventually Consistent

Rename to **Managed Consistency**.

Does not mean **probable** or **hopeful**  
or **indefinite time in the future**.

Describes what happens during a  
failure.

# Eventually Consistent

**During certain scenarios a decision must be made to either return inconsistent data or deny a request.**

**EC** allows you control the level of consistency vs. availability in your application.



# Eventually Consistent

**In order to achieve availability in an asynchronous system, accept that failures are going to happen.**

**Understand failure points and know what you are willing to give up in order to achieve availability.**

**How can we model the operations  
we perform on our data to be  
asynchronous & EC?**

**Model system as a network of independent components.**

**Partition components along functional boundaries.**

**Don't interact with your data as one big global state.**

**This doesn't meant **every** part of  
your system must operate this way!**

**Use **ACID 2.0** to help identify and  
architect components than can.**

# ACID 2.0

## Associative

**Order of operations does not change the result.**

## Commutative

**Operations can be aggregated in any order.**

## Idempotent

**Operation can be applied multiple times without changing the result.**

## Distributed

**Operations are distributed and processed asynchronously.**

# OPS BROS

**Incremental scalability**

**Homogeneous** node responsibilities

**Heterogeneous** node capabilities

# LINKS

[Base: An ACID Alternative](#)

[Into the Clouds on New Acid](#)

[Brewer's CAP theorem](#)

[Embracing Concurrency At Scale](#)

[Amazon's Dynamo](#)



# ME

<http://sorescode.com>

<http://github.com/gotascii>

<http://spkr8.com/s/1>

[@vigemarn](#)