WHERE TO PUT DATA

– or –

What are we going to *do* with all this stuff?

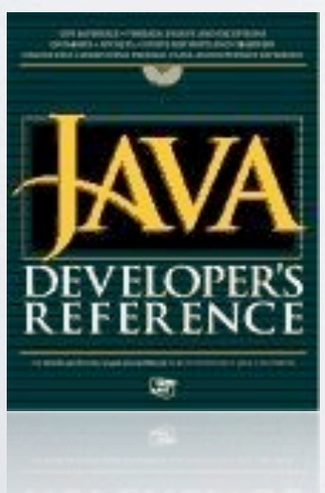
About The Speaker

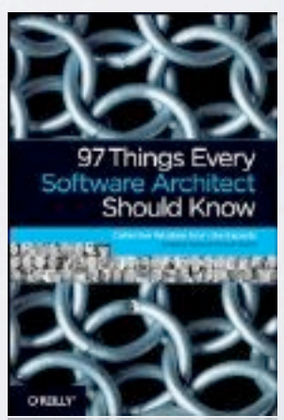
Application Developer/Architect – 20 years

Web Developer – 15 years

IT Operations – 7 years

IT Services Executive - 6 years





MongoDB

Key-value

NetStorage

GridGain

Distributed

Riak

S3

Replicate

HBase

Shard

CloudFront

ACID

Schemaless

Relational

Voldemort

Cluster

Coherence

Document

Memcached

Cassandra

Neo4J

BDB

CouchDB

Cache

Redis

BDB

Graph

Schema

Xindice

HDFS

BASE

eXist

BigMemory

Scalability ⟷ Consistency

Relational ⟷ Not Only SQL

Flexible ⟷ Rigid

ISP

CDN
Egress

Caching
Proxy/LB

Caching
Proxy/LB

App Web

Media Web

API Gateway

App layer

CMS

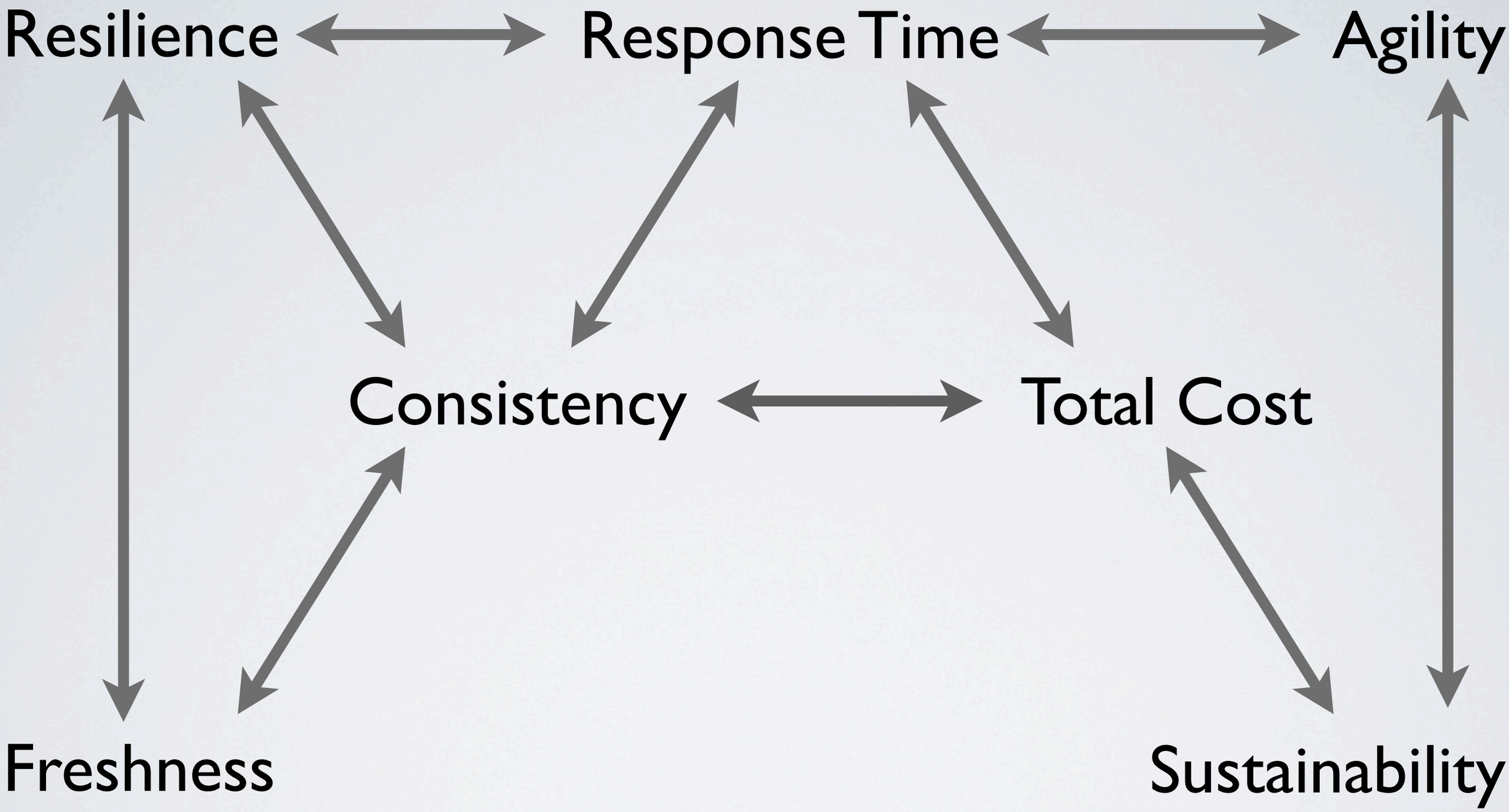
Cache farm

Search farm

Search
Admin

Indexer

Jobbers


Resilience
Response Time
Agility
Consistency
Total Cost
Freshness
Sustainability

BACK IN THE 90'S

BACK IN THE 90'S


imation
ULTRIUM LTO 4
800GB / 1.6TB

Hierarchical ("Network") Database

OS 2200

Relational Mapper

Hierarchical ("Network") Database

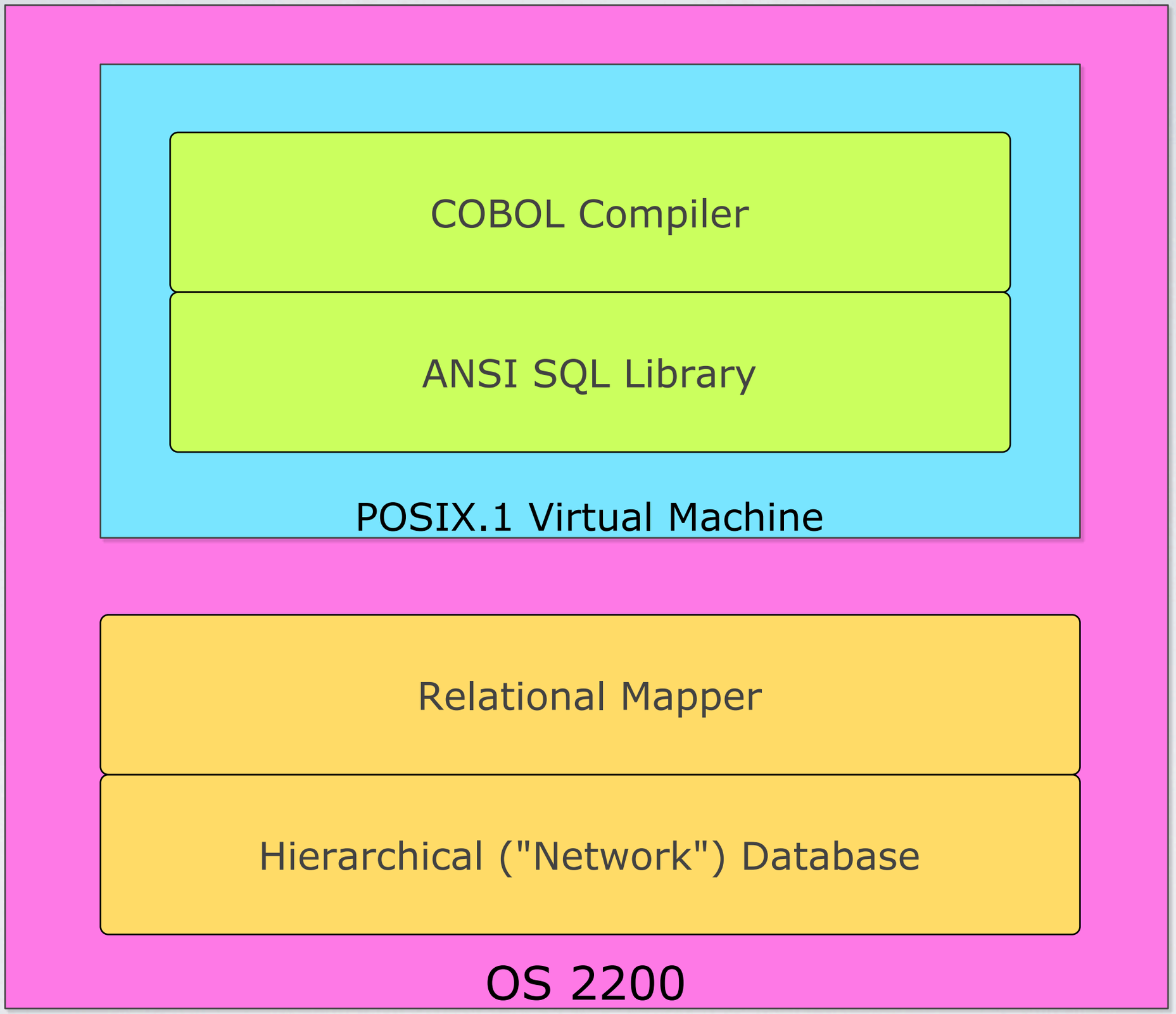
OS 2200

POSIX.1 Virtual Machine

Relational Mapper

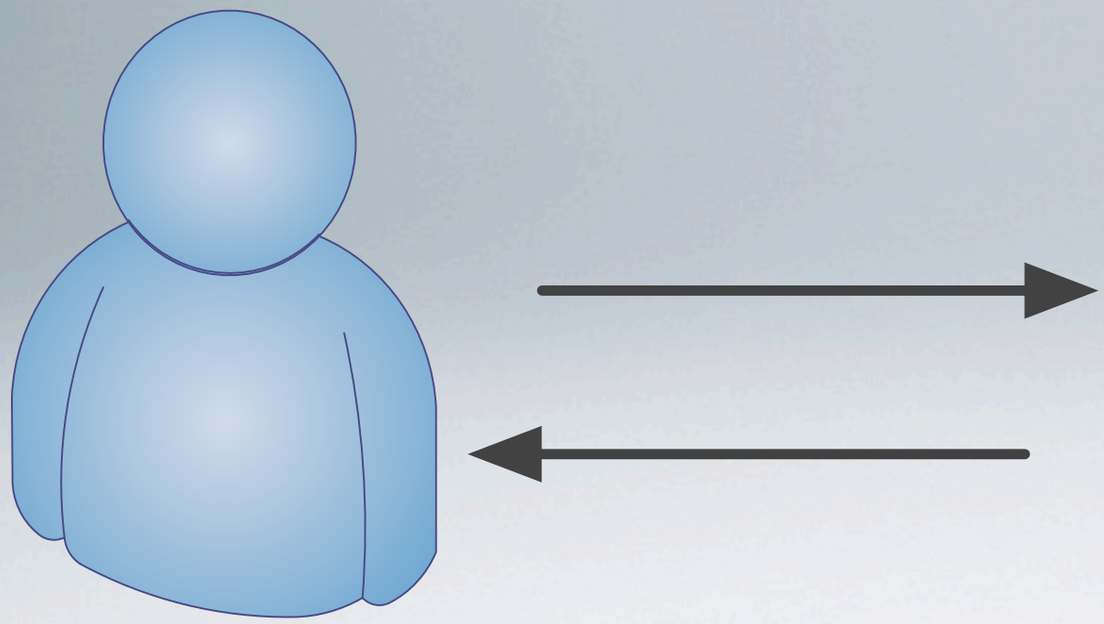
Hierarchical ("Network") Database

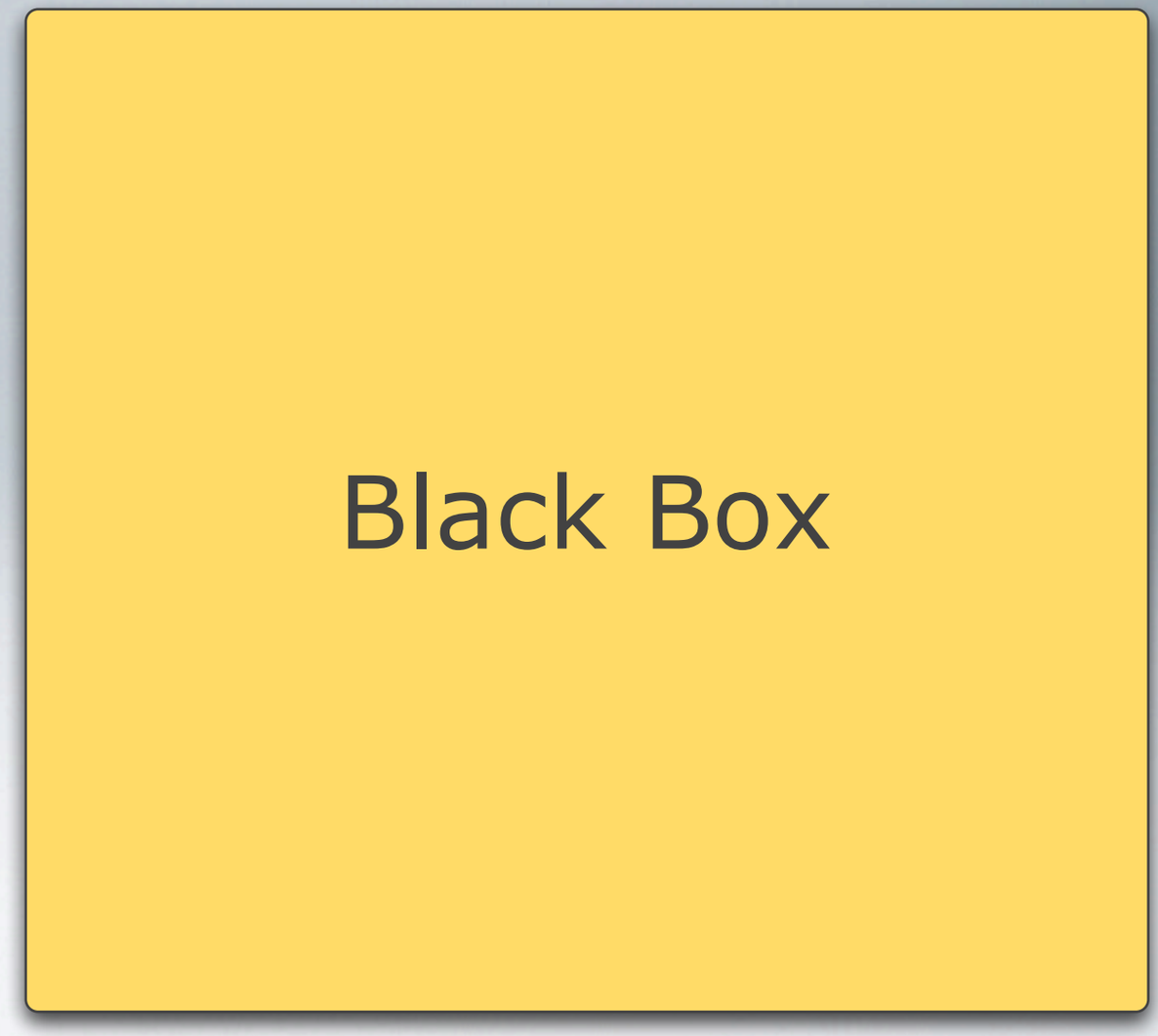
OS 2200


COBOL Compiler
ANSI SQL Library
POSIX.1 Virtual Machine
Relational Mapper
Hierarchical ("Network") Database
OS 2200

Given enough time, and perversity, you can create any query model on top of any storage model.

FUNDAMENTAL PREMISE





THERE ARE THINGS YOU CANNOT KNOW

Will a response arrive?

When?

Was it stored or computed?

Is it still true?

SAY WHEN

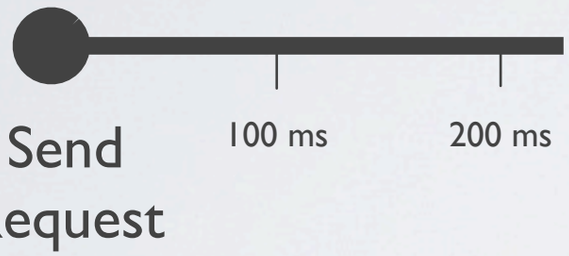
The Importance of Response Time Distribution

ASYMMETRY OF TIME

Send

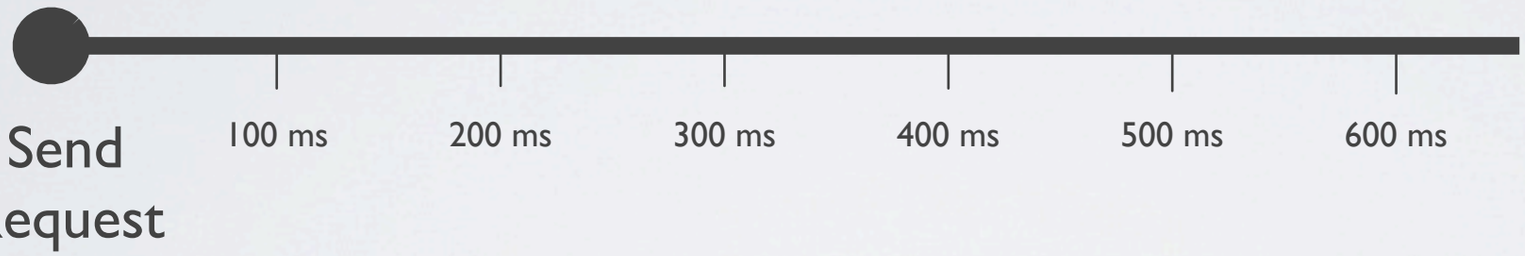
Request

ASYMMETRY OF TIME



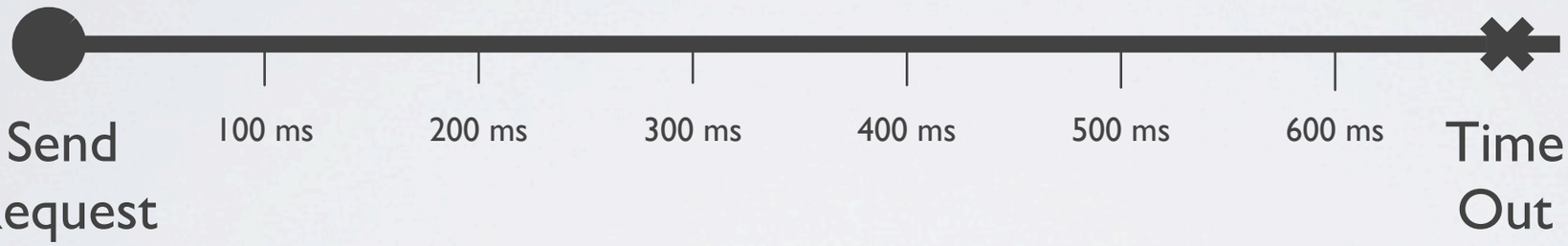
Request

ASYMMETRY OF TIME



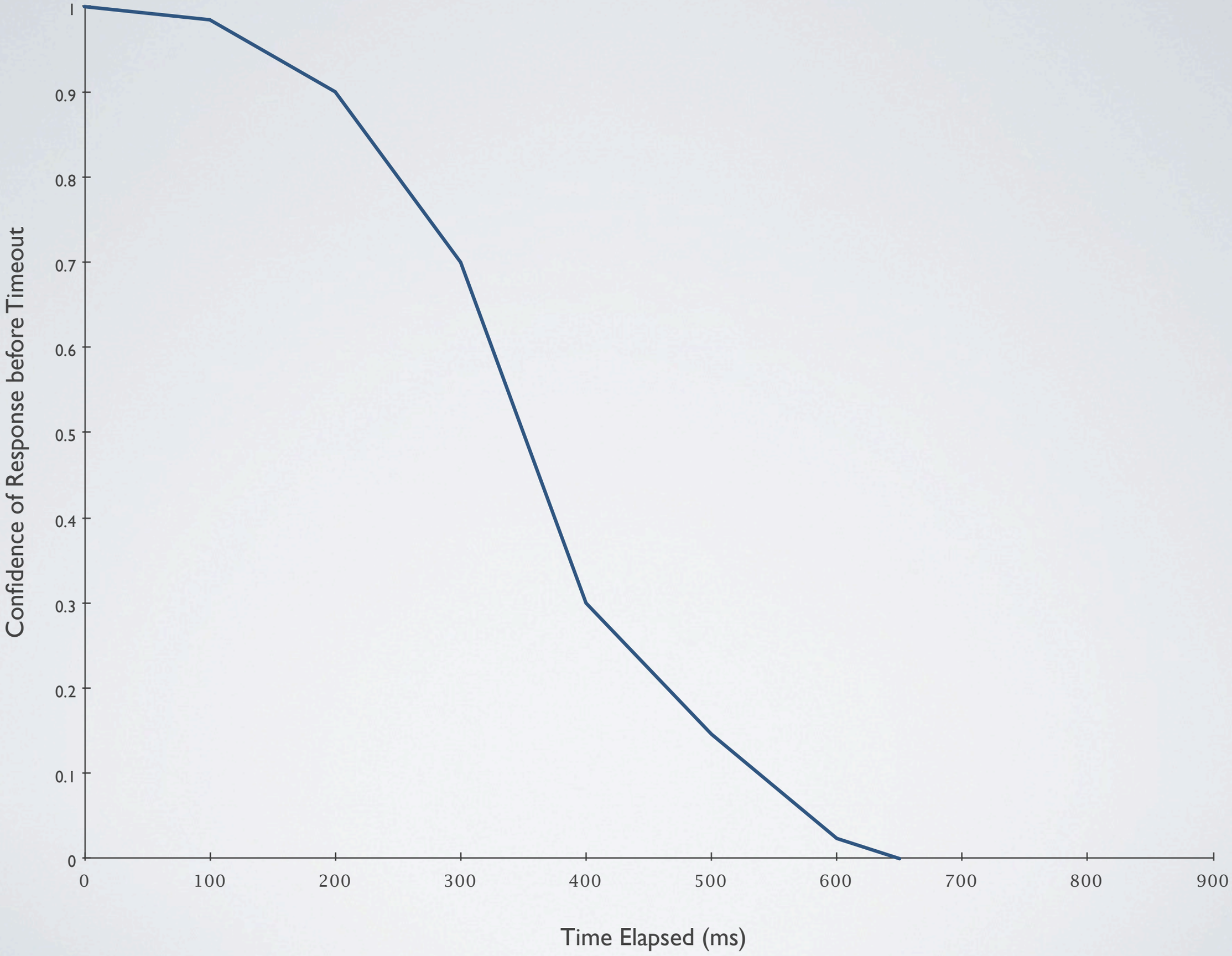
Request

ASYMMETRY OF TIME

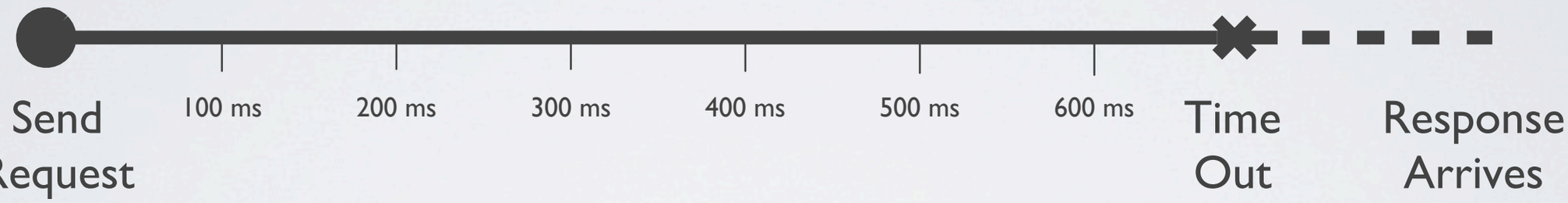


Request

Out

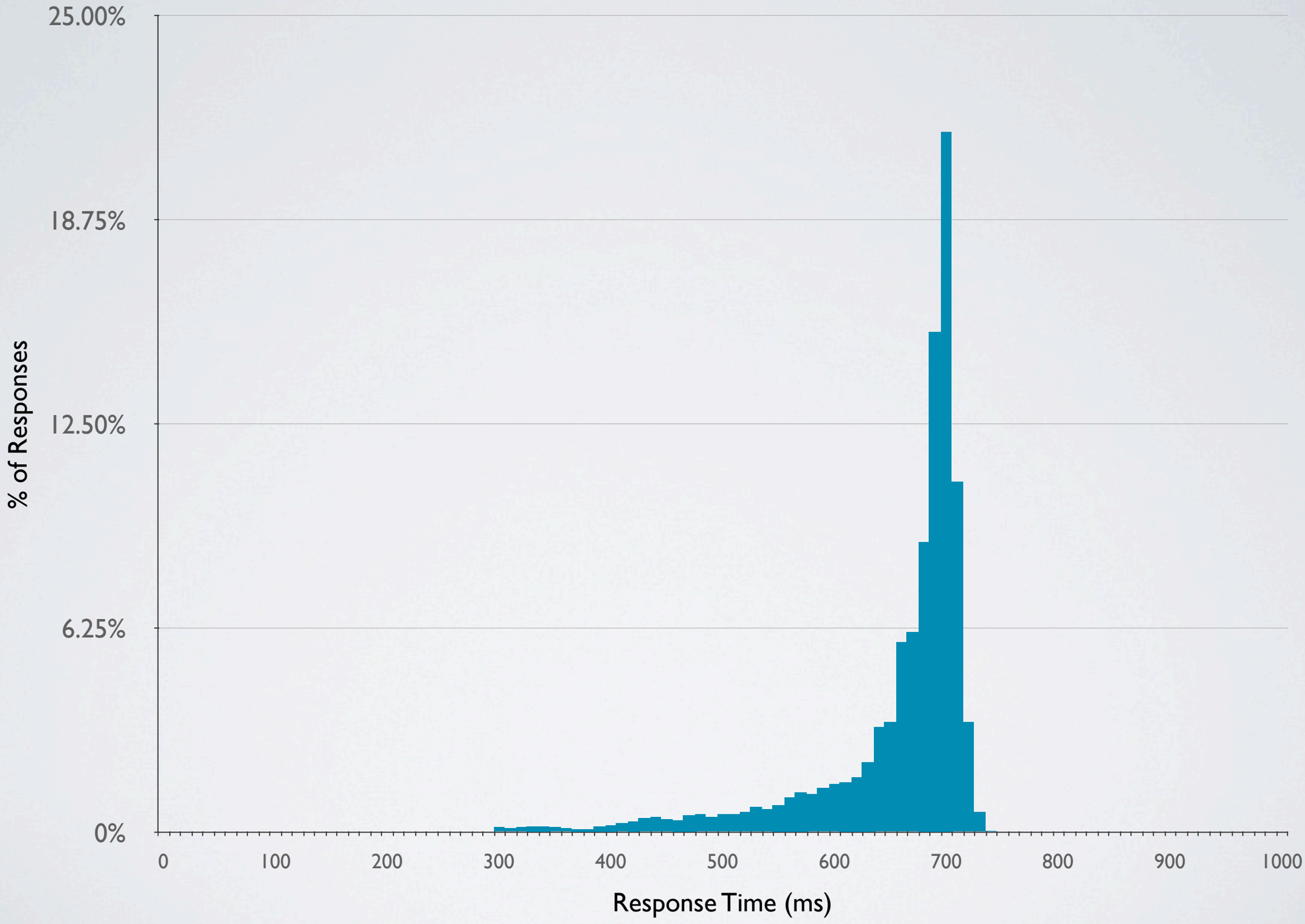

Confidence of Response before Timeout
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
100
200
300
400
500
600
700
800
900
Time Elapsed (ms)

ASYMMETRY OF TIME

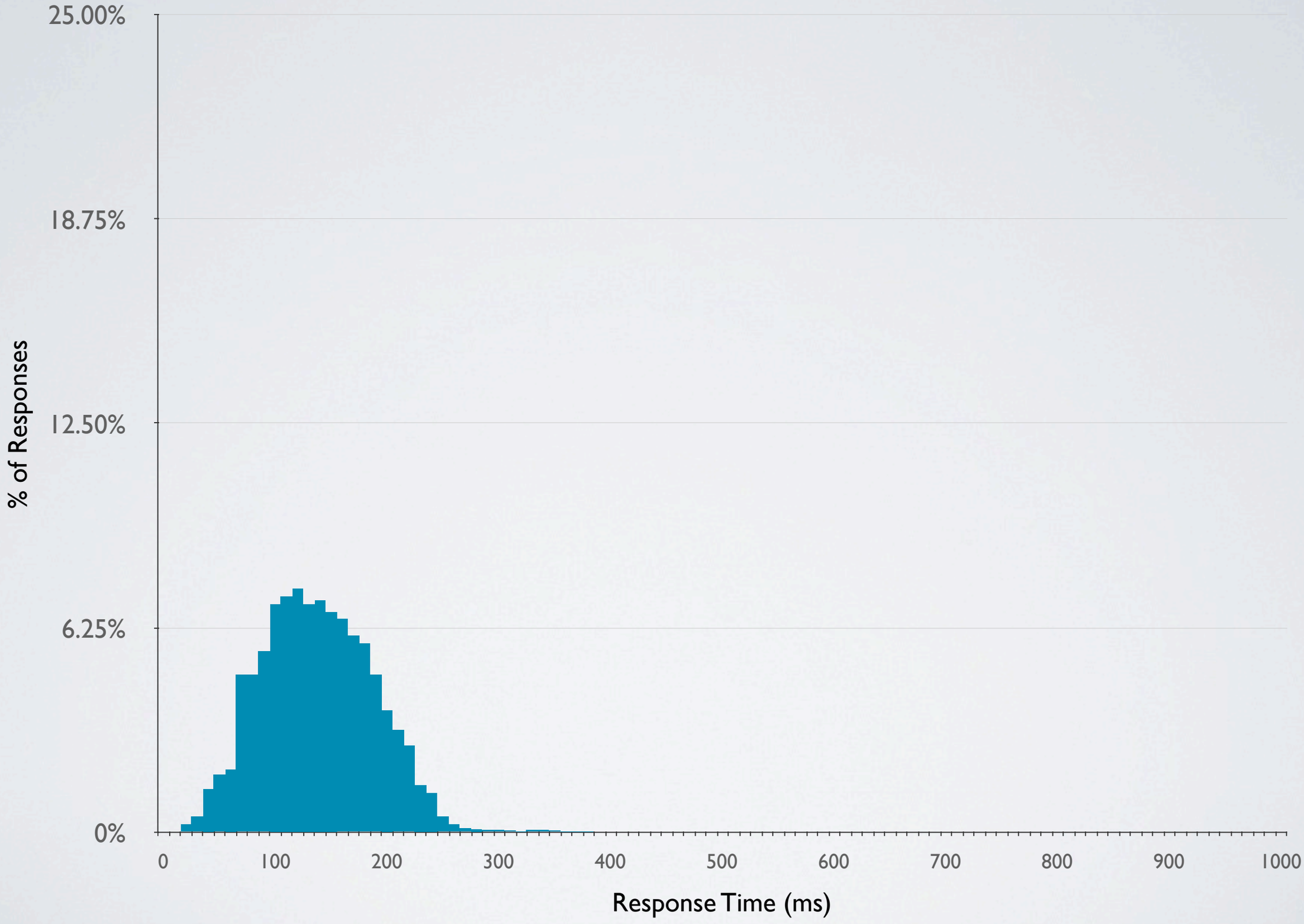


To the observer, there is no difference between “too slow” and “not there”.

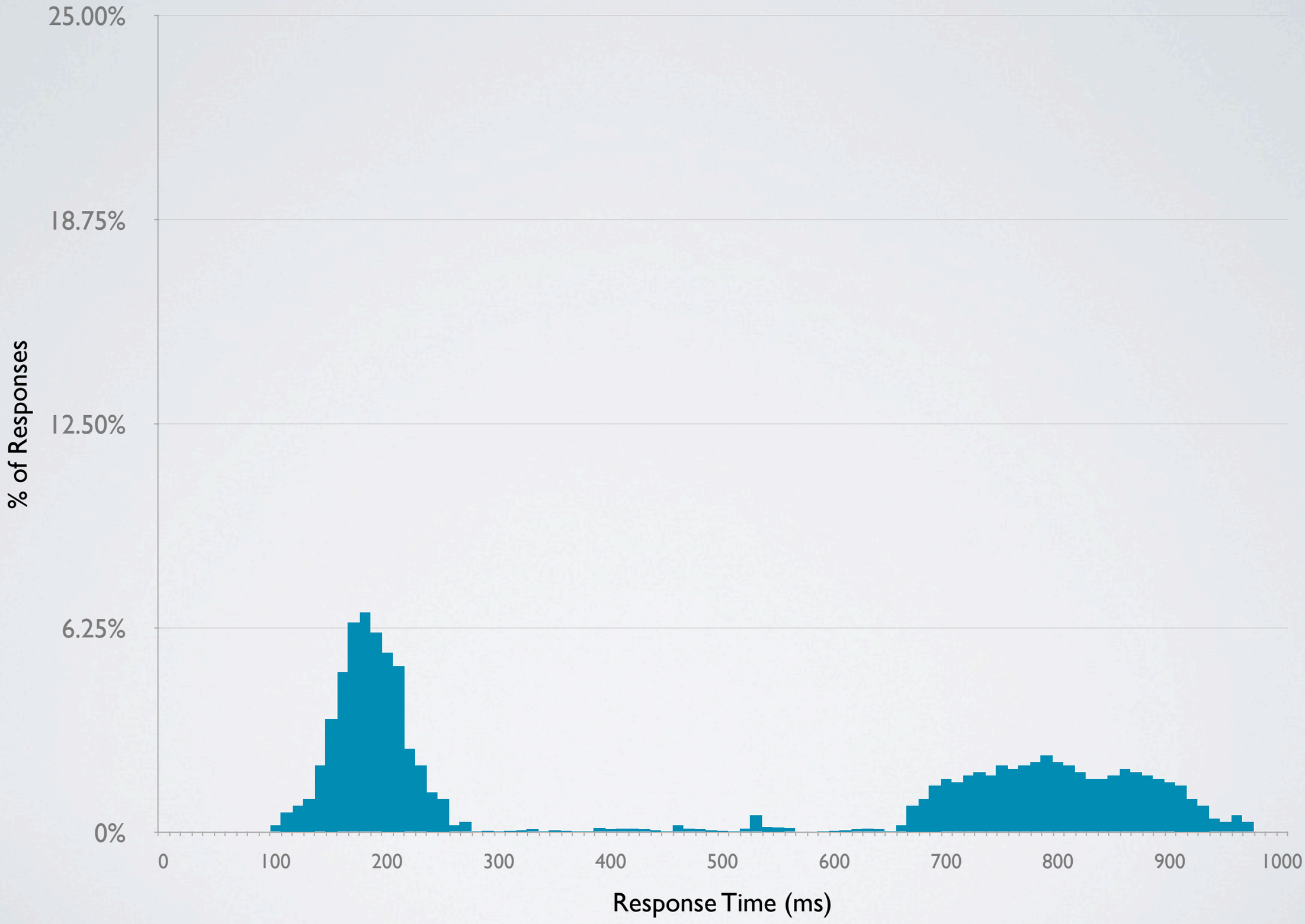
Response Time Histogram



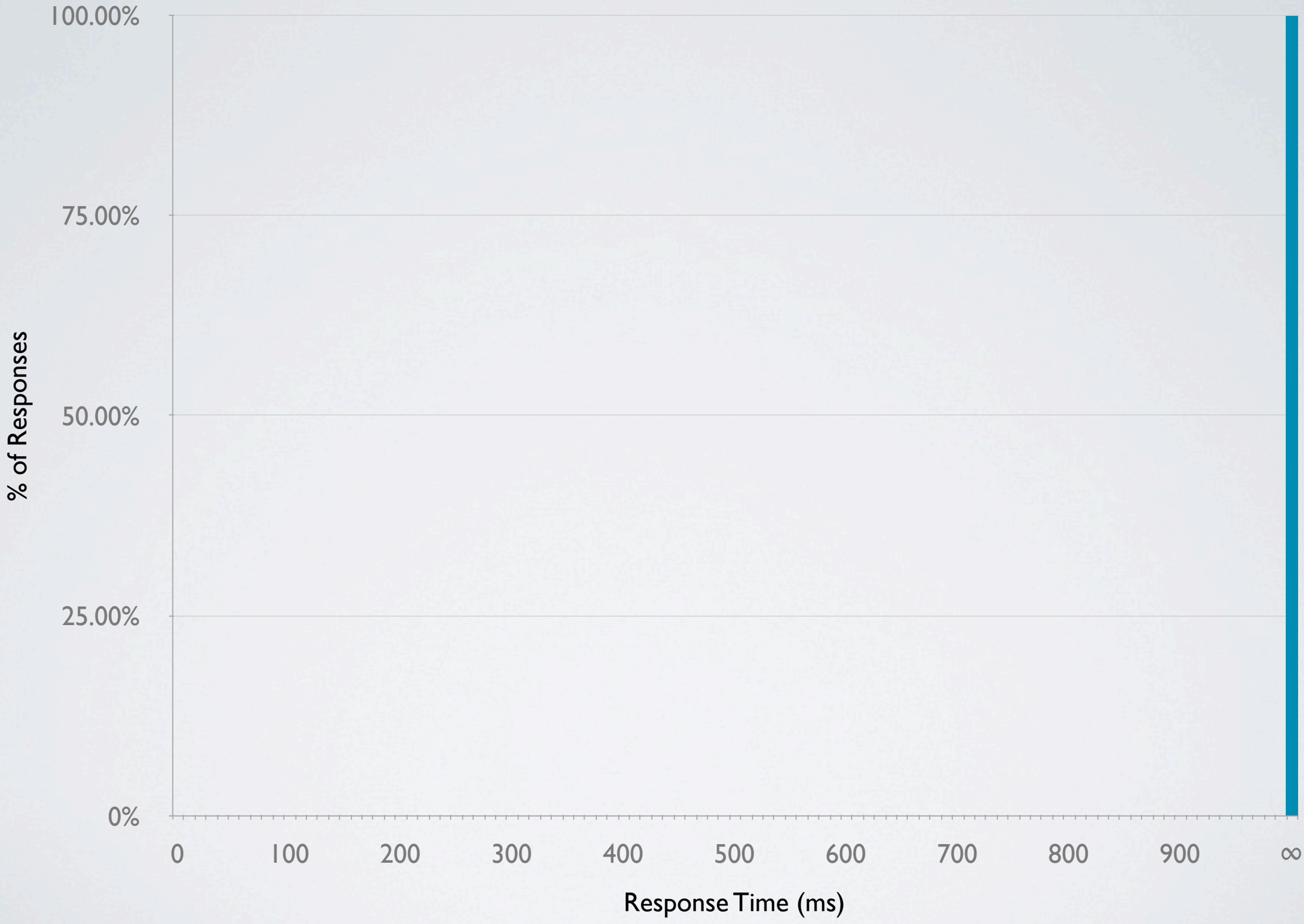
Response Time Histogram



Response Time Histogram



Response Time Histogram

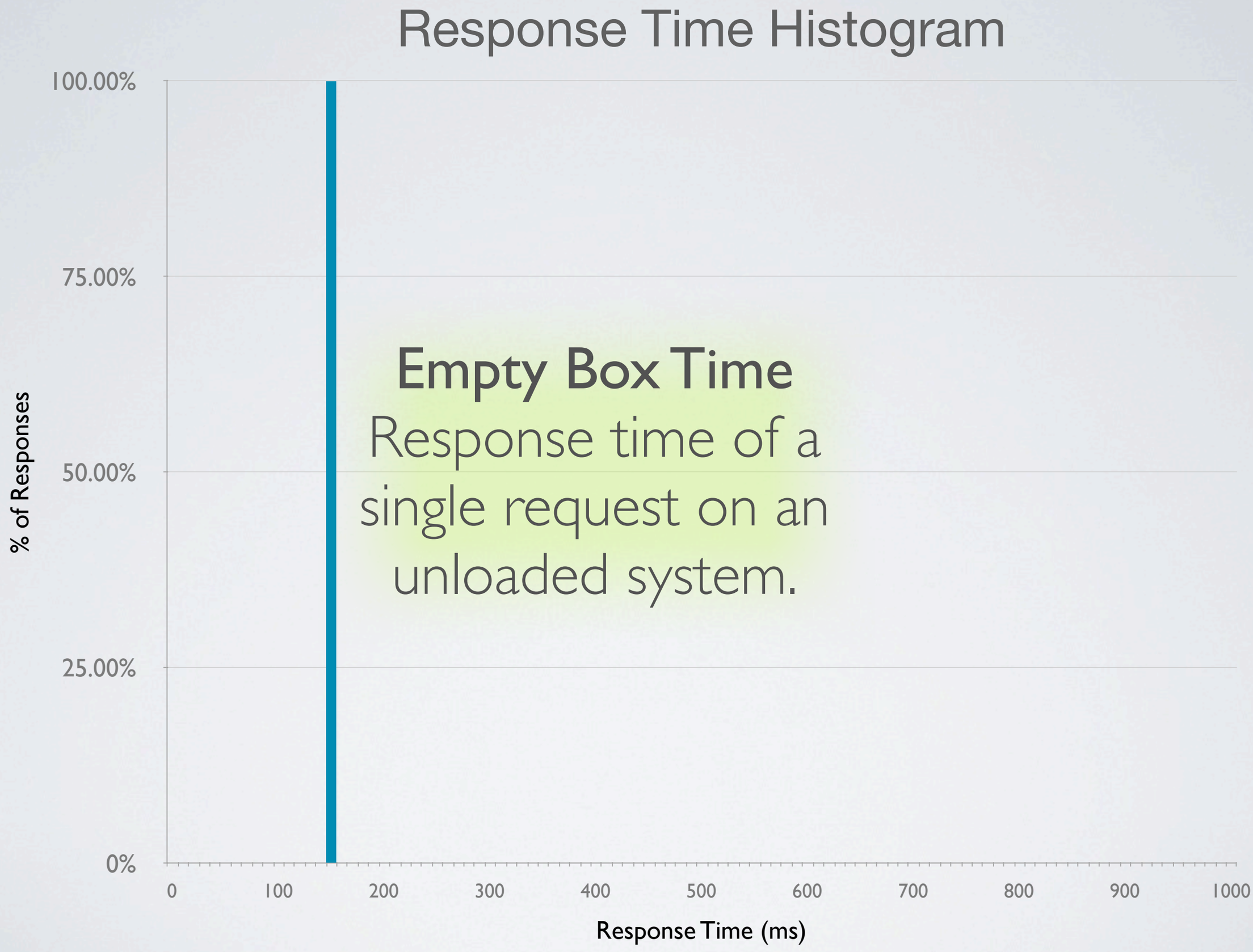


This is a talk about data and data storage.

Why am I talking so much about observers and response time?

What about scalability?

Why do we worry about scalability?


Response Time Histogram
100.00%
75.00%
50.00%
25.00%
0%
% of Responses
0
100
200
300
400
500
600
700
800
900
1000
Response Time (ms)
Empty Box Time
Response time of a single request on an unloaded system.

Scalability is a proxy.

What we want is fast responses, under all loads.

CDN
Egress

Caching
Proxy/LB

Caching
Proxy/LB

App Web

Media Web

API Gateway

App layer

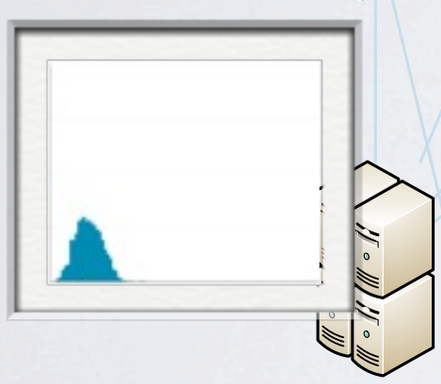
CMS

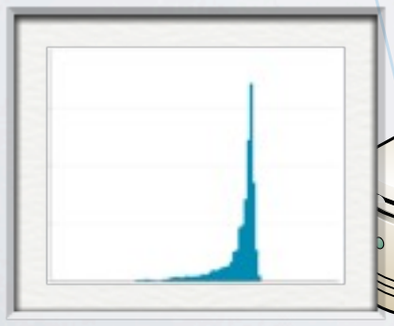
Search farm

Search
Admin

Indexer

Jobbers





HOW TRUE?

CDN Egress

Caching Proxy/LB

Caching Proxy/LB

App Web

Media Web

API Gateway

App layer

CMS

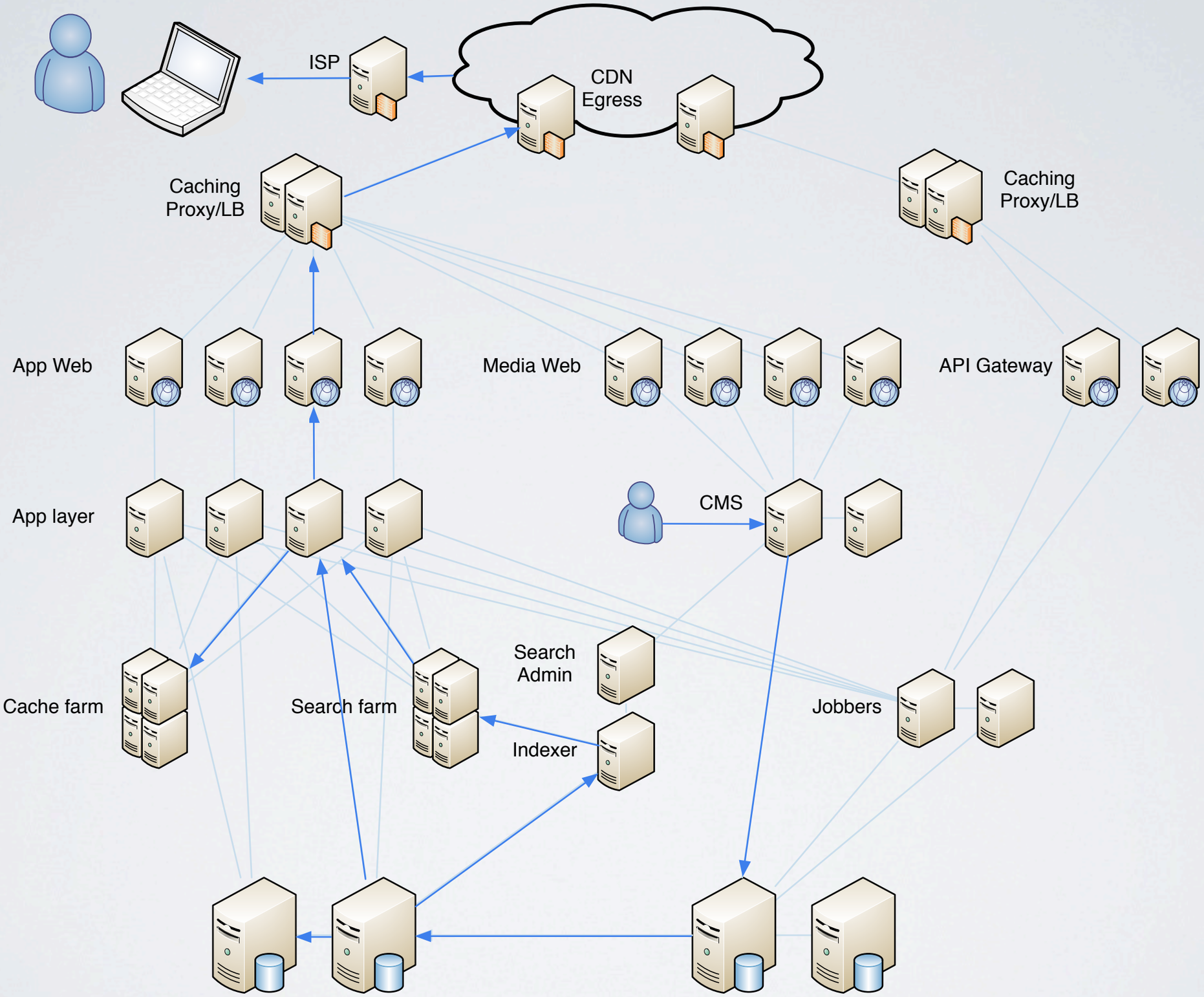
Cache farm

Search farm

Search Admin

Indexer

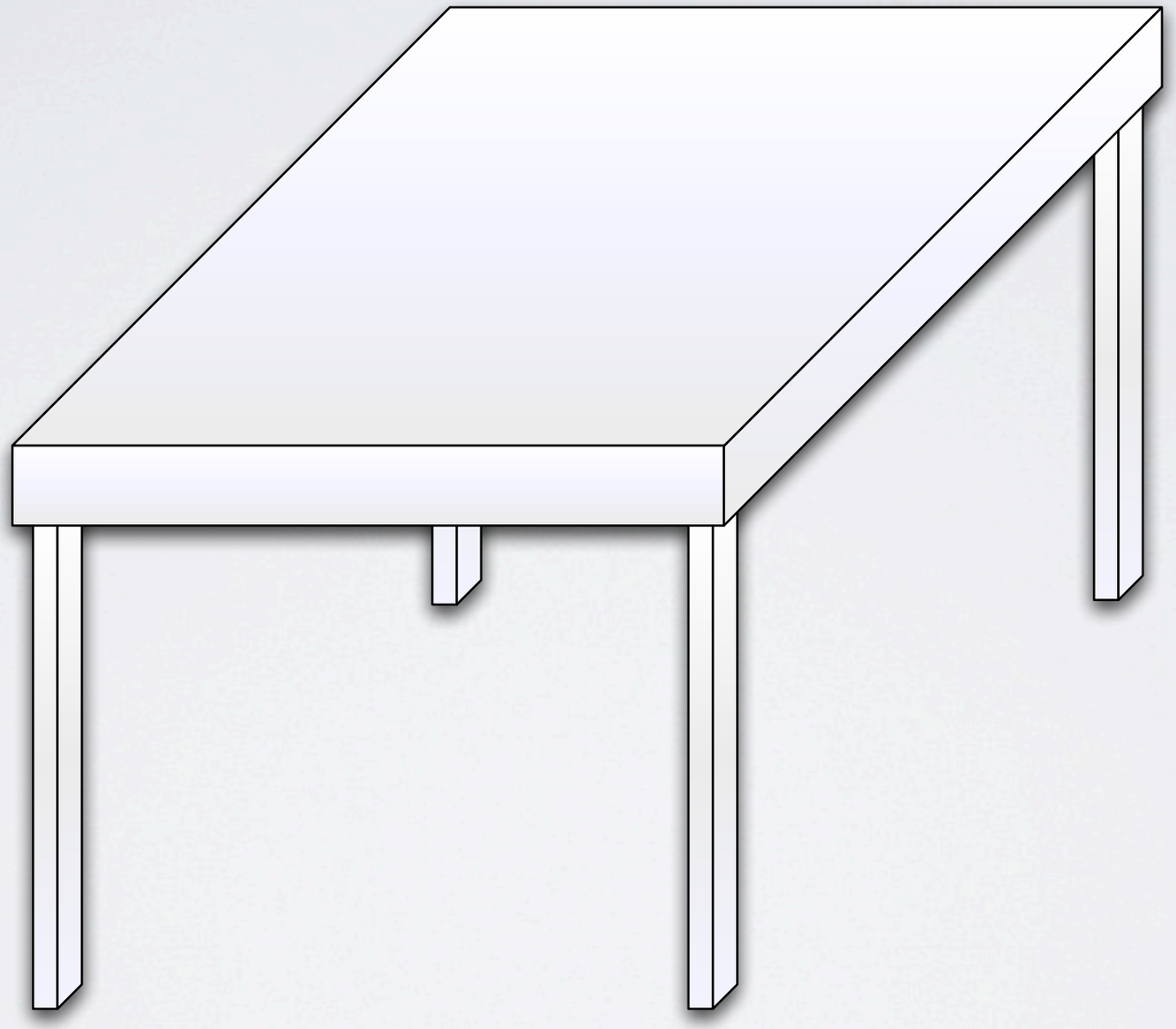
Jobbers


ISP
CDN
Egress
Caching
Proxy/LB
Caching
Proxy/LB
App Web
Media Web
API Gateway
App layer
CMS
Search
Admin
Cache farm
Search farm
Jobbers
Indexer

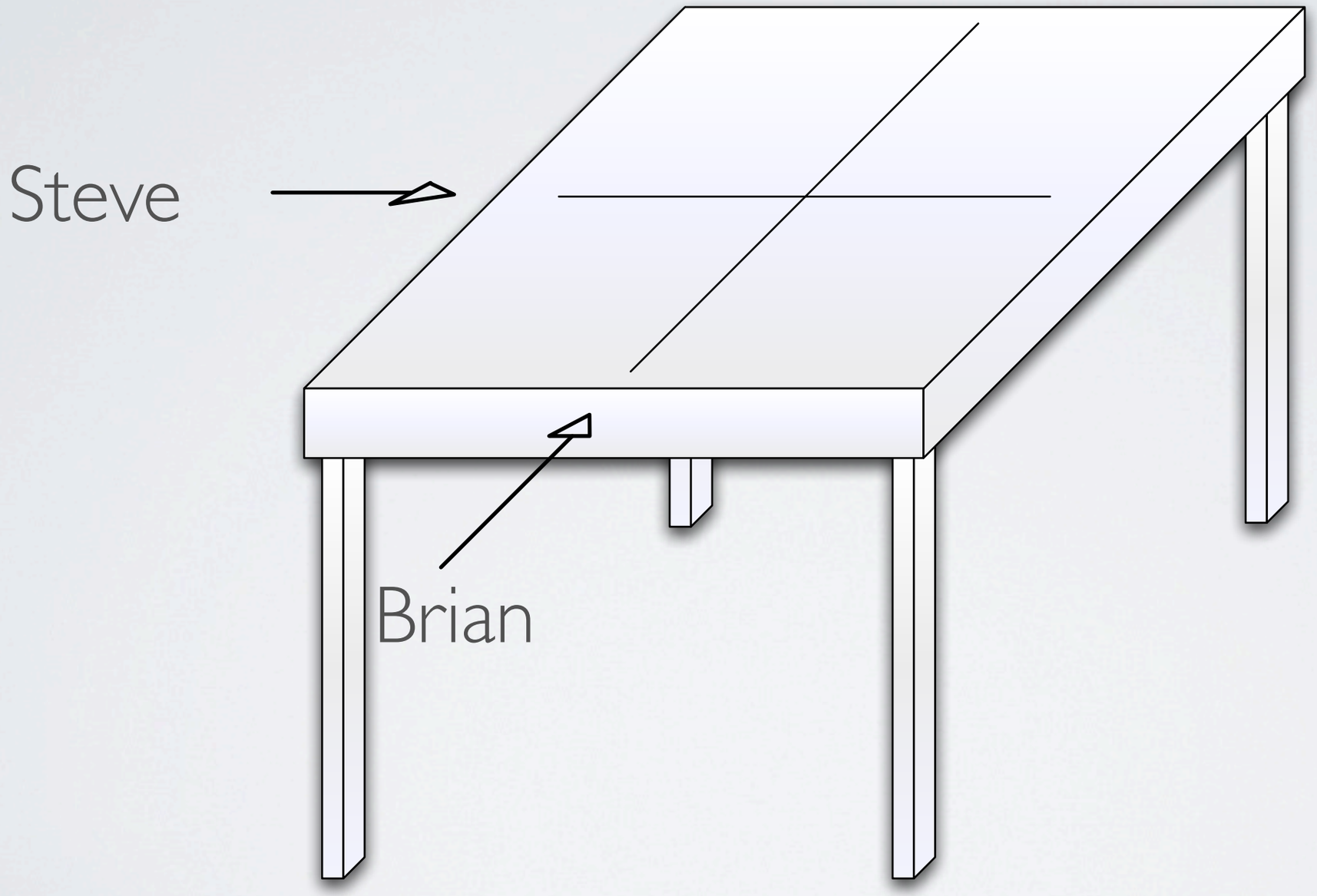
SAYS WHO?

Thoughts on Consistency

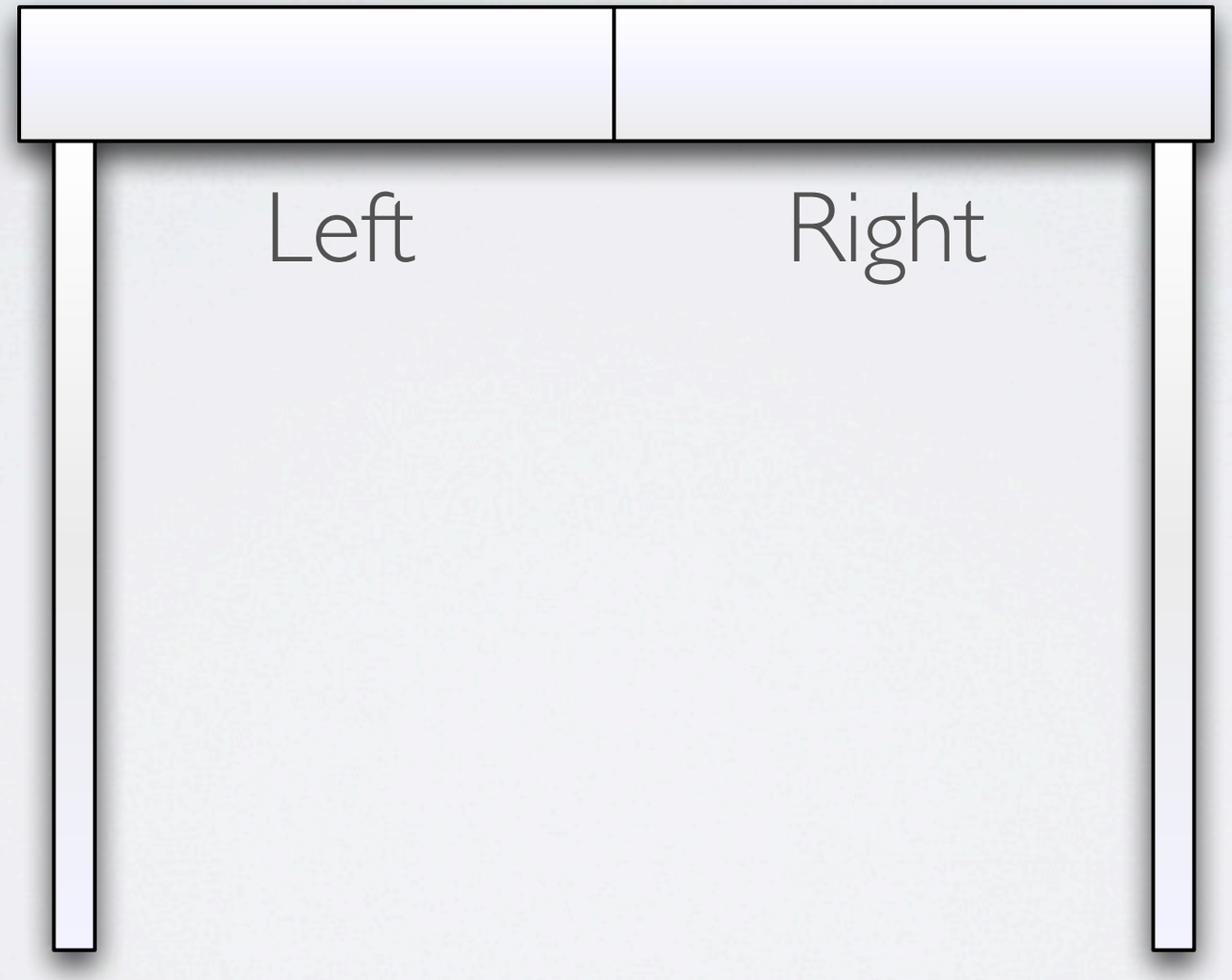
OBSERVABILITY



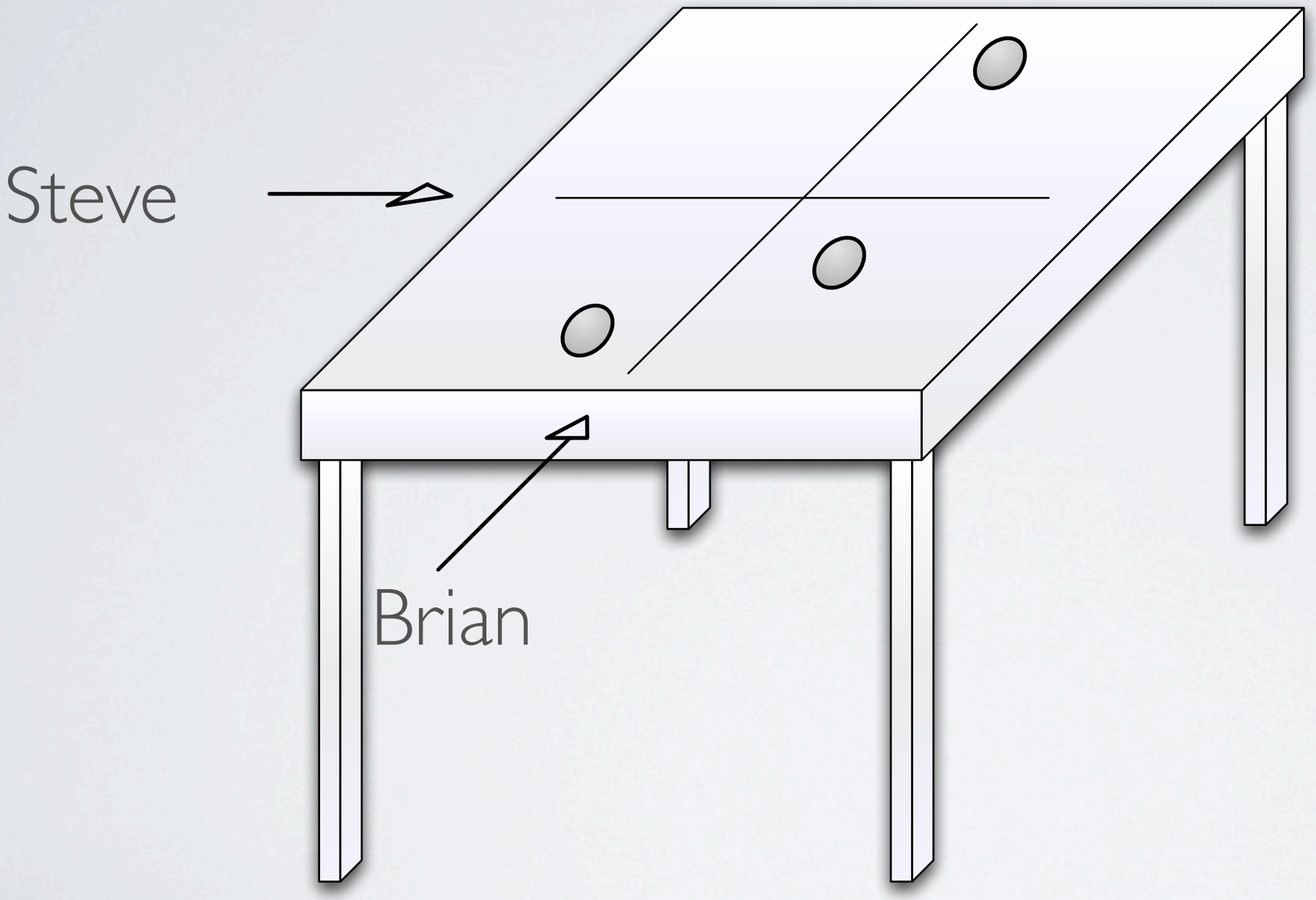
OBSERVABILITY



OBSERVABILITY



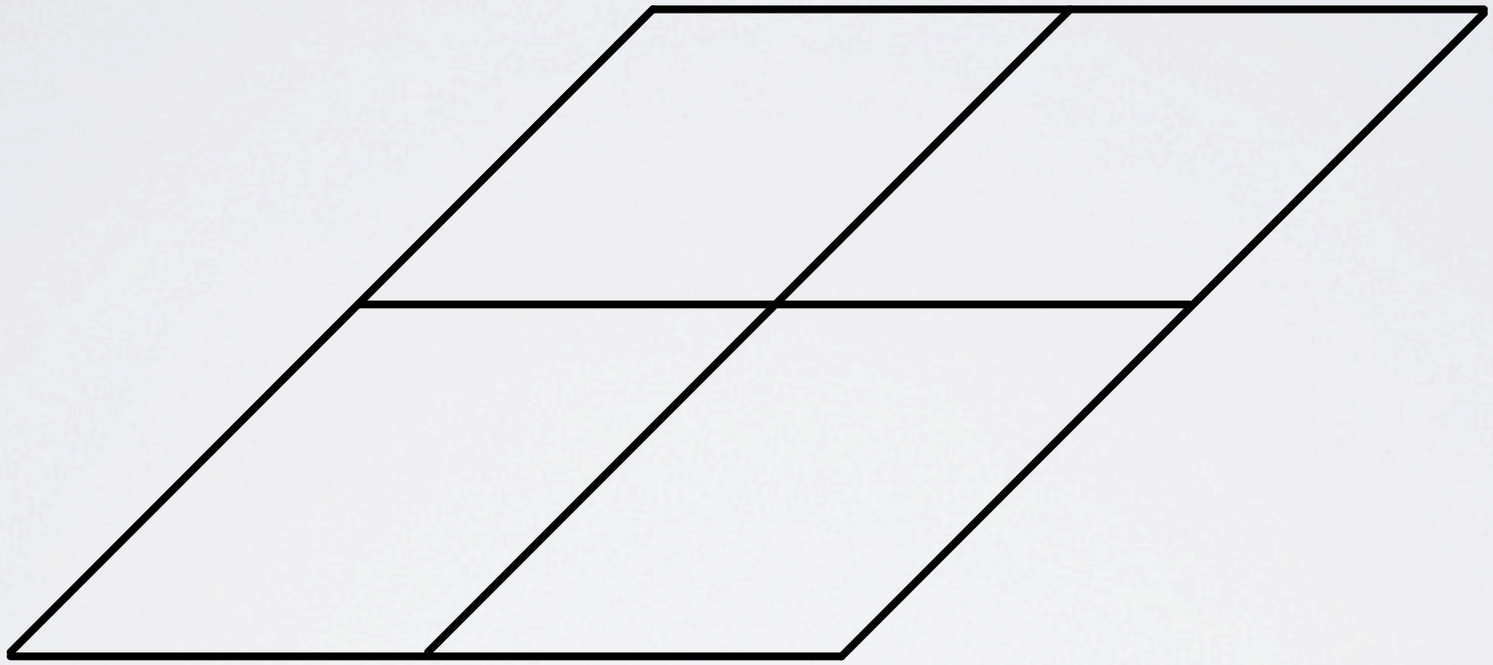
OBSERVABILITY



#	Steve	Brian
1	R	R
2	L	R
3	R	L

STATE SPACE

$X_2 = \{L, R\}$



$X_1 = \{L, R\}$

SUPER-OBSERVER

Has a view which *dominates* the views of all other observers.

SUPER-OBSERVER

There are no one-to-many mappings from the super-observer's states to any other observer's states.

SUPER-OBSERVER

A super-observer is maximally present if it can discriminate among the Cartesian product of all other observations.

Observer	Set of States
Steve	{L, R}
Brian	{L, R}
Super-Observer	{L → B, R → F} × {L, R}

OBSERVABILITY

#	Steve	Brian	Super-Observer
1	R	R	{F, R}
2	L	R	{B, R}
3	R	L	{F, L}

STATE SPACE

Cartesian product of all possible sets of states.

Example

1,000,000 bytes of RAM
8 bits per byte
2 states per bit

8,000,000 dimensions with 2 values each
or
1,000,000 dimensions with 256 values each

STATE SPACE

10,000,000 rows in a table
20 columns

Whole database is a single point in a 200,000,000 dimensional space.

Changes to data are transforms of that point.

State over time is the trajectory of that point.

CONSISTENCY

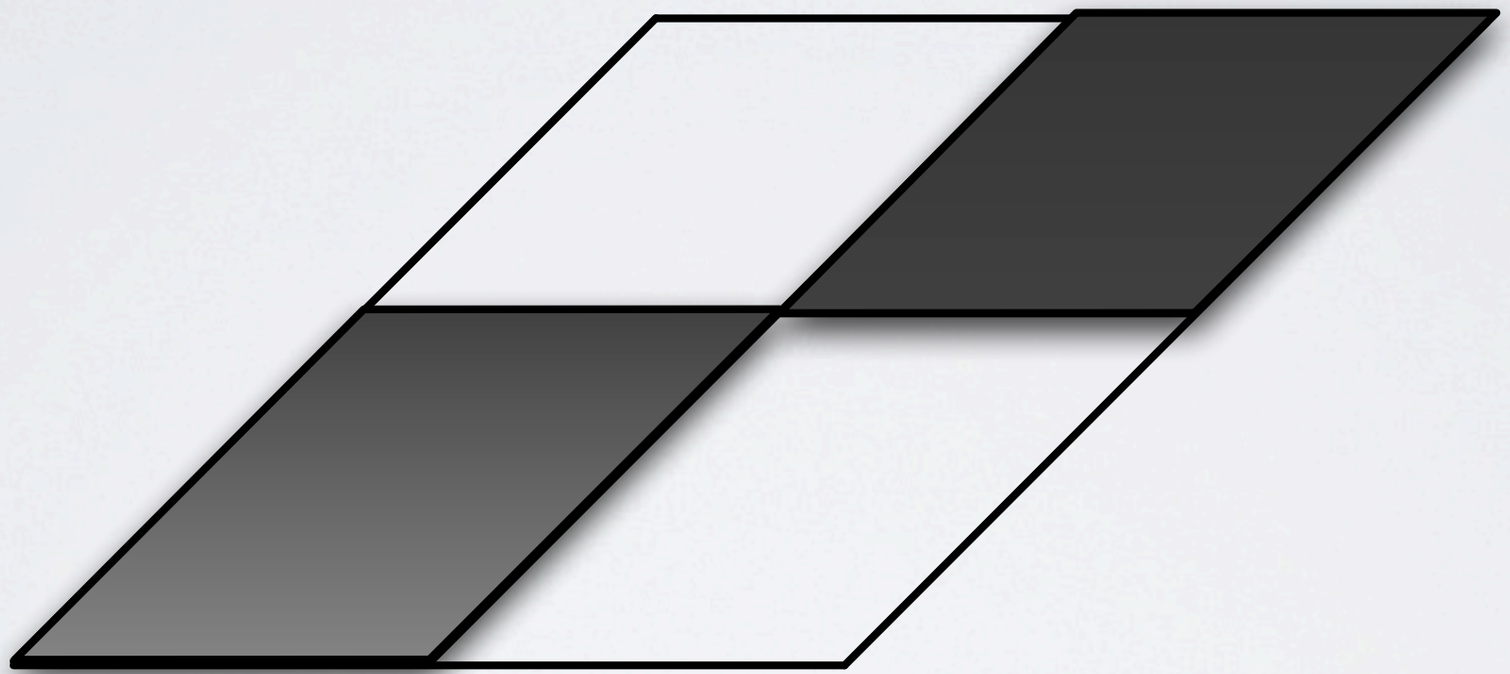
Not every point in state space is allowed.

PORKY PIG

PORKY PIG

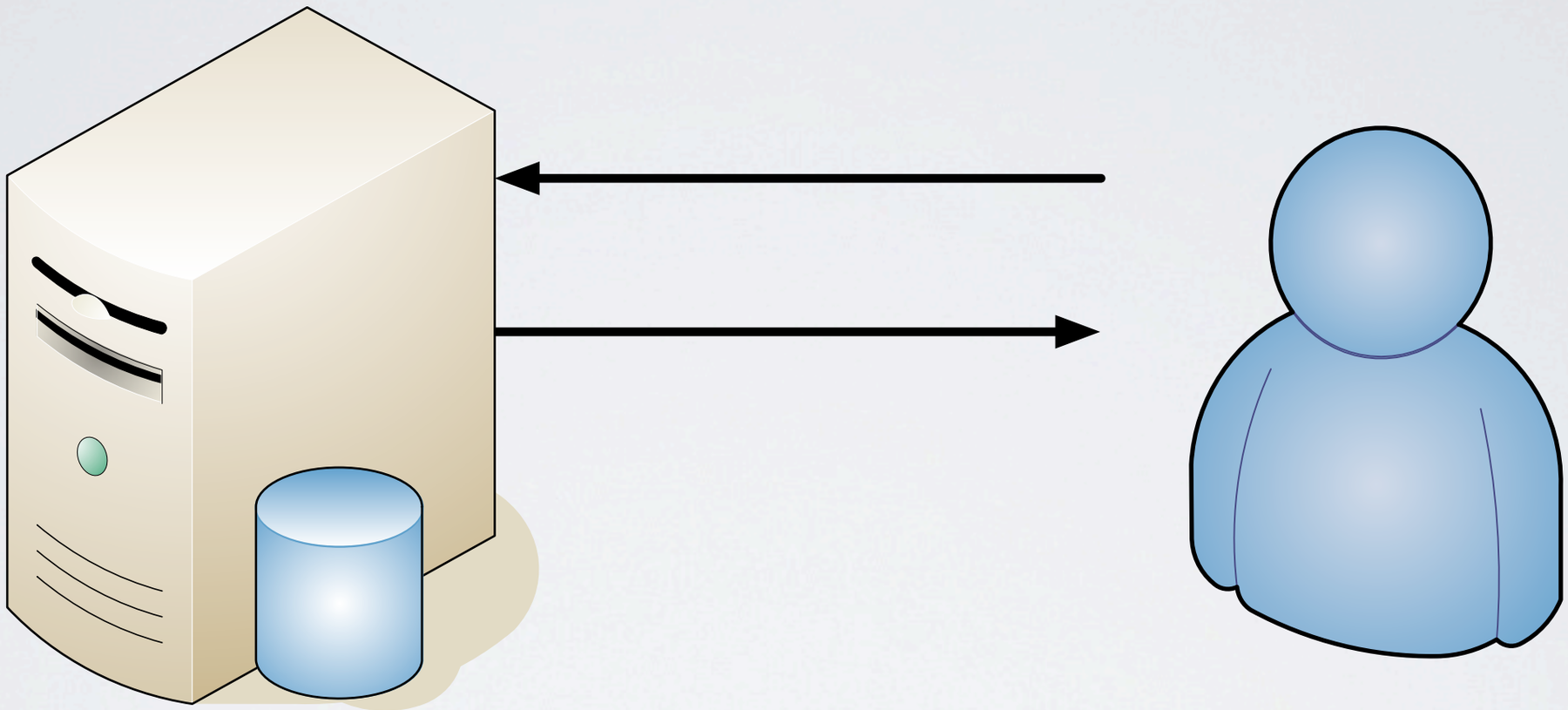
FORBIDDEN STATES

X_2 = {shade open, shade closed}

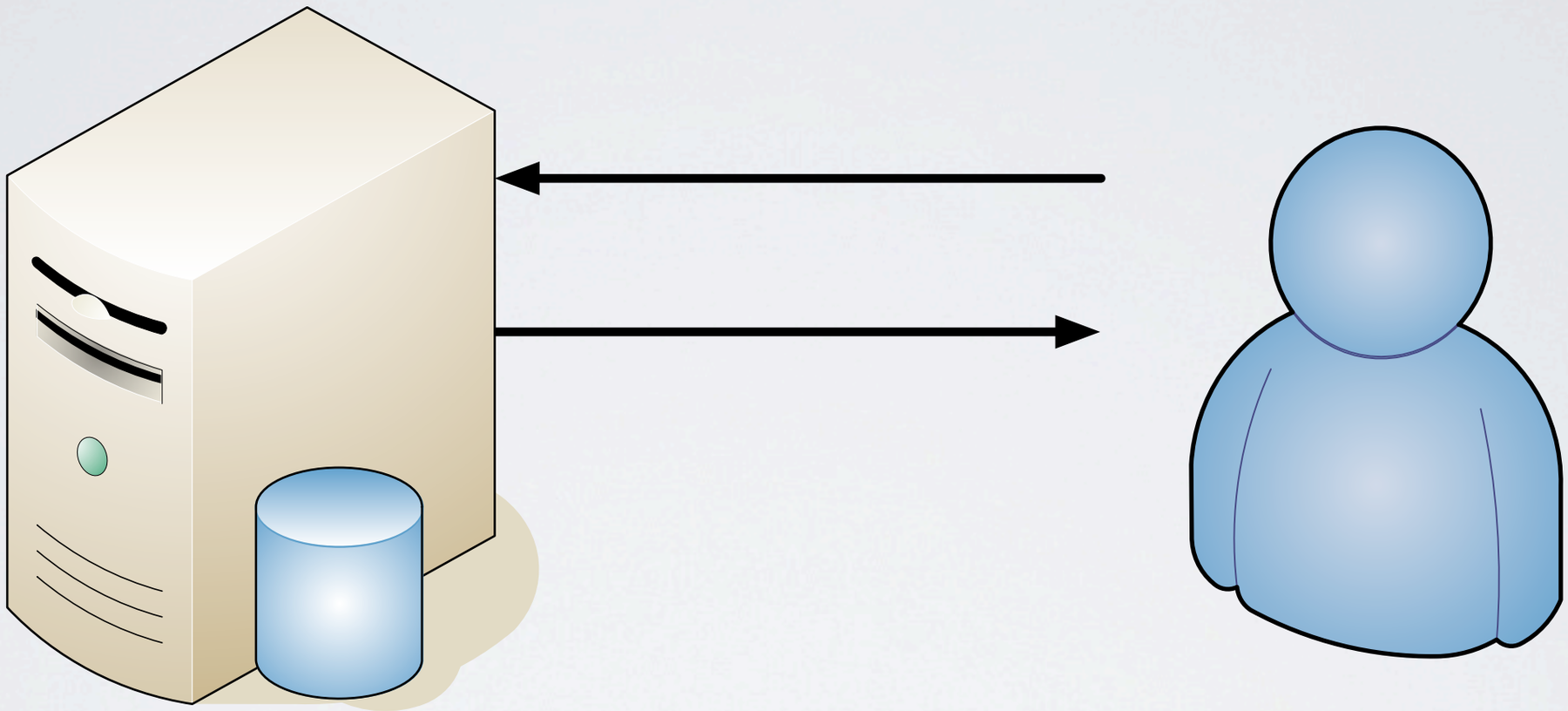


X_1 = {looking, not looking}

BLACK BOX HYPOTHESIS

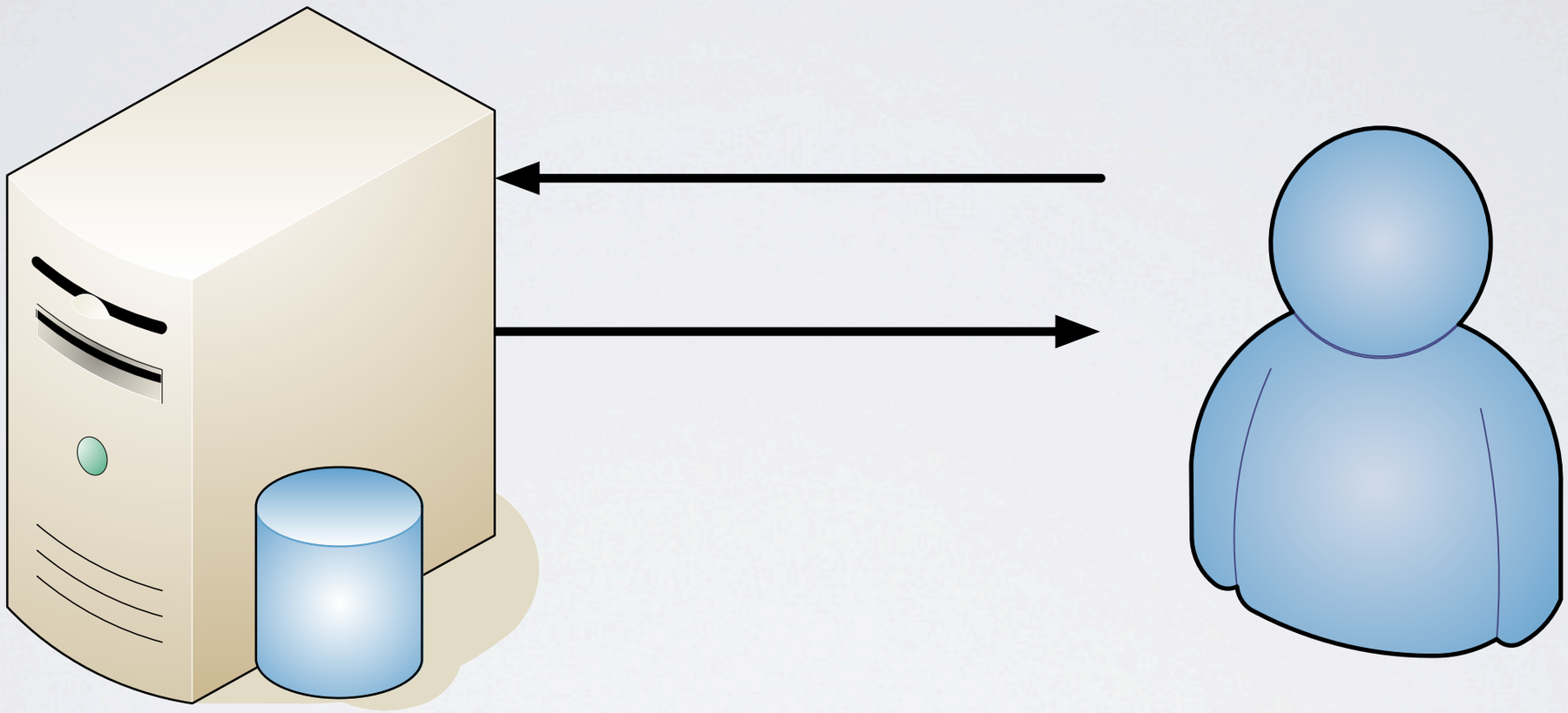


BLACK BOX HYPOTHESIS



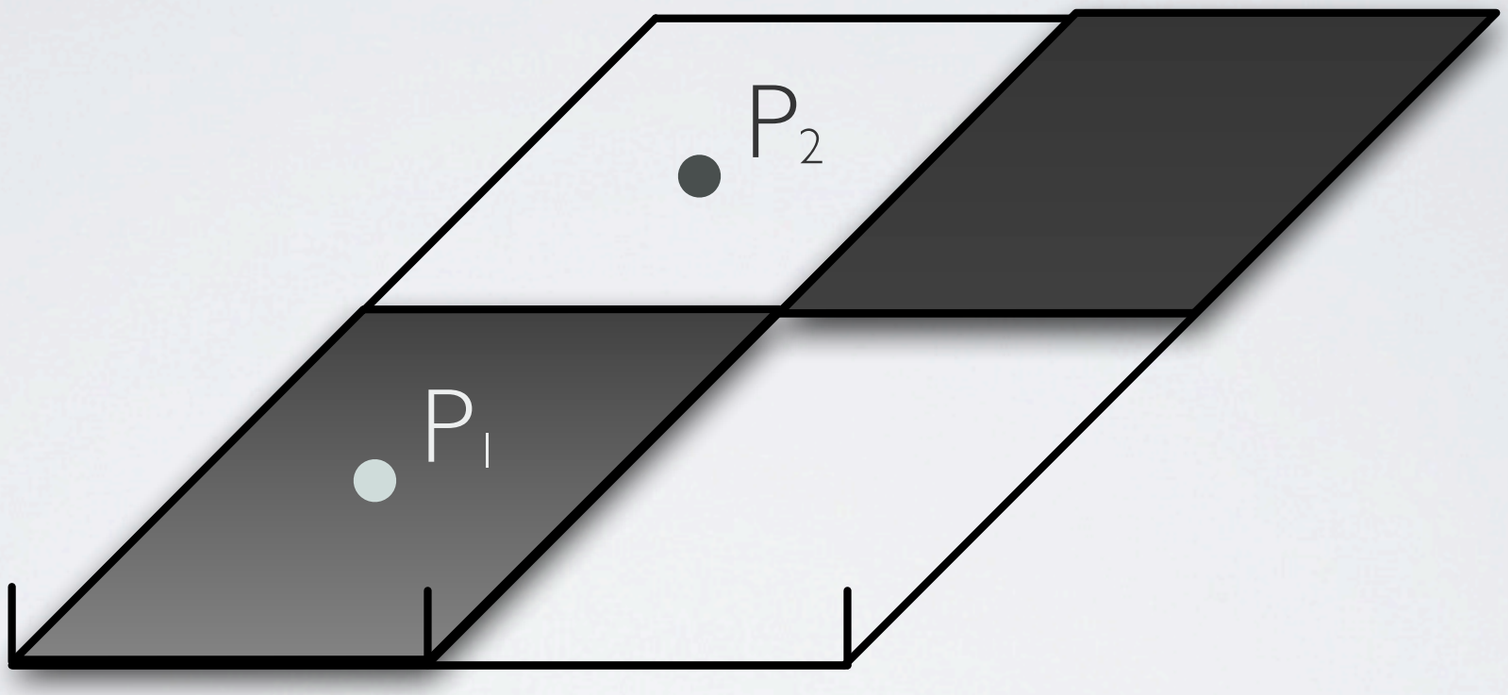
External observers can only ever ask for projections of the state space, at defined points in time.

BLACK BOX HYPOTHESIS



State space trajectories may cross into forbidden states, as long as those are not revealed to observers.

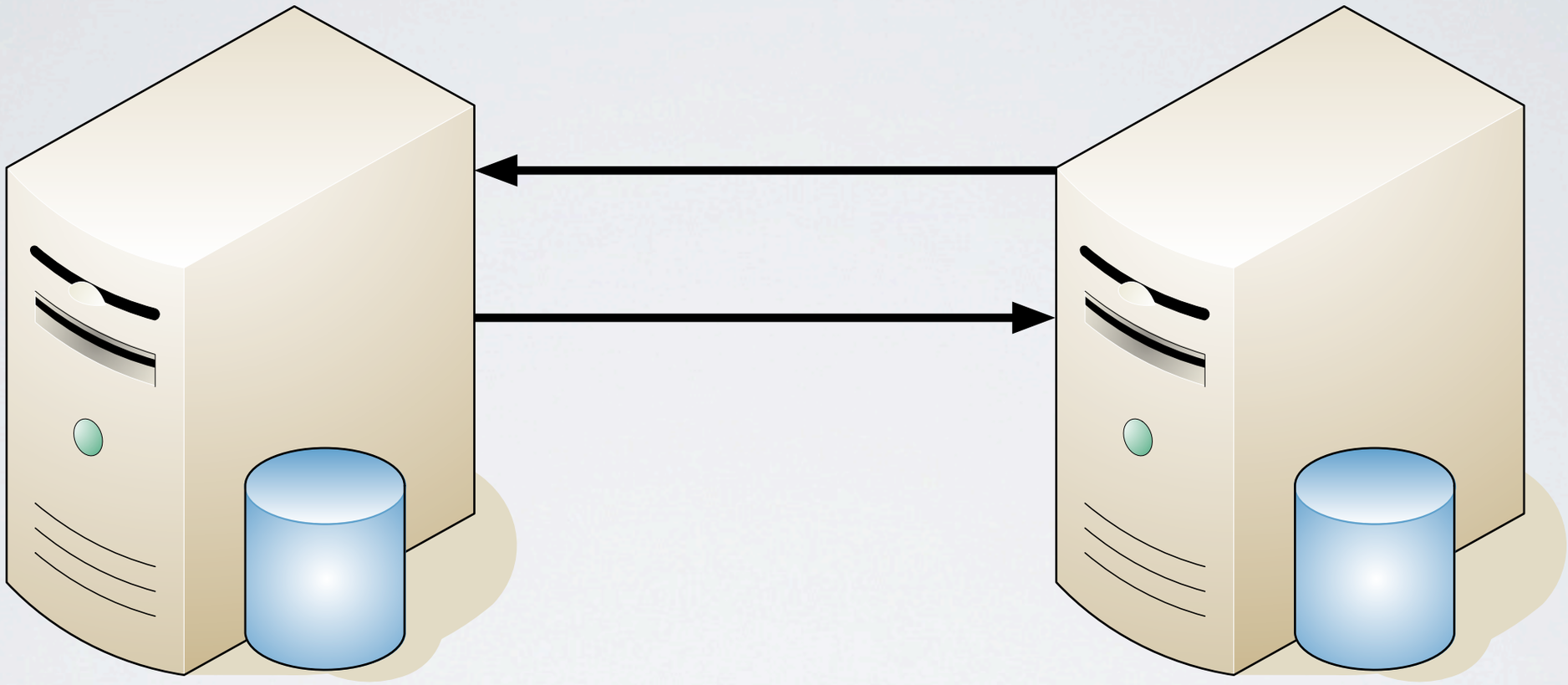
PROJECTION



X_1 = {looking, not looking}

Is Porky looking at the window shade?

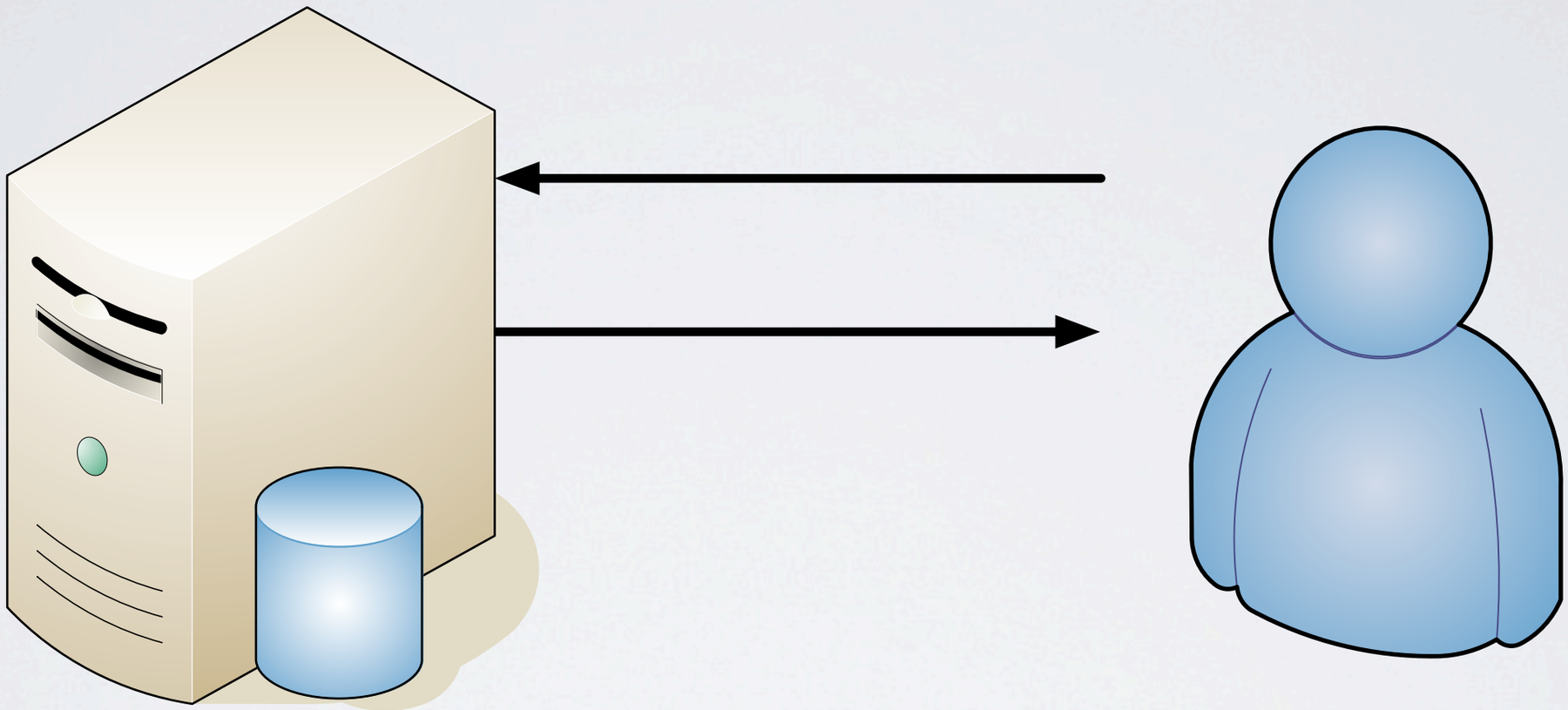
BLACK BOX HYPOTHESIS



Even two clustered machines have their own state spaces.

It's impossible for either to be a superobserver.

OBSERVED CONSISTENCY



Sufficient to ensure that forbidden states cannot be observed.

DOES A SUPEROBSERVER EXIST?

Only if there is exactly one single-threaded CPU, in exactly one computer.

CONSEQUENCES

Consistency doesn't exist in most systems today.

Sometimes we can fake it.

Many times, it doesn't really matter.

WHAT ABOUT CAP?

Consistency:

"...there must exist a total order on all operations such that each operation looks as if it were completed at a single instant."

Seth Gilbert and Nancy Lynch. 2002.
Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.

SIGACT News 33, 2 (June 2002), 51-59. DOI=10.1145/564585.564601
http://doi.acm.org/10.1145/564585.564601

WHAT ABOUT CAP?

Linearizability

Seth Gilbert and Nancy Lynch. 2002.
Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.

SIGACT News 33, 2 (June 2002), 51-59. DOI=10.1145/564585.564601
http://doi.acm.org/10.1145/564585.564601

CONSISTENCY IS NOT BOOLEAN

Eventual

Monotonic read

Read-your-own-writes

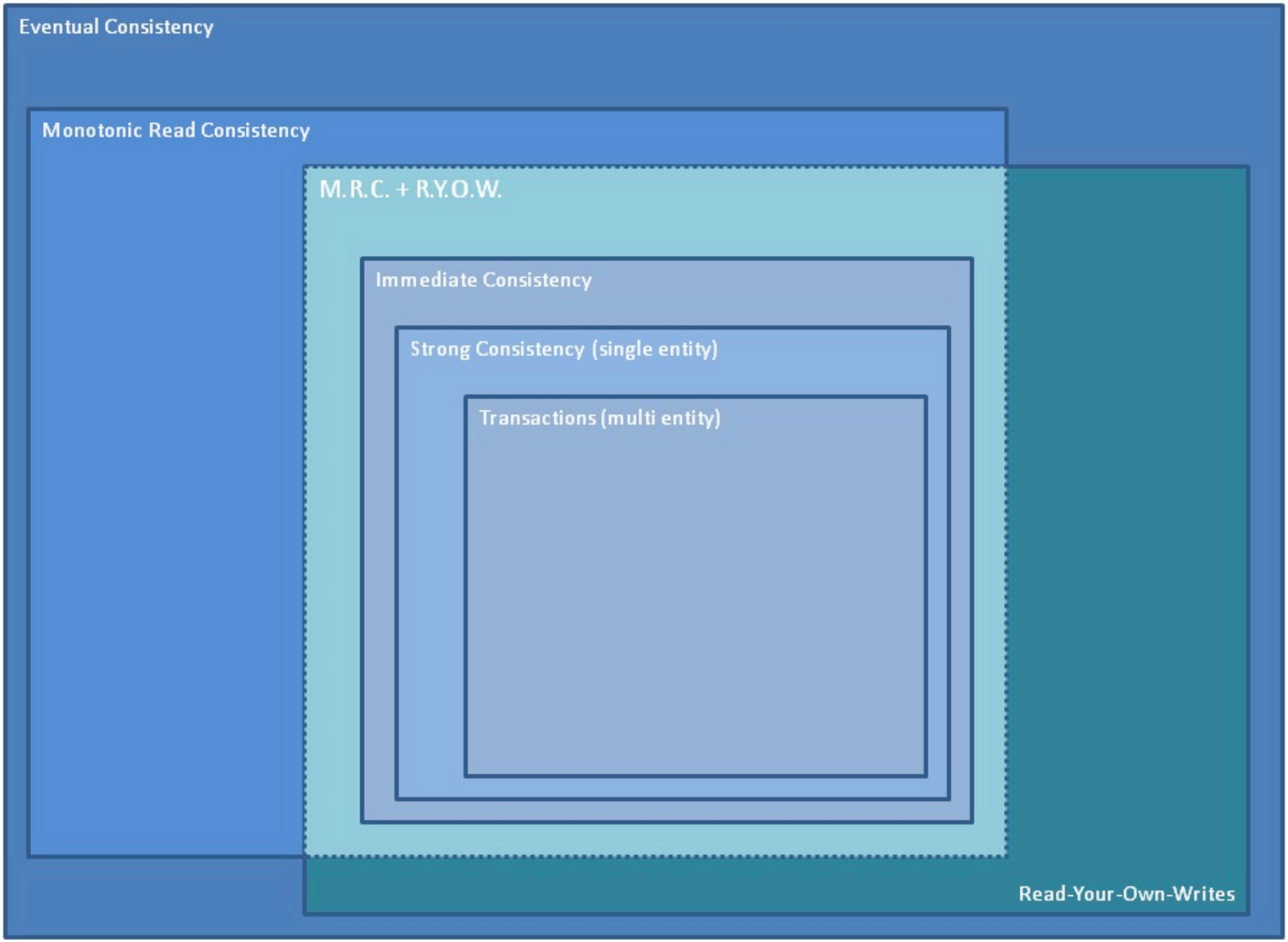
Monotonic + Read-your-own-writes

Immediate

Strong

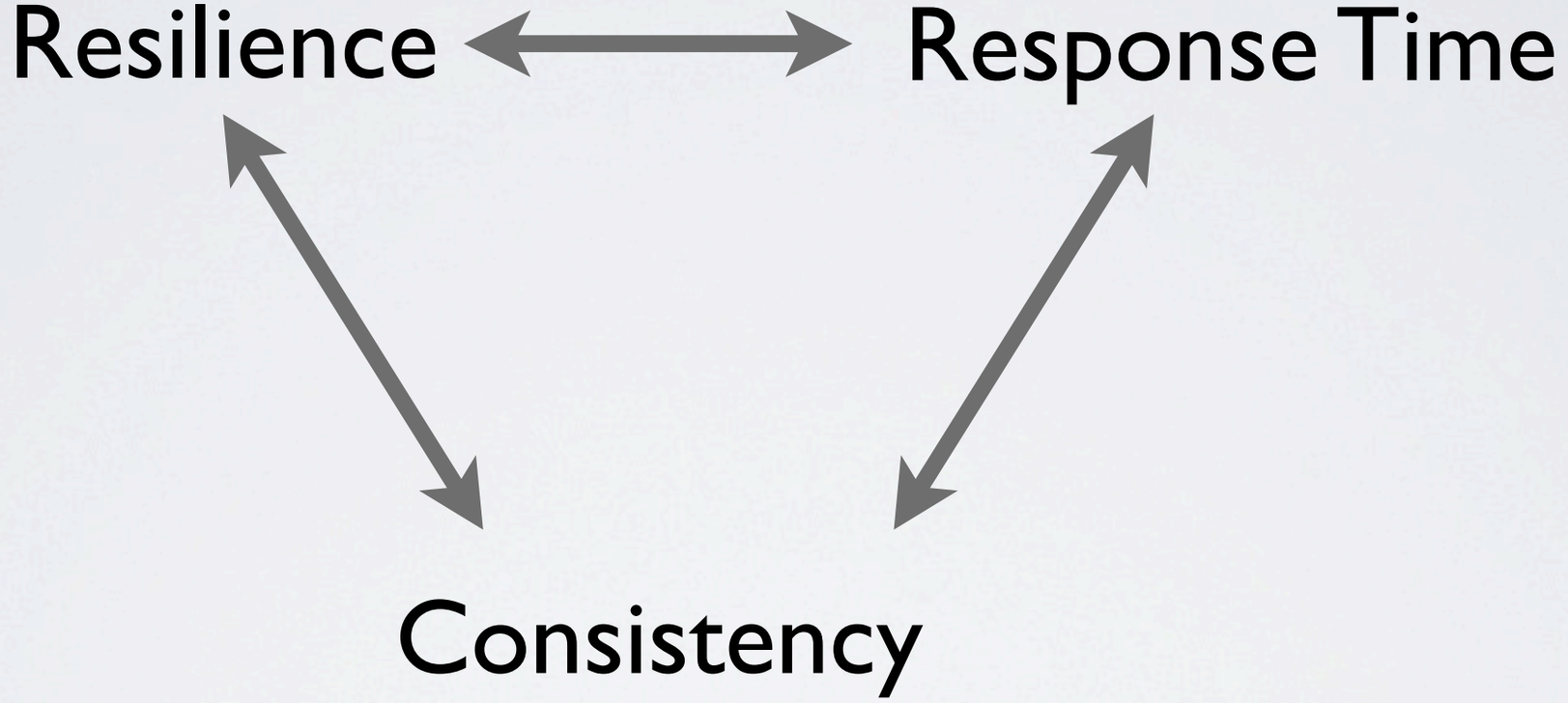
Full transactions

From the MongoDB Blog
http://goo.gl/FQsk → http://blog.mongodb.org/post/523516007/on-distributed-consistency-part-6-consistency-chart



From the MongoDB Blog
http://goo.gl/FQsk

"C" VERSUS "A"?



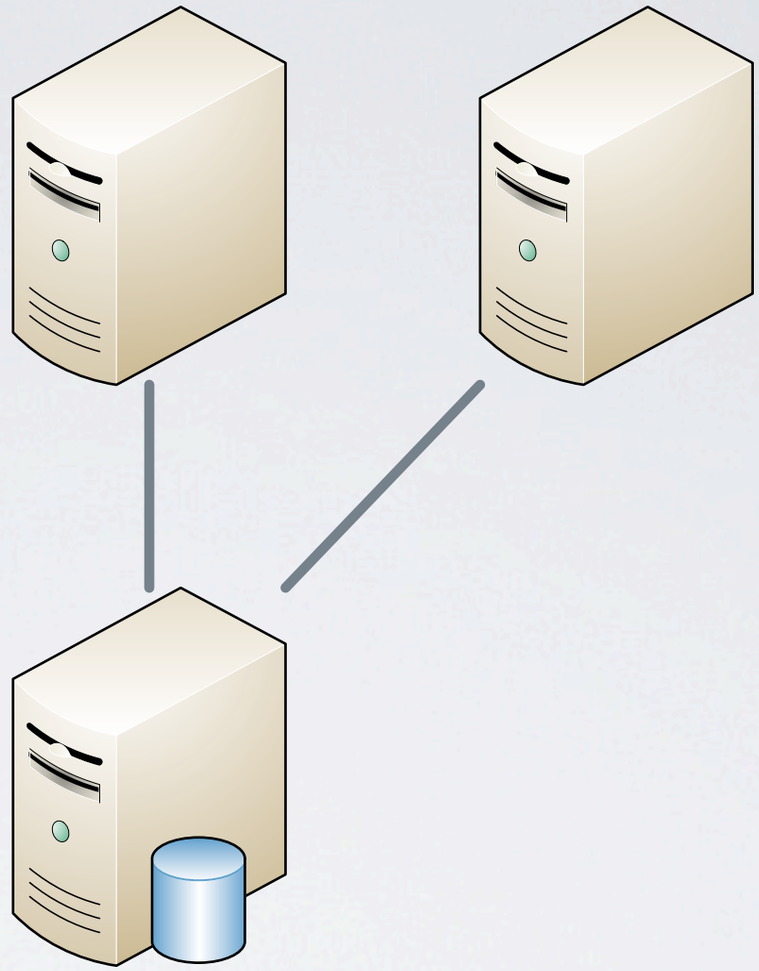
See also: http://goo.gl/1Yv3

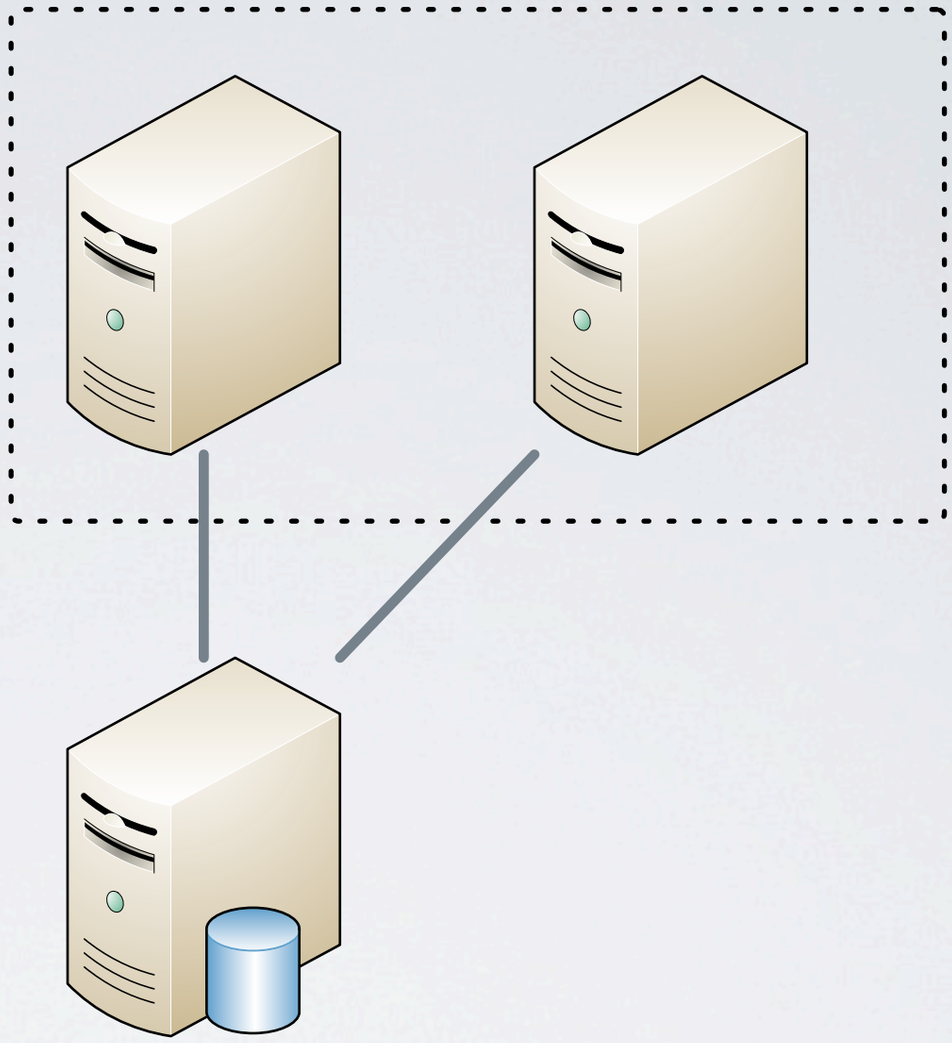
→ http://dbmsmusings.blogspot.com/2010/04/problems-with-cap-and-yahoos-little.html

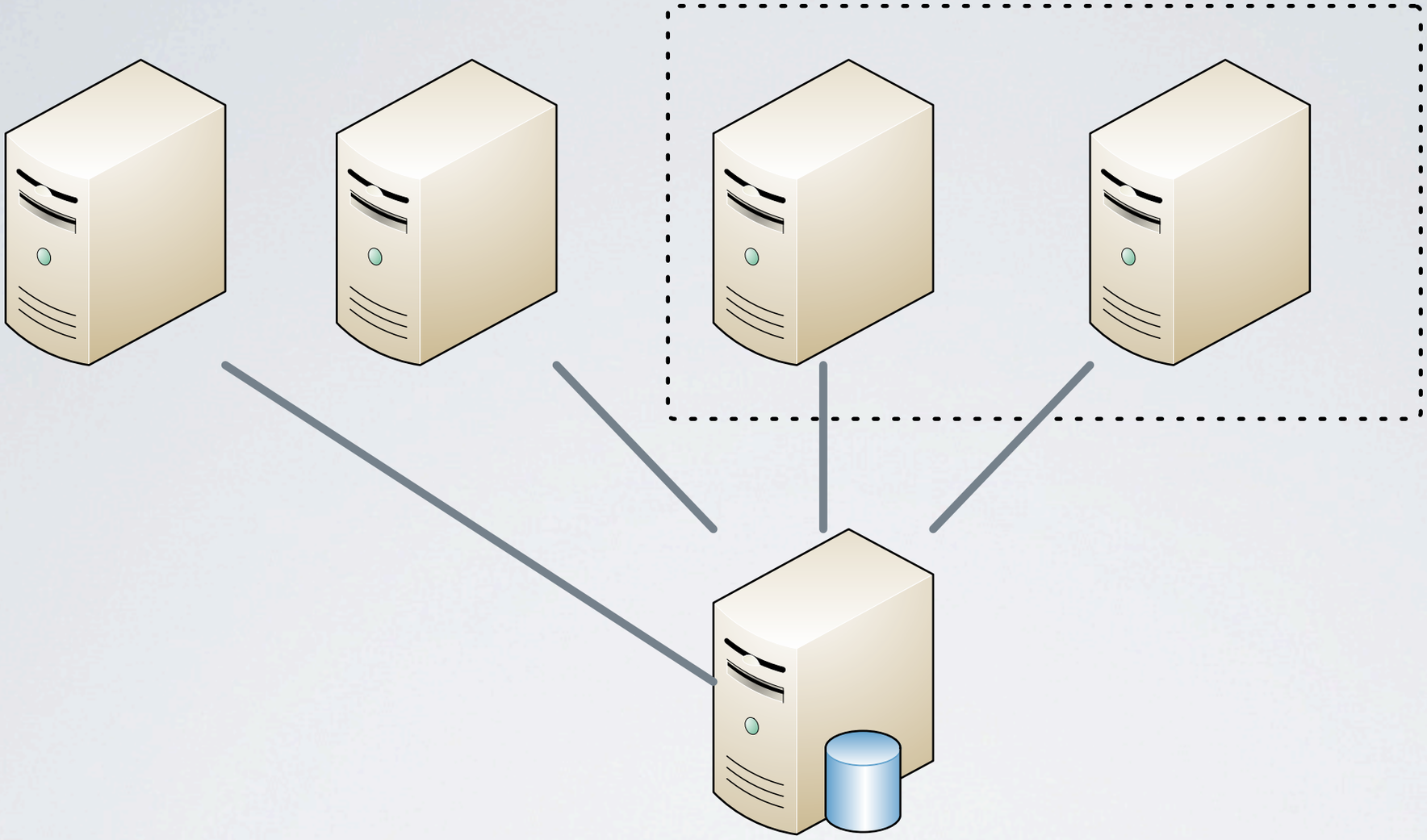
HOW LONG?

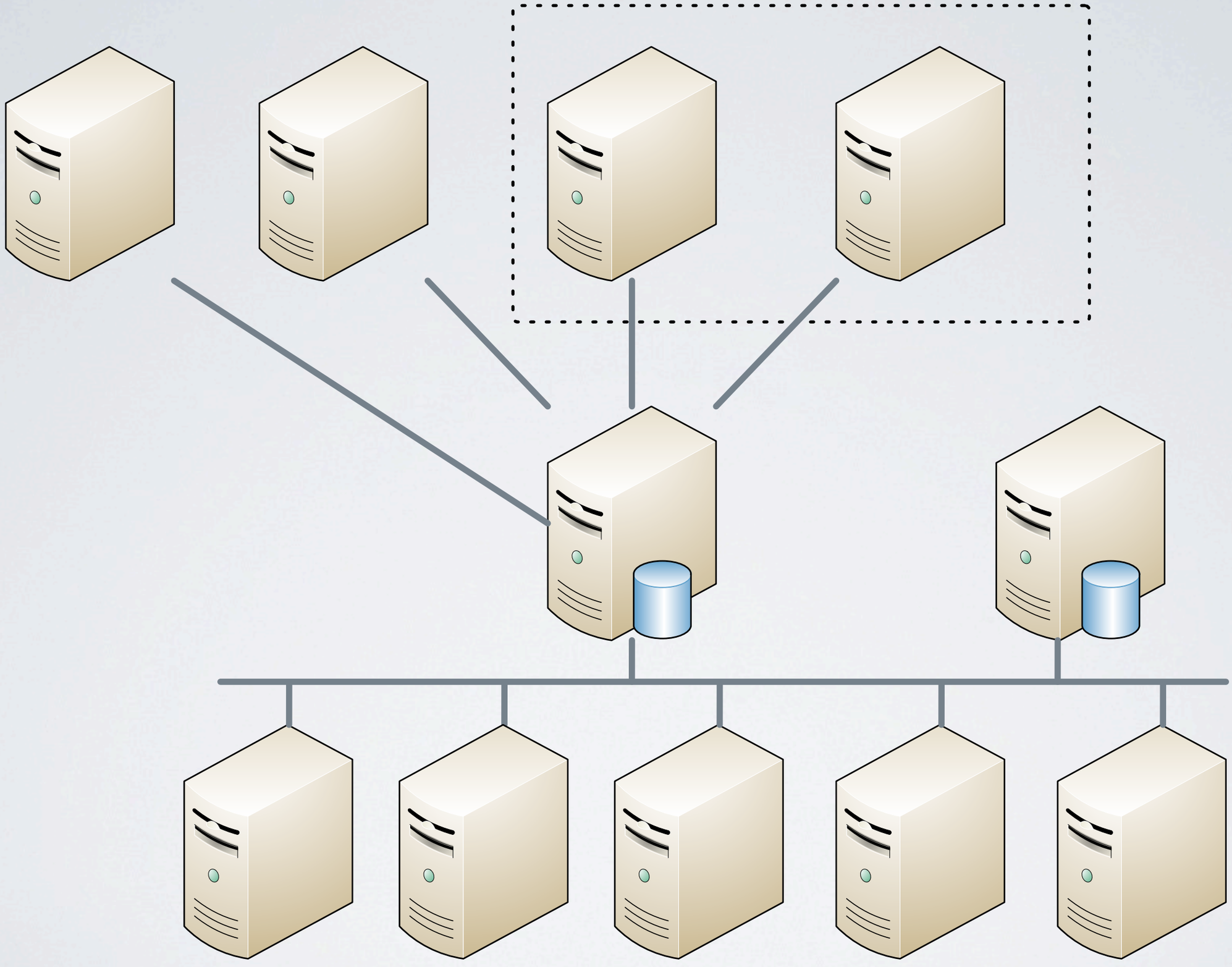
On Lifecycles and Lifespans

Wednesday, October 6, 2010

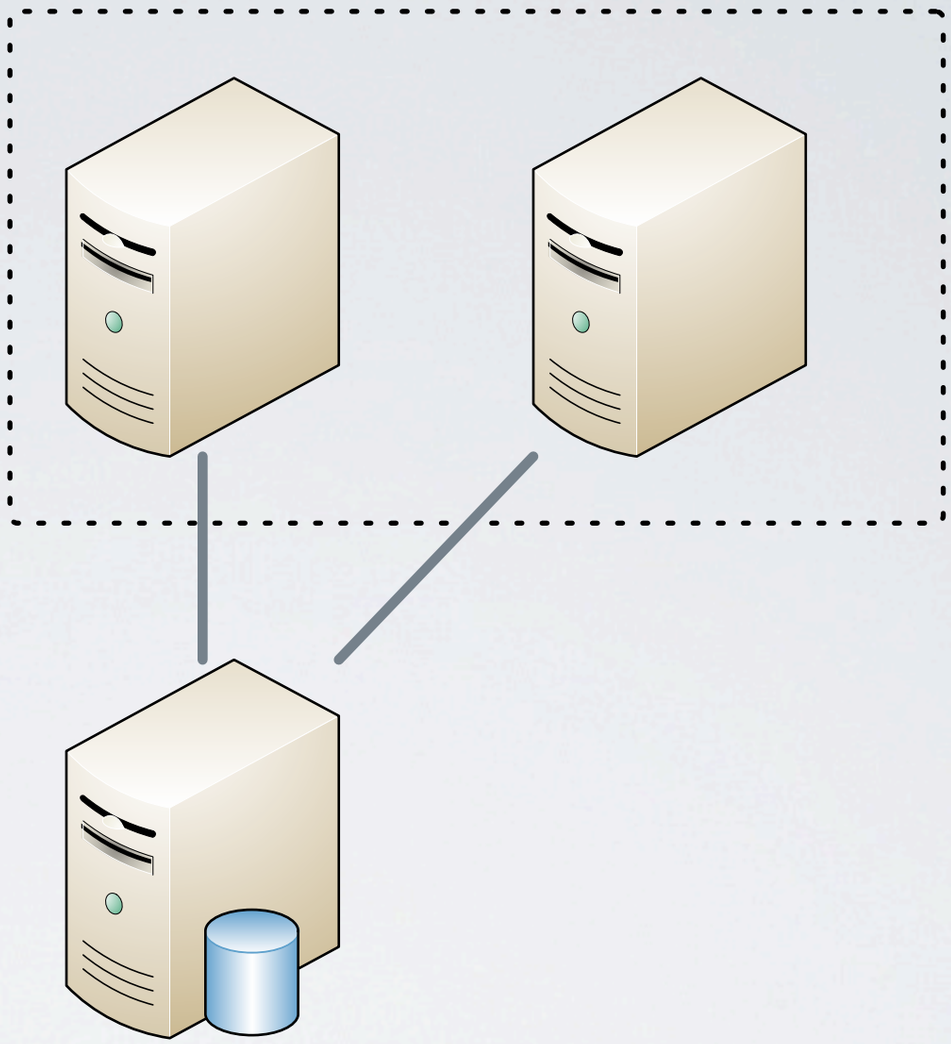


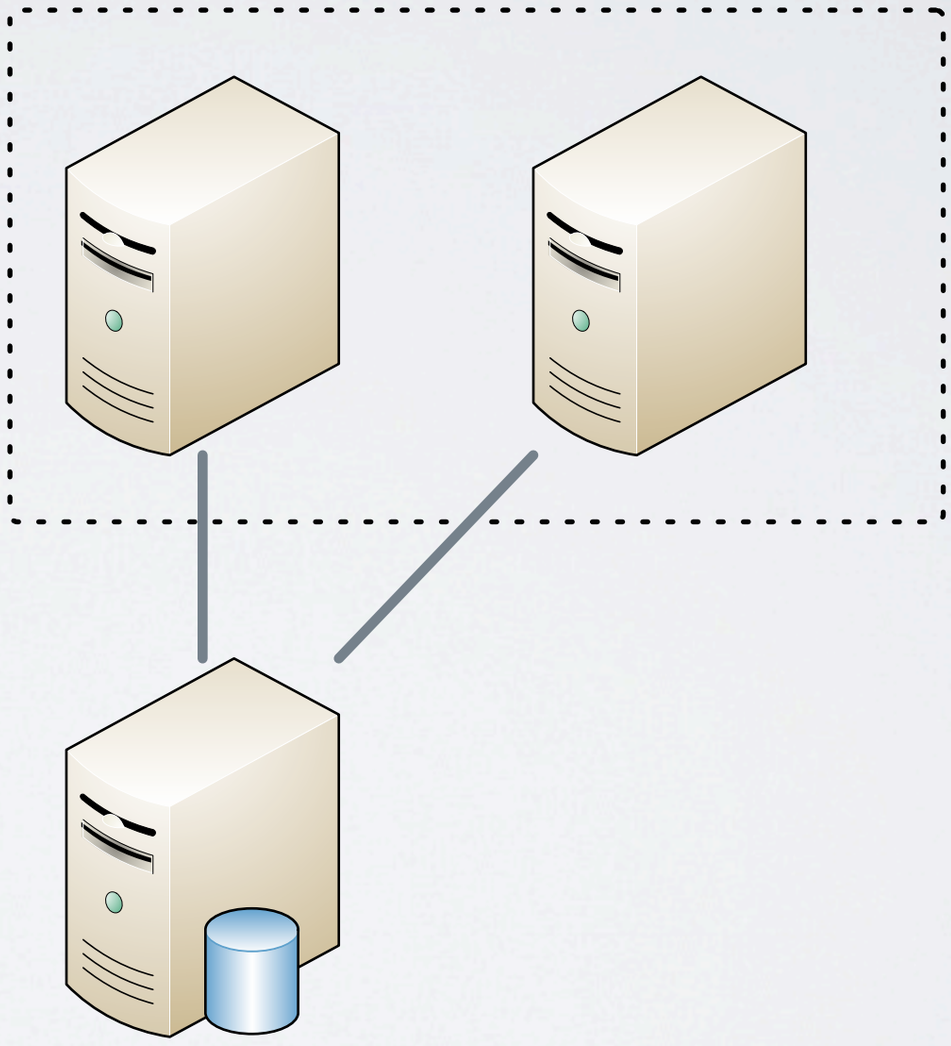


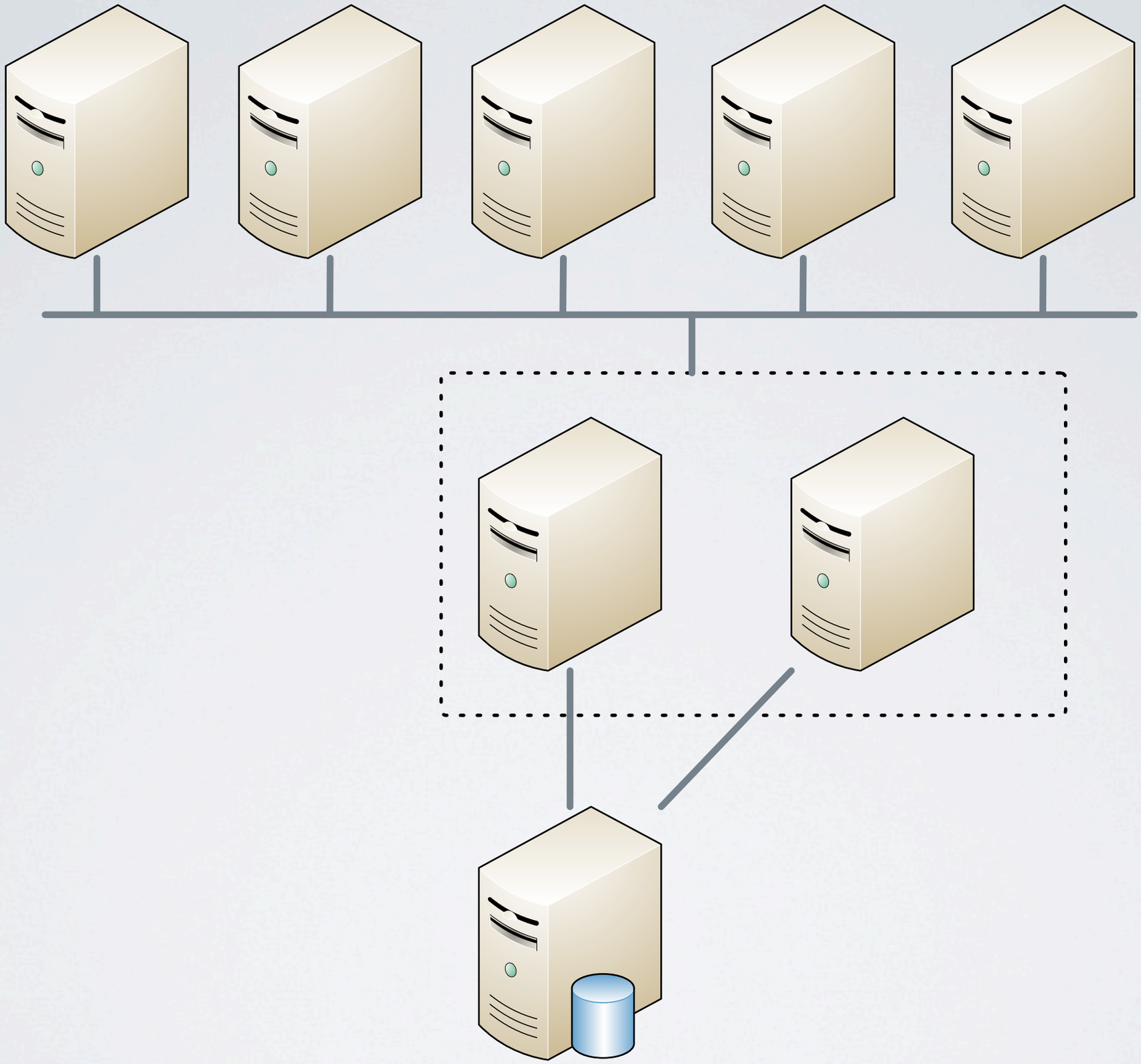




Wednesday, October 6, 2010







DOWN WITH THE IRON FIST

Throw out the DBAs

Throw out the schemas

Unstructured

Semi-structured

DOWN WITH THE IRON FIST

Put the application in charge.

DOWN WITH THE IRON FIST

but...

DIFFICULTIES

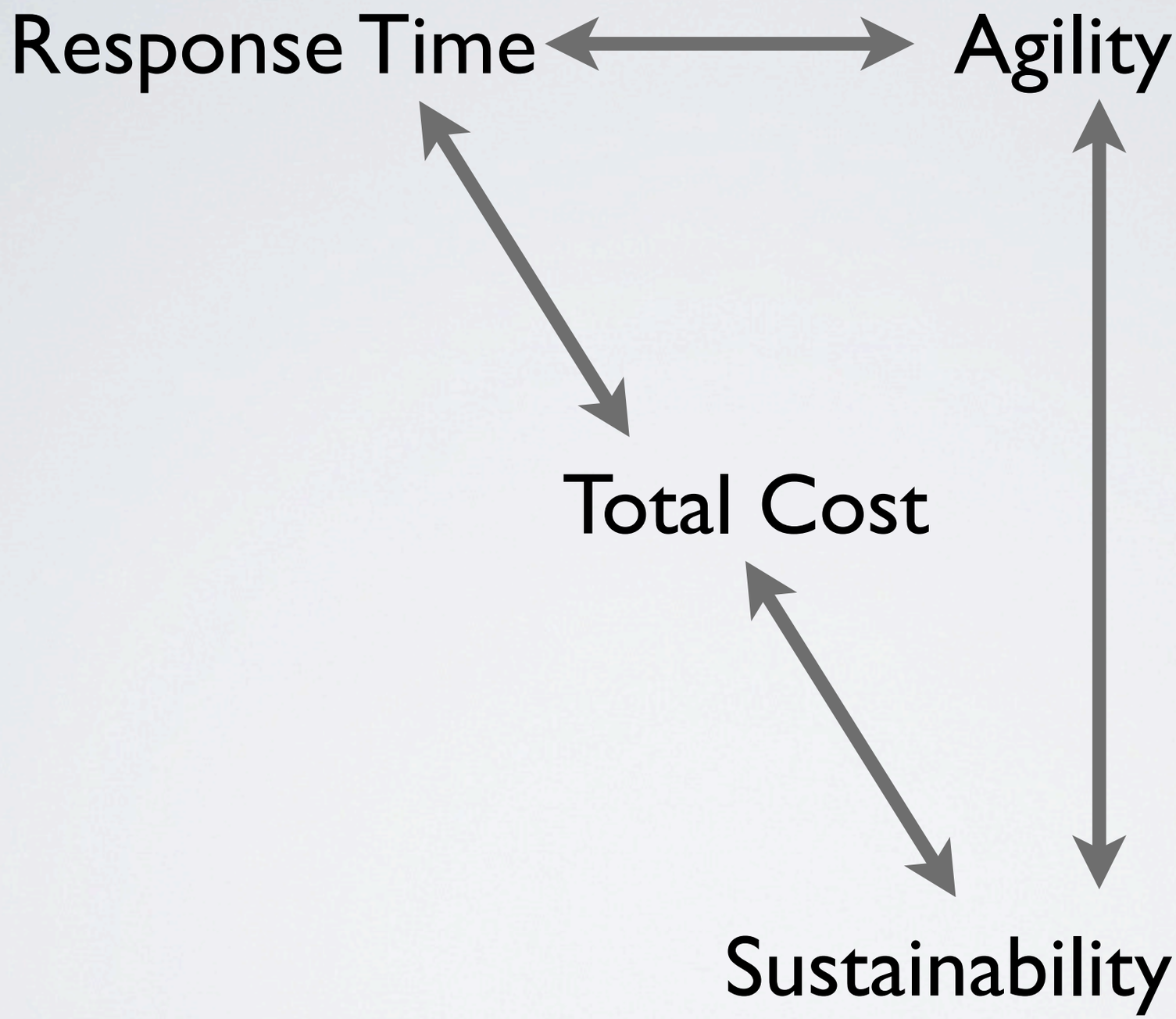
Application versions

Validating correct behavior

Capturing knowledge about that behavior

UGLY TRUTH

Data routinely outlives applications.


Response Time
Agility
Total Cost
Sustainability

WHERE NOW?

WHERE TO PUT YOUR DATA?

Data exists everywhere.

WHERE TO PUT YOUR DATA?

Nothing lasts forever.

WHERE TO PUT YOUR DATA?

Understand freshness.

WHERE TO PUT YOUR DATA?

Engineer a good response time distribution.

WHERE TO PUT YOUR DATA?

Select the consistency model you need.

WHERE TO PUT YOUR DATA?

Be agile and adaptable.

WHERE TO PUT YOUR DATA?

Make it sustainable.

Michael T. Nygard
michael.nygard@n6consulting.com
@mtnygard