

# Embracing Concurrency at scale

(it's about time!)



Justin Sheehy  
[justin@basho.com](mailto:justin@basho.com)

# Concurrency Matters

- distributed data storage
- reliable systems, unreliable components
- your applications

# New Problems, Old Solutions

Distributed Systems matter now more than ever!

As people depend on the Web for more of their needs,  
they expect it to be dependable.

# New Problems, Old Solutions

Distributed Systems matter now more than ever,  
and we must learn from the past to build the future.

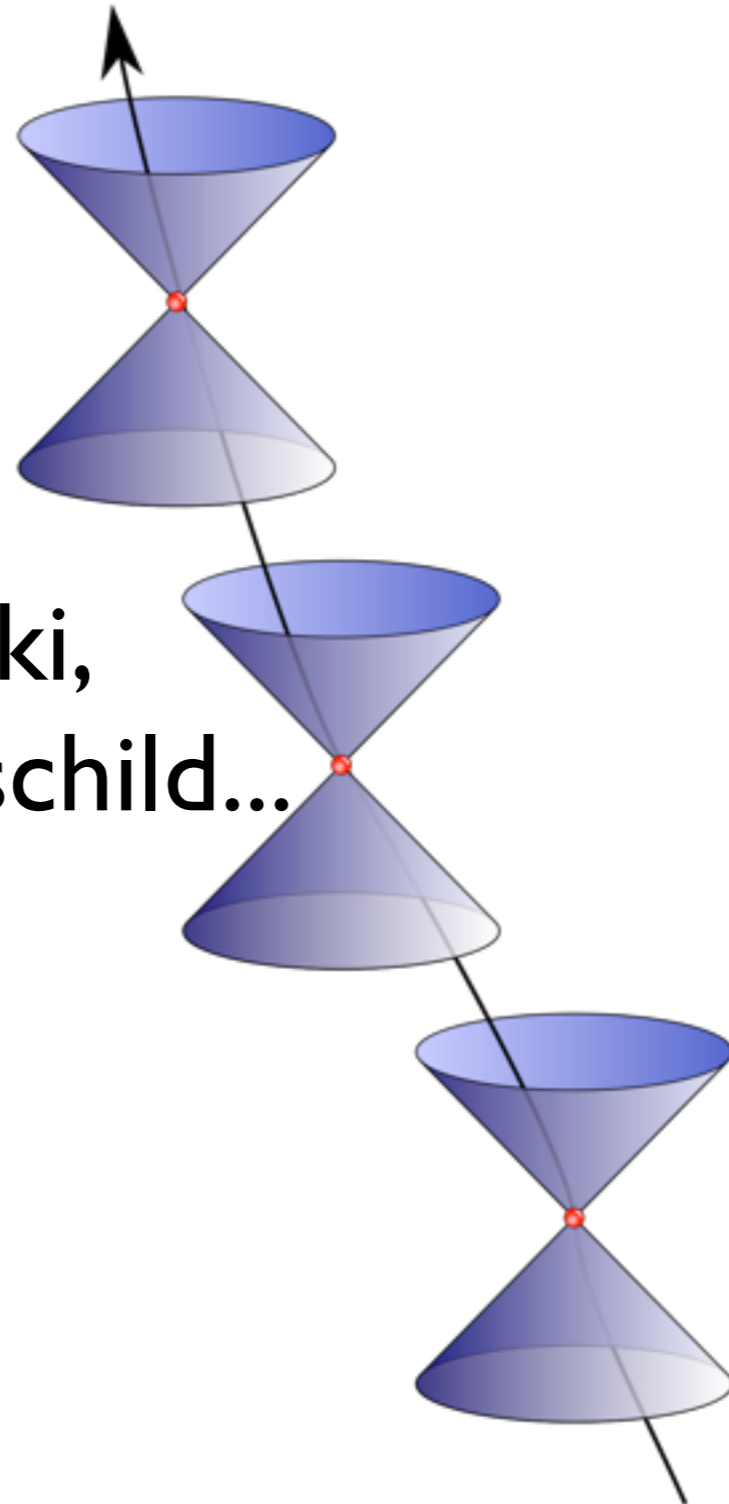
# What is Concurrency?

**concurrent:** occurring at the same time

**concurring:** agreeing with others

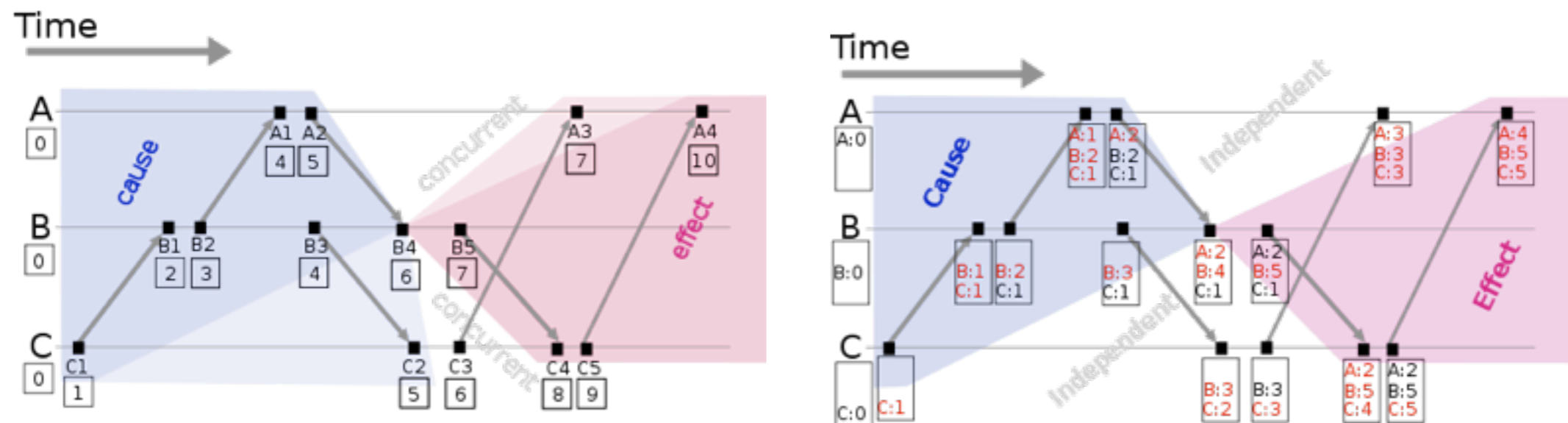
# Time is a Hard Problem

Einstein,  
Minkowski,  
Schwarzschild...



# Time in Computing

Lamport, 1978 -- gave us “happened before”



Mattern, 1989 -- closer to Minkowski causality

# Time is a Hard Problem

In computing, we like to pretend it's easy.

This is a trap!

# Distributed Computing is Asynchronous Computing

Synchrony (distributed transactions) throws away  
the biggest gains of being distributed!

**There is no “Global State”**

**You only know about the past -- deal with it!**

# digression: crypto protocols

A

B

$\{N_a, A, D\}_{K_B}$



$\{N_a, SK\}_{K_A}$



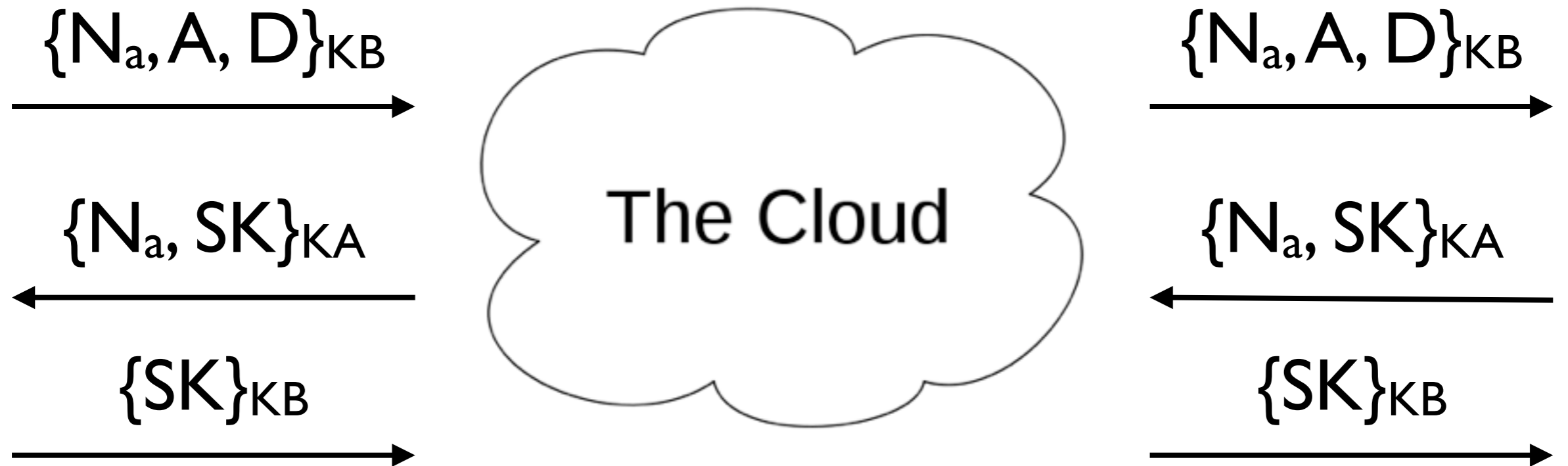
$\{SK\}_{K_B}$



# digression: crypto protocols

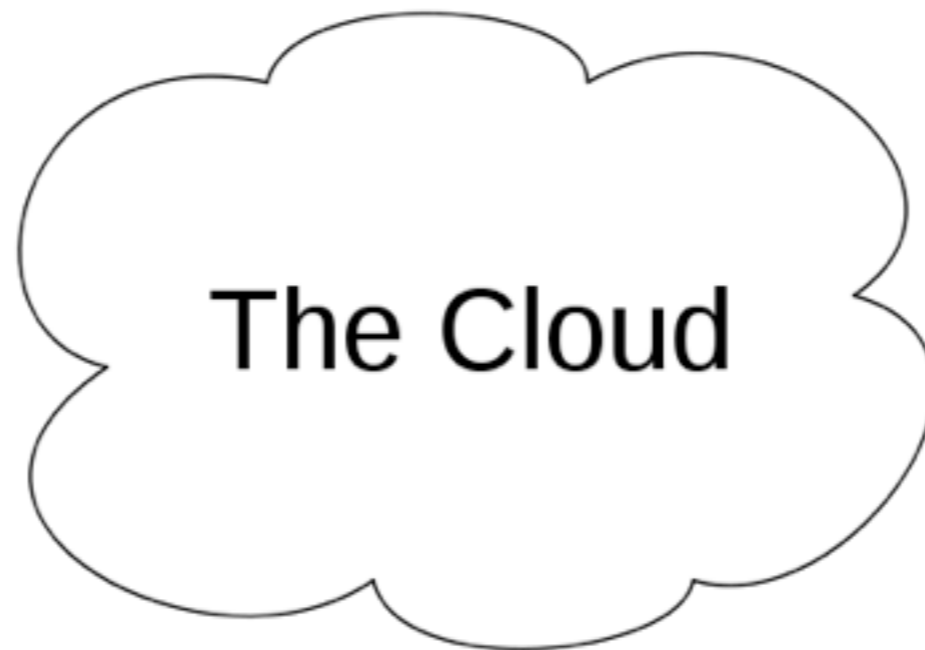
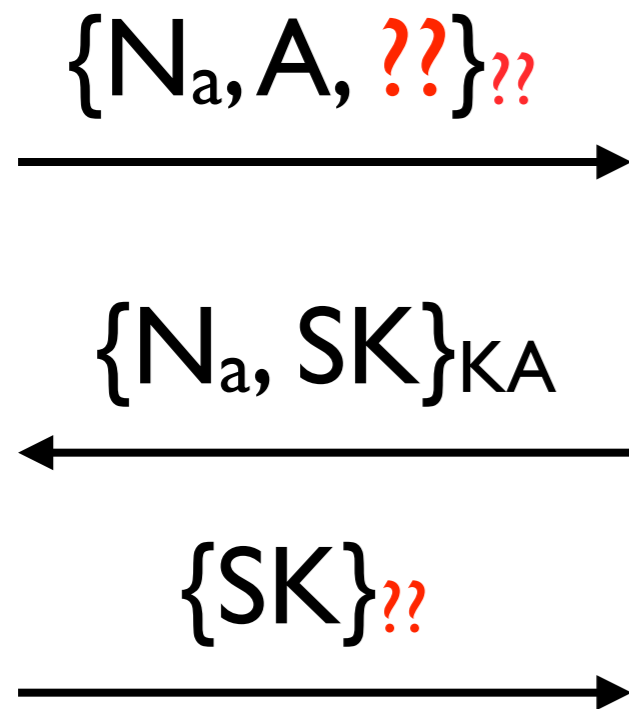
A

B

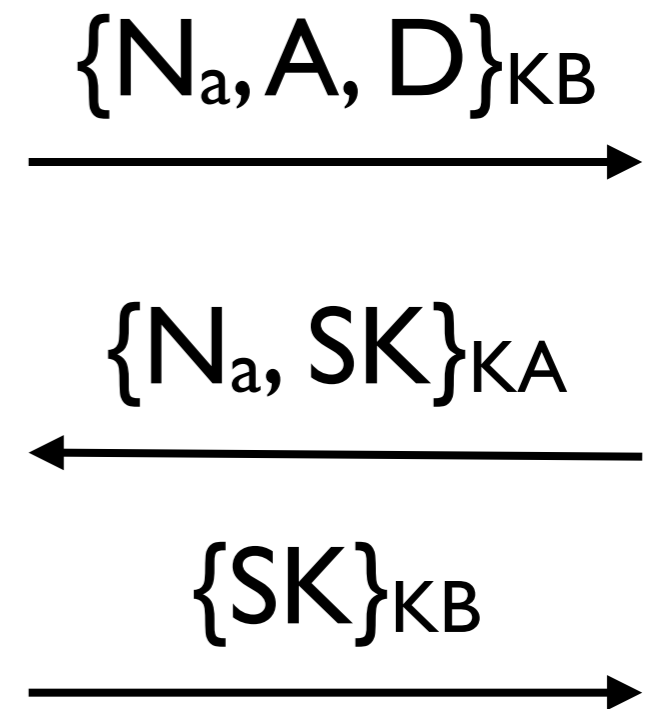


# digression: crypto protocols

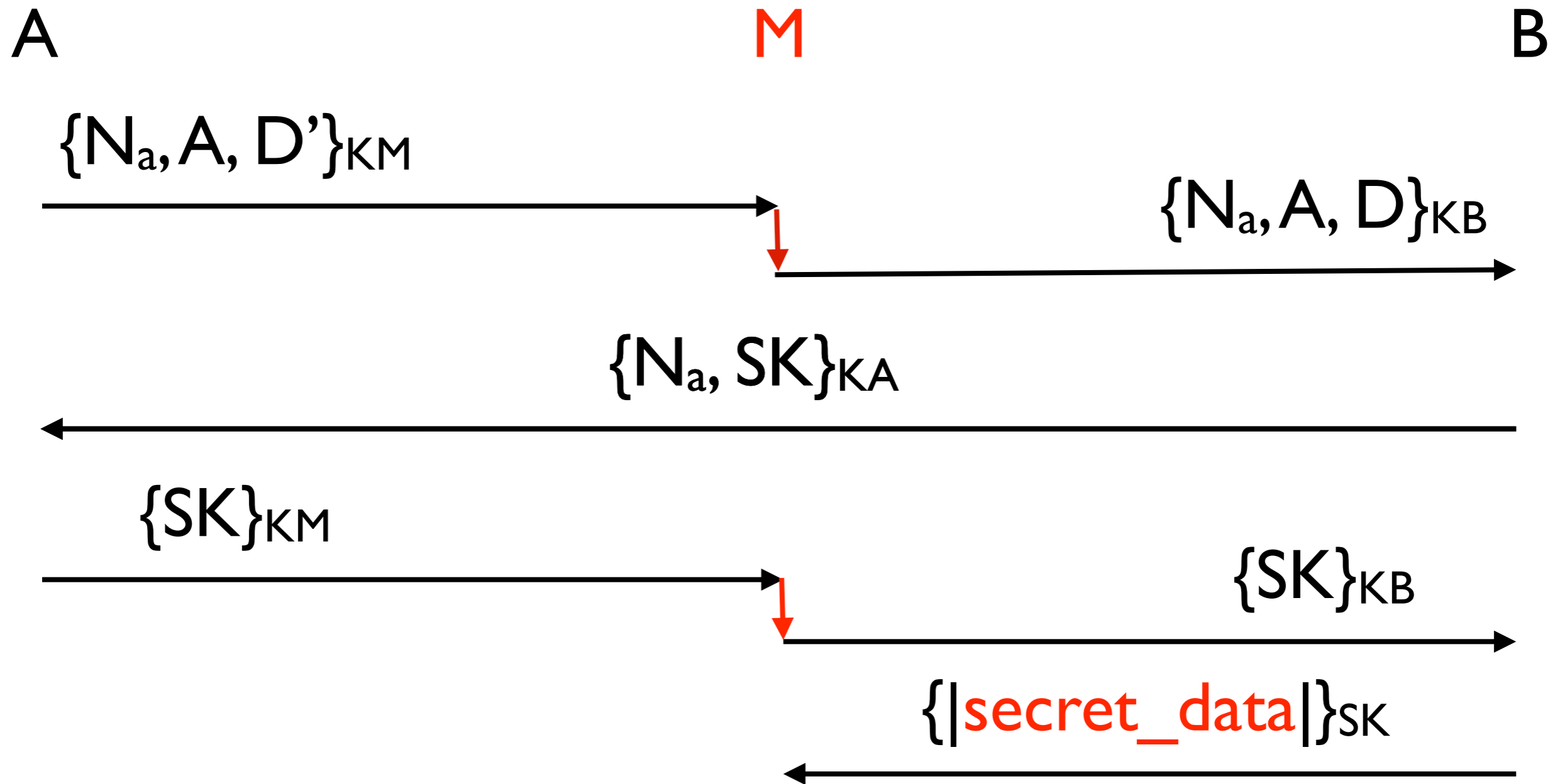
A



B



# digression: crypto protocols



(Gavin Lowe, 1995)

# There is no “Global State”

You only know about the past -- deal with it!

This means giving up on ACID.

# Living without ACID?

This is going to hurt!

(but it might be worth it, sometimes)

**A**tomicity  
**C**onsistency  
**I**solation  
**D**urability

~~Atomicity~~

Consistency

~~Isolation~~

Durability

~~Atomicity~~

Consistency

~~Isolation~~

Durability

Basically  
Available

~~Atomicity~~

Consistency

~~Isolation~~

~~Durability~~

Basically  
Available

~~Atomicity~~

Consistency

~~Isolation~~

~~Durability~~

Basically

Available

Soft State

~~Atomicity~~  
~~Consistency~~  
~~Isolation~~  
~~Durability~~

**B**asically  
**A**vailable  
**S**oft State

~~Atomicity~~  
~~Consistency~~  
~~Isolation~~  
~~Durability~~

Basically  
Available  
Soft State  
Eventually-Consistent

This is a real tradeoff -- if you make it, understand it!

(Eric Brewer, 1997)

**B**asically

**A**vailable

**S**oft State

**E**ventually-Consistent

# CAP tradeoffs

**C**onsistency

**A**vailability

**P**artition-Tolerance

You want all three, but  
you can't have them all at once.

# CAP tradeoffs

**C**onsistency

**A**vailability

**P**artition-Tolerance

**Distributed Transactions**

**(on any real network, this fails)**

# CAP tradeoffs

**C**onsistency

Availability

**P**artition-Tolerance

Quorum Protocols &

typical Distributed Databases

(nodes outside the quorum fail)

# CAP tradeoffs

Consistency

Availability

Partition-Tolerance

Sometimes allow stale data...

...but everything can keep going.

# CAP tradeoffs

Consistency

Availability

Partition-Tolerance

This is where BASE leads us.

This is a real tradeoff -- if you make it, understand it!

**B**asically  
**A**vailable  
**S**oft State  
**E**ventually-Consistent

# Eventually-Consistent

It just forces you to remember that  
**everything is probabilistic.**

It also doesn't mean slow.  
BASE and DIRT are not in conflict!

# Eventually-Consistent doesn't mean “not consistent”!

It just forces you to remember that  
**everything is probabilistic.**

It also doesn't mean slow.  
BASE and DIRT are not in conflict!

# Eventually-Consistent doesn't mean “not consistent”!

It just forces you to remember that  
**everything is probabilistic.**

It also doesn't mean slow.  
BASE and DIRT are not in conflict!

Be operation-centric, involve all layers of app in decisions.

**A**ssociative  
**C**ommutative  
**I**dempotent  
**D**istributed

Be operation-centric, involve all layers of app in decisions.

# Brief Review

- You can't hide from time.
- Be asynchronous to be available.
- There is no global state.
- ACID vs BASE and the CAP tradeoff
- Shipping operations instead of state allows for more robust systems.

# Brief Review

- You can't hide from time.
- Be asynchronous to be available.
- There is no global state.
- ACID vs BASE and the CAP tradeoff
- Shipping operations instead of state allows for more robust systems.

Nice ideas maybe, but how do I use them?

# Throw away RPC.

Treating remote communication like local function calls is a fundamentally bad abstraction.

- Network can fail after call “succeeds”.
- Data copying cost can be hard to predict.
- Tricks you by working locally.  
(and then failing in a real dist sys)
- Prevents awareness of swimlanes.  
(and thus causes cascading failure)

# Protocols vs. APIs

- Explicit understanding of boundaries.  
(trust boundaries, failure boundaries...)
- Better re-use and composition.  
(unintuitive but true in the large)
- Asynchronous reality, described accurately.  
(see Clojure or Erlang/OTP libraries)

# Successful Protocols

**Kings of the Internet:      DNS & HTTP**

What do they have in common?

# Successful Protocols

Kings of the Internet: DNS & HTTP

What do they have in common?

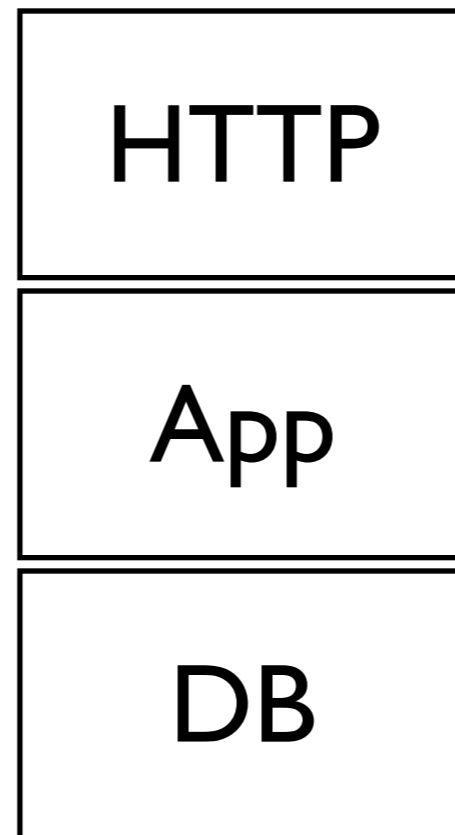
B  
A  
S  
E

# The Web

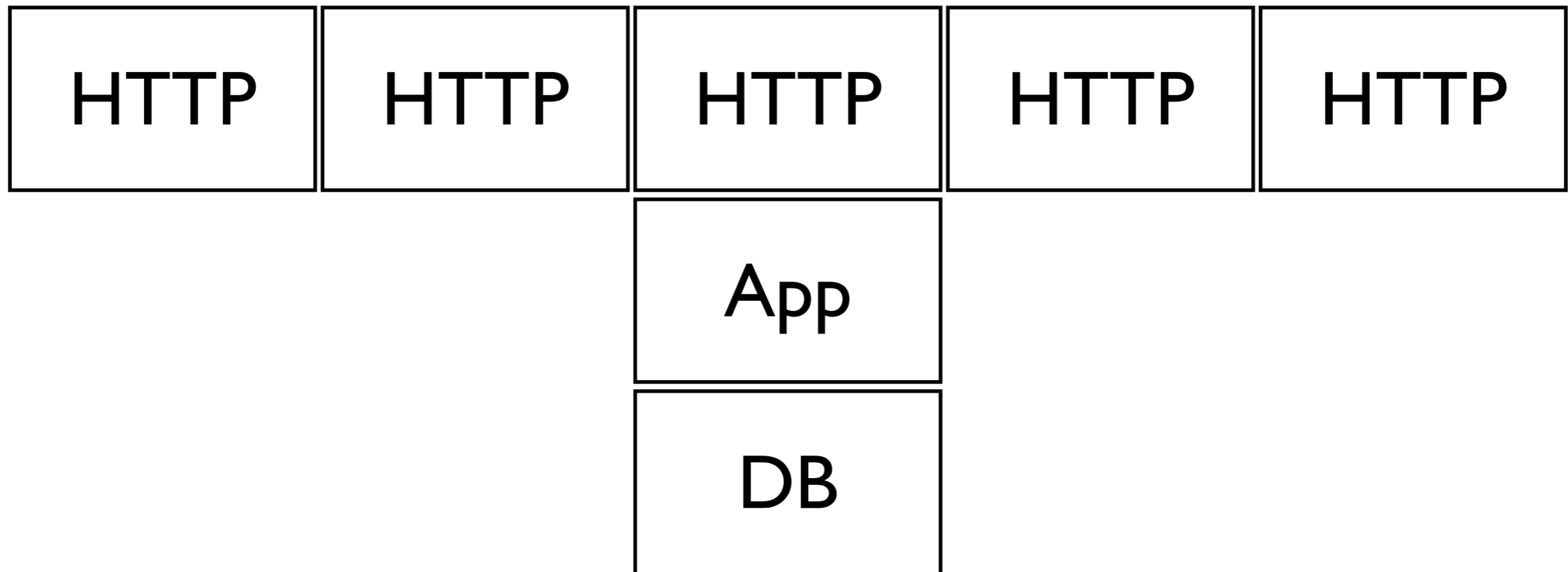
(it's the most successful distributed system ever)

- no global state (closest is DNS and MIME)
- well-defined caching for eventual consistency
  - idempotent operations!
- loose coupling
  - links instead of global relations
  - no must-understands except HTTP

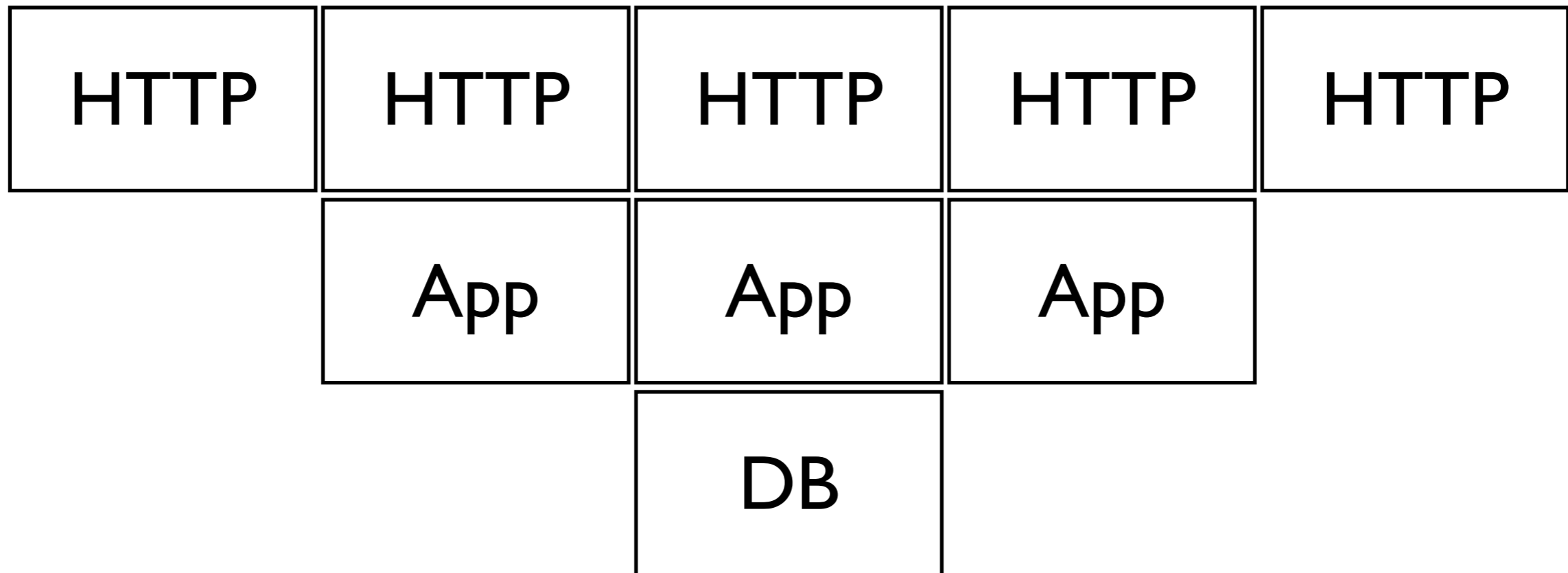
# History of Scaling The Web



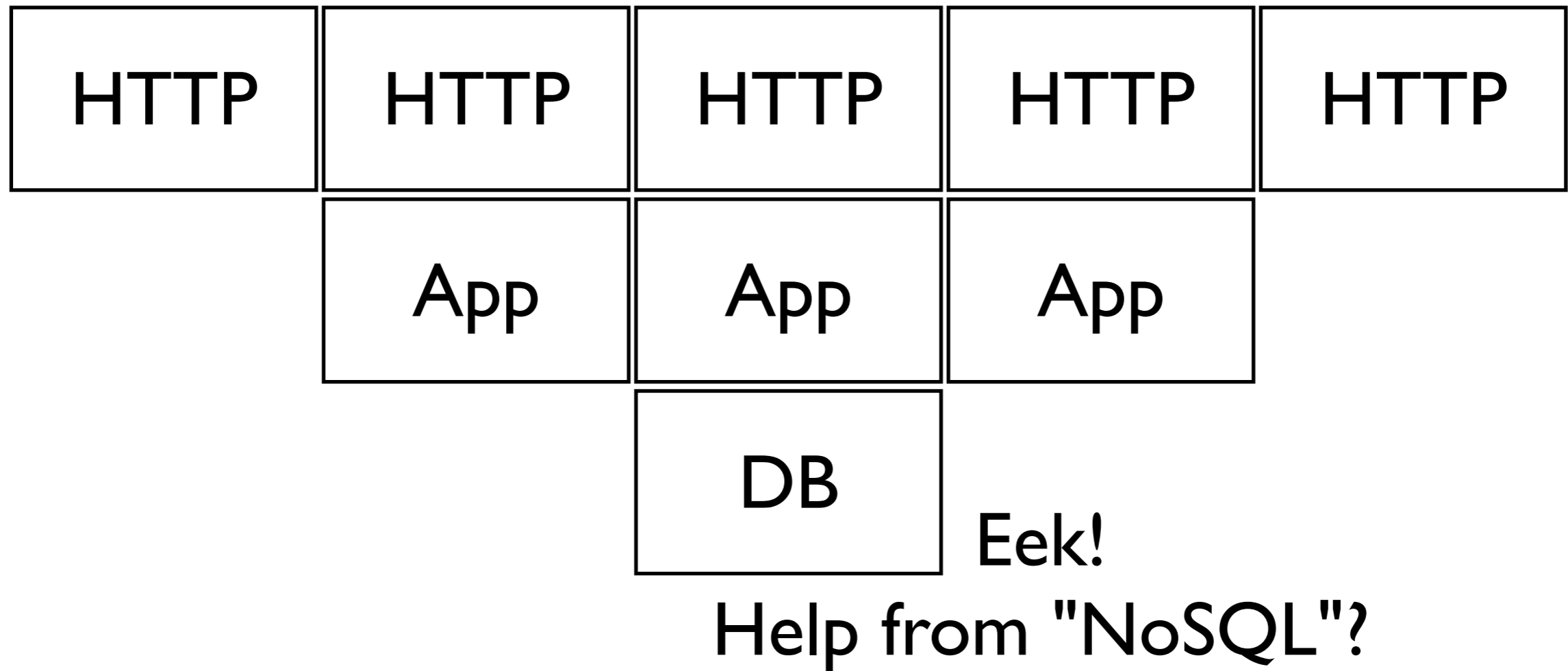
# History of Scaling The Web



# History of Scaling The Web



# History of Scaling The Web



# Scalable

"I can add twice as much  $X$  to get twice as much  $Y$ ."

# Scalable

computers



"I can add twice as much X to get twice as much Y."

# Scalable

computers

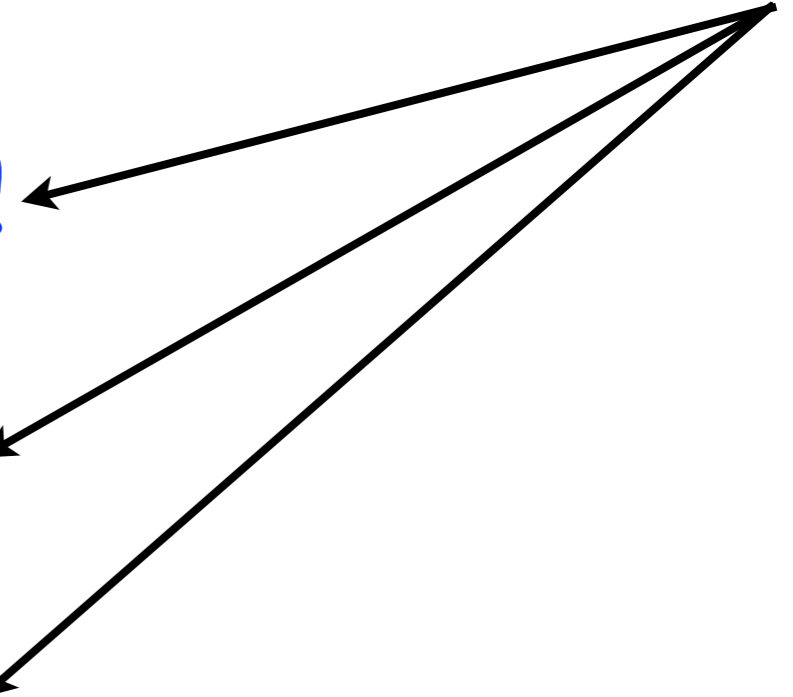


"I can add twice as much  $X$  to get twice as much  $Y$ ."

write-throughput!

storage capacity!

map/red power!



# Linearly Scalable

computers

"I can add twice as much  $X$  to get twice as much  $Y$ ."

write-throughput!

storage capacity!

map/red power!

# Reliable

Doesn't fail?

**"Reliable" is tricky.**

**Everything fails.**

**What do we really mean?**

# Resilient

Assume that **failures will happen.**

Designing whole systems and components with individual failures in mind is a plan for **predictable success.**

# Know How You Degrade

Plan it and understand it before your users do.

You might prevent whole system failure if you're lucky and good, but what happens during **partial failure?**

# Harvest and Yield

**harvest:** fraction of data available to a given request

**yield:** probability of a given request completing successfully

in tension with each other:  
 $(\text{harvest} * \text{yield}) \sim \text{constant}$

# Harvest and Yield

traditional ACID demands 100% **harvest**  
but success of modern applications is  
often measured in **yield**

plan ahead, know when you care!

# Measurement

Today's networked world is full of cascading implicit and explicit SLAs

Reason about your behavior,  
but also measure it in production.

# Measurement

In dist. sys. if you aren't really measuring,  
then you'll pick the wrong bottlenecks.

Measure your systems top to bottom, and  
correlate information cross-system.

# Embracing Concurrency at scale

(it's about time!)



Justin Sheehy  
[justin@basho.com](mailto:justin@basho.com)