Reconciling Eventually-Consistent Data with CRDTs

Starring **Noel Welsh**

A myna Production

In Conjunction With

_.underscore

Showing at

Velocity London 2013


WHY ARE
WE HERE?

CRDTs

[AN OVERVIEW]

MERGE DATA AUTOMATICALLY

Handle your eventually-consistent data store

Simplify distributed systems

Easy data synchronisation

#1

What problem we are solving

#2

How CRDTs work

#3

Issues in practice


WHY ARE
WE HERE?

WE FEEL

THE NEED

FOR...

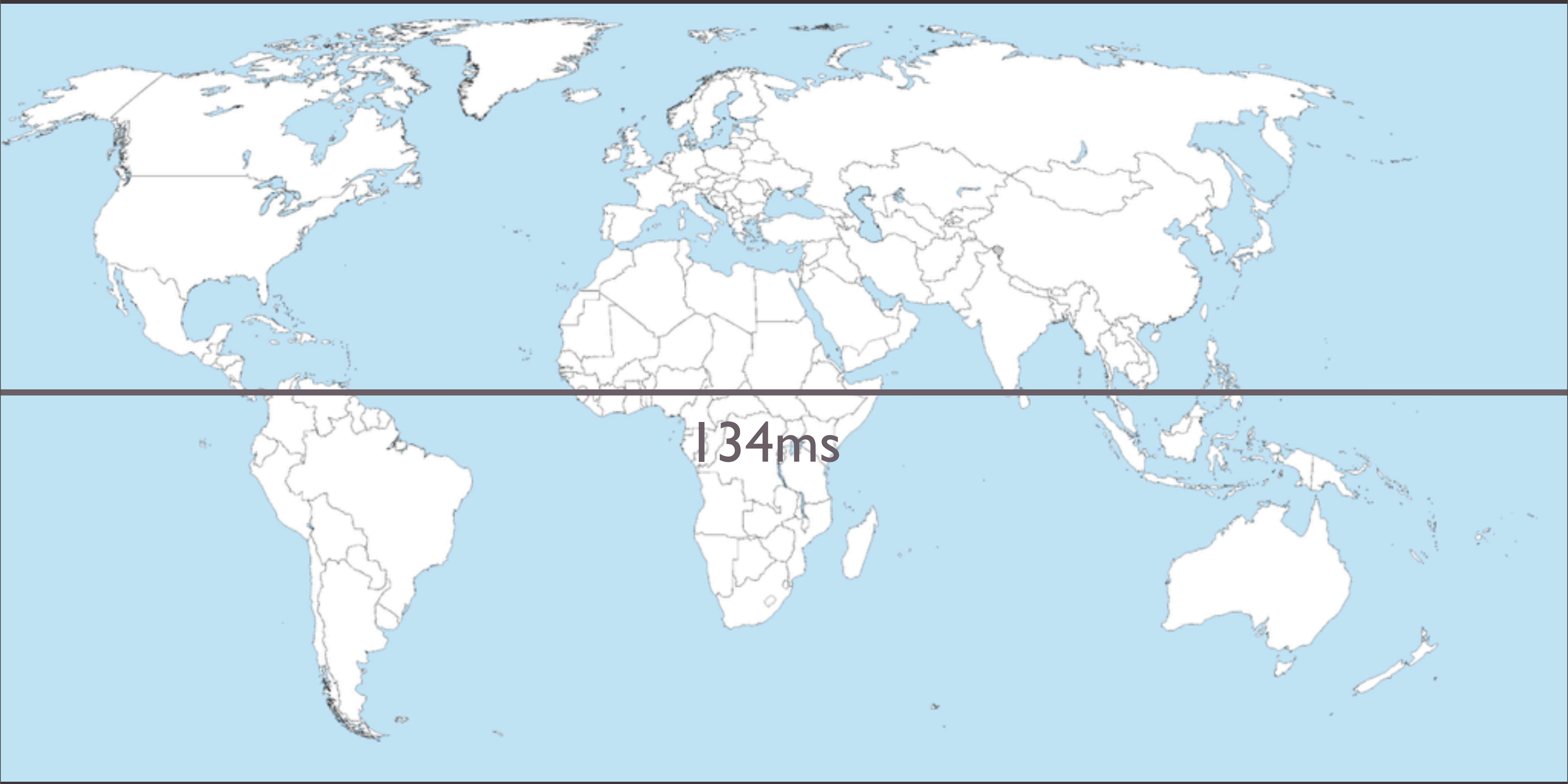
sub-Second

PAGE LOADS

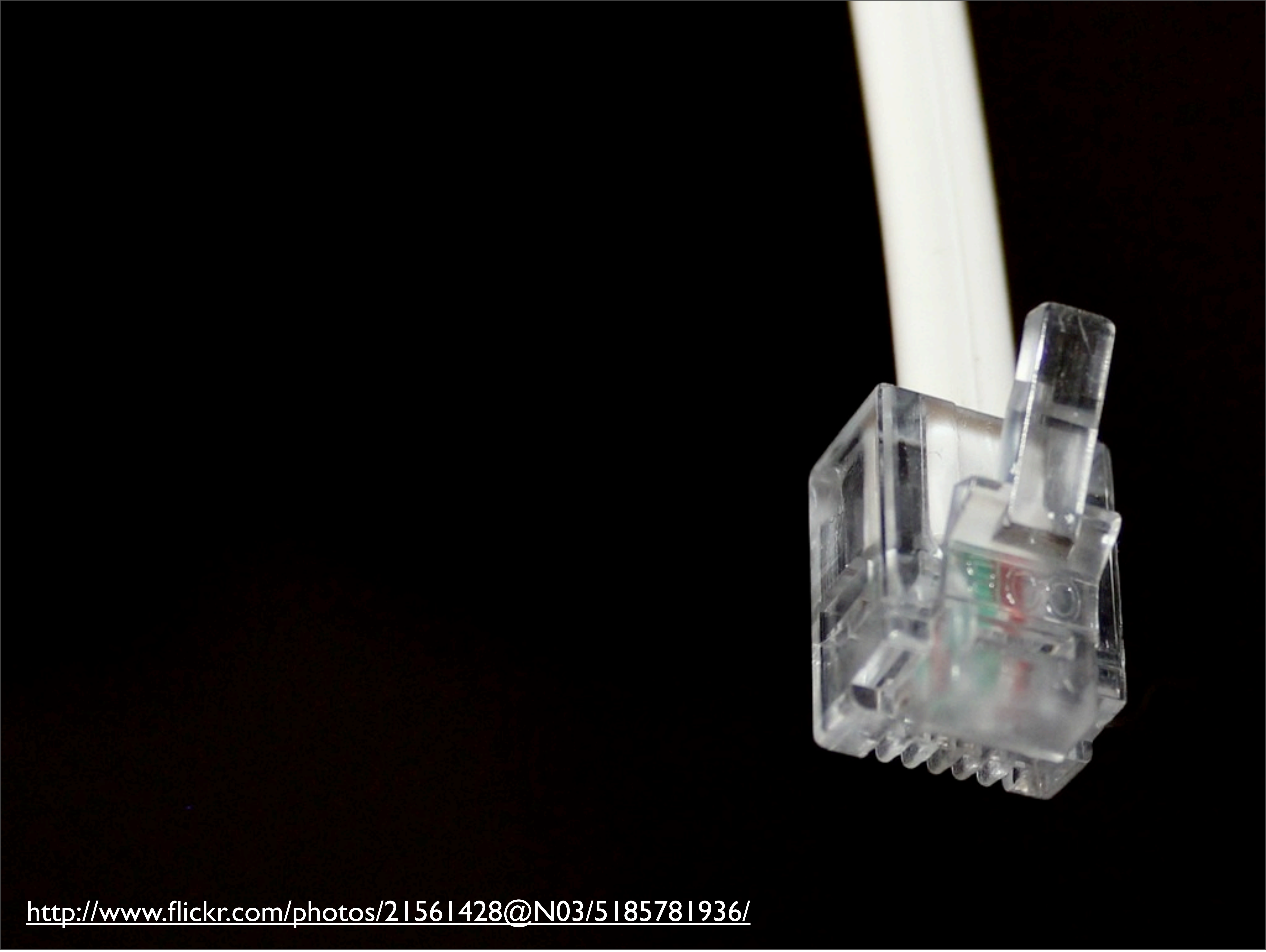
WE HAVE SEEN THE

ENEMY

$$E=MC^2$$



World map from Wikimedia Commons

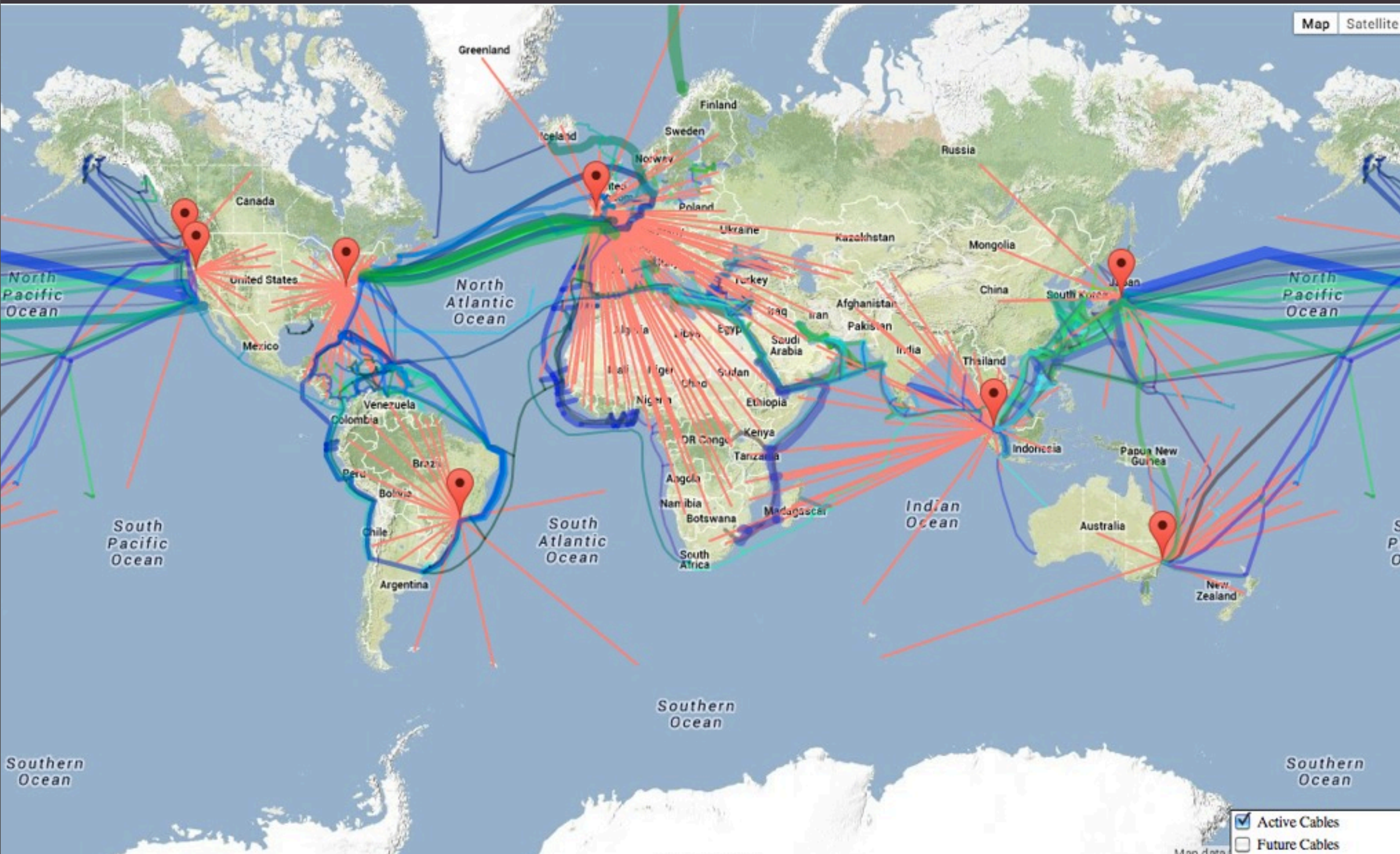


http://www.flickr.com/photos/21561428@N03/5185781936/

LOCATION

LOCATION

LOCATION



http://turnkeylinux.github.io/aws-datacenters/

PROBLEM

SOLVED!

PROBLEM

SOLVED?

HOW DO WE SORT OUT THIS MESS!

WE HAVE

AVAILABILITY

CAN WE REGAIN

CONSISTENCY?

Conflict-free Replicated Data Type

Conflict-free Replicated Data Type

Conflict-free Replicated Data Type

Conflict-free Replicated Data Type

Conflict-free
Convergent
Commutative

G-Counter

A COUNTER THAT CAN ONLY GROW

NAIVE APPROACH

Machine A

0

Machine B

0

SOME TIME LATER...

NAIVE APPROACH

Machine A

5

Machine B

7

WHAT VALUE SHOULD THE COUNTER TAKE?

G-Counter insight: Store a separate counter for each machine

G-COUNTER APPROACH

Machine A	Machine B
A: 0	A: 0
B: 0	B: 0

A MACHINE CAN ONLY INCREMENT ITS OWN COUNTER

G-COUNTER APPROACH

Machine A	Machine B
A: 4	A: 0
B: 0	B: 6

MERGE IS SIMPLY THE

MAX

G-COUNTER APPROACH

Machine A	Machine B
A: 4	A: 4
B: 6	B: 6

THE COUNTER'S VALUE IS SIMPLY THE TOTAL

G-COUNTER APPROACH

Machine A

A: 4
B: 6

Machine B

A: 4
B: 6

Total is 10

We have a distributed eventually-consistent increment-only counter.

PN-Counter

A COUNTER THAT CAN GROW AND SHRINK

Can’t use a G-Counter as we can’t use max to merge

USE

TWO G-COUNTERS!

PN-COUNTER

Machine A

Additions

A: 4, B: 2

Subtractions

A: 5, B: 3

Machine B

Additions

A: 4, B: 7

Subtractions

A: 3, B: 4

MERGE IS SIMPLY THE

MAX

PN-COUNTER

Machine A

Additions

A: 4, B: 7

Subtractions

A: 5, B: 4

Machine B

Additions

A: 4, B: 7

Subtractions

A: 5, B: 4

THE COUNTER'S VALUE IS SIMPLY THE TOTAL

PN-COUNTER

Machine A

Additions

A: 4, B: 7

Subtractions

A: 5, B: 4

Machine B

Additions

A: 4, B: 7

Subtractions

A: 5, B: 4

Total is

(4+7) - (5+4)=

2

A GENERAL RECIPE FOR MERGES

MERGE MUST BE

INVARIANT TO ORDER

MERGE MUST

CONVERGE TO CORRECT VALUE

FORMALLY:

IDEMPOTENT

$$x \bullet x = x$$

FORMALLY:

ASSOCIATIVE

$$(x \bullet y) \bullet z = x \bullet (y \bullet z)$$

FORMALLY:

COMMUTATIVE

$$x \bullet y = y \bullet x$$

AN IDEMPOTENT COMMUTATIVE MONOID

:-]

NUMBERS and MAX

SETS and UNION

PN-Counter also requires addition and subtraction

Set Union & Difference

PN-Set

2P-Set

Machine A

Additions

A: {x}, B: {y}

Subtractions

A: {x}, B: {}

Machine B

Additions

A: {x}, B: {y, z}

Subtractions

A: {}, B: {y}

2P-Set Merge

Machine A

Additions

A: {x}, B: {y, z}

Subtractions

A: {x}, B: {y}

Machine B

Additions

A: {x}, B: {y, z}

Subtractions

A: {x}, B: {y}

2P-Set Total

Machine A

Additions

A: {x}, B: {y, z}

Subtractions

A: {x}, B: {y}

Machine B

Additions

A: {x}, B: {y, z}

Subtractions

A: {x}, B: {y}

Set is

{z}

Deleted elements stored indefinitely. Called

tombstones

2P-Set allows elements to be added and removed **once**

C-Set

Store element and count

C-Set

Machine A

Additions
A: {(x, 2)},
B: {(y, 1)}

Subtractions
A: {(x, 1)},
B: {}

Machine B

Additions
A: {(x, 1)},
B: {(y, 1), (z, 2)}

Subtractions
A: {},
B: {(y, 1)}

C-Set Merge

Machine A

Additions

A: {(x, 2)},

B: {(y, 1), (z, 2)}

Subtractions

A: {(x, 1)},

B: {(y, 1)}

Machine B

Additions

A: {(x, 2)},

B: {(y, 1), (z, 2)}

Subtractions

A: {(x, 1)},

B: {(y, 1)}

C-Set Total

Machine A

Additions
A: {(x, 2)},
B: {(y, 1), (z, 2)}

Subtractions
A: {(x, 1)},
B: {(y, 1)}

Machine B

Additions
A: {(x, 2)},
B: {(y, 1), (z, 2)}

Subtractions
A: {(x, 1)},
B: {(y, 1)}

Set is

{x, z}

C-Set allows elements to be added and removed **many times**

C-Set allows elements to be removed **more times** than they have been added

OR-Set

Store element and unique token

OR-Set

Machine A

Additions

A: {(x, #a), (x, #d)},

B: {(y, #b)}

Subtractions

A: {(x, #a)},

B: {}

Machine B

Additions

A: {(x, #a)},

B: {(y, #b), (z, #c)}

Subtractions

A: {},

B: {(y, #b)}

OR-Set Merge

Machine A

Additions

A: {(x, #a), (x, #d)},

B: {(y, #b), (z, #c)}

Subtractions

A: {(x, #a)},

B: {(y, #b)}

Machine B

Additions

A: {(x, #a), (x, #d)},

B: {(y, #b), (z, #c)}

Subtractions

A: {(x, #a)},

B: {(y, #b)}

OR-Set Total

Machine A

Additions

A: {(x, #a), (x, #d)},

B: {(y, #b), (z, #c)}

Subtractions

A: {(x, #a)},

B: {(y, #b)}

Machine B

Additions

A: {(x, #a), (x, #d)},

B: {(y, #b), (z, #c)}

Subtractions

A: {(x, #a)},

B: {(y, #b)}

Set is

{x, z}

OR-Set **works** the way we expect

From sets, build **trees, graphs, etc.**

CRDTS vs

THE REAL WORLD

Strong Consistency
Memory Usage
Code

STRONG CONSISTENCY

Don’t build your billing platform on CRDTs

MEMORY USAGE

Tombstones

Machine IDs

Tombstones: Establish **causal order** and delete [Bieniusa et al. 2012]

Tombstones: Prune with **heuristics** [often based on time]

Machine IDs: OR-Sets

don't need them

Machine IDs: Hierarchical organisation allows pruning [Almeida & Baquero. 2013]

CODE

Riak 2.0

Various open source libraries


THANK YOU!
NOW GO FORTH AND
DISTRIBUTE!
http://stjost.deviantart.com/art/Stomping-Off-Into-the-Sunset-277086274

MORE:

noelwelsh.com