

# DEVELOPING IN PRODUCTION

develop in production; push to dev

O'REILLY\*

Alibaba Group

# Velocity

China 2012

Web性能与运维大会

Building a Faster and Stronger Web

Beijing, China

Dec 4-5, 2012

# Theo Schlossnagle


Engineer (postwait on github, jesus of the ASF)

Founder at OmniTI, Circonus & Message Systems

Author of Scalable Internet Architectures (Sams)

Member of the ACM and IEEE

# HOW THE WORLD USED TO LOOK

Architectures driven by user interaction:  
(e.g. web page load  web page response)

# HOW THE WORLD USED TO LOOK

## **isolated**

user A's page load doesn't effect user B.

## **latent**

user A's affect shows up "later" in user B's view.

# HOW THE WORLD USED TO LOOK

Problem reproducibility is achievable  
(even if sometimes painful)

# HOW THE WORLD LOOKS NOW

Architectures driven by  
high volume data *and* users

# HOW THE WORLD LOOKS NOW

every piece of data entering the architecture  
can effect a user's subsequent experience.

# HOW THE WORLD LOOKS NOW

sometimes  
the data **must** effect a user's  
subsequent experience.



# HOW THE WORLD LOOKS NOW

Social systems cause indirect impact via user action eroding isolation.

A more responsive web is eroding users' acceptance of latent state.

# HOW THE WORLD LOOKS NOW

Highly distributed systems  
(due to complexity and scalability) means  
no single world state

# HOW THE WORLD LOOKS NOW

Highly distributed systems  
(due to complexity and scalability) means  
difficult repeatability of state and failures

# HOW THE WORLD LOOKS NOW

Highly distributed systems  
(due to complexity and scalability) means  
treacherous debugability

# DESPAIR ALL YE' WHO DEBUG HERE

I studied distributed systems long enough to understand “the suck” you do not wish to have in your lives.

# DESPAIR ALL YE' WHO DEBUG HERE

Expedient debugging requires a quick realization of a precipitating state of failure.

# DESPAIR ALL YE' WHO DEBUG HERE

In distributed systems,  
post-mortem techniques are less useful  
because recreating possible global state is a  
mind-bending (or brain-damaging) exercise.



# DESPAIR ALL YE' WHO DEBUG HERE

Applying older techniques  
to small controlled distributed systems  
is still feasible,

but complex, asynchronous architectures  
can be nearly impossible.

# CASE STUDY: CIRCONUS

Circonus ingests telemetry data  
across an entire business

... as a service

... trillions of data points stored

# SOME CONTEXT ABOUT CIRCONUS

alerting • correlation • visualization • analytics

# HOW WE SEE THE WORLD TODAY

High velocity non-user-generated data inflow.

# HOW WE SEE THE WORLD TODAY

Users expect sub-second reflection of all input data and outcomes (math).

# HOW WE SEE THE WORLD TODAY

Users expect extremely high availability  
(100%, you can only disappoint)

# HOW WE SEE THE WORLD TODAY

I was surprised how  
impatient and demanding users are...

until I used the tool...  
and felt exactly the same way.



## Voltron



## Voltron

- distributed data collection (via brokers)

## Voltron

- distributed data collection (via brokers)
- distributed data ingestion (via aggregators)

## Voltron

- distributed data collection (via brokers)
- distributed data ingestion (via aggregators)
- distributed data distribution! (via message queueing)

## Voltron

- distributed data collection (via brokers)
- distributed data ingestion (via aggregators)
- distributed data distribution! (via message queueing)
- distributed data storage (via snowth)

## Voltron

- distributed data collection (via brokers)
- distributed data ingestion (via aggregators)
- distributed data distribution! (via message queueing)
- distributed data storage (via snowth)
- distributed streaming fault detection (via muppet love)

## Voltron

- distributed data collection (via brokers)
- distributed data ingestion (via aggregators)
- distributed data distribution! (via message queueing)
- distributed data storage (via snowth)
- distributed streaming fault detection (via muppet love)
- ... distributed nightmare.

# DESPAIR ACTUALIZED

I shall not memorialize this in slides.

# DESPAIR ACTUALIZED

I shall only cry in front of you now.





- Leveraging
  - stream mirroring and
  - idempotent behavior.

Allows for the recklessness that we all appreciate in a no-consequences development environment...

with  
production data,  
production volume, and  
production “personality.”

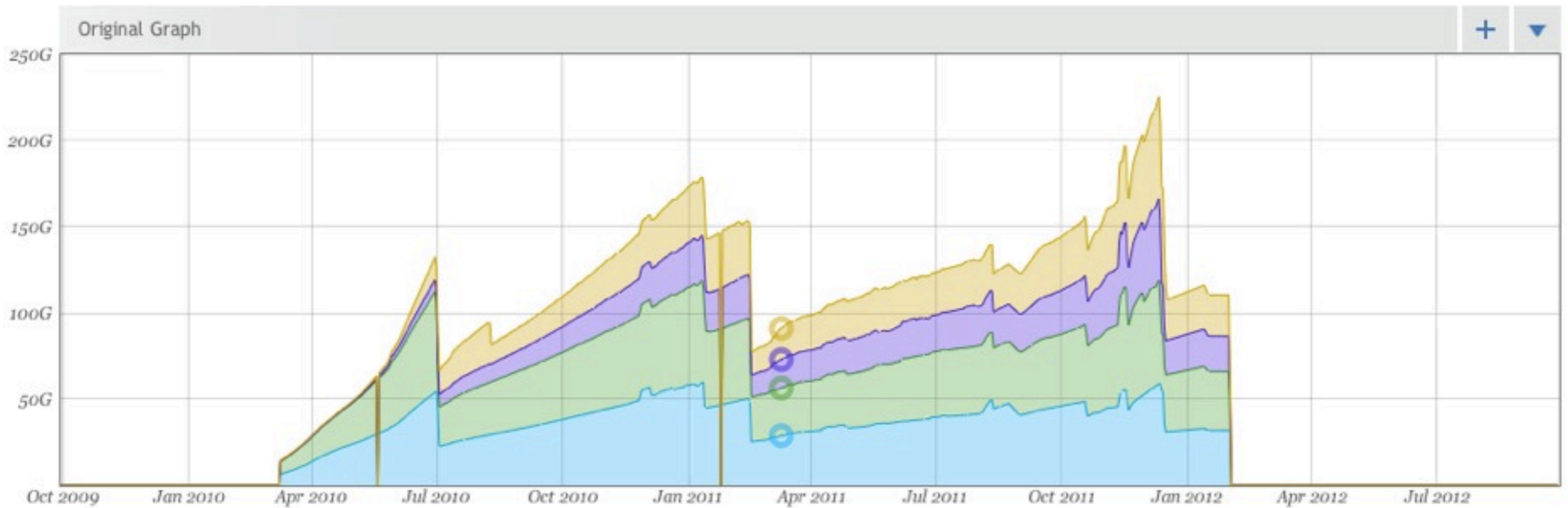
**HOPE ACTUALIZED**



**DEVELOPMENT IN PRODUCTION**

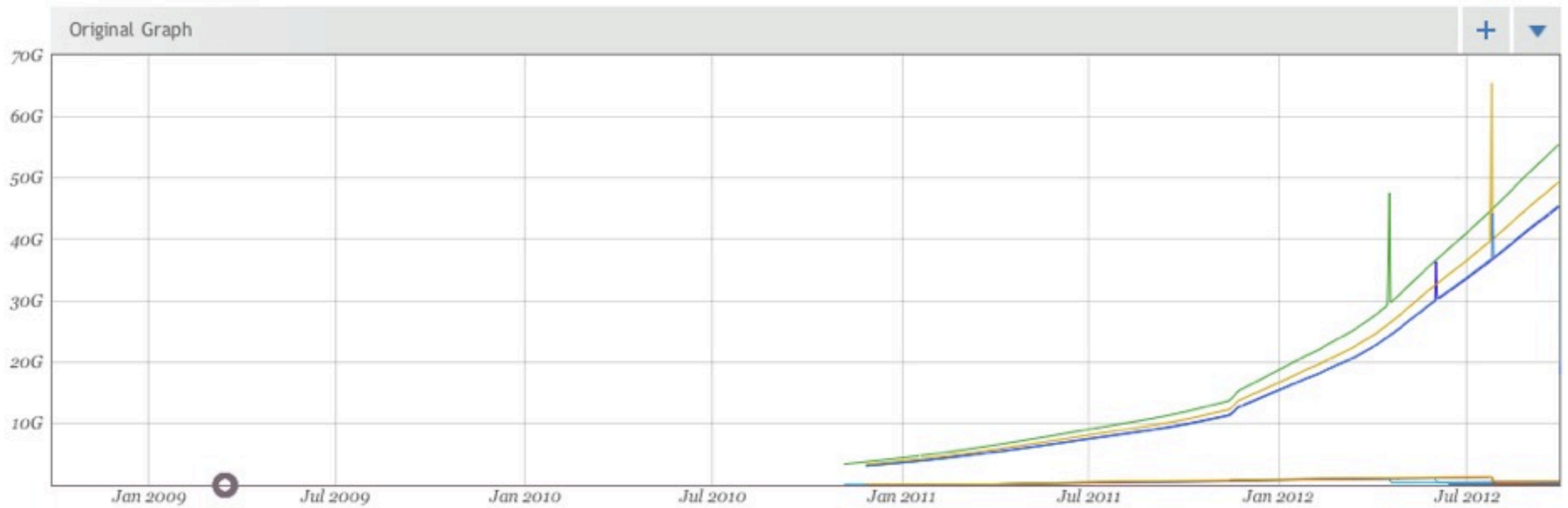
Ex #1: Dual storage backends

# EXPLORATION



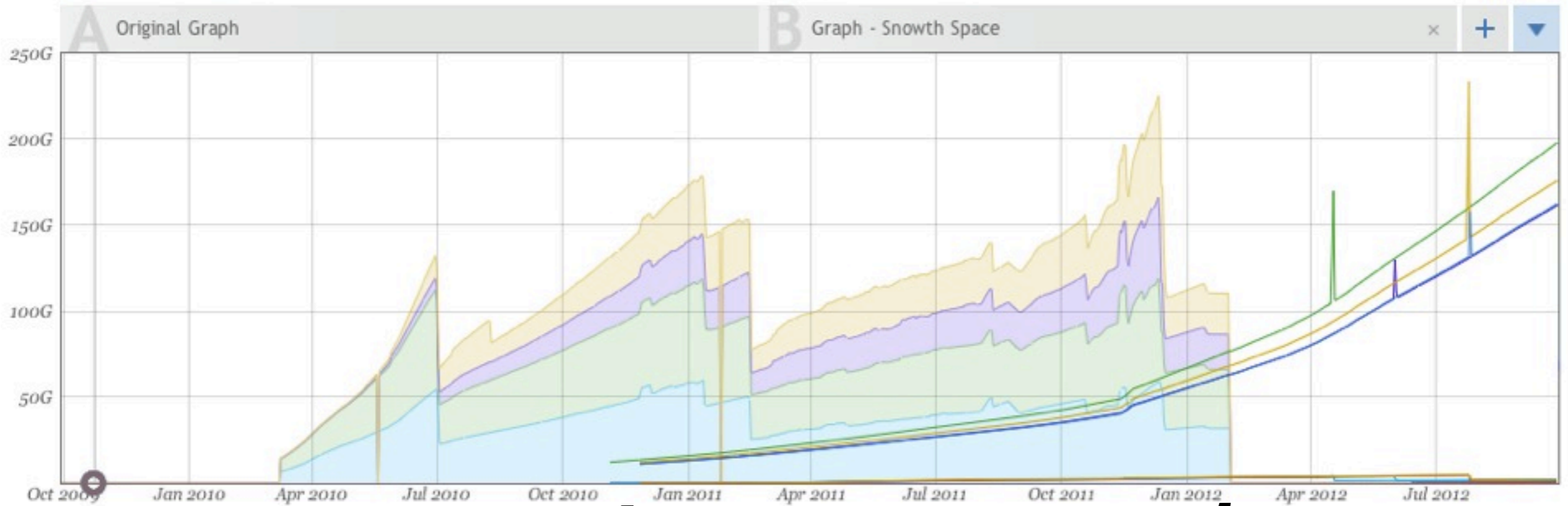
## PostgreSQL

# EXPLORATION



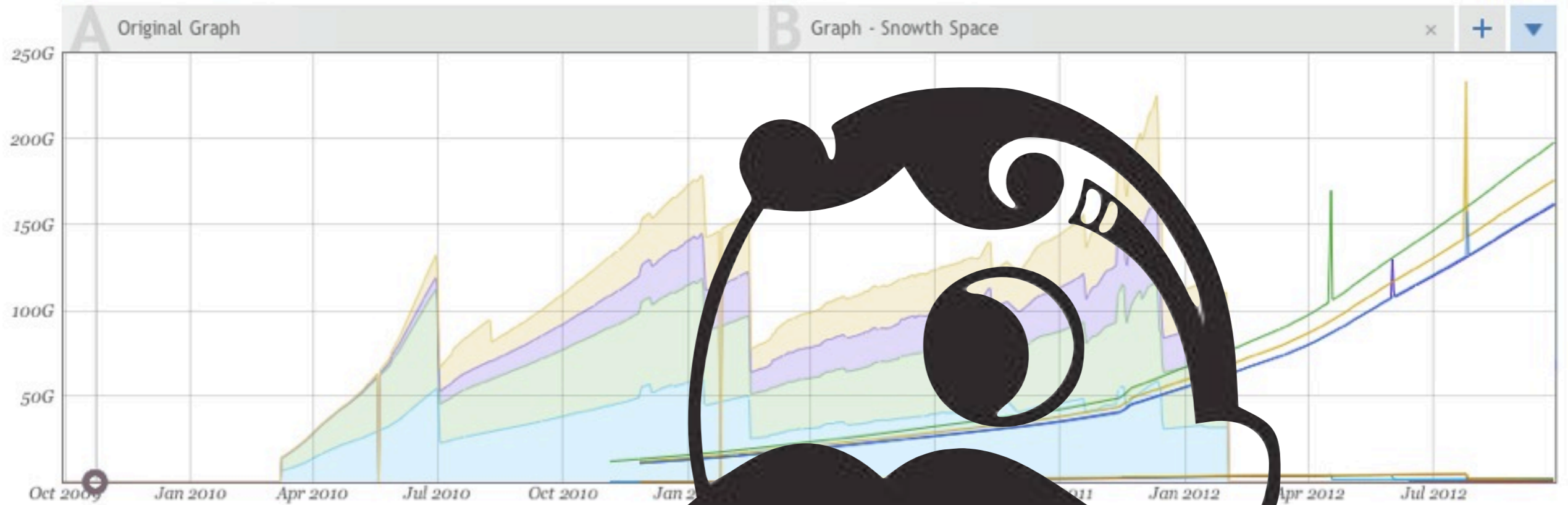
Snowth

# EXPLORATION



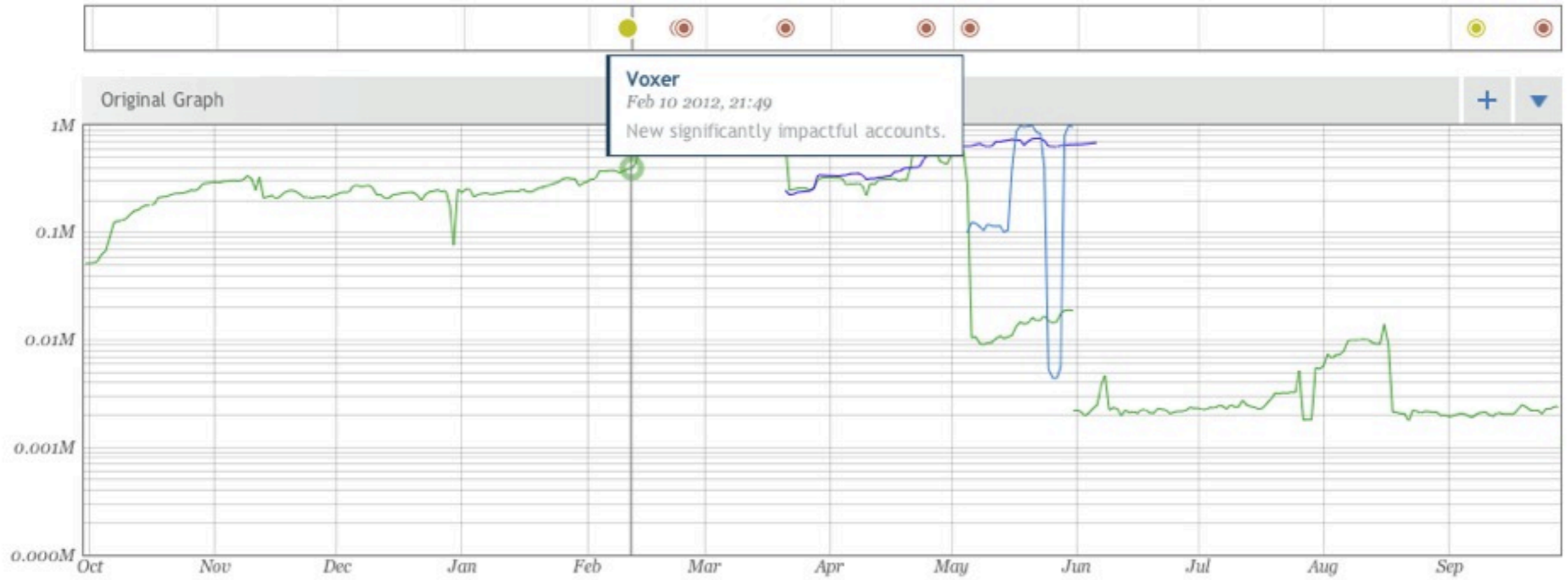
Safety & Happiness



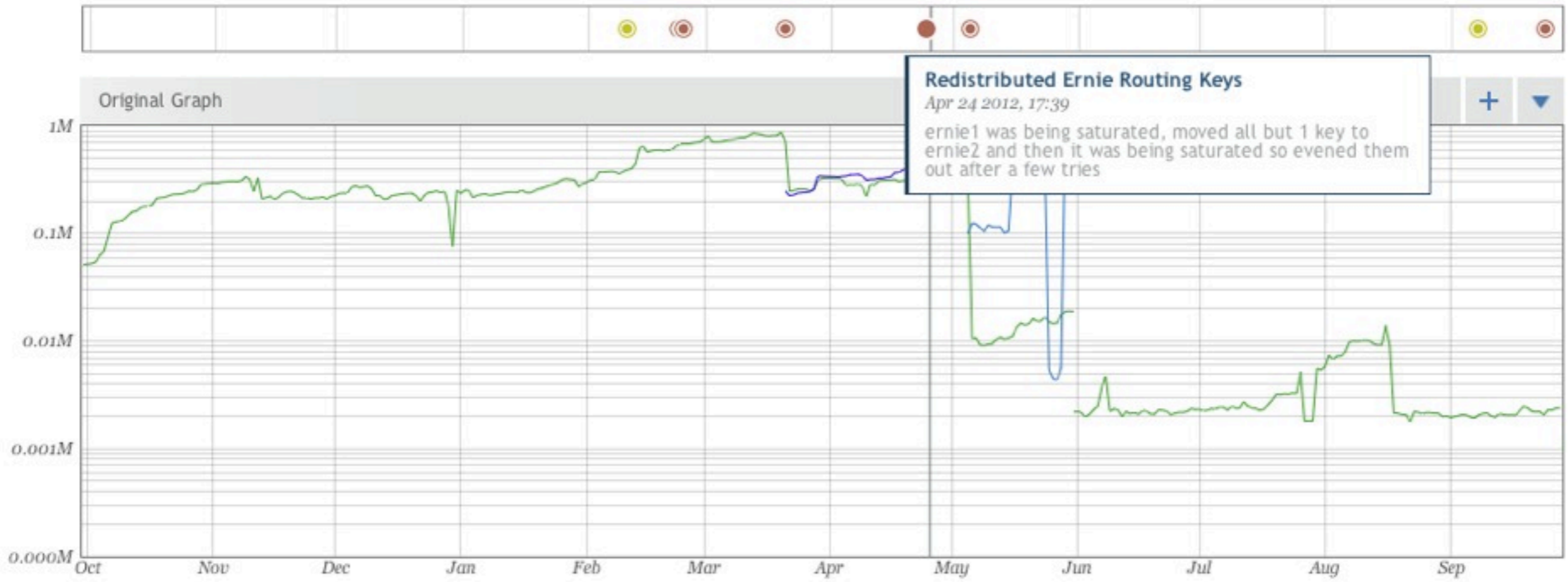


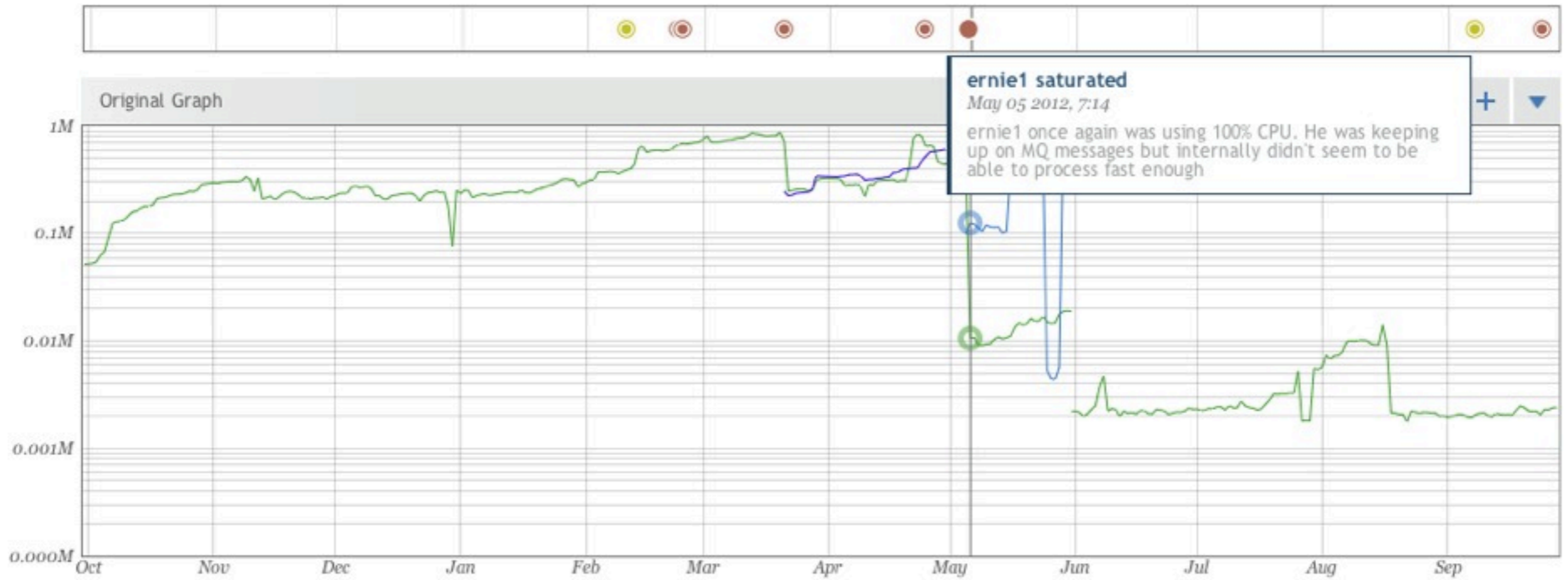
Safety & Happiness

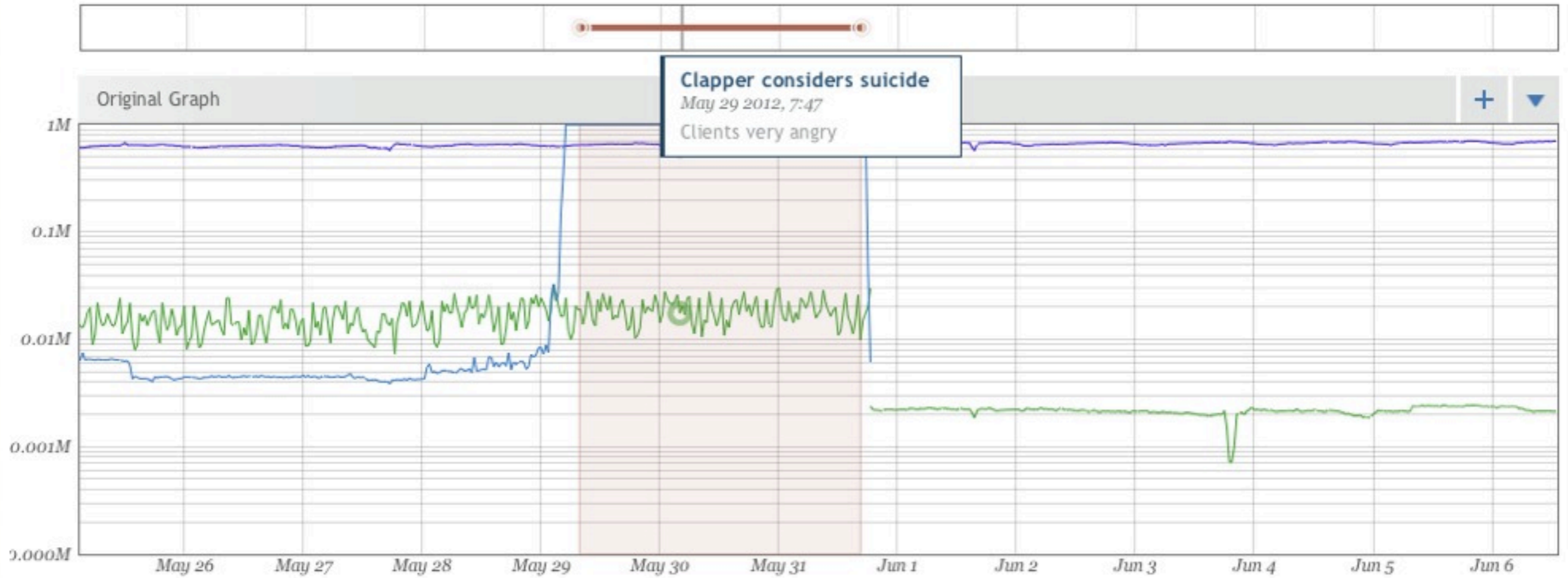
## Ex #2: Dual streaming analysis tools

















## Ex #3: Dual messages queueing systems

Every architecture driven by big data will require these techniques.

**THANK YOU**

Questions?