Scaling to 1,000,000 concurrent users on the JVM

JavaOne 2015 - CON7220

Jo Voordeckers

Sr. Software Engineer - Livefyre platform

@jovoordeckers

jvoordeckers@livefyre.com

Collect

real-time streams of UGC
to scale content creation

COMMENTS

REVIEWS

CHAT

LIVE BLOG

SIDENOTES

PHOTO UPLOAD

Organize

to quickly find and organize
the best social content

PPL WEARING JERSEYS

2015 ALL-STAR GAME

JUMP SHOTS

FAN PHOTOS

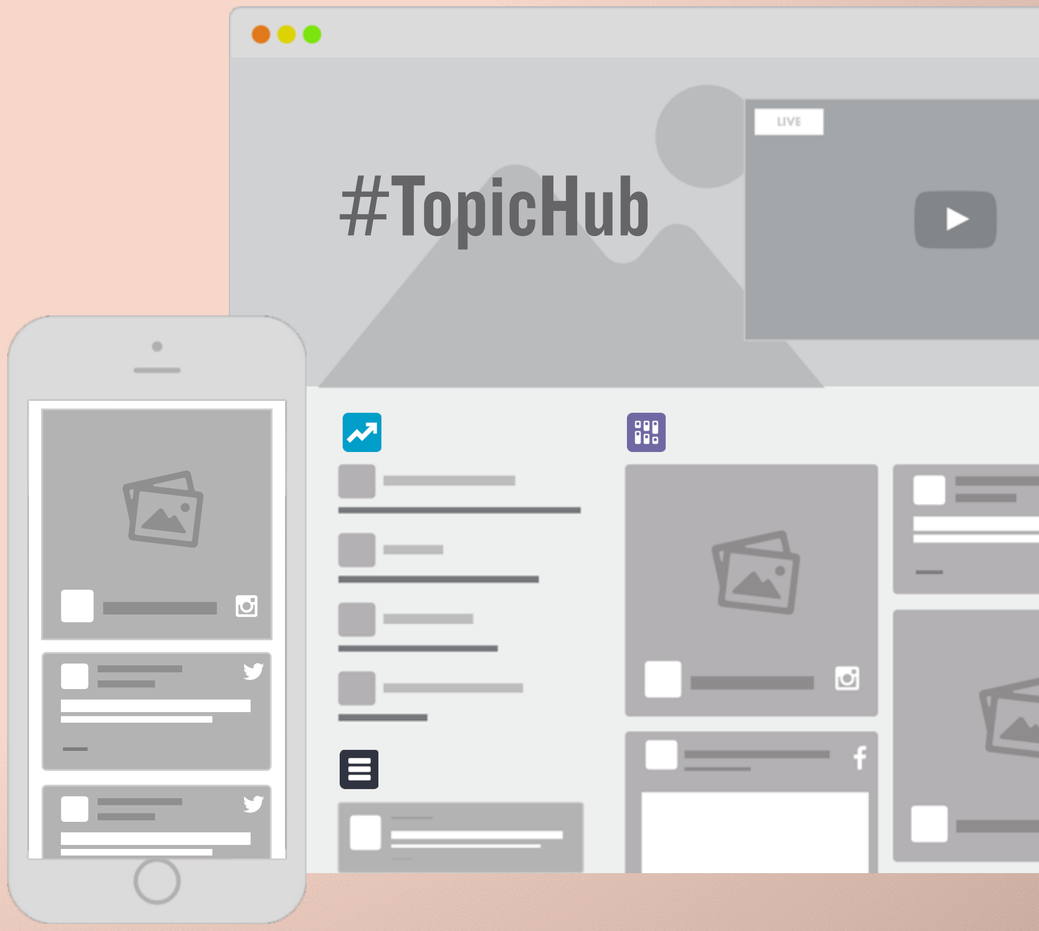
HASHTAG CAMPAIGN

Publish

to your website with
no coding required

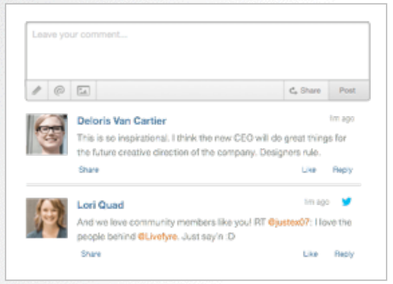
ENGAGE

audiences with best in class engagement tools
to increase time on site and build community



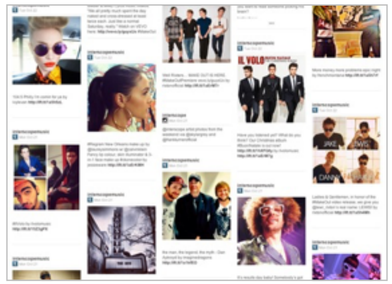
Real-Time Social Applications

Comments

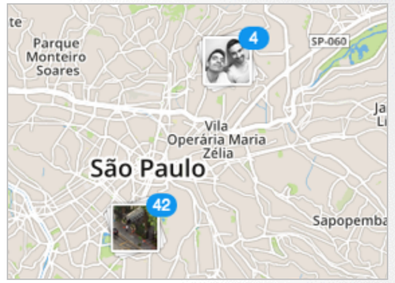


Sidenotes

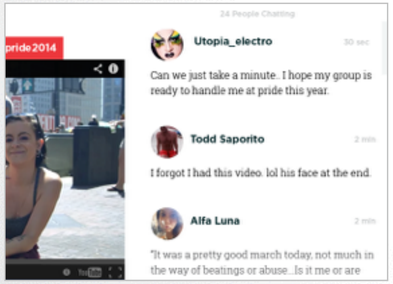
Media Wall



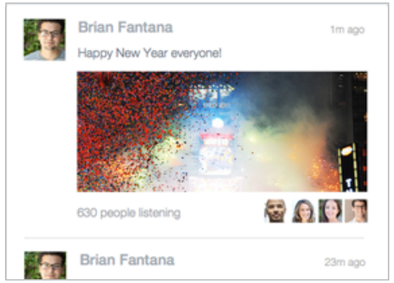
Social Maps



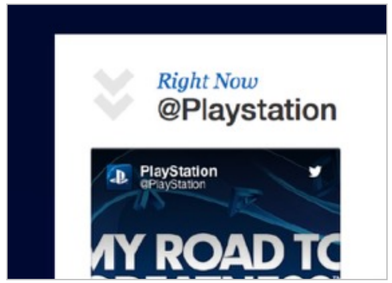
Chat



Live Blog

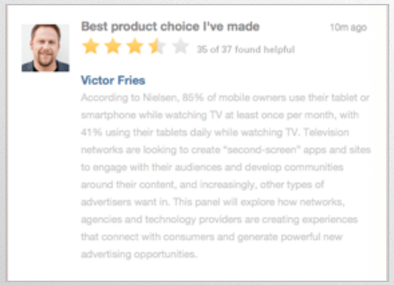


Feed

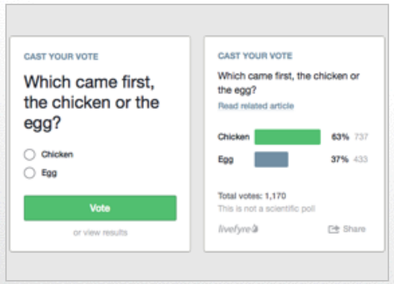


Gallery

Reviews



Polls



Trending

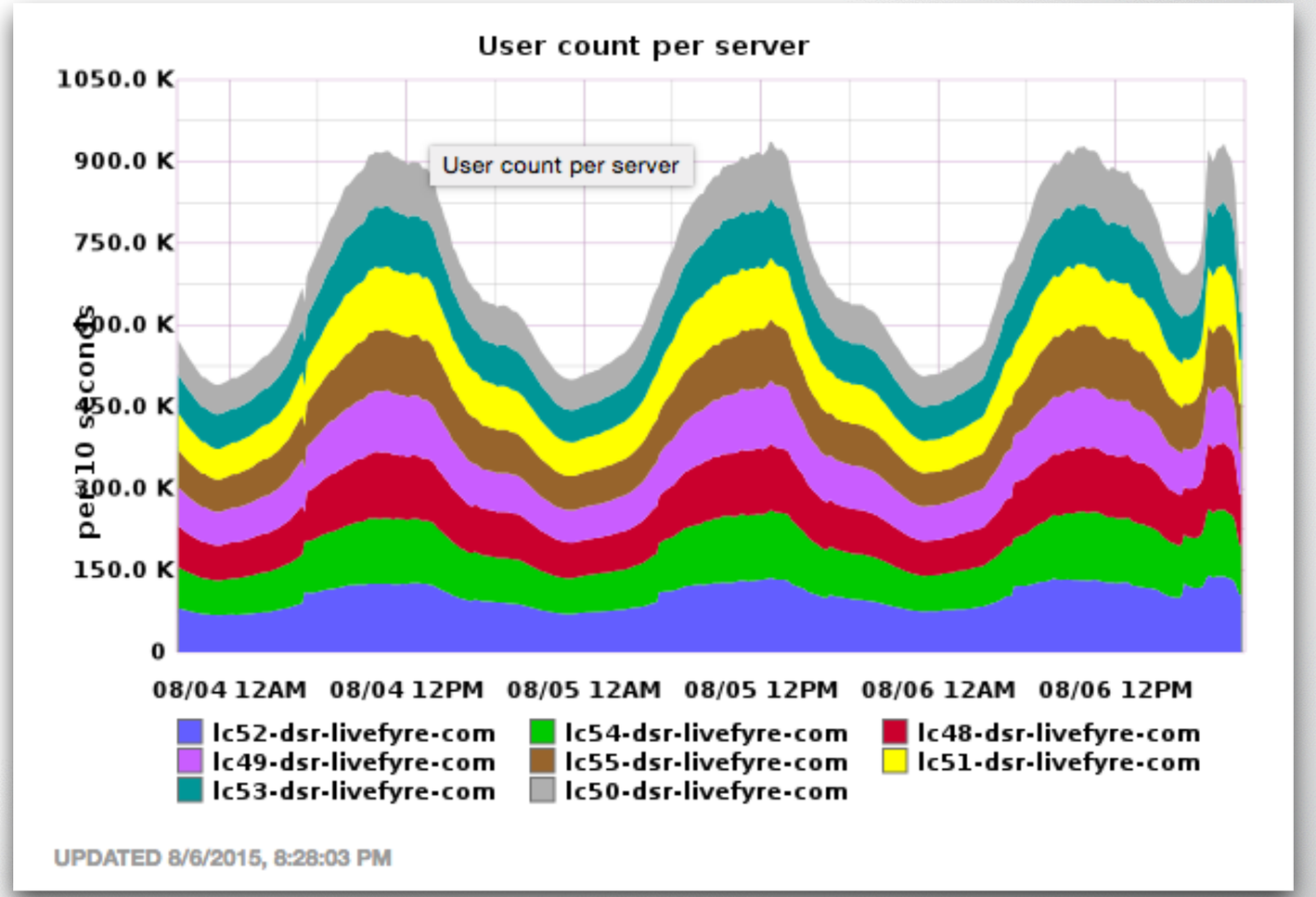
Storify

1/ CHALLENGE

Real-time challenge

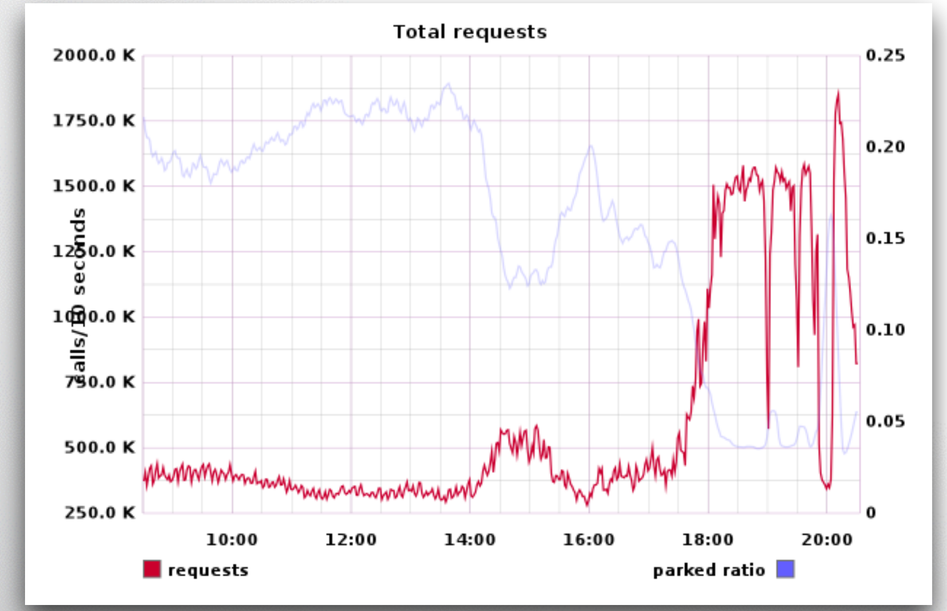
- 1,000,000 concurrent users
  - 150,000 per JVM
  - 100,000 req/s
  - 6-8x c3.2xlarge
  - long-poll + ws

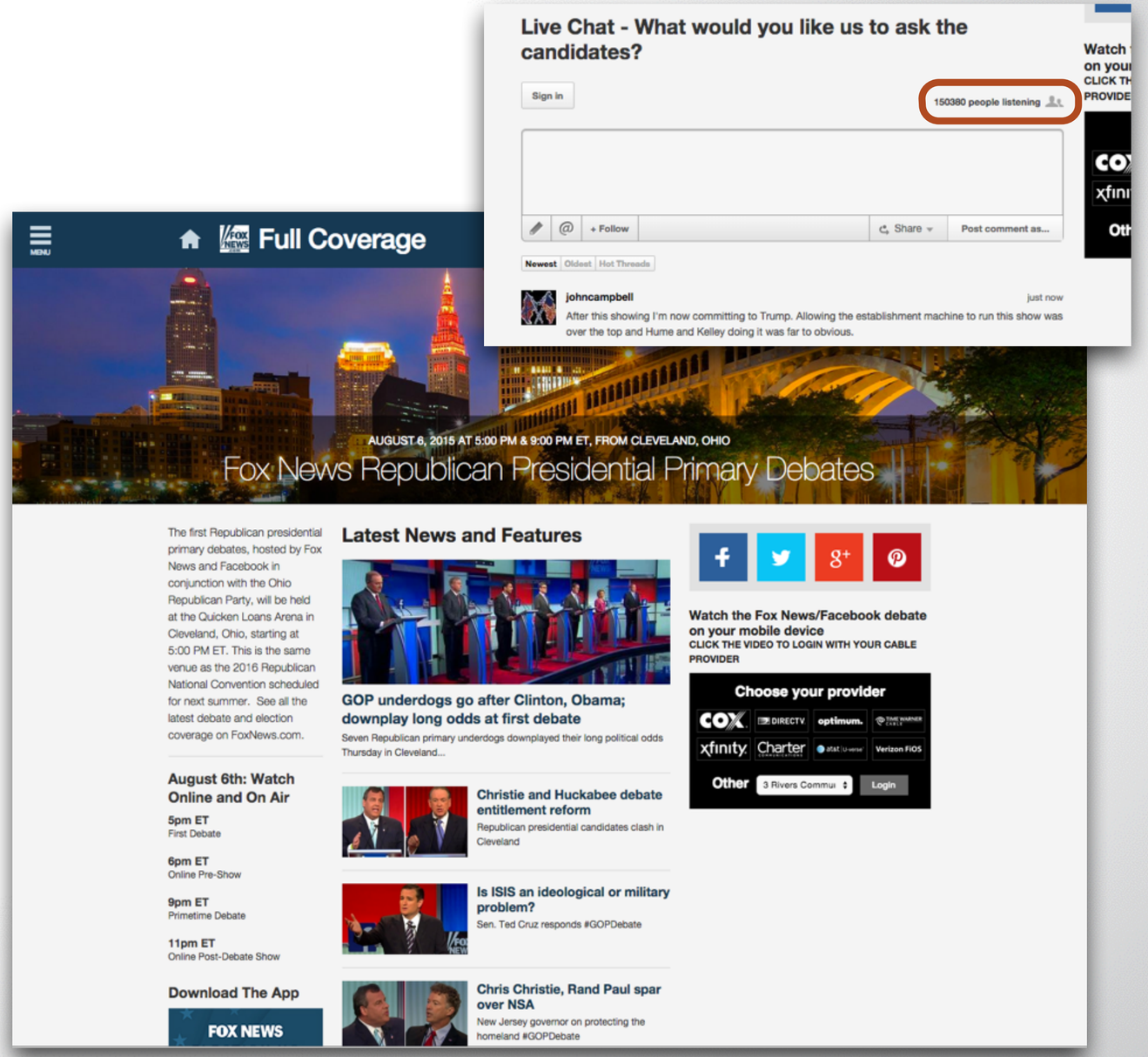
- 100s - 1,000s of listeners per stream
  - up to 250,000 listeners
  - read-heavy
  - updates < 2s



Real-time challenge

- Presidential Debate on Fox News
  - from 50,000 req/s
  - to 200,000 req/s
  - 150,000+ listeners to the stream





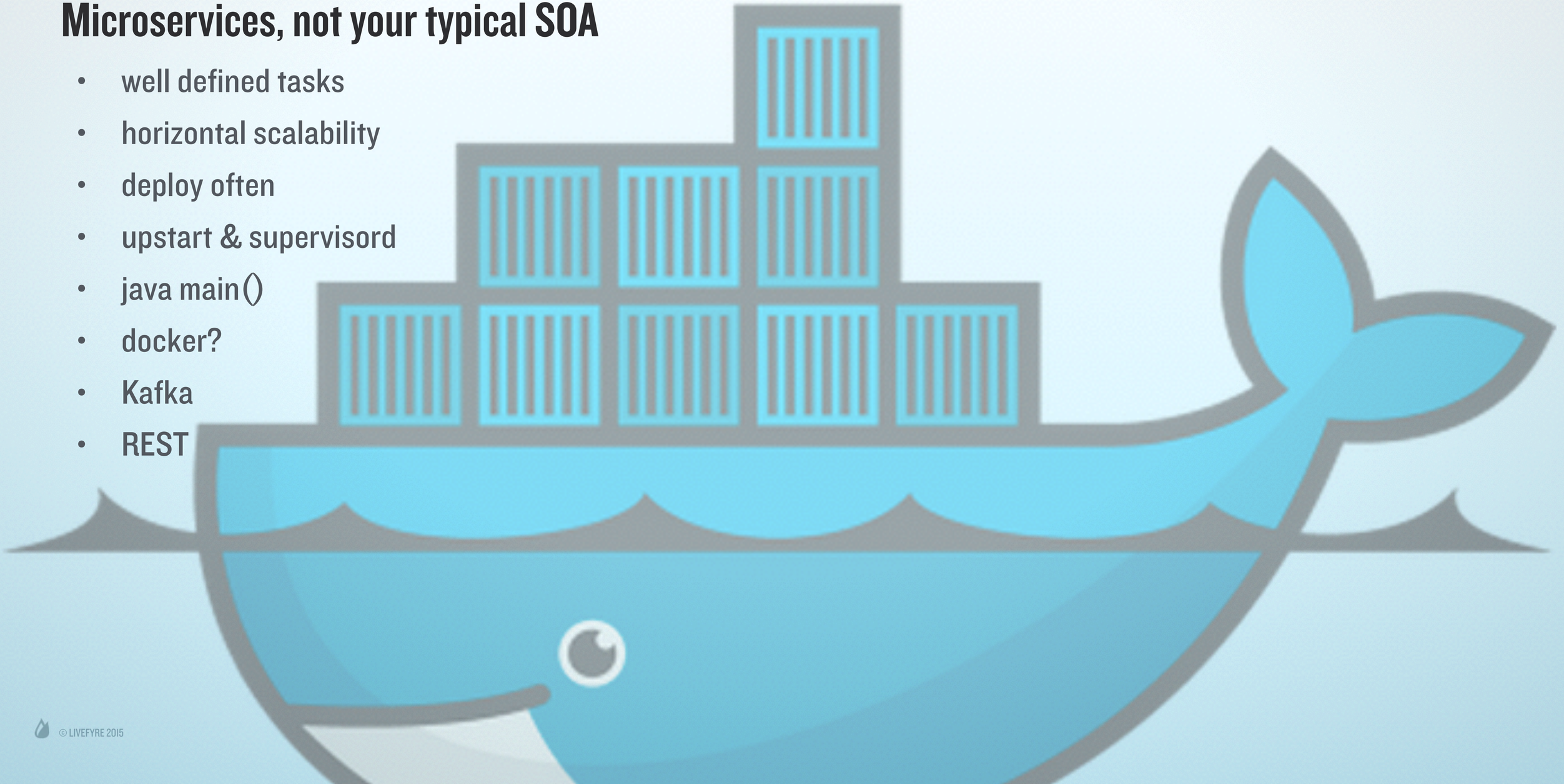
2/BE{TTER,ST} PRACTICES

Don’t use the “tech stack du jour”

- use the right tools for your problem
- embrace polyglot
  - Java, Scala, Jython
  - Python
  - NodeJS
- **KISS + YAGNI**

Microservices, not your typical SOA

- well defined tasks
- horizontal scalability
- deploy often
- upstart & supervisord
- java main()
- docker?
- Kafka
- REST



Netflix & Micro-Services

@bruce_m_wong

Monitor all the things!

are we sad

- error vs success rates and timing
- queue depth or lag
- system resources
- sample high velocity
- /ping and /deep-ping

access patterns

- optimize scaling strategy
- anticipate events

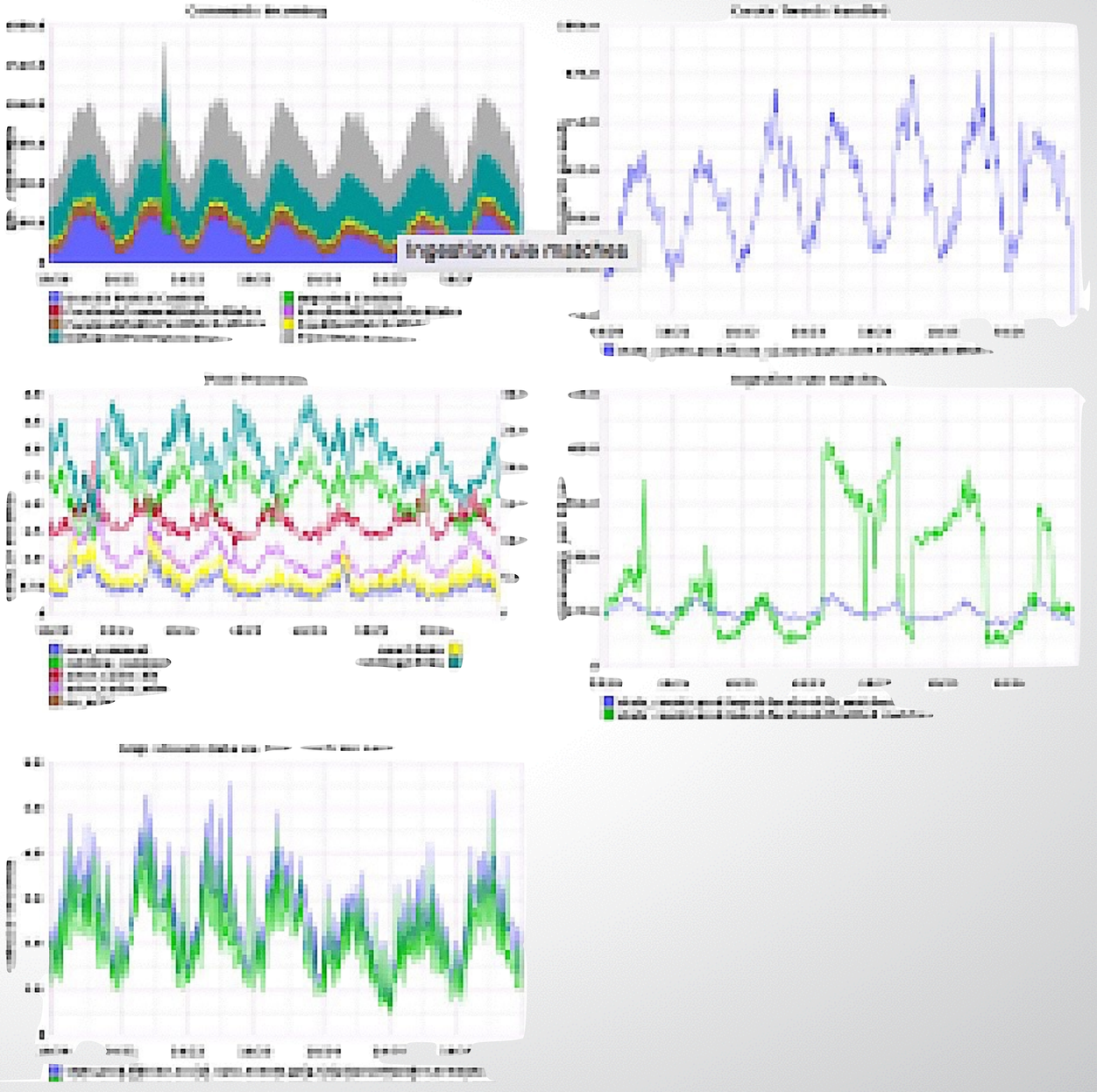
Mo services mo problems

Dashboards

- service vs system health
- correlate “strange events”
- capacity planning
- app specific

Tools

- statsd + graphite + grafana / gdash
- sentry log4j appender
- nagios + pagerduty



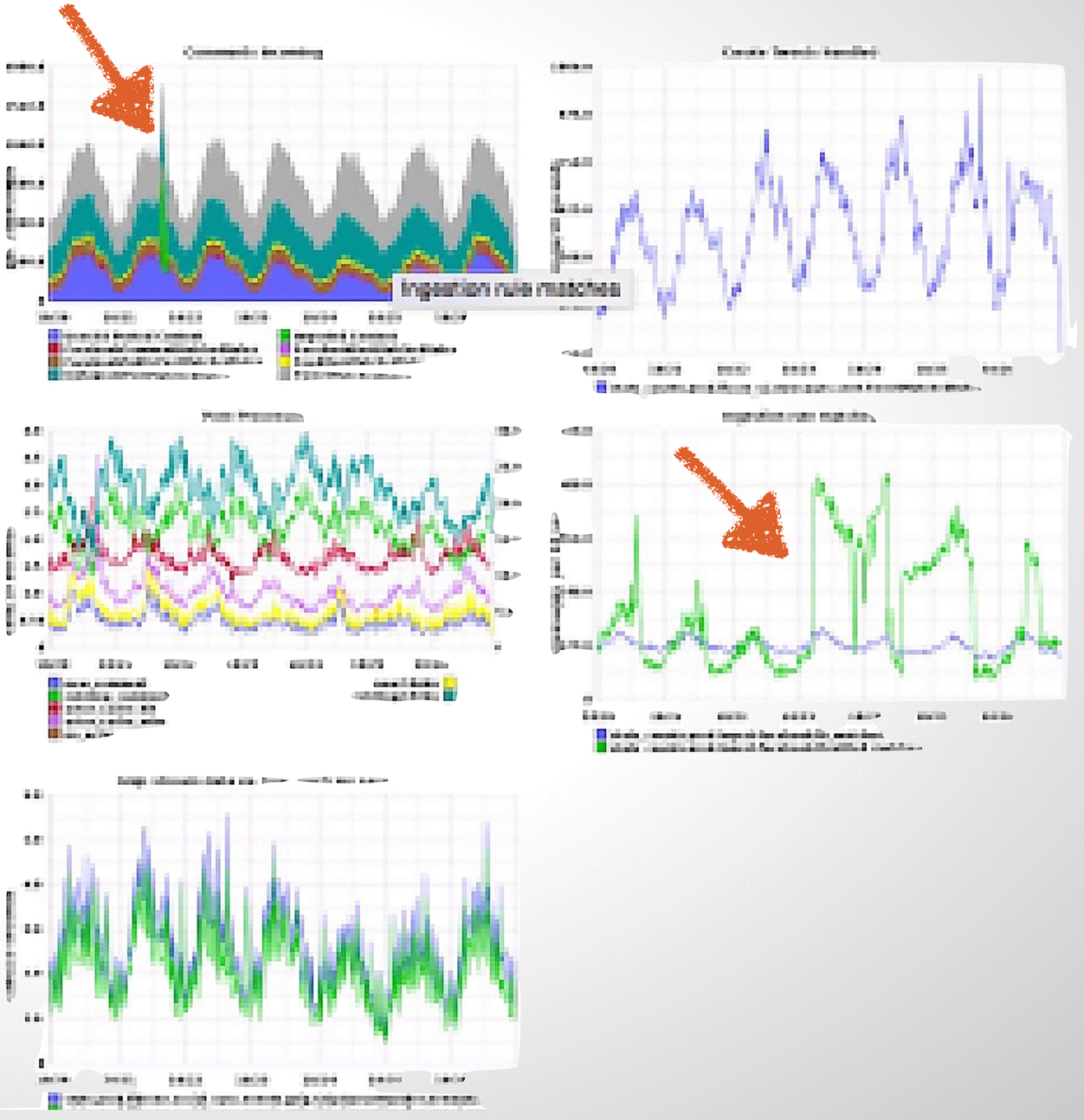
Mo services mo problems

Dashboards

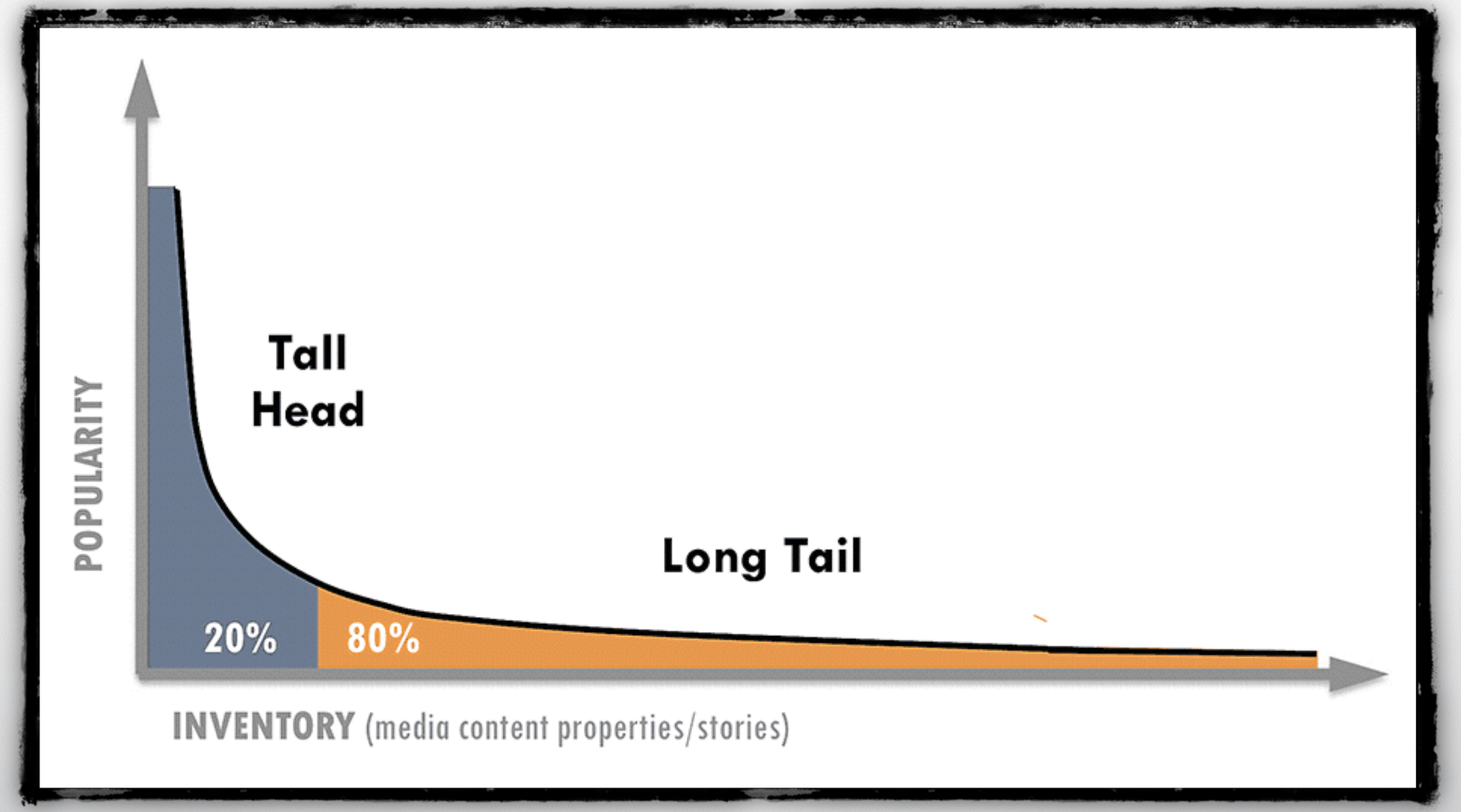
- service vs system health
- correlate “strange events”
- capacity planning
- app specific

Tools

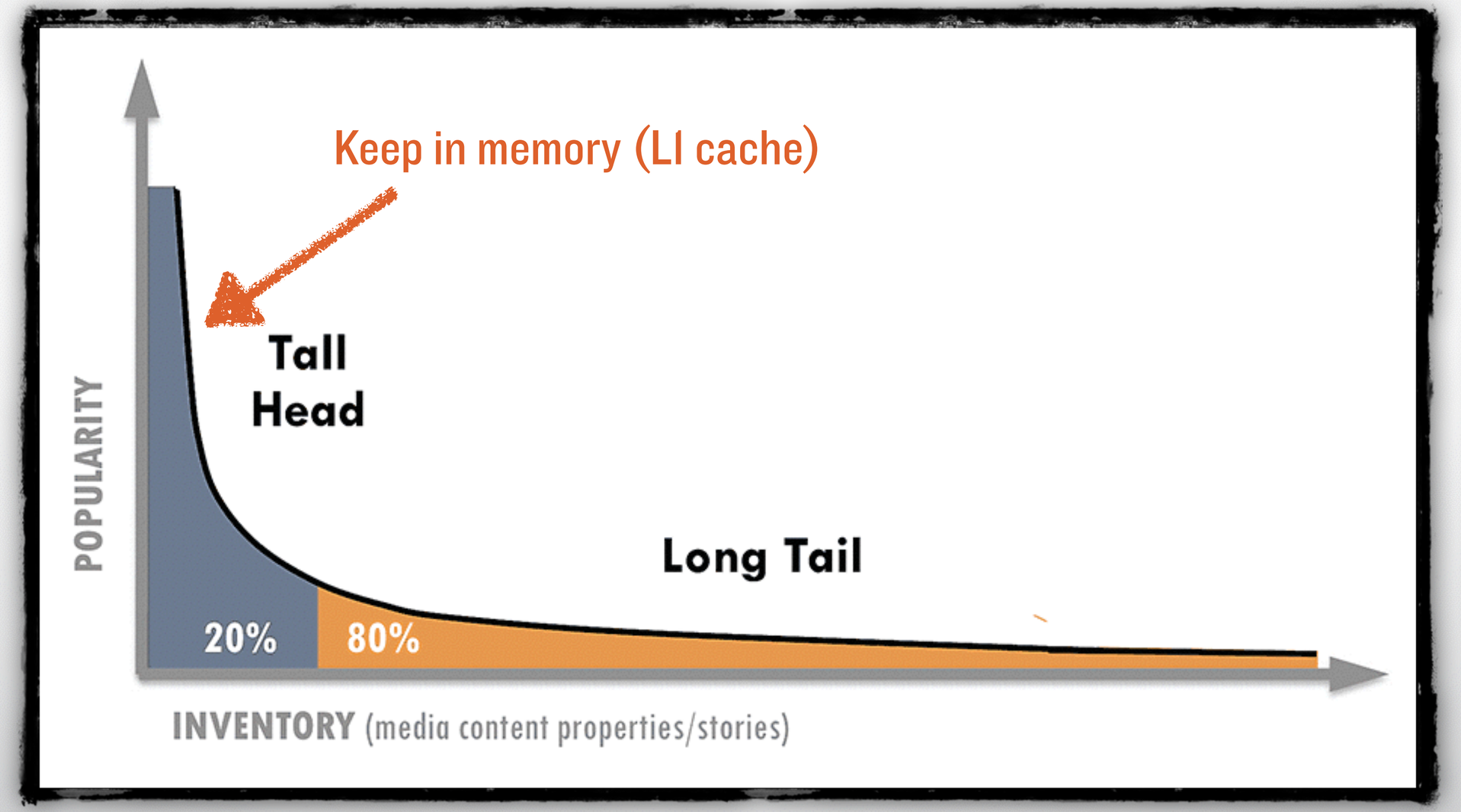
- statsd + graphite + grafana / gdash
- sentry log4j appender
- nagios + pagerduty



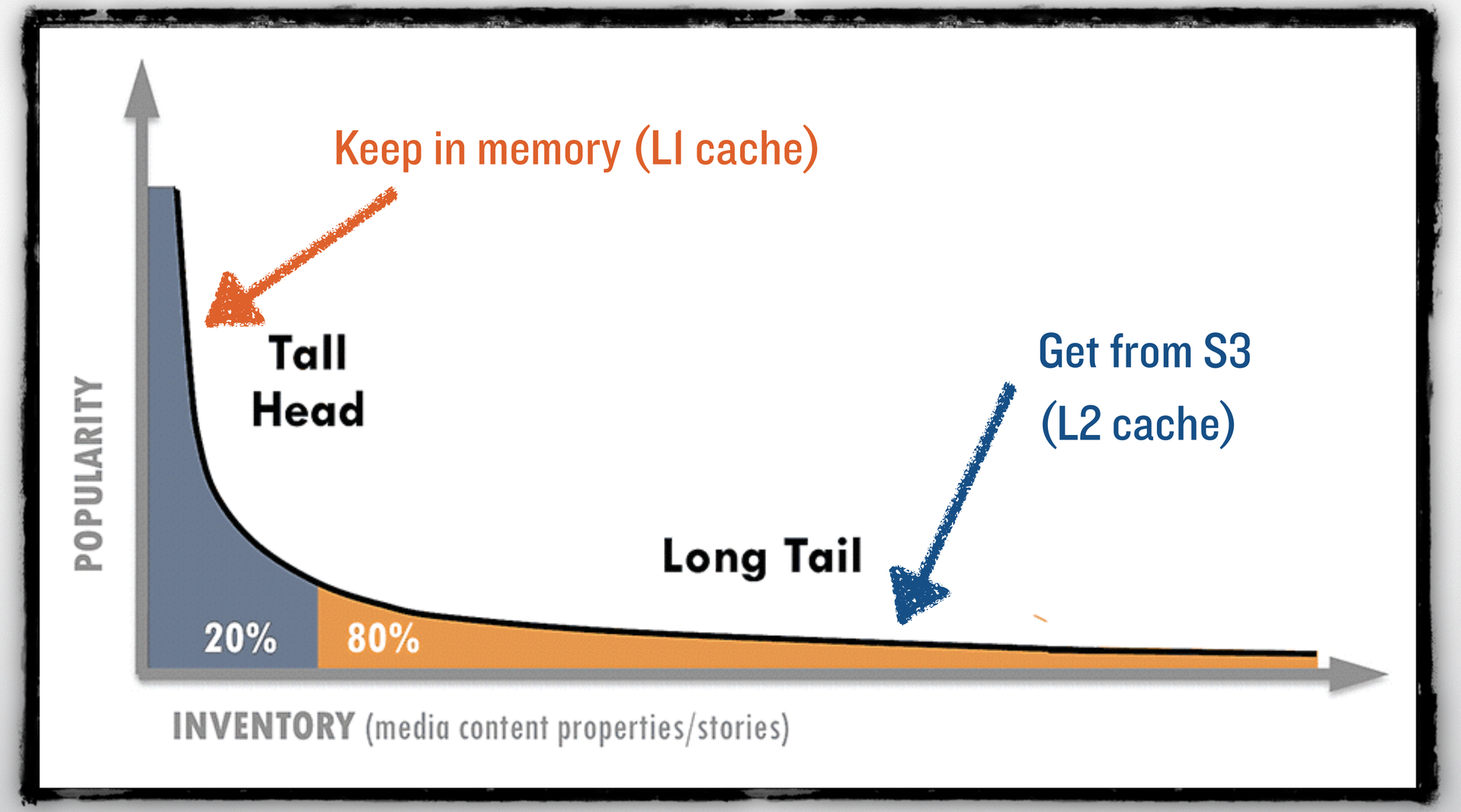
Request distribution or “data access pattern”



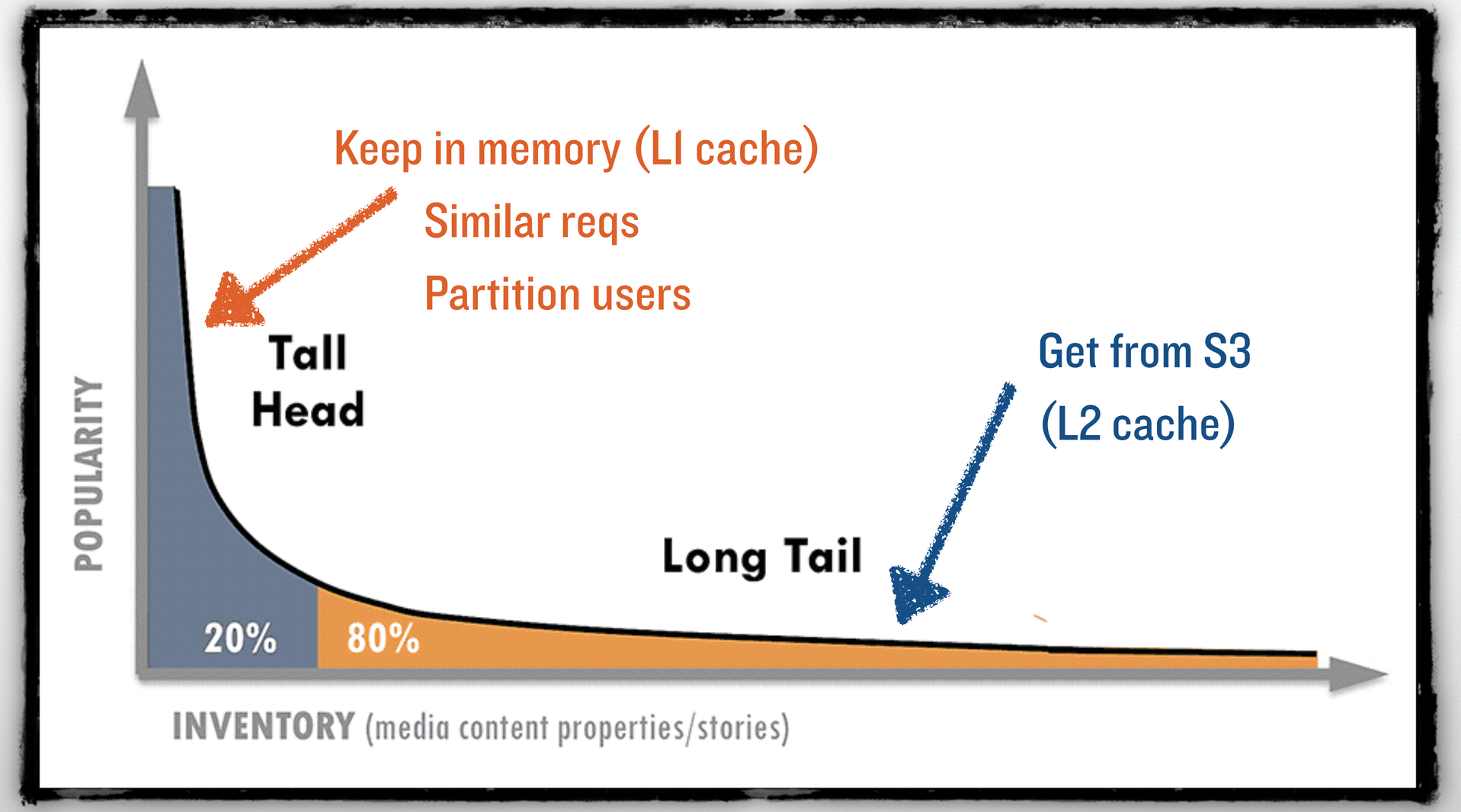
Request distribution or “data access pattern”



Request distribution or “data access pattern”



Request distribution or “data access pattern”



Forcing square pegs in a round hole

- choose the right data stores
  - Database
  - Queue
- sweet spot
  - type of data
  - type of queries
  - some optimized for write
  - some optimized for indexing
- trade off of speed and consistency

Call me maybe - a story of unreliable communication

CARLY RAE JEPSEN

https://aphyr.com/tags/Jepsen

3/BUILDING BLOCKS

Throttling - Leaky bucket algorithm

- capped output flow

  regardless of input flow
- accrue output allowance over time
- drop requests if insufficient allowance
- cost function

```
# 1 item per interval
allowance = rate = 1
# 10 sec interval
throttle_interval = 10
# 1req/10sec = 0.1 qps
qps = rate / throttle_interval
last_check = time()

def throttle(item):
    current = time() # or item.created_at
    size = cost(item) # [0..1]
    time_passed = current - last_check
    last_check = current
    allowance += time_passed * qps
    # Cap to rate
    allowance = min(rate, allowance)

    if allowance < size:
        return True
    allowance -= size
    return False
```

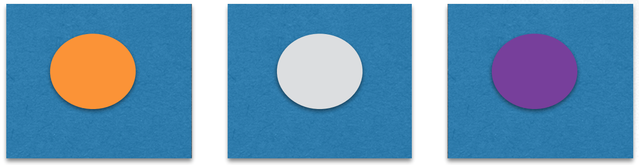
Counting 'Heavy Hitters' - Space Saving Algorithm

```
counts = { } # map of item to count
errors = { } # map of item to error count

for item in stream:
    if len(counts) < k:
        counts[item] += weight
    else:
        if item in counts:
            counts[item] += 1
        else:
            prev_min = item_with_min_count(counts)
            counts[item] = counts[prev_min] + 1
            errors[item] = counts[prev_min]
            counts.remove_key(prev_min)
```

- unbounded stream
- TOP-K in constant space
  - k * (item, count, error)
- overestimates on replace
  - min(count)
- MIN Heap + HashMap

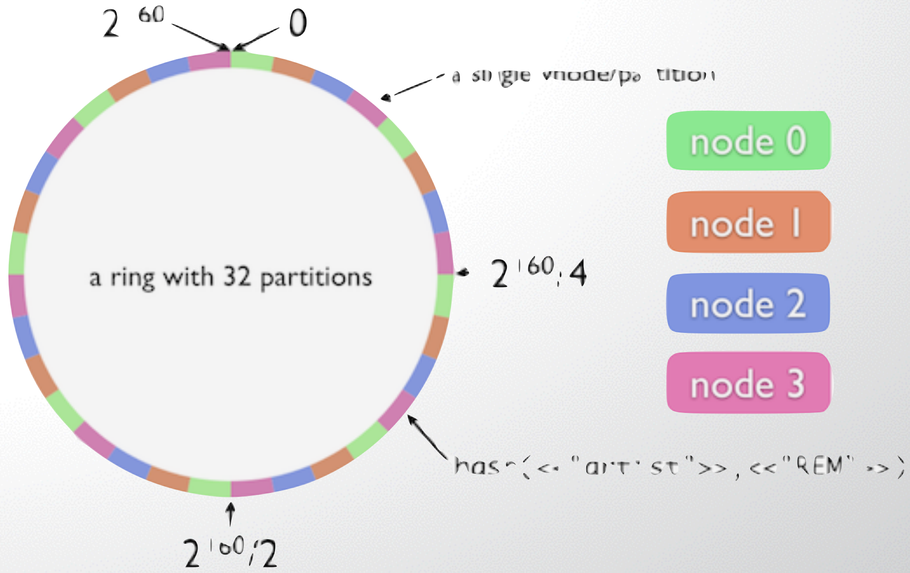
Partitioning - Consistent Hashing



- `article_id % server_count`
  - what if hosts added/removed ?
  - thundering herd!

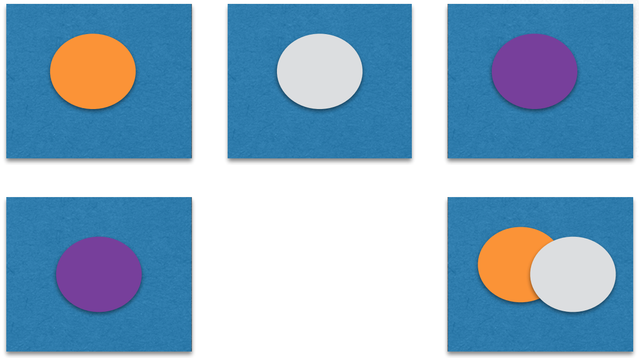
- Hashing.consistentHash(item, server_count)
  - minimizes shuffling

- ConsistentHashRing with virtual nodes
  - `TreeSet` with 100 replicas per node
    - hash("node1:1") .. hash("node1:100")
    - hash("node2:1") .. ("node2:100") ,...
  - `SortedMap.get(hash(item))` or
  - `SortedMap.tailMap(hash(item)).firstKey()`



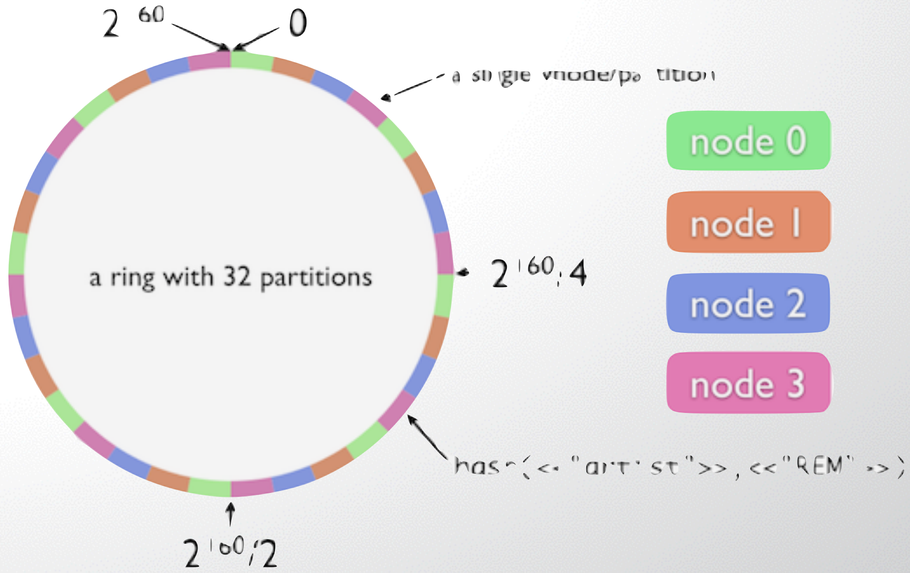
Partitioning - Consistent Hashing

- `article_id % server_count`
  - what if hosts added/removed ?
  - thundering herd!



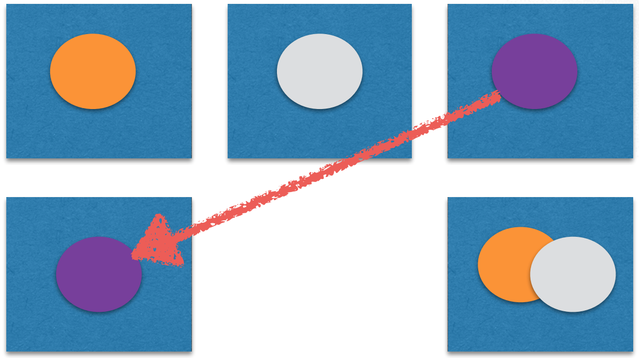
- Hashing.consistentHash(item, server_count)
  - minimizes shuffling

- ConsistentHashRing with virtual nodes
  - `TreeSet` with 100 replicas per node
    - hash("node1:1") .. hash("node1:100")
    - hash("node2:1") .. ("node2:100") ,...
  - `SortedMap.get(hash(item))` or
  - `SortedMap.tailMap(hash(item)).firstKey()`



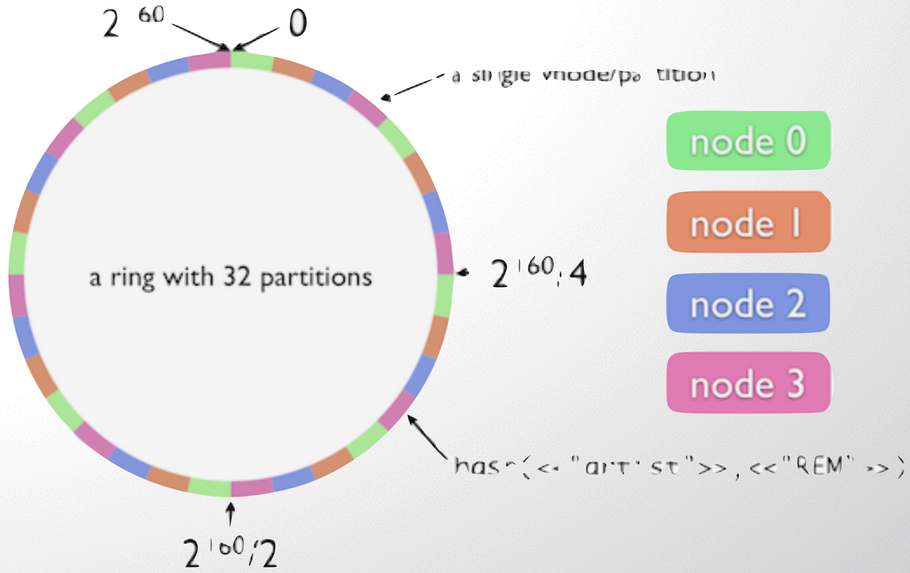
Partitioning - Consistent Hashing

- `article_id % server_count`
  - what if hosts added/removed ?
  - thundering herd!



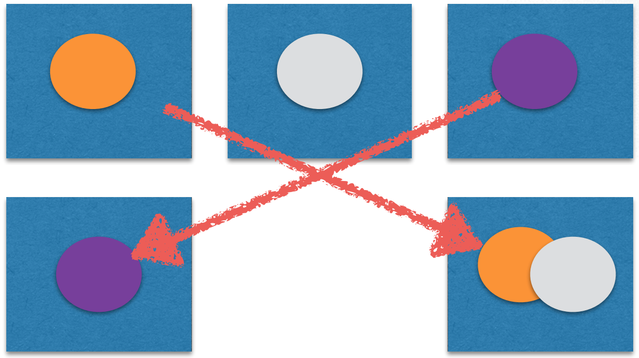
- Hashing.consistentHash(item, server_count)
  - minimizes shuffling

- ConsistentHashRing with virtual nodes
  - `TreeSet` with 100 replicas per node
    - hash("node1:1") .. hash("node1:100")
    - hash("node2:1") .. ("node2:100") ,...
  - `SortedMap.get(hash(item))` or
  - `SortedMap.tailMap(hash(item)).firstKey()`



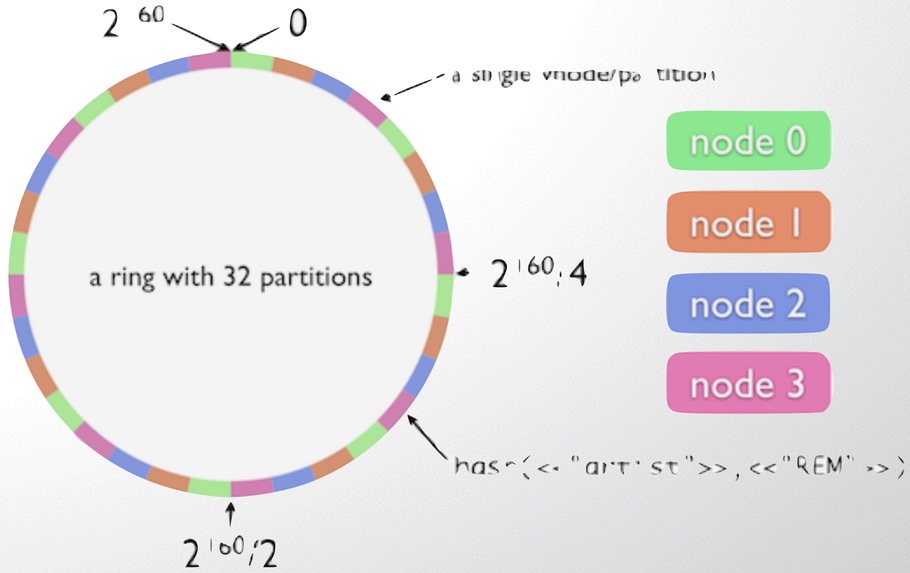
Partitioning - Consistent Hashing

- `article_id % server_count`
  - what if hosts added/removed ?
  - thundering herd!



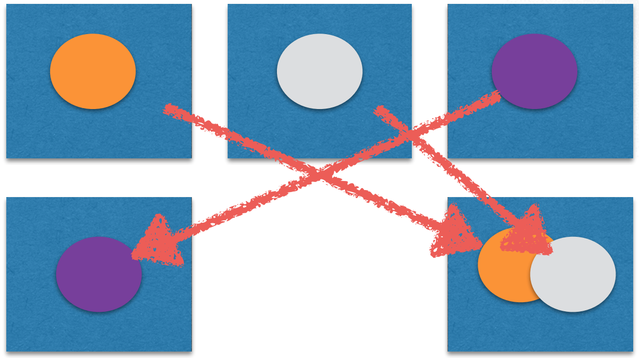
- Hashing.consistentHash(item, server_count)
  - minimizes shuffling

- ConsistentHashRing with virtual nodes
  - `TreeSet` with 100 replicas per node
    - hash("node1:1") .. hash("node1:100")
    - hash("node2:1") .. ("node2:100") ,...
  - `SortedMap.get(hash(item))` or
  - `SortedMap.tailMap(hash(item)).firstKey()`



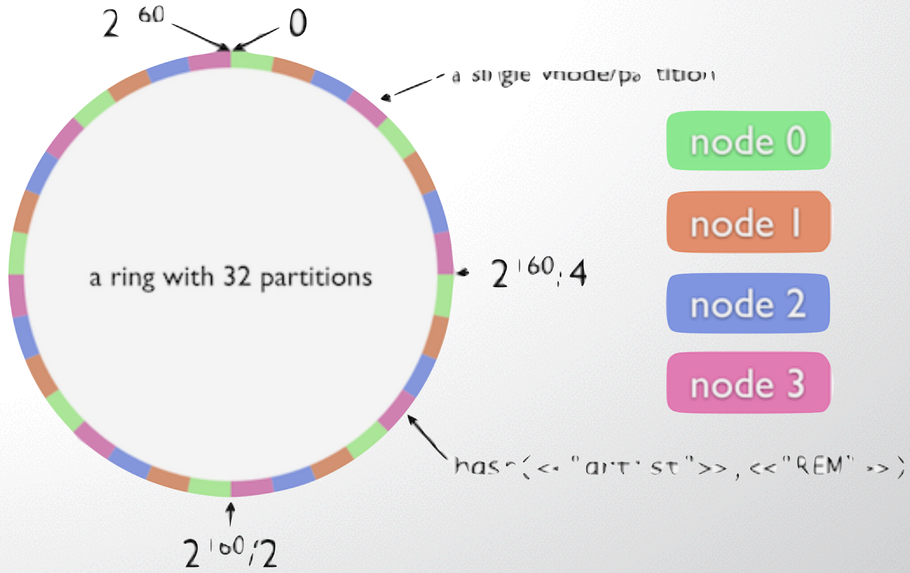
Partitioning - Consistent Hashing

- `article_id % server_count`
  - what if hosts added/removed ?
  - thundering herd!



- Hashing.consistentHash(item, server_count)
  - minimizes shuffling

- ConsistentHashRing with virtual nodes
  - `TreeSet` with 100 replicas per node
    - hash("node1:1") .. hash("node1:100")
    - hash("node2:1") .. ("node2:100") ,...
  - `SortedMap.get(hash(item))` or
  - `SortedMap.tailMap(hash(item)).firstKey()`



Membership test - Bloom Filters

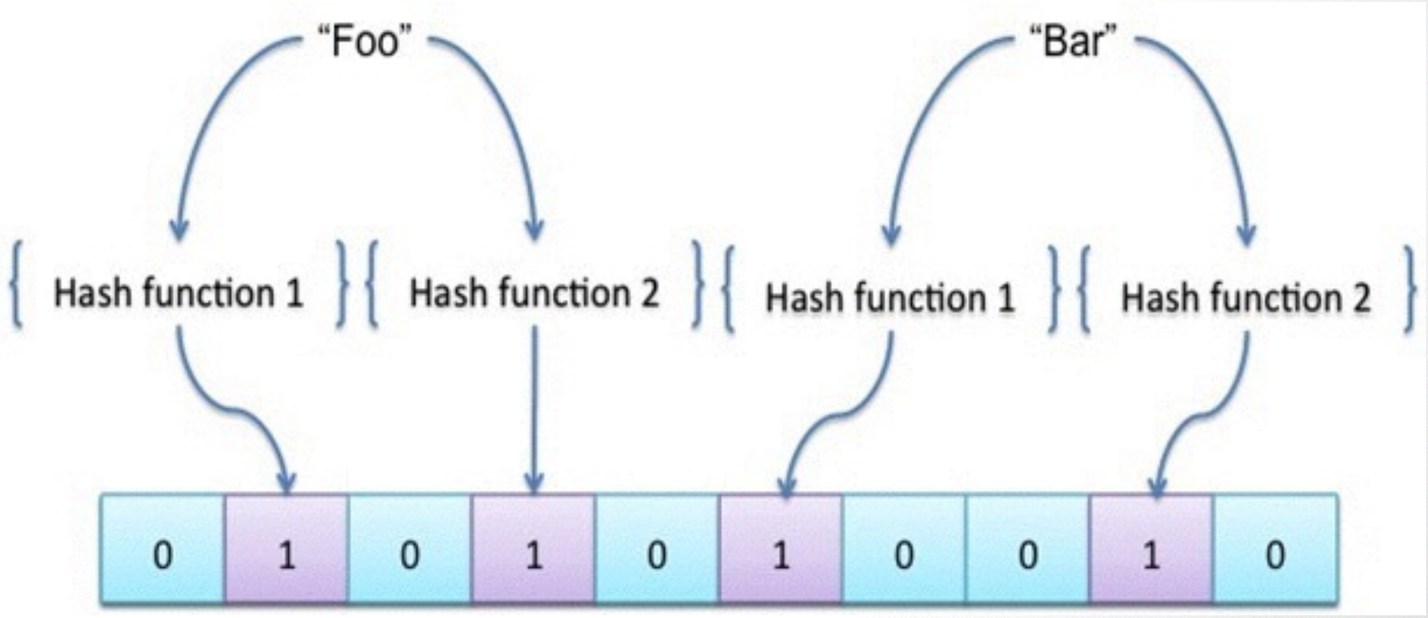
- very memory efficient
- almost as fast as CHM
- small % false pos
- ZERO false neg
- append only
  - see Cuckoo Filter
- `BloomFilter.create()`

Membership test - Bloom Filters

- very memory efficient
- almost as fast as CHM
- small % false pos
- ZERO false neg
- append only
  - see Cuckoo Filter
- `BloomFilter.create()`

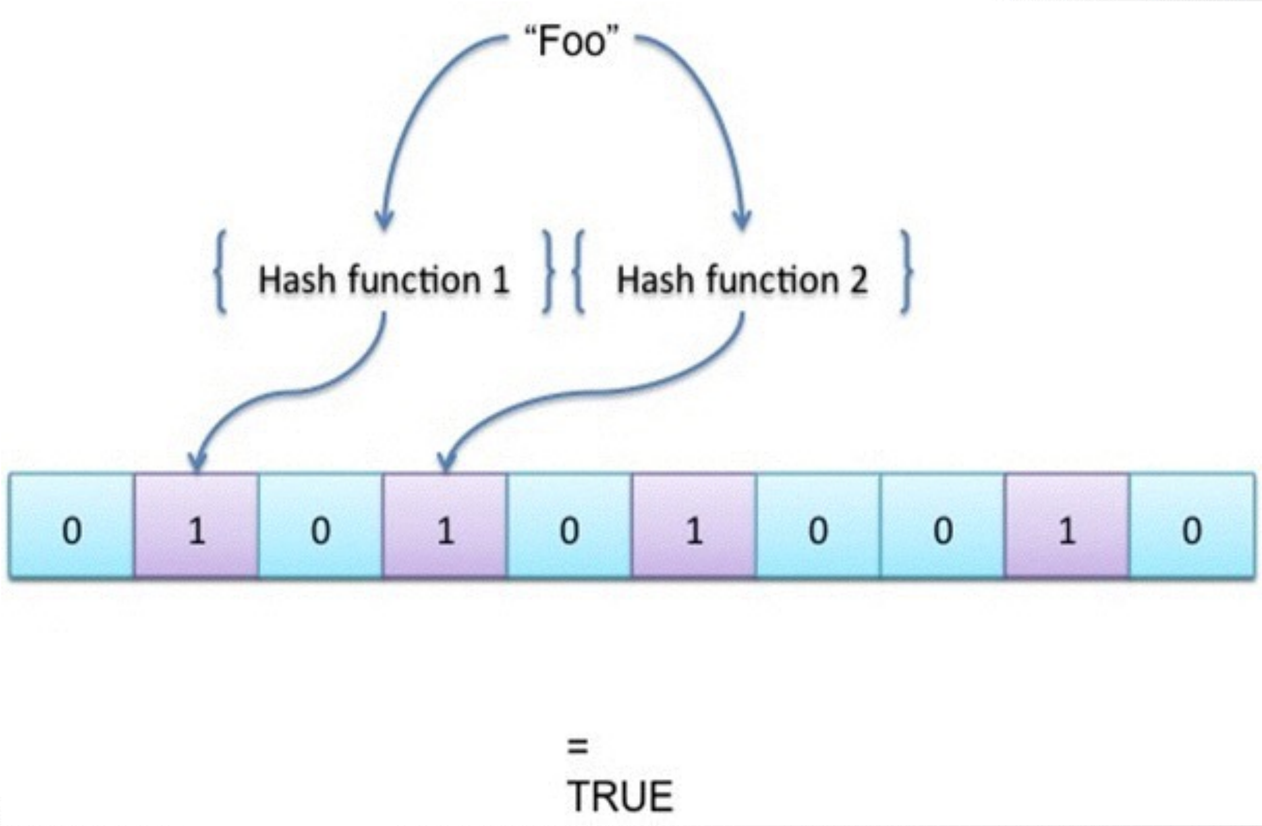
Membership test - Bloom Filters

- very memory efficient
- almost as fast as CHM
- small % false pos
- ZERO false neg
- append only
  - see Cuckoo Filter
- `BloomFilter.create()`



Membership test - Bloom Filters

- very memory efficient
- almost as fast as CHM
- small % false pos
- ZERO false neg
- append only
  - see Cuckoo Filter
- `BloomFilter.create()`



Concurrency for shared resources - Striped Lock

- `ConcurrentHashMap`'s secret
- eg: `ConcurrentBloomFilter`
  - up to n threads non-blocking
  - n shards with a `ReadWriteLock` and `BloomFilter`
  - ConsistentHash index into shards
- `Striped` in Guava

Random Sampling

```
float sampleRate = 0.10f; // 10%

if (ThreadLocalRandom.current().nextFloat() < sampleRate) {

    statsd.increment("high.velocity.request.success");

}
```

- for high velocity events
- NEVER for sparse events

Distributed Consensus - Zookeeper

- metadata store
- set membership
- distributed lock
- leader election
- Netflix Curator
- DON’T TRY THIS AT HOME!

Async IO

- Get up to IM connections, capped by bandwidth
- Netty
  - EPOLL on Linux
  - (Composite)ByteBuf
  - ChannelGroup
  - HashedWheelTimer
  - READ THE SOURCE!
- Others work as well:
  - Vert.x, NodeJS, Python Gevent

Data processing pipelines

- Kafka Queues with many partitions
- Auto-scale group of workers
- commit batches of work to ZK (restart, lag)
- Emit stats (success, error, timing)
- Custom dashboard
  - sampled data from the stream
  - inject data in the stream (debug)

- Future:
  - Spark Streaming
  - Mesos + Marathon + Chronos

Mechanical Sympathy

- Disruptor, lock-free Queue
- BlockingQueue - backpressure!
- JCTools - Multi Prod Single Cons Queue
- CAS - Atomic* & Unsafe
- OpenHFT
  - off-heap storage
  - cpu affinity for JVM threads
  - zero allocation hashing
- mechanical-sympathy.blogspot.com

THANK YOU

Jo Voordeckers

SR. SOFTWARE ENGINEER - LF PLATFORM

Email: jvoordeckers@livefyre.com

@jovoordeckers

livefyre

San Francisco, CA
New York, NY
London, UK

@livefyre.com
press.livefyre.com
blog.livefyre.com