Making Session Stores More Intelligent

KYLE J. DAVIS
TECHNICAL MARKETING MANAGER
REDIS LABS

What is a session store?

redislabs

A session store is…

- An chunk of data that is connected to one “user” of a service
  - ”user” can be a simple visitor
  - or proper user with an account
- Often persisted between client and server by a token in a cookie*
  - Cookie is given by server, stored by browser
  - Client sends that cookie back to the server on subsequent requests
  - Server associates that token with data
- Often the most frequently used data by that user
  - Data that is specific to the user
  - Data that is required for rendering or common use
- Often ephemeral and duplicated

redislabs
home of redis

Session Storage Uses Cases

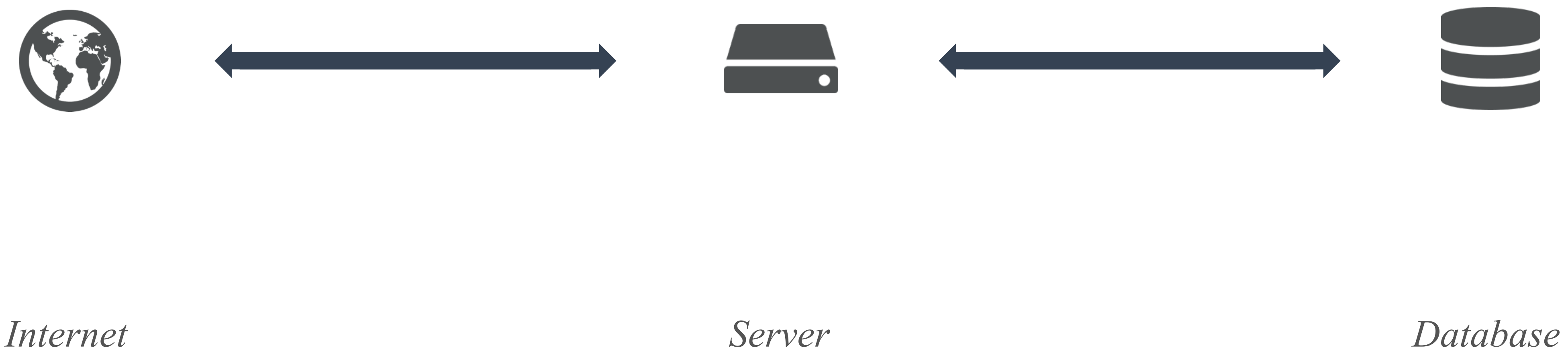
Traditional

- Username
- Preferences
- Name
- “Stateful” data

Intelligent

- Traditional +
- Notifications
- Past behaviour
  - content surfacing
  - analytical information
  - personalization

In a simple world

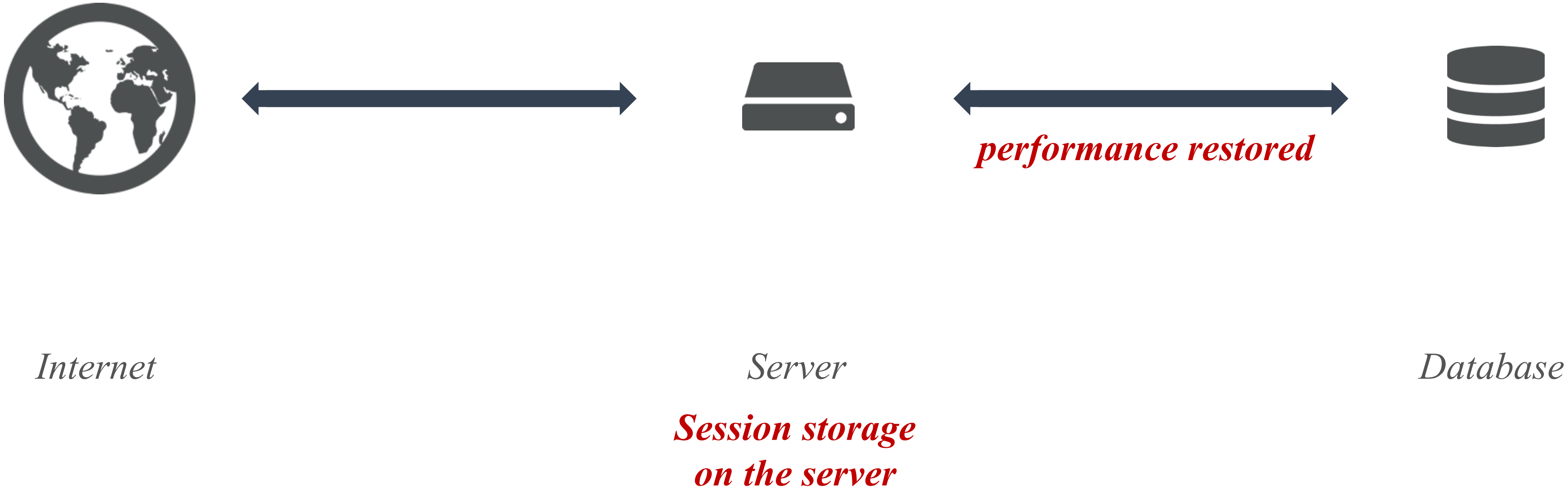


Good problems

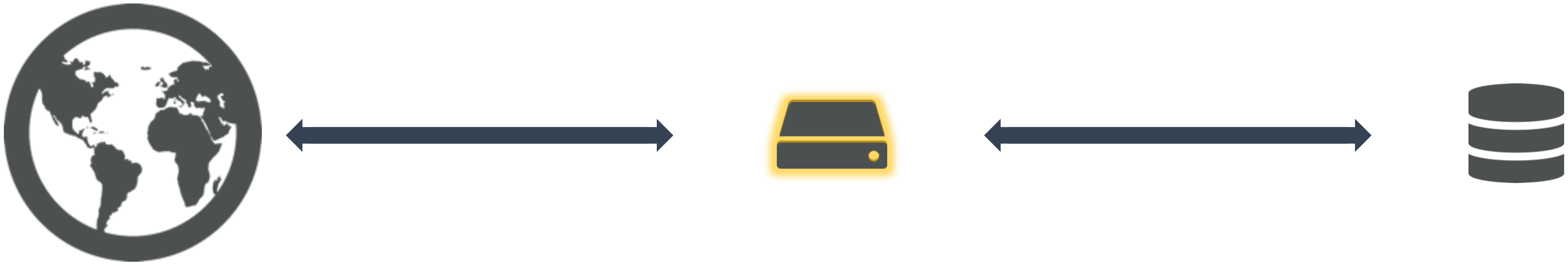
Internet *Server* *Database*

Traffic Grows… ***Struggles***

Good solution



More good problems

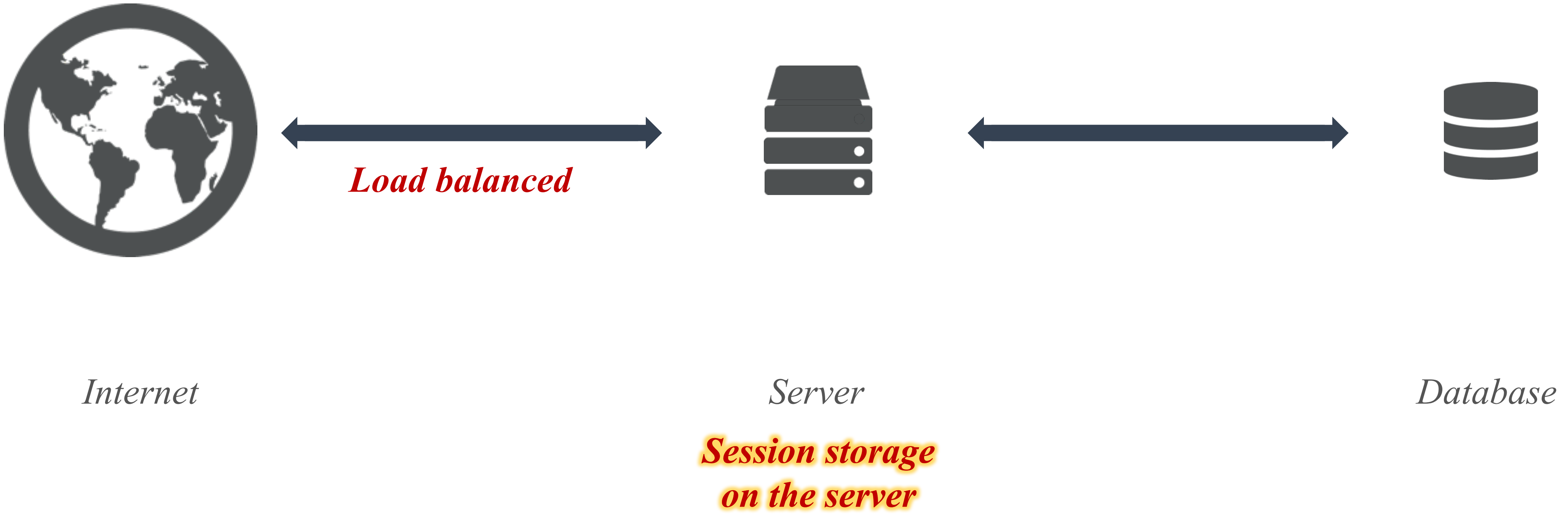


Struggling

Internet *Server* *Database*

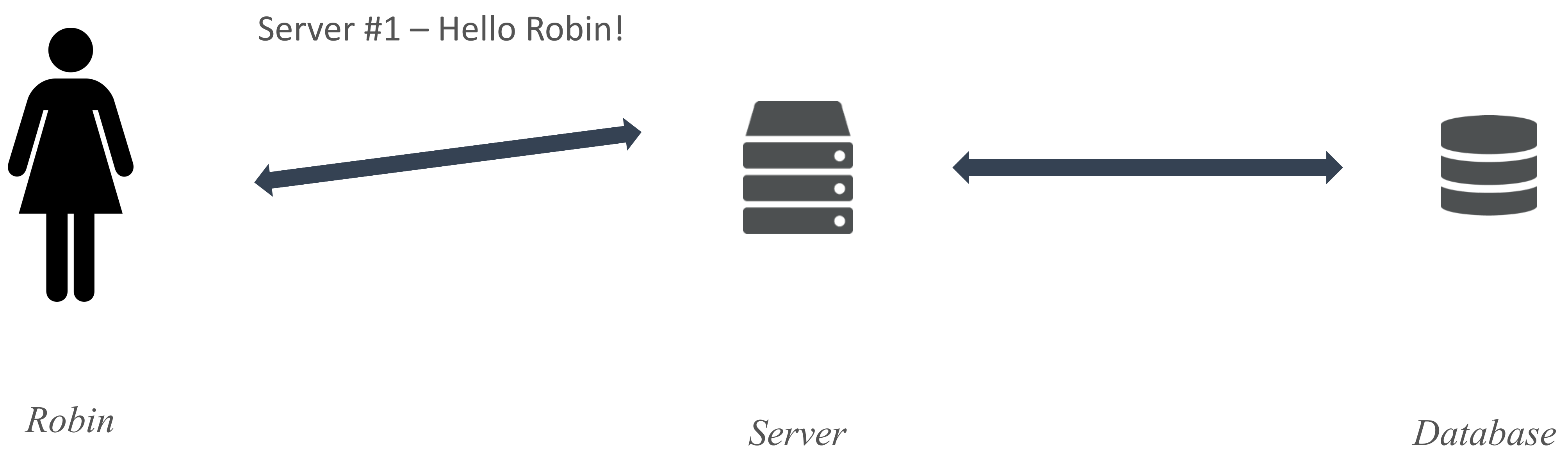
Session storage on the server

Problematic Solutions

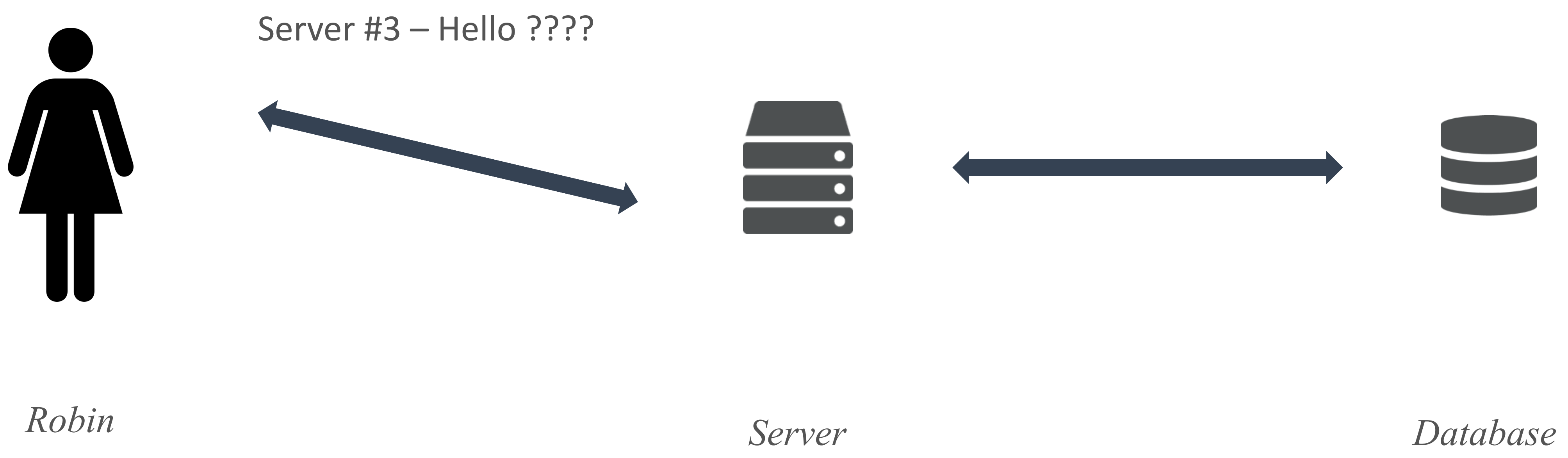


redislabs
home of redis

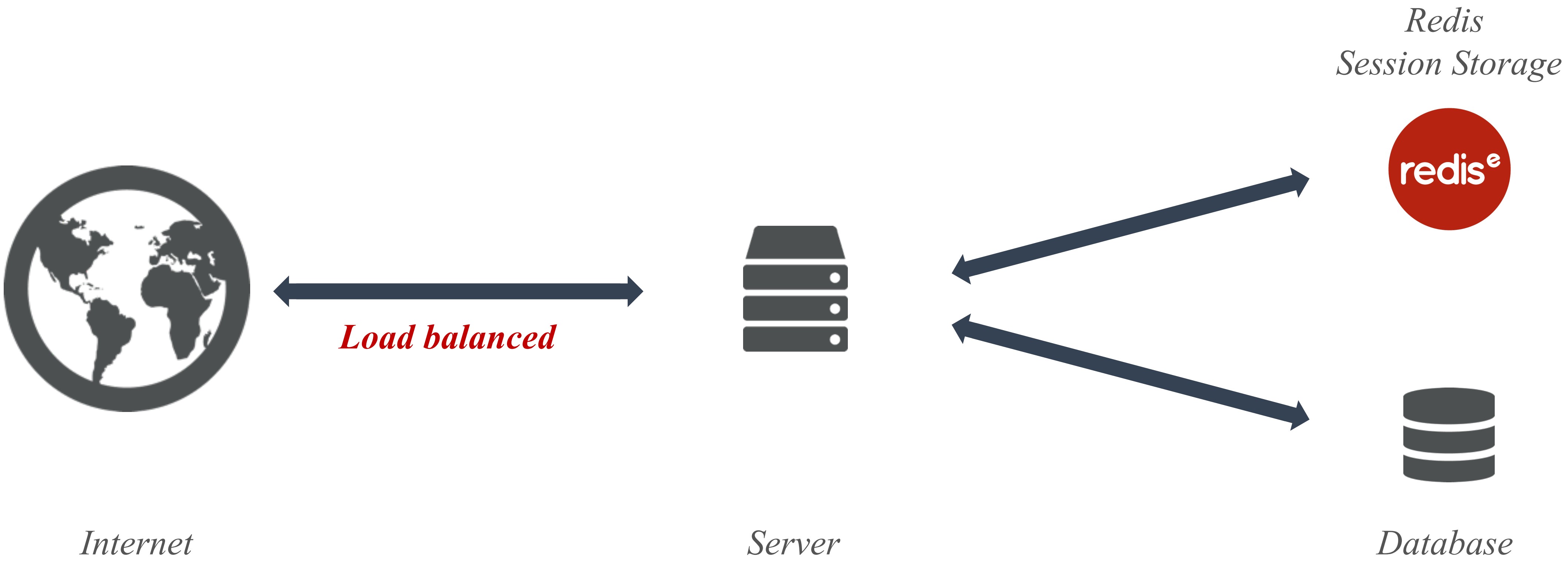
Multiple Servers + On-server Sessions?



Multiple Servers + On-server Sessions?



Better solution



What is Redis?

redislabs

Who We Are

redis

Open source. The leading **in-memory database platform**, supporting any high performance operational, analytics or hybrid use case.

redislabs
home of redis

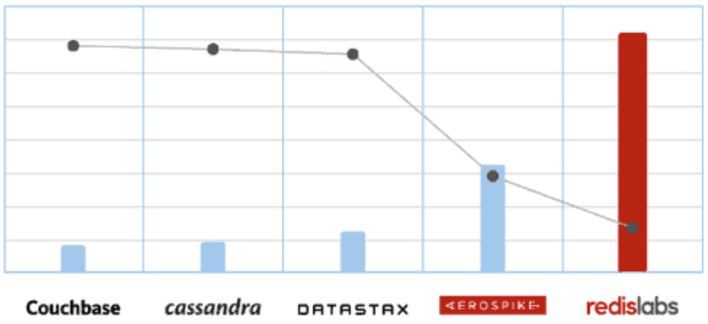
The open source home and commercial provider of **Redis Enterprise** technology, platform, products & services.

Redis Top Differentiators

1

Performance

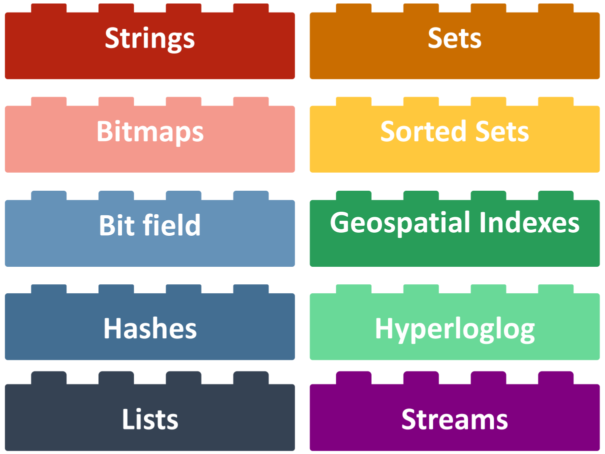
NoSQL Benchmark



2

Simplicity

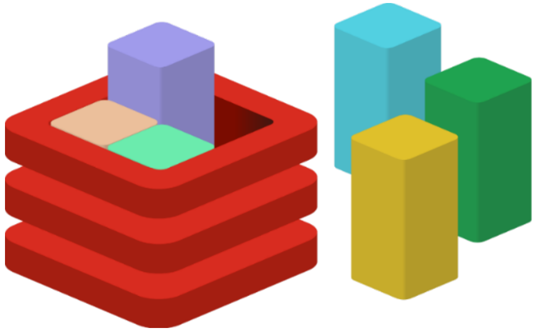
Redis Data Structures



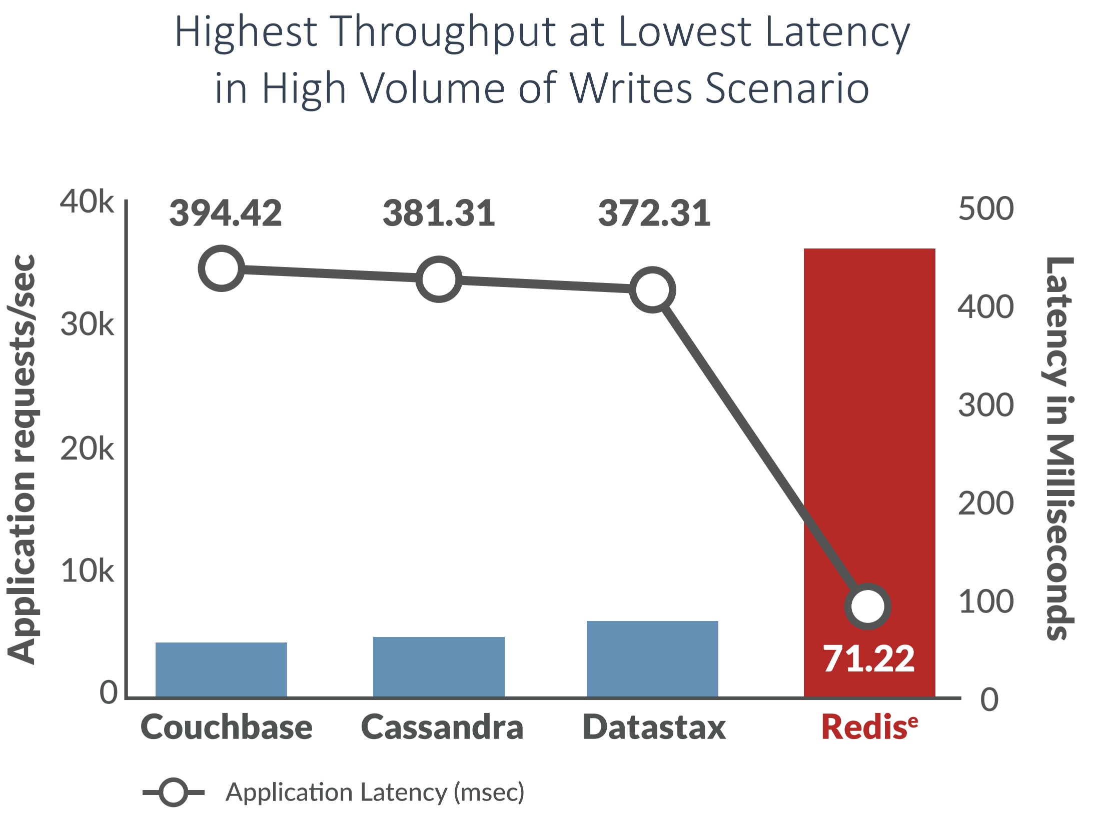
3

Extensibility

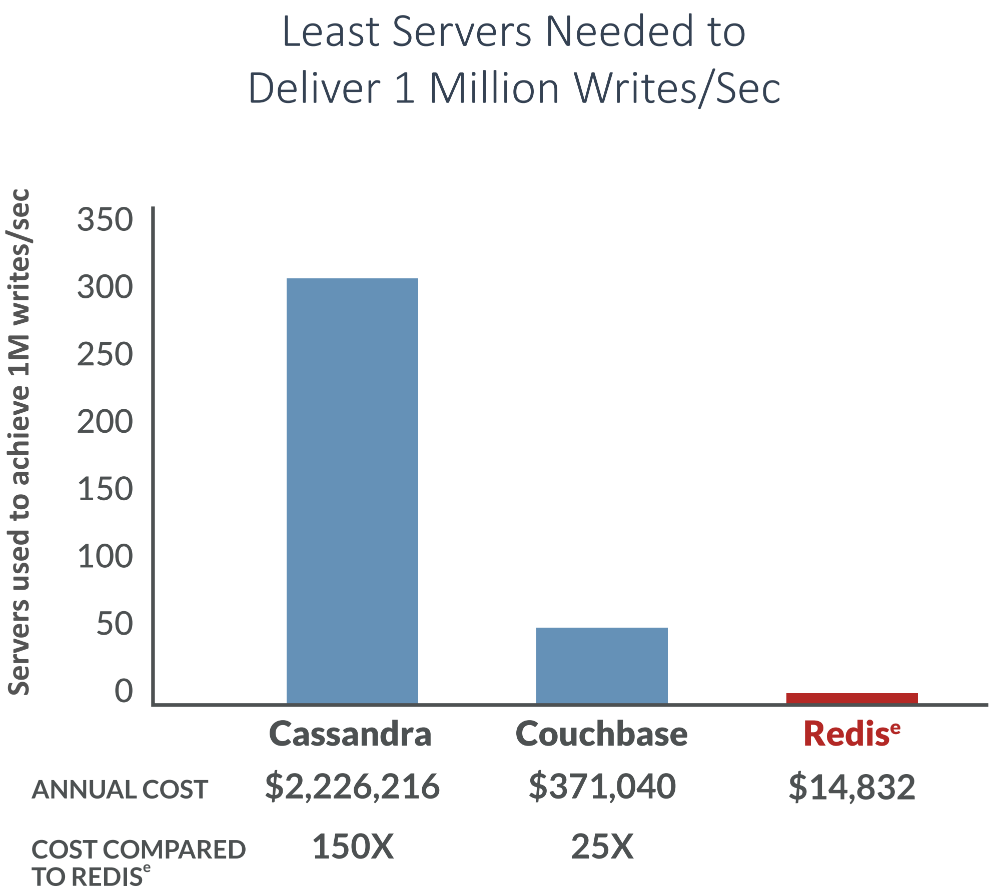
Redis Modules



1 Performance: The Most Powerful Database

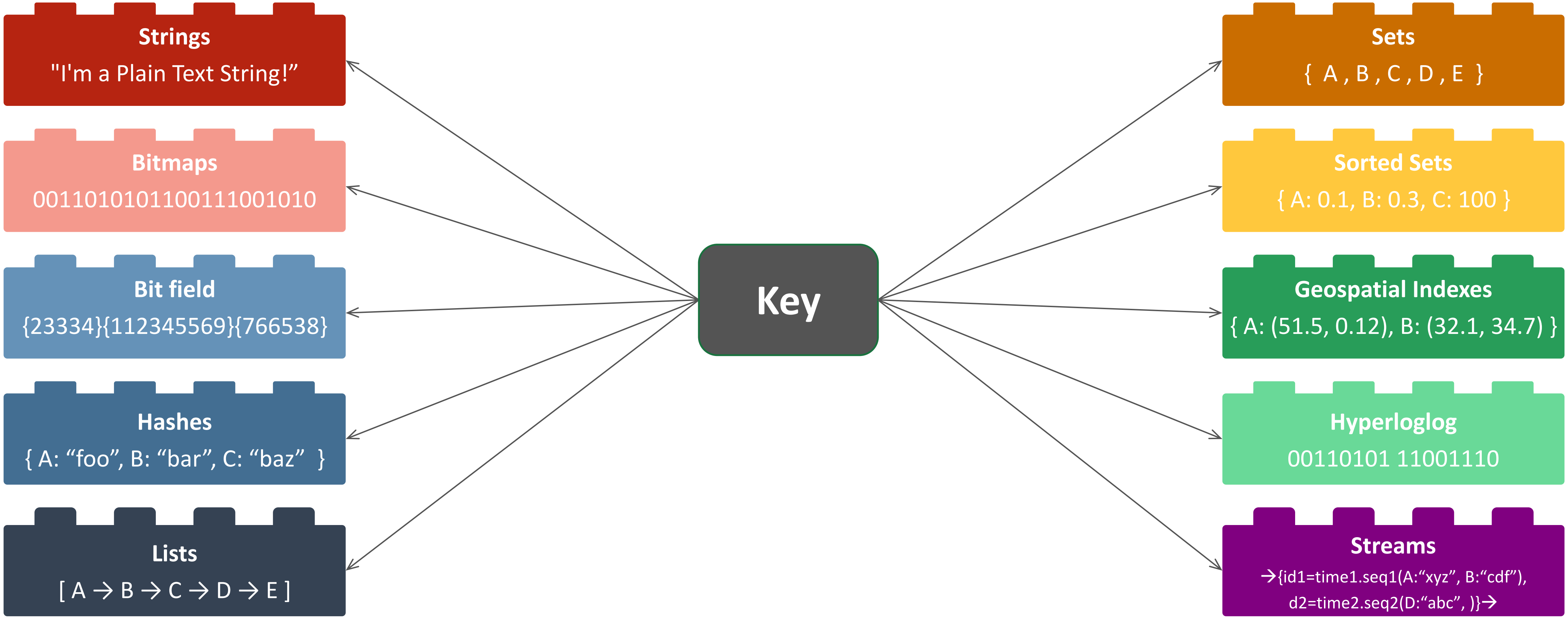


Benchmarks performed by Avalon Consulting Group



Benchmarks published in the Google blog

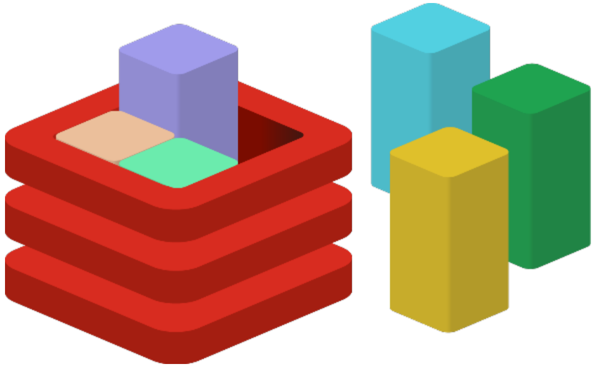
2 Simplicity: Data Structures - Redis' Building Blocks



"Retrieve the e-mail address of the user with the highest bid in an auction that started on July 24th at 11:00pm PST" = ***ZREVRANGE 07242015_2300 0 0***

redislabs
home of redis

3 Extensibility: Modules Extend Redis Infinitely



- Add-ons that use a Redis API to seamlessly support additional use cases and data structures.
- Enjoy Redis' simplicity, super high performance, infinite scalability and high availability.
- Any C/C++/Go program can become a Module and run on Redis.
- Leverage existing data structures or introduce new ones.
- Can be used by anyone; Redis Enterprise Modules are tested and certified by Redis Labs.
- Turn Redis into a **Multi-Model** database

Redis Labs Products

DBaaS ← redis^e → Software

Redis^e Cloud

Fully managed, server-less Redis^e service on hosted resources within AWS, MS Azure, GCP, IBM Softlayer, Heroku, CF and OpenShift

RAM

Redis^e Cloud Private

Fully managed, server-less scaling Redis^e service in VPCs within AWS, MS Azure, GCP and IBM Softlayer

RAM or Flash

Redis^e Pack

Downloadable Redis^e software for any enterprise datacenter or cloud environment

RAM or Flash

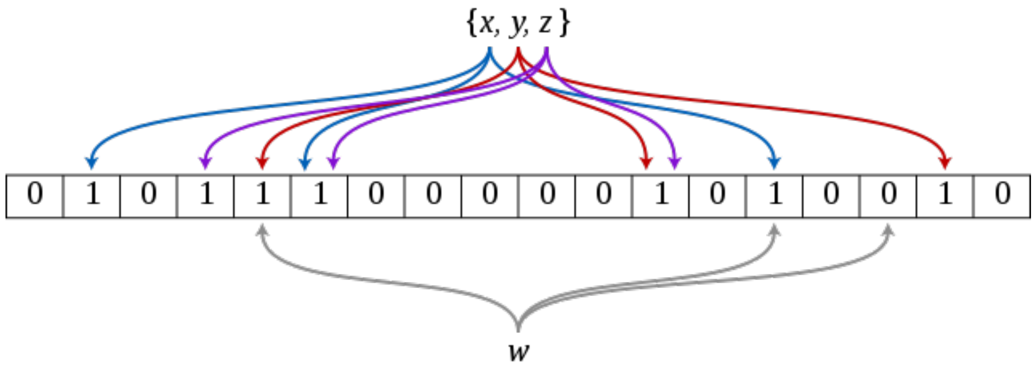
Redis^e Pack Managed

Fully managed Redis^e Pack in private datacenters

RAM or Flash

Concepts

Concept: Bloom Filters (presence)

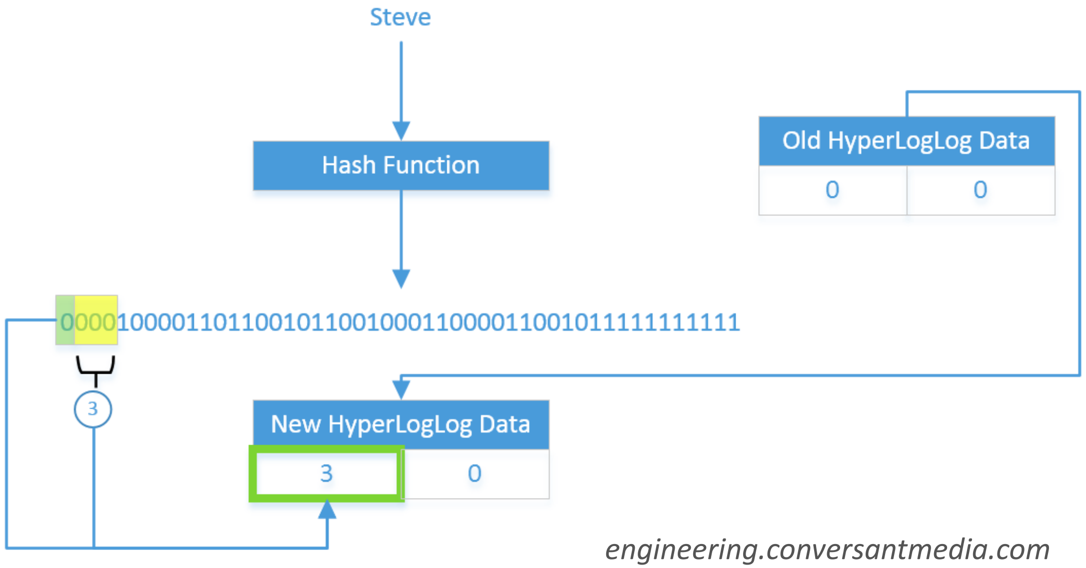


- Probabilistic data structure
- Hash -> sample bits -> set bits
- Properties:
  - False negatives – not possible
  - False positives – possible, but controllable
  - Bits per item stored
  - Add or check if exists
  - Like the Tardis, it’s bigger on the inside than outside
- Availability:
  - Redis Module
  - On top of bitfields

redislabs
home of redis

Concept: HyperLogLog (cardinality)

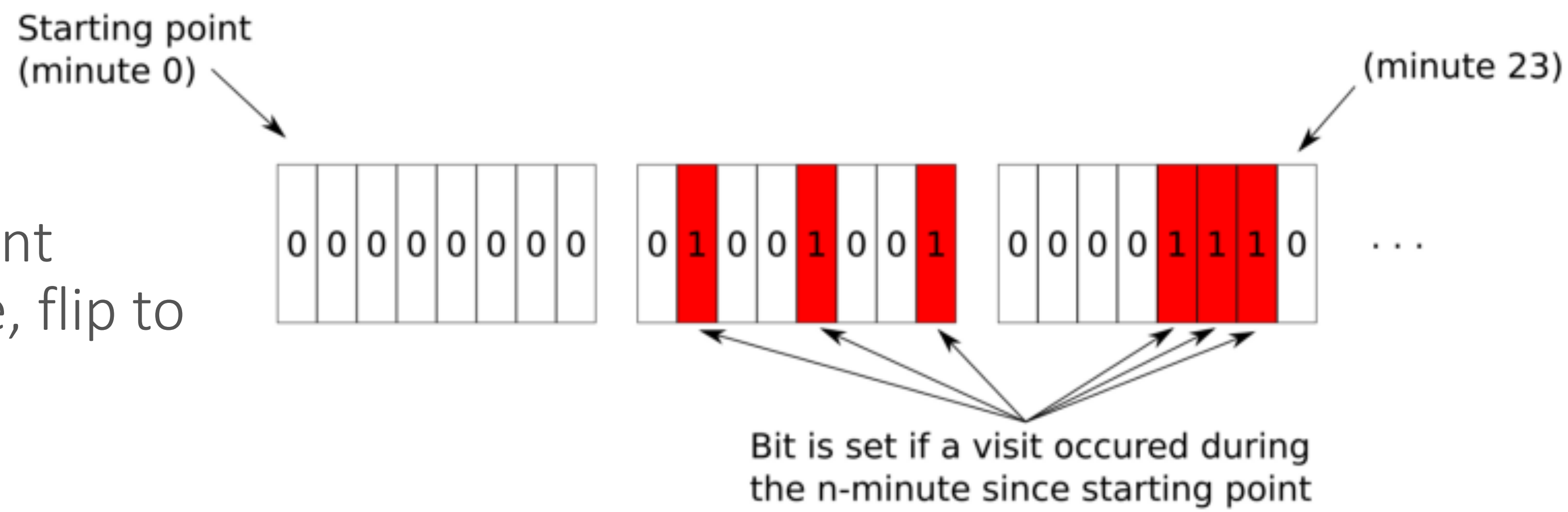
- Probabilistic data structure
- Hash -> count runs -> store runs
- Properties:
  - Estimates unique items
  - Bits per item stored – 2^{64} unique items in 12kb / error rate 0.81%
  - Add, count or merge!
  - Like the Tardis, it's bigger on the inside than outside
- Availability:
  - All versions of Redis



engineering.conversantmedia.com

Concept: Bit counting (time series)

- It's just bits!
- Fixed starting point, each point represents a moment in time, flip to represent activity
- Properties:
  - Size relative to length of time (byte round)
  - Count totals or ranges
  - BITOP (AND/XOR/OR/NOT)
- Availability:
  - All versions of Redis



redislabs
home of redis

Group Notifications

redislabs

Traditional Group Notification Pattern

Process

- Group of users get notification “Sale on sweaters”
- Insert into central table of notifications
- Insert row in table with each user of group with notification and seen flag
- Each time it is needed, query notifications table where seen flag is false.

Traditional Group Notification Pattern

Challenges

- Adding/removing means touching a row for each user in group.
  - Fine for groups of 10 users, what about 1 million?
  - Also multi-step
- Storage is proportional to size of group and notifications
- Constant DB hits, not easily cacheable
- Setting “read” is DB write

Modern & Intelligent Group Notification Pattern

Process

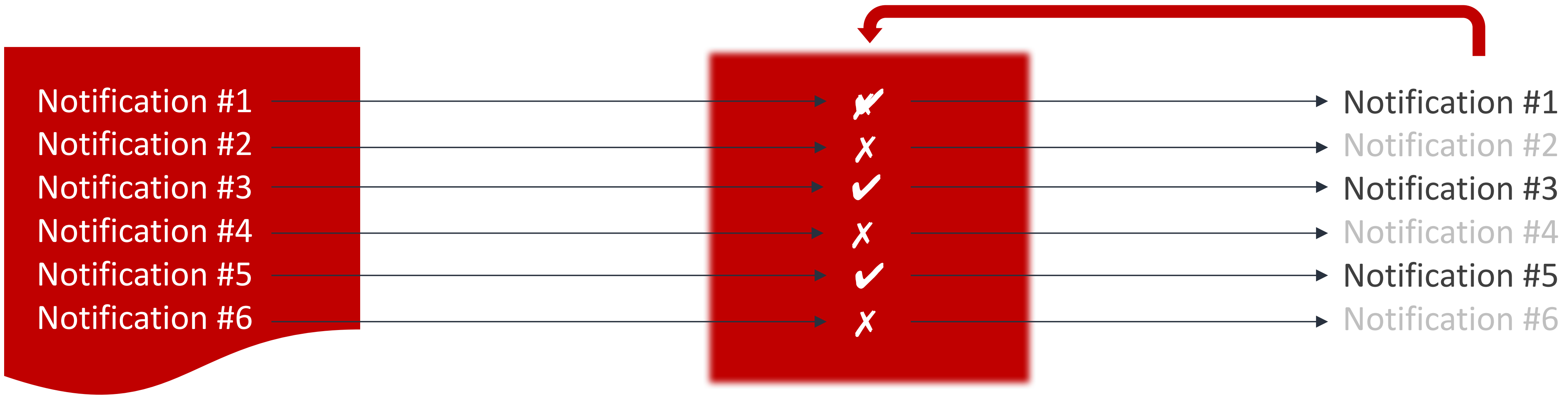
- Add notification to single group based structure or table (easily cacheable)
- First *n* notifications are read by all users in group.
- The notifications are checked to see if they are in a session-based Bloom filter or not.
- Mark read by adding to Bloom filter in session store.

Modern & Intelligent Group Notification Pattern

Advantages

- Adding a notification only writes to a single table, single row.
- Model fits use – unread assumed.
- **Fast**. Checking for read / writing read is unrelated to number of items in the filter. Consistent.
- ~5-*bits* per item, but Bloom filter doesn't always grow.
- Gentle scaling

Visual



redislabs
home of redis

Fresh Content

redislabs

Traditional Content Surfacing Pattern (Basic)

Process

- Hand pick and rotate a small number of content/items
- Stored in DB table
- Served out dumbly to users

Challenges

- May serve content multiple times
- Freshness is linked to a manual curatorial process

Traditional Content Surfacing Pattern (Advanced)

Process

- Batch process builds content list to surface for each user
- List is stored in DB Table
- Served out to user
- Rotated on a schedule

Challenges

- Not Real-time
- May serve content multiple times
- Un-cacheable DB content
- Hard to scale

Modern & Intelligent Content Surfacing Pattern

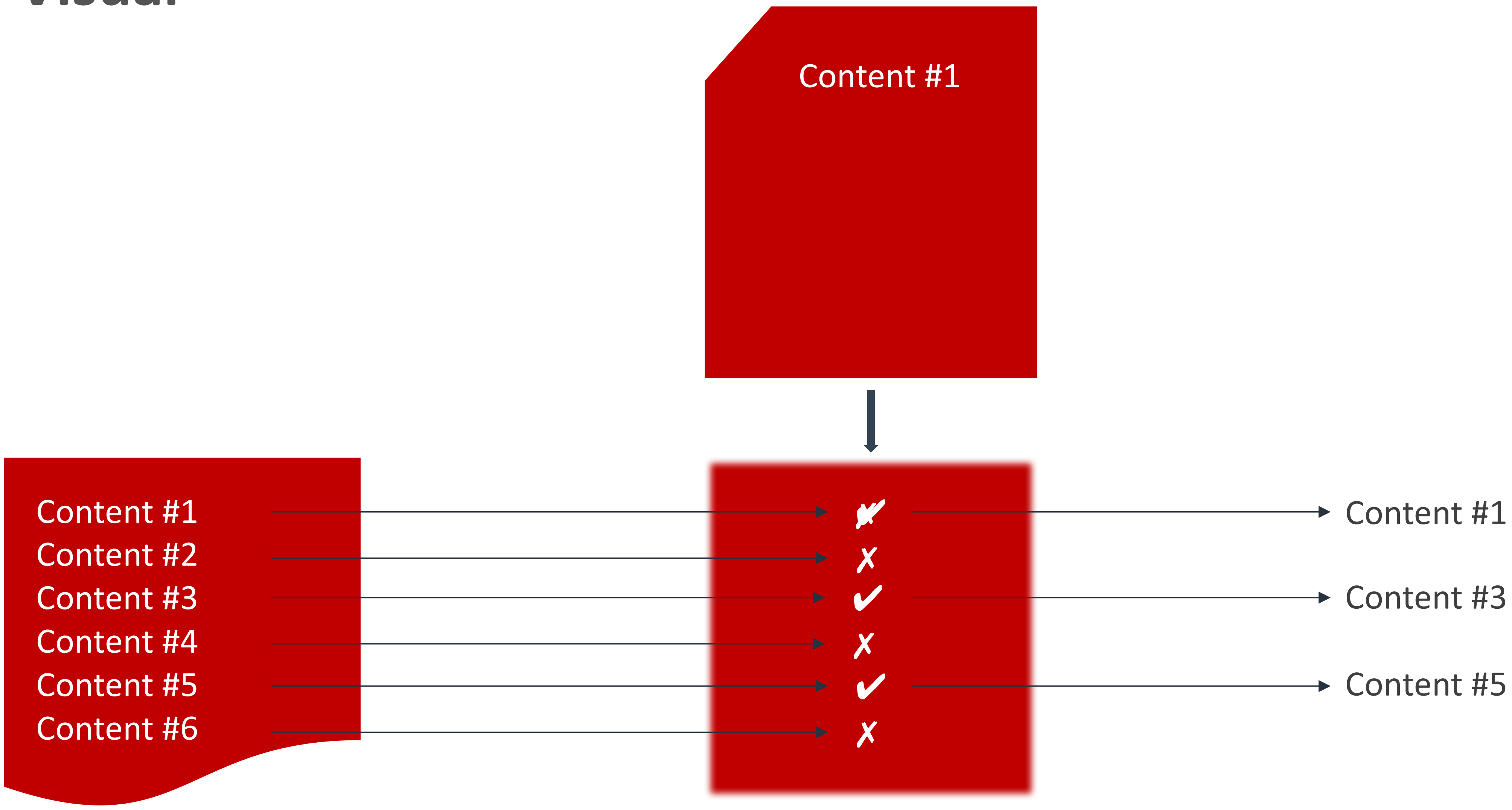
Process

- Middleware adds each content read to a Bloom filter stored in the session
- Featured content list is built, can be extensive.
- Featured items are checked vs Bloom filter on-the-fly

Advantages

- No DB hits for user
- Featured content is cacheable
- Will not to show content multiple times if read
- Tiny storage requirements even at scale
- Freshness can be achieved with zero/low human input
- Real-time recording of activity – immediate impact on fresh content

Visual



redislabs
home of redis

Activity Pattern Monitoring & Personalization

redislabs

Activity Pattern Monitoring & Personalization?

- Monitor the usage behaviour
  - Content viewed
  - Activity over time
  - Combinations of content history and activity
- Personalize the content based on the behaviour
- Seen as difficult to accomplish
  - Analytics data
    - Stored in another service
    - Anonymized
  - Complicated graph or ML based solutions
    - Inferences
    - Black boxes

Activity Pattern Monitoring & Personalization

- Record site activity with bit counting
- Unique page views in HyperLogLog
- Leverage the page visit Bloom filter
- Simpler counter for pages consumed
- Create criteria based on session stored analytics
  - New to a page? Bloom filter
  - New to the site? Unique Page view = 1 (HLL) && Previously Visited = false (Bloom)
  - Inactive user? Sum the bit count over the last five records, if = 0 then inactive
  - Been to a cluster of pages (infer interest)? Check cluster of pages vs Bloom filter – combo!

Activity Pattern Monitoring & Personalization

- Why is this suddenly possible?
  - Probabilistic data structures are small/fast
  - Bit counting is small/fast
  - Decoupled from operational database
- What about privacy?
  - Legitimate concern
  - Non-reversible probabilistic structures
  - Siloed from rest of database

Questions?

redislabs

Thank you!

Demo source code:
https://github.com/stockholmux/qcon-redis-session-store-demo

redislabs
home of redis