redislabs

Home of Redis

Redis for Fast Data Ingest

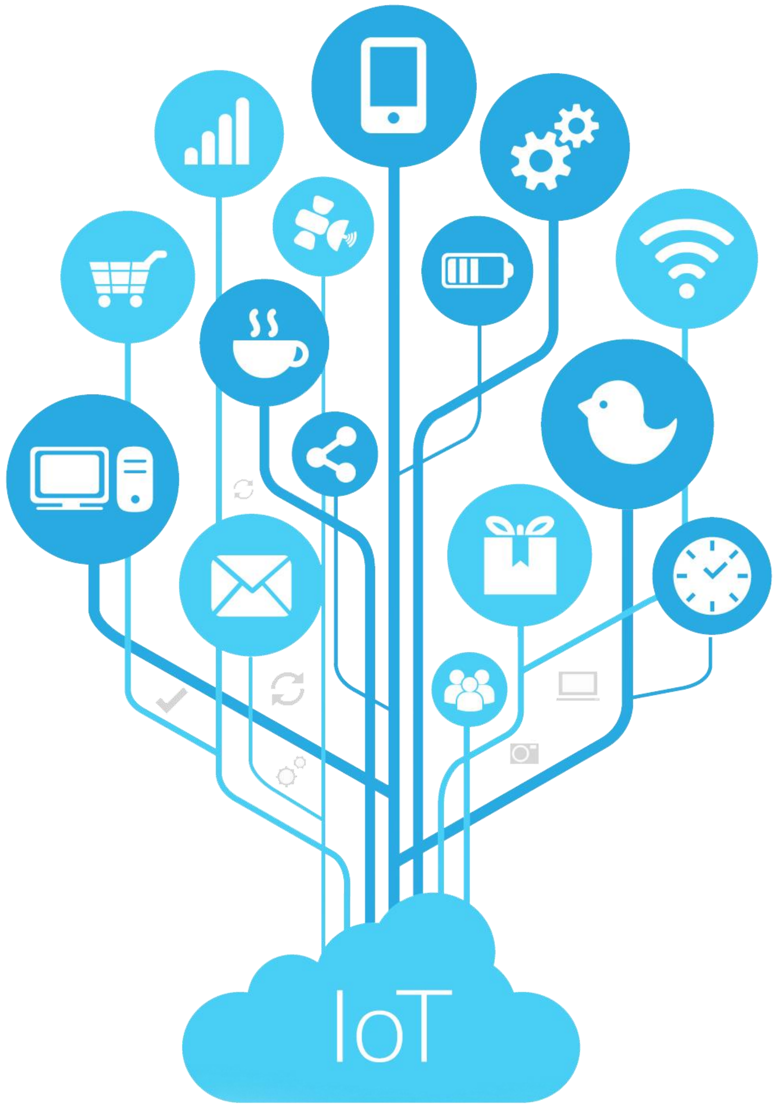
Agenda

- Fast Data Ingest and its challenges
- Redis for Fast Data Ingest
  - Pub/Sub
  - List
  - Sorted Sets as a Time Series Database
- The Demo
- Scaling with Redis^{e} Flash

redislabs

Fast Data Ingest Scenarios

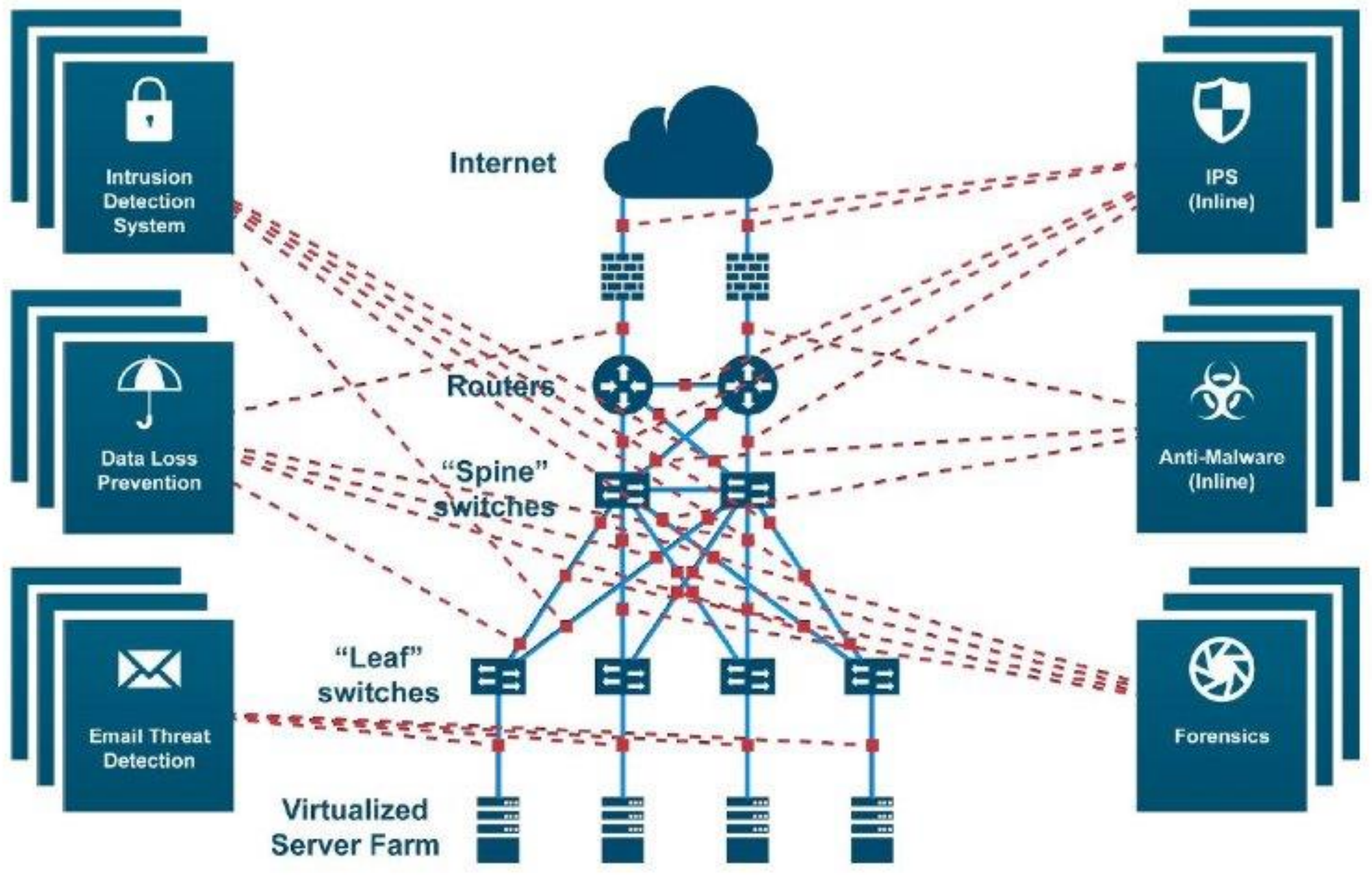
IOT



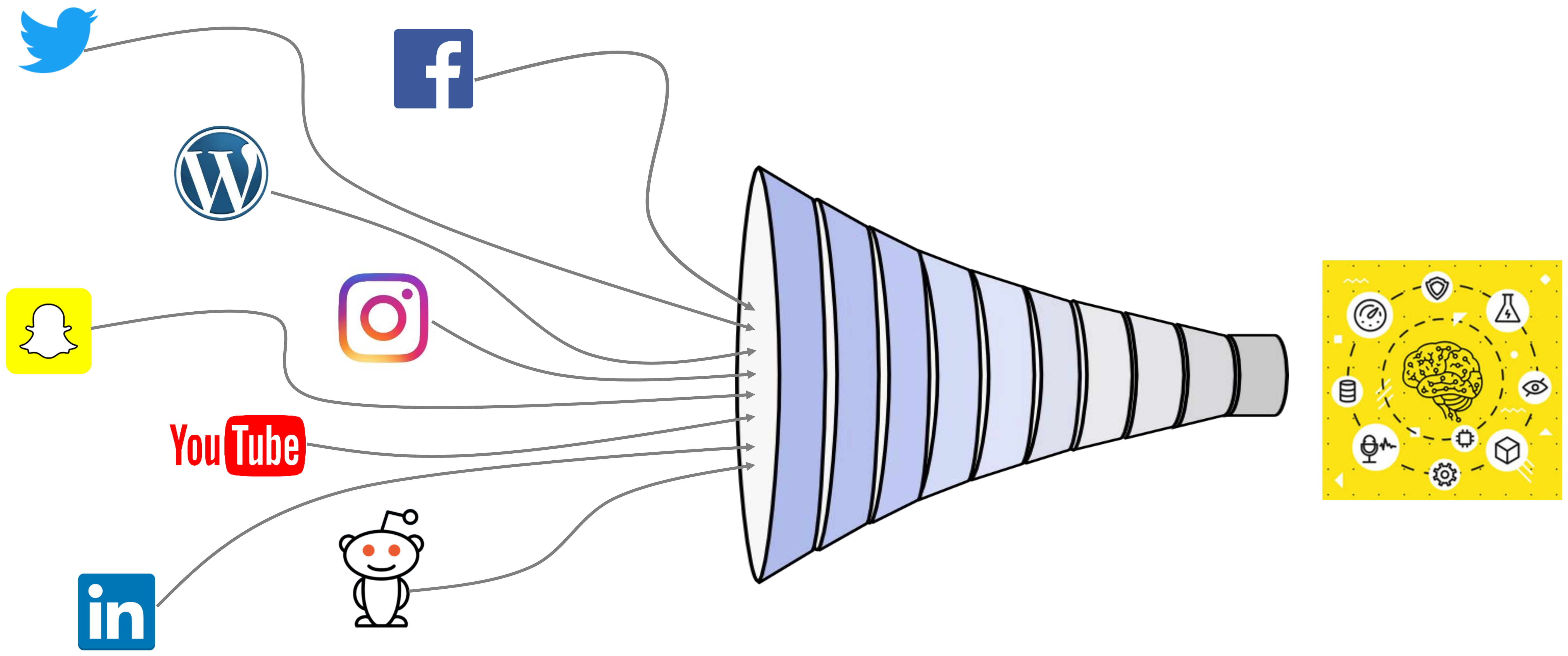
wiselife
Control your business utilities from your phone
Smart Plugs
Light & temperature sensors
Gas & Electricity Meters
Door & window sensors
Boiler
Radiators
Air Conditioning
Lighting (internal and external)

utilitywise

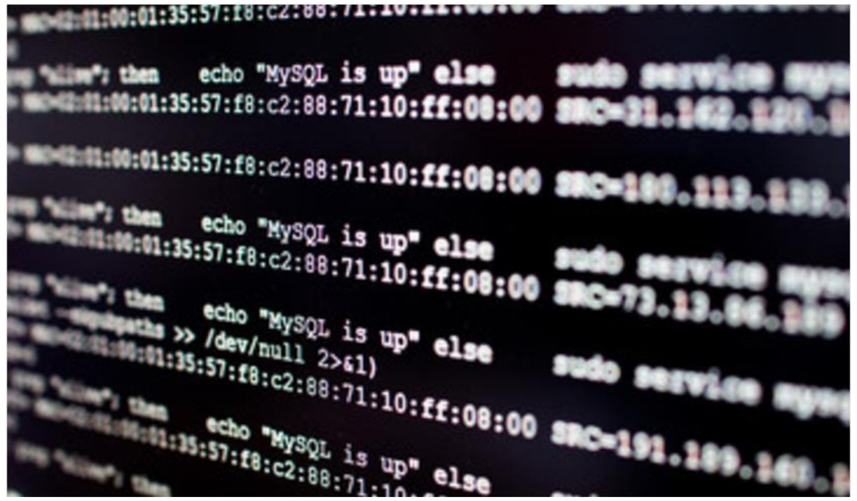
Network Traffic Inspection



Social Media Analysis



More Scenarios



```
:01:35:57:f8:c2:88:71:10:ff:08:00
"alive"; then    echo "MySQL is up" else     sudo service
:00:01:35:57:f8:c2:88:71:10:ff:08:00 SRC=
:00:01:35:57:f8:c2:88:71:10:ff:08:00 SRC=
"alive"; then    echo "MySQL is up" else     sudo service
:00:01:35:57:f8:c2:88:71:10:ff:08:00 SRC=73.13.
"alive"; then    echo "MySQL is up" else     sudo service
    >> /dev/null 2>&1)
:01:35:57:f8:c2:88:71:10:ff:08:00 SRC=191.
"alive"; then    echo "MySQL is up" else
:01:35:57:f8:c2:88:71:10
```

Log Collection

User Activity Tracking

Fintech

Multi-player Gaming

And more…

Fast Data Ingest Challenges

- Keeping up with the pace of data arrival
- Data from multiple sources with no standard data format
- Filter, analyze, and transform data in real-time
- Managing data arriving from sources distributed geographically

Requirements for Fast Data Ingest

- Physical infrastructure – network, computational resources, etc.
- Software stack to:
  - Filter
  - Aggregate
  - Transform
  - Distribute

data in real-time with sub-millisecond latency

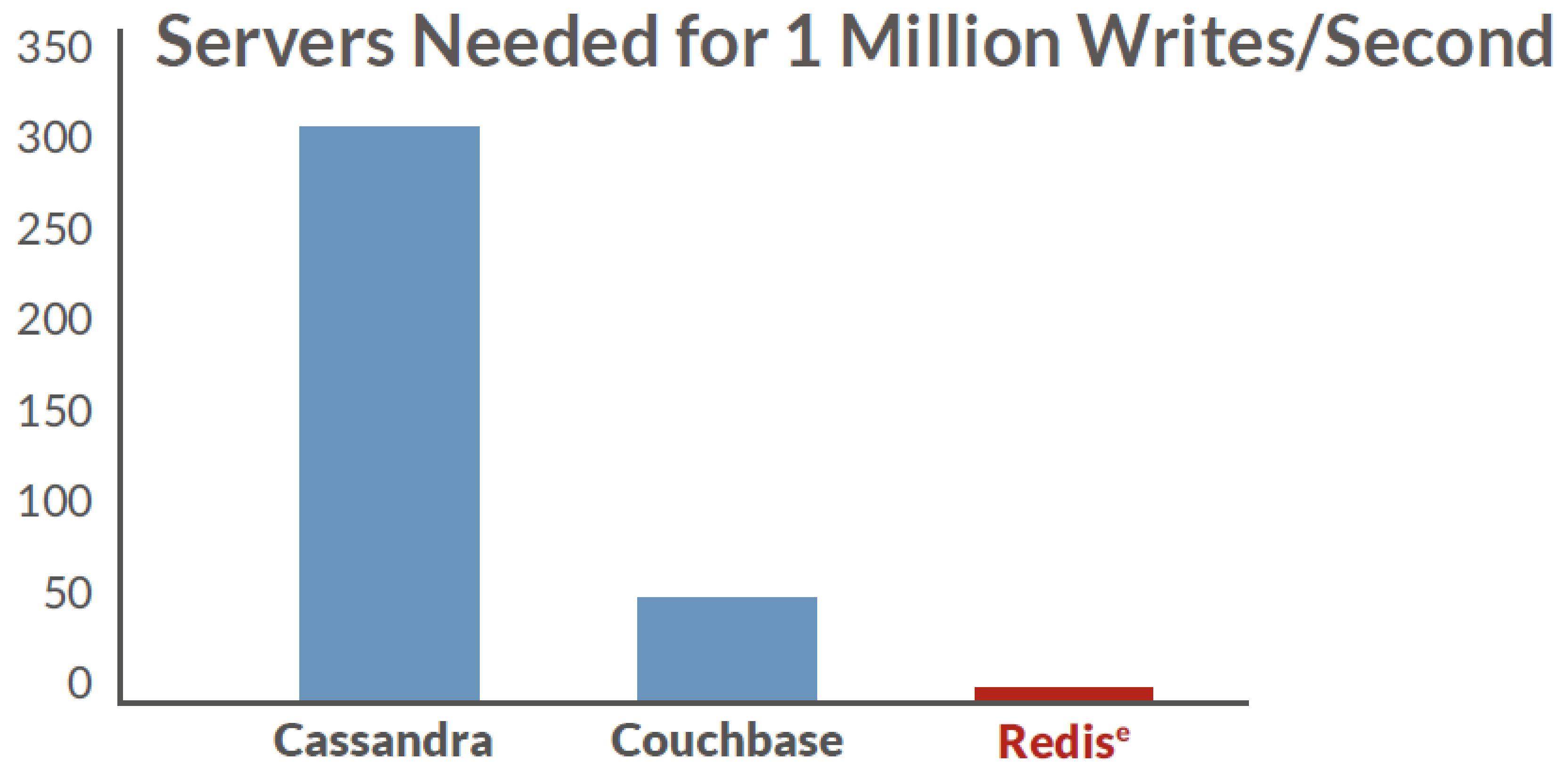
Fast Data Ingest with Redis

About Redis

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

The open source home and commercial provider of **Redis Enterprise (Redis^{e})** technology, platform, products & services.

Redis for Fast Data Ingest

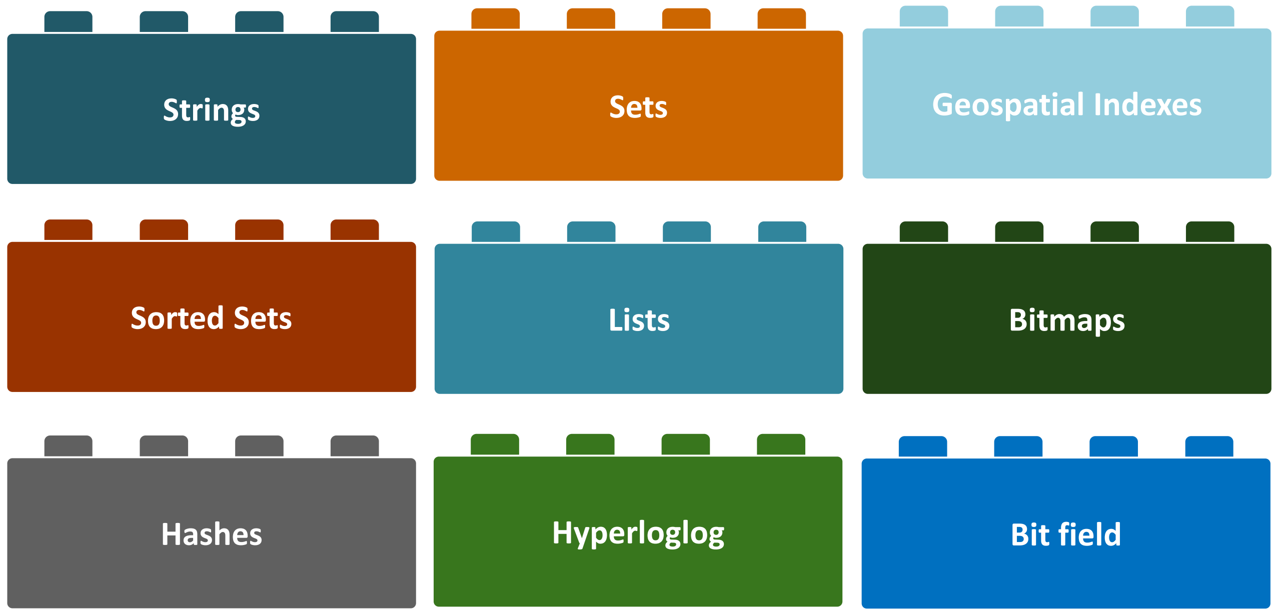


	Cassandra	Couchbase	Redis^e
ANNUAL COST	$2,226,216	$371,040	$14,832
COST COMPARED TO REDIS^e	150X	25X	

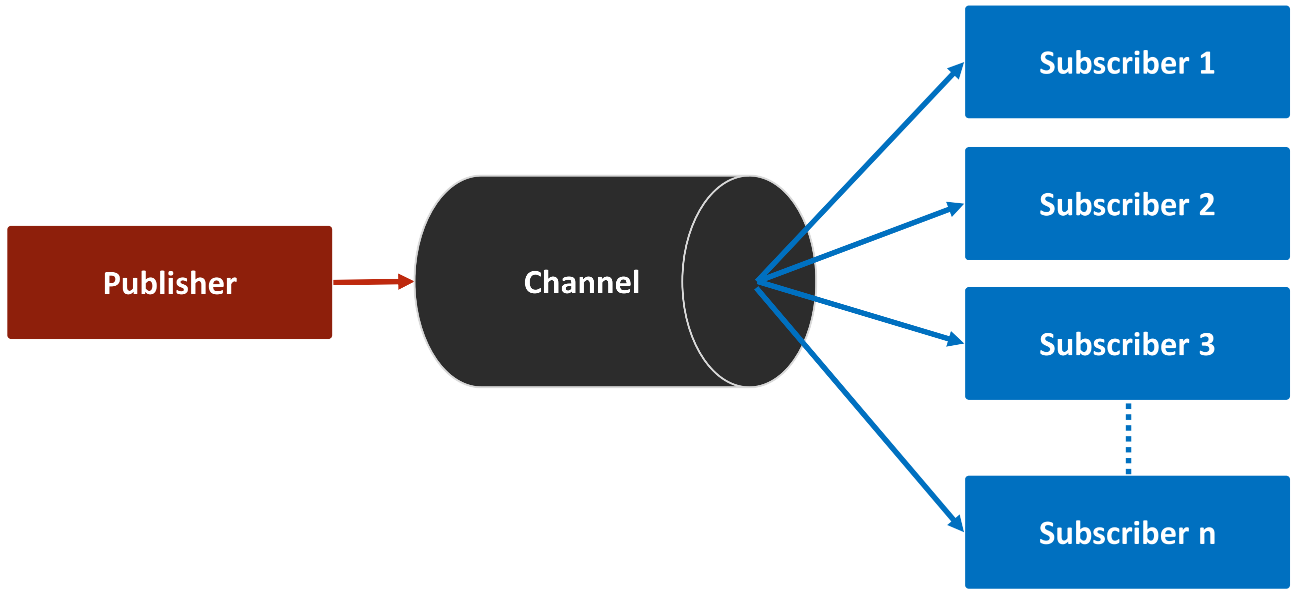
Redis^e delivers the maximum throughput with the lowest number of servers, slashing operational costs by up to 99% (Google Cloud Platform performance benchmarks)

redislabs

Redis for Fast Data Ingest



Redis Data Structures

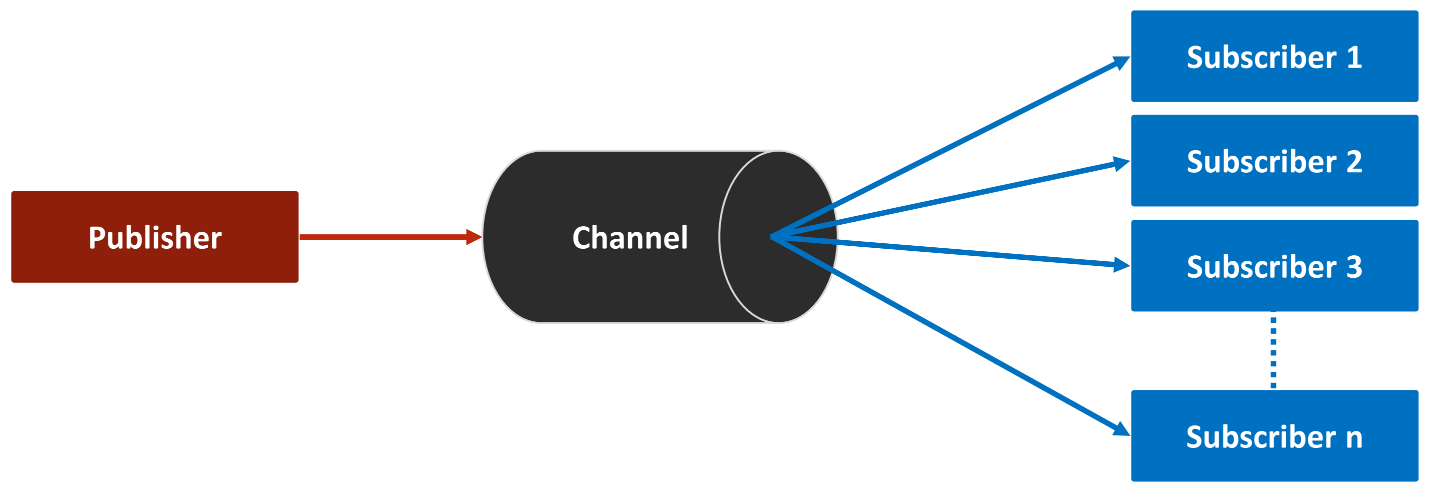


Redis Pub/Sub

redislabs

Common Ingest Techniques in Redis

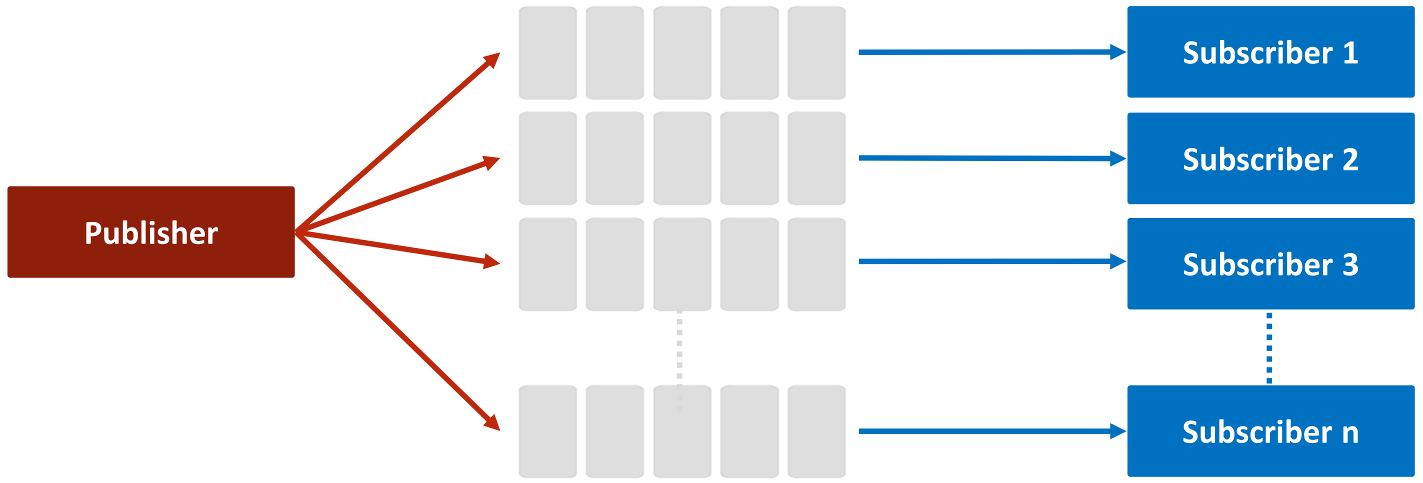
Pub/Sub



Commands

Publisher:	`publish <channel name> <message>`
Subscriber:	`subscribe <channel name>`

List

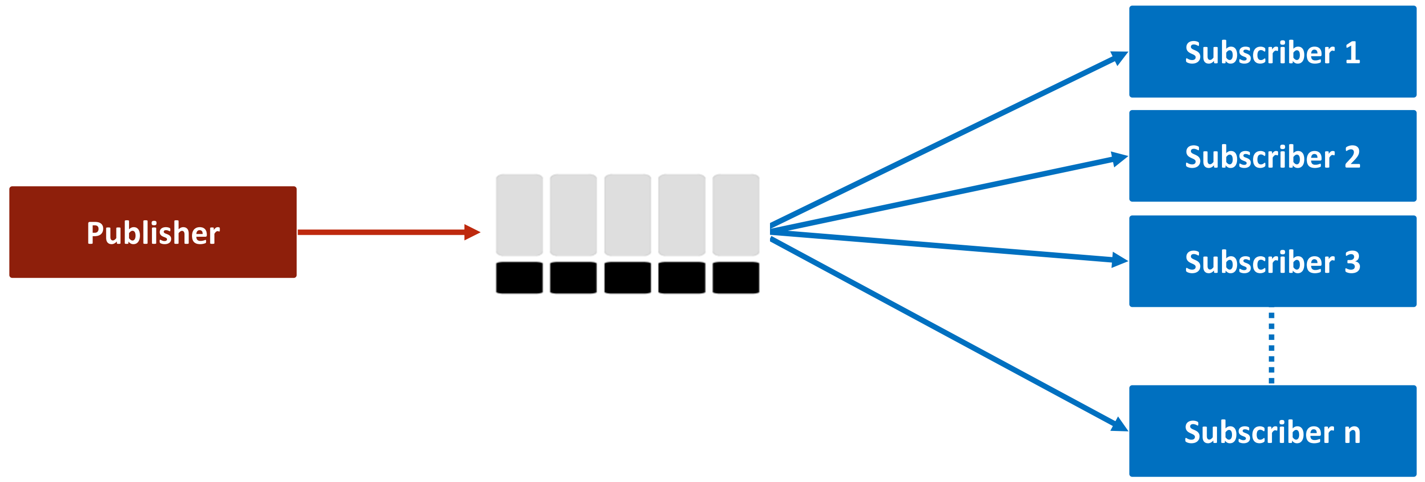


Commands

Publisher: `lpush <list name> <message>`

Subscriber: `brpop <list name> <timeout>`

Sorted Set



Commands

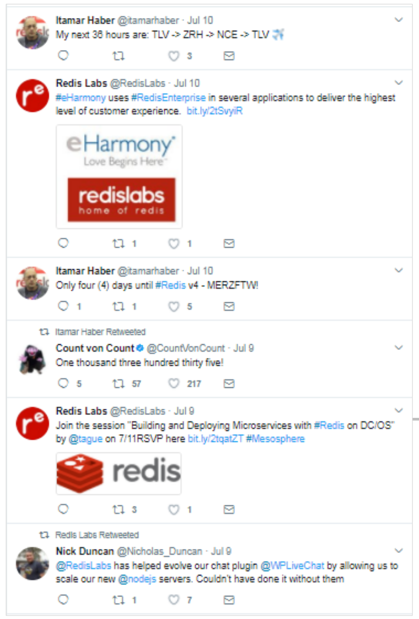
Publisher: `zadd <timeseries name> <timestamp> <message>`

Subscriber: `zrangebyscore <timeseries name> <last timestamp> <current timestamp> WITHSCORES`

redislabs

The Demo

Demo: Problem Description



Itamar Haber @itamarhaber · Jul 10
My next 36 hours are: TLV -> ZRH -> NCE -> TLV ✈
3

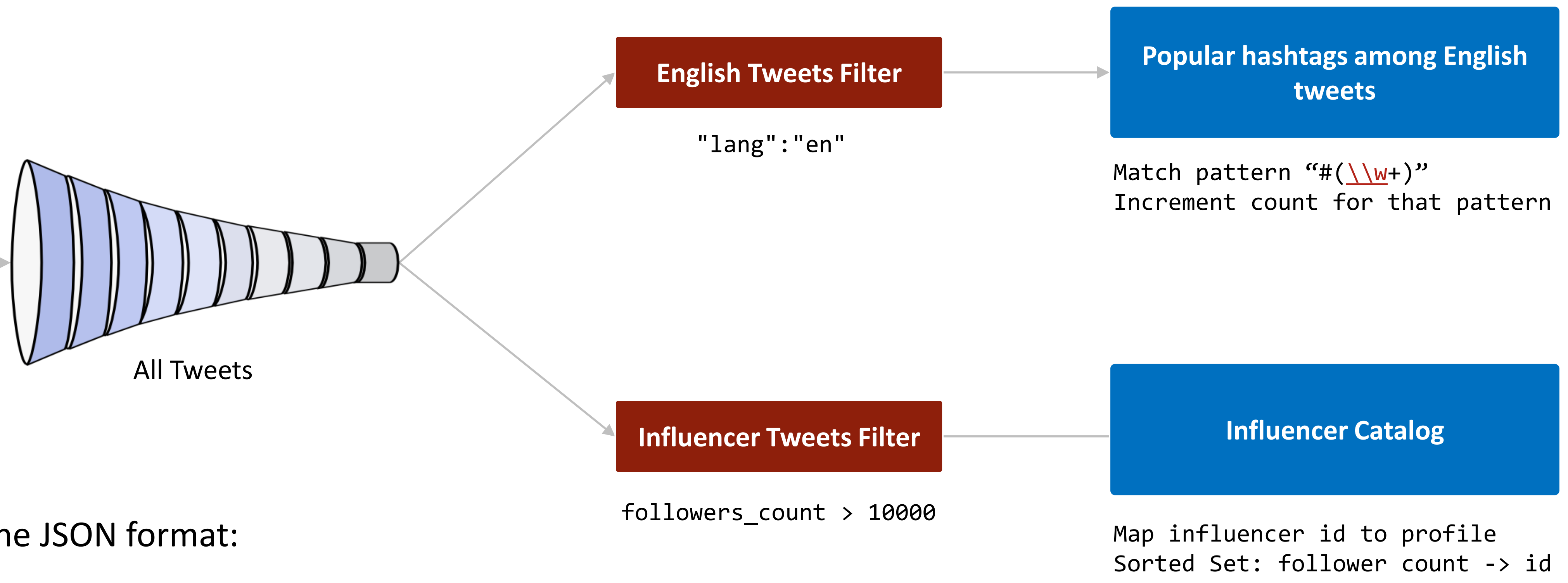
Redis Labs @RedisLabs · Jul 10
#eHarmony uses #RedisEnterprise in several applications to deliver the highest level of customer experience. bit.ly/2tSvyiR
1 1

Itamar Haber @itamarhaber · Jul 10
Only four (4) days until #Redis v4 - MERZFTW!
1 1 5

Itamar Haber Retweeted
Count von Count @CountVonCount · Jul 9
One thousand three hundred thirty five!
5 57 217

Redis Labs @RedisLabs · Jul 9
Join the session "Building and Deploying Microservices with #Redis on DC/OS" by @tague on 7/11RSVP here bit.ly/2tqatZT #Mesosphere
3 1

Redis Labs Retweeted
Nick Duncan @Nicholas_Duncan · Jul 9
@RedisLabs has helped evolve our chat plugin @WPLiveChat by allowing us to scale our new @nodejs servers. Couldn't have done it without them
1 7



Sample Tweet Message in the JSON format:

```
{
   "created_at":"Tue Jul 11 17:06:03 +0000 2017",
   "id":884821096440004600,
   "text":"USGS reports a M2 #earthquake 31km WSW of Enterprise, Utah on 7/11/17 @ 17:01:53 UTC https://t.co/xXQH2Mfy93 #quake",
   "user":{
      "id":1414684496,
      "name":"Every Earthquake",
      "screen_name":"everyEarthquake",
      "location":"Earth",
      "followers_count":18978,
      "friends_count":17,
      "lang":"en"
   }
}
```

Demo Setup

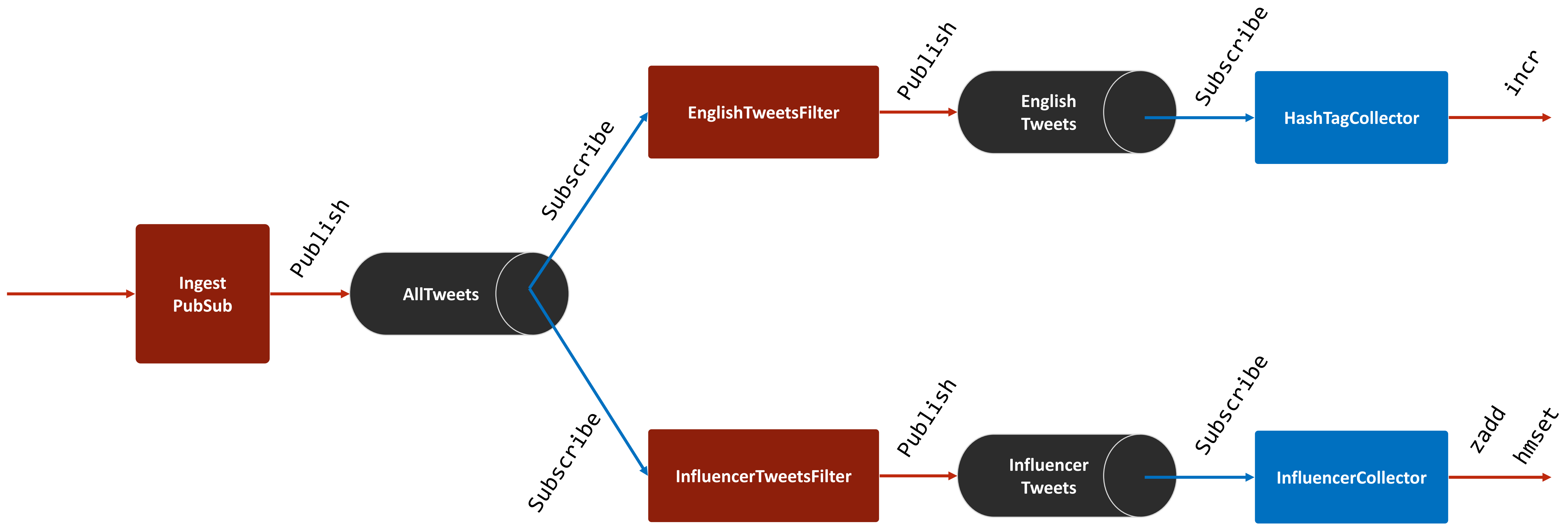
PubNub	Service Provider for Messages
Java	Programming Language for the demo
eclipse	IDE
docker	Redis container on Docker

The Three Data Ingest Techniques

Fast Data Ingest Technique	Pros	Cons
Pub/Sub	• Easy • Decoupled setup • Good for geographically distributed setup	• Not resilient to connection loss • Requires many connections
Lists	• Easy • Resilient to connection loss	• Tightly coupled producers and consumers • Data duplication
Sorted Sets	• Resilient to connection loss • Least chance of losing data • Access to historical data • Loosely coupled producers and consumers	• Consumes space • Complex logic

Technique 1: Fast Data Ingest with Pub/Sub

Fast Data Ingest with Pub/Sub

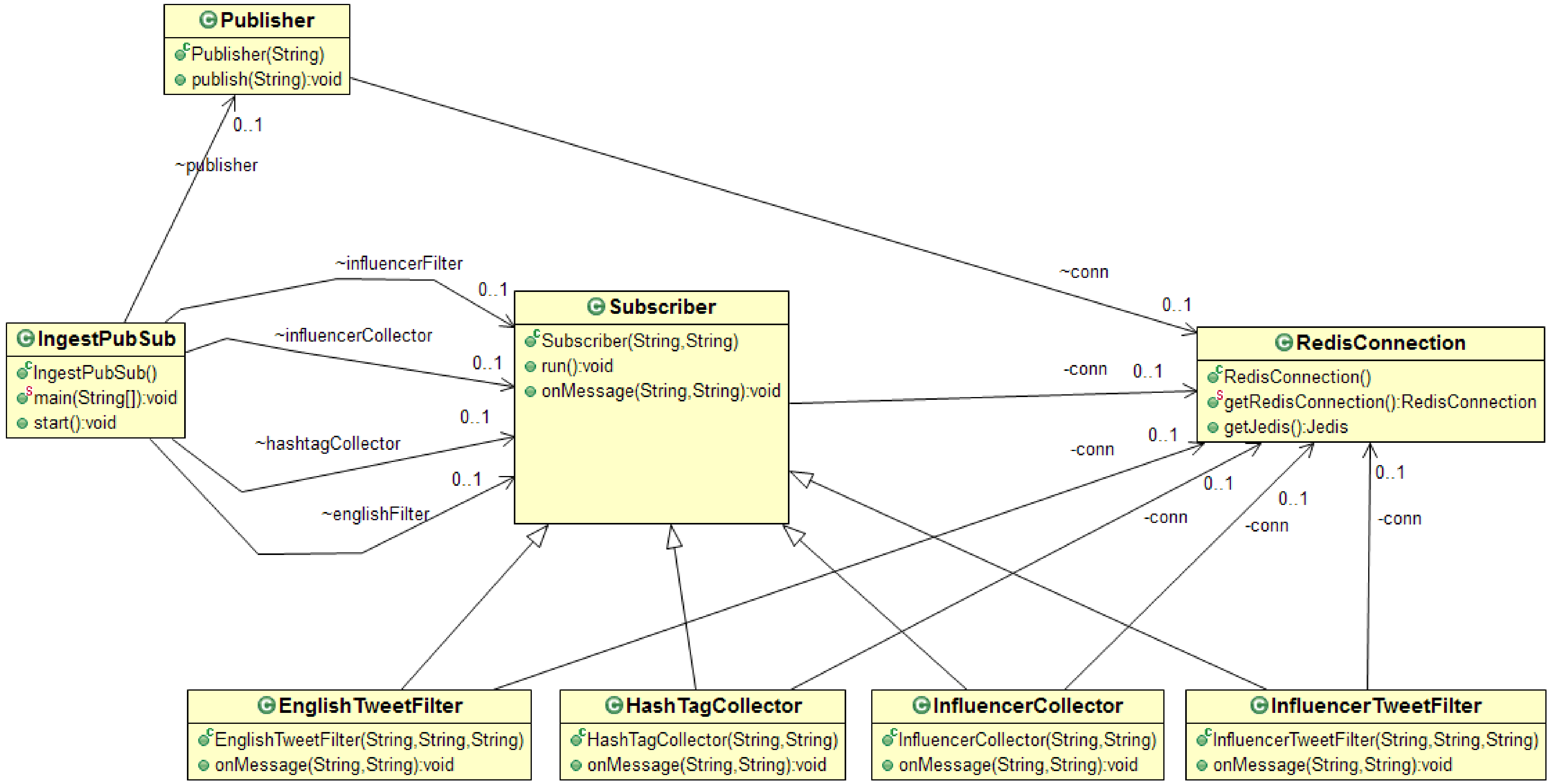


Advantages

- Easy
- Decoupled setup
- Good for geographically distributed setup

redislabs

Class Diagrams and Sample Code

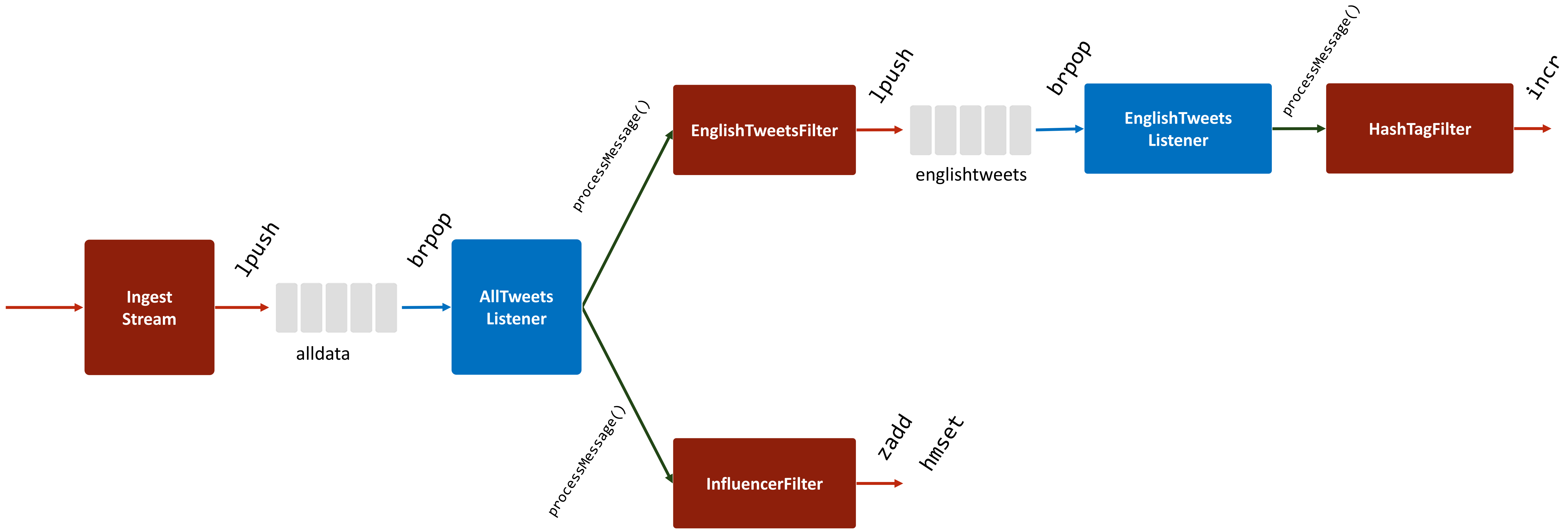


https://github.com/redislabsdemo/IngestPubSub

redislabs

Technique 2: Fast Data Ingest with Lists

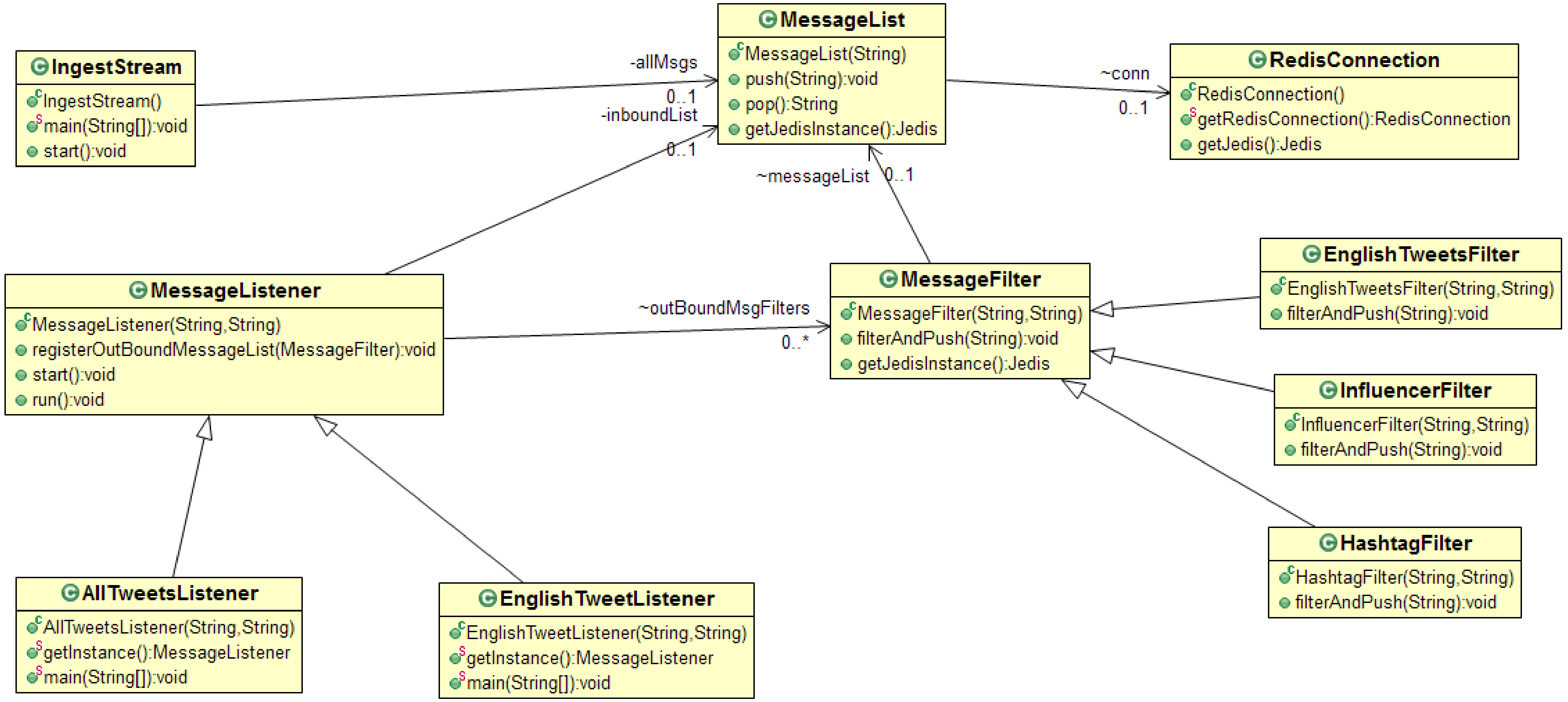
Fast Data Ingest with Lists



Advantages

- Easy
- Resilient to connection loss

Class Diagrams and Sample Code

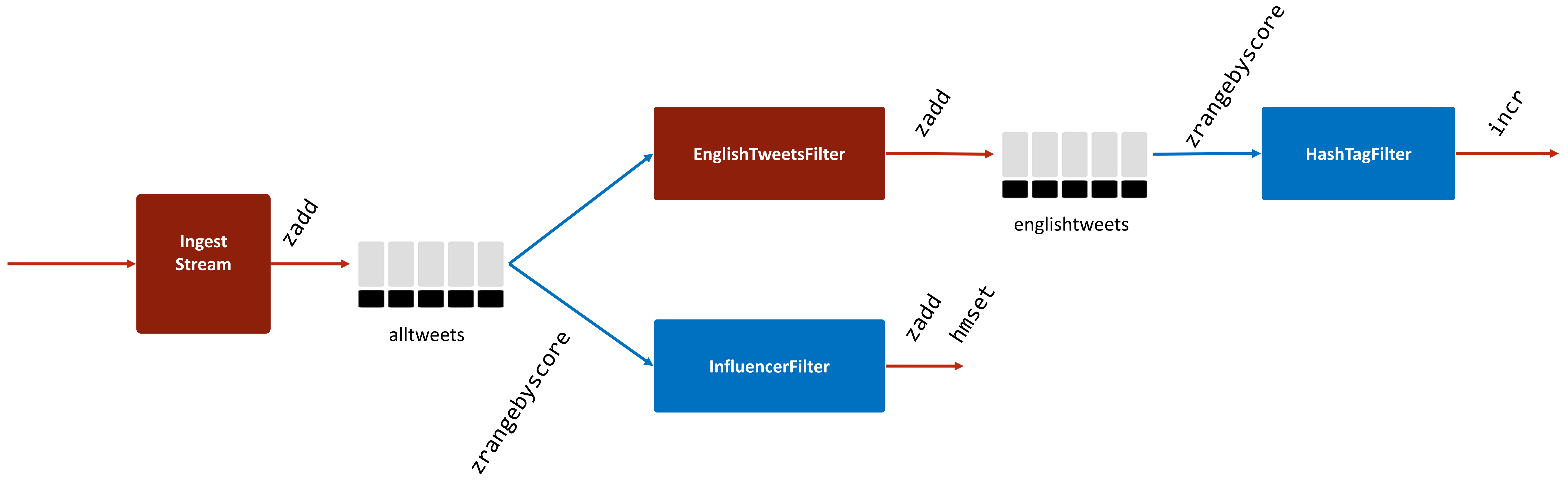

MessageList
MessageList(String)
push(String):void
pop():String
getJedisInstance():Jedis
RedisConnection
RedisConnection()
getRedisConnection():RedisConnection
getJedis():Jedis
IngestStream
IngestStream()
main(String[]):void
start():void
-allMsgs
0..1
-inboundList
0..1
~conn
0..1
~messageList
0..1
MessageListener
MessageListener(String,String)
registerOutBoundMessageList(MessageFilter):void
start():void
run():void
~outBoundMsgFilters
0..*
MessageFilter
MessageFilter(String,String)
filterAndPush(String):void
getJedisInstance():Jedis
EnglishTweetsFilter
EnglishTweetsFilter(String,String)
filterAndPush(String):void
InfluencerFilter
InfluencerFilter(String,String)
filterAndPush(String):void
HashtagFilter
HashtagFilter(String,String)
filterAndPush(String):void
AllTweetsListener
AllTweetsListener(String,String)
getInstance():MessageListener
main(String[]):void
EnglishTweetListener
EnglishTweetListener(String,String)
getInstance():MessageListener
main(String[]):void

https://github.com/redislabsdemo/IngestList

redislabs

Technique 3: Fast Data Ingest with Sorted Sets

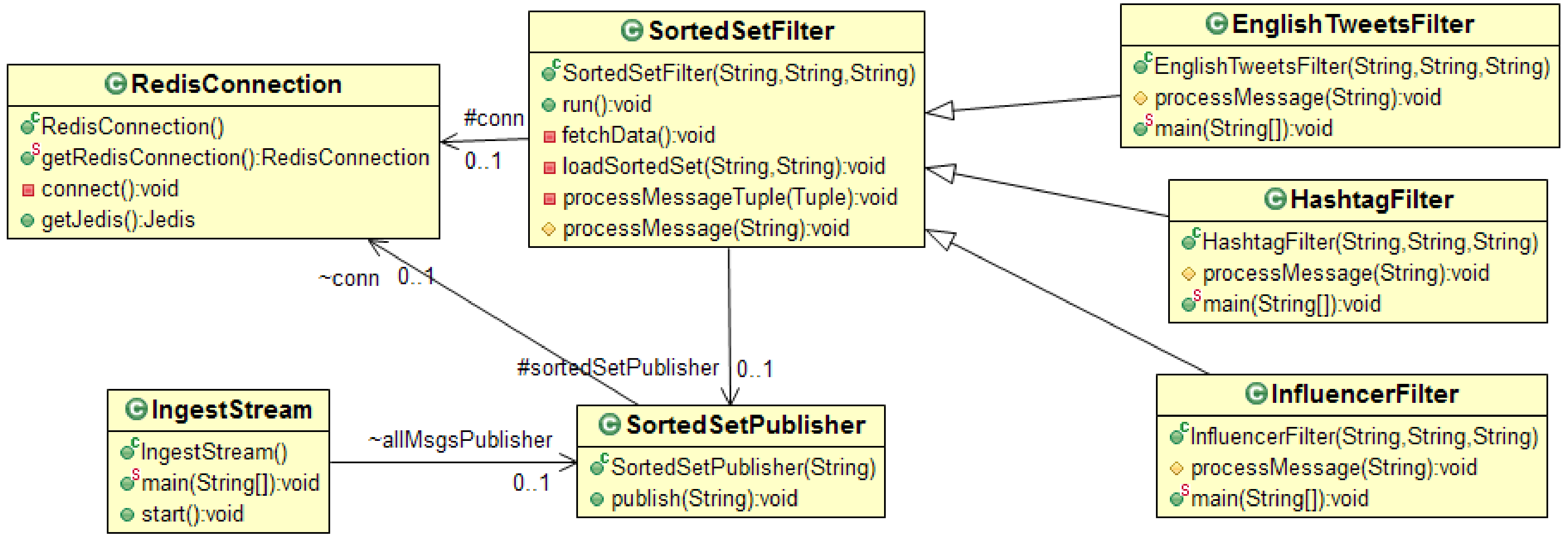
Fast Data Ingest with Sorted Sets



Advantages

- Resilient to connection loss
- Least chance of losing data
- Access to historical data
- Loosely coupled producers and consumers

Class Diagrams and Sample Code



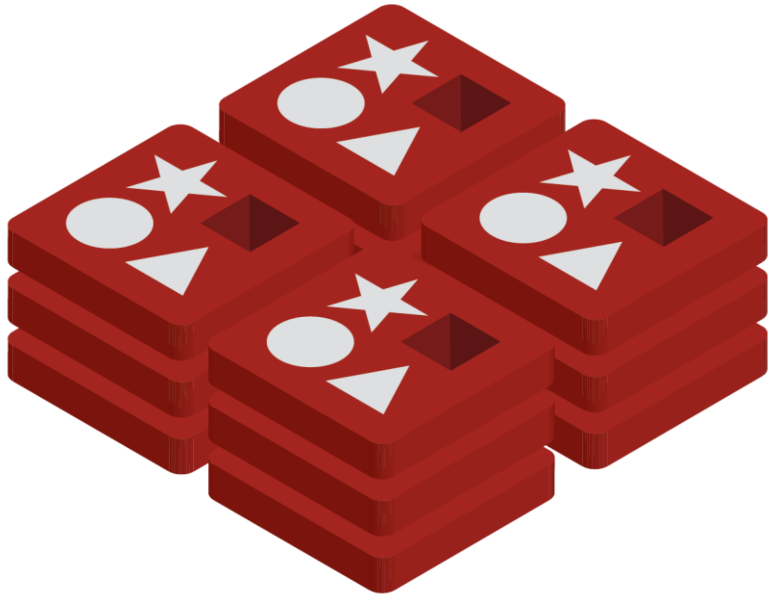
SortedSetFilter
SortedSetFilter(String,String,String)
run():void
fetchData():void
loadSortedSet(String,String):void
processMessageTuple(Tuple):void
processMessage(String):void
RedisConnection
RedisConnection()
getRedisConnection():RedisConnection
connect():void
getJedis():Jedis
EnglishTweetsFilter
EnglishTweetsFilter(String,String,String)
processMessage(String):void
main(String[]):void
HashtagFilter
HashtagFilter(String,String,String)
processMessage(String):void
main(String[]):void
InfluencerFilter
InfluencerFilter(String,String,String)
processMessage(String):void
main(String[]):void
IngestStream
IngestStream()
main(String[]):void
start():void
SortedSetPublisher
SortedSetPublisher(String)
publish(String):void
#conn
0..1
~conn
0..1
#sortedSetPublisher
0..1
~allMsgsPublisher
0..1

https://github.com/redislabsdemo/IngestSortedSet

Redis^{e} for Fast Data Ingest

Redis^e Technology

Redis Database Instances

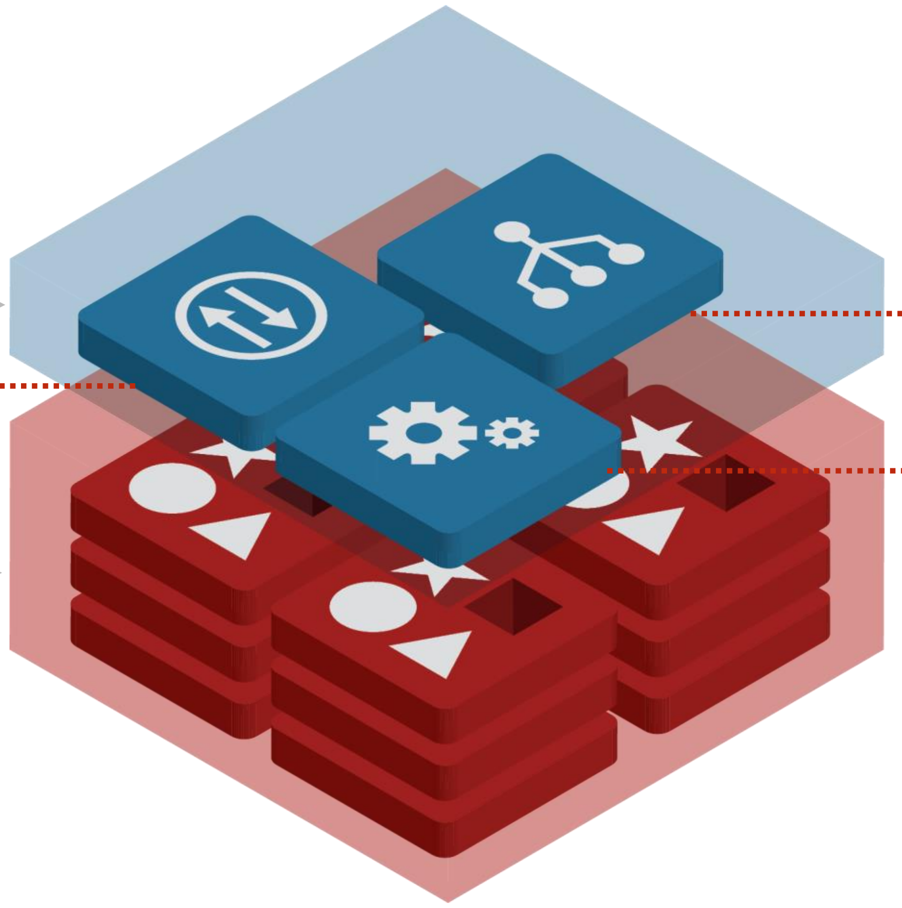


Redis^{e} Technology

Enterprise Layer

Zero latency proxy

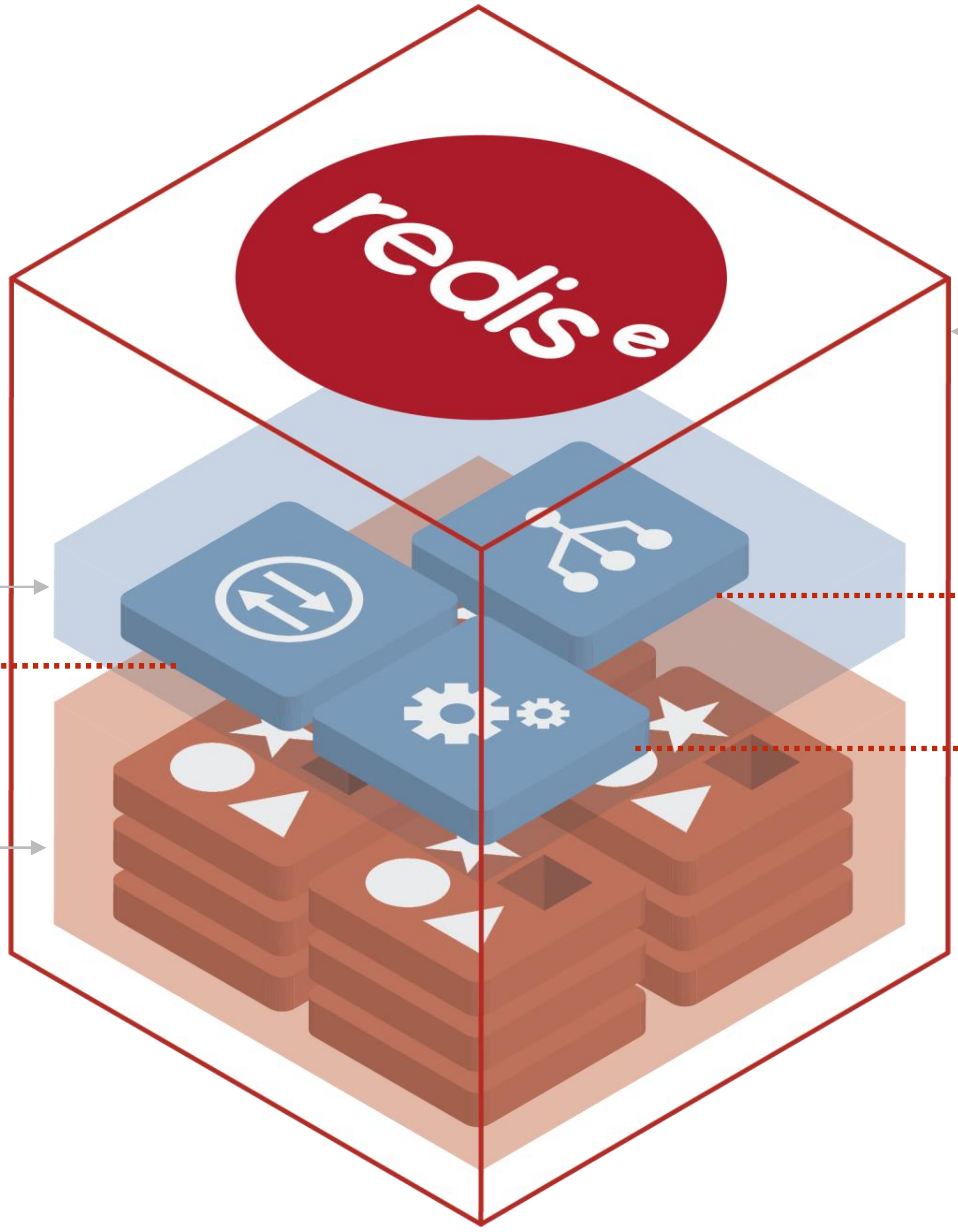
Open Source Layer



Cluster Manager

REST API

Redis^e Technology



Redis^e Node

Enterprise Layer

Cluster Manager

Zero latency proxy

REST API

Open Source Layer

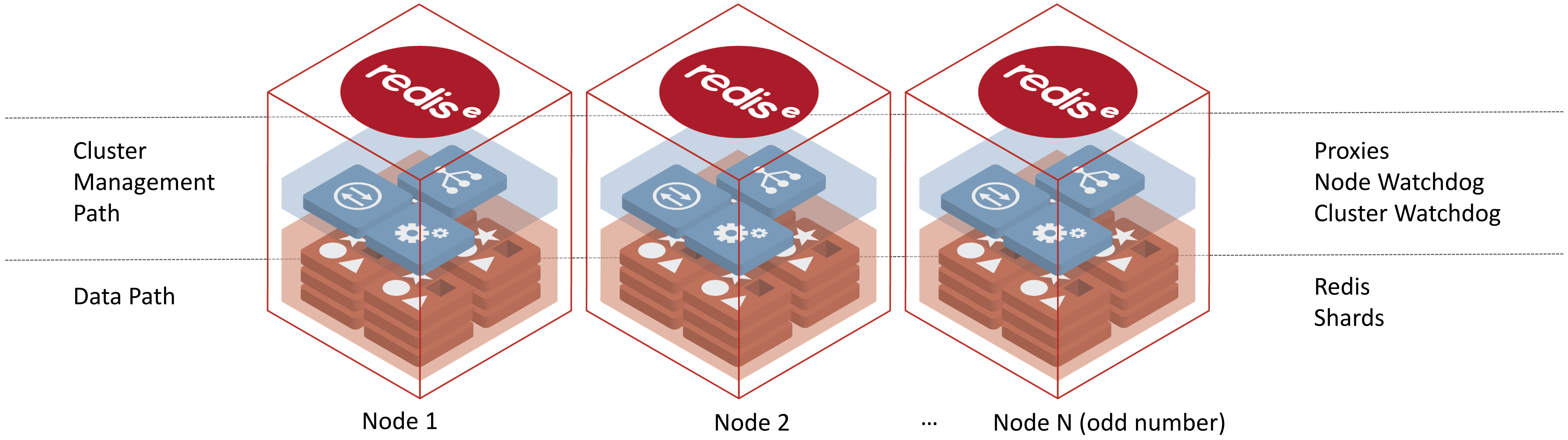
Redis^{e} Technology

Redis^{e} Cluster

- Shared nothing cluster architecture
- Fully compatible with open source commands & data structures

Redis^{e} - Shared Nothing Symmetric Architecture

Distributed Proxies
Single or Multiple Endpoints



Unique multi-tenant “Docker” like architecture enables running hundreds of databases over a single, average cloud instance without performance degradation and with maximum security provisions

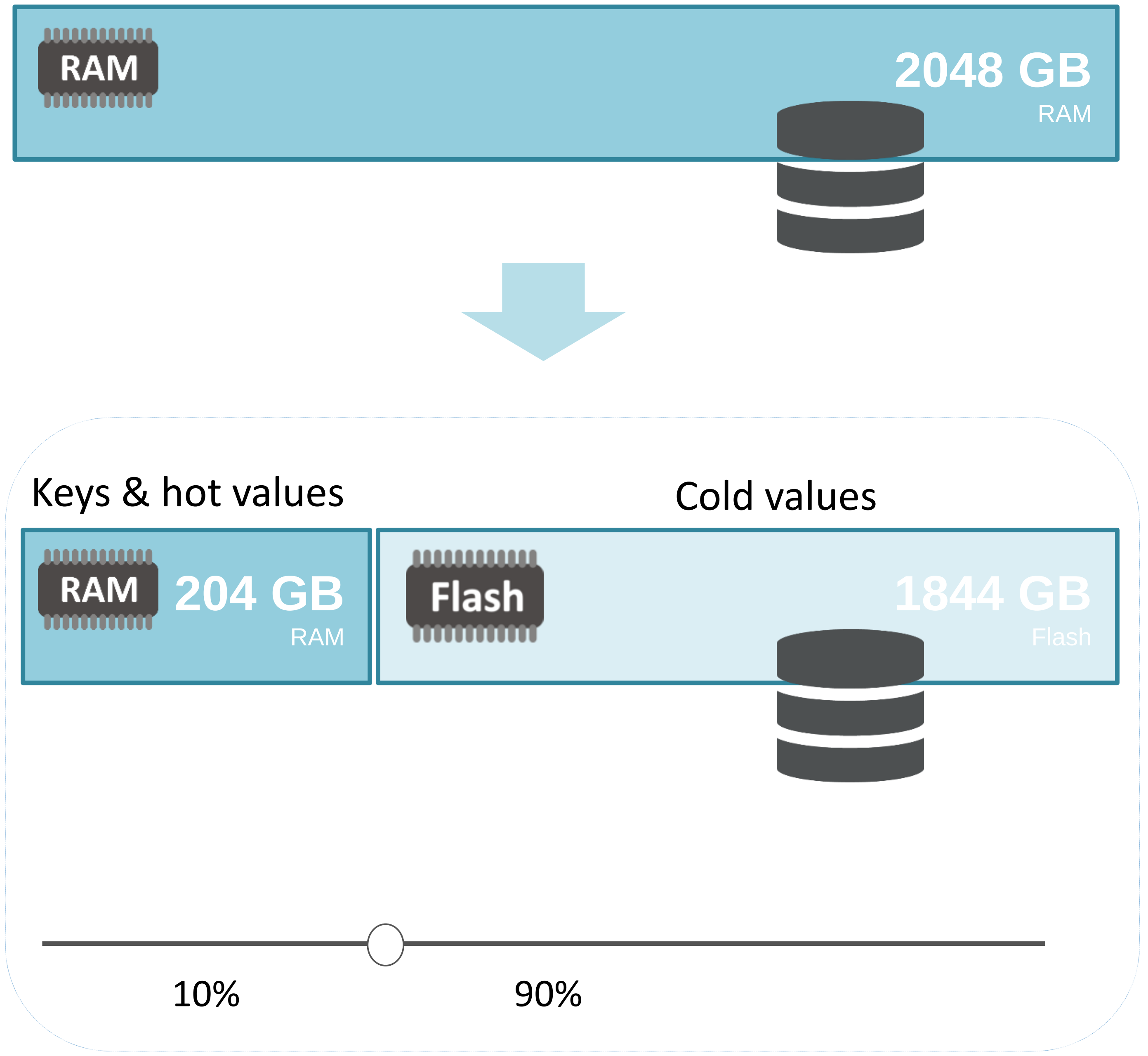
Redis^e Benefits for Data Ingest

Effortless Scaling	Always On Availability	Substantially Lower Costs
Simple, Seamless Clustering. Linear scaling	Instant Failure Recovery, No Data loss	Run on Flash as a RAM extension
ACID Compliance in Cluster Architecture	Stable and Predictable High Performance	Top notch 24x7 expert support

redislabs

Redis^e Flash

- Near-RAM performance at 70%+ lower costs
- Technology treats Flash as a RAM replacement (or extension)
- RAM/Flash ratio can be easily configured
- Pluggable storage engine
- Available on SATA-based SSD, NVMe-based SSD, NVDIMM like 3D XPoint/SCM on x86 and P8 platforms



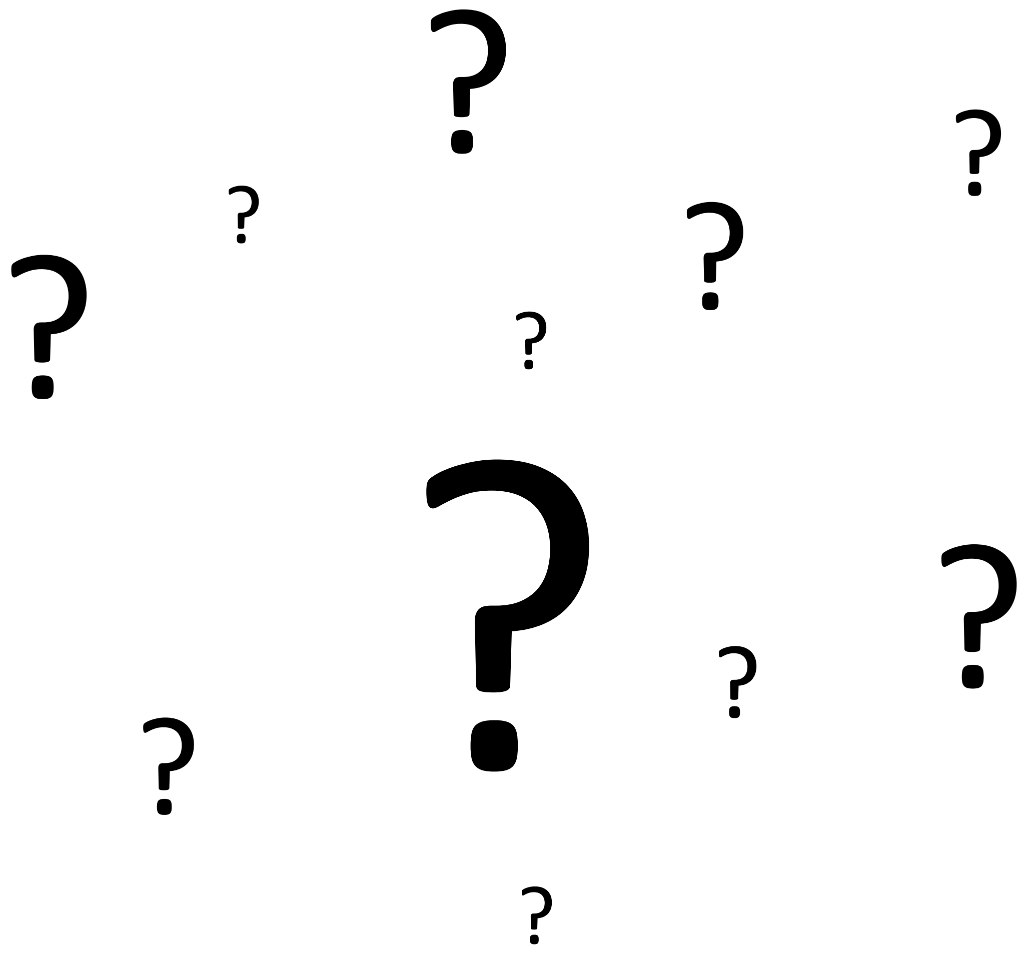
Redis^e Flash - 10TB Redis Deployment on EC2

	Redis on RAM	**Redis^e Flash**
Dataset size	10 TB	10 TB
Database size with replication	30 TB	20 TB*
AWS instance type	x1.32xlarge	i3.16xlarge
Actual instance size (RAM, and RAM+Flash)	1.46 TB	3.66 TB
# of instances needed	21	6 + 1 (for quorum)
Persistent Storage (EBS)	154 TB	110 TB
1 year cost (reserved instances)	$1,595,643	$298,896
Savings	-	**81.27%**

** Redis Enterprise only needs 1 copy of the data because quorum issues are solved at the node level*

redislabs

Questions



One more thing….

redis.conf setting:

```
client-output-buffer-limit pubsub 32mb 8mb 60
```

With this setting, Redis will force the clients to disconnect under two situations:

- If the output buffer grows more than 32mb
- If the output buffer holds 8mb of data consistently for 60 seconds

Thank You

Roshan Kumar

✉ roshan@redislabs.com
@roshankumar

Redis Labs

✉ expert@redislabs.com
@redislabs