

# MADOCA II Data Logging System Using NoSQL Database for SPring-8

**A.Yamashita and M.Kago**  
**SPring-8/JASRI, Japan**

# NoSQL

# **OR: How I Learned to Stop Worrying and Love Cassandra**

# Outline

- **SPring-8 logging database**
- **Why NoSQL, why Cassandra**
- **Implementation**
- **Production Run**

# SPRING-8 Logging database

- **Relational database system (RDBMS) has been used since 1997**
- **Grows from 871 signals to 27,626 signals (end of 2014)**
- **7,000 signal inserts per seconds**
- **4TB raw data at end of 2014**
- **SACLA ( X-ray FEL) is also using it**

# SPring-8 Logging database

- **What made the system live long?**
  - **Uniform data store**
  - **Simple access**

# SPring-8 Logging database

- **What made the system live long?**
  - **Uniform data store**
    - **Every data**
    - **Every time**
    - **in one database**
  - **Simple access**

# SPring-8 Logging database

- **What made the system live long?**
  - **Uniform data store**
  - **Simple access**
    - **Just Key + time range access**  
`get ("sr_mag_ps_b/current_adc",  
"2014/10/10 19:24:12",  
"2014/10/10 22:00:00")`



# RDBMS to NoSQL

- **For the next generation SPring-8-II**
- **We changed logging database for SPring-8 from RDBMS to a NoSQL database; Cassandra**
- **Why NoSQL, Why Cassandra ?**

# RDBMS is great

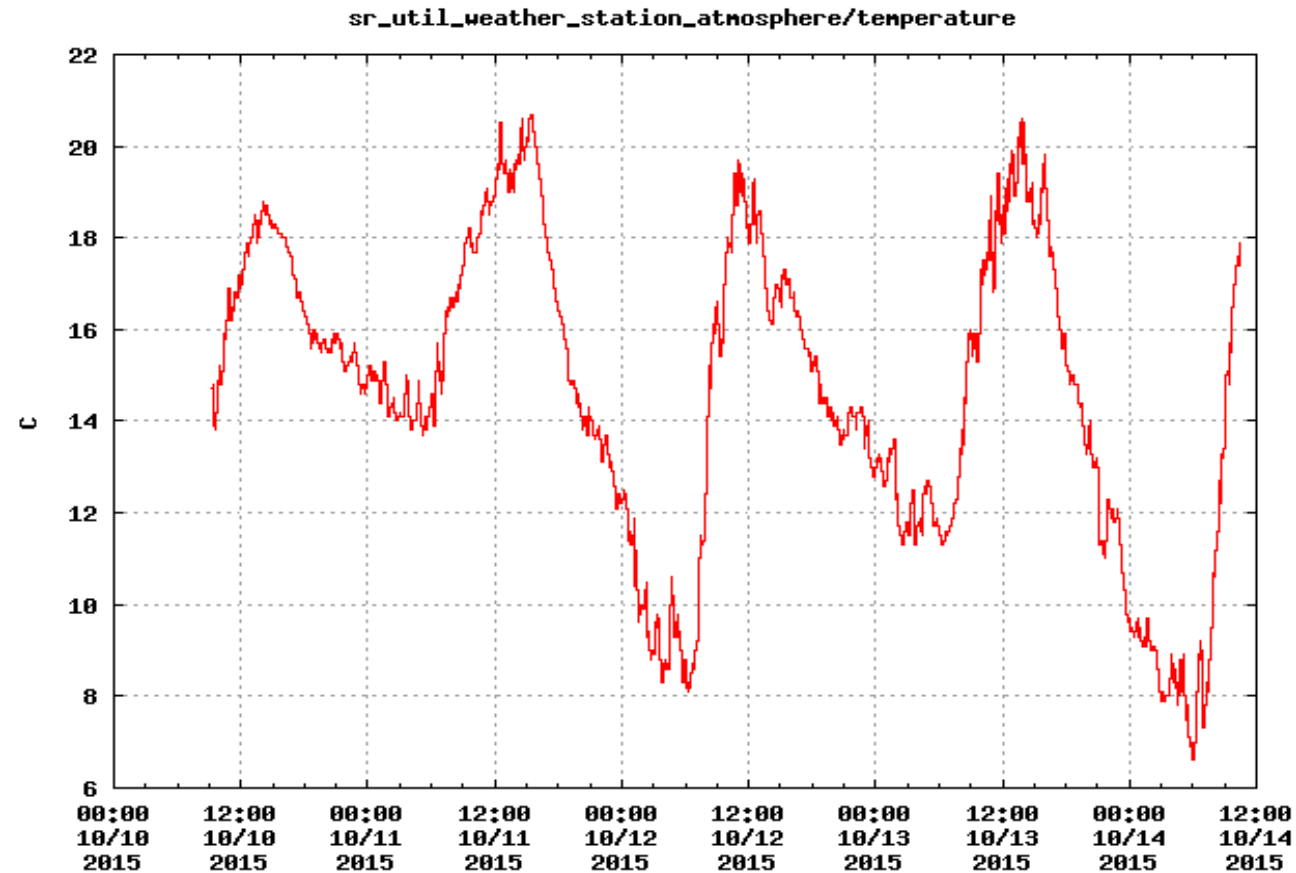
- **We are currently using RDBMS for**
  - **Configuration management**
  - **Parameter management**
  - **Alarm record**
  - **Etc**
- **But,**

# RDBMS limitation in logging

- **Performance**
- **Scalability**
- **Availability**
- **Flexibility**

# Logging in Accelerator Control

- Time series data



# Logging in Accelerator Control

- **Time series data**
- **Write many and rare read**

# Logging in Accelerator Control

- **Time series data**
- **Write many and rare read**
- **Is RDBMS is suitable for this task?**

# Logging in Accelerator Control

- **Time series data**
- **Write many and rare read**
- **Is RDBMS is suitable for this task?**
- **Looking for new database**
  - **Keeping advantage of the old system**
  - **Make up for its shortcomings**

# NoSQL (Not only SQL)

- **Simplicity of design, simpler "horizontal" scaling to clusters of machines, which is a problem for relational databases, and finer control over availability. (Wikipedia)**



# NoSQL (Not only SQL)

- **Simplicity of design, simpler "horizontal" scaling to clusters of machines, which is a problem for relational databases, and finer control over availability. (Wikipedia)**

# NoSQL variations

- **Key-value**
- **Graph**
- **Document**
- **No solutions for time-series data in above NoSQL**
- **Wide-column**

# Wide-column database

- **One type of NoSQL (Not only database)**

**Row Key**

sig1:20130504

sig2:20130504

sig3:20130504

**Column key**

**Column value**

# Wide-column database

- **Columns are added when data added.**

Row Key	Column
sig1:20130504	t0
	value0
sig2:20130504	
sig3:20130504	

**Column key**

**Column value**

# Wide-column database

- Columns are added when data added

Row Key	Column	
sig1:20130504	t0	t1
	value0	value1
sig2:20130504	T'0	
	Value'0	
sig3:20130504	T"0	
	Value"0	

**Column key**

**Column value**

# Wide-column database

- Row is added at any time

Row Key	Column		
sig1:20130504	t0	t1	t2
	value0	value1	value2
sig2:20130504	T'0	T'1	
	Value'0	Value'1	
sig3:20130504	T''0		
	Value''0		
sig4:20130510			

**Column key**

**Column value**

# Wide-column database

- **And it grows**

Row Key	Column			
sig1:20130504	t0	t1	t2	t3
	value0	value1	value2	value3
sig2:20130504	T'0	T'1	T'2	
	Value'0	Value'1	Value'2	
sig3:20130504	T''0			
	Value''0			
sig4:20130510	T'''0			
	Value'''0			

**Column key**

**Column value**

# Wide-column database

- **Suitable for time-series data logging**
  - **Each data has its own time-stamp**
    - **cyclic data + event driven data in same place**
  - **Access**
    - **Key+ column range**
      - **same as current access method**



# Which Wide-column DB?

- **Major wide-column database**
  - **Apache Cassandra**
  - **Apache Hbase**
  - **Hypertable**

# Apache Cassandra

- **We select by its availability**
- **Every node has the same role**
  - **No master node**
    - **No single point of failure (SPOF)**
    - **HBase and Hypertable have masternode: SPOF**



# Apache Cassandra

- **Our criteria**
  - **Reliability**
  - **Scalability**
  - **Flexibility**
- **Consistency is covered by the other DB**

# Reliability

- **Most essential**

# Reliability

- **Most essential**
- **Cassandra**
  - **No master node, no single point of failure**
  - **Data redundancy**
    - **3 data replicas**

# Scalability

- **Just add nodes when you need more power**
  - **No cluster reboot is needed**
- **Apple is operating 100,000 node cluster for iTunes**

# Flexibility

- **Insert at any time**

# Flexibility

- **Insert at any time**
  - **Signal by signal**



# Flexibility

- **Insert at any time**
- **Data type**

# Flexibility

- **Insert at any time**
- **Data type**
  - **Store data using object serialization**
    - **Not using cassandra's data type**
    - **blob type column only**

# Flexibility

- **Insert at any time**
- **Data type**
  - **Store data using object serialization:MessagePack**
    - **Very fast**
    - **Low overhead 8 Byte float -> 9 Byte string**
    - **Self described**
      - **NO Interface Definition Language like Protocol Buffer**

# Consistency

- **Cassandra does not guarantee consistency**
- **In our cluster, it takes about 1 second after insert to obtain consistent value.**
  - **No real-time access**

# Redis

- **Covers Cassandra's inconsistency**

# Redis

- **Covers Cassandra's inconsistency**
- **Stores newest data only**

# Redis

- **Covers Cassandra's inconsistency**
- **Stores newest data only**
- **In-memory key-value database**

# Redis

- **Covers Cassandra's inconsistency**
- **Stores newest data only**
- **In-memory key-value database**
  - **Very fast by key access**
  - **Newest value+ meta data only**
  - **Data packed by MessagePack**



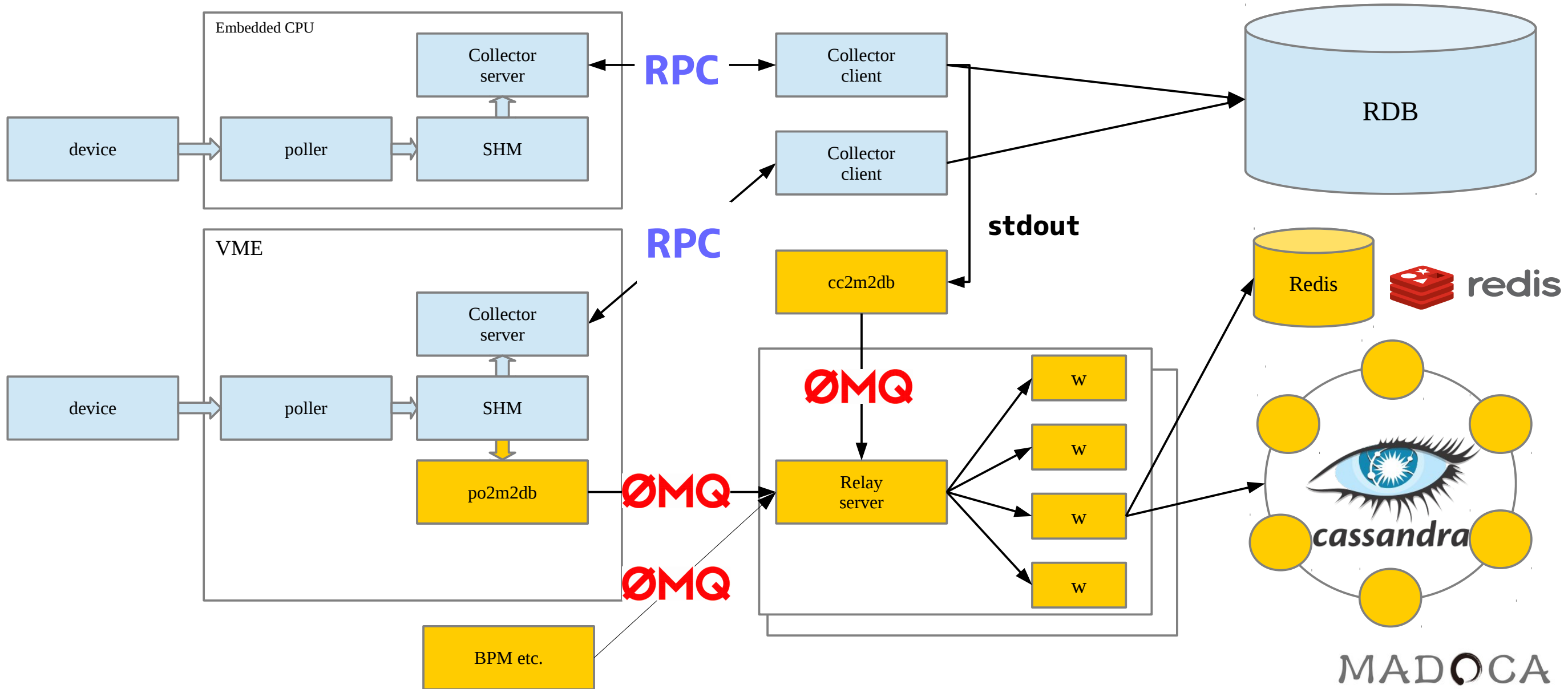
# Redis

- **Covers Cassandra's inconsistency**
- **Stores newest data only**
- **In-memory key-value database**
- **Two redis servers are running in parallel for redundancy**

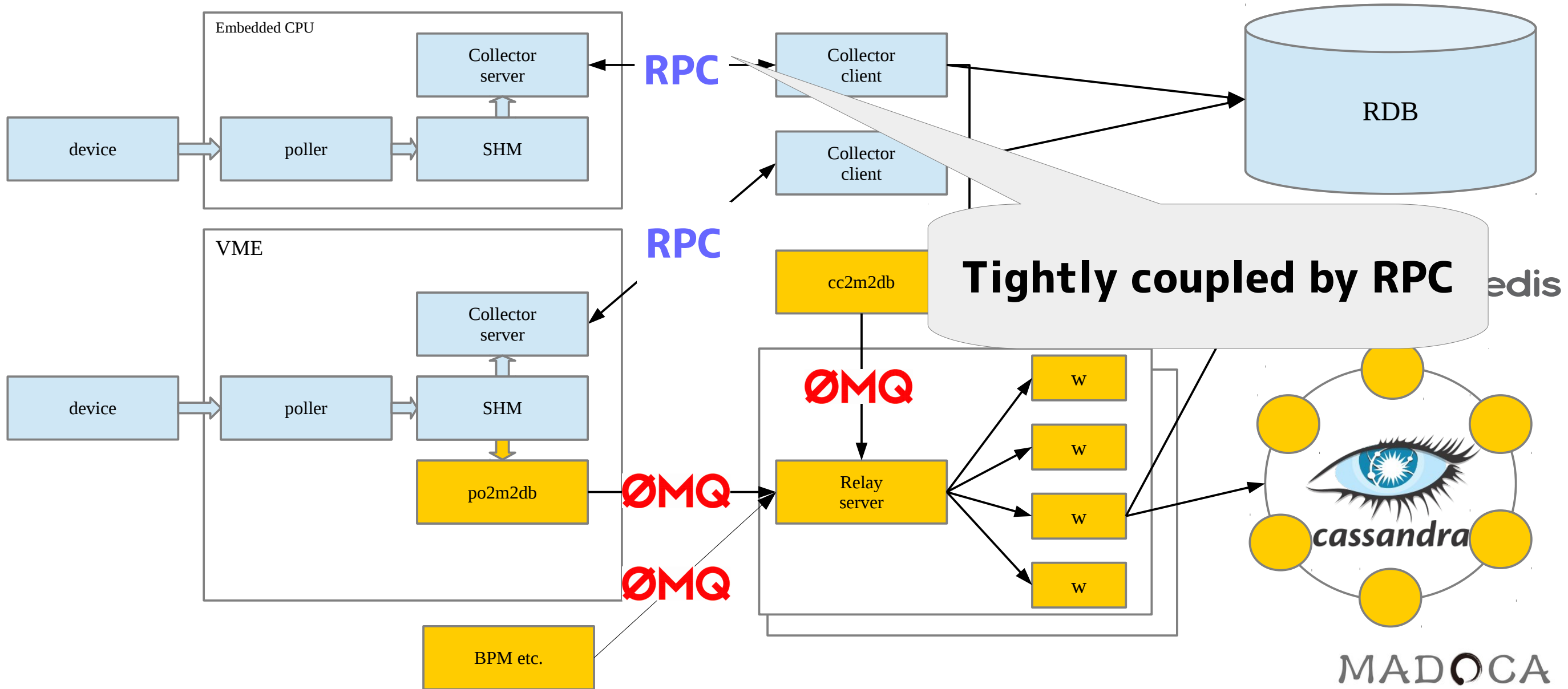
# Implementation

- **Data acquisition system**
- **Cassandra structure**
- **Performance**

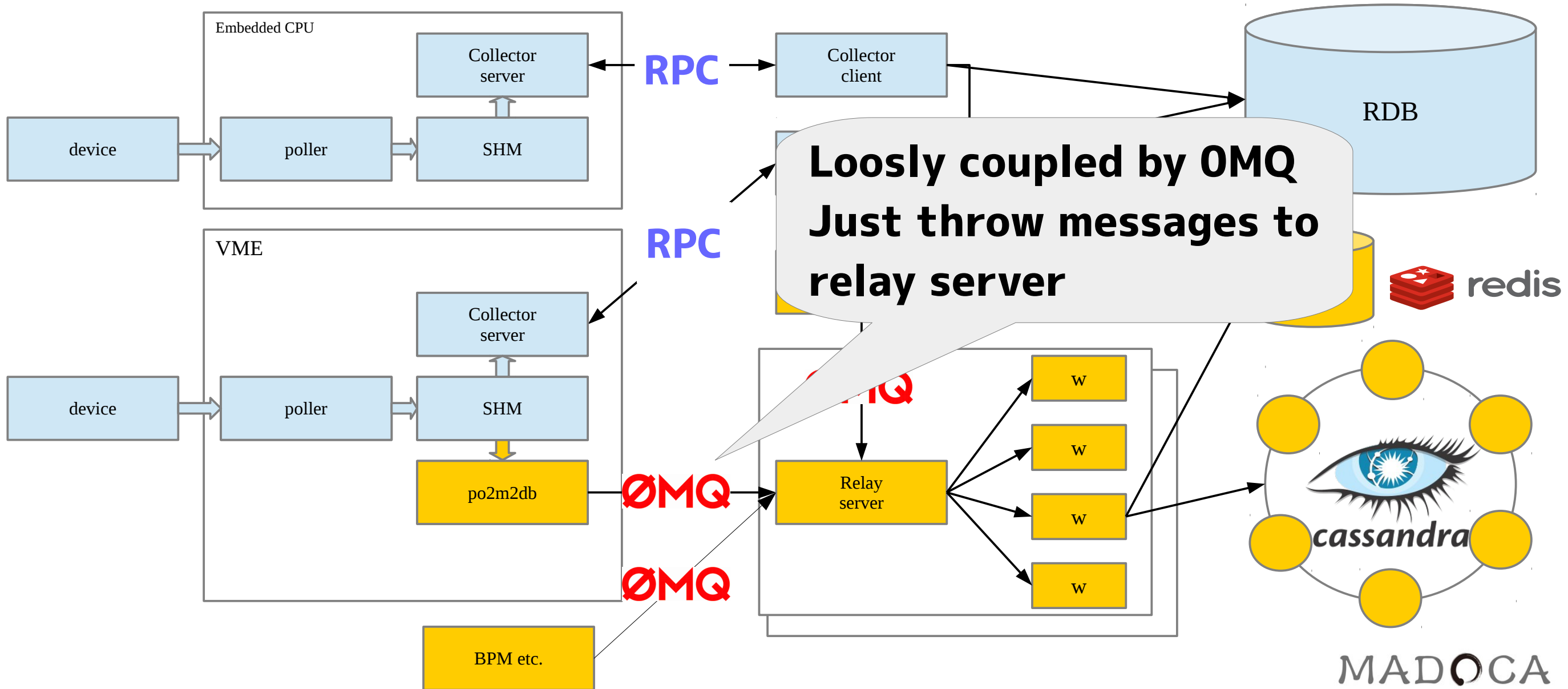
# Entire system



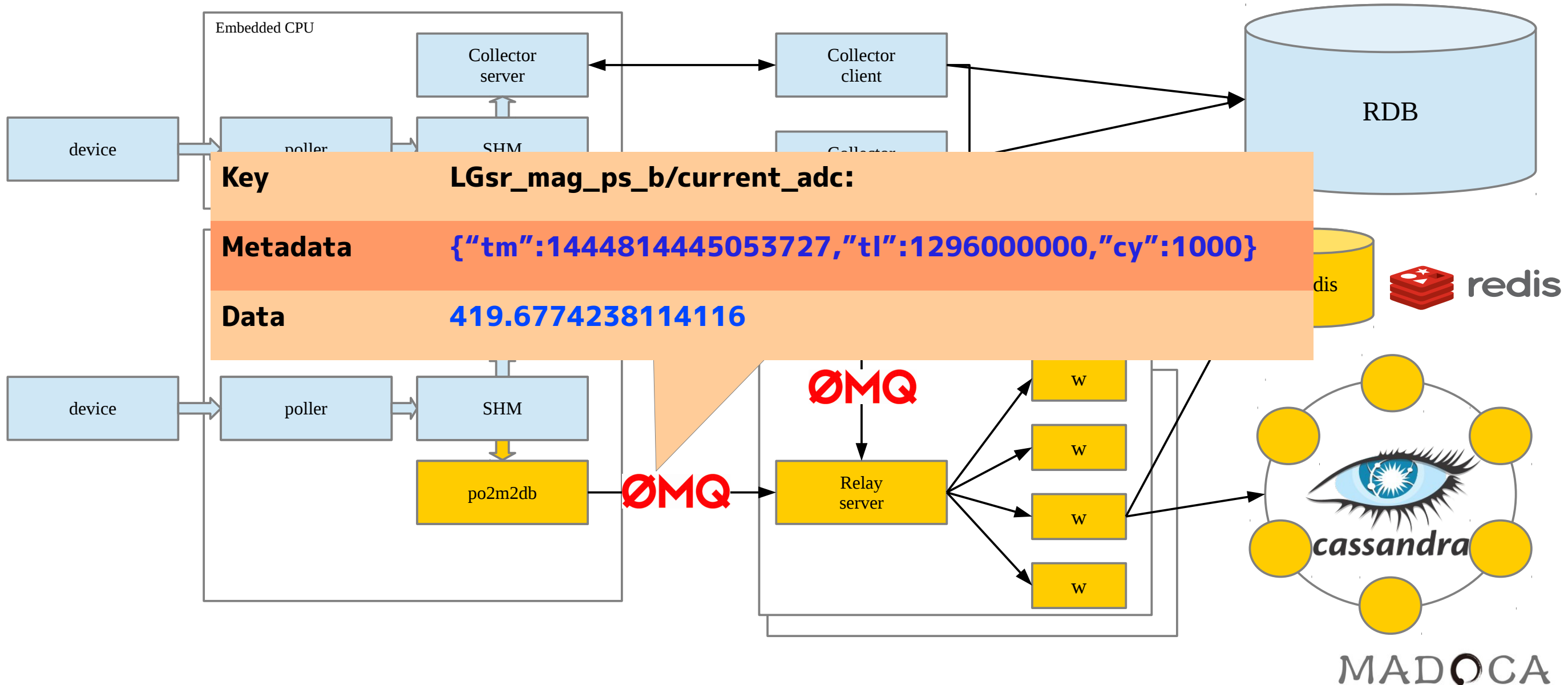
# Entire system



# Entire system



# Message creation



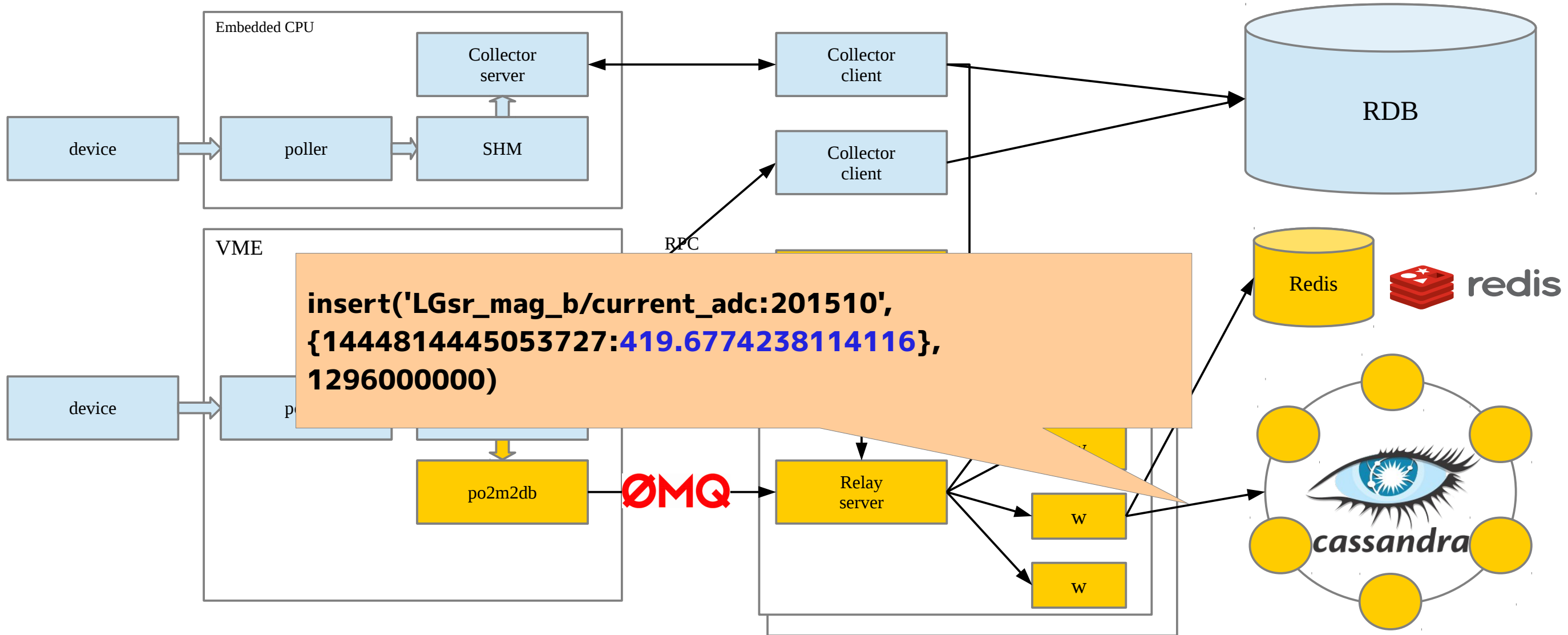
# Message structure

- **3 part message**

Key	LGsr_mag_ps_b/current_adc:
Metadata	<code>{"tm":1444814445053727,"tl":1296000000,"cy":1000}</code>
Data	<code>419.6774238114116</code>

- **Key: raw string**
- **Metadata and data are packed by MessagePack**

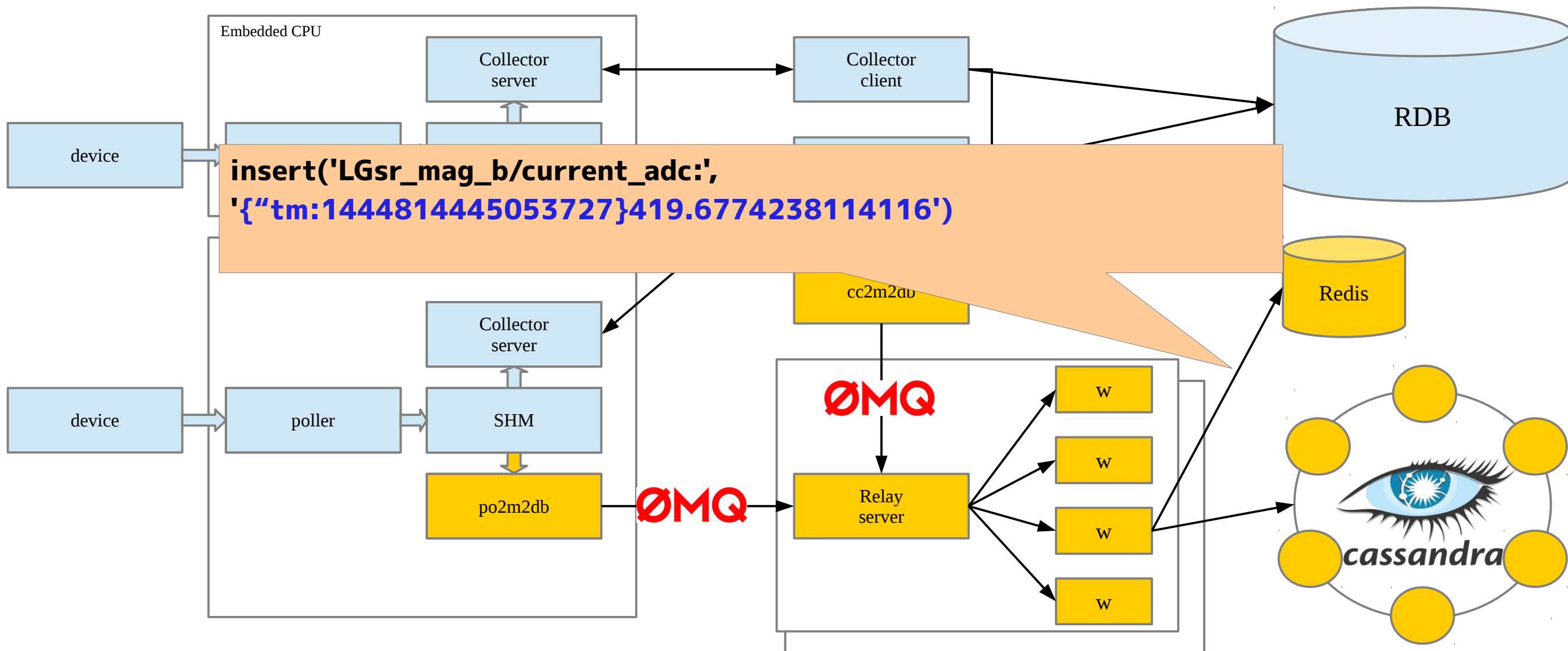
# Convert to Cassandra command



Blue means MessagePacked string



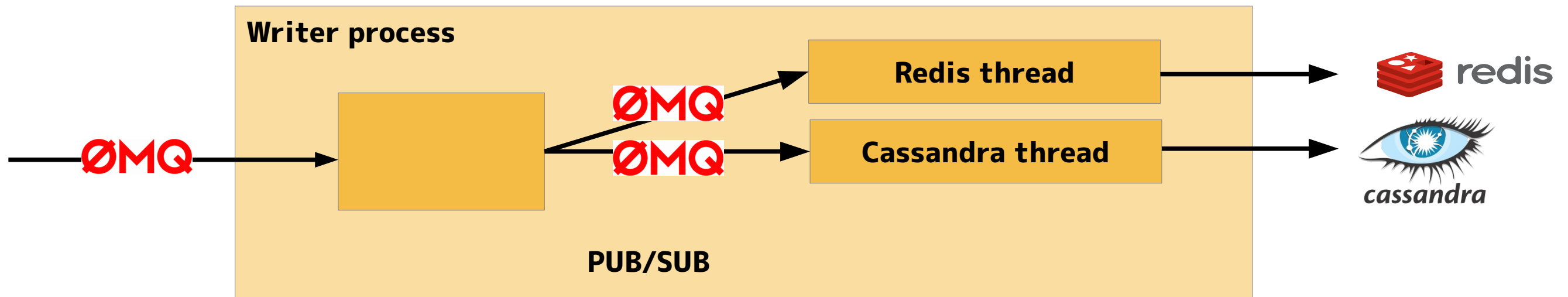
# Convert to Redis Command



Blue means MessagePacked string

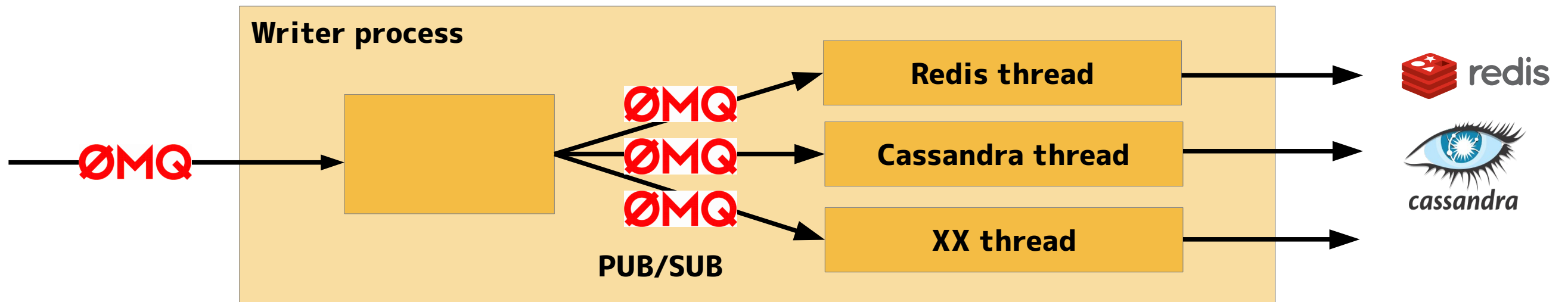
# Writer

- Converts messages to insert commands
- Plug-in structure using 0MQ's in-process pub/sub



# Writer

- Converts messages to insert commands
- Plug-in structure using 0MQ's in-process pub/sub
  - Other DB engine or anything may be added in the future



# Structure of Cassandra

- key : one key / one signal one day
- LGsr\_mag\_ps\_b/current\_adc:20151003

**Keyspace: database**

**Column Family: Table**

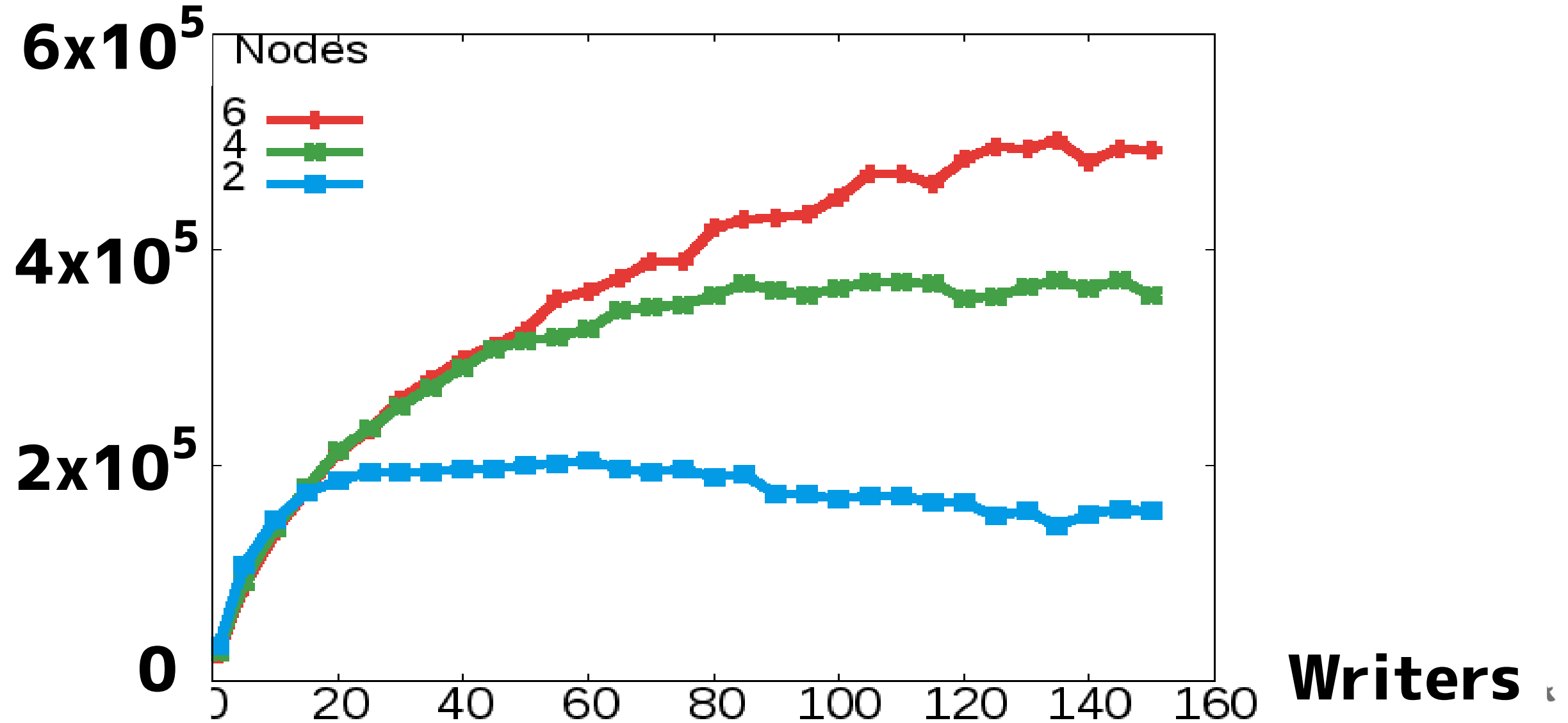
Row Key	Column			
sig1:20130504	t0	t1	t2	t3
	value0	value1	value2	value3
sig2:20130504	t0	t1		
	value0	value1		
sig3:20130504	t0	t1	t2	t3
	value0	value1	value2	value3

**Column key**

**Column Value**

# Performance; write to Cassandra

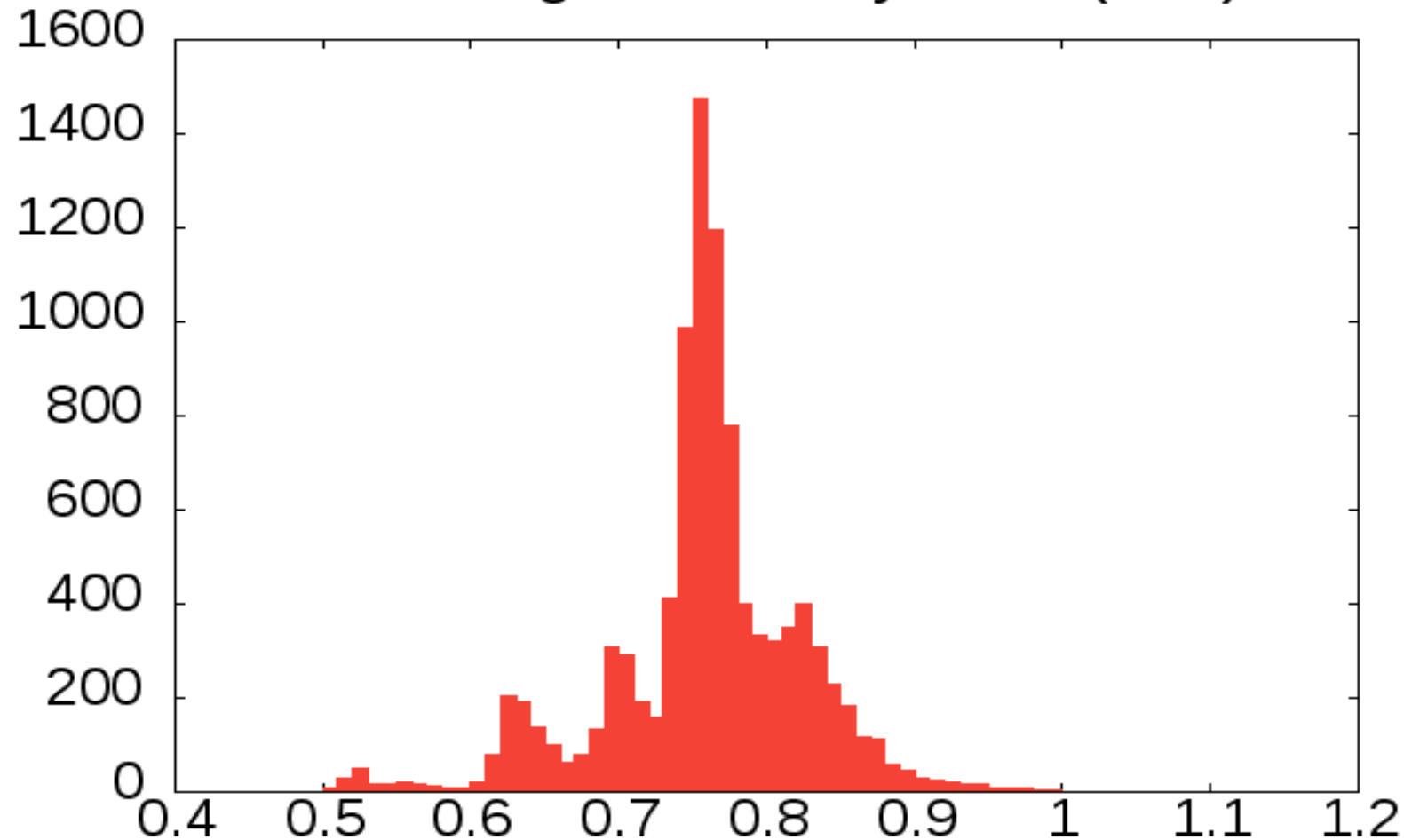
## Values insert/sec (batch)



# Read from Cassandra

- One day data =  $60s * 60min * 24hour$
- Done during normal writing operation

Time to get one day data (sec)

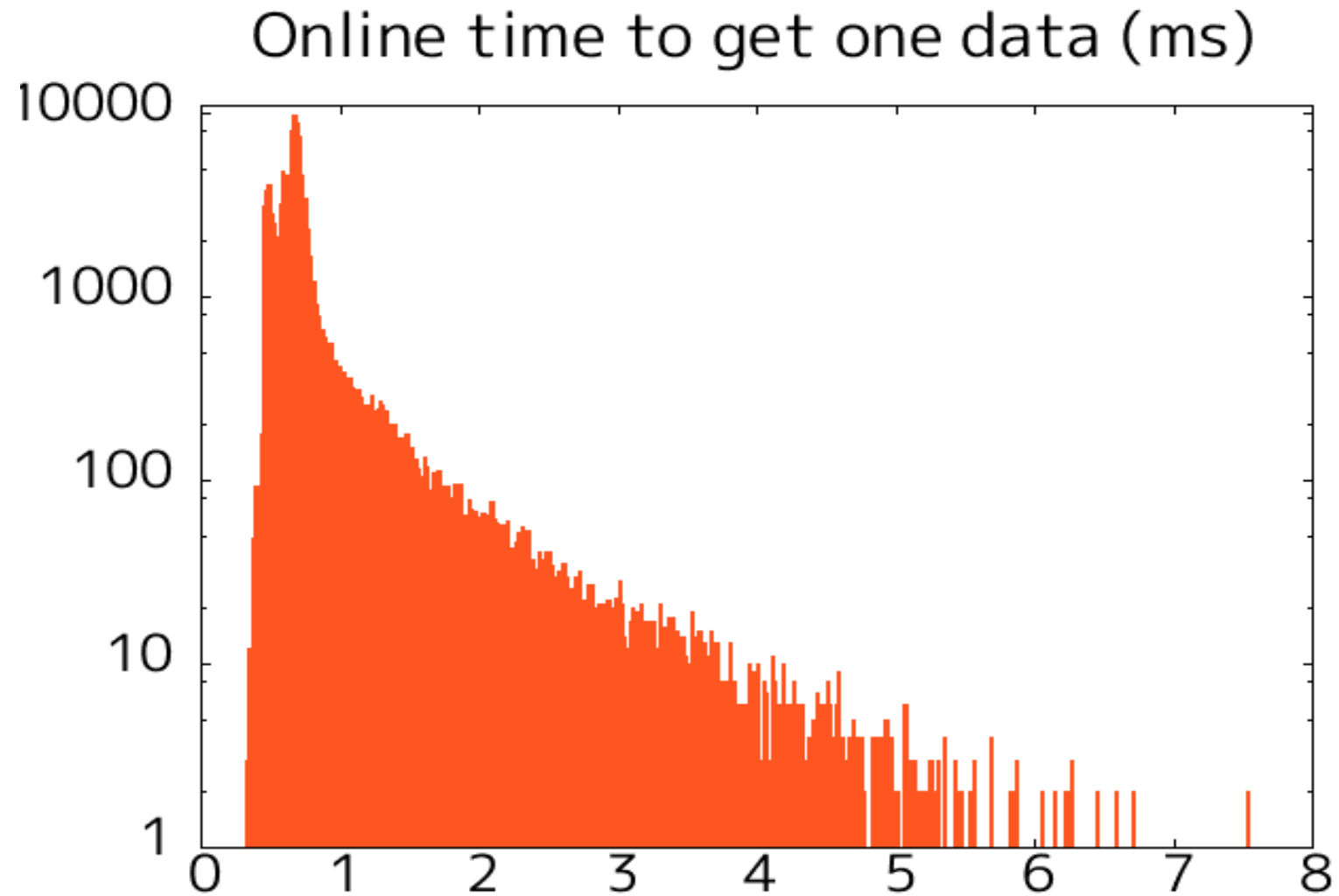


<b>Mean</b>	<b>0.76sec</b>
-------------	----------------

<b>Sigma</b>	<b>0.06sec</b>
--------------	----------------

<b>95%</b>	<b>0.86sec</b>
------------	----------------

# Read from Redis



<b>Mean</b>	<b>0.77ms</b>
<b>Sigma</b>	<b>0.47ms</b>
<b>95%</b>	<b>1.4ms</b>

# One year Test

- **Test performed about one year**
- **No major trouble**
  - **Test**
    - **Forced to shutdown a node and recovery**
- **Some modification are needed for the production run**



# For production run

- **Data migration from RDBMS**
- **Structure modification**
- **Monitoring tools**
- **Client libraries**
- **Node added**

# Data migration from RDBMS

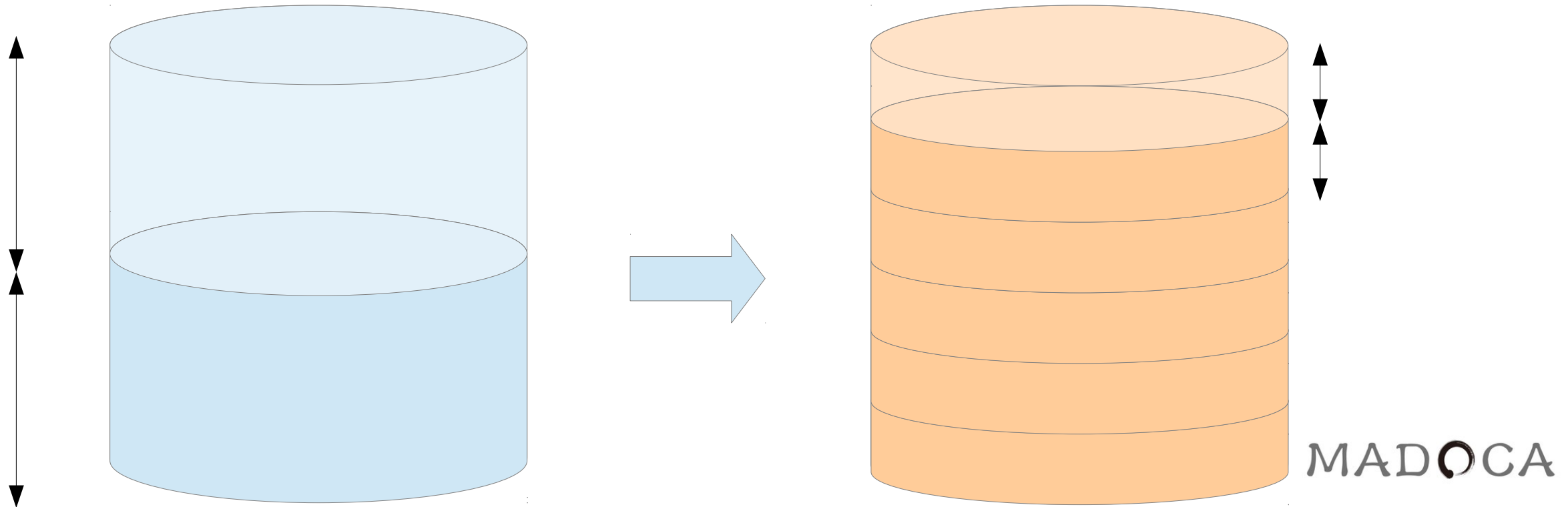
- **Data since 1997**
  - **4TB in RDBMS (logical file size, become larger in RAID disks)**
  - **0.75TB/node 9TB in total in 12 nodes (3 replicas)**

# Structure changed

- **One large column family was divided into small column families of each month**
  - **Cassandra's compaction operation**
    - **Batch operation**
    - **Columns that marked as “delete” are deleted at this time.**

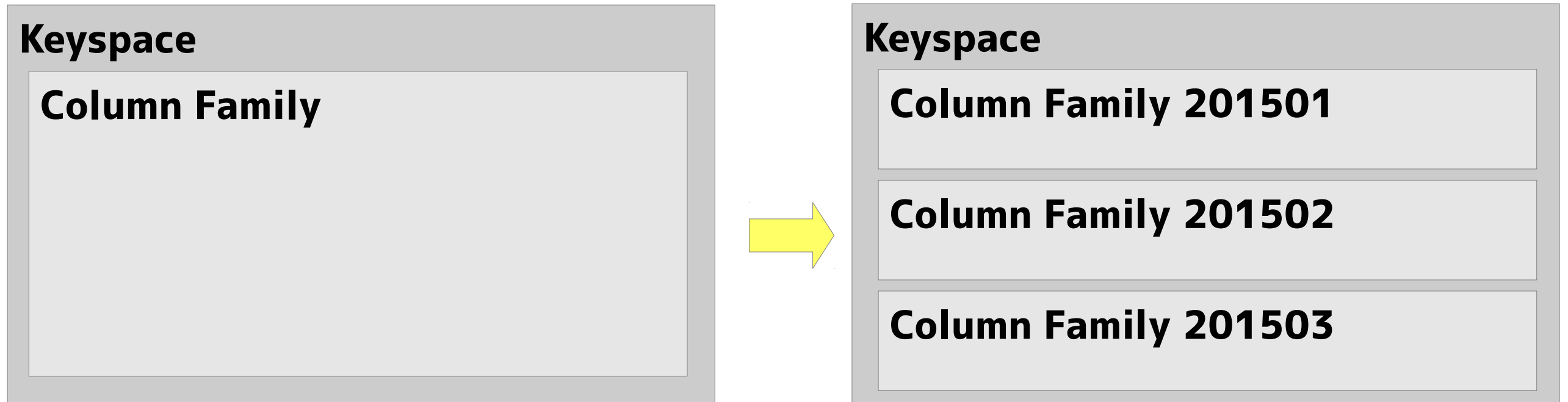
# Temporary disk space for compaction

- One column family needs same size temporary disk space at compaction.
- One big column family cannot be larger than  $\frac{1}{2}$  disk space.



# Structure changed

- **One big column family was divided into small column families of one month**



- **Backup becomes easy by copying separate files**

# Monitoring tool

- **Server system monitoring by Zabbix.**
  - **Not only SNMP but also JXM**
    - **Cassandra is written by JAVA**
    - **JAVA VM monitoring**
- **DAQ system monitoring tool**
  - **For experts**
  - **For operators**

# Zabbix screen

sp8cas-ccr-002: カスタムスクリーン [30秒ごとに更新] - Mozilla Firefox

http://sp8cas-ccr-002/zabbix/screens.php?ddreset=1&sid=5ab1a72e399c36dc

よく見るページ | BL Master | 監視系

ダッシュボード | 概要 | ウェブ | 最新データ | トリガー | イベント | グラフ | 検索 | マップ | ディスカバリ | ITサービス

歴史: 最新データ >> カスタムスクリーン >> カスタムグラフ >> ダッシュボード >> 最新データ

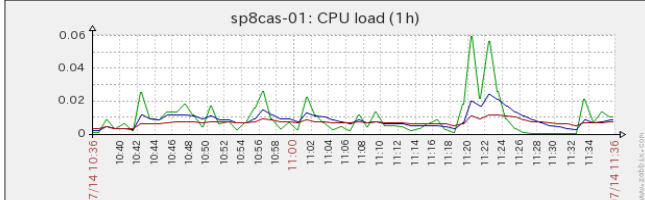
test

ズーム: 1h 2h 3h 6h 12h 1d 7d 14d 1m 3m 6m 1y すべて

2015/07/14 10:36 - 2015/07/14 11:36

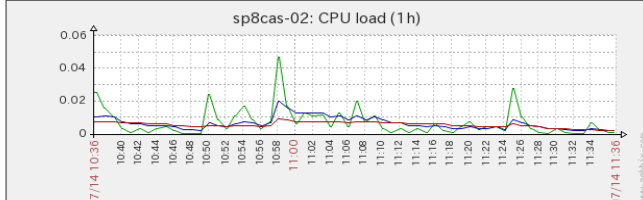
1h (frozen)

sp8cas-01: CPU load (1h)



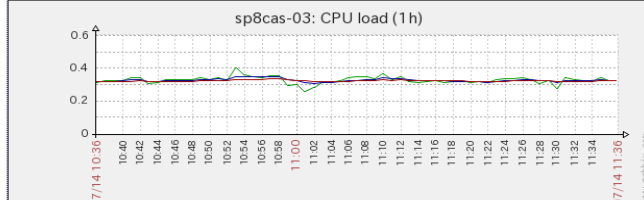
	最新値	最小	平均	最大
Processor load (1 min average per core)	[平均] 0.01	0	0.0098	0.0592
Processor load (5 min average per core)	[平均] 0.0083	0.0017	0.0086	0.0242
Processor load (15 min average per core)	[平均] 0.0075	0.0033	0.0069	0.0117

sp8cas-02: CPU load (1h)



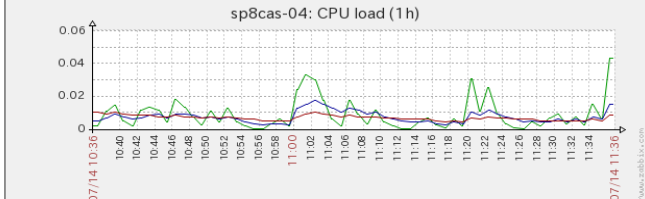
	最新値	最小	平均	最大
Processor load (1 min average per core)	[平均] 0.0008	0	0.0069	0.0467
Processor load (5 min average per core)	[平均] 0.0017	0.0017	0.0063	0.02
Processor load (15 min average per core)	[平均] 0.0017	0.0017	0.0055	0.0092

sp8cas-03: CPU load (1h)



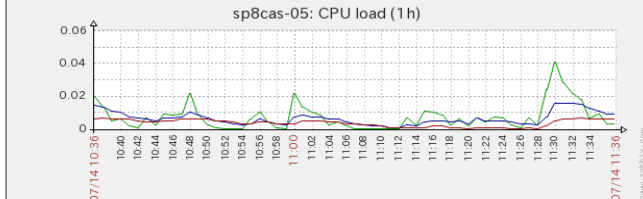
	最新値	最小	平均	最大
Processor load (1 min average per core)	[平均] 0.325	0.26	0.3273	0.4025
Processor load (5 min average per core)	[平均] 0.3258	0.3067	0.326	0.3492
Processor load (15 min average per core)	[平均] 0.3217	0.3175	0.3227	0.3333

sp8cas-04: CPU load (1h)



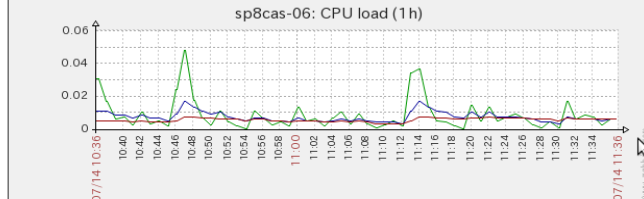
	最新値	最小	平均	最大
Processor load (1 min average per core)	[平均] 0.0433	0	0.0087	0.0433
Processor load (5 min average per core)	[平均] 0.015	0.0025	0.0072	0.0175
Processor load (15 min average per core)	[平均] 0.0083	0.0042	0.0068	0.01

sp8cas-05: CPU load (1h)



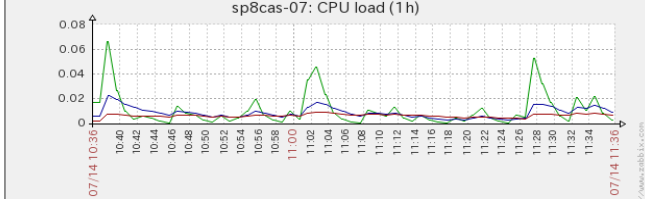
	最新値	最小	平均	最大
Processor load (1 min average per core)	[平均] 0.0033	0	0.0073	0.0408
Processor load (5 min average per core)	[平均] 0.0092	0.0008	0.0063	0.0158
Processor load (15 min average per core)	[平均] 0.0058	0	0.0035	0.0067

sp8cas-06: CPU load (1h)

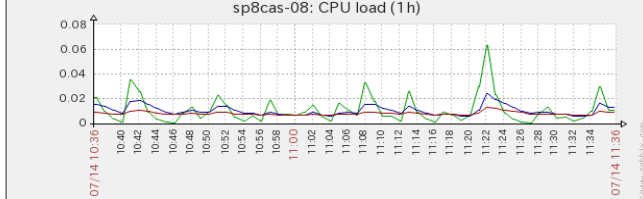


	最新値	最小	平均	最大
Processor load (1 min average per core)	[平均] 0.0058	0	0.0084	0.0483
Processor load (5 min average per core)	[平均] 0.0058	0.0033	0.0073	0.0167
Processor load (15 min average per core)	[平均] 0.0058	0.0033	0.0055	0.0075

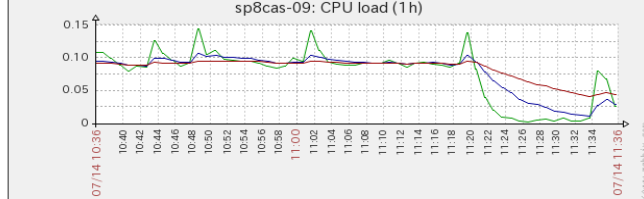
sp8cas-07: CPU load (1h)



sp8cas-08: CPU load (1h)



sp8cas-09: CPU load (1h)

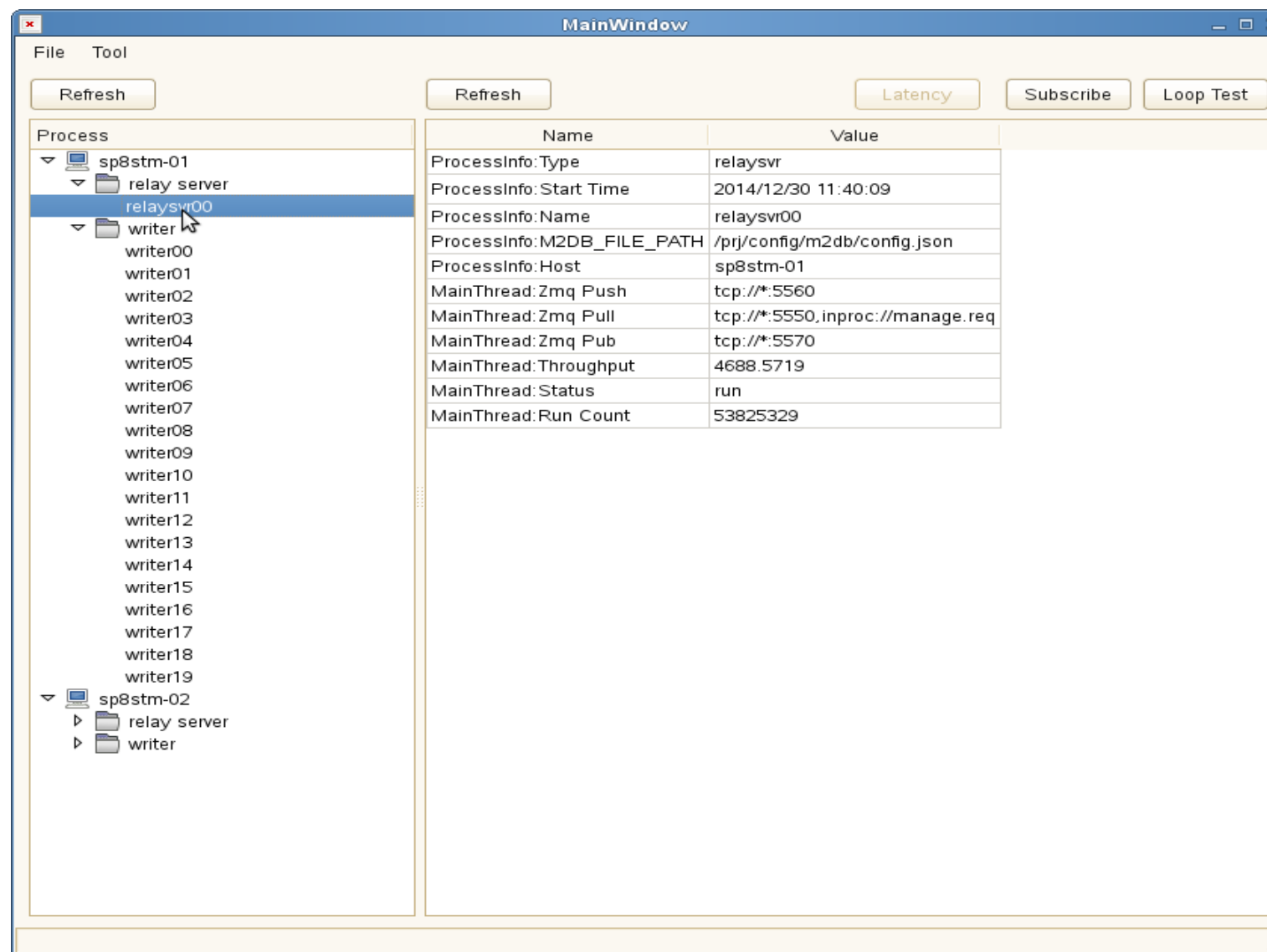


http://sp8cas-ccr-002/zabbix/charts.php?graphid=787&period=3600&stime=20150714103658&sid=5ab1a72e399c36dc

Mozilla Firefox の起動にかかっています...

起動を速くするには(L) 次からは通知しない(A)

# DAQ system monitoring for experts



MainWindow

File Tool

Refresh Refresh Latency Subscribe Loop Test

Process

- sp8stm-01
  - relay server
    - relaysvr00**
    - writer
      - writer00
      - writer01
      - writer02
      - writer03
      - writer04
      - writer05
      - writer06
      - writer07
      - writer08
      - writer09
      - writer10
      - writer11
      - writer12
      - writer13
      - writer14
      - writer15
      - writer16
      - writer17
      - writer18
      - writer19
- sp8stm-02
  - relay server
  - writer

Name	Value
ProcessInfo:Type	relaysvr
ProcessInfo:Start Time	2014/12/30 11:40:09
ProcessInfo:Name	relaysvr00
ProcessInfo:M2DB_FILE_PATH	/prj/config/m2db/config.json
ProcessInfo:Host	sp8stm-01
MainThread:Zmq Push	tcp://*:5560
MainThread:Zmq Pull	tcp://*:5550,inproc://manage.req
MainThread:Zmq Pub	tcp://*:5570
MainThread:Throughput	4688.5719
MainThread:Status	run
MainThread:Run Count	53825329



# DAQ System monitoring for shift operators



The screenshot shows a window titled "MainWindow" with a table of system components. The table has four columns: "Status", "Running", "Name", and "Update Time". Each row represents a different component, all of which are running and have a green status indicator.

Status	Running	Name	Update Time
	12/12	Archive DB (Cassandra)	2015/01/07 12:17:05
	2/2	Online DB (Redis)	2015/01/07 12:17:05
	2/2	Relay Process	2015/01/07 12:17:05
	40/40	Writer Process	2015/01/07 12:17:05

# Client libraries

- **Mainly C and C++**
  - **For applications written in C**
  - **Same interfaces, no modification to source code of application**
    - **Just re-link**
- **Python modules**
  - **for Web applications**
- **We used**
  - **ZeroMQ, Messagepack, Cassandra CQL/Thrift , Redis**

# Cassandra Cluster

<b>Number of nodes</b>	<b>12</b>
<b>Server</b>	<b>Dell PowerEdge R420</b>
<b>CPU</b>	<b>Intel Xeon E5-2420 2.2GHz</b>
<b>Memory</b>	<b>16GB</b>
<b>System disk</b>	<b>600GB 15k rpm</b>
<b>Data disks</b>	<b>3TB 7200 rpm x3</b>
<b>Cassandra</b>	<b>2.0.10</b>
<b>JavaVM</b>	<b>JRE1.7.0-67-b01</b>

# Summary

- **We implemented new data acquisition and store system with new technologies**
- **Apache Cassandra provides high-performance, reliable, scalable and flexible data store that was impossible by RDBMS**
- **We build supporting infrastructures for healthy operations**
- **The system is stably running more than one year including test run**