An attempt to innovate the standard model of control systems

R. Ammendola (INFN Roma TV)

C. Bisegni (INFN-LNF)

S. Calabrò (LAL & INFN-LNF)

L. Catani (INFN Roma TV)

P. Ciuffetti (INFN-LNF)

G. Di Pirro (INFN-LNF)

L. Foggetta (LAL & INFN-LNF)

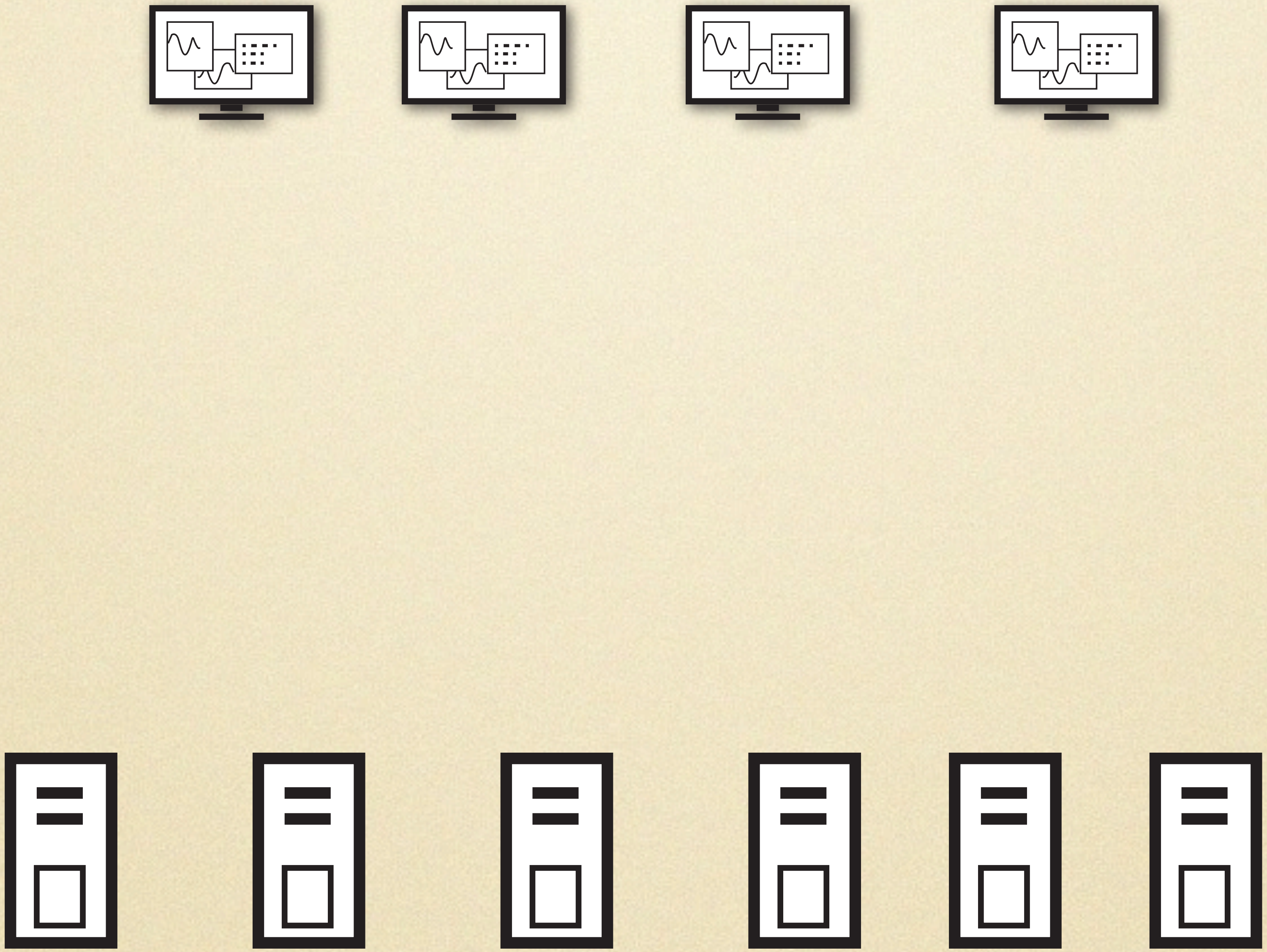
G. Mazzitelli (INFN-LNF)

A. Stecchi (INFN-LNF)

F. Zani (INFN Roma TV)

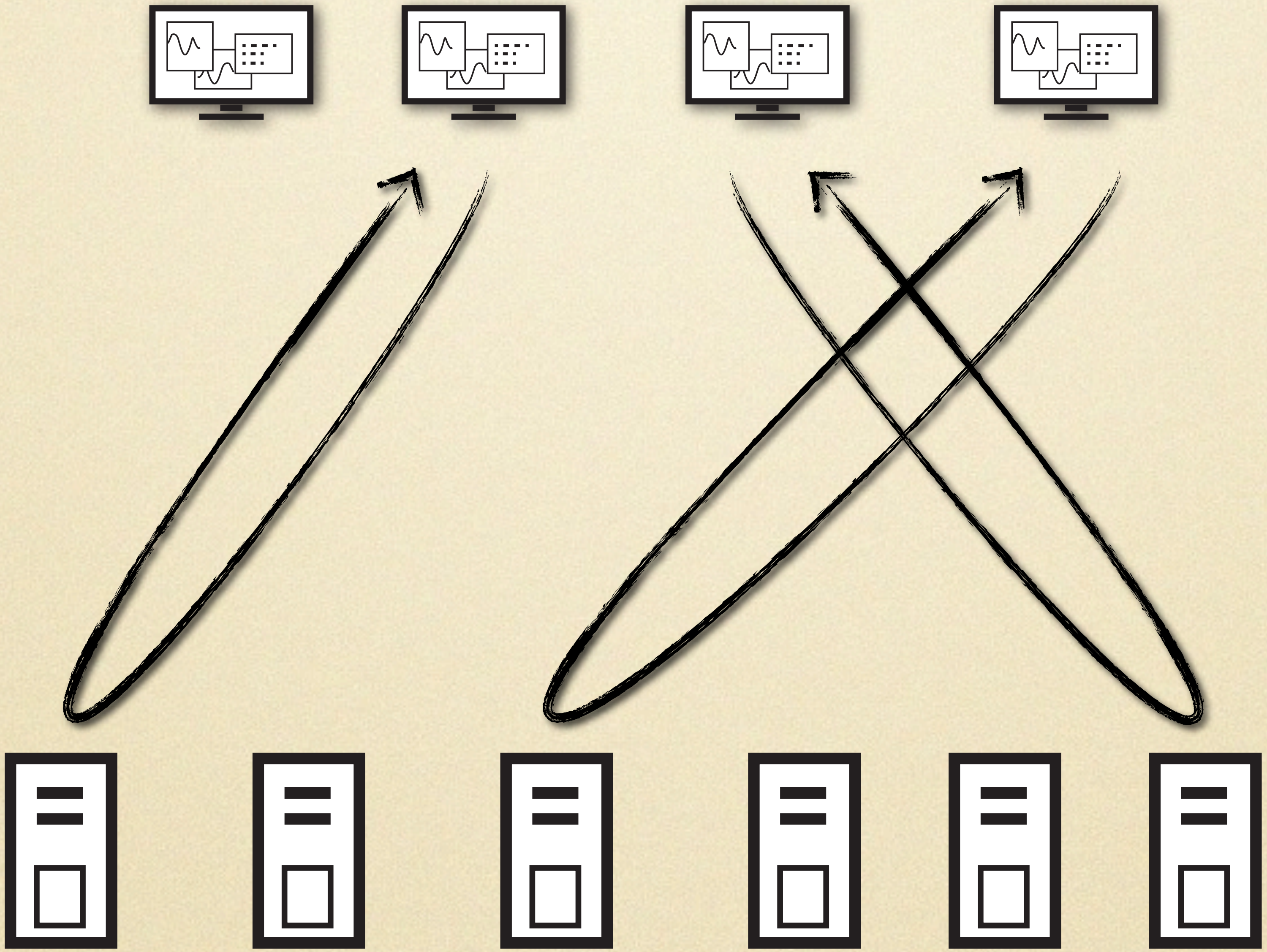
“The *standard model* consists of a local area network providing communication between front end microcomputers, connected to the accelerator, and workstations, providing the operator interface and computational support.”

control room



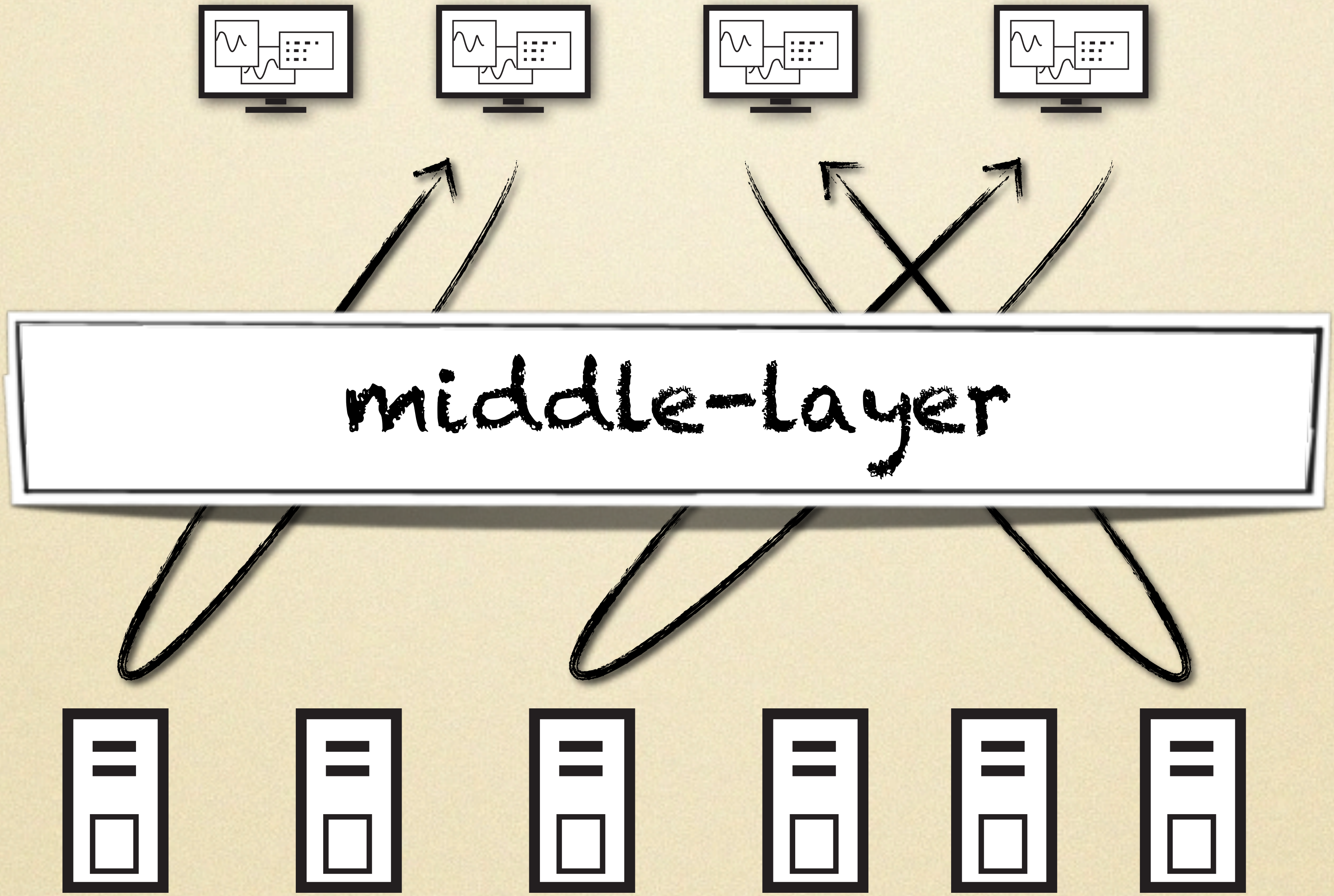
accelerator

control room



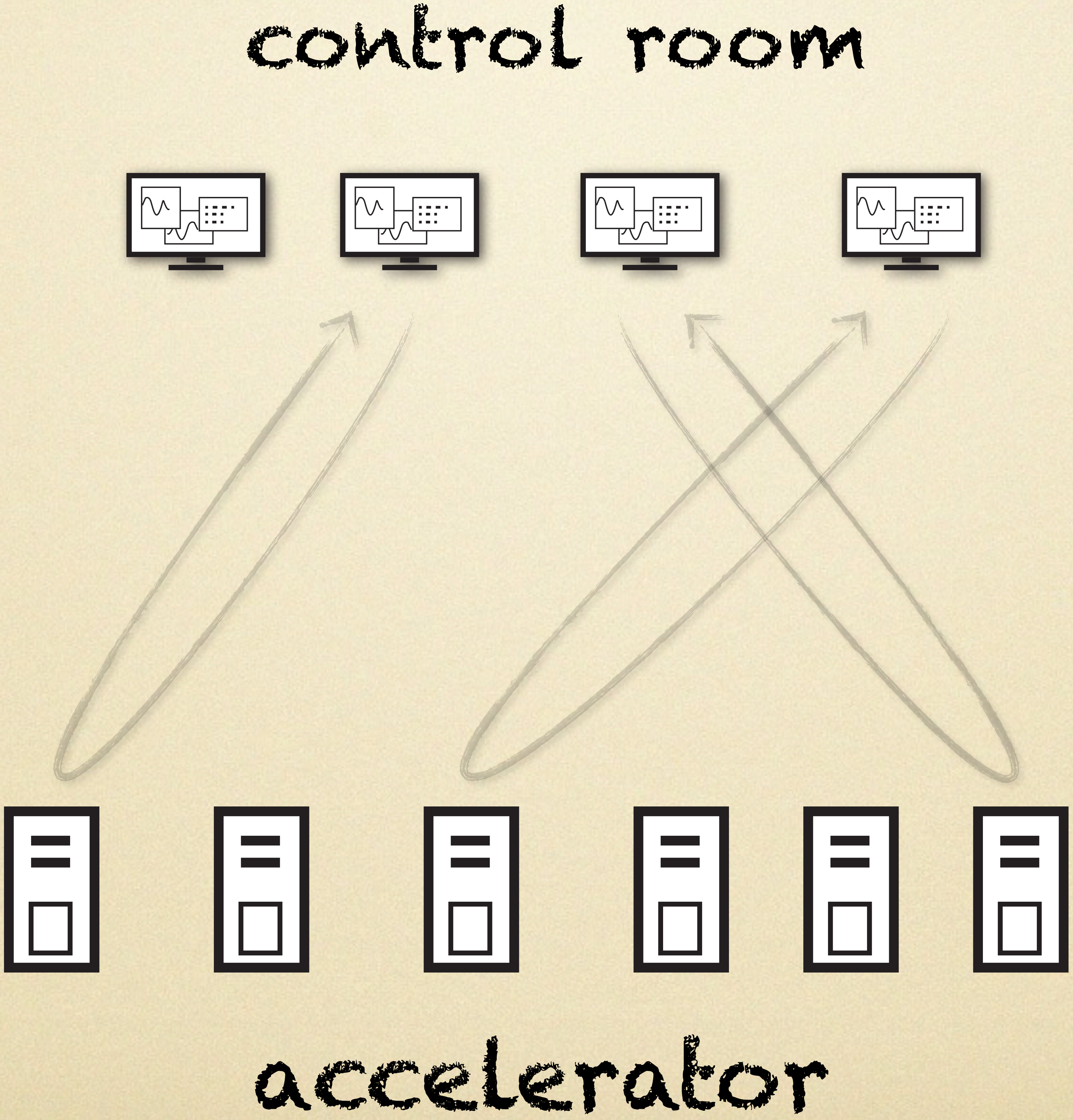
accelerator

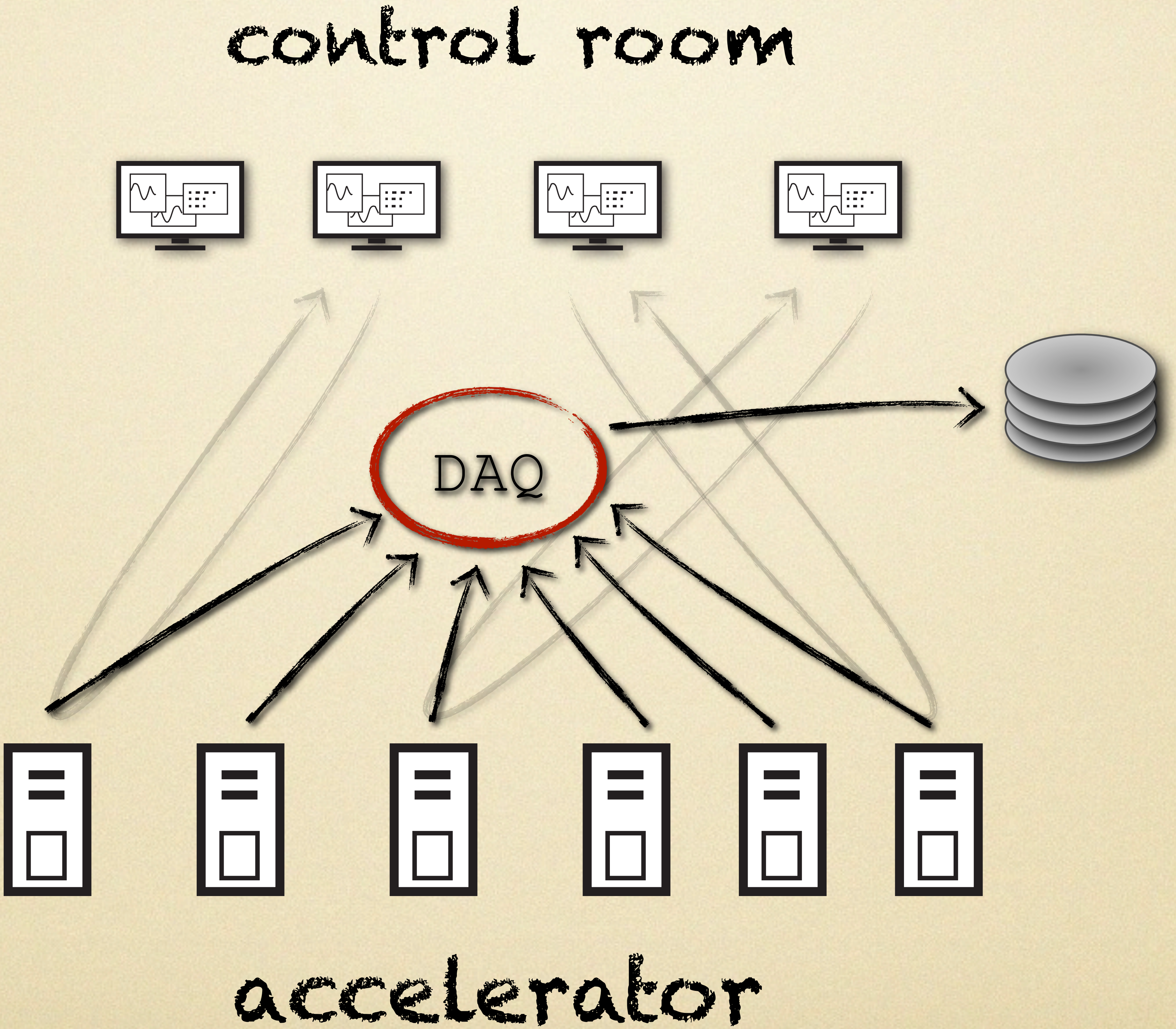
control room



middle-layer

accelerator


control room
accelerator


control room
DAQ
accelerator

the starting point

- **goal**: develop a new solution for a control system's DAQ
- use key/value db as alternative to RDBMS
    - fast, scalable, distributed storage, low-cost servers

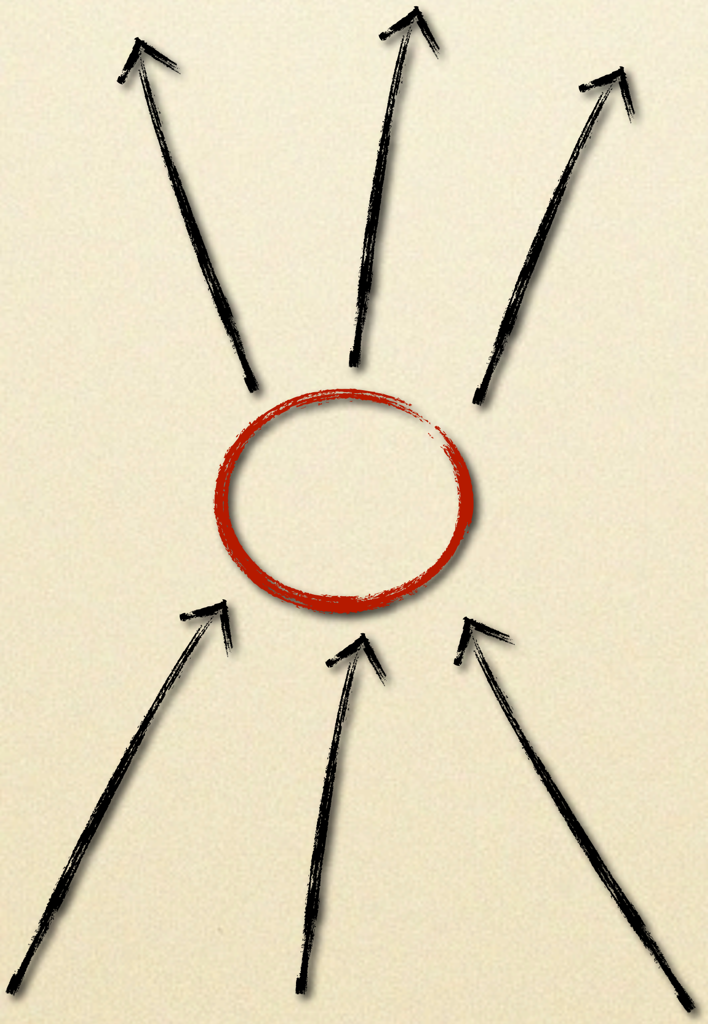
the next step

the next step

- **extended goal**: *key/value* db looks great, can we use it for **live data** ?
  - no, data retrieving too slow

the next step

- **extended goal**: *key/value* db looks great, can we use it for **live data** ?
    - no, data retrieving too slow
- use distributed caching instead
    - same topology, same data structure, similar scalability



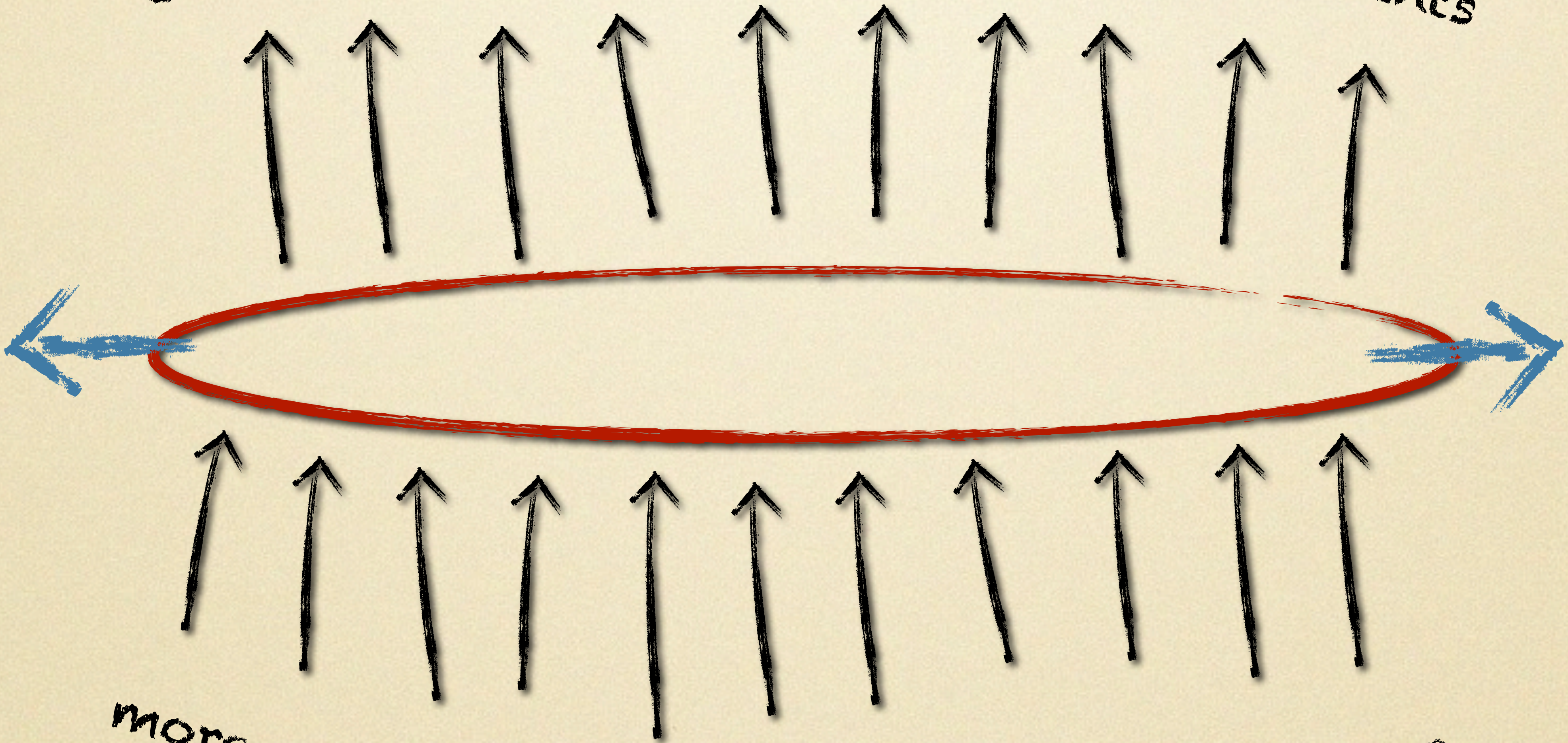
control room

accelerator

control room

more clients

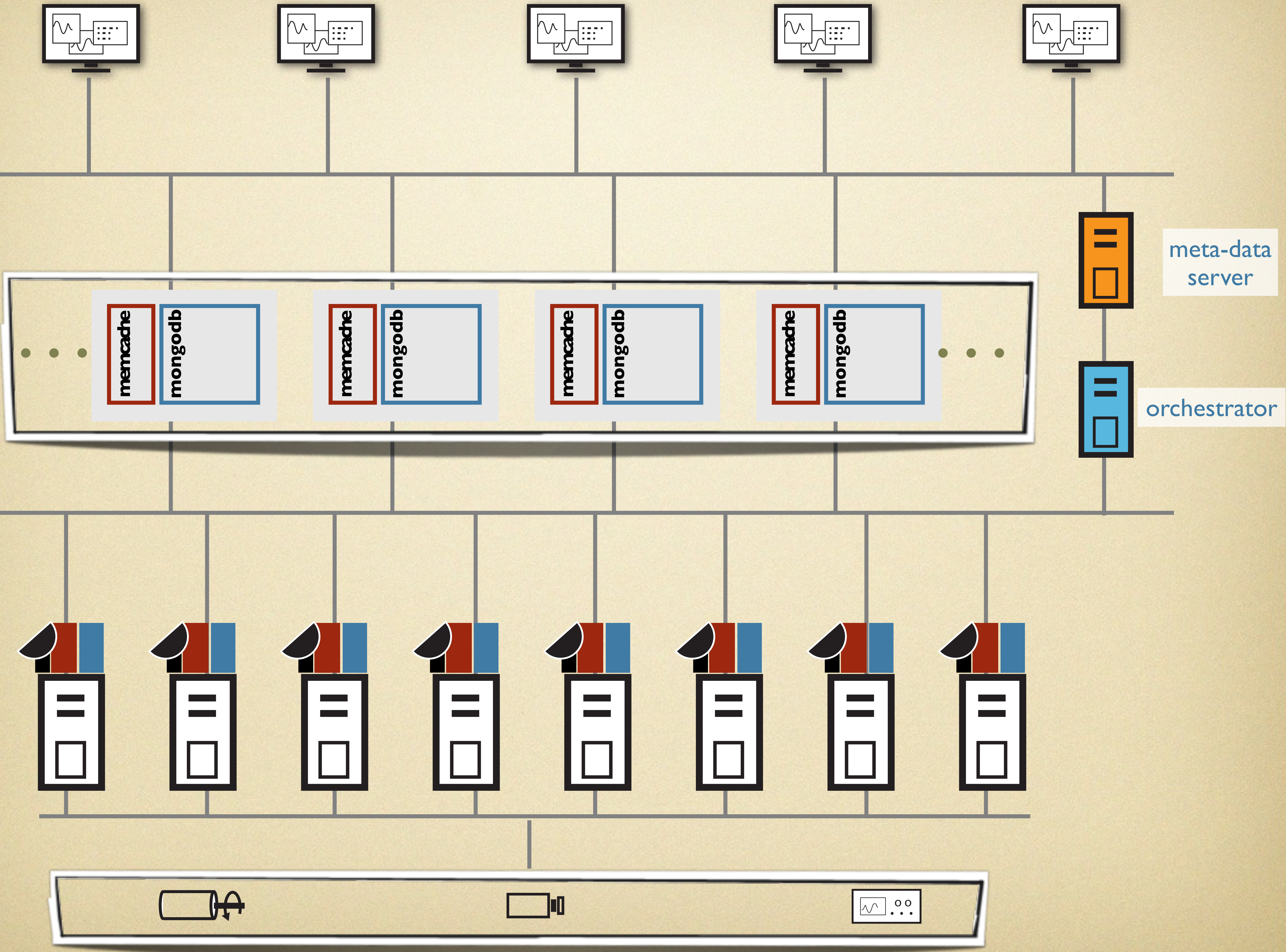
more clients

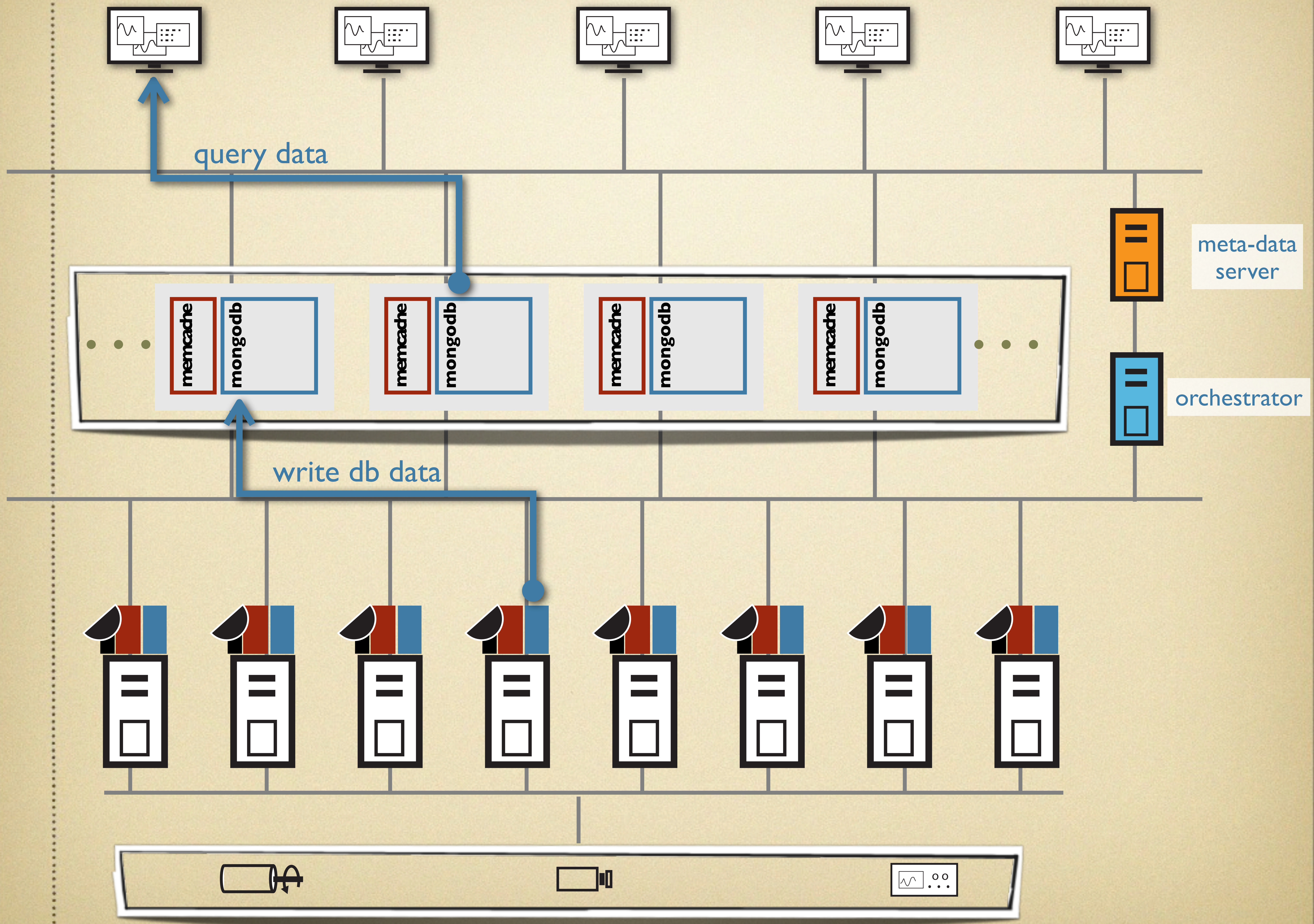


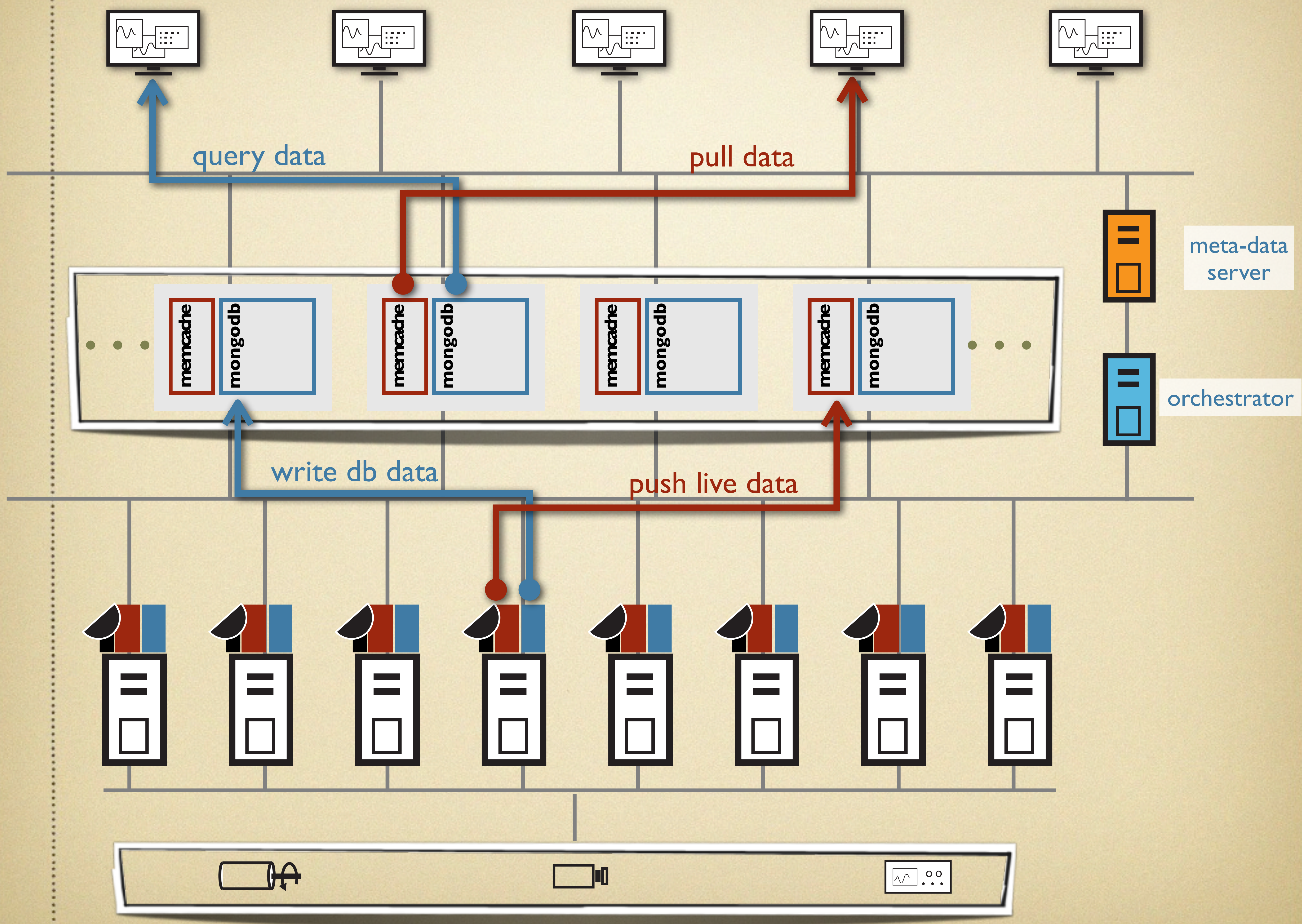
more devices

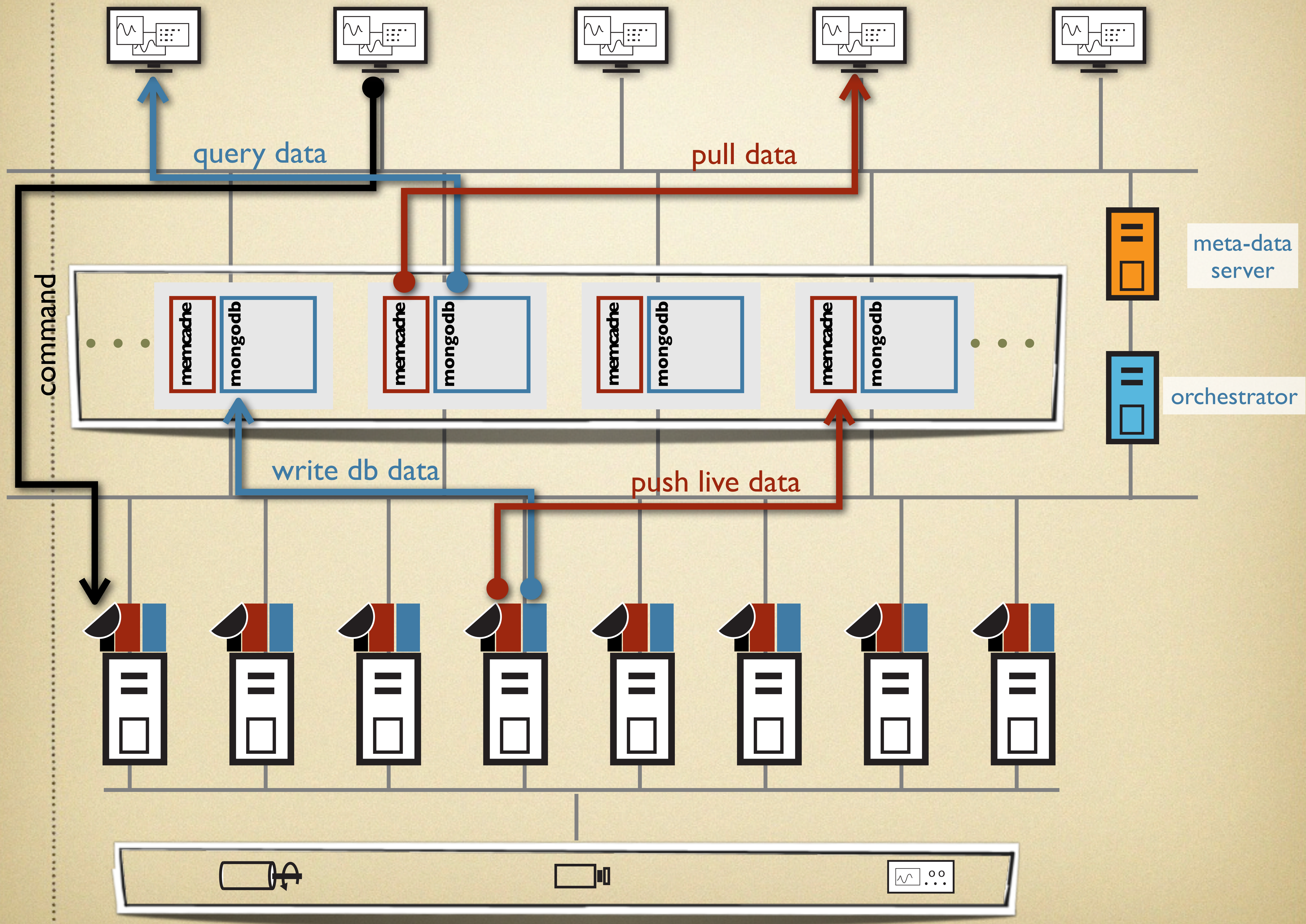
more devices

accelerator

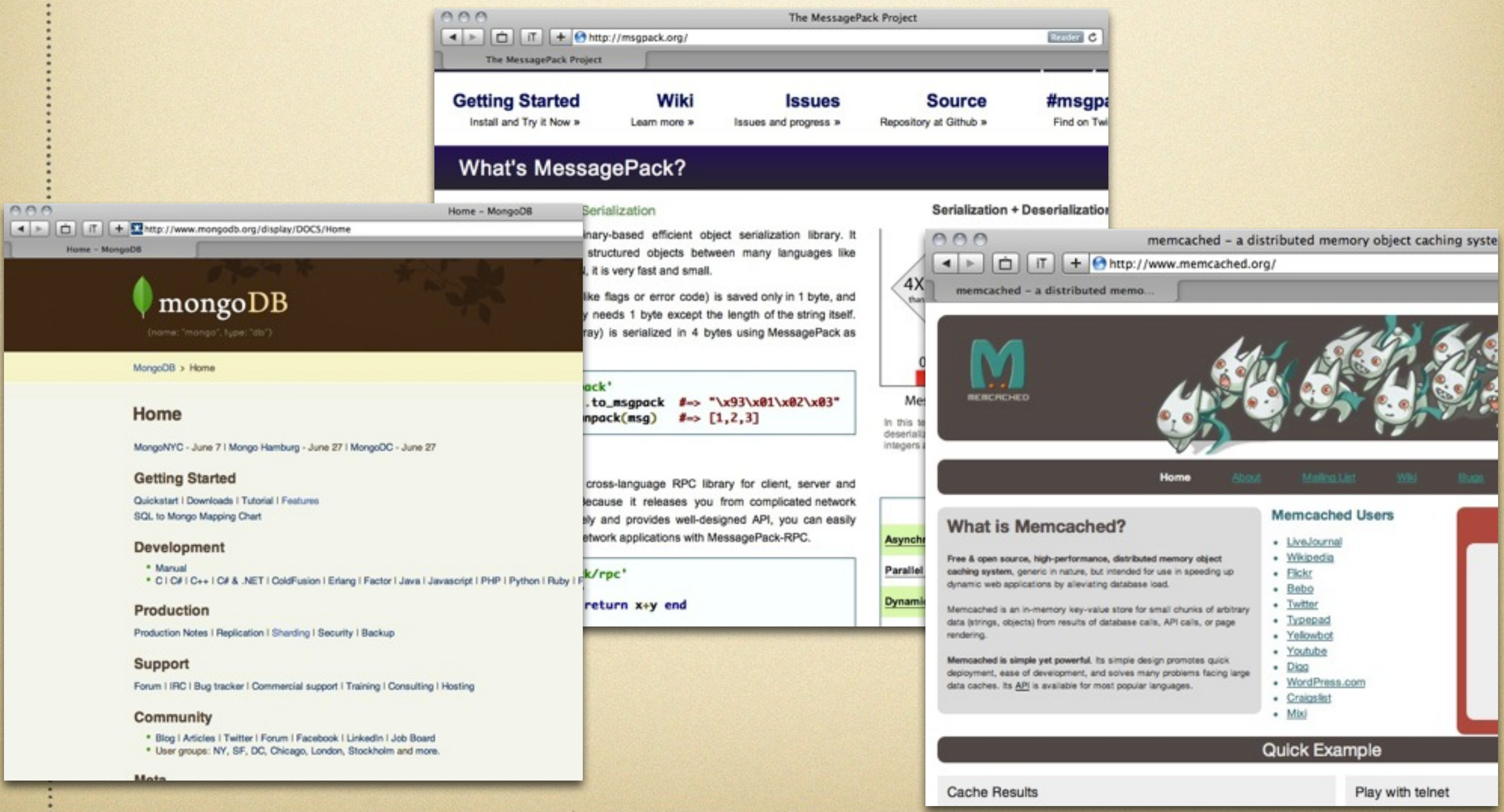

memcache
mongodb
memcache
mongodb
memcache
mongodb
memcache
mongodb
meta-data server
orchestrator


query data
meta-data server
memcache
mongodb
memcache
mongodb
memcache
mongodb
memcache
mongodb
orchestrator
write db data


query data
pull data
meta-data server
orchestrator
memcache
mongodb
memcache
mongodb
memcache
mongodb
memcache
mongodb
write db data
push live data


query data
pull data
meta-data server
orchestrator
command
memcache
mongodb
memcache
mongodb
memcache
mongodb
memcache
mongodb
write db data
push live data

core services candidates



The MessagePack Project

http://msgpack.org/

Getting Started
Install and Try it Now »

Wiki
Learn more »

Issues
Issues and progress »

Source
Repository at Github »

#msgpa
Find on Twi

What's MessagePack?

Serialization

inary-based efficient object serialization library. It structured objects between many languages like , it is very fast and small.

like flags or error code) is saved only in 1 byte, and y needs 1 byte except the length of the string itself. ray) is serialized in 4 bytes using MessagePack as

```
ack'
.to_msgpack  #=> "\x93\x01\x02\x03"
npack(msg)   #=> [1,2,3]
```

cross-language RPC library for client, server and ecause it releases you from complicated network ely and provides well-designed API, you can easily etwork applications with MessagePack-RPC.

```
k/rpc'

return x+y end
```

Serialization + Deserializatio

4X
than

Me

In this te
deserializ
integers

Asynchr
Parallel
Dynami

Home - MongoDB

http://www.mongodb.org/display/DOCS/Home

mongoDB
{name: "mongo", type: "db"}

MongoDB > Home

Home

MongoNYC - June 7 | Mongo Hamburg - June 27 | MongoDC - June 27

Getting Started

Quickstart | Downloads | Tutorial | Features
SQL to Mongo Mapping Chart

Development

- Manual
- C | C# | C++ | C# & .NET | ColdFusion | Erlang | Factor | Java | Javascript | PHP | Python | Ruby | F

Production

Production Notes | Replication | Sharding | Security | Backup

Support

Forum | IRC | Bug tracker | Commercial support | Training | Consulting | Hosting

Community

- Blog | Articles | Twitter | Forum | Facebook | LinkedIn | Job Board
- User groups: NY, SF, DC, Chicago, London, Stockholm and more.

Meta

memcached - a distributed memory object caching syste

http://www.memcached.org/

MEMCACHED

Home About Mailing List Wiki Bugs

What is Memcached?

Free & open source, high-performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load.

Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.

Memcached is simple yet powerful. Its simple design promotes quick deployment, ease of development, and solves many problems facing large data caches. Its API is available for most popular languages.

Memcached Users

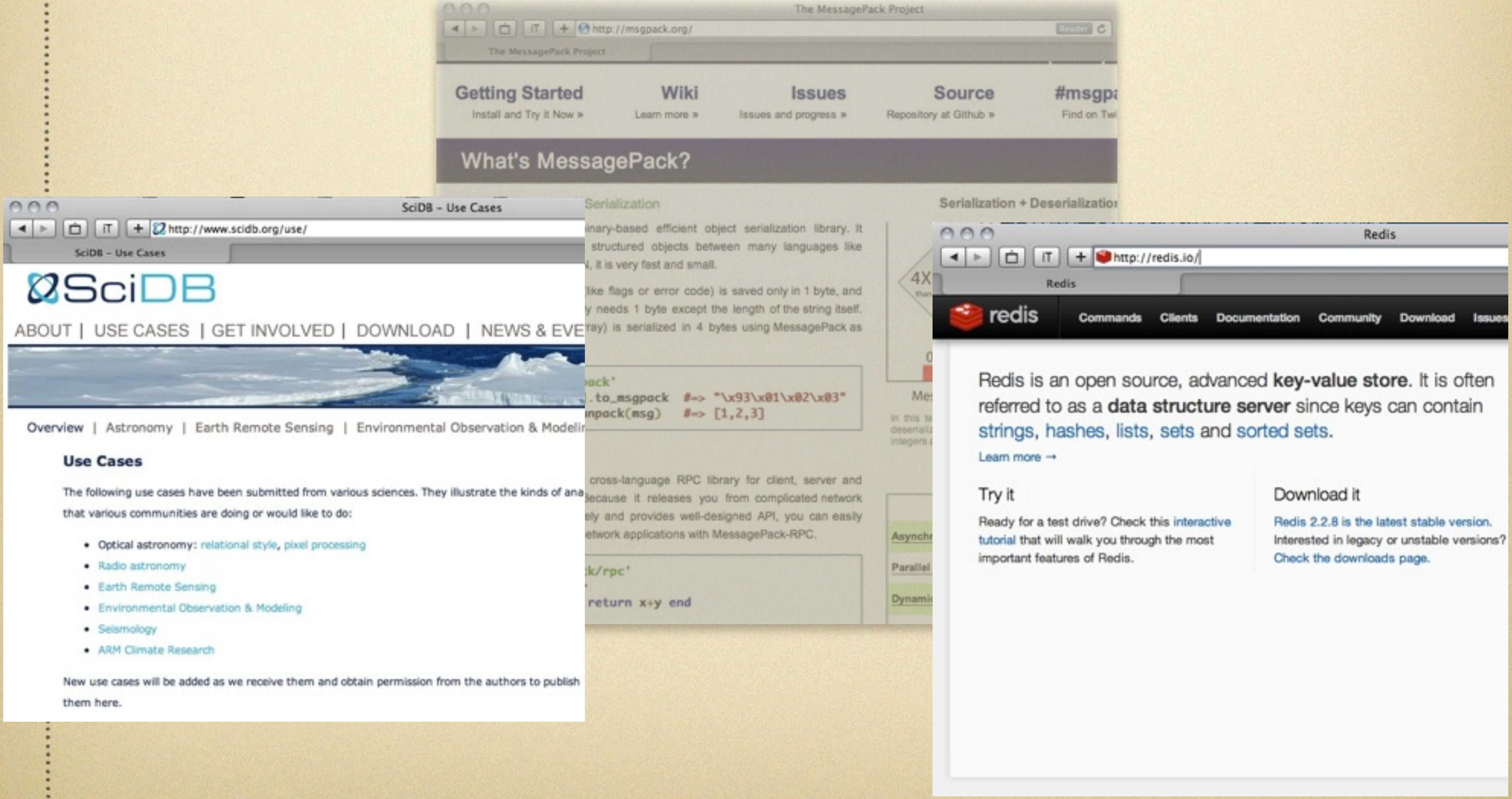
- LiveJournal
- Wikipedia
- Flickr
- Bebo
- Twitter
- Typepad
- Yellowbot
- Youtube
- Digg
- WordPress.com
- Craigslist
- Mixi

Quick Example

Cache Results

Play with telnet

core services alternatives



What's MessagePack?

Serialization

inary-based efficient object serialization library. It structured objects between many languages like N, it is very fast and small.

(like flags or error code) is saved only in 1 byte, and y needs 1 byte except the length of the string itself. ray) is serialized in 4 bytes using MessagePack as

```
ack'
.to_msgpack  #=> "\x93\x01\x02\x03"
npack(msg)   #=> [1,2,3]
```

cross-language RPC library for client, server and Because it releases you from complicated network ely and provides well-designed API, you can easily etwork applications with MessagePack-RPC.

```
k/rpc'

return x+y end
```

Serialization + Deserializatio

SciDB - Use Cases
http://www.scidb.org/use/
SciDB - Use Cases
SciDB
ABOUT | USE CASES | GET INVOLVED | DOWNLOAD | NEWS & EVE
Overview | Astronomy | Earth Remote Sensing | Environmental Observation & Modelin

Use Cases

The following use cases have been submitted from various sciences. They illustrate the kinds of ana that various communities are doing or would like to do:

- Optical astronomy: relational style, pixel processing
- Radio astronomy
- Earth Remote Sensing
- Environmental Observation & Modeling
- Seismology
- ARM Climate Research

New use cases will be added as we receive them and obtain permission from the authors to publish them here.

Redis is an open source, advanced **key-value store**. It is often referred to as a **data structure server** since keys can contain strings, hashes, lists, sets and sorted sets.

Learn more →

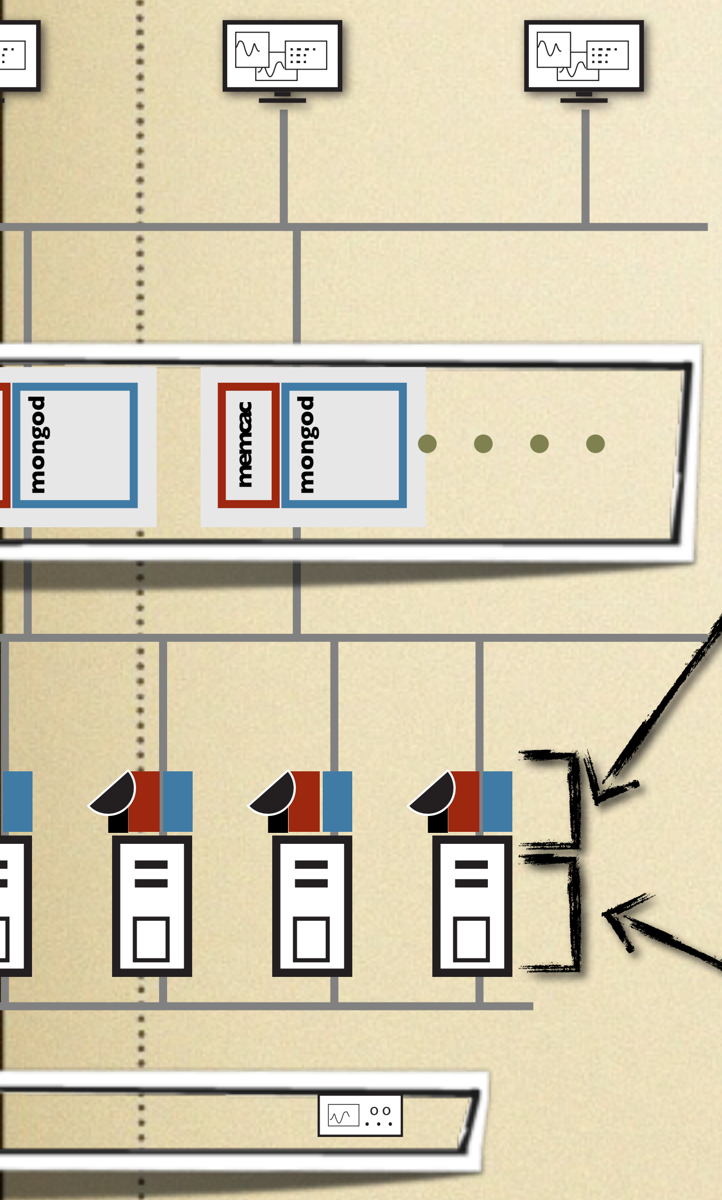
Try it

Ready for a test drive? Check this interactive tutorial that will walk you through the most important features of Redis.

Download it

Redis 2.2.8 is the latest stable version. Interested in legacy or unstable versions? Check the downloads page.

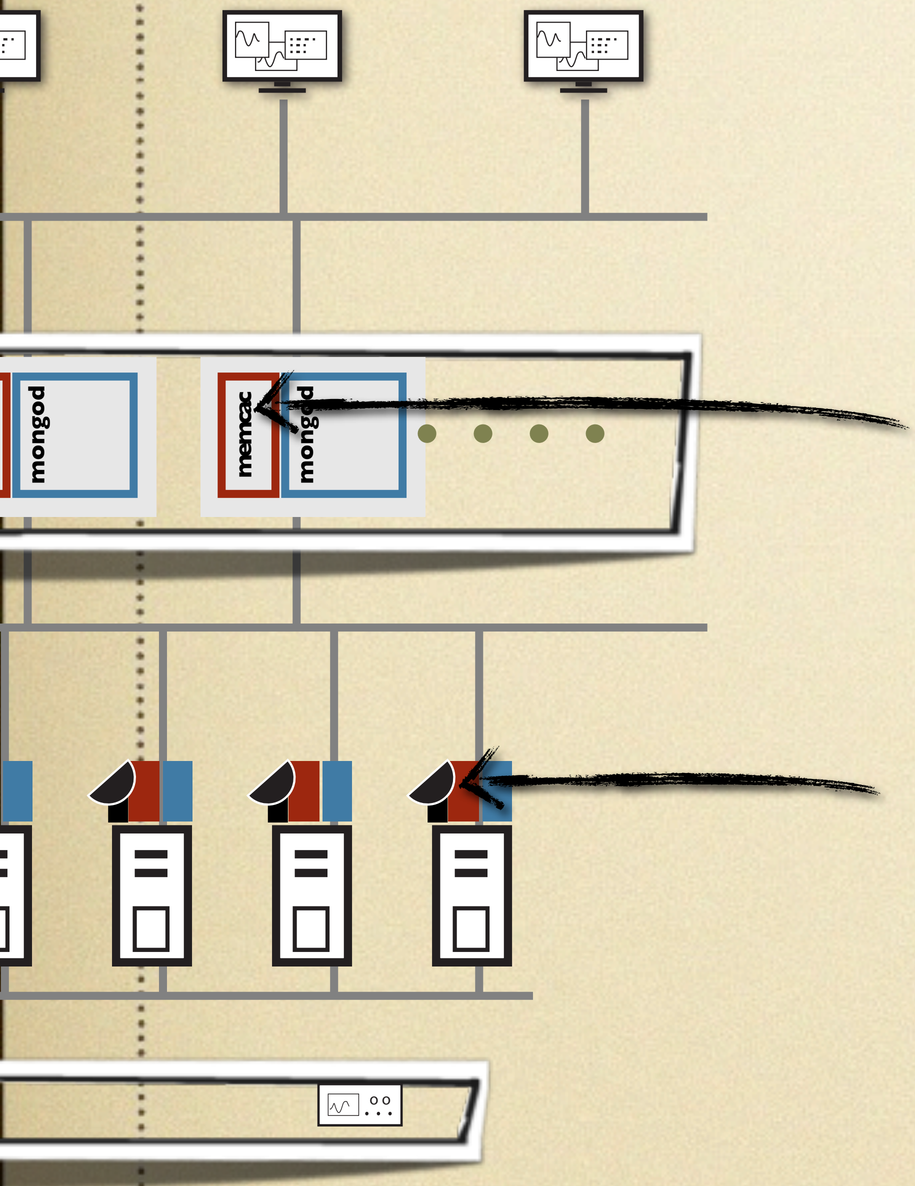
Control Library e Control Unit



multi-threads

- The Control Library is the set of functions needed to hw-driver developer to communicate with the CS. API allows:
  - Manage configurations
  - writing data to Live and History
  - Commands handling
- The Control Unit implements the CL to control an accelerator component or a family thereof.

Live data

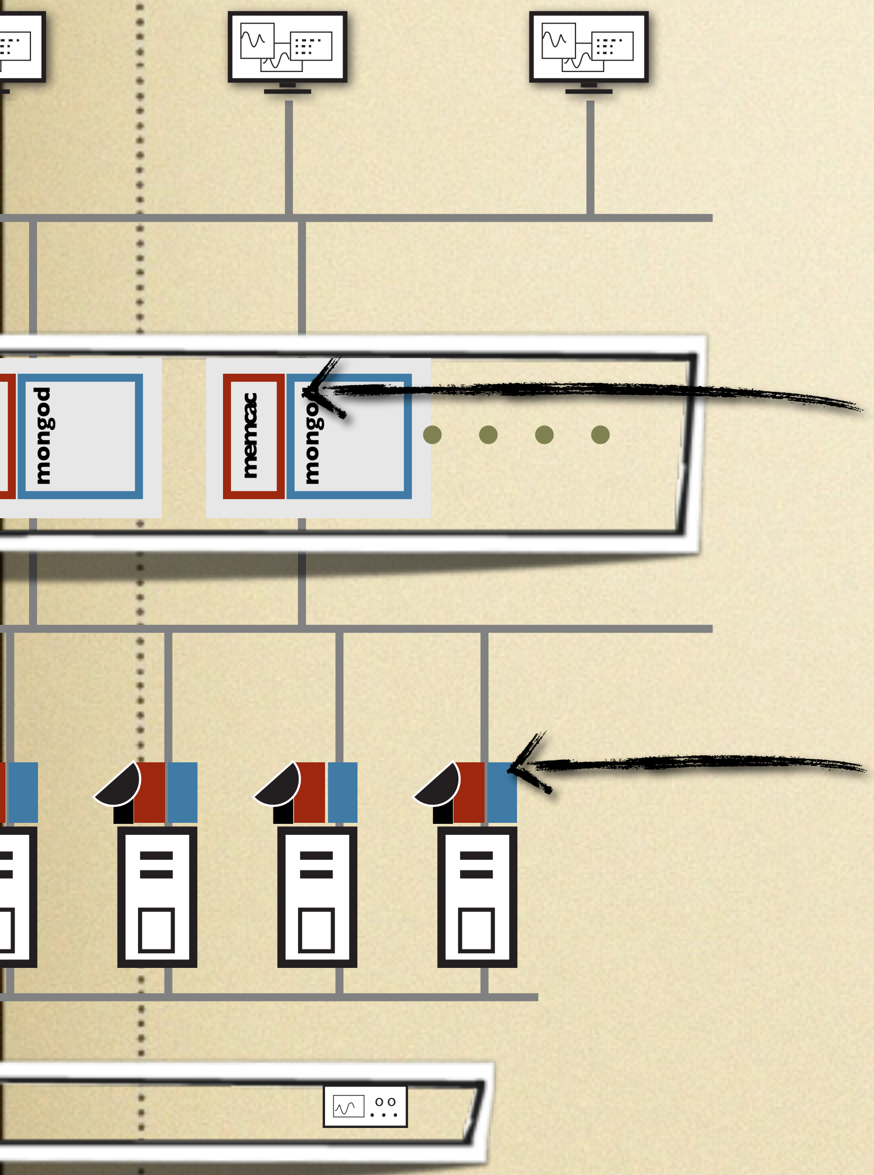


- Allows high-performance caching of data produced by all components managed by CS.

- one *key* per data (a single “container” continuously updated)

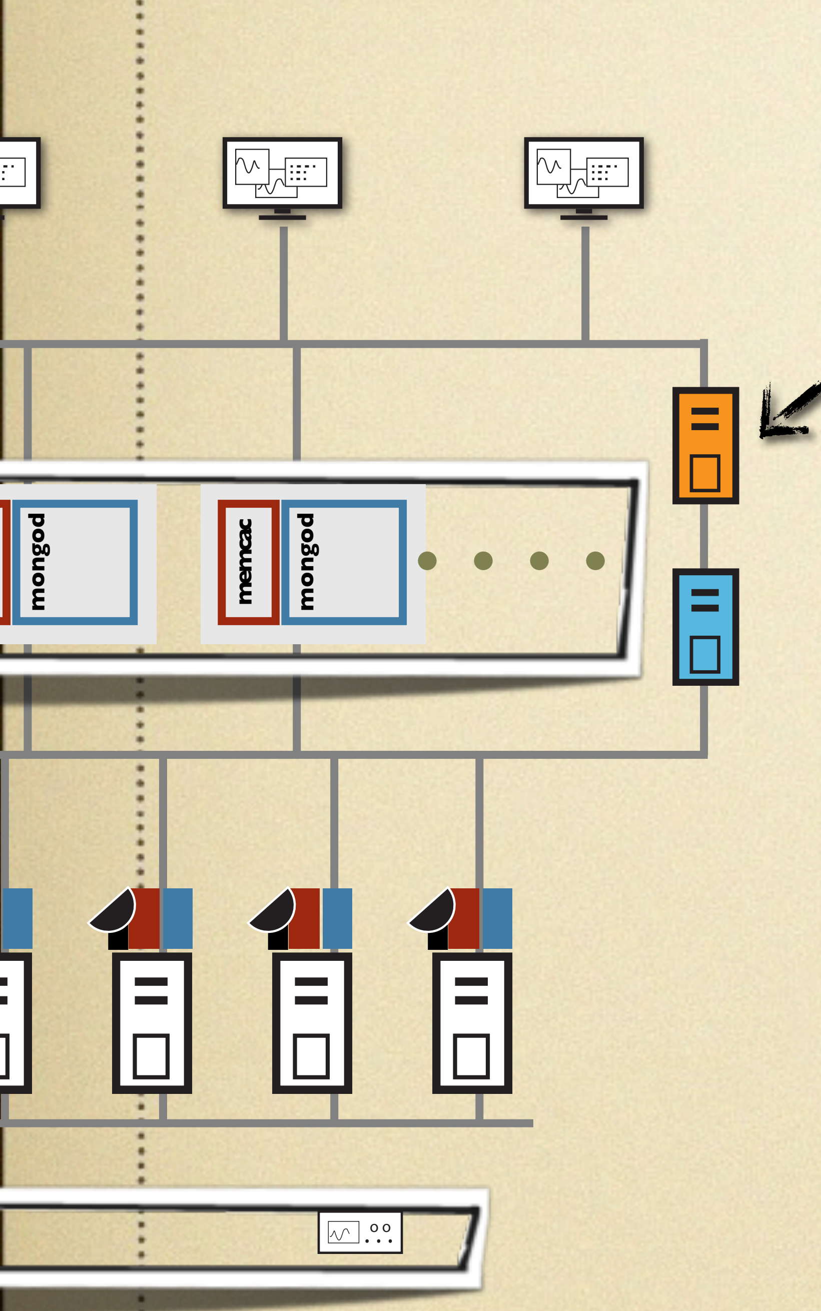
- dynamical *keys* re-distribution allows automatic failover by distributing to other server the load of failed one.
- Scalability is also guaranteed by the same feature

History data



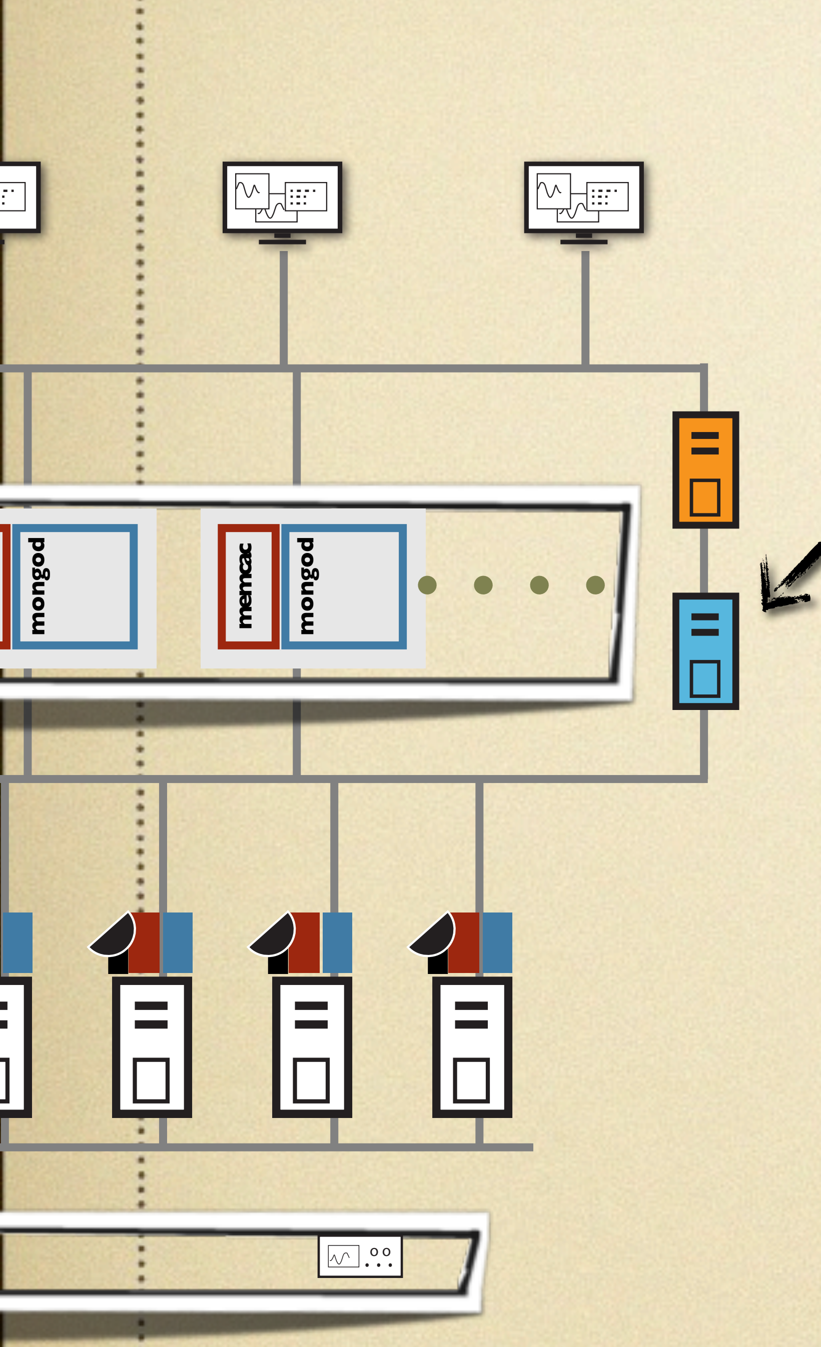
- *key/value* non-relational data-base
- scalability and load balancing by *sharding*
- fast record writing (simpler structure because it has no tables)
- fast queries on primary keys
- (fast) parallel search on cluster nodes

Metadata Server



- CU configuration manager
  (e.g. managing of push data rate)
- Semantic of data (e.g. db records structure)
- Command's list and semantic
- Naming service

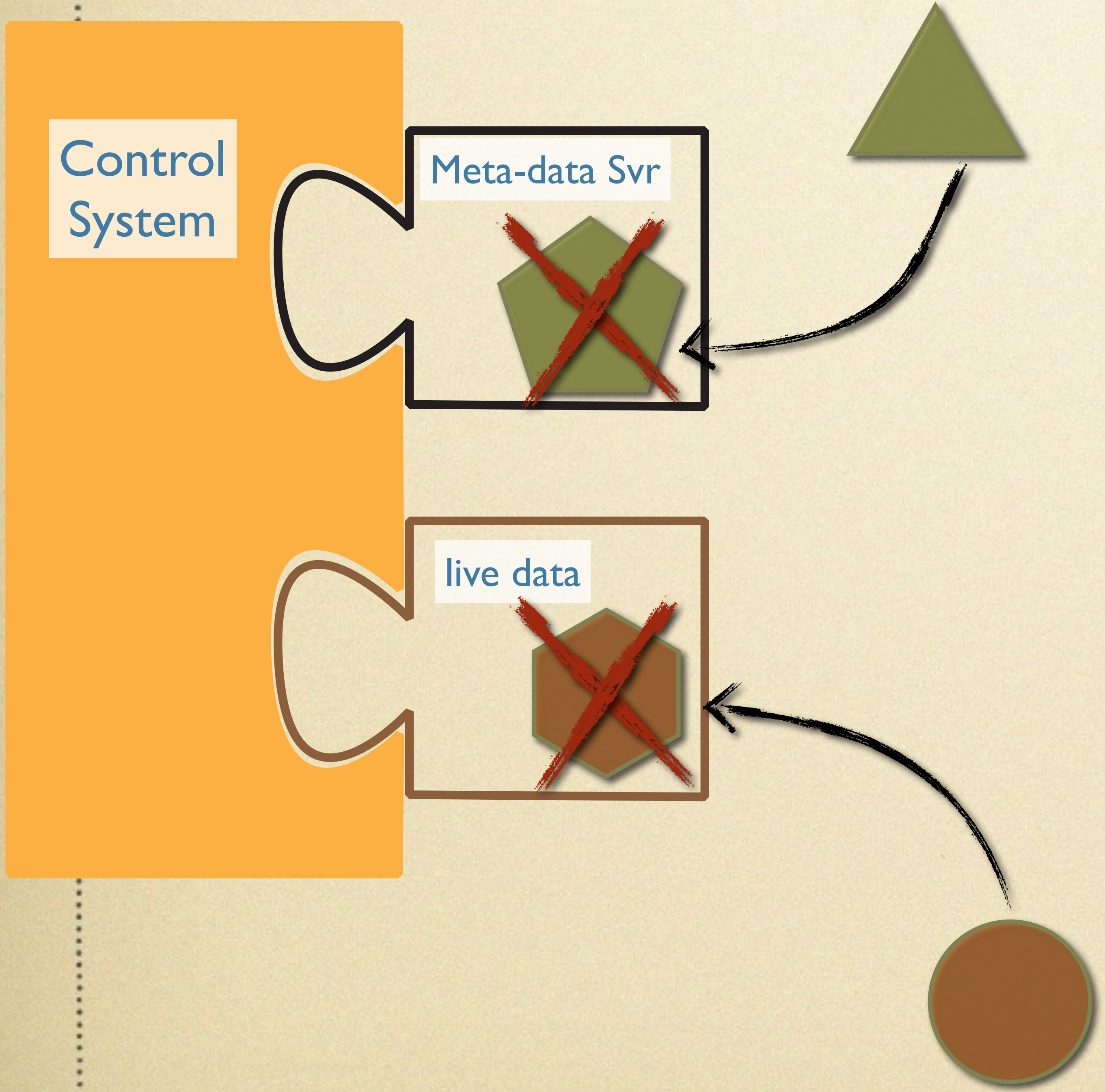
Orchestrator



- Provides middle-layer services, e.g. locking of CUs to prevent command conflicts

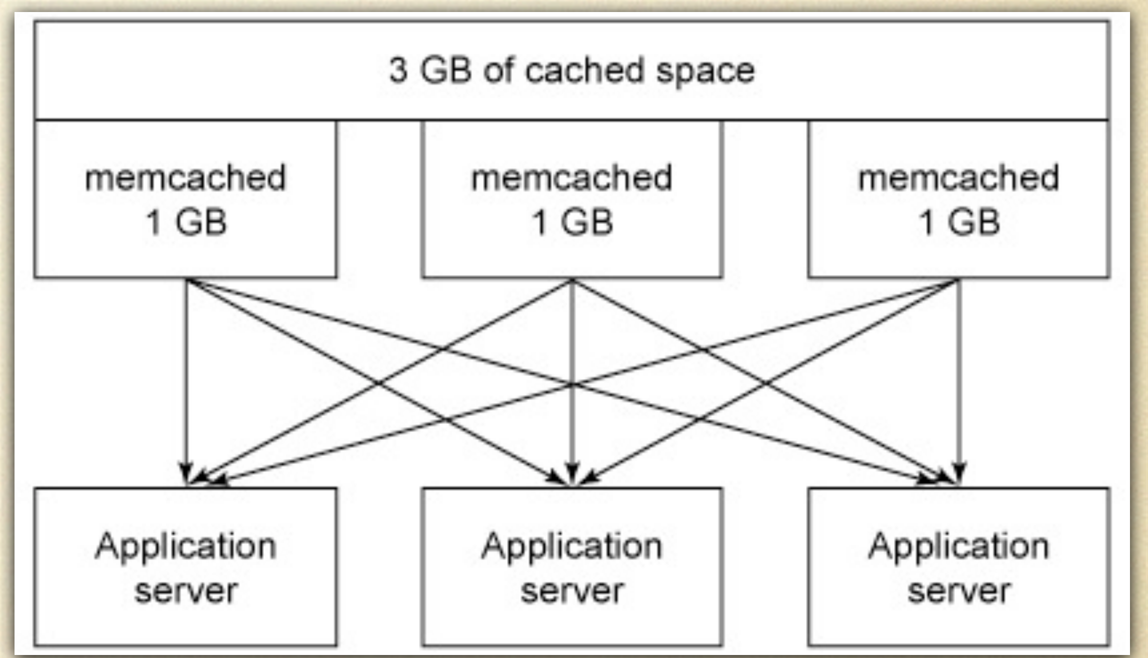
- multi-CUs commands, e.g.
  - global set-points save/restore
  - software feedback
  - on-line measurements
  - ...

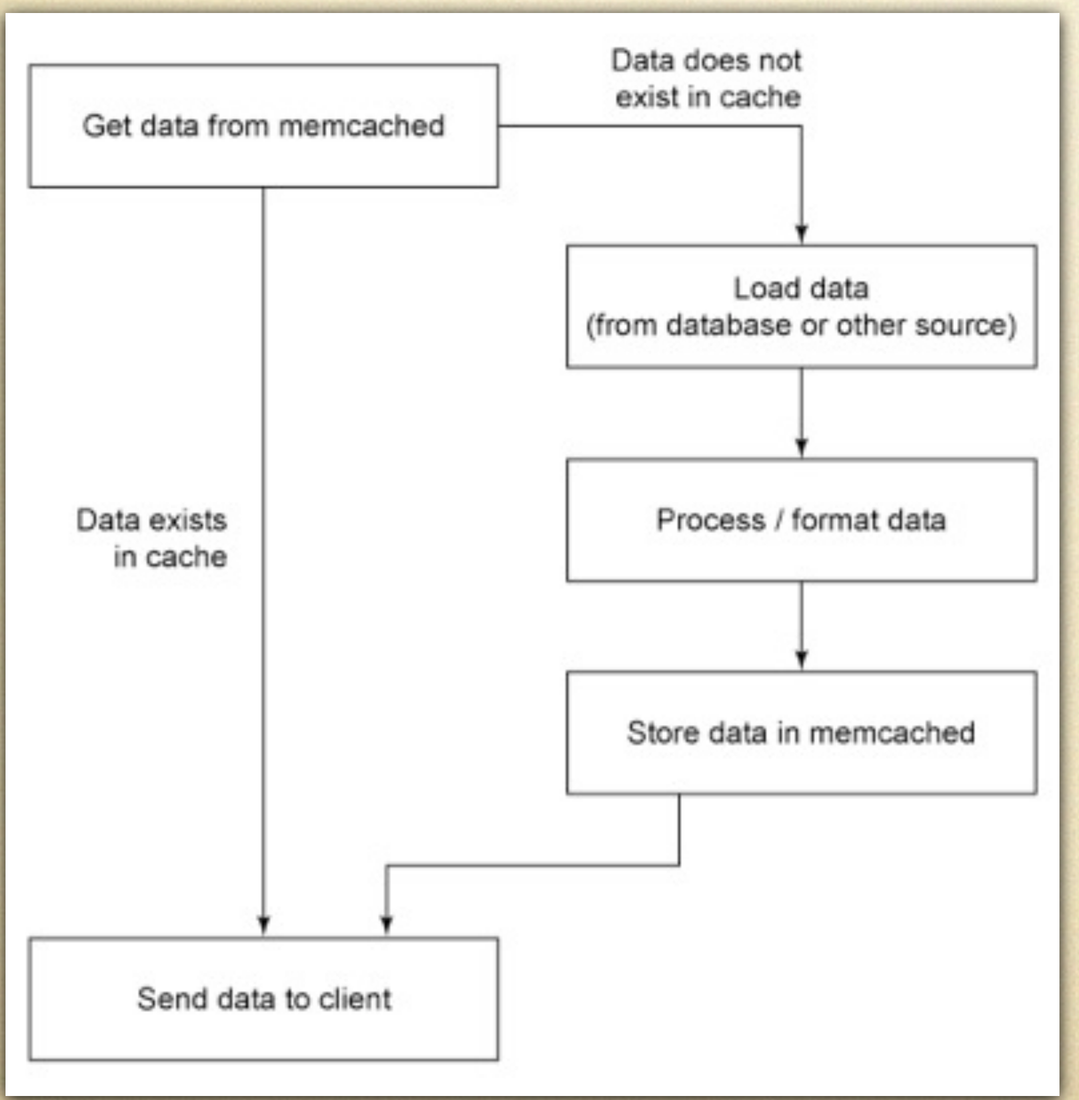
Abstraction of components

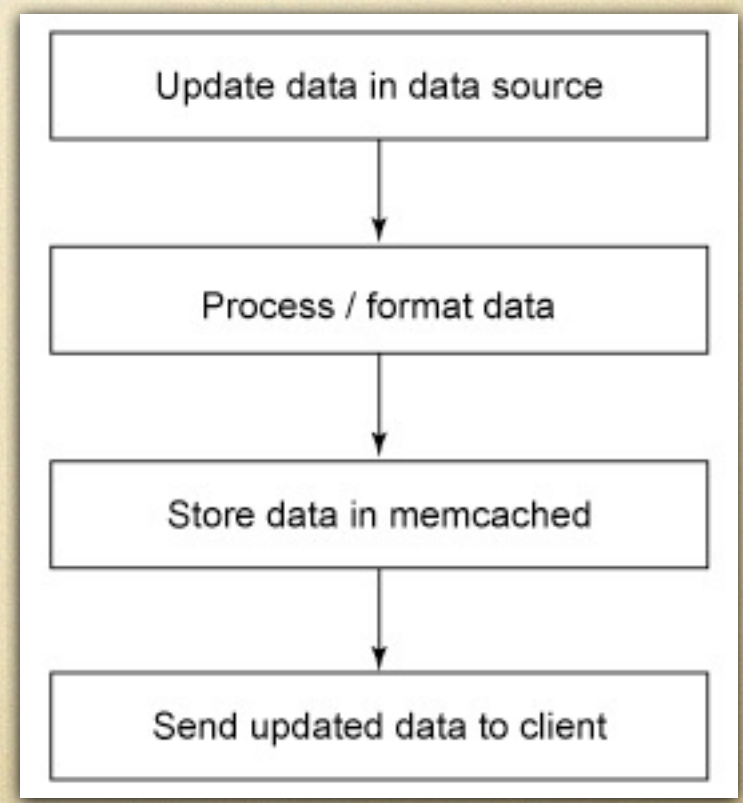


- each service isn’t directly offered to users; glueing and wrapping routines will be developed to provide an high level of abstraction
- updates of core services doesn’t influence the user applications
- higher flexibility in defining API

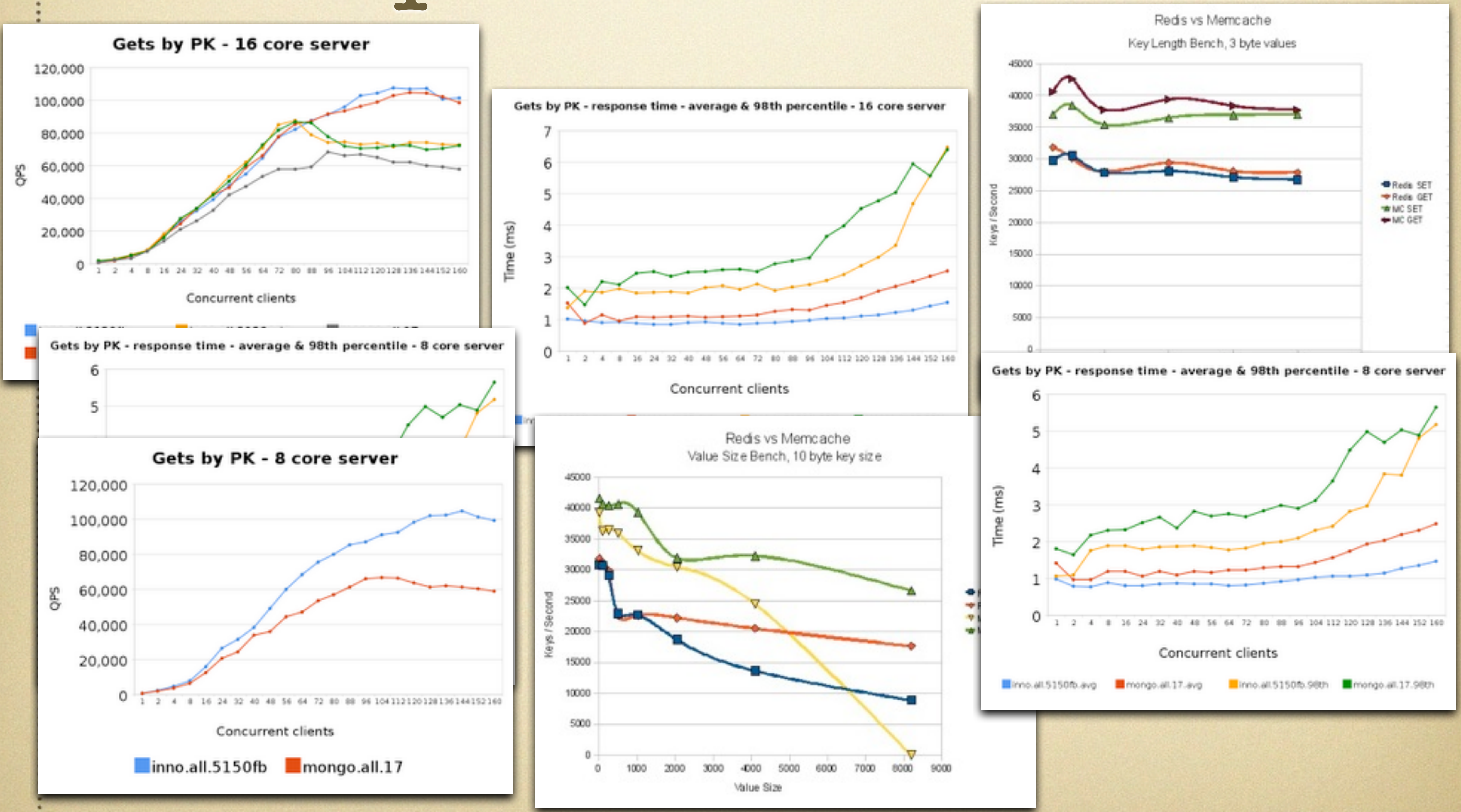
memcached


3 GB of cached space
memcached 1 GB
memcached 1 GB
memcached 1 GB
Application server
Application server
Application server

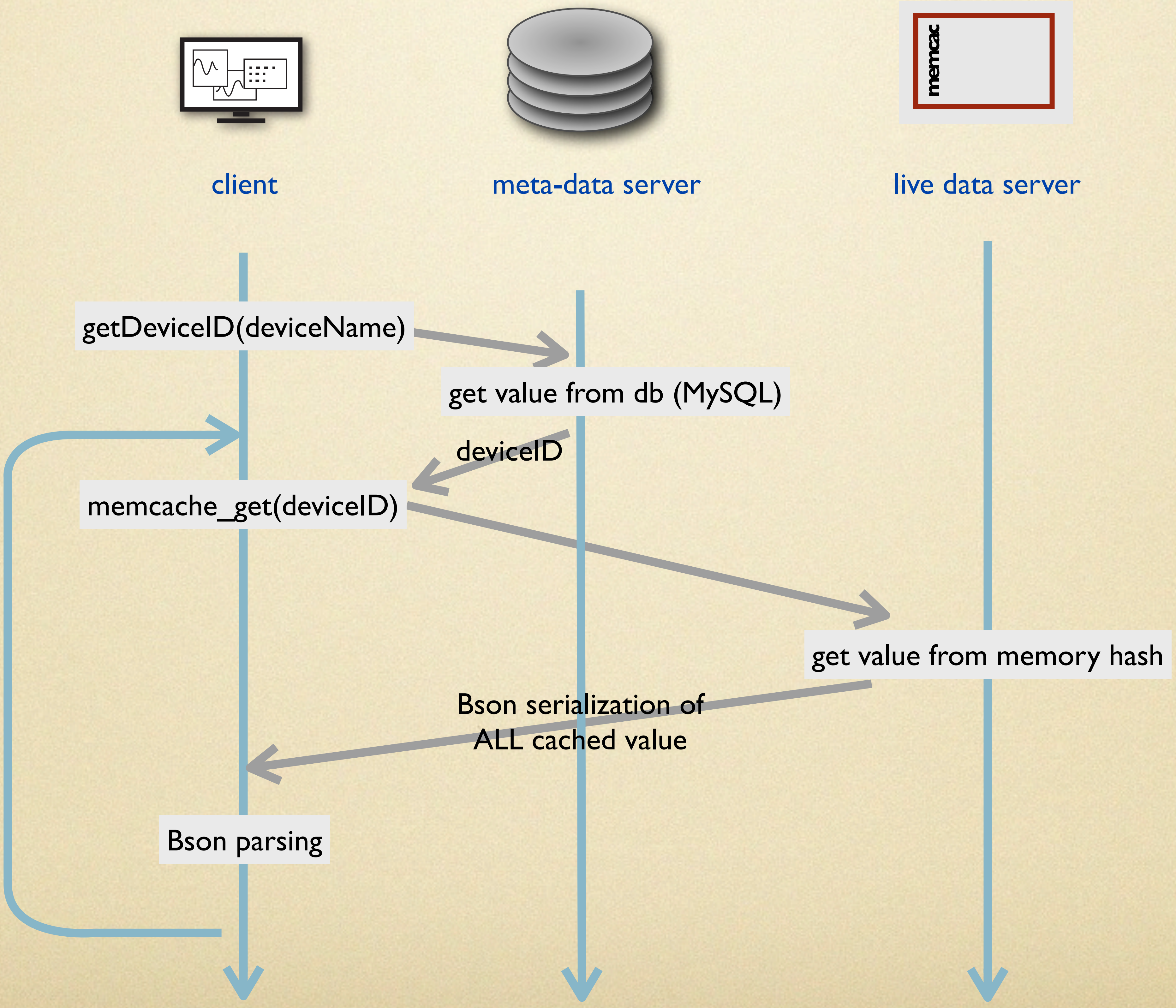

Get data from memcached
Data does not exist in cache
Load data (from database or other source)
Process / format data
Store data in memcached
Data exists in cache
Send data to client


Update data in data source
Process / format data
Store data in memcached
Send updated data to client

memcached performance



pull live data



client

meta-data server

live data server

getDeviceID(deviceName)

get value from db (MySQL)

deviceID

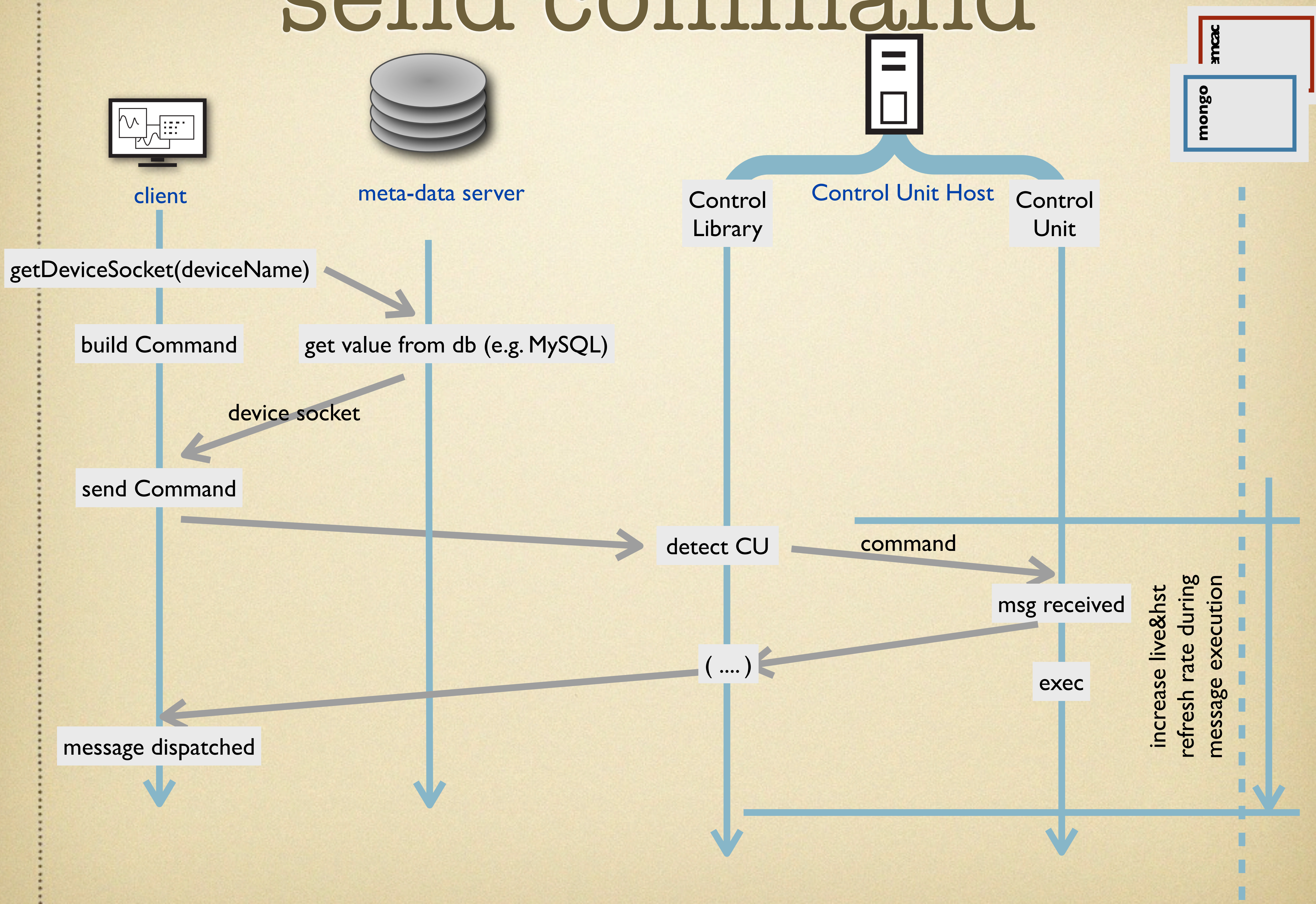
memcache_get(deviceID)

get value from memory hash

Bson serialization of
ALL cached value

Bson parsing

send command



mcac

mongo

client

meta-data server

Control Library

Control Unit Host

Control Unit

getDeviceSocket(deviceName)

build Command

get value from db (e.g. MySQL)

device socket

send Command

detect CU

command

msg received

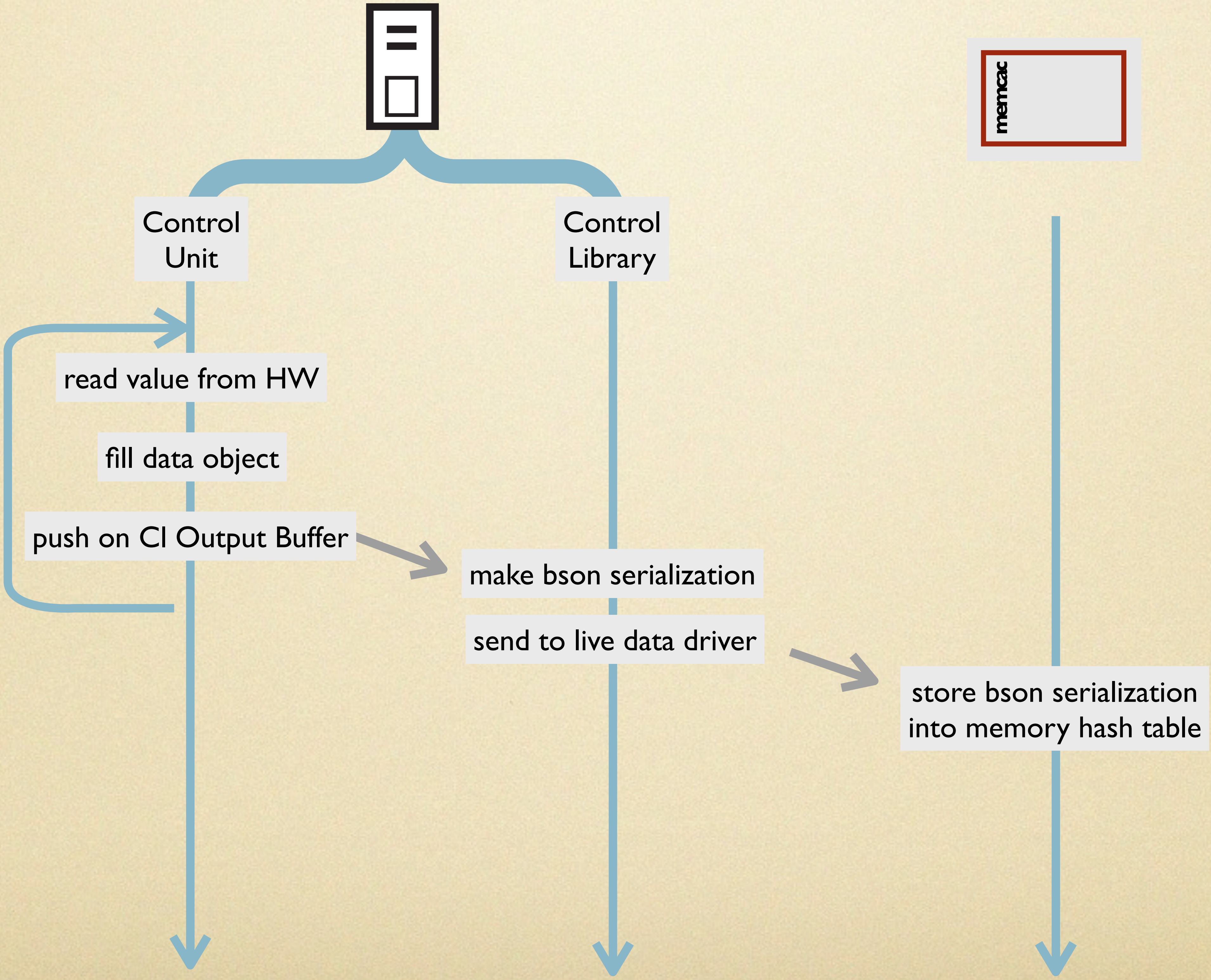
(....)

exec

message dispatched

increase live&hst refresh rate during message execution

push live data



Control Unit

Control Library

read value from HW

fill data object

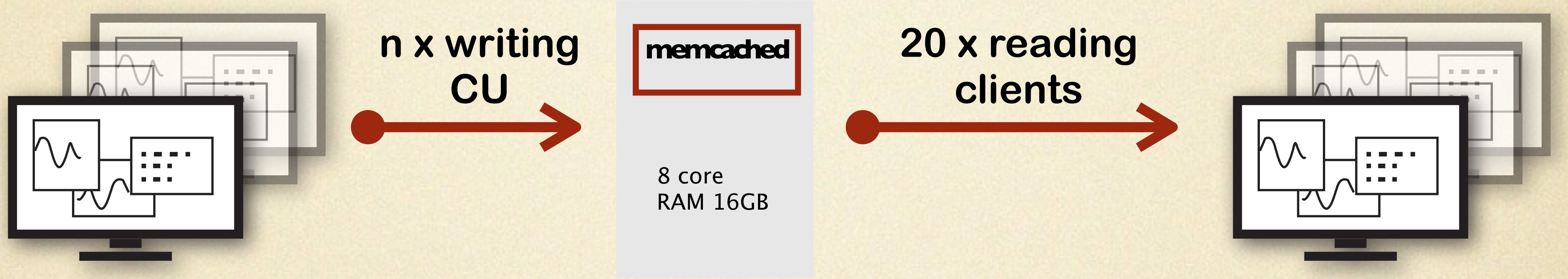
push on CI Output Buffer

make bson serialization

send to live data driver

store bson serialization into memory hash table

test#3.1



n x writing CU

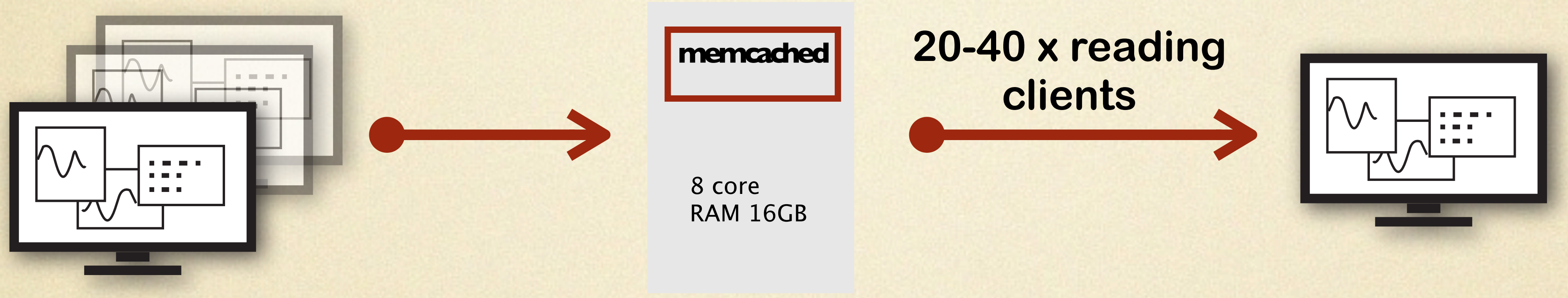
memcached

8 core
RAM 16GB

20 x reading clients

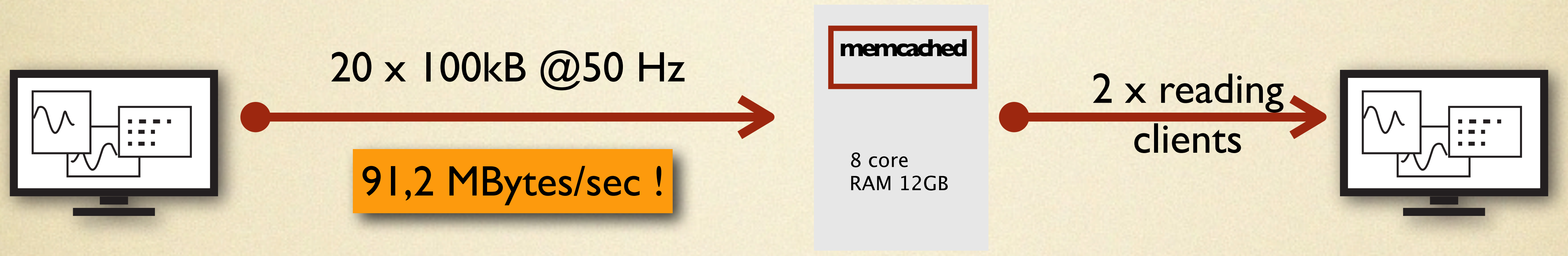
writing every... (msec)	#CU (Write)	#clients (Read)	#servers	#processes/ server	CPU load (%)
20	60	20	1	1	3-5
20	80	20	1	1	4-6
20	80	20	2	1	2-3
50	60	20	1	1	1-3
50	80	20	2	1	0-2
100	60	20	1	1	?
100	80	20	2	1	?

test#3.2



writing every... (msec)	#CU (Write)	#clients (Read)	#servers	#processes/ server	CPU load (%)
20	80	20	1	4 (1 per core)	2-3
20	80	40	1	4 (1 per core)	2-3
		40	1	4 (1 per core)	0

test#4



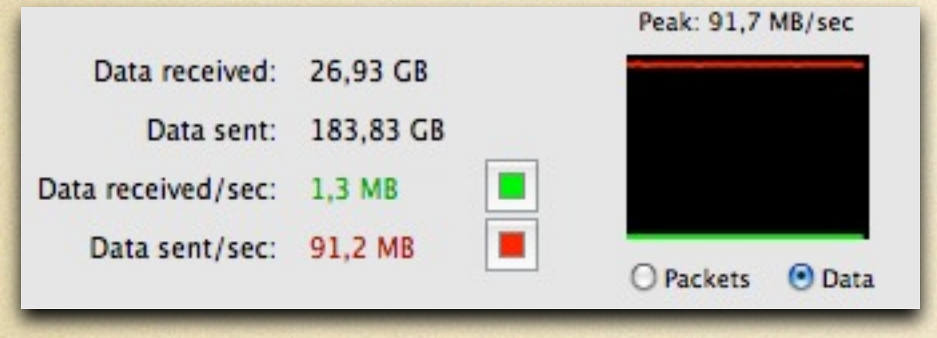
20 x 100kB @50 Hz

91,2 MBytes/sec !

memcached

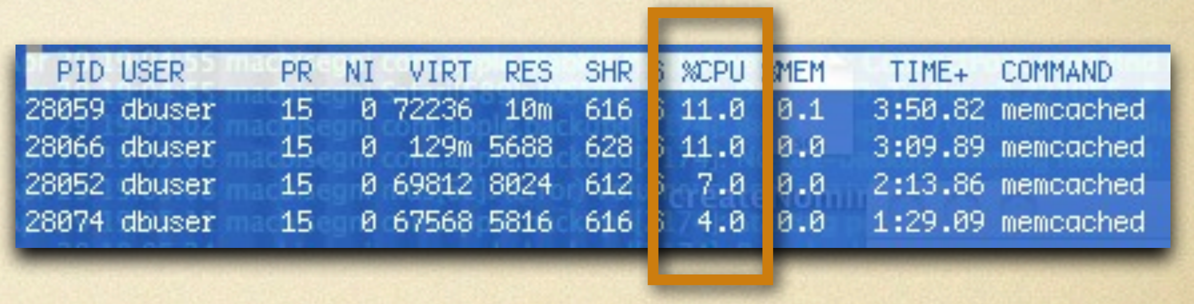
8 core
RAM 12GB

2 x reading clients



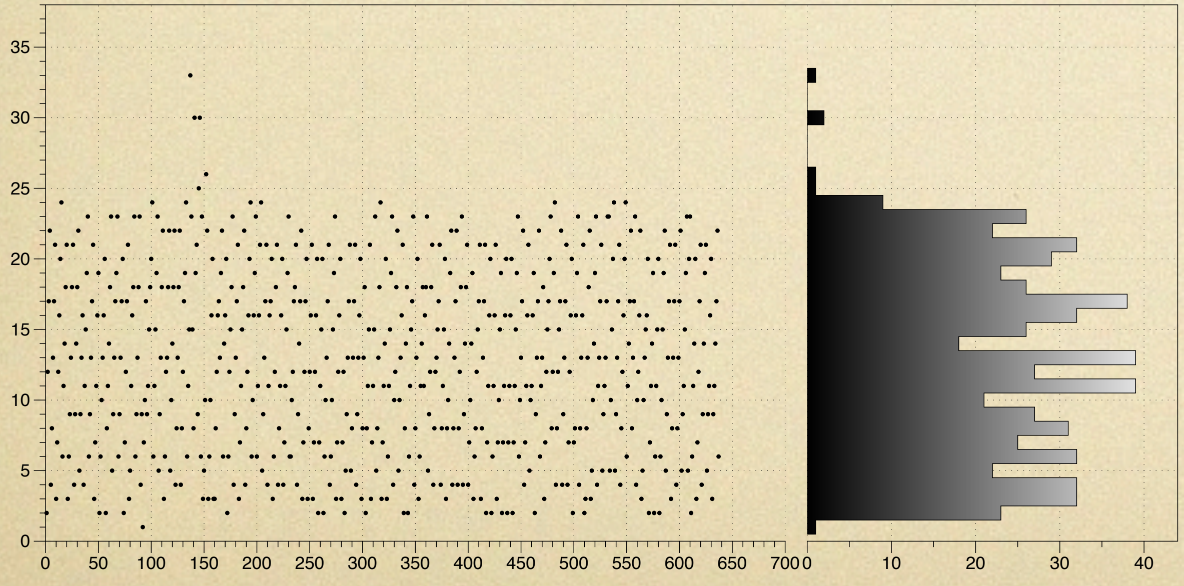
Peak: 91,7 MB/sec

Data received:	26,93 GB	
Data sent:	183,83 GB	
Data received/sec:	1,3 MB	🟩
Data sent/sec:	91,2 MB	🟥

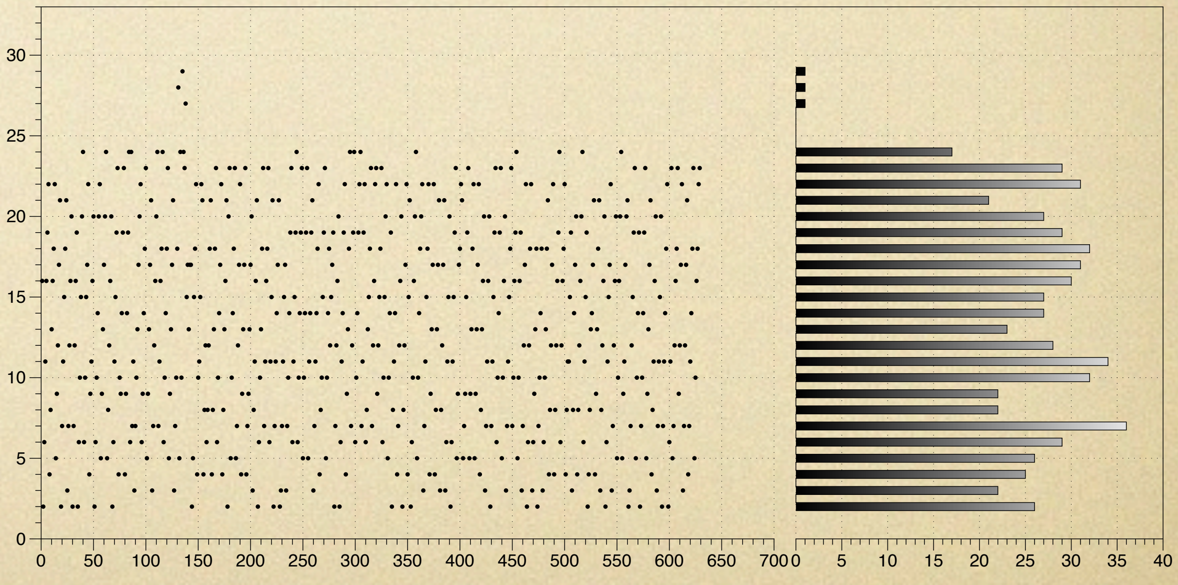


```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
28059 dbuser    15   0 72236  10m  616 S 11.0  0.1   3:50.82 memcached
28066 dbuser    15   0  129m 5688  628 S 11.0  0.0   3:09.89 memcached
28052 dbuser    15   0 69812 8024  612 S  7.0  0.0   2:13.86 memcached
28074 dbuser    15   0 67568 5816  616 S  4.0  0.0   1:29.09 memcached
```

s4_hardware1_w20_m20_buff100000_rd10.log



s4_hardware1_w20_m20_buff100000_rd12.log



conclusions and future plans

motivated by the results of preliminary tests and consistency of the overall design:

- continue R&D for completing system design and continue stress tests of components
- prepare a prototype to be tested on the field (test the system during real-life DAFNE & SPARC operations)
- finalize the project as a candidate for the SuperB Control and DAQ System
- evaluate costs, man power and define time schedule