The SuperB Accelerator Control System: Plans and R&D Status

–

i.e. an attempt to innovate the standar model of control systems

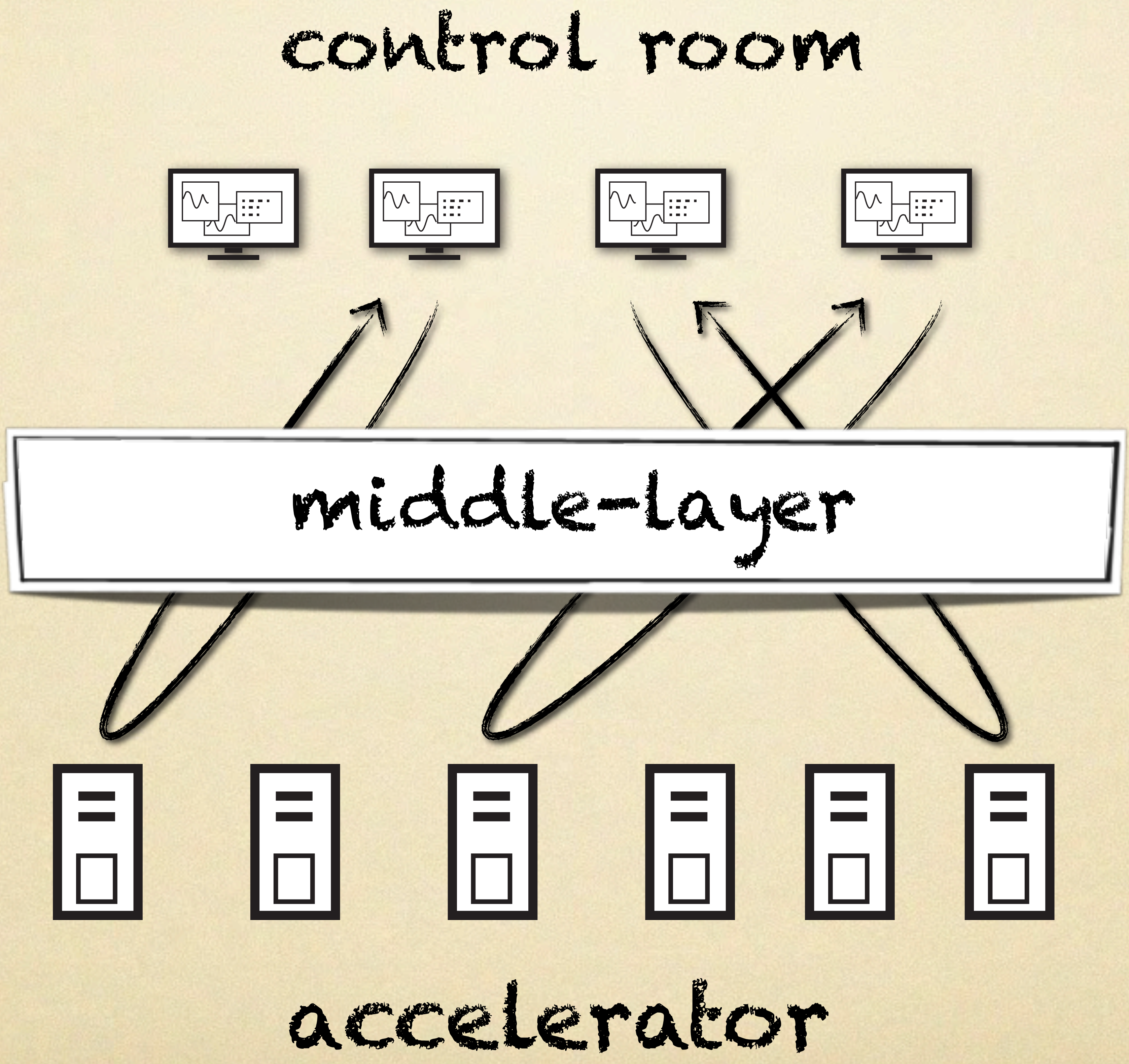
R. Ammendola (INFN Roma TV)
C. Bisegni (INFN-LNF)
S. Calabrò (LAL & INFN-LNF)
L. Catani (INFN Roma TV)
P. Ciuffetti (INFN-LNF)
G. Di Pirro (INFN-LNF)
L. Foggetta (LAL & INFN-LNF)
G. Mazzitelli (INFN-LNF)
A. Stecchi (INFN-LNF)
F. Zani (INFN Roma TV)

SuperB Computing R&D Workshop 2011

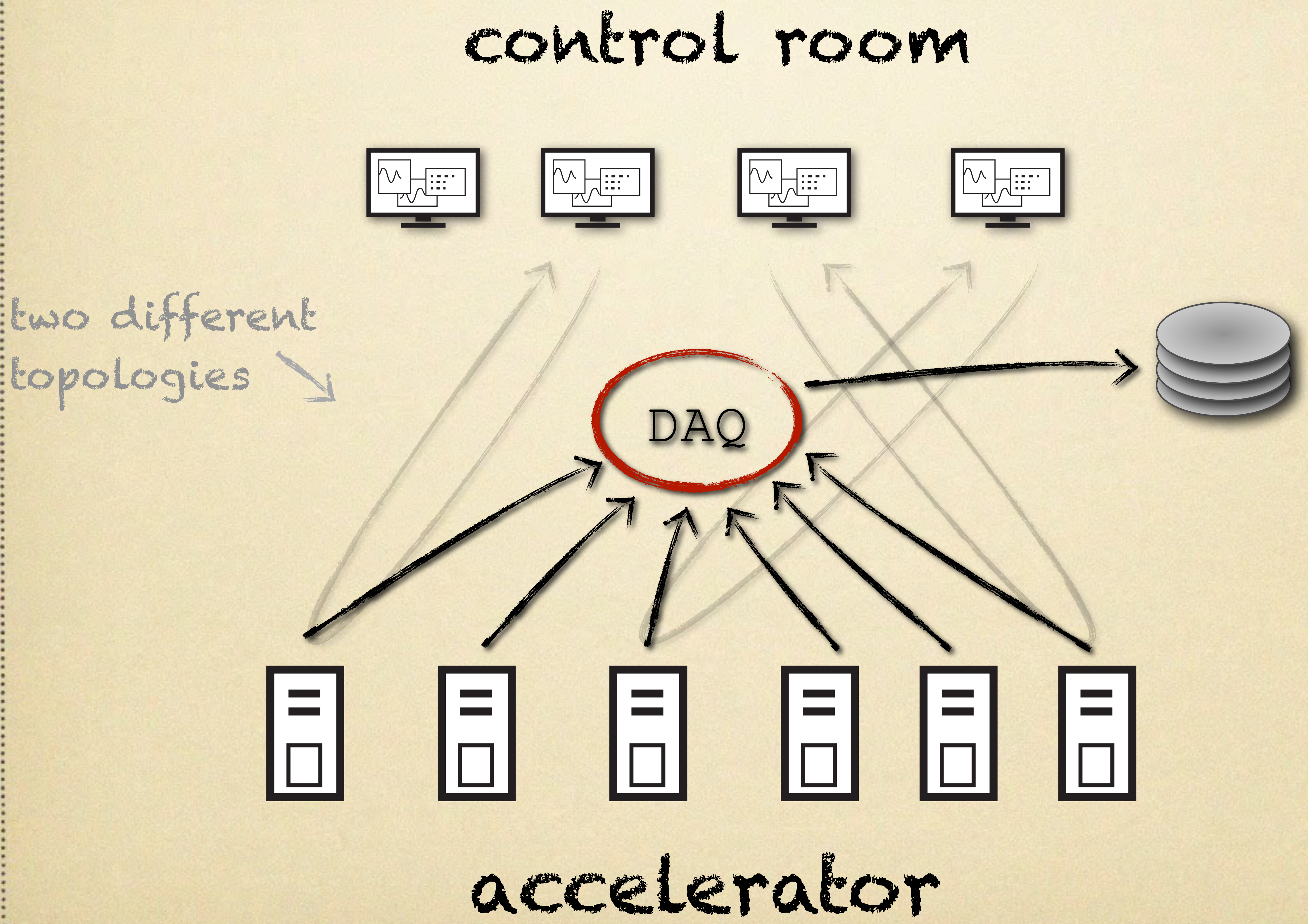
Ferrara, 4-7 July 2011

“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.”

“standard model”



standard model + DAQ



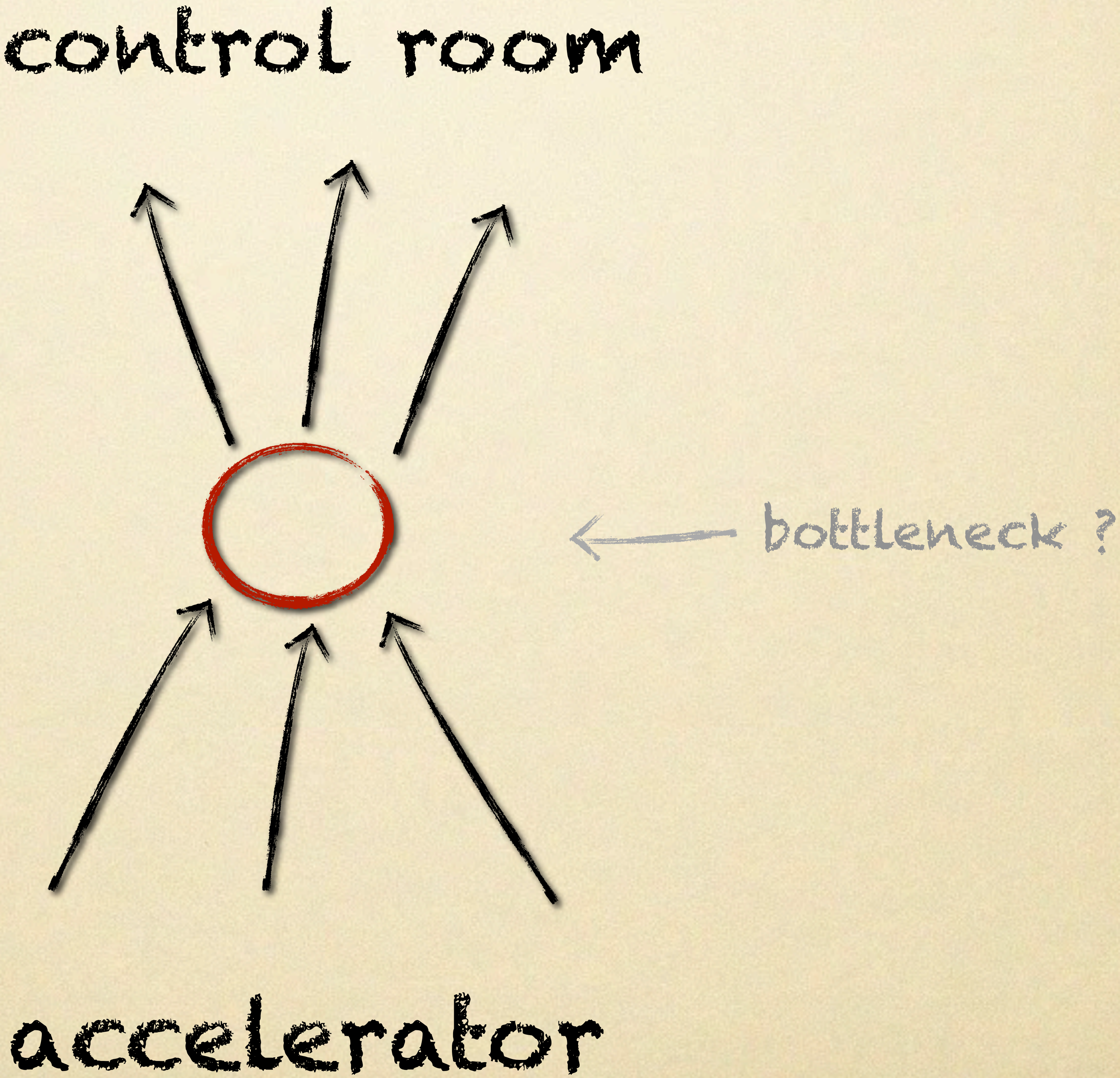
the starting point

- **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

- **extended goal**: *key/value* db looks great, can we use it for **live data** ?
  - data retrieving still slow
- use distributed caching instead
  - same topology, same data structure, same scalability

the new paradigm

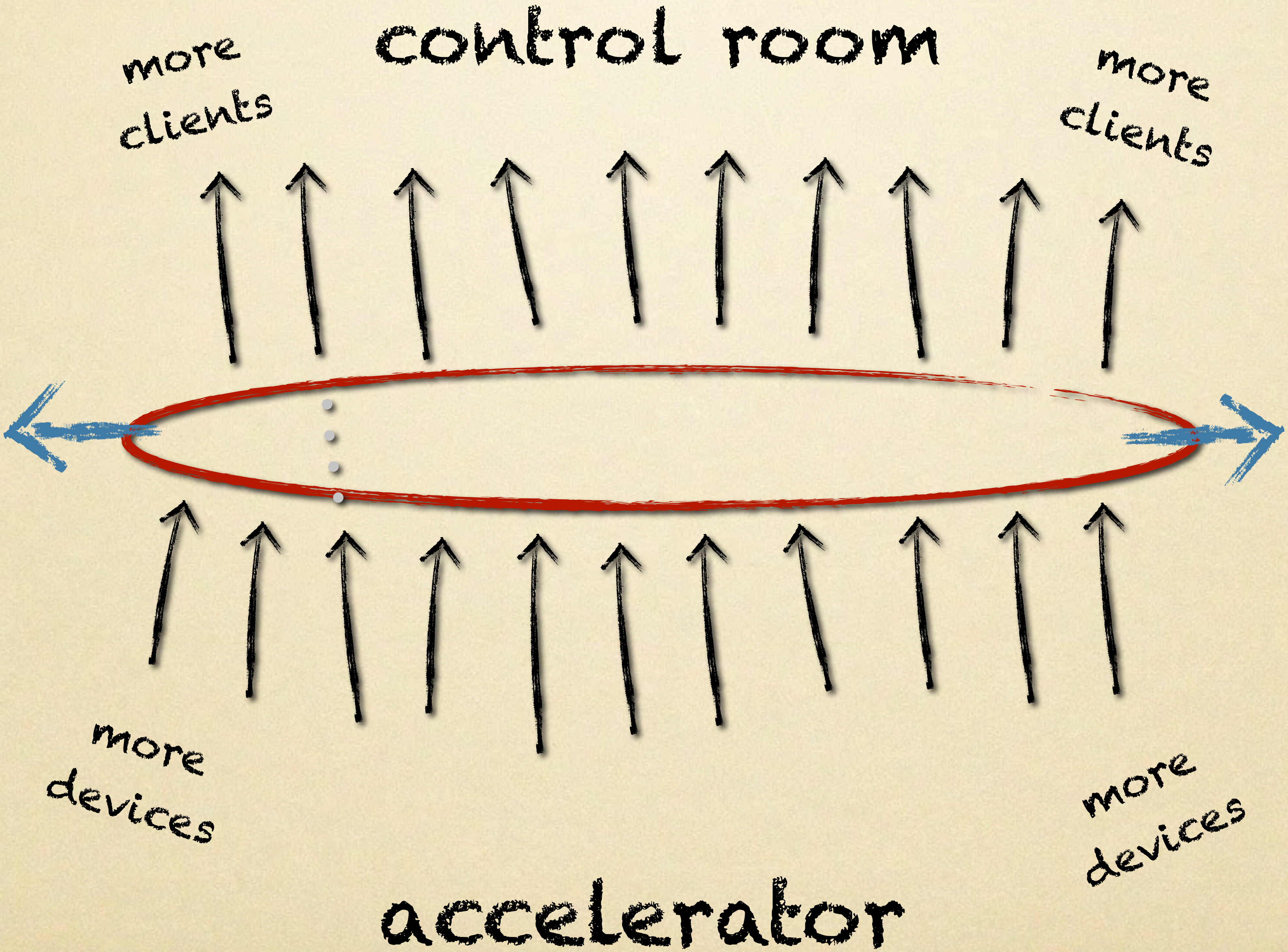


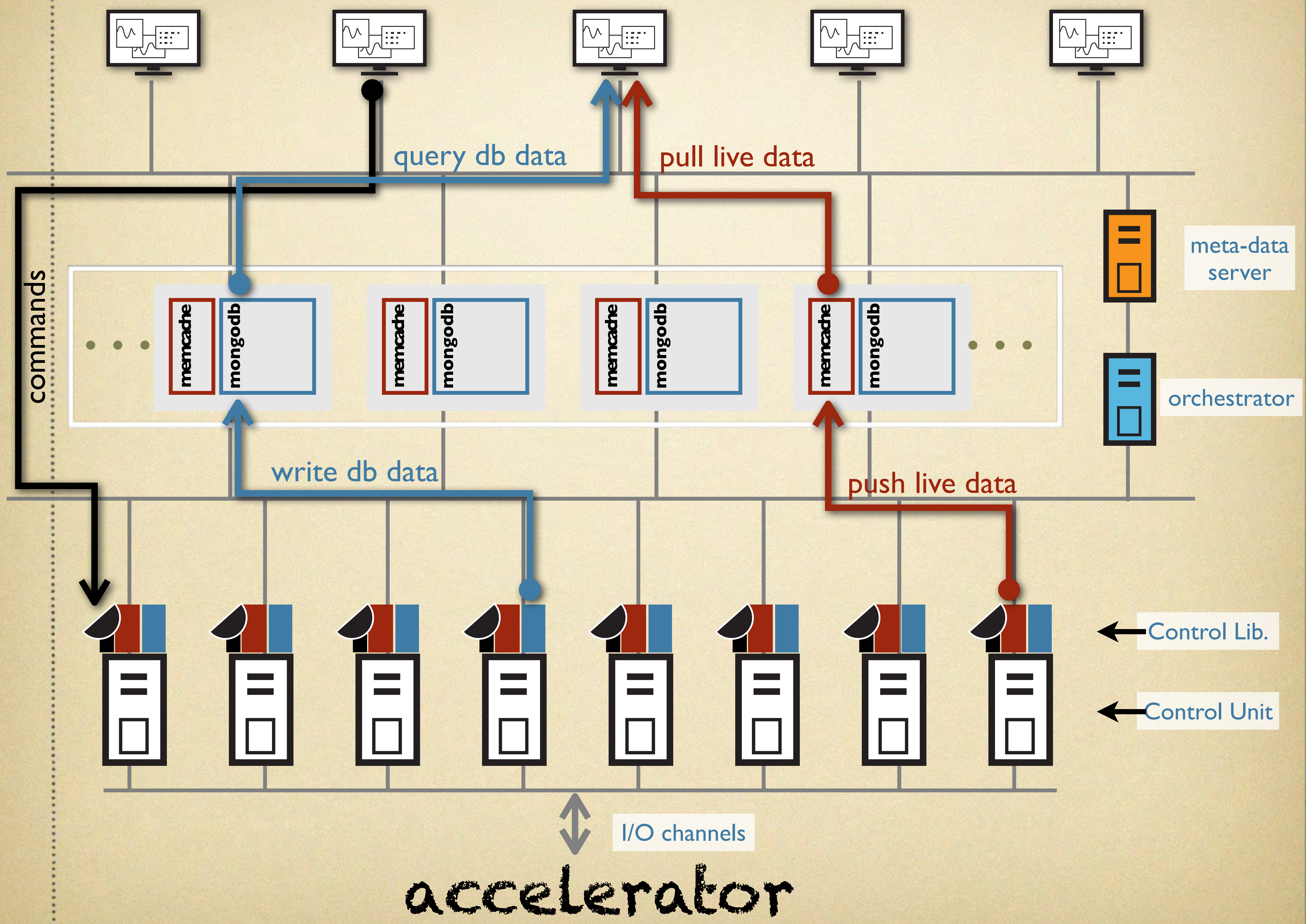
control room

bottleneck ?

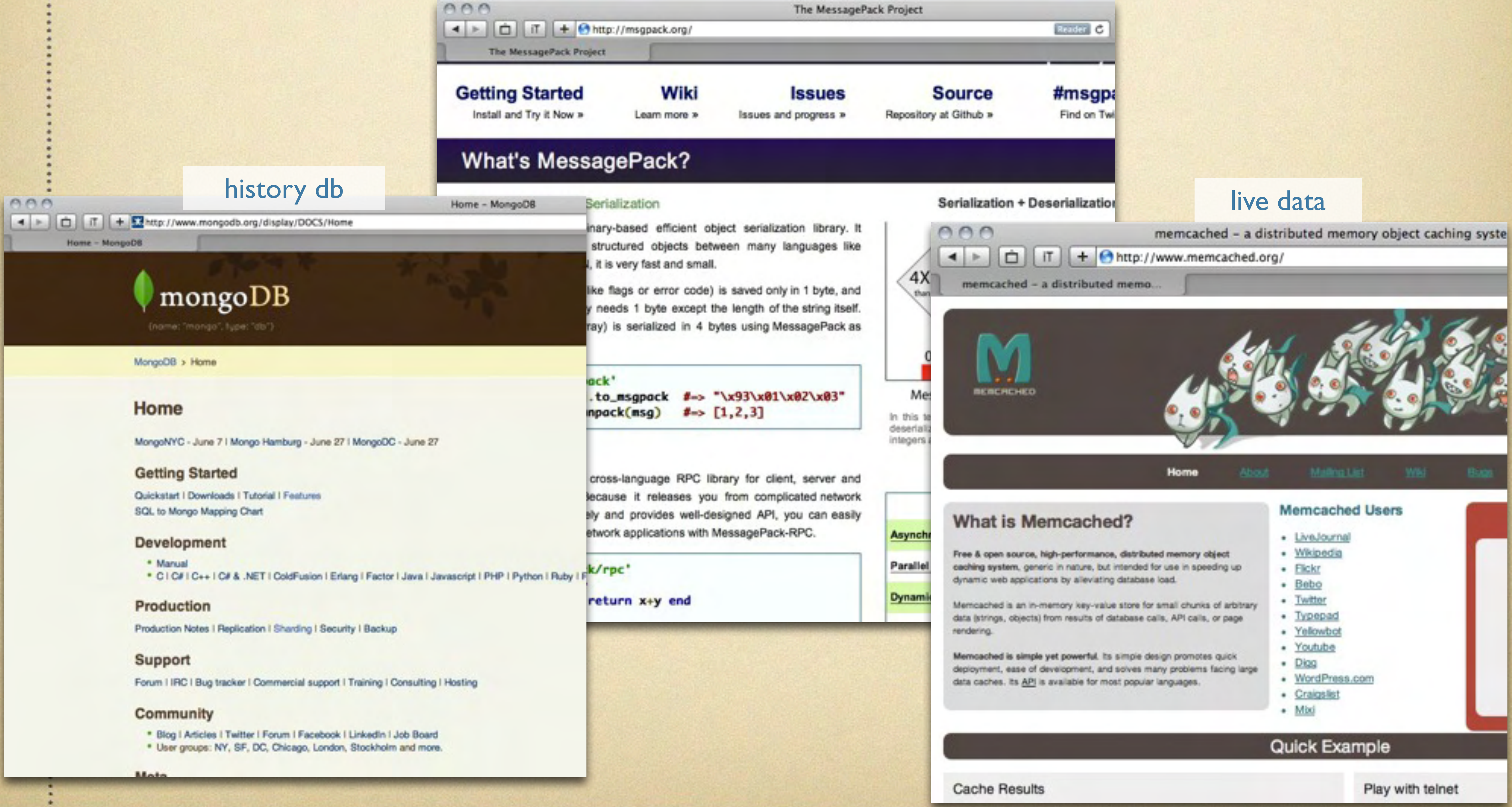
accelerator

the new paradigm:
(scalability eliminates bottleneck)




query db data
pull live data
commands
memcache
mongodb
memcache
mongodb
memcache
mongodb
memcache
mongodb
meta-data server
orchestrator
write db data
push live data
Control Lib.
Control Unit
I/O channels
accelerator

core services candidates



history db

live data

The MessagePack Project

http://msgpack.org/

The MessagePack Project

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 because it releases you from complicated network ly and provides well-designed API, you can easily etwork applications with MessagePack-RPC.

```
k/rpc'

return x+y end
```

Serialization + Deserializatio

Home - MongoDB

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

Home - MongoDB

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 - a distributed memo...

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

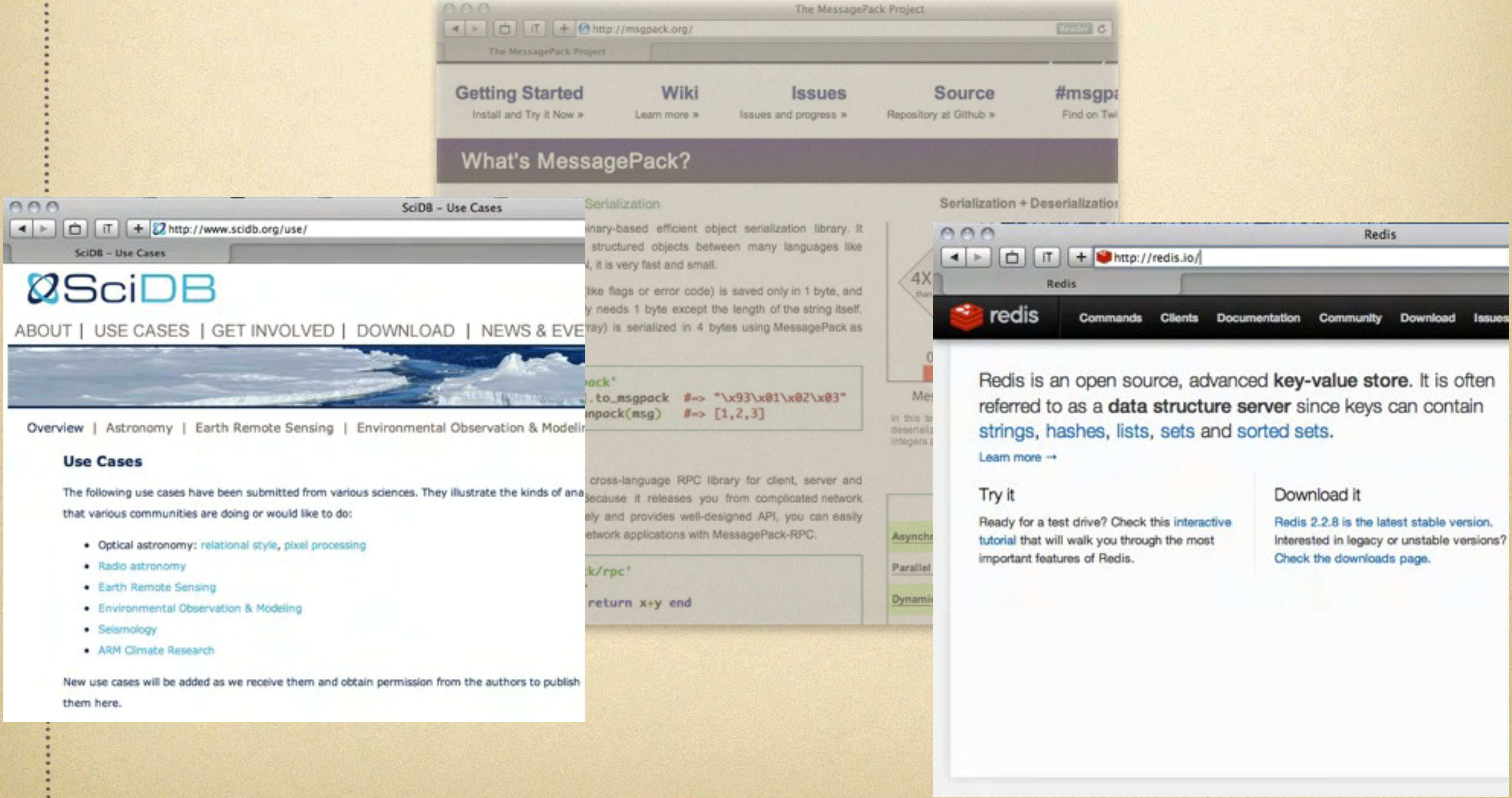
- LiveJournal
- Wikipedia
- Flickr
- Bebo
- Twitter
- Typepad
- Yellowbot
- Youtube
- Digg
- WordPress.com
- Craigslist
- Mixi

Quick Example

Cache Results

Play with telnet

core services alternatives



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

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 →

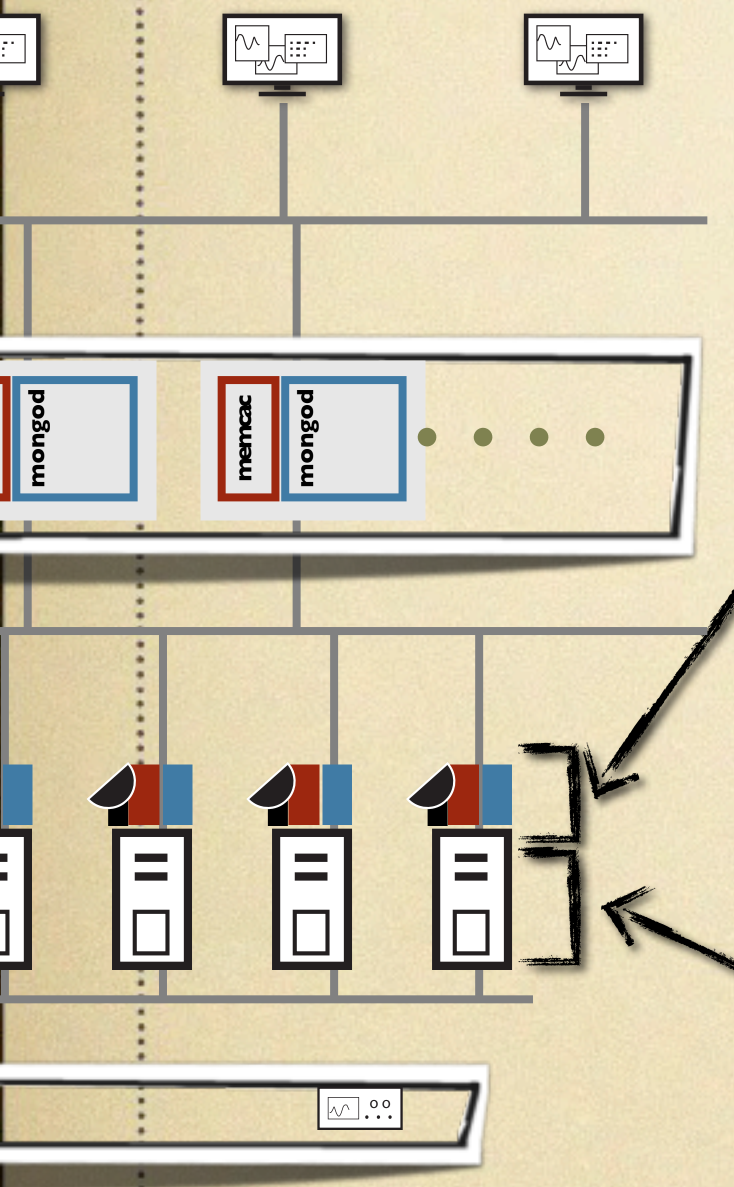
Try it

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.

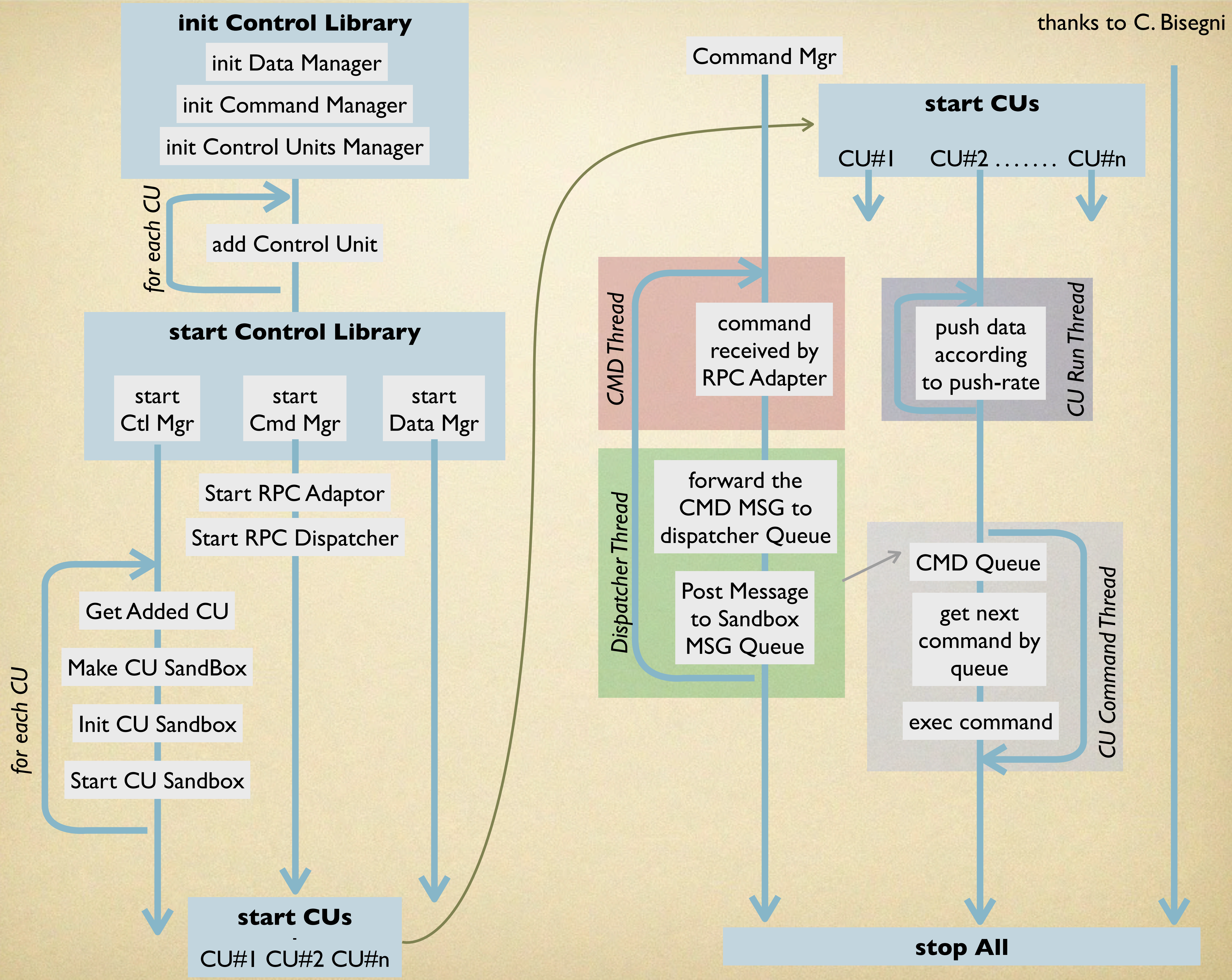
Control Library e Control Unit



multi-threads

- The Control Library (CL) is the set of functions needed to hw-drivers developers for communicating with the CS. It allows:
  - managing configurations
  - writing data to Live and History
  - Commands dispatching & handling

- Each Control Unit employ the CL to export to the CS an accelerator component or a family thereof.



thanks to C. Bisegni

init Control Library

init Data Manager

init Command Manager

init Control Units Manager

for each CU

add Control Unit

start Control Library

start Ctl Mgr

start Cmd Mgr

start Data Mgr

Start RPC Adaptor

Start RPC Dispatcher

for each CU

Get Added CU

Make CU SandBox

Init CU Sandbox

Start CU Sandbox

start CUs

CU#1 CU#2 CU#n

Command Mgr

start CUs

CU#1 CU#2 .... CU#n

CMD Thread

command received by RPC Adapter

Dispatcher Thread

forward the CMD MSG to dispatcher Queue

Post Message to Sandbox MSG Queue

CU Run Thread

push data according to push-rate

CU Command Thread

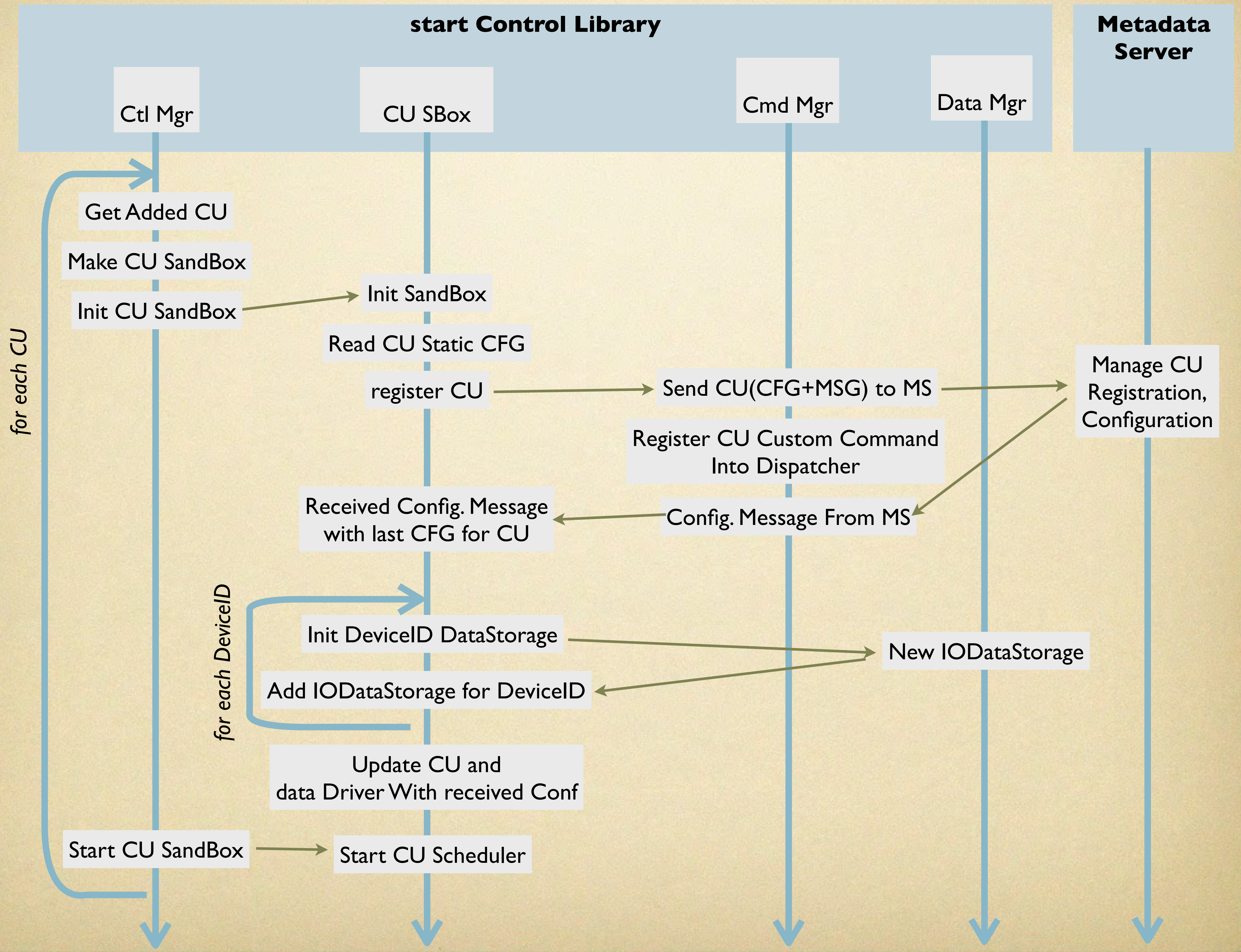
CMD Queue

get next command by queue

exec command

stop All

thanks to C. Bisegni



start Control Library

Metadata Server

Ctl Mgr

CU SBox

Cmd Mgr

Data Mgr

for each CU

Get Added CU

Make CU SandBox

Init CU SandBox

Init SandBox

Read CU Static CFG

register CU

Send CU(CFG+MSG) to MS

Manage CU Registration, Configuration

Register CU Custom Command Into Dispatcher

Config. Message From MS

Received Config. Message with last CFG for CU

for each DeviceID

Init DeviceID DataStorage

New IODataStorage

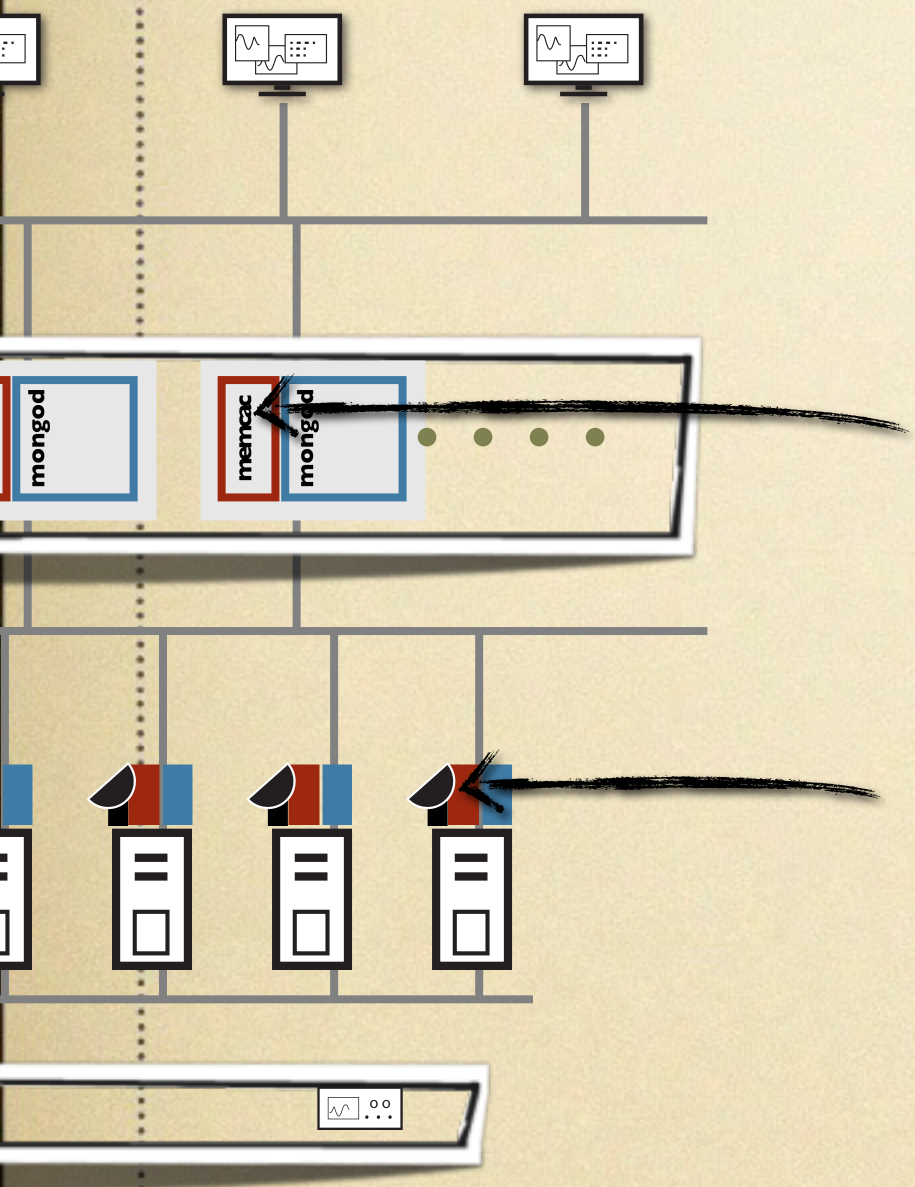
Add IODataStorage for DeviceID

Update CU and data Driver With received Conf

Start CU SandBox

Start CU Scheduler

Live data



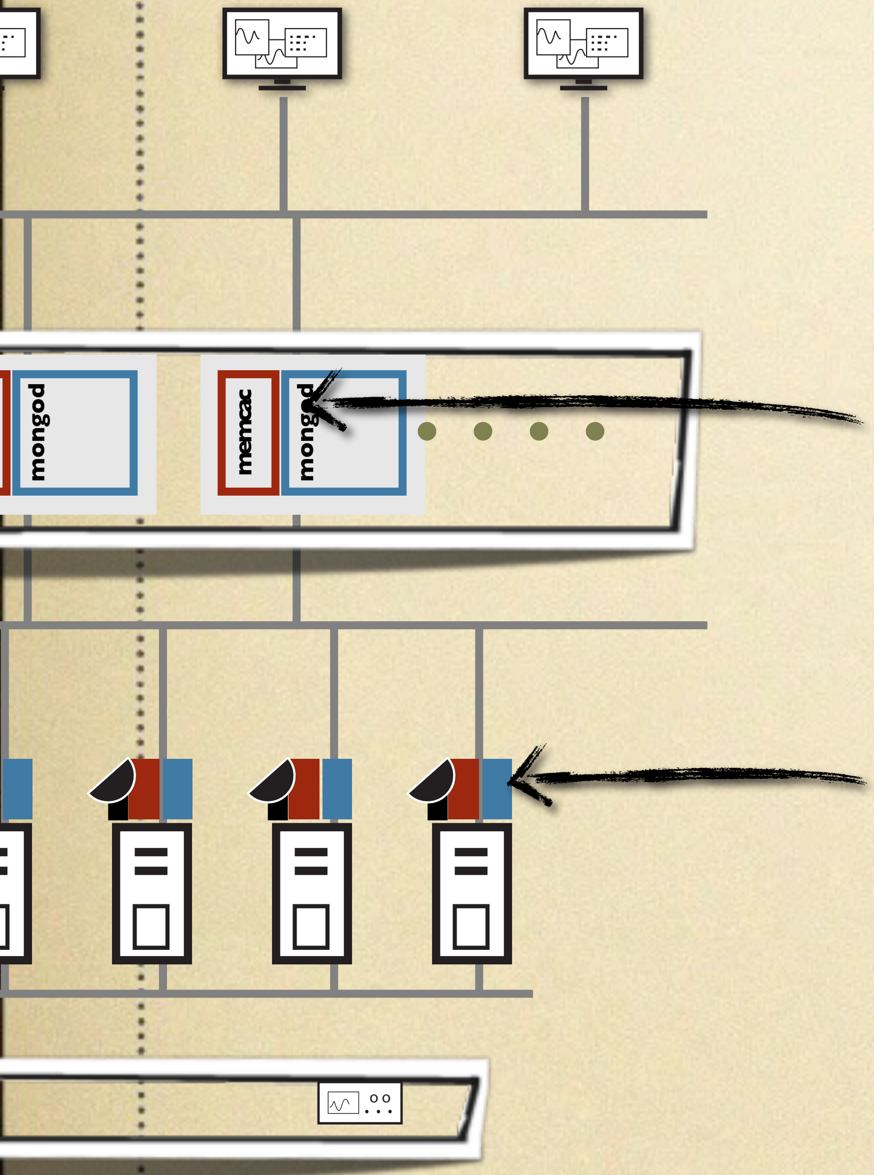
- Allows high-performance caching of data produced by any component managed by CS.

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

r/w of a value subset now available !

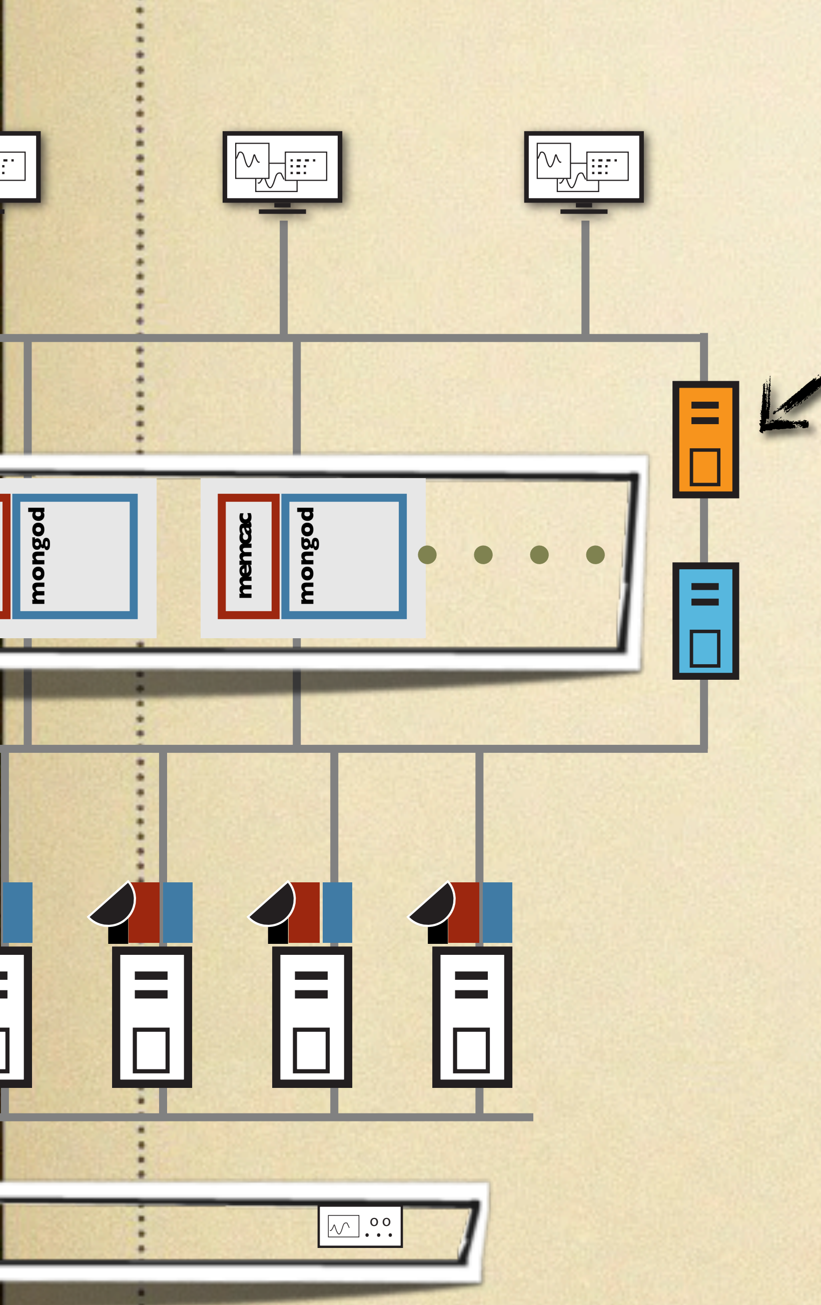
- dynamical *keys* re-distribution allows automatic failover by redirecting to other servers the load of failed one.
- Scalability is also guaranteed by the same feature

History data



- *key/value* non-relational database
- scalability and load balancing by *sharding*
- fast record writing (simpler structure because it doesn't use tables)
- fast queries on primary keys
- (fast) parallel search on cluster nodes

Metadata Server



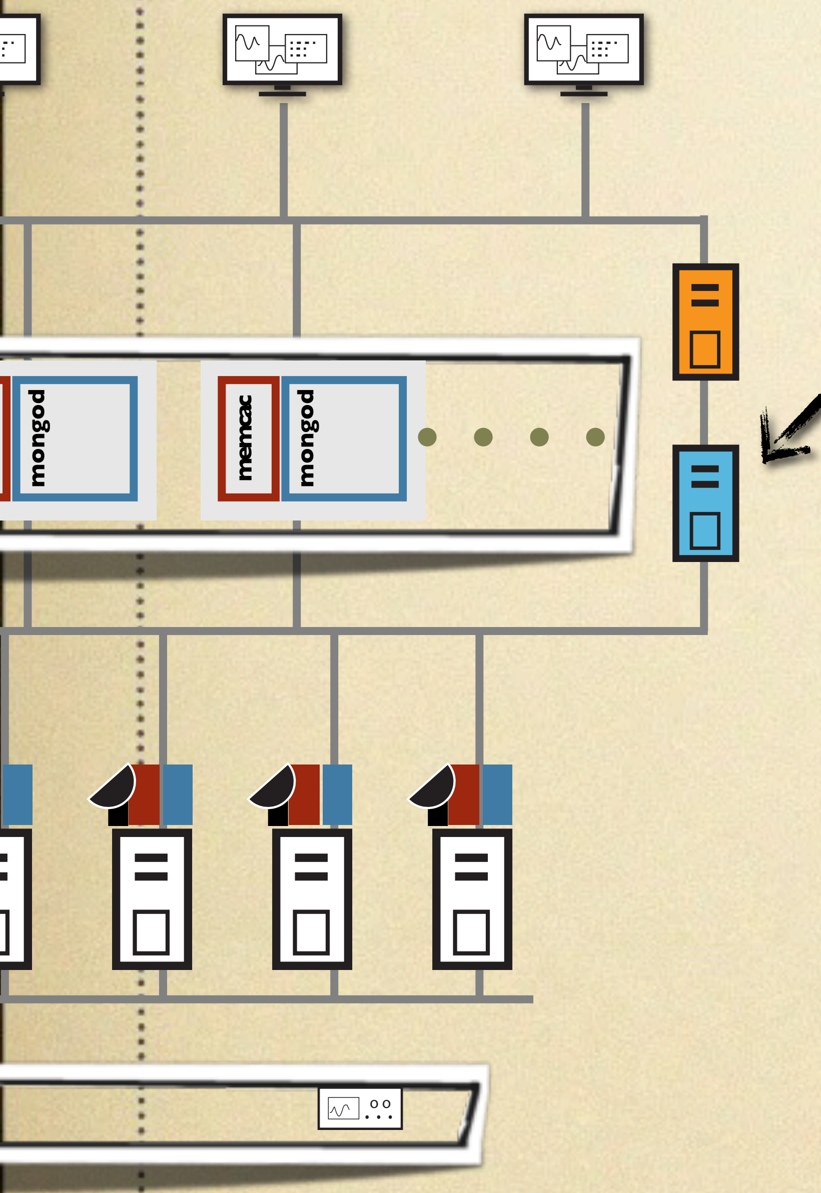
- CU configuration manager
  (e.g. managing of pushing data rate)

- Semantic of data (e.g. db records structure)

- Command's list and semantic

- Naming service

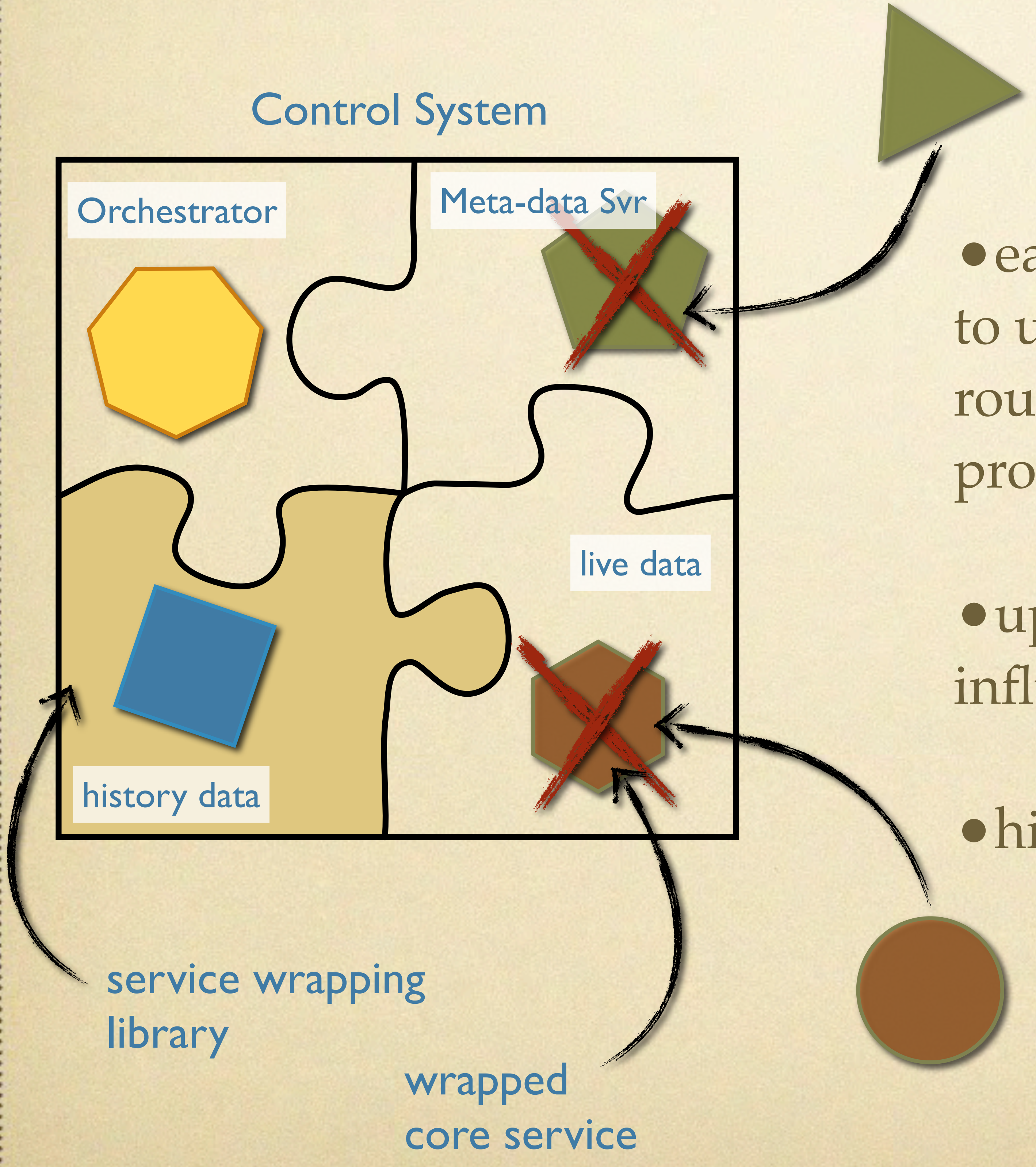
Orchestrator



- 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

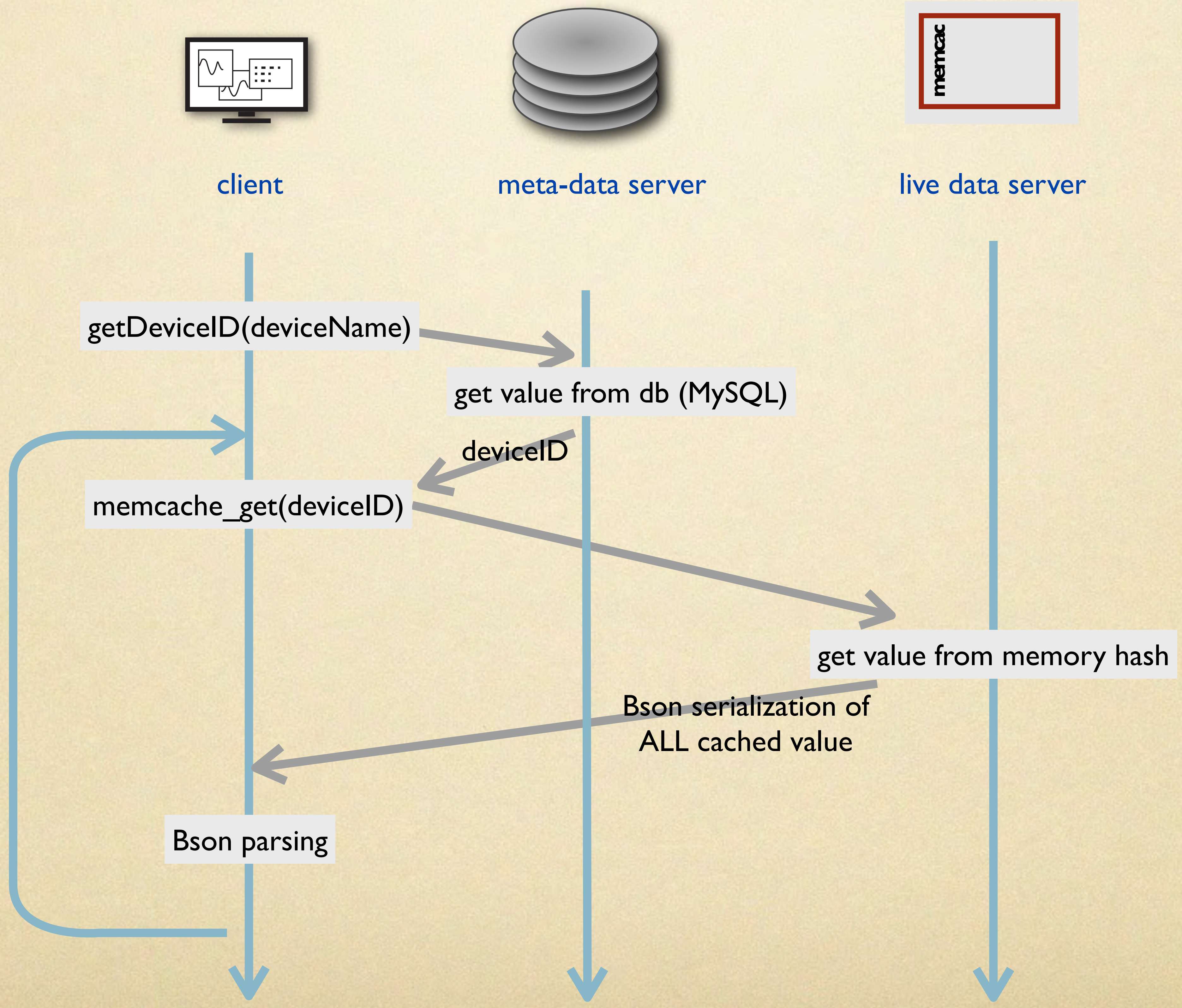


- each service isn't directly offered to users; glueing and wrapping routines will be developed to provide an high level of abstraction

- updates of core services doesn't influence the user applications

- higher flexibility in defining API

pull live data



client

meta-data server

live data server

getDeviceID(deviceName)

get value from db (MySQL)

deviceID

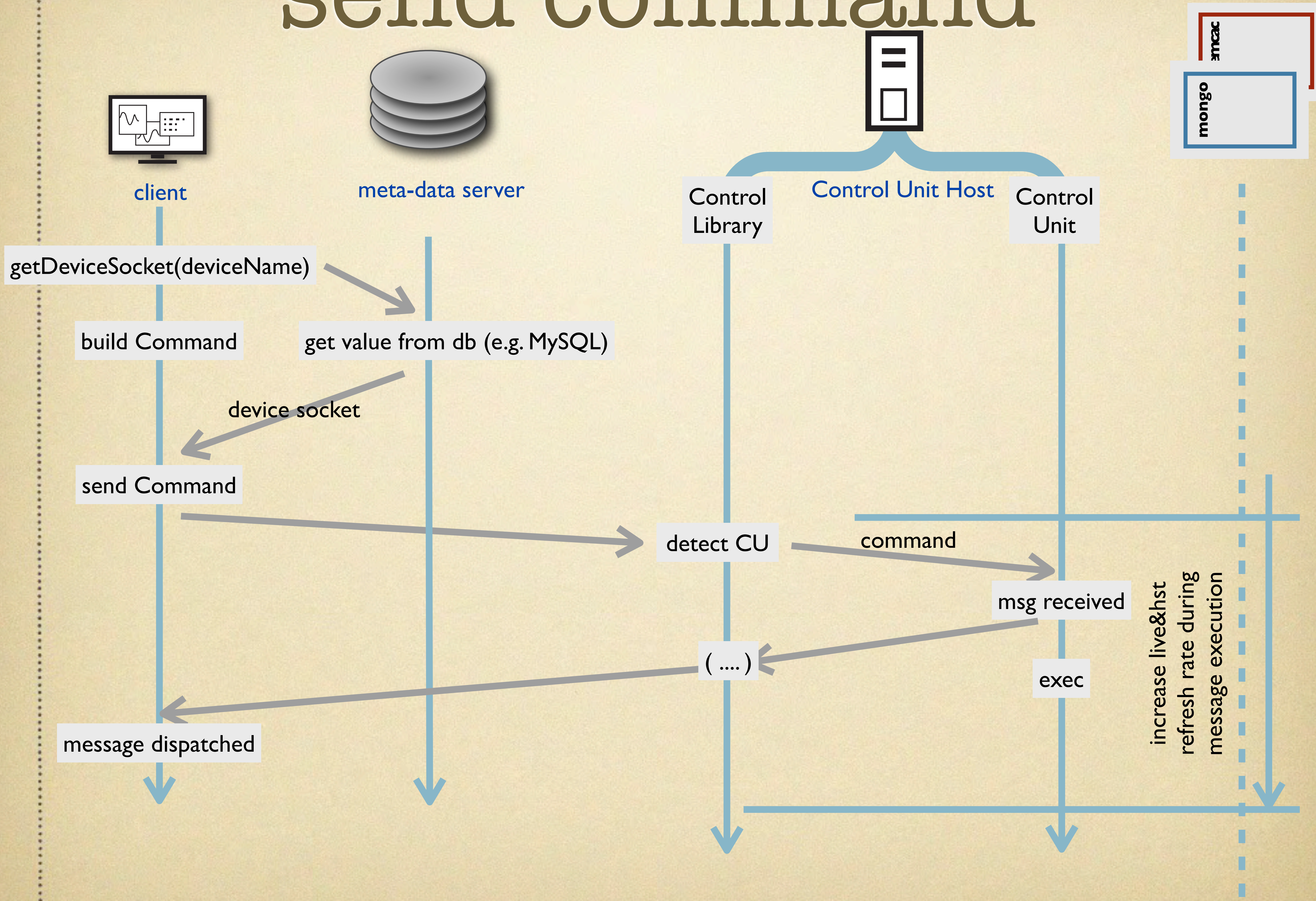
memcache_get(deviceID)

get value from memory hash

Bson serialization of
ALL cached value

Bson parsing

send command



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

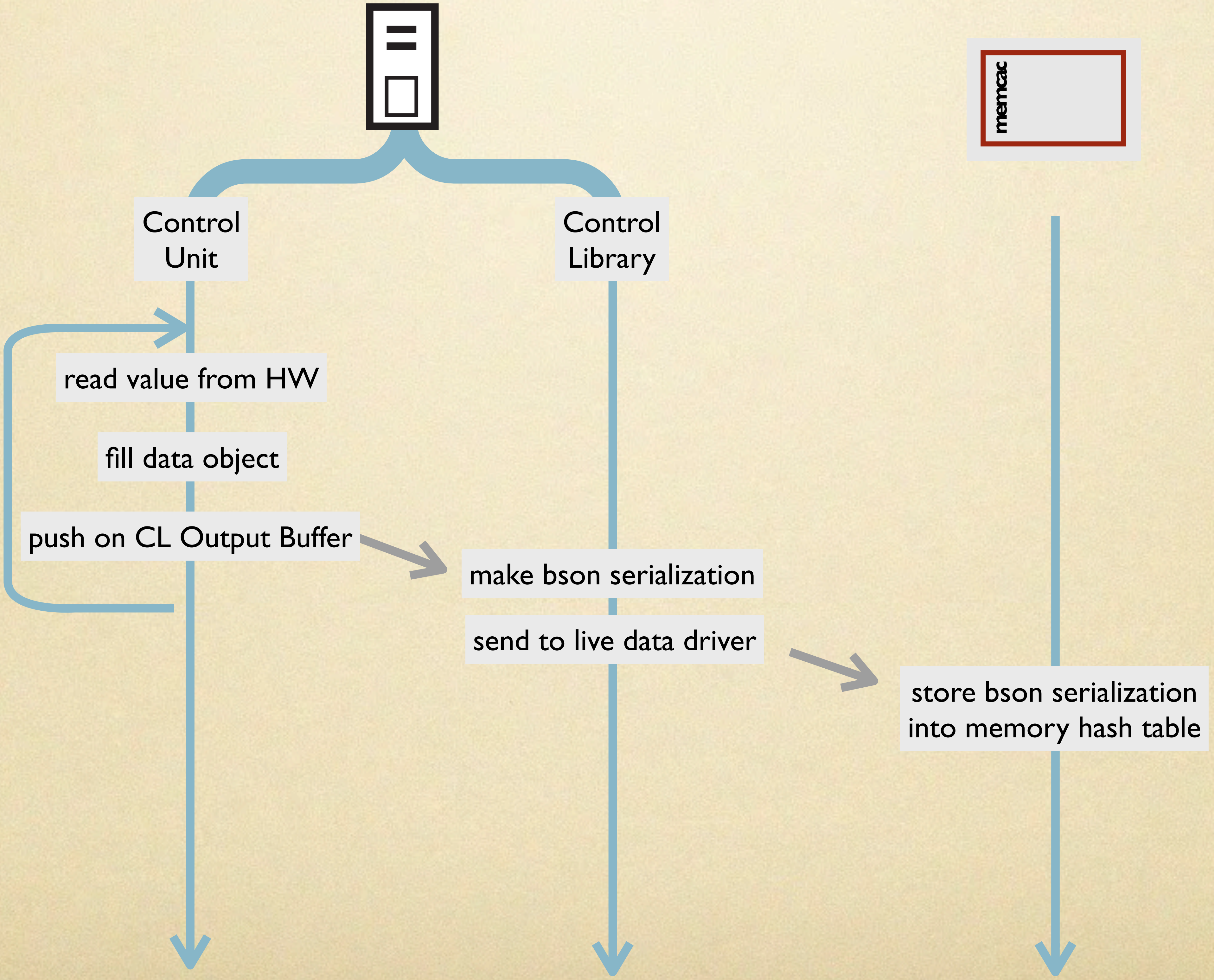
(....)

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

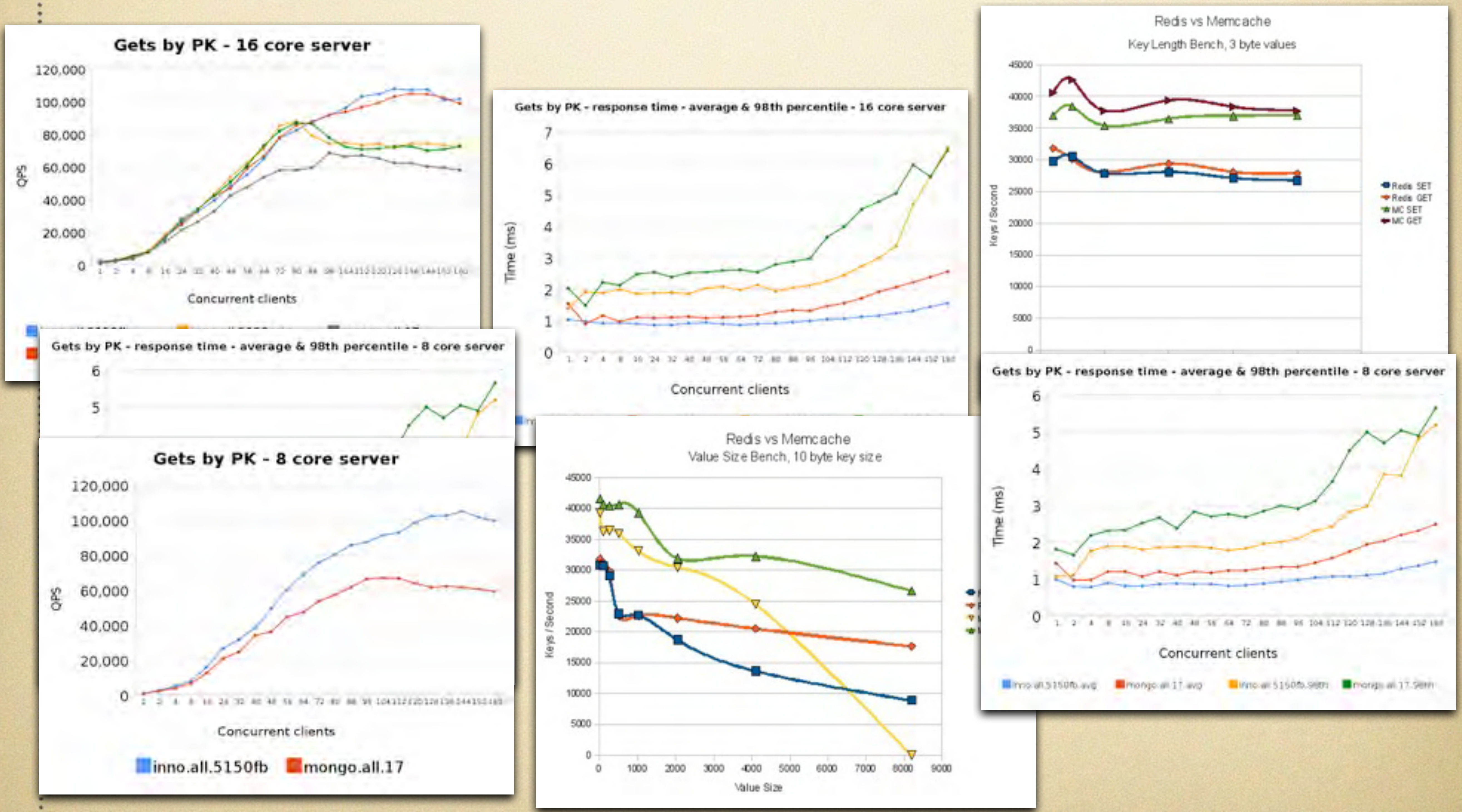
push on CL Output Buffer

make bson serialization

send to live data driver

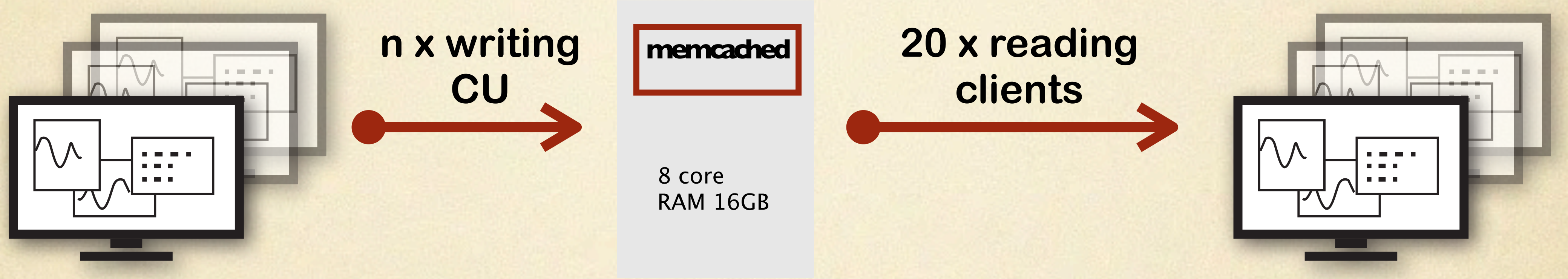
store bson serialization into memory hash table

memcached performance



thanks to C.Bisegni/F.Zani

test#3.1



n x writing CU

memcached

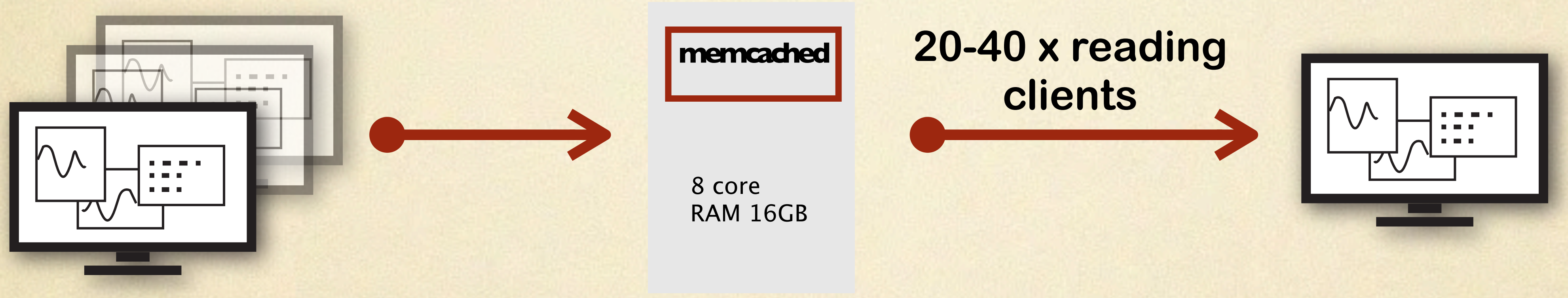
8 core
RAM 16GB

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	?

thanks to C.Bisegni/F.Zani

test#3.2



memcached

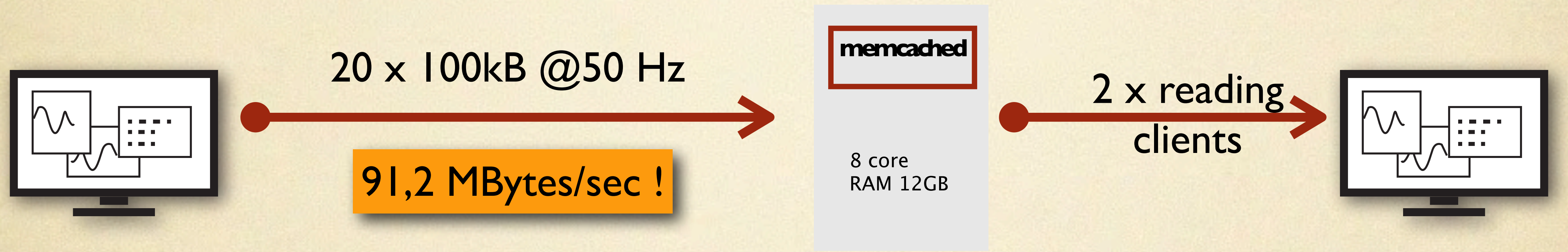
8 core
RAM 16GB

20-40 x reading clients

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

thanks to C.Bisegni/F.Zani

test#4

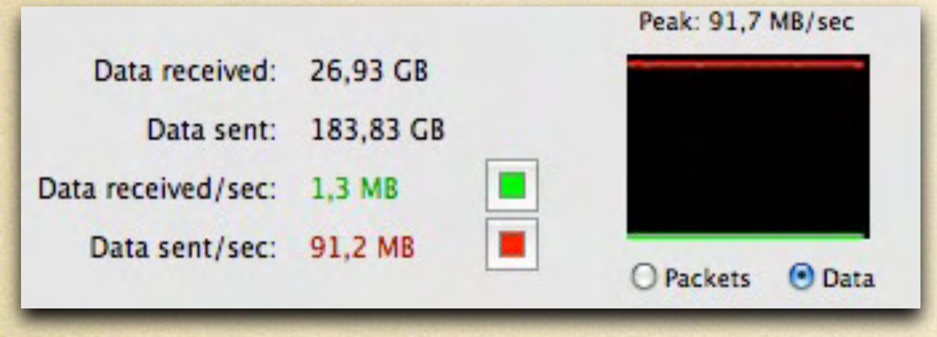


20 x 100kB @50 Hz

91,2 MBytes/sec !

8 core
RAM 12GB

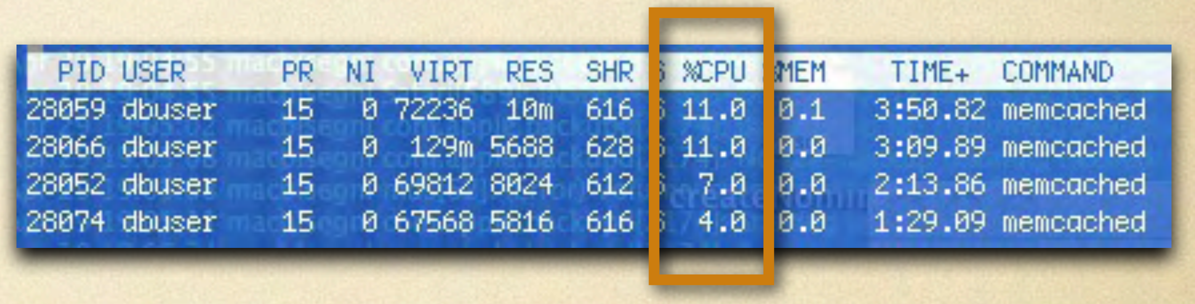
2 x reading clients



Peak: 91,7 MB/sec

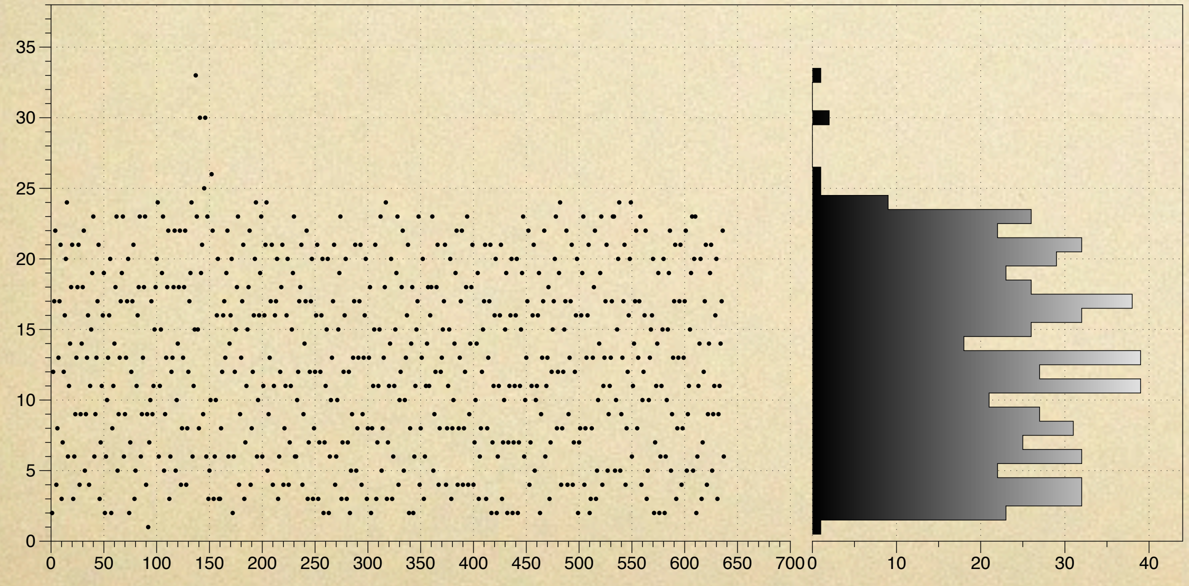
Data received:	26,93 GB
Data sent:	183,83 GB
Data received/sec:	1,3 MB 🟩
Data sent/sec:	91,2 MB 🟥

○ Packets ● Data

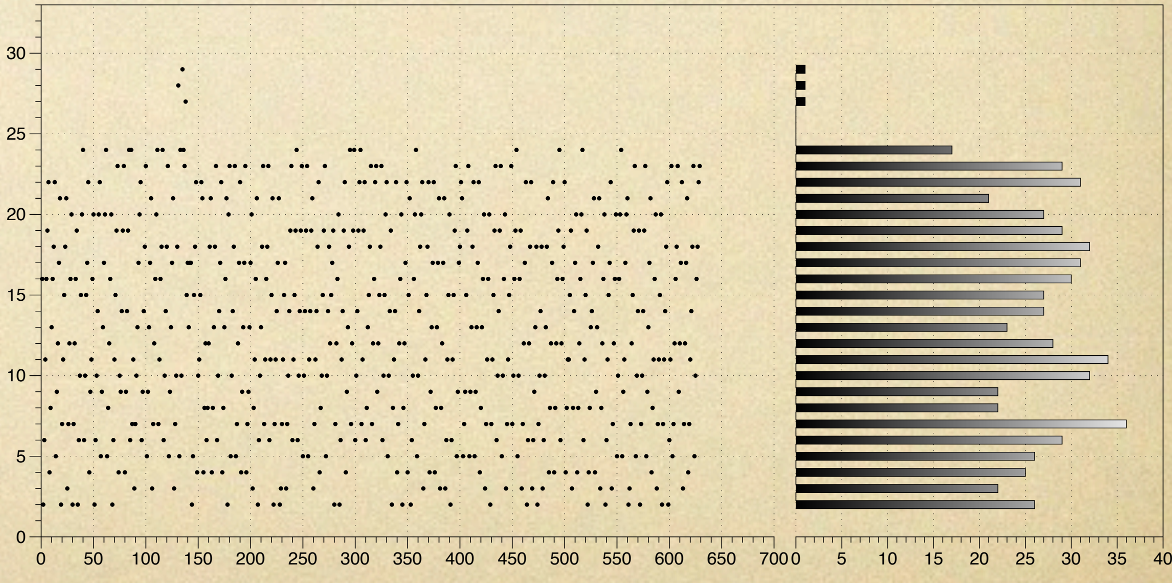


```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
28059 dbuser    15   0 72236  10m  616 S 11.0  0.1   3:50.82 memcached
28066 dbuser    15   0  129m 5688  628 S 11.0  0.0   3:09.89 memcached
28052 dbuser    15   0 69812 8024  612 S  7.0  0.0   2:13.86 memcached
28074 dbuser  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

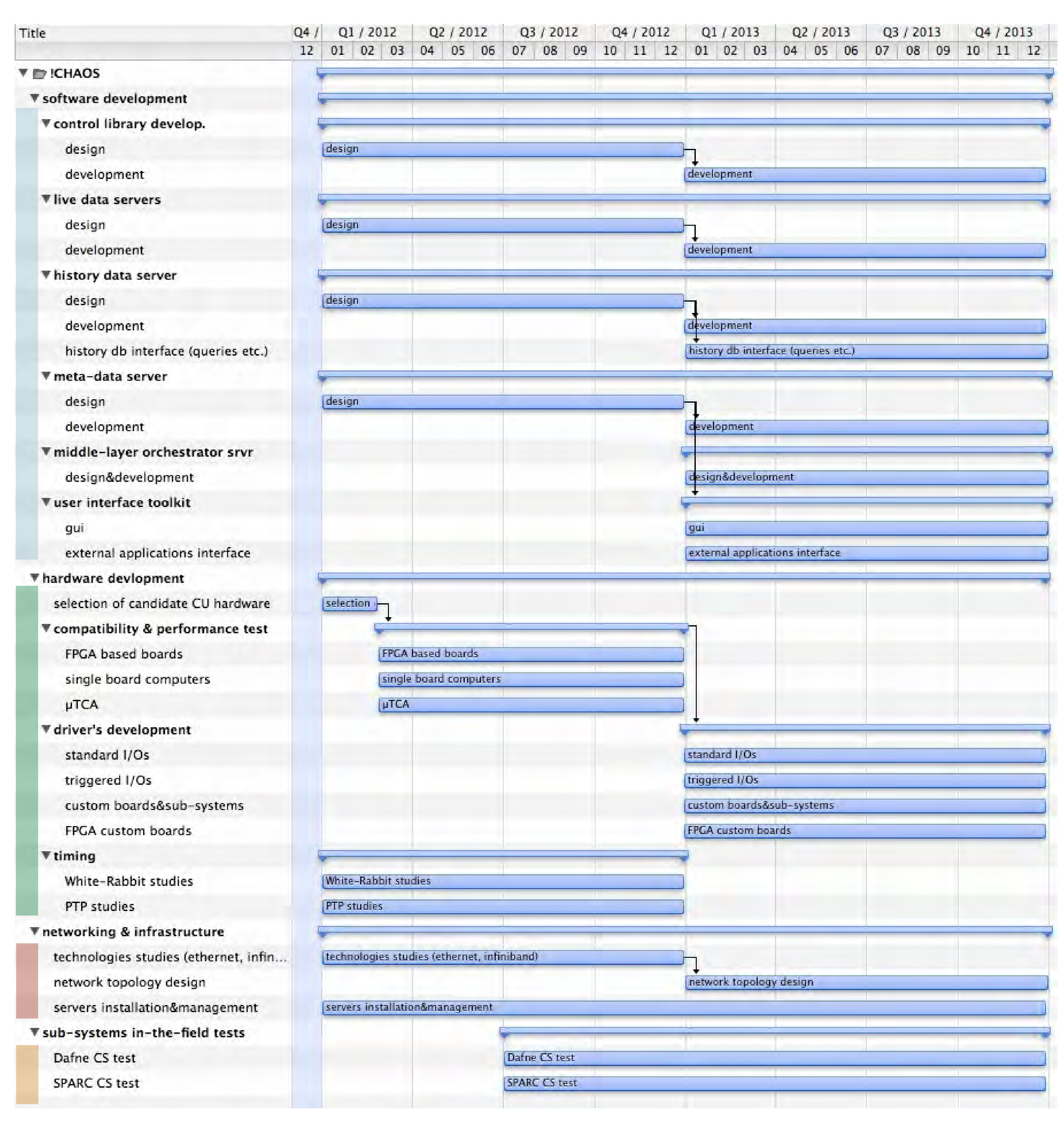
s4_hardware1_w20_m20_buff100000_rd10.log



s4_hardware1_w20_m20_buff100000_rd12.log



The !CHAOS R&D activity



Title

- !CHAOS
  - software development
    - control library develop.
      - design
      - development
    - live data servers
      - design
      - development
    - history data server
      - design
      - development
      - history db interface (queries etc.)
    - meta-data server
      - design
      - development
    - middle-layer orchestrator srvr
      - design&development
    - user interface toolkit
      - gui
      - external applications interface
  - hardware devlopment
    - selection of candidate CU hardware
    - compatibility & performance test
      - FPGA based boards
      - single board computers
      - µTCA
    - driver's development
      - standard I/Os
      - triggered I/Os
      - custom boards&sub-systems
      - FPGA custom boards
    - timing
      - White-Rabbit studies
      - PTP studies
  - networking & infrastructure
    - technologies studies (ethernet, infin...
    - network topology design
    - servers installation&management
  - sub-systems in-the-field tests
    - Dafne CS test
    - SPARC CS test

conclusions and future plans

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

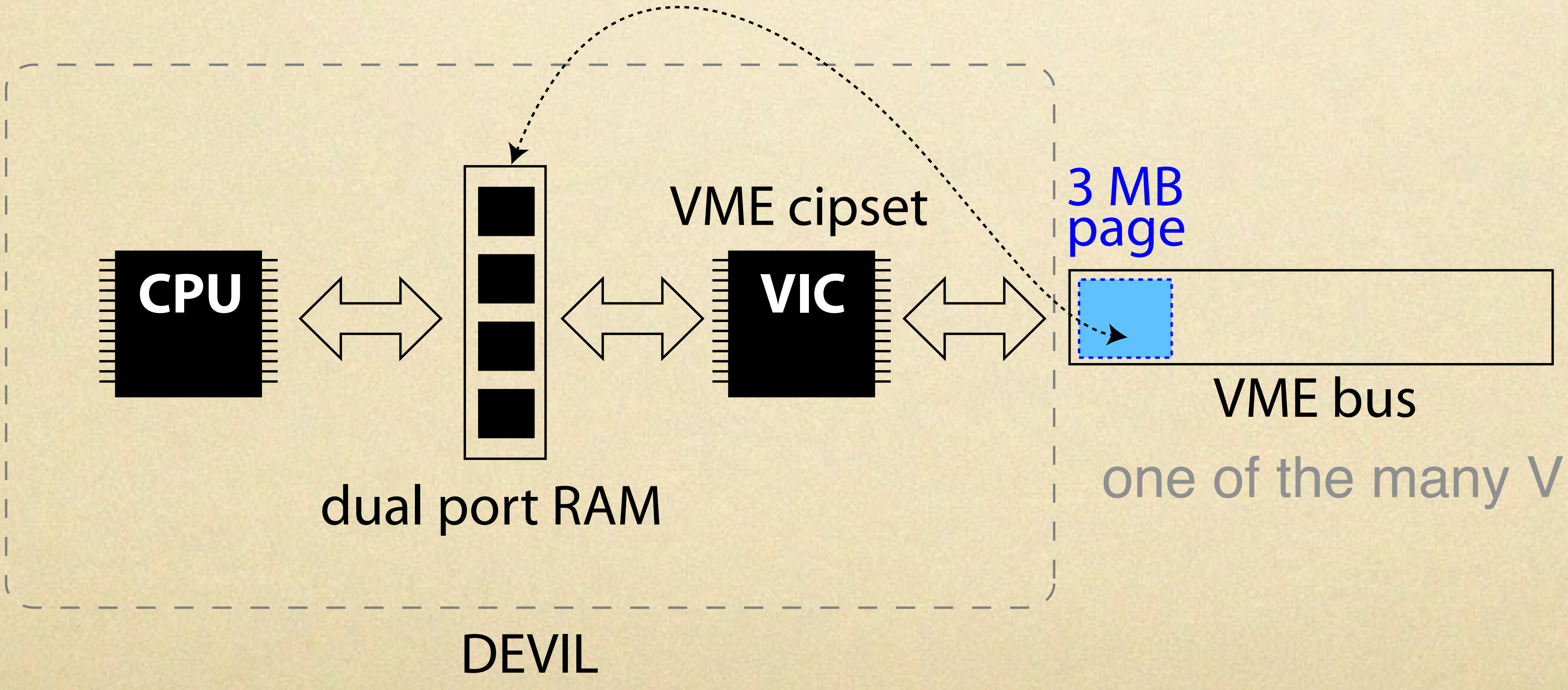
- start an R&D activity (funded by INFN CSN5) for completing system design and continue performance and stress tests of components
- continue tests on the field by adding CS components to Dafne&SPARC prototypes
- finalize the project as a candidate for the SuperB Control and DAQ System
- evaluate costs, man power and define time schedule

!CHAOS

Memcached functional tests in two real contexts: the DAFNE Control System and the SPARC Control System

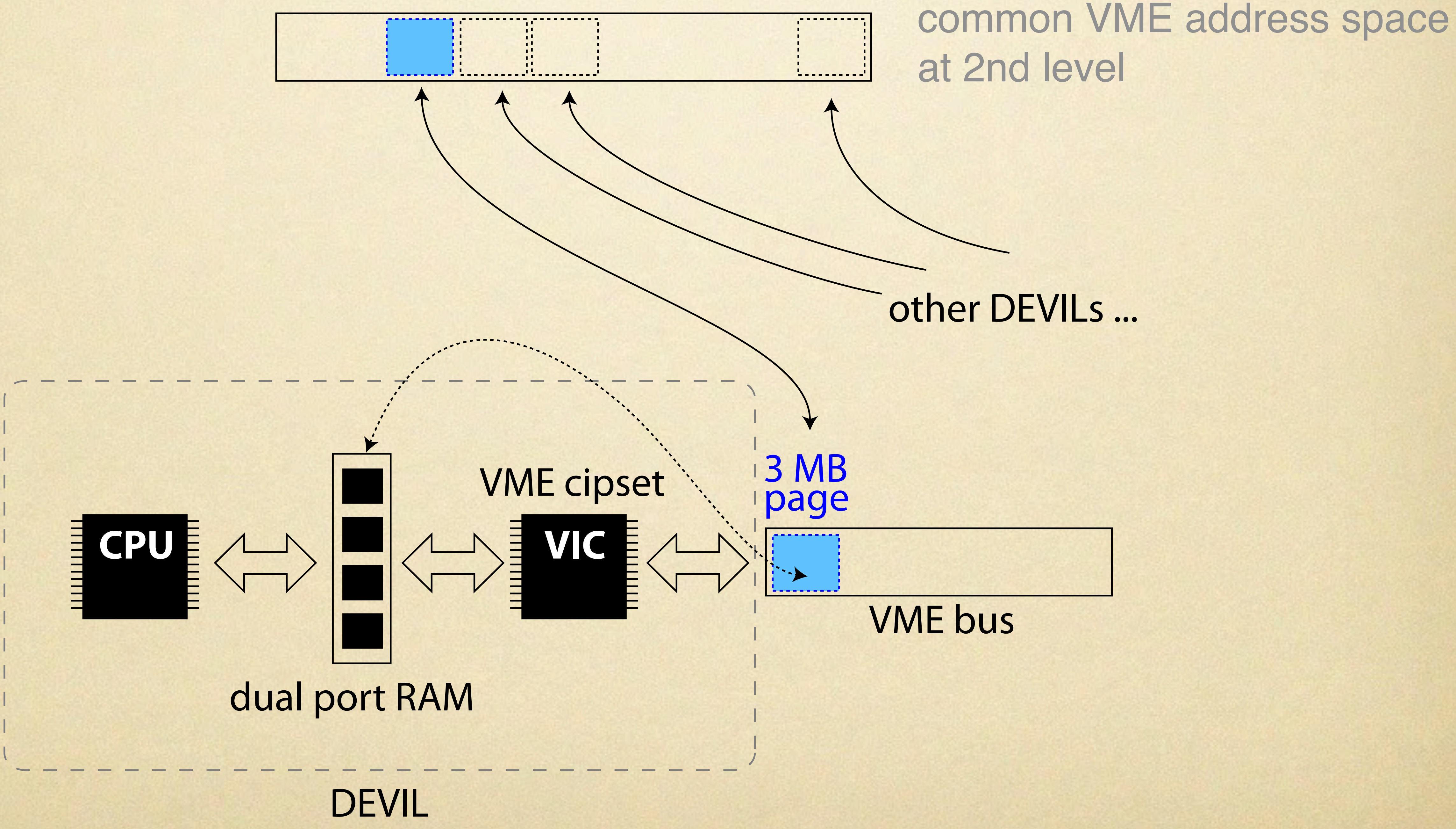
A. Stecchi	INFN - LNF
R. Ammendola	INFN Roma-TV
C. Bisegni	INFN - LNF
S. Calabrò	LAL / INFN - LNF
L. Catani	INFN Roma-TV
P. Ciuffetti	INFN - LNF
G. Di Pirro	INFN - LNF
L. Foggetta	LAL / INFN - LNF
G. Mazzitelli	INFN - LNF
F. Zani	INFN Roma-TV

The DAFNE Control System relies on many distributed VME embedded processors (the 3rd level)

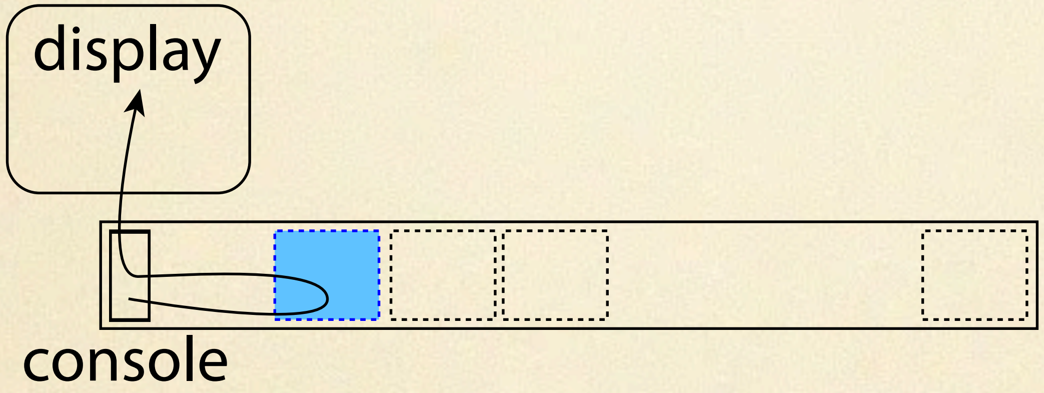


one of the many VME bus at 3rd level

Employing many point-to-point VME optical links, all the distributed processors contribute, with theirs RAMs, to the constitution of a common VME address space (the 2nd level)...

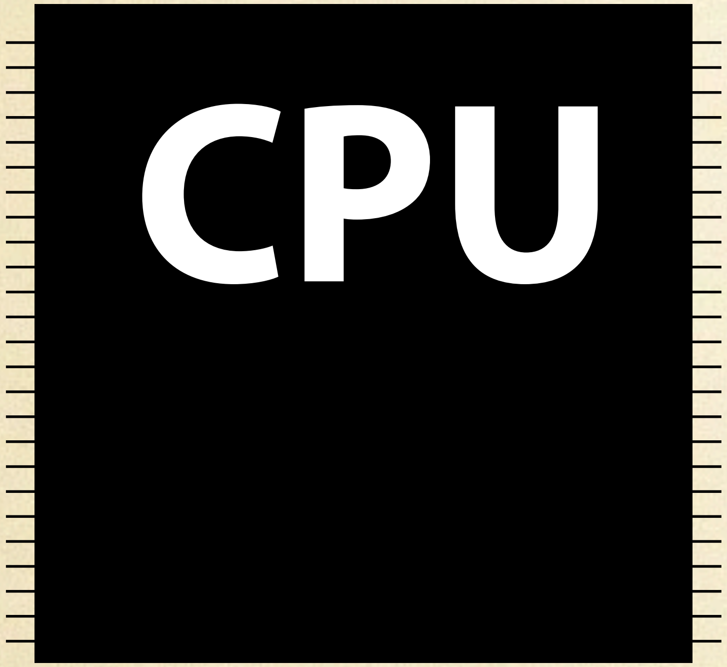


... that is available to the console applications (the 1st level)

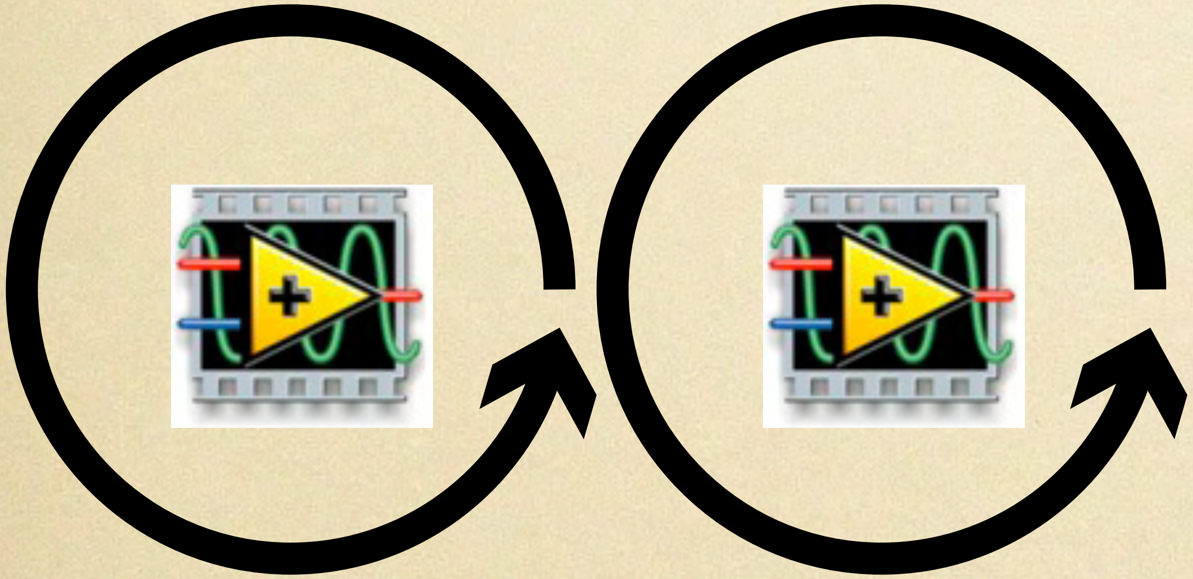


common VME address space at 2nd level

The 3rd level at a glance



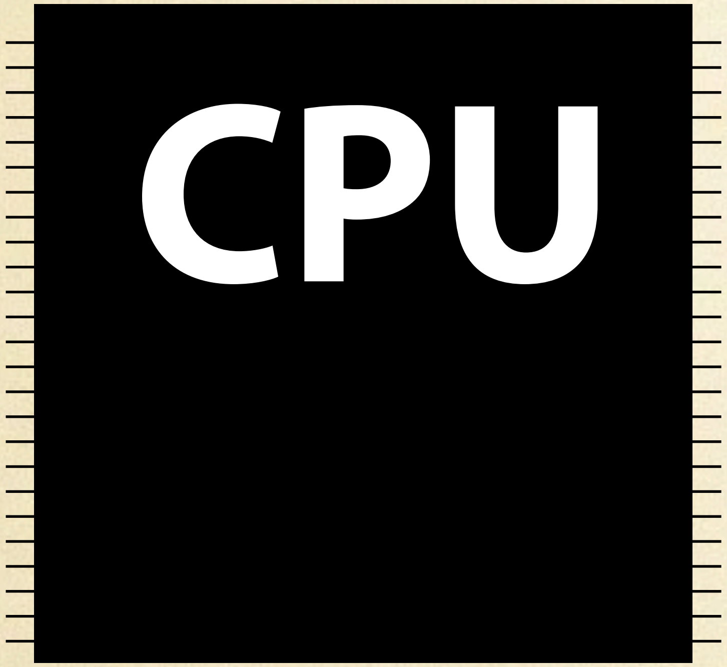
Each distributed CPU runs a LabVIEW® application that takes care of monitoring and controlling the devices under its responsability

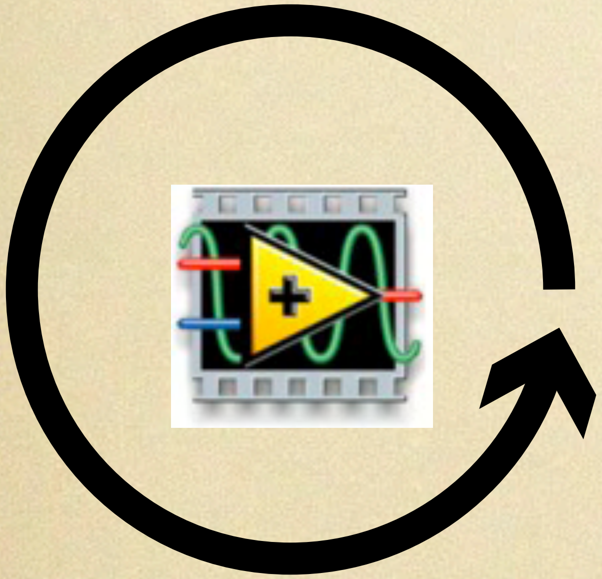


control loop

command loop

The 3rd level at a glance

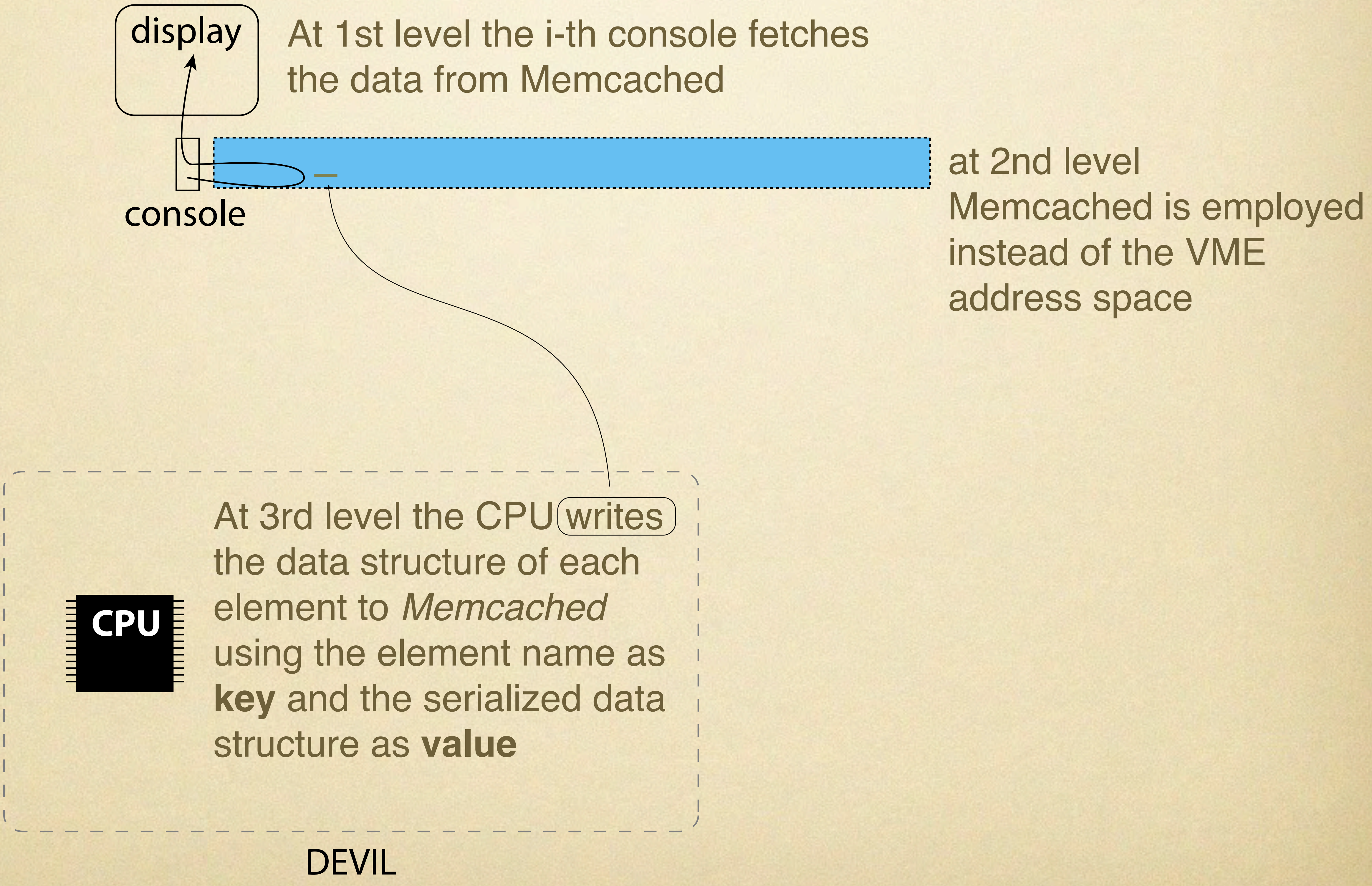




The control loop

- reads the devices *by means of proper drivers*
- for each device:
  - builds a data structure *with the current values*
  - updates the data structure in the local RAM

The idea: replace the 2nd level VME address space with the *Memcached* associative memory



At 1st level the i-th console fetches the data from Memcached

at 2nd level Memcached is employed instead of the VME address space

At 3rd level the CPU writes the data structure of each element to *Memcached* using the element name as **key** and the serialized data structure as **value**

For the preliminary test on the DAFNE Control System, *Memcached* has been installed on a very basic machine:

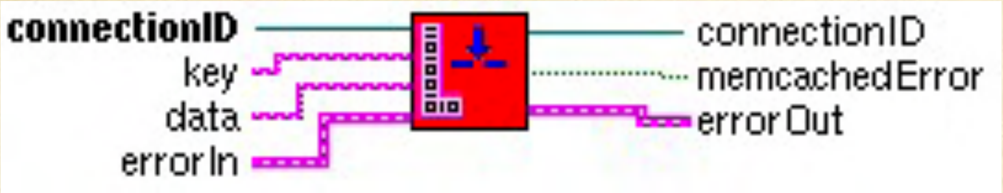
Sun V20z
   AMD Opteron 244 @ 1.8 GHz
   2 GB RAM
   Ethernet @ 100Mbps

The OS is Linux
   CentOS 5.5 (64 bit)

Memcached version 1.4.5 (latest stable)
   with 512 MB of RAM allocated

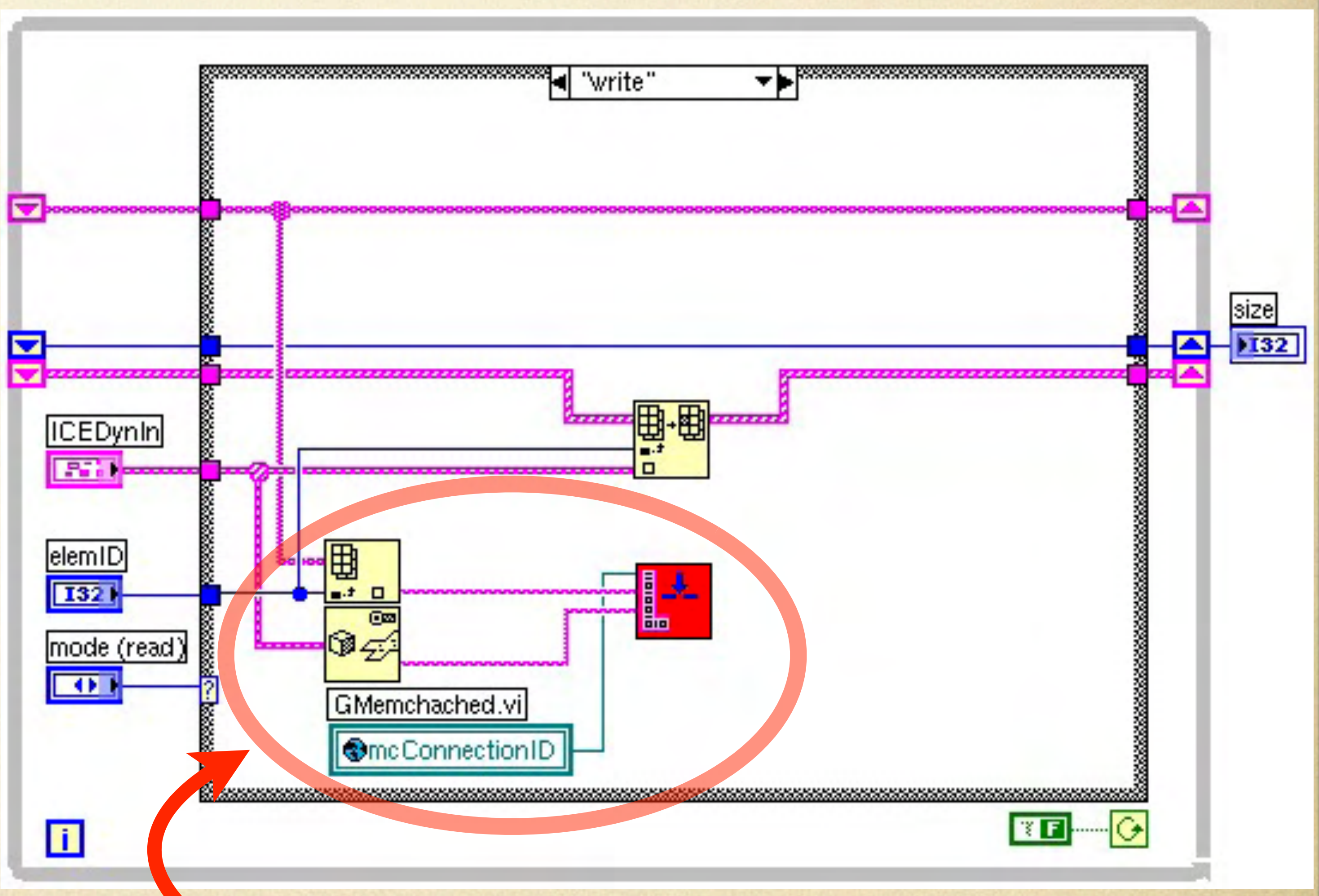
The modification has been applied to the element class ICE (Ion cleaning Electrodes)

In the 3rd level CPU a new LabVIEW routine for writing to memcached has been inserted



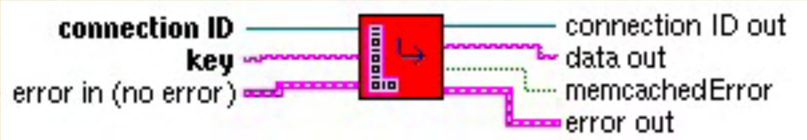
Besides a few initialization tasks, all the rest of the code stays unmodified.

This simple and localized change is sufficient for making the data available to Memcached.

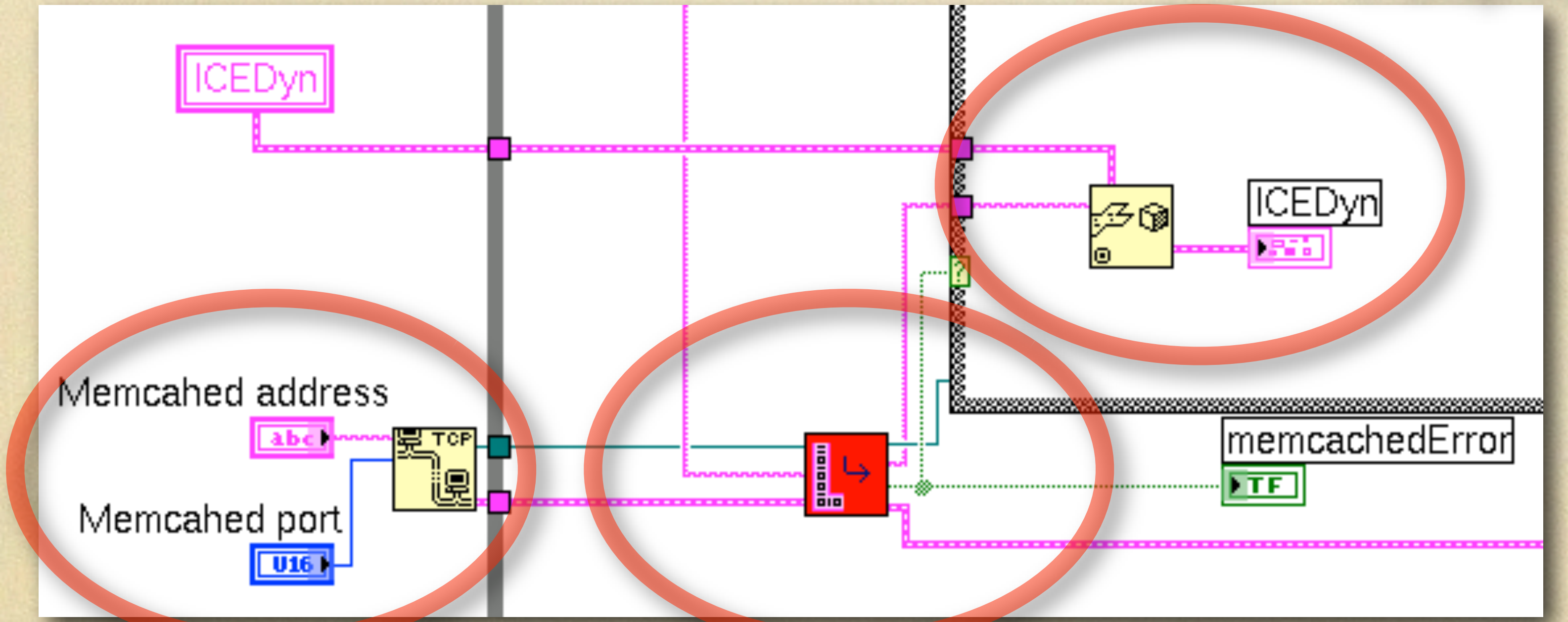


new code for writing to Memcached

The first level reads the element data from Memcached and de-serializes it to restore the original element data structure.



LabVIEW routine for reading from Memcached

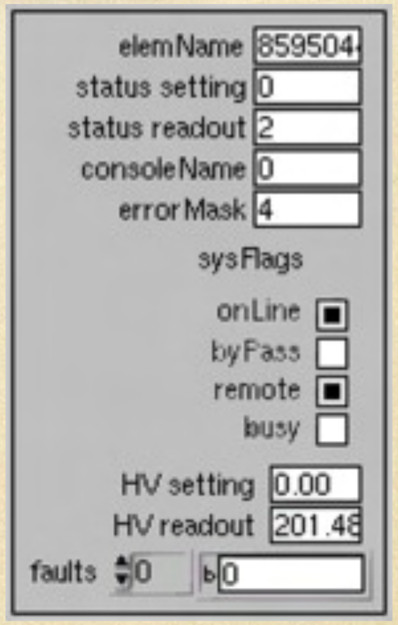


1. Memcached connection

2. Memcached read

3. de-serialization

4. display



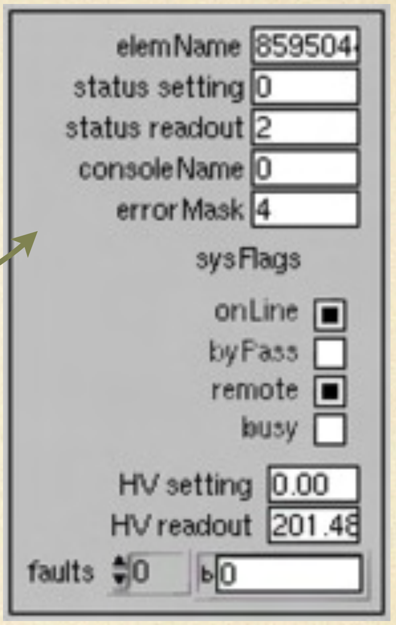
elemName 859504
status setting 0
status readout 2
consoleName 0
errorMask 4

sysFlags

onLine ■
byPass □
remote ■
busy □

HV setting 0.00
HV readout 201.48
faults 0 0

Preliminary measurements



elemName 859504

status setting 0

status readout 2

consoleName 0

errorMask 4

sysFlags

onLine ■

byPass □

remote ■

busy □

HV setting 0.00

HV readout 201.48

faults 0 b 0

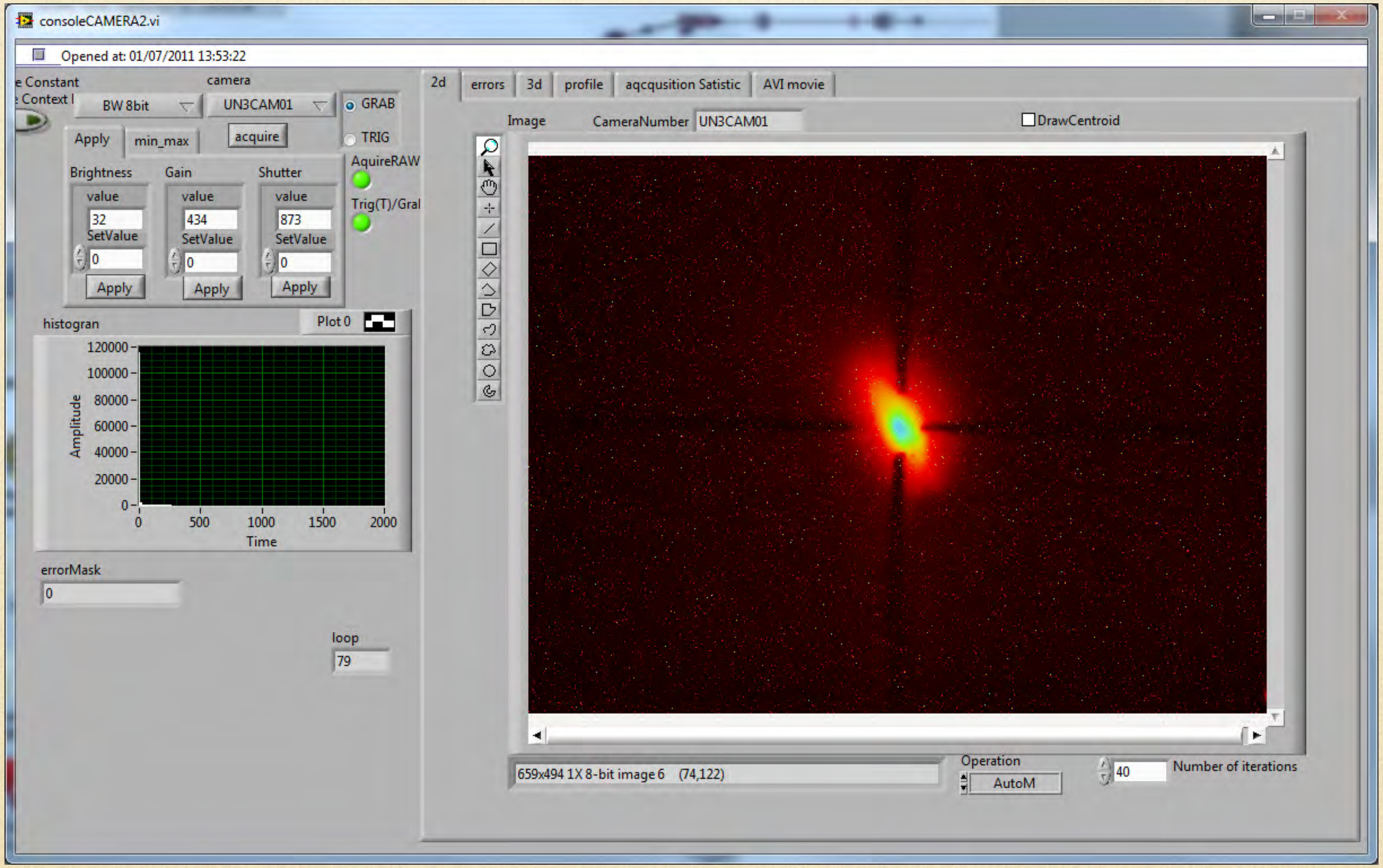
data size: 64 bytes for packet read

fetch frequency ~ 100 Hz
with no dependency on the number of fetching consoles (up to 7 in our test)

Memcached server load (measured with the *top* command)
CPU: 0.3% - 0.7%   memory: ~ 0.1%

An similar test has been carried out on the SPARC Control System

Memcached has been used for storing the beam spot image from a digital camera


consoleCAMERA2.vi

Opened at: 01/07/2011 13:53:22

camera: BW 8bit, UN3CAM01

○ GRAB

○ TRIG

Brightness value: 32, SetValue: 0

Gain value: 434, SetValue: 0

Shutter value: 873, SetValue: 0

errorMask: 0

loop: 79

CameraNumber: UN3CAM01

□DrawCentroid

659x494 1X 8-bit image 6 (74,122)

Operation: AutoM

Number of iterations: 40

- network: Ethernet @1 Gbps
- image size: 640x480@8 bit = 300 kByte
- measured fetch frequency: ~ 25 Hz with no dependency on the number of fetching consoles (up to 4 in our test)

Conclusions:

Memcached demonstrated to be very stable, and to have a low impact on the CPU load

Very good overall performance *(Ethernet 100Mbps has been employed instead of 1 Gbps or even 40 Gbps Infiniband in the future...)*

We are seriously considering the possibility to adopt *Memcached* for a real upgrade of the DAFNE & SPARC Control Systems

We are encouraged to go on with a deeper integration of !CHAOS components in the DAFNE Control System