C ontrol system based on a
H ighly
A bstracted and
O pen
S tructure

L. Catani

Few introductory slides
on Control Systems (CS)

CS evolution

- early in the 60's accelerators and experimental apparatus were remotely controlled by pulling wires from device to control room panels having conventional connectors, knobs, analog gauges of various types
- Pushed by the development of electronic and micro-electronic and, later, by the progress of the information technology (exponential growth of computing power, development of communications networks and an overall reduction of costs and increase of performance) computers entered the game.
- initially used for quasi-on line measurement or feedback
- (expensive) workstations first, (low cost) PC later

CS evolution

- in the early '70s CS included ~ 2500 I/O channels;
- large experimental apparatuses in the half of '90s reached ~ 100,000 I/O channels
- increasing man-power requirements for the developments of CS suggested to avoid home made solutions and independent developments
- tendency is shared development or simply implementation of open source solutions
- in the case of development from scratch limit the number of HW/SW technologies used but offer access points to major languages and SW tools

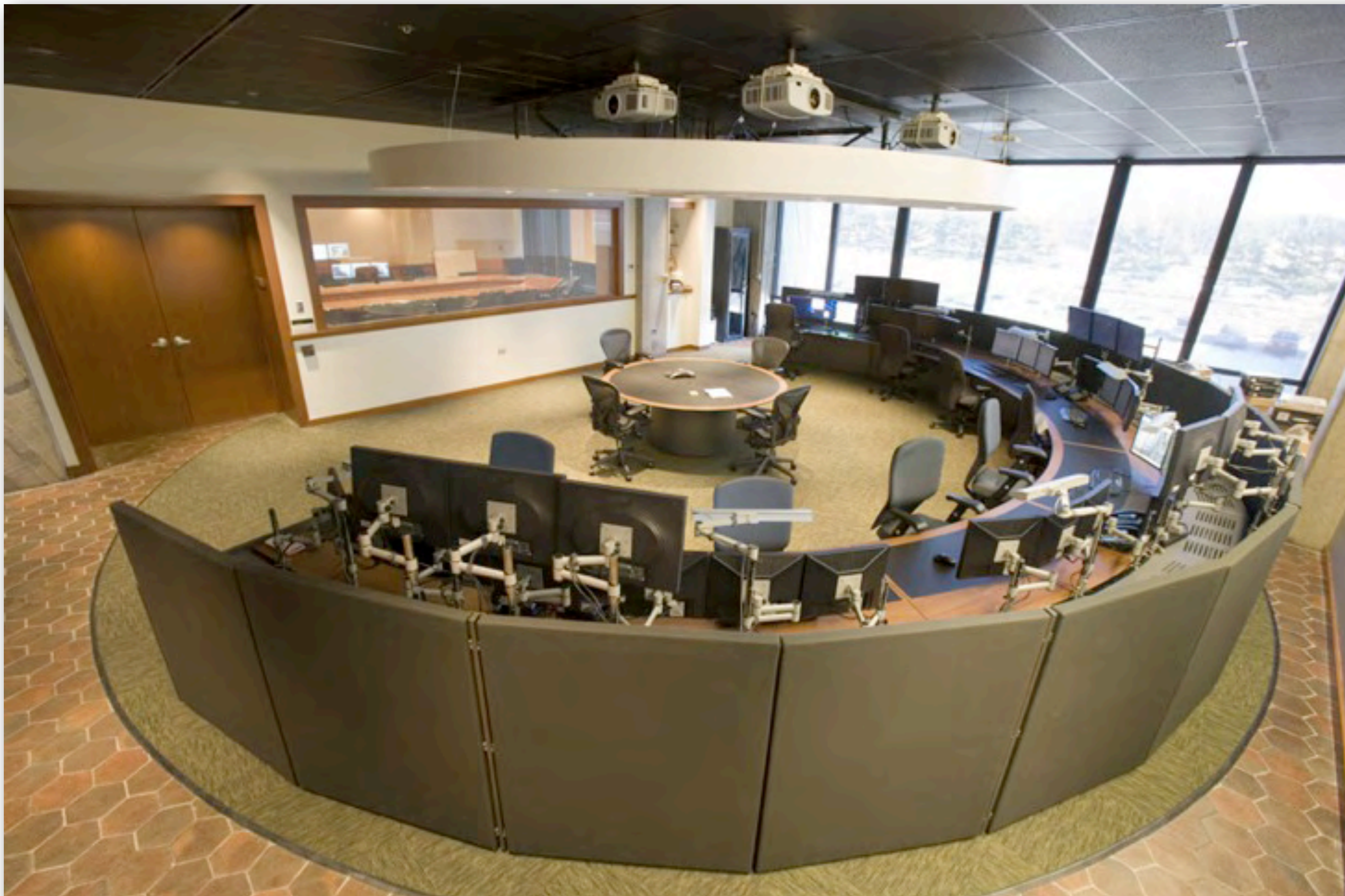
CS evolution

- nowadays CS are integral part not just of the accelerator, but of the entire infrastructure
- all components of the accelerator are managed by the CS (also the local control during maintenance) which then handles thousands of I / O channels of all types
- CS extend the performance of diagnostic systems and allow implementing new features for handling equipment
- information technology offers solutions for providing operators and scientists with more versatile and simple tools for operation

CS evolution

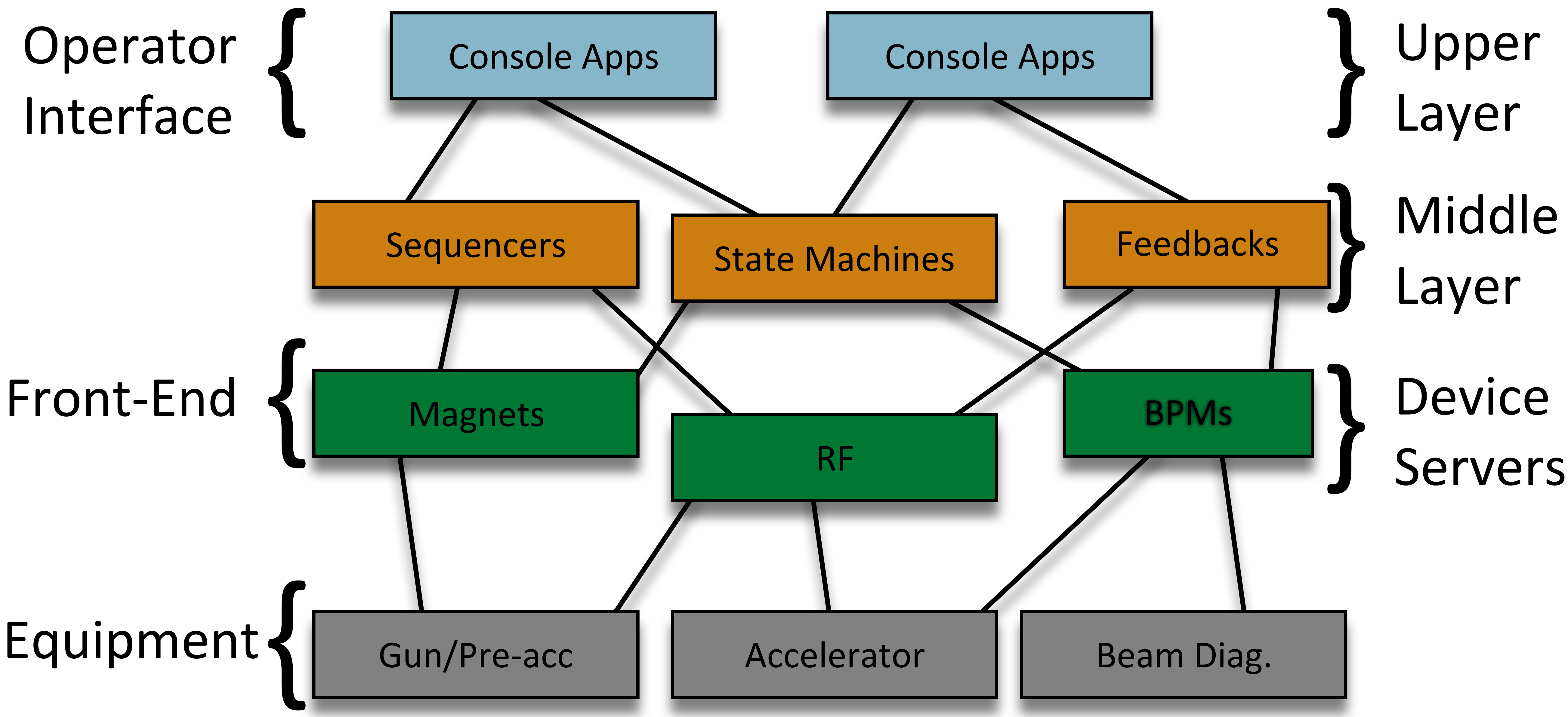
remote operations

A remote operations center, LHC@FNAL, has been built at Fermilab to make it easier for accelerator scientists and experimentalists working in North America to help commission and participate in operations of the LHC and experiments.



Control systems communities

the “standard model”



the "standard model"

First Level

- display data and status of accelerator devices by means of customizable control panels.
- on-line data analysis
- errors and alarms notification
- set-points back up and restore

the “standard model”

Second Level

- communication
- database
- feedback software and “state machine”
- multi-element control procedures
- more...
- access-point for external tools

the “standard model”

Third Level

- interface to accelerator components by means of:
  - I/O channels (ex. an analog voltage output to control the set-point of a magnet power supply)
  - field bus (distributed I/O channels: CANbus, serial etc.)
- interface to stand-alone sub-systems (interlocks, security, components with their own controls)

the “standard model”

distributed controls

- for each level the software development and more appropriate hw can be chosen (with some limitations)
- links between the levels of the system are provided by the communication system
- real time procedures are located near the equipment
- if the communication system is modular, expansions or upgrades of S.diC. can be obtained simply by adding additional third level CPUs

Communication: ethernet networks or...

- communication solutions are now widely based on LAN (Local Area Network)
- early in the'90s alternatives to the network were still considered in medium size CS (VME vertical bus, reflective memory boards)
- though non-deterministic, Ethernet has proven its validity as a physical-layer for communication protocols of CS
- standard networks (1-10 Gb/s) provide sufficient bandwidth, especially if using dedicated networks for CS....
- ..but, if possible, limit data transfer by implementing calculations directly on the front-end

Existing CS: EPICS

What is EPICS?

- Early Major Collaborators
  - America
    - ANL, LANL, LBL, ORNL, SLAC, JLAB
  - Europe
    - DESY, BESSY, PSI
  - Asia
    - KEK
- Recent Major Collaborators
  - America
    - TRIUMF, CLS, BNL
  - Europe
    - DIAMOND Light Source
  - Asia/Australia
    - J-PARC, Australian Synchrotron, IHEP, SSRF, TLS, RRCAT
- Many Other Collaborators/Users

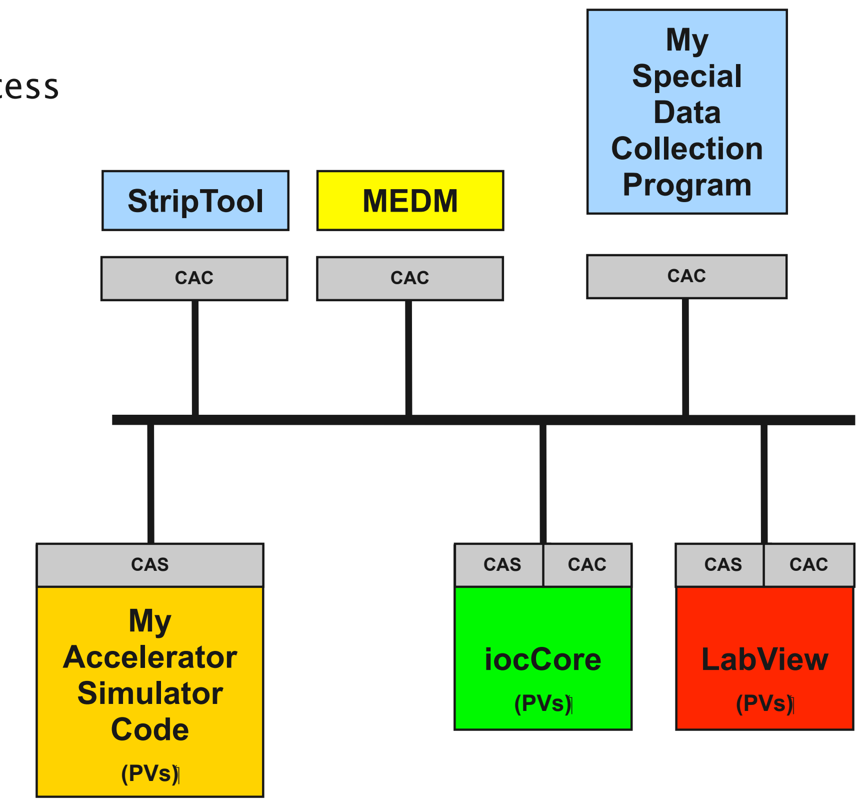
January 2009 EPICS Seminar RRCAT 5

Existing CS: EPICS

What is EPICS?

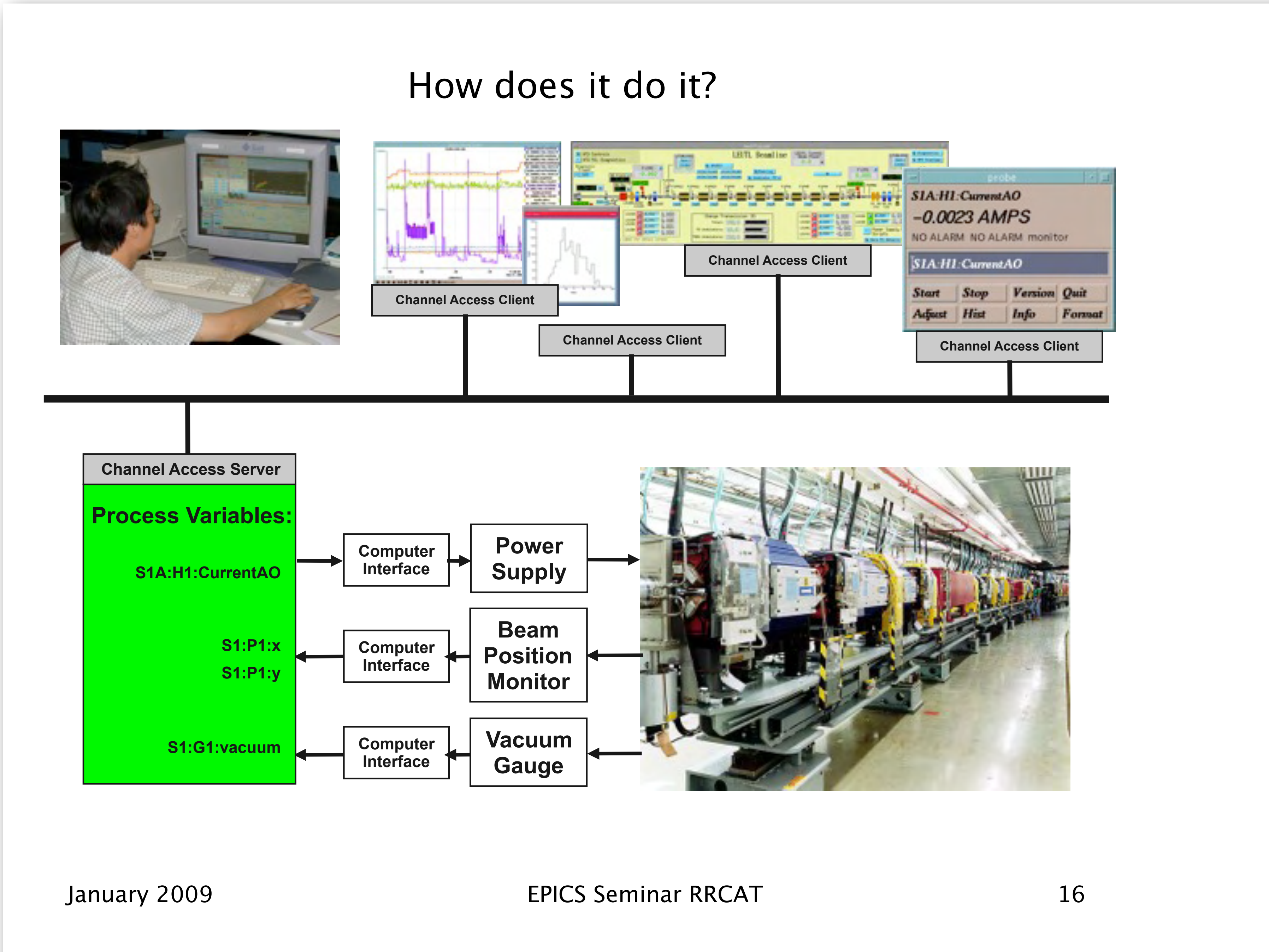
- Any tool/program/application that abides by the Channel Access protocol could be described as "EPICS Compliant".

EPICS can be viewed as a "toolkit" of EPICS compliant programs. One can select the appropriate tool for their need or develop their own.



January 2009 EPICS Seminar RRCAT 15

Existing CS: EPICS

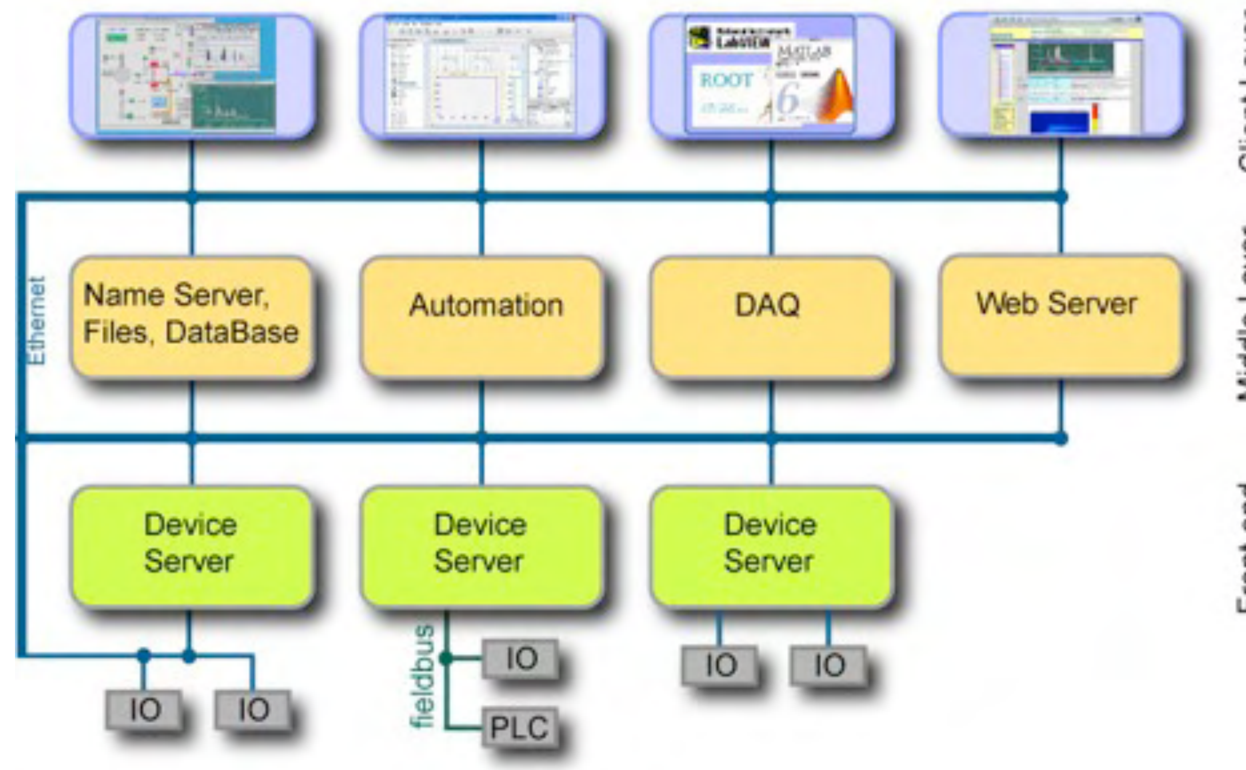


Existing CS: TANGO

The TAco Next Generation Objects (TANGO) control system is a **free open source object-oriented** control system for controlling accelerators, experiments and any kind of hardware or software being actively developed by a consortium of (mainly) synchrotron radiation institutes.
TANGO is a distributed control system.
It runs on a single machine as well as hundreds of machines.
TANGO uses the **omniorb** implementation of CORBA as its network protocol.
The **client-server** model is the basic communication model.
Communication between clients and servers can be synchronous, asynchronous or event driven.

Existing CS: DOOCS

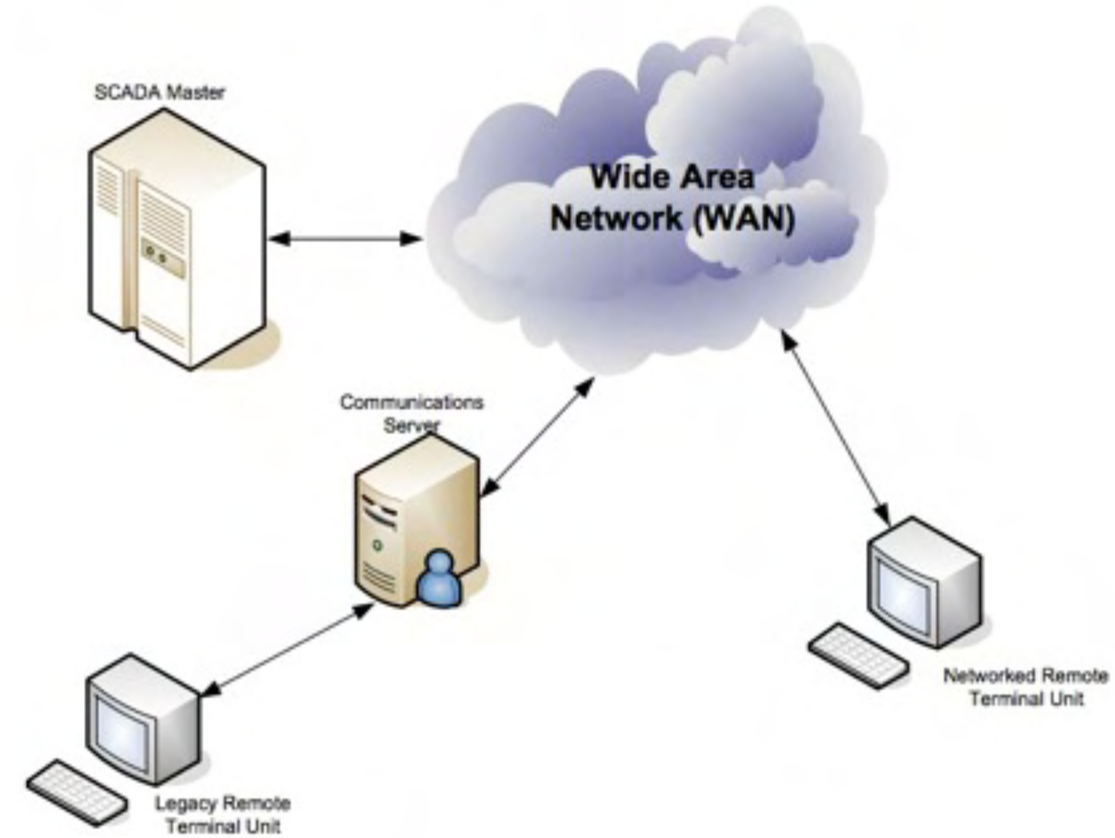
- The Distributed Object Oriented Control System (DOOCS) it is completely written in C++ language and follows the object oriented programming paradigm [2] since the design idea from the device level, including the device servers, up to the user display is based on objects.



- In general, the object oriented approach implements multiple devices of the same type in a device server process on the lowest tier. A certain device is represented as an instance of a device class of its type.
The data of a device, that is accessible as a property of the device from the network, is implemented as data class in a server library.

Existing (industrial) CS: SCADA

- SCADA is an acronym for Supervisory Control and Data Acquisition. SCADA systems are used to monitor and control a plant or equipment in industries such as telecommunications, water and waste control, energy, oil and gas refining and transportation



SCADA systems consist of:

- One or more field data interface devices, usually RTUs, or PLCs, which interface to field sensing devices and local control switchboxes and valve actuators
- A communications system used to transfer data between field data interface devices and control units and the computers in the SCADA central host. The system can be radio, telephone, cable, satellite, etc., or any combination of these.
- A central host computer server or servers (sometimes called a SCADA Center, master station, or Master Terminal Unit (MTU)
- A collection of standard and/or custom software [sometimes called Human Machine Interface (HMI) software or Man Machine Interface (MMI) software] systems used to provide the SCADA central host and operator terminal application, support the communications system, and monitor and control remotely located field data interface devices

back to !CHAOS

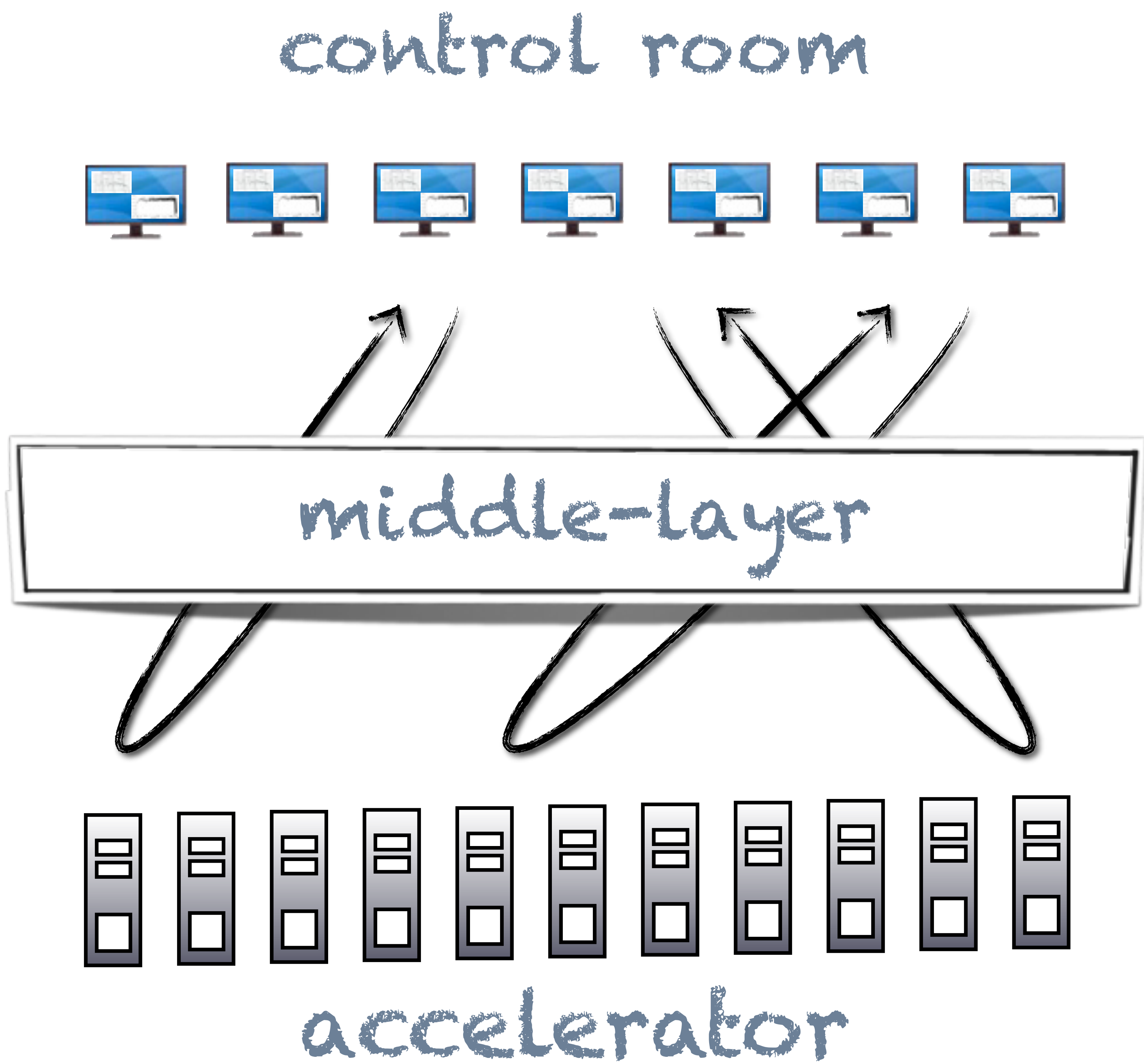
!CHAOS is a **INFN** experiment aimed to defining and validating a new paradigm for distributed control systems (DCS) and fast data acquisition (DAQ) for particle accelerators and experimental apparatuses.

Indeed, its flexibility allows a much wider spectrum of applications.

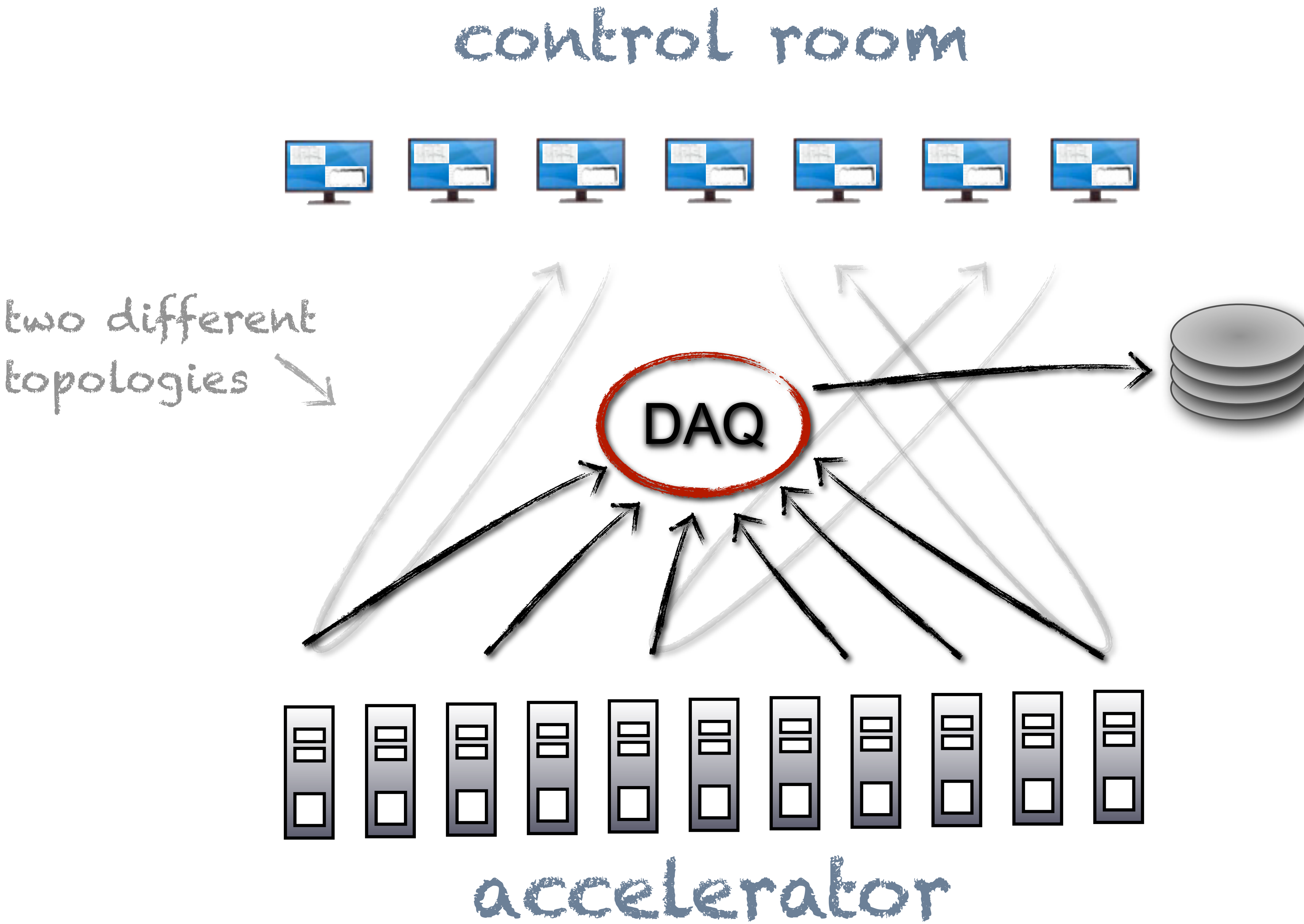
Goals:

- design an innovative control system by implementing the **newest software technologies,** mainly borrowed from the high performance internet services
- introducing the concept of **Control Service** by using a **new services topology** and...
- ...a **new communication solution for live data**, as alternative to client/server
- **high scalability** for each core service;
- **high abstraction** of devices control software;

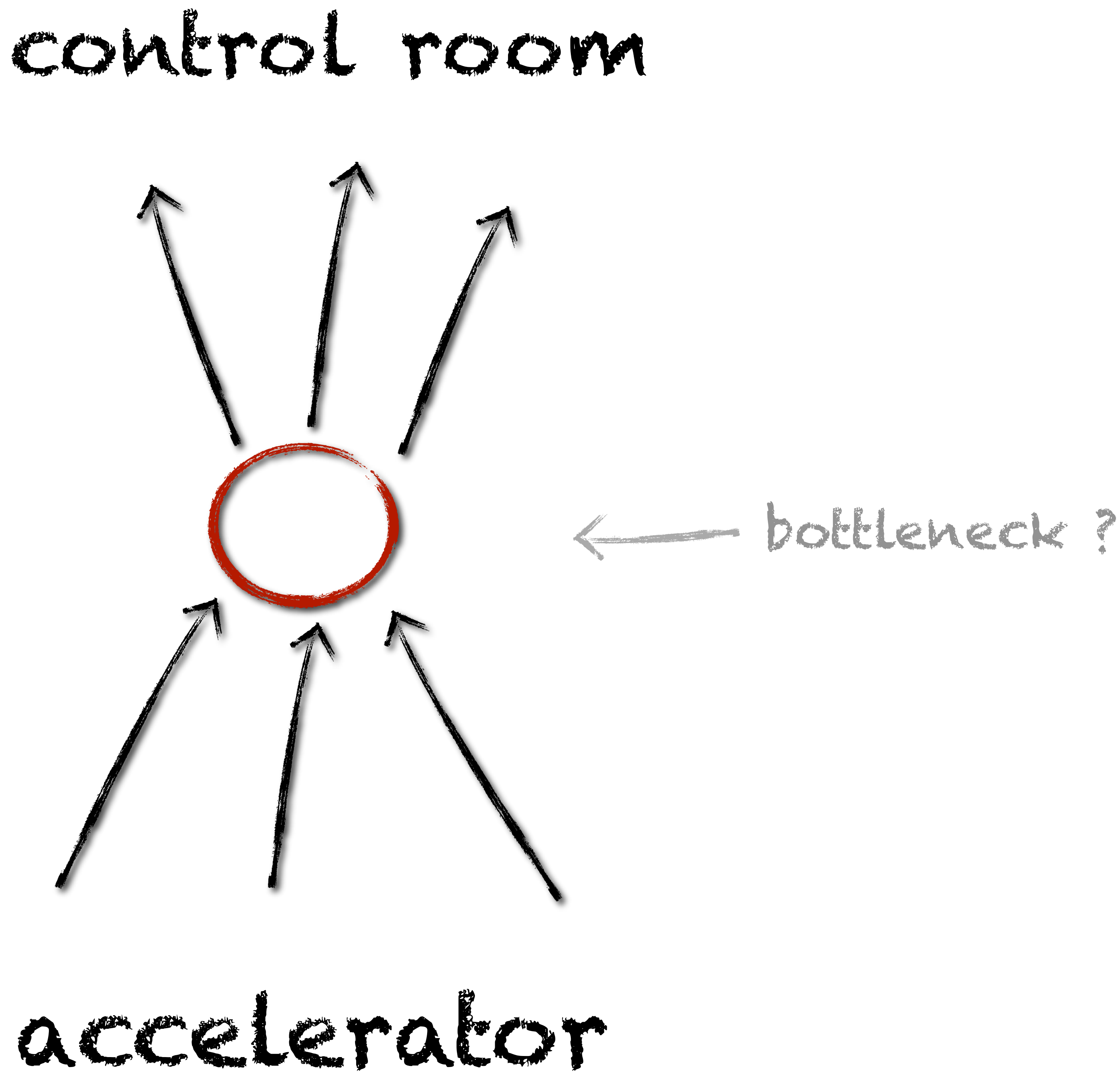
"the standard model"



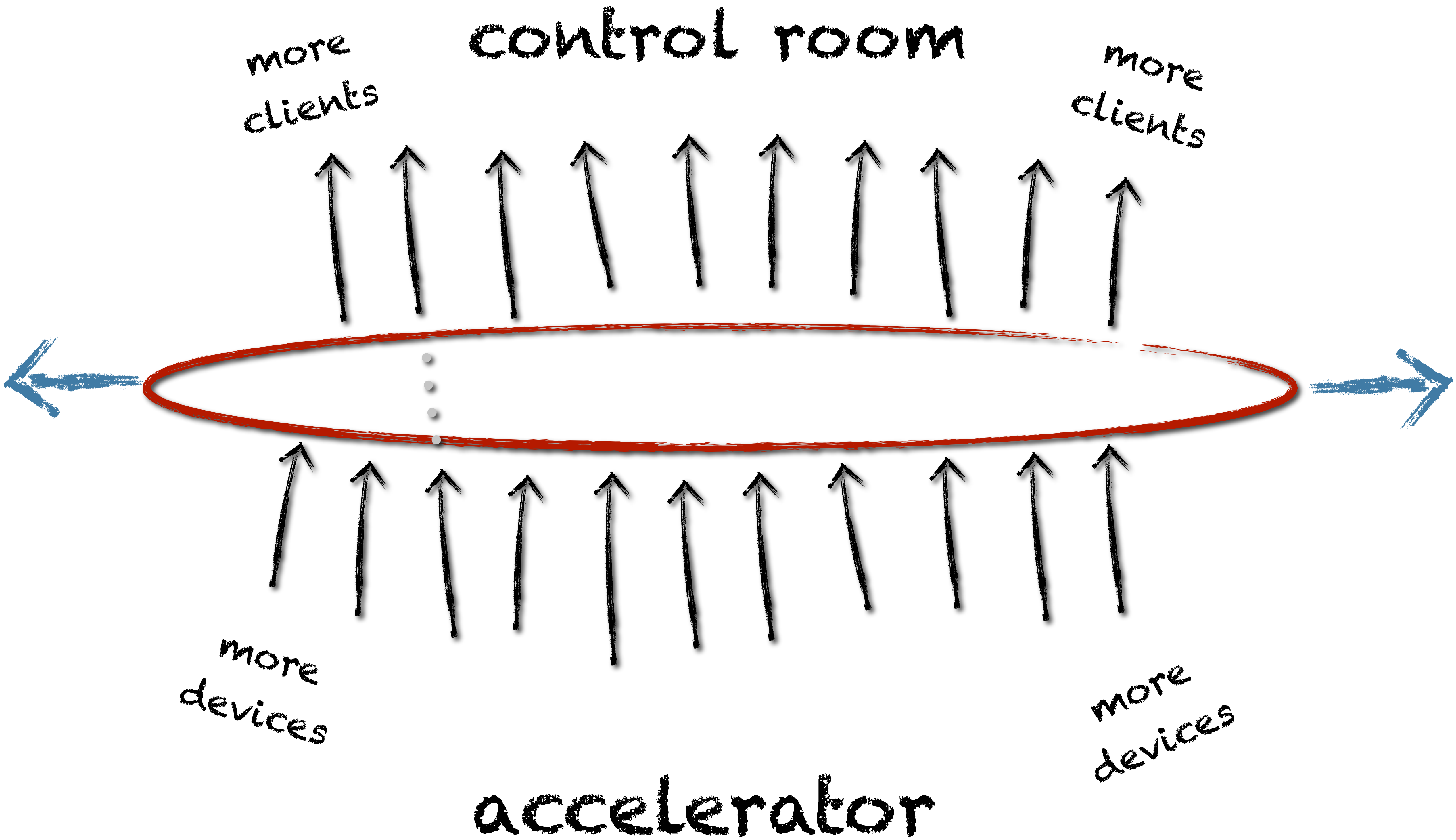
“the standard model”



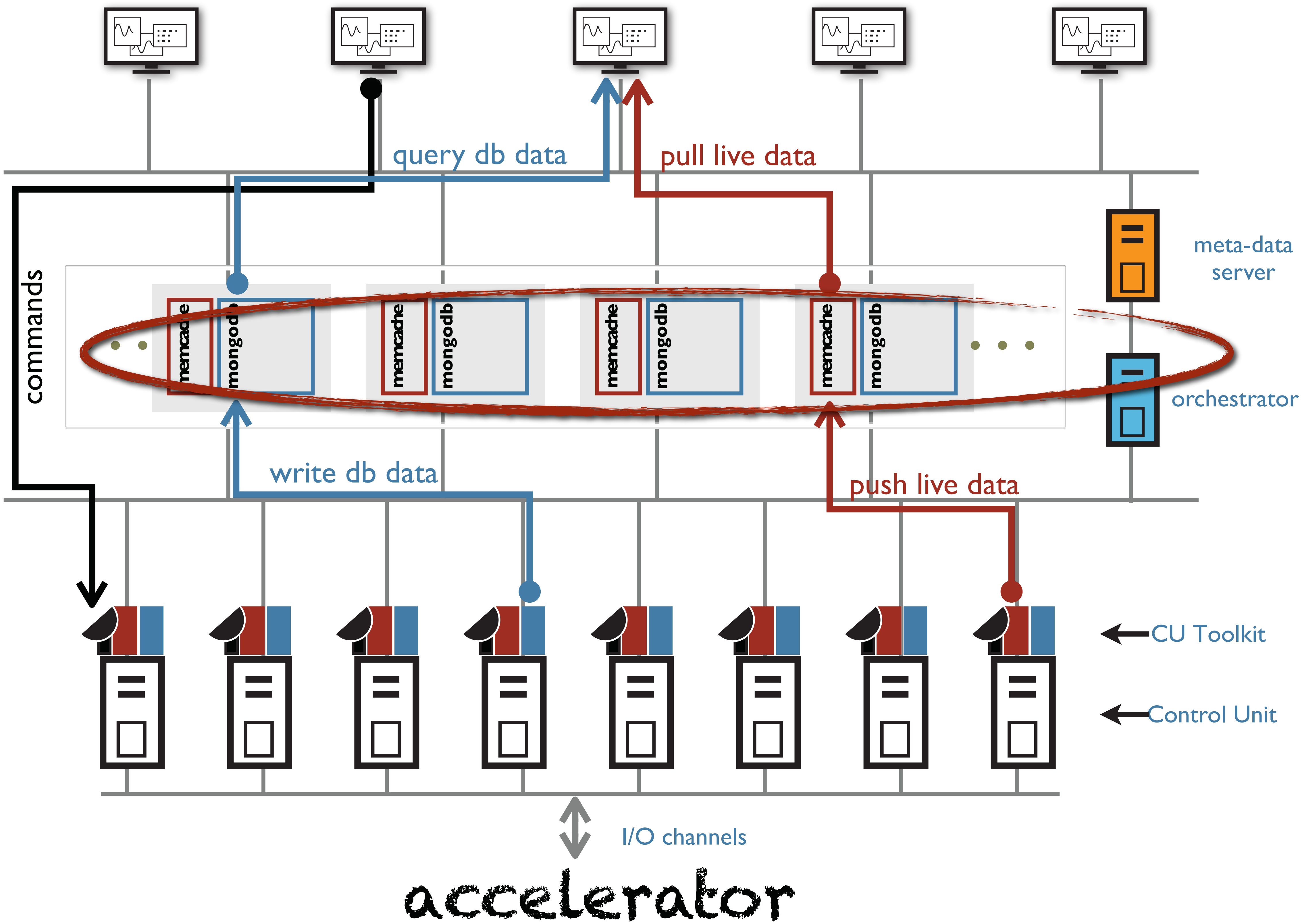
the new paradigm



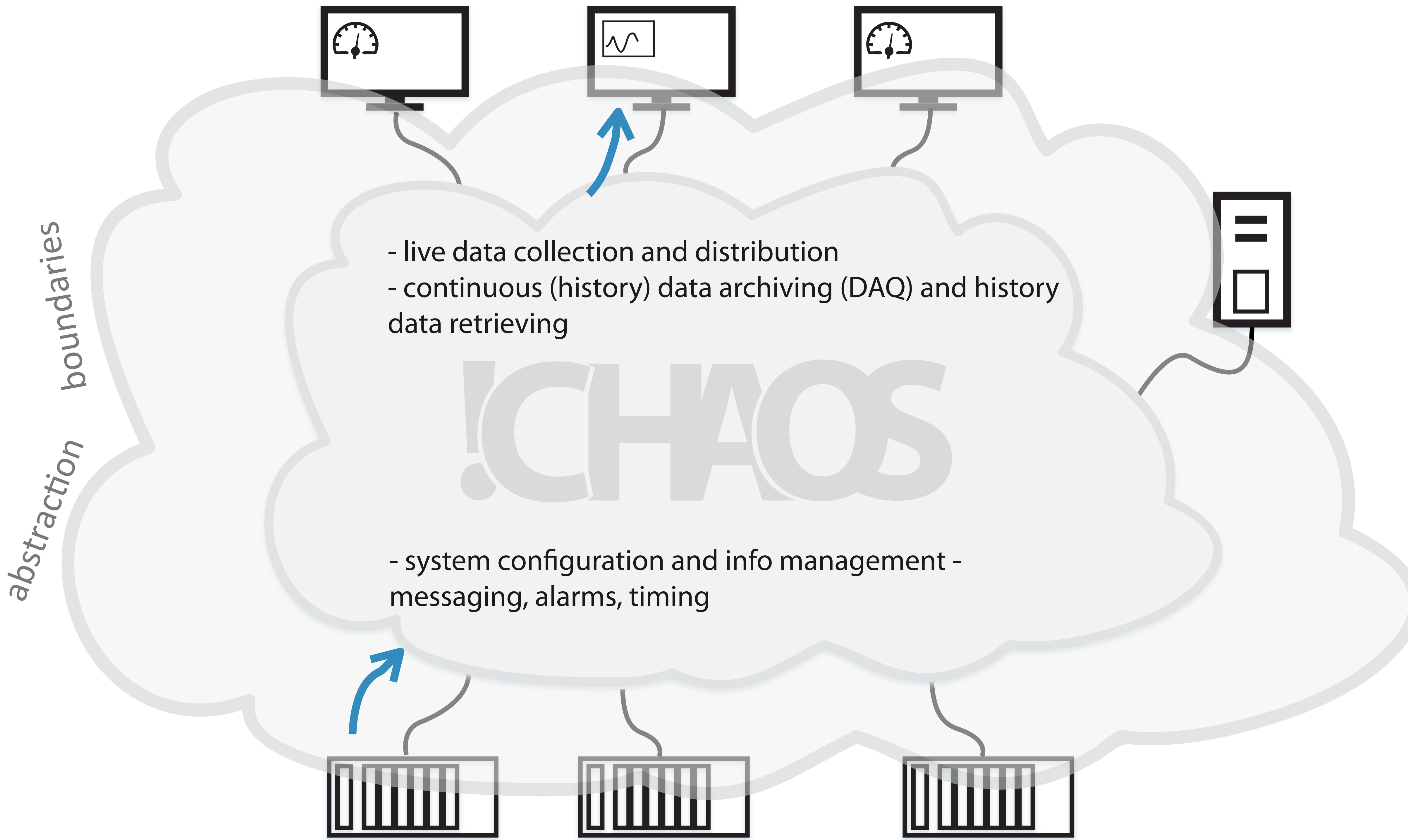
the new paradigm: scalability eliminates bottleneck



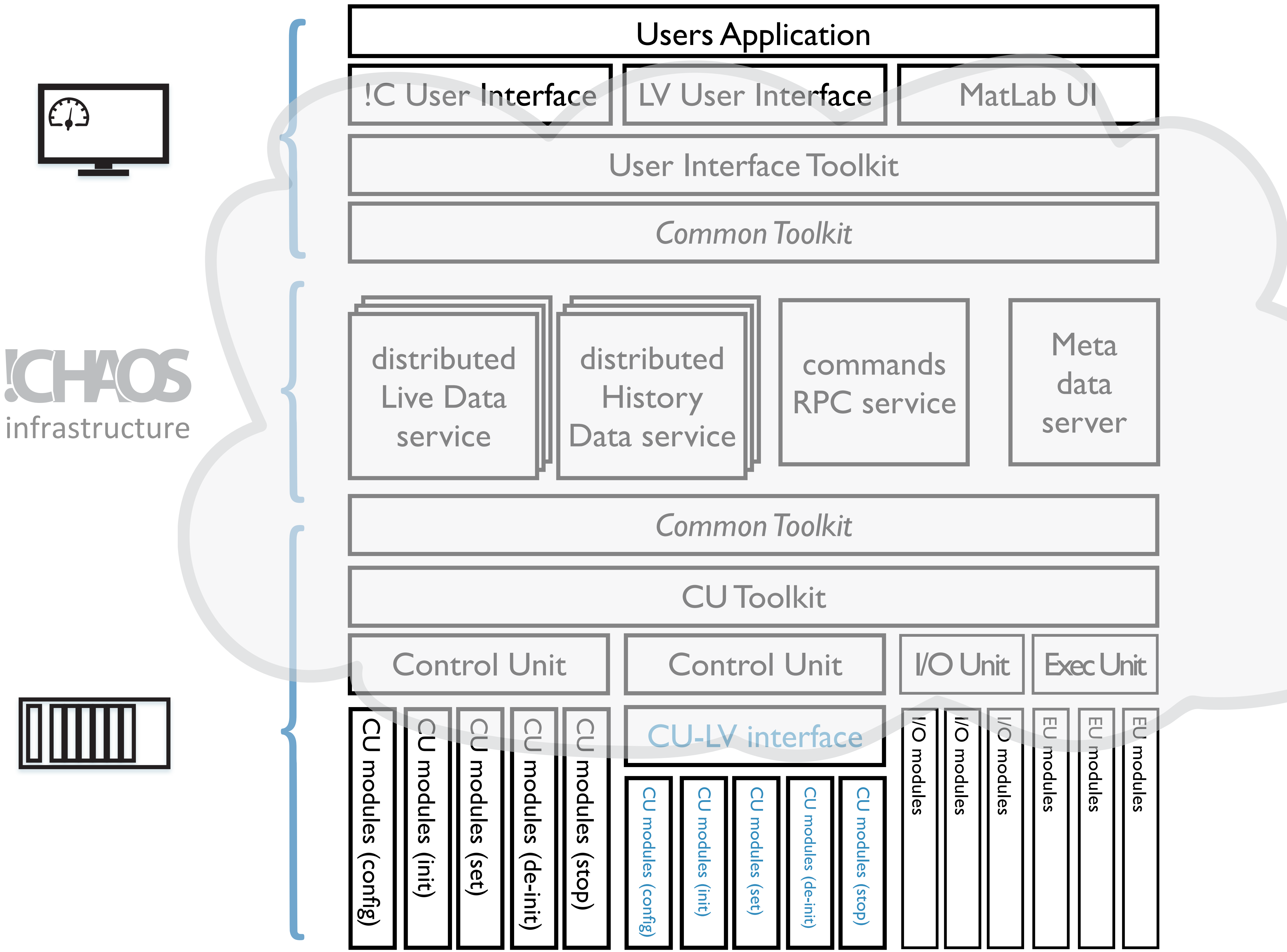
!CHAOS Control System topology



“data consumer” clients



“data producer” clients



Users Application
!C User Interface
LV User Interface
MatLab UI
User Interface Toolkit
Common Toolkit
!CHAOS
infrastructure
distributed Live Data service
distributed History Data service
commands RPC service
Meta data server
Common Toolkit
CU Toolkit
Control Unit
Control Unit
I/O Unit
Exec Unit
CU modules (config)
CU modules (init)
CU modules (set)
CU modules (de-init)
CU modules (stop)
CU-LV interface
CU modules (config)
CU modules (init)
CU modules (set)
CU modules (de-init)
CU modules (stop)
I/O modules
I/O modules
I/O modules
EU modules
EU modules
EU modules

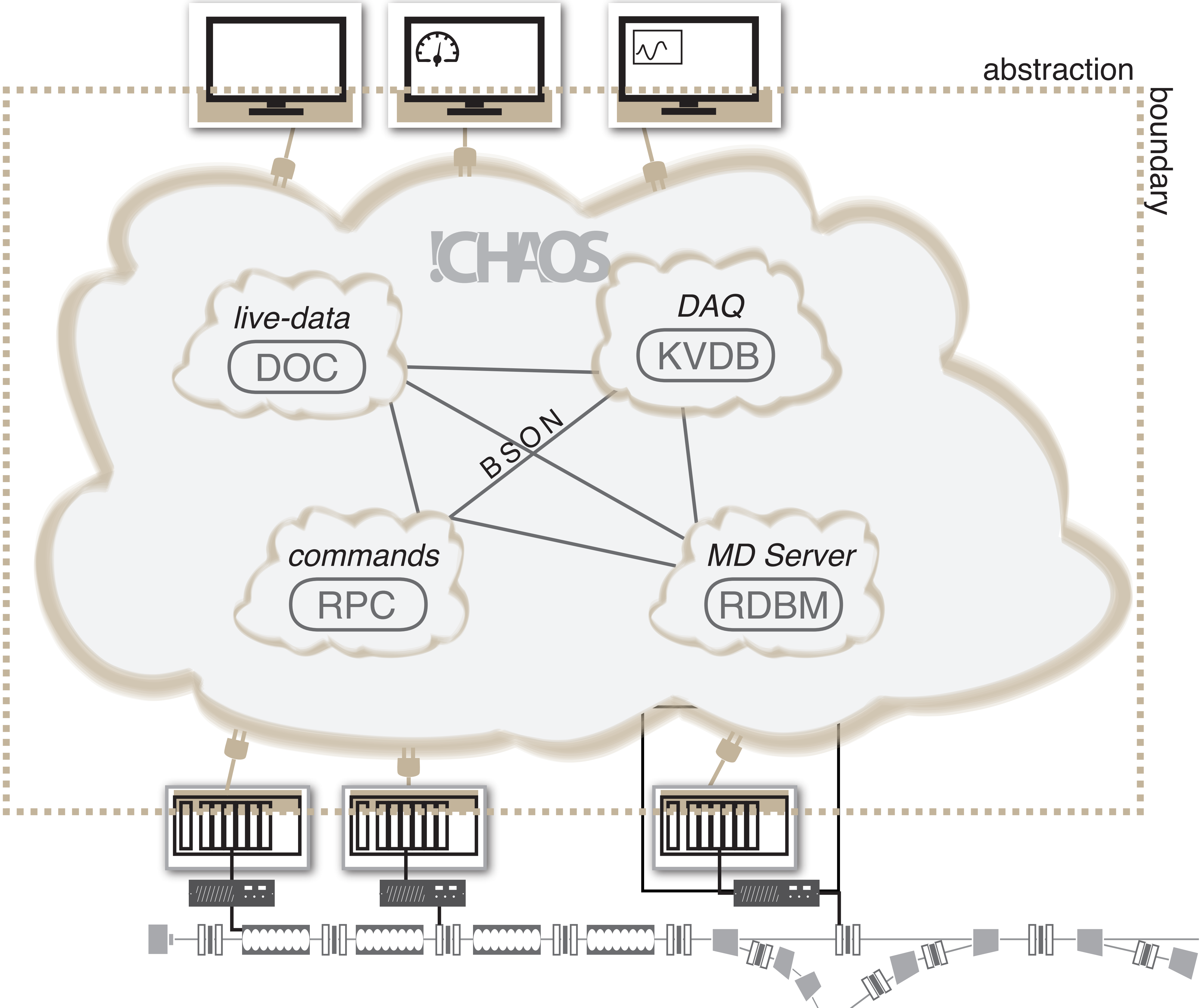
!CHAOS keywords

scalability
- ✓ of performance
- ✓ and size

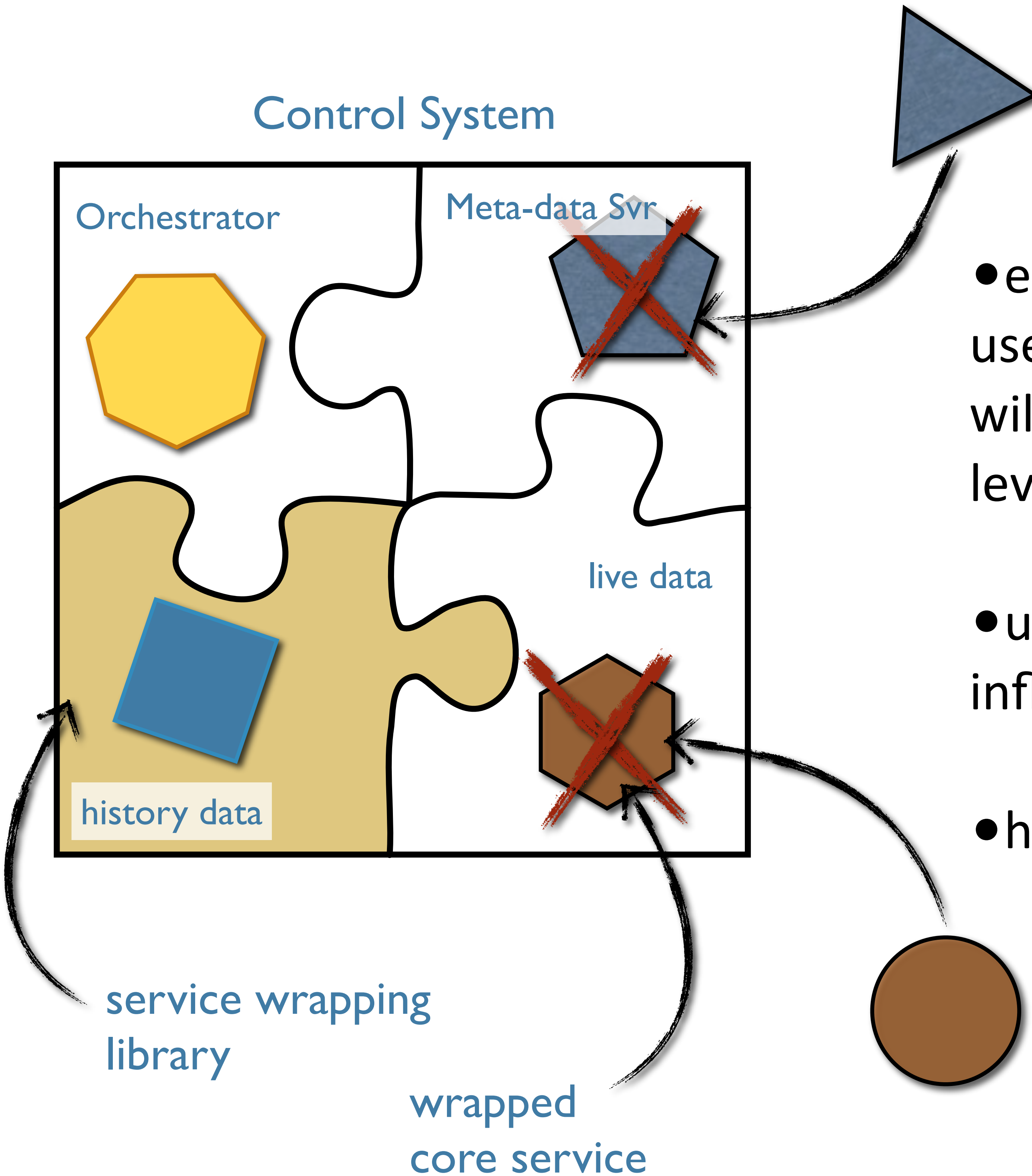
abstraction
- ✓ of services
- ✓ and device

integration
- ✓ all services provided by the SC

fast, easy and modular customization
- ✓ devices' drivers development only


abstraction
boundary
!CHAOS
live-data
DOC
DAQ
KVDB
BSON
commands
RPC
MD Server
RDBM

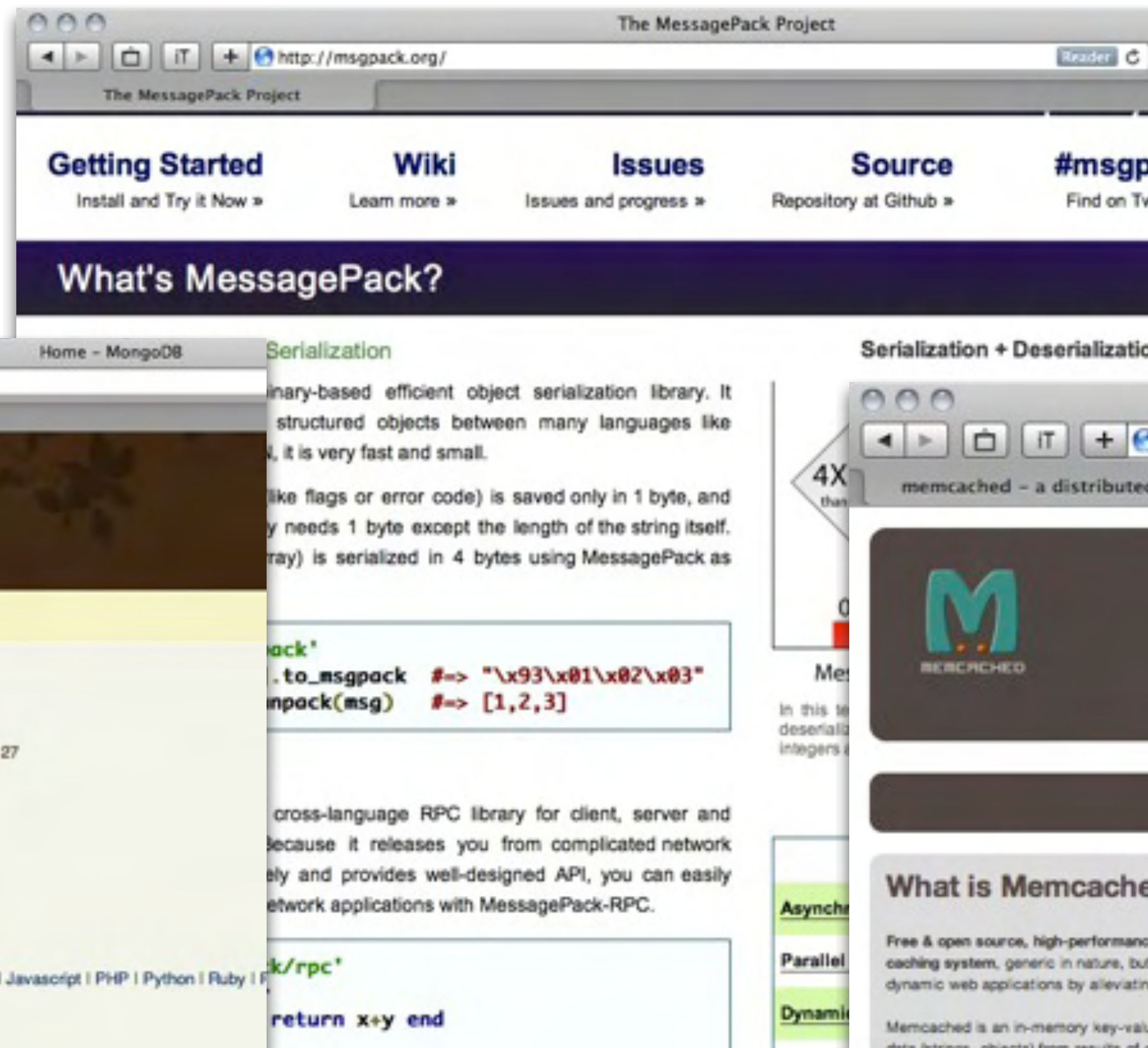
Abstraction of components



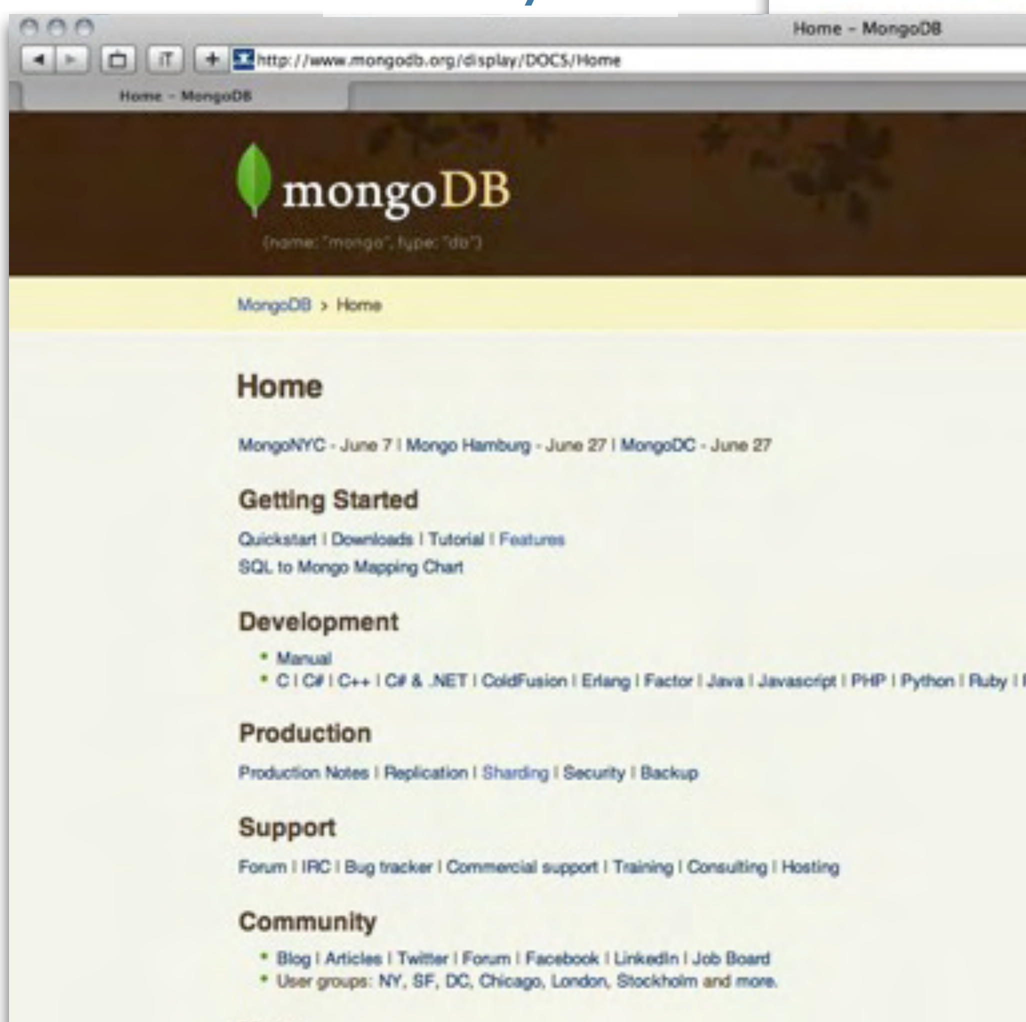
- each service isn't directly offered to users; glueing and wrapping routines will be developed to provide an high level of abstraction
- updates of core services doesn't influence the user applications
- higher flexibility in defining API

core services

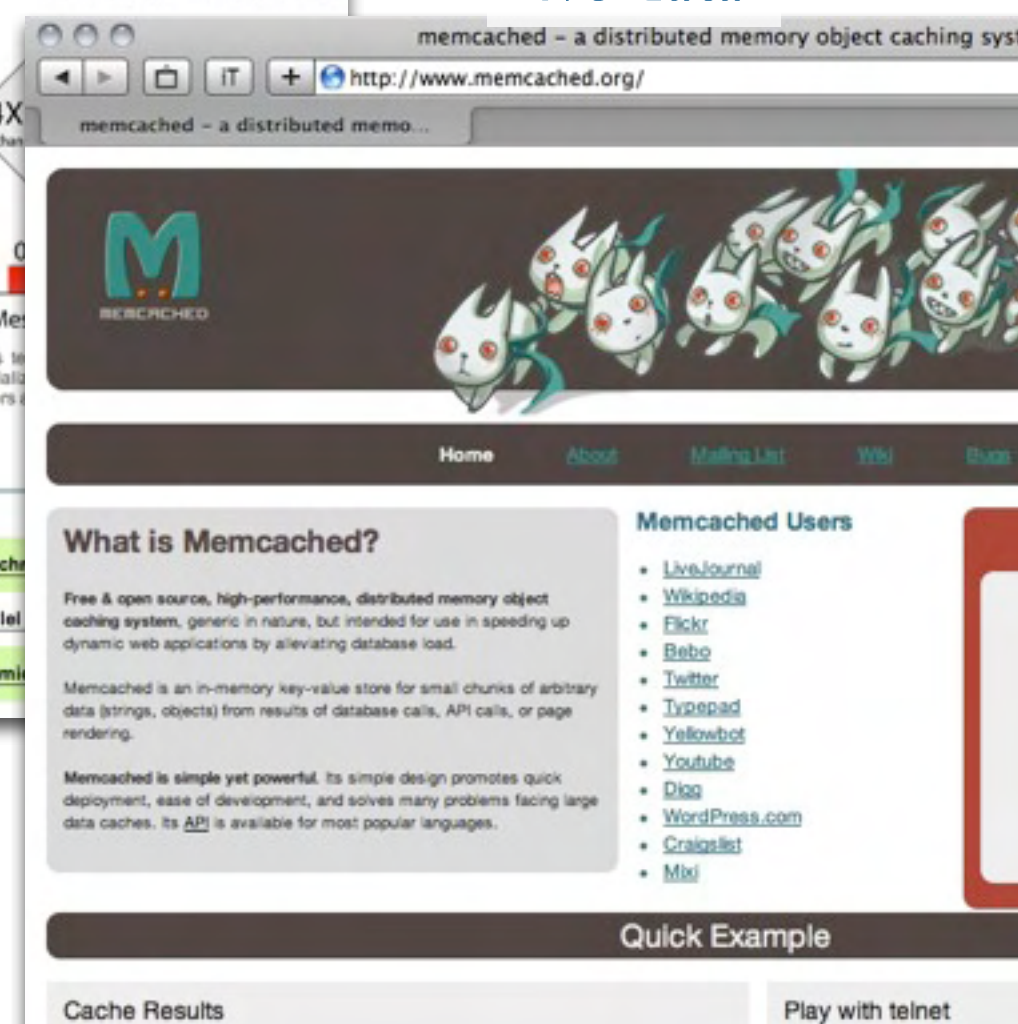
commands/events



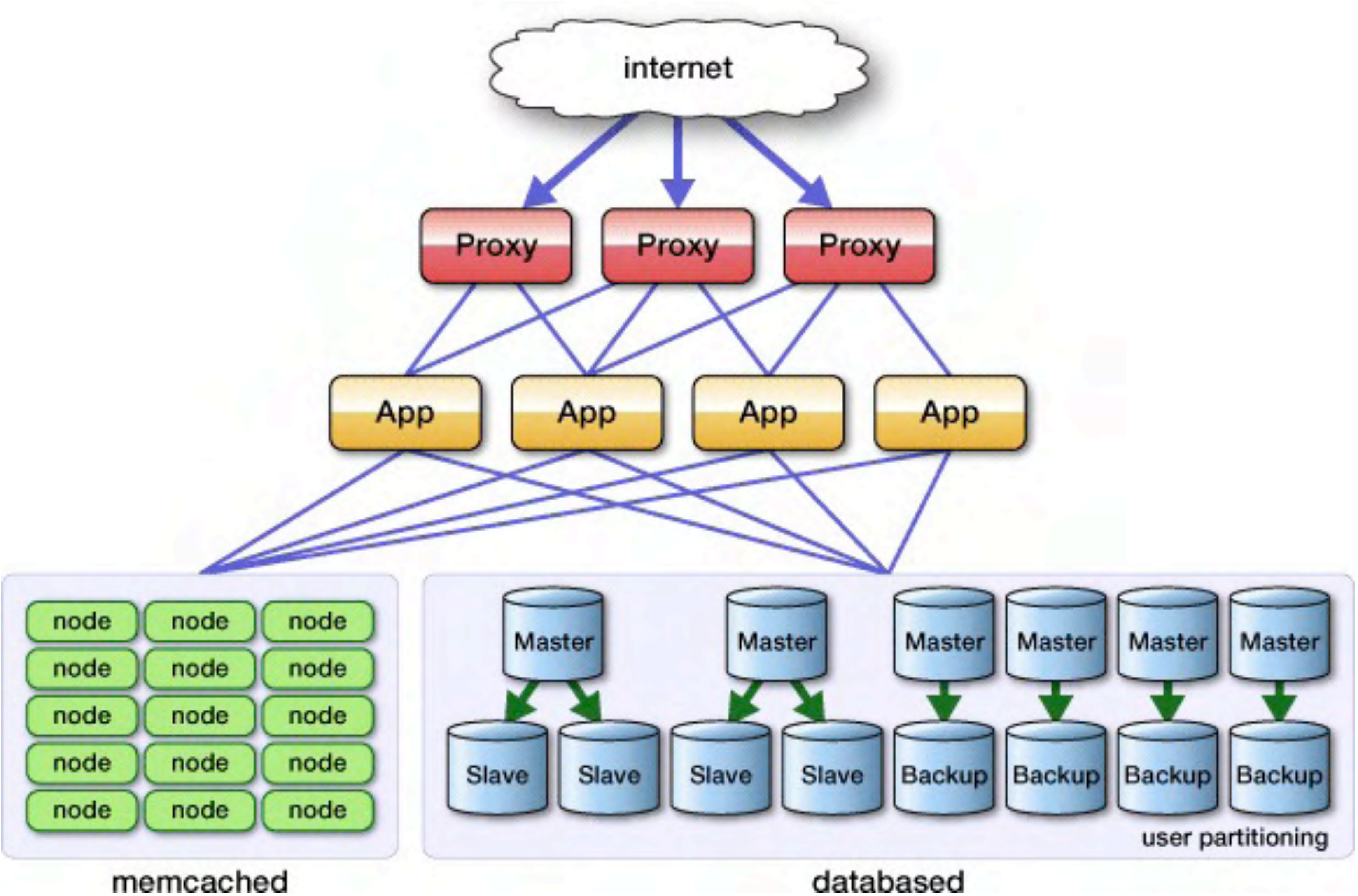
history db



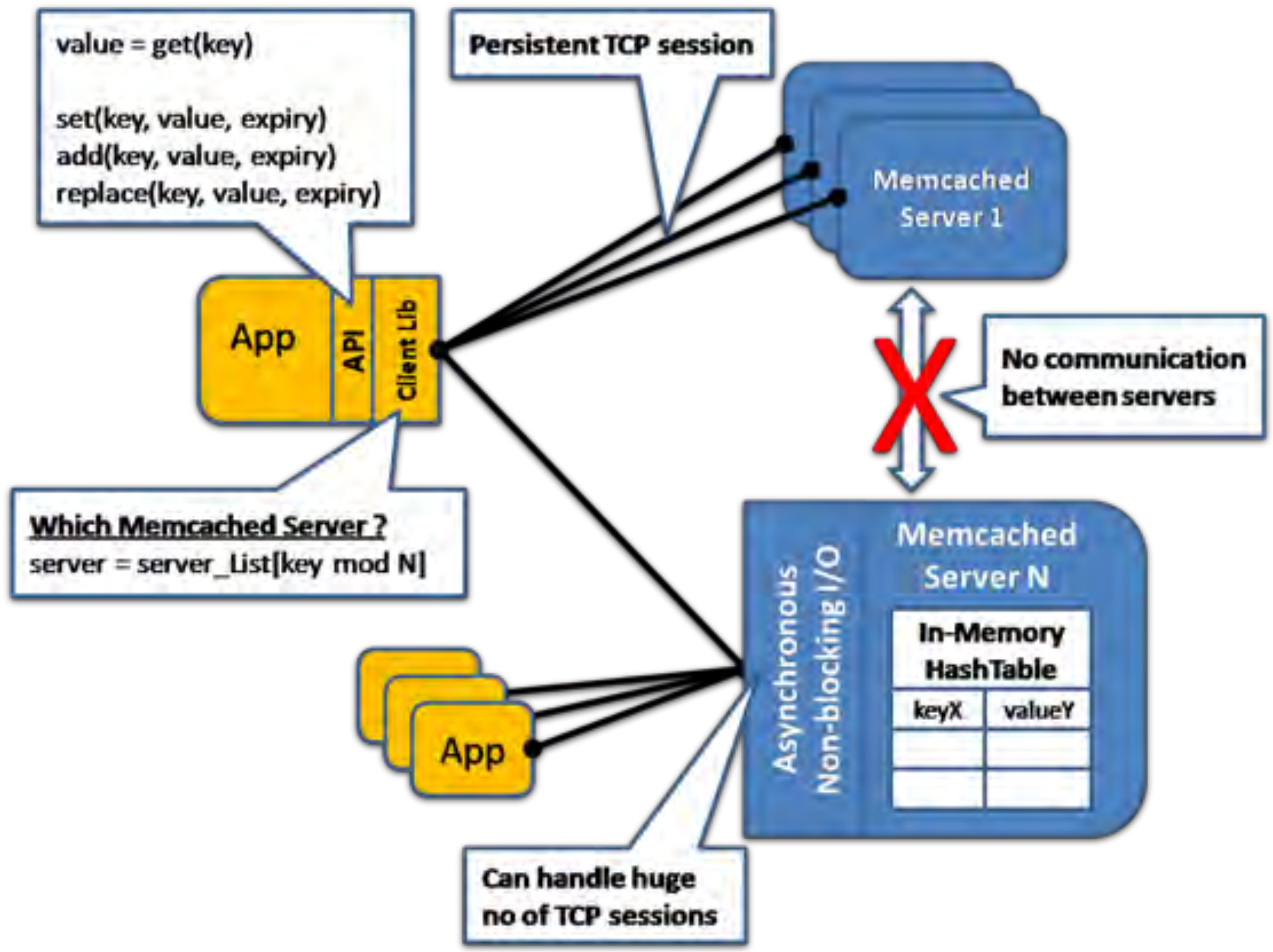
live data



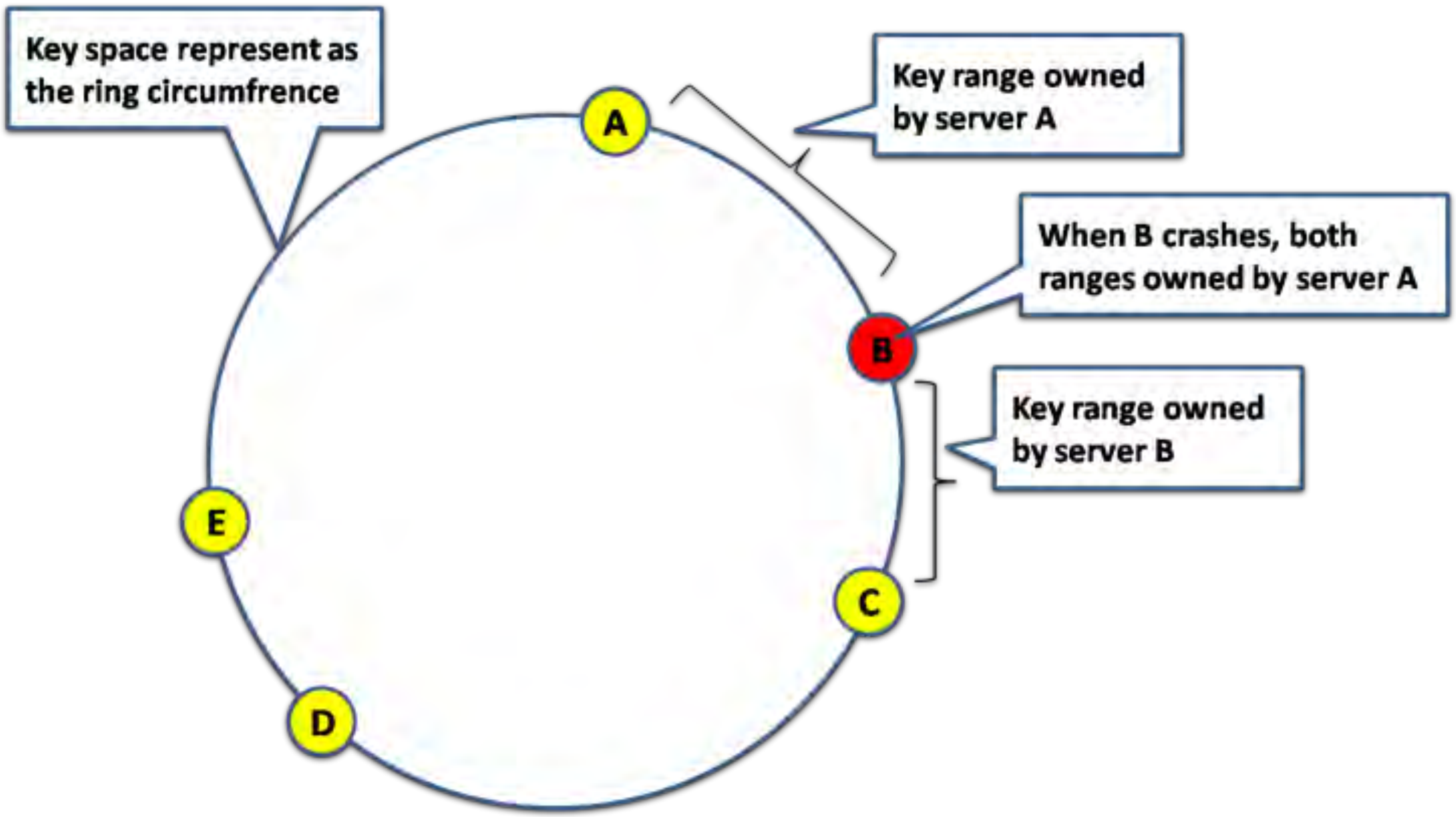
memcached


internet
Proxy
Proxy
Proxy
App
App
App
App
node node node
node node node
node node node
node node node
node node node
memcached
Master
Master
Master
Master
Master
Master
Slave
Slave
Slave
Slave
Backup
Backup
Backup
Backup
user partitioning
databased

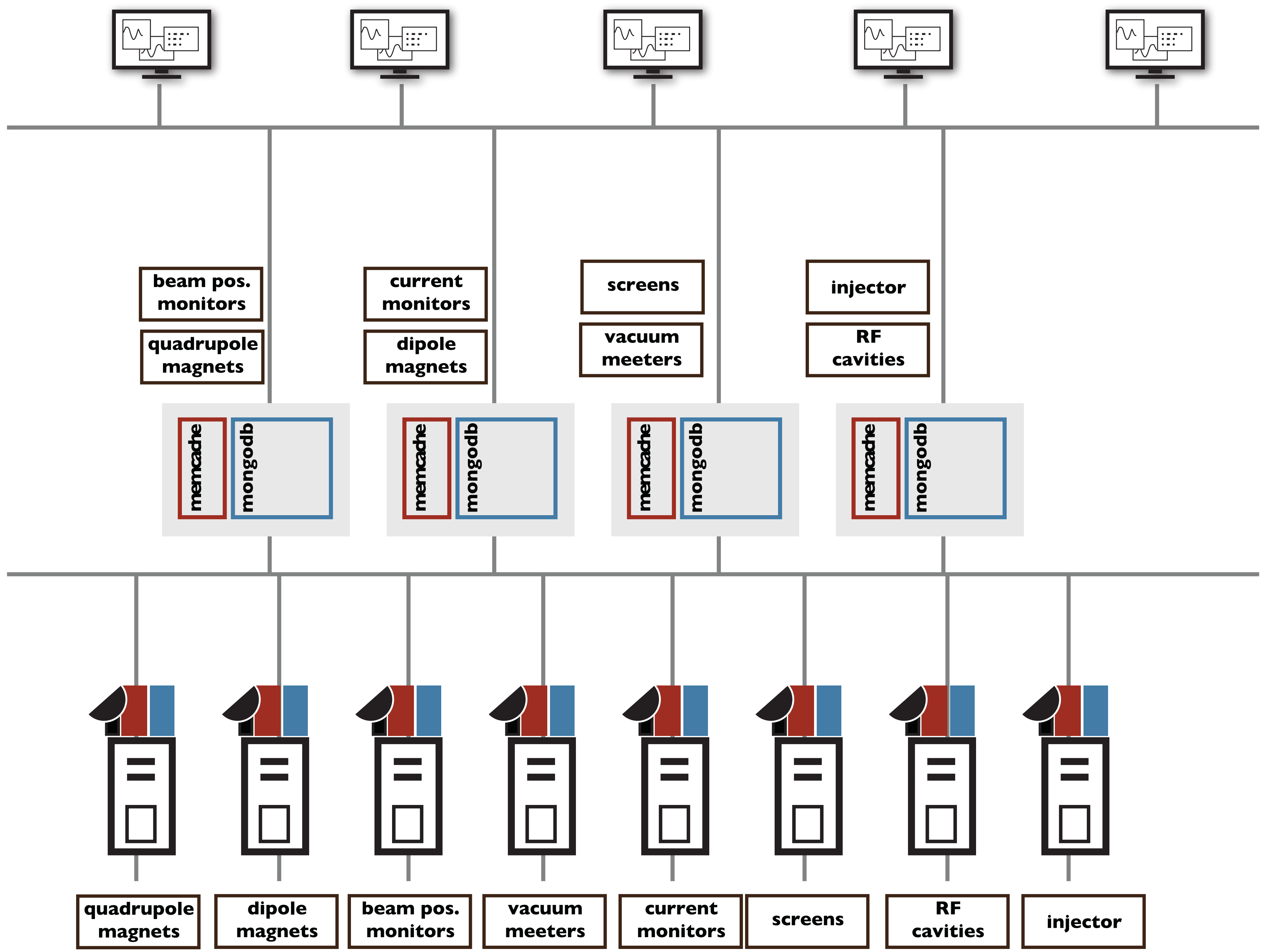
memcached


value = get(key)
set(key, value, expiry)
add(key, value, expiry)
replace(key, value, expiry)
Persistent TCP session
Memcached
Server 1
App
API
Client Lib
No communication
between servers
Which Memcached Server ?
server = server_List[key mod N]
Memcached
Server N
Asynchronous
Non-blocking I/O
In-Memory
HashTable
keyX
valueY
App
Can handle huge
no of TCP sessions

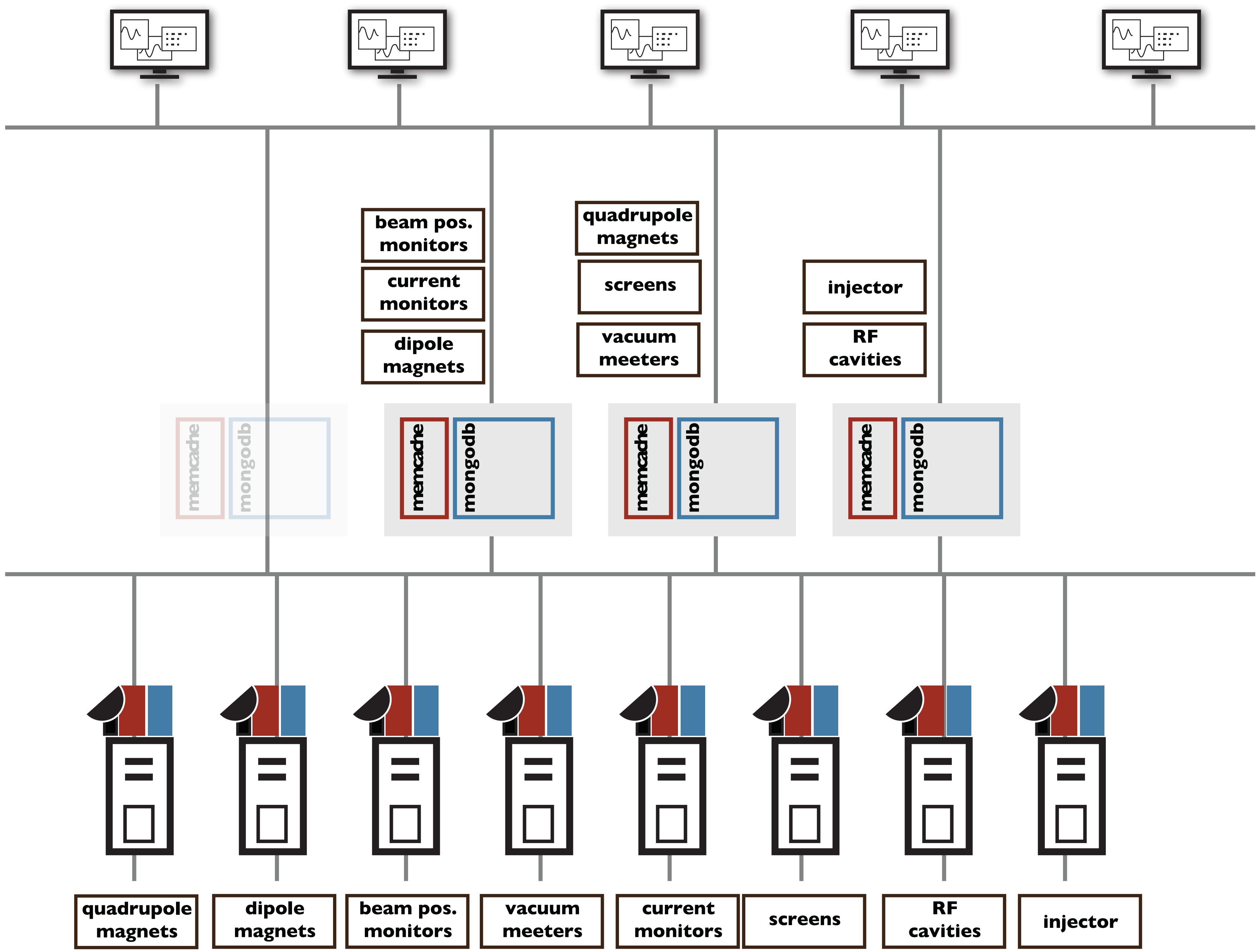
memcached


Key space represent as the ring circumfrence
A
Key range owned by server A
When B crashes, both ranges owned by server A
B
Key range owned by server B
C
D
E

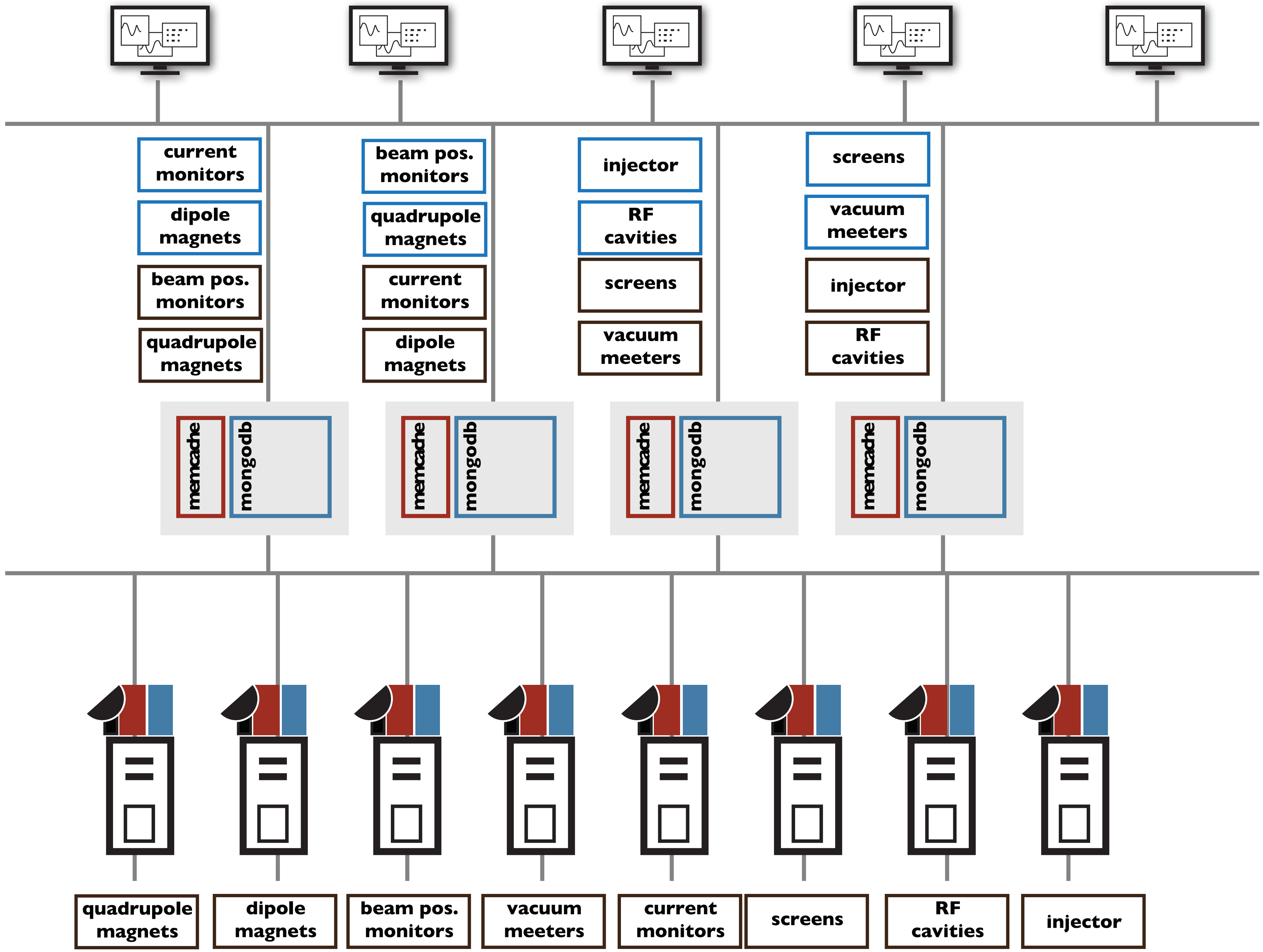
memcached used for live-data



memcached - failover

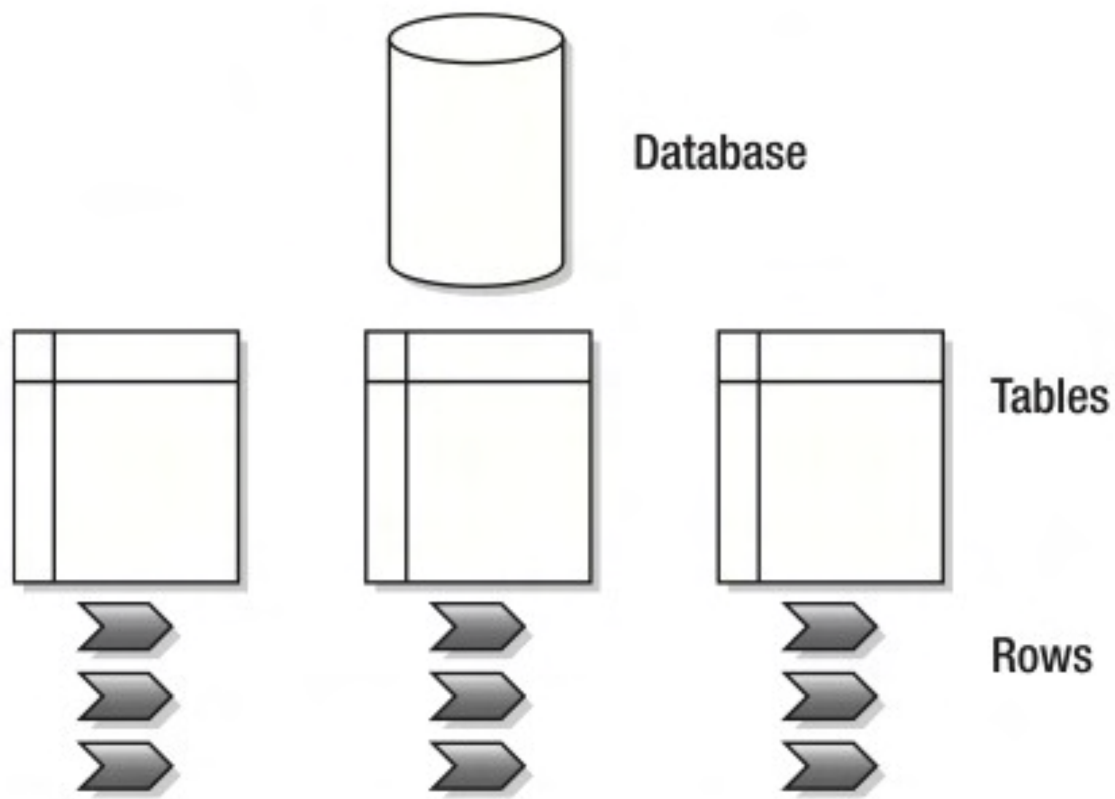


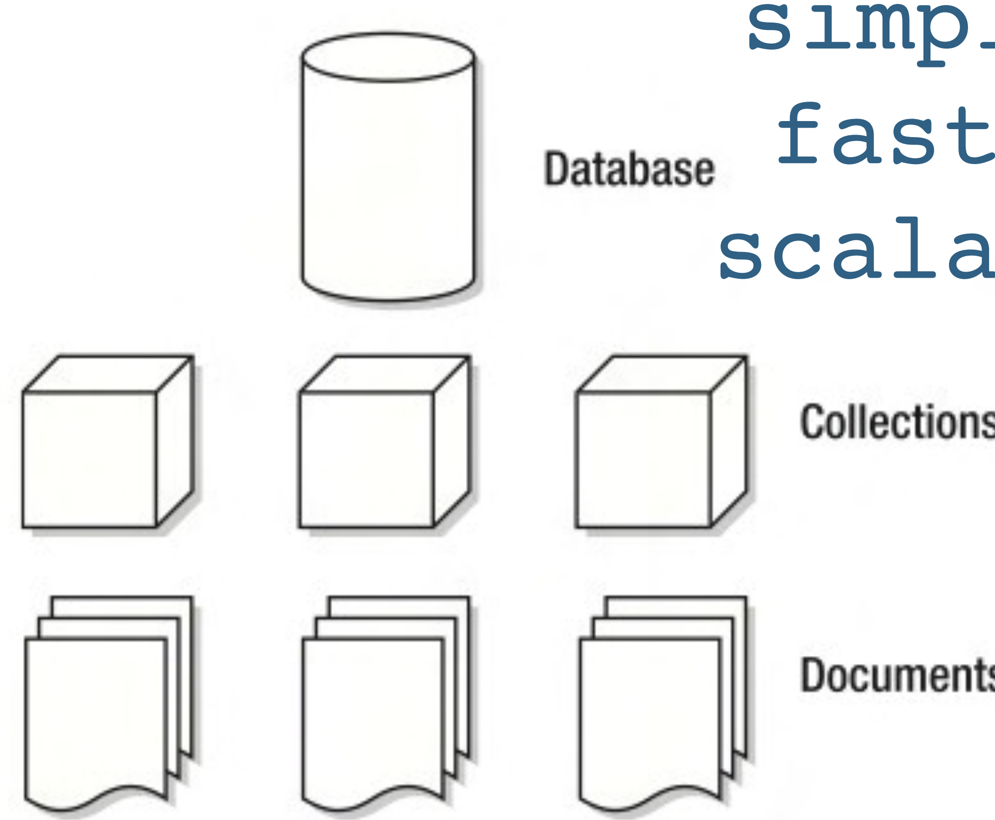
memcached - failover&mirroring



No-SQL key/value DB (ex. MongoDB)

simpler
faster
scalable





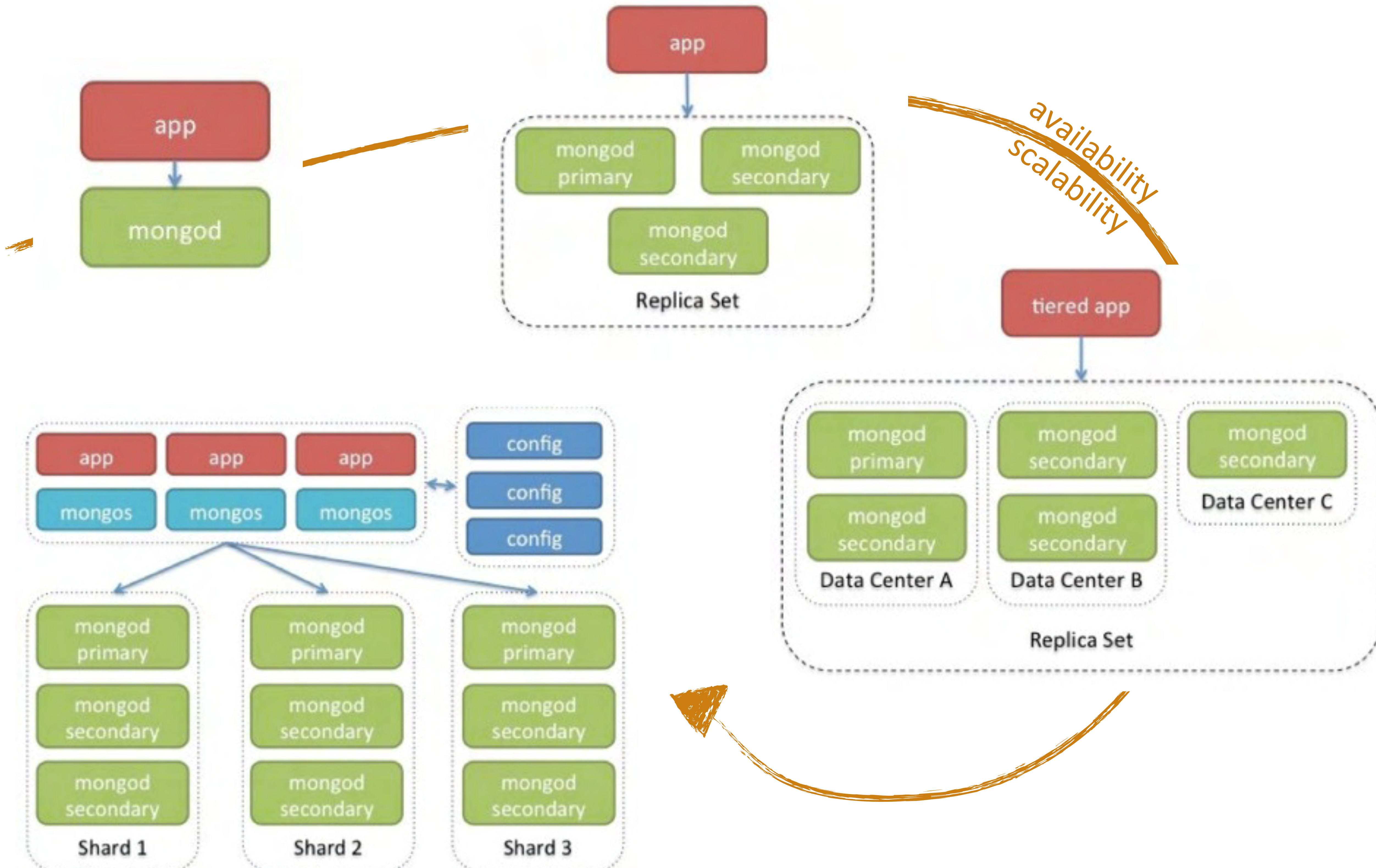
RDBM

l_media
  l_cds
    l_id, artist, title, genre, releasedate
  l_ cd_tracklists
    l_cd_id, songtitle, length

key/value DB

l_media
  l_items
    l_<document>

No-SQL key/value DB (ex. MongoDB)



No-SQL key/value DB (ex. MongoDB)

CSV

```
Surname, Forename, Phone Number
Membrey, Peter, +852 1234 5678
Thielen, Wouter, +81 1234 5678
```

key/value data storage: JSON serialization

JSON

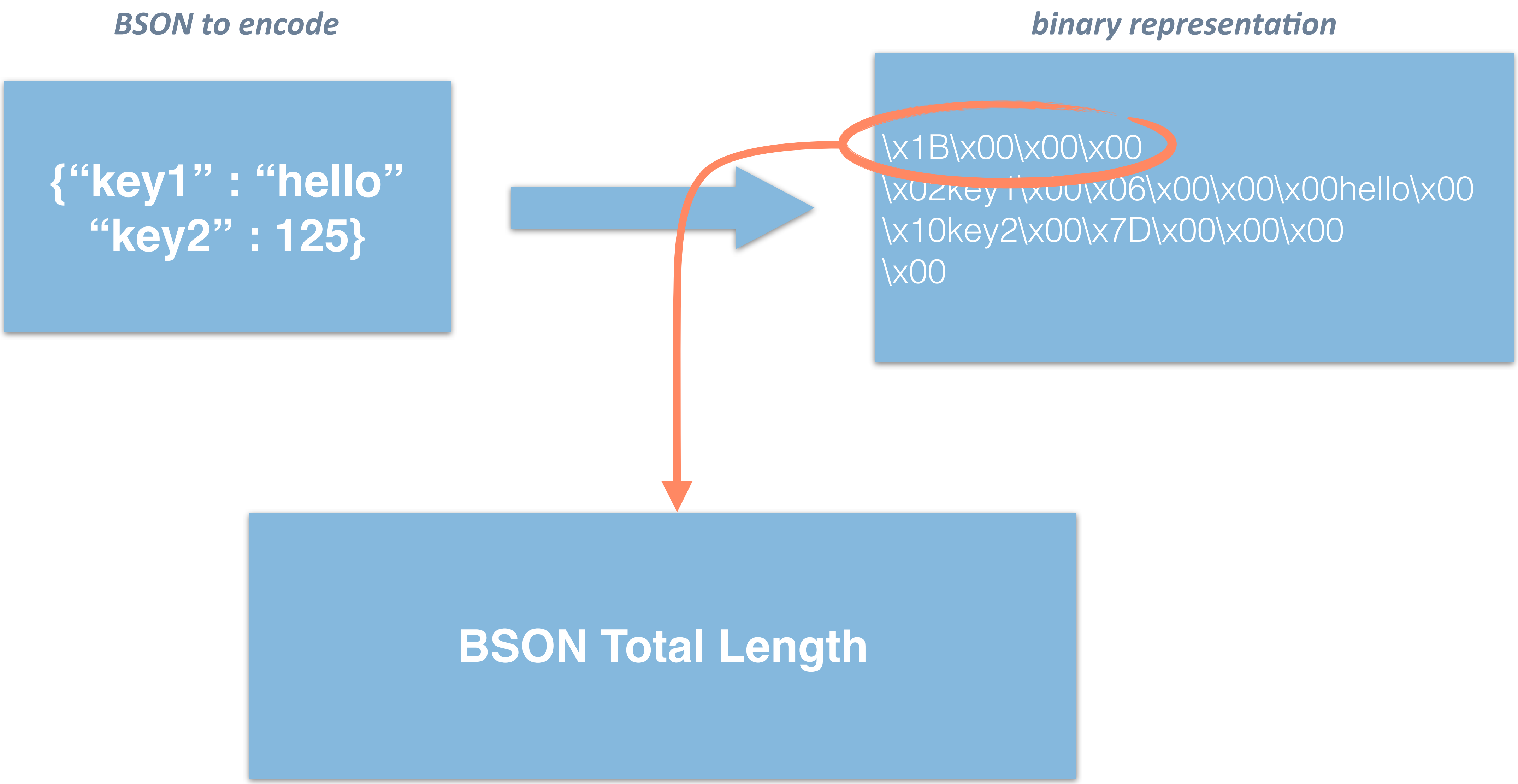
```
{
    "forename": "Peter",
    "surname": "Membrey",
    "phone_numbers": [
        "+852 1234 5678",
        "+44 1234 565 555"
    ]
}
```

```
{
    "forename": "Peter",
    "surname": "Membrey",
    "numbers": [
        {
            "phone": "+852 1234 5678"
}, {
            "fax": "+44 1234 565 555"
        }
] }
```

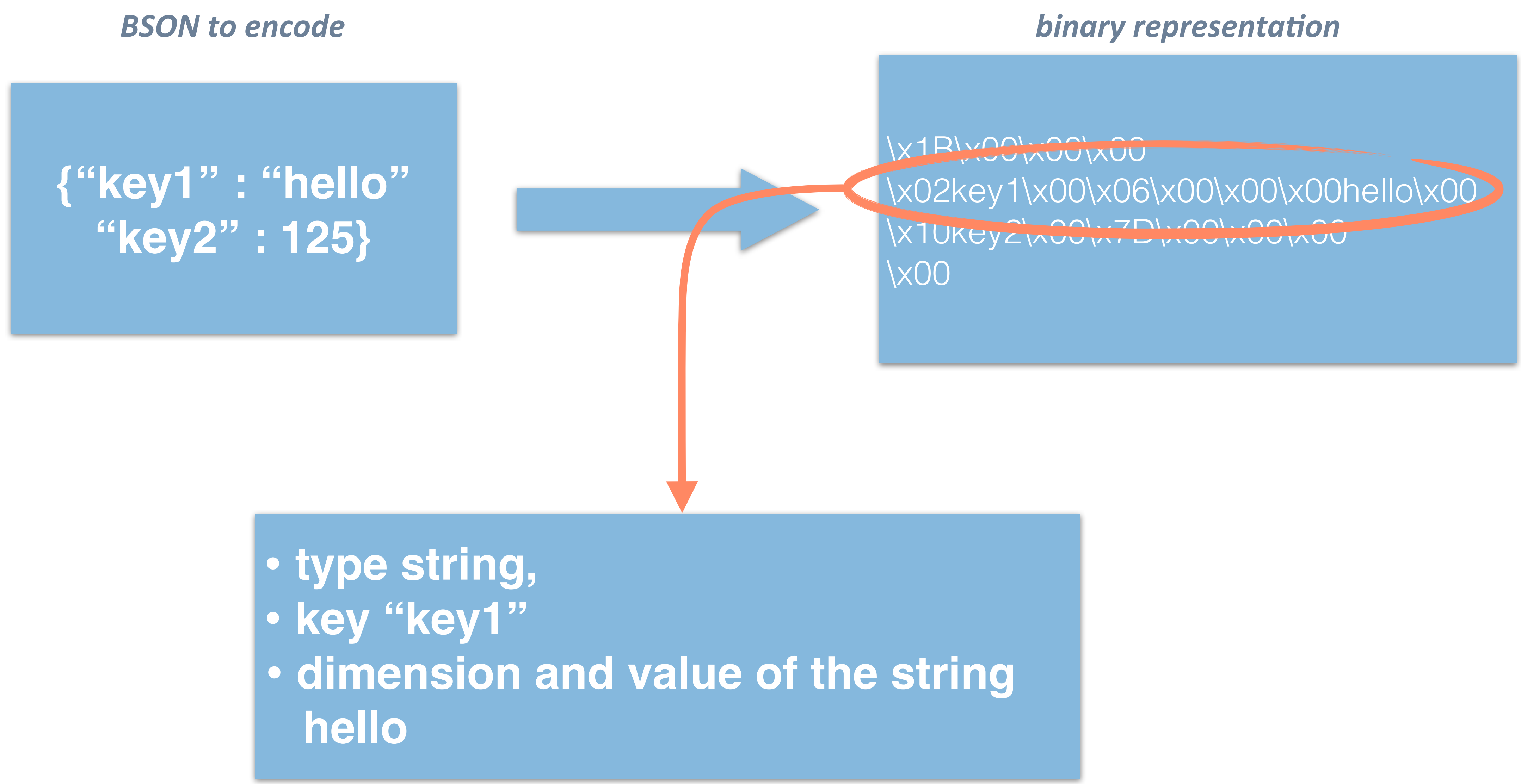
BSON

- aim to be binary version of JSON
- key - value syntax
- every pair is serialized bringing:
  - key name
  - type and dimension
  - value

BSON


BSON to encode
{“key1” : “hello”
“key2” : 125}
binary representation
\x1B\x00\x00\x00
\x02key1\x00\x06\x00\x00\x00hello\x00
\x10key2\x00\x7D\x00\x00\x00
\x00
BSON Total Length

BSON


BSON to encode
{"key1" : "hello"
"key2" : 125}
binary representation
\x1B\x00\x00\x00
\x02key1\x00\x06\x00\x00\x00hello\x00
\x10key2\x00\x7D\x00\x00\x00
\x00
• type string,
• key "key1"
• dimension and value of the string hello

BSON

BSON to encode

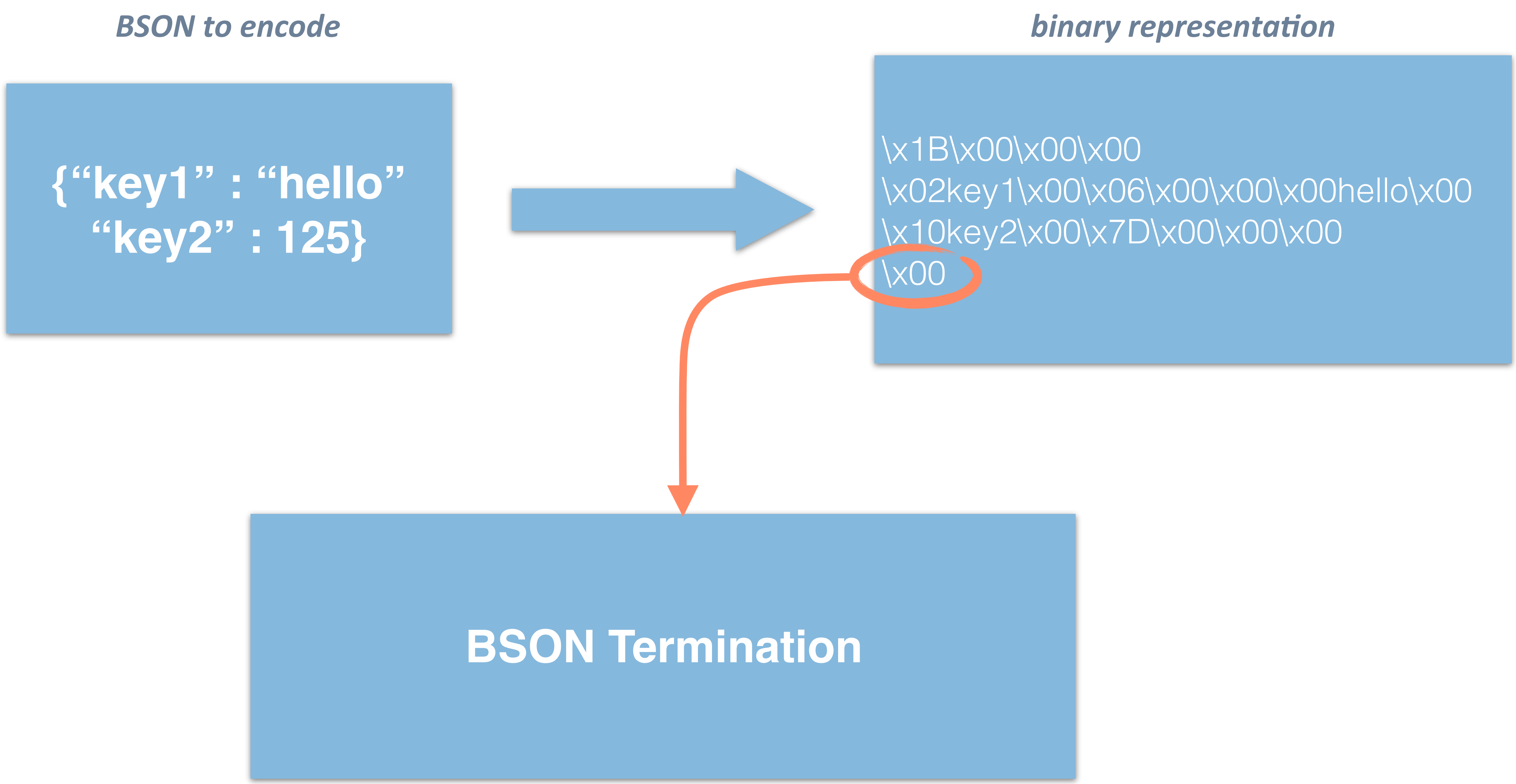
{"key1" : "hello"
"key2" : 125}

binary representation

\x1B\x00\x00\x00
\x02key1\x00\x06\x00\x00\x00hello\x00
\x10key2\x00\x7D\x00\x00\x00
\x00

- type int32,
- key "key2"
- int32 value

BSON

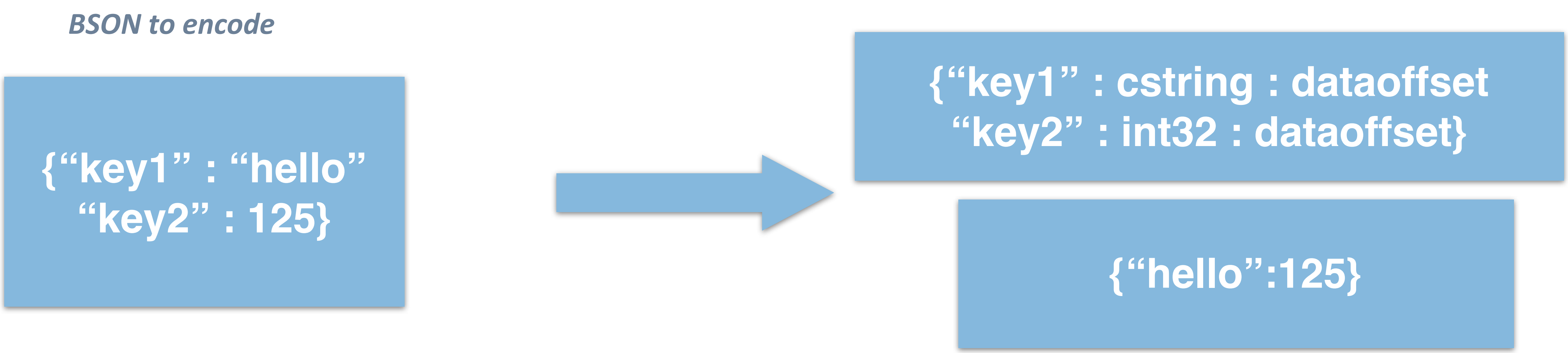


BSON to encode
binary representation
{"key1" : "hello"
"key2" : 125}
\x1B\x00\x00\x00
\x02key1\x00\x06\x00\x00\x00hello\x00
\x10key2\x00\x7D\x00\x00\x00
\x00
BSON Termination

BSON optimization

- **a possible optimization of BSON would be separating the** description **(keys, structures and types) from the** data **(binary representations of the values)**
- **the serialized data must contain all information for de-serialize and calculate**
    - **key offset**
    - **total length of the BSON**

BSON optimization



BSON optimization

{“key1” : cstring:offset
“key2” : int32:offset}

{“hello”:125}

getString(“key1”) == “hello”
getInt32(“key2”) == 125

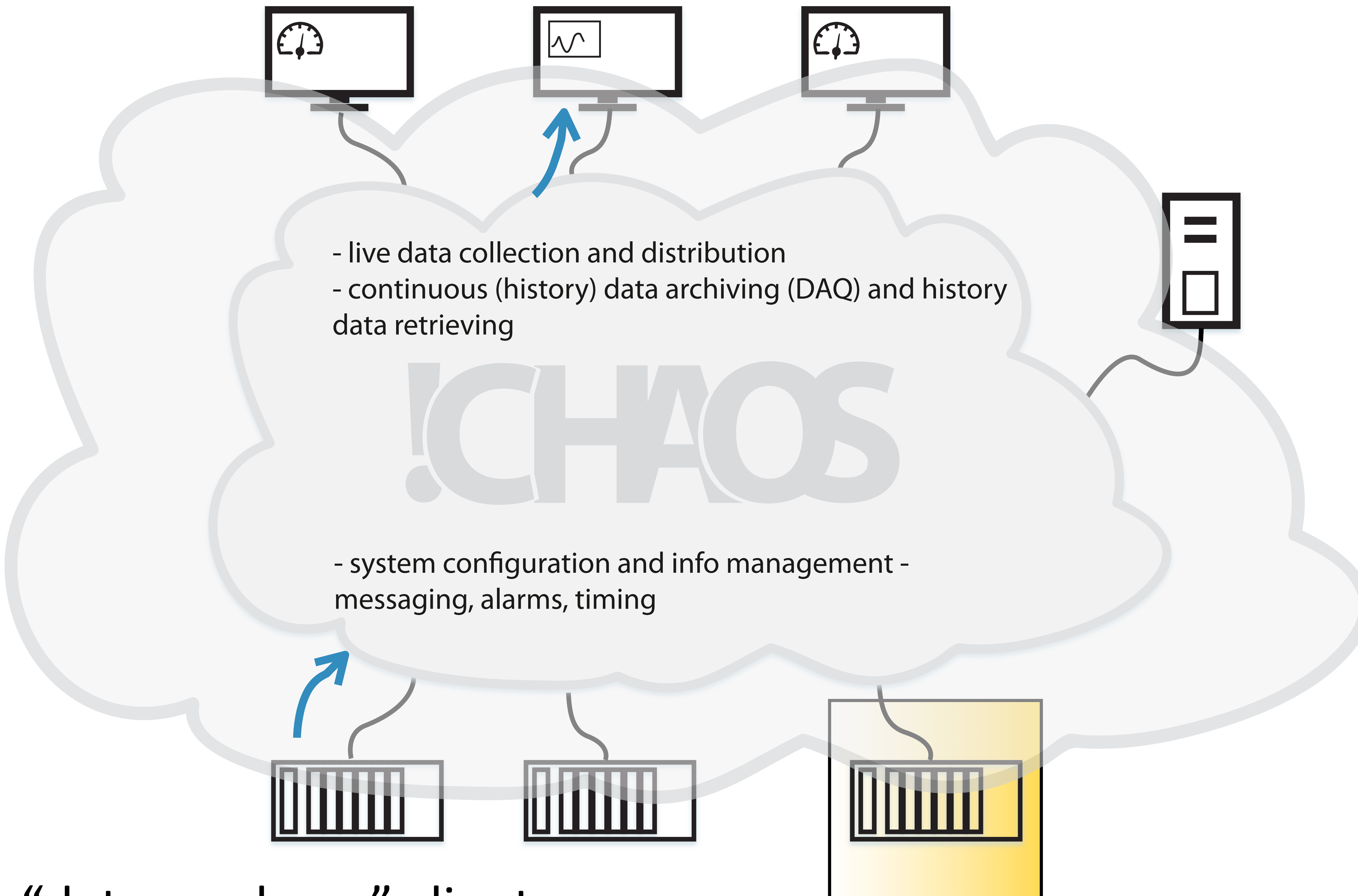
BSON optimization

{“key1” : cstring:offset
“key2” : int32:offset}

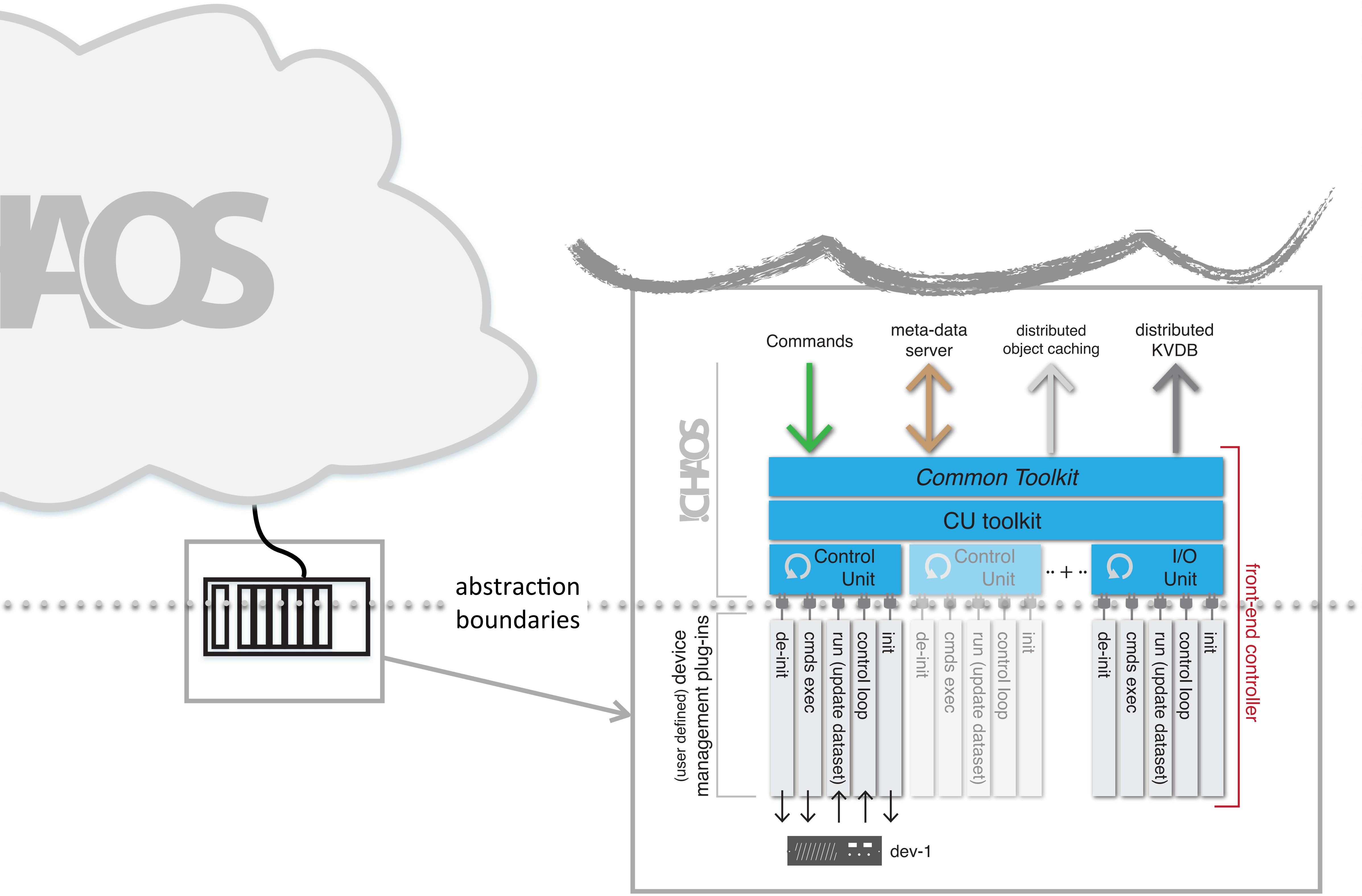
{“world”:228}

getString(“key1”) == “world”
getInt32(“key2”) == 128

“data consumer” clients



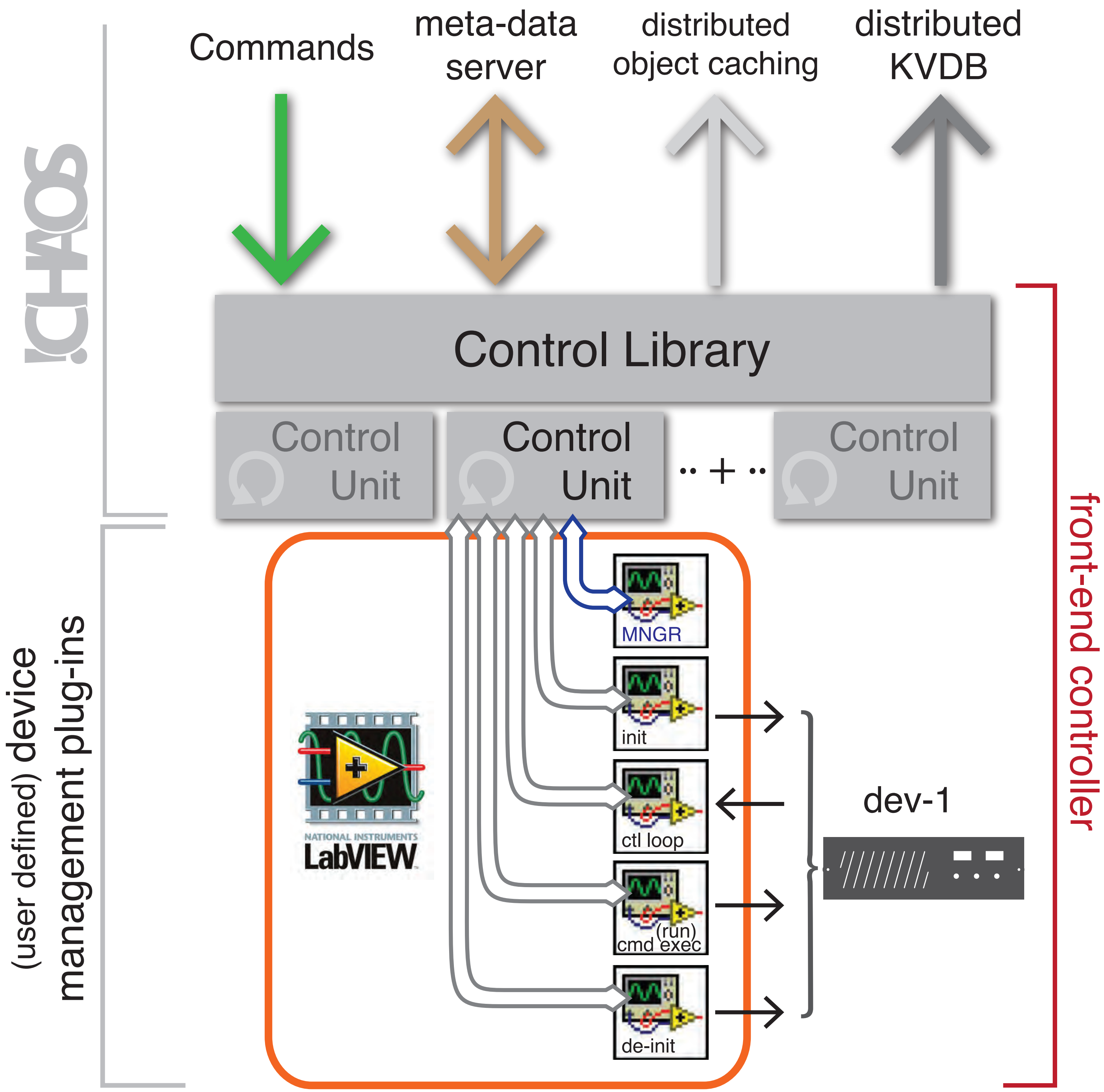
“data producer” clients

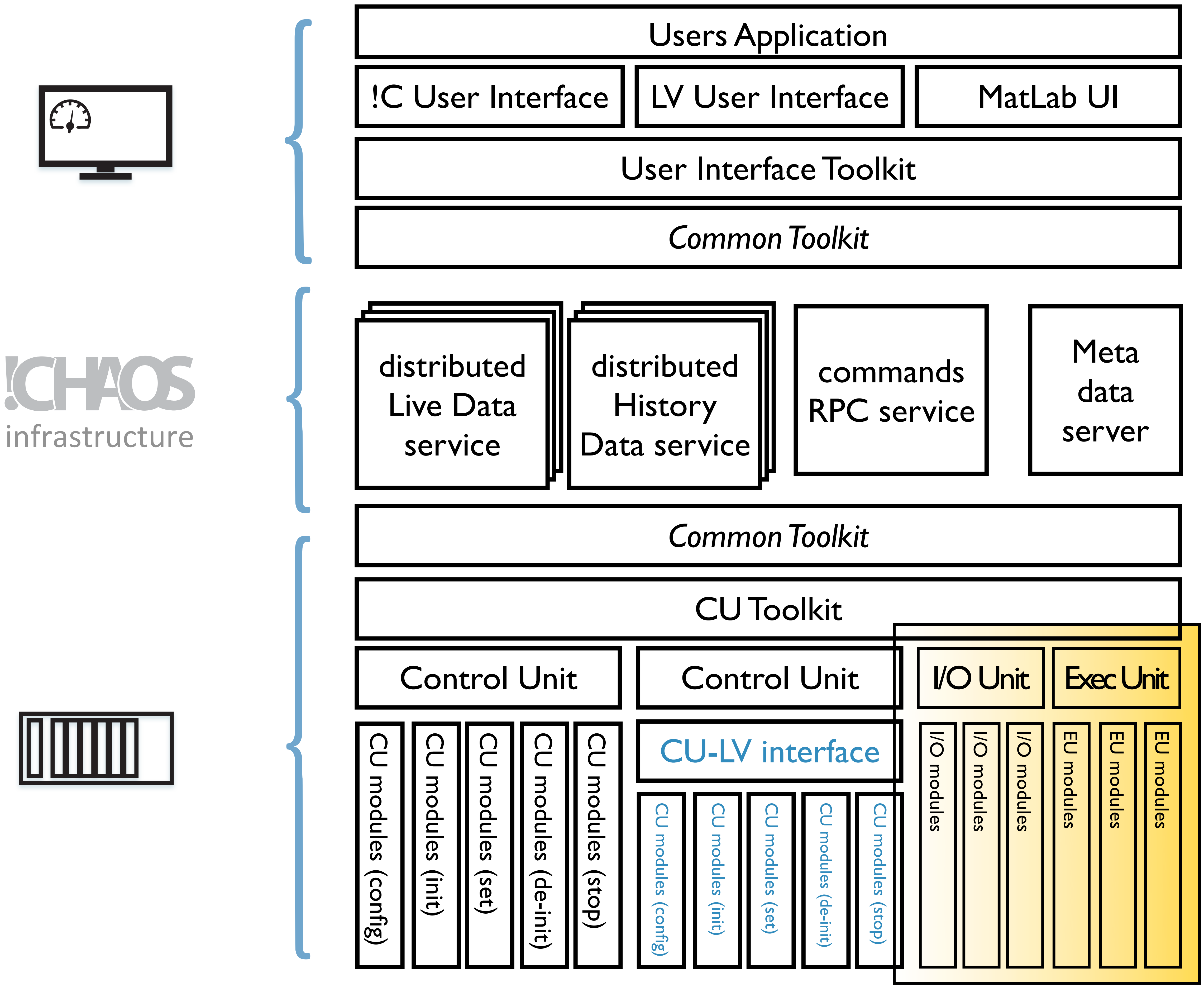

abstraction
boundaries
Commands
meta-data
server
distributed
object caching
distributed
KVDB
Common Toolkit
CU toolkit
Control
Unit
Control
Unit
I/O
Unit
front-end controller
(user defined) device
management plug-ins
de-init
cmds exec
run (update dataset)
control loop
init
dev-1

L. Foggetta

LabVIEW-CHAOS integration

(a very basic solution)

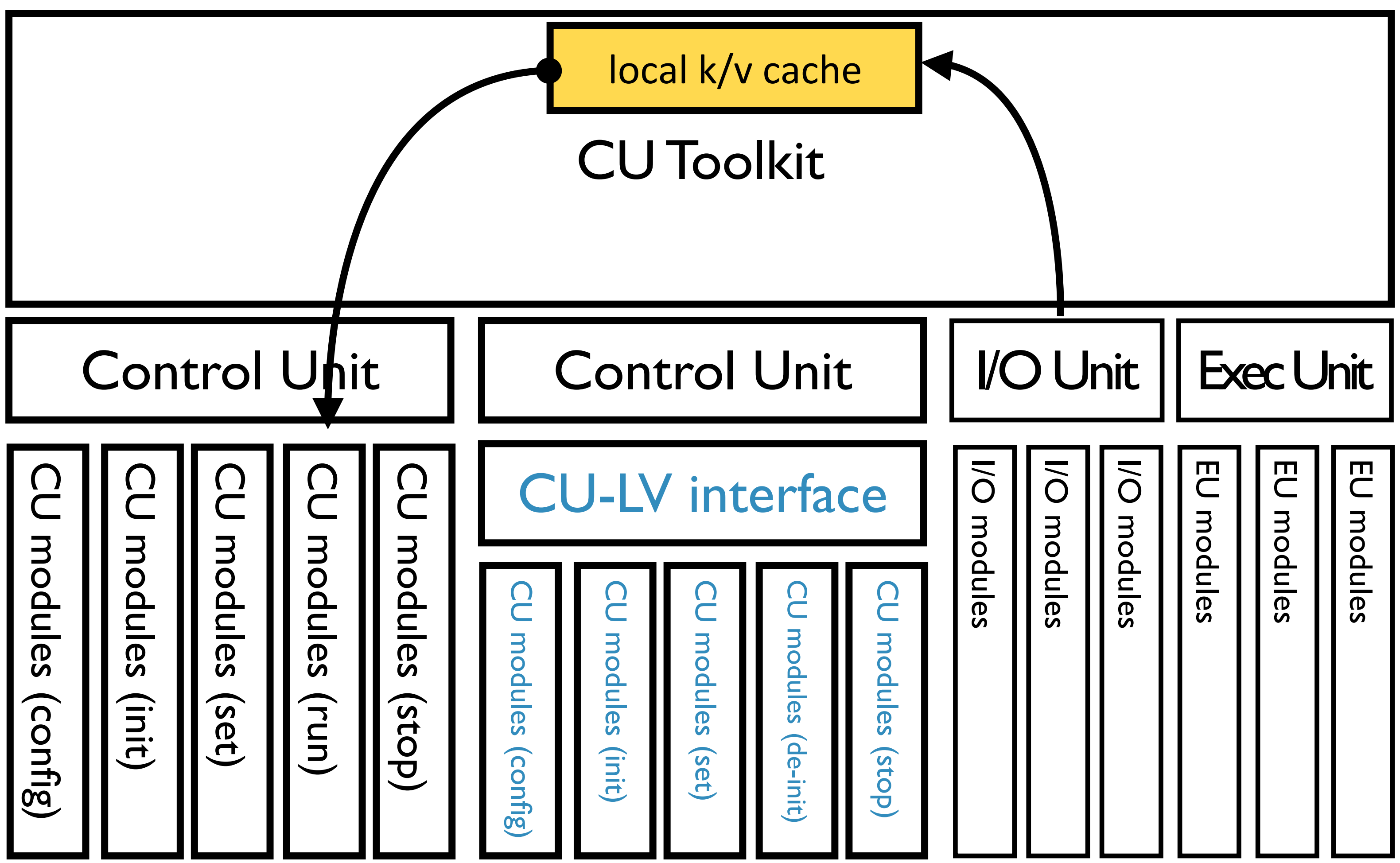



Users Application
!C User Interface
LV User Interface
MatLab UI
User Interface Toolkit
Common Toolkit
!CHAOS
infrastructure
distributed Live Data service
distributed History Data service
commands RPC service
Meta data server
Common Toolkit
CU Toolkit
Control Unit
Control Unit
I/O Unit
Exec Unit
CU modules (config)
CU modules (init)
CU modules (set)
CU modules (de-init)
CU modules (stop)
CU-LV interface
CU modules (config)
CU modules (init)
CU modules (set)
CU modules (de-init)
CU modules (stop)
I/O modules
I/O modules
I/O modules
EU modules
EU modules
EU modules

I/O Unit allows decoupling device logic from signal acquisition

I/O Units, programmed by hw experts, will read Input channels and store data in a local k/v cached (the same technology as for live-data)

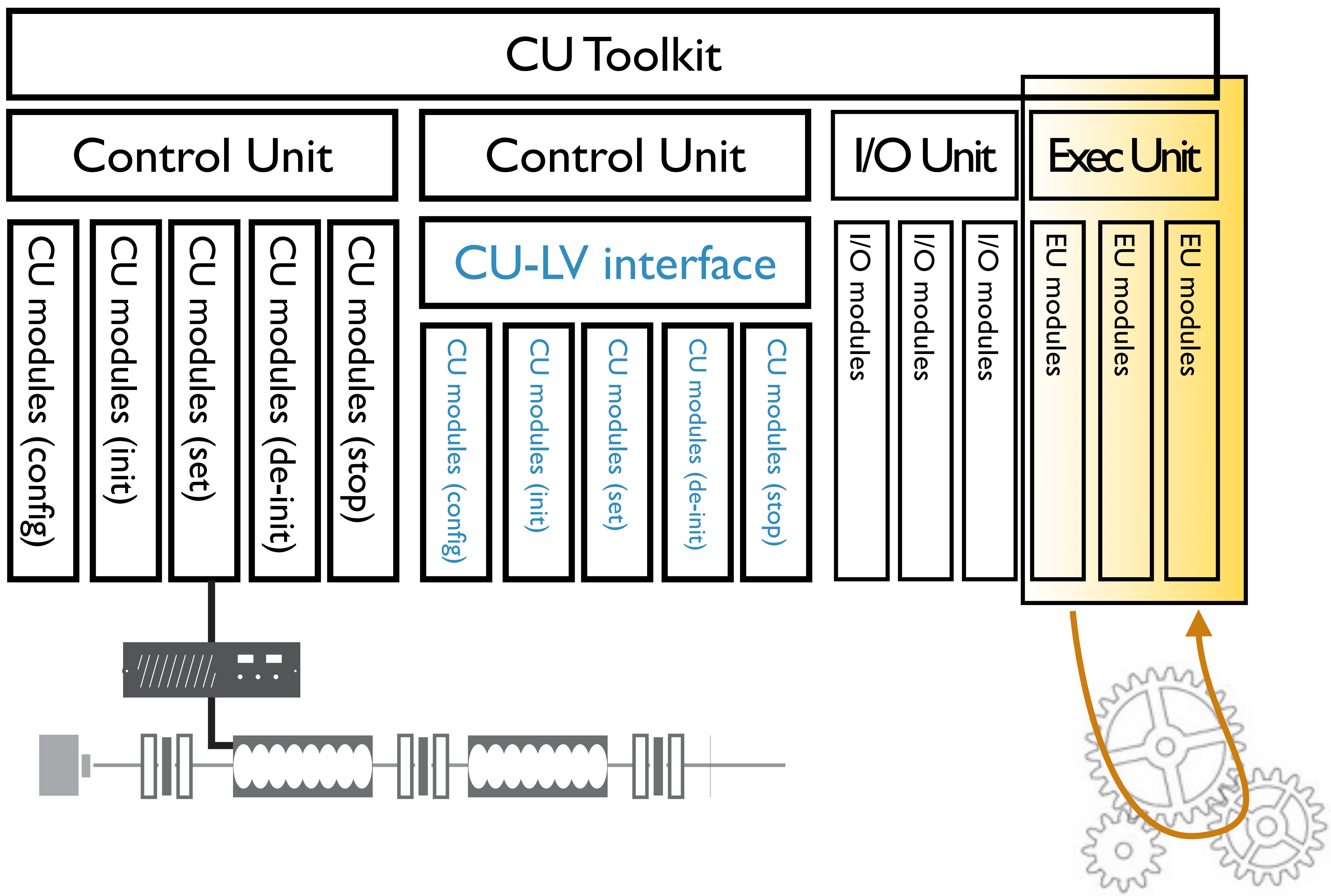
CU programmers will get data from local k/v cache instead of I/O modules directly



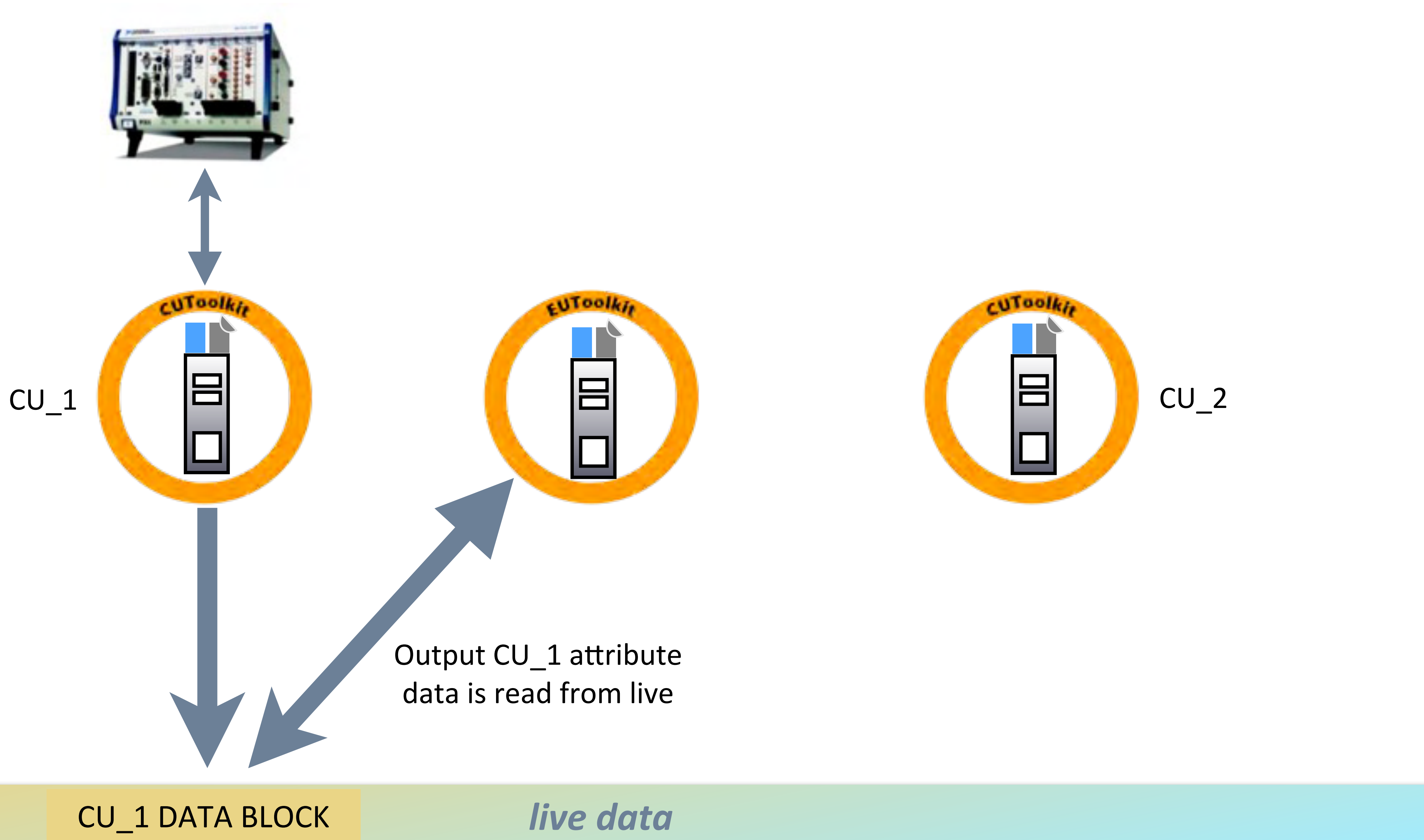
Execution Unit extend the concept of Control Unit to computational tasks

EU are not connected to any device, instead they produce data as result of a logic or calculations. They produce an output data class as consequence of the value defined by the input data class.

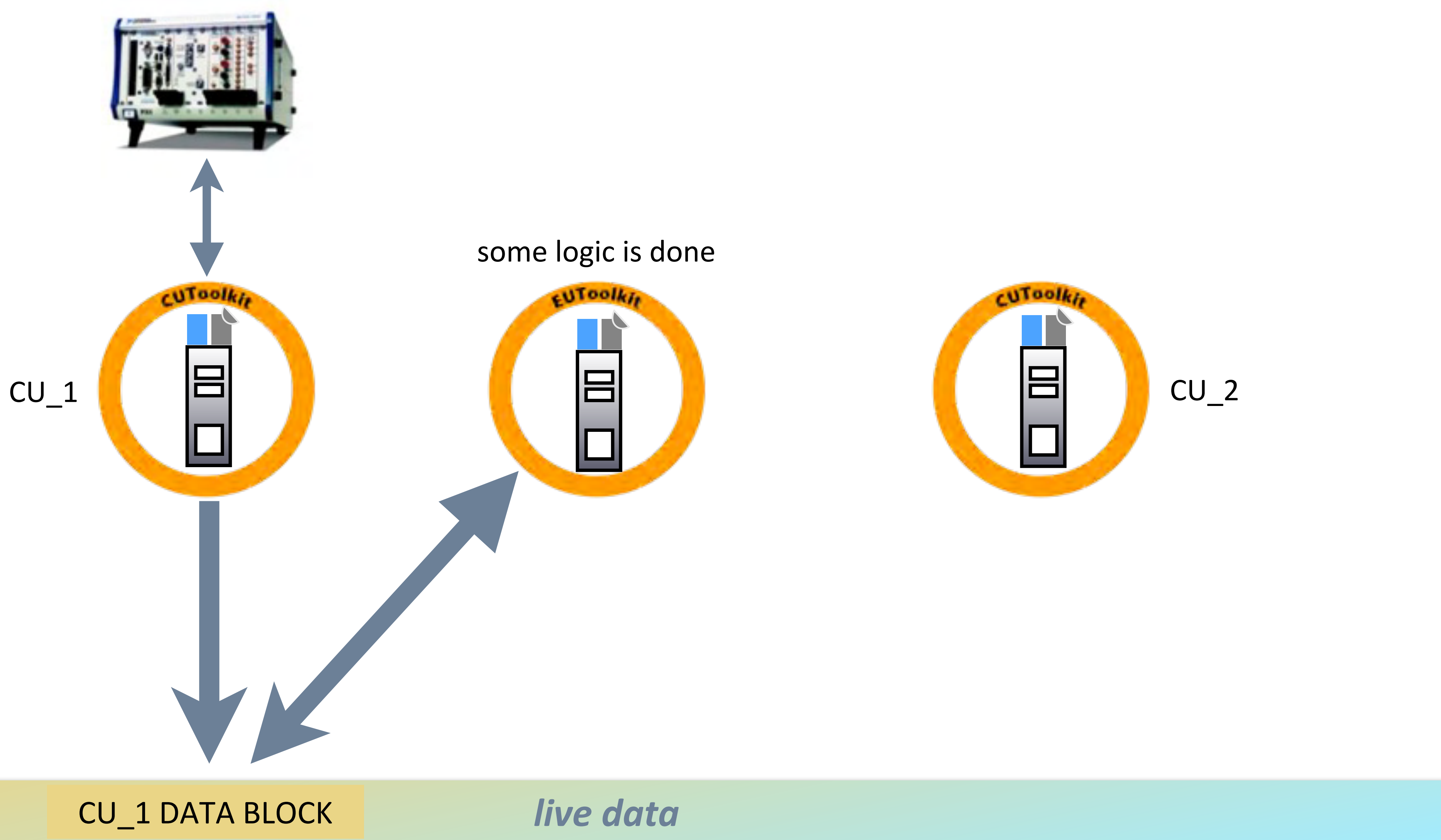
EU wil be used to implement analysis, measurement procedures, feedbacks



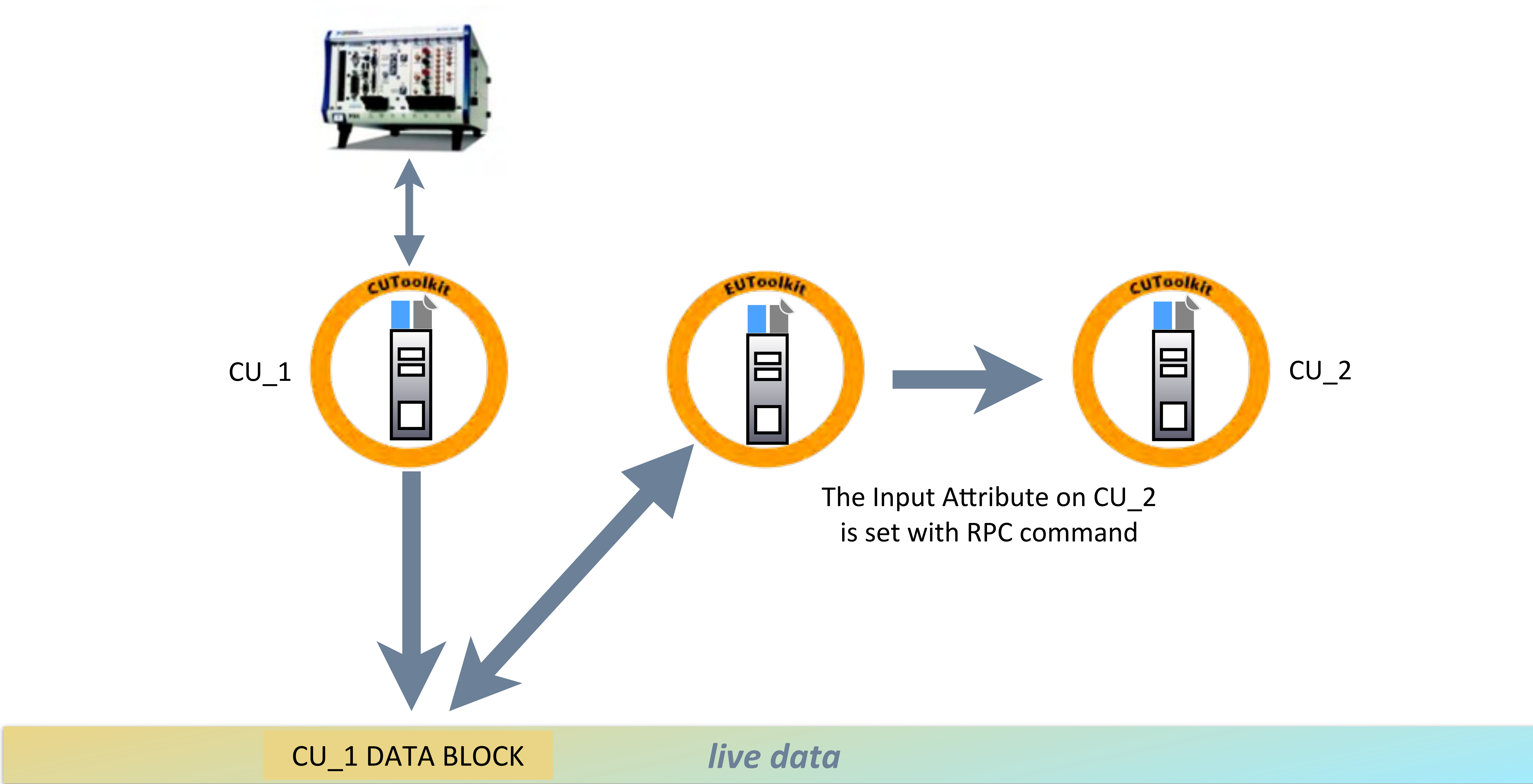
Execution Unit Example 1



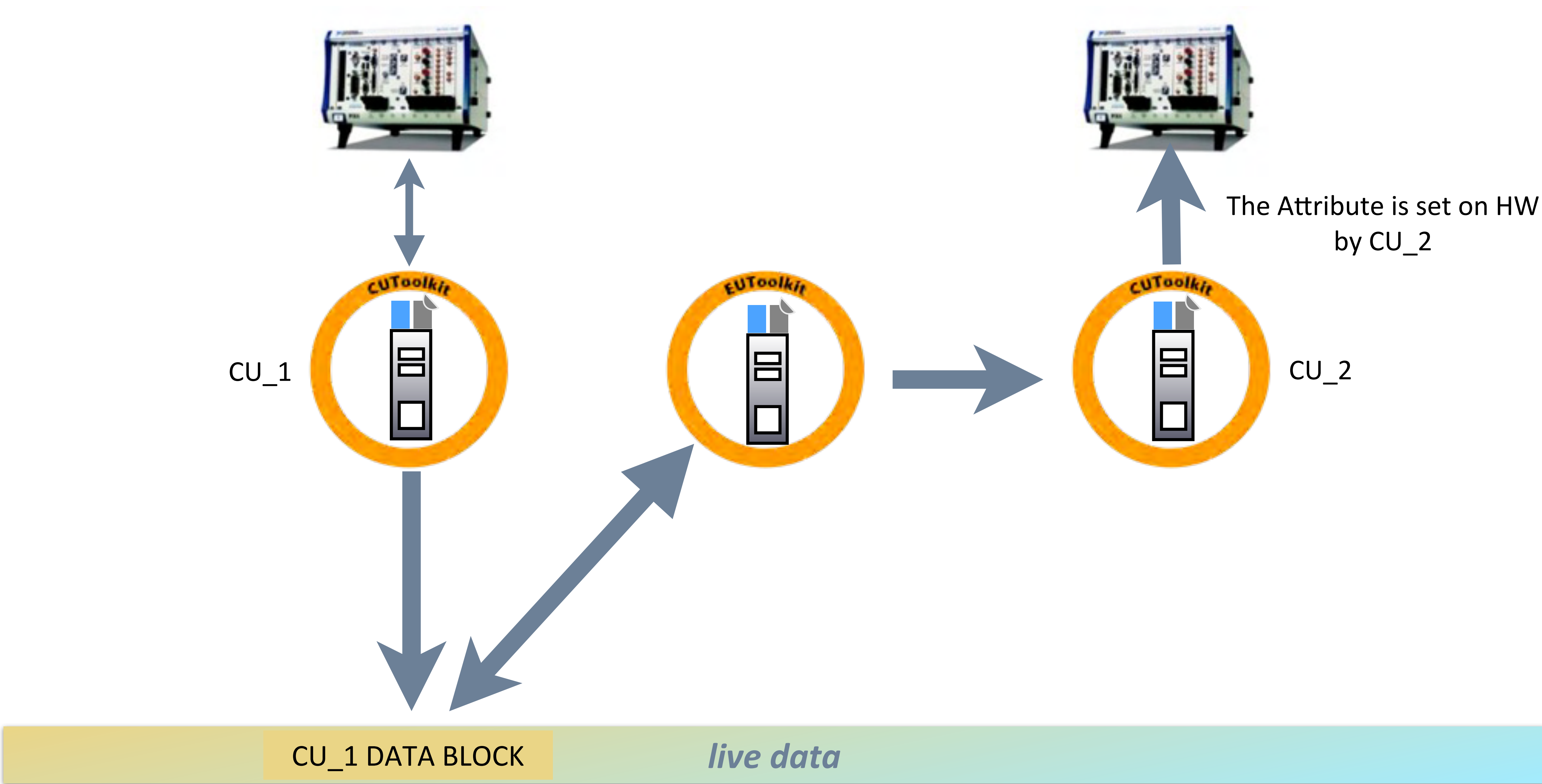
Execution Unit Example 1



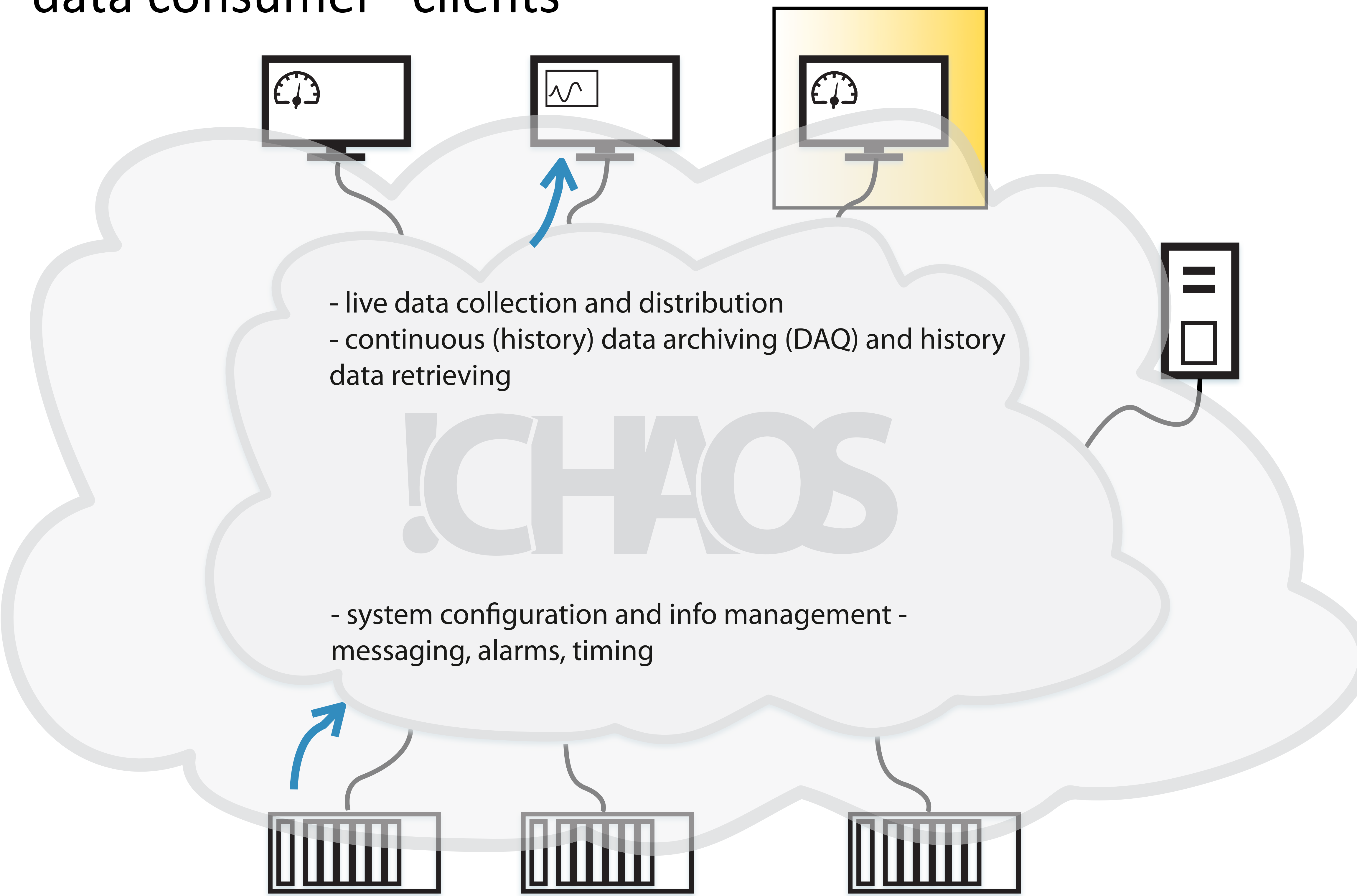
Execution Unit Example 1



Execution Unit Example 1

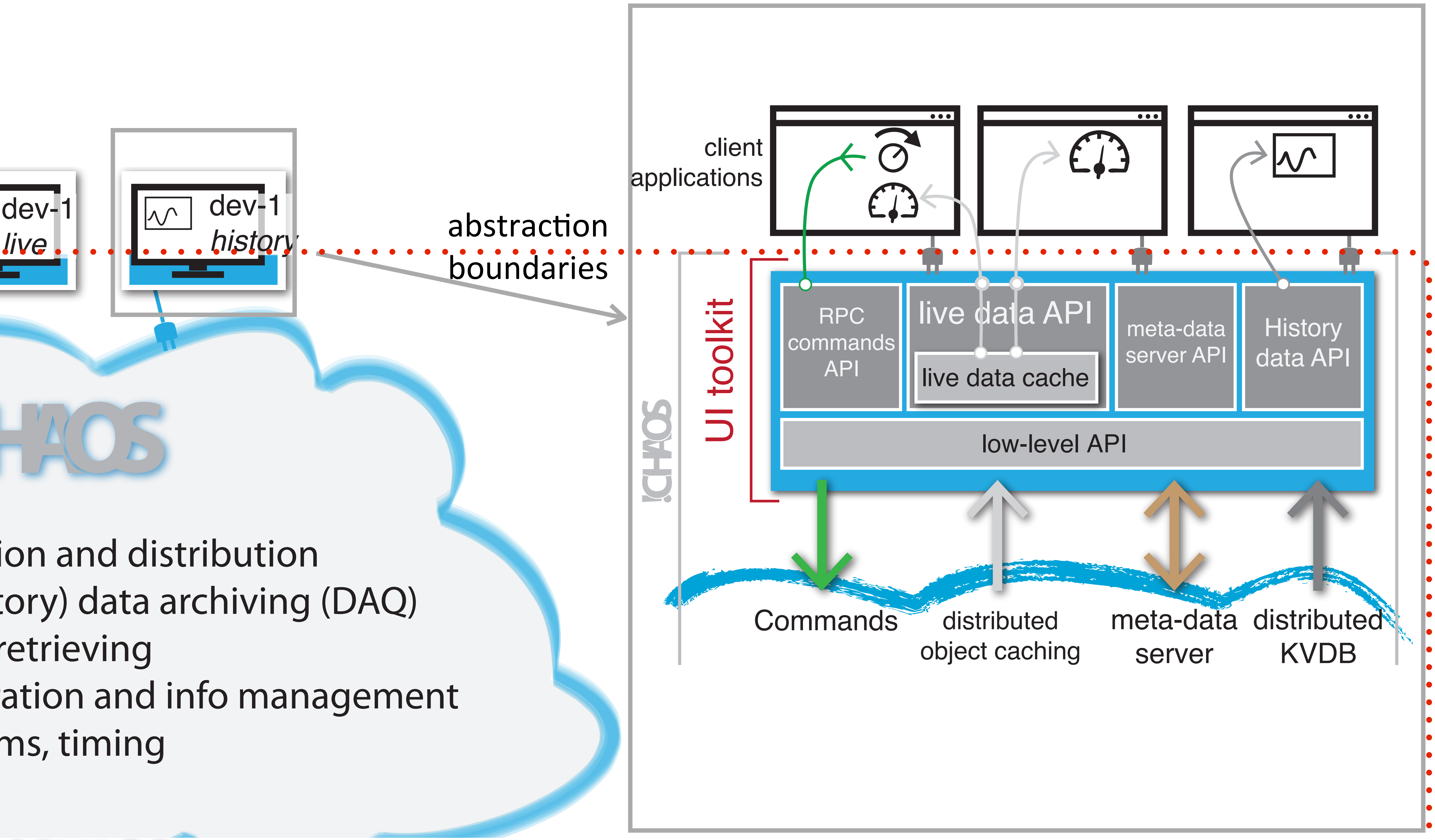


“data consumer” clients

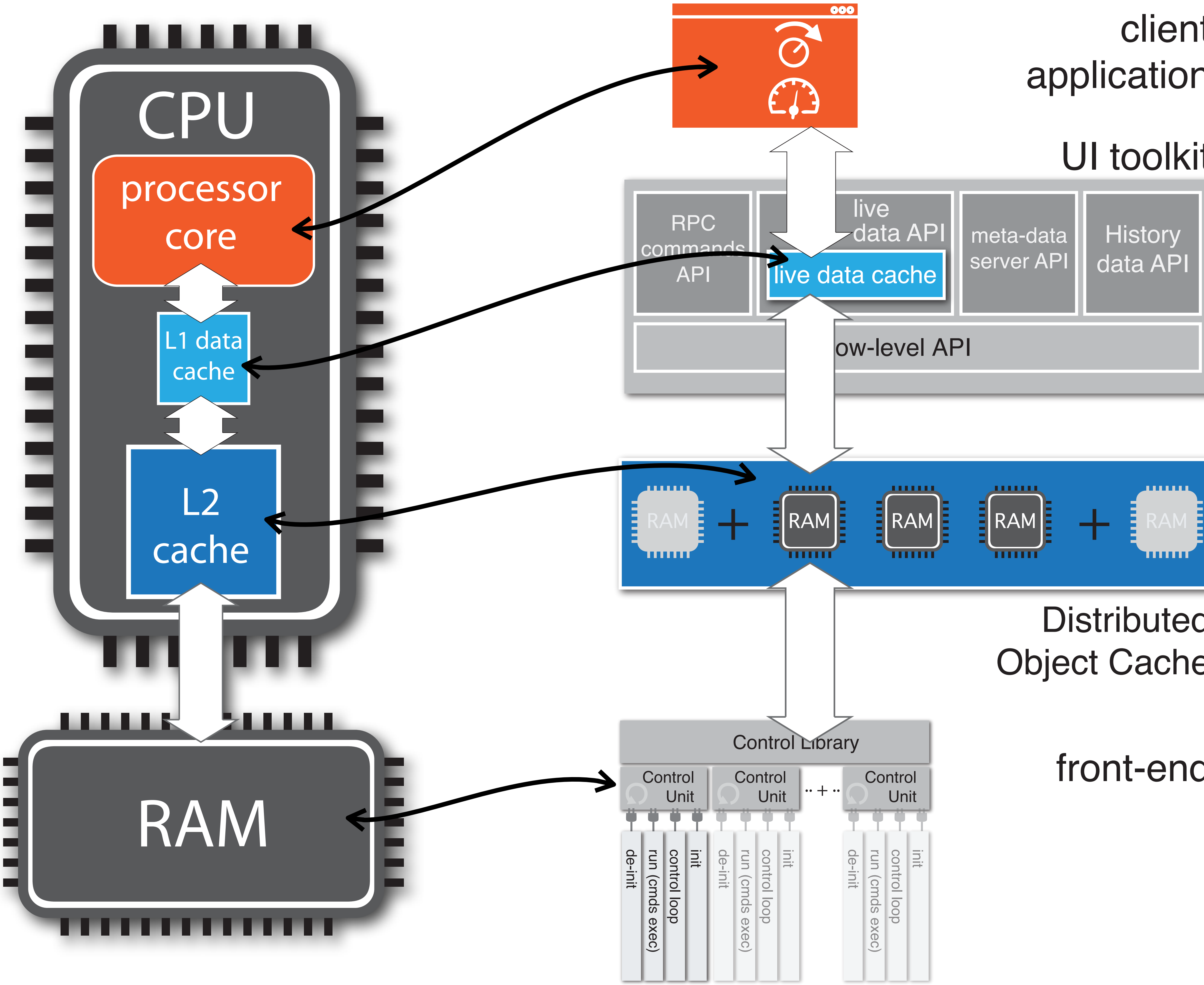


“data producer” clients

client abstraction

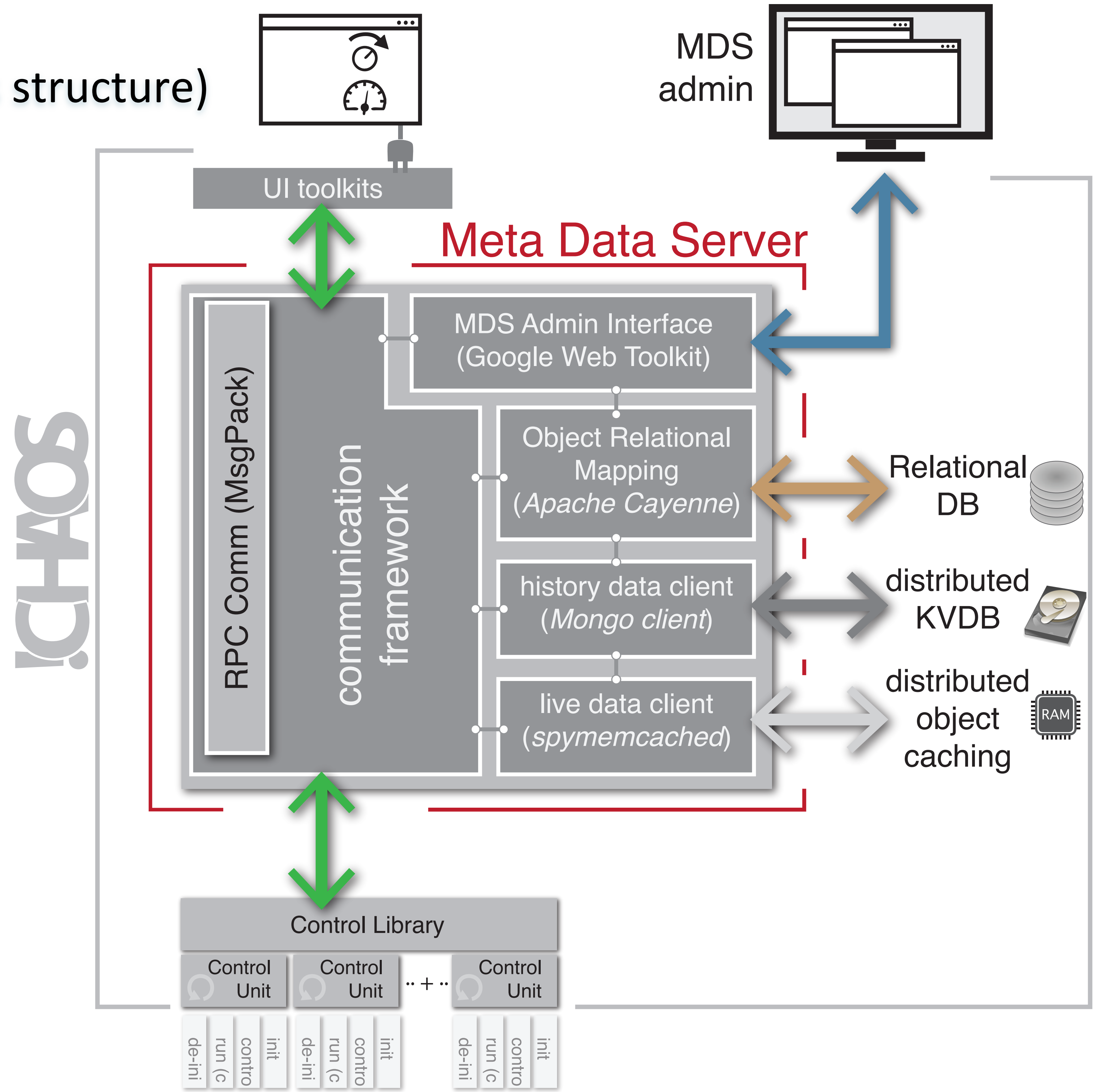


are we simply caching everywhere ?

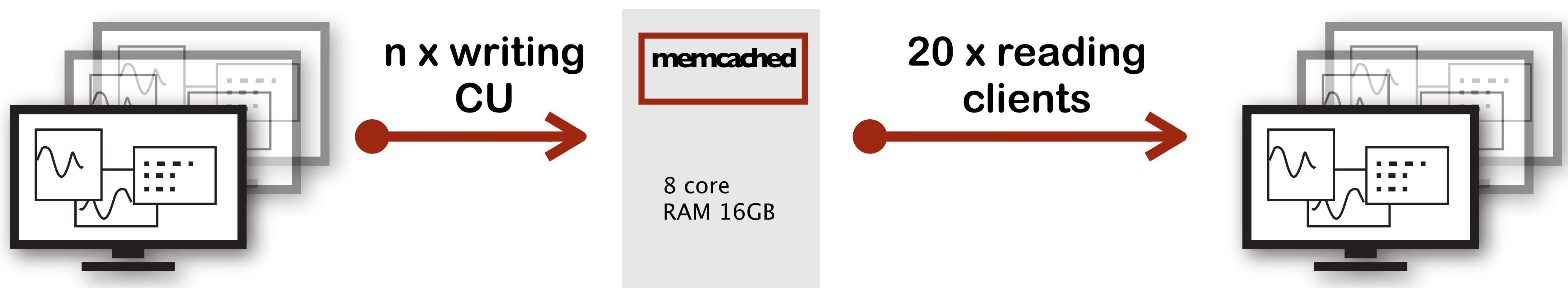


Meta-data Server

- 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

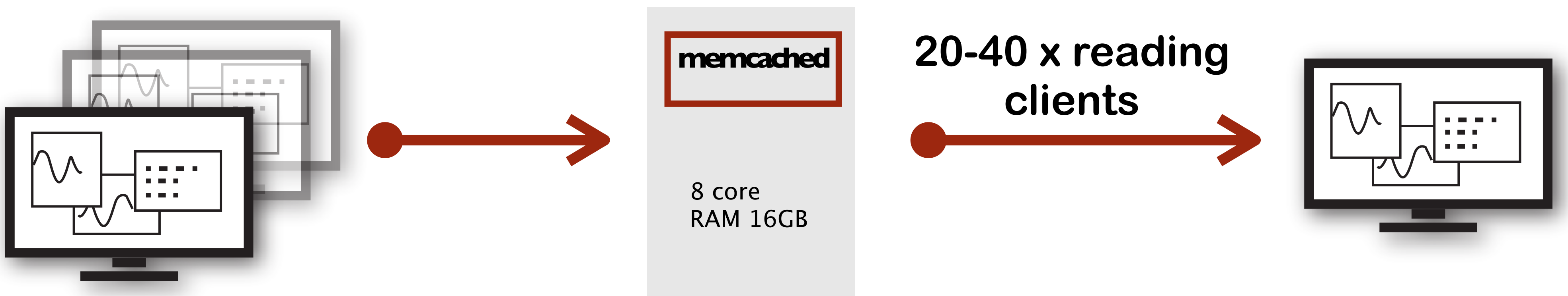


test #3.1



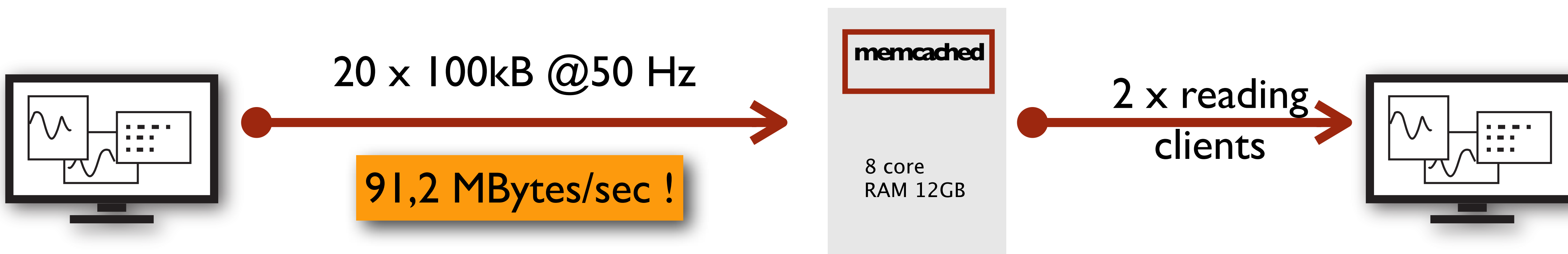
writing every... (msec)	#CU (Write)	#clients (Read)	#servers	#processes/ server	CPU load (%)
20	60	20	1	1	3-5
20	80	20	1	1	4-6
20	80	20	2	1	2-3
50	60	20	1	1	1-3
50	80	20	2	1	0-2
100	60	20	1	1	?
100	80	20	2	1	?

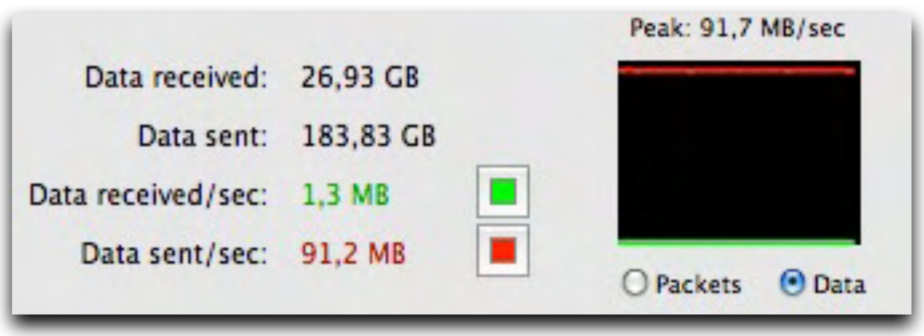
test #3.2

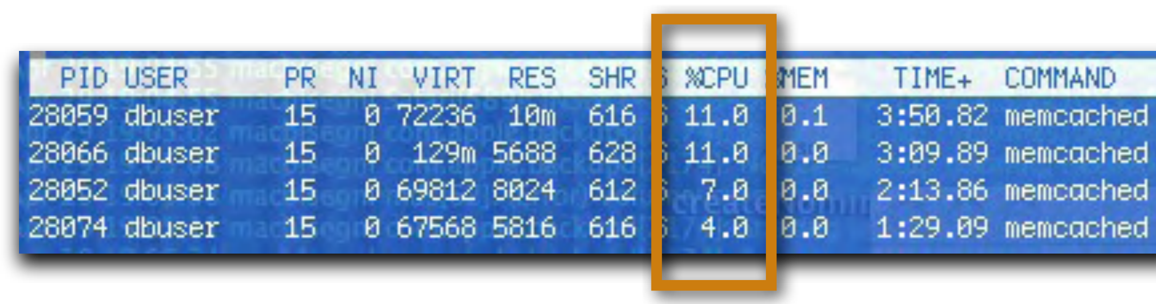


writing every... (msec)	#CU (Write)	#clients (Read)	#servers	#processes/ server	CPU load (%)
20	80	20	1	4 (1 per core)	2-3
20	80	40	1	4 (1 per core)	2-3
		40	1	4 (1 per core)	0

test #4

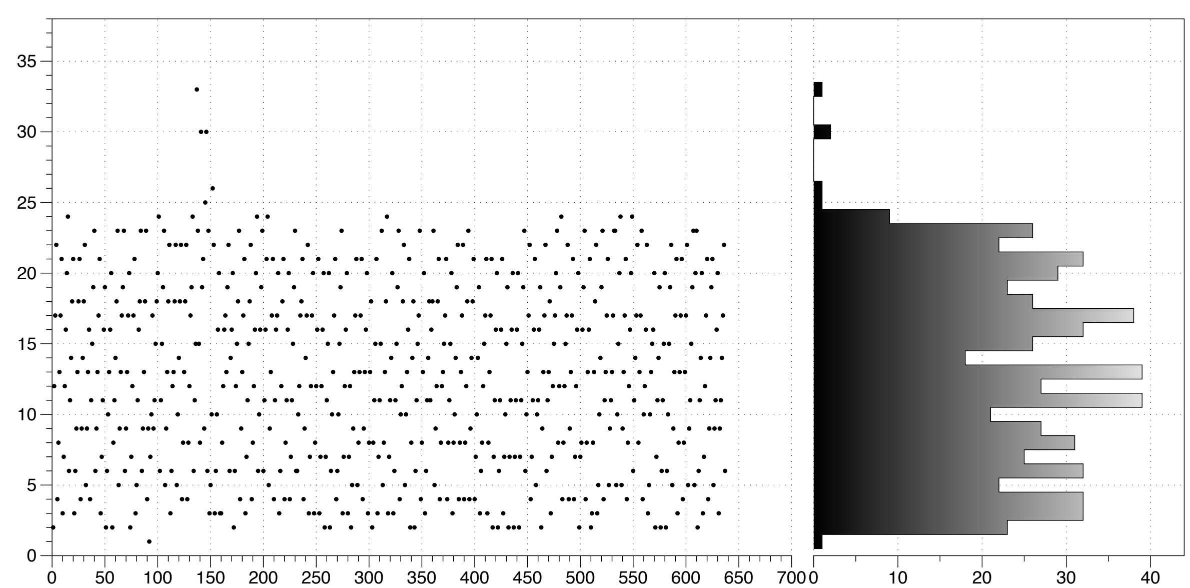




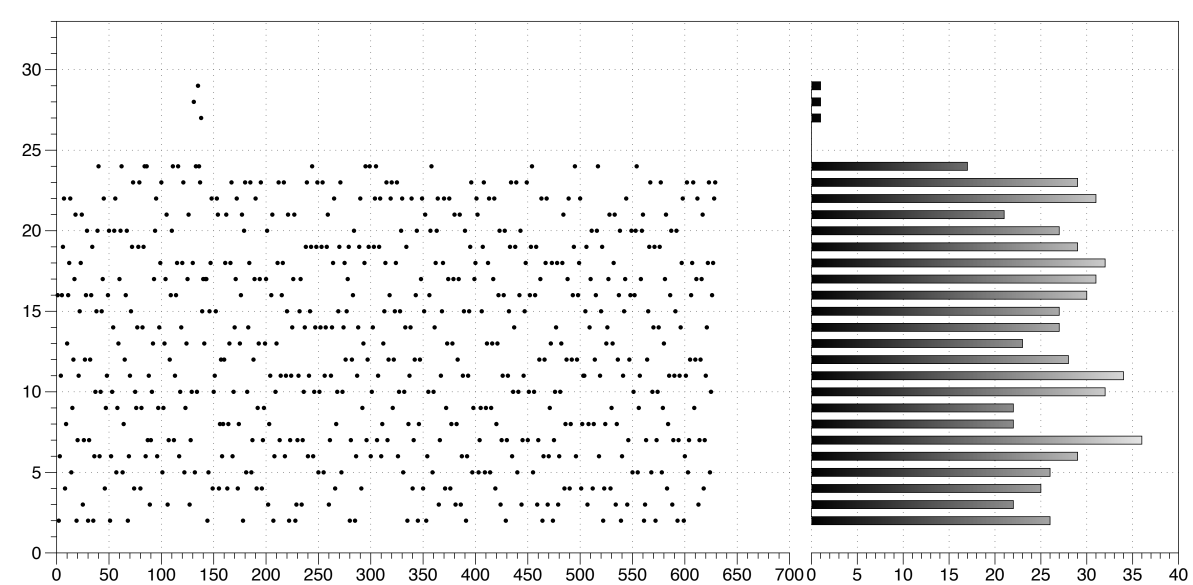


PID	USER	PR	NI	VIRT	RES	SHR	%CPU	%MEM	TIME+	COMMAND
28059	dbuser	15	0	72236	10m	616	11.0	0.1	3:50.82	memcached
28066	dbuser	15	0	129m	5688	628	11.0	0.0	3:09.89	memcached
28052	dbuser	15	0	69812	8024	612	7.0	0.0	2:13.86	memcached
28074	dbuser	15	0	67568	5816	616	4.0	0.0	1:29.09	memcached

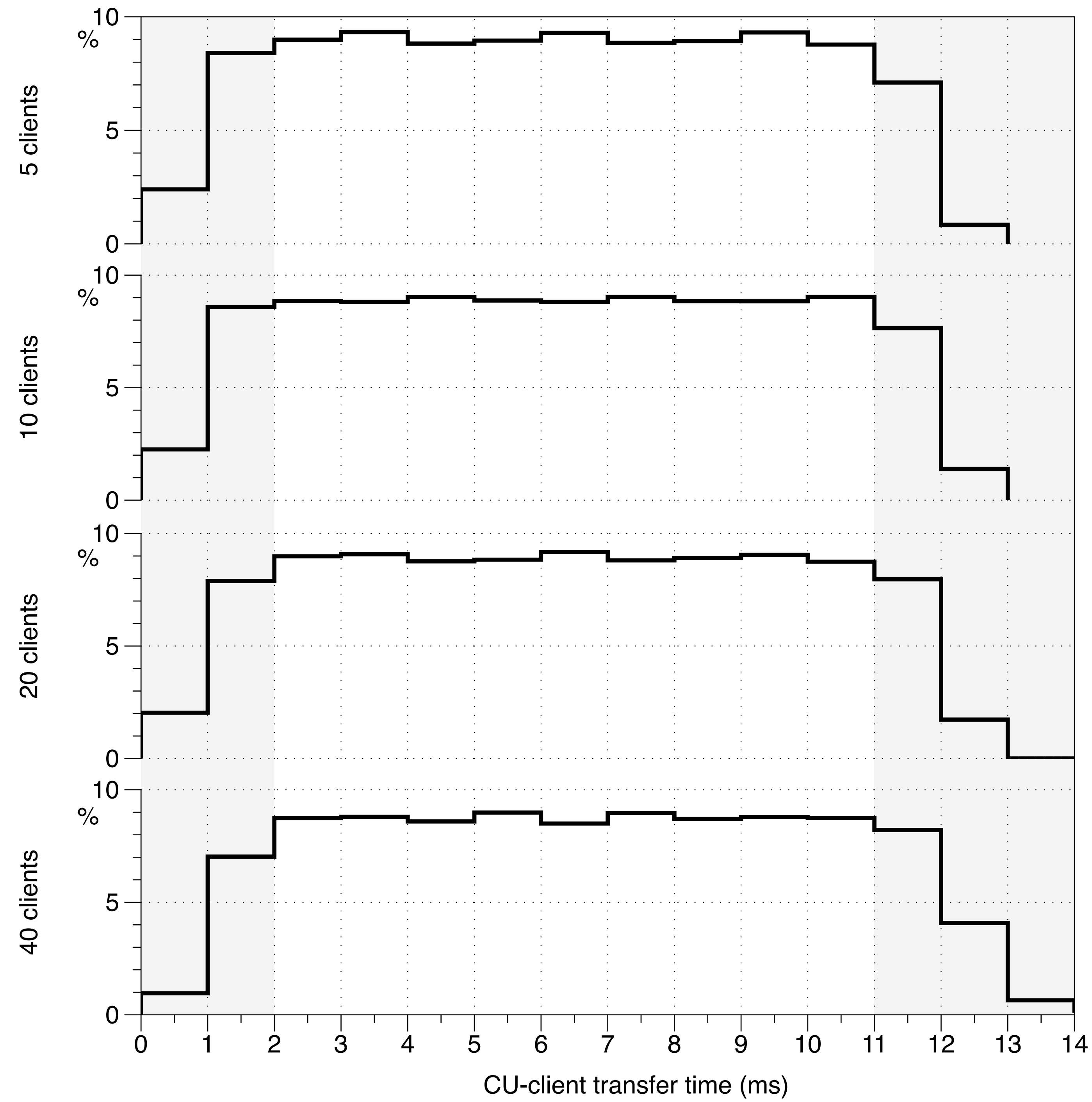
s4_hardware1_w20_m20_buff100000_rd10.log



s4_hardware1_w20_m20_buff100000_rd12.log

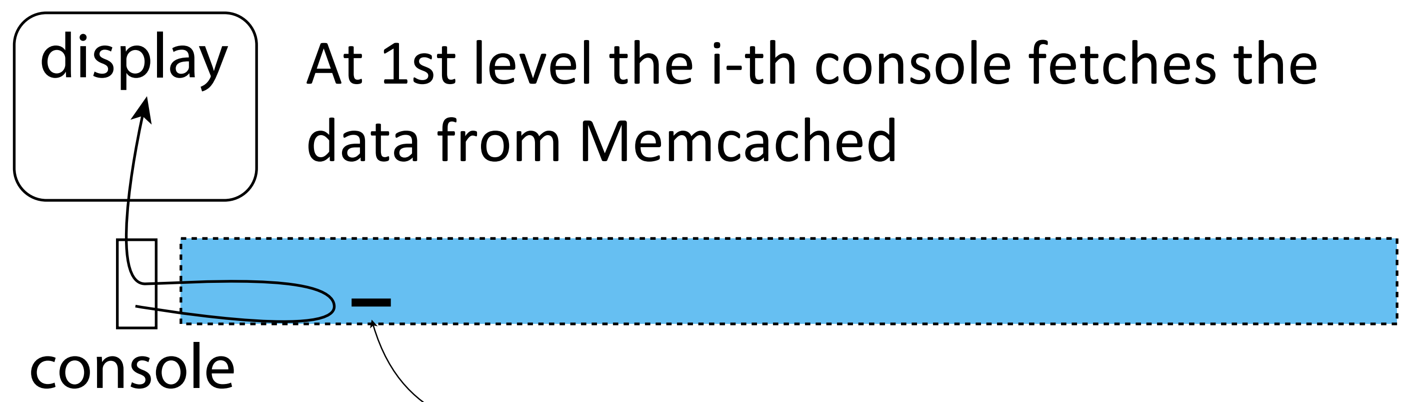


Measured transfer time between front-end CU and a client application via DOC for a different number of concurrent clients reading the same key/value being continuously updated by the CU.



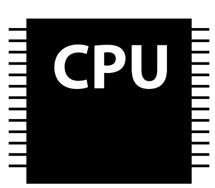
tests at DAFNE

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

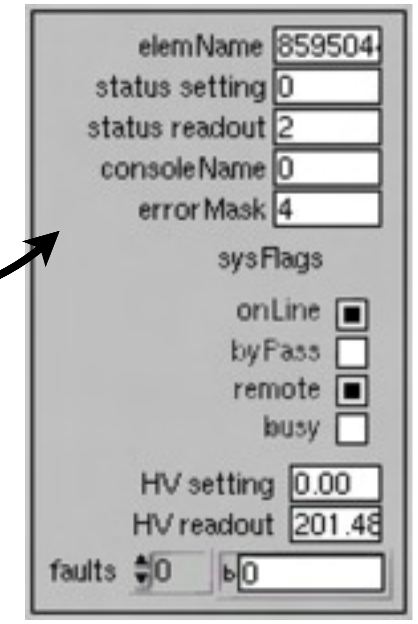
at 2nd level Memcached is employed instead of the VME address space



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**

DEVIL

Preliminary measurements



data size: 64 bytes for packet read

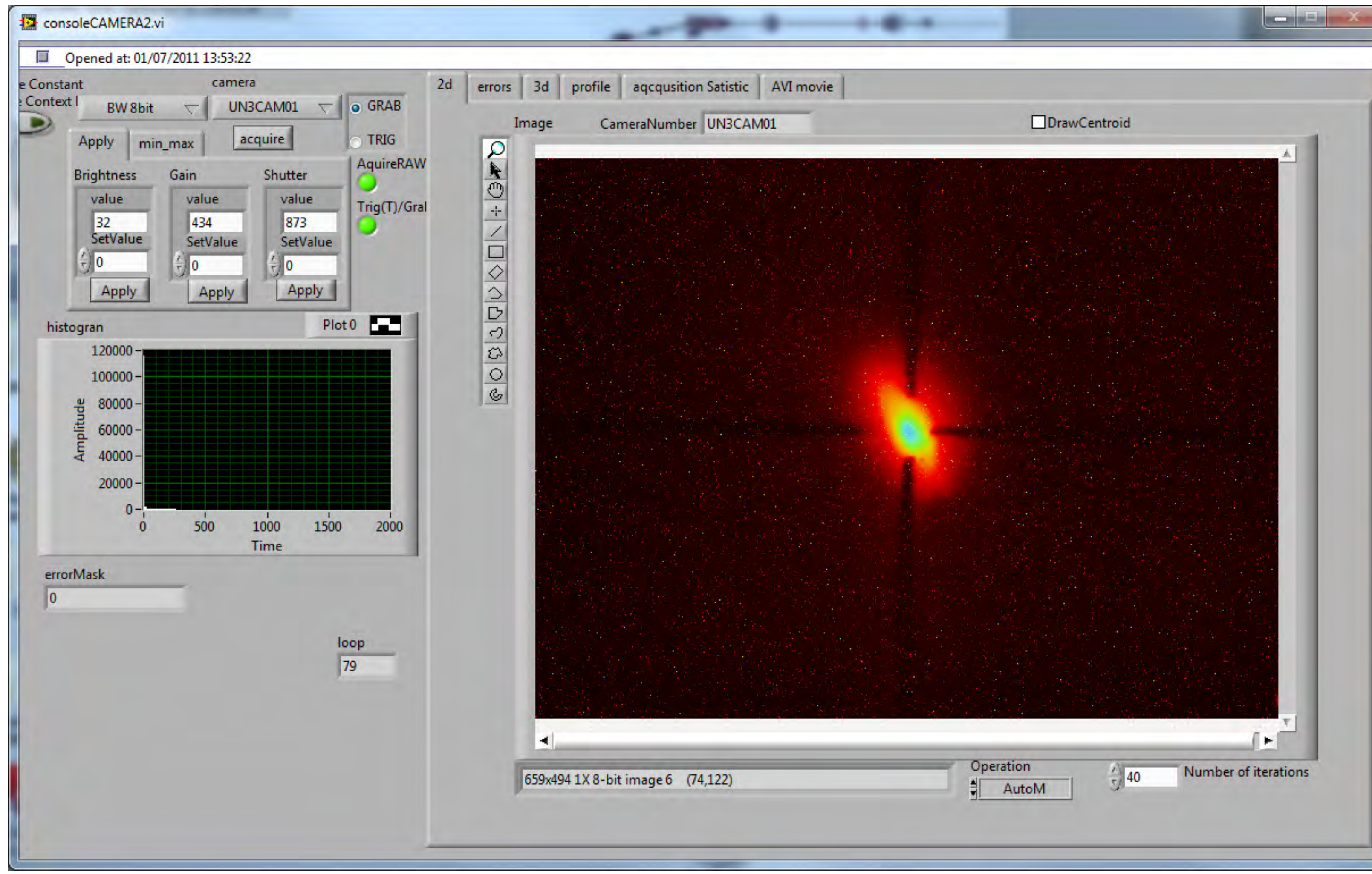
fetch frequency ~ 100 Hz
with no dependency on the number of fetching consoles
(up to 7 in our test)

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

tests at SPARC

Memcached has been used for storing the beam image from a digital camera of beam diagnostic.

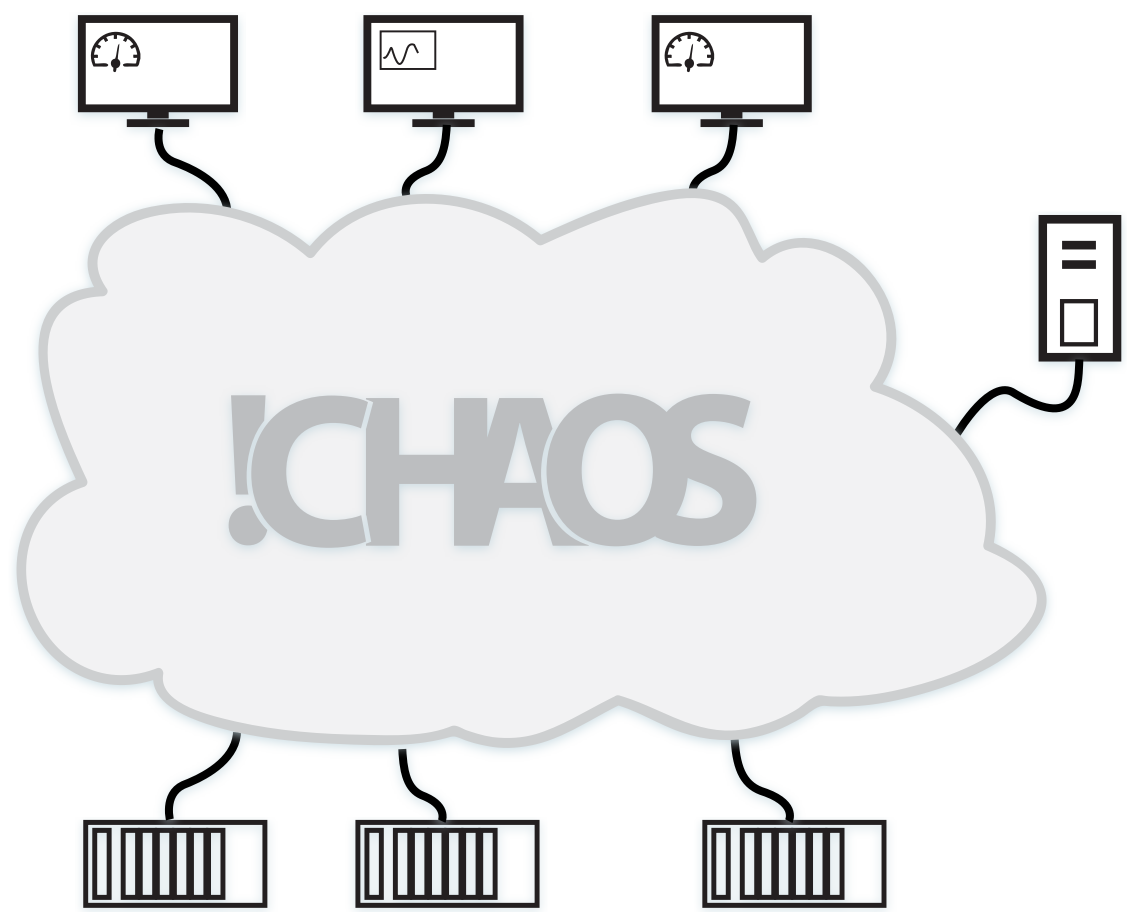
- network: Ethernet @1 Gbps
- image size: 640x480
  - @8 bit = **300 kB**
  - @16 bit = **600 kB**



measured fetch frequency: **8bit ~ 25 Hz**, **16bit ~ 13 Hz**; same transfer rate for all consoles fetching images from memcached (up to 4 in our test)

!CHAOS

- aims at the development of a control system for INFN future accelerators and large apparatuses
- introduces a breaking through paradigm based on high performance internet technologies
- profits from INFN long tradition of control systems for particle accelerators, expertise in computing and information technologies
- can be a complement for future INFN interdisciplinary projects
- generates opportunities for collaborations with industries and technology transfer



http://chaos.infn.it

- !CHAOS launch workshop at LNF (Dec. 2011)
- evaluation release (`!CHAOS_beta_0.1`) ready to download from website
- !CHAOS established as **`Open Source project`** of INFN
- candidate Control System for SuperB accelerator and for slow-controls of experimental apparatus
- collaboration with National Instruments for integration of LabVIEW in !CHAOS
  - NI-!CHAOS meeting at LNF (March 2012)
  - invited at Big Physics Symposium (Zurich - March 2012)
- collaborations with italian industries (agreements for joint developments in preparation)
- academic collaborations: Univ. Roma TV (Fac. Informatica e Ingegneria), Politecnico di Bari, Università di Cagliari (Fac. Ingegneria)