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

software architecture and developer introduction

C. Bisegni

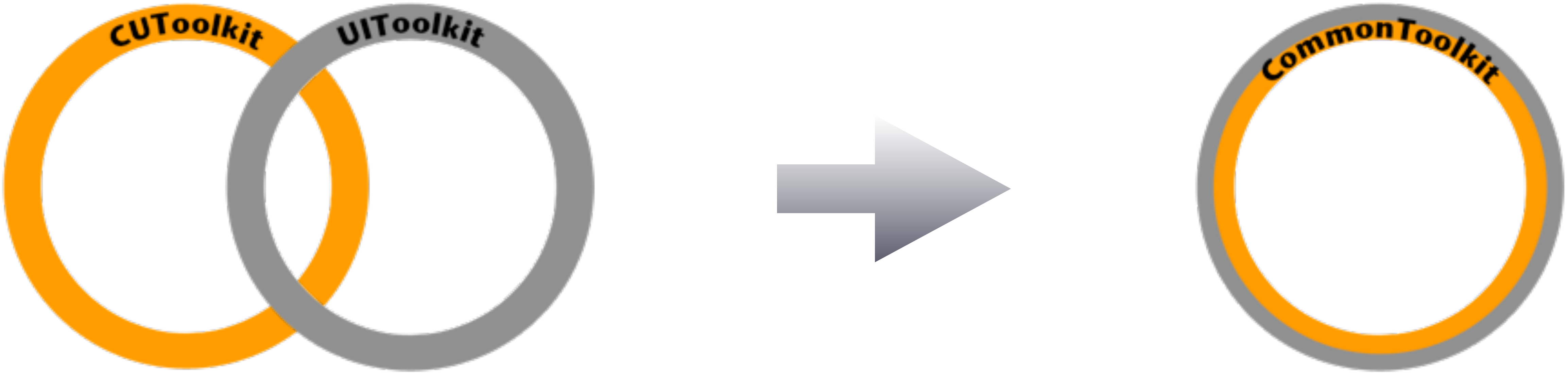
!CHAOS software layer

!CHAOS software layer

CUToolkit, abstract the !CHAOS resources to the device drivers developers.

UIToolkit tools for developing client application that accesses !CHAOS resource

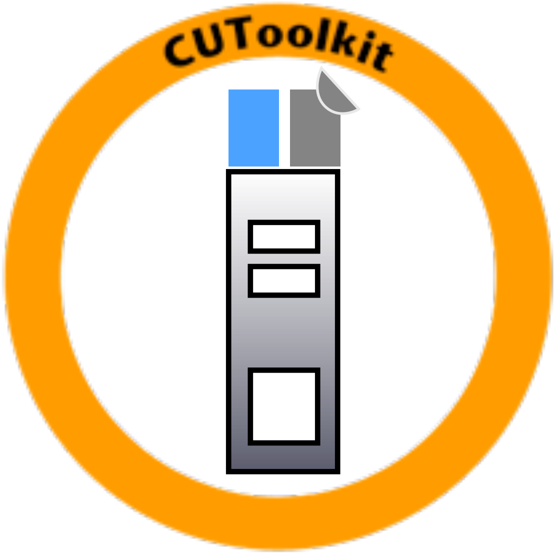
!CHAOS software layer



- The two layer are based on **CommonToolkit** and all they are the CHAOS Framework
- Developed in c++
- Multi Threading

!CHAOS Node & Service

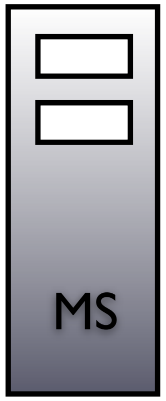
!CHAOS FrontEnd and User Node



Control Unit, a piece of software developed on CUToolkit implementing the device drivers

The Chaos Control GUI is based on the UIToolkit for accessing !CHAOS resources. The UIToolkit is also used by control panels/client applications developers to make their custom application

!CHAOS Middle Layer



MetaData Server, keep track of all information about device DataSet and Command, CU address and other info.

MEMCACHED

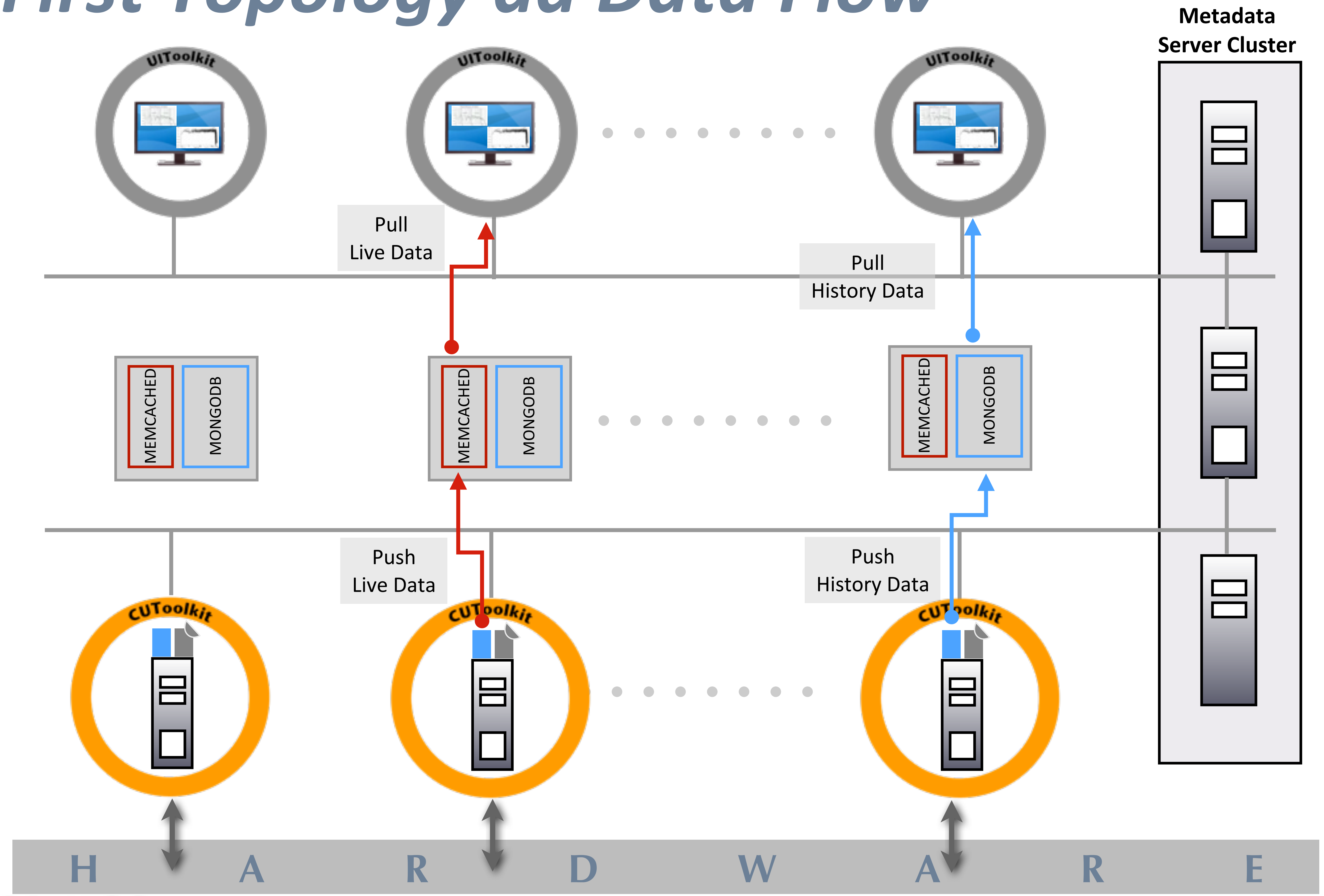
Memcached is used for caching lived data

MONGODB

MongoDB is used for storing history data

Topology and Data flow

First Topology ad Data Flow



Topology Next Step

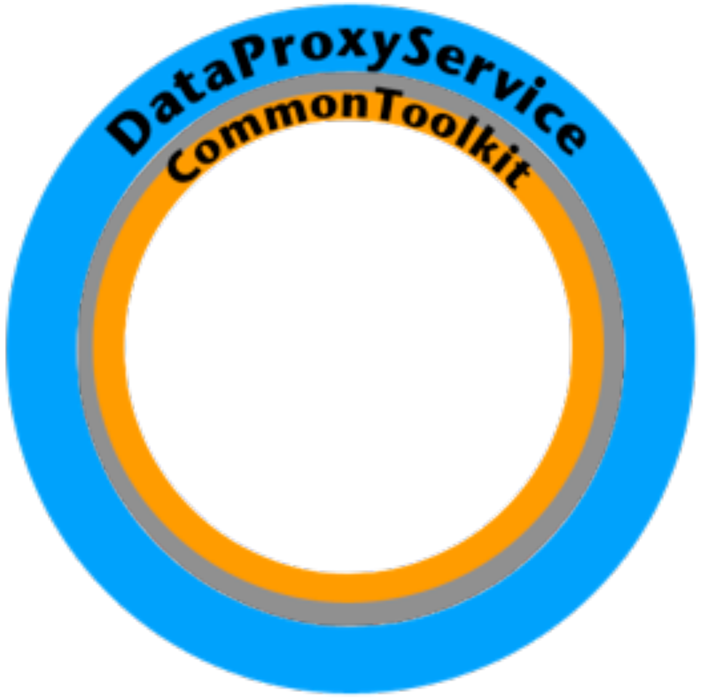
We are in R&D so we have made some adjustment to the CHAOS topology

history data must be managed in different way for different kind of query:

- near time history data
- long time history data
- data warehouse query
- etc.

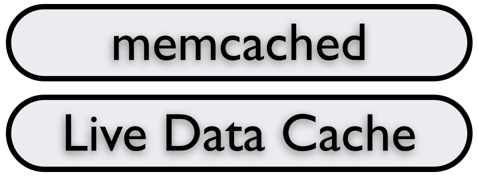
New Node & Service

!CHAOS Client & MS Node



Data Proxy Service, is a scalable service that implement a common proxy for the Live and History data services. It includes memcached and the drivers for implementing ChaosQL for storing or querying history data

!CHAOS Service


memcached

Live Data Cache

Live data cache is a service implemented with Memcached

This is the History Storage Cloud, that can be accessed by means of the Data Service Proxy.

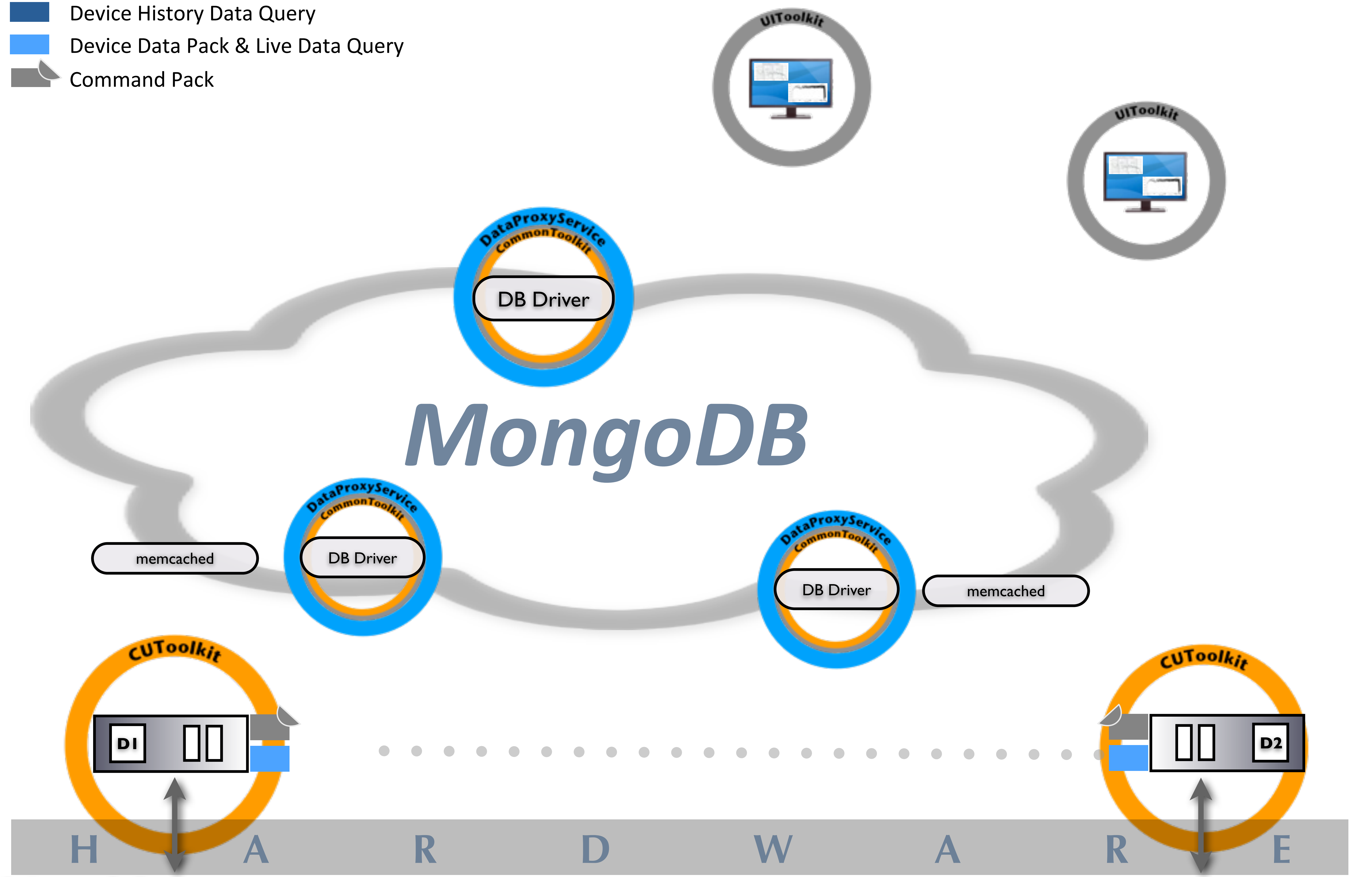
New Topology and Data flow

Topology and Data flow

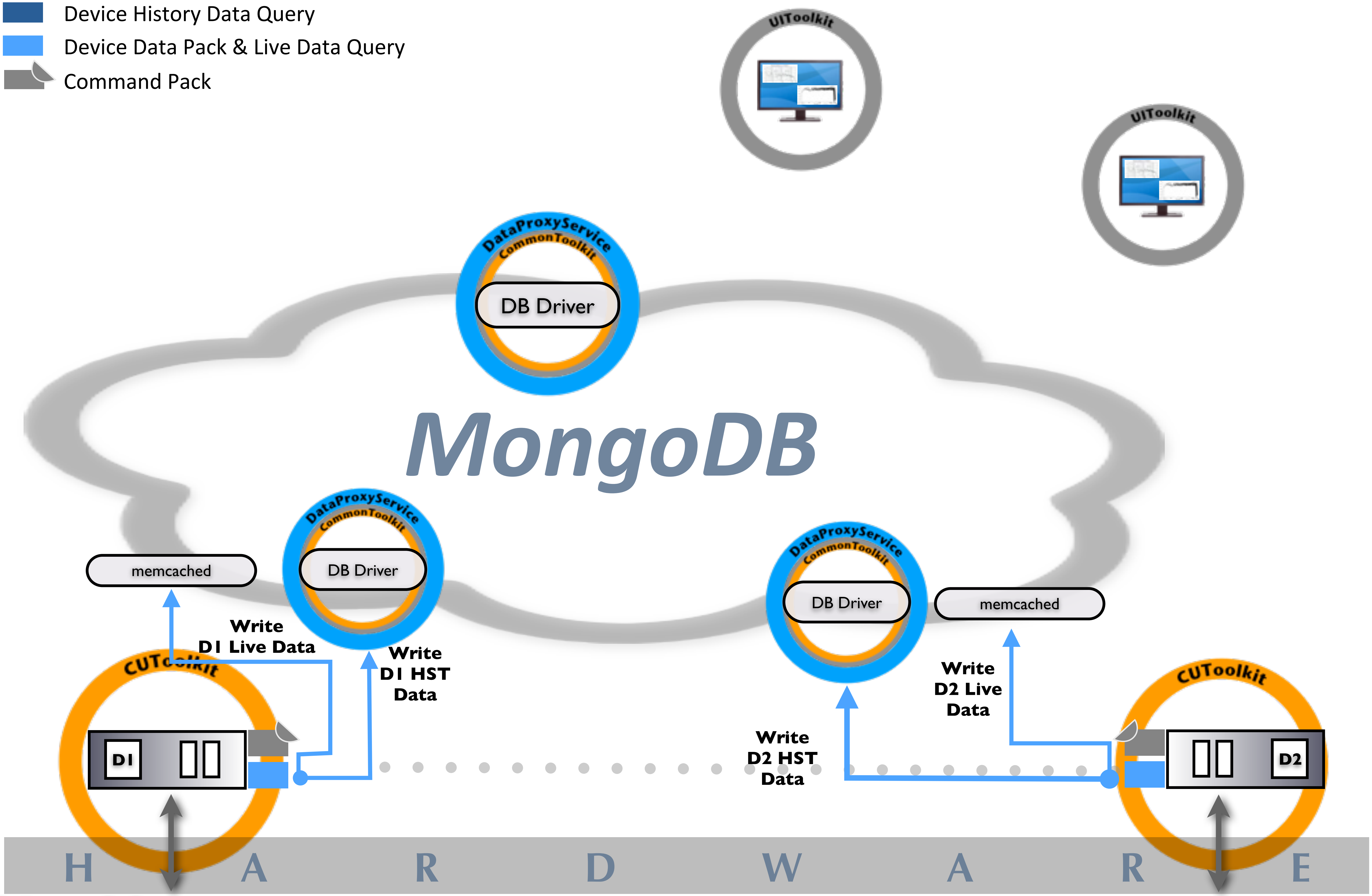
🟦 Device History Data Query

🟦 Device Data Pack & Live Data Query

▬ Command Pack



Topology and Data flow



Device History Data Query

Device Data Pack & Live Data Query

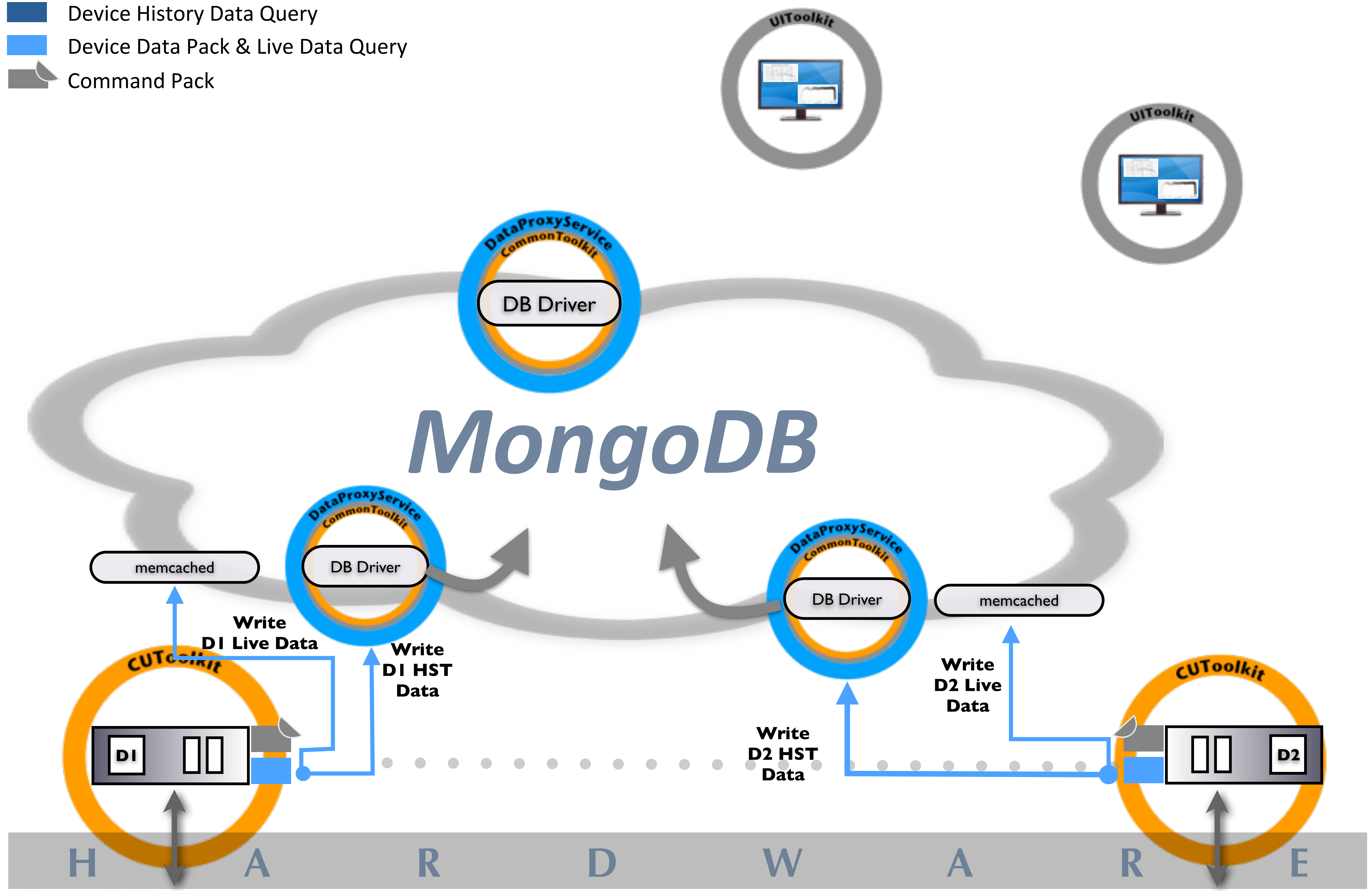
Command Pack

Topology and Data flow

🟦 Device History Data Query

🟦 Device Data Pack & Live Data Query

▬ Command Pack



UIToolkit

UIToolkit

DataProxyService
CommonToolkit
DB Driver

MongoDB

memcached

DataProxyService
CommonToolkit
DB Driver

DataProxyService
CommonToolkit
DB Driver

memcached

Write
D1 Live Data

Write
D1 HST
Data

Write
D2 HST
Data

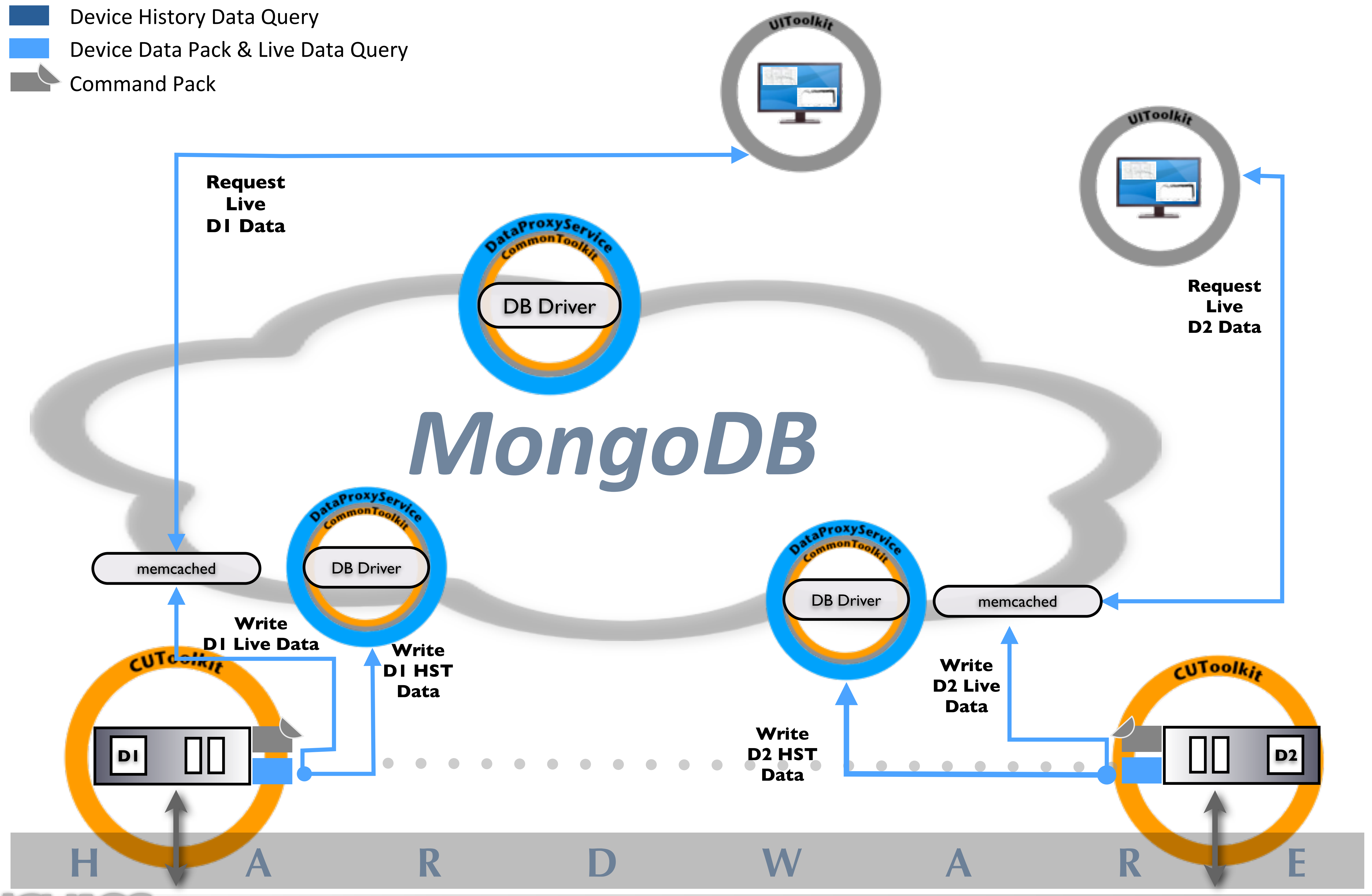
Write
D2 Live
Data

CUToolkit
D1

CUToolkit
D2

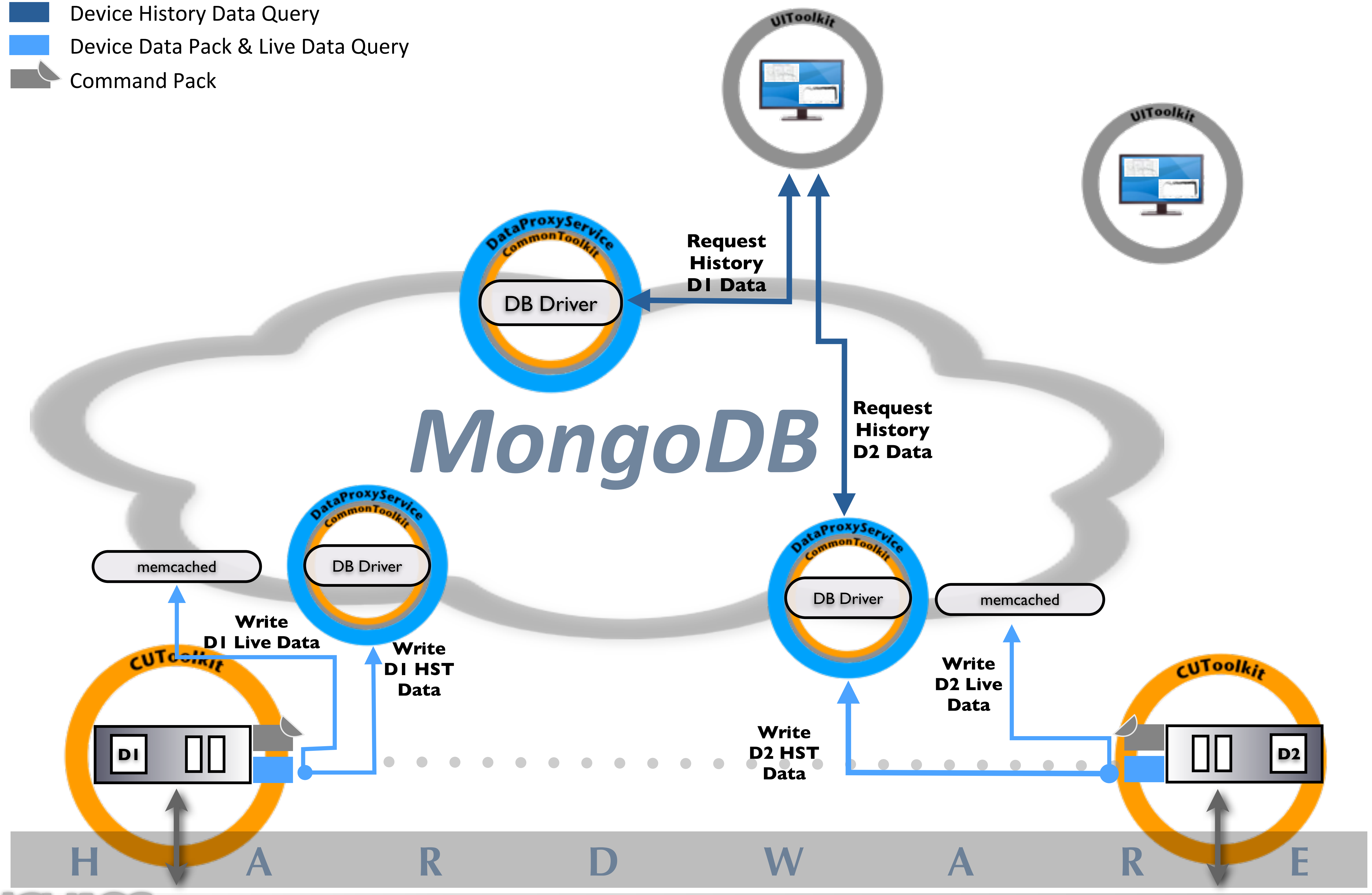
H A R D W A R E

Topology and Data flow



Topology and Data flow

- ▬ Device History Data Query
- ▬ Device Data Pack & Live Data Query
- ▬ Command Pack

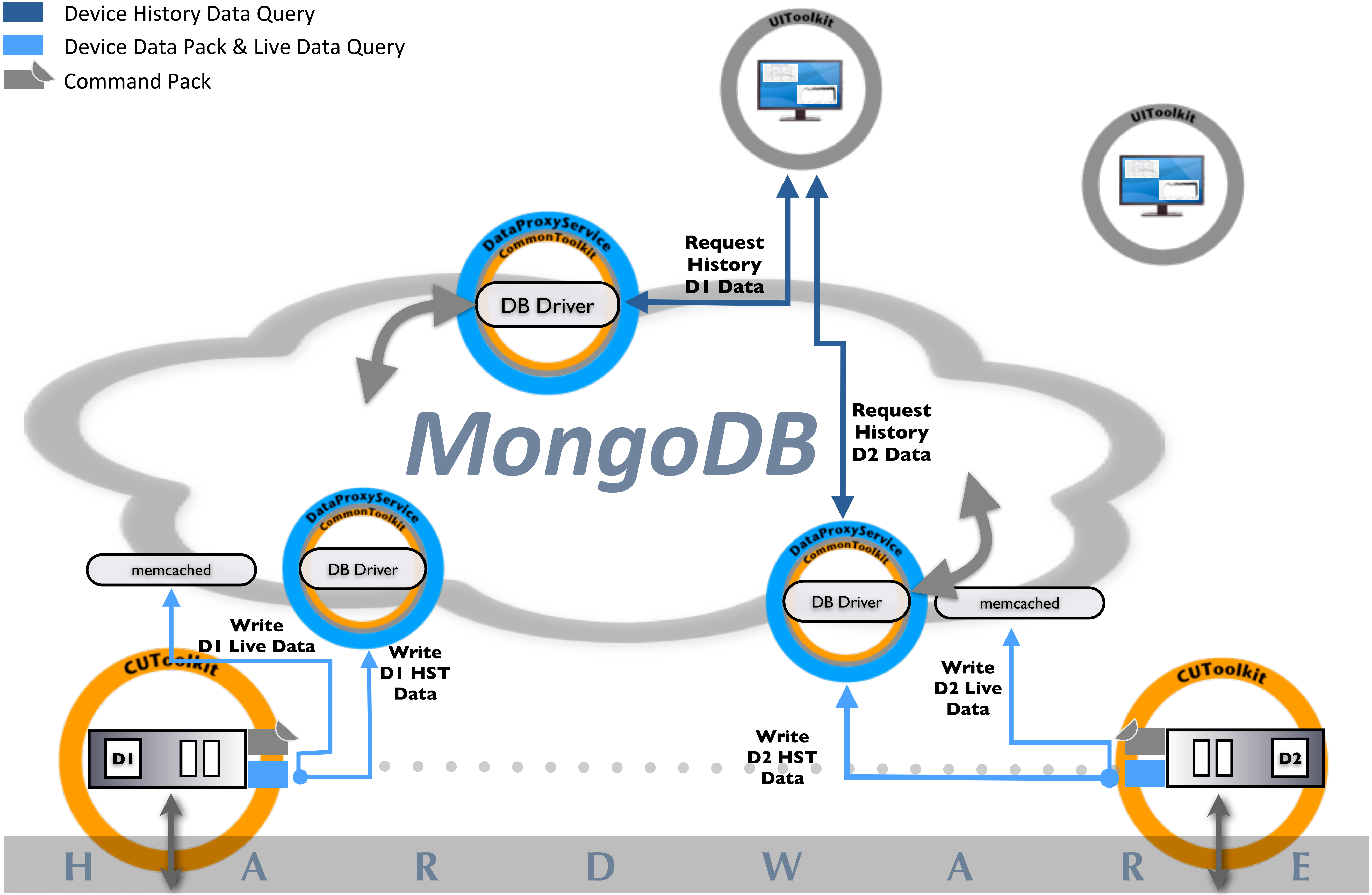


Topology and Data flow

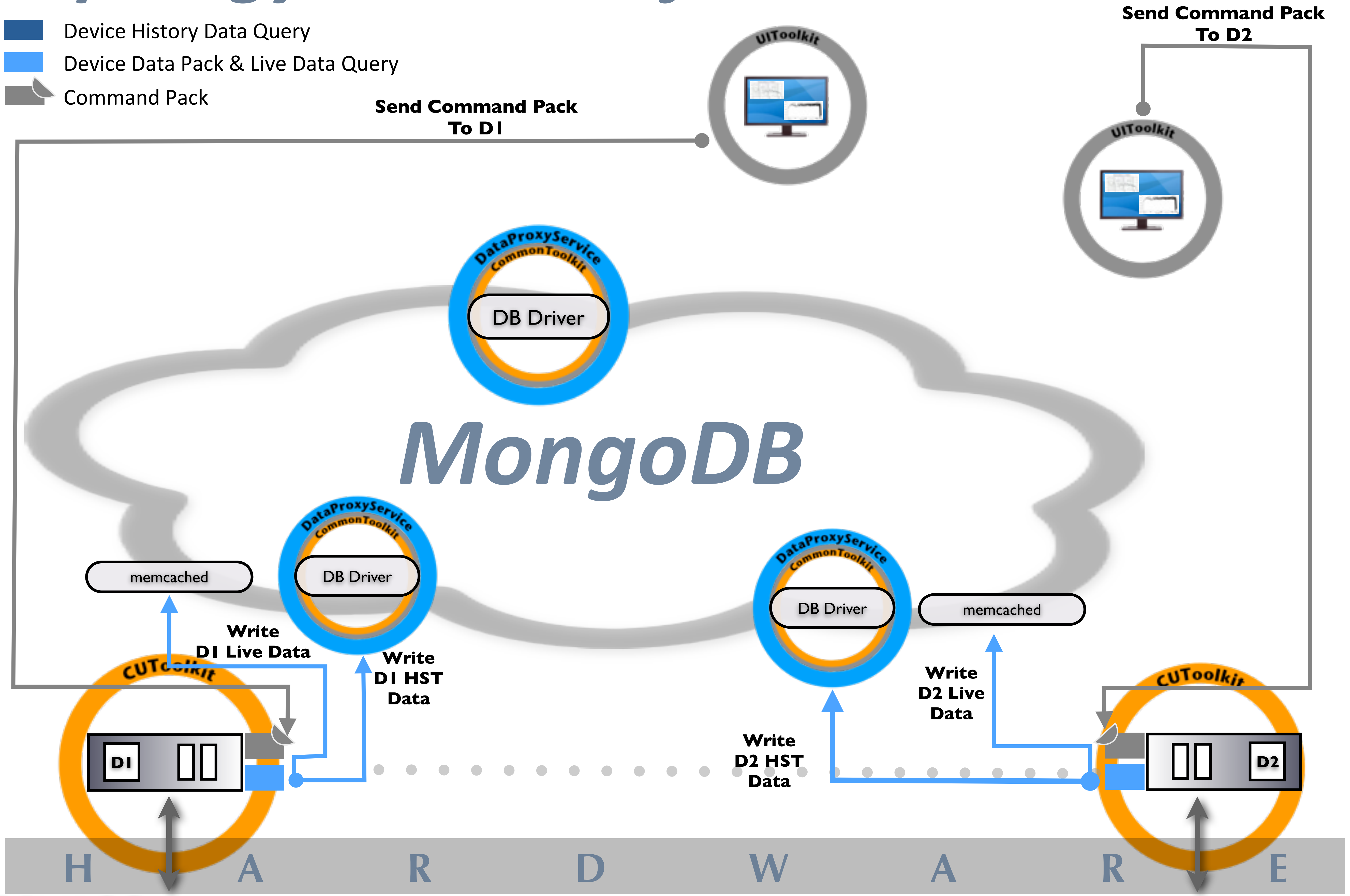
🟦 Device History Data Query

🟦 Device Data Pack & Live Data Query

➡ Command Pack



Topology and Data flow



🟦 Device History Data Query

🟦 Device Data Pack & Live Data Query

▨ Command Pack

Send Command Pack To D1

Send Command Pack To D2

UIToolkit

UIToolkit

DataProxyService CommonToolkit DB Driver

MongoDB

DataProxyService CommonToolkit DB Driver

DataProxyService CommonToolkit DB Driver

memcached

memcached

Write D1 Live Data

Write D1 HST Data

Write D2 HST Data

Write D2 Live Data

CUToolkit D1

CUToolkit D2

H A R D W A R E

Topology Next Step

“issue”

previous flow has an issue, CU needs to use two channels: first is ChaosSQL to write history data, second channel is needed to write directly to memcached

...can we use ChaosQL for both history and live data?

Topology Next Step

“solution?”

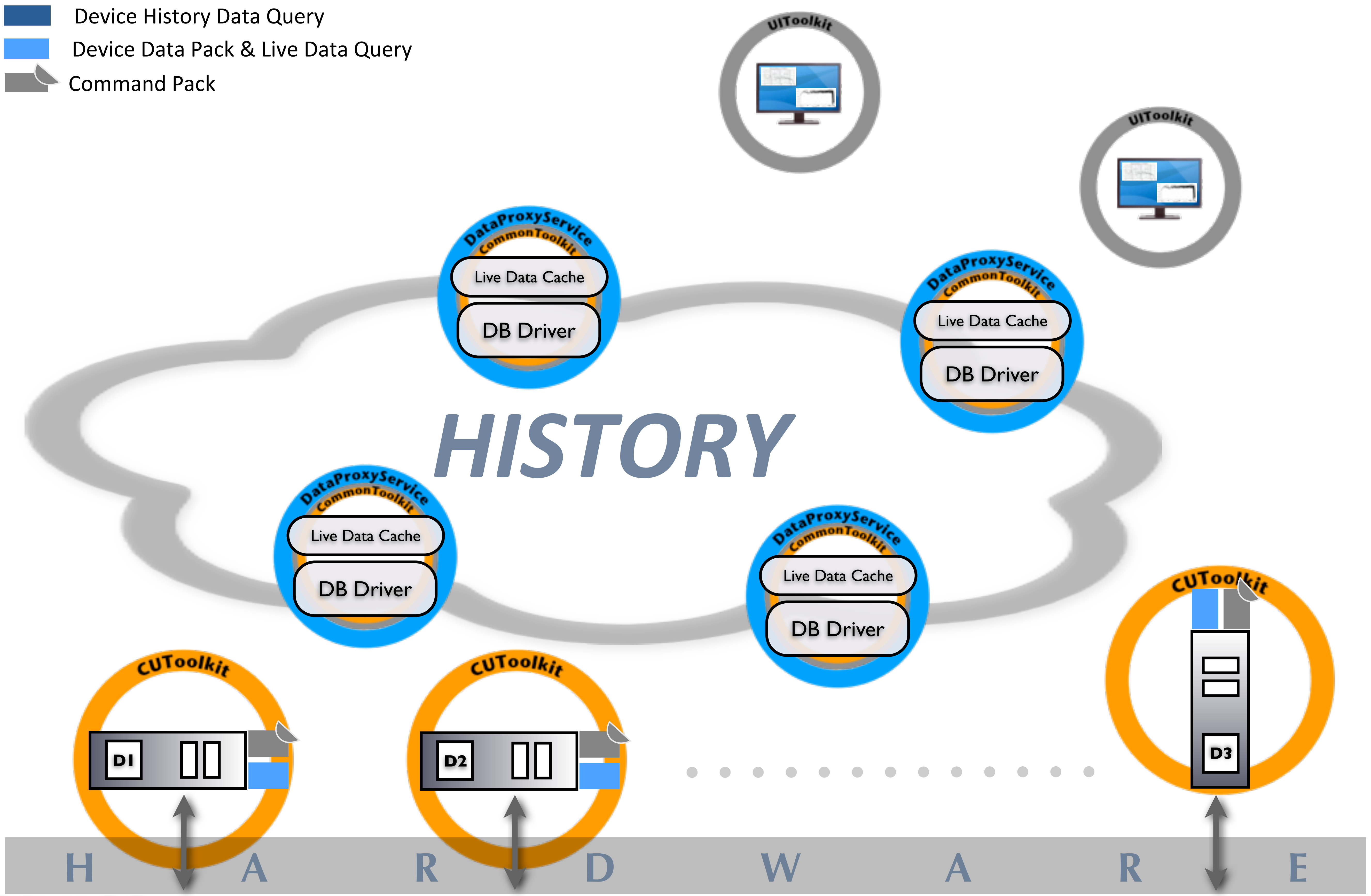
...next version of memcached supports modules and can be embedded...can this be a solution?

...we are considering a little modification of the topology: integration of memcached into the “Data Proxy Service” as “Live Data Cache” service.

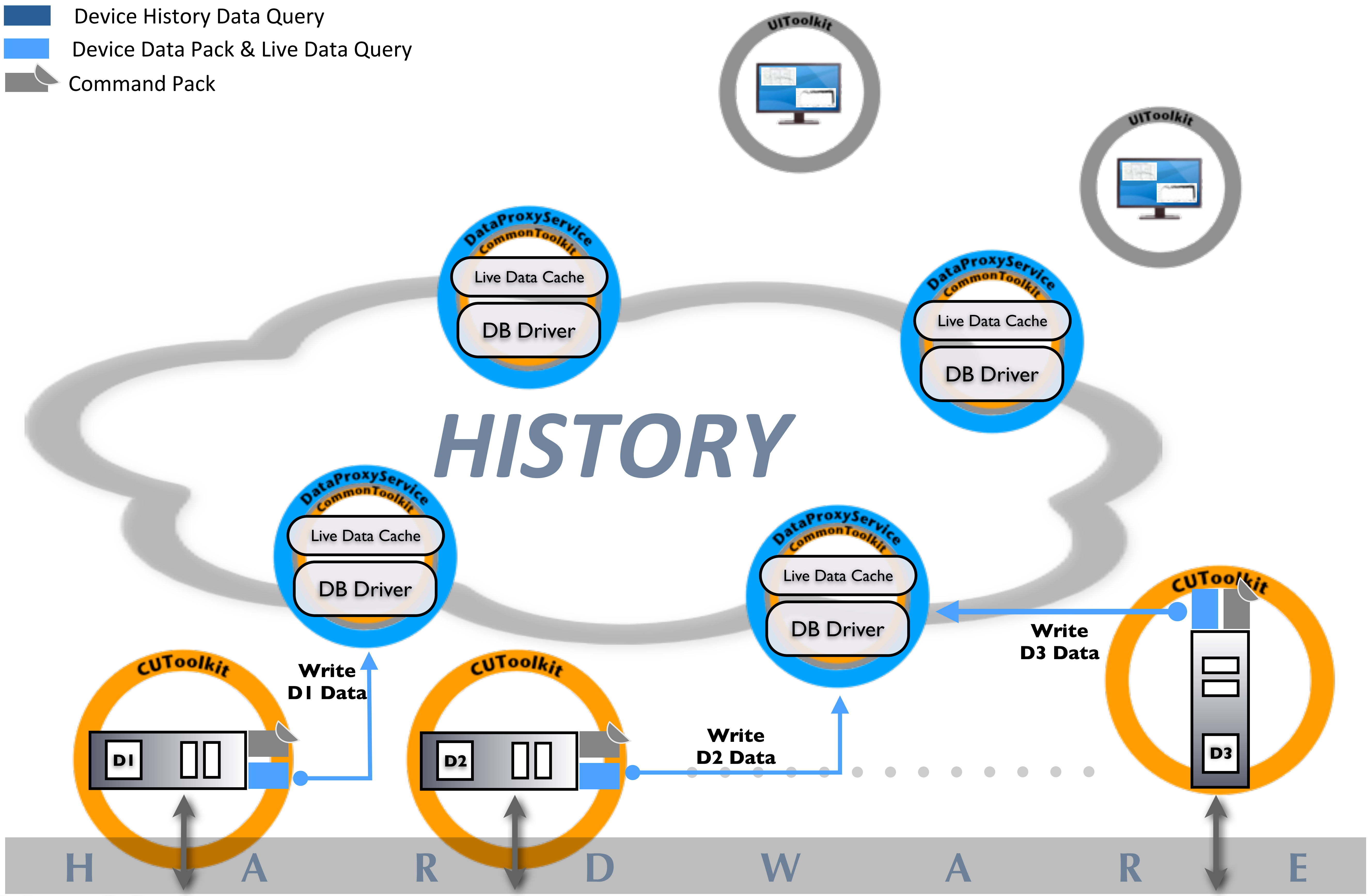
This will permit to:

- reduce the output data flow of CU (write once instead of two times)
- remove the memcached driver from CU and use ChaosQL also for live

Topology and Data flow



Topology and Data flow

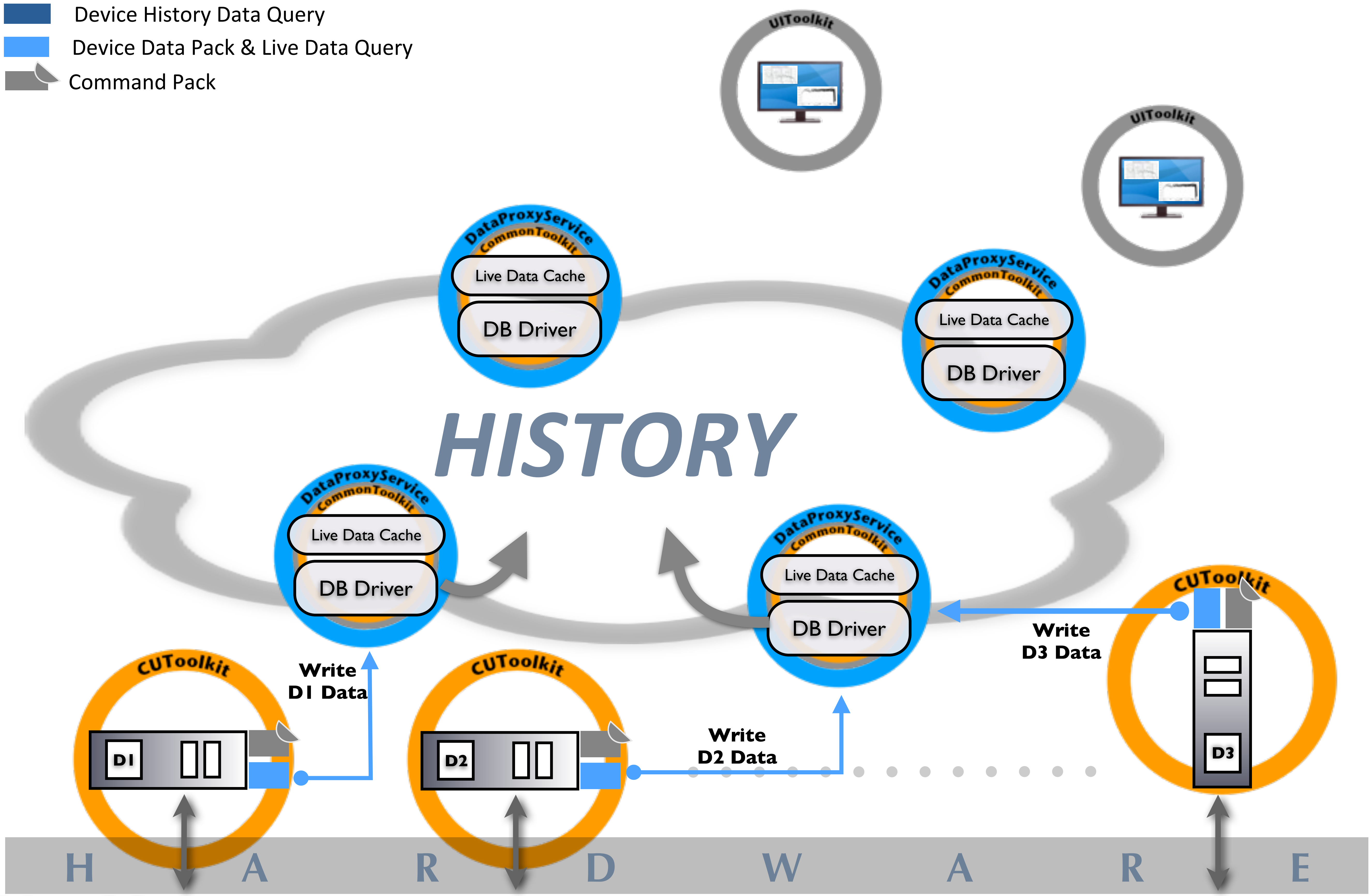


🟦 Device History Data Query

🟦 Device Data Pack & Live Data Query

▨ Command Pack

Topology and Data flow



■ Device History Data Query

■ Device Data Pack & Live Data Query

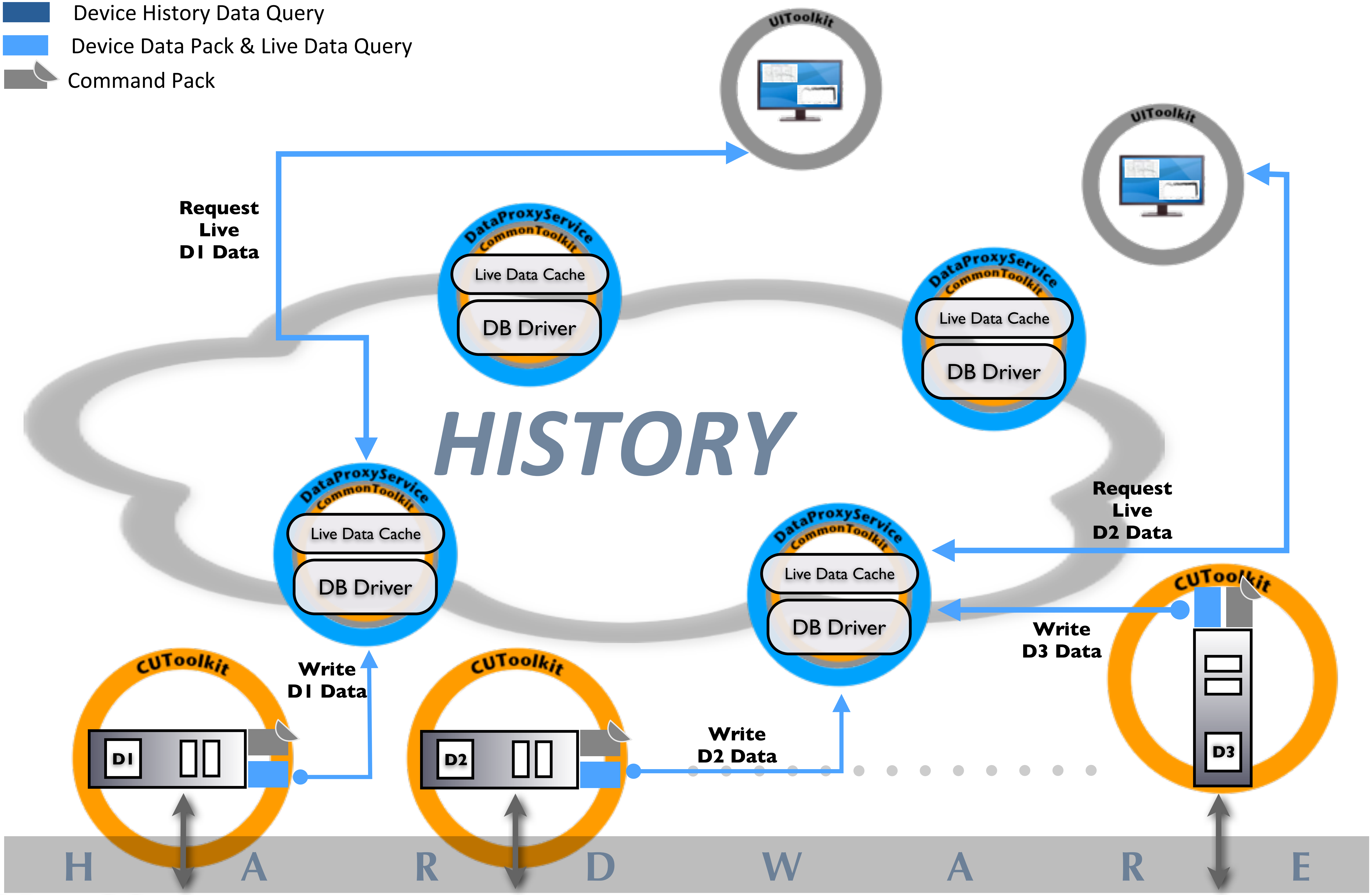
➤ Command Pack

Topology and Data flow

■ Device History Data Query

🟦 Device Data Pack & Live Data Query

▬ Command Pack

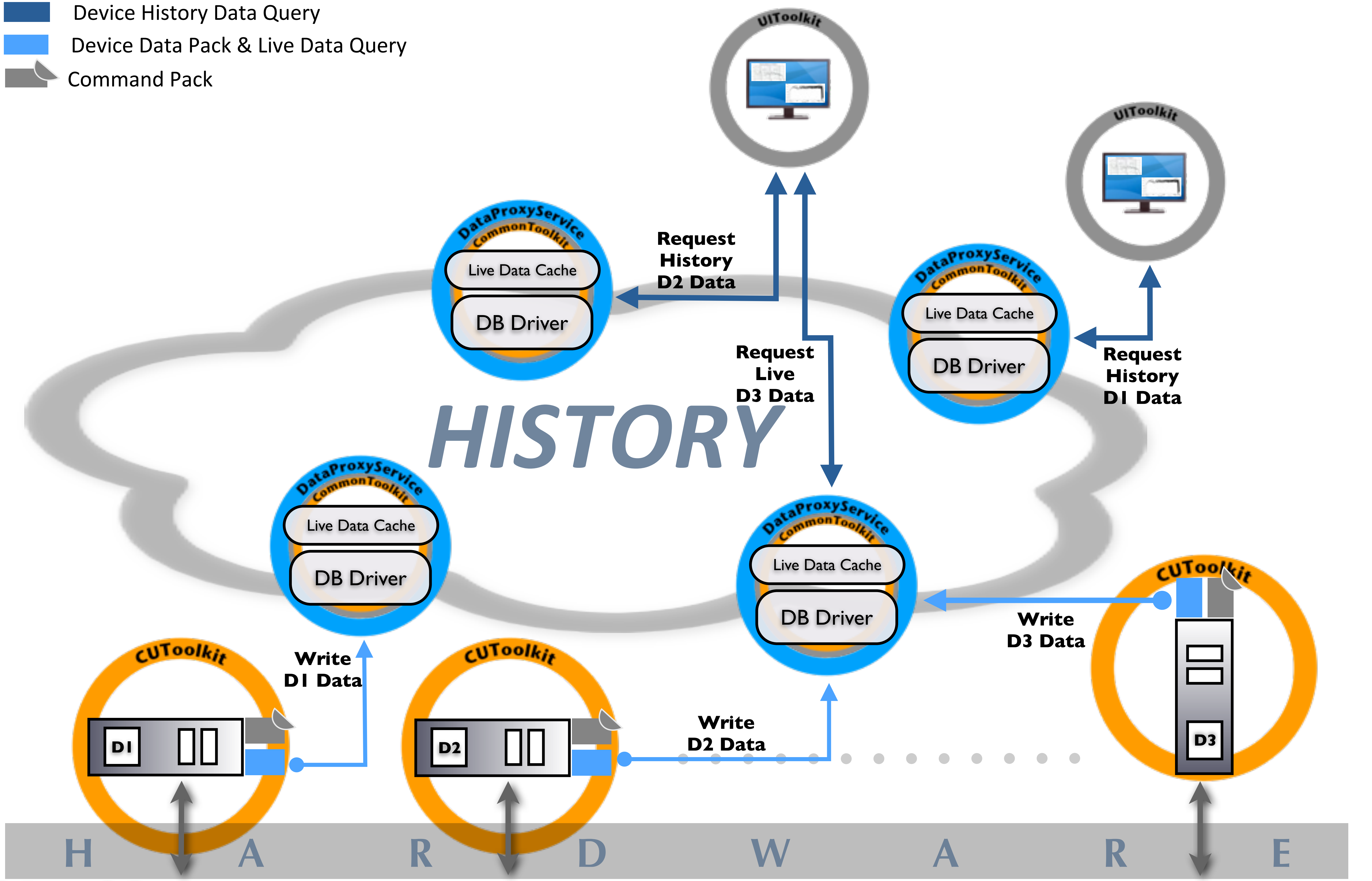


Topology and Data flow

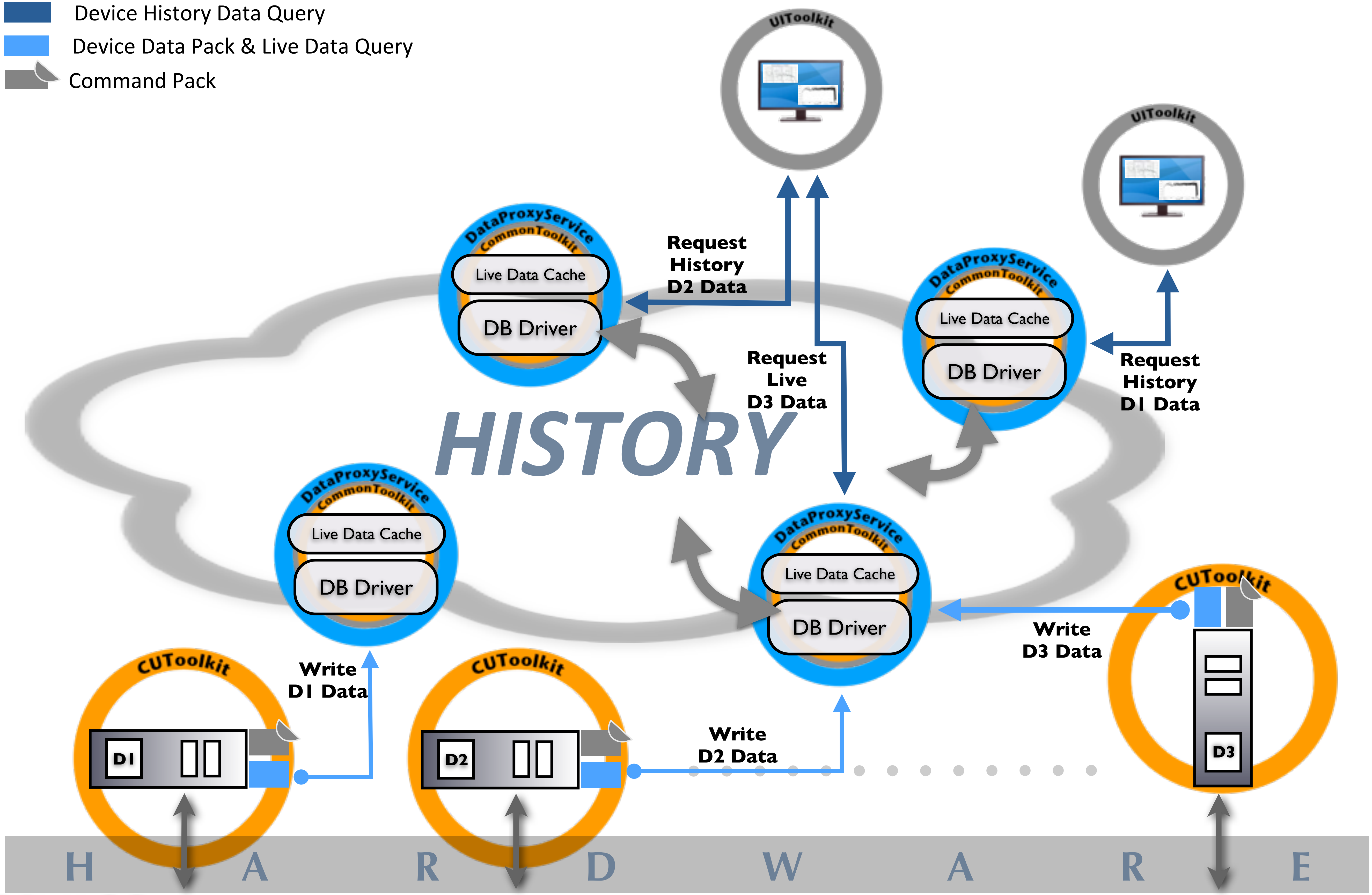
🟦 Device History Data Query

🟦 Device Data Pack & Live Data Query

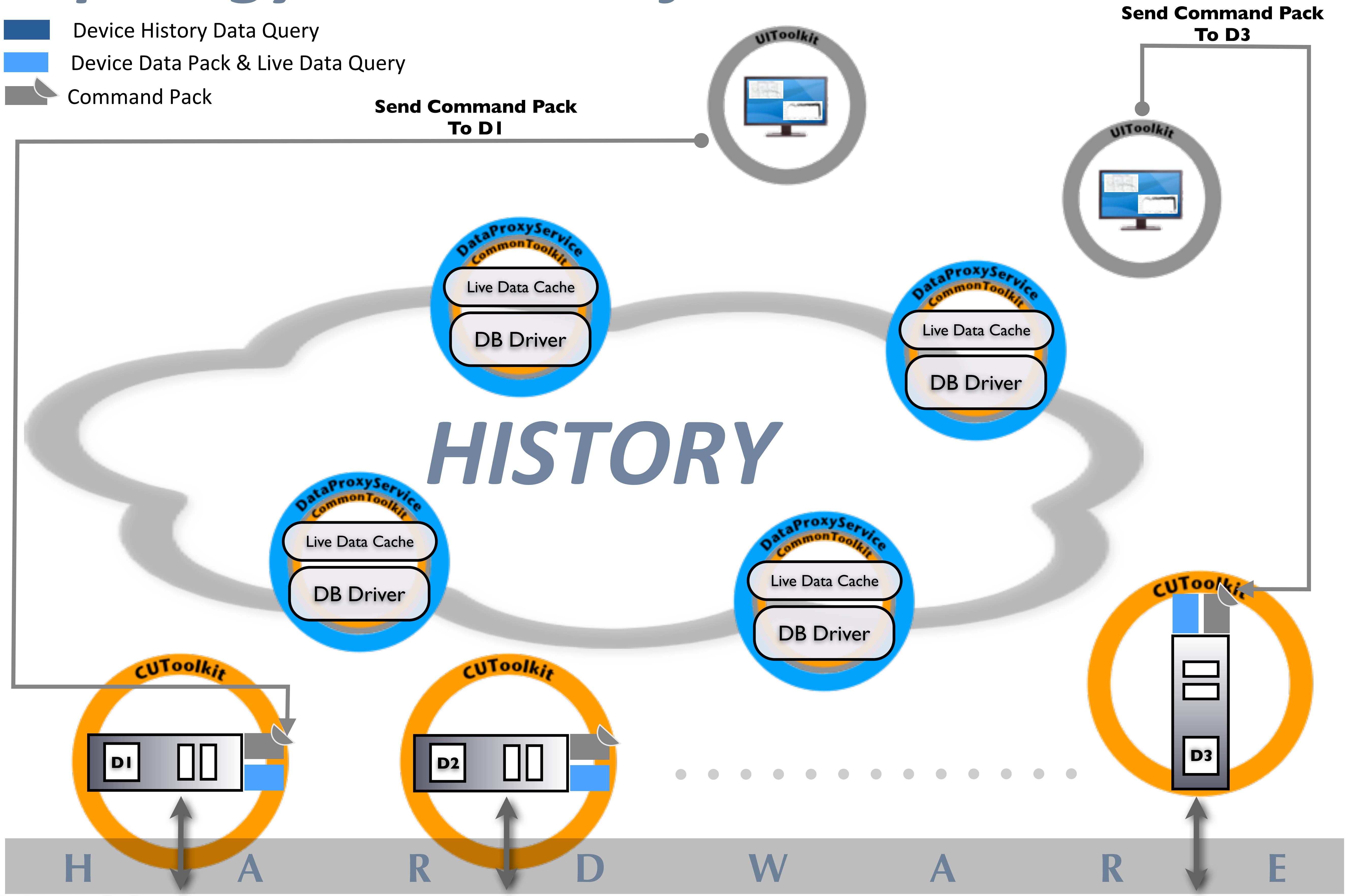
➤ Command Pack



Topology and Data flow



Topology and Data flow



🟦 Device History Data Query

🟦 Device Data Pack & Live Data Query

▨ Command Pack

Topology Next Step

Scaling with *Memcached* and *MongoDB*

Memcached is a key-value cache and scales on key names, each client has an algorithm to link a “key” with “server”

MongoDB scales well on writes and reads (some tricks may be used to increase the write speed)

Topology Next Step

A further improvement for scaling performance (for the last scenario):

achieve the “server scaling” by pre-calculating in which server a CU must write “live” and “history” (send ChaosQL insert message)

every CU will have a list of server; it will begin with the first, when it will not be reachable the second will be used and so on.

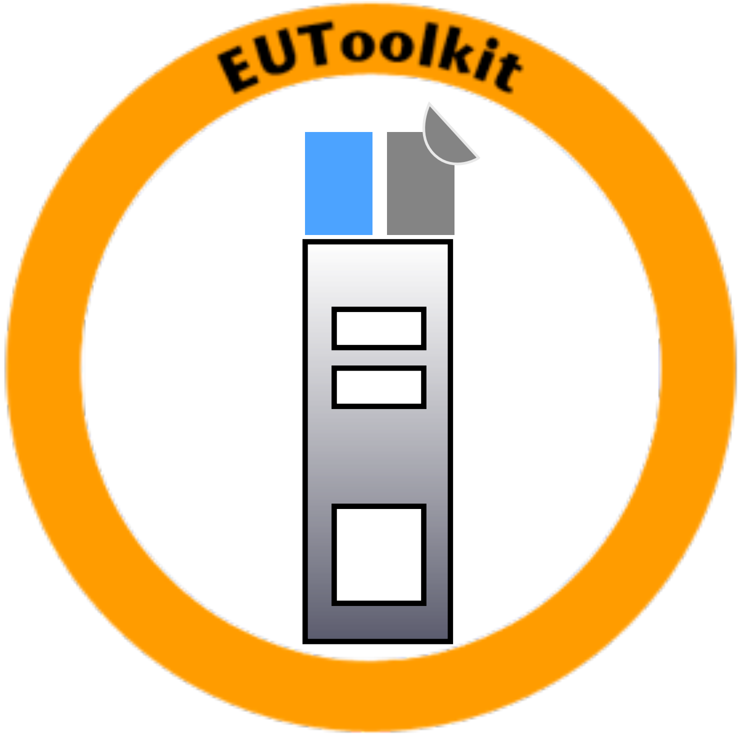
in this way we can scale taking into account the number of CUs and network performance for each server

New Idea for the Control Automation and Computing

An idea for automate the controls

- we are trying to design and add a new “node” into CHAOS
- it will be like a Control Unit but modelled to be used only for controlling other CU or computations, slow controls, etc.
- it will be used for creating a distributed final state machine
- we called it “Execution Unit”

Execution Unit Node



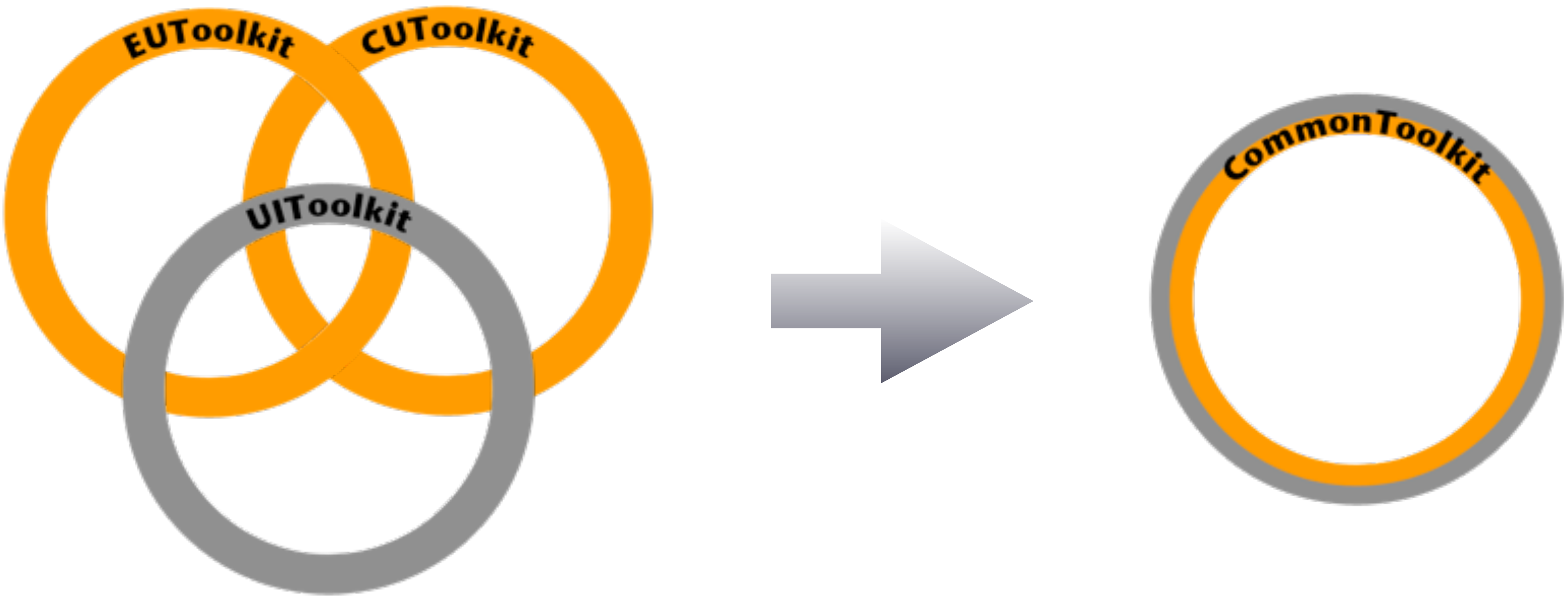
Execution Unit, is a specialized software that implement controls or computing algorithms

🟦 ChaosQL Data Pack Channel

⬛ Chaos Command Pack Channel

ExecutionUnit must define the input and output class of data (HW Dataset or Basic Element) that are needed to do the work

!CHAOS framework with new layer

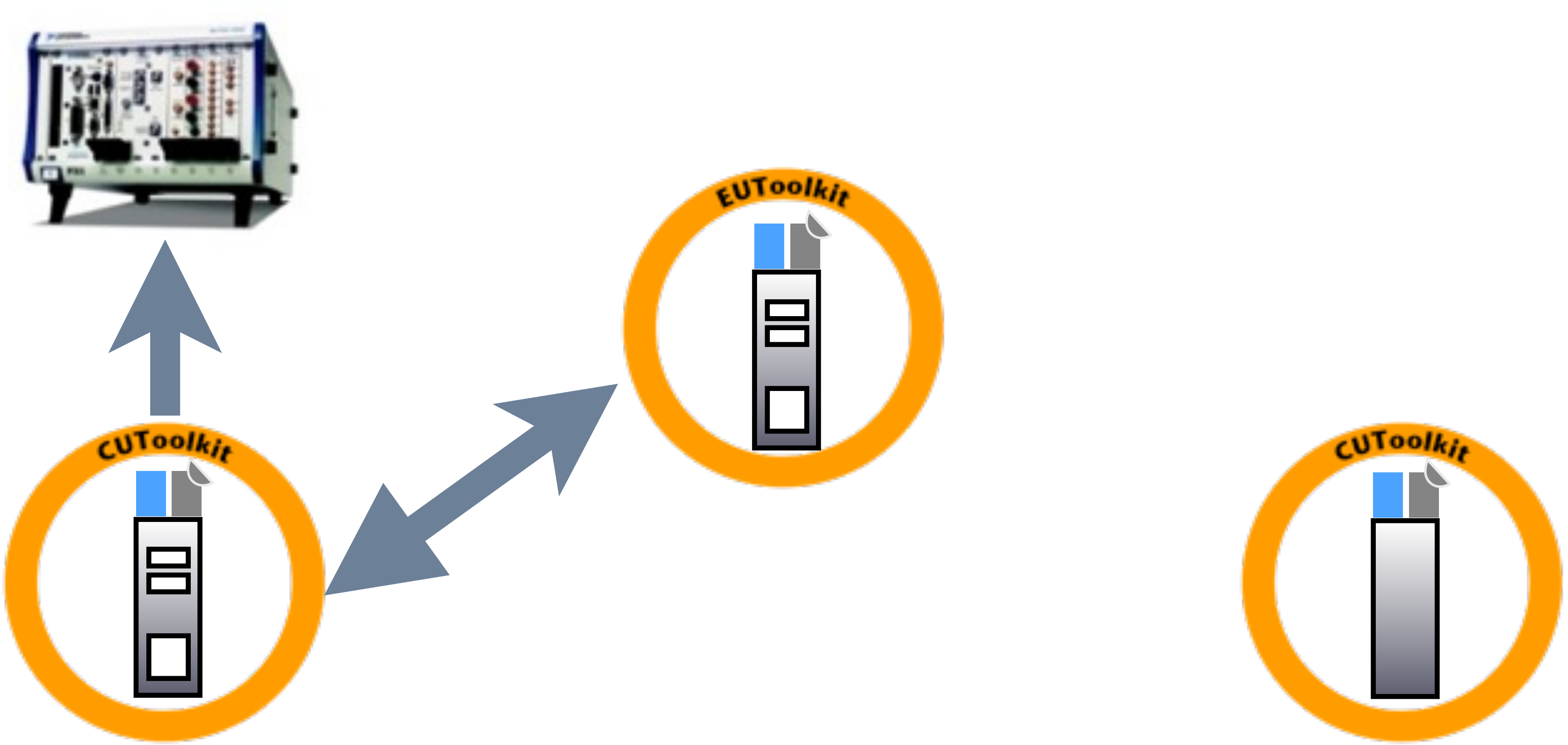


CUToolkit (as for previous Layer) is implemented on CommonToolkit

ExecutionUnit in the work

Execution Unit Example 1

The EU read the data from the **Output** attribute of the HW from the CU_1 data

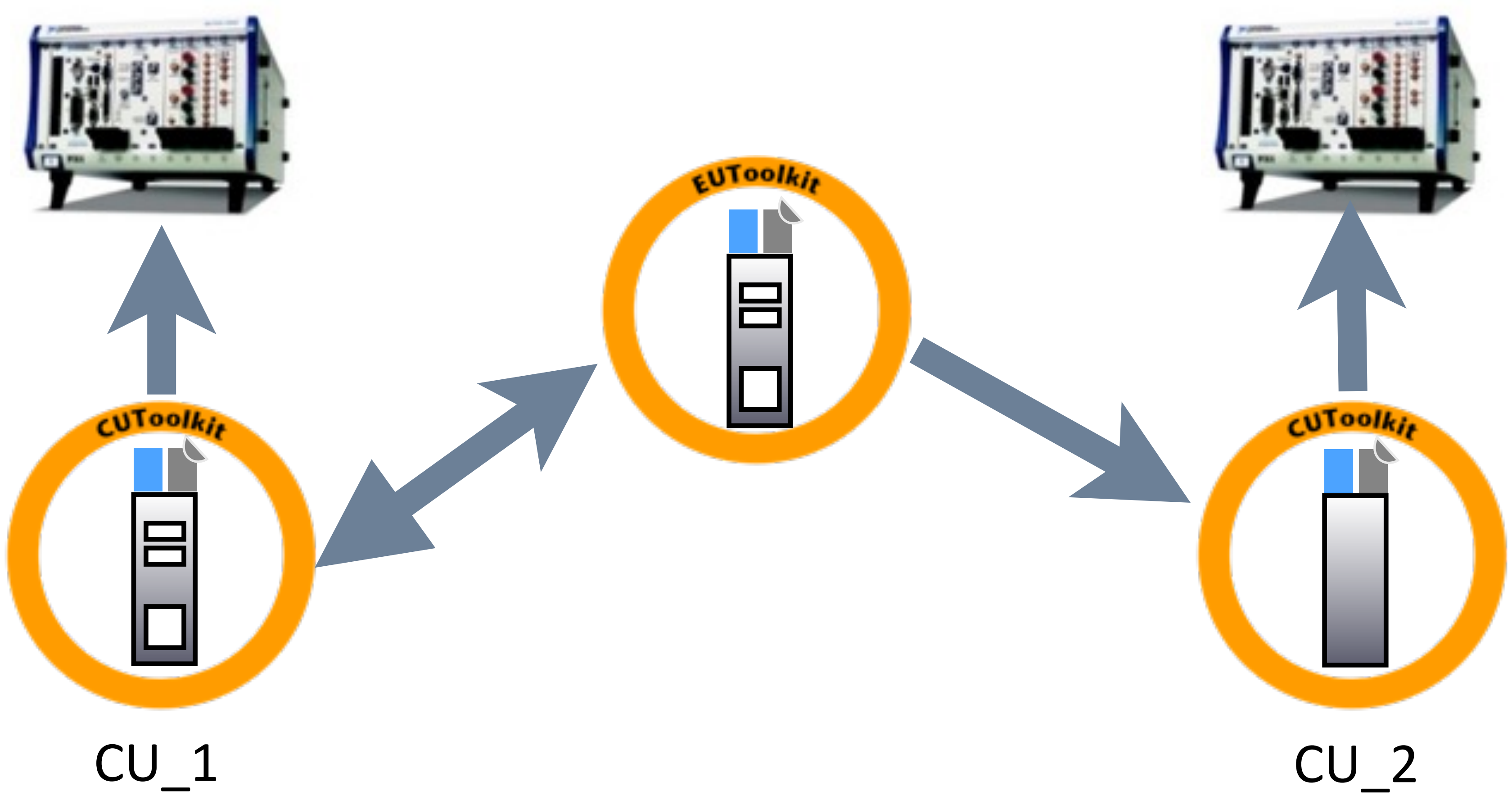


CU_1

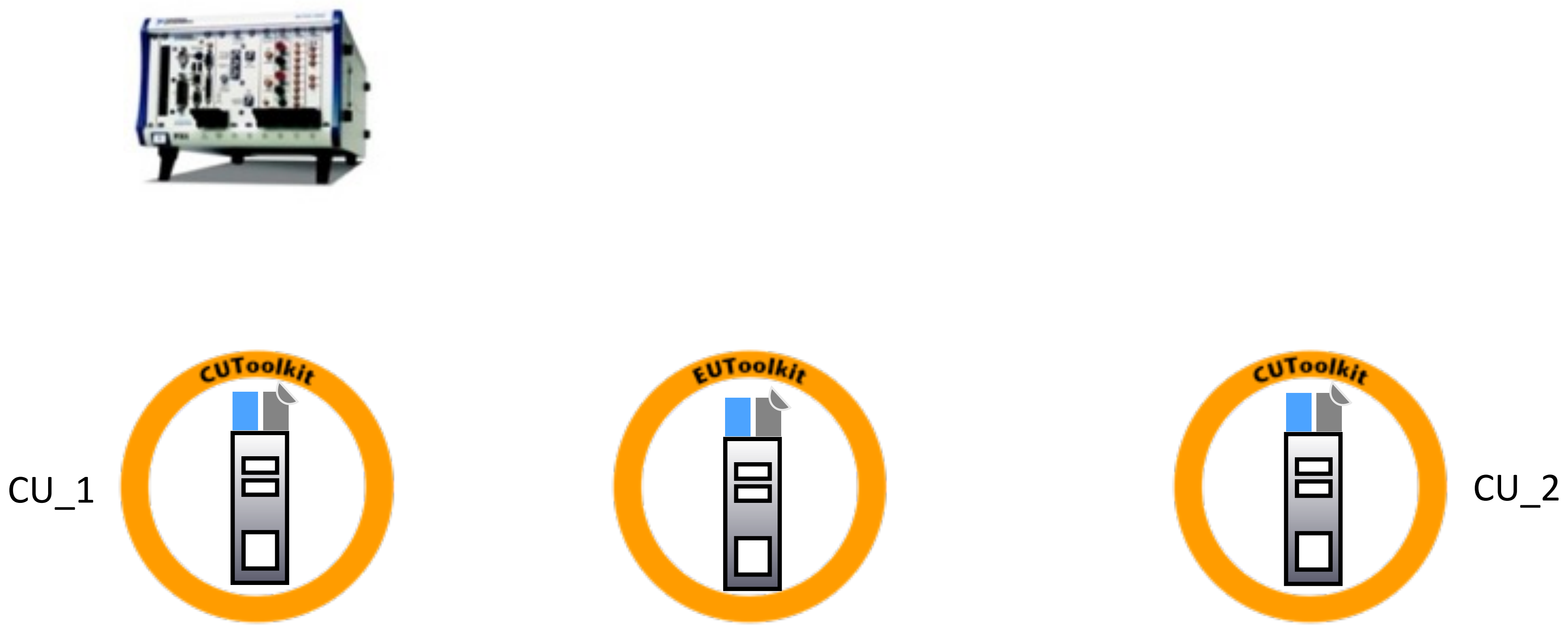
CU_2

Execution Unit Example 1

The EU write the result of computation to a CU_2 for set the **Input** attribute of the HW



Execution Unit Example 1



CU_1

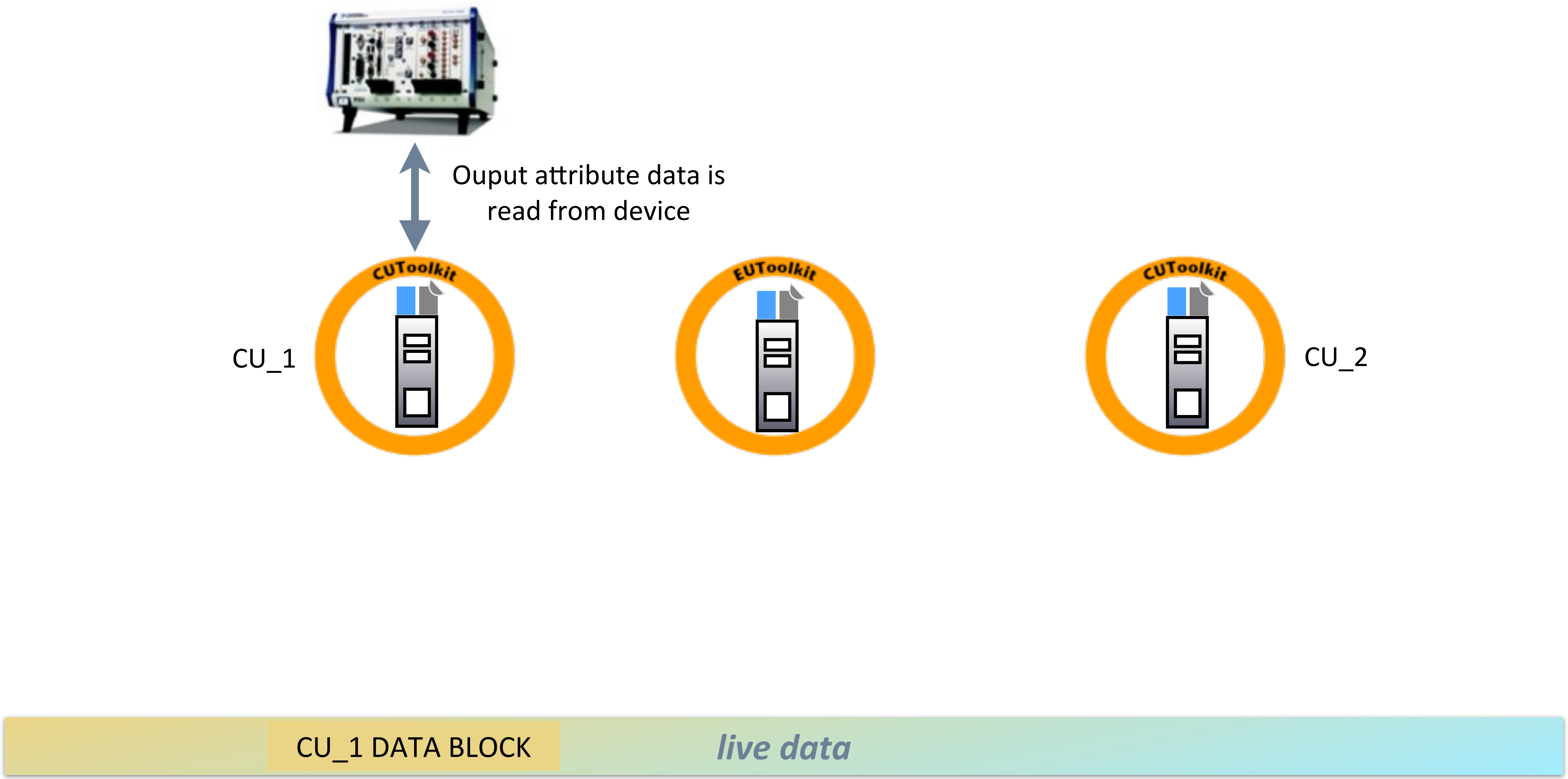
CU_2

CU_1 DATA BLOCK

live data

Execution Unit Example 1

this is the real data flow



Ouput attribute data is read from device

CU_1

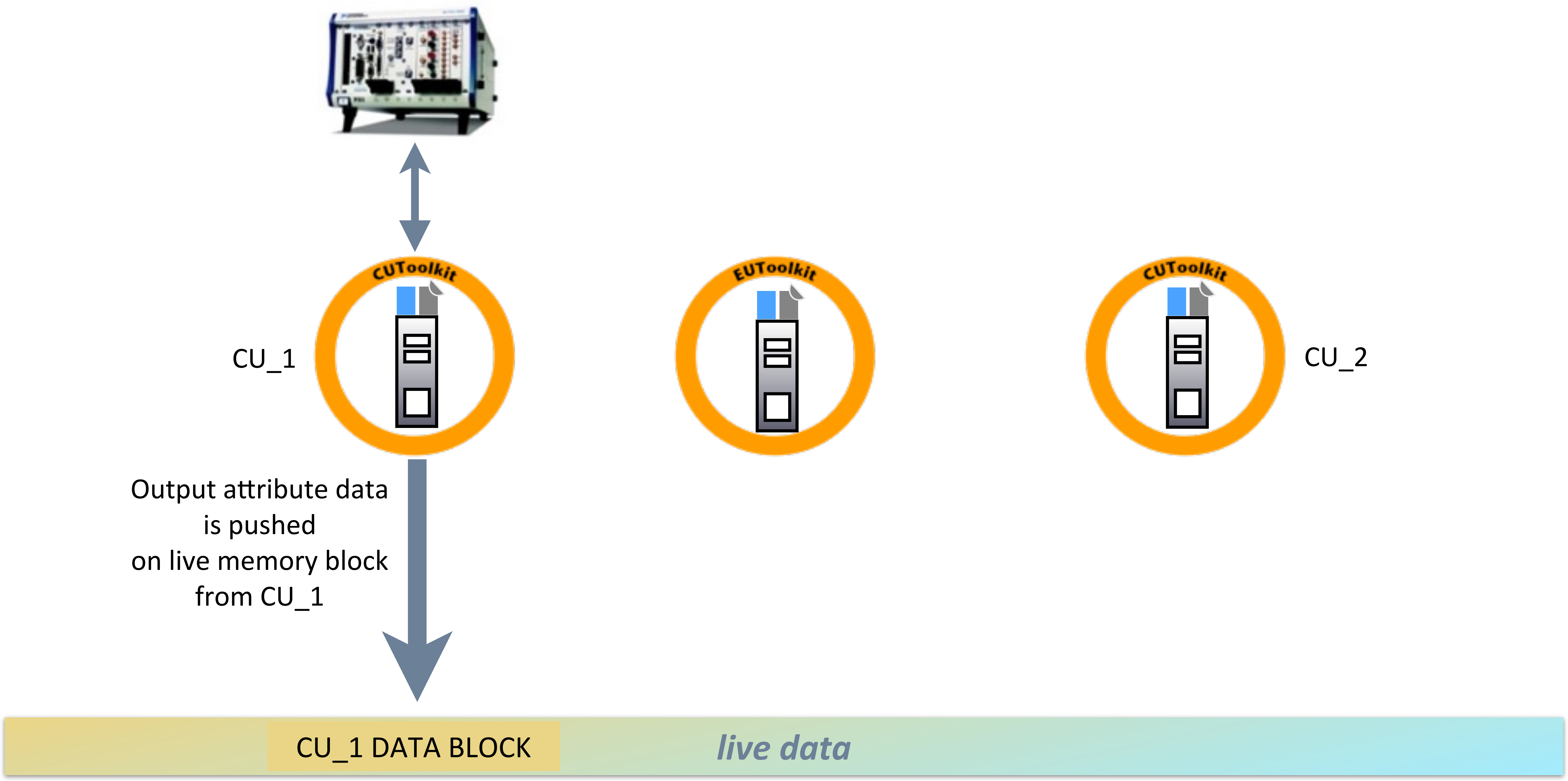
CU_2

CU_1 DATA BLOCK

live data

Execution Unit Example 1

this is the real data flow



CU_1

CU_2

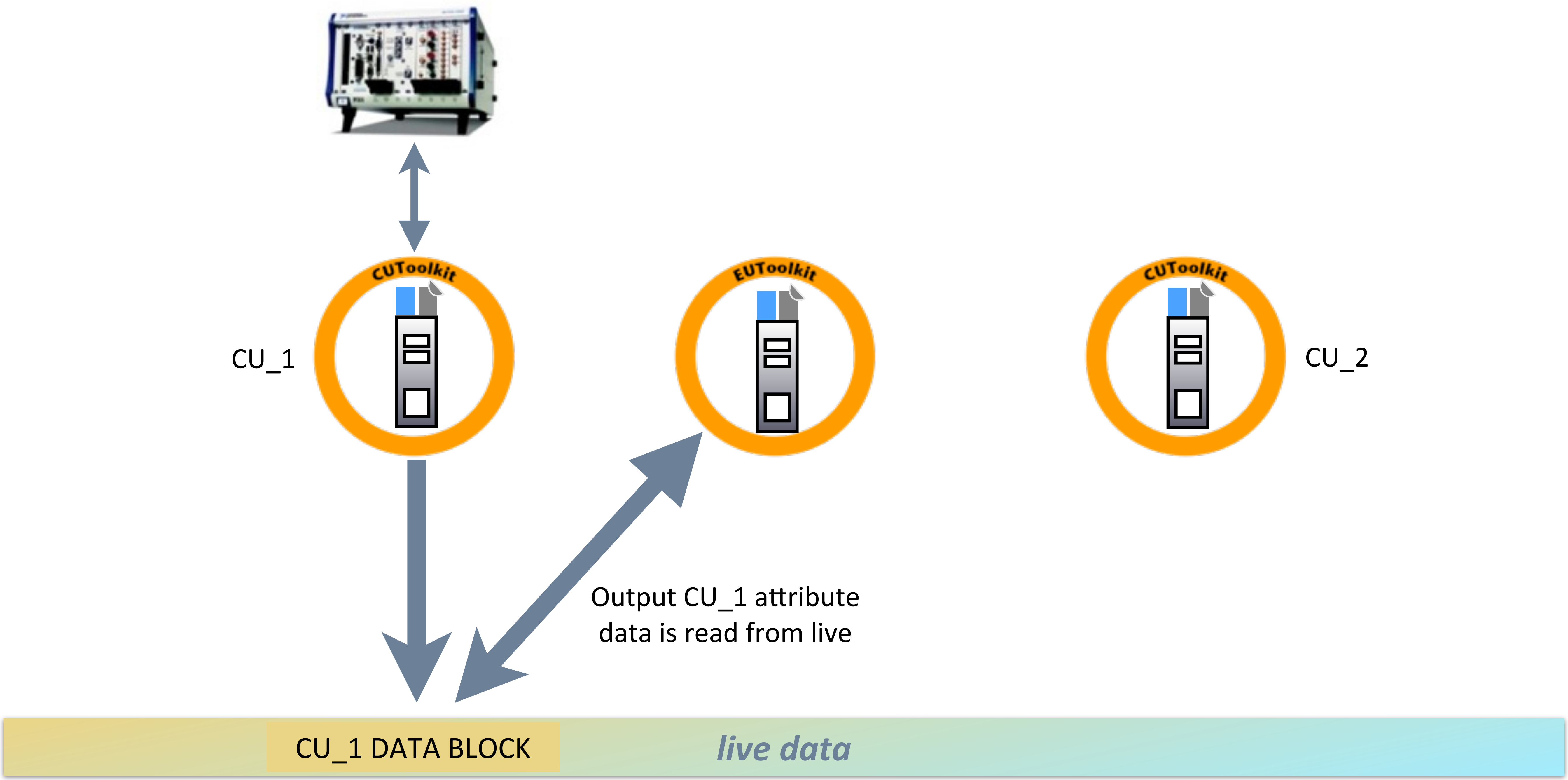
Output attribute data
is pushed
on live memory block
from CU_1

CU_1 DATA BLOCK

live data

Execution Unit Example 1

this is the real data flow



CU_1

CU_2

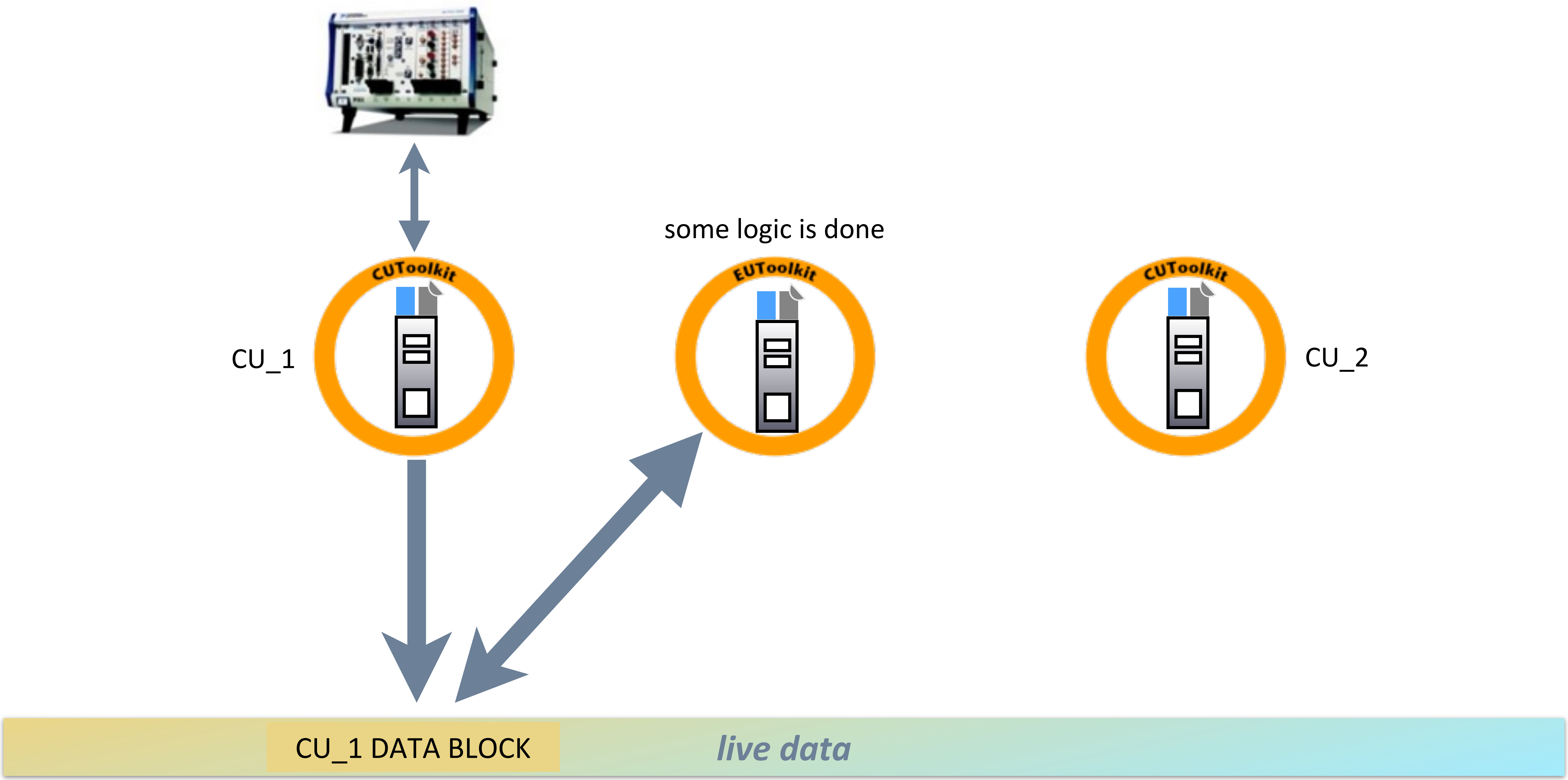
Output CU_1 attribute data is read from live

CU_1 DATA BLOCK

live data

Execution Unit Example 1

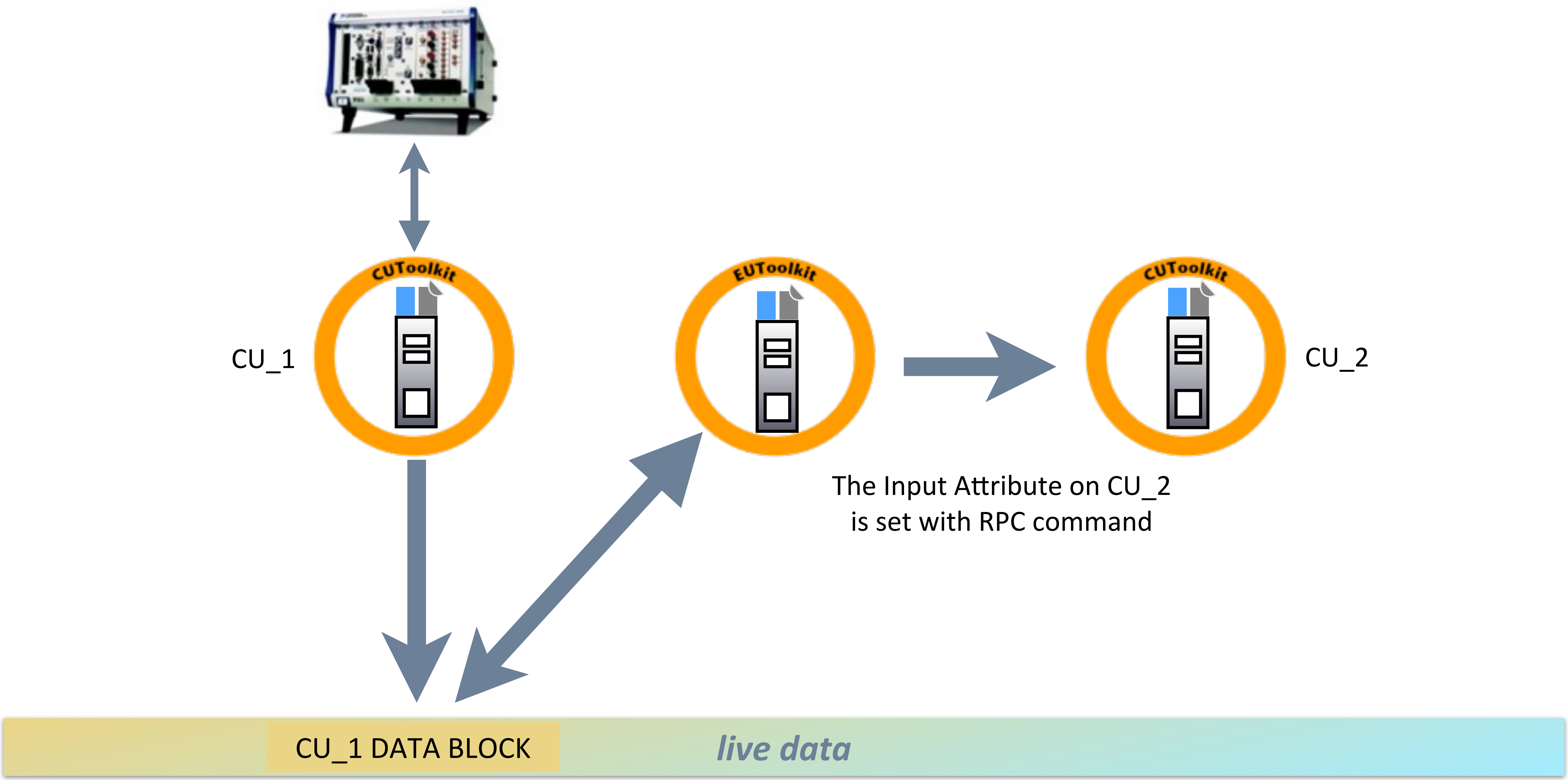
this is the real data flow



some logic is done

Execution Unit Example 1

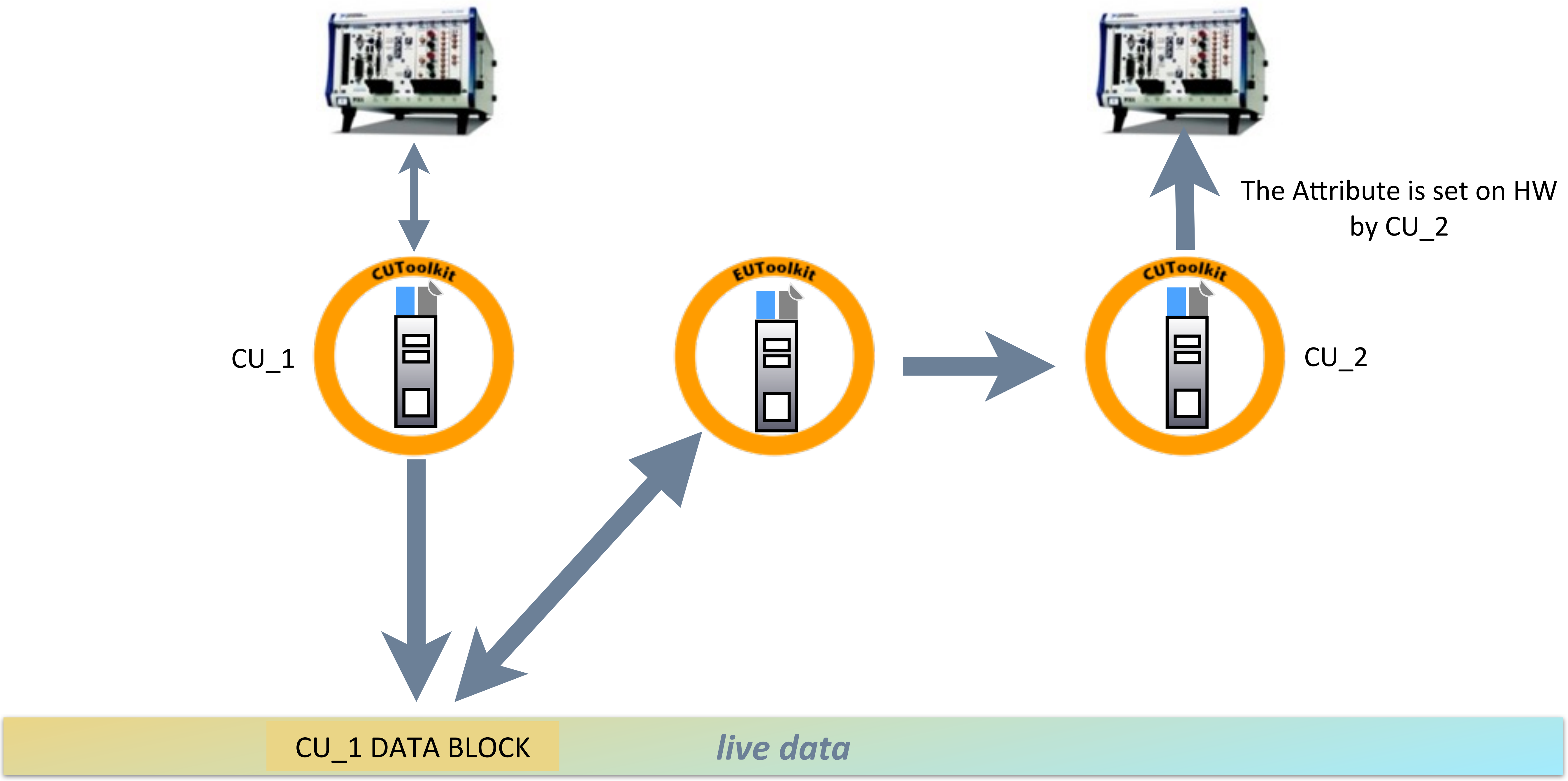
this is the real data flow



The Input Attribute on CU_2 is set with RPC command

Execution Unit Example 1

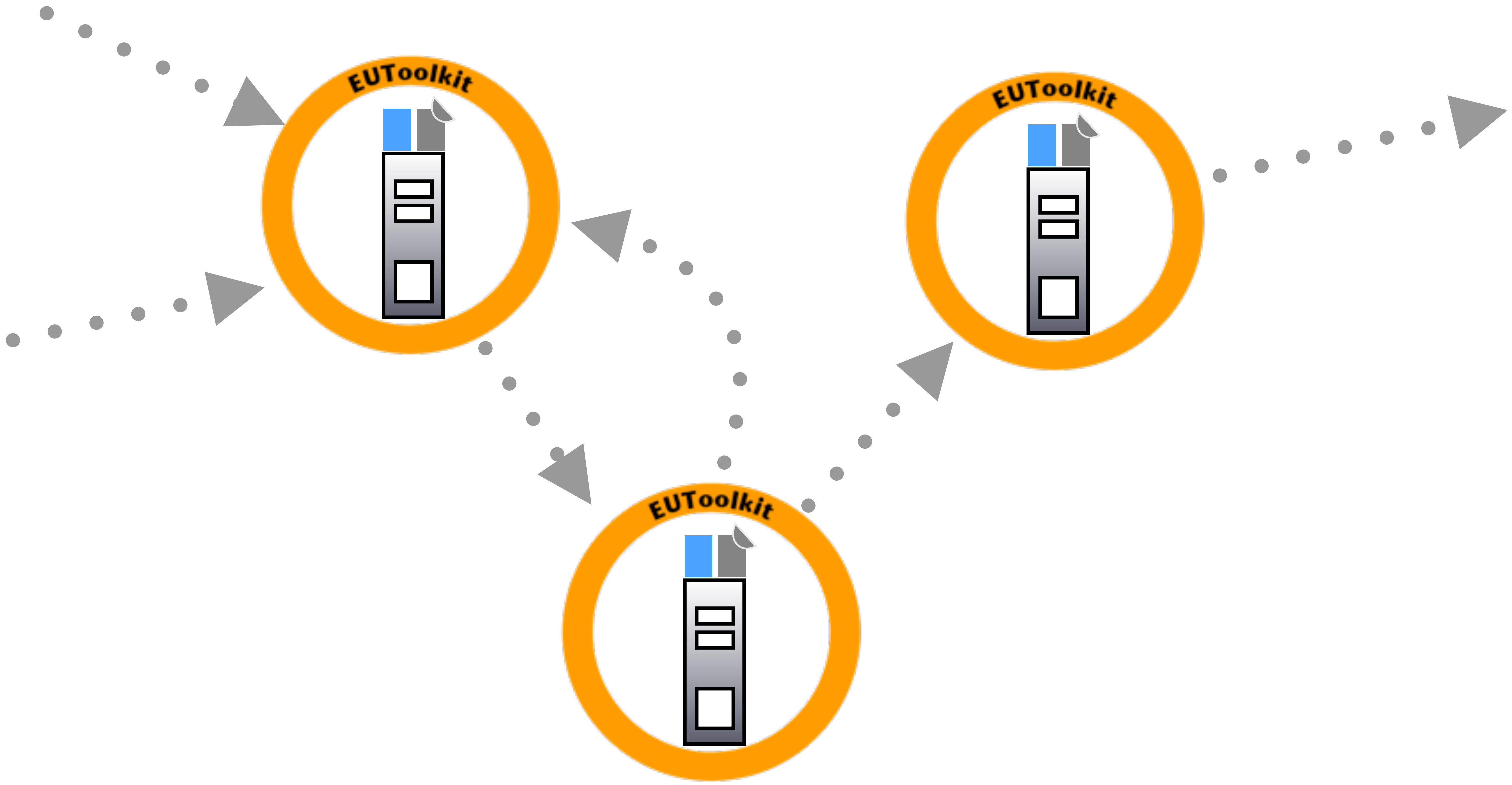
this is the real data flow



The Attribute is set on HW by CU_2

Execution Unit Example 2

Now think to an execution unit that send output to another execution unit and so on...



!CHAOS compared to a Normal PC

- CHAOS can be considered like a distributed computer:
  - Live data is the RAM
  - History data is the Hard Disk
  - CU are the kernel driver
  - EU are the process that do something

Common Toolkit

Common Toolkit

- CommonToolkit has tree important software layers
  - BSON Container for hardware dataset abstraction and RPC pack
  - RPC Driver
  - ChaosQL Driver

- in addition it has a lot of common utility code

CommonToolkit

abstraction

abstraction

- CHAOS use BSON (http://bsonspec.org/) for data description and serialization
- it is used in:
  - Hardware attribute description
  - RPC message between node

abstraction

- BSON is a JSON-like binary document
- a key-value document where a value can be:
    - basic types:
        - int32
        - int64
        - double
        - cstring
        - byte
    - BSON document
    - Array

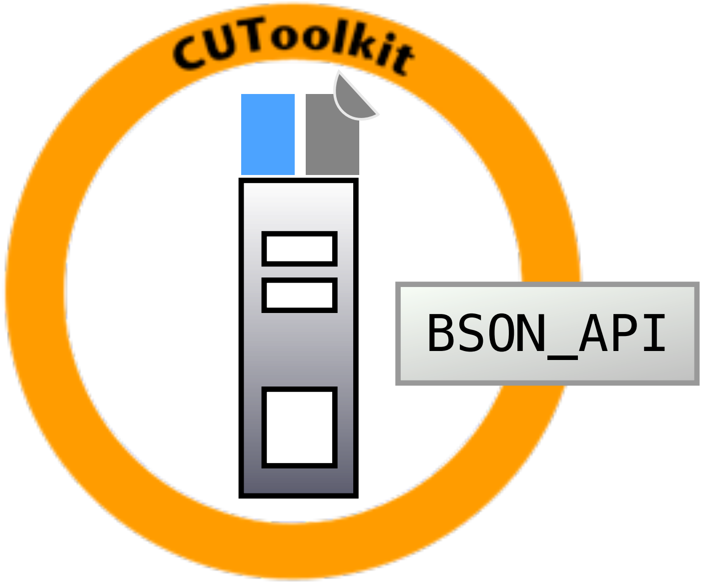
Hardware abstraction

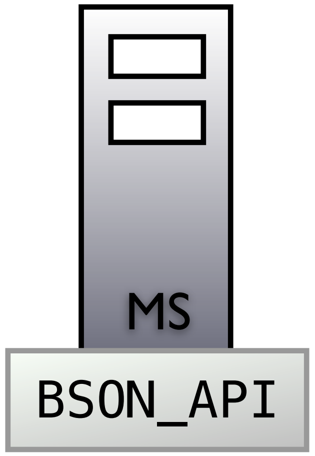
Hardware abstraction is done in two point:

- **Control Unit is needed to connect an Hardware or other “Software Application” to CHAOS system**
- **BSON Serialization is used for describe the hardware property**

Hardware abstraction

Hardware is attached and controlled by CU





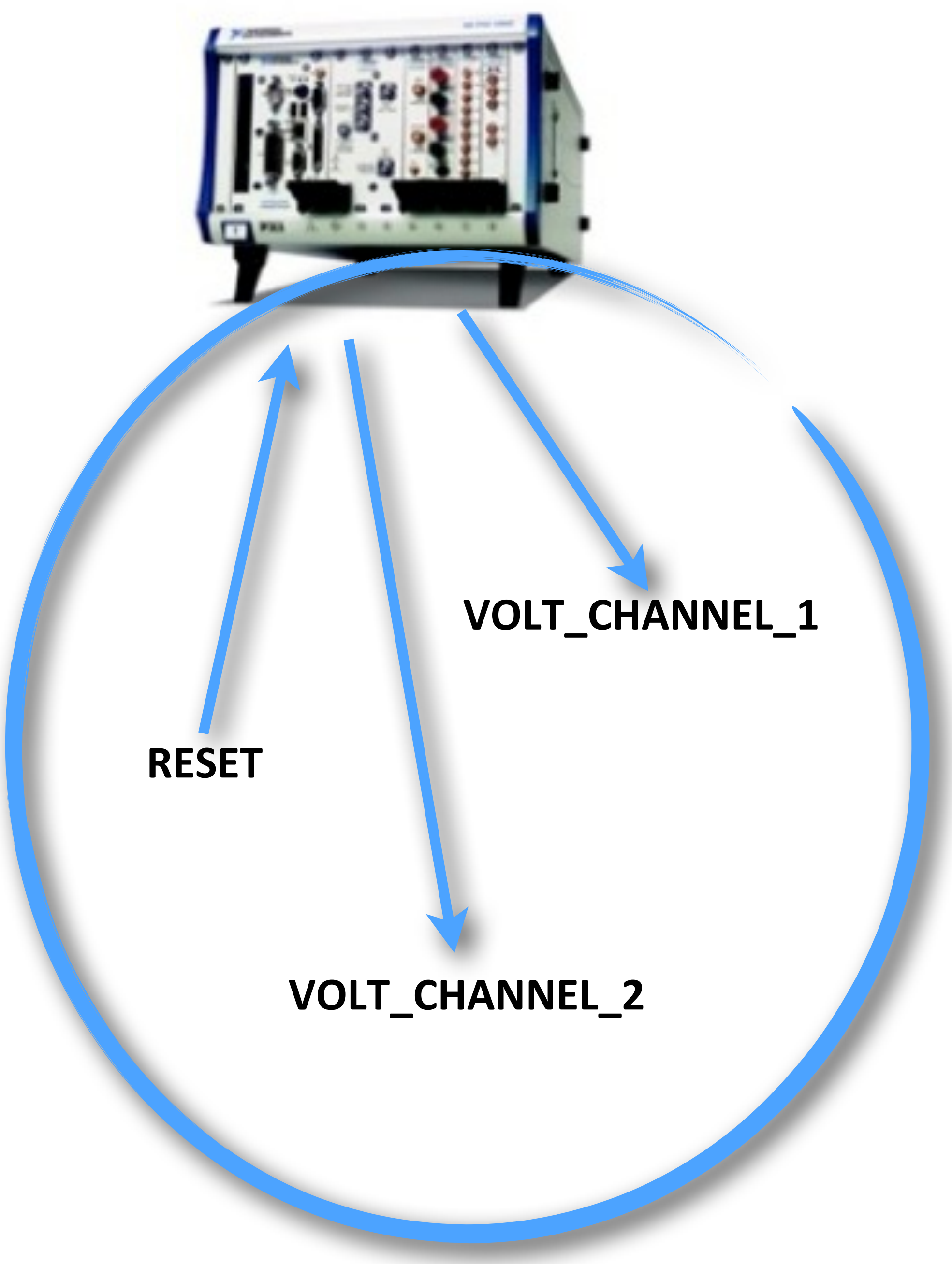
CU or MS expose HW DATASET

Hardware DATASET

Hardware DATASET

- Hardware attribute are described within DATASET
- Each attribute is defined by
    - Name
    - Description
    - Type (kind+basic type ex: VOLT32, CUSTOM_STR, etc...)
    - Cardinality (single or array)
    - Flow (Input, Output, Bidirectional)
    - Range

Hardware DATASET



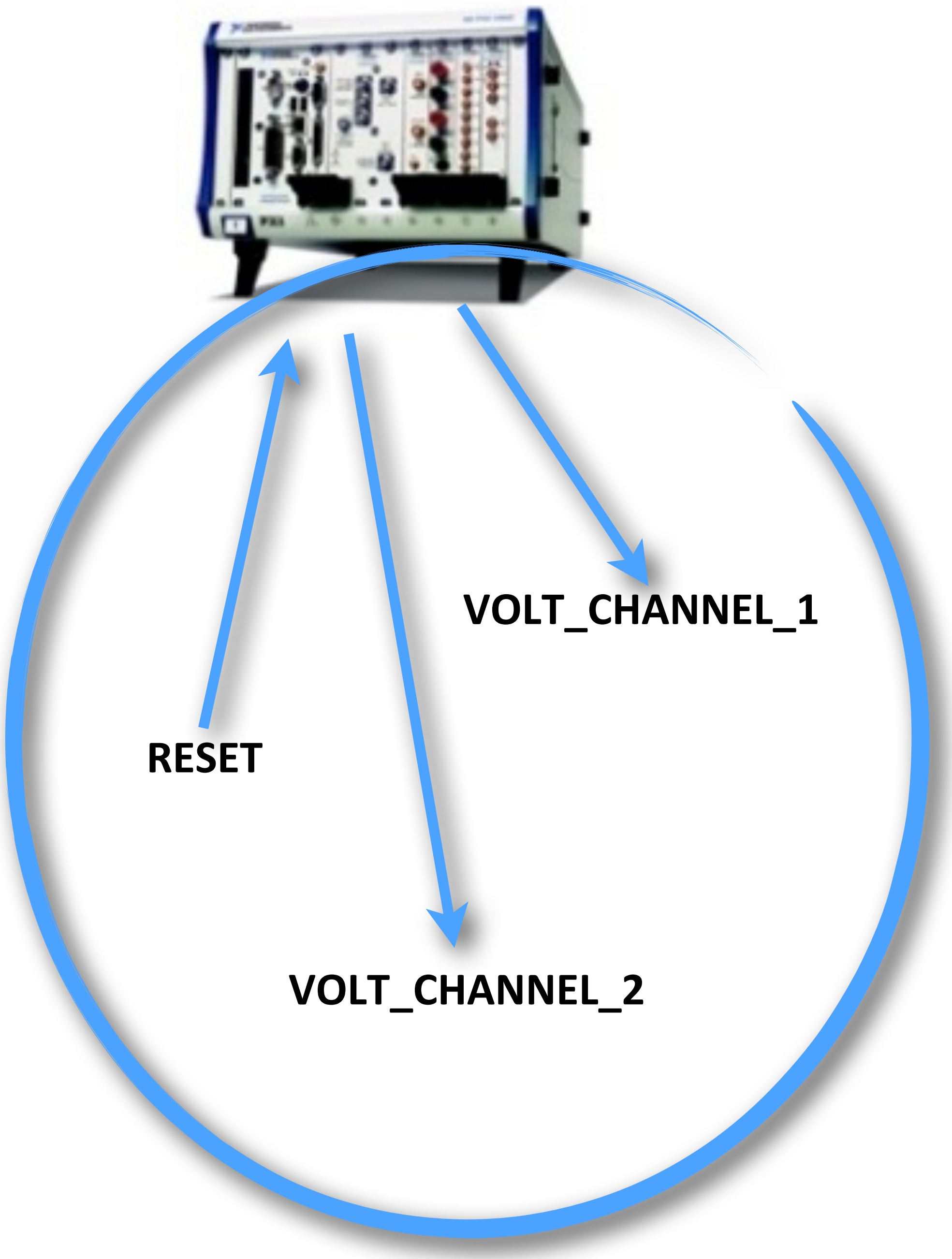
DATASET

name:VOLT_CHANNEL_1
type:VOLT32
flow:output
range:1-20
card: 1

name:VOLT_CHANNEL_2
type:VOLT32
flow:output
range:1-20
card: 1

name: RESET
type: BYTE
flow: input
card: 1

Hardware DATASET



The JSON visualization
of BSON DS representation

```
{“device_id”: DEVICE_ID

 “device_ds”: {
  "name"        : "VOLT_CHANNEL_1",
  "desc"        : "Output volt...",
  "attr_dir"    : 1,
  "type"        : 4,
  "cardinality" : 1
    }

....

}
```

CommonToolkit

RPC System

RPC System

- *RPC System is implemented as a plug-ins System.*
- *It's is abstracted to internal CHAOS framework so we con change it*
- *The RPC layer is only used to send and receive bson data*
- *all the information are in the BSON pack*

RPC System

BSON

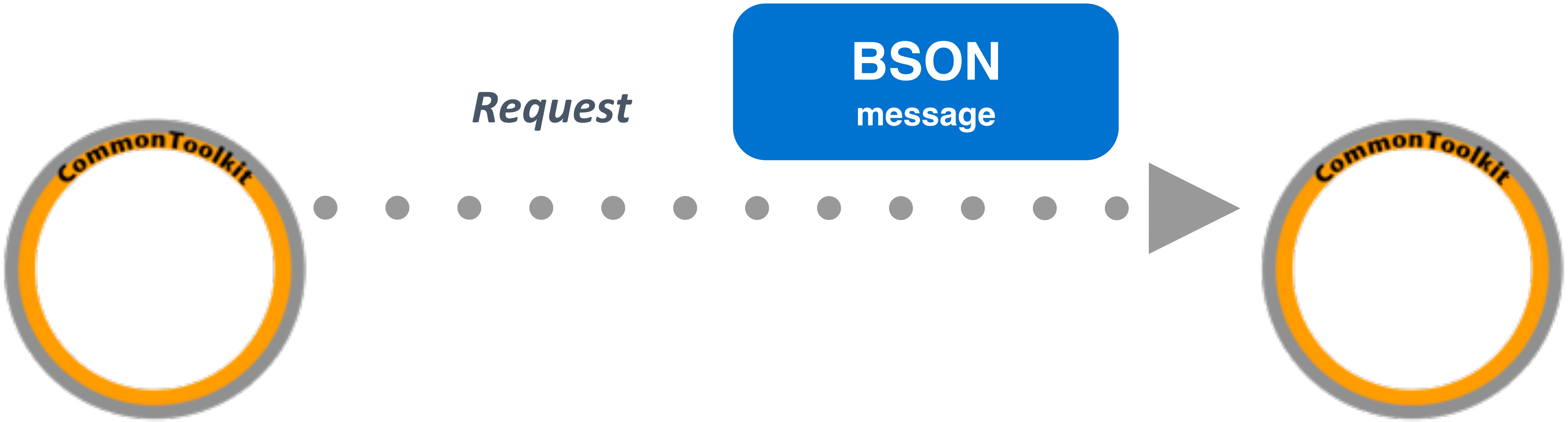
RPC pack

- RPC BSON Pack has some information that CHAOS RPC System use to dispatch the command
    - "domain name" of the command
    - "name" of the command
    - sub-bson object that code the data
    - answer code {used by sender to read the response}
    - sender address{address where send the answer}

RPC System

RPC Message Flow

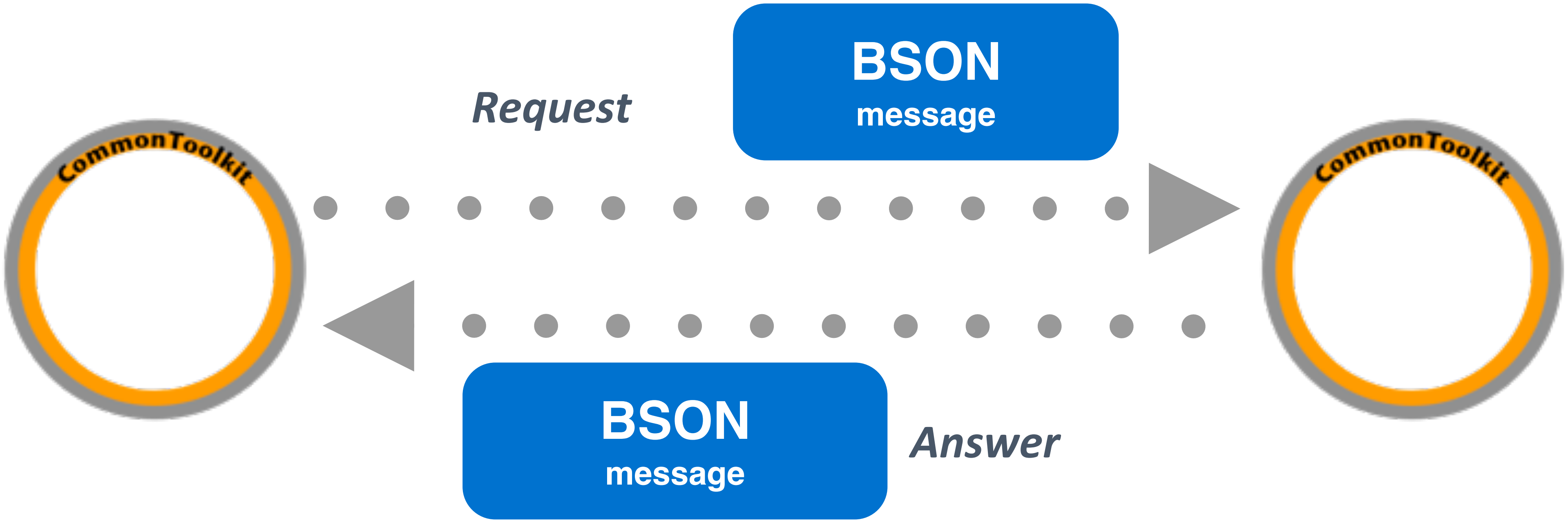
- request message is constructed with BSON, serialized and sent to RPC end point



RPC System

RPC Message Flow

- request message is constructed with BSON, serialized and sent to RPC end point
- if the requester want a response it put answer-code and sender-address in BSON pack,and automatically RPC System wait for the answer



RPC System

- The RPC System is user for:
  - MS <-> CU
    - CU management and retrieval information
    - CU Heartbeat
  - UI <-> CU
    - Set the input attributes of hardware dataset
    - CU management and retrieval information
- CU Management is:
  - init, deinit, start and stop

CUToolkit
CommonToolkit

CUToolkit

CUToolkit and Control Unit

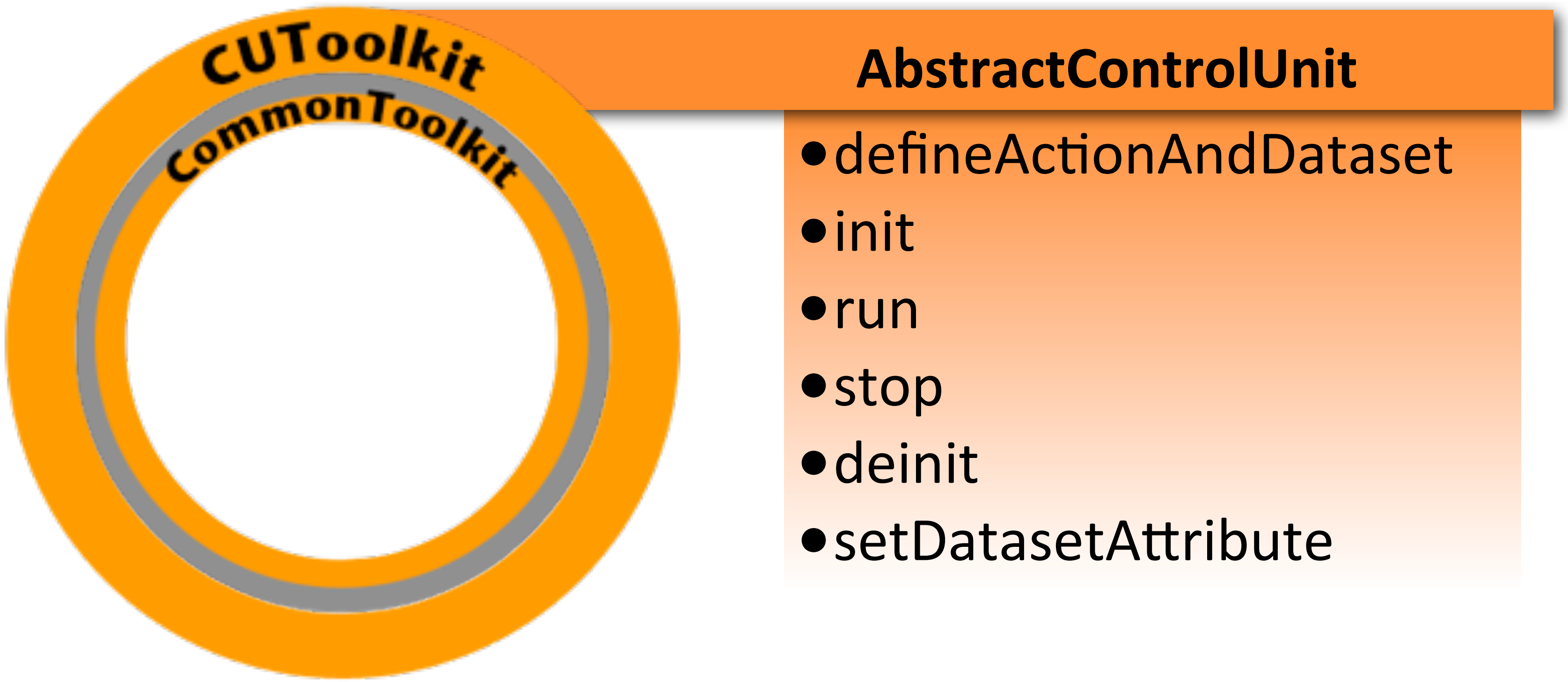
- CUToolkit help the Control Unit development

- Developer need to extend only one class to create a CHAOS driver, the “AbstractControlUnit”

- AbstractControlUnit expose all the APIs needed for interacting with !CHOAS

Hardware controller (CU)

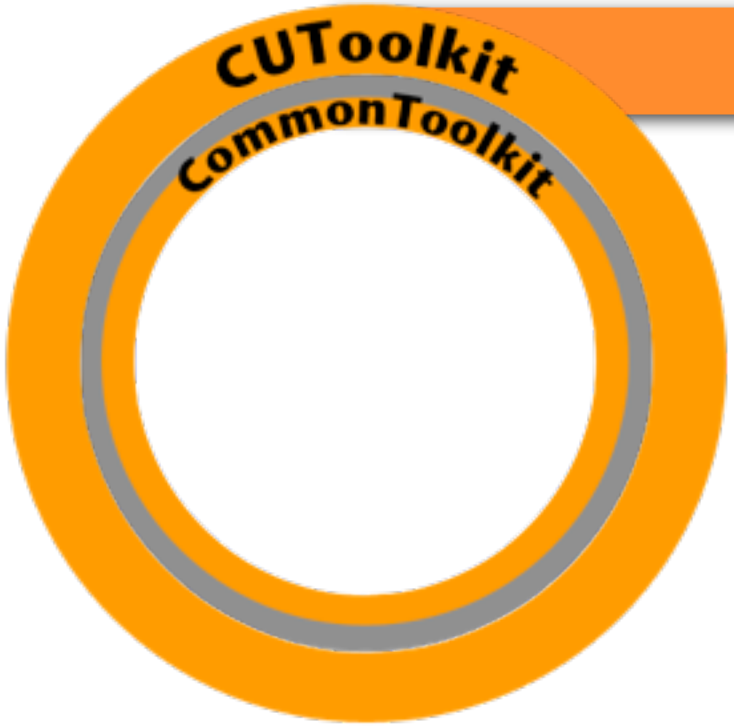
AbstractControlUnit is an abstract cpp class, that force developer to implement some method



AbstractControlUnit

- defineActionAndDataset
- init
- run
- stop
- deinit
- setDatasetAttribute

Hardware controller (CU)



AbstractControlUnit

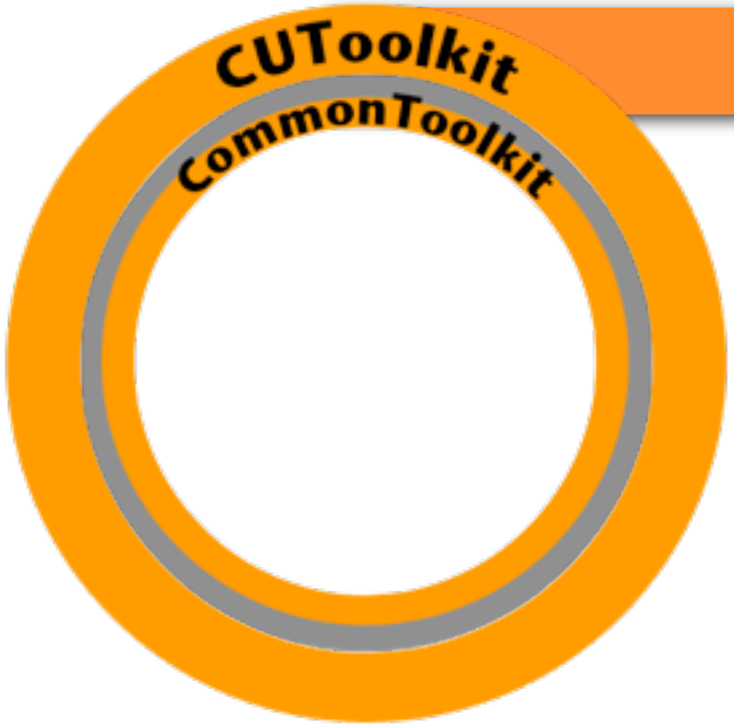
defineActionAndDataset

- permit to define Dataset and/or CU identification, that are sent to MetadataServer

setDatasetAttribute

- receive the value for the input attribute of the Dataset to be passed to the controlled Hardware
  <u>this method is accessible via RPC</u>

Hardware controller (CU)



AbstractControlUnit

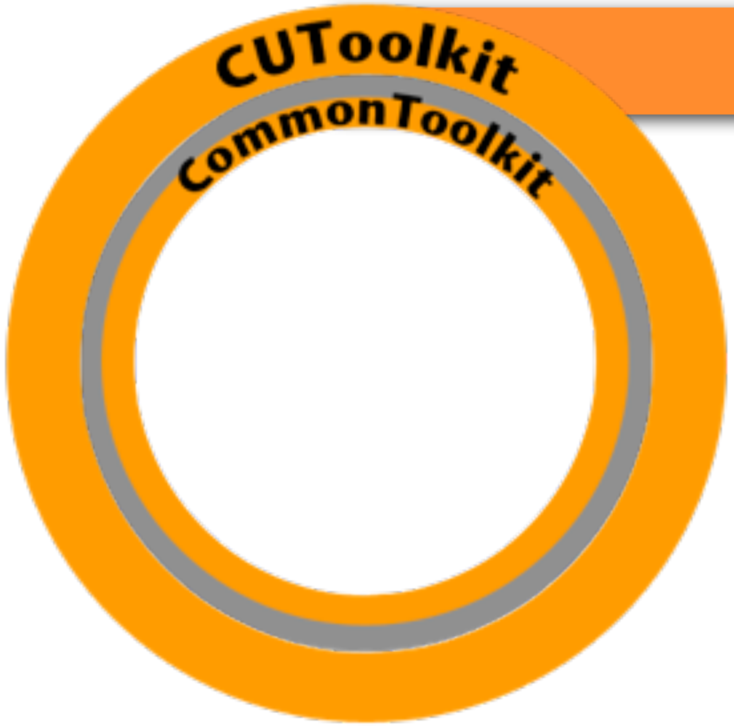
init

- receive from metadata server the dataset with the default value for Input attribute, if the controlled hardware need to be initialize this is the right place where do that

deinit

- this is called when the Control Unit need to be stopped

Hardware controller (CU)



AbstractControlUnit

run

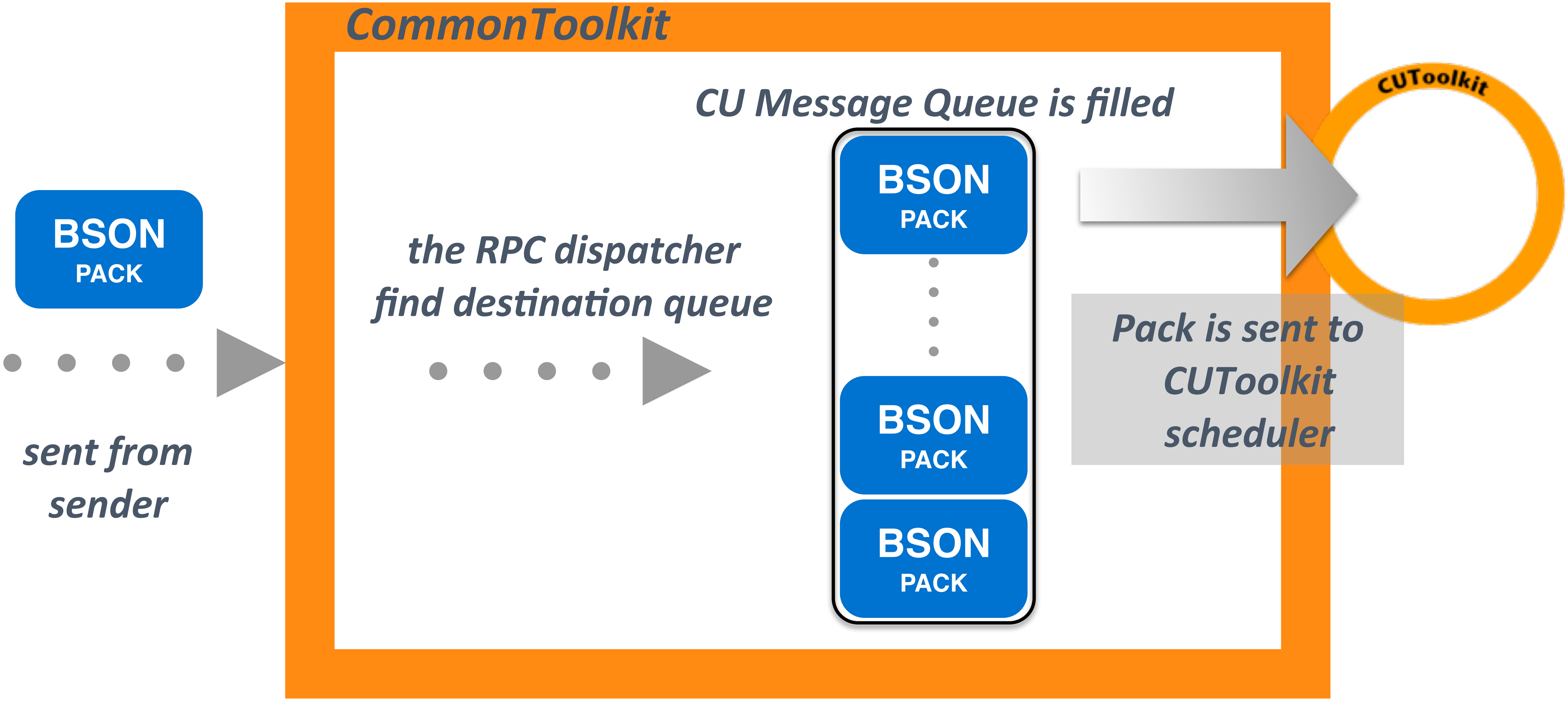
- **thread independent method scheduled with parametrized interval. This is the place where the hardware control and data acquisition is done. Acquired data can be push to live and history data**

stop

- **called before "run" method is paused**

CUToolkit feature

multi-threading messaging dispatcher; every CU has it's own thread for dispatching messages after it has been received by RPC System



CommonToolkit
BSON PACK
sent from sender
the RPC dispatcher find destination queue
CU Message Queue is filled
BSON PACK
BSON PACK
BSON PACK
CUToolkit
Pack is sent to CUToolkit scheduler

either the RPC dispatcher(CommonToolkit) and the CU scheduler of the message are customizable

CUToolkit feature

Management of the attribute priority on set operation

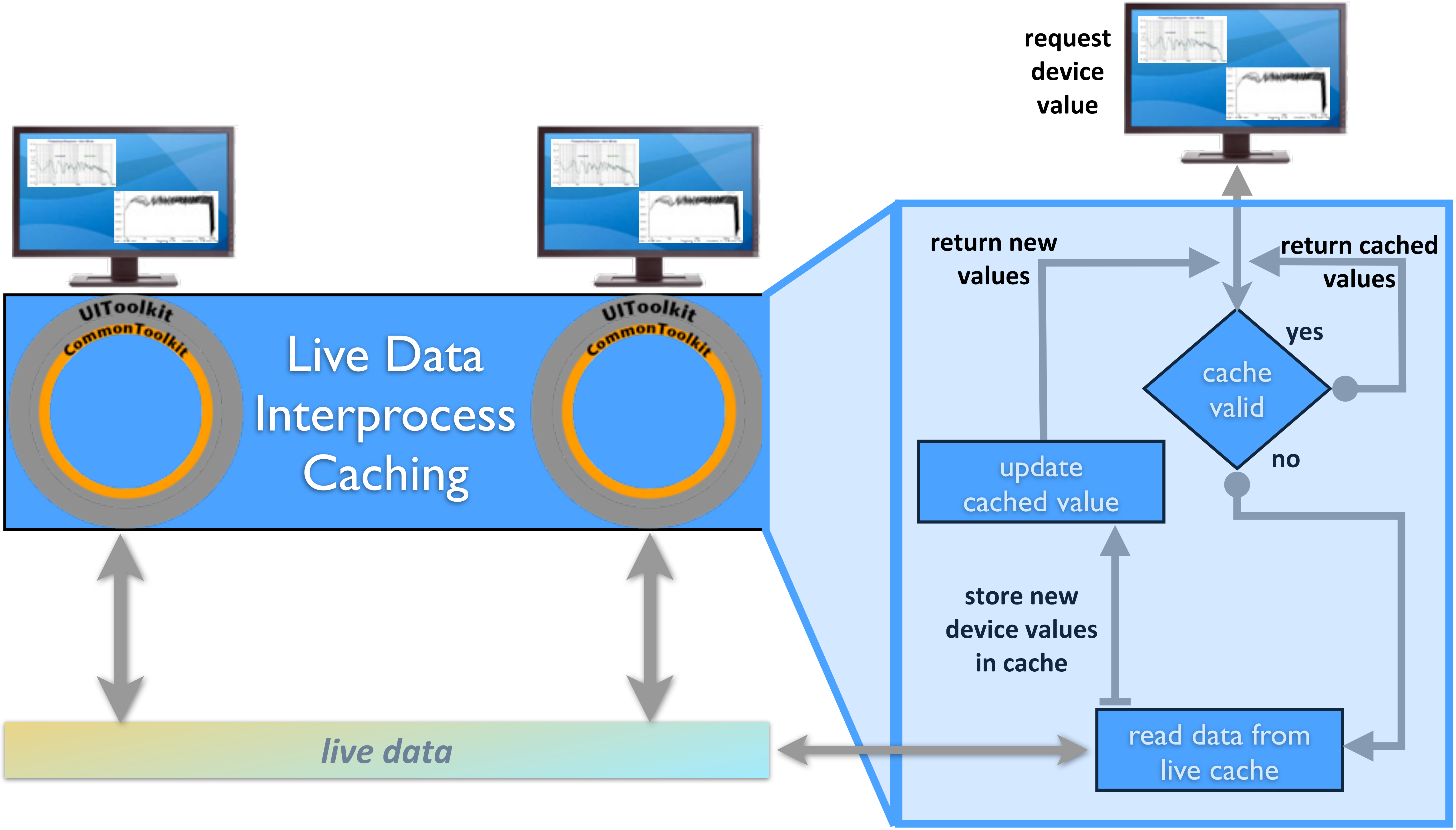
- the “setDatasetElement” RPC message has an embedded customizable queue for regulating the "set" operation of the input hardware attribute of the DATASET
  - for example; take the attributes A1 and A2 and A3
  - if the A2 "setting" operation is “running”, A1 can’t be processed until A2 has finished
  - A3 can be processed in concurrently with A1 and A2
- this will be managed from MetadataServer or statically defined into the ControlUnit

UIToolkit

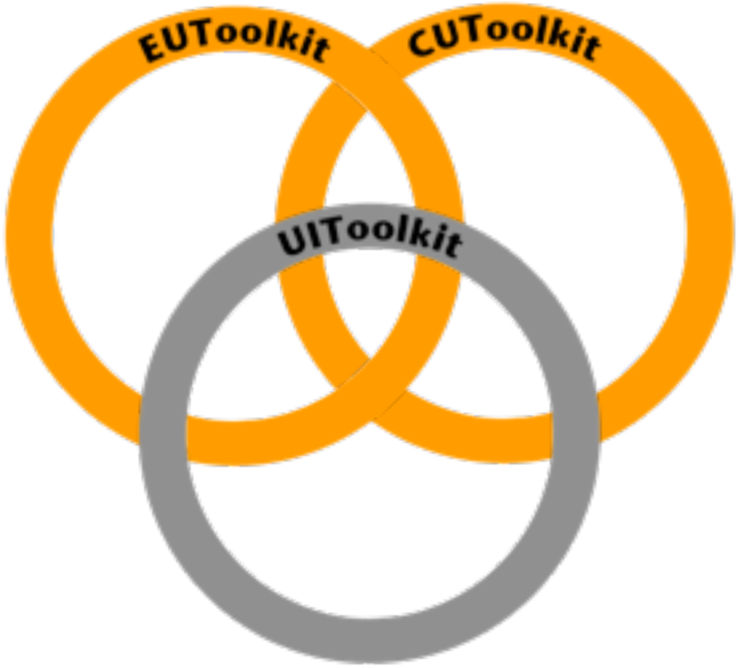
UIToolkit and unified Control GUI

- UIToolkit is the framework layer that permit to developer to create client application that need to access CHAOS resource
- it abstract to application:
    - connection to CU for control a device
    - querying the MetadataServer for retrieve HW information and Dataset
    - caching across UIToolkit process for live data
    - intelligent polling(predict when there will be a new valued on live data storage)
    - Other functionality are in study

UIToolkit and unified Control GUI



!CHAOS



thanks for the time