# EPICS V4 Expands Support to Physics Application, Data Acquisition, and Data Analysis

L. Dalesio, Gabriele Carcassi, Martin Richard Kraimer, Nikolay Malitsky, Guobao Shen, Michael Davidsaver, BNL, Upton, Long Island, New York, U.S.A, Ralph Lange,Bessy, Berlin, Germany, Matej Sekoranja, Cosylab, Ljubljana, Slovenia, James Rowland, Diamond Light Source, Oxfordshire, England, Greg White, SLAC, Menlo Park, California, U.S.A. Timo Korhonen, PSI, Villigen, Switzerland.

ICALREPCS
October 14, 2011

# Outline

- Version 3 recap
- Version 4 to date
- Normative Data Types
- High Level Application Architecture
- Conclusions

# Version 3 – took 6 years to release

- Started at GTA project in 1985 at LANL as a tool set used to develop a space based accelerator ;)
  - Developed core: channel access and process database
  - along with some three display managers: SNL, save/restore, archive, bumpless reboot.
- In 1989 a collaboration started with APS.
  - several other labs showed interest to collaborate.
- EPICS release V3, shows up in 1991
  - process database rewritten to clarify the interface to hardware and new record types.
  - Channel access continues to mature.
- 10% of the original code became part of this release.
  - This project developed three user interfaces to start the tradition

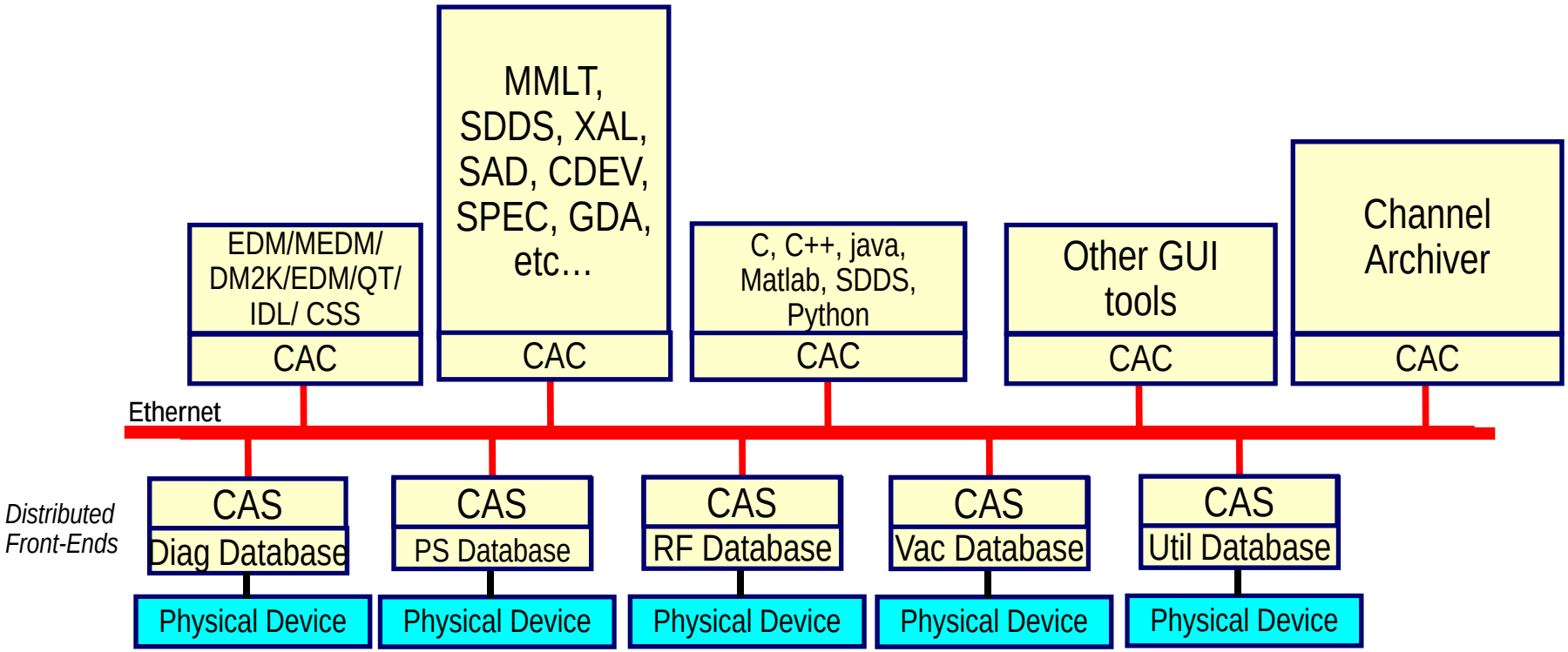
# Version 3 Supports Instrumentation

- Records represented either an input signal, an output signal or an operation to perform on a set of signals
  - Analog input, analog output, (multi-bit)binary input, (multi-bit) binary output, motor, event, PID, calc, etc…..
  - Agreeing on what a device is – is difficult. Is it a power supply or a magnet? Does a motor have an LVDT, an encoder, back lash?
- Records implement continuous control in an autonomous controller to perform DCS functionality.
- Many different types of research and industrial facilities successfully applied this to their plant for equipment control.
- Process Variables (PVs) are available across the network
  - Any field of any record can be a process variable.
  - Functions on PVs are: get, put, monitor
  - This service was designed and implemented to be robust and fast (15K PVs per second to a client on a 100 MB network)
  - Channels always have a time stamp, alarm severity, and alarm status – the simple data type was not useful in most cases
  - Channels have metadata to describe display, control, and alarm information.
- MANY clients were developed on this interface in many languages on many operating systems implementing the full range of SCADA capabilities.
  - With two site developing EPICS, there were two display managers.

# Version 3 Has Limited Support for Devices

- Records did not operate on things more complex than scalar signals.
  - No time domain, no frequency domain, no images.
  - No way to represent things more complicated than scalar signals and 1 dimensional arrays
- Process Variables available across the network could not support everything needed
  - No atomic command / response mechanism
  - No way to ask for a PV subject to parameters.
  - PVs metadata did not always fit properly for every field of a record – such as the display precision – what is the time stamp of this?
  - Typically a get is done on connection for display, alarm limits, and control metadata changes are not reflected.
  - Meta data was sent all of the time, so only time stamp and current alarm information is monitored.
- MANY clients added layers on top of V3 Process Variables to implement more complex data models

# EPICS Version 3 Architecture



# Version 4 is in Year 6 of Development

- Started at Marty's home in 2006 as a tool set used to develop a device based control.
  - Developed core: PVData and Java IOC to support the creation of devices.
- Later in 2006, Cosylab was contracted to develop PVAccess
  - Transport PVData
  - Provide all of the V3 communication along with command/response and others
  - Perform as well as Channel Access V3 on version 3 types.
  - Optimizes the transport by only sending the elements that have changed.
- In 2009 several laboratories showed interest in developing services in this environment
- 
- V4 alpha release shows up in 2011
  - The team is much smaller
- 50% of the original code became part of the release
  - More efficient since we are not developing user interfaces

# Version 4 Supports Complex Data Structures

- Java IOC can represent devices
  - We will likely not implement devices, as it is still difficult to agree on what these are
  - We will use V3 records at NSLS II.
- PVData (PVs) are available across the network
  - Functions on PVs are: get, put, monitor, put/process/get (command/response)
  - It is also hard to create object models on more complex devices such as a telescope or an accelerator.
- Normative types are defined to provide metadata for more complex constructs: multi-channel array, table, N dimensional Array, Image.
  - PVData always has a time stamp, alarm severity, and alarm status
  - Vectors have useful metadata and distinctions: time domain vector, frequency domain vector, histogram
  - Operations can be performed on two PVs with the same normative types.
- PVService supports creating middle layers services
- MANY servers are being developed on this interface to implement middle layer services for accelerator control and data acquisition.

# Specific Normative Data Types

NTMultichannelArray – represent a collection of single values as an ordered array
e.g. all of the temperatures along a beam line or the Australian Synch's Concatenate Record used on AI's

NTTimeDomainArray
e.g. a scope trace from a digitizer or the Circular Buffer in the Compress Record

NTHistogram e.g. information on a 60 Hz power supply RB posted every hour or the Histogram in the Compress Record

NTNDArray e.g. multiple frames of a detector taken at 1 KHz for 1 second

NTFrequencyDomainArray
e.g. FFT of 10 KHz data taken for 1 second to study noise frequencies or FFT record output

NTStatistic e.g. any data being compressed from its original rate – fast sampling in hardware to EPICS DB,
new function on any database waveform, response from a request to an archive server

NTImage e.g. image data being collected at a detector into the areaDetactor application

NTTable e.g. a way to return any list of values or collection of name, value pairs of different data type such as twiss parameters
or the metadata for a camera set up: filter, exposure time, camera used, etc…
This is the catch all data type that can define a structure of single values or arrays (of the same length)

NTChannelFinderDirectory
e.g. returned as an ordered list of PVs from a query to a directory service to populate a multi-channel array or table

# SLAC RDB Server Done Last Night

Example : Get modelled Twiss parameters (and other stuff) of all devices, sorted by Z

```
greg% getRdb modelTwiss:Extant.FullMachine
structure ResultSet
  string normativeType NTTable
  double[] ORDINAL [0.00000,1.00000,2.00000,7.00000,9.00000,...]
  string[] ELEMENT_NAME [CATHODE,DBMARK80,BEGIN_OF_CATHODE TO BXG,SOL1,SOL1,...]
  string[] EPICS_CHANNEL_ACCESS_NAME [CATH:IN20:111, , ,SOLN:IN20:121,SOLN:IN20:121,XCOR:IN20:121...]
  double[] Z_POSITION [2014.70,2014.70,2014.70,2014.80,2014.90,...]
  string[] POSITION_INDEX [BEGIN,BEGIN,BEGIN,BEGIN,MIDDLE,...]
  double[] LEFF [0.00000,0.00000,0.00000,0.200000,0.200000, ...]
  double[] TOTAL_ENERGY [0.00600000,0.00600000,0.00600000,0.00600000,0.00600000,0.00600000,...]
  double[] PSI_X [0.00000,0.00000,0.00000,0.00445946,0.00949753,...]
  double[] BETA_X [22.3858,22.3858,22.3858,20.7061,19.0272,...]
  double[] ALPHA_X [8.91993,8.91993,8.91993,8.57440,8.21451,...]
  double[] ETA_X [0.00000,0.00000,0.00000,0.00000,0.00000,0.00000,...]
  double[] ETAP_X [0.00000,0.00000,0.00000,0.00000,0.00000,...]
  double[] PSI_Y [0.00000,0.00000,0.00000,0.0252115,0.0547781,0...]
  double[] BETA_Y [4.03080,4.03080,4.03080,3.59863,3.17969,...]
  double[] ALPHA_Y [2.32706,2.32706,2.32706,2.17425,2.01510,...]
  double[] ETA_Y [0.00000,0.00000,0.00000,0.00000,0.00000,... ]
  double[] ETAP_Y [0.00000,0.00000,0.00000,0.00000,0.00000,...]
```

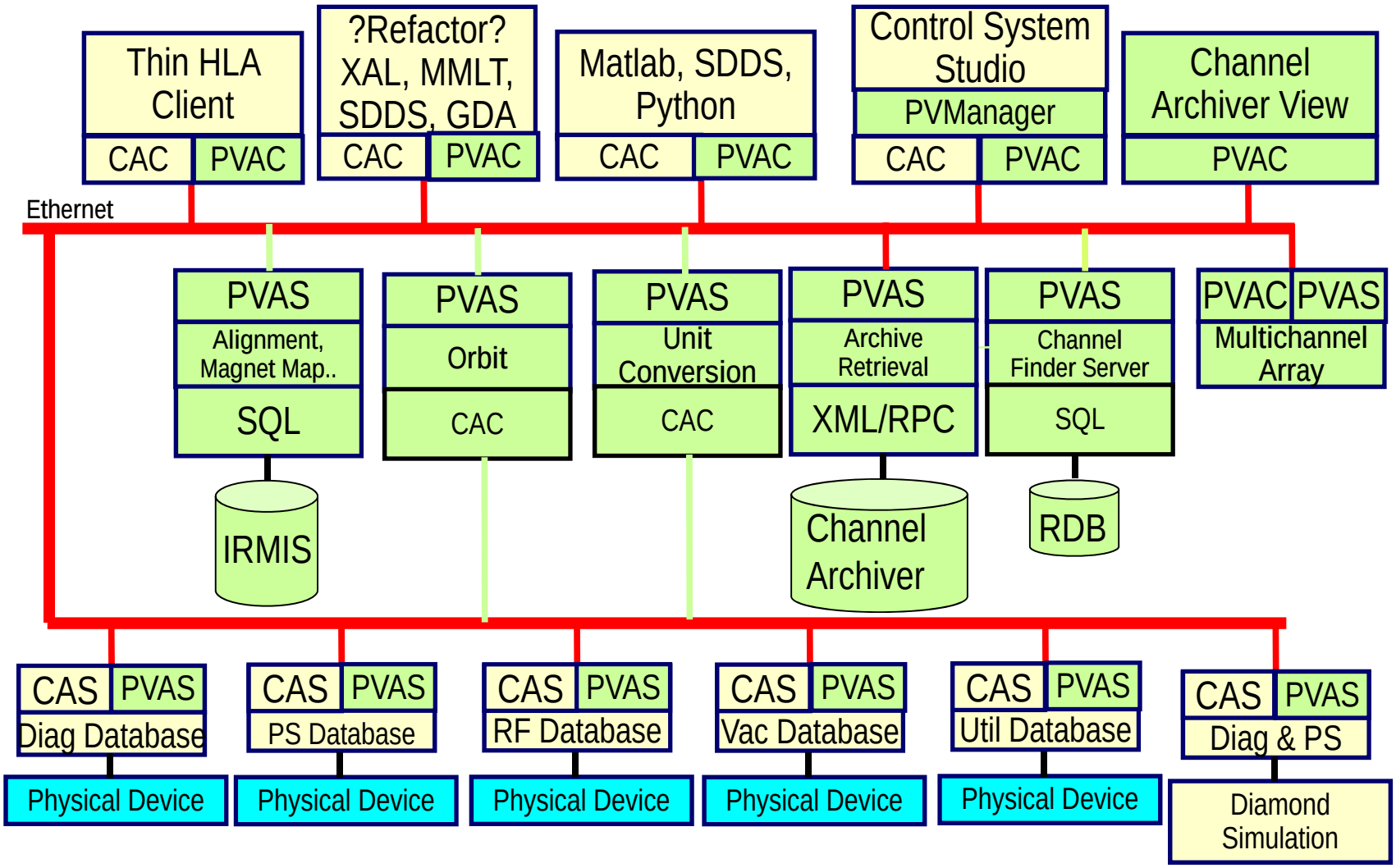
getRdb (a trivial bash script), calls the rdbClient side
which calls rdbService (based on pvService).

rdbService's rdbServiceFactory looks up the SQL SELECT statement equivalent to the argument it was given in a table, and then executes that SELECT statement. It uses JDBC to get the data out of Oracle. The JDBC ResultSet is encoded as pvData and sent back to the client, and what you see is simply what you get on the client when you toString() the returned PVStructure.

# Too Early to Identify Version 4 Limitations

- PVData does not preclude people from defining any arbitrary data strcuture.
- PVAccess already supports the transportation of these non-normative types
- We are not developing clients to support non-normative types.
- Donald Rumsfeld --- “There’s your knowns, your known unknowns, and your unknown  unknowns”……

# Create a V4 Archive Server



# Conclusions

- The interface is intended to allow us to create a standard client/server architecture for high level applications such as: areaDetector, Matlab Middle Layer Toolkit, SAD, SDDS, XAL, GDA, MDS+
- NSLSII is committed to apply this technology to physics applications, and is actively evaluating it for application in experiment control and data acquisition.
- Low level applications are not yet being developed in Java IOC.
- New structures are easy to create – but we plan to carefully limit these to general and useful normative types.
- We are in the stage of development most similar to the transition from GTACS to EPICS: early, immature, risky, changeable, challenging, and fun.