Writing EPICS Channel Access Clients in Python

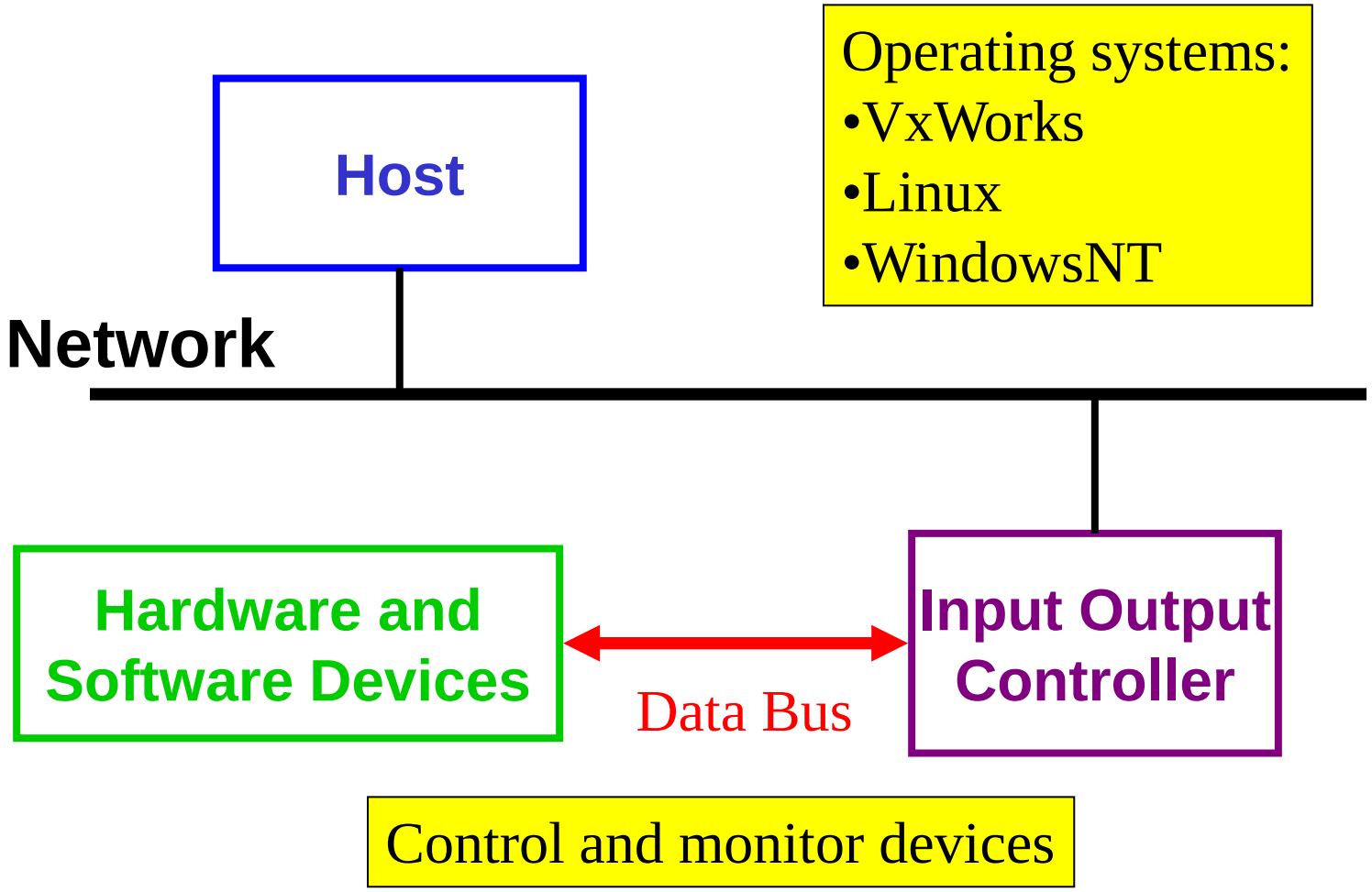
Geoff Savage

March 2003

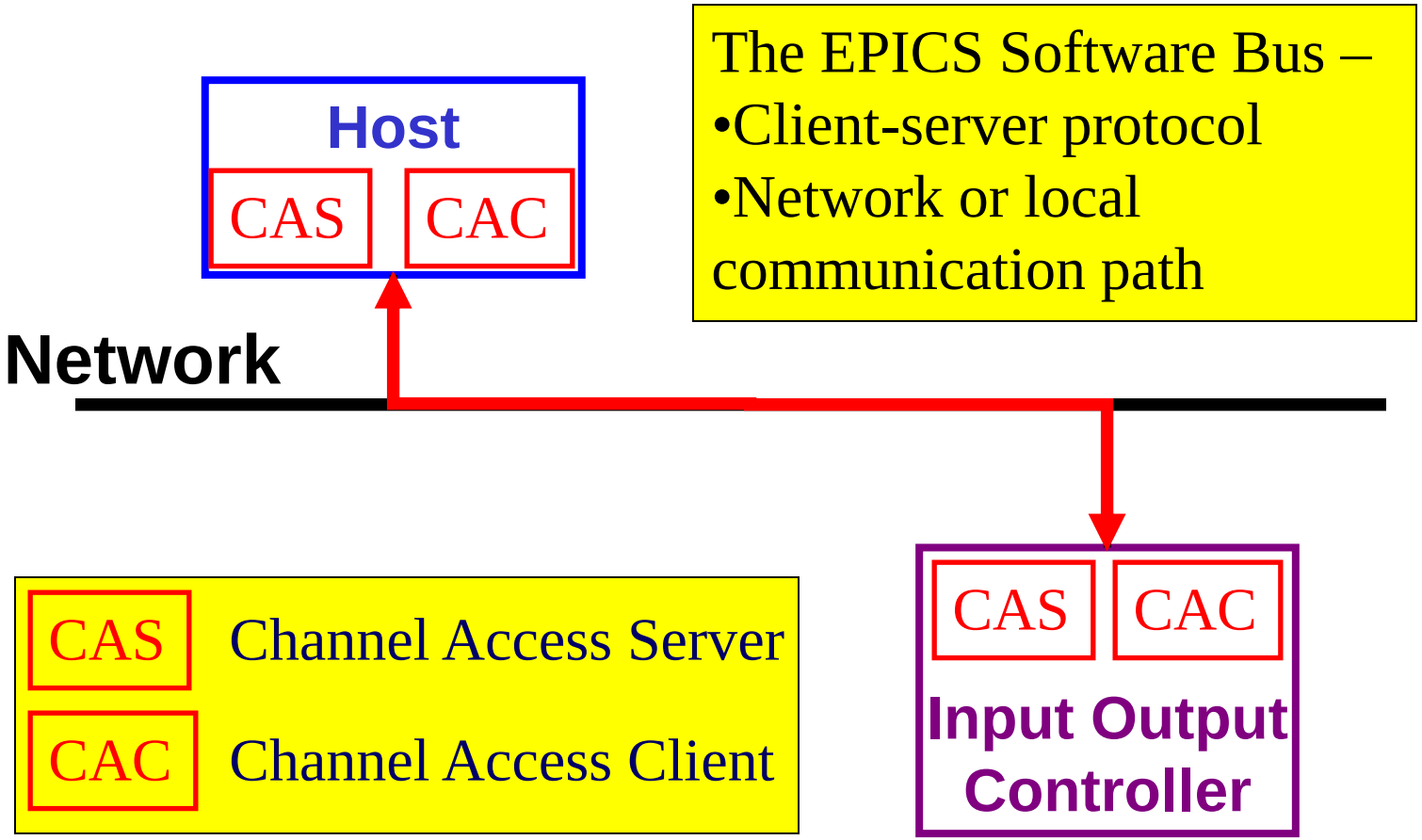
Contents

- EPICS concepts
  - EPICS architecture
  - Channel access (CA)
- EPICS CA and Python
  - Motivation
  - Examples
  - Software architecture
  - CaChannel
  - caPython

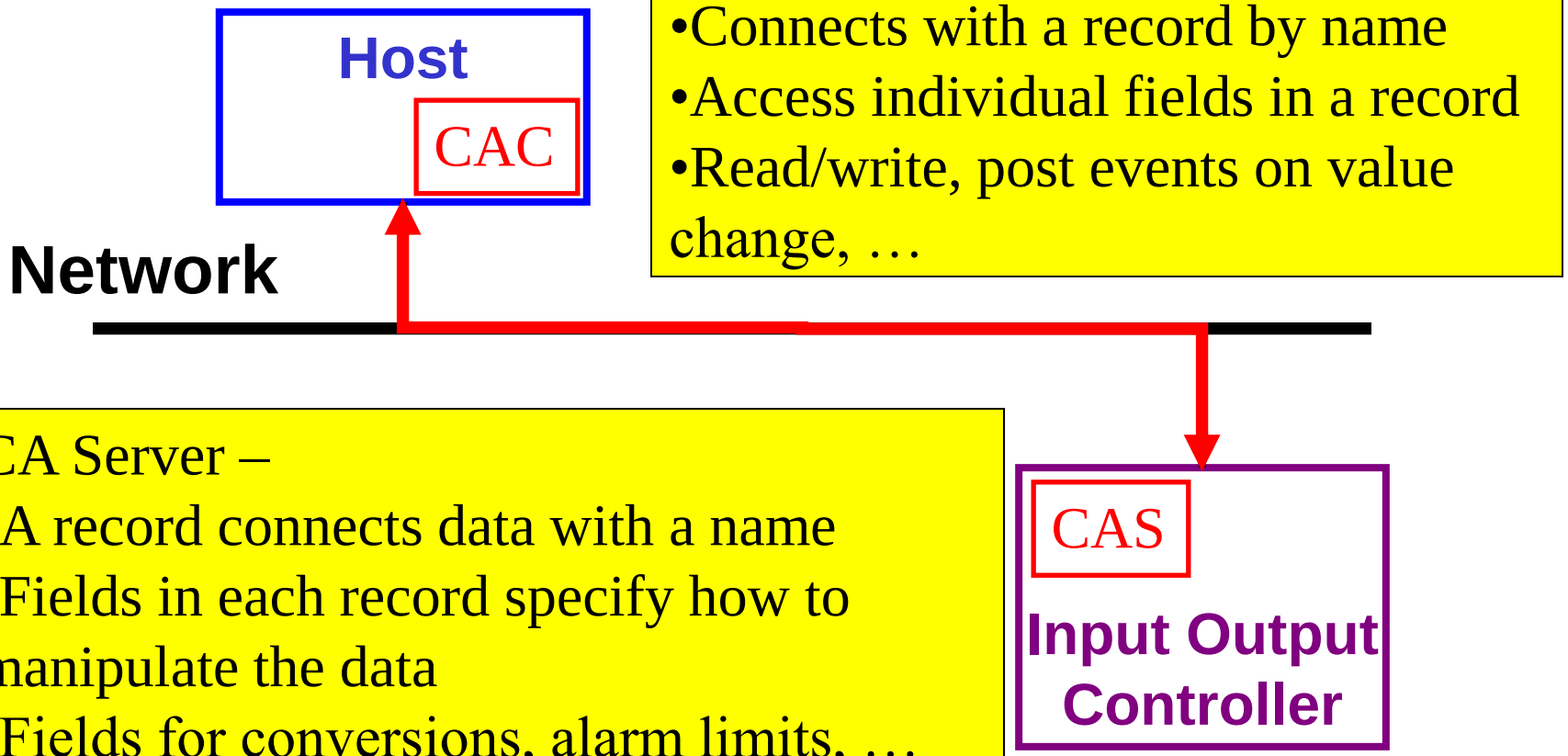
EPICS Architecture



Channel Access



CA Communication



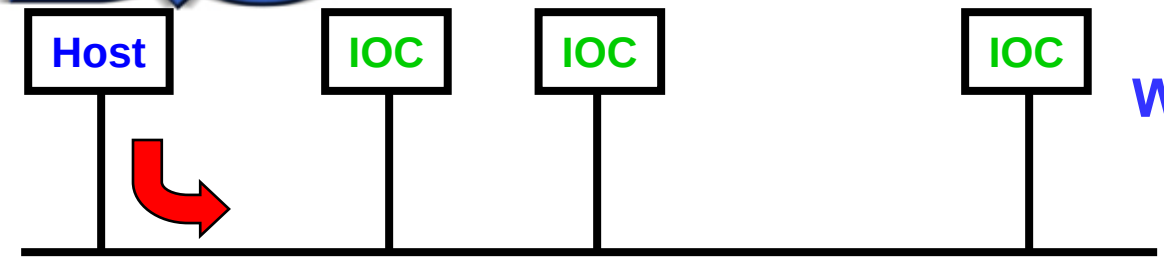
CA Client –
- Connects with a record by name
- Access individual fields in a record
- Read/write, post events on value change, …

CA Server –
- A record connects data with a name
- Fields in each record specify how to manipulate the data
- Fields for conversions, alarm limits, …

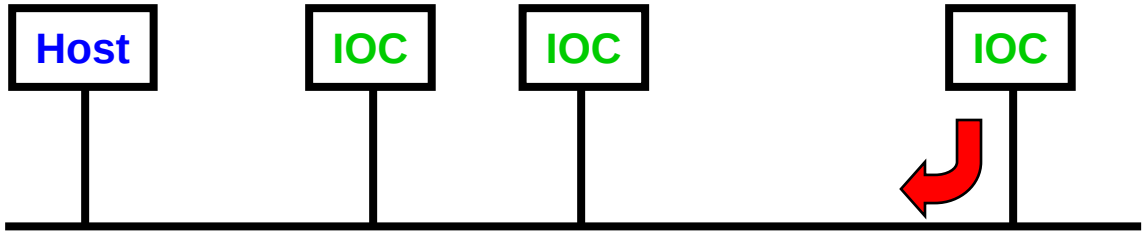
CA Services

- Dynamic channel creation
- Automatic channel reconnect
- Read
- Write
- Monitoring
  - Generates callbacks
  - Events
  - Access control
  - Connection
- Data type conversions
- Composite data structures

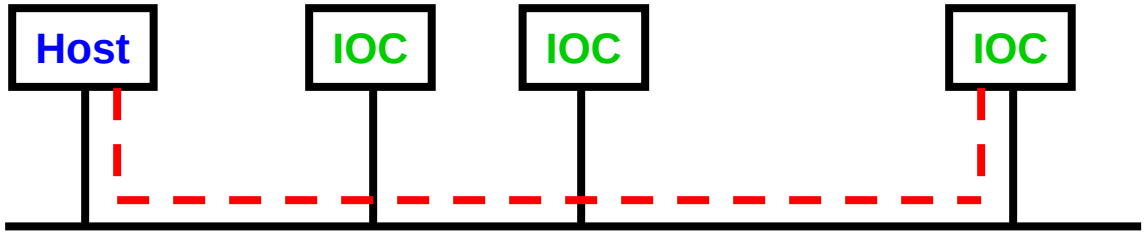
Channel Creation



Who has channel 'XXXX'?
(udp broadcast)



I have channel 'XXXX'.



Negotiate connection.
(point-to-point tcp)

CA Buffering

- "All requests which require interaction with a CA server are accumulated (buffered) and not forwarded to the IOC until one of *ca_flush_io()*, *ca_pend_io()*, *ca_pend_event()*, *ca_sg_pend()* are called allowing several operation to be efficiently sent over the network in one message."
- Read/write transactions can not be in the same message as the connection request

CA Request Status

- “If the server for a channel is located in a different node than the client then the operations that communicate with the server return status indicating the validity of the request.”
- “In other words, success returned from the client library routine indicates that it checked your request, it appeared to be valid, and it forwarded it to the server.”
- Check the status when the transactions have completed

CA Transaction Status

- "Any process variable values written into your program's variables by *ca_get()* should not be referenced by your program until ECA_NORMAL has been receied from *ca_pend_io()*."
- Upon receipt of a callback the transaction is complete and the transaction status is available from within the callback.

CA Operation

- "For proper operation CA must periodically be allowed to take care of background activity. This requires that your application must either wait in one of *ca_pend_event()*, *ca_pend_io()*, or *ca_sg_block()*, or it must call *ca_pend_event()* at least every 15 seconds."

EPICS CA Summary

- EPICS software bus
- Client-server architecture
- Library functionality
  - Create and maintain connections
  - Synchronous transactions
  - Asynchronous transactions
  - Transaction status reporting
  - Buffering for efficiency

Details follow in the discussion of CaChannel.

Why Python?

- "Python is an *interpreted, interactive, object-oriented* programming language"
- Fermilab experience
  - Reduced development time
  - Portable (Linux, NT, OSF1)
  - Graphics (Tcl/Tk)
  - Powerful extension library
  - Easy to extend (SWIG)
  - Short learning curve
- www.python.org

Using CA from Python

- Maintain full CA functionality
  - Callbacks
  - Data types
- Map each connection to an object
- Two step process
  - caPython – make the CA library callable from Python (wrapping)
  - CaChannel – use the wrapped functions in caPython to create channel objects

Simple Interpreter Example

```
[savage@d0ol30 ~]$ setup d0online
[savage@d0ol30 ~]$ python
Python 2.1 (#3, May 16 2001, 15:15:15)
[GCC 2.95.2 19991024 (release)] on linux2
Type "copyright", "credits" or "license" for more information.
>>> from CaChannel import *
>>> ch = CaChannel('catest')
>>> ch.searchw()
>>> ch.putw(123.456)
>>> ch.getw()
123.456
```

Make a connection via a name.

Create the connection.

Write a value.

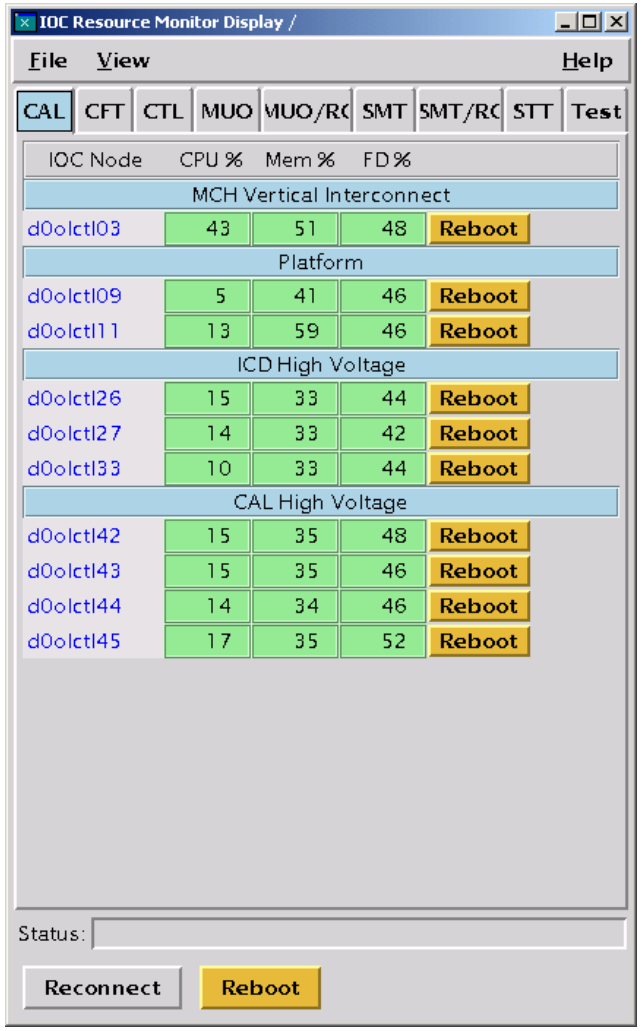
Read a value.

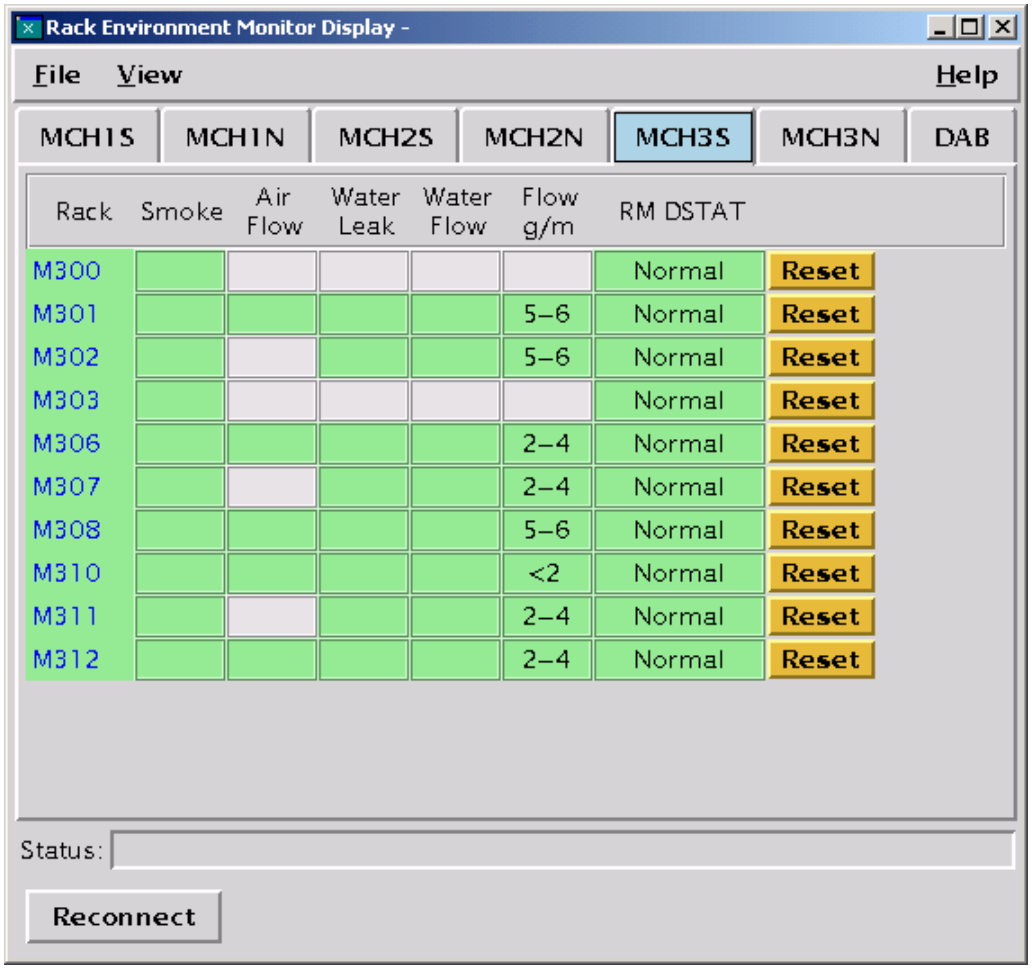
Simple Script Example

```
#! /bin/env python
from CaChannel import *
def main():
    try:
        catest = CaChannel('catest')
        catest.searchw()
        catest.putw(123.456)
        print catest.getw()
    except CaChannelException, status:
        print ca.message(status)
main()
```

Print the error string.

Graphical Interfaces





Architecture

Move from C functions to a Python class interface.

CaChannel
- **Python class**
- **Calls caPython functions**

caPython
- **Python wrapper around the C library**
- **Collection of python functions**

EPICS channel access C library
- **Collection of C functions and macros**
- **Distributed with EPICS**

CaChannel

- Methods
- Data types
- Synchronous I/O
- Asynchronous I/O
- Monitoring
- Use with Tkinter

CaChannel Construction

- CaChannel is constructed from caPython and the SWIG pointer library
- “SWIG is an interface compiler that connects programs written in C and C++ with scripting languages such as Perl, Python, Ruby, and Tcl.”
- caPython – CA functions, macros, and helper functions wrapped using SWIG
- SWIG pointer library – manipulation of C pointers from Python

CaChannel Methods …

- Connection
  - search
  - search_and_connect
  - clear_channel
- Synchronization
  - pend_io
  - pend_event
  - poll
  - flush_io
- Write
  - array_put
  - array_put_callback
- Read
  - array_get
  - array_get_callback
  - getValue
- Monitors
  - add_masked_array_event
  - clear_event

… CaChannel Methods

- Simple and slow
  - searchw = search + pend_io
  - getw = array_get + pend_io
  - putw = array_put + pend_io
- Information
  - field_type
  - element_count
  - name
  - state
  - host_name
  - read_access
  - write_access

Synchronous I/O

```
>>>
>>> ch.search()
>>> ch.pend_io()
>>>
>>> ch.array_put(123.456)
>>> ch.pend_io()
>>>
>>> ch.array_get()
>>> ch.pend_io()
>>> ch.getValue()
123.456
```

- pend_io = flush request buffer and wait for CA requests to complete for all CaChannel objects
- Multiple requests can be issued before each pend_io
- All connections must be made before get or put
- On a get a value is not valid until pend_io has returned with no errors

Data Types …

```
>>> ch.getw()
123.456
>>> ch.getw(ca.DBR_INT)
123
>>> ch.getw(ca.DBR_STRING)
'123'
>>> ch.field_type()
6
>>>ca.dbr_text(ch.field_type())
'DBR_DOUBLE‘
>>>
```

- Each PV has a native type
- Different data types can be requested
- Requests using the native type are most efficient

Not all caPython functions are implemented as methods of CaChannel.

… Data Types …

```
>>> scan =
    CaChannel('catest.SCAN')
>>> scan.searchw()
>>> scan.getw()
6
>>> scan.getw(ca.DBR_STRING)
'1 second'
>>> scan.putw(5)
>>> scan.getw(ca.DBR_STRING)
'2 second‘
>>> ca.dbr_text(scan.field_type())
'DBR_ENUM'
>>>
```

- The SCAN field of a record identifies how often the record is processed
- There is an enumerated list of possible SCAN field values

… Data Types

```
>>> scan.putw('1 second')
Traceback (innermost last):
  File "<stdin>", line 1, in ?
  File "CaChannel.py", line 383, in putw
    count, pval = self.__setup_put(value, req_type)
  File "CaChannel.py", line 131, in __setup_put
    CaChannel.dbr_d[req_type]['convert'](value),
ValueError: invalid literal for int(): 1 second
>>> scan.putw('1 second', ca.DBR_STRING)
>>> scan.getw(ca.DBR_STRING)
'1 second'
```

Writing the enumerated identifier (5) works on the previous slide and writing a value (‘1 second’) fails.

Request Data Types

Request type	Python type	Comments
DBR_STRING	String	
DBR_ENUM	Integer	Enumerated
DBR_CHAR	Integer	8 bits
DBR_INT	Integer	16 bits
DBR_LONG	Integer	32 bits
DBR_FLOAT	Float	
DBR_DOUBLE	Float	
Array of DBR	List of type	

Asynchronous I/O …

- No waiting for CA requests to complete
- A user written callback function is executed when the request has completed
- ca_pend_event = flush the request buffer and wait for timeout or until all CA background activity is processed
- ca_poll = ca_pend_event with a short timeout
- ca_flush_io = flush the request buffer

… Asynchronous I/O

- Asynchronous callbacks do not preempt the main thread
- Instead ca_pend_io, ca_pend_event, or ca_poll must be called at least every 15 seconds to allow background activity to process

Connection Callback

```
ch.search_and_connect('catest', connectCb)
while 1: ch.pend_event()

def connectCb(epics_args, user_args):
  print ‘name =‘, ca.name(epics_args[0])
  print ‘state =‘, epics_args[1]
```

- epics_args = 2 element tuple
  - epics_args[0] = channel identifier, used to get the channels name
  - epics_args[1] = connection state
    - ca.OP_CONN_UP
    - ca.OP_CONN_DOWN
- user_args = tuple containing all python objects specified after the callback in the method

Put Callback

```
ch.array_put_callback(3.3, None, None, putCb)
while 1: ch.pend_event()

def putCb(epics_args, user_args):
    print ca.name(epics_args['chid'])
    print ca.dbr_text(epics_args['type'])
    print epics_args['count']
    print ca.message(epics_args['status'])
    print user_args
```

Default element count

Native data type

- epics_args is a dictionary with keys:
  - chid = channel identifier
  - type = request type (DBR_XXXX)
  - count = element count
  - status = transaction status code from server

Get Callback

```
ch.array_get_callback(None, None, getCb1)
while 1: ch.pend_event()

def getCb1(epics_args, user_args):
    print "pvName = ", ca.name(epics_args['chid'])
    print "type = ", ca.dbr_text(epics_args['type'])
    print "count = ", epics_args['count']
    print "status = ", ca.message(epics_args['status'])
    print "user args = ", user_args
    print "value(s) = ", epics_args['pv_value']
```

Default element count

Native data type

- value(s) = data returned by the server. Multiple data elements are returned in a tuple.

Compound Data Types

- Only supported in asynchronous mode
- Extra information with the value
  - Status – alarm values
  - Time – status + timestamp
  - Graphics – status + alarm limits + display limits
  - Control – graphics + control limits
- Routines are provided to convert DBR type to compound type

Get Status Callback

```
ch.array_get_callback(ca.dbf_type_to_DBR_STS(ch.field_type()), None, getCb2)
while 1: ch.pend_event()

def getCb2(epics_args, user_args):
    print "pvName = ", ca.name(epics_args['chid'])
    print "type = ", ca.dbr_text(epics_args['type'])
    print "count = ", epics_args['count']
    print "status = ", ca.message(epics_args['status'])
    print "user args = ", user_args
    print "value(s) = ", epics_args['pv_value']
    print ca.alarmSeverityString(epics_args['pv_severity'])
    print ca.alarmStatusString(epics_args['pv_status'])
```

Default element count

- pv_severity = alarm severity
- pv_status = alarm status

Monitoring …

- Asynchronous I/O originated at the server
- Client requests notification from the server when
  - the channel's value changes by more than the value dead band or alarm dead band
  - the channel's alarm state changes
- Monitor callbacks match those described under asynchronous I/O

… Monitoring …

```
def eventCb(epics_args, user_args):
    print ca.message(epics_args['status'])
    print "new value = ", epics_args['pv_value']
    print ca.alarmSeverityString(epics_args['pv_severity'])
    print ca.alarmStatusString(epics_args['pv_status'])

try:
        chan = CaChannel()
        chan.searchw('catest')
        chan.add_masked_array_event(
                ca.dbf_type_to_DBR_STS(chan.field_type()),
                None, ca.DBE_VALUE | ca.DBE_ALARM, eventCb)
except CaChannelException, status:
        print ca.message(status)
```

… Monitoring

Monitor mask	Notification condition
ca.DBE_VALUE	when the channel's value changes by more than MDEL
ca.DBE_LOG	when the channel's value changes by more than ADEL
ca.DBE_ALARM	when the channel's alarm state changes

Functions Not Implemented

- ca_change_connection_event
  - use ca_search_and_connect
- ca_add_exception_event
- ca_replace_printf_handler
- ca_replace_access_rights_event
- ca_puser macro
  - not used
- ca_test_event
- Scan groups

Tkinter and CaChannel

- EPICS CA is not thread safe
- All CaChannel activity must be performed in the main thread
- Use the Tkinter after method to interrupt the mainloop at regular intervals to allow CA backgroud activity to execute
- Execute CaChannel calls from the update function called by after

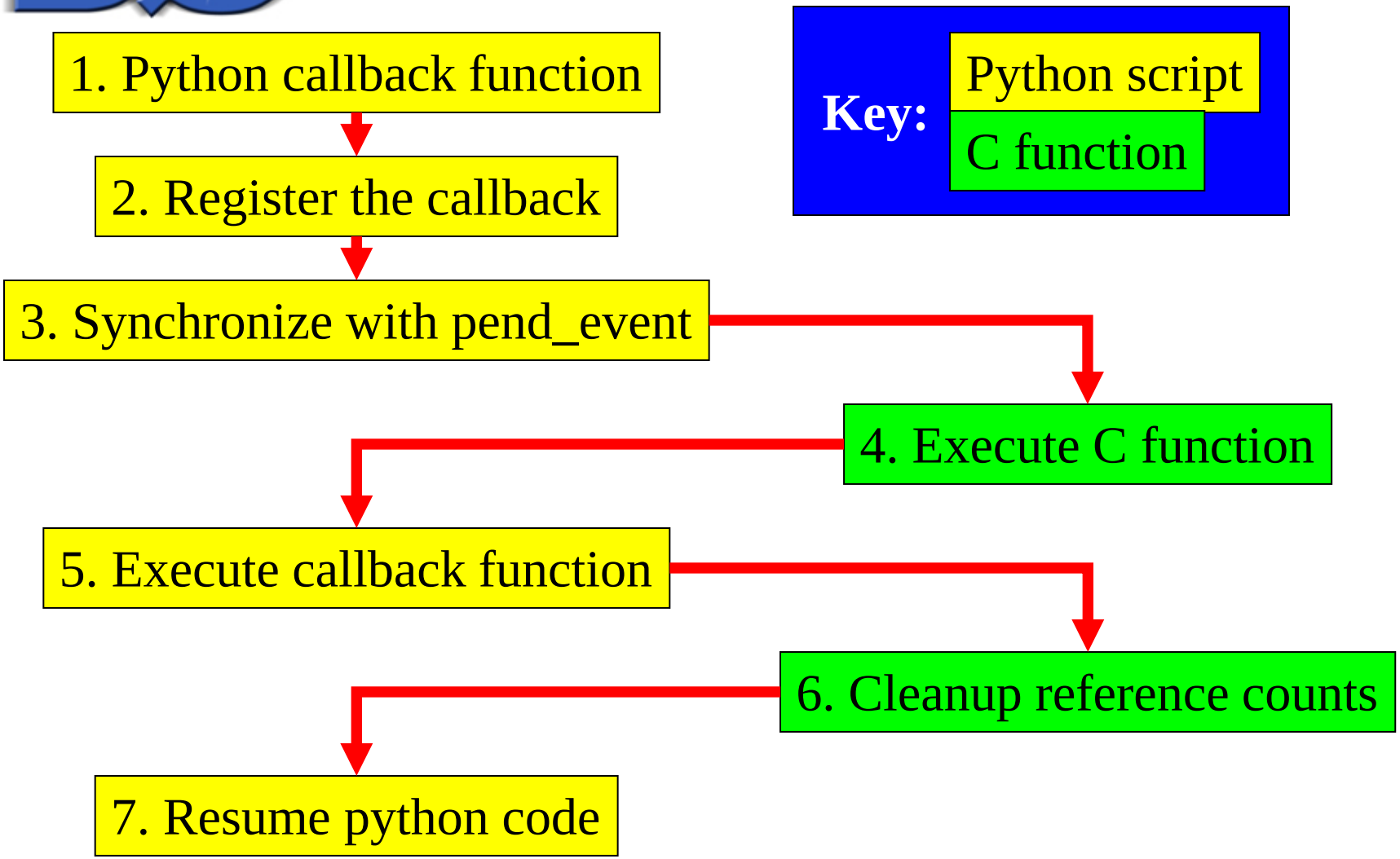
caPython

- Maintain full CA functionality
- SWIG handles the wrapping of functions not associated with callbacks without intervention
  - Label the functions with the C function name minus the ca_ prefix
  - ca_pend_io -> pend_io
- Additional C functions were added to handle callbacks
- Access to constants in the header files
- Macros are wrapped in functions

caPython Initialization

- Provide access to constants in header files
  - DBF_XXXX types – native field types
  - DBR_XXXX types – transaction request types
  - ECA_XXXX – transaction status codes
  - Connection states
  - Event masks
- When the ca module is imported into python execute *ca_task_initialize()*

caPython Callbacks



Callback Example

```
int ca_search_and_connect(
    char *CHANNEL_NAME,
    chid *PCHID,
    void (*USERFUNC)(struct connection_handler_args ARGS),
    void *PUSER);
```

- When ca_search_and_connect() completes it calls USERFUNC with one argument of type struct connection_handler_args
- connectCallback is the C function always called
- The python callback function and user data are combined and passed in through PUSER

Callback Functions

- Convert the channel access C data for return to the Python callback function
- Separate the python callback function and user data found in PUSER
- Call the Python function with two arguments
  - CA data
  - User data - if no user data is specified then the user data tuple is empty
- Cleanup reference counts