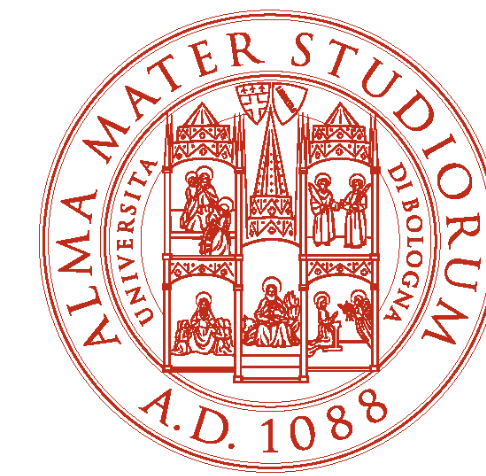


DAQ: slow control

Silvia Biondi, Mauro Villa, Riccardo Ridolfi
University & INFN of Bologna

DAQ Meeting - 17.04.20

DAQ structure

PC
CONTROL DAQ
steering the acquisition process

VME
steering the acquisition process + reading of electronic boards

PC
Readout Module
for non-integrated sub-detectors

PC STORAGE
receiving via fiber or ethernet
and storing data

N PCs
monitoring sub-detectors

PC MONITOR
general monitoring process

EXTERNAL DISK

data

Wave Dream

SoC

det 1 | det 2 | det 3 | ... | ...

Experiment

Data flow - connection

connections going from the same client to the same server

Readout module = CLIENT

1. send **configuration** parameters to server
2. request **monitoring parameters** from server
3. tell the server **when to send data**
4. put the data in the relative **DataChannel** through parallel threads

"Slow" connection

starts when "Configure" mode and uses > 1 thread to handle different simultaneous activities

"Fast" connection

only if the system goes in "RUN" mode,
stops when "stopDC" mode

Sub-detector = SERVER

1. receive **configuration** from client
2. send **monitoring parameter** to client
3. set a device parameter to "run" status and **prepare itself for the data flux**
4. **send the data** to the client

Slow control - Slow connection

Online monitoring - tools

- GNAM
  - need to add all the info from different detectors;
  - possibility to check directly comparisons between different systems.
- Online Histograms (OH)
  - some already implemented, with very basics parameters to check;
  - for now, event size (for all the integrated modules) and time measurement and hit channels for BM.
- Information Service (IS)
  - widely used so far to check detector-specific parameters during the run;
  - each Readout Module can set the frequency of parameters updating.
- DataBase (DB)
  - used to check some interesting parameters at the end of the run;
  - for now only WaveDream readout module uses this tool.
- All this info can be **used by shifters during data taking**
  - not to overload the DAQ shifters too much (from GSI experience);
  - all detector expert should be "shifters" to check their own info.

The Publish method

- **IS and DB make use of the Publish method:**
  - called in **configure**, **unconfigure** and **prepareForRun** states of the machine.

- **Step by step procedure:**
  1. DAQ should know from the very start the set of parameters for each Readout Modules;
  2. it sends a vector of parameters to the Readout Module;

DAQ software: dedicated classes for all the modules

  3. the Readout Module receives the vector and fill it with the values;
  4. it sends the filled vector back to the DAQ;

Detector software: to be implemented in the software interface between detector and DAQ

  5. DAQ propagates the values to the IS and/or DB.

DAQ software: already done for WD module

The Publish method - IS example

- (very) Old example:
  - each Readout Module has its own monitoring part in the DAQ panel to be checked during the run;
  - values are updated depending on the frequency of the parameters updating (we've chosen before the run, i.e. every 10 seconds)

From IS in the DAQ system

Partition 'FootPartition', server 'Monitoring'

Name	Type	Modified	Description
/Sampler/EB-RCD	SamplerInfo	30/11/17 15:59:49.880320	Contains information about
/Sampler/VTX-RCD	SamplerInfo	30/11/17 14:59:13.683121	Contains information about
Conductor	ConductorInfo	30/11/17 16:01:09.666359	contains information about
ModuleEmpty_VertexEmptyInfo	ModuleEmptyInfo	30/11/17 14:59:13.776979	Empty parameters to show o
ModuleVTX_VtxModuleInfo	ModuleVTXInfo	30/11/17 14:59:13.777248	VTX parameters to show on

Value	Type	Name	Description
VtxModule	String	BoardName	BoardName
4368	U32	IP	IPAddress
41000	U32	PortNoSlow	Remote port number for slow connection
41001	U32	PortNoFast	Remote port number for fast connection
footbo1.bo.infn.it	String	HostName	Host name of the remote server
0x2f	U32	EVT	Number of events
0x1	U32	Channels	Number of DataChannels

5 objects | 7 attribute

The Publish method - DB example

- Trigger board example:
  - end of run information stored in DB;
  - for each run (GSI: 2209-2212) we stored:
    - run number, events read by DAQ, received trigger(bare), processed trigger (gated);
  - useful to check that the numbers are the same
    - they should be, but could happen that received triggers are greater than the processed.

From DB in the DAQ system

```
mysql>  select run,daqevt,TRGb,TRGg from V2495Counters where run>2208 and run<2213;
+------+--------+--------+--------+
| run  | daqevt | TRGb   | TRGg   |
+------+--------+--------+--------+
| 2209 |  59590 |  59590 |  59590 |
| 2210 |  20467 |  20467 |  20467 |
| 2211 |  62792 |  62792 |  62792 |
| 2212 | 116400 | 116400 | 116400 |
+------+--------+--------+--------+
4 rows in set (0.00 sec)
```

Supporting material

The Publish method - I

- **IS and DB make use of the Publish method:**
  - called in **configure**, **unconfigure** and **prepareForRun** states of the machine.

- **Step by step procedure:**
  1. DAQ should know from the very start the set of parameters for each Readout Module;
  2. it sends a vector of parameters to the Readout Module;

From ModuleWD integrated into DAQ software

```
/***************************/
void ModuleWD::publishWDInfo()
/***************************/
{
  ERS_INFO(m_boardName<<"::publishWDInfo start");

  slowConn.sendPublish(m_infoToPublish);
```

called at the start of the configuration of all systems, at the start of the run and then every 10 seconds

vector of parameters sent to the remote module to be read and filled with values

SlowConnection is a common class which steers the connection used for all the DAQ-modules dialogues but the data flow.

The Publish method - II

- **IS and DB make use of the Publish method:**
  - called in **configure**, **unconfigure** and **prepareForRun** states of the machine.

- **Step by step procedure:**
  1. DAQ should know from the very start the set of parameters for each Readout Modules;
  2. it sends a vector of parameters to the Readout Module;
  3. the Readout Module receives the vector and fill it with the values;
  4. it sends the filled vector back to the DAQ;

From WaveDAQ-DAQ interface software

```
// process command publish
void TDAQServerBase::publish(){

  if ( m_verbose ) cout << "publish:: Receiving instructions for call: PUBLISH" << endl << endl;
  std::vector<uint32_t> param;
  loadVectorFromBuffer(param);
  std::vector<uint32_t> results;
  p_server->publish(param,results);  // here a vector<int> is needed!!
  if ( m_verbose ) cout << "publish:: got "<<param.size()<<" data to publish" << endl;
  //
  uint32_t* pvalues = new uint32_t[results.size()];
  for(unsigned int i=0; i<results.size(); i++){
    pvalues[i]= results[i];
  }

  size_t len = results.size()*sizeof(int);
  ssize_t n = slowStream->send( (char*) pvalues, len);
  if ( m_verbose ) cout << "publish:: sent out "
                        <<len<<" bytes; return value n="<<n << endl;
  if (n < 0) error("publish:: ERROR writing to socket");
  if ( m_verbose ) cout << "publish:: end" << endl << endl;
}
```

parameters vector sent by the DAQ system

server (WD system) fills the “results” vector filled with parameters values

server (WD system) sends the “results” vector back to the DAQ system

The Publish method - III

- **IS and DB make use of the Publish method:**
  - called in **configure**, **unconfigure** and **prepareForRun** states of the machine.

- **Step by step procedure:**
  1. DAQ should know from the very start the set of parameters for each Readout Modules;
  2. it sends a vector of parameters to the Readout Module;
  3. the Readout Module receives the vector and fill it with the values;
  4. it sends the filled vector back to the DAQ;
  5. DAQ propagates the values to the IS and/or DB.

From ModuleWD integrated into DAQ software

```
// IS info for each WD board
if ( m_infoToPublish.size() > 0 ) {
  for ( unsigned int i = 0; i<bInfo.boardNum.size(); i++ ) {
    // if ( m_verbose )
    //        ERS_LOG("ModuleWD::publishWDInfo  IF info for board: "<< bInfo.boardNum[i] );
    fillISInfo(bInfo.boardNum[i], bInfo.chan[i]);
  }
  fillISTrigInfo();
}
```

function which writes some specific trigger counters and parameters into the IS system

function which writes the values into the IS system for each WD board

function which writes some parameters into the DB and which is called in StopDC state

```
int ModuleWD::writeDBTriggerBoardInfo(){

  std::ostringstream query;

  ERS_LOG("In writeDBTriggerBoardInfo");

  query <<"INSERT INTO WDTriggerBoard (run, trigBoardID, runTotalTime, runLiveTime,
  <<m_runNumber<<", '"
  <<trigData.trigBoardID<<"', "
  <<trigData.runTotalTime<<", "
  <<trigData.runLiveTime<<", "
  // <<trigData.trig0count<<", "
  <<trigData.trigcount<<", "
  <<trigData.margMajcount<<", "
  <<trigData.margORcount<<", "
  <<trigData.TOFcount<<", "
  <<trigData.Pedestal<<", "
  <<trigData.interspillCount<<", "
  <<trigData.returnCount<<", "
  <<trigData.buffOcc0<<", "
  <<trigData.buffOcc1<<", "
  <<trigData.buffOcc2<<");";
```

The Publish method - IV

- Information to be provided:
  - some info already implemented into DataChannel which is **common** for all the remote Readout Modules integrated in the DAQ system;

From ModuleRemoteInfo.schema.xml into DAQ software

```
<class name="ModuleWDInfo" description="WD parameters to show on IS">
 <superclass name="Info"/>
 <attribute name="BoardName" description="BoardName" type="string" init-value="WDModule"/>
 <attribute name="PortNoSlow" description="Remote port number for slow connection"  type="u32"  init-value="0x00000100"/>
 <attribute name="PortNoFast" description="Remote port number for fast connection"  type="u32"  init-value="0x00000100"/>
 <attribute name="HostName" description="Host name of the remote server" type="string"  init-value="footbo1"/>
 <attribute name="EVT" description="Number of events" type="u32" format="dec" init-value="0"/>
 <attribute name="Channels" description="Number of DataChannels" type="u32" format="hex" init-value="0"/>
 <attribute name="MachineStatusValue" description="Status of the machine (3 least significant bits) and Reading_Event, DAQ_Config, DAQ_
 <attribute name="MachineStatusString" description="Status of the machine" type="string"  init-value=""/>
 <attribute name="Errors" description="Errors flags"  type="u32"  init-value="0x00000000"/>
 <attribute name="CircularBufferUsage" description="How much of the Circular Buffer is in use" type="u32"  init-value="0"/>
 <attribute name="EvtIsReady" description="If an event is ready for getNextFragment" type="u32"  init-value="0"/>
 <attribute name="LastEventSeen" description="The number of the last event seen" type="u32"  init-value="0"/>
 <attribute name="Planck" description="Recording strategy: bit 0 = PLANC, bit 1 = PLANA" type="u32"  init-value="0x0"/>
 <attribute name="WhereIsProducer" description="DBG: where in DCR is the producer" type="u32" format="hex" init-value="0"/>
 <attribute name="WhereIsConsumer" description="where in DCR is the consumer" type="u32" format="hex" init-value="0"/>
 <attribute name="EvtsDropped" description="Events dropped due to bad hw# (old or too high)" type="u32" format="dec" init-value="0"/>
 <attribute name="EvtsEmpty" description="Events build without detector data" type="u32" format="dec" init-value="0"/>

</class>
```

The Publish method - IV

- **Information to be provided:**
  - some info already implemented into DataChannel which is **common** for all the remote Readout Modules integrated in the DAQ system;
  - others are **more detectors-dependent** and then **need to be established beforehand with detector experts** (such as the WD example in the previous slides).

From ModuleRemoteInfo.schem

```
<class name="ModuleWDInfo" descript
 <superclass name="Info"/>
 <attribute name="BoardName" descri
 <attribute name="PortNoSlow" descr
 <attribute name="PortNoFast" descr
 <attribute name="HostName" descrip
 <attribute name="EVT" description=
 <attribute name="Channels" descrip
 <attribute name="MachineStatusValu
 <attribute name="MachineStatusStri
 <attribute name="Errors" descripti
 <attribute name="CircularBufferUsa
 <attribute name="EvtIsReady" descr
 <attribute name="LastEventSeen" de
 <attribute name="Planck" descripti
 <attribute name="WhereIsProducer"
 <attribute name="WhereIsConsumer"
 <attribute name="EvtsDropped" desc
 <attribute name="EvtsEmpty" descri

</class>
```

```
<class name="WaveDreamInfo" description="WaveDream parameters to show on IS">
 <superclass name="Info"/>
 <attribute name="BoardNumber" description="Board serial number" type="string" init-value="WD027"/>
 <attribute name="Channels" description="Number of channels" type="u32" init-value="8"/>
 <attribute name="ScalerChannel" description="Scaler of enabled channels" type="u32" is-multi-value="yes"  multi-value-implementati
 <attribute name="CurrentChannel" description="Current of enabled channels" type="float" is-multi-value="yes"  multi-value-implemen
 <attribute name="HVChannel" description="HV of enabled channels" type="float" is-multi-value="yes"  multi-value-implementation="ve
 <attribute name="Temperature" description="Temperature" type="u32" init-value="0"/>
</class>

<class name="WaveDreamTriggerInfo" description="WD parameters to show on IS">
 <superclass name="Info"/>
 <attribute name="RunTotalTime" description="Run total time [s]" type="u32" init-value="0"/>
 <attribute name="RunLiveTime" description="Run live time [s]" type="u32" init-value="0"/>
 <attribute name="TrigFragCounter" description="Prescaled fragmentation trigger counter [Hz]" type="float" format="hex" init-value=
 <attribute name="MargMajCounter" description="Prescaled Margarita Majority trigger rate [Hz]" type="float" format="hex" init-value
 <attribute name="MargORCounter" description="Prescaled Margarita OR trigger rate [Hz]" type="float" format="hex" init-value="0"/>
 <attribute name="TOFaloneCounter" description="Prescaled TOF-alone trigger rate [Hz]" type="float" format="hex" init-value="0"/>
 <attribute name="Pedestal" description="Prescaled Pedestal rate" type="float" init-value="0"/>
 <attribute name="TrigInterspill" description="Prescaled Interspill trigger rate [Hz]" type="float" init-value="0"/>
 <attribute name="ReturnCounter" description="Return trigger counter" type="u32" init-value="0"/>
 <attribute name="BufferOccupancy0" description="Buffer Occupancy 0" type="float" init-value="0"/>
 <attribute name="BufferOccupancy1" description="Buffer Occupancy 1" type="float" init-value="0"/>
 <attribute name="BufferOccupancy2" description="Buffer Occupancy 2" type="float" init-value="0"/>
</class>
```

Example of basics information in common DataChannel

From ModuleWD integrated into DAQ software

```
const std::string entryname = m_is_server + "." + m_boardName+"Info";
ModuleWDInfoNamed is_entry(m_ipcpartition, entryname);

try  {
  is_entry.BoardName = m_boardName;
  is_entry.PortNoSlow = m_portno;
  is_entry.PortNoFast = m_portno+1;
  is_entry.HostName = m_servName;

  // number of events
  if( m_dataChannels.size()>0 ){
    DataChannelRemote* dcp = ((DataChannelRemote*) m_dataChannels[0]);
    is_entry.EVT = dcp->getEvents();
    is_entry.Channels = m_dataChannels.size();
    is_entry.CircularBufferUsage =  dcp->getBufferSize();
    is_entry.EvtIsReady =  dcp->eventIsReady();
    is_entry.LastEventSeen = dcp->getEvents();
    is_entry.Planck = m_planck;
    is_entry.WhereIsProducer = dcp->whereProducer();
    is_entry.WhereIsConsumer =  dcp->whereConsumer();
    is_entry.EvtsDropped = dcp->getEventsDropped();
    is_entry.EvtsEmpty = dcp->getEventsEmpty();
  } else {
    is_entry.EVT = 0; // oppure?
    is_entry.Channels = 0;
    is_entry.CircularBufferUsage = 0;
    is_entry.EvtIsReady = 0;
  }
  is_entry.checkin();
```

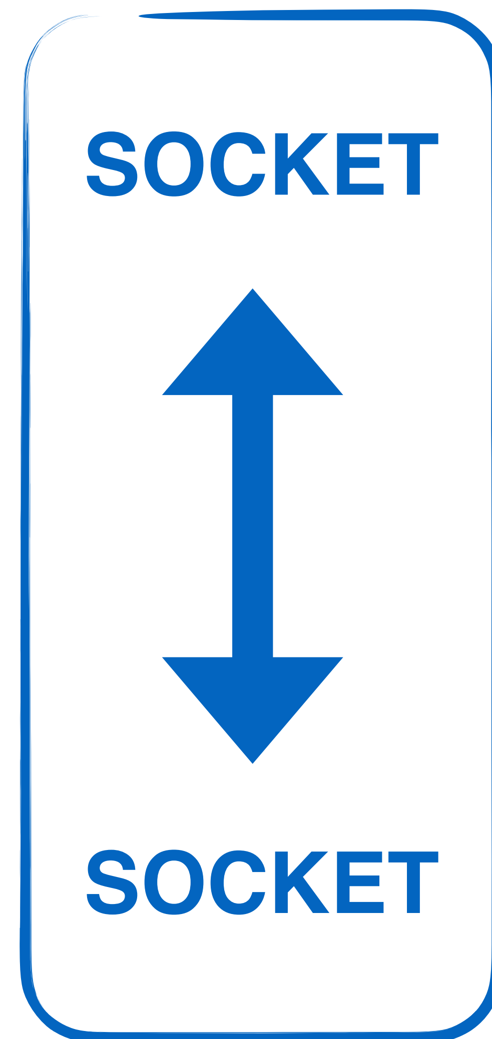
Status of DAQ integration

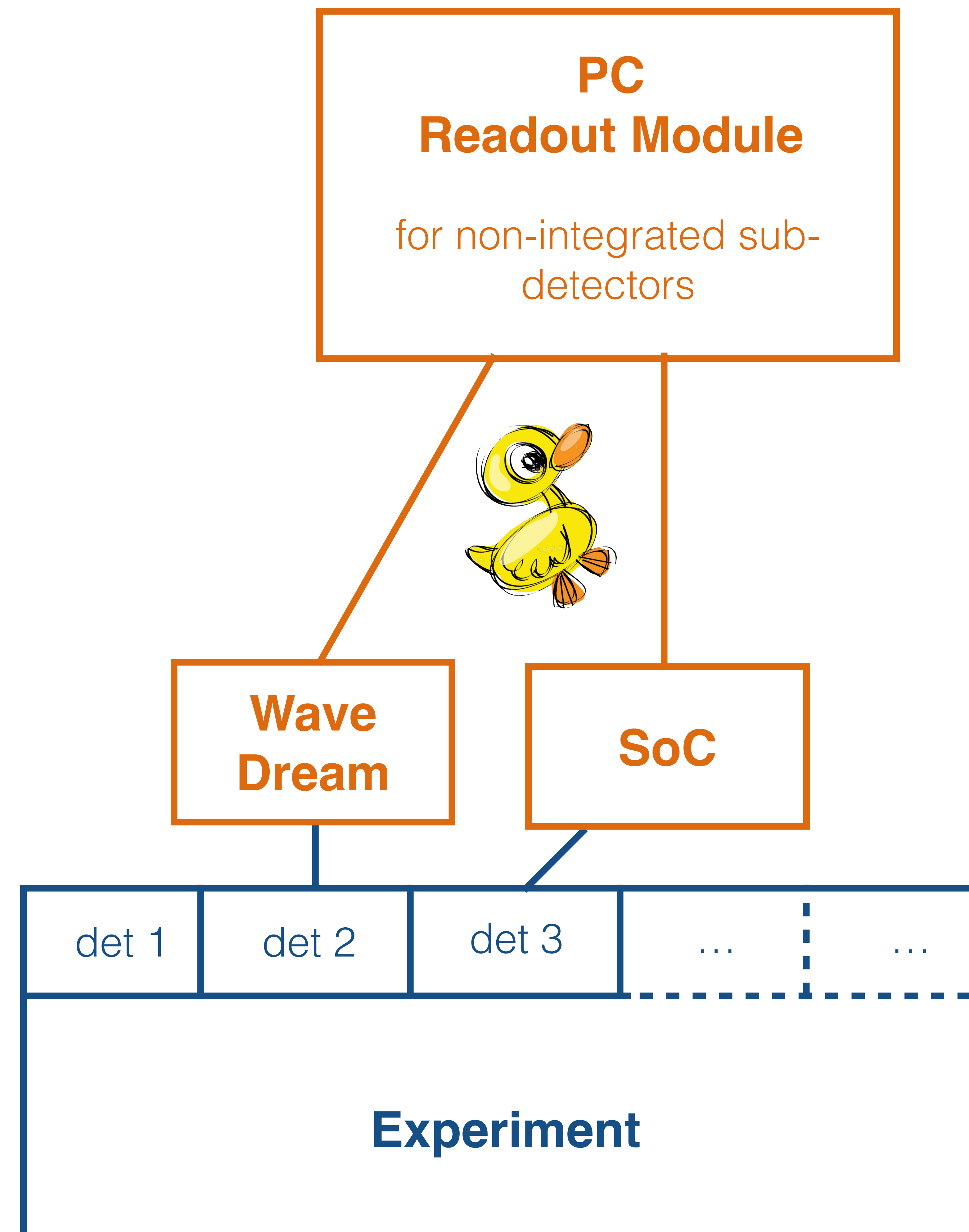
Sub-detector	What we will use	What we need to work on	From which institute	What we have now
Start Counter	Wave Dream	PC interface	Roma+Pisa	working system*
Beam Monitor	TDC	parameters for board configuration	Milano+Roma	TDC (V1190B)*
Vertex	DE10	software for TCP connection (CPU)	Frascati	DE10
IT	DE10	software for TCP connection (CPU)	Frascati	DE10
Micro Strips	DE10	software for board connection (CPU + FPGA)	Perugia	DE10
DE/TOF	Wave Dream	PC interface	Roma+Pisa	working system*
Calorimeter	?	strongly dependent on the type of chosen readout	Torino	-

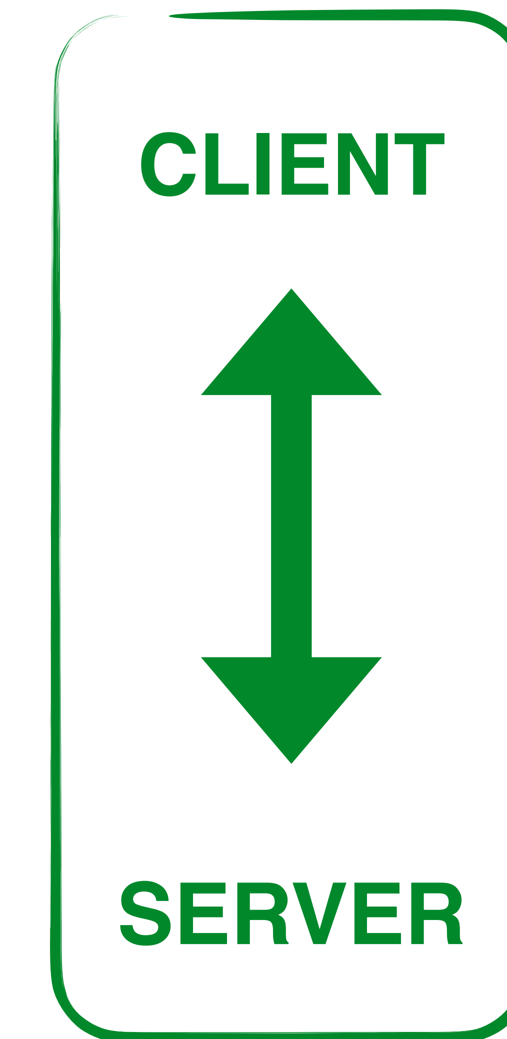
Online monitoring - old table

Sub-detector	Signal type (~ what's inside the event)	Event size (estimated)	Info for online monitoring
Start Counter	Waveforms of all channels (8) [very same as the dE/TOF, common readout]	~16 kB / event [all 8 channels together]	number of channels above threshold, signal integral value & signal peak vs trigger time (difference) for each channel
Beam Monitor	TDC time data for each channel (36)	~0.2 Kbyte/event	number of hits
Vertex	Hits zero suppressed for three consecutive frames.	~0.5 Kbyte/event	Event size per sensor
IT	Hits zero suppressed for three consecutive frames.	~4.0 Kbyte/event	Event size per sensor
Micro Strips	Raw data for each sensor: 1024 strips ID for each chip	1.5 kByte/event	Increasing counter for each time a chip ID is received.
DE/TOF	Waveforrms for hit channels + 2 wfm per board (clock synchronisation channels). Some scaler read out, like Margarita trigger rates.	~0.8 kB x average occupancy (rebinning on the trailing edge)	Channel and trigger rates(coming together with data), channel current (to be read asynchronously from WDB, slow control like).
Calo	Digitization of the channels above threshold	1.5kB x average occupancy	number of hit crystals, pattern of hit crystals,signal peak and integral values, time constants of the shape analysis

Data flow - connection







- Socket uses a **TCP connection**, which is defined by two endpoints (sockets);
- all **C++ Standard Template Library based**;
- provide **reliable two-way communication**.

- **Transmission Control Protocol/ Internet Protocol (TCP/IP)**:
- requires **little central management** and makes **networks reliable**;
- IP is **compatible with all operating systems and with all types of computer hardware and networks**.

Reminder: from our CDR

from CDR

Detector	Board(s)	DAQ channels	max event rate (kHz)	Event size (bytes)
Trigger	V2495	1	10	40 B
Start Counter	DreamWave	4	1	8.2 kB
Beam Monitor	TDC	36	5	0.1 kB
Vertex detector	SoC on DEx	$4 \cdot 10^6$	2	0.9 kB
Inner tracker	SoC on DEx	$28 \cdot 10^6$	2	2.1 kB
Outer tracker	Custom	$6 \cdot 10^3$	2	0.5 kB
$\Delta E/\Delta x$	DreamWave	80	1	8.4 kB
Calorimeter	QDC	400	2	1.7 kB
Total DAQ	Storage PC	-	1	22 kB

- Numbers from GSI experience
  - DAQ (trigger+BM+file structure): 530 B
  - VTX: 650 B
  - SC+TOFW: 29 kB

30 kB

Tools used in the project

SOCKETS

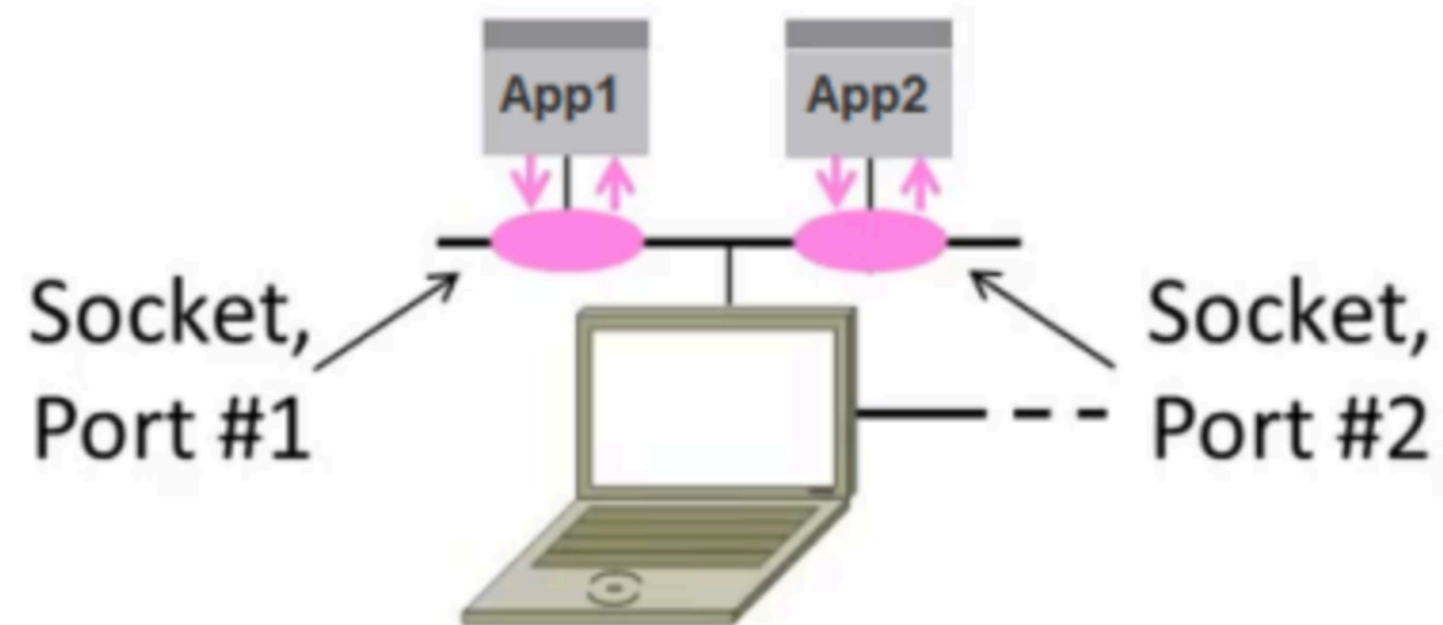
- A way of speaking to other programs using **standard file descriptors**:
  - ➡ make a call to the **socket() system routine**;
  - ➡ after the socket() returns the socket descriptor, start communicate through it using the specialised **send()/recv() socket API calls**.
- Socket uses a **TCP connection**, which is defined by two endpoints (sockets);
- It is the socket pair (the 4-tuple consisting of the **client IP address**, **client port number**, **server IP address** and **server port number**) that specifies the two endpoints that uniquely identifies each TCP connection in an internet;
- The purpose of ports is to **differentiate multiple endpoints** on a given network address.
- **Why using the socket:**
  - ✓ all **C++ Standard Template Library based**;
  - ✓ provide **reliable two-way communication**;
  - ✓ immediate confirmation that **what has been sent actually reached its destination**;
  - ✓ ensure that **data are not lost or duplicated** and that **the order is the same** from the sender to the receiver.

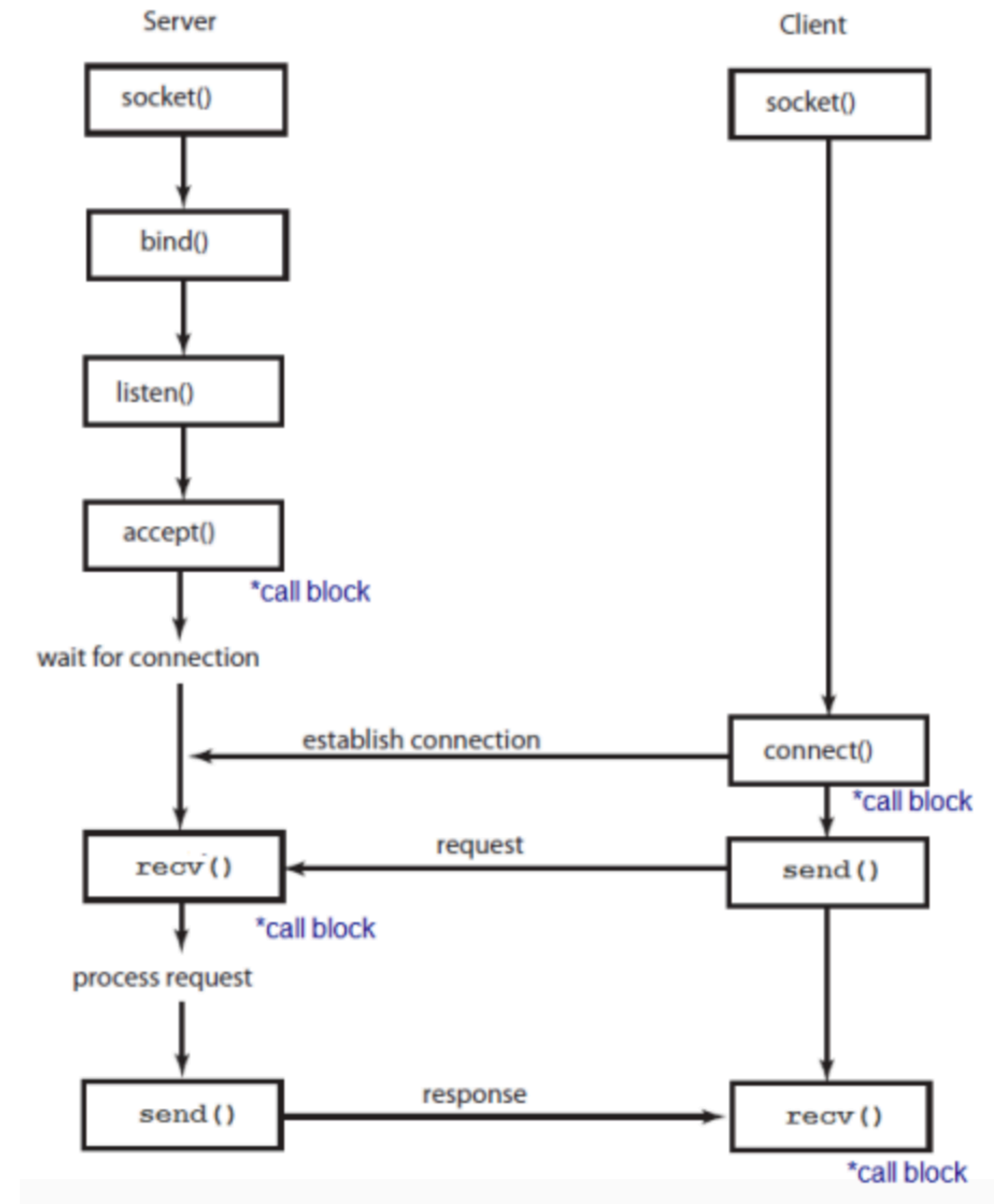
Tools used in the project

CONNECTIONS

- **Transmission Control Protocol/Internet Protocol (TCP/IP)**:
  - ➡ providing **end-to-end communications** that identify how it should be broken into packets, addressed, transmitted, routed and received at the destination.
  - ➡ It uses the **client/server model** of communication in which a user or machine (a client) is provided a service (like sending a message) by another computer (a server) in the network
- **Why using the TCP/IP:**
  - ✓ requires **little central management**;
  - ✓ designed to make **networks reliable**, with the **ability to recover automatically** from the failure of any device on the network;
  - ✓ IP is **compatible with all operating systems and with all types of computer hardware and networks**.

Socket structure





TCP/IP connection

TCP/IP model layers

- TCP/IP functionality is divided into four layers, each of which include specific protocols.

    1. The **application layer** provides applications with standardised data exchange.

    2. The **transport layer** is responsible for maintaining end-to-end communications across the network. TCP handles communications between hosts and provides flow control, multiplexing and reliability.

    3. The **network layer**, also called the internet layer, deals with packets and connects independent networks to transport the packets across network boundaries.

    4. The **physical layer** consists of protocols that operate only on a link - the network component that interconnects nodes or hosts in the network.

Remote device package - (very) little documentation

Functionalities of classes

- **classes for memorisation of data**
  - circular → circular_buffer
  - EventCircularBuffer → public circular_buffer
- **low level classes to comunicate with FPGA**
  - FpgaIOBase → base class for low level actions on FPGA
  - FpgaInterface → public FPGAIOBase → high level actions on FPGA
- **generic classes for detectors**
  - DAQServerInterface → generic DAQ interface
  - SOCServerInterface → public DAQServerInterface → interface for FPGA, uses FpgaInterface class
  - SOCSimuServerInterface → public DAQServerInterface → data simulator for FPGA
- **main Server class**
  - TDAQServerBase → uses DAQServerInterface to interact with connection and device to be read
    can be used as base in derived class
    two threads corresponding to two methods of the class
- **generic classes for TCP connection**
  - tcp* → include all the functions to connect, listen and bind the sockets and to send, receive and read data
- **MainTDAQServer** → main code: instantiates TDAQServerBase and derived class from DAQServerInterface