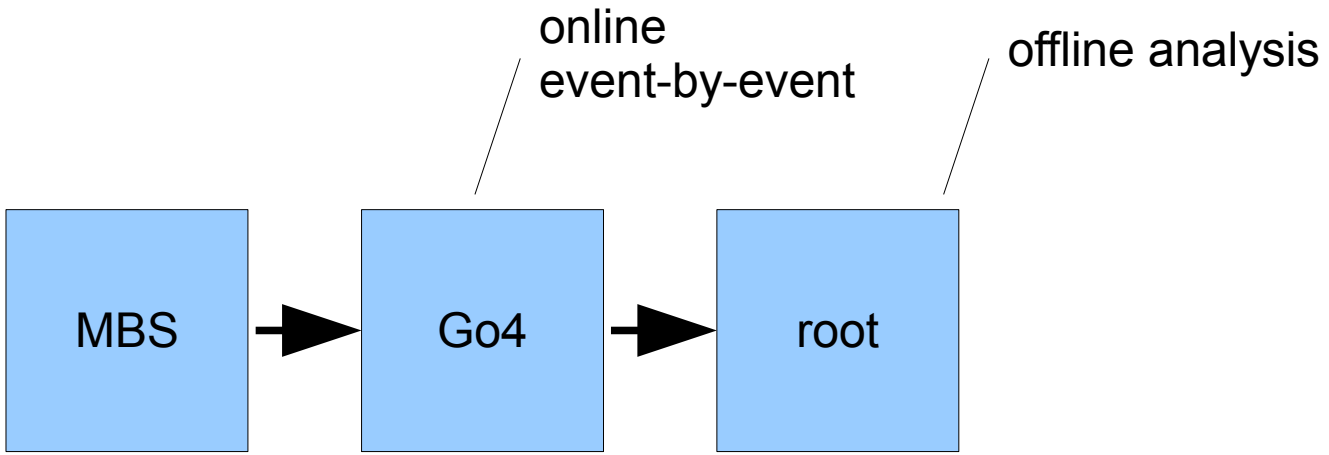
Kickoff meeting

PulSAr

Pulse Shape Analysis for the GSI Scanner Framework

GSI Scanner / PSA Activities

- AGATA symmetric crystal scan (N.G.)
- Cluster Neutron Damage (L.C.)
- Hybrid Test (Palit, T.H.)
- Ge timing (M.B. (summerst.)) ...ongoing...
- Cluster Neutron Damage (summerst.) (no scanner)
- AGATA asymmetric crystal scan (M.L.,T.H.) ... analysis pending...



Coding Example: Baseline subtraction

Subtract baseline from signal

```
for (int seg=0;seg<NSEG;seg++){
    double baseline = 0;
    for (int t=0;t<PRE_N_BASELINE;t++){
        baseline = baseline + this->shape[nbSegs[seg]][t];
    }
    baseline = baseline / (double)PRE_N_BASELINE;
    treeOut->baseline = baseline;
    for (int t=0;t<NBIN;t++){
        treeOut->signal[seg][t] = shape[nbSegs[seg]][t] - baseline;
    }
}
```

Instead of fiddling with arrays and stuff we want to do...

```
treeOut->baseline = subtractBaseline(&signal,nbins);
```

… and

```
double subtractBaseline(double* signal,int nbins){
    double baseline = getArrayAverage(signal,nbins);
    addToArray(signal,-baseline);
    return baseline
}
```

self-documenting code

What root can do for us / What root can't do for us

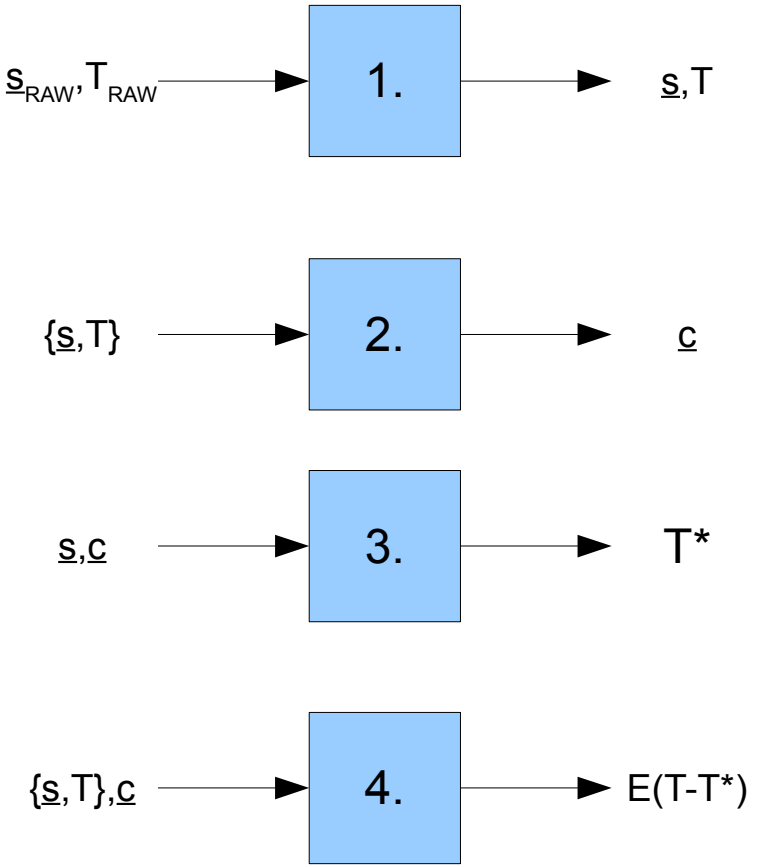
Fitting data can be done by using TlinearFitter

```
TH1D* convergence;
convergence = new TH1D("fit convergence", "fit convergence", 2000,0,2000);

TLinearFitter *linear_fitter=new TLinearFitter(1);
linear_fitter->SetFormula(formula);

Long64_t nentries = fChain->GetEntriesFast();
Long64_t realEntry = 0;

for(Long64_t ev = 0; ev<nentries; ev++){
    fChain->GetEntry(ev);
    realEntry++;
    linear_fitter->AddPoint(input,time_diff);
    if(checkConv && realEntry % 100 == 0 && realEntry>0) {
            linear_fitter->EvalRobust(robust);
            convergence->Fill(realEntry/100, linear_fitter->GetParameter(1));
    }
}
linear_fitter->EvalRobust(robust);
linear_fitter->PrintResults(3);

Double_t* results = new Double_t[linear_fitter->GetNumberFreeParameters()];
cout<<linear_fitter->GetNumberFreeParameters()<<endl;

for(int i=0; i<linear_fitter->GetNumberFreeParameters(); i++){
        results[i]=linear_fitter->GetParameter(i);
}

new TCanvas();
convergence->Draw();
```

Same thing in a (general) function

```
MyLinearFitter *linear_fitter=new MyLinearFitter(...pass all parameters here...);
double* c = linear_fitter->GetCoefficients();
```

"Timing corrections" functions



1. Prepare data, i.e.
- time alignment
- baseline subtraction
- normalize
- calculate rise-times ($\underline{s}$)

2. Fit data
- use root TLinearFitter
- output = coeficients of the fit

3. Apply fit
- use coefficients to estimate $T \approx T^* = \underline{c}.\underline{s}$

4. Verify Fit (actually before 3.)
- check correlations
- output = quality of the fit

Plugins via Interfaces (cf. Go4)

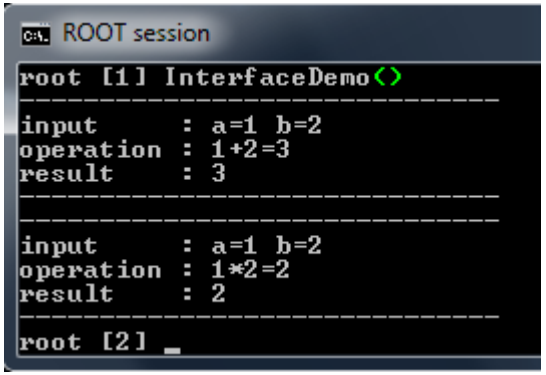
Public Interface

```
void BinaryCalculator(BinaryOperation* operation,double a,double b);
// Interface = Pure Abstract Class
class BinaryOperation {
    public:
        virtual double applyOperation(double a,double b) = 0;
        virtual char*  toString()=0;
};
```

User does not need to know details of implementation but only its interface

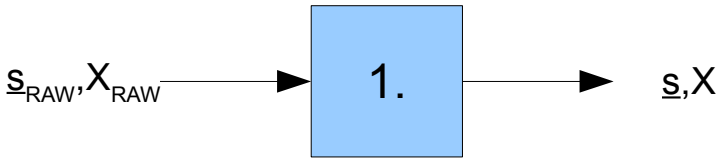
User code

```
void InterfaceDemo(){
    // implement the interface
    class : public BinaryOperation {
        double applyOperation(double a,double b) {return a + b;}
        char* toString(){return "+";}
    } adder;
    // and use it
    BinaryCalculator(&adder,1,2);
    // another implementation of the interface
    class : public BinaryOperation {
        double applyOperation(double a,double b) {return a*b;}
        char* toString(){return "*";}
    } multiplier;
    // another usage
    BinaryCalculator(&multiplier,1,2);
}
```

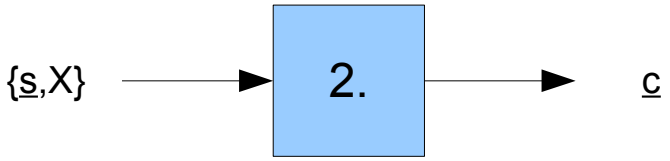


Parts of code that are plugged in via an interface can be exchanged without touching the rest of the code

Abstract parametric PSA functions

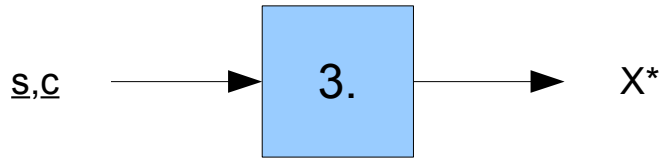


1. „Prepare“

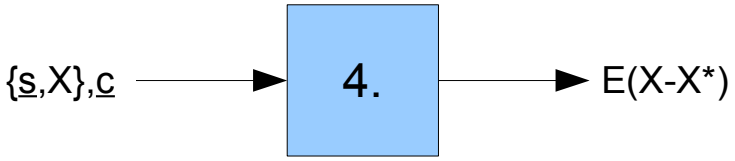


2. „Fitting“

- simple linear fit: $F(\underline{s}) = \underline{c}.\underline{s}$
- model param., e.g $F(s) = a*G(\underline{s}) + b*H(\underline{s})$
- …



3. Apply: $F(\underline{s}) = X^*$



4. Verify Fit

- check correlation
- output = quality of the fit

PulSAr

TODO:
- Fast PSA as NARVAL actor
- Analyse data from AGATA scan -> create Pulse Shape Database
- Continue on "Timing Corrections via PSA"

→ **PulSAr – Framework for Pulse Shape Analysis and GSI Scanner**
One Framework with code needed for PSA & Scanner analysis
Provides interfaces to Go4, NARVAL, ...

Example:

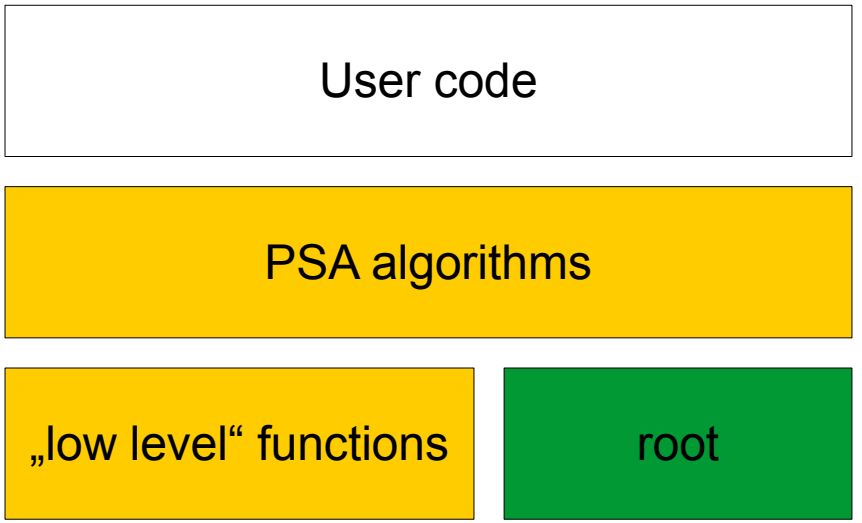
User **reads data** →

and calls

PulseShape->SubtractBaseline() →

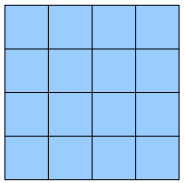
which calls

double getAverage(double* a,int N);
void addToArray(double* a,double b,int N); →

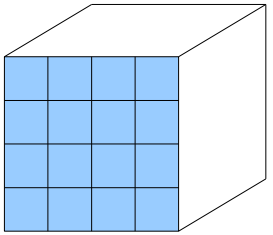


SCANNER ANALYSIS ISSUES

Scanner Analysis: Graphical Cut Overhead



PSD
PxP pixel



Scanned Volume
PxPxP voxel

N – total number of events in each dataset

„no cuts“
For each pair of signals:
Check if trajectories cross
Compare them

→ **Total**
#comparisons = N^2
#loading an event = 2N
#load PSD coords = 0

„old“
For each voxel (i.e. P^3 times) :
Apply graphical cut
N / P^2 events from each dataset
Compare the two datasets
N^2 / P^4 comparisons

→ **Total**
#comparisons = N^2 / P
#loading an event = 2NP
#load PSD coords = $2NP^3$!!!

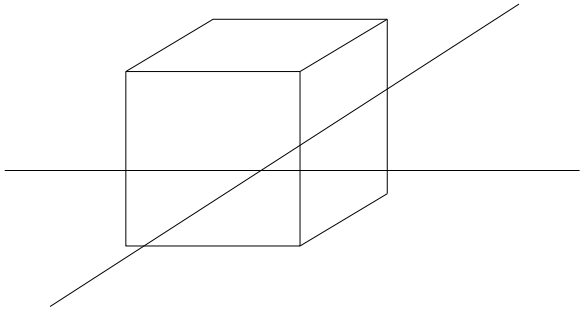
„cut before“
Sort events: one file per psd pixel
N / P^2 events per pixel

For each front pixel (i.e. P^2 times)
Select files from side (N / P events)
Compare them (N^2 / P^3)

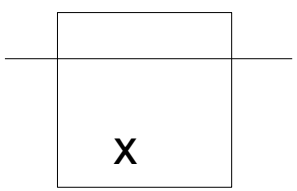
→ **Total**
#comparisons = N^2/P
#loading an event = N(P+1)
#load PSD coords = 0

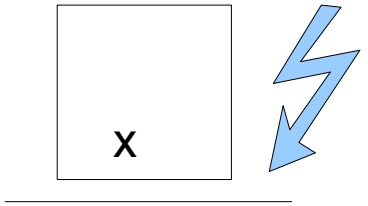
Scanner Analysis Graphical cut continued...

Graphical cut = select one voxel and then all trajectories that pass this voxel



Frontview





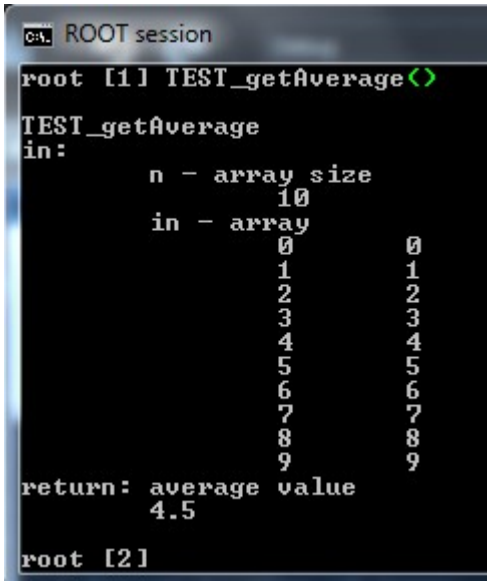
This two events are compared

but not this two

Are we wasting lots of good events?

Write, Test & Debug Only Once

```
/**
 * Calculates average value of an input array.
 * See getAverageTest() for an example.
 * @param in array
 * @param n  size of array
 * @return average value
 */
double getAverage(double* in,int n){
    double sum = 0;
    for (int i=0;i<n;i++){sum = sum+in[i];}
    return sum / (double)n;
}
/**
 * Test for getAverage().
 * @param verbose
 */
void TEST_getAverage(int verbose = 1){
    const int n = 10;                    // size of input array
    double* in = new double[n];          // allocate input array
    for (int i=0;i<n;i++){in[i] = i;}    // input values
    double out = getAverage(in,n);       // call the function
    if (verbose){
        cout << endl << "TEST_getAverage" << endl;
        cout << "in:" << endl;
        cout << "\t" << "n - array size" << endl;
        cout << "\t\t" << n << endl;
        cout << "\t" << "in - array" << endl;
        for (int i=0;i<n;i++){cout << "\t\t" << i << "\t" << in[i] << endl;}
        cout << "return: average value" << endl;
        cout << "\t" << out << endl << endl;
    }
    delete [] in;                        // free memory
}
```



- more work in the beginning

+ write, test & debug only once

PSA & Scanner "Synergies"

PSA for AGATA: T0 determination

- shift measured signal in time w.r.t. basis signal to get best match → T0

Scanner: PSD position determination

- shift profile of collected light w.r.t. reference signal to get best match → x/y position

...seems completely unrelated, but actually it is the same task!

Finding PSD position is already "almost decoupled":

pos_x=brent(XAnodes,Refx,ax, bx, cx,&counter);

+ takes **any** TH1 histograms as input

- but: many "magic numbers"

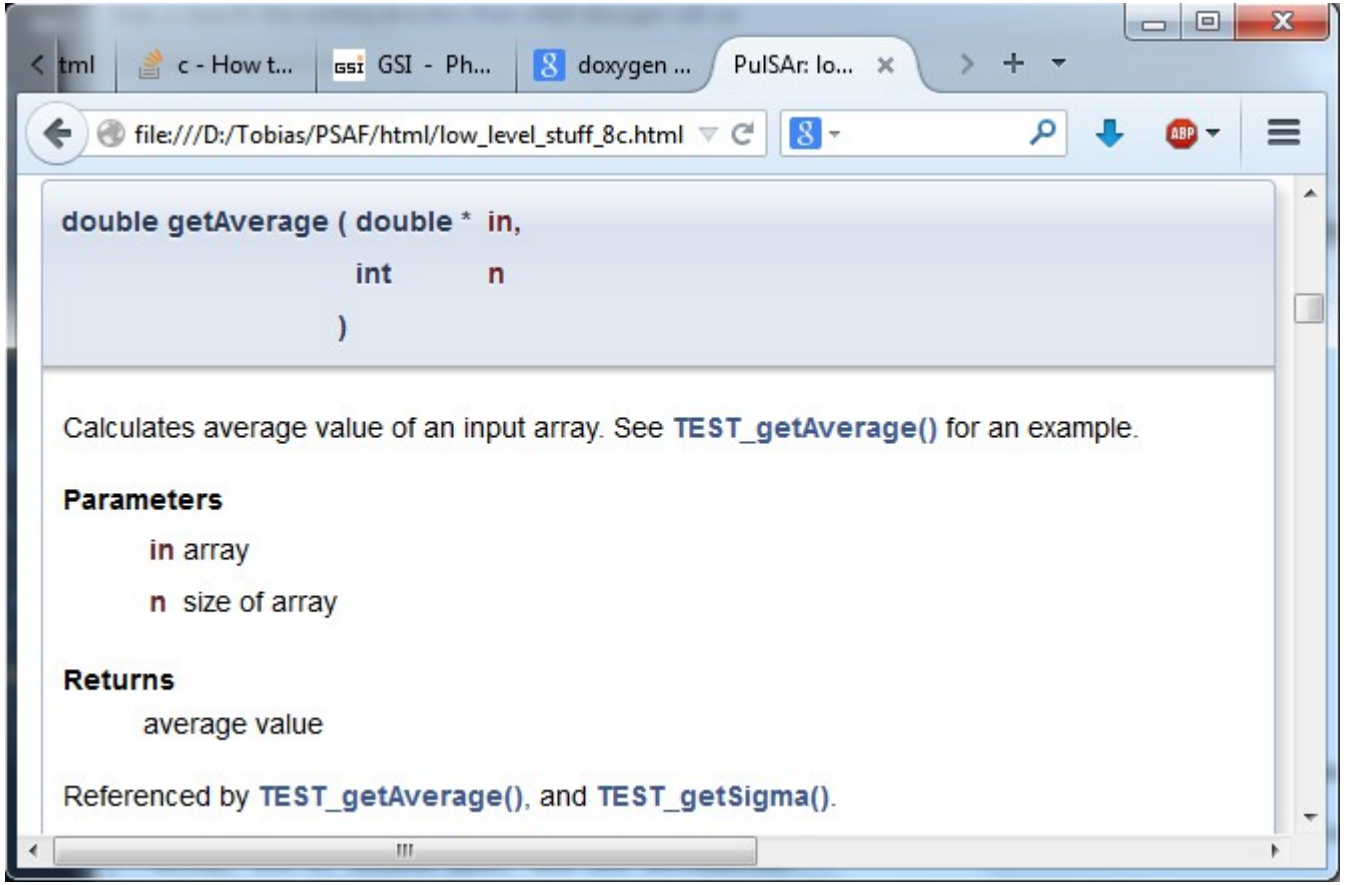
- documentation: ~0

```
66      probbinhigh = h1->GetXaxis()->FindBin(highervalue);
        if(probbinhigh == 17)
68        probbinhigh = 16;
69
70
```

(Scanner: LYSO Gain Matching ...seems very different, but also quite similar)

Documentation

DoxyGen: Standard tool to create documentation from annotated C++ code



+ use wiki@gsi to collect documents