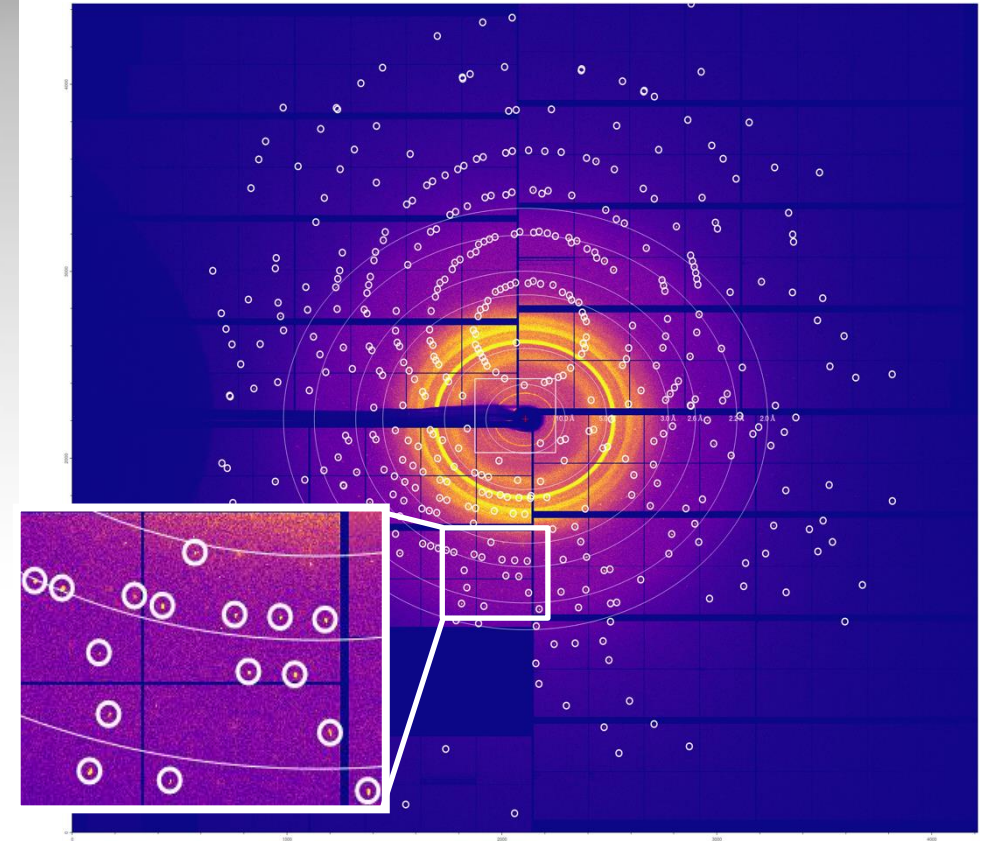
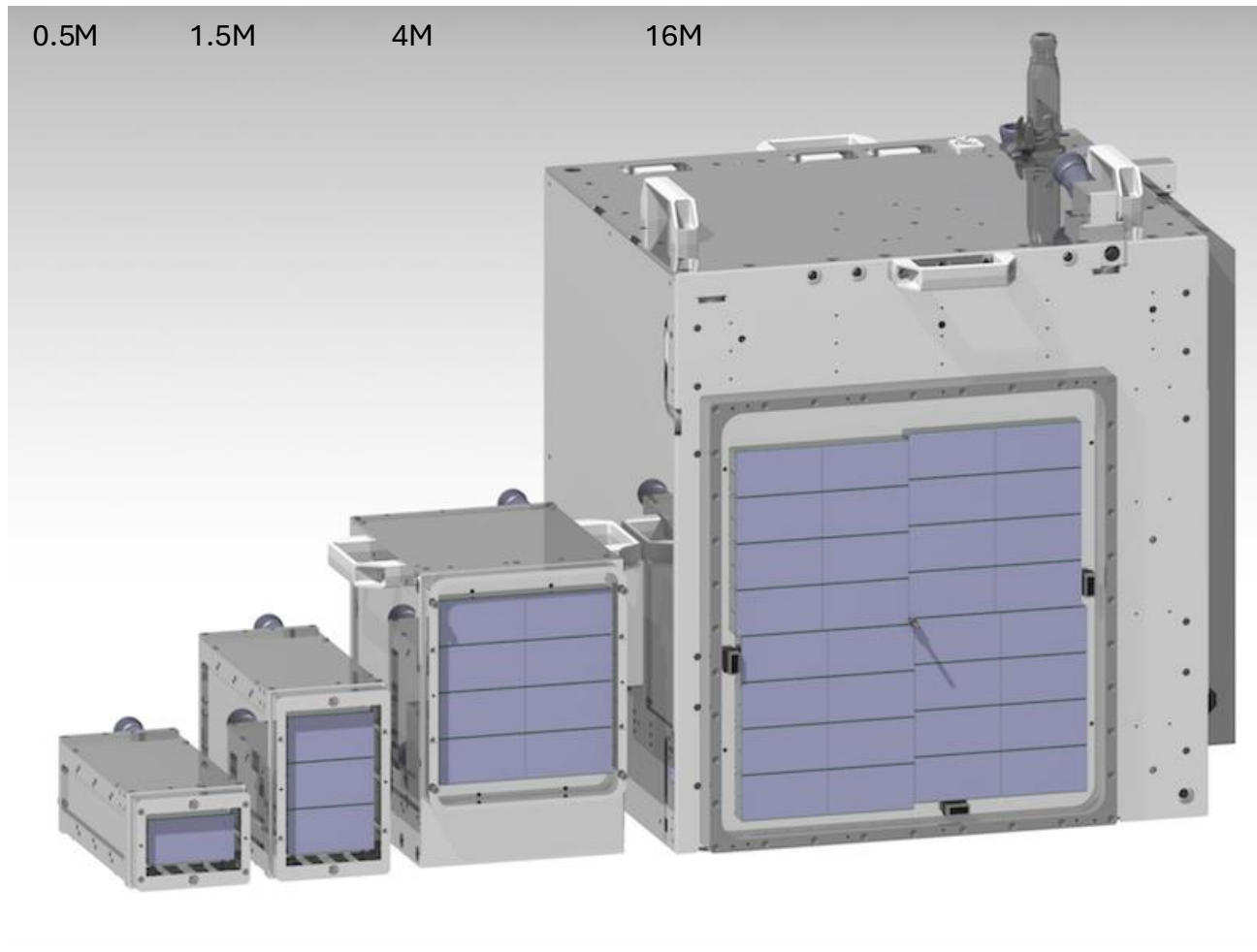


PSI Center for
Photon Science

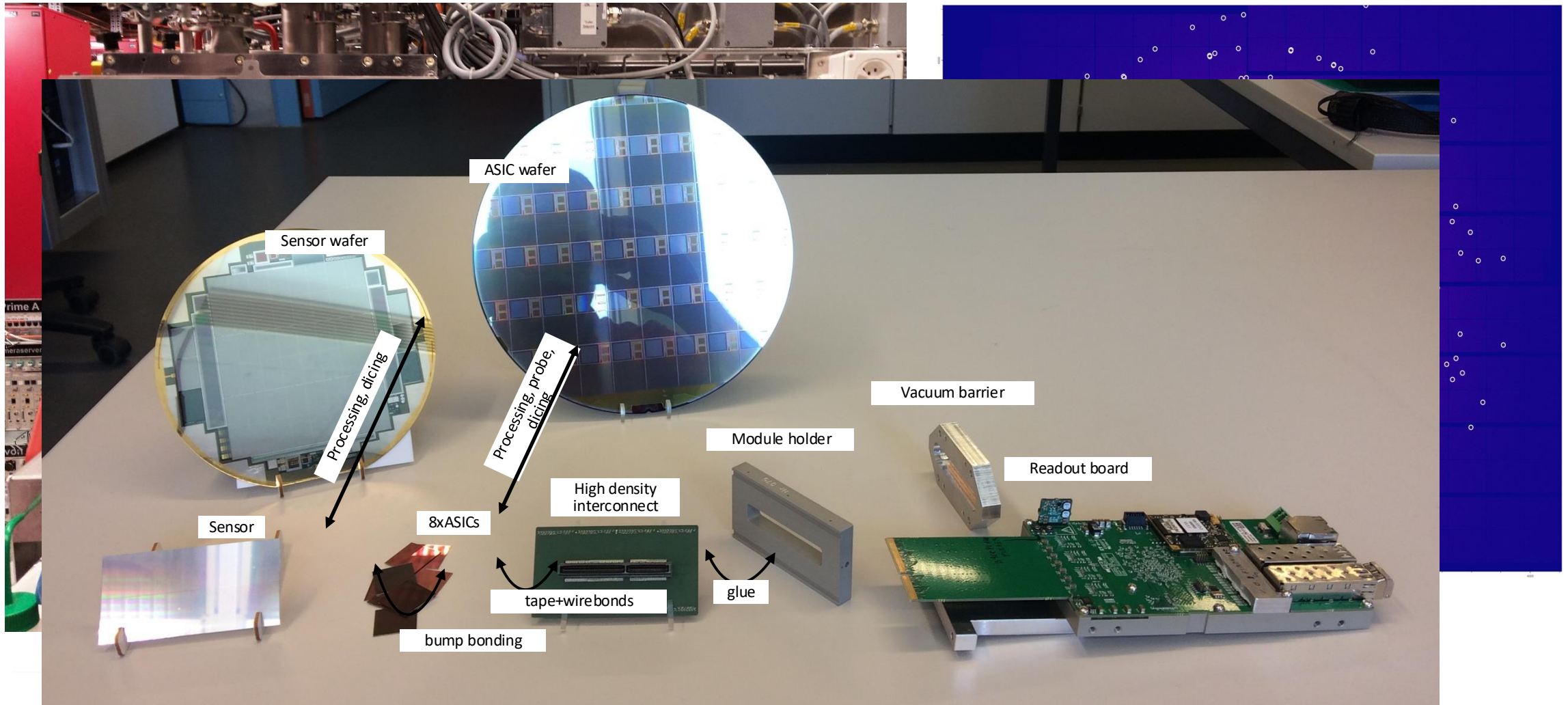
The case for research software engineers

Erik Fröjdh (on behalf of the CPS Detector Group, PSI)
2026-07-31 - RSE Day@ETHZ

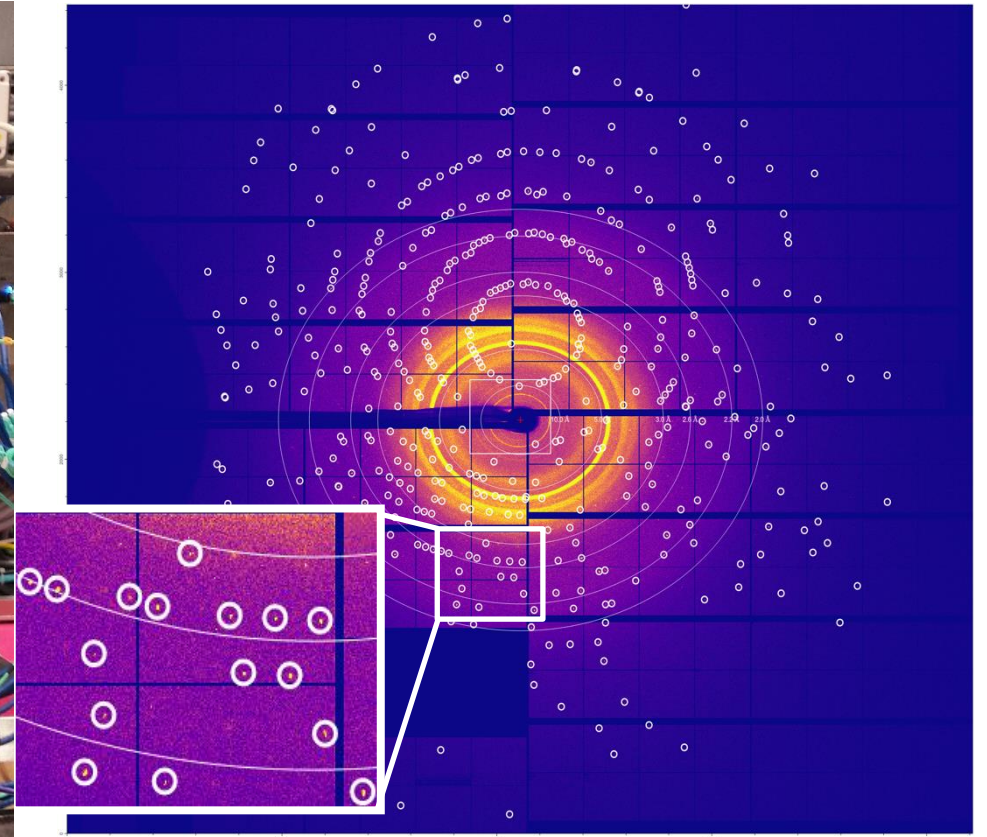
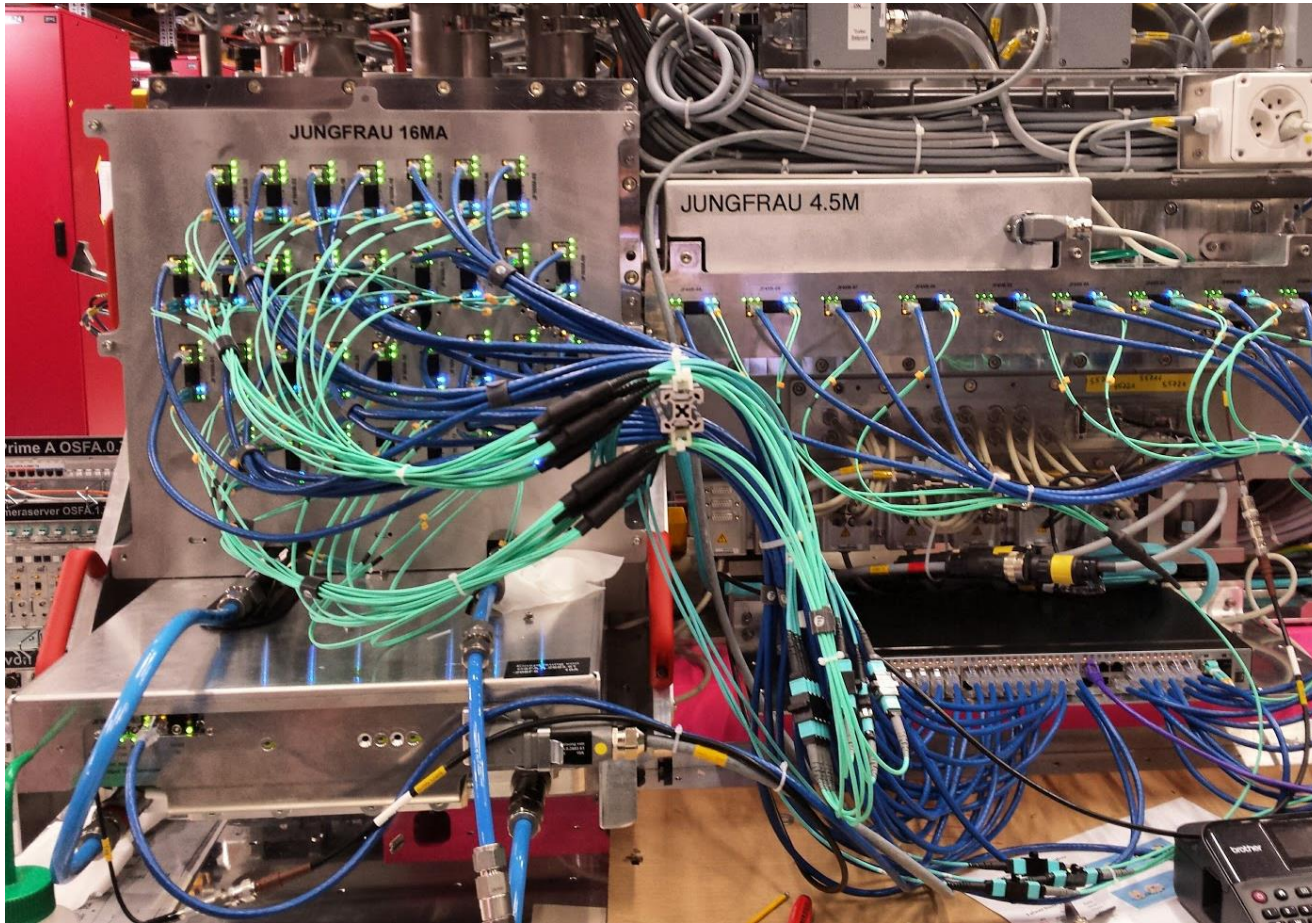
Hybrid pixel detectors



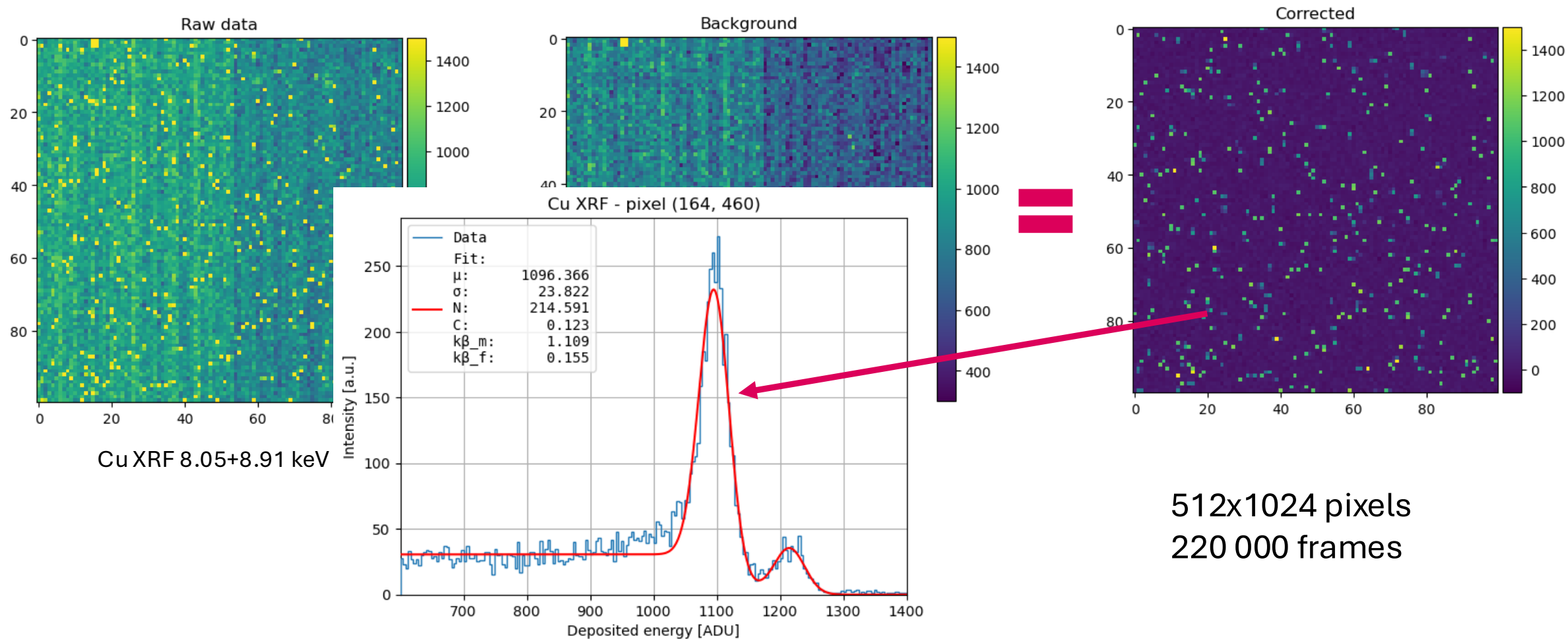
Hybrid pixel detectors



Hybrid pixel detectors



Case study: Calibration of the Jungfrau detector



Benchmarks run on:



- AMD Ryzen Threadripper PRO 7955WX 16-Cores
- 128GB memory 8xDDR5 but limited to 115.2GB/s by the links to CCD (Core Complex Die)
- Stream benchmarks (<https://www.cs.virginia.edu/stream/>)
 - 43GB/s single core
 - 104GB/s 4+ cores
- NFS share, 40Gbit/s fiber
 - 1GB/s single threaded reads
 - Multi threaded reads saturate the 40Gbit/s line if cached but normally limited to around 2GB/s

- C++ (98/undefined) + ROOT v5.34 (2012)
- Implementation per calibration step in a single file
- No tooling
- No documentation
- Runtime: 2h36min (XRF)

Goals:

- Make it 50x faster
- Remove ROOT dependency (for portability)
- Move to Python (parsing, interactivity, script workflow)

Legacy code: <https://gitea.psi.ch/detectors/JFCalibration>

Where to start?



```
10 #include "sls_detector_calibration/jungfrauFile.C"
11 #include "sls_detector_calibration/jungfrauPixelMask.C"
12 #include "sls_detector_calibration/jungfrauPedestal.C"
13 #include "sls_detector_calibration/energyCalibration.h"
14 #include "sls_detector_calibration/energyCalibration.cpp"
15
16 #include "TGraph.h"
17 #include "TGraphErrors.h"
18 #include "TFile.h"
19 #include "TPaveStats.h"
20 #include "TLegend.h"
21 #include "TText.h"
22
23 #include <sys/stat.h>
24 #include <fmt/format.h>
25 #include <unistd.h>
26 #include <limits.h>
27
28 int main(int argc, char* argv[]) {
29     jungfrauStyle();
30     gStyle->SetOptFit(11);
31
32     /*
33     if (argc != 6) {
34         cout << "Correct usage:" << endl;
35         cout << "arg 1: specify module number" << endl;
36         cout << "arg 2: specify HGB or GB" << endl;
37         cout << "arg 3: specify data location" << endl;
38         cout << "arg 4: specify pede file" << endl;
39         cout << "arg 5: specify data file prefix" << endl;
40         cout << " " << endl;
41         exit(1);
42     }
43     */
44     // /uncomment for SR
45     if (argc != 7) {
46         cout << "Correct usage:" << endl;
47         cout << "arg 1: specify module number" << endl;
48         cout << "arg 2: specify HGB or GB" << endl;
49         cout << "arg 3: specify data location folder" << endl;
50         cout << "arg 4: specify pede file prefix" << endl;
51         cout << "arg 5: specify data file prefix" << endl;
52         cout << "arg 6: specify location of analysis root files" << endl;
53         cout << " " << endl;
54         exit(1);
55     }
56     // /uncomment for VH 210906
57     string module_str = argv[1];
58     string gain_str = argv[2];
59     bool isJF11=false;
60     if (gain_str == "HGBJF11") {
61         gain_str = "HGB";
62         isJF11=true;
63     }
64
65     string data_loc = argv[3];
66     string pede_file = argv[4];
67     string data_file = argv[5];
68     string anadata_loc = argv[6]; //uncomment for VH 210906
69
70     int createHistoFile = 1;
71
72     //char histoname[128];
73     //char savename[128];
74     char histoname[256]; // VH 210902
75     char savename[256]; // VH 210902
76     int file_n = 20;
77
78     char buffer[PATH_MAX];
79     getcwd(buffer, PATH_MAX);
80     std::string cwd = buffer;
81     fmt::print("I'm in: {}\n", cwd);
82
83     // create necessary directories with permissions drwxrwxr-x
84     // /data/Mxxx
85     int ret = 0;
86
87     fmt::print("Created directory: {}\n", savename);
88     // plots/Mxxx
89     sprintf(savename, "plots/M%s", module_str.c_str());
90     ret = mkdir(savename, S_IRWXG | S_IROTH | S_IXOTH);
91
92     // plots/Mxxx/CuFluo
93     sprintf(savename, "plots/M%s/CuFluo", module_str.c_str());
94     ret = mkdir(savename, S_IRWXG | S_IROTH | S_IXOTH);
95
96     // plots/Mxxx/CuFluo/GB
97     sprintf(savename, "plots/M%s/CuFluo/GB", module_str.c_str());
98     ret = mkdir(savename, S_IRWXG | S_IROTH | S_IXOTH);
99
100     // plots/Mxxx/CuFluo/HGB
101     sprintf(savename, "plots/M%s/CuFluo/HGB", module_str.c_str());
102     ret = mkdir(savename, S_IRWXG | S_IROTH | S_IXOTH);
103
104     // /mnt/pcnoench_jungfrau_data/jungfrau_ana_sophie/Mxxx/CalibAna
105     //sprintf(savename, "/mnt/sls_det_storage/jungfrau_data/jungfrau_ana_sophie/M%s",
106     //    module_str.c_str()); //uncomment for VH 210906
107     ret = mkdir(savename, S_IRWXG | S_IROTH | S_IXOTH);
108
109     TCanvas* c1 = new TCanvas("c1", "");
110
111     if (createHistoFile) {
112         jungfrauFile *thisfile = new jungfrauFile();
113         jungfrauPedestal *pedestalObject = new jungfrauPedestal();
114         pedestalObject->pedestalSetFrames(100);
115         static uint16_t pedestals16_G0_start[NCH];
116         static double pedestMS16_G0[NCH];
117
118         TH2F* pedestalsG0 = new TH2F("pedestalsG0", "", NC, -0.5, NC-0.5, NR, -0.5, NR-0.5);
119         TH2F* pedestalsG1 = new TH2F("pedestalsG1", "", NC, -0.5, NC-0.5, NR, -0.5, NR-0.5);
120         TH2F* pedestalsG2 = new TH2F("pedestalsG2", "", NC, -0.5, NC-0.5, NR, -0.5, NR-0.5);
121
122         TH2F* pedeMSG0 = new TH2F("pedeMSG0", "", NC, -0.5, NC-0.5, NR, -0.5, NR-0.5);
123
124         // open pede file
125         sprintf(savename, "%s/%s_V%6.6d.dat", data_loc.c_str(), pede_file.c_str()); //VH
126         thisfile->open((char*)savename, 0);
127
128         // calculate pixel mask
129         pixelMaskObject->initialisePixelMask(pixel_mask);
130
131         int nevents = 0;
132         while (thisfile->readNextFrame()) {
133             nevents++;
134             thisfile->rewind();
135             cout << "read " << nevents << " events" << endl;
136
137             if (nevents == 2999 || nevents == 3000) {
138                 for (int i = 0; i < 1000; i++) {
139                     thisfile->readNextFrame();
140                     pixelMaskObject->maskIfGainNot(0, thisfile->getFrameDataHandle(), pixel_mask);
141                 }
142                 for (int i = 0; i < 1000; i++) {
143                     thisfile->readNextFrame();
144                     pixelMaskObject->maskIfGainNot(1, thisfile->getFrameDataHandle(), pixel_mask);
145                 }
146                 for (int i = 0; i < 999; i++) {
147                     thisfile->readNextFrame();
148                     pixelMaskObject->maskIfGainNot(3, thisfile->getFrameDataHandle(), pixel_mask);
149                 }
150             }
151         }
152     }
153
154     while (thisfile->readNextFrame()) {
155         pixelMaskObject->maskIfGainNot(0, thisfile->getFrameDataHandle(), pixel_mask);
156     }
157
158     thisfile->rewind();
159
160     // calculate pedestals
161     if (nevents == 2999 || nevents == 3000) {
162         for (int i = 0; i < 1000; i++) {
163             thisfile->readNextFrame();
164             pedestalObject->addFrameToPedestalCalculation(thisfile->getFrameDataHandle());
165         }
166         for (int i = 0; i < NCH; i++) {
167             if (pixel_mask[i] == true) {
168                 pedestalsG0->Fill(1/NC, 1/NC, pedestalObject->pedestalOfChannel(i));
169                 pedeMSG0->Fill(1/NC, 1/NC, pedestalObject->rmsOfChannel(i));
170             }
171             pedestalObject->pedestalClear();
172         }
173         for (int i = 0; i < 1000; i++) {
174             thisfile->readNextFrame();
175             pedestalObject->addFrameToPedestalCalculation(thisfile->getFrameDataHandle());
176         }
177         for (int i = 0; i < NCH; i++) {
178             if (pixel_mask[i] == true) {
179                 pedestalsG1->Fill(1/NC, 1/NC, pedestalObject->pedestalOfChannel(i));
180                 pedestalsG2->Fill(1/NC, 1/NC, pedestalObject->pedestalOfChannel(i));
181             }
182             pedestalObject->pedestalClear();
183         }
184         for (int i = 0; i < 999; i++) {
185             thisfile->readNextFrame();
186             pedestalObject->addFrameToPedestalCalculation(thisfile->getFrameDataHandle());
187         }
188         for (int i = 0; i < NCH; i++) {
189             if (pixel_mask[i] == true) {
190                 pedestalsG0->Fill(1/NC, 1/NC, pedestalObject->pedestalOfChannel(i));
191                 pedestalsG2->Fill(1/NC, 1/NC, pedestalObject->pedestalOfChannel(i));
192             }
193             pedestalObject->pedestalClear();
194         }
195         // GB again for pede tracking
196         for (int i = 0; i < 1000; i++) {
197             thisfile->readNextFrame();
198             pedestalObject->addFrameToPedestalCalculation(thisfile->getFrameDataHandle());
199         }
200         pedestalsG0->pedestalData(uint16_t* (pedestals16_G0_start));
201         pedestalsG2->pedestalData(pedeMS16_G0);
202         pedestalObject->pedestalResetUpdates();
203     }
204     else {
205         while (thisfile->readNextFrame()) {
206             pedestalObject->addFrameToPedestalCalculation(thisfile->getFrameDataHandle());
207         }
208         for (int i = 0; i < NCH; i++) {
209             if (pixel_mask[i] == true) {
210                 pedestalsG0->Fill(1/NC, 1/NC, pedestalObject->pedestalOfChannel(i));
211                 pedeMSG0->Fill(1/NC, 1/NC, pedestalObject->rmsOfChannel(i));
212             }
213             pedestalObject->pedestalData(uint16_t* (pedestals16_G0_start));
214             pedestalObject->pedestalData(pedeMS16_G0);
215             pedestalObject->pedestalResetUpdates();
216         }
217         thisfile->close();
218         mapcanvas->cd();
219         pedestalsG0->GetXaxis()->SetTitle("Column");
220         pedestalsG0->GetYaxis()->SetTitle("Row");
221         pedestalsG1->GetXaxis()->SetTitle("Column");
222         pedestalsG1->GetYaxis()->SetTitle("Row");
223         pedestalsG2->GetXaxis()->SetTitle("Column");
224         pedestalsG2->GetYaxis()->SetTitle("Row");
225         pedestalsG2->GetYaxis()->SetTitleOffset(0.7);
226         pedestalsG2->Draw("colz");
227         sprintf(savename, "plots/M%s/CuFluo/Ana/pedeG2_%.5s_%.5s.png", module_str.c_str(),
228             module_str.c_str(), "GB", "HGB");
229         mapcanvas->SaveAs((const char *)savename);
230     }
231
232     TH2I* adc2d_1 = new TH2I("adc2d_1", "", adc2d_nbin, -200-0.5, adc2d_nbin-200-0.5, 65);
233     TH2I* adc2d_2 = new TH2I("adc2d_2", "", adc2d_nbin, -200-0.5, adc2d_nbin-200-0.5, 65);
234     TH2I* adc2d_3 = new TH2I("adc2d_3", "", adc2d_nbin, -200-0.5, adc2d_nbin-200-0.5, 65);
235     TH2I* adc2d_4 = new TH2I("adc2d_4", "", adc2d_nbin, -200-0.5, adc2d_nbin-200-0.5, 65);
236     TH2I* adc2d_5 = new TH2I("adc2d_5", "", adc2d_nbin, -200-0.5, adc2d_nbin-200-0.5, 65);
237     TH2I* adc2d_6 = new TH2I("adc2d_6", "", adc2d_nbin, -200-0.5, adc2d_nbin-200-0.5, 65);
238     TH2I* adc2d_7 = new TH2I("adc2d_7", "", adc2d_nbin, -200-0.5, adc2d_nbin-200-0.5, 65);
239     TH2I* adc2d_8 = new TH2I("adc2d_8", "", adc2d_nbin, -200-0.5, adc2d_nbin-200-0.5, 65);
240
241     TH1D* adpcp_spec = new TH1D("adpcp_spec", "", 300, 0, 3000);
242
243     TH2F* pede_updates = new TH2F("pede_updates", "", NC, -0.5, NC-0.5, NR, -0.5, NR-0.5);
244     TH2F* pede_diff = new TH2F("pede_diff", "", NC, -0.5, NC-0.5, NR, -0.5, NR-0.5);
245
246     //Start looping over files -- loop over files in data_loc
247     // exit(1);
248     // file_n = 1;
249     for (int filei = 0; filei < file_n; filei++) {
250         // open data file
251         sprintf(savename, "%s/%s_V%6.6d.dat", data_loc.c_str(), data_file.c_str()); //
252         thisfile->open((char*)savename, filei);
253         while (thisfile->readNextFrame()) {
254             uint16_t* imagedptr = thisfile->getFrameDataHandle();
255             pedestalObject->addFrameToPedestalCalculationWithThreshold(imagedptr, pedestalObject);
256             for (int i = 0; i < NCH; i++) {
257                 uint16_t gain = (imagedptr[i] > 4096) ? 16 : 1;
258                 if (gain == 0) {
259                     int adpcp = (imagedptr[i] > 4096) ? 16 : 1;
260                     adpcp_spec->Fill(adpcp);
261                 }
262                 if (i < (65536*1)) {
263                     adc2d_1->Fill(adpcp, i);
264                 }
265                 else if (i < (65536*2)) {
266                     adc2d_2->Fill(adpcp, i);
267                 }
268                 else if (i < (65536*3)) {
269                     adc2d_3->Fill(adpcp, i);
270                 }
271                 else if (i < (65536*4)) {
272                     adc2d_4->Fill(adpcp, i);
273                 }
274                 else if (i < (65536*5)) {
275                     adc2d_5->Fill(adpcp, i);
276                 }
277                 else if (i < (65536*6)) {
278                     adc2d_6->Fill(adpcp, i);
279                 }
280                 else if (i < (65536*7)) {
281                     adc2d_7->Fill(adpcp, i);
282                 }
283                 else if (i < (65536*8)) {
284                     adc2d_8->Fill(adpcp, i);
285                 }
286             }
287         }
288     }
289 }
```

```
//Set up histograms
int adc2d_nbin = 1600;
TH2I* adc2d_1 = new TH2I("adc2d_1","",adc2d_nbin,-200-0.5,adc2d_nbin-200-0.5,65536,(65536*0-0.5),(65536*1-0.5));
TH2I* adc2d_2 = new TH2I("adc2d_2","",adc2d_nbin,-200-0.5,adc2d_nbin-200-0.5,65536,(65536*1-0.5),(65536*2-0.5));
TH2I* adc2d_3 = new TH2I("adc2d_3","",adc2d_nbin,-200-0.5,adc2d_nbin-200-0.5,65536,(65536*2-0.5),(65536*3-0.5));
TH2I* adc2d_4 = new TH2I("adc2d_4","",adc2d_nbin,-200-0.5,adc2d_nbin-200-0.5,65536,(65536*3-0.5),(65536*4-0.5));
TH2I* adc2d_5 = new TH2I("adc2d_5","",adc2d_nbin,-200-0.5,adc2d_nbin-200-0.5,65536,(65536*4-0.5),(65536*5-0.5));
TH2I* adc2d_6 = new TH2I("adc2d_6","",adc2d_nbin,-200-0.5,adc2d_nbin-200-0.5,65536,(65536*5-0.5),(65536*6-0.5));
TH2I* adc2d_7 = new TH2I("adc2d_7","",adc2d_nbin,-200-0.5,adc2d_nbin-200-0.5,65536,(65536*6-0.5),(65536*7-0.5));
TH2I* adc2d_8 = new TH2I("adc2d_8","",adc2d_nbin,-200-0.5,adc2d_nbin-200-0.5,65536,(65536*7-0.5),(65536*8-0.5));

auto start_time = open_data_file(thisfile);
for (size_t i = 0; i < n_frames && thisfile->readNextFrame(); i++) {
    uint16_t *imagedptr = thisfile->getFrameDataHandle();
    pedestalObject->addG0FrameToPedestalCalculationWThreshold(imagedptr, pedestalObject, pedeRMS16_G0);

    for (int i = 0; i < NCH; i++) {
        uint16_t gain = (imagedptr[i] & 0xc000) >> 14;
        if (gain == 0) {
            int adcpc = (imagedptr[i] & 0x3fff) - pedestalObject->pedestal0fChannel(i);

            if (i < (65536 * 1)) {
                adc2d_1->Fill(adcpc, i);
            } else if (i < (65536 * 2)) {
                adc2d_2->Fill(adcpc, i);
            } else if (i < (65536 * 3)) {
                adc2d_3->Fill(adcpc, i);
            } else if (i < (65536 * 4)) {
                adc2d_4->Fill(adcpc, i);
            } else if (i < (65536 * 5)) {
                adc2d_5->Fill(adcpc, i);
            } else if (i < (65536 * 6)) {
                adc2d_6->Fill(adcpc, i);
            } else if (i < (65536 * 7)) {
                adc2d_7->Fill(adcpc, i);
            } else if (i < (65536 * 8)) {
                adc2d_8->Fill(adcpc, i);
            }
        }
    }
}

report_duration(start_time, n_frames);
thisfile->close();
```

Processed 500 frames.

Elapsed time: 18.37 seconds. --> 27.23 FPS

Processing time for 220000 frames: 2h14m

Processed 500 frames.

Elapsed time: 2.18 seconds. --> 229.41 FPS

Processing time for 220000 frames: 0h15m

Why is the histogram filling so slow?

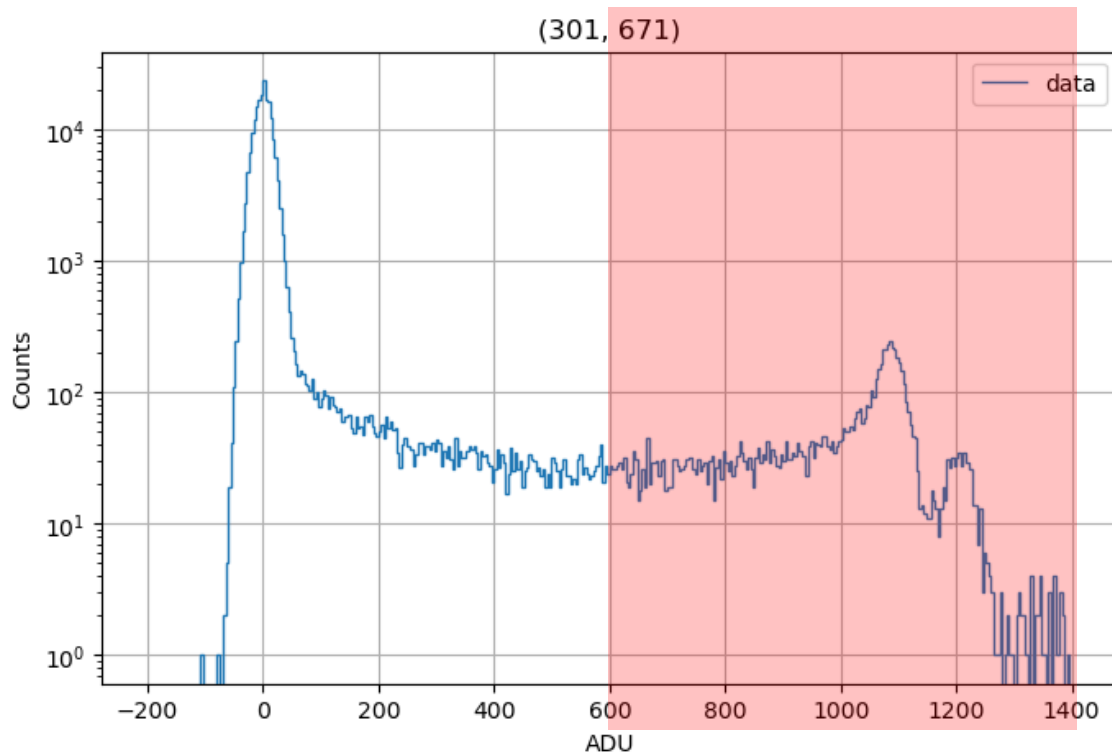
X-axis: 1600 bins (-200, → 1400)

Y-axis: 65535 bins

~3GB of data that we
index into randomly!

What can we do?

- `TH1D *proj_peak = dynamic_cast<TH1D*>(proj->Rebin(4,"proj_peak"));` (4x)
- Interesting region 600-1400 (2x)
- Maximum counts $\sim 300 \rightarrow$ Int32 to int16 as storage type (2x)



$\rightarrow$ 16x smaller histogram: $\sim 200\text{MB}$

Does it affect runtime?

```
int adc2d_nbin= 200;
TH2S* adc2d_1 = new TH2S("adc2d_1","",adc2d_nbin,600.5,1400.5 ,65536,(65536*0-0.5),(65536*1-0.5));
TH2S* adc2d_2 = new TH2S("adc2d_2","",adc2d_nbin,600.5,1400.5 ,65536,(65536*1-0.5),(65536*2-0.5));
TH2S* adc2d_3 = new TH2S("adc2d_3","",adc2d_nbin,600.5,1400.5 ,65536,(65536*2-0.5),(65536*3-0.5));
TH2S* adc2d_4 = new TH2S("adc2d_4","",adc2d_nbin,600.5,1400.5 ,65536,(65536*3-0.5),(65536*4-0.5));
TH2S* adc2d_5 = new TH2S("adc2d_5","",adc2d_nbin,600.5,1400.5 ,65536,(65536*4-0.5),(65536*5-0.5));
TH2S* adc2d_6 = new TH2S("adc2d_6","",adc2d_nbin,600.5,1400.5 ,65536,(65536*5-0.5),(65536*6-0.5));
TH2S* adc2d_7 = new TH2S("adc2d_7","",adc2d_nbin,600.5,1400.5 ,65536,(65536*6-0.5),(65536*7-0.5));
TH2S* adc2d_8 = new TH2S("adc2d_8","",adc2d_nbin,600.5,1400.5 ,65536,(65536*7-0.5),(65536*8-0.5));

auto start_time = open_data_file(thisfile);
for (size_t i = 0; i < n_frames && thisfile->readNextFrame(); i++) {
    uint16_t *imagedptr = thisfile->getFrameDataHandle();
    pedestalObject->addG0FrameToPedestalCalculationWithThreshold(imagedptr, pedestalObject, pedeRMS16_G0);

    for (int i = 0; i < NCH; i++) {
        uint16_t gain = (imagedptr[i] & 0xc000) >> 14;
        if (gain == 0) {
            int adcpc = (imagedptr[i] & 0x3fff) - pedestalObject->pedestal0fChannel(i);

            if (i < (65536 * 1)) {
                adc2d_1->Fill(adcpc, i);
            } else if (i < (65536 * 2)) {
                adc2d_2->Fill(adcpc, i);
            } else if (i < (65536 * 3)) {
                adc2d_3->Fill(adcpc, i);
            } else if (i < (65536 * 4)) {
                adc2d_4->Fill(adcpc, i);
            } else if (i < (65536 * 5)) {
                adc2d_5->Fill(adcpc, i);
            } else if (i < (65536 * 6)) {
                adc2d_6->Fill(adcpc, i);
            } else if (i < (65536 * 7)) {
                adc2d_7->Fill(adcpc, i);
            } else if (i < (65536 * 8)) {
                adc2d_8->Fill(adcpc, i);
            }
        }
    }
}
report_duration(start_time, n_frames);
thisfile->close();
}
```

Processed 500 frames.

Elapsed time: 5.93 seconds. --> 84.38 FPS

Processing time for 220000 frames: 0h43m

Is there more we can do?

- TH2 does not allow you to control axis types or the presence of under/overflow
- We know we have equally spaced bins.
- Unnecessary (for us) work on fill:

```
Int_t TH1::Fill(Double_t x, Double_t w)
{
    if (fBuffer) return BufferFill(x,w);

    Int_t bin;
    fEntries++;
    bin = fXaxis.FindBin(x);
    if (bin < 0) return -1;
    if (!fSumw2.fN && w != 1.0 && !TestBit(TH1::kIsNotW) ) Sumw2();
    if (fSumw2.fN) fSumw2.fArray[bin] += w*w;
    AddBinContent(bin, w);
    if (bin == 0 || bin > fXaxis.GetNbins()) {
        if (!GetStatOverflowsBehaviour()) return -1;
    }
    Double_t z = w;
    fTsumw += z;
    fTsumw2 += z*z;
    fTsumwx += z*x;
    fTsumwx2 += z*x*x;
    return bin;
}
```

Alternatives?



Let's roll our own histogram class!

```
template <typename T, typename StorageType> class PixelHistogramImpl
{
    static_assert(std::is_floating_point_v<T>,
        "PixelHistogramImpl requires a floating-point axis type");
    NDArray<StorageType, 3> m_values;
    NDArray<T, 1> m_edges;
    int m_rows;
    int m_cols;
    int m_n_bins;
    T m_xmin;
    T m_xmax;
    // n_bins / (xmax - xmin), precomputed to keep the hot path
    // division-free.
    T m_scale;

public:
    PixelHistogramImpl(int rows, int cols, int n_bins, T xmin, T xmax)
    {
        m_rows = rows;
        m_cols = cols;
        m_n_bins = n_bins;
        m_xmin = xmin;
        m_xmax = xmax;
        m_scale = (T)(n_bins / (xmax - xmin));
    }

    void fill(const NDView<T, 2> &frame);
    void fill(int row, int col, T value);
    void fill_unchecked(int row, int col, T value);
    // Fill using a row-major pixel index, avoiding repeated 3D
}
```

Templated and axis types
~150 lines of C++

How does it perform



```
aare::PixelHistogramImpl<float, uint16_t> hist(524288, 1, 200, 600.5, 1400.5);  
auto start_time = open_data_file(thisfile);  
for (size_t i = 0; i < n_frames && thisfile->readNextFrame(); i++) {  
  
    uint16_t *imagedptr = thisfile->getFrameDataHandle();  
    pedestalObject->addG0FrameToPedestalCalculationWThreshold(imagedptr, pedestalObject, pedeRMS16_G0);  
  
    for (int i = 0; i < NCH; i++) {  
        uint16_t gain = (imagedptr[i] & 0xc000) >> 14;  
  
        if (gain == 0) {  
            int adcpc = (imagedptr[i] & 0x3fff) - pedestalObject->pedestalOfChannel(i);  
            hist.fill(i, 0, adcpc); // Fill the histogram for pixel i with the pedestal-corrected ADC value  
        }  
    }  
}  
  
report_duration(start_time, n_frames);
```

Processed 500 frames.
Elapsed time: 3.28 seconds. --> 152.58 FPS
Processing time for 220000 frames: 0h24m

Of which 15min is pedestal
tracking!

New pedestal



- Not tracking number of samples per pixel
- Contiguous arrays etc.

```
for (int frame_index = 0; frame_index < n_frames; ++frame_index) {
    fluorescence_file.read_into(&image);
    uint16_t* imagedptr = image.data();

    for (int i = 0; i < NCH; i++) {
        uint16_t gain = (imagedptr[i]&0xc000) >> 14;

        if (gain == 0) {
            const uint16_t adc = imagedptr[i] & 0x3fff;
            if (std::abs(adc - pedestal.mean(i)) < pedestal_rms[i]) {
                pedestal.push_fast(i, adc);
            }
            int adpc = adc - pedestal.mean(i);
            hist.fill(i, 0, adpc); // Fill the histogram for pixel i with the pedest
        }
    }
}
```

Pedestal only

Processed 5000 frames.

Elapsed time: 12.9623 seconds. --> 385.734 FPS

Processing time for 220000 frames: 0h9m

Pedestal and histogramming

Processed 5000 frames.

Elapsed time: 21.9307 seconds. --> 227.99 FPS

Processing time for 220000 frames: 0h16m

Next obvious thing: parallel



```
for (int frame_index = 0; frame_index < n_frames; ++frame_index) {  
    fluorescence_file.read_into(&image);  
    uint16_t* imagedptr = image.data();  
  
    #pragma omp parallel for  
    for (int i = 0; i < NCH; i++) {  
        uint16_t gain = (imagedptr[i] & 0xc000) >> 14;  
  
        if (gain == 0) {  
            const uint16_t adc = imagedptr[i] & 0x3fff;  
            if (std::abs(adc - pedestal.mean(i)) < pedestal_rms[i]) {  
                pedestal.push_fast(i, adc);  
            }  
            int adcpc = adc - pedestal.mean(i);  
            hist.fill(i, 0, adcpc); // Fill the histogram for pixel i with the pedestal-corr  
        }  
    }  
}
```

Processed 5000 frames.

Elapsed time: 2.95348 seconds. --> 1692.92 FPS

Processing time for 220000 frames: 0h2m

This is clearly useful let's build it properly



- Use out data analysis library (aare) to read files (multi threaded)
- Use the improved pedestal class
- Our own histogram class
- Use standalone Minuit2 for fitting (with implementation agnostic API)
- Pipeline where read happens in parallel with histogramming
- Python bindings using pybind11
- CI/CD
- Pypi + conda pkgs
- Online documentation

Reding from python

```
with aare.Timer(verbose = False) as t:
    with aare.File(data_fname) as f:
        for i in tqdm(range(N)):
            frame = f.read_frame()
            ph.fill_async(frame)
```

Processed 10000 frames
Elapsed time: 2.75 seconds --> 3639 FPS
Processing time for 220000 frames: 1m 0.46s

MT reads from C++

```
with aare.Timer(verbose = False) as t:
    ph.fill_from_file(
        data_fname,
        max_frames = N,
        verbose = True,
        reader_threads = 4,
        reader_chunk_size = 512
    )
```

Processed 20000 frames
Elapsed time: 3.94 seconds --> 5073 FPS
Processing time for 220000 frames: 0m 43.37s

Swapping processing order

for frame in data:

 for row in frame:

 for pixel in row:

 update pedestal

 fill histogram

for batch in data: //tens of frames

 for row in range(n_rows):

 for col in range(n_cols):

 for frame in range(n_frames)

 update pedestal

 fill histogram

Processed 50000 frames

Elapsed time: 7.14 seconds --> 6999 FPS

Processing time for 220000 frames: 0m 31.43s

- Total work ~2 weeks (leveraging our existing library)
- Histogram filling + fitting runs ~200x faster
 - ~90x in practice limited by read from NFS share
- Removed ROOT dependency (using internal Minuit2)
- Python bindings with pybind11
- CI/CD pipeline (from aare)
 - Conda/pip
 - Docs as github pages
- Data analysis library on github: <https://github.com/slsdetectorgroup/aare>

- Processing on cluster not foreseen but would require new way of pedestal tracking that works on a chunk of frames (fit gaus or double process)
- Python package to drive calibration
- CI/CD + documentation for the specialized package

Thank you!

