Current and upcoming challenges for data packaging of DECTRIS X-ray detectors

Max Burian – max.burian@dectris.com
Diego Gämperle – diego.gaemperle@dectris.com

2023-09-19

Table of contents

Performance stats

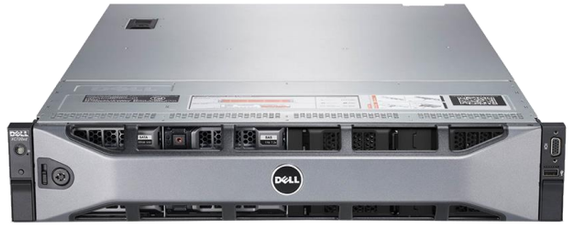
Frame rate (cont.): 560 Hz
Image size: 16.77 Mpixel (w/o gaps)
Data rate in: 160 Gbps
Data rate out Filewriter (max): ~35 Gbps
Data rate out Stream (max) : ~75 Gbps

EIGER2 XE 16M

Frame rate (cont.): 120 kHz
Image size: 31 kpixel (w/o gaps)
Data rate in: 10 Gbps
Data rate out (max): 10 Gbps

ARINA

SW architecture



Detector Control Unit (DCU)

fedora

app-detector

- APIs for detector control
- Hardware drivers
- Data processing

app-calibration

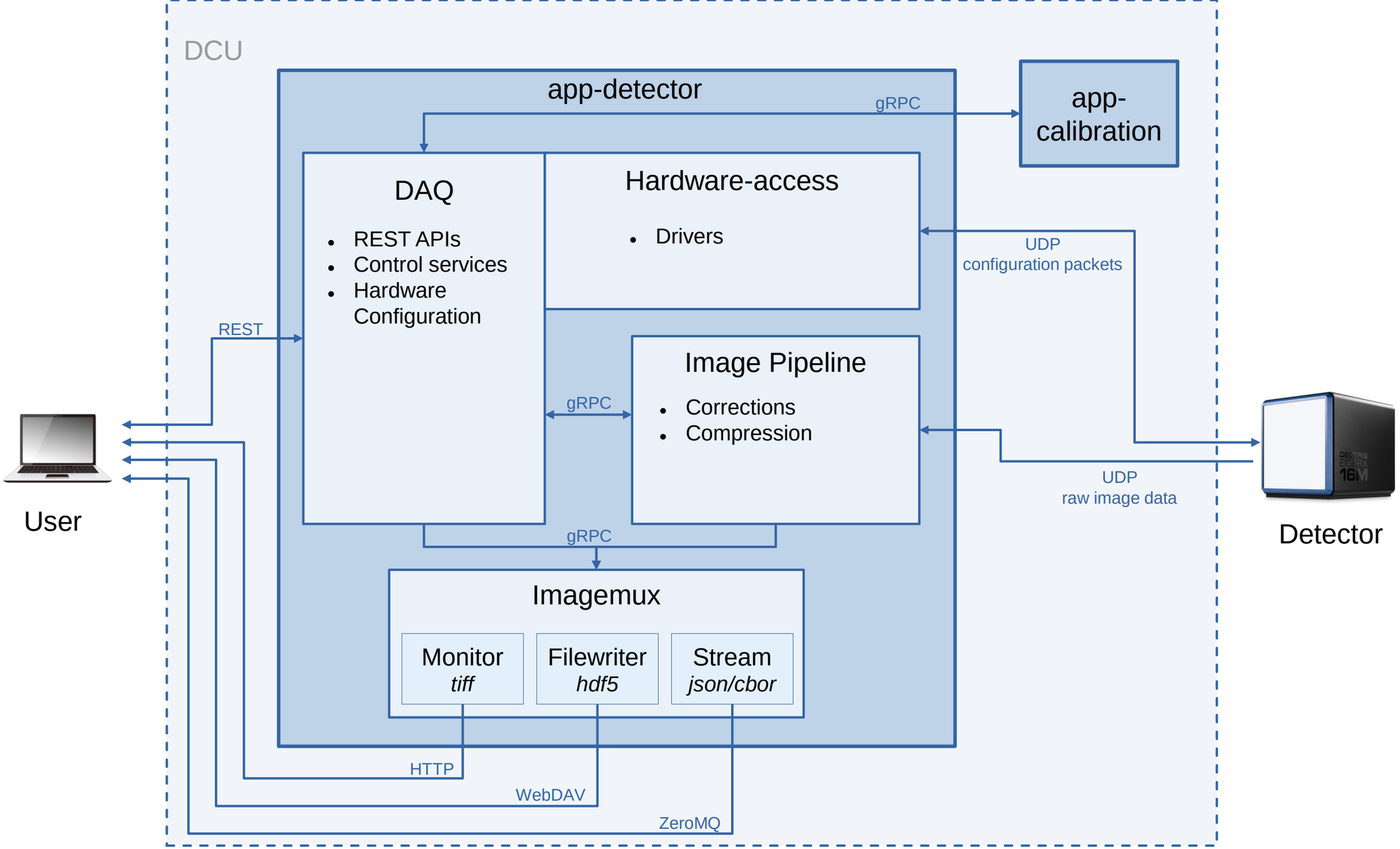
- Module calibrations

Base System

- Web-UI
- Manage apps
- Server config

User

Detector

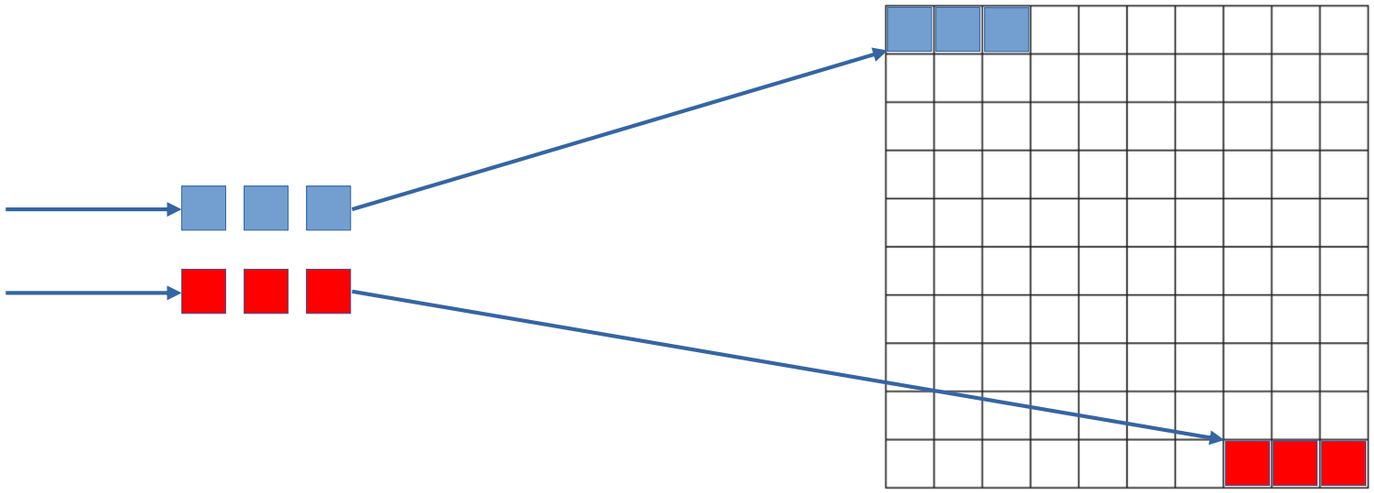

DCU
app-detector
gRPC
app-
calibration
DAQ
REST APIs
Control services
Hardware
Configuration
Hardware-access
Drivers
UDP
configuration packets
REST
Image Pipeline
Corrections
Compression
gRPC
UDP
raw image data
User
Detector
gRPC
Imagemux
Monitor
tiff
Filewriter
hdf5
Stream
json/cbor
HTTP
WebDAV
ZeroMQ

Data handling

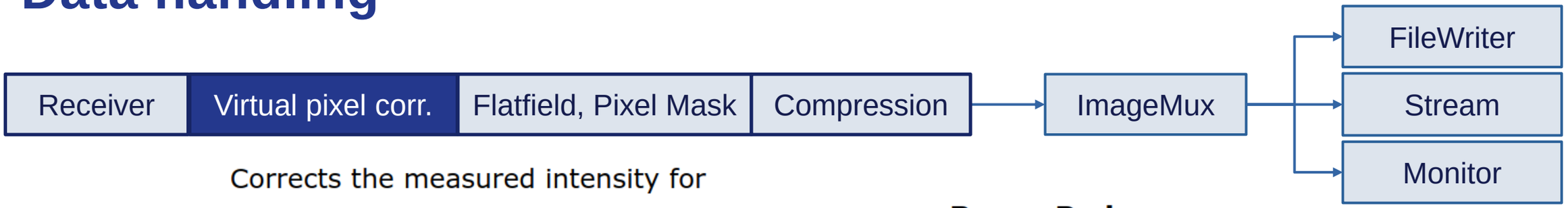
Modules
send UDP packets
with pixel intensities

Pixels values get:

1. countrate corrected *[if correction enabled]*
2. placed at the final position in image
3. summed *[if auto_summation is enabled]*

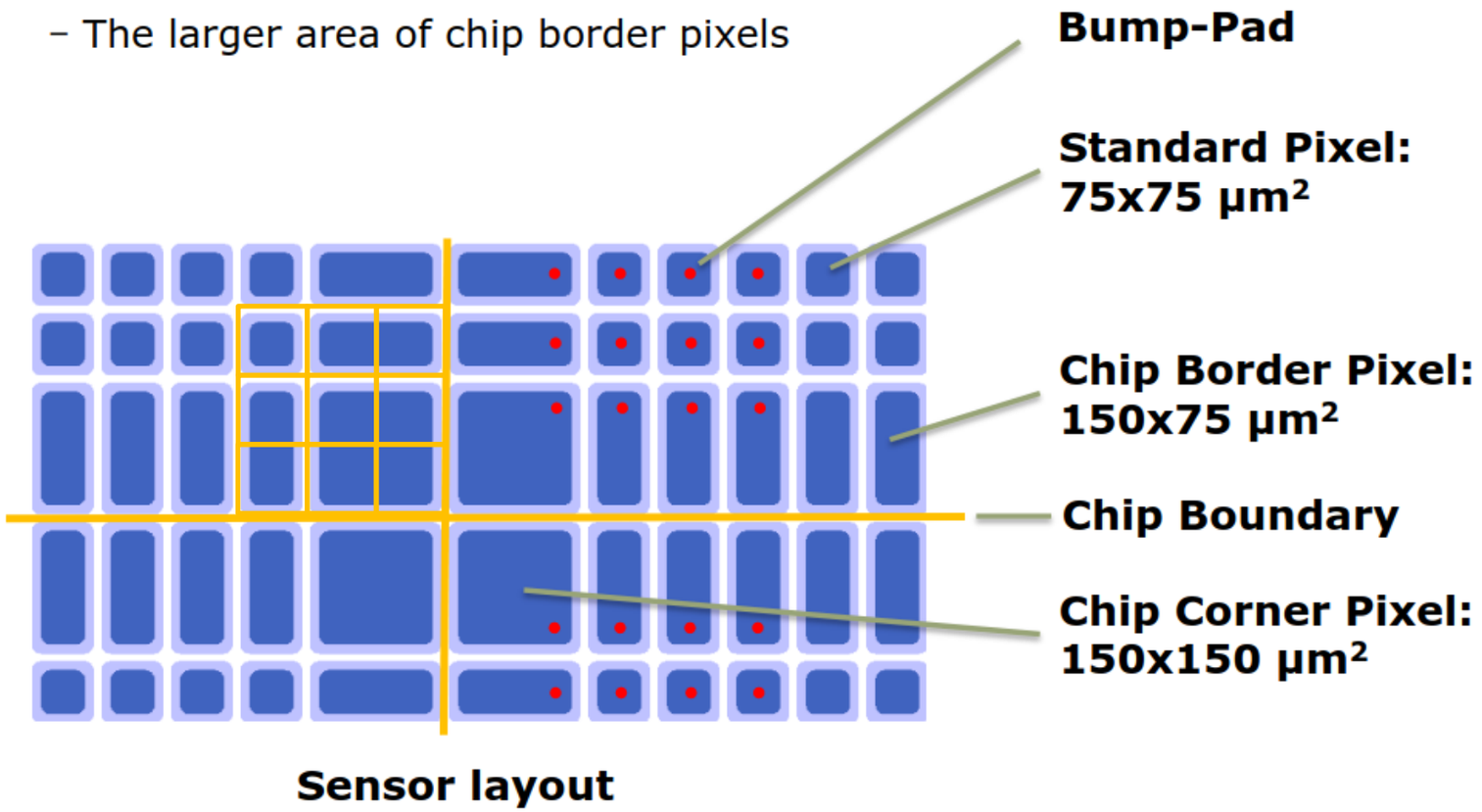


Data handling

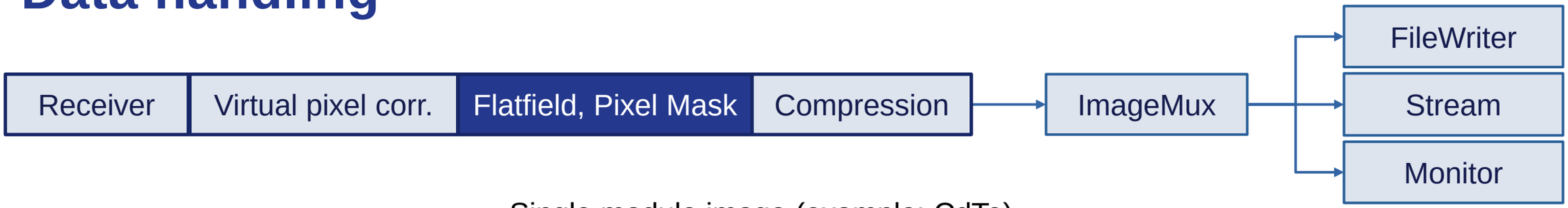


Corrects the measured intensity for

- The larger area of chip border pixels

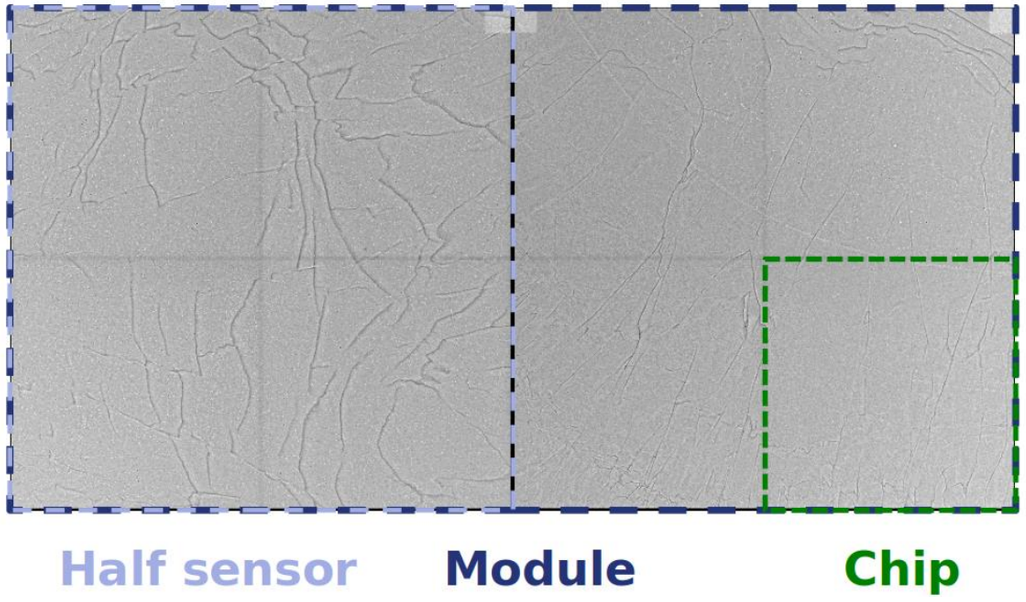


Data handling

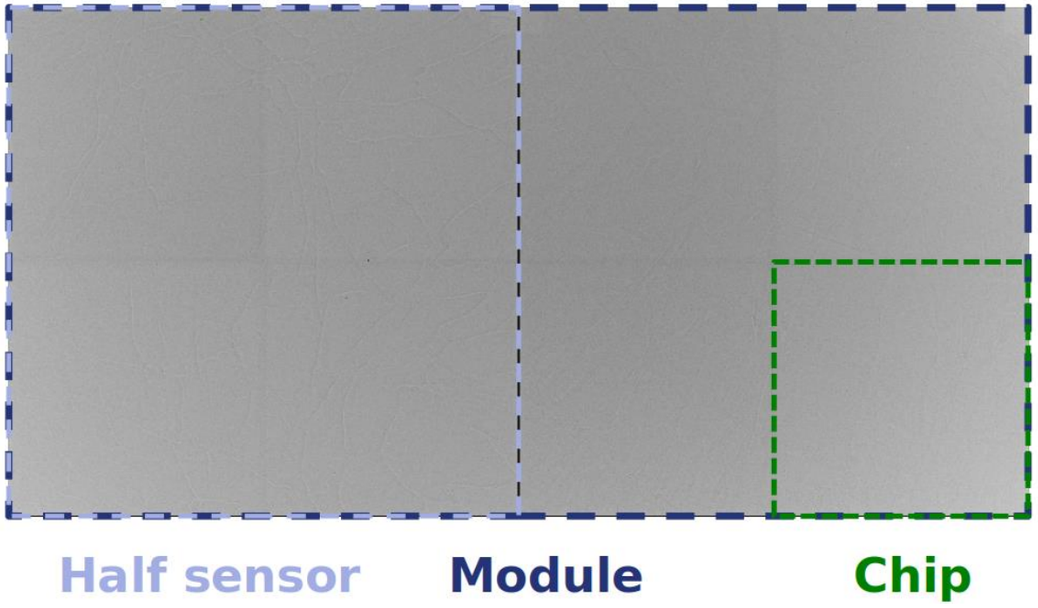


Single module image (example: CdTe)

Without flatfield



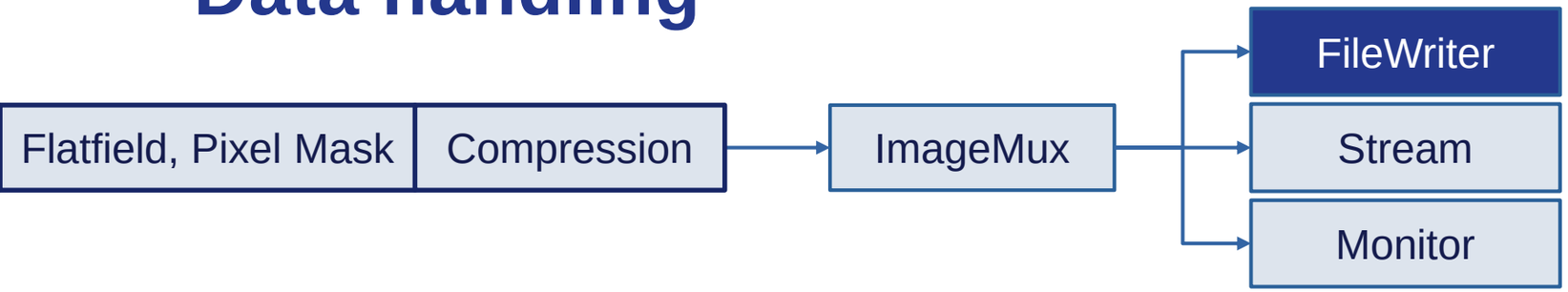
With flatfield



Data handling

- Finally the images are compressed with either “bitshuffle lz4” or “lz4.
- Afterwards they are available in “imagemux” which provides customer facing interfaces:
  - Monitor
  - Filewriter
  - Stream

Data handling



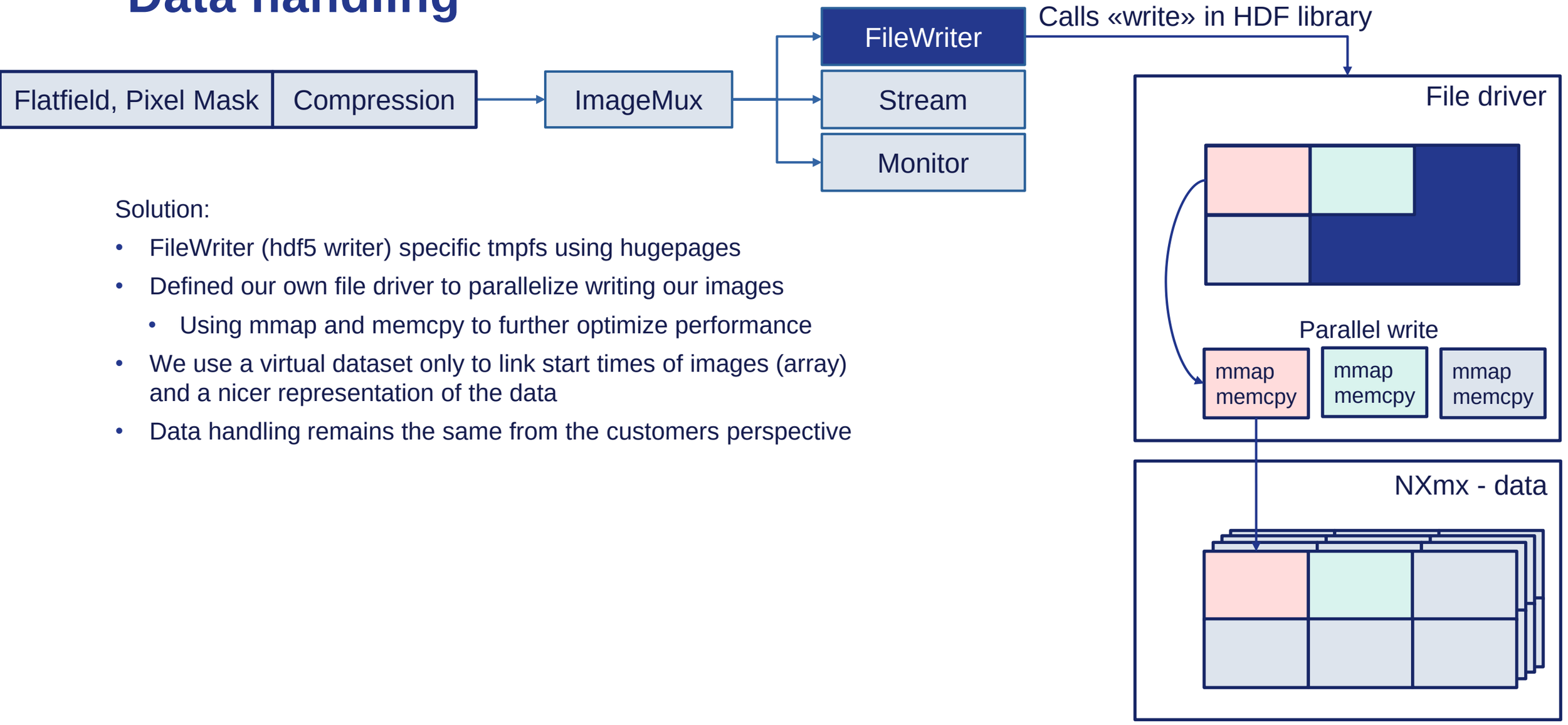
Start situation:

- Image pipeline max load ~ 75 Gbps compressed data
- Realistic compression average ~25% -> ~ 40 Gbps with current detectors
- Data always stays in RAM
- Standard HDF writer not fast enough

Issues:

- Virtual dataset not being used optimally due to downside in post-processing (more performance required)
- No possibility to parallelize writing into single data file

Data handling

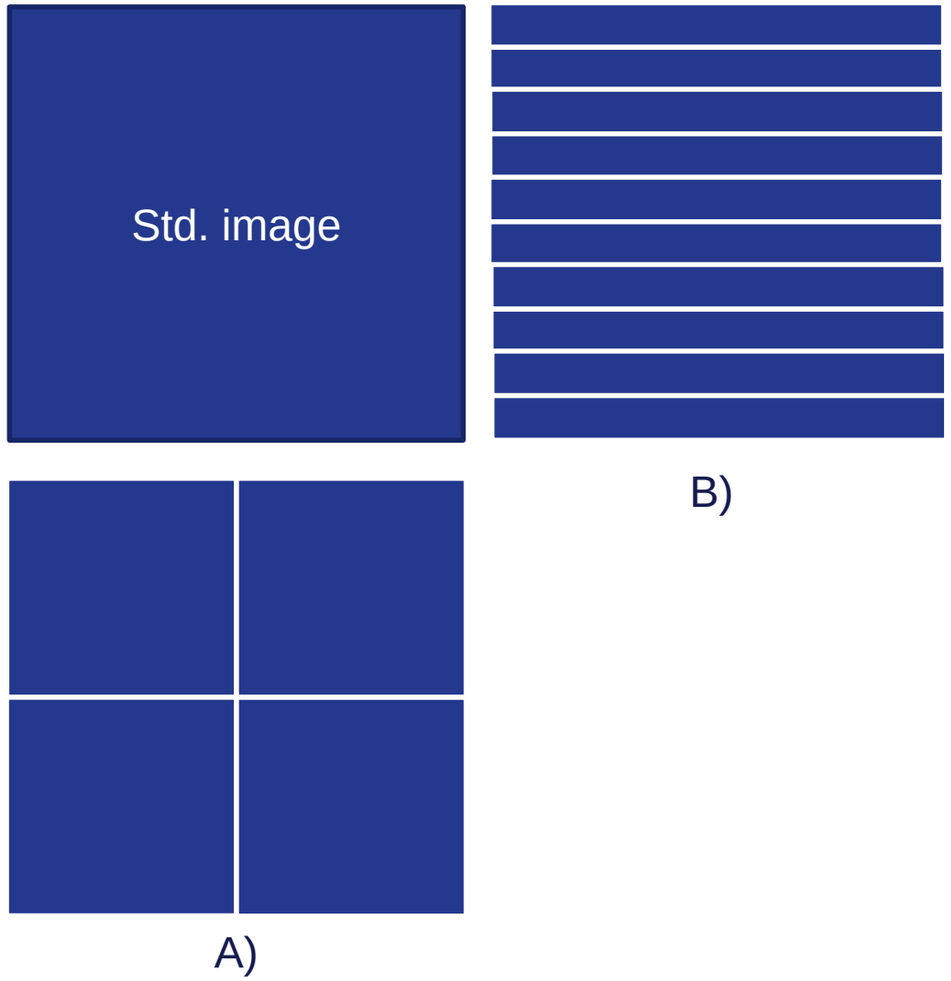


Solution:

- FileWriter (hdf5 writer) specific tmpfs using hugepages
- Defined our own file driver to parallelize writing our images
  - Using mmap and memcpy to further optimize performance
- We use a virtual dataset only to link start times of images (array) and a nicer representation of the data
- Data handling remains the same from the customers perspective

HDF – future challenges

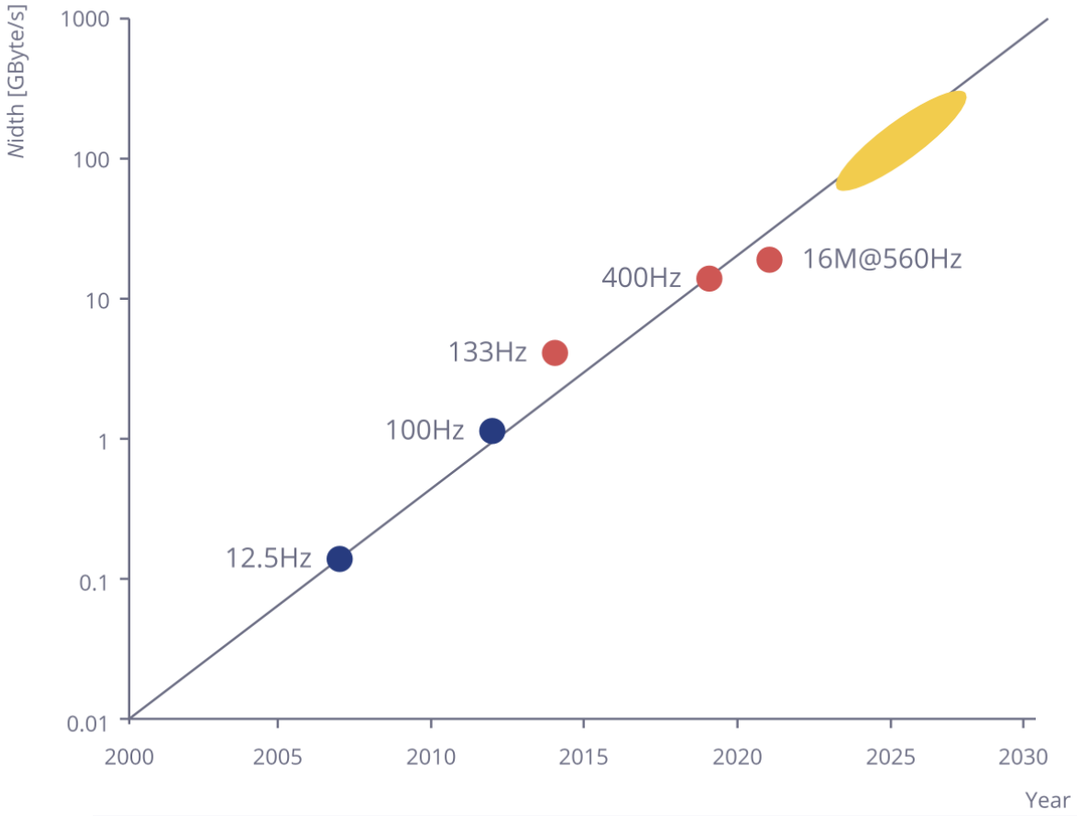
- Virtual datasets don’t automatically solve the parallel writing issue
  - A) Store data module / quadrant wise
  - B) Store data in a row representation
- Post-processing pipelines highly dependable on how the data is written
- Raw data rate increase factor >5
  - How do we write ~200 Gbps of compressed data using HDF?
- Maintenance of specific solution undesired



Evolution of Data Rates

PILATUS

2007: 0.14 Gbyte/s of Raw Data



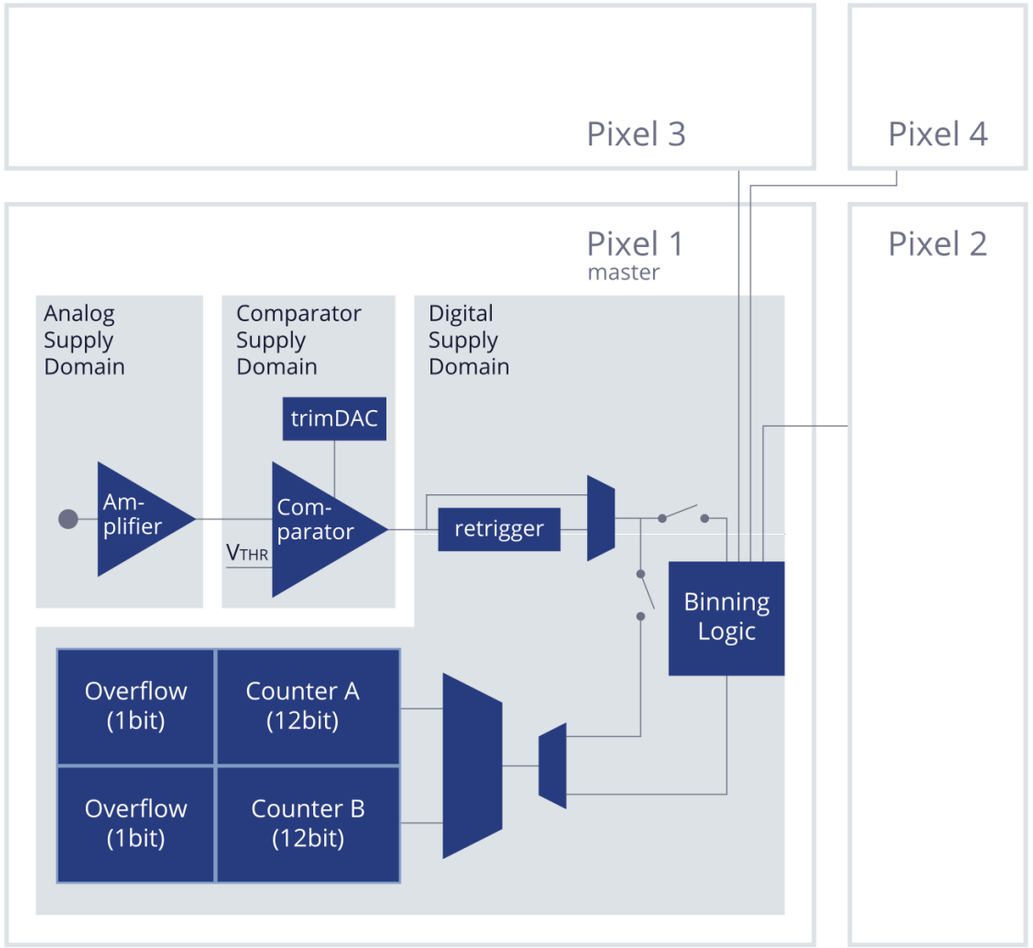
Prototype detectors in development (Jungfrau, Citius, …)

EIGER2/PILATUS4

2022: 18 Gbyte/s of Raw Data

Moore's Law for X-Ray detectors

Example: Fast ASIC development



Pixel size
100 µm × 100 µm

Pixel array
192 × 192 pixels

Readout modes
Full frame / 2 × 2 pixel digital binning

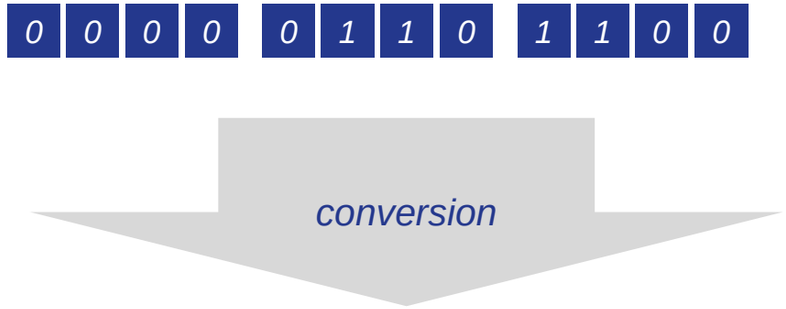
Counter encoding
12-bit integer / 8-bit floating-point

Instant Retrigger technology &
Room temperature operation

8-bit Floating-Point Compression

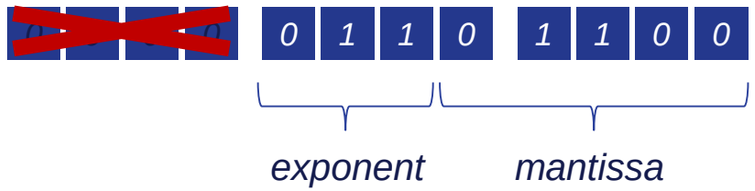
Standard 12 Bit UINT counter:

Range: 4’095 Cnts
Precision: 1 Cnt

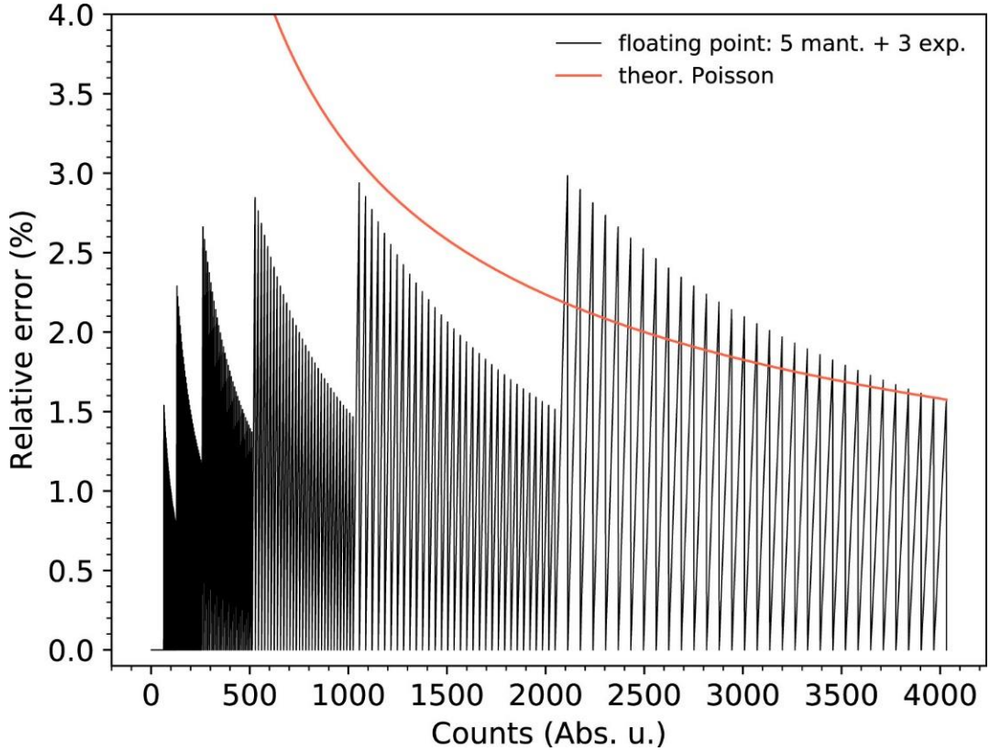


8 Bit Floating-Point counter:

Range: 4’032 Cnts
Precision: depends on signal



Floating Point Rounding Error

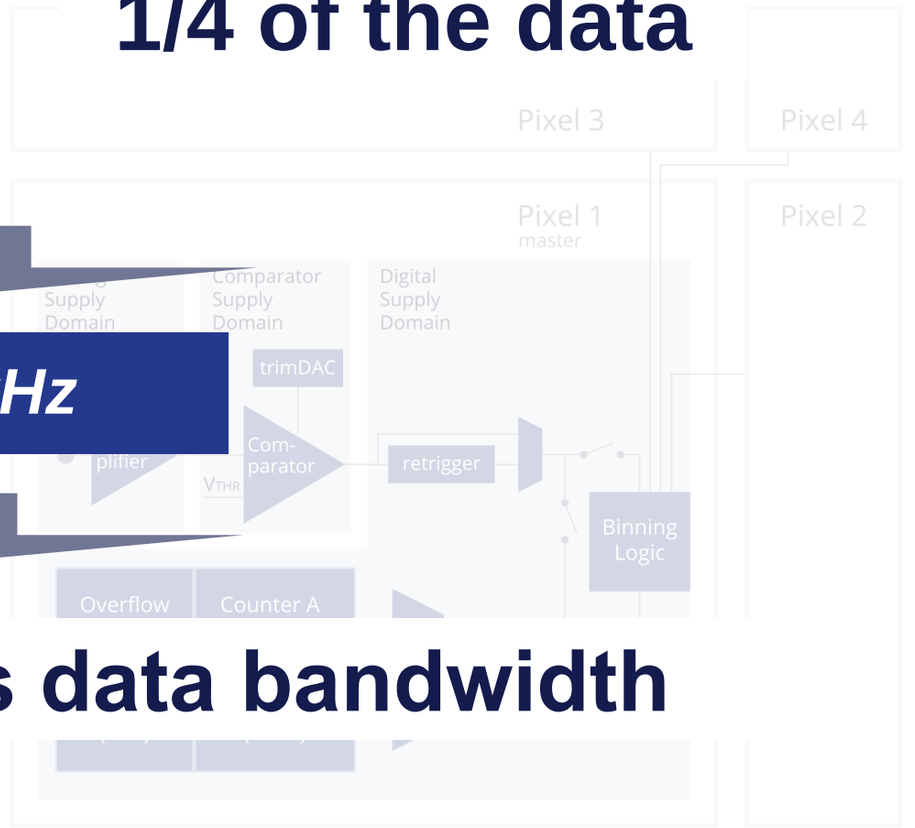


See P.Zambon, et.al. (2023), doi:10.1016/j.nima.2022.167888

FP Compression & Binning

33% bandwidth reduction

1/4 of the data



Framerate 120 kHz

less data in the pixel ≠ less data bandwidth

Recent: BitShuffle Compression

New, highly optimized BitShuffle implementation for modern processors by Kal Cuttler (DECTRIS)

- Implemented in C99
- Optimized with SIMD instructions including SSE2, AVX2, and AVX-512
- Does not allocate memory
- Support for all major compilers and operating systems

Typical **speedup** between **1.3x and 10x** depending on the CPU architecture, function, and arguments

see github.com/dectris/compression/

Test by John Wright, ESRF: **speedup up to >40%** for the entire "bslz4" decode operation

see https://github.com/jonwright/bslz4_to_sparse/issues/1

Edge Computing

Collaboration Project:

Started in April 2023

Radial integration, spot positions, indexing

Aim: no (millisecond) latency
for EIGER2 & PILATUS4

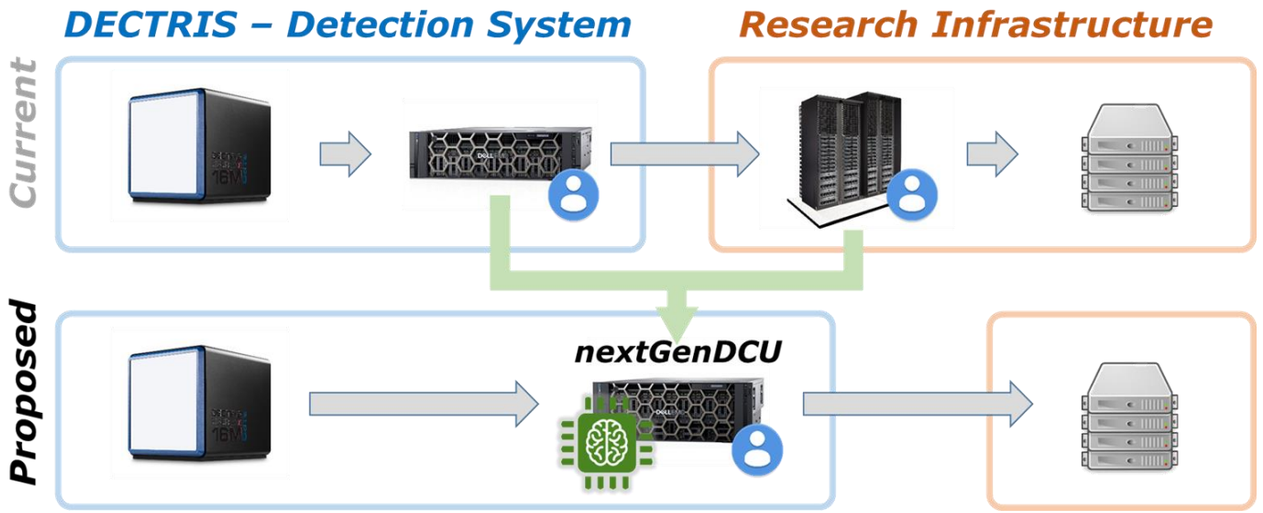
Innosuisse

PAUL SCHERRER INSTITUT

PSI

DECTRIS

Experiment requires live feedback



MX Image Scoring

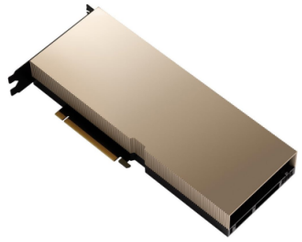
Curtesy of Filip LEONARSKI (PSI, SLS)

on CPU

Spot finding
Peakfinder8: 25-50 ms/image

Indexing
Xgandalf (fast): 100 ms/image

on GPU/FPGA



Spot finding
COLSPOT on GPU: < 200 µs

Indexing
Fast-feedback indexer on GPU: < 1 ms

Image scoring is faster than detector framerate

Future Challenges

- **Scientists always push the limit**
  - Data bandwidth will always be used
  - Data reduction upstream will lead to faster framerates
- **Bslz4 is great, but we need more**
  - Community will have to embrace lossy compression algorithms
- **Compression requires de-compression**
  - FAIR principles are key
  - de-compression overhead and integration are crucial for usability
- **Edge computing will become more**
  - Data should be transformed into actionable insight as soon as possible
  - «Process data where it is generated» -> enable data filtration
- **It’s an «infrastructure» problem**
  - Detectors can easily do more
  - But storage, archiving, transfer, sharing, etc.. become paralyzed

Thank you for your attention!

DECTRIS

5405 Baden-Dättwil

Switzerland

www.dectris.com