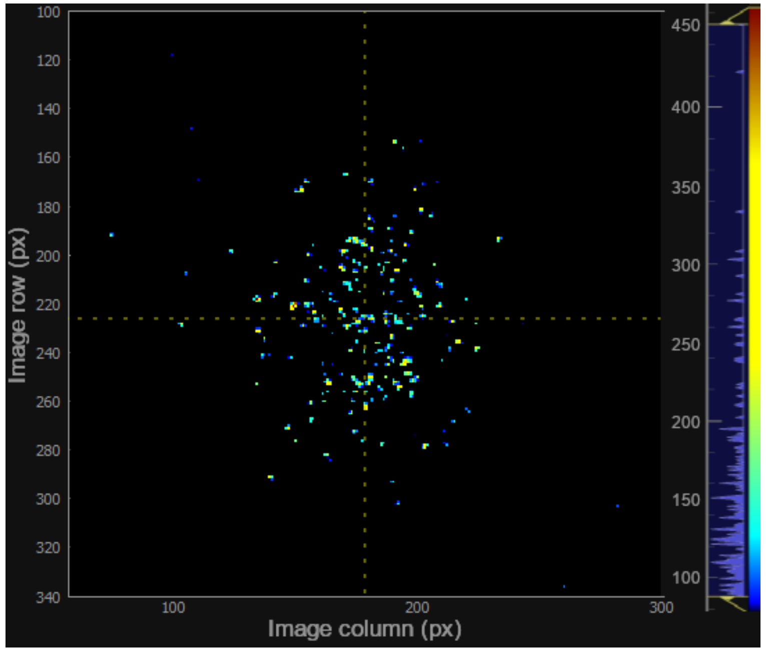
Run-length encoding for mostly black images

Erik Månsson
CFEL-ATTO group
DESY, Hamburg, Germany



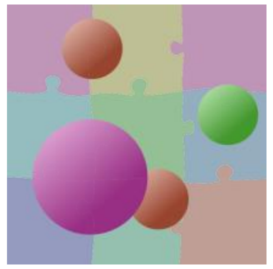
Outline

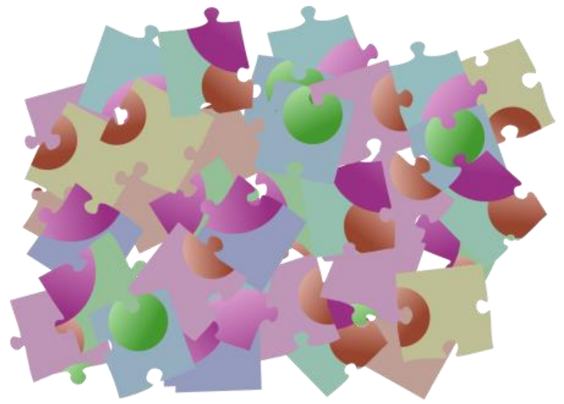
- Scientific background
- The data
- The compression
  - ➢ Seems similar to sparse paradigm described by Elena Pourmal for SLAC-II
- Performance benchmarks so far
- Discussion points

Scientific background

Multiple pieces of the molecular puzzle

Scientific case 1

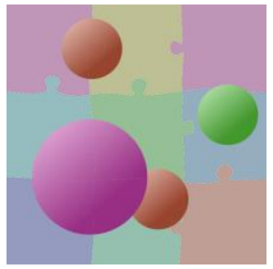


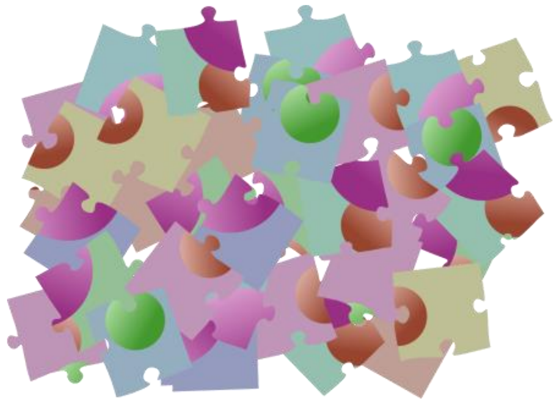


- A doubly ionized molecule often breaks into two charged fragments.
- What combinations of (m_1, m_2) occur, and how often?
  - This says something about the bonds in the molecule, and how the photochemical reaction ends.
- Does $m_1 + m_2$ equal the original molecule's mass?
  - Or were there undetected fragments?
- You can not answer these questions from a simple mass spectrum
  - It only tells the *average* amount of m_1 and the *average* amount of m_2.
  - Not which of them come *together*.
- With an imaging detector, you can get additional coordinates (velocity or momentum) for each fragment.

Multiple pieces of the molecular puzzle

Scientific case 2

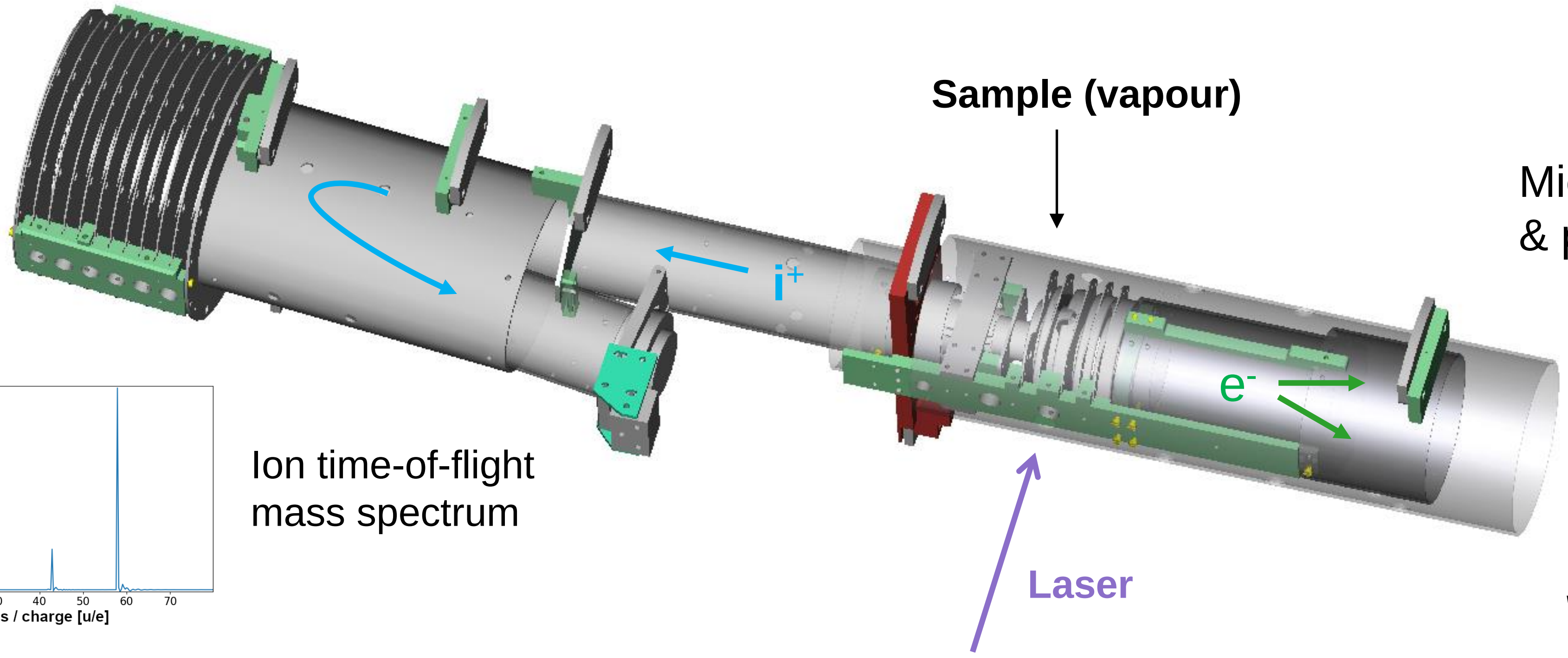


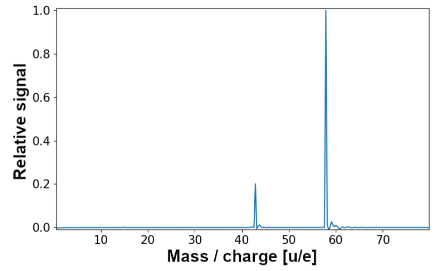


- Already single ionization gives two particles we can study: one electron and one ion.
  - Electron spectrum: state populated by the photoionization, “initial condition” for dynamics in the molecular ion.
  - Ion mass: Did the molecule break? Which bond(s)? Which side of the broken bond has the electron-hole?

- What combinations of electron energy & ion mass occur, and how often?
  - Linking which state leads to which fragmentation outcome
    offers insight into the “landscape” of potential energy curves.
  - Can help improving theoretical/computational models of ultrafast molecular dynamics.

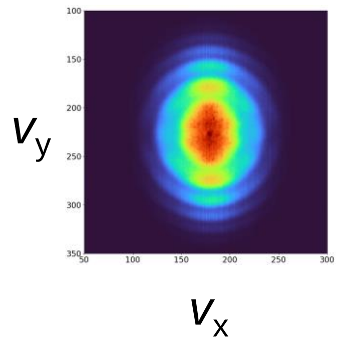
How do we record the pieces?





Ion time-of-flight mass spectrum

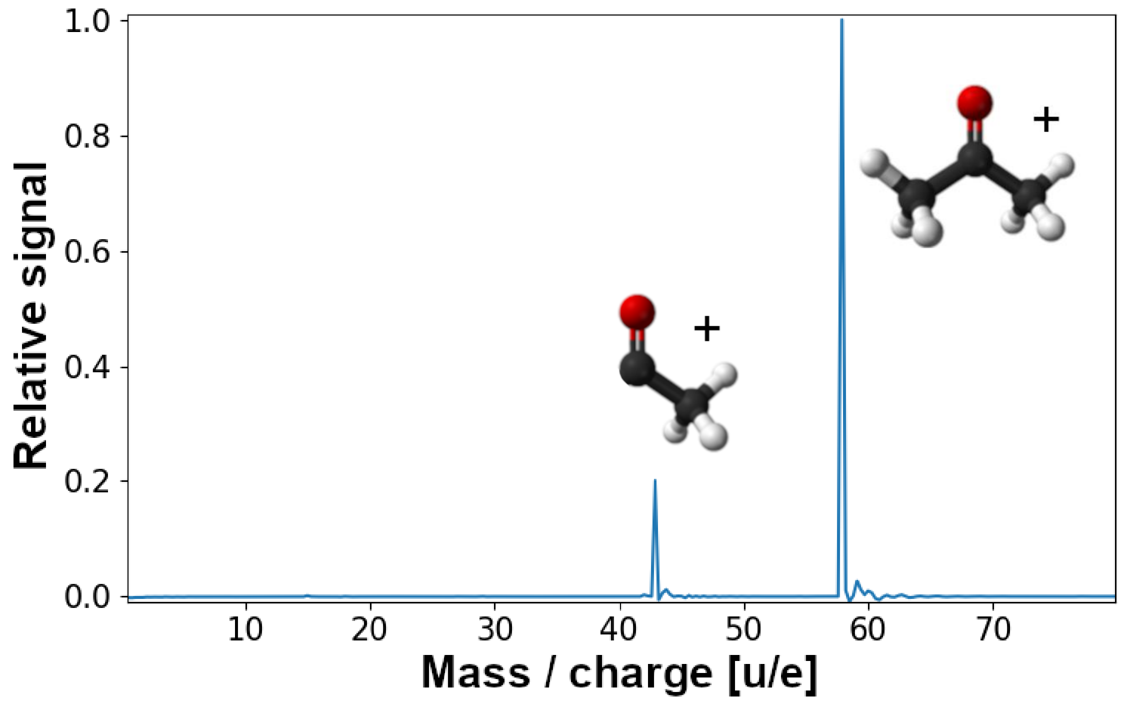
Electron 2D velocity map image

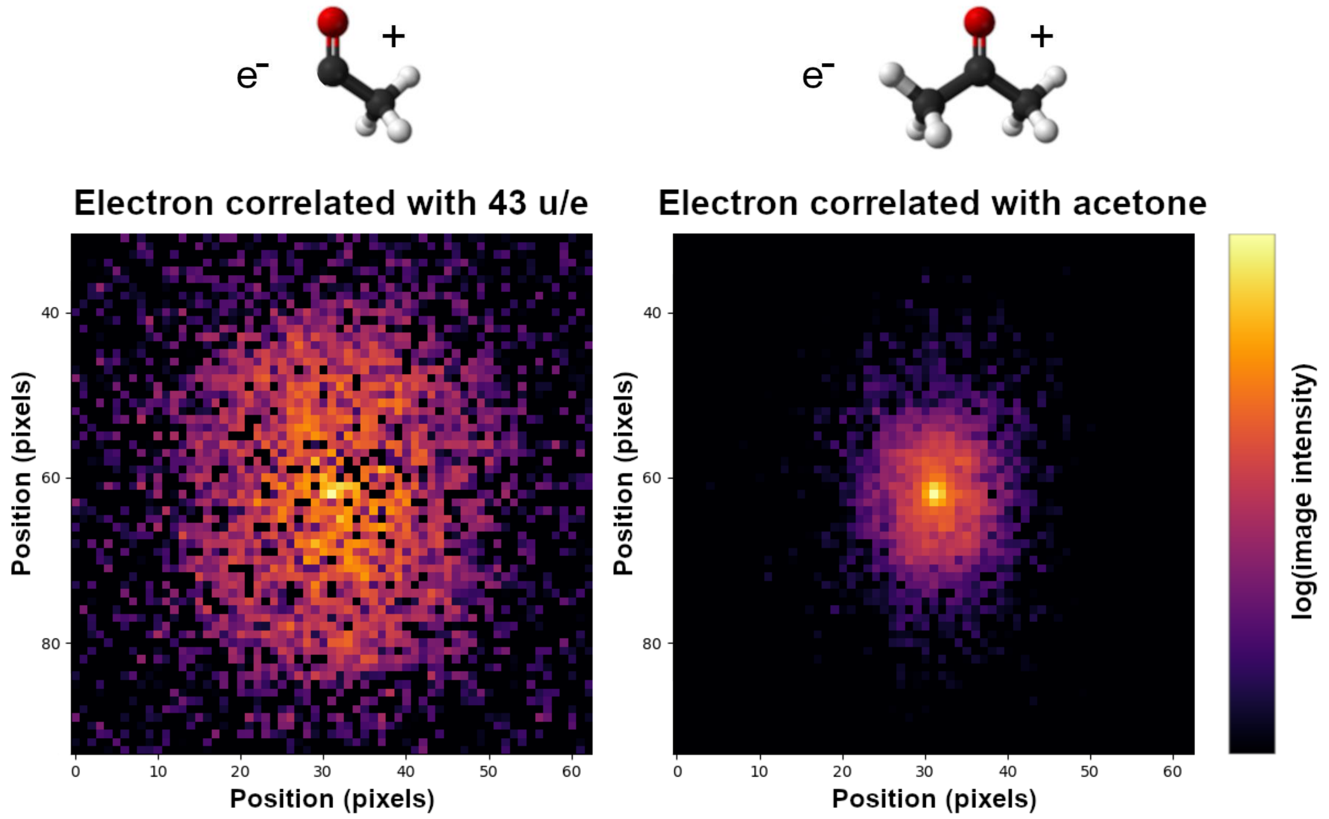


(angularly-resolved electron spectra)

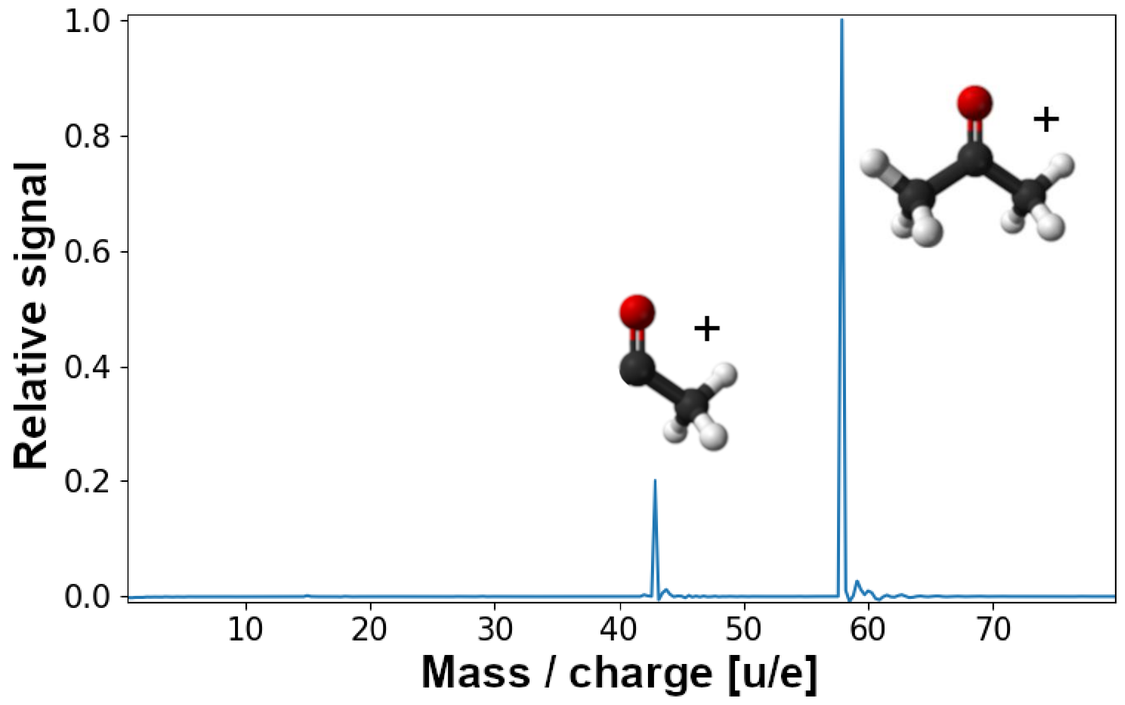
E. Månsson, *et al.* *EPJ Web of Conferences* **205** (2018) 03007

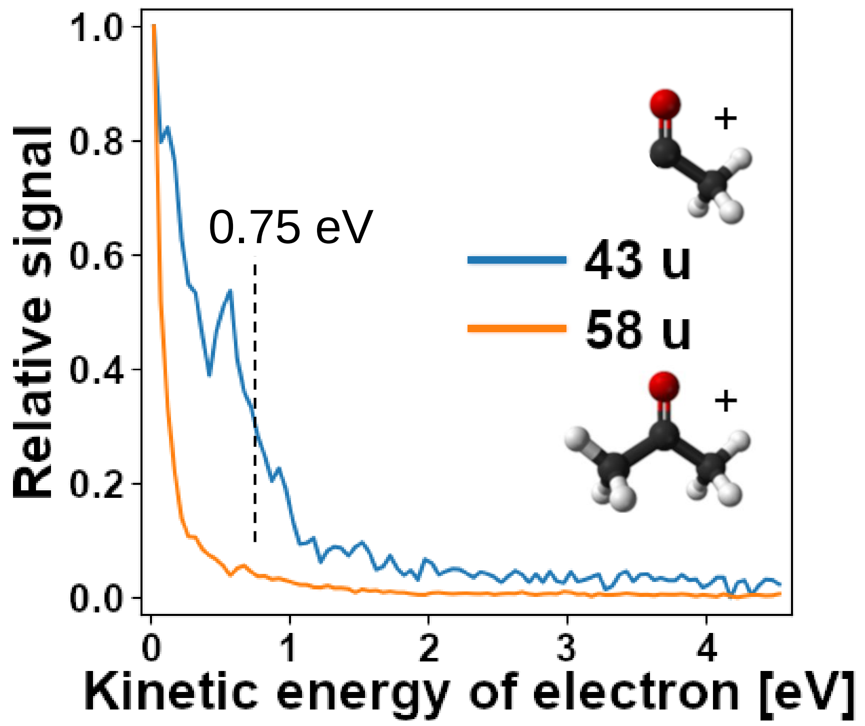
Example: Acetone





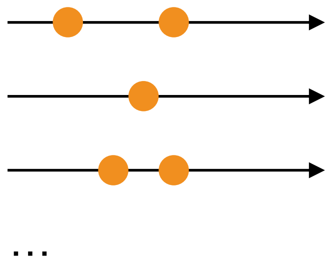
Example: Acetone



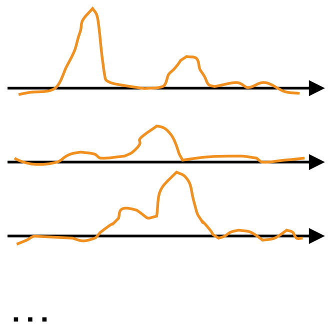


How to record the pieces?

- Coincidence approach
  - "Small data":
    - Find discrete particle hits from each laser shot.
    - Store a few scalar **coordinate values** in a table. (A kind of "sparse data" representation.)
    - Simple mental model for analysis even when multi-dimensional coordinates.
  - Approximates all particles detected for a laser shot really came from *the same* molecule.
    - Requires average < 1 molecule ionized per shot (wastes laser shots without events)
    - OK for synchrotron, bad for lasers with high power but low repetition rate (table-top or FLASH)

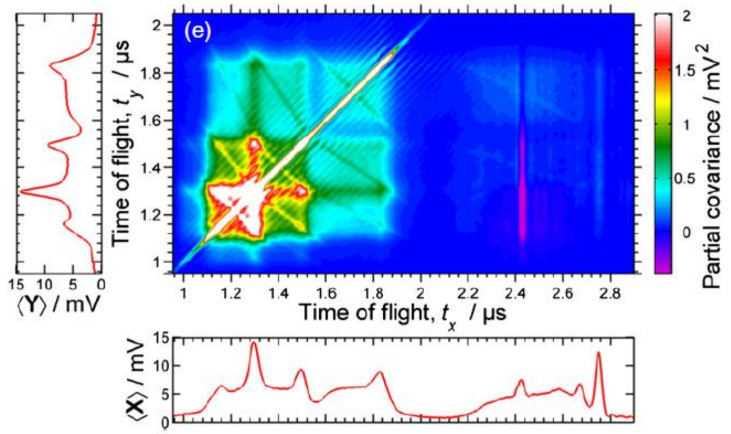


- Covariance approach
  - General statistical analysis, any number of ionized molecules per laser shot.
    - Finds the correlation of "which pieces tend to come together".
    - OK even without being able to isolate discrete particle hits from each laser shot.
  - Typically "big data":
    - When one keeps the raw **array of intensities** (spectrum, waveform, image, histogram) for each laser shot.
    - (Well, if one would give up the ability to refine calibration, it might be possible to compute the covariance live without saving the raw data – but we prefer to keep the ability to calibrate afterwards.)



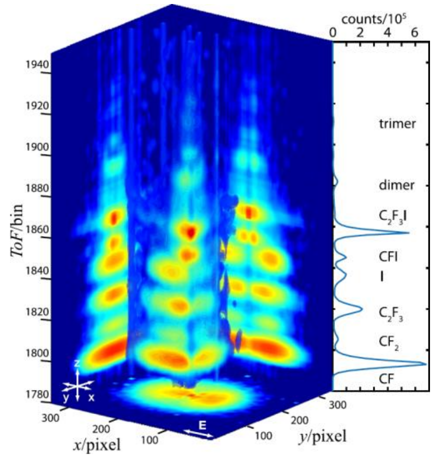
Literature examples

Which ions come together?



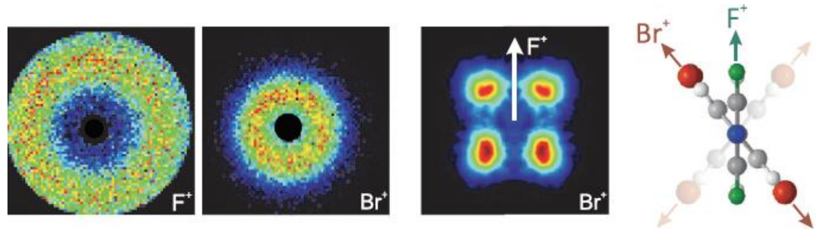
Kornilov *et al.*, J. Phys. B **46,** 164028 (2013)

VMI and TOF?



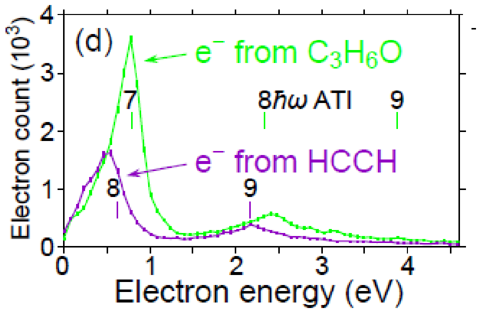
Forbes *et al.*, J. Chem. Phys. **147**, 013911 (2017)

Relative angle between fragments



Slater *et al.*, Phys. Rev. A **89**, 011401 (2014)

Photoelectron spectra for mixture of ions

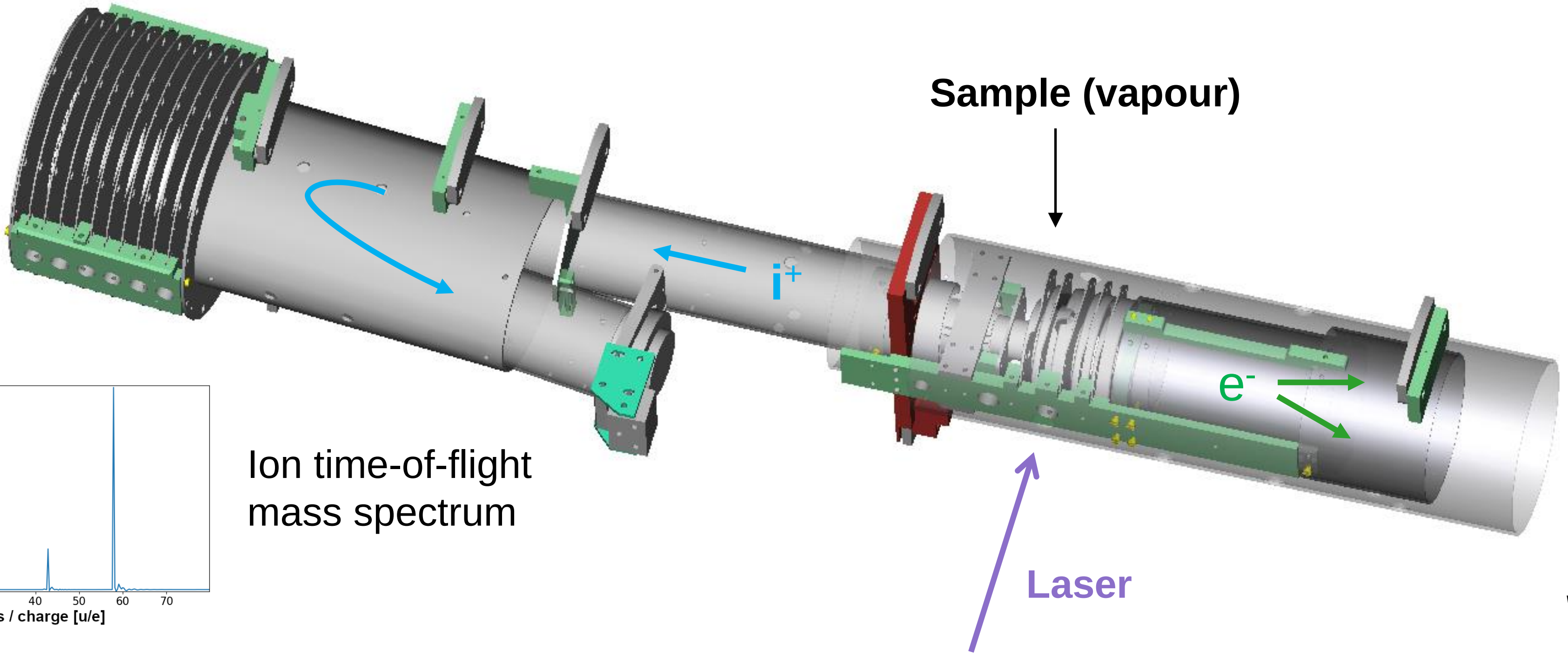


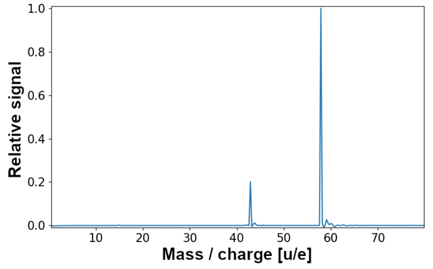
Månsson *et al.*, Rev. Sci. Instrum. **85**, 123304 (2014)

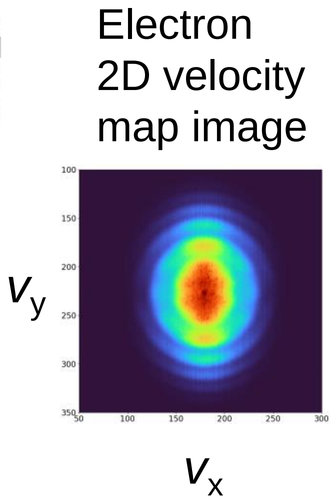
CFEL SCIENCE
Attosecond Science

More about the data

Our set-up and max data rates







- 1 kHz repetition rate is set by the laser.
- Can ionize many molecules within one shot: need covariance analysis.
- Mass spectra (some 10^4 samples of 14-bit ADC as int16, e.g. 25 MB/s) – not so much
- Images of the electron VMI screen (max 1024x1024 12-bit pixels as uint16, i.e. 2.1 GB/s) – much!

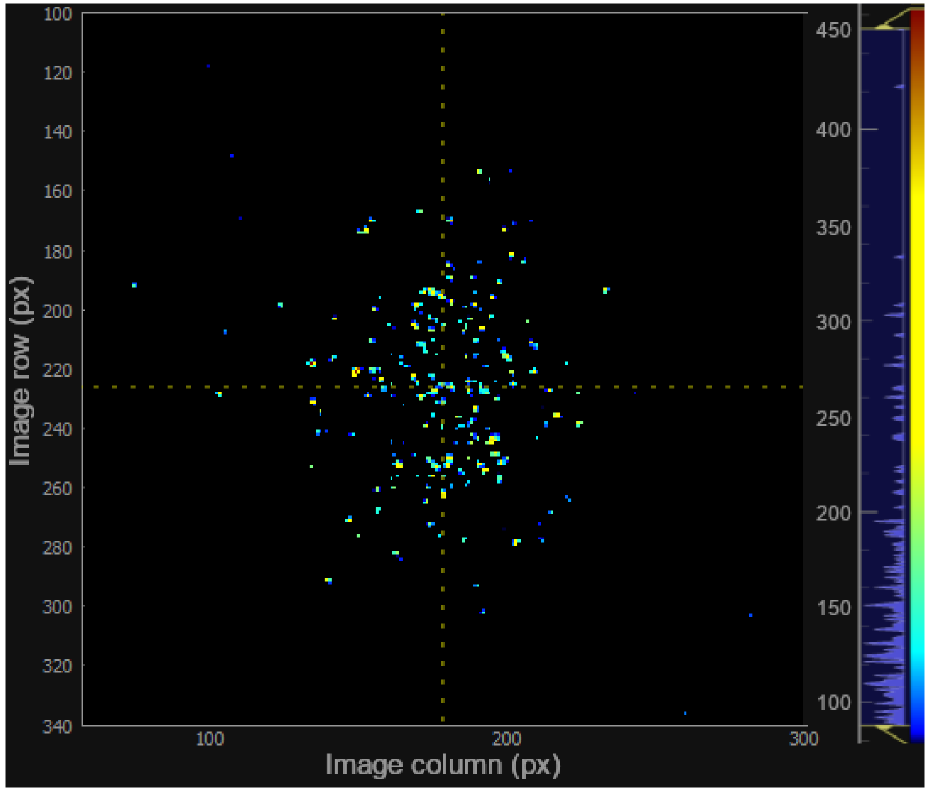
E. Månsson, *et al.* *EPJ Web of Conferences* **205** (2018) 03007

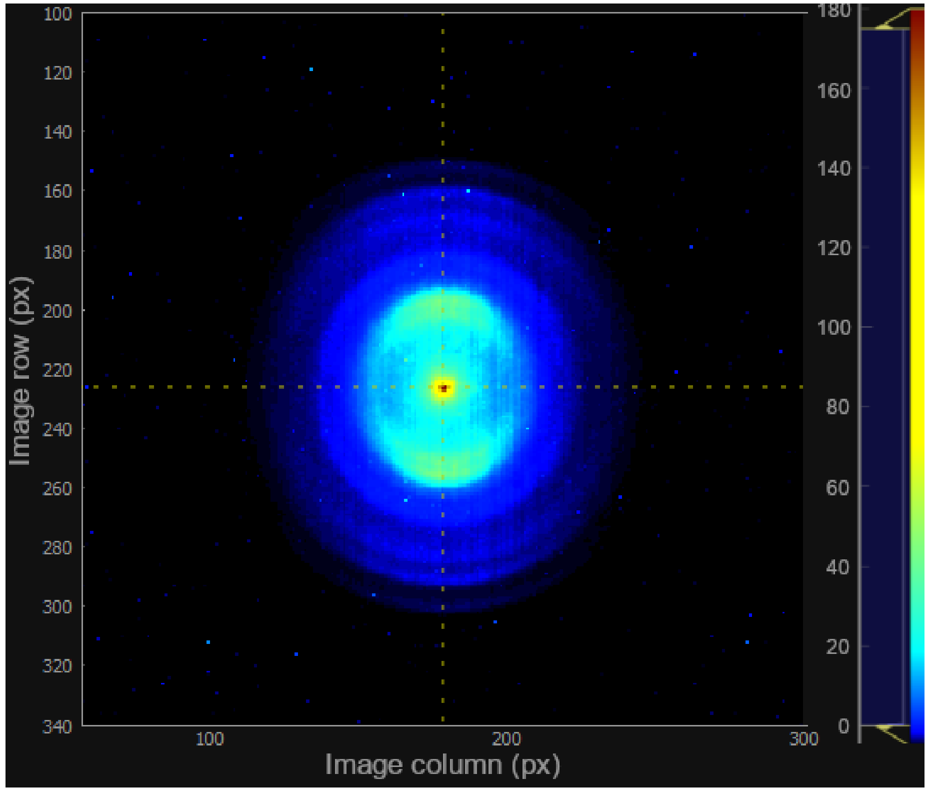
The image data

- The rest of this talk only concerns the images, as they dominate the raw data rate.
  - Optronis CP70-1-M-1000 CMOS-camera capable of 1280x1024 px at 1.05 kHz.
  - Due to square detector, the relevant max size is 1024x1024.
  - 4 CoaXPress cables to Euresys Coaxlink Quad CXP-12 frame grabber on PCIe.
  - Harvesters Python library can handle GenICam communication and read-out.
- Sending 1024x1024 16-bit integers to HDF5 (2.1 GB/s) is too slow
  - 260 Hz in my benchmark on our computer (Intel Xeon E5-1620 v4 3.5 GHz from 2018) with SATA III SSD (or HDD).
  - If uncompressed, likely limited by the storage device.
  - If compressed with standard HDF5-filters packaged with pytables (LZO chosen), likely limited by memory or CPU.
- We typically acquire at least 10 minutes per file: 1.3 TB of raw image data
  - To keep attosecond interferometer stability of experiment we don’t want long pauses between acquisitions, so buffering it in memory for later processing is not a good approach.
  - If we acquire 90% of the time, this is 7 TB/hour of raw data and a storage size problem.

Example images

With 2x2 pixel binning, cropped.





The image for one laser shot,
“Good” amount of signal (it can be less than this).

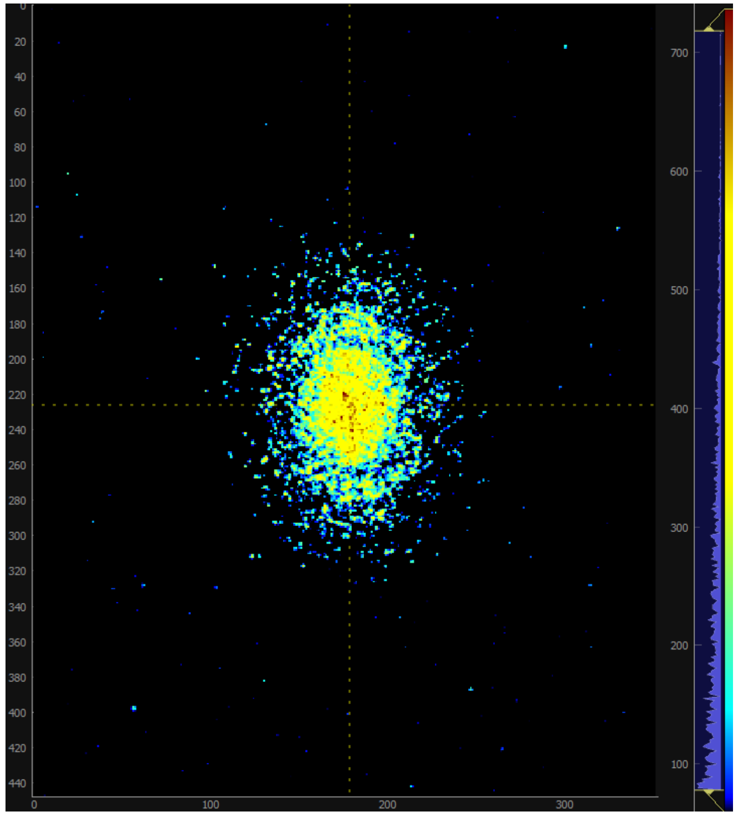
The average of many images.

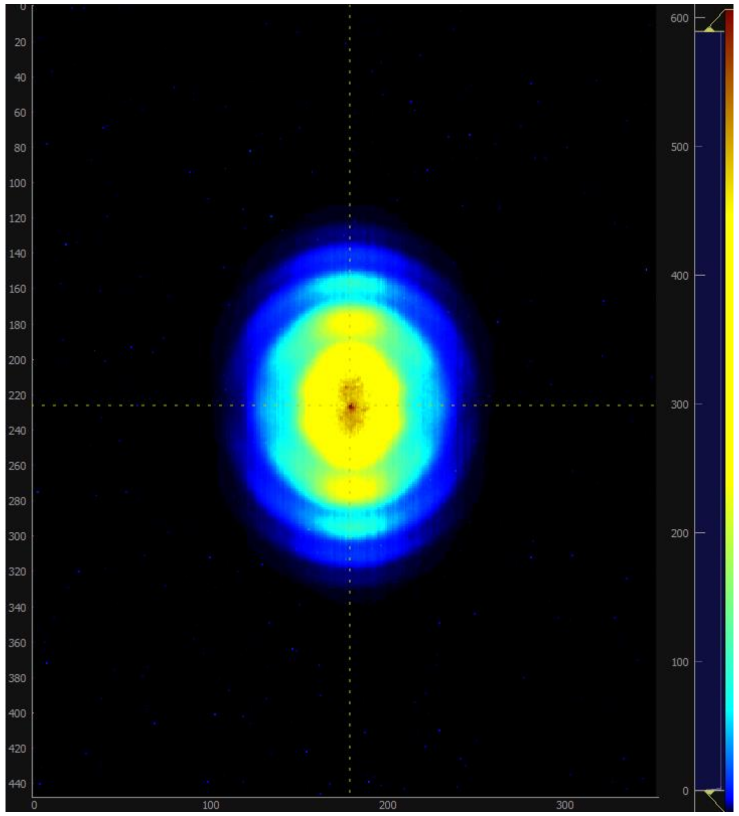
The horizontal and vertical axes correspond to 2D-velocity of electrons, with zero in the middle.

Cranking up the laser intensity

CFEL SCIENCE
Attosecond Science

With 2x2 pixel binning, cropped.



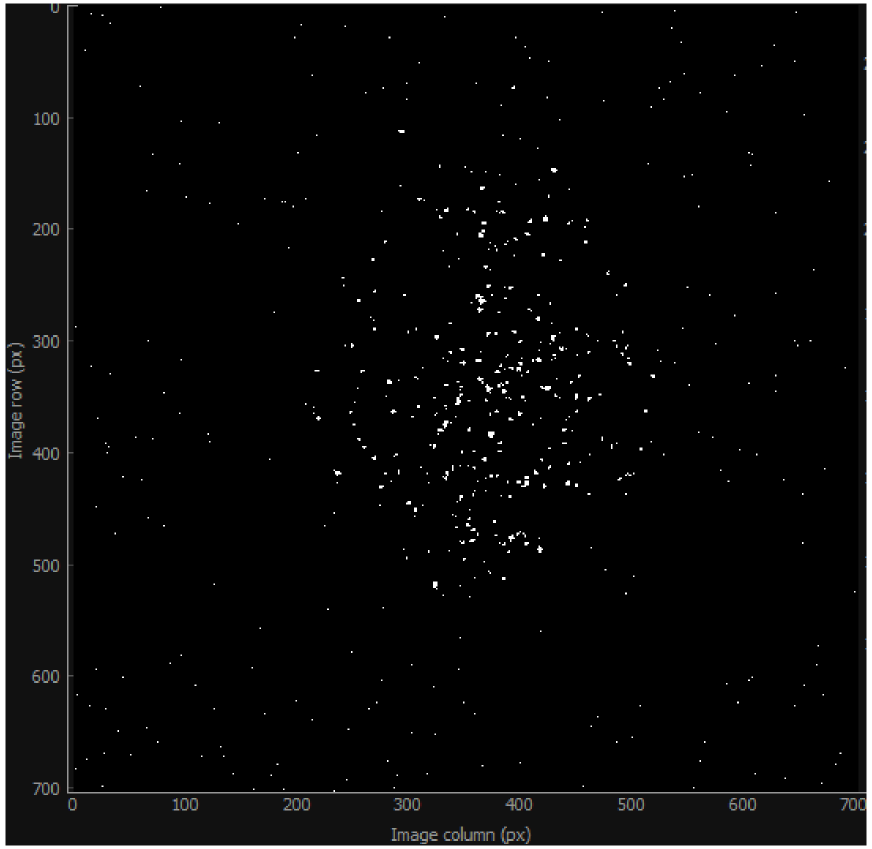


This many electrons per shot is not expected in scientifically relevant experiments,
it is here achieved by just using the near-infrared laser at near full power.

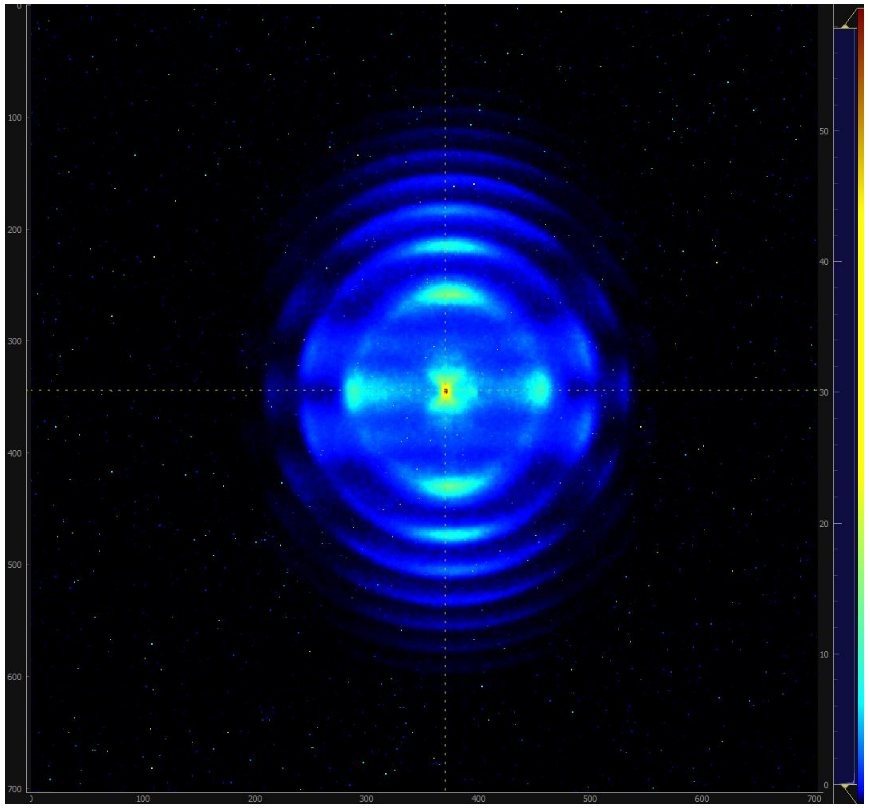
(Normally we want to run experiments with shorter pulses in the ultraviolet or extreme-ultraviolet.)

Larger images

Without binning, cropped to 704 x 704 pixels.



The images of individual laser shots were computed for live display but could so far not be saved at 1 kHz.



The average was saved.

Compression

Lossy compression

Exploit that the images are mostly black

- Pixels darker than a threshold can be set to zero.

- But our data does not seem to allow “spot finding and centroiding
  - (i.e. to just store coordinates of countable particles)
  - We have enough electrons per image that they often overlap
    - Not clear whether a cluster of pixels should be counted as two nearby particles.
    - Not clear whether a brighter spot is the sum of two particles or just random detector variation.

Lossy compression so far

Exploit that the images are mostly black

Details:

1. Subtract a dark image background
   for things like dark current and CMOS readout noise.
2. Set pixels with differential intensity values below a threshold to zero.
3. Increment the “sum of all images” with the current image’s data
   so the GUI can show the average image live. *(Optional, but desired by users.)*
4. Send the image for HDF5-saving, appending it to an EArray with LZO-compression.

   The chunk size is at least 1 image (more if they fit in 1 MiB) and the chunk cache is 64 MiB.

   Gives about 1/10th to 1/70th of the raw size.

- Steps 1 & 2 modify the image buffer **in-place** to not need array-copy!
- Steps 1 & 2 are implemented within the **same loop** over the pixel array,
  compiled to **machine code** via LLVM by Numba. (*TODO: include step 3 in this loop.*)

Software approach so far

The spectrometer has been used for science since 2021

- Stand-alone acquisition program, mainly in Python
  - Numba for performance-critical array-operations (LLVM-compiled to machine code).
  - One GUI-thread (PyQt5, pyqtgraph), one acquisition thread (harvesters, PyTables), with some further multithreading in NumPy and Numba-subroutines
    - Didn’t try sophisticated work-sharing between threads as tricky in Python (global interpreter lock).
    - Using a “hack” to make Harvester’s image buffer writable to not need array-copy for processing.
    - The acquisition loop works in blocks of about a hundred images, a hundred mass spectra, a hundred images, …
    - Both kinds of data are appended to chunked arrays in the same HDF5-file (compound table or two files not better).
- Writing the HDF5-file locally and user-triggered script to transfer to network storage
  - Even if transferred in parallel with the next acquisition this gives better performance than direct writing over network – probably due to access time for “seeking” within the file.
- Have to limit the raw data size, at the cost of half the velocity resolution and some range
  - By 2x2 pixel binning and customizable cropping in the camera hardware.
  - Then LZO-compression and writing is fast enough to do 1 kHz.

New compression approach

Don’t pass all the zero-pixels to HDF5

- Make a kind of “sparse data”-compression in the application layer!

- The approach under development is to change the last step, to avoid letting a HDF5-filter see/copy the large but mostly zero-filled array of pixels (up to 2 MiB).

- The compression instead needs to be handled in the application layer,
so it can be inlined in the single loop over the array
(steps 1, 2 and maybe 3 on the previous slide).

New compression approach

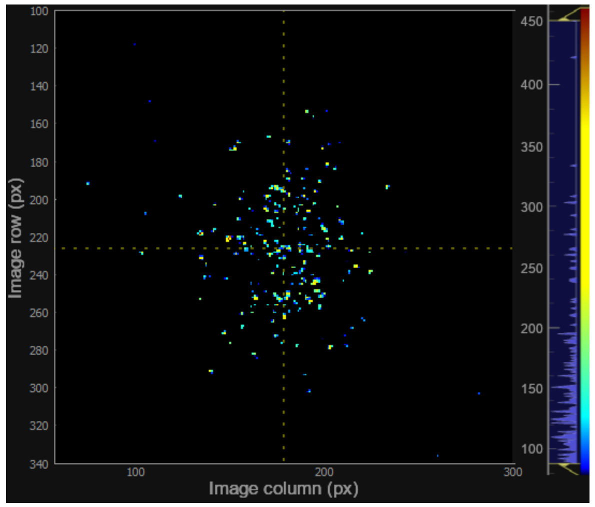
Don’t pass all the zero-pixels to HDF5

- The proposed “darker-than-threshold run length encoding” (DRLE):
- Process the image as an 1D-array of signed 16-bit integers:
  - For a run of L successive pixels below the user-chosen threshold, with $2 \leq L \leq 2^{15}$, the output buffer gets the negative value $-L$ appended.
  - A single dark pixel ($L = 1$) is encoded as 0 to make the decoder’s job easier (not needing to change a -1 to a single 0).
  - Other pixels have positive values (≥ threshold) and are copied to the output buffer.
- This lossy variant of RLE will never make the output longer than the input.
  - (The worst case of alternating bright and dark pixels retains the original length.)

Performance results

Benchmarking results*

Test case		Shuffled LZO HDF5-filter		DRLE, no HDF5-filter	
Pixels in image (10^6)	≥ threshold	Rate (kHz)	File size (MB)	Rate (kHz)	File size (MB)
0.26	0.56%	**1.12**	838	**6.74**	498
0.5	0.56%	0.59	839	**3.81**	500
1.0	0.56%	0.26	837	**1.96**	499

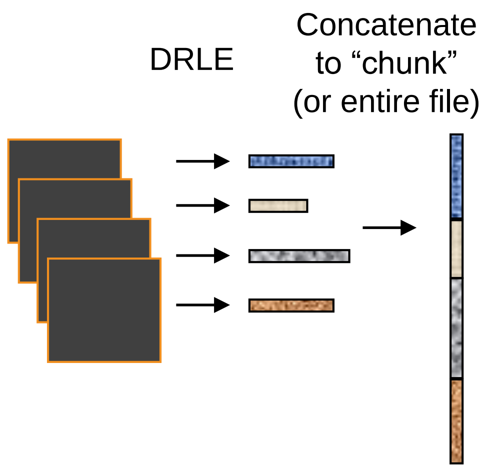


* So far using a script rather than the exact acquisition program

- No mass spectra, background subtraction or GUI communication.

- 30 000 images saved (repeating 100 MB of real data) with PyTables.

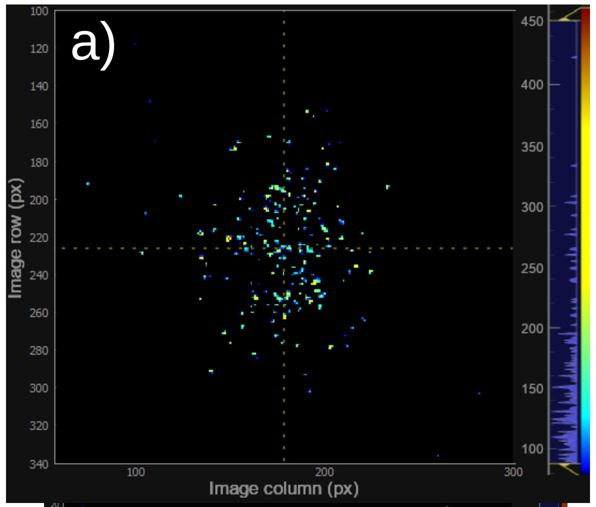
- Variable-length arrays (VLA) for compressed "chunks" of 50 images (circa 1 MB) perform as well as appending to a single extendable 1D-array.
  - I learned yesterday that the second option might be like VLA but in the application layer.
  - (Using VLA with an entry for each individual image was slower.)

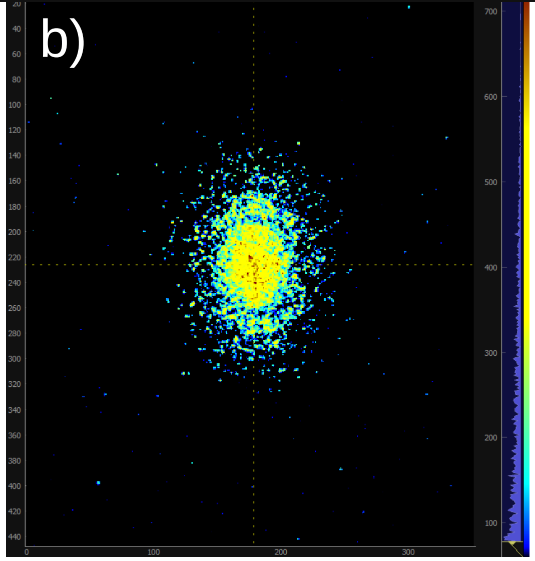


Benchmarking results*

Test case		Shuffled LZO HDF5-filter		DRLE, no HDF5-filter	
Pixels in image (10^6)	≥ threshold	Rate (kHz)	File size (MB)	Rate (kHz)	File size (MB)
a) 0.26	0.56%	**1.12**	838	**6.74**	498
a) 0.5	0.56%	0.59	839	**3.81**	500
a) 1.0	0.56%	0.26	837	**1.96**	499
b) 0.5	5.51%	0.53	3637	**2.5**	3887
b) 1.0	5.51%	0.24	3630	**1.28**	3880

- For typical threshold choices and numbers of electrons per image
  - The old approach with only HDF LZO-compression gives a 71:1 compression ratio
  - while the new DRLE gives 120:1.
- For images with too much signal (fewer black pixels) the ratios even out to 17:1 and 15:1, respectively.
  - The new is still much faster.



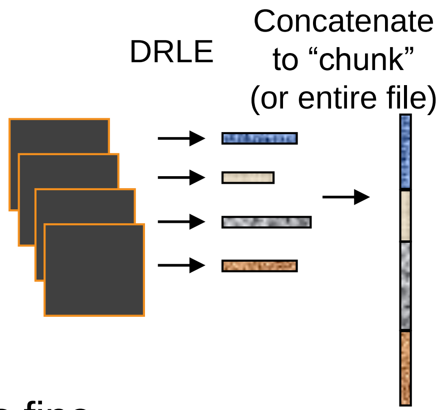


Conclusions

Test case			DRLE, no HDF5-filter		Data rates	
Pixels in image (10^6)	≥ threshold		Rate (kHz)	File size (MB)	Raw (GB/s)	To file (MB/s)
a) 0.26	0.56%		**6.74**	498	3.5	29
a) 1.0	0.56%		**1.96**	499	3.9	33
b) 1.0	5.51%		**1.28**	3880	2.6	165

- Higher compression ratio and simple algorithm avoiding array copies seems promising for full-sized images to be processed and saved at 1 kHz.
- Achieving a high compression ratio is essential for making the file-writing work at 1 kHz,
  - especially when no further HDF5-compressor is used.
  - The choice of threshold level is done by the user while seeing the live data.
  - Experiments will typically be performed with less signal per image than in example b, so sufficient compression seems realistic.

Problems created by this solution



- Since the encoding and decoding is in the application layer, it affects all analysis scripts and file viewers more than a low-level HDF5-filter.
- If I use a VLA Dataset with *N* compressed *images* per entry, the VLA-index is not the image index
  - but *N* can be an attribute so a decoding layer knows where to find image *i*.
  - The analysis application will read all images, sequentially, so needing to decode the entire chunk is fine.
- If I append to one long extendable 1D-Dataset, I will use a second dataset to encode the "start address" of image *i*.

- To hide this from users, a Python class can wraps HDF5 Datasets and override the array indexing operator [ ] to still allow getting an image by indexing as `wrapper[i]` or a pixel as `wrapper[i, row, column]`.

Acknowledgements

CFEL-ATTO, DESY, Hamburg

Involved PhD students:
Sergey Ryabchuk
Lorenzo Colaizzi
Krishna Saraswathula
Ammar Bin Wahid
Josina Hahne

Involved seniors:
Andrea Trabattoni
Vincent Wanie
Francesca Calegari

http://atto.cfel.de

HELMHOLTZ

CFEL SCIENCE Attosecond Science

STARLIGHT

CNR IFN Istituto di Fotonica e Nanotecnologie

erc European Research Council Established by the European Commission

CUI THE HAMBURG CENTRE FOR ULTRAFAST IMAGING

Laserlab Europe The "Integrated Initiative" of European Laser Infrastructures in the 7th Framework Programme of the European Union

DESY.

SFB 925: Light induced dynamics and control of strongly correlated quantum systems

UH Universität Hamburg DER FORSCHUNG | DER LEHRE | DER BILDUNG

Discussion points

- It seems the same sparse paradigm described by Elena Pourmal (proposed for SLAC-II) would be useful also for my application!
  - The call to the C library function for assigning to elements (pixels) in the Dataset must however be fast for use on individual pixels within my “tight loop” over the image pixels.
- Can my kind of compression filter be implemented as a HDF5-plugin?
  - I want something like virtual array-indexing along the chunked dimension to get input data into the filter, to avoid having to copy memory to concatenate data from several images to one chunk before running the filter.
  - This would mean passing an array of pointers to mutable image buffers (not contiguous in memory) as input to the compression filter.
  - Is this something for a “Virtual Object Layer” plugin? – Probably not?
  - *Learned yesterday:* Maybe I should just find a way to call HDF5’s “direct” writing of already compressed chunk? If I also make a decoding filter plugin, there might be no changes to analysis code at all.
- I’d vote for HDF5 v. 2 to support compressing data in arrays with variable-length entries
  - After my application has gotten rid of all the zeros, a sophisticated compressor might still offer some size reduction.
  - Arrays with fixed chunk sizes already give variable-length compressed data, so as a user I feel it should not need new kinds of filters.
  - One option could be to treat each VLA-entry as a chunk and just run the usual kind of filter on it.
  - Alternatively, use (lossless) filter at the lower-level buffer (filter won’t know at which bytes VLA-entries start or end).