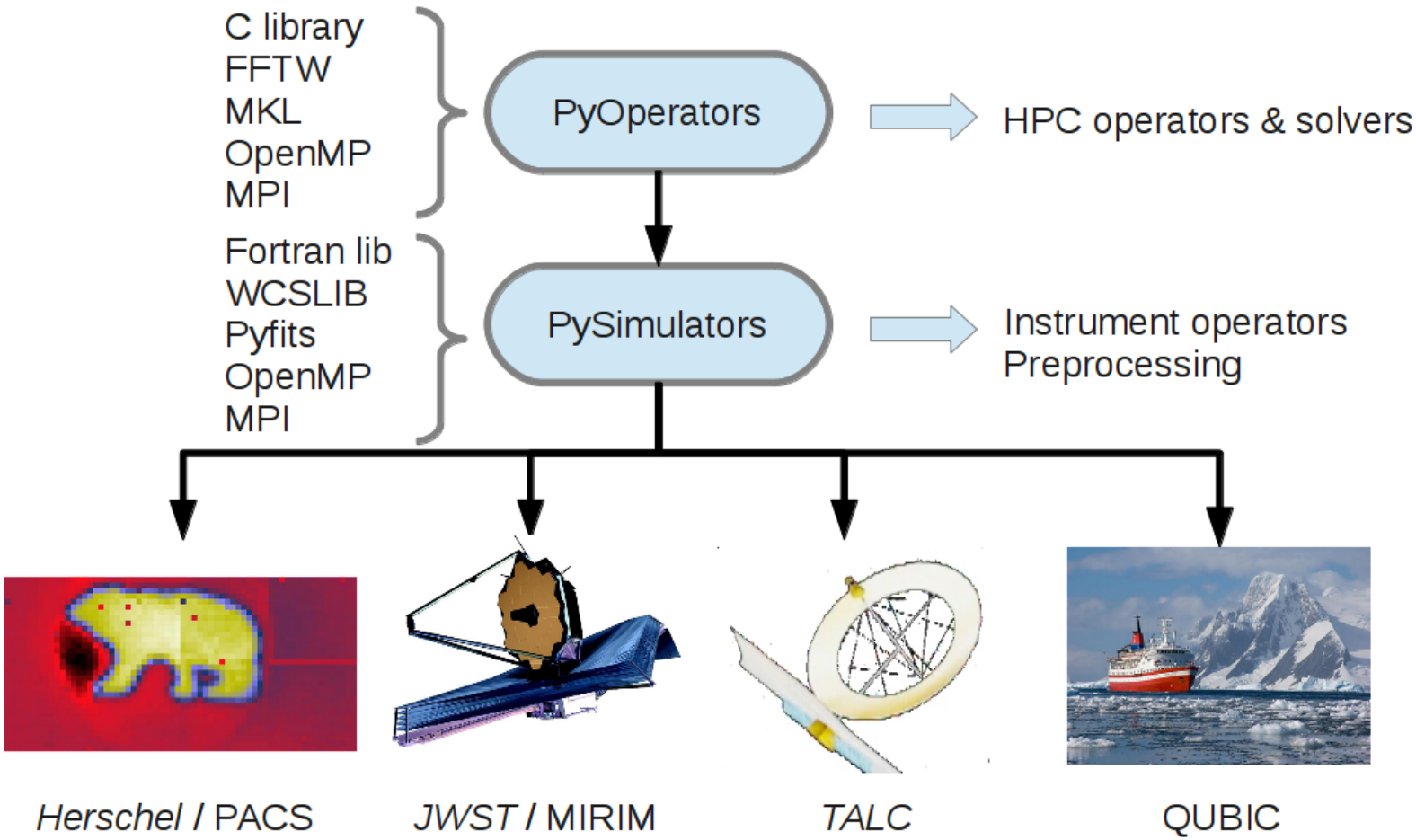
Tamasis

Pierre Chanial & **Pasquale Panuzzo**

The Tamasis project

- Tools for Advanced Map-making, Analysis and SImulations of Submillimeter surveys.
- ASTRONET-funded project, P.I: Marc Sauvage
- Collaboration between 4 institutes:
  - Garching-ESO: cosmological light cones simulations
  - IAS: SPIRE advanced map-making (next talk)
  - Saclay: PACS advanced map-making
  - Leiden: stacking
- massively parallel + super-resolution
- Tools are publicly available to the community:
  - http://pchanial.github.com/pyoperators
  - http://pchanial.github.com/tamasis-pacs

The Tamasis project



A generic pipeline for PACS

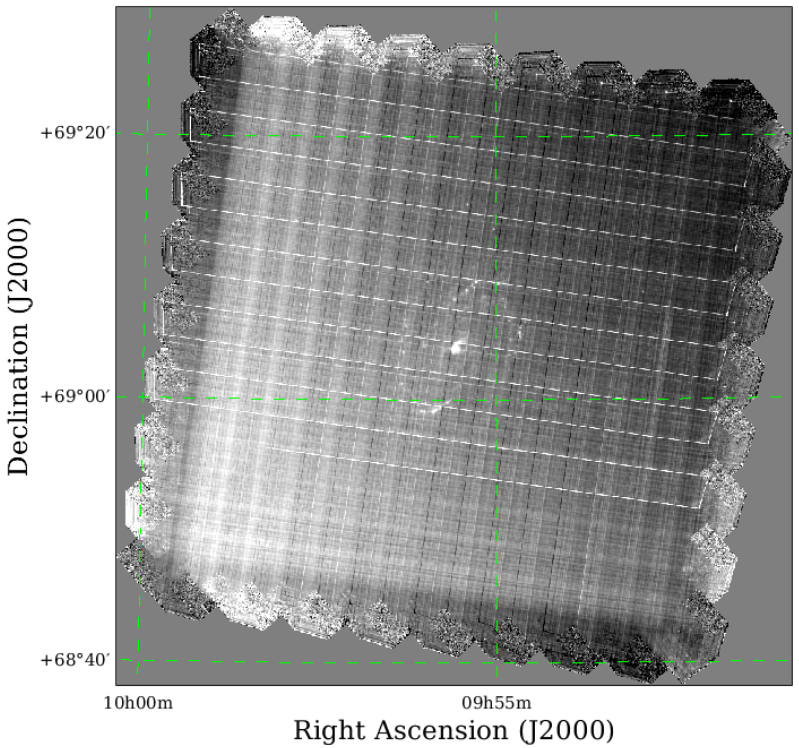
- Preprocessing
  1. Baseline removal
  2. Jump detection
  3. Second-level deglitching
- Map reconstruction by inversion
  1. Instrument model
  2. Objective function
  3. Minimisation

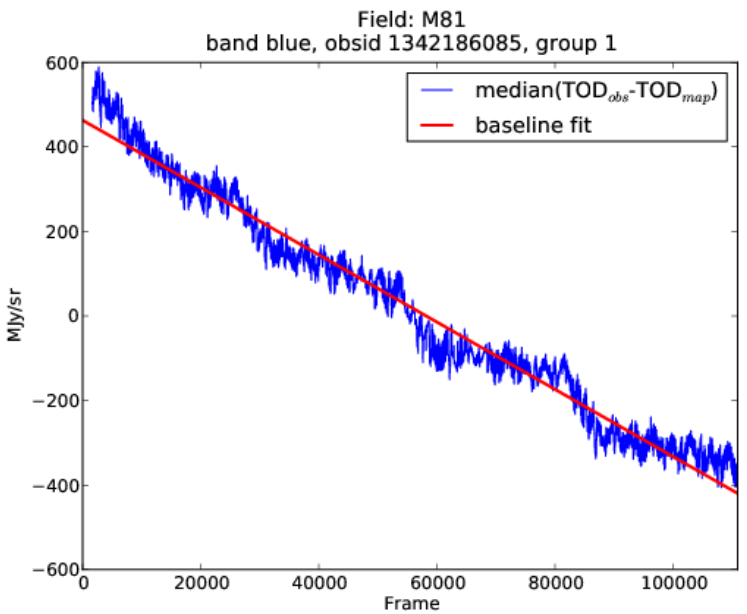
Baseline Removal, part 1

Inspired by the SPIRE destriper.

- Remove median over time for each bolometer and each observation $\Rightarrow y_0$,
- Repeat 3 times, starting with iteration $i = 0$:
  1. Backproject y_i into a map $x_i = \frac{P^{\mathrm{T}}(y_i)}{P^{\mathrm{T}}(1)}$
  2. Compute projection $\tilde{y}_i = P(x_i)$
  3. Compute drift $z_{i,\mathrm{group}} = \langle y_i - \tilde{y}_i \rangle_{\mathrm{group}}$
  4. Compute 1st-order polynomial fit $\tilde{z}_{i,\mathrm{group}}$ to $z_{i,\mathrm{group}}$
  5. $y_{i+1} = y_i - \tilde{z}_{i,\mathrm{group}}$
  6. $i = i + 1$

Baseline Removal, part 1

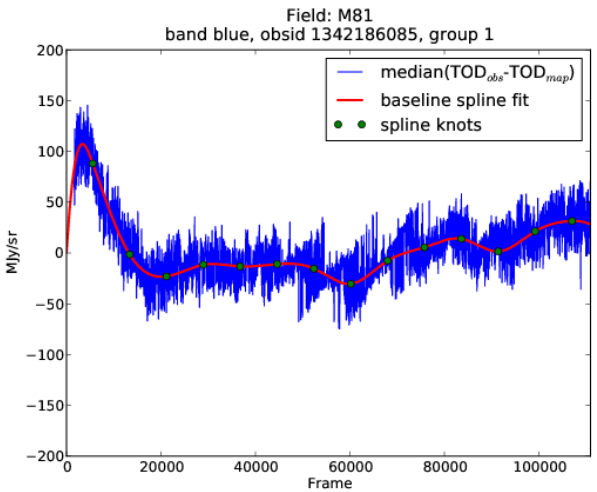




Baseline Removal, part 2

To remove non-linear drifts

1. Backproject y_i into a map $x_i = \frac{P^{\mathrm{T}}(y_i)}{P^{\mathrm{T}}(1)}$
2. Compute projection $\tilde{y}_i = P(x_i)$
3. Compute drift $z_{i,\mathrm{group}} = \langle y_i - \tilde{y}_i \rangle_{\mathrm{group}}$
4. Compute spline fit $\tilde{z}_{i,\mathrm{group}}$ to $z_{i,\mathrm{group}}$
5. $y_{i+1} = y_i - \tilde{z}_{i,\mathrm{group}}$
6. $i = i + 1$

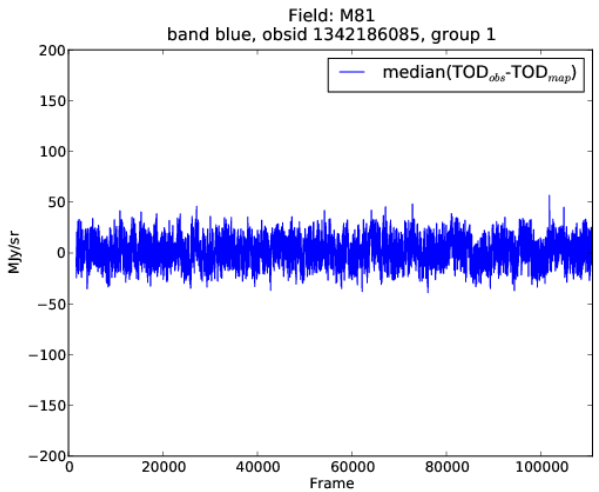


Baseline Removal, part 3

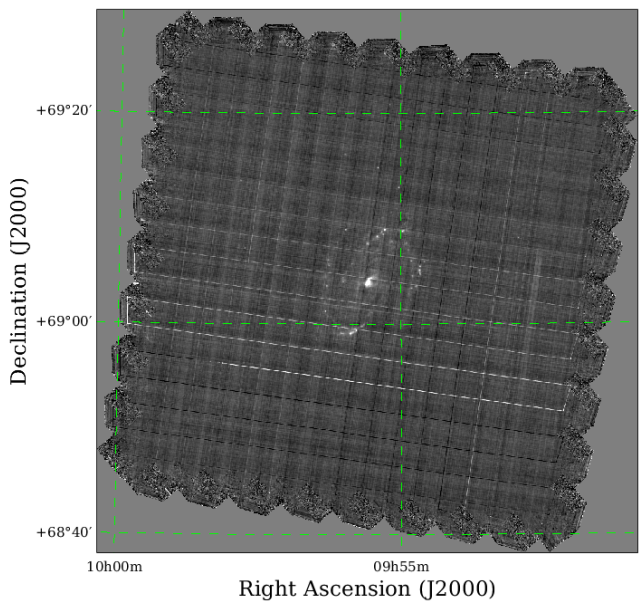
To remove non-correlated drifts

1. Backproject y_i into a map $x_i = \frac{P^{\mathrm{T}}(y_i)}{P^{\mathrm{T}}(1)}$
2. Compute projection $\tilde{y}_i = P(x_i)$
3. Compute drift $z_{i,\mathrm{detector}} = y_i - \tilde{y}_i$
4. Compute linear fit $\tilde{z}_{i,\mathrm{detector}}$ to $z_{i,\mathrm{detector}}$
5. $y_{i+1} = y_i - \tilde{z}_{i,\mathrm{detector}}$
6. $i = i + 1$

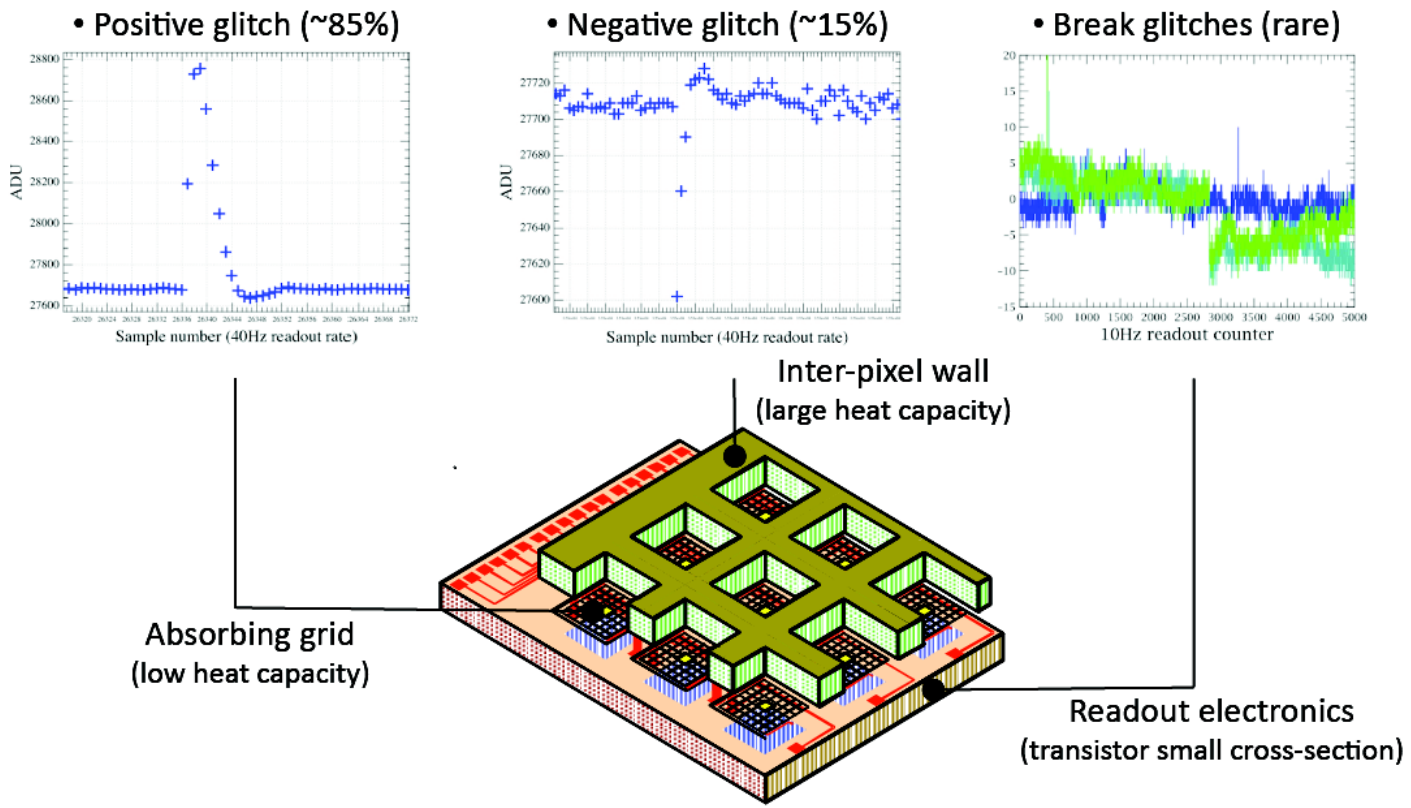
Final residuals.



Final map after baseline removal.

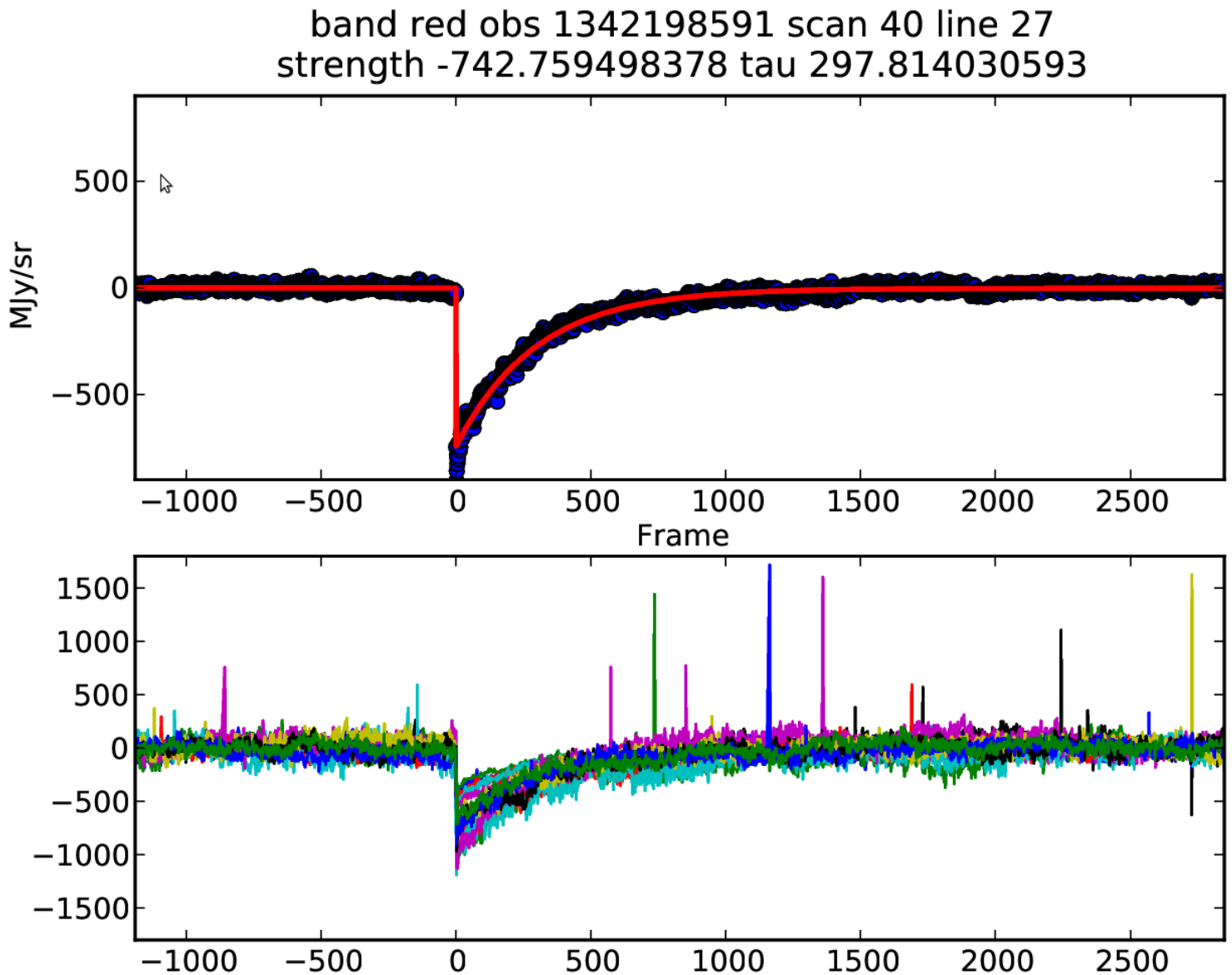


Glitch zoo



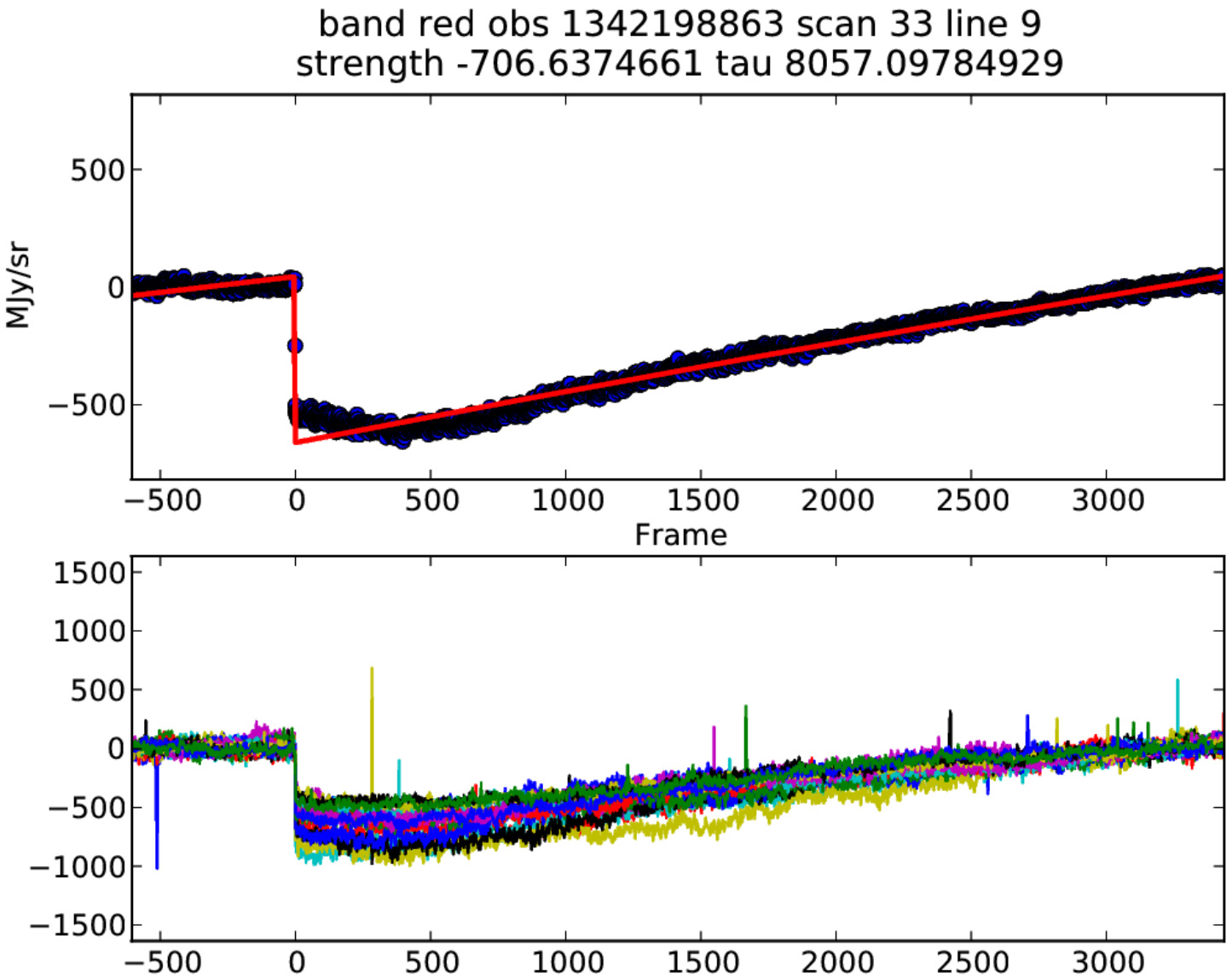
(N. Billot)

Glitch zoo



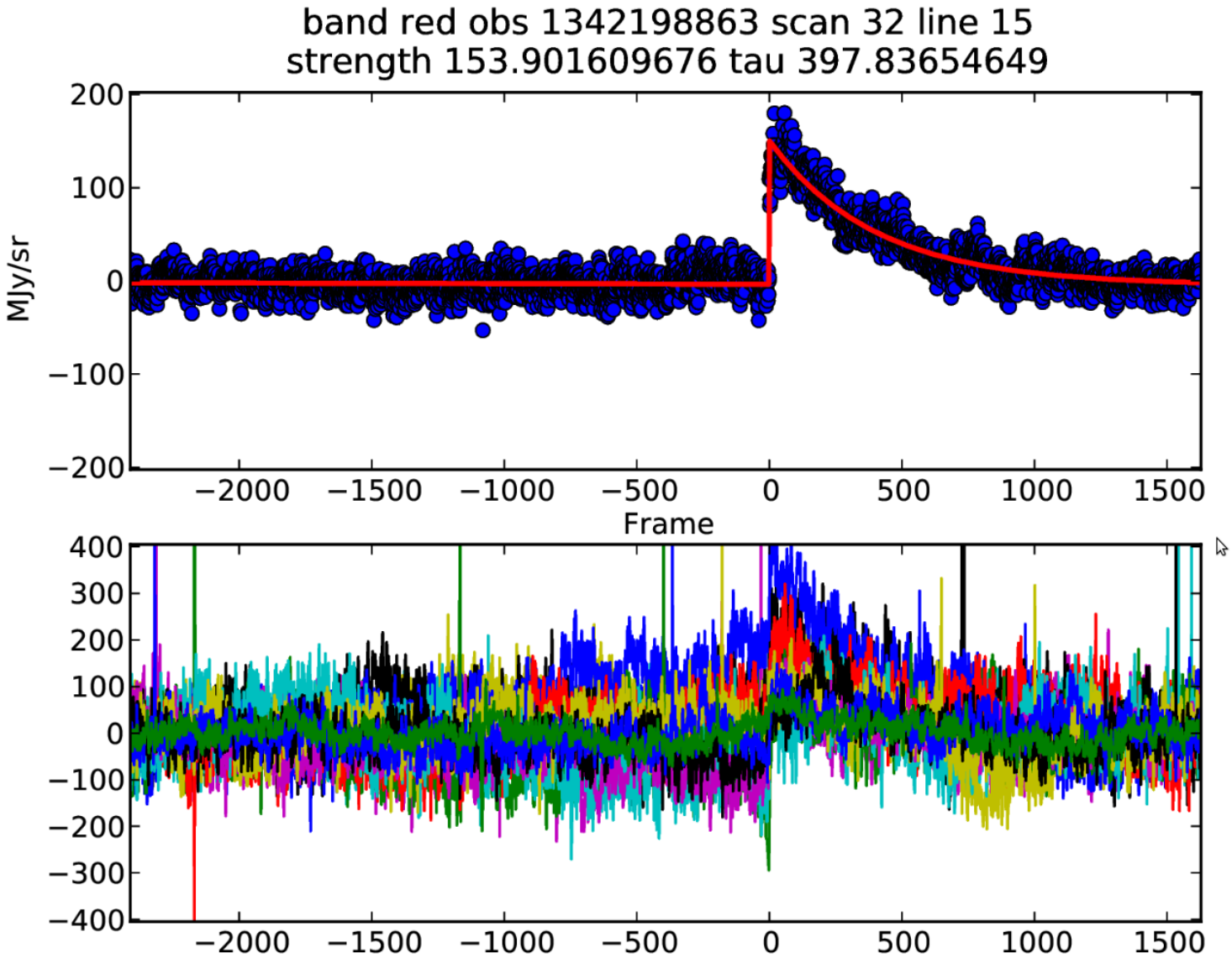
(P. Panuzzo)

Glitch zoo



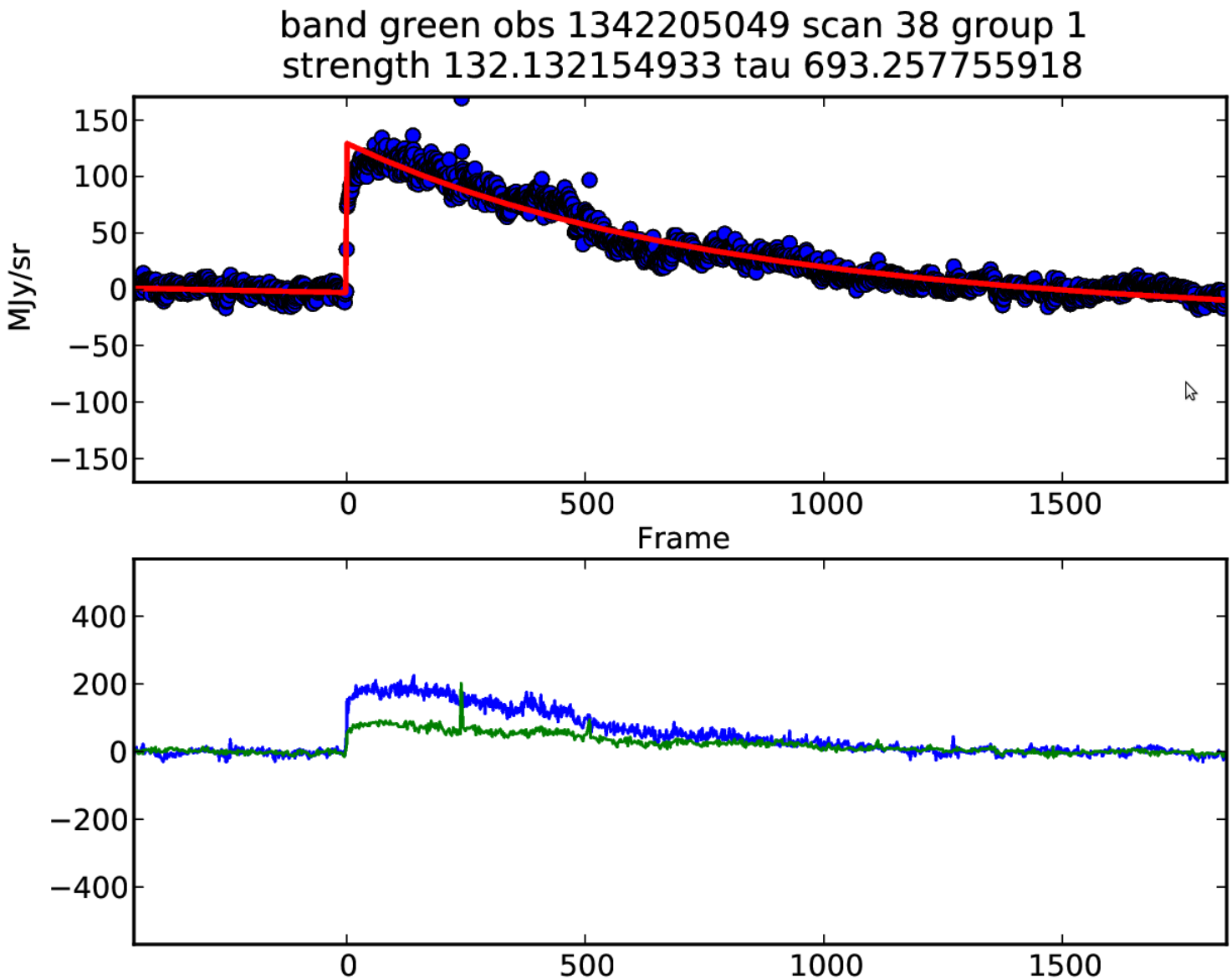
(P. Panuzzo)

Glitch zoo



(P. Panuzzo)

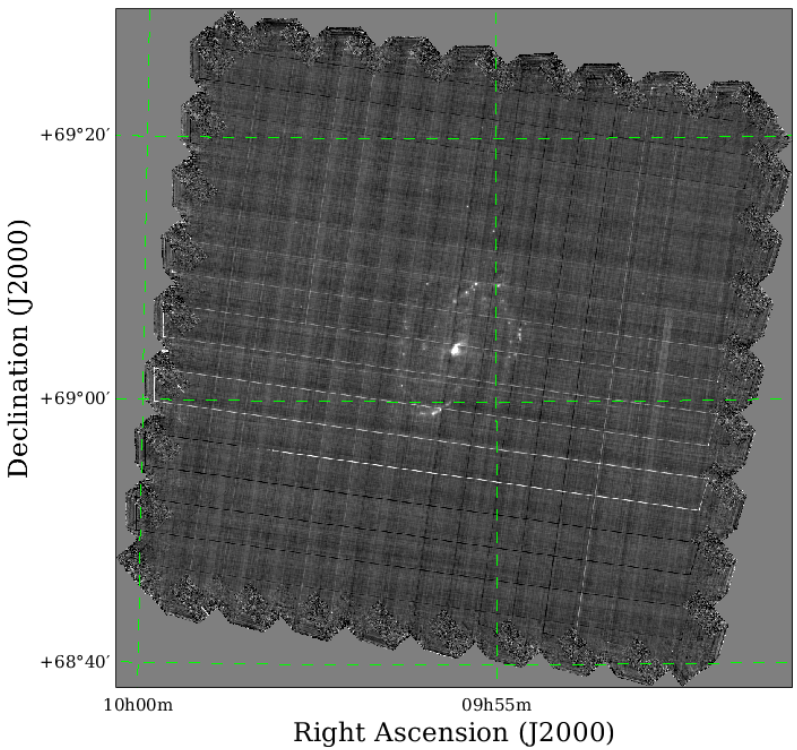
Glitch zoo

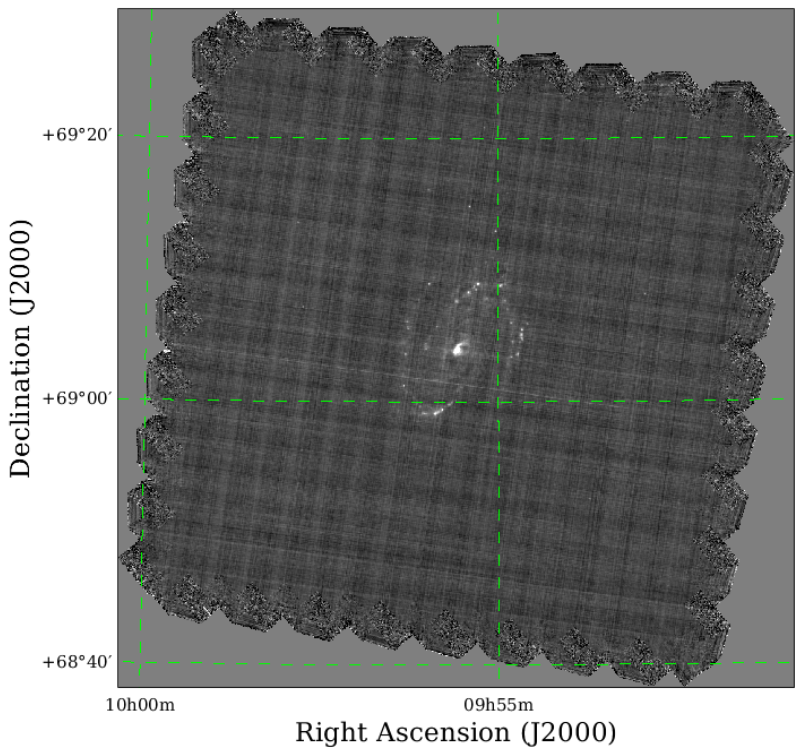


(P. Panuzzo)

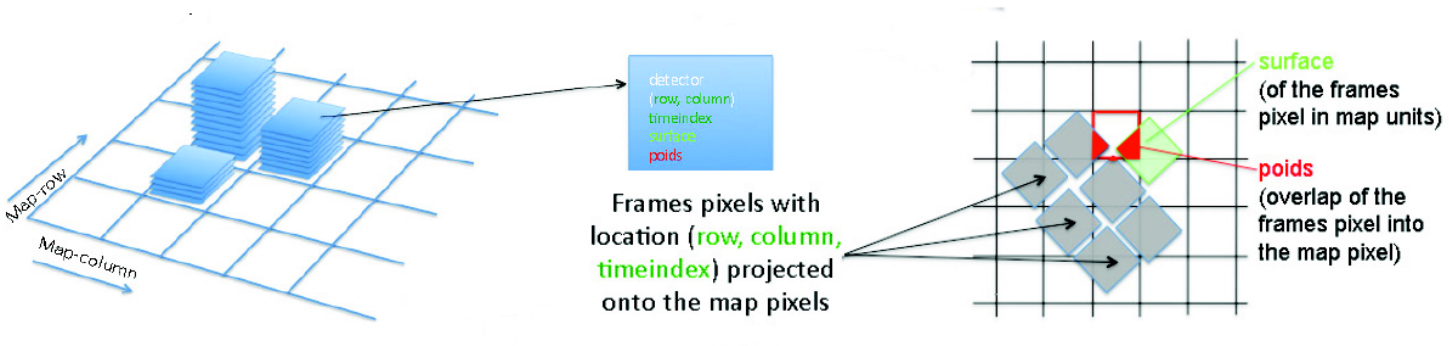
Long glitch removal: Jump detection

- Inspired by SPIRE preprocessing, but with Ω being a group, a line or a single bolometer:
  1. $x_\Omega = \frac{P^{\mathrm{T}}(y - y_\Omega)}{P^{\mathrm{T}}(1 - 1_\Omega)}$
  2. $\tilde{y}_\Omega = P(x_\Omega)$
  3. Detect jumps in $\langle y_\Omega - \tilde{y}_\Omega \rangle_\Omega$ using Haar transform





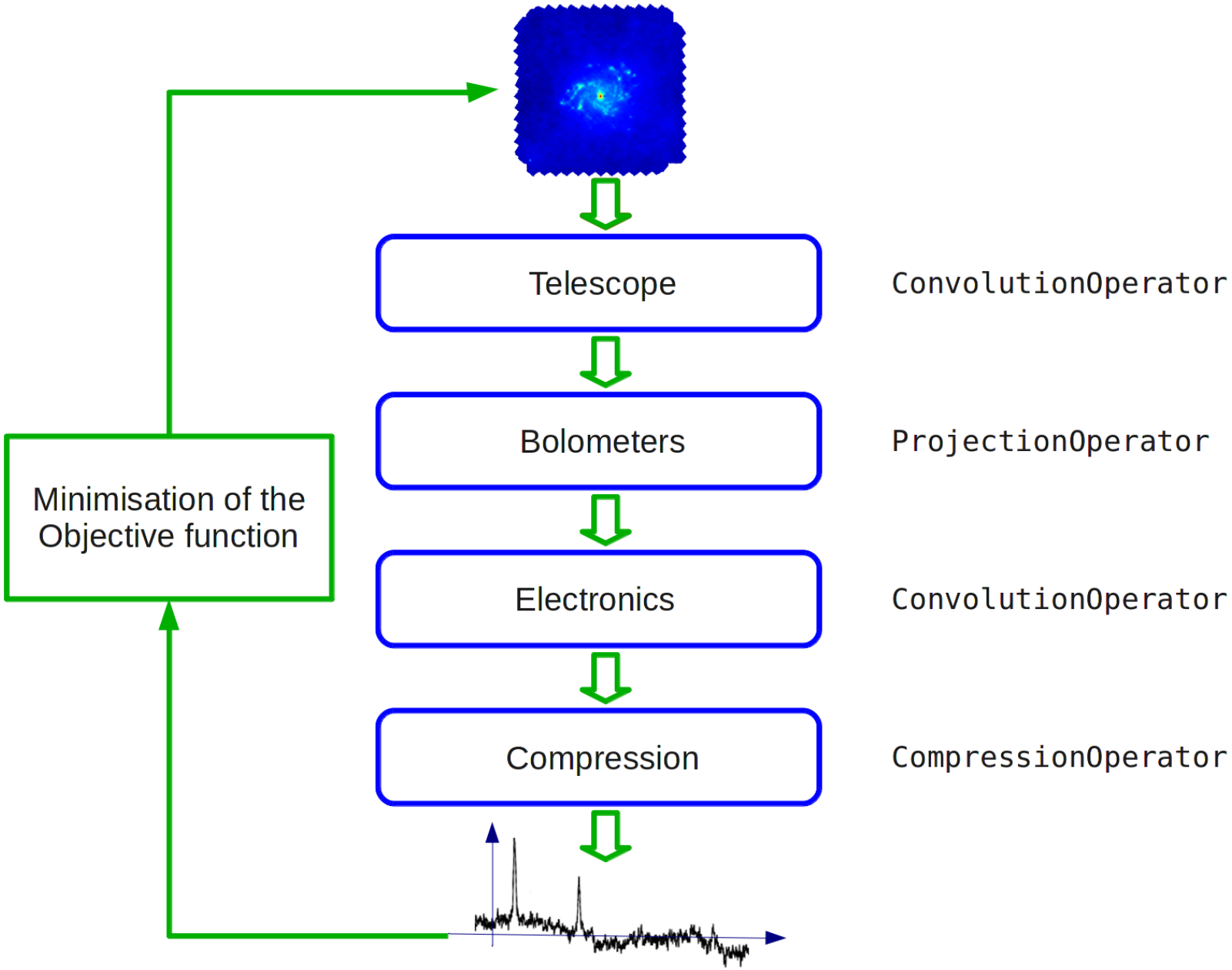
Short glitch removal: Spatial method (2nd level deglitching)



(N. Billot)

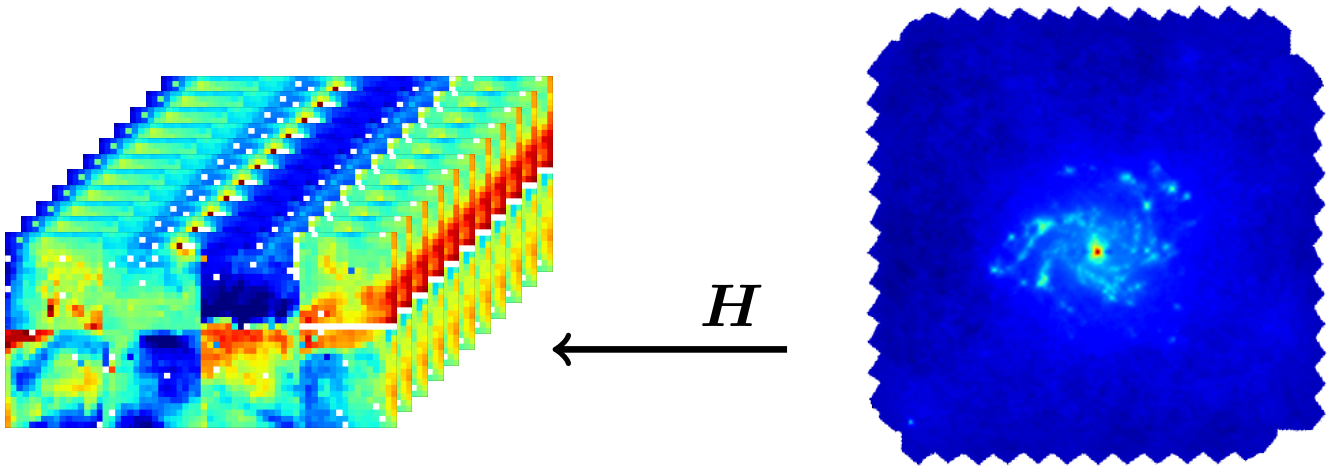
A generic pipeline for PACS

- Preprocessing
  1. Baseline removal
  2. Jump detection
  3. Second-level deglitching
- Map reconstruction by inversion
  1. Instrument model
  2. Objective function
  3. Minimisation


Telescope
ConvolutionOperator
Bolometers
ProjectionOperator
Minimisation of the Objective function
Electronics
ConvolutionOperator
Compression
CompressionOperator

Doing any science with any observation is attempting to solve an inverse problem

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{n}, \quad \boldsymbol{n} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{N})$$



$\boldsymbol{y}$ $\boldsymbol{x}$

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \left\{ ||\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}||^2_{\boldsymbol{N}^{-1}} + \Psi(\boldsymbol{x}) \right\}$$

This method requires $\approx$100$\times$ more compute power.

$\Rightarrow$ Acquisition model $\boldsymbol{H}$ must be very fast

Optimisations

Speed is one of Tamasis main drivers.

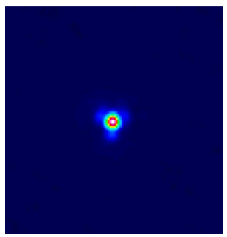
- $\Rightarrow$ gain $\times$180 (standard PACS pipeline, Java)
- $\Rightarrow$ gain $\times$2,4 (MADmap v1, pure C, scalar)

Optimisations

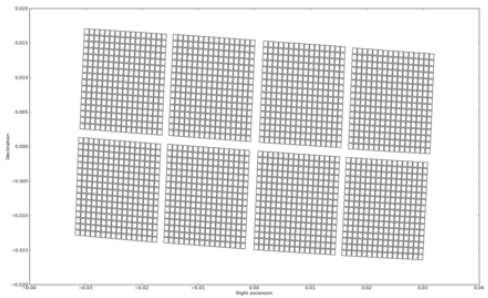
- As much preparation as possible before iterations
- Algebraic rule simplifications
- Pool-based Memory manager: no creation of array temporaries during iterations. Hand buffers to operators according to requirements (contiguity, alignment)
- In-place/out-of-place operators
- In-place reductions
- Block partition: crucial to take advantage of the cache

PACS artifacts

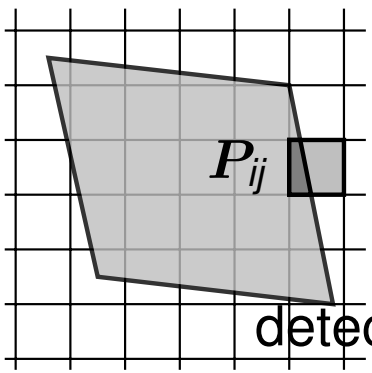
- **Telescope PSF**



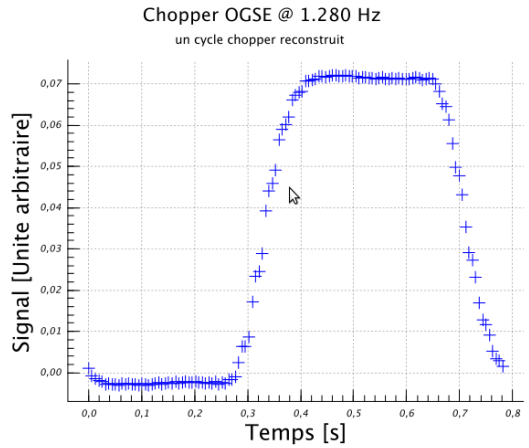
- **Distortion**



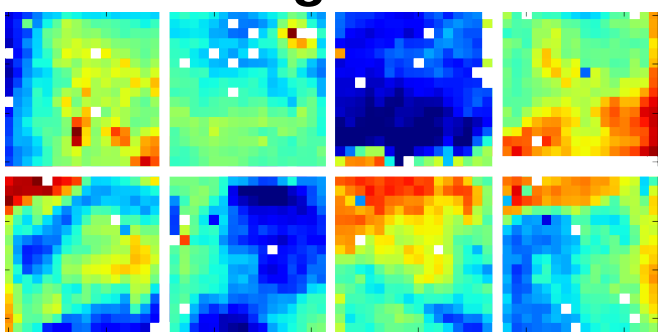
- **Projection**



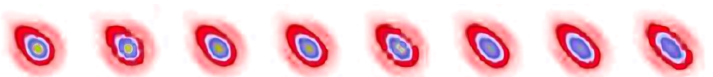
- **Bolometer response**



- **Bolometer gain**

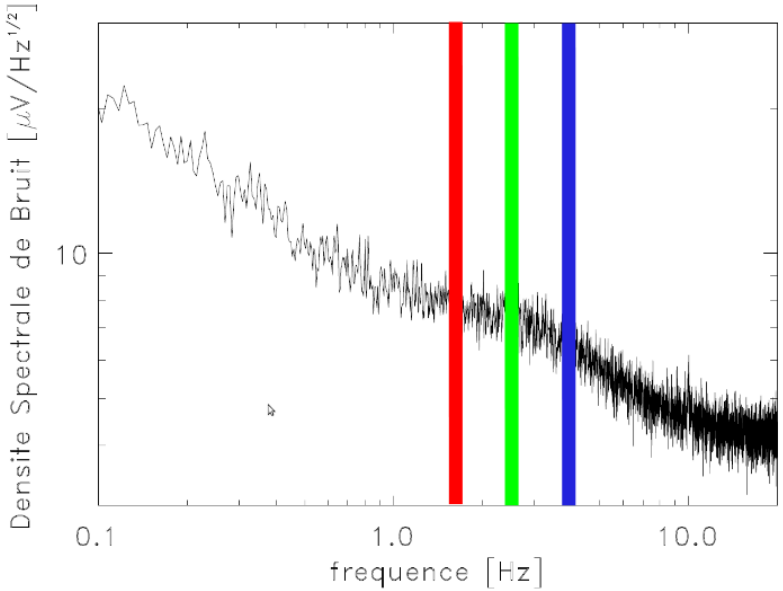


- **On-board Compression**



PACS artifacts

- **Offset drifts** $\Leftrightarrow$ 1/f noise



Even point sources are affected. The slower is the scan, the more we have to deal with underlying 1/f noise.

Step 1: build instrument model

```
obs = PacsObservation(['myobs1.fits', 'myobs2.fits'])

O = ConvolutionOperator(obs.get_psf(...))
P = ProjectionOperator(obs.get_pointing_matrix())
R = ConvolutionTruncatedExponentialOperator(...)
C = CompressionAverageOperator(8)
G = DiagonalOperator(..., broadcast='rightward')
M = MaskOperator(...)
H = M * G * C * R * P * O
```

Step 1: build instrument model

Tamasis can do plain HIPE's MADmap, too:

```
P = ProjectionOperator(obs.get_pointing_matrix(method='nearest',
                                               downsampling=True))
M = MaskOperator(...)
H = M * P
```

Step 1: build instrument model

Tamasis can do vanila HIPE's Photproject, too:

```
obs = PacsObservation(['myobs1.fits', 'myobs2.fits'])
y = obs.get_tod()
y = filter_median(y, 100)

P = ProjectionOperator(downsampling=True)
M = MaskOperator(y.mask)
H = M * P

m = H.T(y) / H.T(ones(y.shape))
```

Step 2: Objective Function

$$\boldsymbol{f}(\boldsymbol{x}) = \Phi(\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}) + \Psi(\boldsymbol{x})$$

- Likelihood (with inv. covariance matrix N^{-1} as an operator)
- prior: regularisation
  - smoothness: $\Psi(\boldsymbol{x}) = \lambda \, ||D_\alpha(\boldsymbol{x}) + D_\beta(\boldsymbol{x})||_2^2$
  - sparseness: $\Psi(\boldsymbol{x}) = \lambda \, |w(\boldsymbol{x})|_1$
  - positivity constraint
- norms: L_1, L_2, L_p, L_{Huber}

For a linear model, it can be equivalent to solving Ax=b

- least-square:

```
A = H.T * invNtt * H,    b = H.T * invNtt * y
```

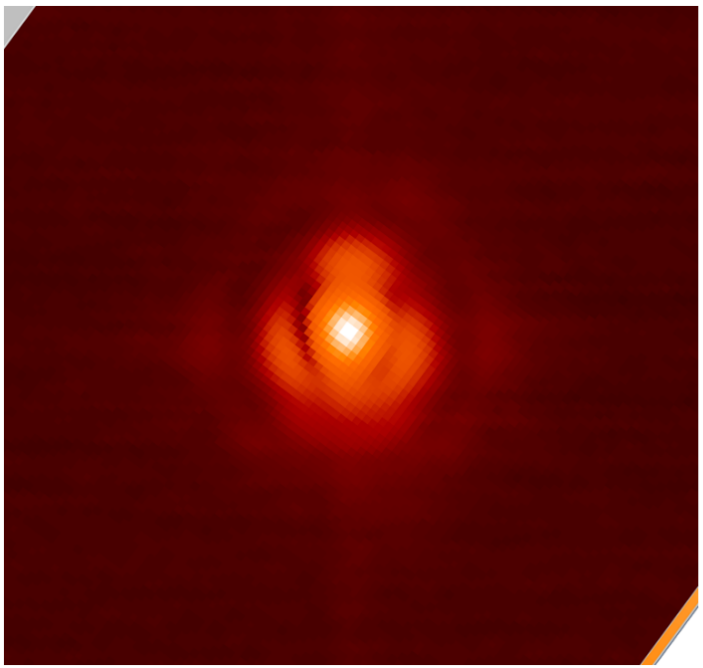
- least-square + prior:

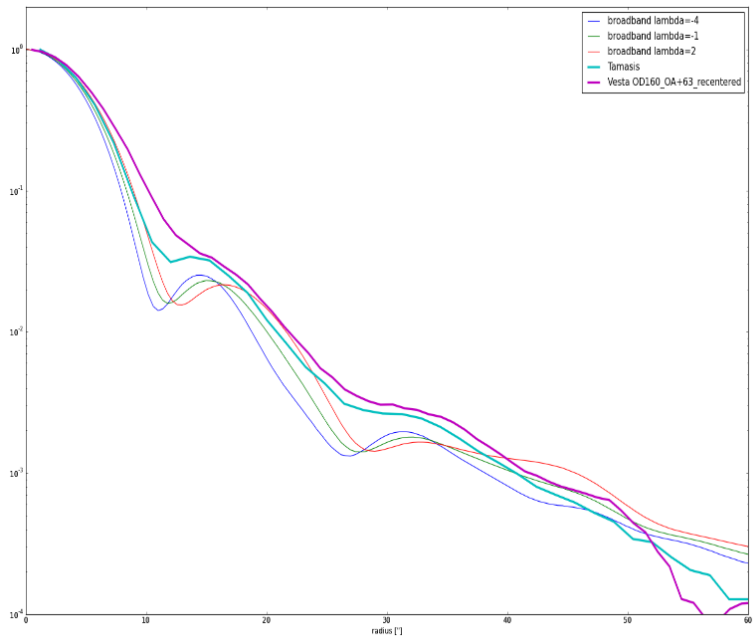
```
A = H.T * invNtt * H + h * D.T * D,    b = H.T * invNtt * y
```

Step 3: Minimisation

- PCG (MPI)
- non-linear PCG (MPI)
- scipy.sparse.linalg (CG, CGS, BICG, BICGSTAB, GMRES, LGMRES, ...)
- scipy.optimize (fmin_bfgs, ...)
- FISTA
- soon: PETSc

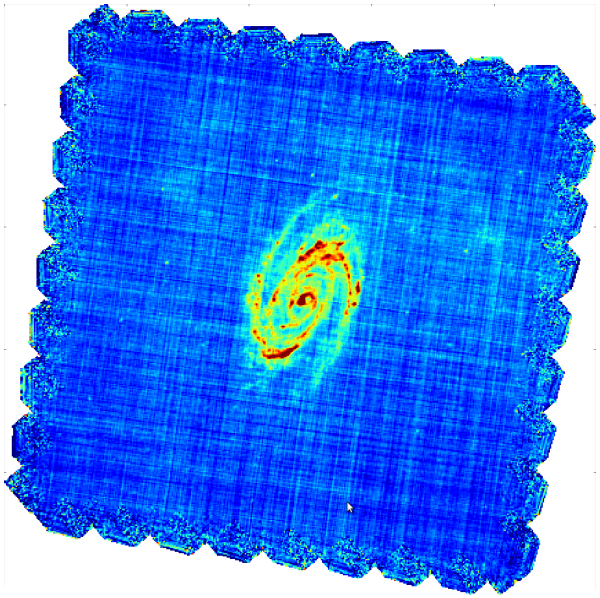
High frequency reconstruction



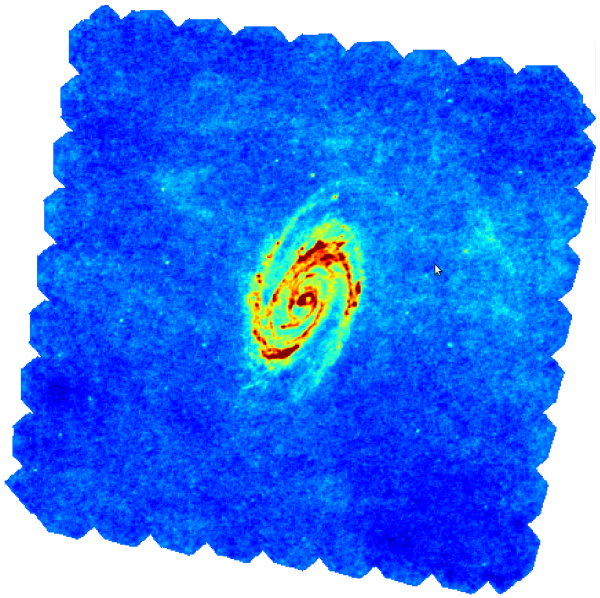


Red channel PACS PSF

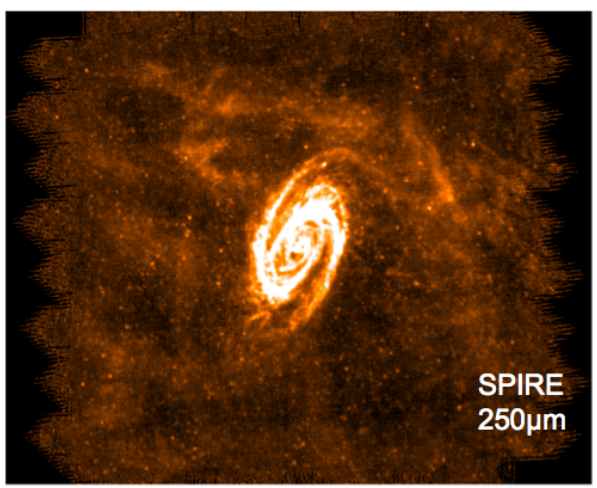
Low frequency reconstruction



“Pipeline” without hpf

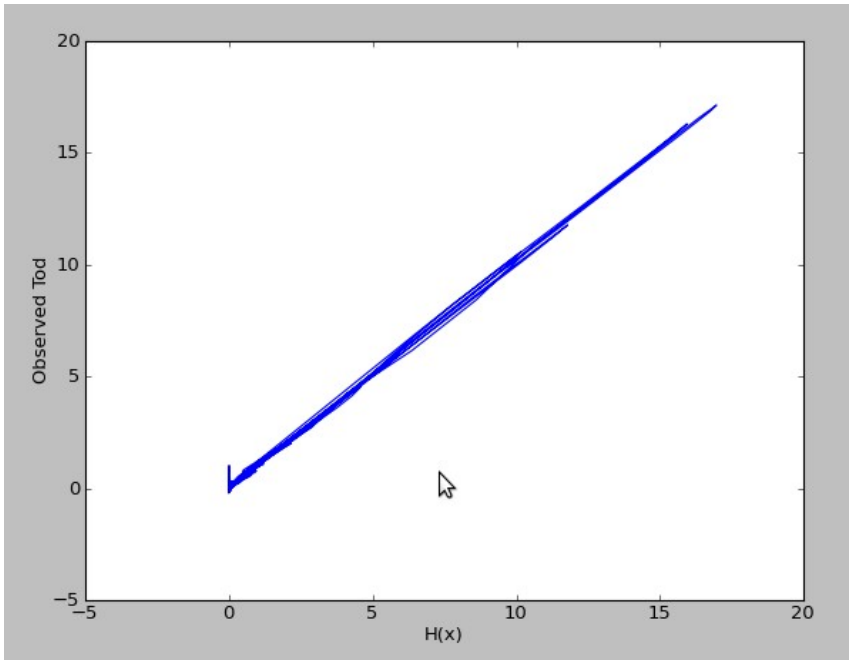


Tamasis RLS

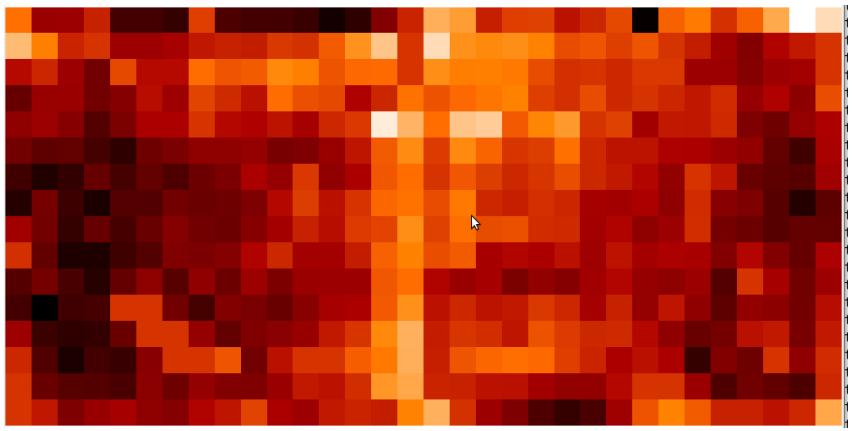


- Removed artifacts
- Real structures

Flat reconstruction

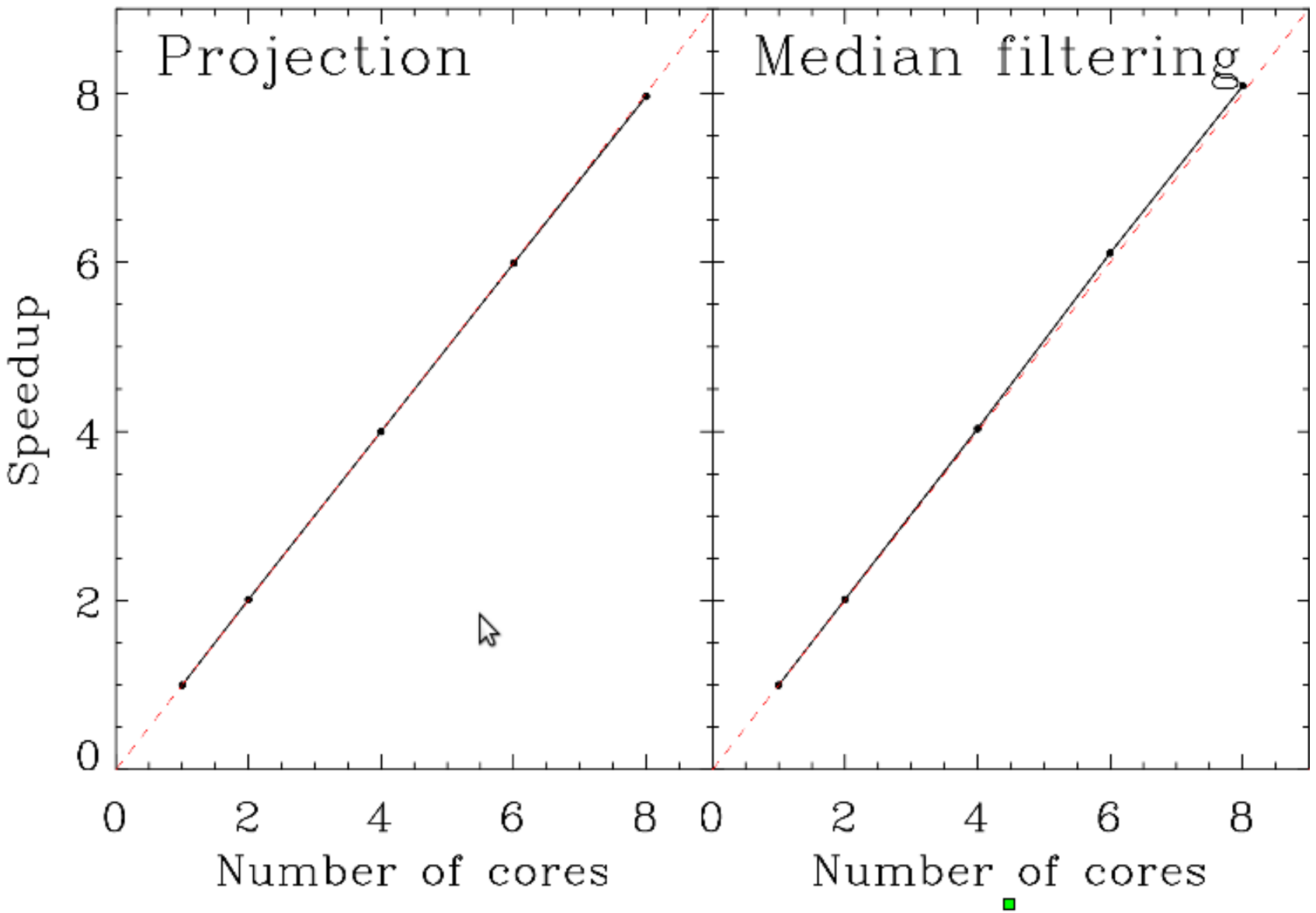


simulation vs observation

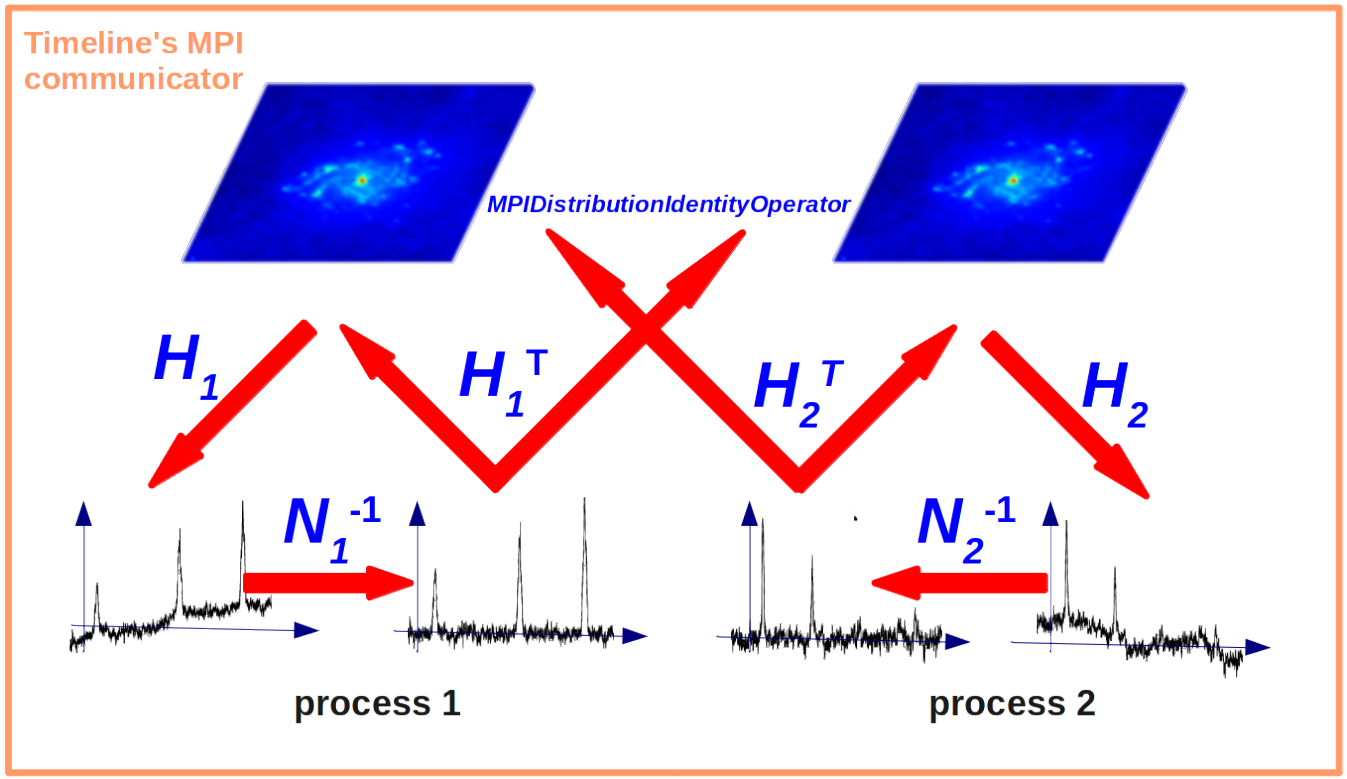


Resulting flatfield

Parallelisation OpenMP



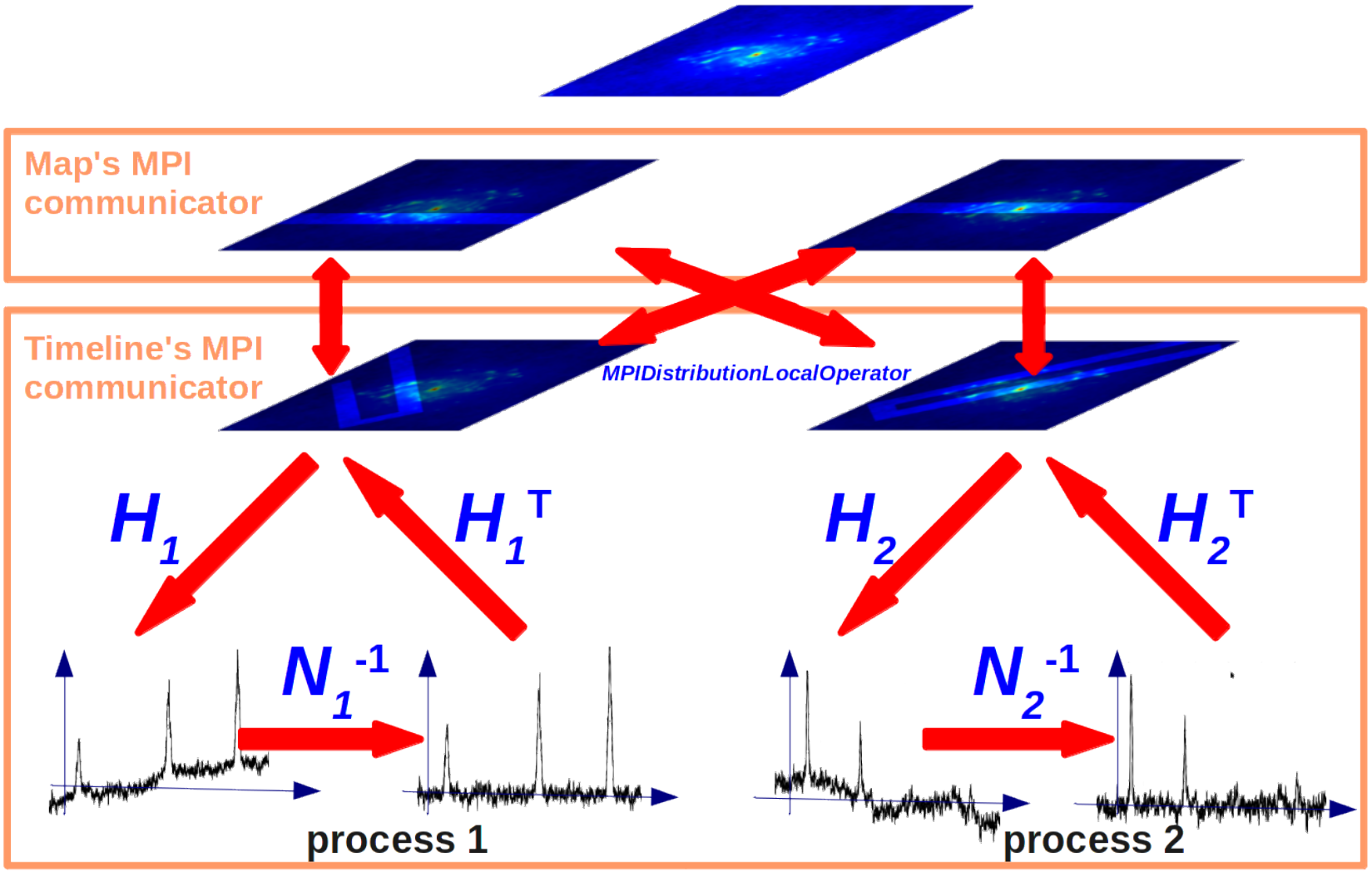
Parallelisation MPI: Distributed TOD



$$\boldsymbol{H}\boldsymbol{D} \Rightarrow \boldsymbol{D}^{\mathrm{T}}\boldsymbol{H}^{\mathrm{T}}\boldsymbol{N}^{-1}\boldsymbol{H}\boldsymbol{D}\boldsymbol{x} = \boldsymbol{D}^{\mathrm{T}}\boldsymbol{H}^{\mathrm{T}}\boldsymbol{N}^{-1}\boldsymbol{y}$$

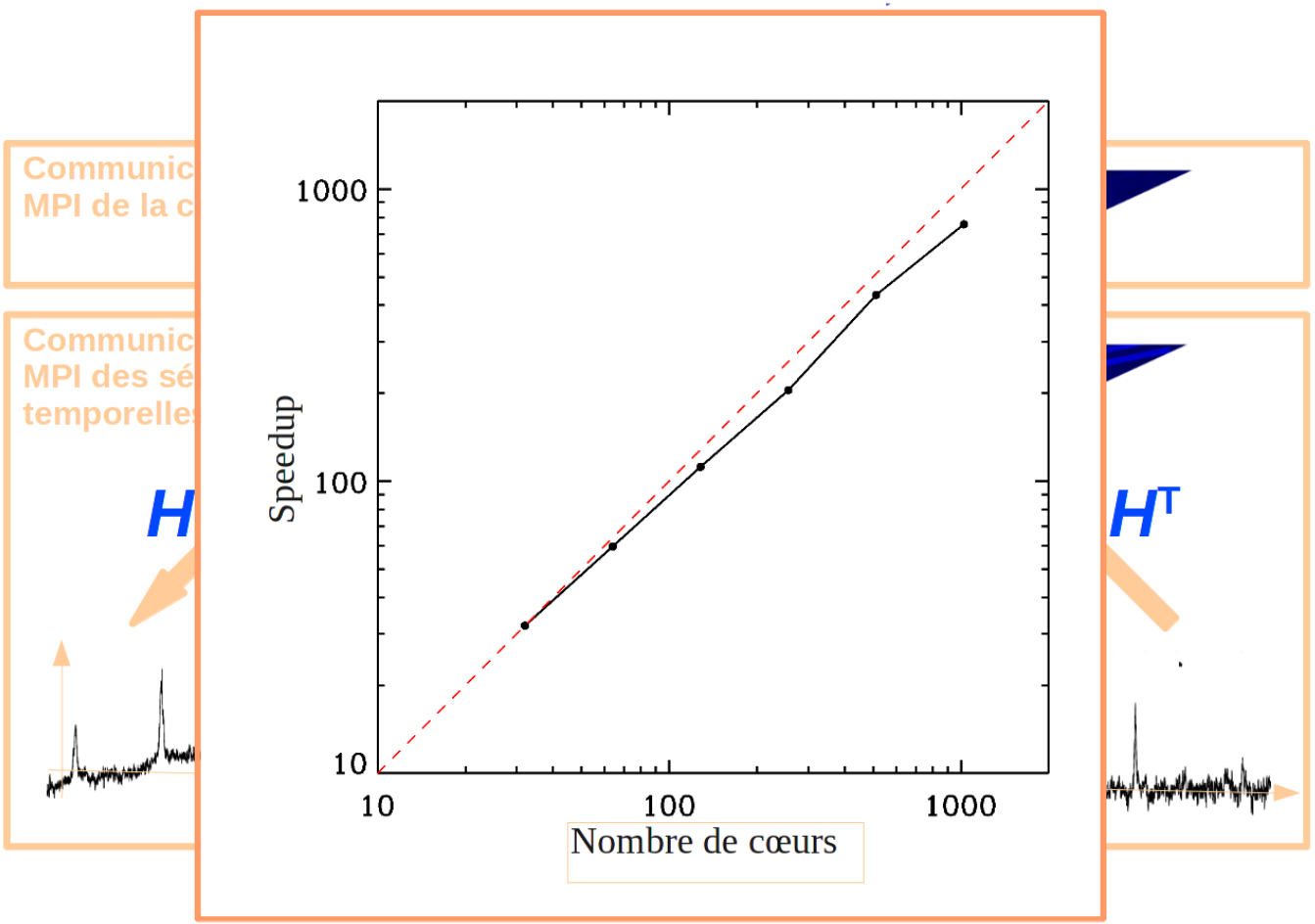
$$\boldsymbol{H} = \begin{pmatrix} H_1 & 0 & \cdots & 0 \\ 0 & H_2 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & H_p \end{pmatrix} \quad \boldsymbol{D} = \begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{I} \\ \vdots \\ \boldsymbol{I} \end{pmatrix}$$

Parallelisation MPI: Distributed TOD and map

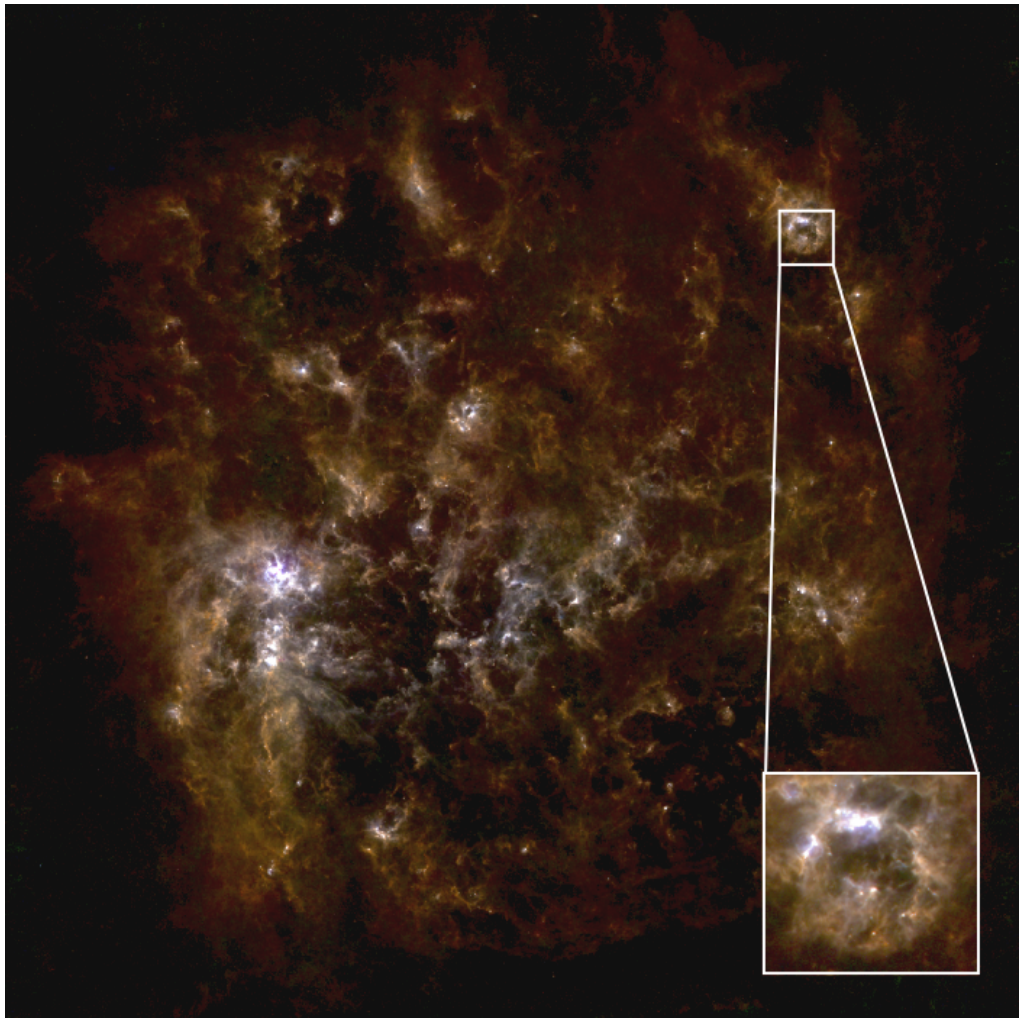


$$\boldsymbol{H}\boldsymbol{D} \Rightarrow \boldsymbol{D}^{\mathrm{T}}\boldsymbol{H}^{\mathrm{T}}\boldsymbol{N}^{-1}\boldsymbol{H}\boldsymbol{D}\boldsymbol{x} = \boldsymbol{D}^{\mathrm{T}}\boldsymbol{H}^{\mathrm{T}}\boldsymbol{N}^{-1}\boldsymbol{y}$$

Parallelisation MPI: Distributed TOD and map



Tamasis: large dataset (LMC)



- Map: $9.2^\circ \times 9.0^\circ \approx 3$ GB ($4\,10^8$ unknowns)
- TOD: 161h, 2048 detectors, $5.2\ 10^9$ samples at 5 Hz, $4.2\,10^{10}$ at 40 Hz.
- Pointing matrix: 3.3 TB (1 detector intersects 10 sky pixels)
- Hardware: 50min, CCRT/Curie, 2432 cores, 152 processors

Future work

- With decent pointing reconstruction:
  - Include telescope PSF *Rightarraow* deconvolution
  - Fix cross-artifacts in a natural way
- Integrate Panuzzo's preprocessing treatment
- Implement SUPREME's hyperparameter handling (next talk). For the workshop, a single value has been chosen for all observations.
- Ease installation

Outstanding issue: pointing reconstruction