PyXim – A fast, GPU accelerated, differentiable HxRG simulation framework

Leander Lacroix
llacroix@ip2i.in2p3.fr
Institut de Physique des 2 Infinis, Lyon, France

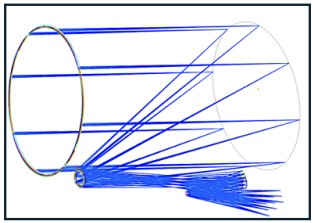
IRIS workshop 2026 @ INAF OAS Bologna, Italy

Context

The Lazuli Space Observatory

The Eric and Wendy Schmidt Observatory System

Telescope

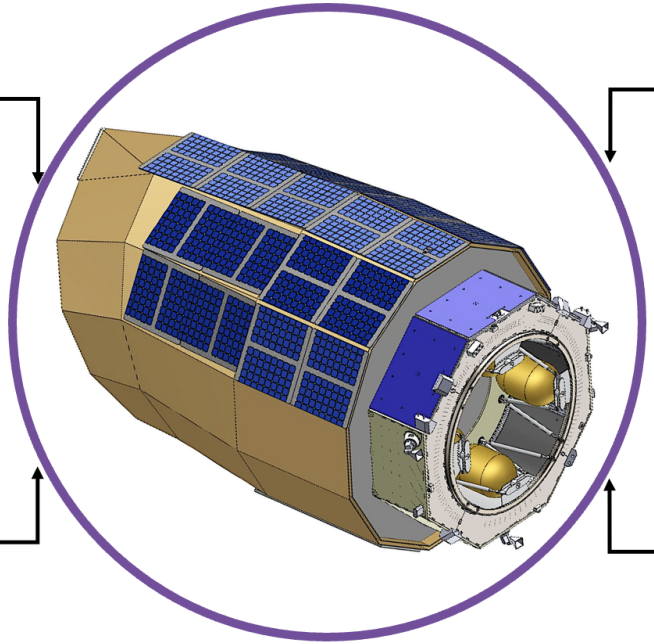


Diameter: 3m
Design: Off-axis TMA
Control: Active Jitter & Alignment Control

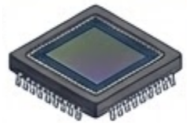
Integral Field Spectrograph (IFS)

Wavelength range: 400–1700nm
Spectral resolution: 100-500
Two parallel fields:

- Narrow: 2.3x4.6″ FoV sampled at 40mas/pix
- Wide: 4.6x8.8″ FoV sampled at 80mas/pix

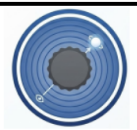


Widefield Context Camera (WCC)



Imaging: Diffraction limited at 633nm
Sensors: 23 large format CMOS sensors
FoV: 35′ x 12′
Filters: Broad + narrow band, covering 350-1000nm

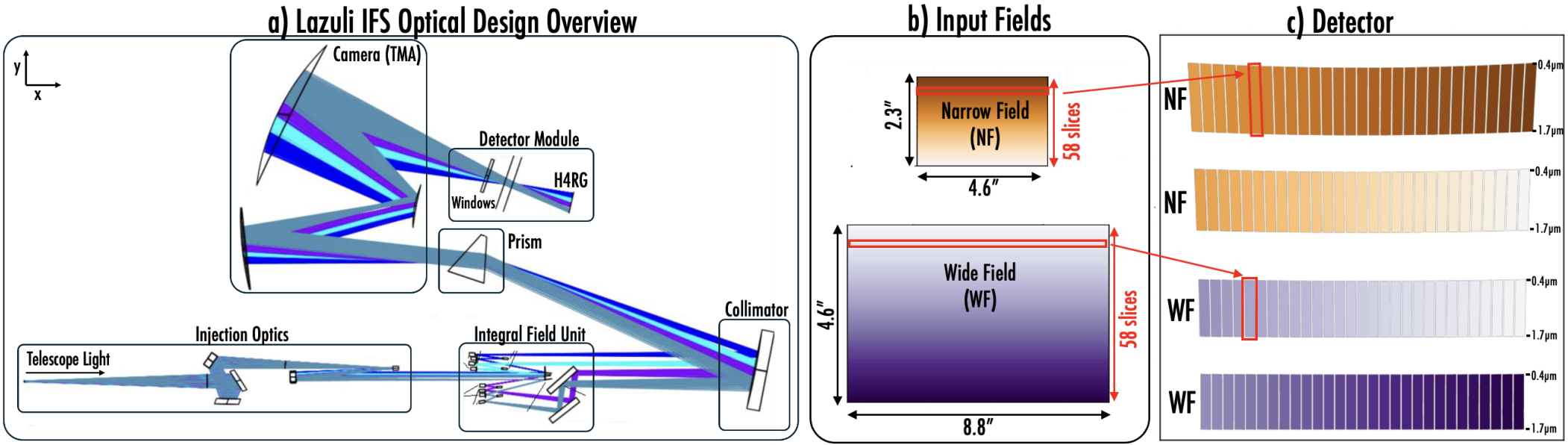
ExtraSolar Camera (ESC)



Method: Focal plane vector-vortex coronographs
Wavelength range: 400-750nm
Wavefront control: Active deformable mirrors
Contrast: 10^{-8}-10^{-9} (post processed)

(figure taken from Roy et al., 2026)

Integral Field Spectrograph



(figure taken from Roy et al., 2026)

Detector used: **H4RG-10**, HgCdTe w/ $\lambda_{cutoff} = 1.7\ \mu m$ @ 120 K

Our team in Lyon: calibration & flux inference pipeline

Lead by M. Rigault

PyXim

- Specialized CMOS simulation framework:
  - ROIC → HxRG
  - Absorber → HgCdTe (could extend to HyViSI)
  - ASIC → SIDECAR/ACADIA/whatever
  - Sample-Up-the-Ramp mode only (no CDS nor guide mode)
- Written in Python/JAX
  - Just-in-time (JIT) compilation: *near-native performance*
  - Accelerator support (GPUs): *massive parallelism*
  - Automatic differentiation: *differentiable modeling*

Goals:
→ Test/prepare flux inference pipeline
→ Test/prepare detector characterization campaign
→ Full forward modeling fit

JAX & JIT

NumPy

- Eager execution
- Pure precompiled C ufuncs (tight loops, BLAS/LAPACK)
- Minimal runtime overhead
- Ubiquitous & simple

Jax – w/ JIT

- Delayed execution (tracing)
- Graph & operation fusing
- Compilation to native binary (via OpenXLA → LLVM)
- **Removal of Python from the hot path**

Pros: Optimized & parallelized computations → Perfect for APS simulations

Cons: 🔪 JAX - The Sharp Bits 🔪

NumPy optimizes *individual* operations
JAX optimizes *whole* programs

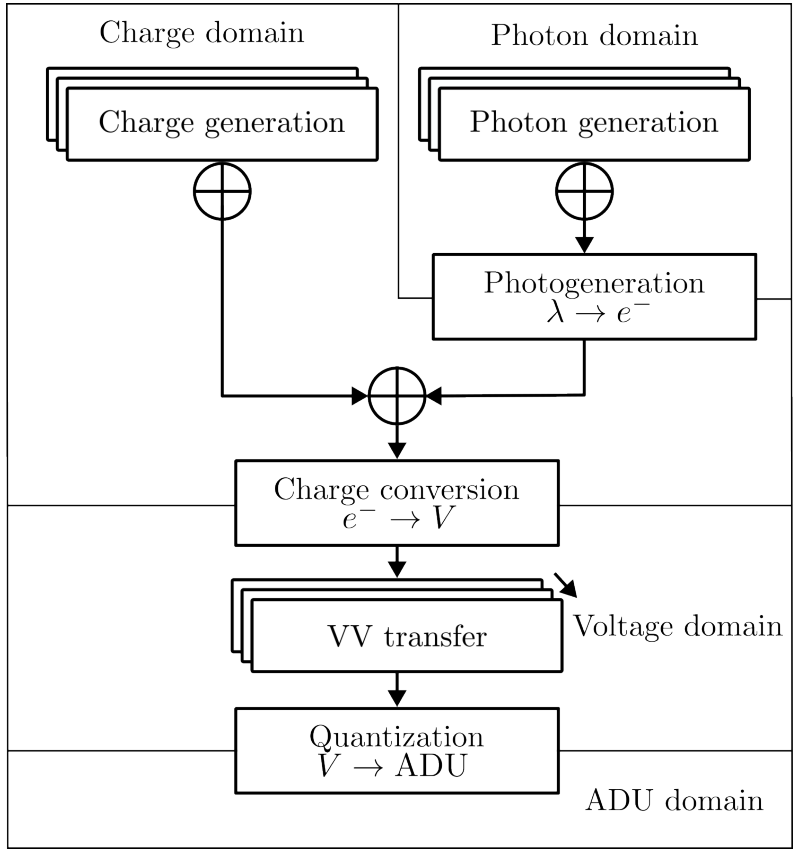
PyXim – Simulation flow

4 domains:
- Photon
- Charge
- Voltage
- ADU

Models:

→ Source
- Photons
- Charges

→ Transfer
- Domain → domain
  - → Photogeneration
  - → Charge conversion
  - → Quantization
- V → V



Simulation flow wo/ BFE

Model definition

Declarative

```toml
[Photogeneration]
model_name = 'FPNPhotogenerationModel'
shot_noise_model = 'Poisson'

[Photogeneration.fpn]
init = 'from_gaussian_dist'
resolution = [1016, 1016]
seed = 1337
Pn = 0.01

[[Illumination]]
label = 'flat'
model_name = 'FlatfieldIlluminationModel'
flux = 3200.0

[[ChargeGeneration]]
label = 'dark'
model_name = 'DarkCurrentModel'
rule = 'rule22'

[[VVTransfer]]
label = "dc_offset"
model_name = 'DCOffsetModel'
enabled = true
dc_offset = 0.01

# kTC noise
[[VVTransfer]]
enabled = true
label = 'reset_noise'
model_name = 'ResetNoiseModel'
seed = 0
```

Imperative

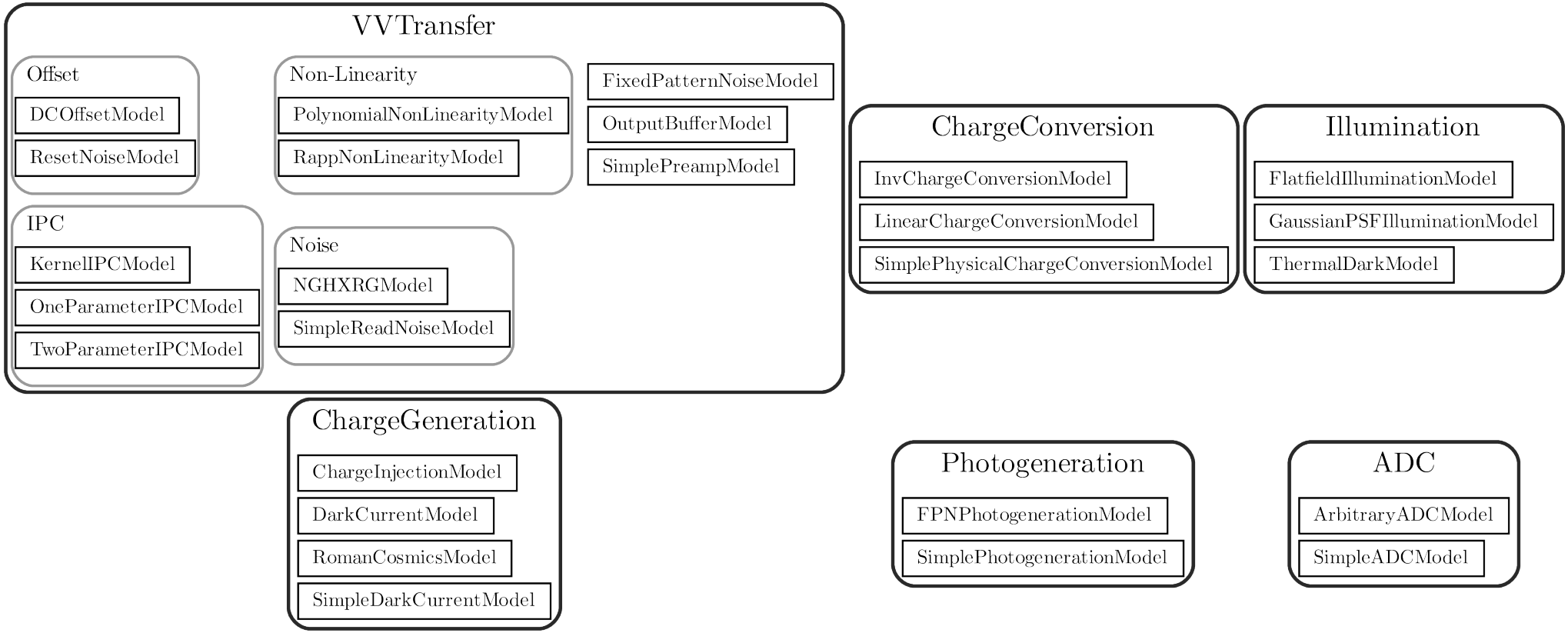
```python
# Define an illumination model, here a simple flatfield with around 800 incident photon/seconds
illum_model = FlatfieldIlluminationModel(flux=850.)

# Add 10 e- readout noise
readout_noise_model = SimpleReadNoiseModel(noise_std=10.*sensor.sn_gain)

# Add an ad-hoc 100 e- DC offset
dc_offset_model = DCOffsetModel(dc_offset=100.*sensor.sn_gain)

# Our FPN-modulated photogeneration model
photogeneration_model = FPNPhotogenerationModel(
    fpn=FixedPatternNoise.from_gaussian_dist(0.01, sensor.resolution, key=jax.random.key(1337)))
```

2 equivalent definition modes:

- Declarative – From text file (.toml)
- Imperative – From code (Python)

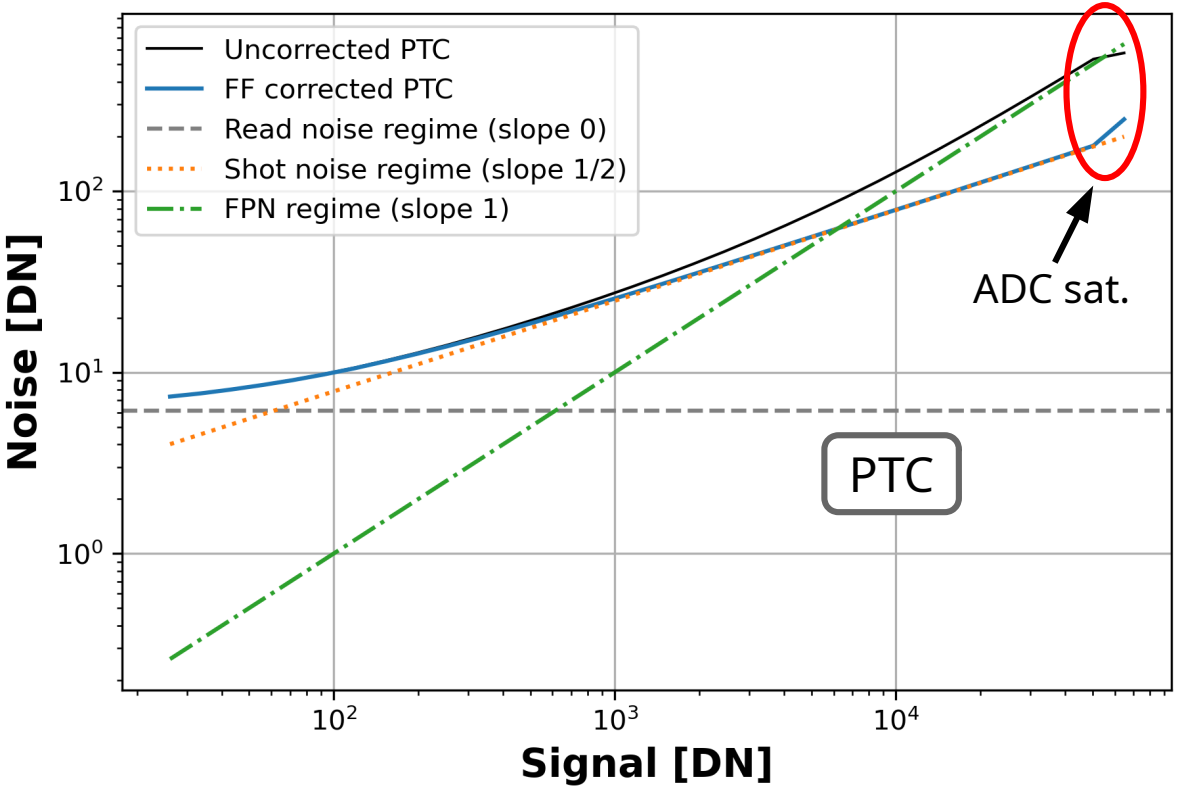
Both exposes the **same** set of functionality

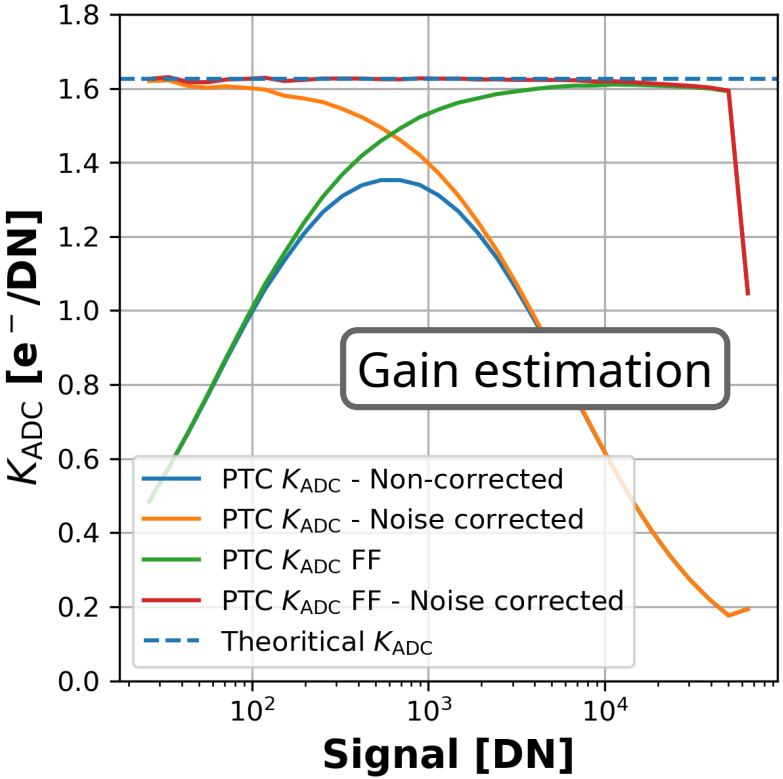
Model catalog



Plugin architecture w/ lazy loading

Feature gallery – Sanity check

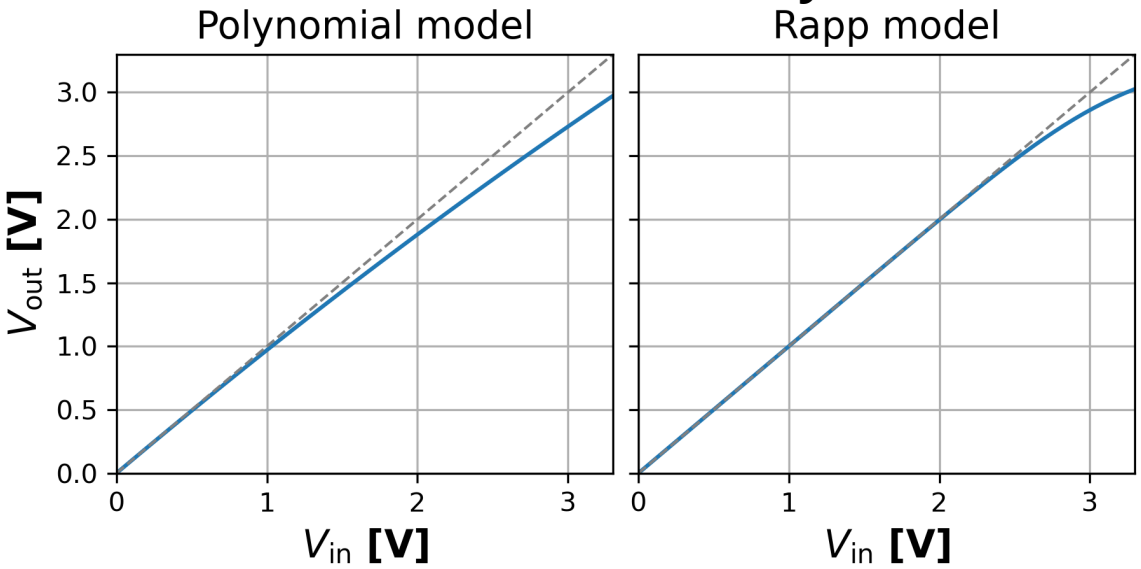




Simple model: readout noise + FPN → follows theory perfectly!

Feature gallery – Classical nonlinearity

V/V nonlinearity

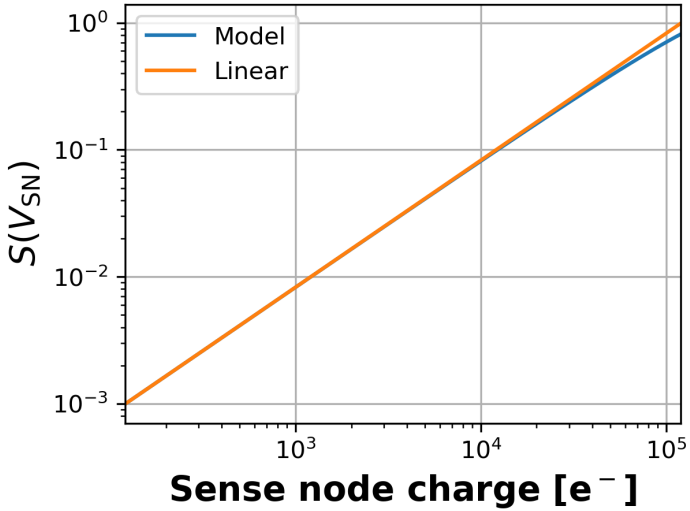


4 granularity:

- Per pixel
- Per column
- Per channel
- Frame

→ Composable

V/e⁻ nonlinearity

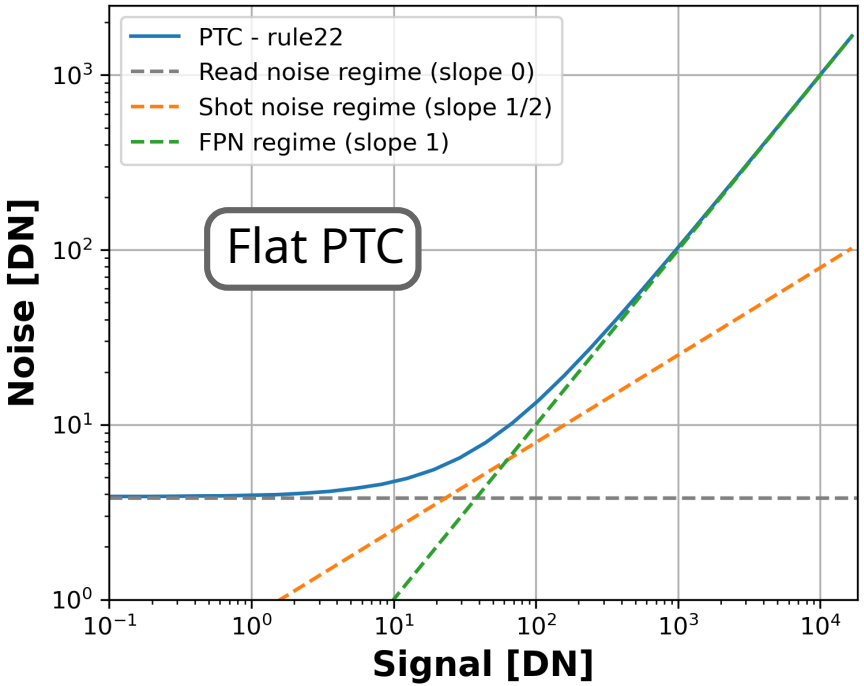


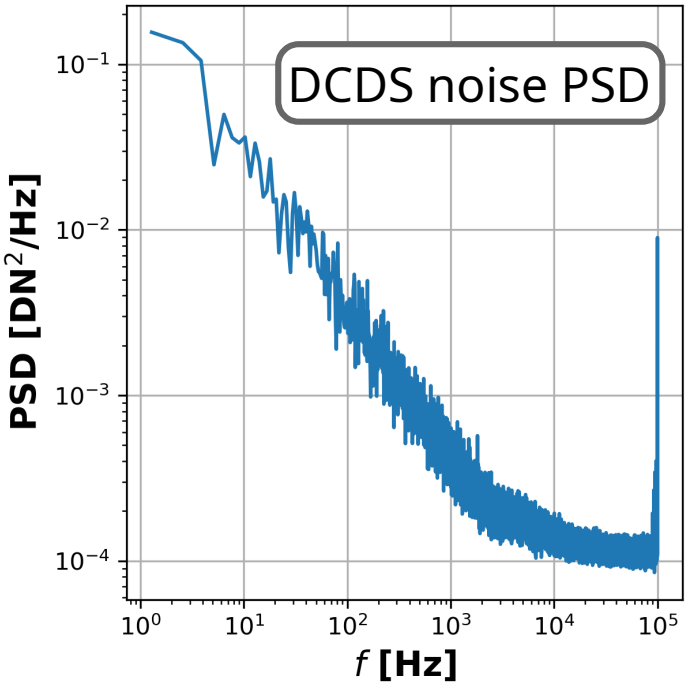
→ Transimpedance amplifier NL

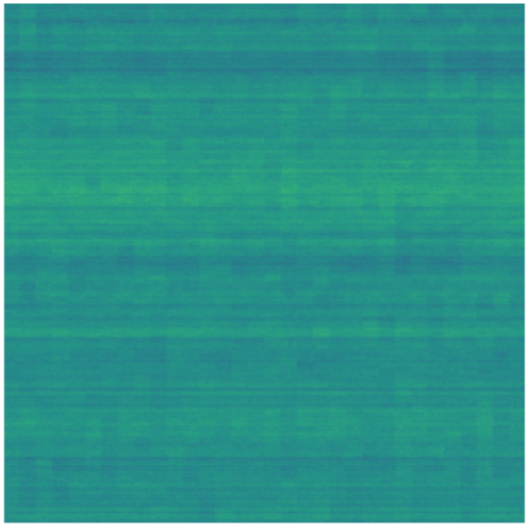
This plot:

- Small signal model
- Assumes abrupt P^+N junction

Feature gallery – Dark + NGHXRG



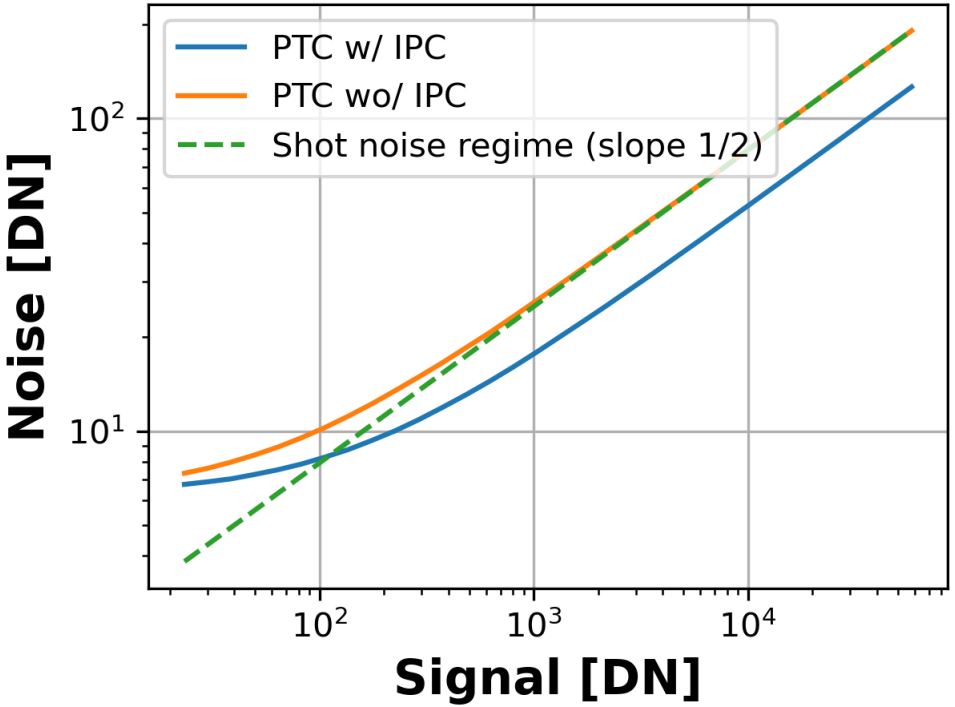




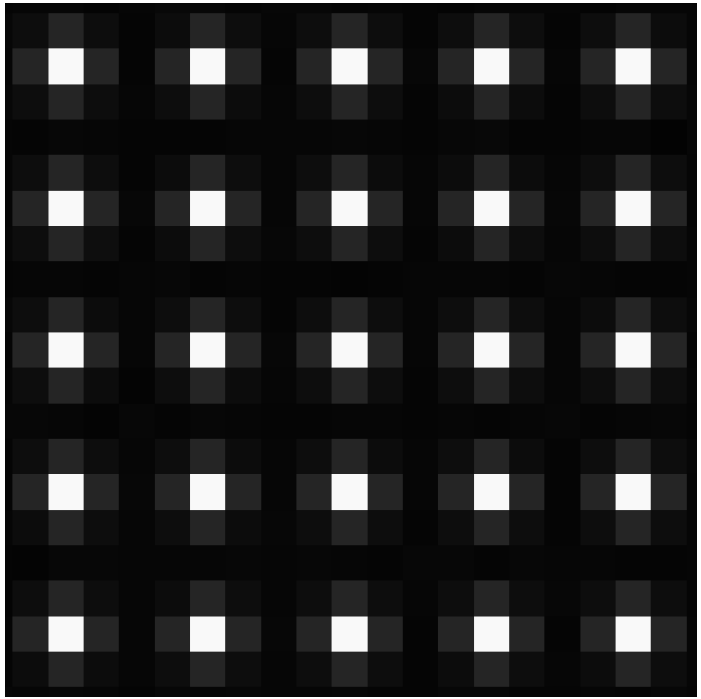
DCDS frame
(4k – 32 chan.)

- Dark model: Rule07 & Rule22 + DFPN
- NGHXRG (Rauscher, 2015) adapted into PyXim

Feature gallery – Inter Pixel Capacitance

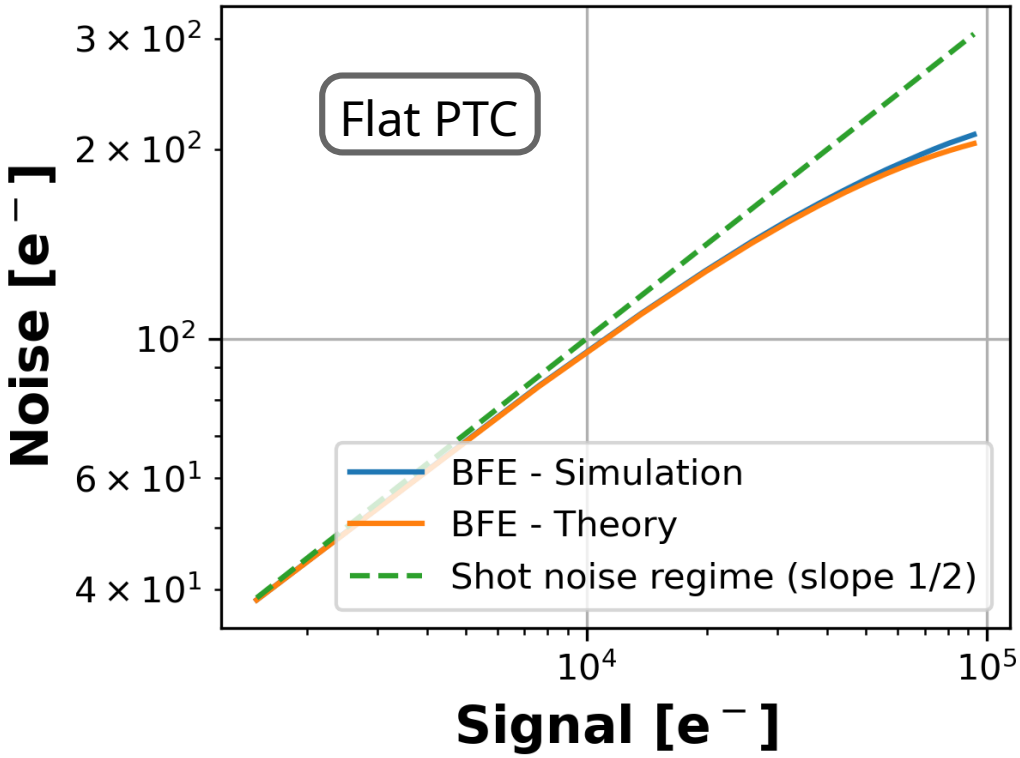


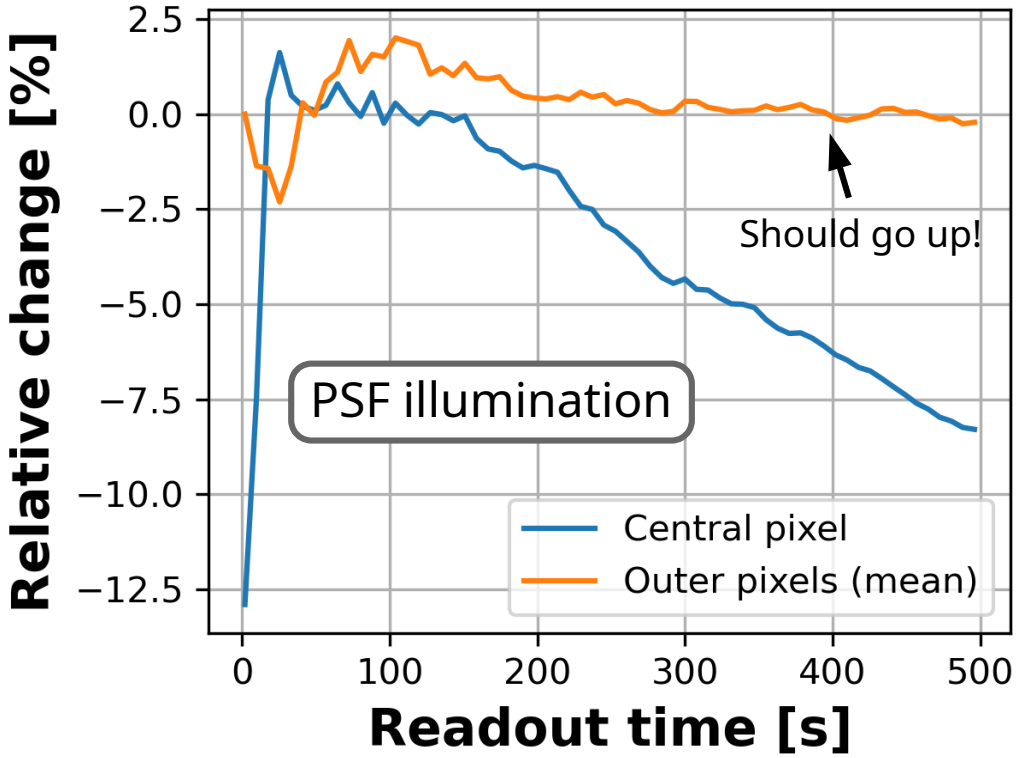
PTC variance deficit due to IPC (exaggerated kernel)



V_{RESET} point grid to estimate IPC kernel

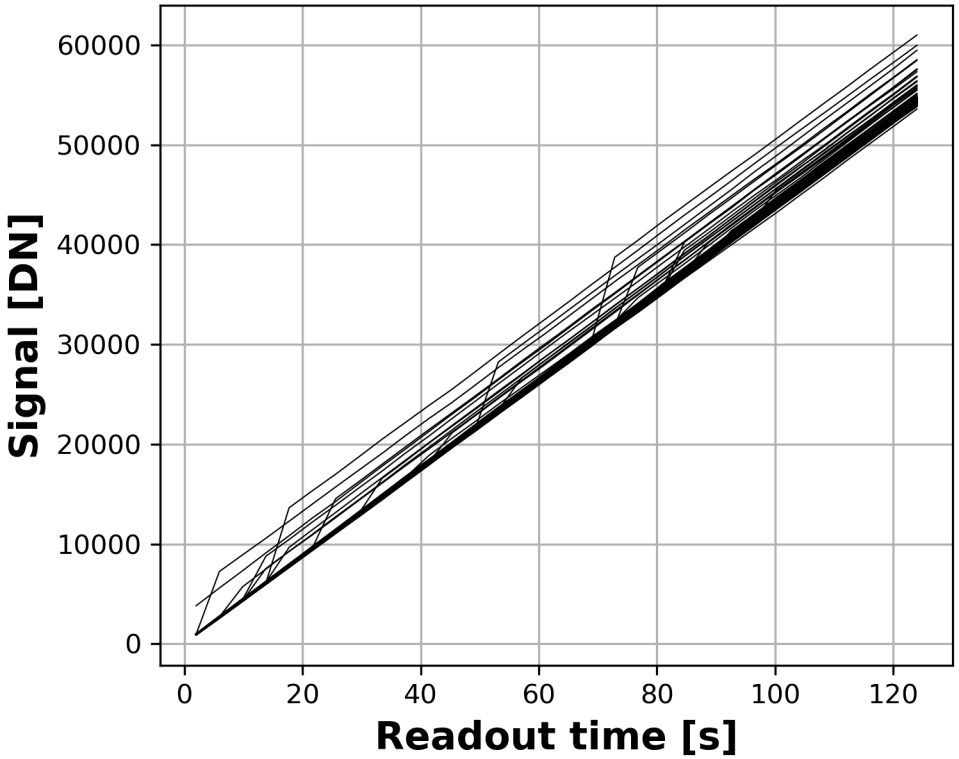
Feature gallery – Brighter-Fatter effect

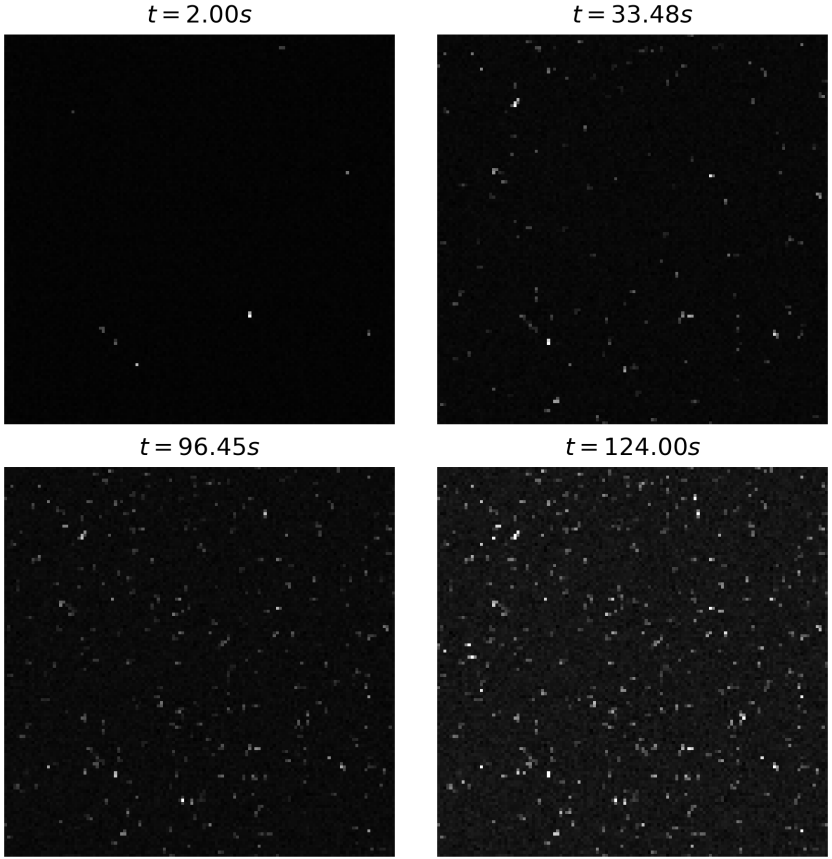




→ Flatfield formulation only! (Astier et al., 2019)
As in Solid-Waffle (Hirata & Choi, 2020)

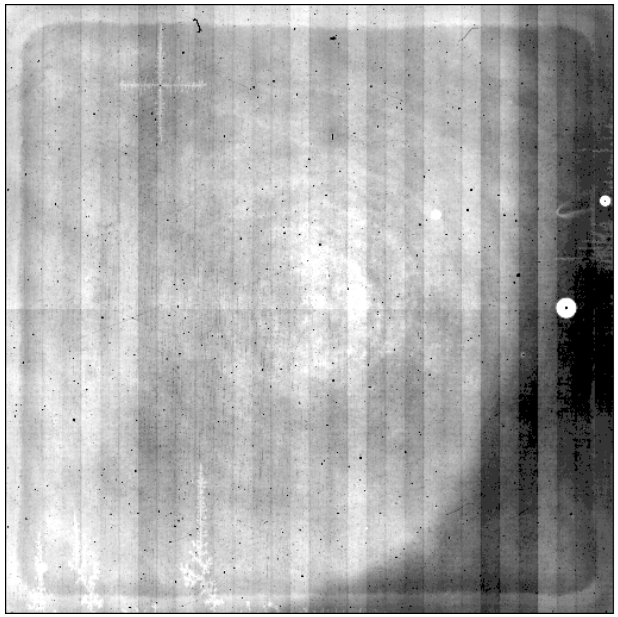
Feature gallery – Cosmics



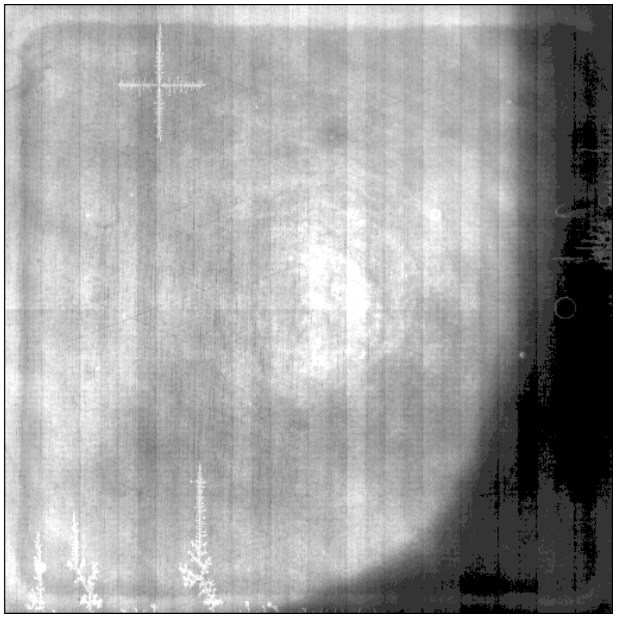


Adapted from Roman-STScI-000502

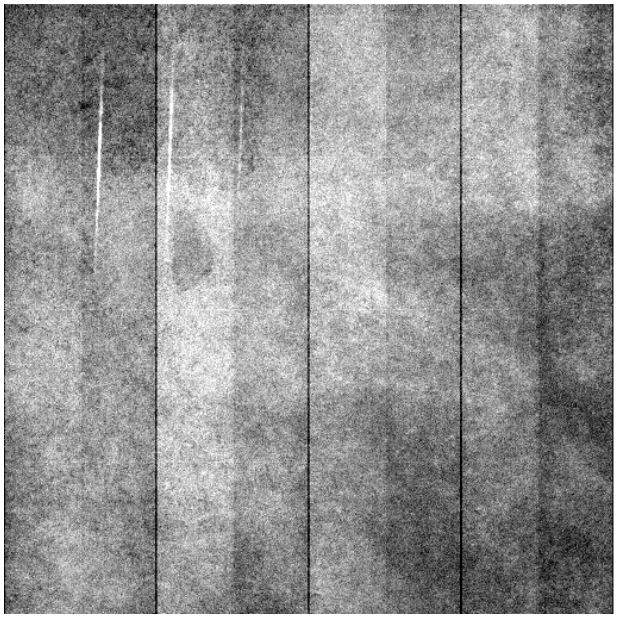
Feature gallery – Lazuli IFS



Roman flat



PyXim flat

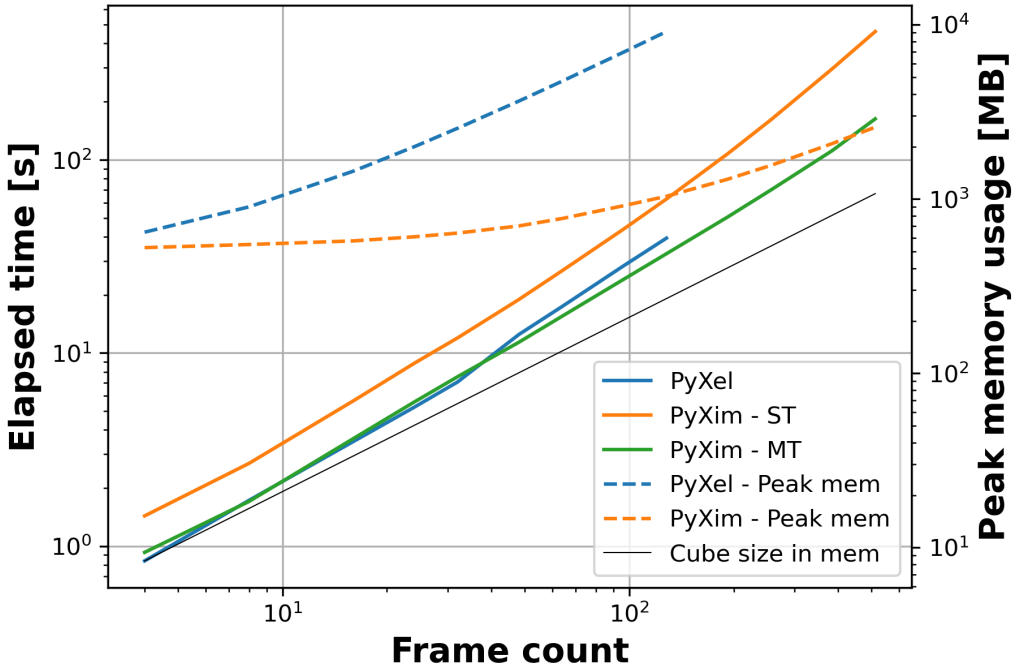


PyXim Lazuli IFS (magnified)

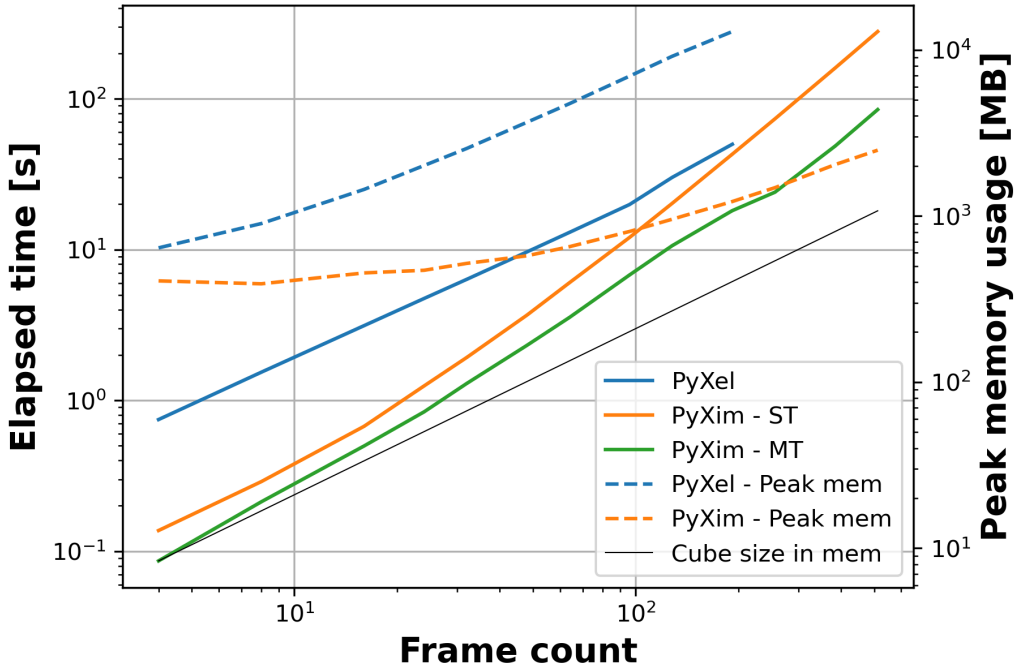
→ Offset & FPN components extracted from Roman (public WFI Triplet Test Data)

Comparison to PyXel – Simple model

Shot noise model: Poisson



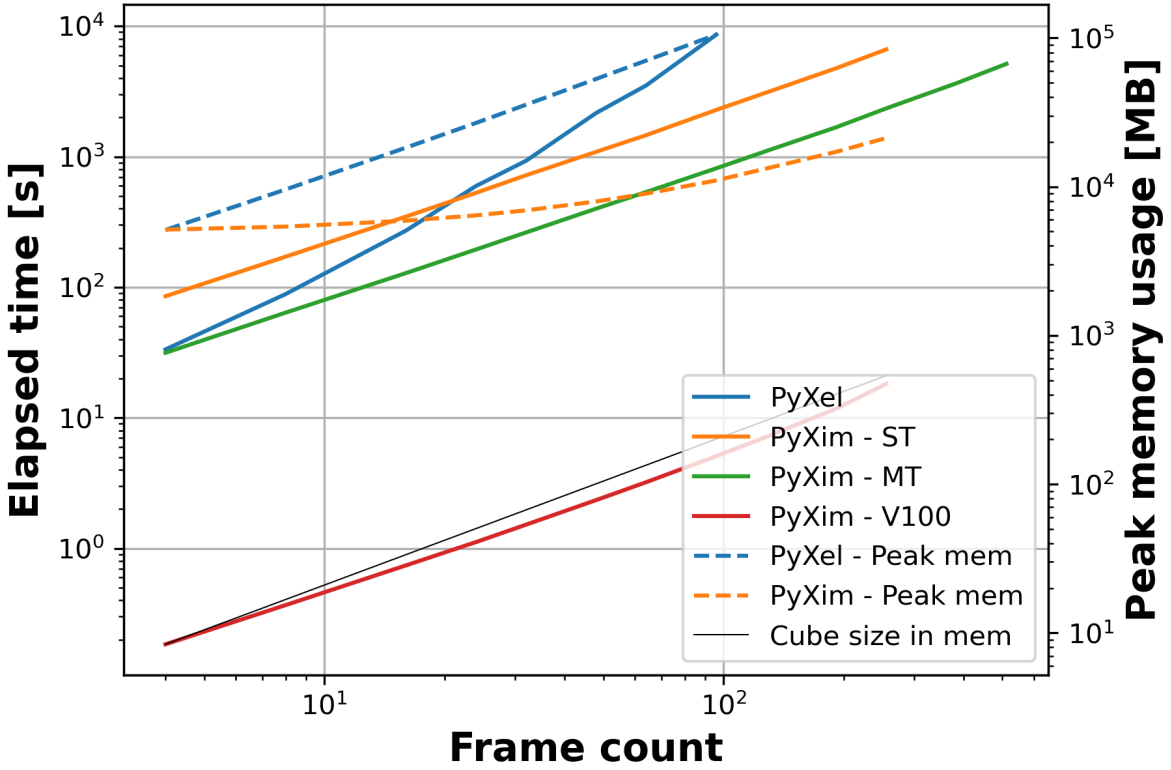
Shot noise model: Normal



Simple model:
- 1024 x 1024
- FPN
- Gaussian read-out noise

Hardware used:
Intel Core i7-1165G7 (2.8 GHz) – 4C/8T
(PyXel version: 2.16.2)

Comparison to PyXel – Complex model



Complex model:
- 4096 x 4096
- Dark (rule 07)
- Reset noise
- Polynomial non-linearity
- NGHXRG noise mode (32 channels)

→ On V100: 256 frames in 18s

Hardware used:
- Intel Xeon Silver 4210R (2.40 GHz) – 10C/20T
- Nvidia V100 32 GB

Forward modeling

Classic inference pipeline
- Characterize non-ideal behaviors
- Correct raw from them
- Fit for flux

Forward modeling
- Define model
- Fit model nuisance parameter spaces
- Fit for flux

- JAX: Automatic Differentiation (gradient, Jacobian, Hessian)
  - Gradient: 1 model evaluation
  - Enables classic ML optimization algorithms
    (Conjugate Gradient, BFGS, ...)
- Forward modeling fit → non-linear maximum likelihood estimation

Pros:
- Automatically account for bias sources
- Fisher matrix

Cons:
- Model dependent
- Computationally expensive

Breaking one degeneracy at a time

Breaking degeneracies calls for different datasets:

- V_{RESET} sweeps (up to V_{DSUB})
  → SF/column/output buffer/preamp transfer functions
- V_{RESET} point grid → Characterize IPC
- Flat ramp to saturation → BFE, full well
- Shorted inputs → Preamp noise
- Dark flats → Dark contribution, noise, amp/mux glow
- Low THD sinusoidal in ASIC → ADC nonlinearities
- CV PN-junction relationship → Photodiode characterization

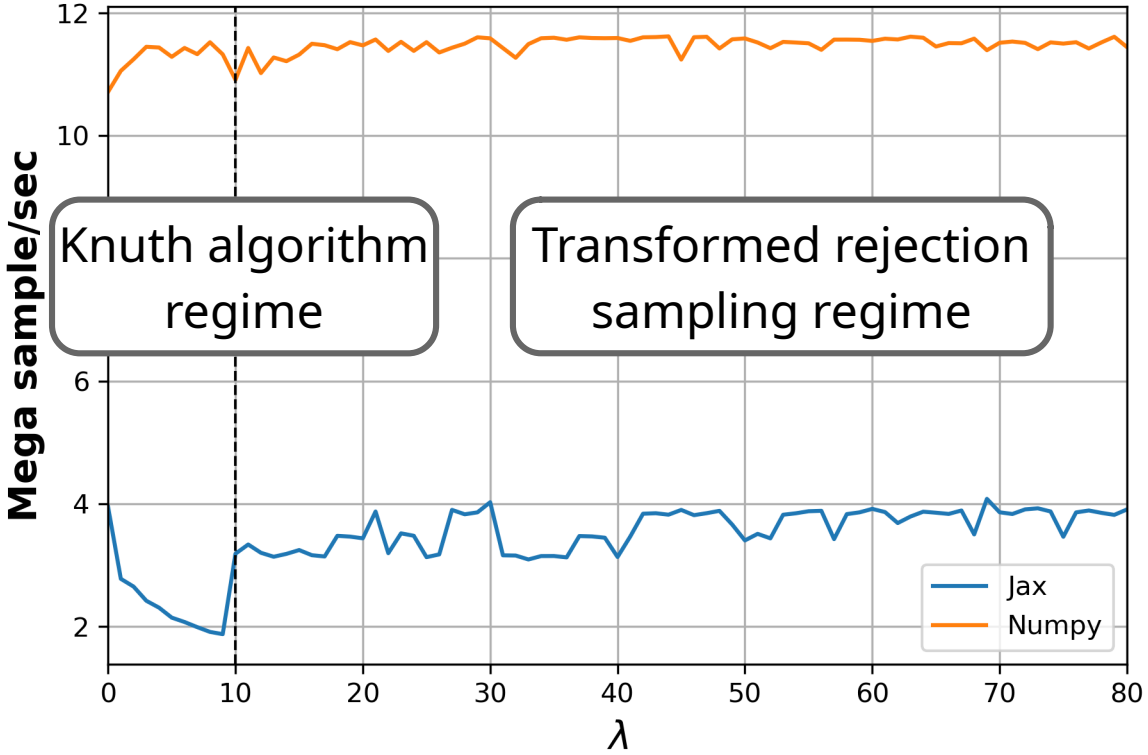
From there: fit for flux! (with nuisance parameters fixed)
→ Forward modeling

The future

- Extend model catalog
  - Crosstalk
  - Persistence/burn in/count rate non-linearity
  - ADC nonlinearities
  - Better sense node transimpedance amplifier models (saturation)
  - Pixel cosmetics
- Start experimentation with forward modeling
- **Write paper & release code!**

Backup slides

Poisson sampling benchmark



JAX: ~3/4 times **slower** than NumPy!

- Stochastic loop
- Branch heavy
- Key splitting cost

Hardware used: Intel Core i7-1165G7 (2.8 GHz) – 4C/8T