Simulation des détecteurs H4RG de HARMONI avec le cadriciel Pyxel de l'ESA+ESO

Antoine Kaszczyc
CRAL, Centre de Recherche en astrophysique de Lyon

HARMONI

- spectrographe intégral de champ
- proche infra rouge
- installé sur l’ELT (Extremely Large Telescope)
- utilise 8 détecteurs CMOS H4RG

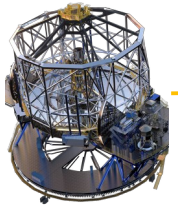
objectif long terme au CRAL pour HARMONI :

un logiciel de réduction
et un logiciel de simulation

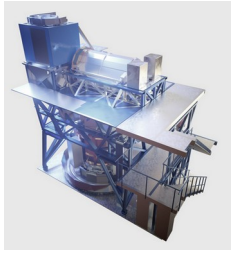
scene simulation

HINM pipeline de simulation

real light

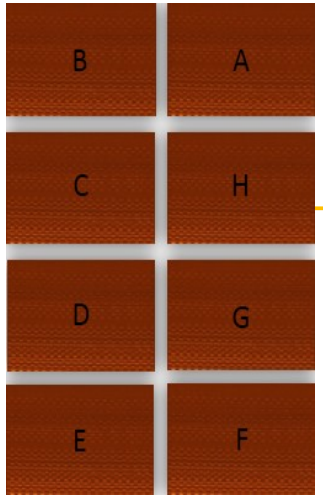


ELT telescope
Atacama, Chile



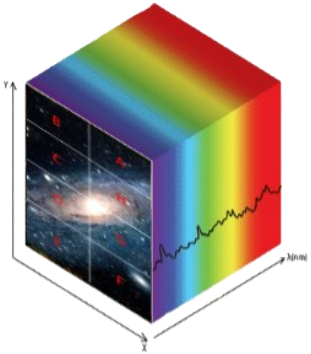
HARMONI instrument

pictures from eso.org



Raw FITS file

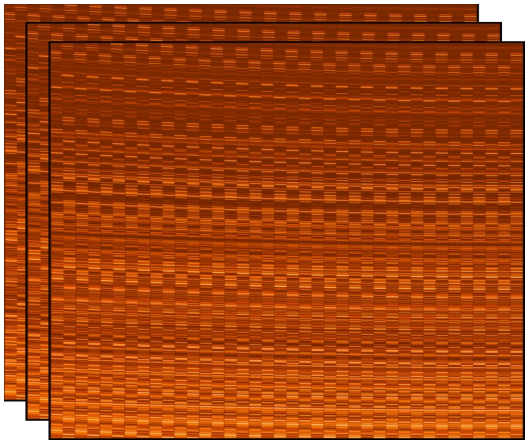
pipeline de réduction



output cube
[X,Y,λ]

objectif long terme au CRAL pour HARMONI :

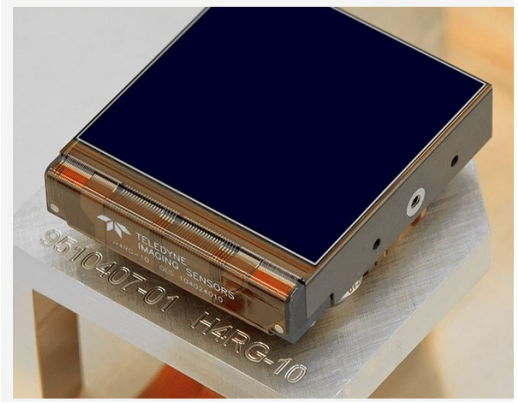
HINM, jumeau digital de HARMONI



- simule la propagation de la lumière dans l’instrument
- jusqu’au détecteurs H4RG et la numérisation du flux de photons en ADUs
- créé des fichiers FITS de même forme que “les vrais”

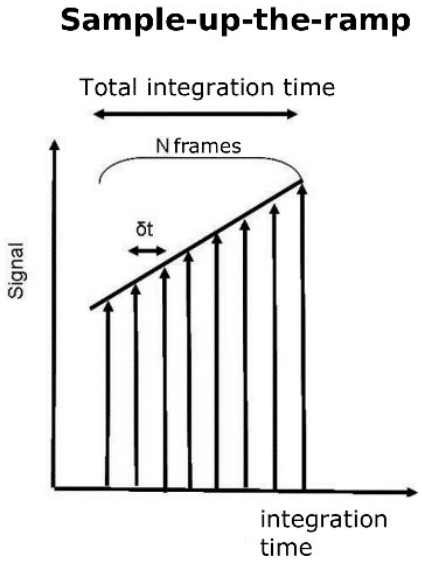
objectif spécifique dans HINM :

simuler le Teledyne HAWAII-4RG (H4RG)



H4RG picture from teledyne-si.com

- 4096×4096 pixels
- canaux de lecture: 1, 4, 16, 32, 64
- pixels de référence en bordure
- multi-lecture non-destructive



nous utilisons le cadriciel Pyxel
pour sa simulation

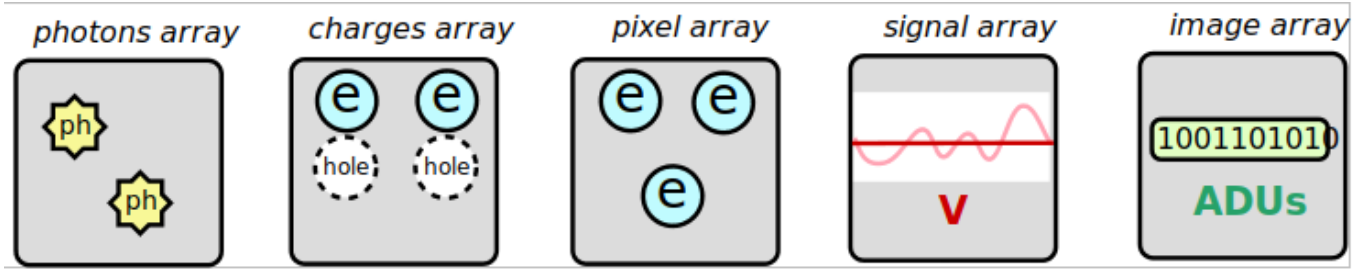
objectif + long terme de chercher les caractéristiques détecteur à partir des fichiers

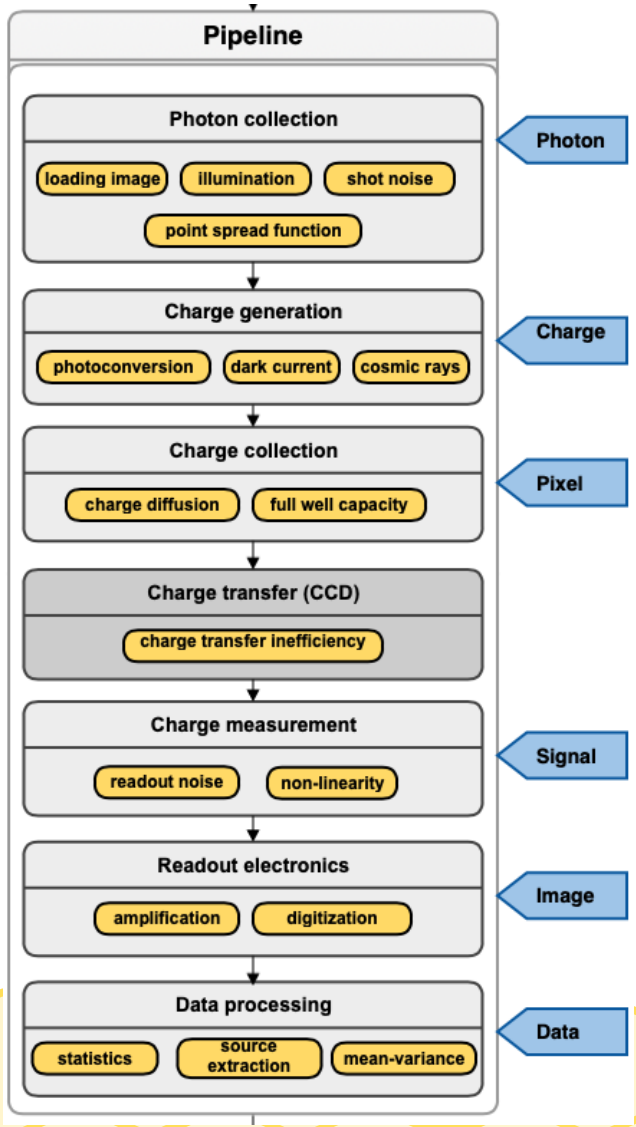
présentation d’aujourd’hui

- introduction rapide au cadriciel Pyxel,
  à travers notre fichier de configuration pour le H4RG

Pyxel

- chaque modèle simule 1 effet.
- “pipeline” séquentielle de modèles
- regroupés par catégorie
- tableaux de stockage (photons, volts...) :





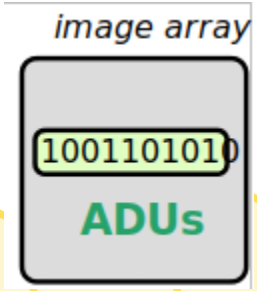
fichier de configuration YAML (partie 1)

```yaml
exposure:
  readout:
    times: [1,2,3] # time points, not durations
    non_destructive:  true
cmos_detector:
  geometry:
    row: 4096 # [px]
    col: 4096 # [px]
    reference_pixels:
        row: [0,1,2,3, 4092,4093,4094,4095]
        col: [0,1,2,3, 4092,4093,4094,4095]
    pixel_vert_size: 15 # [µm]
    pixel_horz_size: 15 # [µm]
    total_thickness: 40 # [µm]
    channels:
      matrix: [[ch1, ch2, ch3, ..., ch64]] # 64 vertical channels
      readout_position: {ch1: top-left, ch2: top-right, ch3: top-left, ..., ch64: top-right}
```

fichier de configuration YAML (partie 2)

```yaml
characteristics:
  full_well_capacity: 100000 # [e]
  charge_to_volt_conversion: 5.7e-6 # [V/e]
  pre_amplification: 20 # [V/V]
  adc_voltage_range: [0,6] # [V,V]
  adc_bit_resolution: 16 # [bit]
environment:
  temperature: 40 # [K]
```

nombre de bits pour chaque pixel

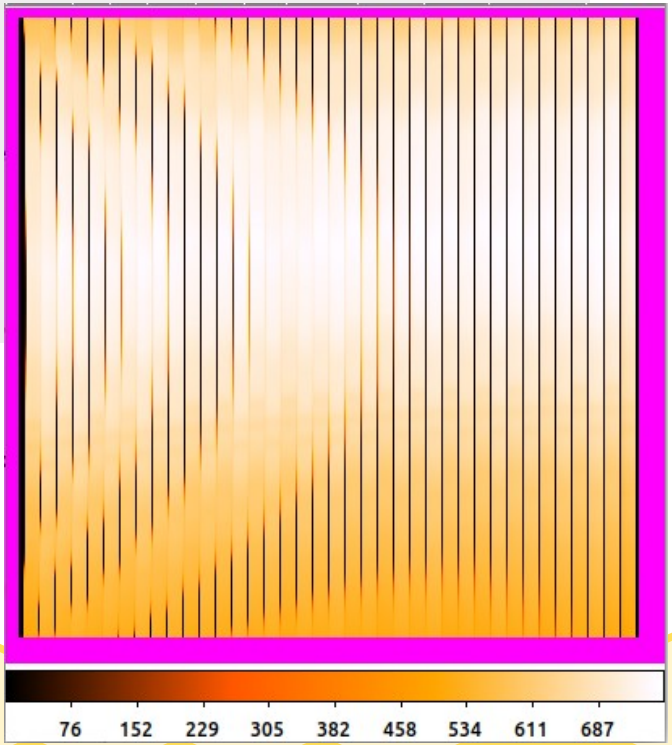


fichier de configuration YAML (partie 3)

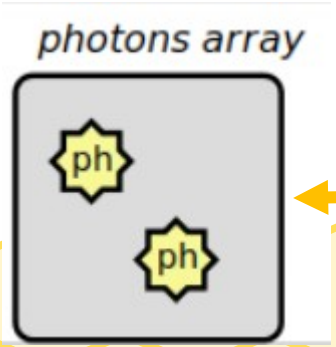
```yaml
pipeline:
  photon_collection:
    - name: load_image # load expected photons per pixel from a FITS matrix
      func: pyxel.models.photon_collection.load_image
      arguments:
        image_file: some_PRM.fits # matrix[ph/s]
        data_path: PATH_A.DATA    # FITS HDU name
    - name: shot_noise # apply poisson law
      func: pyxel.models.photon_collection.shot_noise
```

fichier de configuration YAML (partie 3)

```yaml
pipeline:
  photon_collection:
    - name: load_image # load expected photons per pixel from a FITS matrix
      func: pyxel.models.photon_collection.load_image
      arguments:
        image_file: some_PRM.fits # matrix[ph/s]
        data_path: PATH_A.DATA    # FITS HDU name
    - name: shot_noise # apply poisson law
      func: pyxel.models.photon_collection.shot_noise
```

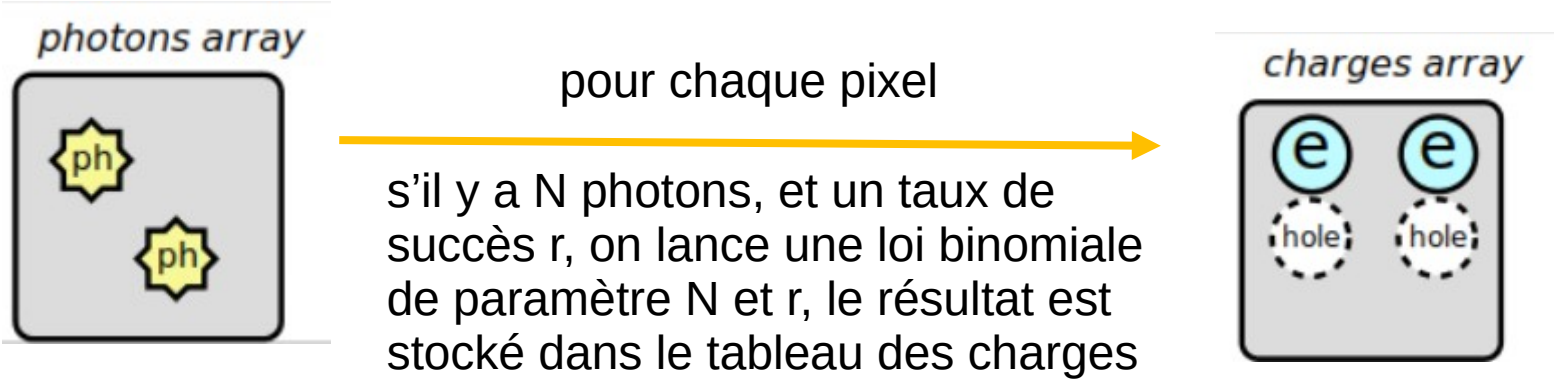


multiplié par temps d'exposition
ajouté au tableau photons



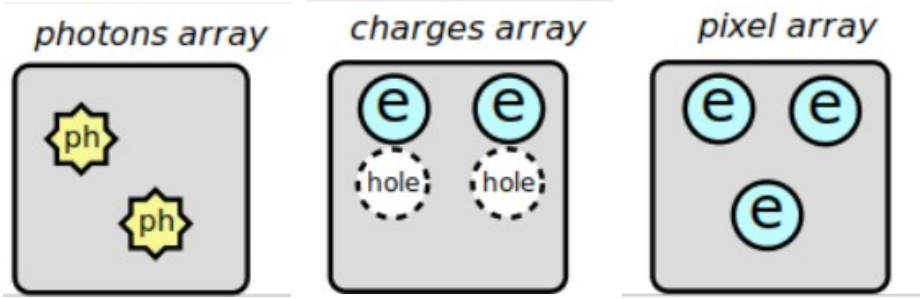
fichier de configuration YAML (partie 4)

```yaml
charge_generation:
   - name: conversion_with_qe_map # convert some photons to charges (apply binomial law)
   - name: simple_dark_current
```



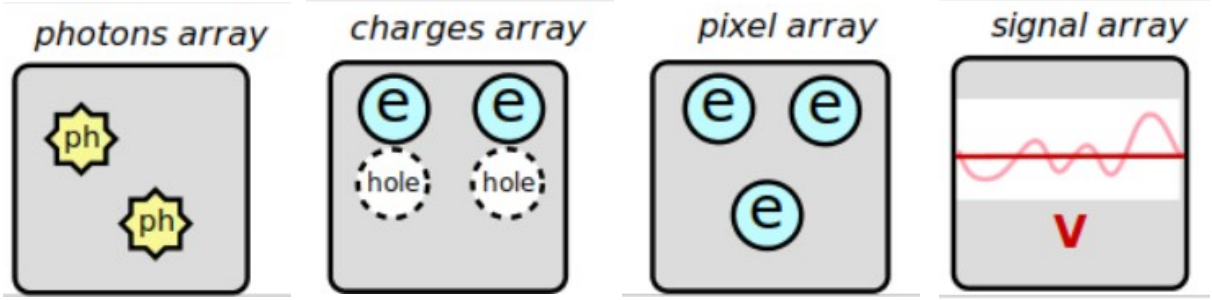
fichier de configuration YAML (partie 5)

```yaml
charge_collection:
  - name: simple_collection # copy charges to electrons
  - name: full_well # cap pixel values
  - name: simple_persistence # (disabled) add parts of previous exposures
```



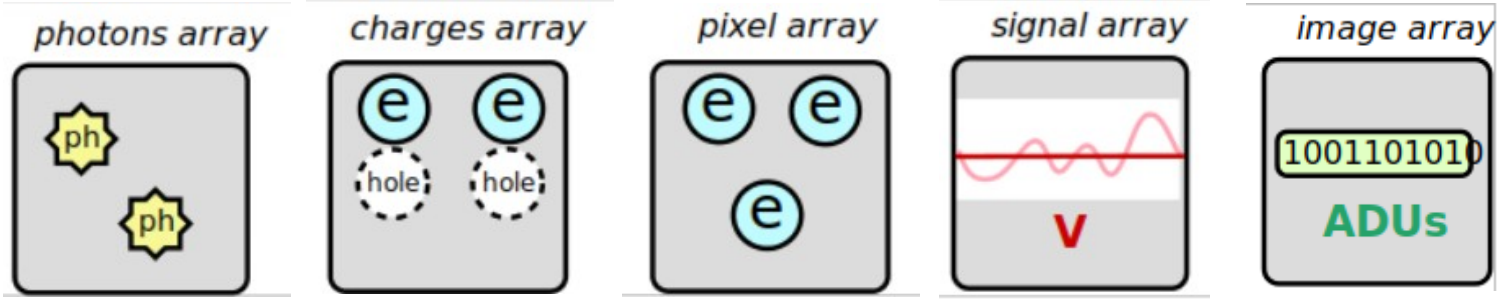
fichier de configuration YAML (partie 6)

```yaml
charge_measurement:
  - name: simple_ipc # convert electrons to volts, with some blur
  - name: dc_offset # add constant reset offset to each channel
  - name: ktc_noise # add reset noise to each channel. same noise for each frame
  - name: correlated_pink_noise_cmos # add correlated channel pink noise
  - name: uncorrelated_pink_noise_cmos # add uncorrelated channel pink noise
  - name: alternating_column_noise_cmos # add odd/even columns pink noise
  - name: output_node_noise_cmos # add white noise
  - name: output_node_linearity_poly # apply polynom to simulate non-linearity
```

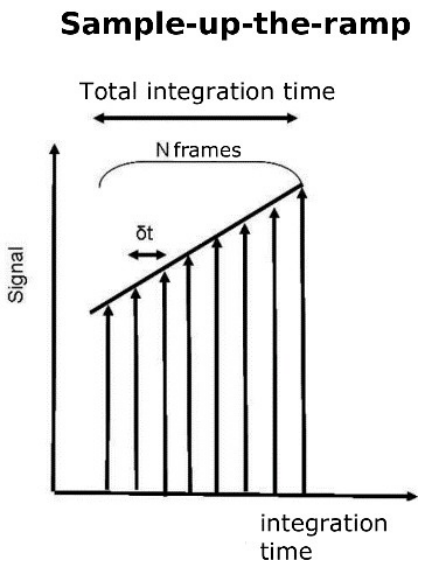


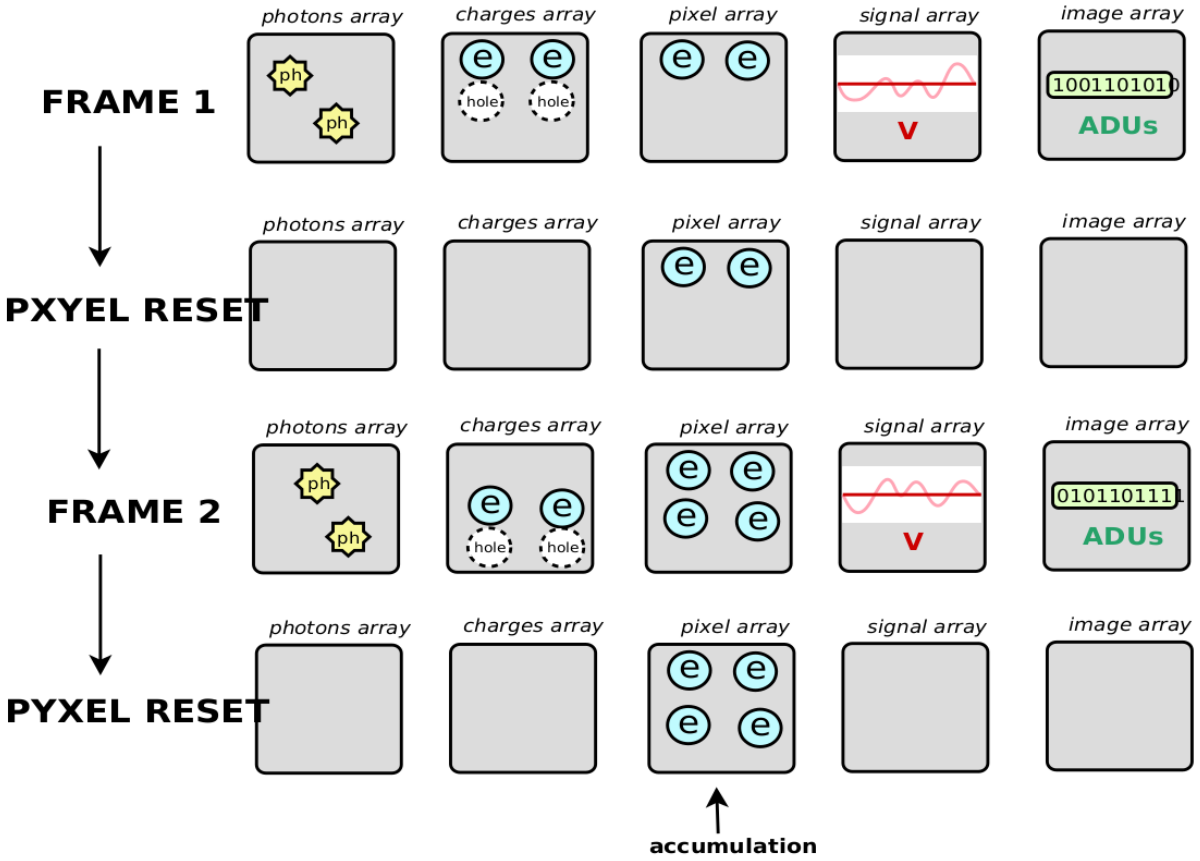
fichier de configuration YAML (partie 7)

```yaml
readout_electronics:
  - name: simple_amplifier # factor volts
  - name: simple_adc # convert volts to ADUs and cap
```

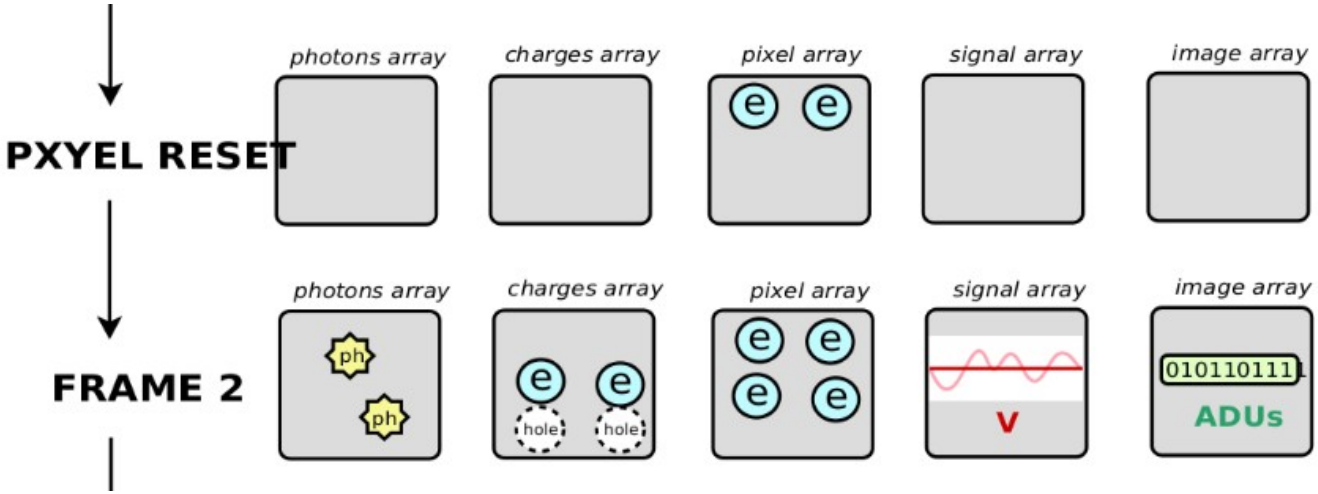


Gestion de la lecture non-destructive





Gestion de la lecture non-destructive



- point fort de Pyxel : garder une séquentialité simple,
  et une séparation forte entre chaque modèle -> code simple et structuré
- souplesse des choix des modèles utilisés

Cadriciel collaboratif

- le travail collaboratif se fait par un GitLab standard
- avec Merge Request, Feature Request avec modèles
- séries de tests automatiques (incluant vérification de type, etc)
- réunions de développements régulières dans l'année, ouvertes à tous, y compris des particuliers
- workshops chaque année
- équipe de développement très favorable à la collaboration
- page de tutoriel pour la contribution au cadriciel
- base de partage de code