

DSA-2000

Real-time aperture synthesis
imaging with the DSA-2000 radio camera

2024 Legion Retreat

Martin Pokorny, Caltech

www.deepsynoptic.org

Caltech



Schmidt Sciences

What is DSA-2000?

DSA-2000

A world-leading radio survey telescope and multi-messenger discovery engine.

- ~2000 x 5m dishes (19 x 15 km)
- Spring Valley, Nevada
- Frequency: 0.7 - 2 GHz band
- Spatial resolution: 3.3 arcseconds
- *Highly optimized for surveys*

Currently in final design phase

Construction start: 2026

First light: 2028

Key surveys: 2028 – 2033

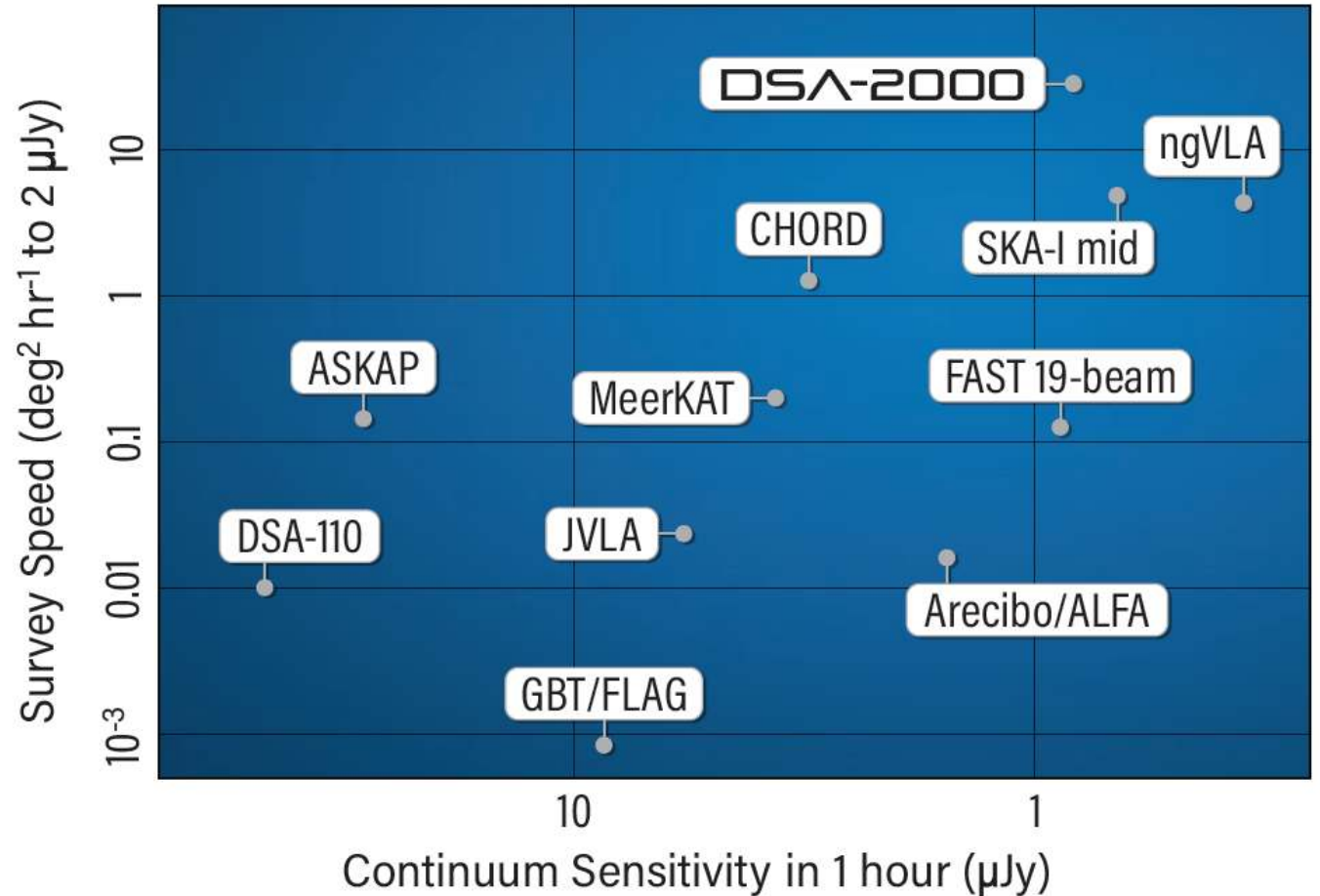


Unparalleled survey speed

- 31,000 deg² to 500 nJy
- >1 billion radio sources (IQUV)
- ~few million galaxies in HI
- ~10⁵ FRBs and pulsars
- ~10⁶ "slow" transients

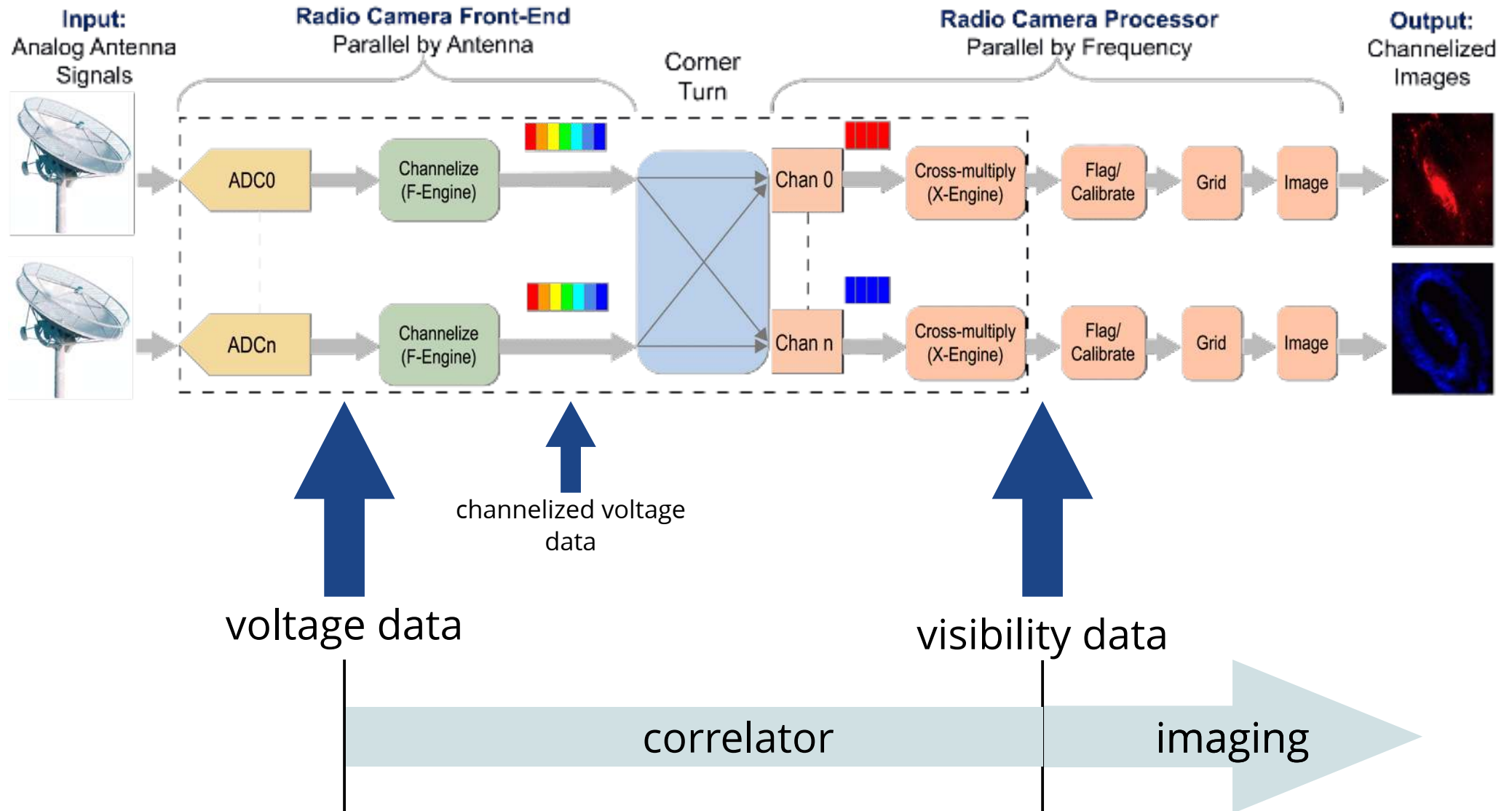
Enabled by two key technologies:

- "Radio camera" digital back-end
- Cryo-free antenna/receiver



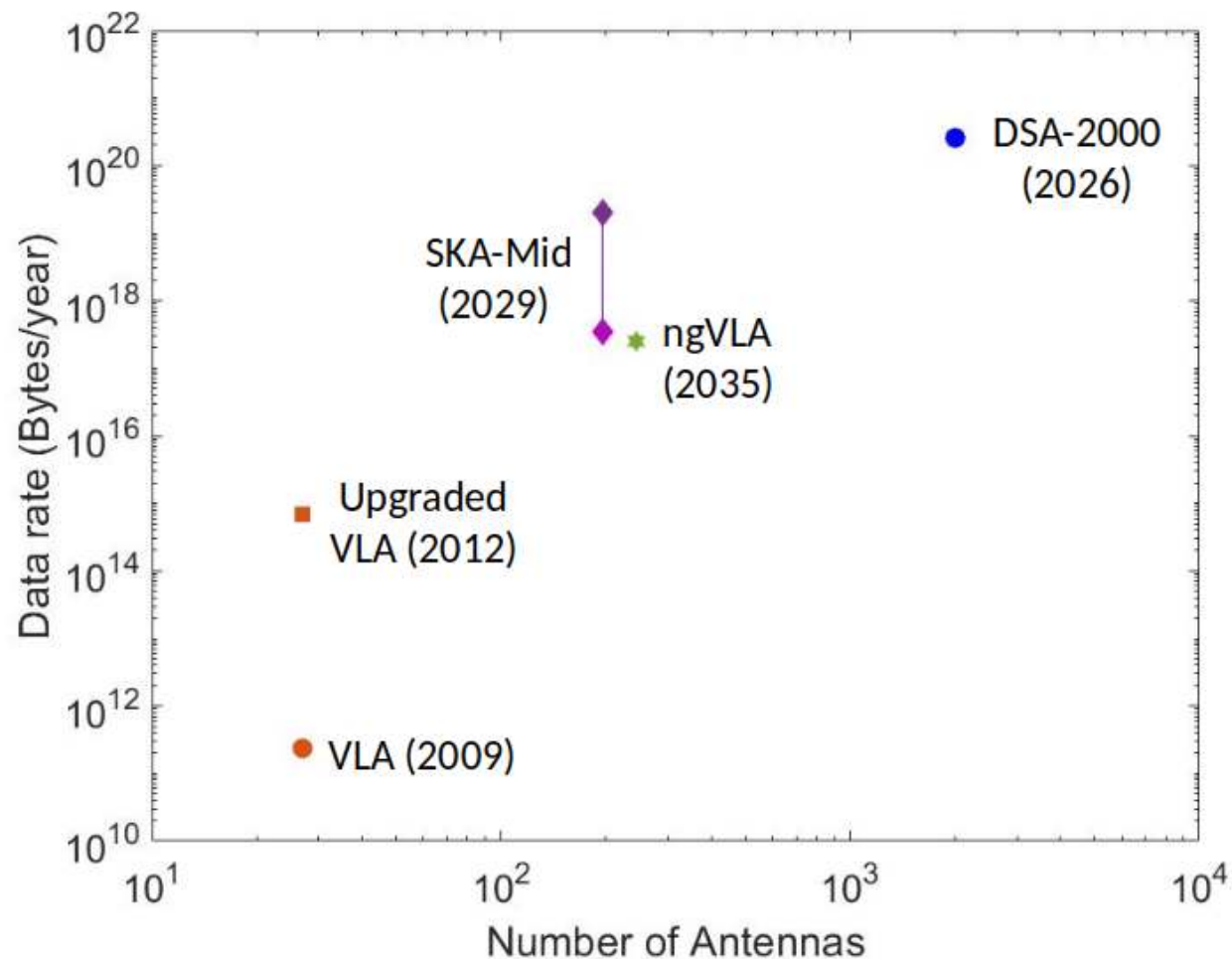
What is a radio camera?

Radio camera schematic



Radio camera motivation

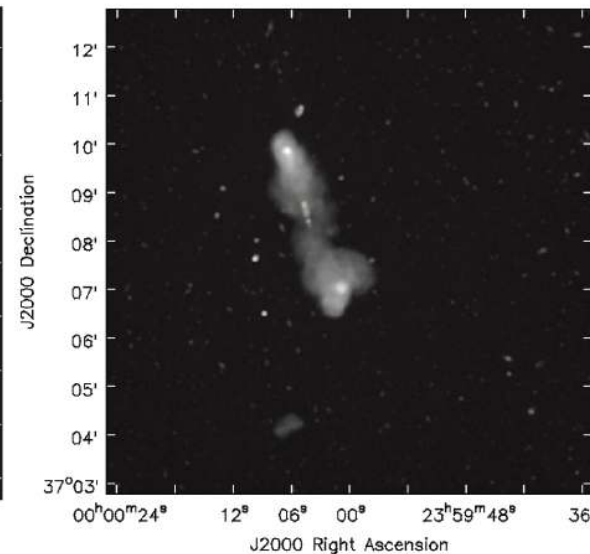
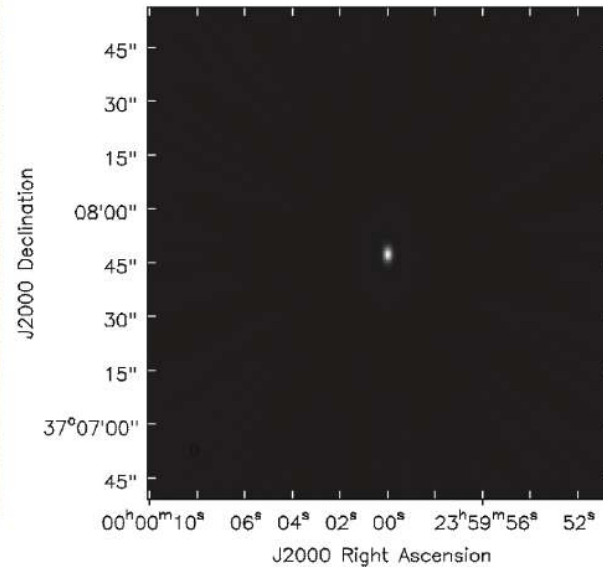
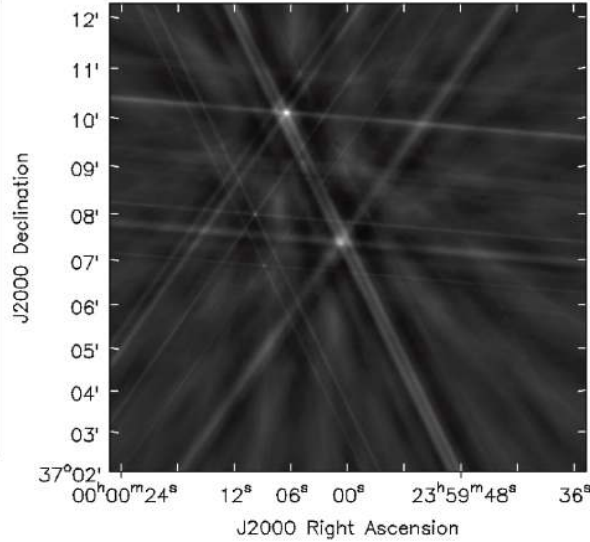
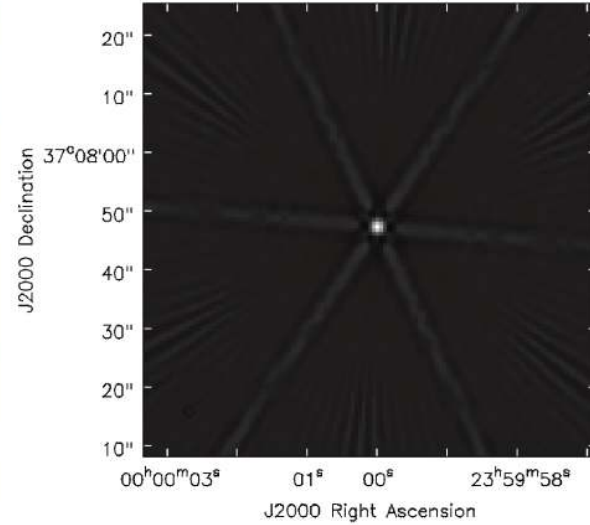
Radio telescope correlator output



DSA-2000

- ~0.8 Zettabytes/yr of voltage data
- 20 Exabytes/yr of visibility data
- \$100 million/yr in storage costs to save visibility data

Radio camera imaging



- No need for visibility-based deconvolution
- Enables a deterministic stream processing pipeline that creates images

Radio camera processing characteristics

- High degree of data parallelism, by frequency channel slices (sub-bands)
 - no cross-sub-band communication, except, maybe...
- Calibration improves with greater bandwidth; may need to span sub-bands.
- RCP input (channelized voltages): 43Tb/s
- RCP output (images): 23 GB/s

| Data | Type | Rate (per input channel) |
|-------------------------|-----------------------------|---------------------------------|
| voltages | 4+4 bit complex integer | 4 Gb/s |
| raw visibilities | 32+32 bit complex integer | 43 MB/s |
| normalized visibilities | complex single precision fp | 43 MB/s |
| gridded visibilities | complex single precision fp | 107 MB/s |
| grid pixel (updates) | complex double precision fp | ~43 GB/s (20x20 CF kernel) |

Radio camera processor deployment

- Goal: 4 input channels per GPU
- All 20 channels in each sub-band are processed by a single node
- 10 GPUs per node: 2 sub-bands per node
 - 3800 GPUs
- baseline GPU: NVIDIA RTX 4000 Ada generation
- 200 GbE network interface (for data input)
- Image output to node-local file system for post-processing

Imaging pipeline software

symcam

(SYnthesis iMaging CAMera)

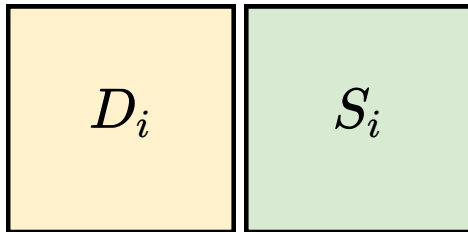
Imaging pipeline inputs

- Data (channelized voltages)
 - for every channel, data arrive from all antennas
 - one input data "stream" for every N (4) contiguous channels
 - each antenna multicasts voltage data *via* UDP on all streams
 - raw data: 7.5 μ sec sample time
 - packetized data: several sample times (maximum possible for MTU)
 - metadata: timestamp, stream identification (channels), flag values (bad data)
- Events
 - control messages (*e.g.*, start/stop imaging, run bandpass calibration, start/stop diagnostic output)
 - system status metadata (*e.g.*, antenna beam patterns, calibration terms)
 - metadata: activation/validity timestamp

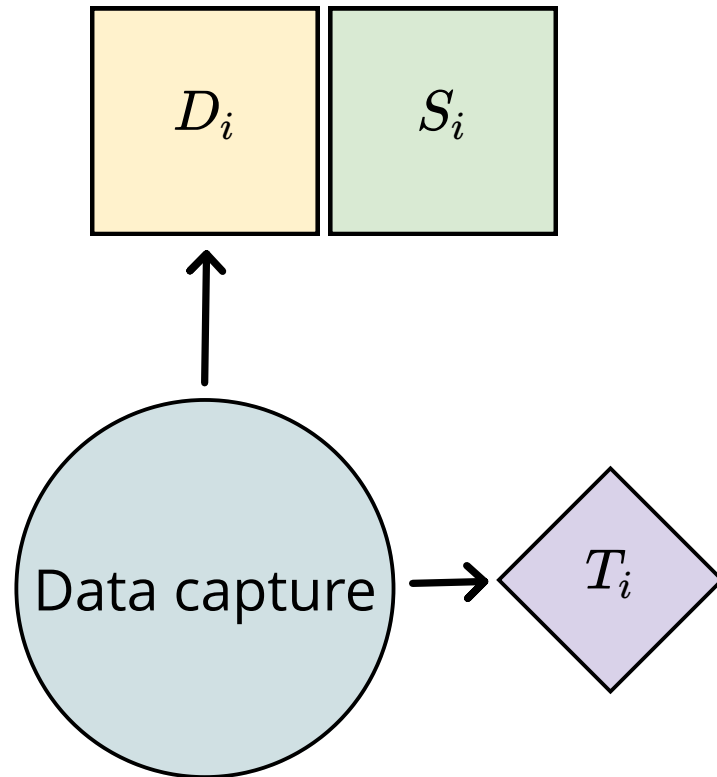
From inputs to regions

- Both data and events are received and buffered by dedicated threads
- Legion tasks launched from the main loop fill regions associated with timestamps with data or events from service thread buffers
- Real-time conditions
 - data timeout: derived from timestamps of received packets
 - event deadline: derived from completion of tasks that fill data buffers
 - no explicit system clock dependency except when no data are received

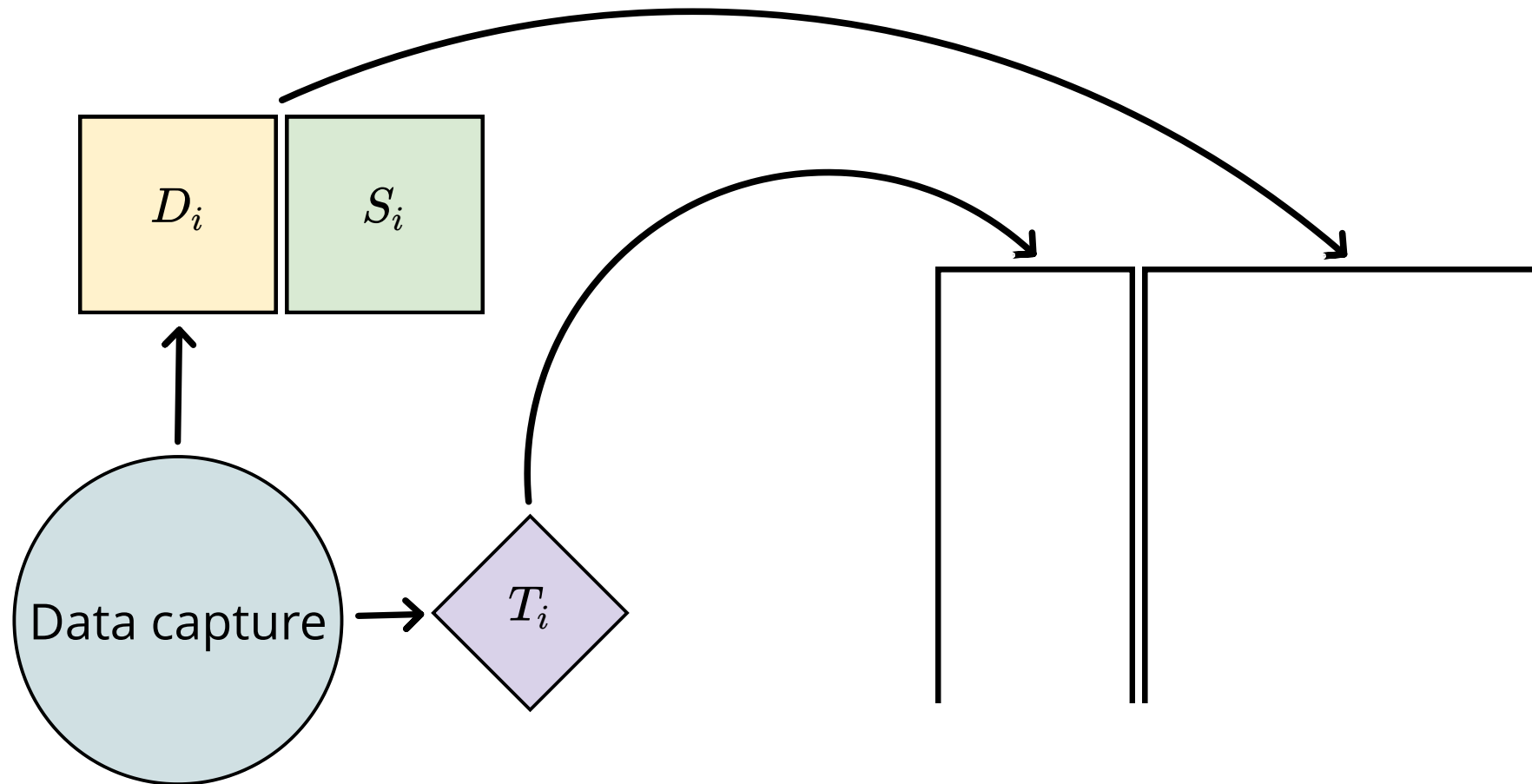
Main loop: input management



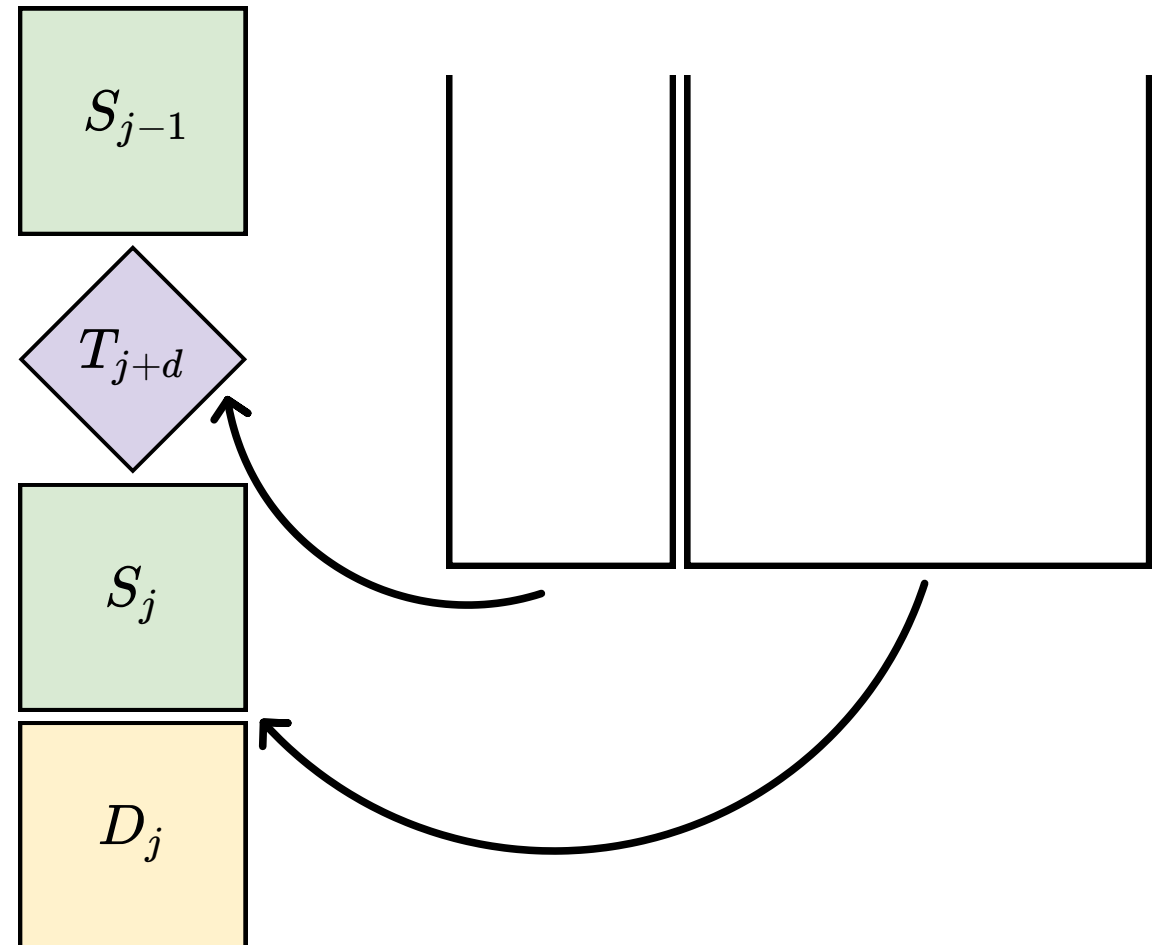
Main loop: input management



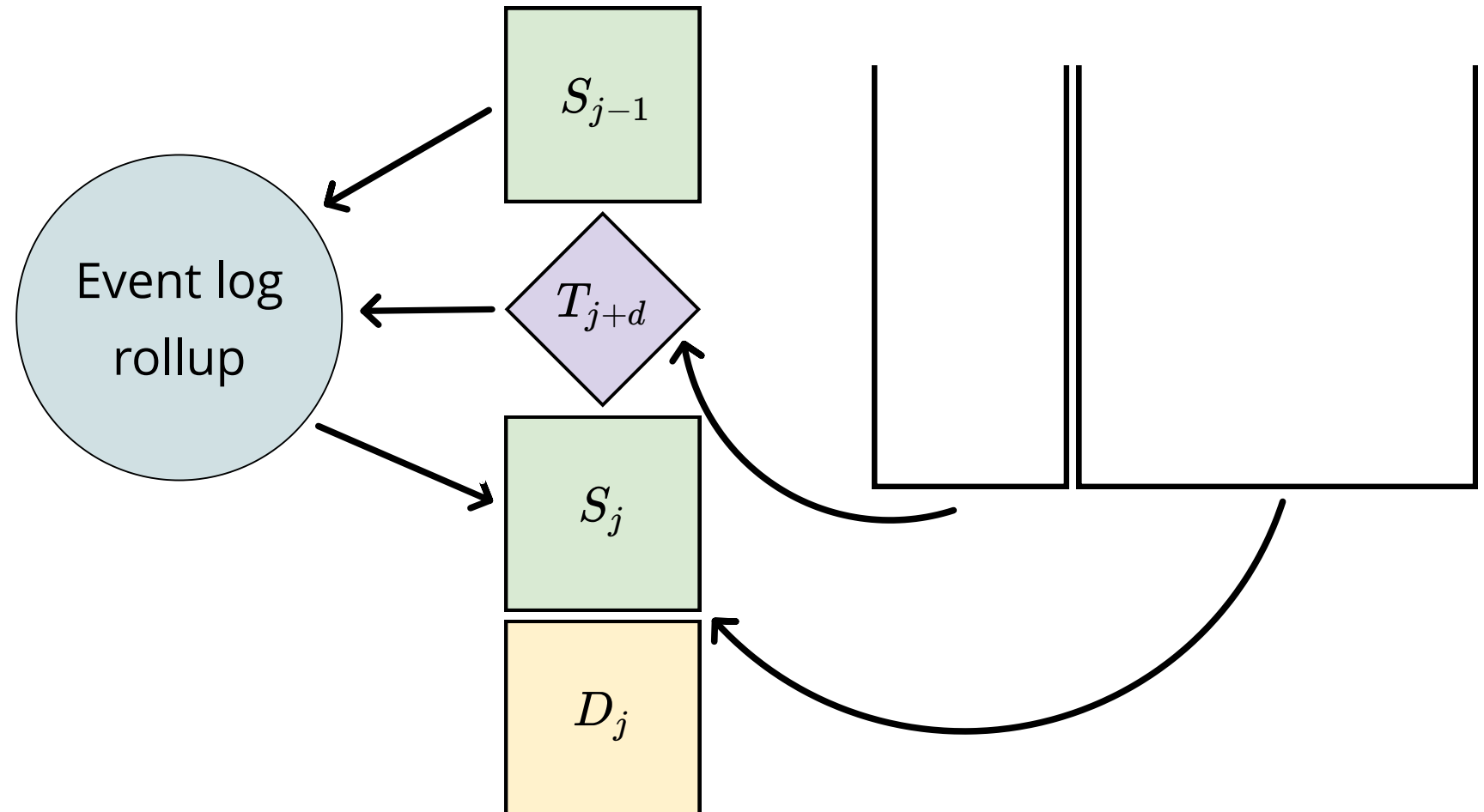
Main loop: input management



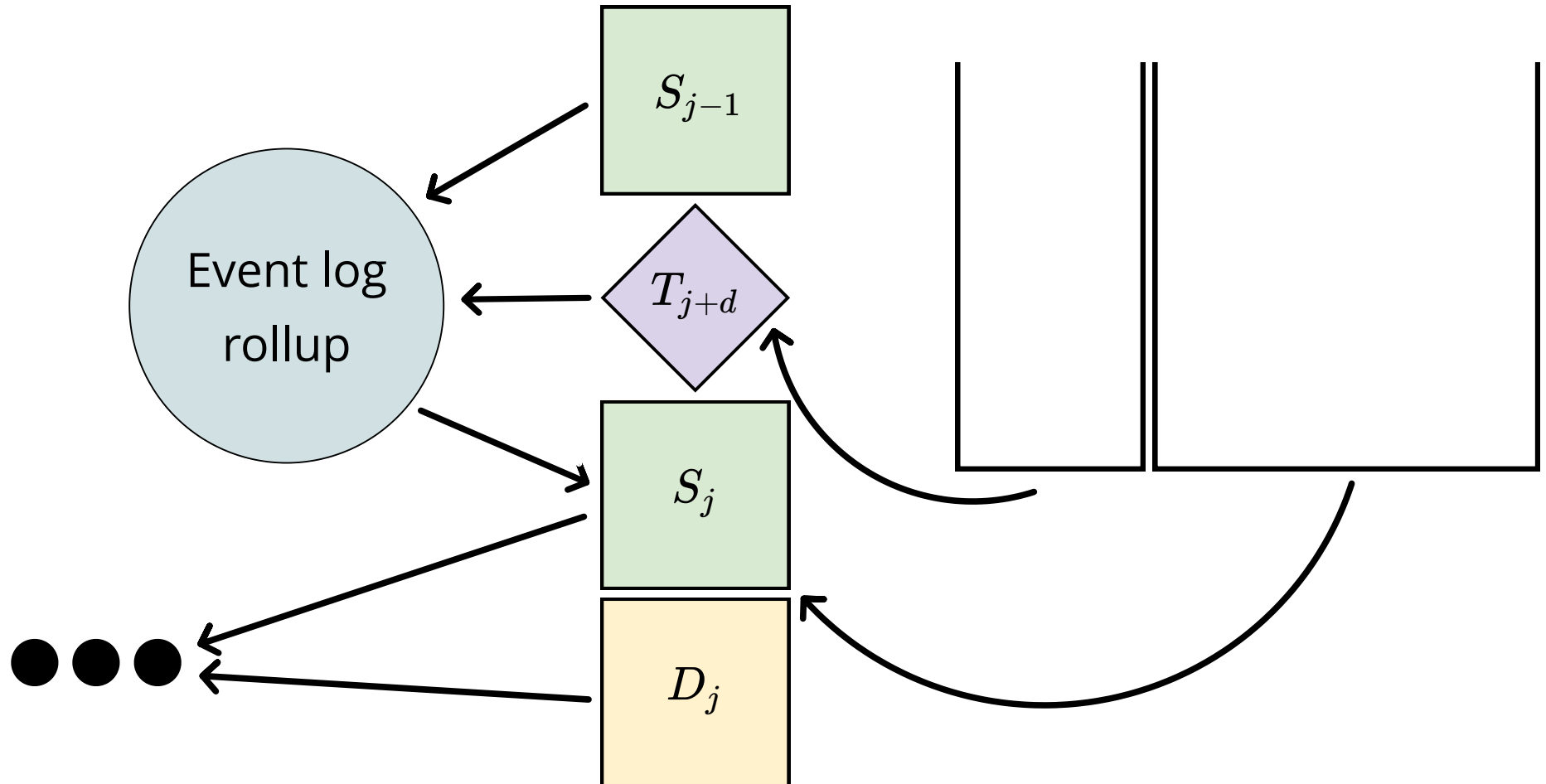
Main loop: input management



Main loop: input management

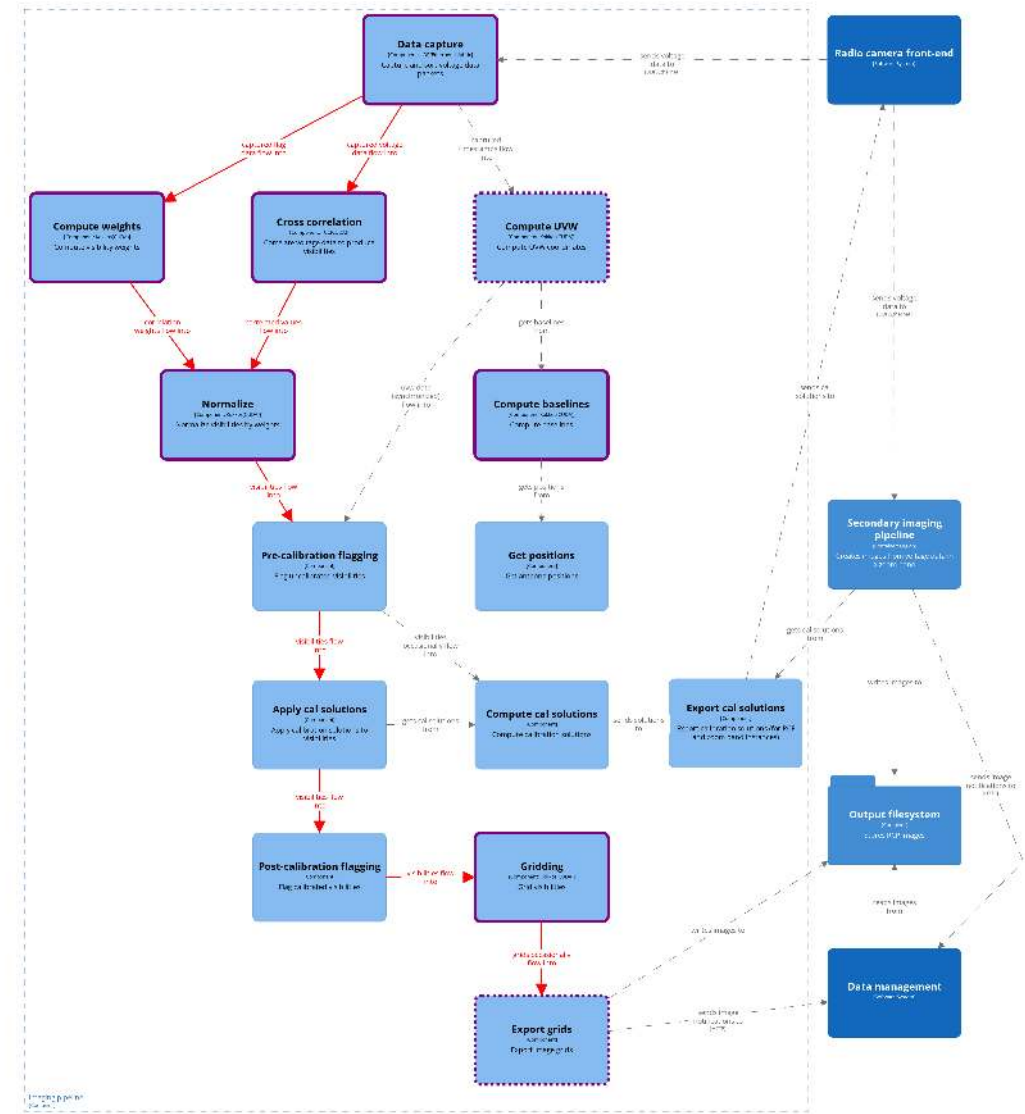


Main loop: input management



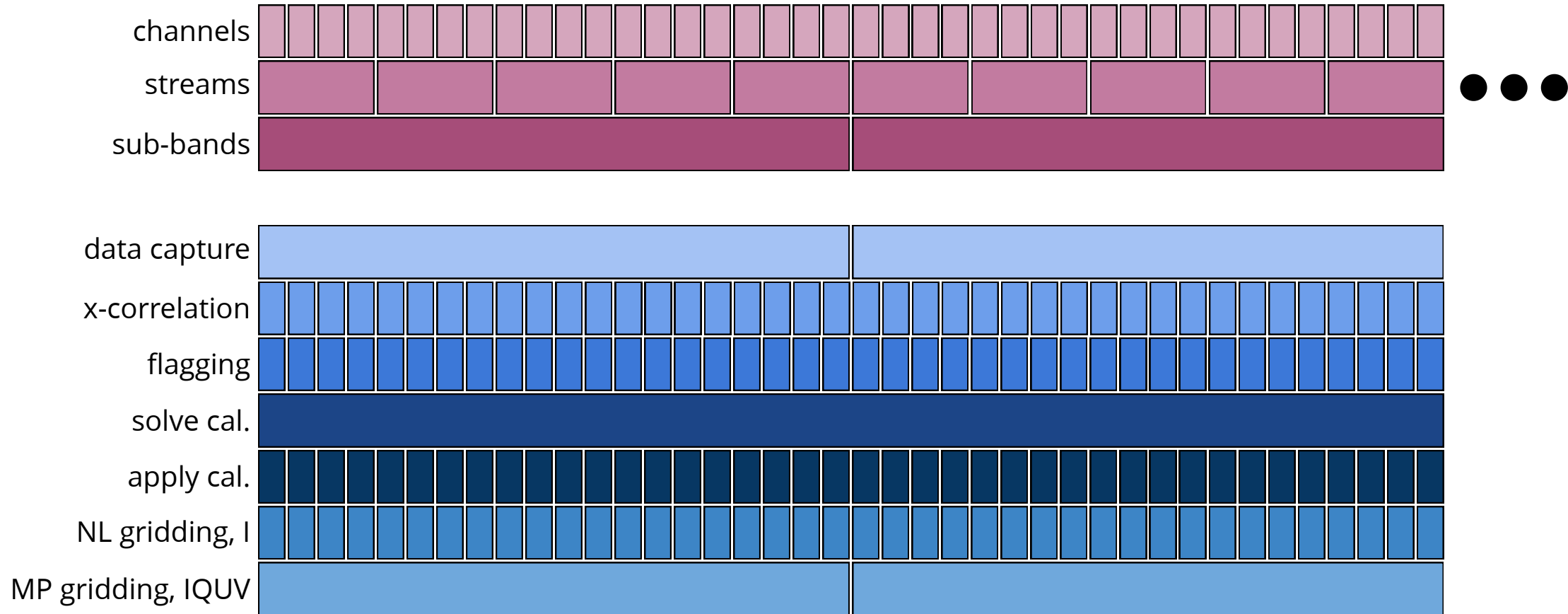
Pipeline states and predication

- a pipeline state is associated with every data timestamp
- states are determined by events
- states determine processing for data with associated timestamps
- pipeline processing is a series of stages, implemented as tasks
- states provide predicates for tasks, effectively making each stage conditional on the pipeline state



Component: radio camera processor - imaging pipeline

Channel data regions & partitions



Current status and next steps

Status

- Events
 - create/destroy subarray
 - shutdown
 - start/stop imaging
 - start/stop flagging
 - start/stop solve calibration
 - start/stop apply calibration (partial)
 - start/stop record data
 - start/stop record events
 - import/export fftw wisdom

Status

- Events
 - create/destroy subarray
 - shutdown
 - start/stop imaging
 - start/stop flagging
 - start/stop solve calibration
 - start/stop apply calibration (partial)
 - start/stop record data
 - start/stop record events
 - import/export fftw wisdom
- Stages
 - data capture
 - cross-correlation
 - data weight computation (flag correlation)
 - normalization
 - gridding
 - Fourier transform
 - image export (partial)

Status

- Events
 - create/destroy subarray
 - shutdown
 - start/stop imaging
 - start/stop flagging
 - start/stop solve calibration
 - start/stop apply calibration (partial)
 - start/stop record data
 - start/stop record events
 - import/export fftw wisdom
- Stages
 - data capture
 - cross-correlation
 - data weight computation (flag correlation)
 - normalization
 - gridding
 - Fourier transform
 - image export (partial)
- Incomplete
 - data flagging (RFI)
 - solve calibration
 - modal (delay, bandpass, polarization)
 - continuous (gain)
 - apply calibration
 - bright source subtraction
 - export calibration solutions
 - import calibration solutions

Integration with calibration

Current situation

- calibration algorithms developed by domain expert
- implemented using Python
 - numpy
 - JAX
 - Ray
- use inter-process comm with a Ray actor for computing calibration solutions
 - ugly, but may work if averaging is done on Legion side

Integration with calibration

Current situation

- calibration algorithms developed by domain expert
- implemented using Python
 - numpy
 - JAX
 - Ray
- use inter-process comm with a Ray actor for computing calibration solutions
 - ugly, but may work if averaging is done on Legion side

Future design

- desirable, but perhaps unnecessary
- Legate
- tasks implemented using XLA

Main loop refactor

- main challenge of main loop is to run -- not just launch -- data capture tasks ahead of time to avoid losing input data
- restrict main loop to issuing data and event capture tasks, and a single, high-level "input data processing" task, which runs after data and required events have arrived
- all states are known when input data processing task runs -- no predication needed