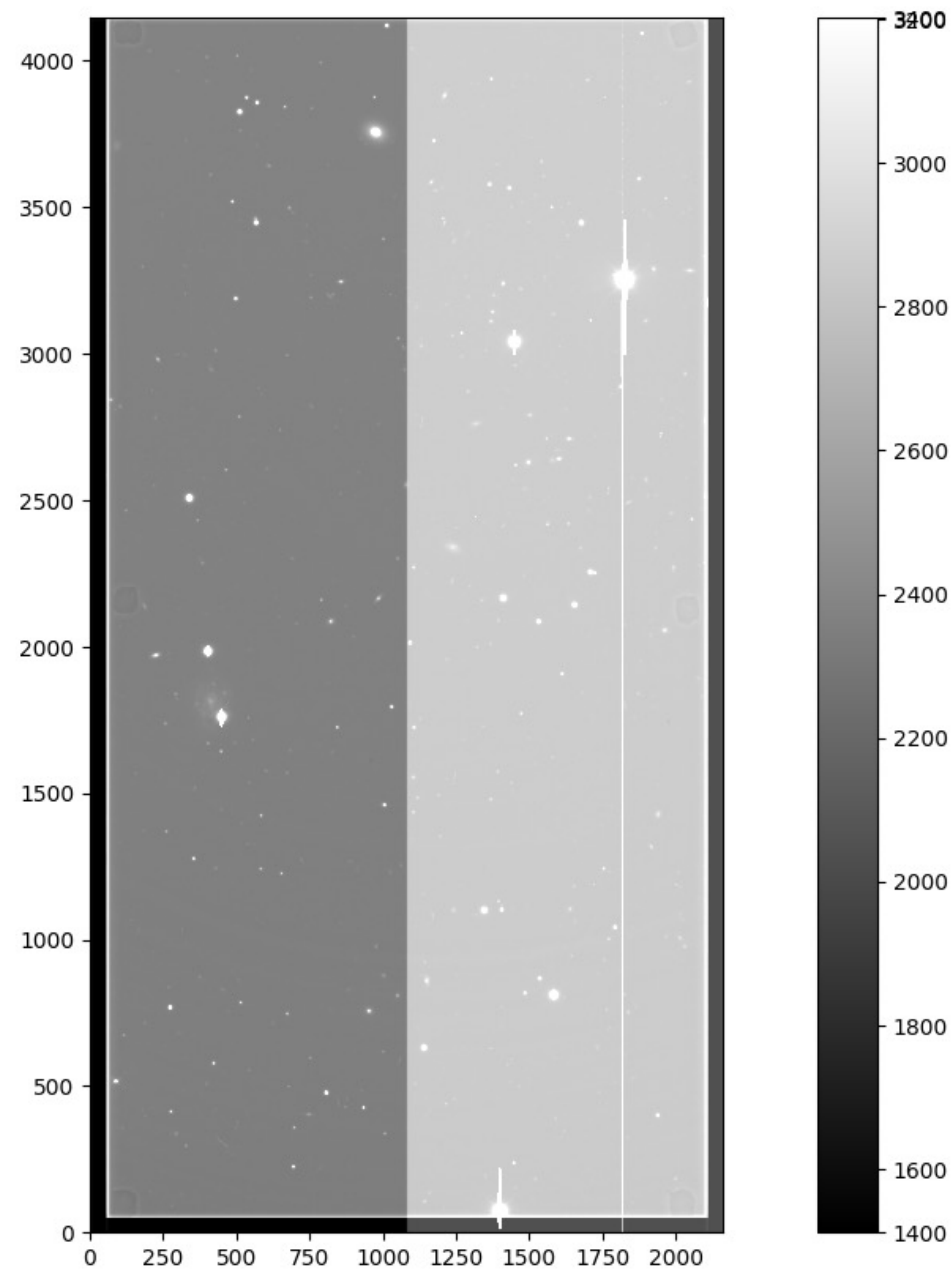
Fermilab | U.S. DEPARTMENT OF ENERGY | Office of Science

Efforts Toward
In-storage Processing of Astronomical Images using FPGAs

Semih Kacmaz, Fermi National Accelerator Laboratory

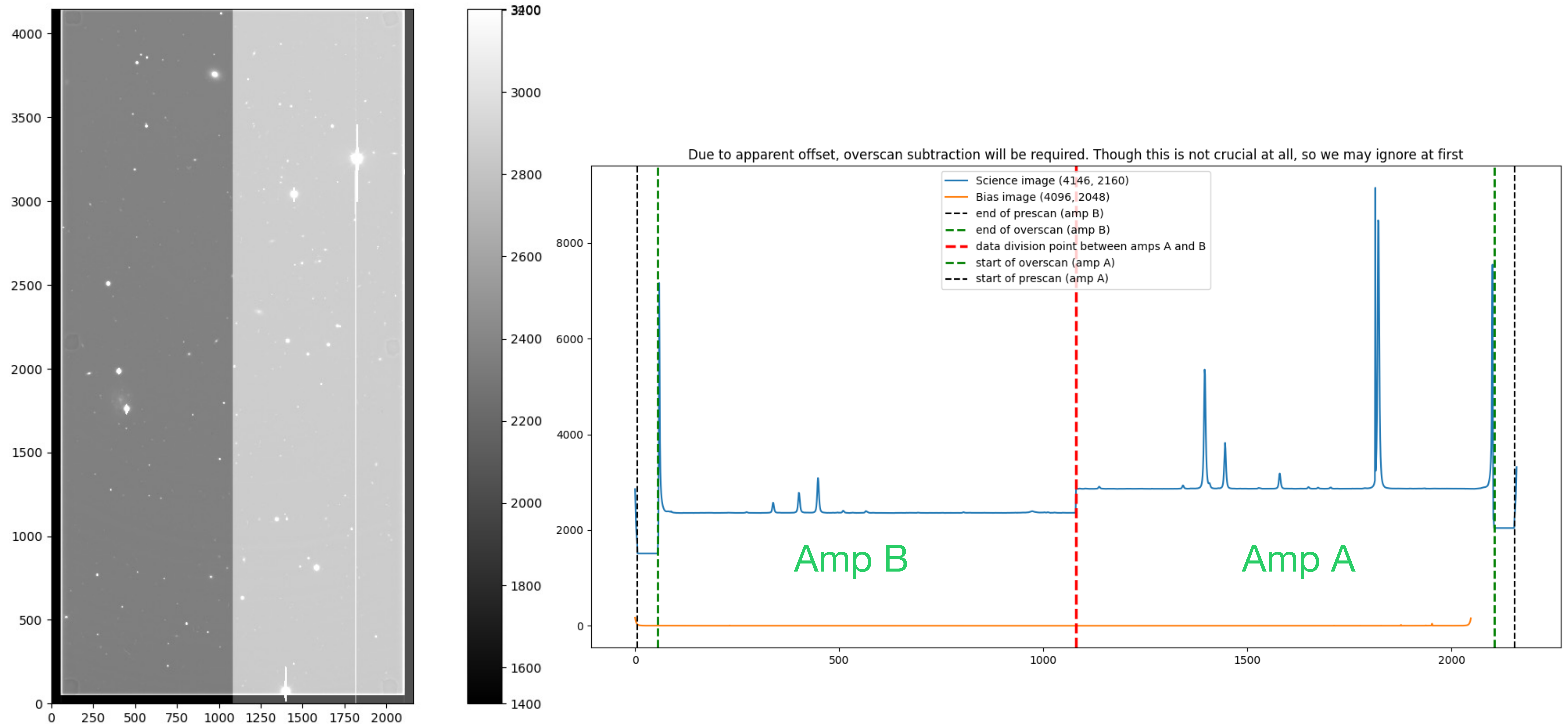
August 11, 2023

Science Image and Objectives



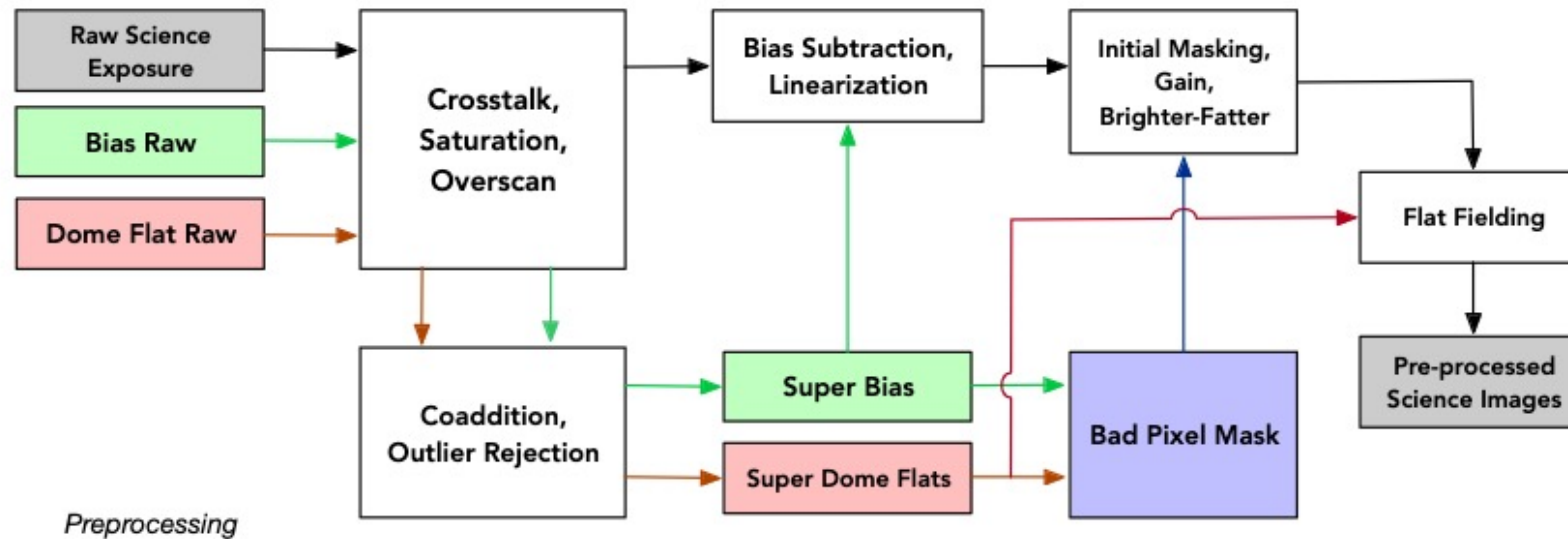
- Raw science image is in 4146 x 2140. We see a sample image obtained as part of the Dark Energy Survey data.
- Two amplifiers and several correction regions
- The actual image data is contained between pixels [50: , 56:2104] — i.e. it has 4K resolution.
- Our objective is to replicate a portion of the Dark Energy Survey's pre-processing pipeline on several levels
- We eventually aim to implement all those steps in FPGAs which can potentially accelerate the pipeline processing timeline by several orders of magnitude.
- The accelerated pipeline performance is expected to be directly useful in supernova detection and GW astronomy.
- Ideal scenario: our efforts become a part of the Rubin observatory prompt pre-processing pipeline

Science Image



Science Image (left) and Its Row-Averaged Distribution (right)

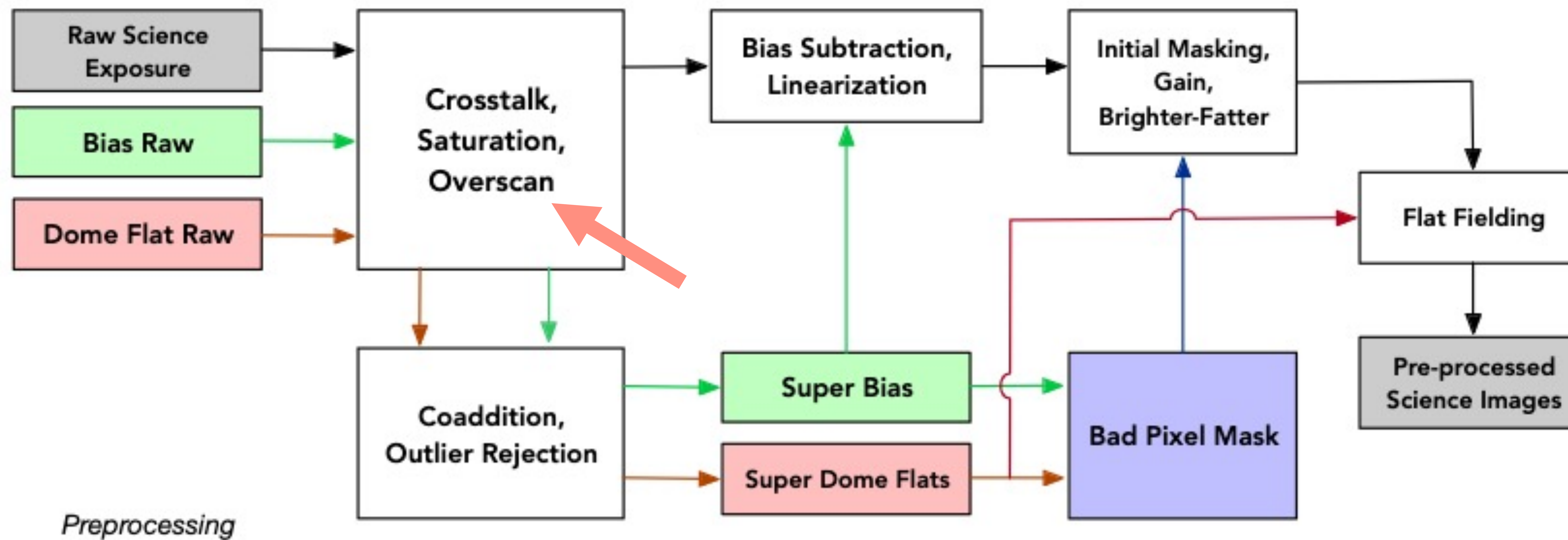
DES Preprocessing Pipeline



Schematic Illustrating the DES Preprocessing Pipeline

Adapted from Morganson et al., The Dark Energy Image Processing Pipeline Preprint Draft Version, 2018, https://arxiv.org/pdf/1801.03177.pdf

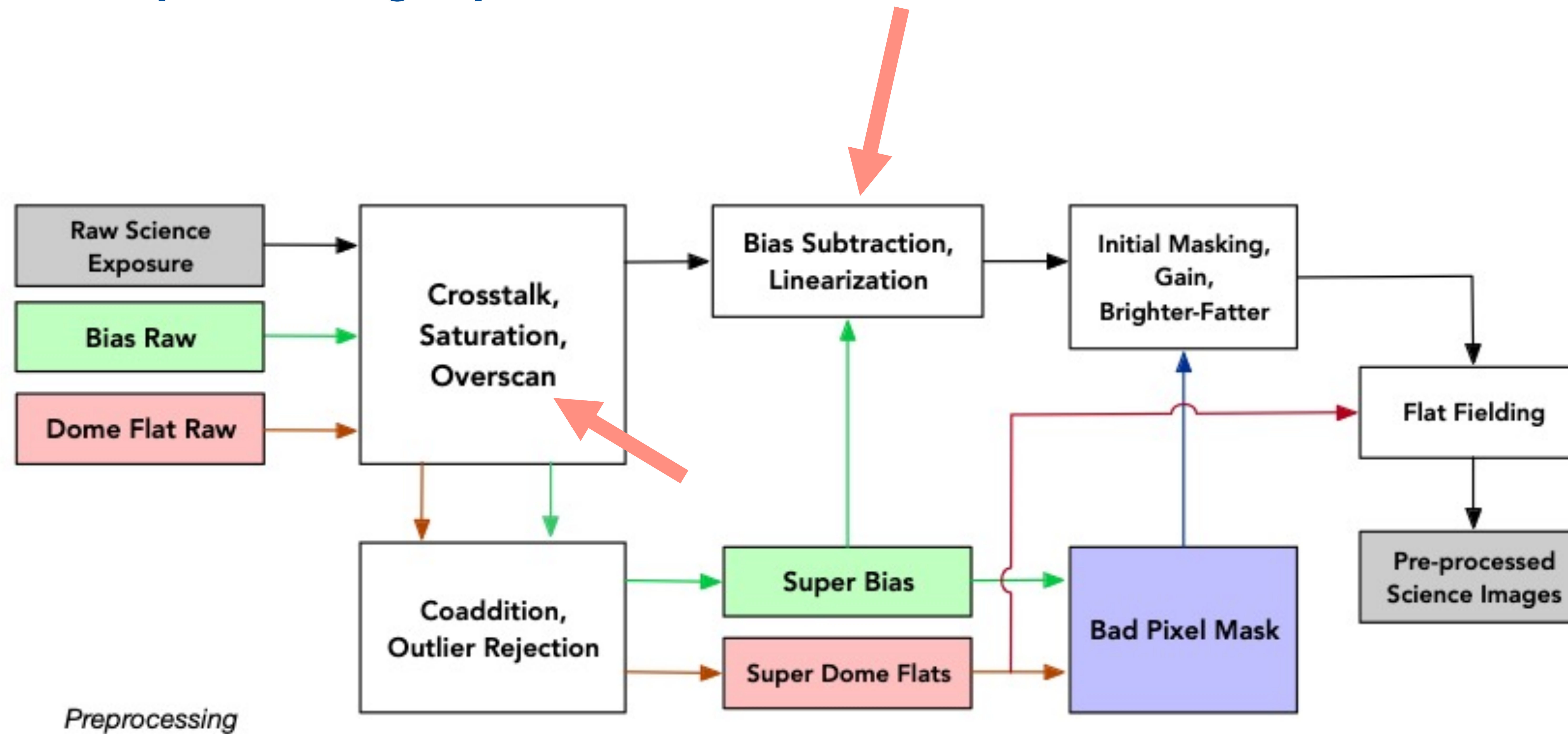
DES Preprocessing Pipeline



Schematic Illustrating the DES Preprocessing Pipeline

Adapted from Morganson et al., The Dark Energy Image Processing Pipeline Preprint Draft Version, 2018, https://arxiv.org/pdf/1801.03177.pdf

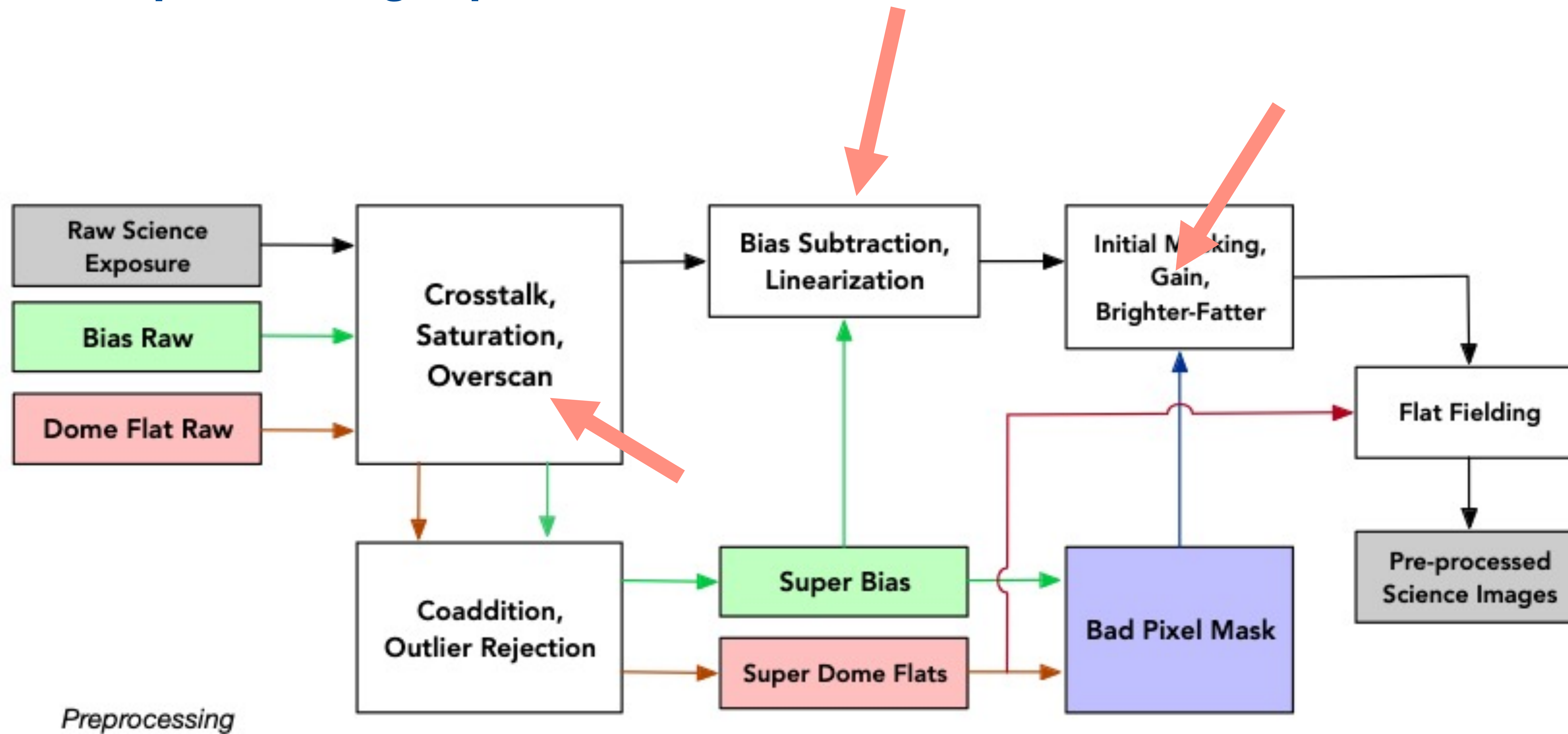
DES Preprocessing Pipeline



Schematic Illustrating the DES Preprocessing Pipeline

Adapted from Morganson et al., The Dark Energy Image Processing Pipeline Preprint Draft Version, 2018, https://arxiv.org/pdf/1801.03177.pdf

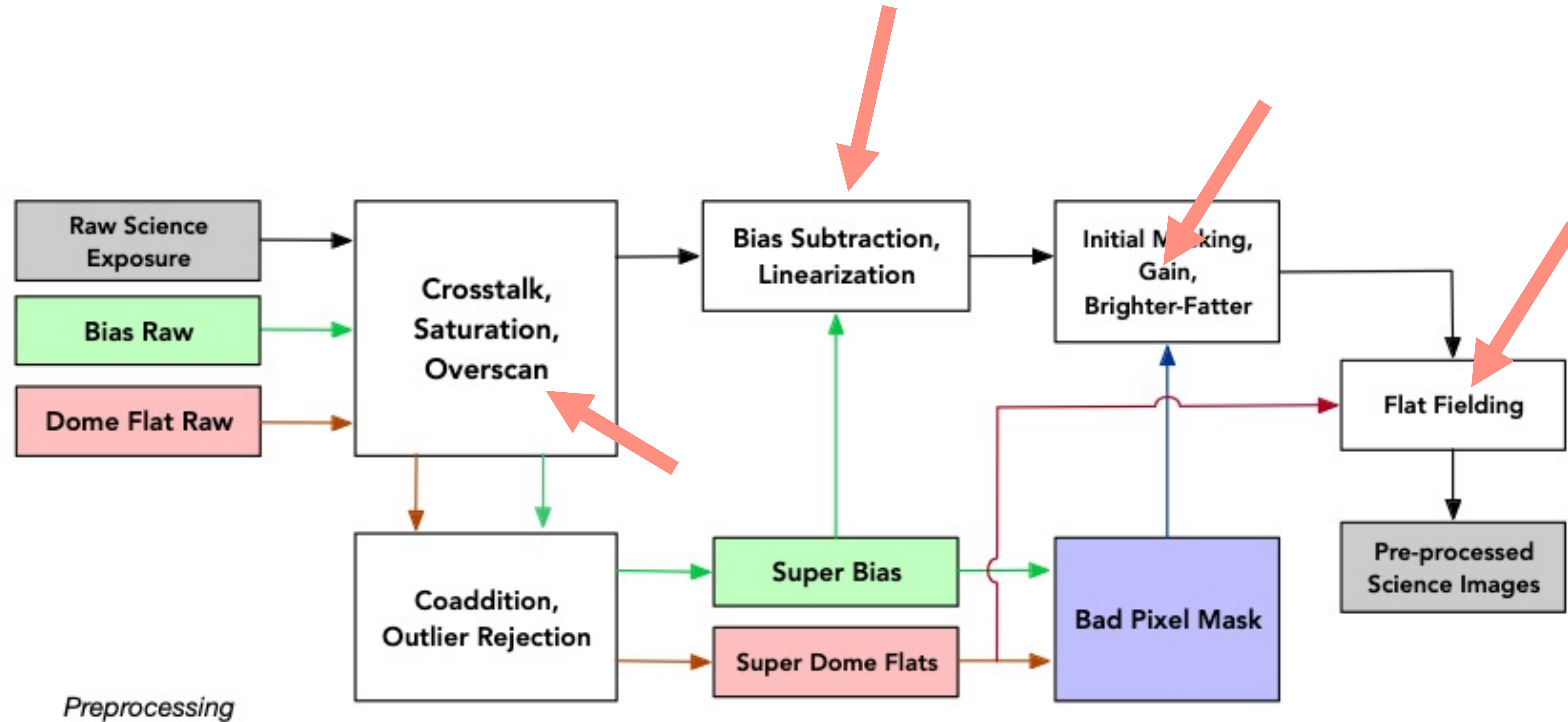
DES Preprocessing Pipeline



Schematic Illustrating the DES Preprocessing Pipeline

Adapted from Morganson et al., The Dark Energy Image Processing Pipeline Preprint Draft Version, 2018, https://arxiv.org/pdf/1801.03177.pdf

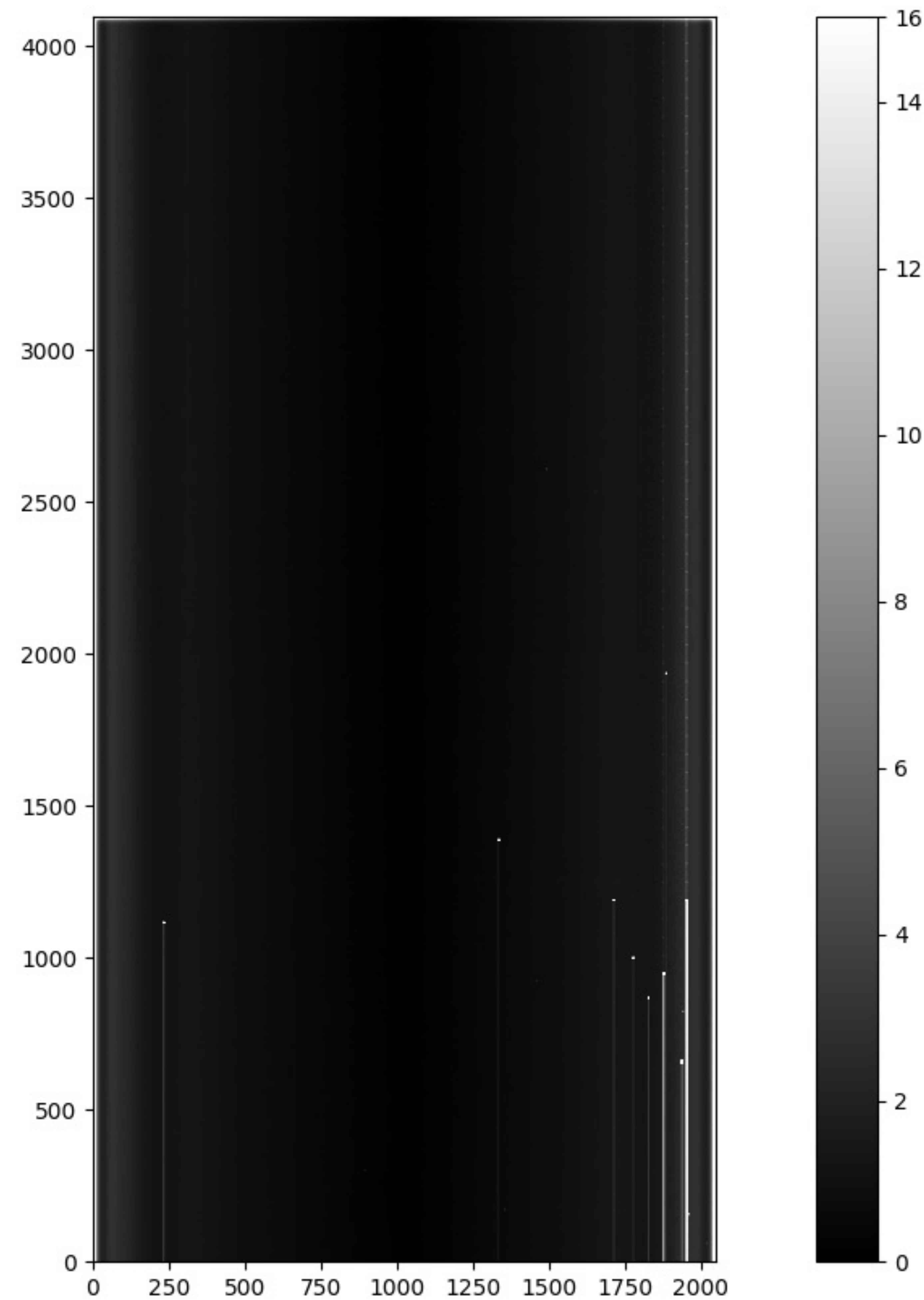
DES Preprocessing Pipeline



Schematic Illustrating the DES Preprocessing Pipeline

Adapted from Morganson et al., The Dark Energy Image Processing Pipeline Preprint Draft Version, 2018, https://arxiv.org/pdf/1801.03177.pdf

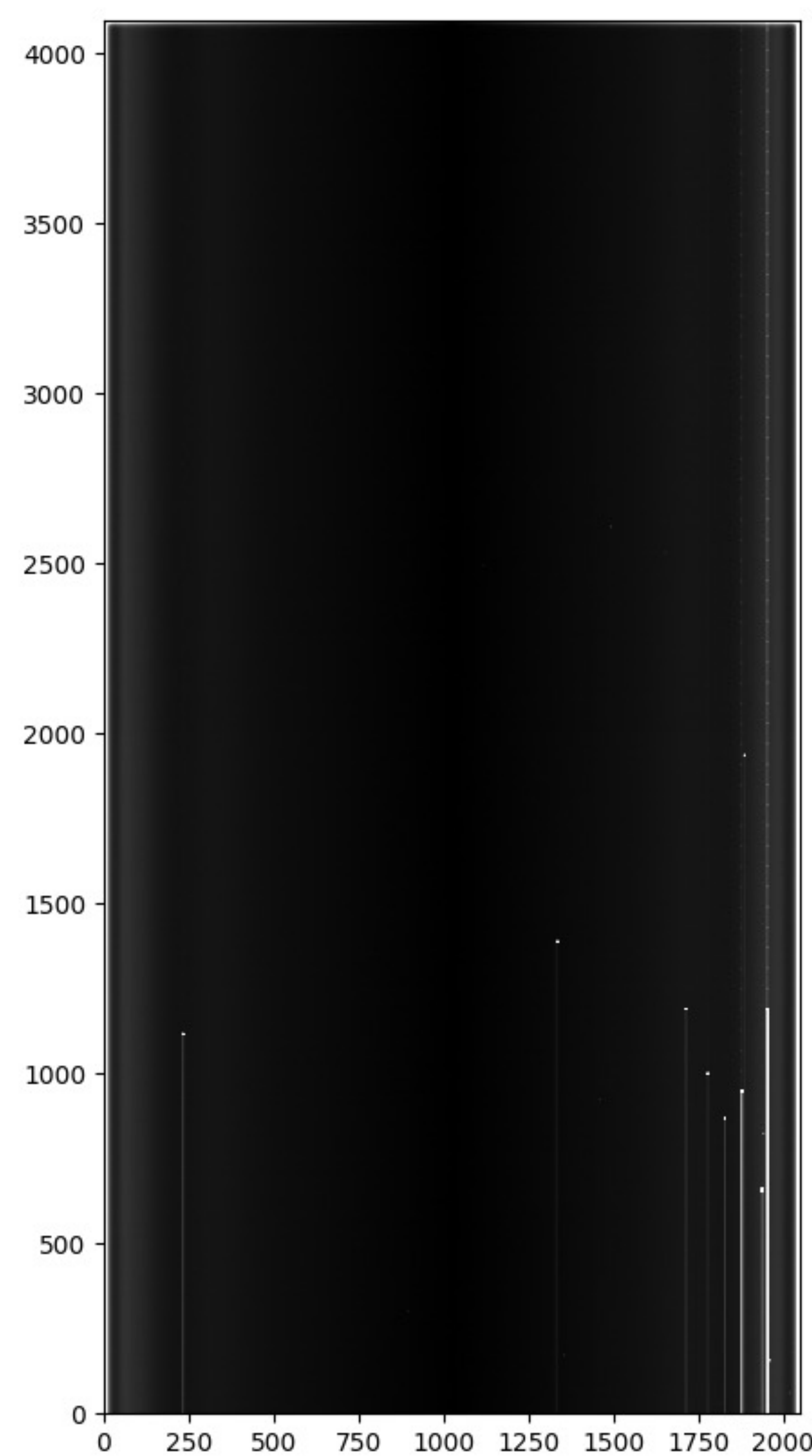
Bias Image



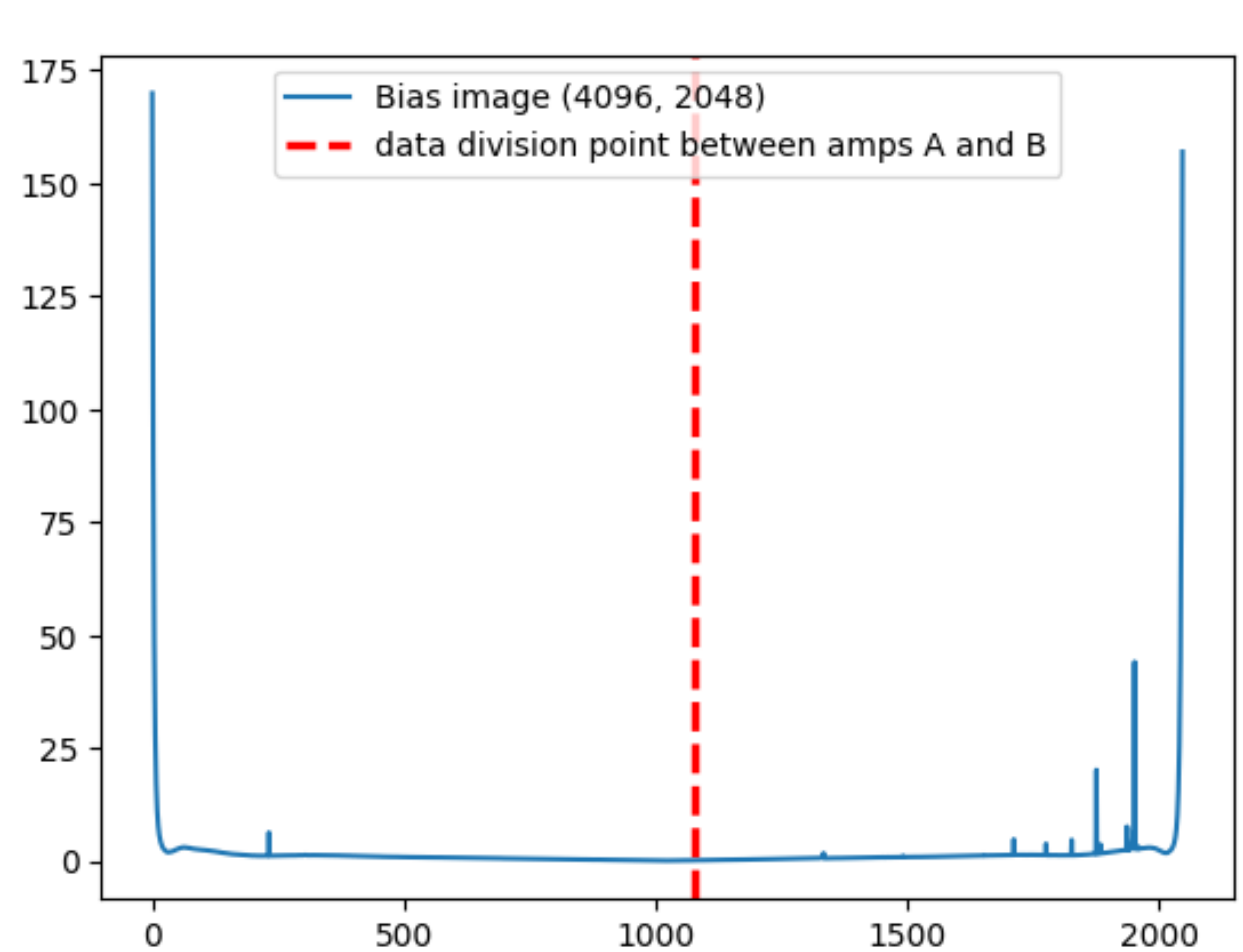
Bias Image

- Bias: Positive DC Offset applied to all pixels
- A picture taken with shutter closed and no exposure time
- Ideally would be the same for each pixel but in practice there are slight variations.
- Often numerous bias calibration pictures are taken and a combined median or averaged master bias is used (The case here).
- This combination reduces the noise.
- Dark current and hot pixels are eliminated with clipping and masking in the original pipeline
- We are ignoring these operations for simplicity

Bias Image

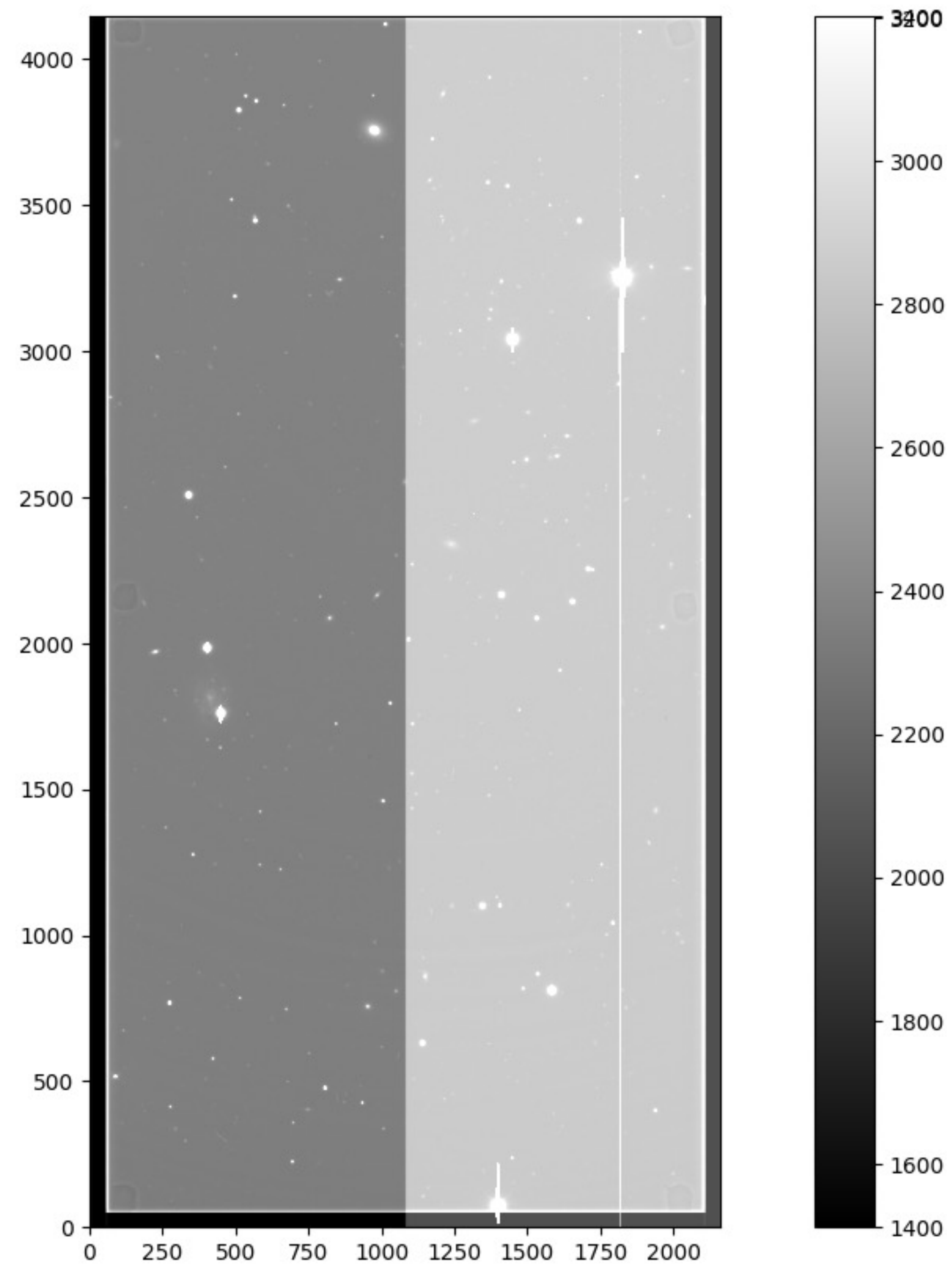


Bias Image

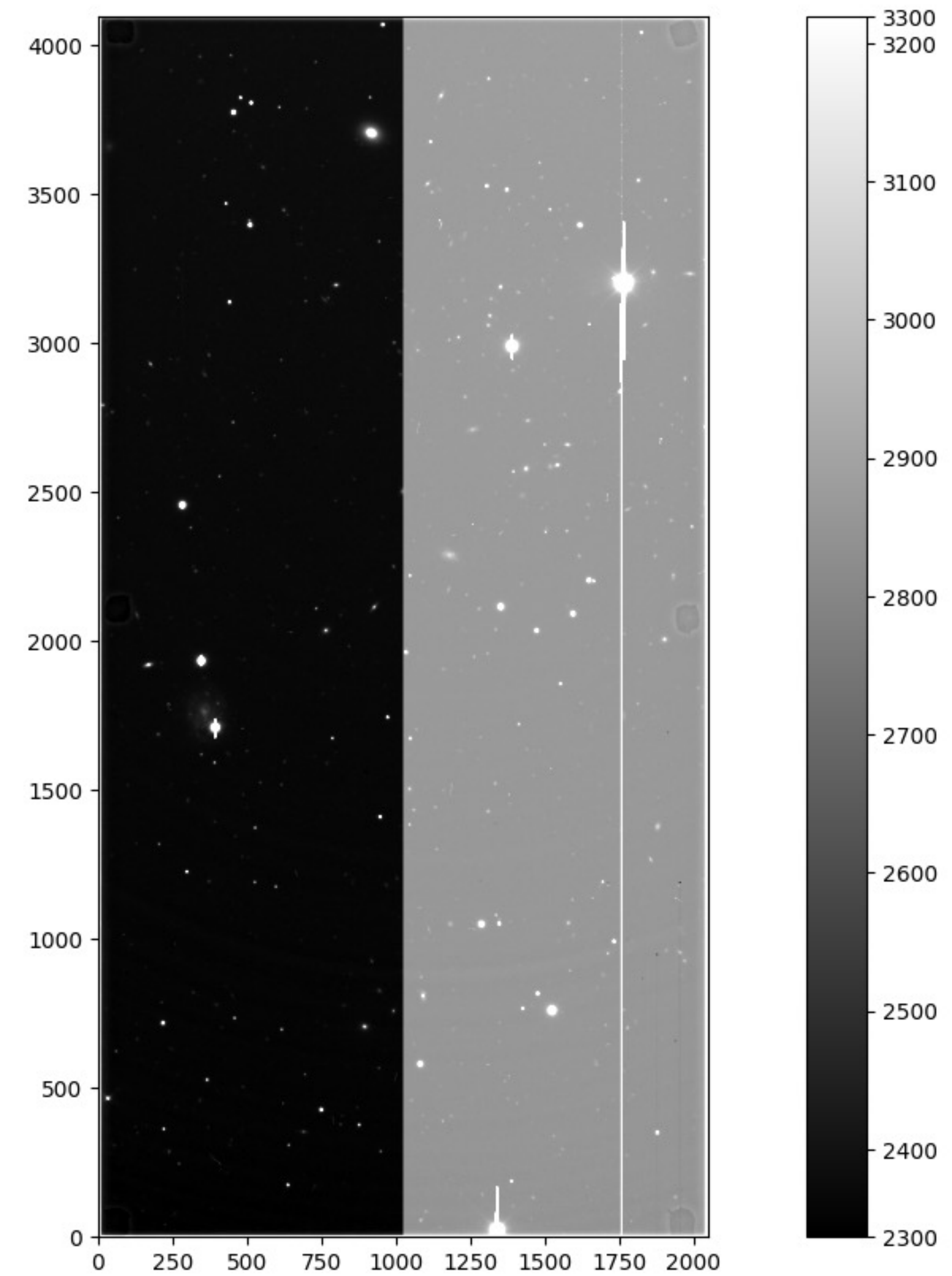


Row-Averaged Cross Section of the Bias Image

Bias Calibration (No Overscan Subtraction)

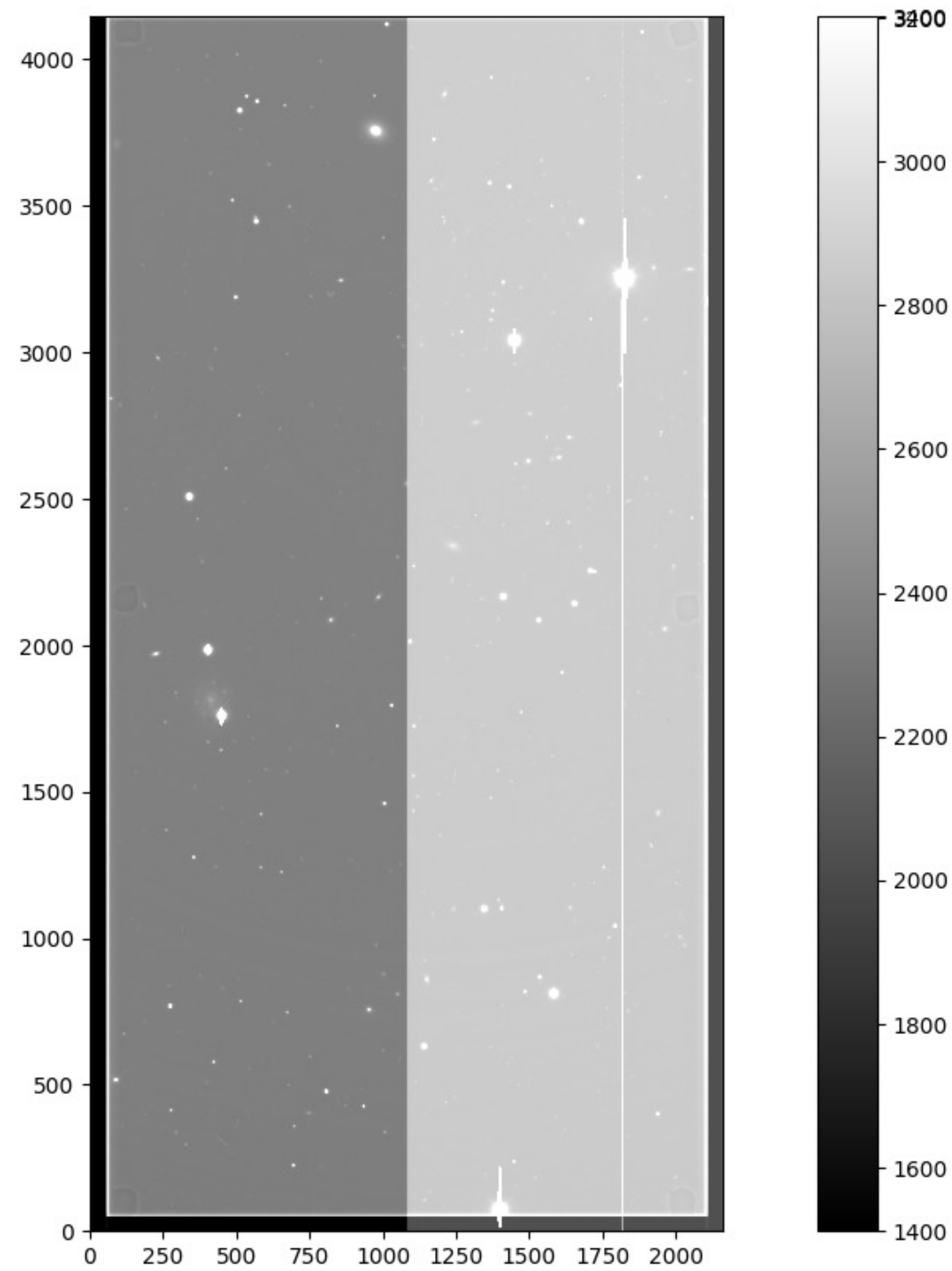


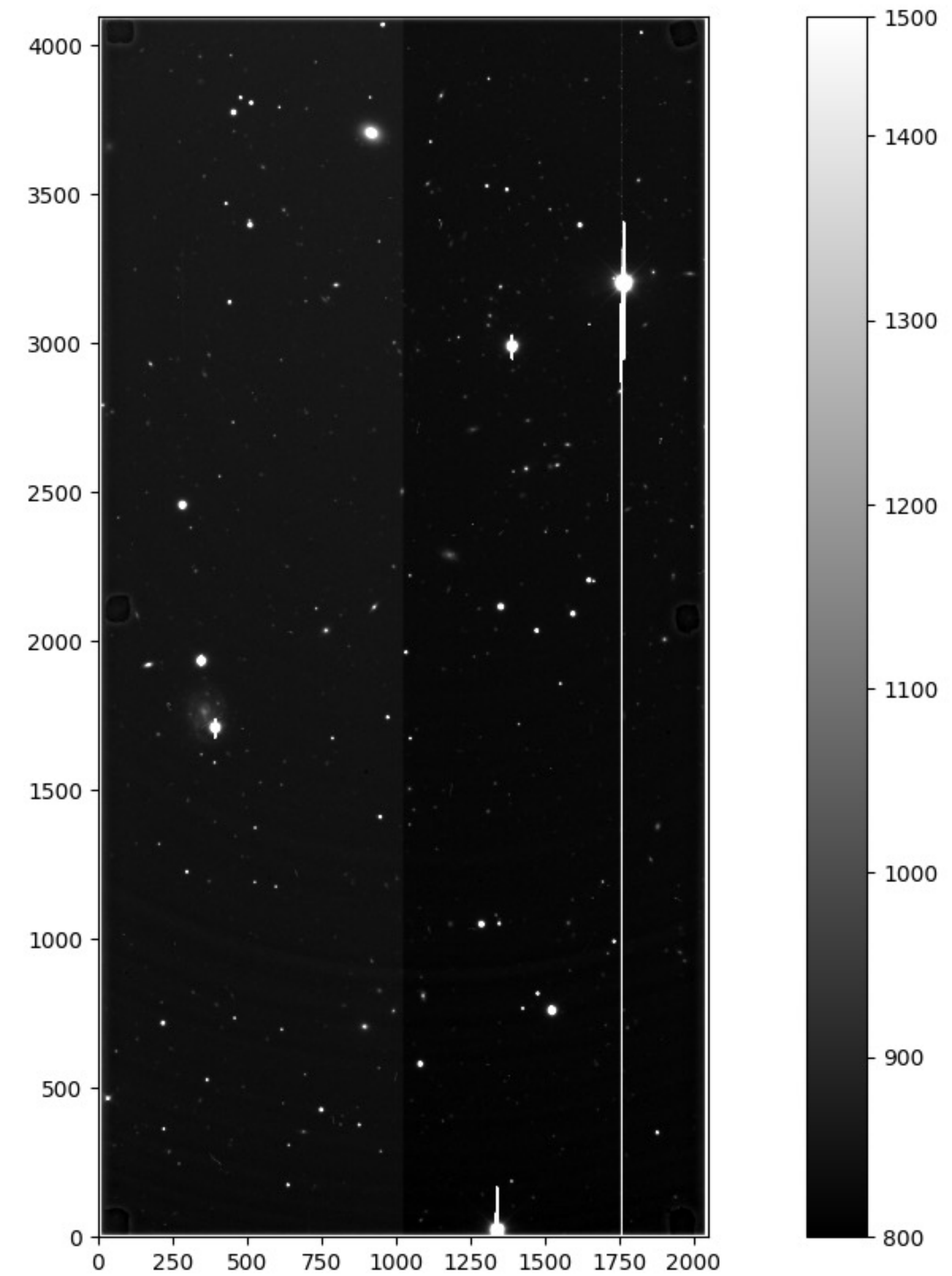
Raw Science Image



Bias Calibrated Science Image w/o OS

Bias Calibration (With Overscan Subtraction)

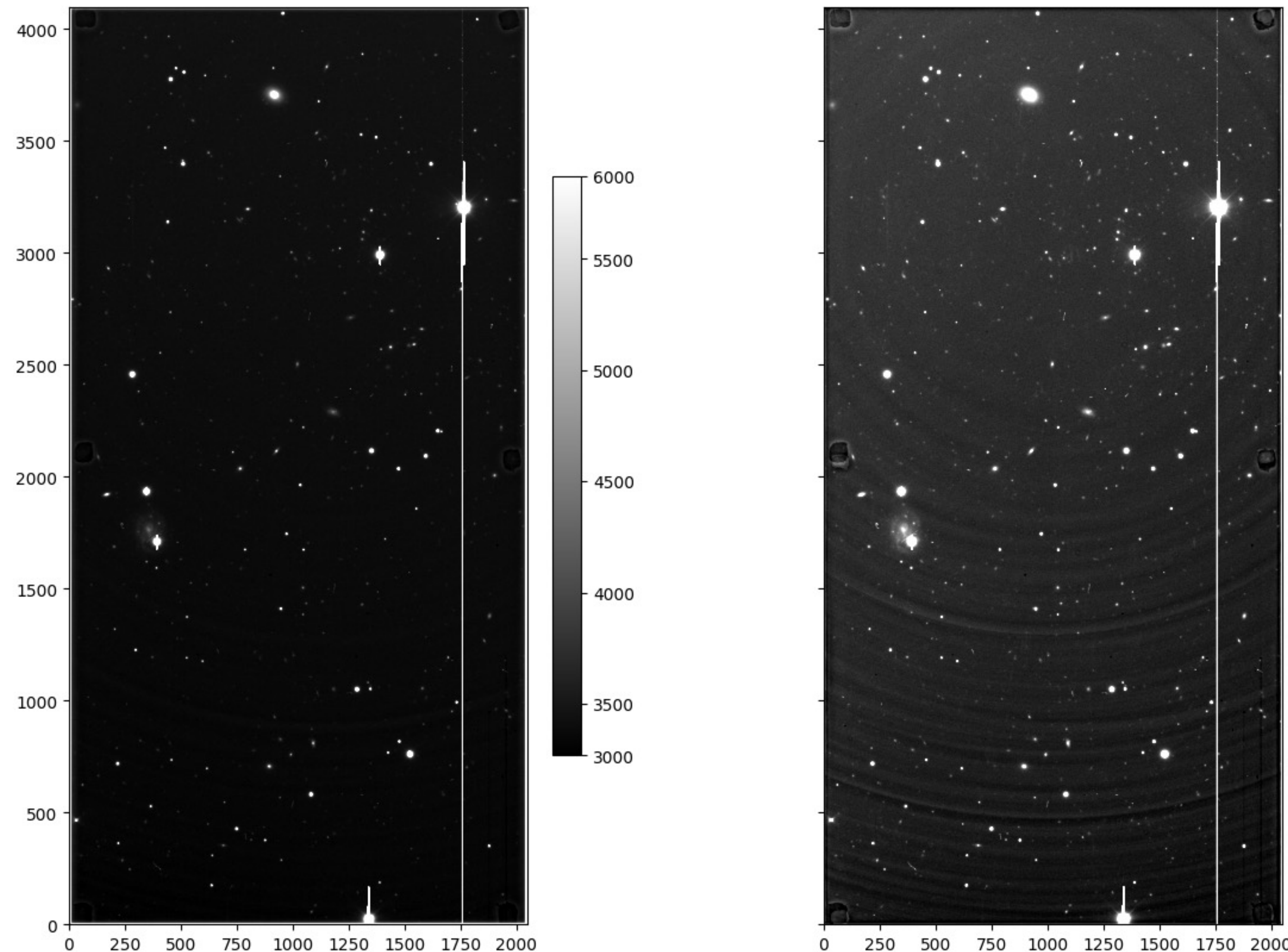




Raw Science Image

Bias Calibrated Science Image with OS

Gain, Nonlinearity, and Flat Corrections



Bias + Overscan + Gain + Nonlinearity Corrected Science Image

Bias + Overscan + Gain + Nonlinearity + Flat-field Corrected Science Image

- Gain correction: Rescaling both the regions based on the gain of amps A and B.
- In practice, this is a unit conversion from e- / ADU to e-
- Nonlinearity correction refers to the correction of the nonlinear behavior of each CCD.
- It is achieved by performing a simple interpolation utilizing a pre-calculated look-up table.
- Flat-field corrections compensate non-uniformity of the CCD response to light
- Potential non-uniformity sources are: pixel sensitivity, dust etc.
- Flat-field calibration images are practically photographs of the dome in our specific case

C++ Implementation for Testing

```
// HLS-friendly (I think?) implementation
for(int i=0; i<NRDAT; i++){
    for(int j=0; j < OVERSCANLENGTH; j++){
        overscanDataB[i][j] = sci[i + GOODROW][j + OVERSCANBSTART];
        overscanDataA[i][j] = sci[i + GOODROW][j + OVERSCANASTART];
    }
    for(int j=0; j<NCDAT/2; j++){
        calsci[i][j] = sci[i + GOODROW][j + OVERSCANBEND];
        overscanB[i] = get_median(overscanDataB[i]);
        calsci[i][j] -= overscanB[i];
        calsci[i][j] -= bias[i][j];

        int idx = int(calsci[i][j]);
        linear_correct(calsci[i][j], idx, linearB[idx], linearB[idx+1]);

        calsci[i][j] *= GAINB;
        calsci[i][j] /= flat[i][j];
    }
    for(int j=NCDAT/2; j<NCDAT; j++){
        calsci[i][j] = sci[i + GOODROW][j + OVERSCANBEND];
        overscanA[i] = get_median(overscanDataA[i]);
        calsci[i][j] -= overscanA[i];
        calsci[i][j] -= bias[i][j];

        int idx = int(calsci[i][j]);
        linear_correct(calsci[i][j], idx, linearA[idx], linearA[idx+1]);

        calsci[i][j] *= GAINA;
        calsci[i][j] /= flat[i][j];
    }
}
```

C++ Implementation for Testing

```
// HLS-friendly (I think?) implementation
for(int i=0; i<NRDAT; i++){
    for(int j=0; j < OVERSCANLENGTH; j++){
        overscanDataB[i][j] = sci[i + GOODROW][j + OVERSCANBSTART];
        overscanDataA[i][j] = sci[i + GOODROW][j + OVERSCANASTART];
    }
    for(int j=0; j<NCDAT/2; j++){
        calsci[i][j] = sci[i + GOODROW][j + OVERSCANBEND];
        overscanB[i] = get_median(overscanDataB[i]);
        calsci[i][j] -= overscanB[i];
        calsci[i][j] -= bias[i][j];

        int idx = int(calsci[i][j]);
        linear_correct(calsci[i][j], idx, linearB[idx], linearB[idx+1]);

        calsci[i][j] *= GAINB;
        calsci[i][j] /= flat[i][j];
    }
    for(int j=NCDAT/2; j<NCDAT; j++){
        calsci[i][j] = sci[i + GOODROW][j + OVERSCANBEND];
        overscanA[i] = get_median(overscanDataA[i]);
        calsci[i][j] -= overscanA[i];
        calsci[i][j] -= bias[i][j];

        int idx = int(calsci[i][j]);
        linear_correct(calsci[i][j], idx, linearA[idx], linearA[idx+1]);

        calsci[i][j] *= GAINA;
        calsci[i][j] /= flat[i][j];
    }
}
```

Overscan Extraction

Amp B

Amp A

C++ Implementation for Testing

```
// HLS-friendly (I think?) implementation
for(int i=0; i<NRDAT; i++){
    for(int j=0; j < OVERSCANLENGTH; j++){
        overscanDataB[i][j] = sci[i + GOODROW][j + OVERSCANBSTART];
        overscanDataA[i][j] = sci[i + GOODROW][j + OVERSCANASTART];
    }
    for(int j=0; j<NCDAT/2; j++){
        calsci[i][j] = sci[i + GOODROW][j + OVERSCANBEND];
        overscanB[i] = get_median(overscanDataB[i]);
        calsci[i][j] -= overscanB[i];
        calsci[i][j] -= bias[i][j];

        int idx = int(calsci[i][j]);
        linear_correct(calsci[i][j], idx, linearB[idx], linearB[idx+1]);

        calsci[i][j] *= GAINB;
        calsci[i][j] /= flat[i][j];
    }
    for(int j=NCDAT/2; j<NCDAT; j++){
        calsci[i][j] = sci[i + GOODROW][j + OVERSCANBEND];
        overscanA[i] = get_median(overscanDataA[i]);
        calsci[i][j] -= overscanA[i];
        calsci[i][j] -= bias[i][j];

        int idx = int(calsci[i][j]);
        linear_correct(calsci[i][j], idx, linearA[idx], linearA[idx+1]);

        calsci[i][j] *= GAINA;
        calsci[i][j] /= flat[i][j];
    }
}
```

Amp B

Amp A

Overscan Extraction

Overscan Subtraction

Overscan Subtraction

Fermilab

C++ Implementation for Testing

```
// HLS-friendly (I think?) implementation
for(int i=0; i<NRDAT; i++){
    for(int j=0; j < OVERSCANLENGTH; j++){
        overscanDataB[i][j] = sci[i + GOODROW][j + OVERSCANBSTART];
        overscanDataA[i][j] = sci[i + GOODROW][j + OVERSCANASTART];
    }
    for(int j=0; j<NCDAT/2; j++){
        calsci[i][j] = sci[i + GOODROW][j + OVERSCANBEND];
        overscanB[i] = get_median(overscanDataB[i]);
        calsci[i][j] -= overscanB[i];
        calsci[i][j] -= bias[i][j];

        int idx = int(calsci[i][j]);
        linear_correct(calsci[i][j], idx, linearB[idx], linearB[idx+1]);

        calsci[i][j] *= GAINB;
        calsci[i][j] /= flat[i][j];
    }
    for(int j=NCDAT/2; j<NCDAT; j++){
        calsci[i][j] = sci[i + GOODROW][j + OVERSCANBEND];
        overscanA[i] = get_median(overscanDataA[i]);
        calsci[i][j] -= overscanA[i];
        calsci[i][j] -= bias[i][j];

        int idx = int(calsci[i][j]);
        linear_correct(calsci[i][j], idx, linearA[idx], linearA[idx+1]);

        calsci[i][j] *= GAINA;
        calsci[i][j] /= flat[i][j];
    }
}
```

Amp B

Amp A

Overscan Extraction

Overscan Subtraction

Bias Subtraction

Overscan Subtraction

Bias Subtraction

C++ Implementation for Testing

```
// HLS-friendly (I think?) implementation
for(int i=0; i<NRDAT; i++){
    for(int j=0; j < OVERSCANLENGTH; j++){
        overscanDataB[i][j] = sci[i + GOODROW][j + OVERSCANBSTART];
        overscanDataA[i][j] = sci[i + GOODROW][j + OVERSCANASTART];
    }
    for(int j=0; j<NCDAT/2; j++){
        calsci[i][j] = sci[i + GOODROW][j + OVERSCANBEND];
        overscanB[i] = get_median(overscanDataB[i]);
        calsci[i][j] -= overscanB[i];
        calsci[i][j] -= bias[i][j];

        int idx = int(calsci[i][j]);
        linear_correct(calsci[i][j], idx, linearB[idx], linearB[idx+1]);

        calsci[i][j] *= GAINB;
        calsci[i][j] /= flat[i][j];
    }
    for(int j=NCDAT/2; j<NCDAT; j++){
        calsci[i][j] = sci[i + GOODROW][j + OVERSCANBEND];
        overscanA[i] = get_median(overscanDataA[i]);
        calsci[i][j] -= overscanA[i];
        calsci[i][j] -= bias[i][j];

        int idx = int(calsci[i][j]);
        linear_correct(calsci[i][j], idx, linearA[idx], linearA[idx+1]);

        calsci[i][j] *= GAINA;
        calsci[i][j] /= flat[i][j];
    }
}
```

Amp B

Amp A

Overscan Extraction

Overscan Subtraction

Bias Subtraction

Nonlinearity Correction

Overscan Subtraction

Bias Subtraction

Nonlinearity Correction

C++ Implementation for Testing

```
// HLS-friendly (I think?) implementation
for(int i=0; i<NRDAT; i++){
    for(int j=0; j < OVERSCANLENGTH; j++){
        overscanDataB[i][j] = sci[i + GOODROW][j + OVERSCANBSTART];
        overscanDataA[i][j] = sci[i + GOODROW][j + OVERSCANASTART];
    }
    for(int j=0; j<NCDAT/2; j++){
        calsci[i][j] = sci[i + GOODROW][j + OVERSCANBEND];
        overscanB[i] = get_median(overscanDataB[i]);
        calsci[i][j] -= overscanB[i];
        calsci[i][j] -= bias[i][j];

        int idx = int(calsci[i][j]);
        linear_correct(calsci[i][j], idx, linearB[idx], linearB[idx+1]);

        calsci[i][j] *= GAINB;
        calsci[i][j] /= flat[i][j];
    }
    for(int j=NCDAT/2; j<NCDAT; j++){
        calsci[i][j] = sci[i + GOODROW][j + OVERSCANBEND];
        overscanA[i] = get_median(overscanDataA[i]);
        calsci[i][j] -= overscanA[i];
        calsci[i][j] -= bias[i][j];

        int idx = int(calsci[i][j]);
        linear_correct(calsci[i][j], idx, linearA[idx], linearA[idx+1]);

        calsci[i][j] *= GAINA;
        calsci[i][j] /= flat[i][j];
    }
}
```

Amp B

Overscan Extraction

Overscan Subtraction

Bias Subtraction

Nonlinearity Correction

Gain & Flat Corrections

Amp A

Overscan Subtraction

Bias Subtraction

Nonlinearity Correction

Gain & Flat Corrections

HLS Implementation of the Top Function

Transposed Inputs

```
void pixcor(data_t sci_in[NCOL][NROW], data_t bias[TN][TM], data_t flat[TN][TM], data_t sci_out[TN][TM]){

#pragma HLS interface mode=ap_fifo port=sci_in
#pragma HLS interface mode=ap_fifo port=bias
#pragma HLS interface mode=ap_fifo port=flat

#pragma HLS loop_merge
LOOP_AMP_B:
    for(int i=0; i < TM*TN/2; i++){
#pragma HLS pipeline II=1
        int k = int(i / TM);
        int j = i % TM;

        auto tmp0 = sci_in[k + OVERSCANBEND][j + GOODROW];
        tmp0 *= BIASSCALE;
        auto tmp1 = bias[k][j];
        auto tmp2 = flat[k][j];

        sci_out[k][j] = (tmp0 - tmp1)*GAINB/tmp2;
    }
LOOP_AMP_A:
    for(int i=TM*TN/2; i < TM*TN; i++){
#pragma HLS pipeline II=1
        int k = int(i / TM);
        int j = i % TM;

        auto tmp0 = sci_in[k + OVERSCANBEND][j + GOODROW];
        tmp0 *= BIASSCALE;
        auto tmp1 = bias[k][j];
        auto tmp2 = flat[k][j];

        sci_out[k][j] = (tmp0 - tmp1)*GAINA/tmp2;
    }
}
```

Inputs are streamed

Amp B

Amp A

Fermilab

HLS Implementation of the Top Function (Interfaces)

- Interfaces: Designed channels for data flow into or out of the design.
- Port protocols control the data flow.
- Some possible options: ap_none (no protocol), ap_fifo (std FIFO), ap_memory (RAM)…
- Initially selected port protocol: ap_fifo (default streaming protocol)
- No special reason behind this choice: They are just easier to work with initially.
- This choice is definitely going to change at later stages.
- Eventual aim is P2P communication between storage device and FPGAs.

HLS Implementation of the Top Function

```
void pixcor(data_t sci_in[NCOL][NROW], data_t bias[TN][TM], data_t flat[TN][TM], data_t sci_out[TN][TM]){

#pragma HLS interface mode=ap_fifo port=sci_in
#pragma HLS interface mode=ap_fifo port=bias
#pragma HLS interface mode=ap_fifo port=flat

#pragma HLS loop_merge
LOOP_AMP_B:
    for(int i=0; i < TM*TN/2; i++){
#pragma HLS pipeline II=1
        int k = int(i / TM);
        int j = i % TM;

        auto tmp0 = sci_in[k + OVERSCANBEND][j + GOODROW];
        tmp0 *= BIASSCALE;
        auto tmp1 = bias[k][j];
        auto tmp2 = flat[k][j];

        sci_out[k][j] = (tmp0 - tmp1)*GAINB/tmp2;
    }
LOOP_AMP_A:
    for(int i=TM*TN/2; i < TM*TN; i++){
#pragma HLS pipeline II=1
        int k = int(i / TM);
        int j = i % TM;

        auto tmp0 = sci_in[k + OVERSCANBEND][j + GOODROW];
        tmp0 *= BIASSCALE;
        auto tmp1 = bias[k][j];
        auto tmp2 = flat[k][j];

        sci_out[k][j] = (tmp0 - tmp1)*GAINA/tmp2;
    }
}
```

Amp B

Amp A

Transposed Inputs

Inputs are streamed

Amp Loops are Merged

Amp Loops are Merged

HLS Implementation of the Top Function

```
void pixcor(data_t sci_in[NCOL][NROW], data_t bias[TN][TM], data_t flat[TN][TM], data_t sci_out[TN][TM]){

#pragma HLS interface mode=ap_fifo port=sci_in
#pragma HLS interface mode=ap_fifo port=bias
#pragma HLS interface mode=ap_fifo port=flat

#pragma HLS loop_merge
LOOP_AMP_B:
    for(int i=0; i < TM*TN/2; i++){
#pragma HLS pipeline II=1
        int k = int(i / TM);
        int j = i % TM;

        auto tmp0 = sci_in[k + OVERSCANBEND][j + GOODROW];
        tmp0 *= BIASSCALE;
        auto tmp1 = bias[k][j];
        auto tmp2 = flat[k][j];

        sci_out[k][j] = (tmp0 - tmp1)*GAINB/tmp2;
    }
LOOP_AMP_A:
    for(int i=TM*TN/2; i < TM*TN; i++){
#pragma HLS pipeline II=1
        int k = int(i / TM);
        int j = i % TM;

        auto tmp0 = sci_in[k + OVERSCANBEND][j + GOODROW];
        tmp0 *= BIASSCALE;
        auto tmp1 = bias[k][j];
        auto tmp2 = flat[k][j];

        sci_out[k][j] = (tmp0 - tmp1)*GAINA/tmp2;
    }
}
```

Amp B

Amp A

Transposed Inputs

Inputs are streamed

Amp Loops are Merged

Each Loop is Flattened And Pipelined

Amp Loops are Merged

Each Loop is Flattened And Pipelined

HLS Implementation of the Top Function (Pipelining)

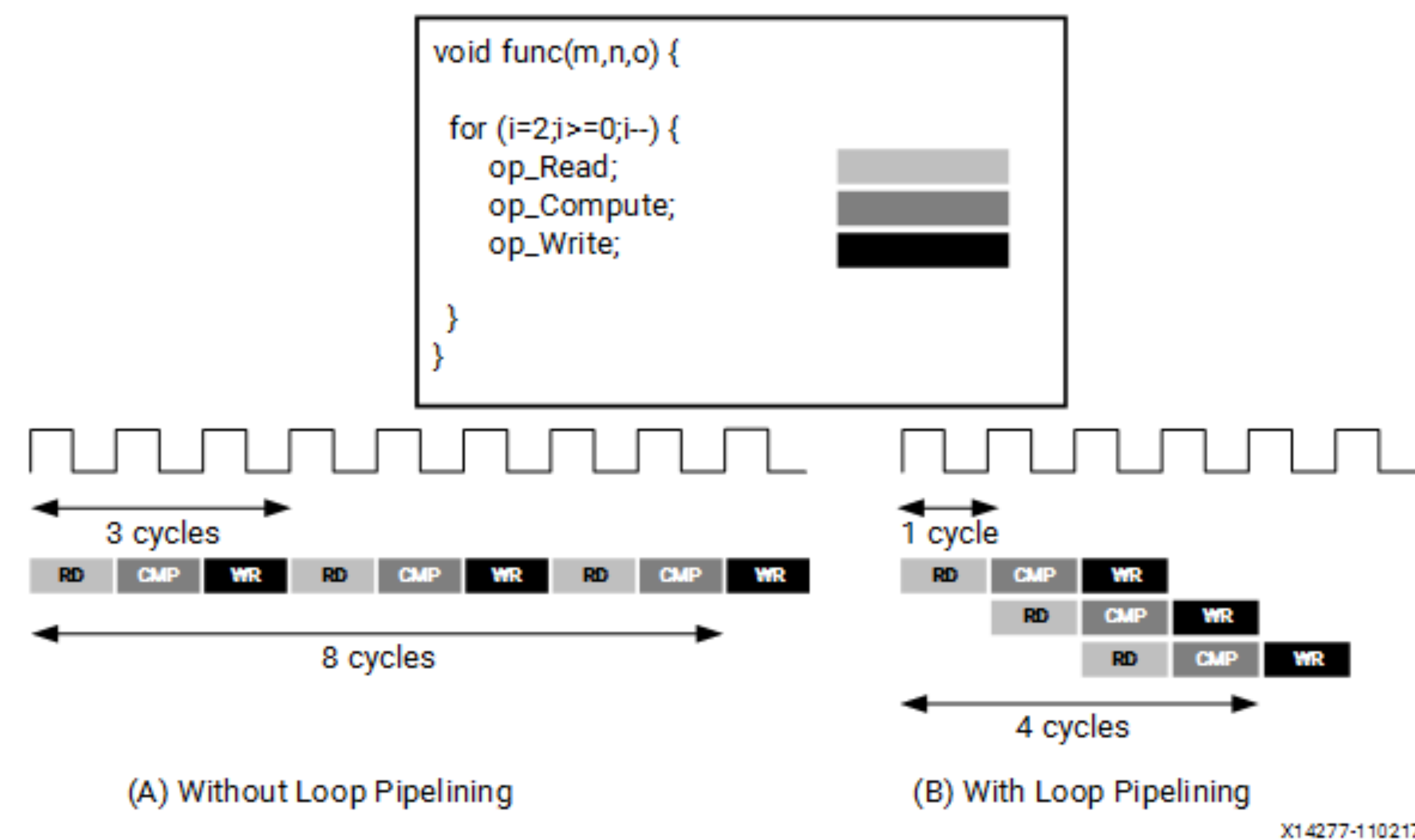


Illustration of the Behavior of the pipeline directive.
(Adapted from https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/pragma-HLS-pipeline)

- “Pipelining a loop allows the operations of the loop to be implemented in a concurrent manner”
- Measure of concurrency: Initiation Interval (II)
- II: Launch frequency between consecutive iterations.
- W/o pipelining RD, CMP, and WR operations have II: 3
- With pipelining, II: 1 (The ideal scenario)
- Obviously has direct effect on the final latency of the design

HLS Implementation of the Top Function

```
void pixcor(data_t sci_in[NCOL][NROW], data_t bias[TN][TM], data_t flat[TN][TM], data_t sci_out[TN][TM]){

#pragma HLS interface mode=ap_fifo port=sci_in
#pragma HLS interface mode=ap_fifo port=bias
#pragma HLS interface mode=ap_fifo port=flat

#pragma HLS loop_merge
LOOP_AMP_B:
    for(int i=0; i < TM*TN/2; i++){
#pragma HLS pipeline II=1
        int k = int(i / TM);
        int j = i % TM;

        auto tmp0 = sci_in[k + OVERSCANBEND][j + GOODROW];
        tmp0 *= BIASSCALE;
        auto tmp1 = bias[k][j];
        auto tmp2 = flat[k][j];

        sci_out[k][j] = (tmp0 - tmp1)*GAINB/tmp2;
    }
LOOP_AMP_A:
    for(int i=TM*TN/2; i < TM*TN; i++){
#pragma HLS pipeline II=1
        int k = int(i / TM);
        int j = i % TM;

        auto tmp0 = sci_in[k + OVERSCANBEND][j + GOODROW];
        tmp0 *= BIASSCALE;
        auto tmp1 = bias[k][j];
        auto tmp2 = flat[k][j];

        sci_out[k][j] = (tmp0 - tmp1)*GAINA/tmp2;
    }
}
```

Amp B

Amp A

Transposed Inputs

Inputs are streamed

Amp Loops are Merged

Each Loop is Flattened And Pipelined

Core Operations

Amp Loops are Merged

Each Loop is Flattened And Pipelined

Core Operations

HLS Implementation of the Top Function (Performance)

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
pixcor				-	8388819	4.194E7	-	8388820	-	no	2	8	2906	4173	0
pixcor_Pipeline_LOOP_FIRSTHALF				-	4194407	2.097E7	-	4194407	-	no	0	0	857	1423	0
LOOP_FIRSTHALF				-	4194405	2.097E7	103	1	4194304	yes	-	-	-	-	-
pixcor_Pipeline_LOOP_SECONDHALF				-	4194407	2.097E7	-	4194407	-	no	0	0	859	1425	0
LOOP_SECONDHALF				-	4194405	2.097E7	103	1	4194304	yes	-	-	-	-	-

- We are only applying the simplest corrections, I.e. no parallelized overscan subtraction or linear correction (yet).
- With overscan subtraction and linear correction the latency increased to 0.1 seconds (not reporting).
- Recall that the science image is in 4K -> There are ~8.5 million pixels to concurrently operate on
- So, even with II = 1, T = 5 ns, and perfect efficiency, concurrent operations on all the pixels gives the latency ~50 ms.
- Vanilla C++ test code takes ~5 secs to run, so there is a considerable improvement.
- Area use seems to be efficient, but that is expected given the ultra-simplistic design.

Concluding Remarks

- Disclaimer: This project is still at a very early stage and work-in progress and there are many aspects to improve and/or modify. We expect to keep working on improvements during the Fall and possibly beyond.
- The steps were: Learn CCD image processing fundamentals —> Learn the basics of HLS. —> Pixcorrect (https://github.com/DarkEnergySurvey/pixcorrect) —> Python —> C++ —> HLS
- Next mandatory step is to use ap_types instead of integers to get a better performance estimate and eliminate some overflow issues during rescaling process.
- We still need to figure out a smart way to implement parallelized overscan subtraction.
- The primary issue with parallelized OS: Calculating the median of each row/column is currently creating a massive bottleneck.
- Concurrent nonlinearity correction should be simple yet it doesn't mean much to implement it without the previous calibration operations
- After resolving all of these issues, we furthermore aim to integrate P2P communication into the picture.

Fermilab | U.S. DEPARTMENT OF ENERGY | Office of Science

THANKS!