Using Bifrost for Synthetic Aperture Radar

Seth Bruzewski
MJD 60097

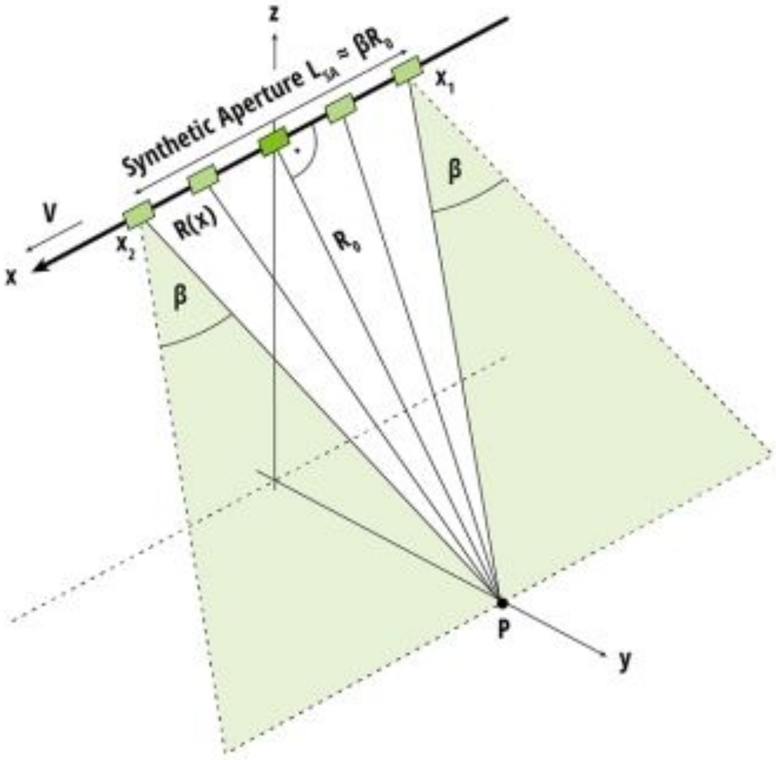
What is Synthetic Aperture Radar?

SAR

- Started in the 90s
- Satellites bounce radar off the Earth
- Amount of returned energy tells you about ground conditions
- Can run during bad weather or nighttime, unlike optical

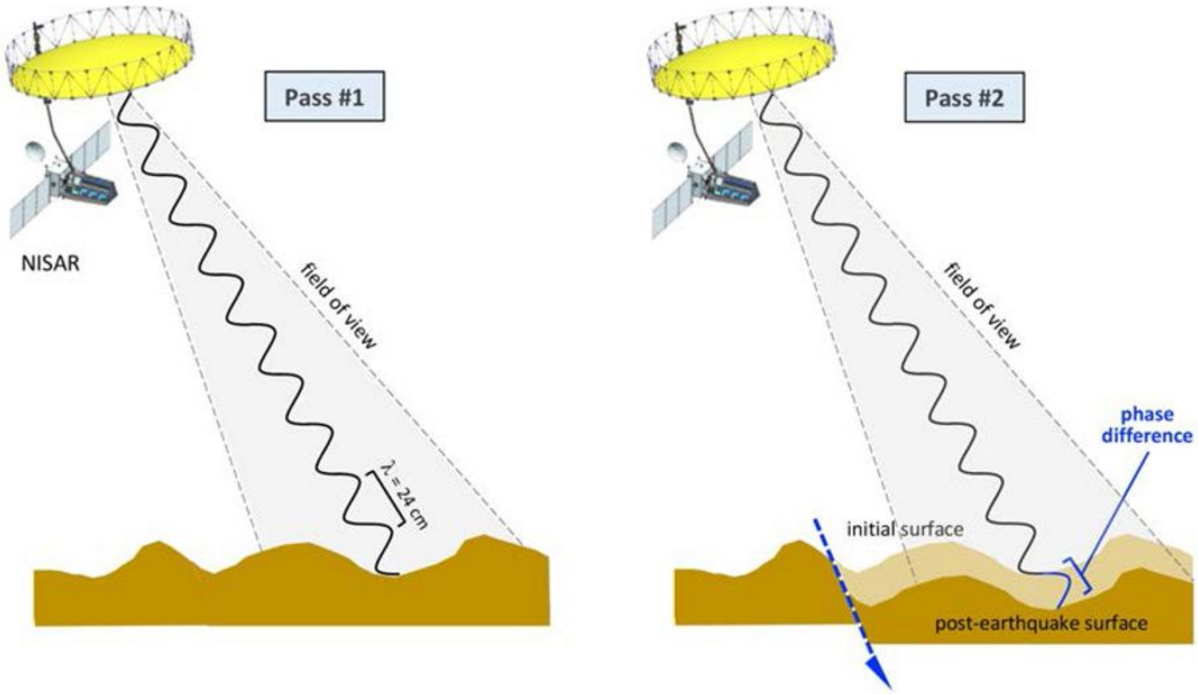
What's Synthetic?

- "Synthetic aperture"
- Same point observed multiple times during a single pass
- Combine together for much higher resolution
- Acts like a single antenna, typically kilometers long



Second Pass

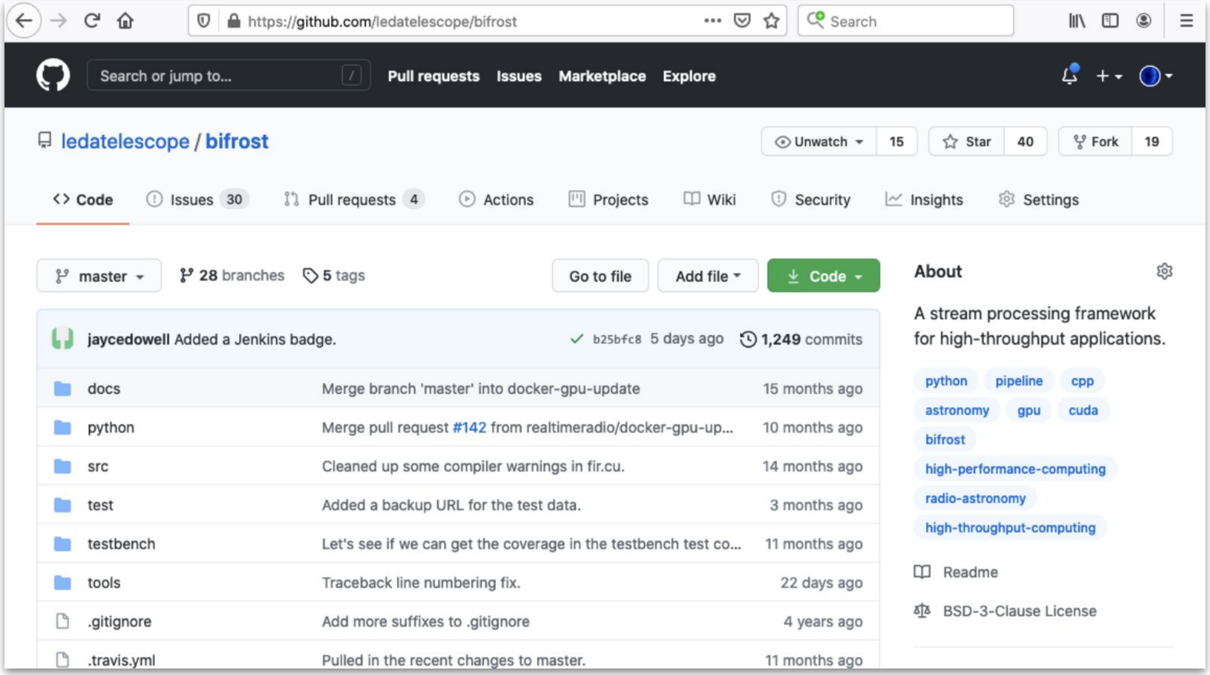
- During next pass, any new deformation will induce a phase change
- Compare the phases of two images (interfere them), and you can model said deformation
- Called **Interferometric Synthetic Aperture Radar** (InSAR)



What is Bifrost?

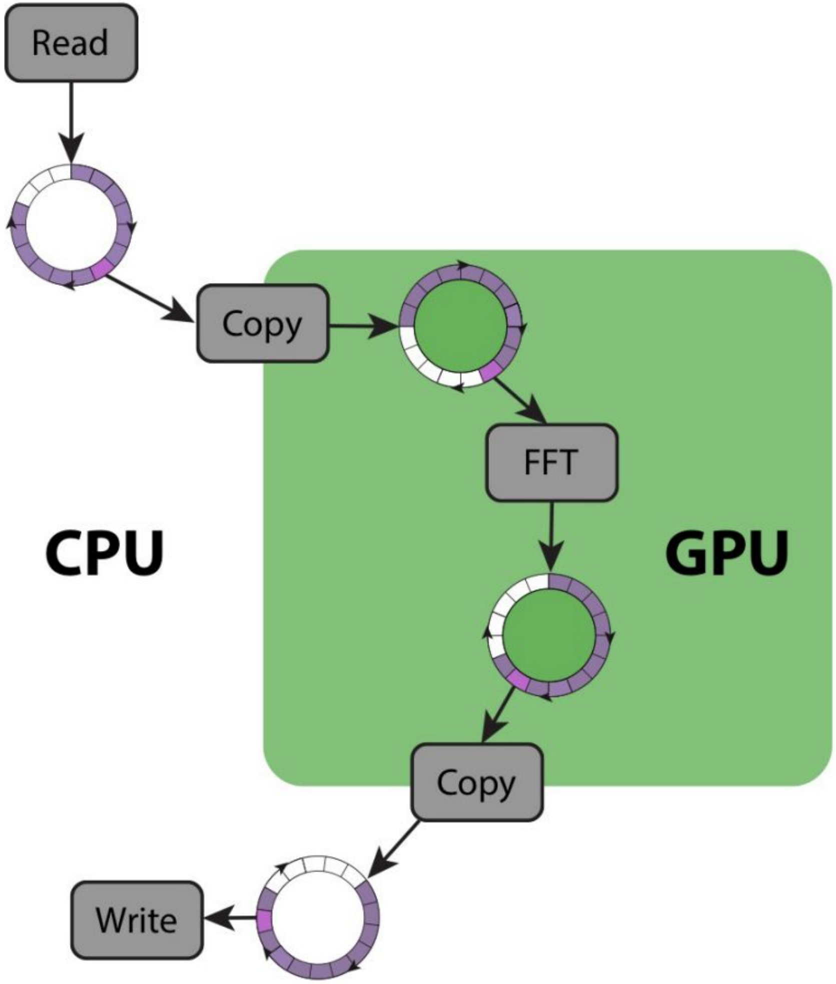
Bifrost is

- “A stream processing framework for high-throughput data”
- Lets you build CPU/GPU pipelines for data processing
- Python frontend, C++/CUDA backend



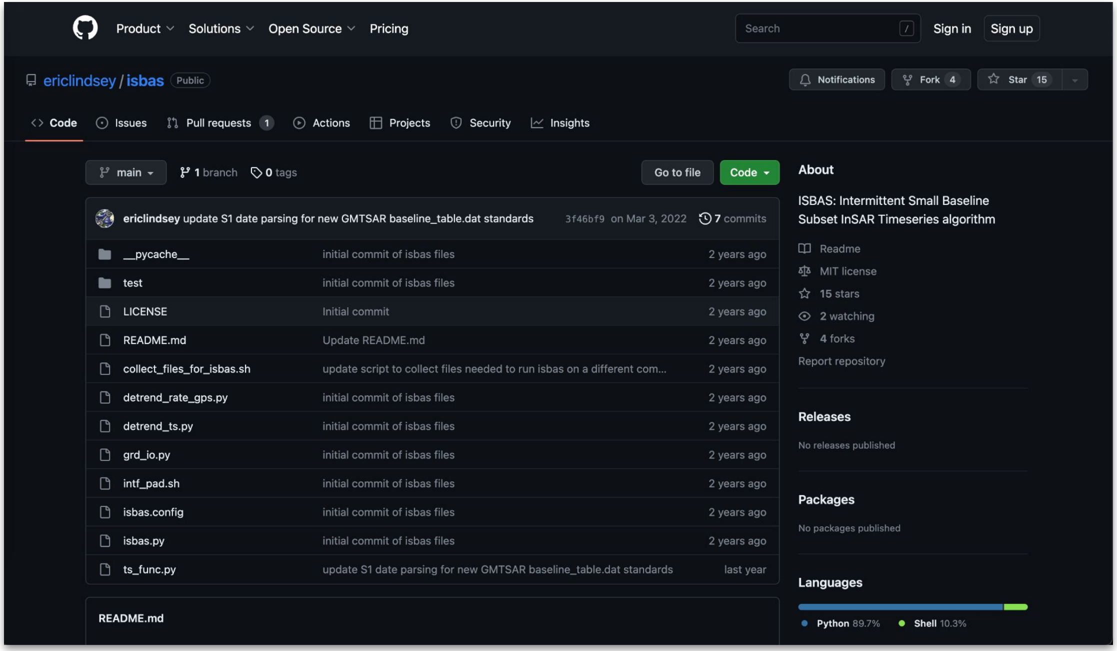
Setup

- Bifrost is built around ring buffers and “blocks” that handle those buffers
- A pipeline sticks together blocks, sometimes custom, to process data
- Cupy integration makes GPU processing super simple
    - np.func() → cp.func()

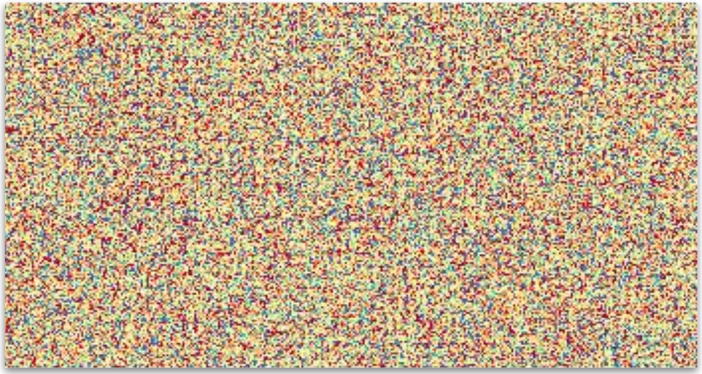


Bifrost Outside Astro

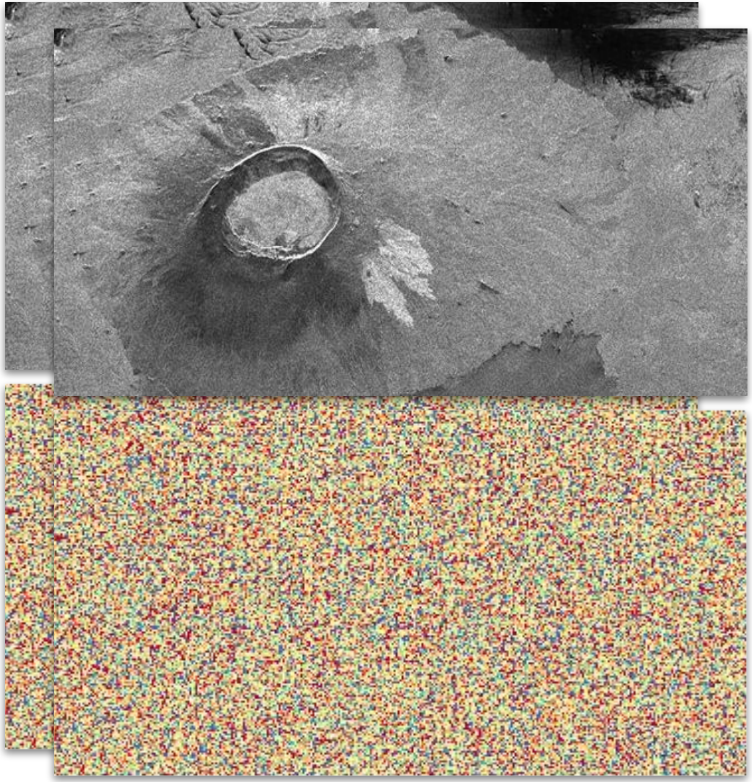
- Bifrost is mainly used for astronomy at the moment
  - Probably hear about it a few times today
- But could easily be useful for other fields
- **Goal: demonstrate Bifrost utility on InSAR data**
- Let’s try to speed up some processing



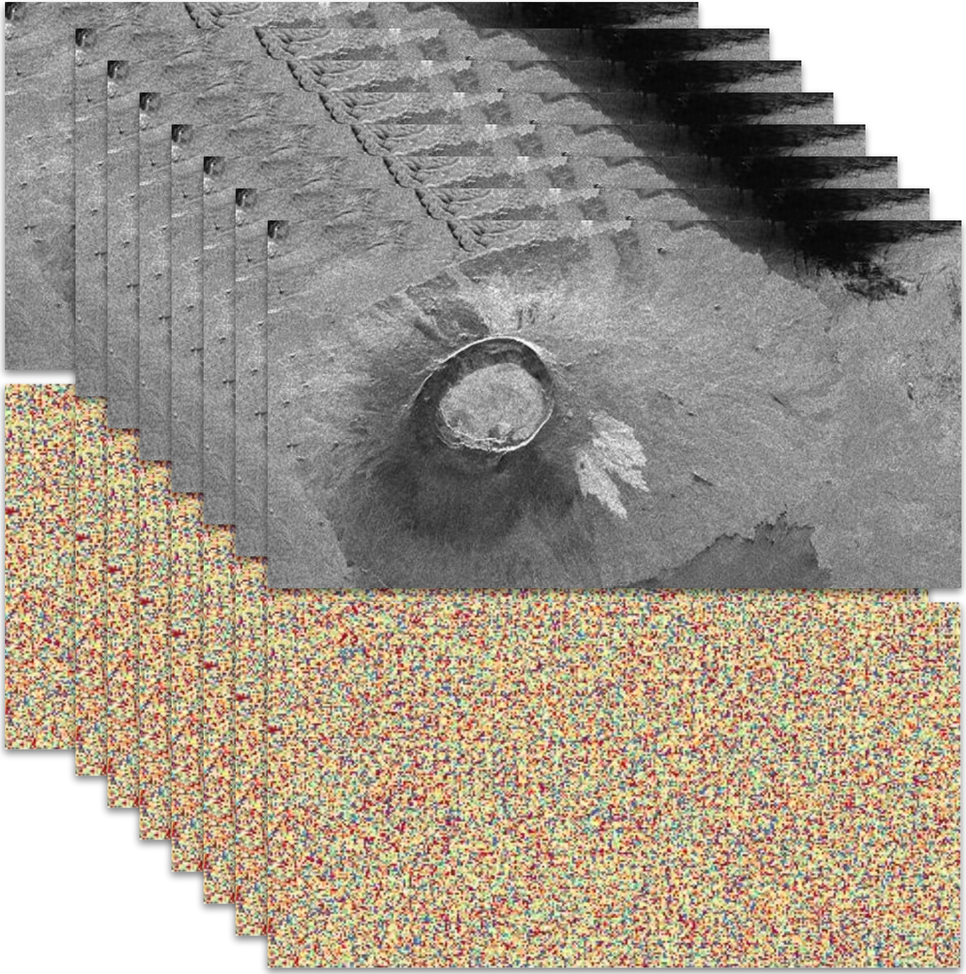
Processing Overview



Take Data

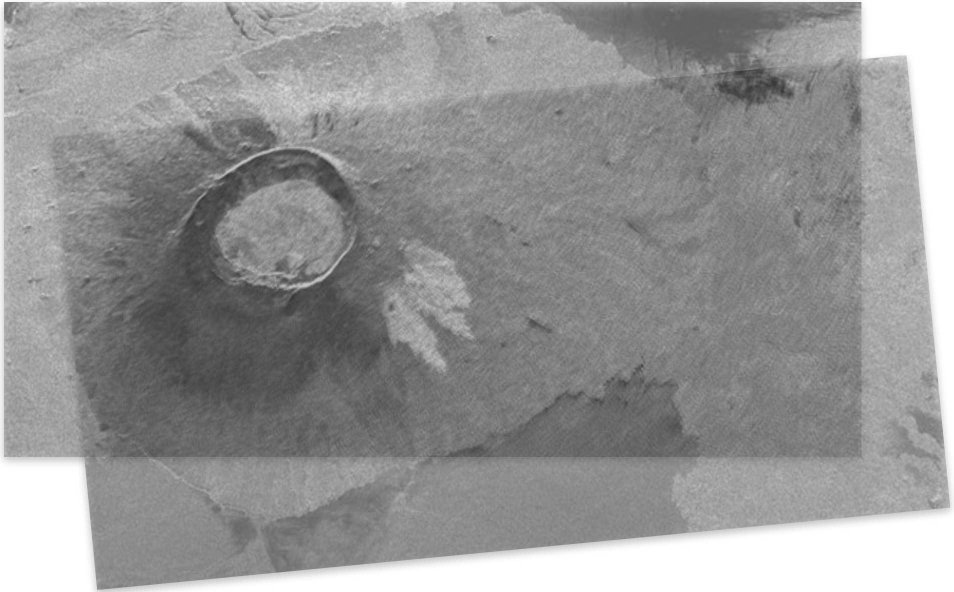


Take Data



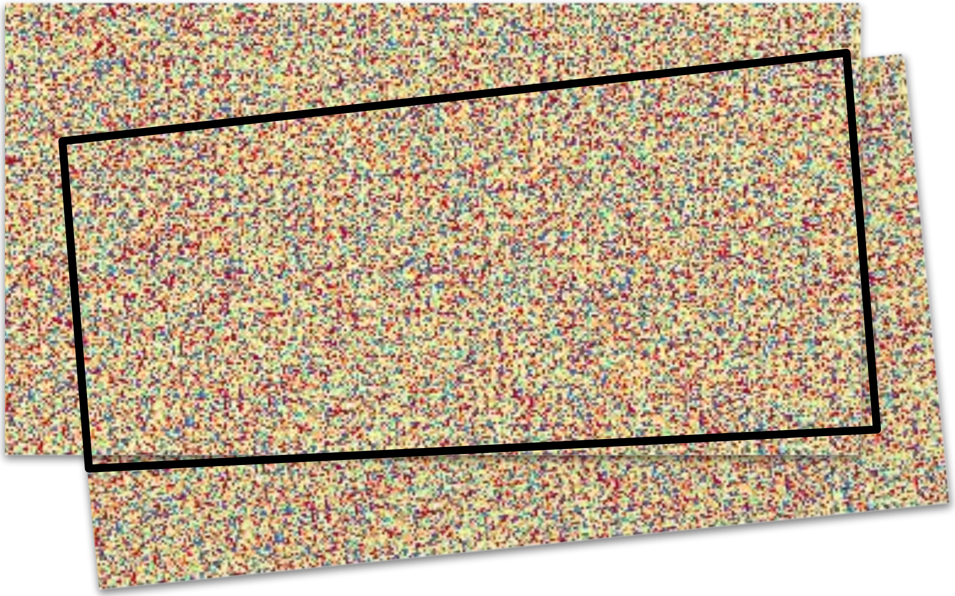
Take Data

Align Images

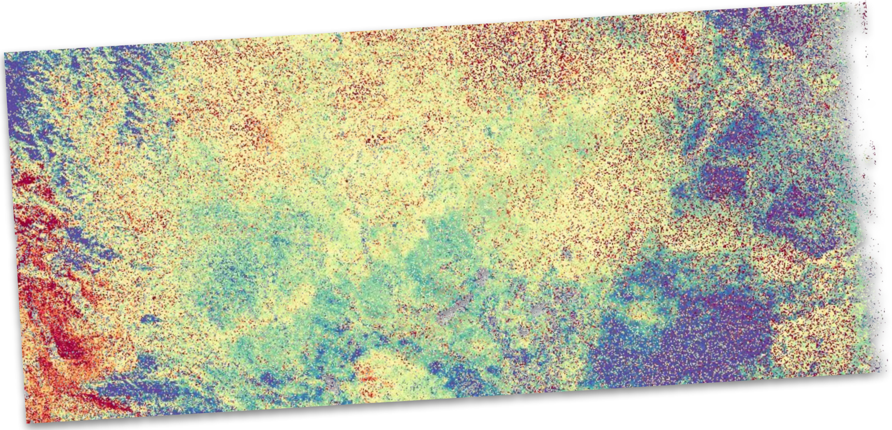


Align Images

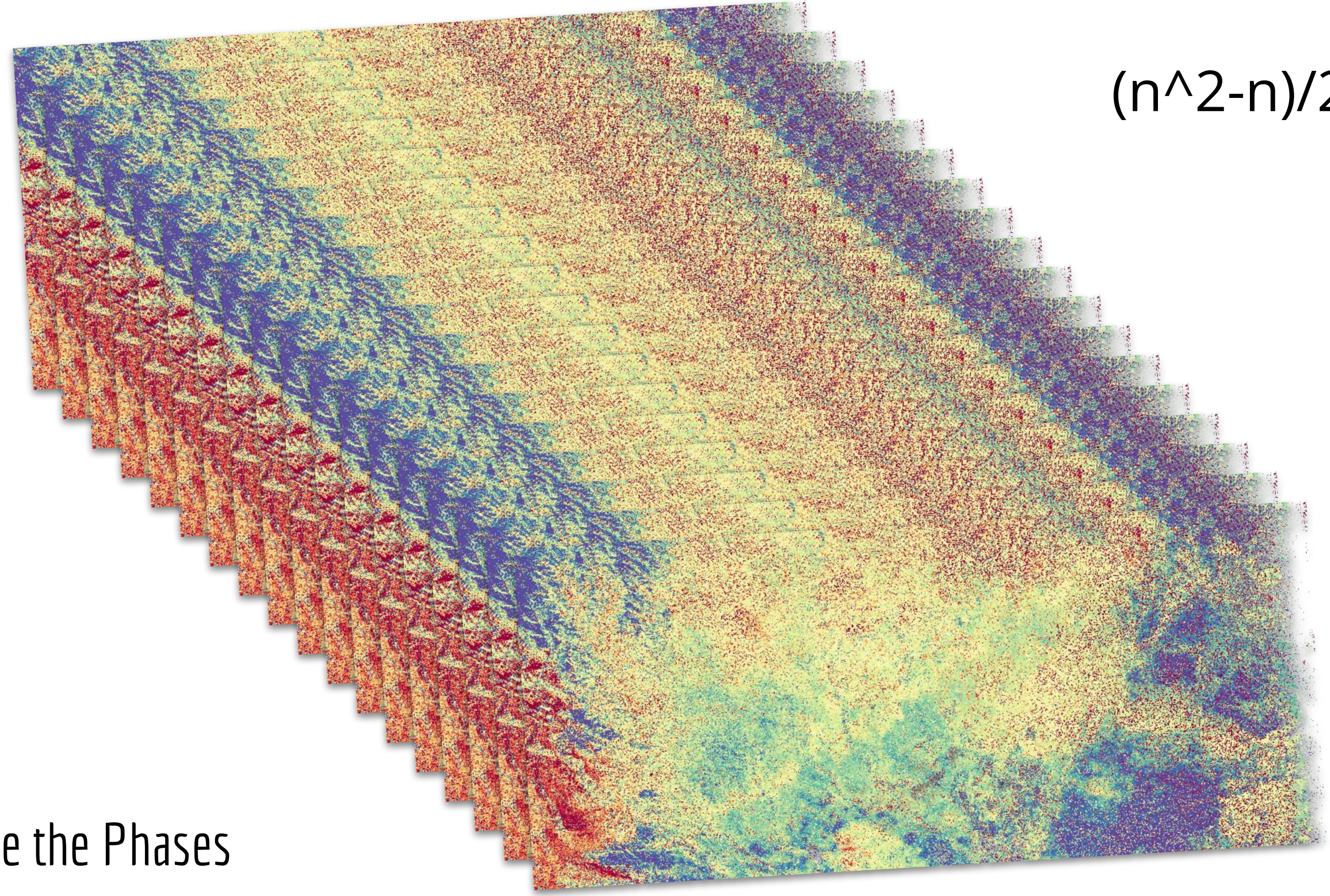
Align Images



Align Images



Interfere the Phases

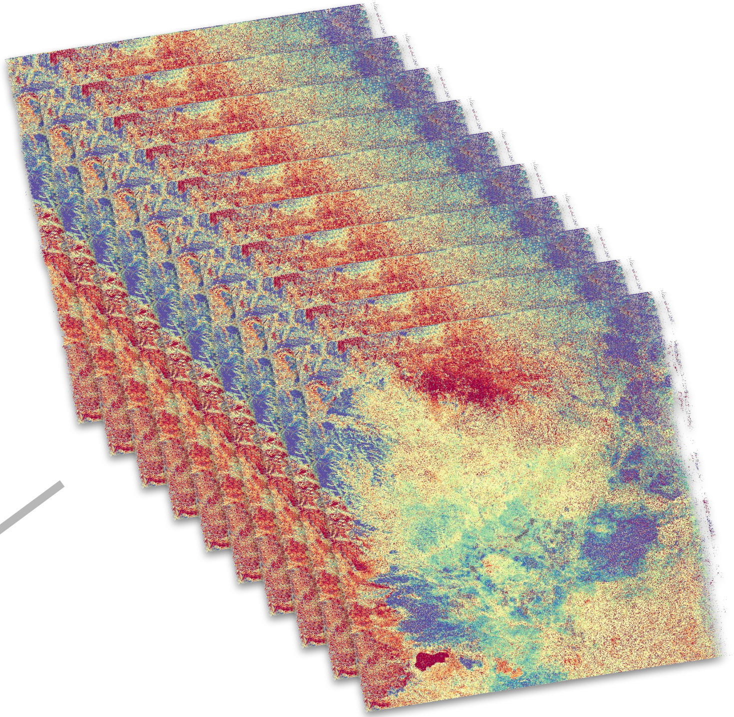


(n^2-n)/2

Interfere the Phases

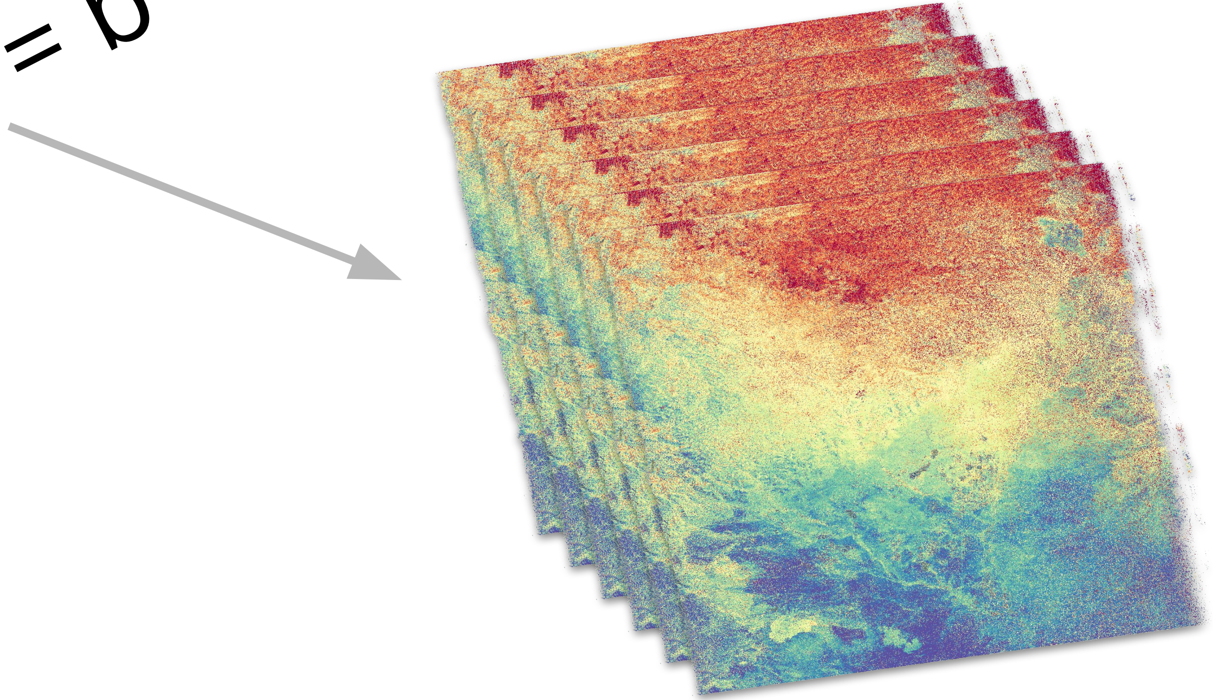
A matrix we can
easily calculate
ahead of time

$$A\ x = b$$

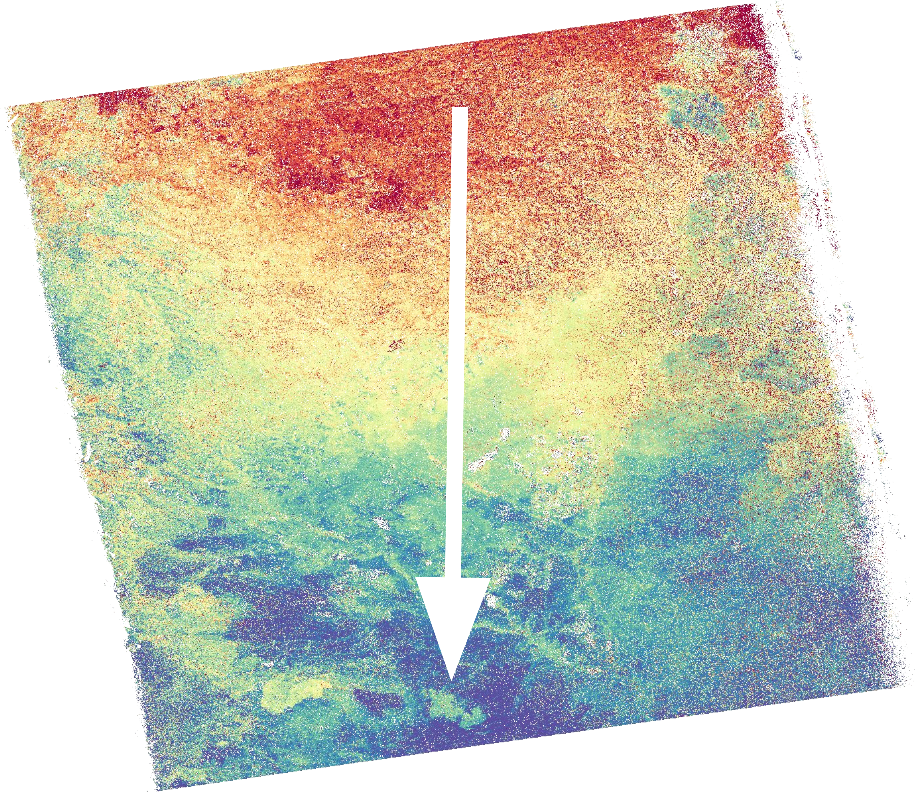


Model Timeseries from Interferograms

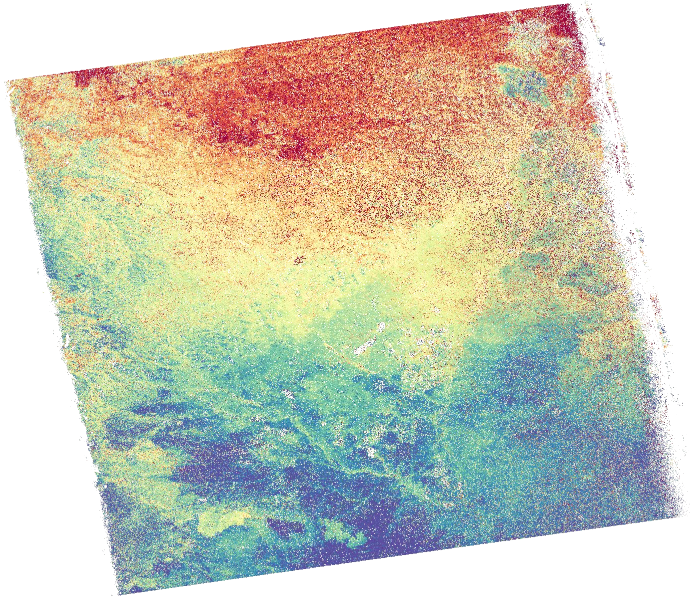
$$A x = b$$

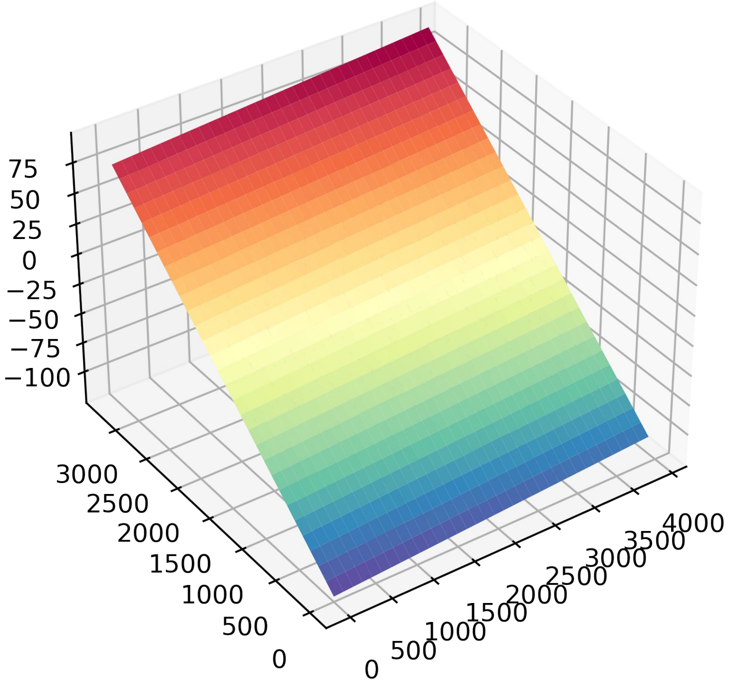


Model Timeseries from Interferograms

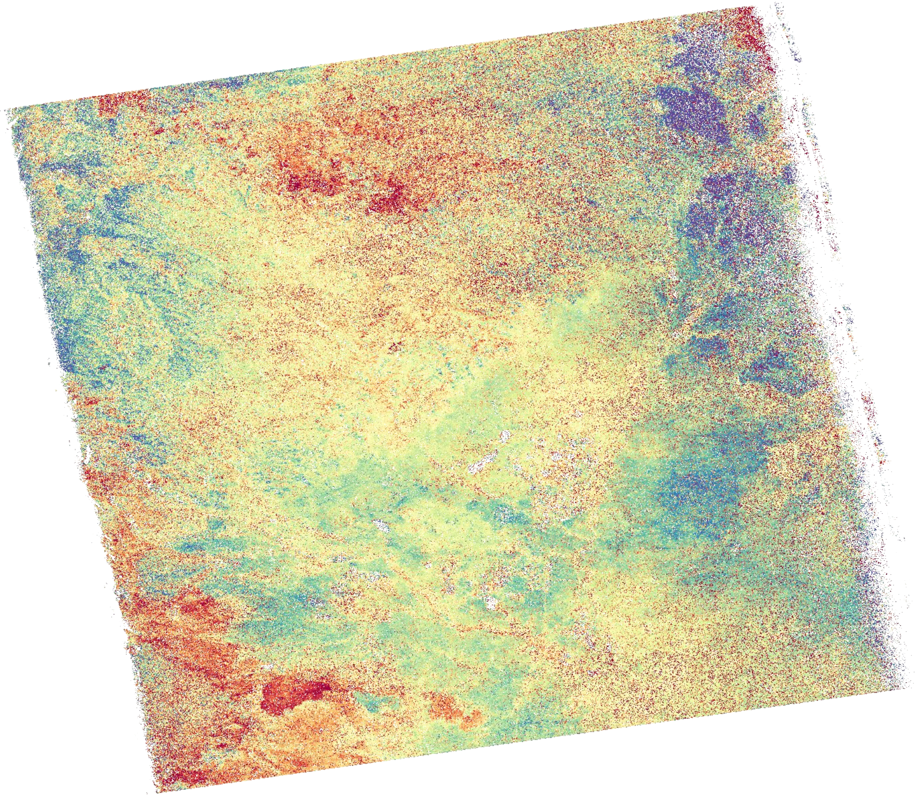


Remove Residual Trend





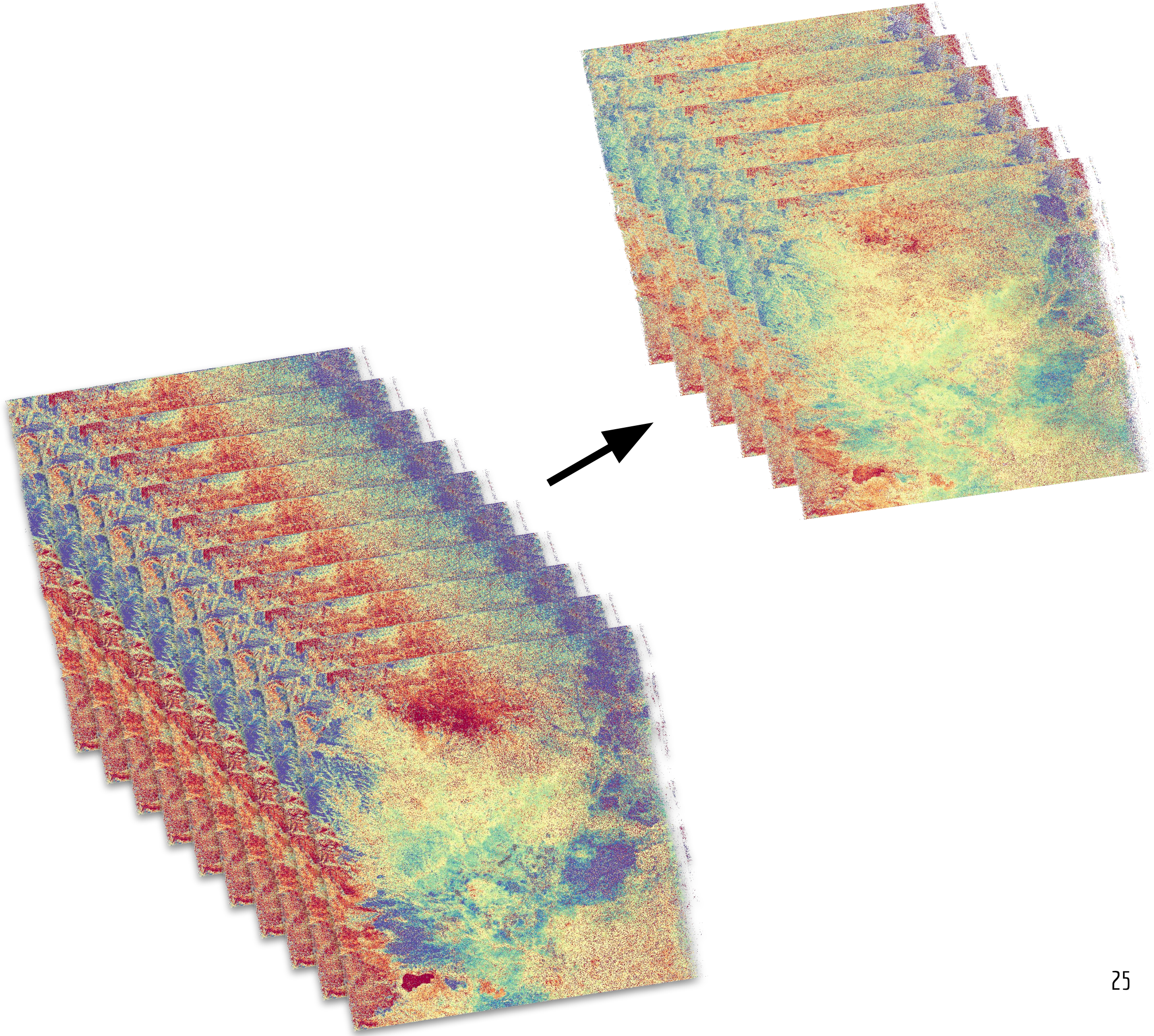
Remove Residual Trend



Final Result

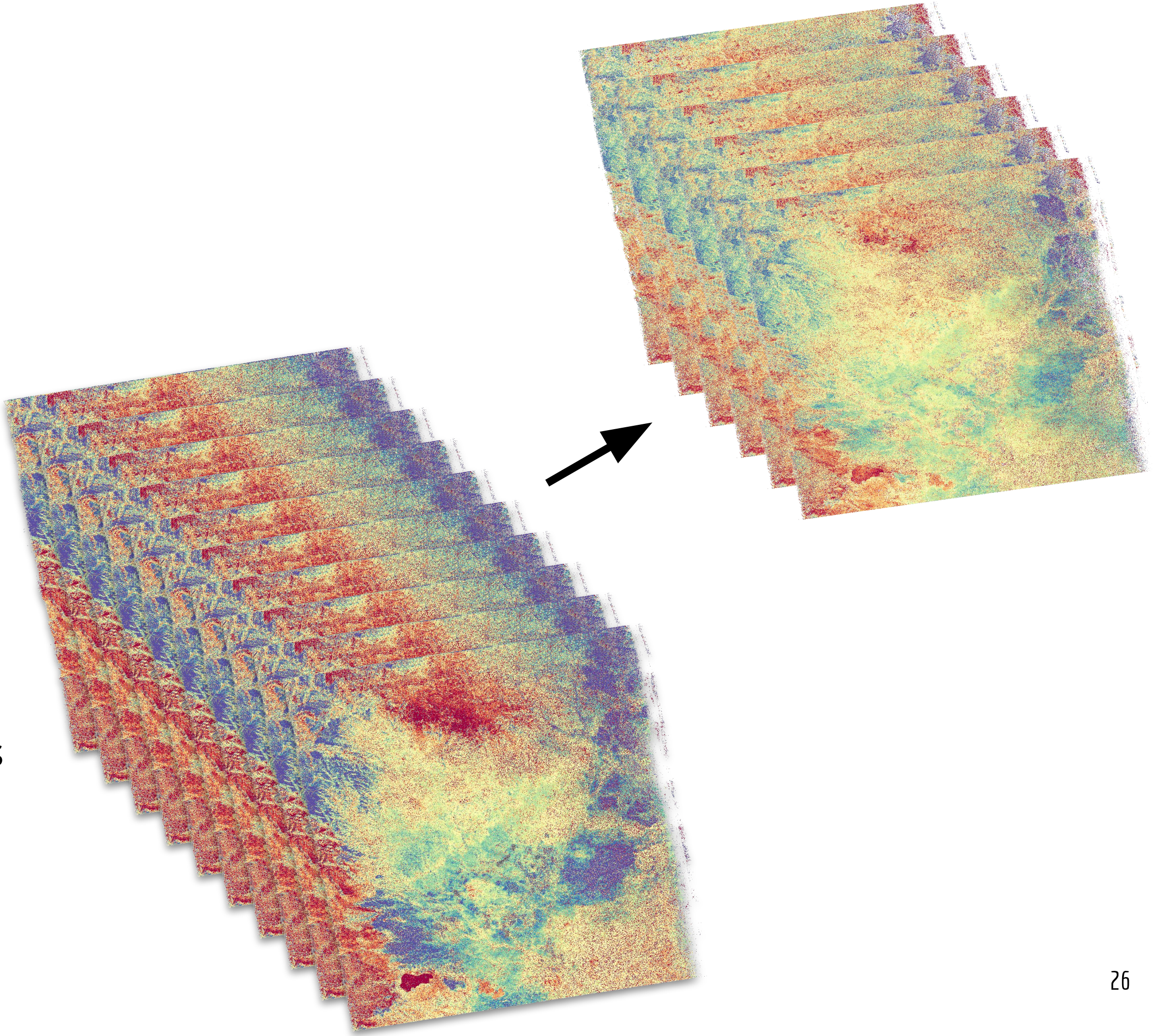
Summary

- Take data
- Align data
- Interfere data
- **Generate timeseries**
- **Detrend timeseries**
- Make pretty pictures



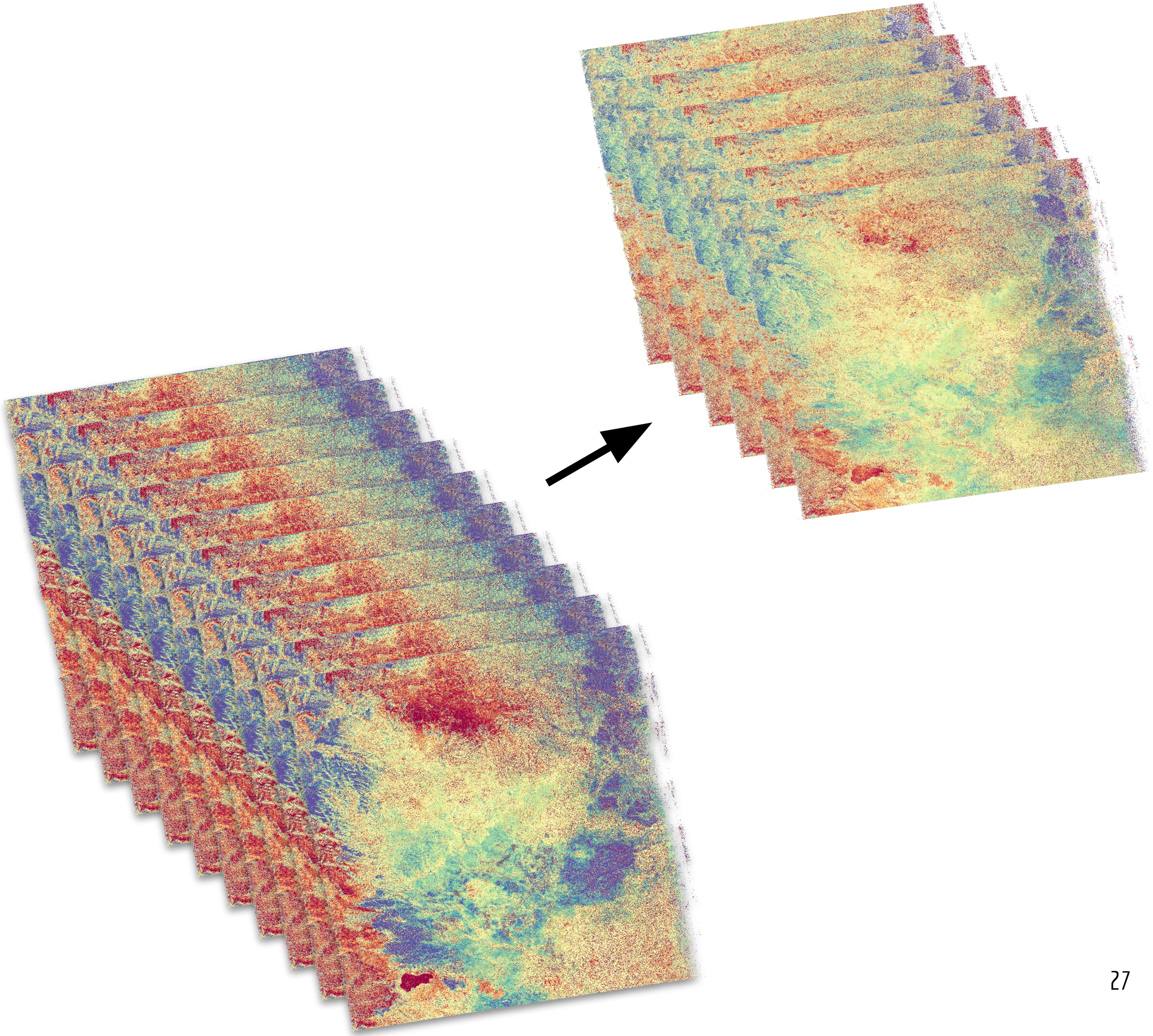
Speed

- Test data-set:
  - 80 interferograms
  - From 20 dates
  - (3250, 3900) Images
- More modern data is bigger, with many more interferograms, takes days-weeks to process
- Dr. Lindsey's code processes test data in about an hour



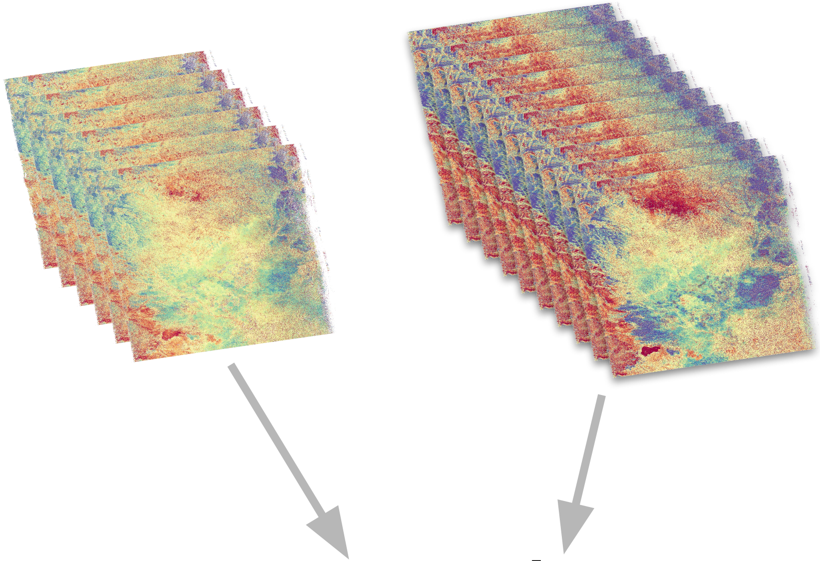
Speed

- **Our Bifrost pipeline does it in <2 minutes**
- **30x faster**



Speedups

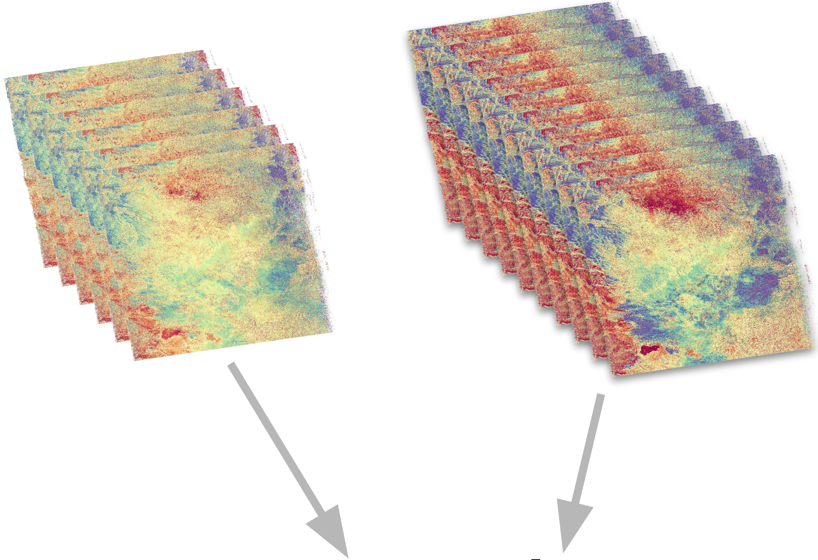
$$
\begin{bmatrix} 11 & 9 & 4 \\ 8 & 11 & 8 \\ 0 & 8 & 12 \\ 0 & 0 & 9 \\ 0 & 0 & 5 \end{bmatrix}
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}
=
\begin{bmatrix} 0.5 \\ 0.2 \\ \text{NaN} \\ 0.5 \\ 0.1 \end{bmatrix}
$$



A x = b

Math Improvements

$$\begin{bmatrix} 11 & 9 & 4 \\ 8 & 11 & 8 \\ 0 & 8 & 12 \\ 0 & 0 & 9 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.2 \\ \text{NaN} \\ 0.5 \\ 0.1 \end{bmatrix}$$



$$A\,x = b$$

Math Improvements

$$\begin{bmatrix} 11 & 9 & 4 \\ 8 & 11 & 8 \\ \cancel{0} & \cancel{8} & \cancel{12} \\ 0 & 0 & 9 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.2 \\ \cancel{\mathrm{NaN}} \\ 0.5 \\ 0.1 \end{bmatrix}$$

$$\begin{bmatrix} 11 & 9 & 4 \\ 8 & 11 & 8 \\ 0 & 8 & 12 \\ 0 & 0 & 9 \\ \cancel{0} & \cancel{0} & \cancel{5} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.3 \\ 0.1 \\ 0.4 \\ \cancel{\mathrm{NaN}} \end{bmatrix}$$

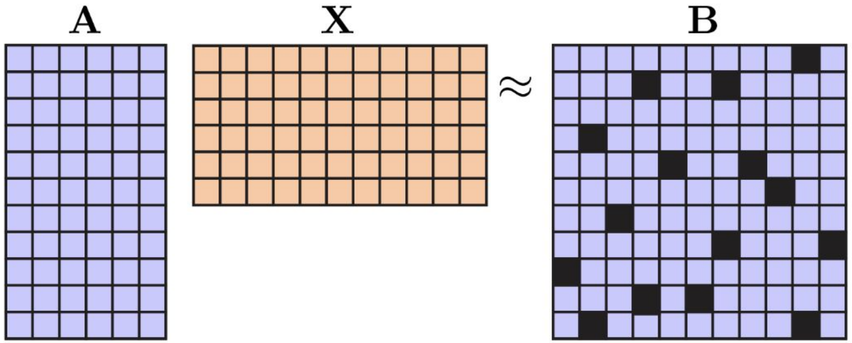
$$\begin{bmatrix} 11 & 9 & 4 \\ \cancel{8} & \cancel{11} & \cancel{8} \\ 0 & 8 & 12 \\ 0 & 0 & 9 \\ \cancel{0} & \cancel{0} & \cancel{5} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.3 \\ \cancel{\mathrm{NaN}} \\ 0.7 \\ 0.4 \\ \cancel{\mathrm{NaN}} \end{bmatrix}$$

Math Improvements

Images from Alex William's blog

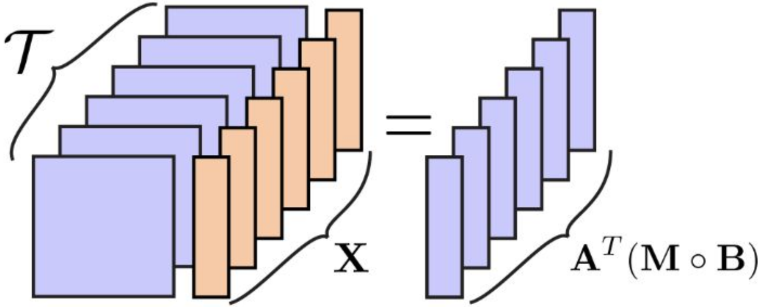
Two Birds, One Tensor

- Turn our pixel-by-pixel problem into a tensor solve over a collection of pixels
- Two benefits:
  - Much faster, matrix multiply then solve
  - Always uses same amount of memory
    - Important for GPU math



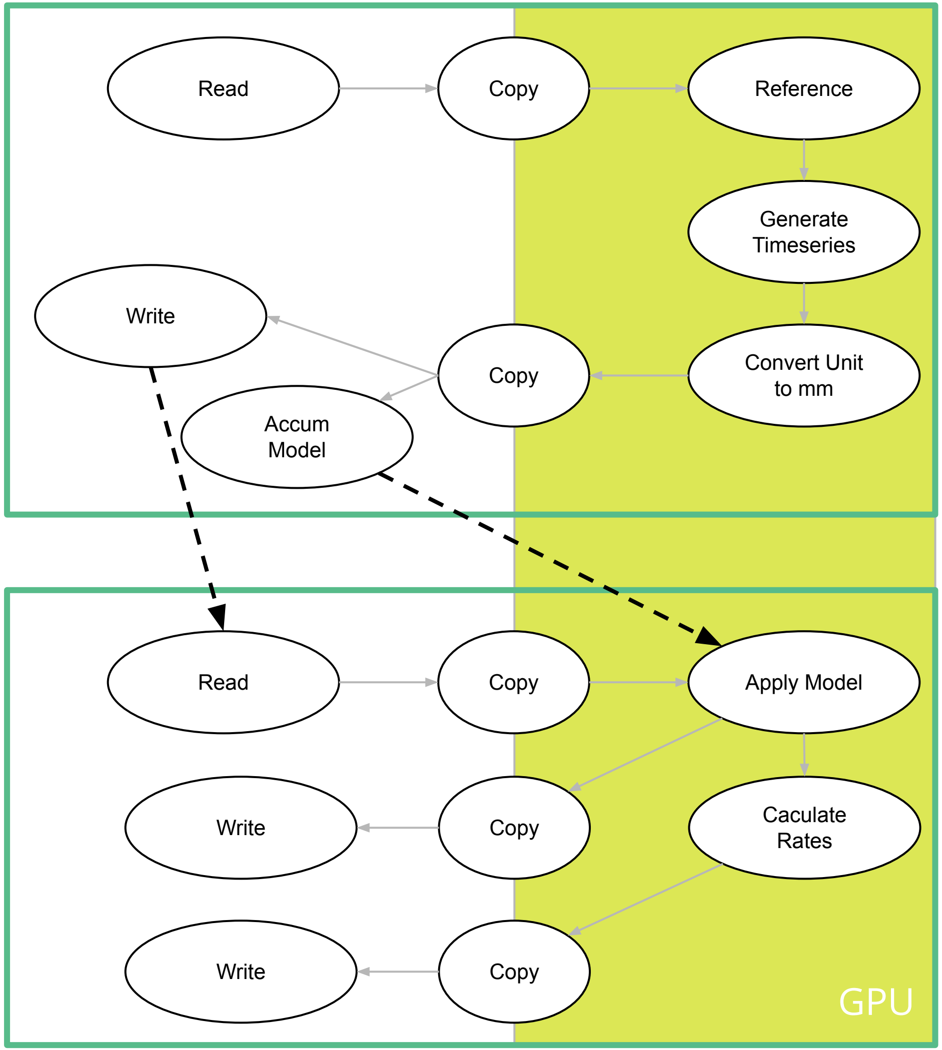
Full Equation

$$\mathbf{A}^T(\mathbf{M} \circ (\mathbf{A}\mathbf{X})) = \mathbf{A}^T(\mathbf{M} \circ \mathbf{B})$$

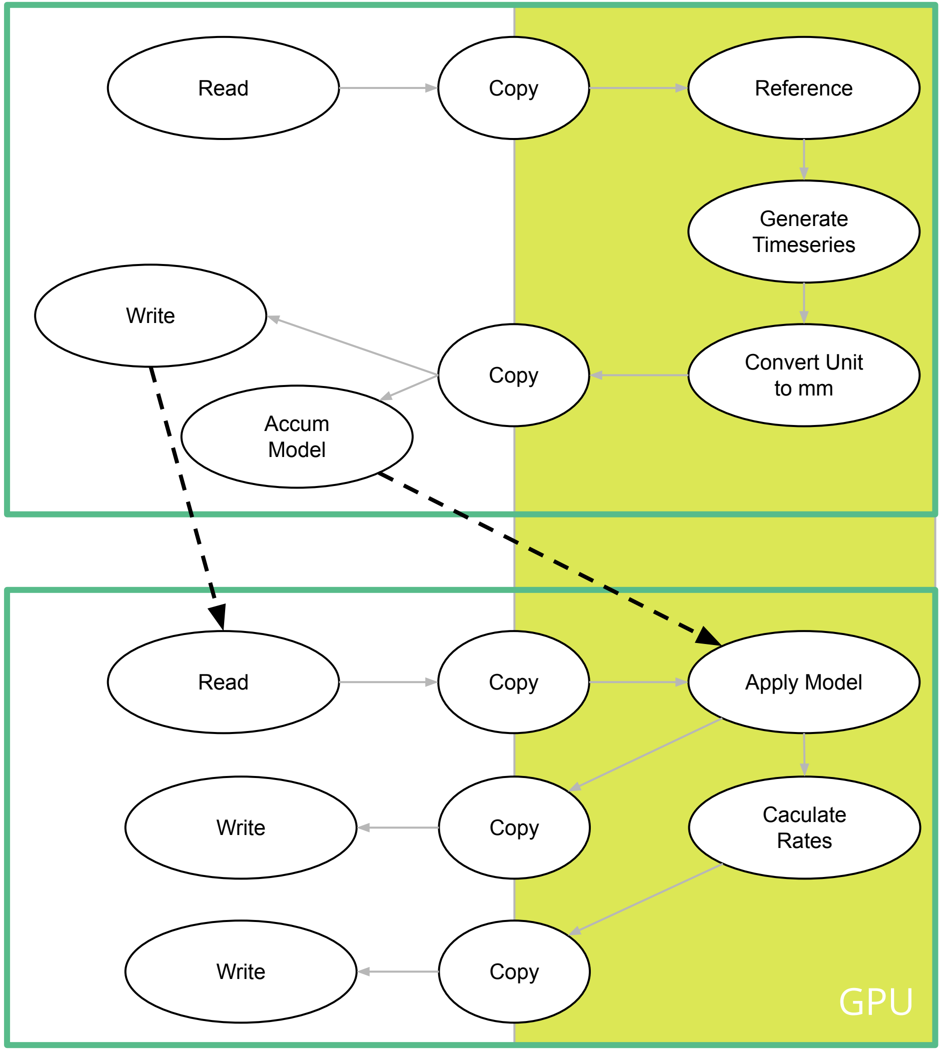


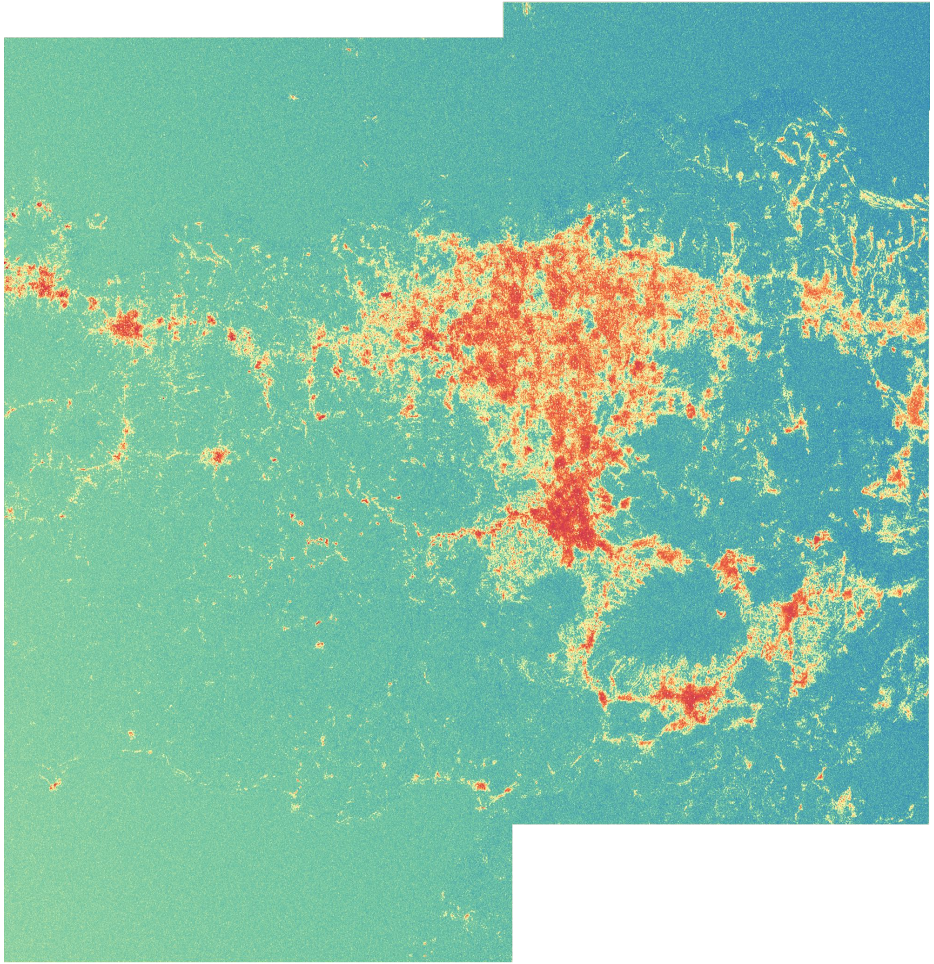
GPU Though

- Tensor trick (plus other small tricks) makes us about 3x faster
- Basically: if we ran our Bifrost pipeline only on CPU, 3x faster
- GPU gives us factor of ~10
- Total ~30x faster



```python
# Generate timeseries
with bf.get_default_pipeline() as PIPELINE1:
    # Read and copy to GPU
    b_read = bisblocks.ReadH5Block(infile, inname, args.gulp, space='system')
    b_read_gpu = bf.blocks.copy(b_read, space='cuda')

    # GPU math then copy back to host
    b_reff_gpu = bisblocks.ReferenceBlock(b_read_gpu, median_stack)
    b_tser_gpu = bisblocks.GenTimeseriesBlock(b_reff_gpu, dates, G)
    b_tsmm_gpu = bisblocks.ConvertToMillimetersBlock(b_tser_gpu, rad2mm_conv)
    b_tsmm = bf.blocks.copy(b_reff_gpu, space='cuda_host')

    # Write timeseries and accumulate model
    b_write = bisblocks.WriteH5Block(b_tsmm, outfile, outname, True)
    b_accum = bisblocks.AccumModelBlock(b_tsmm, trendparams=3)

    PIPELINE1.run()

    # Keep track of accumulated model
    GTG = b_accum.GTG
    GTd = b_accum.GTd

# Generate model from accumulated matrices and constraints
model = helpers.generate_model(outfile, gps, GTG, GTd, True, 3)

# Second pipeline
with bf.Pipeline() as PIPELINE2:
    # Read in data and copy to GPU
    b_read = bisblocks.ReadH5Block(outfile, outname, args.gulp, space='system')
    b_read_gpu = bf.blocks.copy(b_read, space='cuda')

    # Apply the model to the data, then write to disk
    b_amod_gpu = bisblocks.ApplyModelBlock(b_read_gpu, model)
    b_amod = bf.blocks.copy(b_amod_gpu, space='cuda_host')
    b_write2 = bisblocks.WriteH5Block(b_amod, outfile, detrendname)

    # Calculate average rates, then write rate image to disk
    b_rate_gpu = bisblocks.CalcRatesBlock(b_amod_gpu, t_axis)
    b_rate = bf.blocks.copy(b_rate_gpu, space='cuda_host')
    b_racc = bisblocks.WriteRatesBlock(b_rate, outfile, ratename)

    PIPELINE2.run()
```





Teaser


North Jakarta
Jakarta
Tangerang
Bekasi Regency
Depok
Bogor
BANTEN
Pandeglang
Mount Halimun Salak National Park
Pangrango
Garung
Plotot
105°47'33.5"E

Teaser

Summary

- Interferometric Synthetic Aperture Radar is powerful for ground deformation mapping, but modern processing is slowed by large datasets
- Bifrost offers a way to process datasets more quickly leveraging GPUs
- We see significant speedup on testing data
- Currently working on more modern data, expect to take a week of processing down to <1 day
- Other steps in InSAR process could benefit from **Bifrostification**

6. Questions?