Software for the SKA telescope

skatelescope.org, ska-sdp.org

Peter J Braam
SKA Science Data Processor
Cavendish Laboratory, Cambridge University
peter@braam.io

Oct 2018

Code: https://github.com/SKA-ScienceDataProcessor/RC

Reports and plan: will be published by Nov 18 on braam.io blog.

SKA International Design Consortia

Project Management and System Engineering Team based at JBO (UK)

~500 scientists & engineers in institutes & industry in 11 Member countries

Messages from this talk

1. What is the SKA telescope & what will it do?
2. Some information about its data processing
3. Design studies and prototypes for software.
4. Lessons learned

What is SKA – phase 1?

- Two big radio telescopes

- 100x sensitivity
- 1M times faster imaging of the sky

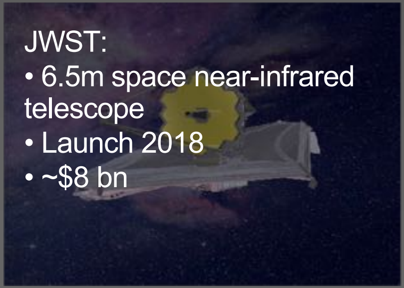
- Worldwide users of the data – like CERN

- SKA Phase 1 – in production 2025: focus of this presentation
- SKA Phase 2 – likely 10x more antennas – 2030's?

SKA – a partner to ALMA, EELT, JWST

Credit:A. Marinkovic/XCam/ALMA(ESO/NAOJ/NRAO)

Credit:ESO/L. Calçada (artists impression)



Credit: Northrop Grumman (artists impression)

Credit: SKA Organisation (artists impression)

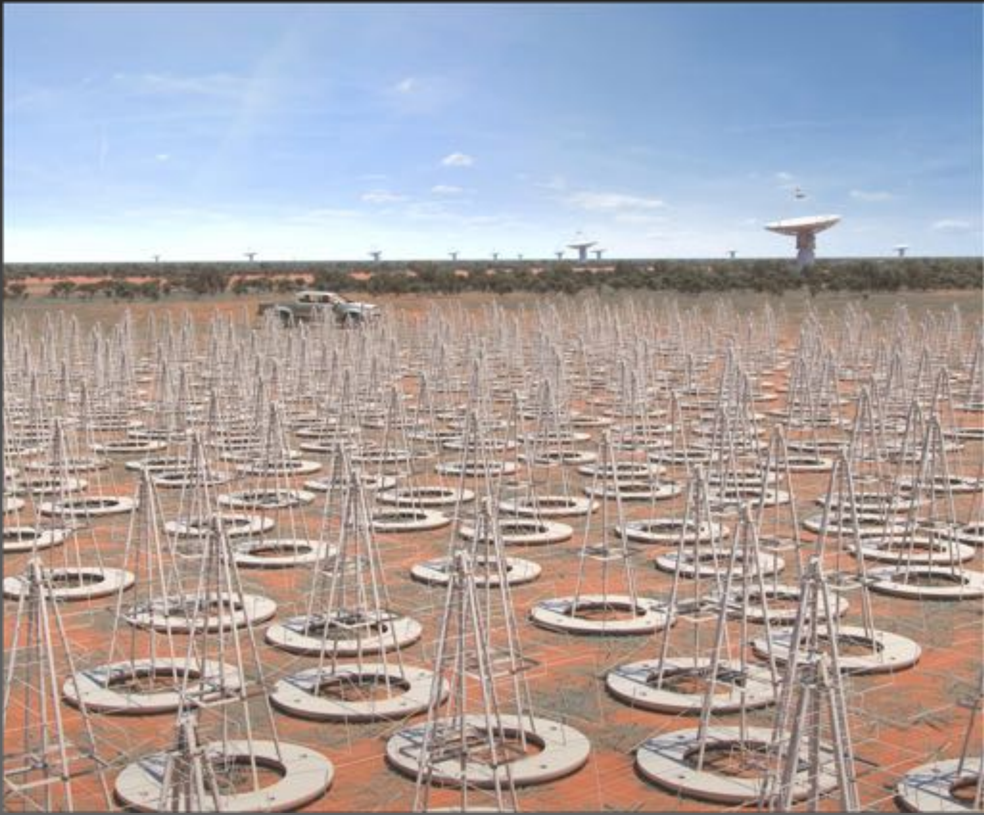
Low Frequency Aperture Array

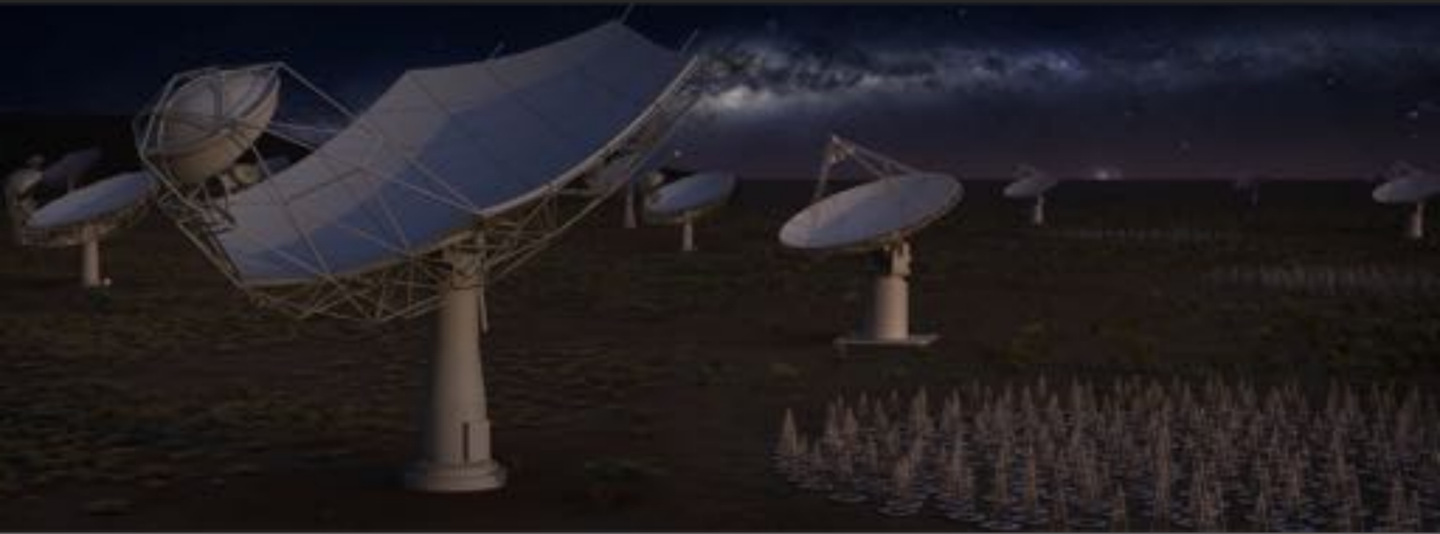
0.05 – 0.5 GHz

Australia

~1000 stations
256 antennas each
phased array with
beamformers

Murchison Desert
0.05 humans/km^2
Compute in Perth





Mid Frequency Telescope
500 MHz – 5GHz

South Africa

250 dishes with single receiver
Karoo Desert, SA - 3 humans / km^2
Compute in Cape Town (400 km)

Antenna array layout

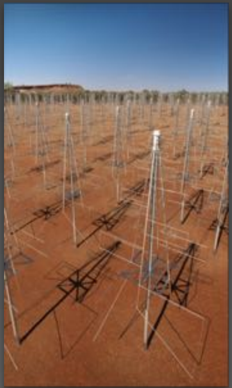
SKA1–MID, –LOW: Max Baseline = 156km, 65 km

SKA – data schematic

Antennas

Central Signal Processing (CSP)

Imaging (SDP) – HPC problem
2024: 100 PBytes/day
2030: 10,000 PBytes/day
Over 100's kms



Transfer antennas to CSP
2024: 20,000 PBytes/day
2030: 200,000 PBytes/day

Over 10's to 1000's kms

High Performance Computing Facility (HPC)

HPC Processing
2024: 300 PFlop
2030: 30 EFlop

In: 20 EB in -> out: 100 TB

Science

Science Headlines

Many key questions in theoretical physics relate to astrophysics
Rate of discoveries in the last 30 years is staggering

Fundamental Forces & Particles

Gravity

- Radio Pulsar Tests of General Relativity
- Gravitational Waves
- Dark Energy / Dark Matter

Magnetism

- Cosmic Magnetism

Origins

Galaxy & Universe

- Cosmic dawn
- First Galaxies
- Galaxy Assembly & Evolution

Stars Planets & Life

- Protoplanetary disks
- Bio-molecules
- SETI

skatelescope.org – two very large books (free!) with science research articles surrounding SKA

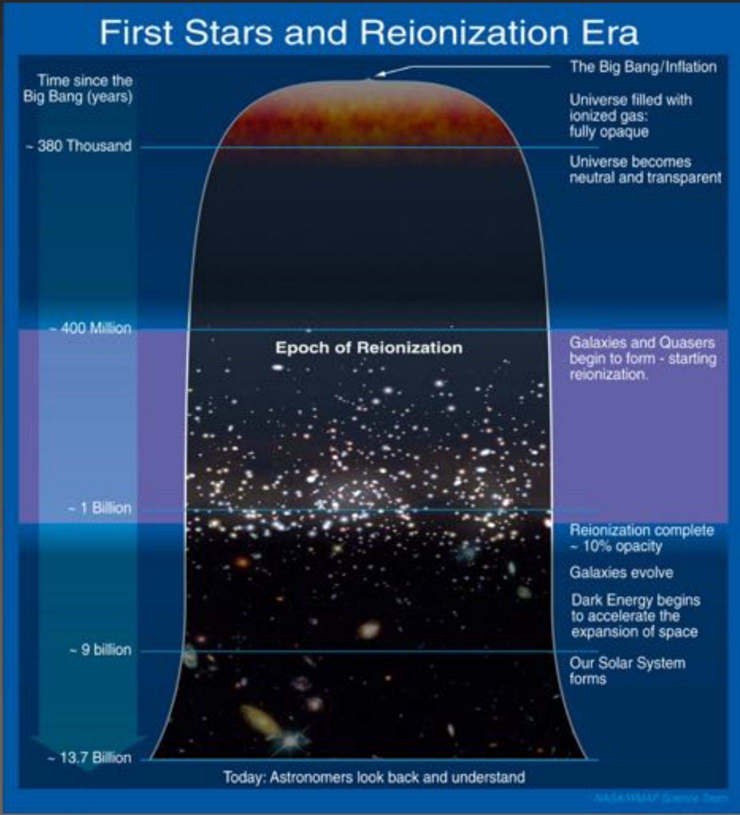
Epoch of Re-Ionisation

21 cm Hydrogen spectral line (HI)

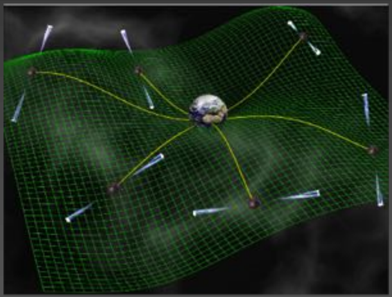
Difficult to detect

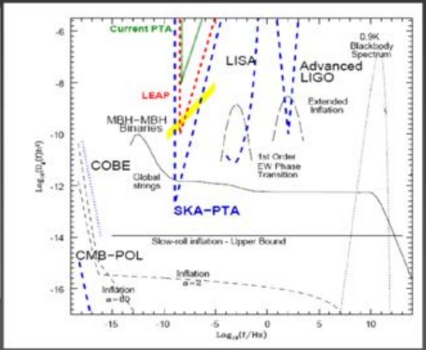
Tells us about the dark age:

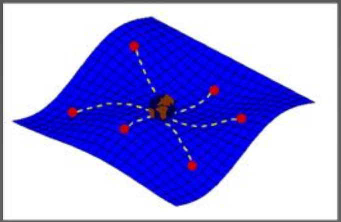
400K – 400M years (current age 13.5G year)



Pulsar Timing Array







What can be found:

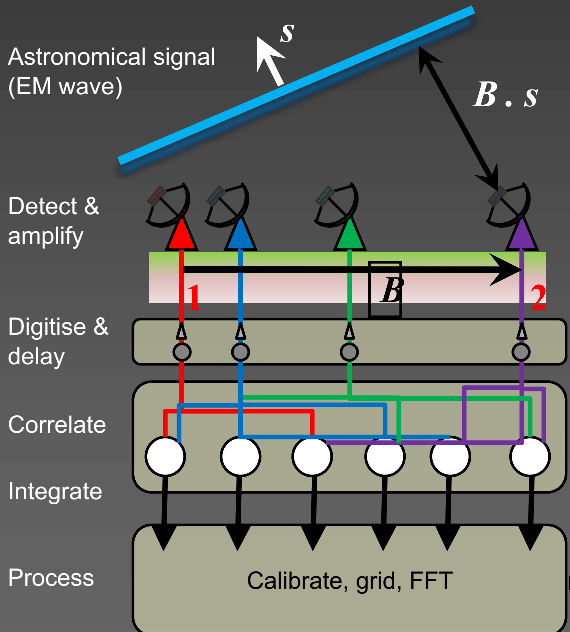
- gravitational waves
- Validate cosmic censorship
- Validate “no-hair” hypothesis

- **Nano-hertz frequency range**
- ms pulsars, fluctuations of 1 in 10^20

- SKA1 should see all pulsars (estimated ~30K) in our galaxy

Imaging Problem

Standard interferometer

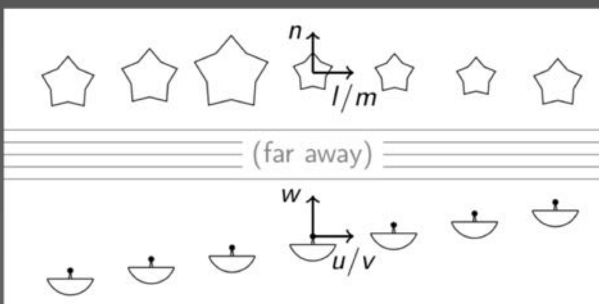


Visibility V(B): what is measured on baselines
Image I(s): image
Solve for I(s)

$V(\boldsymbol{B}) = \; E_1 E_2^* = I(\boldsymbol{s}) \exp(i\,\omega\,\boldsymbol{B}.\boldsymbol{s}/c)$ – image equation

Maximum baseline gives resolution: $\theta_{max} \sim \lambda / B_{max}$
Dish size determines Field of View (FoV): $\theta_{dish} \sim \lambda / D$

Interferometry radio telescope



Simplified

Sky is flat
Earth is flat

Visibilty to image is Fourier transform

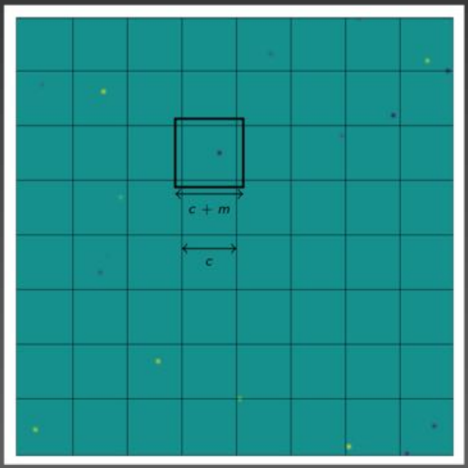
Actually

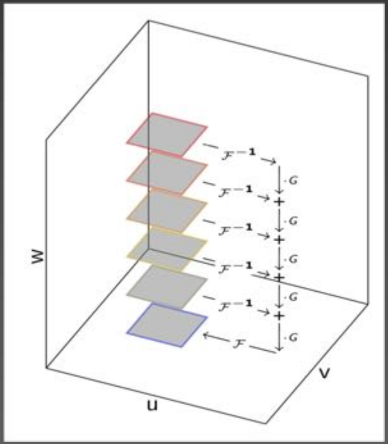
Sky is sphere, earth rotates, atmosphere distorts

Now it is a fairly difficult problem:

1. Non-linear phase
2. Direction, frequency, baseline dependent gain factor
3. Everything formulated with a lot of terminology and formulas

Reducing to 2D





Try to go back from 2D to 3D problem by relating (~100) different w values.
Domain specific optimization.

Grid size is 100K x 100K for 64K frequencies – problem is large
Full FFT is O(k log k), sparse FFT: O(#nonzero log #nonzero). SKA approach is close to this.

Computing in radio astronomy - 101

@Antennas: wave guides, clocks, beam-forming, digitizers

@Correlator (CSP central signal processing): == DSP for antenna data

- Delivers data *for every pair of antenna's (a "baseline")*
- Dramatically new scale for radio astronomy ~100K baselines
- Correlator averages and reduces data, delivers sample every 0.3 sec
- Data is delivered in frequency bands: ~64K bands
- 3 complex numbers delivered / band / 0.3 sec / baseline
- Do math: ~ 1 TB/sec **input** of so called *visibility data*

@Science Data Processor (SDP) – process correlator data

- Create images (6 hrs) & find transients (5 secs) – "science products"
- Adjust for atmospheric and instrument effects *calibration*

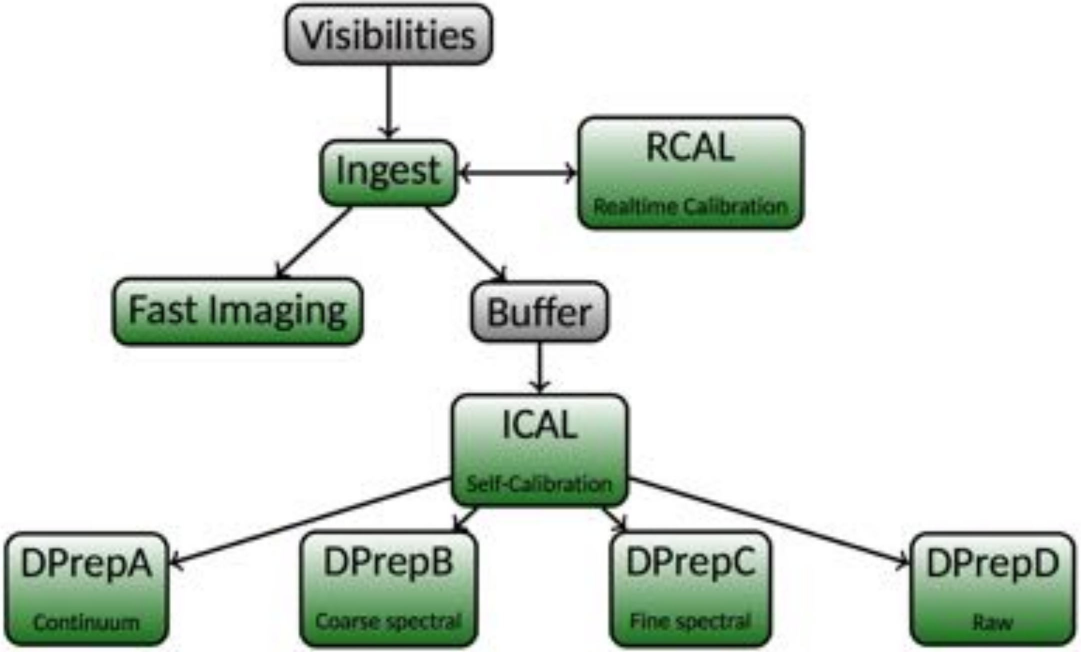
Outline of algorithm

About 5 different analysis on the data are envisaged:
e.g. spectral vs continuum (i.e. all frequencies) images.

Imaging pipelines:

- Iterate until convergence – approximately 10 times
- Compare with an already known model of the sky
  - Subtract everything known and bright, see new faint stuff

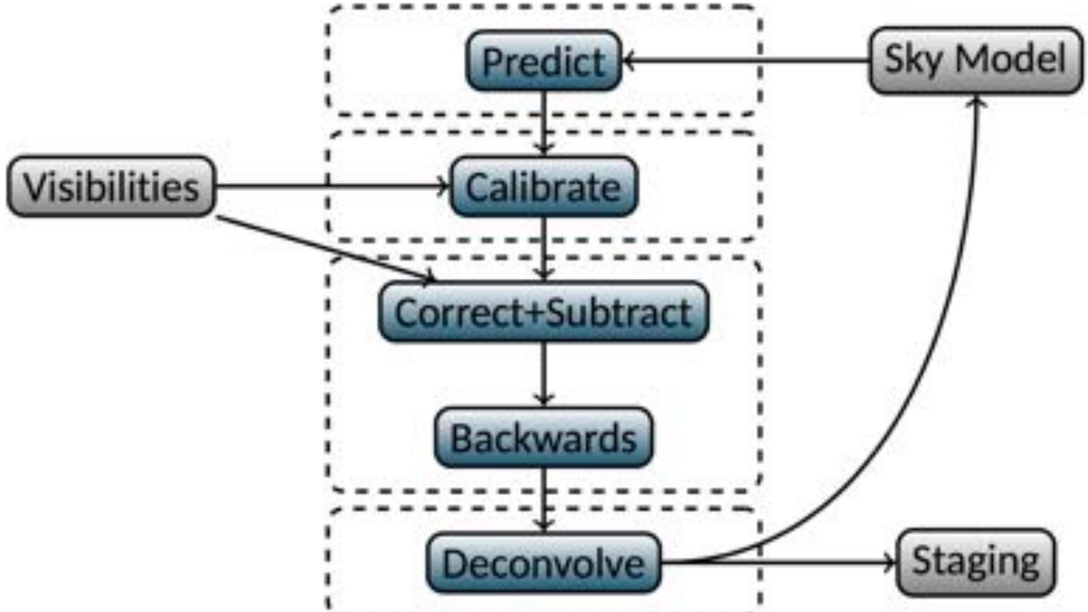
- Incorporates and recalculates calibration data



Follows architecture, allows running multiple data preparations.

Pipeline Internals

Rough structure and distribution pattern of most pipelines:



SDP specific Pipelines

Algorithmic similarities with other image processing

Each step is

- Convolution with some kind of a “filter” – e.g. “gridding”
- Fourier transform
- All-to-all for calibration

Why new & different software?

- Data is very distinct from other image processing
- Problem is very large – much bigger than RAM
- Reconstruction dependencies: sky model & calibration

Engineering Problem

Requirements & Tradeoffs

Turn telescope data into science products soft real time

1. Transient phenomena: time scale of ~10 seconds

2. Images: 1 image ~6 hours

Agility for software development

Telescope lifetime ~50 years

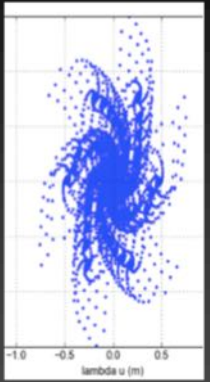
SDP computing hardware refresh ~5 years: portability

Use of large clusters is new in radio astronomy

New telescopes always need new algorithms

Initial 2025 computing system goal: make SKA #1

So – how difficult is this?



Data in the computation

Two principal data types

input is visibility – irregular, sparse uvw - grid of baselines

Image grid - regular grid in sky image

Different kinds of locality

Splitting the stream by frequency

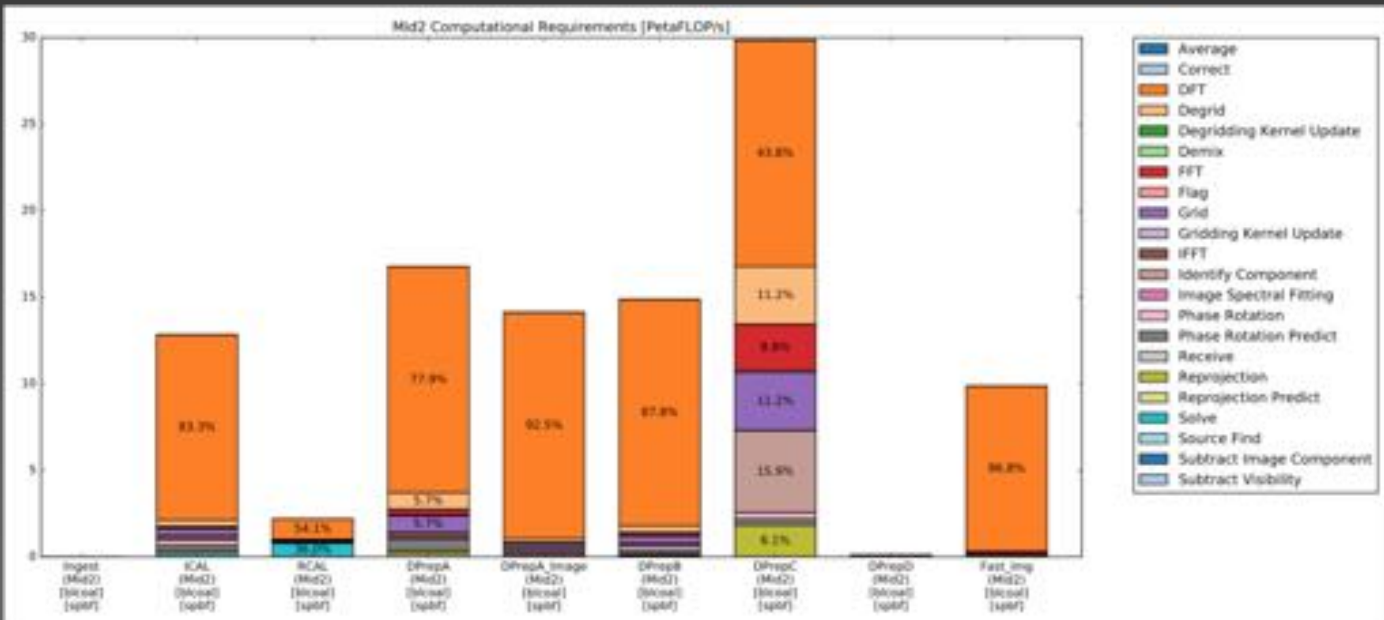
Tiling visibilities by region – visibility “tile” data highly irregular

Analyze visibility structure – 0, sparse, dense:
separate strategies

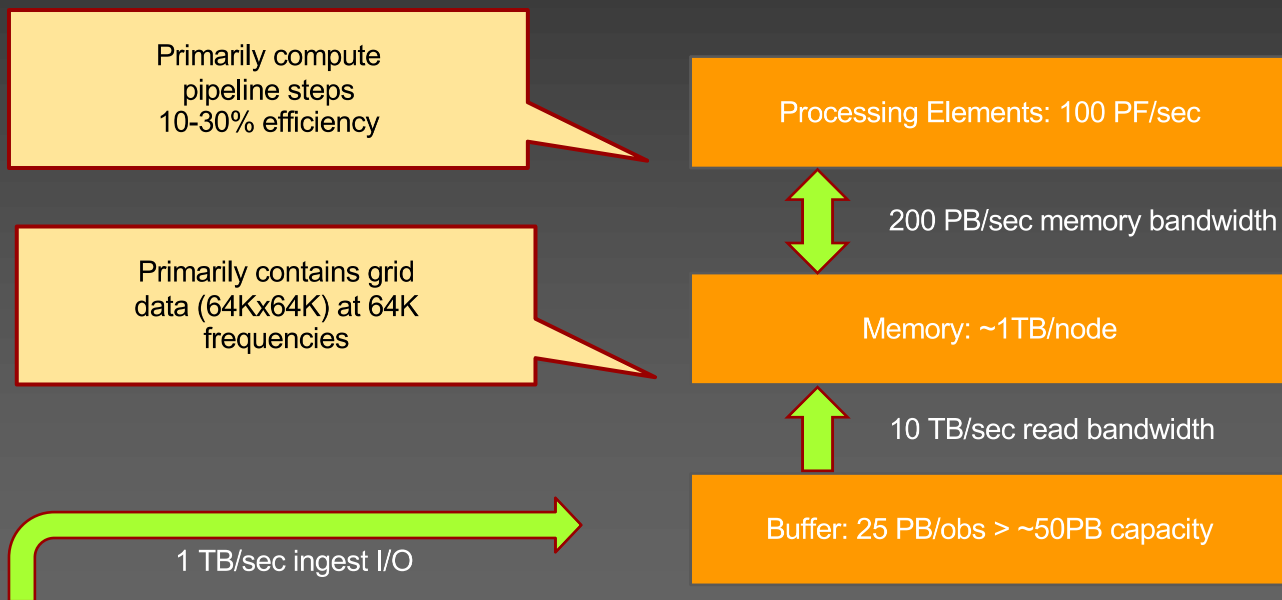
Remove 3rd dimension by *understanding* earth rotation

Data flow model with overlapping movement and computation

Relative kernel cost



Data Movement



SDP “performance engineering” approach

Conservative - this is not computing research

Known-good algorithms, hardware

Perhaps deep math question remains: is problem really O(#antennas^2)?

Parametric model of the computation – BIG SUCCESS of design

Detailed FLOPs, memory use, data movement, energy

Key outcome: 100 PF/sec & move **200 PB/sec** from HBM to CPU

@50 PJ / byte this is ~10MW power

Software

Reference Libraries with Algorithms

Address scalability issues

Software Framework for SKA SDP

- Critical design review is in progress
- Creating software is a very high risk part of the project
- Cluster with some 1000’s of computers is required

- Outcome:
  - Build/use a system like OpenStack with **scheduler**
  - Talk with telescope control system
  - Be flexible about pipeline execution – any framework should work

Imaging 101

2-D examples of visibility functions

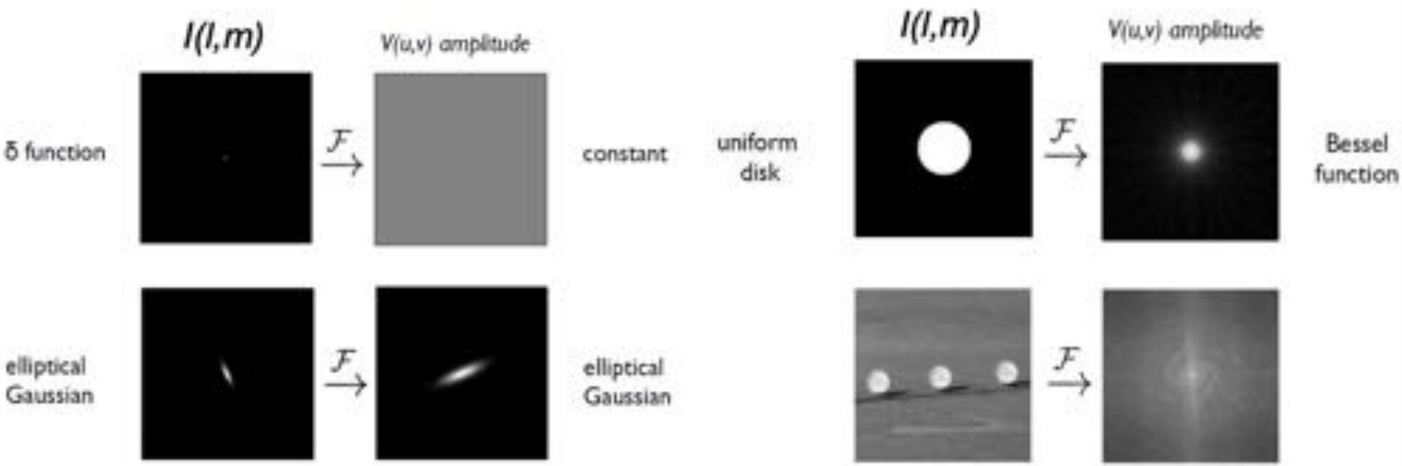
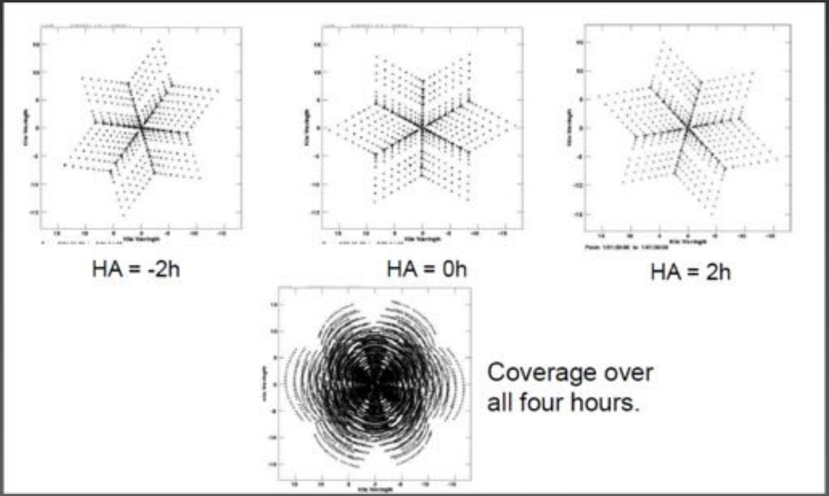
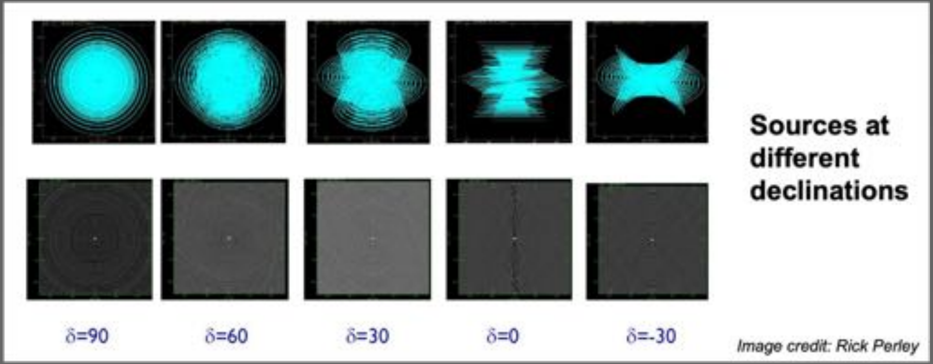


Image credit: David Wilner

Aperture Synthesis – planet rotation





Next steps are …

- Weighting, calibration
- Subtracting bright spots
- Image correction and bright spots

- Telescope will see an airport radar 50 light years away
  - Extreme uncertainty if the image is correct
  - Airplane flies through field of few
  - Cell phone is turned on
  - Satellite flies by
  - Sunshine deforms antennas

SKA phase 1 design starts 2013

- 2013-2016

- Intel and nVidia – explore kernels, e.g. DFT
  - Quickly delivered kernels with performance aligned with hardware

- I had a small team (3 Haskell programmers)
  - I was given a free reign, everything was a greenfield
  - Explore frameworks for distributed execution
  - Deal with resource management for data
    - E.g. schedule for (known) variable execution time and data sizes
  - Address integration of kernels
  - Create DSL for programming

SKA phase 1 design starts 2013

- 2015-2018

- Many key decisions
  - “You don’t get fired over buying IBM”
- Early decisions were:
  - Python
  - Use “cloud” framework – should have good industrial support
  - Haskell effort was called “Lunatic Fringe” ☹
  - Not cutting edge software development
  - Academic institutions decided they would lead development, not outsource it

Haskell project

Milestone 1: Cloud Haskell

- Startup with cluster scheduler SLURM
  - 1 scalability fix: remove all to all communication
- Remote process spawning awfully slow – not fixed
- **Remote functions** – unmanageable “RemoteTable”, “mkclosure”
  - Solved by SPJ + Boespflug – still not landed or documented
- **Fast networking Infiniband with RDMA**
  - Multiplex many input + output streams on 1 network stream
  - Protocol stacking, buffer reservations, error handling
  - Facundo Dominguez and I build Haskell binding to CCI (from DOE)
  - Haskell binding was very complex: 2x slower and as big as CCI

- This phase finished quite upbeat – but already a pile of issues!
- I learned in Shonan: stream processing is far enough along that H-CCI would be a beautiful simpler package than FFI solution

Visibilities & Baselines distribution

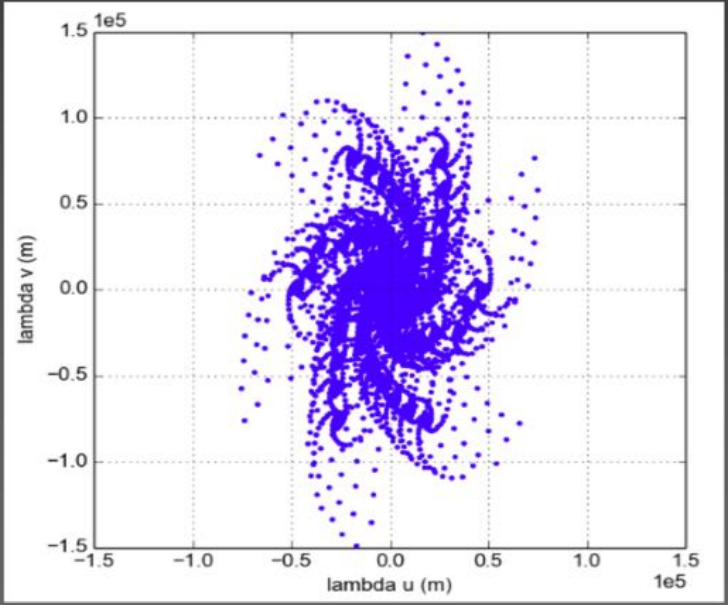
Each pair of telescopes has a baseline

Baselines rotate as time progresses

Each baseline has associated visibility data ("sample")

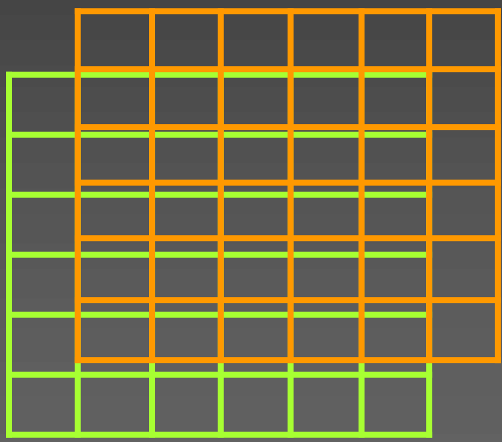
Baselines are sparse & not regular, but totally predictable

The physical data structure strongly enables and constrains concurrency & parallelism



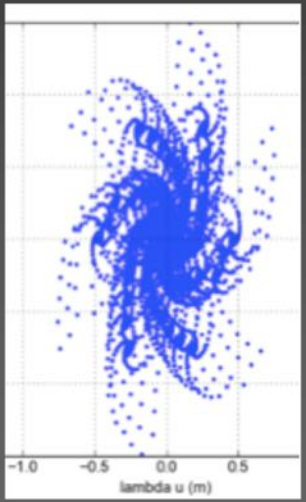
Simulated data from 250 SKA1-MID dishes

Visibility gridding & cache re-use



Time rotation of
UV grid.

Only fetch edges
Re-use core



Milestone 2: a "gridding" pipeline

- Gridding: put visibility data into a regular grid (for FFT later)
  - Convolution and oversampling, get in touch with the baseline spirals
  - complicated convolution kernel, convolution – borrow from C
  - Accelerate: difficulty with irregular visibility data – slow
  - Halide (Stanford Google "image" DSL) – cannot do sparse arrays
    - Halide runs (perhaps) in every Android camera pipeline
    - Data format conversion between C & Haskell
  - Haskell for kernels looking grim – dropped kernels from project
- A requirement was to create a DSL
  - Requirements for the DSL were incomplete
  - DSL ***should have*** been specified by SKA team
  - Also some success - tree reductions, flood fill trees etc. 1 line expression
- We drowned into awful "applied math / physics formulas"
  - Conal Elliott could have saved us, but that wasn't planned

Simple Pipeline

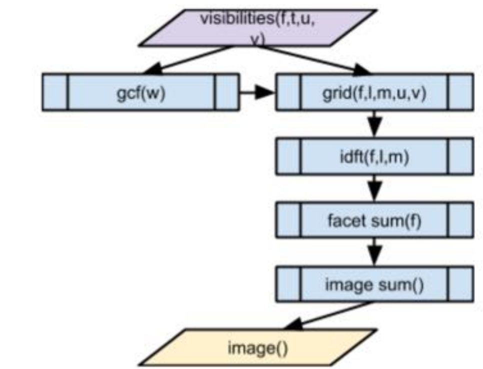


Figure 2: "Imager" compound data flow

Minor cycle:
Imager – make image from visibilities

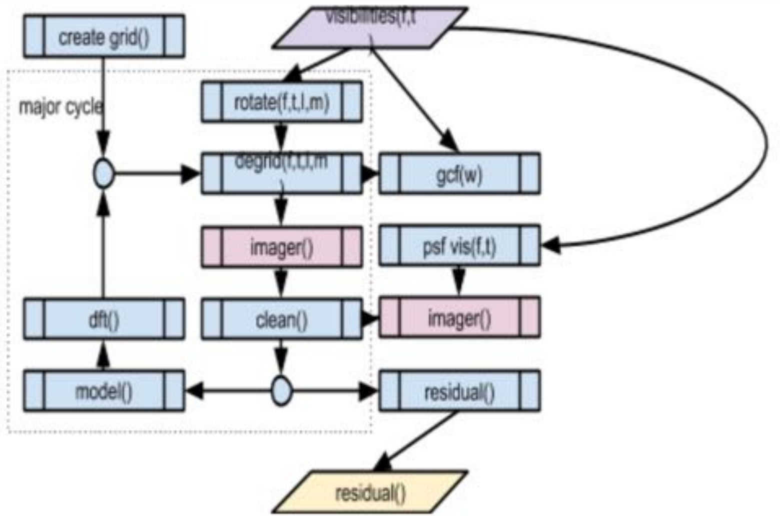
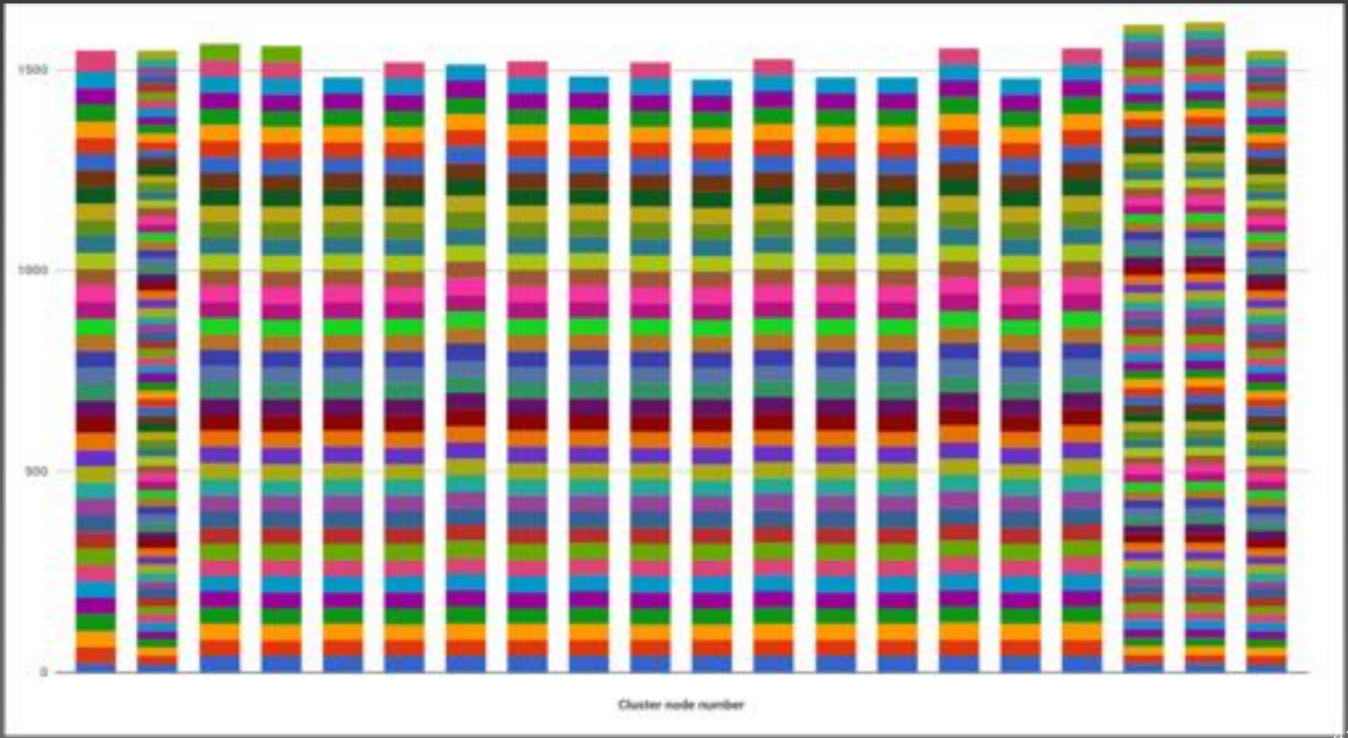


Figure 3: Continuum Pipeline Data Flow

Major Cycle Subtract model, repeat for convergence

Load balancing scheduler



Milestone 3-5: a "full" pipeline

- Continued towards a full imaging pipeline
  - Handled node failures in Cloud Haskell
  - Did not yet create a re-execution protocol
    - Learned here: stream error handling or clocked networks probably enable this easily
    - Fairly complex protocol is required to asynchronously restart a computation, on a subset of nodes, when there is a failure (HPC normally uses a checkpoint)
- Constructed a load balancing scheduler
  - Different frequency channels create 10x spread in compute time
  - Build a plan – all nodes could finish at the same time

- Manuel Chakravarty looked into the DSL's
  - Quickly pointed out what I suspected but couldn't give proper direction "it can indeed be much simpler". We also reinvented Pipes ….

Requirements

Some 60

Messages can be sent in parallel to many child actors with waiting for multiple and blocking only to prevent overflows - if this is arranged by the runtime, its effects shall be clear to the programmer
Messages can be sent in serially to multiple child actors without overflow, with and without waiting for responses - if this is arranged by the runtime, its effects shall be clear to the programmer
Imaging pipelines including those with loops can be expressed nearly mechanically
Through language mechanisms data flow graphs can be forced to have a restricted structure, such as fork join graphs, to keep programs easier to understand
Tree reductions can be implemented conveniently
Actors can have types
When overhead of invoking separate actors exceeds the benefits the runtime system or language can combine the actors.
Run trial computations using a list of strategies for data partitioning and parallelization. Consider profiles and select algorithm to run on leaf nodes
Run computations for collections of input data, analyze profiles, create a load distribution over islands

Clocks?

Fusion

Milestone 5-7: compare 3rd party solns

- Scientific Simulation efforts have created a ~6 bigger initiatives for similar use:
  - **Legion** (Aiken's team @Stanford) – data flow graph is call graph of an imperative program. Nodes are actors. Schedule actors and memory allocations. Have a few big scale users.
  - Produced 60 core requirements, selected 15 sample programs to demonstrate, less than 10 made it
- Others
  - Halide, Parsec (ORNL), Swift/T (Chicago), xStar (nVidia)
  - Many we couldn't even compile (we were not beginners …)
- 2018 – progress in Tensorflow may be very promising
  - SKA perhaps prefers a simple solution they can modify

Haskell issues

- Syntax and programming alien to most applied scientists
- Not ready for new hardware: fast networks, hybrid memory
- HPC basic libraries not available: need NumHa
- Scalability sometimes ignored
- Few real life examples (e.g. of failure recovery)
- Networking performance has to be “line rate”

Imho
– a few M$ would have gone a long way
– community is amazingly helpful but strapped for time

Domain specific problems

- Problem very poorly documented for non-experts
  - Formulas and all kinds of “field knowledge”
  - Quote: “if you haven’t gone around a radio telescope with an oscilloscope to understand every signal, you can’t do this”

- Very troubled history of these software systems – it has always gone wrong in the past

Project specific

- Cannot fail – too much money involved
- Everyone wants that money, particularly research institutions
- Needs to catch buzz – “cloud”, “ML” ….

- I should have built different relationship
  - require commitment

- Several things were tried by others, but decision has been to leave the pipeline implementation for later
  - A library of kernels has been written in Python
  - Full integrated picture was not repeated

Conclusions

Future outlook

- Information encoding with correlators is likely sub optimal (cf. theory of rough paths) – fundamental change of complexity?
- Bandwidth – Google's TPU with systolic array gets us close
  - TPU v3: we would "only" need 10K nodes
- Variable precision number formats – up to 8x smaller
  - Could save 2-5x in bandwidth, storage, memory capacity
  - Best results require a chip change
  - Limited precision formats error estimation – lost art from 1960's
- More automatic transitioning from a mathematical model to numerical models
- CERN in similarly messy calculations is having success with replacing "algorithms" with "learning"

Thank you.

questions?

skatelescope.org

peter@braam.io