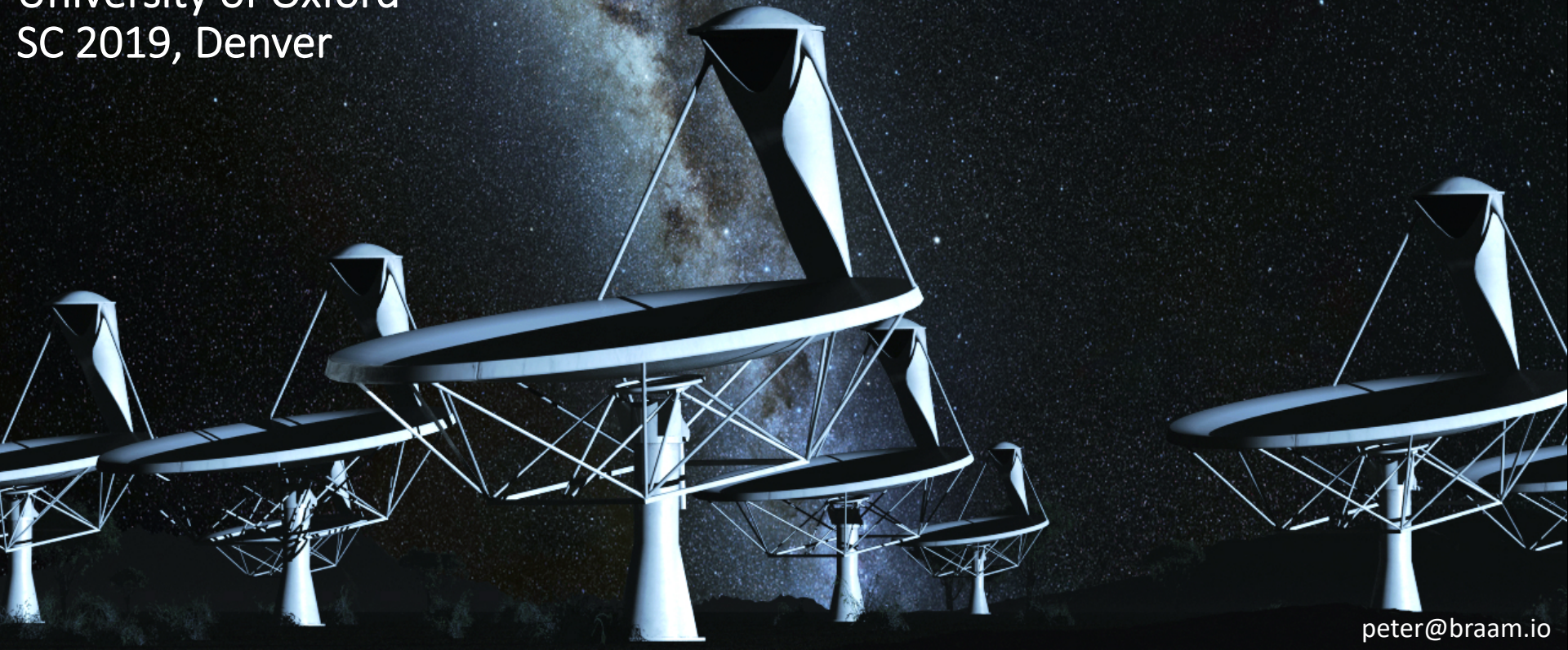
The SKA telescope and lessons for CS

Peter Braam
University of Oxford
SC 2019, Denver



peter@braam.io

me

math physics @oxford (gauge theory, CFT)

distributed systems cs @cmu

@6 startups & jobs @3 acquirers - Lustre

SKA @cambridge

@oxford

work with 100's of largest compute centers and many major system & CPU/GPU vendors

Content

- A grand challenge computing problem for the SKA radio telescope

- Discuss this problem in relation to a PL approaches: specification & DSL's

MPI is not mentioned one single time – and rarely has been in the context of this problem.

Purpose of the talk is to imprint the importance of precision of specification and generality in domain specific software design

Grand Challenges in HPC

- Grand Challenge
  - Often demands 10x-100x more than available from “industry”
- Central themes are:
  - Modifiability of the software
  - Throughput + Latency -for-
  - Compute, Memory, Network, Storage (all heterogeneous)
  - Software is “mission critical”, research secondary
- Presently many new opportunities:
  - Software, number formats, memory hierarchy, accelerator chip design

SKA telescope

Square Kilometer Array – refers to a square kilometer of antenna surfaces

Square Kilometer Array (SKA) radio telescope

- By far the largest designed radio telescope to date (antenna surface > 1 km^2)
- ~10 country collaboration in SKA Treaty Organization (HQ in Jodrell Bank, UK)
- CERN like model

- Two telescopes: in deserts Africa & Australia
- Data processing: electronics (“DSP”) on site, super computers in Cape Town and Perth

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

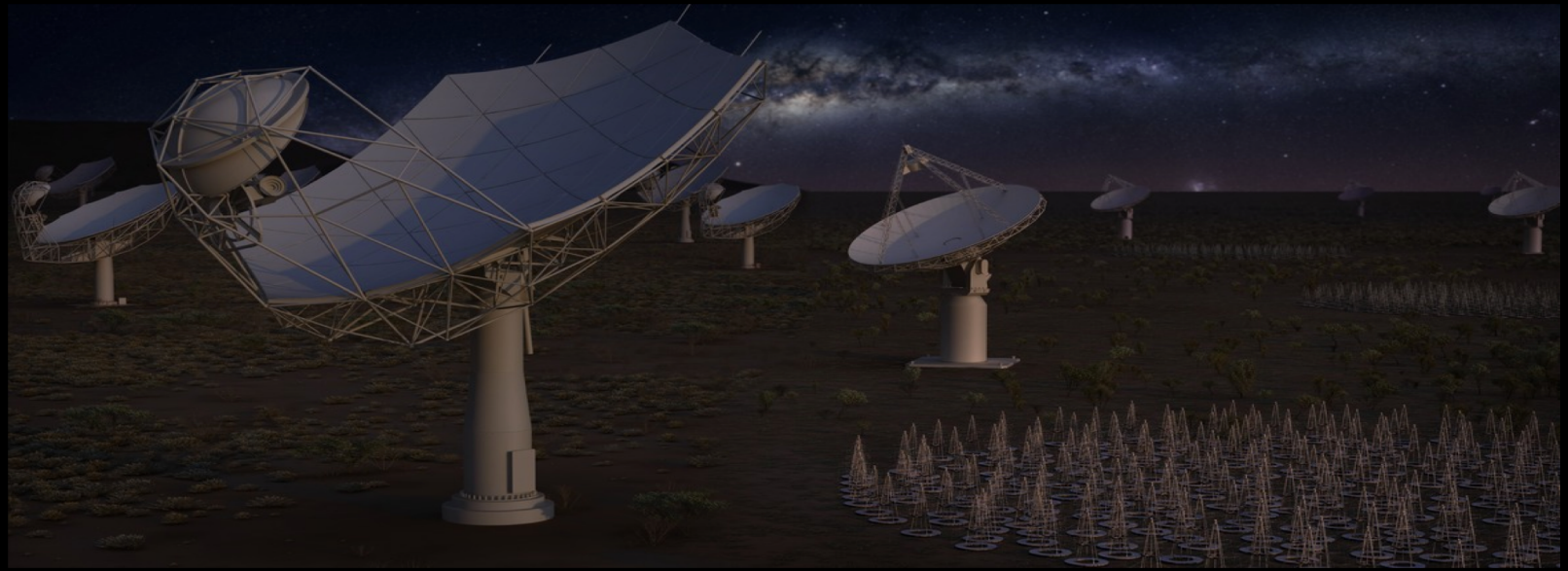
- Worldwide use of the data for research:
- ▪ SKA Phase 1 – in production 2027
  ▪ SKA Phase 2 – likely 10x more antennas – 2030’s?

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)

SKA pre-cursor Meerkat already in operation has 64 antennas

Astrophysics Aims for SKA

Science Headlines

Fundamental Forces & Particles

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

Origins

- Galaxy & Universe
  - Cosmic dawn
  - First Galaxies
  - Galaxy Assembly & Evolution
- Stars Planets & Life
  - Protoplanetary disks
  - Bio-molecules
  - SETI

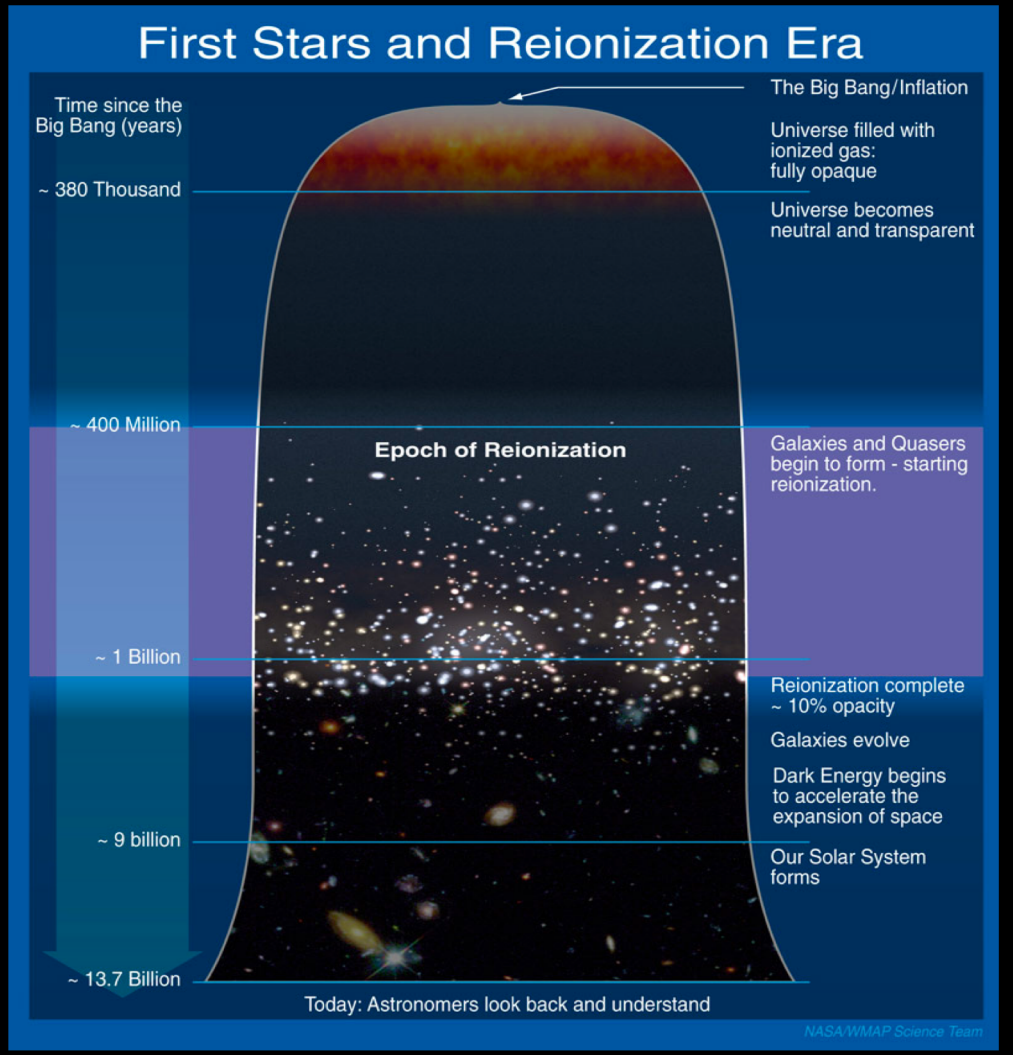
Many key questions in physics maybe answered through astrophysics

Rate of discoveries in the last 30 years is staggering

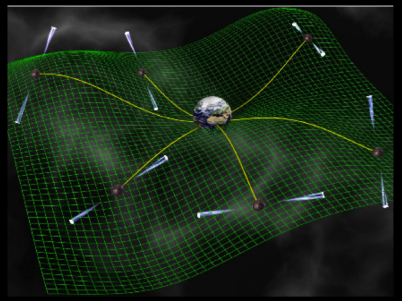
skatelescope.org – two very large books (free!) with science research articles surrounding SKA

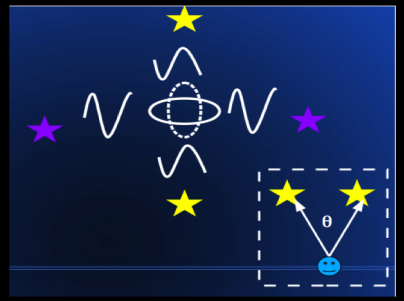
Epoch of Re-Ionisation

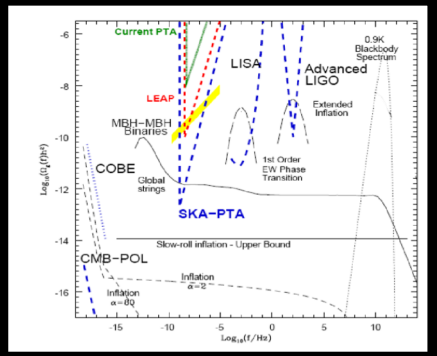
- 21 cm Hydrogen spectral line (HI)
- Difficult to detect
- Tells us about the dark age:
- 400K – 400M years (age now 13.5G year)



Pulsar Timing Array



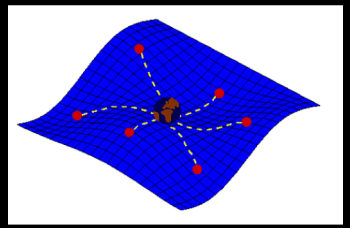




What can be found:
- gravitational waves
- Validate cosmic censorship
- Validate “no-hair” hypothesis

- Some gravitational waves have nano-hertz frequency range
- ms pulsars, fluctuations of 1 in 10^20

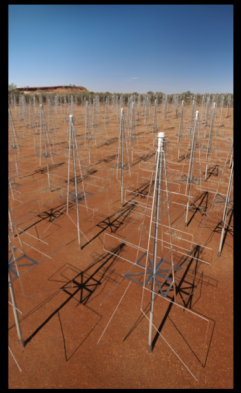
- SKA1 should see all pulsars (estimated ~30K) in our galaxy



SKA Data Processing

SKA dataflow schematic

Antennas



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

Over 10's to 1000's kms

Central Signal Processing (CSP)

Transfer CSP to SDP:
2024: 100 PBytes/day (1TB/sec)
2030: 10,000 PBytes/day
Over 100's kms

Imaging (SDP) – HPC problem

High Performance Computing Facility (HPC)
HPC Processing
2024: 300 PFlop
2030: 30 EFlop

Science Data Archive

Archive – in/near HPC center
World wide replication
100 TB/day
1 EB capacity for 5 years

In: 20 EB in -> out: 100 TB

peter@braam.io

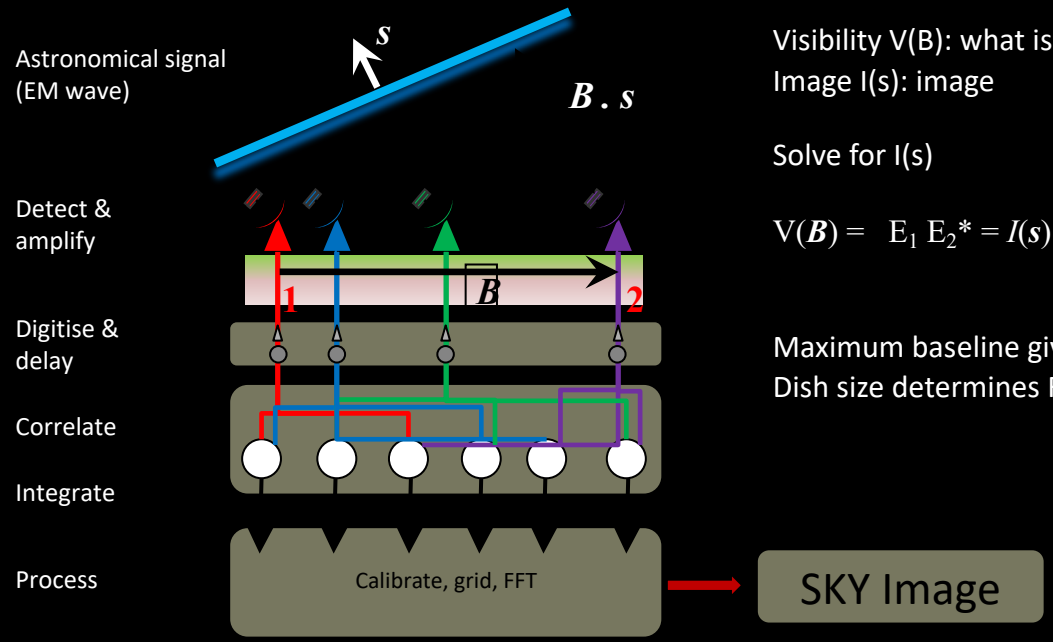
Requirements & Tradeoffs

- 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
  How difficult is this
  Be conservative – it's not software research, it can't fail

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): $\theta_{dish} \sim \lambda / D$

dataflow

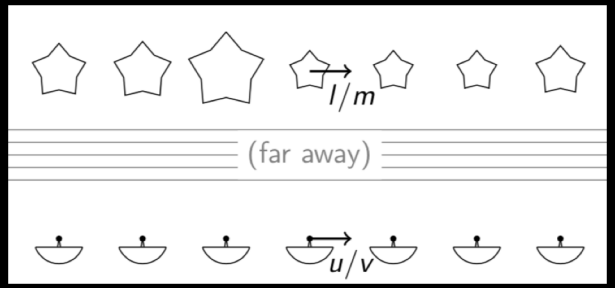
Computing in radio astronomy - 101

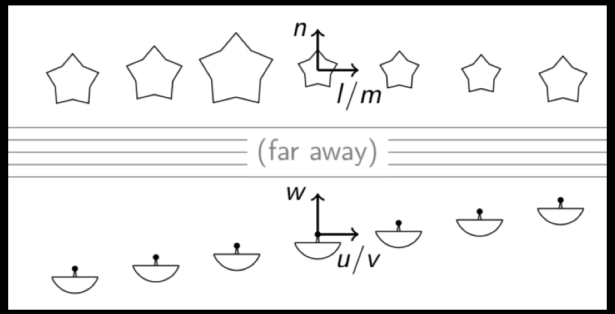
- @Antennas: wave guides, clocks, beam-forming, digitizers

- @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

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

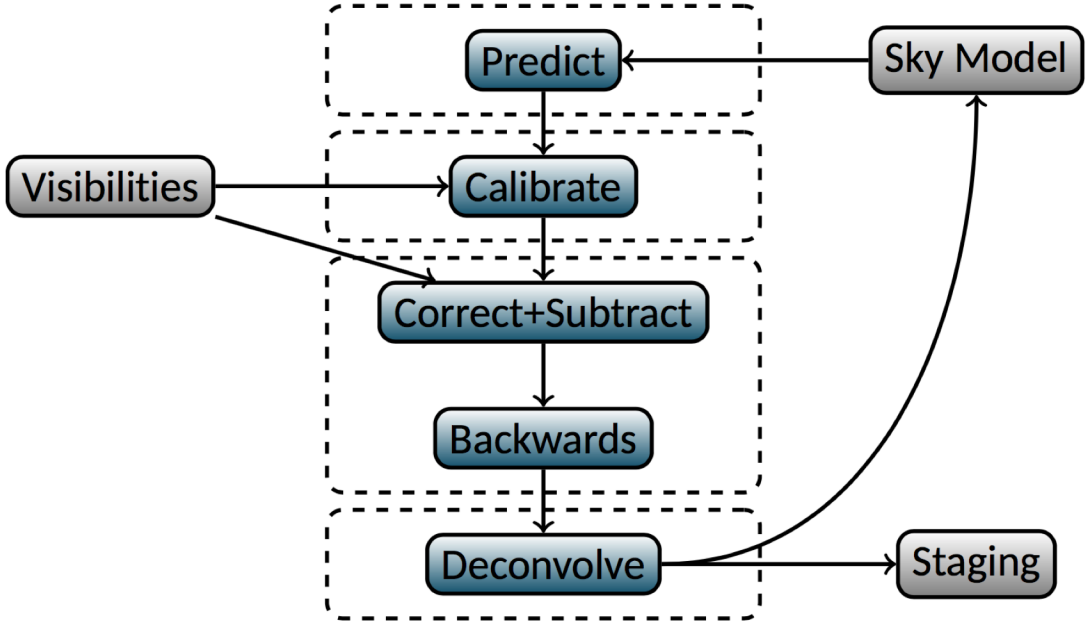
Outline of imaging algorithm

- About 5-10 different analysis on the data are envisaged:
- e.g. spectral vs continuum (i.e. all frequencies) images.

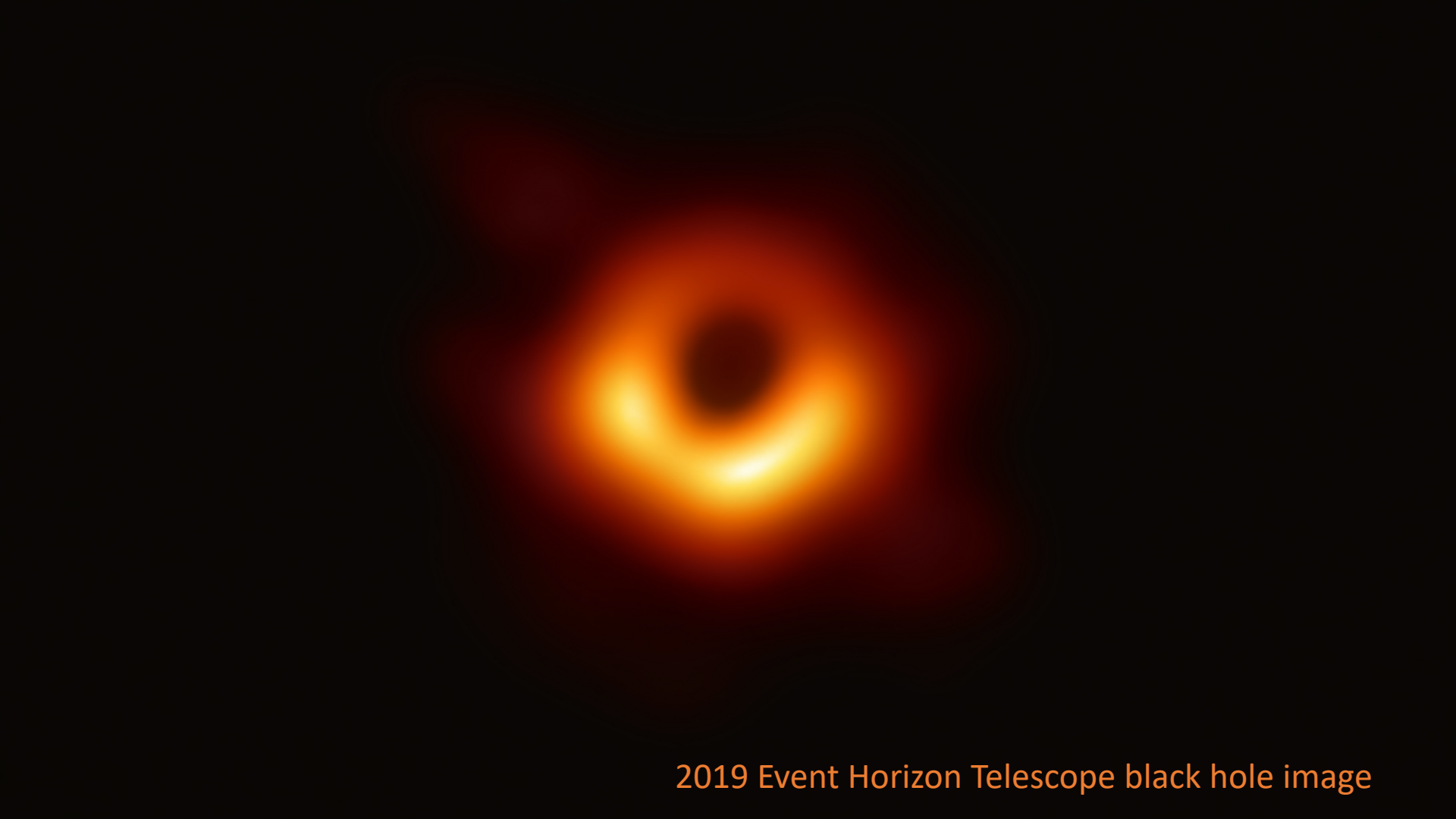
- Imaging pipelines:
  - Iterate until convergence – approximately 10 times
  - Compare with an already known **model of the sky (models require much scientific input)**
  - Subtract everything known and bright, see new faint stuff

- Steps in the pipelines
  - Standard elements: Fourier transformations, convolutions, subtractions etc.
  - Unusual stuff: adjust inputs (part of calibration), remove junk (satellites, airplanes), calibrate and adjust model, irregular data, very complex convolution kernels, very large image grids (10^5 x 10^5 x 64K freq)

Rough structure and distribution pattern of most pipelines:

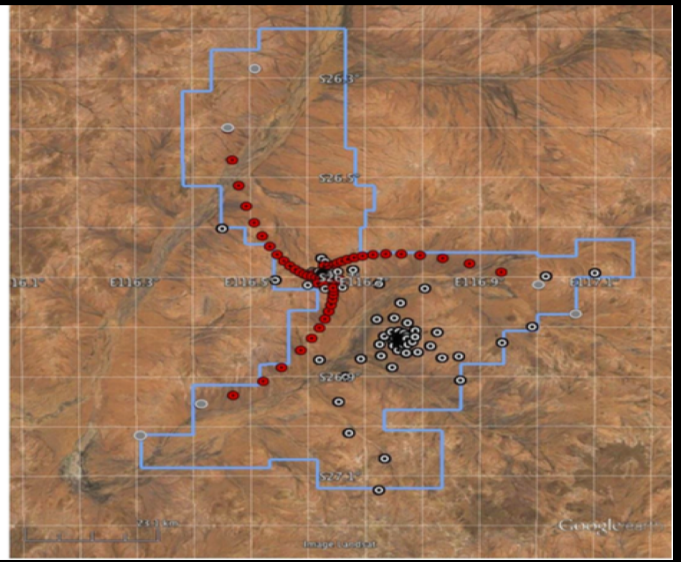


Meerkat image of galactic center (cf. Ian Heywood, Oxford)



2019 Event Horizon Telescope black hole image

Antenna array layout

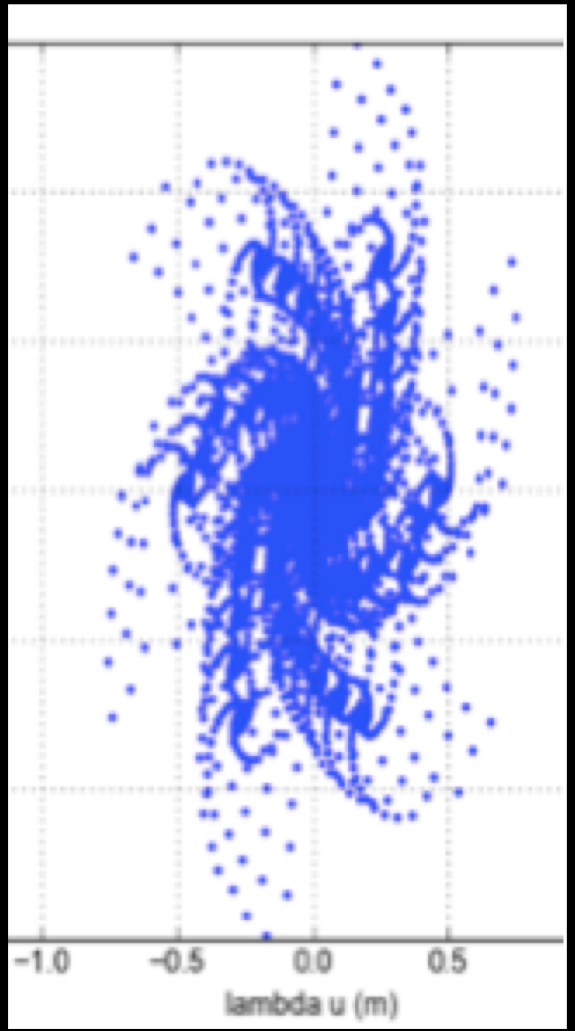


SKA1-LOW Max Baseline = 65 km

SKA1–MID Max Baseline = 156km

Data in the computation

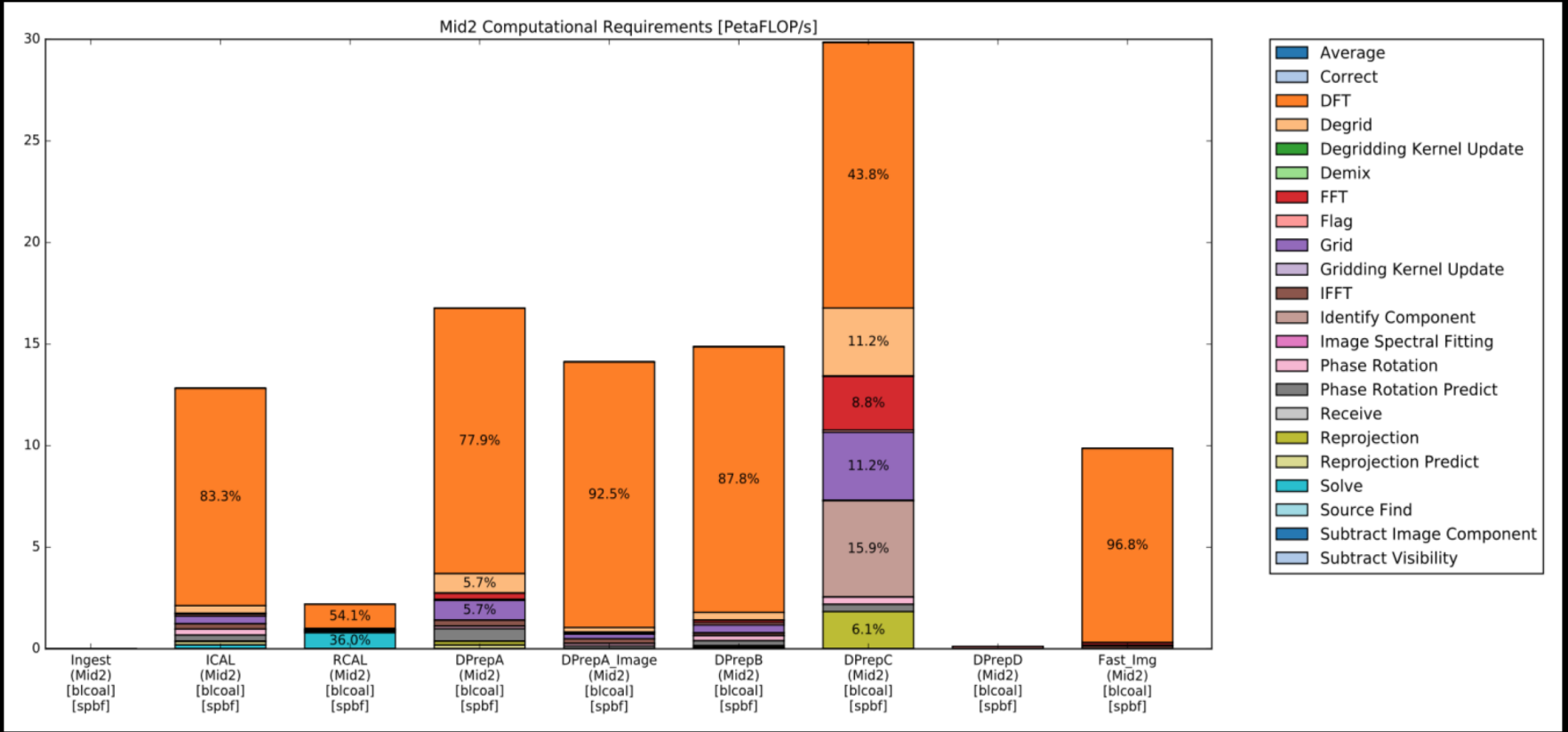
- Two principal data types
  - input is visibility – irregular, sparse uvw - grid of baselines
    - 256 x 256 = 64K baselines
    - 3 x 8B values
    - Data read continuously at 10 TB/sec
  - Image grid - regular grid in sky image:
    - Resolution 10^5 x 10^5
    - Frequency bands: 64K
    - Values: 3 x 8B (3 complex DP floats)
    - Just 1 frequency leads to 240 GB



Example of near disaster …

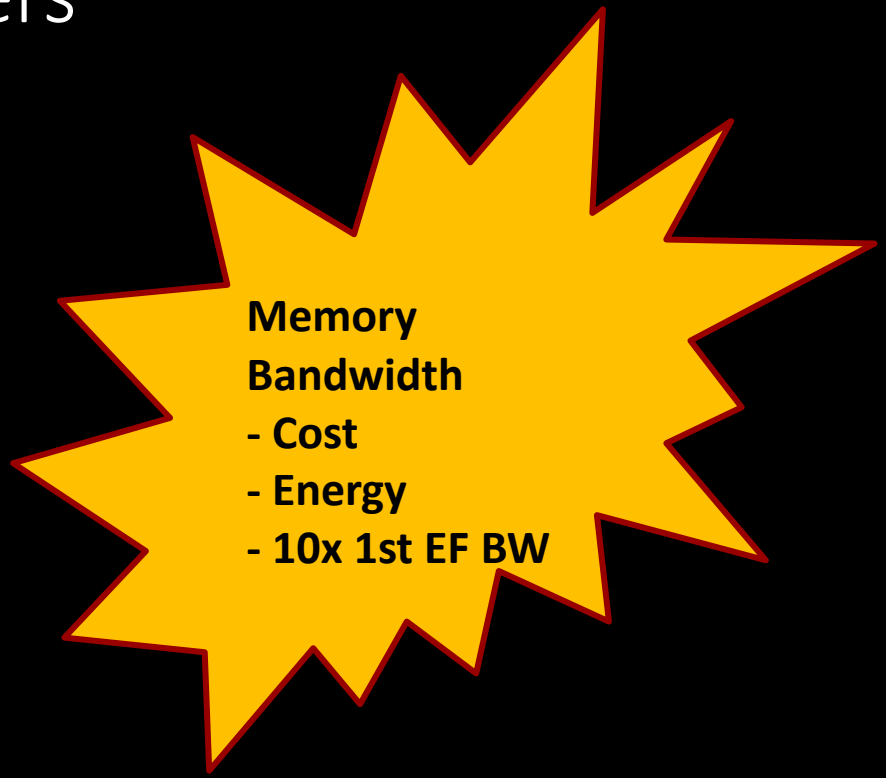
- Memory bandwidth
- The parametric model predicted the 200 PB/s early on.
- By sheer coincidence, in a vendor meeting, we asked what is the energy use for moving 1 byte. It's 50 pJ with HBM2 memory.
- That's actually 10MW, and that was actually more than the original energy budget for the system
- This one was found "just in time".

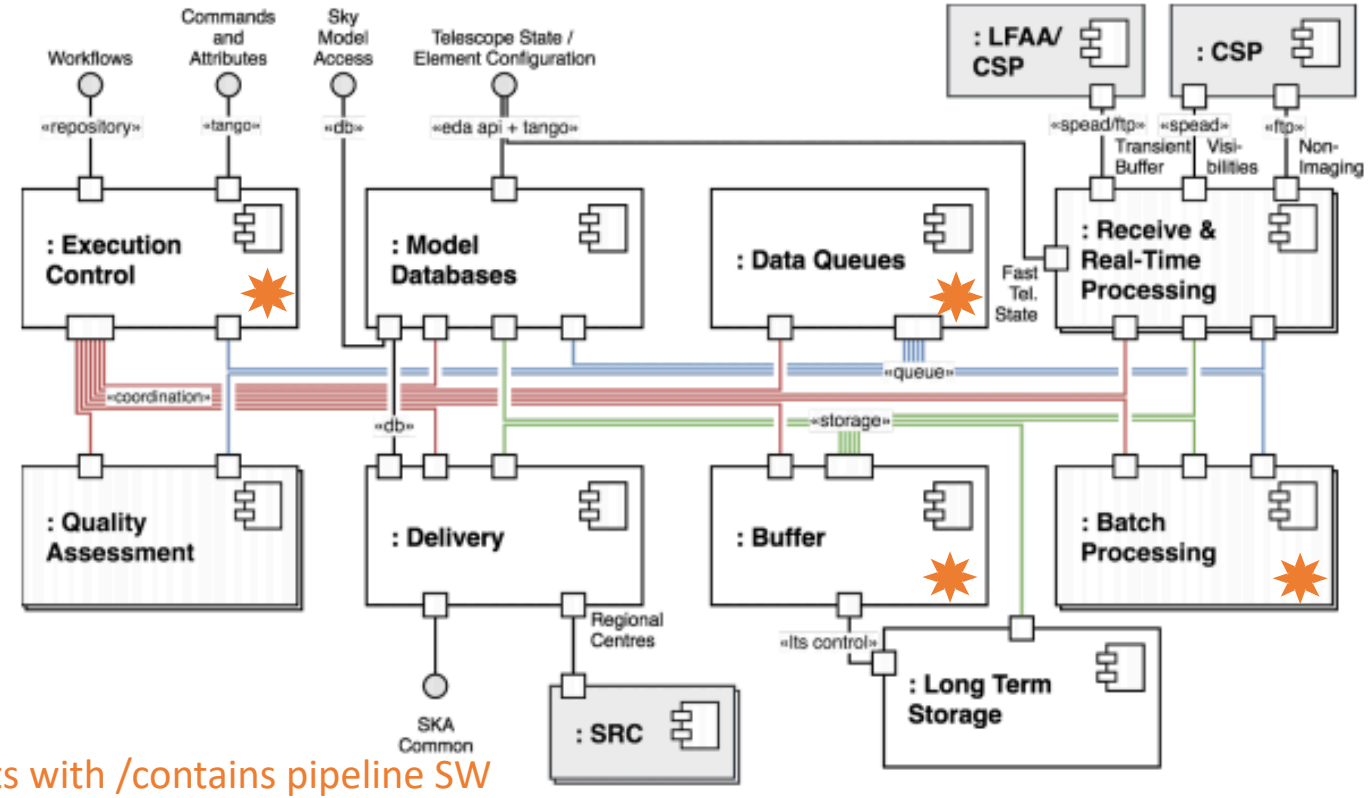
Relative kernel cost (from parametric model)



Supercomputer parameters

2025	LFAA (AU)	Mid (SA)
FLOPS	100 PF	360 PF
Memory bandwidth	**200 Pb/sec**	**200 Pb/sec**
Buffer Ingest	7.3 TB/s	3.3 TB/s
Budget	45 M€	3.3 TB/s
Power	3.5 MW	2 MW
Buffer storage	240 PB	30 PB
Storage / node	85 TB	5 TB
Archive storage	0.5 EB	1.1 EB





Interacts with /contains pipeline SW

Figure 6: Science Data Processor (SDP) Component & Connector Primary Representation (LFAA = Low Frequency Aperture Array, CSP = Central Signal Processor, SRC = SKA Regional Centre)

Co-design challenges

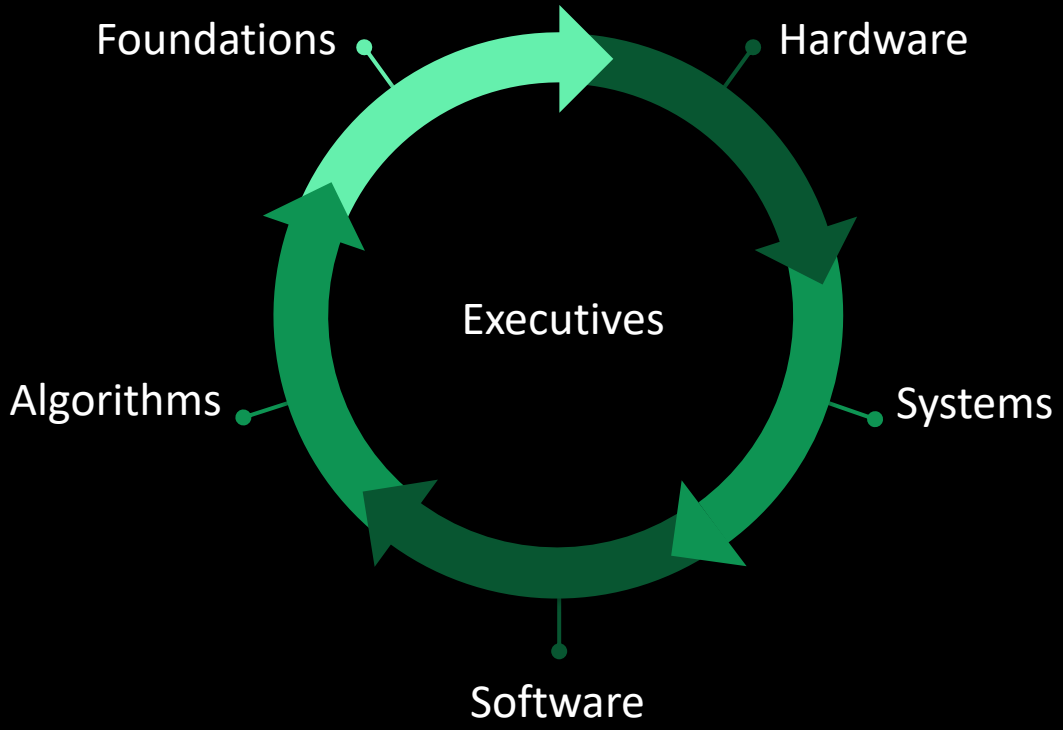
This lecture could now go in multiple directions

- New hardware that is becoming available: AI accelerators like TPU or Graphcore
- Details on algorithmic and mathematical approaches and alternatives
- How the SKA software interacts with the telescope management system

I want to highlight difficulties in getting mature CS approaches to bear on the problem:

- Issues in **problem specification**
- **The pipeline problem: the performance sensitive numerical processing**
  - Outline some steps the SKA community took
  - Some impressions about **HPC DSL's**

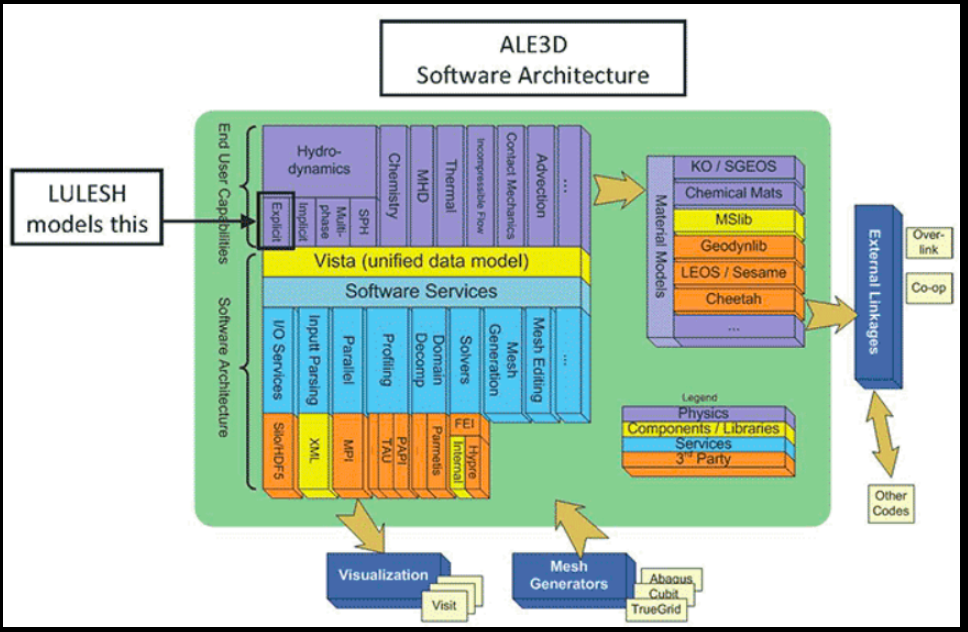
Co-design



Lesson:

This process lacks details
What exactly should be collaborated on?

LLNL Proxy App – success story



Lawrence Livermore National Laboratory (LLNL)

- identified some 10 proxy apps for critical components of exa-scale calculations.
- E.g. Lulesh for 3D fluid motion.

The proxy apps typically saw multiple implementations:

Notably a Haskell implementation was 2x smaller, faster, and easier to understand.

Conclusions …

SKA Conclusions

- Pipelines
  - The top-level SKA compute framework should support almost any framework for pipelines.
  - Leave the pipeline execution framework for later.
  - Selection by those implementing the pipelines.
- Python is necessity, many DSL's too complicated to modify
- Astrophysicists' solutions and optimizations not mathematician's / computer scientists'

Computer Science Conclusions

We have not offered a state-of-the-art solution to SKA

- We need a specification of the SKA problem, truly enabling non-domain experts to assist
- For SKA this is particularly difficult
- We need powerful libraries and/or DSL's supporting domain specialists.

SKA to enable non-experts

What was delivered?

- Parametric model describing data
- Interaction with telescope manager.
- Reference library of routines

Also Needed for non-experts

- Algorithms & math digestible for non-astrophysicists
- Sample pipeline use cases: what has to happen with what constraints
- Scaled down evaluations

CS Lessons: Specification

FireDrake – Simply Stated Scientific Computing

Simple scientific problems often have an extremely short full specification along the following lines:

- solve a differential equation F(D, v) = 0 for v, given the operator F(D, -)
- let v obey initial and boundary conditions
- select a mesh, discretization and solver to approximate the solution
- here we enter “known programming terrain”

FireDrake is a DSL that takes as input a differential equation, boundary conditions and a finite element strategy and produces distributed code!

Many problems are not quite so simple.

Key problem 1: mathematical abstractions usually suffice for succinct problem statements. Programming languages miss the types for these abstractions. Placing them in DSLs inhibits inter-operability.

CS Lessons: DSLs

Domain Specific Languages

Very many domain specific languages have been developed:

- for Machine Learning
- for High Performance Computing
- Image Processing
- Databases

A few are hugely successful: SQL, NumPy, TensorFlow, Halide

FFTW success

1999 paper

Automatic exploration of re-ordering computations

Halide success

~2012 - Graphics pipelines paper

2016 Automatic Scheduling

2018 Distributed Halide paper

Pixel Visual Core Chip, AHA project agile Hardware

Separate schedule & computation

Halide DSL example

```
Func blur_3x3(Func input) {
  Func blur_x, blur_y;
  Var x, y, xi, yi;

  // The algorithm - no storage or order
  blur_x(x, y) = (input(x-1, y) + input(x, y) + input(x+1, y))/3;
  blur_y(x, y) = (blur_x(x, y-1) + blur_x(x, y) + blur_x(x, y+1))/3;

  // The schedule - defines order, locality; implies storage
  blur_y.tile(x, y, xi, yi, 256, 32)
        .vectorize(xi, 8).parallel(y);
  blur_x.compute_at(blur_y, x).vectorize(x, 8);

  return blur_y;
}
```

Halide image processing

State of the art in separating computation from mechanism.

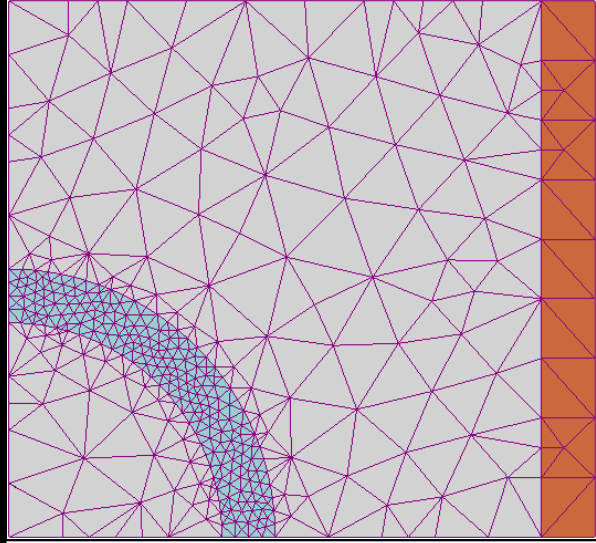
Is the future perhaps something like the following?

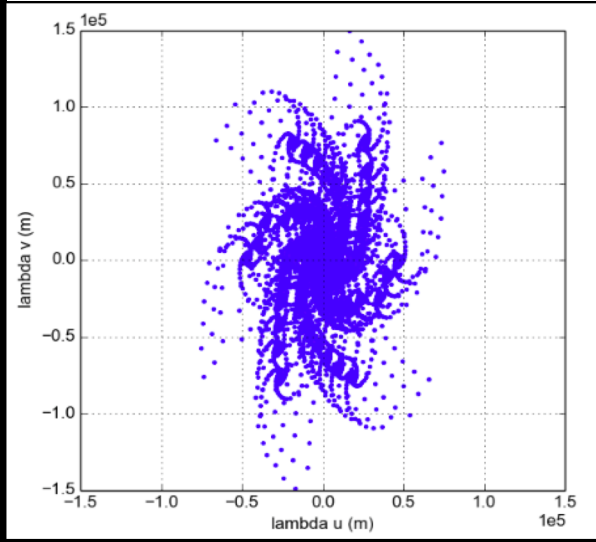
Schedule: a function between linear/dependent data types representing:

- data in the computation
- hardware in the computer

Learned from the DSL studies?

1. Problem specification not easily aligned with existing DSL's
2. HW Architecture support was often raised: tensorflow and MLIR are addressing this beautifully.
3. Data - data can make or break the story – what interface to the data do we need?
4. Algorithmic expressiveness of some DSL's is too limited
5. The DSL's themselves are too complex for domain experts to modify
6. Re-use of types across DSL's is insufficient





SKA pipeline study

We found MANY software projects that looked promising for the pipeline problem, e.g.:

Halide, Legion, Parsec, Swift/T, Cloud Haskell, MPI, DASK, Spark, TensorFlow, ….

Created 50 requirements, and 15 small program specifications. Evaluated 2-3 systems (sadly not TensorFlow).

Initial euphoria about these packages consistently faded as missing elements were identified. Many programs wouldn't run.

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?

See braam.io

Fusion

Memory resource scheduling

Suggested approach

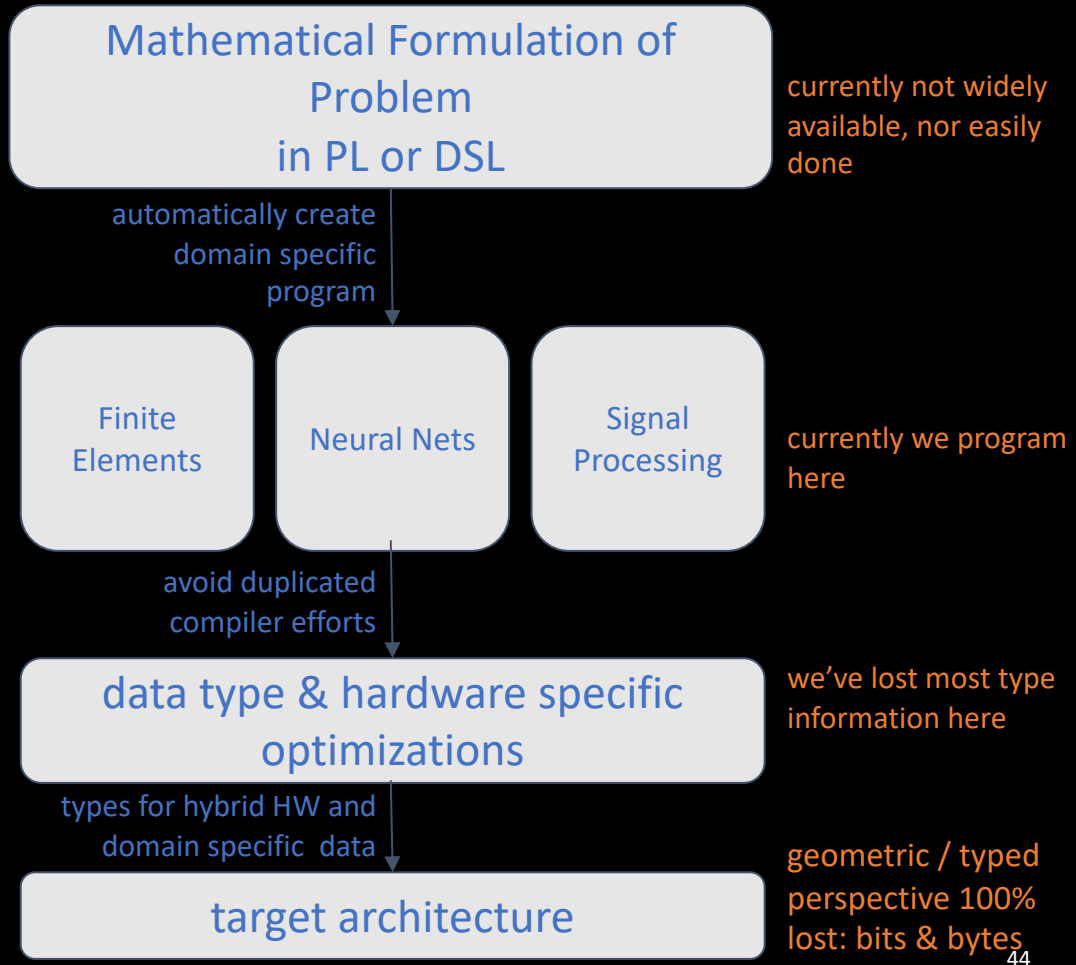
Challenges

1. CS to design good systems
2. drag domain specialists along

Conal Elliott, Paul Kelly, Oleg Kiselov, and many others.

Not the common approach

- Mathematical abstractions become
- Complex Formulas
- Matrix related code
- Handwaving about correctness and precision



Conclusions

Participate …

There are always new projects. Projects always run into major difficulties.

Typically, they appear to be extremely happy to get good computer scientists involved.

Working with new problems is easier than with existing ones
Agree early on what will be decided by a CS team vs a domain specialist team
Offer commitments not alternatives

Conclusions

Key opportunities:

- Participate in new big projects
- Problem specification to join CS and domain communities
- Further PL development for Scientific Computing

Thank you! Questions?

Extra Slides - opportunities

Show the breadth of opportunities, missing API’s

New floating point formats (posithub.org)

- Decouple range and precision
- Significant data savings
- Higher precision calculations

Opportunity:

- In storage & memory use lowest acceptable precision, compress arrays.
- Decompress near computing element & reverse

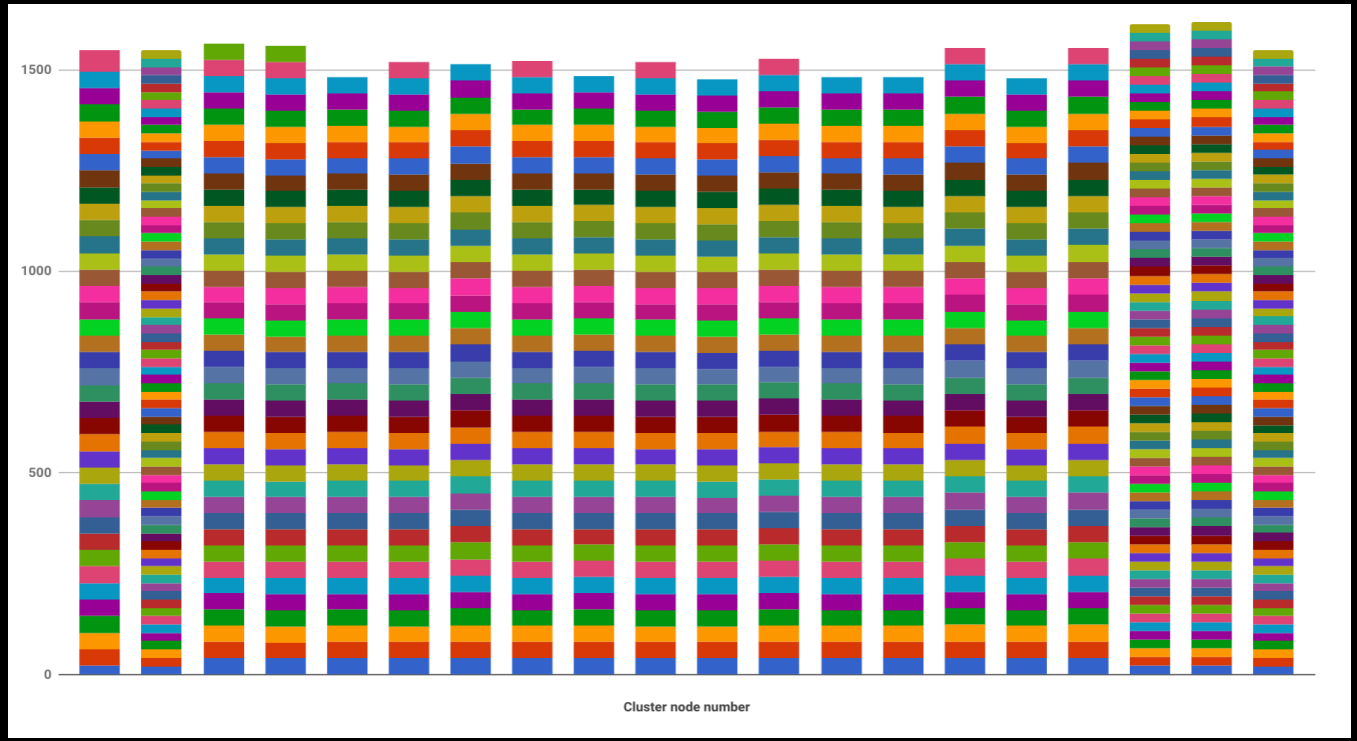
- Lower bandwidth, lower memory and storage capacity! 2x for SKA
- Universal proposal, but technical obstacles
  - Decompressed data should be ”cache only” – limited controls
  - Data needs accessors – bye-bye a[n] in C
  - Decompressing at e.g. 1TB/sec isn’t that easy.

Scheduling work

This depicts a model SKA pipeline scheduled on 20 nodes for load balancing.

There are dozens of job schedulers.

No agreed API's
Few resource balancing support systems

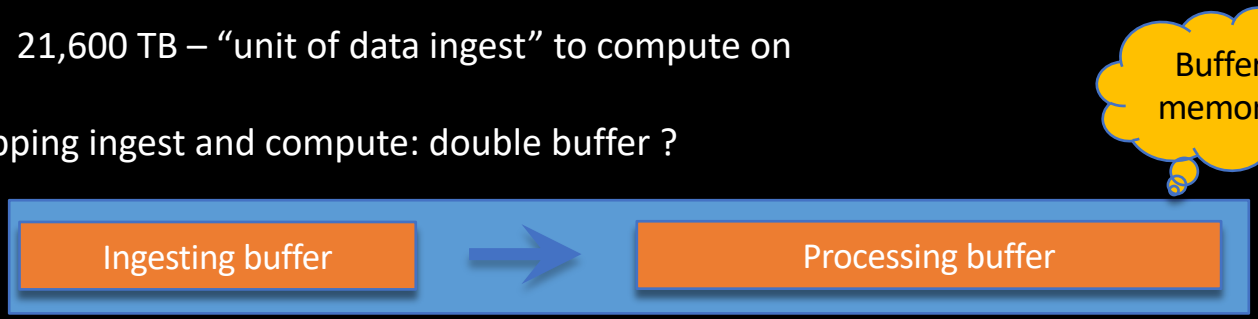


Double buffer pitfall

Processing for SKA requires up to 6 hours of ingest – buffer that.

21,600 TB – “unit of data ingest” to compute on

Overlapping ingest and compute: double buffer ?



Double Buffer: ~50PB, write 1TB/sec, read 10TB/sec

But processing time is uneven –

- **Double buffer: minimizes storage cost, requires all computation to finish during buffering time.**
- **For uneven job lengths, double buffer + e.g. 10% can reduce compute requirements by e.g. 50%**

Cache hierarchies - stitching and decomposing

- Suppose a computation is too large to fit into addressable memory
- Decompose it into pieces either distribute them or sequentially do them on one node
- Examples:
  - Very large Fourier transforms
  - Using memory caches
- Benefit: trade memory capacity for IO

Opportunity: develop libraries with intelligent caching.