Chapel in the (Cosmological) Wild

Nikhil Padmanabhan

2 June 2016

CHIUW 2016

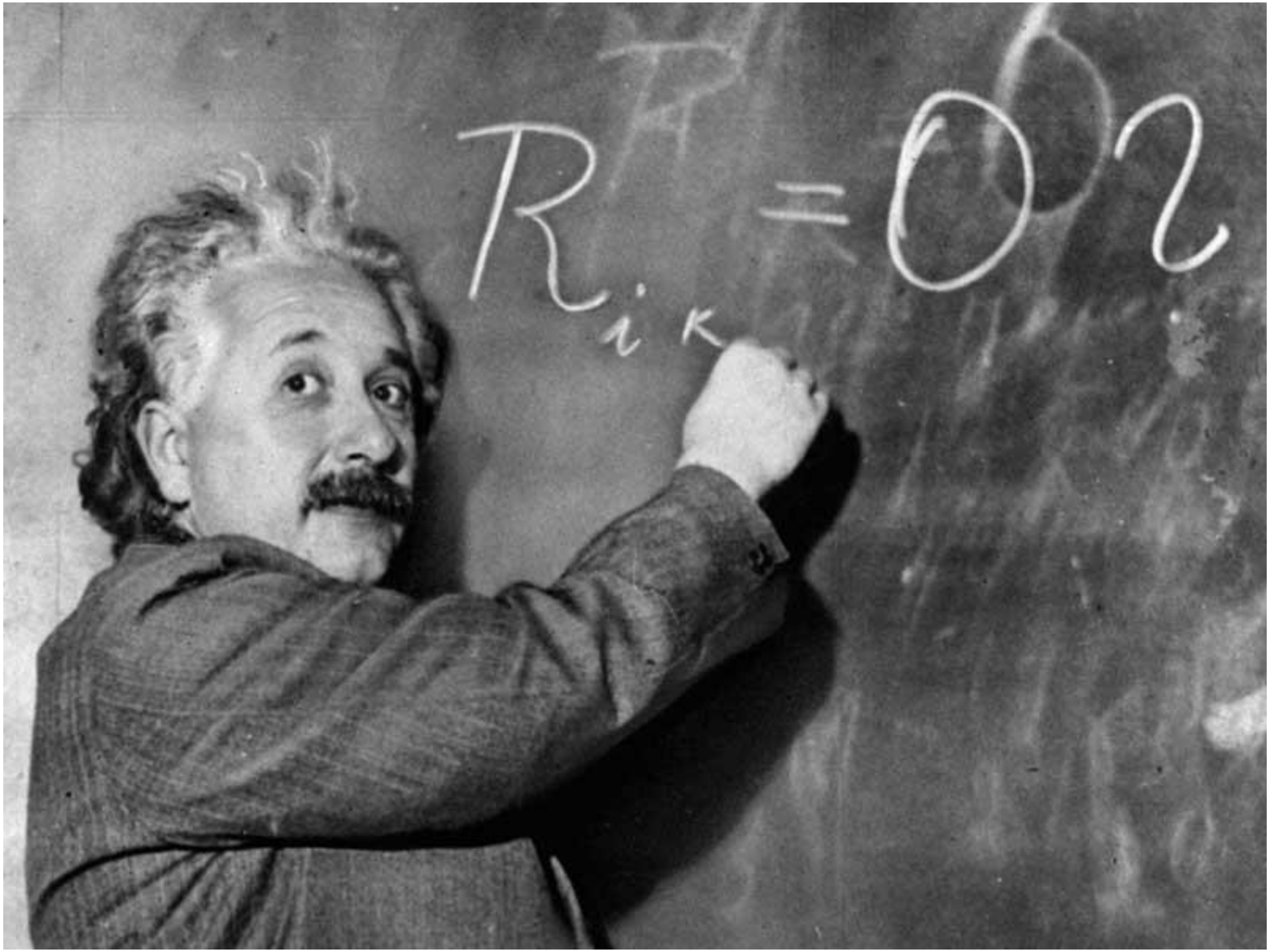
About…

- My day job is as an astrophysicist, specializing in cosmology
- A Chapel enthusiast
  - Bumped into Chapel early in its (public) existence
  - Was intrigued, but not compelled.
  - Revisited around 1.10
    - Language looked more polished/stable
    - Met up with Brad Chamberlain, discussed interest
    - FFTW
  - One use case to date, a few proof-of-principle applications
  - 1.13+ now has most bits that I need, hoping to use more broadly
- Performance is important, but so is ease of prototyping new ideas
  - Happy to take a ~x2 hit over a well-tuned case
  - Absolute "wall"-time matters; often the distinction between 1 min vs 1 s vs 1 ms does not matter (I can't think that fast!)
  - But sometimes it does – so important to be able to find slow steps to optimize
- C++/Mathematica/Python are my usual tools

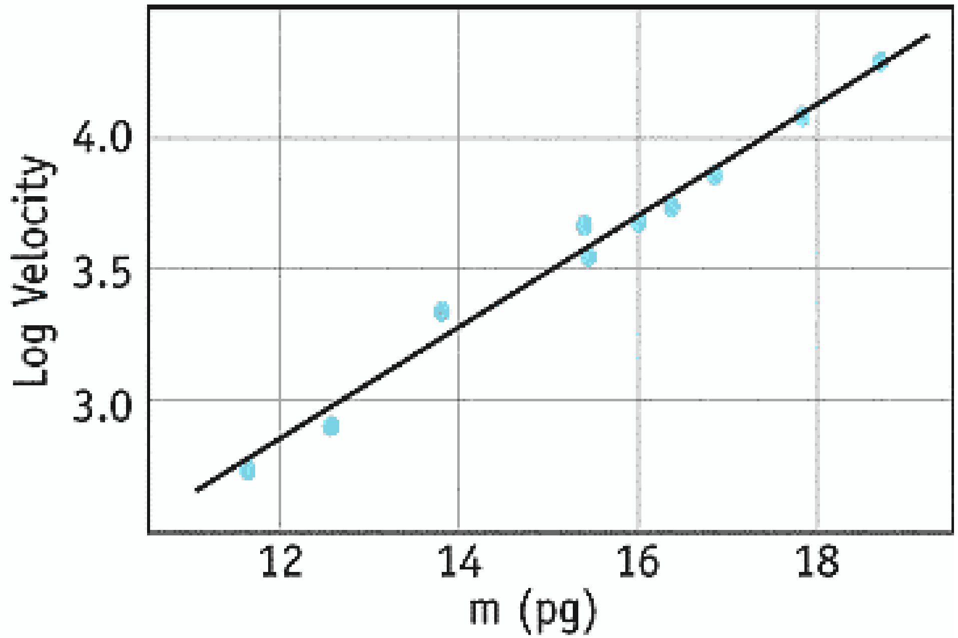
Warnings!

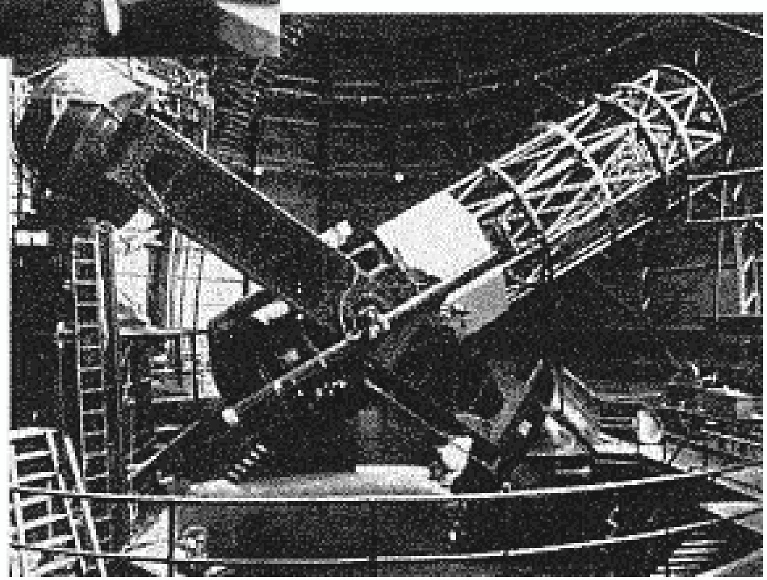
- I'm not trained in CS, nor am I a "computational scientist"
  - Code is just a means to an end…
  - Expect to see non-optimal code
- These slides have not been vetted by the Chapel team
  - Although they've helped significantly in lots of the Chapel code I've written
    - Brad Chamberlain, Michael Ferguson, Ben Harshbarger
  - Mistakes are all mine
  - Some slow code may not be Chapel's fault, but mine!
- Not my usual patter, so apologies in advance for any glitches…

A cosmological constant

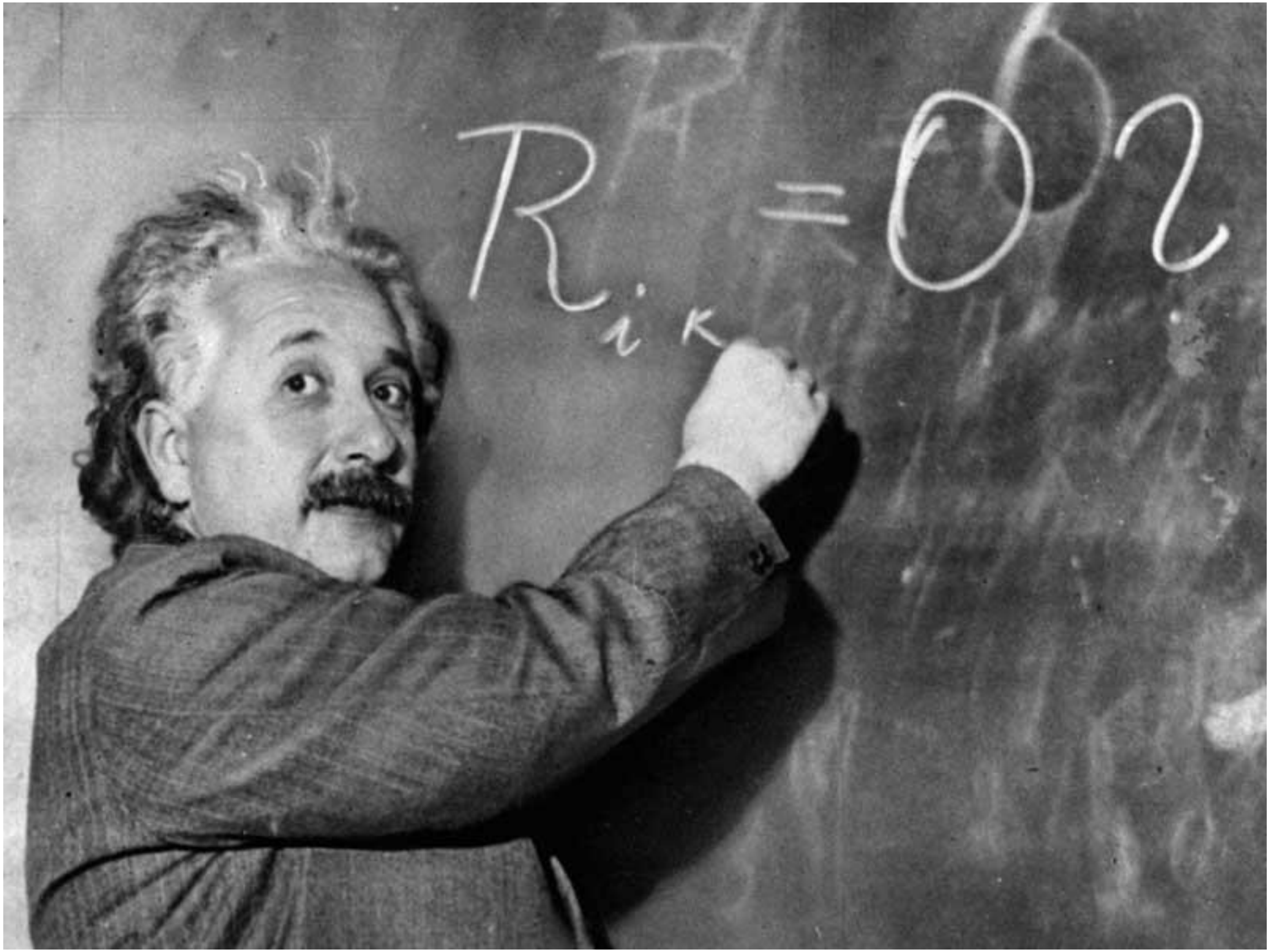



Edwin Hubble

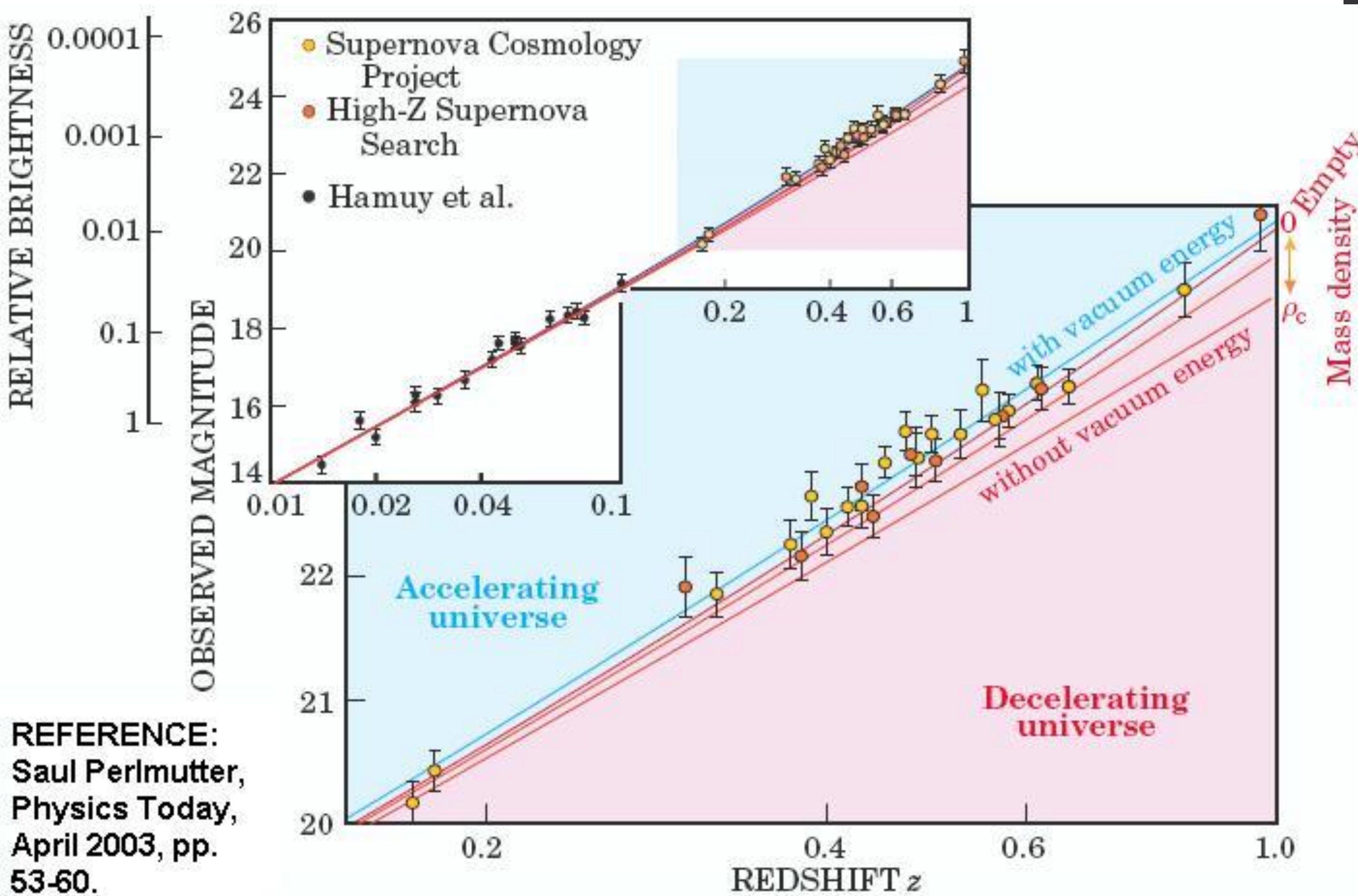



Mt. Wilson
100 Inch
Telescope

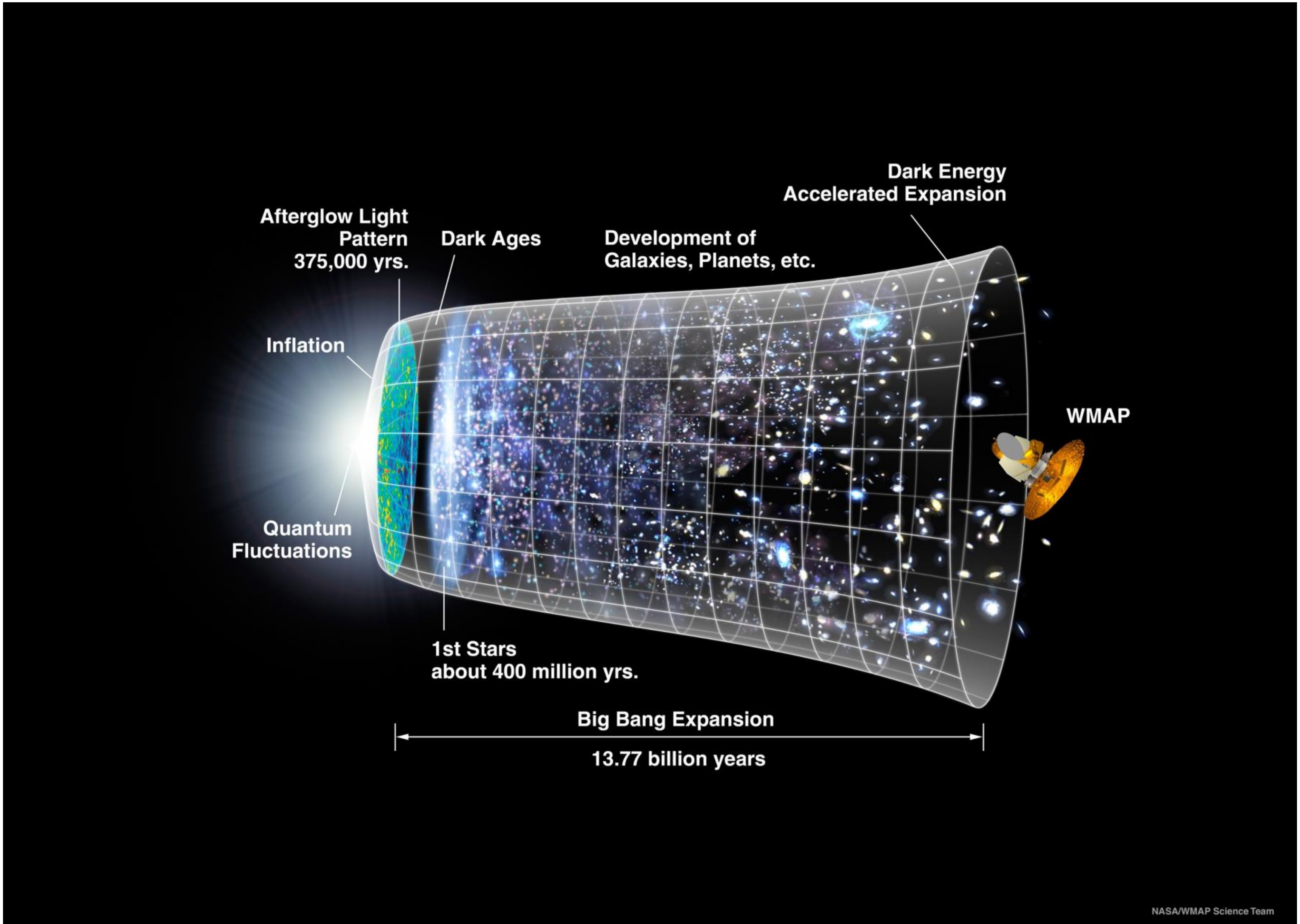
His biggest blunder?



A big surprise : an accelerating Universe



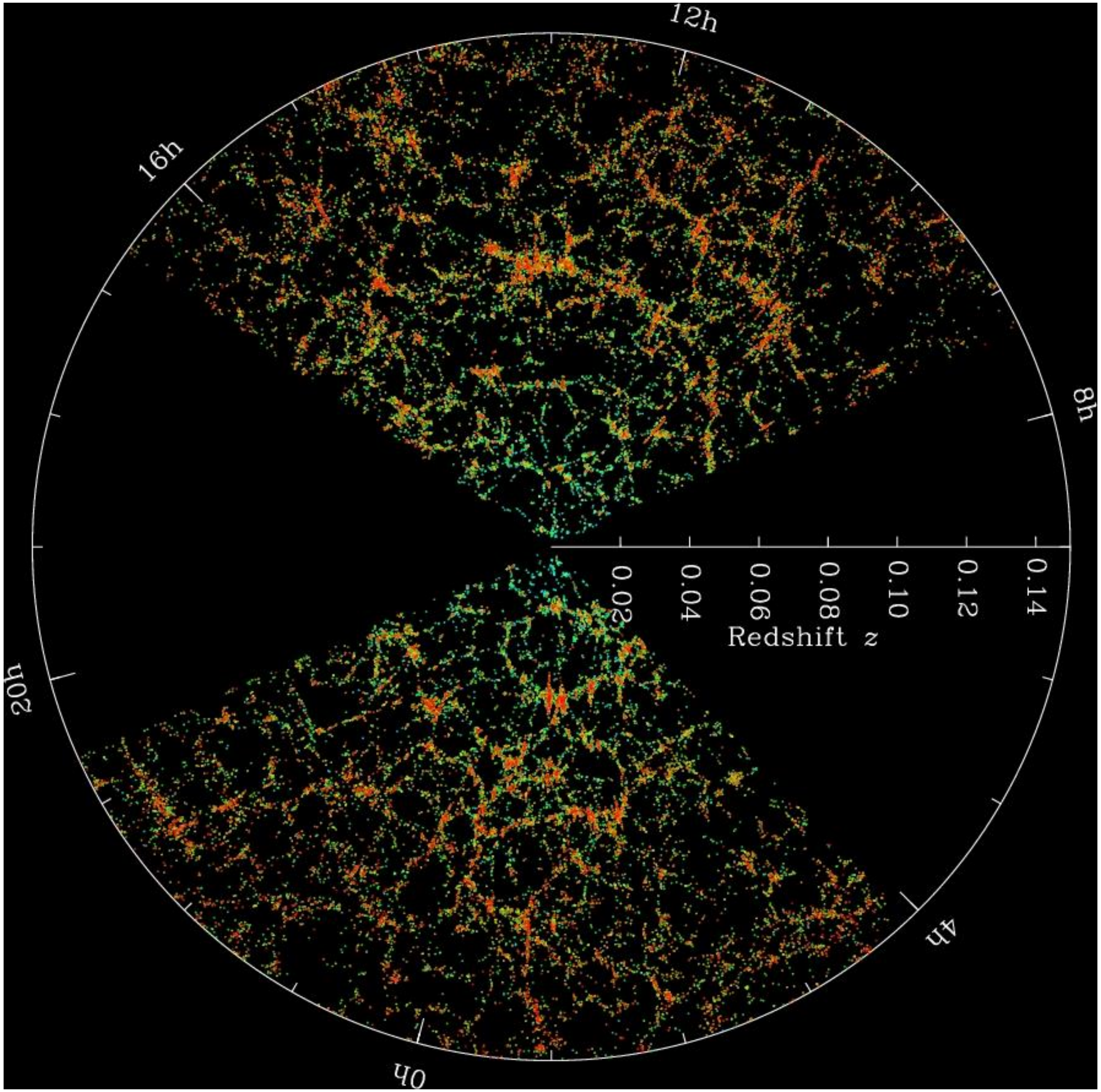
REFERENCE: Saul Perlmutter, Physics Today, April 2003, pp. 53-60.


Dark Energy
Accelerated Expansion
Afterglow Light
Pattern
375,000 yrs.
Dark Ages
Development of
Galaxies, Planets, etc.
Inflation
WMAP
Quantum
Fluctuations
1st Stars
about 400 million yrs.
Big Bang Expansion
13.77 billion years
NASA/WMAP Science Team

Cosmic cartography

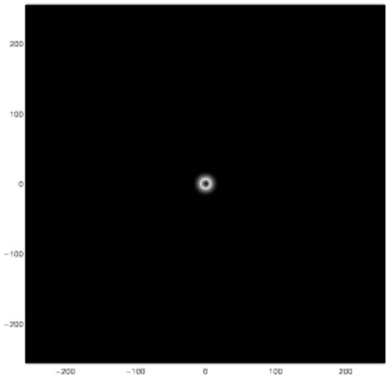
Original in the John Carter Brown Library at Brown University

Cosmic cartography



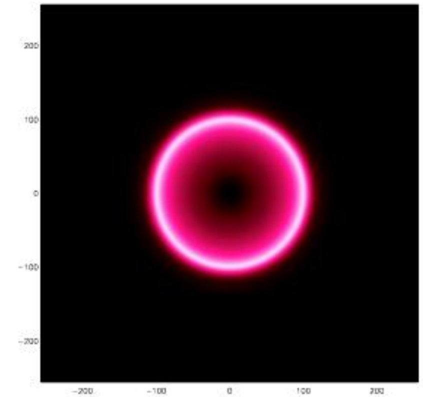
www.sdss3.org

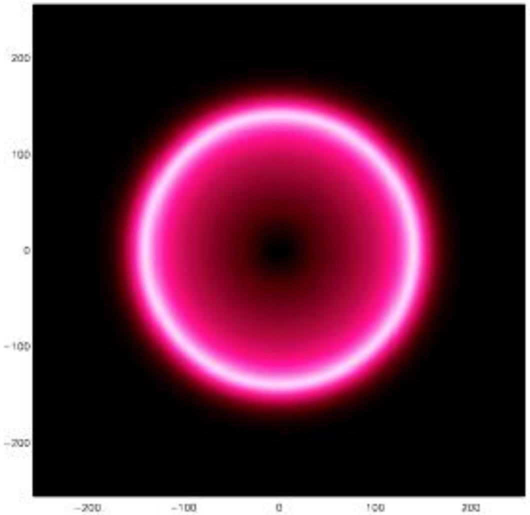
Constructing a Standard Ruler



Begin : hot "soup" of
electrons, photons
A sound wave starts.

Shell expands at speed
of sound 0.578c

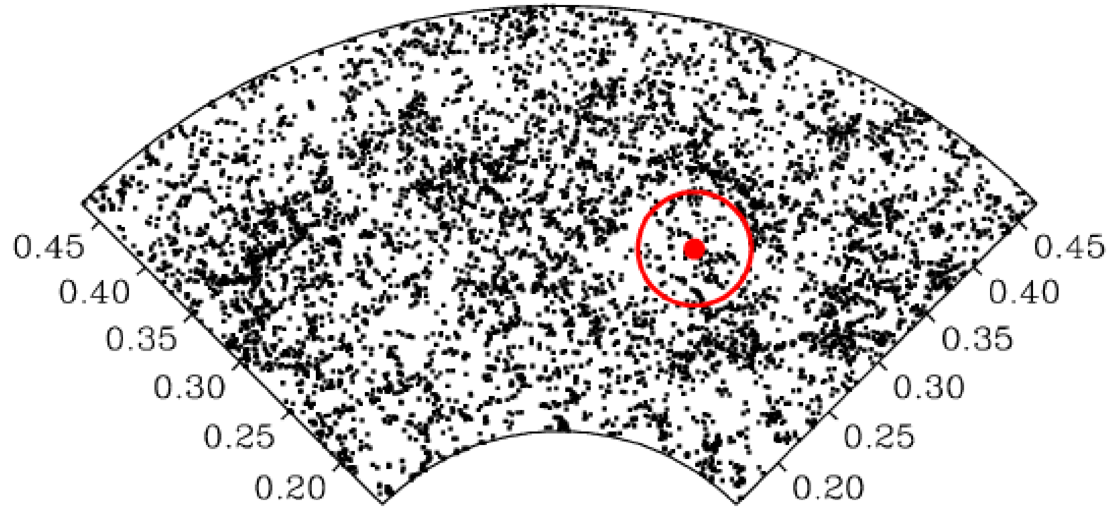




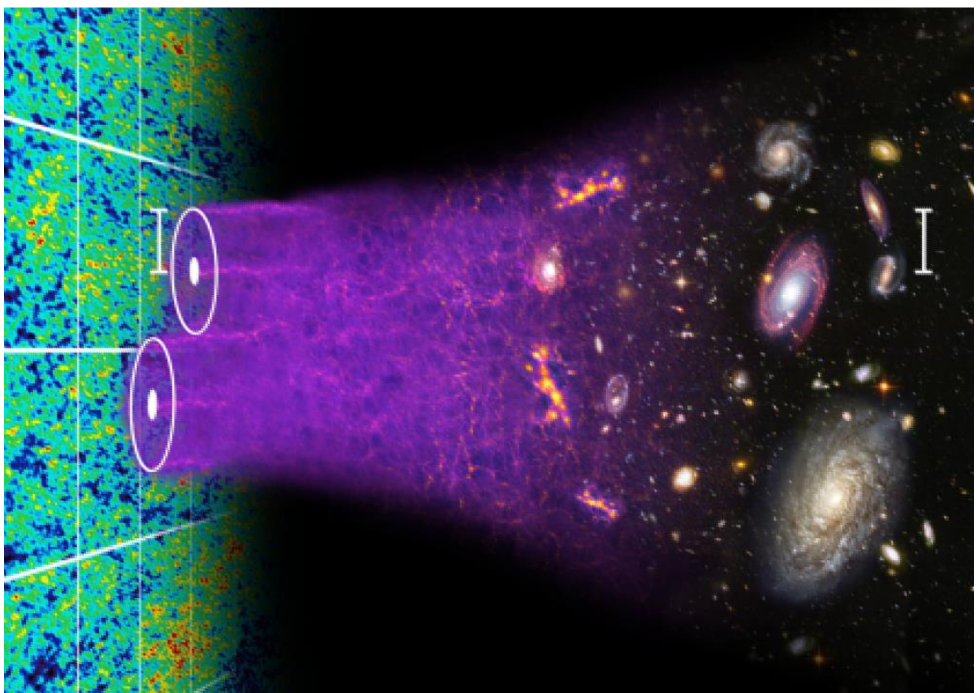
Universe "freezes" 300,000 yrs ABB.
"Ripple" frozen in.
A standard ruler
Statistical in nature

Measuring The Ruler : Galaxies

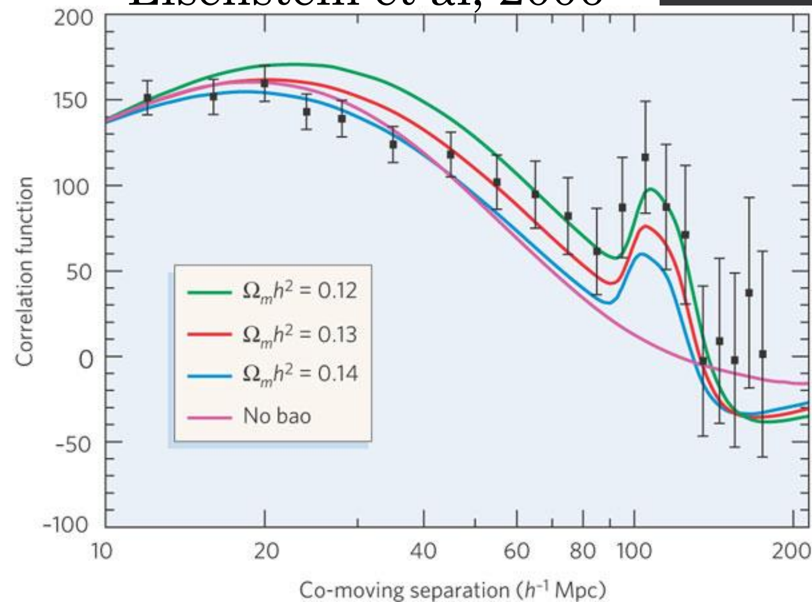
A preferred scale for galaxy separations



www.sdss3.org

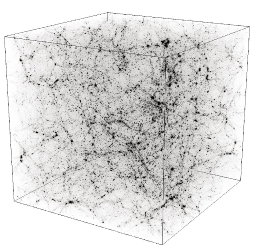


Eisenstein et al, 2006



Constructing a galaxy survey

Su



3D Map

aged 14K
of sky)
ects

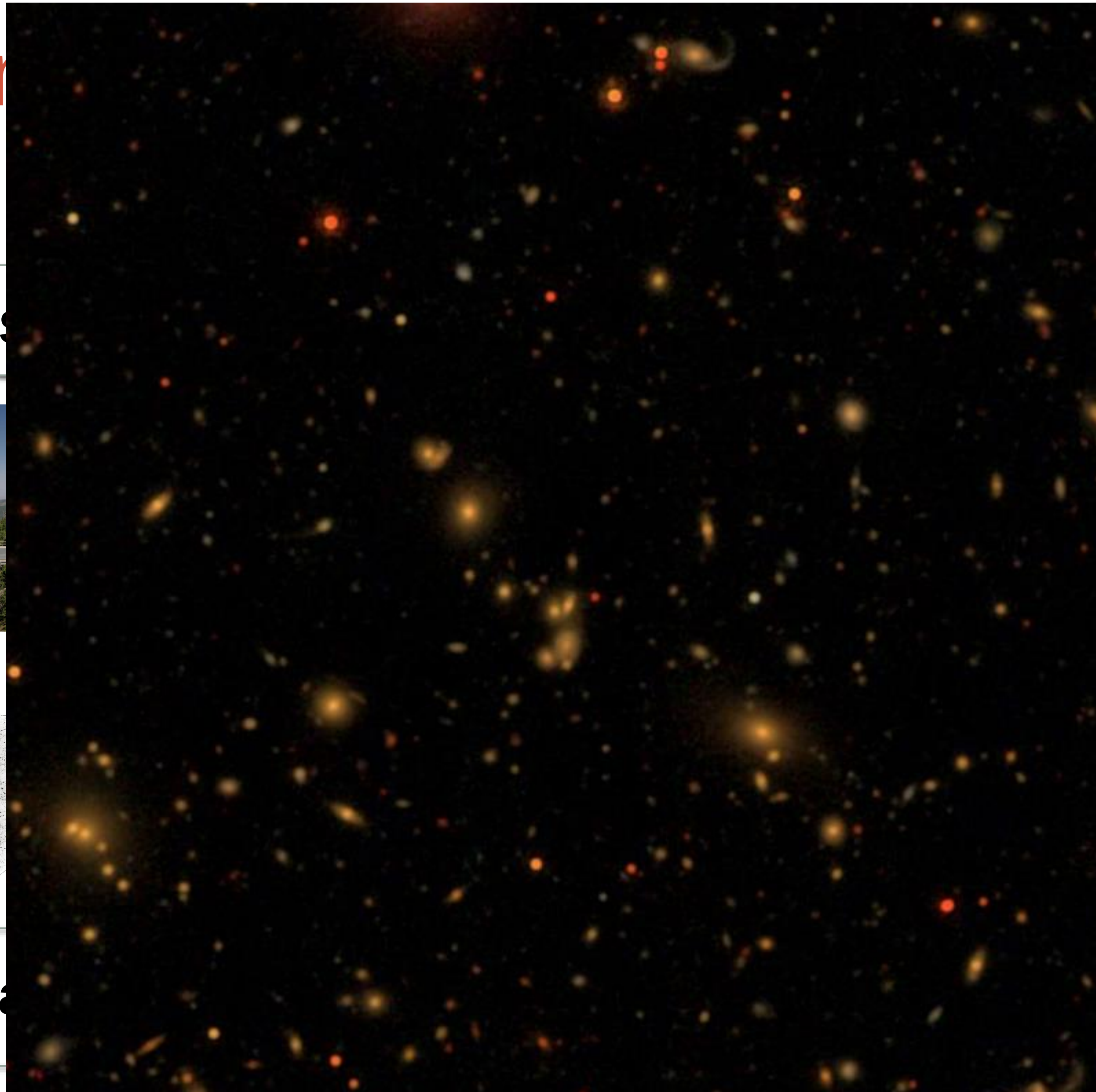
5M

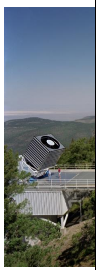
cts

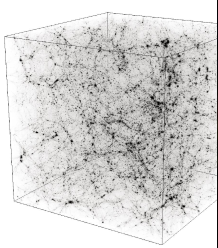
ra

ne

Co



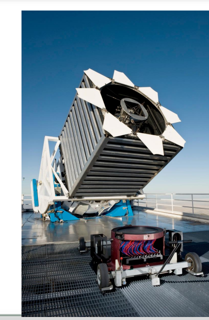




S-I/II imaged 14K
eg (1/3 of sky)
ion objects
cted

1.5
M

ct objects

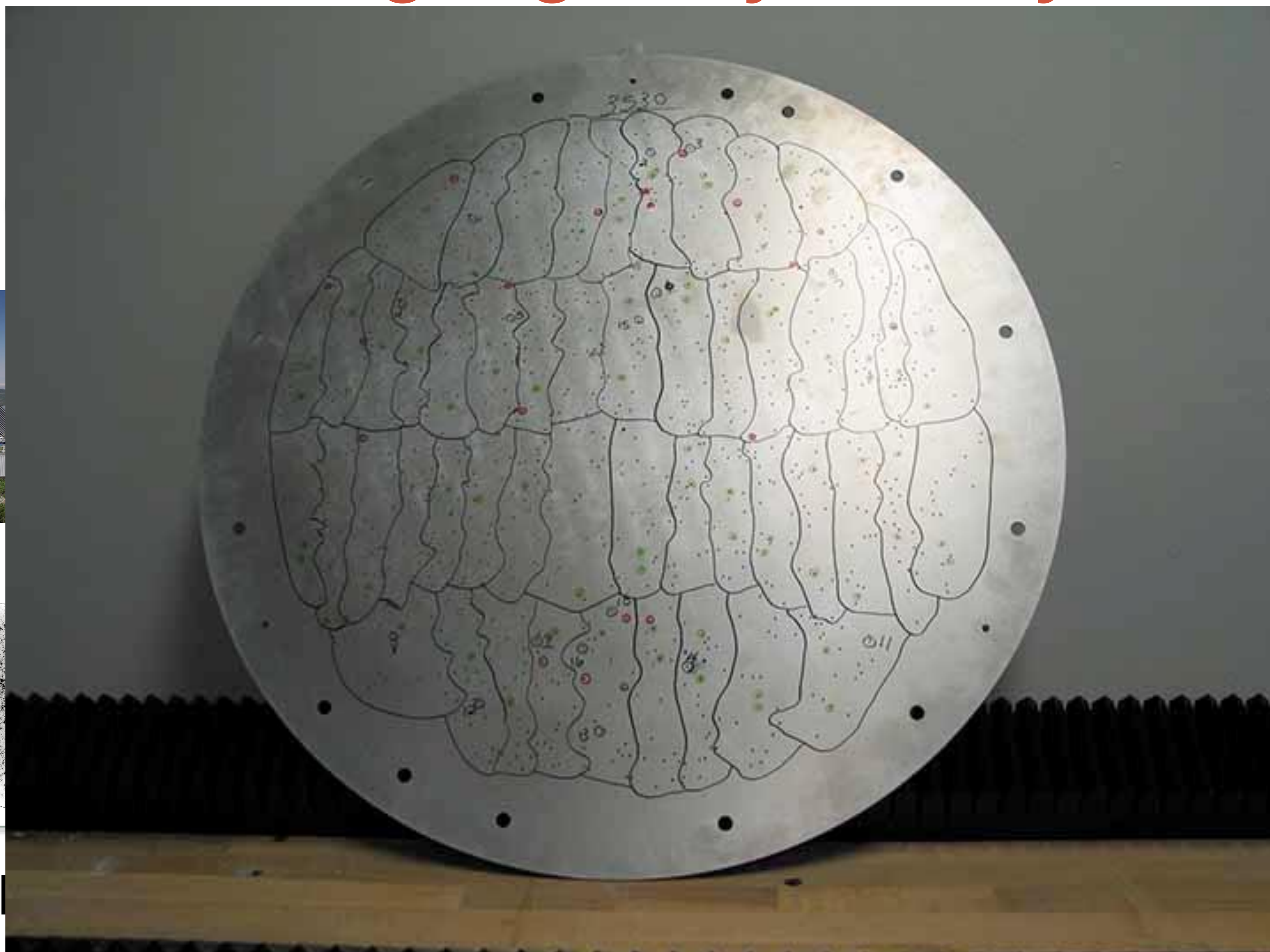


Spectra

1000 at a time

3D Ma

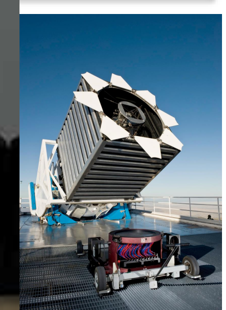
Constructing a galaxy survey



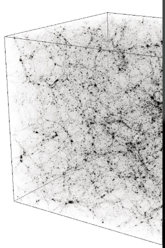
imaged 14K
/3 of sky)
bjects

1.5M

jects



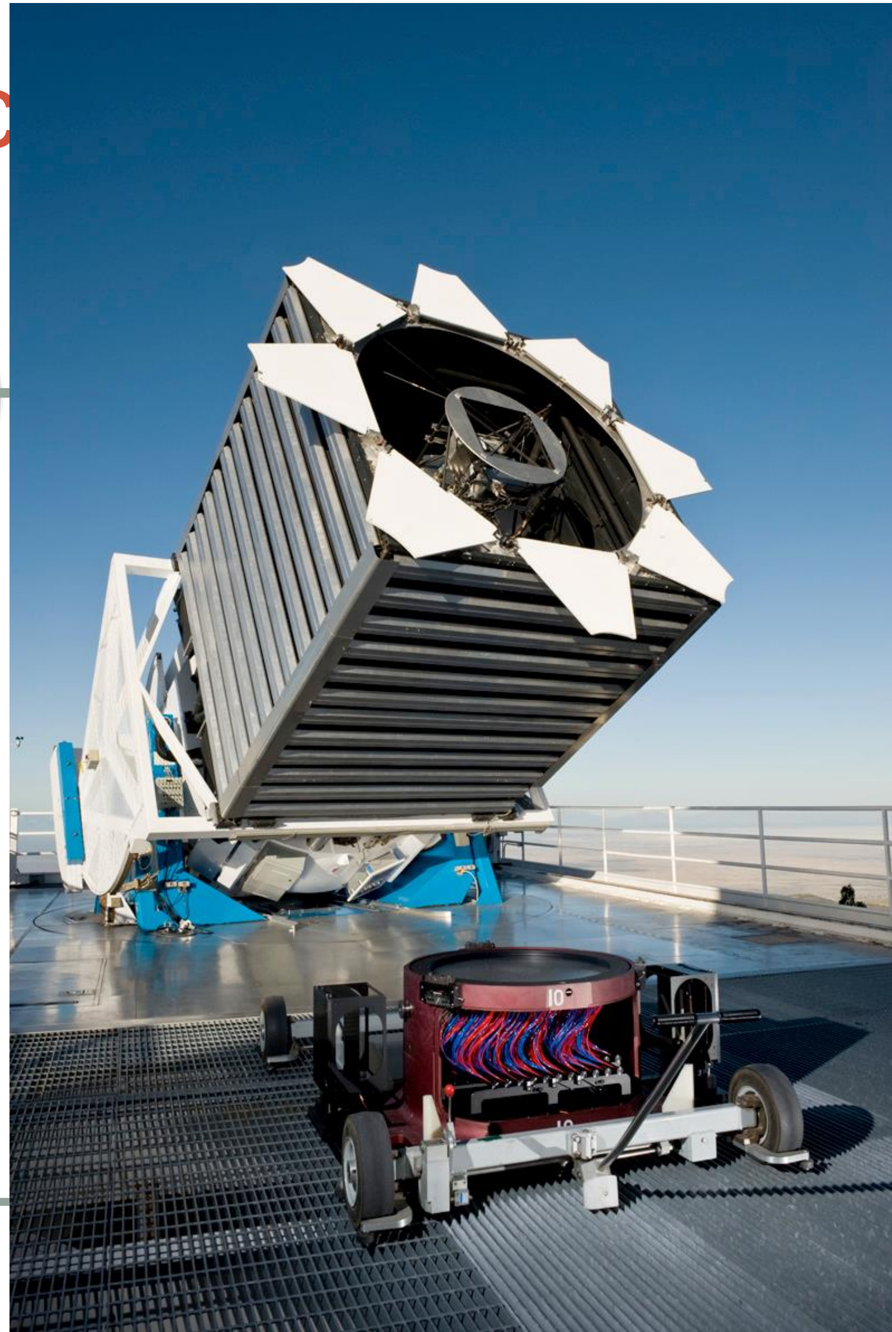
ctra



3D I

1000 at a time

Construc

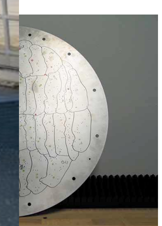


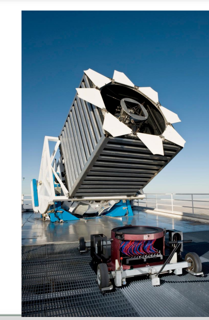
Survey

SDSS-I/II imaged 14K sq. deg (1/3 of sky)
~ billion objects detected

1.5M

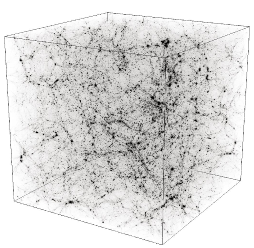
Select objects





Get Spectra

1000 at a time

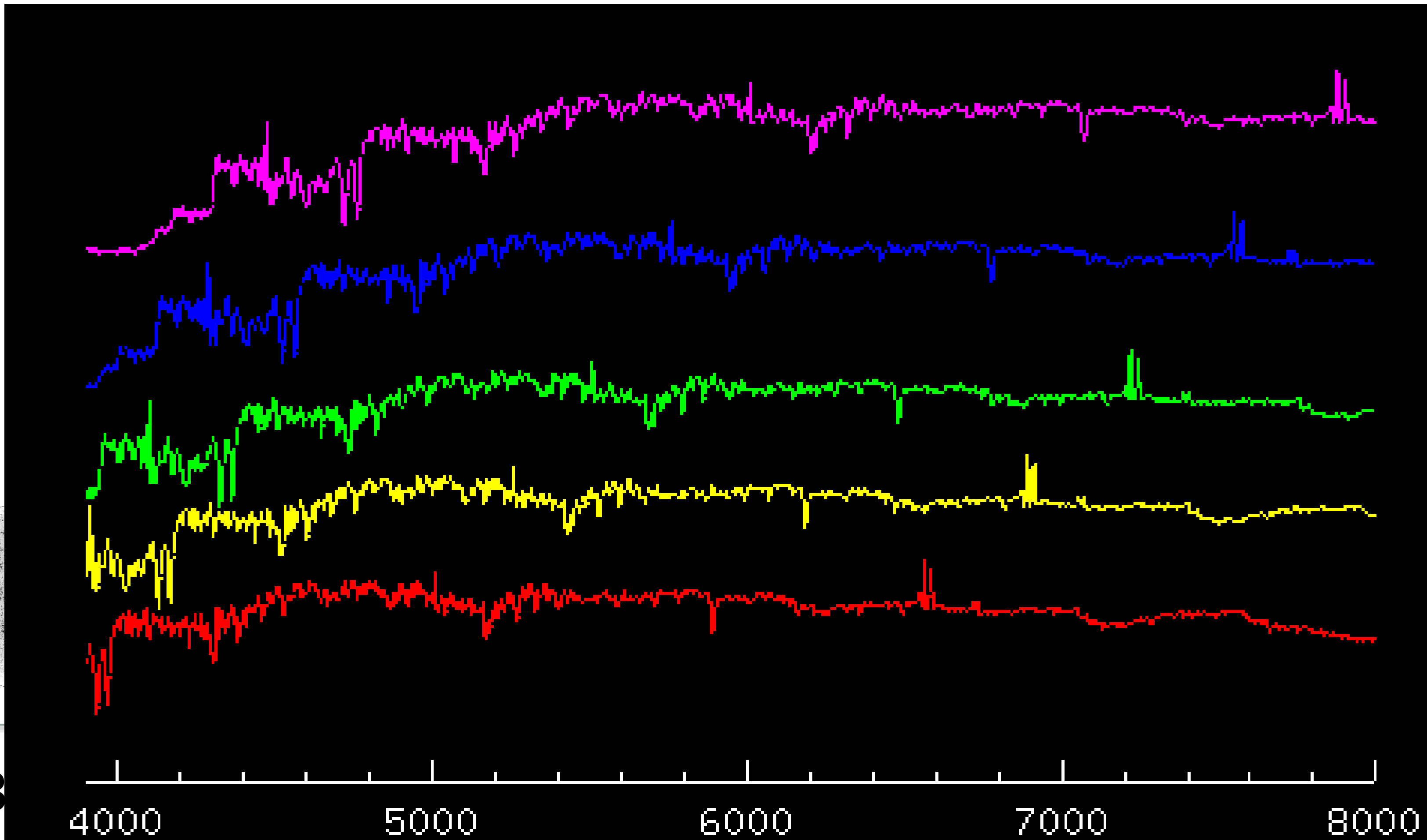


3D Map

Constructing a galaxy survey

d 14K

ky)



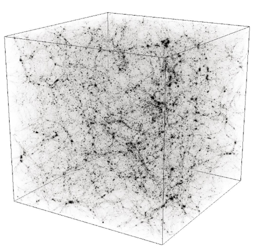
3

redshifts

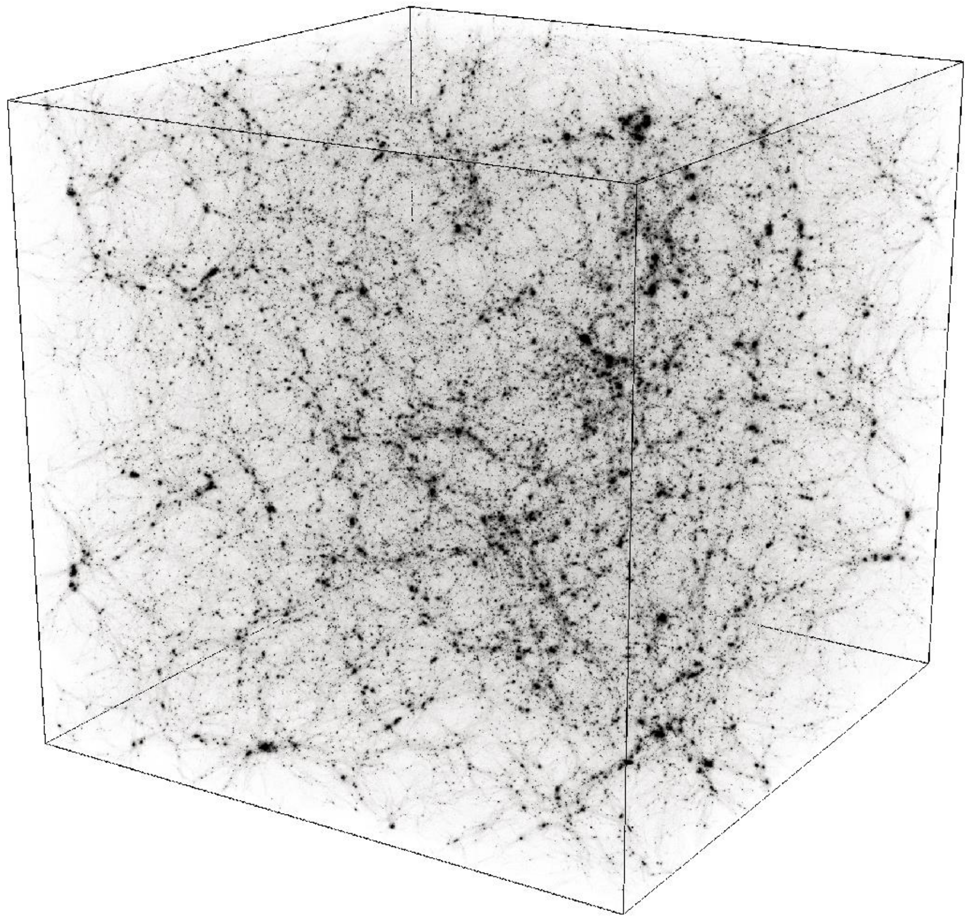
1000 at a time

Constructing a galaxy survey

Surve



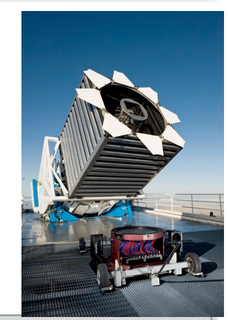
3D Map



S-I/II imaged 14K
leg (1/3 of sky)
lion objects
cted

1.5
M

ct objects



: Spectra

1000 at a time

What kinds of computations

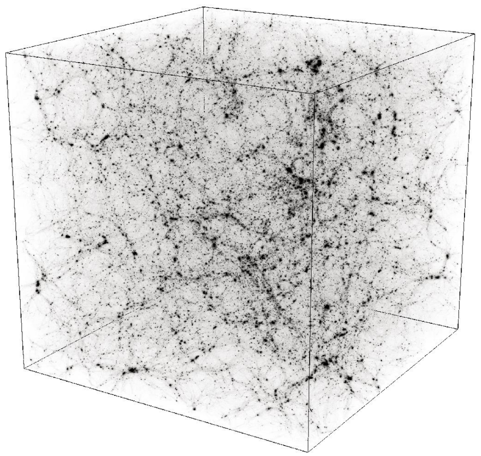
- Often the question isn't one of implementation, it's the question
- Simulations of the formation of structure in the galaxy distribution
  - See Katrin Heitmann's keynote talk yesterday
  - Performance matters!
- Characterize spatial distributions of galaxies
  - N-point functions
  - Find groups/clusters of galaxies
  - Simplest algorithms here are analogous to N-body calculations
- Potential/force calculations
  - Solving variants of the Poisson equation
  - FFTs
  - Multigrid
- Simulations
  - We observe a random realization from all possible Universes.
  - Theory predicts averaged quantities
  - Need to understand the distributions
  - Need to repeat calculations many times
- Many computations are embarrassingly parallel!

Why Chapel? And not something else?

- Python?
  - Python works well when doing well optimized tasks
  - Great ecosystem – lots of users
  - Not so good when first statement is not true.
  - Sometimes forces you to use unnatural idioms for tasks (a for loop is sometimes the simplest answer)
  - Memory/temporaries
- C++/MPI?
  - C++11/14 is getting quite high-level
  - Performance
  - OpenMP/MPI is well-established, good tooling
  - MPI is rather verbose/tedious, especially for simple tasks
  - Still no native multidimensional support
- Chapel?
  - Promise of easy abstractions for parallelism
  - Promise of performance
  - Domains are GREAT!

A Particle-Mesh Code

- "Toy" problem
  - There are more efficient/accurate algorithms
  - The pieces are quite reusable
- Particles
  - Track the distribution of matter
  - Evolve under gravity
- Mesh
  - Used to accelerate gravity calculation by solving Poisson's equation on a grid
  - Thin wrapper around FFTW



Setup

```
config const N=128;
const Nhalf=(N/2+1);
const FullD = {0.. #N, 0.. #N, 0.. #(2*Nhalf)};
const RealD = FullD[..,..,0.. #N];
const ReD = FullD[..,..,0.. #(2*Nhalf) by 2];
const ImD = FullD[..,..,0.. #(2*Nhalf) by 2 align 1];
const FreqD = FullD[..,..,0.. #Nhalf];
const Ntot = N**3;


/* Define the box dimensions */
config const Lbox=2000.0; // Mpc/h
```

Config parameters are great – no longer need to parse input files (reproducibility – saving all config parameters?)

Domains are very expressive (handle FFTW storage)

Grid deposition

```
// Loop over the particles
forall ipart in particles {
  // Compute the grid index
  var ndx : NDim*int;
  for ii in Dims1 do ndx(ii) = floor(ipart(ii)) : int;
  var dlo, dhi : NDim*real(64);
  dhi = ipart - ndx;
  dlo = (1,1,1) - dhi;
  var (ii,jj,kk) = ndx;
  tmp[ ii     , jj     , kk     ].add(dlo(1)*dlo(2)*dlo(3));
  tmp[ ii     , jj     ,(kk+1)%N].add(dlo(1)*dlo(2)*dhi(3));
  tmp[ ii     ,(jj+1)%N, kk     ].add(dlo(1)*dhi(2)*dlo(3));
  tmp[ ii     ,(jj+1)%N,(kk+1)%N].add(dlo(1)*dhi(2)*dhi(3));
  tmp[(ii+1)%N, jj     , kk     ].add(dhi(1)*dlo(2)*dlo(3));
  tmp[(ii+1)%N, jj     ,(kk+1)%N].add(dhi(1)*dlo(2)*dhi(3));
  tmp[(ii+1)%N,(jj+1)%N, kk     ].add(dhi(1)*dhi(2)*dlo(3));
  tmp[(ii+1)%N,(jj+1)%N,(kk+1)%N].add(dhi(1)*dhi(2)*dhi(3));
}
```

Velocity updates

```
for idim in 1..NDim {
  forall (f, phire, phiim, psire, psiim) in zip(FreqD, phi[ReD], phi[ImD], psi[ReD], psi[ImD]) {
    var ki = index2k(f)(idim);
    psire = -ki*phiim;
    psiim = ki*phire;
  }
  grid.psi_c2r();
  psi /= Ntot;
  var psix = inverseCIC(real(32), psi, pos);
  // Store these in pmom
  forall (p1, psi1) in zip(vel, psix) do p1(idim) += fac*psi1;
  /* Debugging -- print mean and standard deviation
  writeln(idim);
  writeln((+ reduce psi[RealD])/Ntot);
  writeln(sqrt((+ reduce (psi[RealD]**2))/Ntot));*/
  writeln("Processed momenta in dimension...",idim);
}
```

NAS Multigrid example

```
MGStencil={-1..1,
           -1..1,
           -1..1},
```

Exercise stencil calculations
Uses StencilDist
Thanks to Ben Harshbarger, Brad Chamberlain

```
// Stencil convolutions
inline proc stencilConvolve(dest : [?Dom] real, src : []real, w : coeff,
    param inc : bool = false, param stride : int=1) {
  var w3d : [MGStencil] real;
  [(i,j,k) in MGStencil] w3d[i,j,k] = w[(i!=0) + (j!=0) + (k!=0)];

  const N1 = src.domain.dim(1).high + 1;
  src.updateFluff();
  // Do the actual stencil convolution
  forall ijk in Dom {
    var tmp = + reduce [off in MGStencil] (src[stride*ijk+off]*w3d[off]);
    if inc then
      dest[ijk] += tmp;
    else
      dest[ijk] = tmp;
  }
}
```

Elegant, but slow (> 10x slower than benchmark)

NAS Multigrid -- Speedup

```
var locdom = dest.localSubdomain();
var locdom2 = {locdom.dim(1),locdom.dim(2)},
    locdom3 = locdom.dim(3);
const klo = locdom3.low,
      khi = locdom3.high;
forall (i1,j1) in locdom2  {
  // Zero
  dest.localAccess[i1,j1,klo-1] = 0.0;
  dest.localAccess[i1,j1,khi+1] = 0.0;
  if !inc {
    [k1 in locdom3] dest.localAccess[i1,j1,k1] = 0.0;
  }
  for k1 in vectorizeOnly(klo..khi) {
    var val = src.localAccess[i1,j1,k1];
    var val1 = src.localAccess[i1+1,j1,k1]+src.localAccess[i1-1,j1,k1]+
      src.localAccess[i1,j1+1,k1]+src.localAccess[i1,j1-1,k1];
    var val2 = src.localAccess[i1+1,j1+1,k1]+src.localAccess[i1-1,j1+1,k1]+
      src.localAccess[i1+1,j1-1,k1]+src.localAccess[i1-1,j1-1,k1];
    dest[i1,j1,k1] += w0*val + w1*val1 + w2*val2;
    var tmp = w1*val + w2*val1 + w3*val2;
    dest.localAccess[i1,j1,k1-1] += tmp;
    dest.localAccess[i1,j1,k1+1] += tmp;
  }
  // Handle periodic boundary conditions
  dest.localAccess[i1,j1,klo] += dest.localAccess[i1,j1,khi+1];
  dest.localAccess[i1,j1,khi] += dest.localAccess[i1,j1,klo-1];
}
```

Within x3 of benchmark, both OpenMP and single thread. "Easy" to parallelize….

Interoperability is important

- Any new language must be able to interface with existing code
  - This is, in part, responsible for the success of Python – wrappers to existing C code
- Most such interfaces are too domain specific to be of general interest
  - These don't need to be general Chapel packages
- An FFI should be lightweight and easy for the end-user.
- Chapel has a compelling C story here.
- Some examples :
  - FFTW (Fourier Transforms – my first real introduction to Chapel)
  - GNU Scientific Library (GSL)
  - MPI

Interfacing to GSL

- GNU Scientific Library
- Collection of common numeric algorithms (special functions, interpolation, random numbers and distributions, integration, etc)
- Large package, many headers
- Chapel's "extern block" supports these natively (thanks to Michael Ferguson, who fixed a few issues remaining in 1.13)

The library provides a wide range of mathematical routines such as random number generators, special functions and least-squares fitting. There are over 1000 functions in total with an extensive test suite.

The complete range of subject areas covered by the library includes,

Complex Numbers	Roots of Polynomials
Special Functions	Vectors and Matrices
Permutations	Sorting
BLAS Support	Linear Algebra
Eigensystems	Fast Fourier Transforms
Quadrature	Random Numbers
Quasi-Random Sequences	Random Distributions
Statistics	Histograms
N-Tuples	Monte Carlo Integration
Simulated Annealing	Differential Equations
Interpolation	Numerical Differentiation
Chebyshev Approximation	Series Acceleration
Discrete Hankel Transforms	Root-Finding
Minimization	Least-Squares Fitting
Physical Constants	IEEE Floating-Point
Discrete Wavelet Transforms	Basis splines
Running Statistics	Sparse Matrices and Linear Algebra

http://www.gnu.org/software/gsl/

Interfacing to GSL

```
use SysCTypes;
require '-lgsl','-lgslcblas';

/* Put in the usual GSL includes here */
extern {
// Special functions
#include "gsl/gsl_sf.h"
// Constants
#include "gsl/gsl_const.h"
// Integration
#include "gsl/gsl_integration.h"
// Random numbers and distributions
#include "gsl/gsl_rng.h"
#include "gsl/gsl_randist.h"
#include "gsl/gsl_cdf.h"
// Interpolation
#include "gsl/gsl_interp.h"
#include "gsl/gsl_spline.h"
}
```

- The C-API is exposed (no better or worse than calls in C)
- Some calls can be a little verbose
- Not hard for the user to wrap as needed, to improve interfacing

```
// Special functions
// We try out both interfaces. Note that you need to explicitly
// use the c_ptrTo function in these cases.
var res : gsl_sf_result;
writeln(gsl_sf_erf(0.1));
// Note how the res structure is available to the user
var ret = gsl_sf_erf_e(0.1, c_ptrTo(res));
writeln(res);
writeln(new string(gsl_strerror(ret)));
```

A rough edge : callbacks into Chapel

A specific use case : integrating a function

```
// Note that, for this particular problem, it might have been
// simpler to just wrap everything in C. Also note that we need to
// write some boilerplate to actually pass things to C.
record Payload {
  var alpha : real;
}
export proc func(x : real, p : c_void_ptr) : real {
  var r = (p : c_ptr(Payload)).deref();
  return log(r.alpha*x)/sqrt(x);
}
extern {
#include <gsl/gsl_integration.h>

  double func(double,void*);
  static void call_qags(void* params, double a, double b, double epsabs, double epsrel, size_t limit,
      gsl_integration_workspace* wk, double *result, double *err)
  {
    gsl_function F;
    F.function = &func;
    F.params = params;
    gsl_integration_qags(&F, a,b,epsabs,epsrel,limit,wk,result,err);
  }
}
var wk = gsl_integration_workspace_alloc(1000);
var result, error: real(64);
var p = new Payload(1.0);
call_qags(c_ptrTo(p):c_void_ptr, 0, 1, 0, 1.e-07,1000,wk,c_ptrTo(result), c_ptrTo(error));
```

Chapel + MPI

- A large number of scientific/numerical packages are built off MPI
  - Chapel needs to interop with these
- Performance
  - Currently (and anecdotally), single locale programs run slower in multi-locale mode, even if minimal/no communication
  - Big hit for otherwise trivially parallelizable jobs
  - Use MPI to fix this
- Parallel programming idioms are often taught with MPI
  - Use Chapel for convenience/productivity
  - MPI for performance
- MPI 1.1 (mostly) support upcoming
  - Currently on master
  - Wrapper mostly auto-generated by a simple Python script + Python-C parser (pycparser : https://github.com/eliben/pycparser)
  - Currently designed for Chapel in single-locale mode
  - Hopefully, can be extended to Chapel in multi-locale mode
    - GASNet already allows for MPI interop

Chapel + MPI : Hello, Chapel!

```
use MPI;
use C_MPI; // Include the C-API, to reduce verbosity of the code.
```

The MPI module does the initialization; currently requires a call to MPI_Finalize().

```
proc hello() {
  /* Simple test of MPI initialization */
  writef("This is rank %i of %i processes saying Hello, Chapel!\n",worldRank, worldSize);
  MPI_Barrier(MPI_COMM_WORLD);
}
```

Chapel + MPI : Ring communication

```
/* Non-blocking communication in a ring */
proc ring() {

  var left = mod(worldRank-1, worldSize);
  var right = mod(worldRank+1, worldSize);
  var toleft : c_int = 1;
  var toright : c_int = 2;
  var fromleft = toright,
      fromright = toleft;


  var buf : [1..2]int(32);
  var requests : [1..4]MPI_Request;
  var status : [1..4]MPI_Status;

  MPI_Irecv(buf[1], 1, MPI_INT, left, fromleft, MPI_COMM_WORLD, requests[1]);
  MPI_Irecv(buf[2], 1, MPI_INT, right, fromright, MPI_COMM_WORLD, requests[2]);

  MPI_Isend(worldRank, 1, MPI_INT, left, toleft, MPI_COMM_WORLD, requests[3]);
  MPI_Isend(worldRank, 1, MPI_INT, right, toright, MPI_COMM_WORLD, requests[4]);

  MPI_Waitall(4, requests, status);

  writef("Rank %i recieved %i from the left, and %i from the right\n",worldRank, buf[1], buf[2]);
  MPI_Barrier(MPI_COMM_WORLD);
}
```

Chapel + MPI : More complicated

```
/* MPI make communicator */
proc test_newcomm() {
  var comm : MPI_Comm,
      ranks1 : [0..1]c_int = [0:c_int, 1:c_int],
      ranks2 : [0..1]c_int = [2:c_int, 3:c_int],
      sum : c_int,
      newrank : c_int,
      origgrp, newgrp : MPI_Group;

  MPI_Comm_group(MPI_COMM_WORLD, origgrp);
  if worldRank < 2 {
    MPI_Group_incl(origgrp, 2, ranks1[0], newgrp);
  } else {
    MPI_Group_incl(origgrp, 2, ranks2[0], newgrp);
  }
  MPI_Comm_create(MPI_COMM_WORLD, newgrp, comm);
  MPI_Allreduce(worldRank, sum, 1, MPI_INT, MPI_SUM, comm);

  MPI_Comm_rank(comm, newrank);

  writef("Rank = %i, new rank = %i, sum = %i\n",worldRank, newrank, sum);
  MPI_Barrier(MPI_COMM_WORLD);
}
```

Interactive Chapel?

- The challenge is often not implementation, but what to implement…
- Trial and error
- Interactivity is a good thing
  - Python/Mathematica/MatLab etc do this very well
  - Jupyter notebooks are becoming very popular
- Chapel needs an interactivity story
  - Cling : CERN's implementation of a C++ REPL, based of Clang/LLVM
  - Doesn't have to be pure Chapel
  - Eg. a maintained Python interface (this is the mode in which I use Python – interfacing into C, thanks to tools like Cython)
  - A Python interface could also ease people into Chapel
  - Easy access to Python package ecosystem

Tooling?

- Debugging/profiling using standard tools hard, because of the C translation
- Tedious to track down performance issues
  - I'd love to be able to quickly see where a program is spending most of its time in a semi-automated manner (i.e. not print statements)
  - Could be at a line/function level (for functions, need to handle inlining)
- Compiler is slow; error messages one at a time
- Rebuild the world from scratch each time around
- Chapel idioms
  - It's easy to write Chapel code like C, harder to determine what better idioms are.
  - Flag what idioms are currently slow, and how to optimize when necessary
    - Eg. When reduce works, when array accesses might be slow etc
  - Maybe time for a Chapel Cookbook!

Some final thoughts

- Chapel is fun to use…
- If I were the only person writing the code, I'd probably use Chapel a significant portion of time…
  - A year ago, that would not have been true
  - Missing interactivity, tooling…
  - Compiler speed
- The Chapel team has been wonderfully responsive -- thanks!