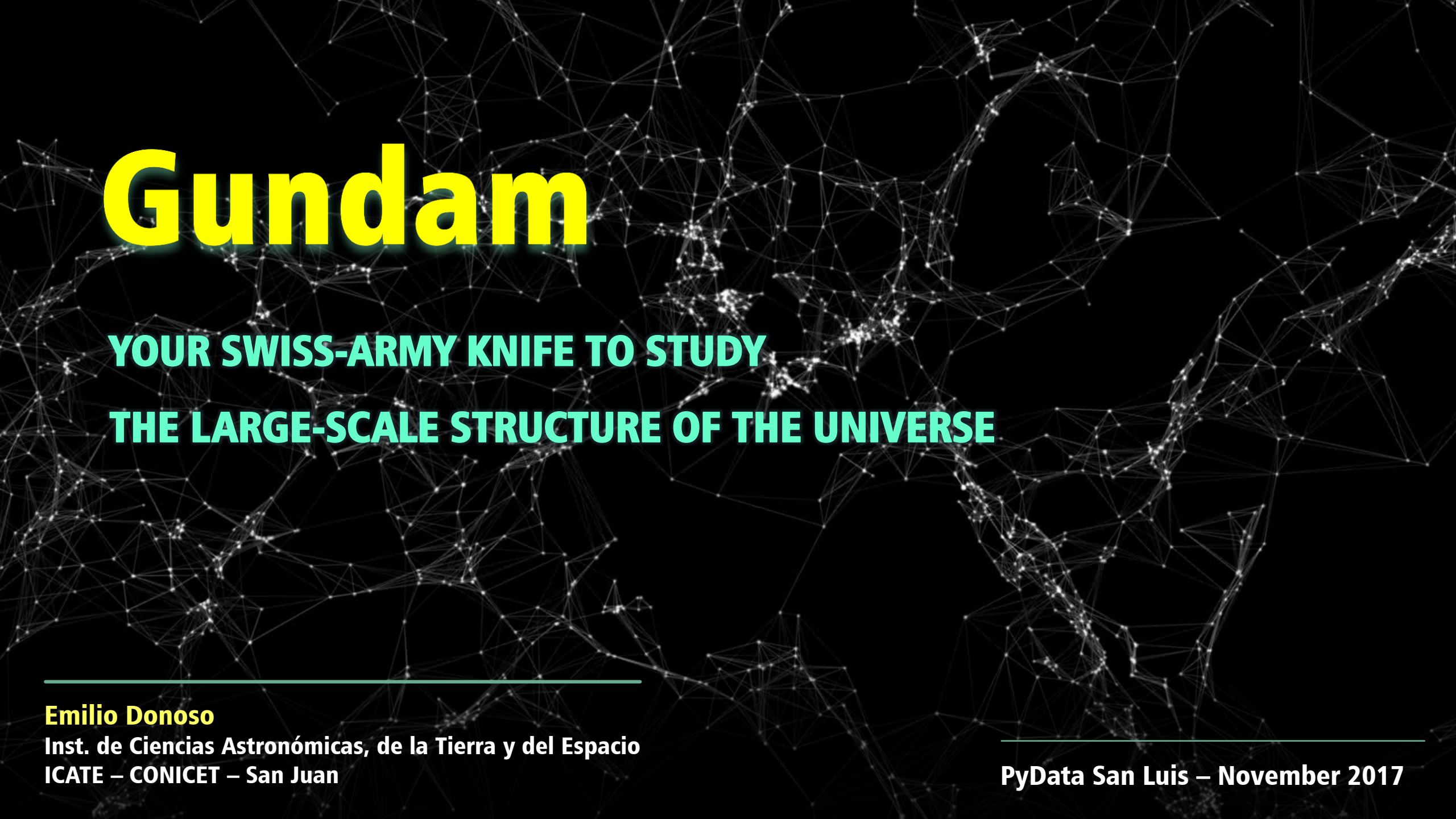


Gundam

YOUR SWISS-ARMY KNIFE TO STUDY THE LARGE-SCALE STRUCTURE OF THE UNIVERSE

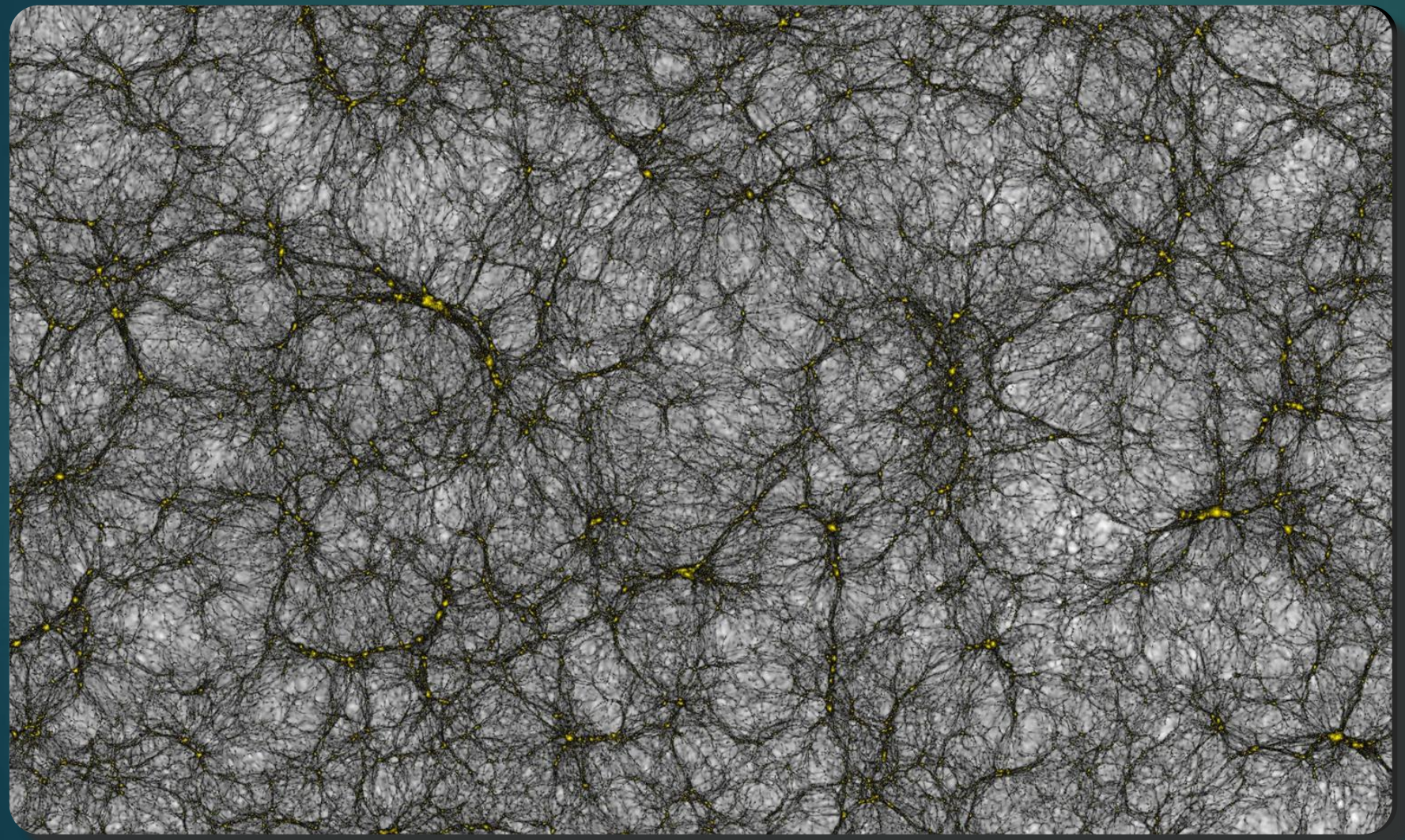
Emilio Donoso
Inst. de Ciencias Astronómicas, de la Tierra y del Espacio
ICATE – CONICET – San Juan

PyData San Luis – November 2017

Agenda

1. Why do we care?
2. Two point correlation function (2PFC). Counting galaxies is not easy
3. Speed things with skip-list / linked list algorithm
4. Need for Speed 1
5. Need for Speed 2
6. Need for Speed 3
7. Gundam in Action

Why matter ?



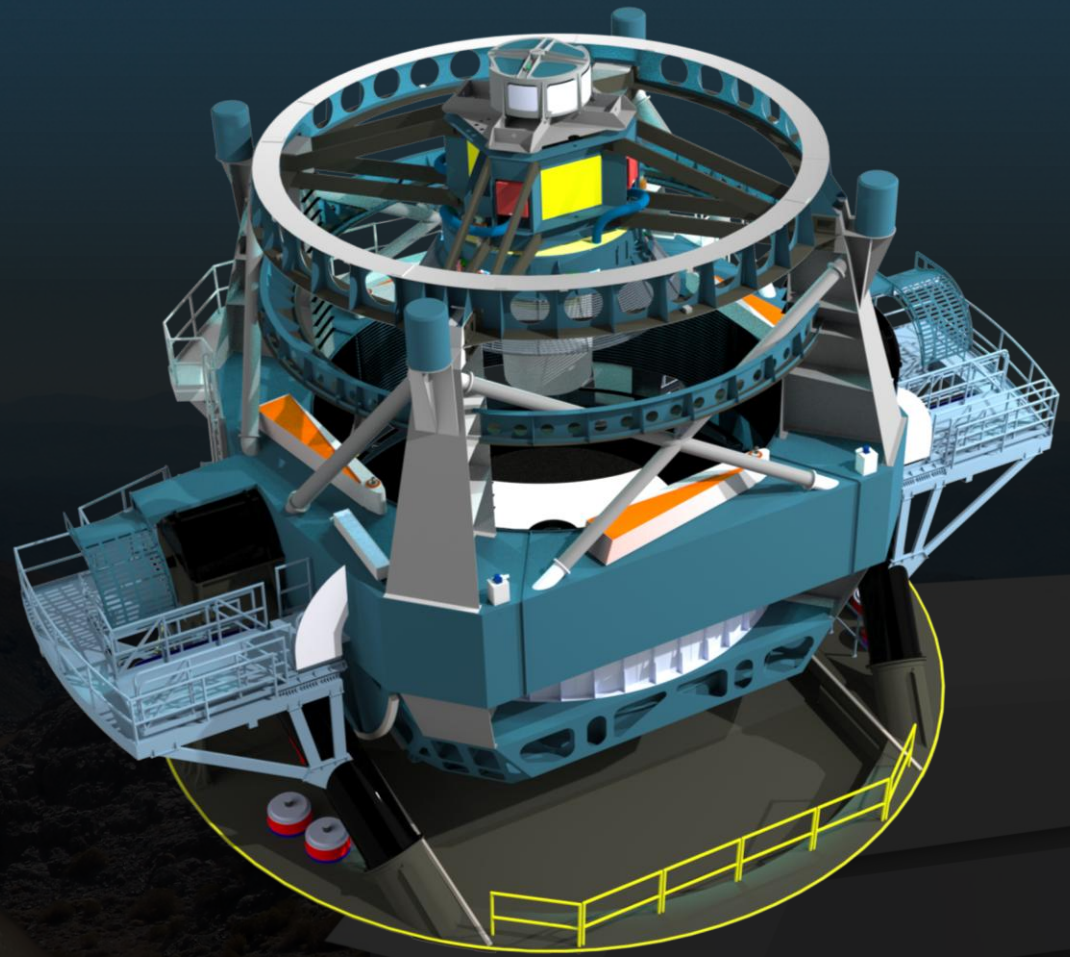
Euclid Flagship Simulation – 2 trillion particles

Redshift surveys show the Universe is largely inhomogeneous, forming structures like filaments, chains and walls

These strucures are in turn build from galaxy clusters and smaller groups

Why matter ?

LSST (Large Synoptic Survey Telescope)



- 8.4 meter mirror
- 3200 megapixel camera
- 10 year survey of the sky
- 1000 pairs of exposures & 15 Terabytes every night

37 billion stars and galaxies !

The Two-Point Correlation Function

$$dP = n[1 + \xi(r)]dV$$

$$\xi(r) = \frac{DD(r)}{RR(r)} - 1$$

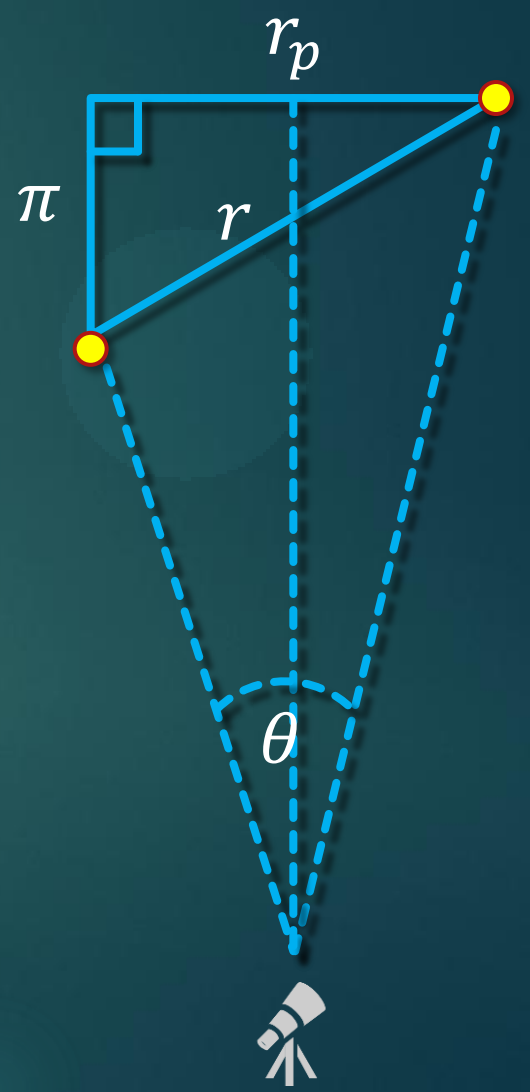
2PT Correlation Function

$$w_p(r_p) = \int_0^{\pi_{max}} \frac{DD(r_p,\pi)}{RR(r_p,\pi)} d\pi - 1$$

Projected 2PT Correlation Function

$$w(\theta) = \frac{DD(\theta)}{RR(\theta)} - 1$$

2PT Correlation Function



Plenty of other estimators w/ various statistical properties

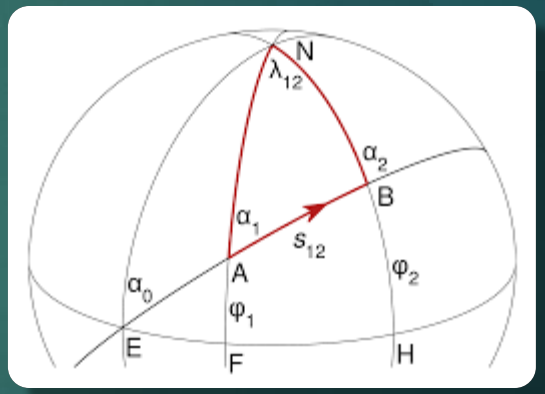
$$\xi_{Ham}(r) = \frac{DD(r) \cdot RR(r)}{[DR(r)]^2} - 1$$

$$\xi_{LS}(r) = \frac{DD(r) - 2DR(r) + RR(r)}{RR(r)}$$

Why matter?

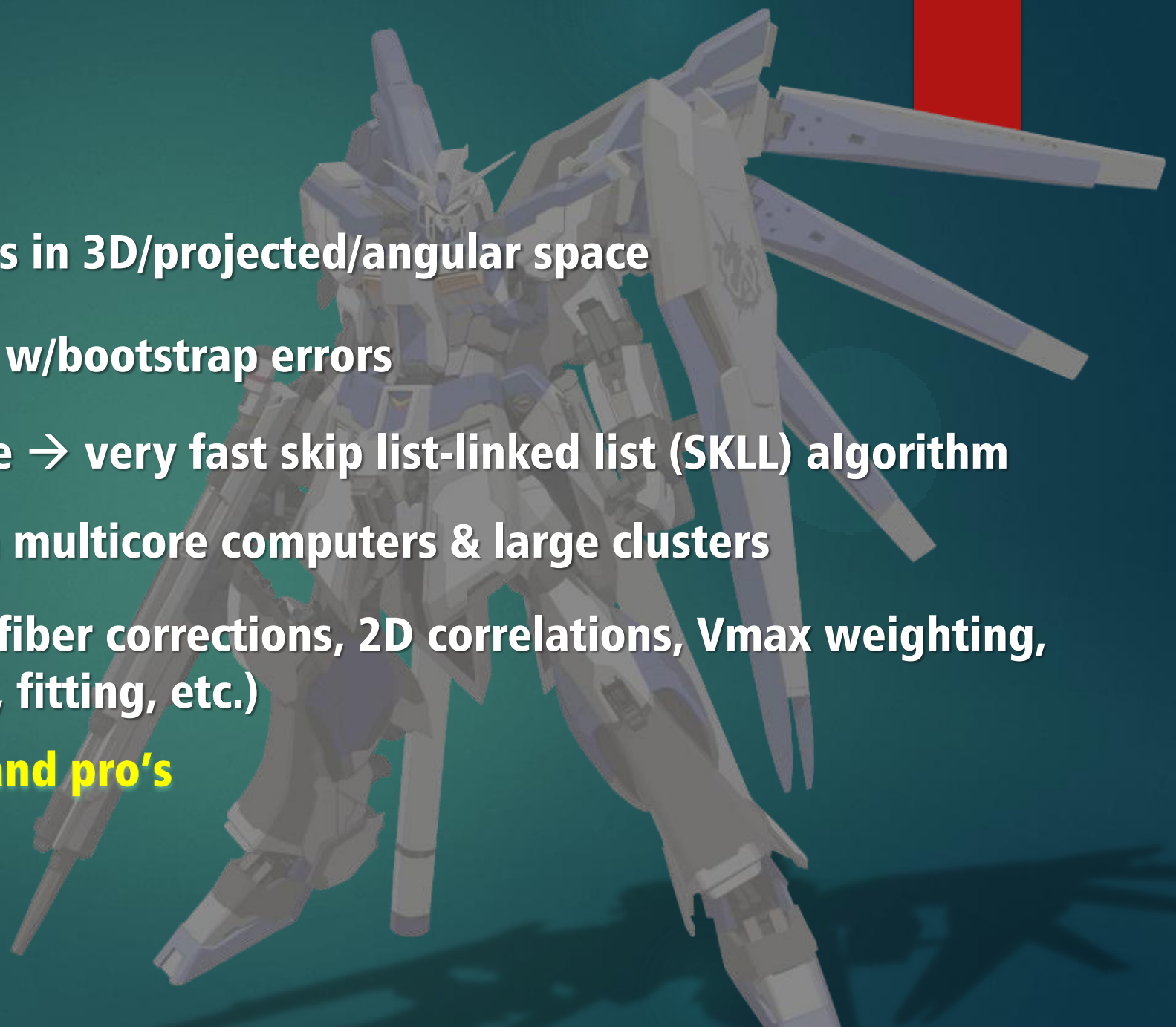
- To reduce shot noise → random samples are ~10-20x the data size
- Naïve CF compute time scales as $O(N^2)$

```
FOR i=1,n
  FOR j=1,n
    calc_distance(i,j)
    bin_distance()
  END
END
```



$$\cos(\theta_{ij}) = \sin(\delta_i)\sin(\delta_j) + \cos(\delta_i)\cos(\delta_j)\sin(\varphi_i - \varphi_j)$$

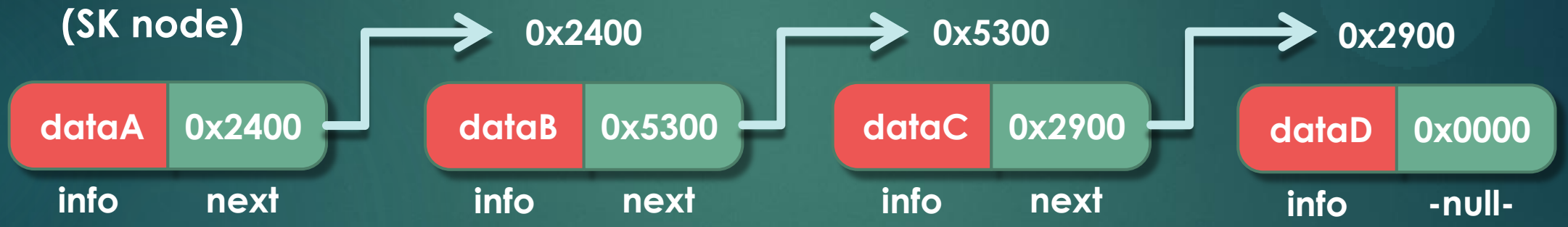
Gundam



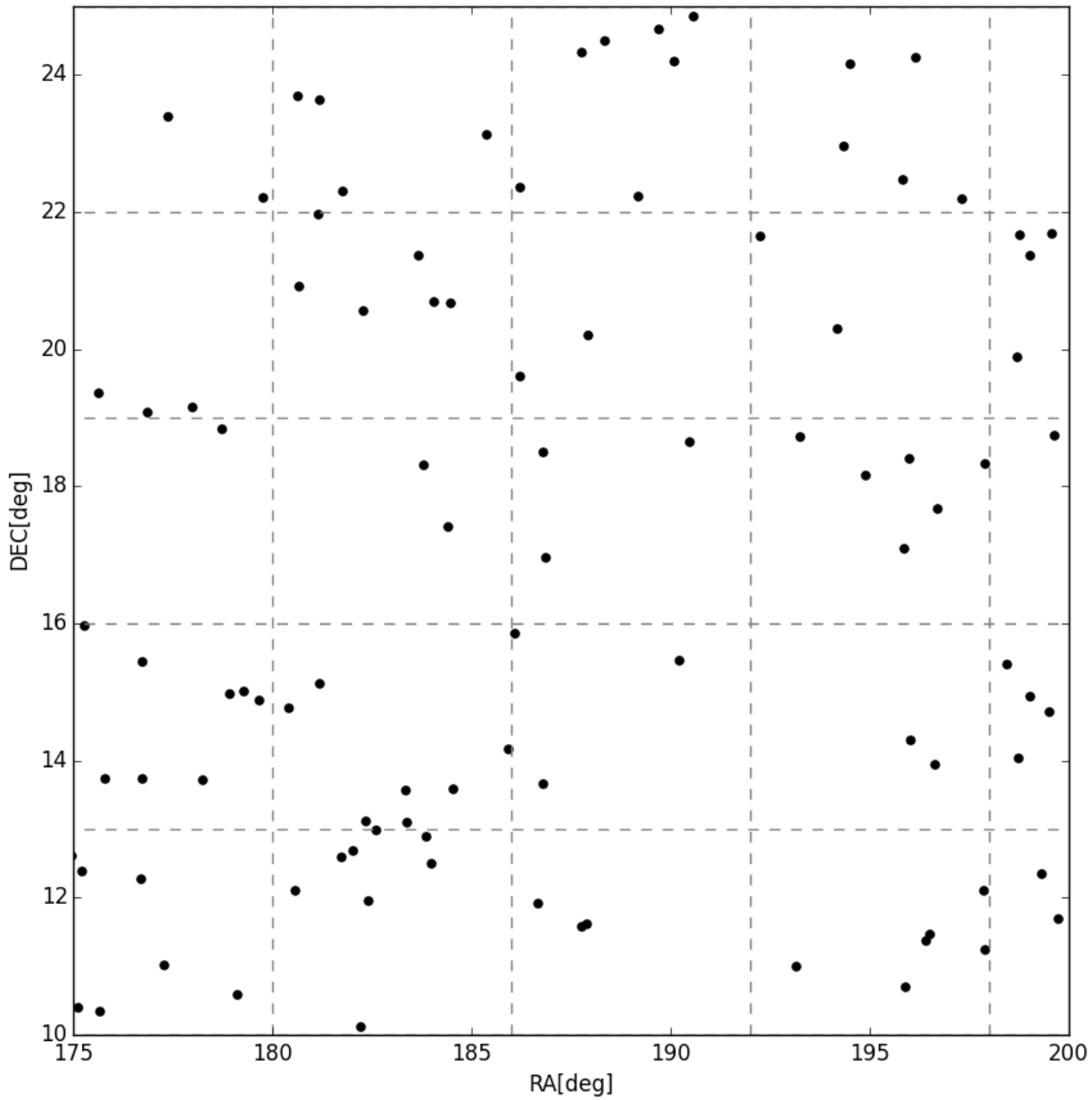
- 2-point correlation functions in 3D/projected/angular space
- Auto and cross-correlations w/bootstrap errors
- Mixed Python+Fortran code → very fast skip list-linked list (SKLL) algorithm
- Automatic paralellization in multicore computers & large clusters
- Tons of extra functionality (fiber corrections, 2D correlations, Vmax weighting, user-defined weights, plots, fitting, etc.)
- **Designed for both, noob's and pro's**

The Bowels of Gundam

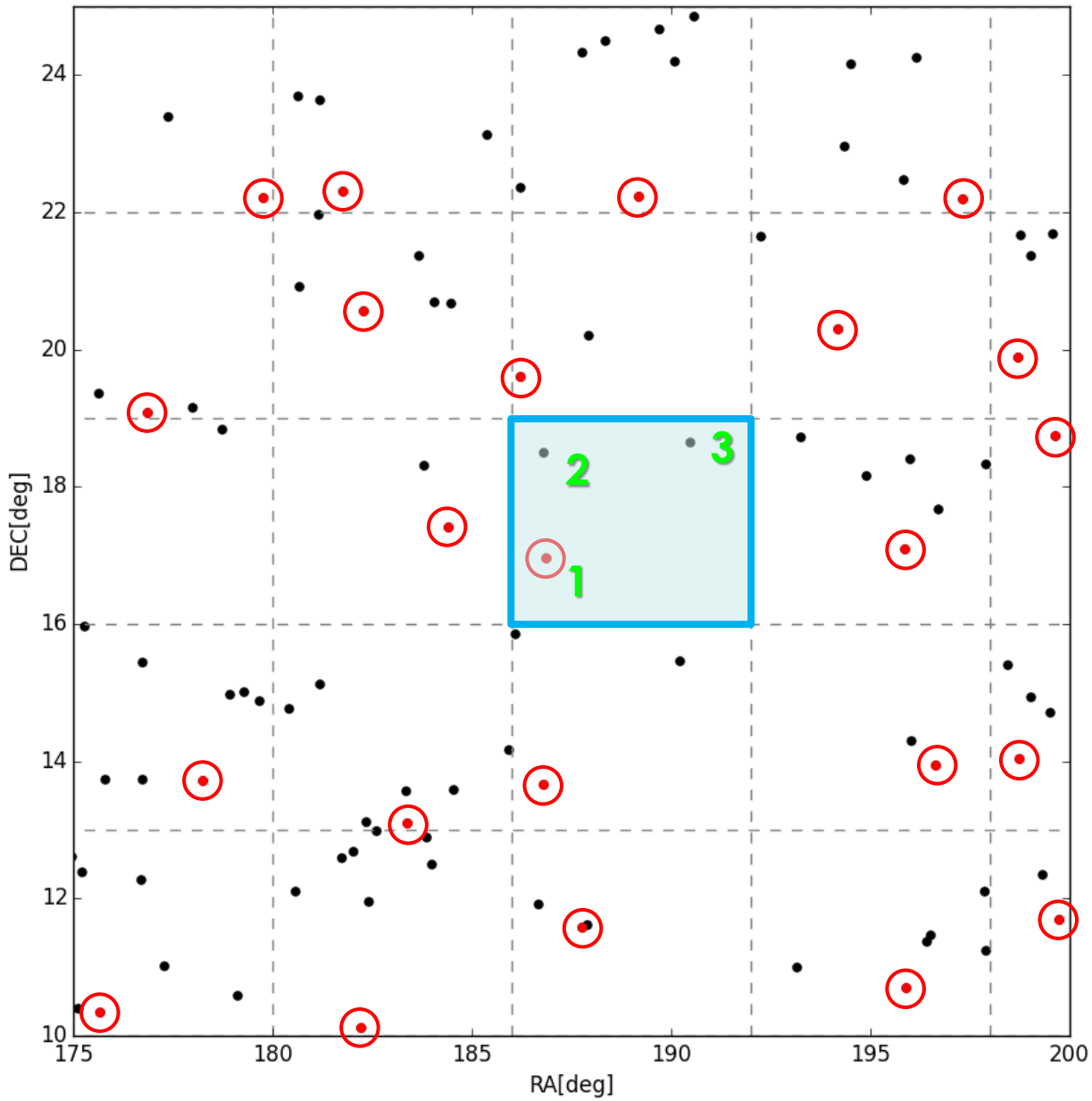
Skip List / Linked List Algorithm → fast-traverse neighboring galaxies & avoid unnecessary distance computations



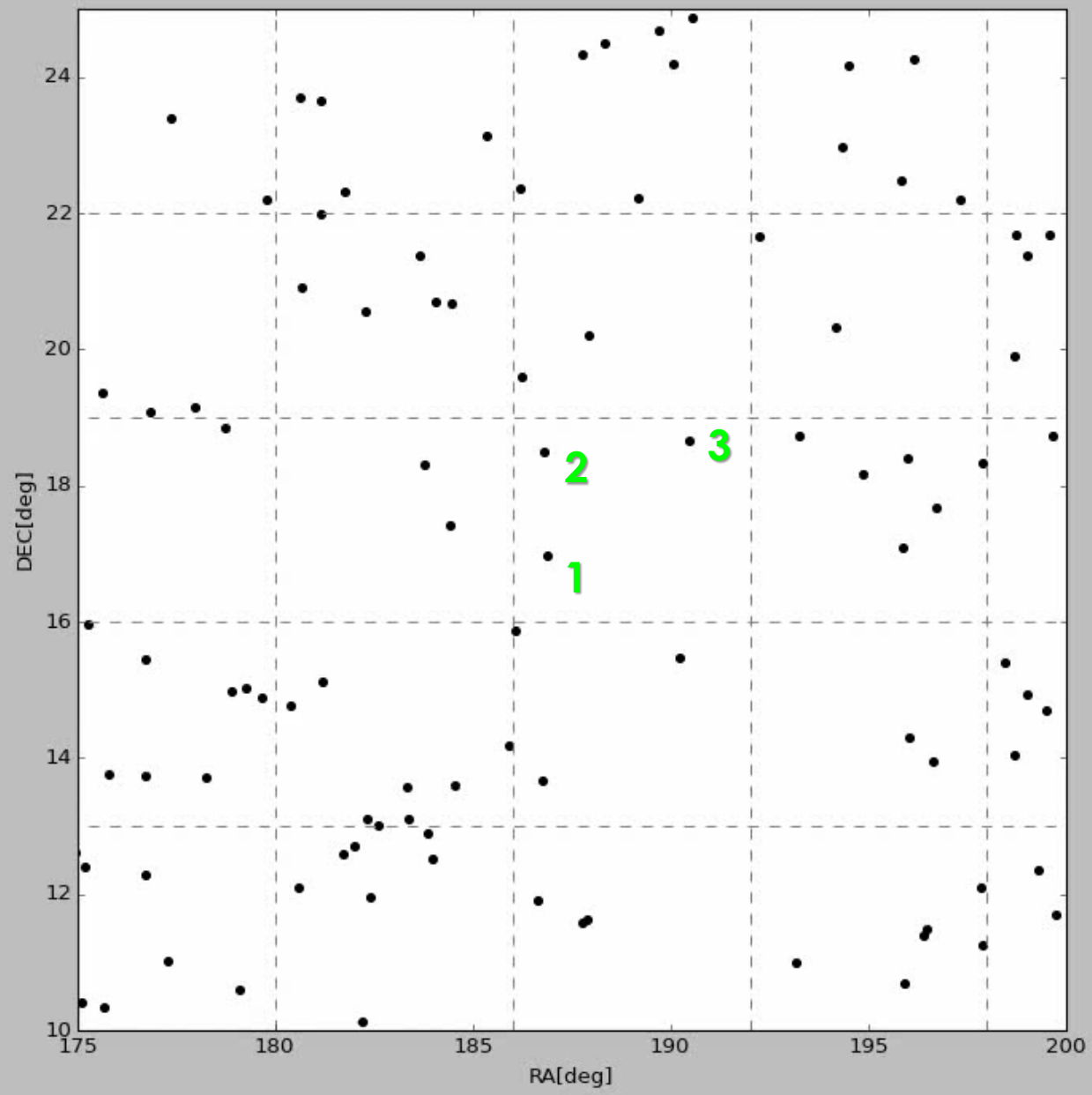
The Bowels of Gundam



The Bowels of Gundam



The Bowels of Gundam



Need For Speed 1 : Call the Dos Equis Man


```
      SUBROUTINE ZADD(A,B,C,N)
C
      DOUBLE COMPLEX A(*)
      DOUBLE COMPLEX B(*)
      DOUBLE COMPLEX C(*)
      INTEGER N
      DO 20 J = 1, N
         C(J) = A(J)+B(J)
 20   CONTINUE
      END
```

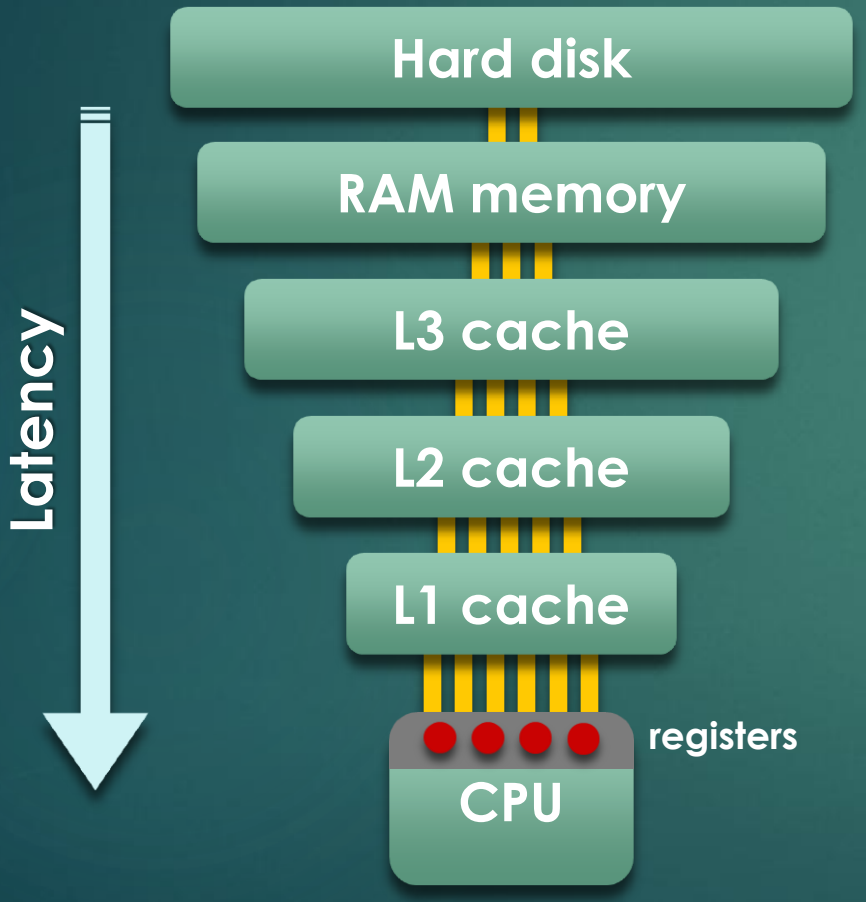
f2py

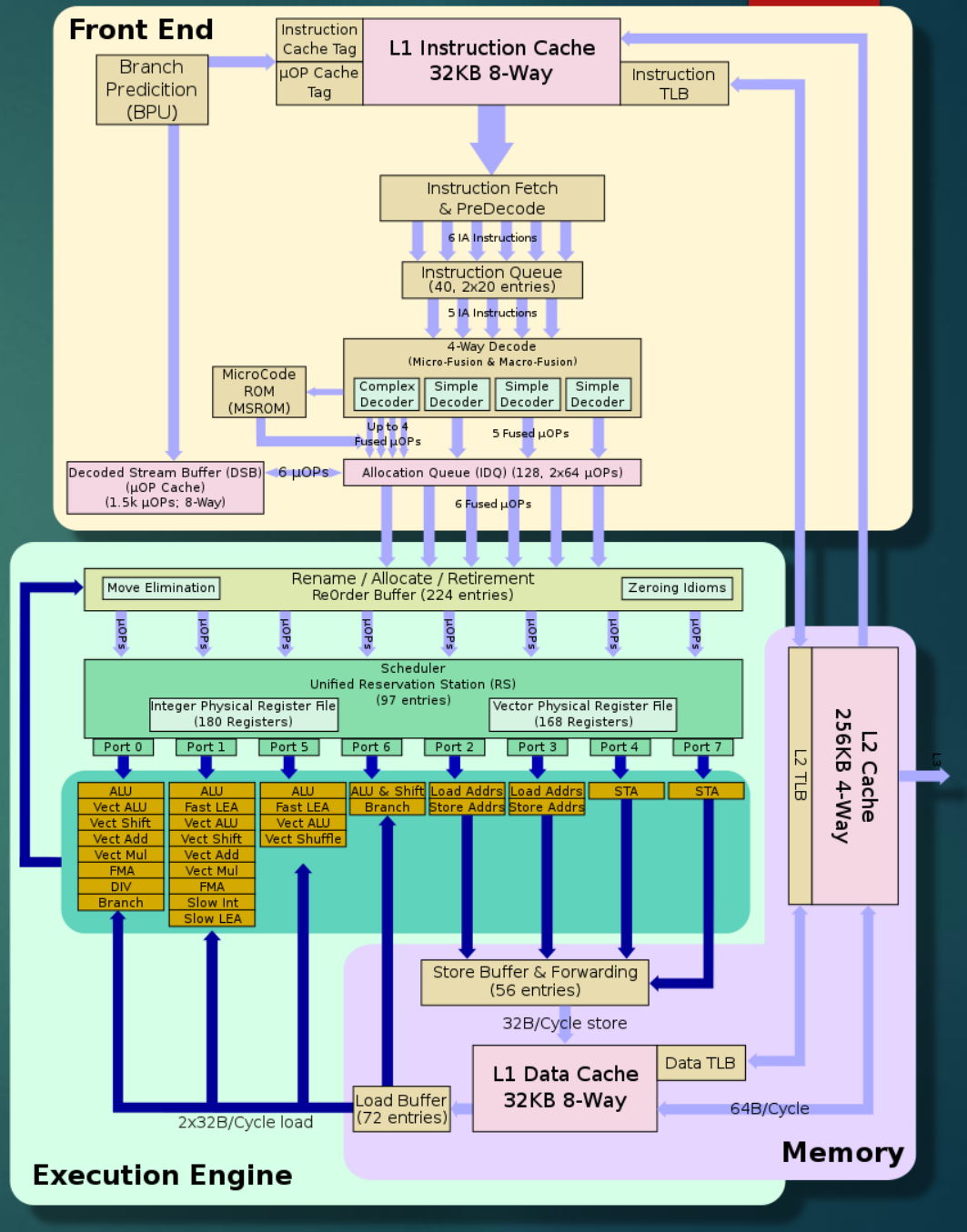
(add.so)

python console

```
> import add
> r = add.zadd([1,2,3],[1,0,4],3)
> r
  array([2,2,7])
```

Need for Speed 2: Ca(t)ching stuff

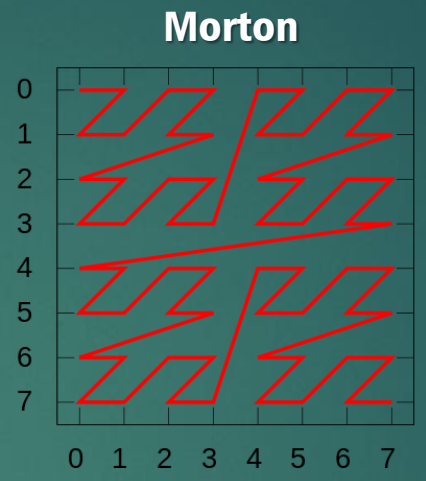


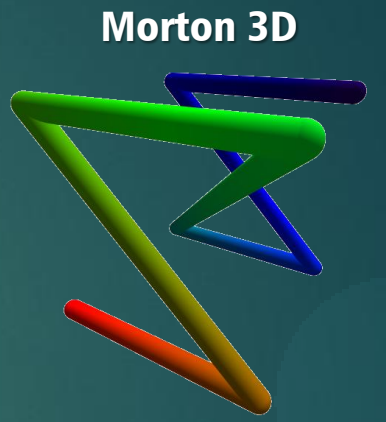


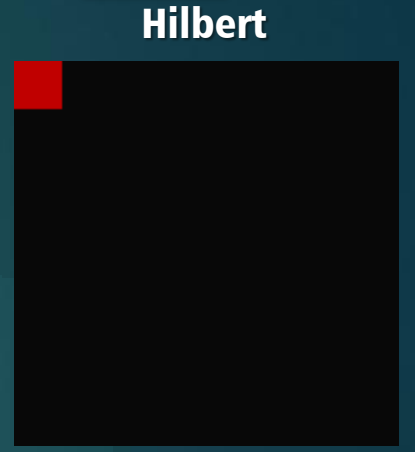
Please save the cache!

Linked lists like to kill CPU caches !

Gundam uses a Morton-like data ordering to map nearby structure into computer memory

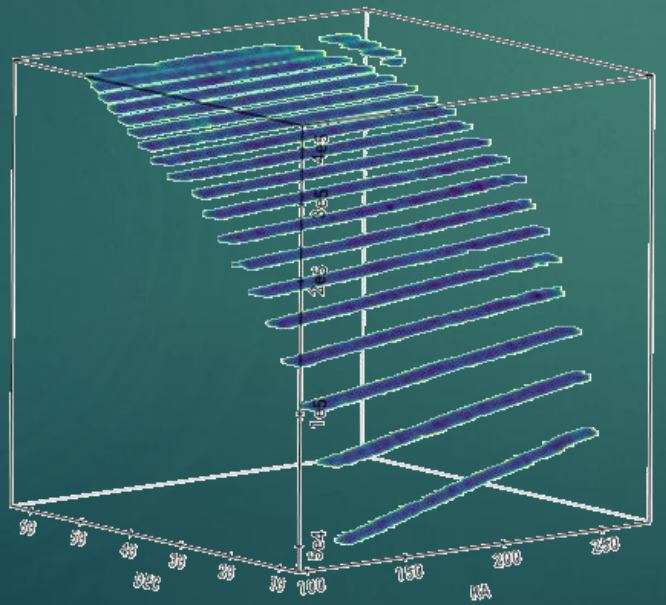


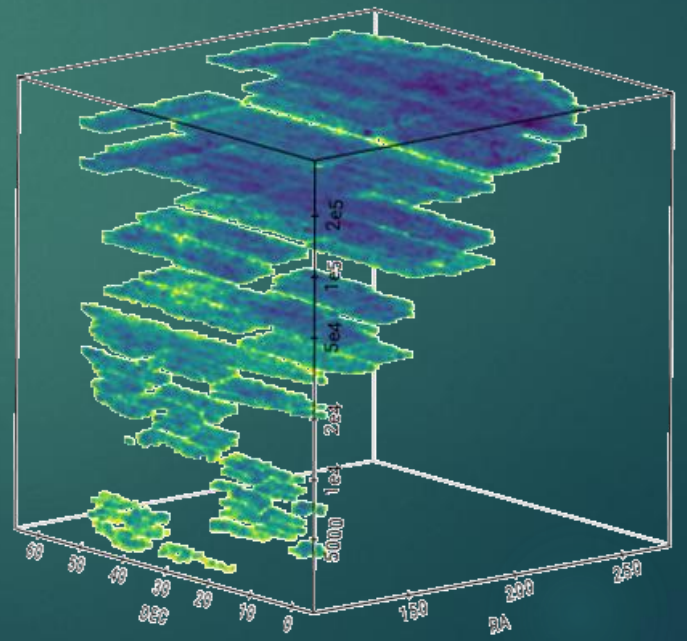




Natural Sorting

(real SDSS data)

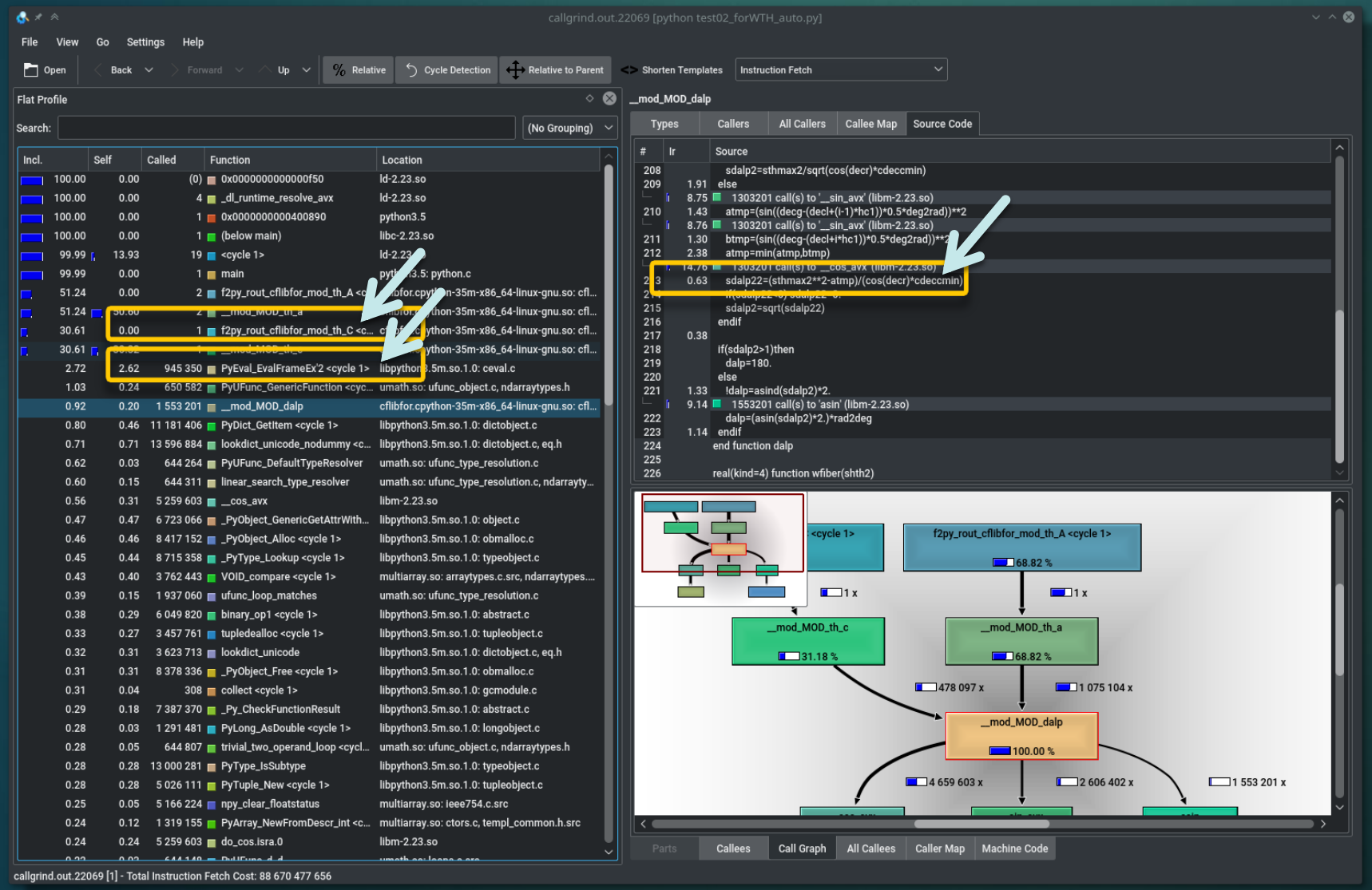




Morton Sorting

(real SDSS data)

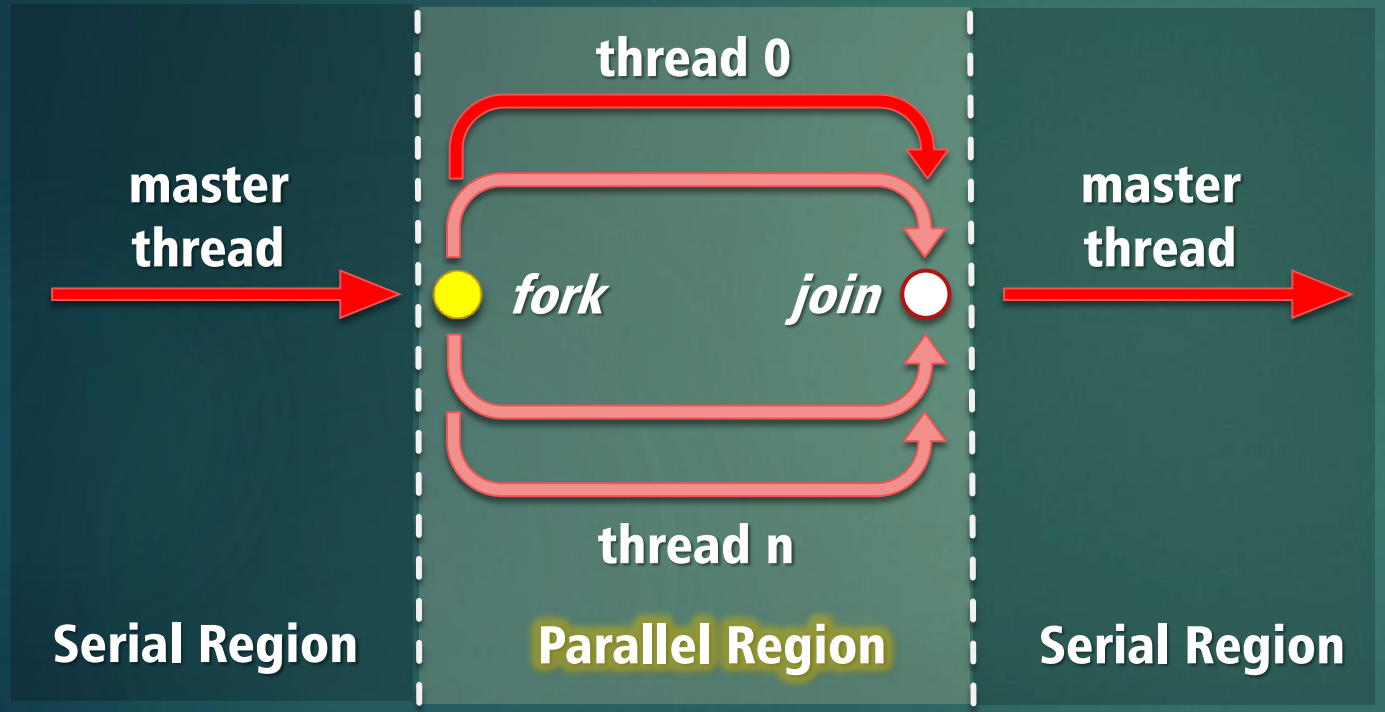
Line-by-Line Profiling with Valgrind



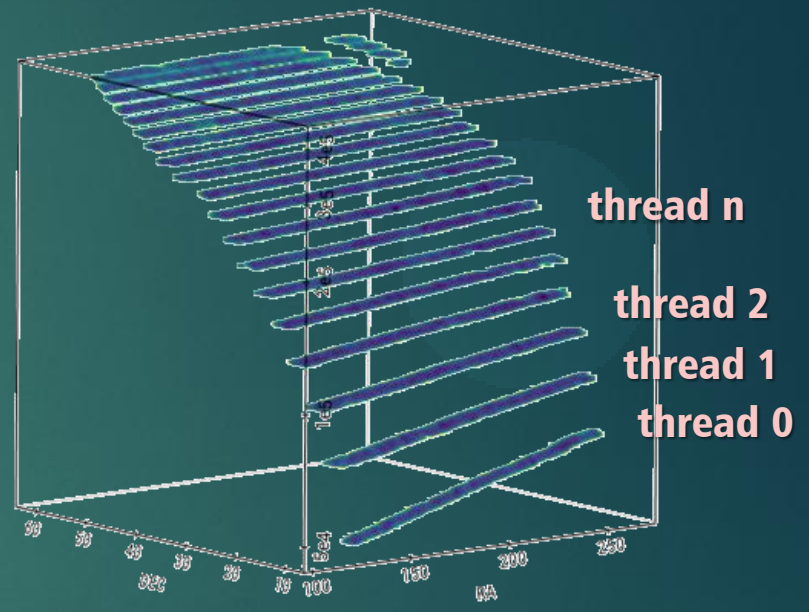
Valgrind measures line-by-line performance and cache hit/miss ratios

Need For Speed 3 : think parallel !

OpenMP
Enabling HPC since 1997



Each stripe scheduled to be processed by a given thread



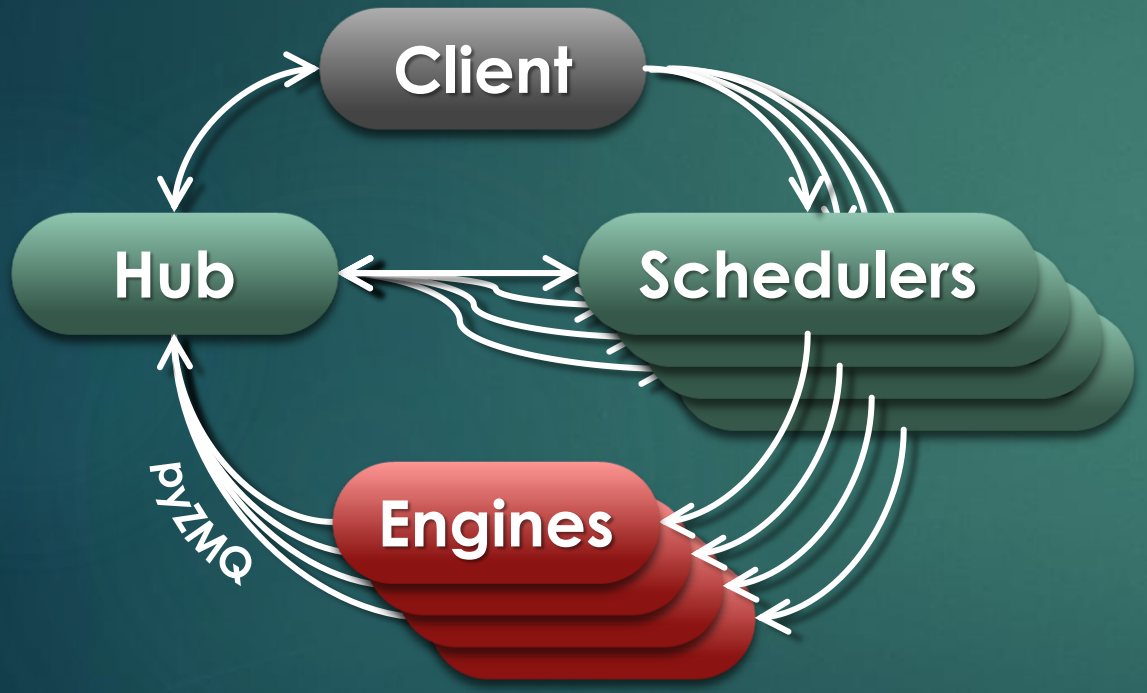
Guided scheduling to handle uneven workload

iteration number

Need For Speed 3 : think parallel again !

ipyparallel library

Min Ragan-Kelley, UC Berkeley




```
:~> ipcluster start -n 4
```

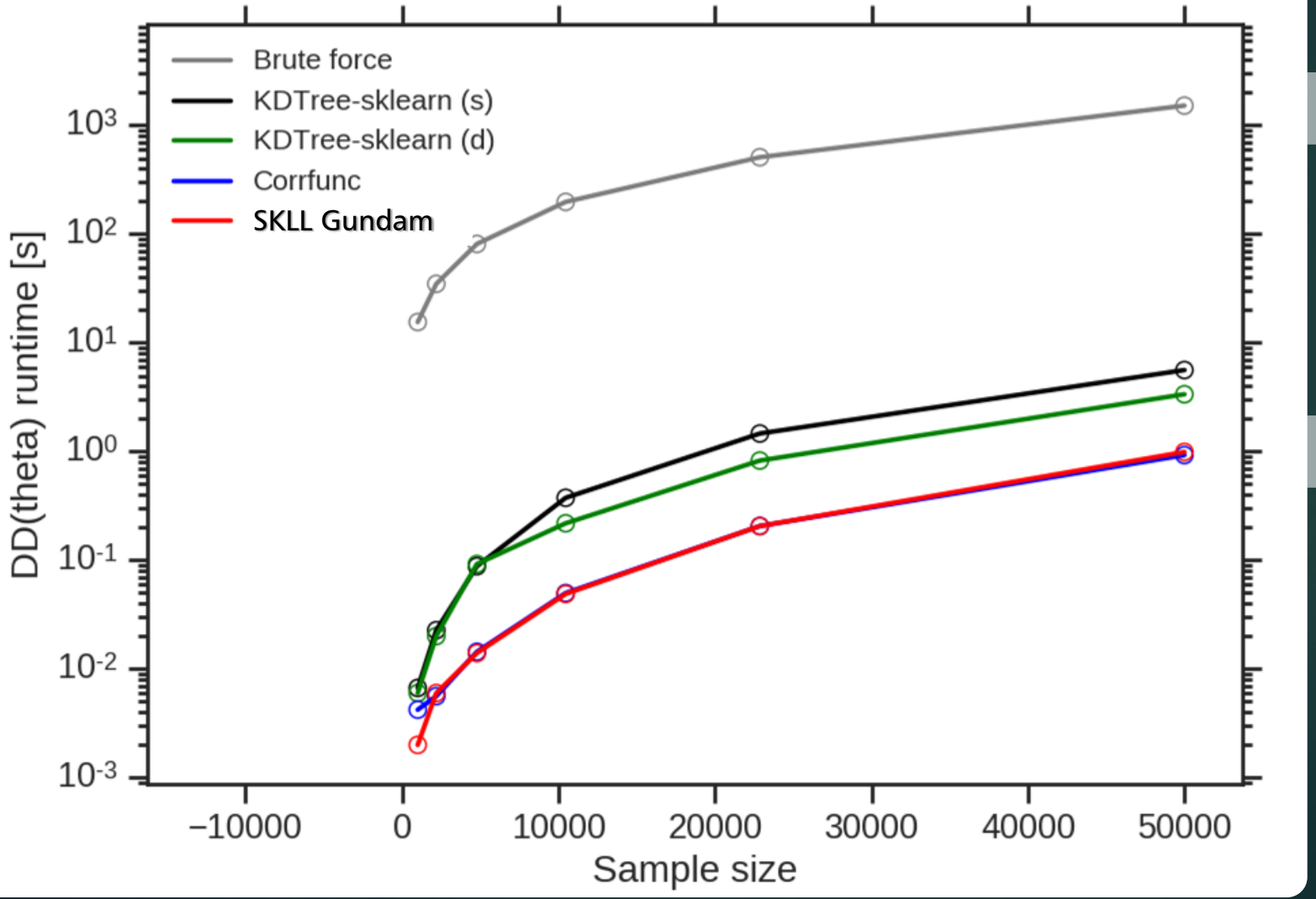
```
> import ipyparallel as ipp
> rc = ipp.Client()
> dv = rc[:]

> def square(x):
      return x**2

> r = dv.map_async(square,[1,2,3,4,5,6])
> r
  [1,4,9,16,25,36]
```

How fast does this work?
(does it?)

Gundam Athletics

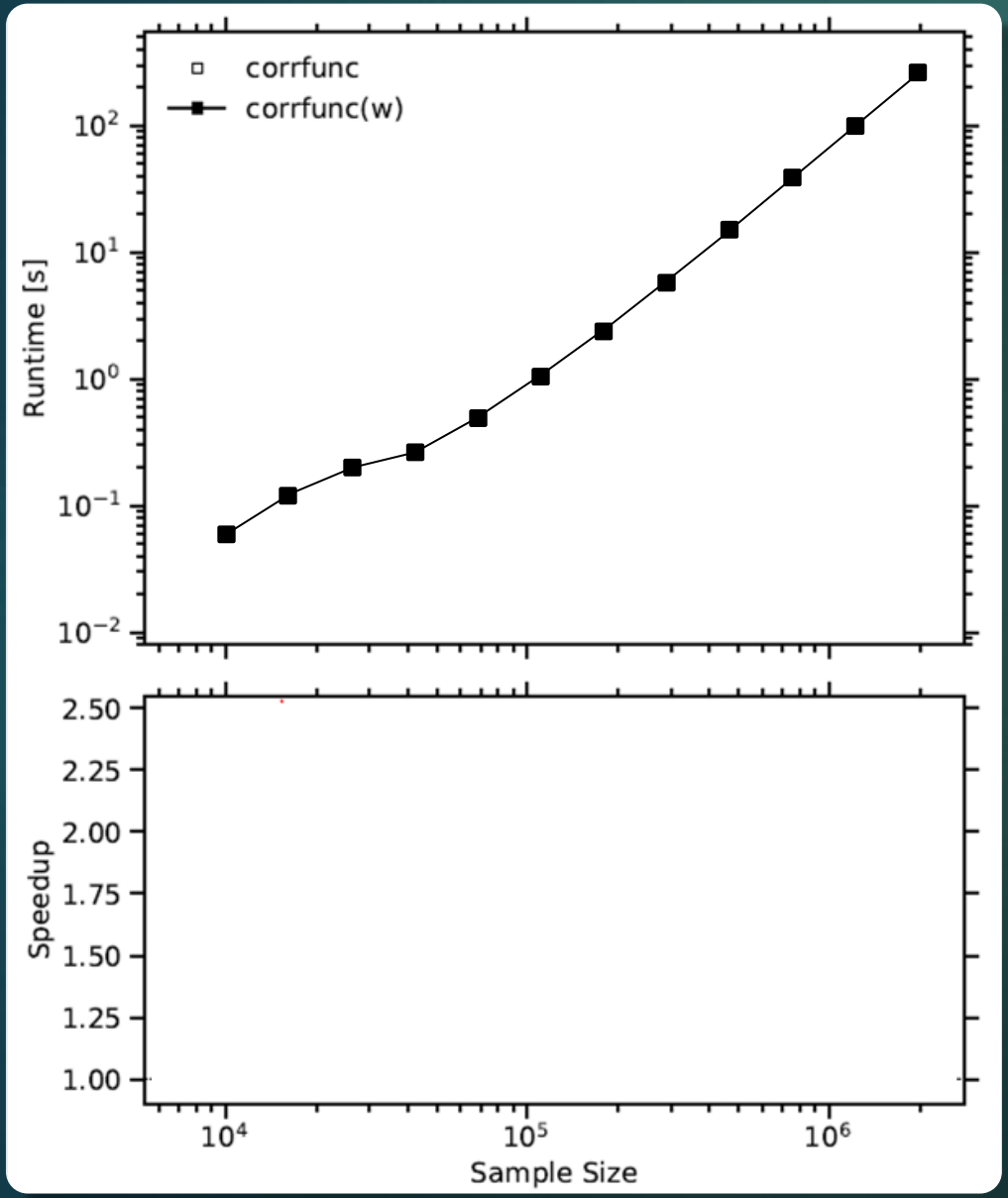


~32 min. !

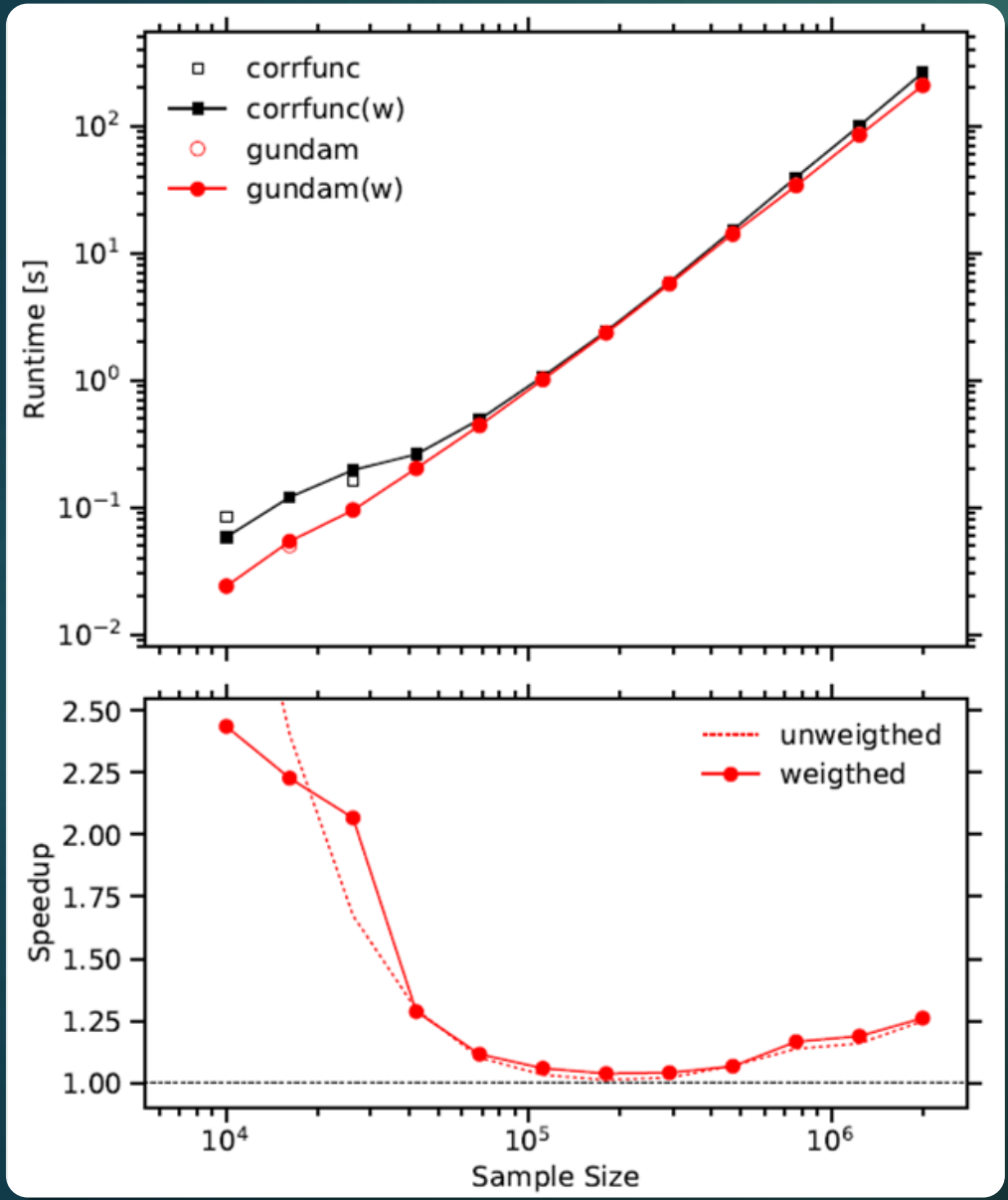
< 1 sec !

Gundam is slightly faster than many options

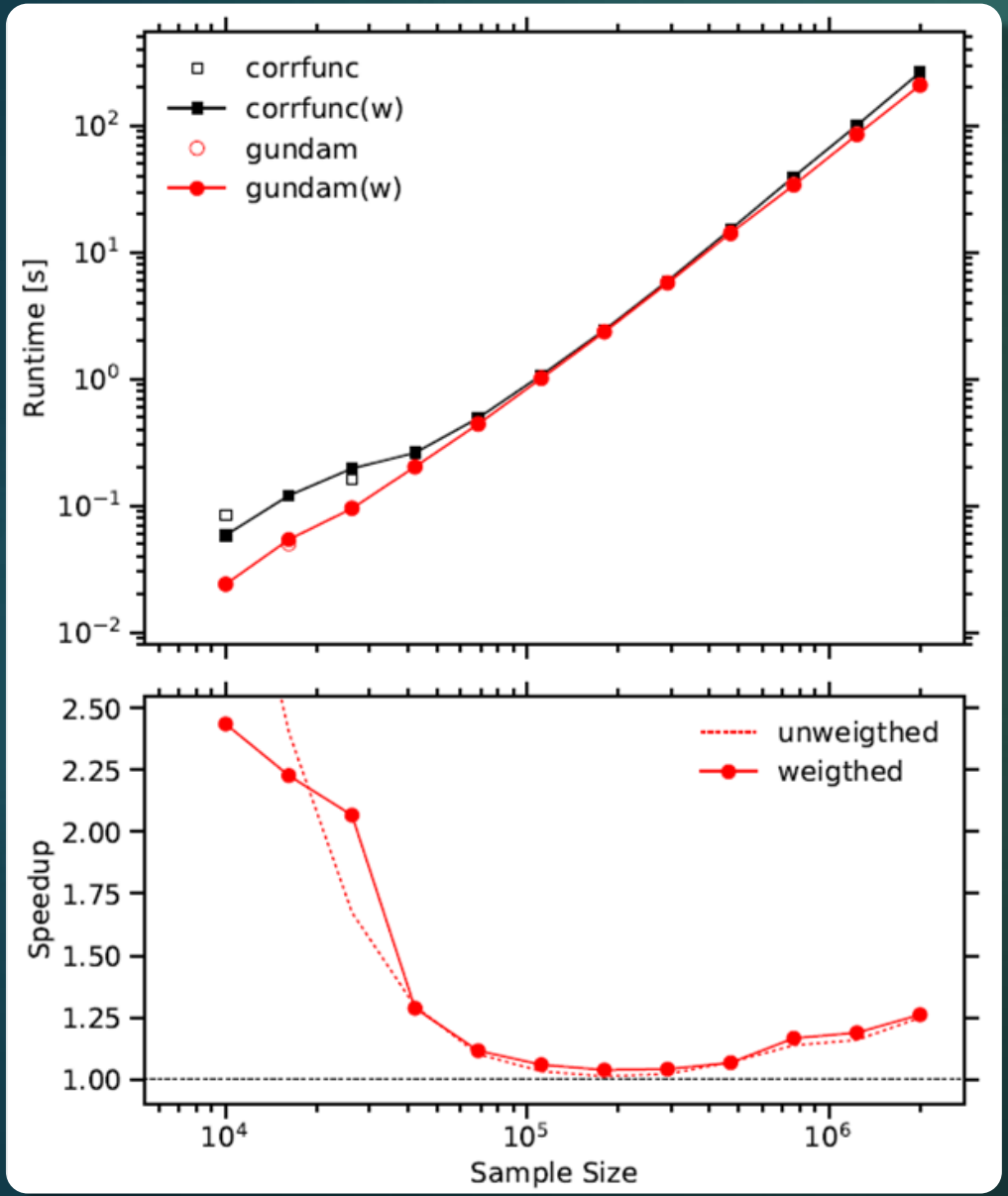
Gundam Athletics

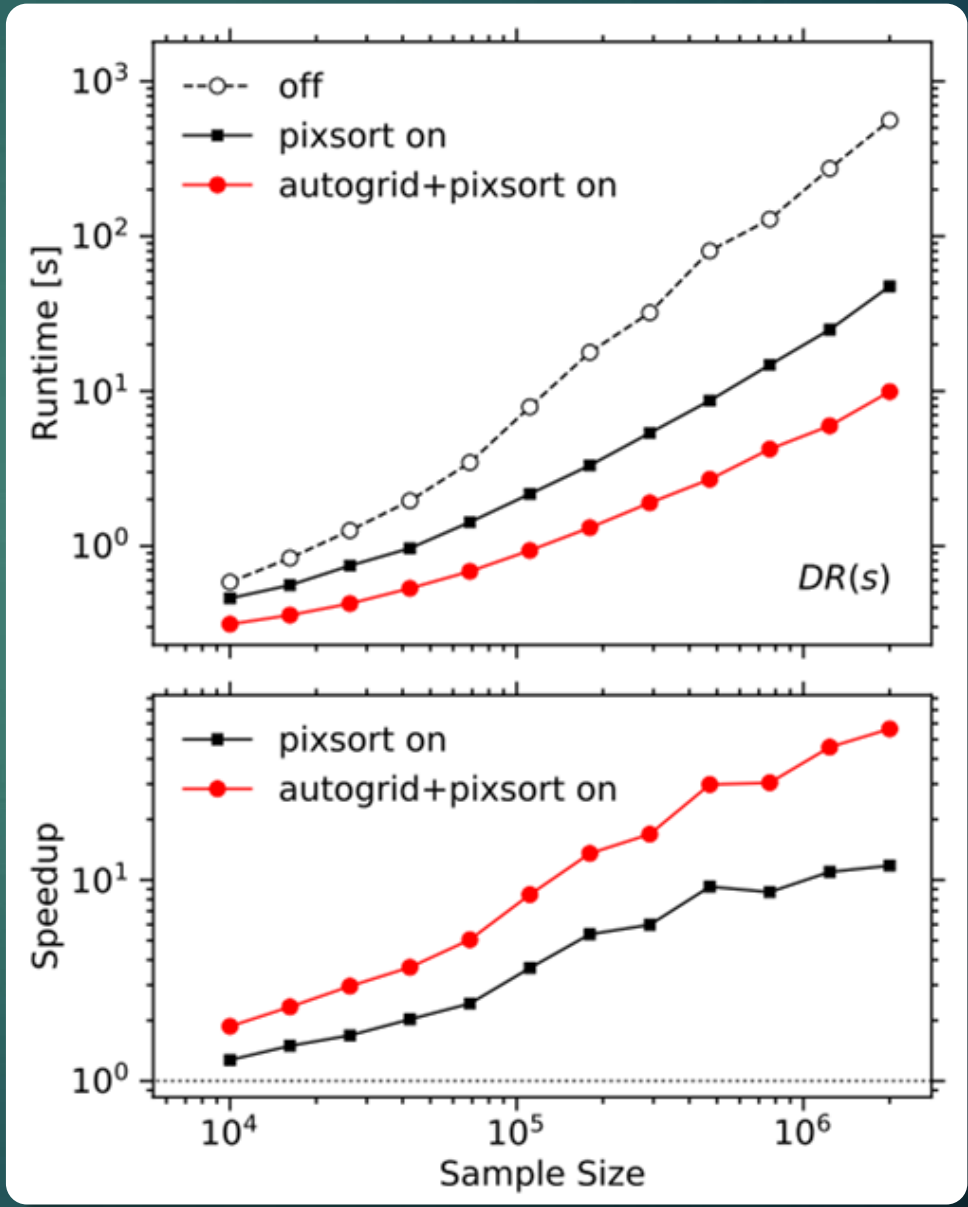


Gundam Athletics

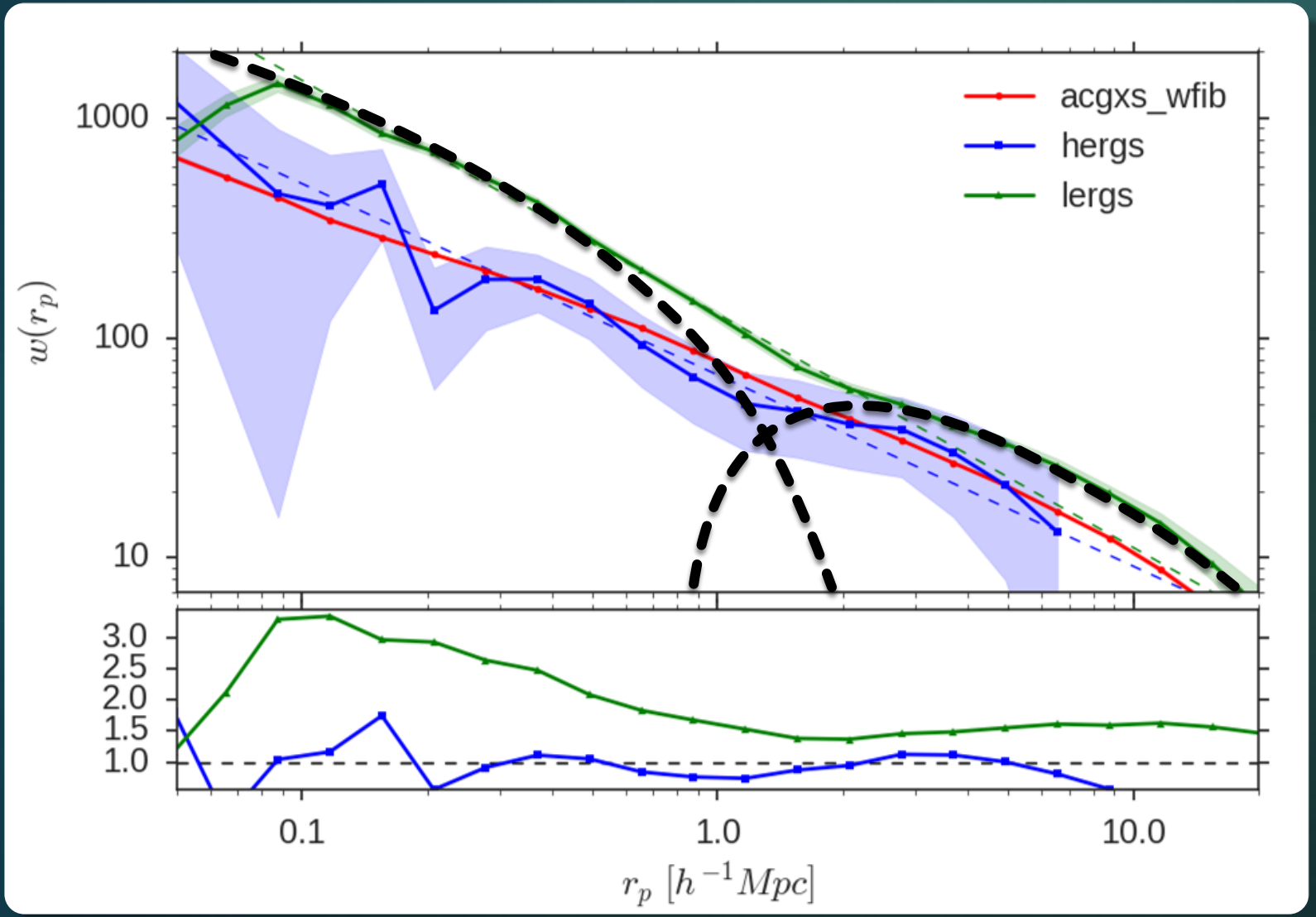


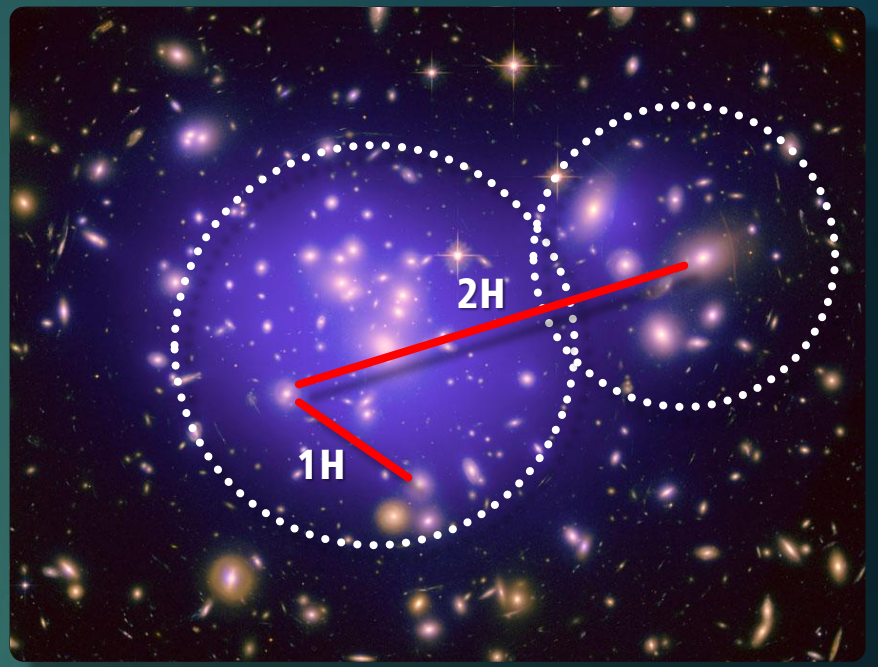
Gundam Athletics



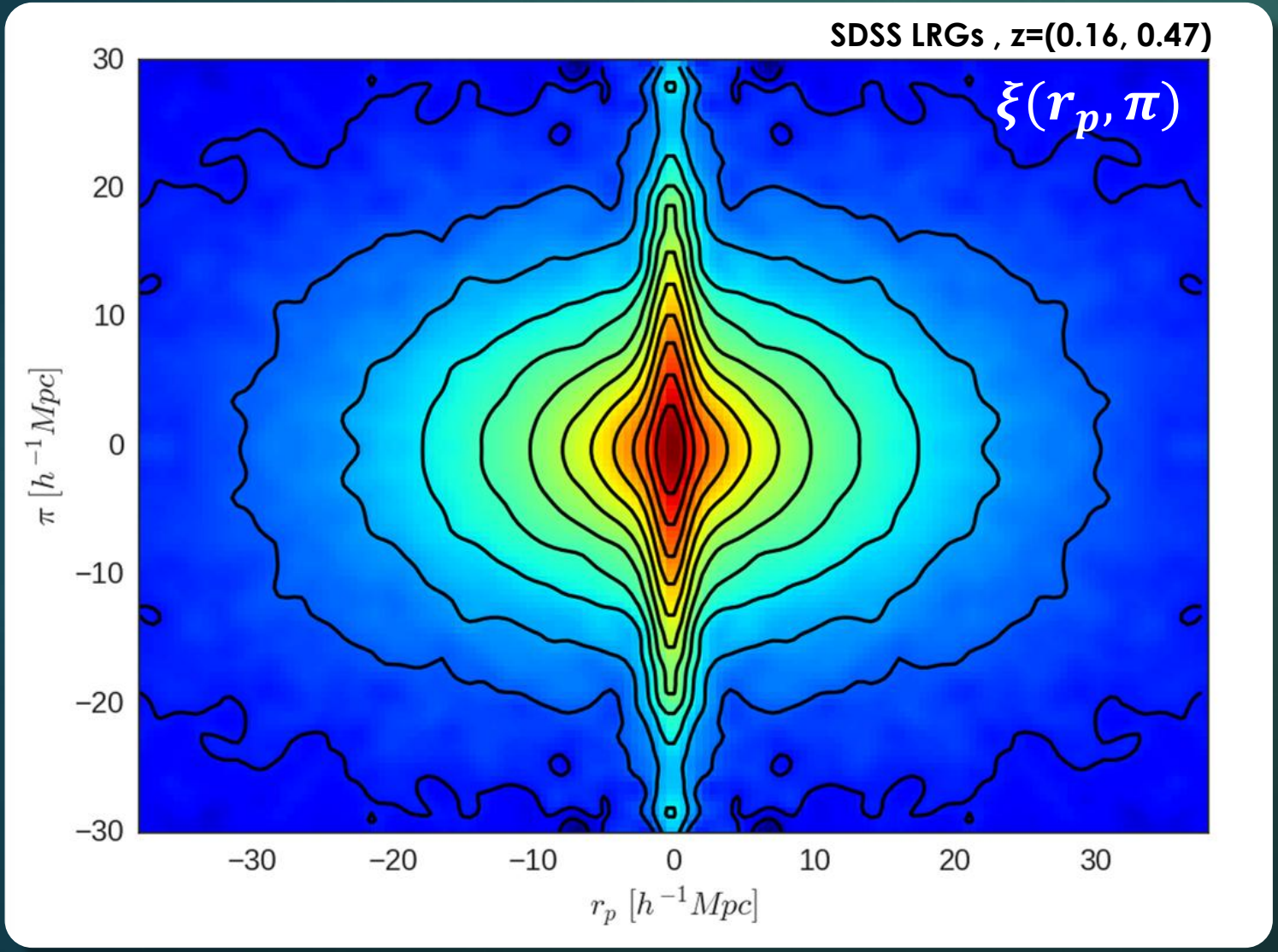


Blazing Fast 1-D Correlations





Blazing Fast 2-D Redshift Distortions



100k LRGs – 1M random galaxies

Gundam can do this in ~29s and 2 lines of code !

```
# read LRG & RAN samples first

> cnt = pcf(LRG, RAN, params, estimator='LS')

> cntplot2d(cnt, estimator='LS')
```

Gundam Implementation

Layer 1 (python)

```
pcf(d,r,pars…)
pccf(d,r,c,pars…)
acf(d,r,pars…)
accf(d,r,c,pars…)
…
```

Layer 2 (python)

```
tpcf_wrp(…)
makebins(…)
packpars(…)
plotcf(…)
comparecf(…)
cntplot2d(…)
pairs_auto(…)
…
```

Layer 3 (fortran)

```
skl12d(…)
thA(…)
thC(…)
skl13d(…)
rppi_A(…)
rppi_Ab(…)
rppi_A_wg(…)
rppi_C(…)
bootstrap(…)
…
```

All 3 levels fully documented and with examples

Gundam in Action

```
1  import gundam as gun

   # DEFINE INPUT PARAMETERS  ========================================
2  pars          = gun.packpars(kind='pcf')
   pars.h0       = 100    # Hubble constante [km/s/Mpc]
   pars.nsepp    = 28     # number of bins of projected separation rp
   pars.dsepp    = 0.125  # bin size of rp (in log space)
   pars.seppmin  = 0.01   # minimum rp in Mpc/h

   # READ DATA FILES  ================================================
3  gals = Table.read('./data/gxs.fits')       # galaxy sample
   rans = Table.read('./data/gxsran.fits')    # random sample

   # DO CORRELATION  =================================================
4  cnt = gun.pcf(gals, rans, pars, estimator='LS', write=True, plot=True, nthreads=48)
```

Gundam in Action

```
1  import gundam as gun

   # DEFINE INPUT PARAMETERS  ====================================
2  pars          = gun.packpars(kind='pcf')
   pars.h0       = 100    # Hubble constante [km/s/Mpc]
   pars.nsepp    = 28     # number of bins of projected separation rp
   pars.dsepp    = 0.125  # bin size of rp (in log space)
   pars.seppmin  = 0.01   # minimum rp in Mpc/h

   # READ DATA FILE
3  gals = Table.re
   rans = Table.re

   # DO CORRELATION
4  cnt = gun.pcf(g
```

Python dictionary with dot-style access

Pickled to a *.cnt* file on disk you can read back later, email, share, etc.

- Correlations and errors
- All the counts
- All the bins
- All parameters
- Entire log of the run
- Description

All this took no more than 4 lines !

The Road Ahead…

Gundam has grown to ~10.000 lines of code+doc

- Optimize LL code to be more cache friendly
- Vectorize LL code with AVX intrinsics ? Keep dreaming
- Adapt to run in large clusters and diverse HPC frameworks (e.g. dask)
- Finish implementing Vmax, corr. matrix, jackknife errors, etc.
- Learn Git and publish the code in Github → Open Science!


THANK
YOU
GRACIAS
ARIGATO
SHUKURIA
BOLZÏN
MERCI
JUSPAXAR
DANKSCHEEN
BÏYAN
SHUKRIA
TINGKI
YAQHANYELAY
TASHAKKUR ATU
SUKSAMA
EKHMET
MEHRBANI
GRAZIE
KOMAPSUMNIDA
MAAKE
GOZAIMASHITA
EFCHARISTO
PALDIES
MINMONCHAR
MAKETAU
SIKOMO
EKOJU
HÄTUR
GUI
UNALCHÉESH
SPASIBO
DENKAUJA
NENACHALHYA
MERSI
ATTO
ANIHA
DHANYABAAD
WABEEJA
MAITEKA
HUI
YUSPAGARÀTAM
CHALTU
NUHUN
SNACHALHUYA
SPASSIBO
SANCO
MERASTAWHY
GAEJTHO
LAH
AGUYJE
FAKAAUE
TAVTAPUCH
MEDAWAGSE
BAIKA

Further questions ?

edonoso@conicet.gov.ar

emiliodon@gmail.com

github.com/samotracio