CosmoSIS:
Modular cosmological parameter estimation

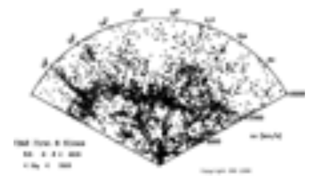
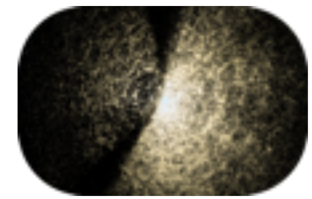
https://bitbucket.org/joezuntz/cosmosis/wiki/Home

Alessandro Manzotti
Sarah Bridle, Scott Dodelson, Elise Jennings,
Jim Kowalkowski, Marc Paterno,
Doug Rudd, Saba Sehrish, Joe Zuntz

CHEP 2015

KICP Kavli Institute for Cosmological Physics at The University of Chicago
MANCHESTER 1824
Fermilab
THE UNIVERSITY OF CHICAGO Research Computing Center

COSMOLOGY IN THE ERA OF BIG DATA

	Past		Present-future
BIG DATASET	 15,000 galaxies	→	 > 1 Million
BIG COLLABORATION		→	
SOFTWARE	Codes developed individually, in different languages. The output is shared.	→	

COLLABORATIVE DEVELOPMENT DIFFERENCES

Cosmologists work **on their own** much more often

- much important code developed by very small groups
- **each** individual or **group** chooses **programming language**, tools, etc. (Python, Fortran, C are most common)
- no central management of software is possible
- collaboration is often informal

HEP collaborations have strong control over their process

- define **choice** of **programming language** (almost all C++)
- single framework used for most development
- centrally managed software
- requires strong control over member of the collaboration

CosmoSIS has to live **within** the **demands** of the **cosmology** community

NEXT GENERATION PARAMETERS ESTIMATION

See also:
Physics Analysis Software Framework for Belle II M. STARIC et al.

CosmoSIS:

Modular framework for parameter estimation.

Example: turn supernovae brightness into constraints on cosmological model parameters.

Consolidate and **connect** together **existing codes**

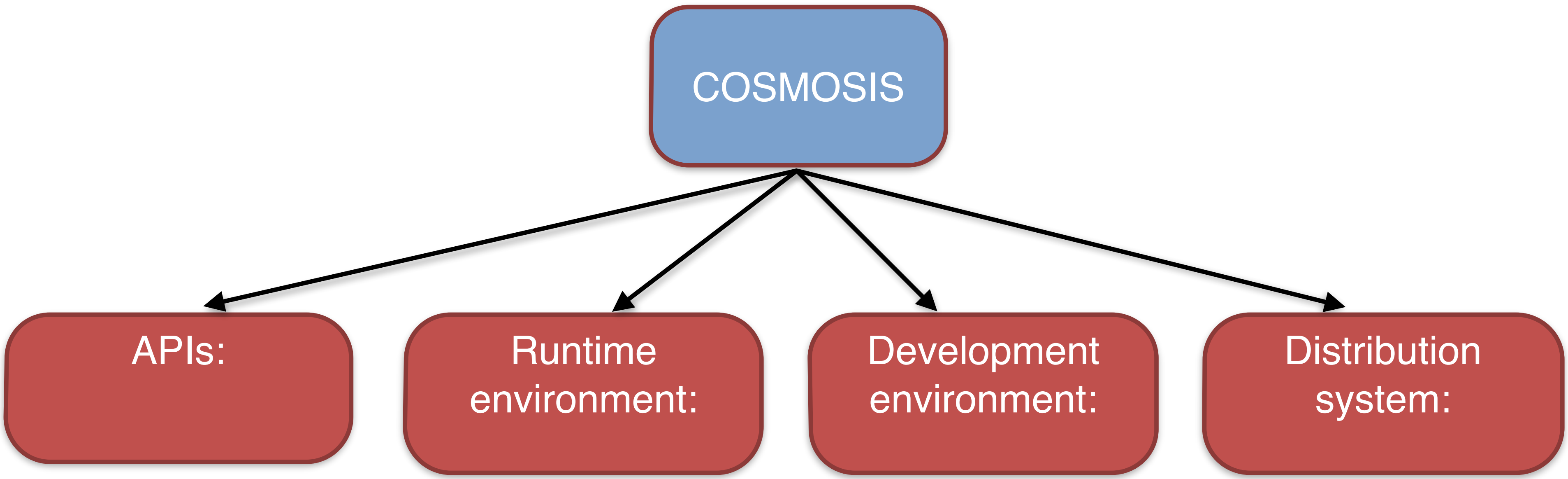
Enhance **collaboration**, and development of new algorithm in different languages

Make it **easy** to deploy, configure, and run **across all supported platforms**

Fast and able to run on HPC **cluster**

Nothing about the framework is **cosmology - specific**

OUR MODULAR SOLUTION: COSMOSIS



APIs: Define module configuration, how to obtain their input data, how they interact with the framework, how outputs are organized, and now new modular components can be added.

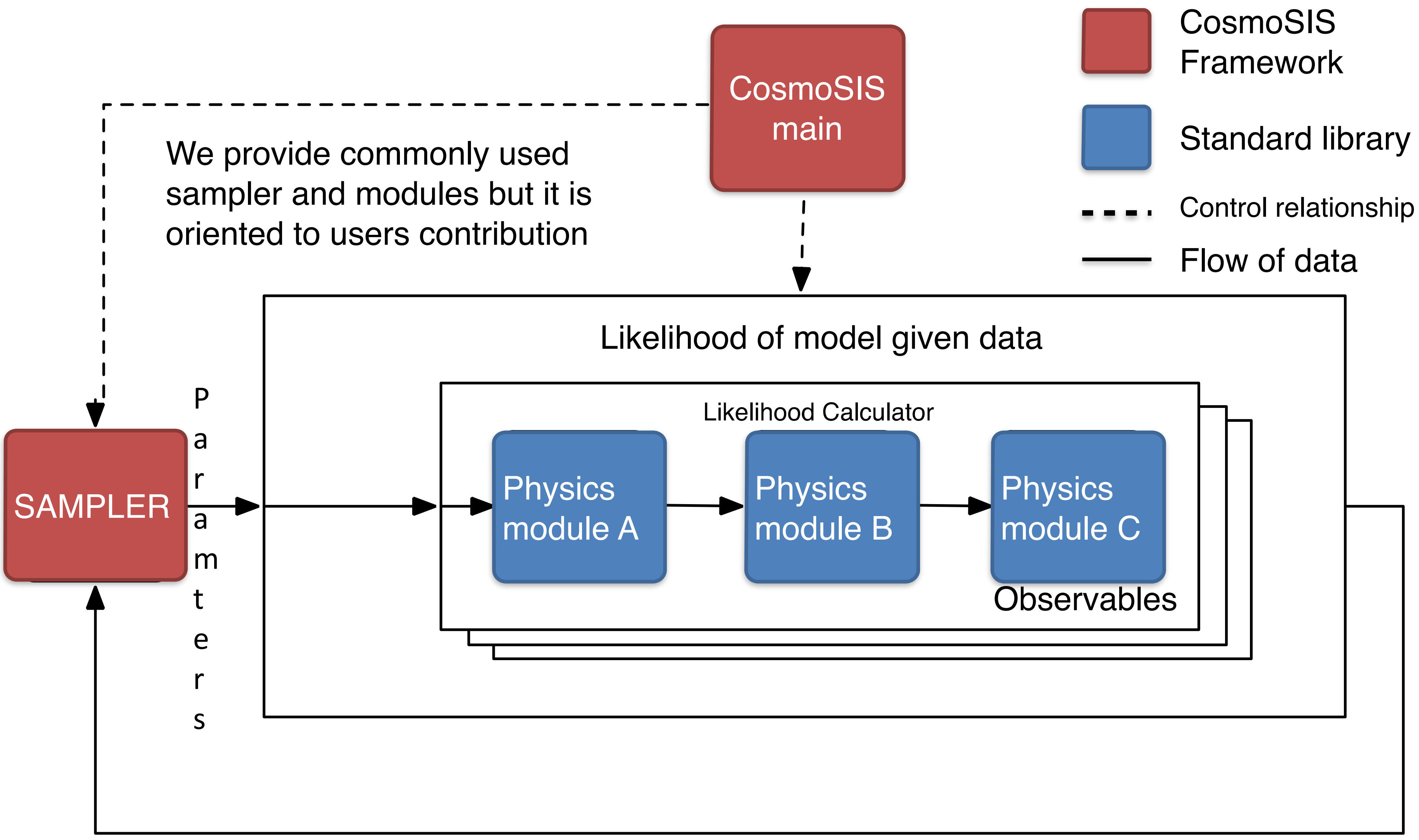
Runtime environment: To reliably configure and run programs that use these modules.

Development environment: To write, build, and test analysis software, and which makes it easy to share what they have developed.

Registration of attribution information (required citations, etc.) for use of any contributed code.

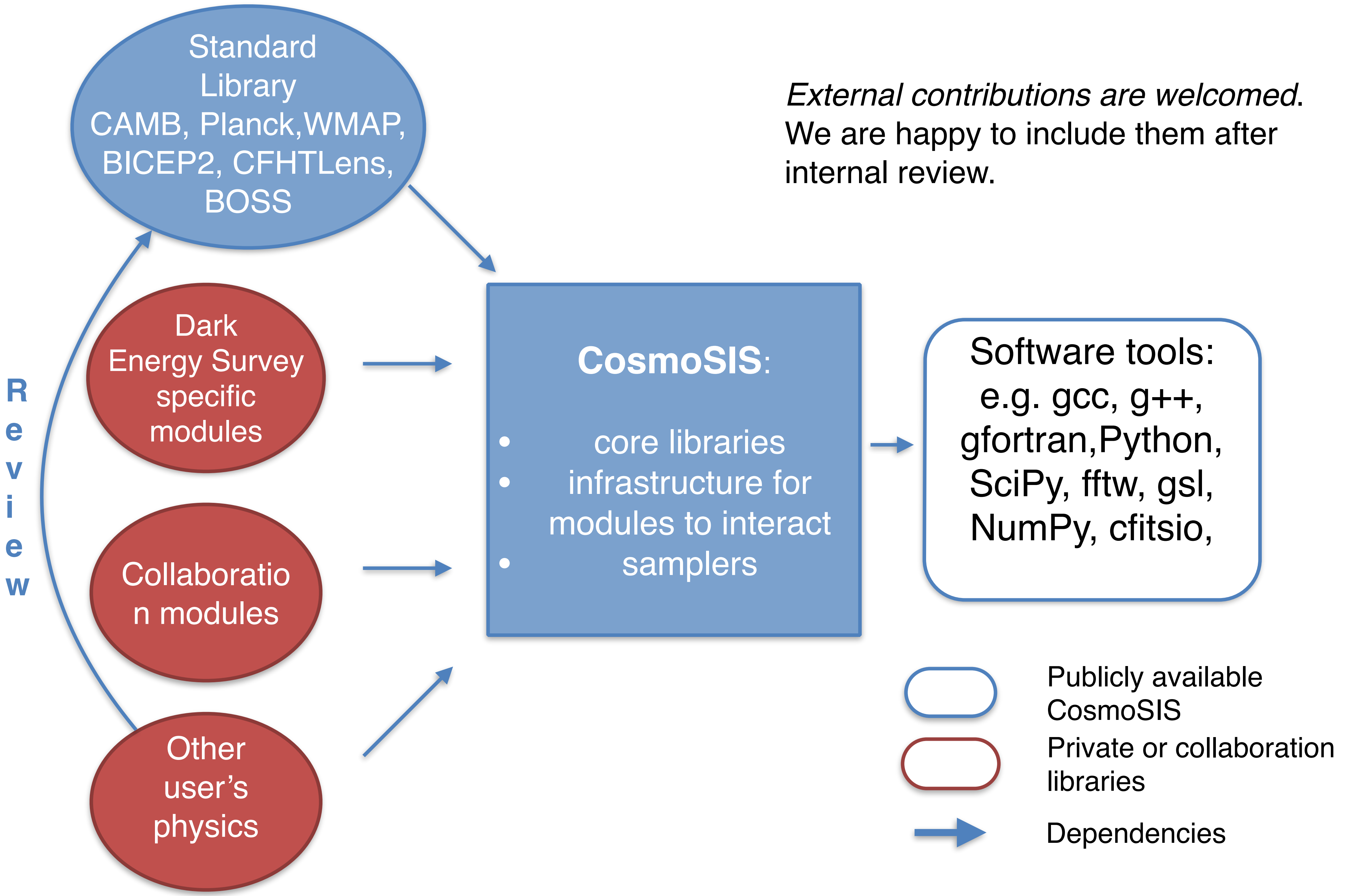
Distribution system: To make it easy to install the code and ensure compatibility.

MODULARITY IS THE KEY

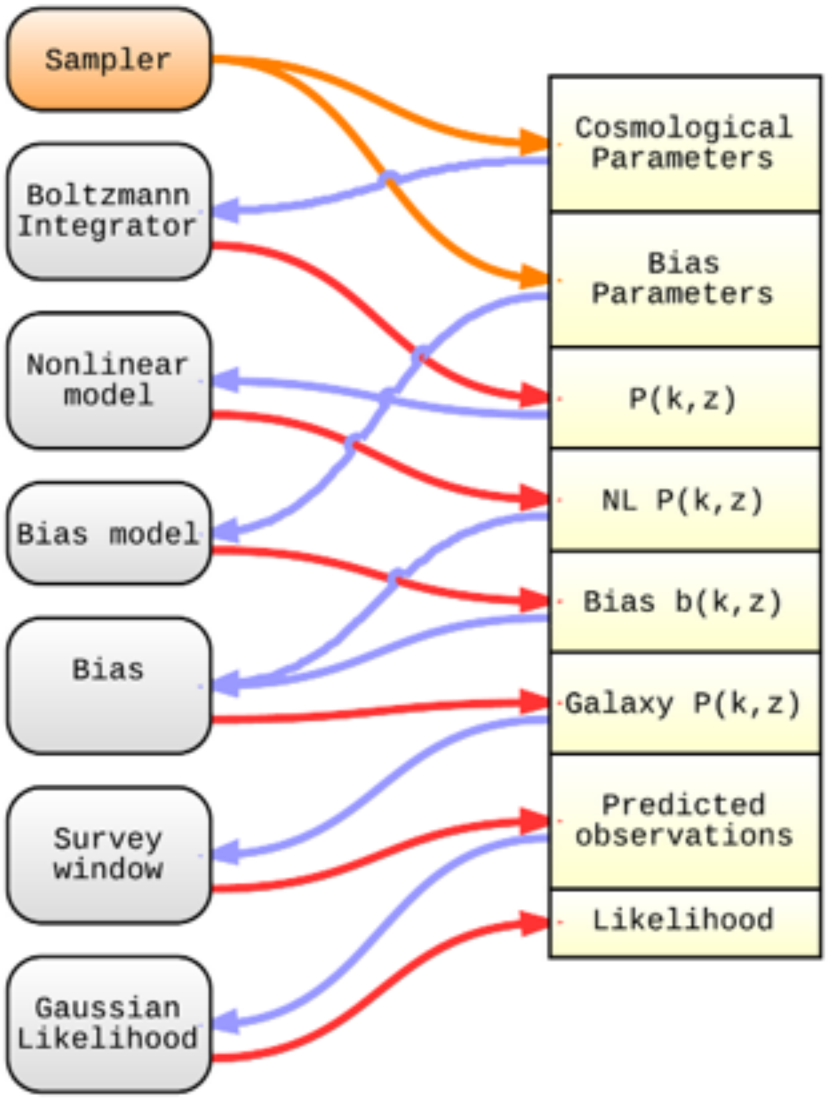


COSMOSIS STRUCTURE

External contributions are welcomed. We are happy to include them after internal review.

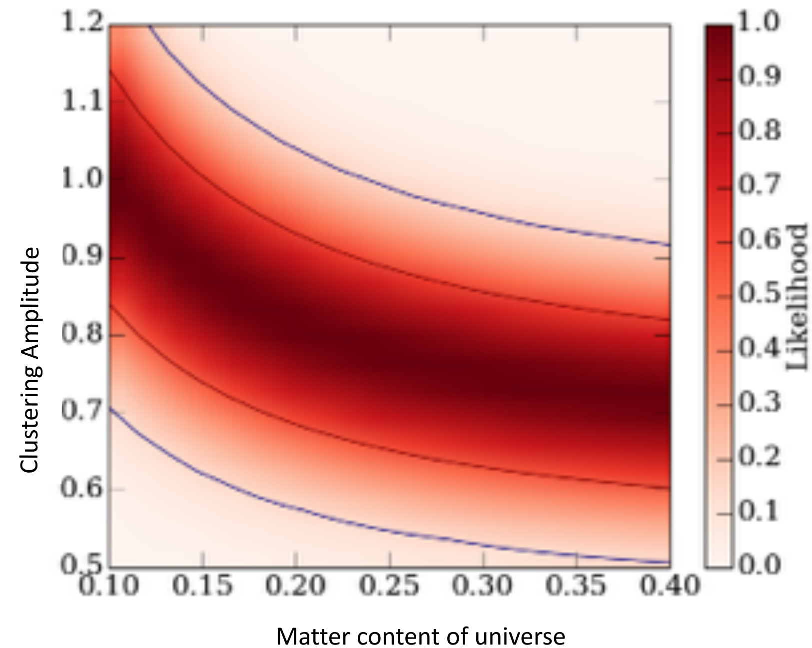


MODULARITY AT WORK



Output

Note: the result from event *n* (the likelihood) influences what happens in event *n+1* (MCMC process)



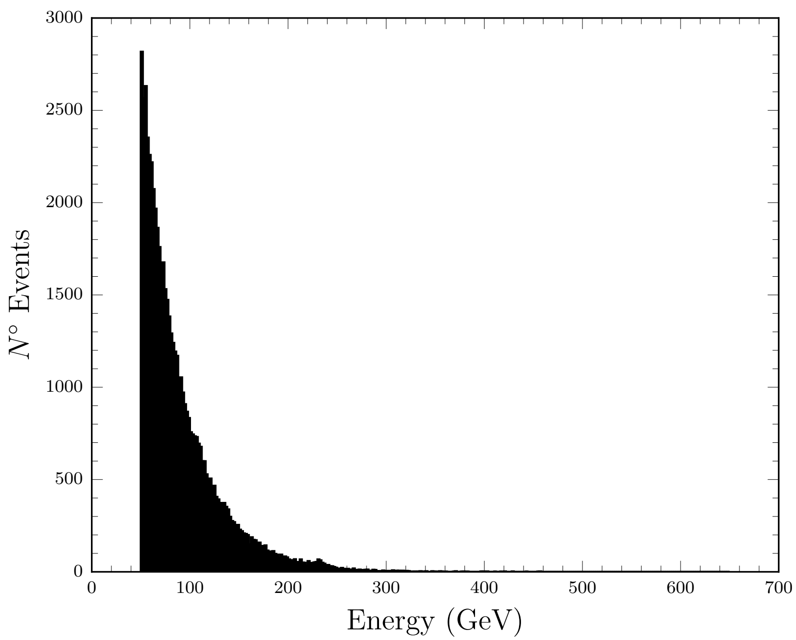
DataBlock passed to each module (similar to a HEP event)

Modules

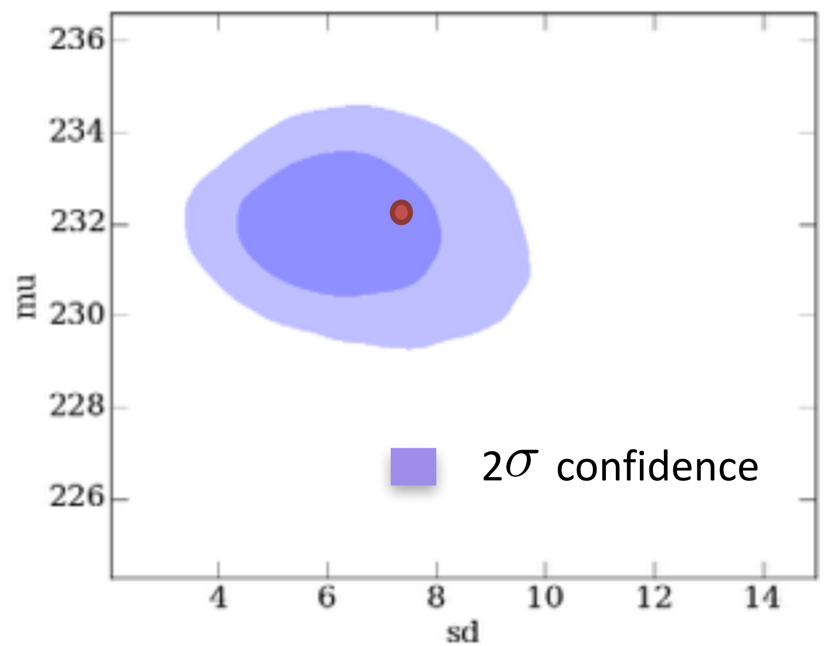
HEP EXAMPLE : BUMP HUNT 1

Bump Hunt toy model.
Exponential background + signal

INPUT:
mu (mass of resonance): 232.2 (units of GeV)
sd (width of resonance): 7.4 (units of GeV)
$\sigma_{\text{background}} = 800\ \sigma_{\text{signal}}$



CosmoSIS →



- Looking for a small Gaussian signal on top of a falling exponential background. We use a binned likelihood, Poisson statistics, and we integrate the cross section dependency across each mass bin

also B.A.T talk by F. BEAUJEAN

HEP EXAMPLE : BUMP HUNT 2

Simple

```
def execute(block, cfg):
    # Read this sample's parameters from the block
    lum     = block[params, "lum"]
    xsecbg  = block[params, "xsecbg"]
    beta    = block[params, "beta"]
    xsecsig = block[params, "xsecsig"]
    mu      = block[params, "mu"]
    sd      = block[params, "sd"]

    # Calculate the expected counts in each bin corresponding to
this
    # sample's parameters
    lows = cfg.lowedges
    highs = cfg.lowedges + cfg.binwidth
    f1 = np.exp(-1.0 * lows / beta)
    f2 = np.exp(-1.0 * highs / beta)
    expected_bkg    = lum * xsecbg * (f1 - f2)

    sqrt2sigma = np.sqrt(2.0)*sd
    g1 = special.erf( (mu-lows)/sqrt2sigma )
    g2 = special.erf( (mu-highs)/sqrt2sigma )
    expected_signal = lum * xsecsig * (g1 - g2) / 2.0

    expected_counts = expected_signal + expected_bkg

    # Now cacluate the log-likelihood for our data, given the
    # expectation for this sample
    loglike = np.sum(-expected_counts + cfg.counts *
np.log(expected_counts) - cfg.lnfactcounts)

    block[likes, "BUMP_HUNT_LIKE"] = loglike
    return 0
```

Well Documented

```
name: "BumpHunt"
version: "2015"
purpose: "Toy bump hunt example for HEP demonstration"
attribution: [Marc Paterno]
rules: "None."
cite:
    - "A. Manzotti et al., 'CosmoSIS: a System for MC Parameter
Estimation', CHEP 2015"

assumptions:
    - "Toy data set with Gaussian bump on exponential background"

explanation: >
    "This is a toy demonstration of using CosmoSIS for a non-
cosmology problem.

    We perform a fit to the binned data.
    "

# List of parameters that can go in the params.ini file in the
section for this module
params:
    datafile: "text, the name of the data file we're using the the
fit"
    lowedge: "float, the low edge of the mass histogram"
    nbins:   "int, the number of bins in the histogram"
    binsize: "float, the width of bins in the histogram"

#Inputs for a given choice of a parameter, from the values.ini
inputs:
    cosmological_parameters:
      lum: "the integrated luminosity for the data set"
      xsecbg: "the cross section for the background process"
      beta: "the exponential background falloff parameter (units of
mass)"
      xsecsig: "the cross section for the signal process"
      mu: "the mass of the bump"
      sd: "the width of the bump"

outputs:
    likelihoods:
        BUMP_HUNT_LIKE: "Likelihood for the observed data, given the
parameters."
```

COSMOSIS: A COMPLETE TOOLKIT

- CosmoSIS parallelism with OpenMP and MPI
  - develop a program on your laptop
  - without change, run using thousands of cores on an HPC cluster
- Tools for diagnosis of convergence, thinning, etc.
  - Gelman-Rubin statistic,auto-correlation length test
  - Continue sampling from a previous chain
- Tools for analysis of posterior densities
  - single parameters and two-parameter posterior density plots
  - Basic statistic of the chains and covariances
- Integration with diverge community supported codes

SOME LESSONS LEARNED (OR RE-LEARNED)

<u>Cosmologists are learning</u>. We see the value and feasibility of well-controlled software, with strong control over versioning and binary compatibility.

<u>Contribution and sharing increases</u>. Open-source model for contribution of modules (with attribution for work) has helped attract interest in sharing code.

<u>HEP might consider</u> adopting a similar attribution concept to help encourage sharing (and rewarding the developers of) useful software. Multi-language systems lower the bar on programming expertise for contribution.

https://bitbucket.org/joezuntz/cosmosis/wiki/Home

KEEP CALM
and
BE MODULAR

https://bitbucket.org/joezuntz/cosmosis/wiki/Home

Additional temp Slides.