CosmoSIS

Joe Zuntz

Outline

- Understanding CosmoSIS
- Using CosmoSIS
- Modifying CosmoSIS

Purpose

- **Connect** other codes together to form Likelihood Pipeline
- Run many **sampling methods**
- Package **standard code library**

Relationships to other codes

- Connects to Camb & Class

- Alternative to Montepython & CosmoMC

Likelihoods

- Likelihood = P(data | parameters)
- Final step (prediction vs observable) sometimes simple function
- Prediction as function of parameters usually complicated
    - Sequence of calculations
    - CosmoSIS = framework for building and sampling likelihood function pipelines

Philosophy

- Many pipelines start with common components like CAMB, CLASS, Astropy
    - Calculate b/g evolution, matter power, growth function, others
- Much easier to **build on** these than **modify/extend** them
    - Use as CosmoSIS modules & do calculations from their output

Components

- Replaceable parts with explicit inputs and outputs
    - Multi-lingual plugin framework
        - Simple data-passing API
- Unified configuration
- Unified interface to many sampling (and other) methods

Philosophy

- Make conceptually separate calculations actually separate in code

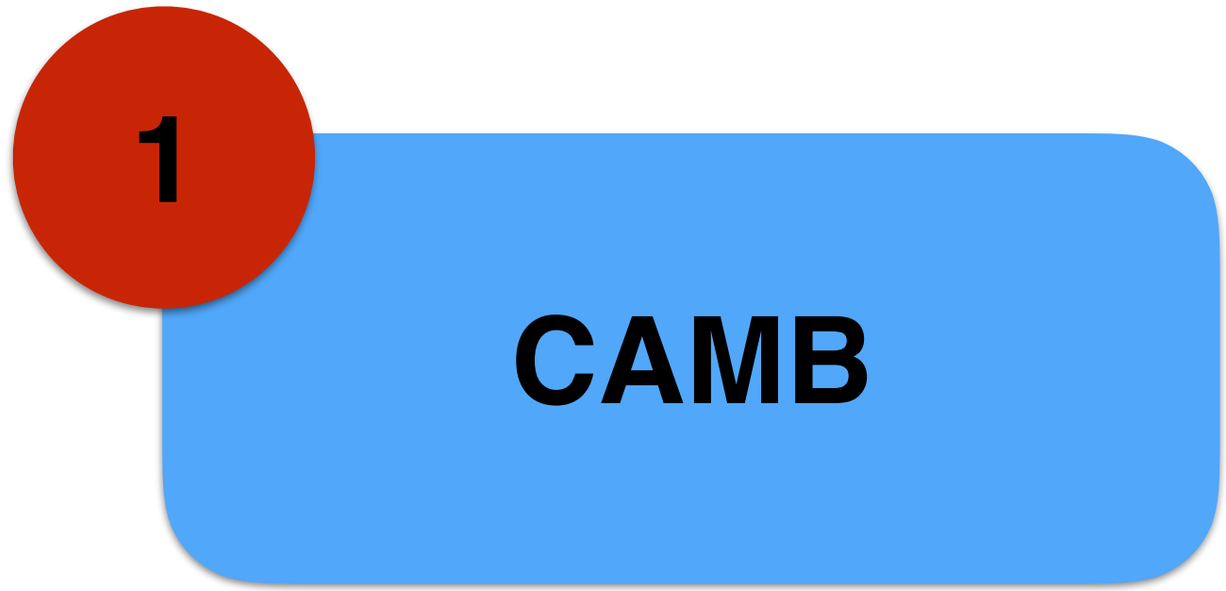
- Example:
  Weak Lensing Likelihood

Example

1. CAMB
2. Halofit
3. Extrapolate P(k)
4. Load n(z)
5. n(z) Biases
6. NLA model
7. 3D P(k) -> 2D C_ℓ
8. Add IA
9. Calibration Biases
10. Correlation Functions
11. Likelihood

Modules

- Each one of these is wrapped as a **CosmoSIS Module**



- Self-contained chunk of code, python, c, c++, f90

- Inputs & outputs from & to CosmoSIS

Modules

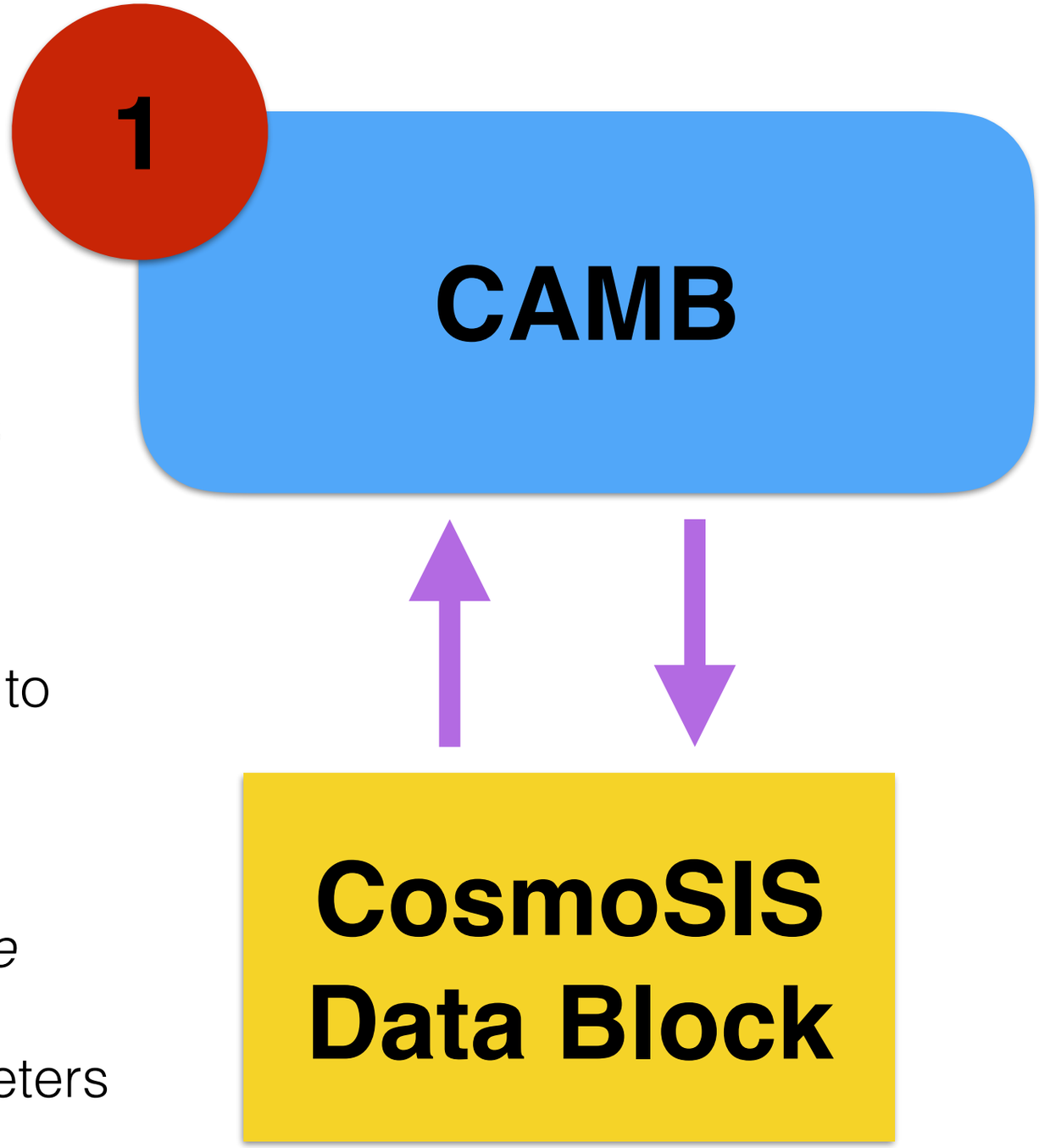
- Large collection provided with CosmoSIS
- You can add your own new ones very easily
- Typically last modules in pipeline compute final likelihood (e.g Gaussian)

Connections

- Two things connect modules into CosmoSIS
  - Module functions:
    `setup` - configure module, at start
    `execute` - run module, for each param set
  - Use collection of functions (API) to
    read inputs from CosmoSIS
    & write results to CosmoSIS

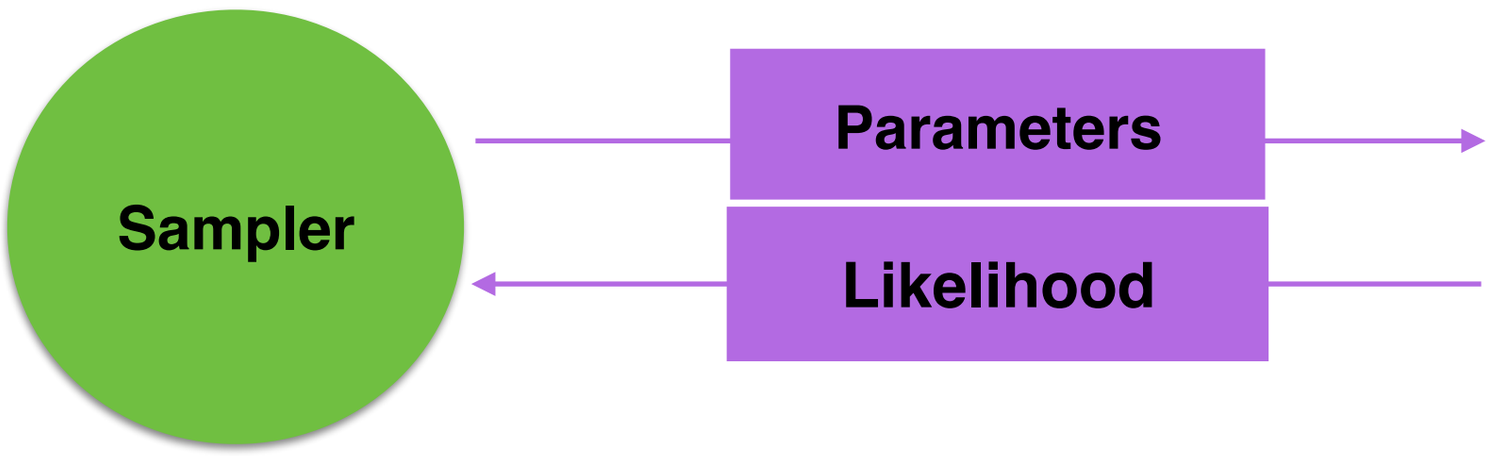
Connections

- Cosmology theory is stored in a *Data Block*
- CAMB:
    - reads cosmological parameters from block
    - writes power spectra (CMB & matter) and distance measures to block
- Block stores numbers, strings, and arrays with two keys, *section* & *name*
    - e.g section=cosmological_parameters name=omega_m



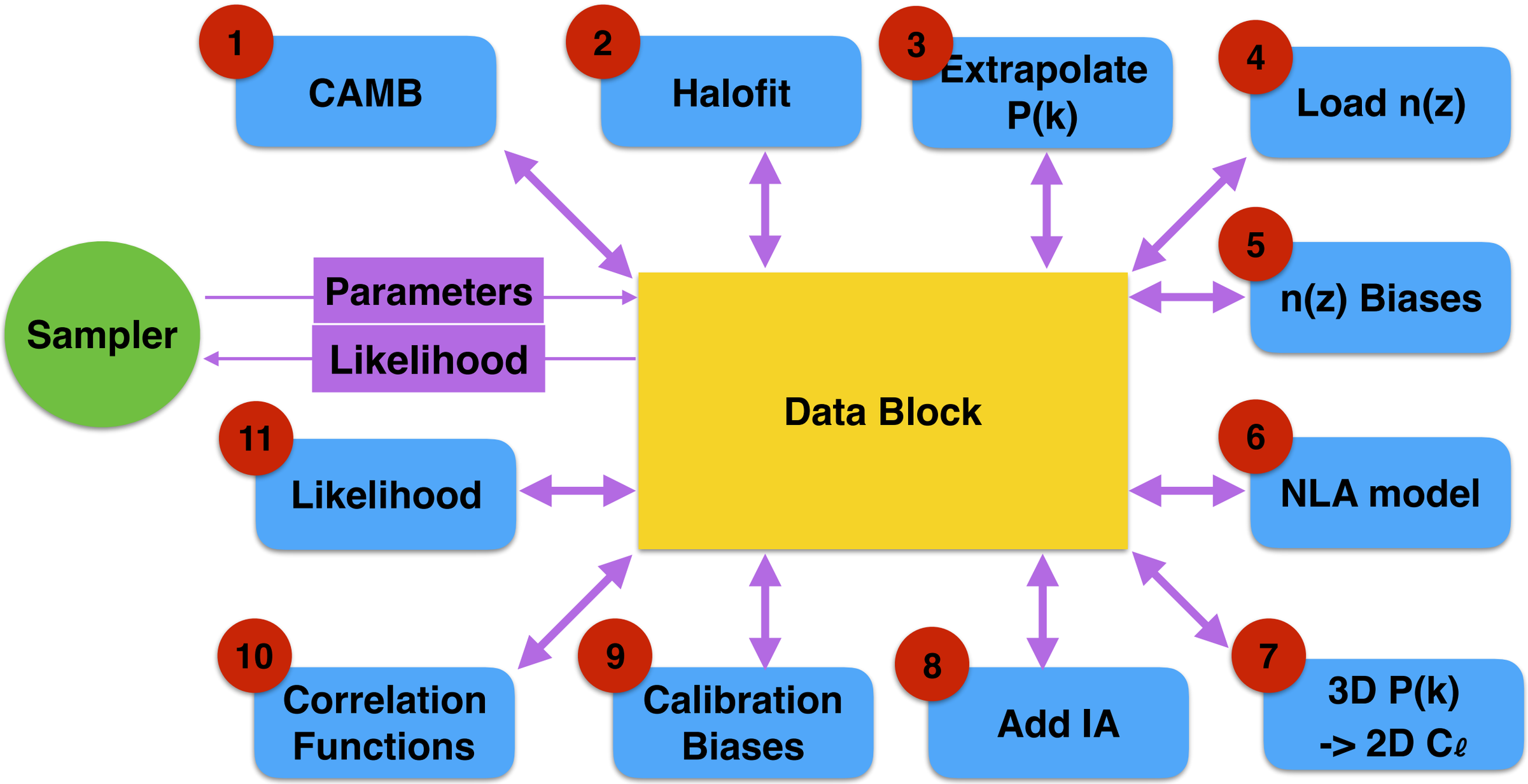
Samplers

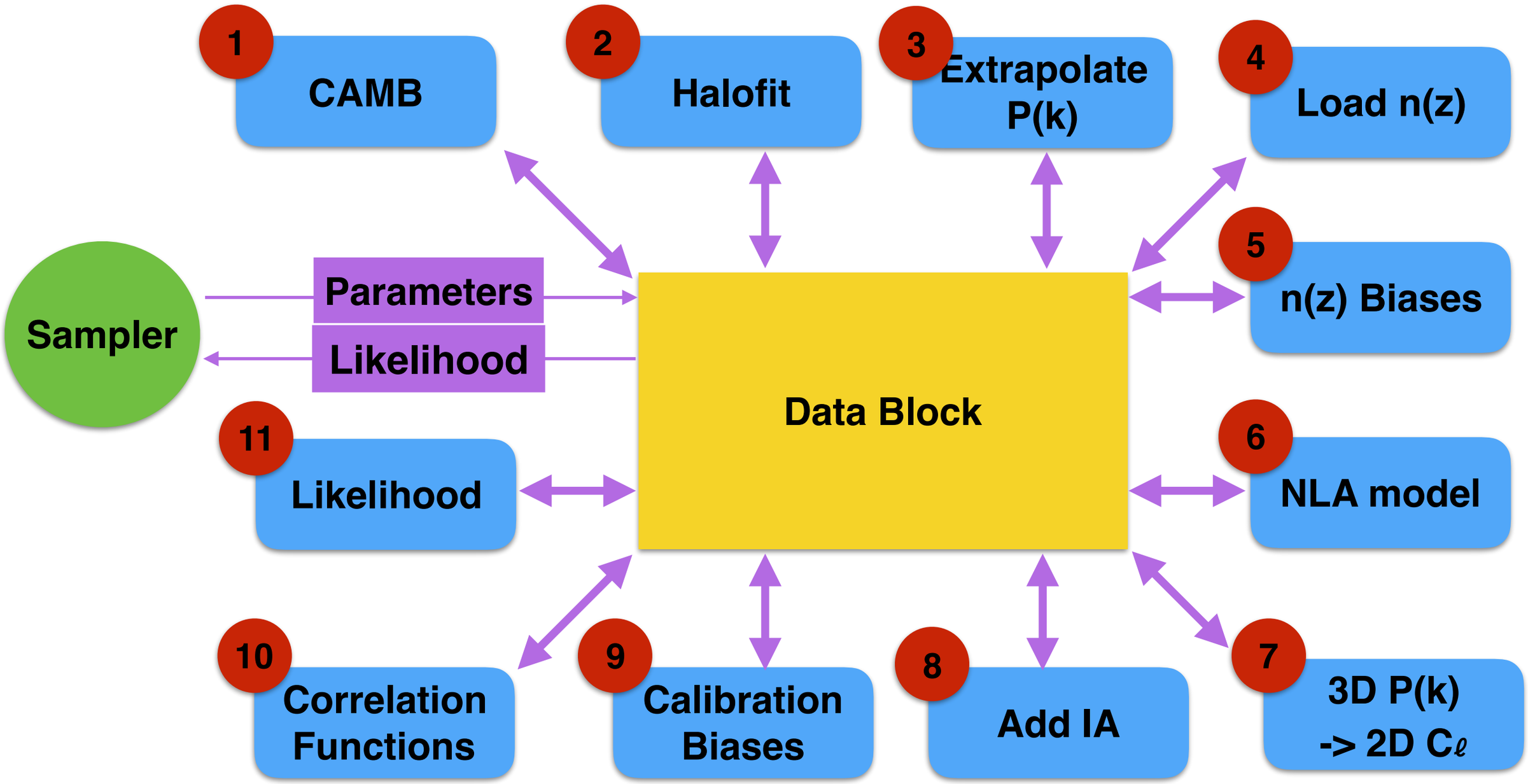
- “Sampler” = Anything that generates 1+ sets of sample parameters
    - e.g MCMC, grids, maximum likelihood
- always adding new samplers



Unified Sampler Interface

- abc
- emcee
- fisher
- grid
- gridmax
- importance
- kombine
- list
- maxlike
- metropolis
- minuit
- multinest
- pmaxlike
- pmc
- polychord
- pymc
- snake
- star
- test


1
CAMB
2
Halofit
3
Extrapolate P(k)
4
Load n(z)
5
n(z) Biases
Sampler
Parameters
Likelihood
Data Block
11
Likelihood
6
NLA model
10
Correlation Functions
9
Calibration Biases
8
Add IA
7
3D P(k) -> 2D Cℓ


1
CAMB
2
Halofit
3
Extrapolate P(k)
4
Load n(z)
5
n(z) Biases
Sampler
Parameters
Likelihood
Data Block
11
Likelihood
6
NLA model
10
Correlation Functions
9
Calibration Biases
8
Add IA
7
3D P(k) -> 2D Cℓ

Using CosmoSIS

Documentation

- See the nearly adequate docs on our wiki:

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

Process

- Figure out your pipeline
- Write configuration files
- Run `cosmosis` to make chains
- Run `postprocess` to make plots

Figuring out pipeline

- Combine existing modules
  https://bitbucket.org/joezuntz/cosmosis/wiki/default_modules
- Write new modules
- Think how they connect together to form a pipeline
- Maybe start from existing pipeline, adding new modules to start, middle, or end

Three Configuration Files

- Parameters - configure modules and sampler
- Values - configure input (fixed and varying) (cosmological, nuisance, etc) params
- Priors (optional) - set priors on Values

Parameter File

```
[runtime]
sampler = metropolis
```

Choose & configure samplers

```
[metropolis]
nsteps=10
random_start=F
samples=100000
covmat=examples/covmat_a.txt
Rconverge = 0.02

[output]
format=text
filename=example_output_a.txt
verbosity=debug
```

Output file

```
[pipeline]
modules = consistency camb wmap
values = examples/values_a.ini
likelihoods = wmap9
extra_output = cosmological_parameters/omega_m cosmological_parameters/omega_b
quiet=F
debug=F
timing=F
```

Pipeline to run

Parameter File

Configure individual modules

```
[consistency]
file=cosmosis-standard-library/utility/consistency/consistency_interface.py
verbose=F

[camb]
file = cosmosis-standard-library/boltzmann/camb/camb.so
mode=cmb
lmax=1300
feedback=0

[wmap]
file = cosmosis-standard-library/likelihood/wmap9/wmap_interface.so
```

Values File

```
[cosmological_parameters]
ommh2 = 0.05   0.128580559014883  0.25
h0  =  0.5    0.72550194314801075    0.9
ombh2 = 0.01  0.022423250049654054   0.04
tau  =  0.05    0.08450321665857593 6    0.11
n_s  =  0.8    0.9751697641353043 2    1.1
a_s  =  2e-09    2.1208320120685232e-09    2.4e-09


omega_k  =  0.0
w  =  -1.0
wa  =  0.0
```

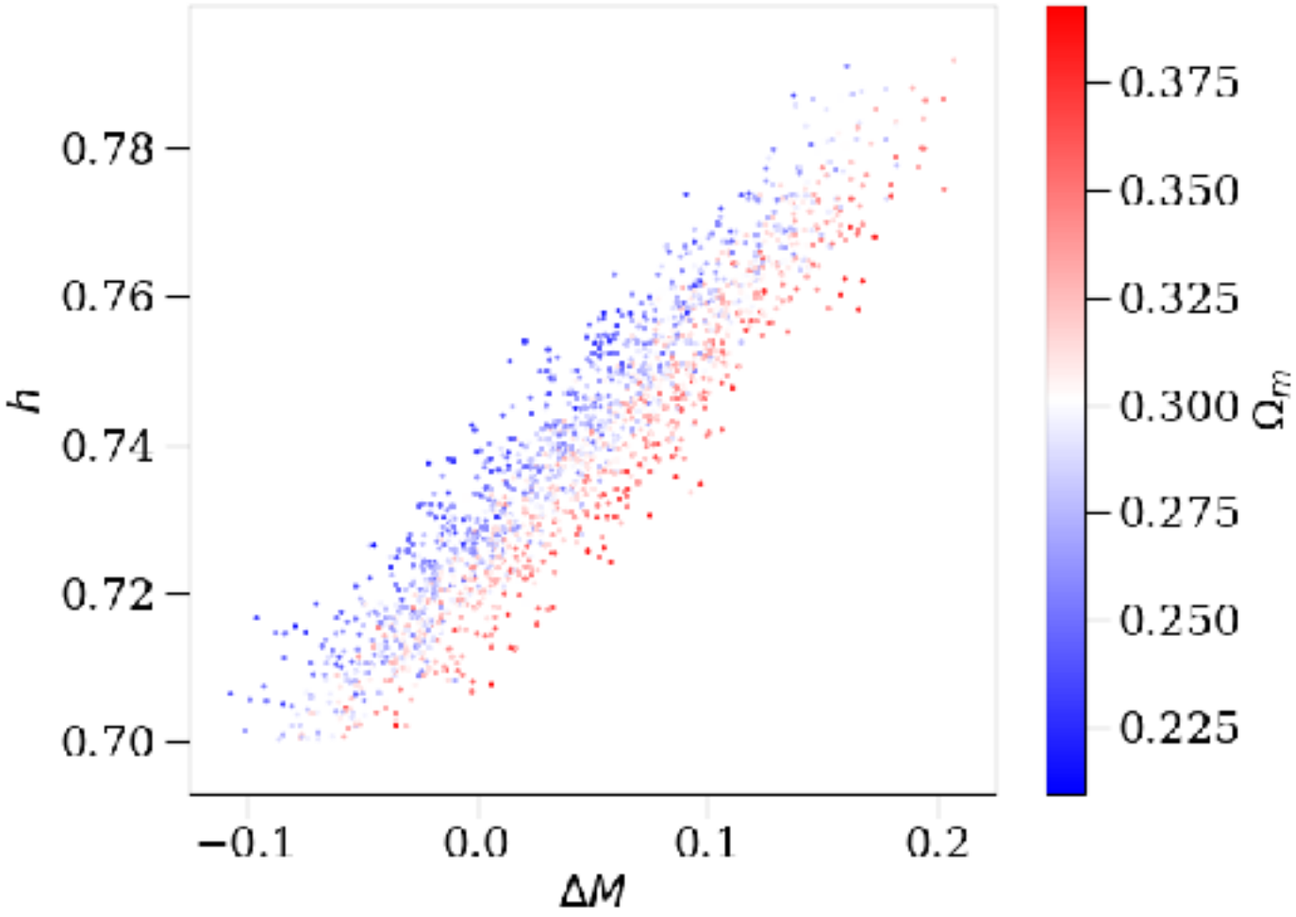
Priors File

```
#examples only!
[cosmological_parameters]
h0 = gaussian 0.738  0.024
omnhu2 = exponential 0.005
omega_b = uniform 0.04 0.05
```

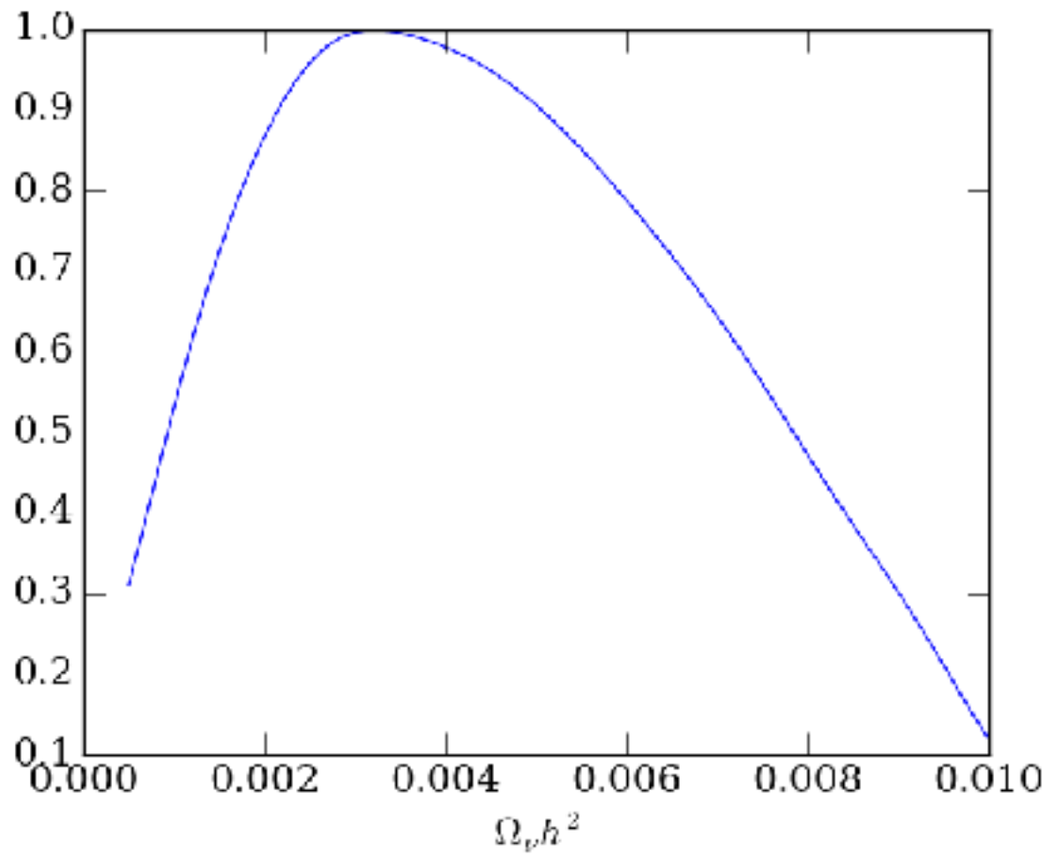
Running

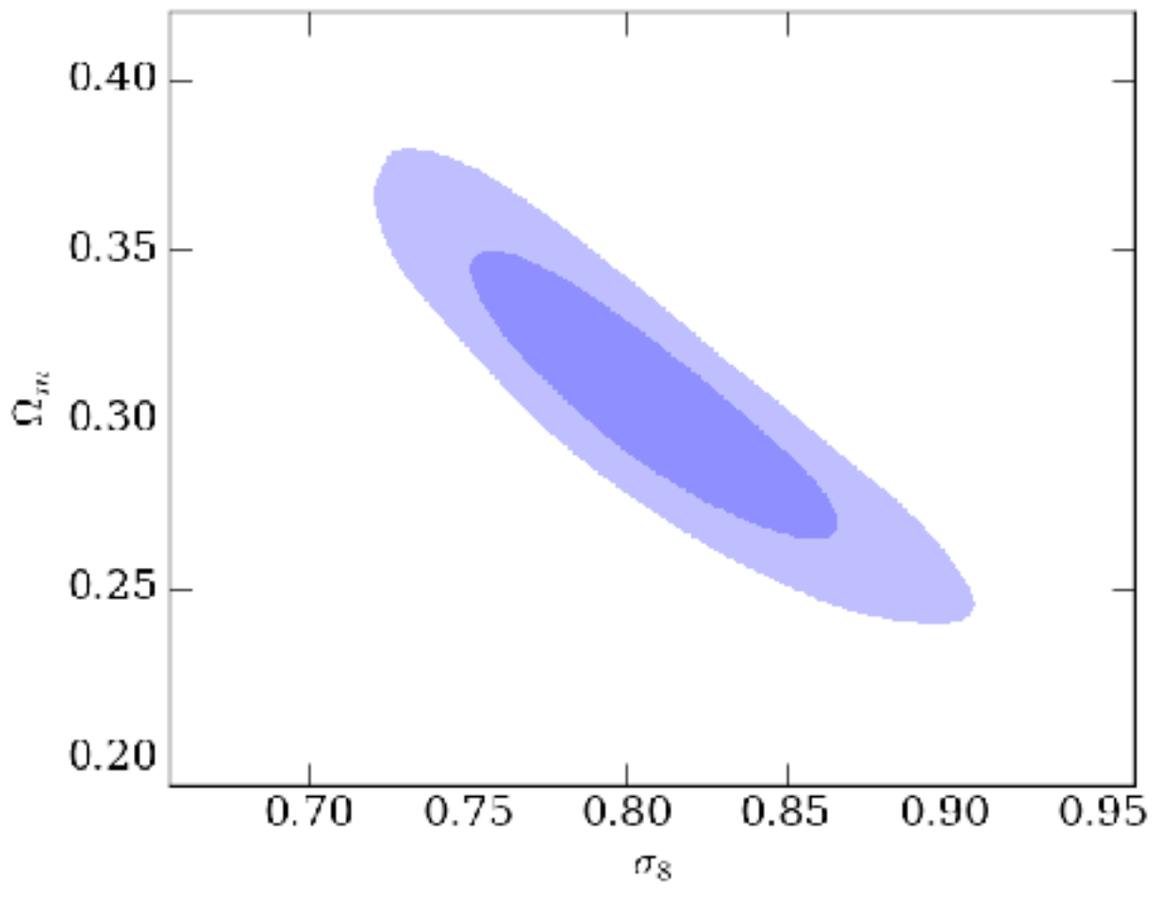
```
> cosmosis params.ini
      (Single process)

> mpirun -n 8 cosmosis --mpi params.ini
      (Multiprocess)
```

```
> postprocess params.ini

> postprocess chain.txt
```


0.78
0.76
0.74
0.72
0.70
h
−0.1
0.0
0.1
0.2
ΔM
0.375
0.350
0.325
0.300
0.275
0.250
0.225
Ω_m

Plots


1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.000
0.002
0.004
0.006
0.008
0.010
Ω_ν h²


0.40
0.35
0.30
0.25
0.20
Ω_m
0.70
0.75
0.80
0.85
0.90
0.95
σ_8

Modifying CosmoSIS

Setup Function

- Argument
    - `options` (datablock, contents of parameter file)
- Returns
    - C-pointer or python object

Execute Function

- Argument
    - `block` (datablock, output from sampler and prev. pipeline)
    - `config` (pointer/object, output of setup function)
- Returns
    - Status Integer (success = 0)

API
Used by setup, execute

- In C, C++, Python, and Fortran

- Get (read) and Put (write) functions/methods:

```
c_datablock_get_double(c_datablock* s,
    const char* section, const char* name,
    double* val);
```

C

```
DataBlock::get_val(std::string section, std::string name, T& val);
```

C++

```
datablock_put_logical(block, section, name, value) result(status)
        integer(cosmosis_status) :: status
        integer(cosmosis_block) :: block
        character(len=*) :: section
        character(len=*) :: name
```

Fortran

```
def get_double_array_1d(self, section, name):
```

Python

- Lookup functions, other introspection

Example: Growth Function

```c
const char * cosmo = COSMOLOGICAL_PARAMETERS_SECTION;
const char * like = LIKELIHOODS_SECTION;
const char * growthparameters = GROWTH_PARAMETERS_SECTION;

typedef struct growth_config {
    double zmin;
    double zmax;
    double dz;
    int nz;
} growth_config;

growth_config * setup(c_datablock * options)
{
    int status = 0;
    growth_config * config = malloc(sizeof(growth_config));
    status |= c_datablock_get_double_default(options, OPTION_SECTION, "zmin", 0.0, &(config->zmin));
    status |= c_datablock_get_double_default(options, OPTION_SECTION, "zmax", 3.0, &(config->zmax));
    status |= c_datablock_get_double_default(options, OPTION_SECTION, "dz", 0.01, &(config->dz));
    config->nz = (int)((config->zmax-config->zmin)/config->dz)+1;
    // status |= c_datablock_get_double(options, OPTION_SECTION, "redshift", config);
    // status |= c_datablock_get_double(options, OPTION_SECTION, "redshift", config);

    printf("Will calculate f(z) and d(z) in %d bins (%lf:%lf:%lf)\n", config->nz, config->zmin, config->zmax, config->dz);
    // status |= c_datablock_get_double(options, OPTION_SECTION, "redshift", config);
        if (status){
                fprintf(stderr, "Please specify the redshift in the growth function module.\n");
                exit(status);
        }
    return config;

}
```

```c
int execute(c_datablock * block, growth_config * config)
{

    int i,status=0;
    double w,wa,omega_m;
    double *dz,*fz,*zbins;
    int nzbins = config->nz;

    //allocate memory for single D, f and arrays as function of z
    double gf[2];
    dz = malloc(nzbins*sizeof(double));
    fz = malloc(nzbins*sizeof(double));
    zbins = malloc(nzbins*sizeof(double));
    //read cosmological params from datablock
        status |= c_datablock_get_double_default(block, cosmo, "w", -1.0, &w);
        status |= c_datablock_get_double_default(block, cosmo, "wa", 0.0, &wa);
        status |= c_datablock_get_double(block, cosmo, "omega_m", &omega_m);
    if (status){
        fprintf(stderr, "Could not get required parameters for growth function (%d)\n", status);
        return status;
    }


    //output D and f over a range of z
    for (i=0;i<nzbins;i++)
    {
        double z = config->zmin + i*config->dz;
        status = get_growthfactor(1.0/(1.0+z),omega_m,w,wa,gf);
        dz[i] = gf[0];
        fz[i] = gf[1];
        zbins[i] = z;
    }
    status |= c_datablock_put_double_array_1d(block,growthparameters, "d_z", dz, nzbins);
    status |= c_datablock_put_double_array_1d(block,growthparameters, "f_z", fz, nzbins);
    status |= c_datablock_put_double_array_1d(block,growthparameters, "z", zbins, nzbins);
    free(fz);
    free(dz);
    free(zbins);

return status;
}
```

Overview

- Break up your calculation into conceptual modules

- Make modules by wrapping calculation
  setup & execute functions calling API to get
  inputs/outputs from from/to pipeline

- Write parameter files to link and configure modules and choose a sampler

- Run the sampler and post-process the output chain

- Learn more by looking at the demos and existing modules

Getting Started

- At this school:

```
git clone http://bitbucket.org/joezuntz/cosmosis
cd cosmosis
git clone http://bitbucket.org/joezuntz/cosmosis-standard-library
cp /home/prof10/setup-cosmosis ./
source setup-cosmosis
update-cosmosis --develop
make
```

- Run the demos: https://bitbucket.org/joezuntz/cosmosis/

Activities

- Modify demo 11 to sample a grid over w_0 as well as modified gravity parameters

- Modify demo 9 to sample over w_0 as well, and compare the Bayesian evidence of the new model