UNIVERSITÉ DE GENÈVE

Open-Source Frameworks for Modelling of Extragalactic Jets

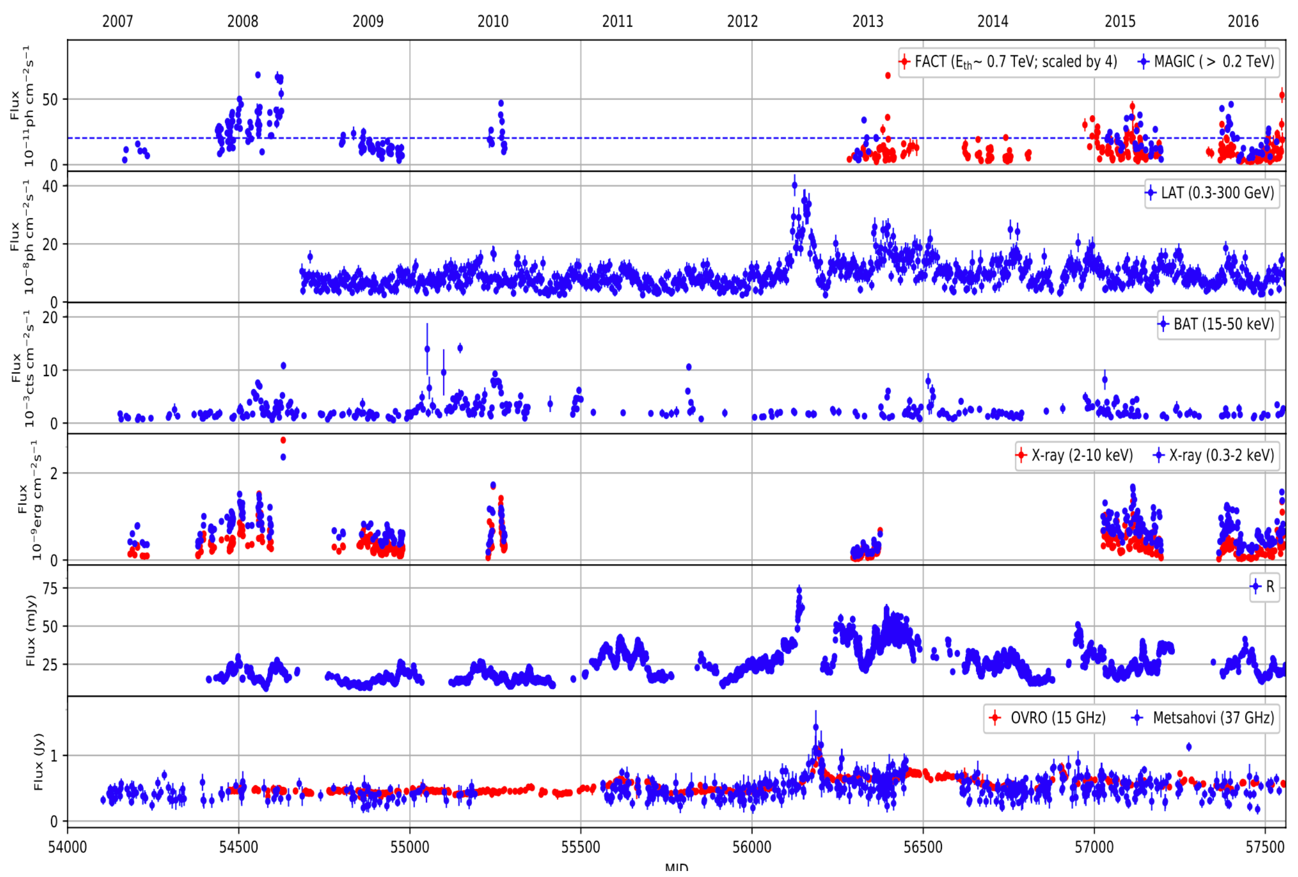
Andrea Tramacere

RICAP 2024

JetSeT

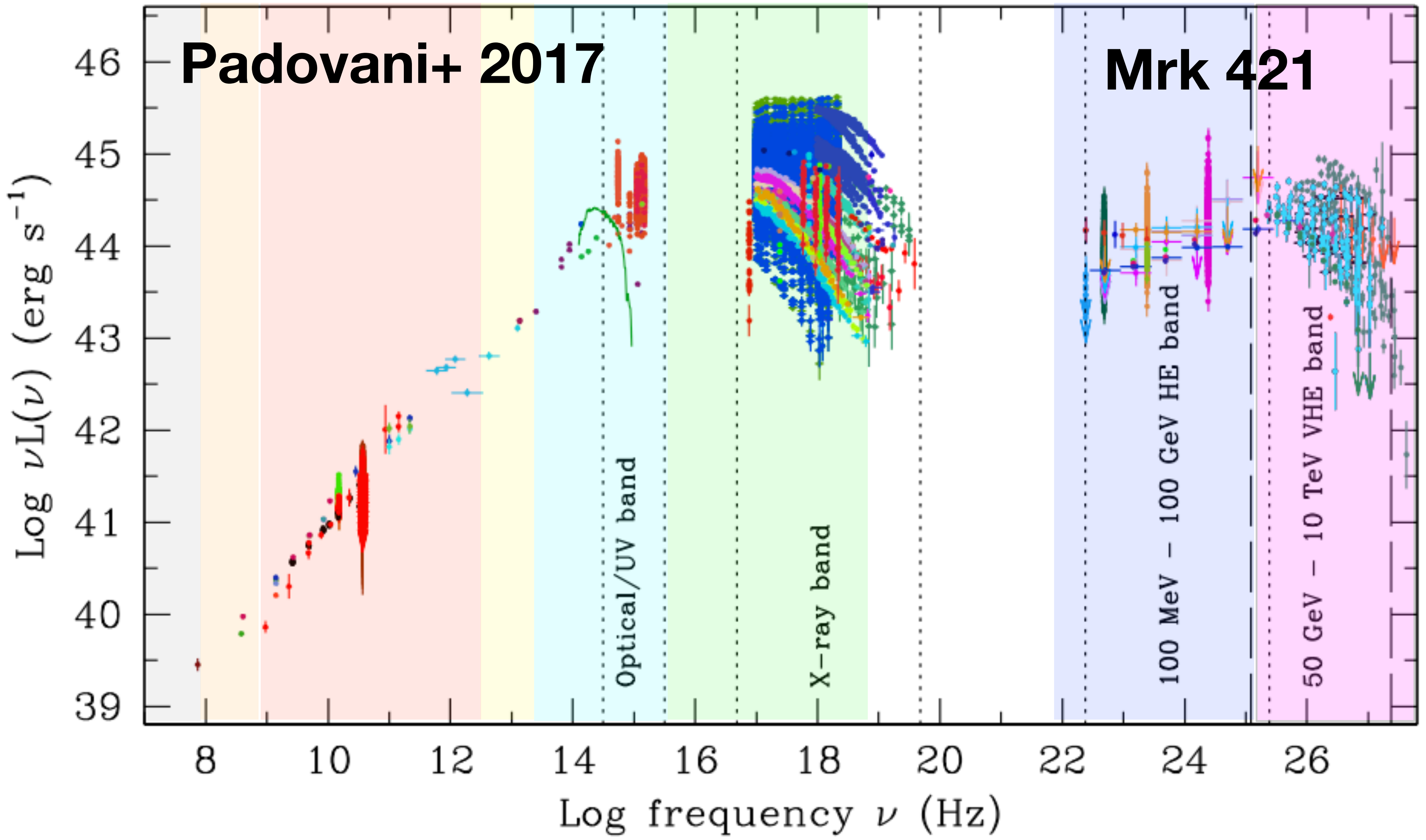
Relativistic jets modelling in the MW and MM era

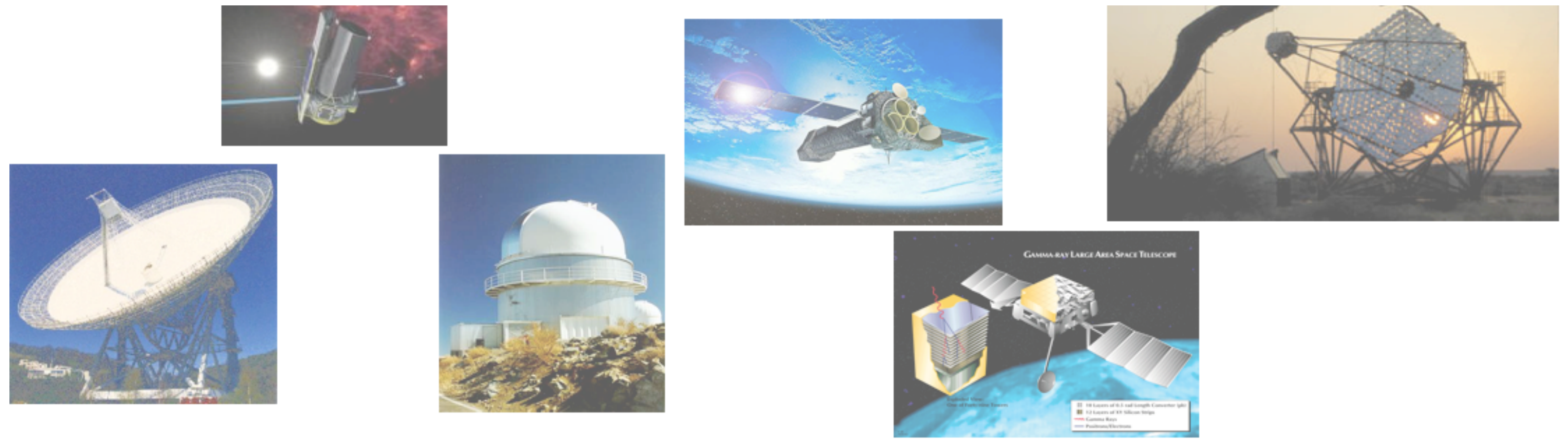
Magic coll. 2020 Mrk 421



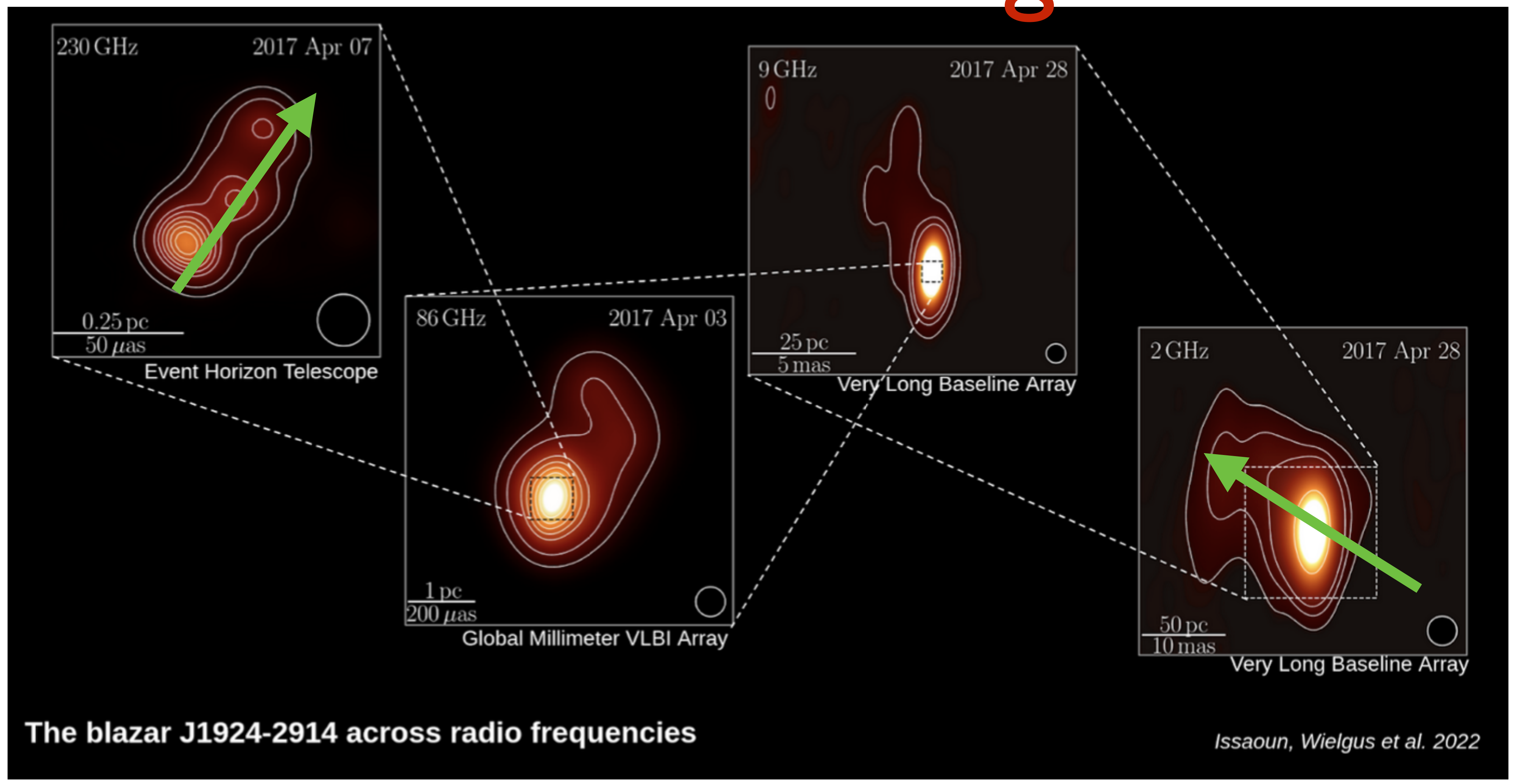
data (photons + ν)

spec+pol.

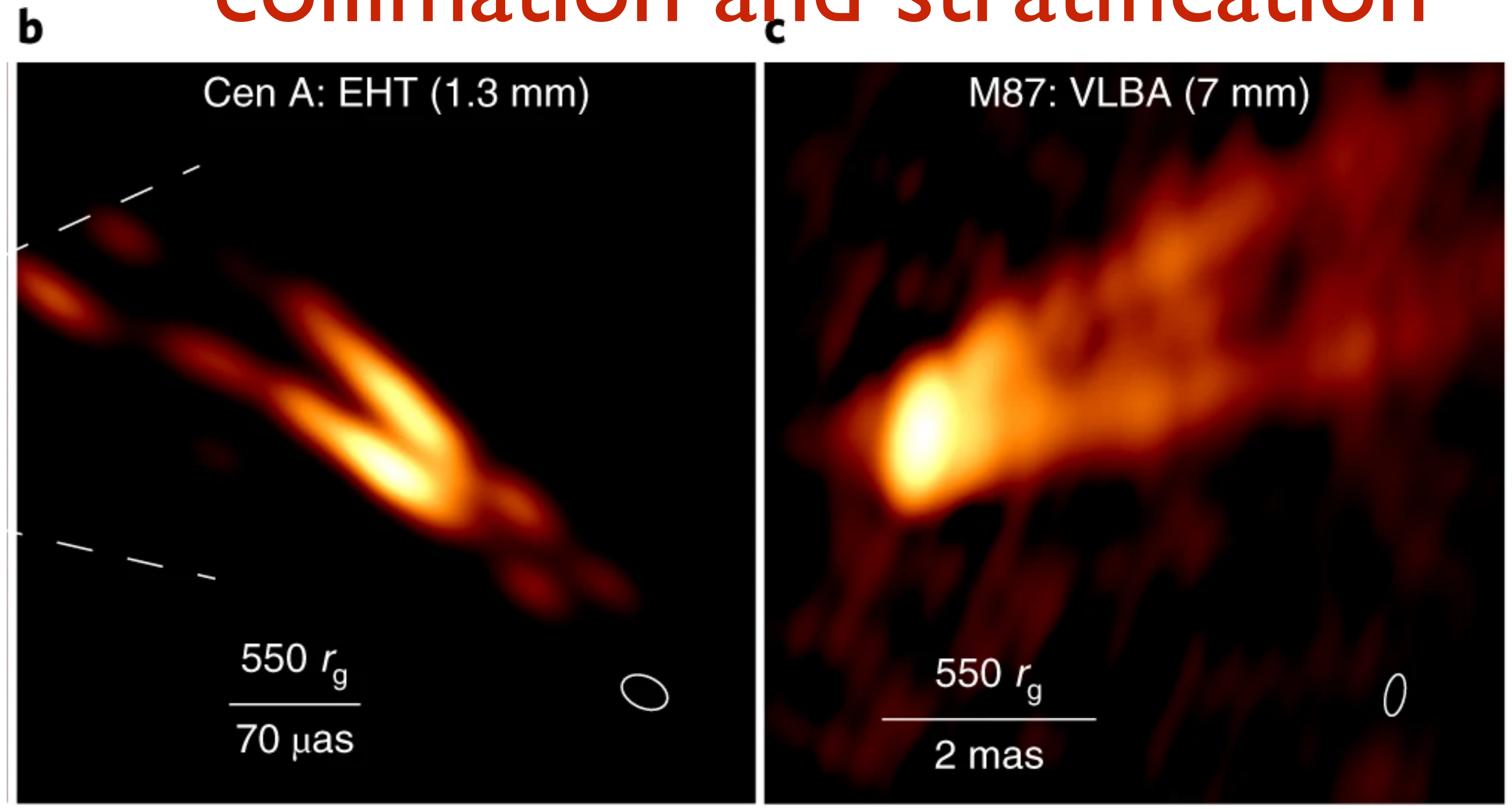




bending



collimation and stratification



a complex phenomenology requires a flexible framework

- complex MW/MM patterns eg. polarization / ν
- radio-gamma delays
- strong spectral evolution
- rapid variability, orphan flares

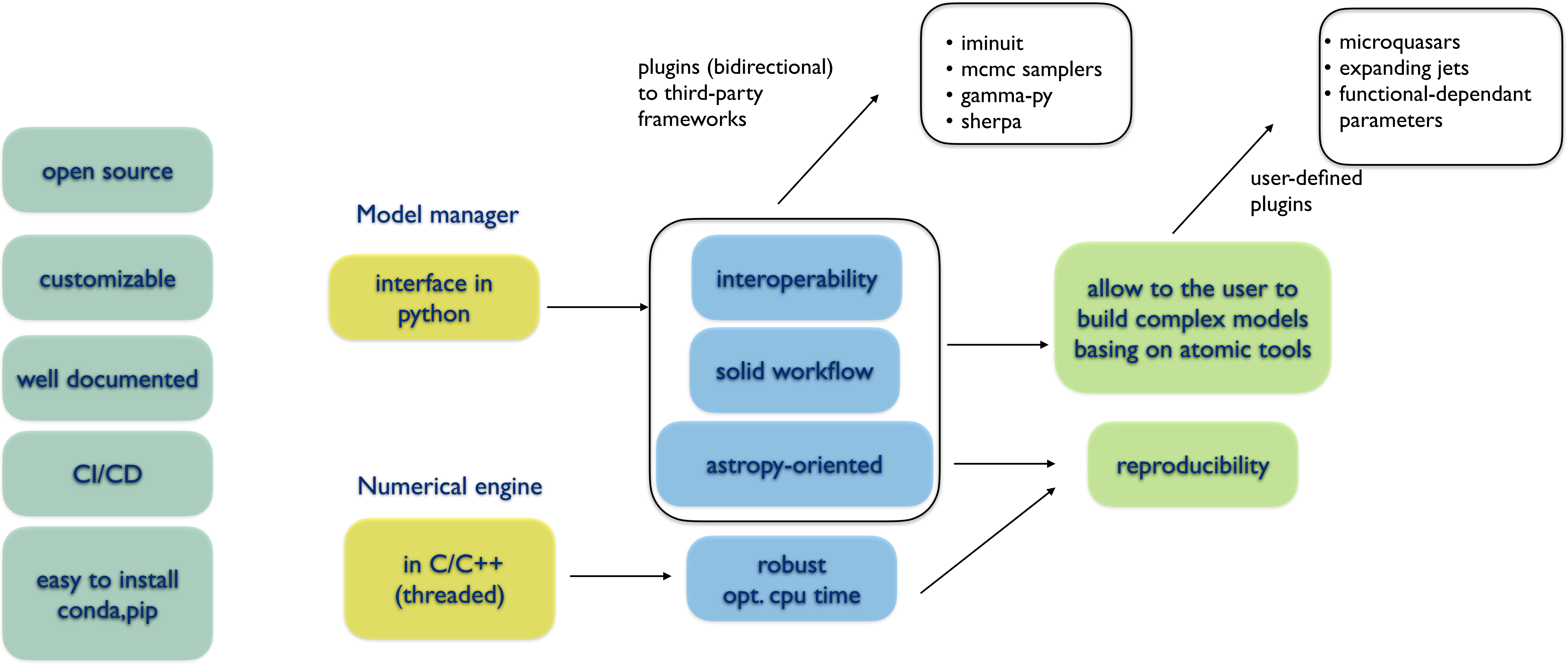
→

- emission at different scales across the jet/stratification (eg. fast-spine/slow-layer)
- interaction of several emitting regions
- disk-corona-wind
- leptonic and lepto-hadronic processes
- cooling/acc competition (pari-production, FI+FII,mag rec.)

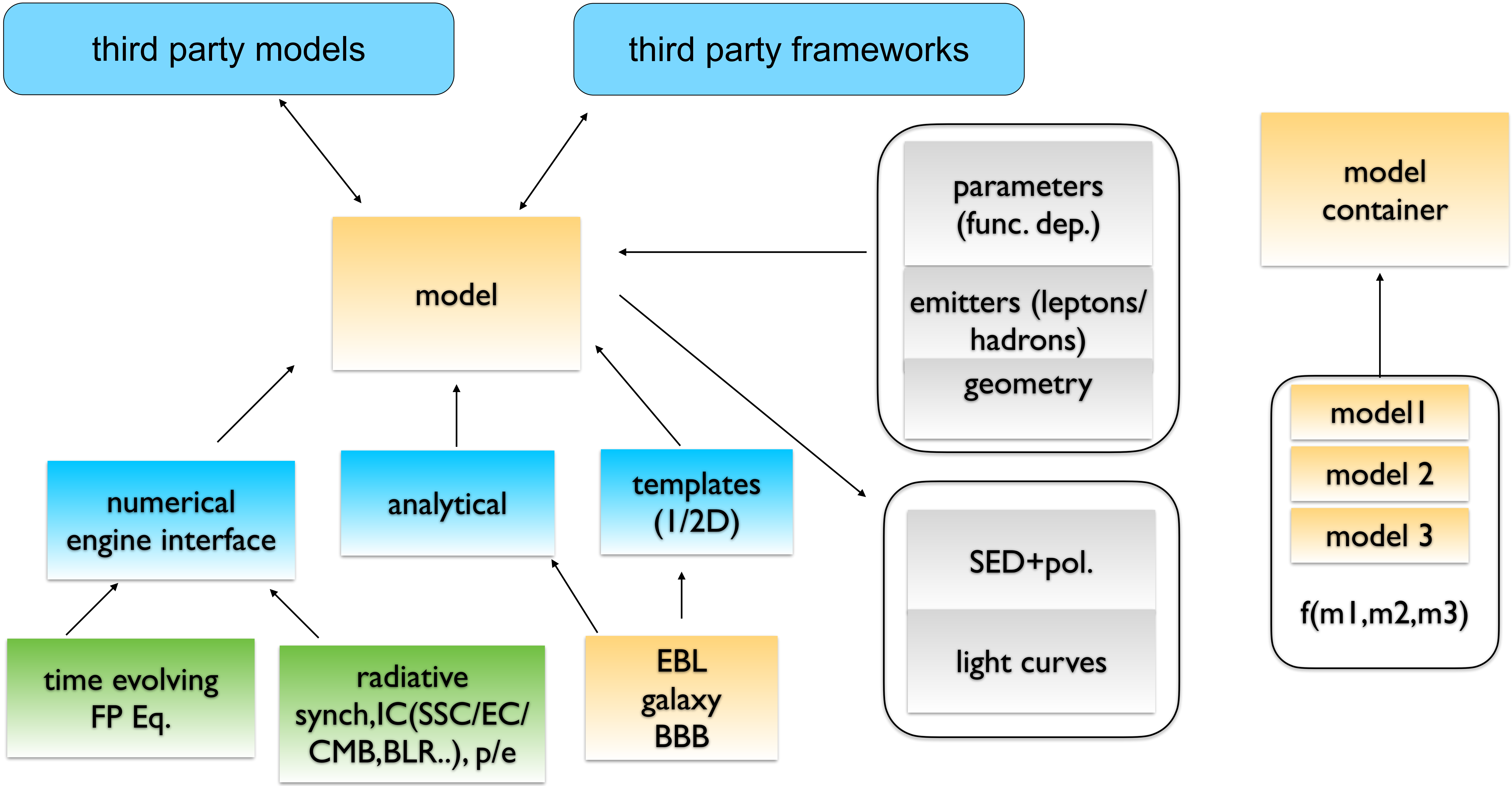
- we have large MW/MM data sets
- complex micro-physics simulations (e.g. PIC/MHD)
- **still we miss the link between micro and macro (complex micro-physics simulation still relies on phenomenological interpretation missing the constraining power from observed data)**

the requirements for an open source framework

Requirements for a framework to reproduce radiative and accelerative processes acting in relativistic jets, and galactic objects (with/without jet), allowing to fit the numerical models to observed data.



the ideal architecture for a model manager (obj oriented)



current frameworks/codes

OK IMPROVE MISSING

code	approach	sources	processes	IC	emitters	polarization	temporal evolution	emitting region	geom	language	c/c++ threads GPU	doc	plugins	install	CI/CD
naima (Python package for computation of non-thermal radiation from relativistic particle populations and MCMC fitting to observed spectra)	numerical	PWN, SNR, GRB	IC,EC, p-synch	aniso, e-,γ	leptons, hadrons(pp)	no	no	single	spher.	python	no	yes	third-party+ user defined	pip,conda	yes
GAMERA	numerical	jetted AGN, PWN,MQ, SNR	SSC,EC,Brems, pp	iso	leptons, hadrons(pp)	no	only cooling	single	spher.	python (C++)	no	yes	no	make file	no(?)
JetSeT	numerical	jetted AGN, PWN,MQ, SNR	SSC,EC,Brems, pp	iso	leptons, hadron(pp)	yes	acc+cooling+ adb exp. particles	multiple(non-inter.)	spher. exp. shell conical	python (C)	C threads	yes	third-party+ user defined	pip, conda	yes
AGNpy	numerical	jetted AGN	SSC,EC, p-synch	aniso,γ	leptons hadron(p-synch)	no	no	single	spher.	python	no	yes	third-party	pip, conda	yes
flaremodel	numerical ray tracing		SSC,synch	iso	leptons	no	only adb, cooling	single	spher/ radial	python (C)	C threads GPU	yes	third-party	pip	yes (missing tests?)
BHJet	numerical/ semi-analyt.	jetted AGN, MQ		iso		no		single	jet(?)	python (C++)	no	no	no	make file	no
AM3	numerical	jetted AGN,TDE	SSC,EC,Brems, ph	iso	leptons, hadrons(pp+ pγ)	no	acc+cooling+ adb exp. particle+phot.	single	spher.	python (C++)	no	yes	user defined	make file	yes

current frameworks/codes

OK | IMPROVE | MISSING

code	approach	sources	processes	IC	emitters	polarization	temporal evolution	emitting region	geom	language	c/c++ threads GPU	doc	plugins	install	CI/CD
naima Python package for computation of non-thermal radiation from relativistic particle populations and MCMC fitting to observed spectra	numerical	PWN, SNR, GRB	IC,EC, p-synch	aniso, e-,γ	leptons, hadrons(pp)	no	no	single	spher.	python	no	yes	third-party+ user defined	pip,conda	yes
GAMERA	numerical	jetted AGN, PWN,MQ, SNR	SSC,EC,Brems, pp	iso	leptons, hadrons(pp)	no	only cooling	single	spher.	python (C++)	no	yes	no	make file	no(?)
JetSeT	numerical	jetted AGN, PWN,MQ, SNR	SSC,EC,Brems, pp	iso	leptons, hadron(pp)	yes	acc+cooling+ adb exp. particles	multiple(non-inter.)	spher. exp. shell conical	python (C)	C threads	yes	third-party+ user defined	pip, conda	yes
AGNpy	numerical	jetted AGN	SSC,EC, p-synch	aniso,γ	leptons hadron(p-synch)	no	no	single	spher.	python	no	yes	third-party	pip, conda	yes
flaremodel	numerical ray tracing		SSC,synch	iso	leptons	no	only adb, cooling	single	spher/ radial	python (C)	C threads GPU	yes	third-party	pip	yes (missing tests?)
BHJet	numerical/ semi-analyt.	jetted AGN, MQ		iso		no		single	jet(?)	python (C++)	no	no	no	make file	no
AM³	numerical	jetted AGN,TDE	SSC,EC,Brems, ph	iso	leptons, hadrons(pp+ pγ)	no	acc+cooling+ adb exp. particle+phot.	single	spher.	python (C++)	no	yes	user defined	make file	yes

Incompleteness & Duplication

some examples from existing codes

JetSeT
Jets SED modeler and fitting Tool

an example of workflow

interoperability: using model fitting plugins (jetset and agnpy example)

Jets SED modeler and fitting Tool

```
sherpa_model_jet=JetsetSherpaModel(prefit_jet)
sherpa_model_gal=JetsetSherpaModel(my_shape.host_gal)
sherpa_model_ebl=JetsetSherpaModel(ebl_franceschini)
```

```
jetset model name R renamed to  R_sh due to sherpa internal naming convention
```

```
sherpa_model=(sherpa_model_jet+sherpa_model_gal)*sherpa_model_ebl
```

```
sherpa_model
```

▼ Model

Expression: (jet_leptonic + host_galaxy) * Franceschini_2008

Component	Parameter	Thawed	Value	Min	Max	Units
jet_leptonic	gmin	☑	470.39174855643597	1.0	1000000000.0	lorentz-factor*
	gmax	☑	2310708.197406515	1.0	1000000000000000.0	lorentz-factor*
	N	☑	7.087120469822453	0.0	MAX	1 / cm3
	gamma0_log_parab	☑	10458.36315393129	1.0	1000000000.0	lorentz-factor*
	s	☑	2.2487867709713574	-10.0	10.0	
	r	☑	0.320557142636666	-15.0	15.0	
	R_sh	☑	1.0569580559768326e+16	1000.0	1e+30	cm
	R_H	☑	1e+17	0.0	MAX	cm
	B	☑	0.0505	0.0	MAX	gauss
	beam_obj	☑	25.0	0.0001	MAX	lorentz-factor*
	z_cosm	☑	0.0336	0.0	MAX	
host_galaxy	nuFnu_p_host	☑	-10.062787651081644	-12.254122641095535	-8.254122641095535	erg / (cm2 s)
	nu_scale	☑	0.017307503006438463	-0.5	0.5	Hz
Franceschini_2008	z_cosm	☑	0.0336	0.0	MAX	

```
sherpa_model_ebl.z_cosm  = sherpa_model_jet.z_cosm
```

AGNpy

```python
# electron energy distribution
n_e = BrokenPowerLaw(
    k=1e-8 * u.Unit("cm-3"),
    p1=2.02,
    p2=3.43,
    gamma_b=1e5,
    gamma_min=500,
    gamma_max=1e6,
)

# initialise the Gammapy SpectralModel
ssc_model = SynchrotronSelfComptonModel(n_e, backend="gammapy")
```

```
ssc_model.parameters.to_table()
```

Table length=11

type	name	value	unit	error	min	max	frozen	is_norm	link
str8	str15	float64	str1	int64	float64	float64	bool	bool	str1
spectral	log10_k	-8.0000e+00		0.000e+00	-1.000e+01	1.000e+01	False	False	
spectral	p1	2.0200e+00		0.000e+00	1.000e+00	5.000e+00	False	False	
spectral	p2	3.4300e+00		0.000e+00	1.000e+00	5.000e+00	False	False	
spectral	log10_gamma_b	5.0000e+00		0.000e+00	2.000e+00	6.000e+00	False	False	
spectral	log10_gamma_min	2.6990e+00		0.000e+00	0.000e+00	4.000e+00	True	False	
spectral	log10_gamma_max	6.0000e+00		0.000e+00	4.000e+00	8.000e+00	True	False	
spectral	z	3.0800e-02		0.000e+00	1.000e-03	1.000e+01	True	False	
spectral	delta_D	1.8000e+01		0.000e+00	1.000e+00	1.000e+02	False	False	
spectral	log10_B	-1.3000e+00		0.000e+00	-4.000e+00	2.000e+00	False	False	
spectral	t_var	8.6400e+04	s	0.000e+00	1.000e+01	3.142e+07	True	False	
spectral	norm	1.0000e+00		0.000e+00	1.000e-01	1.000e+01	True	True	

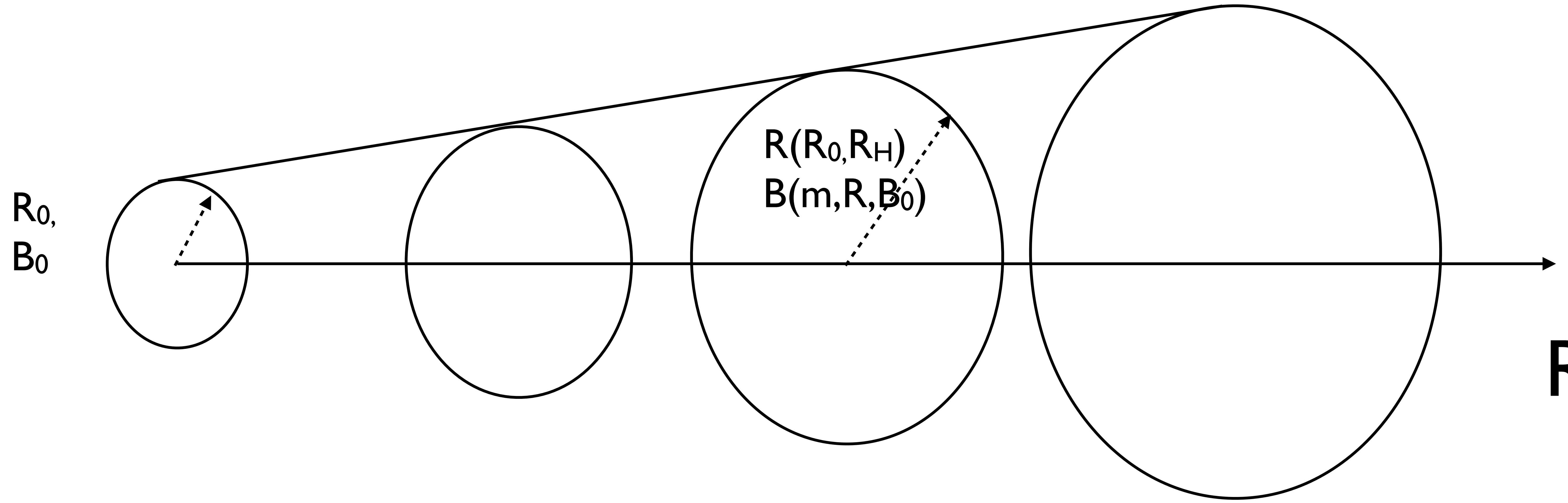
User customization: depending parameters

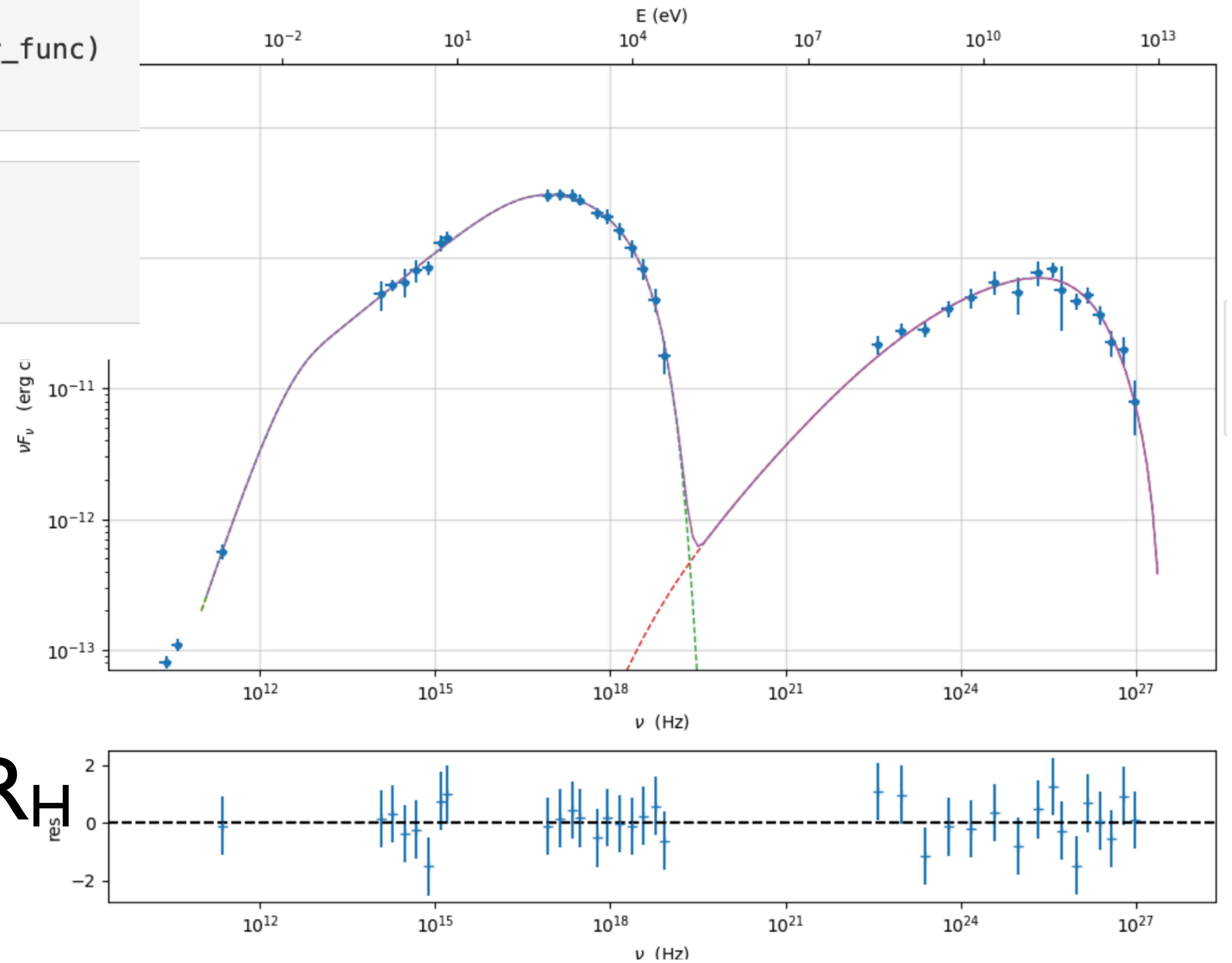
- parameters can be easily linked with functions

```
fit_model.jet_leptonic.add_user_par(name='B0',units='G',val=1E3,val_min=0,val_max=None)
fit_model.jet_leptonic.add_user_par(name='R0', units='cm', val=5E13, val_min=0, val_max=None)
fit_model.jet_leptonic.add_user_par(name='m_B', val=1, val_min=1, val_max=2)
fit_model.jet_leptonic.parameters.R0.frozen=True
fit_model.jet_leptonic.parameters.B0.frozen=True

def par_func(R0,B0,R_H,m_B):
    return B0*np.power((R0/R_H),m_B)

fit_model.jet_leptonic.make_dependent_par(par='B', depends_on=['B0', 'R0', 'R_H','m_B'], par_expr=par_func)
fit_model.parameters
```

```
==> par B is now depending on ['B0', 'R0', 'R_H', 'm_B'] according to expr:B =
def par_func(R0,B0,R_H,m_B):
    return B0*np.power((R0/R_H),m_B)
```

- make emitting region size, and B, depending on the jet opening angle, SSC sensitive to dissipation radius

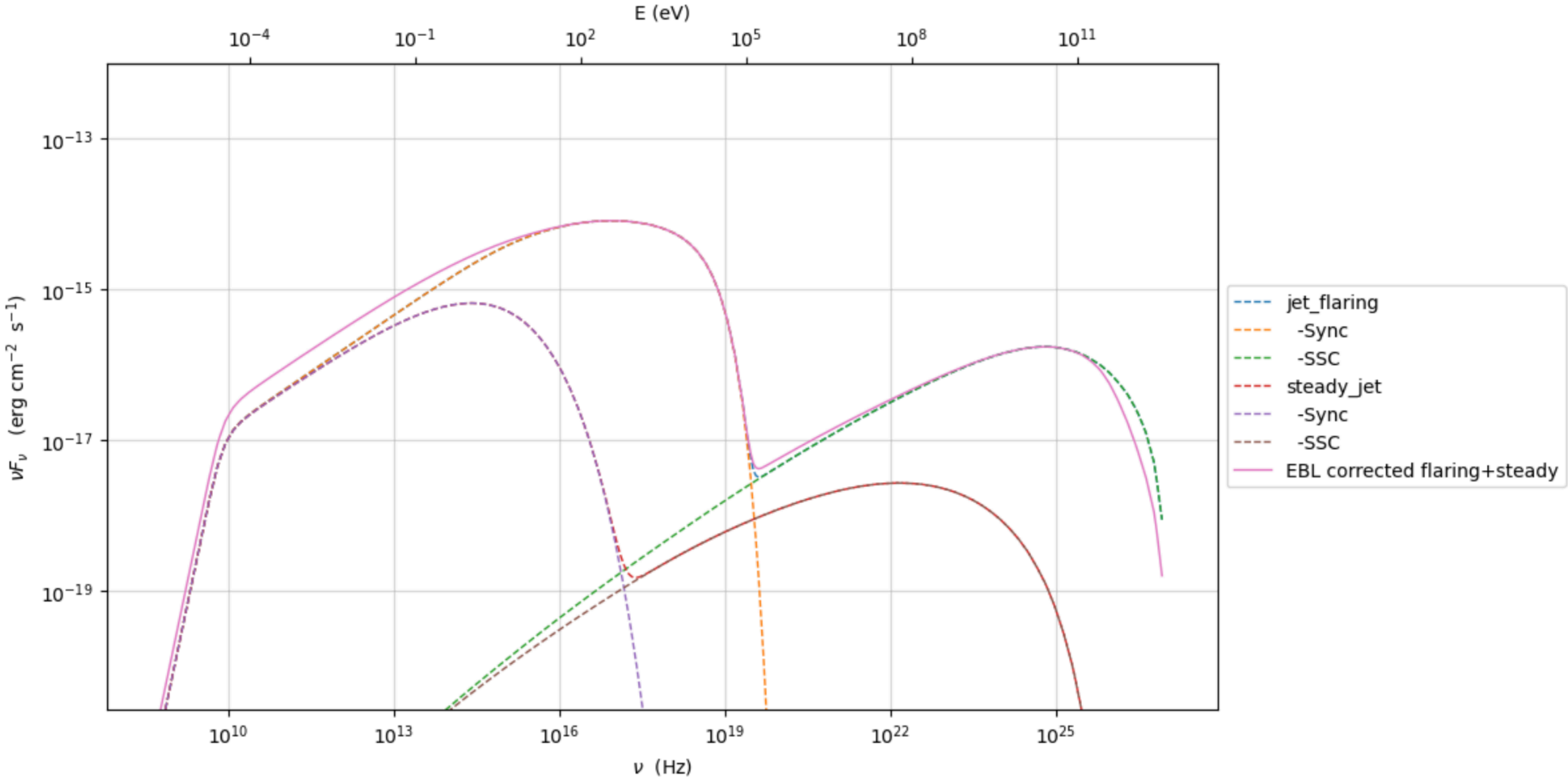


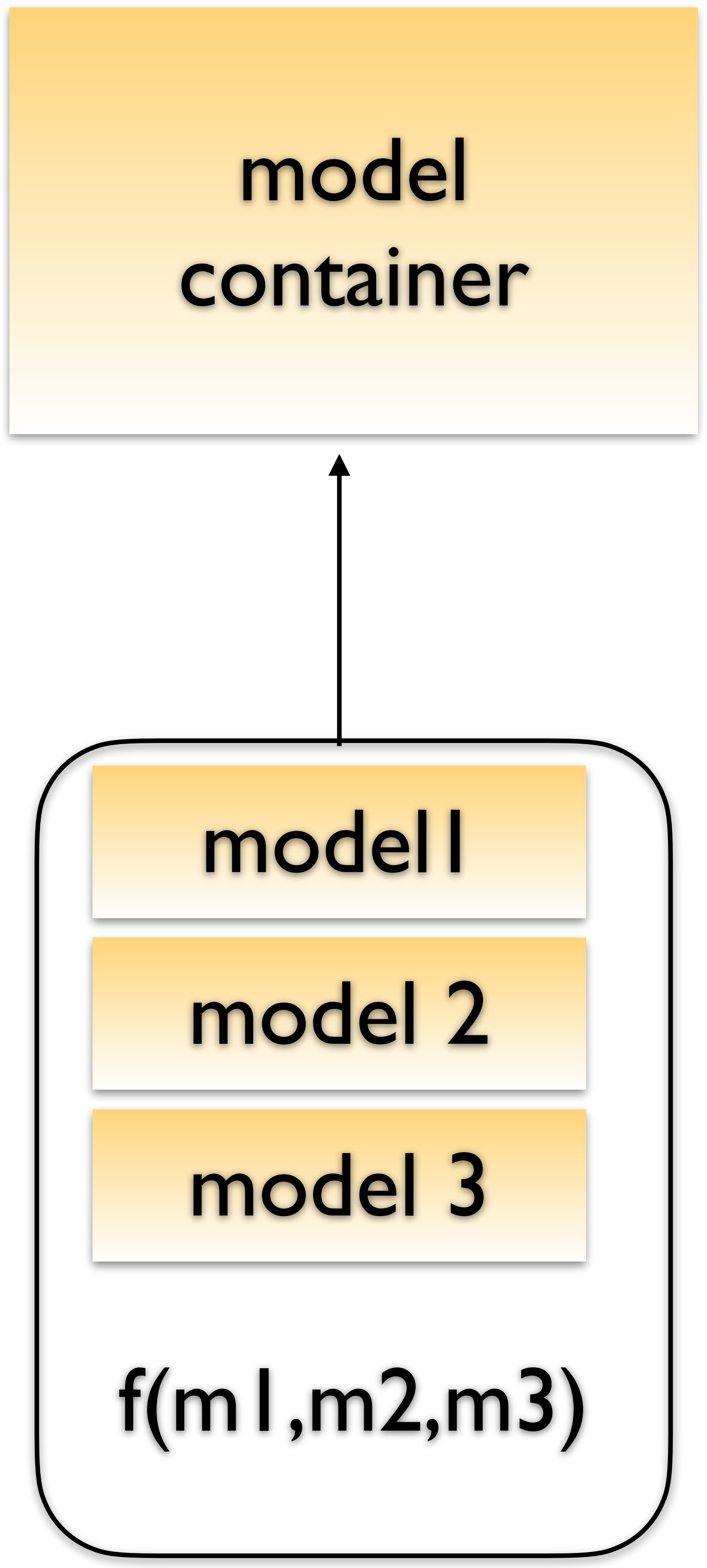


User customization: model composition

- model con be easily combined using math expression

```
composite_model.composite_expr='(jet_flaring + steady_jet) * Franceschini_2008'
```

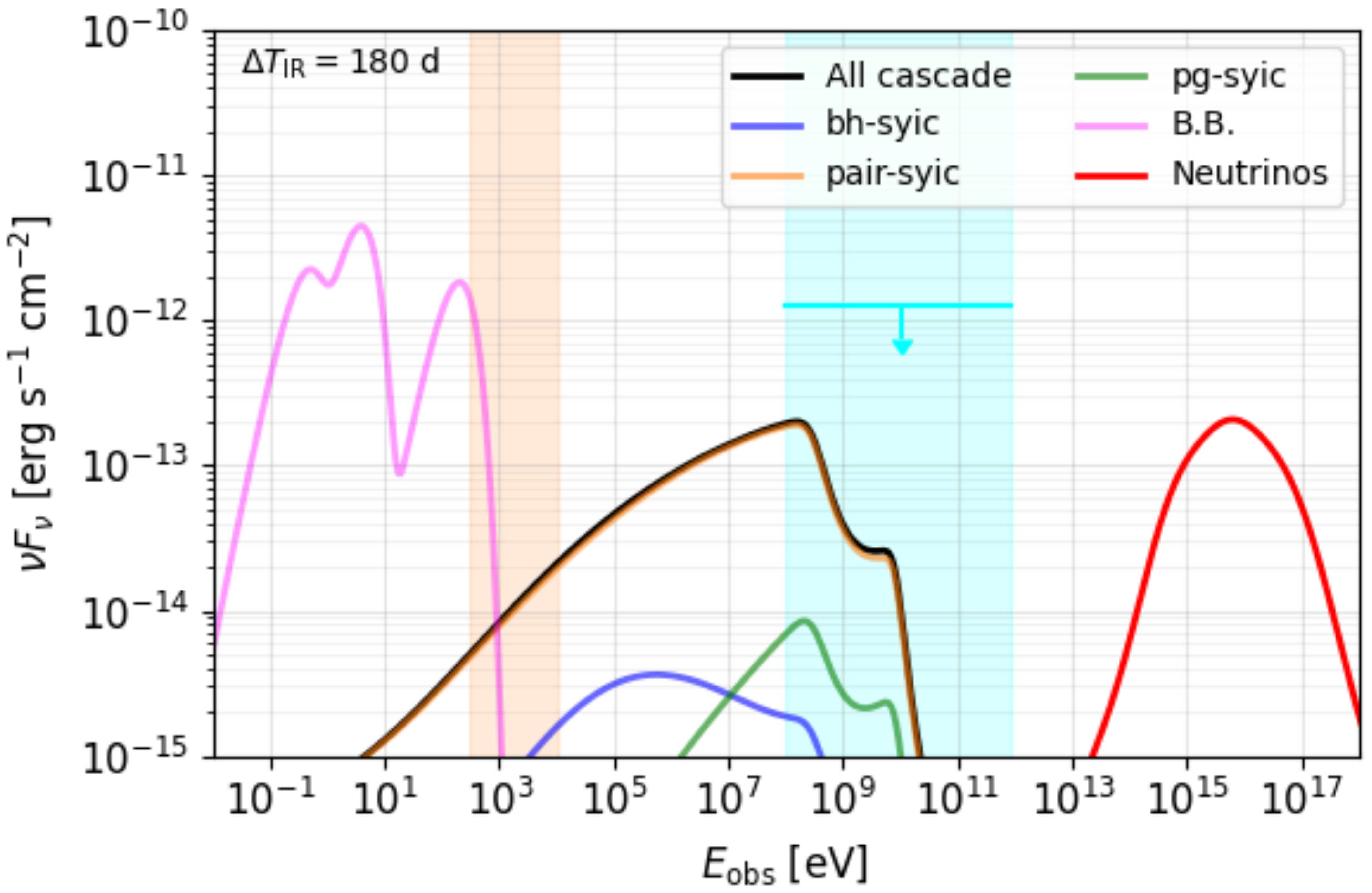




User customization: internal plugins

TDE with AM3

- use internal processes
- add temporal evolution
- get evolved eq. model



2. Define a class for generating external photon spectra

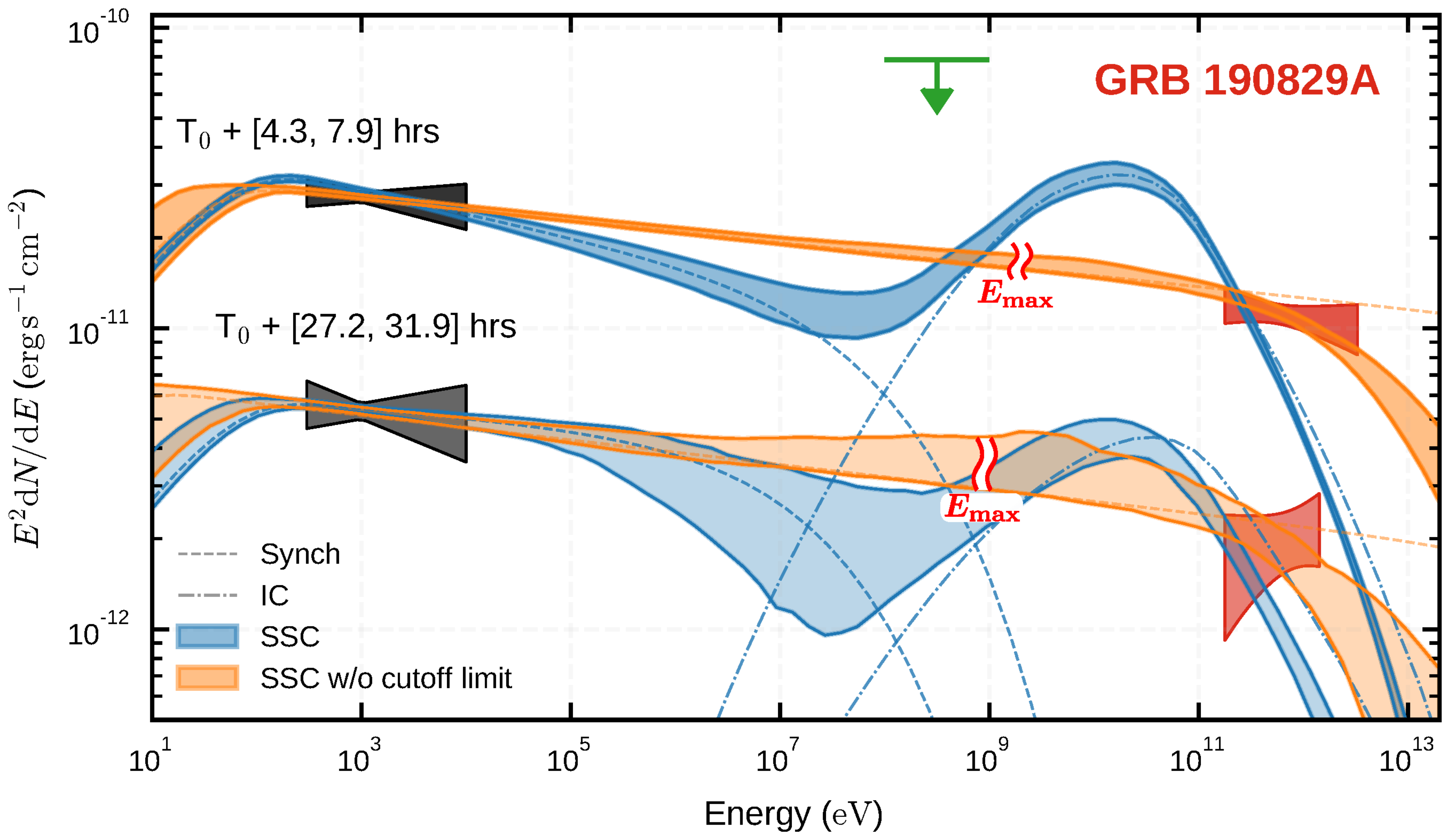
In [2]:

```python
class ExtPH:
    '''
    set_external_G_spec(Obs_time, radius, Elist, corrected_lu, timelist, Ebb)
    input: time in observer's frame, radius of the radiaton zone, energy array for external photon spectra,
           bolometric luminosity array, time array for luminosity, black body energy
    output: external photon rate spectra dN/dlogE/dt[cm^-3 s^-1]
    '''
    @classmethod
    def BlackBodySpec(cls, Energy, Temp0): #in eV, return in arbitrary units, dn/dlnE
        theta = Energy / Temp0
        tt = np.array(np.exp(theta), dtype = float)
        np.clip(tt, 1e-50, 1e100)
        spec = 1 / (tt - 1)
        if hasattr(Energy, "__len__"):
            spec[spec < 1e-20] = 0
        return spec * Energy**2



    @classmethod
    def set_external_G_spec(cls, Obs_time, radius, Elist, corrected_lu, timelist, Ebb):
        IntegralFlux = linear_interpolation(Obs_time, timelist, corrected_lu)
        EnergyUp = Ebb * 1e6
        EnergyLow = Ebb / 1e6
        dLogE = np.log(EnergyUp / EnergyLow) / 200
        EnergyList = np.exp(np.arange(np.log(EnergyLow), np.log(EnergyUp), dLogE))
        dEnergy = EnergyList * np.exp(dLogE) - EnergyList
        integral = sum(dEnergy * cls.BlackBodySpec(EnergyList, Ebb))
        return  IntegralFlux / integral * cls.BlackBodySpec(Elist/(1+redshift), Ebb) / (4*np.pi/3*radius**3) * erg2GeV * 1e9/3

def linear_interpolation(x, index_array, interp_array):
    '''
    1D linear interpolation
    input: x, x-array, y-array
    '''
    length = len(index_array)
    if len(index_array) != len(interp_array):
        print ("Interpolation error!")
        return 0

    elif (x < index_array[0]) or x > (index_array[length-1]):
        return 0
    else:
        for i in range(length-1):
            if x <= index_array[i+1] and x >= index_array[i]:
                return interp_array[i] + (interp_array[i+1] - interp_array[i])/(index_array[i+1] - index_array[i]) * (x - index_array[i])
```

https://am3.readthedocs.io/en/latest/examples/tde_example.html naima

Python package for computation of non-thermal radiation from relativistic particle populations and MCMC fitting to observed spectra

User customization: internal plugins



https://github.com/Carlor87/GRBmodelling

Science 2021, 372, 1081–1085 H.E.S.S. Collaboration

```
import numpy as np
import matplotlib.pyplot as plt
from astropy.table import Table,hstack
import astropy.units as u
from astropy.io import ascii
import naima

from grbloader import *
```

and in the last line we import our GRB class.

After this we set some physical parameters, we read the datapoints and create an astropy table for them with the following:

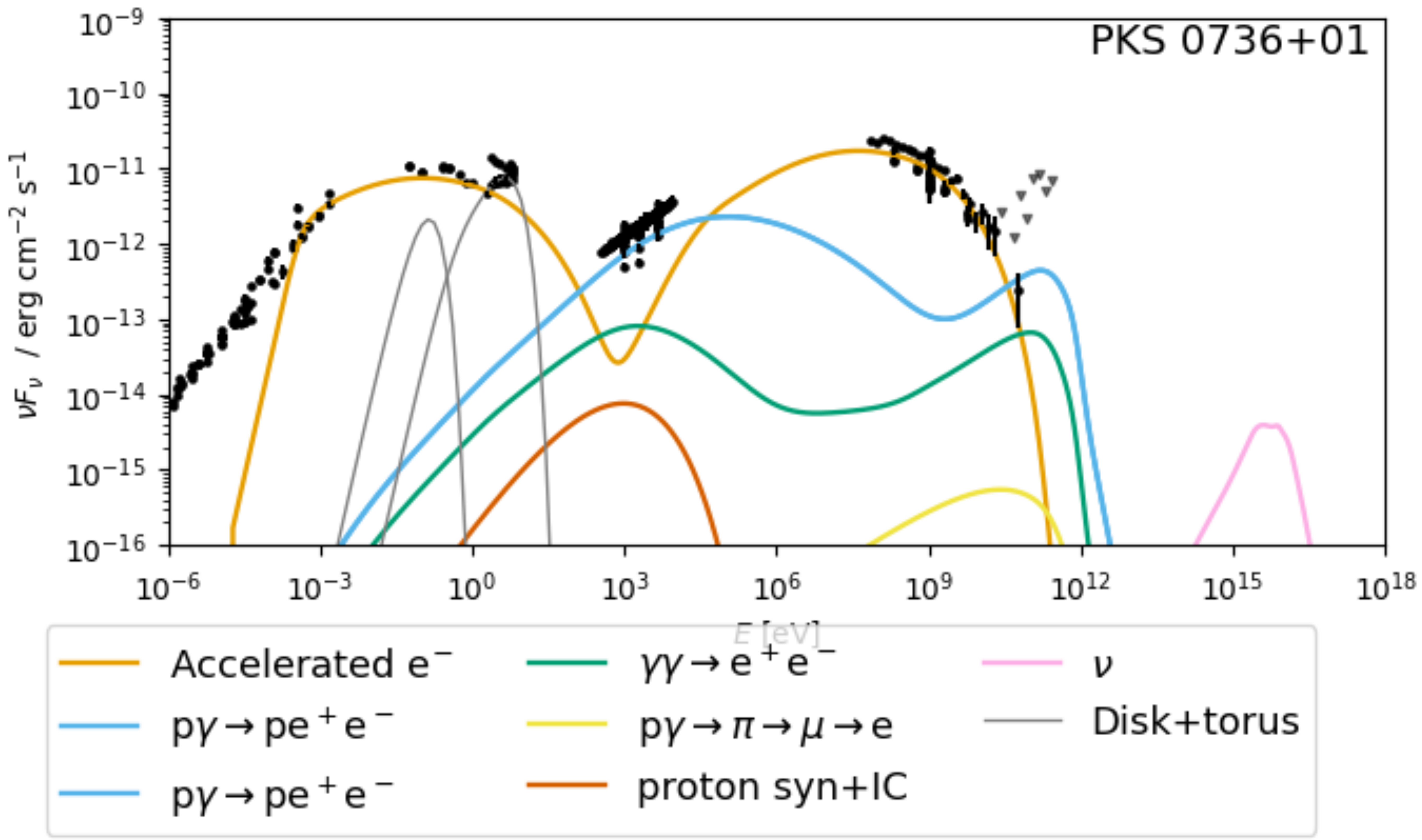
```
Eiso = 8e53    # erg
density = 0.5  # cm-3
redshift = 0.4245
tstart = 68    # s
tstop = 110    # s

tab = ascii.read("magic_int1_points.txt")
newt = Table([tab['energy'], tab['flux'], tab['flux_hi']-tab['flux'], tab['flux']-tab['flux_lo
          names=['energy', 'flux', 'flux_error_hi', 'flux_error_lo'])
```

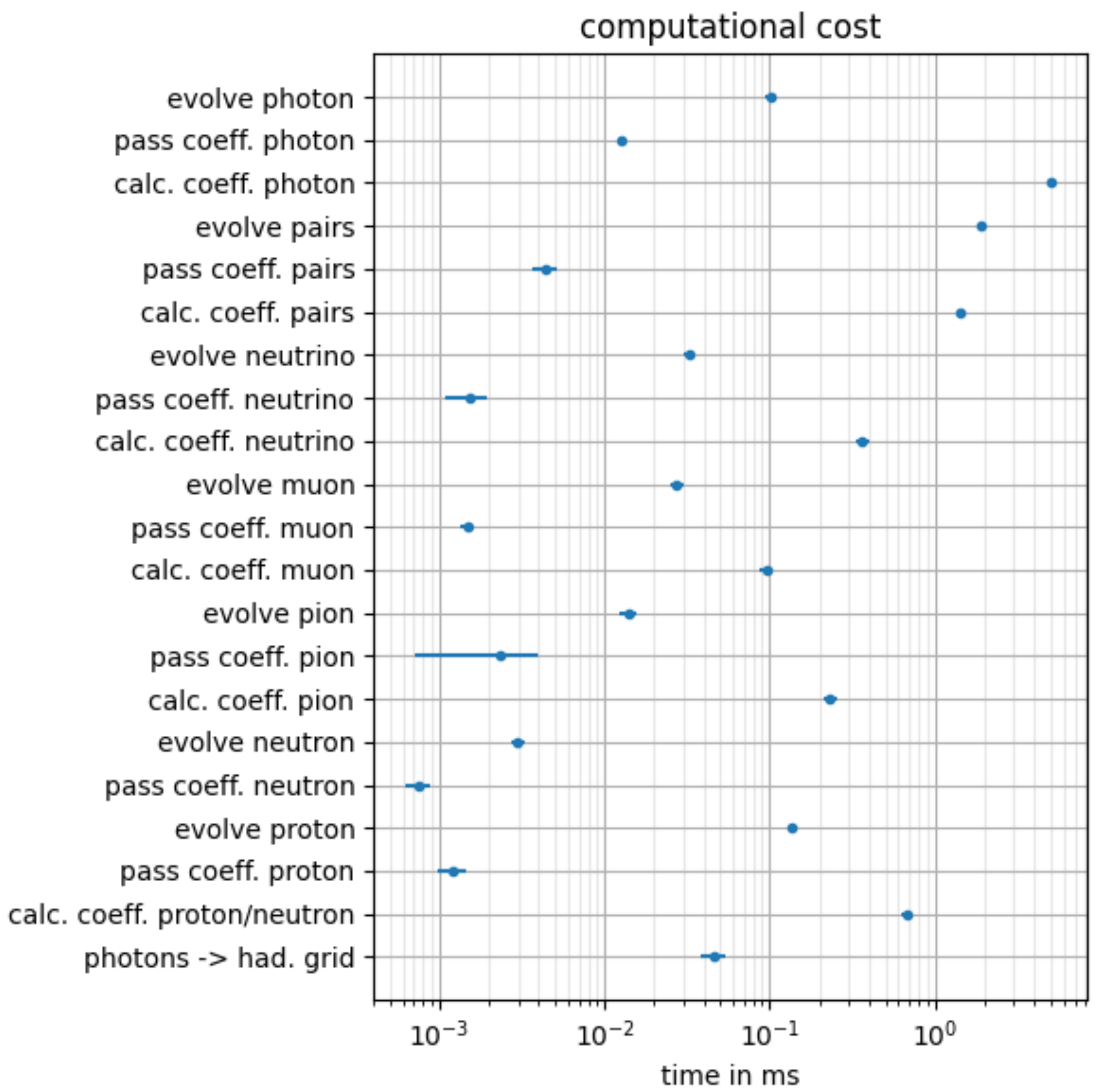
Now we have all that is needed to initialise the GRB class and this is done with

```
magicgrb = GRBModelling(Eiso, density, [newt], tstart, tstop, redshift,
                        [np.log10(0.07), -1.53, 2.87, 0.45, 0.01],
                        ['log10(eta_e)', 'log10(Ebreak)', 'Index2', 'log10(Ec)', 'log10(B)'],
                        scenario='ISM',
                        cooling_constrain=False)
```

GRB with naima

lepto-hadronic one-zone adb. exp. and cooling with AM3



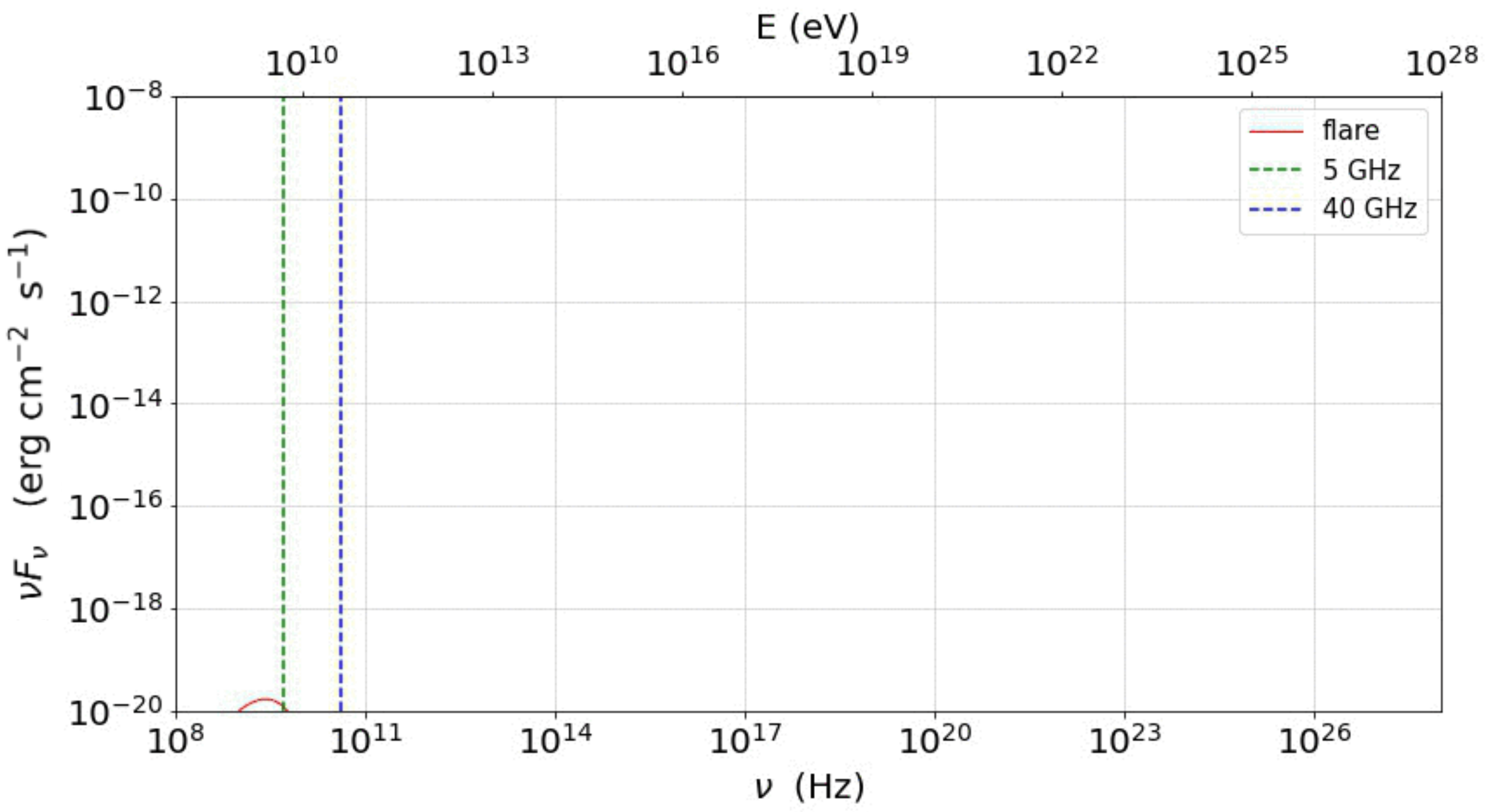


https://am3.readthedocs.io/en/latest/examples/blazar_detailed_example.html

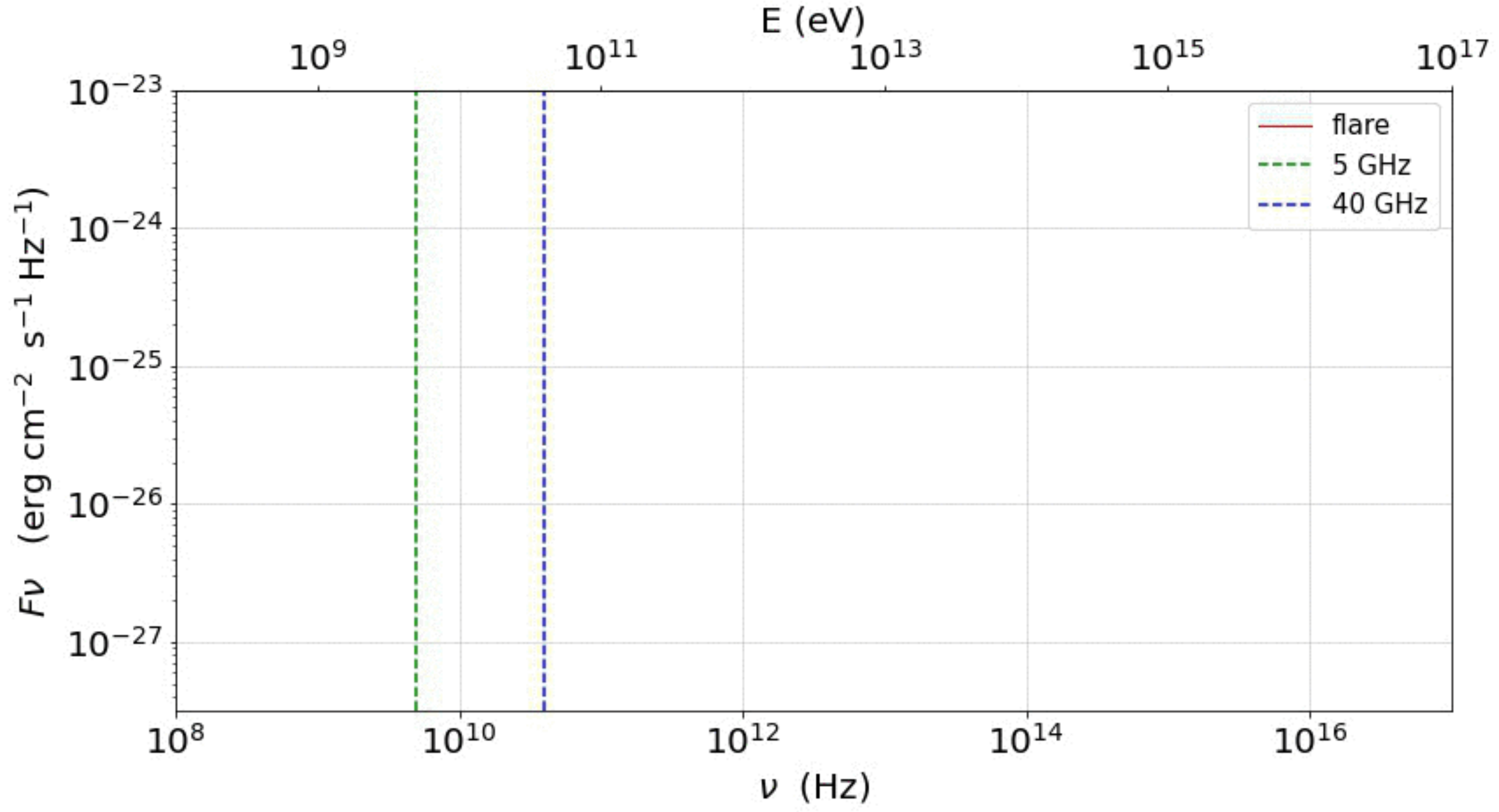
radio-gamma delay: adb. exp. + cooling + acc.

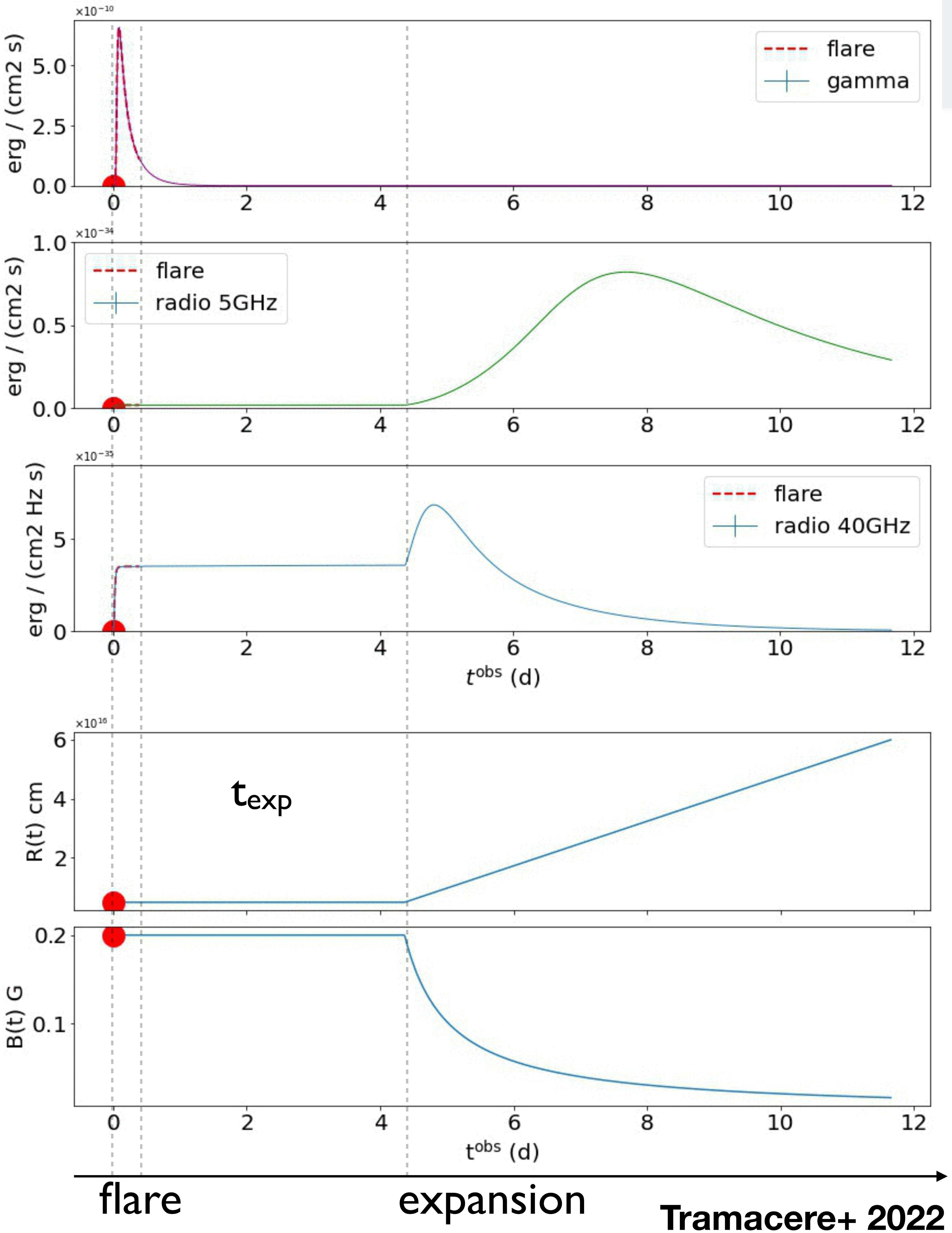
JetSeT
Jets SED modeler and fitting Tool

Tramacere+ 2022



- **duration~3x10^7 s (blob frame)~ 11 d obs**
- **t_{exp}=1x10^7 s**
- **β_{exp}=0.1c**

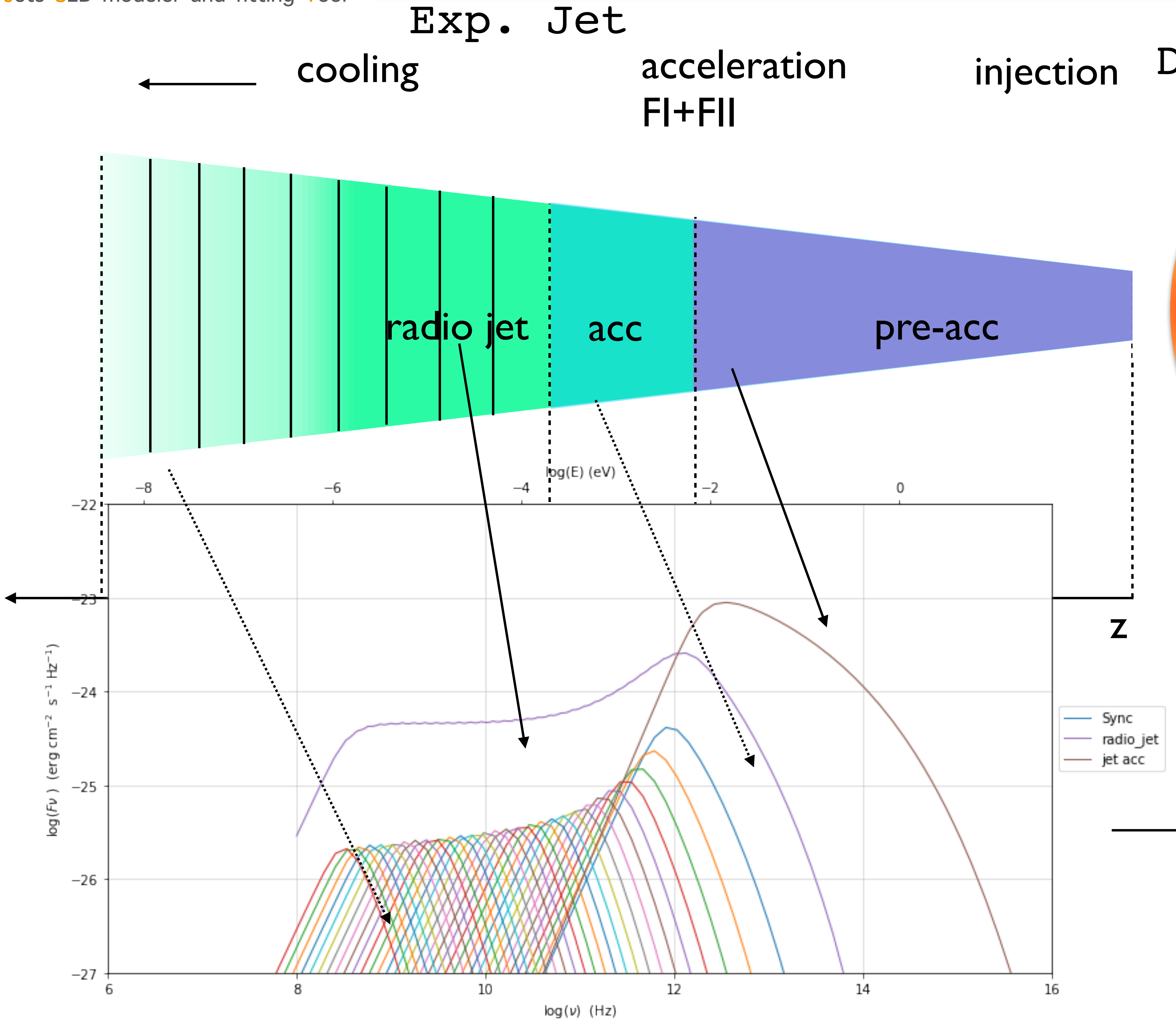




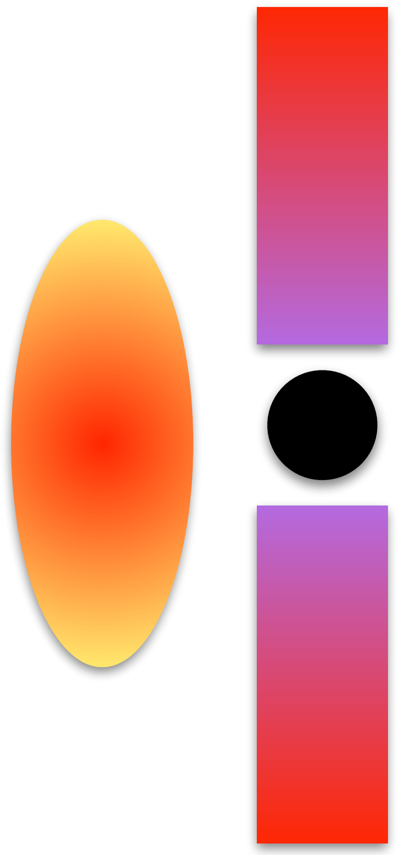
Tramacere+ 2022

adb exp video

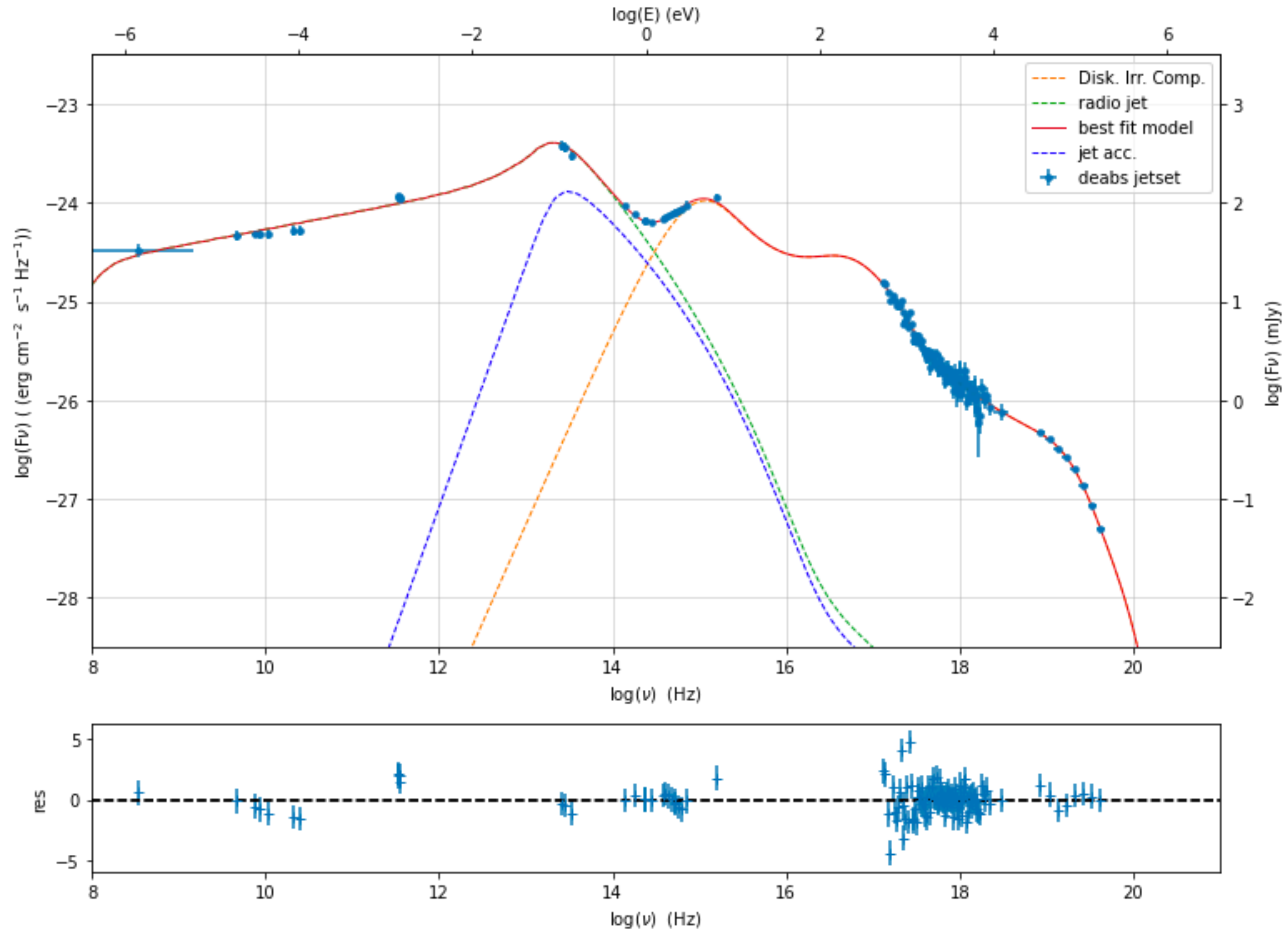
User customization: internal plugins



DiksIrrComp+Corona



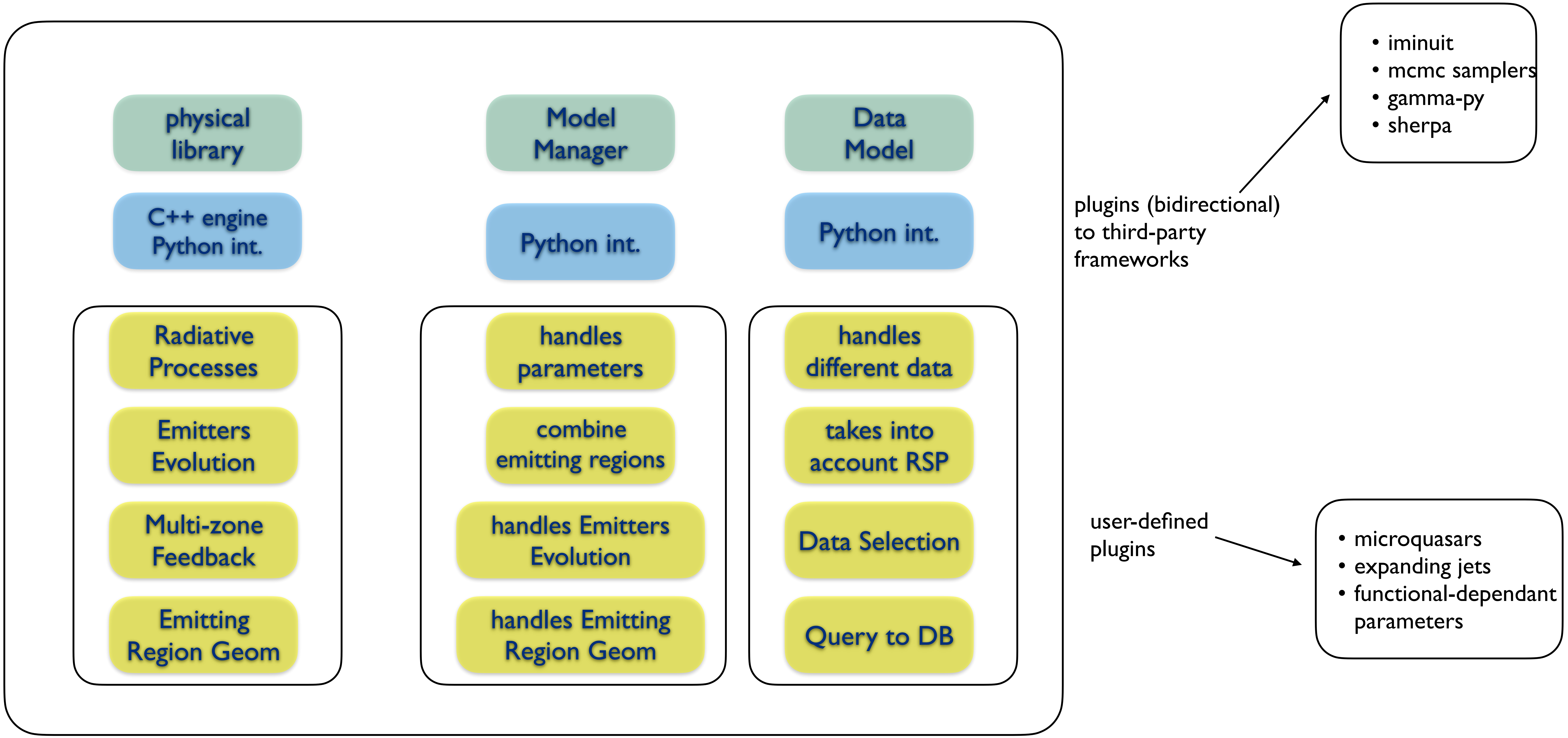
MQ plugin with JetSeT



Rodi,Tramacere+ ApJ2020

https://github.com/andreatramacere/mq_jet

JetSeT the ideal framework: **move from competition to cooperation**



no/low duplication here!

diversification is possible, but…

something already present in sherpa, gamma-py and 3ML

most relevant todos

- feedback among multi-zone components, and raytracing (eg. synch self-abs in stratified geometries)
- disk models, mostly phenom. no actual connection to jet powering
- more interaction between GRB and Blazar communities (developed different expertises, but so far walked on parallel patters), both deal with jets, even though within different scenarios, but systematic differences might provide interesting orthogonal constraints
- customize geometry
- speedup computation time and solve degeneracy for time-evolved model fitting: AI does not solve the problem:
    - works only at equilibrium and is based on templates
    - some black box in the middle
    - for time-dependent parameters, the volume of the templates will become huge
    - is not aware of the underlying physics, whilst frequentist/Bayesian methods get direct feedback on how a model reacts to parameters changes based on the implemented physics, and according to actual state of the system, AI template-based, is linking a finite combination of parameters/template at the equilibrium, actually not following the impact of a parameter at given time

last but not the least

- scientist developing software, needs rewards, currently, metrics push toward a purely competitive approach
- Publishers should favour/promote/push submitters to provide fully reproducible modelling