SEVN2 Walkthrough

For any question/comment:
giuliano.iorio@unipd.it
giuliano.iorio.astro@gmail.com

Resources

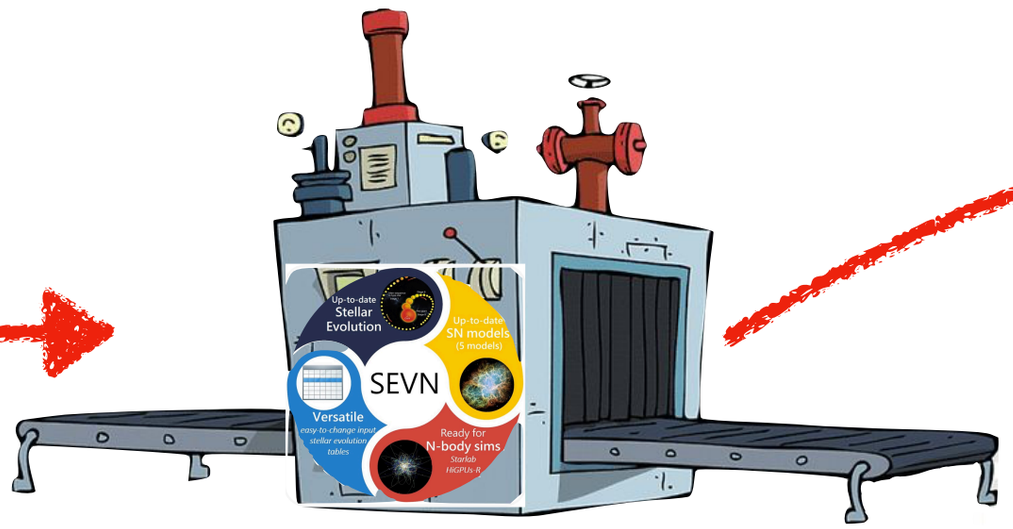
This is an “hands-on” walkthrough showing a complete example of SEVN usage from the installation to the analysis of the outputs.

For a more detailed descriptions of SEVN, its options and its outputs:

- Wiki page on the gitlab repository
- User guide in the folder *resources* shipped with SEVN
- Examples of output analysis
  in the folder *resources/tutorial*
  and in the folder *resources/SEVN_walkthrough*
- Contact me (giuliano.iorio.astro@gmail.com)

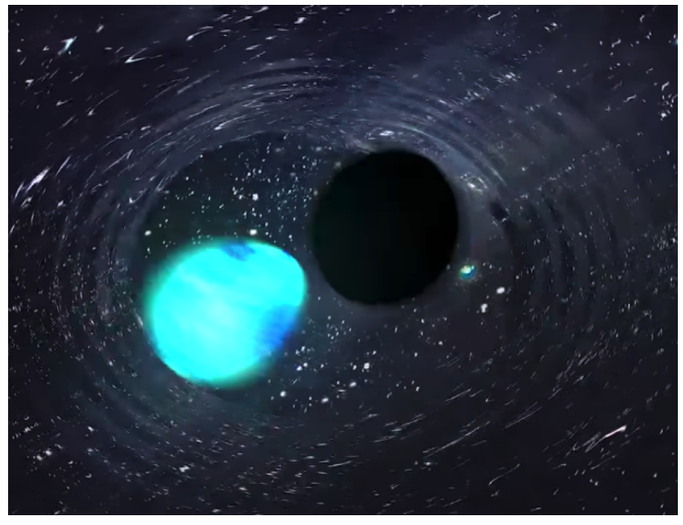
SEVN2 in a nutshell

Initial conditions (binary systems)
Masses, Metallicity, Binary parameters



Evolve with SEVN

Systems of interest
e.g. Black Hole binaries, Xray binaries



Overview

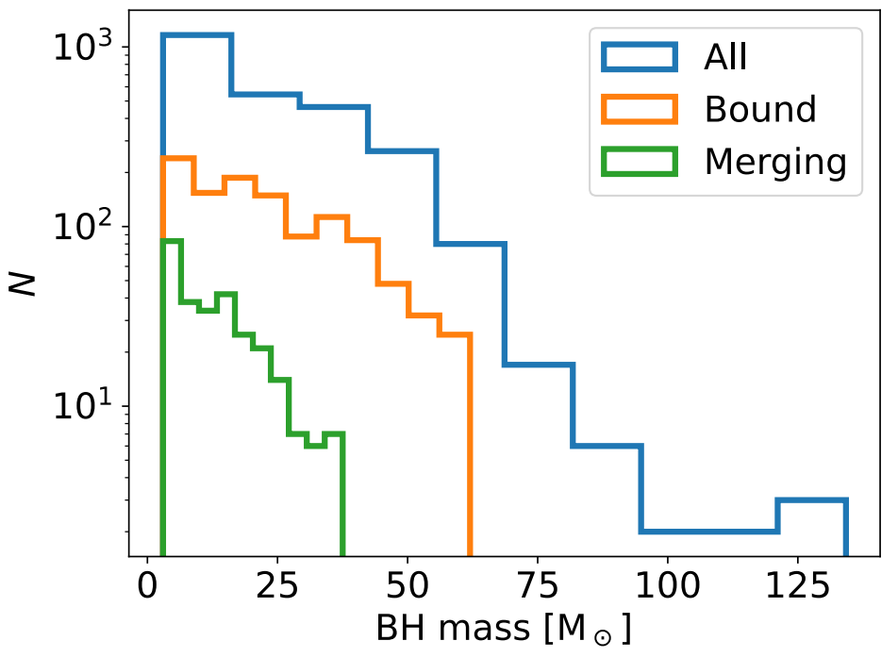
Get SEVN

Install SEVN

Initial Conditions

Run SEVN

Analyse SEVN outputs



Get SEVN

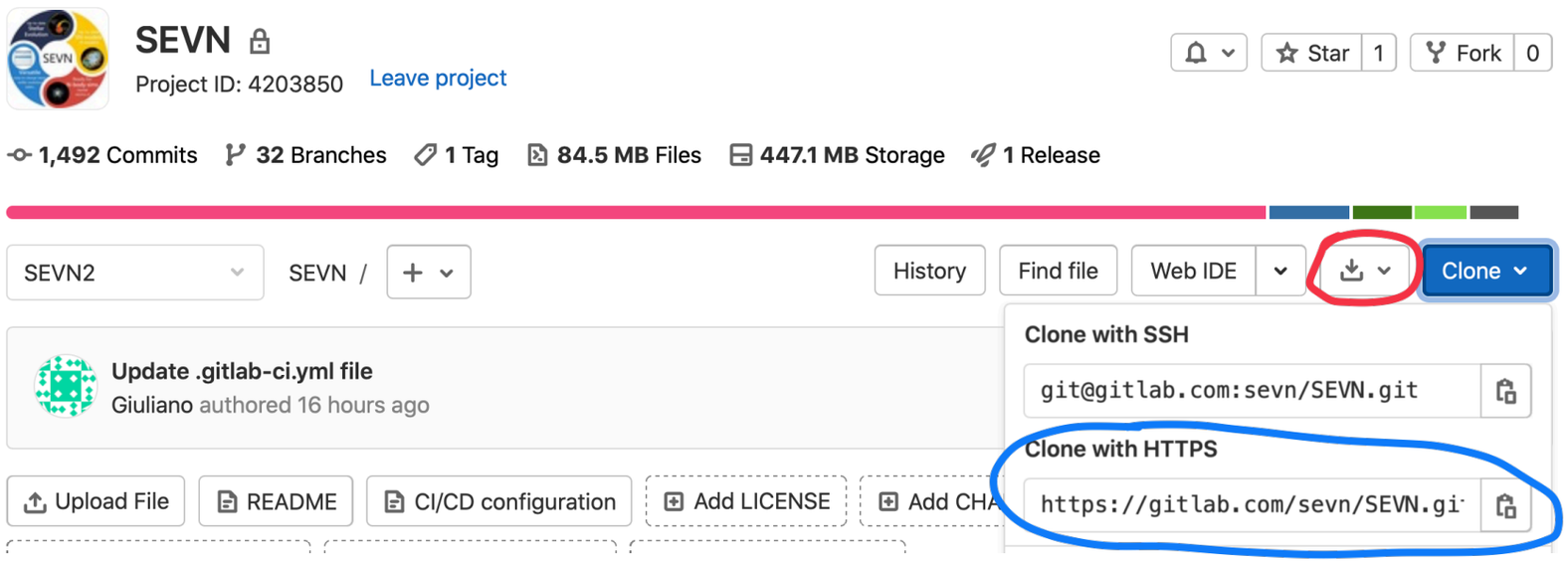
SEVN2 is in a private Gitlab repository:
https://gitlab.com/sevn/SEVN

Temporary User

Username: sevn.guest

Password: Maelstrom

Get SEVN



Two methods to get SEVN2 from the repository:

1- **Get the source file**: use the red highlighted icons to download the source in a given archive format or use this link.

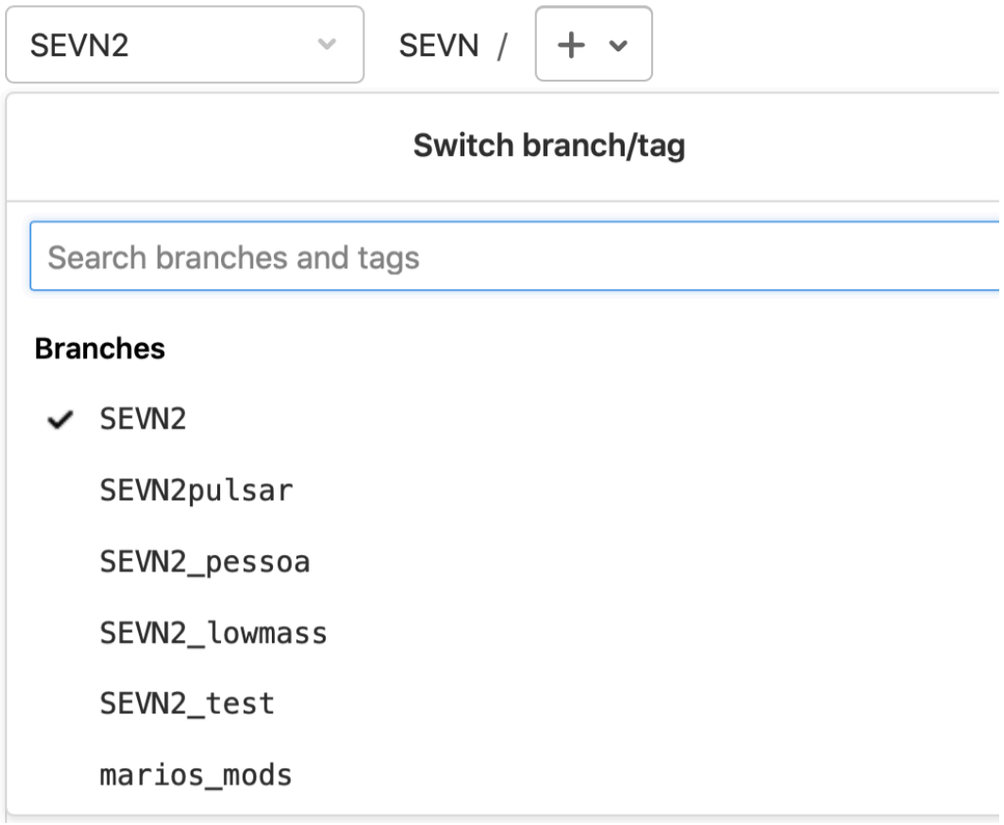
2- **Clone with git (git required, recommended)**: if you have git, it is possible to clone locally the repository using the blu highlighted link (NOT use SSH), i.e.

```
git clone https://gitlab.com/sevn/SEVN.git
```

The local repository can be updated with:

```
git pull
```

SEVN2 branches



The SEVN repository has many branches, (`git branch -a` to show all the branches or see this link). **The most updated stable branch** for which this tutorial is made **is SEVN2.** All the other branches contains experimental features or are not updated. **NOT USE THEM**.

Get SEVN

At this point you should have a SEVN folder

Have a look at its content, in particular:

- *compile.sh*, installation script
- *run_scripts/run.sh*, script to run SEVN
- *resources/SEVN2_user_guide*
- *resources/SEVN_walkthrough*

Install SEVN

Requirements:

- gcc >4.8, type `gcc –version`
- Cmake >2.8, type `cmake –version`

Setting the "environment":

- Create a folder where to run SEVN and collect its outputs
- Move the *compile.sh* and *run.sh* in the folder, include also the file with the initial conditions
  (you can find a test ic file in *resources/ICs/imf2.3_n1e4.dat*)

Install:

- Write the correct path of the SEVN folder in the compile.sh assigning it to the variable SEVN (e.g. SEVN="<path_to_SEVN_folder>")
- Execute the script, *chmod u+x compile.sh; ./compile.sh*

Input file

The binary systems to simulate has to be listed in an ascii file.
Each row must contain a single binary system.
Input order:

M_1	Z_1	Ω_1	sn_1	$t_{\mathrm{start},1}$	M_2	Z_2	Ω_2	sn_2	$t_{\mathrm{start},2}$	a	e	t_{end}	dt_{out}	$seed$ [optional]

1 is for the primary star, 2 for the secondary

```
5.141 xxx 0.0 delayed 0.6 4.3193 xxx 0.0 delayed 0.6 188.4520100826892 0.3954 xxx xxx 2037559752
8.4352 xxx 0.0 delayed 0.6 4.2054 xxx 0.0 delayed 0.6 96.30132668611016 0.3944 xxx xxx 1452206127
```

M: Mass of the star Msun
Z: Metallicity
Spin: stellar rotation (angular velocity over critical angular velocity)
sn: Supernova formalism
tstart: starting age of the star in Myr
a: Semimajor axis of the binary in Rsun
e: Eccentricity of the binary
tend: ending time of the simulation (Myr or option)
dtout: printing time interval (Myr or option)
Seed: random seem (for reproducibility)

Wiki page about SEVN input at this link:
https://gitlab.com/sevn/SEVN/-/wikis/SEVN-V2/run-the-code#input

Run SEVN

We run SEVN using the *run.sh* script

- Open the script and set the input option:

`SEVN=<path_to_the_SEVN_folder>` Executable

`NTHREADS=2` Parallel threads

`DTOUT=events` Output policy (events: print in output just when something happens)

`LISTBIN=<path_to_the_ic_file>` File containing the initial conditions (ic)

`TEND=end` Ending time of the simulations (end: stop after both stars are Remnants)

`TSTART=zams` Starting ages for all the stars

`RSEED=true` If true the ic file contains an extra column with random seeds

`SCOL=Mass:Radius:Luminosity:Temperature:Phase:RemnantType`

`BCOL=BWorldtime:Semimajor:Eccentricity:GWtime:BEvent`

Properties to print in output

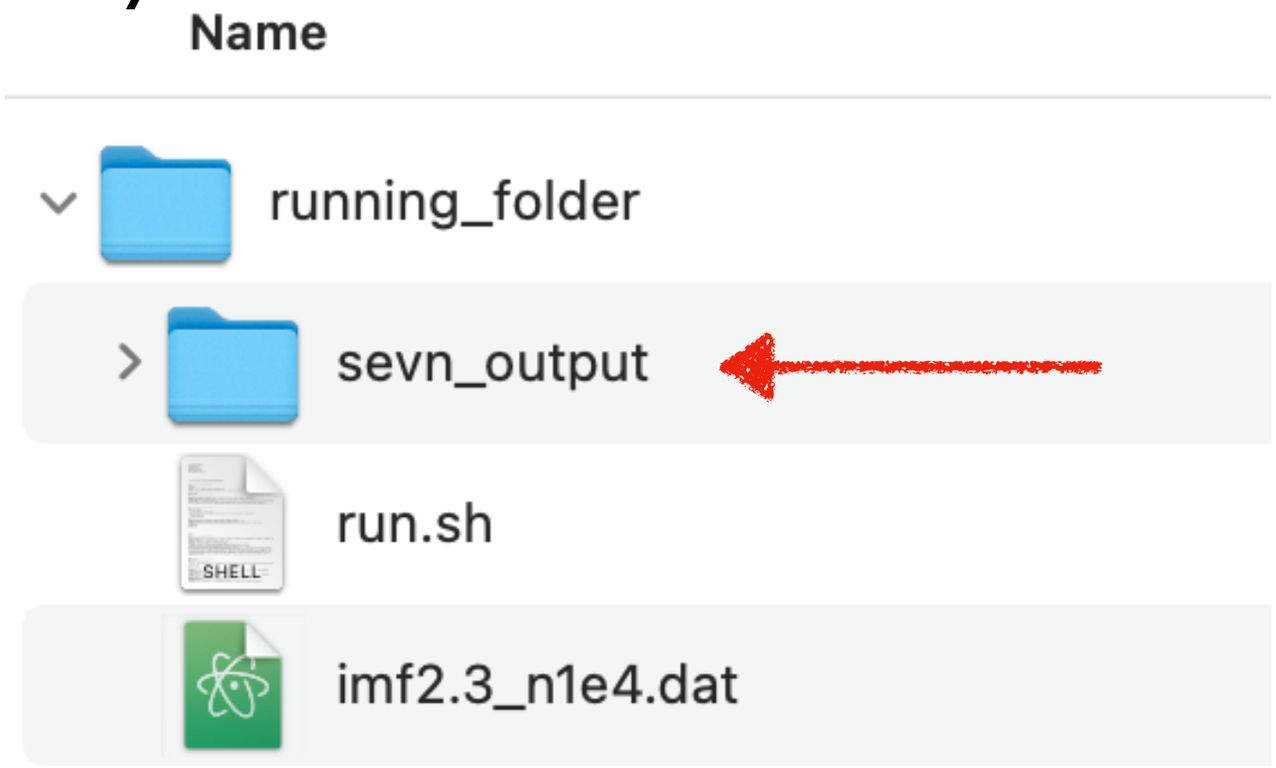
`Z=0.0002` Metallicity

Wiki page about SEVN input at this link:
https://gitlab.com/sevn/SEVN/-/wikis/SEVN-V2/run-the-code#input

- Run the script, *chmod u+x run.sh; ./run.sh*

Run SEVN

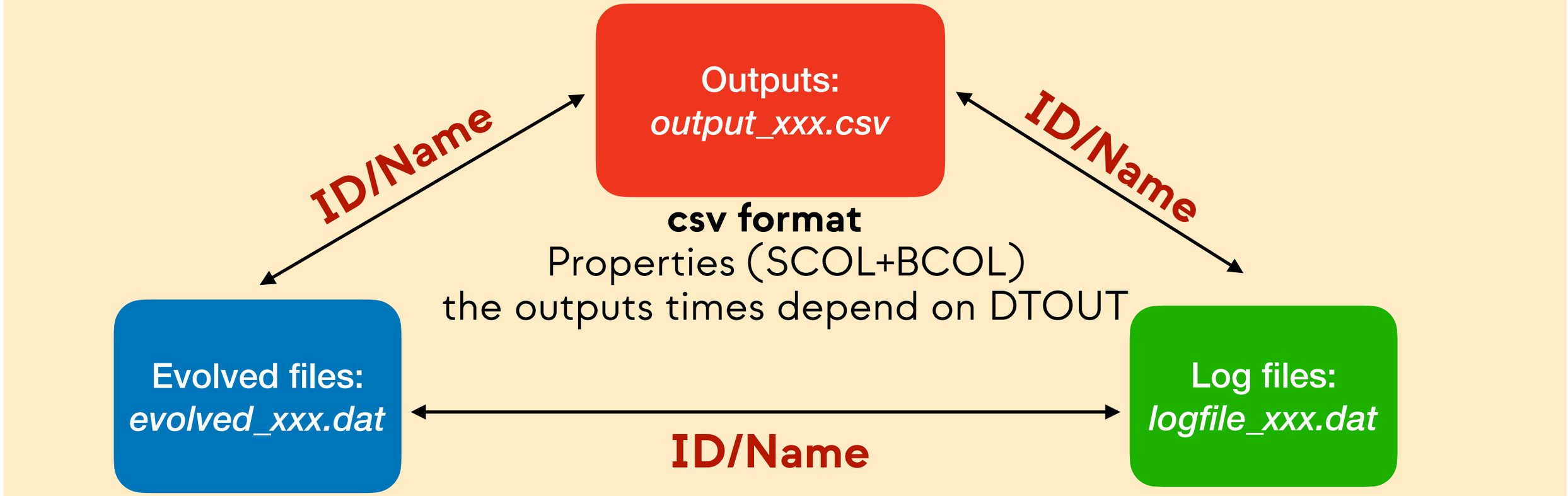
At this point you should have a sevn_output folder



Have a look at its content, there are several files

Analysing outputs

A SEVN2 run produces several output files, the most important are:



xxx: is the number of thread that produced that file, e.g. if NTHREADS=2: output_1.csv, output_2.csv, evolved_0.dat, evolved_1.dat, logfile_0.dat, logfile_1.dat

ID/Name: each system is univocally identified by two numbers (ID and Name) , this information is present in all the sevn outputs.

ID: long integer, it indicates the position of the system in the IC file

Name: long integer, it is a random number

Analysing outputs

A typical analysis of SEVN outputs consists on

- **Filtering**: e.g. we want all the systems that end their life as bound BH-BH

- **Joining**: e.g. we want to add to the outputs the information of the initial masses (from evolved files) and on the number of Common Envelopes (from log files)

- **New quantities**: use the information from SEVN to estimate new quantities, e.g. solve Peters64 equations to find the merging times of a BH-BH system

Useful tools (for Python):

Dask.dataframe

Pandas

Reg[ular]
Ex[pression]
*

re builtin module

See the User Guide and the wiki
for examples using pandas, dask and regex

Pandas/Dask

- **Import pandas**: `import pandas as pd`
- **Read a csv file**: `df=pd.read_csv("sevn_output/output_0.csv")`
- **Columns name**: `df.columns()`
- **Read a ascii file**:

```
df2=pd.read_csv("sevn_output/evolved_0.dat",sep="\s+")
df2=df2.rename(columns={"#ID":"ID"})
```

- **Access a column**: `df.Mass_0 or df["Mass_0"]`
- **Use conditions as filter**:

```
idx = (df.Phase_0==7) & (df.Phase_1==7)
filtered_df = df[idx]
```

- **Join two dataframes**:

```
dfxm = filtered_df.merge(df2, on=["ID","name"], how="left)
```

- **Import Dask**: `import dask.dataframe as dd`
- **Read csv files**: `df=dd.read_csv("sevn_output/output_*.csv")`

dask dataframes can be used exactly as pandas data frame, to return a final pandas data frame from a dask dataframe

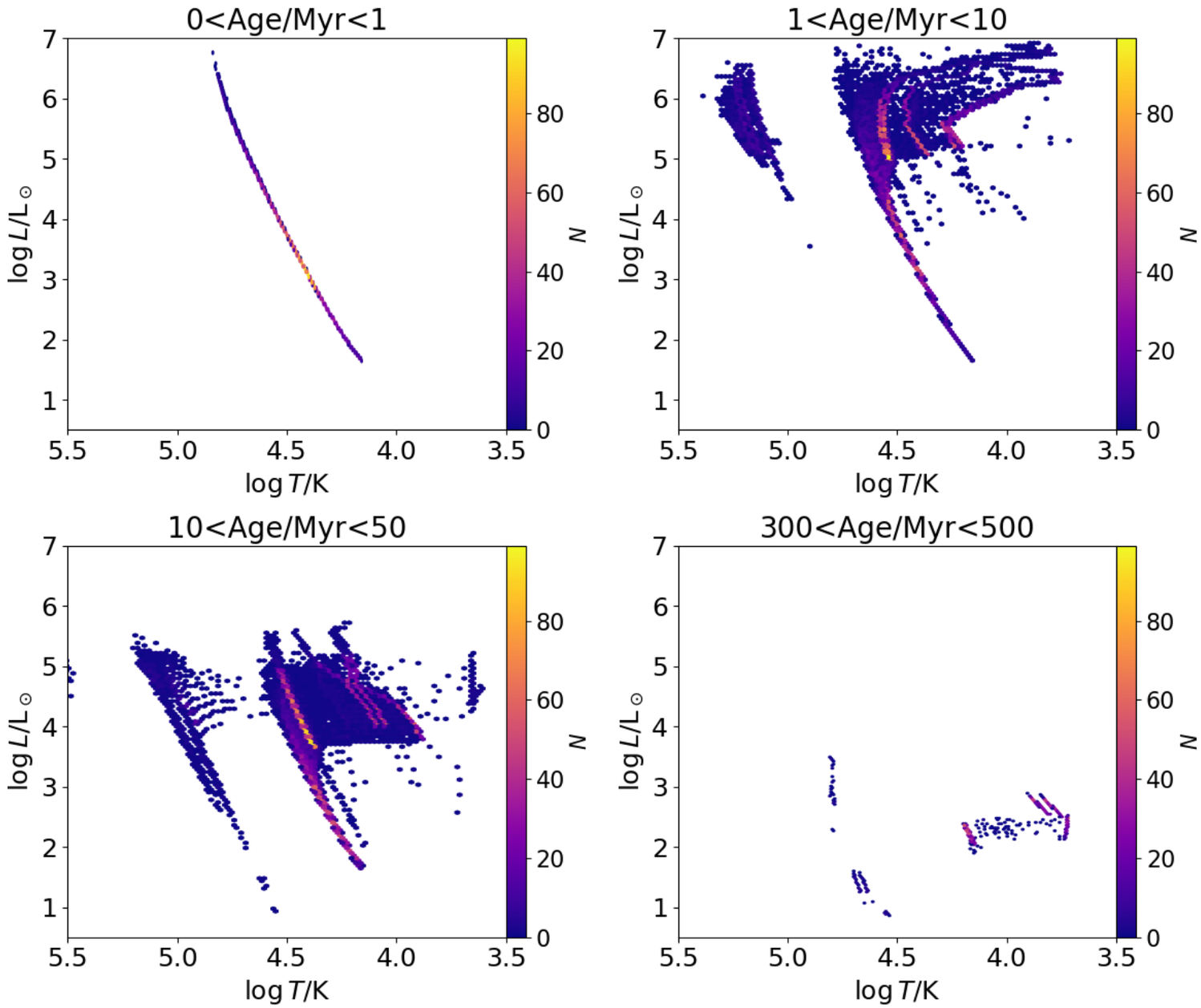
```
dfp=dfd.compute()
```

Analysis 0

We want to plot the HR diagram of all the stars in the outputs files at different time intervals.

Remember:

- The stars (not remnants) can be selected imposing conditions on the Phase

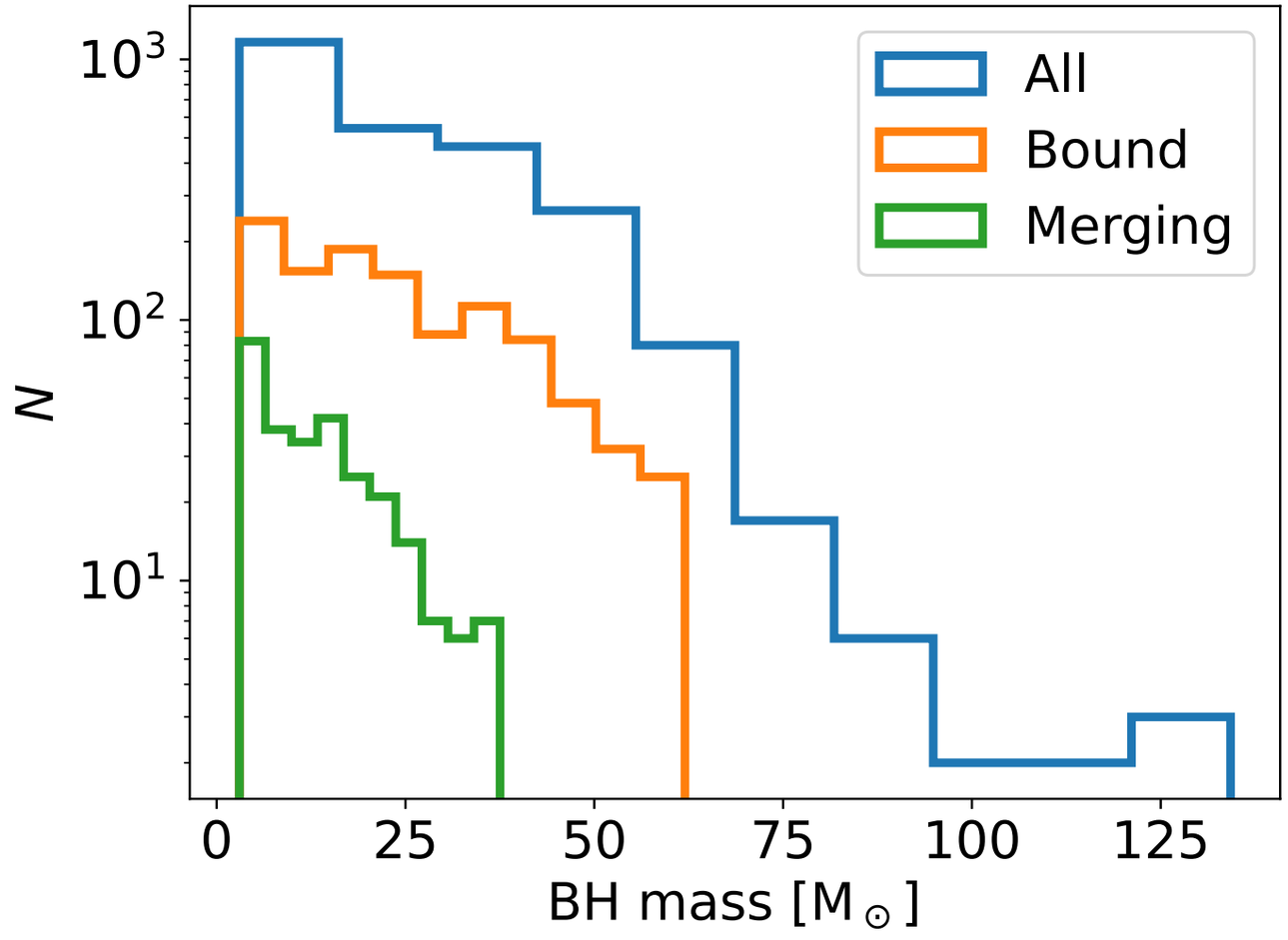


Analysis 1

We want to plot the mass distribution of the Black holes at the end of the simulations, considering all black holes, black holes in binaries and black holes in binaries that merge in a Hubble time

Remember:

- The binary evolution is stopped when both components are remnants (Black hole, Neutron star, white dwarf, empty).
- The same BH can appear more the once in the output files (the properties of the systems are printed in output each time an triggering event occurs)

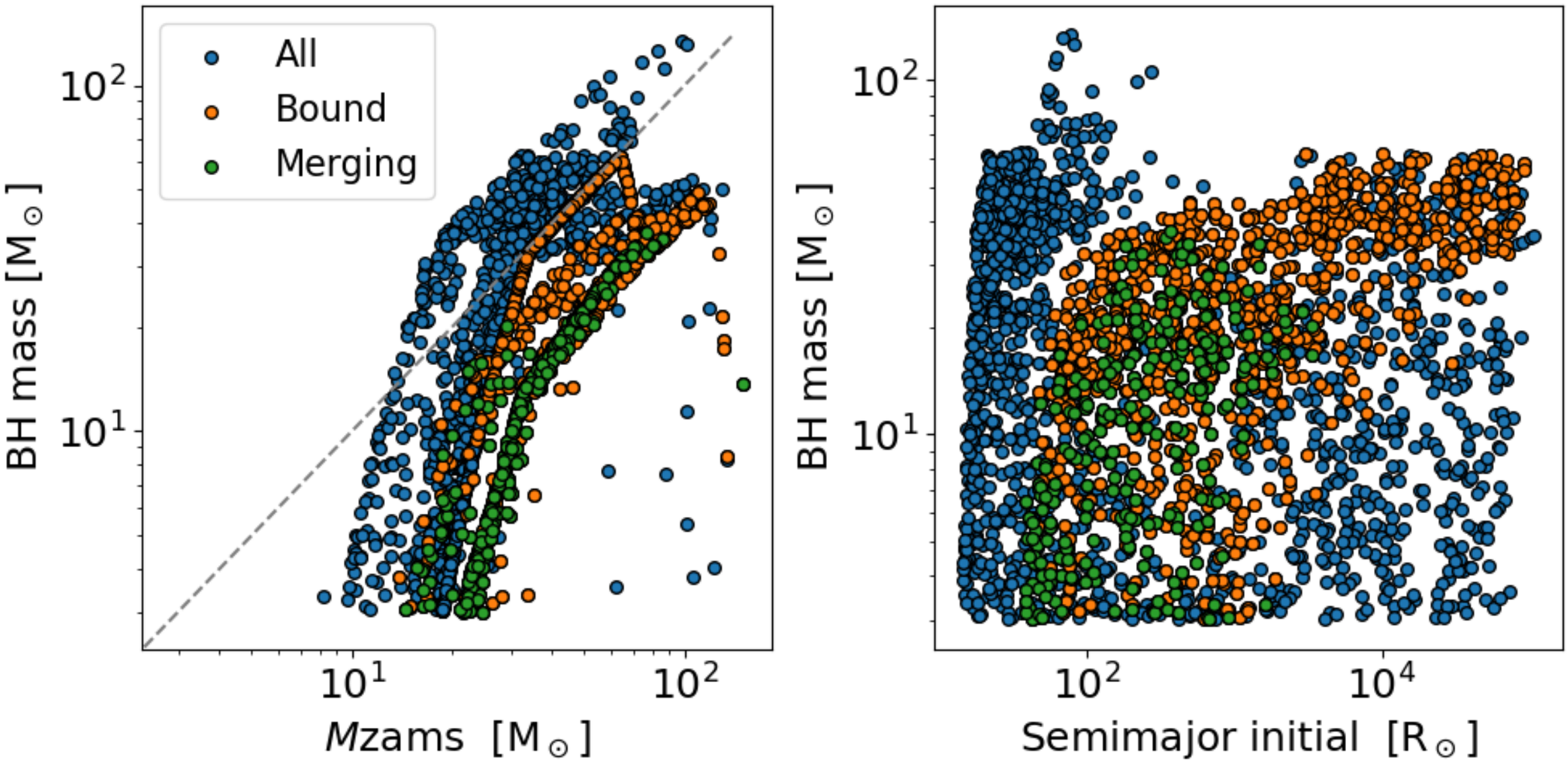


Analysis 2

We want to make a scatter plot of black hole mass vs initial stellar mass, and black hole mass vs initial orbital separation.

Remember:

- Information on the initial masses and orbital parameters are in the evolved file



Hints and tips

Join evolved and output files

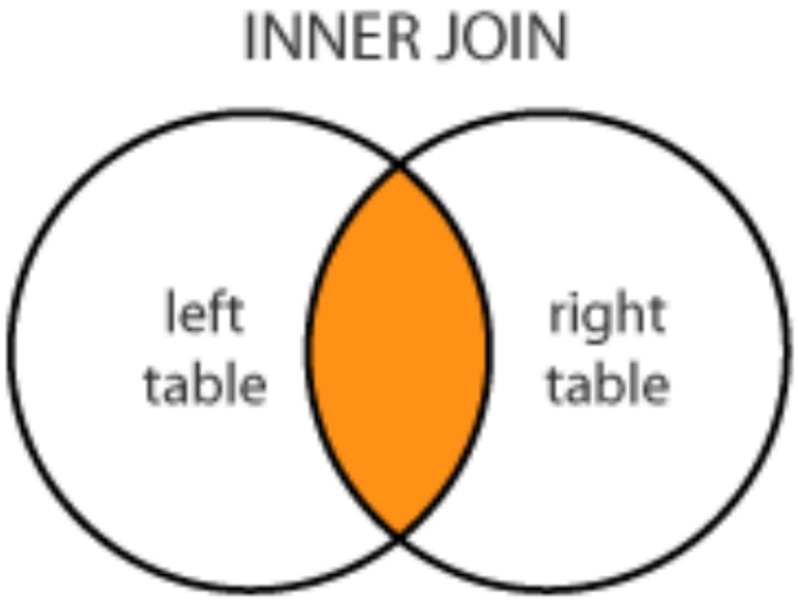
- Sometimes a combination of data in the output and in the evolved files is needed. For example, we want to add information of the initial Mzams and initial orbital properties for all BHBH binaries. The PANDAS method *merge* handles all the possible joins among two data frame.
- Assume we want to merge two dataframes (df_left, df_right) joining together the information of data sharing the same value on a given column (e.g. an ID), this is an inner join.

```
df_merged = df_left.merge( df_right, on="ID",  how="inner", suffixes=("_left","_right") )
```

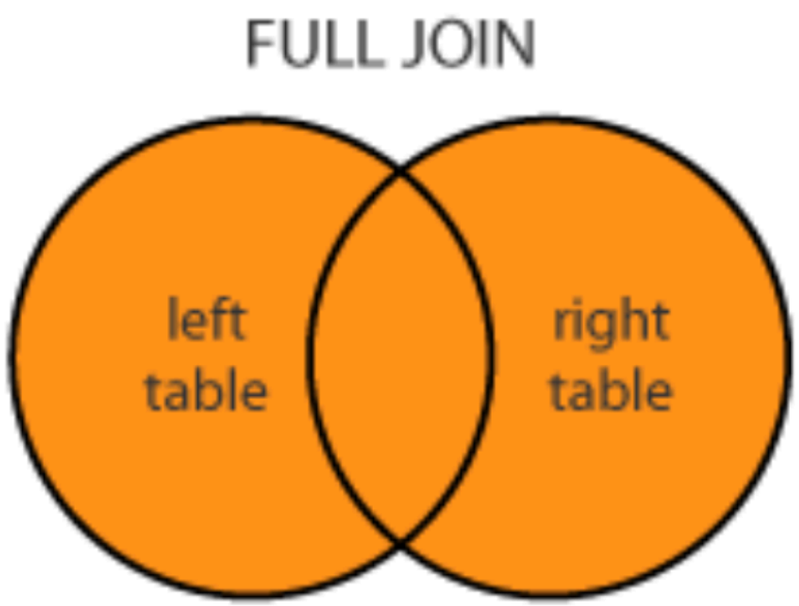
- **on**: column(s, can be a list of columns) to match during the merge of the two tables. The colum(s) has(have) to be present in both the tables
- **how:** type of join to use, see documentation here and the next slide
- **suffixes:** columns with the same name in the two tables (not used in on) will be renamed adding these suffixes.

Hints and tips

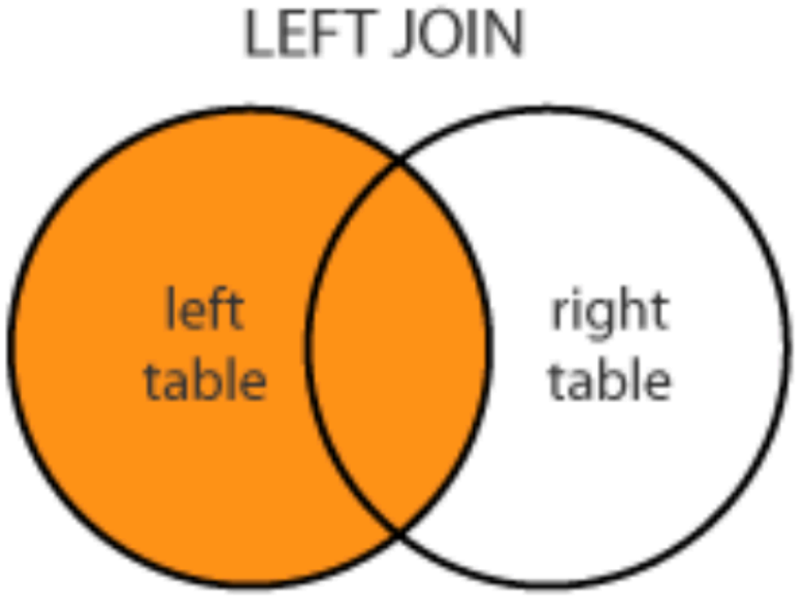
Join evolved and output files - join types



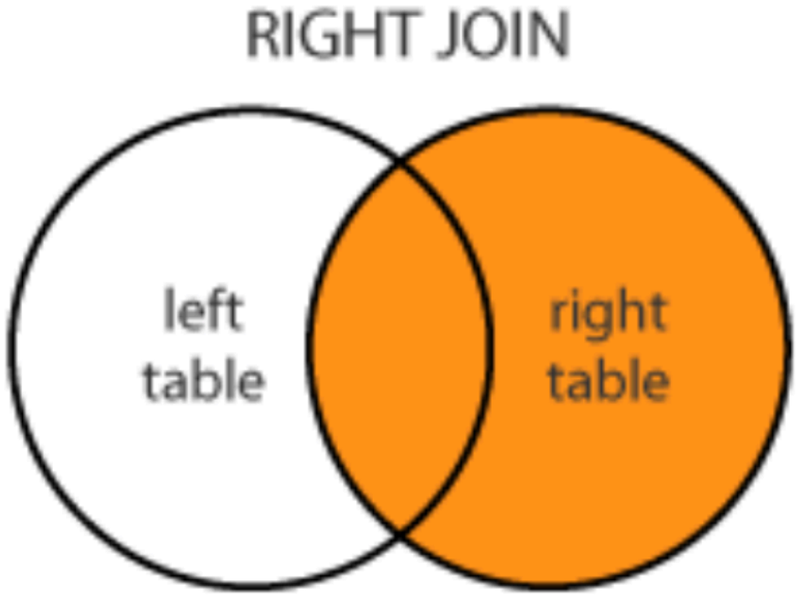
how: “inner”



how: “outer”



how: “left”



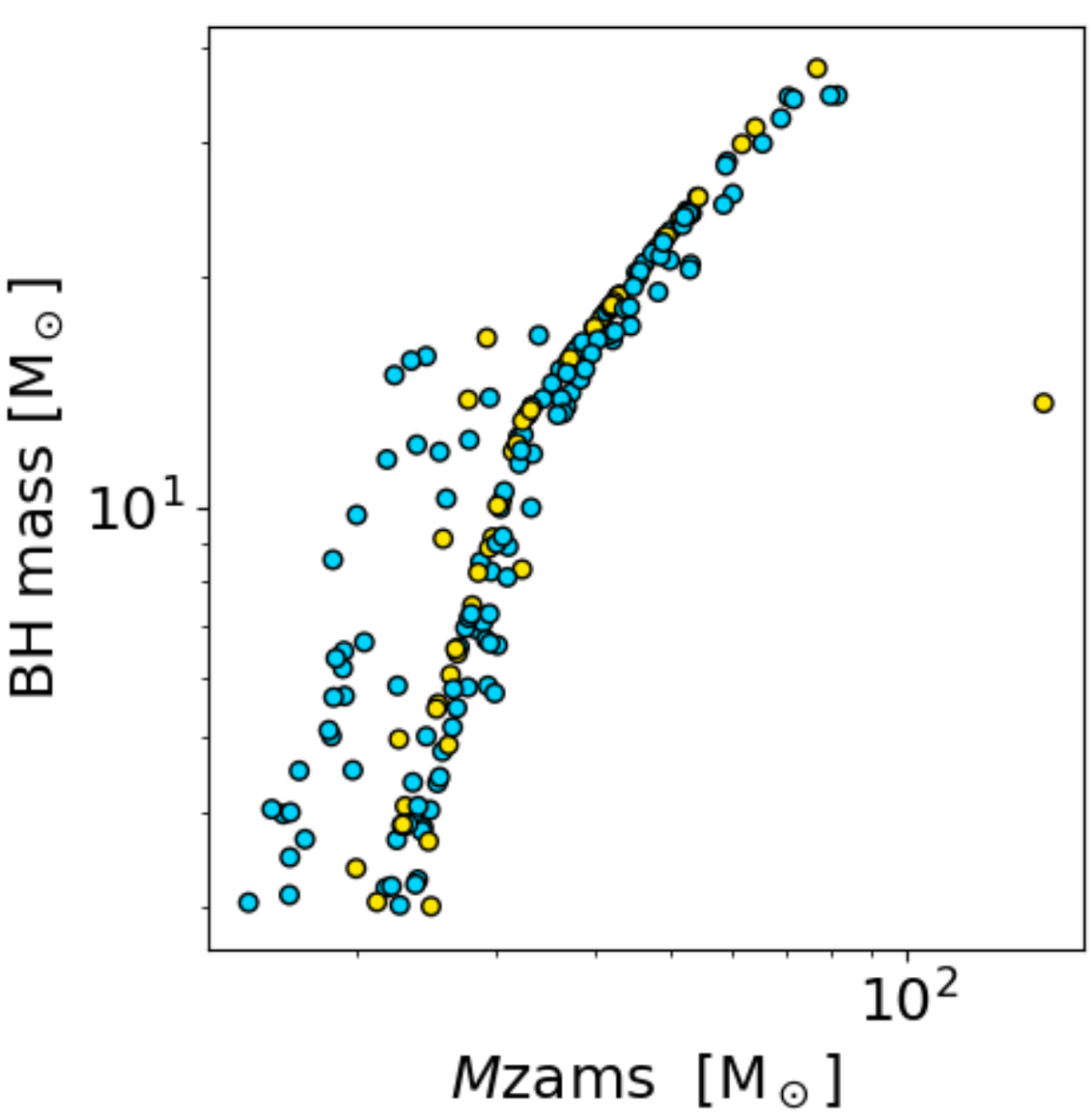
how: “right”

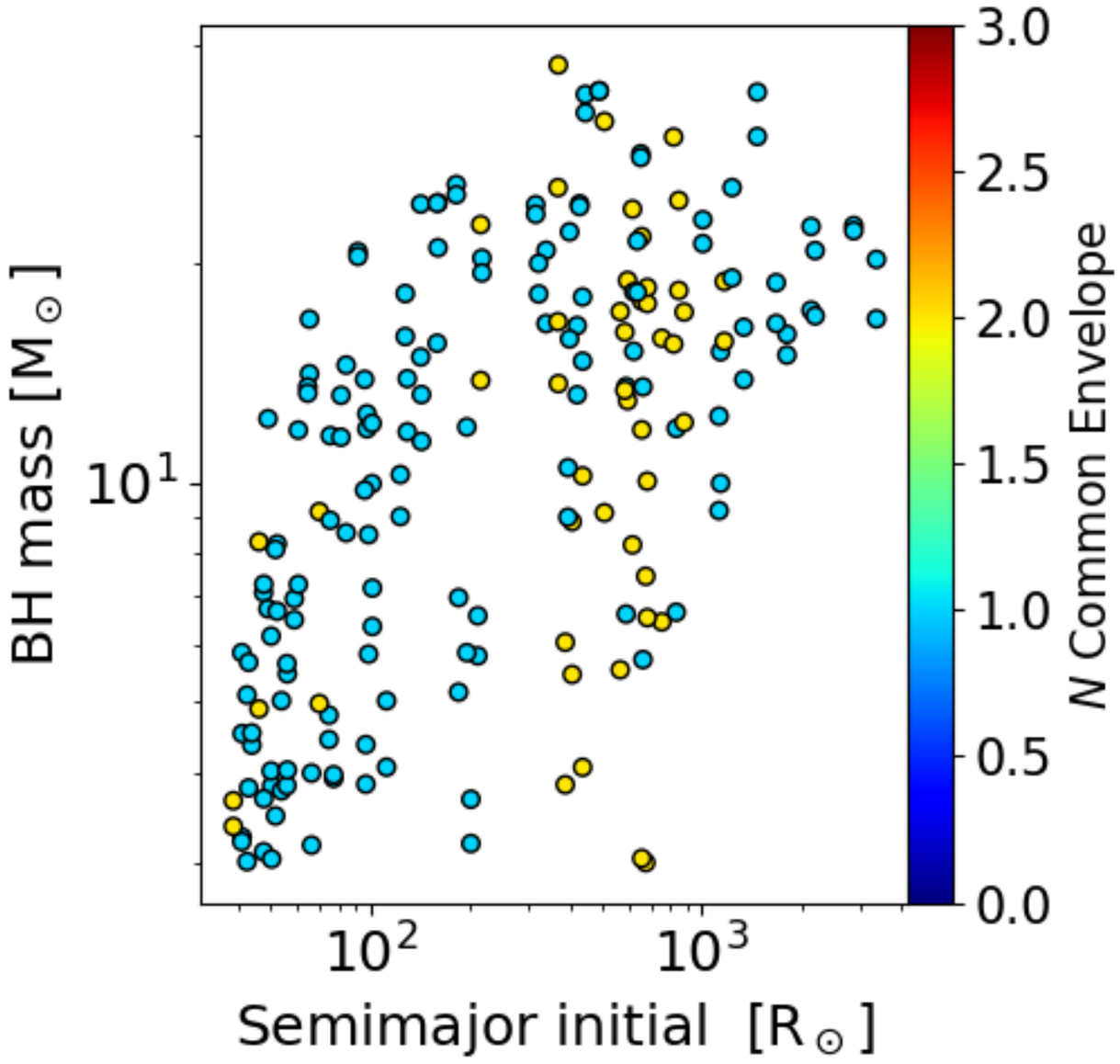
Analysis 3

Replicate the previous plot, considering only merging binary BHs. The color of each point indicates the number of Common Envelope events triggered during the evolution

Remember:

- Complementary information on the binary evolution can be found in the logfiles





Log files: *logfile_xxx.dat*

Content - how to read it: Header

HEADER The first part of each row is the header:

Object;name;ID;event;time;

- ***Object:*** **S** indicates a single star, **B** indicates a binary
- ***name:*** object name (long int), notice the stars have the same name of the binary they belong to. It can be used to link the systems to the output and evolved files
- ***ID:*** id of the object (long int). If the object is a star the id is 0 if it is the first star or 1 if it is the second one. It can be used to link the systems to the output and evolved files.
- ***event:*** a label in capital letter indicating the triggering event. All the possible events are listed in the next slide
- ***time:*** time of the simulation (Myr) when the event is triggered.

B;621064598874129;13;CE;3.348641;

A Common envelope (*event*=CE) is triggered in the binary with *name* 621064598874129 and *ID* 13 at *time* 3.348641 Myr.

Hints and tips

Reg[ular] *
Ex[pression]

Reading and handling logfiles: RegEx

Example: RegEx and re to count the number of CE for each binary of compact objects and add the information to the final output

1. **First, create the RexEx string**, we are interested only on counting CE events, therefore we need just to match the label *CE* in the header of each row and get the ID of the binary

```
regex_str=r'B;\d+;(\d+);CE;'
```

2. **Match and get the ID**,

```
logfile_path=“sevn_output/logfile_0.dat”
with  open(logfile_path,"r") as fo:
    ma=re.findall(regex_str,fo.read())
#ma is a list of all the matched output (they are all strings)
```

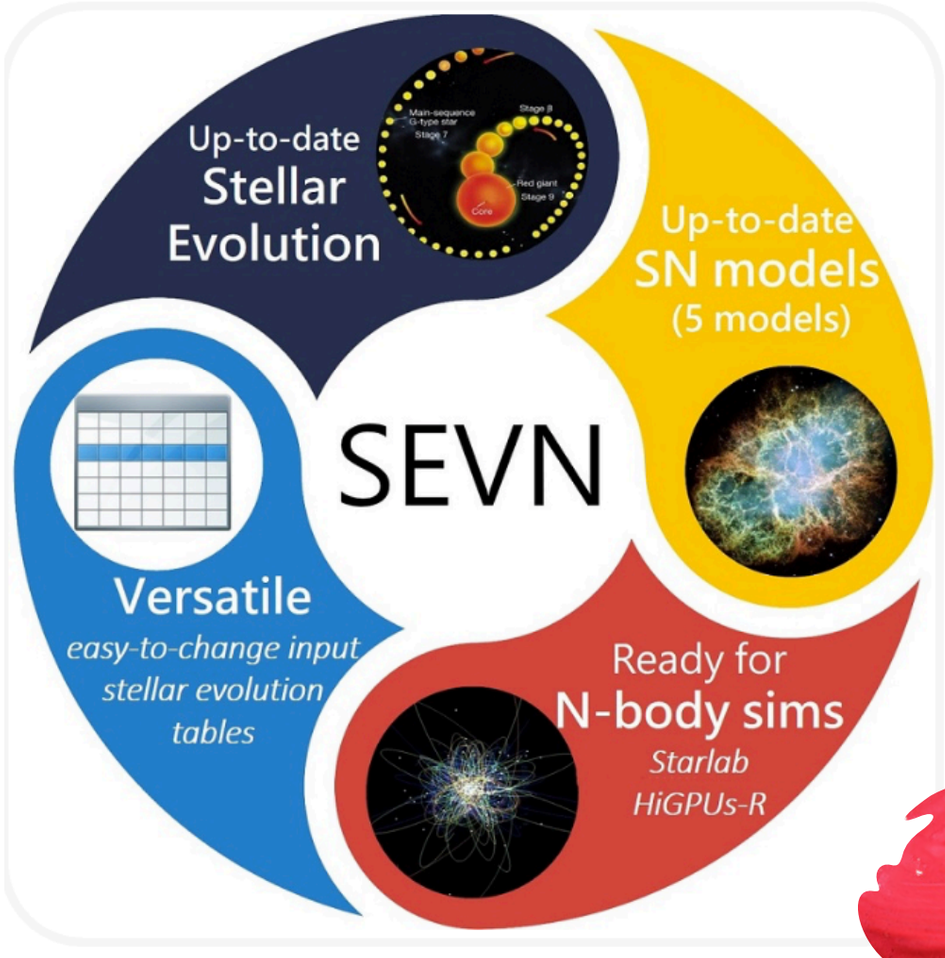
3. **Create a Dataframe and group by ID to count the number of CE events**

```
#Create a dataframe with all the ID obtained in the re findall
BID = pd.DataFrame({'ID':np.asarray(ma,dtype=int)})


#Group by ID and count occurences producing a dataframe with the
#column ID and the column NCE containing the number of CEs
BID= BID.groupby(['ID']).size().reset_index(name='NCE')
```

Questions ?

I will be around if you want more detailed information about SEVN and projects/thesis opportunity



We want you!