# Coordinates and Time

astropy

# astropy.time

Python's built-in `datetime` package handles standard dates, times, but doesn't support astronomical formats (e.g., JD, MJD) or precise timing (e.g. a nanosecond over a Hubble Time)

The `astropy.time` subpackage adds this support

**Key object:** `Time`

```
from astropy.time import Time
```

## astropy.time

```
>>> time = Time(58086.182, format='mjd')
>>> print(time.jd)
2458086.682
>>> print(time.datetime)
datetime.datetime(2017, 11, 29, 4, 22, 4, 800000)
>>> Time.now()
<Time object: scale='utc' format='datetime'
value=2017-11-28 12:42:27.939562>
```

(More examples in the coordinates tutorial…)

# astropy.coordinates

For representing and transforming astronomical coordinates and velocities

**Key object:** `SkyCoord`

```
import astropy.coordinates as coord
coord.SkyCoord(ra=210.6*u.deg,
               dec=-19.3*u.deg)
```

# astropy.coordinates

1st layer: Representations: spatial vectors

Spherical (angles and distance), Cartesian, …

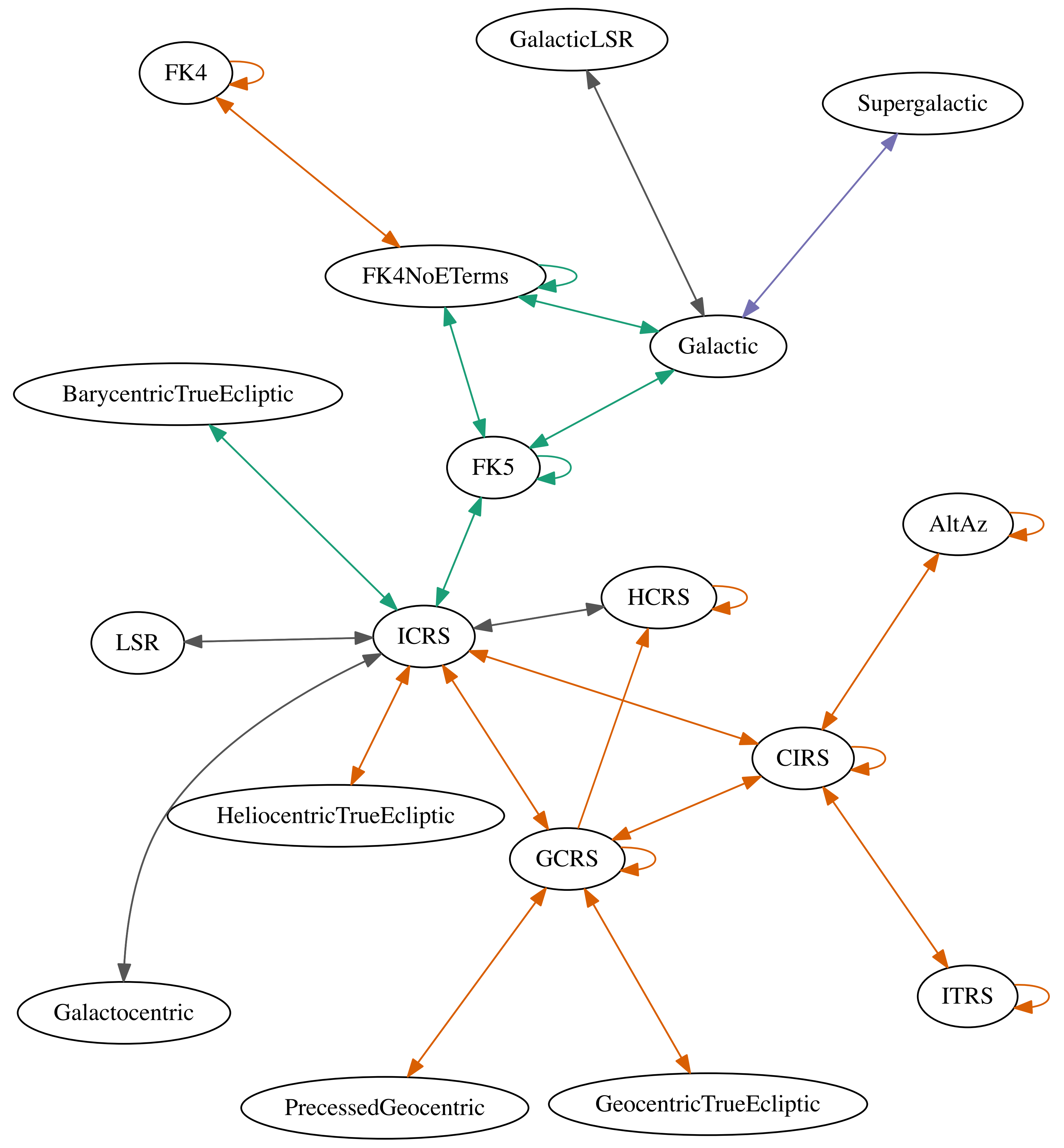
```
import astropy.coordinates as coord
coord.CartesianRepresentation(
x=1*u.kpc, y=2*u.kpc, z=3*u.kpc)
```

# astropy.coordinates

2nd layer: Frame objects - conceptually, frames-of-reference, *may or may not* have data.

ICRS (J2000 equatorial), Galactic, Alt-Az, …

**Key objects:** `frame objects: ICRS, Galactic, etc.`



# astropy.coordinates

3rd layer: Convenient interface with all the bells and whistles - see the tutorial

**Key object:** `SkyCoord`

```
import astropy.coordinates as coord
coord.SkyCoord(ra=210.6*u.deg,
               dec=-19.3*u.deg)
```

## array-like times/coordinates

Beware of a conceptual hurdle: `astropy.time` and `astropy.coordinates` both have the concept that the same object stores scalars *or* arrays.

```
>>> scalar_time = Time(2458086.682, format='jd')
>>> scalar_time.shape
()

>>> arr_time = Time([2458086.682, 2458087.682],
    format='jd')
>>> arr_time.shape
(2,)
```

## Tutorial

Open up `astropy_coordinates.ipynb` and dive in!