This project AIDA receives funding from the European Union’s Horizon 2020 Research and Innovation programme under grant agreement No 776262.

First AIDA school

Space missions data acquisition

Hugo Breuillard
LPP/CNRS

Bologna

21 January 2020

Outline

WARNING:
Some features of the Event Search tool have been updated in this document, thus this section is outdated in the recorded video

Outline

1. **Introduction**

Introduction

- AIDA project overall goal:

  make available to the heliospheric community a new approach for the exploitation of data from space missions: AI used to extract information on ongoing events, processes and features.

  Events identified in a completely automated way and without the need for human intervention

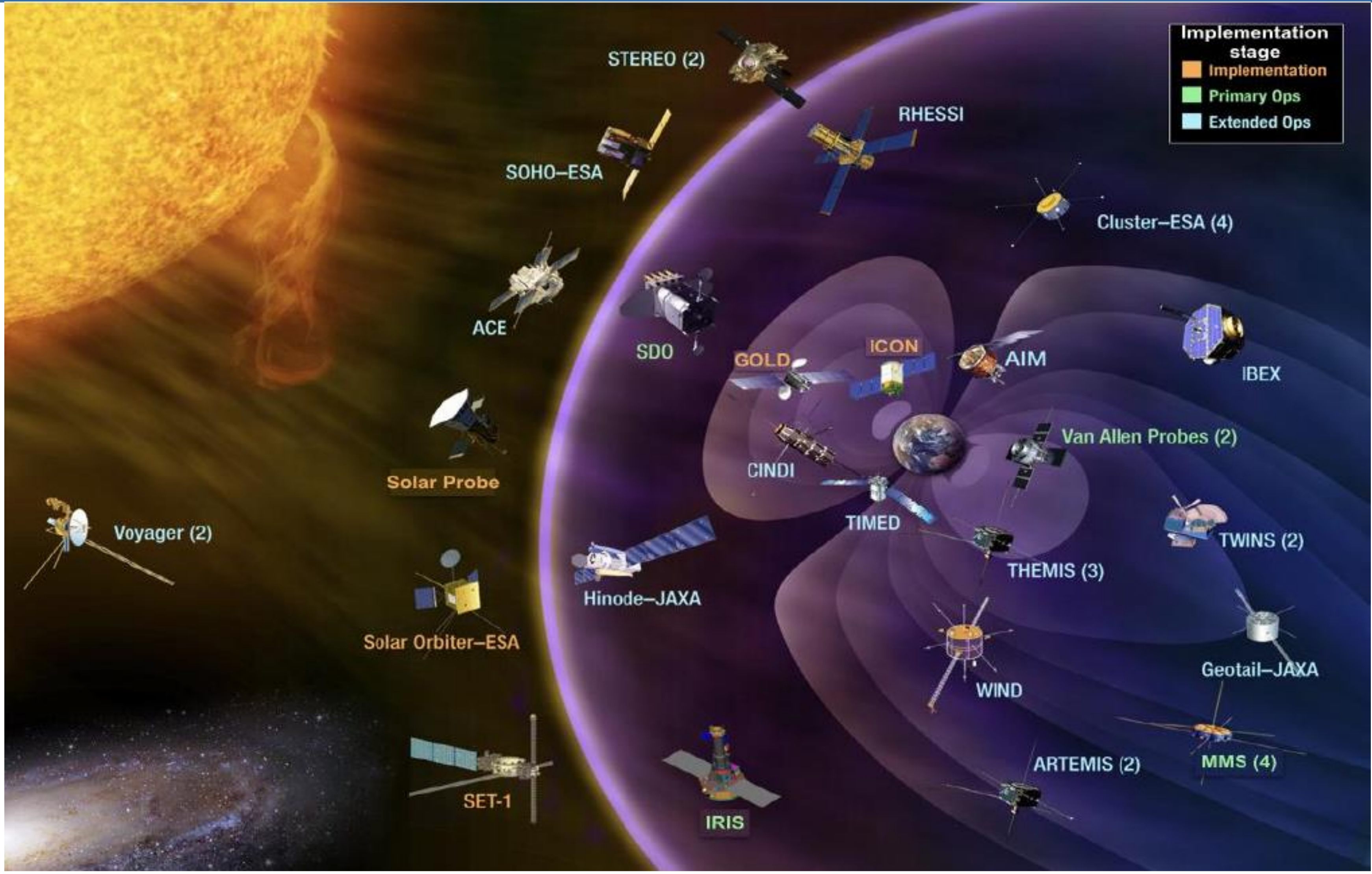
- Two of AIDA’s objectives:
  1. Develop a tool to interface the AIDApy package with existing heliospheric missions: select and retrieve heterogeneous data from (mostly mission-specific) open-access databases (NASA OMNIWeb, Cluster Science Archive, MMS SDC) and deliver standardized data for AI
  2. Develop tools to automatically look for scientific processes of interest (magnetic reconnection, shocks, etc) within the space missions data

Outline

1. Introduction
2. **Space missions**

Space missions

Current and upcoming missions monitoring the Sun-Earth region:

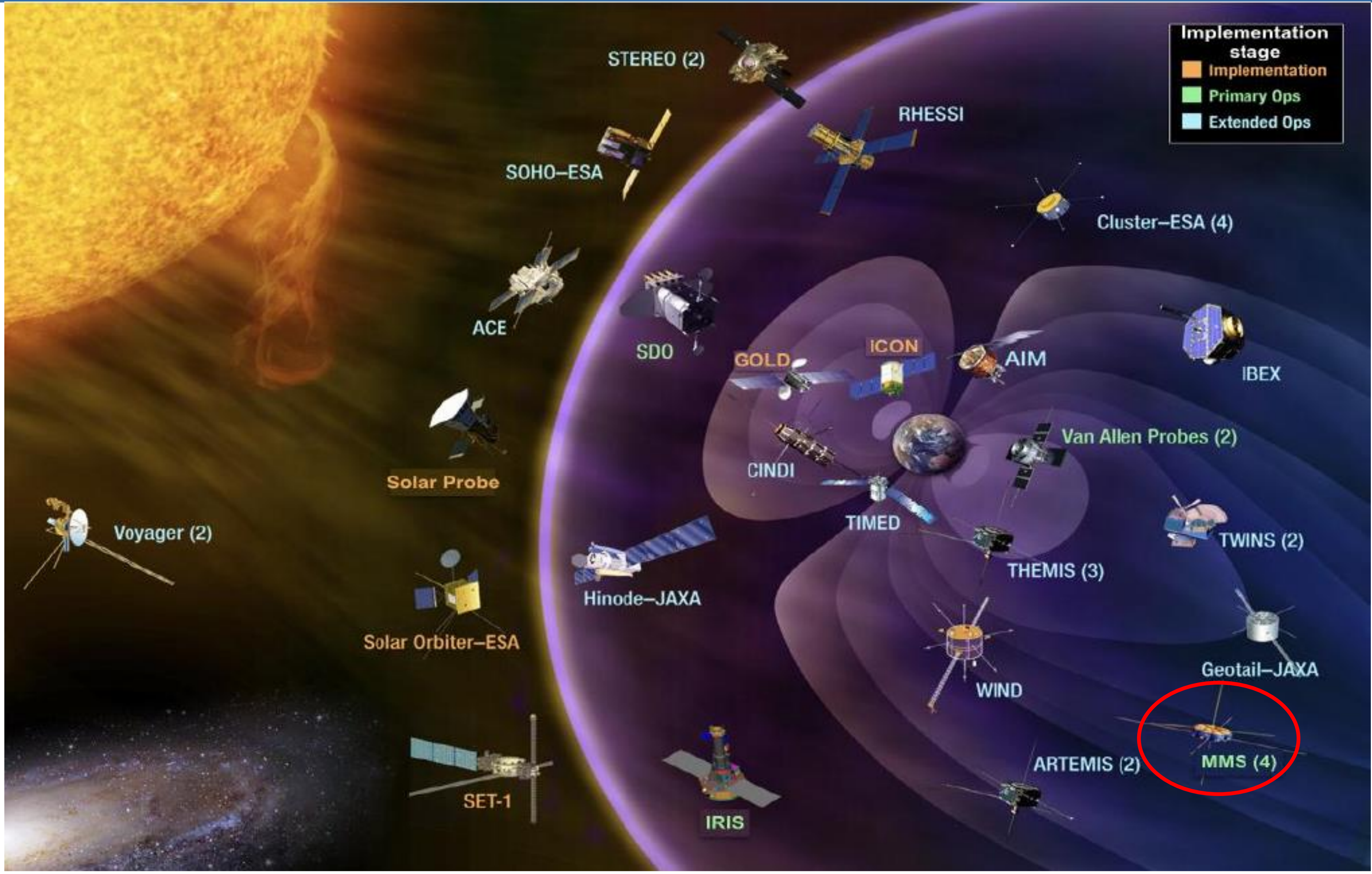


Space missions

Current and upcoming missions monitoring the Sun-Earth region:

As a first step, we focus on three different missions:

- **MMS**: Multi-spacecraft studies of the microphysics of magnetic reconnection, turbulence…

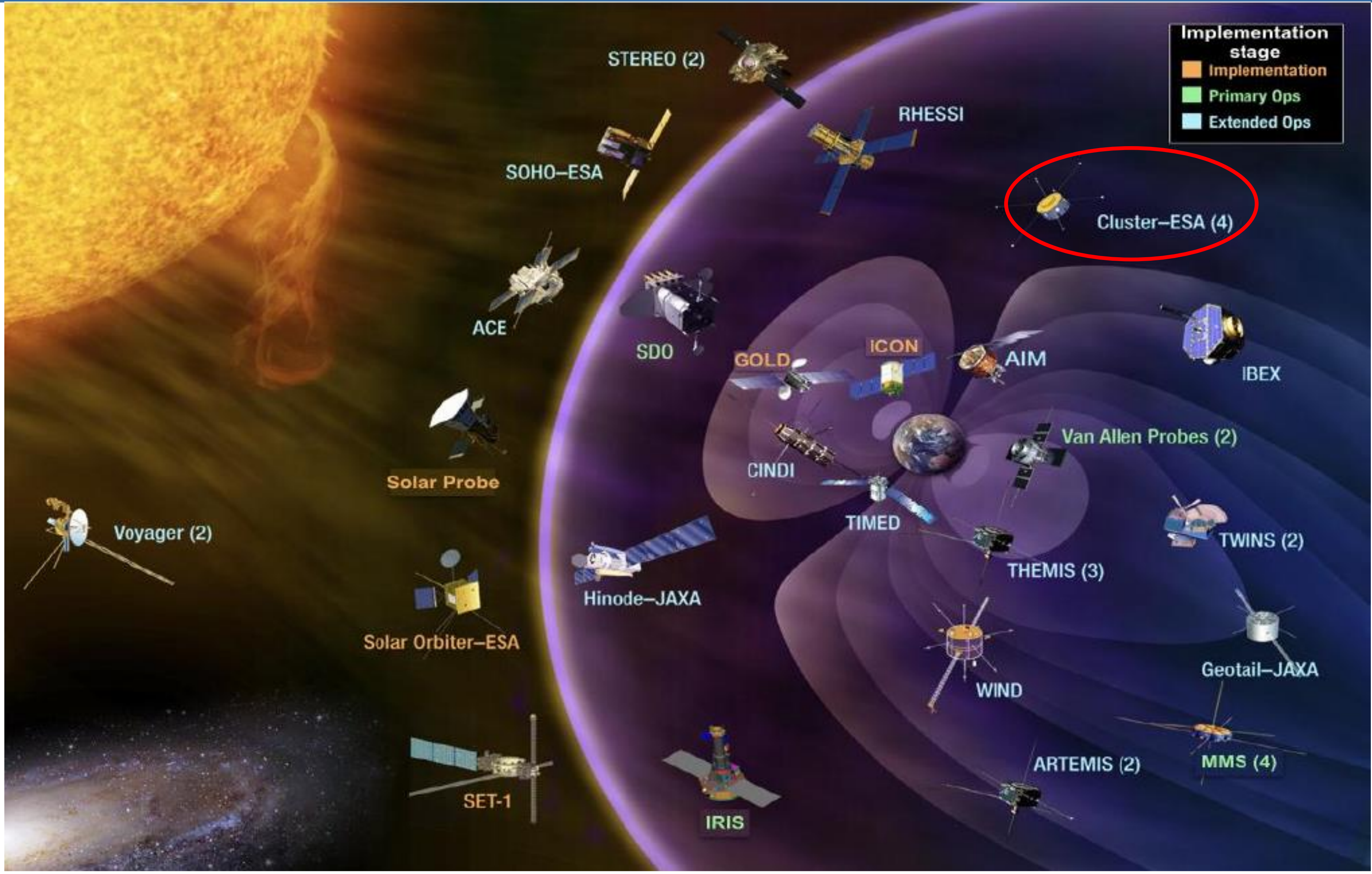


Space missions

Current and upcoming missions monitoring the Sun-Earth region:

As a first step, we focus on three different missions:

- **MMS**: Multi-spacecraft studies of the microphysics of magnetic reconnection, turbulence…
- **Cluster**: Multi-spacecraft studies of magnetospheric regions (e.g., polar cusp, magnetopause, magnetotail)

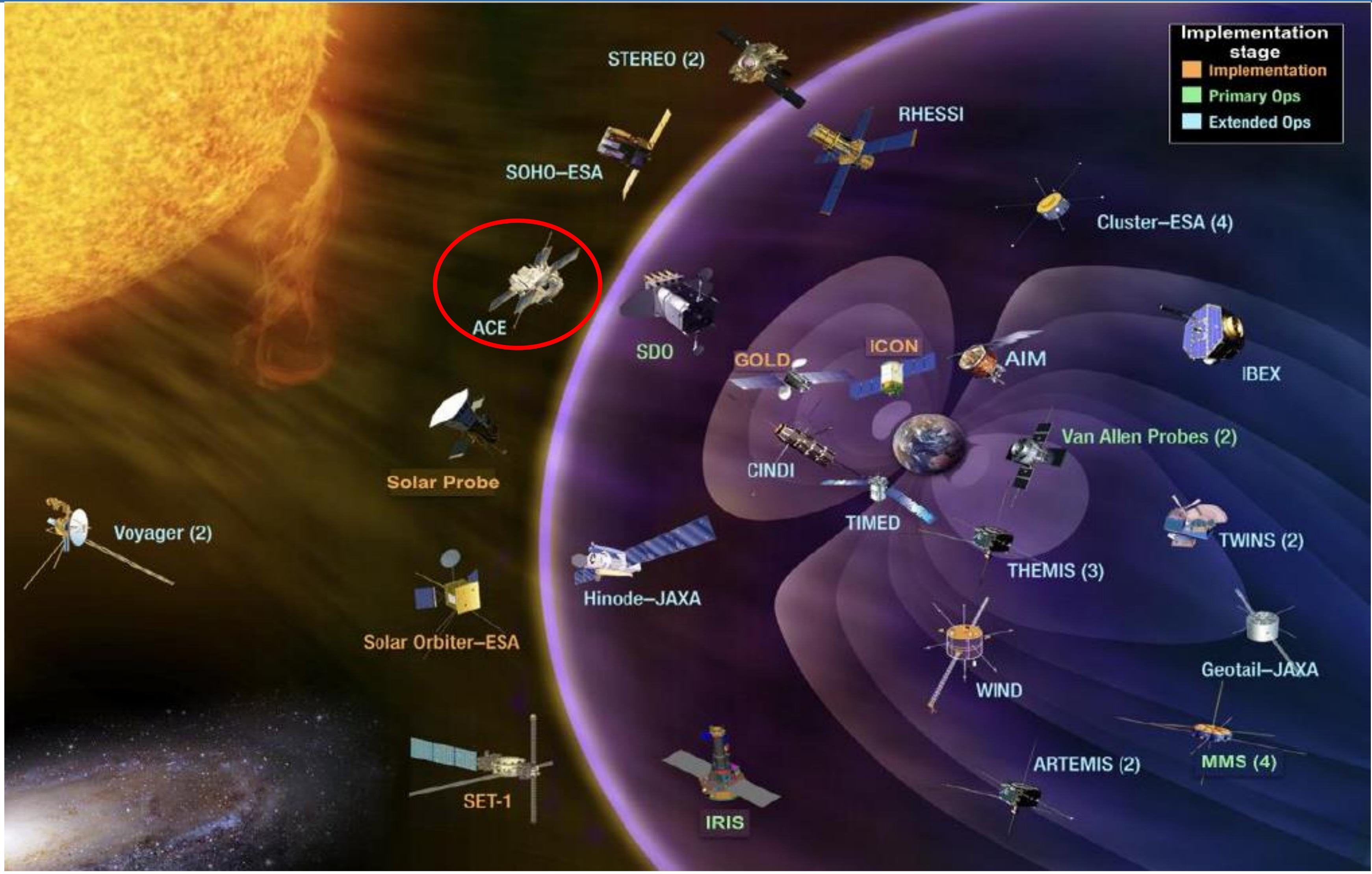


Space missions

Current and upcoming missions monitoring the Sun-Earth region:

As a first step, we focus on three different missions:

- **MMS**: Multi-spacecraft studies of the microphysics of magnetic reconnection, turbulence…
- **Cluster**: Multi-spacecraft studies of magnetospheric regions (e.g., polar cusp, magnetopause, magnetotail)
- **OMNI**: data set of near-Earth solar wind data from several spacecraft (such as ACE) in geocentric or L1 orbits



Outline

1. Introduction
2. Space missions
3. **The Mission tool**

Mission tool : main objectives

- **AIDApy subpackage to automatically select and download data from a variety of open-access in situ, remote and ground plasma data archives**
- **Inputs**
  - Handling of various missions (Cluster, MMS, OMNI at the moment)
  - Precise control of the desired data: the query can ask specific time range, probes, coordinates, etc.
  - Creation of a catalog of all instruments, probes, coordinates, data modes and products (e.g., magnetic field and electron density measurements, ion distribution functions, etc) available allowed for each supported mission
- **Outputs:**
  - Handling of time-varying complex datasets by using the Python package “xarray”: multi-dimensional labeled data arrays (xarray.DataArray object) and datasets (xarray.Dataset object: dictionary of DataArray objects)
  - Proper and easy-to-handle data container providing science data but also time ranges, metadata, etc
  - Uniform interface for other AIDApy’s packages to perform advanced analysis on the spacecraft data.

Mission tool: inputs

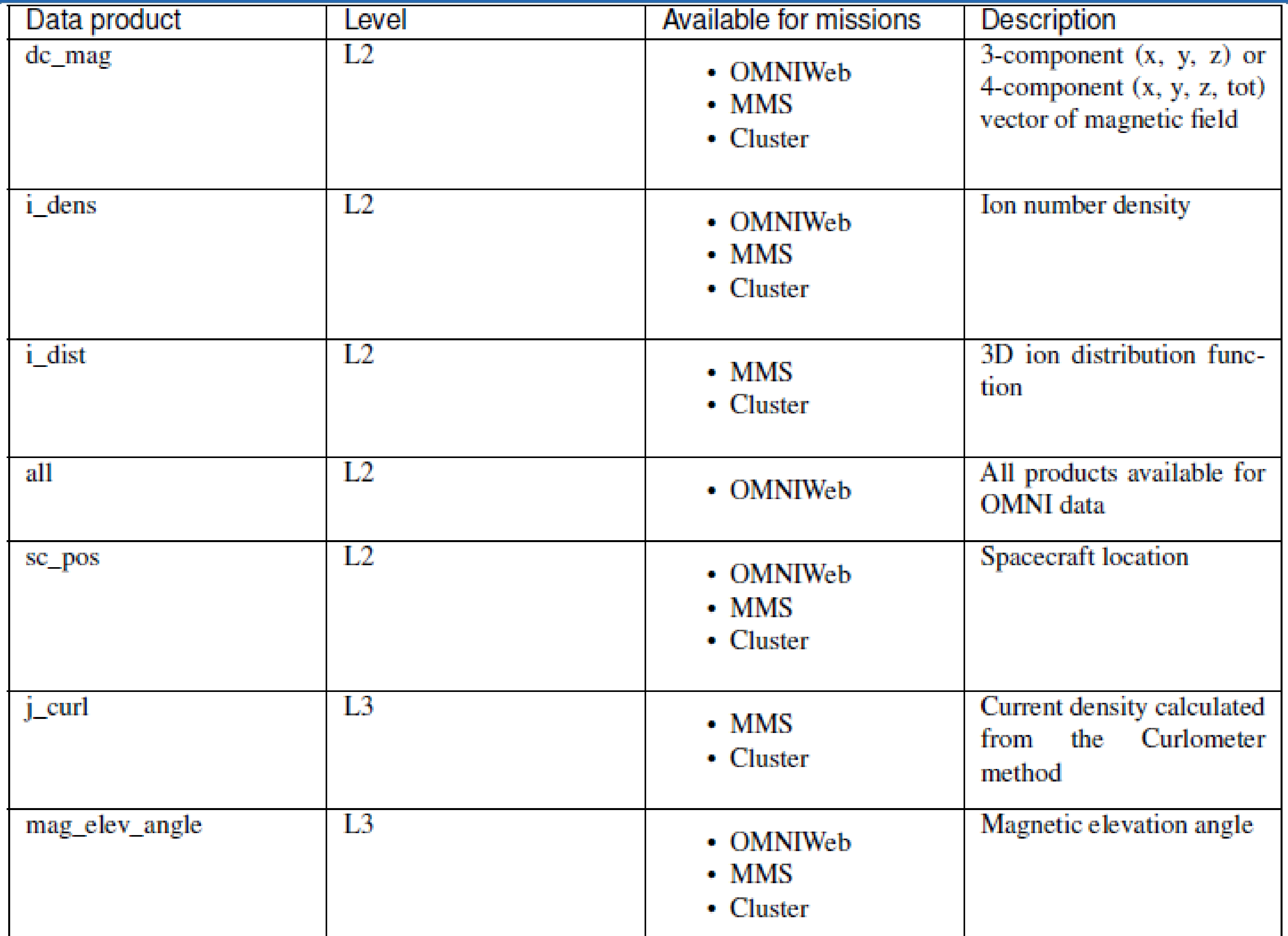
Below is a code snippet showing how to define the inputs for the Mission subpackage using a Python dictionary:

```
# Define the time interval
start_time = datetime(<year>, <month>, <day>, <hour>, <minute>, <second>)
end_time = datetime(<year>, <month>, <day>, <hour>, <minute>, <second>)

# Define the settings as a Python dictionary
settings = {'prod': ['<prod1>', '<prod2>'], 'probes': ['<probe1>', '<probe2>'],
'coords': '<coord_system>'}
```

Once the parameters are set up, we generate the Mission downloader and download the data. This is done by calling the *load_data* function, that tells the Mission subpackage to create a specific downloader for every mission and time interval requested, and download and load the desired data products for the specified probes and coordinate system:

```
data = load_data('mission'='<mission>', start_time, end_time, **settings)
```

Products catalog of the Mission tool



Data product	Level	Available for missions	Description
dc_mag	L2	• OMNIWeb • MMS • Cluster	3-component (x, y, z) or 4-component (x, y, z, tot) vector of magnetic field
i_dens	L2	• OMNIWeb • MMS • Cluster	Ion number density
i_dist	L2	• MMS • Cluster	3D ion distribution function
all	L2	• OMNIWeb	All products available for OMNI data
sc_pos	L2	• OMNIWeb • MMS • Cluster	Spacecraft location
j_curl	L3	• MMS • Cluster	Current density calculated from the Curlometer method
mag_elev_angle	L3	• OMNIWeb • MMS • Cluster	Magnetic elevation angle

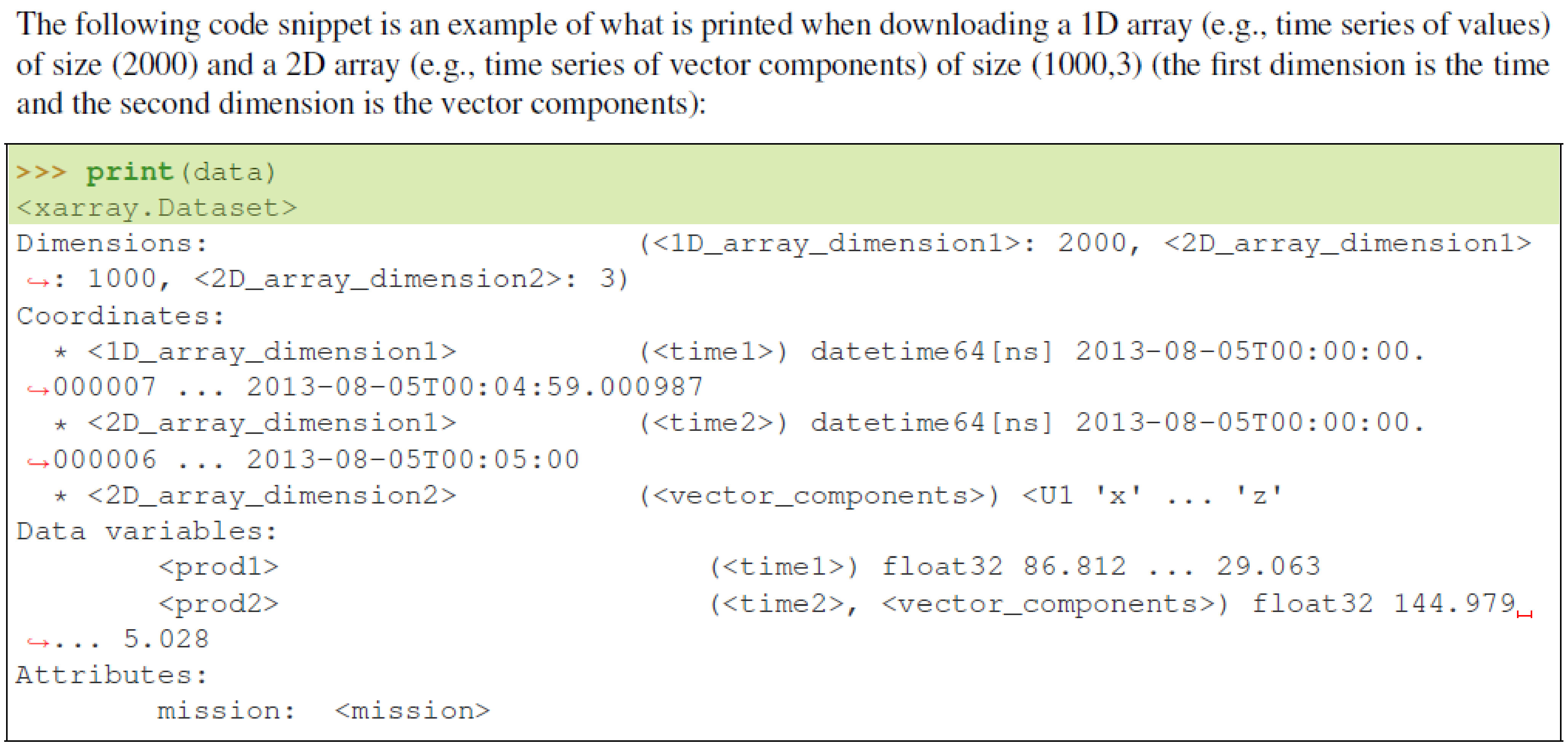
Mission tool: outputs

The following code snippet is an example of what is printed when downloading a 1D array (e.g., time series of values) of size (2000) and a 2D array (e.g., time series of vector components) of size (1000,3) (the first dimension is the time and the second dimension is the vector components):

```
>>> print(data)
<xarray.Dataset>
Dimensions:                          (<1D_array_dimension1>: 2000, <2D_array_dimension1>
↪: 1000, <2D_array_dimension2>: 3)
Coordinates:
  * <1D_array_dimension1>            (<time1>) datetime64[ns] 2013-08-05T00:00:00.
↪000007 ... 2013-08-05T00:04:59.000987
  * <2D_array_dimension1>            (<time2>) datetime64[ns] 2013-08-05T00:00:00.
↪000006 ... 2013-08-05T00:05:00
  * <2D_array_dimension2>            (<vector_components>) <U1 'x' ... 'z'
Data variables:
        <prod1>                          (<time1>) float32 86.812 ... 29.063
        <prod2>                          (<time2>, <vector_components>) float32 144.979␣
↪... 5.028
Attributes:
        mission:  <mission>
```

Mission tool: outputs

Multidimensional data arrays dictionary

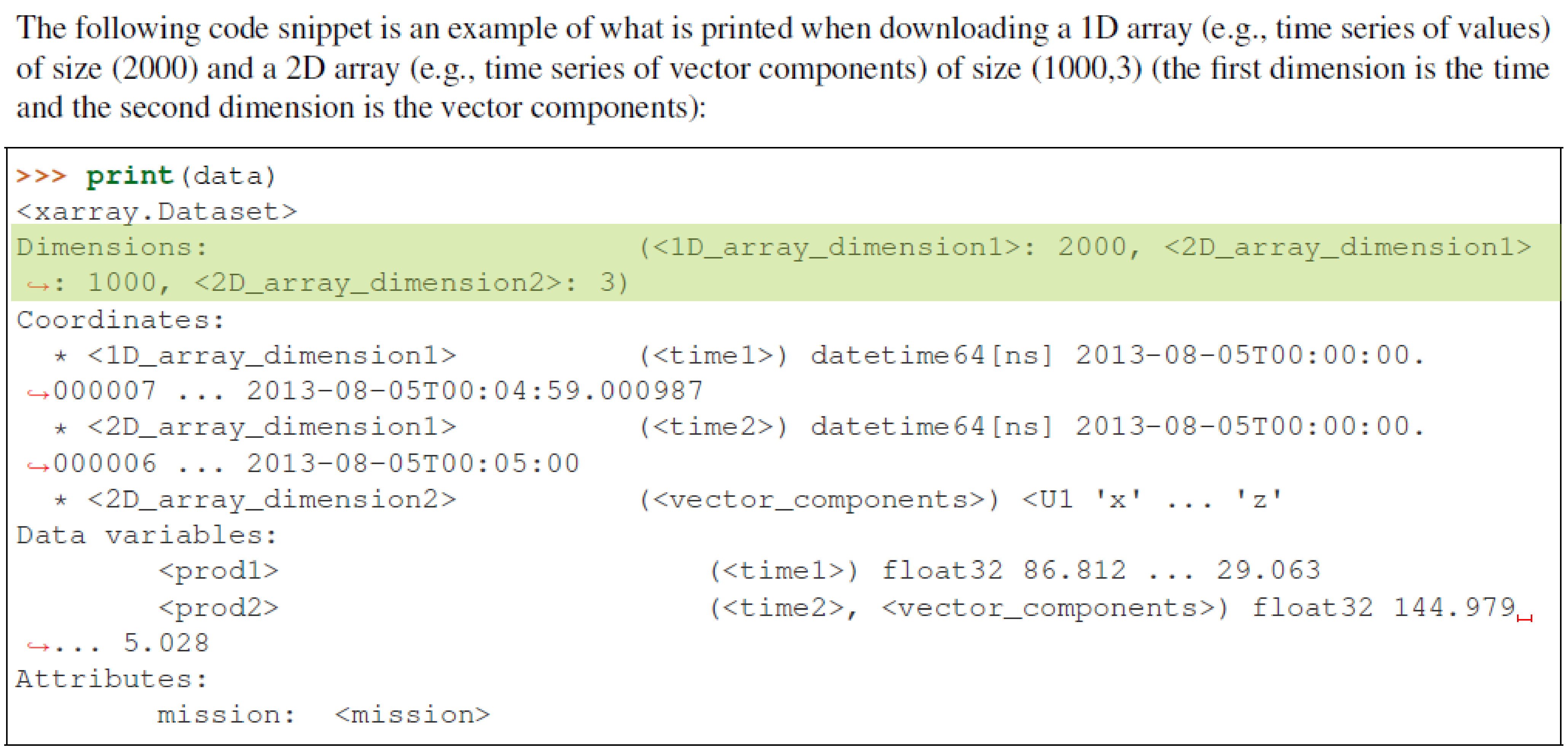


The following code snippet is an example of what is printed when downloading a 1D array (e.g., time series of values) of size (2000) and a 2D array (e.g., time series of vector components) of size (1000,3) (the first dimension is the time and the second dimension is the vector components):

```
>>> print(data)
<xarray.Dataset>
Dimensions:                            (<1D_array_dimension1>: 2000, <2D_array_dimension1>
↪: 1000, <2D_array_dimension2>: 3)
Coordinates:
  * <1D_array_dimension1>              (<time1>) datetime64[ns] 2013-08-05T00:00:00.
↪000007 ... 2013-08-05T00:04:59.000987
  * <2D_array_dimension1>              (<time2>) datetime64[ns] 2013-08-05T00:00:00.
↪000006 ... 2013-08-05T00:05:00
  * <2D_array_dimension2>              (<vector_components>) <U1 'x' ... 'z'
Data variables:
        <prod1>                            (<time1>) float32 86.812 ... 29.063
        <prod2>                            (<time2>, <vector_components>) float32 144.979␣
↪... 5.028
Attributes:
        mission:  <mission>
```

Mission tool: outputs

Data arrays dimensions

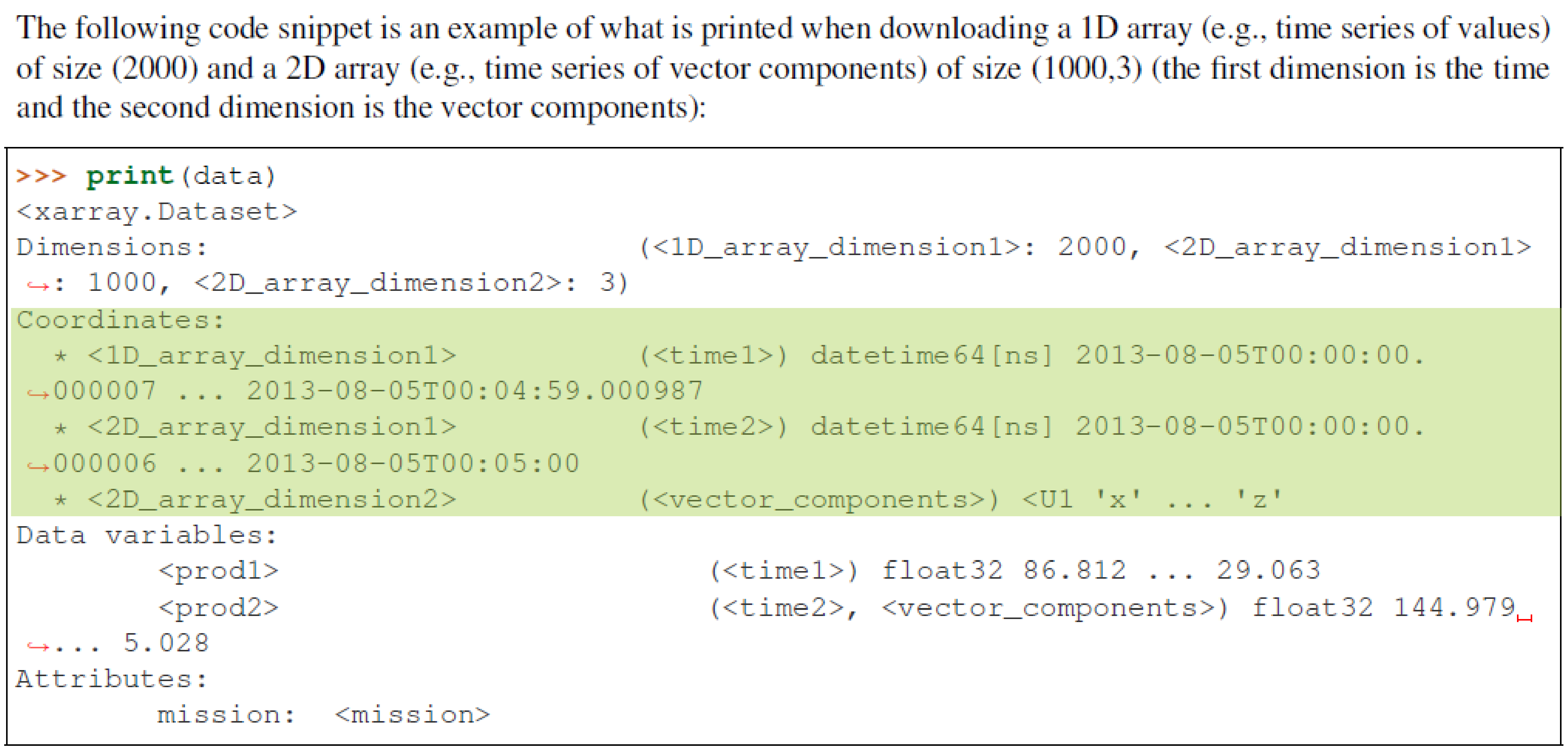


The following code snippet is an example of what is printed when downloading a 1D array (e.g., time series of values) of size (2000) and a 2D array (e.g., time series of vector components) of size (1000,3) (the first dimension is the time and the second dimension is the vector components):

```
>>> print(data)
<xarray.Dataset>
Dimensions:                    (<1D_array_dimension1>: 2000, <2D_array_dimension1>
↪: 1000, <2D_array_dimension2>: 3)
Coordinates:
  * <1D_array_dimension1>      (<time1>) datetime64[ns] 2013-08-05T00:00:00.
↪000007 ... 2013-08-05T00:04:59.000987
  * <2D_array_dimension1>      (<time2>) datetime64[ns] 2013-08-05T00:00:00.
↪000006 ... 2013-08-05T00:05:00
  * <2D_array_dimension2>      (<vector_components>) <U1 'x' ... 'z'
Data variables:
        <prod1>                    (<time1>) float32 86.812 ... 29.063
        <prod2>                    (<time2>, <vector_components>) float32 144.979␣
↪... 5.028
Attributes:
        mission:  <mission>
```

Mission tool: outputs

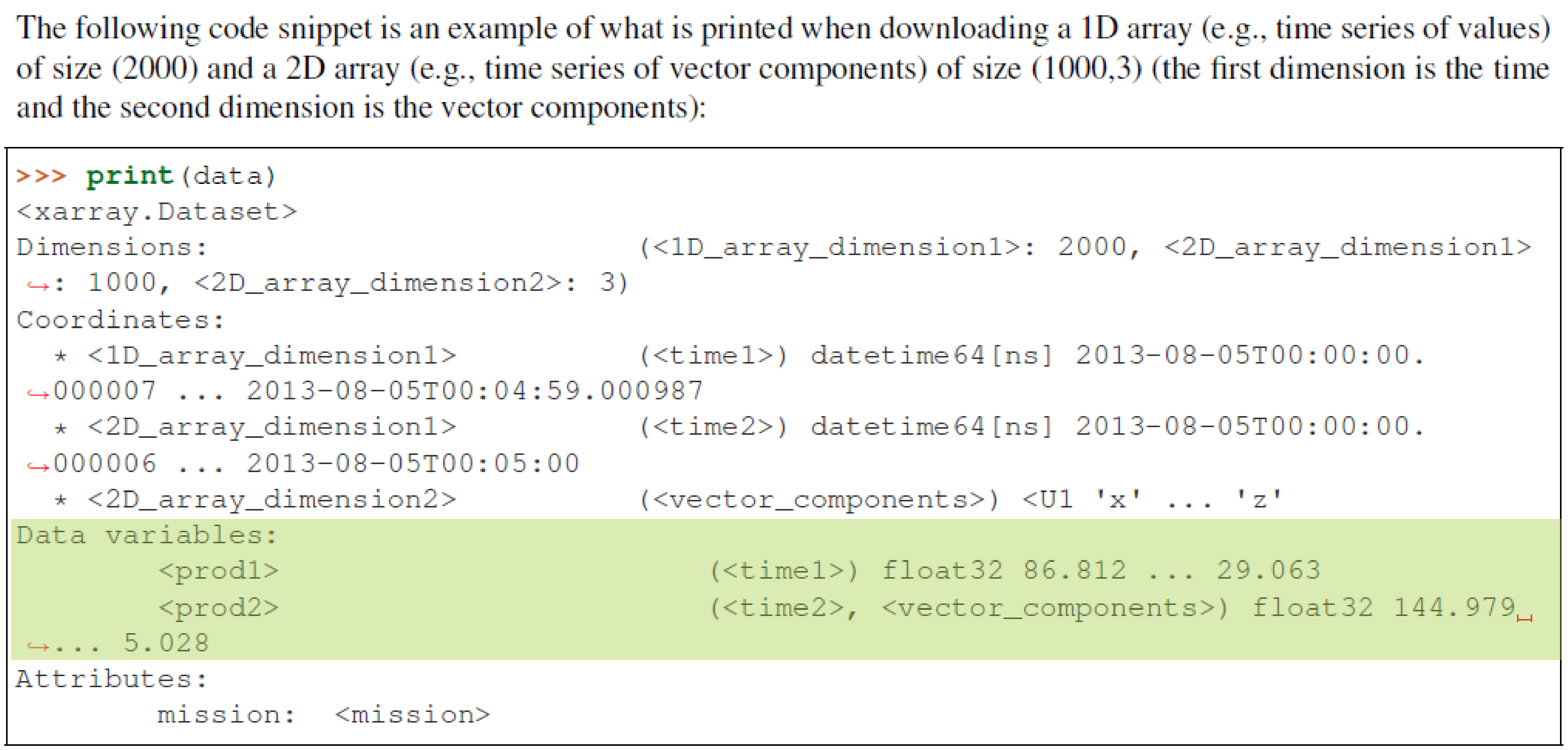
Data arrays labels



The following code snippet is an example of what is printed when downloading a 1D array (e.g., time series of values) of size (2000) and a 2D array (e.g., time series of vector components) of size (1000,3) (the first dimension is the time and the second dimension is the vector components):

```
>>> print(data)
<xarray.Dataset>
Dimensions:                          (<1D_array_dimension1>: 2000, <2D_array_dimension1>
↪: 1000, <2D_array_dimension2>: 3)
Coordinates:
  * <1D_array_dimension1>            (<time1>) datetime64[ns] 2013-08-05T00:00:00.
↪000007 ... 2013-08-05T00:04:59.000987
  * <2D_array_dimension1>            (<time2>) datetime64[ns] 2013-08-05T00:00:00.
↪000006 ... 2013-08-05T00:05:00
  * <2D_array_dimension2>            (<vector_components>) <U1 'x' ... 'z'
Data variables:
        <prod1>                          (<time1>) float32 86.812 ... 29.063
        <prod2>                          (<time2>, <vector_components>) float32 144.979␣
↪... 5.028
Attributes:
        mission:  <mission>
```

Mission tool: outputs



The following code snippet is an example of what is printed when downloading a 1D array (e.g., time series of values) of size (2000) and a 2D array (e.g., time series of vector components) of size (1000,3) (the first dimension is the time and the second dimension is the vector components):

```
>>> print(data)
<xarray.Dataset>
Dimensions:                          (<1D_array_dimension1>: 2000, <2D_array_dimension1>
↪: 1000, <2D_array_dimension2>: 3)
Coordinates:
  * <1D_array_dimension1>            (<time1>) datetime64[ns] 2013-08-05T00:00:00.
↪000007 ... 2013-08-05T00:04:59.000987
  * <2D_array_dimension1>            (<time2>) datetime64[ns] 2013-08-05T00:00:00.
↪000006 ... 2013-08-05T00:05:00
  * <2D_array_dimension2>            (<vector_components>) <U1 'x' ... 'z'
Data variables:
        <prod1>                          (<time1>) float32 86.812 ... 29.063
        <prod2>                          (<time2>, <vector_components>) float32 144.979␣
↪... 5.028
Attributes:
        mission:  <mission>
```

Data arrays variables

Mission tool: outputs

The following code snippet is an example of what is printed when downloading a 1D array (e.g., time series of values) of size (2000) and a 2D array (e.g., time series of vector components) of size (1000,3) (the first dimension is the time and the second dimension is the vector components):

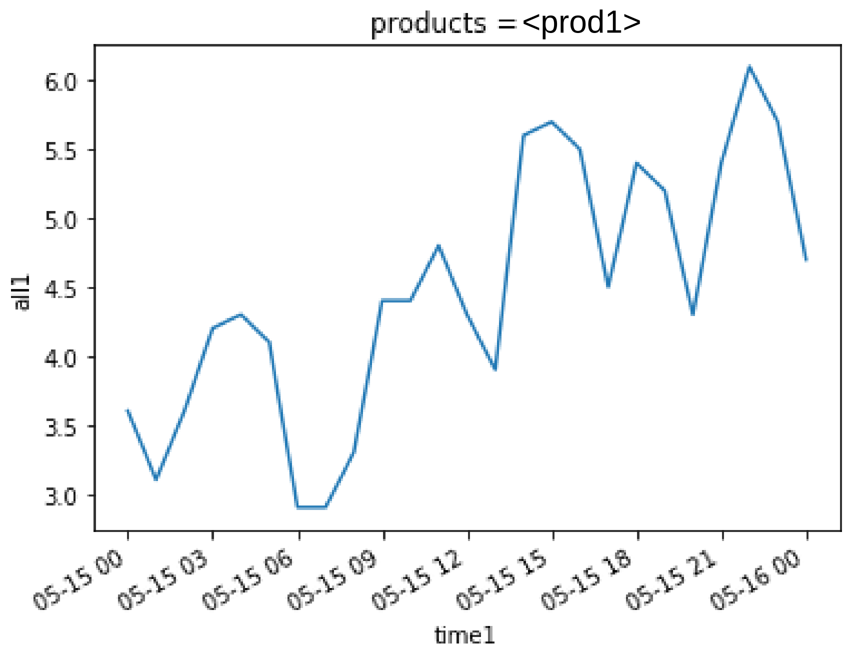
```
>>> print(data)
<xarray.Dataset>
Dimensions:                    (<1D_array_dimension1>: 2000, <2D_array_dimension1>
↪: 1000, <2D_array_dimension2>: 3)
Coordinates:
  * <1D_array_dimension1>      (<time1>) datetime64[ns] 2013-08-05T00:00:00.
↪000007 ... 2013-08-05T00:04:59.000987
  * <2D_array_dimension1>      (<time2>) datetime64[ns] 2013-08-05T00:00:00.
↪000006 ... 2013-08-05T00:05:00
  * <2D_array_dimension2>      (<vector_components>) <U1 'x' ... 'z'
Data variables:
        <prod1>                    (<time1>) float32 86.812 ... 29.063
        <prod2>                    (<time2>, <vector_components>) float32 144.979␣
↪... 5.028
Attributes:
        mission:  <mission>
```

Global metadata

Mission tool: outputs

- Product-specific metadata (units, spacecraft location, etc) are accessible for each product directly from the “Attributes” of the xarray object (e.g., data[<prod1>].attrs)
- Downloaded files are stored locally on the user’s computer in the folder: ~heliopy/data
- The module checks the latest available files from the open-access sources every time data is requested
- Easy plotting functionalities via the xarray.DataArray.plot() method (based on matplotlib)

```
>>> data[<prod1>].plot()
```

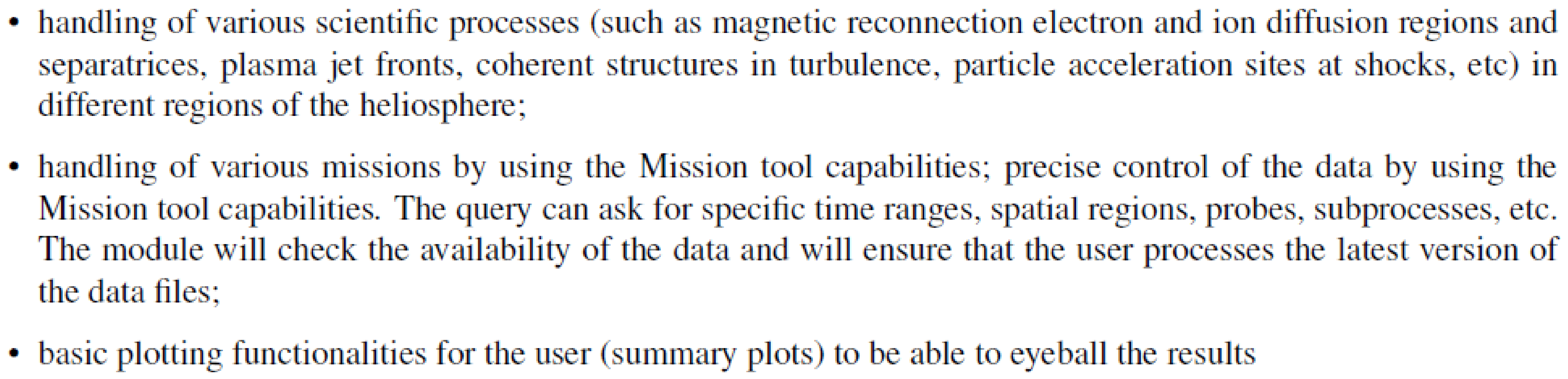


Outline

WARNING:
Some features of the Event Search tool have been updated in this document, thus this section is outdated in the recorded video

Event Search tool: main objectives

- **AIDApy subpackage to process data and create lists of events by using routines based on human experience-driven data selection for different physical processes of interest (magnetic reconnection, turbulence, particle acceleration, etc.)**



- handling of various scientific processes (such as magnetic reconnection electron and ion diffusion regions and separatrices, plasma jet fronts, coherent structures in turbulence, particle acceleration sites at shocks, etc) in different regions of the heliosphere;
- handling of various missions by using the Mission tool capabilities; precise control of the data by using the Mission tool capabilities. The query can ask for specific time ranges, spatial regions, probes, subprocesses, etc. The module will check the availability of the data and will ensure that the user processes the latest version of the data files;
- basic plotting functionalities for the user (summary plots) to be able to eyeball the results

Event Search tool: inputs

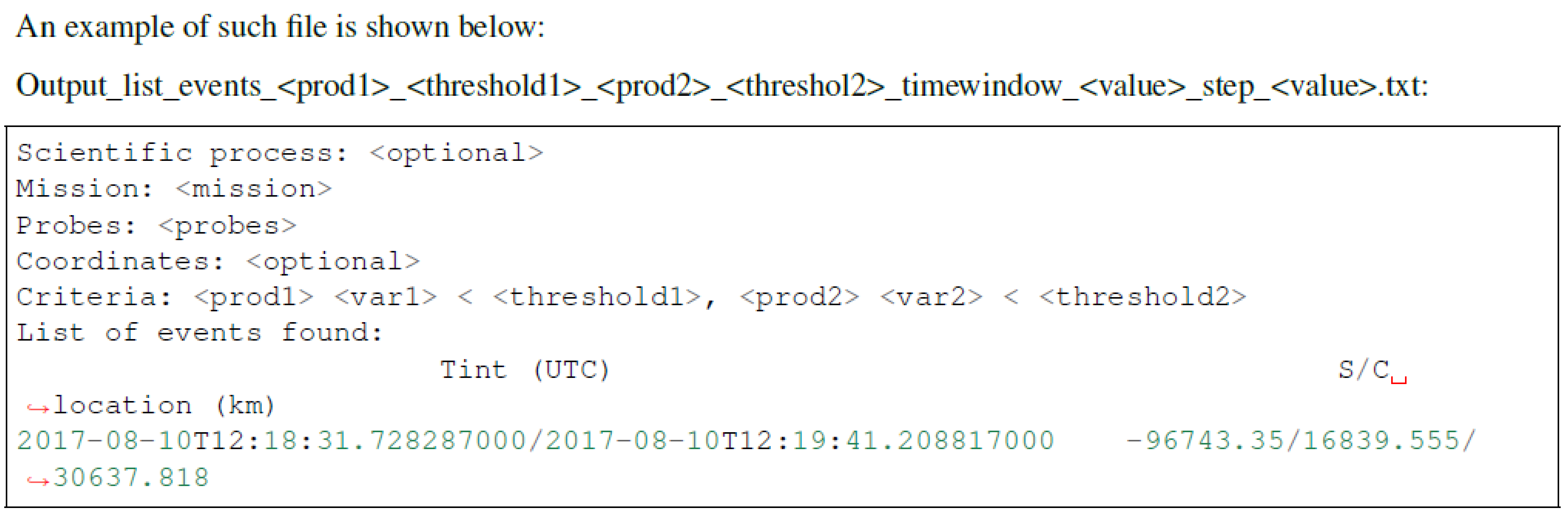
Below is a code snippet showing how to define the input parameters to the *event_search* function:

```python
# Import Python modules
from aidapy import event_search
from datetime import datetime
import numpy

# Define desired time interval
start_time = datetime(<year>, <month>, <day>, <hour>, <minute>, <second>)
end_time = datetime(<year>, <month>, <day>, <hour>, <minute>, <second>)

# Define parameters relevant to the queried scientific process
settings = {"parameters": {'mission': '<mission>',
                           'probes': ['<probe1>', '<probe2>', '<probe3>'],
                           'mode': '<mode>',
                           'time_window': '<value>',
                           'time_step': '<value>'},
            "criteria": lambda <var1>, <var2>: (<var1> < <threshold1>) &
                                               (<var2> > <threshold2>)}

# Search for events using the list_events subpackage
event_search(settings, start_time, end_time)
```

Note that the names of the parameters should be compliant with the AIDApy catalog of products

Event Search tool: outputs

1. An ASCII file containing the list of found events, parameters and metadata



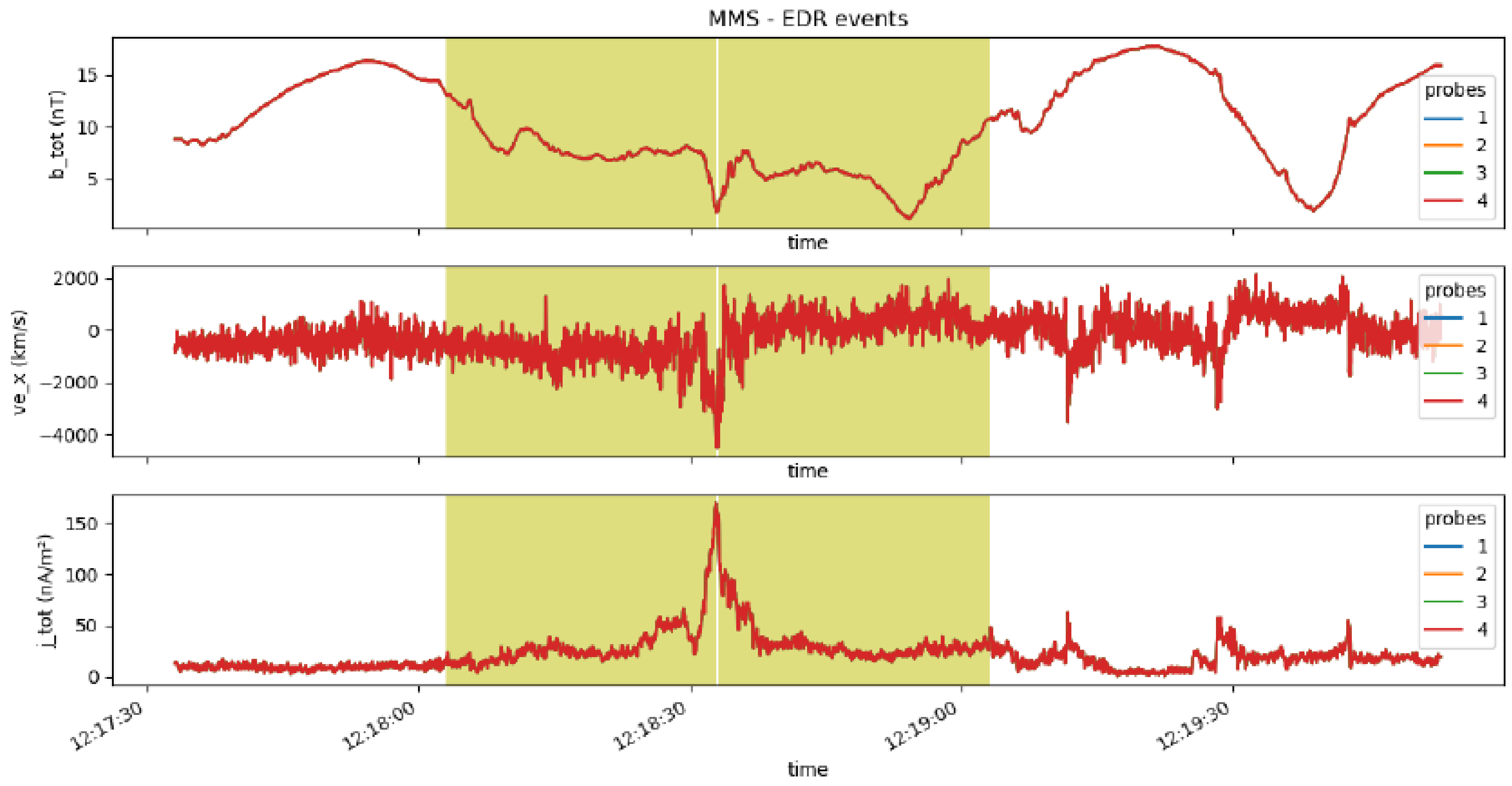
An example of such file is shown below:

Output_list_events_<prod1>_<threshold1>_<prod2>_<threshol2>_timewindow_<value>_step_<value>.txt:

```
Scientific process: <optional>
Mission: <mission>
Probes: <probes>
Coordinates: <optional>
Criteria: <prod1> <var1> < <threshold1>, <prod2> <var2> < <threshold2>
List of events found:
                        Tint (UTC)                                          S/C␣
↪location (km)
2017-08-10T12:18:31.728287000/2017-08-10T12:19:41.208817000    -96743.35/16839.555/
↪30637.818
```

Event Search tool: outputs

2. A summary plot of queried products with highlighted found events



Outline

1. Introduction
2. Space missions
3. The Mission tool
4. The Event Search tool
5. **Examples**

Let’s see some practical examples on Jupyter Notebooks!