EFDA
EUROPEAN FUSION DEVELOPMENT AGREEMENT
Task Force
INTEGRATED TOKAMAK MODELLING

Working Session / Code Camp
March 2011

The ITM general grid description: A tutorial

H.-J. Klingshirn

A general grid description for the ITM

Why?

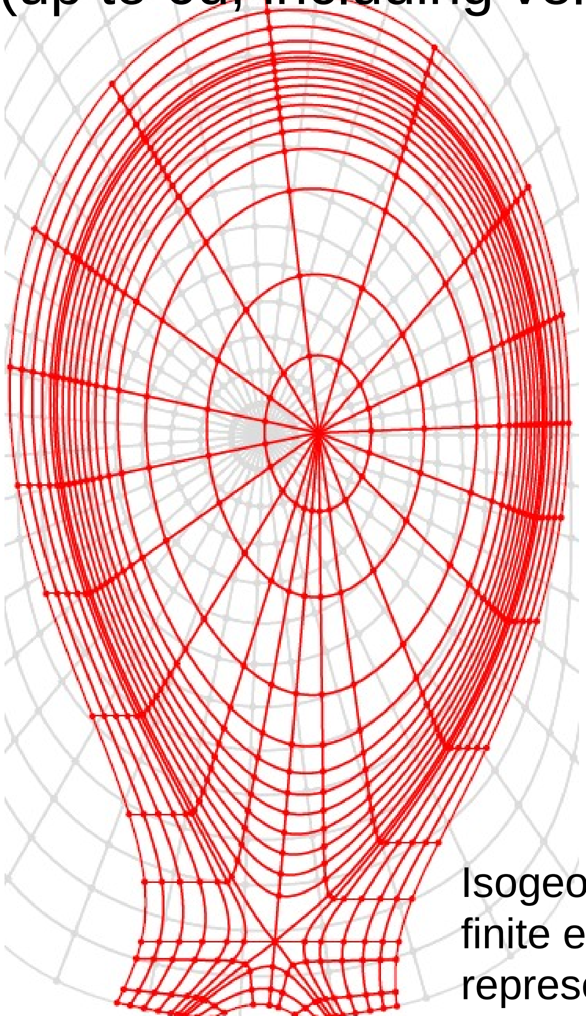
- Codes that treat complex geometries have to use possibly complex grids
- Different codes, different numerical schemes, different grids – but only one CPO!
- This is especially true for the 2d/3d edge codes that have to resolve the vessel geometry

(Other projects might be in a similar situation)

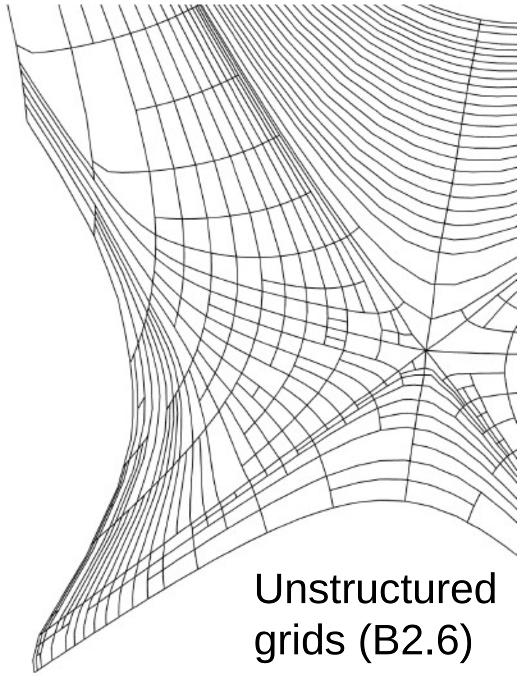
The general grid description tries to provide a practical, reusable solution for a wide range of spatial discretizations

What is the goal?

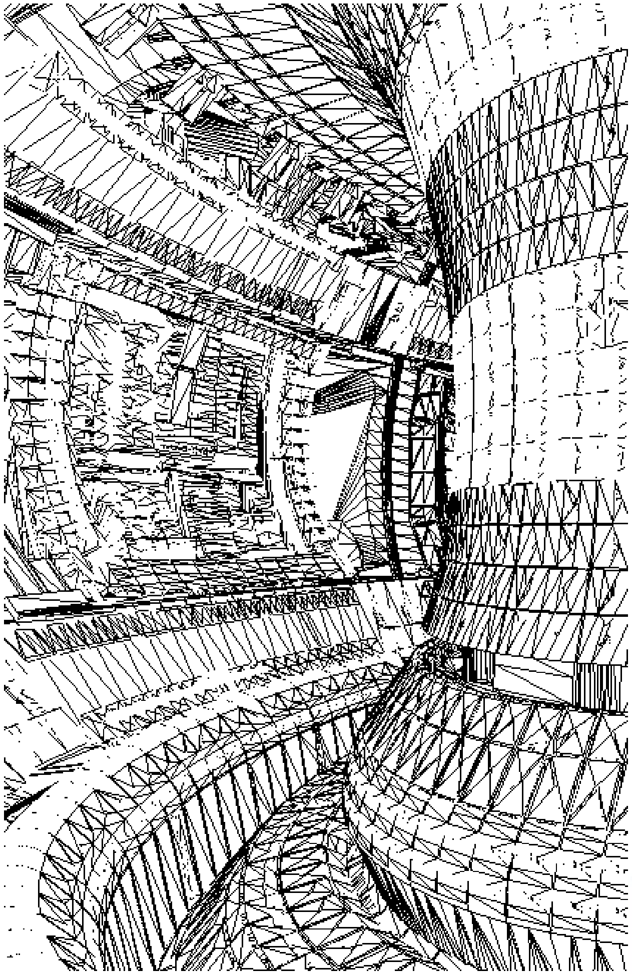
Efficient handling of complex grids and data representations (up to 6d, including velocity space)...



Isogeometric finite element representation (JOREK)



Unstructured grids (B2.6)

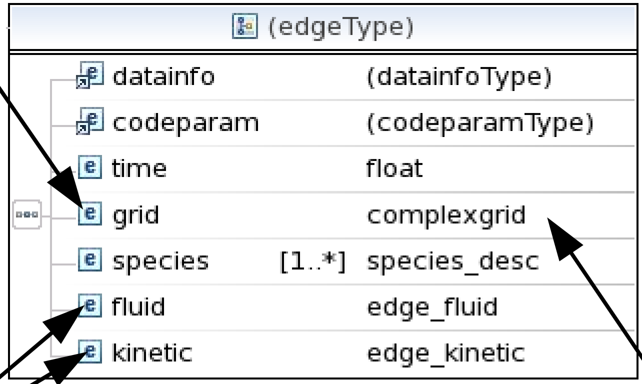


3d tetrahedron grids (ASDEX Upgrade vessel)

...while still being easy to use for simple grids (e.g. structured grids)!

CPO design example: Edge CPO (new 4.09a version)

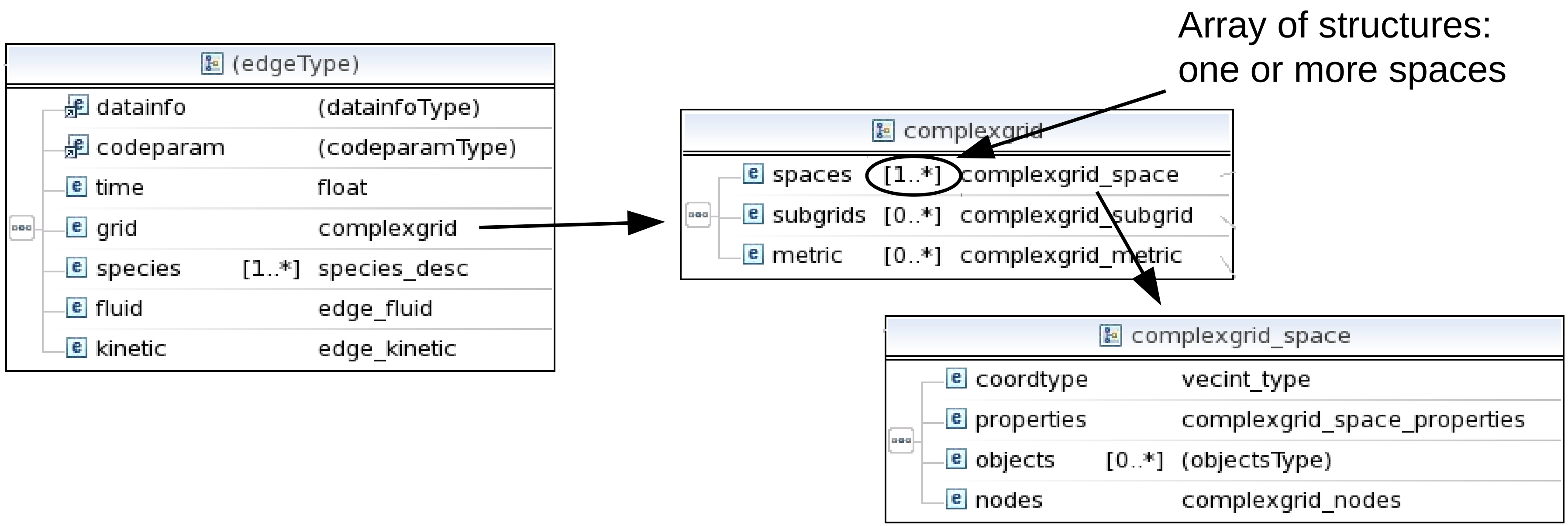
Standardized general grid description at top level



Data stored on grid: refers to grid description (via *subgrids*, see later)

The types of the grid description and the data fields (see later) are standardized and can be used in any CPO → allows use of standardized tools (plotting, interpolation, ...)

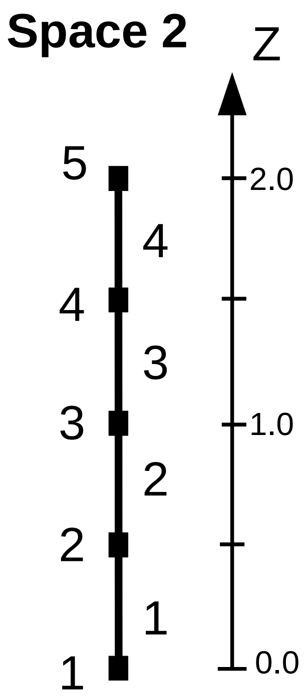
Grid description: Details



Some basic definitions (examples to follow):

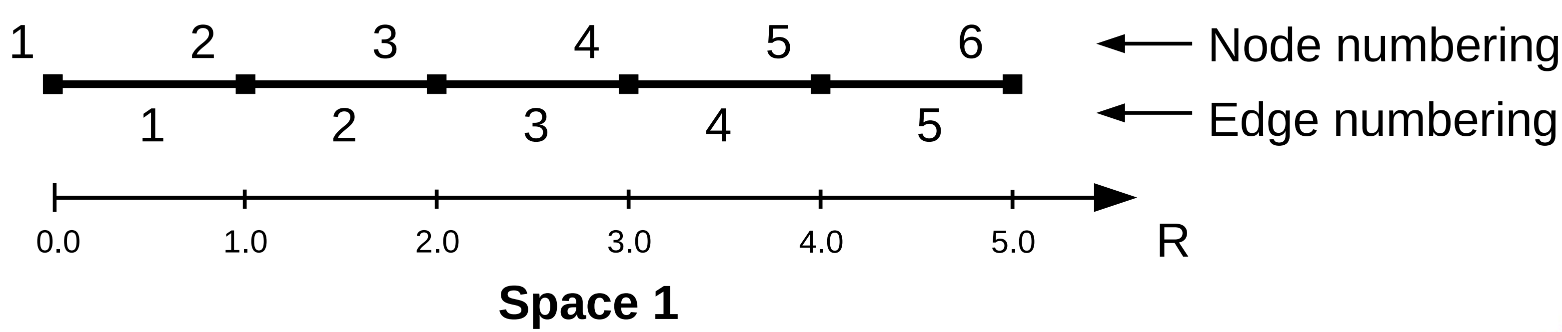
- A grid is composed out of individual ***spaces***, which are themselves discretizations of physical space (but possibly of lower dimension than the grid)
- Objects in a space (called ***grid subobjects***) are defined *explicitly*.
- The objects in the grid (***grid objects***) are then defined *implicitly* by combining the subobjects from the spaces.

Grid & space explained: Simple 2d grid example

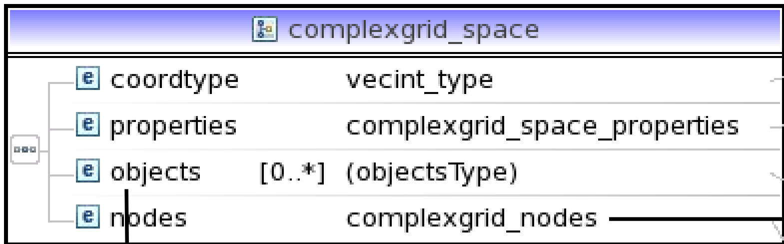


Discretization of 2d space in (R,Z) coordinates:

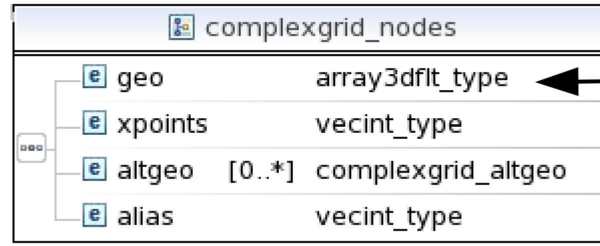
- Define two one-dimensional spaces to discretize the R and Z direction
- Explicitly define the nodes and edges (subobjects) in the spaces
  → stored in the object description of the space



Storing subobject information

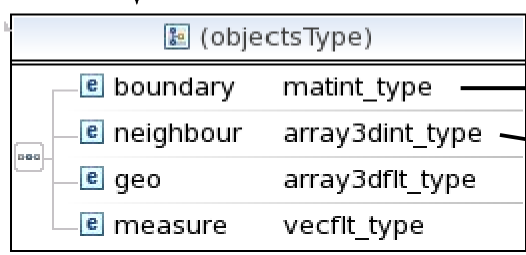


0d subobjects: nodes



Node positions

Nd subobjects (N>0):
assembled from (N-1)d subobjects



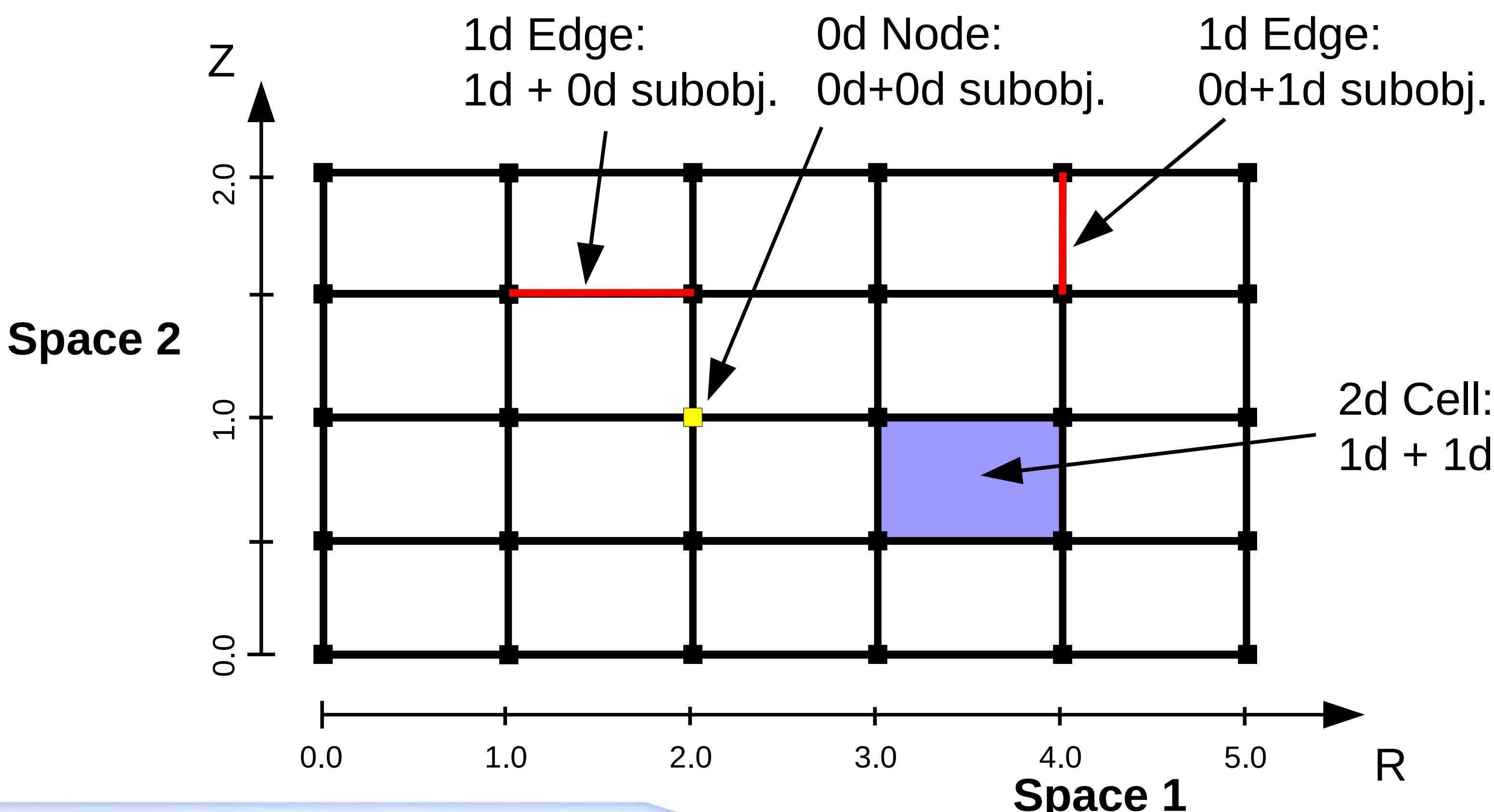
Boundaries of objects
Connectivity

These fields store object indices

Note: the subobjects in a space are defined explicitly. In the space they can be identified uniquely by their dimension and their index.

Simple example (ctd.): Implicit definition of grid objects

- A grid object is built by taking one subobject from every space and combining them.
- The complete grid is given as all possible combinations of subobjects.

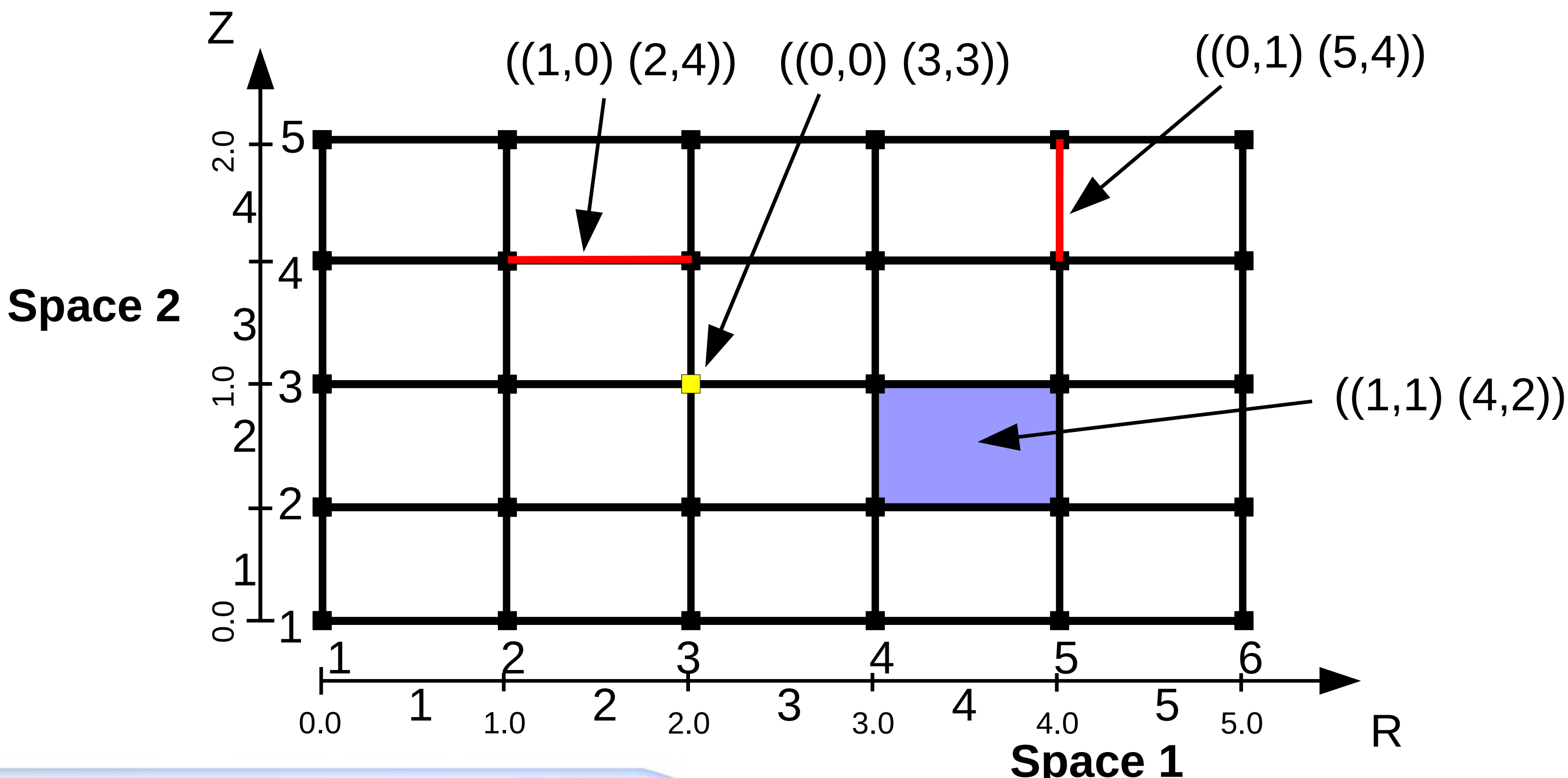


Object descriptor: Notation

Object Descriptor: ((c1, c2,...,cn) (i1,i2,...,in))

Object class:
cj = subobject dim. in space j

Object index:
ij = subobject index in space j



Object descriptor: Notation

Object Descriptor: ((c1, c2,...,cn) (i1,i2,...,in))

Object class:
cj = subobject dim. in space j

Object index:
ij = subobject index in space j

Some notes:

- An object descriptor uniquely identifies a grid object
- An extension of the notation is available to easily denote groups of grid objects (see later: subgrids)

Implicit global object order

- The subobjects in every space have an explict (*local*) order (simply the order in which they are defined)
- For the implicitly defined grid objects, a *global* order is imposed by adopting a simple counting convention (think linear address computation for multidimensional Fortran arrays):

Grid objects of a common object class are counted by varying the leftmost index first.

Every grid object can therefore be uniquely identified by:

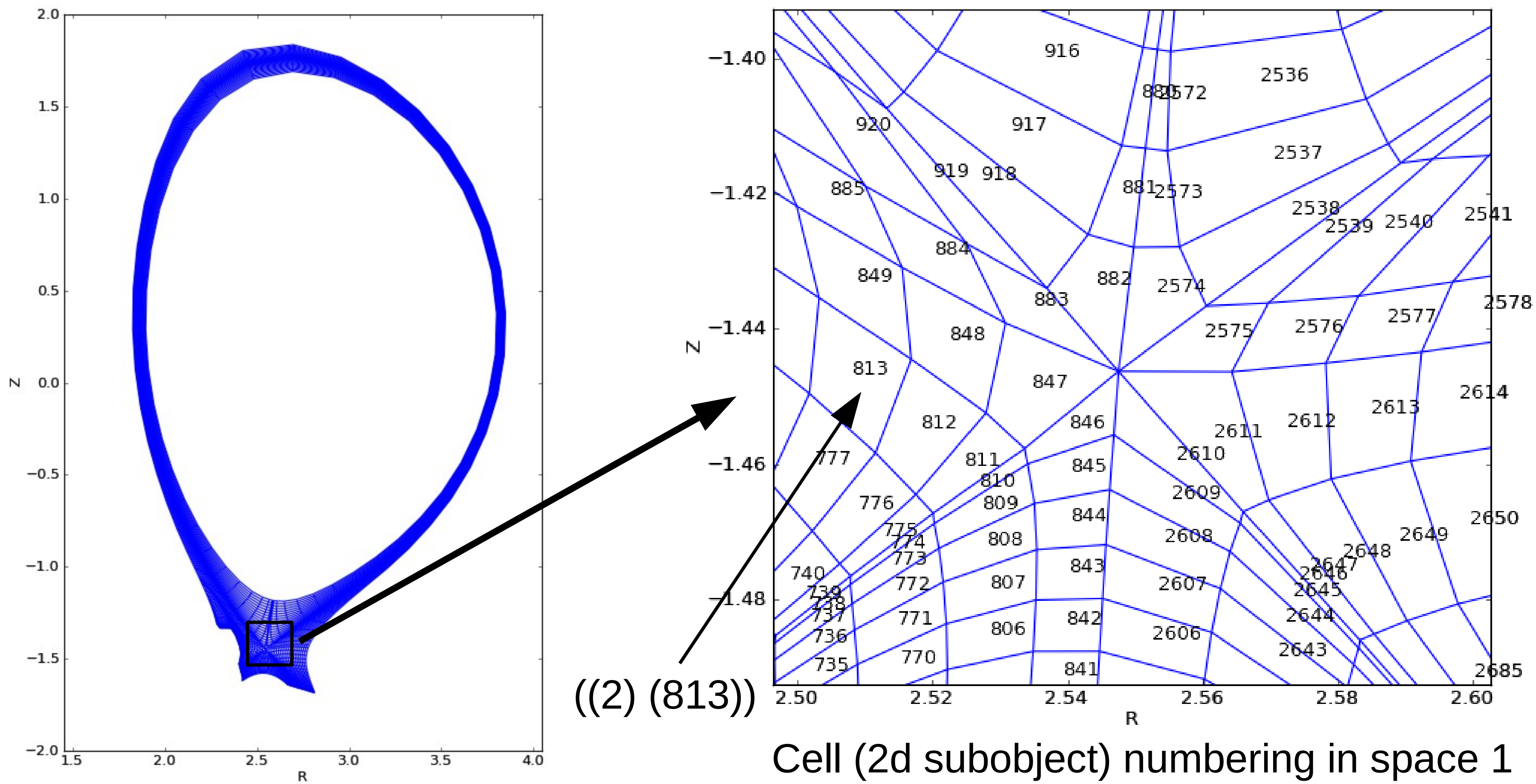
Its object descriptor:
((c1, c2,...,cn) (i1,i2,...,in))

or

Its object class and
global index ig:
((c1, c2,...,cn) ig)

(More on this later in the subgrid part)

Slightly more complex example: A 2d edge code grid

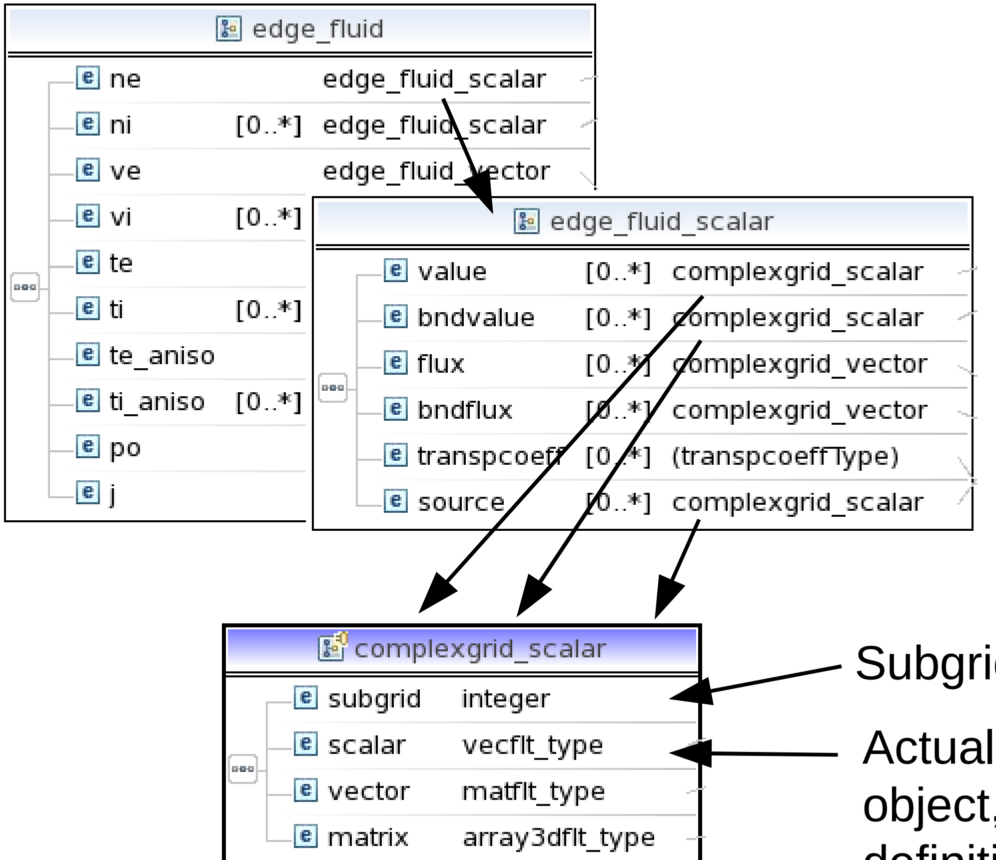


Cell (2d subobject) numbering in space 1

A 2d grid composed out of one 2d space that defines an effectively unstructured grid of quadrilateral cells

(Shameless plug: all plots done with the evolving Python implementation of the grid service library.)

Storing data on grids (again shown with the edge CPO)



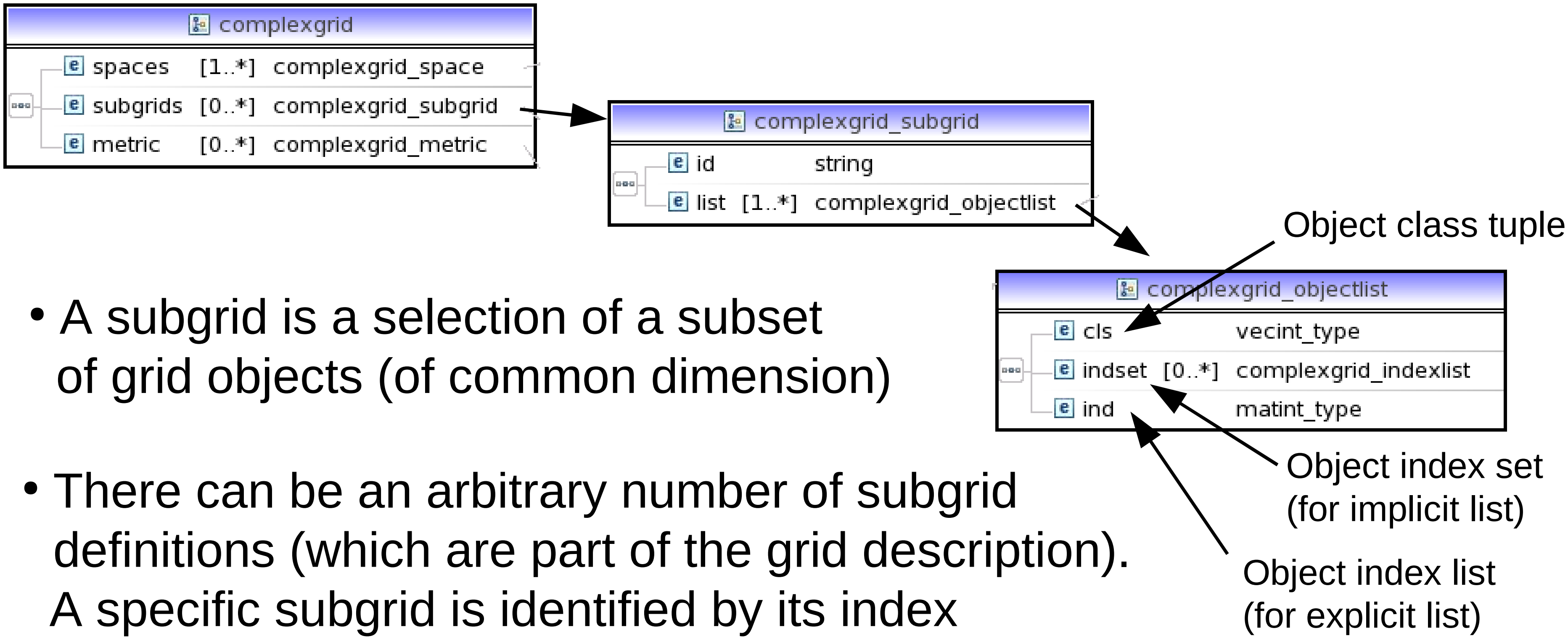
- Data is stored on a ***subgrid***
- *Subgrid* = list of grid objects
- The data is then stored simply as a vector, one entry per object in the subgrid

Subgrid index

Actual data: one entry per subgrid object, in the order defined in the subgrid definition

CPO design:
fields that hold data stored on the grid should be of the generic data type (complexgrid_scalar) → enables use of general tools

Subgrid definition



- A subgrid is a selection of a subset of grid objects (of common dimension)
- There can be an arbitrary number of subgrid definitions (which are part of the grid description). A specific subgrid is identified by its index
- A subgrid is a list of *object lists*. Each object list can be either
  - ***explicit***: an explicit list of object descriptors
  - ***implicit***: an implicit list of object descriptors, selecting a range or an entire class of objects (see next slide)

Subgrids (ctd.): Implicit object list notation

Object List Descriptor: $\underbrace{((c1, c2,...,cn)}\ \underbrace{(s1,s2,...,sn))}$

Object class: cj = subobject dim. in space j
(same as in object descriptor)

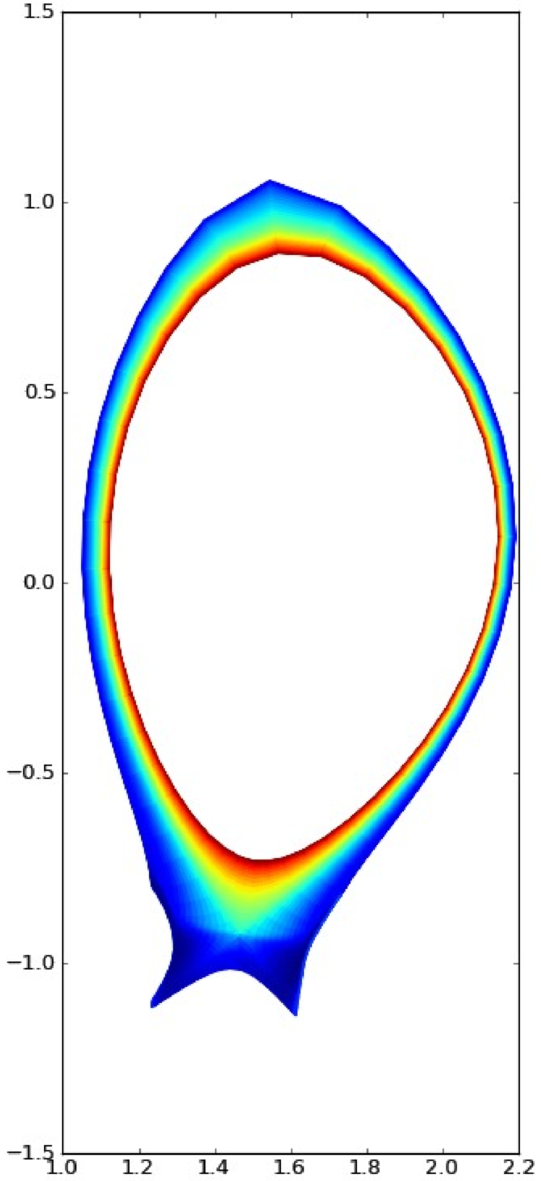
Object index set: specifies multiple indices

sj = [i1, i2, …, iN]	→ explicit list of indices
sj = (i1,i2)	→ range of indices from i1 to i2
sj = UNDEFINED	→ all possible indices

Object order: implicit object lists inherit the implicit object order intrinsic to the underlying grid definition

Why is this important? Space splitting and implicitly defined object order allow efficient handling of datasets on very large (5d, 6d,...) structured grids.

Subgrids example: B2 2d cell data

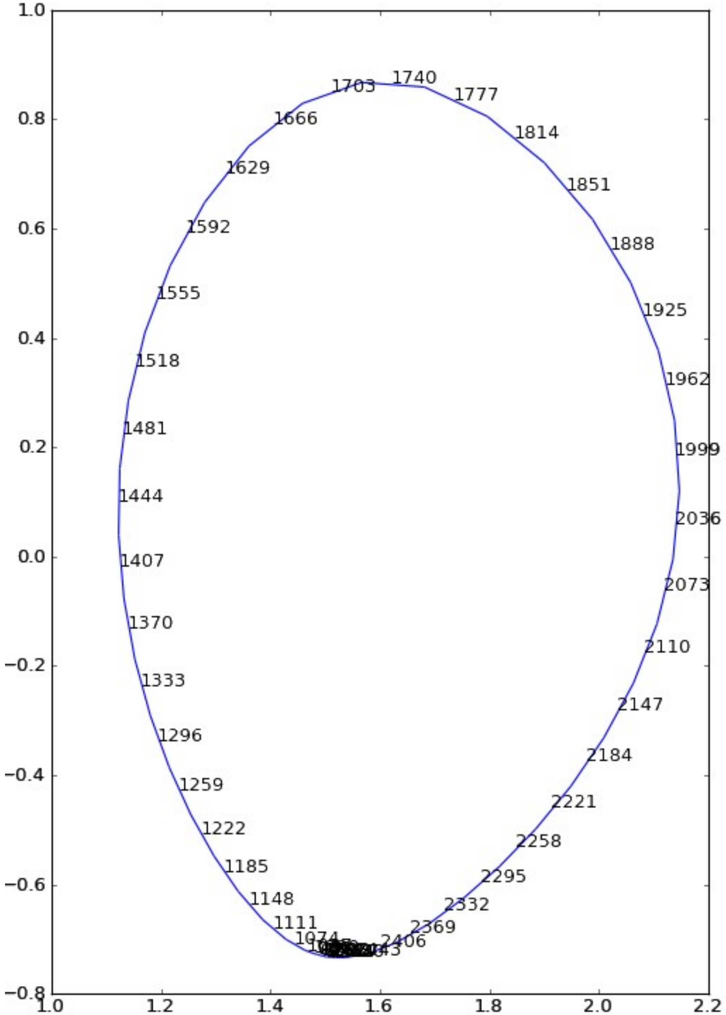


- B2 plasma solution, written to edge CPO
- Subgrid: implicit list of all 2d cells (~3500 cells)
- Patch plot of electron density

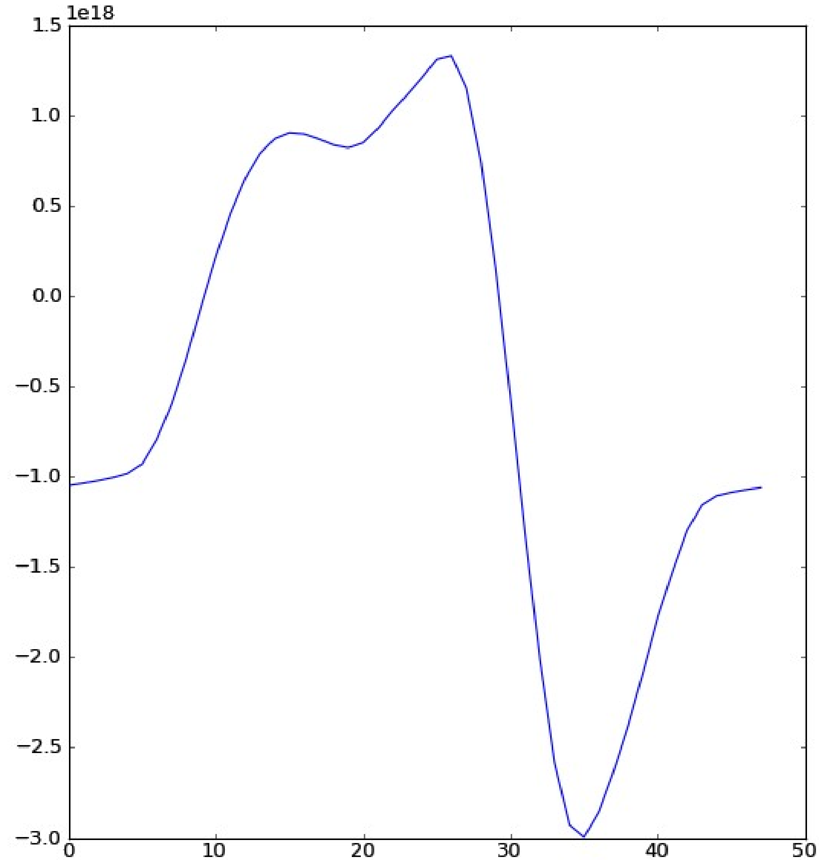
(Another shameless plug: all plots done with the evolving Python implementation of the grid service library.)

Subgrid example: B2 core boundary flux

- B2 grid, core boundary. Subgrid: explicit list of 48 core faces.



Core boundary subgrid



Core boundary electron flux (plotted vs. face number along core boundary)

The grid service library

This is all so complicated!

- You have to understand the grid description.
- You have to understand object descriptors and subgrids.
- You have to read and write your data to these strange data structures.

→This is the price we have to pay for having a general description that allows for complex grids.

I don't want to do this for my simple grids!

Simple things should be simple.

The grid service library is for you.

The grid service library (ctd.)

The grid service library is designed to help you when working with the general grid description:

- **Low-level routines**:
  - access basic information (for spaces, objects, subgrids…)
  - helper routines for reading and assembling complex grids (e.g. unstructured grids)

- **High-level routines**:
  - Easy-to-use read/write routines for grid and data for specific classes of grids (e.g. structured grids)
  - General visualization (1-3d plots, cuts/projections,...)
  - For the future: simple data transformations (interpolation, operators, …)

High level service routines Example: structured grid & data

Using the Fortran 90 implementation:

grid description structure in the CPO

write 2d grid

```
call gridSetupStructuredAlt( cpo%grid, ndim = 2, &
    & c1 = COORDTYPE_R, x1 = r_coords, &
    & c2 = COORDTYPE_Z, x2 = z_coords )
```

dim. 1: R coordinates

dim. 2: Z coordinates

```
call gridStructWriteData( cpo%grid, GRID_STRUCT_CELLS, data, cpo%field )
```

subgrid identifier (predefined)

data (2d array)

data field in the CPO (complexgrid_scalar)

Reading: equivalent routines exist

For simple grids, the details of the grid description are hidden completely from the user by the grid service library

Grid service library: Implementations

Supported languages:

- Fortran 90:
  - reference implementation for data structure I/O
- C
  - will have functionality similar to F90 version
- Python
  - focus on plotting functionality

Notes:

- Besides a common core set of functions, the features of the different implementations will differ (depending on the needs)
- The library is currently still evolving towards a first release (which is expected after the release of 4.09a)
- SVN: http://gforge.efda-itm.eu/svn/itmshared/branches/grid/

Thanks!

For questions: hmk@ipp.mpg.de