MITK Tutorial

Theory Session

Caspar Goch & Stefan Kislinskiy

Overview

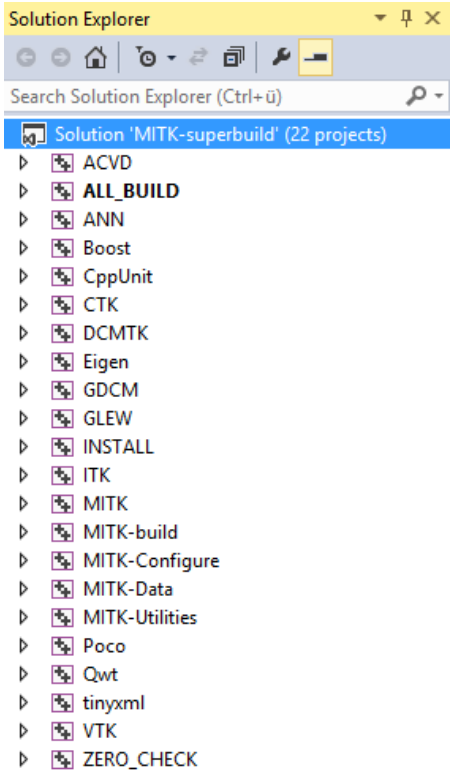
- MITK (super-)superbuild
  - Third-party vs. MITK
  - Source/build tree layout
- MITK architecture
  - High-level
    - Modules, plugins, command line apps
  - Low-level
    - mitk::BaseData environment

MITK (super-)superbuild

- Two build levels
  - MITK superbuild
  - MITK build
- Same source directory, nested build directories

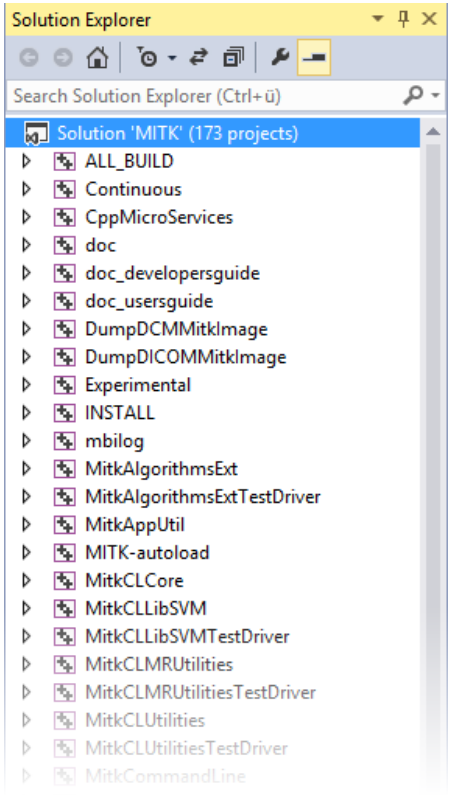
MITK (super-)superbuild

- MITK superbuild level
  - Download, configure, generate, and build third-party toolkits
  - Configure, generate, and build MITK
  - Switch on third-party toolkits via MITK_USE_<toolkit> CMake variables



MITK (super-)superbuild

- MITK build level
  - Successfully run superbuild first
  - Already built by superbuild
  - Re-configure, generate, and build as needed
    - Switch on MITK plugins



MITK (super-)superbuild

- Source tree layout

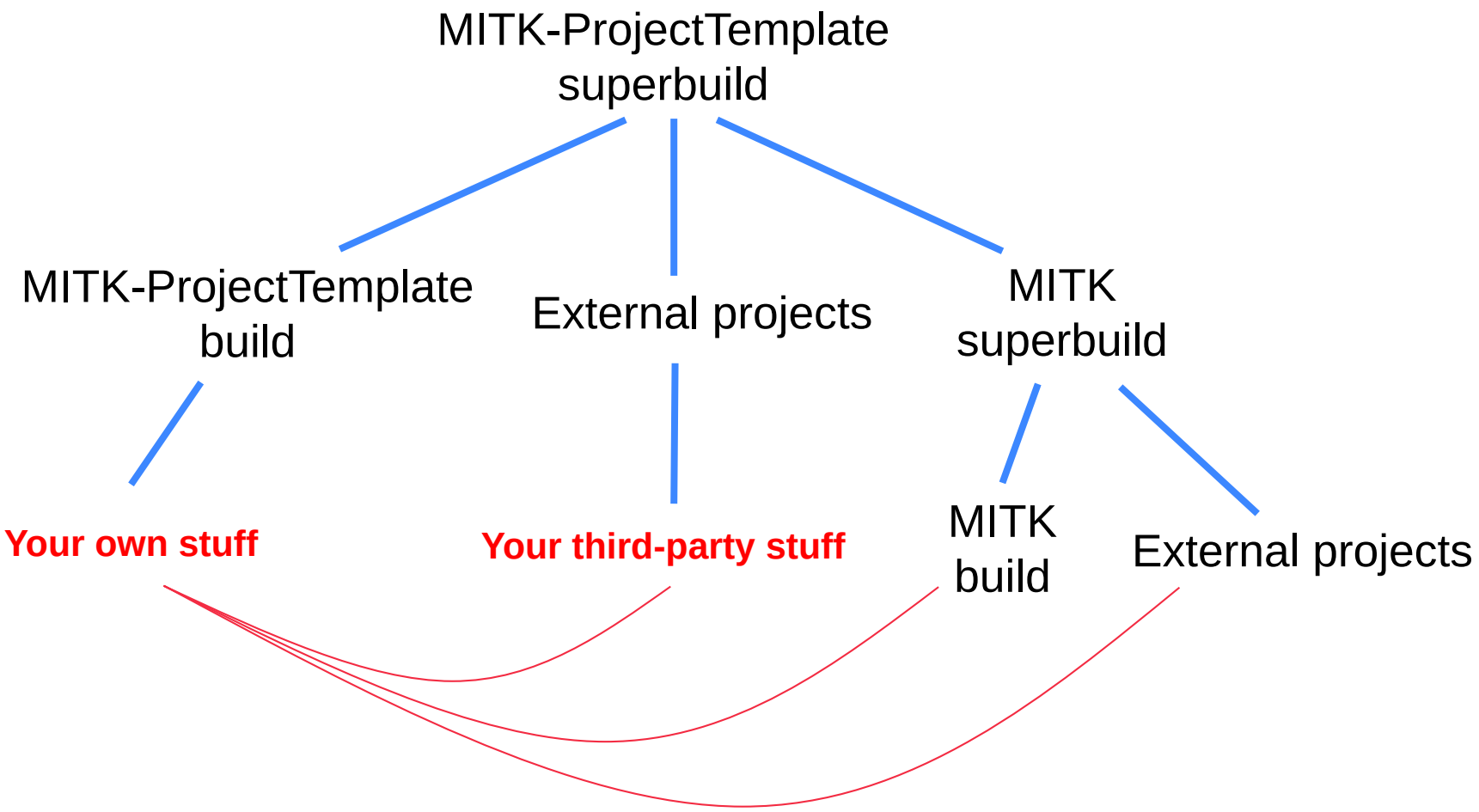
```
/
  Applications        ← e.g. MITK Workbench
  CMake               ← CMake macros and functions
  CMakeExternals      ← Third-party toolkit scripts
  Documentation
  Examples
  Modules             ← Modules & command line apps
  Plugins             ← Application plugins
  Utilities
```

MITK (super-)superbuild

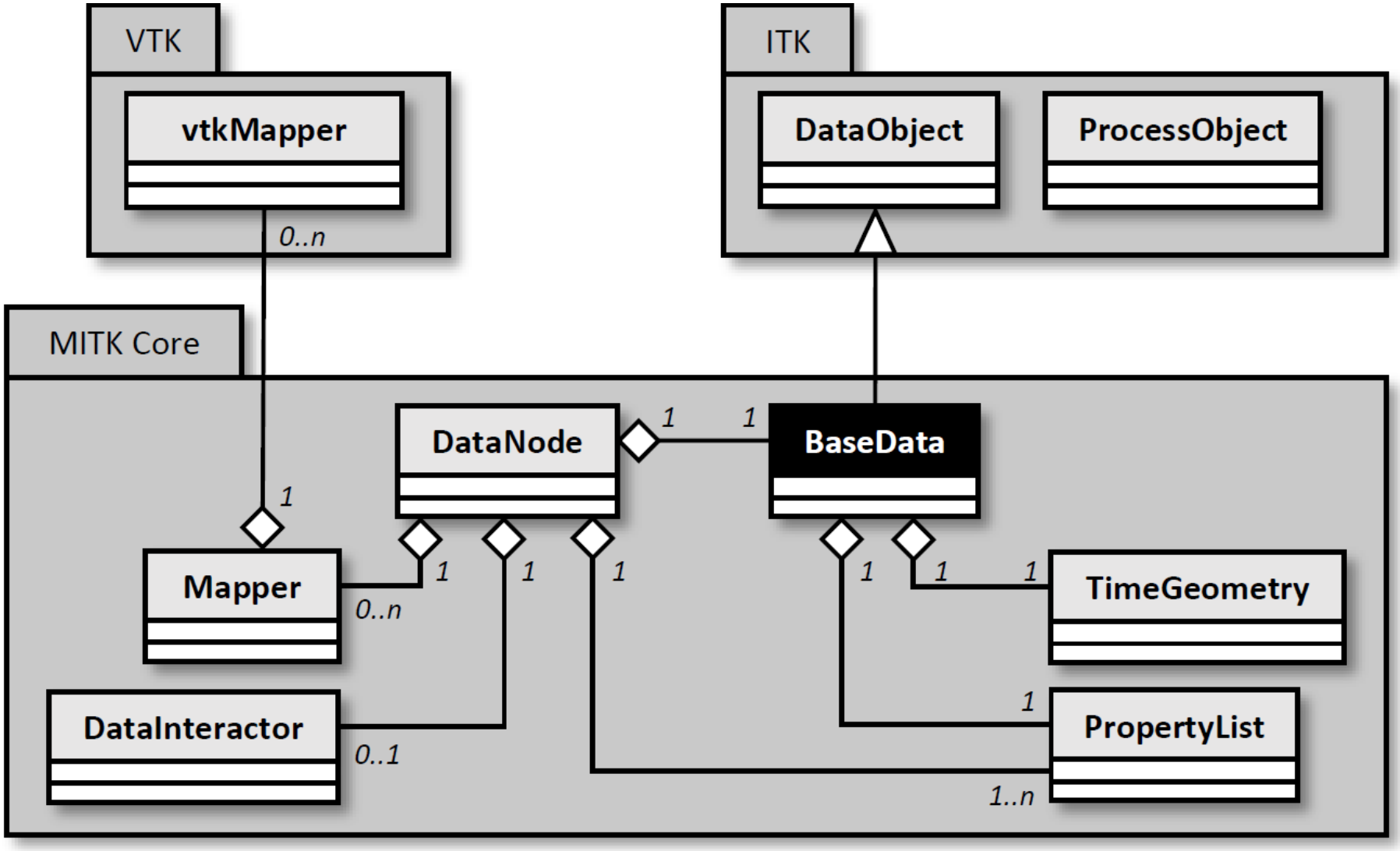
- Superbuild tree layout

```
/                 ← Superbuild MSVC solution / make directory
  MITK-build      ← MITK build MSVC solution / make directory
    bin
    lib
  MITK-Data       ← useful test data (BUILD_TESTING = ON)
  ep              ← External projects (third-party)
    bin     }
    lib     } ←     install destinations
    include }
    src           ←     source/build directories
```

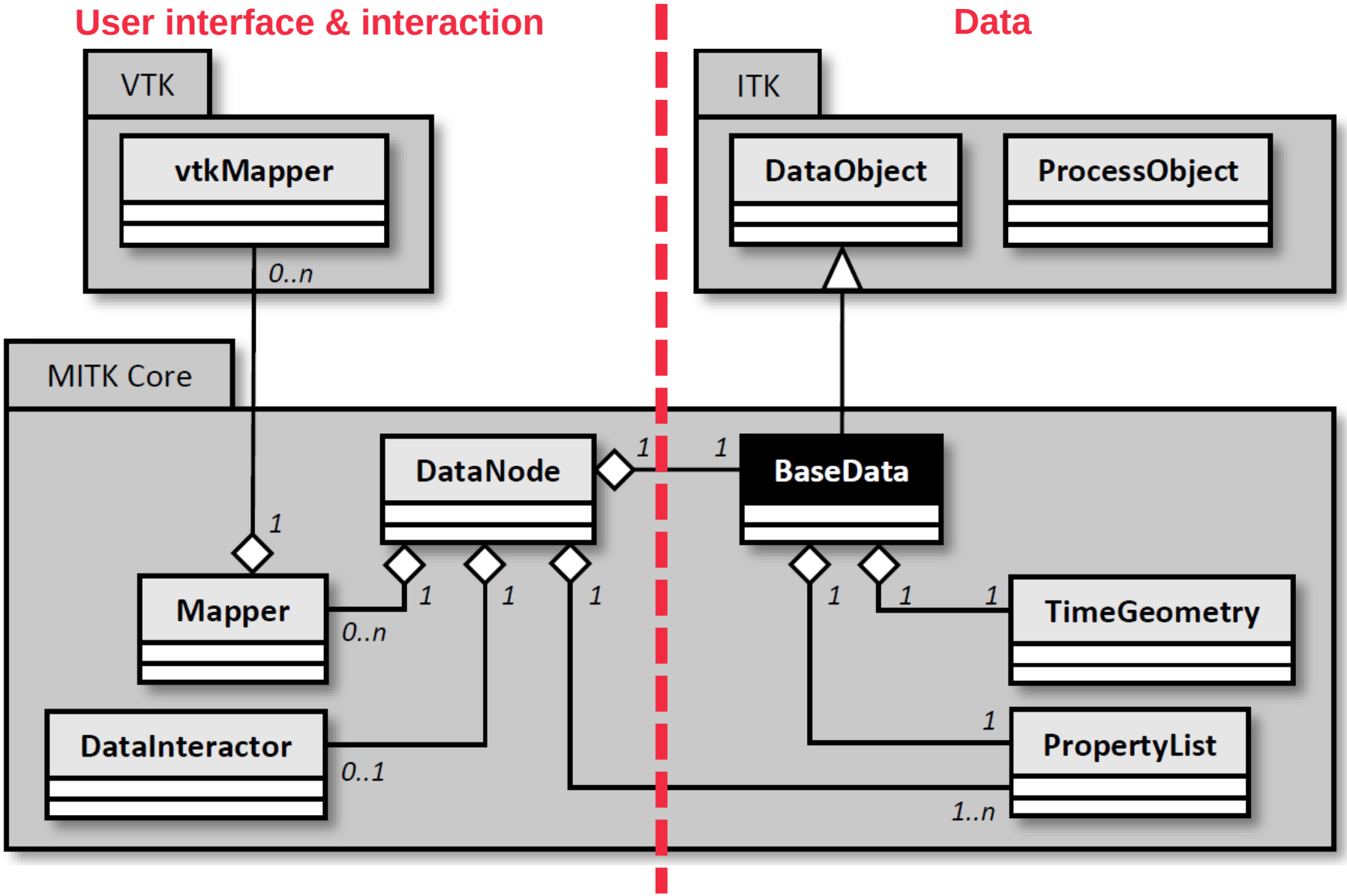
MITK (super-)superbuild



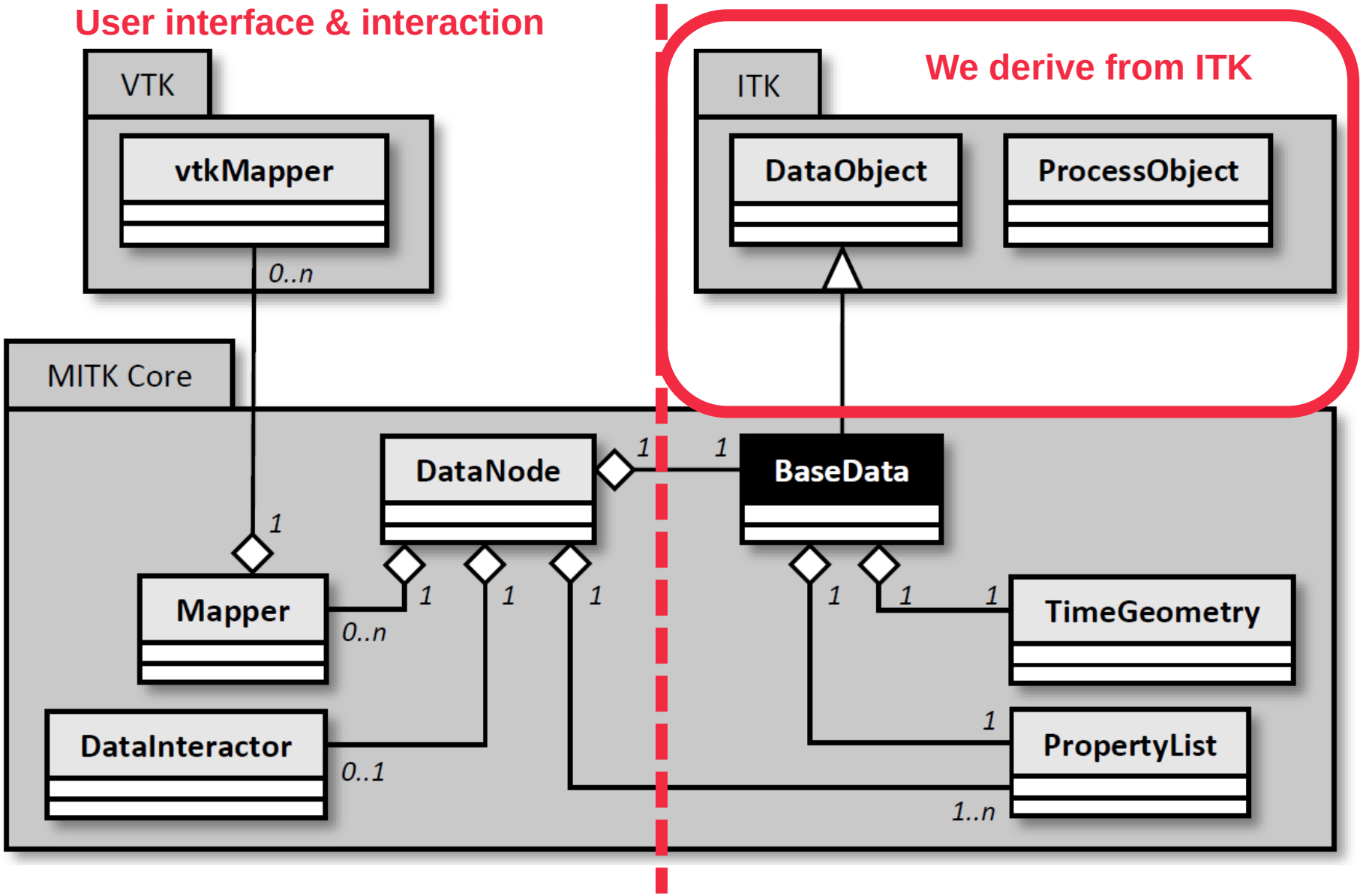
MITK architecture: Low-level



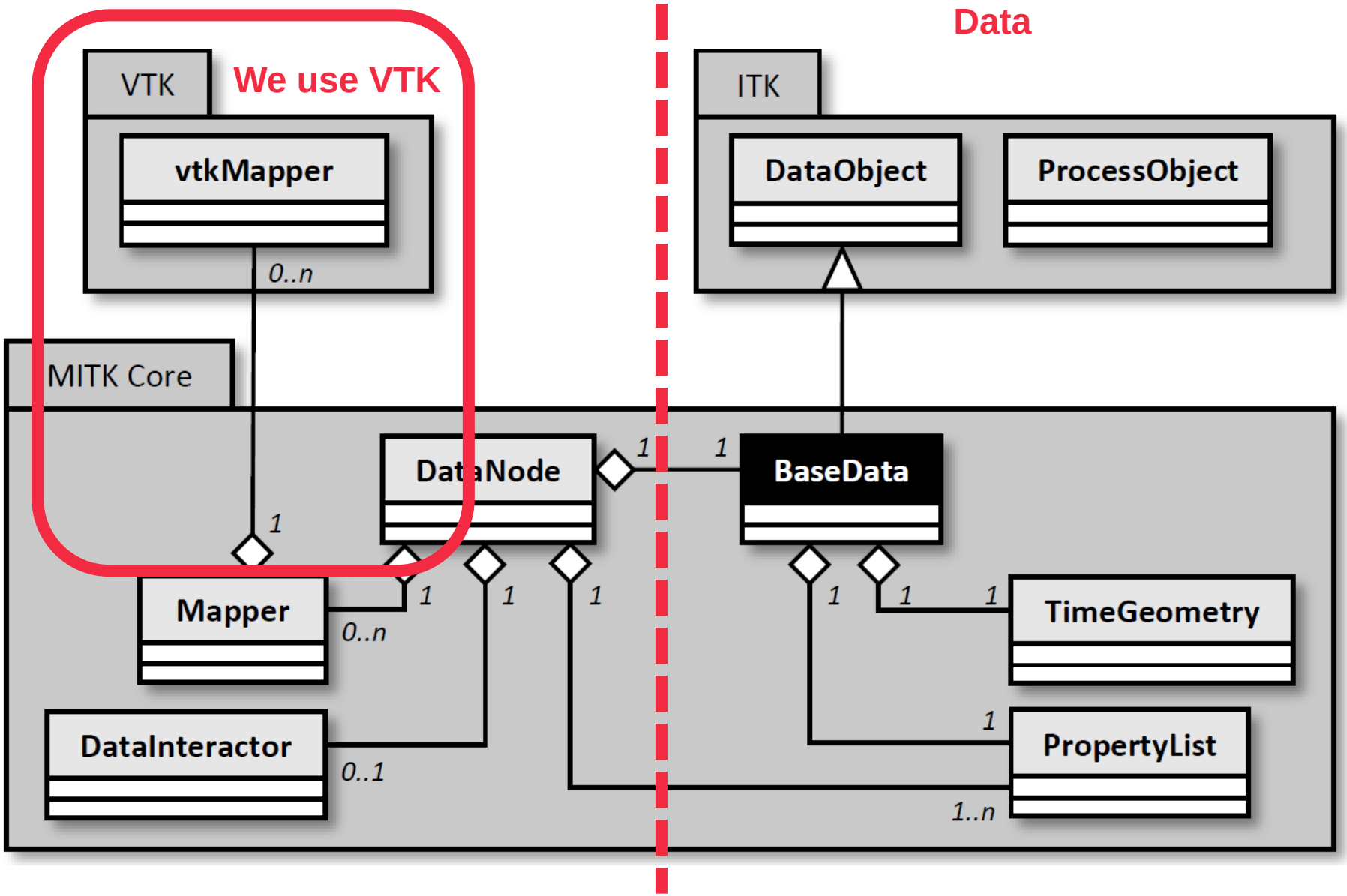
MITK architecture: Low-level



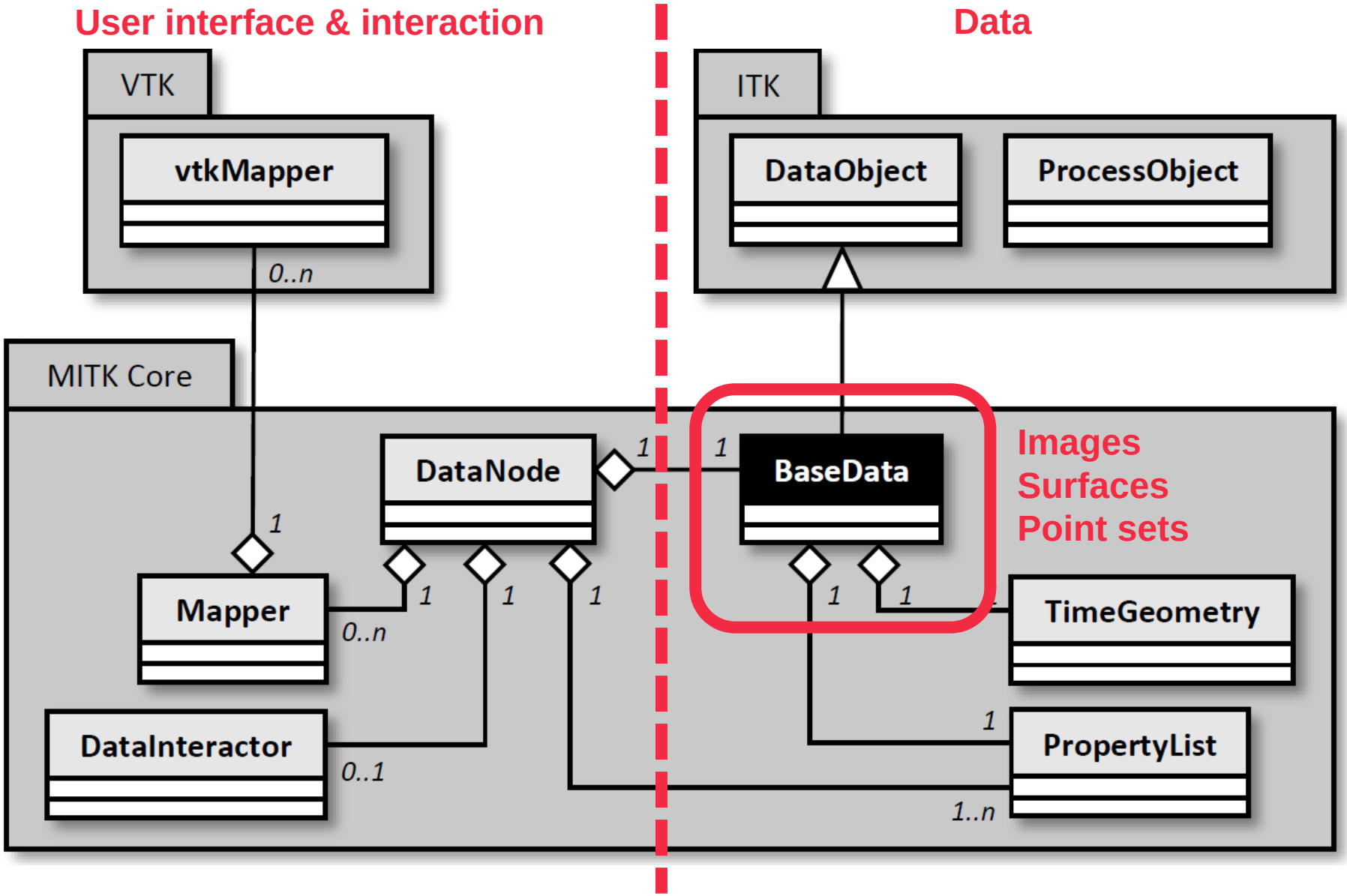
MITK architecture: Low-level



MITK architecture: Low-level



MITK architecture: Low-level



MITK derives from ITK, uses/wraps VTK

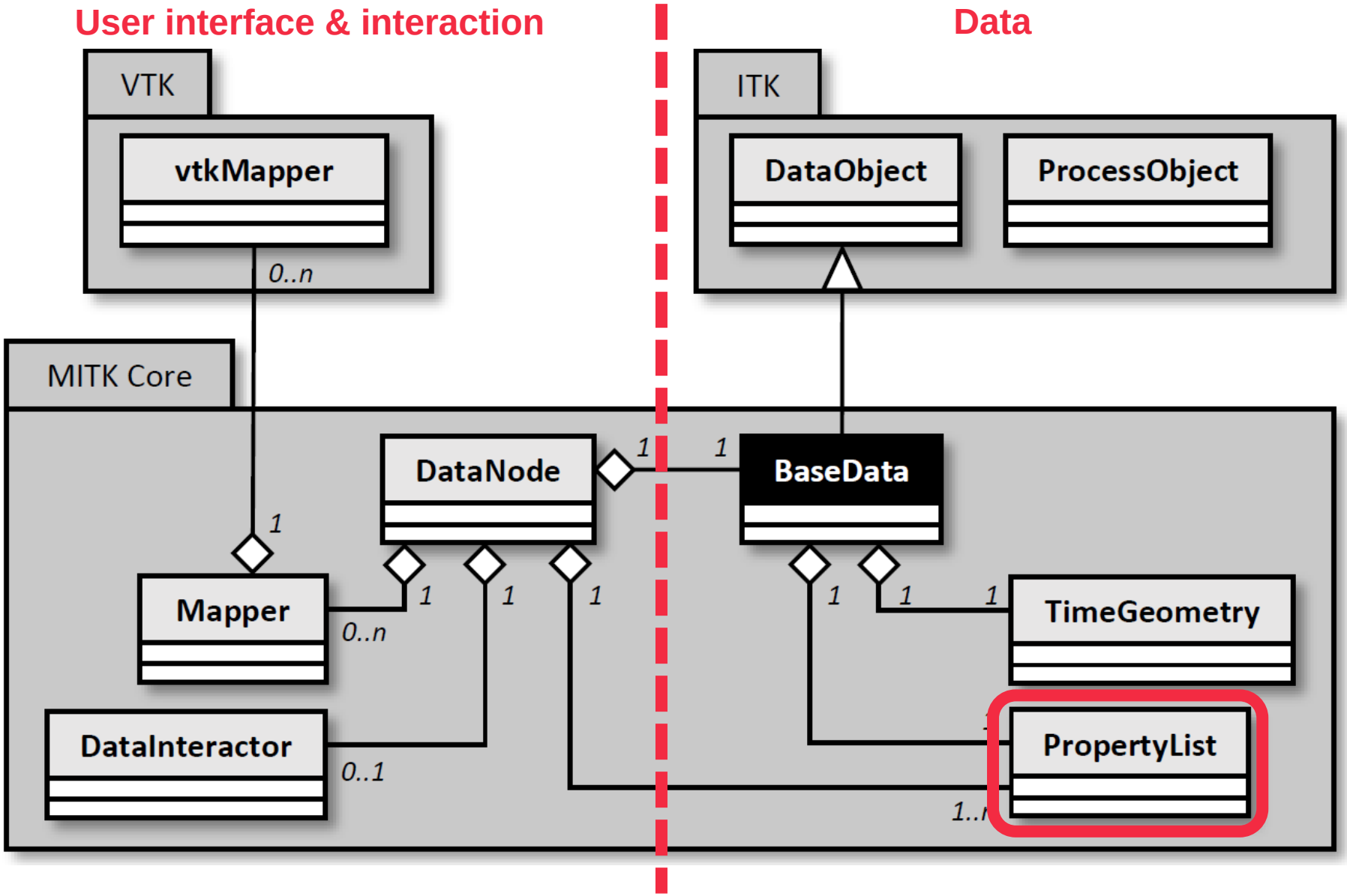

```
class BaseData : public itk::DataObject
{
};
```

ITK

```
class Surface : BaseData
{
  std::vector<vtkPolyData*> m_PolyDatas;
};
```

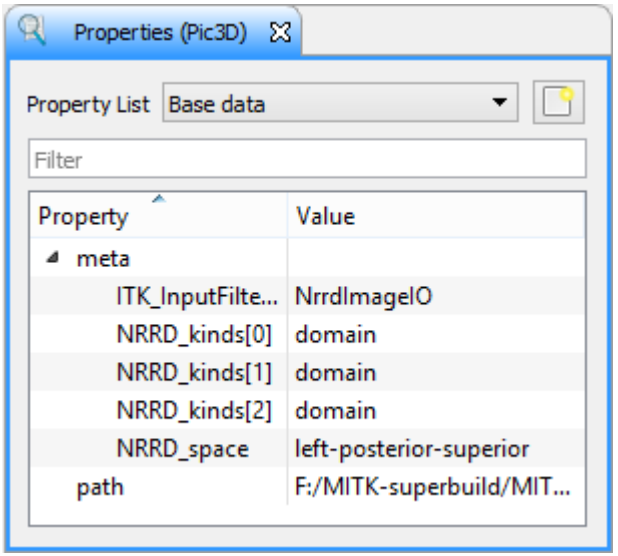
VTK

MITK architecture: Low-level

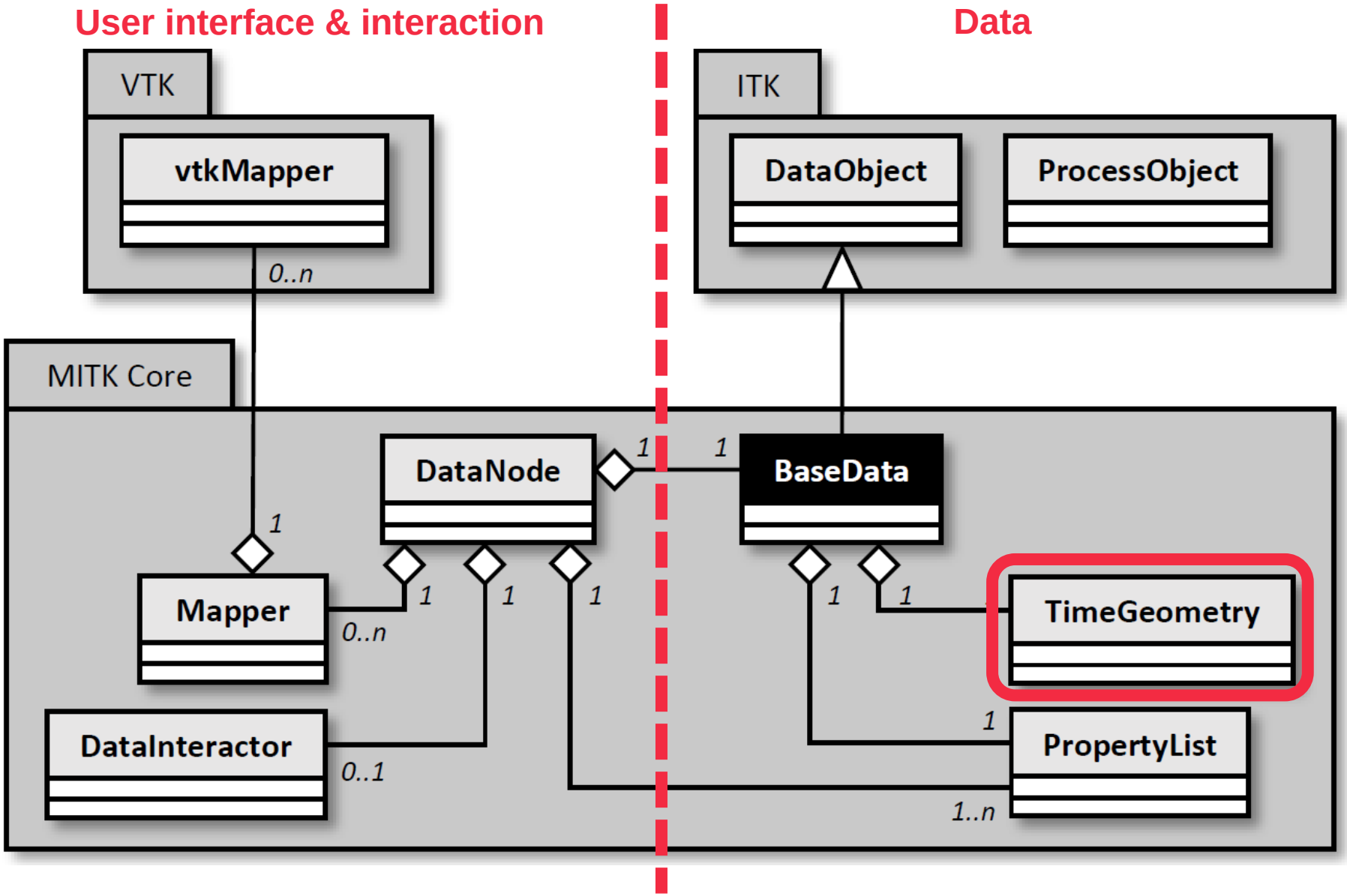


mitk::PropertyList

- Meta-data dictionary
- Properties are key-value pairs
  - Key is a string
  - Value type depends on property type
    - mitk::BoolProperty
    - mitk::StringProperty
    - mitk::EnumProperty
    - mitk::LevelWindowProperty
    - …

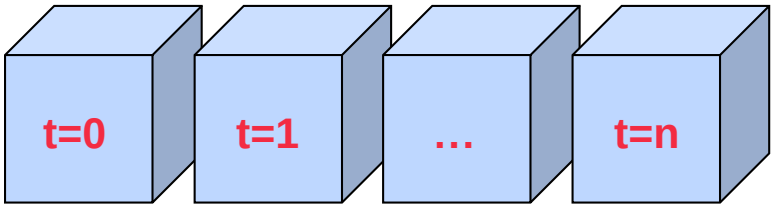


MITK architecture: Low-level



mitk::TimeGeometry

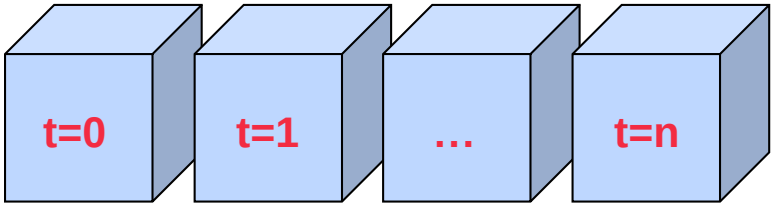
- Encapsulates spatial and temporal information of data



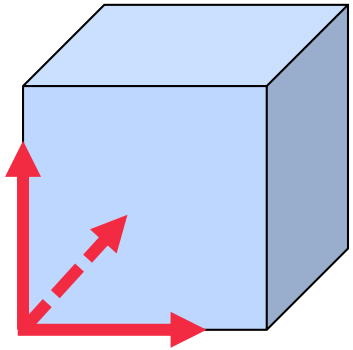
Temporal sequence of 3-d image volumes

mitk::TimeGeometry

- Encapsulates spatial and temporal information of data



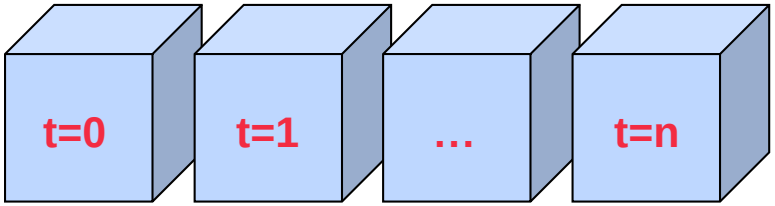
Temporal sequence of 3-d image volumes



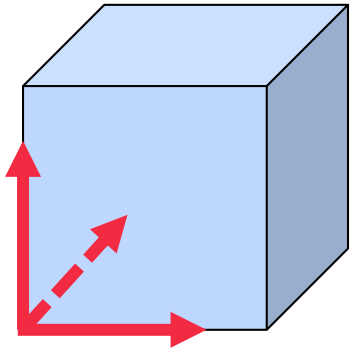
Spatial extent of 3-d image volume

mitk::TimeGeometry

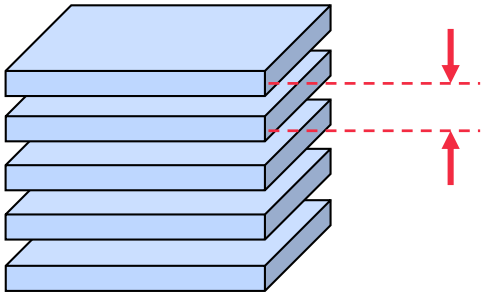
- Encapsulates spatial and temporal information of data



Temporal sequence of 3-d image volumes

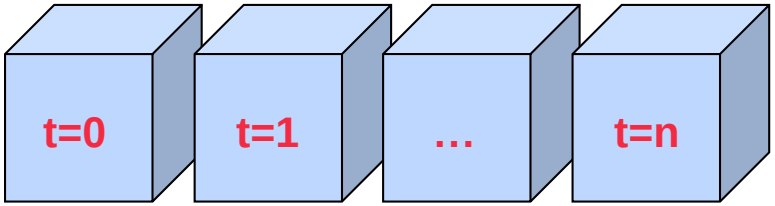


Spatial extent of 3-d image volume

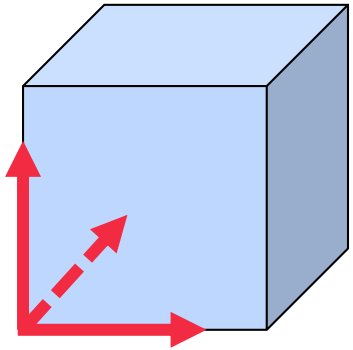


mitk::TimeGeometry

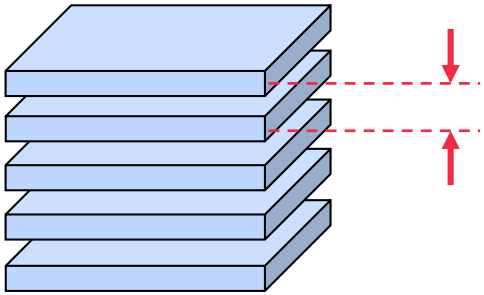
- Encapsulates spatial and temporal information of data



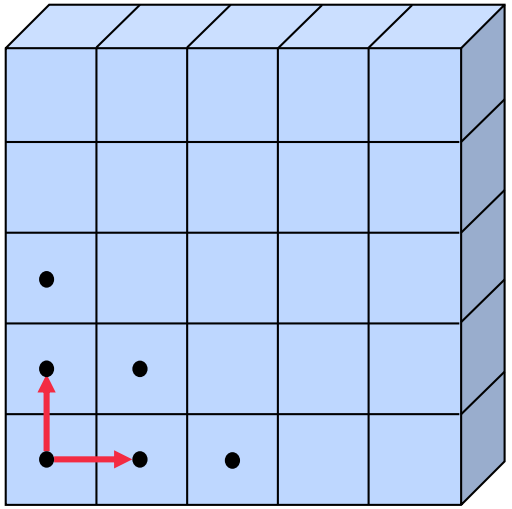
Temporal sequence of 3-d image volumes



Spatial extent of 3-d image volume



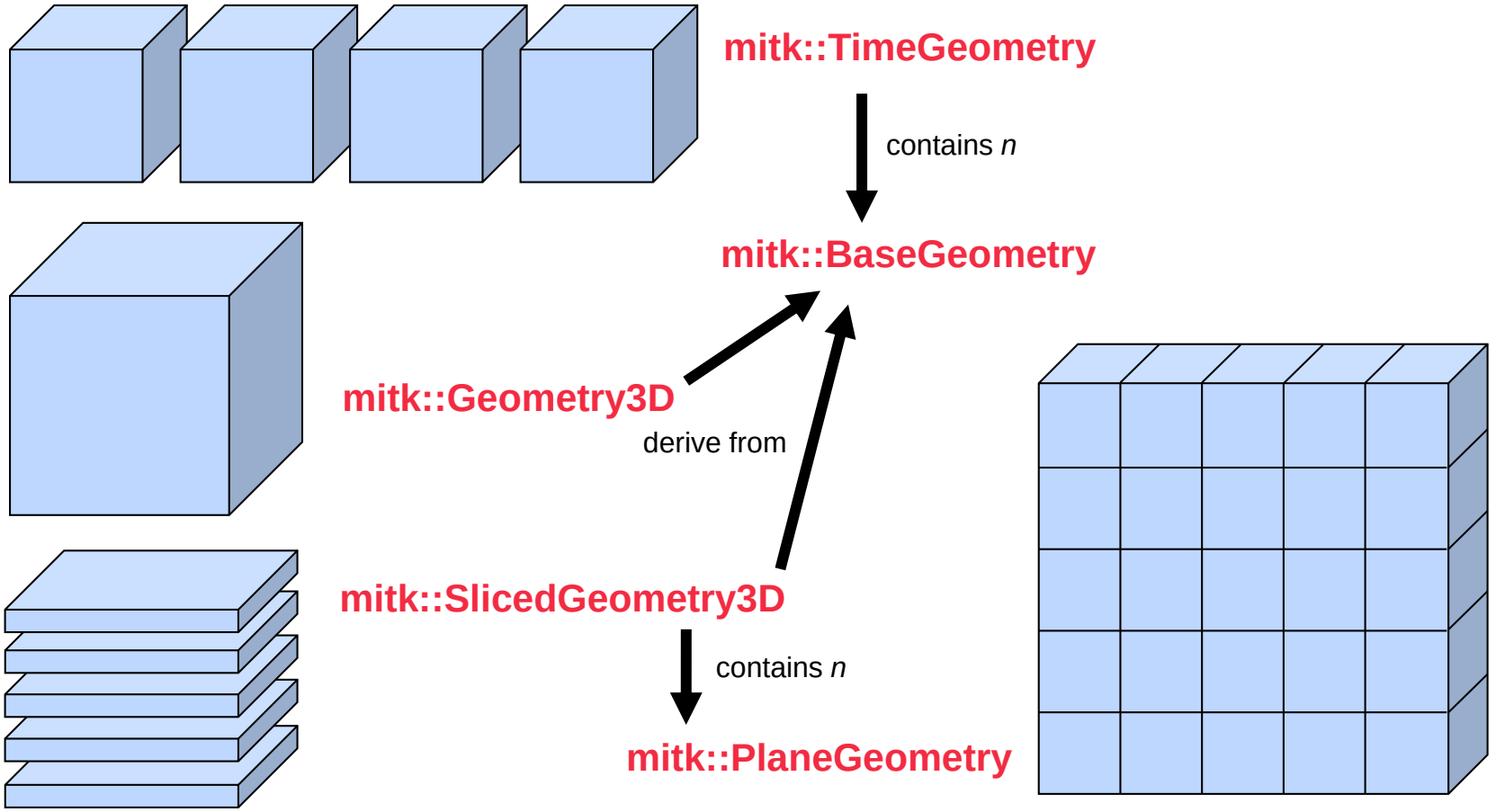
Slice spacing



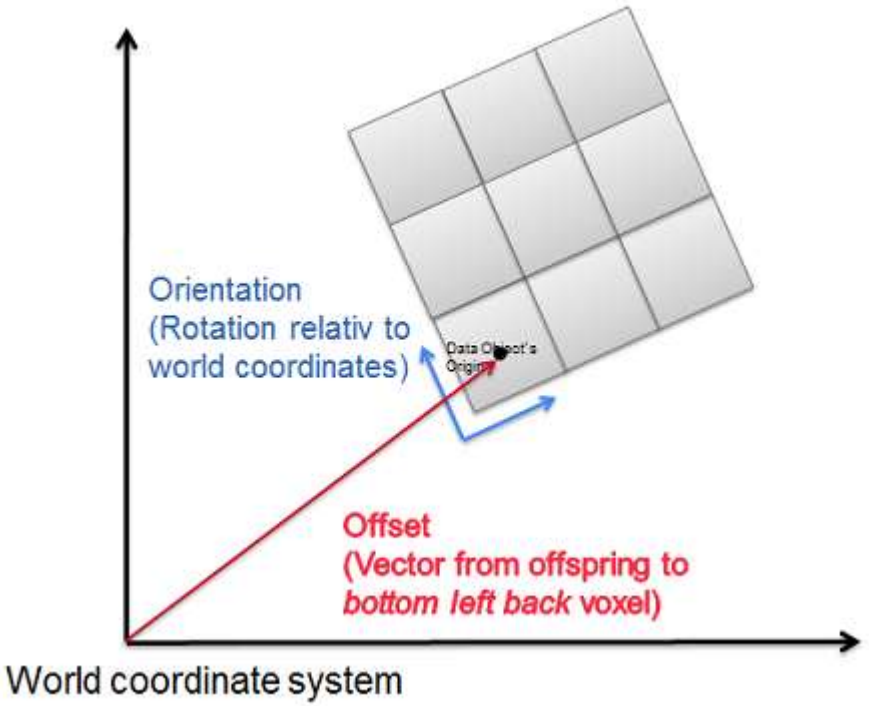
Pixel spacing

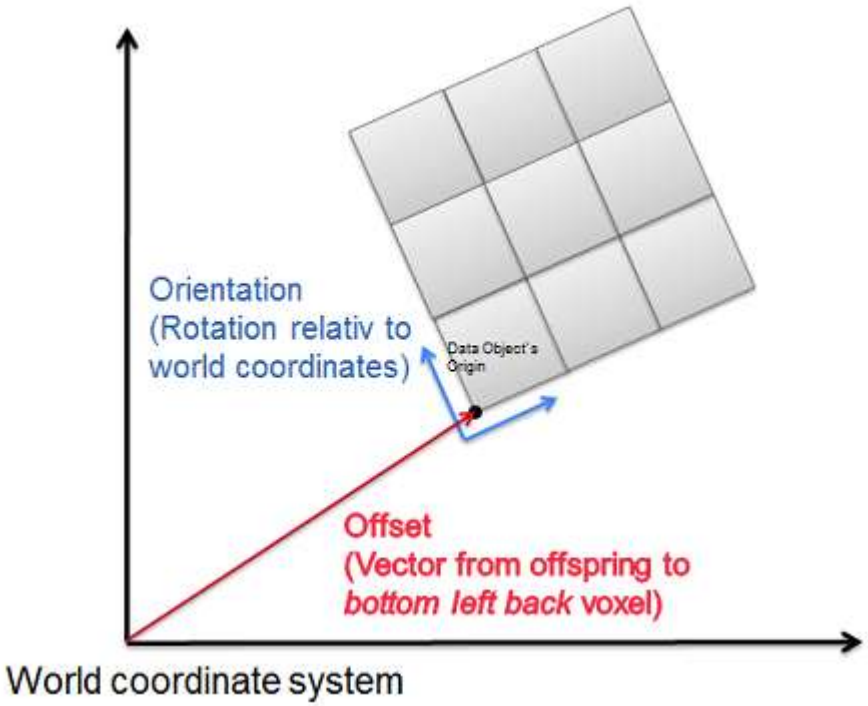
mitk::TimeGeometry

- Encapsulates spacial and temporal information of data



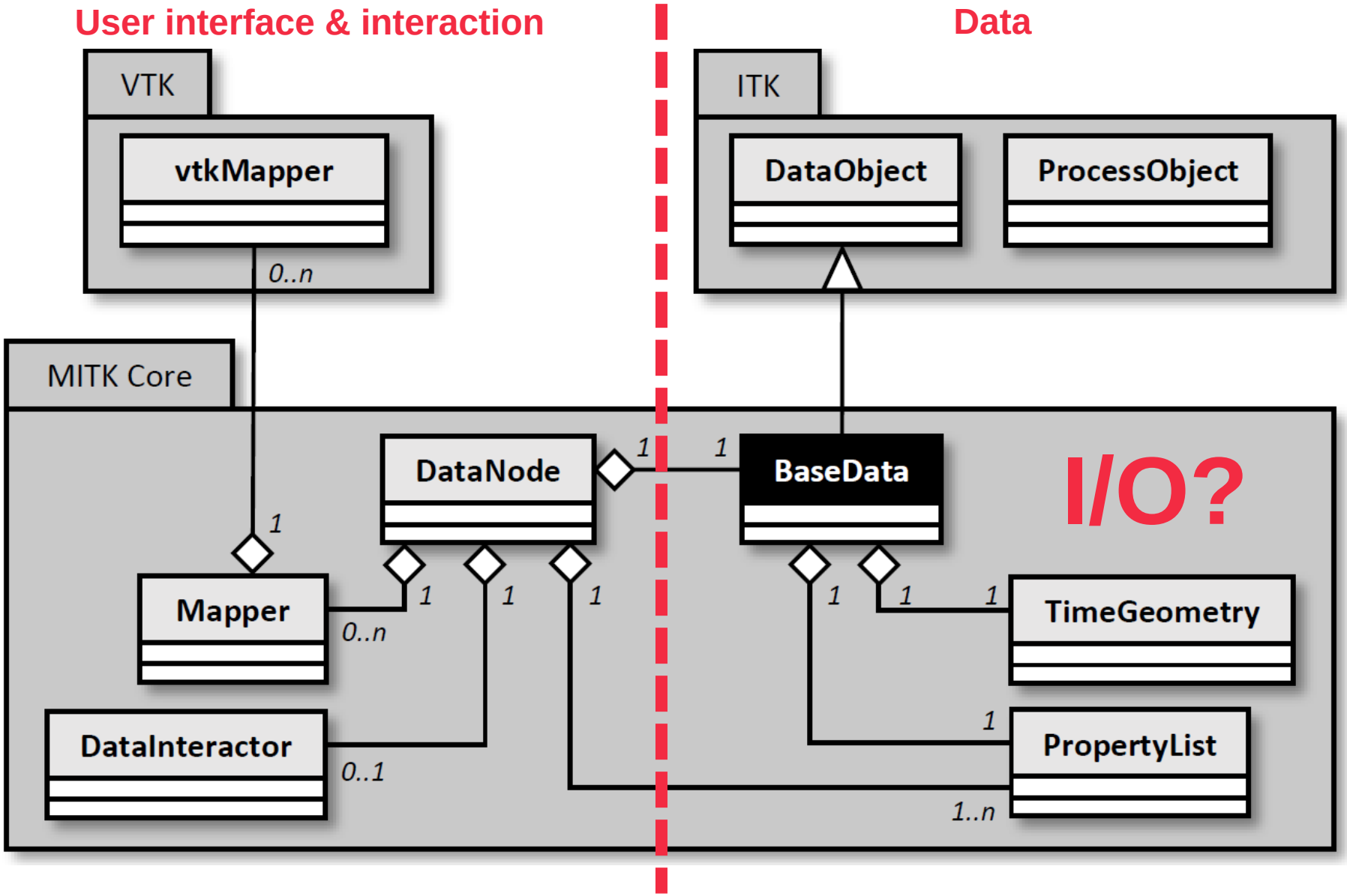
Center-based vs. corner-based coordinates





http://docs.mitk.org/2015.05/GeometryOverviewPage.html

MITK architecture: Low-level

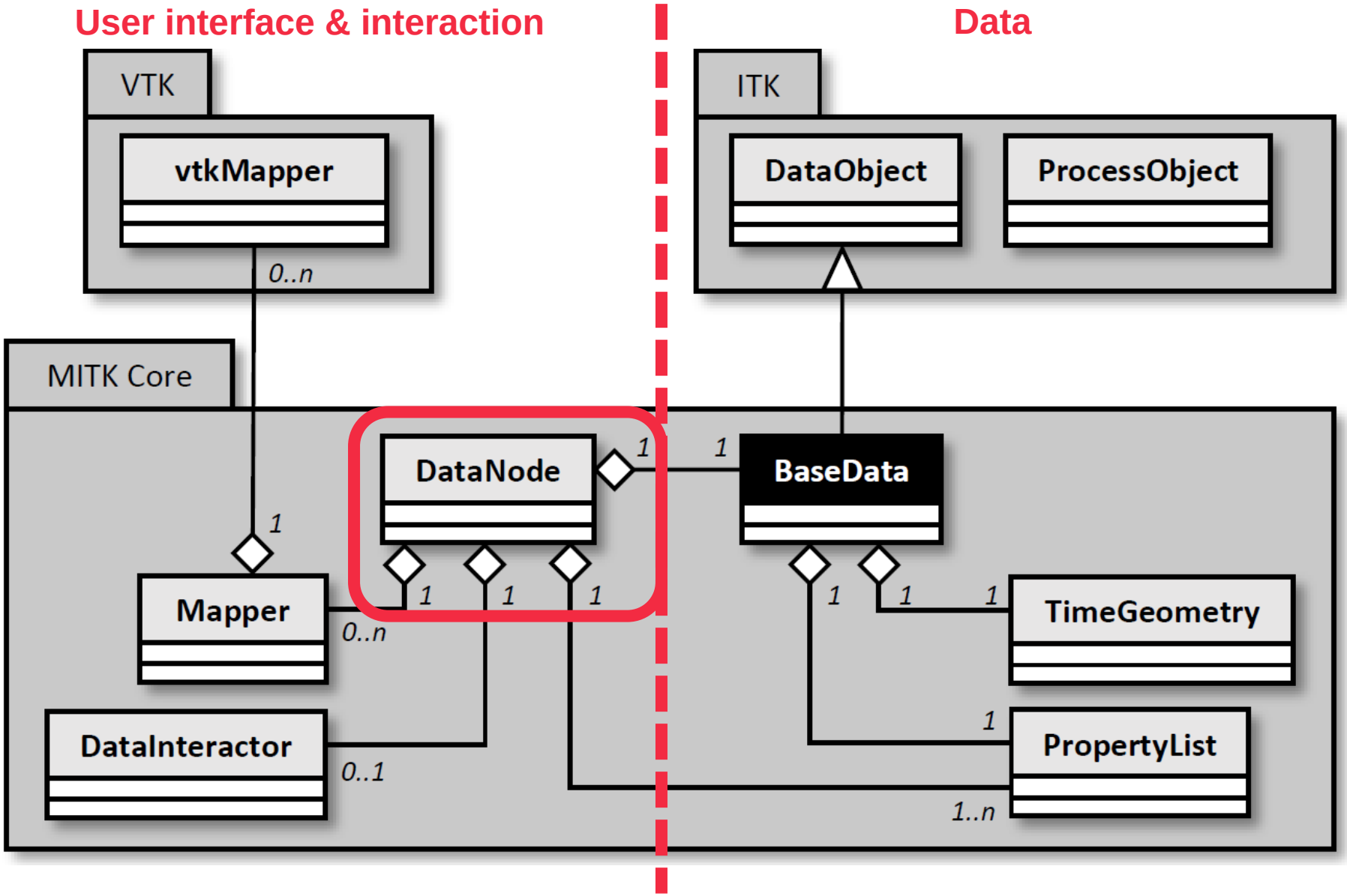


Input/Output

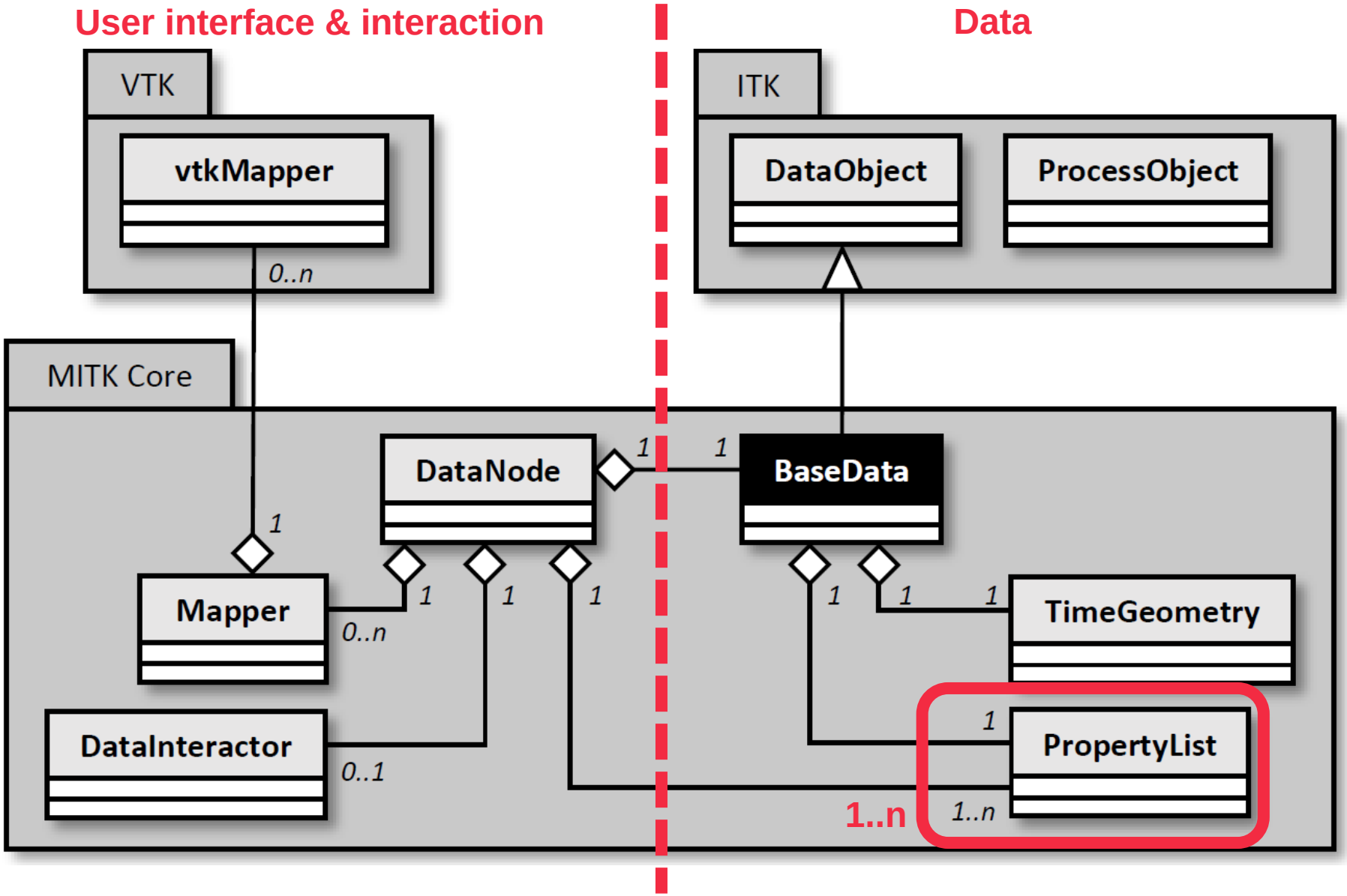

```
#include <mitkIOUtil.h>

try
{
  auto data = mitk::IOUtil::Load("path/to/file.ext");
  // or
  mitk::IOUtil::Save(data, "path/to/file.ext");
}
catch (...)
{
}
```

MITK architecture: Low-level

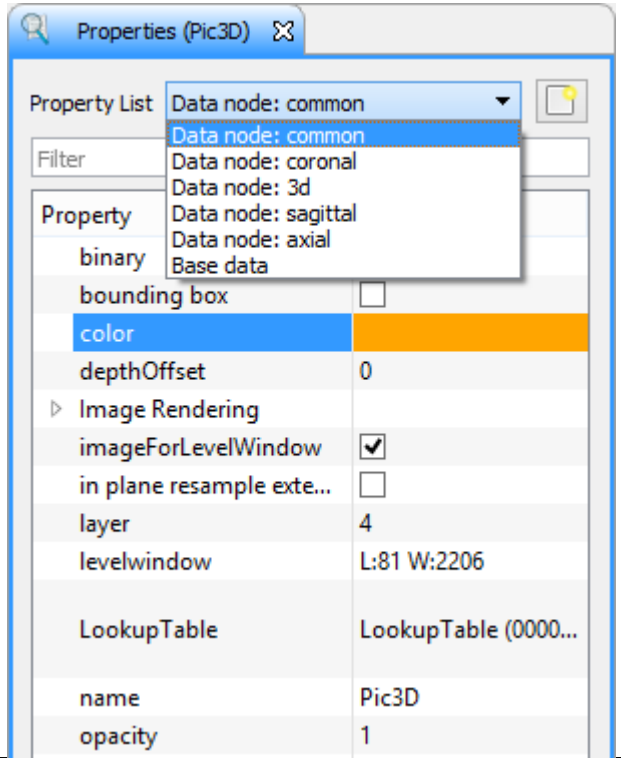


MITK architecture: Low-level

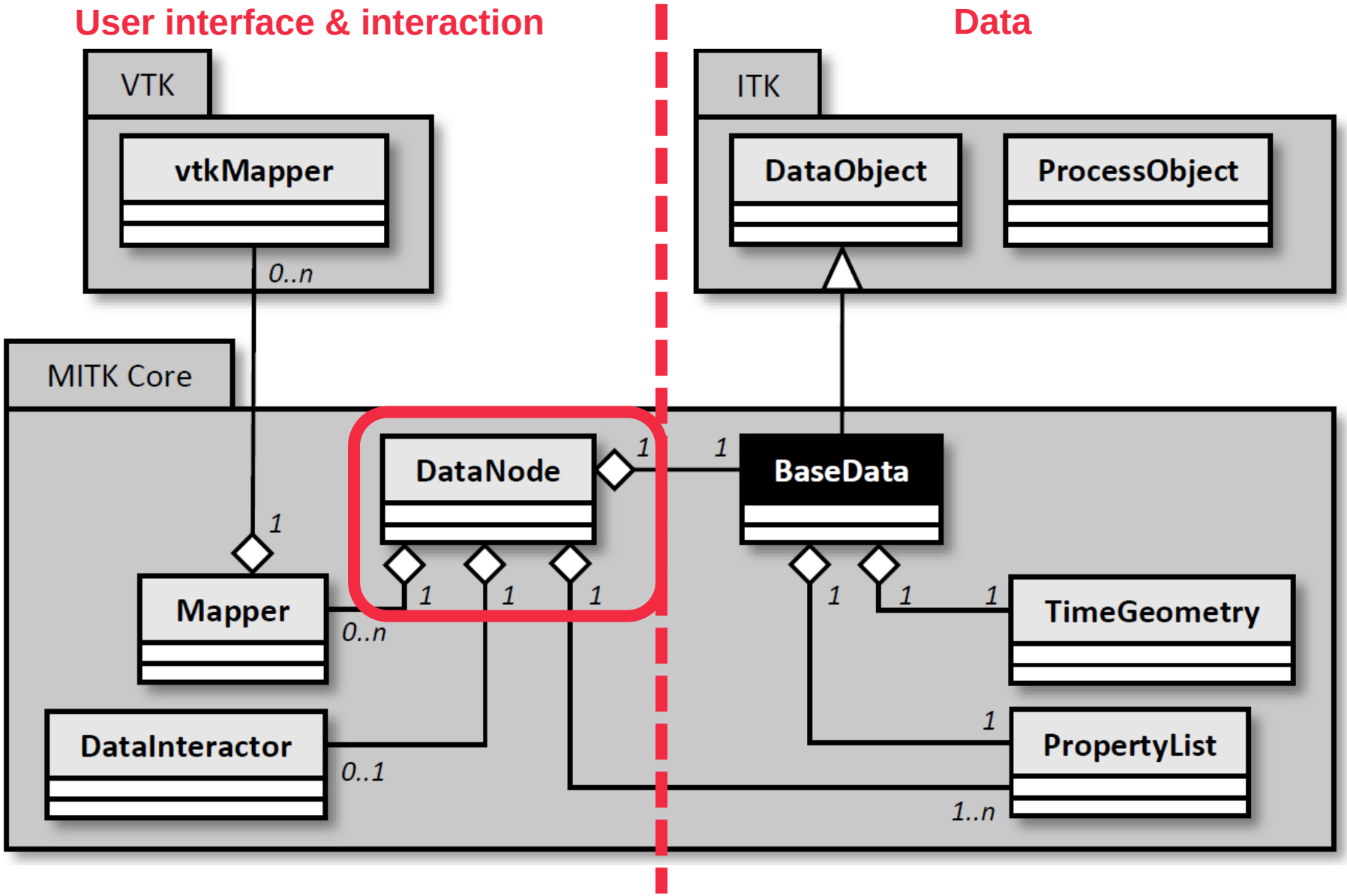


mitk::PropertyList revisited

- mitk::DataNode has a property list, too
  - Contains properties related to rendering / user interface
  - Also has specific property lists for each render window
    - Override common properties

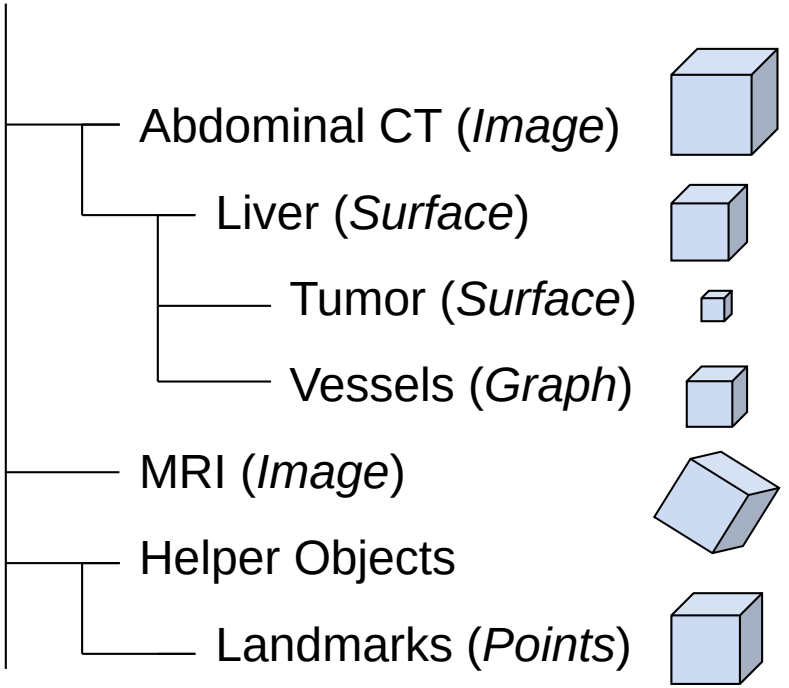


MITK architecture: Low-level

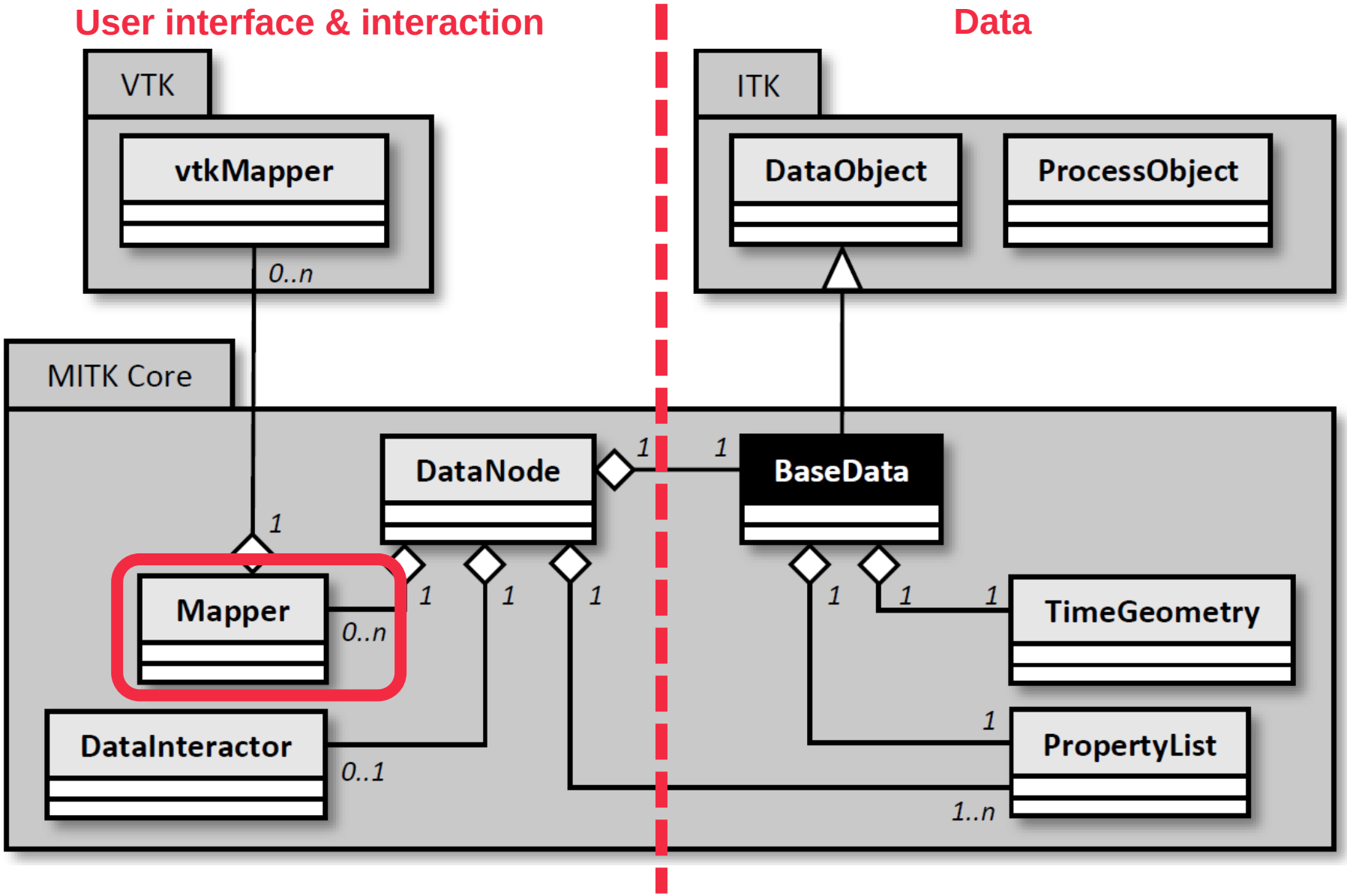


mitk::DataNode *n* — *1* ◆ mitk::DataStorage

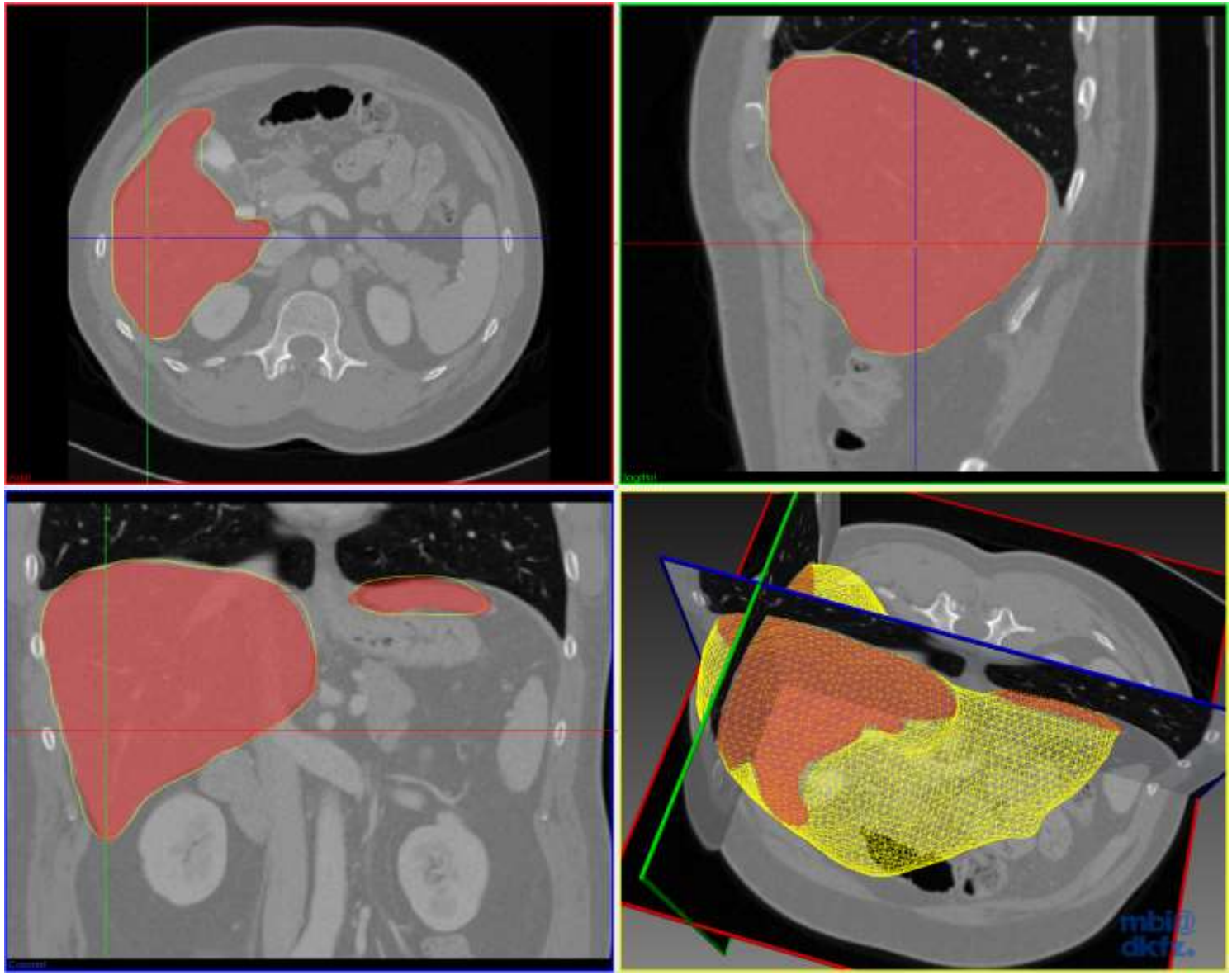
- Share data between modules/plugins
- Any number of data items
- Any kind of data item
- Related to each other through geometry



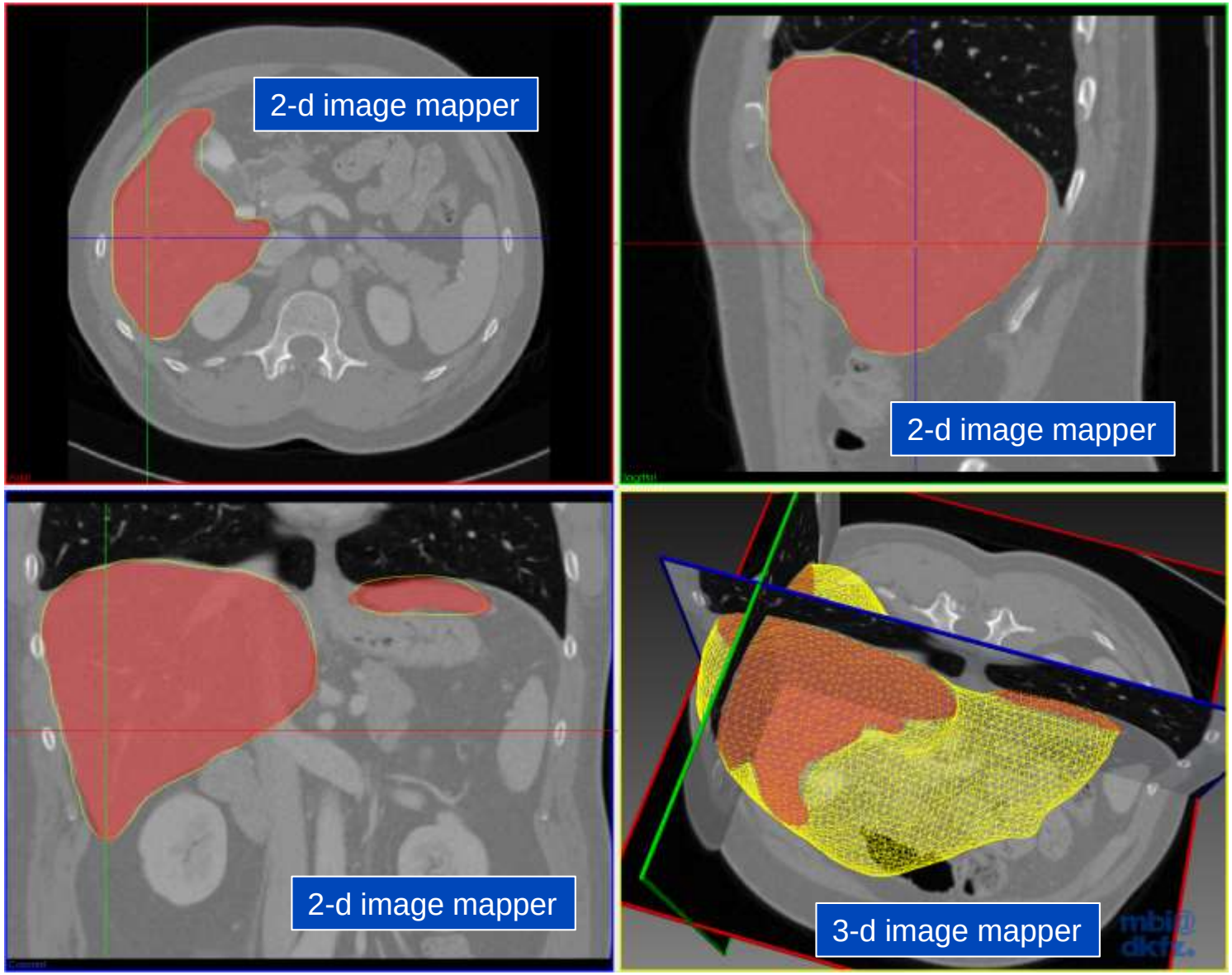
MITK architecture: Low-level



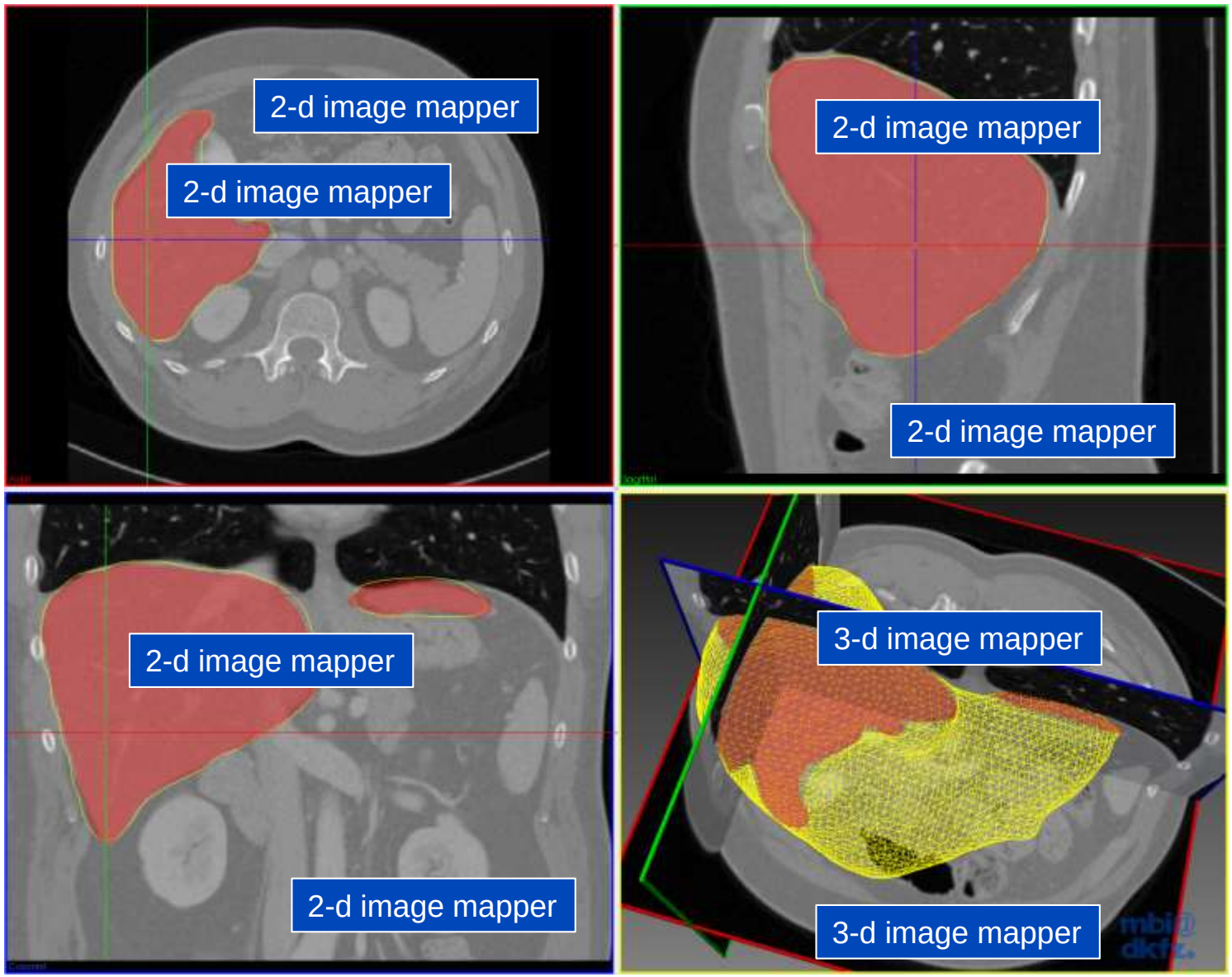
mitk::Mapper



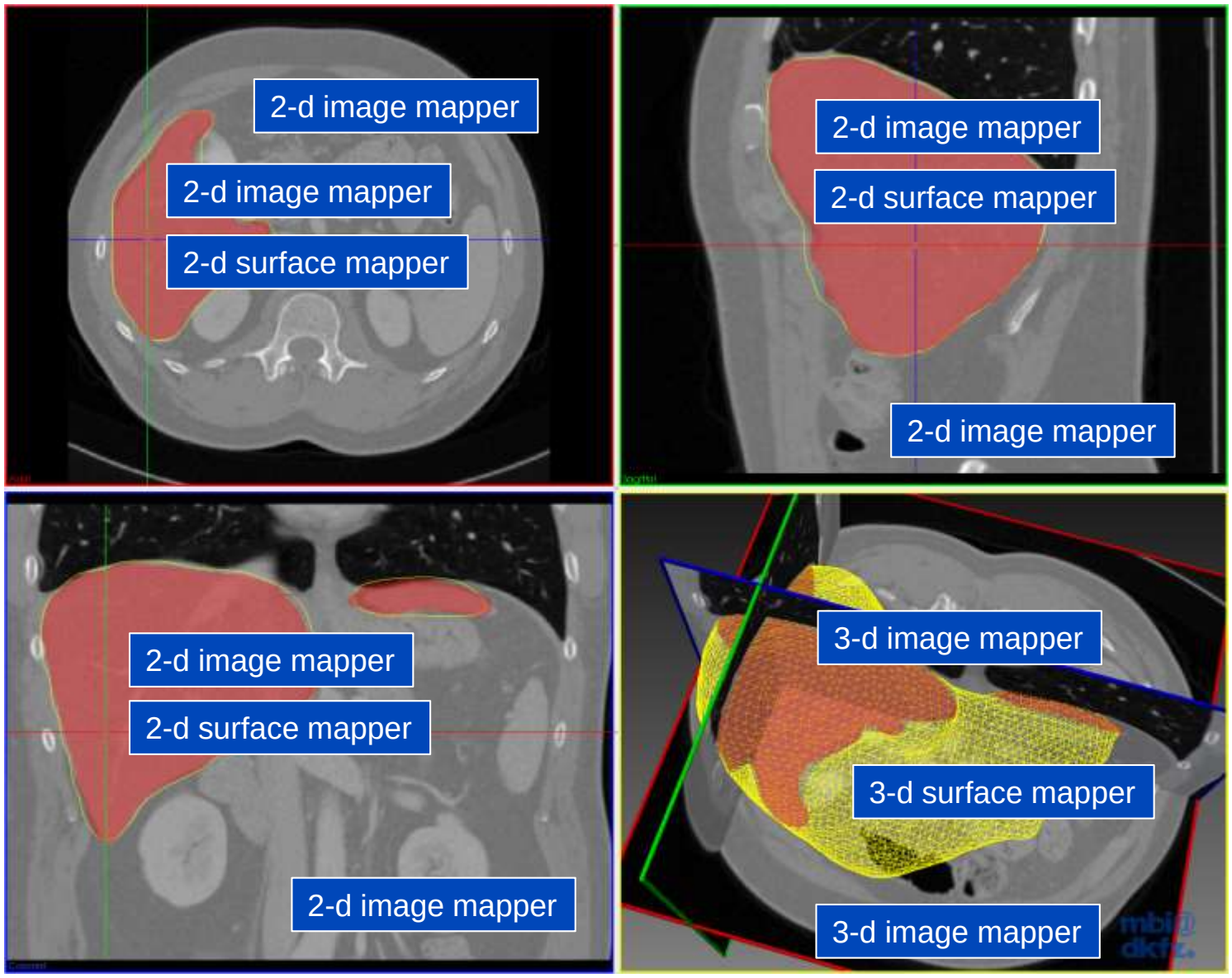
mitk::Mapper



mitk::Mapper



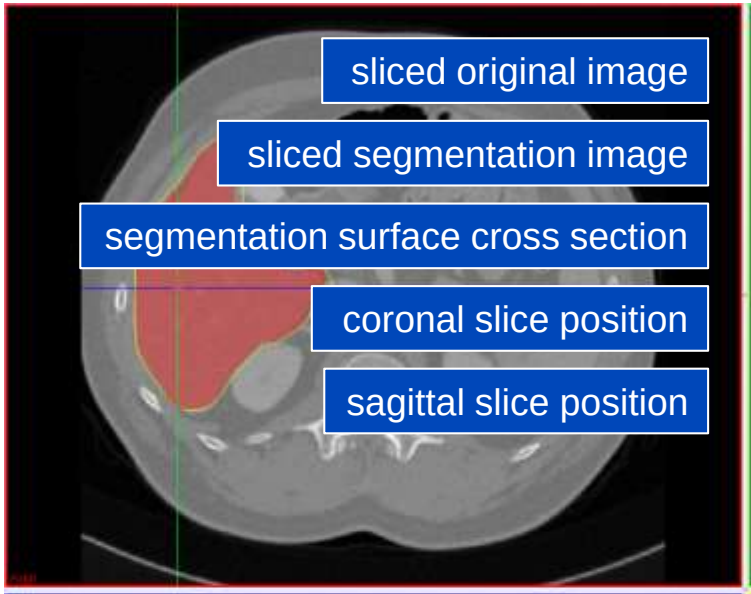
mitk::Mapper

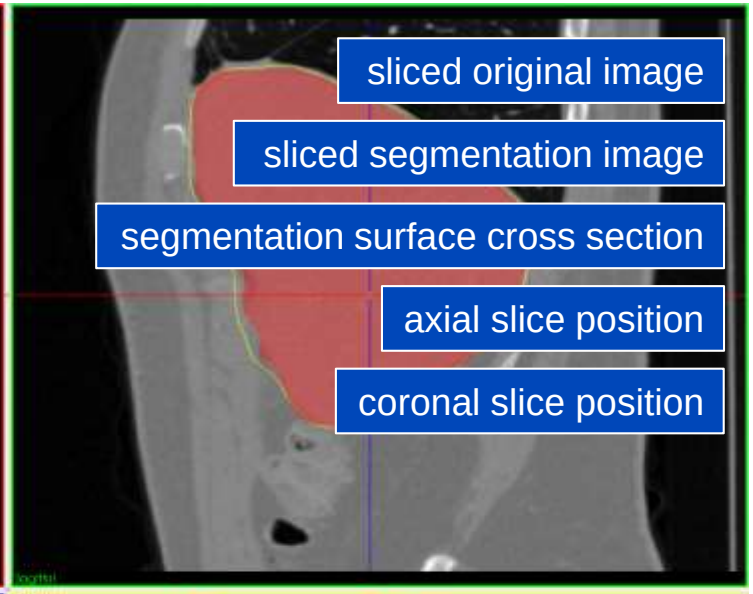


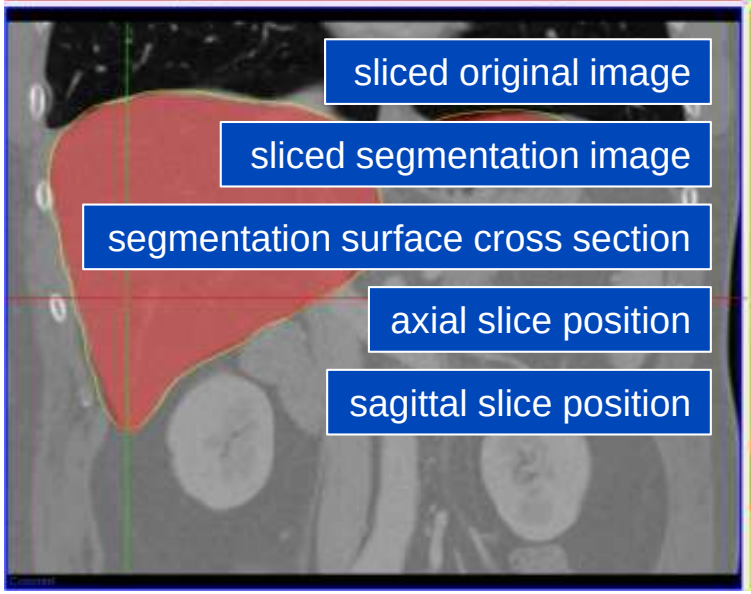
Good news

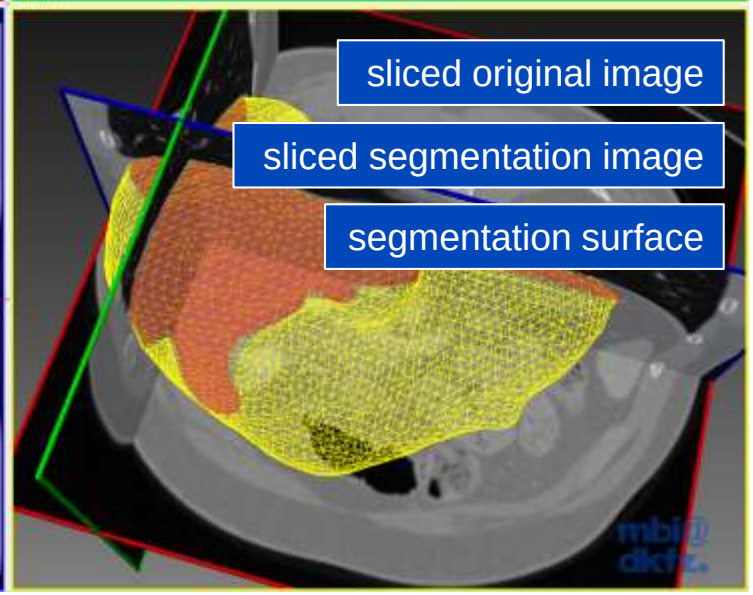
- Mappers are automatically attached to data nodes based on the type of their data

Actually great news





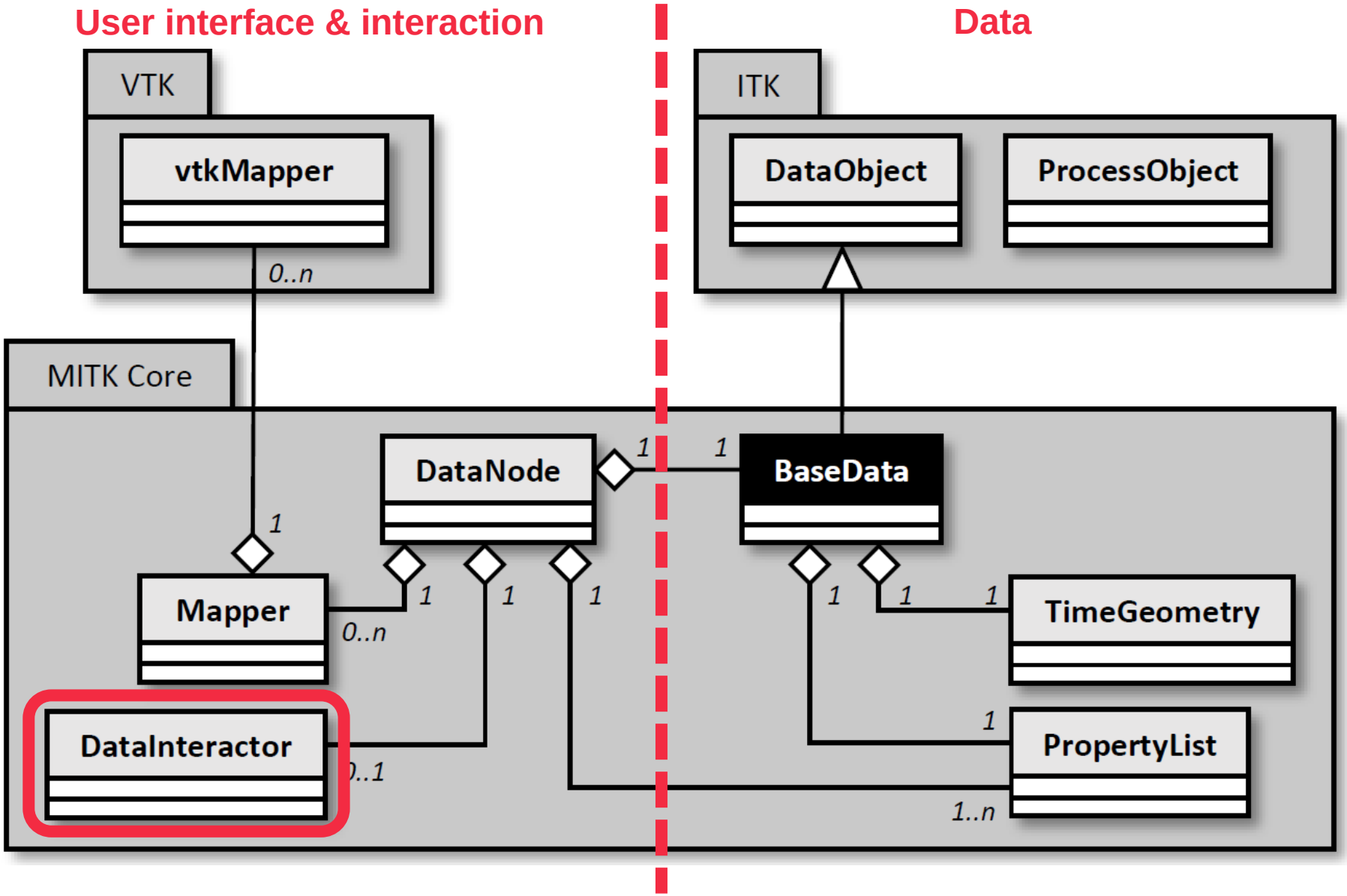




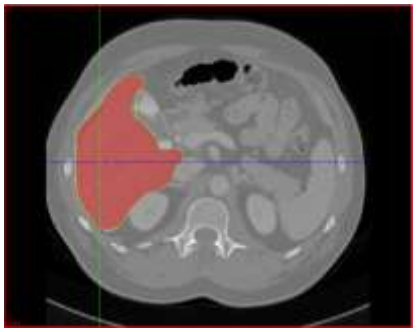
Actually great news

- 18 entities in 4 scene graphs
- Each entity has its own mitk::Mapper
- Everything is generated automatically
- From 3 mitk::DataNodes in mitk::DataStorage
  - Original Image
  - Segmentation Image
  - Segmentation Surface

MITK architecture: Low-level



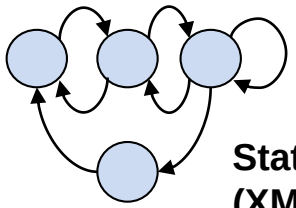
mitk::DataInteractor



Keyboard / mouse events

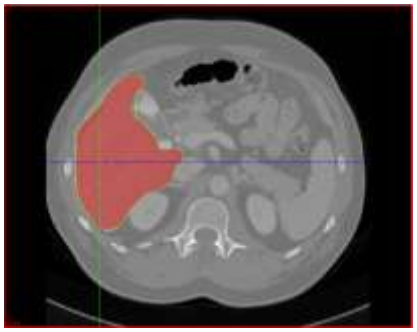
mitk::Dispatcher

mitk::DataInteractor



State machine
(XML)

mitk::DataInteractor – Undo/Redo Support



Keyboard / mouse events

mitk::Dispatcher

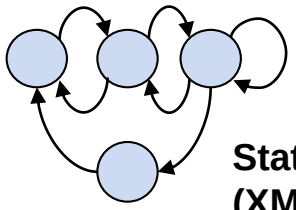
mitk::UndoController

mitk::Operation

inverse

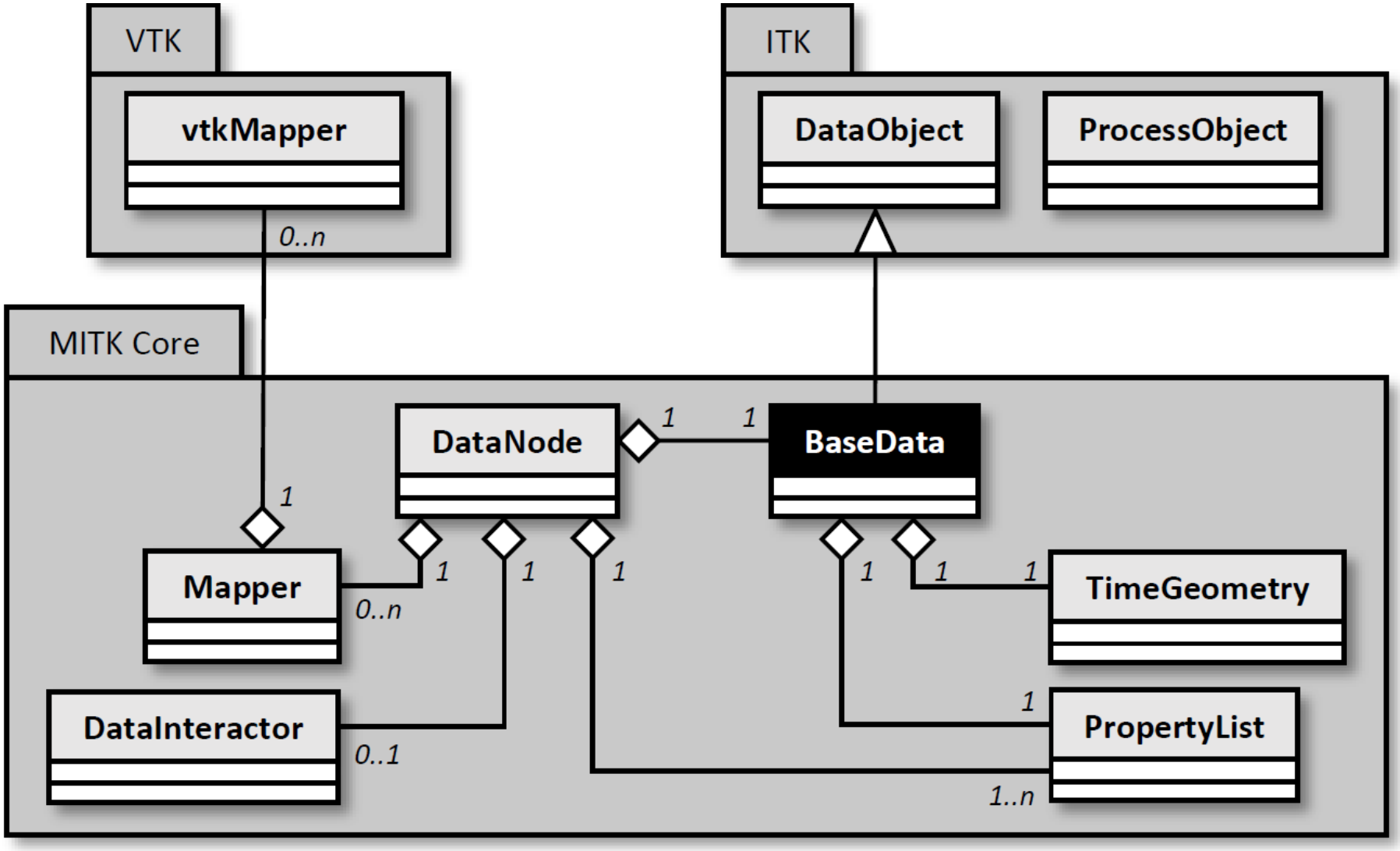
mitk::Operation

mitk::DataInteractor



State machine
(XML)

MITK architecture: Low-level - Summary



Thank you for your attention!

Further information on www.dkfz.de

dkfz. GERMAN CANCER RESEARCH CENTER IN THE HELMHOLTZ ASSOCIATION

50 Years – Research for A Life Without Cancer