VCT tutorial

Introduction to the platform

Outline

1. Purpose
2. Architecture
3. Configuring and running a simulation
4. Utilities
5. Data in VCT: Containers and Components
6. Writing a Module
7. Contributing
8. Installation
9. Practice

1. Purpose

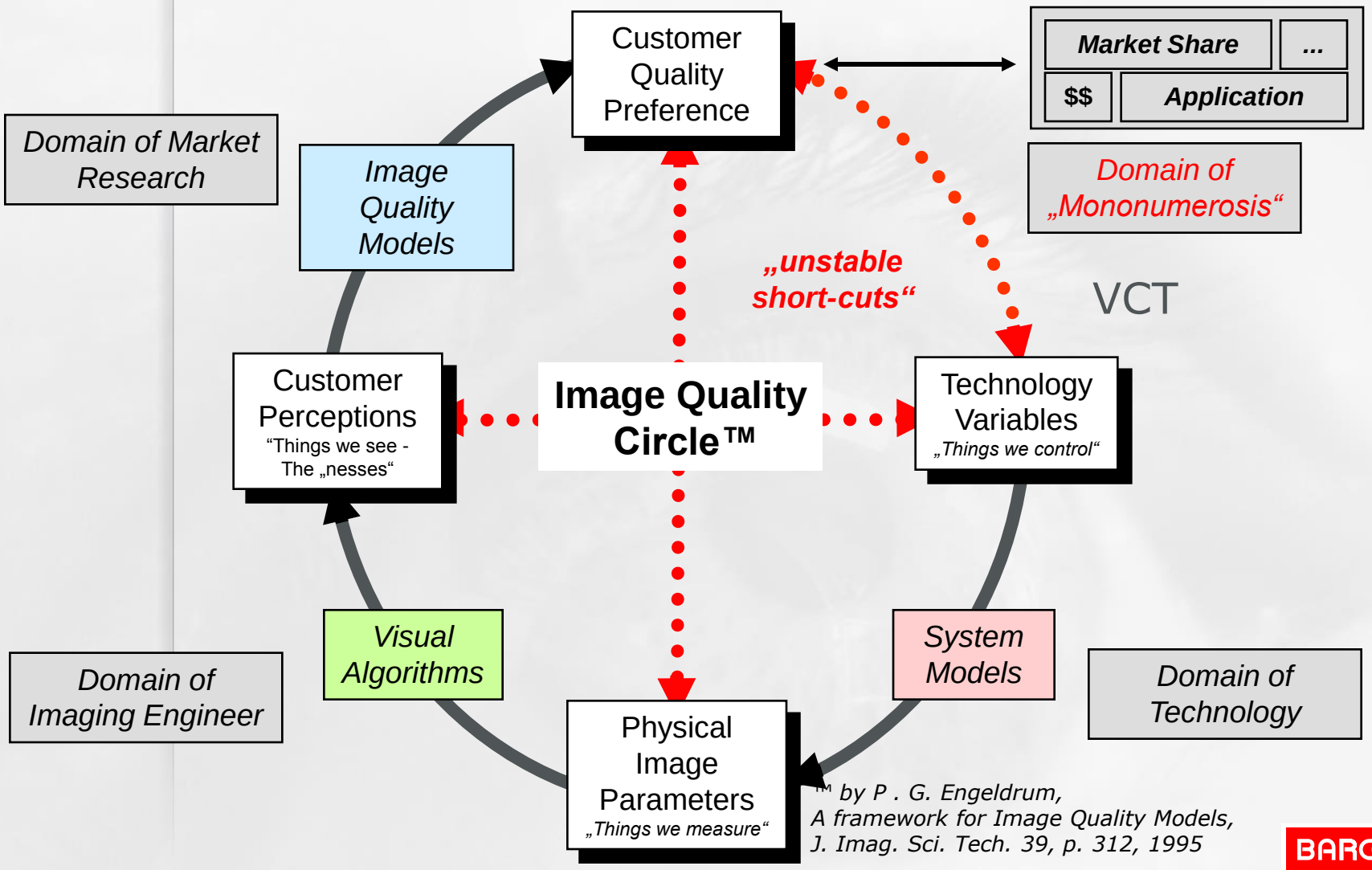
Typical problems

- We need to take design decisions during development of medical systems:
  - **Which panel? (compromise: viewing angle, noise, …)**
  - **Which backlight?**
  - **How bright?**
  - **Noise reduction or not?**
  - **Can we apply lossy compression?**

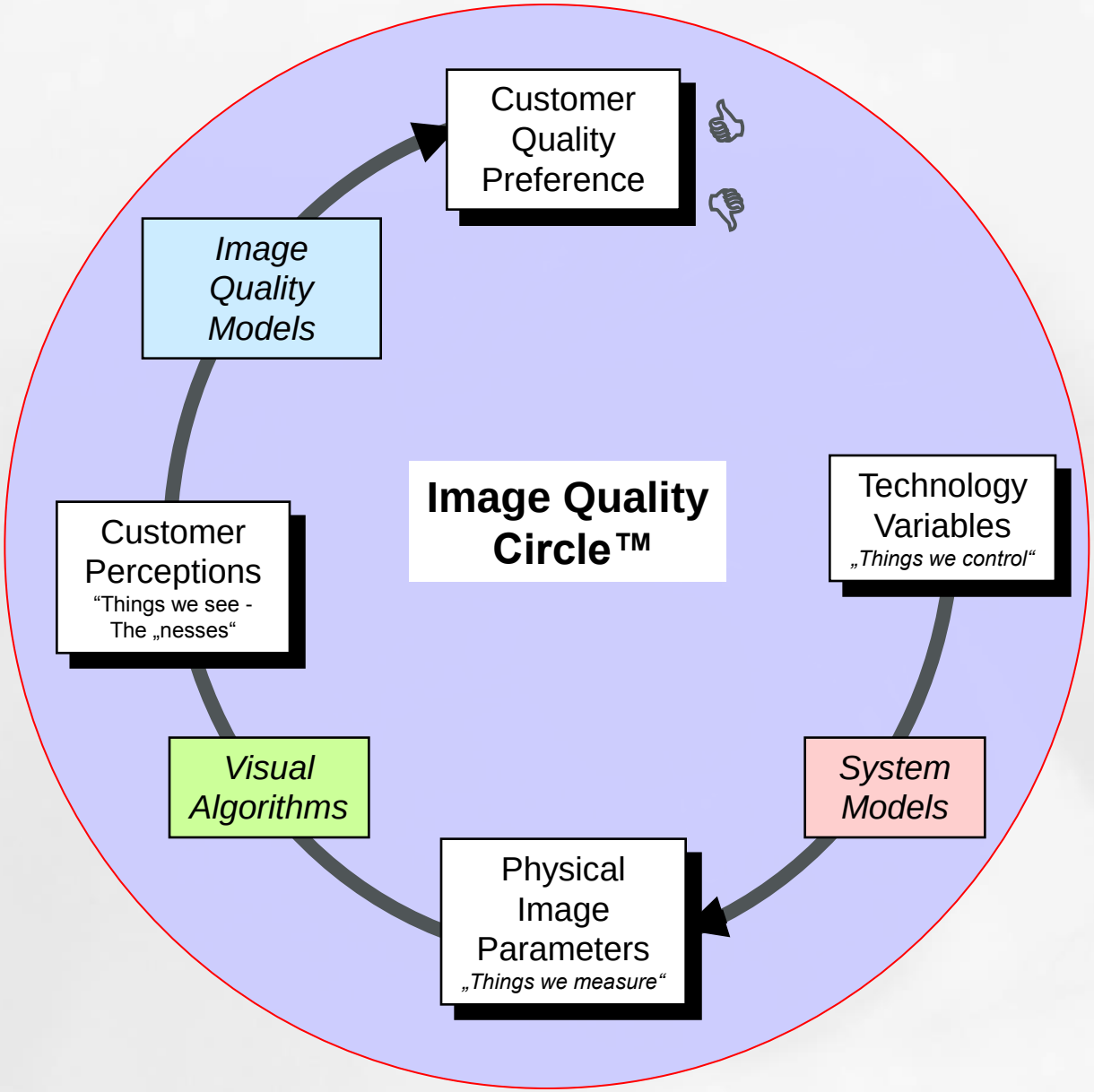
- Today: we build a prototype, evaluate it internally, show it to a few clinical people and then decide if it is a good configuration.
- To prove clinical quality of the display we need to perform costly time-consuming psychovisual/clinical tests.

- Today: first decisions are mostly made based on physical measurements and not based on clinical quality (perception)

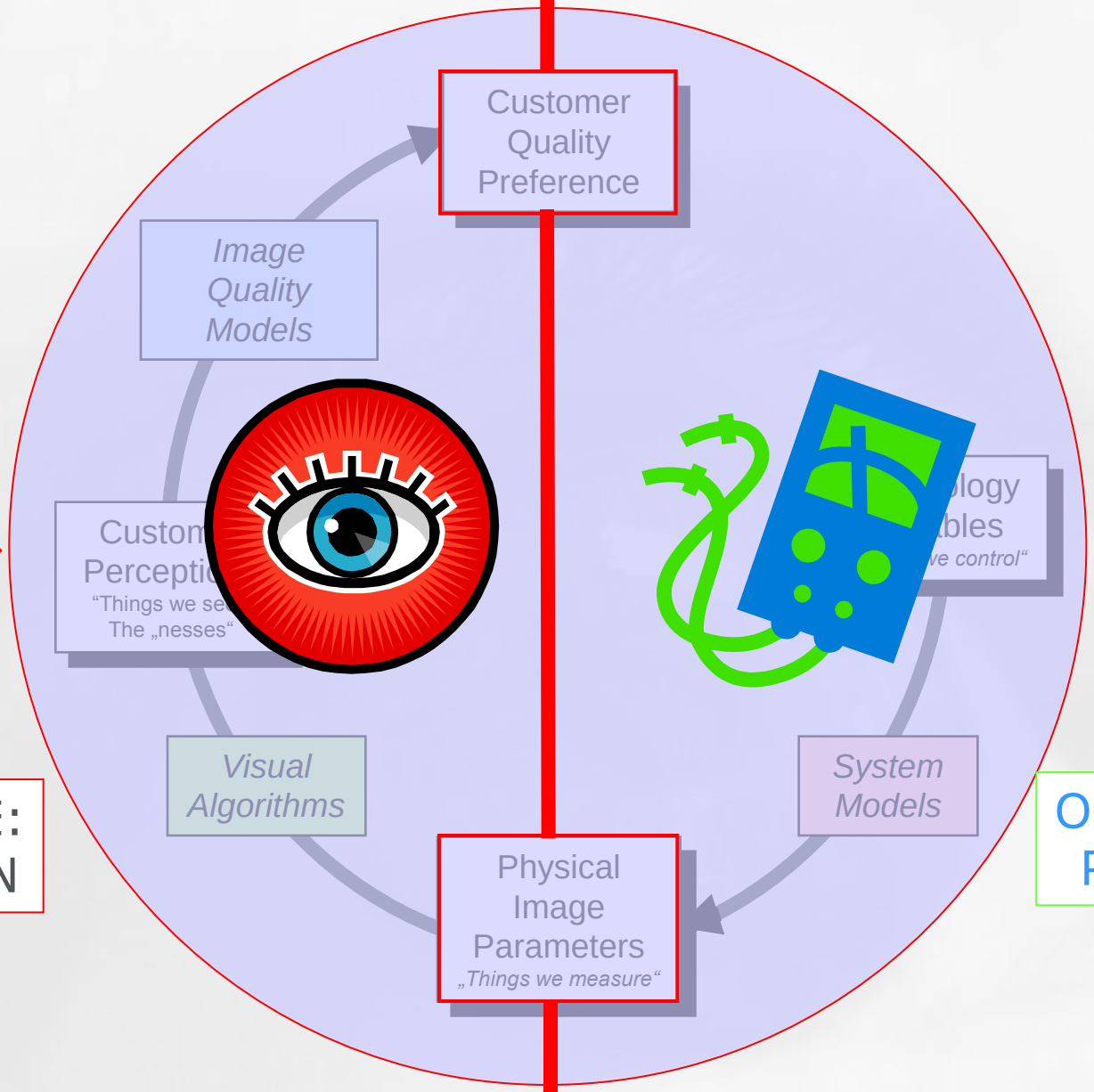
VCT from Image Quality Circle



™ by P . G. Engeldrum,
A framework for Image Quality Models,
J. Imag. Sci. Tech. 39, p. 312, 1995



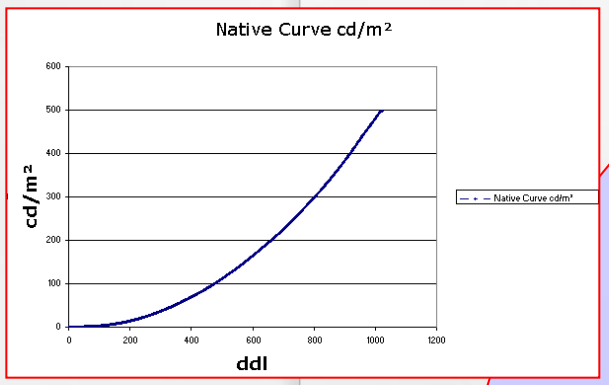
1. Purpose



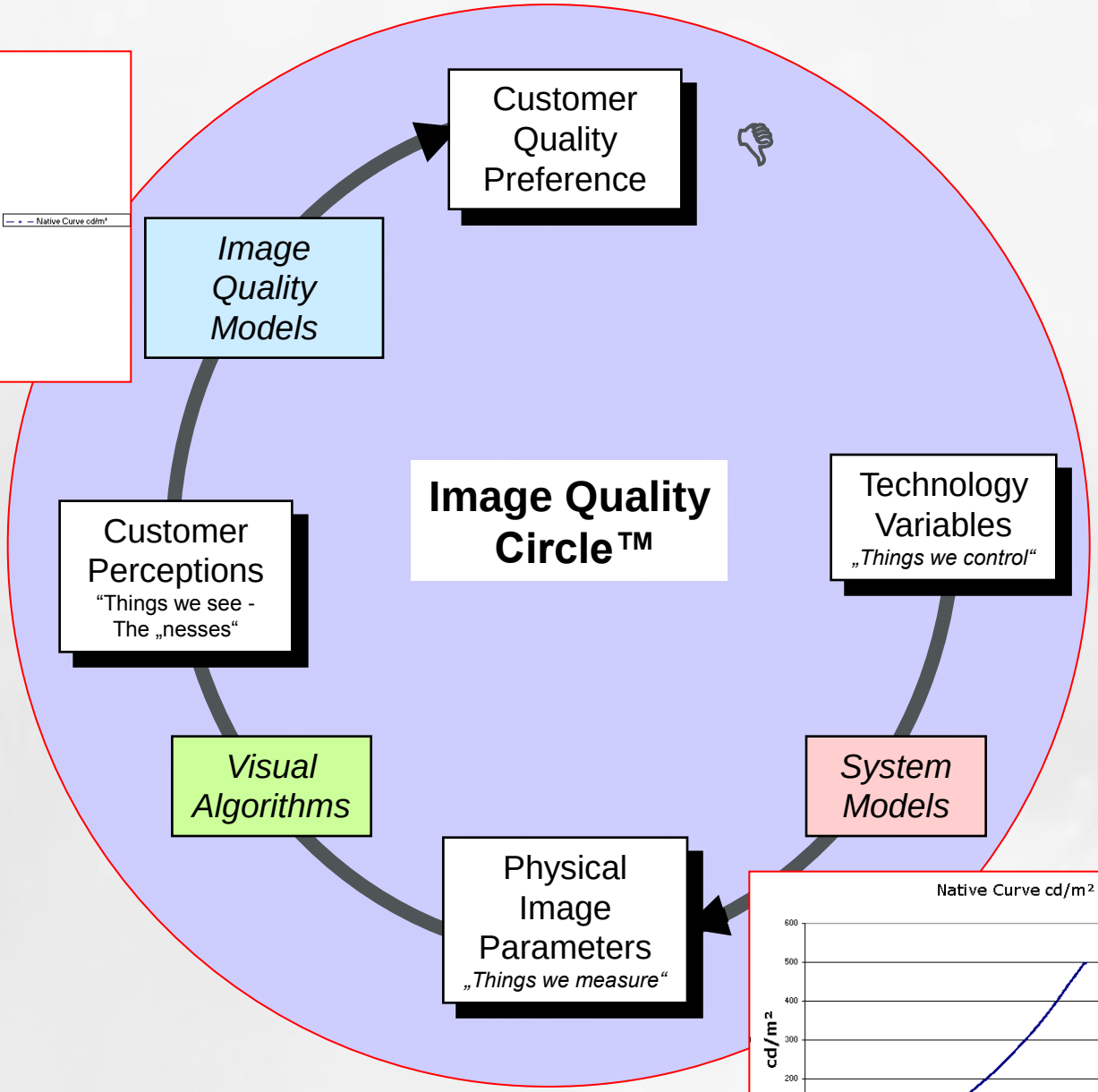
SUBJECTIVE: PERCEPTION

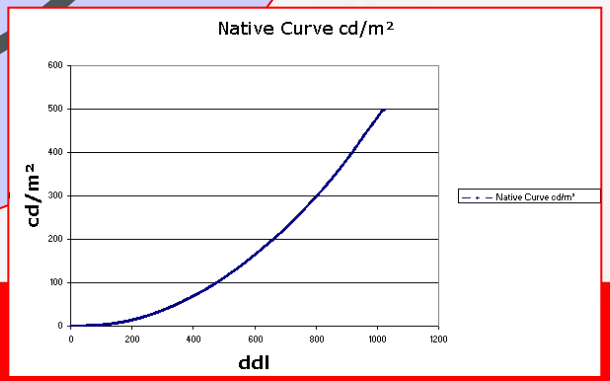
OBJECTIVE: PHYSICAL

Example of the Image Quality Circle with GSDF*-DICOM



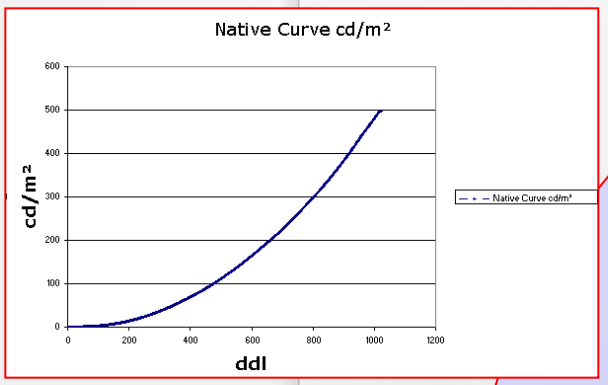
Perception not taken into account

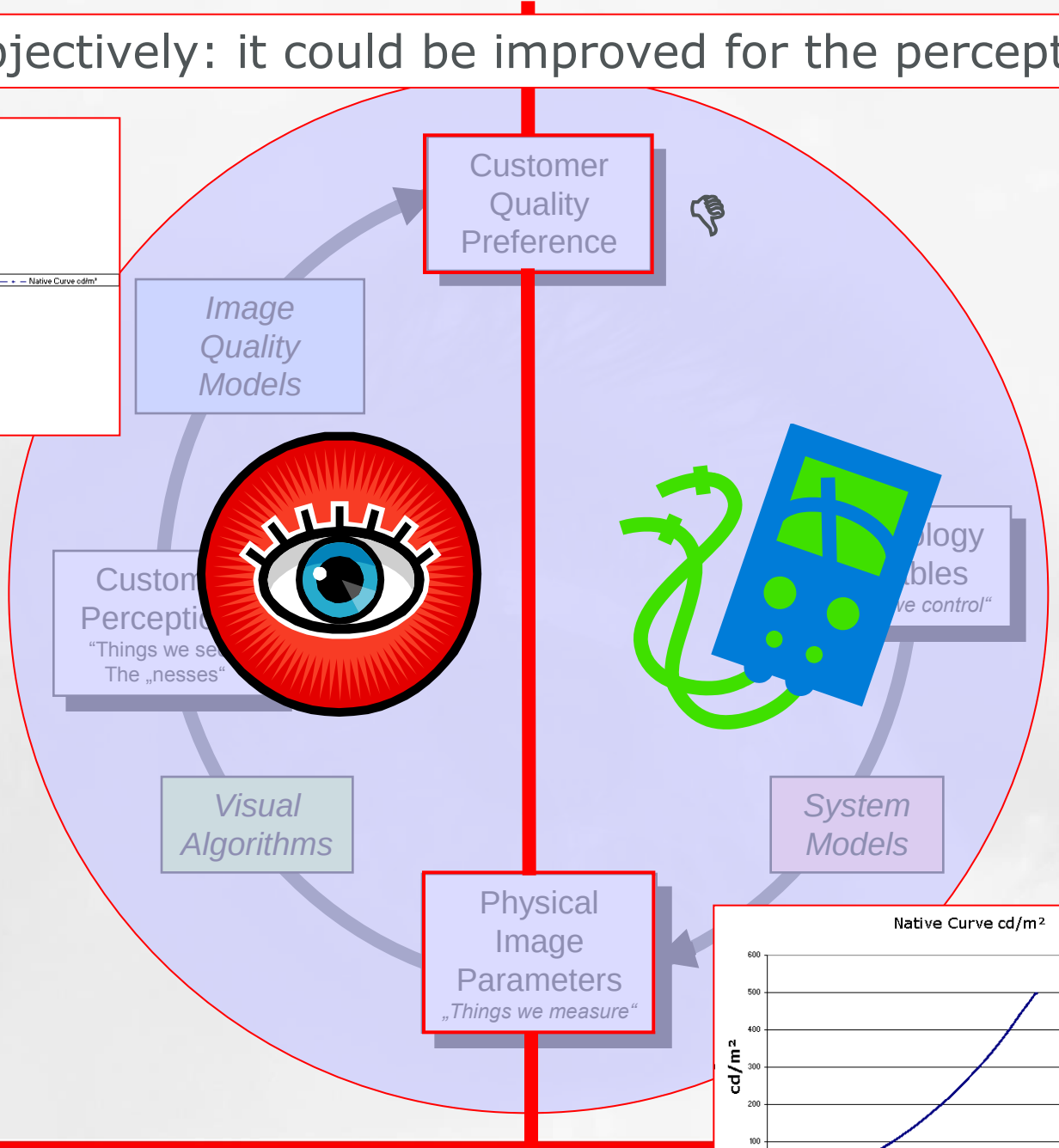




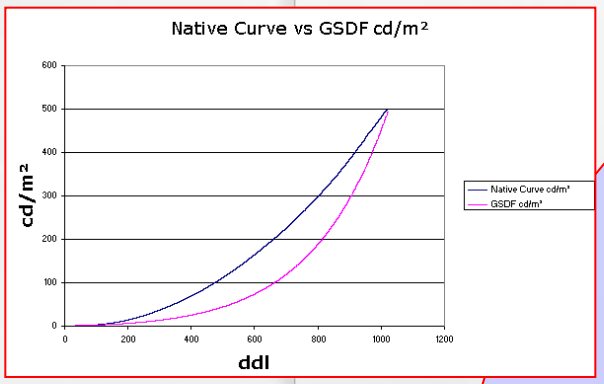
*GSDF: Grayscale Display Function

Subjectively: it could be improved for the perception

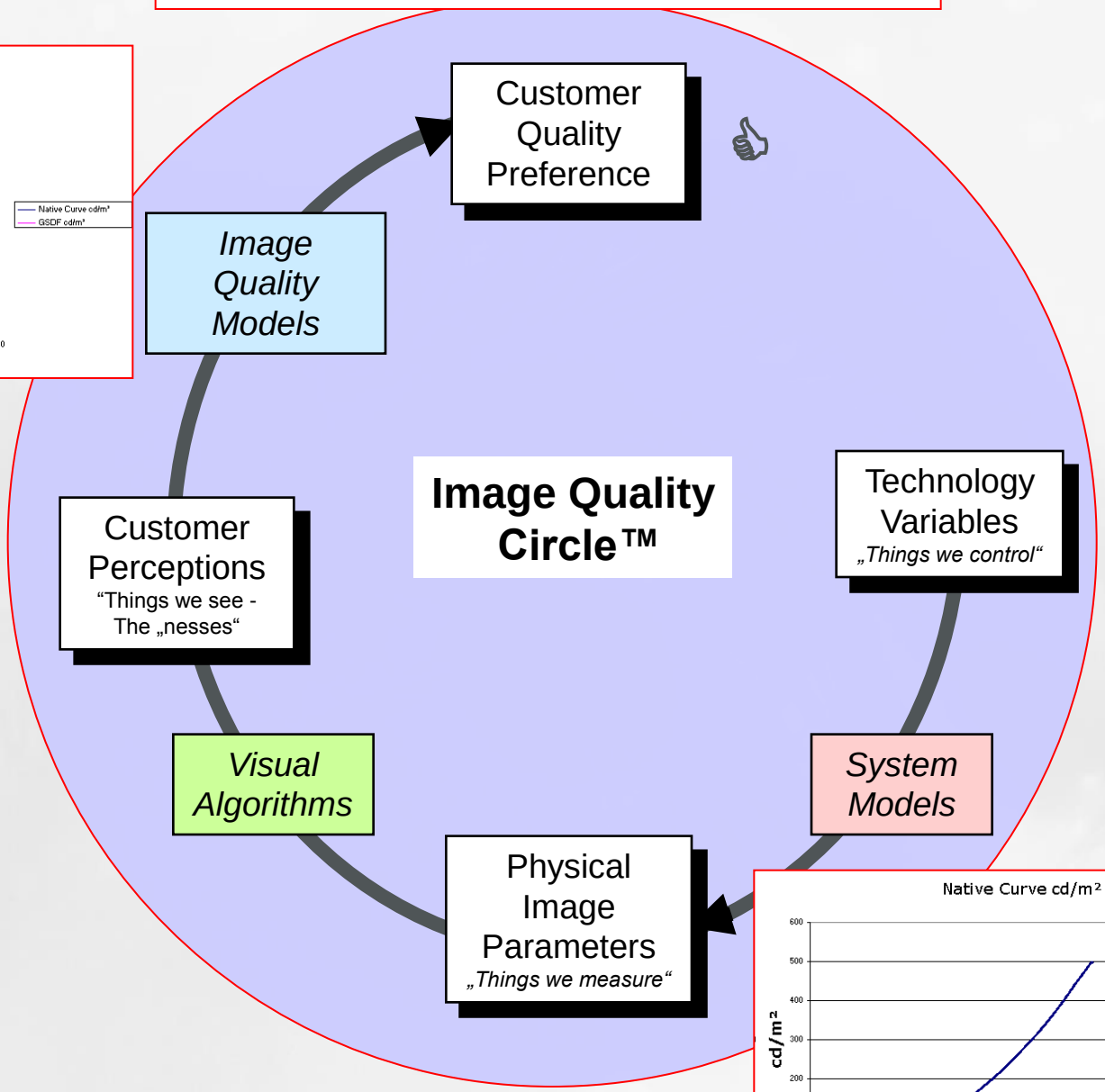


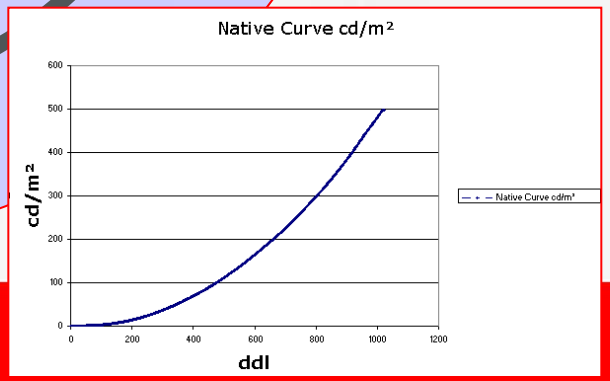


Subjectively: better perception

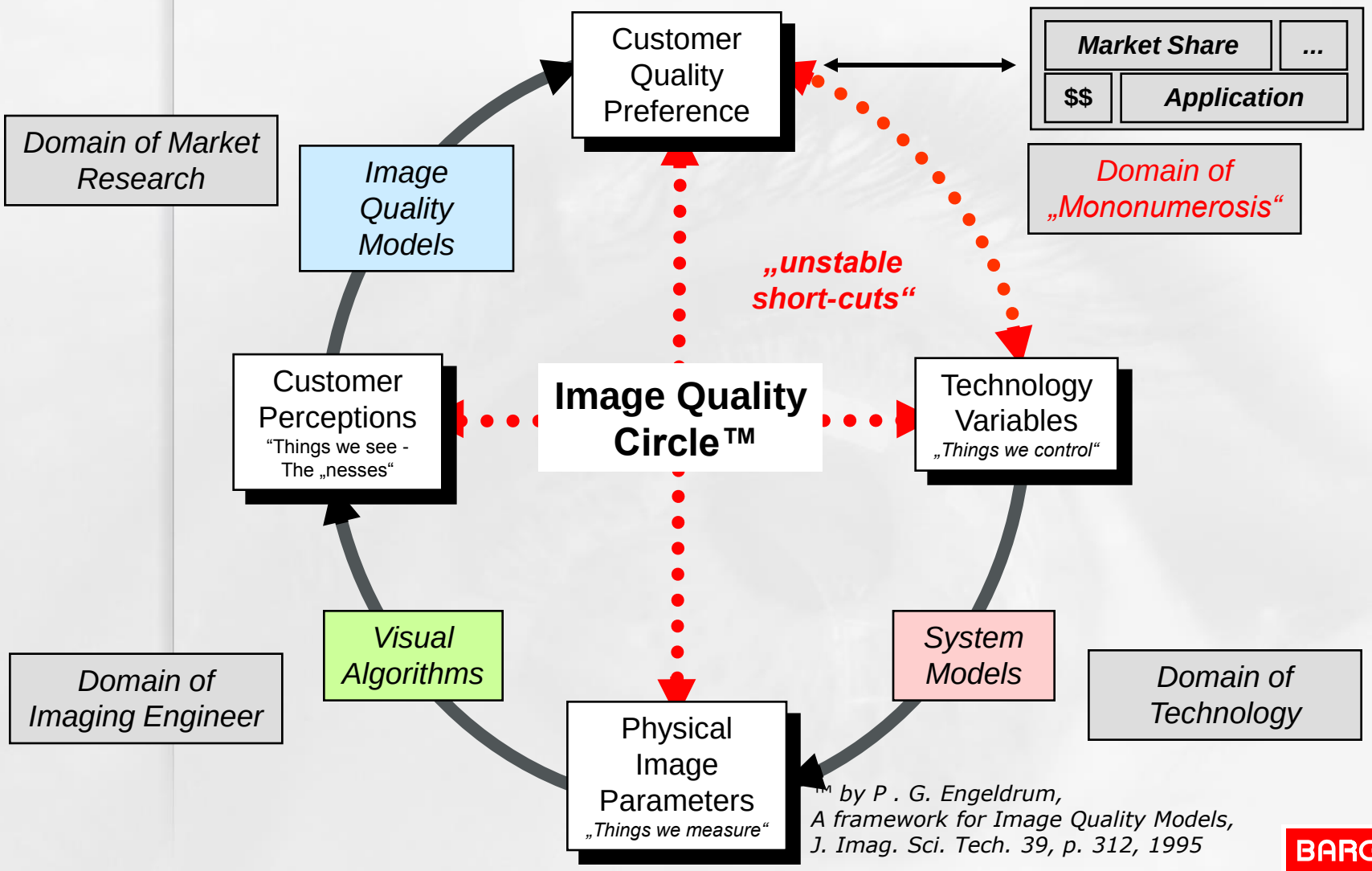


Perception taken into account & improved





VCT from Image Quality Circle



™ by P . G. Engeldrum,
A framework for Image Quality Models,
J. Imag. Sci. Tech. 39, p. 312, 1995

VCT

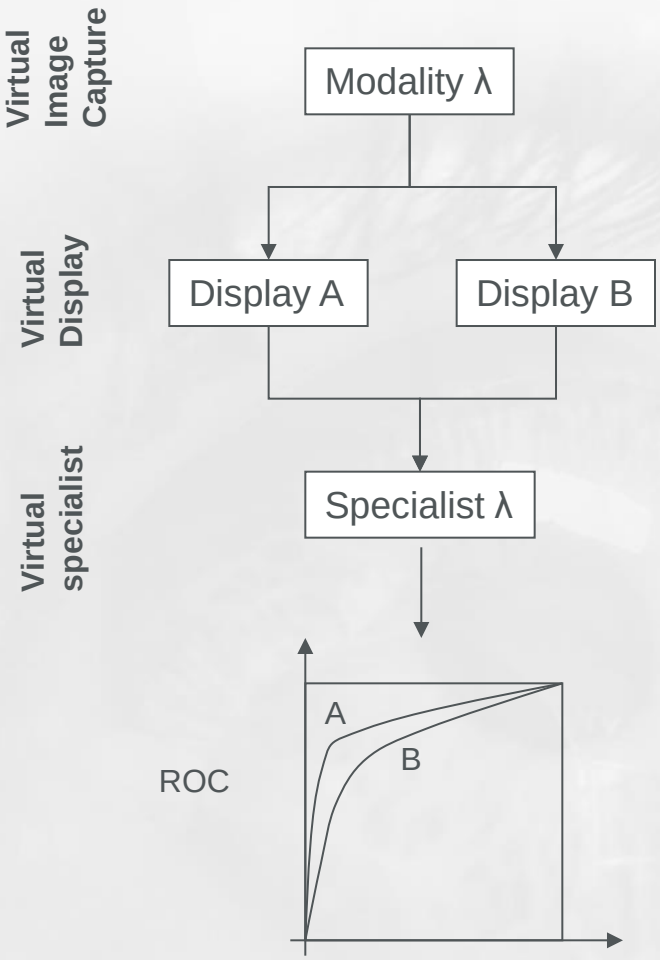
- VCT aims to develop a virtual medical imaging chain,

Start **Starting from: simulation of the image acquisition,**

  - **Over a hardware and software image processing pipeline,**

End **Ending in the visualization by the medical specialist on the image display.**

Goal



→For the specialist λ and modality λ,
display A is better than display B !

VCT

- Objective:
  - **Reduce number of observer studies**
  - **To quantify perceived image quality**
  - **Facilitate design of new medical systems**

VCT papers

- Reference paper:
  - **C. Marchessoux, T. Kimpe and T. Bert. Virtual image chain for perceived and clinical image quality of medical display. Invited paper to special issue on medical displays, IEEE Journal of Technology Display, volume 4, number 4, pp 356-368, September 2008, ISSN 1551-319X**
- All papers provided with the tutorial (pdf version) and a mevic_upenn.bib file with all the references

VCT papers

N. Odlum, G. Spalla, N. Van Assche, B. Vandenberghe, R. Jacobs, M. Quirynen & C Marchessoux. Preliminary display comparison for dental diagnostic applications. SPIE MI 2011

C. Marchessoux, N. Vivien, A. Kumcu and T. Kimpe. Validation of a new digital breast tomosynthesis medical display. SPIE MI 2011

L. Platisa, C. Marchessoux, B. Goossens and W. Philips. Performance evaluation of medical LCD displays using 3D channelized Hotelling observers. SPIE MI 2011

M. Vaz, Q. Besnehard and C. Marchessoux. 3D lesions insertion in digital breast tomosynthesis images. SPIE MI 2011

T. Kimpe, A. Xthona and C. Marchessoux. Medical Display Optimized for Digital Breast Tomosynthesis. breast imaging session, RNSA 2010

A. Vetsuypens, C. Marchessoux and T. Kimpe. A novel methodology for display 2D MTF evaluation: the Pixel Spread Function (PxSF). SPIE Medical Imaging, San Diego, 2010.

G. Braeckman, Joeri Barbarien, Q. Besnehard and C. Marchessoux. Perceptually optimal compression for heterogeneous image content in the context of medical networked applications. Submitted to SPIE Electronic Imaging, San Jose, 2010

G. Spalla, C. Marchessoux, M. Vaz and A. Ricker. Optimization of ct reconstruction parameters by using a medical virtual imaging chain. Medical Image Perception Conference XIII, Santa Barbara, 2009.

C. Marchessoux et al. Medical Virtual Imaging Chain. Medical Image Perception Conference XIII, Santa Barbara, 2009.

M.S. Vaz, R. Mersereau, G. Spalla and C. Marchessoux.CT Reconstruction from Truncated Scans. Fully3D, Beijing China, 5-10 September 2009

L. Ilic, E. Vansteenkiste, Bart Goossens, C. Marchessoux, T. Kimpe and W. Philips. Optimization of medical imaging display systems: using the channelized Hotelling observer for detecting lung nodules – experimental study. SPIE MI 2009

Mevic project invited for the SPIE 2009 Medical Imaging conference during the workshop session on observer interactions.

T. Kimpe, C. Marchessoux, G. Spalla, B. Goossens, H. Hallez, E. Vansteenkiste, S. Staelens and W. Philips. A software simulation framework to predict clinical performance of medical displays. Invited paper to SID, IDW 2008, Japan, December 2008

C. Marchessoux. Invited talk, Medical display conference, UKDL, 15th of October, 2008, London, UK

C. Marchessoux, G. Spalla and T. Kimpe. A new methodology for clinical and perceived quality of medical displays. SID, LA, USA, May 2008.

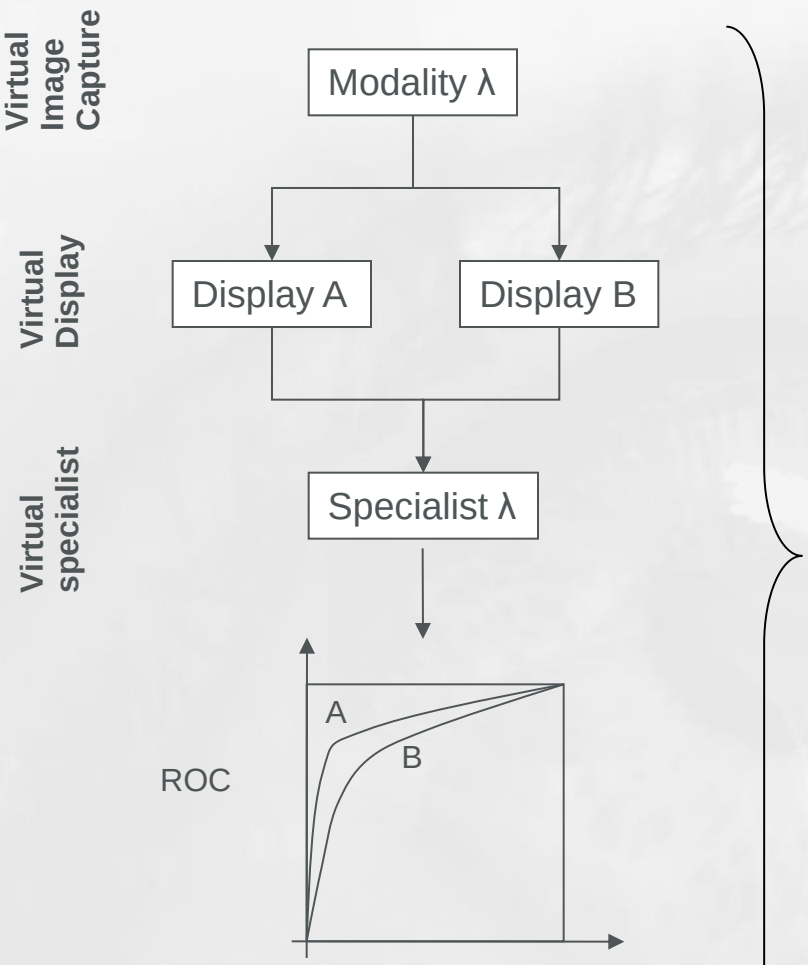
C. Marchessoux, A. Rombaut, T. Kimpe, B. Vermeulen and P. Demeester. Extension of a human visual system for display simulation. IS&T/SPIE EI, 2008.

C. Marchessoux, and T. Kimpe. Evaluating clinical performance of Coronis Fusion 6MP DL. White paper, 2007

C. Marchessoux and T. Kimpe. Specificities of a psycho-physical test room dedicated for medical display applications. International symposium, Society for Information Display, SID 2007, Long Beach, 15.3, book II, pp. 971-974, 2007

2. Architecture

Introduction



→For the specialist λ and modality λ, display A is better than display B !

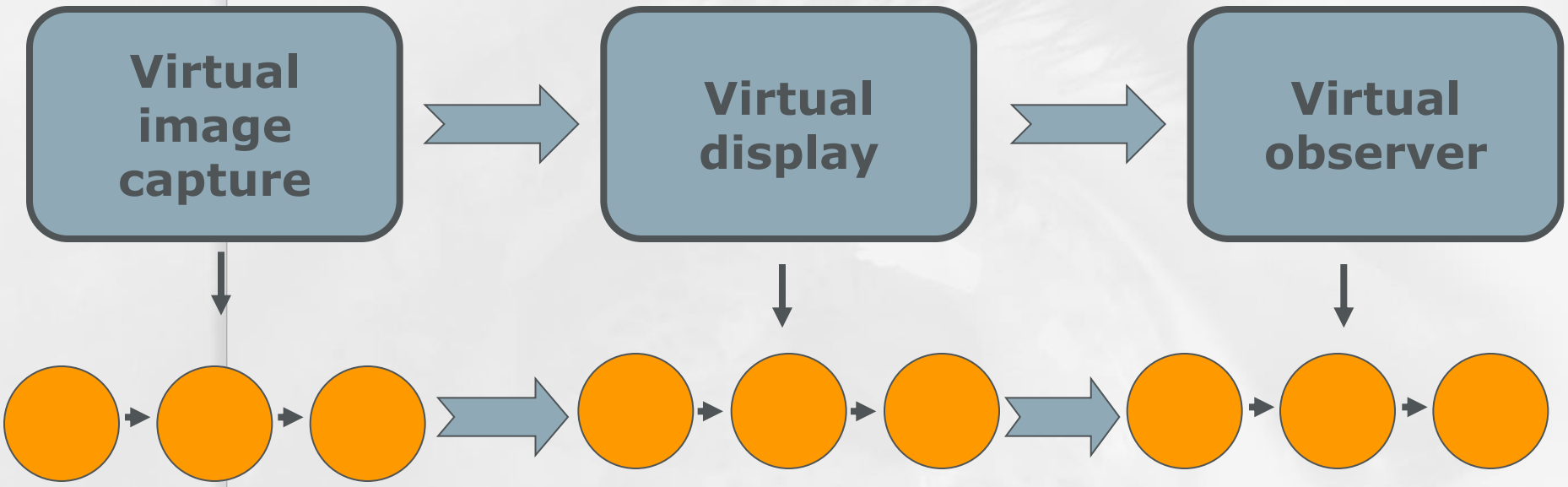
The simulation chain is a pipeline!

What do we need?

- A container to store the results at each stage
- Modules in C++ for data processing
- A configuration file defining:
  - Inputs
  - Modules:
    - the parameters names
    - the parameters values
  - Outputs

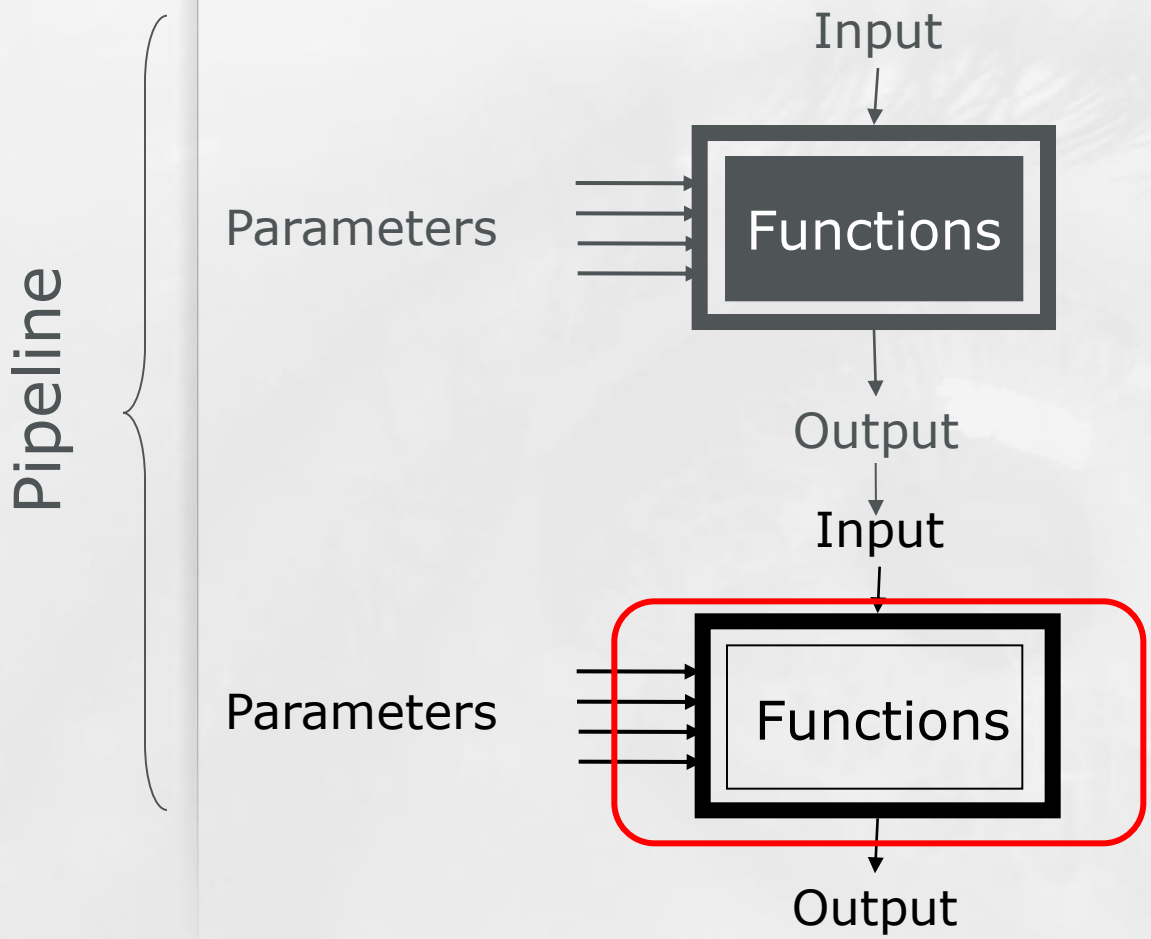
Principle

Principle



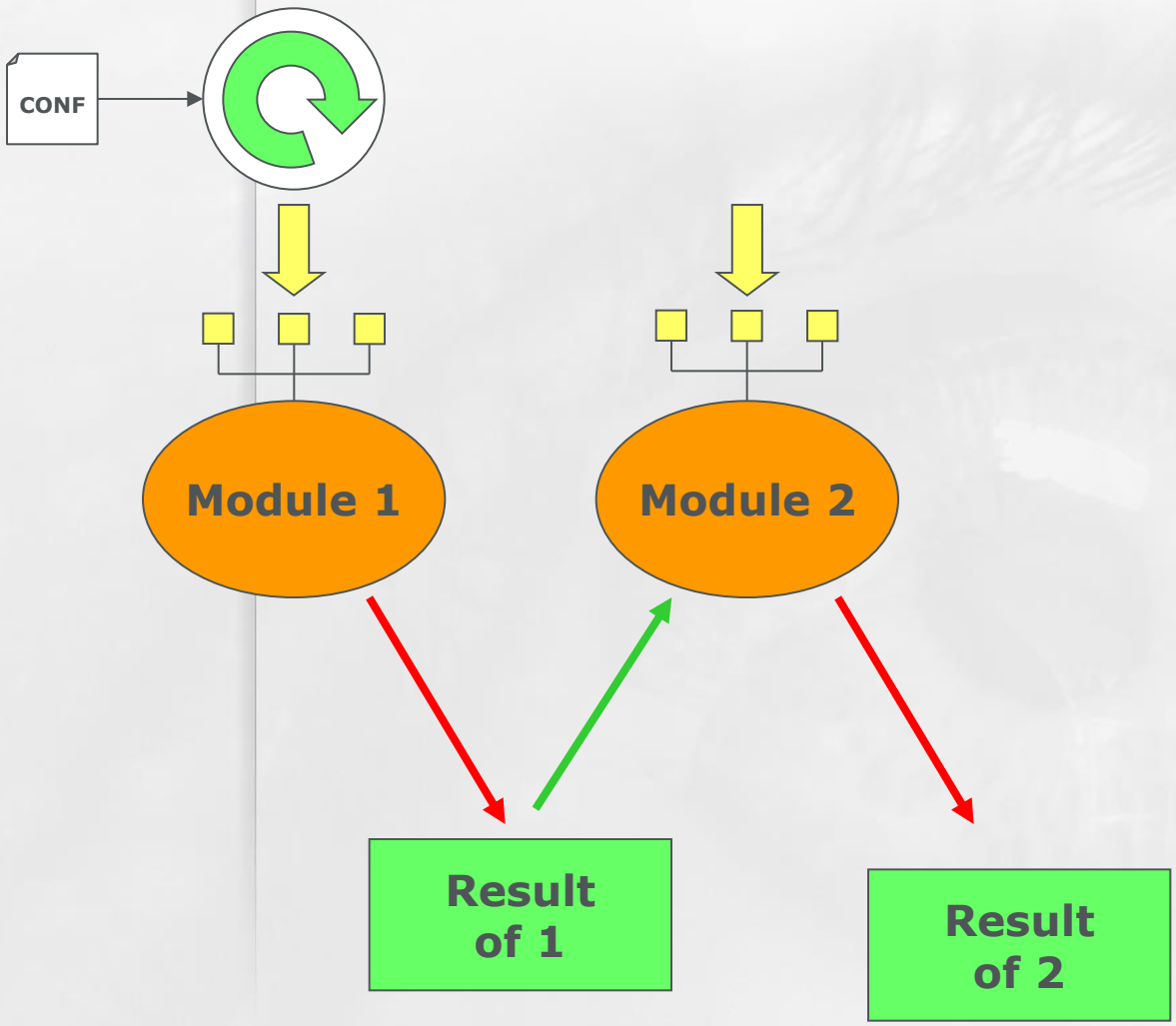
Each part is broken down as a chain of Processing units (or «modules»)

Mevic = Pipeline



Module:
Generates output data depending on input and parameters

Principle

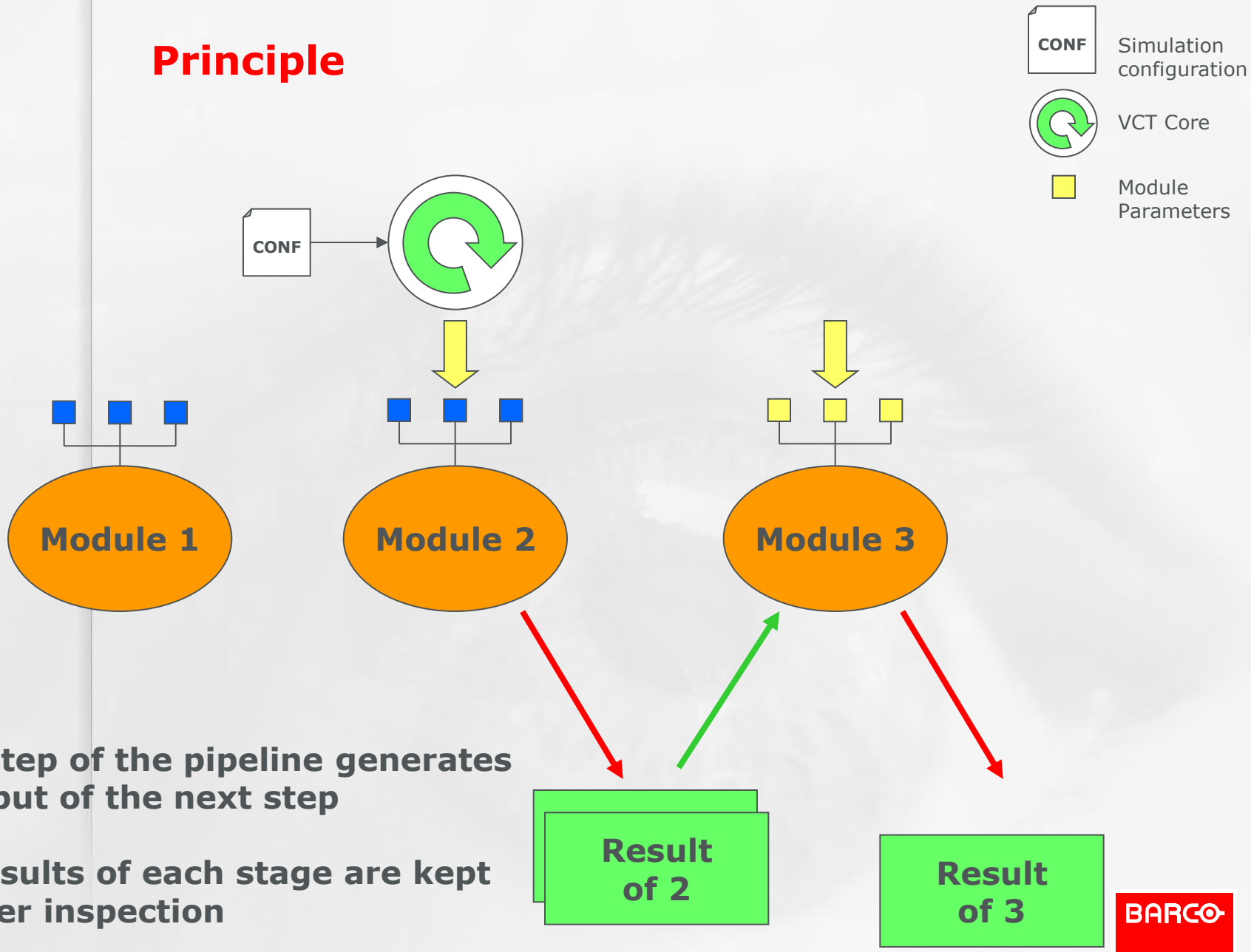


CONF Simulation configuration

MEVIC Core

Module Parameters

Principle



Each step of the pipeline generates the input of the next step

The results of each stage are kept for later inspection

Modules - Virtual image capture part

- Sequence loading (~dynamic background)
  - **SequenceRawGeneratorModule → *module to be used for loading raw images & sequences***

Modules - virtual display

- SequenceRAWGeneratorModule → *also used for controlling the frame repeat*
- VideoCardModule
- DisplayModule
- DisplayLutModule
- Rgb2XYZDisplayModule
- sRgbDisplayModule

Modules - virtual specialist

- Channelized Hottelling Module:
  - **SingleSliceCHOModule**
- Statistical analysis:
  - **MRMCModule**

Modules available today - others

- Various input and output modules
  - **ReaderModule: to load a simulation**
  - **WriterModule: to save a simulation**
- Various conversion modules
  - **ConversionDDL2CDModule: ddl to cd/m²**
  - **ConvertRawToChoBinModule: convert raw images to simulation files**
- Color metrics:
  - **DeltaE2000Module: ΔE_{2000}**

Modules - others

- Convenient modules for debugging:
  - **SaveFrameTXTModule**

    → *Module used currently for saving and checking intermediate results in the simulation, mainly for debugging*
  - **SaveFrameBMPModule**
  - **SaveFrameRAWModule**

Dependencies

- Boost
- OpenCV
- Meschach
- OpenCL
- xmlParser
- zlib

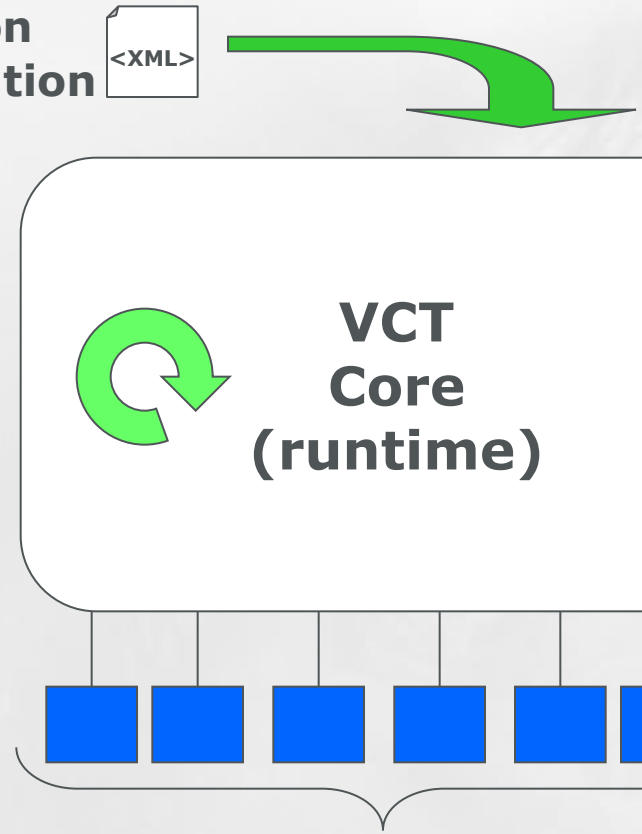
Why to use it?

- Dedicated platform for simulation:
  - **Runs fast (C++)**
  - **Can handle any kind of image format:**
    - n-channels: 1, 3, 4 ....
    - n-frames....
    - n-Dimensions…
    - int, float, int[10]....
    - Any kind of representation and unit
    - The only limit is our mind and computer capacity!
  - **Easy integration of new algorithms as Modules**
  - **Easily configurable automated simulation**
    - Through SuperXML and "common xml" configuration files
  - **Support for compressed output (Huffman)**

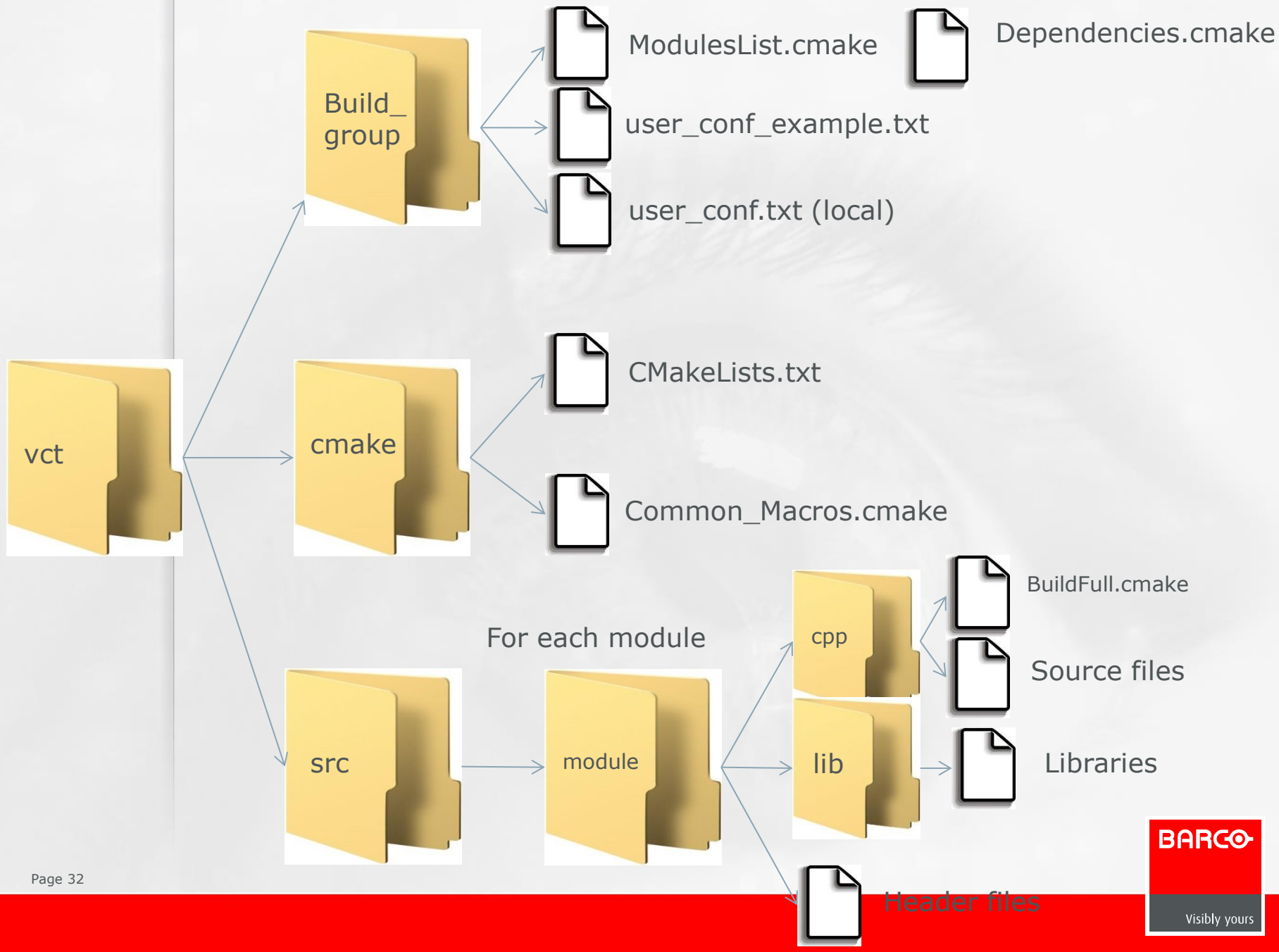
Platform architecture

User input

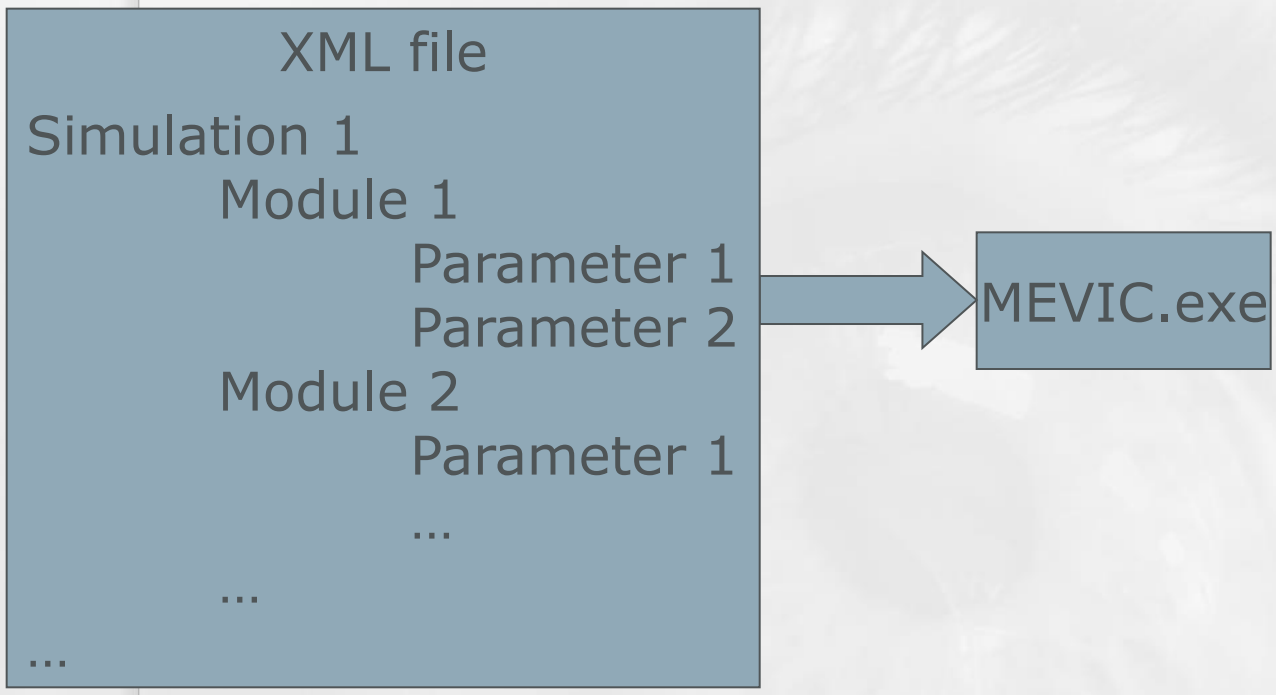
- Simulation configuration
- Data



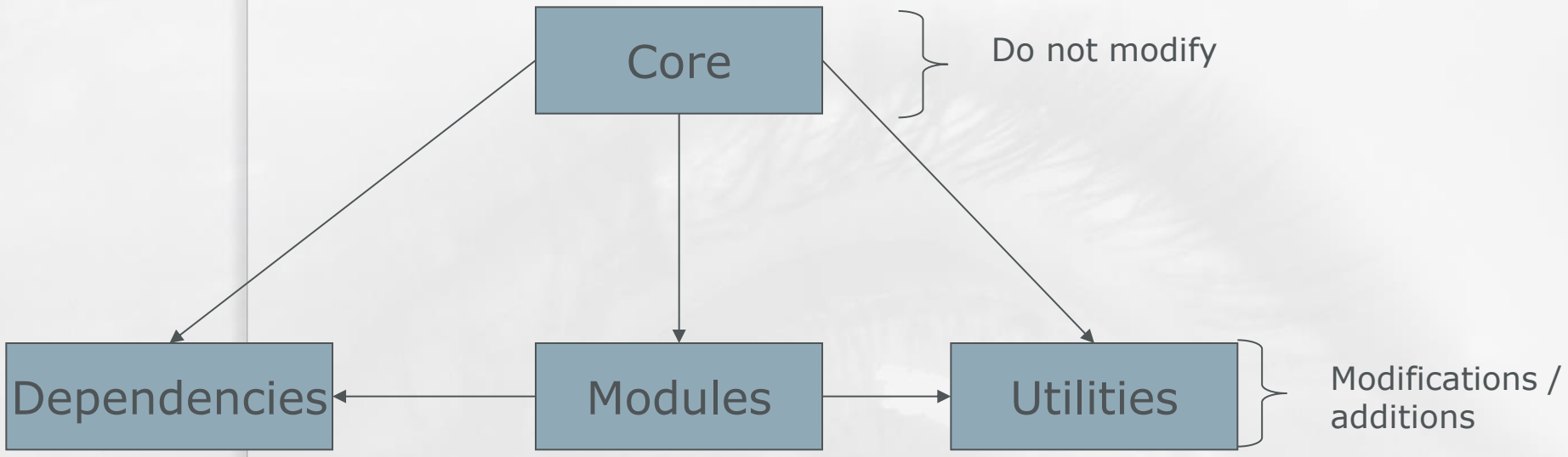
Project organization

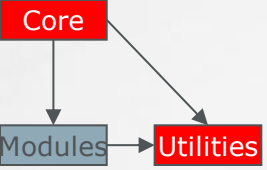


Software design



Software design





Software design

Parameters management

SuperMEVICXML

Simulation management

FactoryModule

ListOfModules

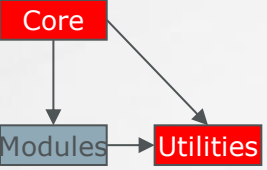
AutomaticRun

ComponentHeader

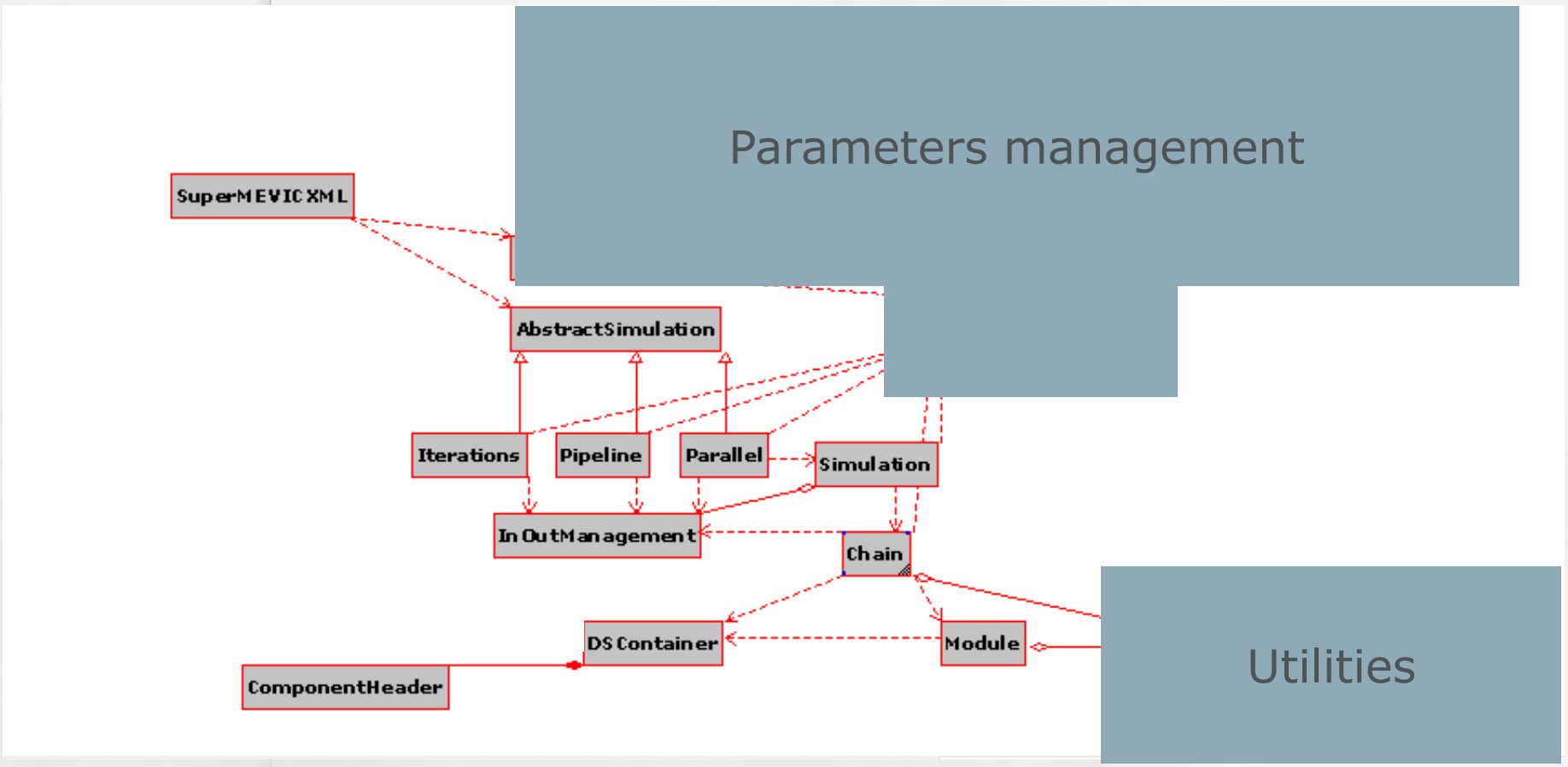
DS Container

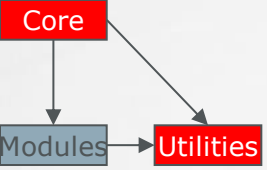
Module

Utilities

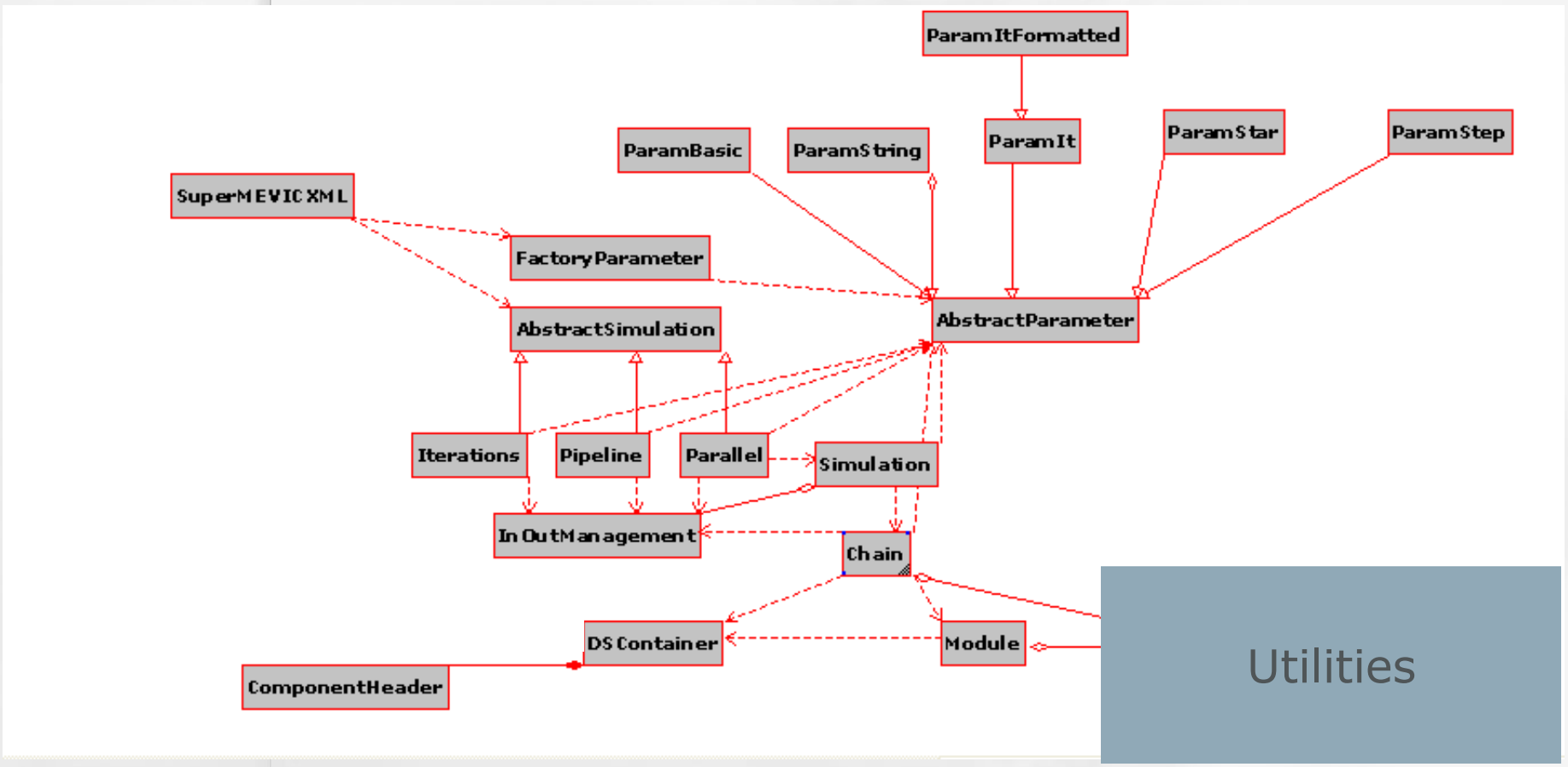


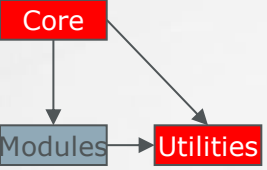
Software design



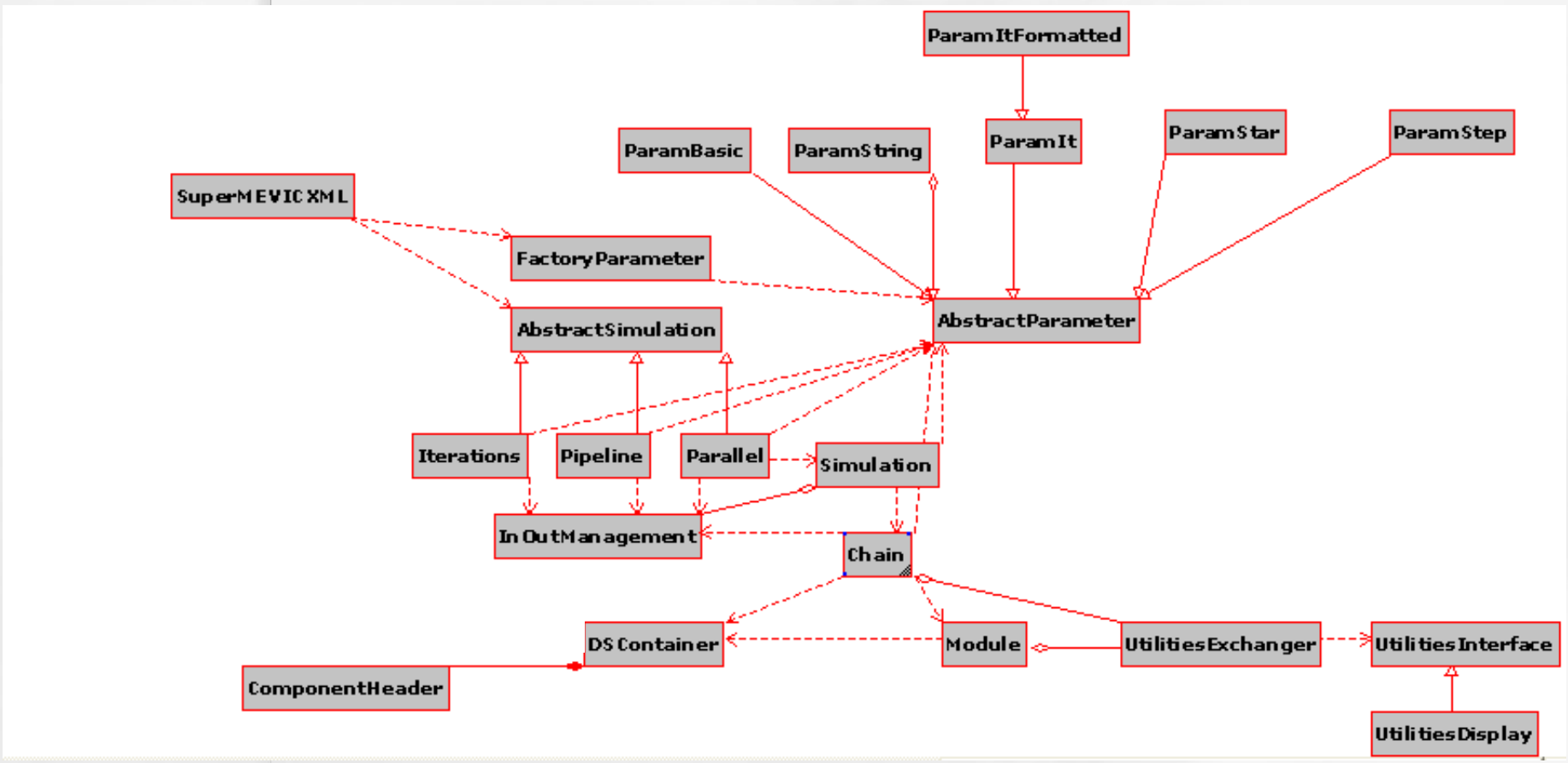


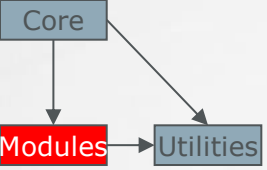
Software design



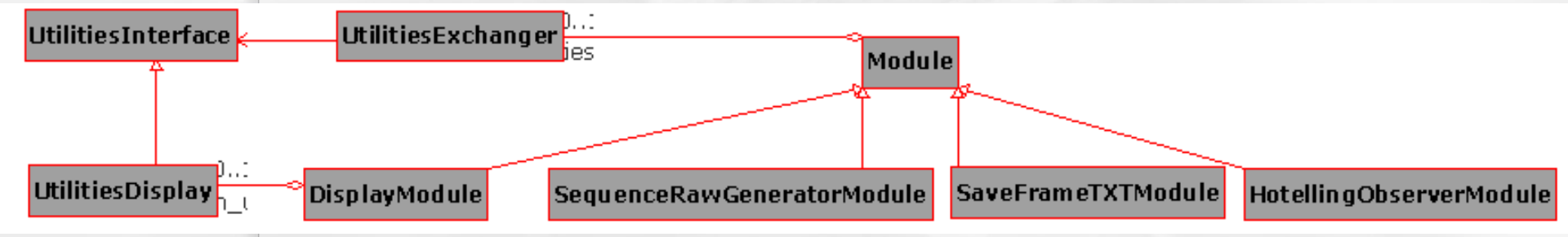


Software design





Software design



(many more modules are present)

3. Configuring a simulation

SuperXML

- The configuration format – SuperXML – tries to address the following problems :
  - **One simulation at a time: requires workarounds to perform simulations on a batch of data.**
  - **Fixed file paths: the XML needs to be edited for each machine that needs to run the simulation.**
  - **Parameters "lost" in the XML file**

SuperXML

- Many simulations in one configuration file

Root node: the same for all SuperXML files

Special headers described later

Each simulation is given a unique arbitrary name: here REF, TEST, JND

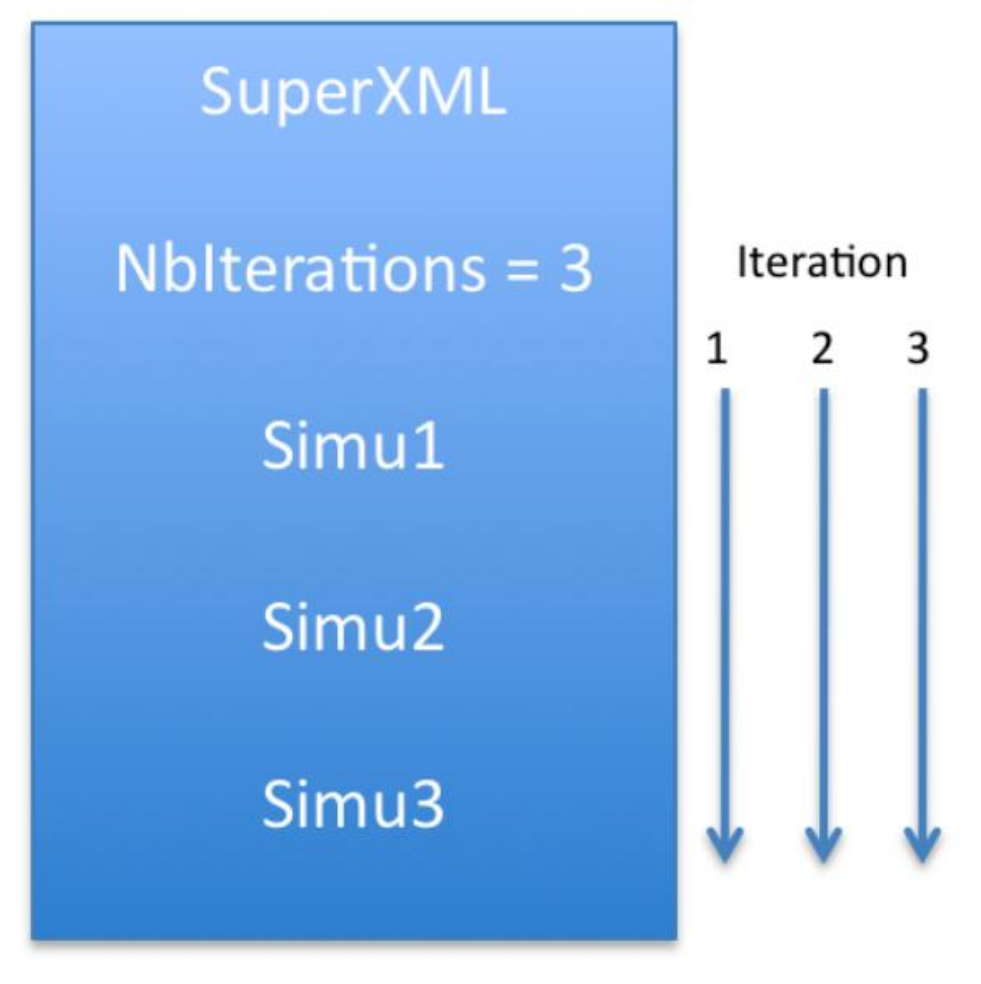
Each simulation has its own list of Modules

```
<SuperXML>
    <LIST_OF_ITERATIONS name = "pipeline" value="1">
    <LIST_OF_PARAMETERS>
    <TEMPLATE_MEVIC_SIMULATION>
        <REF description="MEVIC_SIMULATION">
            <BackgrColor24BMPGeneratorModule>
                <Filename dataType="char" value="$" />
            </BackgrColor24BMPGeneratorModule>
            <SimpleColorDisplayModule>
                <Filename dataType="char" description="Filename" value="F:\qube\MEVICforCompression\full.csv"/>
            </SimpleColorDisplayModule>
            <WriterModule name="WriterModule">
                <Debug dataType="bool" description="Debug" value="false" />
                <Filename dataType="string" description="Filename" value="$" />
                <Compression dataType="bool" description="Compression" value="true" />
                <DeleteContainer dataType="bool" description="DeleteContainer" value="true" />
            </WriterModule>
        </REF>
        <TEST description="MEVIC_SIMULATION">
            <BackgrColor24BMPGeneratorModule>
                <Filename dataType="char" value="$" />
            </BackgrColor24BMPGeneratorModule>
            <SimpleColorDisplayModule>
                <Filename dataType="char" description="Filename" value="F:\qube\MEVICforCompression\full.csv"/>
            </SimpleColorDisplayModule>
            <WriterModule name="WriterModule">
                <Debug dataType="bool" description="Debug" value="false" />
                <Filename dataType="string" description="Filename" value="$" />
                <Compression dataType="bool" description="Compression" value="true" />
                <DeleteContainer dataType="bool" description="DeleteContainer" value="true" />
            </WriterModule>
        </TEST>
        <JND description="MEVIC_SIMULATION">
            <ReaderModule module="RM1">
                <Debug dataType="bool" description="Debug" value="false" />
                <Filename dataType="string" description="Filename" value="$" />
                <Compression dataType="bool" description="Compression" value="true" />
            </ReaderModule>
            <ReaderModule module="RM2">
                <Debug dataType="bool" description="Debug" value="false" />
                <Filename dataType="string" description="Filename" value="$" />
```

SuperXML: pipeline mode

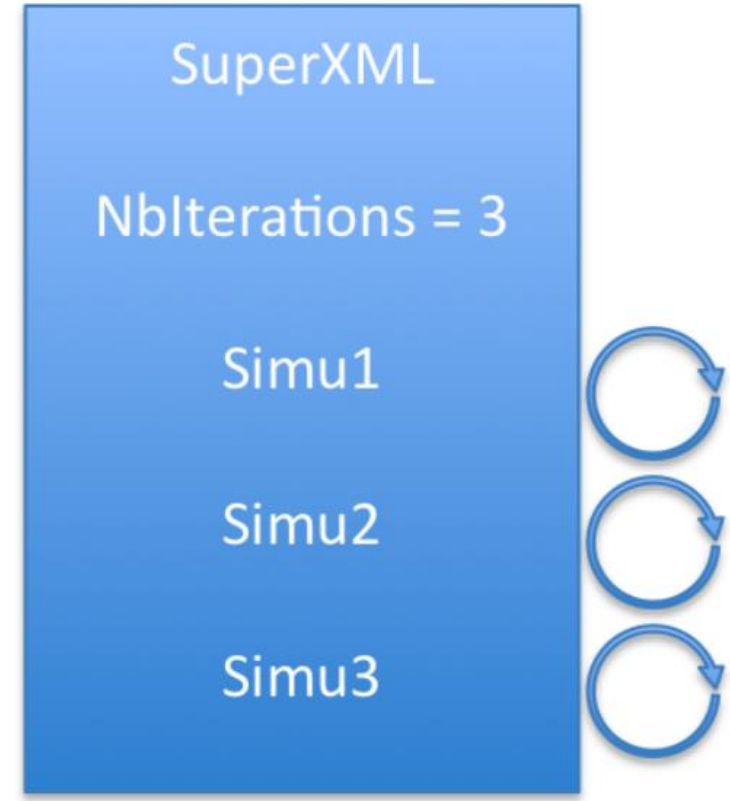
<LIST_OF_ITERATIONS name = "pipeline" value = "3">

Special header
just after the SuperXML node



SuperXML: iteration mode

- <LIST_OF_ITERATIONS name = "iterations">
  <Simu1 value="3">
  <Simu2 value="4">
  <Simu3 value="10">
- </LIST_OF_ITERATIONS>



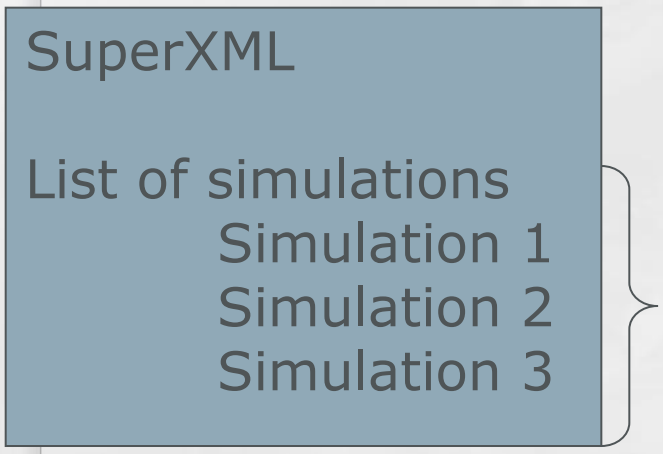
Simu1 is performed 3 times. Then…

… Simu2 is performed 4 times. And finally…

… Simu3 is performed 10 times.

SuperXML: parallel mode

- <LIST_OF_ITERATIONS name="parallel" value="5">



1 List of simulation per processor.

To take into account: optimize the number of simulations according to the number of processors.

SuperXML

- Increasing parameters
  - <SSIMModule name="pathForMaps" value = "[0:1:9]"/>
  - **0, 1, 2, 3, …, 9**
- Formated parameters
  - <SSIMModule name="pathForMaps" value = "[0:1:9][number:000]"/>
  - **001, 002, 003, …, 009**
- List of parameters
  - <BackgrBMPGeneratorModule name = "Filename" value = "[0,0.4,9,8]"/>
  - **0, 0.4, 9, 8**
- Wildcards parameters
  - <BackgrBMPGeneratorModulename name="Filename" value ="F:\gusp\[*].bmp"/>
  - **All bmp files in the "F:\gusp\" directory**

SuperXML

```
xml version="1.0" encoding="ISO-8859-1"?>
uperXML>
  <LIST_OF_ITERATIONS name = "pipeline" value="1">
  </LIST_OF_ITERATIONS>
  <LIST_OF_PARAMETERS>
      <GlobalParameter name ="number_of_slices" value="10"/>
      <LocalParameter name = "REF">
      <SequenceRawGeneratorModule name="directory" value = "#\exercises\input\sequence_raw_generator_module\ref\raw"/>
      <WriterModule name="Filename" value = "#\exercises\test_global_param_3_a\writer_module\reference\reference.bin"/>
      </LocalParameter>
      <LocalParameter name = "TEST">
      <SequenceRawGeneratorModule name="directory" value = "#\exercises\input\sequence_raw_generator_module\test\raw"/>
      <WriterModule name="Filename" value = "#\exercises\test_global_param_3_a\writer_module\test\test.bin"/>
      </LocalParameter>
  </LIST_OF_PARAMETERS>
  <TEMPLATE_MEVIC_SIMULATION>
      <REF>
          <SequenceRawGeneratorModule>
              <directory dataType="char" value = "$"/>
              <number_of_slices dataType="int" value = "$number_of_slices"/>
              <width dataType = "int" value = "512"/>
              <height dataType = "int" value = "512"/>
              <nbBitsRange dataType = "int" value = "8"/>
              <_0bigEndian_1littleEndian dataType = "int" value = "1"/>
              <_1WhiteIs0_00therwise dataType = "int" value = "0"/>
              <nbBitsOutput dataType = "int" value = "8"/>
              <nbBitsPrecision dataType = "int" value = "8"/>
              <frame_repeat dataType = "int" value = "1"/>
              <_1RGB_0GRAY dataType = "int" value = "0"/>
          </SequenceRawGeneratorModule>
          <WriterModule>
              <Filename dataType="char" value="$"/>
              <Compression dataType="int" value="true"/>
              <DeleteContainer dataType="int" value="1"/>
          </WriterModule>

      </REF>
      <TEST>
          <SequenceRawGeneratorModule>
              <directory dataType="char" value = "$"/>
              <number_of_slices dataType="int" value = "$number_of_slices"/>
              <width dataType = "int" value = "512"/>
              <height dataType = "int" value = "512"/>
              <nbBitsRange dataType = "int" value = "8"/>
              <_0bigEndian_1littleEndian dataType = "int" value = "1"/>
              <_1WhiteIs0_00therwise dataType = "int" value = "0"/>
              <nbBitsOutput dataType = "int" value = "8"/>
              <nbBitsPrecision dataType = "int" value = "8"/>
              <frame_repeat dataType = "int" value = "1"/>
              <_1RGB_0GRAY dataType = "int" value = "0"/>
          </SequenceRawGeneratorModule>
          <WriterModule>
              <Filename dataType="char" value="$"/>
              <Compression dataType="int" value="true"/>
              <DeleteContainer dataType="int" value="1"/>
          </WriterModule>
      </TEST>
  </TEMPLATE_MEVIC_SIMULATION>
SuperXML>
```

The global parameter "name" with a value of "10" can be used in any simulations of the pipeline

The use of a global parameter is marked by a "$name" symbol followed by the name of the parameter

- Global Parameters
  - **Available for all parts of the simulation**
  - **For pipeline only**

SuperXML

- One extra parameter given as argument with '#':

MEVIC.exe SuperXML 3_a.xml d:\data\project\

```
xml version="1.0" encoding="ISO-8859-1"?>
uperXML>
  <LIST_OF_ITERATIONS name = "pipeline" value="1">
  </LIST_OF_ITERATIONS>
  <LIST_OF_PARAMETERS>
      <GlobalParameter name ="number_of_slices" value="10"/>
      <LocalParameter name = "REF">
      <SequenceRawGeneratorModule name="directory" value = "#\exercises\input\sequence_raw_generator_module\ref\raw"/>
      <WriterModule name="Filename" value = "#\exercises\test_global_param_3_a\writer_module\reference\reference.bin"/>
      </LocalParameter>
      <LocalParameter name = "TEST">
      <SequenceRawGeneratorModule name="directory" value = "#\exercises\input\sequence_raw_generator_module\test\raw"/>
      <WriterModule name="Filename" value = "#\exercises\test_global_param_3_a\writer_module\test\test.bin"/>
      </LocalParameter>
  </LIST_OF_PARAMETERS>
  <TEMPLATE_MEVIC_SIMULATION>
      <REF>
          <SequenceRawGeneratorModule>
              <directory dataType="char" value = "$"/>
              <number_of_slices dataType="int" value = "$number_of_slices"/>
              <width dataType = "int" value = "512"/>
              <height dataType = "int" value = "512"/>
              <nbBitsRange dataType = "int" value = "8"/>
              <_0bigEndian_1littleEndian dataType = "int" value = "1"/>
              <_1WhiteIs0_00therwise dataType = "int" value = "0"/>
             <nbBitsOutput dataType = "int" value = "8"/>
              <nbBitsPrecision dataType = "int" value = "8"/>
              <frame_repeat dataType = "int" value = "1"/>
              <_1RGB_0GRAY dataType = "int" value = "0"/>
           </SequenceRawGeneratorModule>
          <WriterModule>
              <Filename dataType="char" value="$"/>
              <Compression dataType="int" value="true"/>
              <DeleteContainer dataType="int" value="1"/>
          </WriterModule>

      </REF>
```

SuperXML

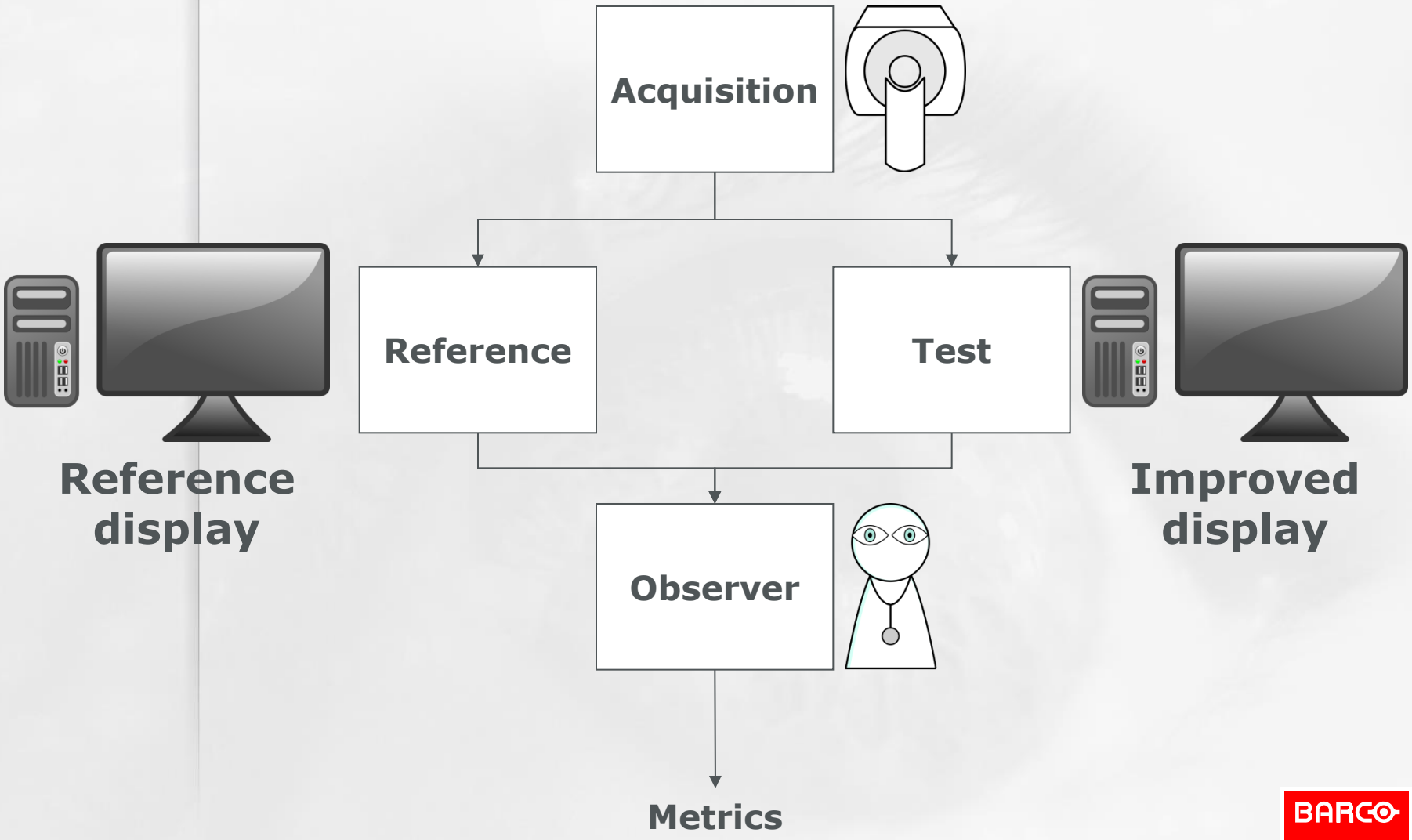
- More than one parameter given as argument with '#n':

MEVIC.exe SuperXML 3_a.xml d:\data\project\ d:\data\project2\

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SuperXML>
    <LIST_OF_ITERATIONS name = "pipeline" value="1">
    </LIST_OF_ITERATIONS>
    <LIST_OF_PARAMETERS>
        <GlobalParameter name ="number_of_slices" value="10"/>
        <LocalParameter name = "REF">
        <SequenceRawGeneratorModule name="directory" value = "#0\exercises\input\sequence_raw_generator_module\ref\raw"/>
        <WriterModule name="Filename" value = "#0\exercises\test_global_param_3_a\writer_module\reference\reference.bin"/>
        </LocalParameter>
        <LocalParameter name = "TEST">
        <SequenceRawGeneratorModule name="directory" value = "#1\exercises\input\sequence_raw_generator_module\test\raw"/>
        <WriterModule name="Filename" value = "#1\exercises\test_global_param_3_a\writer_module\test\test.bin"/>
        </LocalParameter>
    </LIST_OF_PARAMETERS>
```

Up to 5 extra arguments

SuperXML - Plugging pipelines together



SuperXML - Plugging pipelines together

- Solution: temporary files and simulation linkage

These parameters values tell the runtime that the value should be automatically generated by the platform

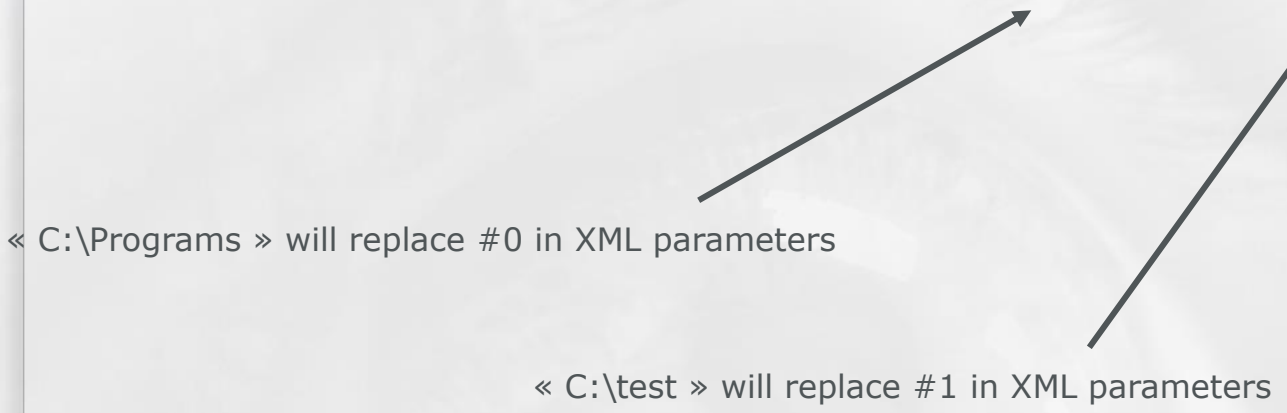
These parameters values tell the runtime that the value should be those generated by the [output:#] directives

```
<TEMPLATE_MEVIC_SIMULATION>
    <REF>
        <WriterModule>
            <Filename dataType="string" description="Filename" value="[output:1]"/>
        </WriterModule>
    </REF>
    <TEST>
        <WriterModule>
            <Filename dataType="string" description="Filename" value="[output:2]"/>
        </WriterModule>
    </TEST>
    <SSIM>
        <ReaderModule module="RM1">
            <Filename dataType="string" description="Filename" value="[input:1]"/>
        </ReaderModule>
        <ReaderModule module="RM2">
            <Filename dataType="string" description="Filename" value="[input:2]"/>
        </ReaderModule>
    </SSIM>
</TEMPLATE_MEVIC_SIMULATION>
```

SuperXML: argument parameters

- Its possible to pass parameters to a SuperXML file through the command line

```
MEVIC.exe SuperXML F:\test\test.xml C:\Programs C:\test
```



- If this line is present in the XML file…

```
<Filename dataType=«char» description=«Filename» value=«#1\data.bin»/>
```

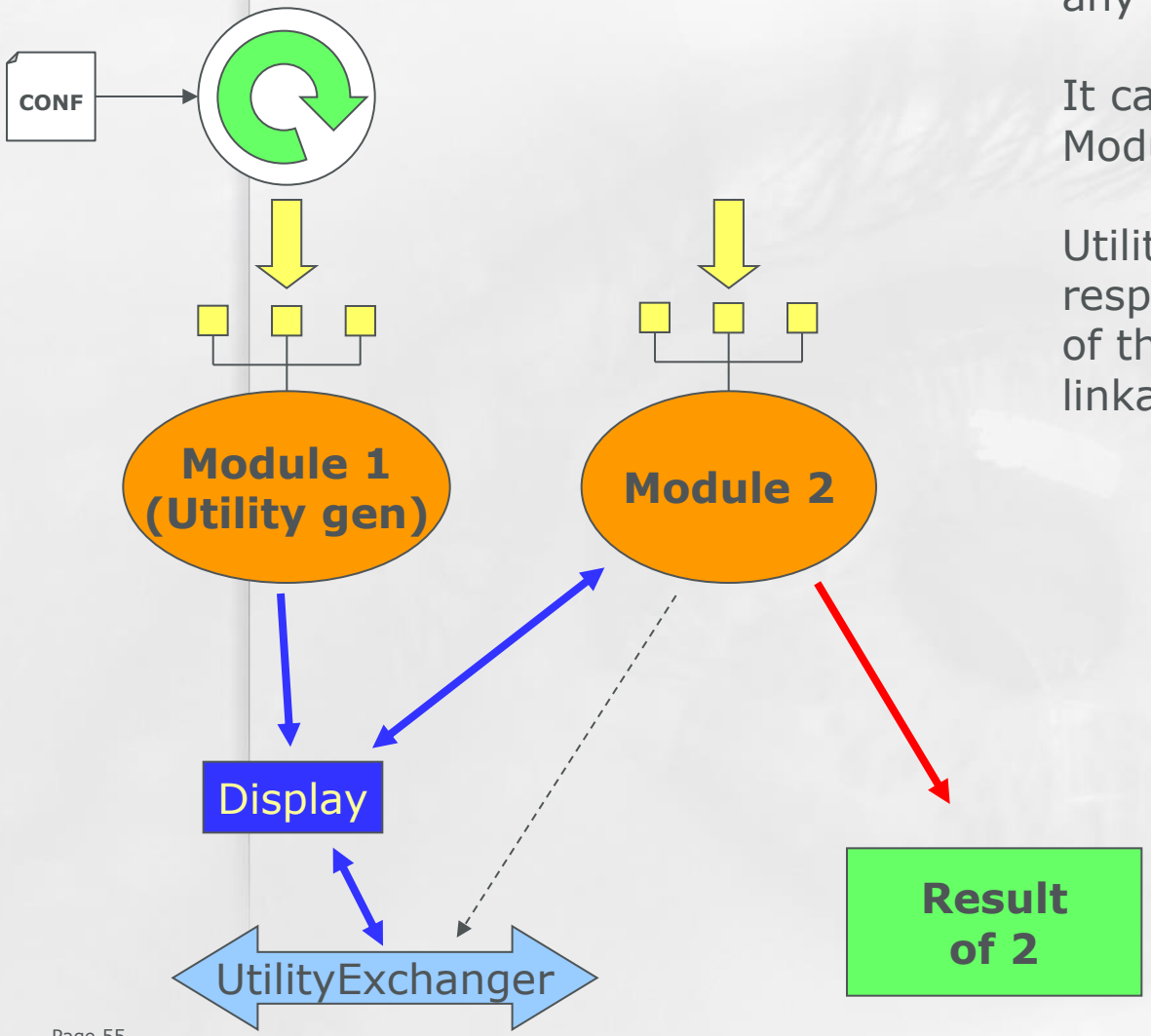
- … then « C:\test\data.bin » will be the value of the parameter for the running simulation

4. Utilities

Principle

- Utilities are shared data objects.
- They are used to pass data between Modules when the data cannot be expressed as image data.
- Example:
  - **Many modules in the Virtual Display part need information about the display they simulate (e.g. native curve, calibrated curve,…).**
  - **Therefore a UtilityDisplay exists.**
- Utilities are created by inserting “Utility generation” Modules in the simulation.

Principle



The Utility Exchanger is an object that can be accessed by any Module.

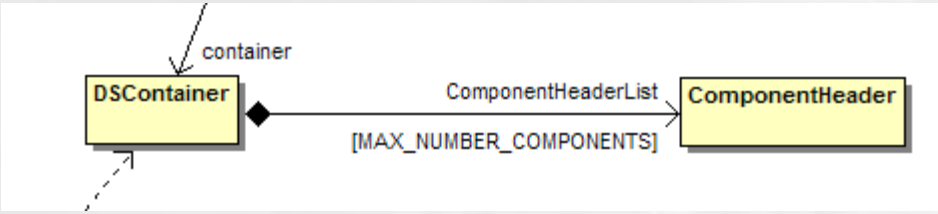
It can be queried by the Modules for existing utilities.

Utility generation Modules are responsible for the instantiation of the utility objects and their linkage to the Exchanger.

5. Data in VCT: Containers and Components

In 2 files and 2 classes

- DSContainer.h
- DSContainer.cpp
  - **2 classes:**

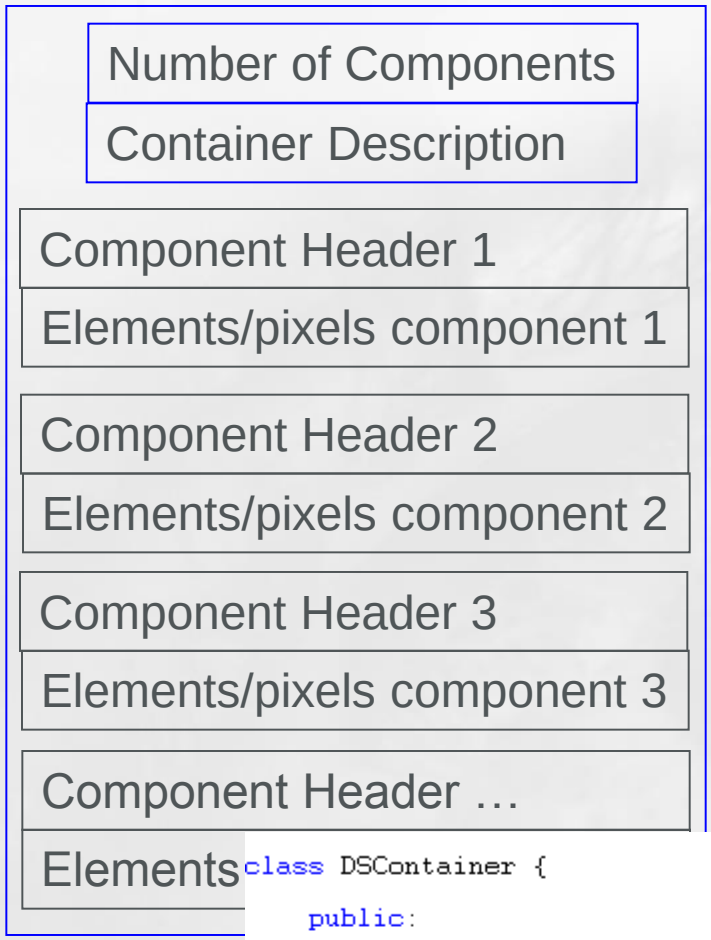


ComponentHeader describes the data

```
/*!
 * \class ComponentHeader
 * \brief this class store all the information about a component
 */
class ComponentHeader { ... };
/*!
 * \class DSContainer
 * \brief This class store a list of components
 *        and allow to manage them
 *        There are 4 possibility to create a component:
 *          - with default component values
 *          - with all component values using enum
 *          - with all component values without using enum
 *          - from an existing component (used to transfer components between different memory spaces (OpenCL))
 */
class DSContainer { ... };
#endif // DSCONTAINER_H
```

DSContainer is a list of ComponentHeader objects. It provides an interface to create, access and modify ComponentHeader objects.

Container organization



The whole structure is very simple so as to be saved in a binary file

```
class DSContainer {

    public:

        DSContainer();
        ~DSContainer();

        ComponentHeader ComponentHeaderList[MAX_NUMBER_COMPONENTS];
```

Container description

```
ComponentHeader ComponentHeaderList[MAX_NUMBER_COMPONENTS]; //!< component header of the container simulation
```

```
// container header
std::string m_description; //!< container description
```

Create a component

- 4 methods:

```
/*!
 * \fn CreateComponent
 * \brief function to create a component without enum information
 * \param nbRowElemPerFrame: nb row per frame
 * \param nbColumnElemPerFrame: nb column per frame
 * \param elemSizeCol: the size of a column
 * \param elemSizeRow: the size of a row
 * \param elemType: the enum type of element
 * \return int: 0 if ok, -1 otherwise
 */
int CreateComponent(unsigned long nbRowElemPerFrame, unsigned long nbColumnElemPerFrame, enum ComponentType elemType);

/*!
 * \fn CreateComponent
 * \brief Method CreateComponent using char*
 *        Function to create a component
 * \param nbRowElemPerFrame: nb row per frame
 * \param nbColumnElemPerFrame: nb column per frame
 * \param elemSizeCol: the size of a column
 * \param elemSizeRow: the size of a row
 * \param unit: the enum unit of the component
 * \param nbBits: the number of bits of the component (8, 16, 24...)
 * \param whitePt: X0 Y0 Z0 values of the white point
 * \param illum: the enum illuminant
 * \param obs: the enum observer
 * \param elemType: the enum type of element
 * \param nbFrames: the number of frames
 * \param frameRate: the m_frameRate if we work on a video
 * \param allocator_type: type of memory where the data is stored (optional)
 * \return int: 0 if ok, -1 otherwise
 */
int CreateComponent(unsigned long nbRowElemPerFrame, unsigned long nbColumnElemPerFrame,
                    float elemSizeCol, float elemSizeRow,
                    const std::string& unit,
                    unsigned int nbBits,
                    const float whitePt[3],
                    const std::string& illum,
                    const std::string& obs,
                    enum ComponentType elemType,
                    unsigned int nbFrames,
                    float frameRate,
                    enum AllocatorType allocator_type = default_allocator
                    );

/*!
 * \fn CreateComponent
 * \brief function to create a component using enum
 * \param nbRowElemPerFrame: nb row per frame
 * \param nbColumnElemPerFrame: nb column per frame
 * \param elemSizeCol: the size of a column
 * \param elemSizeRow: the size of a row
 * \param unit: the enum unit of the component
 * \param nbBits: the number of bits of the component (8, 16, 24...)
 * \param whitePt: X0 Y0 Z0 values of the white point
 * \param illum: the enum illuminant
 * \param obs: the enum observer
 * \param elemType: the enum type of element
 * \param nbFrames: the number of frames
 * \param frameRate: the m_frameRate if we work on a video
 * \param allocType: type of memory where the data is stored (optional)
 * \return int: 0 if ok, -1 otherwise
 */
int CreateComponent(unsigned long nbRowElemPerFrame, unsigned long nbColumnElemPerFrame,
                    float elemSizeCol, float elemSizeRow,
                    enum UNIT unit,
                    unsigned int nbBits,
                    const float whitePt[3],
                    enum ILLUMINANT illum,
                    enum OBSERVER obs,
                    enum ComponentType elemType,
                    unsigned int nbFrames,
                    float frameRate,
                    enum AllocatorType allocType = default_allocator
                    );

/*!
 * \fn CreateComponent
 * \brief function to create a component with default values
 * \param nbComp: the number of the component to create from
 * \param targetAllocator: the allocator to use for the new component
 * \return int: 0 if ok, -1 otherwise
 */
int CreateComponent(int nbComp, enum AllocatorType targetAllocator);
```

Example:

```
DSContainer* container = new DSContainer();
container->CreateComponent(10,10,1);
delete container;
```

Component Header

```
/*!
 * \class ComponentHeader
 * \brief this class store all the information about a component
 */
class ComponentHeader
{
    public:
        /*!
         * Constructor
         */
        ComponentHeader();

        /*!
         * Destructor
         */
        ~ComponentHeader();

        static const size_t DESCRIPTION_SIZE = 1024; //!< description size 1024 characters

        // component header attributes
        unsigned char * m_baseAddress; //!< raw data
        enum AllocatorType m_allocType;
        unsigned long m_length;
        unsigned long m_nbElemPerFrame;
        enum ComponentType m_elemType;
        char m_description[DESCRIPTION_SIZE];
        int m_validComp;
        float m_frameRate; //!< m_frameRate=0 means not a video, static image
        unsigned int m_nbFrames;
        unsigned long m_nbRowElemPerFrame; //!< height
        unsigned long m_nbColumnElemPerFrame; //!< width
        unsigned int m_nbBits; //!< 8 10 12 ...
        float m_elemSizeCol; //!< in mm
        float m_elemSizeRow; //!< in mm
        enum UNIT m_unit; //!< cd/m2, gray, JND
        float m_whitePt[3]; //!< Xo, Yo, Zo in cd/m2
        enum ILLUMINANT m_illum; //!< A, B, C....D50...F11
        enum OBSERVER m_obs; //!< no_observer, deg_2, deg_10
        int m_elemSize;


    };
```

Component Header: base_address

- void pointer to memory where to store pixel elements

```
        // component header attributes
        unsigned char * m_baseAddress; //!< raw data
        enum AllocatorType m_allocType;
        unsigned long m_length;
        unsigned long m_nbElemPerFrame;
        enum ComponentType m_elemType;
        char m_description[DESCRIPTION_SIZE];
        int m_validComp;
        float m_frameRate; //!< m_frameRate=0 means not a video, static image
        unsigned int m_nbFrames;
        unsigned long m_nbRowElemPerFrame; //!< height
        unsigned long m_nbColumnElemPerFrame; //!< width
        unsigned int m_nbBits; //!< 8 10 12 ...
        float m_elemSizeCol; //!< in mm
        float m_elemSizeRow; //!< in mm
        enum UNIT m_unit; //!< cd/m2, gray, JND
        float m_whitePt[3]; //!< Xo, Yo, Zo in cd/m2
        enum ILLUMINANT m_illum; //!< A, B, C....D50...F11
        enum OBSERVER m_obs; //!< no_observer, deg_2, deg_10
        int m_elemSize;


    };
```

ComponentHeader: component valid

- Valid if == 1

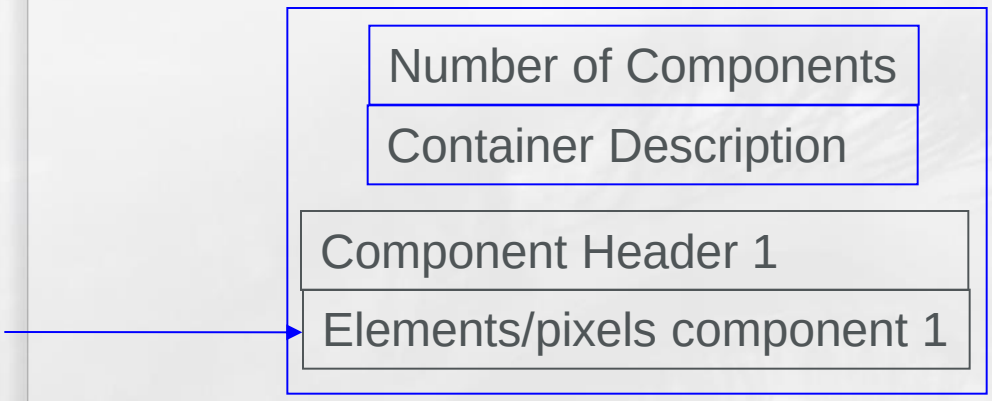
```
int m_validComp;
```

```
/*!
 * \fn GetComponentValid
 * \brief accessor for checking if a component is valid
 * \param nbComp: the number of the component
 * \return int: 0 if ok, -1 otherwise
 */
int GetComponentValid(int nbComp);
```

ComponentHeader: element type

```
enum ComponentType m_elemType;
```

Describes the content and the intent of the component: physical/digital, int/float/double/…, number of channels




```
enum ComponentType
{
    TYPE_BYTE=0,
    TYPE_FLOAT=1,
    TYPE_DOUBLE=2,
    TYPE_LONG=3,
    TYPE_CHAR=4,
    TYPE_INT=5,

    TYPE_FFT=6, // 2 channels: Fast Fourier Transform, channel 1: Real or Amplitude part, channel 2: Imaginary or Phase part

    // Color space representation
    TYPE_IMAGE_GRAY=7,  // 1 channel: Gray
    TYPE_IMAGE_RGB=8,   // 3 channels: Red Green Blue
    TYPE_IMAGE_XYZ=9,   // 3 channels, X Y Z channels in absolute cd/m2
    TYPE_IMAGE_XYZR=10, // 4 channels, X Y Z channels in absolute cd/m2, R: rod channels
    TYPE_IMAGE_LMS=11,  // 3 channels, L (long), M (medium), S (short) cone response
    TYPE_IMAGE_LMSR=12, // 4 channels, L (long), M (medium), S (short) cone response and R (rods) rod response
    TYPE_IMAGE_Lab=13,  // from CIE (Commmission Internationale de l'Eclairage), L (Luminance), a (contrast Red/Green), b (contrast blue/Yellow)
    TYPE_IMAGE_AC1C2=14, // 3 channels, antagonist color space: A (Achromatic), C1 (contrast Red/Green), C2 (contrast blue/Yellow)

    TYPE_IMAGE_JND1=15, // 1 channel = 1 map for JNDmetrix result, Just Noticeable Differences
    TYPE_IMAGE_JND3=16, // 3 channels = 3 maps for JNDmetrix result, Just Noticeable Differences
    TYPE_IMAGE_ROC=17,  // output of the Channelized Hotelling Observer: ROC curve  (contains X and Y component)

    TYPE_IMAGE_YUV=18,  // 3 channels: Y U V
    TYPE_UNSIGNED_SHORT=19, // 1 channel: Gray 16bit integer
    TYPE_UNSIGNED_INT=20, // 1 channel: Gray 32 bit integer
};
```

ComponentHeader: m_elementSize

- m_elementSize = sizeof(ComponentType)

- Number of bytes for ONE pixel of the component

```
int m_elemSize;
```

ComponentHeader: image size

```
unsigned long m_nbRowElemPerFrame; //!< height
```

```
unsigned long m_nbColumnElemPerFrame; //!< width
```

ComponentHeader: length

- length = number_of_elements * sizeof(element_type)
- Number of bytes of data in the component

```
unsigned long m_length;
```

ComponentHeader: number of bits

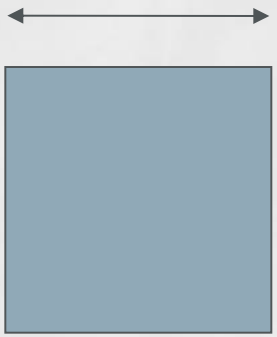
```
unsigned int m_nbBits; //!< 8 10 12 ...
```

For digital data represented by integers, the number of significant bits really used.

E.g. data could be represented in 10 bits (0..1024) but needs to be stored in 16 bits in the component (TYPE_UNSIGNED_SHORT)

ComponentHeader: element size column

- = element-pixel size in mm



```
float m_elemSizeCol; //!< in mm
```

ComponentHeader: element size row

- = element-pixel size in mm

```
float m_elemSizeRow; //!< in mm
```

ComponentHeader: illum

- enum ILLUMINANT

```
/*!
 * \enum ILLUMINANT
 * \brief color of the white point
 *        Not used in the virtual image capture part
 *        Used in the virtual display part
 *        Used in the virtual observer part
 */
enum ILLUMINANT {
    no_illuminant=0,
    A=1,
    B=2,
    C=3,
    D50=4,
    D55=5,
    D65=6,
    D75=7,
    F2=8,
    F7=9,
    F11=10
};
```

- Describes the illuminant of the scene for an observer study
- http://en.wikipedia.org/wiki/Standard_illuminant for more details
- In method: CreateComponent, choice:
  - **enum**
  - **char***

ComponentHeader: white_point

- In cd/m²

```
float m_whitePt[3]; //!< Xo, Yo, Zo in cd/m2
```

ComponentHeader: obs

- enum OBSERVER

```
/*!
 * \enum OBSERVER
 * \brief Observer parameter defined by the CIE 31 or 76, 2 deg. or 10 deg. in the fovea
 *        Not used in the virtual image capture part
 *        Not used in the virtual display part
 *        Used in the virtual observer part
 */
enum OBSERVER {
    no_observer=0,
    deg_2=2,
    deg_10=10
};
```

- Describes the "observer" used for the study
- http://en.wikipedia.org/wiki/CIE_1931_color_space#The_CIE_standard_observer for more details
- In method: CreateComponent, choice:
  - **enum**
  - **char***

ComponentHeader: unit

- enum UNIT

```
enum UNIT {
    no_unit=0,
    cd_m2=1,
    contrast_ratio=2,
    video_level=3,
    kelvin=4,
    degrees_celsius=5,
    percent=6,
    JND=7
};
```

- In method: CreateComponent, choice:
  - **enum**
  - **char***

Load and save a container from a file

Save

```
display_container->CreateComponent(height,width,TYPE_IMAGE_XYZ);
int number_of_bits=8;
display_container->SetComponentNumberOfBits(2,(int *) (&number_of_bits));
float xyz[3]={0};

for(i=0;i<height;i++)
    for(j=0;j<width;j++) {
        xyz[0]=i;
        xyz[1]=j;
        xyz[2]=i+j;
        display_container->SetComponentElement(2,i*width+j,(void *) &xyz);
    }
display_container->SaveToFile("NewContainer.bin");
delete display_container;
```

Load

```
DSContainer* display_container = new DSContainer();
display_container->LoadFromFile("containerRGB_XYZ_noisy.bin");
```

Get/Set: accessors to component properties from the Container

- Get/Set

```
/*!
 * \fn GetComponentAllocatorTyp
 * \brief accessor for getting the allocator type
 * \param nbComp: the number of the component
 * \param atype: output parameter with the allocator type of the component data
 * \return int: 0 if ok, -1 otherwise
 */
int GetComponentAllocatorType(int nbComp, enum AllocatorType& atype);

/*!
 * \fn GetComponentDescription
 * \brief accessor for getting the component description
 * \param nbComp: the number of the component
 * \param description: the description of the component to get
 * \return int: 0 if ok, -1 otherwise
 */
int GetComponentDescription(int nbComp, std::string& description);

/*!
 * \fn SetComponentDescription
 * \brief accessor for setting the component description
 * \param nbComp: the number of the component
 * \param description: the description of the component to set
 * \return int: 0 if ok, -1 otherwise
 */
int SetComponentDescription(int nbComp, const std::string& description);

/*!
 * \fn GetComponentElementType
 * \brief accessor for getting the component element type
 * \param nbComp: the number of the component
 * \param element_type: the element type to get
 * \return int: 0 if ok, -1 otherwise
 */
int GetComponentElementType(int nbComp, enum ComponentType* element_type);

/*!
 * \fn GetComponentElement
 * \brief accessor for getting a component element
 * \param nbComp: the number of the component
 * \param nbElem: element number to set
 * \param nbFrame: the number of the frame
 * \param element: the element to get
 * \return int: 0 if ok, -1 otherwise
 */
int GetComponentElement(int nbComp, int nbElement, int nbFrame, void* element);

/*!
 * \fn SetComponentElement
 * \brief accessor for setting the value of a component element
 * \param nbComp: the number of the component
 * \param nbElem: element number to set
 * \param nbFrame: the number of the frame
 * \param element: the element value to set
 * \return int: 0 if ok, -1 otherwise
 */
int SetComponentElement(int nbComp, int nbElem, unsigned int nbFrame, void* element);

/*!
 * \fn GetComponentNumberOfBits
 * \brief accessor for getting the number of bits of a compenent
 * \param nbComp: the number of the component
 * \param nbBits: the number of bits to get
 * \return int: 0 if ok, -1 otherwise
 */
int GetComponentNumberOfBits(int nbComp, unsigned int* nbBits);

/*!
 * \fn SetComponentNumberOfBits
 * \brief accessor for setting the value of number of bits of a component
 * \param nbComp: the number of the component
 * \param nbBits: the number of bits to set
 * \return int: 0 if ok, -1 otherwise
 */
int SetComponentNumberOfBits(int nbComp, unsigned int* nbBits);

/*!
 * \fn GetComponentNumberOfRowElementsPerFrame
 * \brief accessor for getting the height of a component (number of rows per frame)
 * \param nbComp: the number of the component
 * \param nbRowElems: the number of row elements per frame to get (height)
 * \return int: 0 if ok, -1 otherwise
 */
int GetComponentNumberOfRowElementsPerFrame(int nbComp, unsigned long * nbRowElems);
```

6. Writing a Module

Writing a module

- Base class

```
#ifndef MODULE_H
#define MODULE_H

#include <vector>

#include "DSContainer.h"
#include <DataExchanger/UtilitiesExchanger.h>
#include <DataExchanger/ContainerExchanger.h>
#include <ErrorManagement/MevicLogger.h>

const double pi = 3.14159265358979323846;

/*!
 * \class Module
 * \brief This class is the core of the modules chain
 *        This class can not be instanciated because it is an abstract class
 */
class Module
{
public:

    /*!
    *  Default constructor of the class Module
    */
    Module();

    /*!
     * \fn SetUtilitiesExchanger
     * \brief add by EMPEC for data exchange management
     * \param pUtilitiesExchanger: The UtilitiesExchanger (pointer) to set in the module
     * \return void
     */
    void SetUtilitiesExchanger(UtilitiesExchanger * pUtilitiesExchanger);

    /*!
     * \fn SetContainerExchanger
     * \brief assign a ContainerExchanger with this module
     * \param pContainerExchanger: The ContainerExchanger (pointer) to set in the module
     * \return void
     */
    void SetContainerExchanger(ContainerExchanger * pContainerExchanger);

    /*!
     * \fn SetMevicLogger
     * \brief Assign a MevicLogger with this module
     * \param mevicLogger*: pointer to the mevic logger object
     */
    void SetMevicLogger(MevicLogger * mevicLogger);

    //virtual functions
    /*!
     * \fn SetParameter
     * \brief virtual function for the inherited classes for setting the parameter values
     *                from the xml
     * \param name: name of the pointer (string)
     * \param  value: value of the parameter (string)
     */
    virtual void SetParameter(const std::string & name, const std::string & value )=0;

    /*!
     * \fn Simulate
     * \brief virtual function for the inherited classes for running the simulation of one module
     * \param container: vector containing the containers of the pipeline simulation
     */
    virtual void Simulate(std::vector<DSContainer*>& container )=0;

    /**
     *  Default destructor of the class Module
     */
    virtual ~Module();

private:

    // Copy Constructor (used to create an object from an existing one)
    Module(const Module& oneModule);

protected:

    bool m_debug; //!< for debug mode

    UtilitiesExchanger * m_pUtilitiesExchanger; //!< utility exchanger

    ContainerExchanger * m_pContainerExchanger; //!< container exchanger

    MevicLogger * m_pMevicLogger; //!< mevic logger for the error management
};

#endif // MODULE_H
```

set parameters

Run module

Example: ReaderModule

```cpp
#include <Module.h>

#ifndef READERMODULE_H_
#define READERMODULE_H_

/*!
 * \class class ReaderModule
 * \brief This class allows to read a complete container
 *        in a binary file, and to store it in the chain
 */
class ReaderModule : public Module
{

public:

    /*!
     * Constructor of the class.
     */
    ReaderModule();

    /*!
     * \fn Simulate
     * \brief Opens a container file and adds it to containers
     *        Main function for the simulation
     * \param list: list of containers to process
     * \return void
     */
    void Simulate(std::vector<DSContainer*>& containers);

    /*!
     * \fn SetParameter
     * \brief allowing to set the value of a class parameter
     * \param name:  name of the parameter
     * \param value: value of the parameter, string type
     * \return void
     */
    void SetParameter(const std::string & name, const std::string & value );

    /*!
     * Destructor of the class.
     */
    ~ReaderModule();

private:
    std::string m_filename; //!< filename of the container file to read
    bool m_compression; //!< condition: 1 if the file to read is compressed, 0 otherwise
};

#endif /* READERMODULE_H_ */
```

Derived from Module

Same interface as Module

Example: ReaderModule

```
void ReaderModule::SetParameter(const std::string & name, const std::string & value )
{
    string sName  = boost::to_lower_copy(name);
    if(sName == "filename")
    {
        m_filename = value;
    }
    else if(sName == "compression")
    {
        try
        {
            m_compression = boost::lexical_cast<bool>(value);
        }
        catch(boost::bad_lexical_cast e)
        {
            m_pMevicLogger->logError("ReaderModule::SetParameter: the parameter provided in \"Compression\" cannot be casted to bool");
        }
    }
    else
    {
        m_pMevicLogger->logError("ReaderModule::SetParameter: Invalid parameter name");
    }
}
```