Data sharing meeting

UMEMOTO

Purpose of todays meeting

- Show our progress about data sharing
- Discuss about it
- Also I would like to know your system after last meeting

Purpose of data sharing

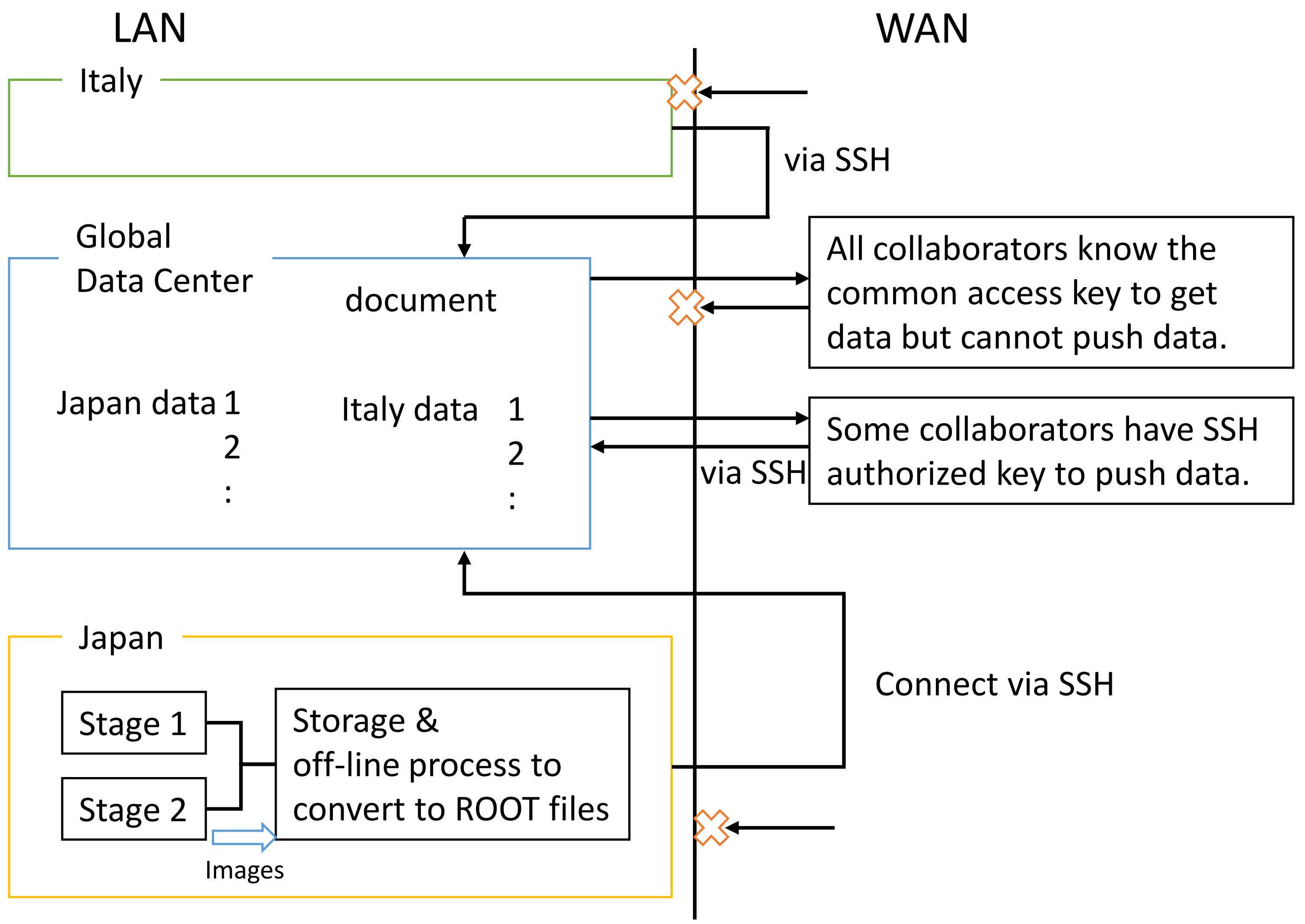
- We would like to share the scanning data (Elliptical parameter and image) for other collaborators who don’t have scanning system

  (yandex and Turkey people can analyze our data)

- Need workstations to analyze scanned data and database to save and keep run data

- Then the data format should be useful for all

Network and storage diagram (yoshimoto slide)



Idea

- Scanning information
  - RunID, PlateID, etc.
  - Chamber structure
  - Emulsion
  - Readout
  - Comment

$\Rightarrow$ Json file

- Parameter and image
  - View Header
  - Cluster
  - Grain
  - MicroTrack
  - Images

$\Rightarrow$ ROOT file

Root file

- Elliptical Parameter

It is easy to set these parameter in ROOT file

Please show the file “param2.root”

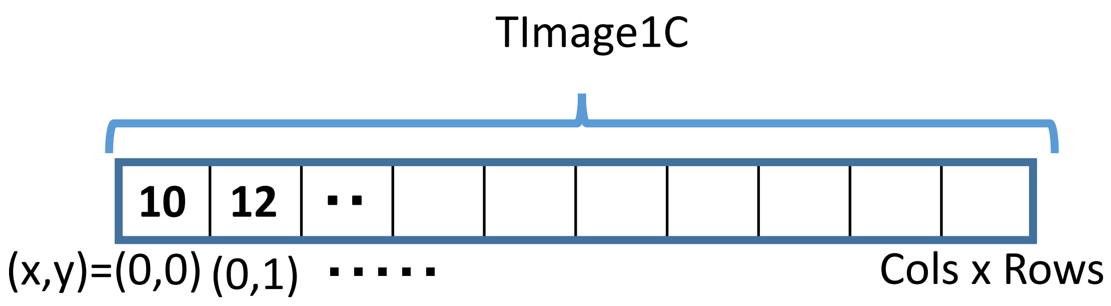
- Optical image

It is difficult to set image file (.bmp) in root file by usual ROOT class

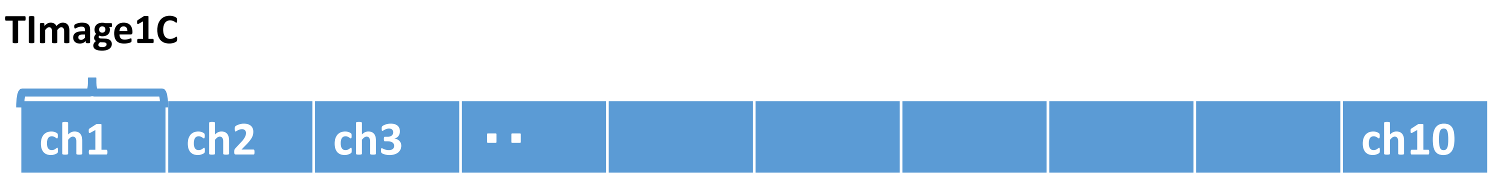
We use OpenCV library for image processing so we would like to take image file in and out with OpenCV format

TImageMC

- We make a new root class named “TImageMC” to keep image file in root file
  - inside of TImageMC, also make other 2 class (TArrayUC and TImage1C)
  - put image brightness to TArrayUC



- We also put Timage1C in TClonesArray
  - using TClonesArray, can get a semi permanent virtual memory (not remake)
  - can get and put image with this memory
  - we can put multi channel image(for color or defocus event cluster)



compatibility

- Can use OpenCV format image
- Can use root library
- In Italy, this is OK or not?

Summary

- We study about data sharing format
- It is more difficult to put image file to root file
- Made ROOT library for Japanese analysis

Backup

http://emulsion.na.infn.it/svn/DMDS/dm2root/README.txt

dm2root - project dedicated to data exchange between the scanning laboratories using root data format

The library libDMRoot containes definitions for all basic structures:

1. DMRClusters - 2-dimensional cluster object with position and shape information
2. DMRGrain - 3-dimensional grain object with position and shape information
3. DMRMicrotrack - microtrack object
4. DMRImageCl - image object associated with cluster
5. DMRFrame - image object associated with frame

Expected usage is the following:

- for each scanning software should be prepared the own converter for scanned data writing into the root tree using the above structures
- - the resulting root files may be processed and analyzed by common scrips or programs on any operation system where root is installed (Linux, Windows, etc)

Application "dmrun" in src/appl/examples shows how the writing part of the interface works. In this example file run.dm.root with a tree Vdmr inside is created Each entry of this tree containes data for one microscope view (i.e. volume scanned by vertical movement when X,Y are fixed).

In each view we keep several arrays corresponding to different tree branches:

cl - array of DMRCluster objects
gr - array of DMRGrain
mt - array of DMRMicrotrack
im - array of DMRImageCl
fr - array of DMRFrame

There is also a branch for a view header hd - with the general information about this view (coordinates, etc)
The advantage of this structure that it's relatively simple and convenient for interactive analysis using root - all variables are directly accessible from the root command line (as for example: Vdmr->Draw("cl.npx"))
Relations between objects located in different branches implemented using indexes. The converters should take care about correct indexes filling.

In the script src/macros/check_image.C is the example of access to the image information. Note that the images can be saved or not depending on the necessity. This is valid in general for any branches and fields - if some information is not available it's not necessary to fill it with a dummy values - the writing part will work normally anyway. Of cause if the essential information is missing the analysis may become limited, so it's better to provide complete data.

```
class DMRCluster : public TObject
{
public:
                UInt_t id;
                UInt_t flags;
                Float_t x,y,z; // coordinates
                Float_t lx,ly,phi;                // elliptical fit: major, minor, direction
                Float_t gh,gb,gq;                 // gaussian fit: height, bg, quality
                UInt_t npx;                        // number of pixels
                UInt_t vol;    // total volume
                Float_t pol;   // polarization angle

                Int_t img;                        // index of corresponding image
                Int_t ifr;                        // owing frame index
                Int_t igr;                        // owing grain index
                Int_t imt;                        // owing microtrack index

                Int_t is_nt;   // 1 if this cluster is a nanotrack
```