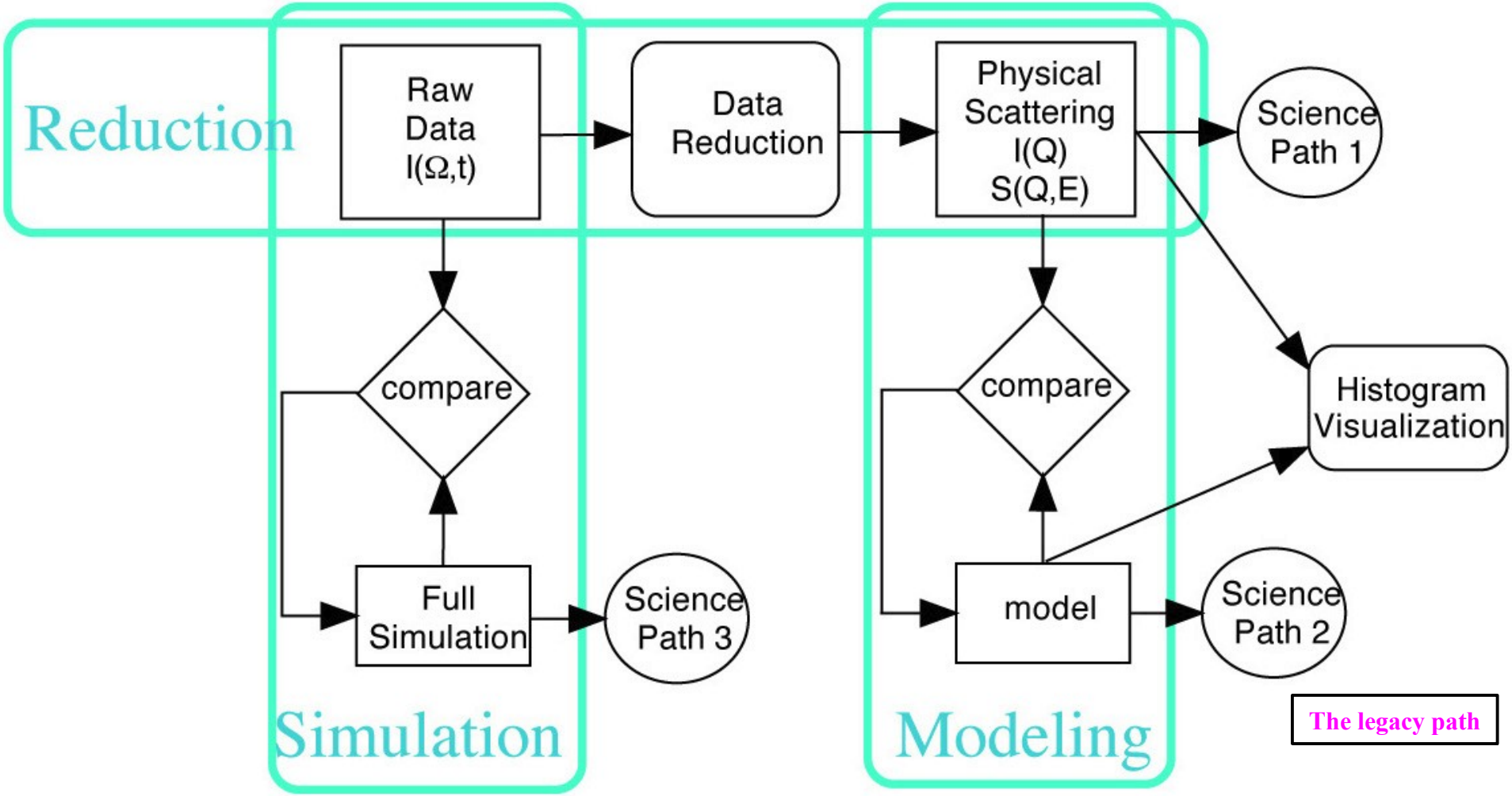
Standards for data analysis software

NMI3-II WP6

ILL, ISIS, PSI, FRM2, JCNS GKSS HZB ESS

Science path

A reminder about terminology [Ref: DANSE]



Our tasks and resources

For the first time, we have resources from EU to gather our knowledge and strength.

Our tasks:

Task .1: Review existing data analysis software and practices of software developers
Task .2: Review existing solutions for a common data analysis infrastructure
Task .3: Develop prototype software in chosen solution for representative applications
Task .4: Evaluate prototype software

Our resources:

- 4 months of each of the other participants (that is about 2-3 days per month for each of us).
- 30 months position funding.
- Our smiles.

Memories from the Past

Success and failure

Any 'new' project should start by an evaluation from past attempts:

- LAMP
- Gumtree
- DANSE (PDFgui, SANSView,
- Horace/Mslice, Fullprof, SANSView/SASFit, PDFgui, GenX, McStas, ResTrax, Vitess, vTas, Isaw, Frida, …
- Mantid
- DAVE
- ROOT (CERN)
- … (and too many others)

What is ***"good, bad and ugly"*** in these ?
Do we need to re-invent the wheel ?
How to optimize our investment ?
Any initiative should start by a review.

Dead software: warning

There is only **'old' software** around when a new one starts.

Missing collaboration brings **single-point of failure**.
The development team must be of at least 2 people on every project.

User community is a good collaboration scheme. It also minimises maintenance, ensures long life-time and gives credit.

Most dead software are limited in size (less than 10 kLOC) and are relatively easy to refactor or include.

Many dead software are **not available** any more. Some do not compile or install (VMS ).

In order to decide on what to do, we should analyse what makes a 'good' code, so we know **what not to do**.

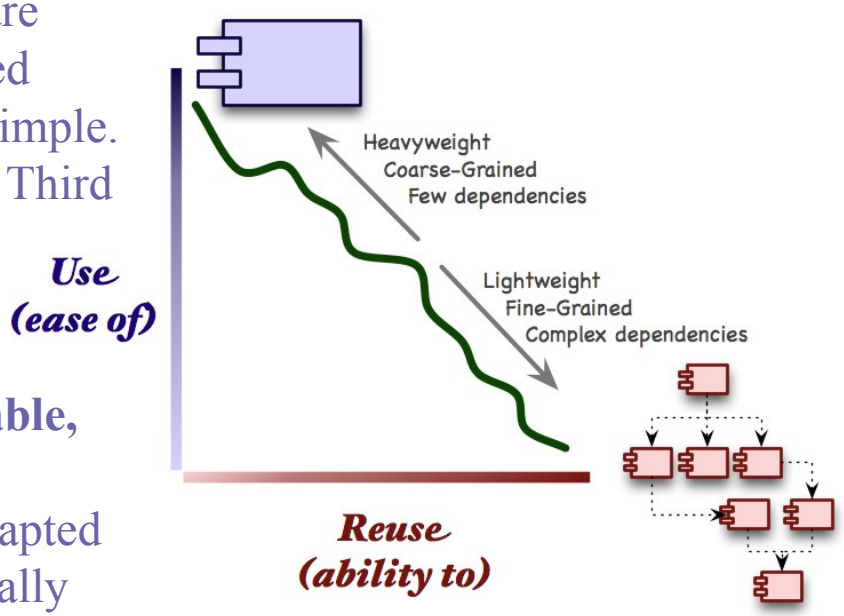
Code granularity

Maximizing reuse complicates use

[Ref: Clemens Szyperski]

Granularity: Coarse-grained components are easier to use, but fine-grained components are more reusable.
Coarse-grained components (e.g. integrated apps.) are easier to use and have more features, but fine-grained components (many objects) are more reusable and simple. In practice, very few 'low-level' objects are re-used. Third party libraries are fine-grained, and introduce dependencies.

Weight: Lightweight components are more reusable, but heavyweight components are easier to use.
lightweight components require to be configured/adapted to their environment. Heavyweight components usually contains their own configuration settings.



Code complexity, Lines of Code

Code complexity

- A project with many languages is harder to maintain.
- A greater number of components means that there are more places where the system can fail.

Lines of Code

- A programmer writes about 12 working LOC/day, but this efficiency is twice higher for small projects (100 kLOC).
- Total LOC in a project is a good indicator of software complexity.
- Higher level languages need less LOC per feature, and are easier to convert to lower level than vice-versa.
- A greater number of LOC has correlation with the number of bugs the software has.
- A single programmer can maintain 50-100 kLOC.
- 100 kLOC cost about 1M$ total.
- A good programmer is 20-25 times more efficient than a bad one.

Criteria for a 'good' project

We have limited resources, so we can estimate what is within reach.

- Software must use **high level language**, and not too many different.
- Software must remain **available** and **installable**.
- Software must **minimize** the number of **classes**.
- Software must **minimize** its **dependencies** (libs and external classes).
- Software must **minimize complexity**: think '*simple*' first.
- Concentrate on **science**, **ignore interfaces** as far as possible (too much work).
- **Unit testing** is essential to provide quality software.
- One person for 30 months → **10-50 kLOC** code maximum.

Suggested actions: Task 1

Review existing data analysis software and practices of software developers

Inquire about software usage (downloads/day and nb of users, unique features) to estimate if old codes must be maintained.

Test/analyse software:

- Ricardo will make a presentation about that.
- Jon will present Mantid
- Use e.g. the NMI3 LiveDVD to make-up your mind.

Objective: Build a table of 'recommended' software

Any other suggestion is welcome.

Suggested actions: Task 2

Review existing solutions for a common data analysis infrastructure

Common infrastructure could be:

- Documentation about the common practises to follow (define standards).
- Common interface layout (only guidelines as we won't code that yet).
- Common data format: NeXus seems unavoidable.
- Common workflow standards for function calls
- List of common low level functions that should be shared by all.
- Common web site to hold information, documents and code.
  See http://software.pan-data.eu
- Centralized development area (a 'forge') for the code, trac/tickets and documents.
  Could be http://software.nmi3.eu directing to e.g. GitHub or other SVN-repos.
- Common naming/terminology, e.g. *out=call(in, ...)*. We could envisage to have aliases.
- Start to define a common ontology to describe data processing, in the style of e.g. http://geneontology.org.

Suggested actions: Task 3

Develop prototype software in chosen solution for representative applications

Identify representative application(s)

- Should be reactor source oriented (as we mainly use reactors around this table, and Mantid does the job for spallation sources).
- Must correspond to a need (something new if possible)..
- **Proposed applications** (reactor): only data reduction
  - Multiplexed TAS (RITA/Flat-Cone style).
  - Powder or SX diffractometer.
  - … ?
- Whatever be the choice, should make use as much as possible of existing codes (Mantid/VATES, NeXus stuff, vTAS, LAMP, …).

Techno-Geek Collection 4 / 26

ELITE CODER KEYBOARD

[http://www.virtual-trading-cards.com]

Possible Technical solutions:

- Convert files into Mantid/NeXus format, compute S(q,w) from vTAS (java) and use VATES.
- Write a full set of Loader/Algorithm for Mantid (C++).
- Use IDL (LAMP, DAVE) or Matlab (mFit, iFit) to design an application.
- Write an independent application.