Software for the ILC: Simulation and reconstruction frameworks

Ties Behnke, DESY

Speaking for the DESY ILC software group

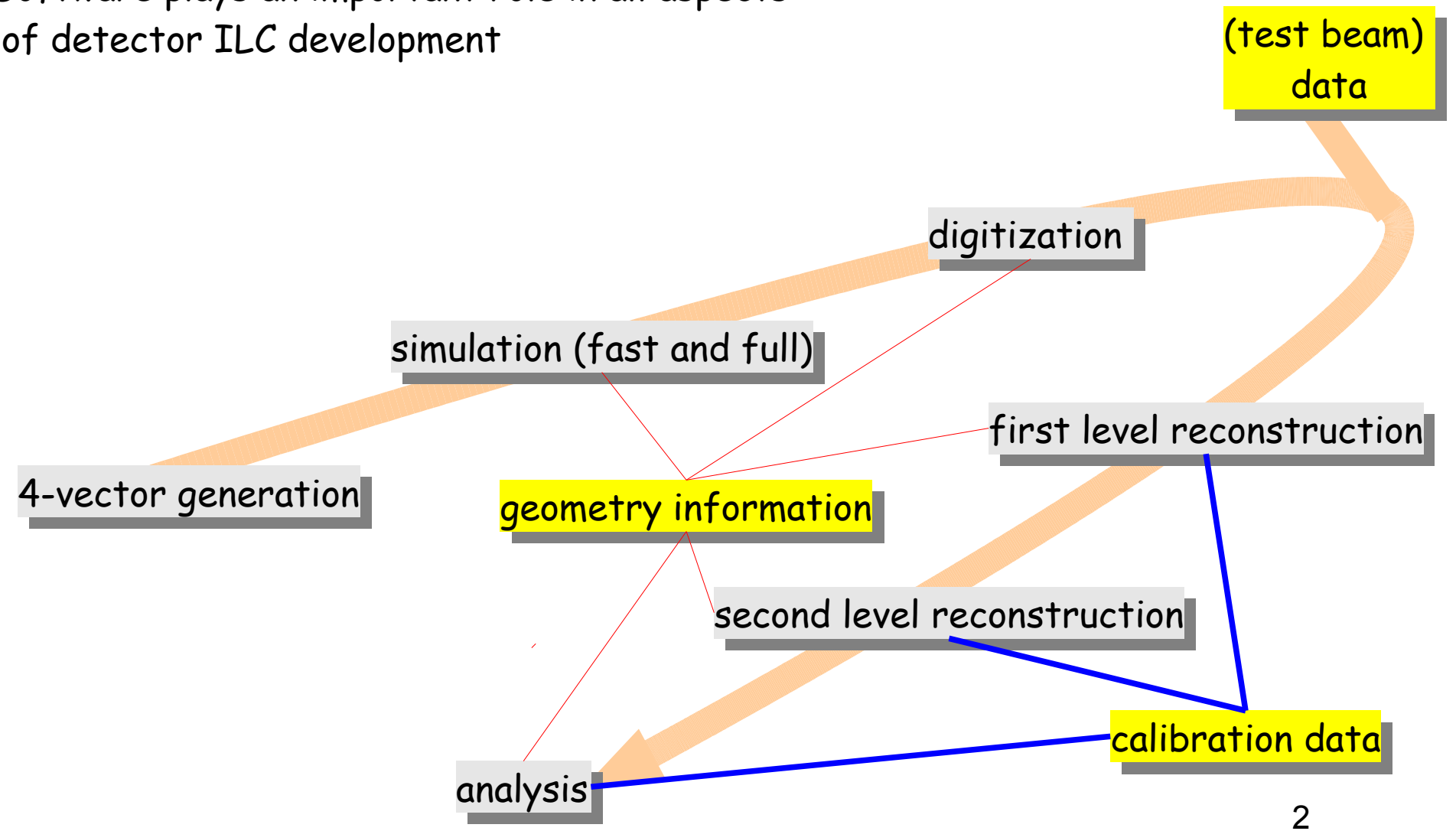
Software for the ILC: a brief review

Components of a software system at the ILC

Collaboration

Software for the ILC

Software plays an important role in all aspects of detector ILC development



The Software Situation

	Description	Detector	Language	IO-Format	Region
Simdet	fast Monte Carlo	TeslaTDR	Fortran	StdHep/LCIO	EU
SGV	fast Monte Carlo	simple Geometry, flexible	Fortran	None (LCIO)	EU
Lelaps	fast Monte Carlo	SiD, flexible	C++	SIO, LCIO	US
Mokka	full simulation – Geant4	TeslaTDR, LDC, flexible	C++	ASCI, LCIO	EU
Brahms-Sim	Geant3 – full simulation	TeslaTDR	Fortran	LCIO	EU
SLIC	full simulation – Geant4	SiD, flexible	C++	LCIO	US
LCDG4	full simulation – Geant4	SiD, flexible	C++	SIO, LCIO	US
Jupiter	full simulation – Geant4	JLD (GDL)	C++	Root (LCIO)	AS
Brahms-Reco	reconstruction framework (most complete)	TeslaTDR	Fortran	LCIO	EU
Marlin	reconstruction and analysis application framework	Flexible	C++	LCIO	EU
hep.lcd	reconstruction framework	SiD (flexible)	Java	SIO	US
org.lcsim	reconstruction framework (under development)	SiD (flexible)	Java	LCIO	US
Jupiter-Satelite	reconstruction and analysis	JLD (GDL)	C++	Root	AS
LCCD	Conditions Data Toolkit	All	C++	MySQL, LCIO	EU
GEAR	Geometry description	Flexible	C++ (Java?)	XML	EU
LCIO	Persistency and datamodel	All	Java, C++, Fortran	-	AS,EU,US
JAS3/WIRED	Analysis Tool / Event Display	All	Java	xml,stdhep, heprep,LCIO,	US,EU

The Problem

Many packages exist (Too many ...???)

Different languages
different authors
different philosophies

We are dublicating efforts on 50% of the needed functionality

and never get around to attack the other 50% beacuse of lack of personpower

My pledge: we have to work together more closely
we have to find ways to do that even though there are
deep and conceptual differences in the way people think

The solution

Design Criteria

ILC software has to live for a long time

Approval will take a few more years,
construction will take O(8) years
ILC will run for O(20) years

ILC software should be able to follow the project throughout the life cycle

Need to be able to follow IT developments

Have to avoid the pitfalls of another change in paradigm
(like Fortran to OO conversion)

A reminder: we are still not at the same level in OO as we were with the FORTRAN based system!

Functionality

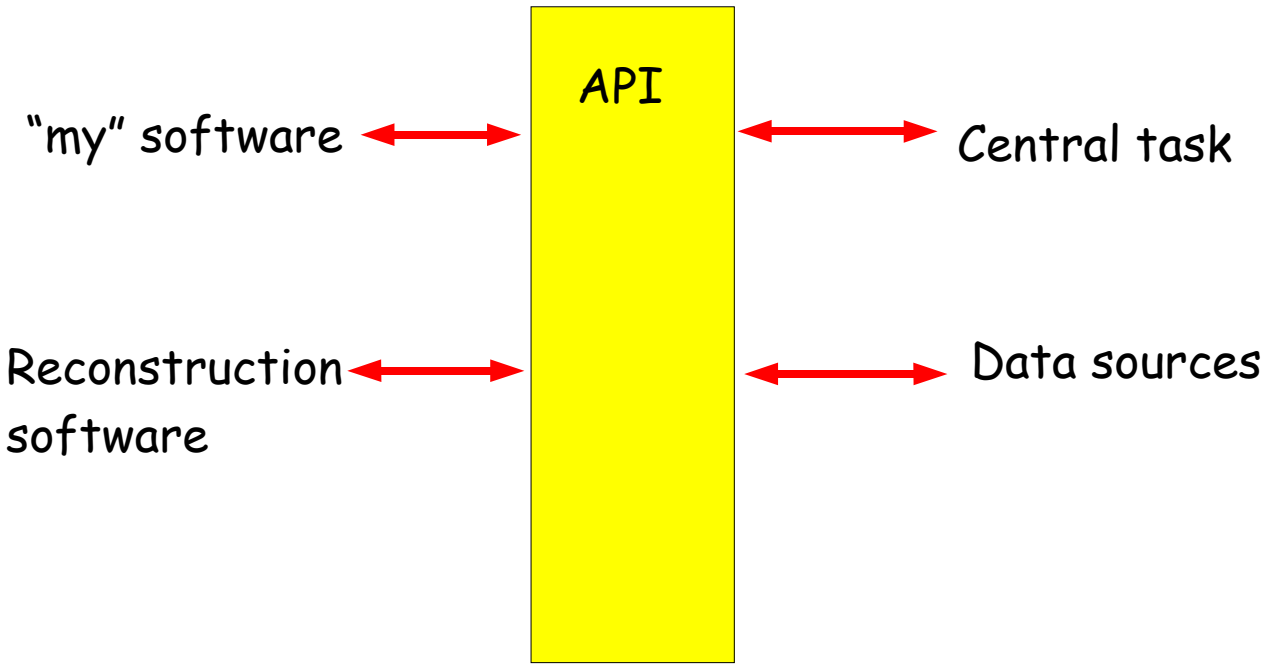
Long lifetime DOES NOT mean:

todays programs will be used tomorrow

But it means that todays software is designed with a view on the changes of tomorrow

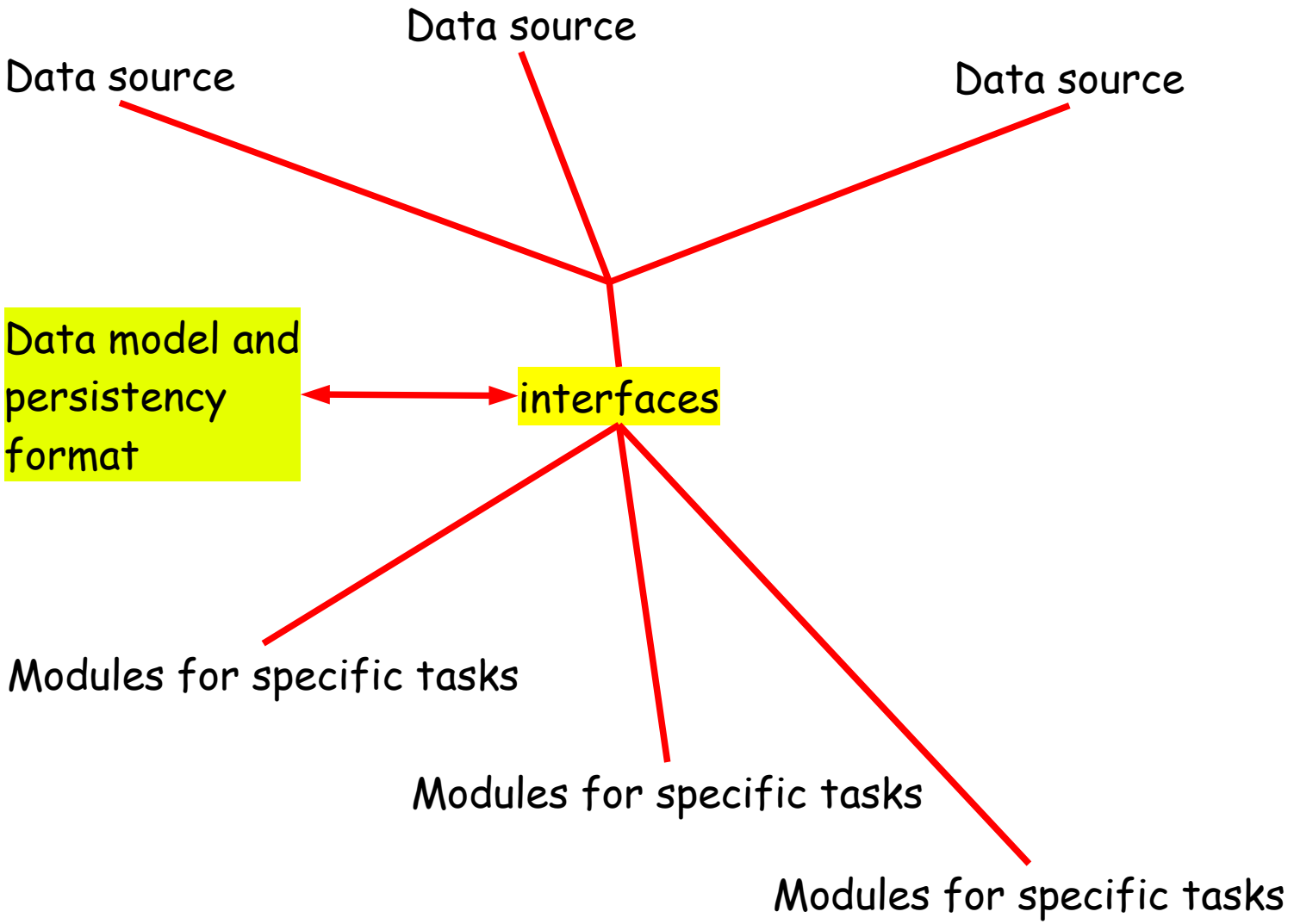
Try to avoid sudden breaks, try to maintain continuity

Modules and Interfaces



Communication between program and "data" only through well defined interfaces

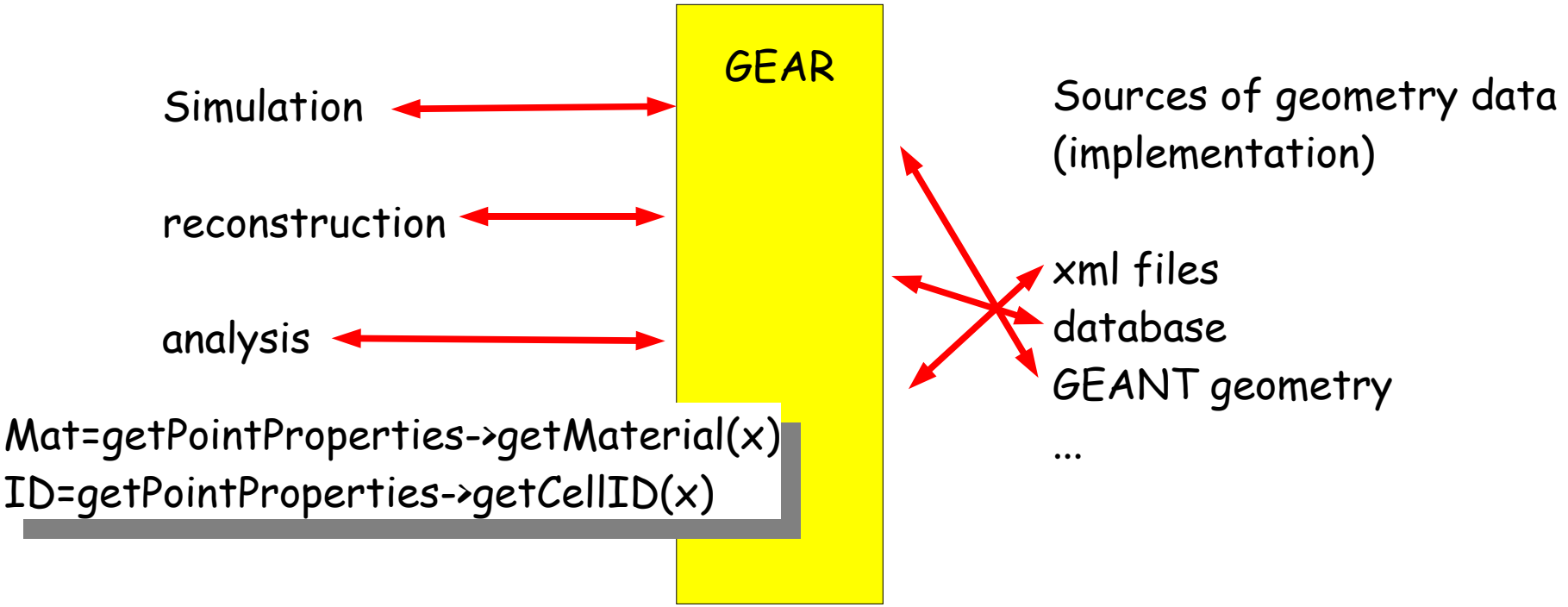
Framework Structure



Interfaces: Example

Geometry information is used in many places

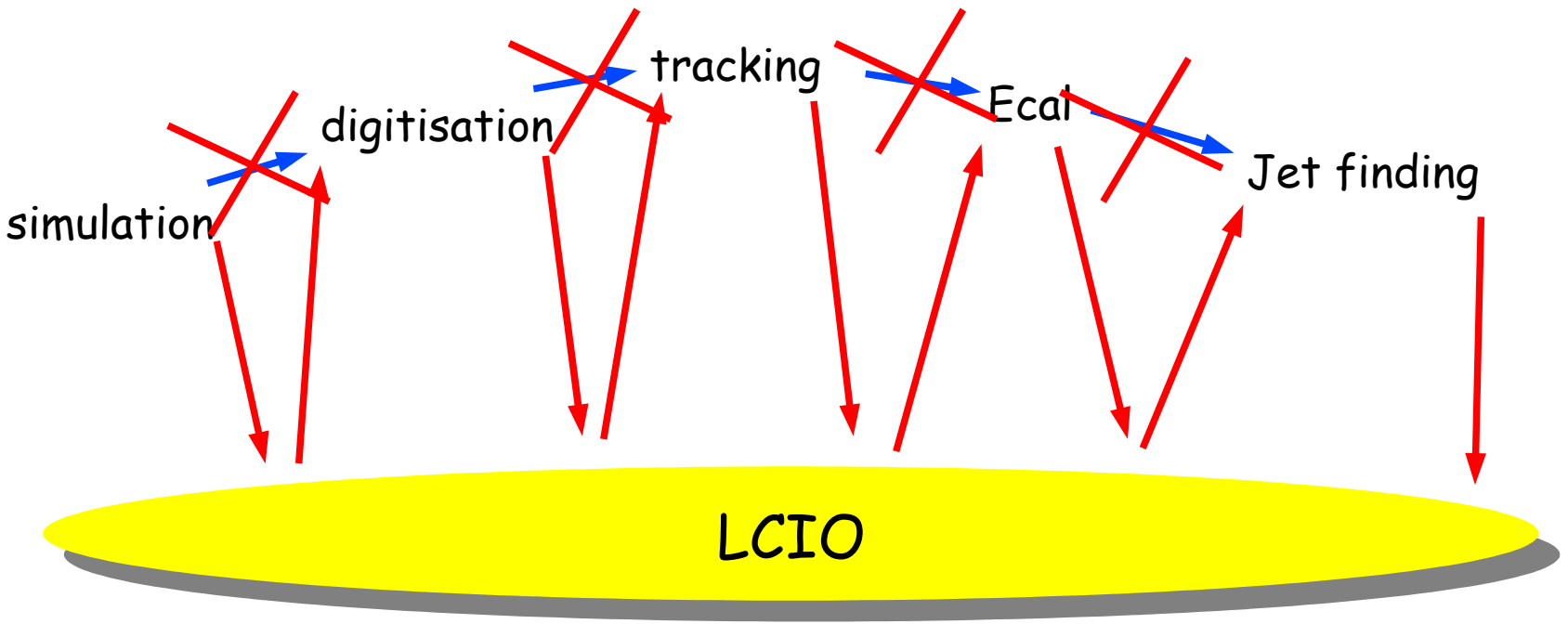
- Very detailed, but local: simulation
- Less detailed, but know surroundings: reconstruction
- Little detail: e.g. Event display



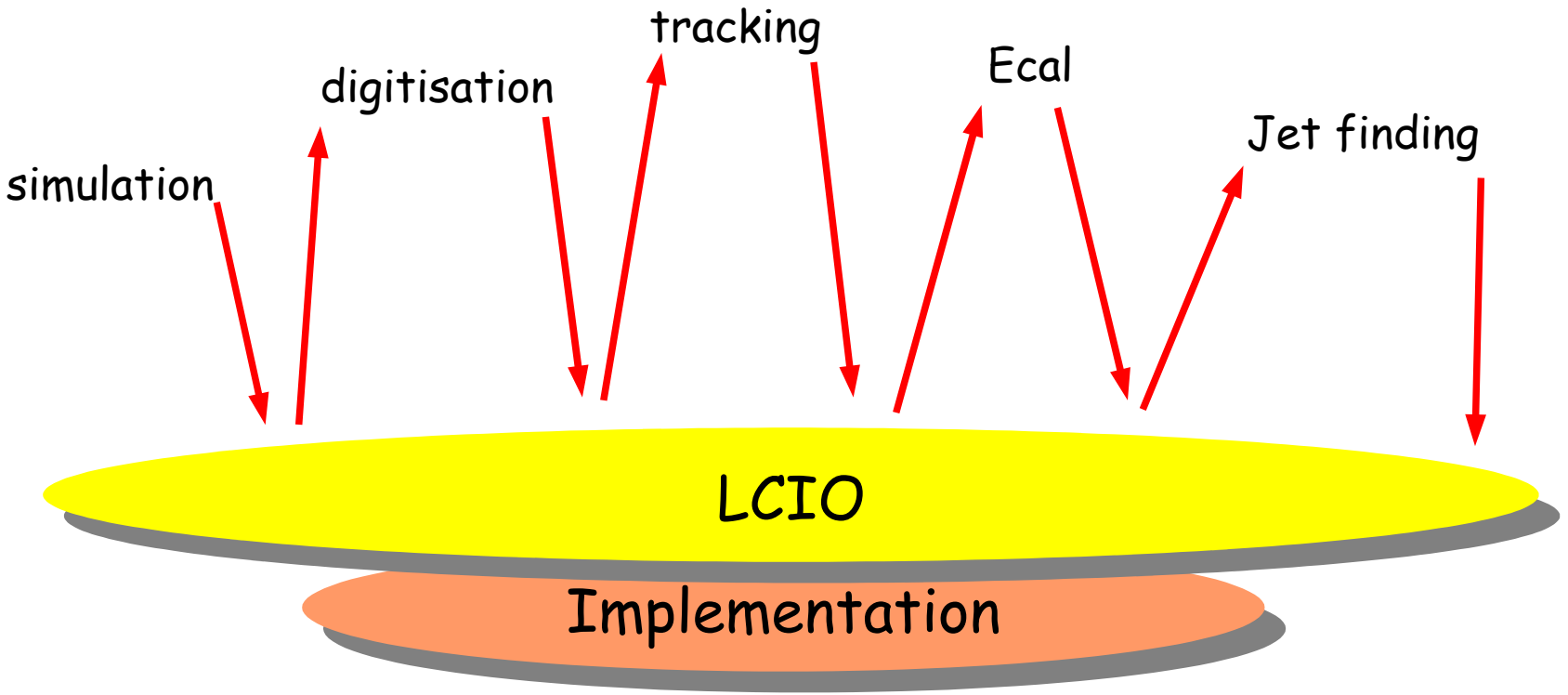
Software Structure

The "old" way

Software Structure



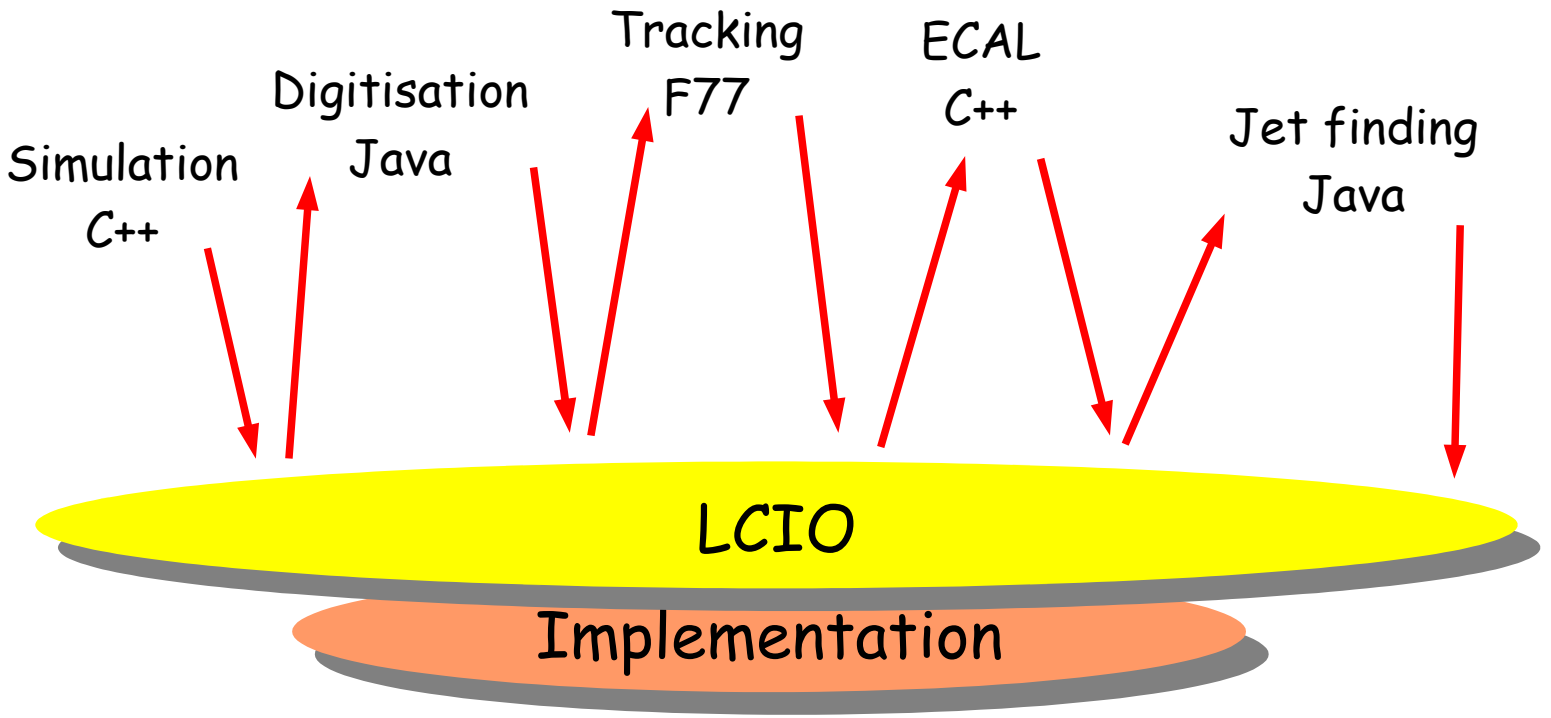
Software Structure



Common event model allows fundamental modularisation of software

Multi Language Support

Design around common event model:
allows multi language support eventually



Condition: the same event can be accessed through different languages
Compling/ Linking etc has to be solved

Interfaces in use

Heavy reliance on the definition of interfaces

Example: LCIO is primarily an interface
AIDA is an histogramming interface
GEAR is a geometry interface
.....

- Independent from the implementation (SIO in LCIO, ROOT in AIDA, ...)
- Software remains portable and adaptable
- Scales with the number of systems and complexities

Practical implications

Interfaces need to be defined:

Significant amount of work
usually defined interfaces do not answer your immediate needs...

Interfaces have to be accepted by the developers and users

Saving are not immediatly apparent
often it is seen as restrictive and slowing down the work

"I need to get this information f[illegible] processor A to processor B, therefore I created a stati[illegible]ss ....."
"I cannot be bothered [illegible] dealing with the interface, is slows down my work..."

Its ALL or NOTHING!!

Current Situation: LCIO

LCIO: the linear collider Input Output format

Widely accepted, used by US and EU software frameworks
supported in part by Asian framework

provides a basic foundation for software at the ILC and
exchanges of software

but it is based on a outdated implementation (SIO) at the moment

LCIO has been a very good example for a interface which works

Current State: GEAR

GEAR: geometry interface

Provide interfaces to access geometry information in reconstruction and analysis

well defined access functions,

implementation is hidden from the user (at the moment XML + GEANT4)

If used more widely and further developed GEAR promises to significantly ease the porting of software between different detectors

Current State: AIDA

Histogramming interface

jointly developed by SLAC and CERN

Provides all histogramming etc functionality independent of a particular system (like JAS, ROOT, PAW, ...)

Currenctly implementations exist in Java, C++ (XML output files) and (soon) Root

If used properly histogramming becomes independent of the presenter and analysis system.

Software Framework

Framework provides minimal functionality

- Event loop
- Steering mechanism
- Possibly some logging capability

Functionality in framework comes from individual modules (standalone)

Communication to the outside happens only through defined interfaces

Central software should be as light-weight as possible
ease of installation
ease of maintenance
the work should go into the definition of proper interfaces

MARLIN

MARLIN (see talk by O.Wendt)

is one such framework

But if interfaces are used systematically frameworks are exchangable,
modules can be swapped
languages matter much less (ideally)

But MARLIN is just one example

If Jupiter and friends are based on the same model, exchange is possible of the functionality

Conclusion

Noone will object: Software plays a central role in ILC developments

A highly modular ansatz with well defined API's (interfaces) is (in my opinion) the most promising way to better and more common software

Whatever we do, we should avoid to tie ourselves too closely to particular implementations (like root, like JAS, like SIO...) to be able to follow developments in IT

There should be more collaboration on the definition of the API's