

Plans for EDNA

Olof Svensson
EDNA Project Manager
ESRF

What is **EDNA**?

- **EDNA** is the name of a new collaboration which has for aim an automatic collection of data on synchrotron beamlines.
- **EDNA** was initially called **DNA 2.0** but we decided to change the name for many reasons:
 - **EDNA** is a new collaboration, not a continuation of **DNA**. No code developed for **DNA** will be used by **EDNA**.
 - **EDNA** will have a project agreement and a license (LGPL). Code of the **EDNA** core will be shared, i.e. no single developer will own a **EDNA** core module.
 - **EDNA** has a management structure with a project manager and a technical manager.
 - **EDNA** is designed to be modular. New functionality should be easy to implement.
- **EDNA** stands for **E**nhanced **a**utomated **D** collection **N** of data **A** or (**E**nhanced **DNA**)

Outline

- **DNA 1.1** – current status
- **EDNA** goals
- Summary of **EDNA** developments
 - The first spike, June 5th meeting
 - Use Cases
 - Project Agreement, October 23rd meeting
 - Skeleton, November 21st - 22nd workshop
- **EDNA** Planning
 - MOSFLM indexing plugin – deadline end of December 2007
 - Prototype implementing the “Characterisation taking into account radiation damage” Use Case – deadline June 2008

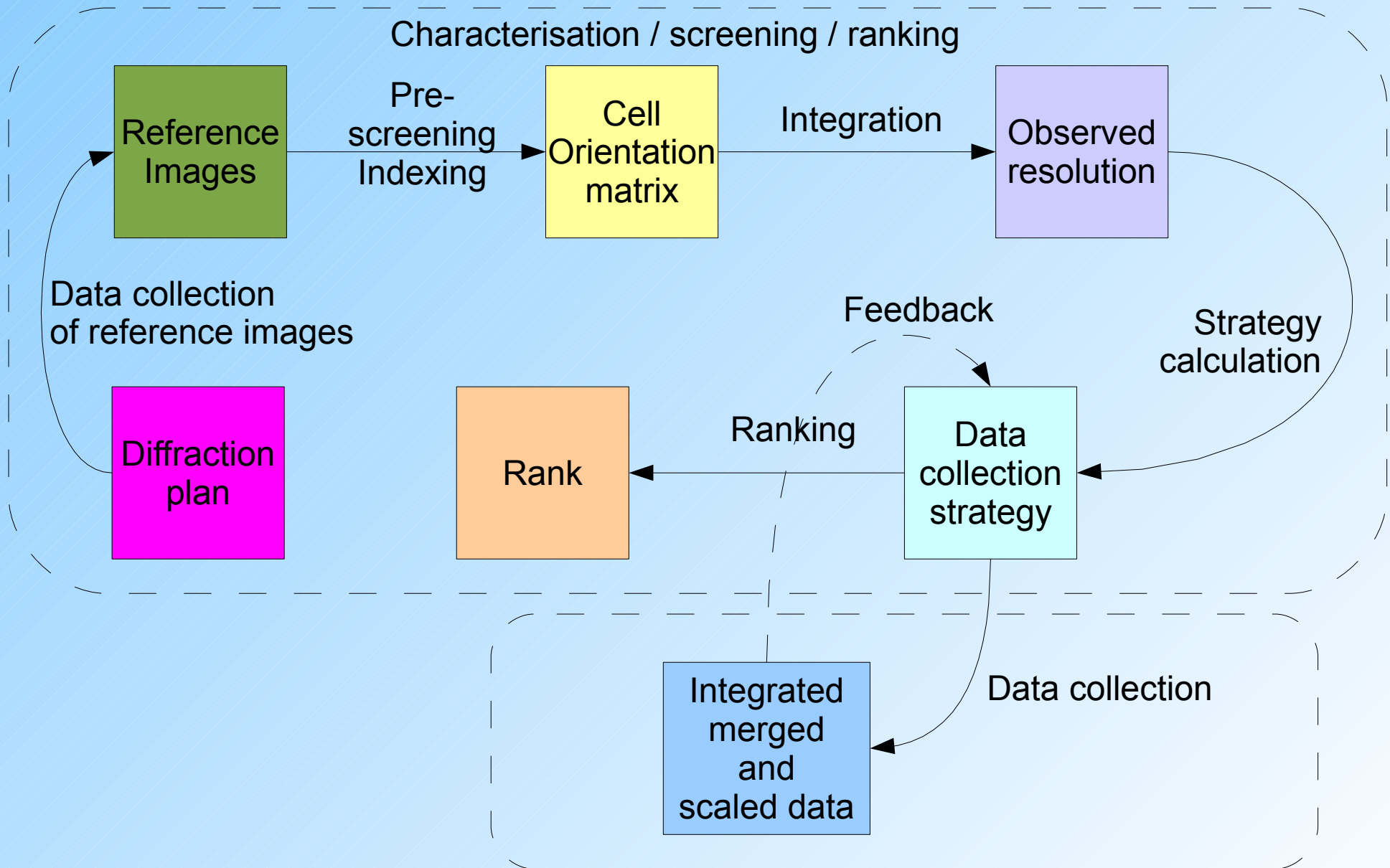
DNA 1.1 Status

- Recent developments for DNA 1.1:
 - Many bug fixes during the Spring and Summer
 - New MOSFLM version 7.0.1 of August 20th 2007
 - New BEST version 3.1.0d of July 16th 2007
 - Better integration with ISPyB – DNA results stored in archive file system
- Current version installed on ESRF beamlines working well – now many users use DNA
- Some of DNA 1.1 shortcomings :
 - Slow implementation of new features
 - Max exposure time before radiation damage set per beamline – it should be possible for the user to modify
 - Rare: Indexing fails even when indexing manually works with MOSFLM
 - Integration and scaling works but can be improved
 - Not possible to change resolution between screening and automatic data collection

EDNA goals

- **Robust :**
 - EDNA should never crash
 - EDNA should solve scientific problems, i.e. if a pattern can be indexed EDNA should be able to index it.
- **Modular :**
 - Implementation of new scientific features should be easy
 - New features should not make EDNA unstable
- **Grid enabled :**
 - EDNA should be able to use distributed processors in order to increase throughput
- **Collaborative :**
 - New developers should be able to join rapidly the collaboration
 - Development practice like code style and code review will help make the code base homogeneous and easy to understand for new developers

DNA 1.1 workflow



The first spike, June 5th meeting

- Test of frameworks:
 - AALib was retained
 - CCPN was not retained
- BEST module developed
 - Successful execution of the BEST program
 - Many problems during data model development
- Conclusions:
 - Project agreement to be signed in an October meeting
 - Use Cases needed
 - Prototype by June 2008

EDNA Use Cases

- The EDNA Use Cases serves several purposes:
 - Tool for communication between scientists and developers – do the developers have all the information needed for the development?
 - Tool for communication between scientists – do scientists agree with what EDNA should do?
 - Documentation for new developers
 - Documentation for EDNA users
- Problems between the first spike and the October 23rd meeting :
 - Use Cases much more time consuming to develop than anticipated
 - No clear view on how to go from a Use Case to an implementation

October 23rd meeting

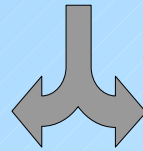
- Project Agreement :
 - Project agreement discussed and improved
 - Proposal for shared code licence for EDNA core
 - Now in the process of being finalised
 - To be signed by collaboration facilities / laboratories directors
- Planning of developments :
 - Pre-screening use case dropped for the moment
 - Prototype by June 2008
- November developers workshop :
 - AALib introduction
 - How to go from Use Cases to code

November 21st - 22nd workshop

- Introduction to the AALib framework :
 - Plugin architecture
 - Thread management and thread safety
 - Running both in Python and in Java (via Jython)
- How to go from Use Case to code :
 - Top down changed to bottom up
 - We first concentrate on developing modules / plugins for running MOSFLM, BEST, XDS etc
 - Data models will be program specific
 - We then combine these modules in more complex modules, for example indexing using MOSFLM or XDS, characterisation etc.
 - Data models will be generic

From Use Case to Module








Use Case



Architecture

+

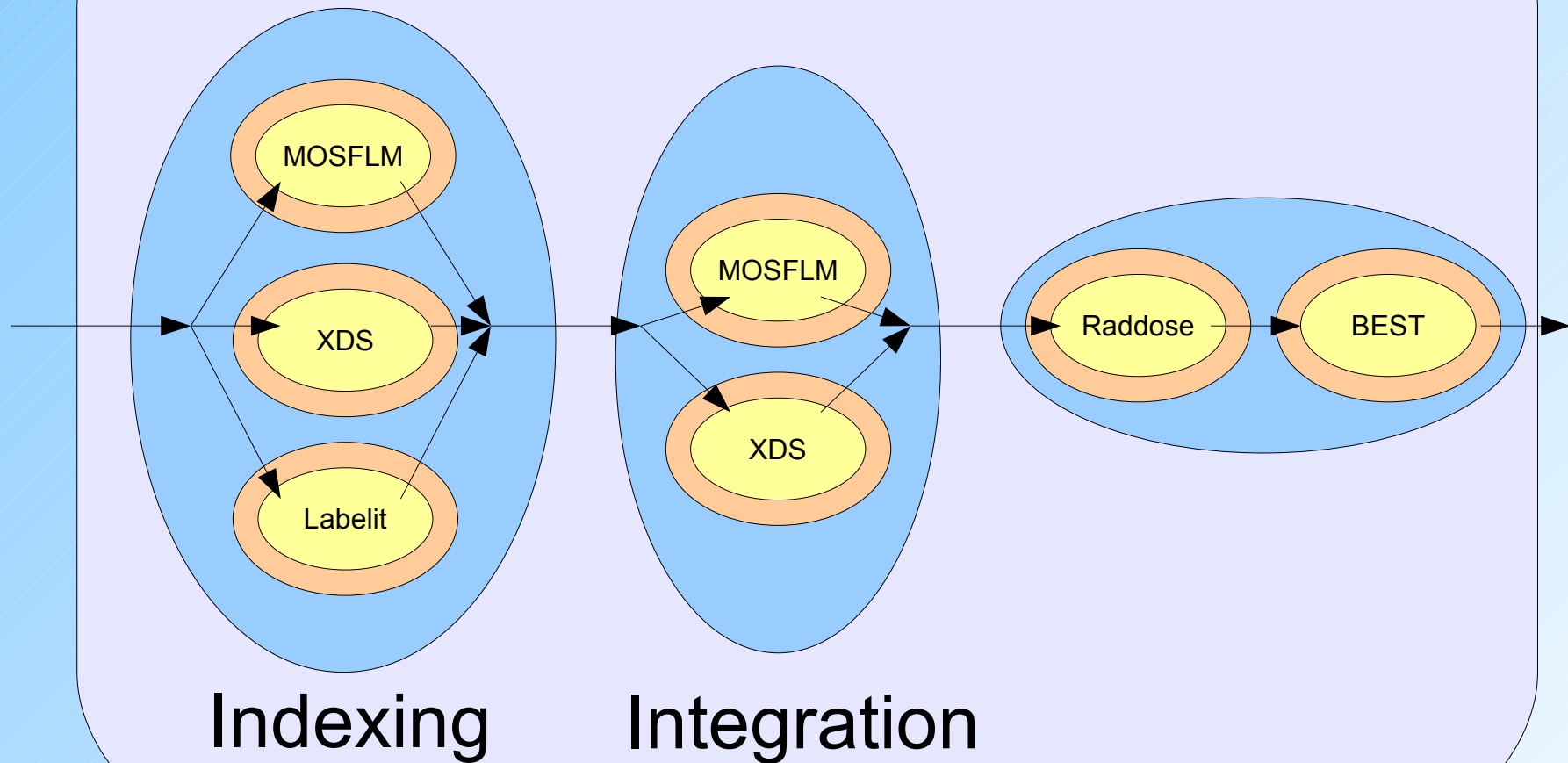
Data

	Skeleton	Configuration	Input - Output
1st Level:			
2nd Level: (Abstract level)	  	Ext. programs	Abstract layer / Generic Data?
3rd Level:	  	Working dir	Specific Data / ext program
...			

TOP
DOWN

BOTTOM
UP

Characterisation Module



Generic Data Model

Specific Data Model

Translation layer

EDNA Members

Diamond: Alun Ashton, Karl Levik, Katherine McAuley

EMBL-Grenoble: Sandor Brockhauser

EMBL-Hamburg: Gleb Bournekov, Thomas Schneider

ESRF: Marie-Françoise Incardona, Romeu Pieritz, Alexander Popov, Olof Svensson, Sean McSweeney, Gordon Leonard, Elspeth Gordon

Global Phasing: Gerard Bricogne, Clemens Vonrhein, Peter Keller

MRC Cambridge: Harry Powell and Andrew Leslie

NSLS: Robert Sweet, John Skinner

SLS: Clemens Schulze, Takashi Tomizaki

Soleil: Pierre Legrand, Olga Rudenkov, Andrew Thompson

University of York: Johan Turkenburg