Anaphe
status and prospects

Andreas Pfeiffer
CERN IT/API
andreas.pfeiffer@cern.ch

Outline

- Introduction to Anaphe
  - What, why, how
- The AIDA project
  - Abstract Interfaces for Data Analysis
- Lizard
  - Interactive Data Analysis using C++ and Python
- Summary

Anaphe
OO libraries for data analysis

Anaphe: what it is

- Modular (OO/C++) replacement of CERNLIB functionality for use in HEP experiments
  - memory management
  - I/O
  - foundation classes
  - histogramming
  - minimizing/fitting
  - visualization
  - interactive data analysis
  - simulation
- Trying to use standards wherever possible
- Trying to re-use existing class libraries

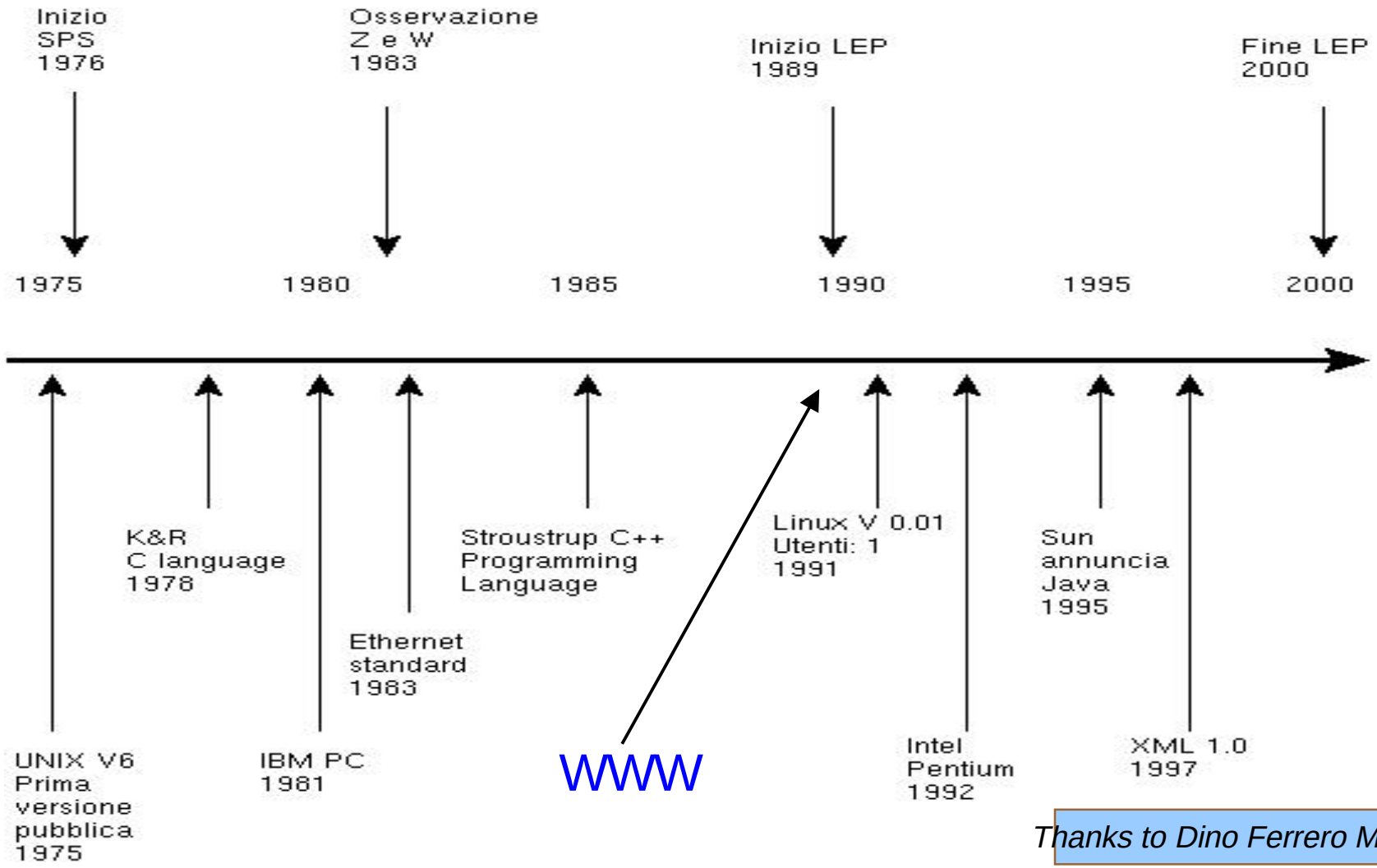
Anaphe Components

Data Analysis	Lizard - AIDA
Custom graphics (2-D)	Qt - Qplotter
Basic graphics (3-D)	OpenInventor – OpenGL
Basic math	NAG C
HEP foundation	CLHEP
HEP math	FML - Gemini - CLHEP
Histograms	HTL
Database	HepODBMS
Persistency	ODMG/Objectivity DB
C++	Standard Libraries

"Structured" Approach

- Basic functionalities (histograms, fitting, etc.) are available as individual C++ class libraries.
  - Easy replacing one part without throwing away everything
- Insulate components through Abstract Interfaces
  - "wrapper" layer to implement Interfaces in terms of existing libs
- Apply s/w quality control tools
  - code checking, testing
- Consider long lifetime needed for LHC
  - Changes are inevitable
  - "Code for now, design for the future"

Lifetime of LHC software = 25 yrs



Thanks to Dino Ferrero Merlino(I

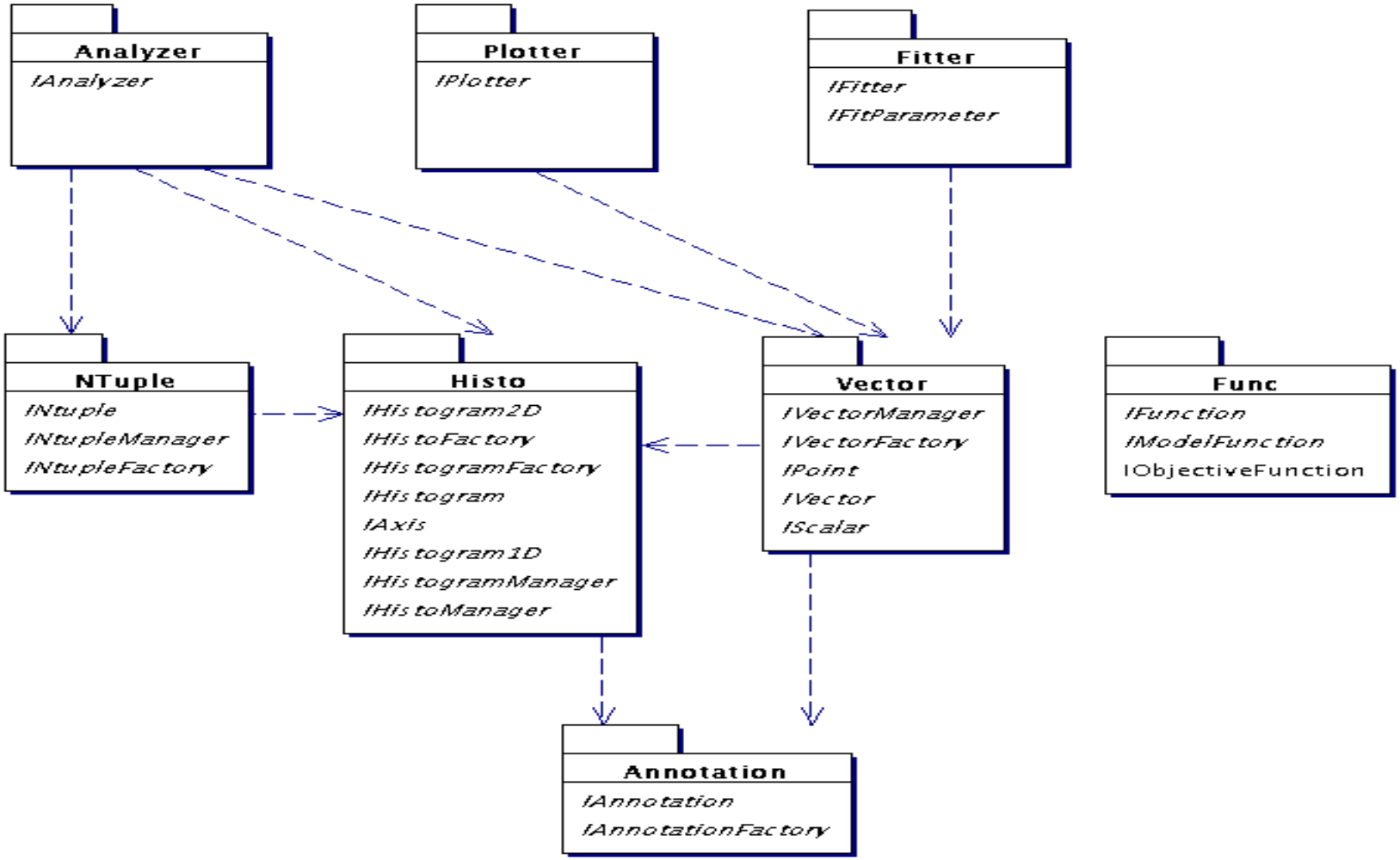
Architectural issue: Components

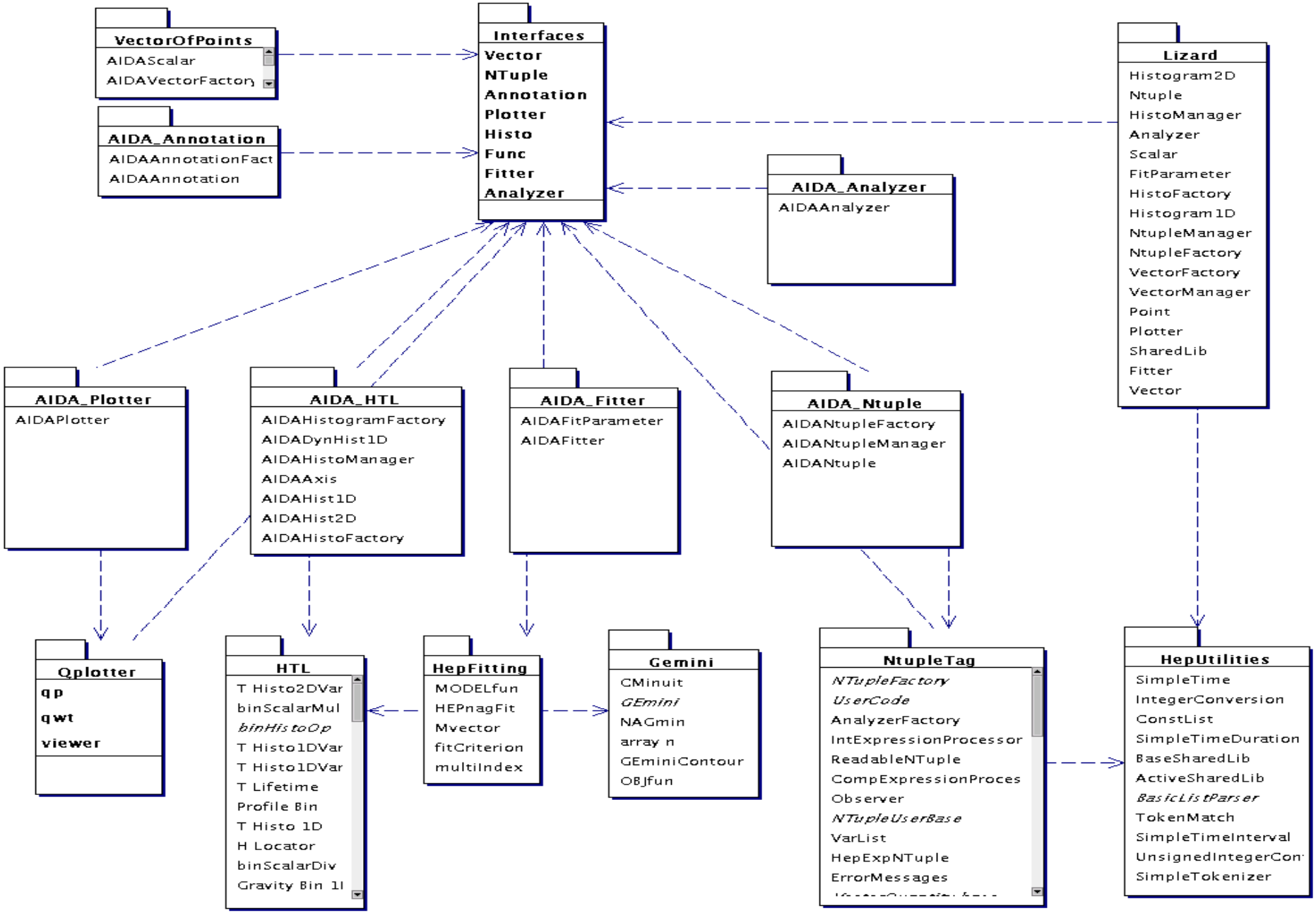
- Identify components by functionality
  - not by "historic use"
- Separation of different aspects for each component
  - example: Histogram
    - statistical entity (density distribution of a physics quantity)
    - view of a "collection of data points" (which *can* be a density distribution but also a detector efficiency curve)
    - command to manipulate/store/plot/fit/…
  - "User's view" is different from "implementor's view"
    - separate Abstract Interfaces for both aspects
    - "command-layer" vs. "implementation-layer(s)"
- Identify and use patterns - avoid anti-patterns
  - learn from other people's experiences/failures

Architectural issue: Abstract Interfaces

- Abstract Interfaces
  - only pure virtual methods, inheritance only from other A.I.
  - components use other components only through their A.I.
  - defines a kind of a "protocol" for a component
  - maximize flexibility and re-use of packages
  - allow each component to develop independently
  - re-use of existing packages to implement components reduces start-up time significantly
  - independent components ease maintenance
- AIDA project (Abstract Interfaces for Data Analysis) was initiated at the HepVis'99 workshop
  - **aida.freehep.org/**

Anaphe Internals: (Abstract) Interfaces




VectorOfPoints
AIDAScalar
AIDAVectorFactory
AIDA_Annotation
AIDAAnnotationFact
AIDAAnnotation
Interfaces
Vector
NTuple
Annotation
Plotter
Histo
Func
Fitter
Analyzer
Lizard
Histogram2D
Ntuple
HistoManager
Analyzer
Scalar
FitParameter
HistoFactory
Histogram1D
NtupleManager
NtupleFactory
VectorFactory
VectorManager
Point
Plotter
SharedLib
Fitter
Vector
AIDA_Analyzer
AIDAAnalyzer
AIDA_Plotter
AIDAPlotter
AIDA_HTL
AIDAHistogramFactory
AIDADynHist1D
AIDAHistoManager
AIDAAxis
AIDAHist1D
AIDAHist2D
AIDAHistoFactory
AIDA_Fitter
AIDAFitParameter
AIDAFitter
AIDA_Ntuple
AIDANtupleFactory
AIDANtupleManager
AIDANtuple
Qplotter
qp
qwt
viewer
HTL
T Histo2DVar
binScalarMul
binHistoOp
T Histo1DVar
T Histo1DVar
T Lifetime
Profile Bin
T Histo 1D
H Locator
binScalarDiv
Gravity Bin 1l
HepFitting
MODELfun
HEPnagFit
Mvector
fitCriterion
multiIndex
Gemini
CMinuit
GEmini
NAGmin
array n
GEminiContour
OBJfun
NtupleTag
NTupleFactory
UserCode
AnalyzerFactory
IntExpressionProcessor
ReadableNTuple
CompExpressionProces
Observer
NTupleUserBase
VarList
HepExpNTuple
ErrorMessages
HepUtilities
SimpleTime
IntegerConversion
ConstList
SimpleTimeDuration
BaseSharedLib
ActiveSharedLib
BasicListParser
TokenMatch
SimpleTimeInterval
UnsignedIntegerCon
SimpleTokenizer

AIDA
Abstract Interfaces for Data Analysis

The AIDA project

- AIDA project (Abstract Interfaces for Data Analysis) was initiated at the HepVis'99 workshop
  - since then several (mini-) workshops (main ones Paris 2000 and Boston 2001) have been organized
- Presently active mainly developers from existing packages
  - JAS (Tony Johnson)
  - Lizard (Andreas Pfeiffer)
  - OpenScientist (Guy Barrand)
  - Wired (Mark Dönszelmann)

Introduction

- Design Interfaces for Data Analysis (in HEP)
  - "*The goals of the AIDA project are to define abstract interfaces for common physics analysis tools, such as histograms. The adoption of these interfaces should make it easier for developers and users to select to use different tools without having to learn new interfaces or change their code. In addition it should be possible to exchange data (objects) between AIDA compliant applications*."
- Open for contributions of any kind
  - suggestions, code, implementations, …

History

- started in Sept. 1999 (HepVis'99, Orsay)
- release 1.0 summer 2000
  - concentrated on "developers view"
    - Histograms only
- release 2.0 spring 2001
  - much more complete set -- for feedback on design
  - Concentrated discussions on
    - Management
      - directory like Tree
    - Store
    - Tuple

Status

- release 2.2 fall 2001
  - refinements after collecting feedback and starting to implement
  - About 25 Interface classes were defined
    - written down in Java and checked into FreeHep-CVS
- Used by Gaudi/Athena (LHCb, Atlas, Harp)
  - Gaudi people involved in design
- Used by IGUANA project (CMS)
- Adopted and used in Geant-4 examples

Status of Components

Histogram	close to complete, implementations exist
Tree	discussed, being implemented
Cloud	discussed, being implemented
Store	discussed, being implemented
Tuple	discussed, being implemented
Function	discussed, being implemented
Fitter	discussed, being implemented
MasterFactory	discussed, being implemented
Plotter/Document	discussion not yet started

Analysis Tools

- Java Analysis Studio
  - started on reference implementation
  - JAIDA package to access Java components from C++
    - C++ proxies implement AIDA using Java Interfaces
- Open Scientist
  - most advanced in implementation
- Lizard / Anaphe
  - will provide parallel implementation for some time for user convenience
- All plan to have full implementation by end 2001

Lizard: a tool for Interactive Data Analysis

Scripting in Lizard

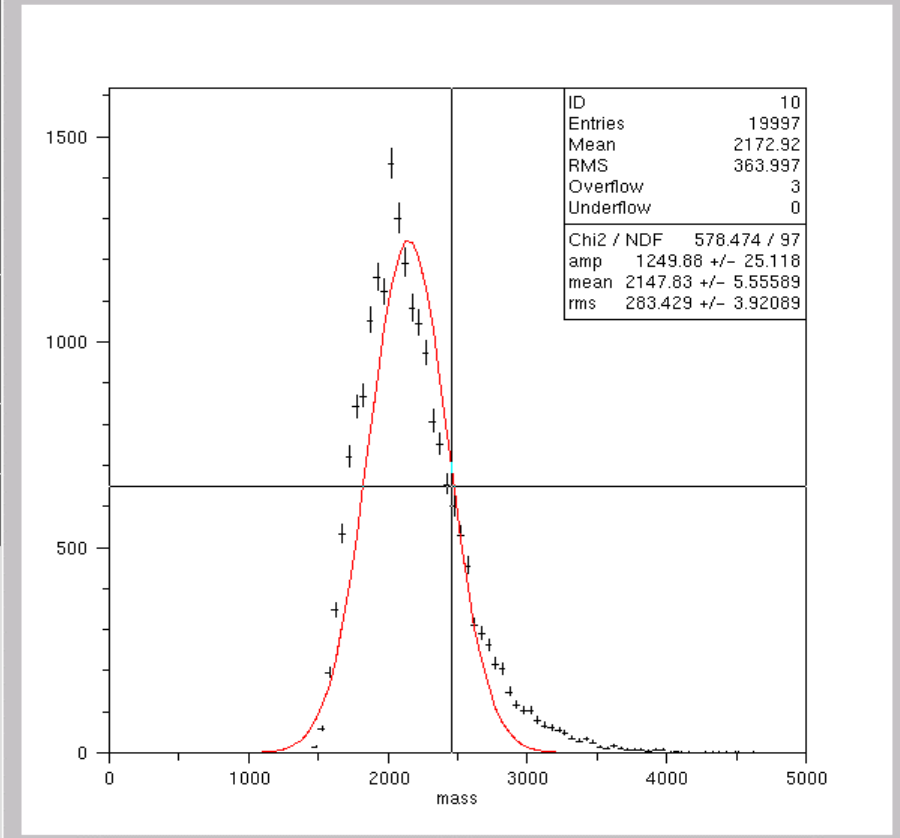
- Typical use of scripting is quite different from programming (reconstruction, analysis, ...)
  - history "go back to where I was before"
  - repetition/looping - with "modifiable parameters"
- SWIG to (semi-) automatically create connection to chosen scripting language
  - allows flexibility to choose amongst several scripting languages
  - Python, Perl, Tcl, Guile, Ruby, (Java) ...
- Python - OO (scripting) language
  - stable language for user-interface
  - independent of language for reconstruction/analysis
- Make use of language specific features as needed
  - templates in C++, interactivity in Python

Example script (ntuple)

```
# get list of names of all tuples from tuplemanager
ntm.listTuples()
nt1=ntm.findNtuple("Charm1")   # retrieve tuple by name

# create 1D histos to project into
h1=hm.create1D(10, "mass" ,100, 0., 5000.)
h2=hm.create1D(20, "mass for pt1>10" ,100, 0., 5000.)

# project the attribute "MASS" into histo h1 without  cut ("")
nt1.project1D( h1, "" , "MASS")

# project the attribute "MASS" into histo h2 with  cut ("PT1>10")
nt1.project1D( h2, "PT1>10" , "MASS")
```

(Control Client) | xterm | xterm <2> | xterm <4> | emacs: ntup.py | Qplotter | (xterm <3>)

Qplotter



X Coordinate : 2456.9 Y Coordinate : 648.406

emacs: ntup.py

File Edit Apps Options Buffers Tools

```
ntm.listNtuples()                          # get list of names of all tuples from tupl

nt1=ntm.findNtuple("Charm1")               # retrieve tuple by name
# nt1.listAttributes()                     # prints names and types of attributes

h1=hm.create1D(10,"mass",100,0.,5000.)  # create 1D histo for one of the attribute
h2=hm.create1D(20,"mass for pt1>10",100,0.,5000.)

# for the updating plot, reset the plotter to a single zone
pl.zone(1,1)
h1.reset()
start = 0
num = 1000
for i in range(0.,20.):
    nt1.project1D(h1,"MASS","",start,num)
    vTemp=vm.from1D(h1)
    pl.plot(vTemp)
    del vTemp
    start=start+num

# that's it !
```

```
d classes initialised
```

```
:-) exe("ntup.py")

Explorables present:

        Charm1

:-) err()
:-) h1=hm.retrieveHisto1D(10)
:-) vFit = hfit(h1,"G")
+++Some of the bins have zero errors. They will be ignored in the chi-square fit.
+++Use setDefaultError(double) to redefine zero errors.

+++ Performing a chi-square fit.
:-)
```

One | Two | Three | Four | Five | Six | 15:05

Lizard: History and Present Status

- Started after CHEP-2000
- Full version out since June 2001
  - "PAW like" analysis functionality **plus:**
  - on-demand loading of compiled code using shared libraries
    - gives full access to experiment's analysis code and data
  - based on Abstract Interfaces
    - flexible and extensible
- "License free" version since fall 2001
  - uses HBook and Minuit
  - flexible selection of configuration automatically or at startup
    - `lizard --noLicense`
    - `lizard --useHBook --useMinuit`

Future Enhancements

- Implementation of AIDA version 2.x Interfaces
  - also: JAS (Tony Johnson) and OpenScientist (Guy Barrand)
- Access to other implementations of components
- Reading of ROOT (> V3.0) files
  - similar to Tony Johnson's (Java) RootIO package
- AIDA Ntuple/Histo store
  - optimized for Ntuples; Histograms as (compressed) XML
- Communication with Java tools/packages (JAS, Wired)
  - via AIDA
- Adding other scripting languages
- GUI from Iguana ?

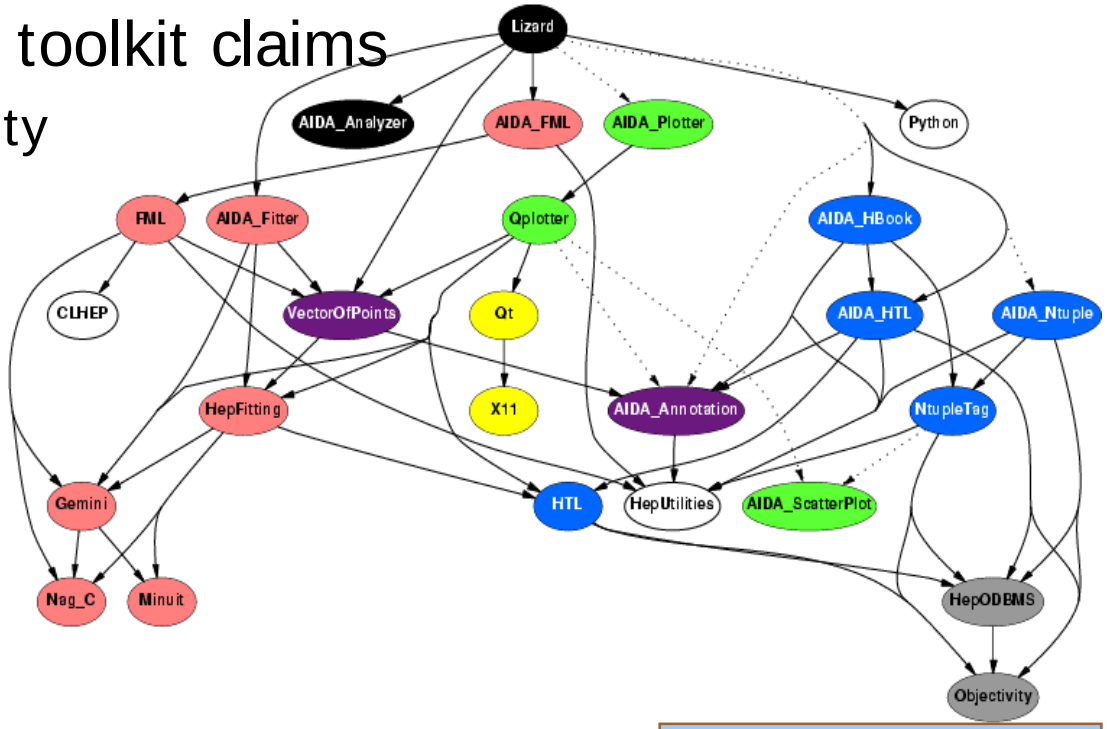
Software quality control

- Using tools for testing/checking has started
  - Insure++, CodeWizard
- Package dependencies: Ignominy
  - Set of perl and shell scripts by Lassi Tuura (CMS)
  - Ignominy **scans...**
    - Make dependency data produced by the compilers (*.d files)
    - Source code for #includes (resolved against the ones actually seen)
    - Shared library dependencies ("ldd" output)
    - Defined and required symbols ("nm" output)
  - And **maps...**
    - Source code and binaries into packages
    - #include dependencies into package dependencies
    - Unresolved/defined symbols into package dependencies

> **ignominy:** *dishonour, disgrace, shame; infamy; the condition of being in disgrace, etc.*
>
> (Oxford English Dictionary)

Ignominy Analysis of Anaphe

- Distribution of tools and utilities for LHC era physics
  - Combination of commercial, free and HEP software
  - Claims to be a toolkit
- Seems to live up to its toolkit claims
  - Good work on modularity
  - Clean design is evident in many places
  - Dependency diagrams often split naturally into functional units

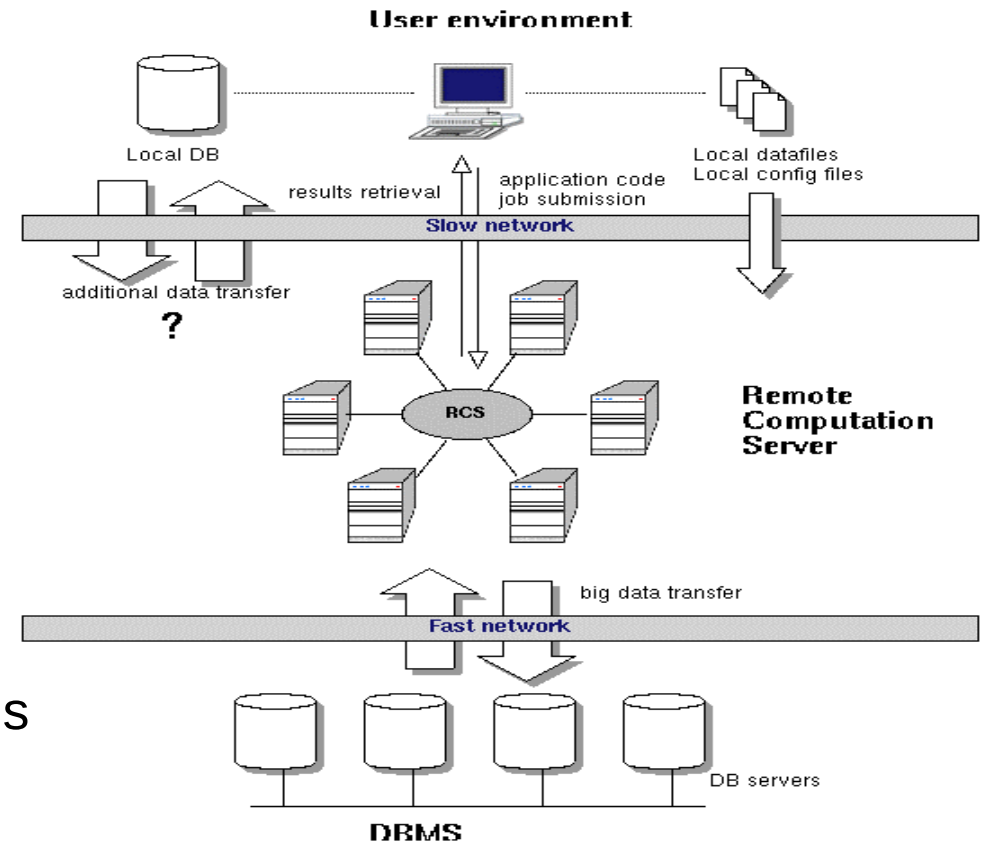


Anaphe 3.6.1 Direct Logical Depend
Lizard by Purpose

Thanks to Lassi Tuura (CMS

Challenge: Distributed Computing

- Motivation
  - move code to data
  - parallel analysis
- Techniques
  - services via AI
  - late binding
  - plug-in architecture
- End-user (Lizard)
  - look-and-feel of local analysis



- R&D started and first prototype available soon
  - CORBA based

Summary

- The architecture of Anaphe shows some important items for flexible and modular data analysis:
  - weak coupling between components through use of Abstract Interfaces
    - standards defined by AIDA project
  - basic functionality is covered by individual C++ class libraries
  - emphasis on usability and maintainability

- Major criteria are flexibility, extensibility and interoperability
  - recent example: GEANT-4 examples using analysis components based on AIDA

- Lizard is an Interactive Data Analysis Tool based on Anaphe components and the Python scripting language (through SWIG)
  - Lizard is young but has very solid base in mature Anaphe libraries
  - real plug-in structure

More information

- **cern.ch/ Anaphe**
- **cern.ch/ Anaphe/ Lizard**
- **aida.freehep.org/**

Users and Collaborations

- AIDA spoken here!
  - IGUANA (CMS visualization)
  - GAUDI (LHCb/HARP) framework
  - ATHENA (Atlas) framework
  - Analyzer modules in Geant 4
  - JAS
  - Open Scientist
  - …you?

Package Metrics

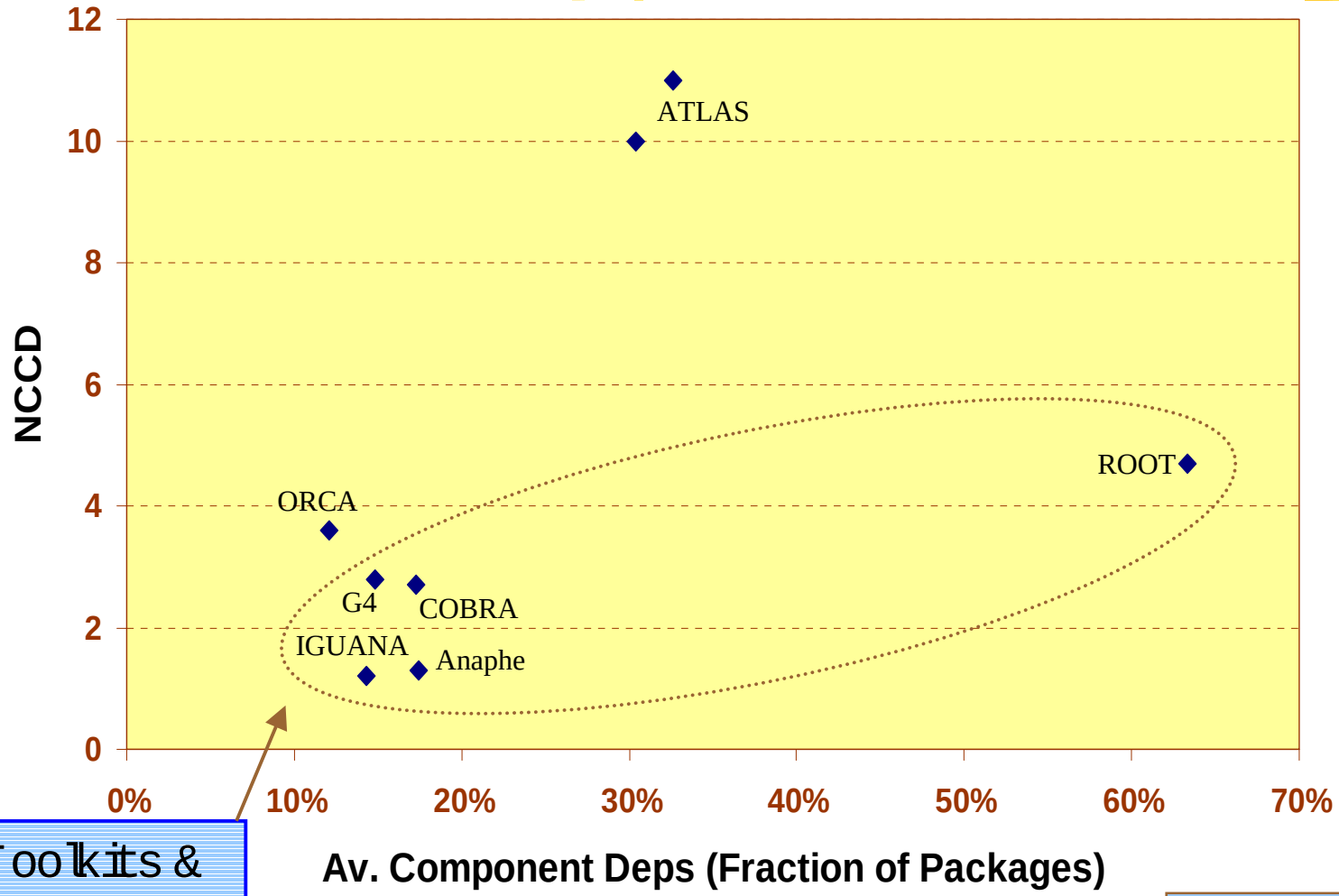
Project	Release	Packages	Average # of direct dependencies	Cycles (Packages Involved)	# of levels	ACD*	CCD*	NCCD*	Size
Anaphe	3.6.1	**31**	**2.6**	**--**	8	5.4	167	**1.3**	630/170k
ATLAS	1.3.2	**230**	**6.3**	**2 (92)**	96	70	16211	**10**	1350k
	1.3.7	**236**	**7.0**	**2 (92)**	97	77	18263	**11**	1350k
CMS/ORCA	4.6.0	**199**	**7.4**	**7 (22)**	35	24	4815	**3.6**	420k
CMS/COBRA	5.2.0	**87**	**6.7**	**4 (10)**	19	15	1312	**2.7**	180k
CMS/IGUANA	2.4.2	**35**	**3.9**	**--**	6	5.0	174	**1.2**	150/38k
Geant4	4.3.2	**108**	**7.0**	**3 (12)**	21	16	1765	**2.8**	680k
ROOT	2.25/05	**30**	**6.4**	**1 (19)**	22	19	580	**4.7**	660k

**) John Lakos, Large-Scale C++ Programming*

- Size = total amount of source code (not normalised across projects!)
- ACD = average component dependency (~ libraries linked in)
- CCD = sum of single-package component dependencies over whole release
  - Indicates testing/integration cost
- NCCD = Measure of CCD compared to a balanced binary tree
  - A good toolkit's NCCD will be close to 1.0
  - < 1.0: structure is flatter than a binary tree (= independent packages)
  - > 1.0: structure is more strongly coupled (vertical or cyclic)
  - **Aim: NCCD ~ 1 for given software/ functionality**

Thanks to Lassi Tuura (CMS)

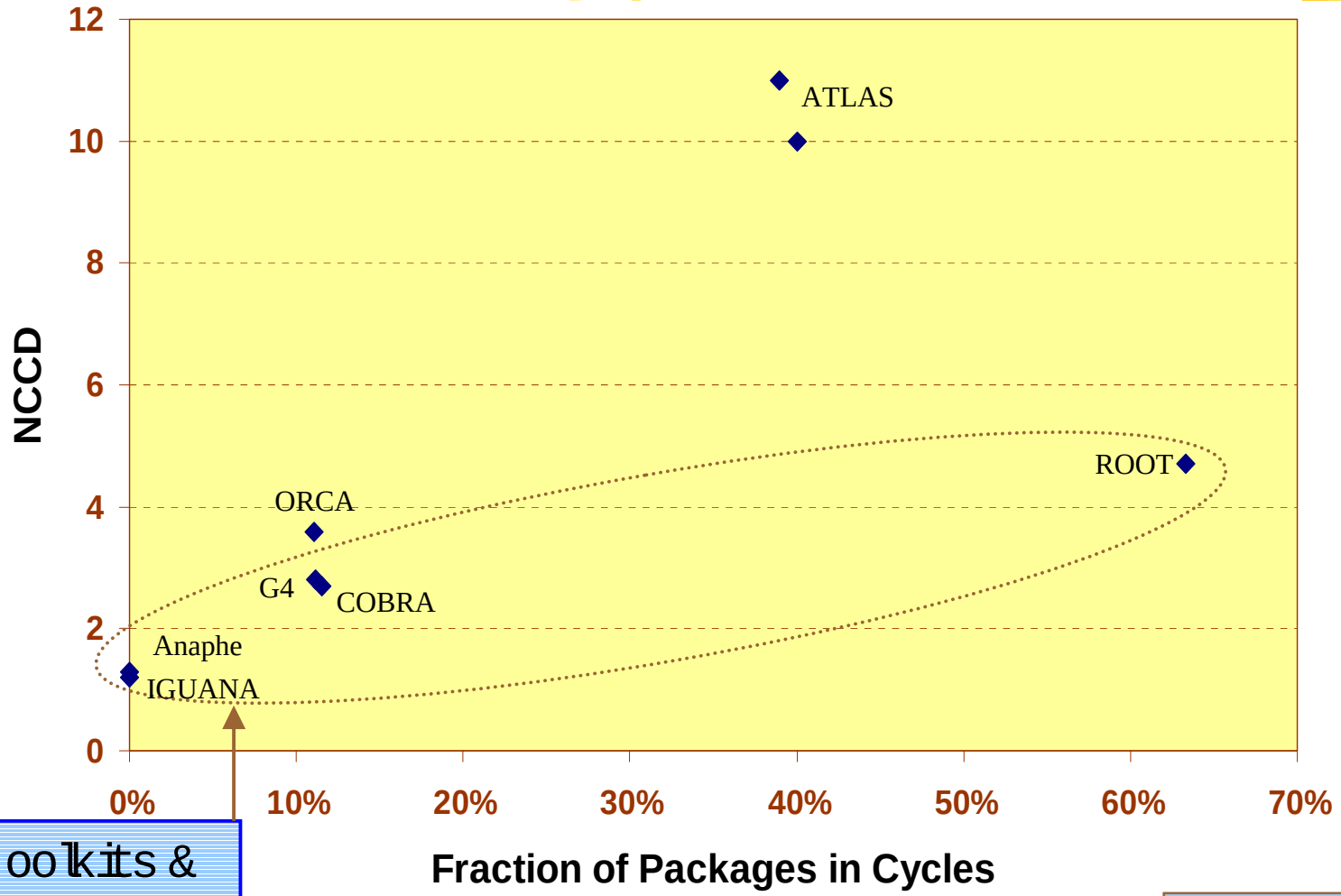
Metrics: NCCD vs ACD



Toolkits & Frameworks

Thanks to Lassi Tuura (CMS

Metrics: NCCD vs Cycles

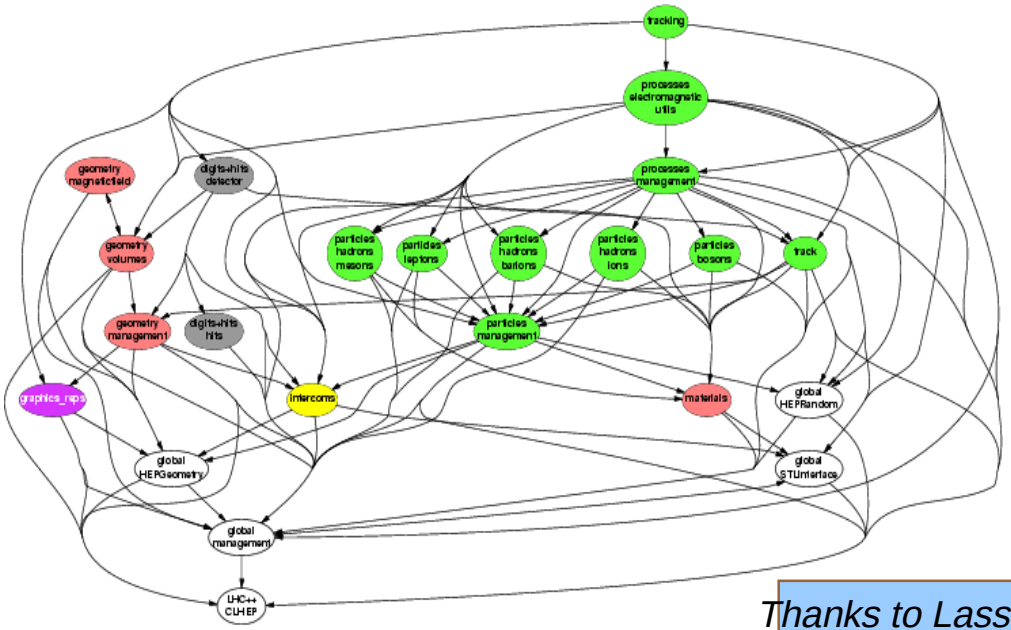


Toolkits & Frameworks

Thanks to Lassi Tuura (CMS

Analysis of Geant4

- Fairly large C++ project
  - Very fine-grained (and multi-level) package structuring
  - Seems quite clean from the preliminary analysis
- Fine package subdivision helps in many ways but makes analysis and code understanding more complicated
- One subsystem seems strongly coupled and needs attention
- Need to study the use of the internal command system



Thanks to Lassi Tuura (CMS

Analysis of ROOT

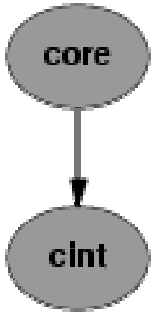
- ROOT developers have done a formidable job of breaking binary (shared library) dependencies, but…
  - For example: By static analysis, **nothing** seems to use the postscript package directly (no incoming dependencies), but there is this code:

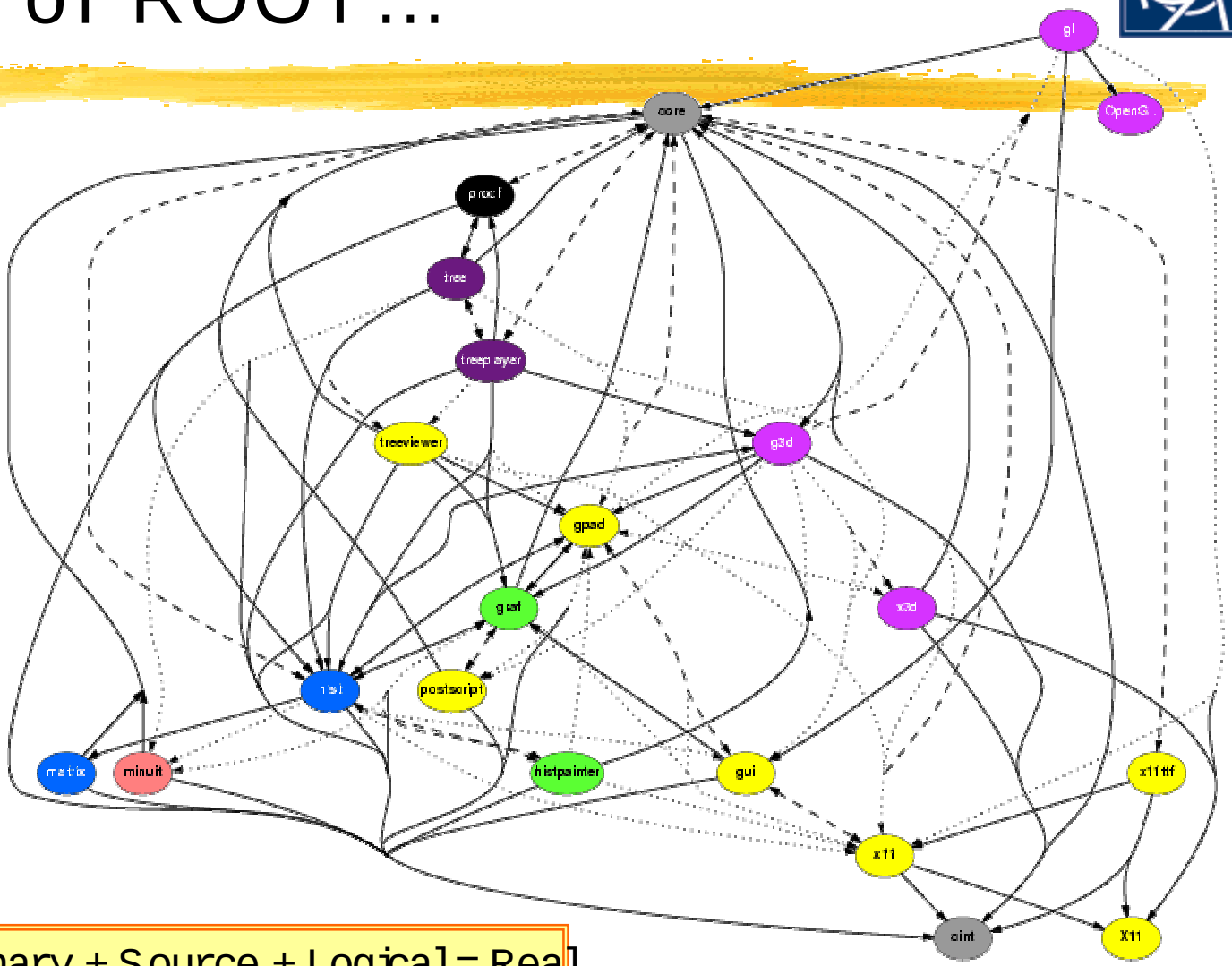
```
void TPad::Print (const char *filename, Option_t *option) { […]
     TVirtualPS *psave = gVirtualPS;
     if (gROOT->LoadClass("TPostScript","Postscript")) return;
     gROOT->ProcessLineFast("new TPostScript()");
     gVirtualPS->Open(psname,pstype);
     gVirtualPS->SetBit(kPrintingPS); […] }
```

  - Taking these and global objects into account makes the dependency diagrams **very different**
- Sign of fast growth? Need a "next evolutionary step"?
  - So "coherent" that replacing parts could get painful…

Thanks to Lassi Tuura (CMS

Analysis of ROOT...



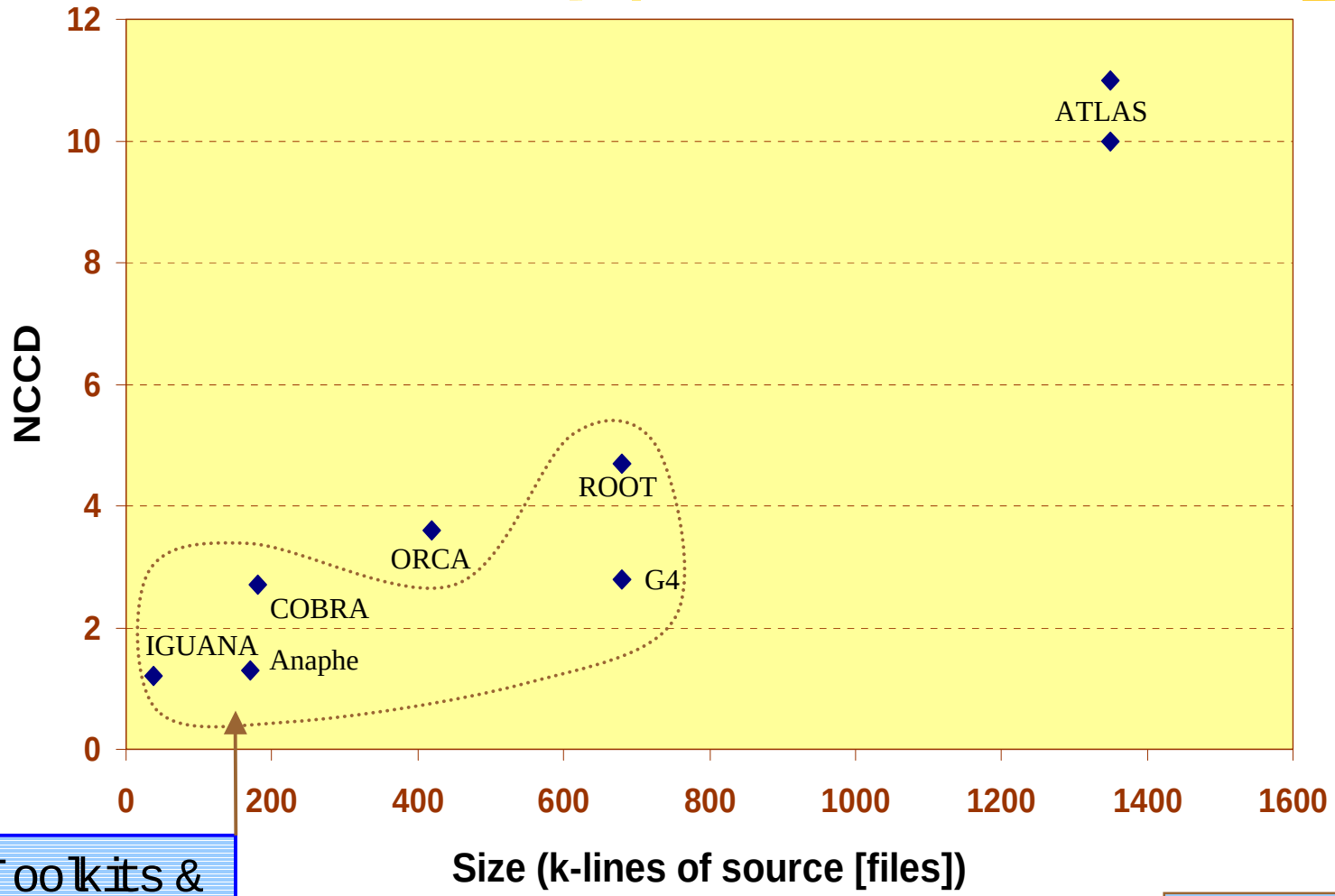


Binary only

Binary + Source + Logical = Real

Thanks to Lassi Tuura (CMS

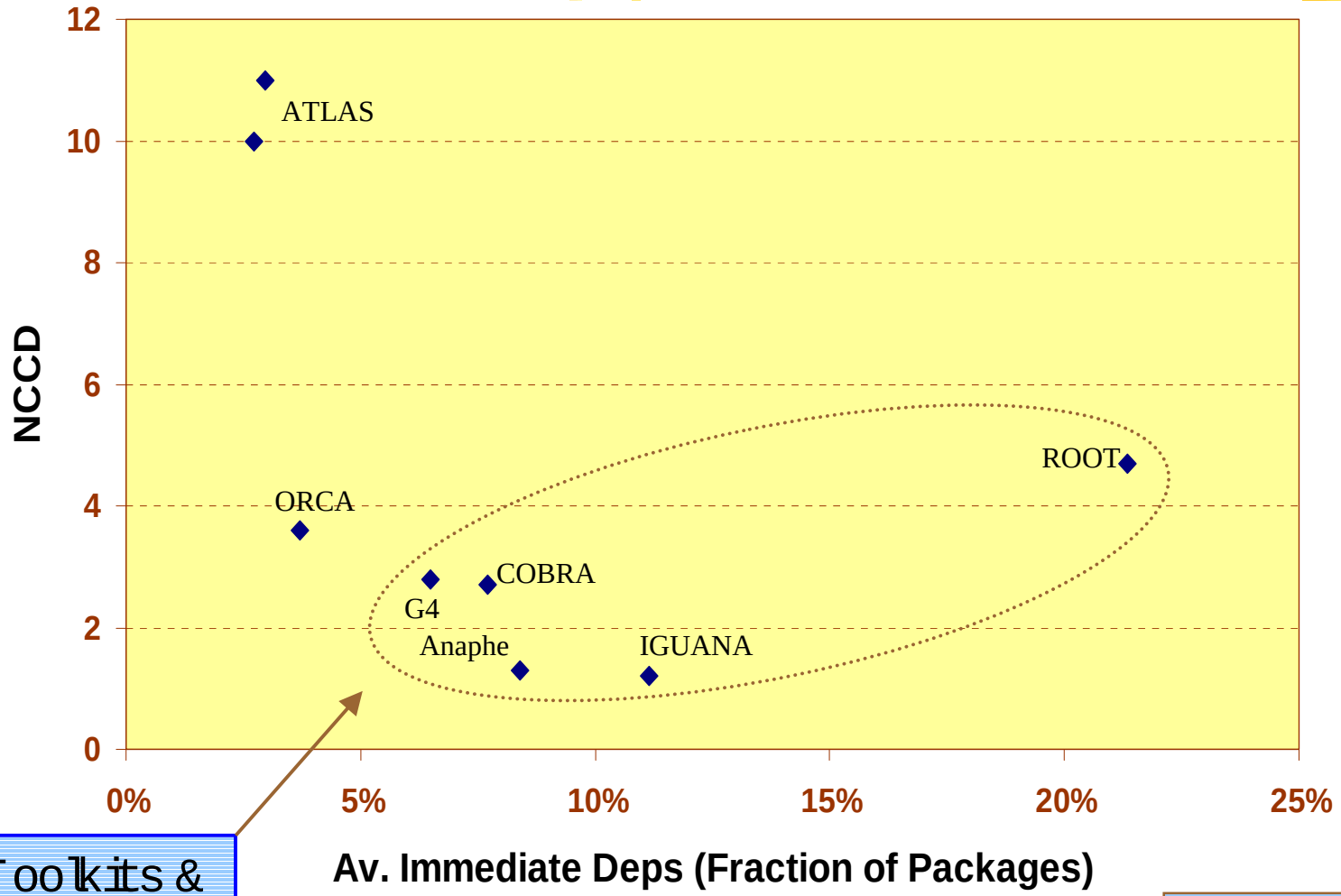
Metrics: NCCD vs Size



Toolkits & Frameworks

Thanks to Lassi Tuura (CMS

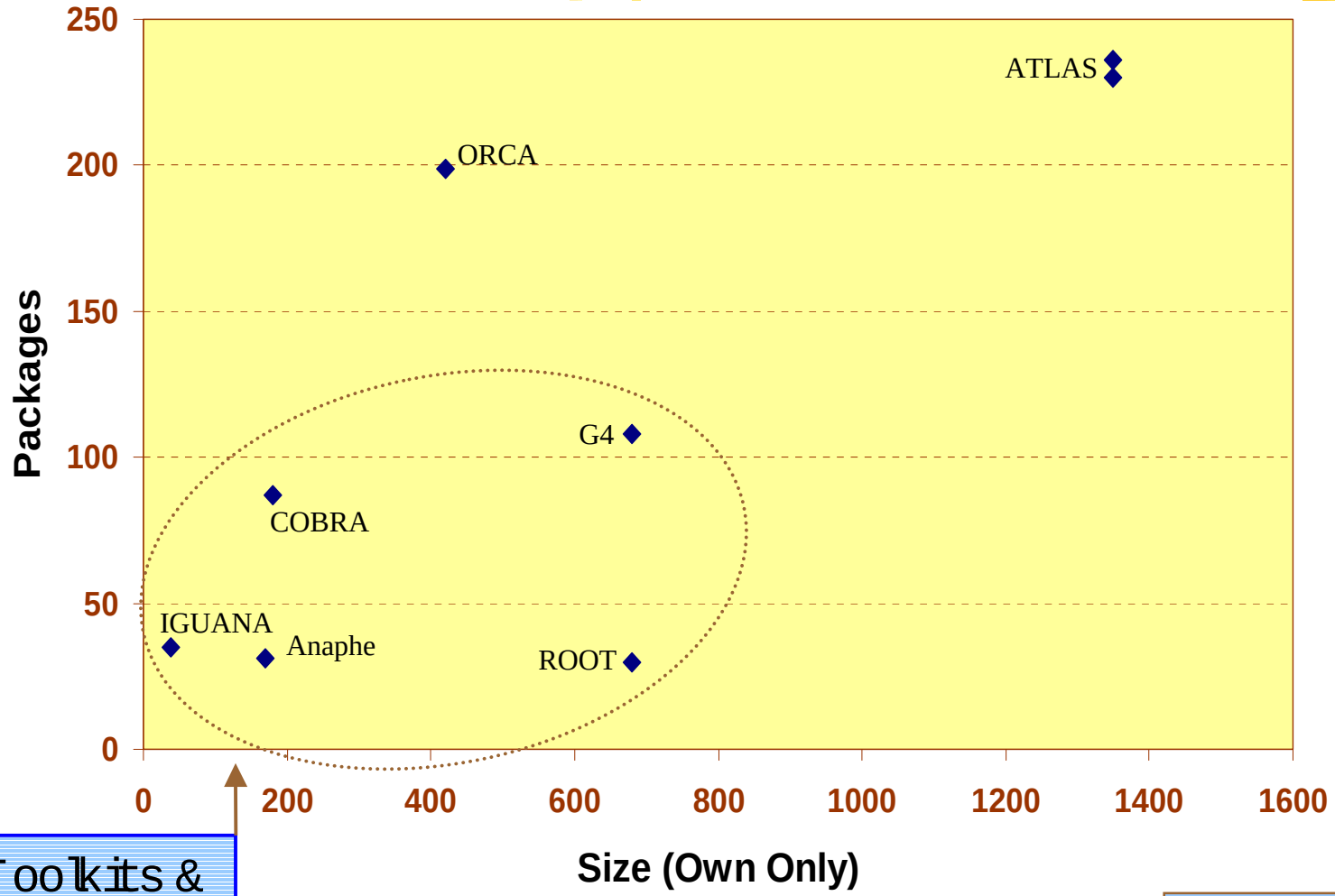
Metrics: NCCD vs AID



Toolkits & Frameworks

Thanks to Lassi Tuura (CMS

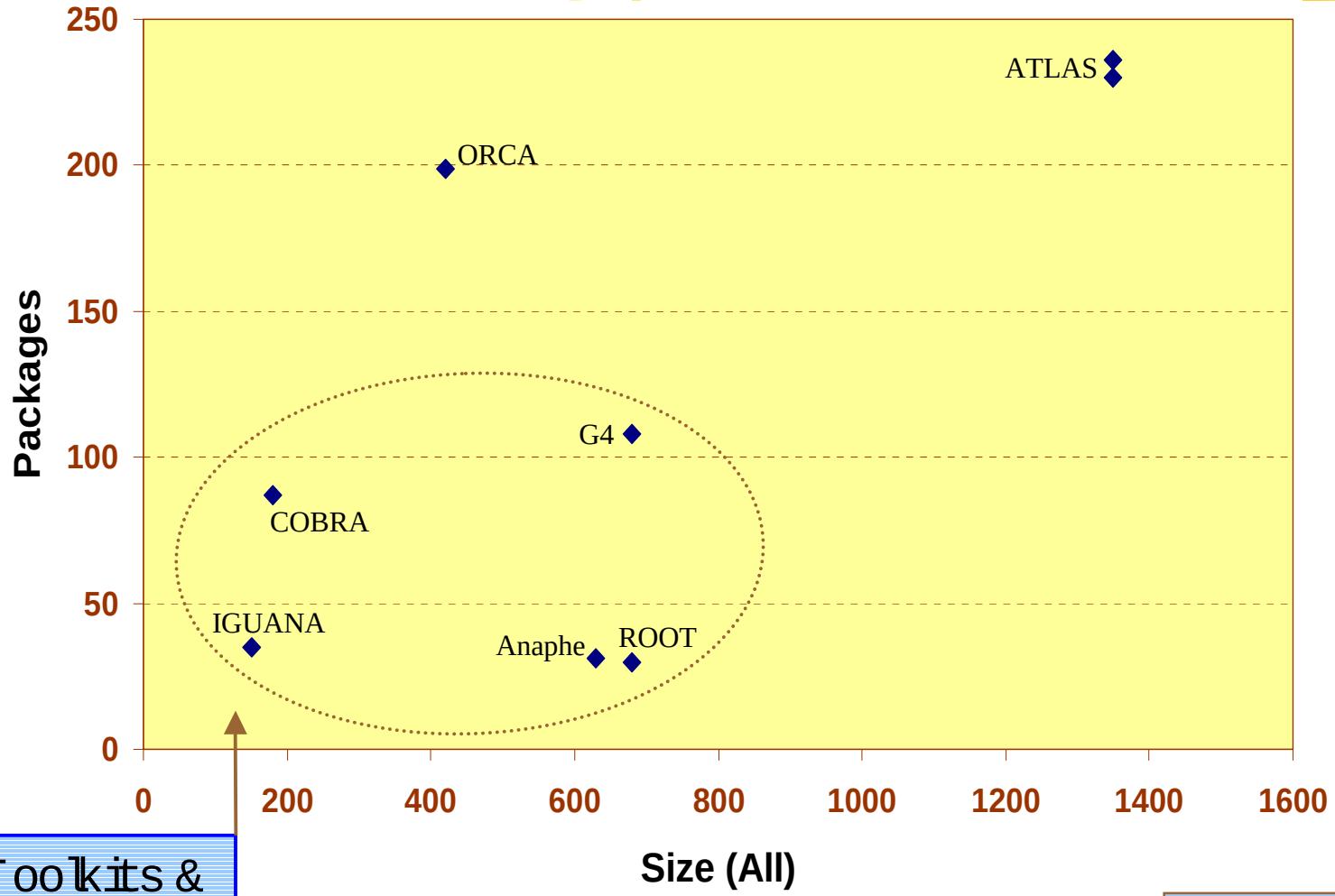
Metrics: Packages vs Size



Toolkits & Frameworks

Thanks to Lassi Tuura (CMS

Metrics: Packages vs Size



Toolkits & Frameworks

Thanks to Lassi Tuura (CMS