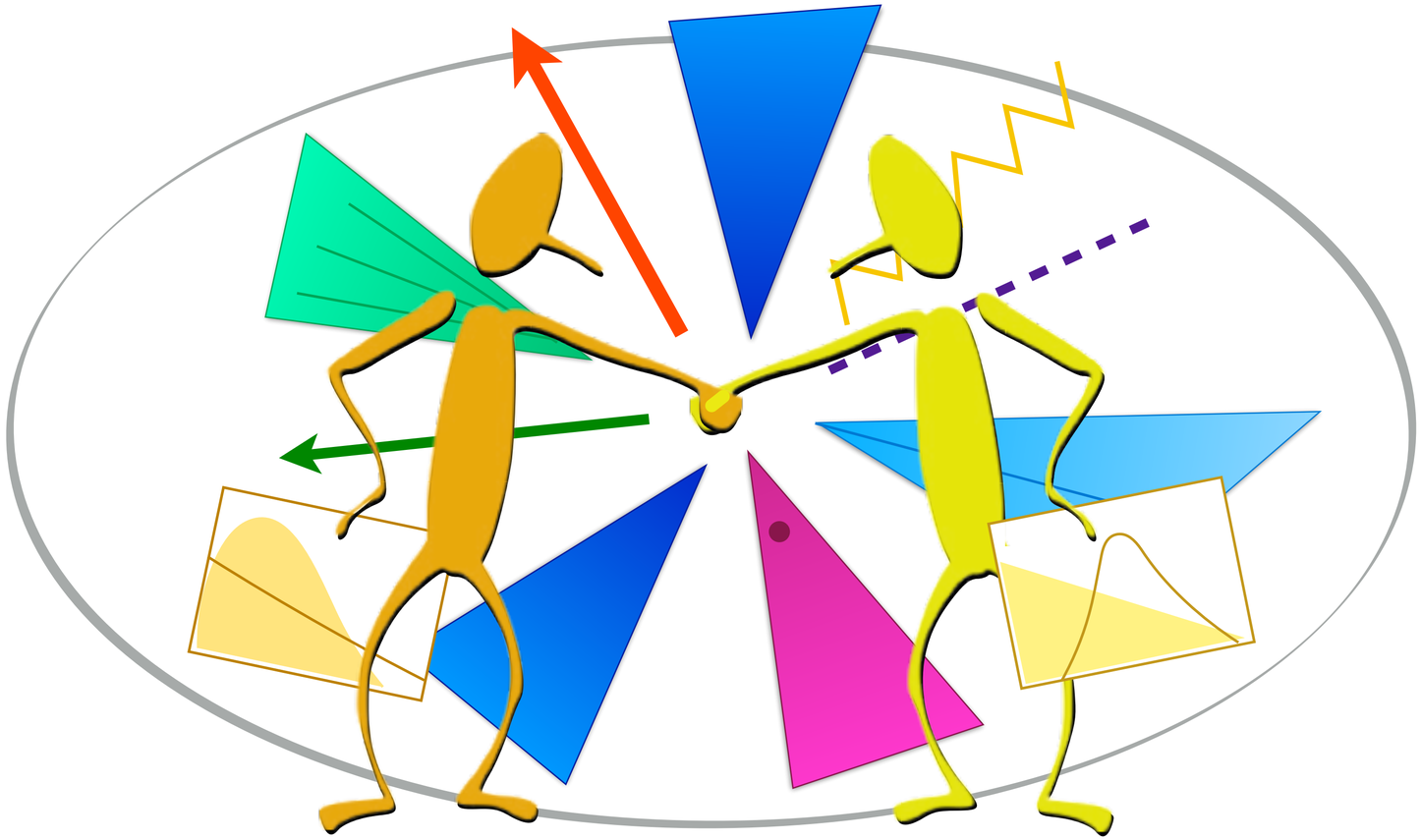


Analysis Description Languages for the LHC

Sezen Sekmen

Kyungpook National University

LHCP 2020, 25-30 May 2020, Online

Welcome to the LHC physics analysis jungle

Inclusive analyses with hundreds of selection regions

Overlaps between different analyses?

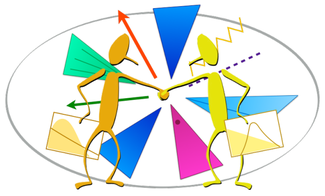
Is my control region your signal region???

Multiple analyses exploring similar final states

Many alternative definitions for analysis objects

Many variables, ambiguous definitions

…time to get better organized to work more efficiently!



Analysis description languages for LHC

An **Analysis Description Language** for the LHC is:

- A special language capable of describing an LHC analysis in a standard and unambiguous way.
  - Customized to express analysis-specific concepts.
- Designed for use by anyone with an interest in, and knowledge of, LHC physics : experimentalists, phenomenologists, other enthusiasts…

Earlier physics / analysis-related HEP formats/languages proved successful and useful:

- SUSY Les Houches Accord
- Les Houches Event Accord

The features of an analysis description language:
(Les Houches 2015 new physics WG report, arXiv:1605.02684, sec 16)

- Should be complete in content, demonstrably correct, easily learnable and sustainable.
- Can be easy to read, self-contained and analysis framework-independent.

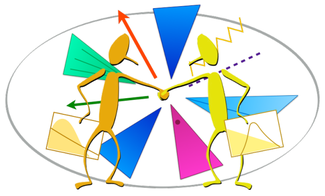
What kind of a language?

What we mainly use at the LHC now:

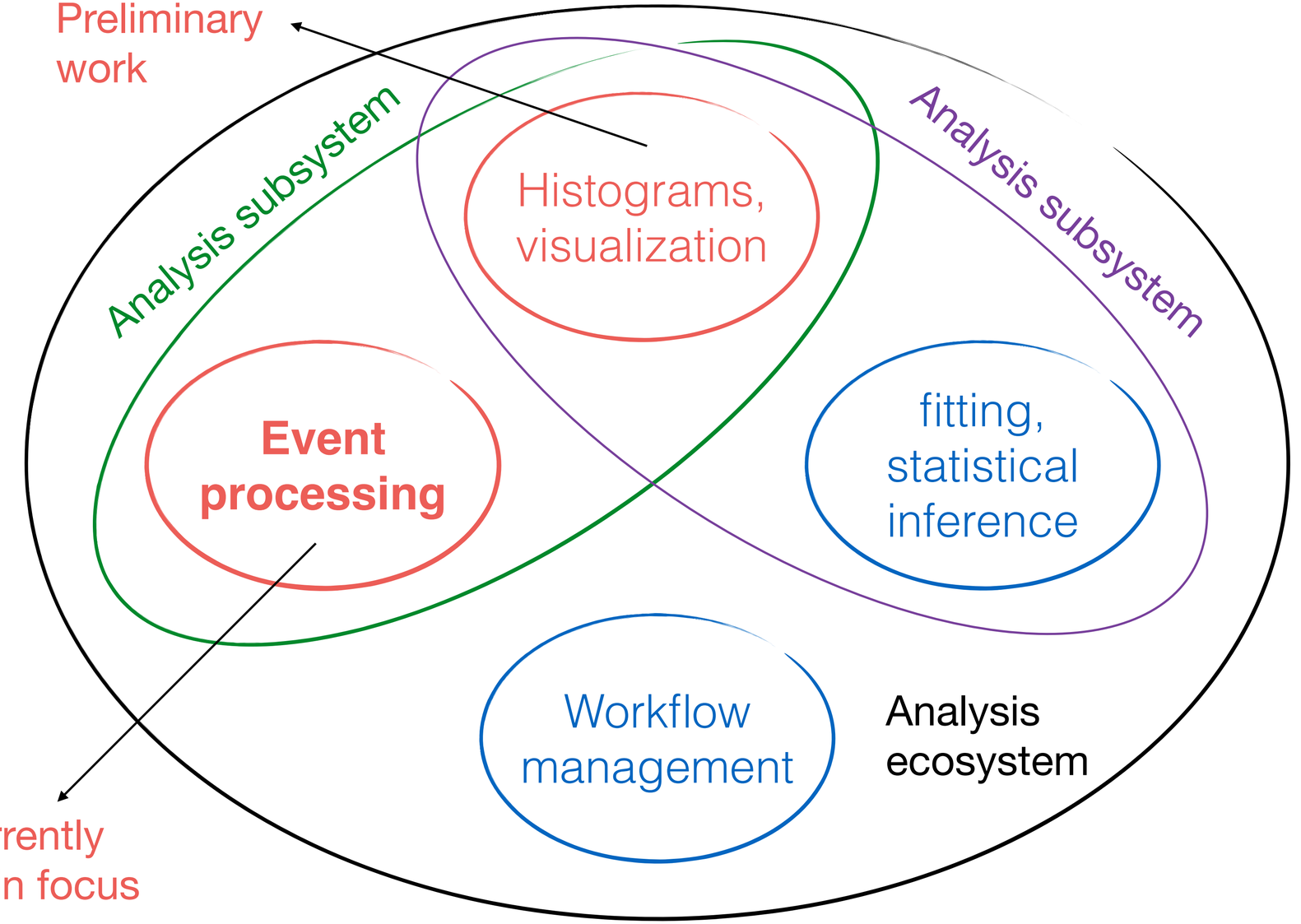
- General purpose language (GPL): A computer language that is broadly applicable across many application domains (FORTRAN, c++, python, …).
  - Used for solving very wide range of problems.

What we can consider for the future:

- Domain-specific language (DSL): a computer language specialized to a particular application domain (regular expressions, Makefile, SQL, …).
  - Designed to model how domain experts think about specific problems they wish to solve.
- Embedded DSL (eDSL): A DSL based on the syntax of a GPL (embedded SQL, LINQ, …)
- Declarative language: A language that expresses the logic of a computation without describing its control flow. Describes what needs to be done, but not how to do it.



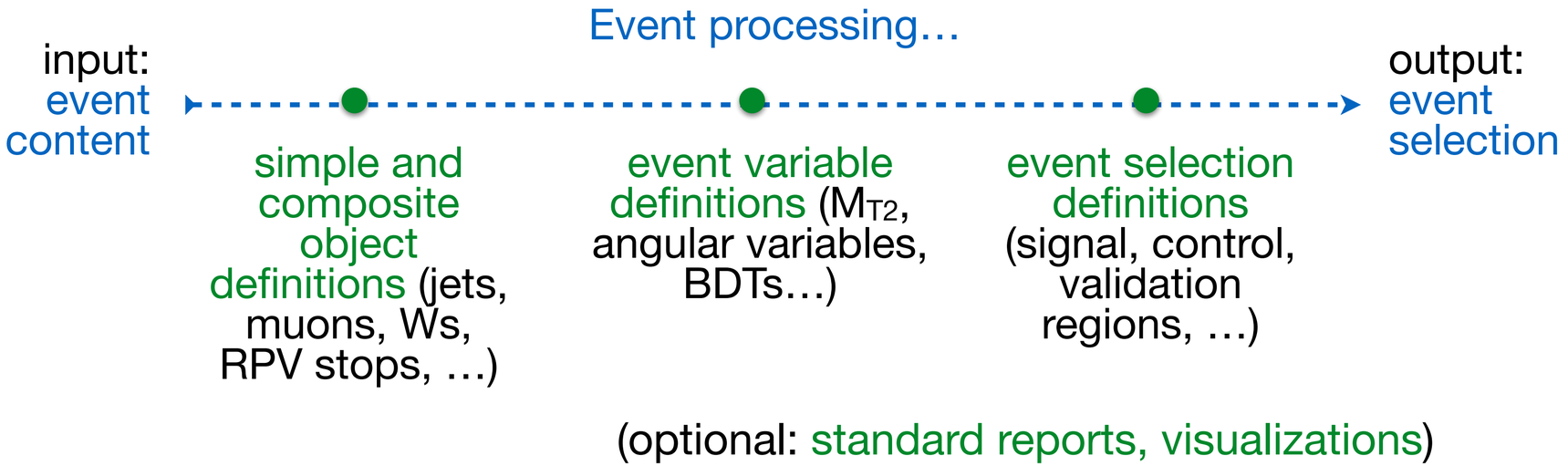
What does an analysis description language describe?



Language scope

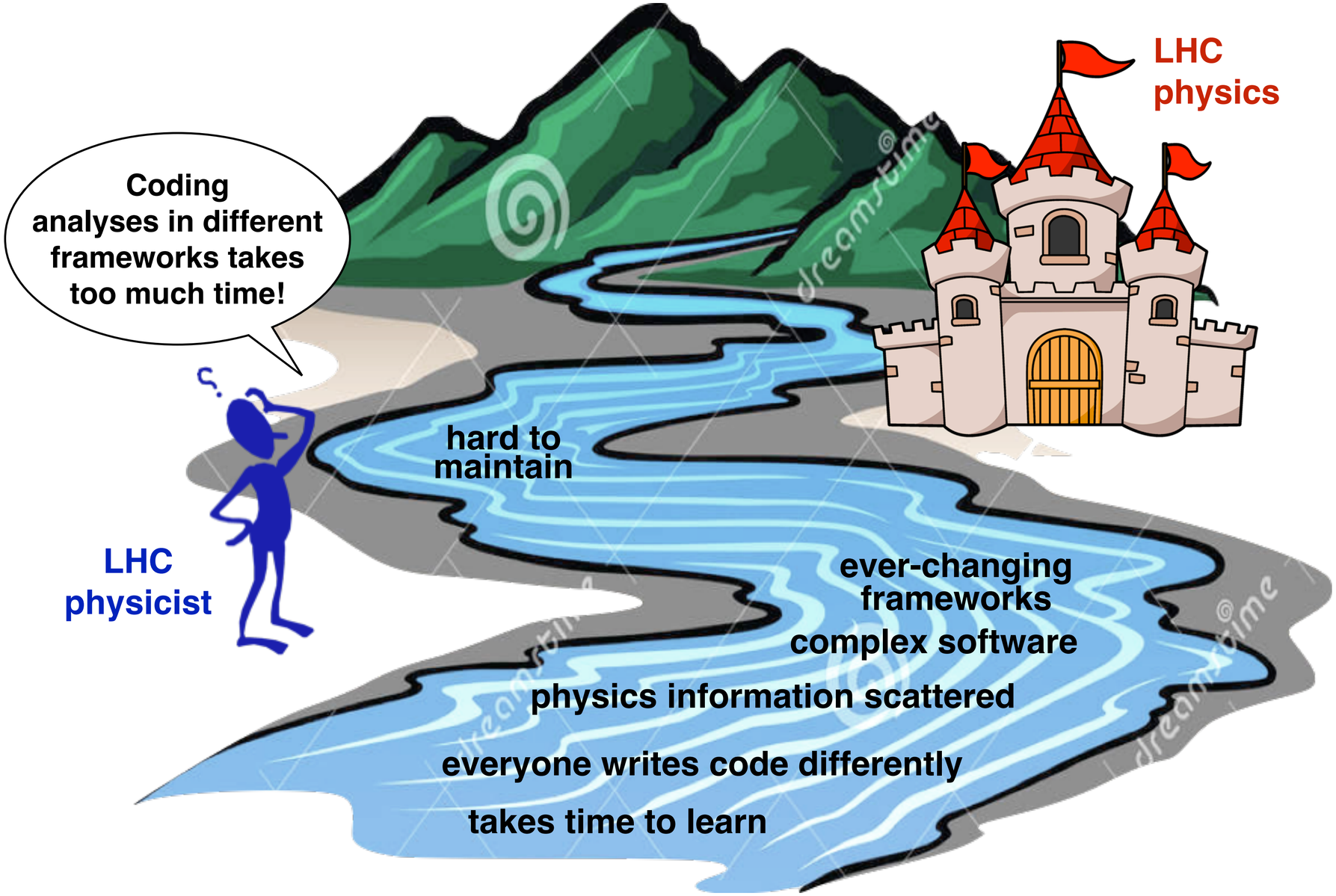
By construction, an analysis description language is not designed to be general purpose; therefore, getting the right scope is key.

The **core** of any analysis description language for the LHC should include

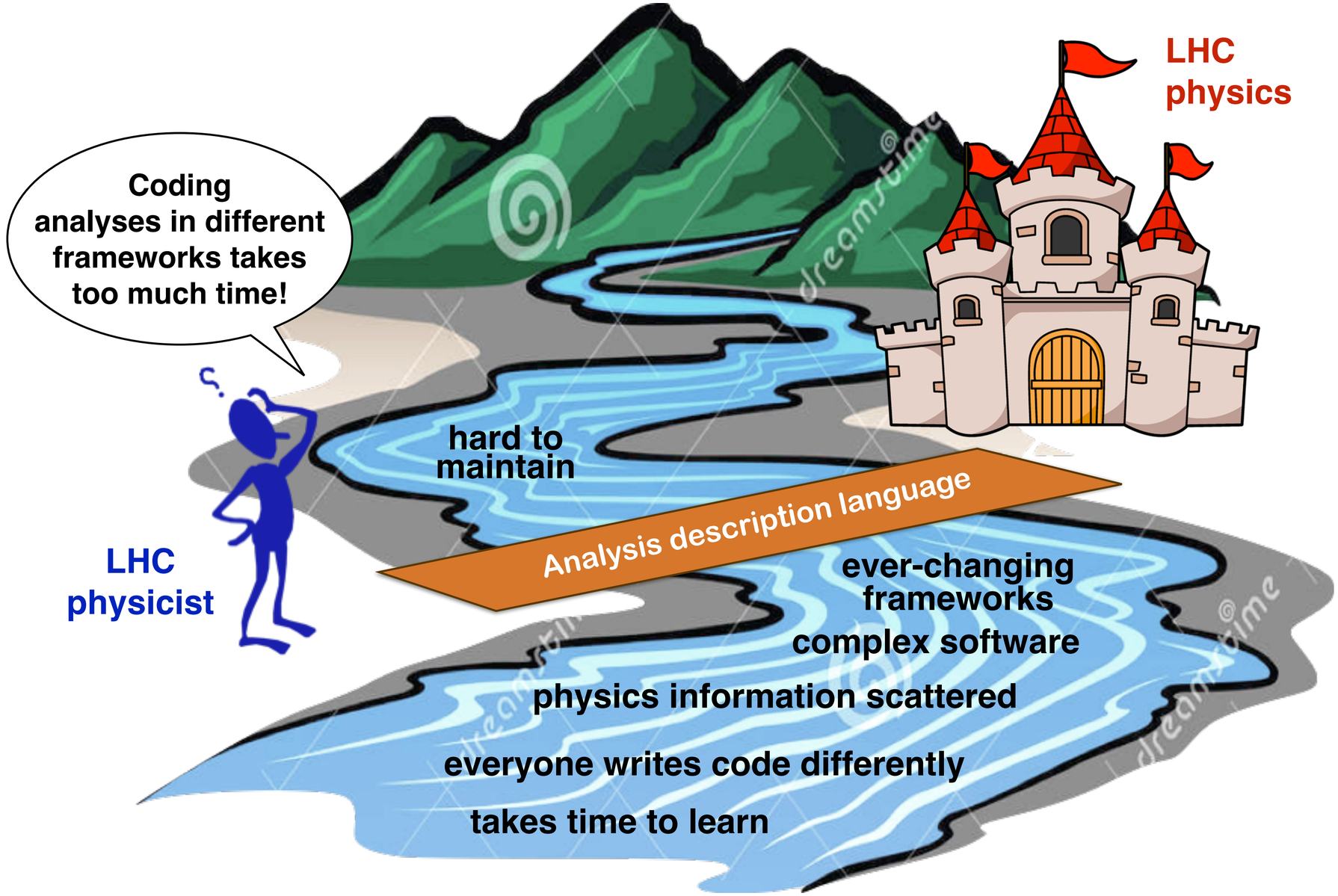


Further operations with selected events (background estimation methods, scale factor derivations, etc.) can vary greatly, and thus may not be easily considered within a standard language scope.

Framework independence of analysis description highly desirable



Framework independence of analysis description highly desirable



A dedicated language could instead offer a standard input to multiple analysis frameworks.

An analysis description language would help everyone

Motivation / use case	Exp	TH/Pheno	Public
Analysis abstraction, design, implementation	✔	✔	✔
Improve our way of thinking about our analyses modelling and structure	✔	✔	✔
Analysis communication, synchronization, visualization	✔	✔	✔
Analysis review by internal or external referees	✔	✔	✔
Easier comparison; combination of analyses	✔	✔	
Interpretation studies, analysis reimplementation	✔	✔	✔
Analysis preservation (ongoing discussions with CERN Analysis Preservation Group)	✔	✔	✔
Well-suited for handling increasingly large data — scaling.	✔		

LHADA/CutLang —> ADL

ADL consists of

- a plain text file describing the analysis using a easy-to-read DSL with dedicated syntax rules.
- a library of self-contained functions encapsulating variables that are non-trivial to express with the ADL syntax (e.g. MT2, ML algorithms). Internal or external (user) functions.

- ADL database with 15 LHC analyses: https://github.com/ADL4HEP/ADLLHCanalyses

- Separate object, variable, event selection definitions into blocks with a keyword value structure, where keywords specify analysis concepts and operations.

```
blocktype blockname
  keyword1 value1
  keyword1 value2
  keyword3 value3 # comment
```

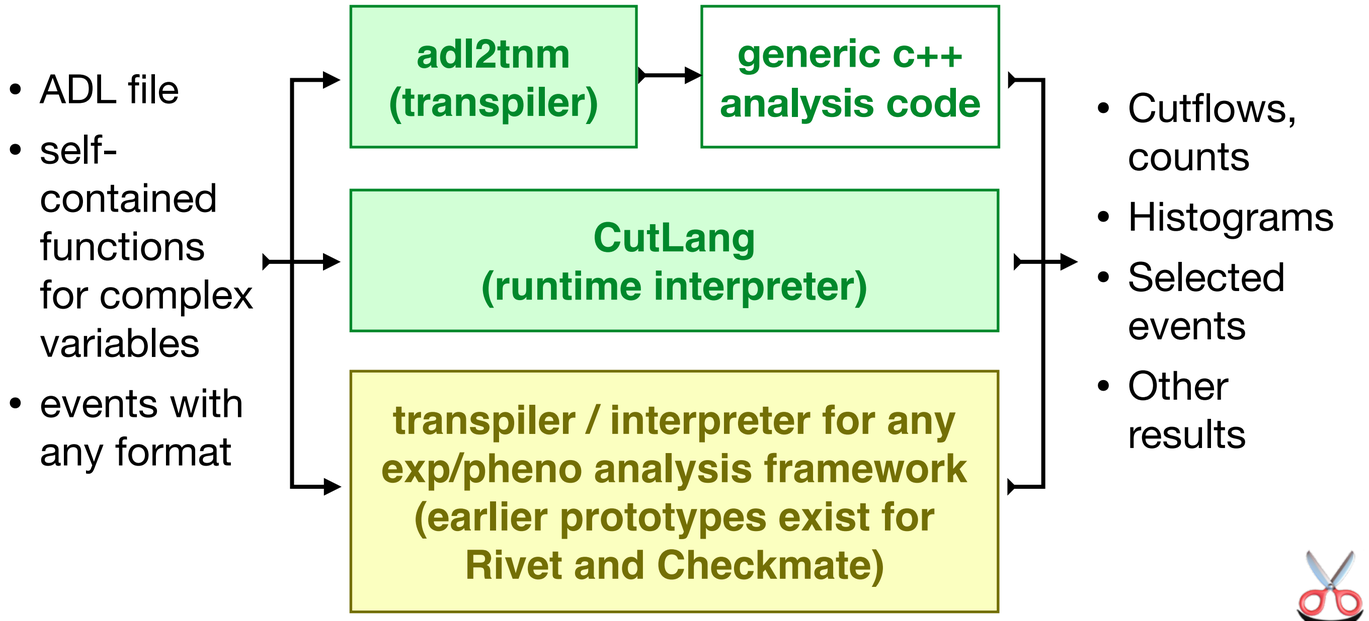
- Syntax includes mathematical and logical operations, comparison and optimization operators, reducers, 4-vector algebra and HEP-specific functions (dϕ, dR, …).

LHADA (Les Houches Analysis Description Accord): Les Houches 2015 new physics WG report (arXiv:1605.02684, sec 17)

CutLang: Comput.Phys.Commun. 233 (2018) 215-236 (arXiv:1801.05727), ACAT 2019 proceedings (arXiv:1909.10621)

Running analyses with ADL

Experimental / phenomenology analysis model with ADLs



adl2tnm Github: https://github.com/hbprosper/adl2tnm

CutLang Github: https://github.com/unelg/CutLang

Earlier prototype **lhada2rivet** Github: https://github.com/lhada-hep/lhada/tree/master/lhada2rivet.d

adl2tnm (Harrison Prosper, SS), arXiv:1803.10379

- python transpiler converting ADL to c++ code executed within the TNM (TheNtupleMaker) generic analysis framework. Only depends on ROOT.
- Uses adapters to translate input to standard extensible object types.

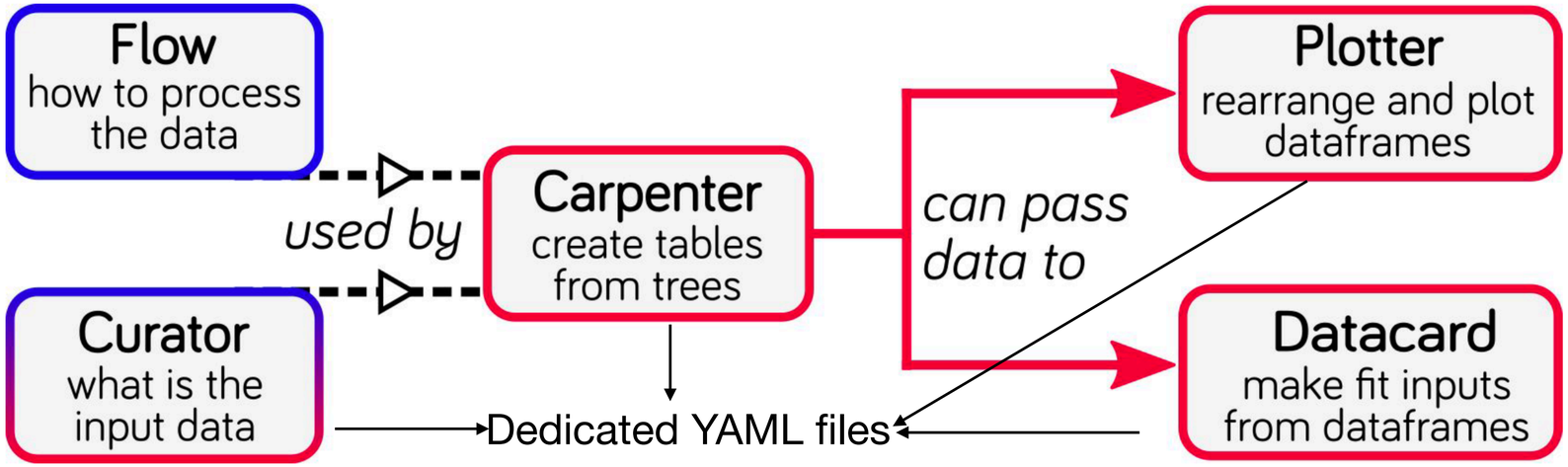
CutLang (Gökhan Unel, Bora Örgen, Arpon Paul, Nant Ravel, SS, Jinen Setpal, Anna Monıca Toon), arXiv:1801.05727, (arXiv:1909.10621)

- Runtime in terpreter written in c++
- Based on ROOT classes for Lorentz vector operations and histograms
- ADL parsing by Lex & Yacc: relies on automatically generated dictionaries and grammar.
- Includes many internal functions.
- Used in 2 ATLAS analyses.

F.A.S.T. F.A.S.T analysis tools based on YAML

Ben Krikler

YAML-based analysis configuration files handled and executed by python



- YAML-based description: Naturally declarative, easy to read.
- Data processing described in stages (which can be any python-importable class).
- Uses AlphaTwirl as backend. Can be developed to work with SPark or RDataFrame.
- Used in 2 CMS analyses, DUNE, FCC, LUX-ZEPLIN.

F. A. S. T. Github: https://gitlab.cern.ch/fast-hep/public/fast-carpenter/

NAIL Natural Analysis Implementation Language

Andrea Rizzi

- Expresses event processing operations in a declarative form. No explicit loops over events or objects. Started very recently.
- Embedded DSL written in python. Has a python interface
- Syntax resembles RDataFrame.
  - Additionally view objects as a whole, select object subcollections, define new object properties, define operations between collections.
- Generates a ROOT RDataFrame-based code
  - c++ program to compile and run OR
  - a c++ library loadable with ROOT, even in a python environment.
- Allow to express new variable definitions with c++ code snippets.
- Autodetects inputs while writing c++ code (reimplementing a feature of RDataFrame).
- Used in a CMS H —> μμ analysis.
- Ultimate scope: Define event processing in a declarative way, include building signal/background models from set of histograms, and define statistical interpretation.

NAIL Github: https://github.com/arizzi/nail

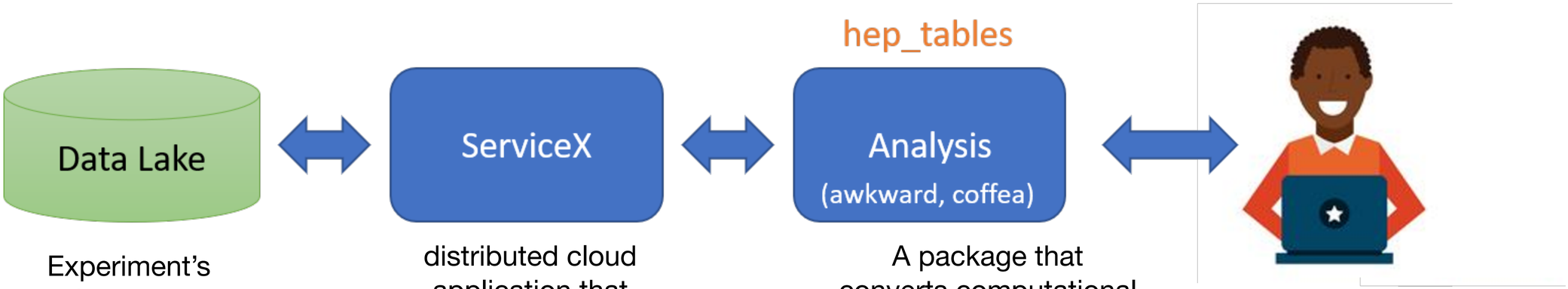
hep_tables and dataframe_expressions

Gordon Watts

Analysis system that allows easy columnar-like access to hierarchical data

- for cases when the data processing backend is in an analysis facility
- via using Jupyter notebooks or python source files to drive the analysis.

- No loops, if statements.
- Single object and event selection cuts using numpy-like slicing/masks.
- Map and lambda functions for multiple object combinations
- Basic plotting.



Experiment's data store

XRootD

Wire-level protocol used to move files

distributed cloud application that extracts columns of data quickly from the experiment's data.

qastle

Lisp-like language used to specify columns and simplified cuts

A package that converts computational graph into commands to `ServiceX` and `awkward` and `coffea` to do the user's bidding.

dataframe_expressions

allows the user to express data manipulation in a numpy-like way

Github: https://gordonwatts.github.io/hep_tables_docs/intro

Recent effort. Work in progress!

PartiQL, AwkwardQL

PartiQL (Jim Pivarski)

Toy language demonstrating features that would be a radical departure from GPLs, addressing problems specific to particle physics.

Based on applying SQL data model to individual events (not collections of events).

- Block structure for named, nested cuts and systematic variations.
- Histograms as side effects within the block structure.
- Identity defined by surrogate keys.
- Manipulation in terms of set operations.
- Syntactic macros to avoid repetitive typing.

AwkwardQL (Lindsey Gray)

SQL-like language for doing set operations on data in awkward arrays.

- Derived from PartiQL and targets awkward 1.0.
- Starting to focus on concrete particle physics use cases.

Work in progress.

PartiQL github: https://github.com/jpivarski/PartiQL

AwkwardQL github: https://github.com/lgray/AwkwardQL

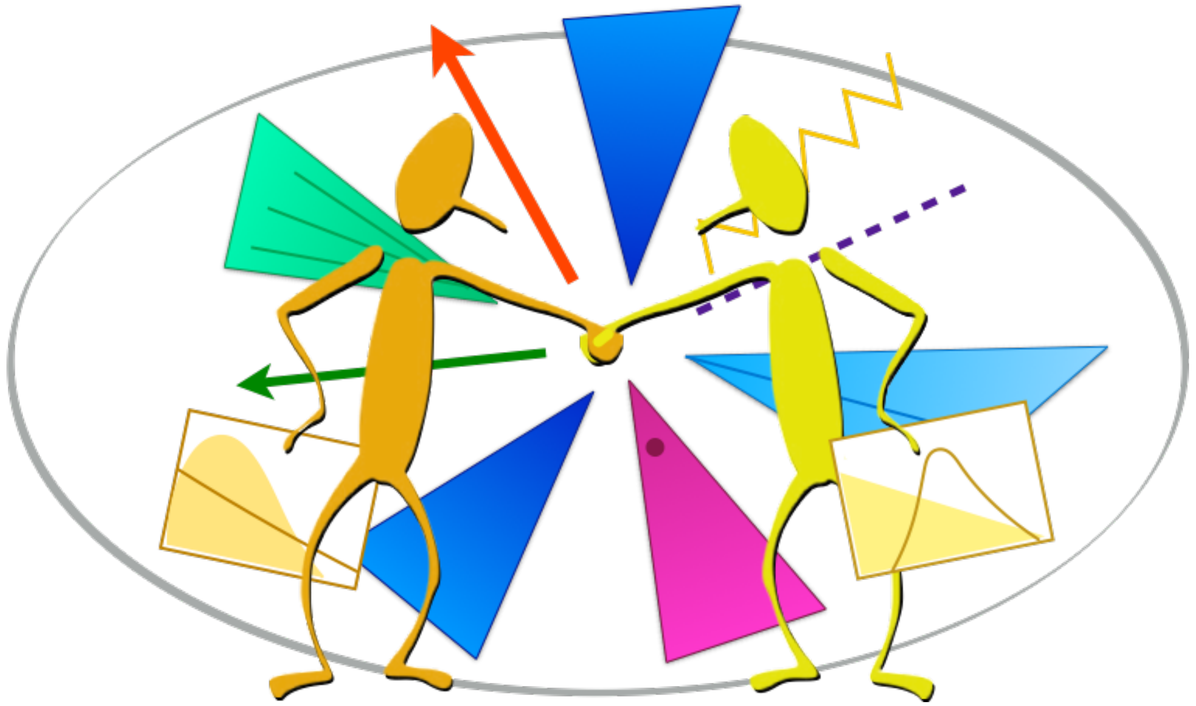
Making this a community effort

Activities and events:

- Discussions and work at the Les Houches PhysTeV workshops among phenomenologists and experimentalists (contributions in Les Houches 2015, 2017, 2019 proceedings).
- LHADA workshop, Grenoble, 25-26 Feb 2016
- LHADA workshop, CERN, 16-18 Nov 2016 (link)
- Discussions and activities within HSF data analysis WG and IRIS-HEP.
- 1st dedicated Analysis Description Languages for the LHC workshop (with experimentalists, phenomenologists, computing experts), Fermilab LPC, 6-8 May 2019 (link)
- 1st Data Analysis with ADL+CutLang School (3-7 Feb 2020), Istanbul (link)
- 2nd Analysis Description Languages workshop being planned for 2020 or 21.

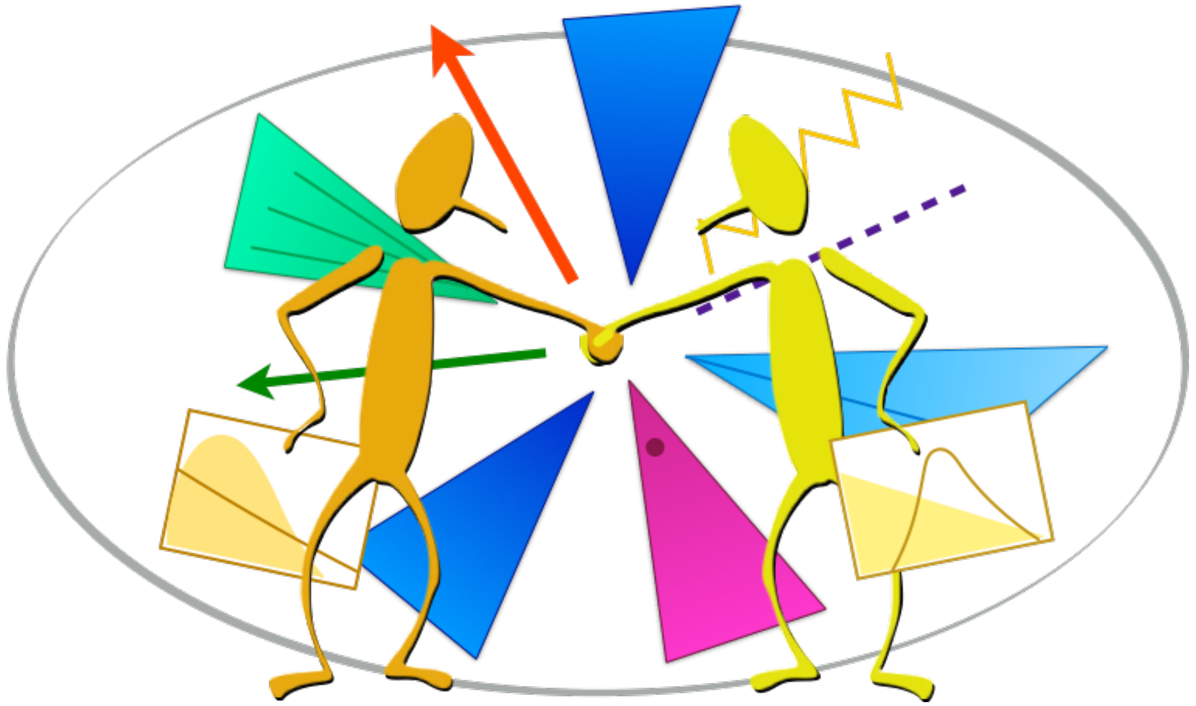
Discussion and contribution platforms:

- Frontiers journal research topic “Innovative Analysis Ecosystems for HEP data” within Big Data and AI in HEP, with a special focus on analysis description languages (link)
- Gitter forum for discussions (link)



To conclude

- Analysis description languages would greatly help to maximize the physics impact of LHC.
- Several examples using different approaches proved the feasibility of the concept.
- Work in progress. Still many intriguing problems to solve!
- Discussions will continue in forums like HSF, IRIS-HEP, Les Houches, LHC (Re)interpretation Forum, …
- This is a community effort. Everyone is welcome to join!



BACKUP: Syntax examples from existing languages

LHADA/CutLang —> ADL: syntax

Color legend:

new object,
variable, region
existing object
object attribute
existing variable
existing region
internal function
external function

Objects

```
# AK8 jets
object AK8jets
  take FatJet
  select pt > 200
  select abs(eta) < 2.4

# mass-tagged jets
object WjetsMassTag
  take AK8jets
  select msoftdrop [] 65 105

# W-tagged jets
object Wjets
  take WjetsMassTag
  select tau2 / tau1 <= 0.4
```

Variable definitions

```
define MR = fMR(megajets)
define Rsq = sqrt(fMTR(megajets, met) / MR)
define METl = MET + leptonsVeto[0]
define Rsql = sqrt(fMTR(megajets, METl) / MR)
define MT = fMT(leptonsVeto[0], MET)
define Mll = fMll(leptonsTight[0], leptonsTight[1])
define dphimegajets = dPhi(megajets[0], megajets[1])
```

Event selection regions

```
# preselection region
region preselection
  select size(AK4jets) >= 3
  select size(AK8jets) >= 1
  select MR > 800
  select Rsq > 0.08
```

```
# control region for tt+jets
region ttjetsCR
  select preselection
  select size(leptonsVeto) == 0
  select size(Wjets) >= 1
  select dphimegajets < 2.8
  select MT > 100
  select size(bjetsLoose) >= 1
  bin MR [] 800 1000 and Rsq [] 0.08 0.1
  bin MR [] 800 1000 and Rsq [] 0.1 0.2
  bin ...
```

From CMS SUSY razor analysis CMS-SUS-16-017
Phys.Rev.D97(2018) no.1, 012007, arxiv:1710.11188

Full implementation link

F.A.S.T. F.A.S.T analysis tools based on YAML: Syntax

Ben Krikler

Definitions: Combine uproot + numexpr, works with jagged arrays.

```
BasicVars:
  variables:
      - Muon_Pt: "sqrt(Muon_Px ** 2 + Muon_Py ** 2)"
      - IsoMuon_Idx: (Muon_Iso / Muon_Pt) < 0.10
      # This next variable will create a single
      # number for each event, using a set of inputs
      # whose length varies for each event
      - NIsoMuon:
          formula: IsoMuon_Idx
          reduce: count_nonzero
      - HasTwoMuons: NIsoMuon >= 2
      # Capture first muon's Pt, padded
      # with NaNs if NMuon < 1
      - Muon_lead_Pt: {reduce: 0, formula: Muon_Pt}
      # Capture second muon's Pt, padded
      # with NaNs if NMuon < 2
      - Muon_sublead_Pt: {reduce: 1, formula: Muon_Pt}
```

- Outputs a cutflow table with raw and weighted yields.
- Writes output data as pandas dataframes.

Event selection: specified as a nested dictionary

```
DiMu_controlRegion:
    weights: {nominal: weight}
    selection:
        All:
          - {reduce: 0, formula: Muon_pt > 30}
          - leadJet_pt > 100
          - All:
            - DiMuon_mass > 60
            - DiMuon_mass < 120
          - Any:
            - nCleanedJet == 1
            - DiJet_mass < 500
            - DiJet_deta < 2
```

Fill histograms

```
DiMuonMass:
  dataset_col: true
  binning:
      - in: DiMuon_Mass
        out: dimu_mass
        bins: {low: 60, high: 120, nbins: 60}
  weights: {weighted: EventWeight}
```

NAIL Natural Analysis Implementation Language: syntax

Andrea Rizzi

```
flow.Define("Muon_p4","@p4v(Muon)")
flow.Define("Electron_p4","@p4v(Electron)")
flow.Define("Electron_pid","Electron_pt*0+11")
flow.Define("Muon_pid","Muon_pt*0+13")
flow.MergeCollections("Lepton",["Muon","Electron"])
flow.Define("Lepton_index","Range(nLepton)")
flow.Distinct("LPair","Lepton")
flow.Selection("twoLeptons","nLepton>=2")
flow.Define("isOSSF","LPair0_charge != LPair1_charge && LPair0_pid == LPair1_pid",requires=["twoLeptons"])
flow.Selection("hasOSSF","Sum(isOSSF) > 0")
flow.TakePair("Z","Lepton","LPair","Argmax(-abs(MemberMap((LPair0_p4+LPair1_p4),M() )*isOSSF-91.2))",requires=["hasOSSF"])
flow.Selection("threeLeptons","nLepton>=3",requires=["twoLeptons","hasOSSF"])
flow.SubCollection("ResidualLeptons","Lepton",sel="Lepton_index != LPair0[Z_indices] && Lepton_index != LPair1[Z_indices]",r
flow.ObjectAt("ResidualLepton","ResidualLeptons","Argmax(ResidualLeptons_pt)")
flow.Define("MET_eta","0.f")
flow.Define("MET_mass","0.f")
flow.Define("MET_p4","@p4(MET)")
flow.Define("METplusLepton_p4","ResidualLepton_p4+MET_p4")
flow.Define("METplusLepton_Mt","METplusLepton_p4.Mt()")

histosPerSelection={
"threeLeptons" : ["METplusLepton_Mt"]
}

flow.binningRules = [
(".*Mt.*","100,0,1000")
]
```

In events with ≥3 leptons and a same-flavour opposite-sign lepton pair, find the best same-flavour opposite-sign lepton pair (mass closest to 91.2 GeV), and plot the transverse mass computed from the missing energy and the leading other lepton

Can handle complex analysis algorithms, systematic variations.

hep_tables and dataframe_expressions: syntax

Gordon Watts

Associating electrons with corresponding GEN particles:

```
mc_part = df.TruthParticles('TruthParticles')
mc_ele = mc_part[(mc_part.pdgId == 11) | (mc_part.pdgId == -11)]

eles = df.Electrons('Electrons')

def good_e(e):
    'Good electron particle'
    return (e.ptgev > 20) & (abs(e.eta) < 1.4)

good_eles = eles[good_e]
good_mc_ele = mc_ele[good_e]
```

Key line: for each good electron, select only the very near good MC electrons.

Filter by all that are near by for each electron we are “looking” at!

Build the data model to make life easy…

```
def associate_particles(source, pick_from):
    '''
    Associate each particle from source with a close by one from the particl

    Args:
        source            The particles we want to start from
        pick_from         For each partcile from source, we'll find a close
        name              Naming we can use when we extend the data model.

    Returns:
        with_assoc        The source particles that had a close by match
    '''
    def dr(p1, p2):
        'short hand for calculating DR between two particles.'
        return DeltaR(p1.eta(), p1.phi(), p2.eta(), p2.phi())

    def very_near(picks, p):
        'Return all particles in picks that are DR less than 0.1 from p'
        return picks[lambda ps: dr(ps, p) < 0.1]

    source[f'all'] = lambda source_p: very_near(pick_from, source_p)

    source[f'has_match'] = lambda e: e.all.Count() > 0
    with_assoc = source[source.has_match]
    with_assoc['mc'] = lambda e: e.all.First()

    return with_assoc

matched = associate_particles(good_eles, good_mc_ele)
```

Note the lambda capture!!!

otebook