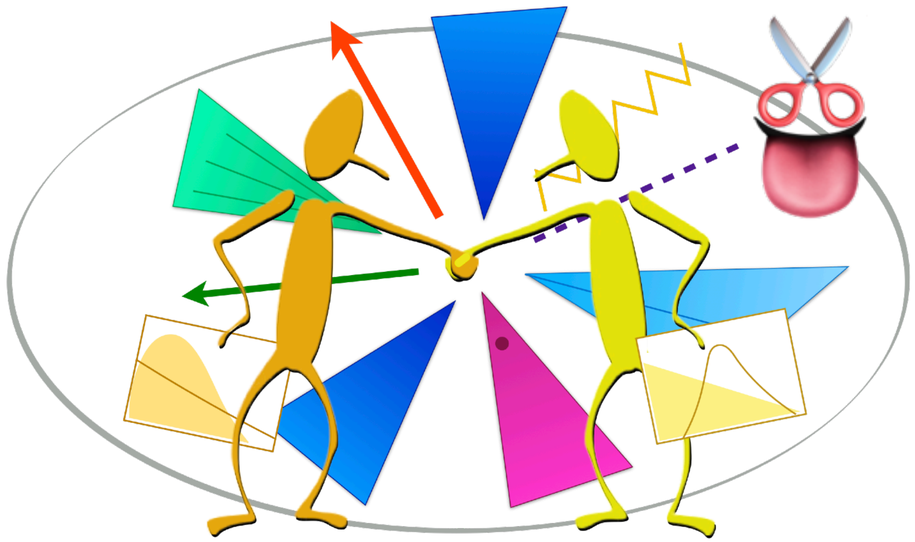
Documentation and references : cern.ch/adl



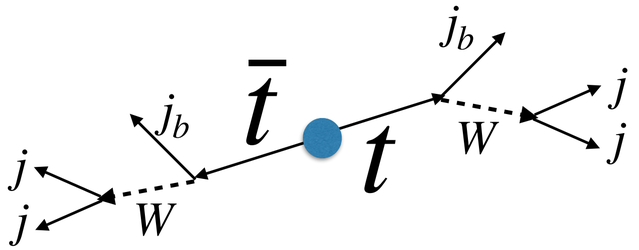
Extending ADL/CutLang with a new dynamic multipurpose protocol

Daniel Riley (Florida State U.)
Berare Göktürk (Boğaziçi U.)
Sezen Sekmen (Kyungpook Nat. U.)
Burak Şen (METU)
Gökhan Ünel (UC Irvine & Boğaziçi U.)
and the rest of the ADL/CutLang team

Towards physics-focused HEP data analyses

We traditionally perform analyses using analysis software frameworks:

- Frameworks are based on general purpose languages like C++ / Python,
- Physics content and technical operations are intertwined,
- Code hard to read, maintain and communicate.



with the χ^2 defined as:

$$\chi^2 = \frac{(m_{b_1 j_1 j_2} - m_{b_2 j_3 j_4})^2}{\sigma^2_{\Delta m_{bjj}}} + \frac{(m_{j_1 j_2} - m_W^{\mathrm{MC}})^2}{\sigma^2_{m_W^{\mathrm{MC}}}} + \frac{(m_{j_3 j_4} - m_W^{\mathrm{MC}})^2}{\sigma^2_{m_W^{\mathrm{MC}}}}.$$

Could there be an alternative way that...

- Allows more direct interaction with data
- Decouples the physics information from purely technical tasks, thereby shifting the focus to the physics algorithm
- Improves the clarity and accessibility of analysis logic, and thereby its communicability and preservation?

Analysis Description Language for HEP

ADL is a declarative domain specific language (DSL) that describes the physics content of a HEP analysis in a standard and unambiguous way.

- External DSL: Custom-designed syntax to express analysis-specific concepts. Reflects conceptual reasoning of particle physicists. Focus on physics, not on programming.
- Declarative: States what to do, but not how to do it.
- Easy to read: Clear, self-describing syntax.
- Designed for everyone: experimentalists, phenomenologists, students, interested public…

ADL is framework-independent —> Any framework recognizing ADL can perform tasks with it.

- Decouples physics information from software / framework details.
- Multi-purpose use: Can be automatically translated or incorporated into the GPL / framework most suitable for a given purpose, e.g. exp. analysis, (re)interpretation, analysis queries, …
- Easy communication between groups: exp., pheno, referees, students, public, …
- Easy preservation of analysis logic.

The ADL construct

ADL consists of

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

- ADL file consists of blocks separating object, variable and event selection definitions. Blocks have a keyword-instruction structure.
  - keywords specify analysis concepts and operations.

```
blocktype blockname
  keyword1 instruction1
  keyword1 instruction2
  keyword3 instruction3 # comment
```

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

ADL syntax with usage examples: link

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), Front. Big Data 4:659986, 2021
Several proceedings for ACAT and vCHEP

A very simple analysis example with ADL

```
# OBJECTS
object goodMuons
  take muon
  select pT(muon) > 20
  select abs(eta(muon)) < 2.4

object goodEles
  take ele
  select pT(ele) > 20
  select abs(eta(ele)) < 2.5

object goodLeps
  take union(goodEles, goodMuons)

object goodJets
  take jet
  select pT(jet) > 30
  select abs(eta(jet)) < 2.4
  reject dR(jet, goodLeps) < 0.4
```

```
# EVENT VARIABLES
define HT = sum(pT(goodJets))
define MTl = Sqrt( 2*pT(goodLeps[0]) * MET*(1-cos(phi(METLV[0]) - phi(goodLeps[0]) )))

# EVENT SELECTION
region baseline
  select size(goodJets) >= 2
  select HT > 200
  select MET / HT <= 1

region signalregion
  baseline
  select Size(goodLeps) == 0
  select dphi(METLV[0], jets[0]) > 0.5

region controlregion
  baseline
  select size(goodLeps) == 1
  select MTl < 120
```


ADL implementations of some LHC analyses: https://github.com/ADL4HEP/ADLLHCanalyses

CutLang runtime interpreter and framework

CutLang runtime interpreter:

- No compilation. User writes an ADL file and runs CutLang directly on events.
- CutLang itself is written in C++, works in any modern Unix environment.
- Based on ROOT classes for Lorentz vector operations and histograms.
- ADL parsing by Lex & Yacc.

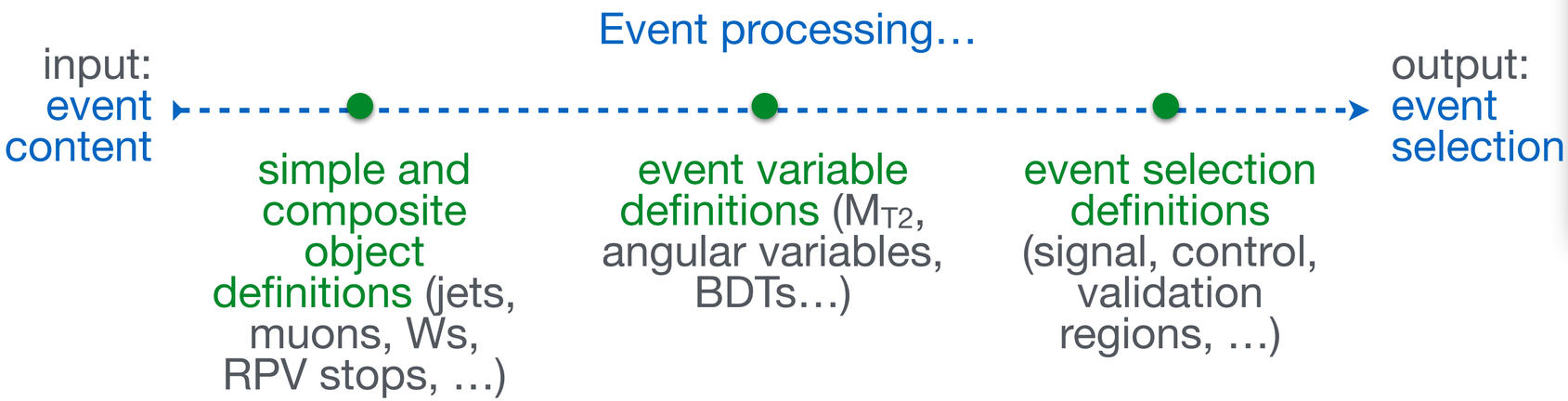
CutLang framework: interpreter + tools

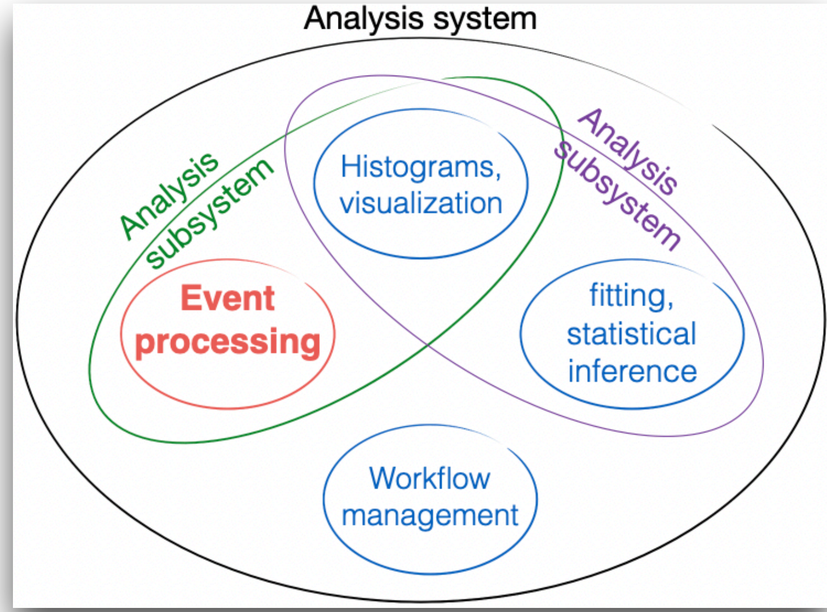
- Input events via ROOT files.
  - multiple input formats: Delphes, CMS NanoAOD, ATLAS/CMS Open Data, LVL0, FCC. More can be easily added.
  - All event types converted into predefined particle object types. —> can run the same ADL file on different input types.
- Includes many internal functions.
- Output in ROOT files: ADL file, cutflows, bins and histograms for each region in a separate directory.
- Available in Docker, Conda, Jupyter (via Conda or binder). (win/lin/mac + portables)

CutLang Github repository: https://github.com/unelg/CutLang
Comput.Phys.Commun. 233 (2018) 215-236 (arXiv:1801.05727),
Front. Big Data 4:659986, 2021 (arXiv:2101.09031),
Several proceedings for ACAT and vCHEP

ADL scope

- Event processing: *Priority focus!*

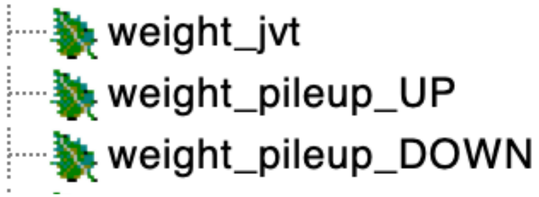




- Analysis results, i.e. counts and uncertainties: Available
- Histogramming: Available => HistoSets, 1D, 2D, variable width...
- Systematic uncertainties: ATLAS type syntax now available.
- Data/MC comparison, limits: Within the scope, implementation being tested.
- Operations with selected events, e.g. background estimation, scale factor derivation: Very versatile. Not yet within the scope.

Systematics in ADL *(ATLAS style)*

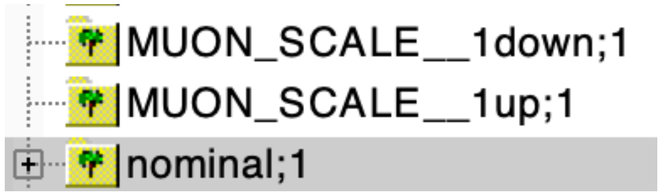
- All necessary information already in the NTUPLE (incl. up & down variations)
  - as event weights (TBranch)



in ADL file

ON/OFF	Name of the UP systematics	Name of the DOWN systematics	Name of the nominal branch
systematic ON	"weight_pileup_UP"	"weight_pileup_DOWN"	weight_pileup
systematic ON	"weight_leptonSF_EL_SF_Trigger_UP"	"weight_leptonSF_EL_SF_Trigger_DOWN"	weight_leptonSF
systematic ON	"weight_leptonSF_EL_SF_Reco_UP"	"weight_leptonSF_EL_SF_Reco_DOWN"	weight_leptonSF
systematic ON	"weight_leptonSF_EL_SF_ID_UP"	"weight_leptonSF_EL_SF_ID_DOWN"	weight_leptonSF
systematic ON	"weight_leptonSF_EL_SF_Isol_UP"	"weight_leptonSF_EL_SF_Isol_DOWN"	weight_leptonSF
systematic ON	"weight_leptonSF_MU_SF_Trigger_STAT_UP"	"weight_leptonSF_MU_SF_Trigger_STAT_DOWN"	weight_leptonSF
systematic ON	"weight_leptonSF_MU_SF_Trigger_SYST_UP"	"weight_leptonSF_MU_SF_Trigger_SYST_DOWN"	weight_leptonSF
systematic ON	"weight_leptonSF_MU_SF_ID_STAT_UP"	"weight_leptonSF_MU_SF_ID_STAT_DOWN"	weight_leptonSF

- as full event data (TTree)



in ADL file

```
systematic ON       "MET_SoftTrk_ResoPerp"              "MET_SoftTrk_ResoPara"              ttree
systematic ON       "MET_SoftTrk_ScaleUp"               "MET_SoftTrk_ScaleDown"             ttree
systematic ON       "MUON_ID__1up"                      "MUON_ID__1down"                    ttree
systematic ON       "MUON_MS__1up"                      "MUON_MS__1down"                    ttree
systematic ON       "MUON_SAGITTA_RESBIAS__1up"         "MUON_SAGITTA_RESBIAS__1down"       ttree
systematic ON       "MUON_SAGITTA_RHO__1up"             "MUON_SAGITTA_RHO__1down"           ttree
systematic ON       "MUON_SCALE__1up"                   "MUON_SCALE__1down"                 ttree
```

ADL helps to design and document a single analysis in a clear and organized way.

BONUS: Library functions guaranteed to be bug free*

WYGIWYS analysis, no double counting, correct sorting, $\chi 2$ evaluation, combinatorics, unions...

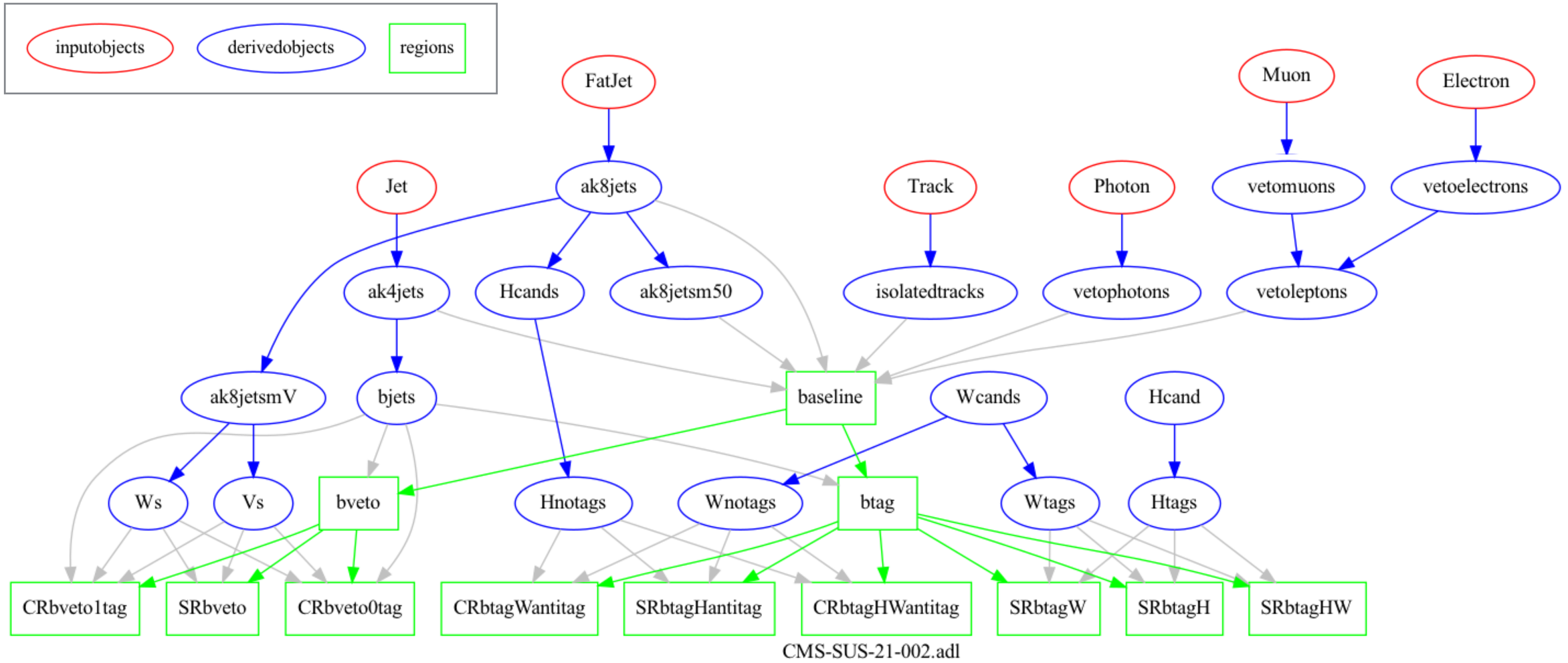
Its distinguishing strength is in navigating and exploring the multi-analysis landscape.

* as much as possible

Uses for analyses written with ADL

- Use existing analyses to design new ones: Answer questions such as
  - *"Which final states **did** the existing analyses look at?"*
  - *"Which final states **are** unexplored?"*
  - *"How much overlap exists **between my analysis and the existing ones**?"*
- Use existing objects:
  - Directly implement in a new analysis, compare analyses choices, work with definition of the same object in different data tiers.
- Visualize & review analyses:
  - Build graphs and tables from analyses using automated tools. *(next page)*
- Query analysis or object databases: Answer via automated query tools questions such as
  - *"Which analyses require missing ET > at least 300?"*
  - *"Which analyses use b-jets tagged with criterion X? ", "Which muons use isolation?"*
- Compare / combine analyses: Determine analysis overlaps, identify disjoint analyses or search regions; find the feasible combinations with maximal sensitivity; automate large scale combinations of analyses.
- Reinterpretation: Reimplement & validate analyses for reinterpretation in new models and parameter space regions.
- Education: Provide a learning database for students.

Auto-generated graph of an ADL analysis (using graphviz)



arXiv:2205.09597: CMS Search for Electroweak SUSY in WW, WZ and WH hadronic final states

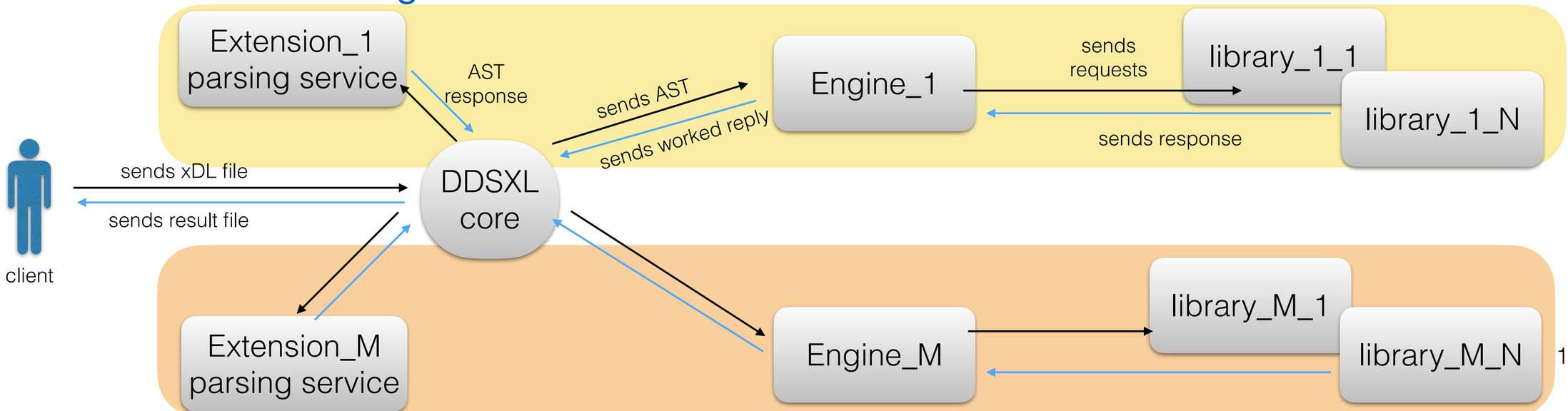
Flexible Function & Particle Encoding

- The current ADL / CutLang structure has been around for ~5 years.
  - It has grown into a complex monolithic structure. Adding new variable, function, ... requires lengthy edits
- We are decoupling the grammar from the DSL engine
  - Function and particle names should no longer be hardcoded in ADL
  - After initial parsing, the DSL should match function and particle names to those within an external library
- This approach has serious advantages:
  - This allows for a more flexible system, which does not require direct maintenance on the core code
  - Easily link to different function implementations
  - Improves portability between data formats
  - All attributes (pT, eta, etc…) are removed from the grammar
  - New attributes can also be linked from an external library
    - Libraries can be specified at the CLI
      - ./adl --attr-lib attr.lib --particle-lib part.lib experiment.adl

not yet merged with main branch

DDSXL

- After DSL-grammar decoupling, next is multiple grammars for multiple domains.
- We designed a new protocol called Dynamic Domain Specific eXtensible Language (DDSXL)
  - it can contain numerous programming languages and frameworks.
    - each developer to integrate their own module independently from other modules
    - 3 independent developer types: maximum efficiency for the developers
    - it allows each micro team to use/integrate solutions they are experts in
  - it integrates a domain ecosystem (such as CL) into the development environment
  - a set of rules determined through communication over the network.



DDSXL components

- DDSXL Core
  - main service & entry point, always alive, no dependencies
  - all packages register to this service
- Extension (ADL)
  - produces an Abstract Syntax Tree (AST) for the associated engine
- Engine (CutLang)
  - receives the AST, can do basic arithmetic & logic,
  - depends on Library/ies for specific functions
- Library/ies (ML functions, complex kinematic variable functions...)
  - offers recipes for specific functions,
  - can be many running on different hosts / addresses

DDSXL development & status

```
region   testZ
  select      ALL              # to count all events
  select      Size (ELE) >= 2  # events with 2 or more electrons
  histo       h1mReco, "Z candidate mass (GeV)", 100, 0, 200, {ELE_0 ELE_1}m
  select      {ELE[0] ELE[1] }q == 0   # Z is neutral
  histo       h2mReco, "Z candidate mass (GeV)", 100, 0, 200, {ELE_0 ELE_1}m
```

An example of an ADL script on the top, and an example of AST output on the bottom

- Developer types in DDSXL excosystem
  - Core developer: experts in RPC, network communications, etc...
  - Extension developer: specializes in parsers, compilers, AST etc...
  - Engine developer: experts in the relevant domain that can solve problems
  - Library developer: researchers in the relevant domain only
- Status
  - Execution protocol steps and technologies to be used are identified
    - gRPC (https://grpc.io/) & GraphQL (https://graphql.org/)
  - Test servers and clients are written, functionality validated
  - Run time library addition successfull
  - Development ongoing

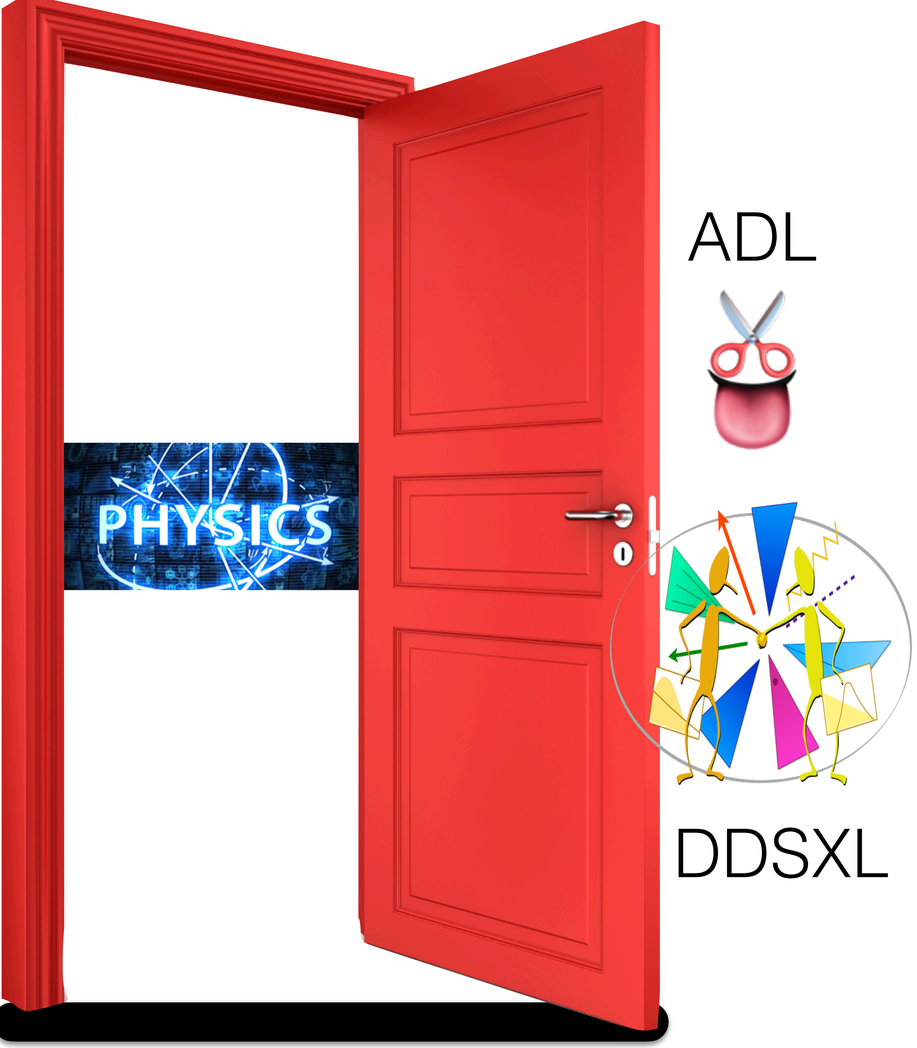

```
{
  "regions": {
    "testZ": [
      {
        "type": "select",
        "command": "ALL"
      },
      {
        "type": "select",
        "command_1": {
          "func": "S",
          "entry": "ELE"
        },
        "condition": ">=",
        "command_2": "2"
      },
      {
        "type": "histo",
        "command": {
          "name": "h1mReco",
          "title": "Z candidate mass (G
          "axis": "100,0,200",
          "select": {
            "func": "M",
            "entry": "ELE_0 ELE_1"
          }
        }
      },
      {
        "type": "select",
        "command_1": {
          "func": "Q",
          "entry": "ELE_0 ELE_1"
        },
        "condition": "==",
        "command_2": "0"
      },
```


```
_entities(
  representations: [_Any!]!
): [_Entity]!

UNION DETAILS

union _Entity =
  CutlangStd
  FMt2
```

Documentation of graphql types of cutlang-std and f-mt2 packages that extend each other on CutLang engine (graphql playground tool is used)

```
@Module({
  imports: [
    GraphQLModule.forRoot<ApolloGatewayDriverConfig>({
      driver: ApolloGatewayDriver,
      gateway: {
        supergraphSdl: new IntrospectAndCompose({
          subgraphs: [
            { name: 'cutlang-std', url: 'http://localhost:8001/graphql' },
            { name: 'f-mt2', url: 'http://localhost:8002/graphql' },
          ],
        }),
      },
    }),
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

A type example that extends the algorithm by collecting the types of the cutlang-std and f-mt2 packages (graphql is used, and the gateway is built on the CutLang engine)



ADL

DDSXL

backup slides
1. ADL syntax
2. Use cases
3. Q&A

To conclude:

- ADL is an emerging, paradigm-shifting approach that puts physicists and physics at the center of HEP data analysis

  - CutLang is the first successful runtime interpreter for ADL
  - Research & education uses confirm the feasibility of ADL
- ADL syntax and tools are under constant development.
  - Grammar - DSL Engine being decoupled
  - DDSXL protocol is being developed to adress
    - ease of development, ease of portability
    - application to multiple domains & functions
- Join the mattermost channel to explore ADL/CutLang and provide feedback:

ADL / CL is a community effort !
Everyone is welcome to join the development of the language and tools.

ADL syntax: main blocks, keywords, operators

Block purpose	Block keyword
object definition blocks	object
event selection blocks	region
analysis or ADL information	info
tabular information	table

Keyword purpose	Keyword
define variables, constants	define
select object or event	select
reject object or event	reject
define the mother object	take
apply weights	weight
bin events in regions	bin, bins
sort objects	sort
define histograms	histo
save variables for events	save

Operation	Operator
Comparison operators	> < => =< == != [] (include)][(exclude)
Mathematical operators	+ - * / ^
Logical operators	and or not
Ternary operator	condition ? truecase : falsecase
Optimization operators	~= (closest to) ~! (furthest from)
Lorentz vector addition	LV1 + LV2 LV1 LV2

Syntax also available to write existing analysis results (e.g. counts, errors, cutflows…).

Syntax develops further as we implement more and more analyses.

ADL syntax rules with usage examples: https://twiki.cern.ch/twiki/bin/view/LHCPhysics/ADL

ADL syntax: functions

Standard/internal functions: Sufficiently generic math and HEP operations could be a part of the language and any tool that interprets it.

- Math functions: abs(), sqrt(), sin(), cos(), tan(), log(), …
- Collection reducers: size(), sum(), min(), max(), any(), all(),…
- HEP-specific functions: dR(), dphi(), deta(), m(), ….
- Object and collection handling: union(), comb()…

External/user functions: Variables that cannot be expressed using the available operators or standard functions would be encapsulated in self-contained functions that would be addressed from the ADL file and accessible by compilers via a database.

- Variables with non-trivial algorithms: M_{T2}, aplanarity, razor variables, …
- Non-analytic variables: Object/trigger efficiencies, variables/efficiencies computed with ML, …

ADL syntax rules with usage examples: https://twiki.cern.ch/twiki/bin/view/LHCPhysics/ADL

Physics with ADL

Designing new analyses:

- Experimental analyses:
  - 2 ATLAS EXO analyses ongoing
- Phenomenology studies:
  - E6 isosinglet quarks at HL-LHC & FCC w/ CutLang (Eur Phys J C 81, 214 (2021))
- Analysis of LHC Open Data:
  - Tutorial with full implementation of a CMS vector-like quark analysis with 2015 data for CMS Open Data workshop : link
  - Other exercises for earlier workshops and the CERN summer student programme.
- Analysis optimization via differentiable programming (under development).

Using existing analyses:

ADL analysis database with ~15 LHC analyses:
https://github.com/ADL4HEP/ADLLHCanalyses
(more being implemented).

Validation of these analyses in progress.

- Reinterpretation studies:
  - Integrating ADL into the SModelS framework
- Analysis queries, comparisons, combinations:
  - Automated tools under development
- Long term analysis preservation

ADL for reinterpretation

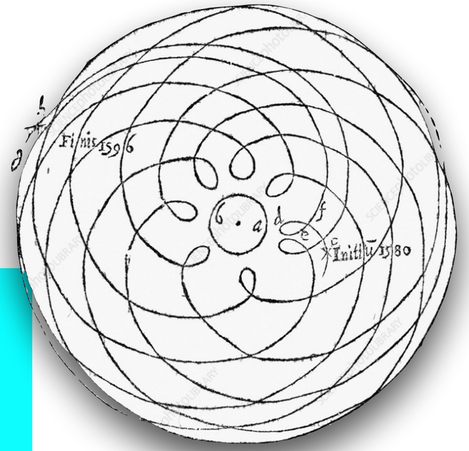
ADL allows practical exchange of experimental analysis information with the pheno community.

- Clear description of the complete analysis logic.
- Enables straightforward adaptation from experiments to public input event formats.
  - Biggest difficulty is in reproducing an analysis is adapting object definitions:
    In ADL, e.g. just swap experimental object IDs with numeric efficiency maps.
  - Event selections stay ~the same (can swap trigger selections with efficiencies)
- Generic structure available for expressing analysis output in the ADL file:
  Data counts, BG estimates, signal predictions —> counts, uncertainties, cutflows.
  - Running CutLang puts preexisting results in histograms with the same format as the run output. —> Direct comparison of cutflows, limit calculations.
- ADL could facilitate providing information on analysis results to HEPDATA or similar platforms.

but... I like my epicycles!

Q & A

- **This is yet another syntax to learn, valid only for HEP.**
  - It is english + mathematics + logic.
  - will be extended to other data processing jobs too.
- **Python is so simple!**
  - It is not the language but the person who programs with it
- **We have lots of Python libraries**
  - We had lots of horses, yet we use cars now.
- **We have Rivet / MadAnalysis / CheckMate**
  - These are glorified libraries not paradigm shifters
- **But I want my students to know Python / C++! It will help them to find a job.**
  - If these are Phys students, we should help them do physics, if they are computero-philes, they should learn programming from professionals.
  - Writing consecutive if statements is not what you want.



```
double HT = 0.;
for(int j = 0; j < hardjets.size(); j++)
   HT += hardjets[j]->PT;
double mEffincl = HT + missingET->PT;

double mEff2 = missingET->PT + jets[0]->PT + jets[1]->PT;
double rEff2 = missingET->PT/mEff2;

double mEff3 = 0;
if (jets.size() >= 3)
    mEff3 =  mEff2 + jets[2]->PT;
double rEff3 = 0;
if (jets.size() >= 3)
    rEff3 = missingET->PT/mEff3;

double mEff4 = 0;
if (jets.size() >= 4)
    mEff4 =  mEff3 + jets[3]->PT;
double rEff4 = 0;
if (jets.size() >= 4)
    rEff4 = missingET->PT/mEff4;
```

CheckMate

Rivet

```
const Jets jets =
  applyProjection<JetAlg>(evt, "Jets").jetsByPt(20*GeV);
foreach (const Jet& j, jets) {
  foreach (const Particle& p, j.particles()) {
    const double dr =
      deltaR(j.momentum(), p.momentum());
  }
}
```

MadAnalysis

```
// Muons
for(unsigned int mu=0; mu<event.rec()->muons().size(); mu++)
{
  const RecLeptonFormat *CurrentMuon = &(event.rec()->muons()[mu]);
  if(CurrentMuon->pt()>10. && fabs(CurrentMuon->eta())<2.7)
    Muons.push_back(CurrentMuon);
}
```