APACHECON NORTH AMERICA
OCTOBER 3 - 6, 2022
NEW ORLEANS
LOUISIANA
WWW.APACHECON.COM

How Functional Programming leads to tight C-code at Runtime

APACHE Daffodil™

Mike Beckerle, Principal Engineer
Apache Daffodil PMC
mbeckerle at apache.org

OWL Cyber Defense

Outline/Agenda

- What is DFDL and What is Apache Daffodil?
- Daffodil's DFDL *schema compiler*
  - Functional programming
  - Object-Oriented Lazy Attribute Grammars (OOLAG).
- New runtime environment
  - C-code generator
    - C-language source code for standard C compilers
- Extras:
  - new Daffodil VSCode-based data debugger
  - EXI - dense binary "XML"

What is DFDL
(Data Format Description Language)

and

What is Apache Daffodil?

Got EDIFACT Data?

```
UNA:+.?*'
UNB+UNOC:4+5790000274017:14+5708601000836:14+990420:1137+17++INVOIC++++1'
UNH+30+INVOIC:D:03B:UN'
BGM+380+539602'
DTM+137:19990420:102'
RFF+CO:01671727'
NAD+BY+5708601000836::9'
RFF+VA:UK37499919'
NAD+SU++IBM UK'
RFF+VA:UK19430839'
RFF+ADE:00000767'
NAD+DP+++MyCompany+MyStreet+MyTown++1234+UK'
CUX+2:GBP:9'
LIN+1++V0370246:IN'
```

Got bit-packed binary data ?

Bytes are

09 20 42 F0 0D B8 DD

Fields are decribed as:

Message Number	**XXXXXX00 00001xxx**
FPI for Message Subtype	**XXXXX0xx**
FPI for File Name	**XXXX0xxx**
FPI for Message Size	**XXX0XXXX**
Operation Indicator	**X01XXXXX**
Retransmit Indictor	**0XXXXXXX**
Message Precendence Codes	**XXXXX010**
Security Classification	**XXX00XXX**

Got NACHA Data?

```
101 121000248 1210002480608080107A094101WFB-W EDI CUST. DATA   WFB-E ACH SPINAT DATA
5200APD TX/FINCL SVC323413684           9666666606CCDAPD - TAX 0608020608022141021000024030649
62710700543200004001191477   0542151200614007046488KD8BRODY LABORATORIES INCFS0021000024030840
820000000100107005430005421512000000000000009666666606                         021000024030649
5200ARAR                                9000290001CCD PAYMENT        0608072191021000027294149
62205100141200004945037059   00004036762394128        VIA LICENSING CORP      0021000027294283
820000000100051001410000000000000000004036769000290001                         021000027294149
5200ACME   WORLDWIDEDIRECT DEPOSIT      9954245682CCDA/P             0608022141122000030000219
62210700543200004001191477   000005000010000142611008 100815047371712006      0122000036548030
62210700543200004001191477   000014750010000142611008 100815047375772006      0122000036548031
820000000200214010860000000000000000001975009954245682                         122000030000219
5200BEST BANK NA    5046001042958       9560900031CCDEDI PAYMNT      0511181231021101100000014
6220510014149995275638771    0000037500504058967      XXXXXXXX BRANCH INC     1021101100001681
705RMR*OI*0140611**-1170.49*25*1170.36*
9999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
```

Got ISO8583 Data?

```
1111FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1912345678901234567
8912345612345678901212345678901212345678901212312359591
2345678123456781234567812345699123112000099129912991231
1231123112341231231 23A1B2C3D4E5F61231231234123419912311
2312345678901234567890123408C00015001112345678901111234
567890128123456789012345678901234567826;111111111111111
11=1215=?1062;112222222222222222222=1231123412341234121
123456112121212341?1A1B2C3D4E5F6A1B2C3123123A1A#A1A#A1#
A1#A1#A1#A1#15A1#A1#A1#A1#A1#15A1#A1#A1#A1#A1#35%A11111
111111111111^JOHNDOE^1215^?015A1#A1#A1#A1#A1#015A1#A1#A
1#A1#A1#015A1#A1#A1#A1#A1#ABCABCABC08008123456780080111
2301100110011001100110011
```

DFDL = Data Format Description Language

- A standard from Open Grid Forum (OGF)
- Started 2001, Ratified 2022
- Big - 200+ pages
- **DFDL → DaFfoDiL**

- DFDL is *mostly* not new ideas
  - *Standardizes existing practice* of data integration tools 1995 - 2010
- DFDL has some innovations
  - Especially for *unparsing binary data*

GFD-R-P.240
OGF DFDL WG
dfdl-wg@ogf.org

Michael J Beckerle, Owl Cyber Defense/Tresys
Stephen M Hanson, IBM
February 2021

Data Format Description Language (DFDL) v1.0 Specification

Status of This Document
Grid Final Draft (GFD)

Obsoletes
This document incorporates all errata and clarifications to earlier DFDL v1.0 specification documents and therefore obsoletes both:

- GFD-P-R.207 dated September 2014 [OBSOLETE_DFDL_207]
- GFD-P-R.174 dated January 2011 [OBSOLETE_DFDL_174]

Copyright Notice
Copyright © Global Grid Forum (2004-2006). Some Rights Reserved. Distribution is unlimited.
Copyright © Open Grid Forum (2006-2021). Some Rights Reserved. Distribution is unlimited.

Abstract
This document provides a definition of a standard Data Format Description Language (DFDL). This language allows description of text, dense binary, and legacy data formats in a vendor-neutral declarative manner. DFDL is an extension to the XML Schema Description Language (XSD).

Use Daffodil: NACHA as JSON Please…

```
{ "ACHFile":
{ "FileHeaderRecord":
{ "RecordTypeCode": "1",
"PriorityCode": "01",
"ImmediateDestination": " 123456789",
"ImmediateOrigin": " 987654321",
"FileCreationDate": "071030",
"FileCreationTime": "1634",
"FileIdModifier": "A",
"RecordSize": "094",
"ImmediateDestinationName": "TEST Destination ",
"ImmediateOriginName": "TEST Origination ", "ReferenceCode": " " },
"Batch": [ { …
```

Use Daffodil: NACHA as XML Please…

```
<ACHFile xmlns="ach:2013">
<FileHeaderRecord>
<RecordTypeCode>1</RecordTypeCode>
 <PriorityCode>01</PriorityCode>
<ImmediateDestination> 123456789</ImmediateDestination>
<ImmediateOrigin> 987654321</ImmediateOrigin>
<FileCreationDate>071030</FileCreationDate>
<FileCreationTime>1634</FileCreationTime>
<FileIdModifier>A</FileIdModifier>
<RecordSize>094</RecordSize>
<ImmediateDestinationName>TEST Destination </ImmediateDestinationName>
<ImmediateOriginName>TEST Origination </ImmediateOriginName>
<ReferenceCode> </ReferenceCode>
</FileHeaderRecord> ...
```

Introduction to

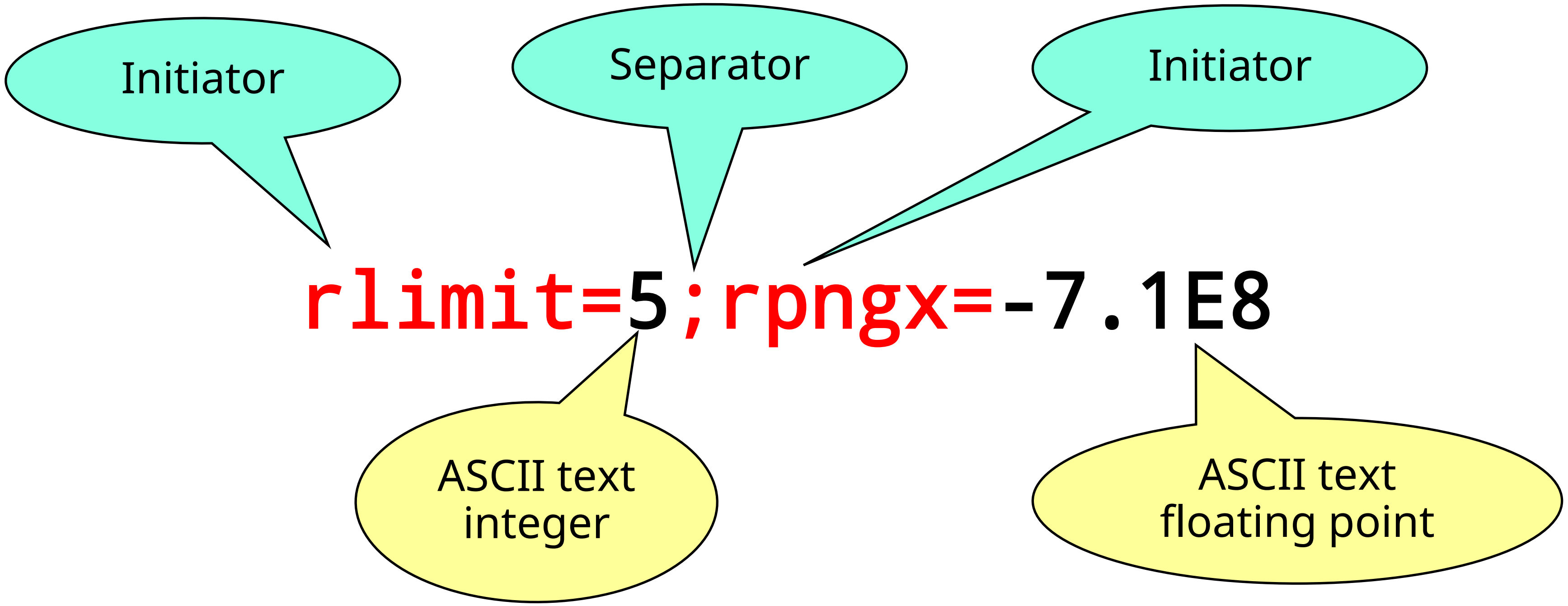
Data Format Description Language

aka DFDL

Example – Delimited Text Data

```
rlimit=5;rpngx=-7.1E8
```

Example – Delimited Text Data



Initiator

Separator

Initiator

`rlimit=5;rpngx=-7.1E8`

ASCII text integer

ASCII text floating point

- Separators, initiators (aka tags), & terminators are all examples in DFDL of *delimiters*
- Delimiters are one kind of *Framing.*
- DFDL divides the data into *content* (becomes values) and Framing (surrounds values)

DFDL Schema

```
<complexType name=“rPair">
  <sequence>
    <element name=“rlim" type="xs:int"/>

    <element name=“rpng" type="xs:float"/>

  </sequence>
</complexType>
```

Logical Elements

Simple Type

DFDL Schema

OWL Cyber Defense

Top level format declaration block applies to this entire schema *file.*

```
<annotation>
  <appinfo source="http://www.ogf.org/dfdl/">
     <dfdl:format representation="text"
             textNumberRep="standard" encoding="ascii"
             lengthKind="delimited“ .../>
  </appinfo>
</annotation>

<complexType name=“rPair">
  <sequence dfdl:separator=";">
    <element name="rLim" type="xs:int"
                 dfdl:initiator=“rLimit=" />
    <element name=“rpng" type="xs:float"
                 dfdl:initiator=“rpngx=" />
  </sequence>
</complexType>
```

;

rLimit=5

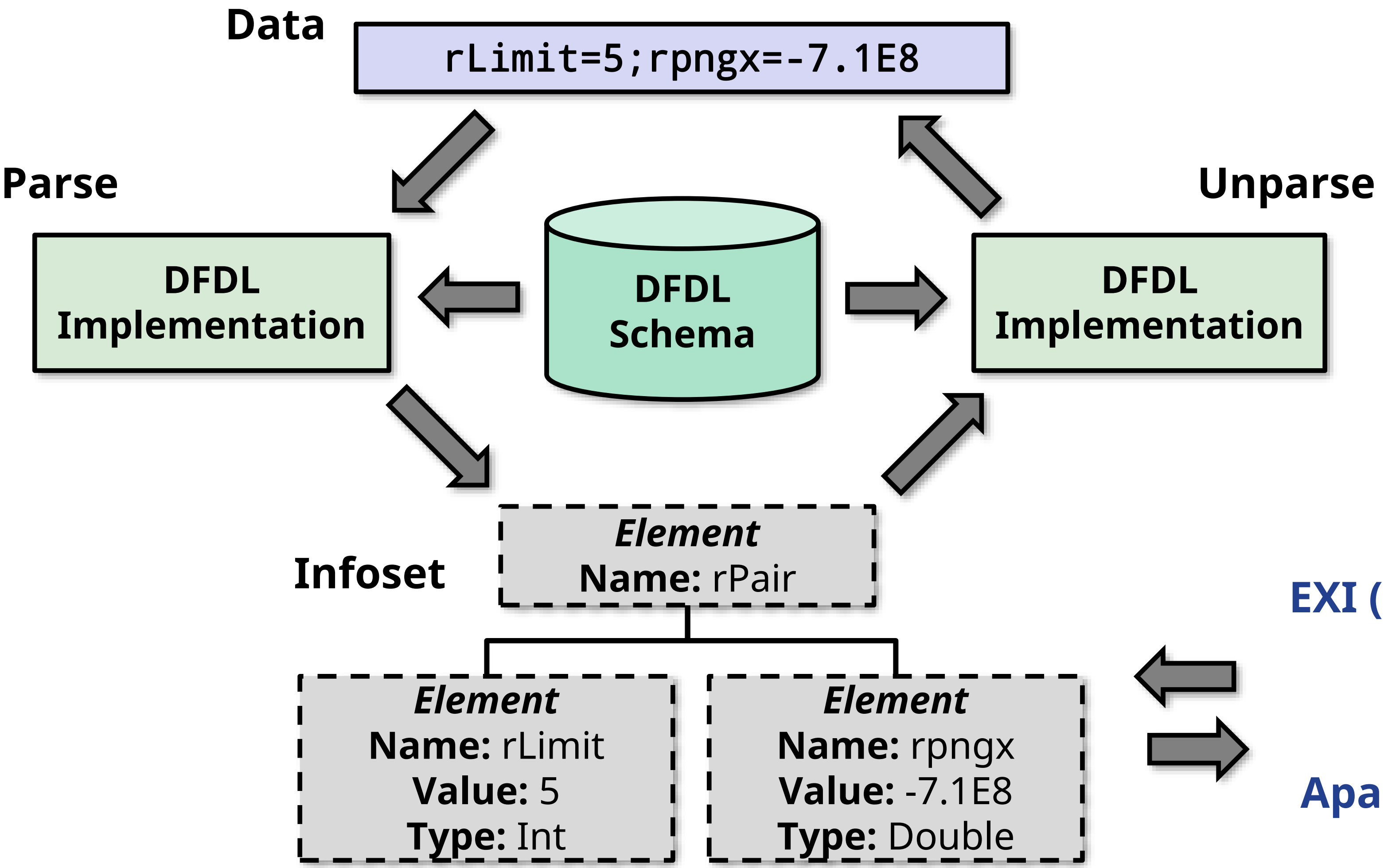
rpngx=-7.1E8

DFDL properties

APACHECON NORTH AMERICA OCTOBER 3 - 6, 2022 NEW ORLEANS LOUISIANA WWW.APACHECON.COM

APACHE Daffodil

DFDL Data and Infoset Lifecycle

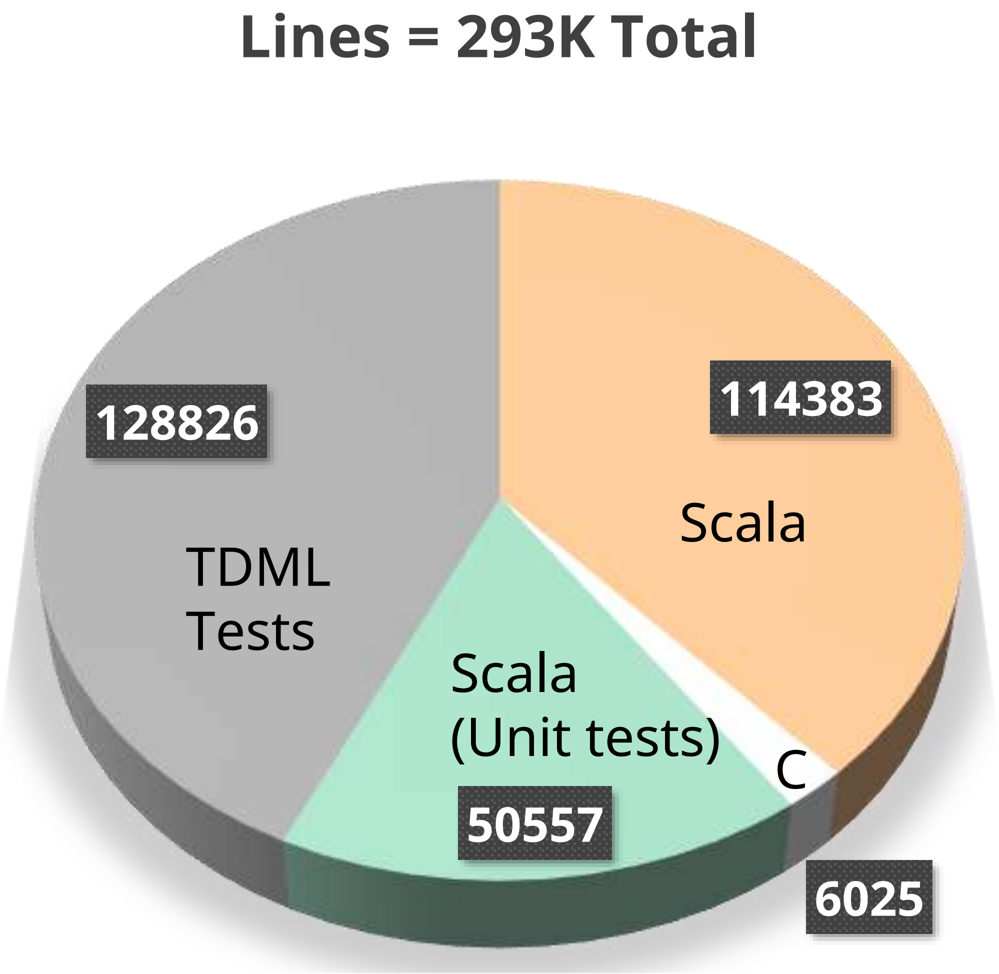


XML
or
EXI (binary XML) ***new!***
or
JSON
or
Apache NiFi Records
or
Apache Drill ….
etc.

Internals of Apache Daffodil

Apache Daffodil

- Daffodil contains
  - Full-blown compiler for DFDL schemas
  - JVM-based low-level runtime for parse/unparse
  - Big test suite (TDML)
  - New: C-code generating runtime for parse/unparse
- Written in Scala
  - Extensive use of Functional Programming



Apache Daffodil

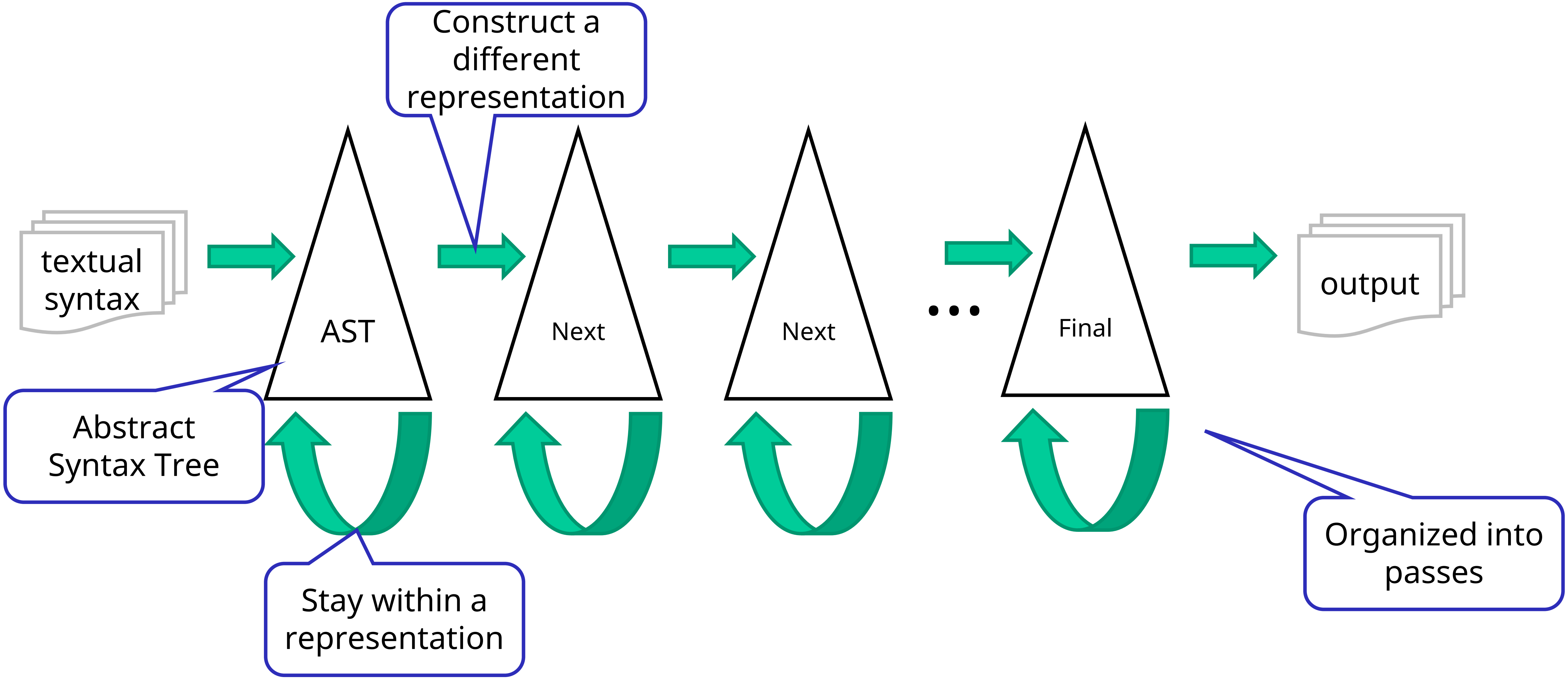
If you download it, what do you get?

- Jar libraries – runs on JVM
  - ***DFDL Schema Compiler***, runtime, utilities, TDML runner
  - Signed Jars available from Maven Central
  - Java & Scala API with documentation
  - new: C-generator backend
    - (today: handles small subset of DFDL)

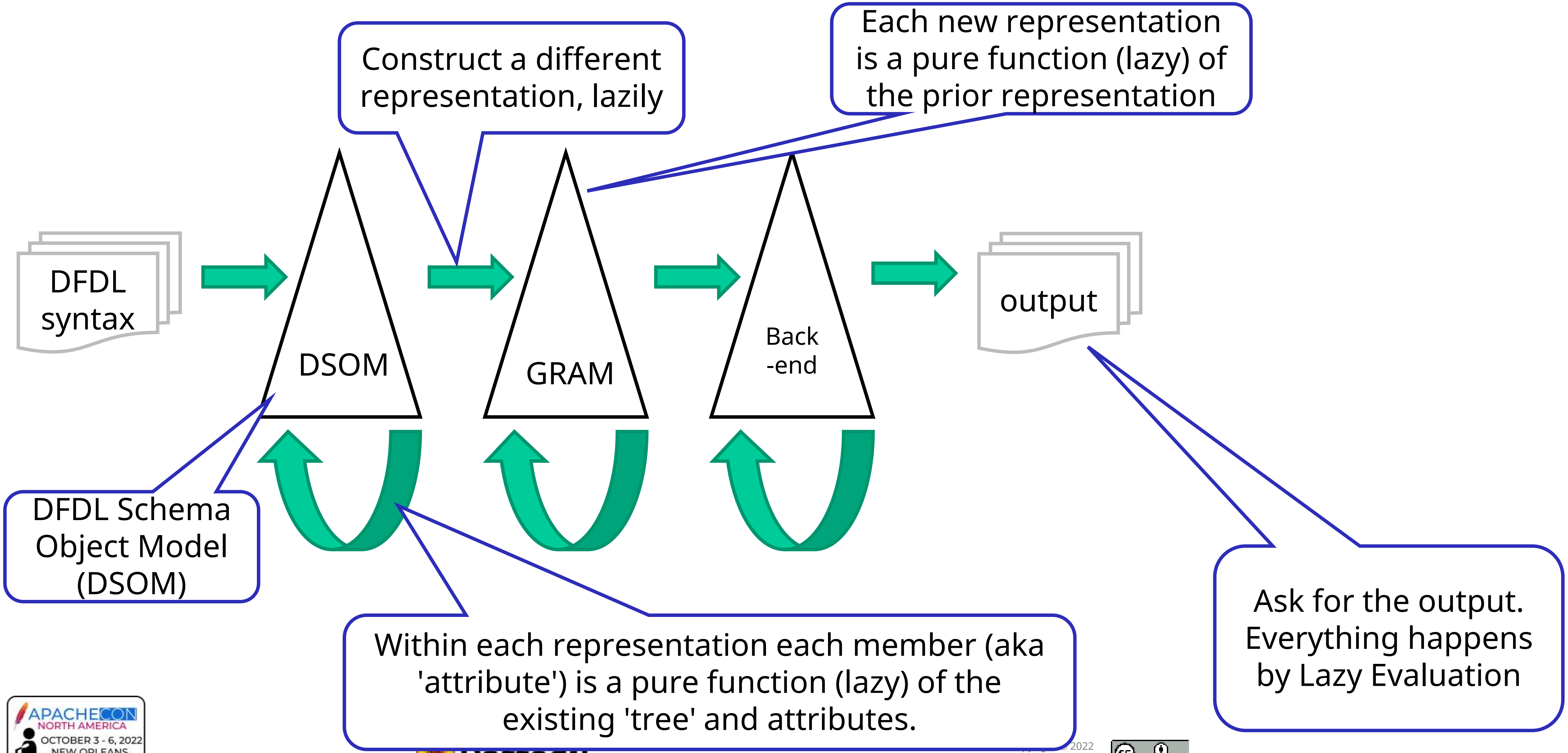
- Command Line Interface
  - Interactive CLI debugger and trace
  - XML, JSON, and EXI (new!) for parse-output, unparse-input

- New: Can also get the Daffodil VSCode Extension

Any Compiler



Daffodil Schema Compiler



Construct a different representation, lazily

Each new representation is a pure function (lazy) of the prior representation

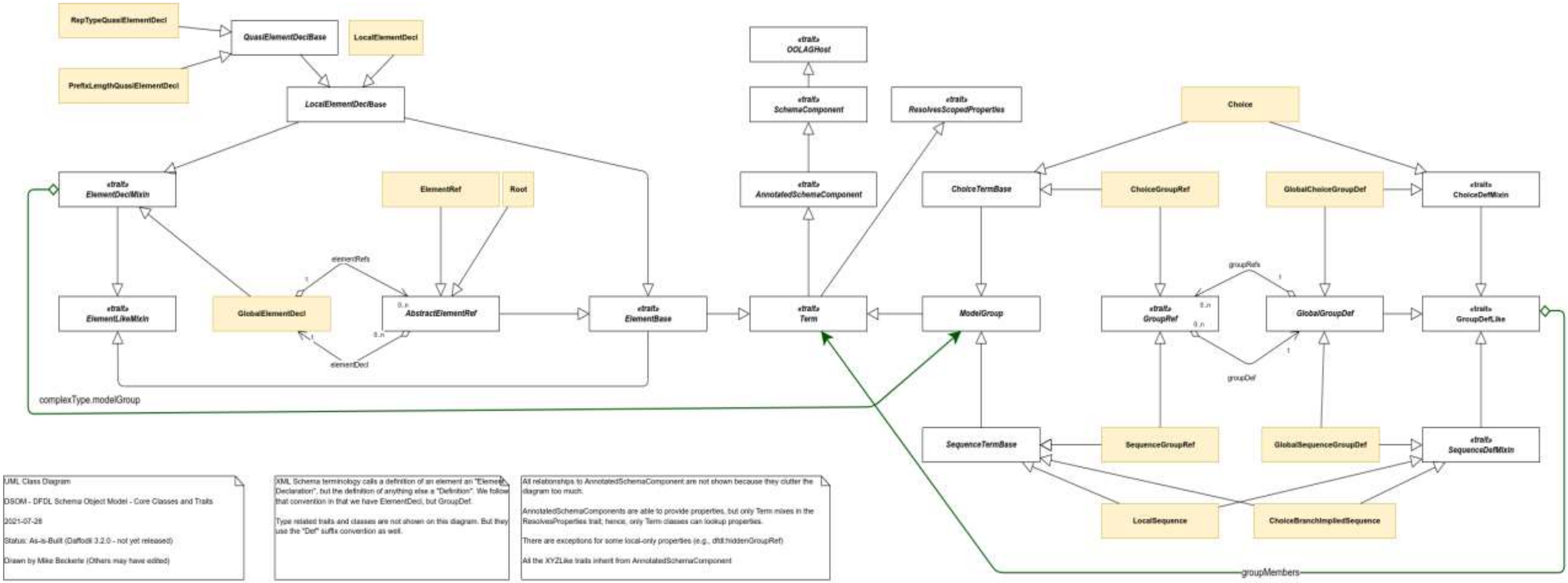
DFDL Schema Object Model (DSOM)

Within each representation each member (aka 'attribute') is a pure function (lazy) of the existing 'tree' and attributes.

Ask for the output. Everything happens by Lazy Evaluation

DSOM

- Similar to XML Schema Object Model (XSOM)
- Scala, lazy functional programming



UML Class Diagram

DSOM - DFDL Schema Object Model - Core Classes and Traits

2021-07-28

Status: As-is-Built (Daffodil 3.2.0 - not yet released)

Drawn by Mike Beckerle (Others may have edited)

XML Schema terminology calls a definition of an element an "Element Declaration", but the definition of anything else a "Definition". We follow that convention in that we have ElementDecl, but GroupDef.

Type related traits and classes are not shown on this diagram. But they use the "Def" suffix convention as well.

All relationships to AnnotatedSchemaComponent are not shown because they clutter the diagram too much.

AnnotatedSchemaComponents are able to provide properties, but only Term mixes in the ResolvesProperties trait; hence, only Term classes can lookup properties.

There are exceptions for some local-only properties (e.g., dfdl:hiddenGroupRef)

All the XYZLike traits inherit from AnnotatedSchemaComponent

OOLAG - Object Oriented Lazy Attribute Grammars

- Functional Programming Idiom for Compilers
  - Johnsson, Thomas. (1995). *Attribute Grammars as a Functional Programming Paradigm*. LNCS. 274. 10.1007/3-540-18317-5_10.
- Attribute grammars - a grammar with 'attribute' computations
  - Wikipedia "Attribute Grammar"
  - synthetic (bottom up)
  - inherited (top down) (Not the OO notion of inherit)
- Object-Orientation
  - Mixins (via Scala traits), Inheritance
- Powerful pattern for Rich Transformations (like compilers)

OOLAG - Object Oriented Lazy Attribute Grammars

- Lazy Evaluation - Avoids organizing compiler into 'passes'
- All values are special OOLAGValue that allow the answer of a computation to be
    - an ordinary value
    - a set of diagnostic objects, one or more of which are errors
    - both (a value and diagnostics that are only warnings)
- Code is structured into OOLAGValue calculations (using the LV idiom) and OOLAGHost objects
- OOLAGHost objects carry a list of required evaluations that must be evaluated to insure they are 'done' and can answer the isError test.

Lazy Evaluation in Scala

on class Choice (extends Term)

```
lazy val hasKnownRequiredSyntax: Boolean = LV {
 hasFraming || groupMembers.forall { _.hasKnownRequiredSyntax }
}.value
```

on class Term (a supertype of Choice)

```
def hasKnownRequiredSyntax: Boolean

lazy val hasFraming = LV {
  hasInitiator || hasTerminator || !hasNoSkipRegions
}.value

lazy val hasNoSkipRegions = LV {
 leadingSkip == 0 && trailingSkip == 0
}.value
```

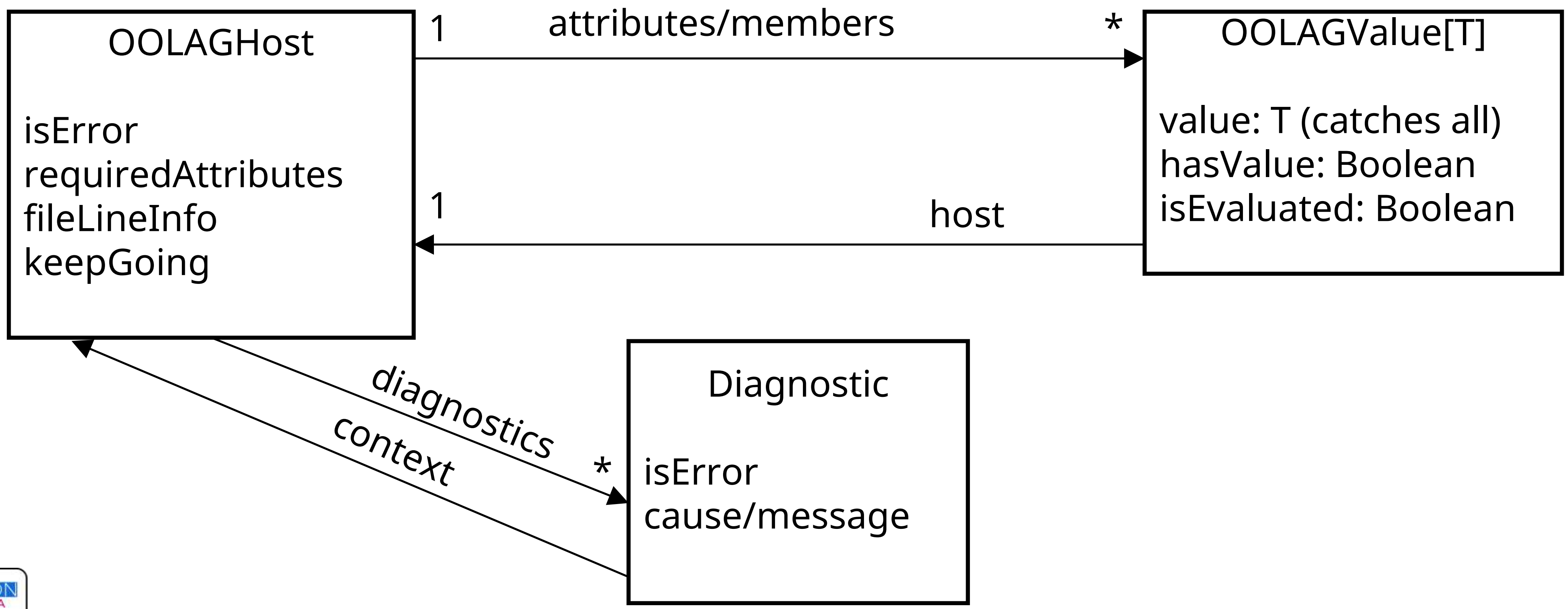
Lazy Evaluation in Scala

on class Sequence (extends Term)

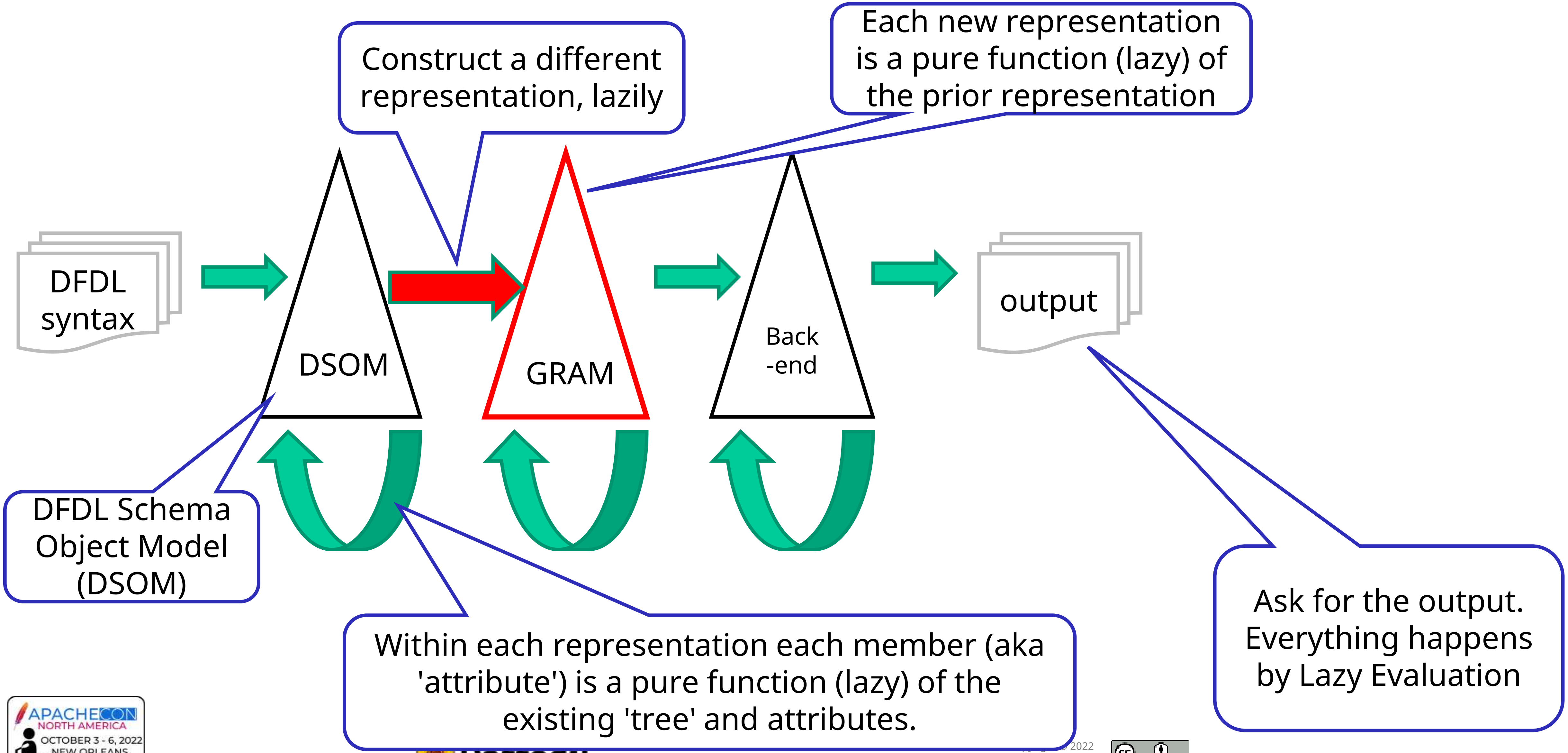
```
lazy val hasKnownRequiredSyntax: Boolean = LV {
 hasFraming ||
 groupMembers.exists { m =>
   m.isRequired &&
   m.hasKnownRequiredSyntax
 }
}.value
```

OOLAG Host and OOLAG Value

- Error accumulation
  - Gathering more than one error before giving up
  - Avoiding duplicates
  - Associating file and line number with the right object to 'blame' from the DFDL schema
- Circular Definition Detection



Daffodil Schema Compiler



DFDL syntax

DSOM

GRAM

Back -end

output

Construct a different representation, lazily

Each new representation is a pure function (lazy) of the prior representation

DFDL Schema Object Model (DSOM)

Within each representation each member (aka 'attribute') is a pure function (lazy) of the existing 'tree' and attributes.

Ask for the output. Everything happens by Lazy Evaluation

Gram - Grammar Trees

- Data Grammar
  - Based on concepts *Scala Combinator Parsers*
    - Yet Another Functional Programming Pattern
- Optimizations done on Grammar Trees
  - Back-end independent

Grammar Rules in Scala

OWL Cyber Defense

on trait ModelGroupGrammarMixin

```
lazy val termContentBody = prod {
  startGroupStmts ~ groupLeftFraming ~ groupContentWithDelims ~
  groupRightFraming ~ endGroupStmts
}

lazy val groupLeftFraming = prod {
  LeadingSkipRegion() ~ AlignmentFill()
}

lazy val groupContentWithDelims = prod {
  initiatorRegion ~ groupContent ~ terminatorRegion
}
```

on class InitiatedTerminatedMixin

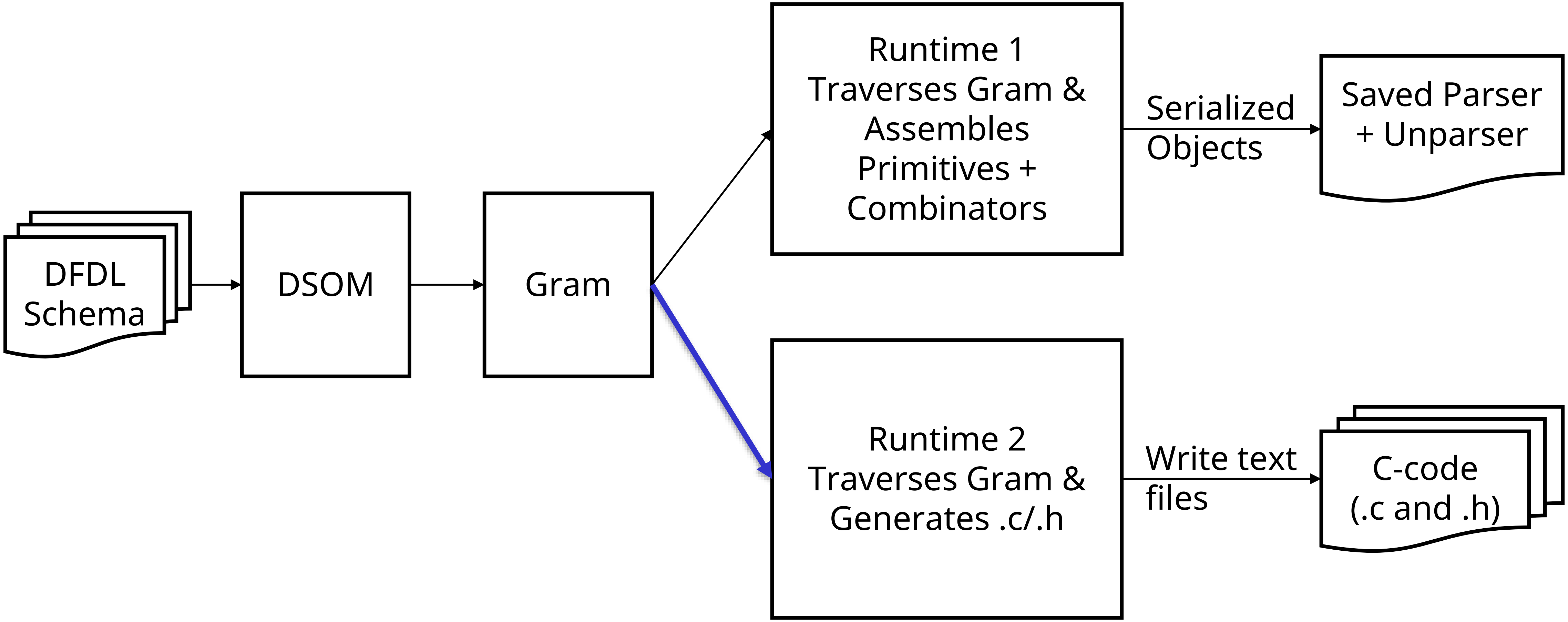
```
lazy val terminatorRegion =
  prod(hasTerminator) { delimMTA ~ Terminator() }
```

C-code Runtime

aka "Runtime 2"

Mostly a contribution of John Interrante of GE Research

Daffodil Schema Compilation


DFDL Schema
DSOM
Gram
Runtime 1 Traverses Gram & Assembles Primitives + Combinators
Serialized Objects
Saved Parser + Unparser
Runtime 2 Traverses Gram & Generates .c/.h
Write text files
C-code (.c and .h)

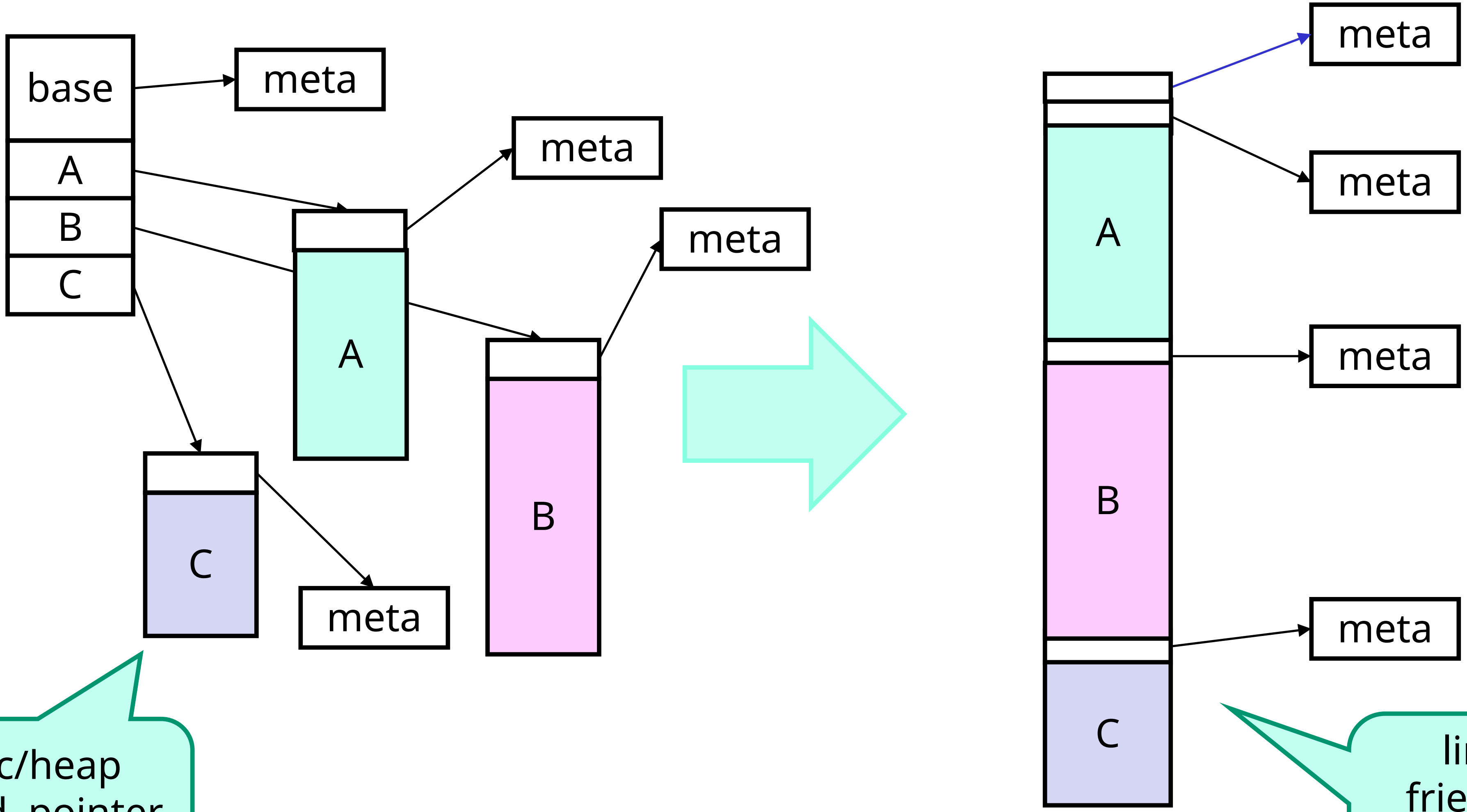
Runtime 2: Different Goals

- Accepts Restrictions on DFDL for simplicity and performance
  - Deterministic - no backtracking
  - Must fit in memory - no streaming
  - Focus on binary data, mostly fixed-length fields
- Generates C code
  - Fast, small footprint
  - Statically allocate everything possible
- Future: generate VHDL for FPGA hardware realization

Runtime 2 Infoset

- More efficient than what programmers would typically create
- Generality needed to handle all of DFDL infoset
- Walkable: metadata connected
- Cache/Prefetch friendly - localized/linearized (not pointers)
    - Similar concepts to Java 'Valhalla' JVM design goals

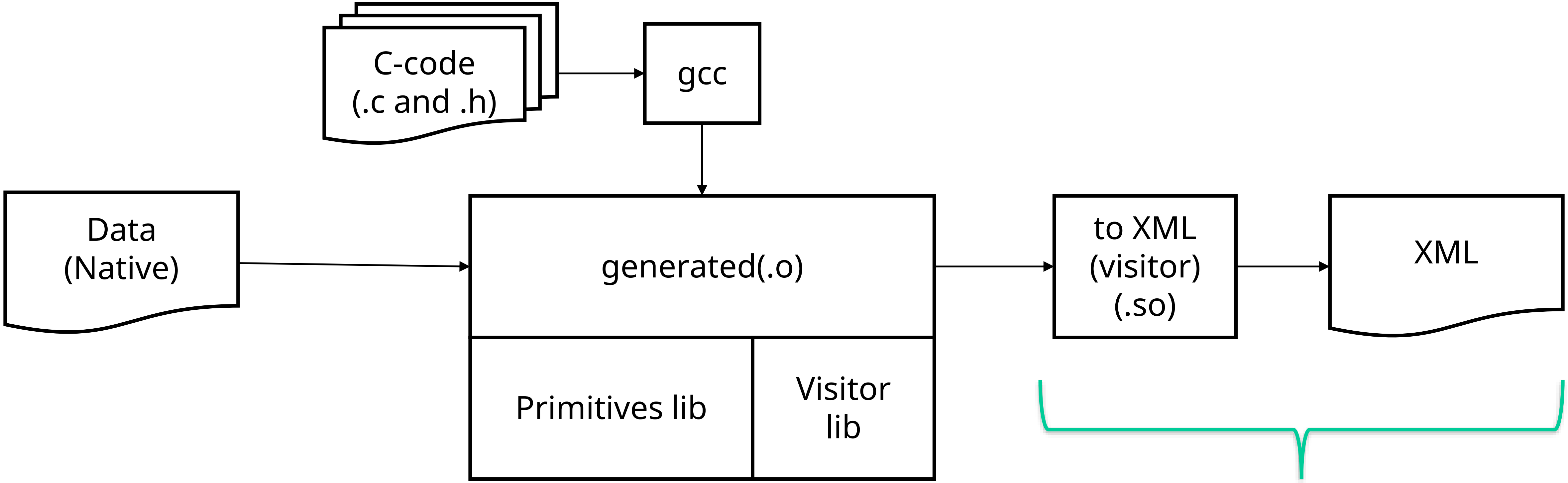
Runtime 2 Infoset - Localized



malloc/heap oriented, pointer intensive

linear, cache-friendly, prefetch friendly, hardware friendly

Runtime for Daffodil Runtime 2



Conveniently plugs into all our test infrastructure

Makes the infoset tangible

C Back-end (aka Runtime 2)

- Two simple tuples

```
<complexType name="FooType">
  <sequence>
    <element name="a" type="xs:int"/>
    <element name="b" type="xs:int"/>
    <element name="c" type="xs:int"/>
  </sequence>
</complexType>

<complexType name="BarType">
  <sequence>
    <element name="x" type="xs:double"/>
    <element name="y" type="xs:double"/>
    <element name="z" type="xs:double"/>
  </sequence>
</complexType>
```

C Back-end (aka Runtime 2)

- Tagged union of two tuples

```
<complexType name="NestedUnionType">
  <sequence>
    <element name="tag" type="xs:int32"/>
    <element name="data">
      <complexType>
        <choice dfdl:choiceDispatchKey="{xs:string(../tag)}">
          <element name="foo" type="idl:FooType"
             dfdl:choiceBranchKey="1 2"/>
          <element name="bar" type="idl:BarType"
             dfdl:choiceBranchKey="3 4"/>
        </choice>
      </complexType>
    </element>
  </sequence>
</complexType>
```

Generated C Highlights

- the tuples

```c
typedef struct foo_data_NestedUnionType_
{
    InfosetBase _base;
    int32_t     a;
    int32_t     b;
    int32_t     c;
} foo_data_NestedUnionType_;

typedef struct bar_data_NestedUnionType_
{
    InfosetBase _base;
    double      x;
    double      y;
    double      z;
} bar_data_NestedUnionType_;
```

Generated C Highlights

```c
typedef struct data_NestedUnionType_
{
    InfosetBase _base;
    size_t      _choice; // choice of which union field to use
    union
    {
        foo_data_NestedUnionType_ foo;
        bar_data_NestedUnionType_ bar;
    };
} data_NestedUnionType_;

typedef struct NestedUnion_
{
    InfosetBase _base;
    int32_t     tag;
    data_NestedUnionType_ data;
} NestedUnion_;
```

Generated C Highlights

```
static void
data_NestedUnionType__unparseSelf(const data_NestedUnionType_ *instance, UState *ustate)
{
    static Error error = {ERR_CHOICE_KEY, {0}};

    ustate->error = instance->_base.erd->initChoice(&instance->_base, rootElement());
    if (ustate->error) return;

    switch (instance->_choice)
    {
    case 0:
        foo_data_NestedUnionType__unparseSelf(&instance->foo, ustate);
        if (ustate->error) return;
        break;
    case 1:
        bar_data_NestedUnionType__unparseSelf(&instance->bar, ustate);
        if (ustate->error) return;
        break;
    default:
        // Should never happen because initChoice would return an error first
        ...
        return;
    }
}
```

```
static void
foo_data_NestedUnionType__unparseSelf(
  const foo_data_NestedUnionType_ *instance,
  UState *ustate)
{
  unparse_be_int32(instance->a, 32, ustate); if (ustate->error) return;
  unparse_be_int32(instance->b, 32, ustate); if (ustate->error) return;
  unparse_be_int32(instance->c, 32, ustate); if (ustate->error) return;
}
```

C-code Generator Status

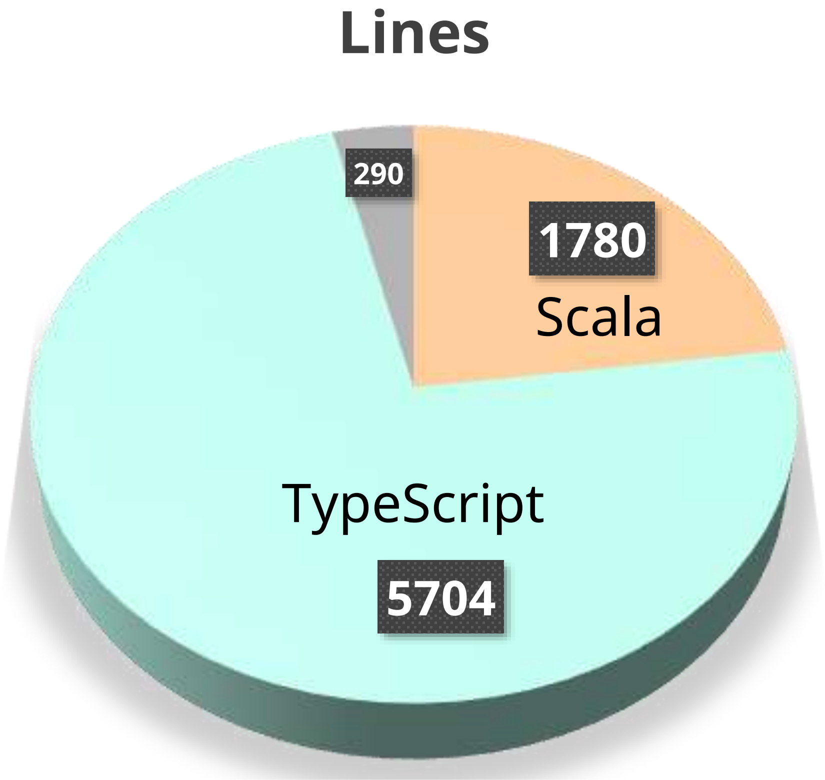
- Still Partial
  - Needs strings, variable-length arrays, expressions

More cool stuff…

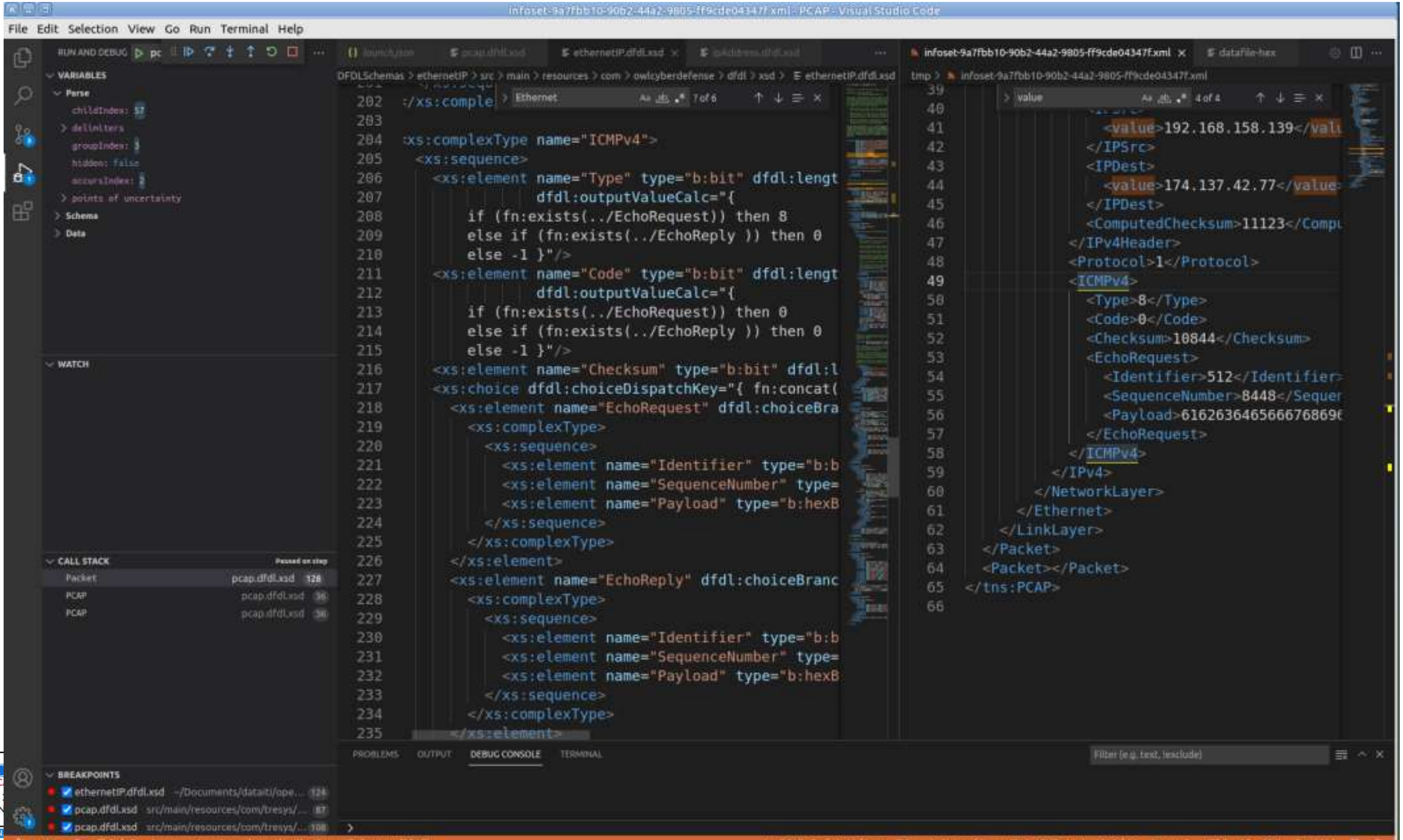
Apache Daffodil VSCode Debugger

OWL Cyber Defense

- Data Format Debugger
  - Eventually a full Data-Format-Oriented IDE
- Extension to VSCode
  - Front-end - Typescript
    - Strongly typed Javascript
  - Back-end server - Scala
    - Uses the Daffodil library (Scala backend)
    - More functional programming idioms: typelevel FS2 & Cats Effect



Apache Daffodil VSCode Extension



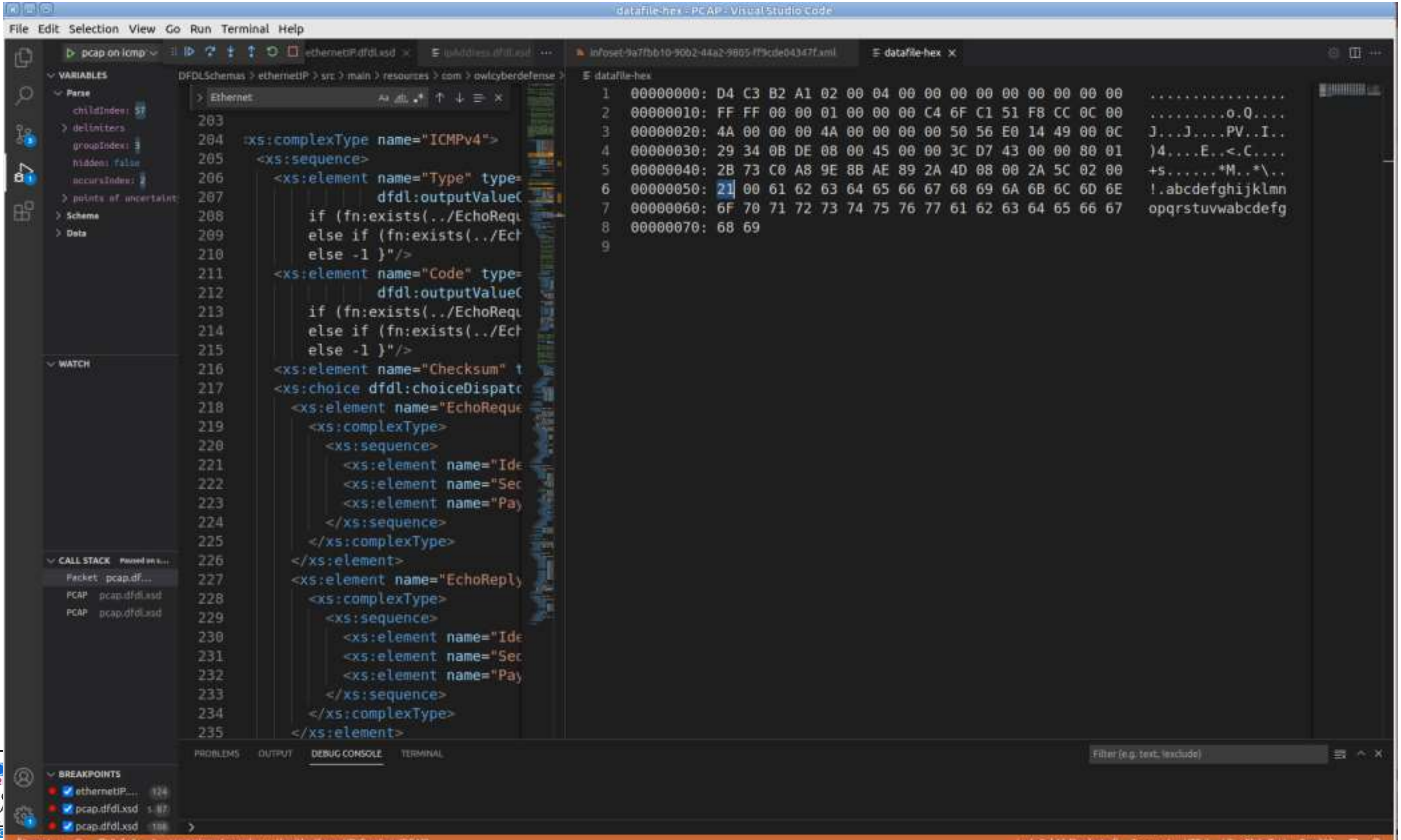
infoset-9a7fbb10-90b2-44a2-9805-ff9cde04347f.xml - PCAP - Visual Studio Code
File Edit Selection View Go Run Terminal Help
RUN AND DEBUG
ethernetIP.dfdl.xsd
infoset-9a7fbb10-90b2-44a2-9805-ff9cde04347f.xml
datafile-hex
DFDLSchemas > ethernetIP > src > main > resources > com > owlcyberdefense > dfdl > xsd > ethernetIP.dfdl.xsd
tmp > infoset-9a7fbb10-90b2-44a2-9805-ff9cde04347f.xml
Ethernet
7 of 6
value
4 of 4
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Filter (e.g. text, !exclude)

```
VARIABLES
  Parse
    childIndex: 57
  > delimiters
    groupIndex: 3
    hidden: false
    occursIndex: 2
  > points of uncertainty
> Schema
> Data

WATCH

CALL STACK                              Paused on step
  Packet                            pcap.dfdl.xsd  128
  PCAP                              pcap.dfdl.xsd   36
  PCAP                              pcap.dfdl.xsd   36

BREAKPOINTS
  ☑ ethernetIP.dfdl.xsd  ~/Documents/dataiti/ope...  124
  ☑ pcap.dfdl.xsd  src/main/resources/com/tresys/...  87
  ☑ pcap.dfdl.xsd  src/main/resources/com/tresys/...  108
```

```xml
:/xs:comple

:xs:complexType name="ICMPv4">
  <xs:sequence>
    <xs:element name="Type" type="b:bit" dfdl:lengt
                dfdl:outputValueCalc="{
      if (fn:exists(../EchoRequest)) then 8
      else if (fn:exists(../EchoReply )) then 0
      else -1 }"/>
    <xs:element name="Code" type="b:bit" dfdl:lengt
                dfdl:outputValueCalc="{
      if (fn:exists(../EchoRequest)) then 0
      else if (fn:exists(../EchoReply )) then 0
      else -1 }"/>
    <xs:element name="Checksum" type="b:bit" dfdl:l
    <xs:choice dfdl:choiceDispatchKey="{ fn:concat(
      <xs:element name="EchoRequest" dfdl:choiceBra
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Identifier" type="b:b
            <xs:element name="SequenceNumber" type=
            <xs:element name="Payload" type="b:hexB
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="EchoReply" dfdl:choiceBranc
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Identifier" type="b:b
            <xs:element name="SequenceNumber" type=
            <xs:element name="Payload" type="b:hexB
          </xs:sequence>
        </xs:complexType>
      </xs:element>
```

```xml
                <value>192.168.158.139</valu
              </IPSrc>
              <IPDest>
                <value>174.137.42.77</value>
              </IPDest>
              <ComputedChecksum>11123</Compu
            </IPv4Header>
            <Protocol>1</Protocol>
            <ICMPv4>
              <Type>8</Type>
              <Code>0</Code>
              <Checksum>10844</Checksum>
              <EchoRequest>
                <Identifier>512</Identifier>
                <SequenceNumber>8448</Sequer
                <Payload>616263646566676869(
              </EchoRequest>
            </ICMPv4>
          </IPv4>
        </NetworkLayer>
      </Ethernet>
    </LinkLayer>
  </Packet>
  <Packet></Packet>
</tns:PCAP>
```

Apache Daffodil VSCode Extension



datafile-hex - PCAP - Visual Studio Code

File Edit Selection View Go Run Terminal Help

pcap on icmp

ethernetIP.dfdl.xsd · ipAddress.dfdl.xsd · infoset-9a7fbb10-90b2-44a2-9805-ff9cde04347f.xml · datafile-hex

VARIABLES

- Parse
  - childIndex: 57
  - delimiters
  - groupIndex: 3
  - hidden: false
  - occursIndex: 2
  - points of uncertaint
- Schema
- Data

WATCH

CALL STACK Paused on s...

Packet	pcap.df...
PCAP	pcap.dfdl.xsd
PCAP	pcap.dfdl.xsd

BREAKPOINTS

☑	ethernetIP....	124
☑	pcap.dfdl.xsd	5 87
☑	pcap.dfdl.xsd	108

DFDLSchemas > ethernetIP > src > main > resources > com > owlcyberdefense >

Ethernet

```

<xs:complexType name="ICMPv4">
  <xs:sequence>
    <xs:element name="Type" type=
                dfdl:outputValueC
          if (fn:exists(../EchoRequ
          else if (fn:exists(../Ech
          else -1 }"/>
    <xs:element name="Code" type=
                dfdl:outputValueC
          if (fn:exists(../EchoRequ
          else if (fn:exists(../Ech
          else -1 }"/>
    <xs:element name="Checksum" t
    <xs:choice dfdl:choiceDispatc
      <xs:element name="EchoReque
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Ide
            <xs:element name="Sec
            <xs:element name="Pay
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="EchoReply
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Ide
            <xs:element name="Sec
            <xs:element name="Pay
          </xs:sequence>
        </xs:complexType>
      </xs:element>
```

datafile-hex

```
00000000: D4 C3 B2 A1 02 00 04 00 00 00 00 00 00 00 00 00   ................
00000010: FF FF 00 00 01 00 00 00 C4 6F C1 51 F8 CC 0C 00   .........o.Q....
00000020: 4A 00 00 00 4A 00 00 00 00 50 56 E0 14 49 00 0C   J...J....PV..I..
00000030: 29 34 0B DE 08 00 45 00 00 3C D7 43 00 00 80 01   )4....E..<.C....
00000040: 2B 73 C0 A8 9E 8B AE 89 2A 4D 08 00 2A 5C 02 00   +s......*M..*\..
00000050: 21 00 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E   !.abcdefghijklmn
00000060: 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67   opqrstuvwabcdefg
00000070: 68 69
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Filter (e.g. text, !exclude)

EXI - Dense Binary XML Alternative

- EXI = Efficient XML Interchange Format (W3C)
- Coming in Daffodil 3.4.0 (soon)
- Wrings all the redundancy and inefficiency out of XML text

Example: Aircraft messaging data format

- Original Message: 174 bytes
- Daffodil -> XML Text Infoset: 1493 bytes
- Daffodil -> EXI infoset: 160 bytes

Conclusion/Review

- Quick Intro to DFDL and Apache Daffodil

- Daffodil Schema Compiler - Functional Programming & Scala
  - Useful idioms for DFDL compilation
  - Enables Code-generation for fast runtimes

- More Cool Stuff
  - VSCode Data Debugger/IDE
  - EXI dense binary alternative to XML text

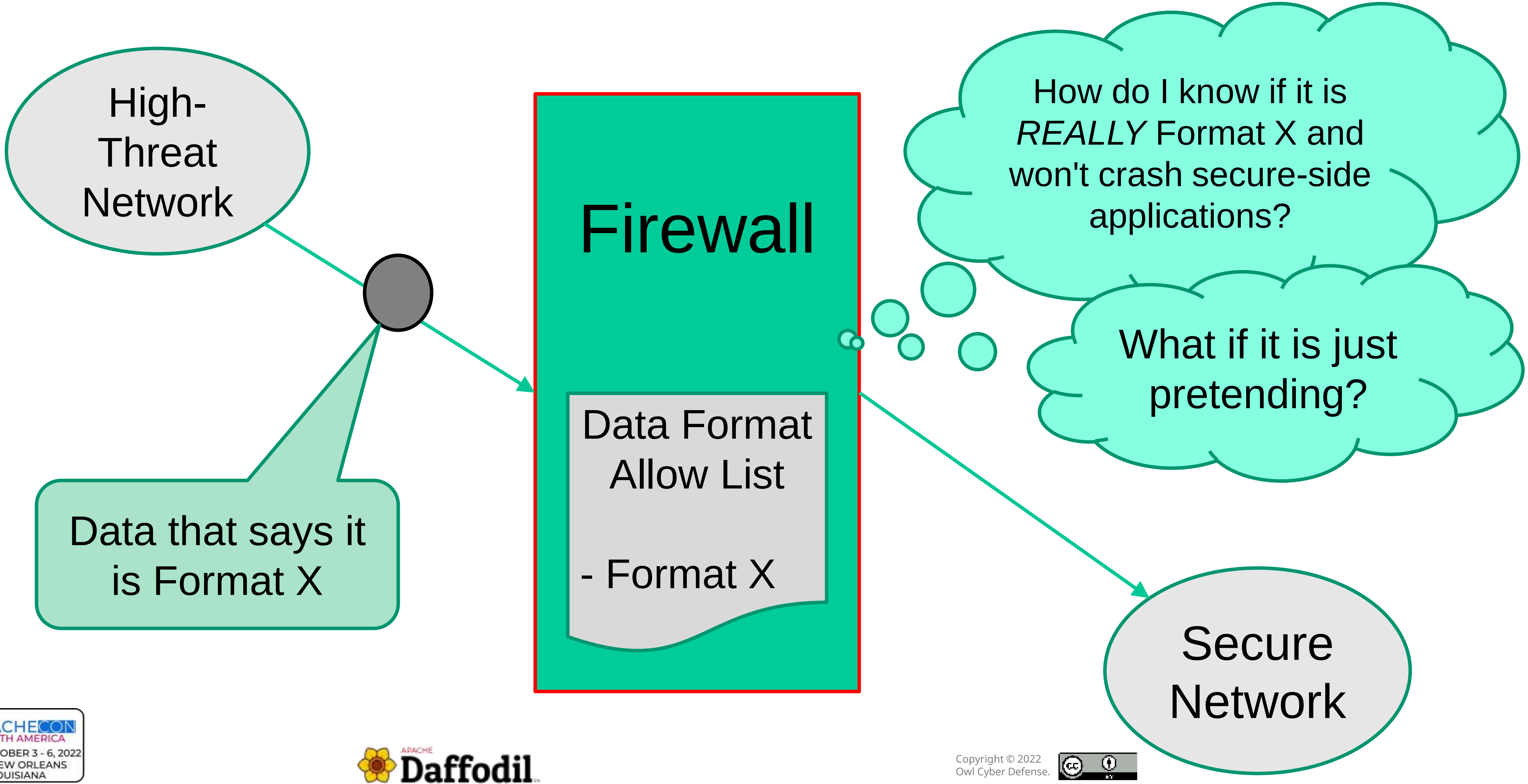
- What am I working on this week?
  - Integration of Daffodil with Apache Drill !

END

Use Case

DFDL AND CYBER SECURITY

Cyber-Security Use Case: Bad Data DoS Attack



High-Threat Network

Data that says it is Format X

Firewall

Data Format Allow List

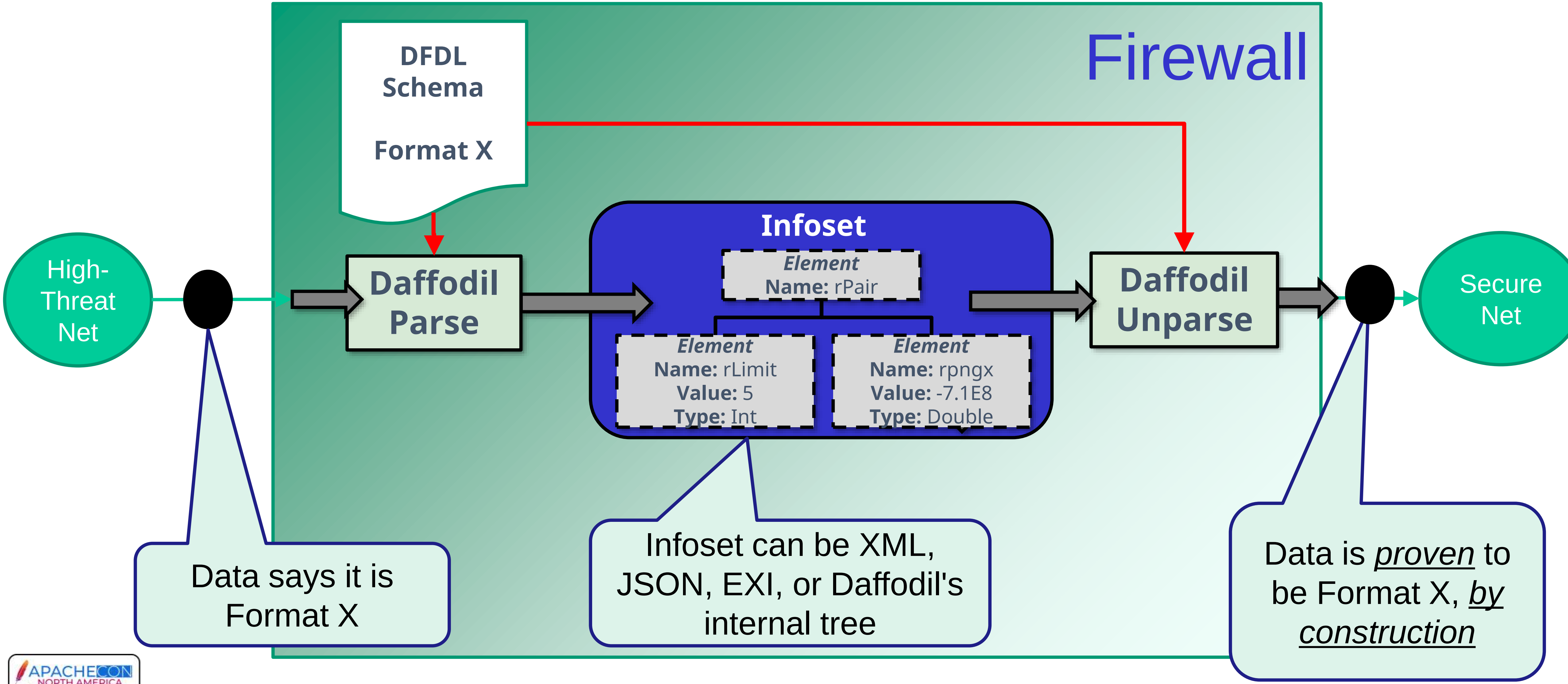
- Format X

How do I know if it is *REALLY* Format X and won't crash secure-side applications?

What if it is just pretending?

Secure Network

Cyber-Security Use Case: Full Protocol Break



Data says it is Format X

Infoset can be XML, JSON, EXI, or Daffodil's internal tree

Data is *proven* to be Format X, *by construction*