END

Extra slides may follow for use in optional discussion.

Current DFDL v1.0 Language Limitations

Recursive types

- DFDL v1.0 ***not*** a ***Turing-Complete*** language
- On purpose - it's a feature, not a bug

Position of elements "by offset"

- Random jumping around data
- Ex: TIFF file format
  - TIFF cannot be described in DFDL v1.0

Thanks to
http://langsec.org/occupy/

APACHE Daffodil™

TRESYS Deep.

Things DFDL (v1.0 + *BLOB*) Does

DFDL is for Images and Video

- Originally not in scope
- Large user demand to use DFDL on the metadata content of image file formats
  - Cybersecurity applications
- Adding BLOB (Binary Large Object) feature to DFDL language to enable DFDL to describe image files

Why is DFDL Needed?

Q But what about...

- Apache Avro
- Apache Thrift
- Google GPBs
- ASN.1 BER (or PER/DER/XER)

A Those are great, but are *prescriptive*.

They don't describe formats, they *are* data formats themselves.

We need a *descriptive* language.

Why is DFDL Needed? - ASN.1 ECN

What about ASN.1 Encoding Control Notation?

- Already an ISO Standard (since 2008)
- Conceptually similar
  - Logical schema language + notations for physical representation
- Very different in the details.
- Developers [Love | Hate] [ASN.1 | XML]

Differences that matter:

- ASN.1 ECN
  - No open source implementation (as of 2018-08-29)
  - Extension of a binary data standard ASN.1 BER/PER/DER
  - Goal to describe legacy protocol messages
- DFDL
  - Open source Daffodil implementation
  - Extension of a textual data standard XML
  - Goal to be union of data integration tool capabilities for format description

Some Technical Coolness

Daffodil Runtime Implementation of Streaming Unparser with Calculated-Value Elements

APACHE Daffodil™

TRESYS Deep.

What is Harder: Parsing or Unparsing?

- Most CompSci people think parsing
  - backtracking - state saving/restoring
  - lookahead (bounded/unbounded)
- Unparsing seems simple in comparison
  - Just a tree-walk of the Infoset outputting each element……

But…

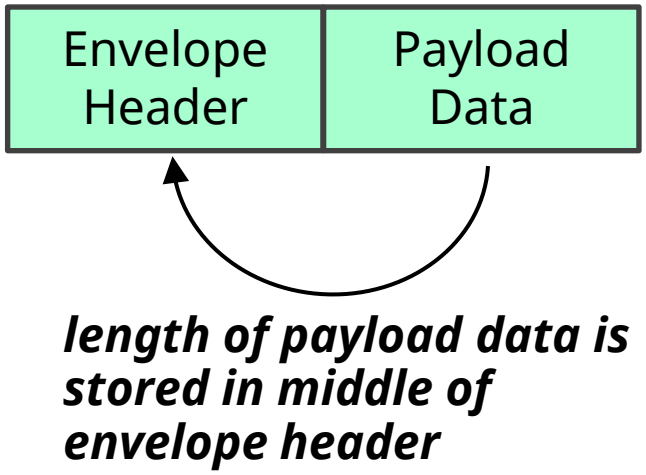
- DFDL has computed elements
  - New innovation over prior-gen format languages
- Let's take a look at what that means for unparsing

Streaming Unparsing

- Limit memory footprint
- Unparser consumes a stream of XML-like events (like SAX - Streaming API for XML)
- The Infoset tree does *NOT* exist
  - We did NOT just parse the data
  - We are starting from the XML events
- Infoset tree is built up as events arrive
  - Can be pruned when tree nodes are no longer needed
  - Infoset stays small even though data stream implies unbounded size

Common: Envelope + Payload

- Data Payload is surrounded by Envelope
- Envelope holds *Stored data length of payload*



Stored Length Limitations on Streaming

```
<Packet>
    <PacketHeader>
      <Seconds>1371631556</Seconds>
      <USeconds>838904</USeconds>
      <InclLen> ??? </InclLen>
      <OrigLen> ??? </OrigLen>
    </PacketHeader>
    <pcap:LinkLayer>
      <pcap:Ethernet>
        <MACDest>005056E01449</MACDest>
        <MACSrc>000C29340BDE</MACSrc>
        <Ethertype>2048</Ethertype>
        ....
      </pcap:Ethernet>
    </pcap:LinkLayer>
 </Packet>
```

Unparse of Envelope needs length of Payload

Payload is a complex object. Must unparse it to determine its length.

Schema with Stored Length

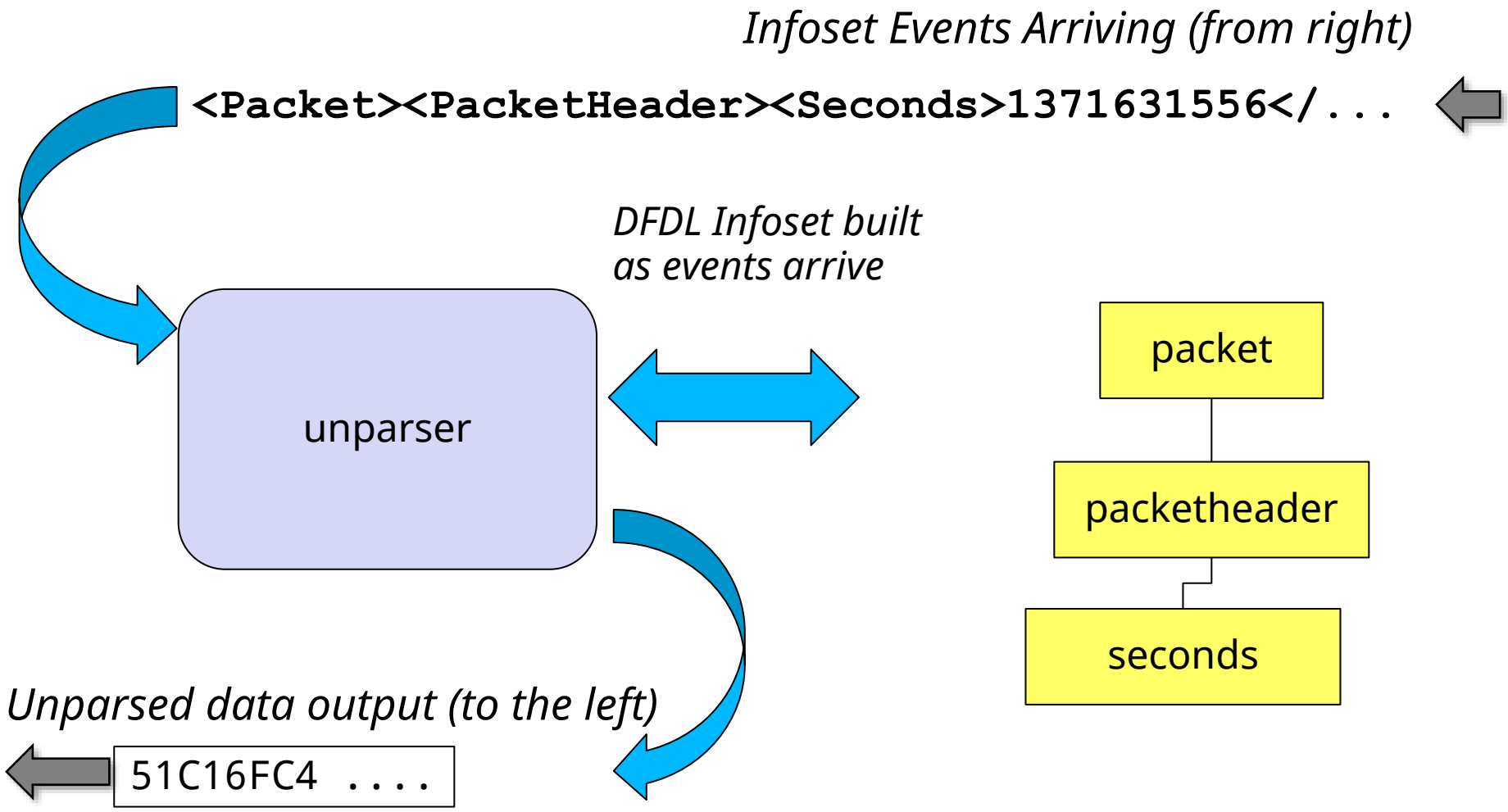
```
<xs:element name="packetHeader">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Seconds" type="pcap:uint32"/>
         <xs:element name="USeconds" type="pcap:uint32"/>
         <xs:element name="InclLen" type="pcap:uint32"
           ...
         />
      </xs:sequence>
   </xs:complexType>
</xs:element>
...
<xs:element ref="pcap:LinkLayer"
   dfdl:lengthUnits="bytes" dfdl:lengthKind="explicit"
   dfdl:length="{ ../packetHeader/InclLen }"/>
```

Schema with Stored Length

```
<xs:element name="packetHeader">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Seconds" type="pcap:uint32"/>
         <xs:element name="USeconds" type="pcap:uint32"/>
         <xs:element name="InclLen" type="pcap:uint32"
          dfdl:outputValueCalc="{
           if (dfdl:valueLength(
                    ../../pcap:LinkLayer/pcap:Ethernet,
                    'bytes') le 60) then 60
           else
              dfdl:valueLength(
                    ../../pcap:LinkLayer/pcap:Ethernet,
                    'bytes') }"
         />
         ....
      </xs:sequence>
   </xs:complexType>
</xs:element>
...
<xs:element ref="pcap:LinkLayer"
   dfdl:lengthUnits="bytes" dfdl:lengthKind="explicit"
   dfdl:length="{ ../header/InclLen }"/>
```

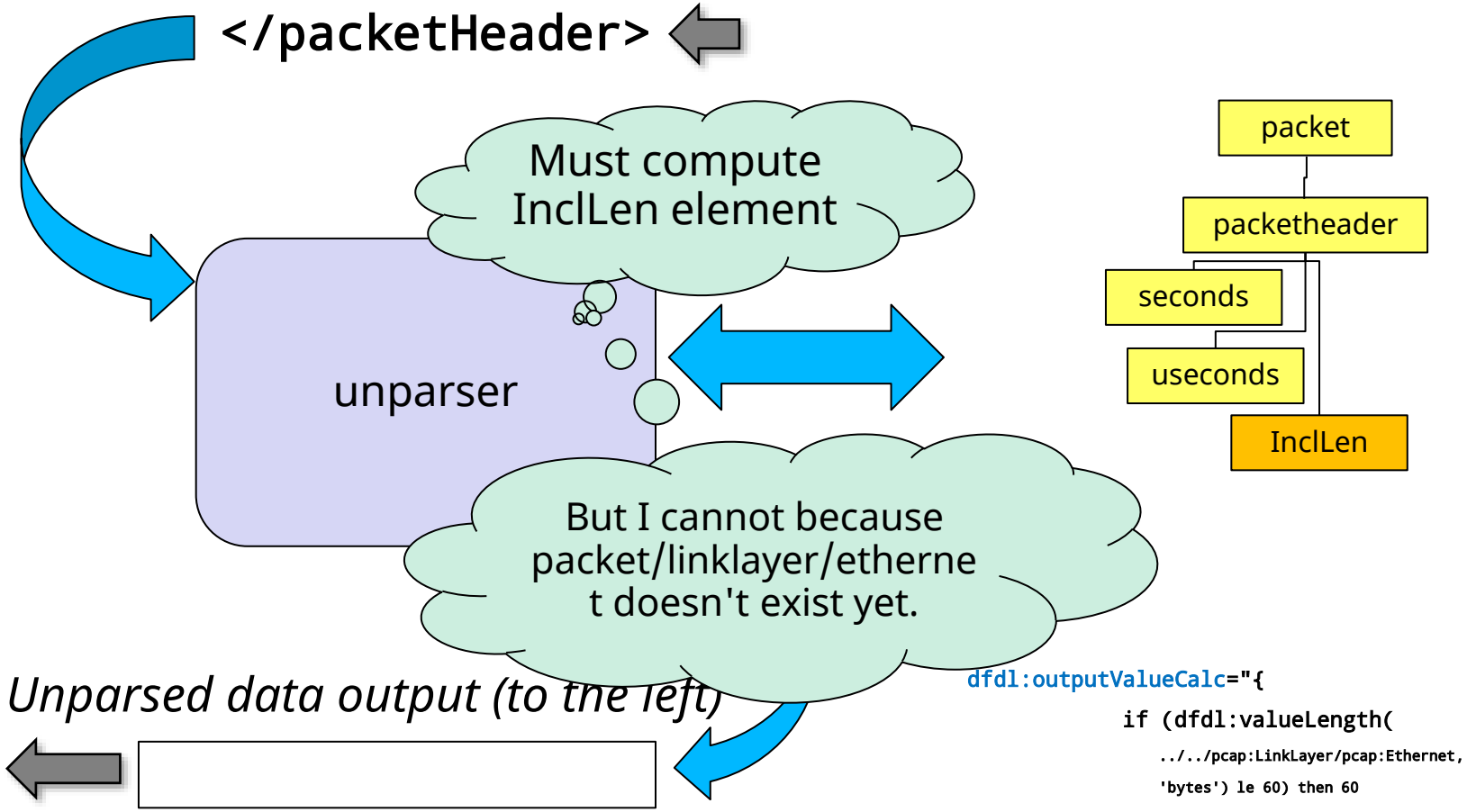
Expressions in DFDL's expression language (like XPath)

Streaming Unparsing Illustration



PCAP Streaming Unparsing Illustration

Infoset Events Arriving (from right)

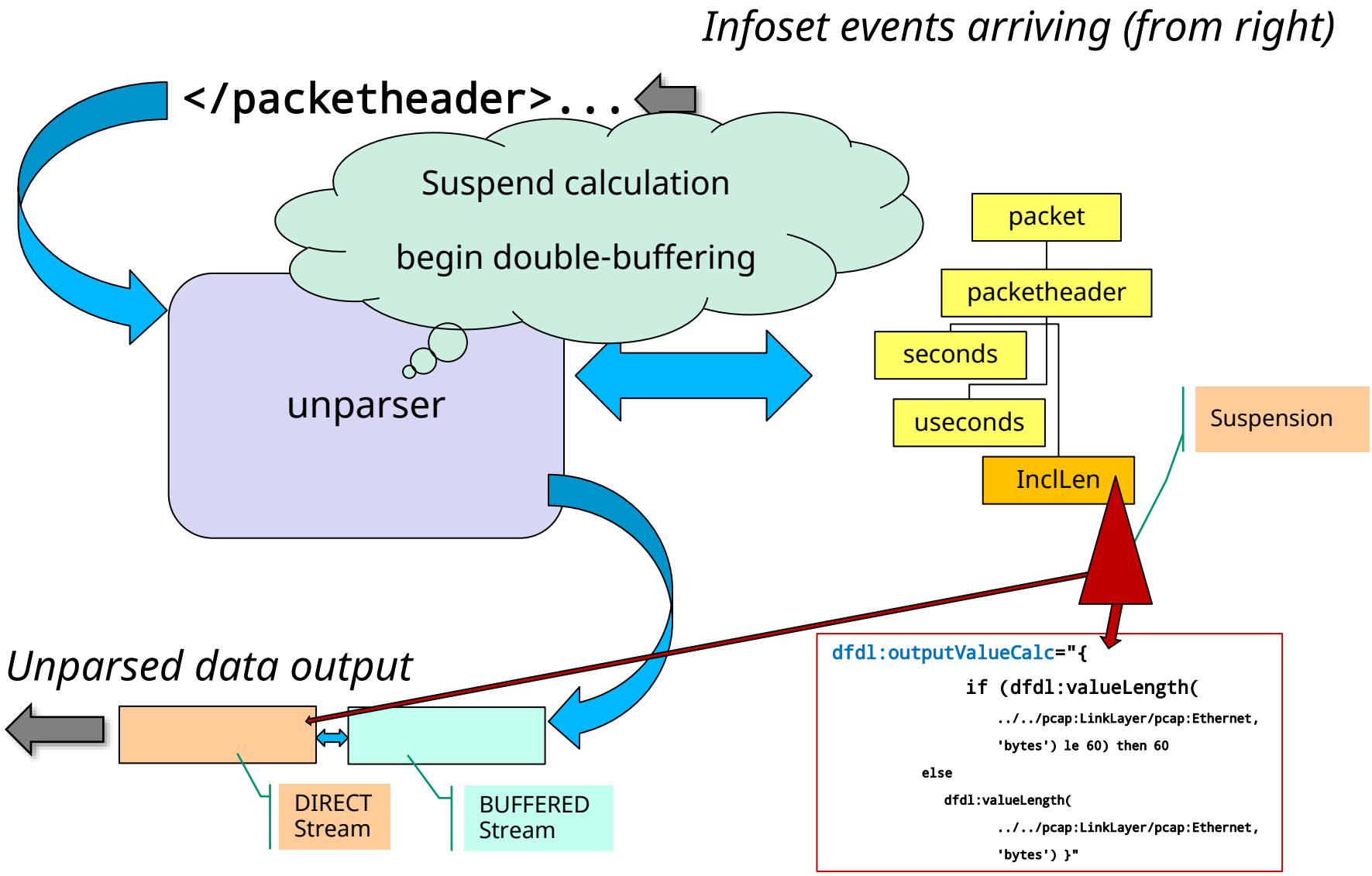



```
dfdl:outputValueCalc="{
        if (dfdl:valueLength(
            ../../pcap:LinkLayer/pcap:Ethernet,
            'bytes') le 60) then 60
    else
       dfdl:valueLength(
            ../../pcap:LinkLayer/pcap:Ethernet,
            'bytes') }"
```

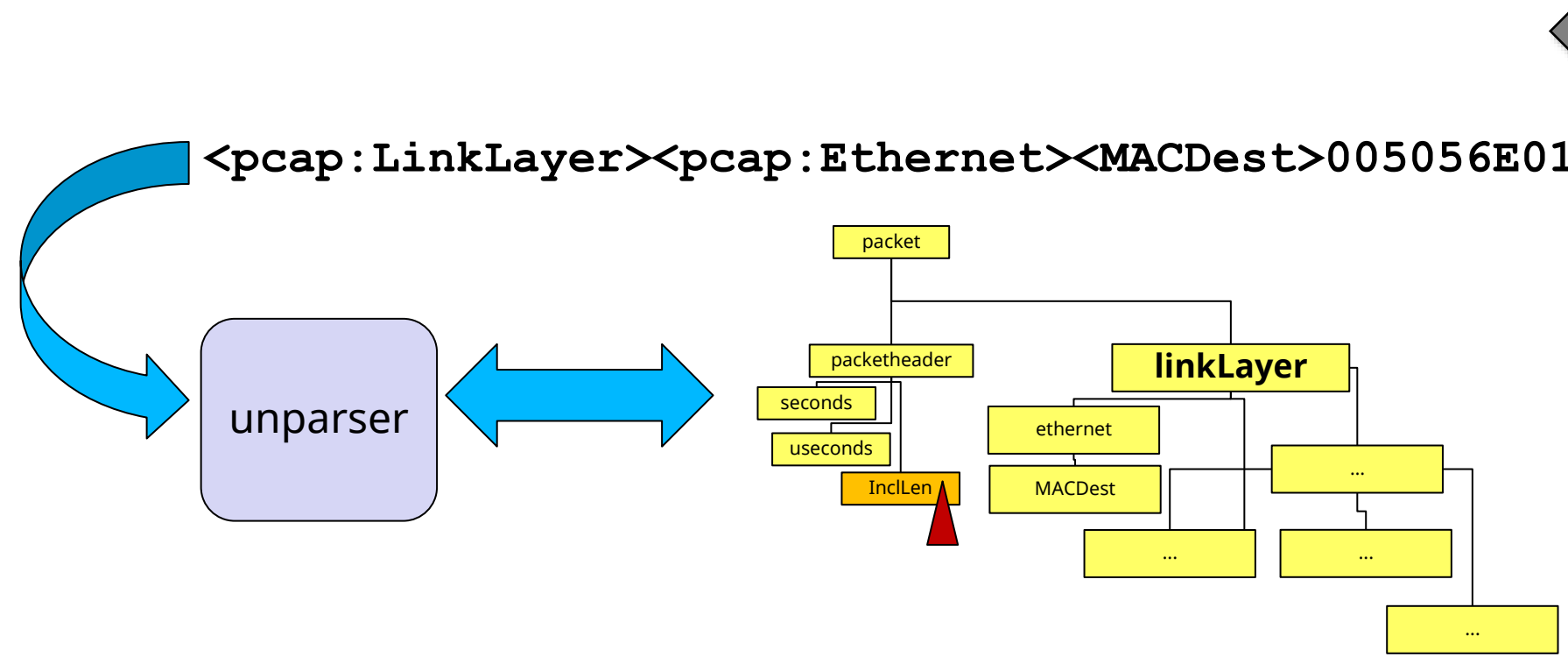
Suspensions and Double Buffering

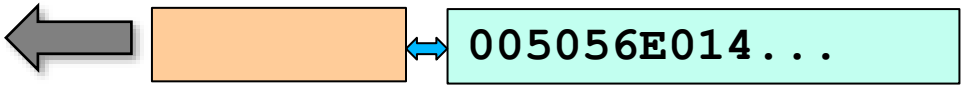
- Must suspend outputValueCalc computation
  - Suspension object - like a co-routine object
- Blocked until packet/linklayer/ethernet
  - Event arrives
  - Added to Infoset tree
  - All its sub-events/elements arrive and are added
  - And… all of it is unparsed to determine the length of its representation

PCAP Streaming Unparsing Illustration

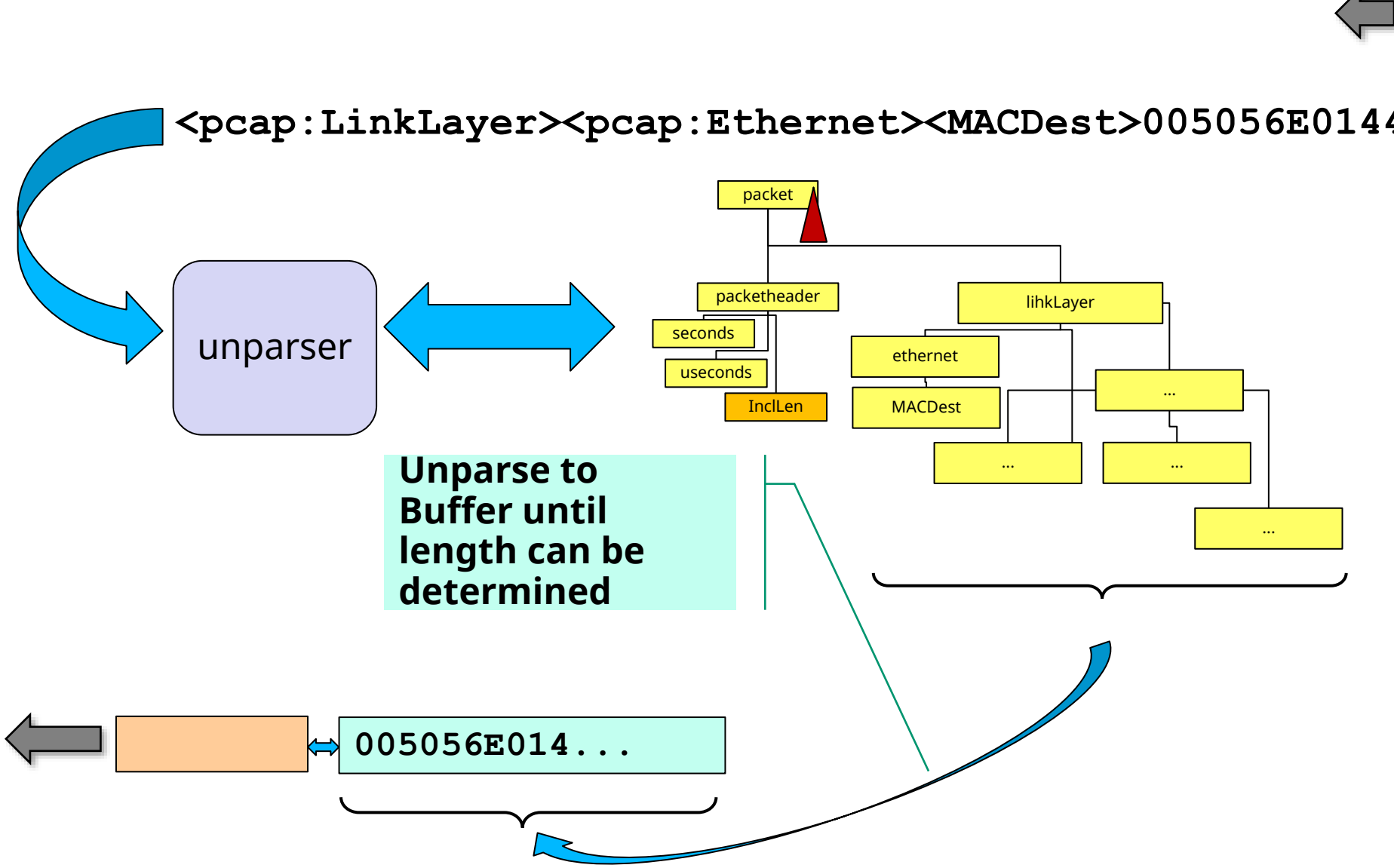


PCAP Streaming Unparsing Illustration

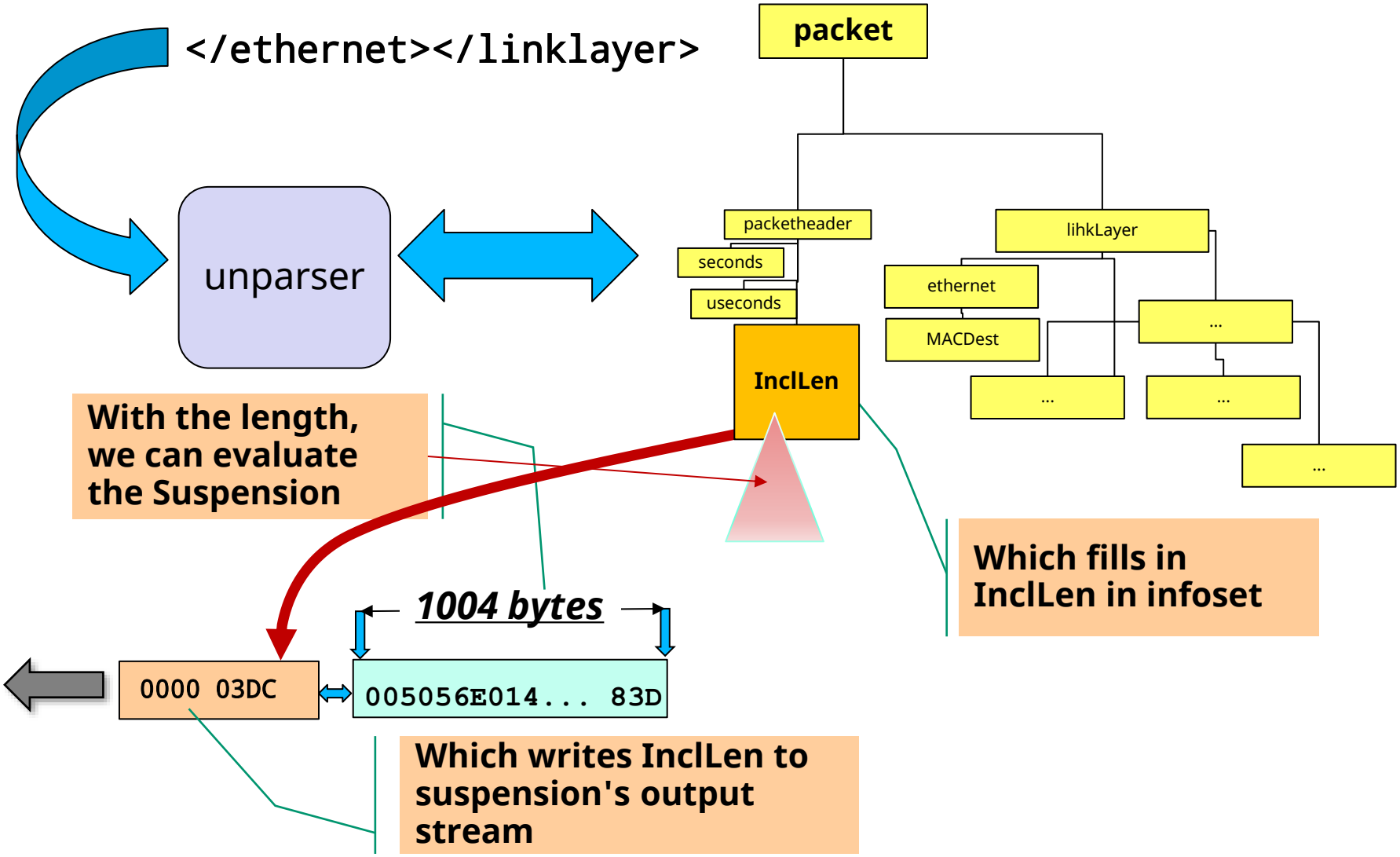




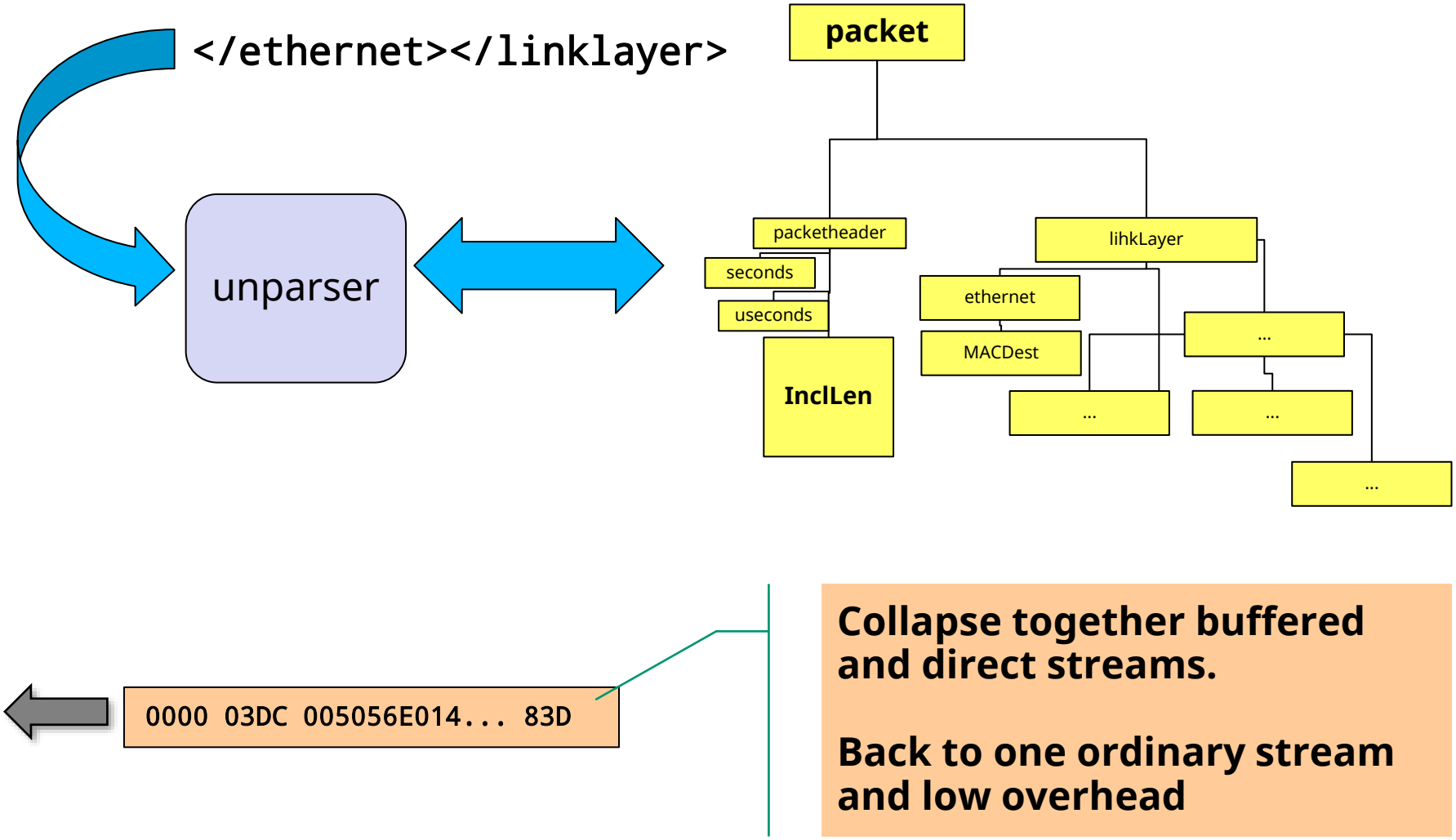
PCAP Streaming Unparsing Illustration



PCAP Streaming Unparsing Illustration



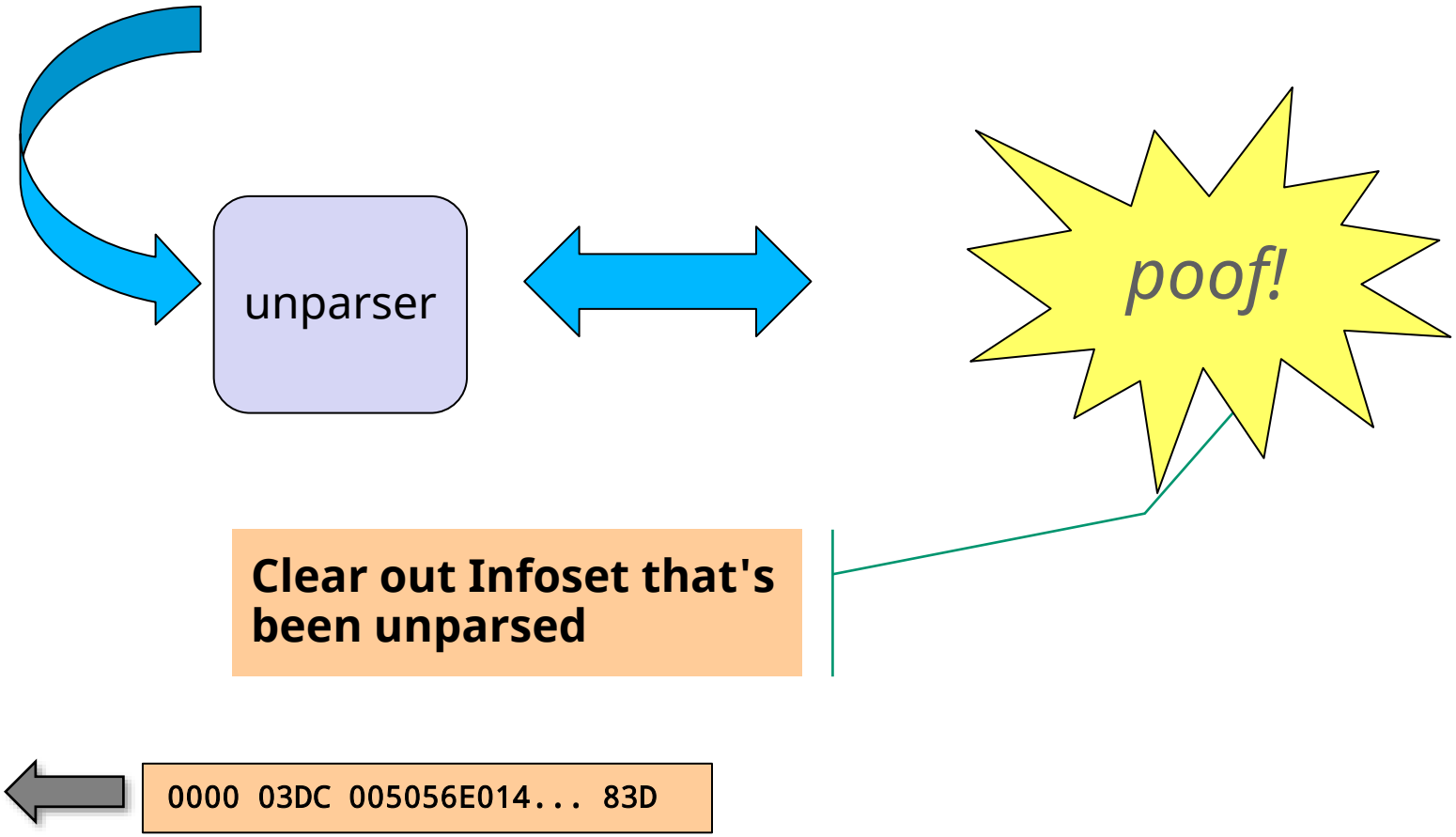
PCAP Streaming Unparsing Illustration



Collapse together buffered and direct streams.

Back to one ordinary stream and low overhead

PCAP Streaming Unparsing Illustration



Streaming Unparsing Illustration

In summary

- Unparsing in DFDL is quite complex
- IMHO: Harder than parsing
- DFDL is much more expressive than other format description languages
  - dfdl:outputValueCalc - computed value elements
- Applications can truly be unaware of stored-lengths - leave that up to Daffodil!