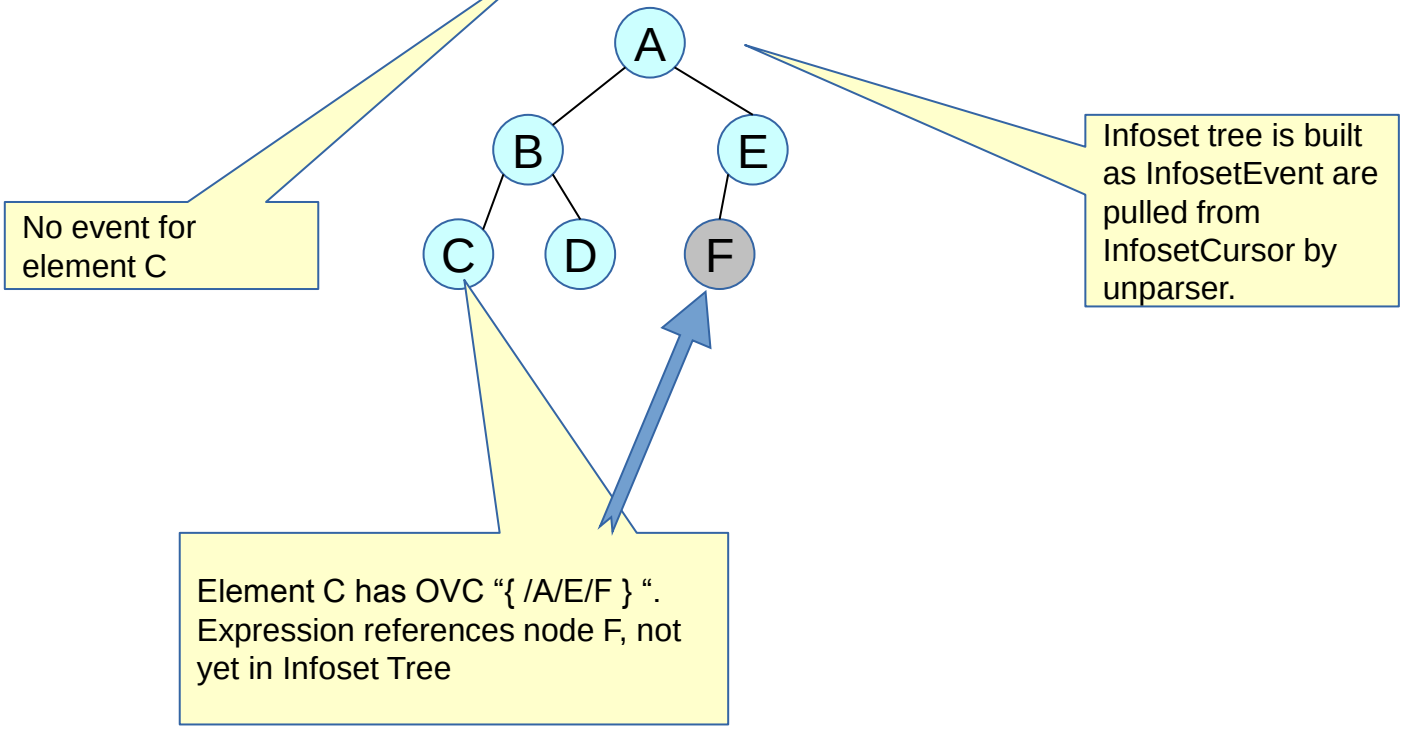
Daffodil Unparser dfdl:outputValueCalc Scenario (2016-05-05)

- Ilustrates how infoset events are received, infoset tree is built incrementally
- Output-value-calc element nodes are added to tree
- Producer Co-routine for expression evaluation is queued on nodes waiting for values or children
- Data output streams start direct, split off a buffering part, then are collapsed back.

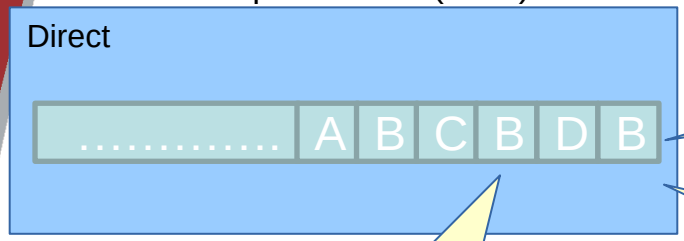
Drawing conventions (so you can understand these slides)

- XML-like infoset events
- Infoset Tree
- Direct and Buffered Data Output Streams (DOS)

XML Events: <a><d></d><e>….



Data Output Stream (DOS)

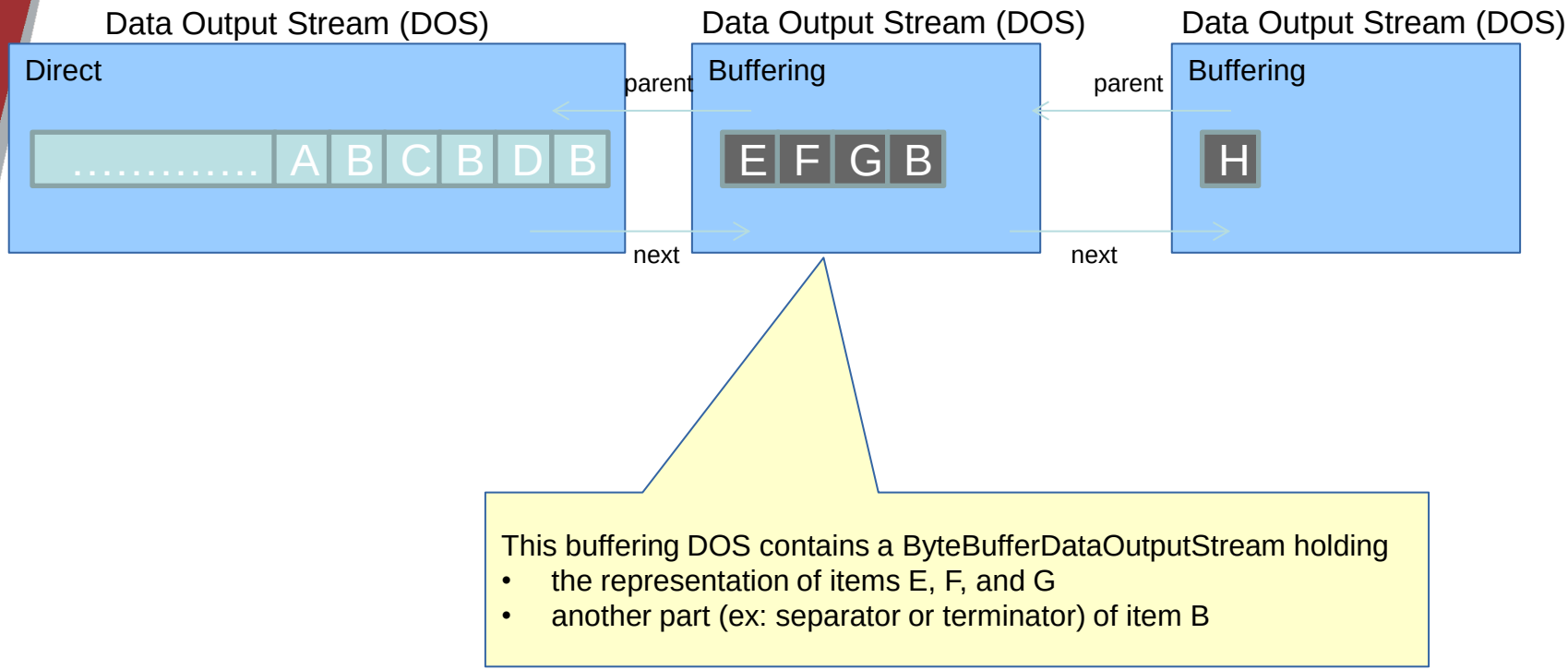


Direct DOS has a java.io.OutputStream. Data is added to this end.

Detail that there is also a fragment byte which is bits not yet making up a whole byte, so they cannot yet be put into the java.io.OutputStream, is not illustrated here.

This stream contains some data (unspecified) followed by

- the representation of item A
- the start of the representation of complex element (or model group) item B
- the representation of item C
- another part (ex: separator) of item B,
- the representation of item D
- another part (ex: separator or terminator) of item B


Data Output Stream (DOS)
Direct
A B C B D B
parent
Data Output Stream (DOS)
Buffering
E F G B
next
parent
Data Output Stream (DOS)
Buffering
H
next
This buffering DOS contains a ByteBufferDataOutputStream holding
• the representation of items E, F, and G
• another part (ex: separator or terminator) of item B

Animated Output-value-calc scenario

XML Events: …….

Scenario starts with unparser running on the main thread, with a direct DataOutputStream. No events have been pulled.

Data Output Stream (DOS)

Direct

XML Events: <a>…

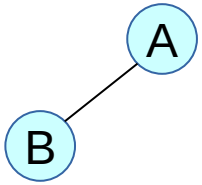
Start event for A, infoset node for A is created, initiator for A is output to the data output stream.

Data Output Stream (DOS)

Direct

A

XML Events: <a><b>…



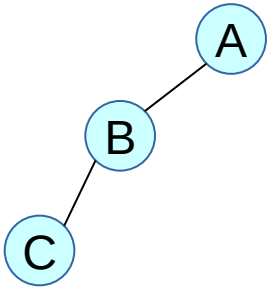
Start event for B, infoset node for B is added to tree, initiator for B is output to the data output stream.

Data Output Stream (DOS)

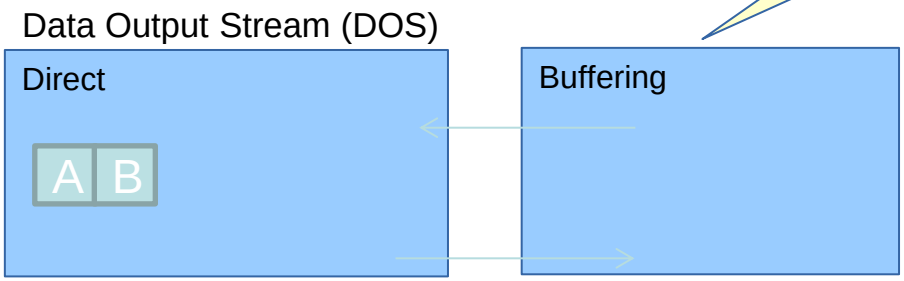
Direct

XML Events: <a><b>…

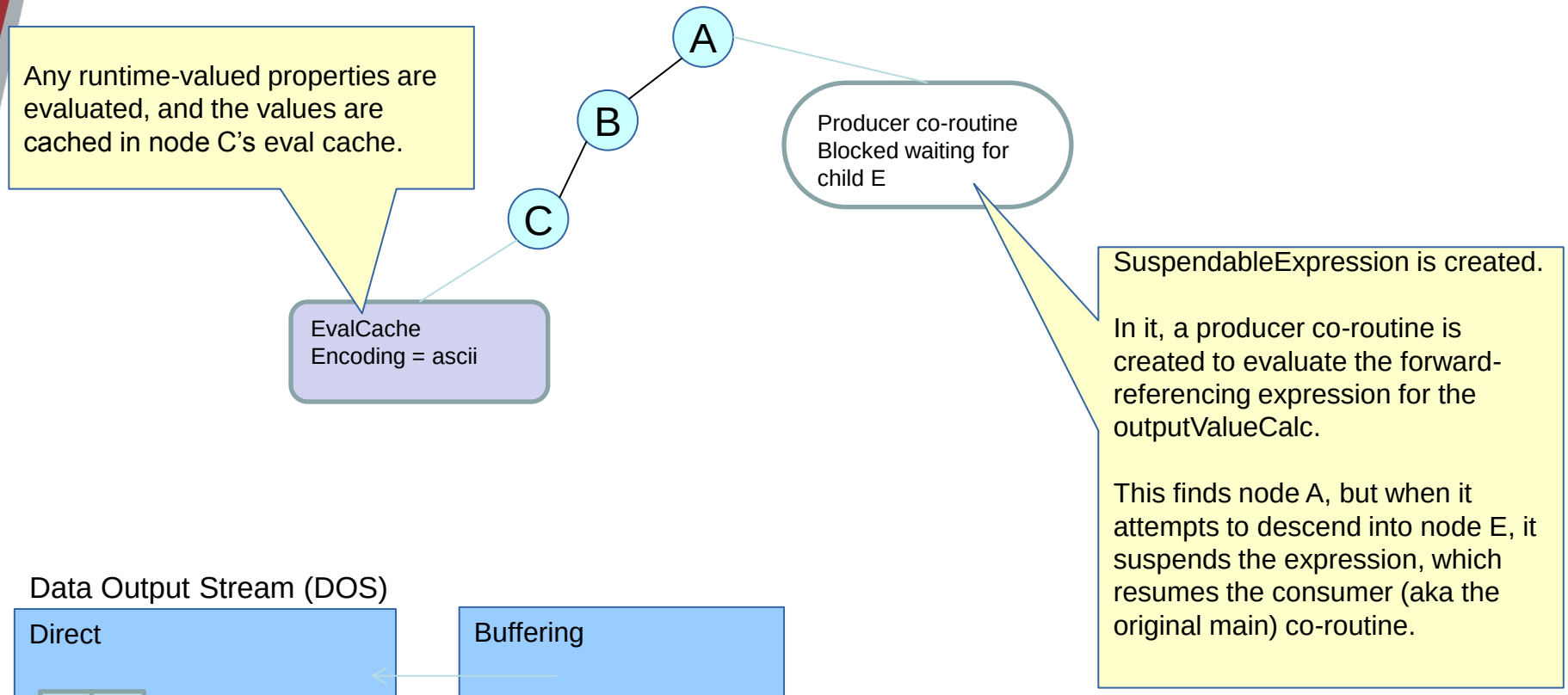
Element C, which has OVC is created because it is a required child of Element B.



Data output stream is split into a buffering part, and the original direct part.

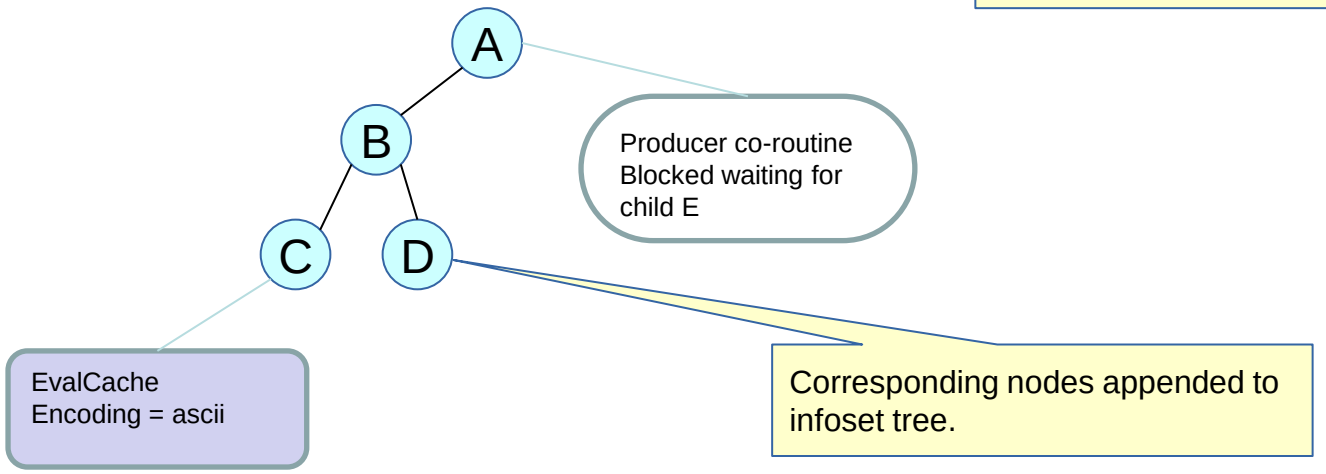


XML Events: <a><b>…

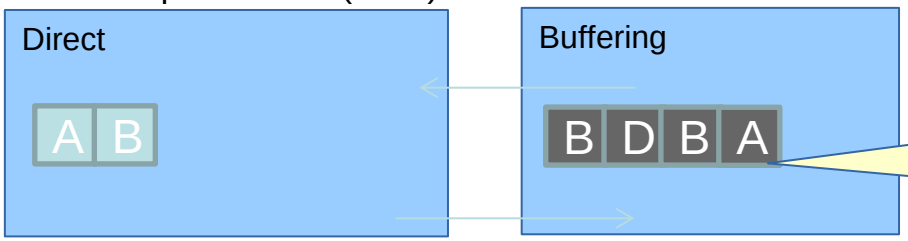


XML Events: <a> **<d></d>**

Unparsing (on original main thread) continues and pulls events.



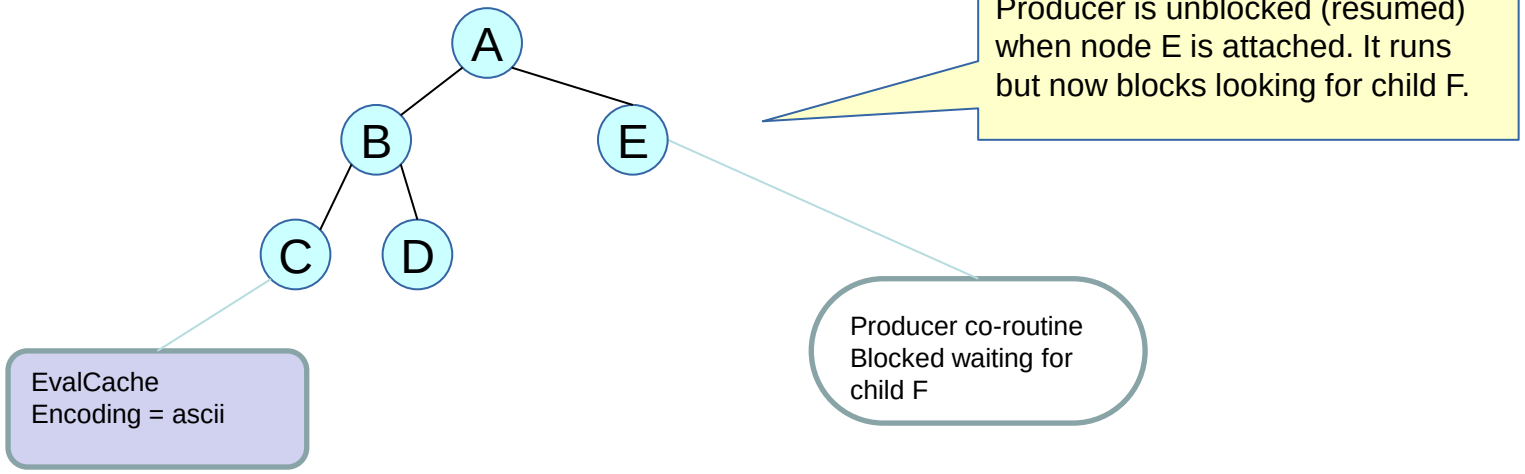
Data Output Stream (DOS)

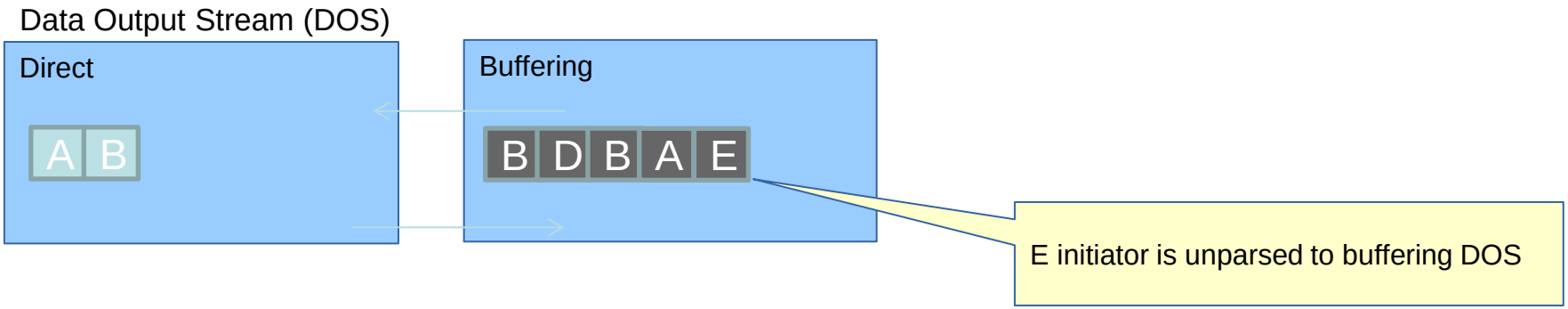


Unparsed data output goes to the buffering DOS
The pieces are:

- B separator (between C and D)
- D
- B terminator
- A separator (between B and E)

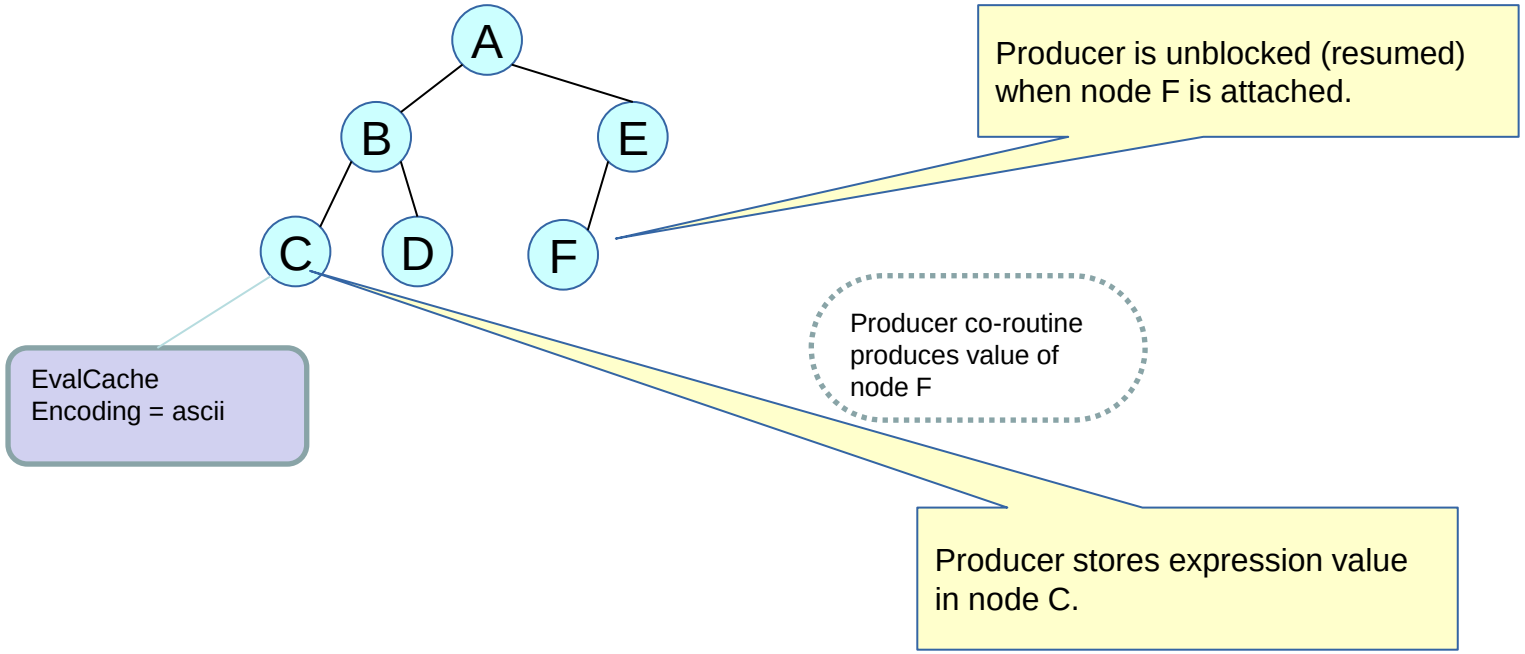
XML Events: <a> <d></d> **<e>** …



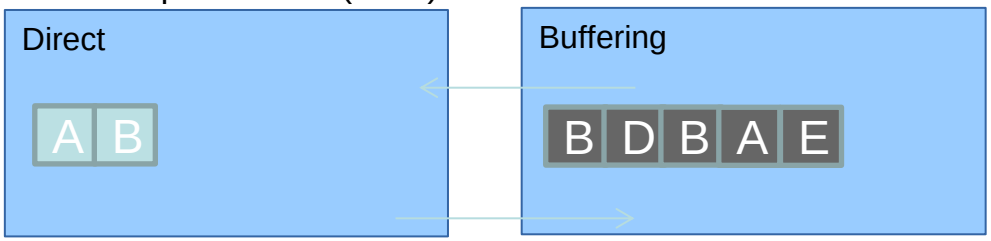


TRESYS TECHNOLOGY

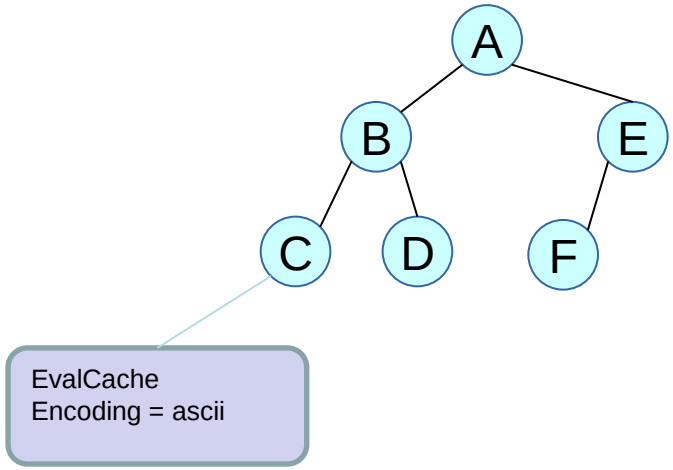
XML Events: <a> <d></d><e> **<f></f>** …



Data Output Stream (DOS)

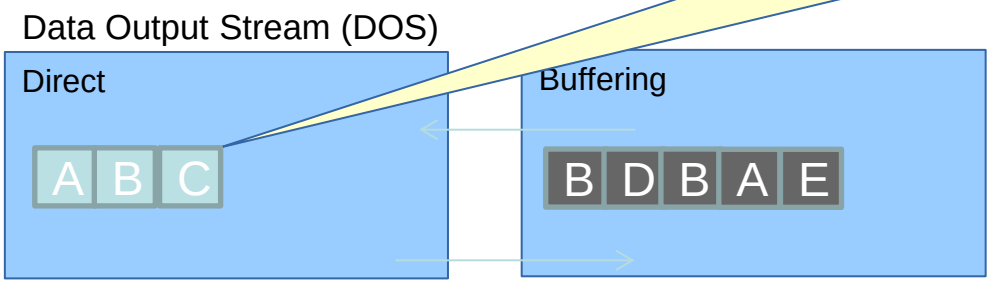


XML Events: <a> <d></d><e> <f></f>…

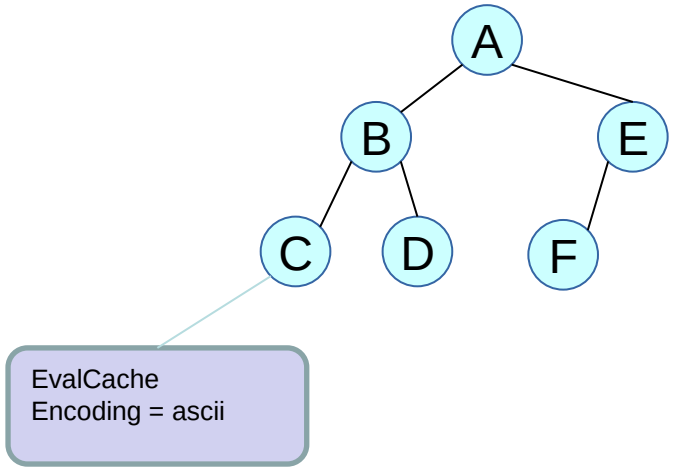


Producer co-routine then unparses node C into original DOS.

For this unparse, properties with runtime values are taken from the EvalCache of node C, where these values have already been computed and cached.

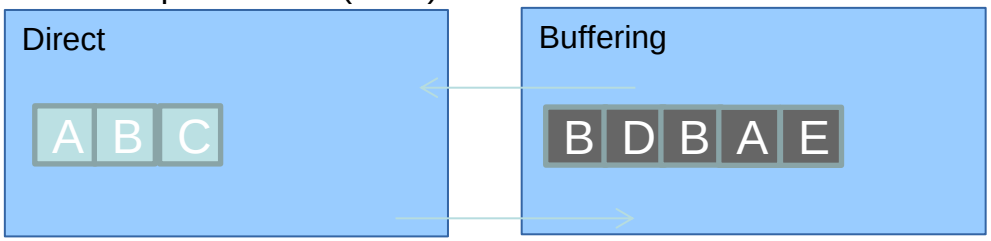


XML Events: <a> <d></d><e> <f></f>…

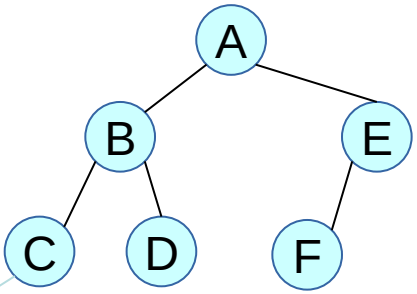


Producer co-routine then calls setFinished on original DOS.

Data Output Stream (DOS)



XML Events: <a> <d></d><e><f></f>…



EvalCache
Encoding = ascii

Data output streams are collapsed!

- Buffered DOS morphs into a direct.
- Buffered data is output to java.io.OutputStream which becomes output stream in this DOS

Old “original” DOS is discarded.

Producer co-routine terminates (and is discarded), resuming the original “consumer” co-routine, which is the original unparser “main” thread.

Data Output Stream (DOS)

Deallocated

Direct

That ends the scenario. We’re back to unparsing on the original main unparse thread, to a direct DOS.