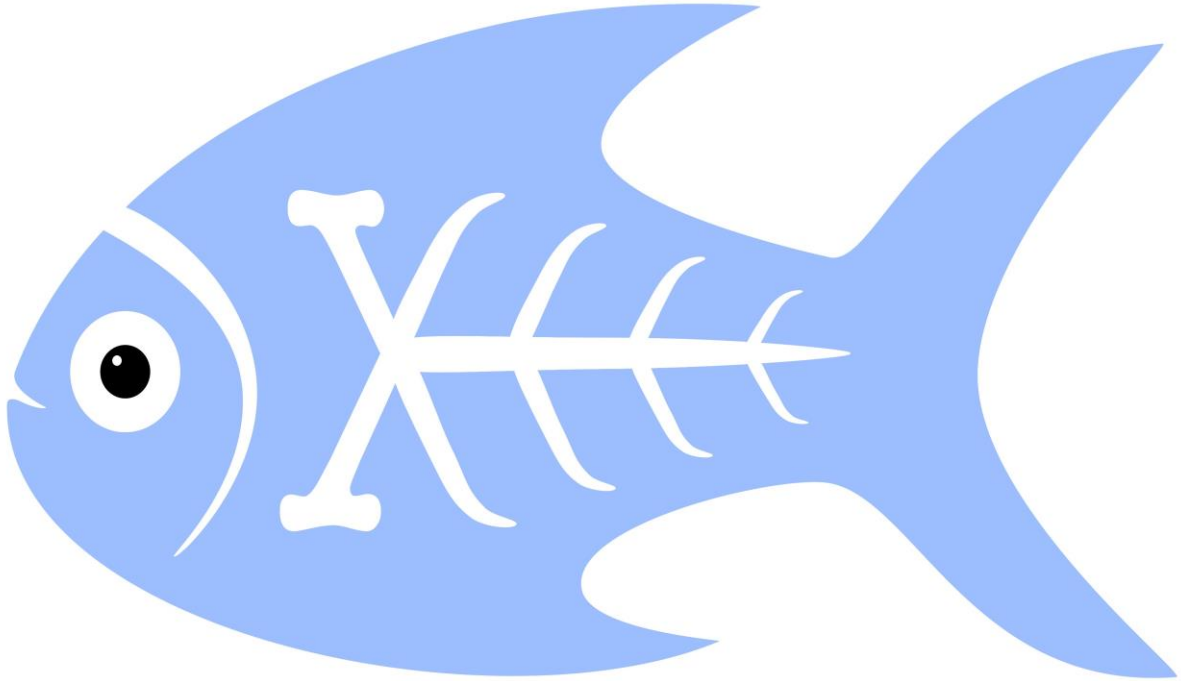
Introduction to XProc 3.0

The 19th ACM Symposium on
Document Engineering
September 26-29 2019, Berlin



While waiting, maybe you can:

- Install (or check) the Java JRE (1.8.x): https://www.java.com/en/download/
- Download and unpack the course materials: https://github.com/eriksiegel/DocEng-2019-XProc/releases

Who Am I?

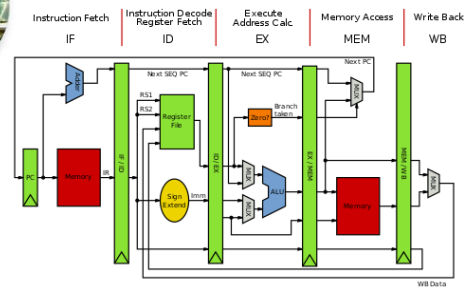
- Erik Siegel
- Content Engineer and XML specialist
- One-man company: Xatapult
  - Groningen, The Netherlands
- Customers mostly in publishing and standardization
- Part of the XProc 3.0 editing committee
- Contact:

erik@xatapult.nl
www.xatapult.com
www.linkedin.com/in/esiegel/
+31 6 53260792

XProc?

- XProc is an XML based programming language for complex data processing - pipelining
- Extensible set of small, sharp tools for creating and transforming XML and other documents
- V1.0 available (two processor implementations to run your pipelines)
- Specification and implementation V3.0 under development

Why should I bother?



- Pipelines are ubiquitous all around us
- Solve problems with a set of small, sharp tools that combine in many ways
  - Like the UNIX command line
- Very natural choice for document processing
- Compose small tools into something bigger, pipelines…
- XProc beats the alternatives

A successful example of large-scale application of XProc (1.0) pipelines doing document engineering:
https://www.le-tex.de/en/transpect.html

History and status 1

- V1 (2010) turned out to be
  - hard to use and understand
  - verbose
  - Became outdated with respect to underlying standards
- V2 Initiative (non-XML) – not enough support
  - And, besides, all the important XML standards used 3.x …
- V3 Initiative (2016) - W3C Community Group
  - Stay close to existing syntax
  - Make language more usable, understandable and concise
  - Update underlying standards
  - Allow other document types to flow through
  - Clean up loose ends
- Editors
  - Norman Walsh, Achim Berndzen, Gerrit Imsieke, Erik Siegel

History and status 2

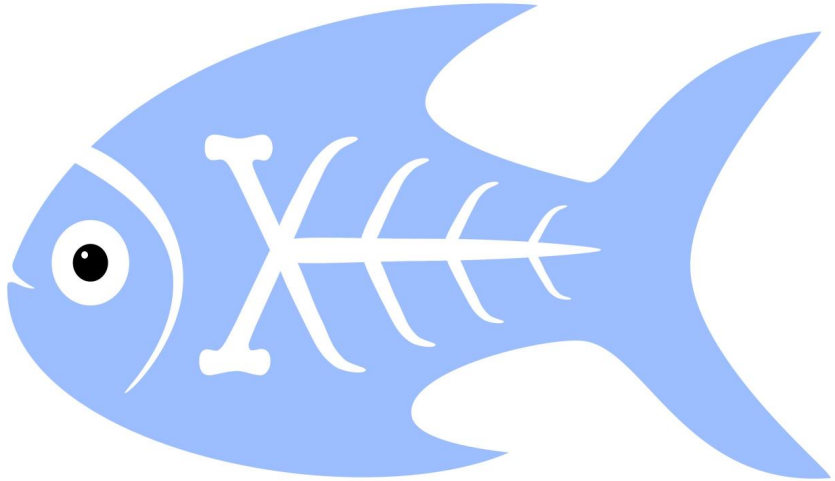
- Final call core spec
- Working on the steps
- Plan: End before end 2019
- In the making:
  - A specification (http://spec.xproc.org)
  - Two processor implementations (XML Calabash, MorganaXProc)
  - A programmer's reference book

And its name is…

Kanava

Logo!

Thanks to
Bethan Tovey



Links

- XProc 1.0:
  - Specification: https://www.w3.org/TR/xproc/
  - XML Calabash processor: https://xmlcalabash.com/
  - Morgana XProc processor: https://www.xml-project.com/
- XProc 3.0:
  - Specification: **http://spec.xproc.org**
  - Github: https://github.com/xproc/
  - W3C: https://www.w3.org/community/xproc-next/

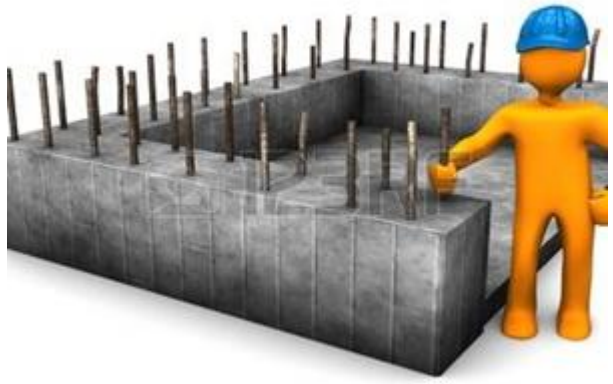
Next meeting of the XProc 3.0 working group:
November 9-10, Cologne. **Feel free to join!**
(https://github.com/xproc/Workshop-2019-11)

Hands-on: Installation and pre-flight check

- Go to https://github.com/eriksiegel/DocEng-2019-XProc/releases and download the latest release zip
- Unzip this somewhere on your machine
- JRE (V1.8.x works, maybe others too…) must be installed! https://www.java.com/en/download/
- Open a command window in **`exercises/01-hello-xproc/`**
- Try **`run.bat`** or **`run.sh`**

```
==================================
MorganaXProc-III 0.8.21-alpha
Copyright 2011-2019 by <xml-project /> Achim Berndzen
==================================

<hello-xproc timestamp="2019-08-22T11:37:55+01:00"/>
```

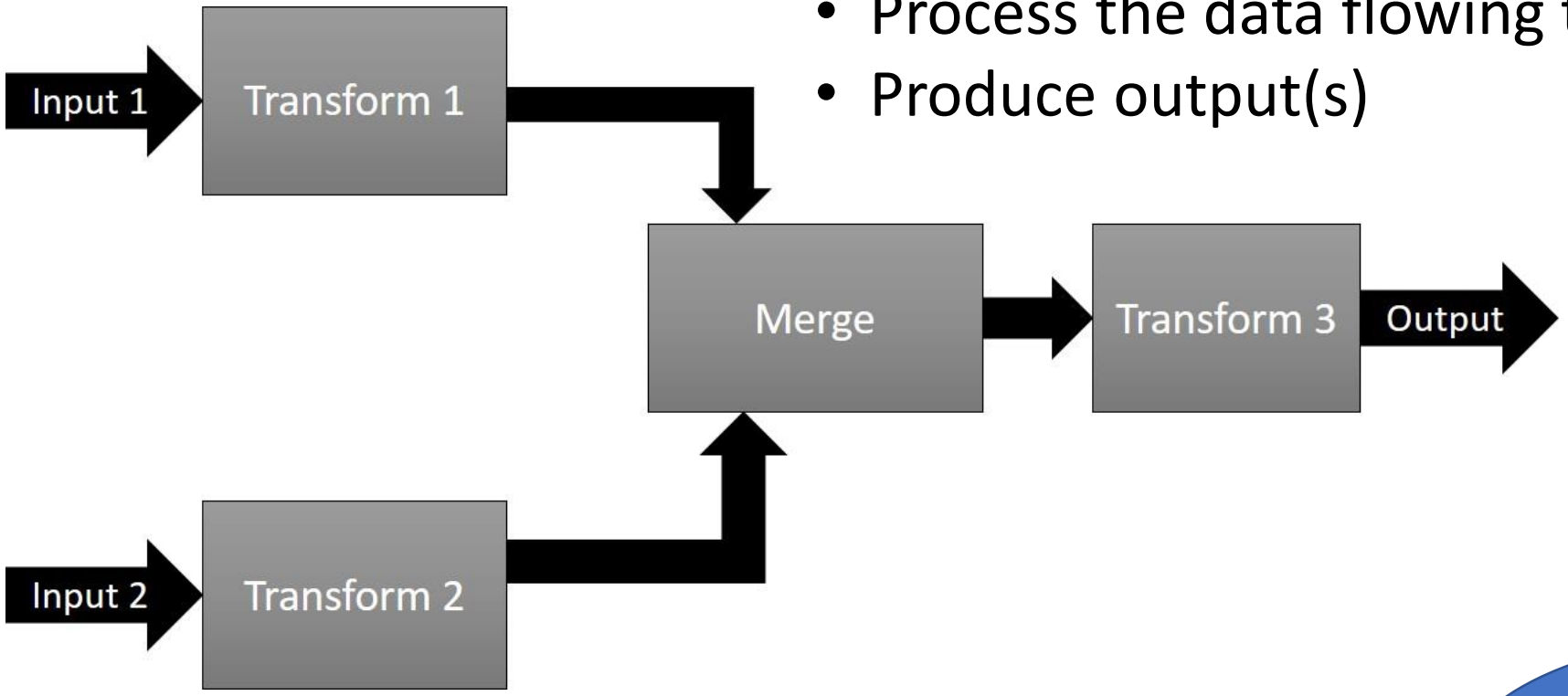
XProc fundamentals



You need to understand this!

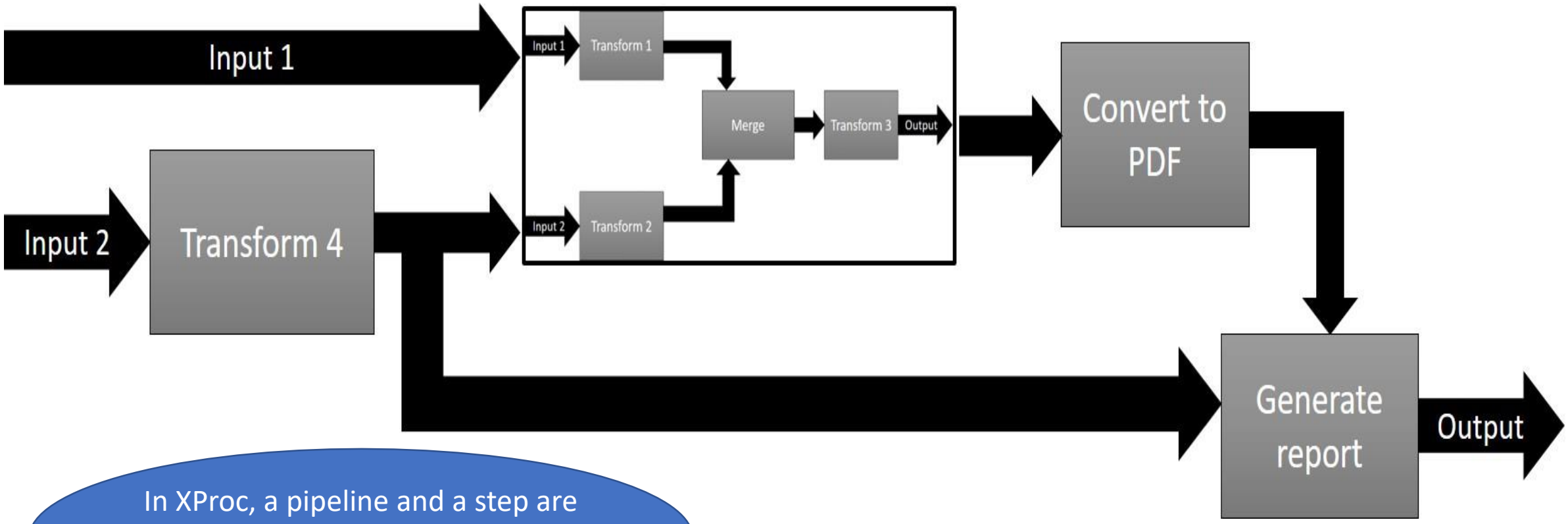
Pipelines, steps

- Document(s) as input
- Process the data flowing through using steps
- Produce output(s)



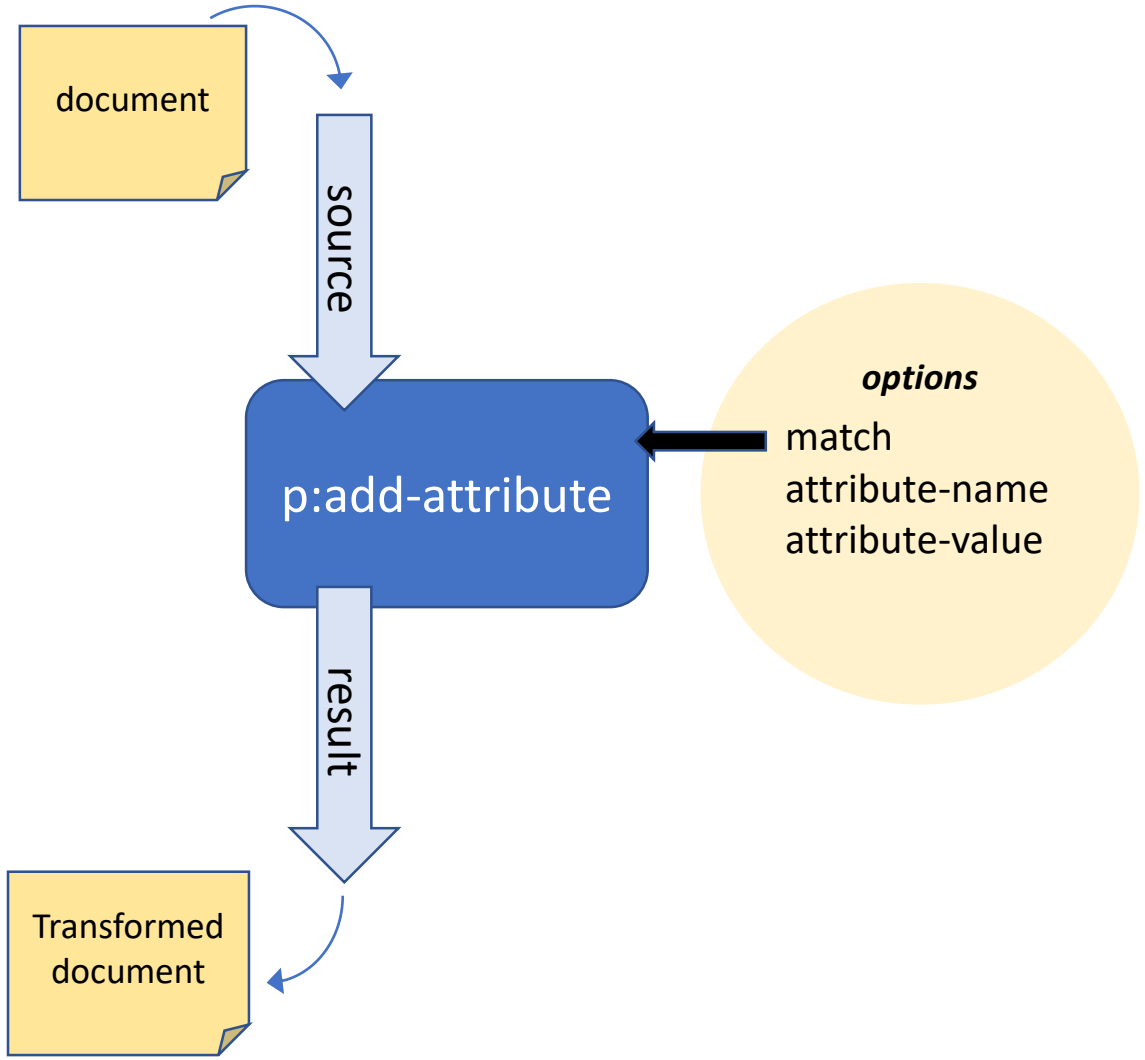
Documents can be of any type, not just XML!

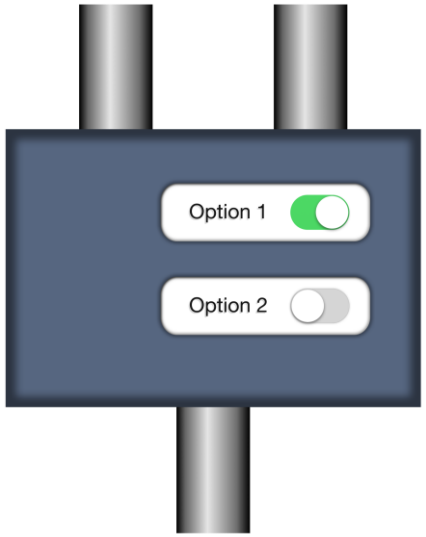
Pipelines, steps



In XProc, a pipeline and a step are essentially the same. The terms can be used interchangeably!

Steps/pipelines, ports, options





Have a look ate the step specification:
http://spec.xproc.org/master/head/steps/#c.add-attribute

Step/pipeline that adds an attribute - root

Root p:declare-step element

Namespace and preferred prefix (p:)

XProc version

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0"
     name="add-attribute-pipeline">

  <p:input port="source"/>

  <p:output port="result">
    <p:pipe step="add-attribute-1" port="result"/>
  </p:output>

  <p:add-attribute name="add-attribute-1">
    <p:with-input port="source">
      <p:pipe step="add-attribute-pipeline" port="source"/>
    </p:with-input>
    <p:with-option name="match" select="'/*'"/>
    <p:with-option name="attribute-name" select="'timestamp'"/>
    <p:with-option name="attribute-value" select="current-dateTime()"/>
  </p:add-attribute>

</p:declare-step>
```

Step/pipeline that adds an attribute - in/output ports

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0"
     name="add-attribute-pipeline">

  <p:input port="source"/>

  <p:output port="result">
    <p:pipe step="add-attribute-1" port="result"/>
  </p:output>

  <p:add-attribute name="add-attribute-1">
    <p:with-input port="source">
      <p:pipe step="add-attribute-pipeline" port="source"/>
    </p:with-input>
    <p:with-option name="match" select="'/*'"/>
    <p:with-option name="attribute-name" select="'timestamp'"/>
    <p:with-option name="attribute-value" select="current-dateTime()"/>
  </p:add-attribute>

</p:declare-step>
```

Input port

Output port

Emit the result of an output port of a step in *this* pipeline

Step/pipeline that adds an attribute - connect a port

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0"
     name="add-attribute-pipeline">

  <p:input port="source"/>

  <p:output port="result">
    <p:pipe step="add-attribute-1" port="result"/>
  </p:output>

  <p:add-attribute name="add-attribute-1">
    <p:with-input port="source">
      <p:pipe step="add-attribute-pipeline" port="source"/>
    </p:with-input>
    <p:with-option name="match" select="'/*'"/>
    <p:with-option name="attribute-name" select="'timestamp'"/>
    <p:with-option name="attribute-value" select="current-dateTime()"/>
  </p:add-attribute>

</p:declare-step>
```

Connect the source port of p:add-attribute to what flows into the encompassing step

Port connections are always defined on the *input* ports

Step/pipeline that adds an attribute - set options

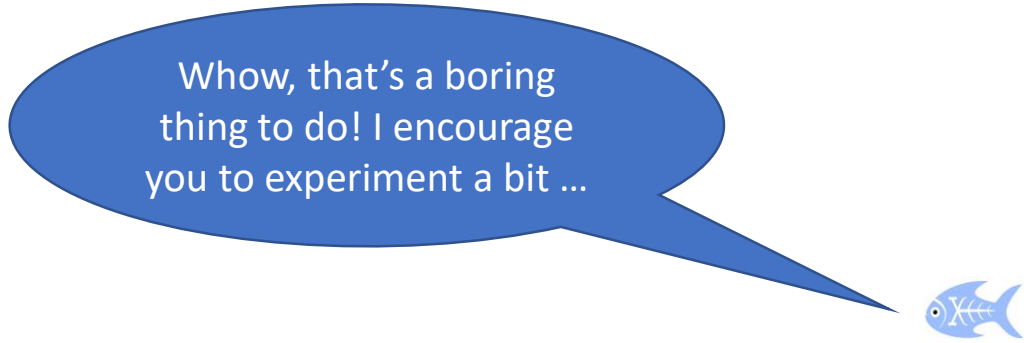
```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0"
     name="add-attribute-pipeline">

  <p:input port="source"/>

  <p:output port="result">
    <p:pipe step="add-attribute-1" port="result"/>
  </p:output>

  <p:add-attribute name="add-attribute-1">
    <p:with-input port="source">
      <p:pipe step="add-attribute-pipeline" port="source"/>
    </p:with-input>
    <p:with-option name="match" select="'/*'"/>
    <p:with-option name="attribute-name" select="'timestamp'"/>
    <p:with-option name="attribute-value" select="current-dateTime()"/>
  </p:add-attribute>

</p:declare-step>
```

Set the 3 options of p:add-attribute

XProc uses XPath 3.1 for all expressions, just like XSLT 3.0 and XQuery 3.1

👉 Hands-on: Add a second attribute

- Open a command window in **`exercises/02-add-attribute-1/`**
- Try **`run.bat`** or **`run.sh`**.
  - The input comes from **`input.xml`** in the same directory
- Change the pipeline and add a *second* **`p:add-attribute`** step that also adds an **`enabled="true"`** attribute to the root element.
  - Connect the **`source`** port of your new **`p:add-attribute`** explicitly to the **`result`** port of the existing **`p:add-attribute`**.
- Try it!



Add a second attribute - solution

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0"
    name="add-attribute-pipeline">

  <p:input port="source"/>

  <p:output port="result">
    <p:pipe step="add-attribute-2" port="result"/>
  </p:output>

  <p:add-attribute name="add-attribute-1">
    …
  </p:add-attribute>

  <p:add-attribute name="add-attribute-2">
    <p:with-input port="source">
      <p:pipe step="add-attribute-1" port="result"/>
    </p:with-input>
    <p:with-option name="match" select="'/*'"/>
    <p:with-option name="attribute-name" select="'enabled'"/>
    <p:with-option name="attribute-value" select="true()"/>
  </p:add-attribute>

</p:declare-step>
```

Get the final result from the *new* p:add-attribute

Read from the result port of the *first* p:add-attribute

Awkward and verbose. I don't like it!

Connect ports using the pipe attribute

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0"
name="add-attribute-pipeline">

  <p:input port="source"/>

  <p:output port="result" pipe="result@add-attribute-1"/>

  <p:add-attribute name="add-attribute-1">
    <p:with-input port="source" pipe="source@add-attribute-pipeline"/>
    <p:with-option name="match" select="'/*'"/>
    <p:with-option name="attribute-name" select="'timestamp'"/>
    <p:with-option name="attribute-value" select="current-dateTime()"/>
  </p:add-attribute>

</p:declare-step>
```

Use a shorthand pipe="port@step" notation

Same here

Hands-on: Add a second attribute using the pipe attribute

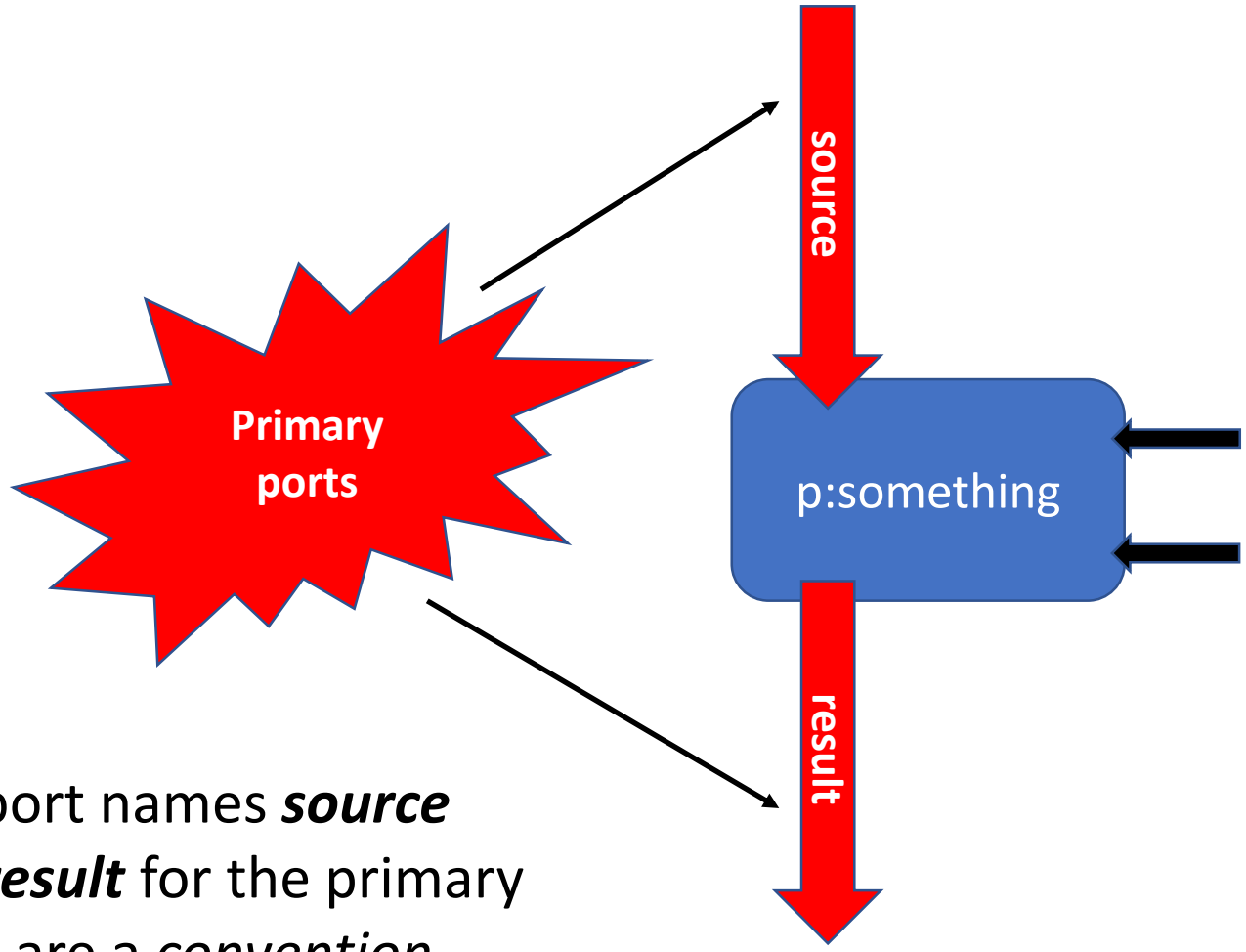
- Open a command window in **`exercises/03-add-attribute-2/`**
- Try **`run.bat`** or **`run.sh`**.
- Change the pipeline and add a second **`p:add-attribute`** step that also adds an **`enabled="true"`** attribute to the root element.
  - Connect the **`source`** port of your new **`p:add-attribute`** explicitly to the **`result`** port of the existing **`p:add-attribute`** using a **`pipe`** attribute.
- Try it!

Add a second attribute using the pipe attribute - solution

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0"
name="add-attribute-pipeline">

  <p:input port="source"/>

  <p:output port="result" pipe="result@add-attribute-2"/>

  <p:add-attribute name="add-attribute-1">
    <p:with-input port="source" pipe="source@add-attribute-pipeline"/>
    <p:with-option name="match" select="'/*'"/>
    <p:with-option name="attribute-name" select="'timestamp'"/>
    <p:with-option name="attribute-value" select="current-dateTime()"/>
  </p:add-attribute>

  <p:add-attribute name="add-attribute-2">
    <p:with-input port="source" pipe="result@add-attribute-1"/>
    <p:with-option name="match" select="'/*'"/>
    <p:with-option name="attribute-name" select="'enabled'"/>
    <p:with-option name="attribute-value" select="true()"/>
  </p:add-attribute>

</p:declare-step>
```

Better, shorter, but still…
Why would I need to explicitly connect ports *at all* if its clear that they should connect anyway?

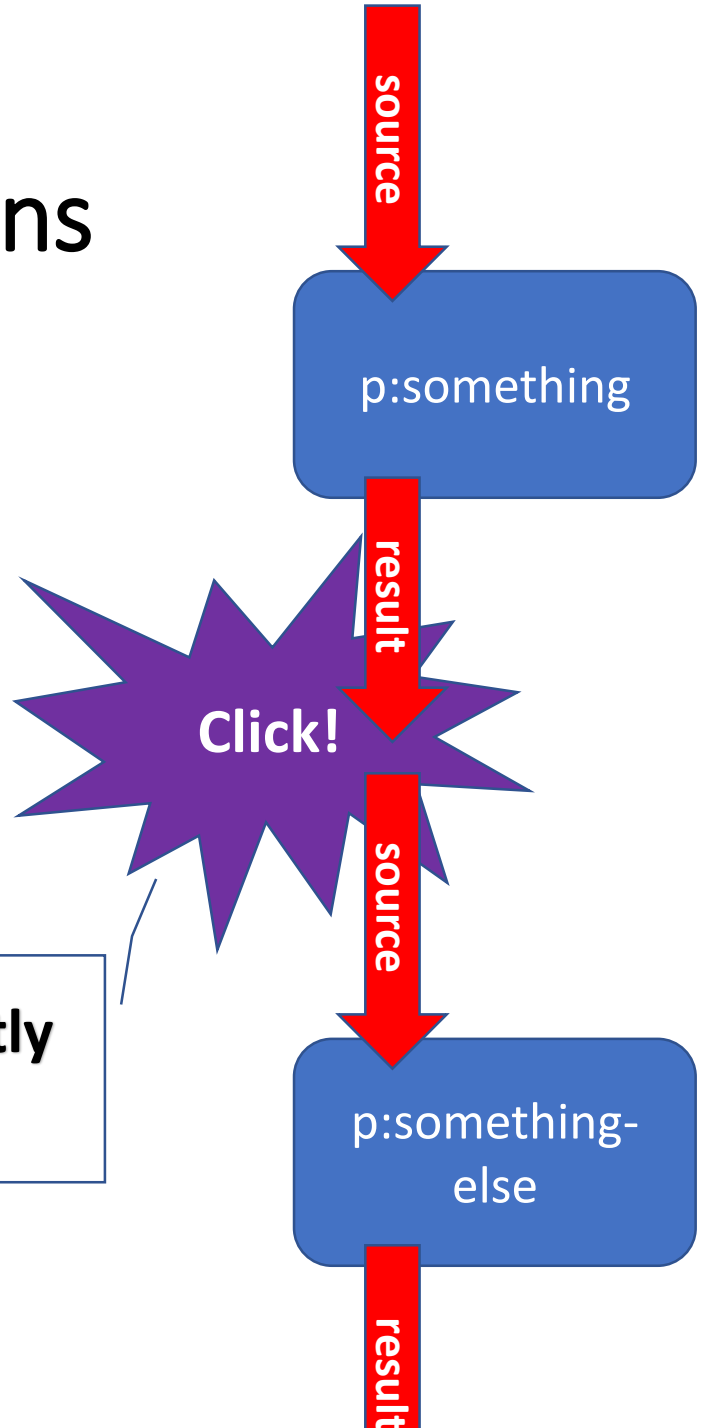
Primary ports



The port names ***source*** and ***result*** for the primary ports are a *convention*

Not all ports are created equal…

Primary ports, implicit connections



Primary ports implicitly connect

Think of primary ports having little magnets that *snap* automagically together

Primary ports, implicit connections

If a step has only a single input or output port, they're primary by default. But you can set the primary status *explicitly* using a primary="true/false" attribute here.

Implicit connection of primary input port to first step

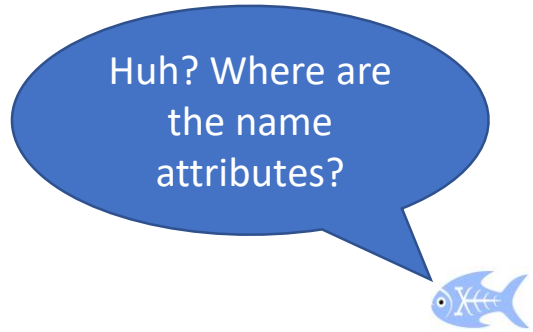
Implicit connection of last step to primary output port

Implicit connection of steps

```
<p:declare-step … >

  <p:input port="source"/>
  <p:output port="result"/>

  <p:add-attribute>
    …
  </p:add-attribute>


  <p:add-attribute >
    …
  </p:add-attribute >


</p:declare-step>
```

Connect ports implicitly

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0">

  <p:input port="source"/>

  <p:output port="result"/>

  <p:add-attribute>
    <p:with-option name="match" select="'/*'"/>
    <p:with-option name="attribute-name" select="'timestamp'"/>
    <p:with-option name="attribute-value" select="current-dateTime()"/>
  </p:add-attribute>

</p:declare-step>
```

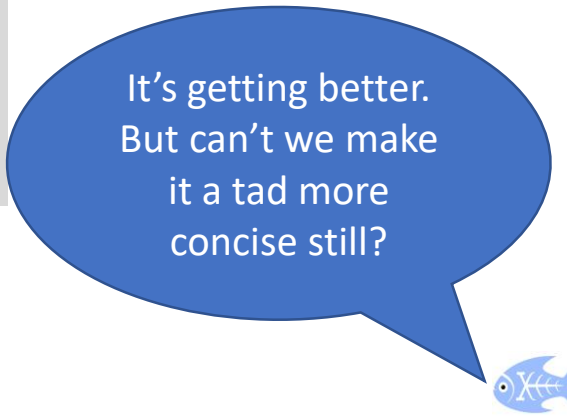


Hands-on: Add a second attribute using implicit connections

- Open a command window in **`exercises/04-add-attribute-3/`**
- Try **`run.bat`** or **`run.sh`**.
- Change the pipeline and add a second **`p:attribute`** step that also adds an **`enabled="true"`** attribute to the root element.
  - Use the primary ports *implicit* connections
- Try it!

Connect ports implicitly - solution

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0">

  <p:input port="source"/>

  <p:output port="result"/>

  <p:add-attribute>
    <p:with-option name="match" select="'/*'"/>
    <p:with-option name="attribute-name" select="'timestamp'"/>
    <p:with-option name="attribute-value" select="current-dateTime()"/>
  </p:add-attribute>

  <p:add-attribute>
    <p:with-option name="match" select="'/*'"/>
    <p:with-option name="attribute-name" select="'enabled'"/>
    <p:with-option name="attribute-value" select="true()"/>
  </p:add-attribute>

</p:declare-step>
```



Setting options using attributes

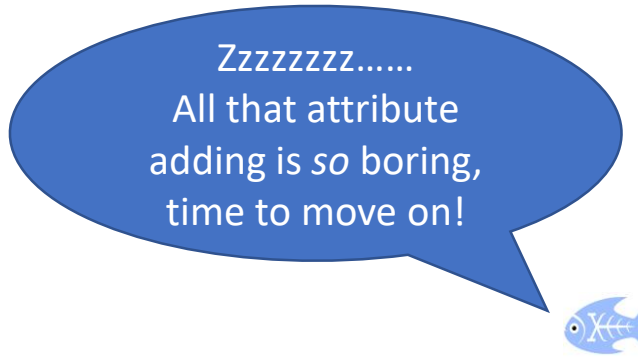
```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0">

  <p:input port="source"/>

  <p:output port="result"/>

  <p:add-attribute match="/*" attribute-name="timestamp"
     attribute-value="{current-dateTime()}"/>

</p:declare-step>
```

You can specify option values as attributes on the step invocation

To compute stuff use Attribute-Value-Templates (AVTs), just like in XSLT.

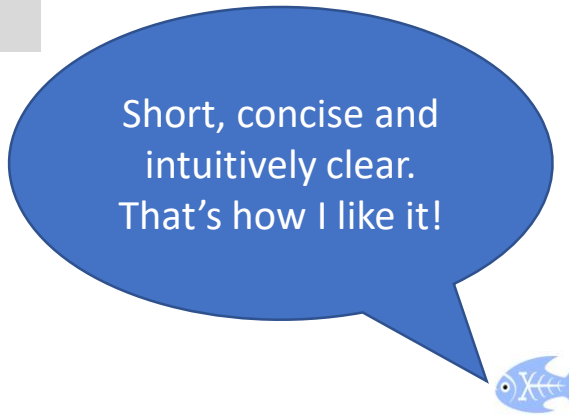
Hands-on: Add a second attribute using option values set by attributes

- Open a command window in **`exercises/05-add-attribute-4/`**
- Try **`run.bat`** or **`run.sh`**.
- Change the pipeline and add a second **`p:add-attribute`** step that also adds an **`enabled="true"`** attribute to the root element.
  - Set all the options by attributes
- Try it!



Set options using attributes - solution

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0">

  <p:input port="source"/>

  <p:output port="result"/>

  <p:add-attribute match="/*" attribute-name="timestamp"
      attribute-value="{current-dateTime()}"/>

  <p:add-attribute match="/*" attribute-name="enabled"
      attribute-value="true"/>

</p:declare-step>
```



Intermezzo 1: Your own options

- We've seen that built-in steps can have options
  - `match`
  - `attribute-name`
  - `attribute-value`
- What if you want to add an option to your own step?

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0">

  <p:input port="source"/>
  <p:output port="result"/>

  <p:option name="username"/>

  <p:add-attribute match="/*" attribute-name="username"
      attribute-value="{upper-case($username)}"/>

</p:declare-step>
```

Declare the option in the prolog of your step

Reference the option using the $... notation, just like XSLT and XQuery

You can make an option required, set a datatype, supply a default, etc.

See: **exercises/06-own-option/**

Intermezzo 2: Variables

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0">

  <p:input port="source"/>
  <p:output port="result"/>

  <p:option name="username"/>

  <p:variable name="id" select="upper-case($username) || '-' ||
    p:system-property('p:episode')" />

  <p:add-attribute match="/*" attribute-name="id" attribute-value="{$id}"/>

</p:declare-step>
```

Declare the variable anywhere

Reference the variable using the $... notation, just like XSLT and XQuery

Variables can be of *any* datatype, just like in XSLT or XQuery

See: **exercises/07-variables/**

Intermezzo 2: Variables

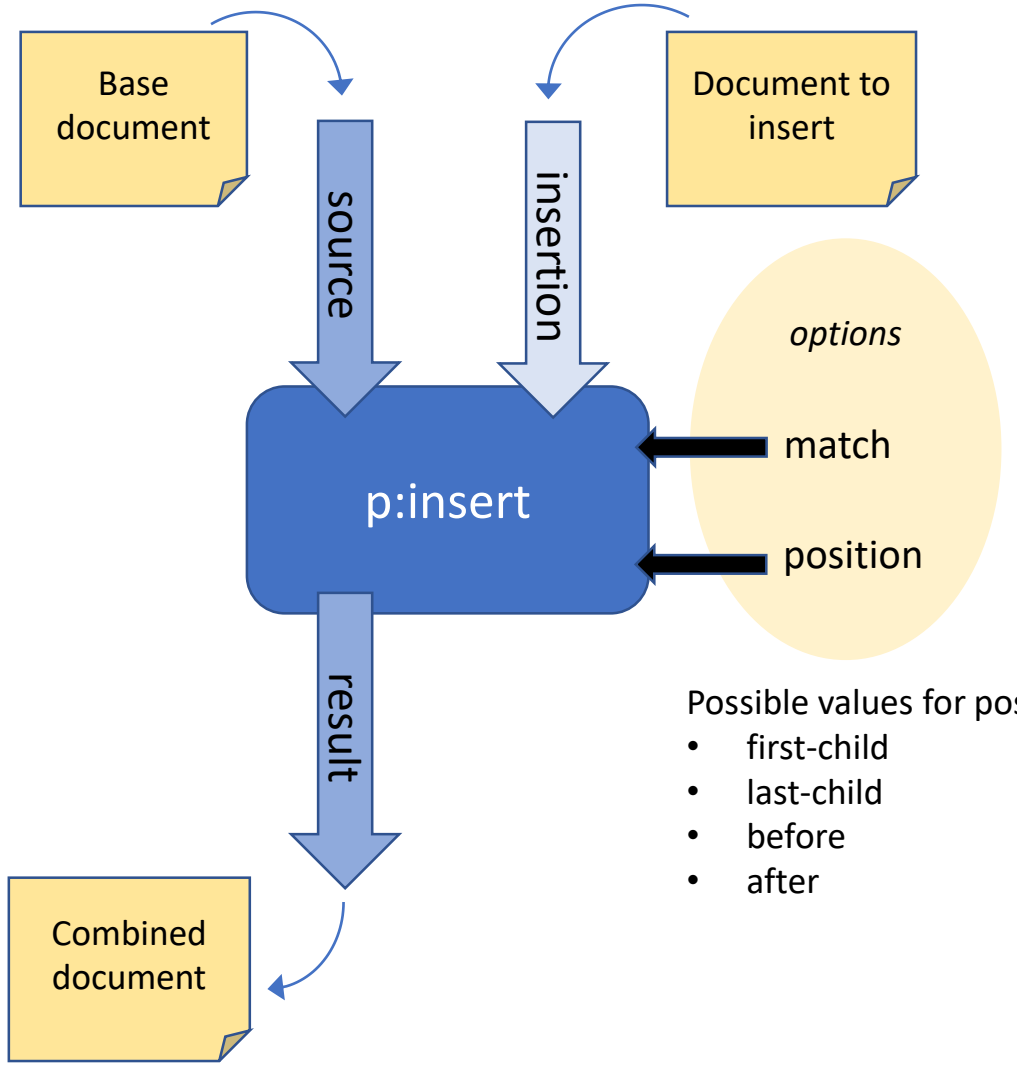
```
<some-root status="final">
  …
</p:some-root>
```

Read values from the document flowing through!

```
<p:variable name="status" select="/*/@status" />
```

You can even base its value on something flowing from another output port

The p:insert step



Possible values for position:

- first-child
- last-child
- before
- after

See:
http://spec.xproc.org/master/head/steps/#c.insert

The source and result port are primary, the insertion port is not…

Connect a port to an inline document

```
<p:some-step ...>

  <p:with-input port="port-name">
    <p:inline>
      ... (inline document) ...
    </p:inline>
  </p:with-input>

  ...

</p:some-step>
```

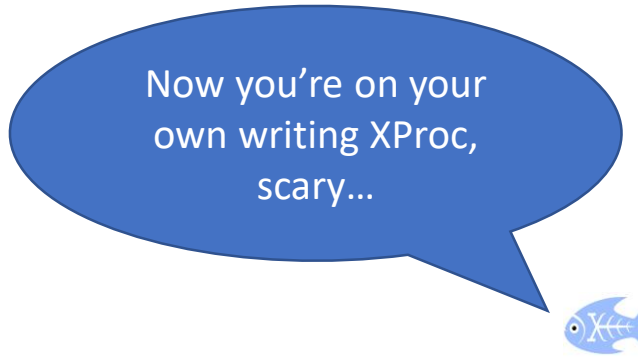
In most cases you can leave out the <p:inline> wrapper

You can use expressions between curly braces {...} in your inline document

Expressions between curly braces are called TVTs (Text-Value-Templates and AVTs (Attribute-Value-Templates)

Hands-on: Add an additional child element using an inline document

- Open a command window in **`exercises/08-connect-inline/`**
- Finish the pipeline so it adds a **`<location>Berlin 2019</location>`** element after the **`<presenter>`** element.
  - Use the **`p:insert`** step
  - Use a **`<p:inline>`** wrapper
- Compute the current year using a **`{…}`** construction
  - XPath cheat: `year-from-date(current-date())`
- Try it with **`run.bat`** or **`run.sh`**
- Remove the **`<p:inline>`** wrapper and try again. Any differences?



Insert inline document - solution

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0">

  <p:input port="source"/>
  <p:output port="result"/>

  <p:insert match="/*" position="last-child">
    <p:with-input port="insertion">
      <p:inline>
        <location>Berlin {year-from-date(current-date())}</location>
      </p:inline>
    </p:with-input>
  </p:insert>

</p:declare-step>
```

Connect a port to an external document

```
<p:some-step ...>

  <p:with-input port="port-name">
    <p:document href="reference-to-document"/>
  </p:with-input>

  ...

</p:some-step>
```

In most cases you can put the href attribute directly on the <p:with-input>, no need for a <p:document> then!

The href attribute is an AVT: You can use expressions between curly braces {…} inside

We have no means to add the current year now, like we did in the last exercise…

Hands-on: Add an additional child element using an external document

- Open a command window in **`exercises/09-connect-external/`**
- Finish the pipeline so it adds the contents of **`insert.xml`** after the **`<presenter>`** element.
  - Use the **`p:insert`** step
  - Use a **`<p:document>`** element
- Try it with **`run.bat`** or **`run.sh`**
- Put the **`href`** attribute directly on the **`<p:with-input>`**
  Any differences?
- Can you add a variable with the name of the file and use that to reference it?

Insert external document - solution

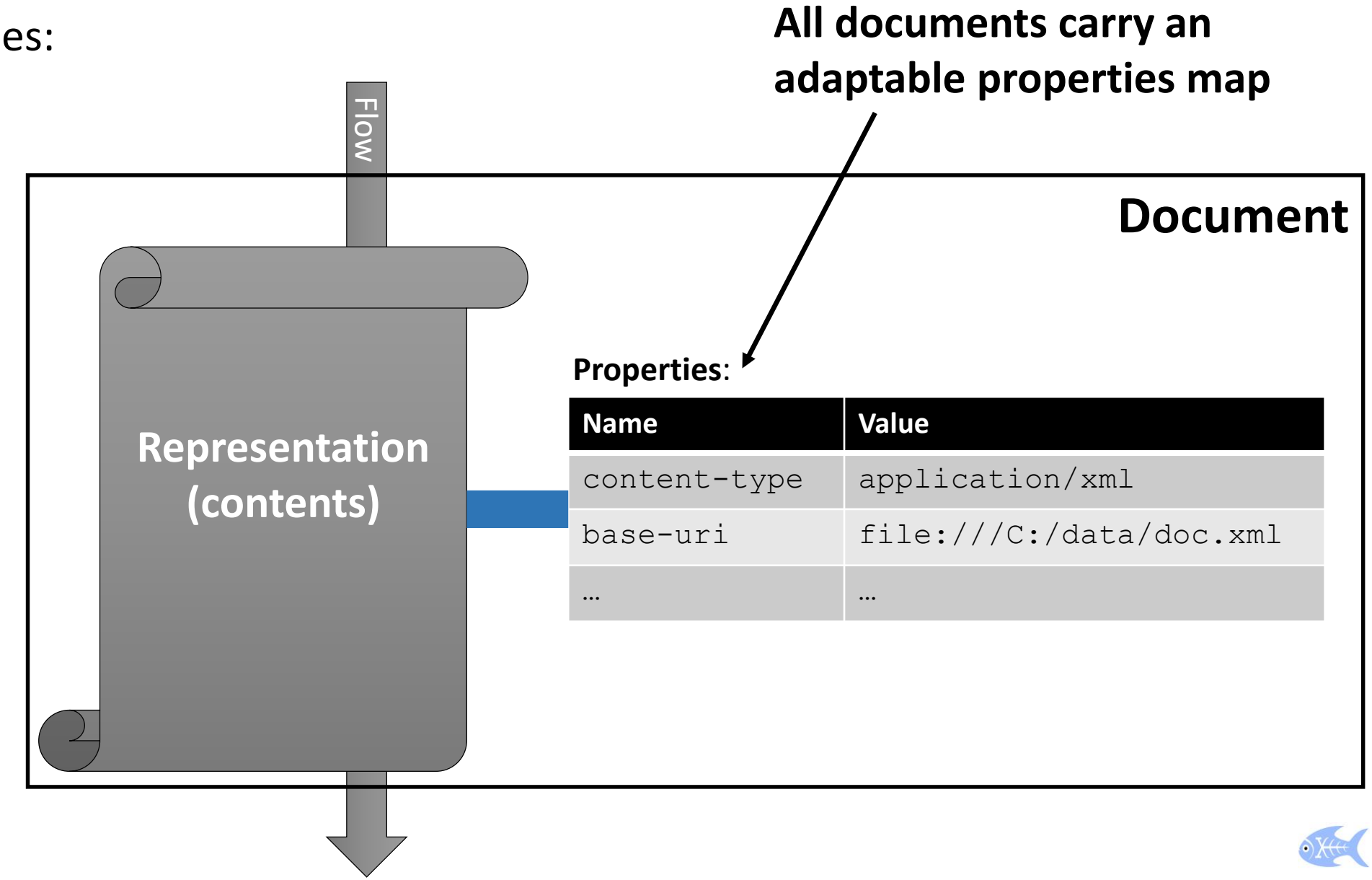
```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0">

  <p:input port="source"/>
  <p:output port="result"/>

  <p:insert match="/*" position="last-child">
    <p:with-input port="insertion">
      <p:document href="insert.xml"/>
    </p:with-input>
  </p:insert>

</p:declare-step>
```

Intermezzo: Documents flowing through

Native document types:

- XML
- HTML
- JSON
- Text
- Other

It is also possible to flow multiple documents or none at all.



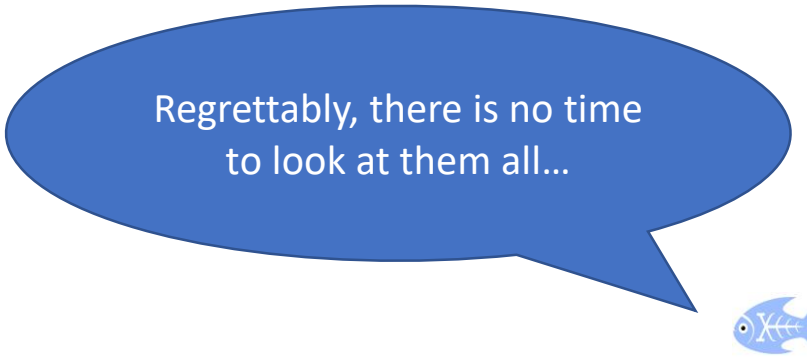
Intermezzo: The step libraries

- Standard steps, see http://spec.xproc.org/master/head/steps/
  - These steps *must* be there in a conformant XProc processor!

- Additional steps, see http://spec.xproc.org/master/head/#steps/
  - Implementation is optional (but recommended)
  - If such a step is implemented it must conform to what is written there

Intermezzo: The core (or compound) steps

- **`p:for-each`**: loop over multiple documents or parts of a document
- **`p:choose / p:when / p:otherwise`**: Make choices
- **`p:if`**: Make a single choice (there is no else)
- **`p:viewport`**: Work on only a part of a document
- **`p:try / p:catch`**: Error catching and handling
- **`p:group`**: Grouping of instructions



Use p:for-each to split a document - Input

```
<documents>

  <doc filename="output1.xml">
    <contents>This is document number 1</contents>
  </doc>

  <doc filename="output2.xml">
    <contents>This is document number 2</contents>
    <more>It has some more...</more>
  </doc>

</documents>
```

Split this in multiple documents

The filenames are in filename attributes

Boring contents
Erik

Use p:for-each to split a document – Basic pipeline

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0">

  <p:input port="source"/>
  <p:output port="result"/>

  <p:for-each>
    <p:with-input select="//doc"/>

    <p:store href="{/*/@filename}"/>
  </p:for-each>

</p:declare-step>
```

p:for-each has an anonymous input port…

p:store stores a document to disk. The href attribute tells it where

p:store emits on its result port the same document as it received on its source port

select is a standard attribute of p:with-input

Hands-on: Use p:for-each to split a document 1

- Open a command window in **`exercises/10-for-each-1/`**
- Try it with **`run.bat`** or **`run.sh`**
- It does not run… why? What does the error message tell you?

- Make the output port of the pipeline accept a sequence by adding a **`sequence="true"`** attribute

Use p:for-each to split a document 1 – Solution

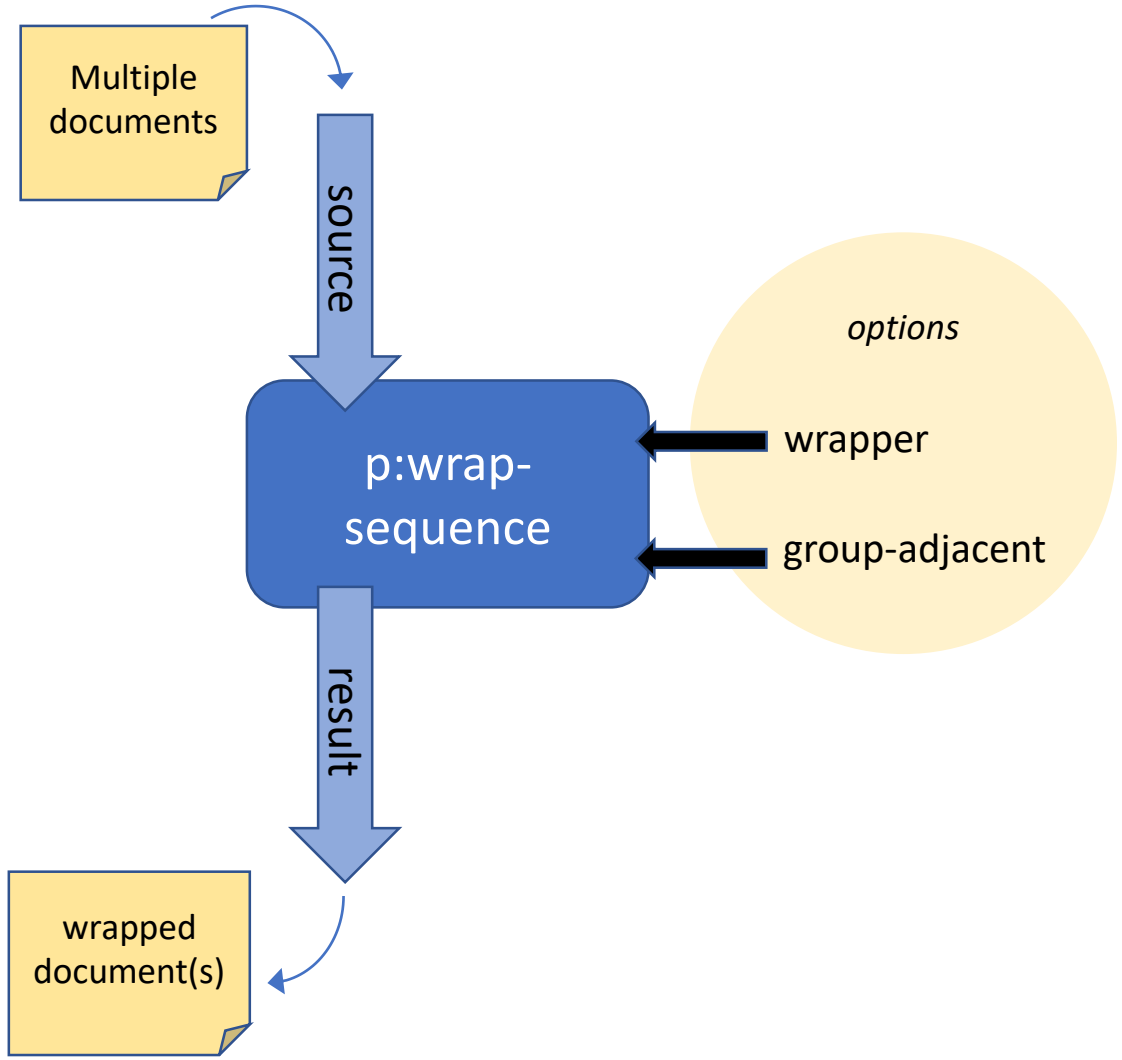
Make the output port accept a sequence

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0">

  <p:input port="source"/>
  <p:output port="result" sequence="true"/>

  <p:for-each>
    <p:with-input select="//doc"/>

    <p:store href="{/*/@filename}"/>
  </p:for-each>

</p:declare-step>
```

You can also define what document types a port will accept

The p:wrap-sequence step

See: http://spec.xproc.org/master/head/steps/#c.wrap-sequence



With the group-adjacent option you can group incoming documents based on an XPath expression. We're not going to try that now

Hands-on: Use p:for-each to split a document 2

- Open a command window in **`exercises/10-for-each-2/`**
- Add a **`p:wrap-sequence`** step after the **`p:for-each`** and use this to wrap the results in a **`<result>`** element
- Try it with **`run.bat`** or **`run.sh`**

Use p:for-each to split a document 2 – Solution

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0">

  <p:input port="source"/>
  <p:output port="result"/>

  <p:for-each>
    <p:with-input select="//doc"/>

    <p:store href="{/*/@filename}"/>
  </p:for-each>

  <p:wrap-sequence wrapper="results"/>

</p:declare-step>
```

Wrap the results in a <result> element

Goodbye and thank the fish!

- Specification in the making: http://spec.xproc.org/
  - Norman Walsh, Achim Berndzen, Gerrit Imsieke, Erik Siegel
- We hope to finish before end 2019
  - Next meeting of the XProc 3.0 working group: November 9-10, Cologne
- On its way, beside the specification:
  - Two processor implementations
    - https://www.xml-project.com/ (MorganaXProc)
    - https://xmlcalabash.com/ (XML Calabash)
  - A Programmer's Reference Guide
    - To be published by XML Press beginning 2020
- Your guide today: Erik Siegel – erik@xatapult.nl