Introduction to XProc 3.0
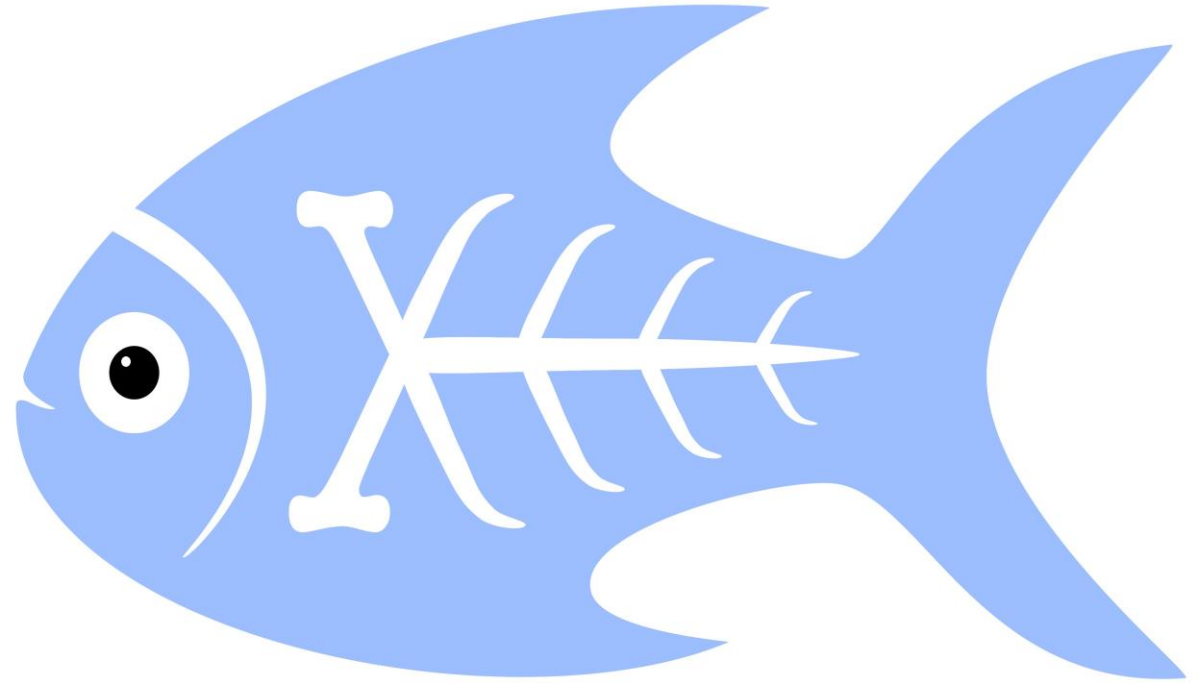
Declarative Amsterdam
November 4 and 5, 2021

While waiting, maybe you can do some preparations?
Go to **https://da.xatapult.com** for instructions!

Who Am I?

- Erik Siegel

- Content Engineer and XML specialist

- One-man company: Xatapult
  - Groningen, The Netherlands

- Member of the XProc 3.0 editing committee

- Author of the XProc 3.0 Programmer Reference

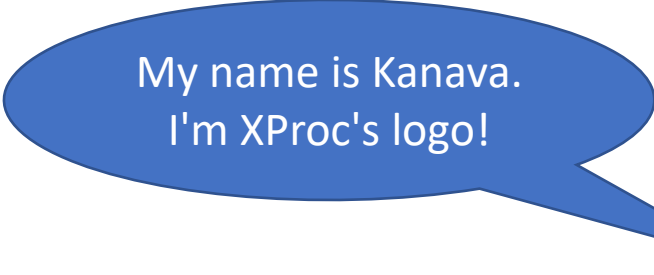- Contact:

  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 around since 2010 (two processor implementations to run your pipelines)

- Specification of 3.0 in "last call" status

- One working processor (MorganaXProc-IIIse)
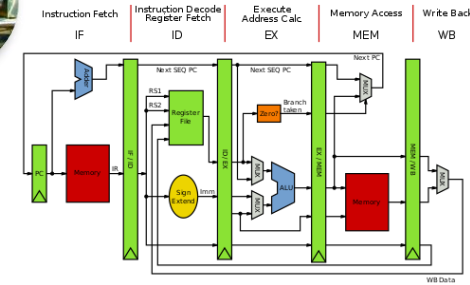
- One under way (XML Calabash 3), almost there

My name is Kanava.
I'm XProc's logo!

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
- Compose small tools into something bigger, pipelines…
- Very natural choice for document processing
- XProc beats the alternatives

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

Tutorial site: https://da.xatapult.com

Hands-on: Installation and pre-flight check

- Done all preparations?
  - Download/cloned the GitHub repository for this tutorial?
    https://github.com/xatapult/da-2021-xproc
  - Java working on your machine?
  - Download and unpacked MorganaXProc-IIIse?
    https://sourceforge.net/projects/morganaxproc-iiise/
  - Added Morgana's main directory to the system's path?

- Go to where you cloned/downloaded the tutorial's GitHub repository

- Open a command window in **exercises/01-hello-xproc/**

- Command: **morgana pipeline.xpl**

```
===============================
MorganaXProc-IIIse 0.9.6-beta
Copyright 2011-2021 by <xml-project /> Achim Berndzen
===============================


<?xml version="1.0" encoding="UTF-8"?>
<hello-xproc timestamp="2021-09-23T11:14:27.21+02:00"/>
```

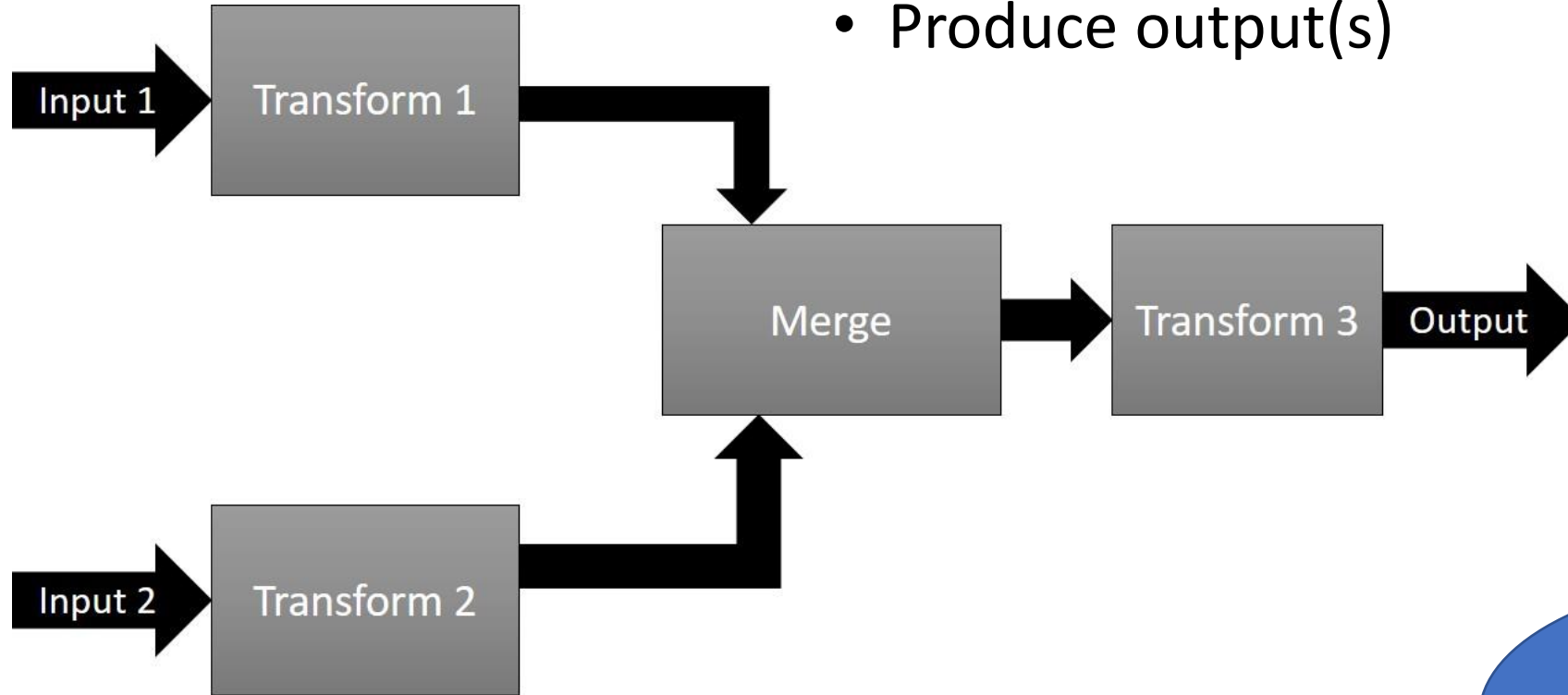If it works you've just run your first XProc pipeline!

XProc fundamentals

You need to understand this!

Pipelines, steps

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

Input 1 → Transform 1 →
Input 2 → Transform 2 →
Merge → Transform 3 → Output

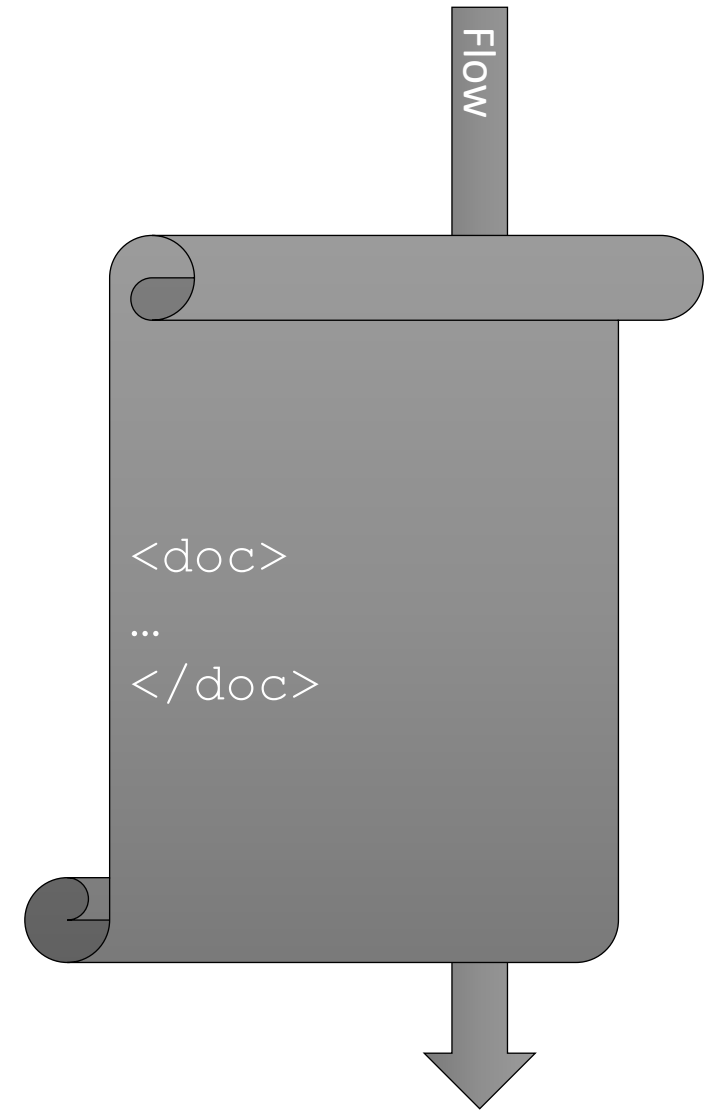Documents can be of any type, not just XML!

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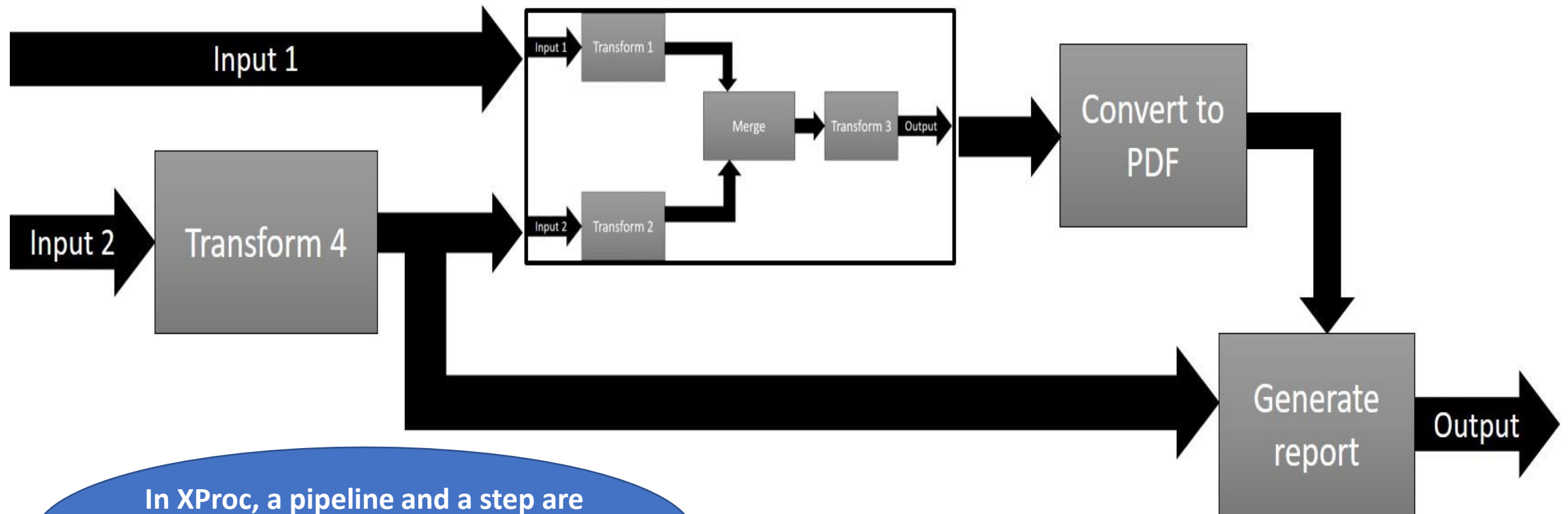
In XProc, a pipeline and a step are essentially the same. The terms can be used interchangeably!

Supported document types

- XML

- HTML

- JSON

- Text

- Binary/other
  - For instance: zip

Flow

```
<doc>
…
</doc>
```

Steps/pipelines, ports, options

document

source

p:add-attribute

result

Transformed
document

options:

match
attribute-name
attribute-value

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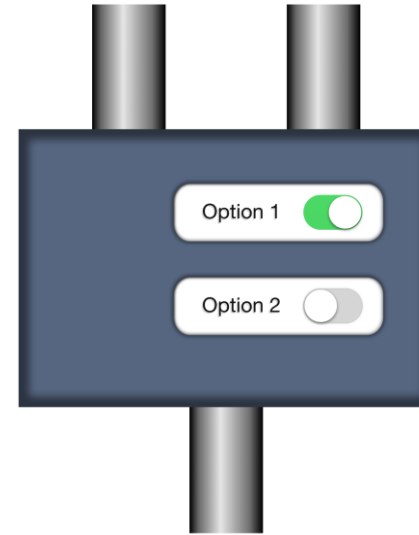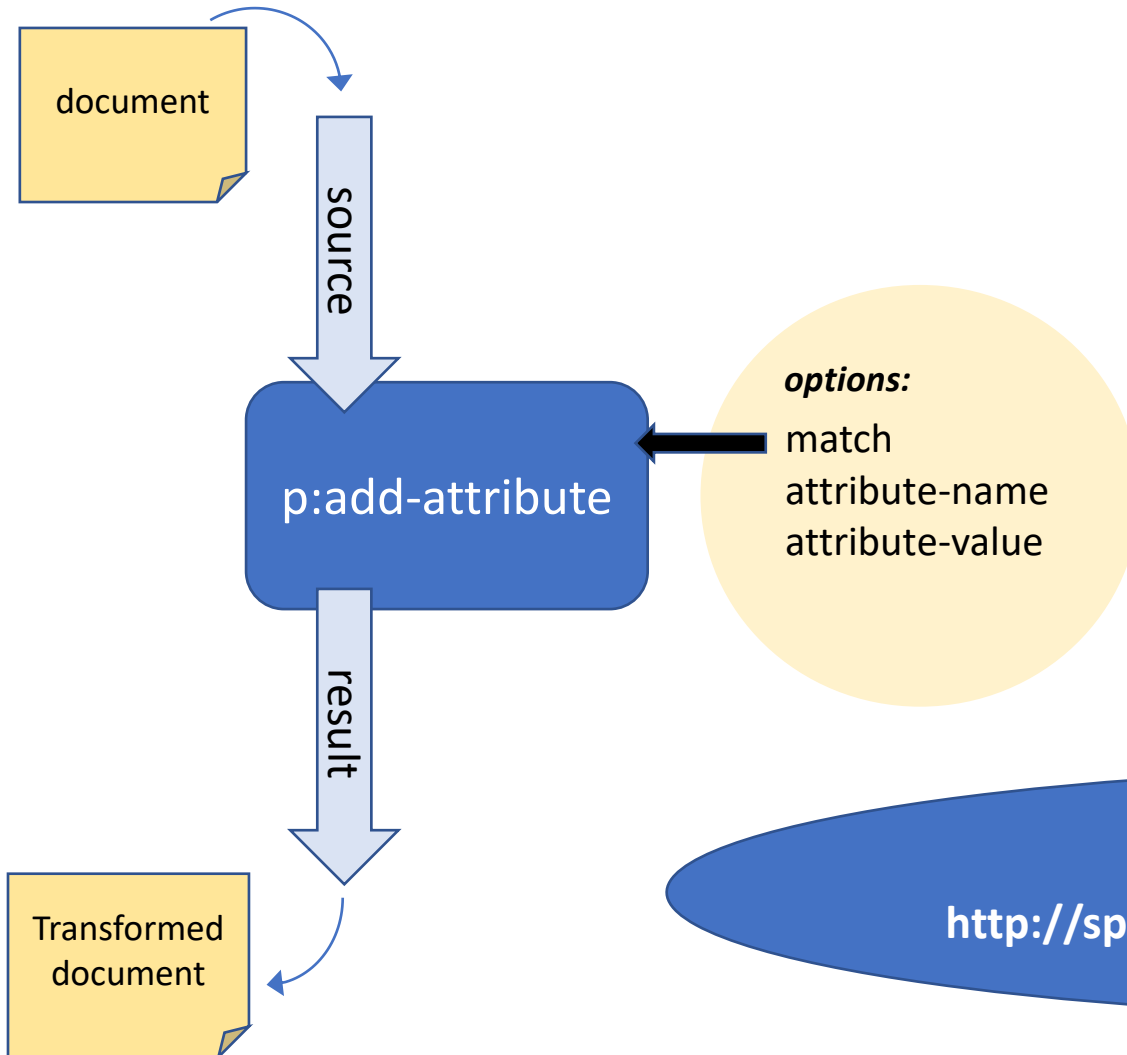
The step libraries

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

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

There are over 45 standard steps!

Step/pipeline that adds an attribute to the 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">
```

Input port declaration

```
    <p:input port="source"/>
```

Output port declaration

```
    <p:output port="result"/>
```

A step invocation

You can specify option values as attributes on the step invocation

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

</p:declare-step>
```

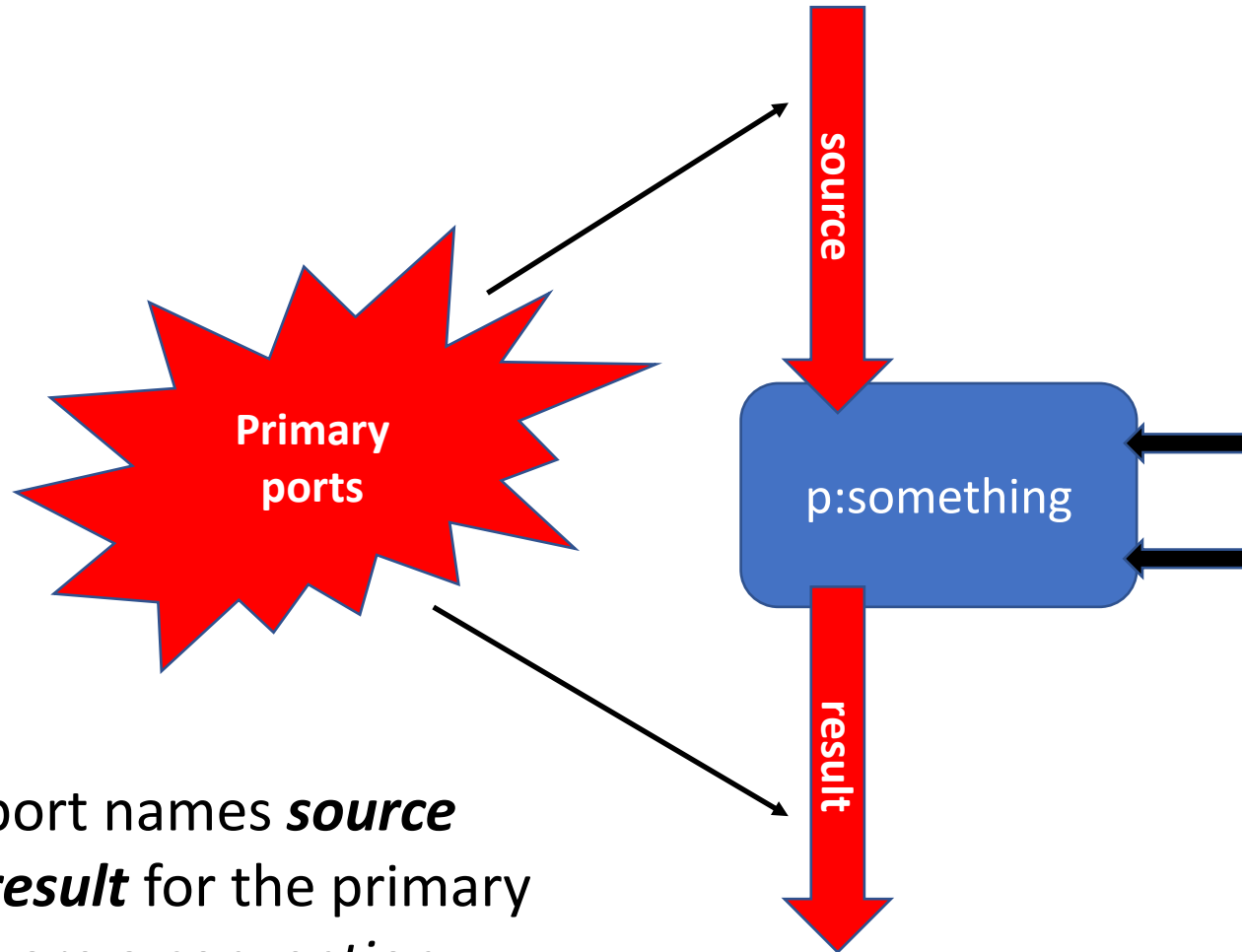To compute stuff use Attribute-Value-Templates (AVTs), just like in XSLT.
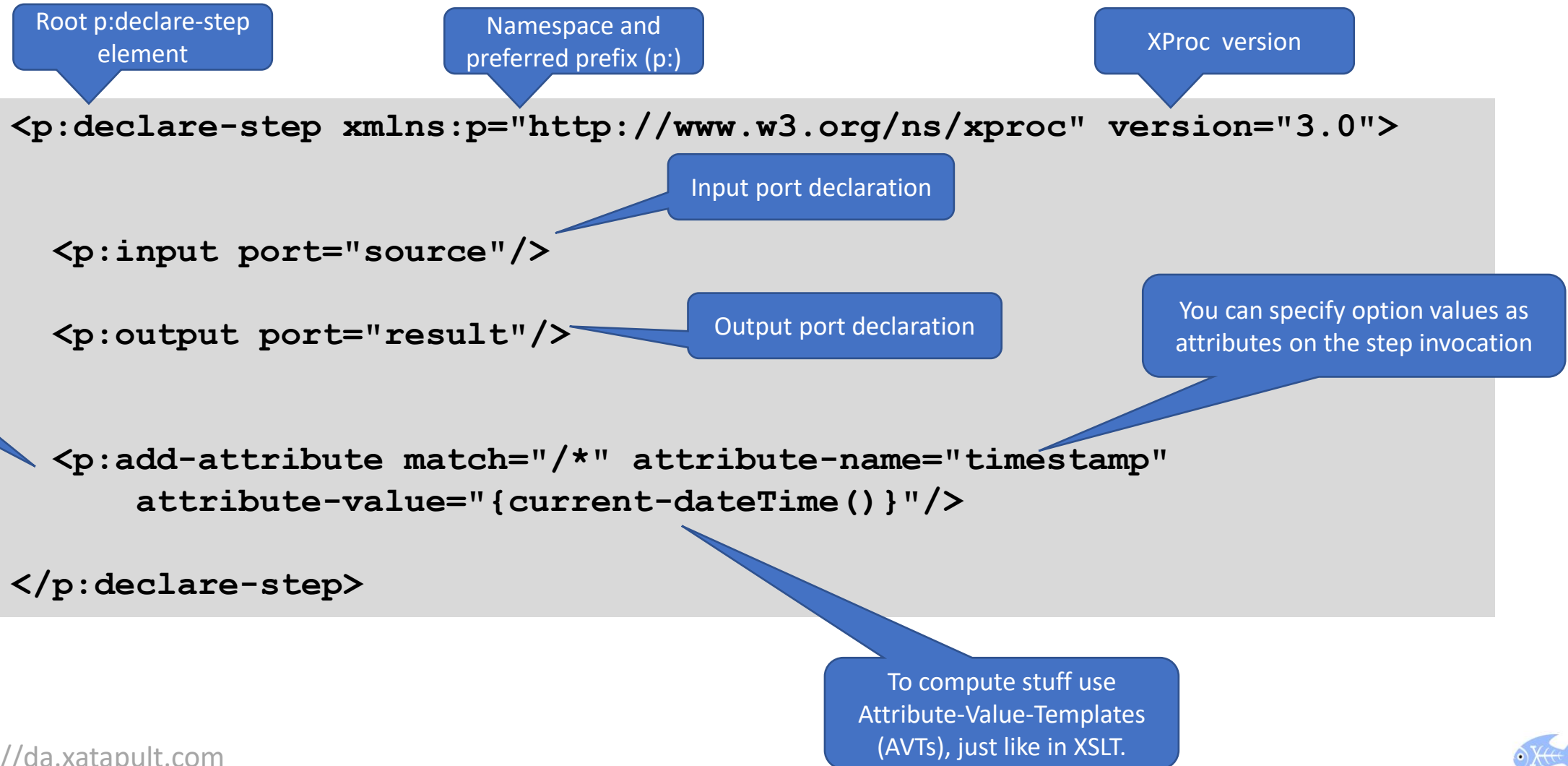
👉 Hands-on: Try it out

- Open a command window in **`exercises/02-add-attribute/`**

- Command: **`morgana pipeline.xpl -input:source=input.xml`**
- Try it!

Whow, that's a boring thing to do! I encourage you to experiment a bit …

Primary ports
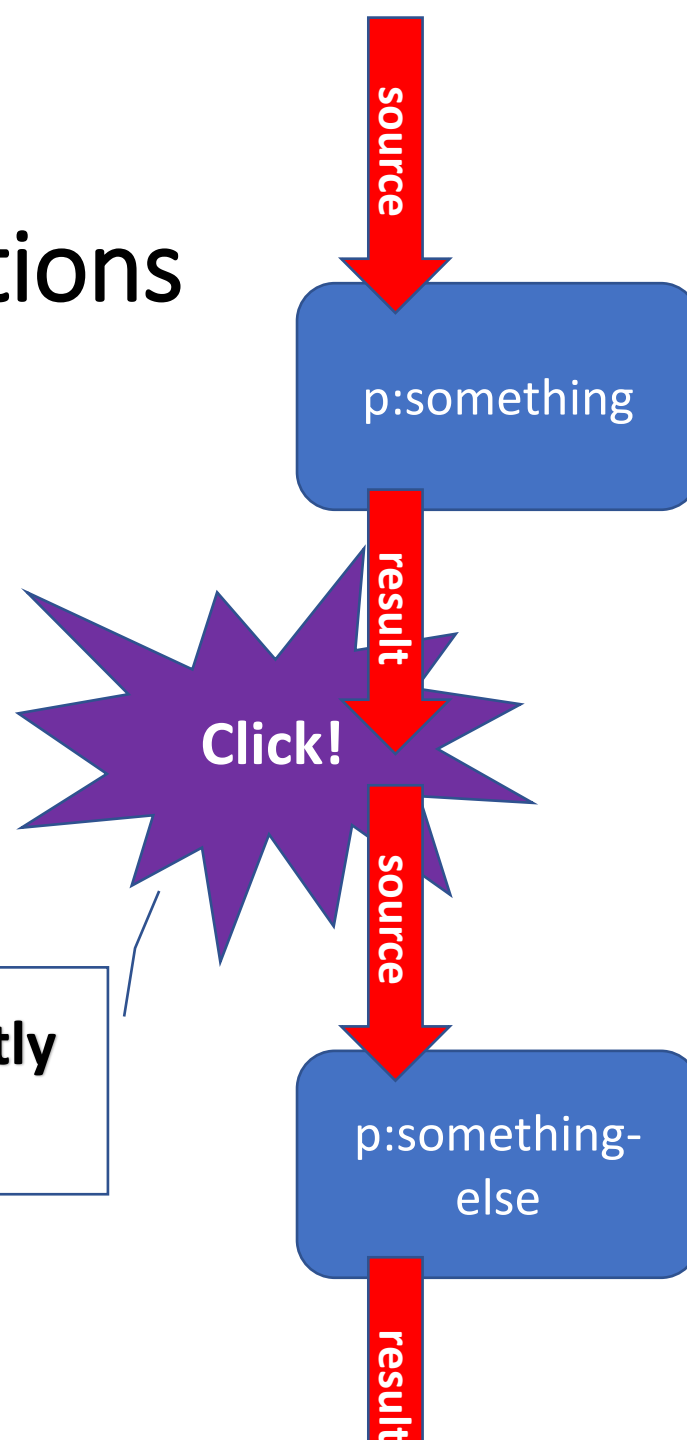

Primary ports

source

p:something

result

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

Not all ports are created equal...

Primary ports, implicit connections

source

p:something

result

Click!

source

p:something-else

result

```
<p:something …>
  …
</p:something>

<p:something-else …>
  …
</p:something-else>
```
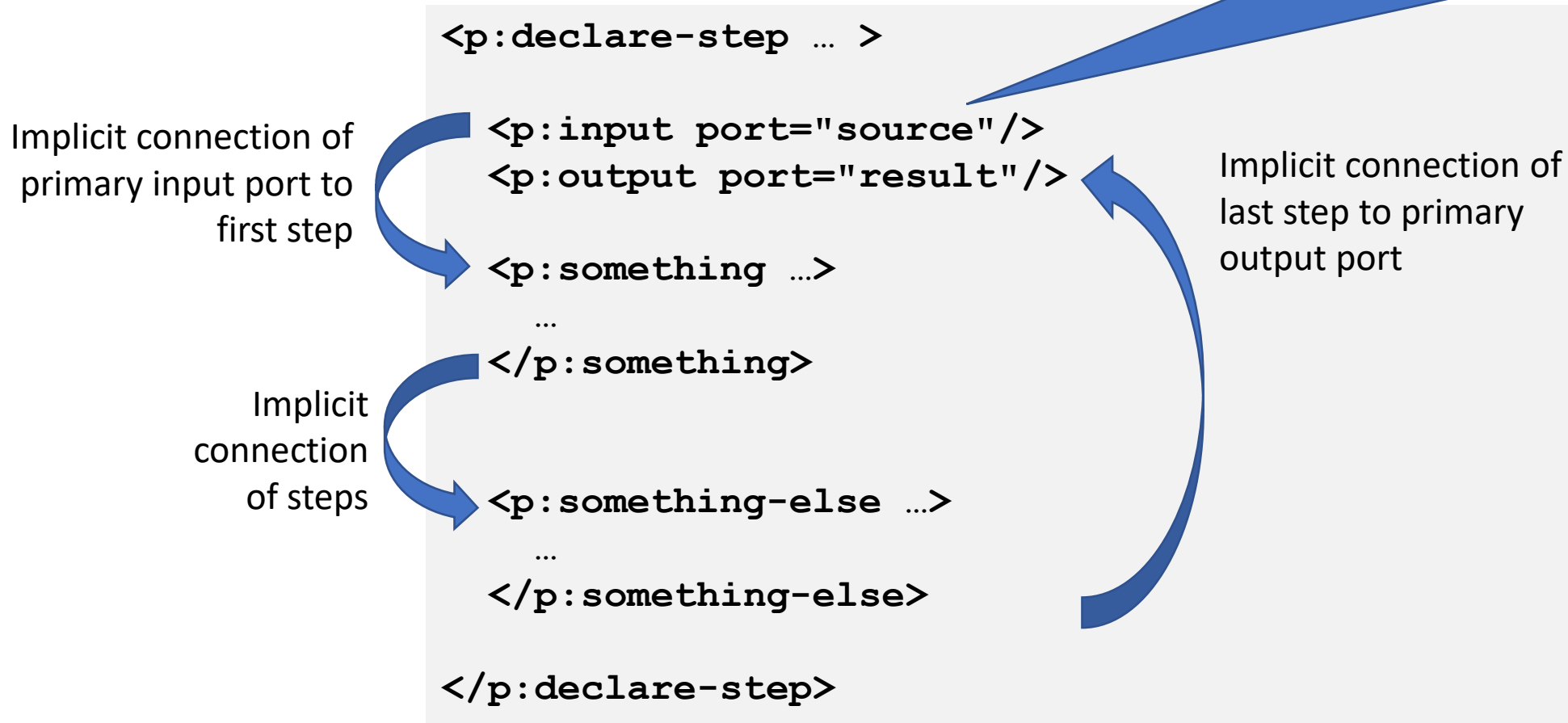
Click!

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.

```
<p:declare-step … >

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

    <p:something …>
      …
    </p:something>


    <p:something-else …>
      …
    </p:something-else>

</p:declare-step>
```

Implicit connection of primary input port to first step

Implicit connection of last step to primary output port

Implicit connection of steps

👉 Hands-on: Add a second attribute

- Open a command window in **`exercises/02-add-attribute/`**

- Change the pipeline and add a *second* **`p:add-attribute`** step that adds another attribute to the root element (or somewhere else).

- Command: **`morgana pipeline.xpl -input:source=input.xml`**

- Try it!

Easy….

Solution: Add a second attribute

```
<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>
```
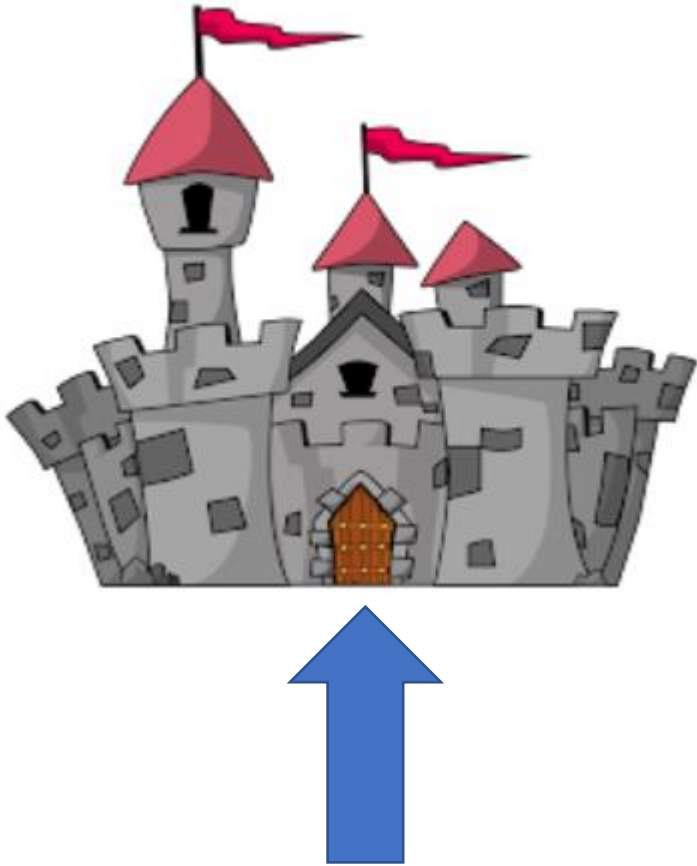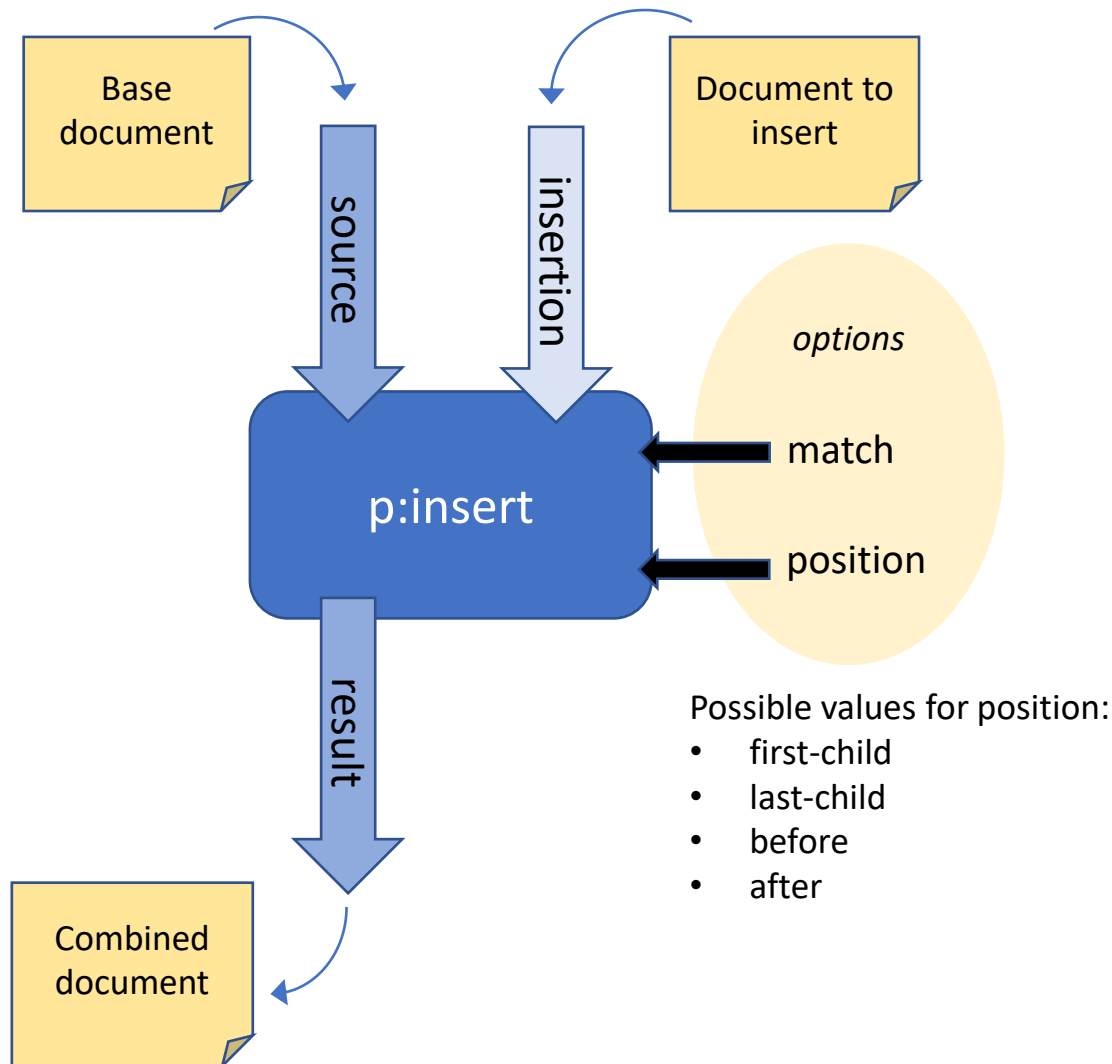
Ports and explicit connections

1. To an inline document
2. To an external document
3. To a port in the pipeline

The p:insert step

Base document

Document to insert

source

insertion

options

→ match

p:insert

→ position

result

Possible values for position:
- first-child
- last-child
- before
- after

Combined document

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

Connect a port to an inline document

Explicitly connect something to a port using p:with-input

```
<p:somestep ...>

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

  ...

</p:somestep>
```

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 with p:insert

- Open a command window in **exercises/03-connect-inline/**
- Finish the pipeline so it adds a
  <location>Amsterdam 2021</location>
  element *after* the **<presenter>** element
  - Use the **p:insert** step with an inline document
  - Options: **match="/*" position="last-child"**
  - Connect the inline document to the **insertion** port using **<p:with-input port="…">**
- Compute the current year using a **{…}** construction
  - XPath cheat: `year-from-date(current-date())`

- Command: **morgana pipeline.xpl -input:source=input.xml**
- Try it!

Now you're on your own writing XProc, scary…

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">
      <location>Amsterdam {year-from-date(current-date())}</location>
    </p:with-input>
  </p:insert>

</p:declare-step>
```

Connect a port to an external document

```
<p:somestep ...>

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

  ...

</p:somestep>
```

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 with p:insert

- Open a command window in **exercises/03-connect-inline/**
- Finish the pipeline so it adds the contents of **insert.xml** *after* the **<presenter>** element
  - Use the **p:insert** step with an href attribute
  - Options: **match="/*" position="last-child"**

- Command: **morgana pipeline.xpl -input:source=input.xml**
- Try 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" href="insert.xml"/>
  </p:insert>


</p:declare-step>
```
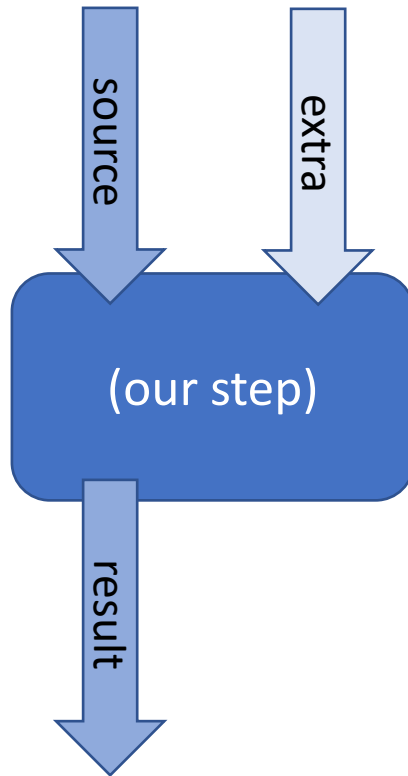
Add another port to our step and connect p:insert to it

source

extra

(our step)

result

The source and result port are primary, the extra port is not...

Add an additional input port to our step

We have more than two ports now: make explicit which ones are primary and which ones are not

Add another input port to our step using <p:input>

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

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

  <p:input primary="false" port="extra"/>


  ...
</p:declare-step>
```

Connect a port to another port in the same pipeline

Name the step/pipeline

Use the pipe attribute with "portname@stepname"

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

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

  <p:input primary="false" port="extra"/>

  <p:insert match="/*" position="last-child">
    <p:with-input port="insertion" pipe="extra@my-pipeline"/>
  </p:insert>

</p:declare-step>
```

Hands-on: Add an external document with p:insert

- Open a command window in **`exercises/05-connect-internal/`**

- Command: **`morgana pipeline.xpl -input:source=input.xml -input:extra=insert.xml`**

- Try it!

- Change this pipeline so it now inserts its primary input document in itself, directly after the inserted **`<location>`** element…

```
<tutorial>
   <name>Introduction to XProc 3.0</name>
   <presenter>Erik Siegel</presenter>
   <location>Amsterdam</location>
   <tutorial>
      <name>Introduction to XProc 3.0</name>
      <presenter>Erik Siegel</presenter>
   </tutorial>
</tutorial>
```

Not very useful but nonetheless insightful

Insert external document - solution

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

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

  <p:input primary="false" port="extra"/>

  <p:insert match="/*" position="last-child">
    <p:with-input port="insertion" pipe="extra@my-pipeline"
  </p:insert>

  <p:insert match="/*" position="last-child">
    <p:with-input port="insertion" pipe="source@my-pipeline"/>
  </p:insert>

</p:declare-step>
```
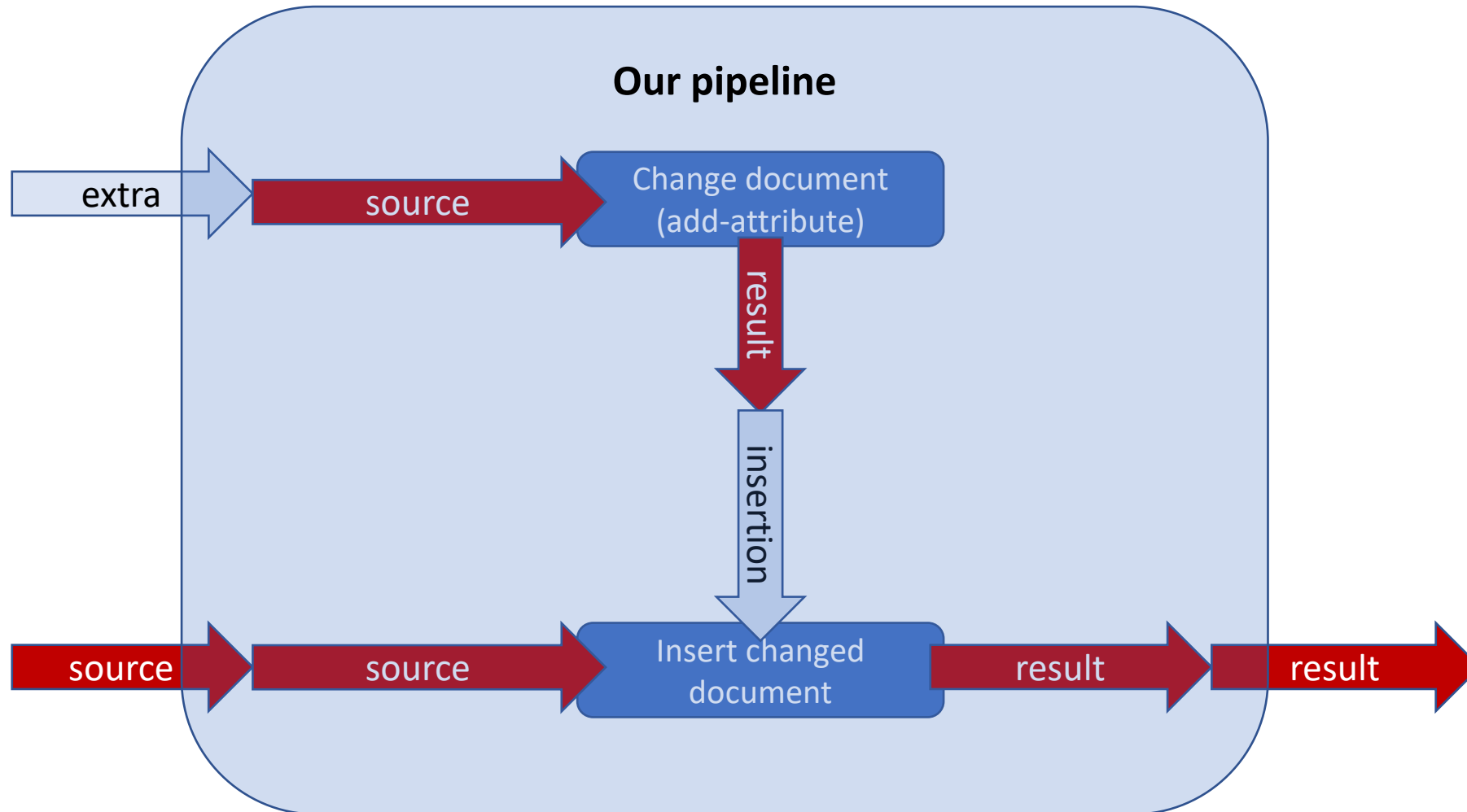
Re-read the document from the step's source port

Hands-on: Change a document and insert it into the primary document

Our pipeline

extra → source → Change document (add-attribute)

result → insertion → Insert changed document

source → source → Insert changed document → result → result

Hands-on: Change a document and insert it into the primary document

- Open a command window in `exercises/05b-connect-internal/`

- Change this pipeline so the extra document is changed first before the insert

- Command: `morgana pipeline.xpl -input:source=input.xml -input:extra=insert.xml`

- Try it!
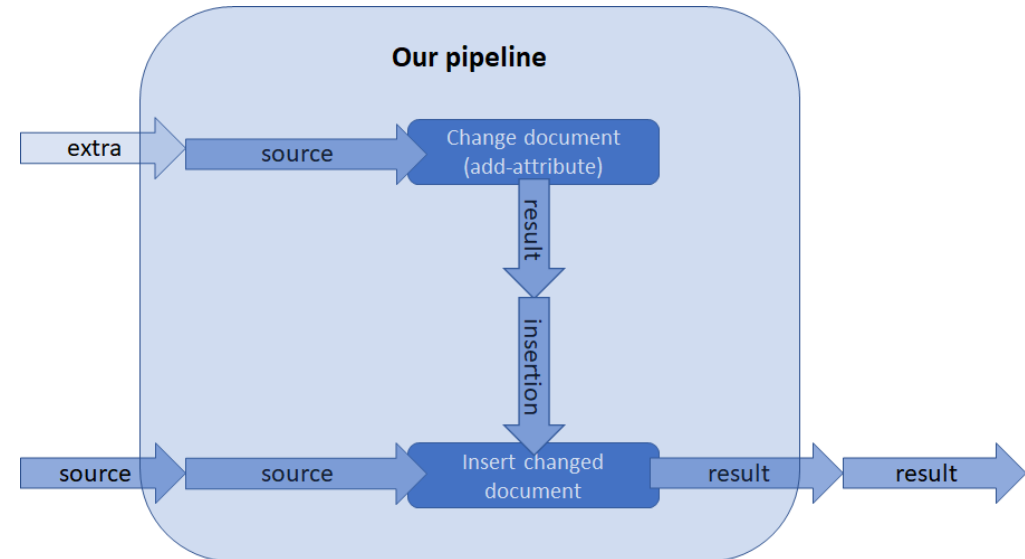
Change a document and insert it into the primary document - solution

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

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

  <p:input primary="false" port="extra"/>

  <p:add-attribute match="/*" attribute-name="extra" attribute-value="true"
    name="change-extra-document">
    <p:with-input port="source" pipe="extra@my-pipeline"/>
  </p:add-attribute>

  <p:insert match="/*" position="last-child">
    <p:with-input port="source" pipe="source@my-pipeline"/>
    <p:with-input port="insertion" pipe="result@change-extra-document"/>
  </p:insert>

</p:declare-step>
```
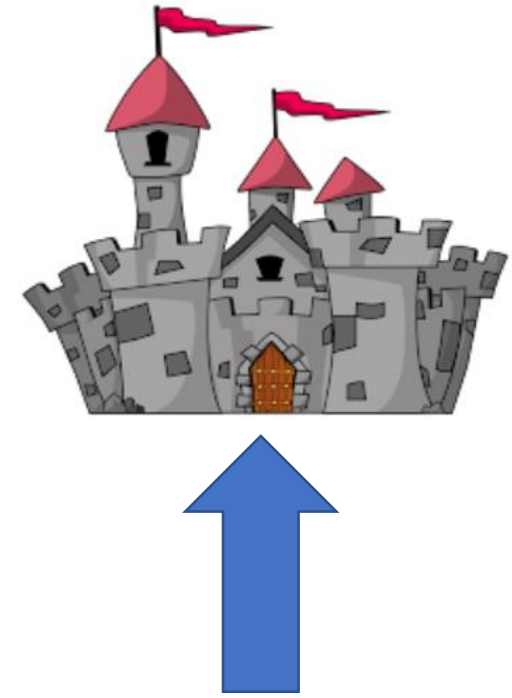
Explicit connection because p:insert is no longer the first step!

Ports and implicit connections

1. To an inline document
   - Add the XML inside the `<p:with-input>`
2. To an external document
   - Use the `href` attribute on `<p:with-input>`
3. To a port in the pipeline
   - Name the steps you want to read from and use the `pipe` attribute on `<p:with-input>`

Options and variables

1. Add an option to your pipeline
2. Variables
3. Other ways to set options

Your own options

- We've seen that built-in steps can have options

- What if you want to add an option to your own step?

Declare the option in the prolog of your step

Optionally add a default value (XPath expression!)

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

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

<p:option name="username" select="'erik'"/>

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

</p:declare-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.

👉 Hands-on: Add options

- Open a command window in **exercises/06-add-option/**

- Command: **morgana pipeline.xpl -input:source=input.xml -option:username=*(your-name)***

- Try it!

- Change this pipeline so the *name* of the username attribute can be changed also, using an option called **aname**

Add an option - solution

```
<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" select="'erik'"/>
  <p:option name="aname" select="'username'"/>

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

</p:declare-step>
```

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

Variables values from the documents flowing through

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


Read values from the **document** flowing through!

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

👉 Hands-on: Use a variable

- Open a command window in **`exercises/07-use-variable/`**

- Add a variable that catches the value of the **`<presenter>`** element
  - **`<p:variable name="…" select="…"/>`**

- Add this value as an attribute to the root

- Command: **`morgana pipeline.xpl -input:source=input.xml`**

- Try it!

Use a variable - solution

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

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

  <p:variable name="presenter" select="//presenter[1]"/>

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

</p:declare-step>
```

Get some value from the document flowing through

Reference the variable using the $... notation

Alternative...

```
<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="presenter"
     attribute-value="{//presenter[1]}"/>

</p:declare-step>
```

How to set an option?

1. As attribute on a step's invocation:
   `<p:add-attribute match="…" …/>`
   - Works for data types that can be cast from a string (strings, booleans, integers doubles, etc.)
   - If the option's data type is a *map* you can use a map constructor:
     `parameters="map{ 'par1':'value-for-par1' }"`

2. Using the `<p:with-option>` child element… but be careful!

Using <p:with-option>

```
<p:add-attribute match="/*" attribute-name="x" attribute-value="{1 + 2}"/>
```

The match expression should be interpreted *by the step* (and not by the *pipeline*), so it must be passed as a string!

```
<p:add-attribute>
  <p:with-option name="match" select="'/*'"/>
  <p:with-option name="attribute-name" select="'x'"/>
  <p:with-option name="attribute-value" select="1 + 2"/>
</p:add-attribute>
```

The select attribute(s) contain **XPath** expressions

👉 Hands-on: Use p:with-option

- Open a command window in **`exercises/08-use-with-option/`**

- Replace the attributes on the **`<p:add-attribute>`** by **`<p:with-option>`** elements

- Command: **`morgana pipeline.xpl -input:source=input.xml`**

- Try it!

- Try what happens when you forget to make the **`match`** option a string…

Use p:with-option - 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="string(current-dateTime())"/>
  </p:add-attribute>

</p:declare-step>
```

When to use what for setting options?

- Style…
  - Attributes are easier (?)
  - `<p:with-option>` is more explicit (?)

- When the datatype of an option cannot be cast from a string, you *must* use `<p:with-option>`
  - For instance, when the option requires a *sequence* as value:
    `<p:with-option name="…" select="(…, …, …)"/>`

Options and variables

1. Add an option to your pipeline
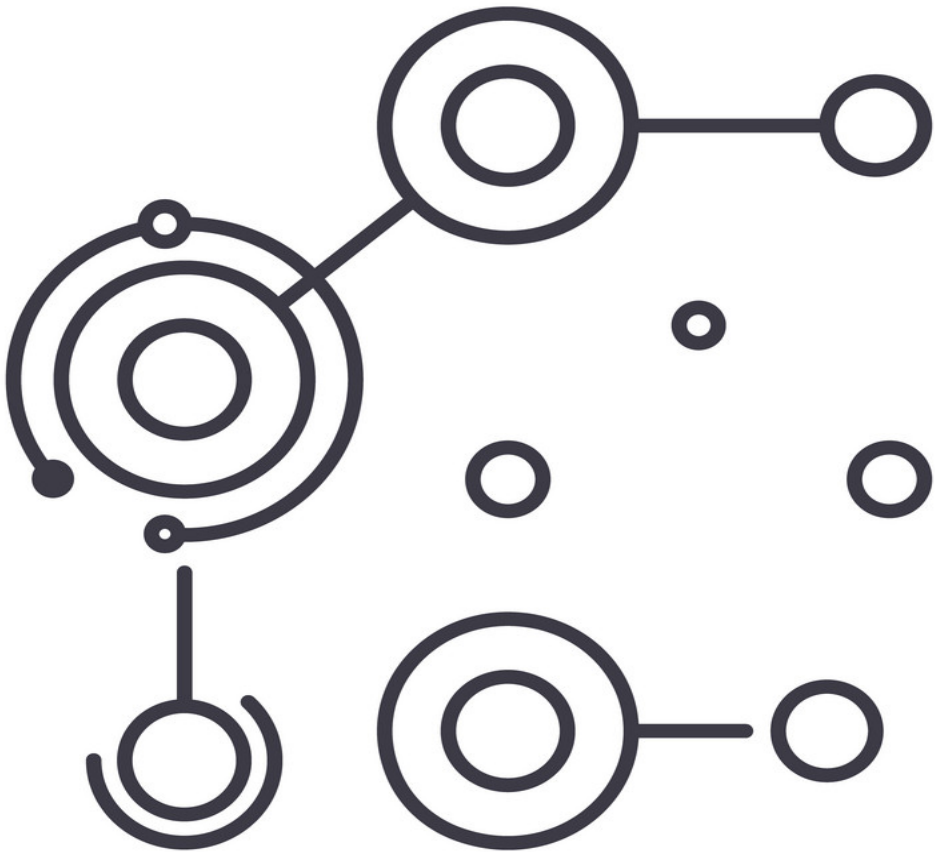   - **`<p:option>`** in the prolog
   - Reference option as a variable
2. Variables
   - Declare them anywhere using **`<p:variable>`**
3. Other ways to set options
   - Either attribute or **`<p:with-option>`**

Compound steps

1. What are compound steps?
2. What steps do we have?
3. Example of using the p:for-each step

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 (no else)
- **`p:viewport`**: Work on only a part of a document
- **`p:try / p:cat`**ch: Error catching and handling
- **`p:group`**: Grouping of instructions

Regrettably, there is no time to look at them all…

Use p:if to make a decision

```
<p:if test="…">


</p:if>
```

The code here only executes when the test expression is true

When the expression is false, the input simply "falls through", without change

👉 Hands-on: Use p:if to make a decision

- Open a command window in **exercises/09-use-if/**

- The input document **input.xml** has a **status** attribute

- Write the code for **pipeline.xpl** so that:
  If **/*/@status** has the value **error**, add an additional attribute to the root:
  special-handling="true"

- Try it!

- Command: **morgana pipeline.xpl -input:source=input.xml**

Use p:if to make decision – Solution

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

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

  <p:if test="/*/@status eq 'error'">
    <p:add-attribute match="/*" attribute-name="special-handling" attribute-value="true"/>
  </p:if>

</p:declare-step>
```

Multiple decisions? Use p:choose

```
<p:choose>
  <p:when test="…">

    …
  </p:when>
  <p:when test="…">

    …
  </p:when>
  <p:otherwise>

    …
  <p:otherwise>
</p:when>
```

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

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

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/**

- Command: **morgana pipeline.xpl -input:source=input.xml**

- Try it!

- It does not run… why? What does the error message tell you?

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

- What is the result of this pipeline?

So how many documents flow out of this step now?

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

Make the output port accept a sequence

```
<p:declare-step xmlns:p="http://www.w3.org/ns/xpro    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>
```
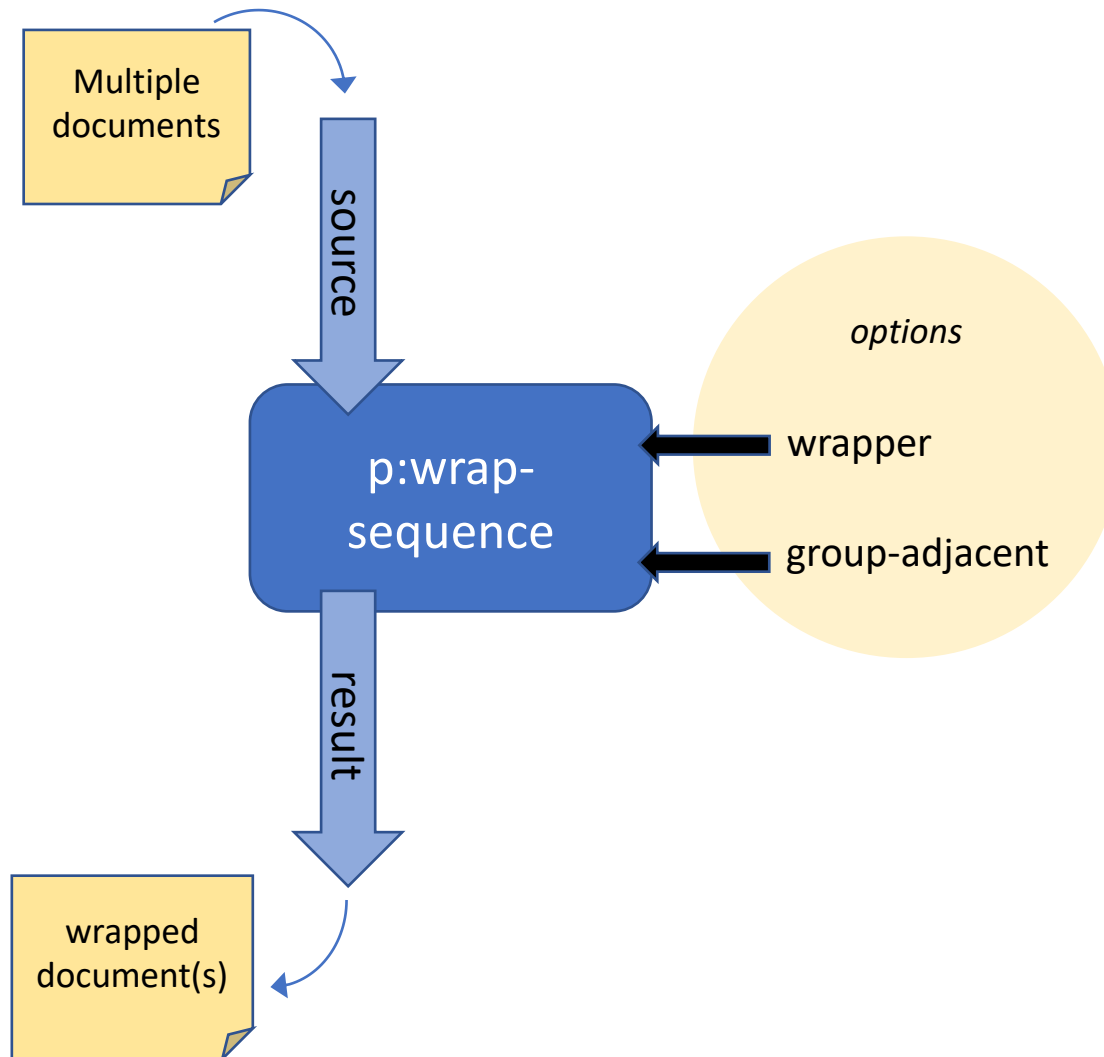
What flows out of the p:for-each are *a sequence* of documents!

You can also define what document types a port will accept

The p:wrap-sequence step

Multiple documents

source

options

p:wrap-sequence

wrapper

group-adjacent

result

wrapped document(s)

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

- (Re)Open a command window in **`exercises/10-for-each/`**
- Add a **`p:wrap-sequence`** step to wrap the sequence of result documents in a **`<result>`** element
- Command: **`morgana pipeline.xpl -input:source=input.xml`**
- Try it!

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="result"/>

</p:declare-step>
```

Wrap the results in a
<result> element

Use p:viewport to change parts of a document - Input

```
<documents>
  <doc keep="true">
    <p>Hi there!</p>
  </doc>
  <doc keep="false">
    <p>Nonsense!</p>
  </doc>
  <doc keep="true">
    <p>Hi there again!</p>
  </doc>
</documents>
```
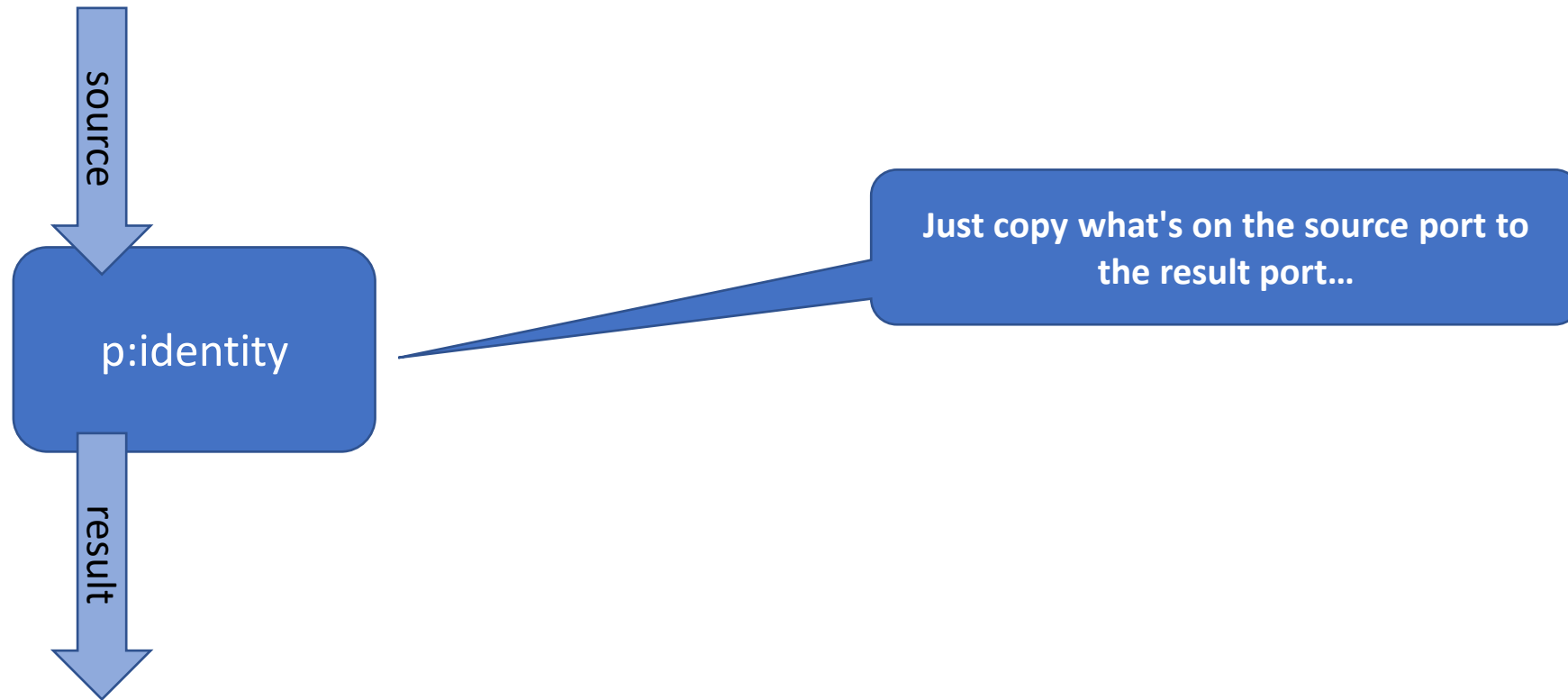
Replace every doc with @keep="false" with <DELETED/>

p:viewport

```
<p:viewport match="…">



</p:viewport>
```

Replace the matched nodes with you produce in here

The p:identity step

source

p:identity

Just copy what's on the source port to the result port...

result

Hands-on: Use p:viewport to change parts of a document

- Open a command window in **exercises/11-viewport/**

- Finish the code so every **<doc>** with @**keep="false"** is replaced by a **<DELETED/>** element

- Try it!

- Command: **morgana pipeline.xpl -input:source=input.xml**

Use p:viewport to change parts of a 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:viewport match="doc[not(xs:boolean(@keep))]">
    <p:identity>
      <p:with-input port="source">
        <DELETED/>
      </p:with-input>
    </p:identity>
  </p:viewport>

</p:declare-step>
```

Use p:identity to create an element

Wrap up

- Fundamentals
  - Steps
  - Ports
  - Primary ports, implicit port connections
  - Explicit port connections
    - To an inline document, external document or some port elsewhere in the pipeline
  - Options and variables
    - Declare options and variables
    - Use attributes or `<p:with-option>`
  - Compound steps

But there is much more to explore!

Goodbye and thank the fish!

- Main site: https://xproc.org/

- Two processor implementations
  - https://www.xml-project.com/ (MorganaXProc)
  - https://xmlcalabash.com/ (XML Calabash, 3.0 work in progress)

- Programmer Reference: https://xmlpress.net/publications/xproc-3-0/

- Your guide today: Erik Siegel – erik@xatapult.nl

- Kanava stickers: FREE!
- Book: €25

Goodbye!
And remember, Kanava says:
XProc rocks...