APP – Architecture for Pipelined Processing

in XML Processing Model WG

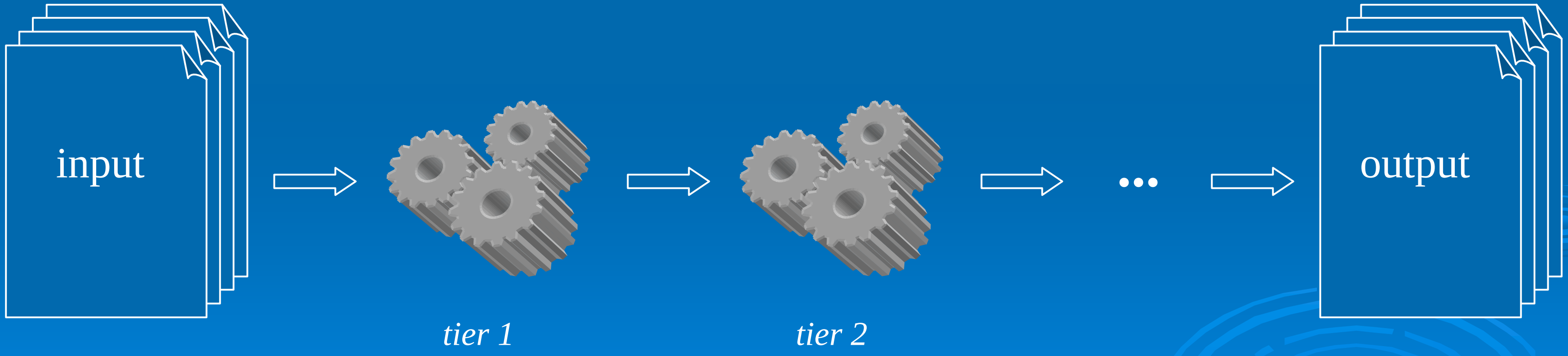
Rui Lopes
`rlopes@di.fc.ul.pt`

(Some) Requirements

- Complex XML processing applications
  - Rich Digital Books automated production
  - Offine processing of big contents (think Digital Libraries)
- Support developer and producer tasks
  - Separation of Concerns
  - There is always a need for some manual configuration

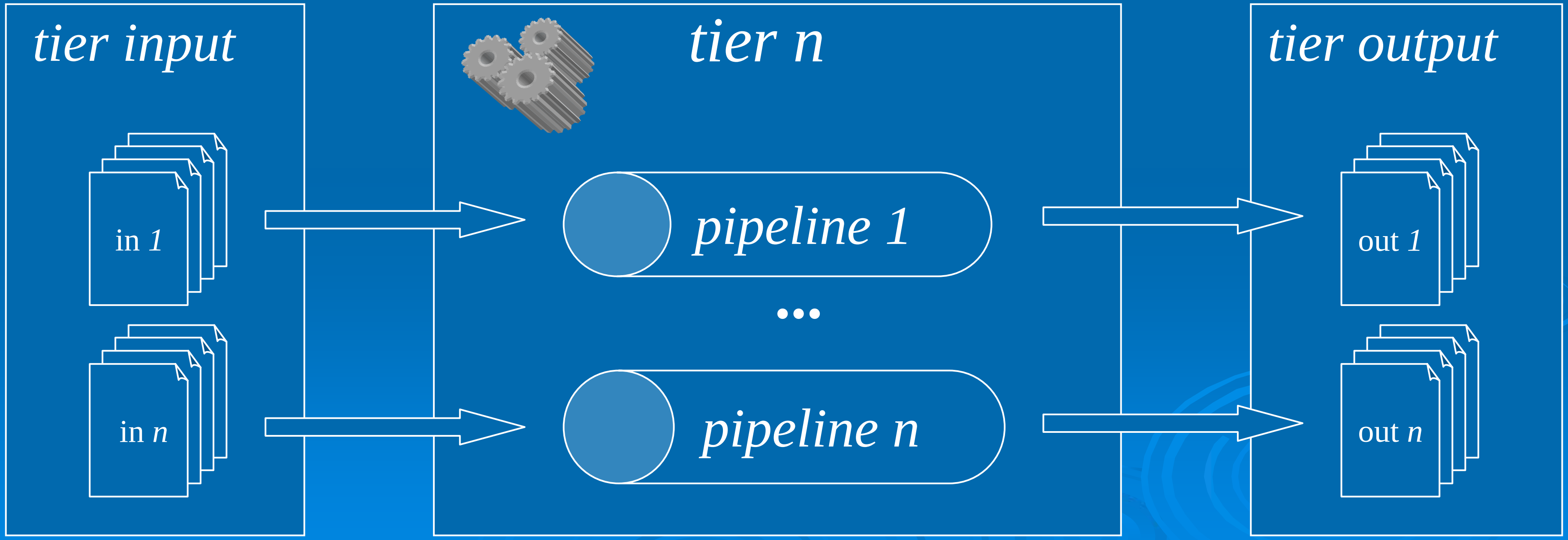
Processing Model I

- Given an input set, produce an output set
- **Project**: Sequence of (easily interchangeable) tiers



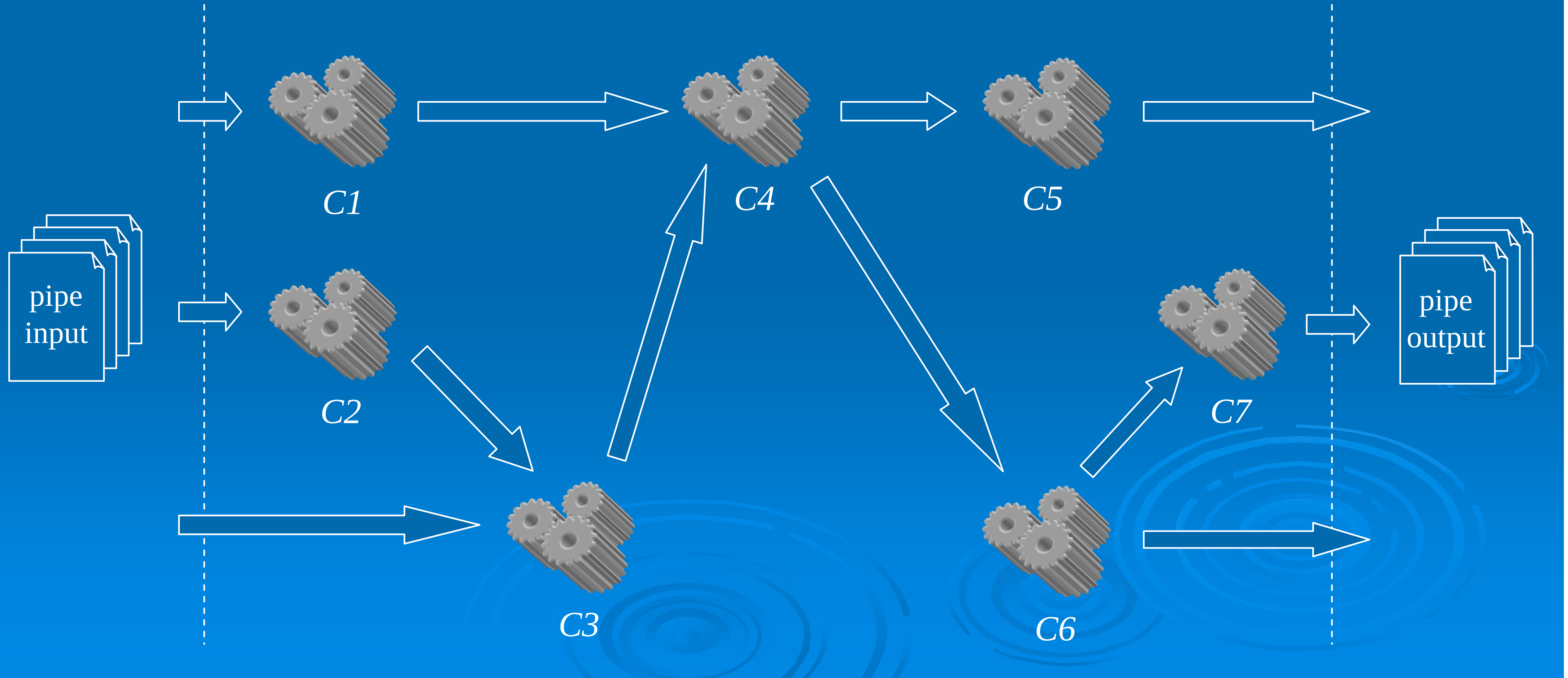
Processing Model II

- **Tier**: set of pipelines, working on disjoint subsets of the tier's input



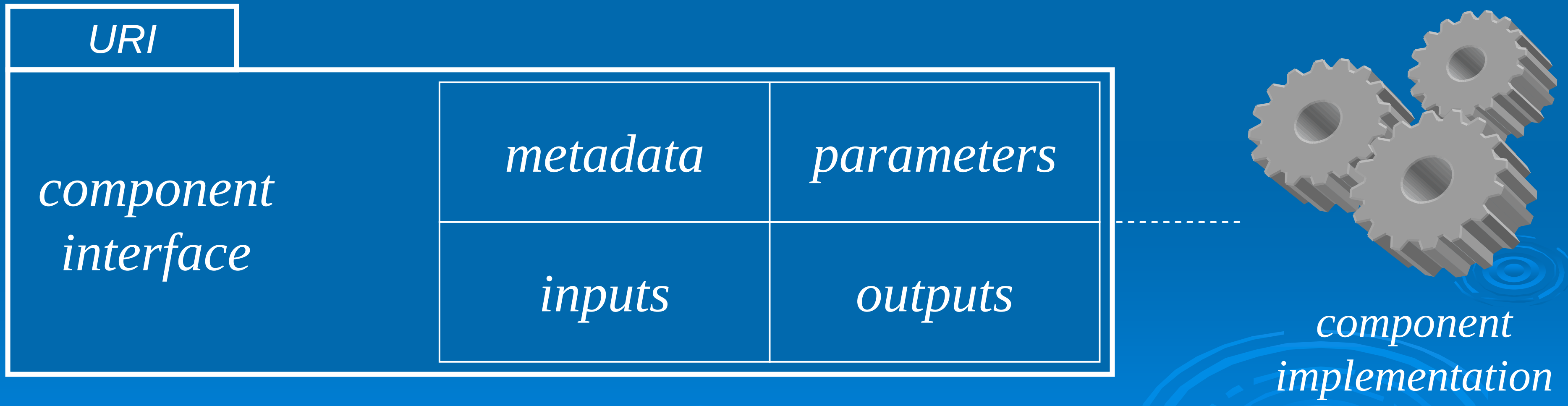
Processing Model III

- **Pipeline**: Acyclic digraph of processors



Processing Model IV

- **Processor**: interface *vs.* implementation *vs.* usage



Processing Language: Project

```
<rdf:RDF ...>
   <rdf:Description rdf:about="urn:app:project#metadata">
        ...
   </rdf:Description>

   <rdf:Description rdf:about="urn:app:project#stages">
        <rdf:Seq>
               <rdf:li rdf:resource="..." />
               ...
        </rdf:Seq>
   </rdf:Description>
</rdf:RDF>
```

Processing Language: Pipelines

```
<config ...>
  <pipeline>
     <component reg:ref="..." />
     <component reg:ref="..." />
     <component reg:ref="...">
          <param name="..." value="..." />
     </component>
     ...
  </pipeline>
  ...
</config>
```

Processing Language: Registry

```
<rdf:RDF ...>
  <rdf:Description rdf:about="urn:app:registry">
    <rdf:Bag>
      <rdf:li rdf:id="someid" reg:type="someproctype" rdf:resource="someres">
        <rdf:Description rdf:about="someid#metadata" />
        <rdf:Description rdf:about="someid#plugs">
          <plug:in>
            <rdf:Bag>
              <rdf:li rdf:resource="..." plug:default="yes|no" />
              ...
            </rdf:Bag>
          </plug:in>
          <plug:out ... />
        </rdf:Description>
        <rdf:Description rdf:about="someid#params">
          <plug:param name="..." use="required|optional" />
          ...
        </rdf:Description>
      </rdf:li>
      ...
    </rdf:Bag>
  </rdf:Description>
</rdf:RDF>
```

Discussion: Pros

- Separation of Concerns
  - Interchangeable components without touching the pipelines
- RDF based: tools already available
- (Rudimentary) composition
- Easily extensible for new processors (no need to redefine schema)
- Strong static verification: good for interactively design pipeline applications
- Implementation neutral: Ant+XSLT, Groovy

Discussion: Cons

- No iteration/test: doable (no conflit with the model), but just solves some manual configurations, not all
- RDF based, overly complex: "***APP on rails***" solves
- No support for 2nd generation resources
- No support for wildcards (e.g. agreggation, chunking)

Thoughts

- Having *n* levels of composition is good: **modularization**, **maintenance**, **services**
- Explicit mechanisms for composition better than plain XInclude (*function call vs. preprocessing*)
- Indirection level of multiple inputs requires some sort of mapping – Think batching