29. Composition of Stream-Based Tools (Data Exchange) and the Resulting Macromodels

Prof. Dr. Uwe Aßmann

Technische Universität Dresden

Institut für Software- und Multimediatechnik

http://st.inf.tu-dresden.de

Version 19-1.1, 06.01.20

1) Architecture of Stream-Based Software Factories
2) Extension of Stream-Based Tools
3) Stream-based Macromodels
4) Stream-based XML-Mashups
5) End
    1) Aspect-Oriented Extension
    2) EAI-Decomposition of Tools
    3) EAI-Based Composition of Tools

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Literatur

- Informatik Forum http://www.infforum.de/
- Structured Analysis Wiki http://yourdon.com/strucanalysis/wiki/index.php?title=Introduction
- Ed Yourdon. Just Enough Structured Analysis. Free pdf-book on:
  - http://www.yourdon.com/jesa/pdf/JESA_xtru.pdf
- De Marco, T.: Structured Analysis and System Specification; Yourdon Inc. 1978/1979. Siehe auch Vorlesung ST-2
- McMenamin, S., Palmer, J.: Strukturierte Systemanalyse; Hanser Verlag 1988
- Raasch, J.: Systementwicklung mit Strukturierten Methoden; Hanser Verlag (3.Aufl.) München 1993
- [Altinel07] Mehmet Altinel, Paul Brown, Susan Cline, Rajesh Kartha, Eric Louie, Volker Markl, Louis Mau, Yip-Hing Ng, David E. Simmen, and Ashutosh Singh. DAMIA - A data mashup fabric for intranet applications. In C. Koch, et.al., editors, VLDB, pages 1370-1373. ACM, 2007.

29.1 Architecture of Stream-Based Software Factories

An Integrated Development Environment is a Tool Suite with Data, Control, Process, and UI-Integration.

UNIX Programmers Workbench (PWB): Stream- and File-Based

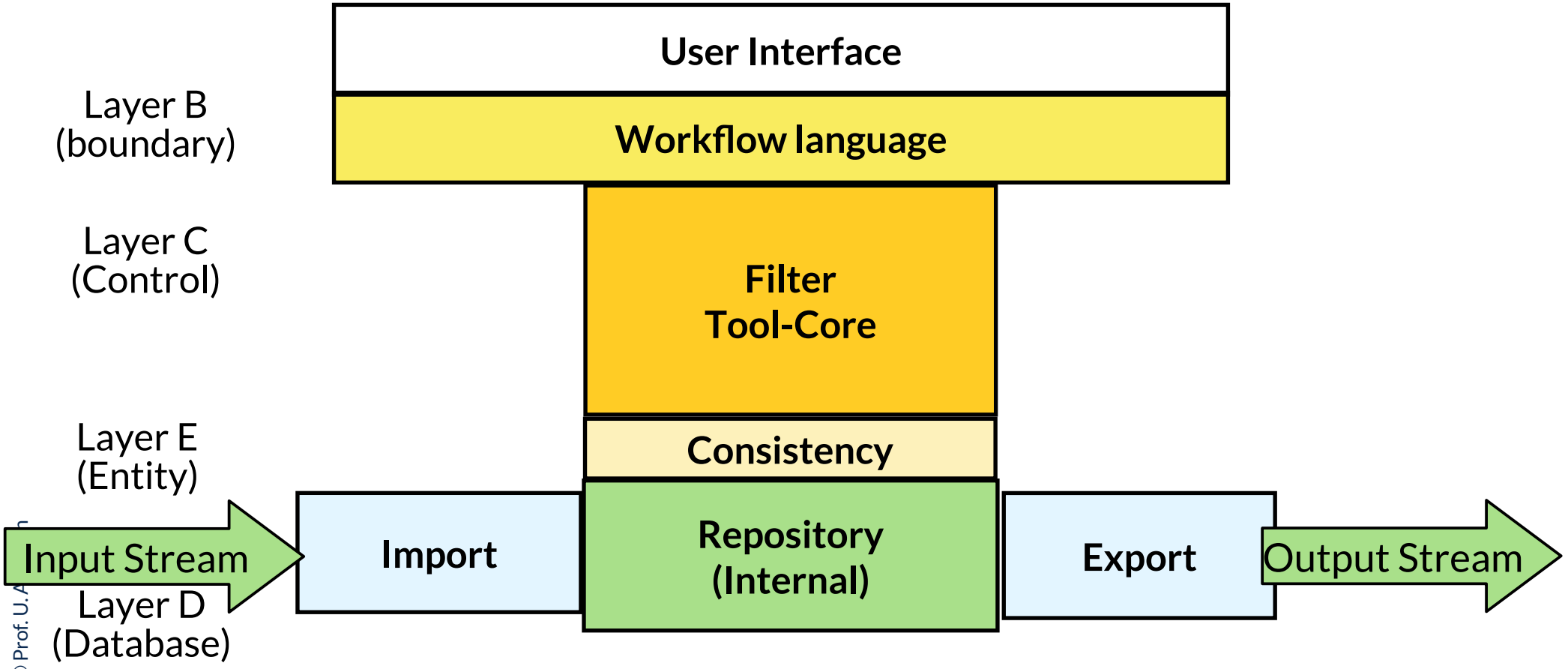
- Bell Labs developed a stream-based **UNIX Programmers' Workbench (PWB)** in 1976
  - UNIX had introduced the file system and streams (for C programs and shell scripts)
  - http://en.wikipedia.org/wiki/Programmer%27s_Workbench_UNIX
- CACM publication:
  - http://delivery.acm.org/10.1145/360000/359856/p746-ivie.pdf?key1=359856&key2=5161309211&coll=GUIDE&dl=GUIDE&CFID=55168257&CFTOKEN=9543918
- "Notable firsts in PWB include:
  - The Source Code Control System, the first revision control system, written by Marc J. Rochkind
  - The remote job entry batch-submission system
  - The PWB shell, written by John R. Mashey, which preceded Steve Bourne's Bourne shell
  - The restricted shell (rsh), an option of the PWB shell
  - The troff -mm (memorandum) macro package, written by John R. Mashey and Dale W. Smith
  - The make utility for build automation
  - Utilities like find, cpio, expr, all three written by Dick Haight, xargs, egrep and fgrep
  - yacc and lex, which, though not written specifically for PWB, were available outside of Bell Labs for the first time in the PWB distribution"

29.2 Extension of Stream-Based Tools by Workflow Languages and DFD

And composition of stream-based tools

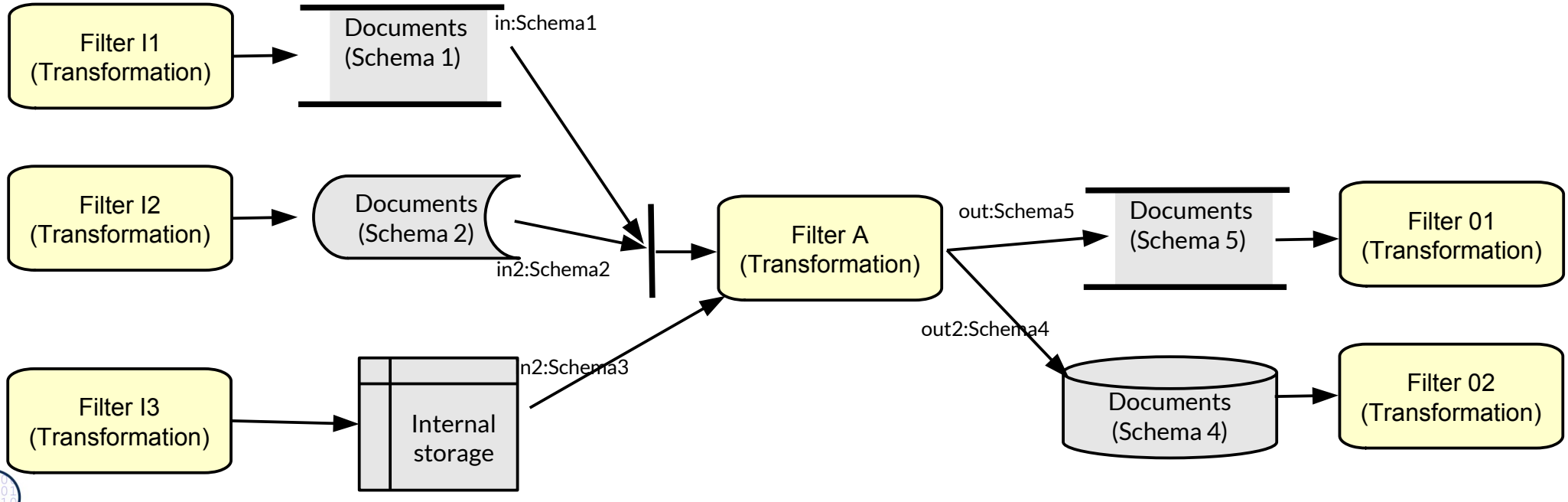
Q6: Architecture of Stream-Based Transformation Tools (*Filters*)

- In a ***filter tool***, the work, the transformation of a material, is done on one (or few) material(s) at a time
- By a DFD or Workflow (Mashup), simple tools can be composed to more complex tools, written in a DFD- or Workflow-language



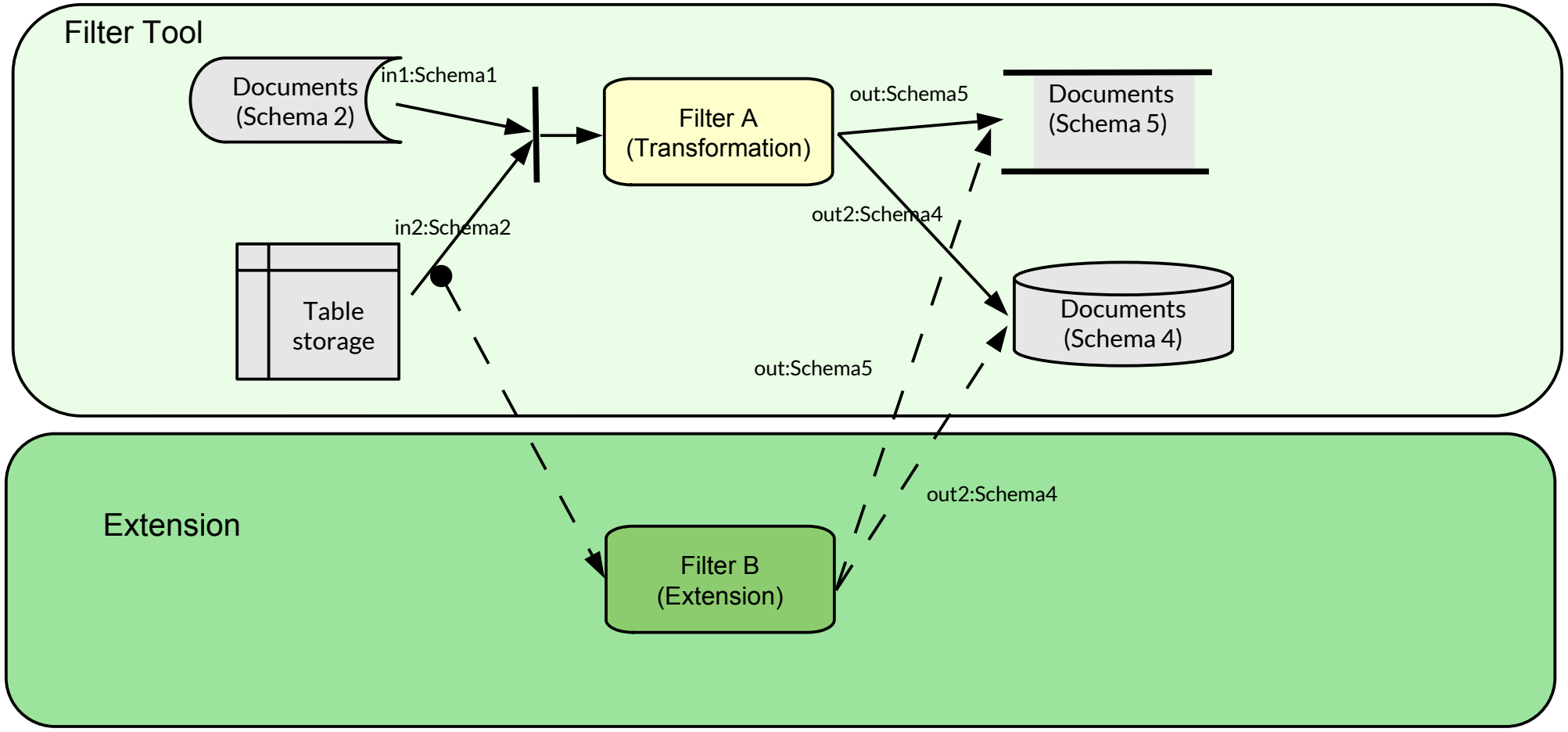
Composition of Tools by Stream Merging

- The architecture and composition of stream-based tools can be described by DFD, workflows, or (Web-)mashups
- Three composition operations are important:
  - **Input stream synchronization:** does a process read from input channels synchronously or alternatingly?
  - **Input stream merge:** how does a process merge two input channels?
  - **Output stream replication:** does a process replicate output data in different streams or produce different output formats?



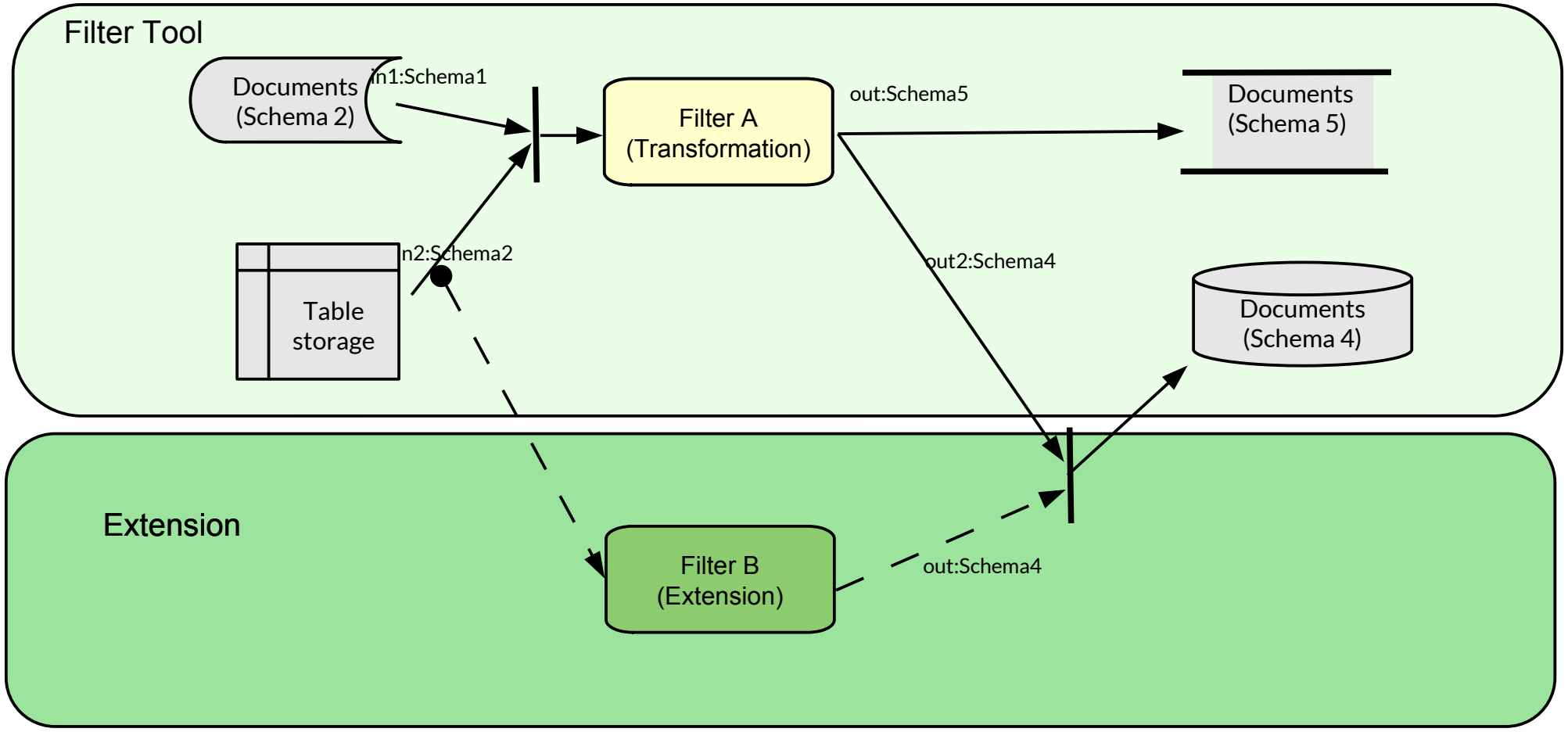
Tool Extension by Stream Duplication and Asynchronous Merge

- DFD are easily extensible, because input streams can be replicated to deliver their content into the processes of the extension (extension listening on stream of core)
- Output streams of extensions can write asynchronously into output storages or streams (asynchronous merge)



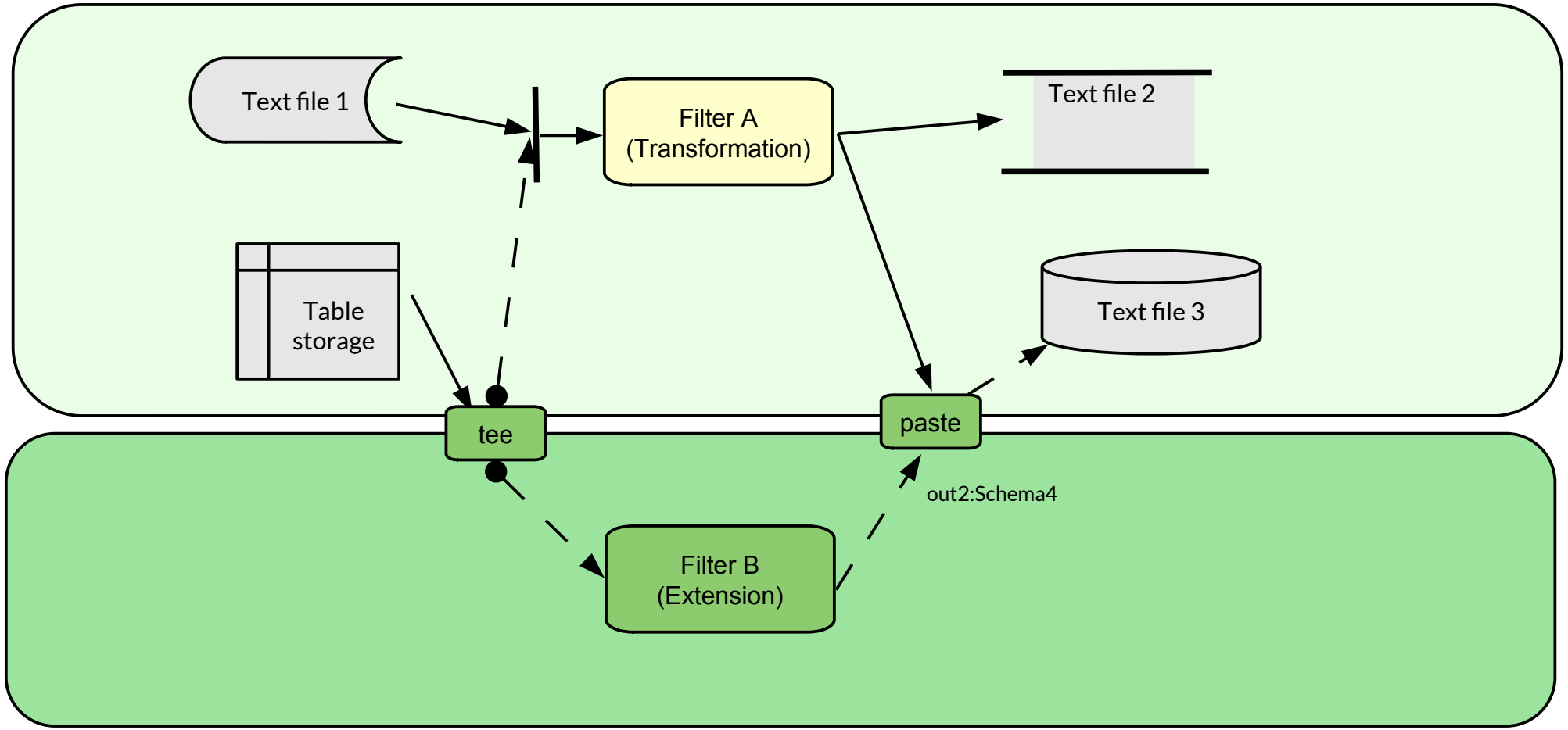
Synchronizing Extension of Core Tool

- Output streams of extensions can write synchronously into output storages by adding new synchronizing activities guarding output storages



Example: Shell Script Extension in Linux

- Streams are text streams (untyped)
- `tee` is a filter replicating a text stream
- `paste` or `lam` are filters merging two streams



29.3. Extensible Stream-Based Tools: DQL und DTL in DFD-Mashups

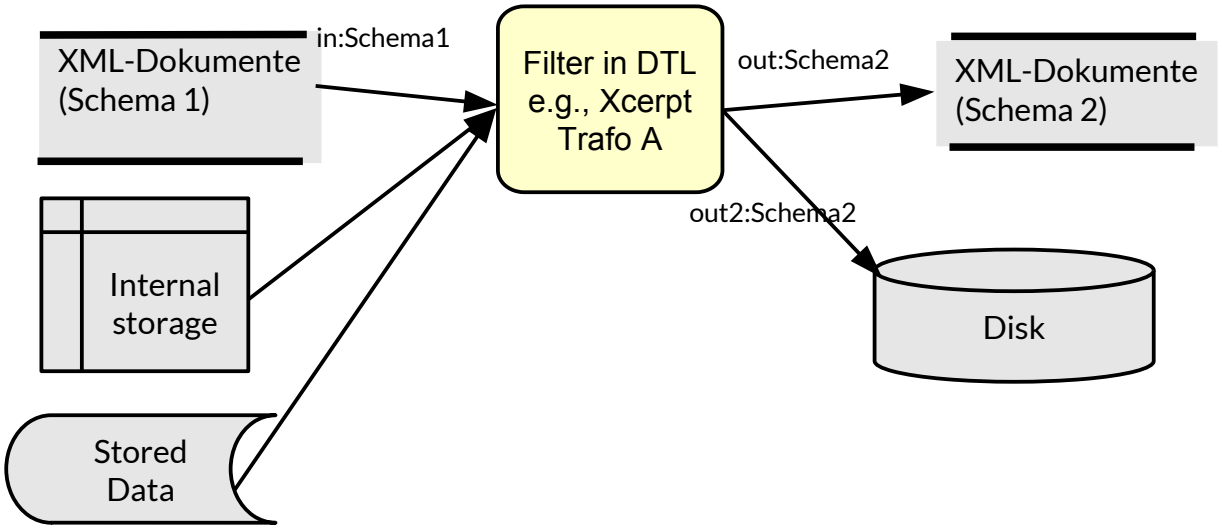
Ex.: Technical Space Treeware-XML

XML Mashups are special DFD

The example can be transferred to Graphware or Grammarware using other DQL and DTL

Use of DQL and DTL in DFD (e.g., Web Mashups)

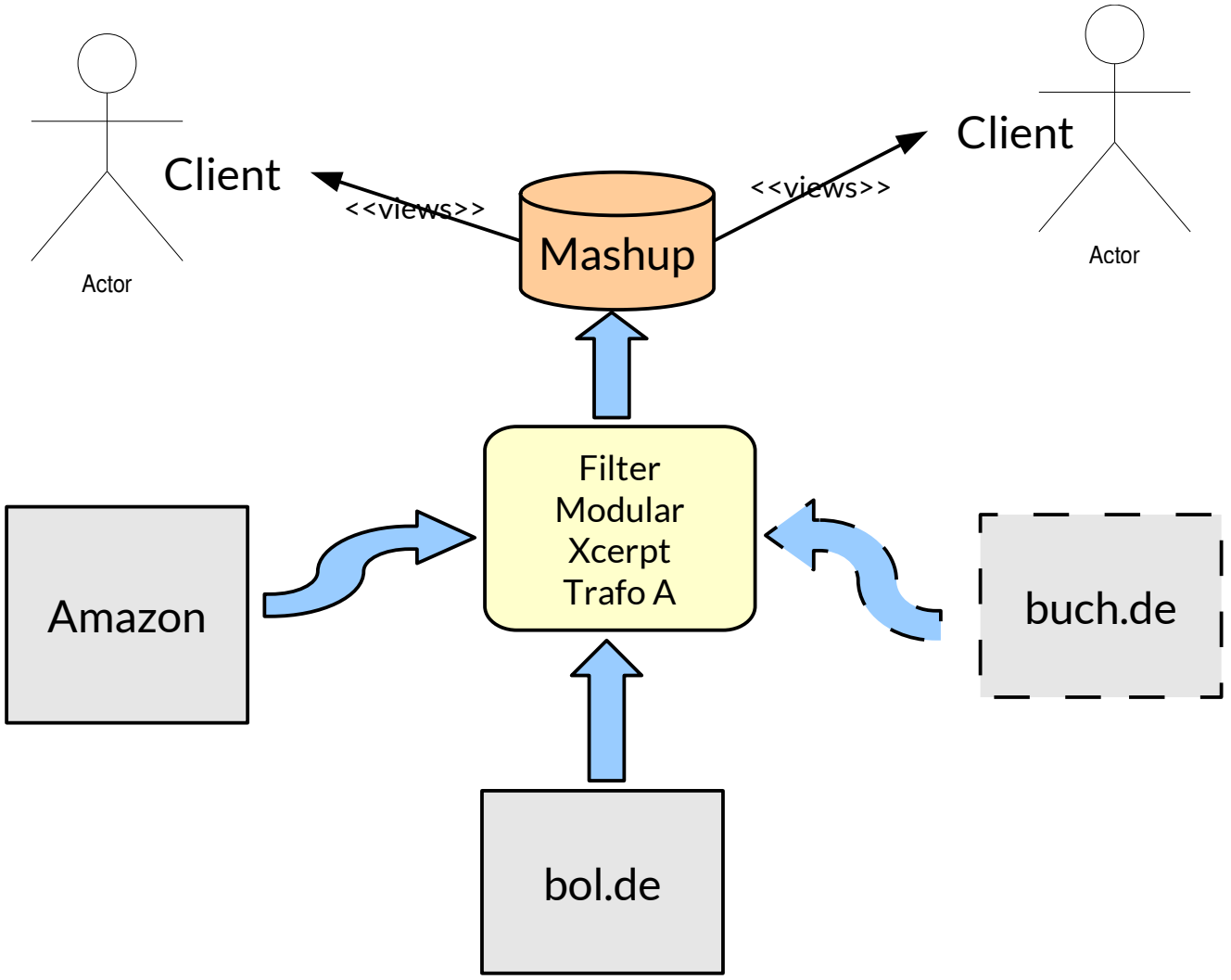
- DTL and DQL (Xquery, Xcerpt and others) can be employed for filters, generators and transformers (processes in DFD)
  - A DDL describes the types of data on the streams (types, schemata)
  - String rewrite systems can be used to specify processes if streams transport texts
  - Term rewrite systems can be used to specify processes if streams transport trees
  - XML rewrite systems: With XML and XSD, Xcerpt can be used
  - Graph rewrite systems can be used if streams transport graphs
- Mashups are XML-DFD
  - easily extensible, because channels can be replicated and extended
  - extremely important for extensible tools



XML-Mashups with Modular Xcerpt

Use Modular Xcerpt for creating a CD mashup of our favourite music LPs

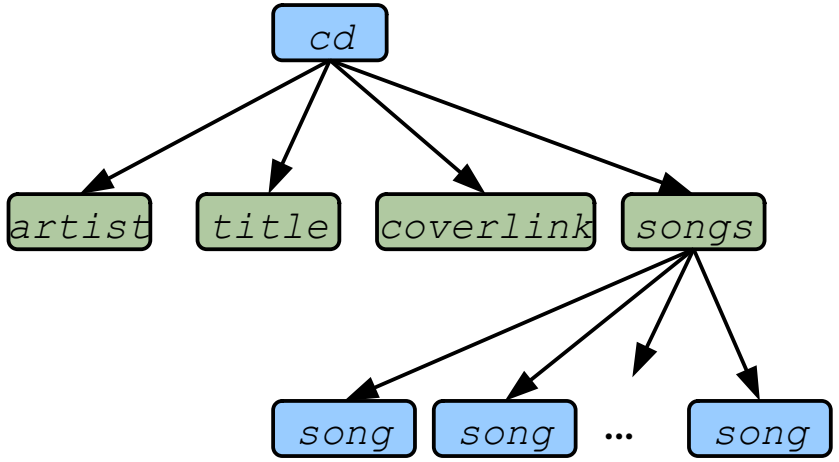
- "mashing-up" freely available data from online stores
- easily extensible with new sources or processing steps



Mashups with Modular Xcerpt

- First we need a data structure for CDs, so that we can use it for our virtual store of aggregated data
- Model with Xcerpt data terms (XML trees)

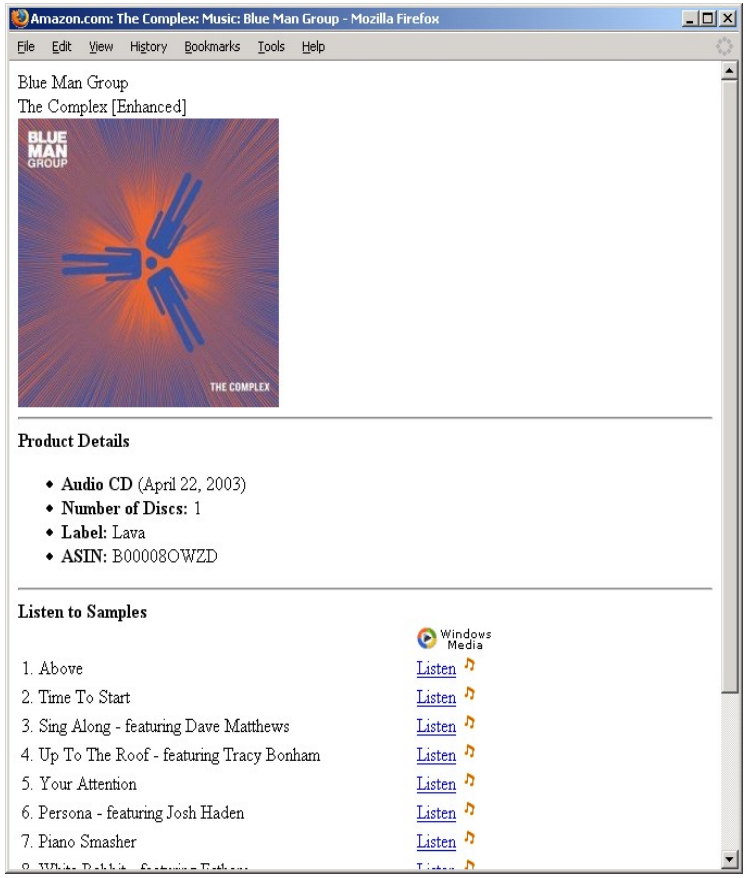
```
                _
cd [
            artist,
            title,
            coverlink,
            songs [
                        song, song … song
            ]
]
```



Mashups with Modular Xcerpt

- Next step: creating import modules to aggregate data from our sources

```
MODULE AmazonQuery
FROM
public html [
                head [[ ]],
                  body [[
                              var ARTIST, br,
                              var TITLE, br,
                              img {
                                attributes {src { var COVERLINK }}
                              },
                              table [[
                                  tr [
                                    th [[ ]]
                                  ],
                                tr[
                                    td [ var SONGTITLE ],
                                  td [[ ]]
                                ]
                        ]]
          ]]
     ]
CONSTRUCT
public cd [
              artist [ var ARTIST ],
              title [ var TITLE ],
              coverlink [ var COVERLINK ],
              songs [
                all song [ var SONGTITLE ]
              ]
]
END
```



(Example HTML Source)

Mashups with Modular Xcerpt

- Import modules are independent from a concrete source
  - pass the resource locations to the modules
  - collect all data from modules by introducing a virtualroot node (dummy)

```
MODULE MainProgram

IMPORT /import/AmazonQuery.mxcerpt AS Amazon
IMPORT /import/BuchdeQuery.mxcerpt AS BuchDE

CONSTRUCT to Amazon (
    var DATA
)
FROM
  in {
    resource { "file:data/amazon-blue_man_group-
                        the_complex.html", "xml" },
    var DATA
  }
END

CONSTRUCT to BuchDE

  ...
END
```

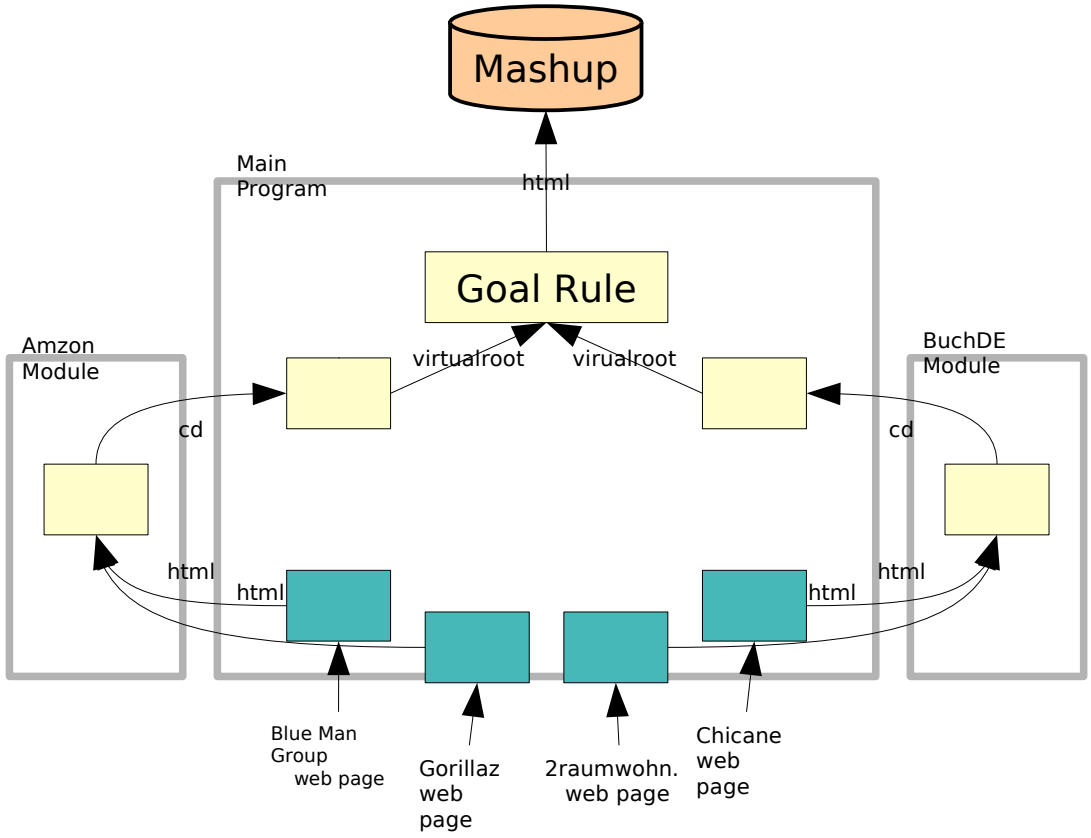
```
// Filling variable CDINFO with
// dummy virtual root node
CONSTRUCT
            virtualroot [ all var CDINFO ]
FROM in Amazon (
            var CDINFO -> cd [[ ]]
)
END

CONSTRUCT
            virtualroot [ all var CDINFO ]
FROM in BuchDE (
            var CDINFO -> cd [[ ]]
)
END
```

Mashups with Modular Xcerpt

- Construct rules "mash up" the data – create a new webpage
  - in Xcerpt a goal rule must be specified (program entry point)

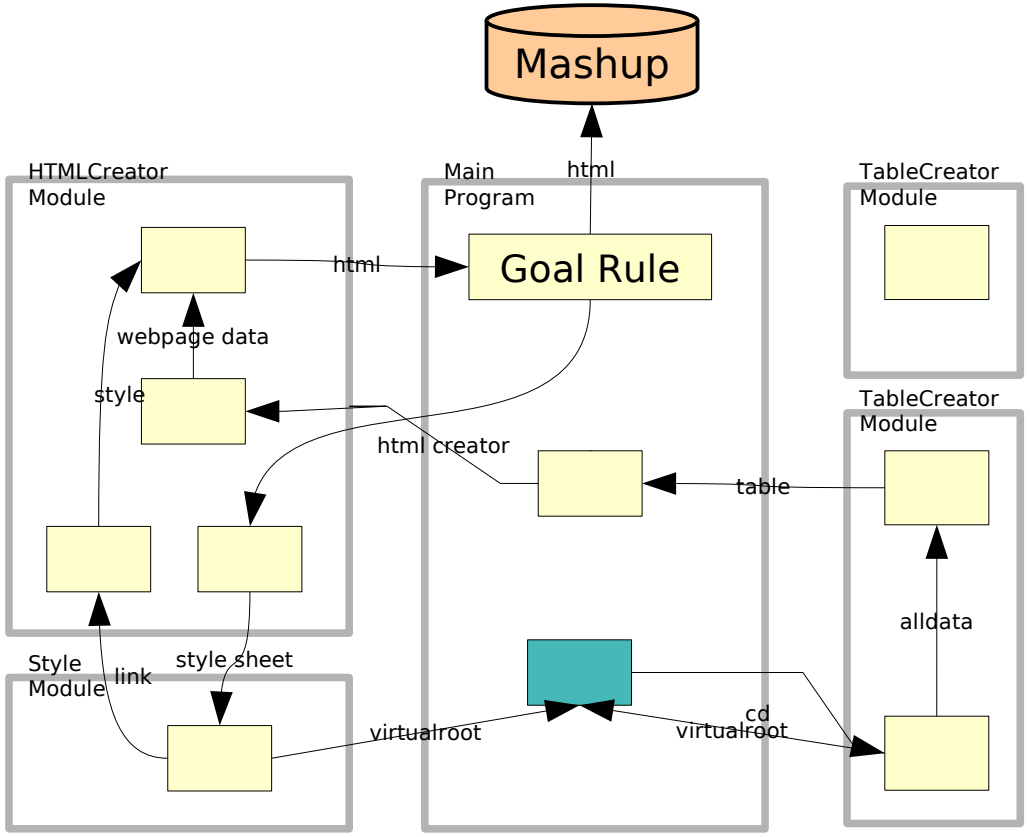
```
FROM
virtualroot [[
   cd [[
         artist [ var ARTIST ],
         title [ var TITLE ]
      ]]
  ]]
GOAL
out {
   resource {"file:mashup.html", "xml"},
   html [
      head [
         title ["Mashup"]
      ],
      body [
         table [
                                 all tr [
                     td [ var ARTIST ],
                     td [ var TITLE ]
               ]
         ]
      ]
  ]
}
END
```



(Structure of the Modular Xcerpt program)

Further Decomposition of Mashup Possible

- Further decomposition of program possible
  - HTML creator can be an extra module
  - Table layout and style sheet linking can be made configurable

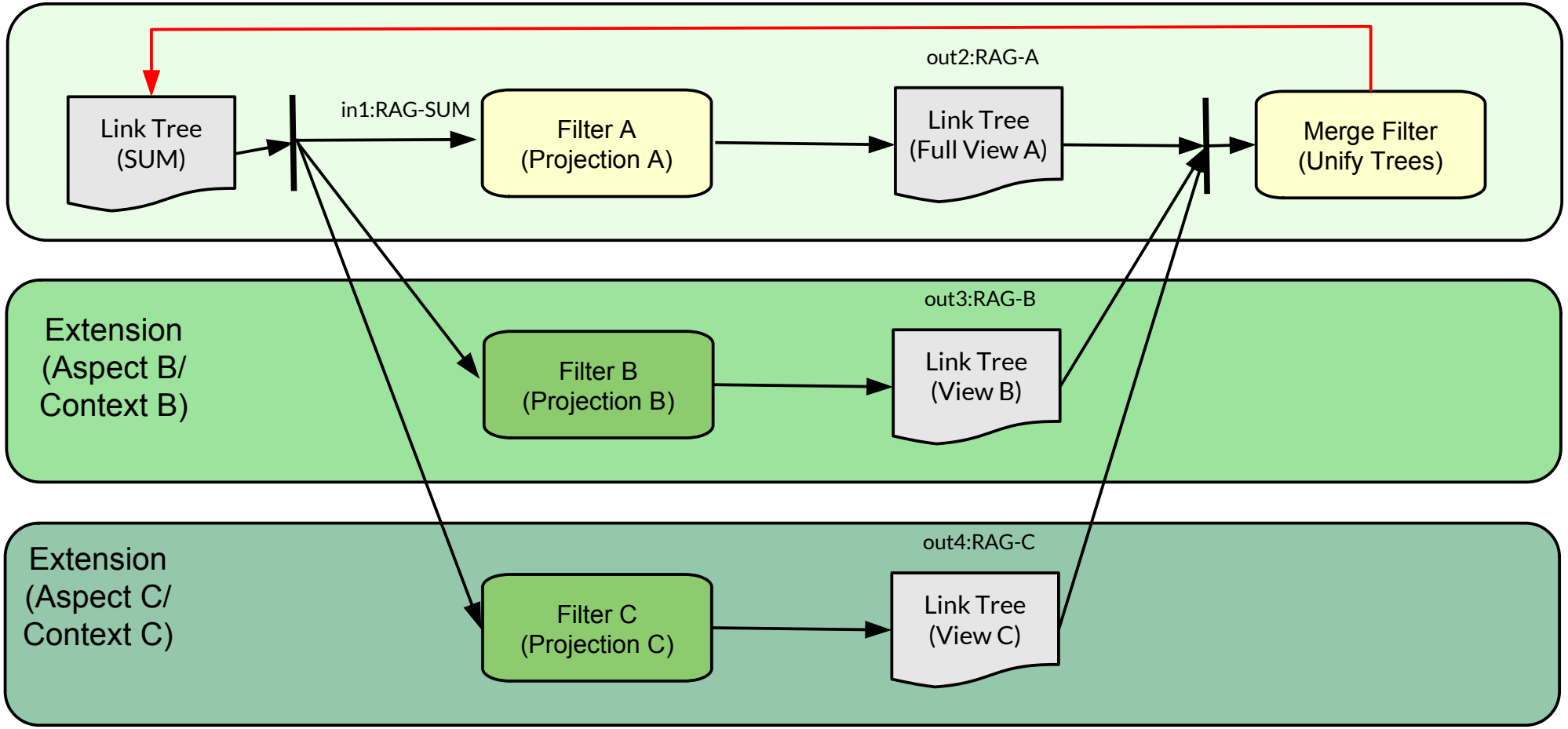


(advanced Modular Xcerpt program)

29.4. Macromodels with Stream-Based Tools (DFD Aspects)

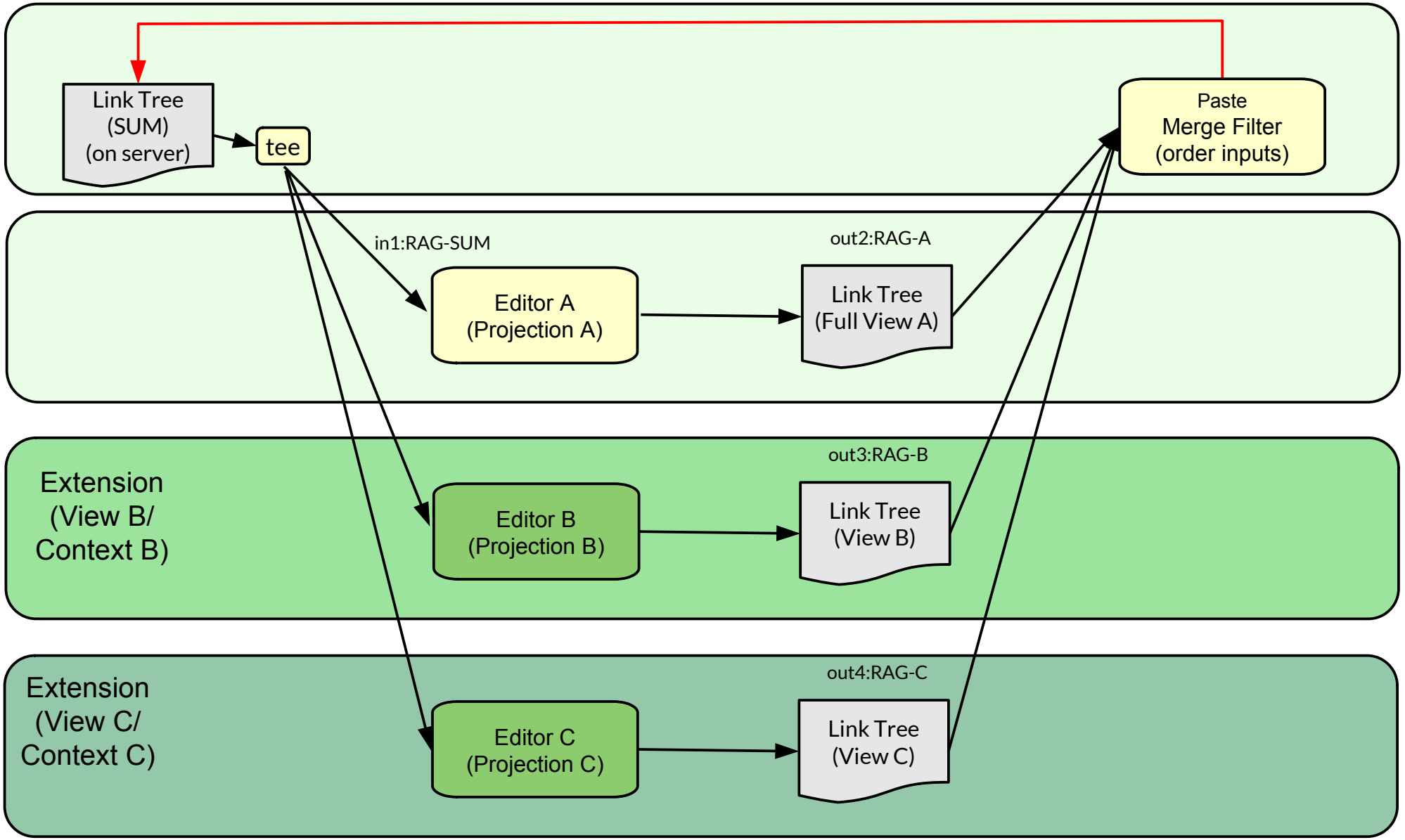
Stream-Skeleton-SUM
View Extension and Merge

- Projection operations can be used to form Views of a SUM (view-get operation)
- Adding a new extension adds a new view
- If filters are implemented by RAG, projections (view-get) can be calculated by projection on attributions
- RAG-A, -B and -C are assumed to be subsets of RAG-SUM



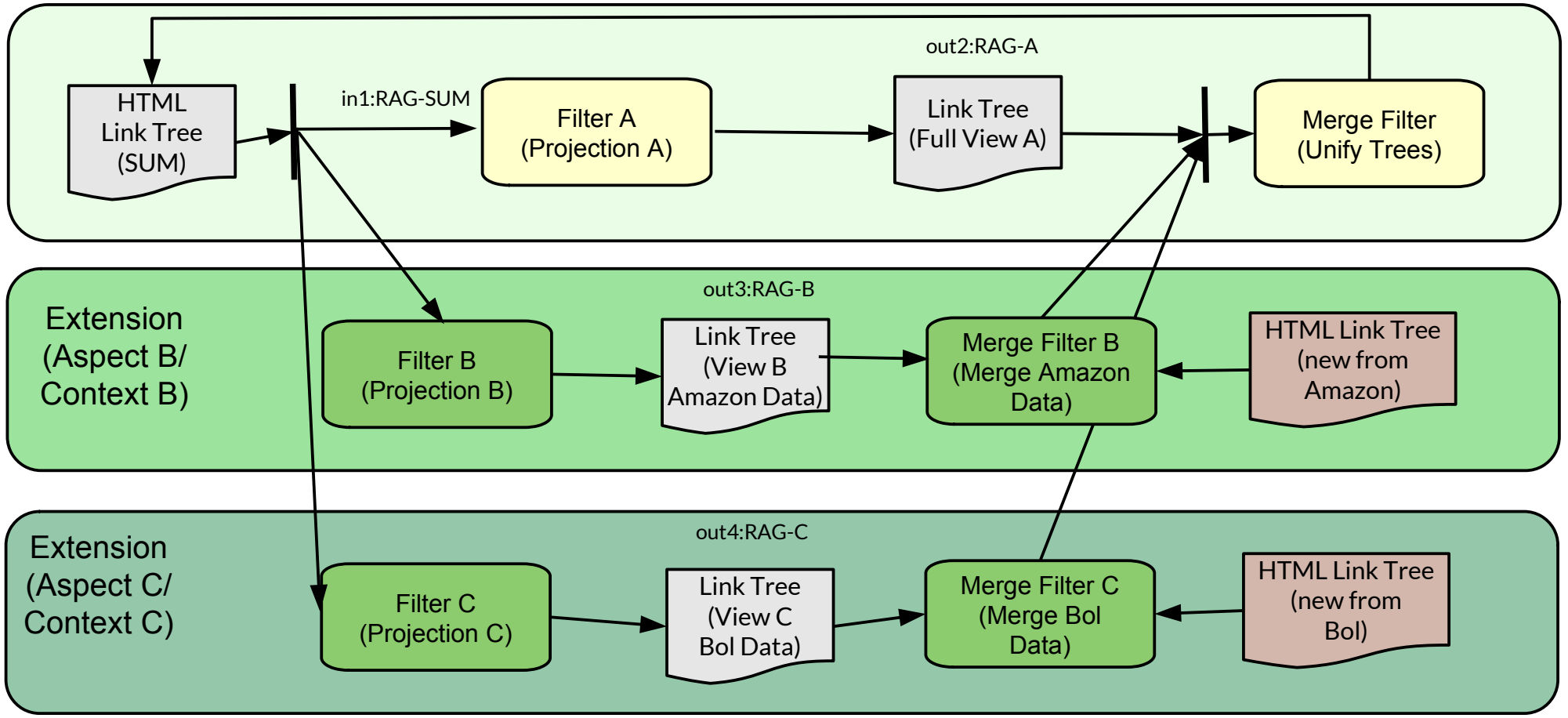
Google Docs as Stream-Skeleton-SUM View Extension and Merge

- Document is a link tree in XML format



Stream-Skeleton-SUM for Mashedup Websites, with Intake of New Data

- New data can be taken into the SUM from every view

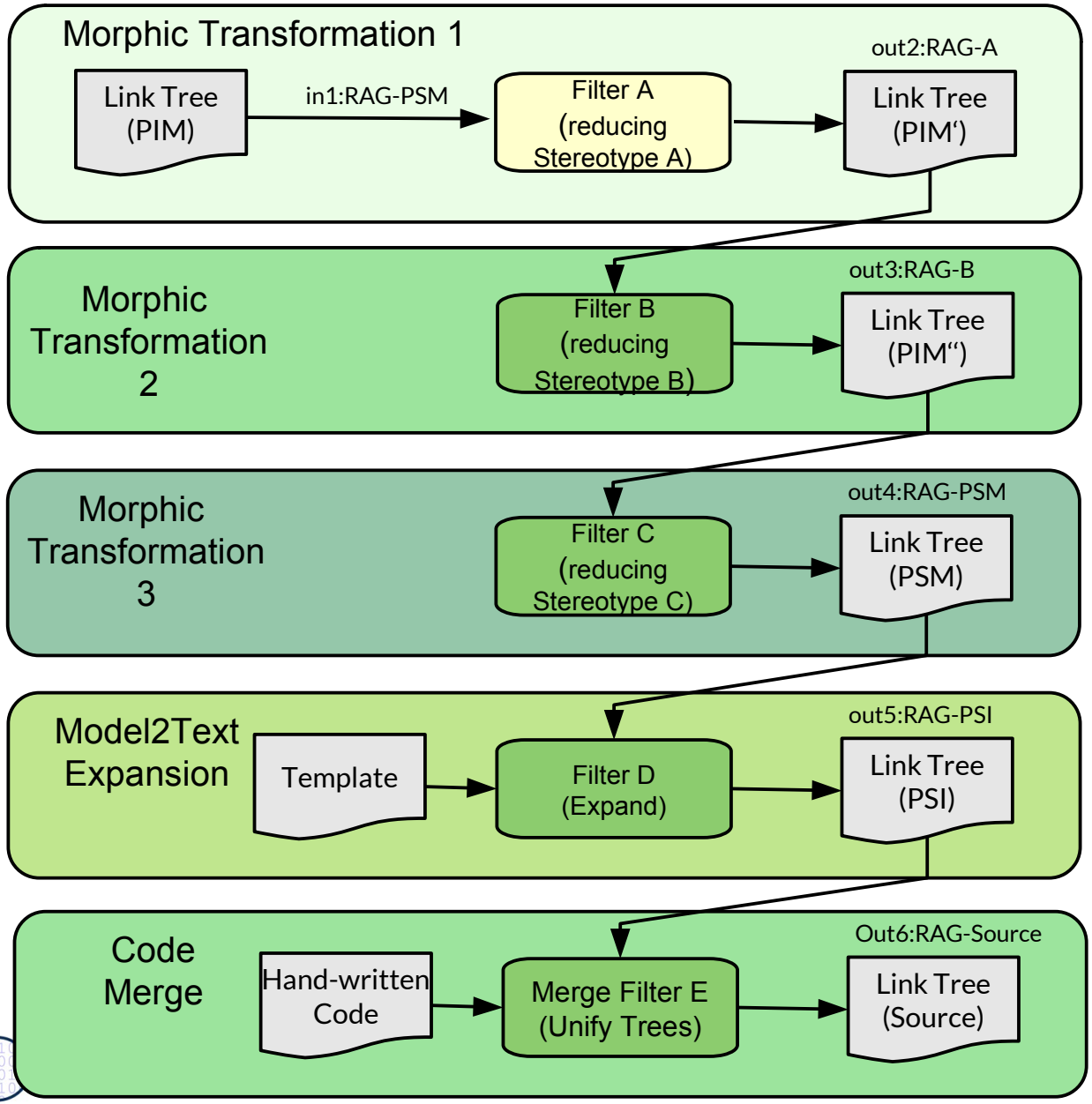


29.4.2. MDA Macromodels with Stream-Based Tools (DFD Aspects)

MDA by Composition of DFD Aspects

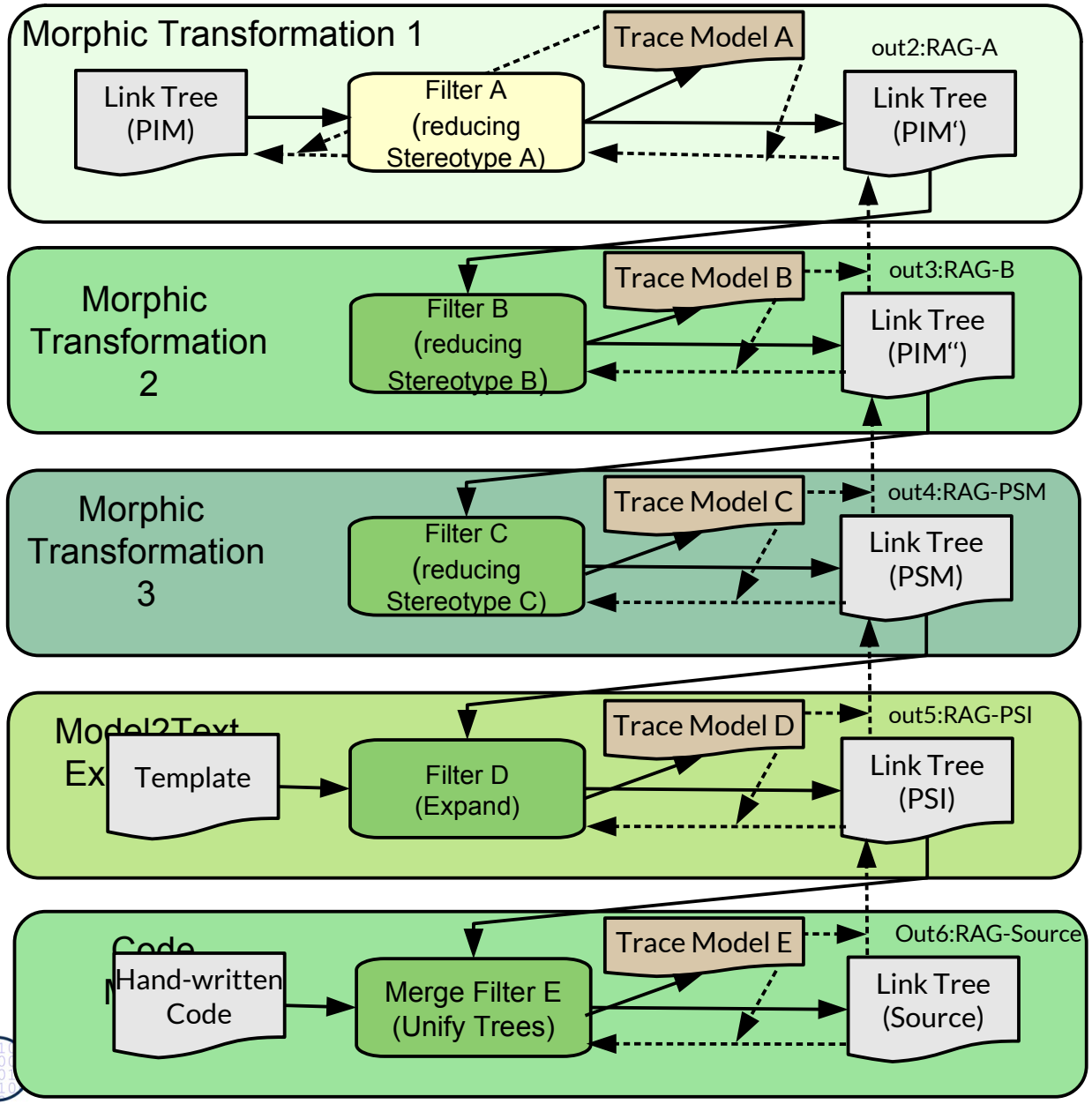
- DFD modules can be used as MDA cartridges
  - They compose process extensions “around” stream names
  - Model weaving is done by stream copying, decomposition and composition
- Model Transformation and Template expansion (in MDA) is done by modular composition (aspect composition) with DFD modules and filters
  - Model synchronisation is done by re-composition
  - DFD-MDA supports **composable** and **decomposable macromodels**

MDA Stream-based Lowerings (Forward Transformations)



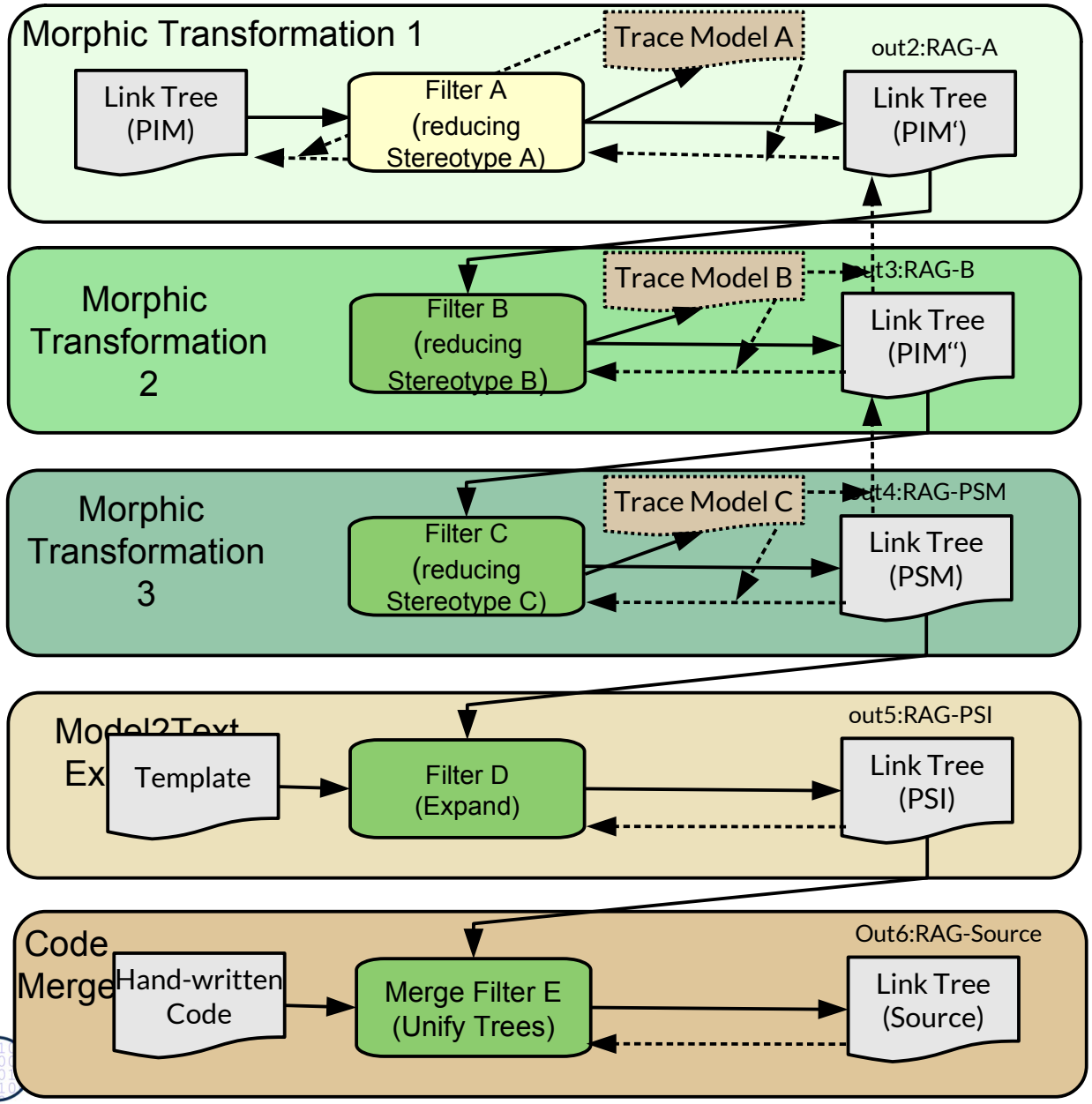
- Morphic lowering operations can be used transform the PSM step by step to the PIM
- As well as template-expand and code-merge

MDA Stream-based Lowerings with TraceModels (MDA Stream Macromodel)



- Morphic lowering operations can be **traced in a trace model**
- This trace model allows for inverting all transformations by disambiguating reductions
- A macromodel results: Every Link Tree can be edited and synchronized with the other models

MDA Stream-based Lowerings with RAGs (MDA Stream Macromodel)



- With RAG, trace models can often be slim, because the dynamic dependency graph of attributions records traces anyway!
- Saved:
  - Trace model of Template expansion
  - Merging Trace Models
- If we combine

  RAG-SUM = RAG-A + RAG-B + RAG-PSM + RAGPSI + RAG-Source,

  the RAG-SUM computes all traces in the dependency graphs

How Do I Construct a Macromodel Myself?

- 1) Decide on stream-based or repository-based architecture
- 2) Decide on Skeleton-SUM or MDA
  - Use RAGs for all models
  - Use CROM/Scroll
- Repository-based Skeleton-SUM:
  - Get full traceability and synchronization for GET and PUT operations
  - Scroll programming for GET and PUT
- Repository-based MDA:
  - Use RAG aspects and get full traceability
  - ROSIMA is a Mono-Skeleton-SUM with GET as deactivation of contexts and PUT as activation of contexts
- Stream-based Skeleton-SUM:
  - Use RAGs for all models to save trace models and get SUM as in-place transformation
- Stream-based MDA:
  - Use RAGs for all models to save trace models and get traceability as good as possible

The End – What did we learn?

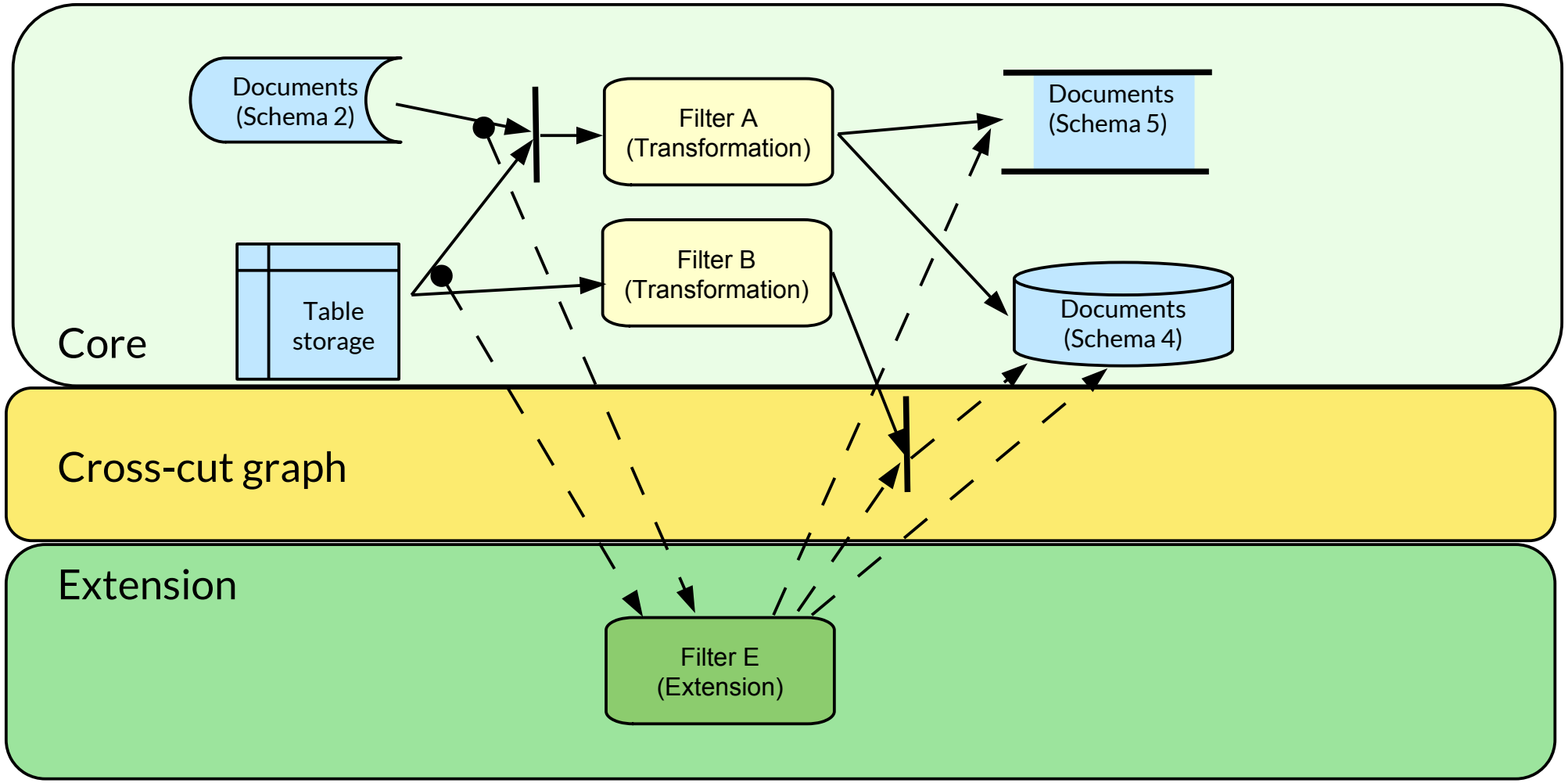
- Stream-based tools, filters, can easily be extended and composed
  - with input stream replication
  - with asynchronous or synchronous output stream merge
  - with aspect-oriented extension
- Tools should be composed only with regard to their Essence, disregarding Administration and Infrastructure aspects
- Macromodels can be stream-based
  - Explain a stream-based Skeleton-SUM macromodel
  - Explain a stream-based MDA macromodel
  - How are trace models saved?

29.5. Aspect-Oriented XML-Weaving with XML Transformations

- For aspect-orientied extensions of DFD und Mashups

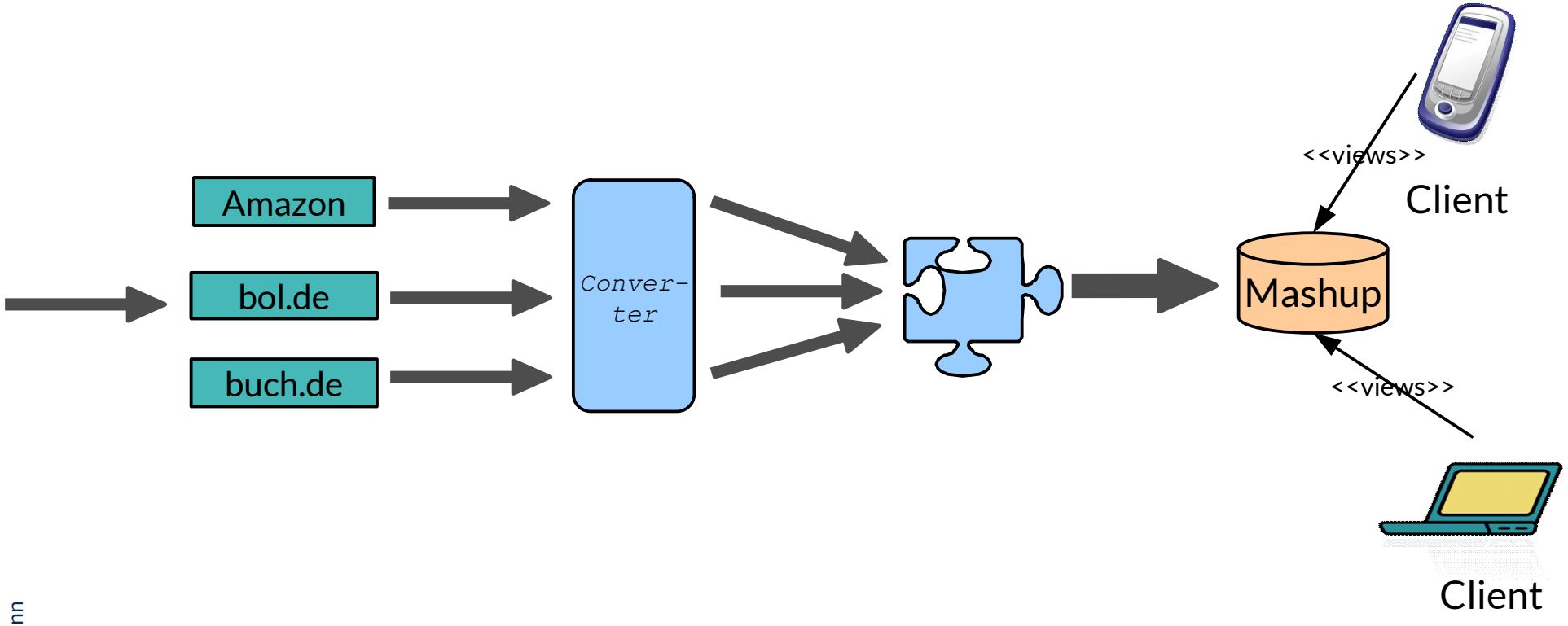
Aspect-Oriented Tool Extension by Crosscut-Graph between Core and Extension

- If an extension extends many places in a core (scattering), a ***crosscut-graph*** results describing the scattering



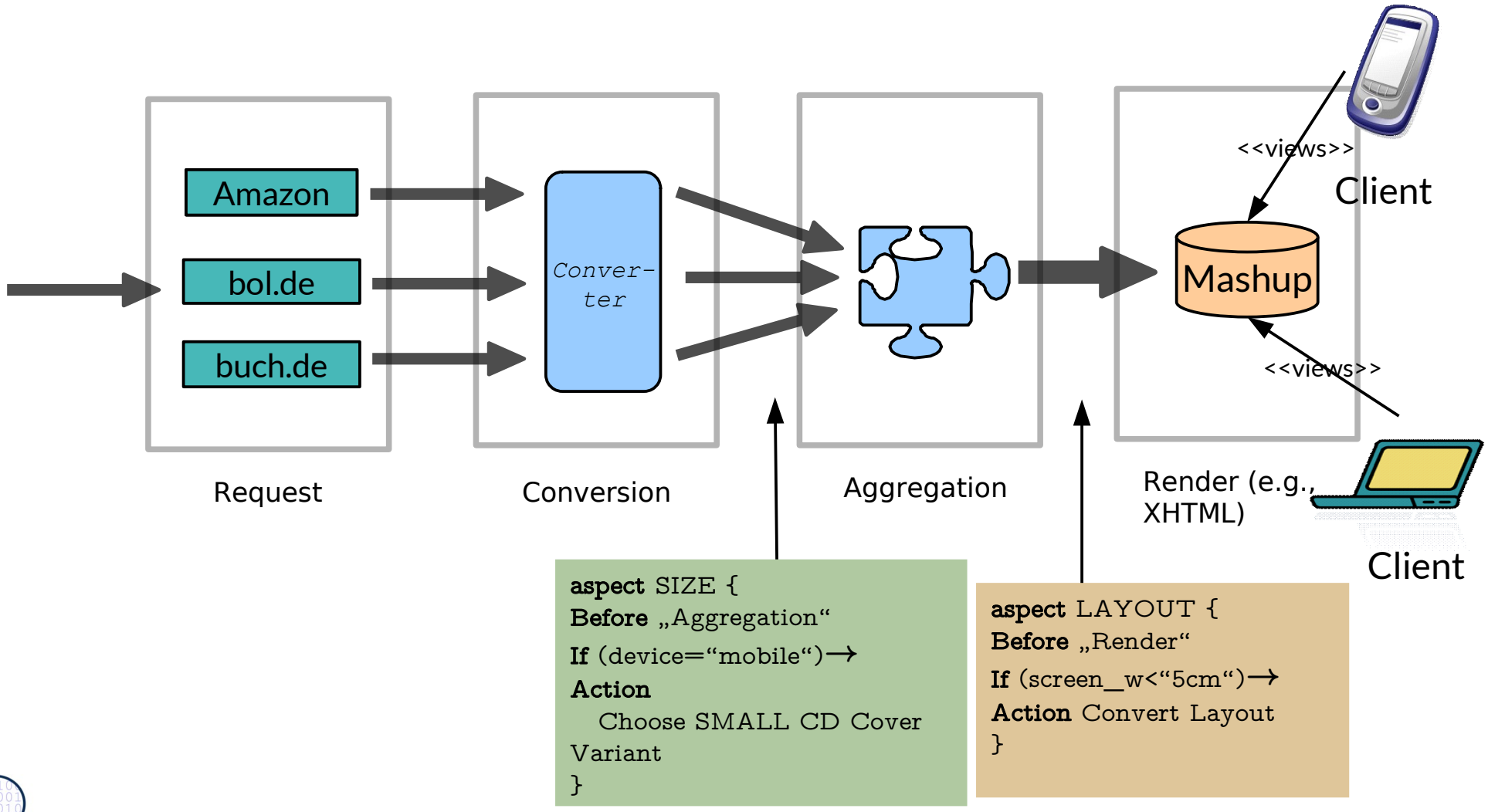
XML Adaptation Aspects (HyperAdapt Weaver)

- Xcerpt mashups induce data-flow architecture
- Mashups should be rendered for different target devices, e.g., mobiles, tablets → *Adaptation Aspects*



XML Adaptation Aspects (HyperAdapt Weaver)

- The tool "HyperAdapt Weaver" modifies the streams by transformation: "aspect actions" are "woven" into the stream



XML Adaptation Aspects (HyperAdapt Weaver)

- Example: Virtual Storage Music Database before aggregation phase as plain XML
- Selection of fragments with regard to device type (global variable)

aspect SIZE {
Before „Aggregation“
If (device=“mobile“)→
Action Choose SMALL CD Cover Variant
}

```
<music-database xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://music music.xsd" xmlns="http://music">
   <album inStock="Yes">
      <title>How to Be a Megastar-Live!</title>
      <artist>
            <pseudonym>Blue Man Group</pseudonym>
      </artist>
      <id>B00166GLVO</id>
          <edition>First</edition>
          <publisher>Rhino (Warner)</publisher>
          <image size="SMALL" url="..."/>
          <image size="LARGE" url="...SS500_.jpg"/>
          <image size="TINY" url="...SS500_tiny.jpg"/>
          <media>
             <medium kind="CD">
                <tracks>
                   <song name="Above" length="3.30" />
                   <song name="Drumbone" length="3.25" />
                   <song name="Time To Start" length="4.22" />
                   <song name="Up To The Roof" length="4.16" />
                   <song name="Altering Appearances" length="2.23" />
                   <song name="Persona" length="4.12" />
                   <song name="Your Attention" length="4.04" />
                   <song name="Piano Smasher " length="6.01" />
                   <song name="Shirts And Hats" length="4.40" />
                   <song name="Sing Along" length="3.10" />
                </tracks>
             </medium>
          </media>
   </album>
</music-database>
```

(Pictures from amazon.de)

XML Adaptation Aspects (HyperAdapt Weaver)

- Example: Document adaptation specified as HyperAdapt Adaptation Aspect, written in the XML-based HyperAdapt Aspect Language
  - Interpreting these aspects, the weaver weaves aspect slice into streams

```
<?xml version="1.0" encoding="UTF-8" ?>
<aspect name="choose-image">
   <interface>
       <core id="core" type="http://music" />
   </interface>
   <adviceGroup>
      <scope>
            <xpath>/music:music-database</xpath>
            <before>Aggregation</before>
      </scope>
      <advices>
        <chooseVariant>
              <pointcut>/music:album/music:image[1]</pointcut>
        </chooseVariant>
      </advices>
   </adviceGroup>
</aspect>
```

document namespace

process stage (joinpoint)

adaptation rule (advice)

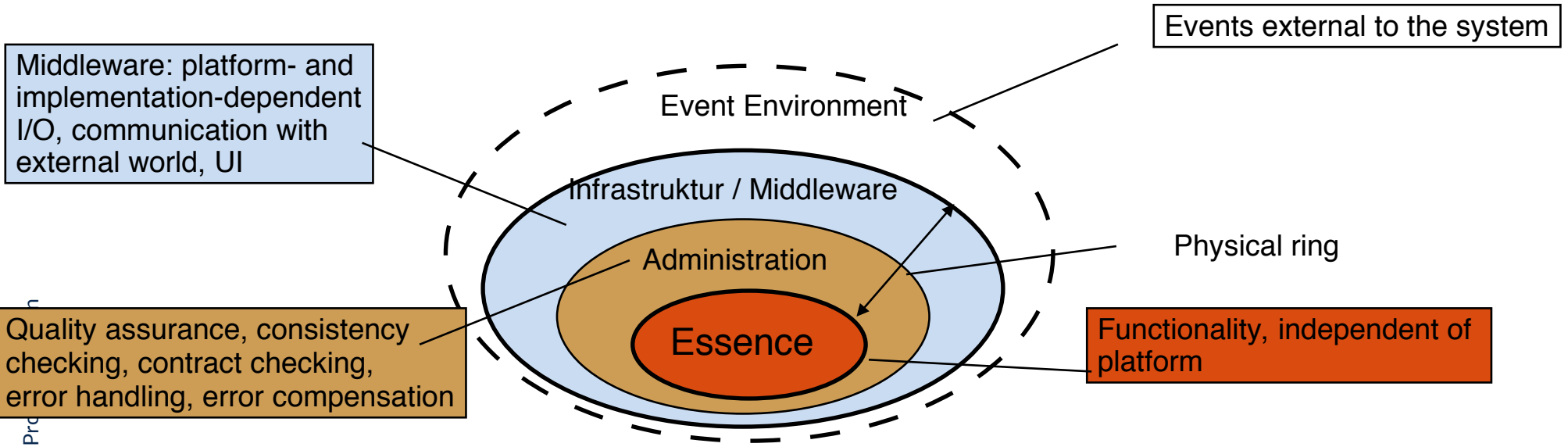
29.6 Essential Decomposition of Tools

Development with DFD

- **Prozess-oriented Refinement/Decomposition** refines processes/activities step by step into smaller processes (divide-and-conquer)
  - One dimension of decomposition
- **Essential Decomposition** uses aspect-oriented decomposition and distinguishes three aspects: [McMenamen/Palmer]
  - Essence (E): essential processes, activities, storage. Functionality that cannot be stripped
  - Administration (A): administrative activities (for consistency checking of data in internal storages;for contract checking of processes on input and output streams)
  - Infrastructure (I): activities for communication and adaptation to platform (platform-specific details)

Olympic Rings (EAI-Decomposition)

- **Essential decomposition (EAI decomposition)** separates the **essence** of a system from implementation-specific parts (**infrastructure**) and quality assurance (**administration**).
- Essential functionality assumes perfect technology [McMenamen/Palmer]
  - Processes do not need time, storage with unlimited capacity
-



EAI-Decomposition of DFD-Based Tools

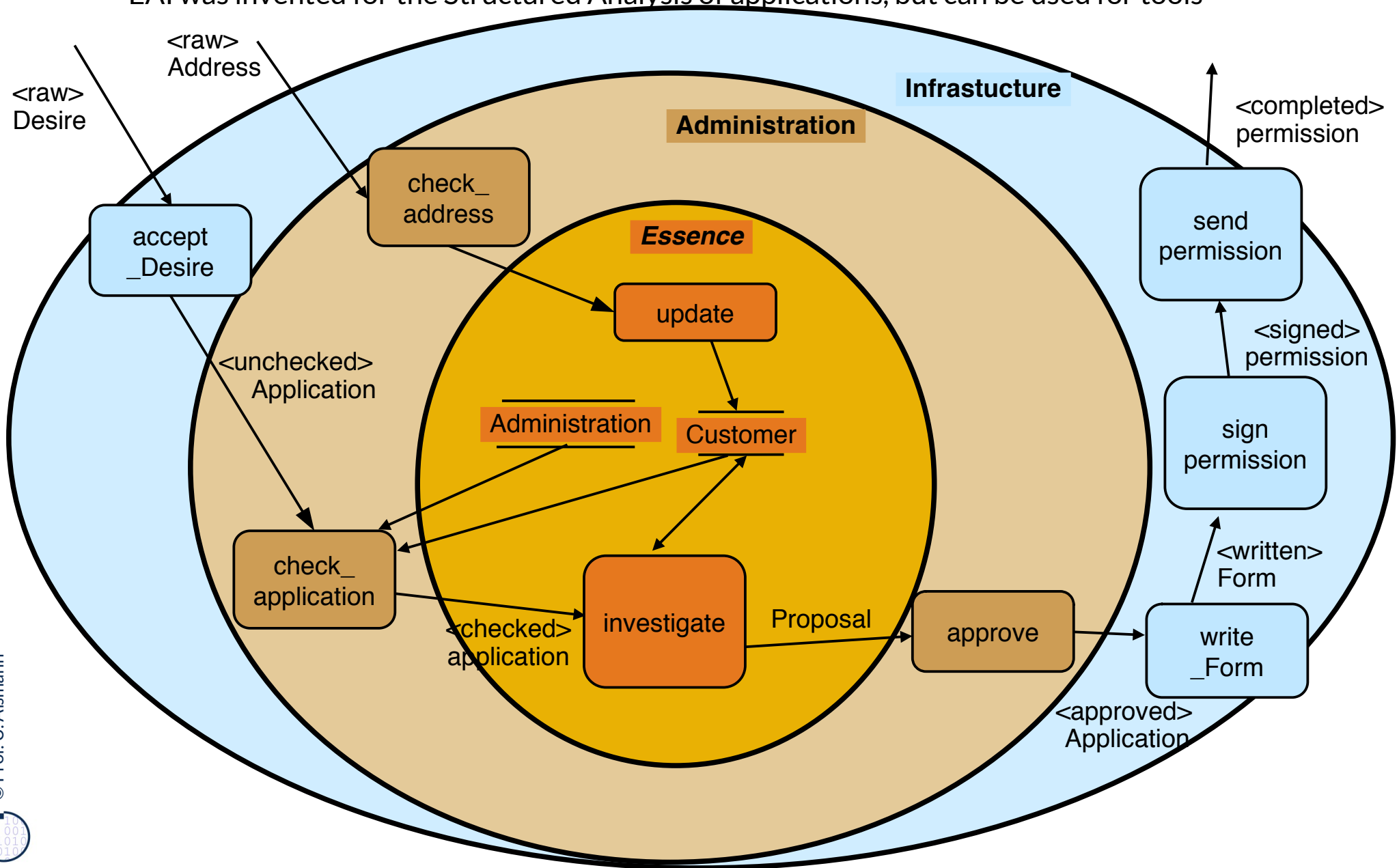
- With DFD, the decomposition into EAI-aspects (Essence, Administration, Infrastructure) is simple
  - Every model element is given a *direct concern* by the user
  - The rest is graph slicing

EAI-concerns of a tool:

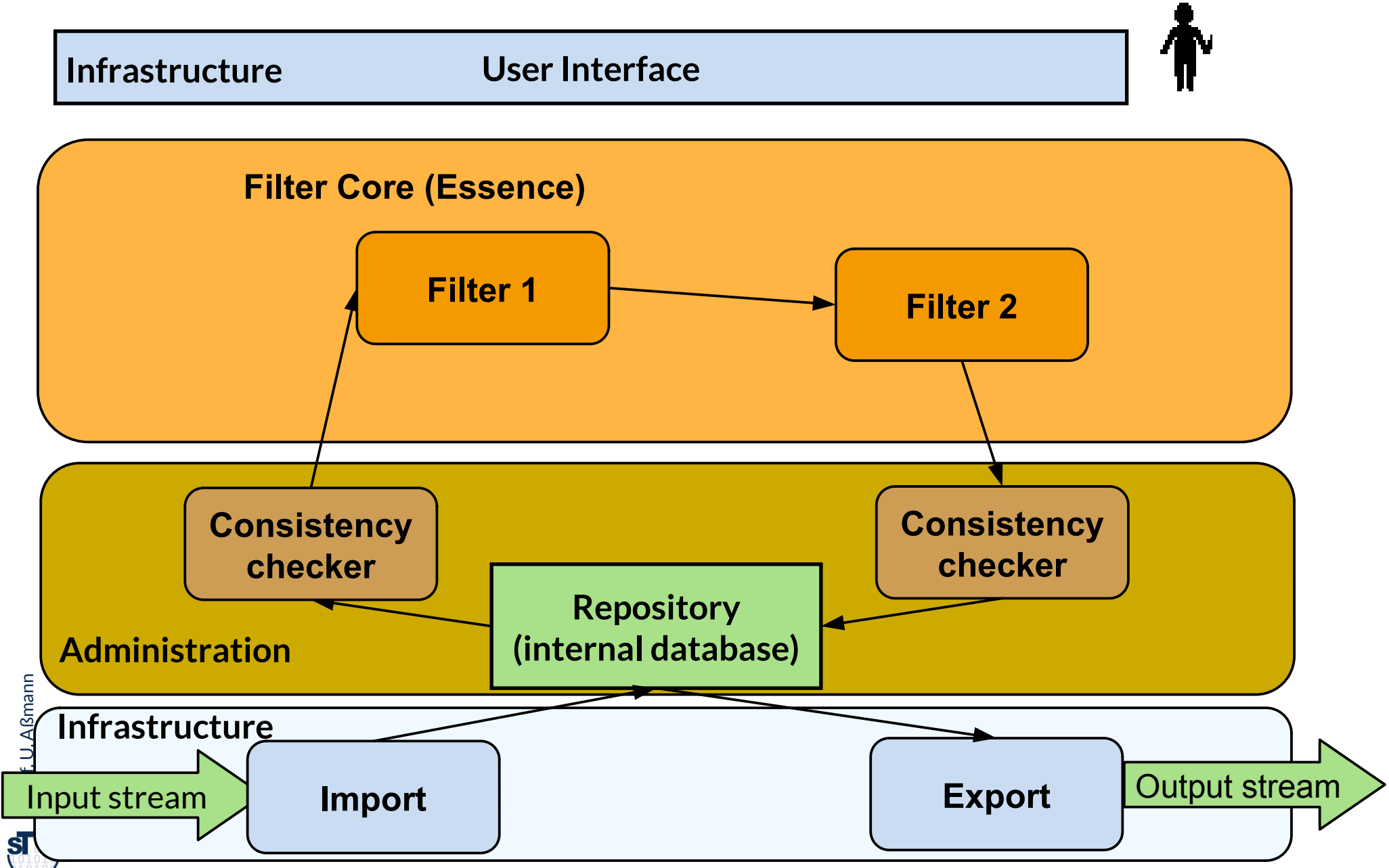
- **Essence** of a tool:
  - Functionality assuming perfect technology
- **Administration** of a tool:
  - Constraint checker
  - Contract checkers on streams
  - Wellformedness checker on internal repository
- **Infrastructure** of a tool:
  - Parser, tree constructor (import)
  - Pretty printer, code generator (export)

Ex. EAI-Decomposition of a Process of a Tool “Task Management System”

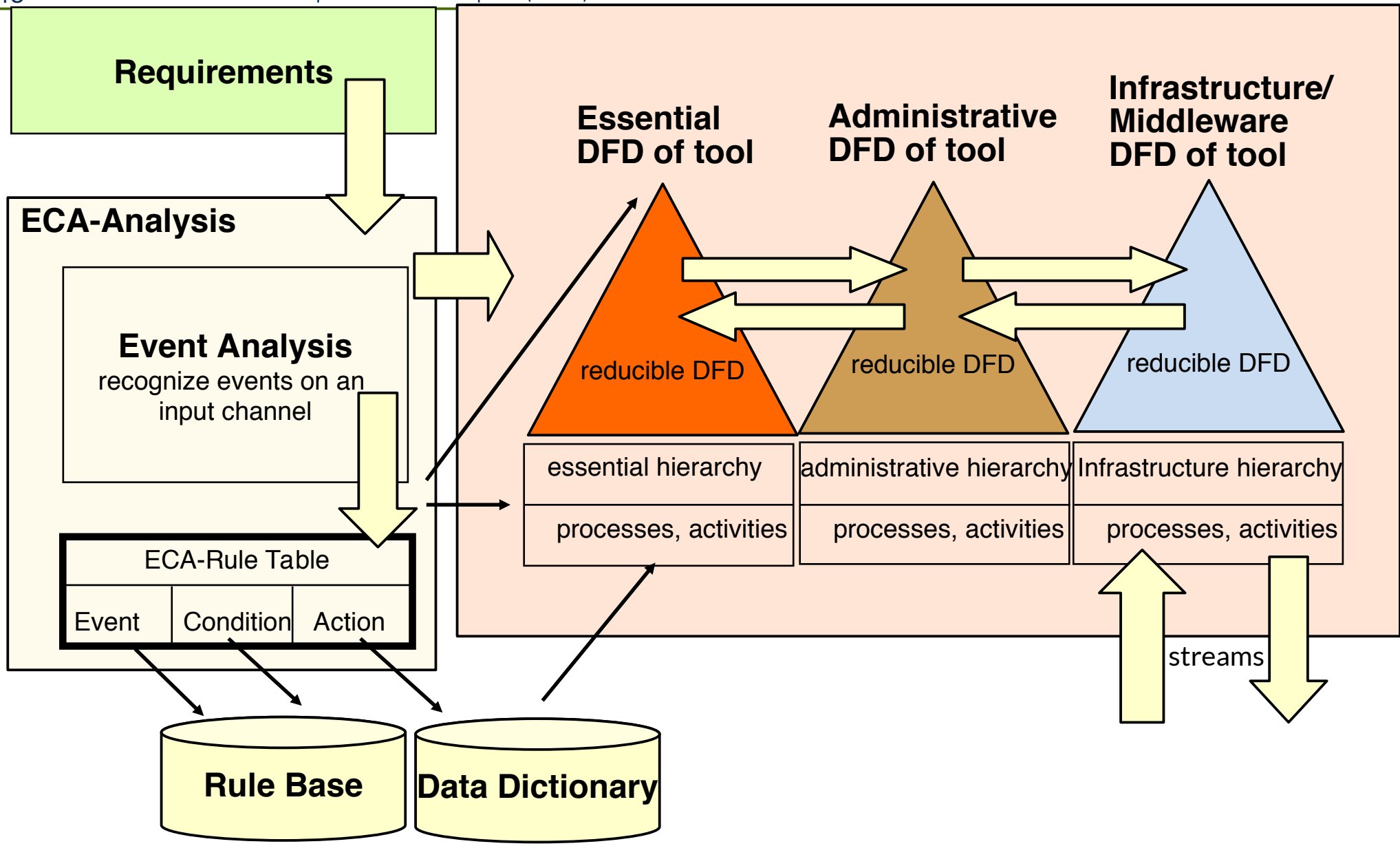
- EAI was invented for the Structured Analysis of applications, but can be used for tools



EAI-Decomposition of a Stream-Based Tool



Essential Structured Analysis for Tools

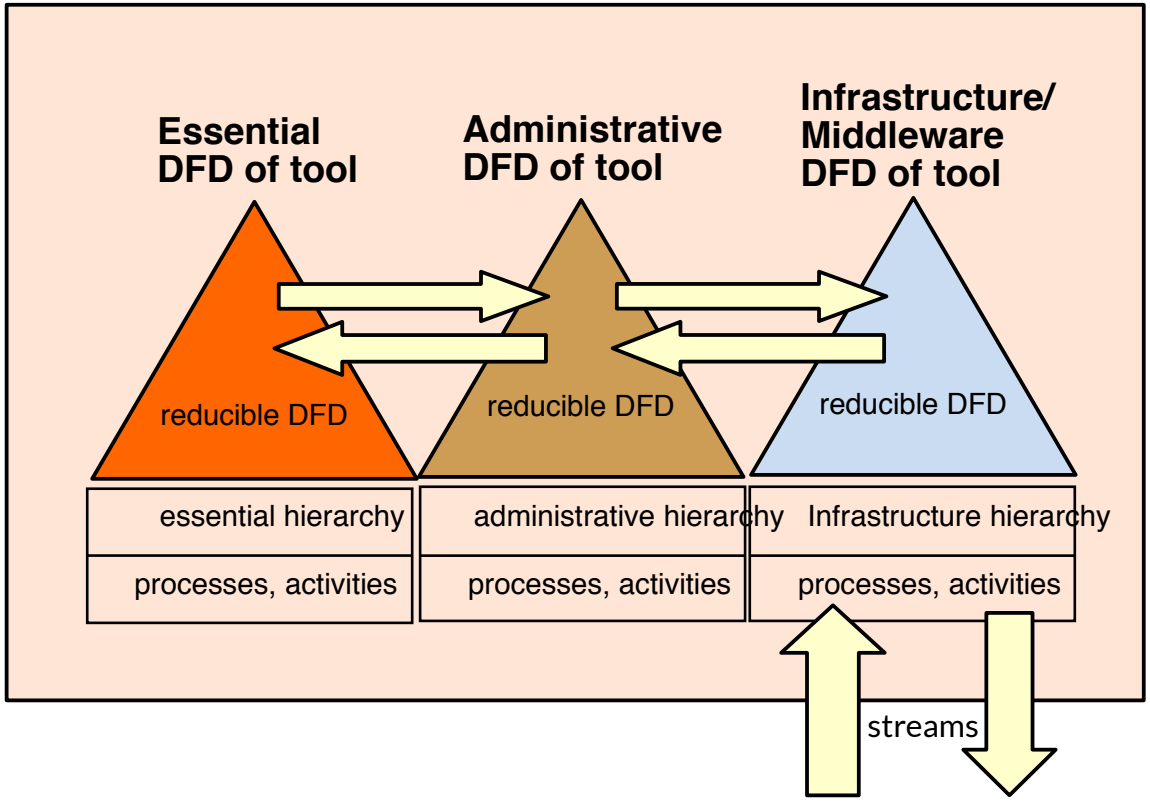


Quelle: Raasch, J.: Systementwicklung mit Strukturierten Methoden; Hanser Verlag (3.Aufl.) München 1993

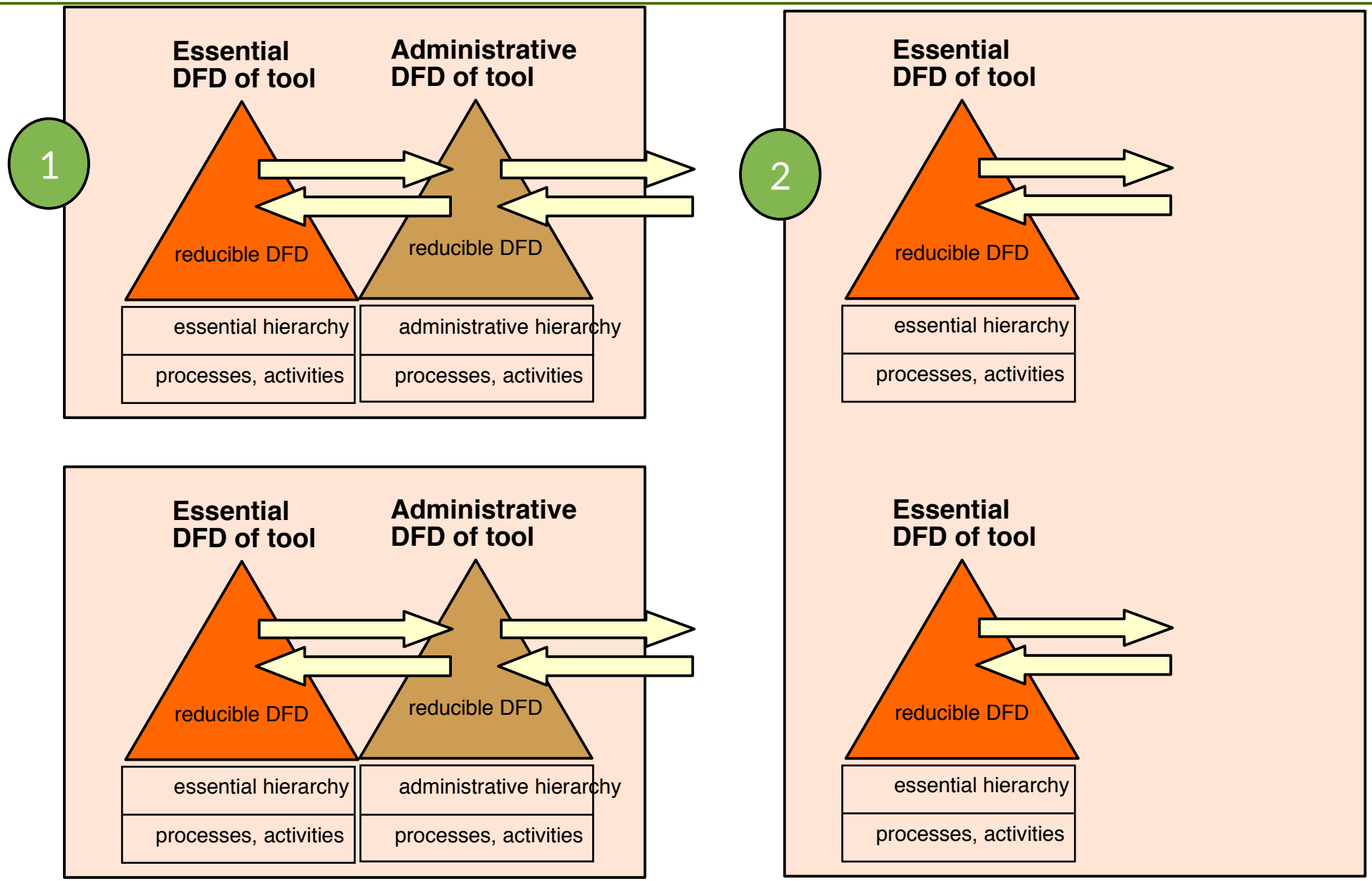
29.7 Composition of Stream-Based Tools

Process for Composition of Stream-Based Tools

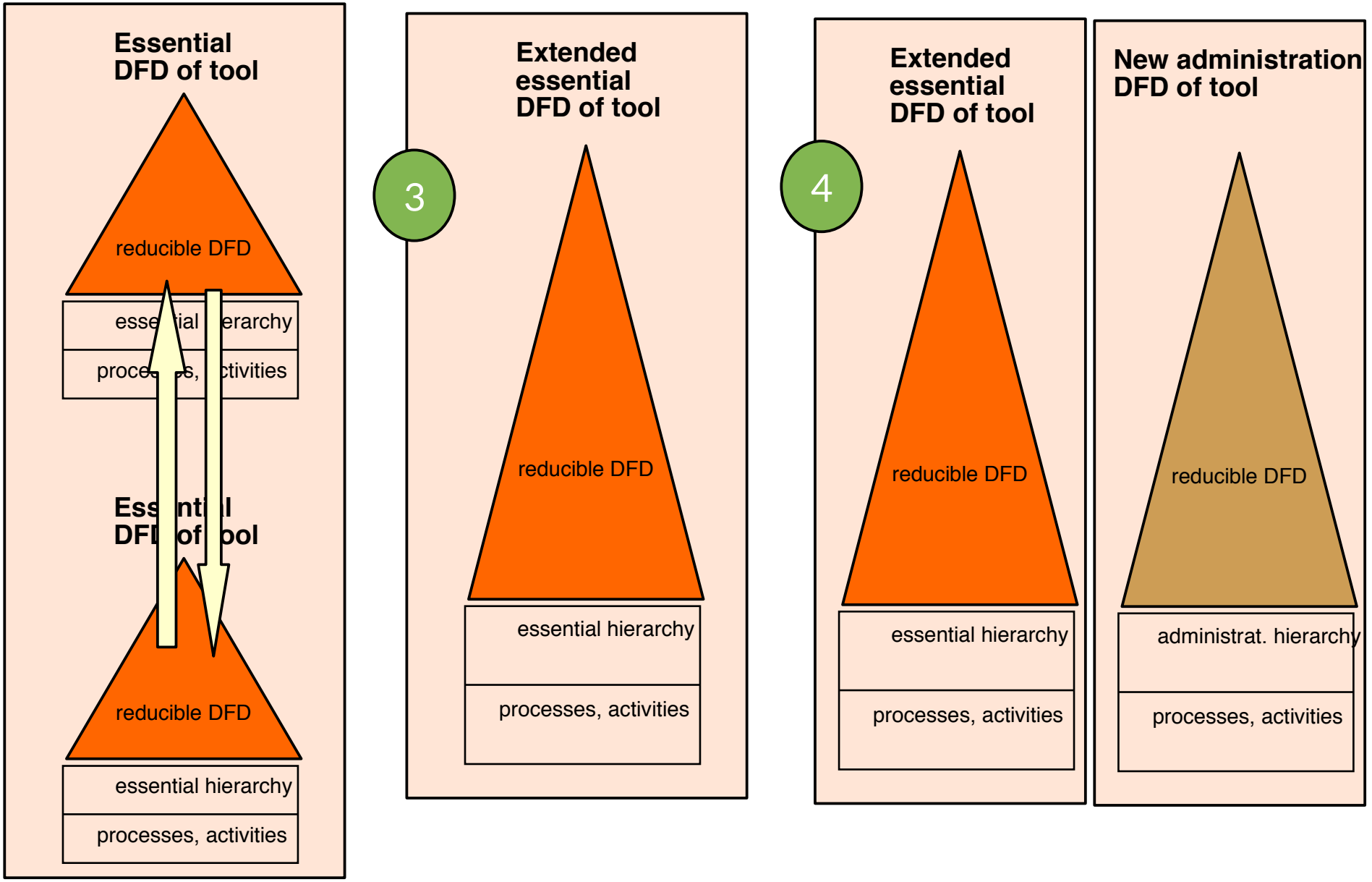
1) Strip the DFD: Strip Essence of Administration and Infrastructure:
    1) remove parser, printer, GUI, etc.
2) Compose the essential DFD of the tools
    - Extend and merge streams with the same schema (respect typing)
    - Extend core tools by asynchronous merge of output streams
    - Extend core tools by synchronous merge of output streams
    - Use aspect-oriented extension with cross-cut-graphs
3) Add Administration
4) Add Infrastructure to the composed DFD



1) Strip Infrastructure 2) Strip Administration



3) Extend Essence 4) Add Administration



5) Add New Infrastructure