Design by Transformation: Encoding Domain Knowledge to Derive Optimized Program Architectures

Rui C. Gonçalves
Universidade do Minho

Universidad Jaume I, November 2012

Context and Motivation

- Complexity of hardware platforms is increasing
  - Burden of improving software has moved from hardware manufacturers to developers
- Development of efficient software is a complex task
  - Requires deep knowledge of the application domain **and** target hardware platform
  - Tedious, time-consuming, and error-prone task
  - Non-experts cannot reproduce, or even understand the development process
  - Knowledge used to optimize programs is not reusable

Design by Transformation (DxT)

- Encode domain knowledge as transformations
  - Transform models to map high-level specifications to efficient implementations
  - Make domain knowledge reusable
  - Enable automation
- Specify interfaces of domain operations and their implementations
- Specify domain-specific optimizations
- High-level representation of knowledge that makes it more understandable by non-experts
- Requires regular and mature domains of application

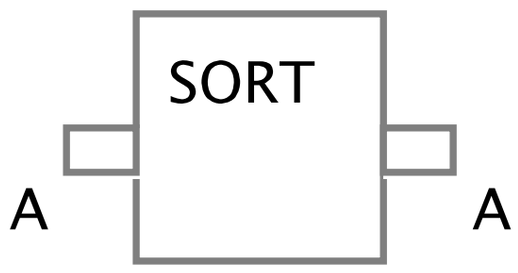
ReFIO Tool

- MDE tool for modeling application domains and synthesize efficient program architectures
- Modeling application domains
  - Specify interfaces, algorithms (pipe-and-filter implementations), primitives (code implementations)
  - Specify interfaces, primitives and algorithms *propagation functions* (used to model properties propagation functions, preconditions)
- Synthesize efficient program architectures
  - Refine and optimize an initial high-level architecture, to obtain an efficient architecture
  - Obtain the properties of the architecture (e.g., cost) to evaluate its quality

Modeling the Domain

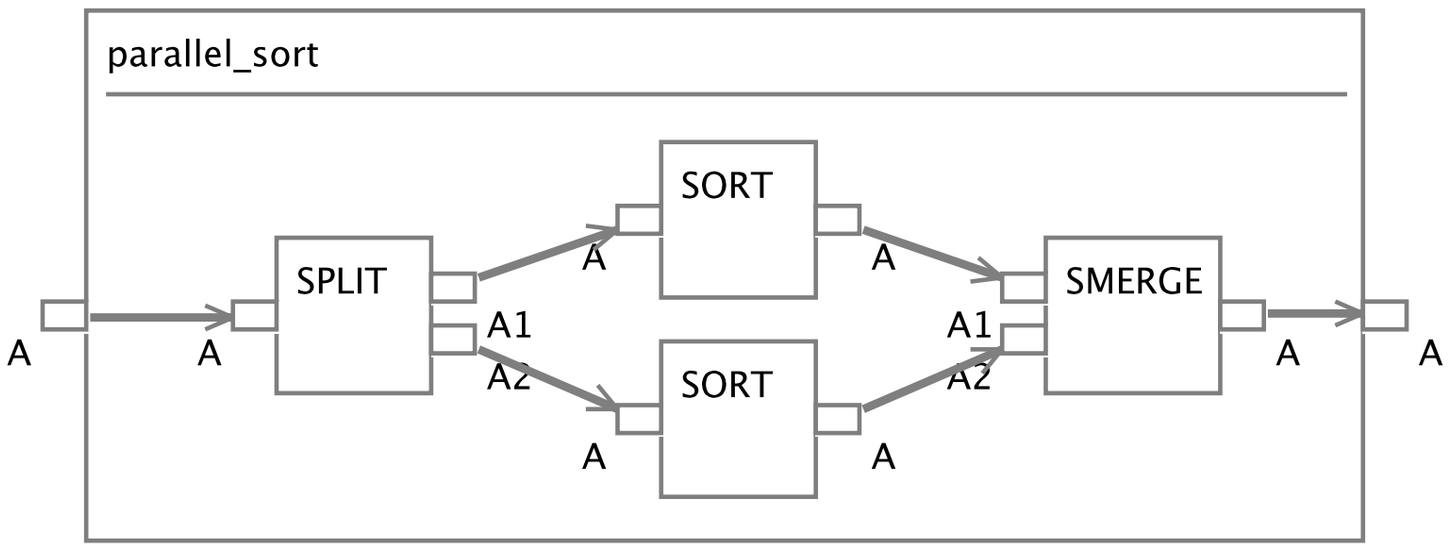
Interfaces

- To specify the interface for an operation, we create an *interface* box
- Interface boxes have
  - Name, input/output ports, and additional parameters
    - Ports have name and data type
    - Additional parameters are inputs that are not shown as ports
  - Properties propagation functions



Algorithms

- Specify how an interface can be implemented
  - Composes interfaces, using *connectors* to create a pipe-and-filter graph
- Algorithm boxes have
  - Name, input/output ports, and additional parameters
  - Properties propagation functions

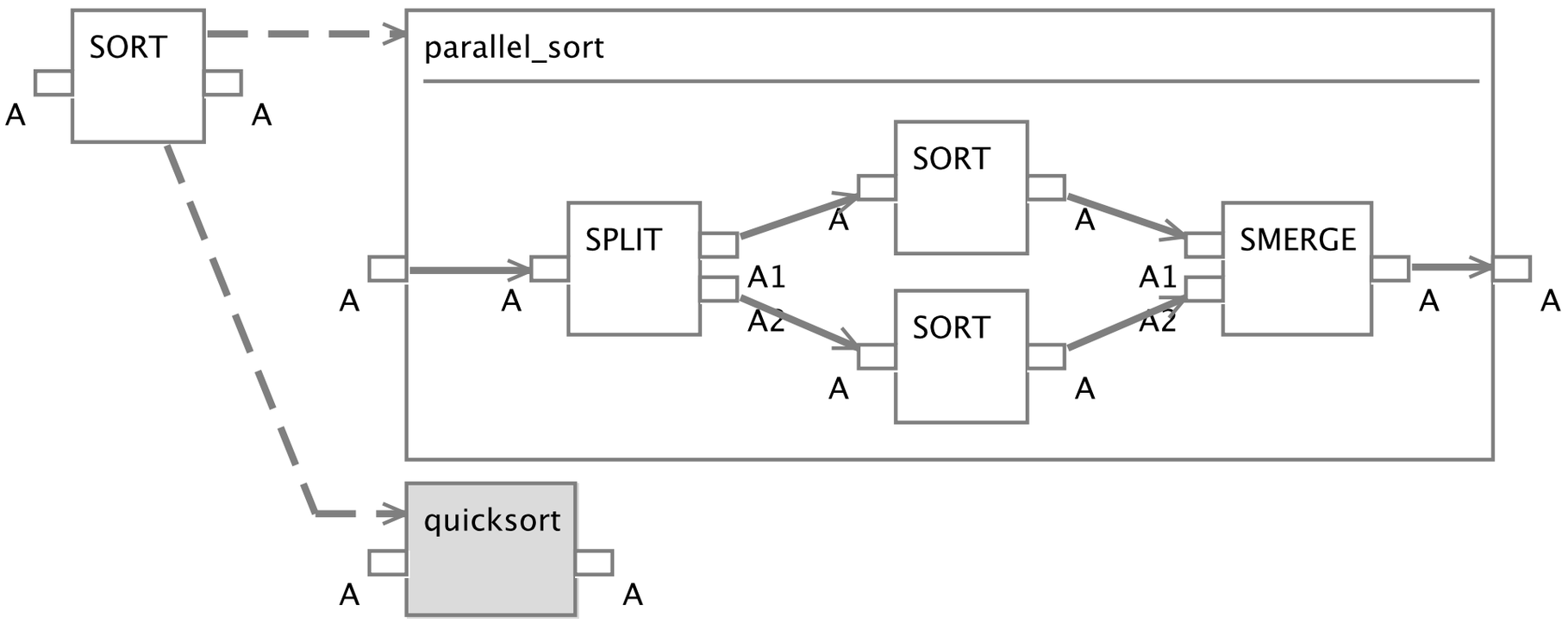


Primitives

- Specify the code implementations available for an interface
- Primitive boxes have
  - Name, input/output ports, and additional parameters
  - Properties propagation functions

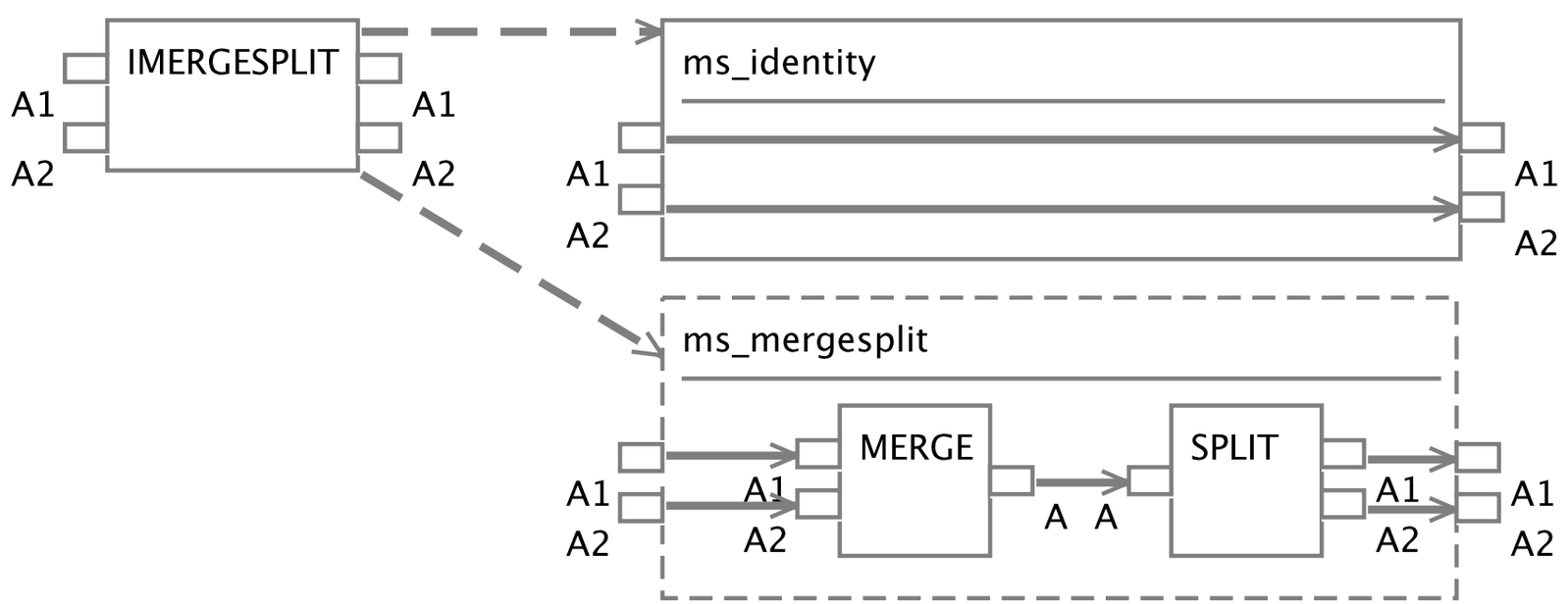
DxT Domain Model

- Pairs interfaces with their implementations
  - An interface may be implemented by an algorithm or a primitive
- Pairs form rewrite rules, that define valid replacements for interfaces (module preconditions)



DxT Domain Models

- Rewrite rules also specify how to optimize inefficient compositions of boxes



Additional Parameters

- Some inputs represent data that flows through the diagram boxes (e.g. streams in databases, matrices in BLAS)
  - Called *Essential Parameters*
- Other inputs are just constants, or copied from parent box (e.g. attribute used as key when sorting a stream, transposition attributes in BLAS)
  - Called *Additional Parameters*
  - Not shown in graphical representations

Propagation Functions

Propagation Functions

- Propagation Functions allow to associate behavior with boxes, to later animate models
  - Extract info (or *properties*) from architectures
  - Example: associating to each box how it affects the data sizes (a propagation function), we will be able to later predict the data sizes of an architecture outputs
  - Similar ideas may be used to compute costs or perform type checking, for example

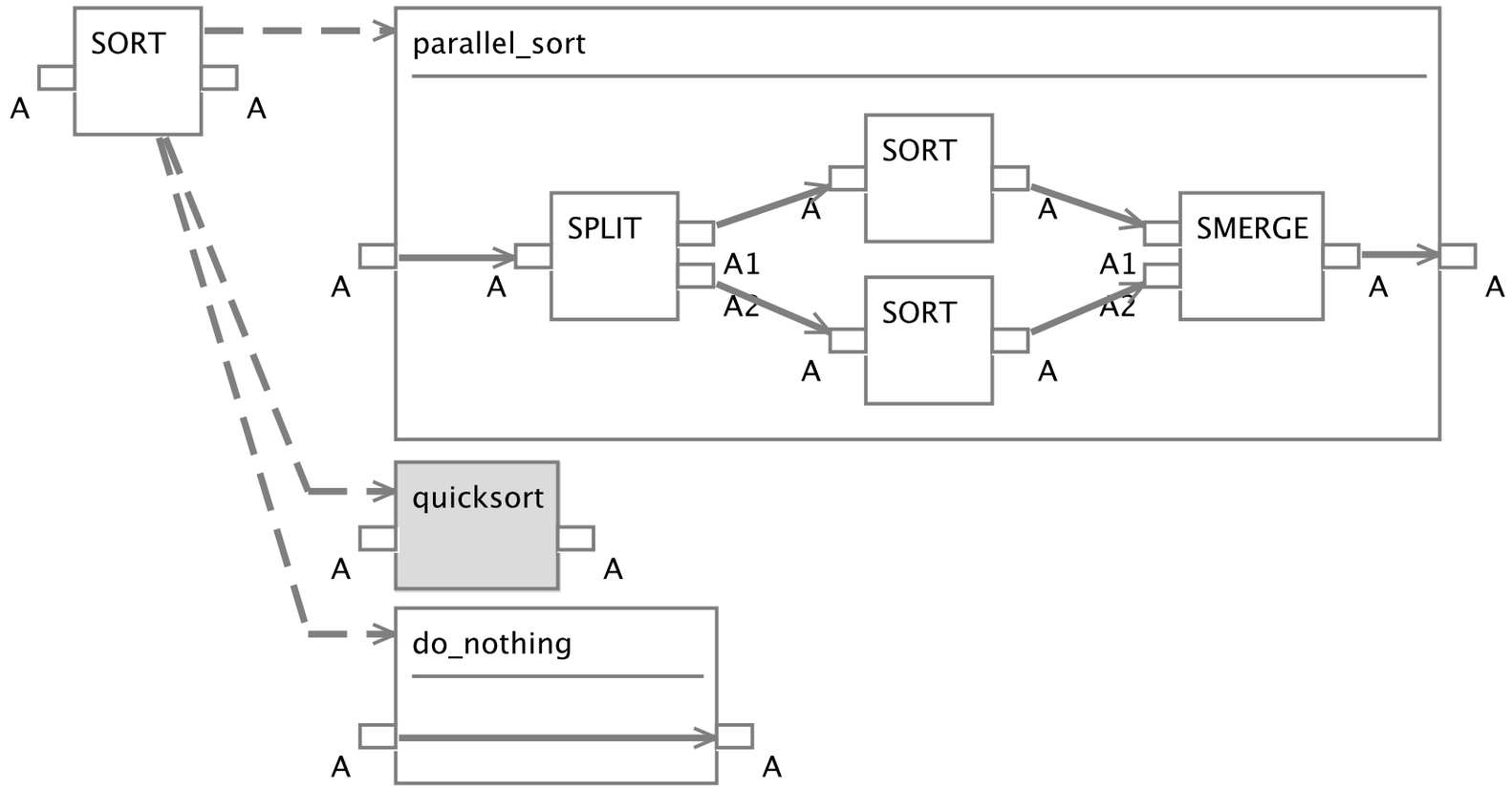
Propagation Functions

- Providing code for each box allows us to generate code for any architecture and execute it
- Providing properties propagation functions for each box allows us to animate an architecture and extract properties from it
- Several propagation functions may be executed in sequence
  - E.g.: to compute costs, we may first have to compute data sizes
- Specified in Java

Preconditions

- Interfaces, primitives and algorithms have preconditions
- Preconditions restrict the architectures where a box can be used
  - Boolean-valued function of properties of inputs and additional parameters
- Preconditions are defined using propagation functions

Preconditions



- do_nothing is a valid implementation for SORT when the input stream is already sorted

Propagation Functions and Preconditions

- SORT: output is always sorted

```
public void compute() {
     setOutputProperty("A", "IsSorted", true);
}
```

- SPLIT: outputs are sorted iff input is sorted

```
public void compute() {
     Boolean inA = getInputProperty("A", "IsSorted");
     setOutputProperty("A1", "IsSorted", inA);
     setOutputProperty("A2", "IsSorted", inA);
}
```

- do_nothing (precondition): if input is not sorted *throw* an error

```
public void compute() {
     Boolean inA = getInputProperty("A", "IsSorted");
     if(!inA) addError("Input is not sorted!");
}
```

Program Architecture Transformation/Synthesis

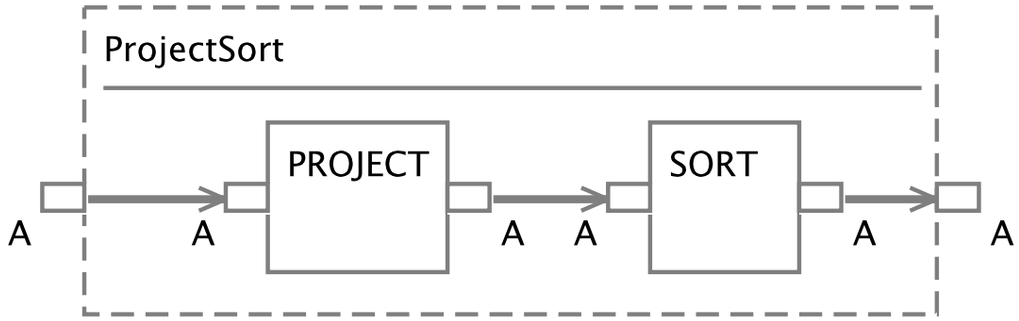
Initial Architecture

- Initial architecture is represented by an *architecture* box (similar to an algorithm box), which specifies an implementation of the program to optimize
- To use properties, we must provide the properties of program inputs/outputs, that ReFIO will propagate
  - Properties may be just strings denoting variables

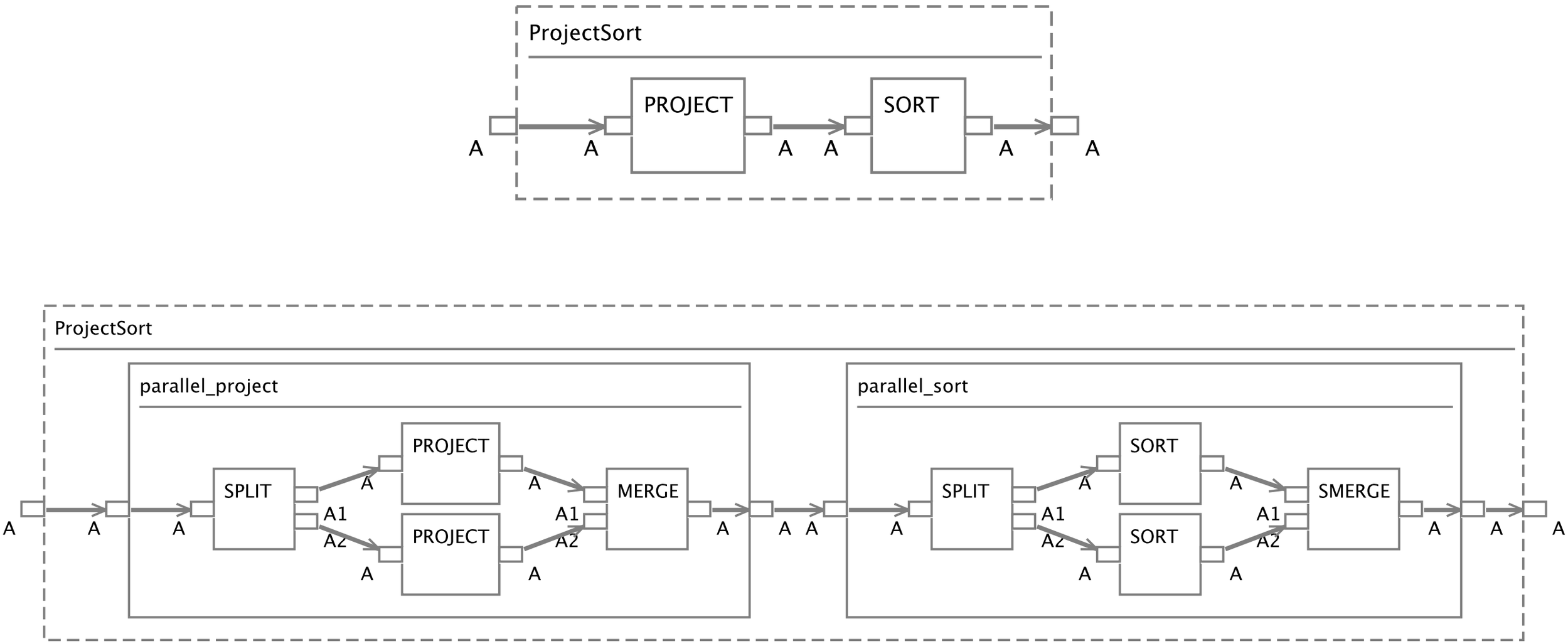
Refinements

- Replace an interface with one of its implementations
- Implementation preconditions (if any provided) are tested to build the list of possible implementations
- Flatten transformation may then be used to remove the algorithm boundaries

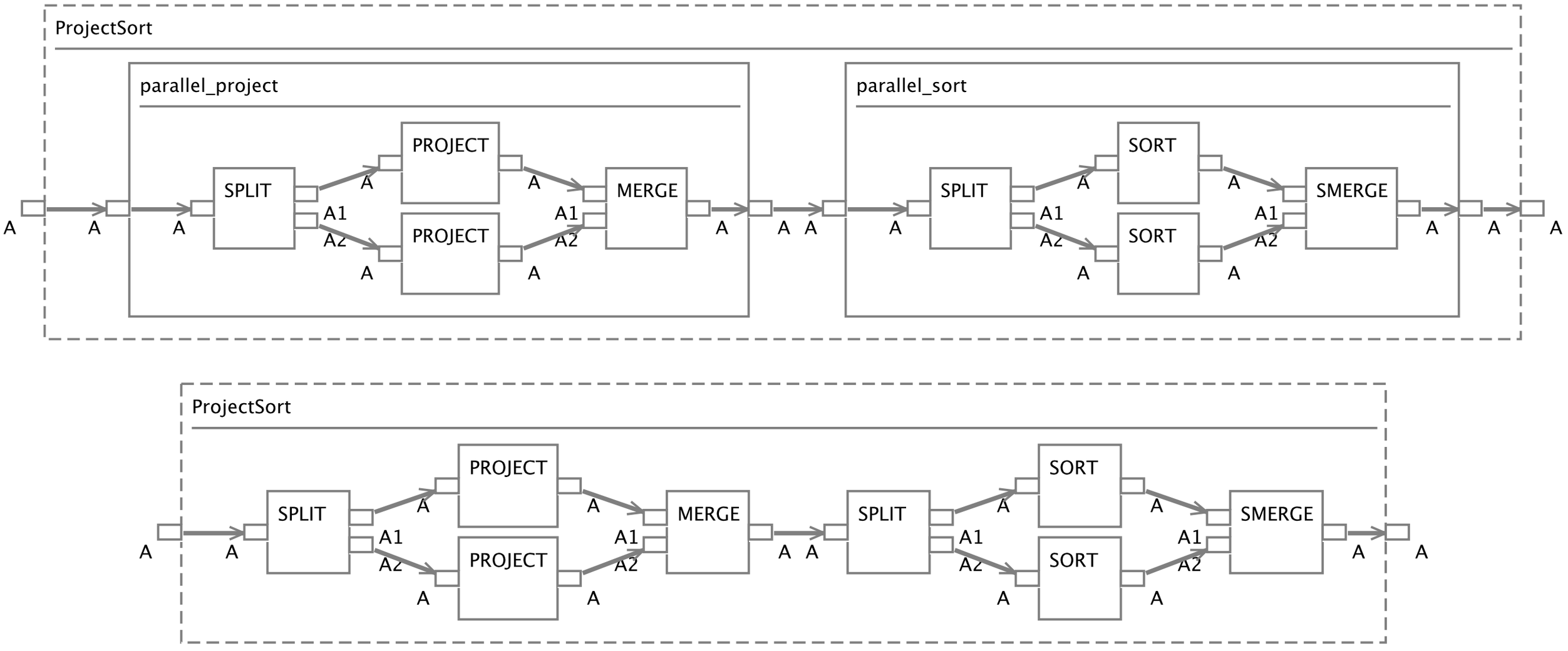
Project Sort Architecture



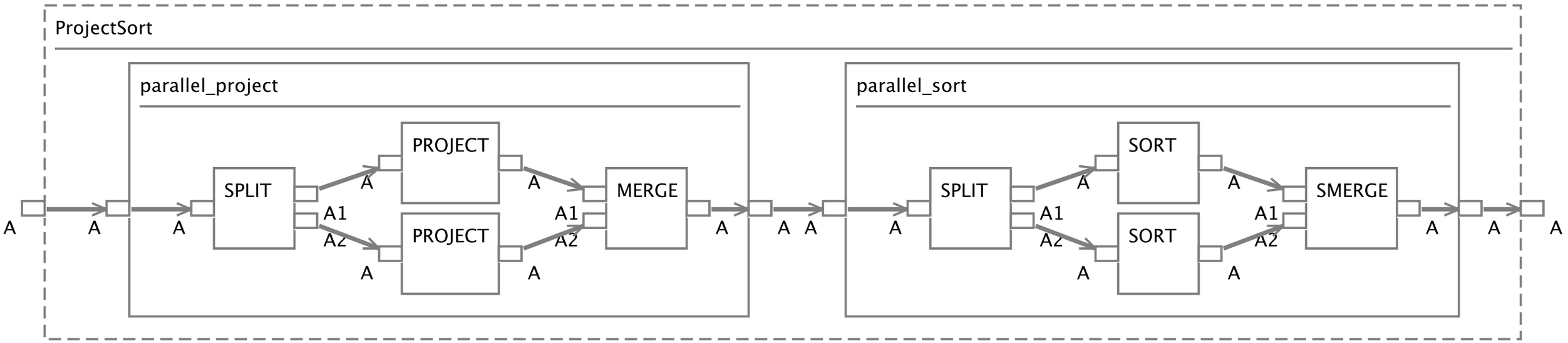
Project Sort Architecture

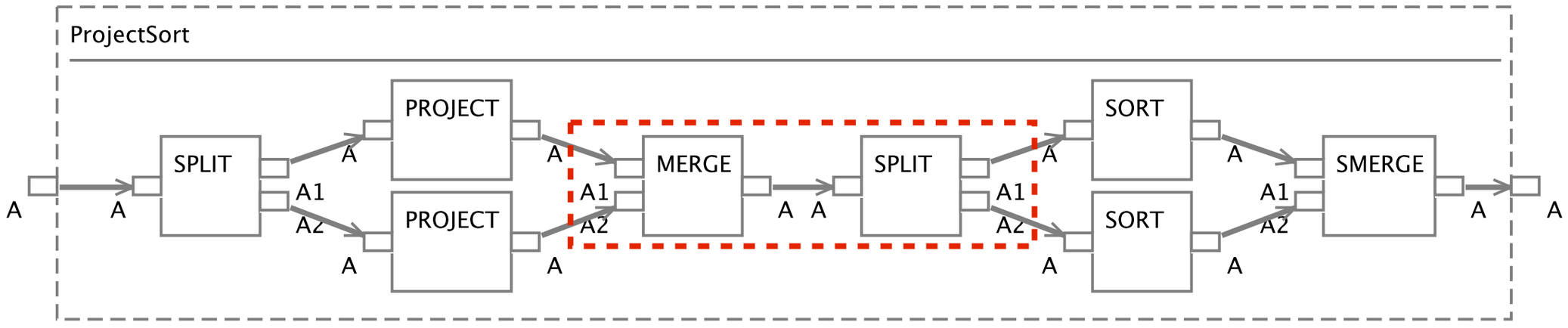


Project Sort Architecture



Project Sort Architecture



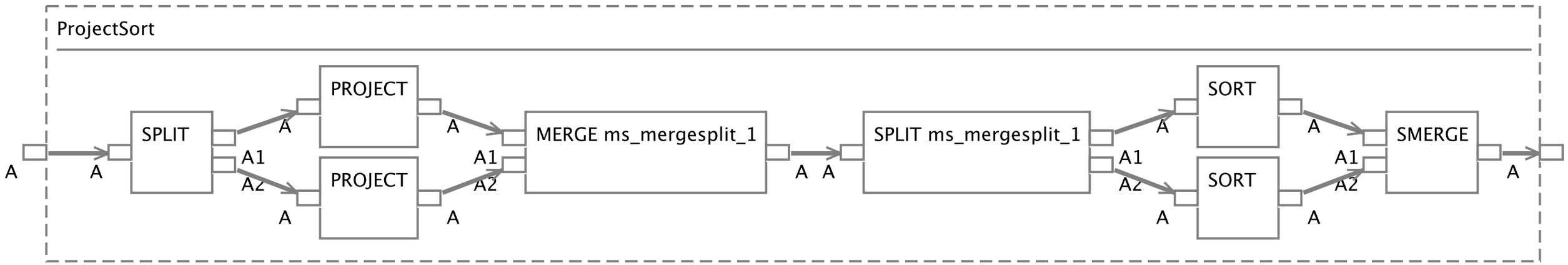


- The removal of the modular boundaries exposes inefficiencies

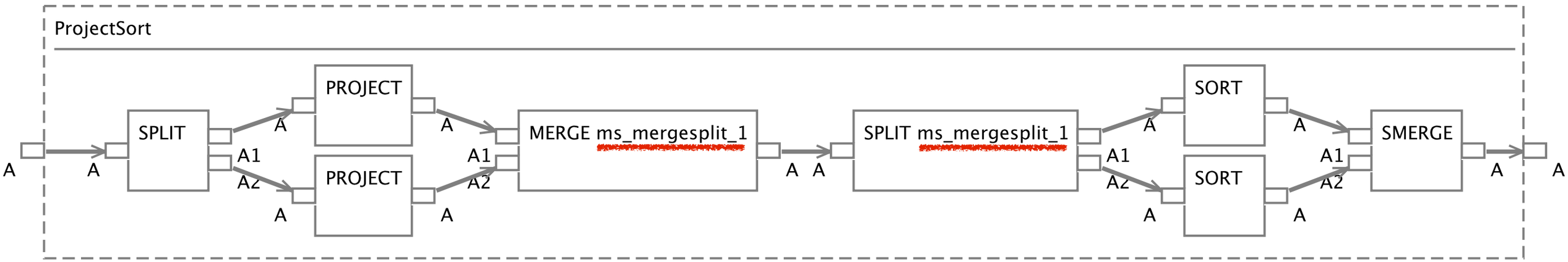
Optimizing Abstractions

- Aggregates a set of boxes, that are composed as an algorithm
- Preconditions are tested to ensure that architecture remains correct after the transformation
- Operation *Find Patterns* automatically identifies the sets of boxes that can be abstracted

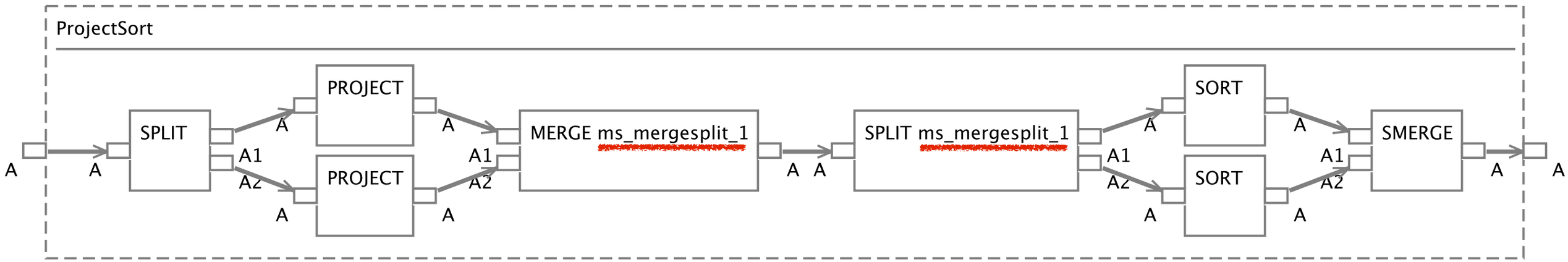
Project Sort Architecture

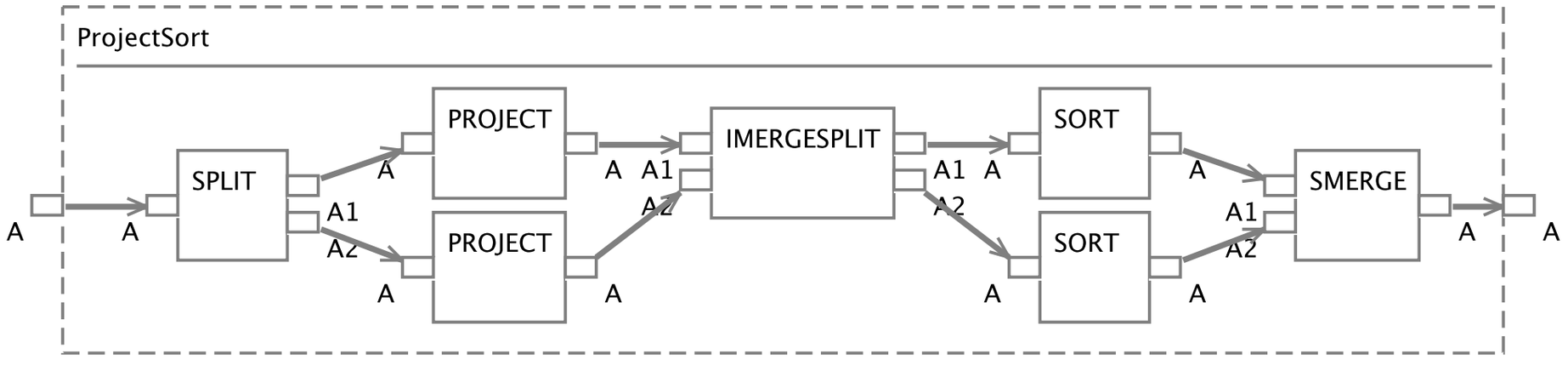


Project Sort Architecture

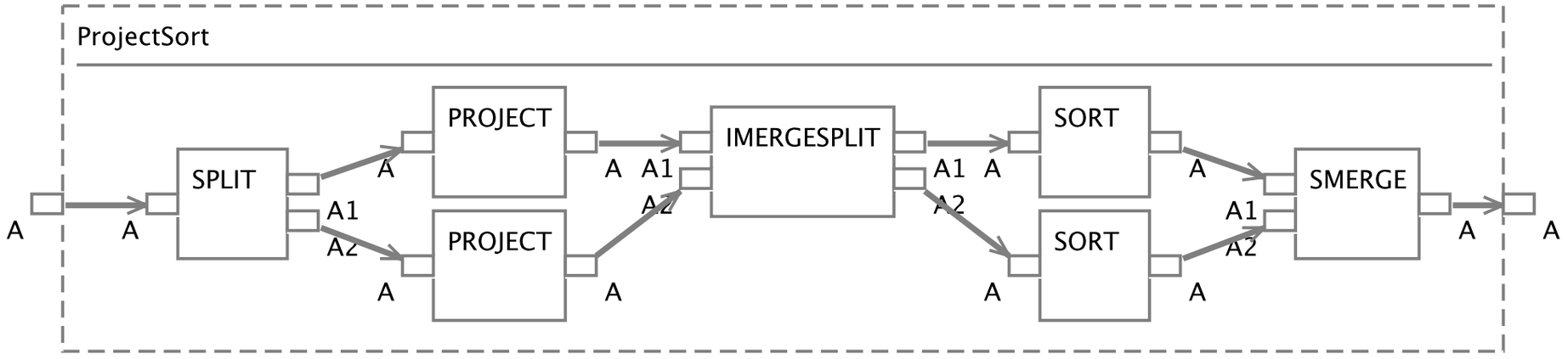


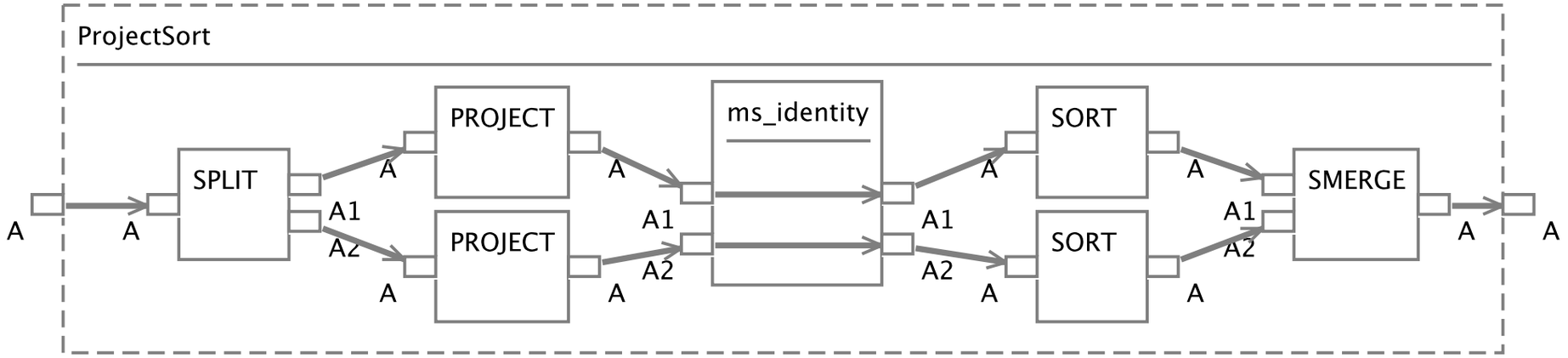
Project Sort Architecture



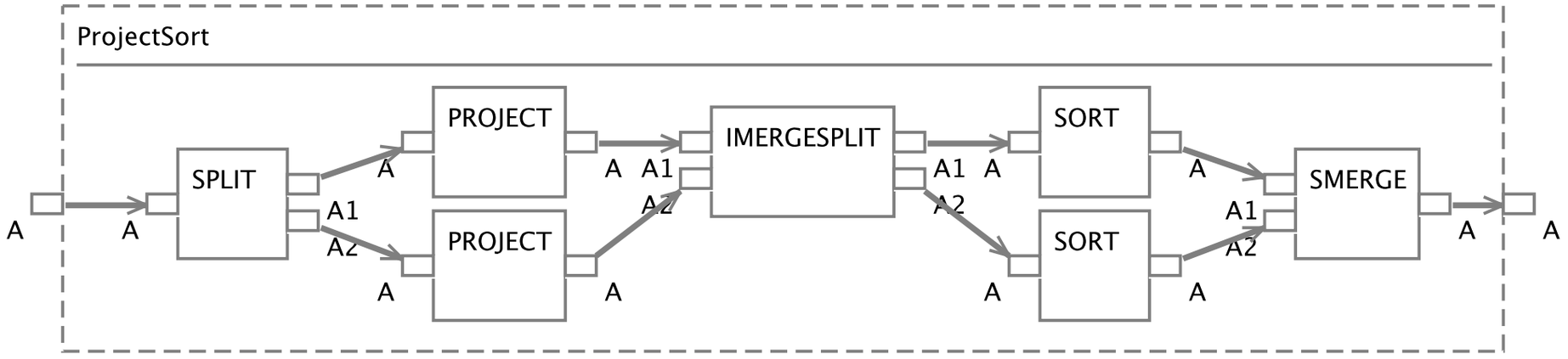


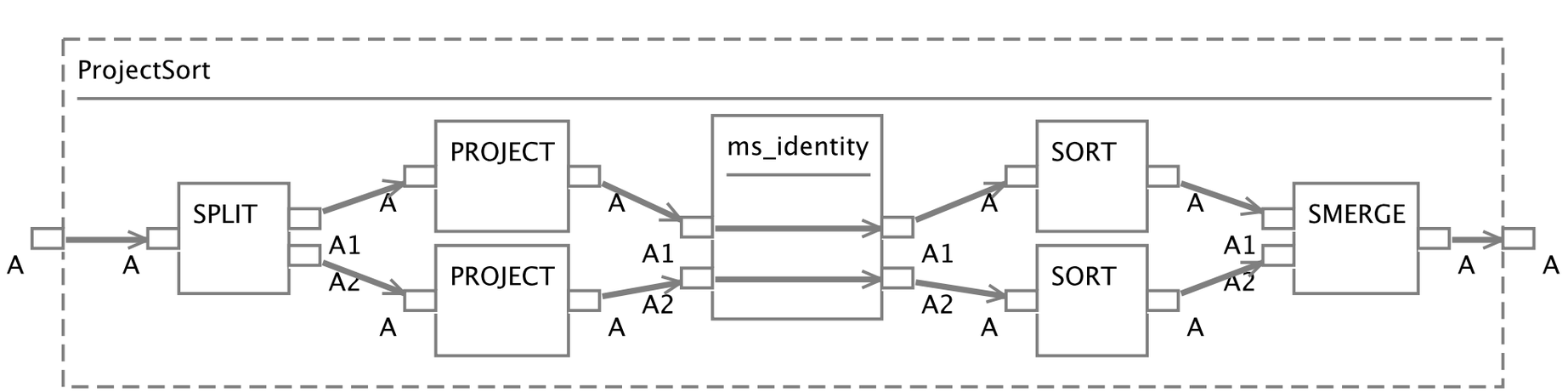
Project Sort Architecture

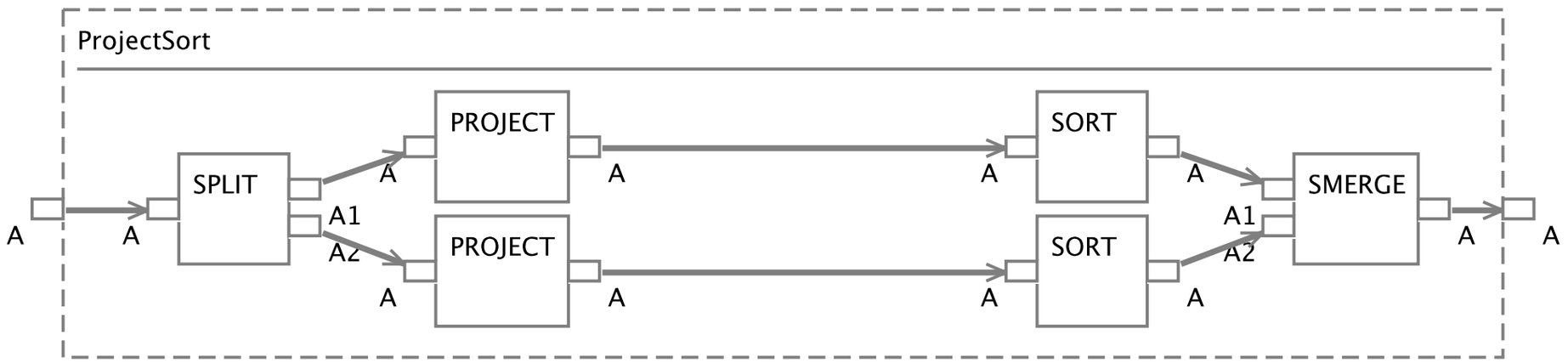




Project Sort Architecture







Running Propagation Functions

- At any point during the synthesis process, we can run propagation functions
  - Just select the architecture, and enter the propagation functions to run
- Allows us to keep track of the evolution of costs (or other quality attributes) of the architectures

Other Features of ReFlO

- Export DxT Domain Models to HTML documentation
  - Requires the user to document the model elements (boxes, ports)
- Export DxT Domain Models to external tool (DxTer), that automates the search for the best implementation
  - Some code needs to be manually provided as *model annotations* (*e.g.*, preconditions)

Summary

- Approach/framework to encode (reusable) domain knowledge using a pipe-and-filter notation
  - Operations and their implementations
  - Optimizations
- Make domain knowledge accessible by non-experts
- Transformation system for pipe-and-filter graphs to synthesize program architectures
  - Incrementally build optimized architectures
  - Enable automation
- Animate architectures, and compute properties about them

Thank You!

- Some references:
  - http://www.cs.utexas.edu/users/schwartz/DxT/
  - Taylor Riché, Don Batory, Rui Gonçalves, Bryan Marker. *Architecture Design by Transformation*. UT-CS Technical Report TR-10-39, 2010.
  - Taylor Riché, Rui Gonçalves, Bryan Marker, Don Batory. *Pushouts in Software Architecture Design*. Generative Programming and Component Engineering 2012.
  - Bryan Marker, Jack Poulson, Don Batory, and Robert van de Geign. *Designing Linear Algebra Algorithms by Transformation: Mechanizing the Expert Developer*. International Workshop on Automatic Performance Tuning 2012.

- **Questions?**