X-CSR: Dataflow Optimization for Distributed XML Process Pipelines

Daniel Zinn,
Shawn Bowers, Timothy McPhillips, Bertram Ludäscher

ICDE 2009

Department of Computer Science
University of California at Davis

April 1st, 2009

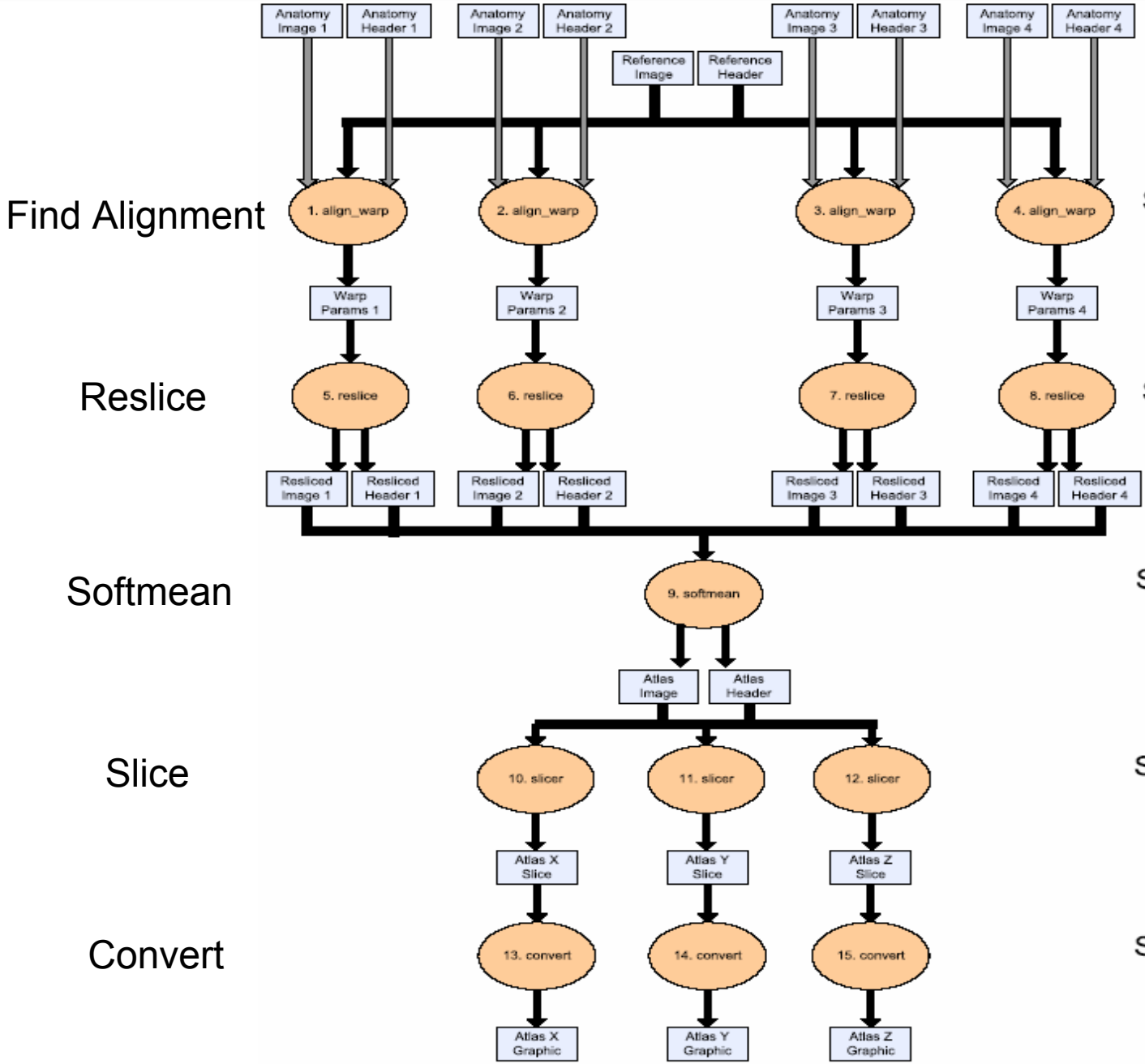
Outline

- **XML Processing Pipelines (∆-XML)**
- **X-CSR: Dataflow-Optimization for ∆-XML**

Motivation: Scientific Workflows

- Phylogenetics
  - Gather DNA sequence data
  - Align sequences
  - Reconstruct phylogeny using multiple alignments – often obtaining a large number of trees
  - Compute consensus (or otherwise estimate the reliable components of the evolutionary history)
  - Perform post-tree analyses

- **Functional Magnetic Resonance Imaging** (fMRI)
  - Brain scans while performing activities
  - Voxel-picture post-processing (command-line tools)


Anatomy Image 1
Anatomy Header 1
Anatomy Image 2
Anatomy Header 2
Anatomy Image 3
Anatomy Header 3
Anatomy Image 4
Anatomy Header 4
Reference Image
Reference Header
Find Alignment
1. align_warp
2. align_warp
3. align_warp
4. align_warp
Stage 1
Warp Params 1
Warp Params 2
Warp Params 3
Warp Params 4
Reslice
5. reslice
6. reslice
7. reslice
8. reslice
Stage 2
Resliced Image 1
Resliced Header 1
Resliced Image 2
Resliced Header 2
Resliced Image 3
Resliced Header 3
Resliced Image 4
Resliced Header 4
Softmean
9. softmean
Stage 3
Atlas Image
Atlas Header
Slice
10. slicer
11. slicer
12. slicer
Stage 4
Atlas X Slice
Atlas Y Slice
Atlas Z Slice
Convert
13. convert
14. convert
15. convert
Stage 5
Atlas X Graphic
Atlas Y Graphic
Atlas Z Graphic

Motivation for XML Processing Pipelines

Combining existing software to build complex scientific analysis systems while saving machine cycles and human cycles

- To save developer‘s time, systems should be…
  - Easy to build
  - Easy to evolve and maintain
  - Easy to share and re-use
  - Self-documenting
- To save execution time,
  - Systems should efficiently execute

Setting Characteristics

Software components are heterogeneous:

- Languages: Matlab, C, Java, Pascal, R, ...
- Invocation: Portals, Web-Services, local executables, libraries (BioPerl)

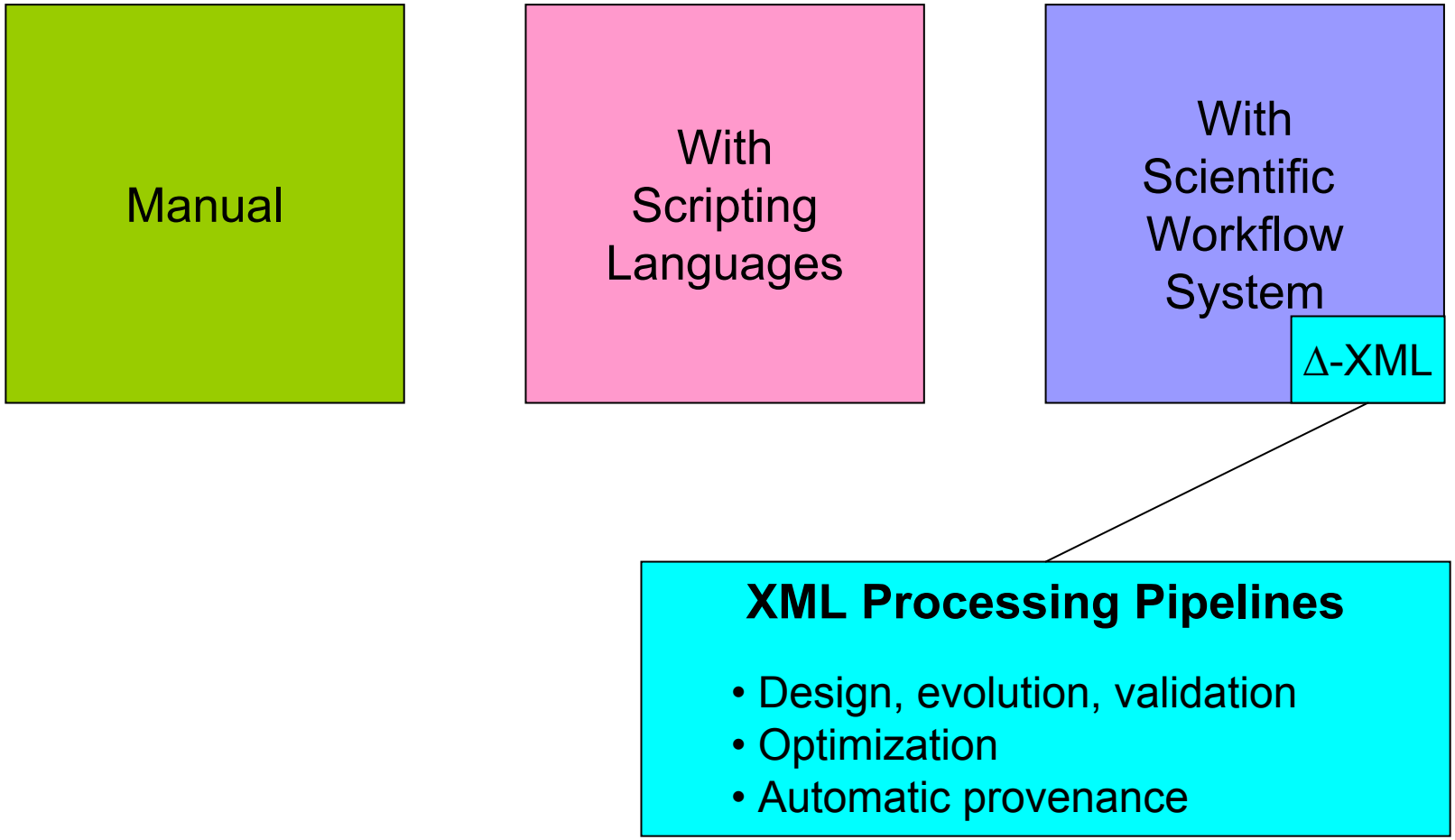
Process is exploratory:

- Refining steps and their parameter after new insights

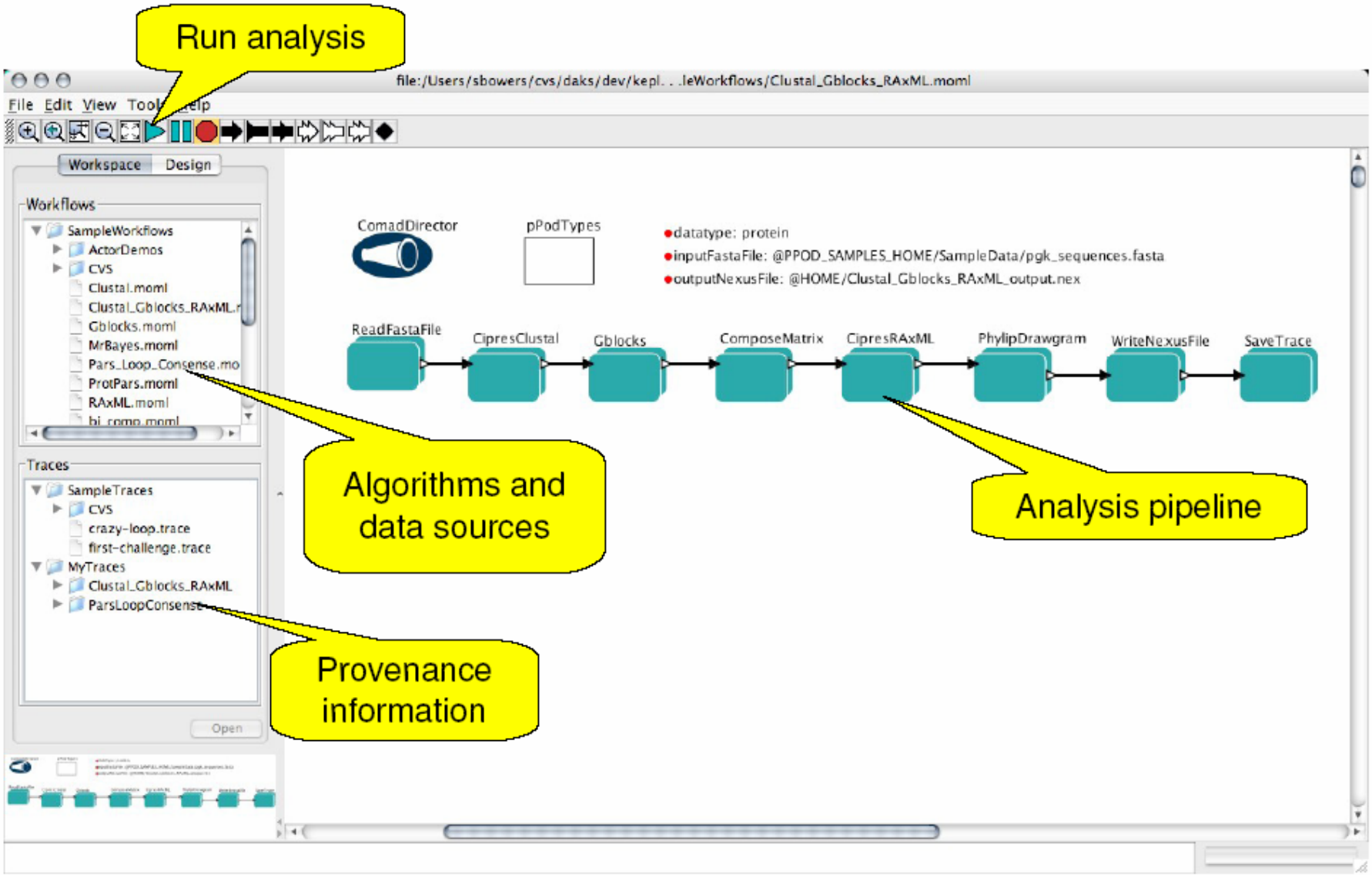
Workflows are data- and/or compute-intensive:

- Phylogenetics (NP-hard problems, exponential convergence, …)
- Astronomy (LSST project: nightly data of 15TB)

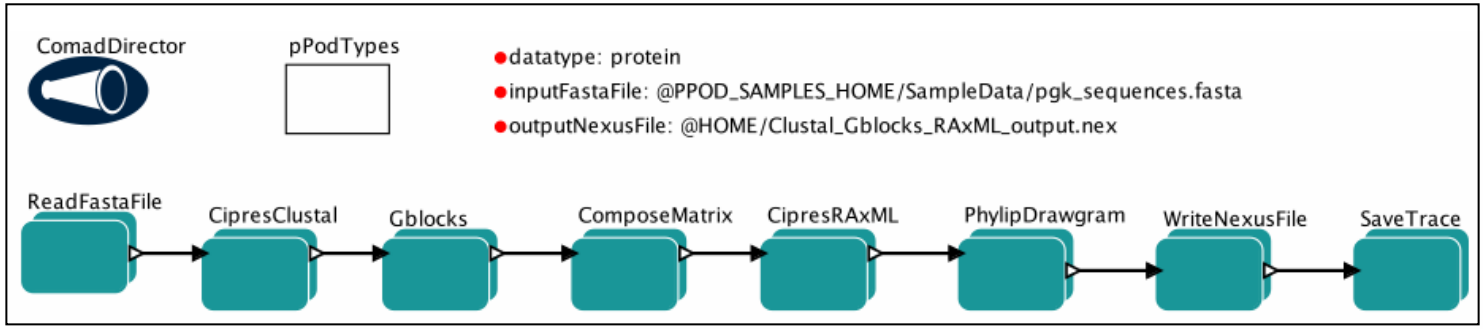
Current Approaches



Approach: Scientific Workflow Systems



Workflow 101



Actors

- Components with ports

Channels with Types

- Explicit communication between actors

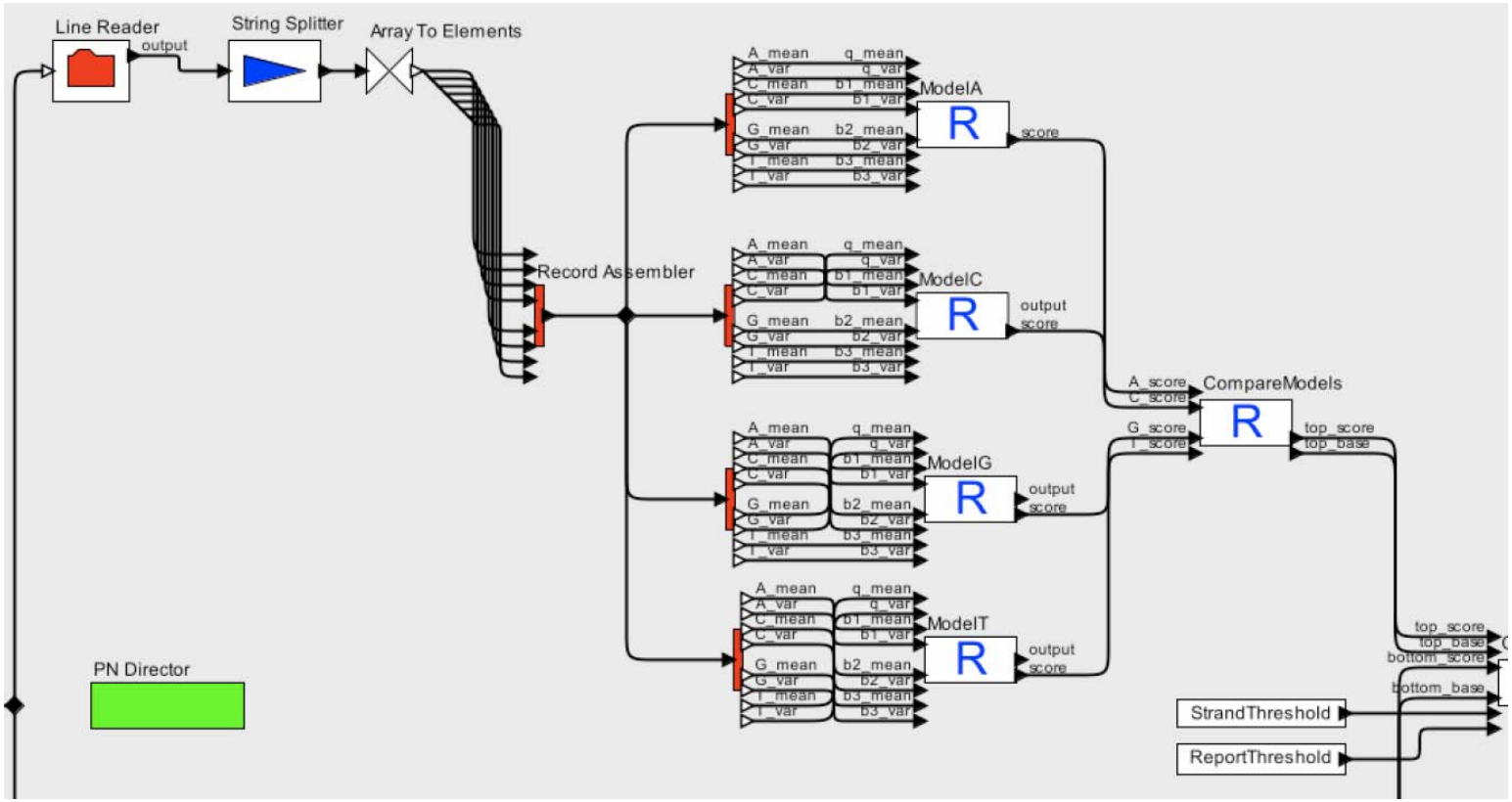
Tokens

- Data entities sent on channels

Model of Computation (MoC)

- Determines how workflow graph is executed

But…Problems with Nested Data



Problems

- Problems:
  a) **Data associations**: workflow graph exposes details of low-level data "massaging"
  b) **Complex-control flow**: workflow wiring can be overly complex to deal with exceptions, loops, etc.
- → Overly complex workflow graphs
  - Tedious to develop
  - Difficult to maintain and evolve
  - Data massaging encoded into workflow graph
  - Impossible to feed with structurally different input data

Solutions

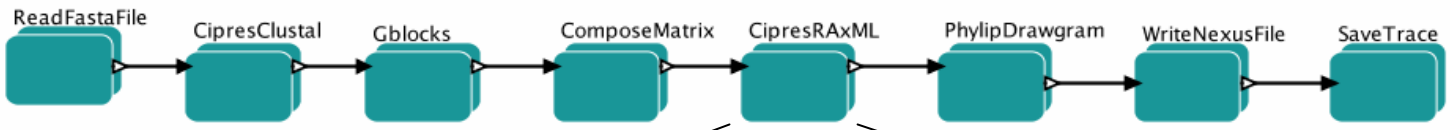
- Ideally, switch to a paradigm / MoC that …
  a) automatically handles complex data
  b) simplifies control-flow specification
- Solutions:
  a) Nested collections (e.g. a la XML)
  b) Virtual Assembly-Lines with actor configurations
     - Mostly linear sequence of actors
     - Every actor sees everything but only works on selected data

     ➔ Exception handling e.g. via “tagging” faulty parts that will be removed or replaced, handled downstream (or simply ignored)

     ➔ Increased flexibility for adding, removing, swapping actors

3-Layer Architecture of Δ-XML Pipelines

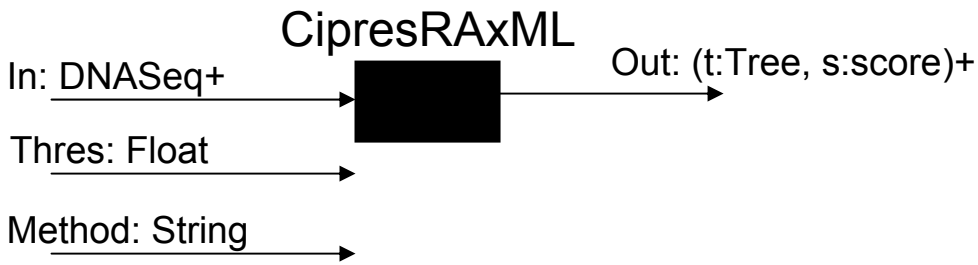
WF Graph



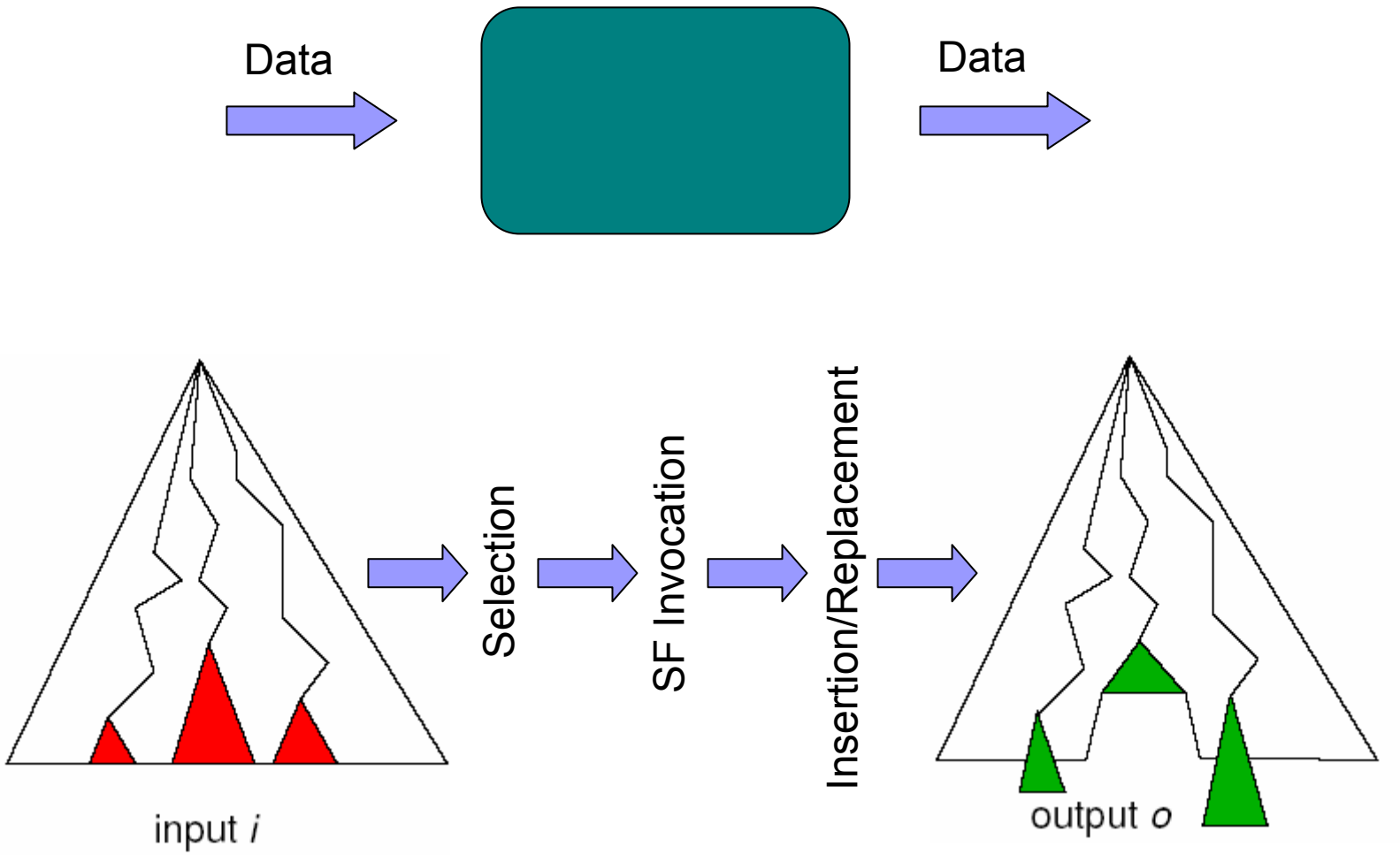
Configurations
(white-box)

- Access data in XML stream
- Call Scientific Functions (Services)
- Put results back into stream

Scientific Functions
(black-boxes)



Δ-XML Actor


Data
Data
Selection
SF Invocation
Insertion/Replacement
input *i*
output *o*

XML Processing Pipelines

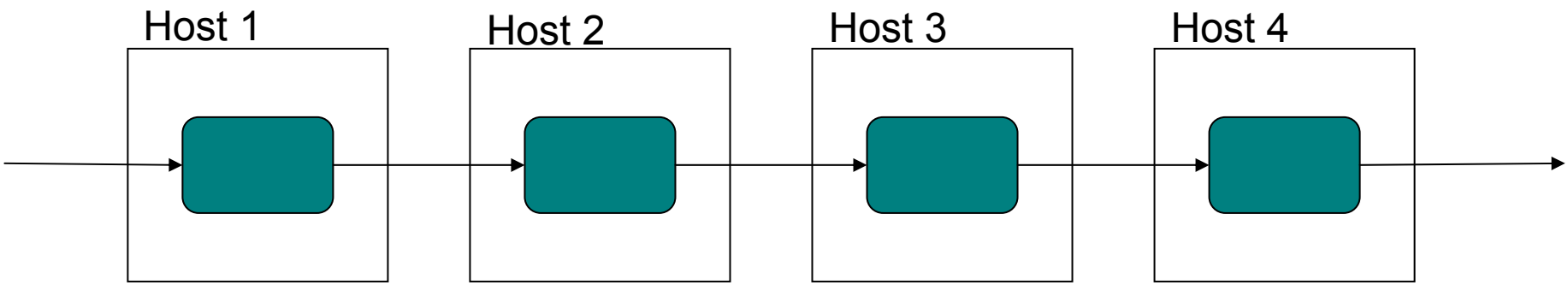
- ∆-XML = XML processing with
  - Sequence of updates instead of queries
  - Calls to external functions
  - Data-centric XML data

Outline

- **∆-XML: XML Processing Pipelines**
- **X-CSR: Dataflow-Optimization for ∆-XML**

Dataflow Optimization X-CSR

- ∆-XML execution scenario



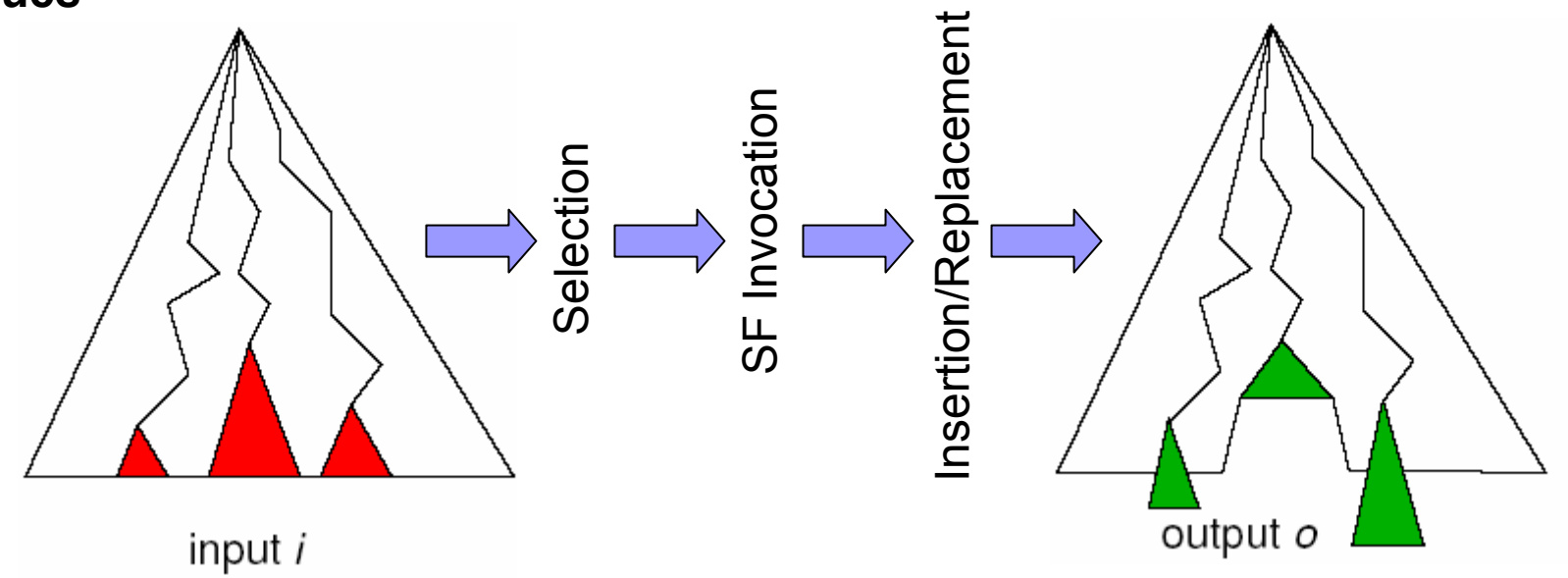
- ∆-XML advantages come with overhead:
  - Unnecessary data shipments (selective reads)
- X-CSR: Only ship ***selected*** data

Abstracting Actor Behavior

- Q1: What is *selected* by A?
- Q2: How does A change the stream?
- ➔ Signature of A

Schemas

Values

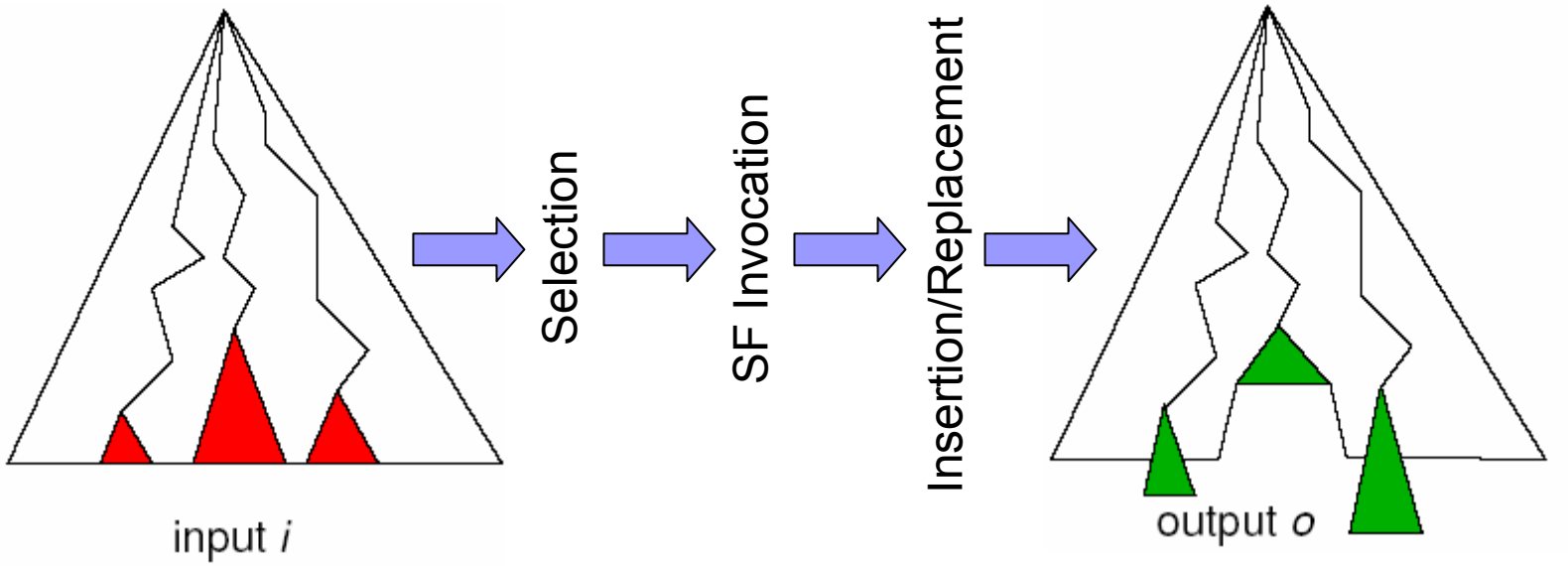


Abstracting Actor Behavior

Tree Grammar → Actor Signature → Tree Grammar

Schemas

Values



Tree Grammars

- Characterize sets of labeled trees / valid input instances for an actor
- Similar to DTDs, but can capture upper-context information
- Roughly eqivalent to XMLSchema

Tree Grammars

Formal Definition

- Set of *type names* $\mathbf{T}$
- Set of *labels* $\mathbf{L}$
- Set of *type declarations* of the form

$$A ::= \langle \mathtt{a} \rangle \, \mathcal{R}(\mathbf{T}) \qquad \text{, with } A \in \mathbf{T}, \langle \mathtt{a} \rangle \in \mathbf{L}$$

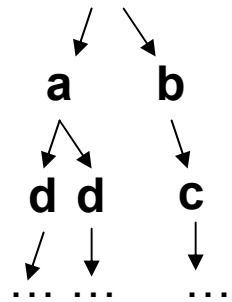
Example

$$\begin{aligned} \tau = \{ S &::= \langle \mathtt{s} \rangle \, (A \,|\, B)^*, \\ A &::= \langle \mathtt{a} \rangle \, D^+ \,|\, E^*, \\ B &::= \langle \mathtt{b} \rangle \, C, \\ C &::= \langle \mathtt{c} \rangle \, Z, \\ D &::= \langle \mathtt{d} \rangle \, F^* G, \ldots \} \end{aligned}$$

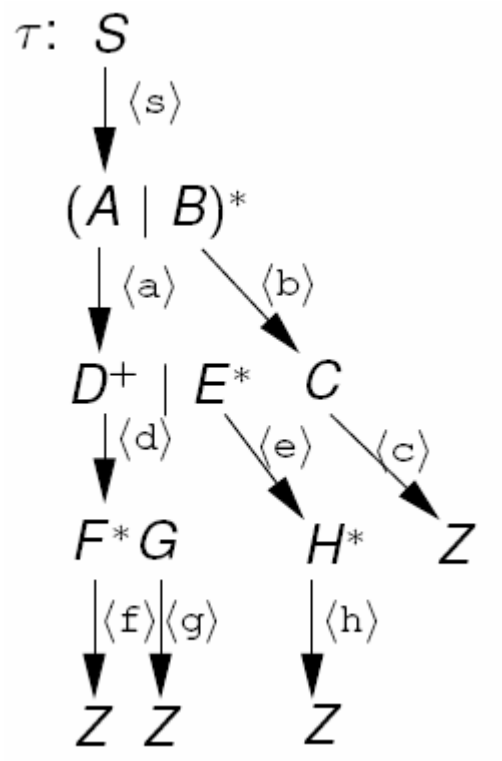
$i =$

```
<s>
  <a>
    <d>…</d>
    <d>…</d>
  </a>
  <b>
    <c>…</c>
  </b>
</s>
```

$i =$



Example Schema



$$\mathbf{T}_\tau = \{S, A, \ldots, Z\}$$
$$\mathbf{L}_\tau = \{\langle s\rangle, \langle a\rangle, \ldots, \langle h\rangle\}$$

- Use type Z to denote base data (holds the actual data)
- Non-ambiguous
- Non-recursive

Actor Signatures

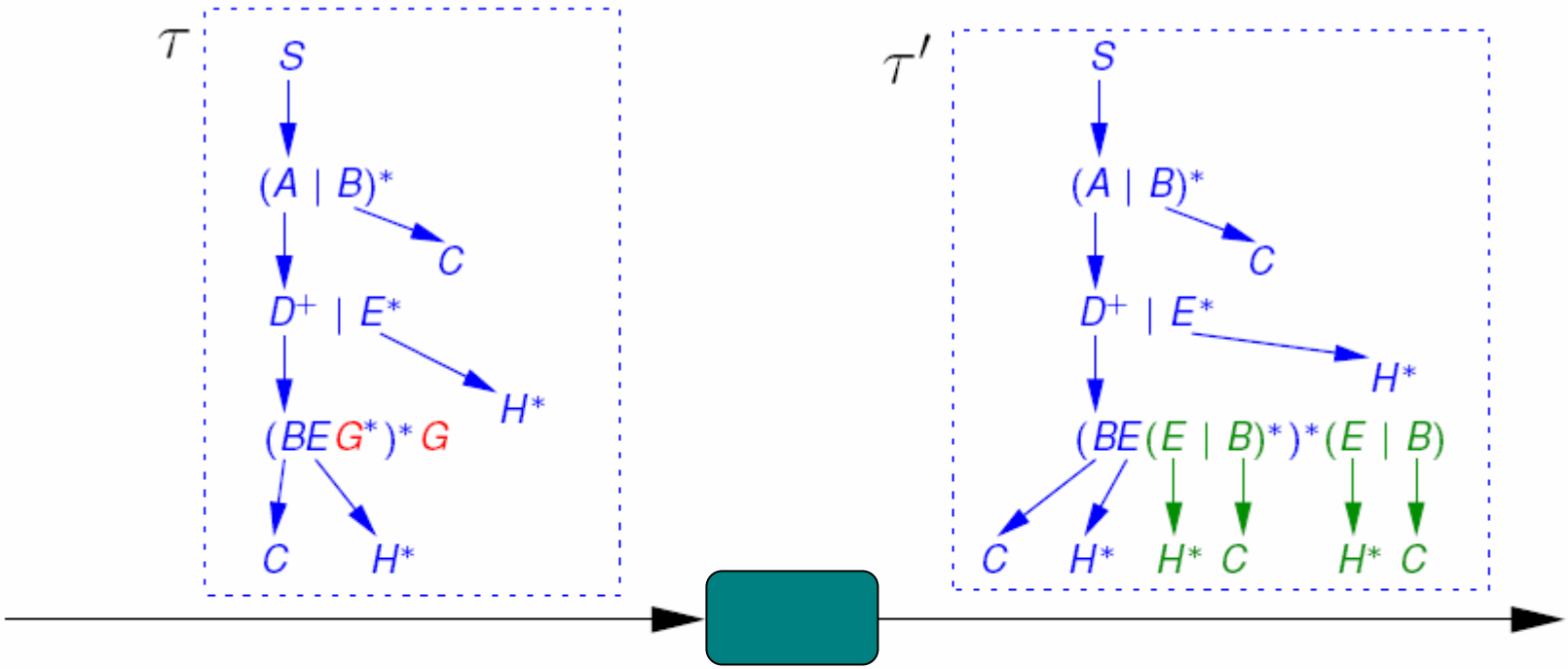
Signature captures …

- What the actor reads, and
- How stream is modified

Formal:

- $\Delta_A = \langle \sigma, \tau_\alpha, \tau_\omega \rangle$
- τ_α – input selection schema
- τ_ω – new output schema parts
- σ – set of match rules, each of the form $X \rightarrow \mathcal{R}$ with
  - $X \in \tau_\alpha$, and
  - $\mathcal{R}$ regexp. over types in $\tau_\alpha \cup \tau_\omega$

Example



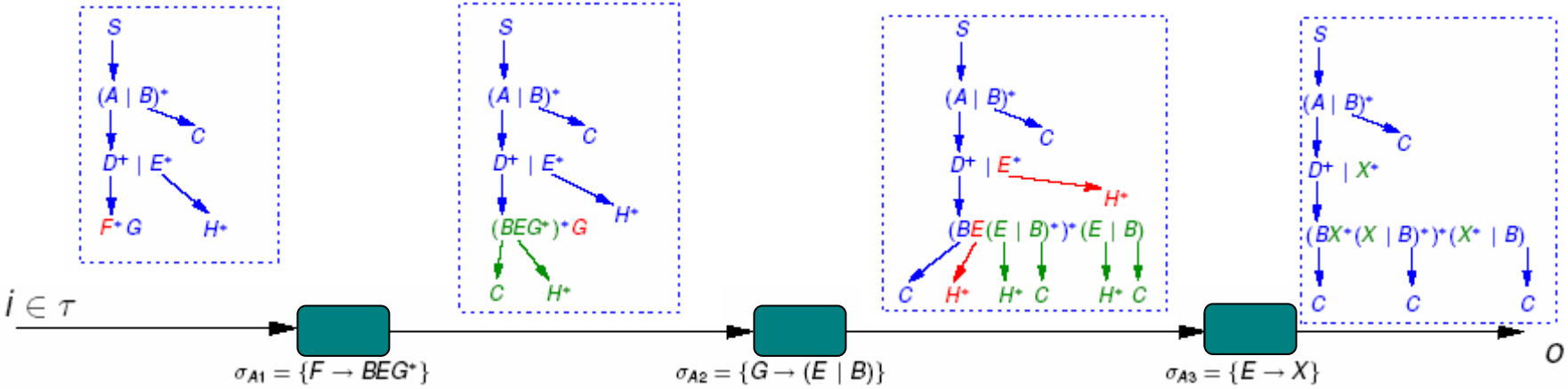
$\Delta_A = \langle \sigma, \tau_\alpha, \tau_\omega \rangle$ with

$\sigma : \{G \rightarrow E \mid B\}$,

$\tau_\alpha : \{G ::= \langle \mathrm{g} \rangle Z\}$, and

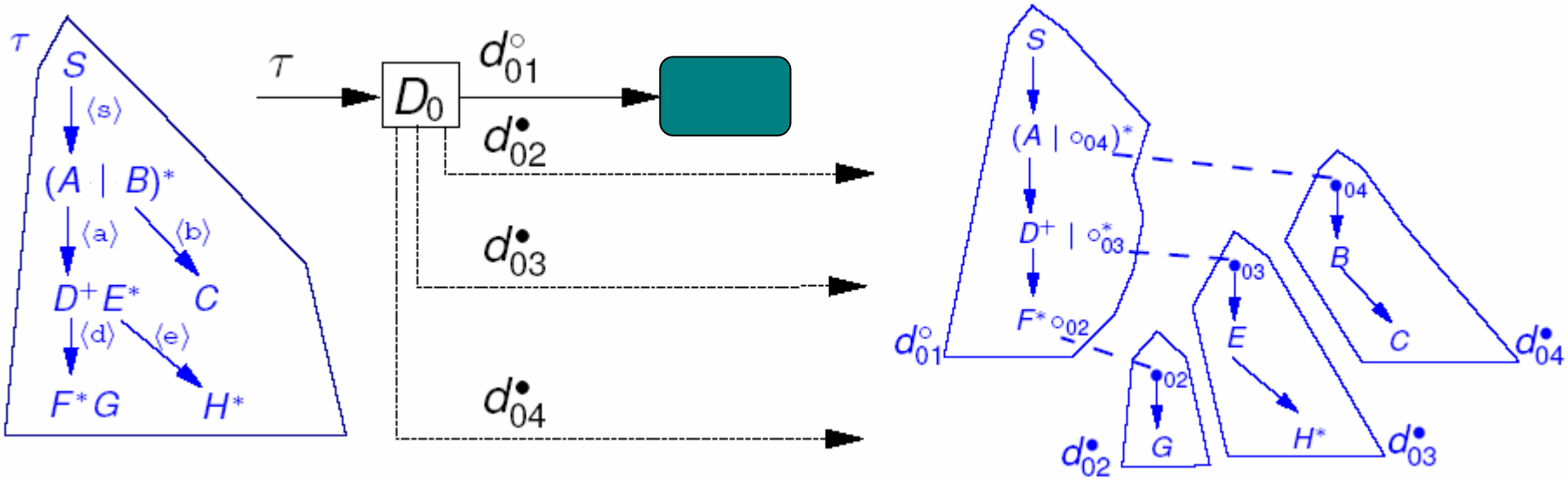
$\tau_\omega : \{E ::= \langle \mathrm{c} \rangle H^*,\ B ::= \langle \mathrm{b} \rangle C, H ::= \langle \mathrm{h} \rangle Z,\ C ::= \langle \mathrm{c} \rangle Z\}$.

Modeled Workflow



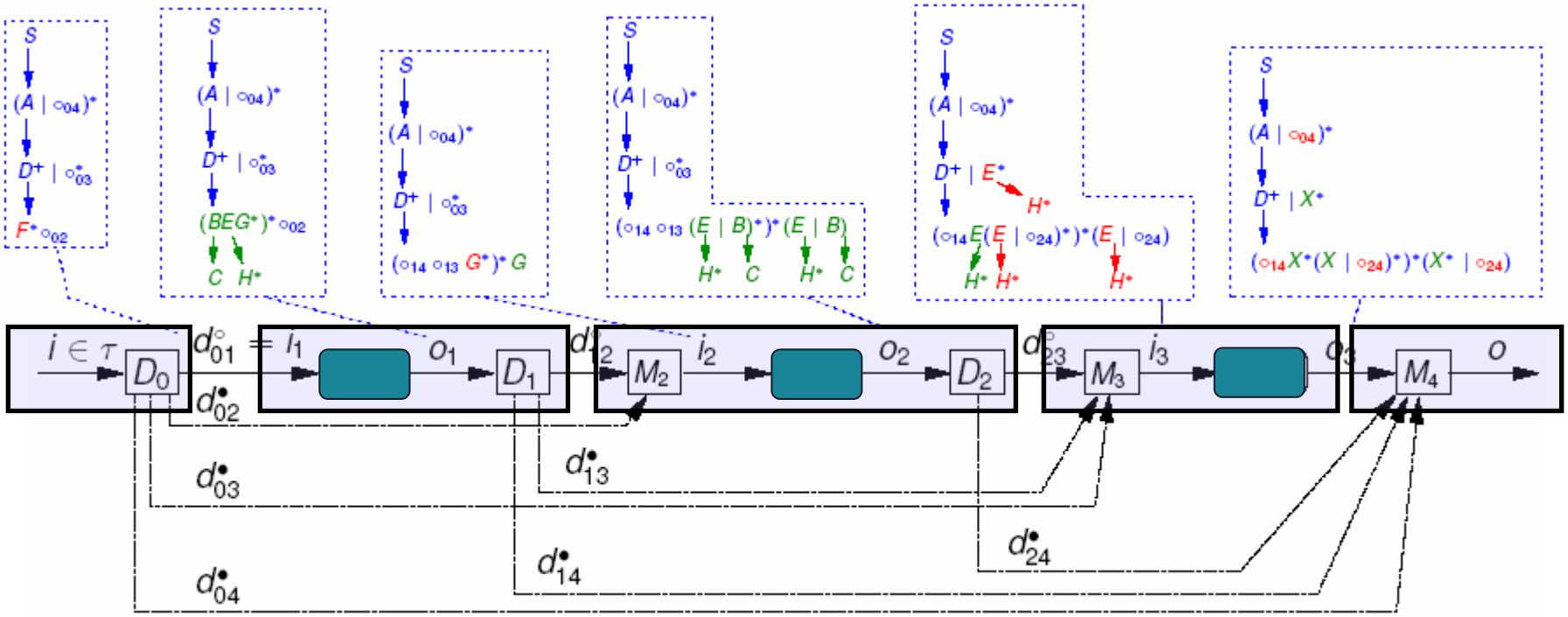
- Type propagation
- Data dependency analysis

Shipping Idea



- Bypassing approach
- Keep associations via holes and fillers

Auto-generated Shipping-optimized Workflow



- Distributors and mergers
- Direct connections between all hosts

Distributors

Compile-time

- Routing specification computed by dataflow-analysis algorithm
- Input schema, annotated with destination for each type

Runtime

- Glushkov automata to parse input stream
- Data is sent to different output channels and holes and fillers are inserted

Holes Need Two Indexes

i -- From which distributor

- If not, merger would not know from where to read the data

j -- To which merger

- If not, merger would not know whether it should fill the hole (or pass on)

Mergers

Merger `j`:

- 1. Read from main line until i,j-hole is found (forward all other holes and data)
- 2. Read from bypass line i until closing i,j-filler
- 3. Go back to Step 1

Correctness and Optimality

Correctness

- If data fragment X is bypassed around A, then X is not read by A

Optimality

- If data fragment X is merged into A, then X is read by A

O(shipping savings) ?

- Linear in size of bypassed data
- Linear in number of bypassed actors

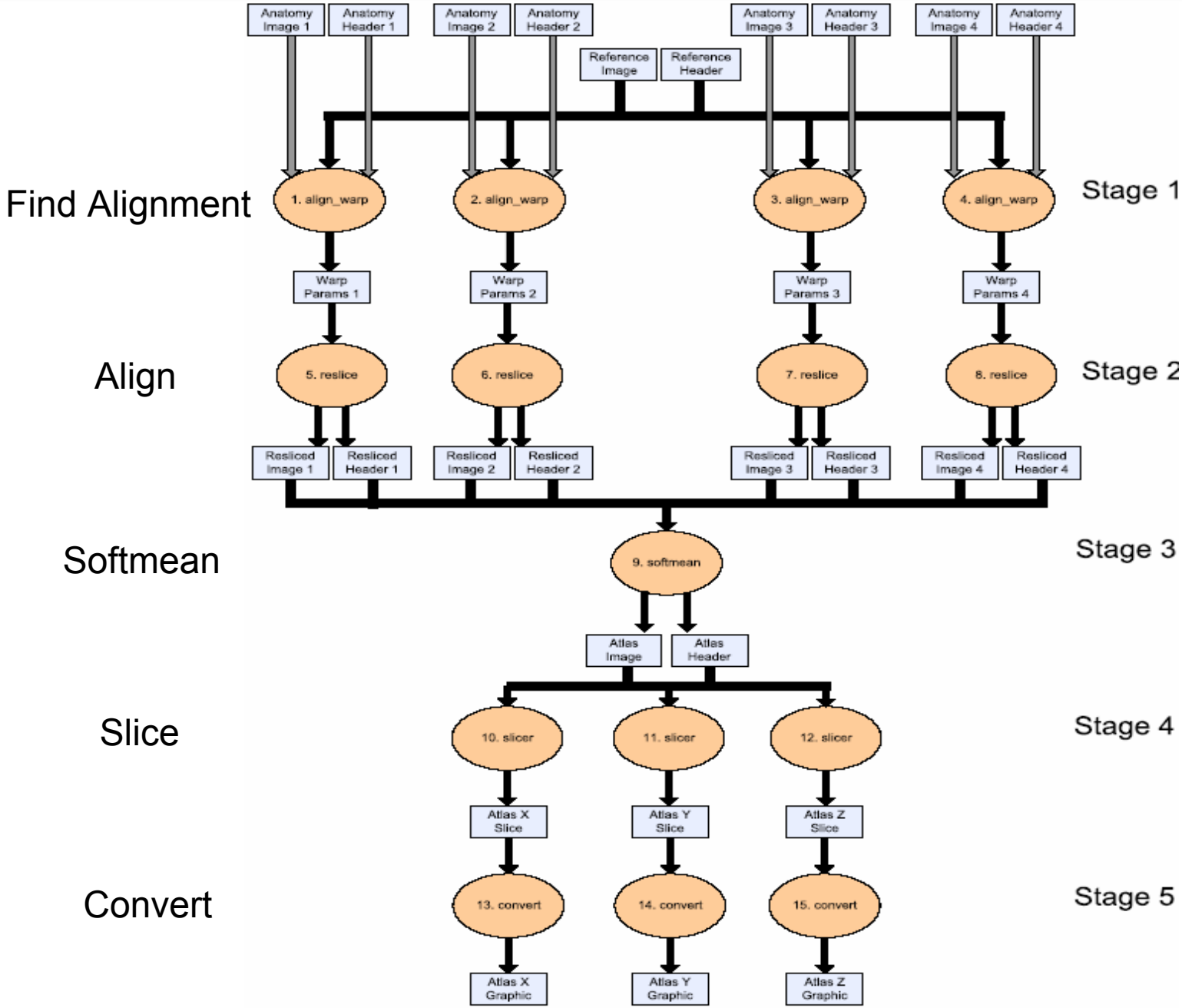
Experimental Evaluation

Environment

- Cluster of 40 Linux nodes
- 100 MBit/s switched network

System Implementation

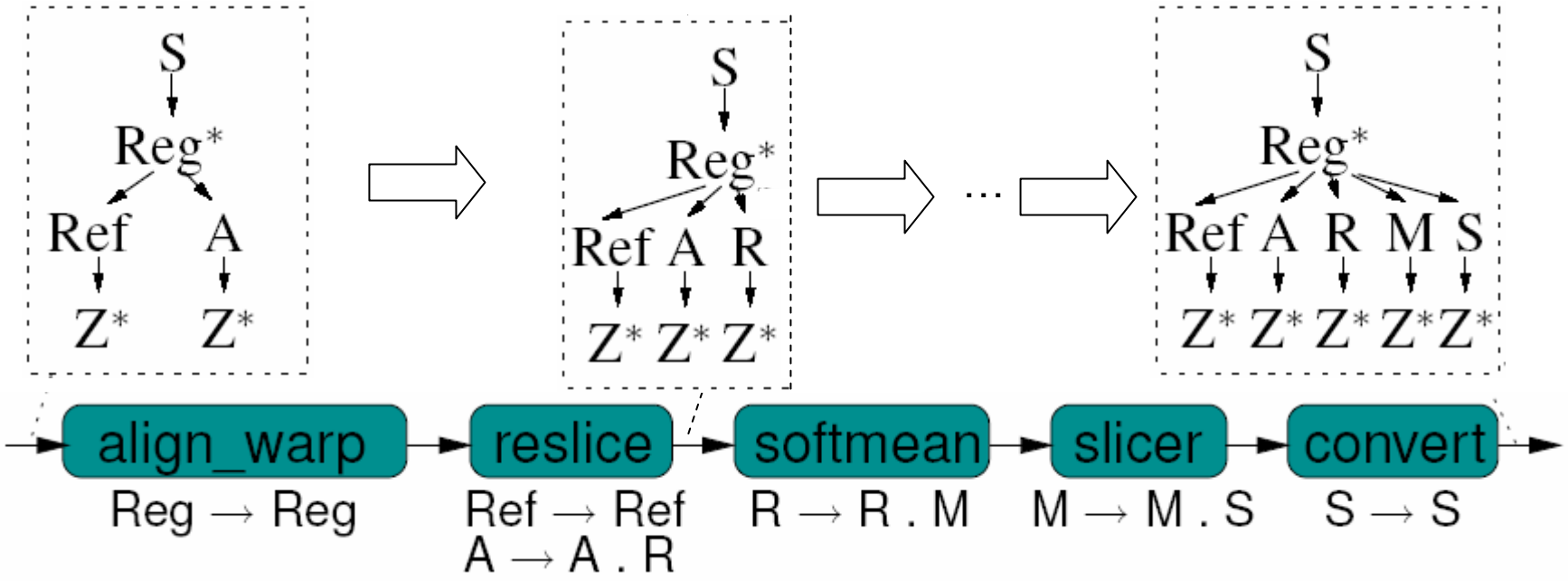
- Each actor as Unix process
- Written in C++/Perl on top of PVM
- PVM for passing Collection Tokens
- Base data as local files, scp to move


Find Alignment
Align
Softmean
Slice
Convert
Stage 1
Stage 2
Stage 3
Stage 4
Stage 5
Anatomy Image 1
Anatomy Header 1
Anatomy Image 2
Anatomy Header 2
Anatomy Image 3
Anatomy Header 3
Anatomy Image 4
Anatomy Header 4
Reference Image
Reference Header
1. align_warp
2. align_warp
3. align_warp
4. align_warp
Warp Params 1
Warp Params 2
Warp Params 3
Warp Params 4
5. reslice
6. reslice
7. reslice
8. reslice
Resliced Image 1
Resliced Header 1
Resliced Image 2
Resliced Header 2
Resliced Image 3
Resliced Header 3
Resliced Image 4
Resliced Header 4
9. softmean
Atlas Image
Atlas Header
10. slicer
11. slicer
12. slicer
Atlas X Slice
Atlas Y Slice
Atlas Z Slice
13. convert
14. convert
15. convert
Atlas X Graphic
Atlas Y Graphic
Atlas Z Graphic

fMRI Workflow

Functional Magnetic Resonance Imaging

- Post-processing of voxel images of the human brain
- Data intensive
- Computationally expensive



Experimentation Results for fMRI Workflow

Input/Output data size:

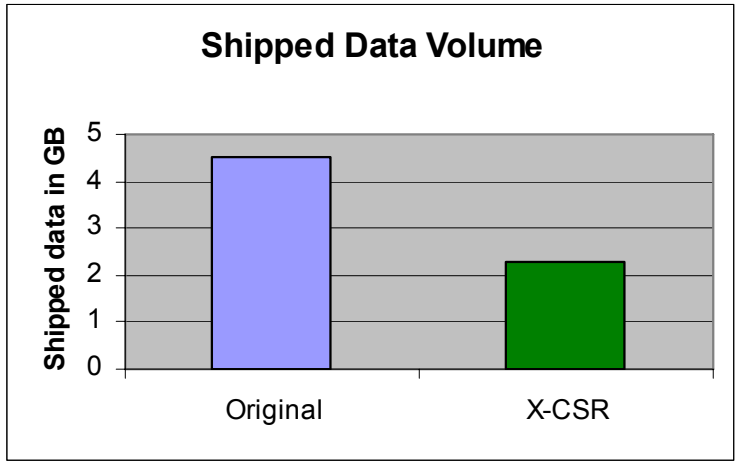
400MB/900MB

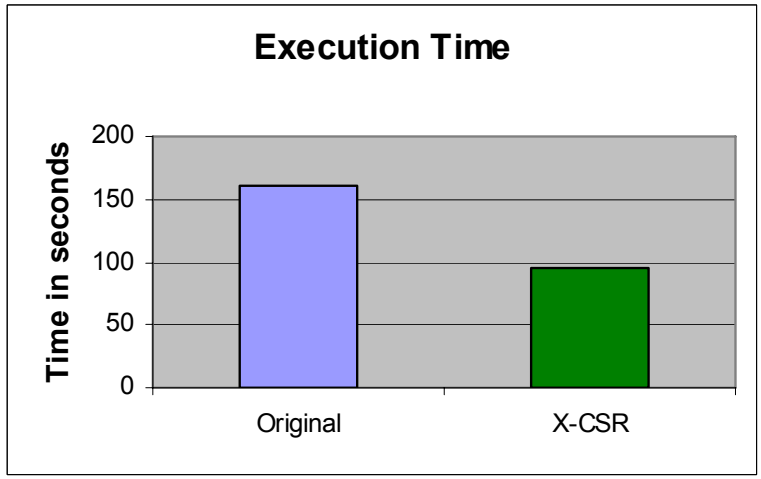
Data shippings:

4.5GB vs. 2.3GB (-49%)

Execution time:

160s vs. 95s (-40%)





Summary

∆-XML

- Flexible paradigm for processing tree structured data
- Currently used in scientific workflow system Kepler

Static Analysis

- Signatures approximate actor behavior
- More than type-in + type-out
- Track data dependencies

Shipping Optimization

- Automatic optimization informed by static knowledge
- 40% faster for scientific example

Thank you.

Questions?