Scientific Workflow Design with Data Assembly Lines

Daniel Zinn Shawn Bowers Timothy McPhillips Bertram Ludäscher

WORKS 2009

Big Picture / Motivation

SciWF to integrate components to build scientific Applications

Advantages

- Provenance, distributed resource usage, resource and component discovery, etc.
- Design easier than general-purpose scripting language
- Self-explanatory – worth sharing

Yet, workflow design is often slow, and error prone

- We identify and illustrate recurring workflow challenges
- Currently addressed by *shims* and lots of wiring

→ Propose modeling paradigm to address these

Outline

Scientific Workflow Notions

Common Challenges in SciWF-Design

Virtual Data Assembly Lines

Challenges Addressed

Discussion and Future Work

Outline

Scientific Workflow Notions

Common Challenges in SciWF-Design

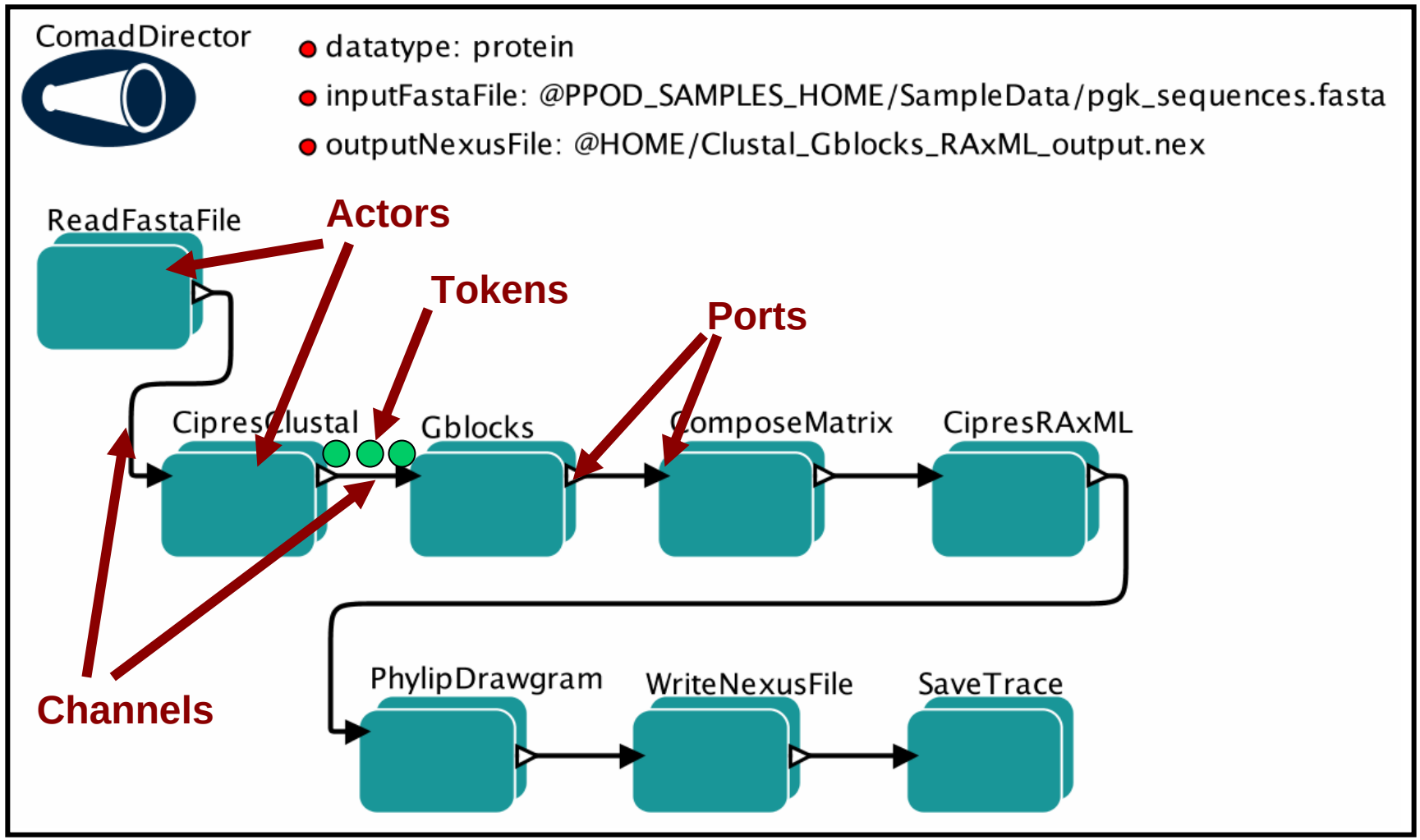
Virtual Data Assembly Lines

Challenges Addressed

Discussion and Future Work

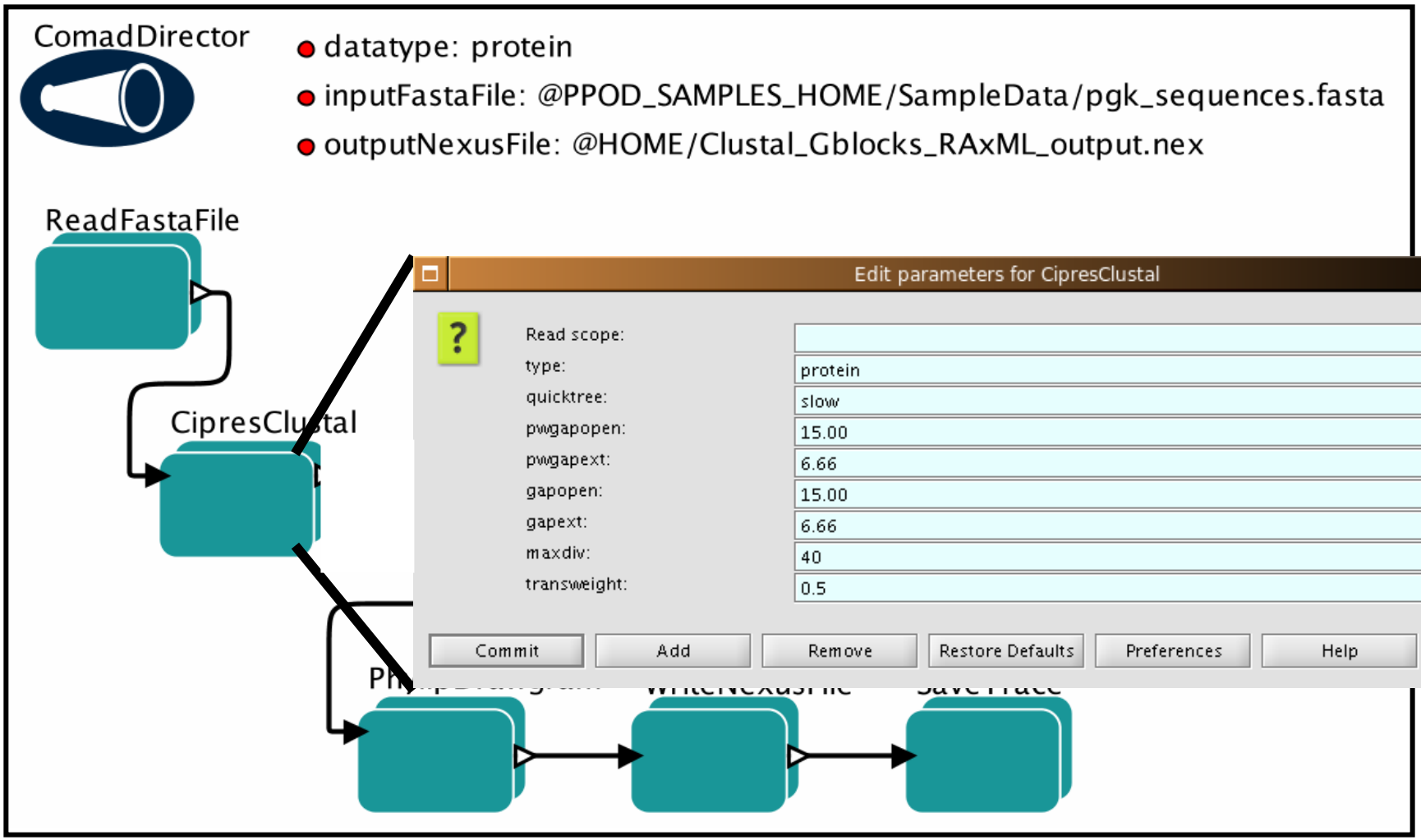
SciWF Notions and Terms

SciWF = Executable specification of Scientific Method



SciWF Notions and Terms

SciWF = Executable specification of Scientific Method



Outline

Scientific Workflows (our notions and terms)

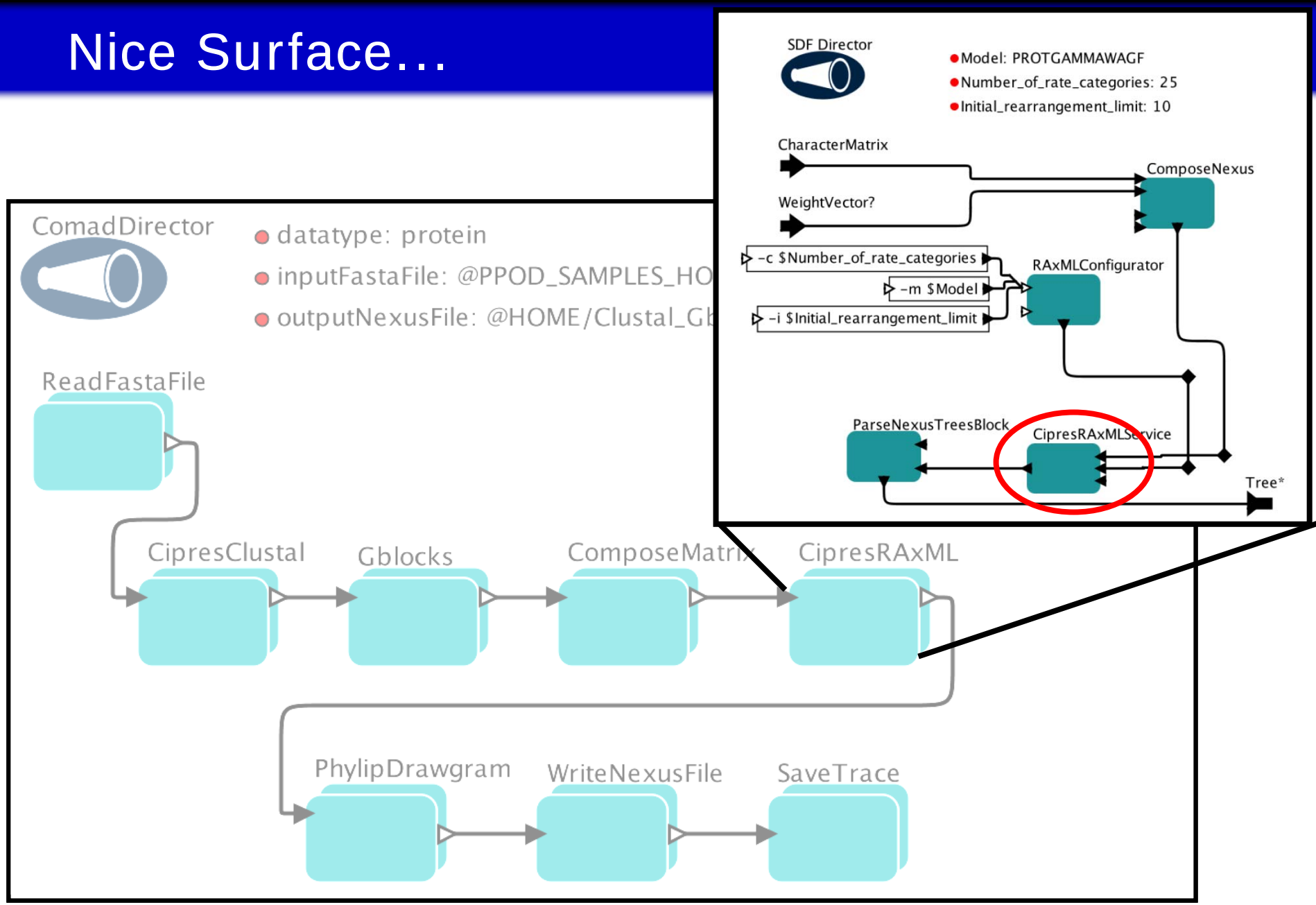
Common Challenges in SciWF-Design

Virtual Data Assembly Lines

Challenges Addressed

Discussion and Future Work

Nice Surface...

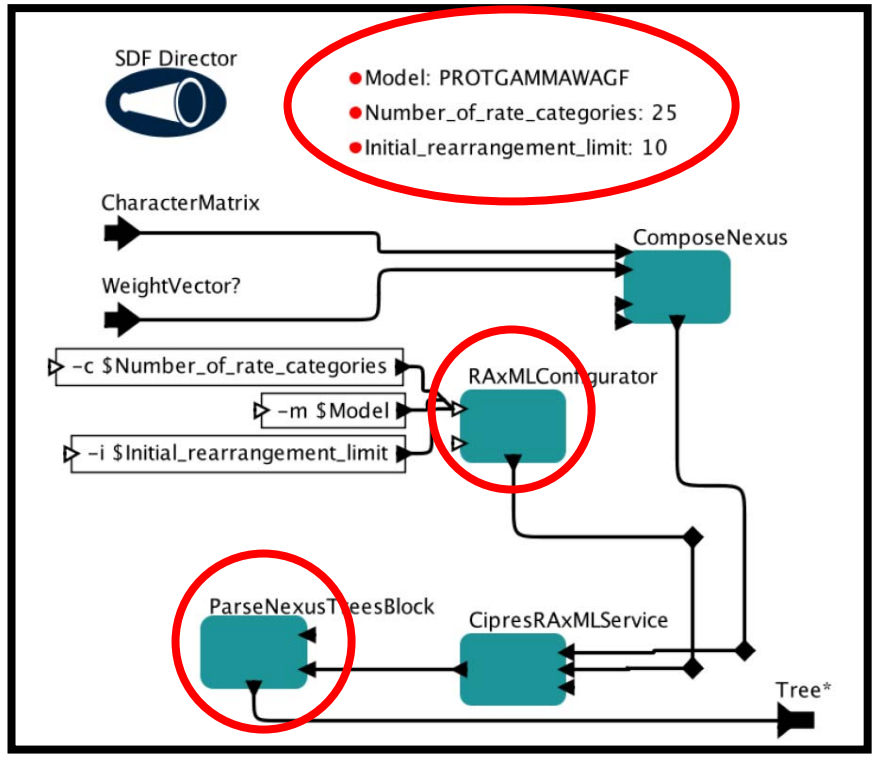


Challenge 1: Parameter-Rich Services

Algorithms not only require input data, also many parameters

Example:

- RAxML: 32 cmdline params and input data!
- DNAML: 10 params + data



Problem

- 1 param → 1 port → 1 wire

Current Approach: Composites w/

- Actor parameters (cannot set from input data)
- Packaging input and outputs into ad-hoc records (who builds the records?)

Challenge 2: Maintaining Data Cohesion

Relationships between data is often important

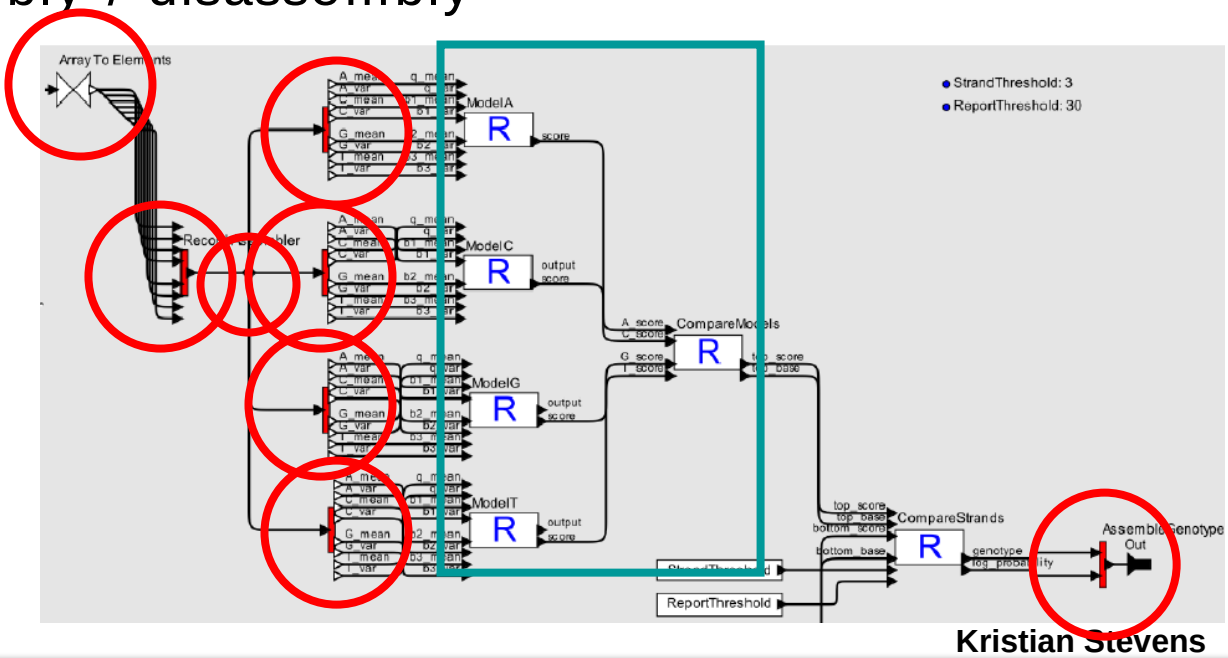
- "Work package for actor"
- Related output

Current Approach: Ad-hoc record tokens

- Manual assembly / disassembly

Problems

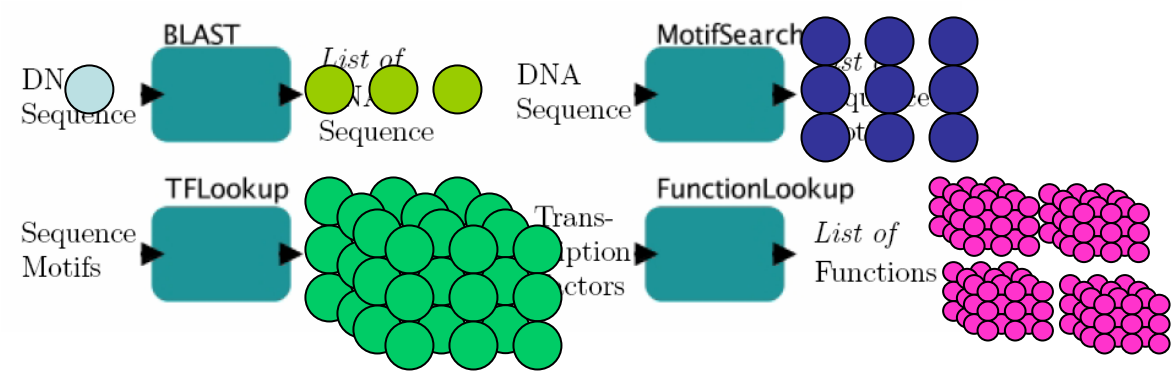
- Tedious
- Not change resilient
- Concurrency drawbacks



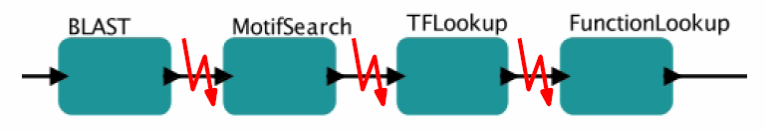
Kristian Stevens

Challenge 3: Handling [Nested] Arrays

Actors often create lists as outputs



Napkin-Drawing

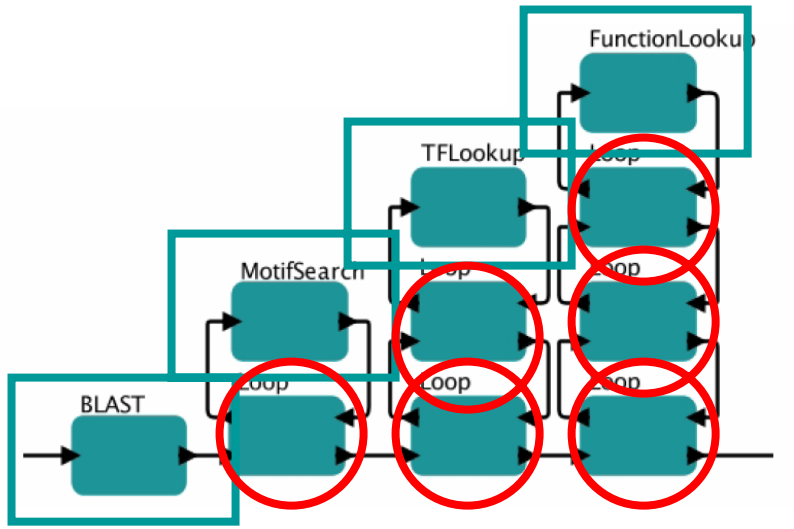


→ Looses nesting information

Relating Functions with DNA Sequence hard for subsequent WF steps

→ With arrays not compatible

Actual Workflow

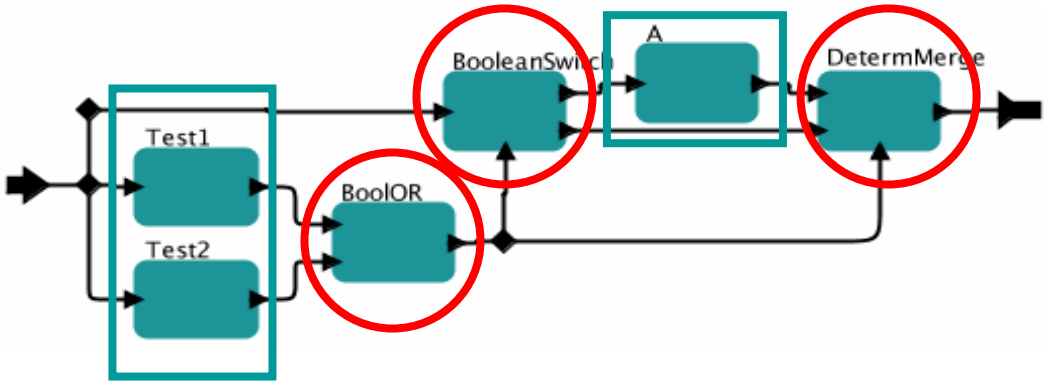


Challenge 4: Conditional Execution

If (Test1 || Test2) Then A

- Scientific actors: A, Test1, Test2

PN Solution:



Problems

- Seems a little complex for a simple if-then
- Does not work if A outputs more than one token

Imagine more complex control-flow encoded in WF Graph

Challenge 5: Iterations over Cross Products

SWFs to automate multiple calls for varying input data sets & parameters

Current approaches

- Modeled in the WF Graph
- Write/use custom actor

Problems

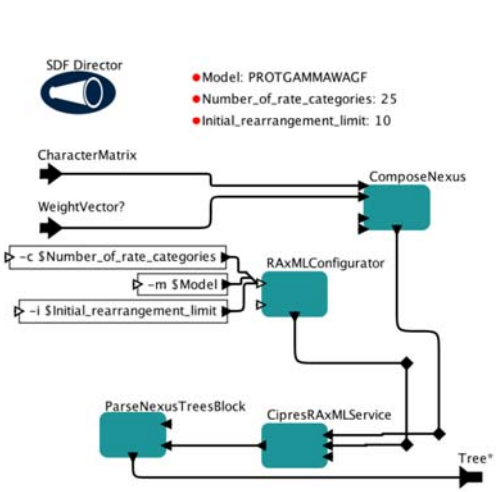
- Many actors
- Data as stream or array?

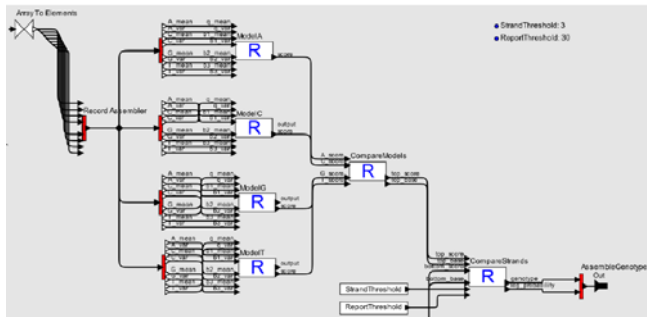
Common Theme

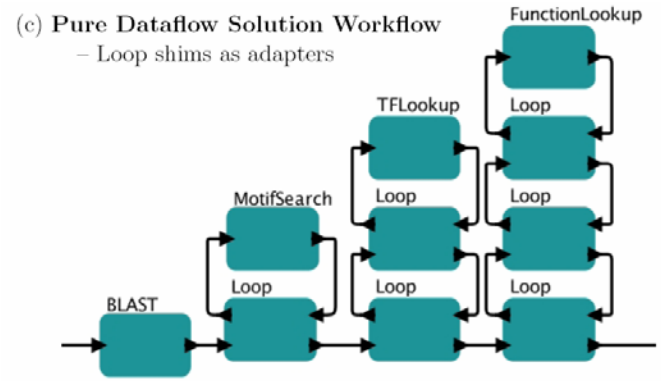
Use-cases

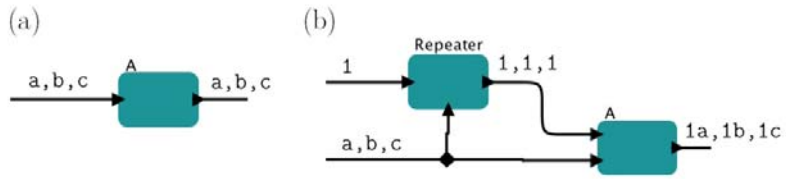
[Parameter-rich, Data Cohesion (Records & Arrays), Conditionals, Cross-products]

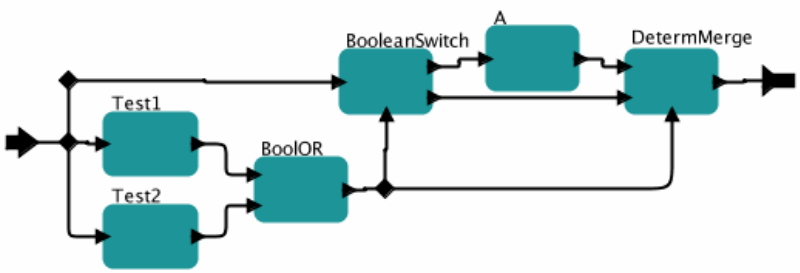
are addressed with composites, shims and wires

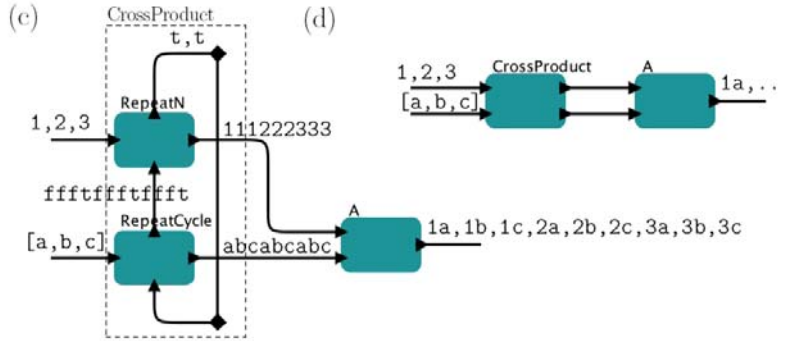












Problems with [too many] Shims and Wires

Shims need to be placed and connected

- Tedious, error-prone

Distract from scientific meaningful actors

- Non-descriptive workflows – worth sharing?

Data Organization is encoded in workflow structure

- Not robust to data changes

Shims often lead to complex designs

- Imagine all previous `design-patterns' intertwined
- GOTO-programming

VDAL: *Raising the level of abstraction*

- Localized control-flow
- Data management not done via wires
- Actors are coupled not by wire but by data!

Outline

Scientific Workflows (our notions and terms)

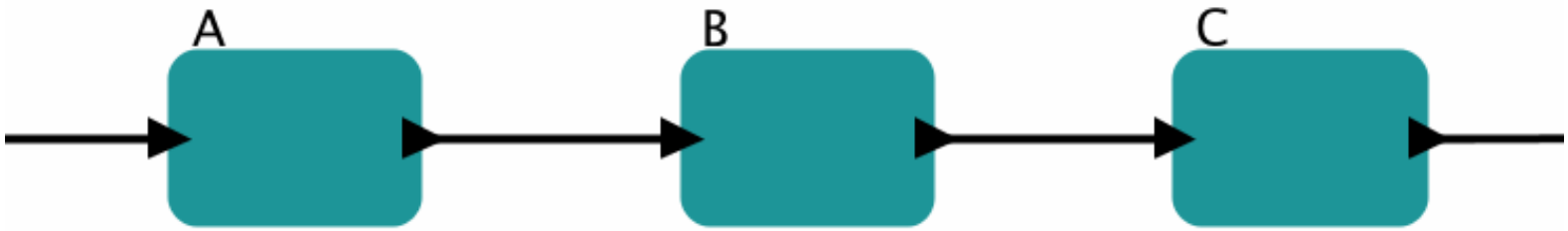
Common Challenges in SciWF-Design

Virtual Data Assembly Lines

Challenges Addressed

Discussion and Future Work

Virtual Data Assembly Lines (COMAD in Kepler)



Data is organized as XML-like tree structures

Encapsulate actor within a configurable *shell*

- selection, binding, iteration for input data
  - Based on data organization and not on up-stream workflow
- placement of output data
  - Into XML structure and not to specific input ports

Let's see an example

VDAL: Actor Example

RAxML Service

Interface:

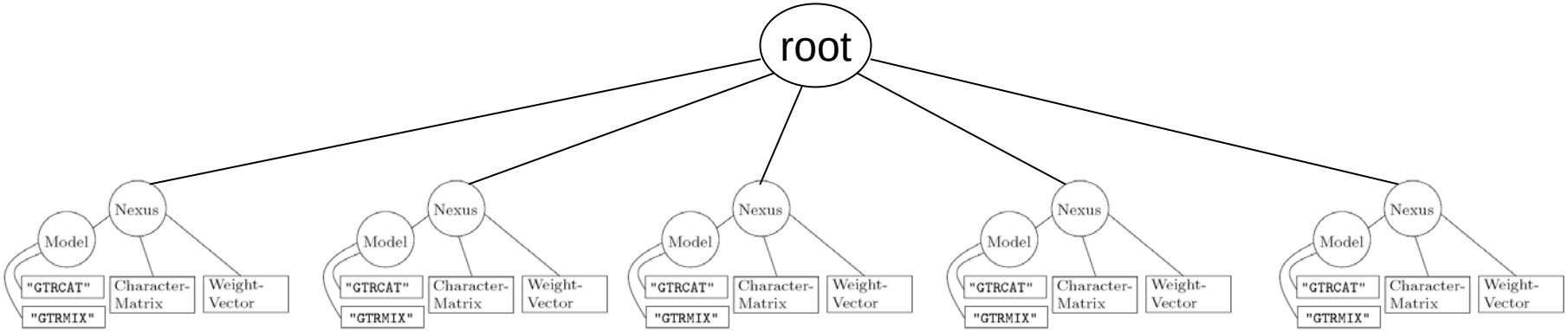
ScientificActor: CipresRAxML
Input: model **of String**
cha_matrix **of CharacterMatrix**
weight_vec **of WeightVector***
rate_cats **of Integer**
init_rearr_limit **of Integer**
Output: trees **of PhyloTree***

Task:

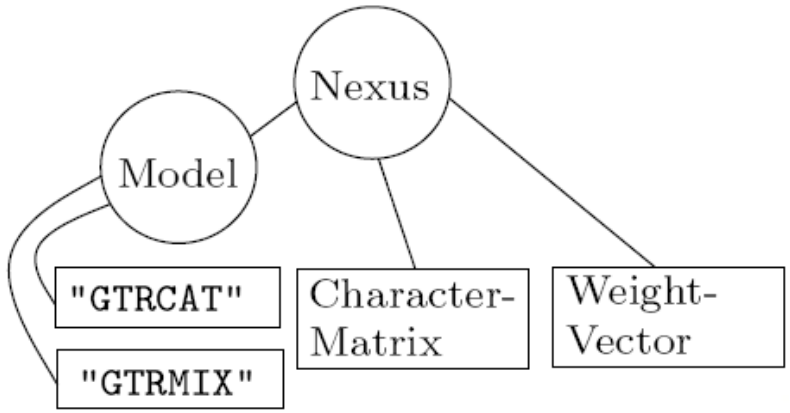
- Input: Collection of multiple input data sets D
- For-each D in input call RAxML with different configurations
  - Try all *models* given in D
  - Get *CharacterMatrix* and *WeightVector* from D
  - Try as *rate_cats* 25 and 100
  - Set *init_rearr_limit* to 100
- Enrich collection D with output trees

VDAL: Data Modeling

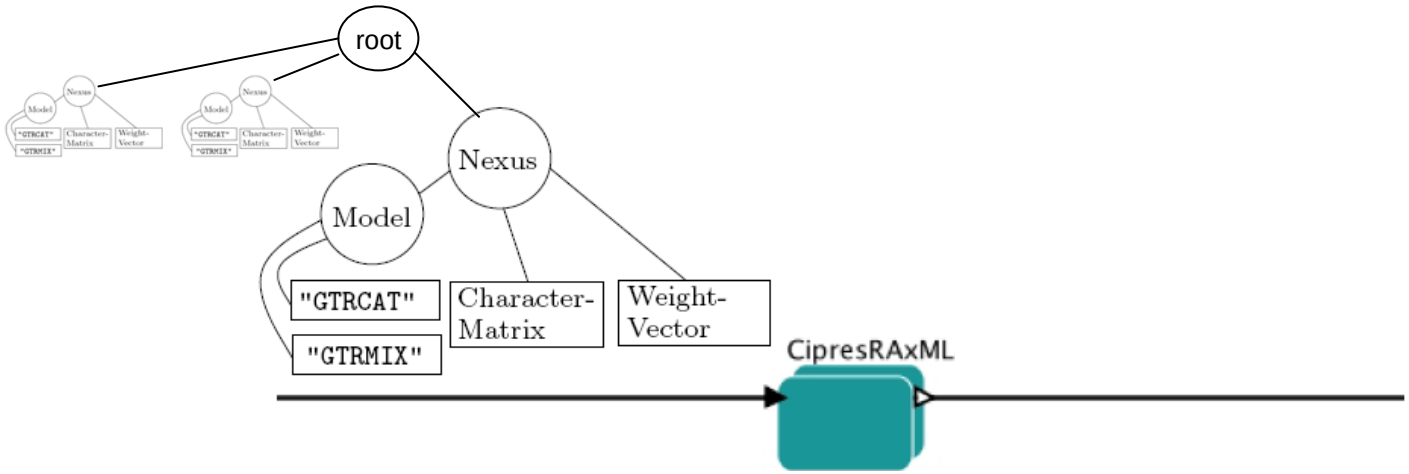
Input data structure:



One data package D:



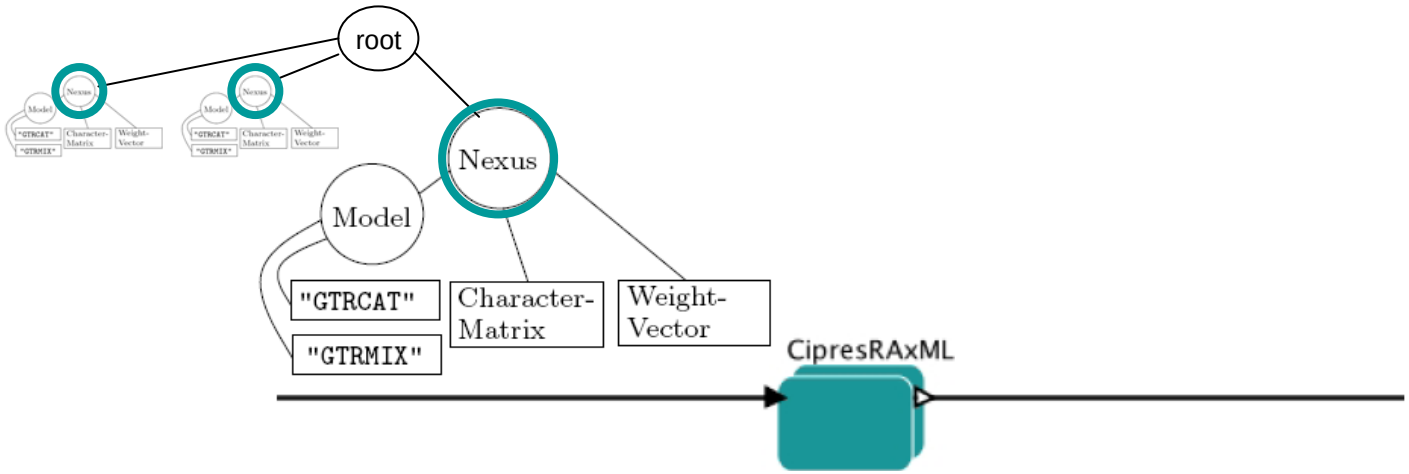
Actor Configuration Example



Configuration comprises three major parts:

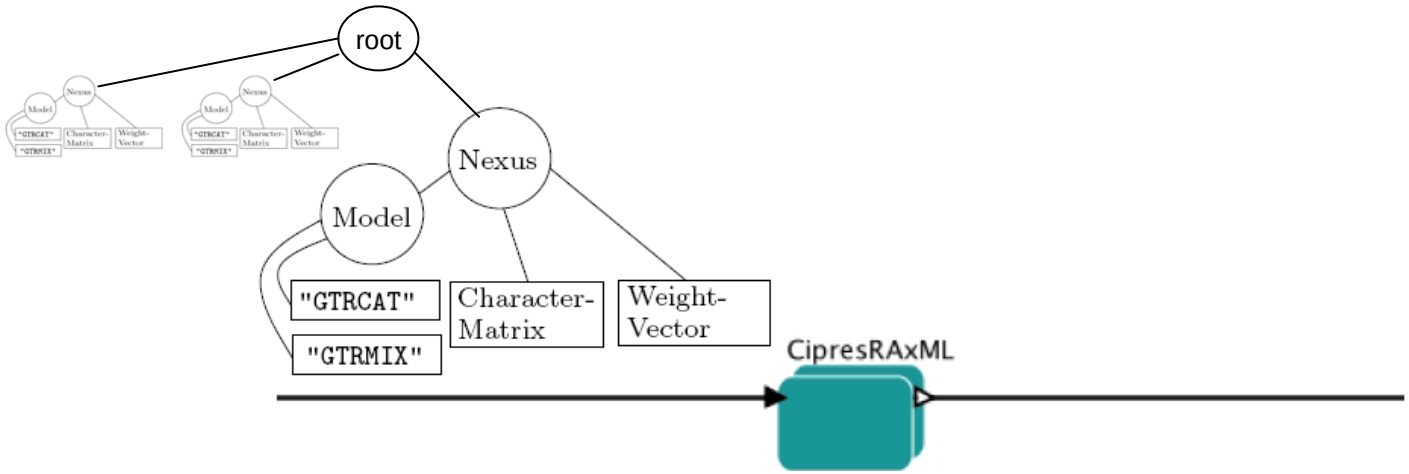
Scope σ – to select scope of actor invocation
Input assembler γ – to create inputs for A
Write expression ω – to write results back into data stream

VDAL: Actor Example Deployed




```
8   σ: //Nexus
9   γ: model          ← foreach $p in //Model return $p/String
10     cha_matrix     ← //CharacterMatrix
11     weight_vec     ← //WeightVector
12     rate_cats      ← foreach $r in {25},{100} return $r
13     init_rearr_limit ← 100
14  ω: INSERT AS LAST INTO . VALUE Trees[ $result/trees ]
```

VDAL: Actor Example Deployed

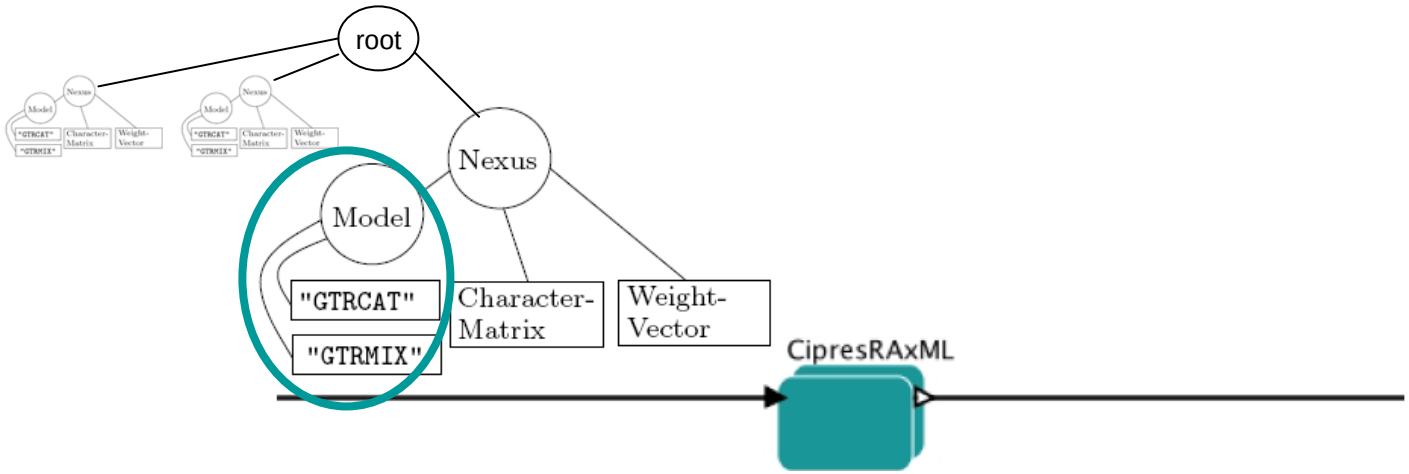


Input data for service invocation

model	cha_matrix	weight_vec	rate_cats	init_rearr_..

```
8   σ: //Nexus
9   γ: model           ← foreach $p in //Model return $p/String
10     cha_matrix      ← //CharacterMatrix
11     weight_vec      ← //WeightVector
12     rate_cats       ← foreach $r in {25},{100} return $r
13     init_rearr_limit ← 100
14  ω: INSERT AS LAST INTO . VALUE Trees[ $result/trees ]
```

VDAL: Actor Example Deployed

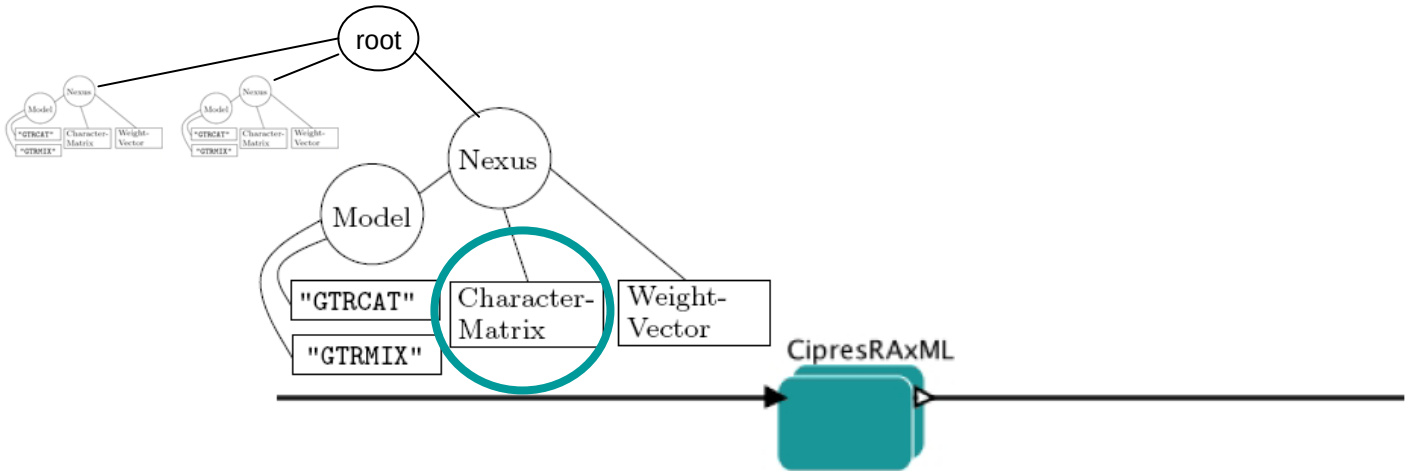


Input data for service invocation

model	cha_matrix	weight_vec	rate_cats	init_rearr_..
GTRCAT				
GTRMIX				

```
8   σ: //Nexus
9   γ: model          ← foreach $p in //Model return $p/String
10     cha_matrix     ← //CharacterMatrix
11     weight_vec     ← //WeightVector
12     rate_cats      ← foreach $r in {25},{100} return $r
13     init_rearr_limit ← 100
14  ω: INSERT AS LAST INTO . VALUE Trees[ $result/trees ]
```

VDAL: Actor Example Deployed

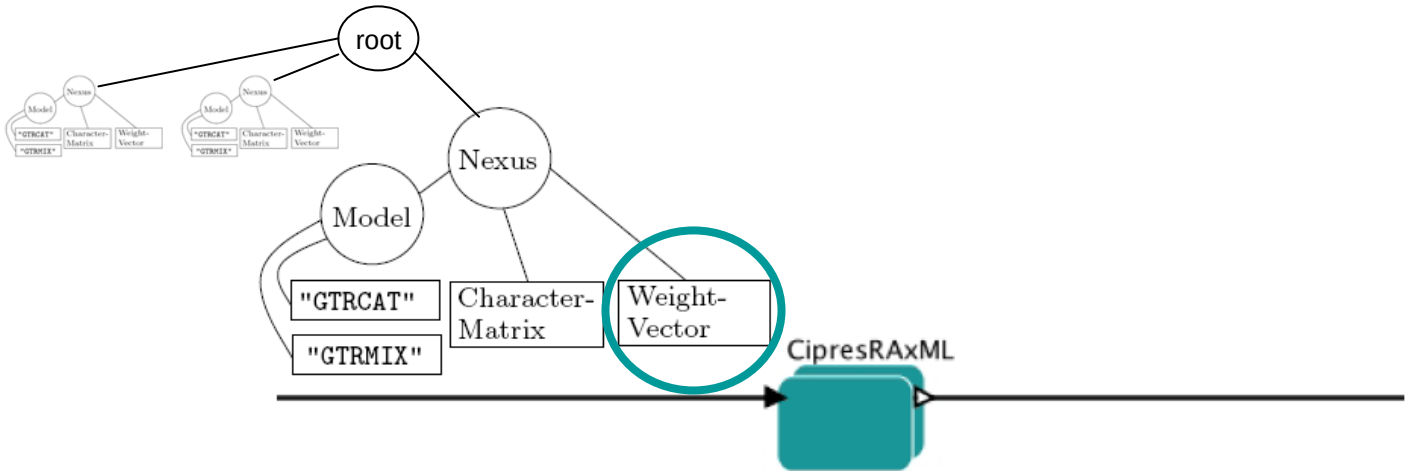


Input data for service invocation

model	cha_matrix	weight_vec	rate_cats	init_rearr_..
GTRCAT	Matrix 1			
GTRMIX	Matrix 1			

```
8   σ: //Nexus
9   γ: model          ← foreach $p in //Model return $p/String
10     cha_matrix     ← //CharacterMatrix
11     weight_vec     ← //WeightVector
12     rate_cats      ← foreach $r in {25},{100} return $r
13     init_rearr_limit ← 100
14  ω: INSERT AS LAST INTO . VALUE Trees[ $result/trees ]
```

VDAL: Actor Example Deployed

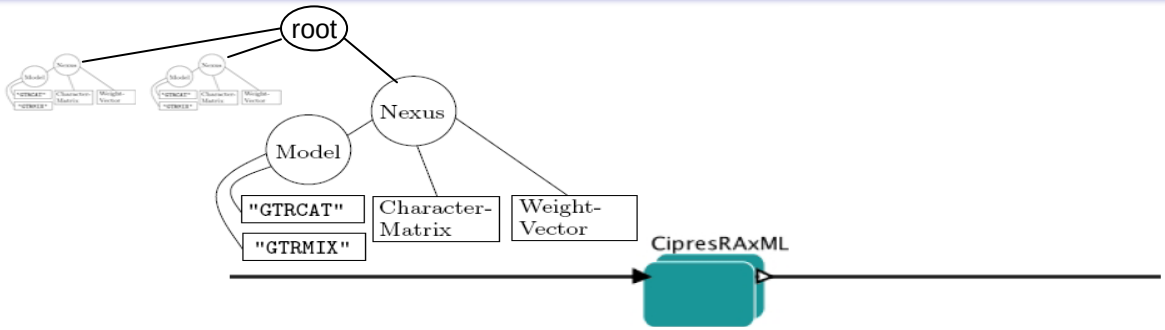


Input data for service invocation

model	cha_matrix	weight_vec	rate_cats	init_rearr_..
GTRCAT	Matrix 1	Vector 1		
GTRMIX	Matrix 1	Vector 1		

```
8   σ: //Nexus
9   γ: model          ← foreach $p in //Model return $p/String
10     cha_matrix     ← //CharacterMatrix
11     weight_vec     ← //WeightVector
12     rate_cats      ← foreach $r in {25},{100} return $r
13     init_rearr_limit ← 100
14  ω: INSERT AS LAST INTO . VALUE Trees[ $result/trees ]
```

VDAL: Actor Example Deployed

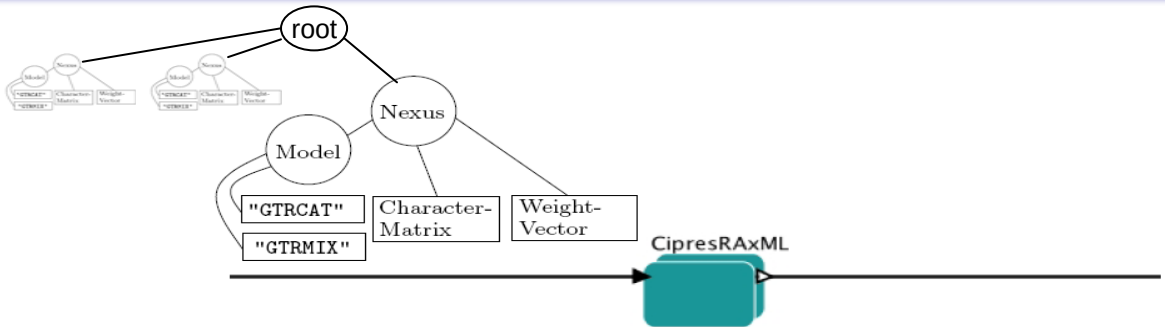


Input data for service invocation

model	cha_matrix	weight_vec	rate_cats	init_rearr_..
GTRCAT	Matrix 1	Vector 1	25	
GTRMIX	Matrix 1	Vector 1	25	
GTRCAT	Matrix 1	Vector 1	100	
GTRMIX	Matrix 1	Vector 1	100	

```
8   σ: //Nexus
9   γ: model          ← foreach $p in //Model return $p/String
10     cha_matrix     ← //CharacterMatrix
11     weight_vec     ← //WeightVector
12     rate_cats      ← foreach $r in {25},{100} return $r
13     init_rearr_limit ← 100
14  ω: INSERT AS LAST INTO . VALUE Trees[ $result/trees ]
```

VDAL: Actor Example Deployed

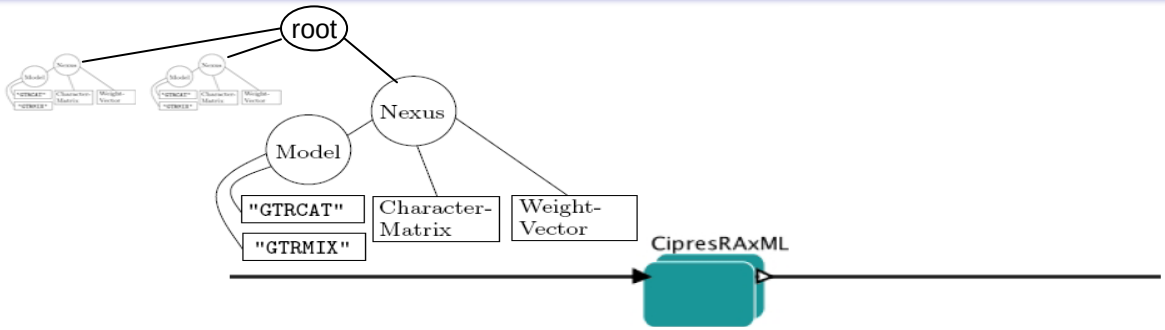


Input data for service invocation

model	cha_matrix	weight_vec	rate_cats	init_rearr_..
GTRCAT	Matrix 1	Vector 1	25	100
GTRMIX	Matrix 1	Vector 1	25	100
GTRCAT	Matrix 1	Vector 1	100	100
GTRMIX	Matrix 1	Vector 1	100	100

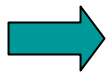
```
8   σ: //Nexus
9   γ: model           ← foreach $p in //Model return $p/String
10     cha_matrix      ← //CharacterMatrix
11     weight_vec      ← //WeightVector
12     rate_cats       ← foreach $r in {25}, {100} return $r
13     init_rearr_limit ← 100
14  ω: INSERT AS LAST INTO . VALUE Trees[ $result/trees ]
```

VDAL: Actor Example Deployed



Input data for service invocation

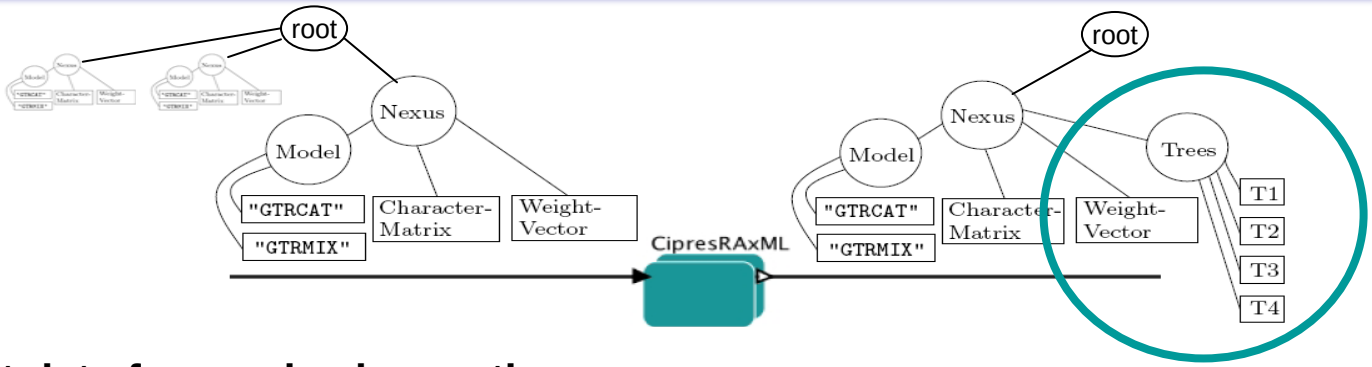
model	cha_matrix	weight_vec	rate_cats	init_rearr_..
GTRCAT	Matrix 1	Vector 1	25	100
GTRMIX	Matrix 1	Vector 1	25	100
GTRCAT	Matrix 1	Vector 1	100	100
GTRMIX	Matrix 1	Vector 1	100	100



trees
T1
T2
T3
T4

```
8   σ: //Nexus
9   γ: model          ← foreach $p in //Model return $p/String
10     cha_matrix     ← //CharacterMatrix
11     weight_vec     ← //WeightVector
12     rate_cats      ← foreach $r in {25},{100} return $r
13     init_rearr_limit ← 100
14  ω: INSERT AS LAST INTO . VALUE Trees[ $result/trees ]
```

VDAL: Actor Example Deployed



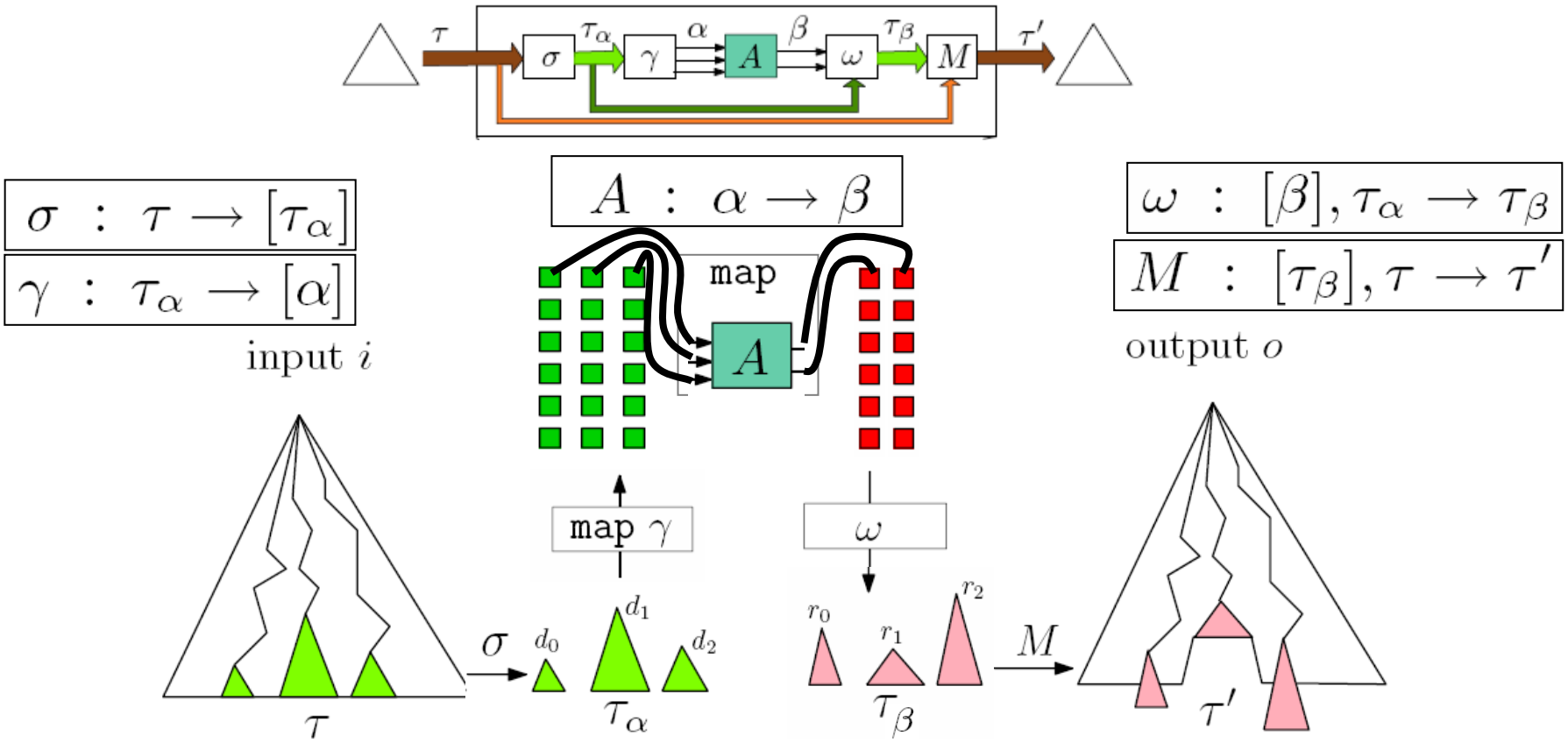
Input data for service invocation

model	cha_matrix	weight_vec	rate_cats	init_rearr_..
GTRCAT	Matrix 1	Vector 1	25	100
GTRMIX	Matrix 1	Vector 1	25	100
GTRCAT	Matrix 1	Vector 1	100	100
GTRMIX	Matrix 1	Vector 1	100	100

trees
T1
T2
T3
T4

```
8   σ: //Nexus
9   γ: model          ← foreach $p in //Model return $p/String
10     cha_matrix     ← //CharacterMatrix
11     weight_vec     ← //WeightVector
12     rate_cats      ← foreach $r in {25},{100} return $r
13     init_rearr_limit ← 100
14  ω: INSERT AS LAST INTO . VALUE Trees[ $result/trees ]
```

VDAL Actor Execution Semantics



Scope σ – to select scope of actor invocation

Input assembler γ – to create inputs for A

Write expression ω – to write results back into data stream

Outline

Scientific Workflows (our notions and terms)

Common Challenges in SciWF-Design

Virtual Data Assembly Lines

Challenges Addressed

Summary and Future Work

C1: Parameter-rich Black Boxes

- Parameters selected from input or specified as literal values

→ Declarative specification via XPath-like expressions

→ Input is not bound to up-stream actor but to labels in structure-rich input stream

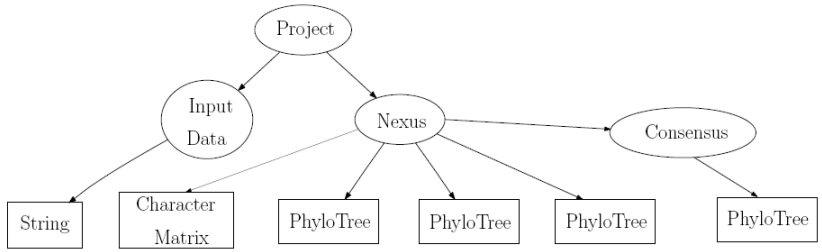
```
8   σ: //Nexus
9   γ: model          ← foreach $p in //Model return $p/String
10     cha_matrix     ← //CharacterMatrix
11     weight_vec     ← //WeightVector
12     rate_cats      ← foreach $r in {25},{100} return $r
13     init_rearr_limit ← 100
14  ω: INSERT AS LAST INTO . VALUE Trees[ $result/trees ]
```

→Decoupled from WF Graph and previous actors!
Uses XML structure as indirection

C2 & C3: Maintaining Data Cohesion

Records and Arrays in XML Structure

→ Declarative and robust access



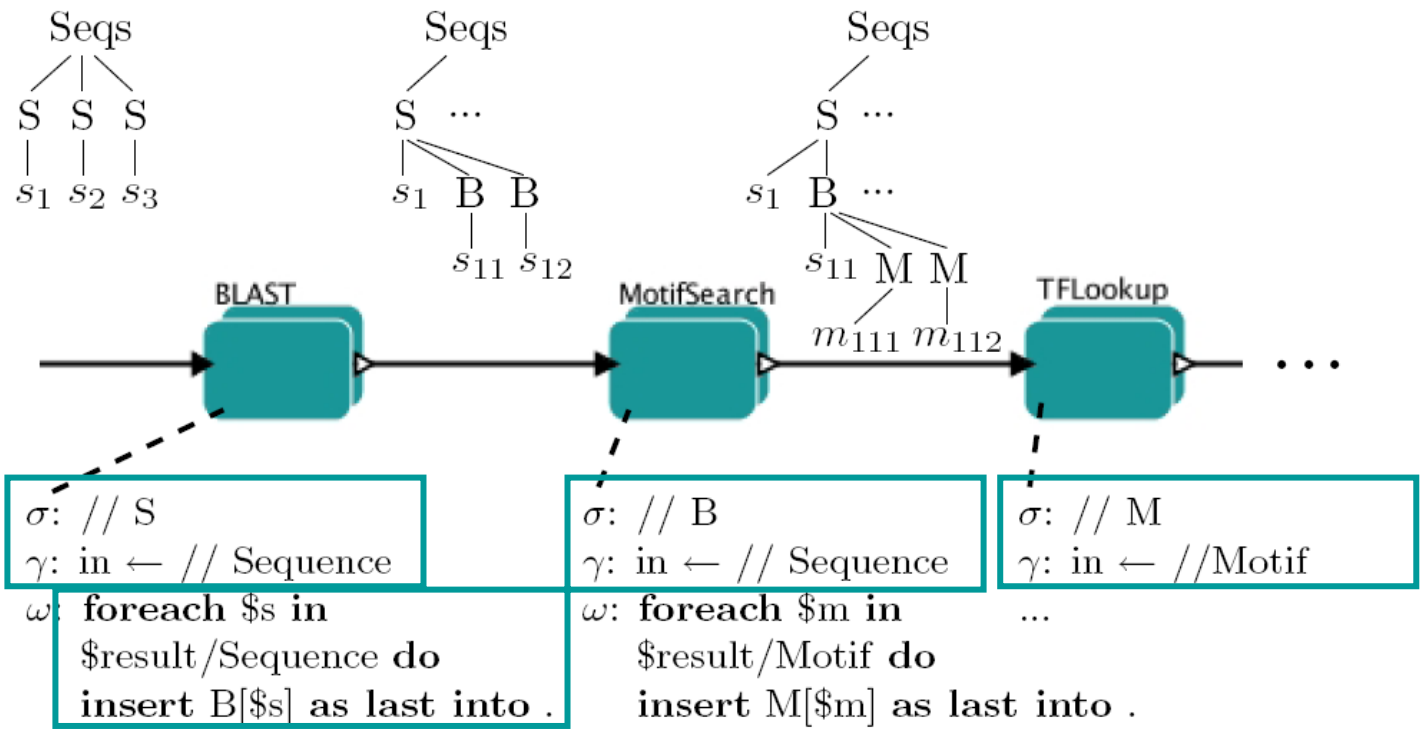
Example:

BLAST

MotifSearch

TFLookup

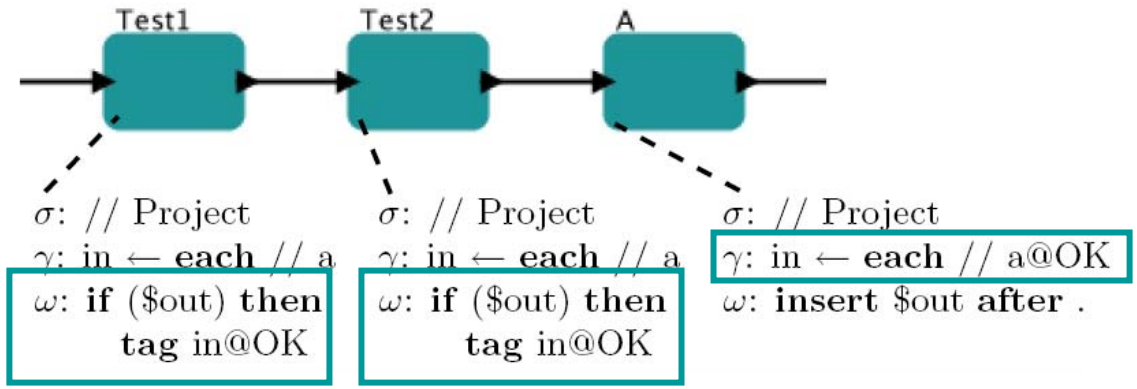
FunctionLookup



C4: Conditional Execution

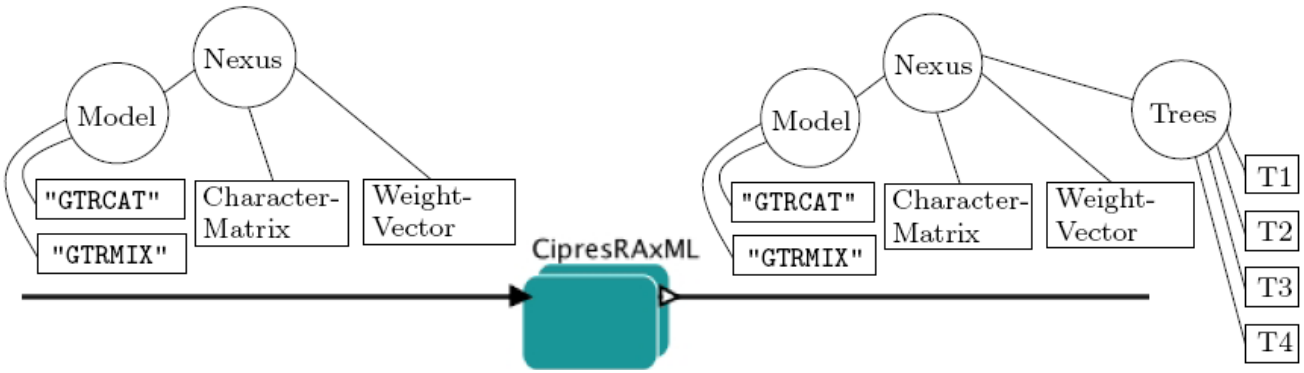
Tests add “tags”, actor branches on tag-information

→Local routing



C5: Iterations and Cross Products

Iteration over input data and/or parameters "built-in"




```
8   σ: //Nexus
9   γ: model          ← foreach $p in //Model return $p/String
10     cha_matrix     ← //CharacterMatrix
11     weight_vec     ← //WeightVector
12     rate_cats      ← foreach $r in {25},{100} return $r
13     init_rearr_limit ← 100
14  ω: INSERT AS LAST INTO . VALUE Trees[ $result/trees ]
```

Outline

Scientific Workflows (our notions and terms)

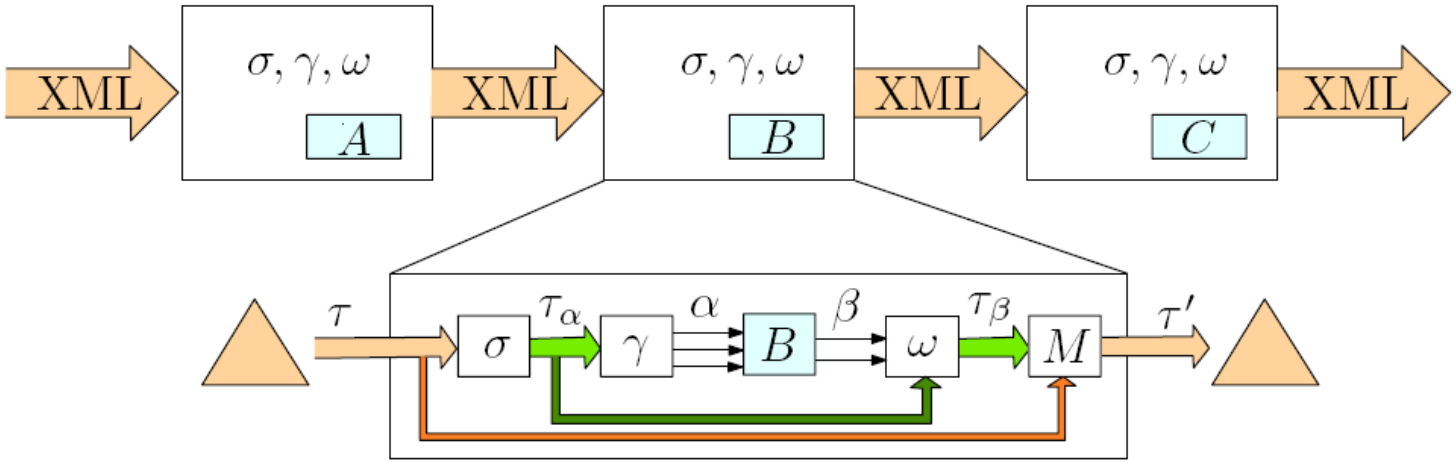
Common Challenges in SciWF-Design

Virtual Data Assembly Lines

Challenges Addressed

Summary and Future Work

Summary: VDAL = Linear & Declarative WFs



1. Data is organized as XML-like tree structures
2. Encapsulate actor within a configurable *shell*
3. Workflows are linear (as in assembly lines)

→ Declarative data selection, grouping and iteration

→ Decouples workflow graph from data organization

→ Actors are coupled by data and not by wires

→ Possibilities for static analysis and runtime optimizations

Discussion and Future Work

Is being implemented in Kepler as COMAD 2.0

Efficient Execution / Static Analysis

- Parallelization on Scope via MapReduce (JCSS10)
- Shipping Optimization (ICDE09)
- Compilation to FLUX (ICDE10)

Support During WF-Design

- Actor-data dependencies
- Schema propagation and configuration-wizards

Expressiveness

- Branching merging
- While loops in WF Graph

Thank you.

Questions?