

# WINGS/Pegasus: Semantic Metadata Reasoning for Large Scientific Workflows

Jihie Kim  
Yolanda Gil  
Varun Ratnakar

USC Information Sciences Institute  
{jihie, gil, varunr}@isi.edu



“Semantic Metadata Generation for Large Scientific Workflows”

Jihie Kim, Yolanda Gil, and Varun Ratnakar, ISWC 2006.

<http://www.isi.edu/ikcap/scec-it/papers/Wings-metadata-ISWC-2006.pdf>

“Wings for Pegasus: A Semantic Approach to Creating Very Large Scientific Workflows”

Yolanda Gil, Varun Ratnakar, Ewa Deelman, Marc Spraragen, and Jihie Kim. (Under review.)

# Creation of Workflows in Layers of Increasing Detail

1. **Workflow Template (generic)**
  - Specifies executables and dataflow
  - No data specified, just their type
2. **Workflow Instance (user specific)**
  - Specifies data files for a given template
  - Logical file names, not physical file replicas
3. **Executable Workflow (actual run)**
  - Specifies physical locations of data files, hosts/pools for execution of jobs, and data movement jobs

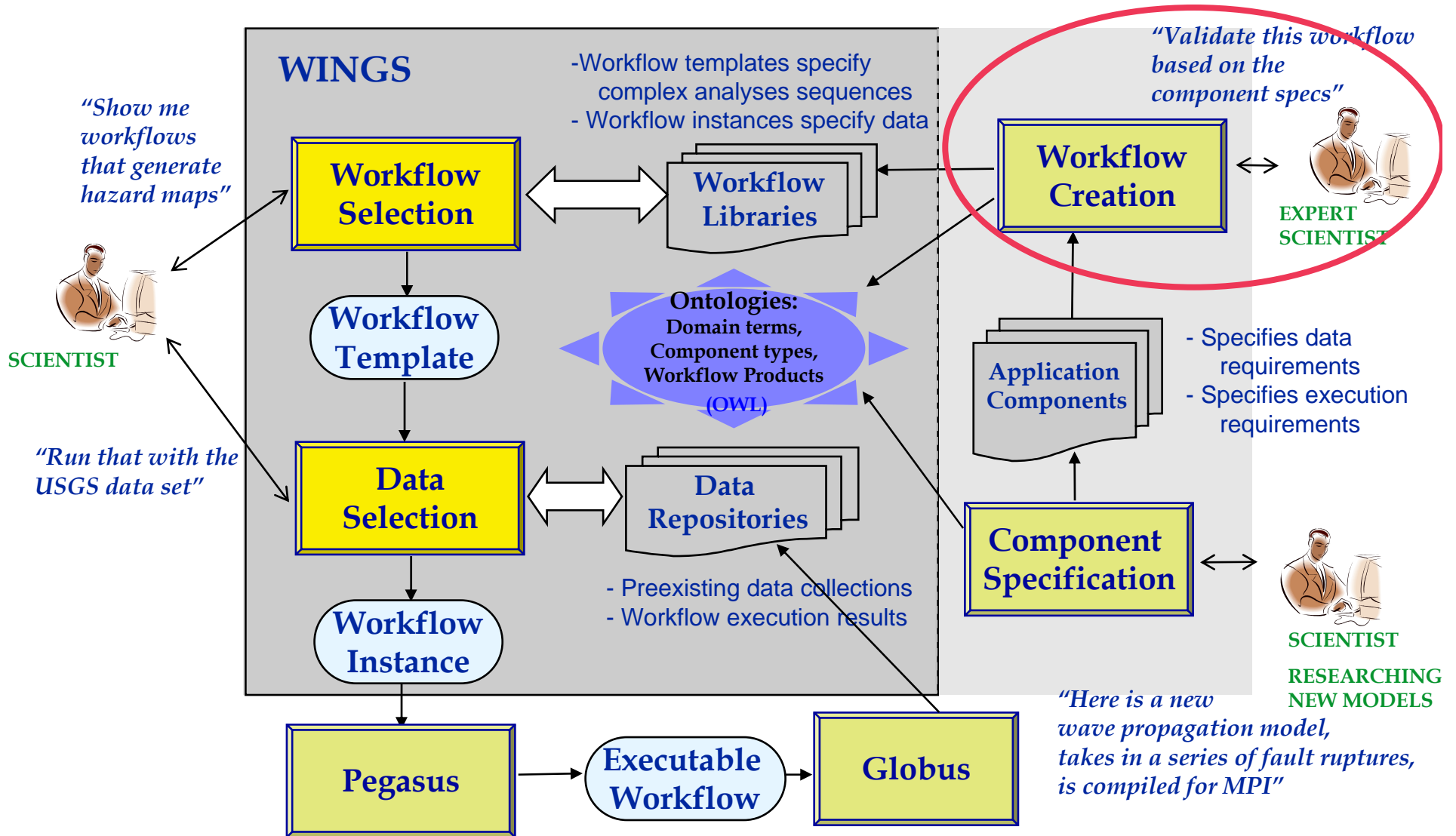
# The Process of Creating an Executable Workflow

User guided

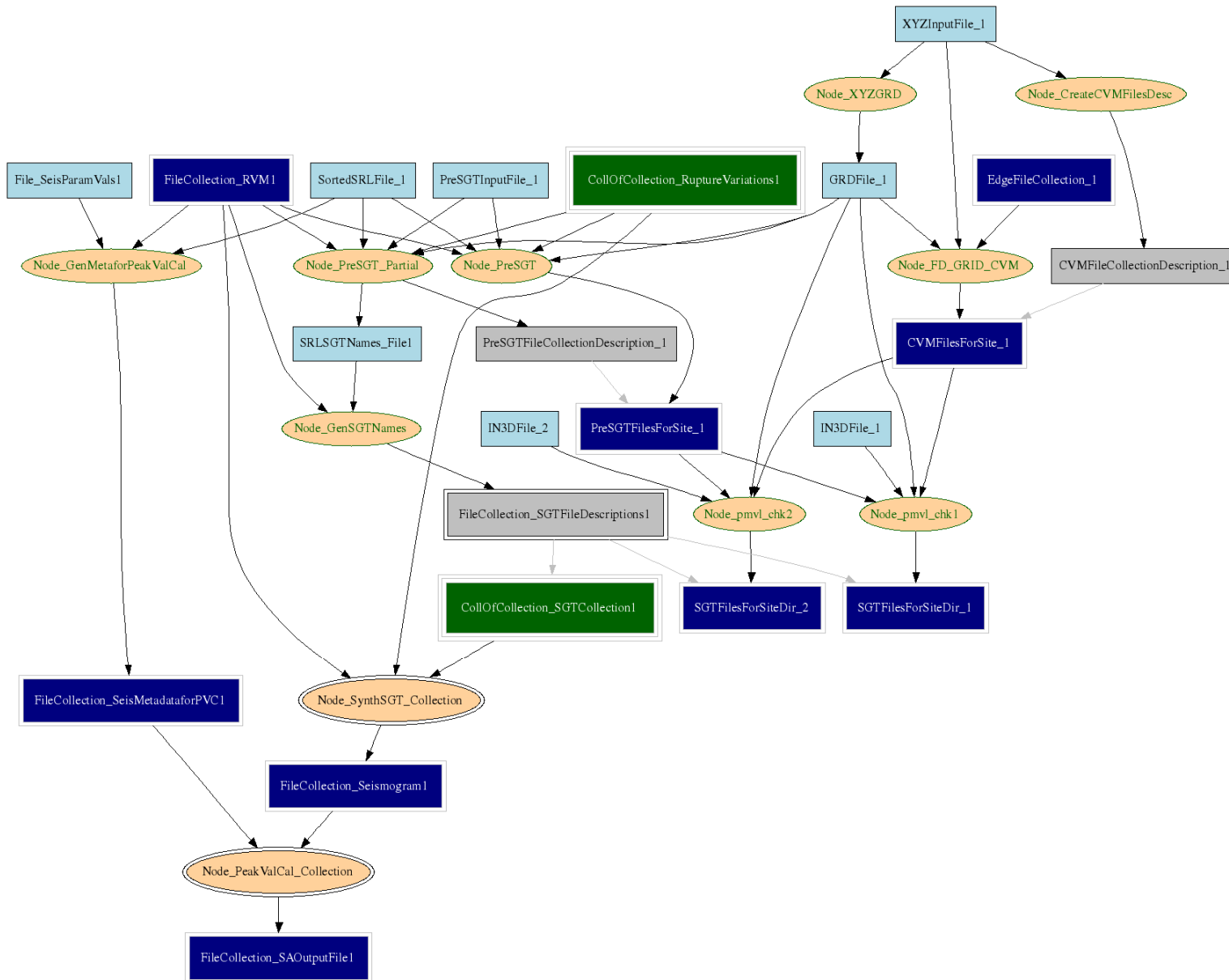
1. Creating a valid workflow template
  - Selecting application components and connecting inputs and outputs to specify data flow
  - Adding other steps for data conversions/transformations
2. Creating instantiated workflow
  - Providing input data to pathway inputs (logical assignments)
3. Creating executable workflow
  - Given requirements of each model, find and assign adequate resources for each model
  - Select physical locations for logical names
  - Include data movement steps, including data deposition steps

Automated

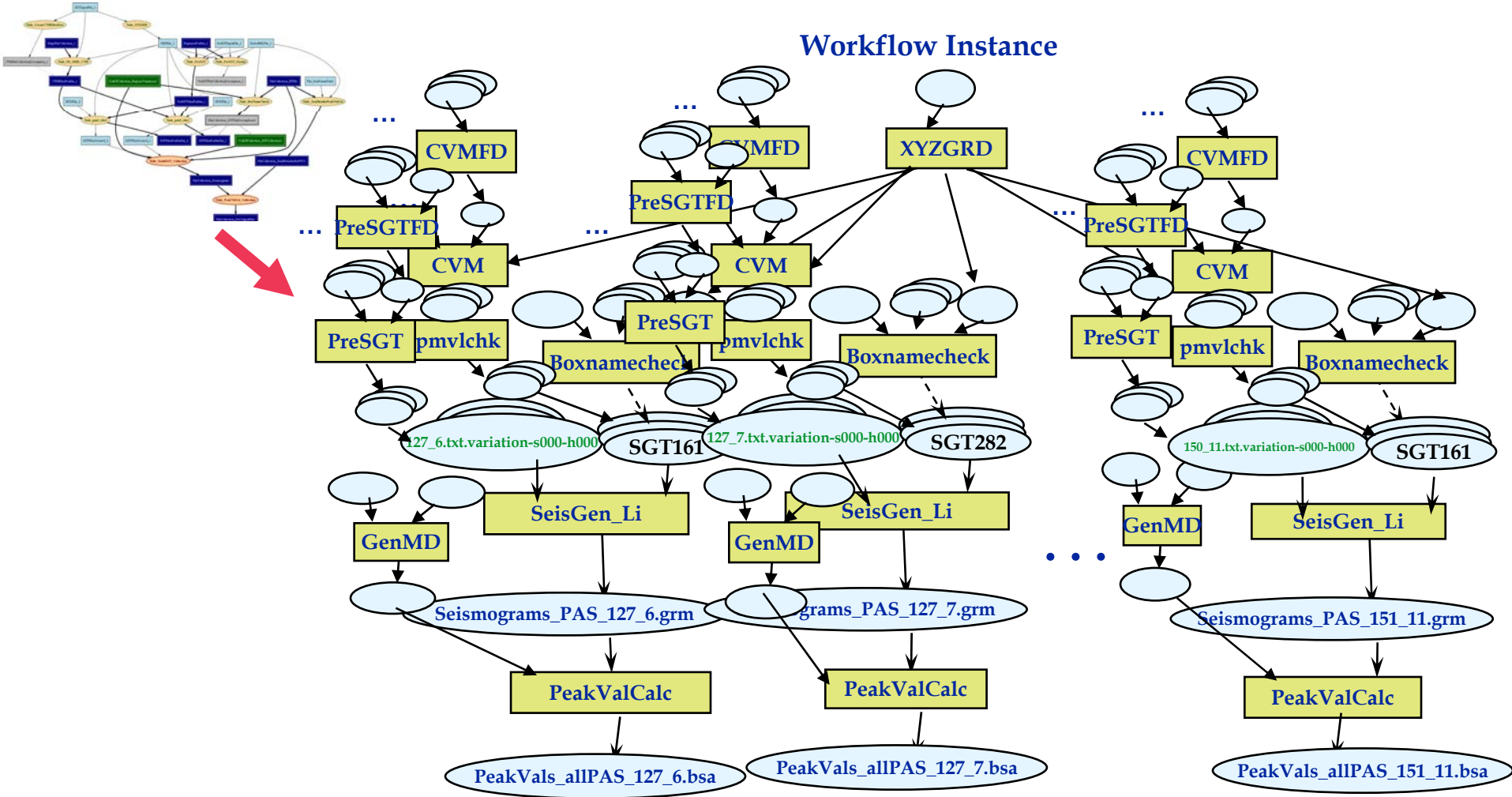
# WINGS/Pegasus: Workflow Instance Generation and Selection



# An example Workflow Template in Earthquake Science : Seismic Hazard Analysis



# Generating Metadata/Provenance *while Creating a Workflow Instance*

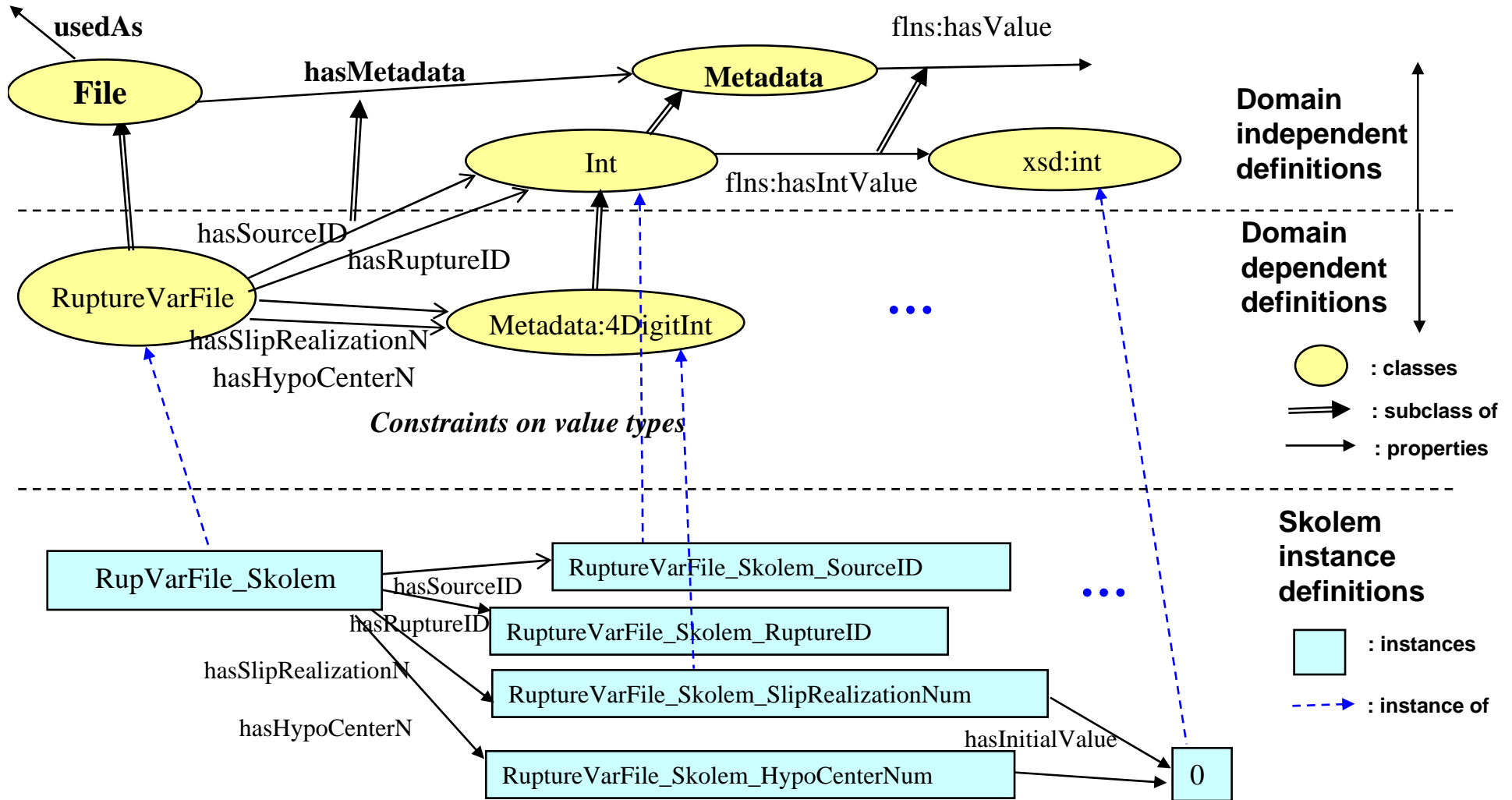


~4,000 ruptures,  
 >100,000 variations for a site,  
**wings**

# Metadata Constraints (in OWL ontology)

- Constraints on Files
  - metadata attributes: data types and default values  
e.g. simulation\_out\_timesamples of SeisParamValsFile should be an integer and the default value is 1801
- Constraints on collections and collection of collection
  - Type of each element
  - Relations between metadata of a collection and metadata of individual items  
e.g. Each rupture variation has the same source/rupture ids as the rupture variation collection
- Component level constraints on metadata attributes of input/output files or collections
  - Deriving metadata of output files from metadata of input files  
e.g. The output of PeakValCalc (SA output file) should have the same site name as the seismogram file
- Template level constraints on metadata attributes of files or collections
  - Input/output files of different components can have the same metadata  
e.g. The RVM collection input for SeismogramGen should have the same site name as the CollOfCollection rupture variations input
  - Checking number of items in collections  
e.g. number of RVM files and the number of rupture var collections should be equal

# Constraints on Files

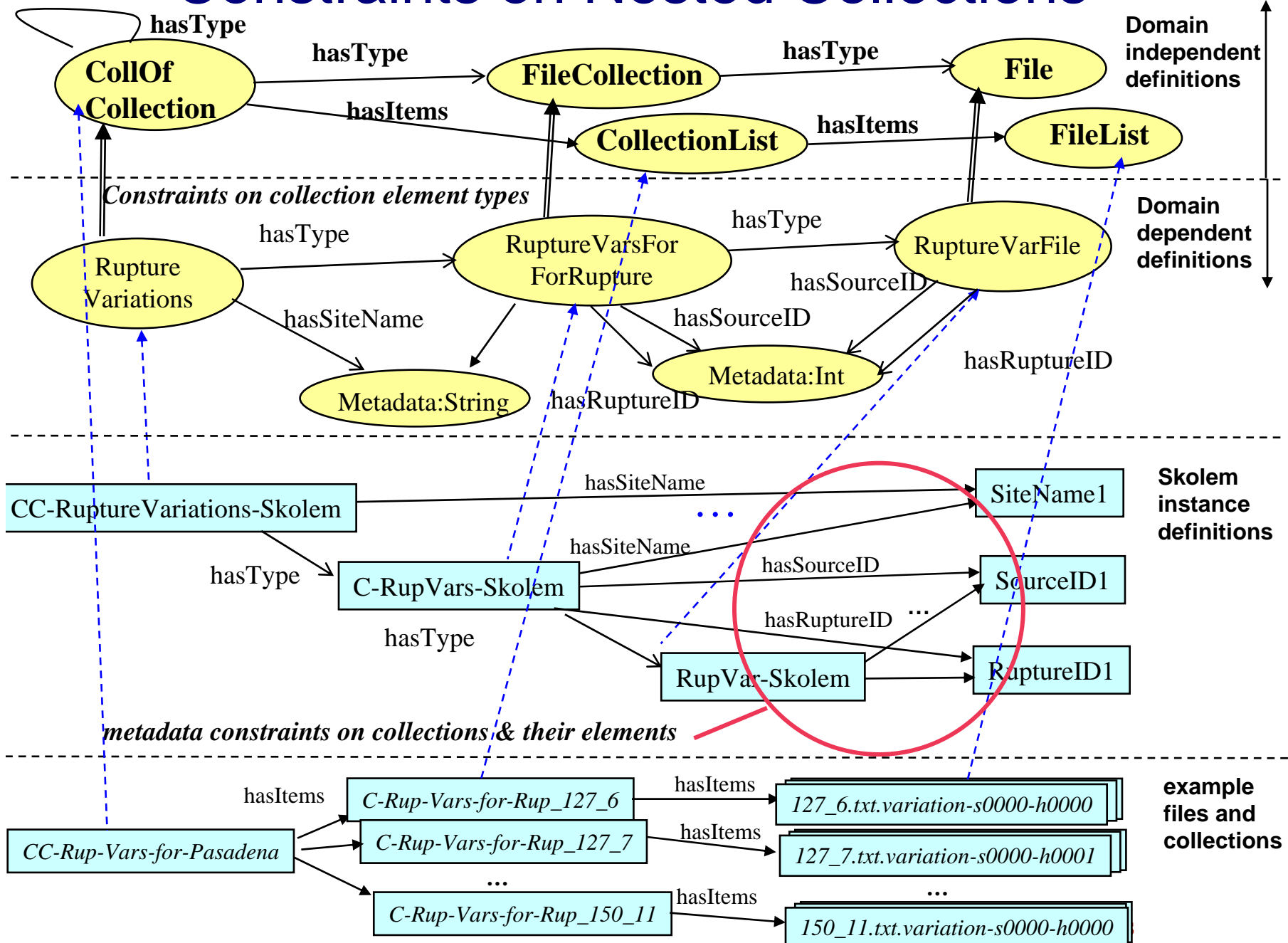


```

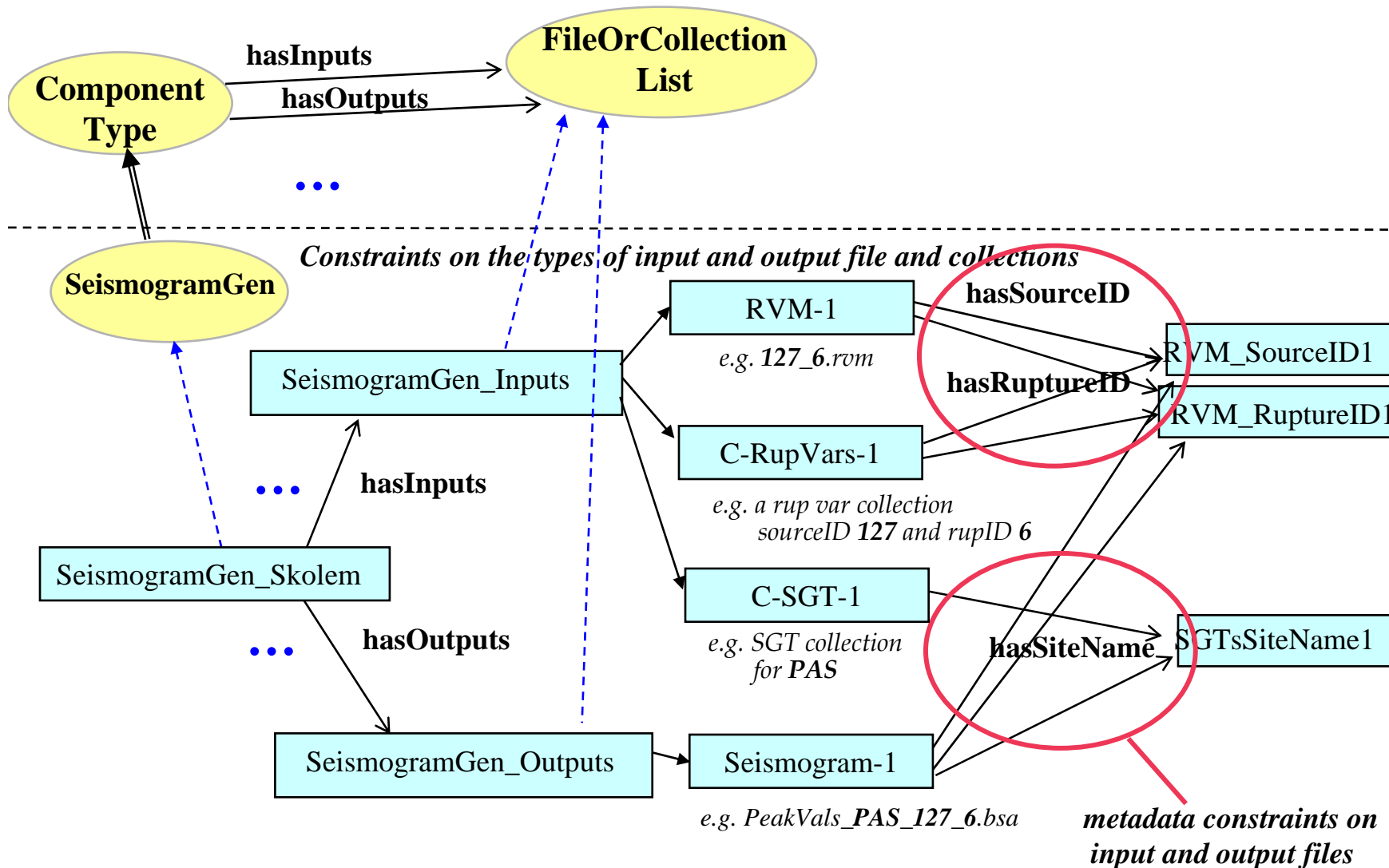
<RuptureVariationFile rdf:ID="RuptureVarFile_Skolem"> <rdf:type rdf:resource="&flns;FileSkolem"/>
  <hasRuptureID rdf:resource="#RuptureVarFile_Skolem_RuptureID"/>
  <hasSlipRealizationNumber rdf:resource="#RuptureVarFile_Skolem_SlipRealizationNum"/>
...</RuptureVariationFile>
<flns:Int rdf:ID="RuptureVariationFile_Skolem_RuptureID"/>
<FourDigitInt rdf:ID="RuptureVarFile_Skolem_SlipRealizationNum"> <flns:hasInitialValue rdf:datatype="&xsd:int">0</flns:hasInitialValue>
...</FourDigitInt>

```

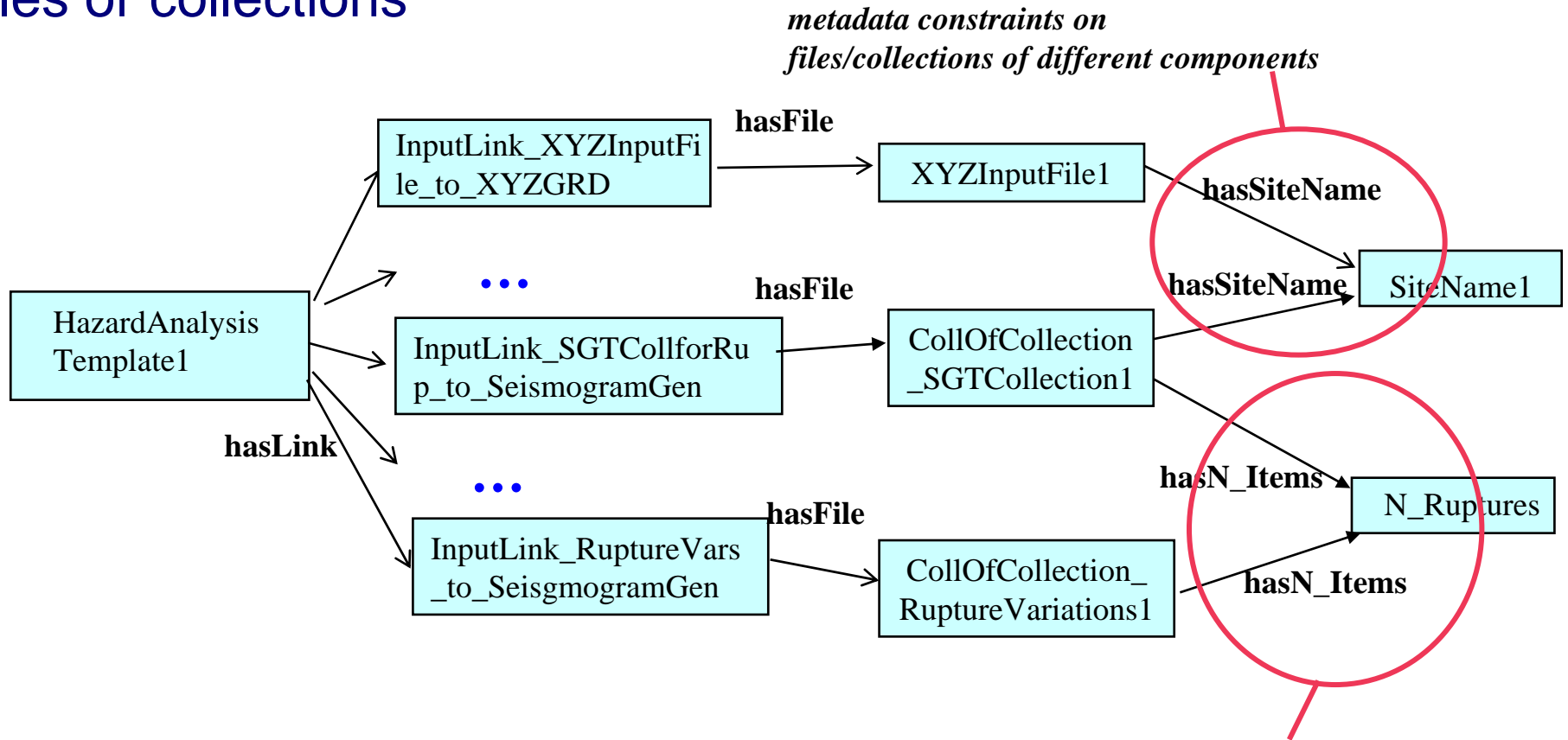
# Constraints on Nested Collections



# Component level constraints on metadata attributes of input/output files or collections



# Template level (global) constraints on metadata attributes of files or collections



*Constraints on number of elements in different collections*

```

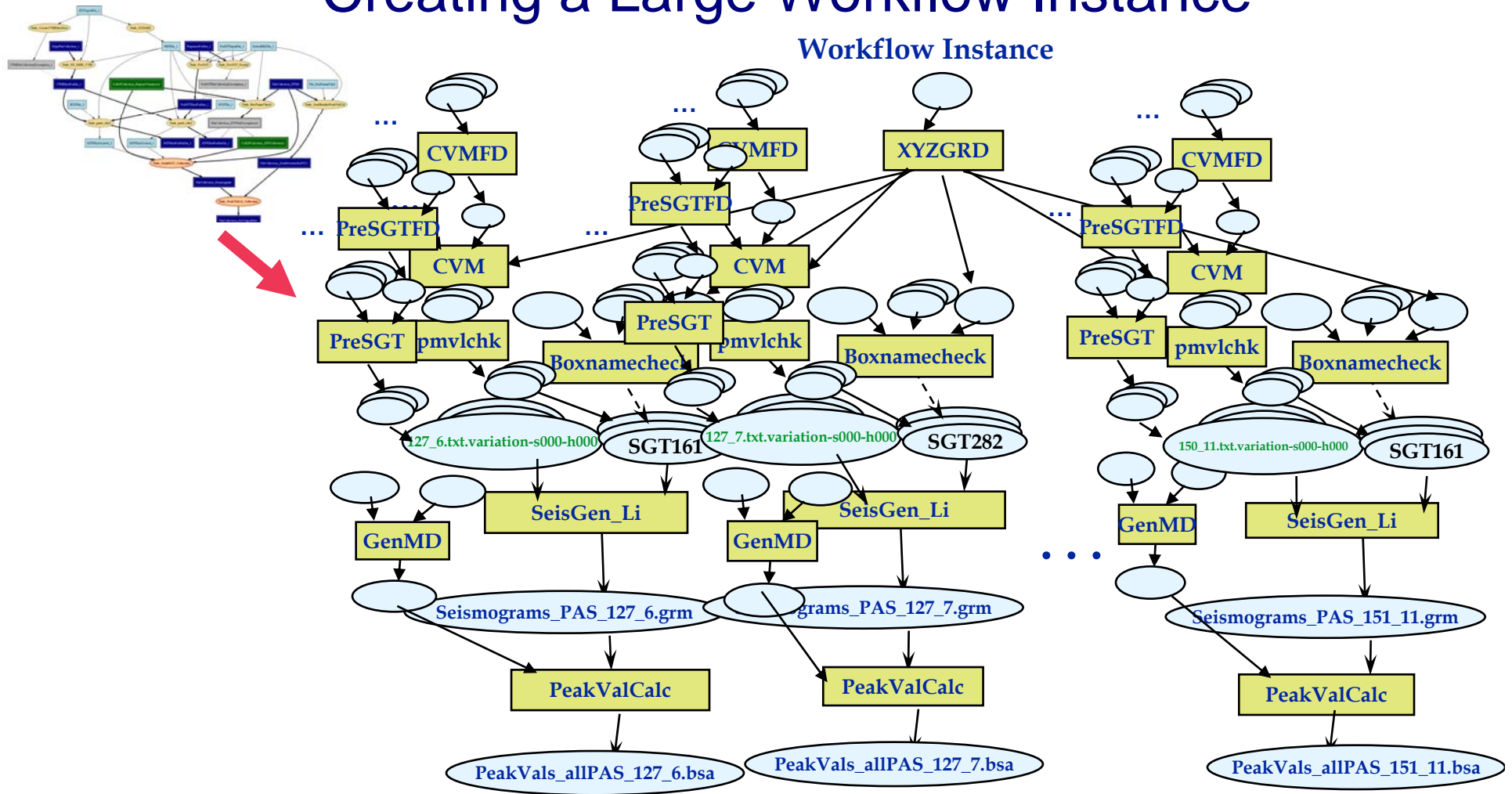
<scecfIns:XYZInputFile rdf:ID="XYZInputFile_1"> <scecfIns:hasSiteName rdf:resource="#SiteName_1"/> </scecfIns:XYZInputFile>
<scecfIns:SGTCollection rdf:ID="CollOfCollection_SGTCollection1"> <scecfIns:hasSiteName rdf:resource="#SiteName_1"/>
  <flns:hasN_items rdf:resource="#N_Ruptures"/> <flns:hasCollectionType
  rdf:resource="#SynthSGTInput_SGTCollectionforRupture"/> <flns:hasDescriptionFile
  rdf:resource="#FileCollection_SGTFileDescriptions1"/> </scecfIns:SGTCollection>
<scecfIns:RuptureVariations rdf:ID="CollOfCollection_RuptureVariations1"> <scecfIns:hasSiteName
  rdf:datatype="#xsd:string"/> </scecfIns:hasSiteName> <flns:hasN_items rdf:resource="#N_Ruptures"/> <flns:hasCollectionType
  rdf:resource="#RuptureVariationsforRupture_1"/> </scecfIns:RuptureVariations>
  
```

# Metadadata/Provenance Generation & Validation

## **Bind&ValidateWorkflow** (WorkflowTemplate wt, InputLinks ILinks)

1. Assign ILinks to LinksToProcess.
2. While LinksToProcess is not empty
  - 2.2. Remove one from LinksToProcess and assign it to L1.
  - 2.2. Let F1 be the link skolem for binding files or collections to L1.
  - 2.3. *If metadata for F1 should be generated from an execution of a component*
    - 2.3.1. *if the execution results are not available, continue.*  
;; *i.e. exclude this link in the sub-workflow*
  - 2.4. If any metadata of F1 depends on a link L2 that is not bound yet,
    - 2.4.1. mark L1 as a dependent of L2 and continue.
  - 2.5. If L1 is an input link,
    - 2.5.1. get metadata of the file from the user or a file server
    - 2.5.1. check consistencies with links that L1 depends on ;; **consistency check**
    - 2.5.2. check consistencies with existing bindings based on template-level constraints  
;; **consistency check**
    - 2.5.3. If any metadata are inconsistent, report inconsistency and return.
    - 2.5.4. Bind file/collection name and metadata to F1.
    - 2.5.5. If the file type for F1 is a collection, recursively get the metadata of its elements
  - 2.6. Else (i.e. L1 is InOutLink or OuputLink)
    - 2.6.1. Generate file names and metadata base on the definition of the depending links.  
;; **metadata propagation**
  - 2.7. For each link L2 that is dependent on l1,
    - 2.7.1. if all the links that L2 is depending on are bound, put L2 in LinksToProcess.
  - 2.8. If L1 is an output link, continue.
  - 2.9. Else (L1 is InputLink or InOutLink)
    - 2.9.1. If all the inputs to the destination node (i.e. the component that L1 provides an input to) have been bound,
      - 2.9.1.1. Add all the OutputLinks and InOutLinks from the destination node to the LinksToProcess.

# Generating Metadata/Provenance while Creating a Large Workflow Instance



	Creation time	Number of files	Number of OWL individuals
Sub workflow	7 mins 59 sec	15,888	322,473
Full workflow	22 mins 52 sec	117,379	<b>2,001,972</b>

4,000 ruptures,  
100,000 variations for a site,

# Technical Contributions: Semantic Metadata Approach to Creating Large Scientific Workflows

- Semantic representations of workflow templates to express **repetitive computational structures and collections**
- Expanding template to instances that **orchestrate large amounts of computations** reflecting the workflow template structure
- Generating appropriate **metadata descriptions for all the new data** created during execution and full elaboration of workflow specs
- **Ensuring validity** of workflow instance (Bind&Validate algorithm)
  - Keeping track of constraints on dataset used, including global constraints among multiple components as well as local constraints within individual components.
- Mapping equivalent datasets, detecting pre-existing intermediate data, and **prevent unnecessary execution** of workflow parts when datasets already exist.

<http://www.isi.edu/ikcap/scec-it/>