

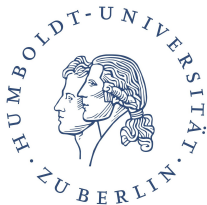
Cuneiform

A Functional Workflow Language Implementation in Erlang

Jörgen Brandt

Humboldt-Universität zu Berlin

2015-12-01



Cuneiform

A Functional Language for Large Scale Scientific Data Analysis



Cuneiform

A Functional Language for Large Scale Scientific Data Analysis

- Black-box operator model

Pro: Operators can be any piece of software



Cuneiform

A Functional Language for Large Scale Scientific Data Analysis

- Black-box operator model

Pro: Operators can be any piece of software

- Black-box data model

Pro: Input and Output data can be anything



Cuneiform

A Functional Language for Large Scale Scientific Data Analysis

- Black-box operator model

Pro: Operators can be any piece of software

- Black-box data model

Pro: Input and Output data can be anything

- Features of advanced workflow languages

Abstractions, lists, operations on lists, conditions



Cuneiform

A Functional Language for Large Scale Scientific Data Analysis

- Black-box operator model

Pro: Operators can be any piece of software

- Black-box data model

Pro: Input and Output data can be anything

- Features of advanced workflow languages

Abstractions, lists, operations on lists, conditions

- Light-weight Foreign Function Interface (FFI)

Wrapping in R, Matlab, Octave, Python, Lisp, Perl, Bash





Cuneiform

A Functional Language for Large Scale Scientific Data Analysis

- Black-box operator model

Pro: Operators can be any piece of software

- Black-box data model

Pro: Input and Output data can be anything

- Features of advanced workflow languages

Abstractions, lists, operations on lists, conditions

- Light-weight Foreign Function Interface (FFI)

Wrapping in R, Matlab, Octave, Python, Lisp, Perl, Bash

- Automatic parallelization

Scalability with large data sets

Motivation

“New hardware is increasingly parallel, so new programming languages must support concurrency or they will die.”

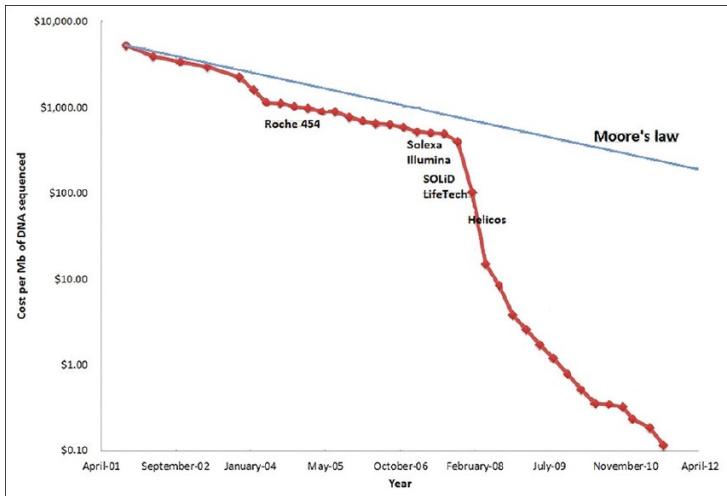
Joe Armstrong



“New Hardware”



DNA Sequencing is becoming cheap



Decentralized software development

Browse data - SEQwiki - Mozilla Firefox

seqanswers.com/wiki/Special:BrowseData

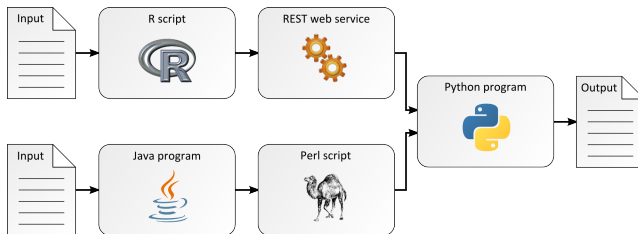
Bioinformatics methods:

- Ab-initio gene prediction (2) · Adapter Removal (software) (5) · **Alignment (61)** · Alignment Analysis (6) · Alignment viewer (7) · Allele calling (1) · Allelic imbalance (1) · **Alternative Splicing (5)** · Analysis Pipeline (2) · **Annotation (17)** · **Assembly (74)** · Assembly QC (3) · Assembly editing (2) · Assembly quality evaluation (2) · Assembly validation (3) · **Assembly visualization (11)** · BLAST (1) · **Basecaller (12)** · Basespace (2) · Biological Contextualization (2) · Biological interpretation (1) · Bisulfite SNP calling (2) · **Bisulfite mapping (16)** · Bloom filters (1) · Burrows-Wheeler (3) · ChIP-Seq analysis (3) · ChIP-seq differential analysis (1) · ChIP seq (1) · Chromatin motif finding (1) · Chromatogram management (1) · Chromatogram viewer (2) · Classification (1) · **Clustering (5)** · Clustering and alignment (1) · Collapsing Methods (1) · **Colorspace (11)** · Command line tool wrappers (3) · Community Analysis (1) · Comparative genomics (2) · Contaminant filtering (3) · **Conversion (6)** · Copy number estimation (3) · Cost estimation (1) · DNase I footprinting (1) · **Data compression (9)** · Database (5) · Database interface (1) · Database submission preparation (2) · De-novo assembly (3) · **De Bruijn graph (10)** · De novo Assembly (1) · Deduplication (1) · Differential Binding (1) · Differential binding sites (1) · **Differential expression (6)** · Differentially expressed gene identification (9) · Differentially methylated regions identification and annotation (1) · Digital genomic footprinting (2) · Downstream analysis (1) · Empirical Bayes (1) · **Error correction (13)** · Exome analysis (3) · **Expectation Maximization (4)** · Expression profiling (3) · **FM-Index (5)** · **Filtering (12)** · **Format conversion (6)** · Functional analysis (1) · Fusion genes (1) · GPU (5) · Gap extension (1) · Gene Set Testing (1) · Gene expression analysis (2) · Gene fusions discovery (2) · Gene ontology (1) · Gene ontology analysis (2) · Gene set enrichment (1) · Gene set enrichment analysis (1) · General bioinformatics (2) · **Genetic variation annotation (5)** · **Genome Alignment (2)** · Genome Indexing (1) · **Genome browser (10)** · Genome wide association studies (1) · Genomic correlations (2) · Genomic overlaps (1) · Genomic region matching (2) · Genomics (2) · Genotyping (1) · Gibbs motif sample (1) · Graph reduction (1) · HLA typing (1) · **Hadoop (8)** · **Haplotype reconstruction (4)** · Hash Table Based (1) · Heatmaps (2) · **Hidden Markov Model (13)** · Hybrid assembly (2) · InDel discovery (3) · Integrated Solution (4) · K-mer analysis (8) · LIMS (2) · Learning algorithm (1) · Localized reassembly/realignment (4) · MCMC (1) · Machine Learning (2) · **MapReduce (7)** · Mapping (127) · Mapping and variant calling (1) · **Methylation Calling (4)** · Methylation analysis (1) · MIRNA Prediction (1) · MIRNA analysis (Ref and Ab-initio) (1) · MIRNA profiling (1) · Micro assembly (2) · Mixture model (1) · **Motif analysis (4)** · Motif comparison (1) · Motif detection (2) · **Motif discovery (5)** · Motif scanning (1) · Multiple sequence alignment viewer (4) · Mutation detection (2) · Myers Bitvector Algorithm (1) · Normalization (4) · Novel gene discovery (1) · OLC (1) · PCR Primer Design (2) · Paired End (3) · Pathway analysis (1) · Peak-pair calling (2) · **Peak calling (18)** · Peak detection (4) · Phase pattern prediction (1) · Pipeline Management (2) · Pooled samples (5) · Post-analysis (1) · Preprocessing (3) · Primer removal (1) · Profiles (1) · Profiling short tandem repeats from short reads (1) · **Programming Library (18)** · Protein Binding Peak Detection (2) · Protocol Management (1) · QC (4) · **Quality Control (8)** · Quality Trimming (1) · Quality assessment (3) · RNA-Seq analysis (4) · RNA filtering (1) · **Read Alignment (7)** · Read alignment (1) · Read depth analysis (3) · Read mapping (8) · Read pre-processing (5) · Read storage (1) · Read summarization (2) · Regression. (1) · SAMtools (1) · **SNP calling (6)** · SNP set enrichment analysis (1) · SWIFT Filter (1) · Sample Barcoding (8) · Sample Tracking (1) · Scaffolding (8) · Segmentation (1) · Sequence alignment (2) · Sequence alignment comparison (1) · Sequence alignment to a reference genome (1) · **Sequence analysis (16)** · Sequence annotation (2) · Sequence motif discovery (1) · Sequence parsing (3) · Sequence variation analysis (2) · **Sequencing Quality Control (25)** · Signal (1) · **Simulation (12)** · Smith-Waterman (3) · Somatic variant calling (4) · Species frequency estimation (1) · Split-read (3) · Statistical Modelling (2) · Statistical testing (7) · Statistics (9) · Structural variation discovery (4) · Suffix arrays (2) · Targeted de novo



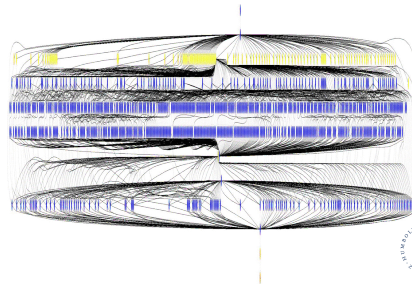
Scientific Workflow Systems

Scientific Workflow Systems

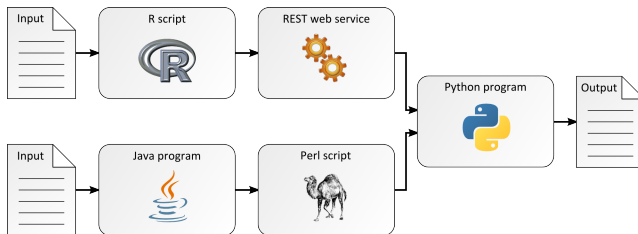


Workflows as DAGs

- Scientific Workflows are DAGs

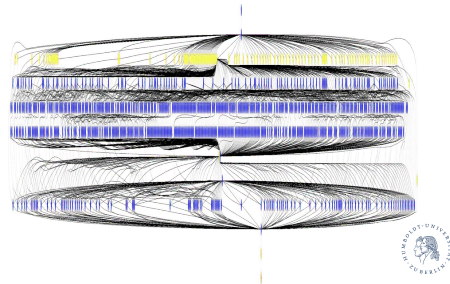


Scientific Workflow Systems

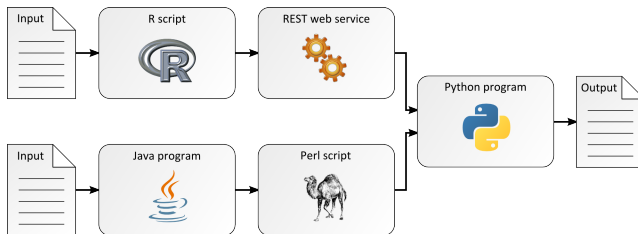


Workflows as DAGs

- Scientific Workflows are DAGs
- Nodes are tasks

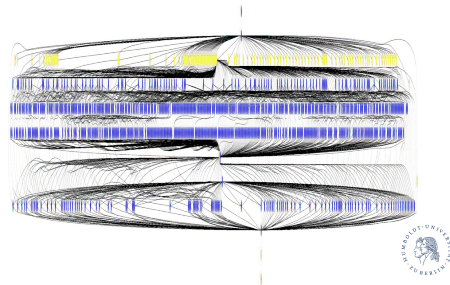


Scientific Workflow Systems

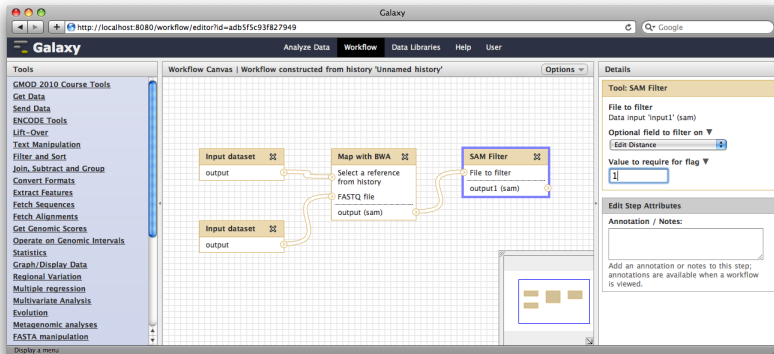


Workflows as DAGs

- Scientific Workflows are DAGs
- Nodes are tasks
- Edges are data dependencies



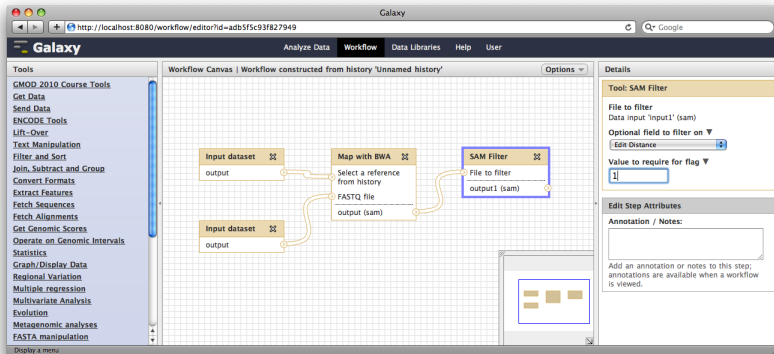
Example: Galaxy Workflow System



Focus on

■ Usability

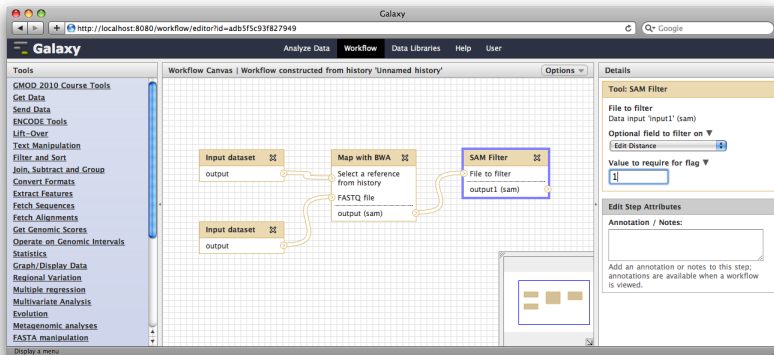
Example: Galaxy Workflow System



Focus on

- Usability
- Integration of tools/libraries

Example: Galaxy Workflow System



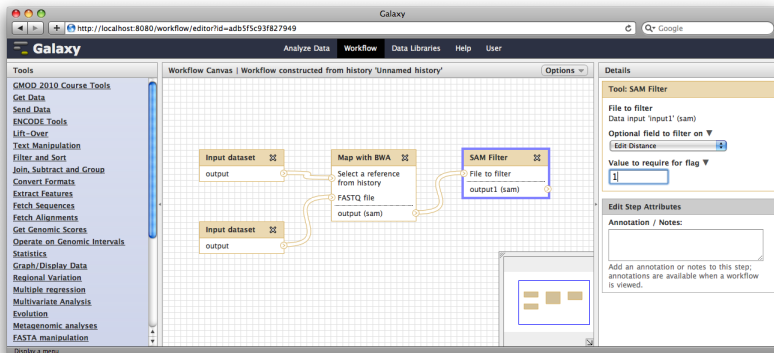
Focus on

- Usability
- Integration of tools/libraries

- Systematic documentation



Example: Galaxy Workflow System



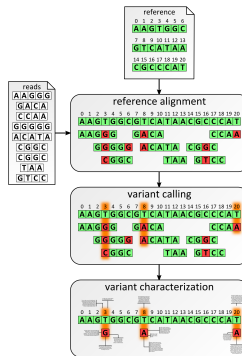
Focus on

- Usability
- Integration of tools/libraries

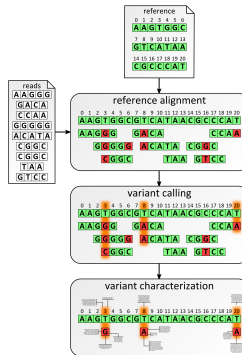
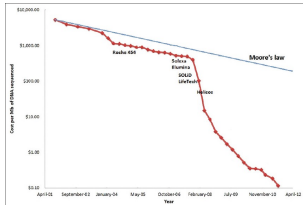
- Systematic documentation
- Reproducibility



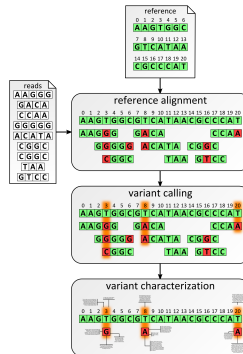
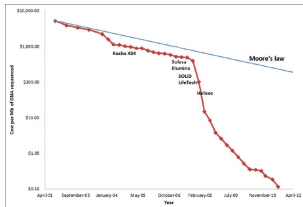
The Next Generation Sequencing use case

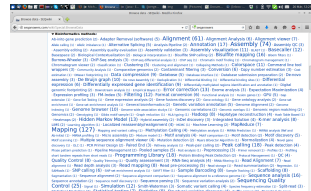


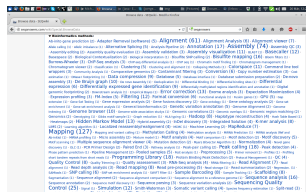
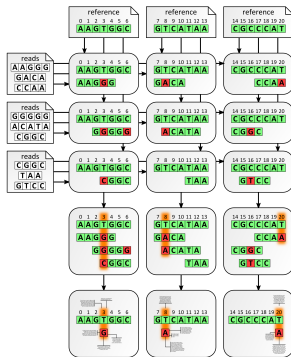
The Next Generation Sequencing use case



The Next Generation Sequencing use case







Desired Features in a Language

Is there a language that is ...

Desired Features in a Language

Is there a language that is ...

- Like a workflow language

So we can integrate all the tools

Desired Features in a Language

Is there a language that is ...

- Like a workflow language

So we can integrate all the tools

- Like MapReduce

So we can derive parallelism and distribute the work

Desired Features in a Language

Is there a language that is . . .

- Like a workflow language

So we can integrate all the tools

- Like MapReduce

So we can derive parallelism and distribute the work

- Like a functional programming language

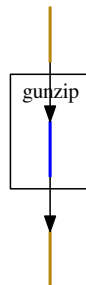
So we can write arbitrary programs using lists and operations on lists

Cuneiform example

```
deftask gunzip( out( File ) : gz( File ) )in bash *{  
  gzip -c -d $gz > $out  
}*
```

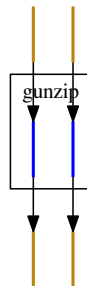
Cuneiform example

```
deftask gunzip( out( File ) : gz( File ) )in bash *{  
    gzip -c -d $gz > $out  
}*  
  
gunzip(  
    gz: 'myarchive1.gz'  
);
```



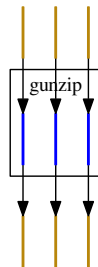
Cuneiform example

```
deftask gunzip( out( File ) : gz( File ) )in bash *{  
    gzip -c -d $gz > $out  
}*  
  
gunzip(  
    gz: 'myarchive1.gz' 'myarchive2.gz'  
);
```



Cuneiform example

```
deftask gunzip( out( File ) : gz( File ) )in bash *{  
  gzip -c -d $gz > $out  
}*  
  
gunzip(  
  gz: 'myarchive1.gz' 'myarchive2.gz' 'myarchive3.gz'  
);
```

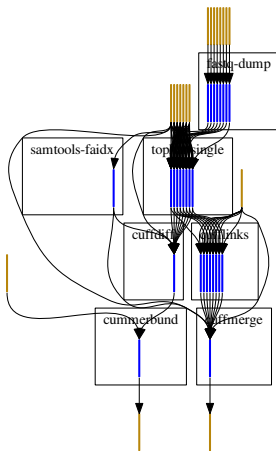


Workflow Implementations Available

Available example workflows:

- Variant calling

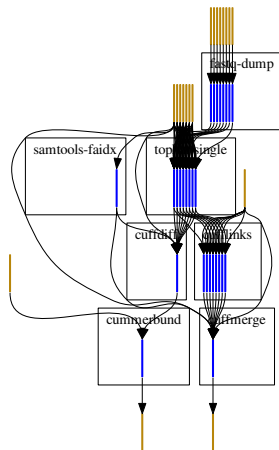
<https://www.github.com/joergen7/variant-call>



Workflow Implementations Available

Available example workflows:

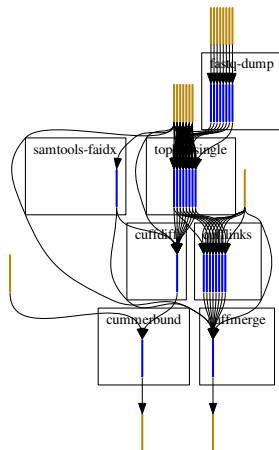
- Variant calling
<https://www.github.com/joergen7/variant-call>
- Methylation
<https://www.github.com/joergen7/methylation>



Workflow Implementations Available

Available example workflows:

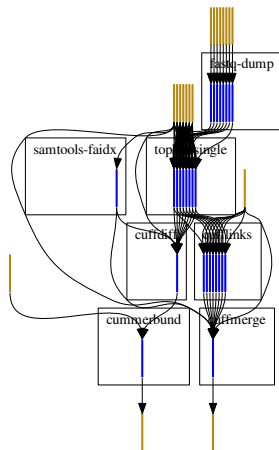
- Variant calling
<https://www.github.com/joergen7/variant-call>
- Methylation
<https://www.github.com/joergen7/methylation>
- RNA-Seq <https://www.github.com/joergen7/rna-seq>



Workflow Implementations Available

Available example workflows:

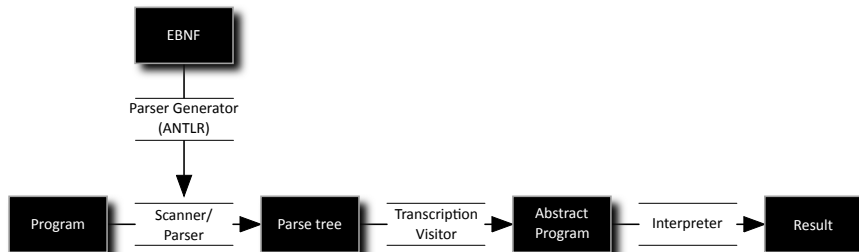
- Variant calling
<https://www.github.com/joergen7/variant-call>
- Methylation
<https://www.github.com/joergen7/methylation>
- RNA-Seq <https://www.github.com/joergen7/rna-seq>
- etc (ChIP-Seq, miRNA detection, consensus prediction, ...)



Cuneiform Operational Semantics

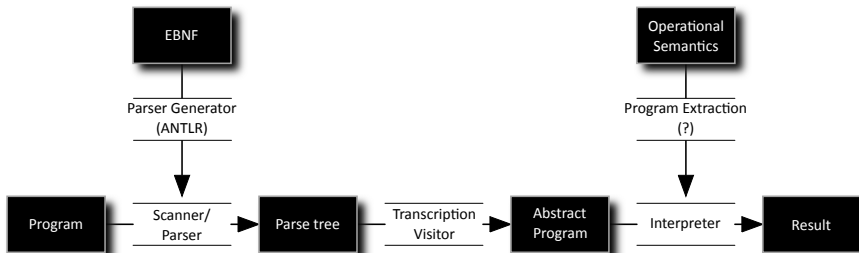
Program Interpretation in Cuneiform

Current design in Java implementation:



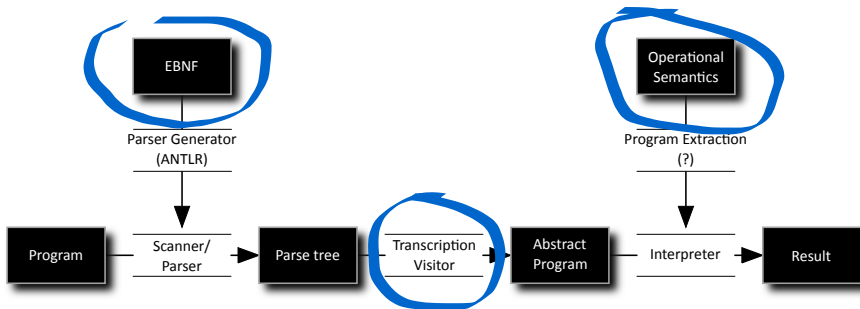
Program Interpretation in Cuneiform

Designated design but never implemented:



Program Interpretation in Cuneiform

Designated design in Erlang:



The eval Function

fun *eval*(*Expr*, ρ , *GetFuture*, *Global*, *Fin*) \rightarrow *Result* **when**

<i>Expr</i>	::	<i>expr</i>
ρ	::	<i>string</i> \Rightarrow <i>expr</i>
<i>GetFuture</i>	::	<i>fun</i>
<i>Global</i>	::	<i>string</i> \Rightarrow <i>lam</i>
<i>Fin</i>	::	<i>id</i> \Rightarrow <i>expr</i>
<i>Result</i>	::	<i>expr</i>

The eval Function

fun *eval*(*Expr*, ρ , *GetFuture*, *Global*, *Fin*) \rightarrow *Result* **when**

Expr :: *expr*

ρ :: *string* \Rightarrow *expr*

GetFuture :: *fun*

Global :: *string* \Rightarrow *lam*

Fin :: *id* \Rightarrow *expr*

Result :: *expr*

Expr The expression to be evaluated

The eval Function

fun *eval*(*Expr*, ρ , *GetFuture*, *Global*, *Fin*) \rightarrow *Result* **when**

Expr :: *expr*

ρ :: *string* \Rightarrow *expr*

GetFuture :: *fun*

Global :: *string* \Rightarrow *lam*

Fin :: *id* \Rightarrow *expr*

Result :: *expr*

Expr The expression to be evaluated

ρ Current scope

The eval Function

fun *eval*(*Expr*, ρ , *GetFuture*, *Global*, *Fin*) \rightarrow *Result* **when**

Expr :: *expr*

ρ :: *string* \Rightarrow *expr*

GetFuture :: *fun*

Global :: *string* \Rightarrow *lam*

Fin :: *id* \Rightarrow *expr*

Result :: *expr*

Expr The expression to be evaluated

ρ Current scope

GetFuture A function returning a future for a foreign task application

The eval Function

fun *eval*(*Expr*, ρ , *GetFuture*, *Global*, *Fin*) \rightarrow *Result* **when**

<i>Expr</i>	::	<i>expr</i>
ρ	::	<i>string</i> \Rightarrow <i>expr</i>
<i>GetFuture</i>	::	<i>fun</i>
<i>Global</i>	::	<i>string</i> \Rightarrow <i>lam</i>
<i>Fin</i>	::	<i>id</i> \Rightarrow <i>expr</i>
<i>Result</i>	::	<i>expr</i>

Expr The expression to be evaluated

ρ Current scope

GetFuture A function returning a future for a foreign task application

Global Task definitions



The eval Function

fun *eval*(*Expr*, ρ , *GetFuture*, *Global*, *Fin*) \rightarrow *Result* **when**

<i>Expr</i>	::	<i>expr</i>
ρ	::	<i>string</i> \Rightarrow <i>expr</i>
<i>GetFuture</i>	::	<i>fun</i>
<i>Global</i>	::	<i>string</i> \Rightarrow <i>lam</i>
<i>Fin</i>	::	<i>id</i> \Rightarrow <i>expr</i>
<i>Result</i>	::	<i>expr</i>

Expr The expression to be evaluated

ρ Current scope

GetFuture A function returning a future for a foreign task application

Global Task definitions

Fin Results of foreign task applications



The eval Function

fun *eval*(*Expr*, ρ , *GetFuture*, *Global*, *Fin*) \rightarrow *Result* **when**

<i>Expr</i>	::	<i>expr</i>
ρ	::	<i>string</i> \Rightarrow <i>expr</i>
<i>GetFuture</i>	::	<i>fun</i>
<i>Global</i>	::	<i>string</i> \Rightarrow <i>lam</i>
<i>Fin</i>	::	<i>id</i> \Rightarrow <i>expr</i>
<i>Result</i>	::	<i>expr</i>

Expr The expression to be evaluated

ρ Current scope

GetFuture A function returning a future for a foreign task application

Global Task definitions

Fin Results of foreign task applications

Result The result of evaluation (may contain futures)



```
eval(Expr,  $\rho$ , GetFuture, Global, Fin)  $\rightarrow$   
  Next = step(Expr,  $\rho$ , GetFuture, Global, Fin)  
  case Next of  
    Expr  $\rightarrow$  Expr  
    -  $\rightarrow$  eval(Next,  $\rho$ , GetFuture, Global, Fin)  
  end
```


$eval(Expr, \rho, GetFuture, Global, Fin) \rightarrow$
 $Next = step(Expr, \rho, GetFuture, Global, Fin)$
case $Next$ **of**
 $Expr \rightarrow Expr$
 $- \rightarrow eval(Next, \rho, GetFuture, Global, Fin)$
end

- single step is computed

```
eval(Expr,  $\rho$ , GetFuture, Global, Fin)  $\rightarrow$   
  Next = step(Expr,  $\rho$ , GetFuture, Global, Fin)  
  case Next of  
    Expr  $\rightarrow$  Expr  
    -       $\rightarrow$  eval(Next,  $\rho$ , GetFuture, Global, Fin)  
  end
```

- single step is computed
- If the step has no effect evaluation terminates

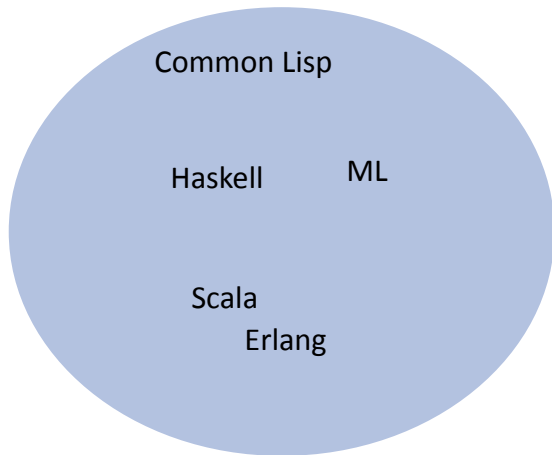
```
fun eval(Expr,  $\rho$ , GetFuture, Global, Fin)  $\rightarrow$   
  Next = step(Expr,  $\rho$ , GetFuture, Global, Fin)  
  case Next of  
    Expr  $\rightarrow$  Expr  
    -  $\rightarrow$  eval(Next,  $\rho$ , GetFuture, Global, Fin)  
  end
```

- single step is computed
- If the step has no effect evaluation terminates
- Otherwise eval is called recursively

Languages Suitable for Operational Semantics

Choosing a language

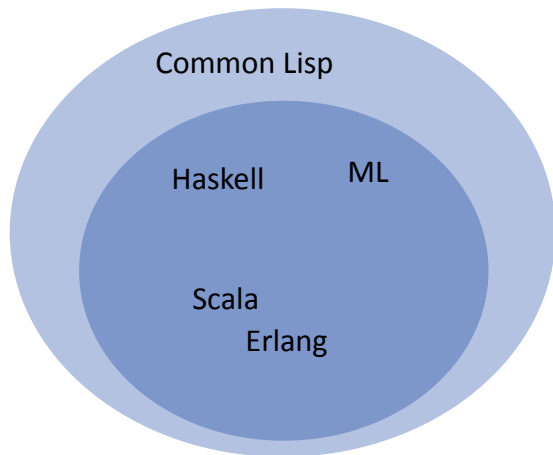
- Functional Language



Formalisms Suitable for Operational Semantics

Choosing a language

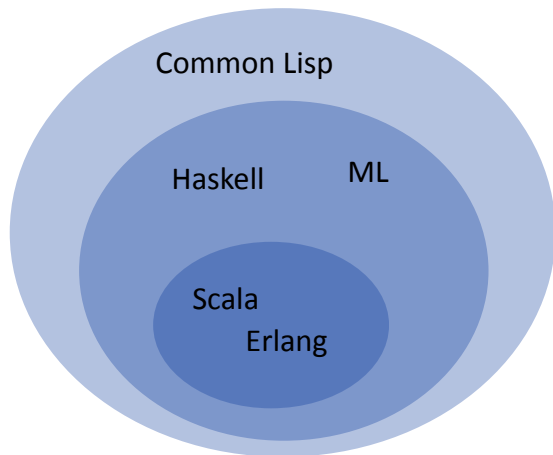
- Functional Language
- With Pattern Matching



Formalisms Suitable for Operational Semantics

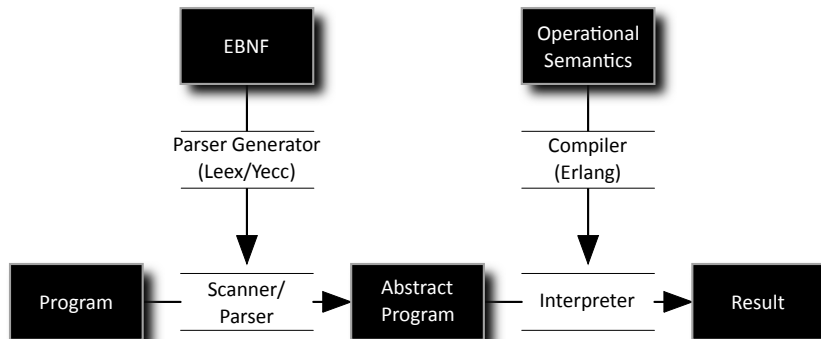
Choosing a language

- Functional Language
- With Pattern Matching
- Concurrency Orientation



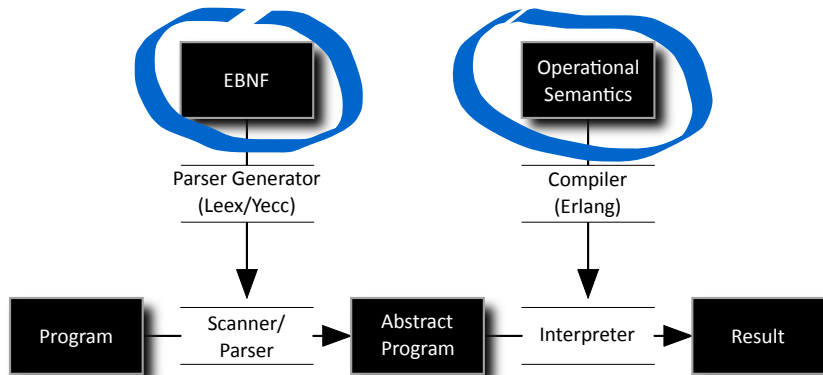
Program Interpretation in Cuneiform

New design:



Program Interpretation in Cuneiform

New design:



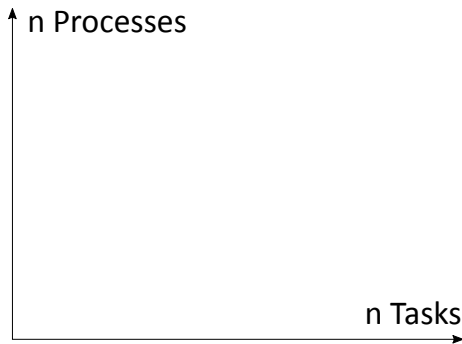
Fault Tolerance

Two types of complexity



Distributed applications can be complex in two independent ways:

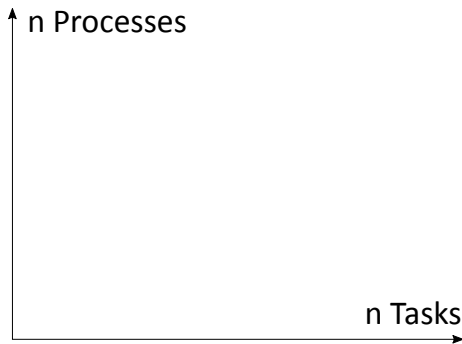
Two types of complexity



Distributed applications can be complex in two independent ways:

- in the number of processes involved to compute one task
the more system components the more likely one component fails

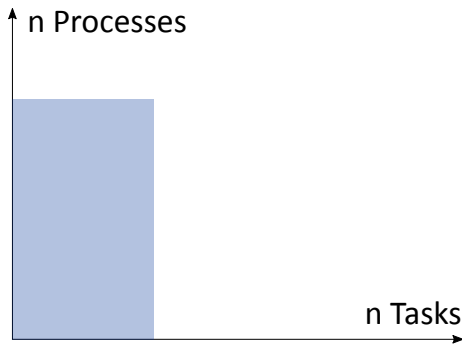
Two types of complexity



Distributed applications can be complex in two independent ways:

- in the number of processes involved to compute one task
the more system components the more likely one component fails
- in the number of tasks contributing to a workflow
the more tasks the more likely one task fails

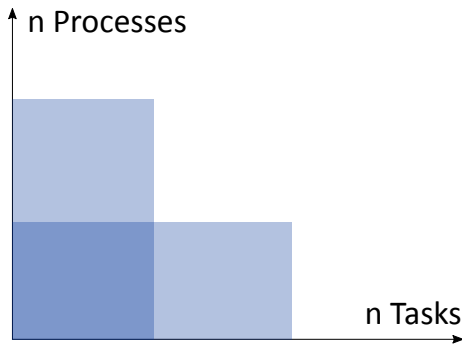
Two types of complexity



Distributed applications can be complex in two independent ways:

- in the number of processes involved to compute one task
the more system components the more likely one component fails
- in the number of tasks contributing to a workflow
the more tasks the more likely one task fails

Two types of complexity



Distributed applications can be complex in two independent ways:

- in the number of processes involved to compute one task
the more system components the more likely one component fails
- in the number of tasks contributing to a workflow
the more tasks the more likely one task fails

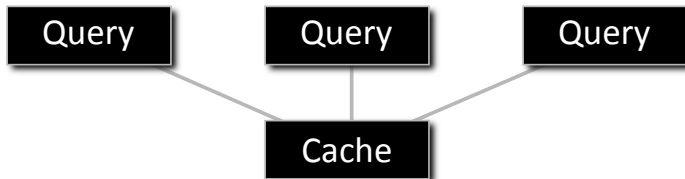
Distributed Application: Workflow System

Query

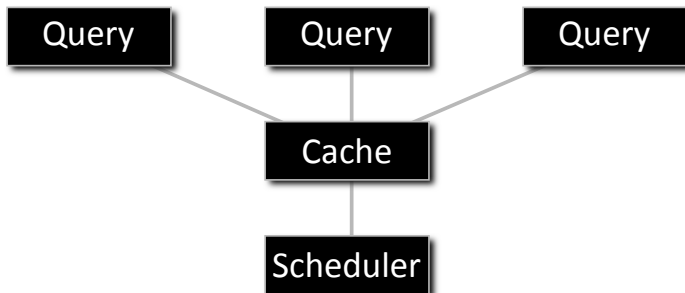
Query

Query

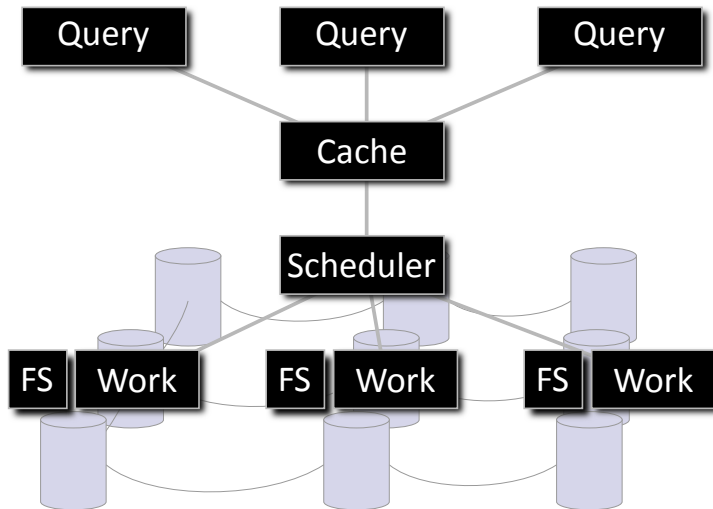
Distributed Application: Workflow System



Distributed Application: Workflow System



Distributed Application: Workflow System



How to achieve fault tolerance when

How to achieve fault tolerance when

- Workflow systems are complex in both
 - Number of processes involved in computing one task
 - Number of tasks in one workflow

so failures are likely

How to achieve fault tolerance when

- Workflow systems are complex in both
 - Number of processes involved in computing one task
 - Number of tasks in one workflow

so failures are likely

- All components need to maintain state
so plain restarting of components is not enough

How to achieve fault tolerance when

- Workflow systems are complex in both
 - Number of processes involved in computing one task
 - Number of tasks in one workflow

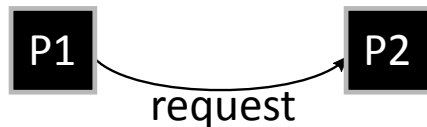
so failures are likely

- All components need to maintain state
so plain restarting of components is not enough
- Restarting of workflow helps only if workflows are small and system has few components

Generic process behaviour

Golden path:

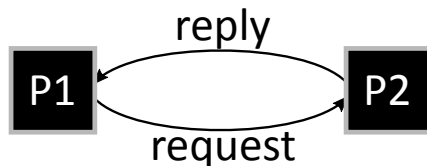
- P_1 sends request to P_2



Generic process behaviour

Golden path:

- P_1 sends request to P_2
- Asynchronously P_1 receives reply



Generic process behaviour

P_2 may fail:

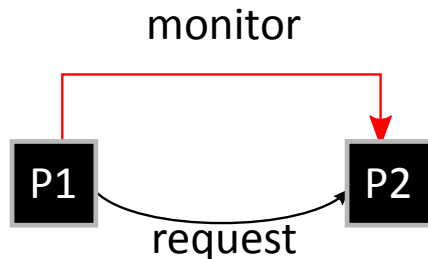
- P_1 sends request to P_2
- P_2 fails



Generic process behaviour

P_2 may fail:

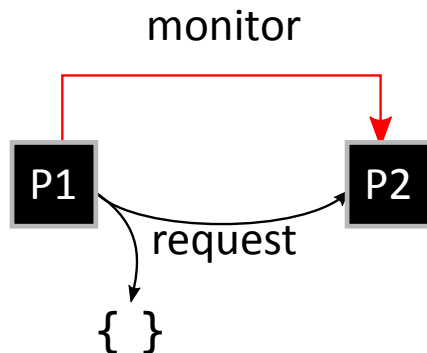
- P_1 sends request to P_2
- P_1 creates monitor on P_2



Generic process behaviour

P_2 may fail:

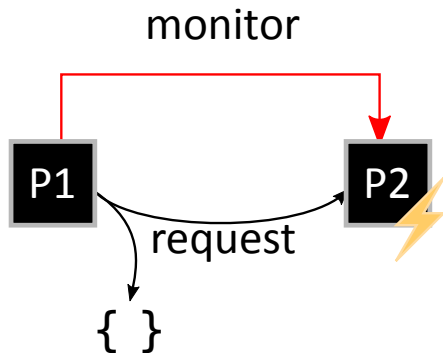
- P_1 sends request to P_2
- P_1 creates monitor on P_2
- P_1 memorizes request



Generic process behaviour

P_2 may fail:

- P_1 sends request to P_2
- P_1 creates monitor on P_2
- P_1 memorizes request
- P_2 fails



Generic process behaviour

P_2 may fail:

- P_1 sends request to P_2
- P_1 creates monitor on P_2
- P_1 memorizes request
- P_2 fails
- P_2 supervisor restarts P_2

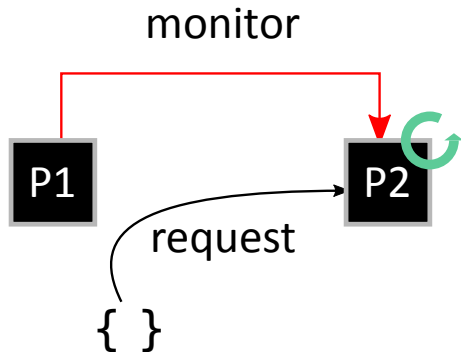


{ }

Generic process behaviour

P_2 may fail:

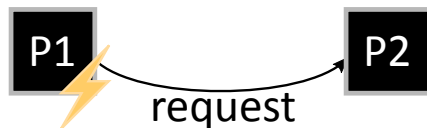
- P_1 sends request to P_2
- P_1 creates monitor on P_2
- P_1 memorizes request
- P_2 fails
- P_2 supervisor restarts P_2
- request is replayed to P_2
- monitor is recreated



Generic process behaviour

P_1 may fail:

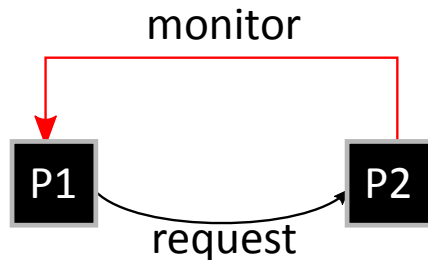
- P_1 sends request to P_2
- P_1 fails



Generic process behaviour

P_1 may fail:

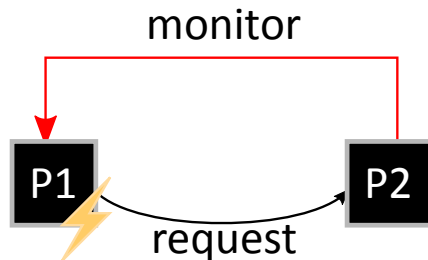
- P_1 sends request to P_2
- P_2 creates monitor on P_1



Generic process behaviour

P_1 may fail:

- P_1 sends request to P_2
- P_2 creates monitor on P_1
- P_1 fails



Generic process behaviour

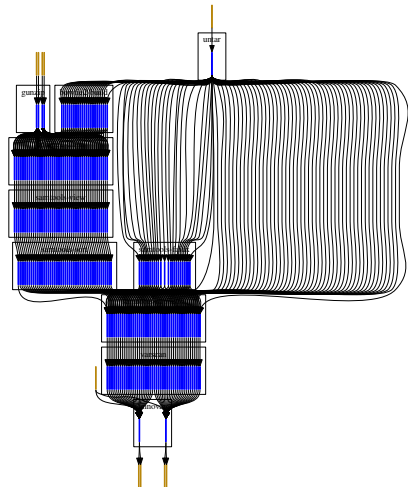
P_1 may fail:

- P_1 sends request to P_2
- P_2 creates monitor on P_1
- P_1 fails
- request is canceled
- supervisor restarts P_1



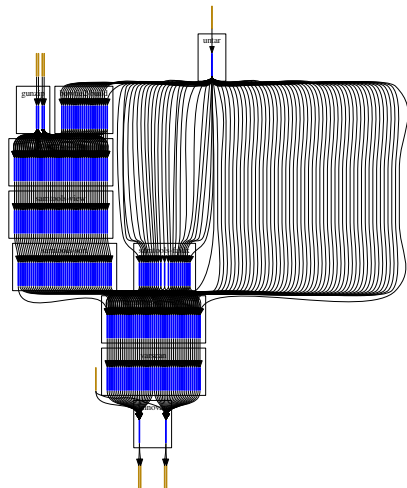
Conclusion

■ Cuneiform:



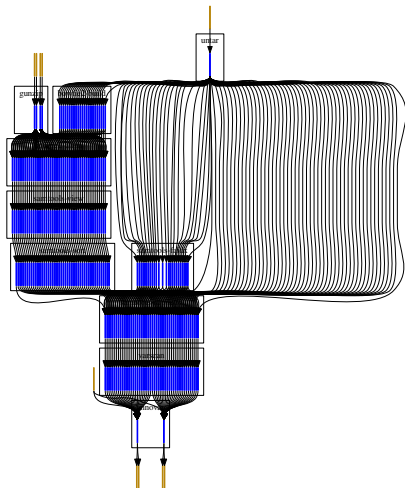
Conclusion

- Cuneiform:
 - Functional



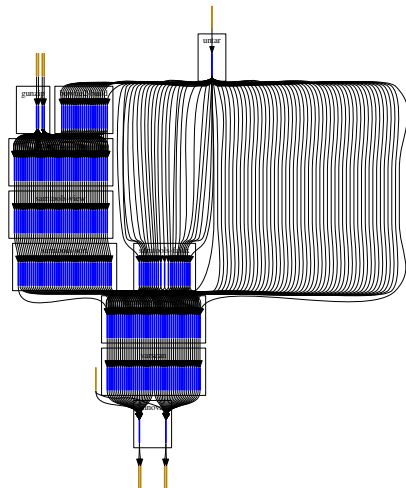
Conclusion

- Cuneiform:
 - Functional
 - Integrate anything



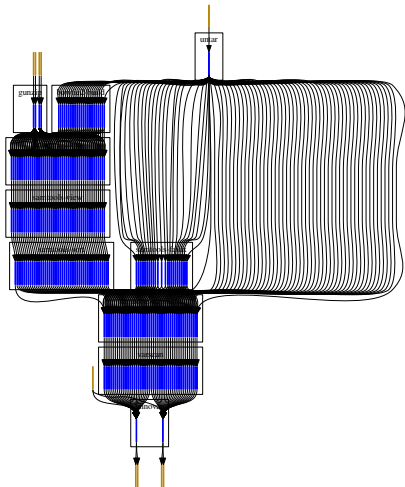
Conclusion

- Cuneiform:
 - Functional
 - Integrate anything
 - Parallelism



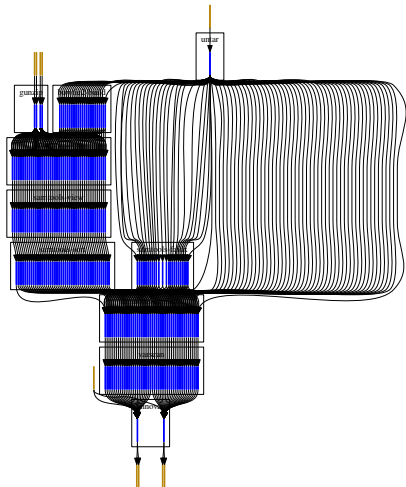
Conclusion

- Cuneiform:
 - Functional
 - Integrate anything
 - Parallelism
- Runs on Hadoop



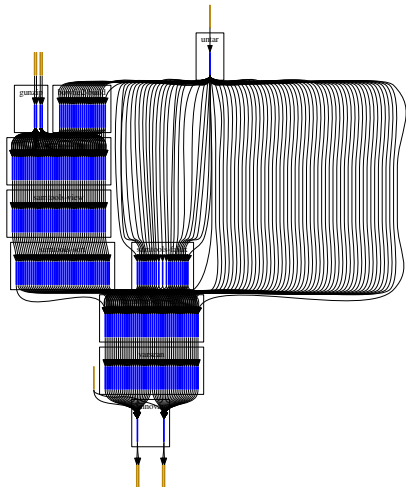
Conclusion

- Cuneiform:
 - Functional
 - Integrate anything
 - Parallelism
- Runs on Hadoop
- Implementation in Erlang:



Conclusion

- Cuneiform:
 - Functional
 - Integrate anything
 - Parallelism
- Runs on Hadoop
- Implementation in Erlang:
 - Concise stateless semantics



Conclusion

- Cuneiform:
 - Functional
 - Integrate anything
 - Parallelism
- Runs on Hadoop
- Implementation in Erlang:
 - Concise stateless semantics
 - Fine-grained fault tolerance

