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

Cuneiform

A Functional Language for Large Scale Scientific Data Analysis

- Black-box operator model

  Pro: Operators can be any piece of software

Cuneiform

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

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

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

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

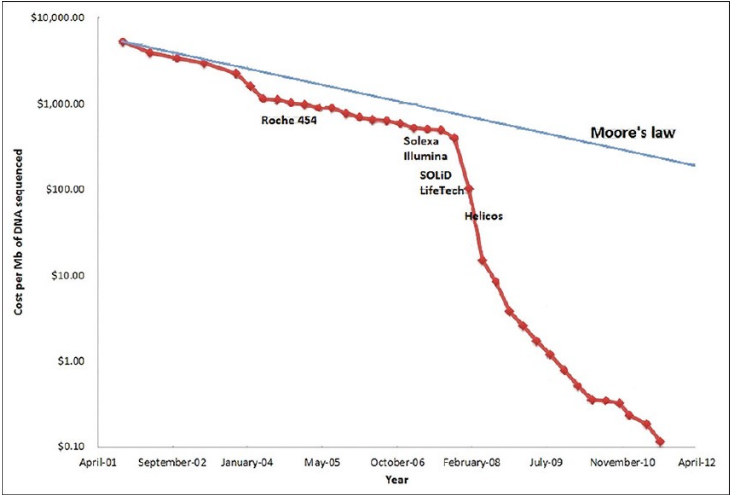
  Scalability with large data sets

Motivation

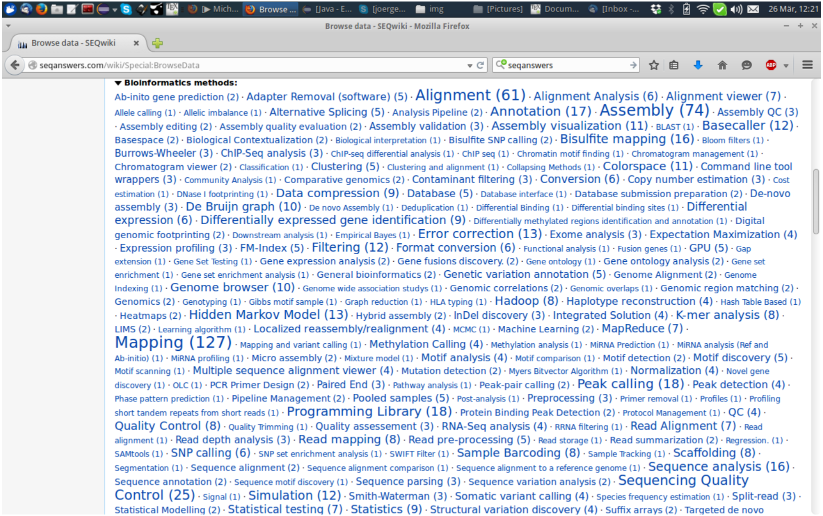
> "New hardware is increasingly parallel, so new programming languages must support concurrency or they will die."
>
> — Joe Armstrong

“New Hardware”

DNA Sequencing is becoming cheap

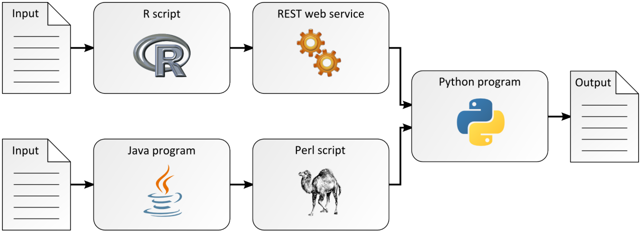


Decentralized software development



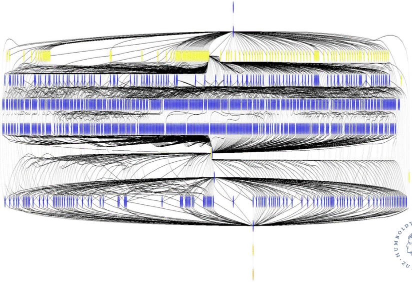
Scientific Workflow Systems

Scientific Workflow Systems

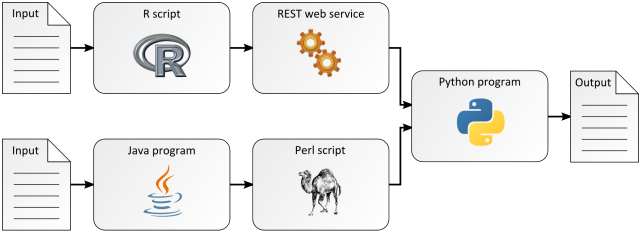


Workflows as DAGs

- Scientific Workflows are DAGs

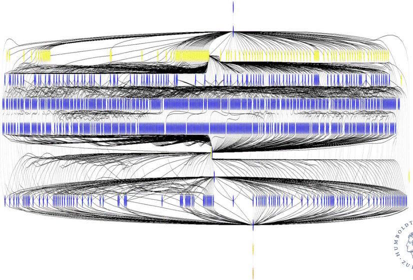


Scientific Workflow Systems

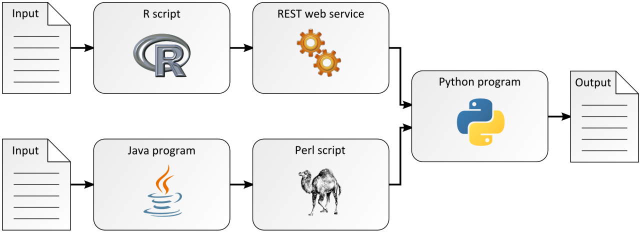


Workflows as DAGs

- Scientific Workflows are DAGs
- Nodes are tasks

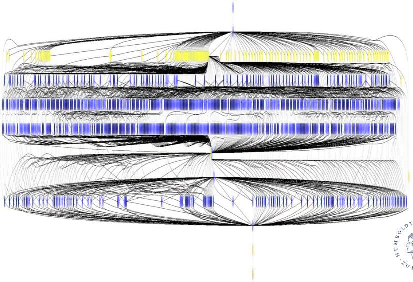


Scientific Workflow Systems

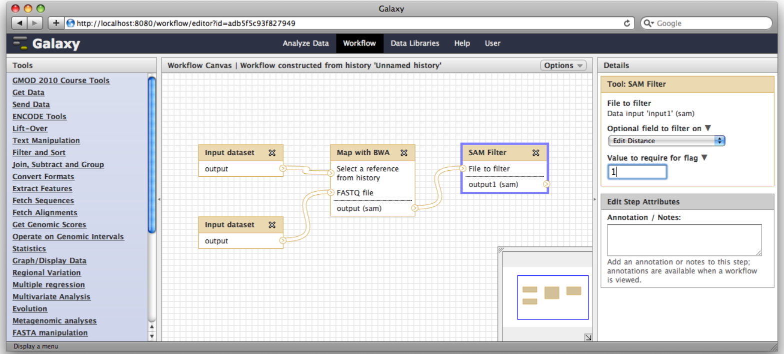


Workflows as DAGs

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



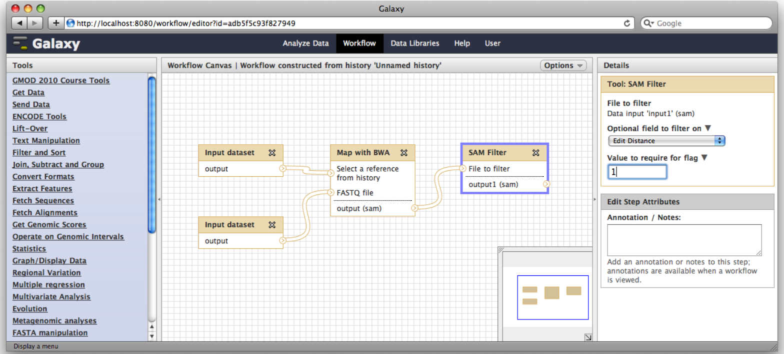
Example: Galaxy Workflow System



Focus on

- Usability

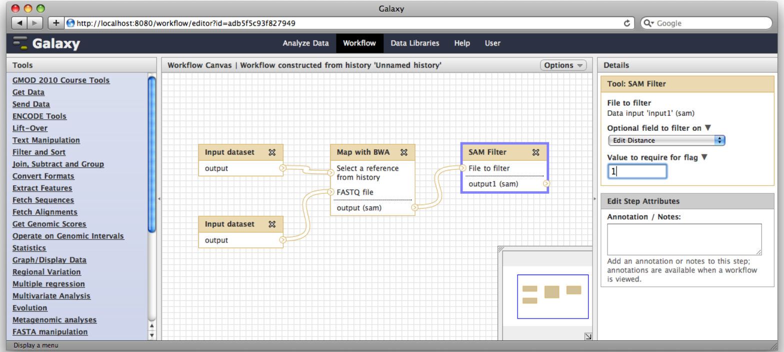
Example: Galaxy Workflow System



Focus on

- Usability
- Integration of tools/libraries

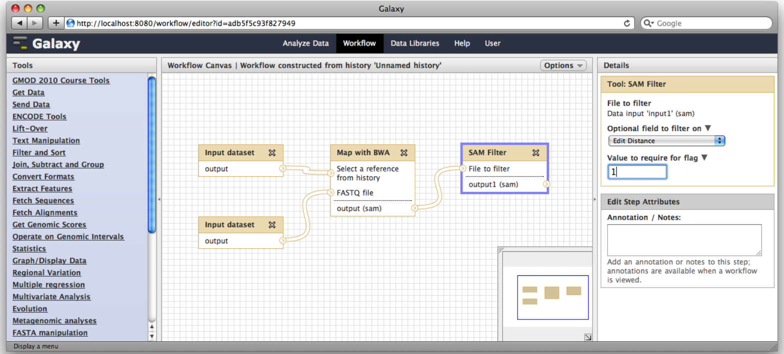
Example: Galaxy Workflow System



Focus on

- Usability
- Integration of tools/libraries
- Systematic documentation

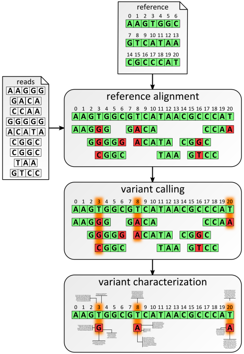
Example: Galaxy Workflow System



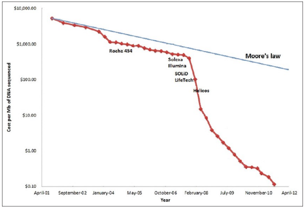
Focus on

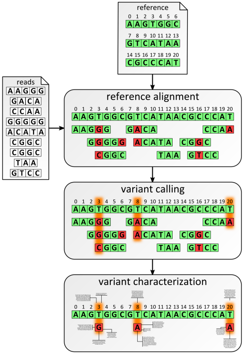
- Usability
- Integration of tools/libraries
- Systematic documentation
- Reproducibility

The Next Generation Sequencing use case

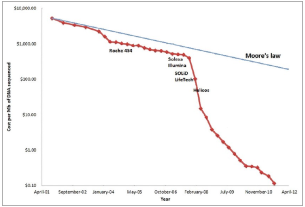


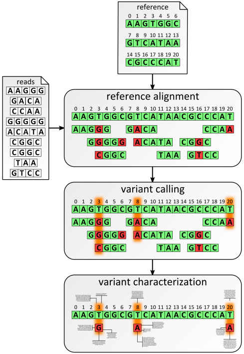
The Next Generation Sequencing use case



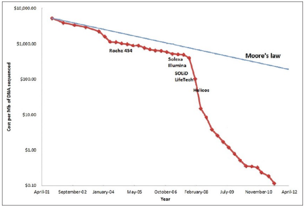


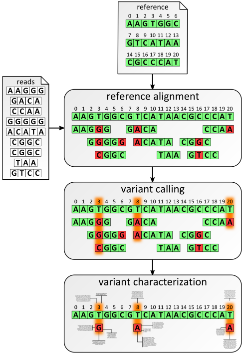
The Next Generation Sequencing use case

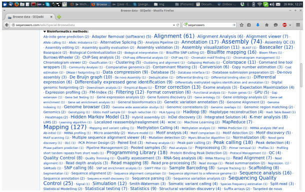




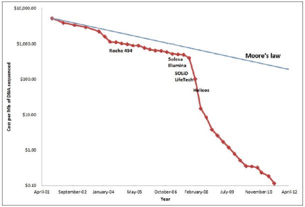
The Next Generation Sequencing use case

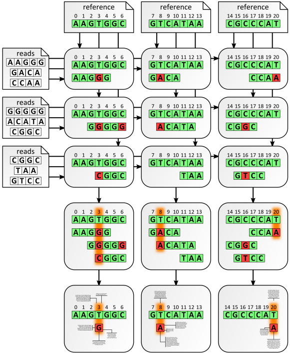


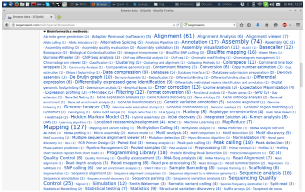




The Next Generation Sequencing use case







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

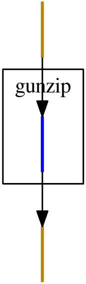
  So we can write arbitrary programs using lists and operations on lists

Cuneiform example

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

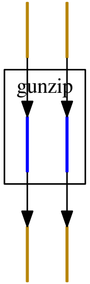
Cuneiform example

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

gunzip(
  gz: 'myarchive1.gz'
);
```



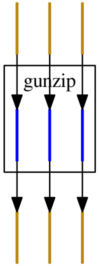
Cuneiform example

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

gunzip(
  gz: 'myarchive1.gz' 'myarchive2.gz'
);
```



Cuneiform example

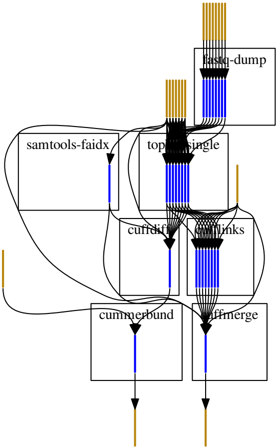
```
deftask gunzip( out( File ) : gz( File ) )in bash *{
  gzip -c -d $gz > $out
}*

gunzip(
  gz: 'myarchive1.gz' 'myarchive2.gz' 'myarchive3.gz'
);
```



Workflow Implementations Available

Available example workflows:

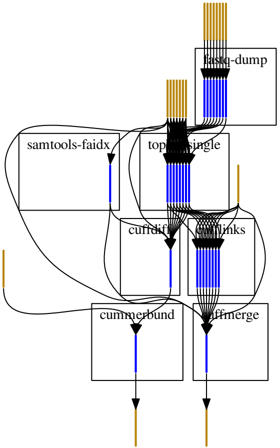
- Variant calling
  `https://www.github.com/joergen7/variant-call`



Workflow Implementations Available

Available example workflows:

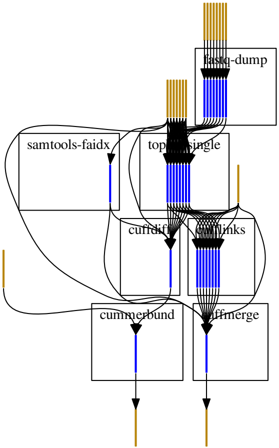
- Variant calling
  `https://www.github.com/joergen7/variant-call`
- Methylation
  `https://www.github.com/joergen7/methylation`



Workflow Implementations Available

Available example workflows:

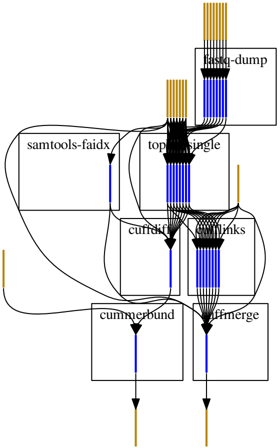
- Variant calling https://www.github.com/joergen7/variant-call
- Methylation https://www.github.com/joergen7/methylation
- RNA-Seq https://www.github.com/joergen7/rna-seq



Workflow Implementations Available

Available example workflows:

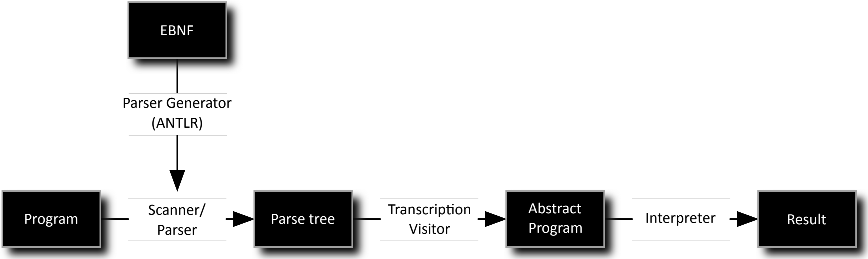
- Variant calling `https://www.github.com/joergen7/variant-call`
- Methylation `https://www.github.com/joergen7/methylation`
- RNA-Seq `https://www.github.com/joergen7/rna-seq`
- etc (ChIP-Seq, miRNA detection, consensus prediction, ...)



Cuneiform Operational Semantics

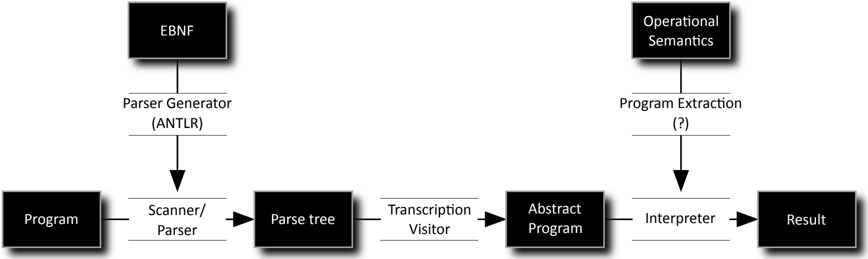
Program Interpretation in Cuneiform

Current design in Java implementation:



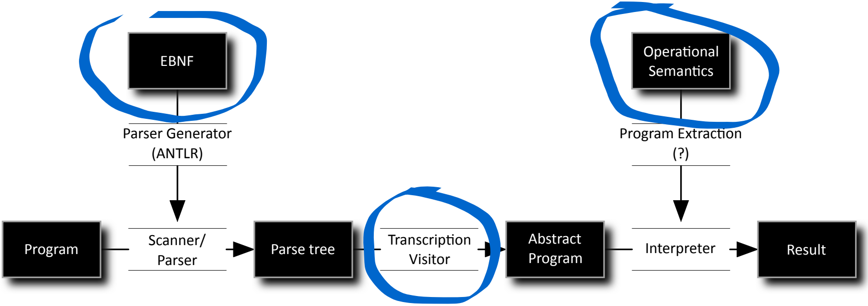
Program Interpretation in Cuneiform

Designated design but never implemented:



Program Interpretation in Cuneiform

Designated design in Erlang:



The eval Function

$$\textbf{fun}\ \mathit{eval}(\mathit{Expr}, \rho, \mathit{GetFuture}, \mathit{Global}, \mathit{Fin}) \rightarrow \mathit{Result}\ \textbf{when}$$

$$\begin{array}{lcl}
\mathit{Expr} & :: & \mathit{expr} \\
\rho & :: & \mathit{string} => \mathit{expr} \\
\mathit{GetFuture} & :: & \mathit{fun} \\
\mathit{Global} & :: & \mathit{string} => \mathit{lam} \\
\mathit{Fin} & :: & \mathit{id} => \mathit{expr} \\
\mathit{Result} & :: & \mathit{expr}
\end{array}$$

The eval Function

$$\textbf{fun } eval(Expr, \rho, GetFuture, Global, Fin) \rightarrow Result \textbf{ when}$$

$$\begin{array}{lcl} Expr & :: & expr \\ \rho & :: & string => expr \\ GetFuture & :: & fun \\ Global & :: & string => lam \\ Fin & :: & id => expr \\ Result & :: & expr \end{array}$$

Expr The expression to be evaluated

The eval Function

fun $\mathit{eval}(\mathit{Expr}, \rho, \mathit{GetFuture}, \mathit{Global}, \mathit{Fin}) \rightarrow \mathit{Result}$ **when**

$$
\begin{array}{lcl}
\mathit{Expr} & :: & \mathit{expr} \\
\rho & :: & \mathit{string} => \mathit{expr} \\
\mathit{GetFuture} & :: & \mathit{fun} \\
\mathit{Global} & :: & \mathit{string} => \mathit{lam} \\
\mathit{Fin} & :: & \mathit{id} => \mathit{expr} \\
\mathit{Result} & :: & \mathit{expr}
\end{array}
$$

Expr The expression to be evaluated

ρ Current scope

The eval Function

$$\textbf{fun}\ \mathit{eval}(\mathit{Expr}, \rho, \mathit{GetFuture}, \mathit{Global}, \mathit{Fin}) \rightarrow \mathit{Result}\ \textbf{when}$$

$$\begin{array}{lcl} \mathit{Expr} & :: & \mathit{expr} \\ \rho & :: & \mathit{string} => \mathit{expr} \\ \mathit{GetFuture} & :: & \mathit{fun} \\ \mathit{Global} & :: & \mathit{string} => \mathit{lam} \\ \mathit{Fin} & :: & \mathit{id} => \mathit{expr} \\ \mathit{Result} & :: & \mathit{expr} \end{array}$$

Expr The expression to be evaluated

ρ Current scope

GetFuture A function returning a future for a foreign task application

The eval Function

$$\textbf{fun } \mathit{eval}(\mathit{Expr}, \rho, \mathit{GetFuture}, \mathit{Global}, \mathit{Fin}) \rightarrow \mathit{Result} \textbf{ when}$$

$$\begin{array}{lcl}
\mathit{Expr} & :: & \mathit{expr} \\
\rho & :: & \mathit{string} => \mathit{expr} \\
\mathit{GetFuture} & :: & \mathit{fun} \\
\mathit{Global} & :: & \mathit{string} => \mathit{lam} \\
\mathit{Fin} & :: & \mathit{id} => \mathit{expr} \\
\mathit{Result} & :: & \mathit{expr}
\end{array}$$

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

$$\textbf{fun}\ \mathit{eval}(\mathit{Expr}, \rho, \mathit{GetFuture}, \mathit{Global}, \mathit{Fin}) \rightarrow \mathit{Result}\ \textbf{when}$$

$$\begin{array}{lcl} \mathit{Expr} & :: & \mathit{expr} \\ \rho & :: & \mathit{string} => \mathit{expr} \\ \mathit{GetFuture} & :: & \mathit{fun} \\ \mathit{Global} & :: & \mathit{string} => \mathit{lam} \\ \mathit{Fin} & :: & \mathit{id} => \mathit{expr} \\ \mathit{Result} & :: & \mathit{expr} \end{array}$$

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 $\mathit{eval}(\mathit{Expr}, \rho, \mathit{GetFuture}, \mathit{Global}, \mathit{Fin}) \rightarrow \mathit{Result}$ **when**

$$\begin{array}{lcl} \mathit{Expr} & :: & \mathit{expr} \\ \rho & :: & \mathit{string} => \mathit{expr} \\ \mathit{GetFuture} & :: & \mathit{fun} \\ \mathit{Global} & :: & \mathit{string} => \mathit{lam} \\ \mathit{Fin} & :: & \mathit{id} => \mathit{expr} \\ \mathit{Result} & :: & \mathit{expr} \end{array}$$

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)

Computational Semantics

$$
\begin{array}{l}
\mathit{eval}(\mathit{Expr}, \rho, \mathit{GetFuture}, \mathit{Global}, \mathit{Fin}) \rightarrow \\
\quad \mathit{Next} = \mathit{step}(\mathit{Expr}, \rho, \mathit{GetFuture}, \mathit{Global}, \mathit{Fin}) \\
\quad \textbf{case } \mathit{Next} \textbf{ of} \\
\qquad \begin{array}{lcl}
\mathit{Expr} & \rightarrow & \mathit{Expr} \\
\_ & \rightarrow & \mathit{eval}(\mathit{Next}, \rho, \mathit{GetFuture}, \mathit{Global}, \mathit{Fin})
\end{array} \\
\quad \textbf{end}
\end{array}
$$

Computational Semantics

$$
\begin{array}{l}
\mathit{eval}(\mathit{Expr}, \rho, \mathit{GetFuture}, \mathit{Global}, \mathit{Fin}) \rightarrow \\
\quad \mathit{Next} = \mathit{step}(\mathit{Expr}, \rho, \mathit{GetFuture}, \mathit{Global}, \mathit{Fin}) \\
\quad \textbf{case } \mathit{Next} \textbf{ of} \\
\quad\quad \begin{array}{lcl} \mathit{Expr} & \rightarrow & \mathit{Expr} \\ \_ & \rightarrow & \mathit{eval}(\mathit{Next}, \rho, \mathit{GetFuture}, \mathit{Global}, \mathit{Fin}) \end{array} \\
\quad \textbf{end}
\end{array}
$$

- single step is computed

Computational Semantics

$$
\begin{array}{l}
eval(Expr, \rho, GetFuture, Global, Fin) \rightarrow \\
\quad Next = step(Expr, \rho, GetFuture, Global, Fin) \\
\quad \textbf{case } Next \textbf{ of} \\
\qquad \begin{array}{lcl} Expr & \rightarrow & Expr \\ \_ & \rightarrow & eval(Next, \rho, GetFuture, Global, Fin) \end{array} \\
\quad \textbf{end}
\end{array}
$$

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

Computational Semantics

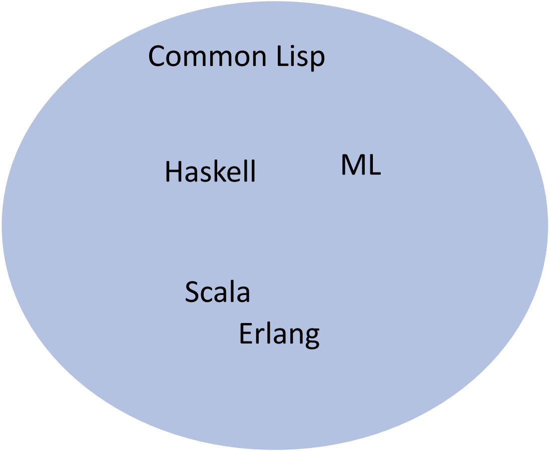
$$
\begin{array}{l}
\textbf{fun } \mathit{eval}(\mathit{Expr}, \rho, \mathit{GetFuture}, \mathit{Global}, \mathit{Fin}) \rightarrow \\
\quad \mathit{Next} = \mathit{step}(\mathit{Expr}, \rho, \mathit{GetFuture}, \mathit{Global}, \mathit{Fin}) \\
\quad \textbf{case } \mathit{Next} \textbf{ of} \\
\quad\quad \begin{array}{lcl} \mathit{Expr} & \rightarrow & \mathit{Expr} \\ \_ & \rightarrow & \mathit{eval}(\mathit{Next}, \rho, \mathit{GetFuture}, \mathit{Global}, \mathit{Fin}) \end{array} \\
\quad \textbf{end}
\end{array}
$$

- 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

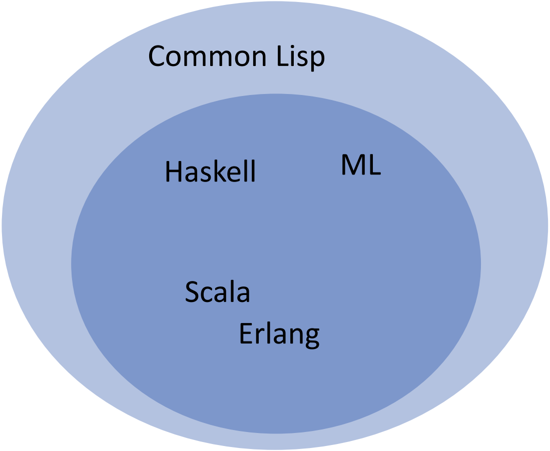
- Functional Language



Formalisms Suitable for Operational Semantics

Choosing a language

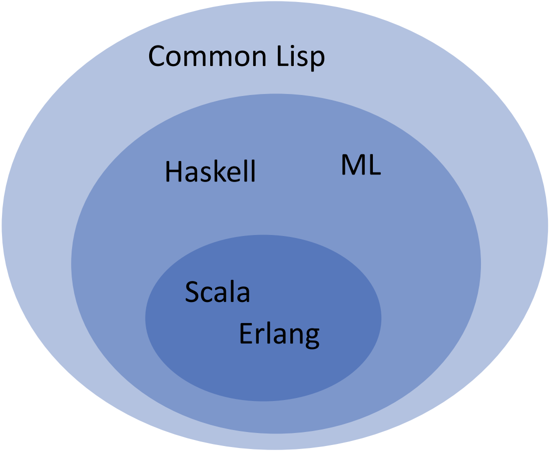
- Functional Language
- With Pattern Matching



Formalisms Suitable for Operational Semantics

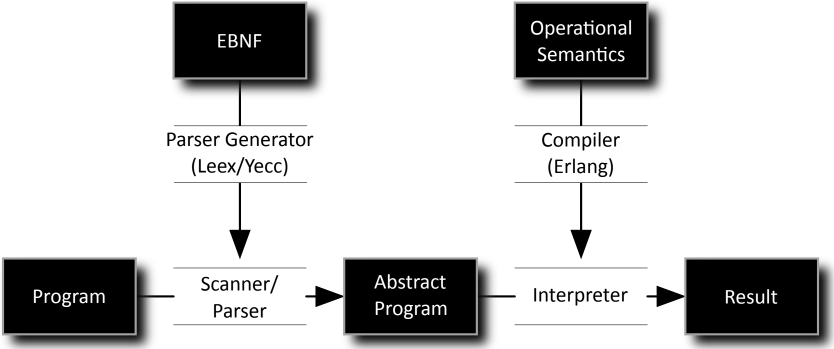
Choosing a language

- Functional Language
- With Pattern Matching
- Concurrency Orientation



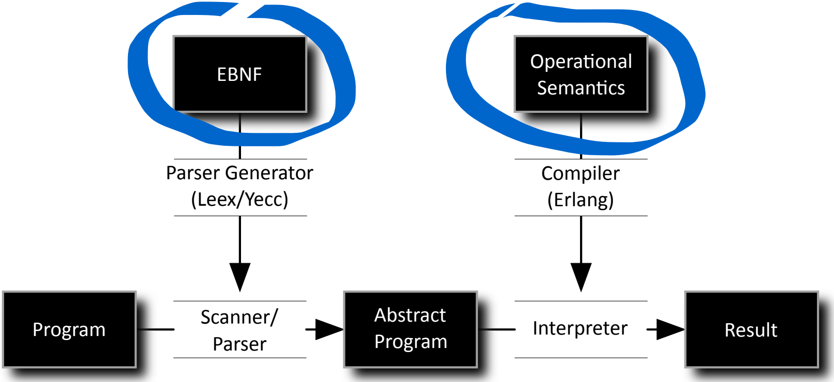
Program Interpretation in Cuneiform

New design:



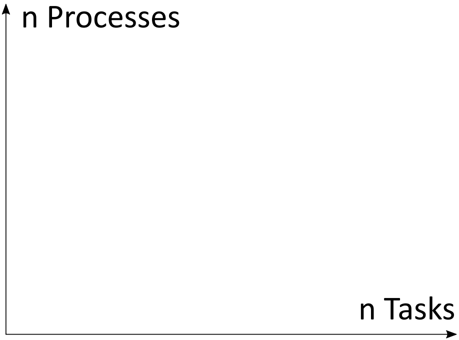
Program Interpretation in Cuneiform

New design:



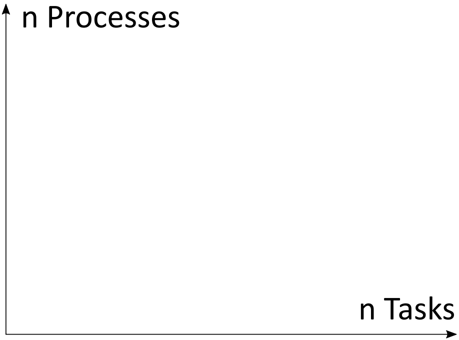
Fault Tolerance

Two types of complexity



Distributed applications can be complex in two independent ways:

Two types of complexity

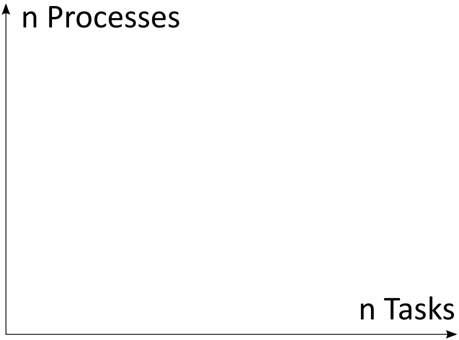


Distributed applications can be complex in two independent ways:

- 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:

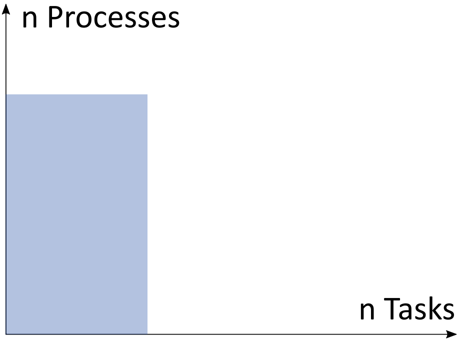
- in the number of processes involved to compute one task

  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:

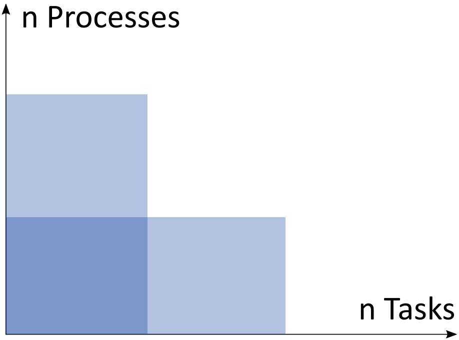
- in the number of processes involved to compute one task

  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

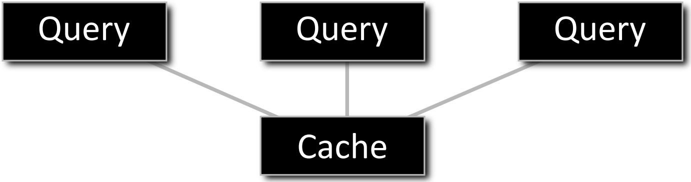
  the more system components the more likely one component fails

- in the number of tasks contributing to a workflow

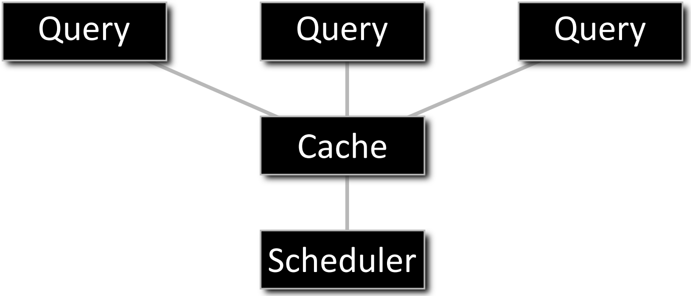
  the more tasks the more likely one task fails

Distributed Application: Workflow System

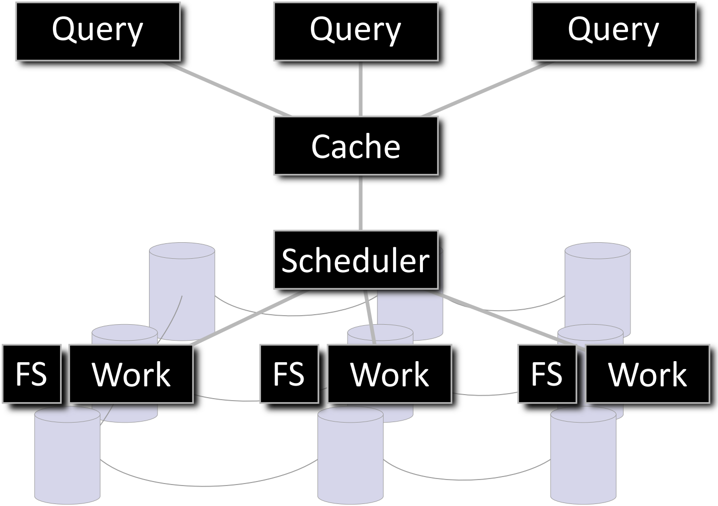
Distributed Application: Workflow System



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

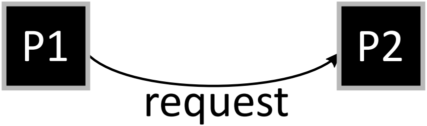
  so failures are likely
- All components need to maintain state

  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:

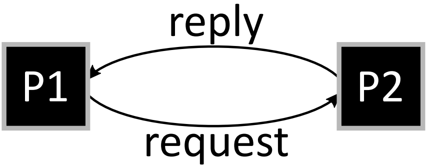
- P_1 sends request to P_2



Generic process behaviour

Golden path:

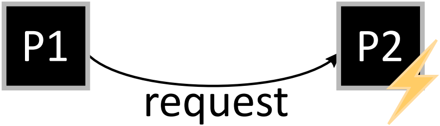
- P_1 sends request to P_2
- Asynchronously P_1 receives reply



Generic process behaviour

P_2 may fail:

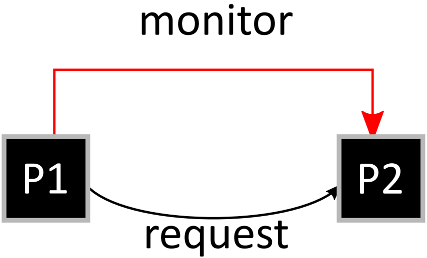
- P_1 sends request to P_2
- P_2 fails



Generic process behaviour

P_2 may fail:

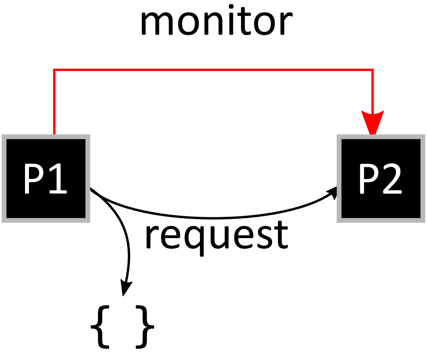
- P_1 sends request to P_2
- P_1 creates monitor on P_2



Generic process behaviour

P_2 may fail:

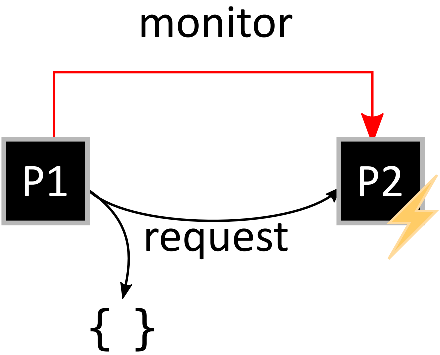
- P_1 sends request to P_2
- P_1 creates monitor on P_2
- P_1 memorizes request



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

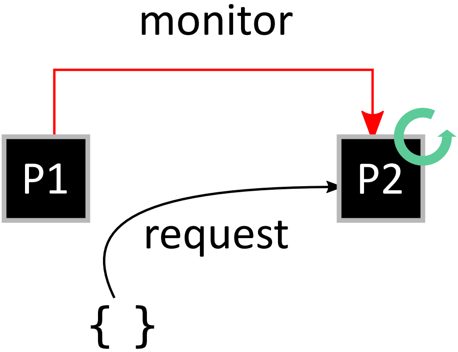
P_2 may fail:

- 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:

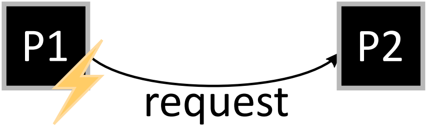
- P_1 sends request to P_2
- P_1 creates monitor on P_2
- P_1 memorizes request
- P_2 fails
- P_2 supervisor restarts P_2
- request is replayed to P_2
- monitor is recreated



Generic process behaviour

P_1 may fail:

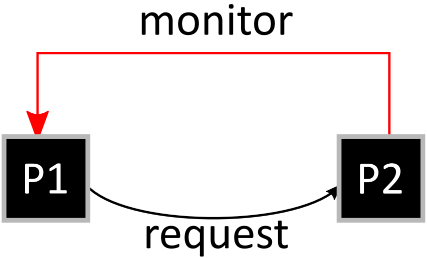
- P_1 sends request to P_2
- P_1 fails



Generic process behaviour

P_1 may fail:

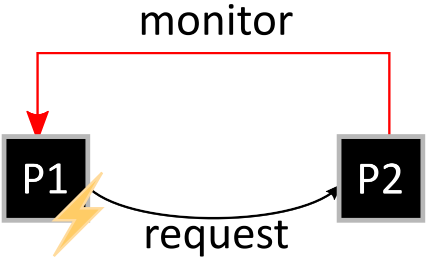
- P_1 sends request to P_2
- P_2 creates monitor on P_1



Generic process behaviour

P_1 may fail:

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

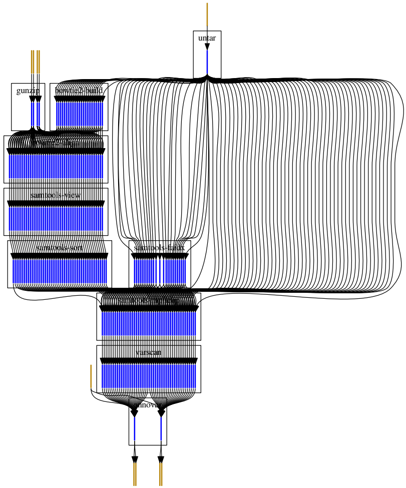


Generic process behaviour

P_1 may fail:

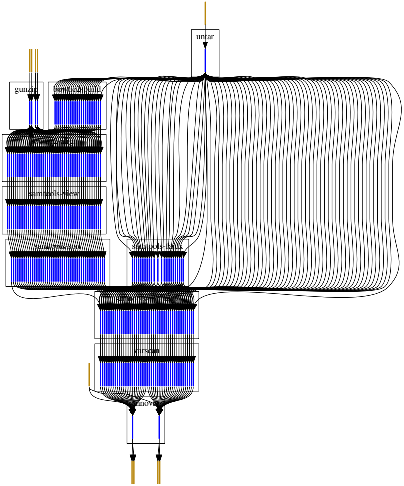
- P_1 sends request to P_2
- P_2 creates monitor on P_1
- P_1 fails
- request is canceled
- supervisor restarts P_1

- Cuneiform:



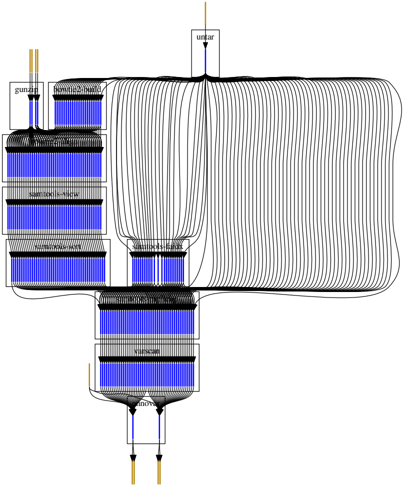
Conclusion

- Cuneiform:
  - Functional



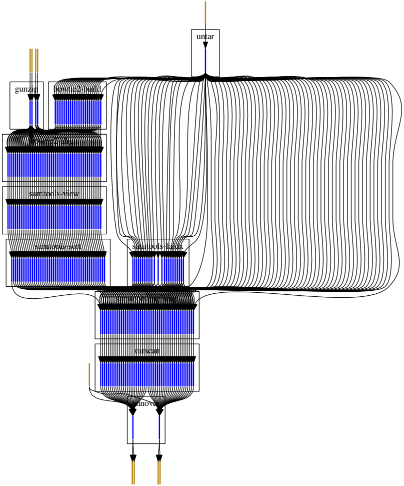
Conclusion

- Cuneiform:
  - Functional
  - Integrate anything



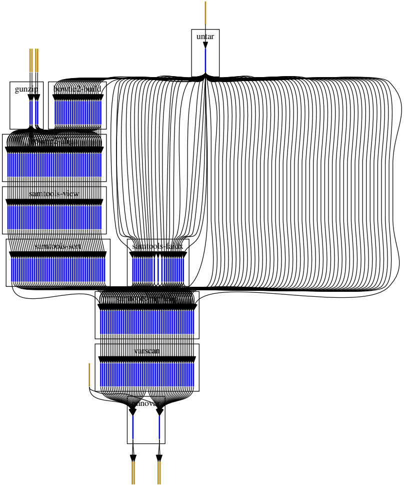
Conclusion

- Cuneiform:
  - Functional
  - Integrate anything
  - Parallelism



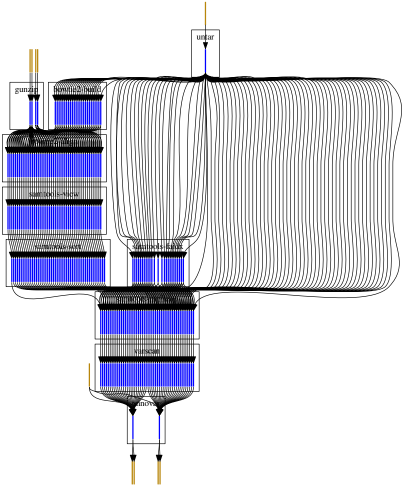
Conclusion

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



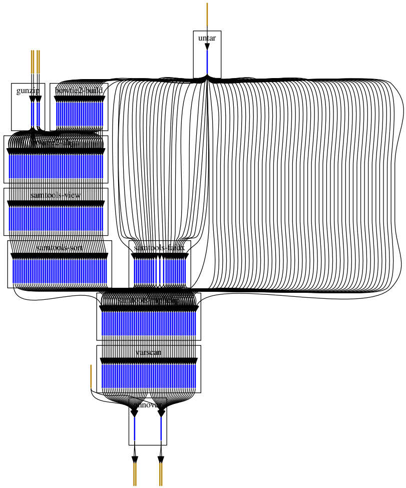
Conclusion

- 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