Martlet

A Scientific Work-Flow Language for Abstracted Parallisation

Daniel Goodman

Daniel.Goodman@ComLab.ox.ac.uk

climate*prediction*.net

Contents

- About climate*prediction*.net
- Example problem
- Martlet
- Extending use through JIT Compilers
- Conclusion

climate*prediction*.net

Introducing climate*prediction*.net

Use distributed computing to:

climate*prediction*.net

Introducing climate*prediction*.net

Use distributed computing to:

- improve the parametrisation of climate models both future and present

climate*prediction*.net

Introducing climate*prediction*.net

Use distributed computing to:

- improve the parametrisation of climate models both future and present
- generate a huge set of analysable perturbed climate model runs as a resource for scientists.

climate*prediction*.net

Introducing climate*prediction*.net

Use distributed computing to:

- improve the parametrisation of climate models both future and present
- generate a huge set of analysable perturbed climate model runs as a resource for scientists.

The framework to achive this is currently built on top of the Berkeley Open Infrastructure for Network Computing (BOINC)

climate*prediction*.net

Achived so far

climate*prediction*.net

Achived so far

- Over 62,250 machines running models

climateprediction.net

Achived so far

- Over 62,250 machines running models
- Over 180,000 full climate models run

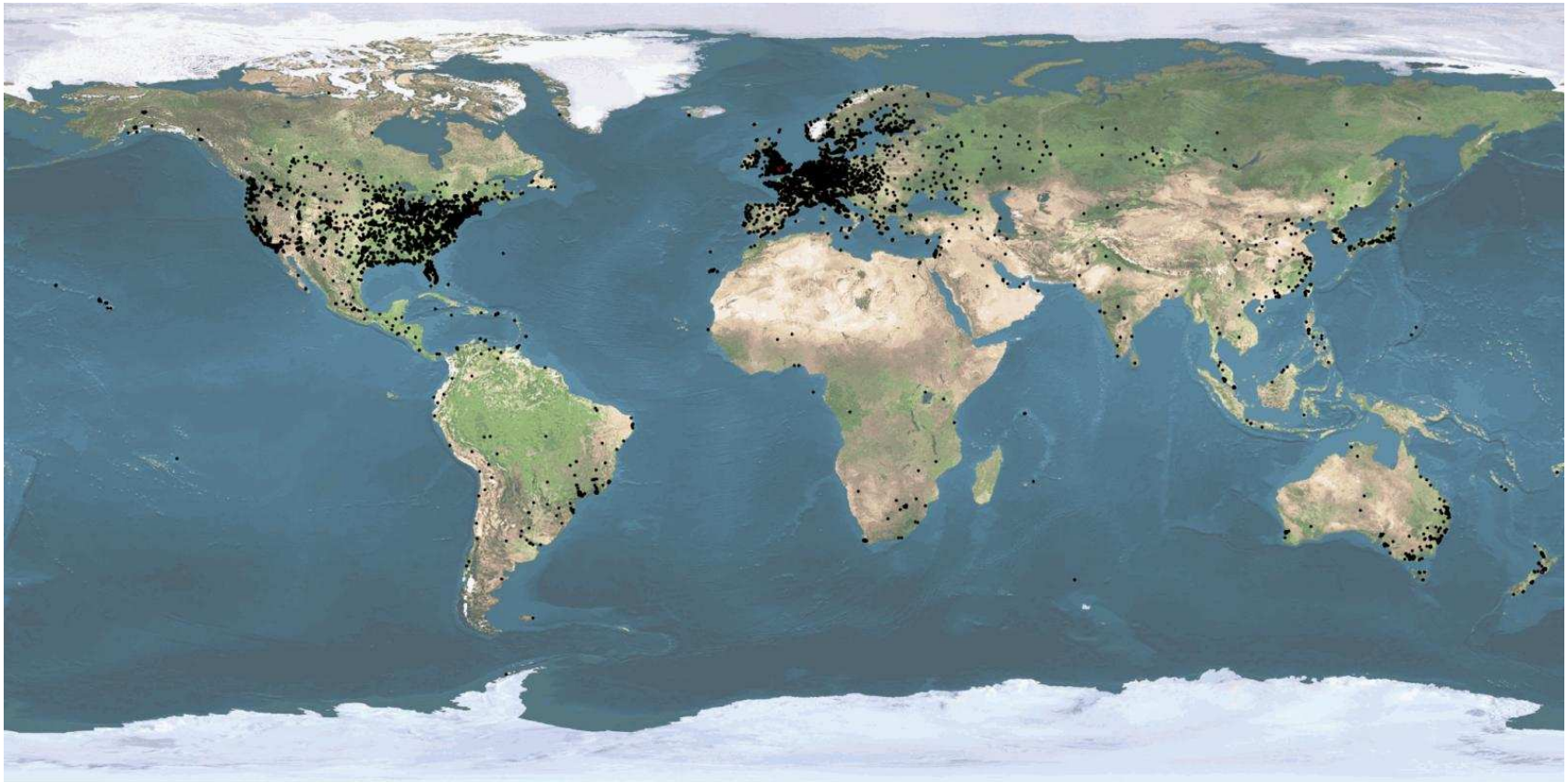
climateprediction.net

Achived so far

- Over 62,250 machines running models
- Over 180,000 full climate models run
- Totalling over 17,100,000 years modelled

climateprediction.net

Map of Users



Users on all seven continents

climateprediction.net

Returned Runs

Returned runs create a data set that is:

- too big to return to a single location
- trivially partitioned

climate*prediction*.net

Returned Runs

Returned runs create a data set that is:

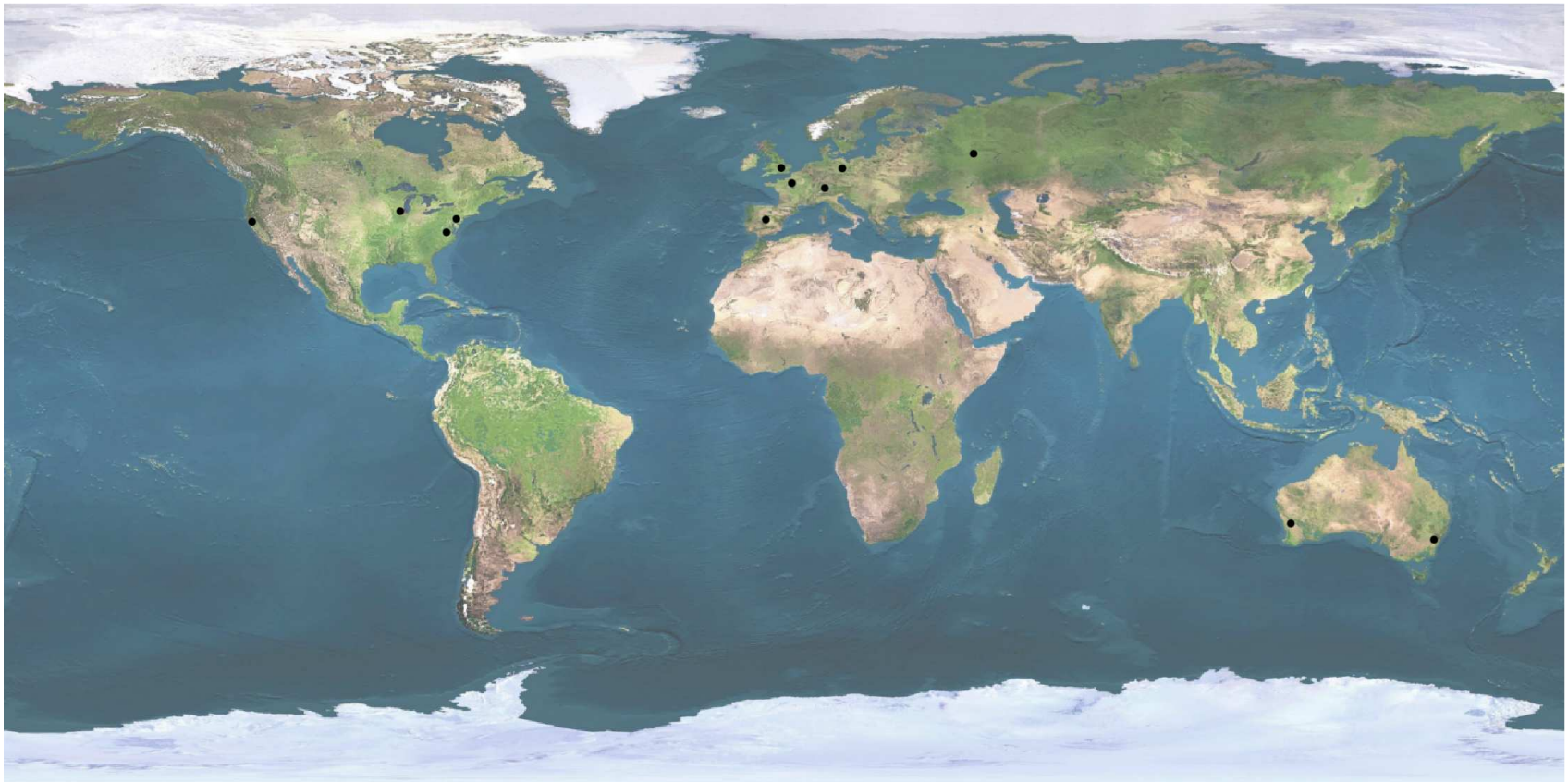
- to big to return to a single location
- trivially partitioned

Because of this we:

- ensure that individual runs are not split
- use many donated computers to store returned runs

climateprediction.net

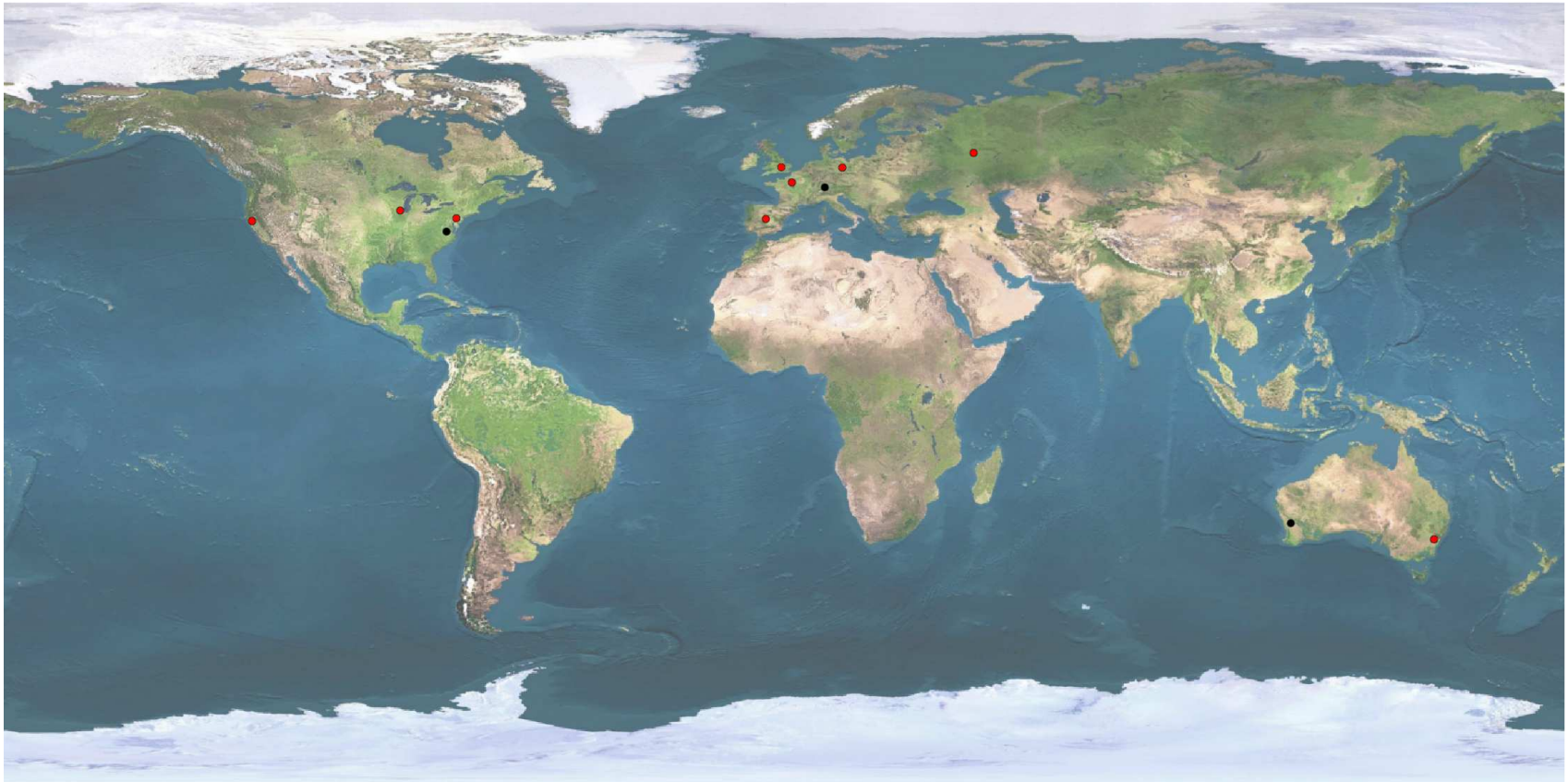
Example problem



Locations of upload servers

climateprediction.net

Example problem



Upload severs holding revent data to query X

climateprediction.net

Example problem

$$y_0 = \sum_{i=0}^{n_1-1} x_i$$
$$z_0 = n_1$$
$$y_1 = \sum_{i=n_1}^{n_2-1} x_i$$
$$z_1 = n_2 - n_1$$
$$\vdots$$
$$y_{a-1} = \sum_{i=n_{a-1}}^{n_a-1} x_i$$
$$z_{a-1} = n_a - n_{a-1}$$

$$\overline{x} = \frac{\sum_{i=0}^{a-1} y_i}{\sum_{i=0}^{a-1} z_i}$$

Using existing workflow languages would require the user to know the value of a at the time they write the workflow, and to update the workflow when the value of a changes.

climateprediction.net

Returned Runs

As analysis is typically data intensive, this means most direct analysis of the data needs to occur locally to the upload servers. This requires that an interface is provided to:

climate*prediction*.net

Returned Runs

As analysis is typically data intensive, this means most direct analysis of the data needs to occur locally to the upload servers. This requires that an interface is provided to:

- Prevent users running their own binaries

climateprediction.net

Returned Runs

As analysis is typically data intensive, this means most direct analysis of the data needs to occur locally to the upload servers. This requires that an interface is provided to:

- Prevent users running their own binaries
- Hide the complexity of the distribution

climateprediction.net

Returned Runs

As analysis is typically data intensive, this means most direct analysis of the data needs to occur locally to the upload servers. This requires that an interface is provided to:

- Prevent users running their own binaries
- Hide the complexity of the distribution
- Hide the internal state from the user

climateprediction.net

Functional Programming

A function for reversing a list

```
reverse :: [a] -> [a]
reverse [] = []
reverse (x:xs) = reverse xs ++ [x]
```

Note the separation between the base case and the inductive case

climateprediction.net

Introducing Martlet

```
define{
   uri1 = baseFunction:system:http://cpdn.net:8080/Martlet;
}

proc(A,B){

   Y = new dismatrix(A);
   Z = new disinteger(A);
   ZTotal = new integer(B);

   map{
      matrixSum:uri1(A,Y);
      matrixCardinality:uri1(A,Z);
   }
```

climateprediction.net

Introducing Martlet

```
   tree((YL,YR)\Y -> B, (ZL,ZR)\Z -> ZTotal){
      matrixSumToVector:uri1(YL,YR,B);
      integerSum:uri1(ZL,ZR,ZTotal);
   }

   matrixDivide:uri1(B,Z_Total,B);
}
```

climateprediction.net

Introducing Martlet

- Two classes of data structure
  - Local data
  - Distributed data

climateprediction.net

Introducing Martlet

- Two classes of data structure
  - Local data
  - Distributed data
- Two types of statement
  - Normal statements
  - Expandable statements

climateprediction.net

Introducing Martlet

- Two classes of data structure
  - Local data
  - Distributed data
- Two types of statement
  - Normal statements
  - Expandable statements
- Both functions and data structures are referenced by URIs

climateprediction.net

Introducing Martlet

- Two classes of data structure
  - Local data
  - Distributed data
- Two types of statement
  - Normal statements
  - Expandable statements
- Both functions and data structures are referenced by URIs
- Functions are first class values

climateprediction.net

Data Structures

```
define{
   uri1 = baseFunction:system:http://cpdn.net:8080/Martlet;
}

proc(A,B){
   Y = new dismatrix(A);
   Z = new disinteger(A);
   ZTotal = new integer(B);

   map{
      matrixSum:uri1(A,Y);
      matrixCardinality:uri1(A,Z);
   }
```

climateprediction.net

Local Data Structures

```
define{
   uri1 = baseFunction:system:http://cpdn.net:8080/Martlet;
}

proc(A,B){
   Y = new dismatrix(A);
   Z = new disinteger(A);
   ZTotal = new integer(B);

   map{
      matrixSum:uri1(A,Y);
      matrixCardinality:uri1(A,Z);
   }
```

climateprediction.net

Local Data Structures

Local data structure

- Exists only in a single location
- Can be acted on directly

climateprediction.net

Distributed Data Structures

```
define{
   uri1 = baseFunction:system:http://cpdn.net:8080/Martlet;
}

proc(A,B){
   Y = new dismatrix(A);
   Z = new disinteger(A);
   ZTotal = new integer(B);

   map{
      matrixSum:uri1(A,Y);
      matrixCardinality:uri1(A,Z);
   }
```

climateprediction.net

Distributed Data Structures

Distributed data structure

- Abstractly, a list of local data structures
- Can be split across many locations
- Can be acted on only in conjunction with an expandable statement

climateprediction.net

Statements

Normal statements

- Sequential composition
- Asynchronous composition
- If-else
- While
- Temporary Variables
- Process calls

climateprediction.net

Statements

Expandable statements

- Map
- Foldr
- Foldl
- Tree

climate*prediction*.net

Map

```
define{
   uri1 = baseFunction:system:http://cpdn.net:8080/Martlet;
}

proc(A,B){

   Y = new dismatrix(A);
   Z = new disinteger(A);
   ZTotal = new integer(B);

   map{
      matrixSum:uri1(A,Y);
      matrixCardinality:uri1(A,Z);
   }
```

climateprediction.net

Map

Haskell definition

```
map f [] = []
map f (x:xs) = (f x):(map f xs)
```

climateprediction.net

Map

Haskell definition

```
map f [] = []
map f (x:xs) = (f x):(map f xs)
```

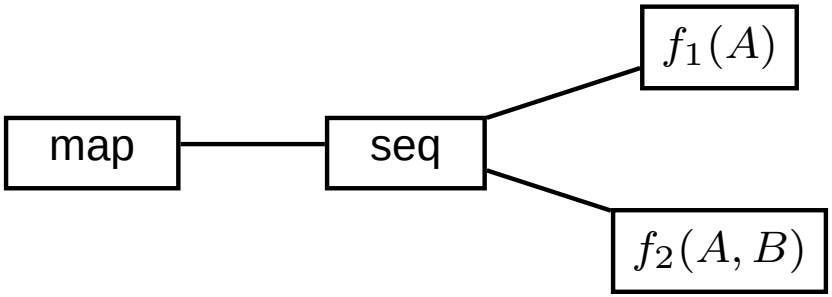
Example Martlet code

```
map{
    f1(A);
    f2(A,B);
}
```

climateprediction.net

Map

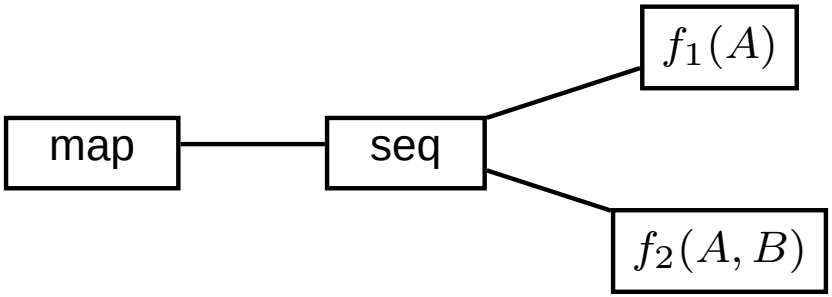
The unexpanded abstract syntax tree



climateprediction.net

Map

The unexpanded abstract syntax tree



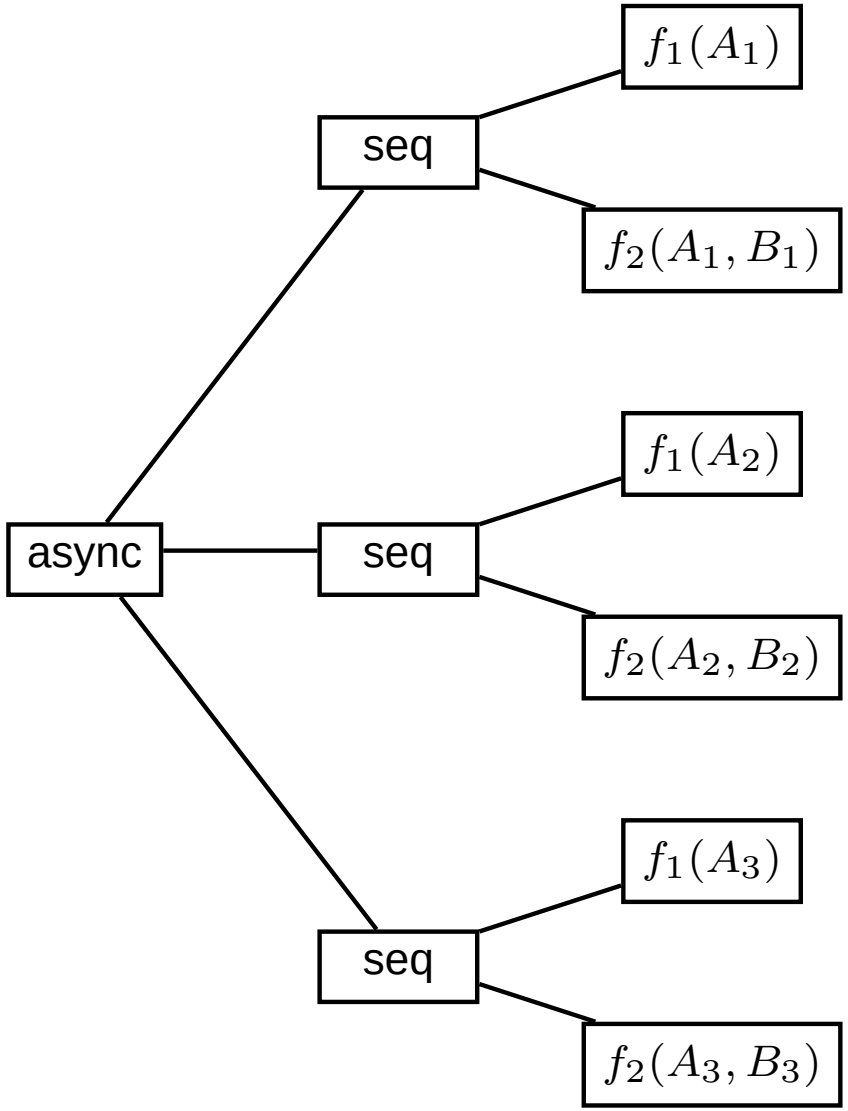
Concrete input data

$$A = [A_1, A_2, A_3]$$

$$B = [B_1, B_2, B_3]$$

climateprediction.net

Map



climateprediction.net

Tree

```
    tree((YL,YR)\Y -> B, (ZL,ZR)\Z -> ZTotal){
        matrixSumToVector:uri1(YL,YR,B);
        integerSum:uri1(ZL,ZR,ZTotal);
    }

    matrixDivide:uri1(B,ZTotal,B);
}
```

climateprediction.net

Tree

Haskell definition

```
tree f [x] = x
tree f (x:y:ys) =
  f (tree f xs') (tree f ys')
     where (xs',ys') = split (x:y:ys)
```

climateprediction.net

Tree

Haskell definition

```
tree f [x] = x
tree f (x:y:ys) =
  f (tree f xs') (tree f ys')
     where (xs',ys') = split (x:y:ys)
```

Example Martlet code

```
tree((XL,XR)\X -> A){
   f(A,XL,XR);
}
```

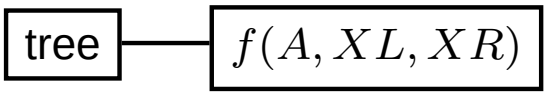
climateprediction.net

Tree

Example Martlet code

```
tree((XL,XR)\X -> A){
   f(A,XL,XR);
}
```

The unexpanded abstract syntax tree



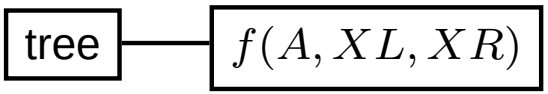
climateprediction.net

Tree

Example Martlet code

```
tree((XL,XR)\X -> A){
   f(A,XL,XR);
}
```

The unexpanded abstract syntax tree



Concrete input data

$$X = [X_1, X_2, X_3, X_4, X_5]$$

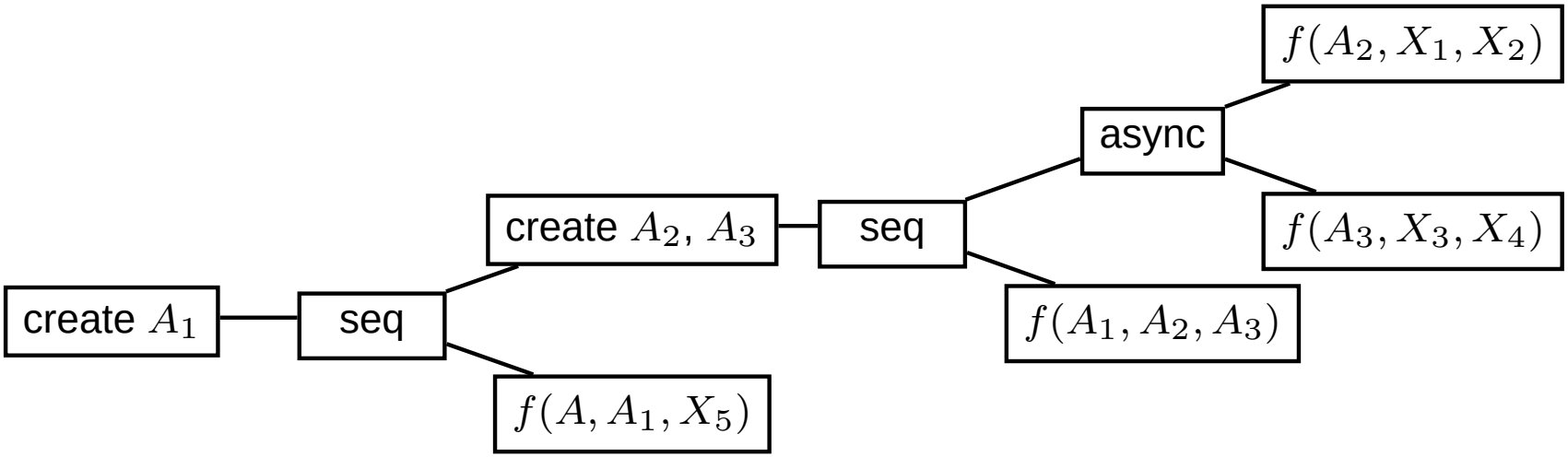
climateprediction.net

Tree

Concrete input data

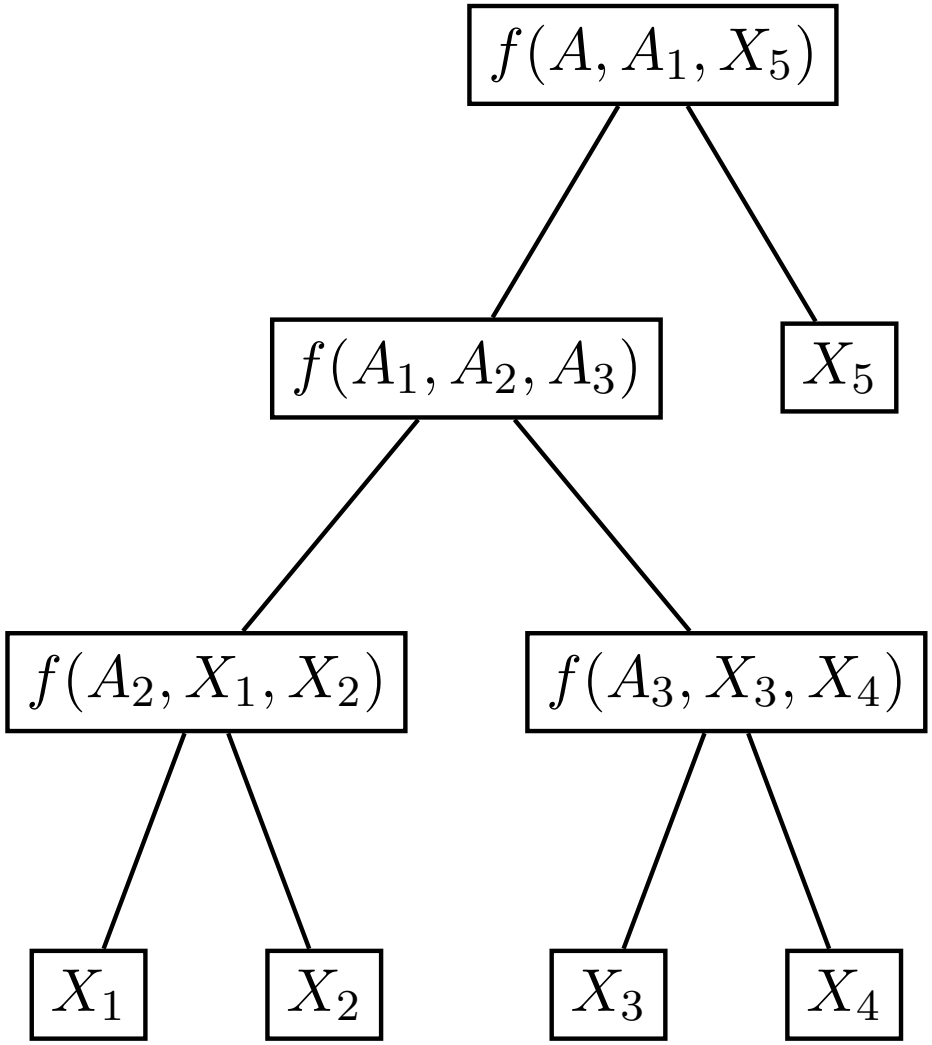
$$X = [X_1, X_2, X_3, X_4, X_5]$$

Expanded concrete syntax tree



climateprediction.net

Tree



climateprediction.net

Example

$$y_0 = \sum_{i=0}^{n_1-1} x_i$$
$$z_0 = n_1$$

$$y_1 = \sum_{i=n_1}^{n_2-1} x_i$$
$$z_1 = n_2 - n_1$$

$$\vdots$$

$$y_{a-1} = \sum_{i=n_{a-1}}^{n_a-1} x_i$$
$$z_{a-1} = n_a - n_{a-1}$$

$$\overline{x} = \frac{\sum_{i=0}^{a-1} y_i}{\sum_{i=0}^{a-1} z_i}$$

climateprediction.net

Example

```
define{
   uri1 = baseFunction:system:http://cpdn.net:8080/Martlet;
}

proc(A,B){

   Y = new dismatrix(A);
   Z = new disinteger(A);
   ZTotal = new integer(B);

   map{
      matrixSum:uri1(A,Y);
      matrixCardinality:uri1(A,Z);
   }
```

climateprediction.net

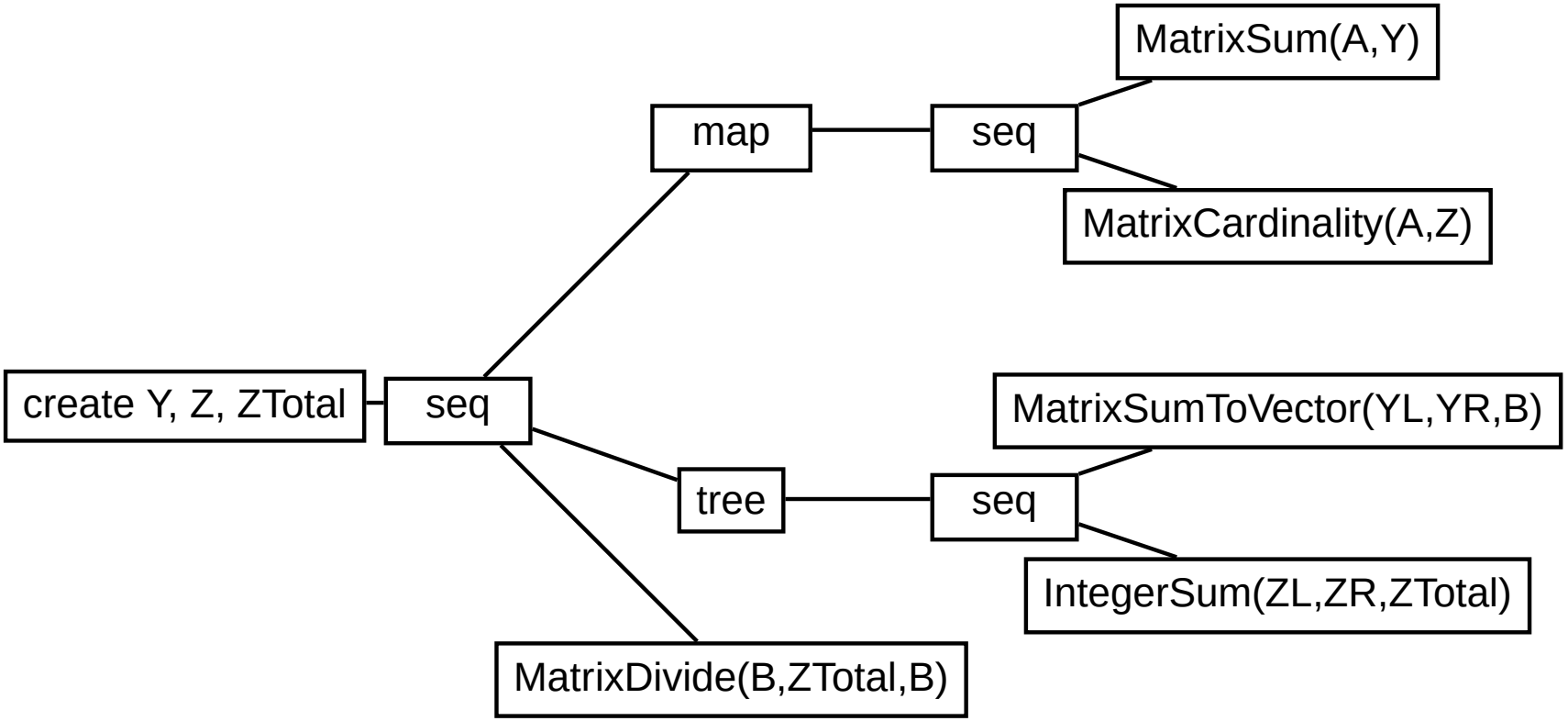
Example

```
   tree((YL,YR)\Y -> B, (ZL,ZR)\Z -> ZTotal){
      matrixSumToVector:uri1(YL,YR,B);
      integerSum:uri1(ZL,ZR,ZTotal);
   }

   matrixDivide:uri1(B,Z_Total,B);
}
```

climateprediction.net

Example



climateprediction.net

JIT Compliers

- Produce XML to represent the concrete syntax tree
- Use a JIT compiler to convert this to the workflow language of your choice
- Execute on existing resources
- Store returned results

climateprediction.net

Conclusions

climateprediction.net

Conclusions

- Solution to a new style of problem

climate*prediction*.net

Conclusions

- Solution to a new style of problem
- Can be used to build on existing resources with relative ease and without affecting their existing functionality

climateprediction.net

Conclusions

- Solution to a new style of problem
- Can be used to build on existing resources with relative ease and without affecting their existing functionality
- Ideas can be absorbed into the next generation of workflow languages

climateprediction.net

Conclusions

- Solution to a new style of problem
- Can be used to build on existing resources with relative ease and without affecting their existing functionality
- Ideas can be absorbed into the next generation of workflow languages

More information please email

`Daniel.Goodman@comlab.ox.ac.uk`

climate*prediction*.net