result2 = pa.scatter([0,0,1,1,2,2], chooseMax);     // <1,3,5,undefined,undefined, undefined>
result = pa.scatter([0,3,1,4,2,5]);                 // <0,2,4,1,3,5>
pa = new ParallelArray([0,1,2,3,4,5]);

function chooseMax(valA, valB){

===Example: Resolve conflict with the larger value===

s length.
Value to place in ''result[indices[index]]''
To ensure determinism it is the programmer
*  ''this'' the entire ParallelArray

===Arguments===
<code>conflictFunction(valA, valB)</code>

A conflict occurs when multiple elements are scattered to the same location. It results in a call to ... third argument to scatter. If ''conflictFunction'' is undefined, ''scatter'' throws an exception when a conflict occurs.

===Example: an identity function===

-  ''A[indices[i]] = conflictFunction(valA, valB)'' when multiple elements are scattered to the same location. see below
-  ''A[indices[i]] = this[i]'', when ''indices[i]'' is unique

*  ''indices'' array of indices in the resulting array\n
===Arguments===
<code>myArray.scatter(indices, defaultValue, conflictFunction, length)</code>

Typically the programmer will only call ''scan'' with associative and commutative functions but there is nothing ... will lead to a result that is guaranteed only to be consistent with some ordering of applying the elemental function.

===Discussion===
<code>pa.scan(function(a, b){return b;})</code>

===Returns===
<code>function (a, b) </code>

===Elemental Function===
''elementalFunction'' described below
<code>myArray.scan(elementalFunction)</code>

===Synopsis===
====scan====
''reduce'' also requires the elemental function be commutative since ''reduce'' can induce reordering of the arguments passed to the elemental functions.
Reduce is free to group calls to the elemental function and reorder the calls. For an elemental function that is ... Average(3, 9)) produces the value 4. ''reduce'' is permitted to chose whichever call ordering it finds convenient.

===Discussion===
The final value, if the ParallelArray has only 1 element then that element is returned.
*  ''this'' The entire ParallelArray

''elementalFunction'' described below
<code>myArray.reduce(elementalFunction)</code>

===Synopsis===
====ParallelArray ====
myPlusPA = myPA.map(function(element){return element+1;});             // <2, 3, 4>
This simple example creates a three element ParallelArray myPA using new. It then uses the prototype method ''map'' and ... element+1;} '' to create a freshly minted ParallelArray ''myPlusPA ''with each element in ''myPA'' incremented by 1.

====Example====

====Prototype Methods====
The system will determine if the elemental function can be optimized to take advantage of data parallel hardware. If ... data, e.g. ParallelArray objects that contain only floating point numbers, are good candidates for optimization.

We add ParallelArray, a new data type, to EcmaScript. ParallelArray is a read only array-like data structure that is ... object can be created by specifying an iteration space and providing a function that maps indices to values.

We have chosen a set of parallel constructs that we feel is minimal and upon which other data parallel constructs can be ... a "do few things well" implementation strategy while ensuring the composability needed to build other abstractions.

*  The concept of an elemental function which is passed to the parallel constructs and typically returns a single data element.

The design of RiverTrail is based on three pillars:

======Three Pillar Approach======

======Proposal======
The goal of this proposal is to enable data-parallelism in web applications. Browser applications and in particular ... hardware due to the lack of appropriate programming models. This proposal puts the parallel computing power of client
== filter ==

### Synopsis

```
myArray.filter(elementalFunction)
```

### Arguments

''elementalFunction'' described below

### Elemental Function

```
function (index1, index2, . . .)
```

*   ''this'' The ParallelArray
*   ''index1, index2, . . .'' The indices specifying the location in ''this'' where the source element is found.

The result of the elemental function is interpreted as a truthy value. If the result equals ''true'', the corresponding element will be included in ''filter'''s result. Otherwise it will be omitted.

### Returns

A freshly minted ParallelArray holding the source elements for which the results of applying the elemental function ... order of the elements in the returned ParallelArray is the same as the order of the elements in the source ParallelArray.

### Throws

When ''elementalFunction'' is not a function.

### Example: an identity function

```
result = pa.filter(function(){return true;})
```

== flatten ==

### Synopsis

```
myArray.flatten()
```

### Arguments

none

### Returns

A freshly minted ParallelArray whose outermost two dimensions have been collapsed into one.

### Example

```
pa = new ParallelArray([ [1,2], [3,4] ])
pa.flatten()                              // <1,2,3,4>
```

== partition ==

### Synopsis

```
myArray.partition(size)
```

### Arguments

''size'' the size of each element of the newly created dimension; the outermost dimension of ''myArray'' needs to be divisible by ''size''.

### Returns

A freshly minted ParallelArray where the outermost dimension has been partitioned into elements of size ''size''.

### Example

```
pa = new ParallelArray([1,2,3,4])
pa.partition(2)                        // <<1,2>,<3,4>>
```

### Discussion

While one could implement both ''flatten'' and ''partition'' using the other constructs we call them out here to make it easy for the compiler to recognize ''flatten'' or ''partition'' and make optimizations easier.

### Throws

When outermost dimension is not divisible by ''size''.

== [] ==

### Synopsis

```
myArray[index];
```

### Arguments

''index'' a number value representing a valid index in the outermost dimension of ''myArray''.

### Returns

The value found at ''index'' or ''undefined'' if no such value exists. If ''myArray'' is a multi-dimensional array, ''[]'' can be used to further slice the ParallelArray or ultimately select single elements.

### Example

```
pa = new ParallelArray([ [0,1,2,3,4], [10,11,12,13,14], [20,21,22,23,24] ])
pa[1];                           // <10,11,12,13,14>
```

### Discussion

Since ParallelArrays are immutable using ''[]'' as part of the left hand side of an assignment is not allowed and results in a throw.

== get ==

### Synopsis

```
myArray.get(indices);
```

### Arguments

''indices'': an array of number values that represent valid indices into ''myArray''. The first index references the outer most dimension, the second index references the next dimension and so forth.

### Returns

The value found at the indices or ''undefined'' if no such value exists.

### Throws

If ''indices'' is not an array like object or if the length of ''indices'' is larger than the number of dimensions in the source array.

### Example

```
pa = new ParallelArray([ [0,1,2,3,4], [10,11,12,13,14], [20,21,22,23,24] ])
pa.get([1,1]);                    // 11 same as pa[1][1].
pa.get([1]);                      // <10,11,12,13,14>, same as pa[1].
```

== length ==

### Synopsis

```
myArray.length
```

### Returns

The toplevel (first dimension) length of the ParallelArray.

### Example

```
pa = new ParallelArray([1,2,3,4])       // <1,2,3,4>
pa.length                               // 4
```

== shape ==

### Synopsis

```
myArray.shape
```

### Returns

An Array containing the length of each dimension of the ParallelArray starting with the outermost dimension.

### Discussion

''pa.shape.length'' gives the dimensionality of the parallel array.

### Example

```
pa = ParallelArray([ [1,2,3], [4,5,6] ])
// <<1,2,3>,<4,5,6>>
pa.shape                                        // [2, 3]
```