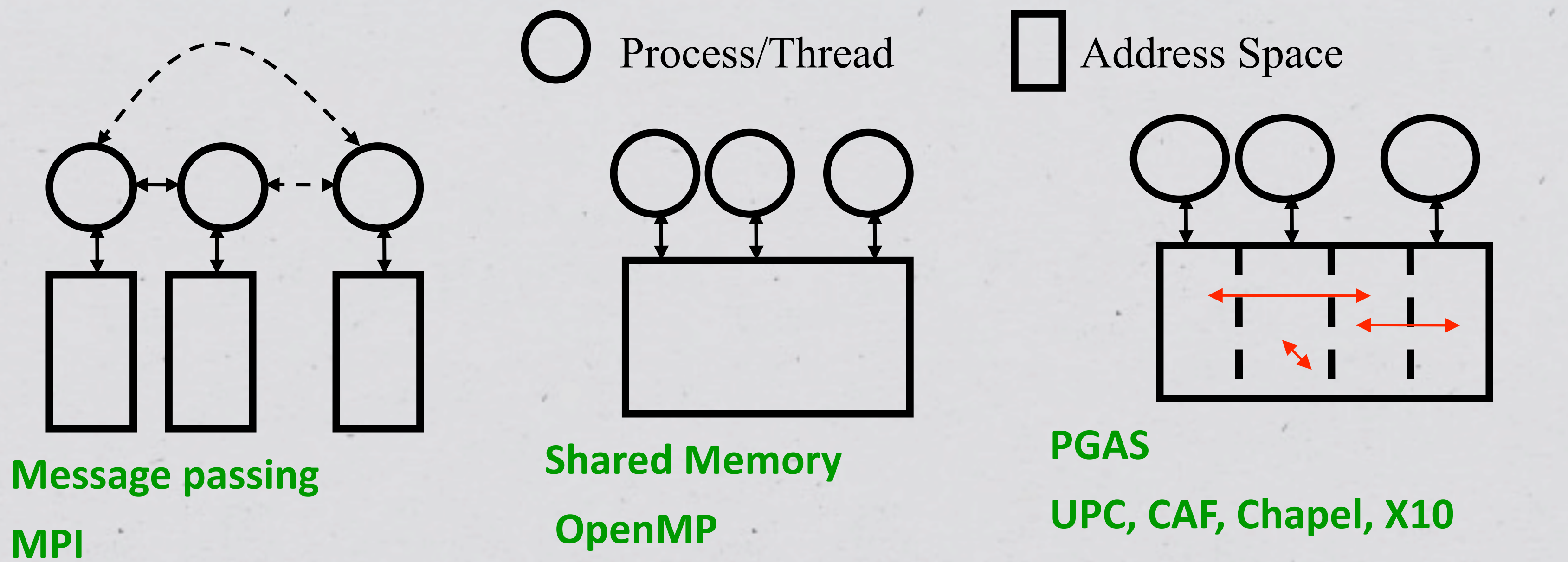
ACCUMULATOR VARIABLES

Extending the X10 language

Nathaniel Clinger
Tanay Tandon
Jaya Allamsetty
Neha Srivastav

What is Partitioned Global Address Space



Computation is performed in multiple places.
A place contains data that can be operated on remotely.
Data lives in the place it was created, for its lifetime.

A datum in one place may reference a datum in another place.
Data-structures (e.g. arrays) may be distributed across many places.
Places may have different computational properties (e.g. PPE, SPE,GPU, ...).

A place expresses locality.

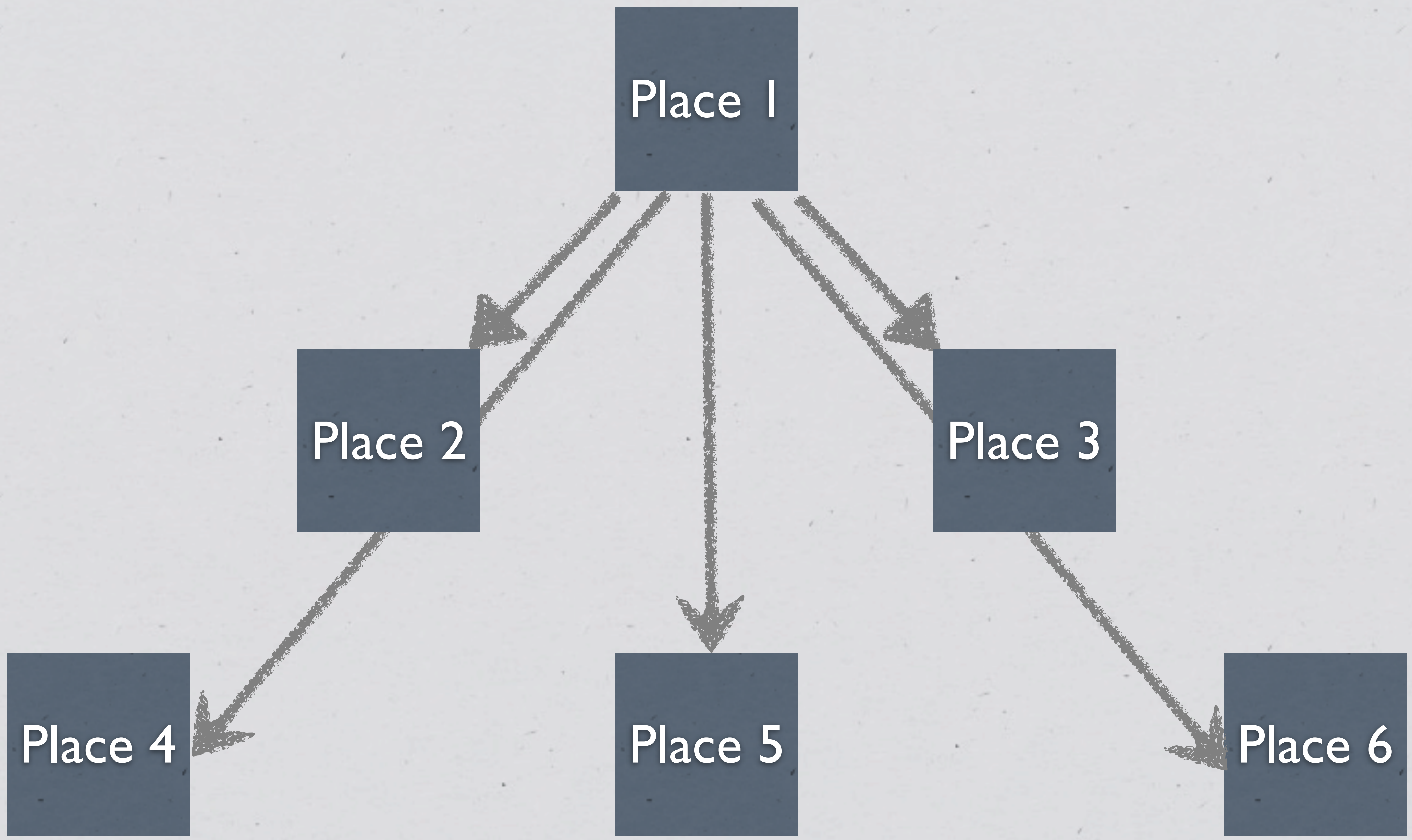
http://x10.codehaus.org/X10+2.1+Tutorial+%28SC+2010%29

Hello Whole World

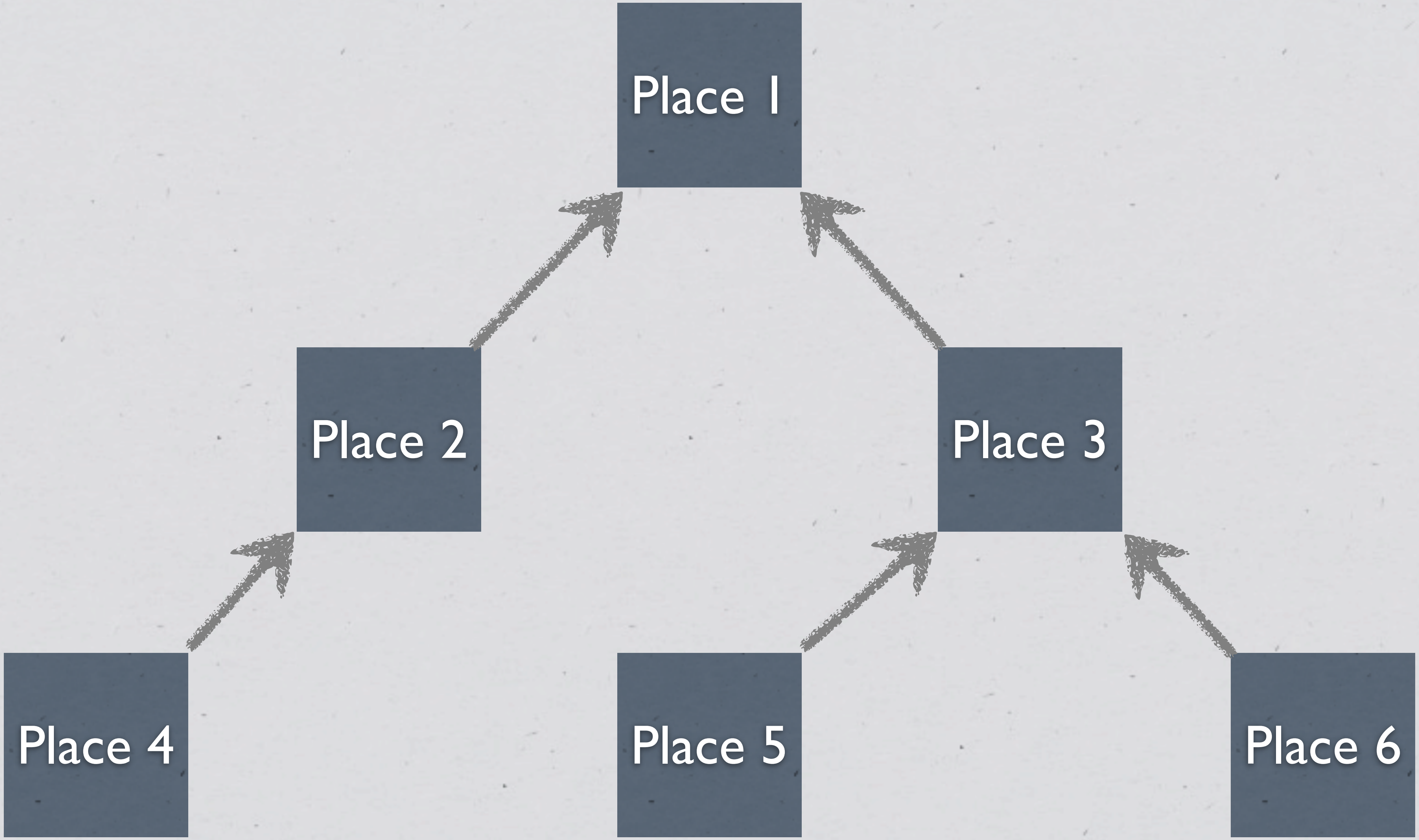
```
import x10.io.Console;

class HelloWholeWorld {
  public static def main(Array[String]) {
    finish for (p in Place.places()) {
      async at (p)
        Console.OUT.println("Hello World from place" +p.id);
    }
  }
}

(%1) x10c++ -o HelloWholeWorld -O HelloWholeWorld.x10

(%2)  runx10 -n 4 HelloWholeWorld
Hello World from place 0
Hello World from place 2
Hello World from place 3
Hello World from place 1

(%3)
```

http://x10.codehaus.org/X10+2.1+Tutorial+%28SC+2010%29

Accumulator Variable



Accumulator Variable



Accumulator Syntax

- acc myAcc:Int = Reducer() ;
  - initiate a new acc n to type Int with a reducer
- myAcc = 5 ;
  - Add value 5 to the reducer
- var result = myAcc ;
  - Read the result from myAcc and store it in result

Initialization

* class c()
```
{
    acc x:Int = IntReduce() ;    //  ERROR:  Cannot initialize field

    def m()
    {
        acc x2:Int = IntReducer() ;   // This is fine
    }
}
```

Read-Write and Write-Only

```
acc x:Int = IntReduce() ;
x = 5 ;
var r1 = x ;    // In Read-Write state so legal
finish
{
    x = 2 ;       // In Write-Only
    var r2 = x ;    // ERROR: In Write-Only state
}
var r3 = x ;   // Back in Read-Write state
```

No-Write State

* acc x:Int = IntReduce() ;

```
async
{
    x = 5 ;    // ERROR: No-Write state
    var r4 = x ;    // ERROR: Cannot read either!
}
```

Passing to a method

* acc x:Int = IntReduce() ;

```
m( x ) ;    // ERROR: Cannot use in method call outside of finish

finish
{
    m( x ) ;    // Can be passed to a method now
}

def m( x:Int ) {    ... }
```

Prevent acc escaping to heap

- Acc cannot be captured by a closure
  - ```
    acc i:Int = new IntReducer()
    val closure = ()=>i ;  // ERROR: Cannot capture an acc
    ```
- Acc cannot be capture by method
  - ```
    val anon = new Object() {
        def m() = i ;
    } ;
    ```

Some other static checks

* Acc cannot be a type
  * Array[acc] ;   // ERROR
* Acc must be initialized with a reducer
  * acc i:Int ;   // ERROR

Runtime

- Loads the environment and gets the information about Max threads, static threads, etc. that are permitted for this instance.
- Runtime has methods for explicit memory management like alloc and dealloc of objects.
- Runtime has methods defined for initiating work stealing in local or remote places by polling.
- Runtime acquires a worker thread, locks it and then releases it.

Runtime cont.

* Every worker has a queue, activity and ID bound to it. As well as methods for push or steal activities from a queue.
* Runtime has methods for starting collecting finish, stopping collecting finish, running activities at remote places, etc.

Collecting Finish

- Collection Finish is a special type of finish implementation
- Collection Finish has an additional accept method, which performs reduction over a SINGLE variable that is shared across all the activities.
- All the activities (worker threads) can perform reduction to that SINGLE variable.
- The single variable is implicit and cannot be explicitly handled.

Collection Finish cont.

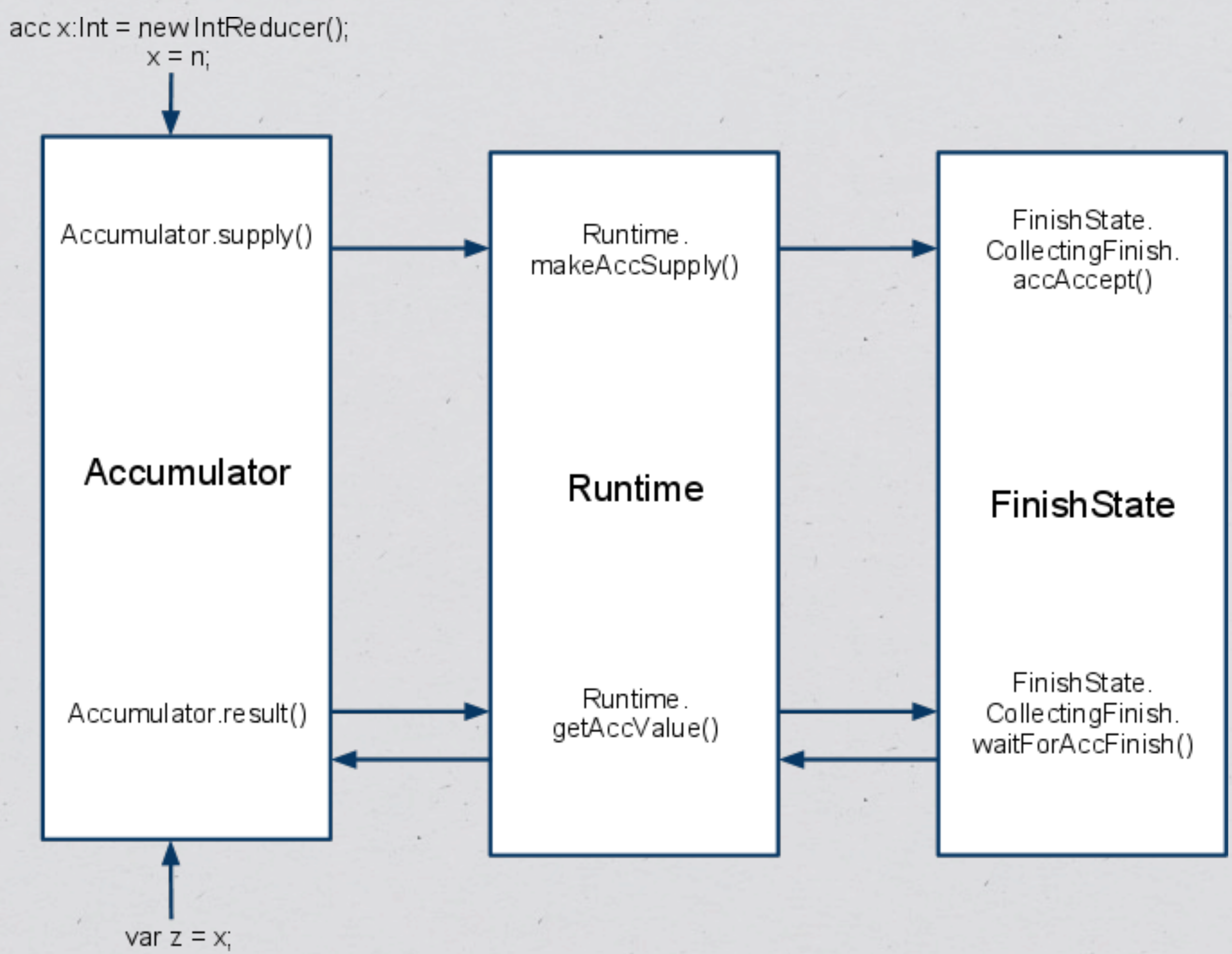
- At the end of the Collection Finish, a call to waitForFinishExpr is mad by the Runtime environment.
- The waitForFinishExpr ensures that all activities have been completed and also computes the final value of the Collection Finish construct.

Comparison

* ```
class FibAccumulators {
  def fib(n:Int):Int {
   acc x:Int = new IntReducer();
   finish {
     fib1(n, x);
   }
   return x;
  }
  def fib1(n:Int, acc z:Int) {
    if (n < 2) {
       z=n;
      return;
    }
    async fib1(n-1, z);
    fib1(n-2, z);
  }
}
```

* ```
class CollectingFinish_Fib {
  def fib(n:Int):Int {
   var x:Int;
   x = finish (new IntReducer()) {
     fib1(n);
   };
   return x;
  }
  def fib1(n:Int) offers Int {
    if (n < 2) { offer n; return; }
    async fib1(n-1);
    fib1(n-2);
  }
}
```

Control Flow



Thank you :)