# A Computing Model for Large-Scale Reconfigurable Systems

Carl Ebeling, Scott Hauck, Larry Ruzzo, Corey Olson, Maria Kim, Cooper Clausen, Boris Kogon

University of Washington

# A Computing Model for Large-Scale Reconfigurable Systems

Or “How do you program 10,000 FPGAs?”

# A Talk in Two Parts

- Part I
  - Genome Reassembly Problem
- Part II
  - A new programming model for reconfigurable computing

# Re-sequencing a Genome

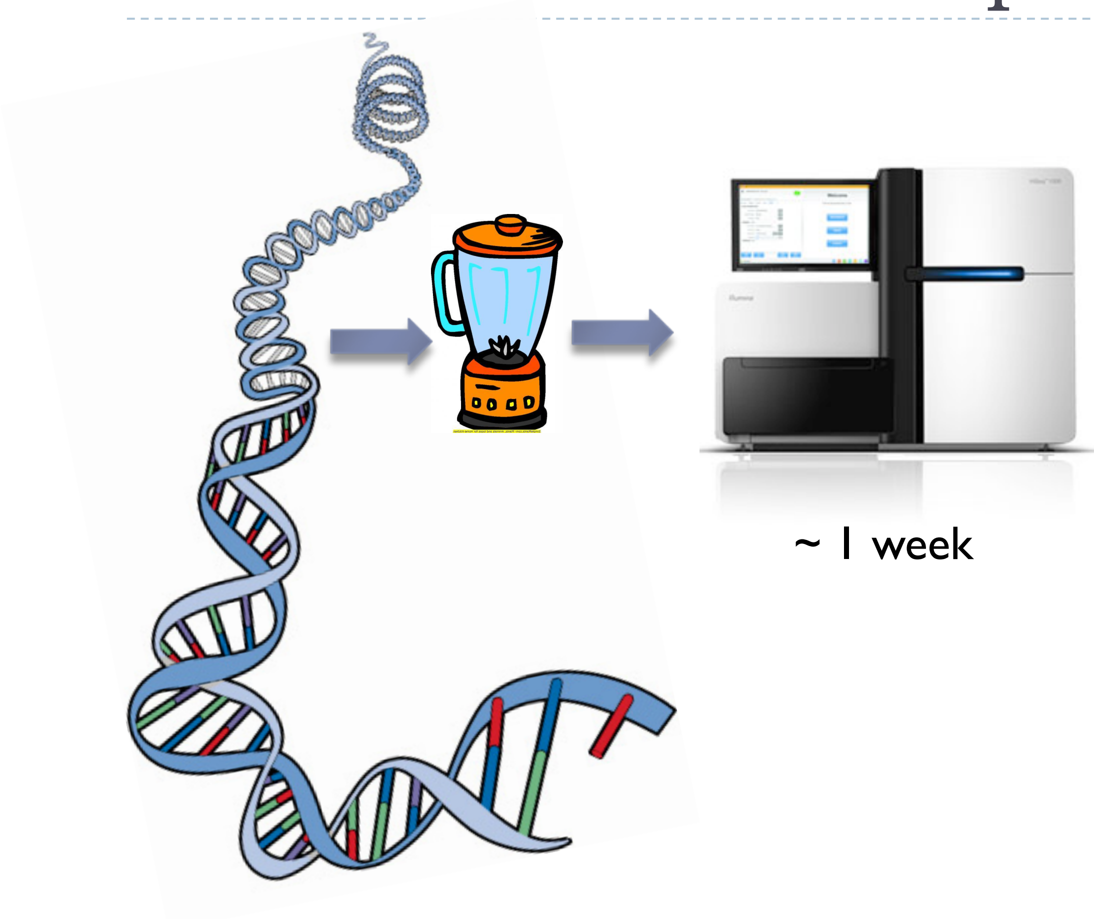
3 billion base pairs

ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATA
TCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAAC
CGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTG
TACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCG
CGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATAC
ACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGC
GATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCAT
TACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD
ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATAT
CGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACC
GCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGT
ACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGC
GCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACA
CGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCG
ATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATT
ACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDA
TTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGDATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATAT

# Cut the Genome into Short Pieces

3 billion base pairs

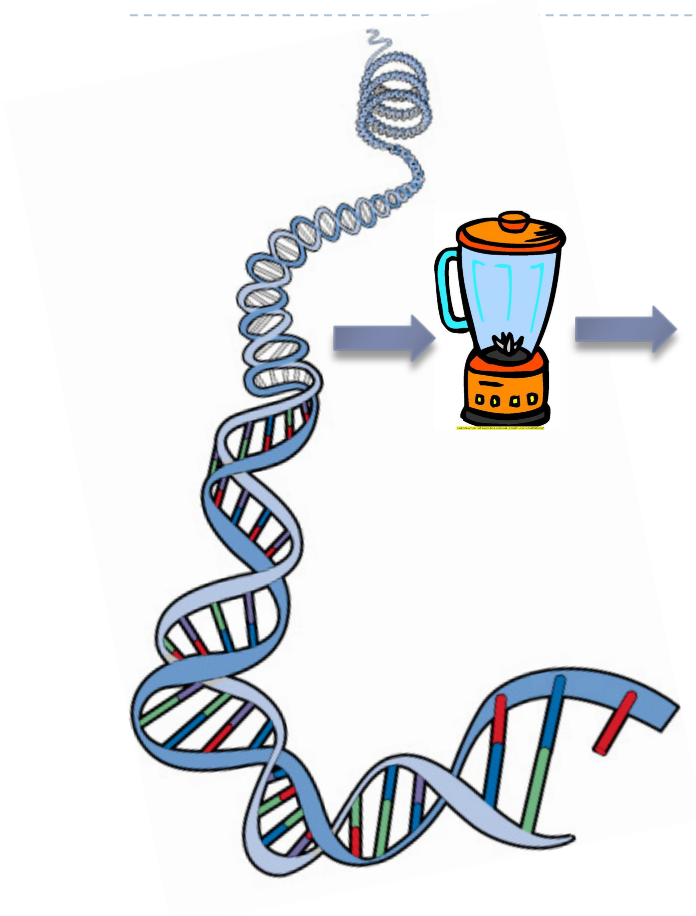
# “Read” the Short Sequences



~ 1 week

3 billion base pairs

# Produces 200 million Short Reads



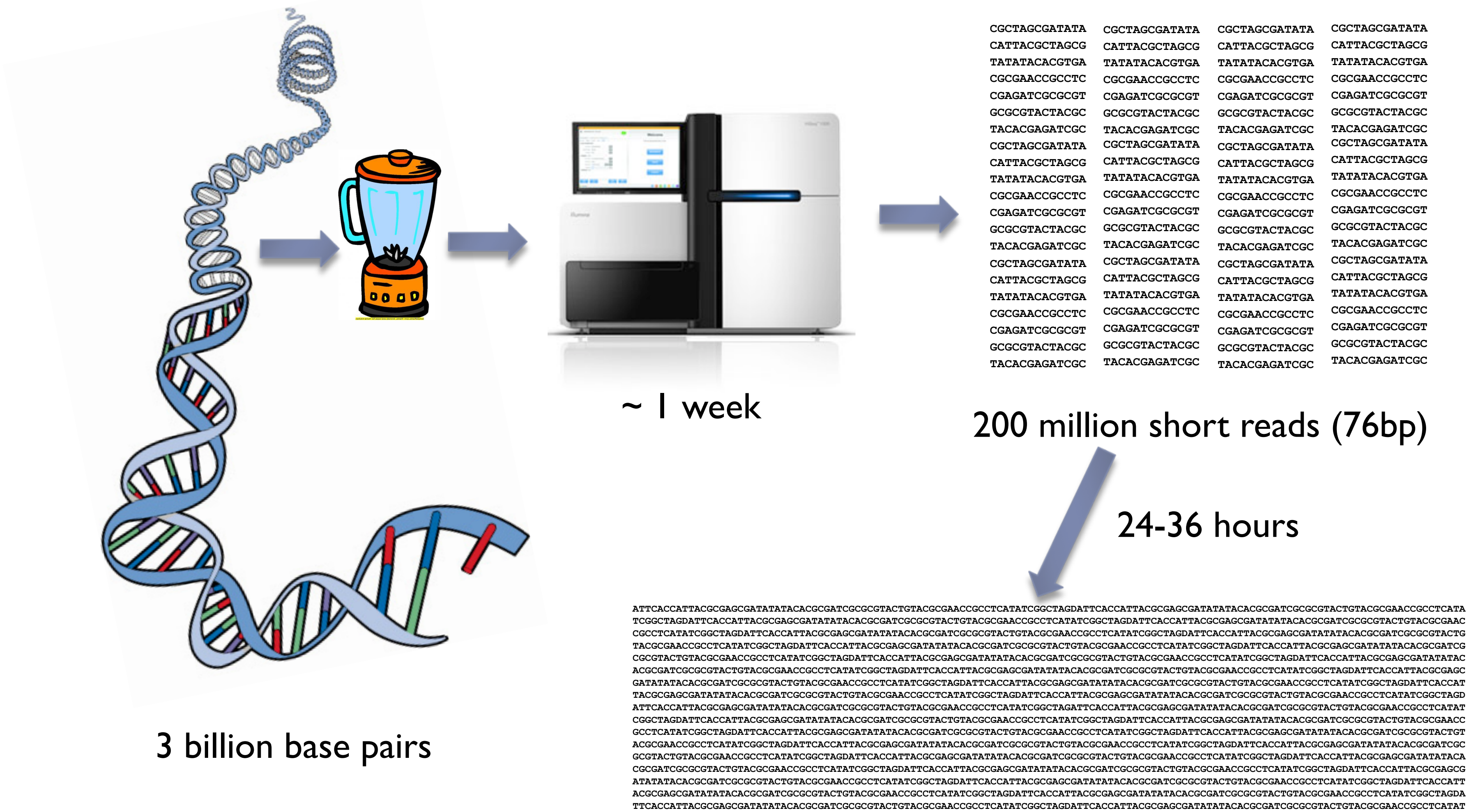
3 billion base pairs

~ 1 week

| | | | |
|---|---|---|---|
| CGCTAGCGATATA | CGCTAGCGATATA | CGCTAGCGATATA | CGCTAGCGATATA |
| CATTACGCTAGCG | CATTACGCTAGCG | CATTACGCTAGCG | CATTACGCTAGCG |
| TATATACACGTGA | TATATACACGTGA | TATATACACGTGA | TATATACACGTGA |
| CGCGAACCGCCTC | CGCGAACCGCCTC | CGCGAACCGCCTC | CGCGAACCGCCTC |
| CGAGATCGCGCGT | CGAGATCGCGCGT | CGAGATCGCGCGT | CGAGATCGCGCGT |
| GCGCGTACTACGC | GCGCGTACTACGC | GCGCGTACTACGC | GCGCGTACTACGC |
| TACACGAGATCGC | TACACGAGATCGC | TACACGAGATCGC | TACACGAGATCGC |
| CGCTAGCGATATA | CGCTAGCGATATA | CGCTAGCGATATA | CGCTAGCGATATA |
| CATTACGCTAGCG | CATTACGCTAGCG | CATTACGCTAGCG | CATTACGCTAGCG |
| TATATACACGTGA | TATATACACGTGA | TATATACACGTGA | TATATACACGTGA |
| CGCGAACCGCCTC | CGCGAACCGCCTC | CGCGAACCGCCTC | CGCGAACCGCCTC |
| CGAGATCGCGCGT | CGAGATCGCGCGT | CGAGATCGCGCGT | CGAGATCGCGCGT |
| GCGCGTACTACGC | GCGCGTACTACGC | GCGCGTACTACGC | GCGCGTACTACGC |
| TACACGAGATCGC | TACACGAGATCGC | TACACGAGATCGC | TACACGAGATCGC |
| CGCTAGCGATATA | CGCTAGCGATATA | CGCTAGCGATATA | CGCTAGCGATATA |
| CATTACGCTAGCG | CATTACGCTAGCG | CATTACGCTAGCG | CATTACGCTAGCG |
| TATATACACGTGA | TATATACACGTGA | TATATACACGTGA | TATATACACGTGA |
| CGCGAACCGCCTC | CGCGAACCGCCTC | CGCGAACCGCCTC | CGCGAACCGCCTC |
| CGAGATCGCGCGT | CGAGATCGCGCGT | CGAGATCGCGCGT | CGAGATCGCGCGT |
| GCGCGTACTACGC | GCGCGTACTACGC | GCGCGTACTACGC | GCGCGTACTACGC |
| TACACGAGATCGC | TACACGAGATCGC | TACACGAGATCGC | TACACGAGATCGC |

200 million short reads (76bp)

# Reassemble Short Reads into Genome



# Reassembly Using a Reference Genome

- Use a reference genome as a scaffold
- Very small difference between individual genomes
  - SNPs
  - CNVs
  - indels
  - Total is < 1%
- Read error rate is ~1%

# Reference

**ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD**

# Short Reads

**CGCTAGCGATATA**

**CATTACGCTAGCG**

**TATATACACGTGA**

**CGCGAACCGCCTC**

**CGAGATCGCGCGT**

**GCGCGTACTACGC**

**TACACGAGATCGC**

**ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD CGCTAGCGATATA**

**CATTACGCTAGCG**

**TATATACACGTGA**

**CGCGAACCGCCTC**

**CGAGATCGCGCGT**

**GCGCGTACTACGC**

**TACACGAGATCGC**

**ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD**

**CGCTAGCGATATA**

**CATTACGCTAGCG**

**TATATACACGTGA**

**CGCGAACCGCCTC**

**CGAGATCGCGCGT**

**GCGCGTACTACGC**

**TACACGAGATCGC**

**ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD**

**CGCTAGCGATATA**

**CATTACGCTAGCG**

**TATATACACGTGA**

**CGCGAACCGCCTC**

**CGAGATCGCGCGT**

**GCGCGTACTACGC**

**TACACGAGATCGC**

**ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD**

**CGCTAGCGATATA**

**CATTACGCTAGCG**

**TATATACACGTGA**

**CGCGAACCGCCTC**

**CGAGATCGCGCGT**

**GCGCGTACTACGC**

**TACACGAGATCGC**

ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD

CGCTAGCGATATA

CATTACGCTAGCG

TATATACACGTGA

CGCGAACCGCCTC

CGAGATCGCGCGT

GCGCGTACTACGC

TACACGAGATCGC

ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD

CGCTAGCGATATA

CATTACGCTAGCG

TATATACACGTGA

CGCGAACCGCCTC

CGAGATCGCGCGT

GCGCGTACTACGC

TACACGAGATCGC

ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD

CGCTAGCGATATA

CATTACGCTAGCG

TATATACACGTGA

CGCGAACCGCCTC

CGAGATCGCGCGT

GCGCGTACTACGC

TACACGAGATCGC

**ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD**

**CGCTAGCGATATA**

**CATTACGCTAGCG**

**TATATACACGTGA**

**CGCGAACCGCCTC**

**CGAGATCGCGCGT**

**GCGCGTACTACGC**

**TACACGAGATCGC**

**ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD**

**CGCTAGCGATATA**

**CATTACGCTAGCG**

**TATATACACGTGA**

**CGCGAACCGCCTC**

**CGAGATCGCGCGT**

**GCGCGTACTACGC**

**TACACGAGATCGC**

**ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD**

**CGCTAGCGATATA**

**CATTACGCTAGCG TATATACACGTGA**

**CGCGAACCGCCTC**

**CGAGATCGCGCGT**

**GCGCGTACTACGC**

**TACACGAGATCGC**

**ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD**

**CGCTAGCGATATA** **CGCGAACCGCCTC**

**CATTACGCTAGCG** **TATATACACGTGA**

**CGAGATCGCGCGT**

**GCGCGTACTACGC**

**TACACGAGATCGC**

**ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD**

**CGCTAGCGATATA CGAGATCGCGCGT CGCGAACCGCCTC**

**CATTACGCTAGCG TATATACACGTGA**

**GCGCGTACTACGC**

**TACACGAGATCGC**

```
ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD
           CGCTAGCGATATA    CGAGATCGCGCGT      CGCGAACCGCCTC
      CATTACGCTAGCG  TATATACACGTGA GCGCGTAC--TACGC
```

```
TACACGAGATCGC
```

```
ATTCACCATTACGCGAGCGATATATACACGCGATCGCGCGTACTGTACGCGAACCGCCTCATATCGGCTAGD
           CGCTAGCGATATA      CGAGATCGCGCGT    CGCGAACCGCCTC
      CATTACGCTAGCG TATATACACGTGA  GCGCGTAC--TACGC
                        TACACGAGATCGC
```

# Avoiding Brute Force Comparison

- Look up short reads in an index
  - A hash table of all subsequences of length 76
- Fails because of differences
- Solution: Use short subsequences to avoid differences

# Reference Index Table

- Precompiled Hash Table
  - Maps each short subsequence to locations in genome
- Example for length 4 subsequences

Candidate Alignment Locations

| | | | | |
|---|---|---|---|---|
| **AAAA** | 632 | | | |
| **AAAC** | 254 | 591 | | |
| **AAAG** | 361 | | | |
| **AAAT** | 47 | 174 | 822 | |
| … | … | … | … | … |
| **AATC** | 48 | 823 | | |
| … | … | … | … | … |
| **ATCG** | 135 | 724 | | |
| … | … | … | … | … |
| **TCGA** | 725 | | | |
| … | … | … | … | … |
| **TTTG** | 24 | 201 | | |
| **TTTT** | 23 | 200 | 275 | 301 |

# Using Seeds to Avoid Differences

Short Read

**AAAATCGAATCGA**

Candidate Alignment Locations

| | | | | |
|---|---|---|---|---|
| **AAAA** | 632 | | | |
| **AAAC** | 254 | 591 | | |
| **AAAG** | 361 | | | |
| **AAAT** | 47 | 174 | 822 | |
| … | … | … | … | … |
| **AATC** | 48 | 823 | | |
| … | … | … | … | … |
| **ATCG** | 135 | 724 | | |
| … | … | … | … | … |
| **TCGA** | 725 | | | |
| … | … | … | … | … |
| **TTTG** | 24 | 201 | | |
| **TTTT** | 23 | 200 | 275 | 301 |

# Using Seeds to Avoid Differences

AAAATCGAATCGA

AAAA

Seed

Candidate Alignment Locations

| | | | | |
|---|---|---|---|---|
| AAAA | 632 | | | |
| AAAC | 254 | 591 | | |
| AAAG | 361 | | | |
| AAAT | 47 | 174 | 822 | |
| … | … | … | … | … |
| AATC | 48 | 823 | | |
| … | … | … | … | … |
| ATCG | 135 | 724 | | |
| … | … | … | … | … |
| TCGA | 725 | | | |
| … | … | … | … | … |
| TTTG | 24 | 201 | | |
| TTTT | 23 | 200 | 275 | 301 |

# Using Seeds to Avoid Differences

AAAATCGAATCGA

AAAA

AAAT

Candidate Alignment Locations

| | | | | |
|---|---|---|---|---|
| AAAA | 632 | | | |
| AAAC | 254 | 591 | | |
| AAAG | 361 | | | |
| AAAT | 47 | 174 | 822 | |
| … | … | … | … | … |
| AATC | 48 | 823 | | |
| … | … | … | … | … |
| ATCG | 135 | 724 | | |
| … | … | … | … | … |
| TCGA | 725 | | | |
| … | … | … | … | … |
| TTTG | 24 | 201 | | |
| TTTT | 23 | 200 | 275 | 301 |

# Using Seeds to Avoid Differences

AAAATCGAATCGA

AAAA

AAAT

AATC

Candidate Alignment Locations

| | | | | |
|---|---|---|---|---|
| AAAA | 632 | | | |
| AAAC | 254 | 591 | | |
| AAAG | 361 | | | |
| AAAT | 47 | 174 | 822 | |
| … | … | … | … | … |
| AATC | 48 | 823 | | |
| … | … | … | … | … |
| ATCG | 135 | 724 | | |
| … | … | … | … | … |
| TCGA | 725 | | | |
| … | … | … | … | … |
| TTTG | 24 | 201 | | |
| TTTT | 23 | 200 | 275 | 301 |

# Using Seeds to Avoid Differences

AAAATCGAATCGA

AAAA

AAAT

AATC

ATCG

Candidate Alignment Locations

| | | | | |
|---|---|---|---|---|
| AAAA | 632 | | | |
| AAAC | 254 | 591 | | |
| AAAG | 361 | | | |
| AAAT | 47 | 174 | 822 | |
| … | … | … | … | … |
| AATC | 48 | 823 | | |
| … | … | … | … | … |
| ATCG | 135 | 724 | | |
| … | … | … | … | … |
| TCGA | 725 | | | |
| … | … | … | … | … |
| TTTG | 24 | 201 | | |
| TTTT | 23 | 200 | 275 | 301 |

# Using Seeds to Avoid Differences

AAAATCGAATCGA

AAAA

AAAT

AATC

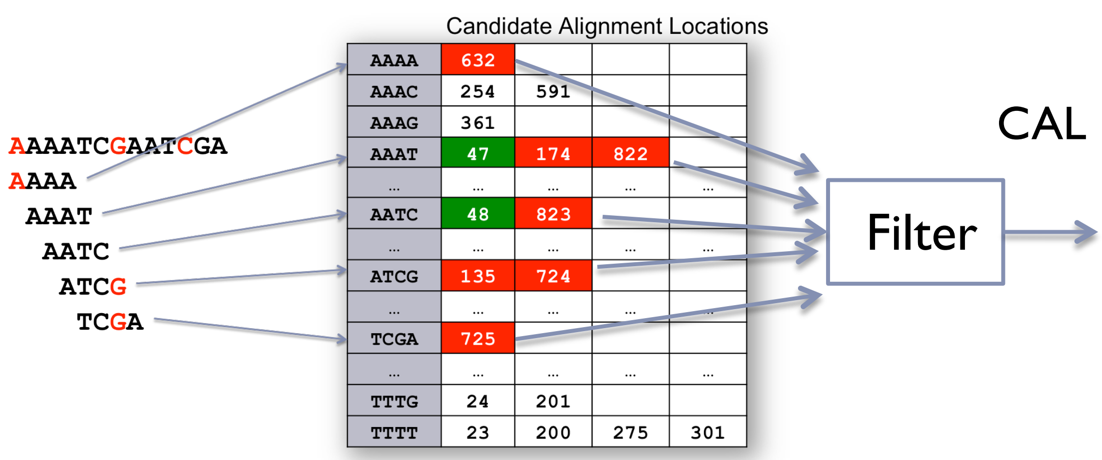
ATCG

TCGA

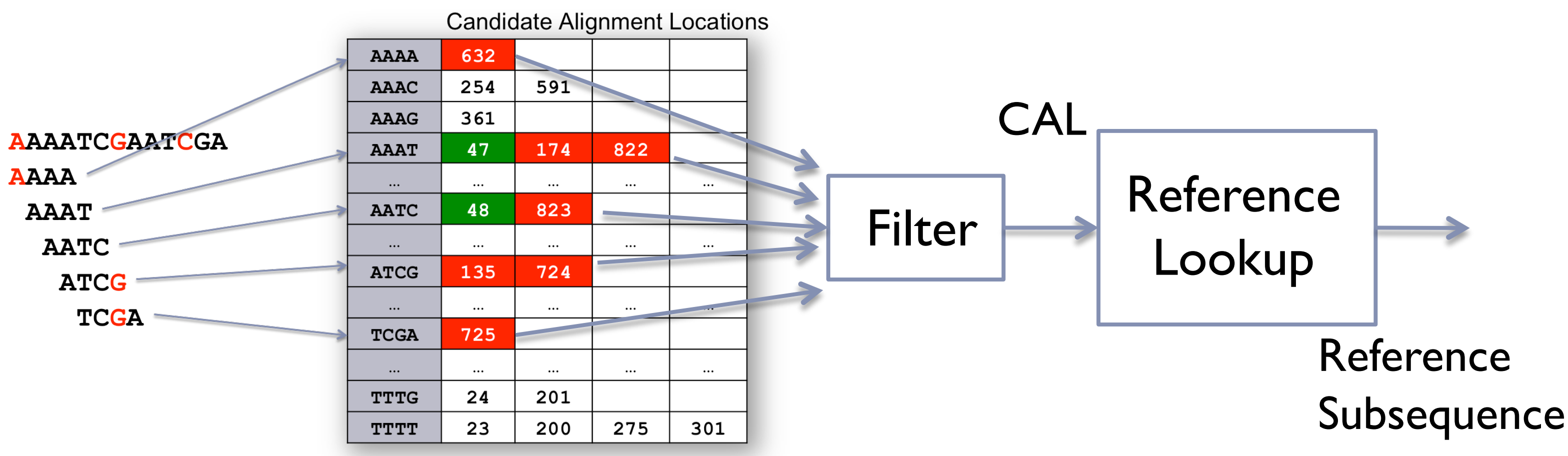
Candidate Alignment Locations

| | | | | |
|---|---|---|---|---|
| AAAA | 632 | | | |
| AAAC | 254 | 591 | | |
| AAAG | 361 | | | |
| AAAT | 47 | 174 | 822 | |
| … | … | … | … | … |
| AATC | 48 | 823 | | |
| … | … | … | … | … |
| ATCG | 135 | 724 | | |
| … | … | … | … | … |
| TCGA | 725 | | | |
| … | … | … | … | … |
| TTTG | 24 | 201 | | |
| TTTT | 23 | 200 | 275 | 301 |

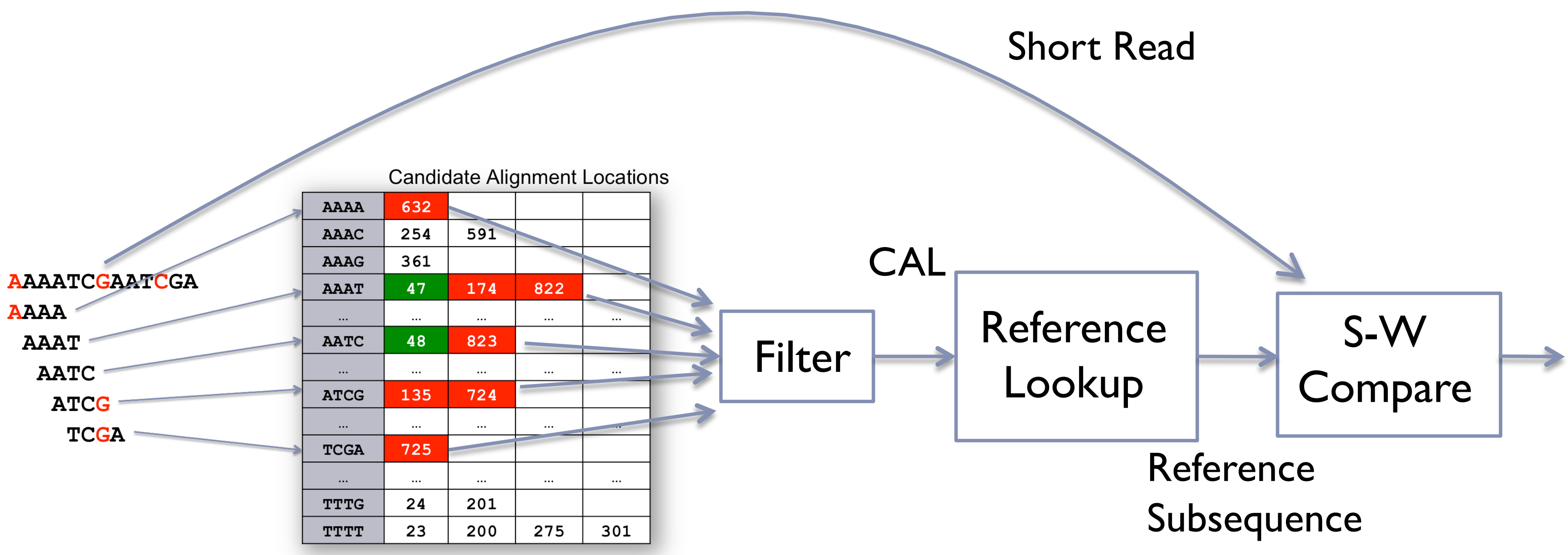
# Alignment of Each Short Read



# Alignment of Each Short Read



# Alignment of Each Short Read



# Mapping Alignment to Hardware

- Relatively straightforward algorithm
- Lots of data
- Lots of computation
- Lots of available parallelism
- Perfect for reconfigurable computing

# Mapping Alignment to Hardware

- Relatively straightforward algorithm
- Lots of data
- Lots of computation
- Lots of available parallelism
- Perfect for reconfigurable computing

- So why does it have to be so hard?!

# A Model for Programming Reconfigurable Computers

- What we are doing is closer parallel and distributed computation than traditional hardware design
- Why not leverage all the work that's already been done?

# Foundation: "Object" Model

- We call HW components "Objects"
  - More in common with SW objects than HW modules
- A system is comprised of Objects
  - Objects are static – constructed at compile-time
  - Constructors execute at compile-time
- Objects comprise:
  - Data/State that the object manages
  - Methods

# Objects

- Object Data
  - Variables (registers)
  - Arrays (memory)
  - Object data is shared by the object methods

# Objects

- Object Data
  - Variables (registers)
  - Arrays (memory)
  - Object data is shared by the object methods
- Methods
  - Methods are concurrently running HW threads
    - Methods all begin at system start
  - A method waits for a method call
    - Executes with provided parameters
    - Waits for next call

# Method Calls

- Objects interact via method calls (ala RPC)
  - object.method(parameter-list)
  - Sends parameters from source method to destination method
  - Via a "connection"
    - Direct or time-multiplexed bus/network

# Method Calls

- Asynchronous methods – "Fire and Forget"
  - a la Active Messages
  - No return – caller continues immediately
  - Return values forwarded via callback/callforward method
    - (explicit or implicit)
- Synchronous methods
  - Caller blocks until method completes (with return value)
  - Expensive except for local method calls

# Simple Example

```
bar:
  foo.init();
  M.mult(a, b);
  M.mult(c, d);
```

```
M:
  mult(int x, int y) {
    foo.add(x * y)
  }
```

```
foo:
   int sum;
   SYNC init() {
     sum = 0;
   }
   add(int val) {
     sum += val;
   }
```

# Simple Example

```
bar:
  foo.init();
  M.mult(a, b);
  M.mult(c, d);
```

```
M:
  mult(int x, int y) {
    foo.add(x * y)
  }
```

```
foo:
   int sum;
   SYNC init() {
     sum = 0;
   }
   add(int val) {
     sum += val;
   }
```

# Simple Example

```
bar:
   foo.init();
   M.mult(a, b);
   M.mult(c, d);
```

```
M:
   mult(int x, int y) {
      foo.add(x * y)
   }
```

```
foo:
    int sum;
    SYNC init() {
       sum = 0;
    }
    add(int val) {
       sum += val;
    }
```

# Simple Example

```
bar:
  foo.init();
  M.mult(a, b);
  M.mult(c, d);
```

```
M:
  mult(int x, int y) {
    foo.add(x * y)
  }
```

```
foo:
   int sum;
   SYNC init() {
     sum = 0;
   }
   add(int val) {
     sum += val;
   }
```

# Simple Example

```
bar:
  foo.init();
  M.mult(a, b);
  M.mult(c, d);
```

```
M:
  mult(int x, int y) {
    foo.add(x * y)
  }
```

```
foo:
   int sum;
   SYNC init() {
     sum = 0;
   }
   add(int val) {
     sum += val;
   }
```

# Simple Example (cont)

```
bar:
   foo.init();
   M1.mult(a, b);
   M2.mult(c, d);
```

```
M1:
  mult(int x, int y) {
  foo.add(x * y)
  }
```

```
M2:
  mult(int x, int y) {
    foo.add(x * y)
  }
```

```
foo:
   int sum;
   SYNC init() {
     sum = 0;
   }
   add(int val) {
     sum += val;
   }
```

# Simple Example (cont)

```
bar:
   foo.init();
   M1.mult(a, b);
   M2.mult(c, d);
```

```
M1:
  mult(int x, int y) {
  foo.add(x * y)
  }
```

```
M2:
  mult(int x, int y) {
    foo.add(x * y)
  }
```

```
foo:
   int sum;
   SYNC init() {
     sum = 0;
   }
   add(int val) {
     sum += val;
   }
```

# Simple Example (cont)

```
bar:
    foo.init();
    M1.mult(a, b);
    M2.mult(c, d);
```

```
M1:
  mult(int x, int y) {
  foo.add(x * y)
  }
```

```
M2:
  mult(int x, int y) {
    foo.add(x * y)
  }
```

```
foo:
    int sum;
    SYNC init() {
       sum = 0;
    }
    add(int val) {
       sum += val;
    }
```

# Simple Example (cont)

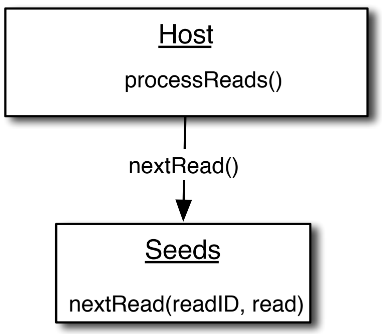
```
bar:
    foo.init();
    M1.mult(a, b);
    M2.mult(c, d);
```

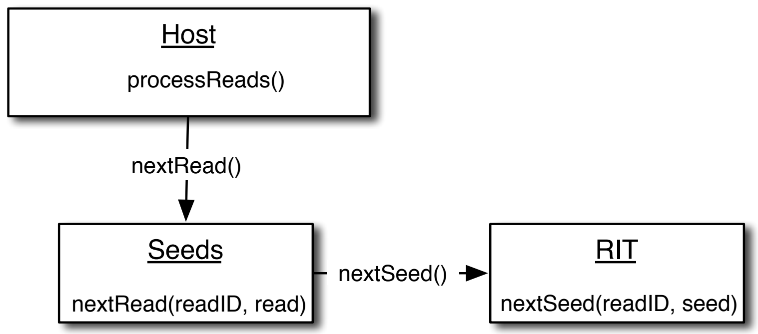
```
M1:
  mult(int x, int y) {
  foo.add(x * y)
  }
```

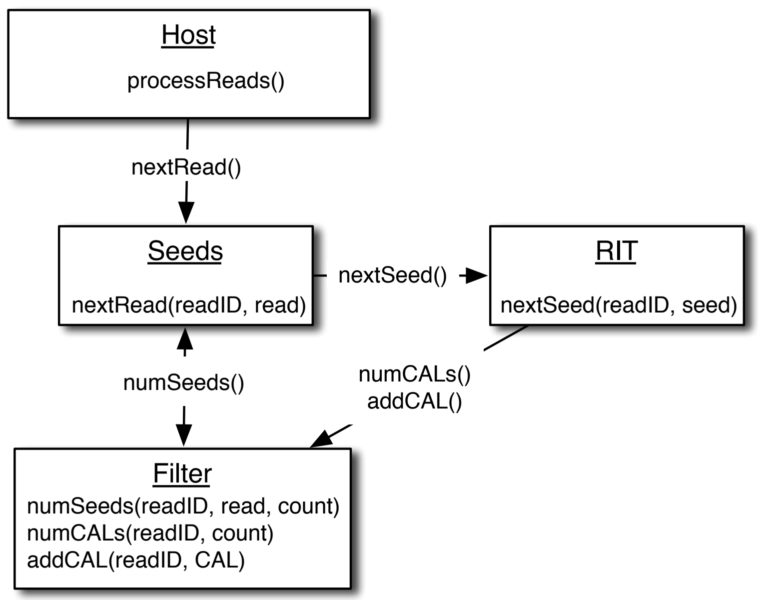
```
M2:
  mult(int x, int y) {
    foo.add(x * y)
  }
```

```
foo:
    int sum;
    SYNC init() {
      sum = 0;
    }
    add(int val) {
      sum += val;
    }
```

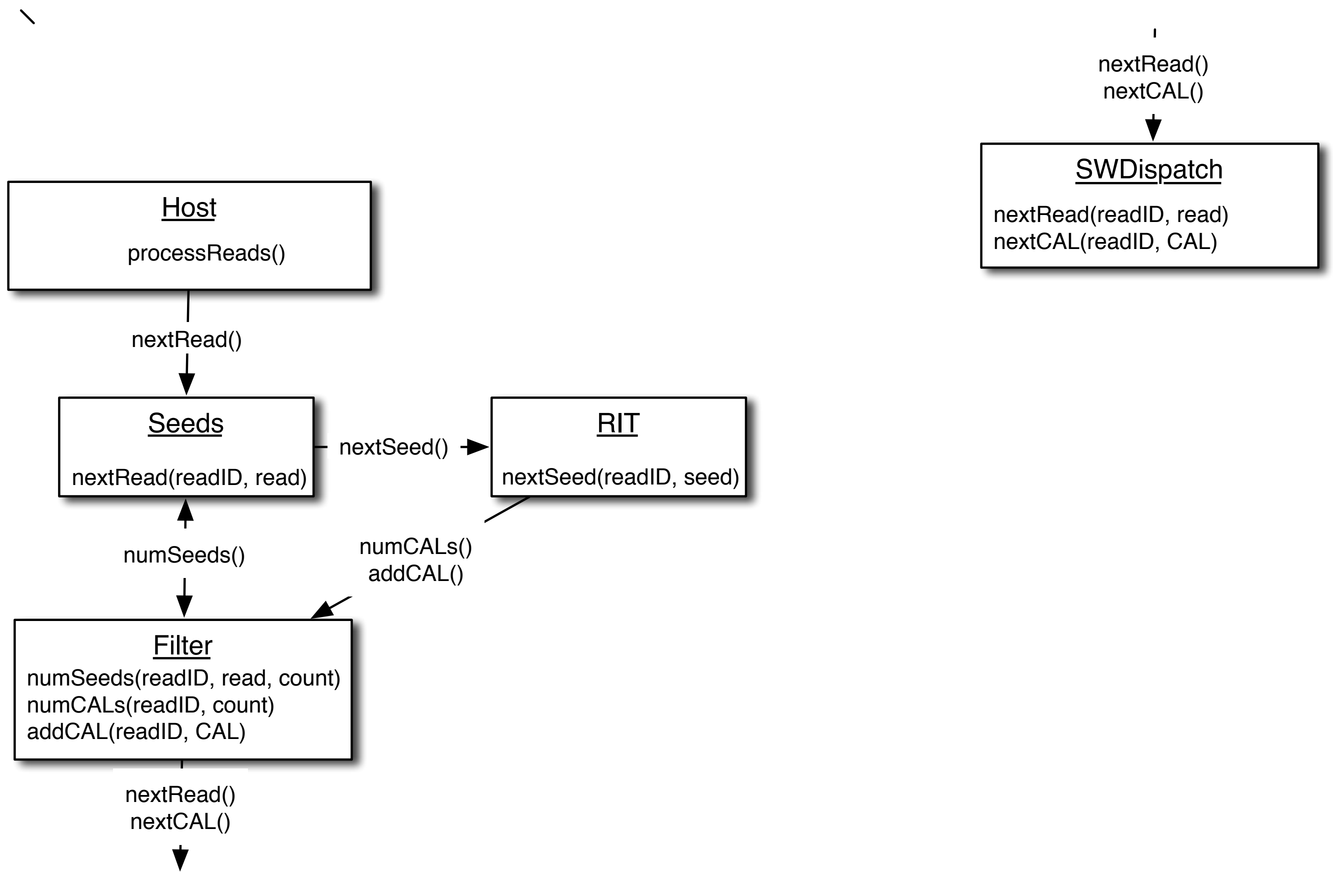
# Short Read Alignment

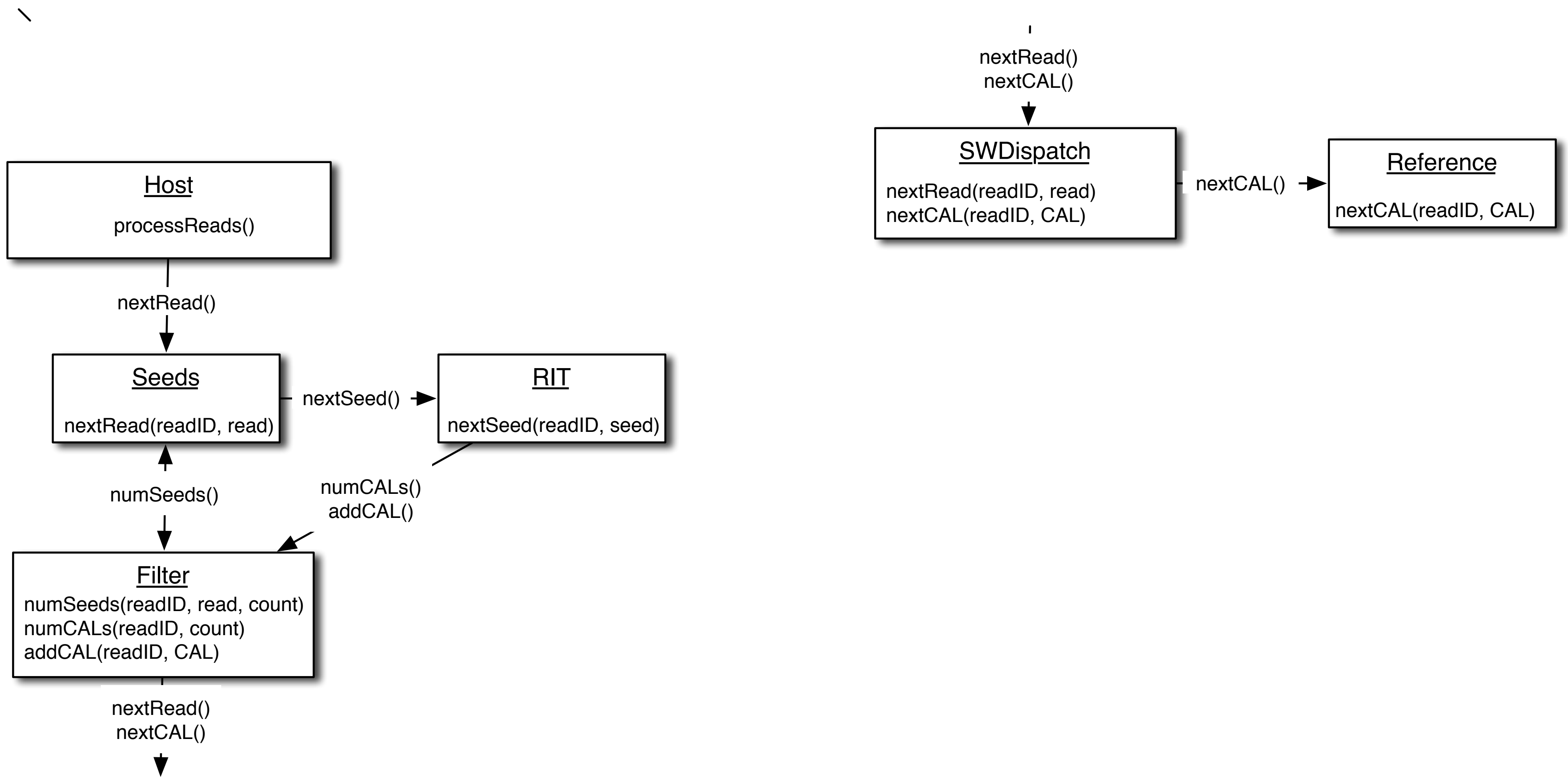



Host
processReads()
nextRead()
Seeds
nextRead(readID, read)
nextSeed()
RIT
nextSeed(readID, seed)

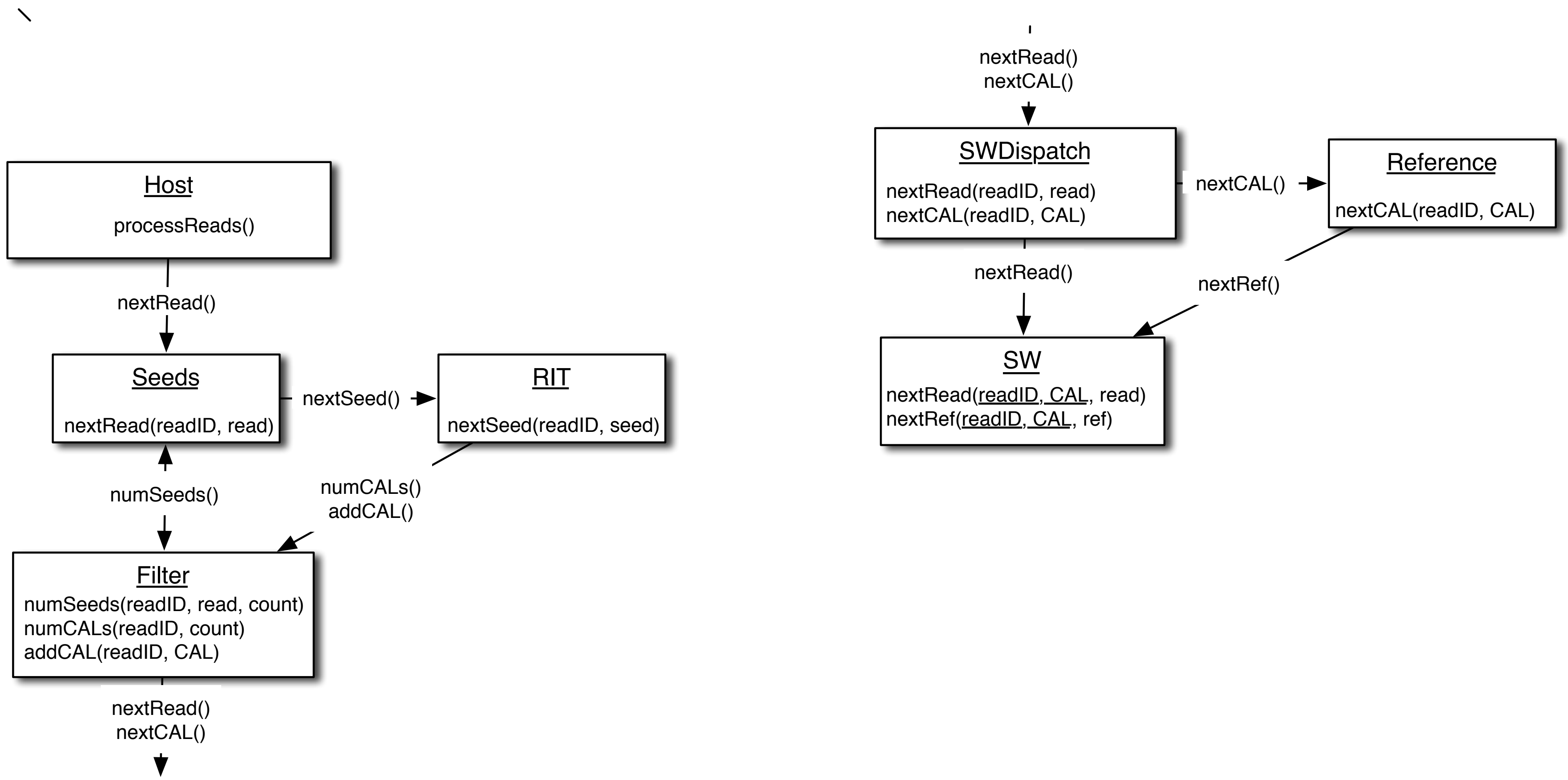


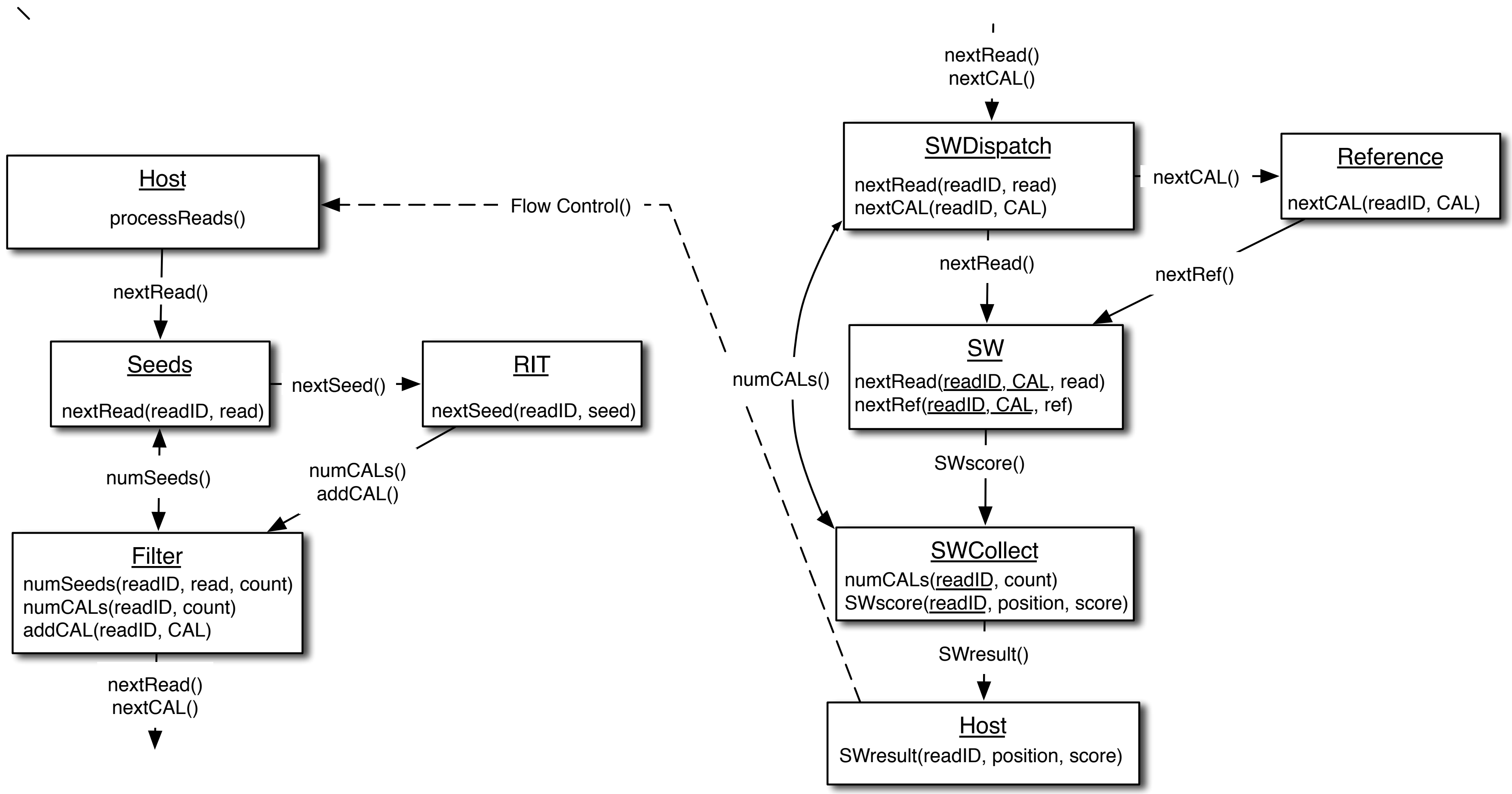
Host
processReads()
nextRead()
Seeds
nextRead(readID, read)
nextSeed()
RIT
nextSeed(readID, seed)
numSeeds()
numCALs()
addCAL()
Filter
numSeeds(readID, read, count)
numCALs(readID, count)
addCAL(readID, CAL)


Host
processReads()
nextRead()
Seeds
nextRead(readID, read)
nextSeed()
RIT
nextSeed(readID, seed)
numSeeds()
numCALs()
addCAL()
Filter
numSeeds(readID, read, count)
numCALs(readID, count)
addCAL(readID, CAL)
nextRead()
nextCAL()
nextRead()
nextCAL()
SWDispatch
nextRead(readID, read)
nextCAL(readID, CAL)



nextRead()
nextCAL()
SWDispatch
nextRead(readID, read)
nextCAL(readID, CAL)
nextCAL()
Reference
nextCAL(readID, CAL)
Host
processReads()
nextRead()
Seeds
nextRead(readID, read)
nextSeed()
RIT
nextSeed(readID, seed)
numSeeds()
numCALs()
addCAL()
Filter
numSeeds(readID, read, count)
numCALs(readID, count)
addCAL(readID, CAL)
nextRead()
nextCAL()

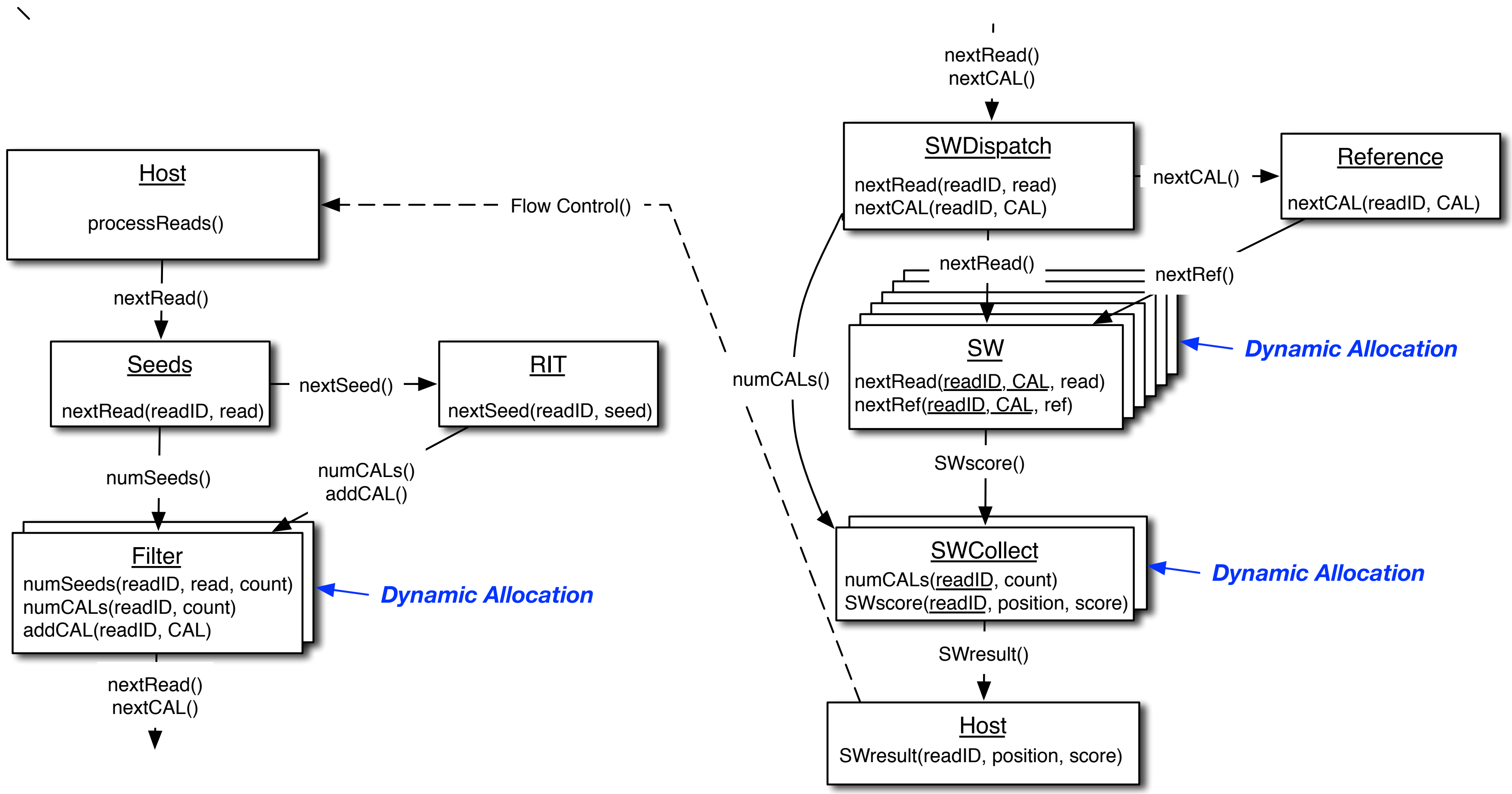

Host
processReads()
nextRead()
Seeds
nextRead(readID, read)
nextSeed()
RIT
nextSeed(readID, seed)
numSeeds()
numCALs()
addCAL()
Filter
numSeeds(readID, read, count)
numCALs(readID, count)
addCAL(readID, CAL)
nextRead()
nextCAL()
nextRead()
nextCAL()
SWDispatch
nextRead(readID, read)
nextCAL(readID, CAL)
nextCAL()
Reference
nextCAL(readID, CAL)
nextRead()
nextRef()
SW
nextRead(readID, CAL, read)
nextRef(readID, CAL, ref)


Host
processReads()
nextRead()
Seeds
nextRead(readID, read)
nextSeed()
RIT
nextSeed(readID, seed)
numSeeds()
numCALs()
addCAL()
Filter
numSeeds(readID, read, count)
numCALs(readID, count)
addCAL(readID, CAL)
nextRead()
nextCAL()
Flow Control()
numCALs()
nextRead()
nextCAL()
SWDispatch
nextRead(readID, read)
nextCAL(readID, CAL)
nextCAL()
Reference
nextCAL(readID, CAL)
nextRead()
nextRef()
SW
nextRead(readID, CAL, read)
nextRef(readID, CAL, ref)
SWscore()
SWCollect
numCALs(readID, count)
SWscore(readID, position, score)
SWresult()
Host
SWresult(readID, position, score)

# Dynamic Objects

- Automatic allocation
  - Object is allocated when method with new ID is called
  - All calls with same ID go to the same object
  - Object deallocates itself when done
- Implementation
  - Pool of objects (static)
  - Object manager
    - Allocates and maps objects
    - Directs method calls to appropriate object

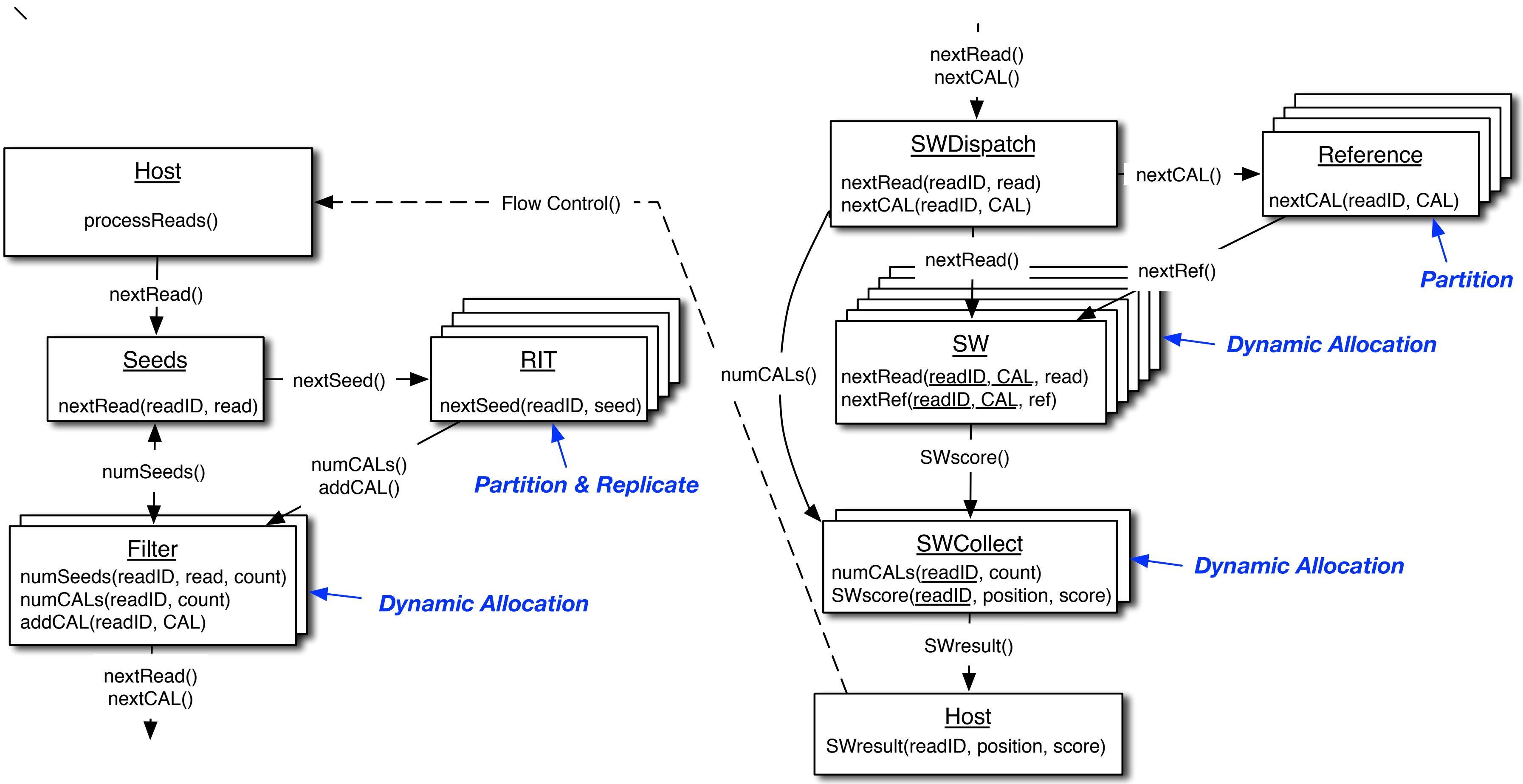
# Parallelism with Dynamic Objects



# Distributed Objects

- Partitioned memory arrays
  - Concurrent accesses
  - Increased data bandwidth
- Partitioning memory causes associated objects to be copied and distributed with partitions
- Method calls automatically directed to right partition

# Distributed Solution



# Summary

- FPGAs are great for data/compute intensive applications
  - Power efficient
  - Massively parallel
- We need a better programming model
  - Write applications using object model
  - Programmer describes the parallelism using dynamic and distributed objects
  - Compile to a distributed accelerator system
  - Compiler handles memory partitioning, object duplication and distribution, and communication between objects