LIQUID METAL

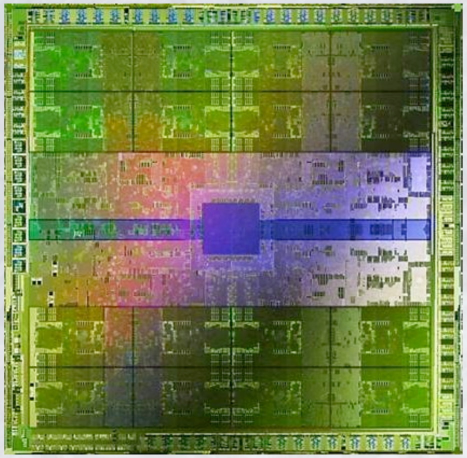
OBJECT-ORIENTED PROGRAMMING OF HETEROGENEOUS MACHINES

2011 FCCM Workshop on High-Level Synthesis and Parallel Computation Models

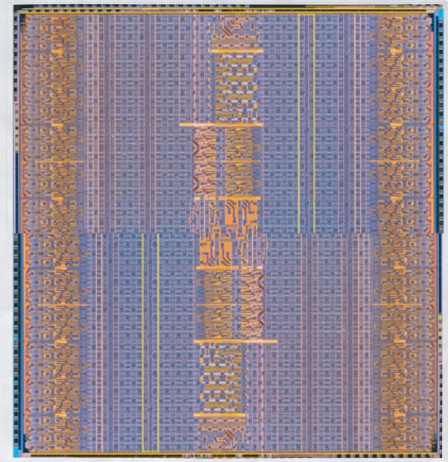
Joshua Auerbach
David F. Bacon
Perry Cheng
Steven Fink
Rodric Rabbah
Sunil Shukla

Christophe Dubach
Yu Zhang

The Heterogeneous Era

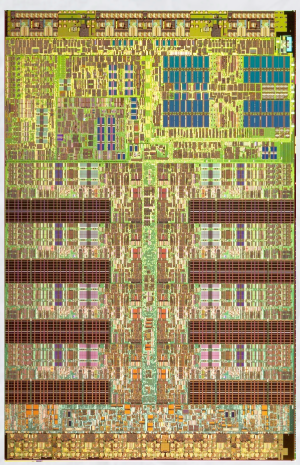


GPU

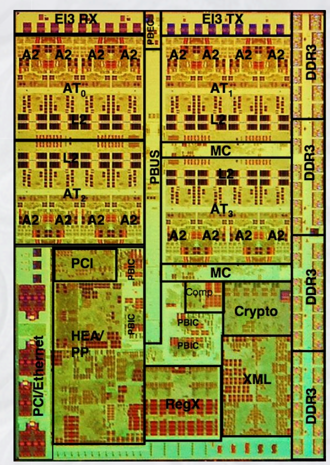


FPGA

Tilera 64



Cell BE



IBM PowerEN

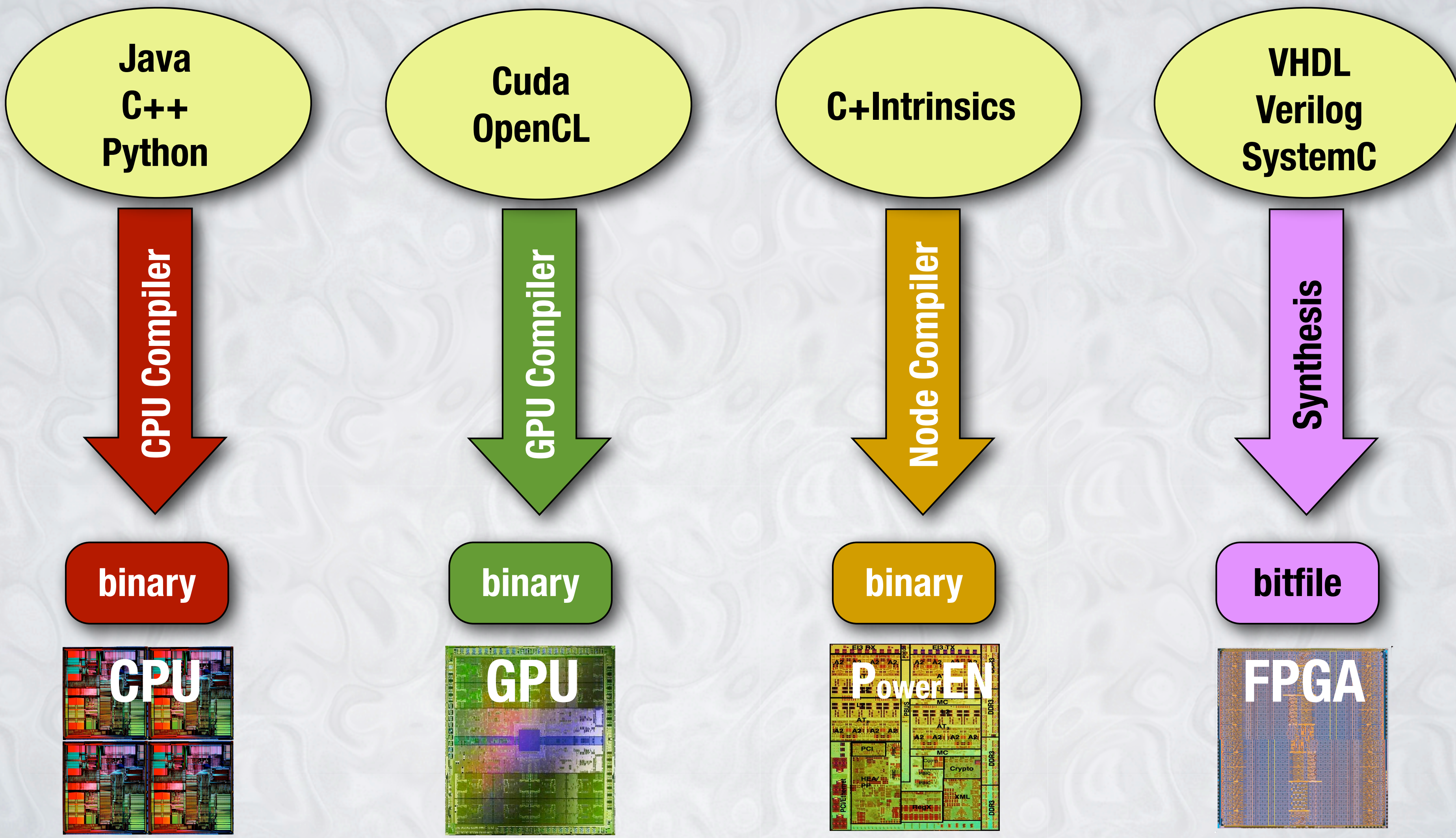
The FORTRAN Automatic Coding System

J. W. BACKUS†, R. J. BEEBER†, S. BEST‡, R. GOLDBERG†, L. M. HAIBT†, H. L. HERRICK†, R. A. NELSON†, D. SAYRE†, P. B. SHERIDAN†, H. STERN†, I. ZILLER†, R. A. HUGHES§, AND R. NUTT||

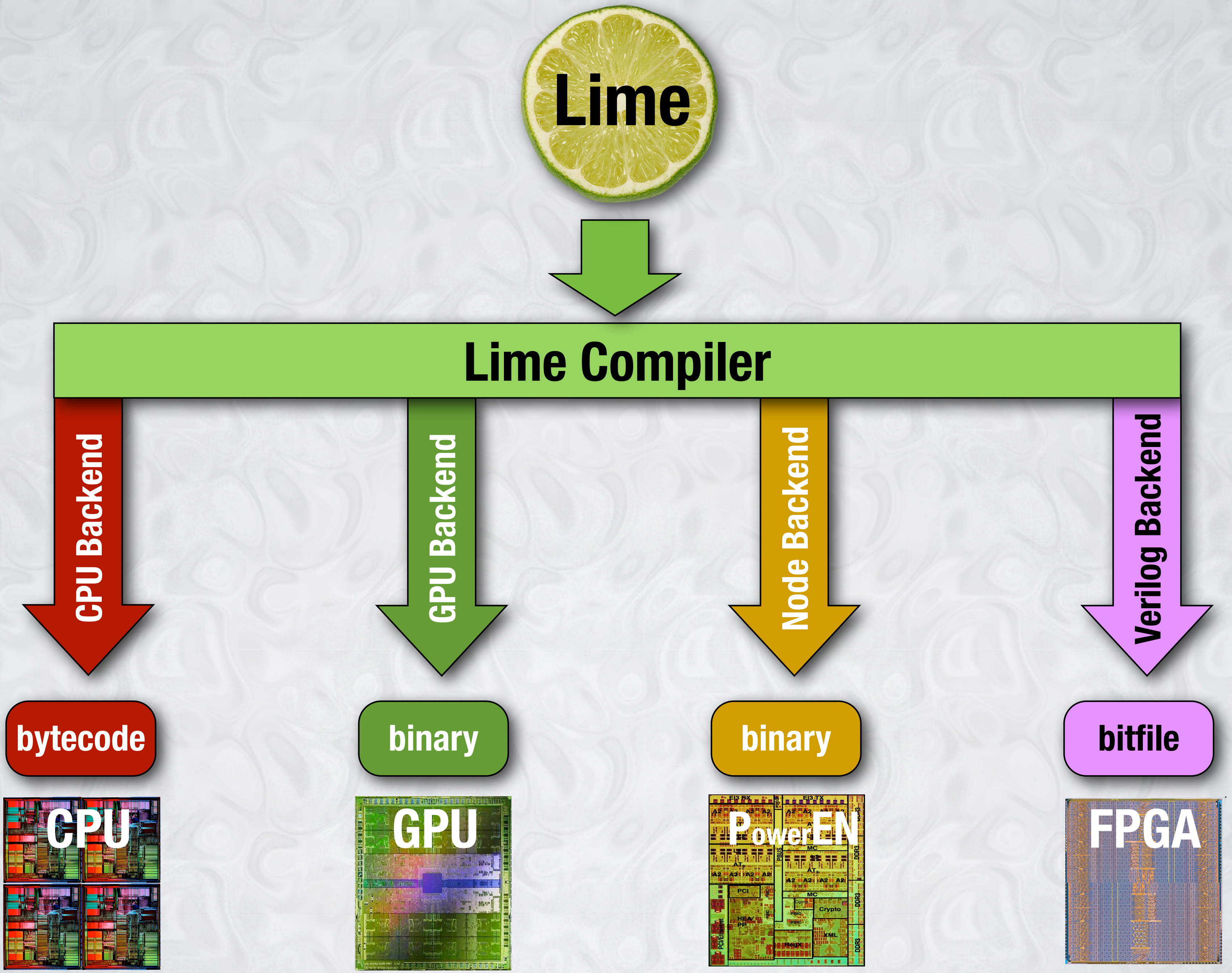
Introduction

THE FORTRAN project was begun in the summer of 1954. Its purpose was to reduce by a large factor the task of preparing scientific problems for IBM's next large computer, the 704. If it were possible for the 704 to code problems for itself and produce as good programs as human coders (but without the errors), it was clear that large benefits could be achieved. For it was known that about two-thirds of the cost of solving most scientific and engineering problems on large computers was that of problem preparation. Furthermore, more than 90 per cent of the elapsed time for a problem was usually devoted to planning, writing, and debugging the program. In many cases the de-

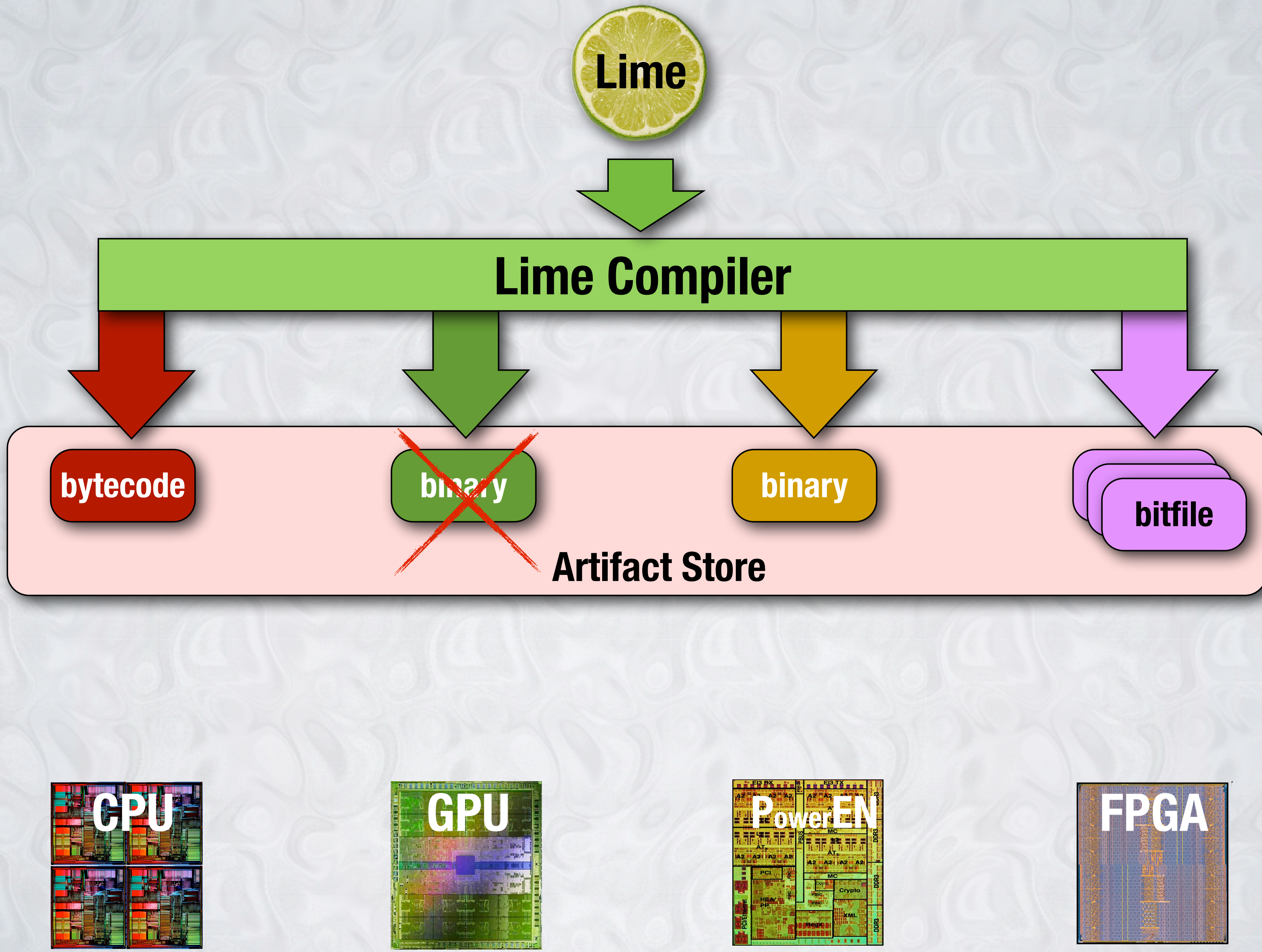
Heterogeneous Programming Today



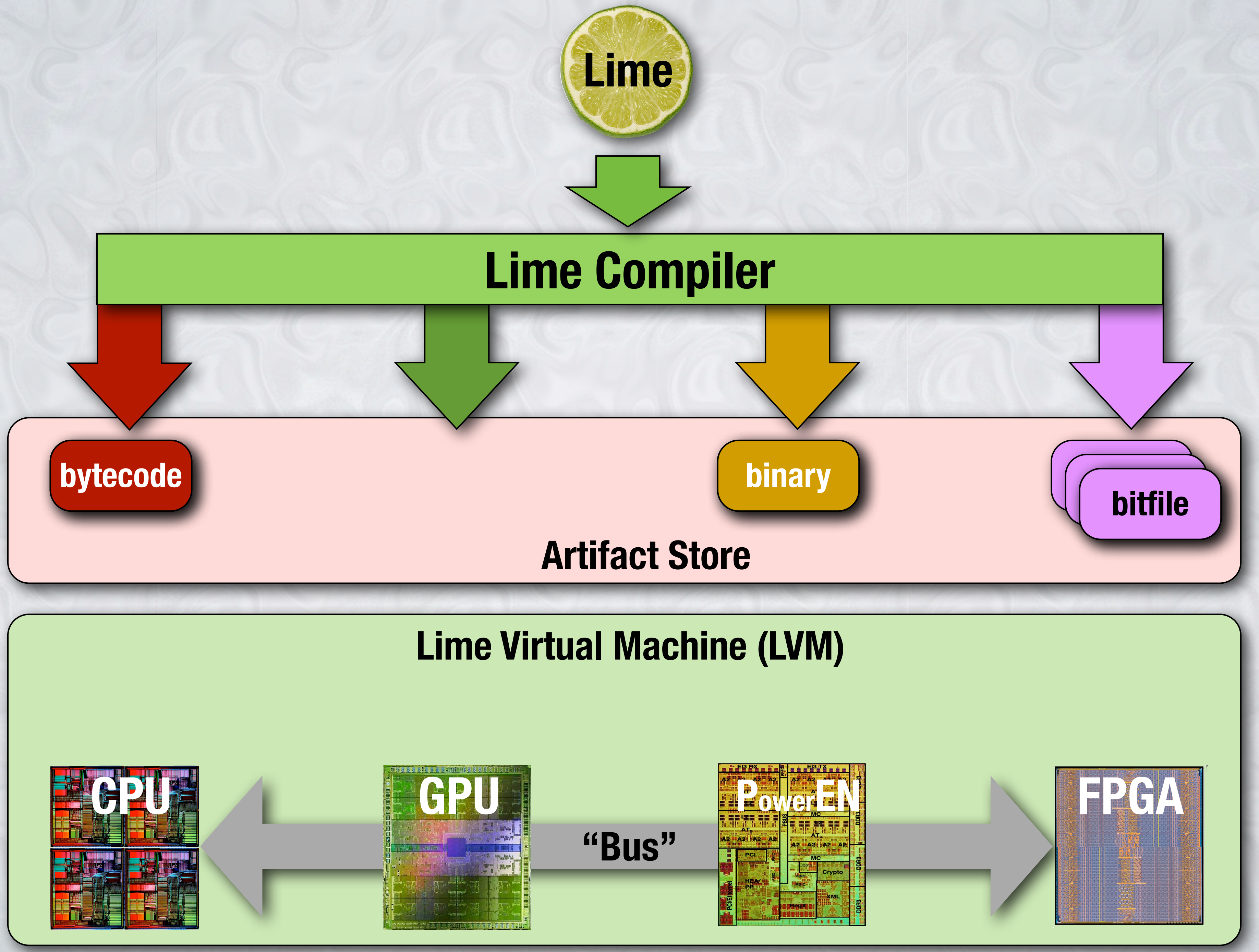
The Liquid Metal Programming Language



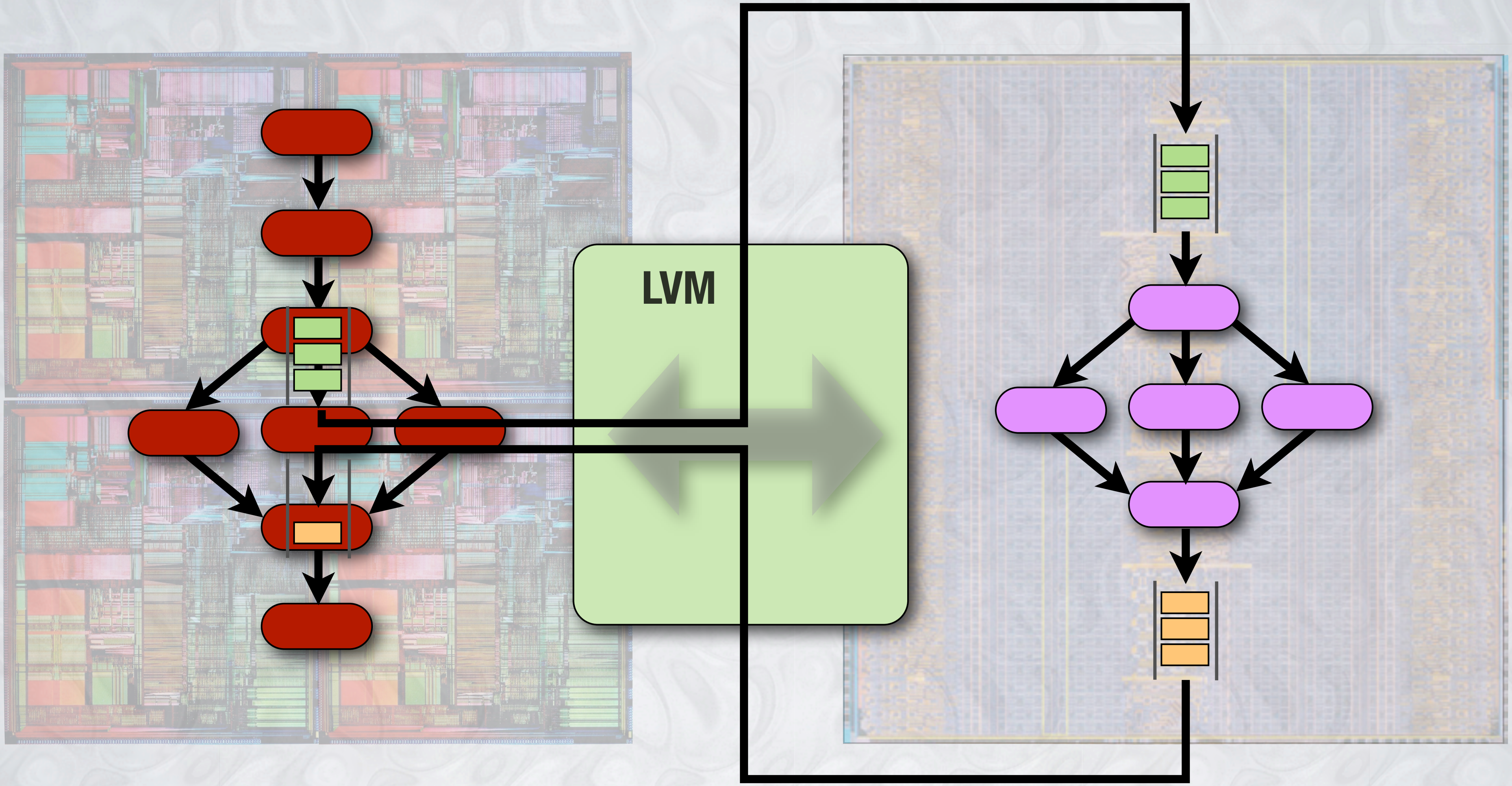
The Artifact Store & Exclusion



HETEROGENEOUS EXECUTION OF LIME



EXECUTION, COMMUNICATION, AND REPLACEMENT



THE LIME LANGUAGE

LIME: JAVA IS (ALMOST) A SUBSET

```
% javac MyClass.java
% java MyClass
```

```
% mv MyClass.java MyClass.lime
% limec MyClass.lime
% java MyClass
```

INCREMENTALLY USE LIME FEATURES

LIME LANGUAGE OVERVIEW

Core Features

Programmable Primitives
Map & Reduce Operations
Stream Programming

BIT-LEVEL PARALLELISM

PIPELINE PARALLELISM

Immutable Types
Bounded Types
Bounded Arrays
Primitive Supertypes

DATA PARALLELISM

Graph Construction
Isolation Enforcement
Closed World Support
Rate Matching
Messaging

Supporting Features

Reifiable Generics
Ranges, Bounded "for"
User-defined operators

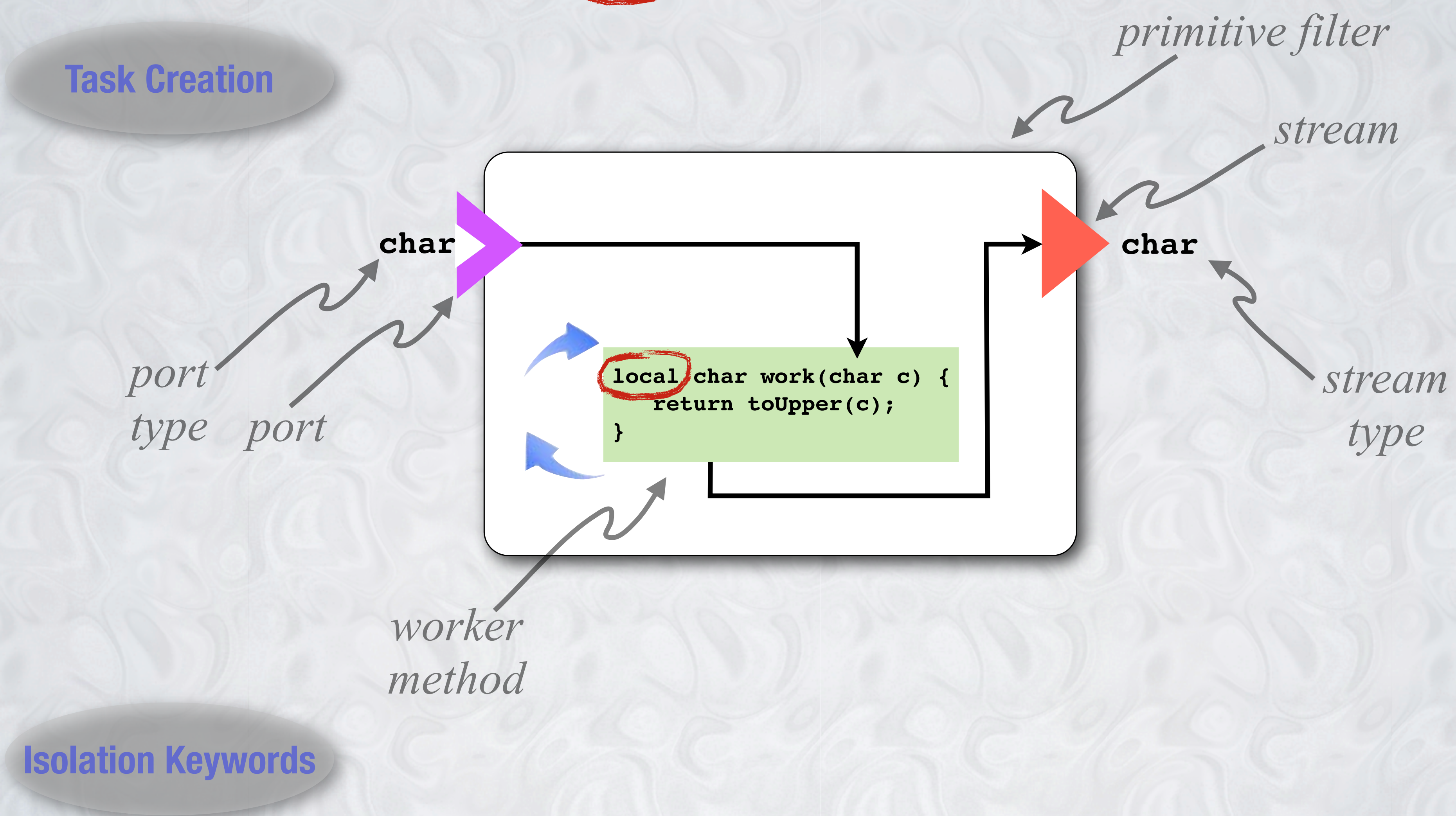
Typedefs
Local type inference
Tuples

STREAMING COMPUTATION

PIPELINE PARALLELISM

TASK CREATION (STATELESS)

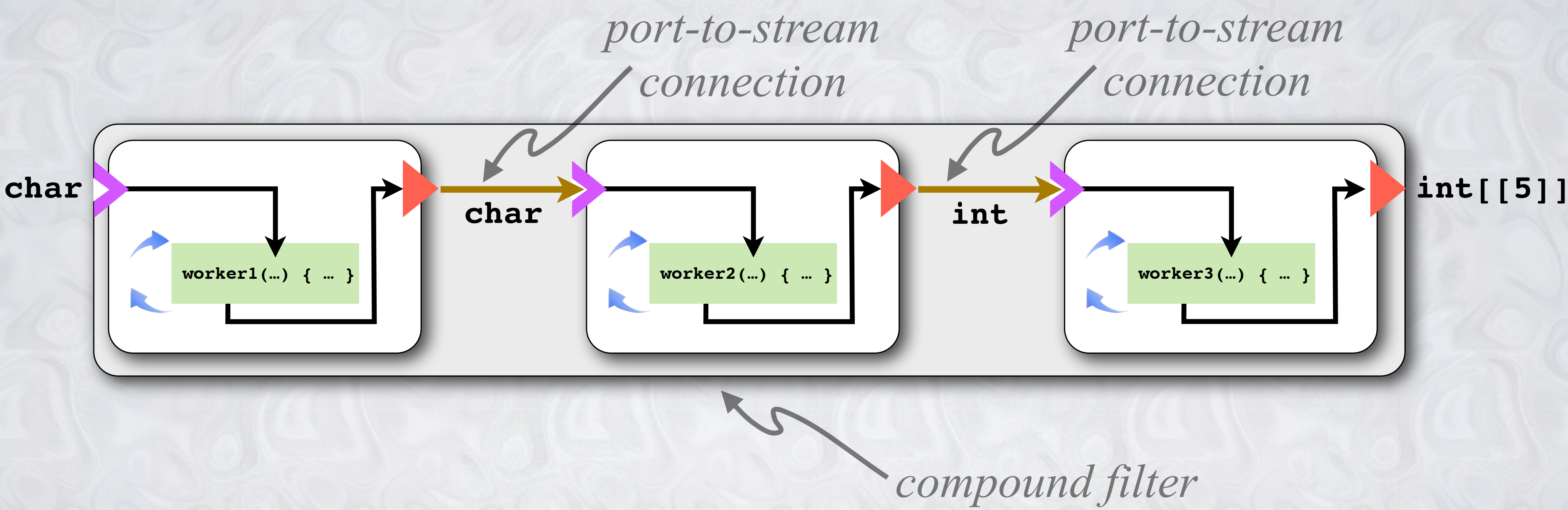
var uppercaser = task work;



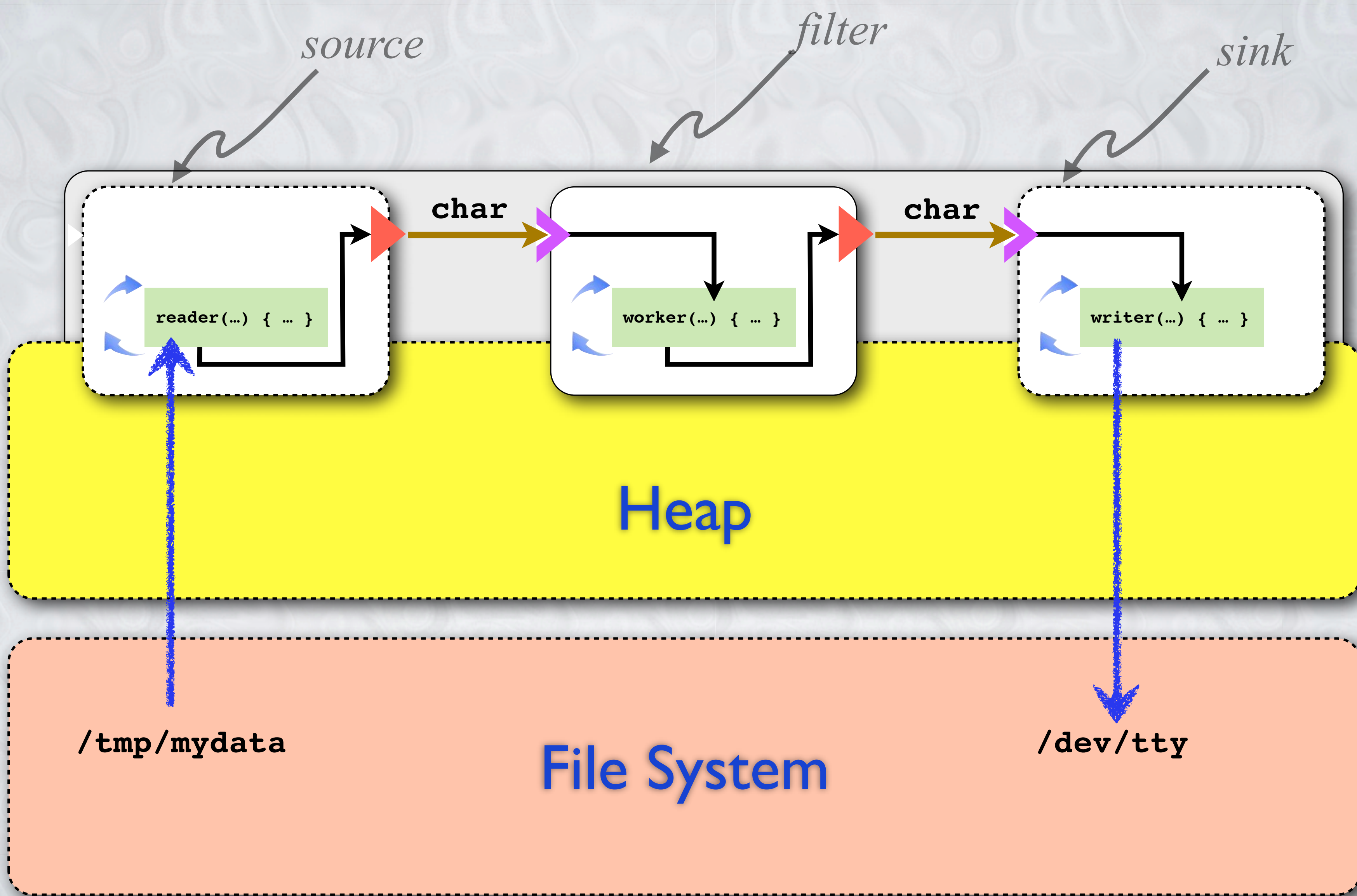
PIPELINES

Graph Construction

var pipeline = task worker1 => task worker2 => task worker3;



SOURCES AND SINKS



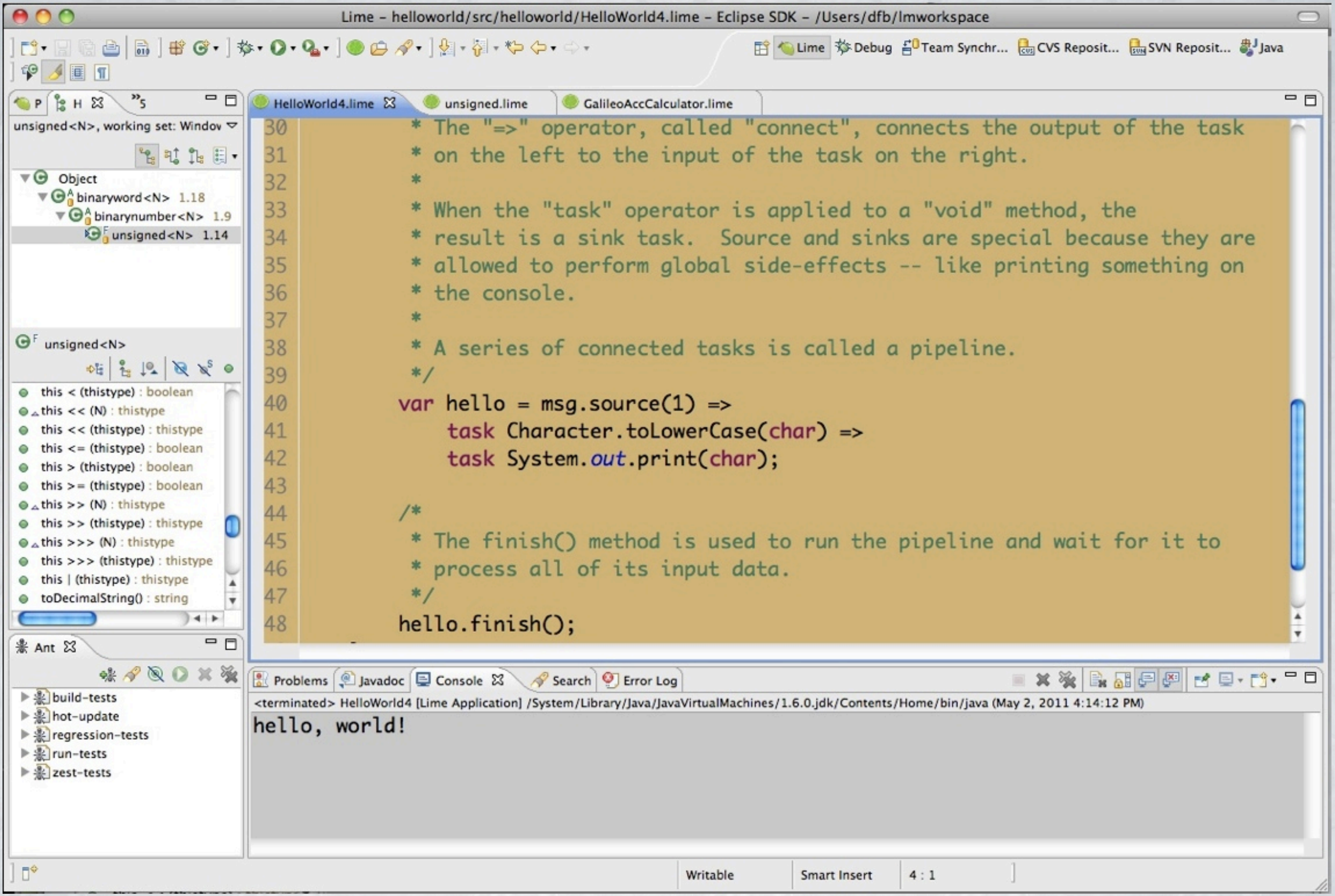
Hello World, Streaming Style

```
public static void main(string[[]] args) {

  char[[]] msg = { 'H','E','L','L','O',',',' ',
                   'W','O','R','L','D','!','\n' };

  var hello = msg.source(1) =>
              task Character.toLowerCase(char) =>
              task System.out.print(char);

  hello.finish();
}
```

DEMO

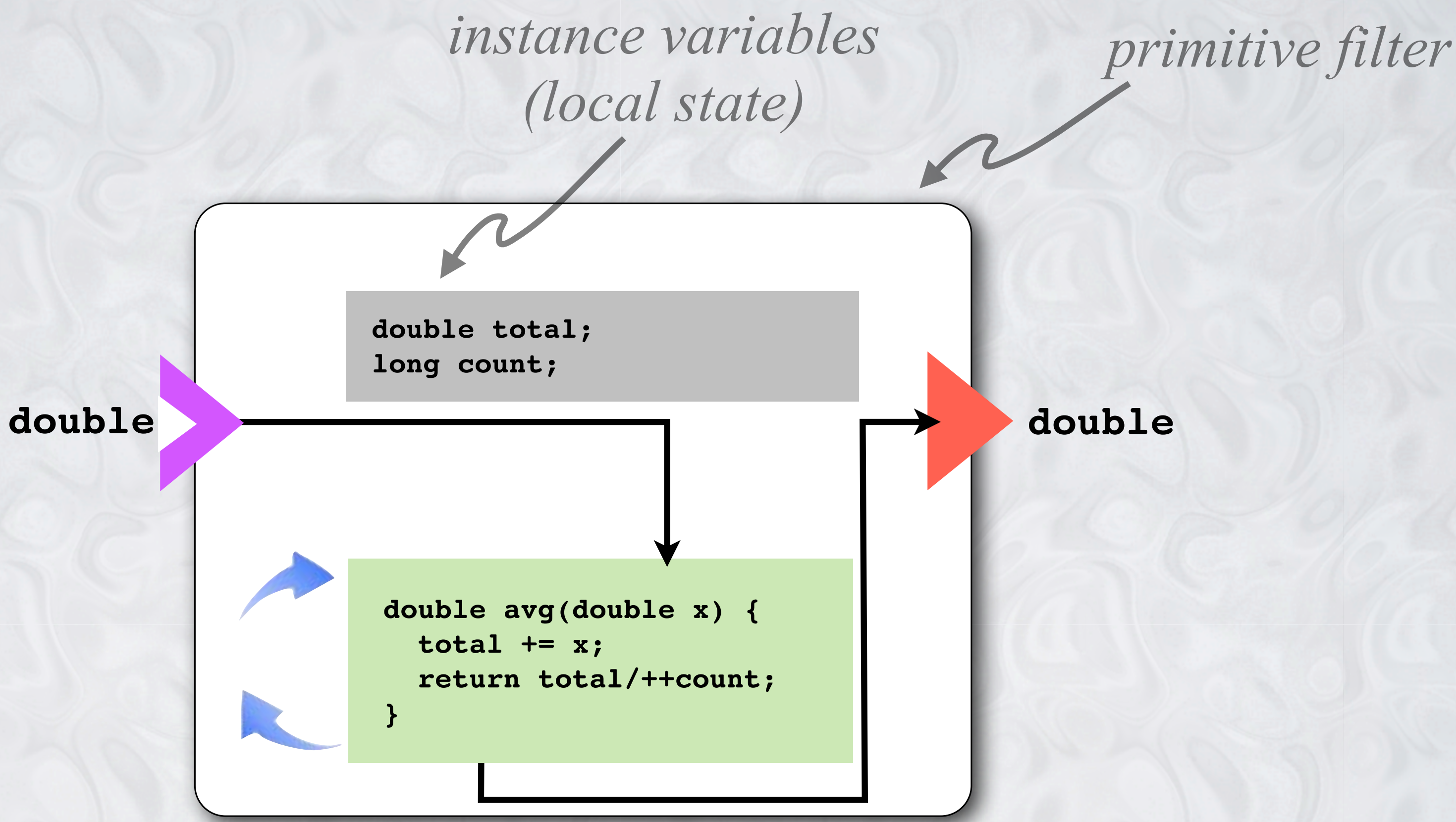
HELLO WORLD

LIME/ECLIPSE ENVIRONMENT


Lime - helloworld/src/helloworld/HelloWorld4.lime - Eclipse SDK - /Users/dfb/lmworkspace
Lime
Debug
Team Synchr...
CVS Reposit...
SVN Reposit...
Java
HelloWorld4.lime
unsigned.lime
GalileoAccCalculator.lime
* The "=>" operator, called "connect", connects the output of the task
* on the left to the input of the task on the right.
*
* When the "task" operator is applied to a "void" method, the
* result is a sink task. Source and sinks are special because they are
* allowed to perform global side-effects -- like printing something on
* the console.
*
* A series of connected tasks is called a pipeline.
*/
var hello = msg.source(1) =>
task Character.toLowerCase(char) =>
task System.out.print(char);
/*
* The finish() method is used to run the pipeline and wait for it to
* process all of its input data.
*/
hello.finish();
unsigned<N>, working set: Window
Object
binaryword<N> 1.18
binarynumber<N> 1.9
unsigned<N> 1.14
unsigned<N>
this < (thistype) : boolean
this << (N) : thistype
this << (thistype) : thistype
this <= (thistype) : boolean
this > (thistype) : boolean
this >= (thistype) : boolean
this >> (N) : thistype
this >> (thistype) : thistype
this >>> (N) : thistype
this >>> (thistype) : thistype
this | (thistype) : thistype
toDecimalString() : string
Ant
build-tests
hot-update
regression-tests
run-tests
zest-tests
Problems
Javadoc
Console
Search
Error Log
<terminated> HelloWorld4 [Lime Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (May 2, 2011 4:14:12 PM)
hello, world!
Writable
Smart Insert
4 : 1

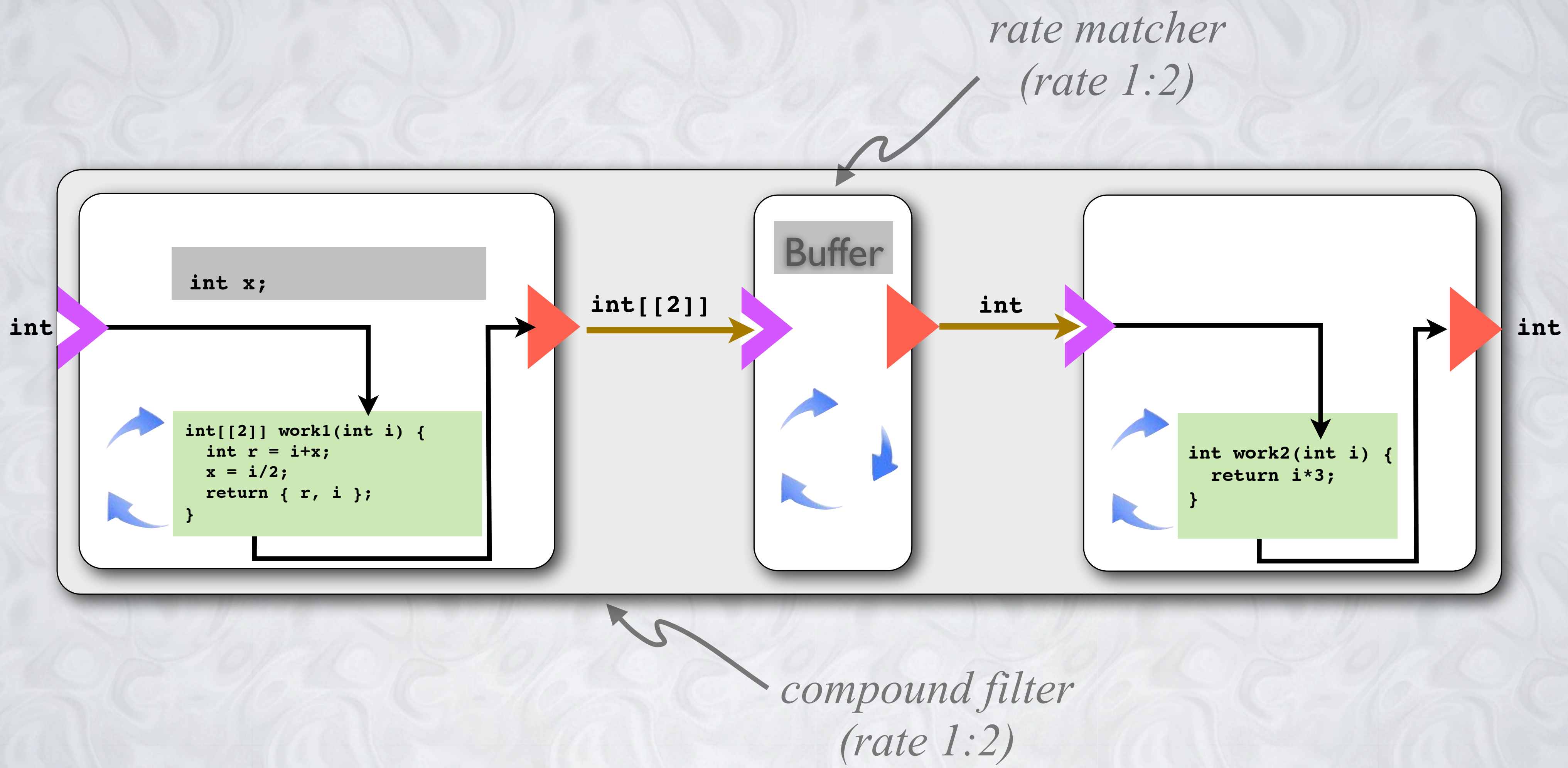
STATEFUL TASKS

```
var averager = task Averager().avg;
```



Rate Matching

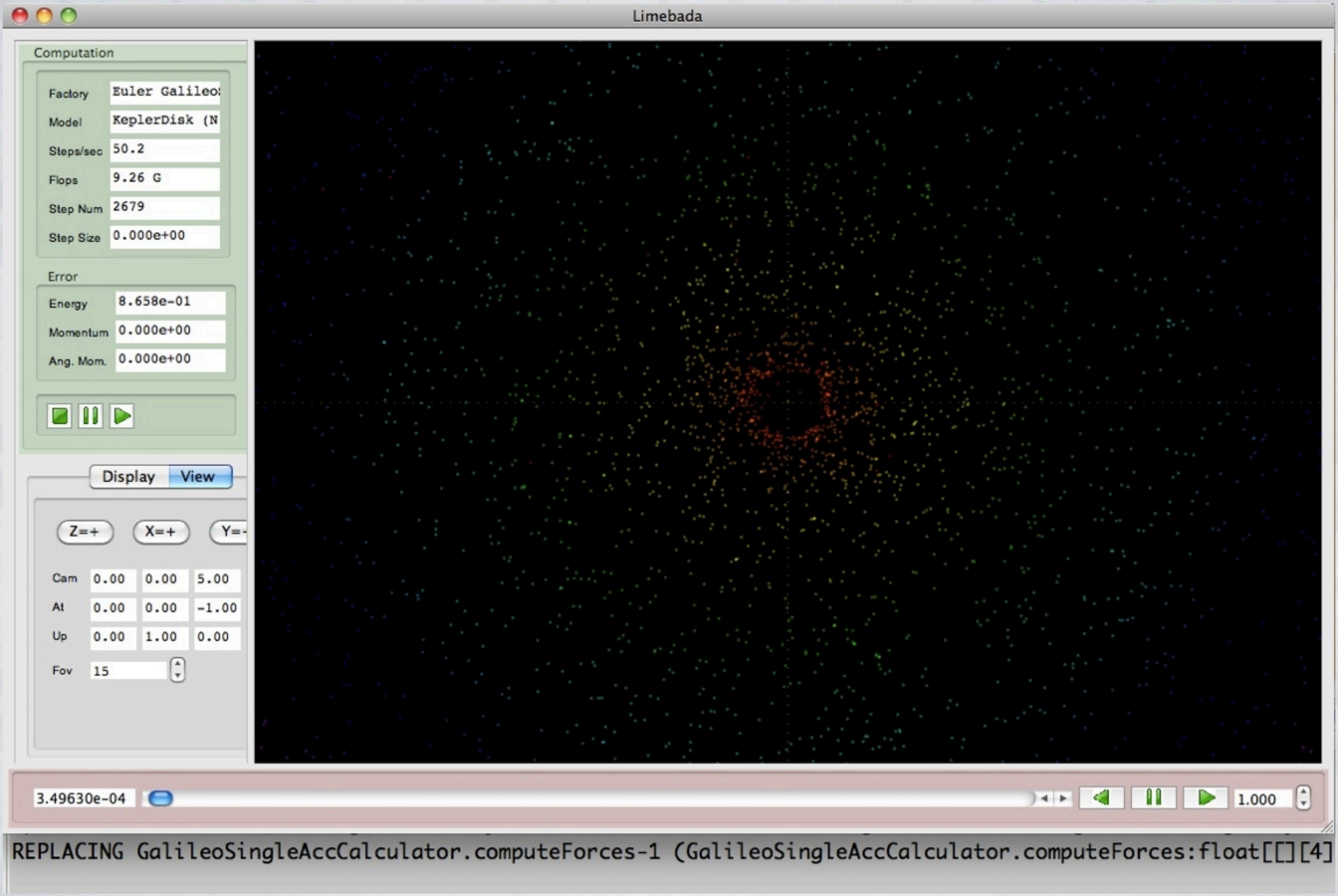
var matchedpipe = task AddStuff().work1 => # => task work2;



DEMO

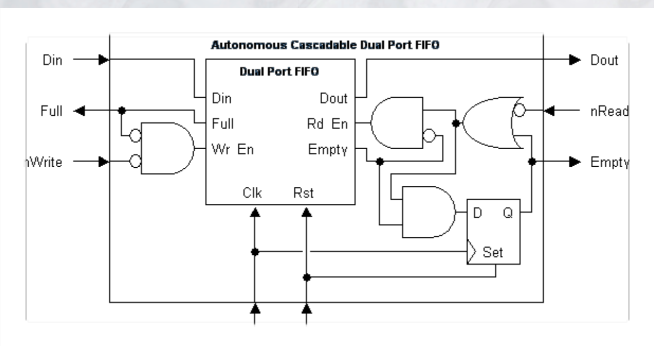
N-BODY SIMULATION: CPU VS. GPU

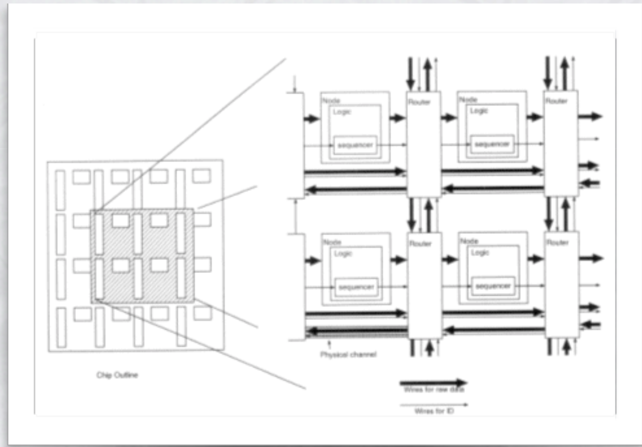
9x Speedup (9.26 Gflops) on Laptop

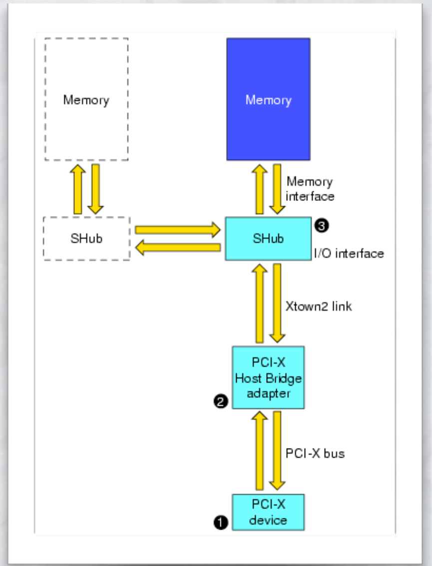


VIRTUALIZATION OF DATA MOVEMENT

=>







MAP & REDUCE OPERATIONS

DATA PARALLELISM

ARRAY PARALLELISM

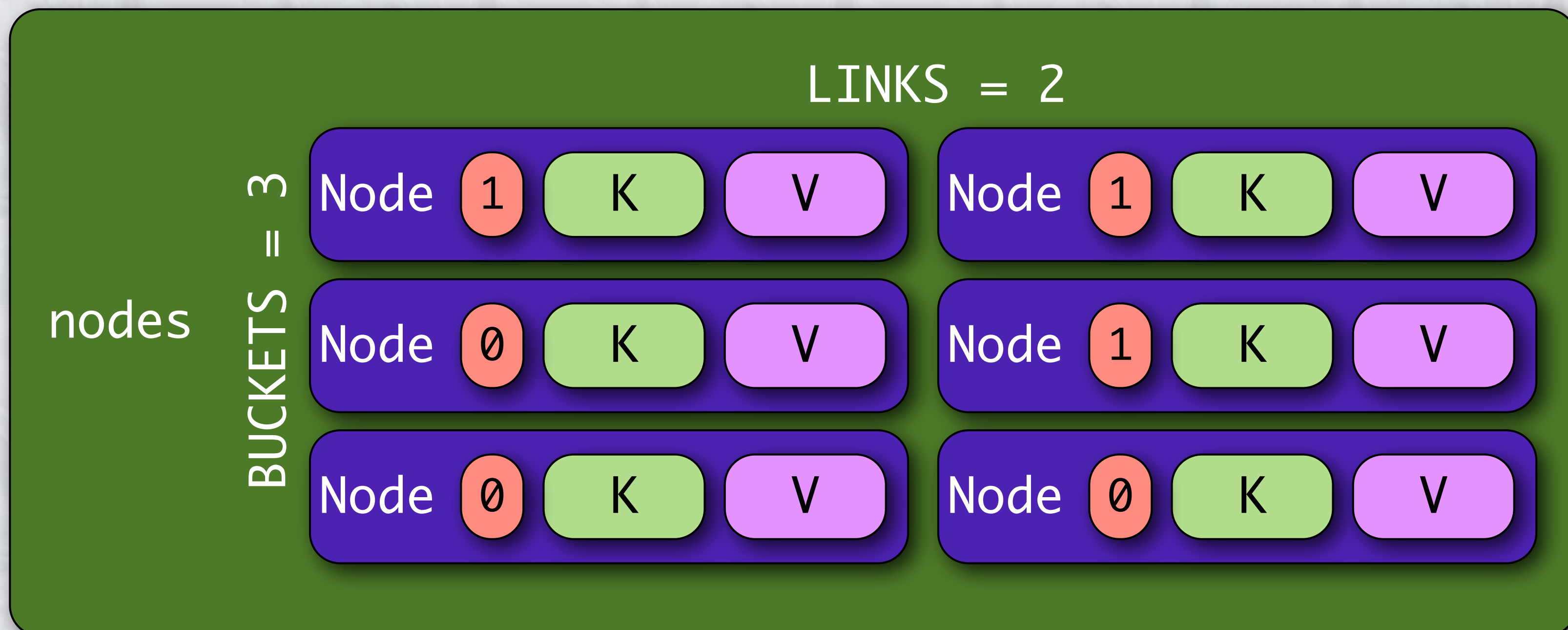
```
float[ ] a = ...;
float[ ] b = ...;

float[ ] c = a @+ b;

float sum = + ! c;
```

Indexable<int,float>

Collectable<int,float>

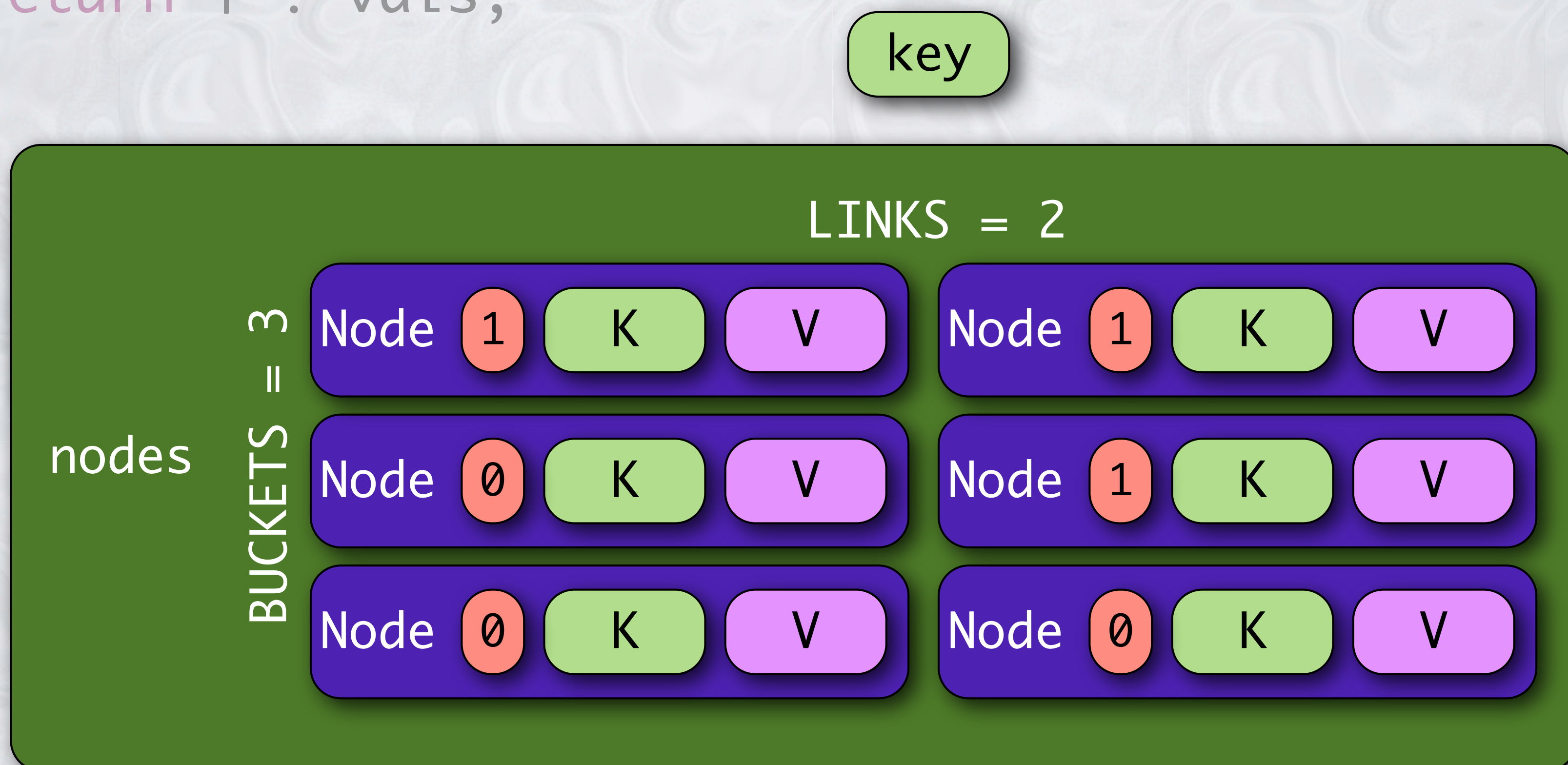
Map & Reduce Operations in Action

```
package lime.util.synthesizable;

public class FixedHashMap<K extends Value, V extends Value,
     BUCKETS extends ordinal<BUCKETS>, LINKS extends ordinal<LINKS>>
  extends AbstractMap<K,V>
{
  protected final nodes = new Node<K,V>[BUCKETS][LINKS];
```



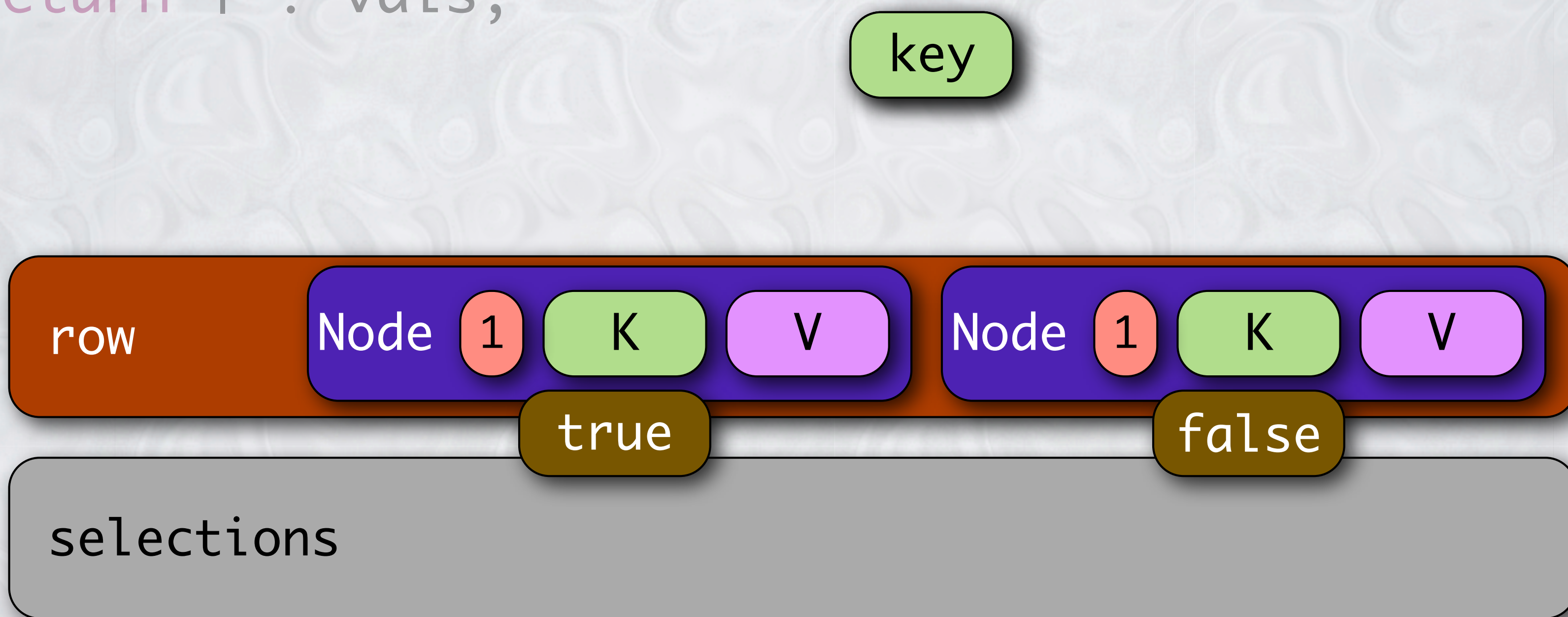
Get Operation, Step 1: Select Row

```
public local V get(K key) {
   Node[LINKS] row = nodes[hash(key)];
   boolean[LINKS] selections = row @ compareKey(key);
   V[LINKS] vals = row @ getValueOrDefault(selections);
   return | ! vals;
}
```



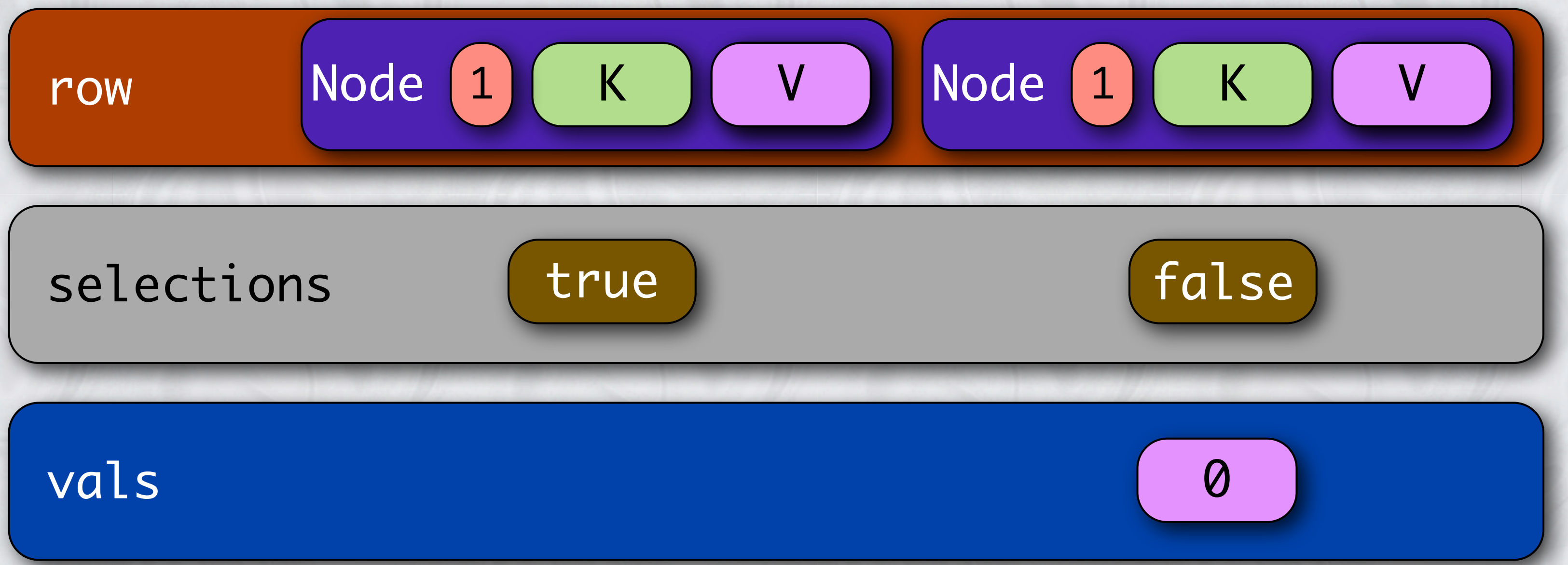
Step 2: Compare All Keys

```
public local V get(K key) {
   Node[LINKS] row = nodes[hash(key)];
   boolean[LINKS] selections = row @ compareKey(key);
   V[LINKS] vals = row @ getValueOrDefault(selections);
   return | ! vals;
}
```



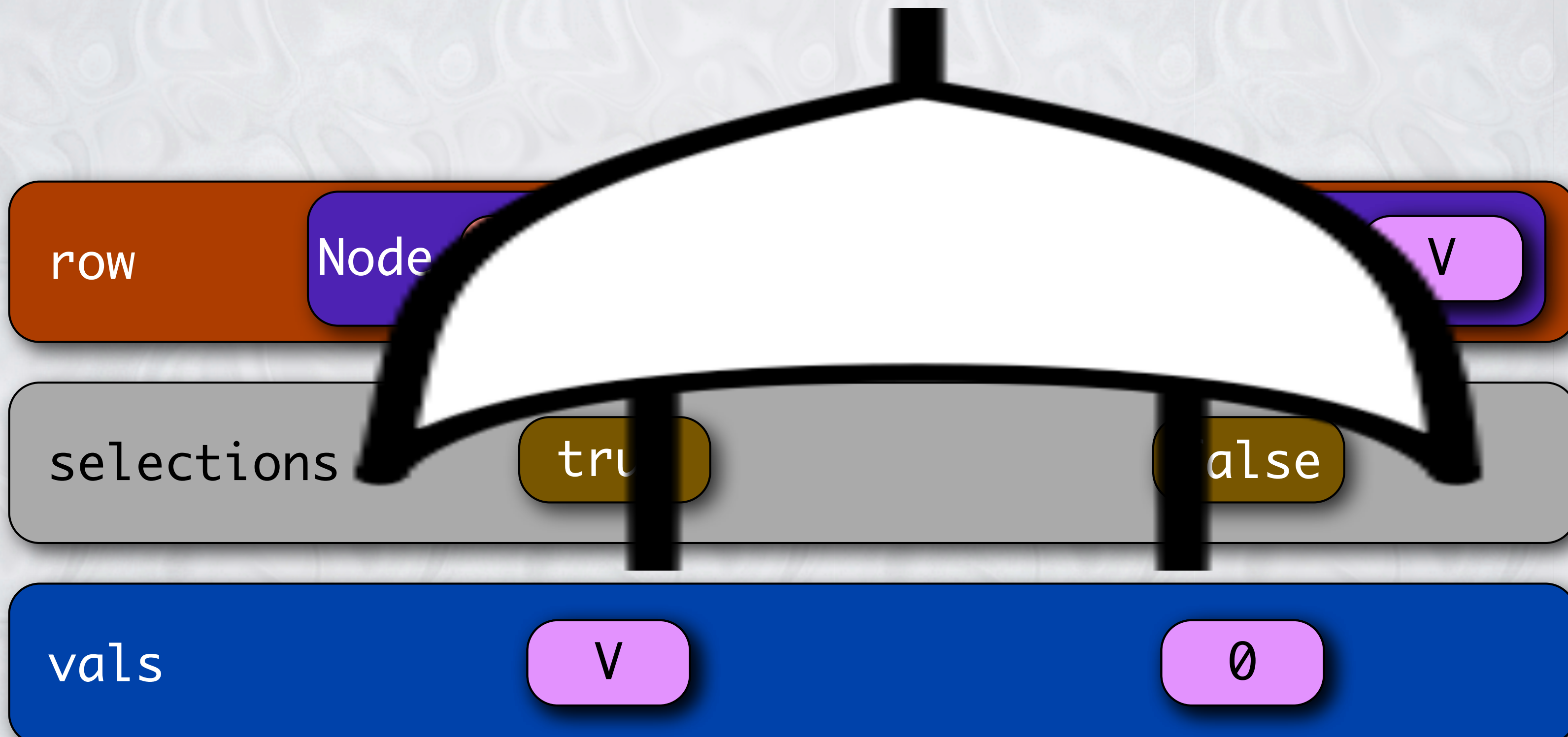
Step 3: Get Values/Zeros

```
public local V get(K key) {
   Node[LINKS] row = nodes[hash(key)];
   boolean[LINKS] selections = row @ compareKey(key);
   V[LINKS] vals = row @ getValueOrDefault(selections);
   return | ! vals;
}
```



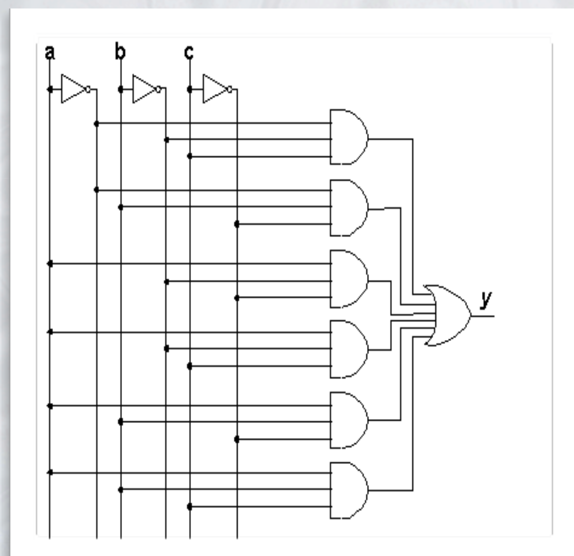
Step 4: Or-Reduce For Result

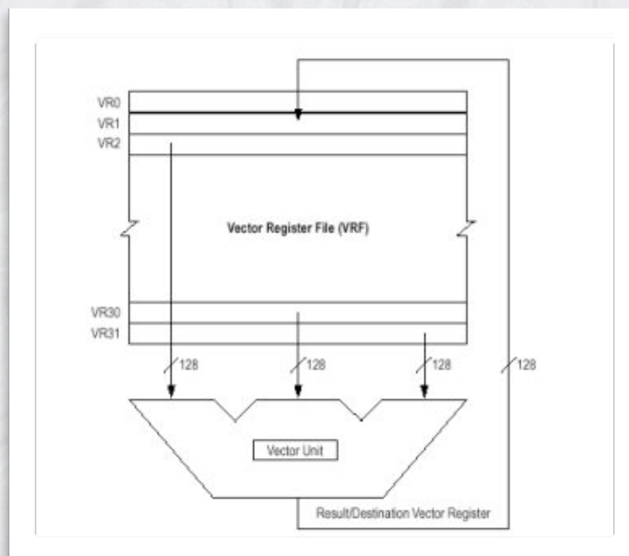
```
public local V get(K key) {
   Node[LINKS] row = nodes[hash(key)];
   boolean[LINKS] selections = row @ compareKey(key);
   V[LINKS] vals = row @ getValueOrDefault(selections);
   return | ! vals;
}
```

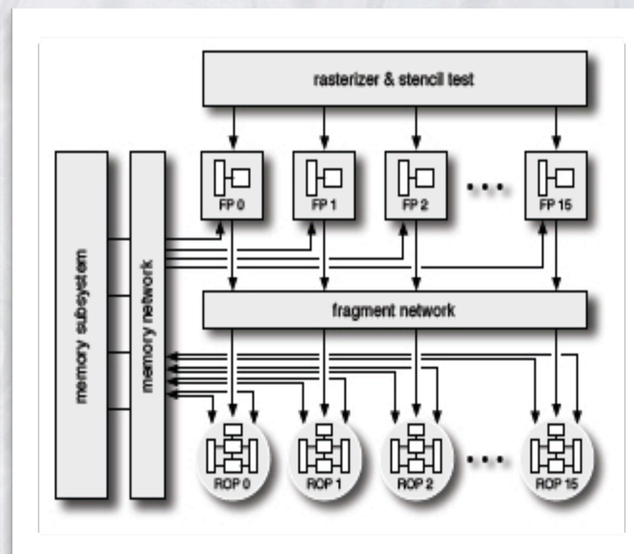


Virtualization of Data Parallelism

@ !







CURRENT RESULTS

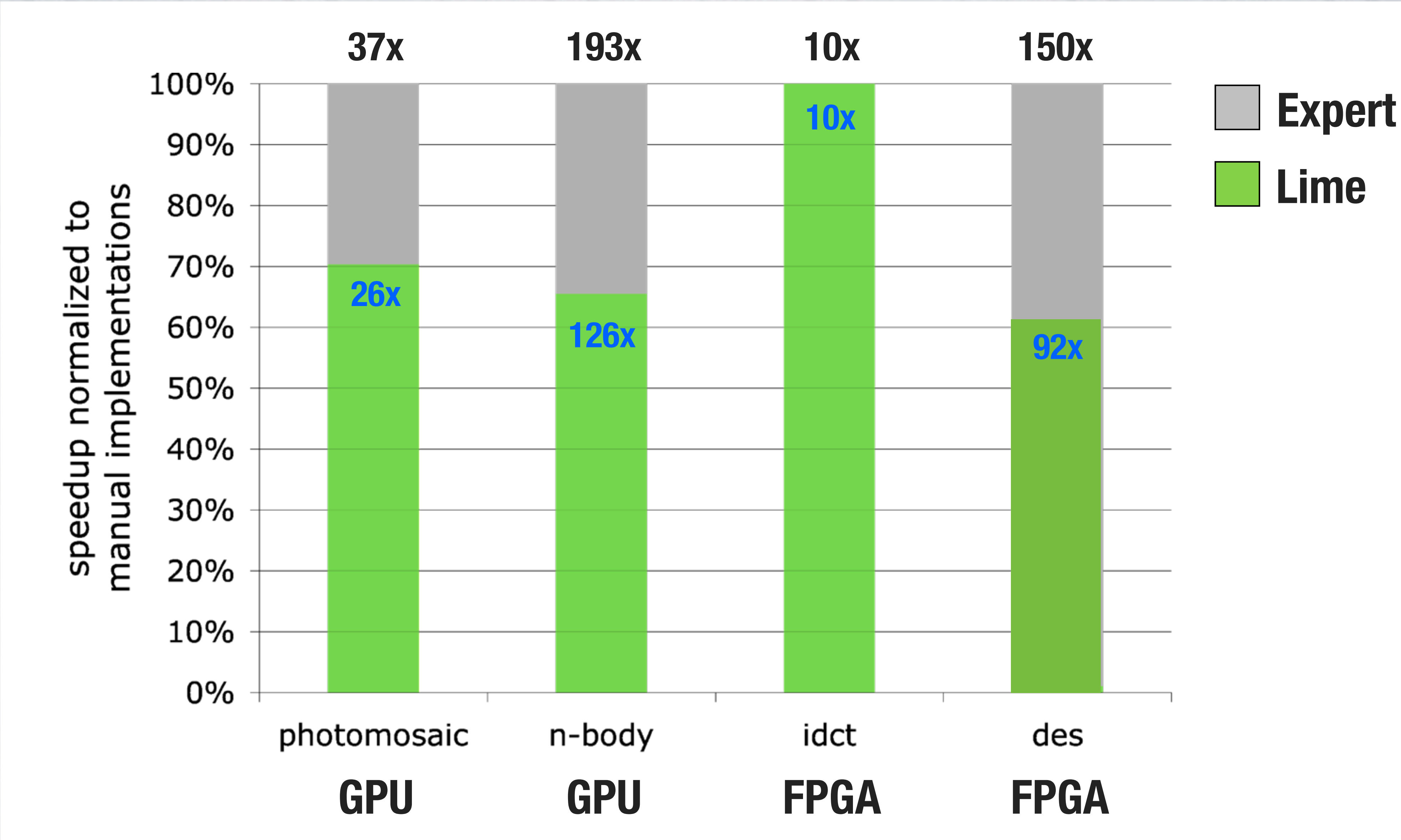
How Do we Evaluate Performance?

- Speedup for Naïve Users
  - How much faster than Java?
- Slowdown for Expert Users?
  - How much slower than hand-tuned low-level code?

- Our methodology:
  - Write/tune/compare 4 versions of each benchmark:
    - Java, Lime, OpenCL, Verilog
  - Doesn’t address flops/watt, flops/watt/$, productivity

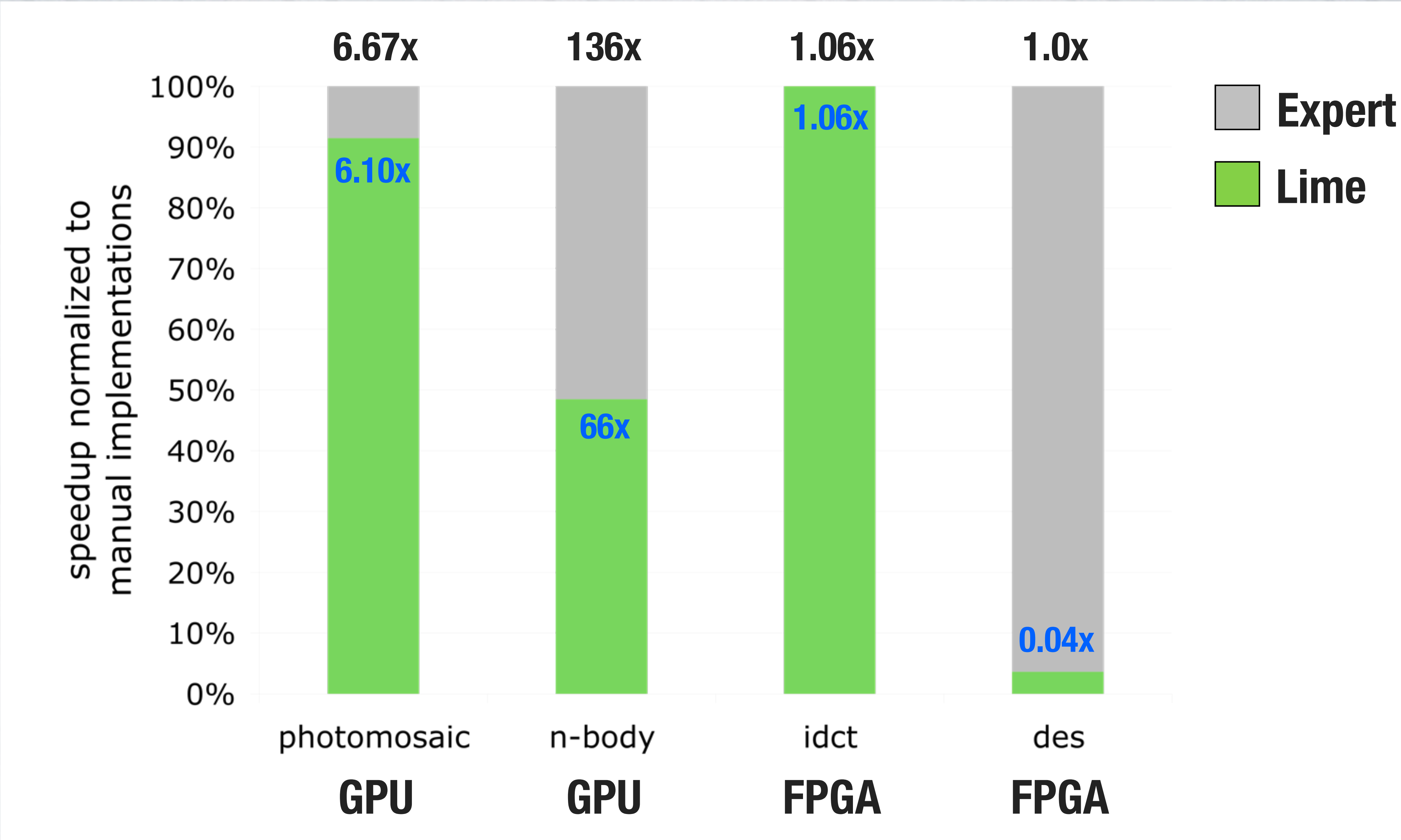
Expert vs Naïve Speedup: Kernel Time

(Java Baseline)



Expert vs Naïve Speedup: End-to-End

(Java Baseline)



Liquid Metal: Summary

- Can we program HW with an object-oriented language?
  - Yes we can!
  - Steadily increasing feature set (e.g. dynamic allocation/GC)
- Many hurdles remain
  - Quality of code, area, predictability, …
  - FPGA tool flow, culture, and business model (vs. GPU)
- We're hiring! Permanent staff, post-docs, and interns

Questions?