Liquid Metal:
Taming Architecture Heterogeneity

Rodric Rabbah

IBM Research

research.ibm.com/liquidmetal

Berkeley

11/2/12

Liquid Metal Team

Joshua Auerbach

David Bacon

Ioana Burcea

Perry Cheng

Steven Fink

Rodric Rabbah

Sunil Shukla

- Past Members

  Christophe Dubach

  Yu Zhang

- Interns

  Charlie Curtsinger

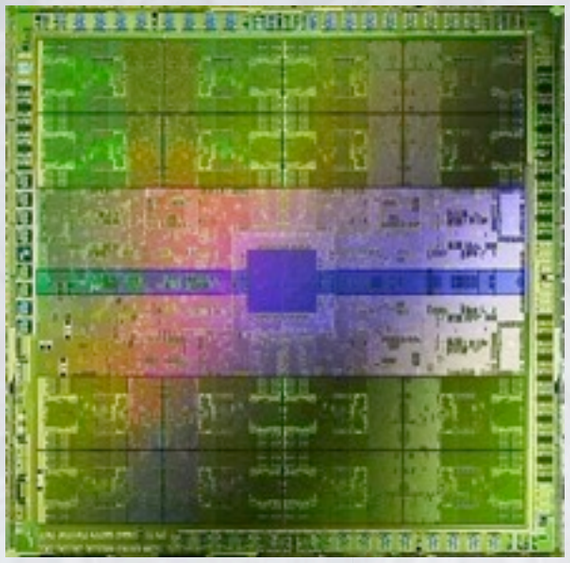
  Andrei Hagiescu

  Amir Hormati

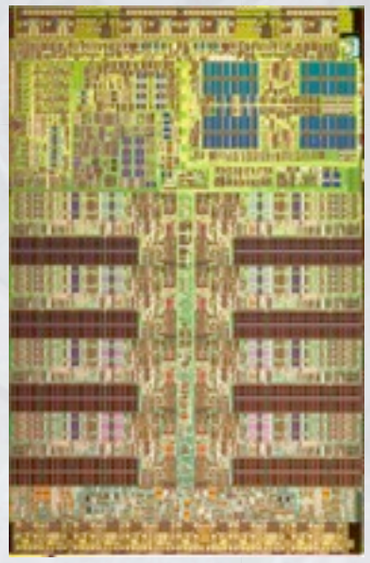
  Shan Shan Huang

  Myron King

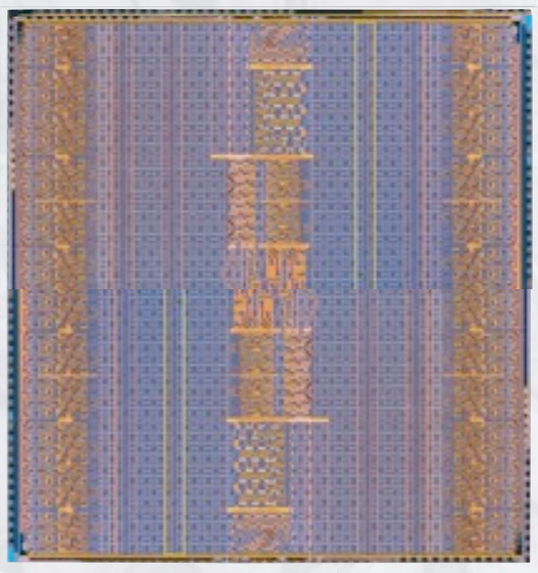
A Heterogeneous Reality



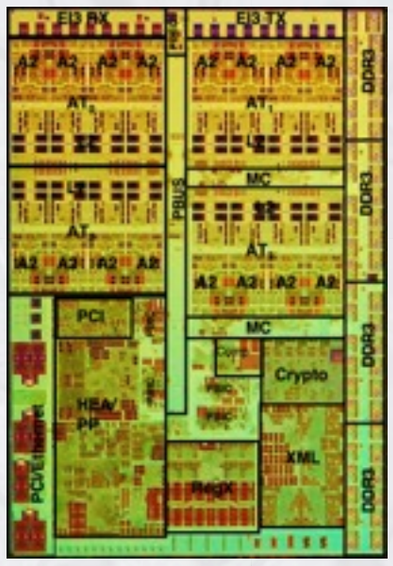
GPU



Multicore

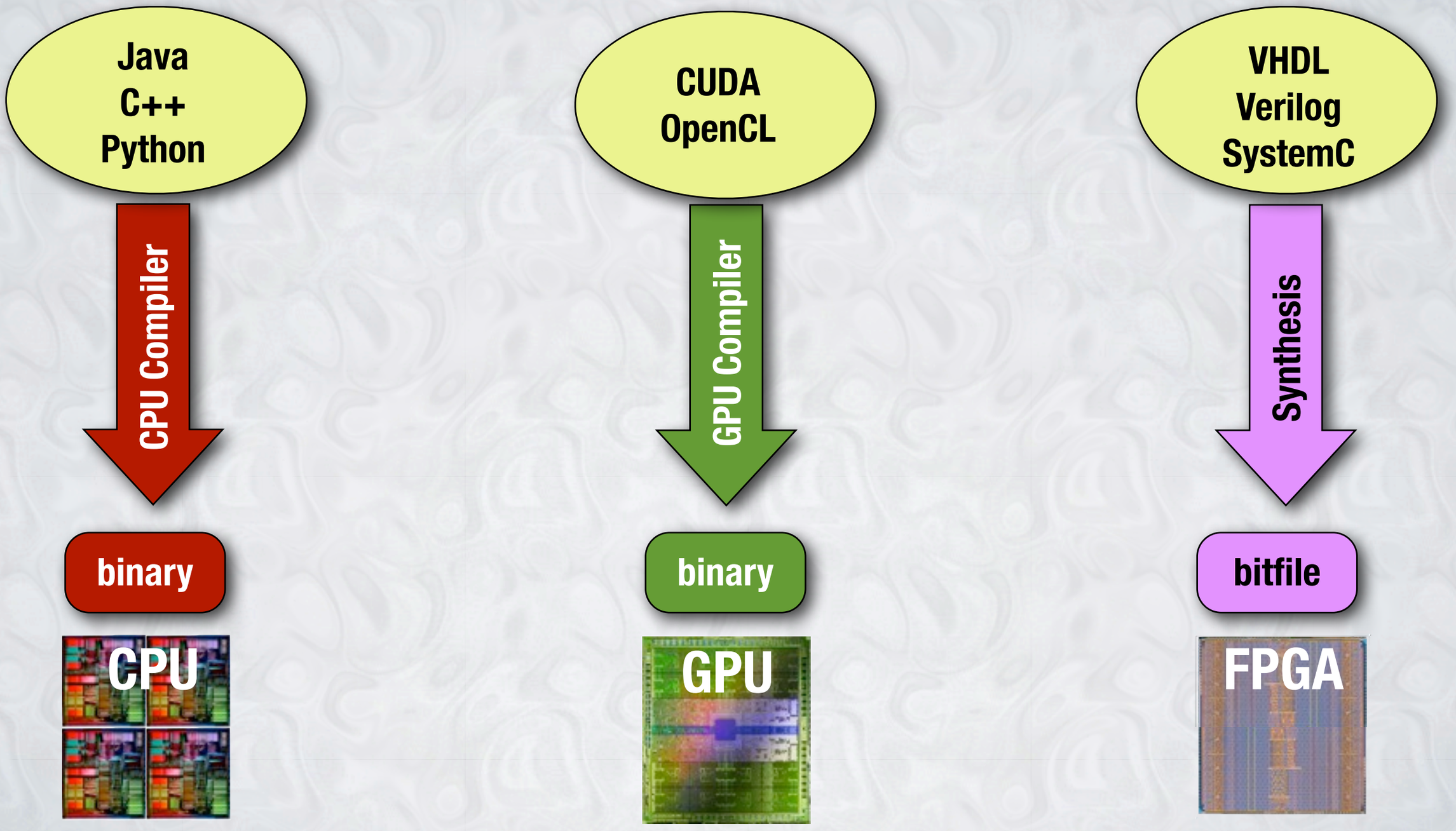


FPGA

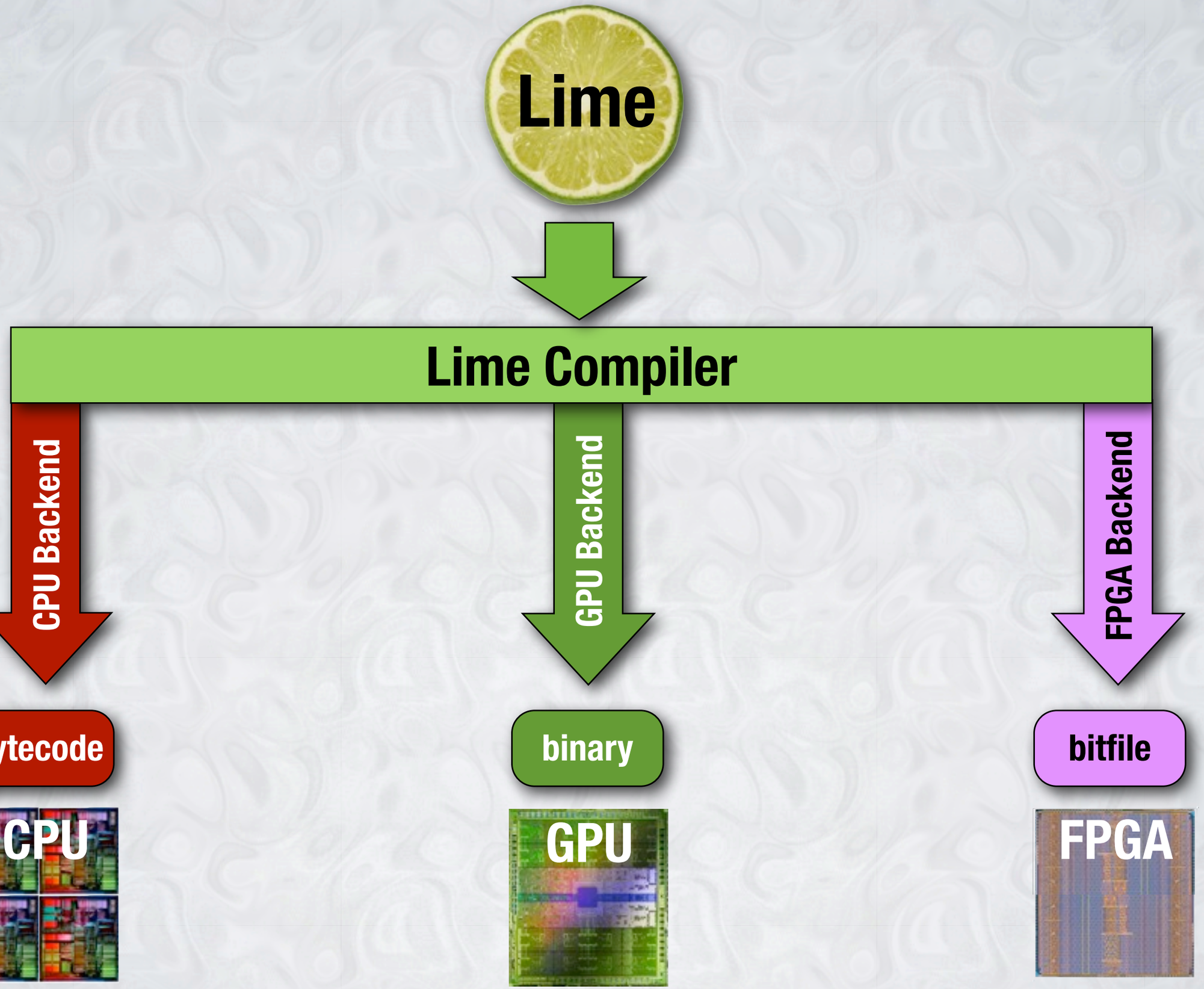


ASIP

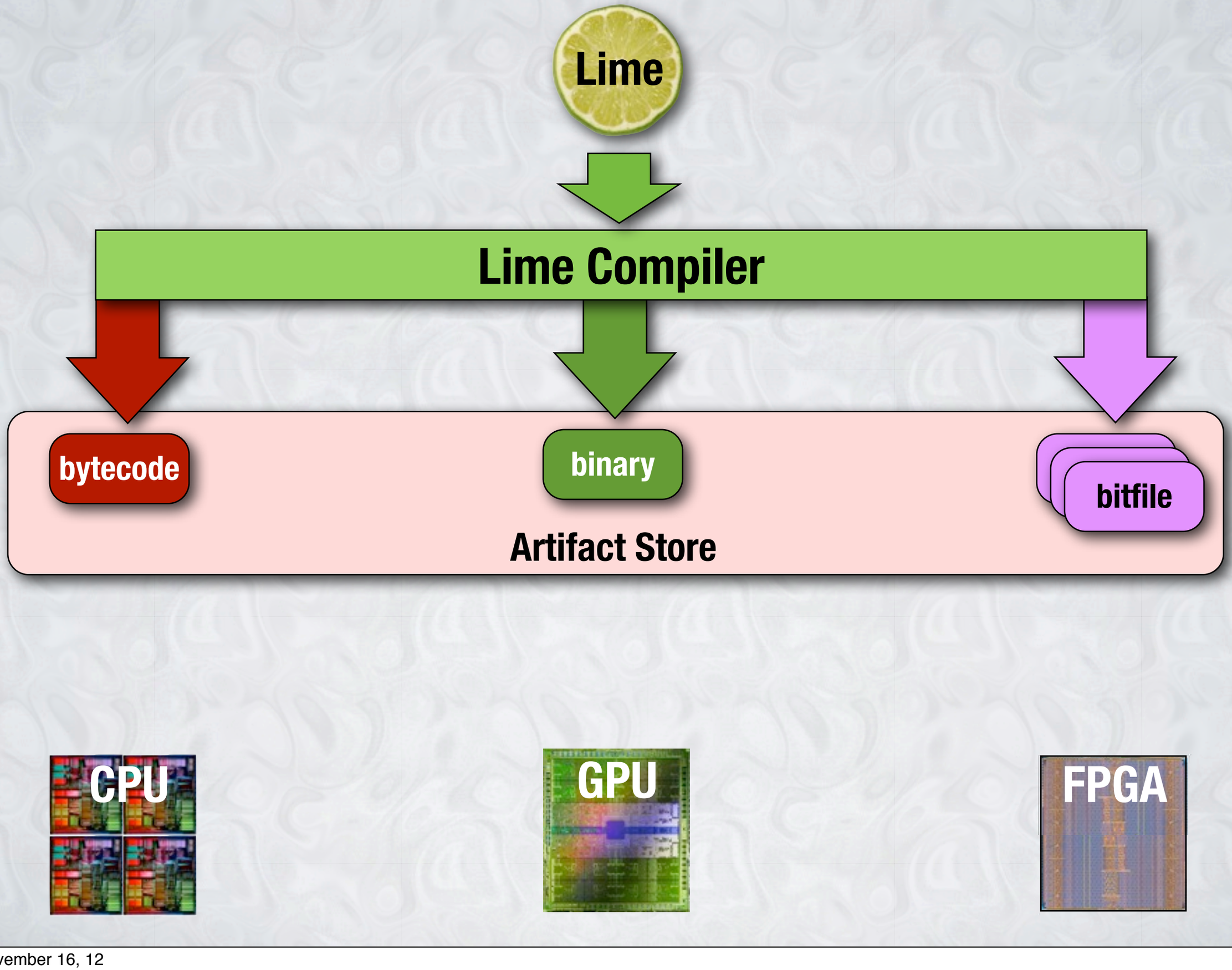
The Real Software Burden: Heterogeneous Programming



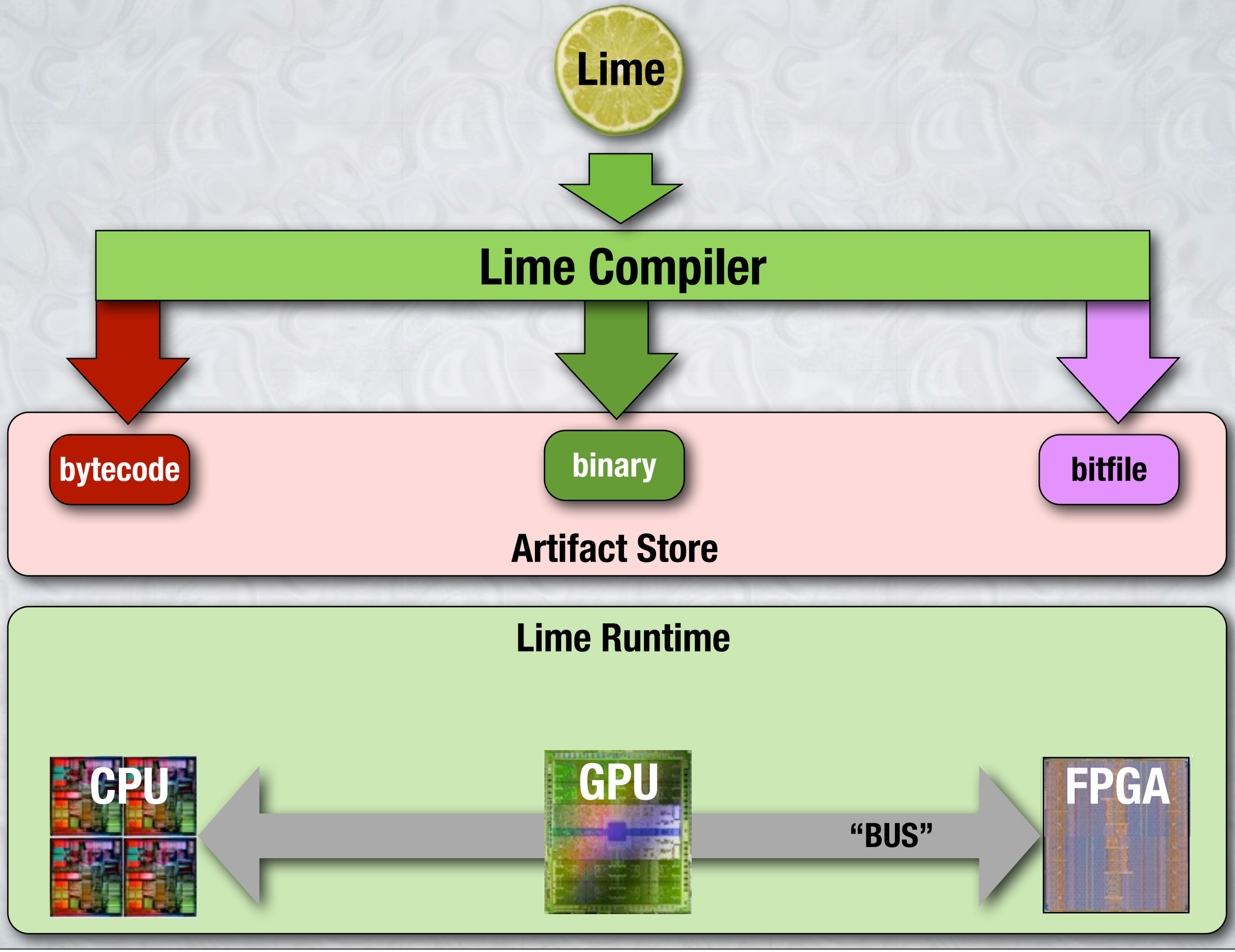
The Liquid Metal Programming Language



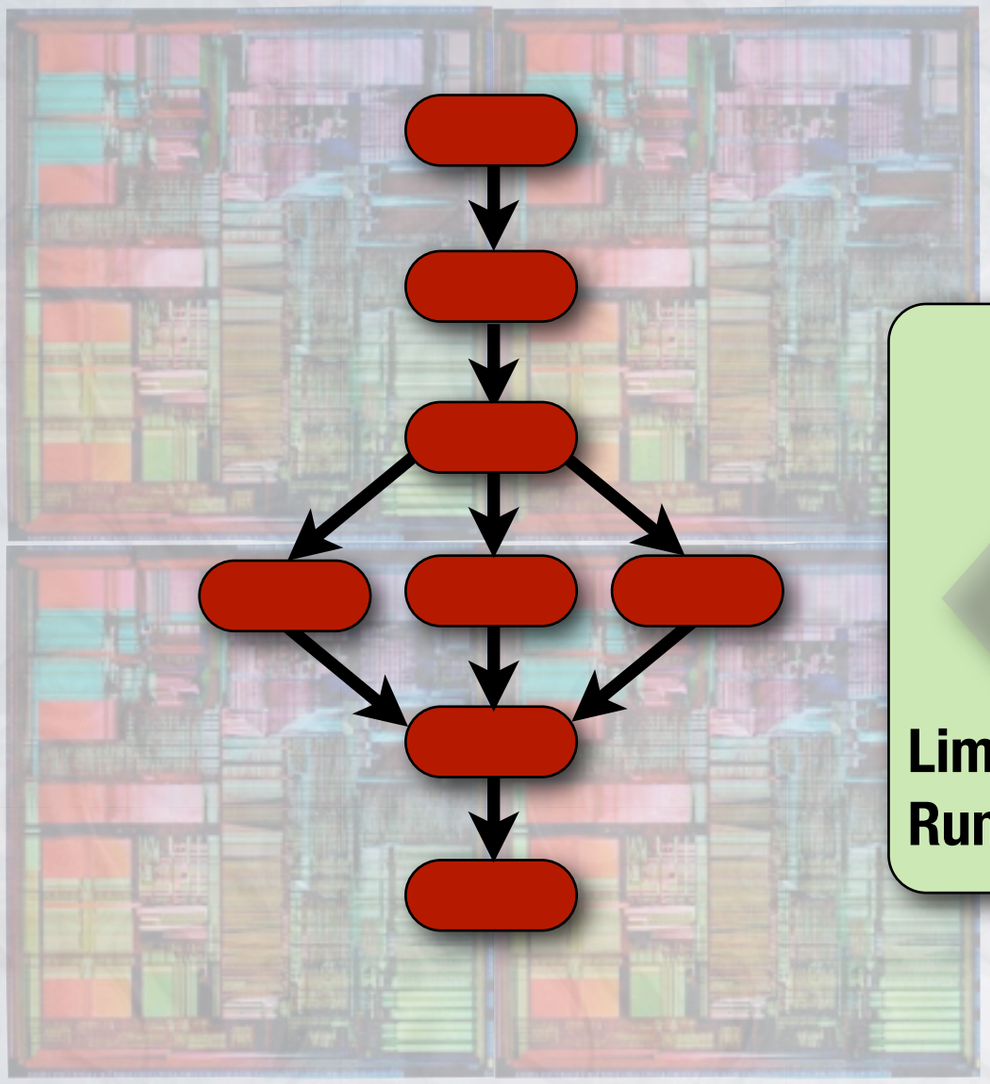
Lime Artifact Store

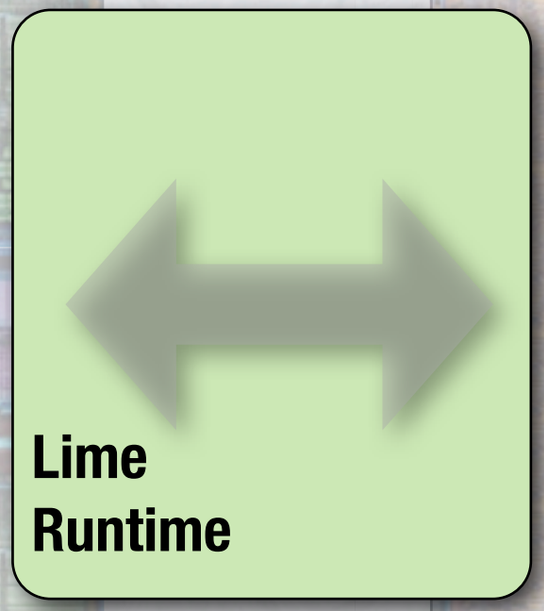


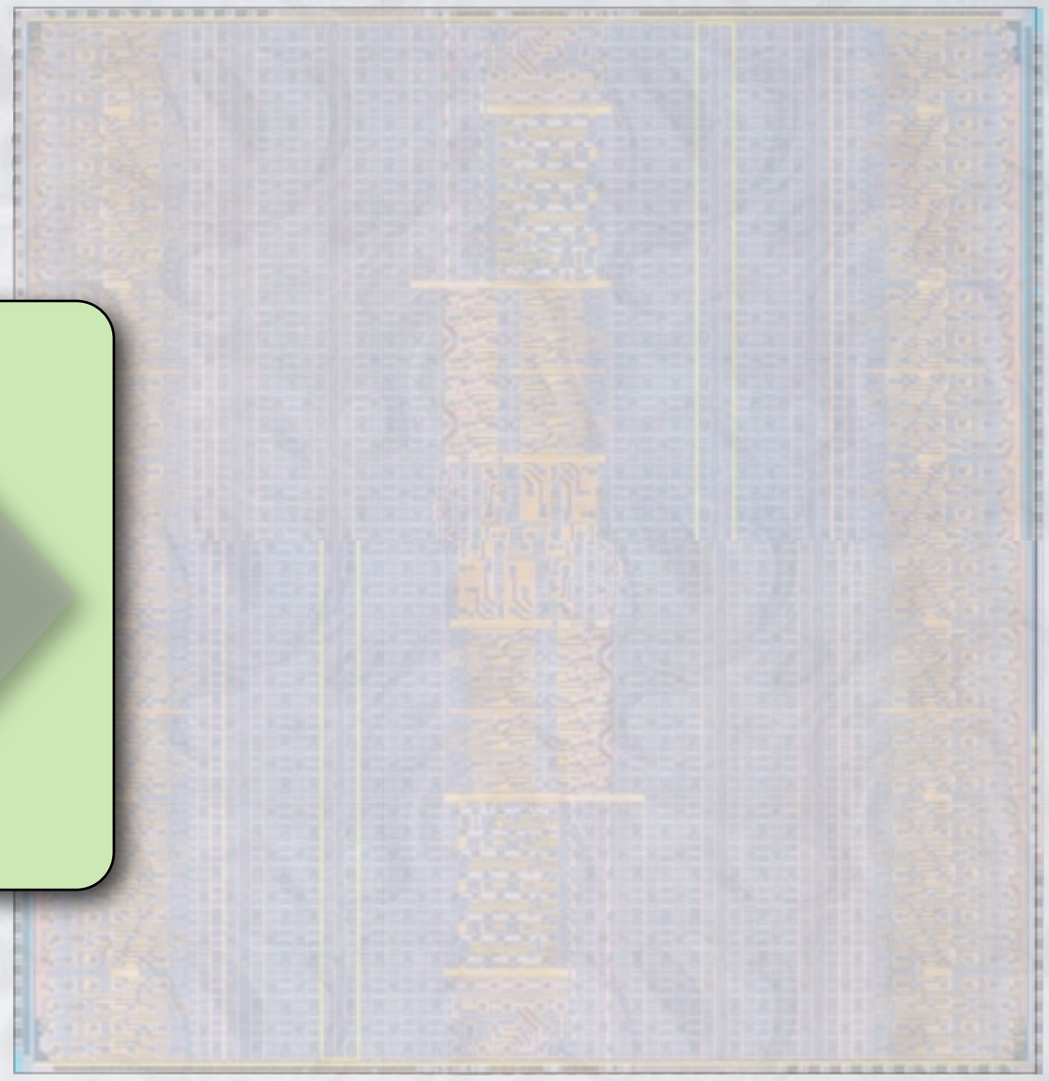
Lime Virtual Machine



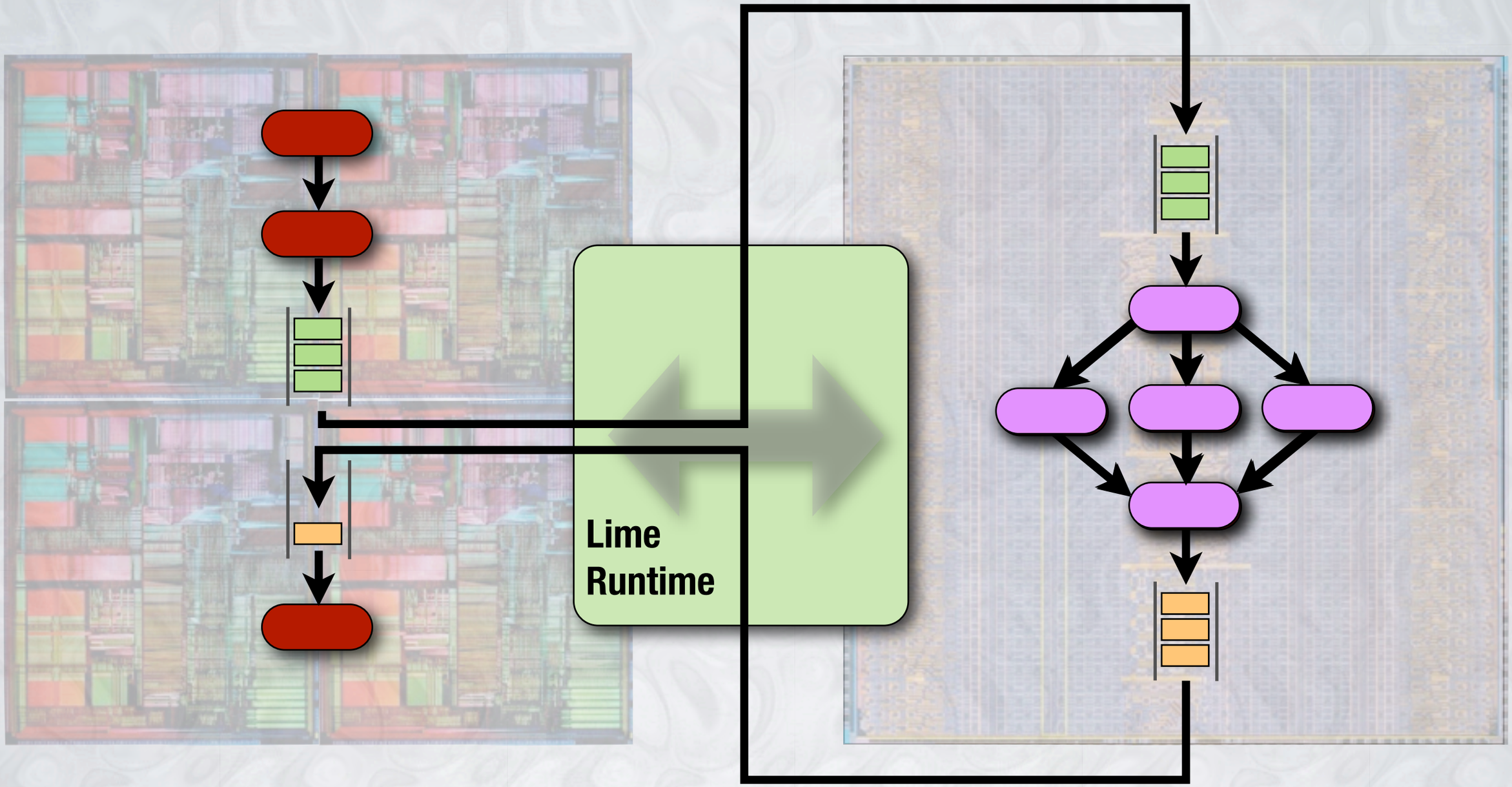
Co-Execution and Migration



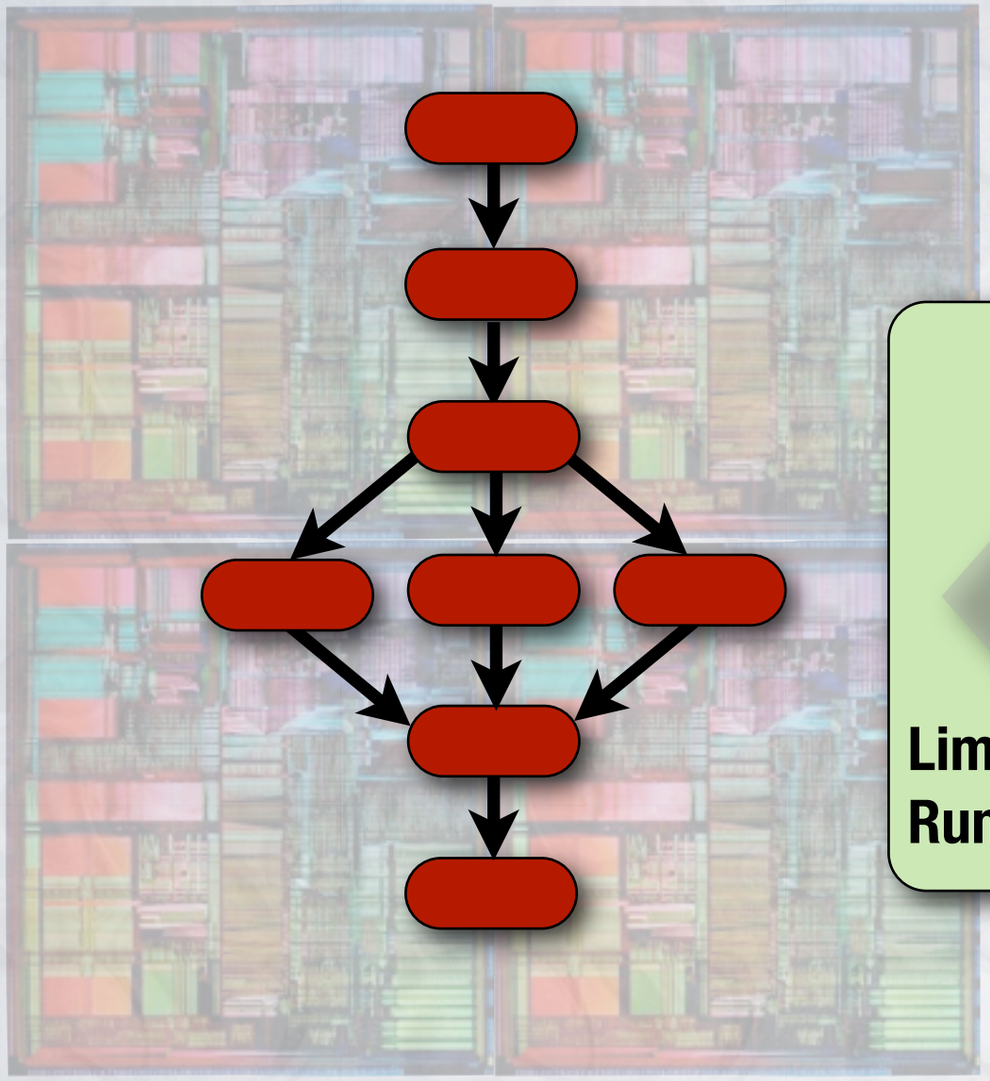


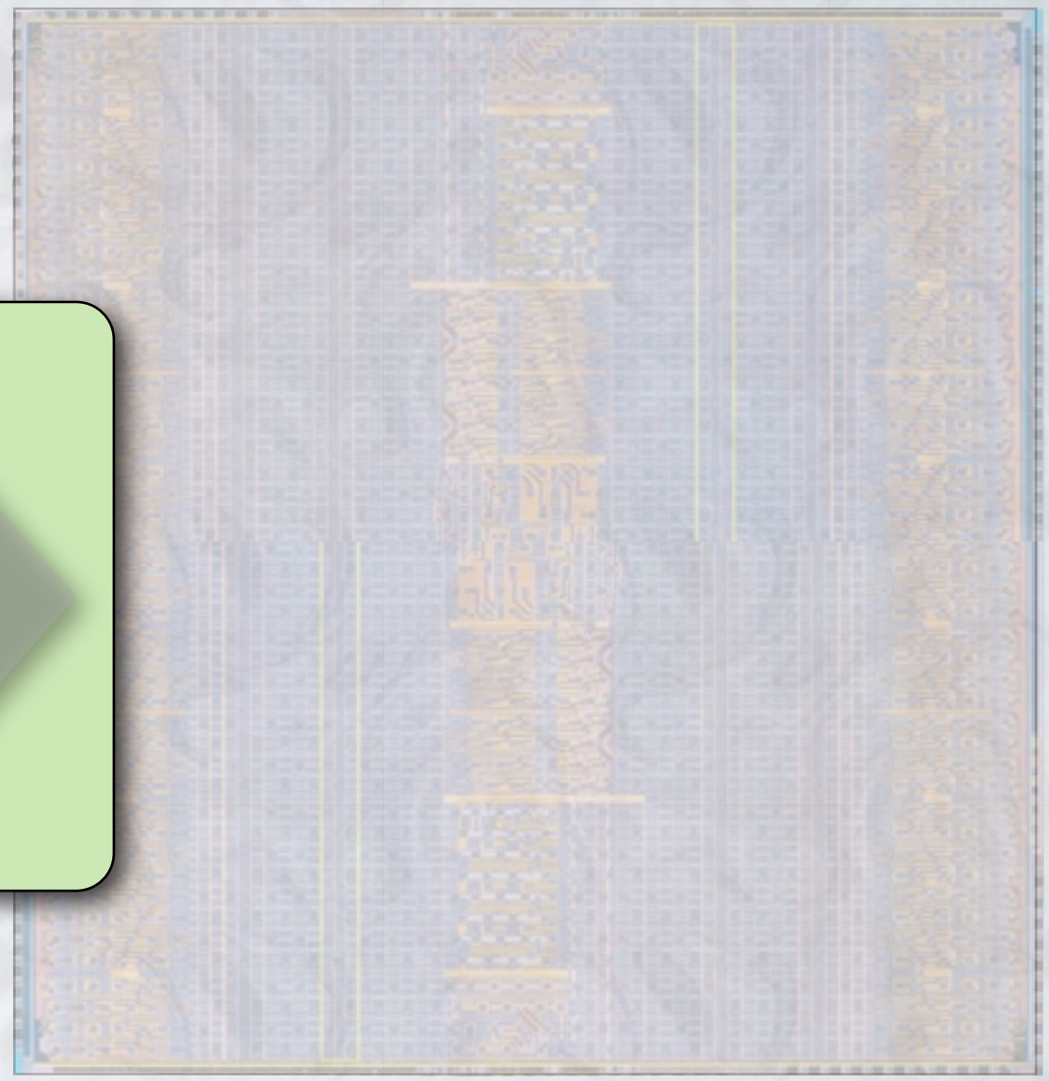


Co-Execution and Migration



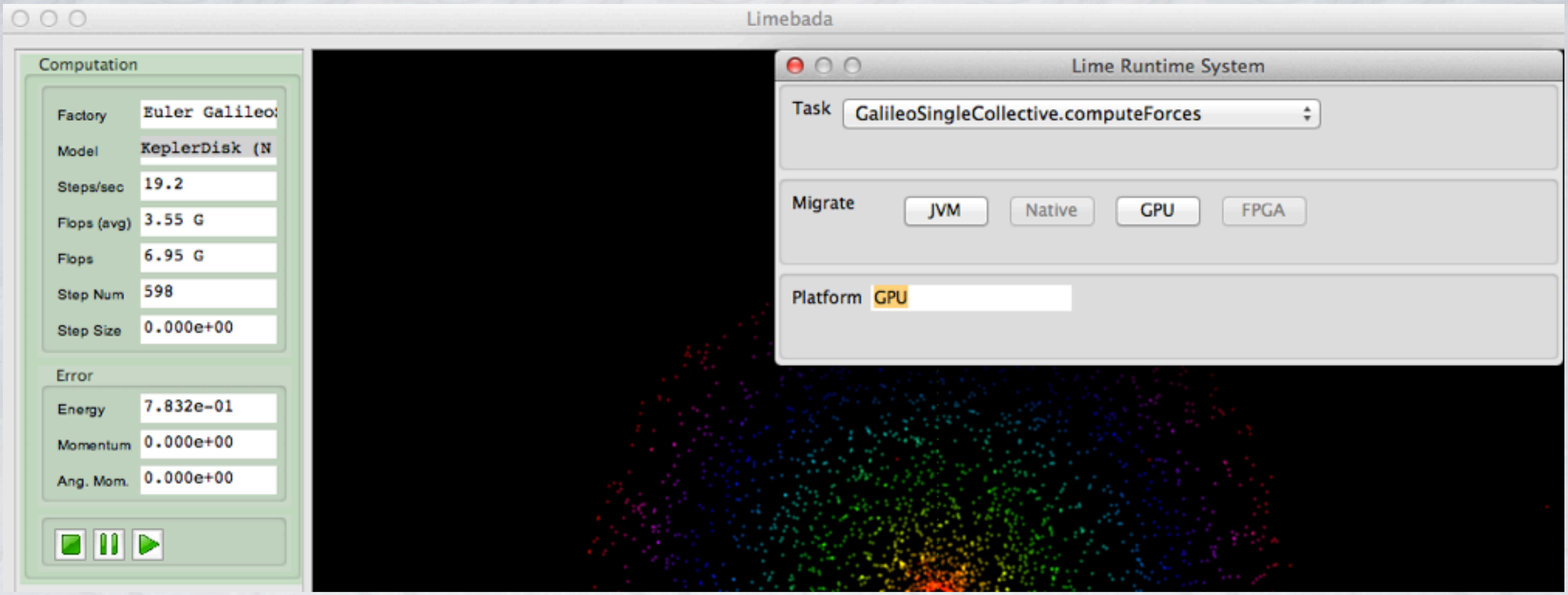
Co-Execution and Migration





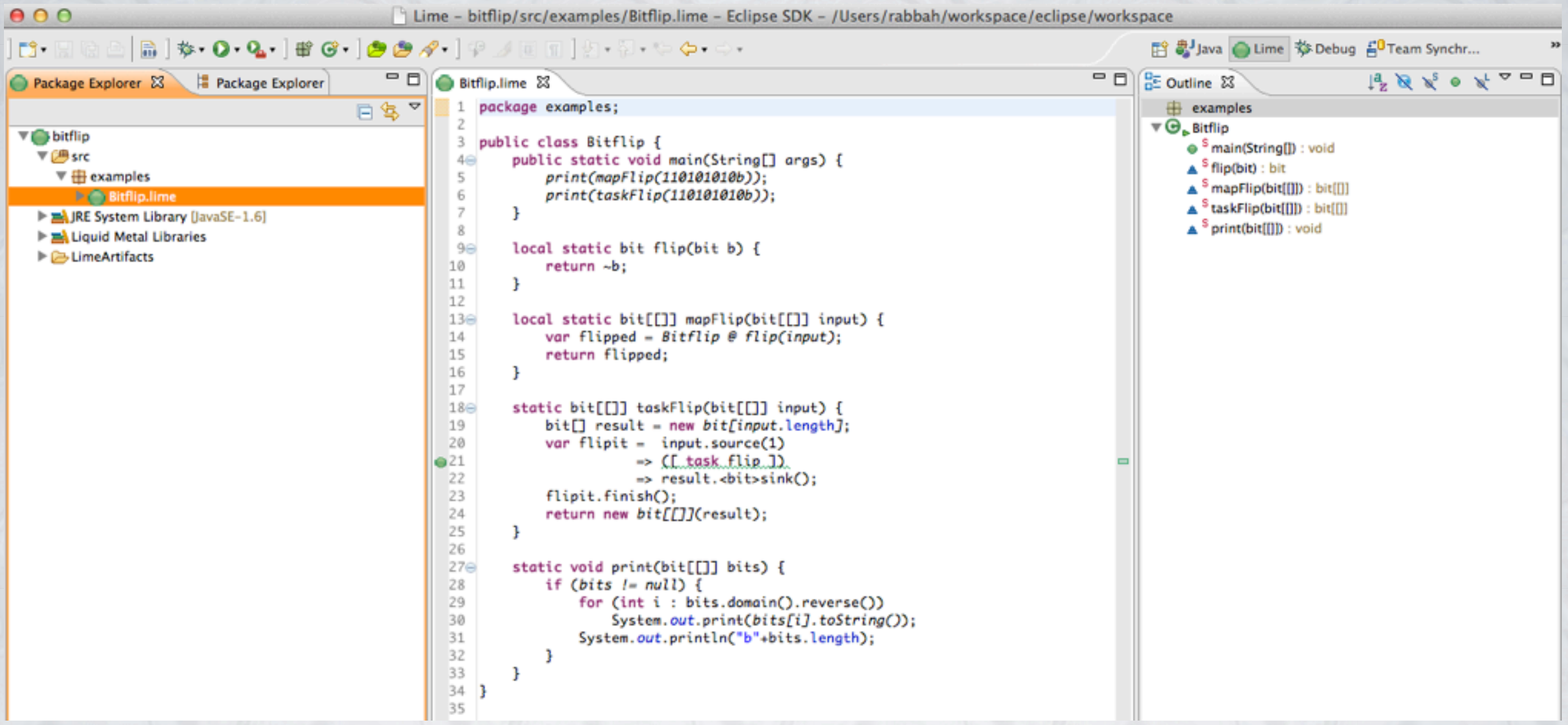
Lime
Runtime

Live Demo



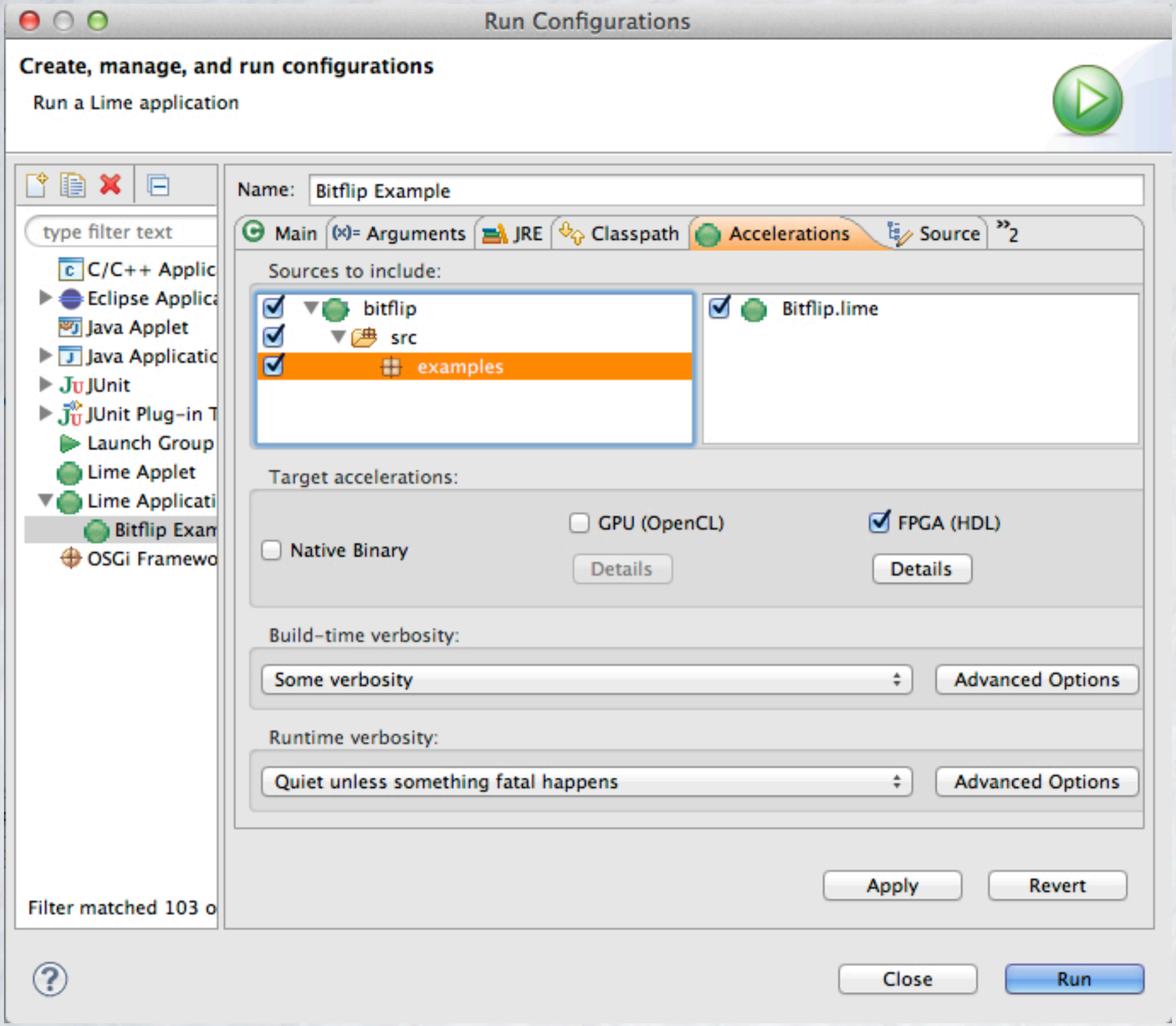
- N-Body simulation, dynamic migration from CPU to GPU
- 9x performance improvement (CPU: 1GFLOP, GPU: 9GFLOP)

ECLIPSE-BASED IDE



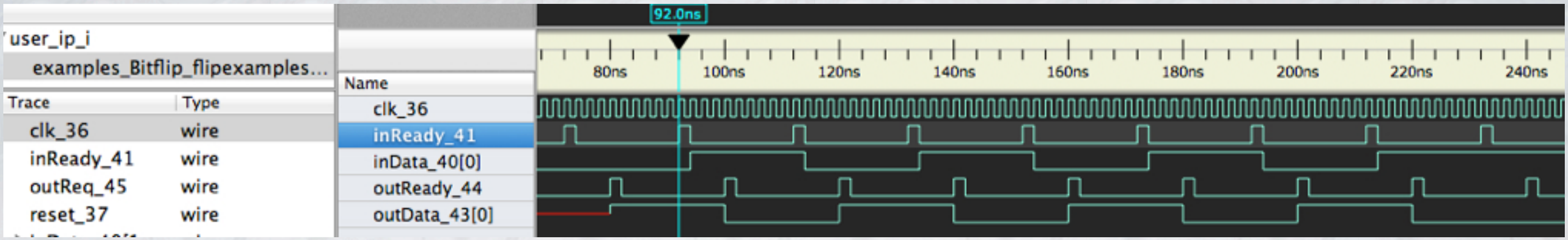
Runs on Windows, Linux, and Mac OS X (anywhere Eclipse/Java can run)

Integrated Compile & Run



- AMD & NVidia GPUs
- FPGA/RTL simulators
- Xilinx & Altera FPGAs

FPGA Co-Execution & Simulation



THE LIME LANGUAGE

Lime Core Language Features

Programmable Primitives

immutable & bounded types

Stream Programming

“functional” core

Map/Reduce Operations

operator overloading

A Lime Example

```
flipBits: 1101b ⟶ 0010b
```

A Lime Example - Features 1/2

```
local static bit flip(bit b) { return ~b; }
```

A Lime Example - Features 1/2

value types

```
local static bit flip(bit b) { return ~b; }
```

A Lime Example - Features 1/2

value types

user defined operators

```
local static bit flip(bit b) { return ~b; }
```

A Lime Example - Features 1/2

value types

user defined operators

local side-effects

```
local static bit flip(bit b) { return ~b; }
```

A Lime Example - Features 1/2

value types

user defined operators

local side-effects

```
local static bit flip(bit b) { return ~b; }
```

PURE FUNCTION = VALUE + LOCAL + STATIC

A Lime Example - Features 2/2

```
static bit[[]] flipBits(bit[[]] input) {

  bit[] result = new bit[input.length];

  var flipit =  input.source()
                => task flip
                => result.<bit>sink();

  flipit.finish();

  return new bit[[]](result);
}
local static bit flip(bit b) { return ~b; }
```

A Lime Example - Features 2/2

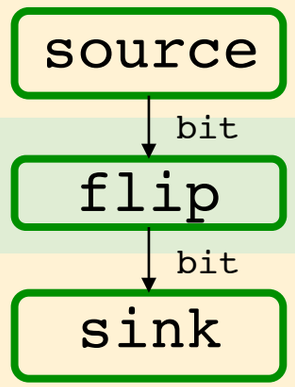
```
static bit[[]] flipBits(bit[[]] input) {

  bit[] result = new bit[input.length];

  var flipit =  input.source()
                => task flip
                => result.<bit>sink();

  flipit.finish();

  return new bit[[]](result);
}
local static bit flip(bit b) { return ~b; }
```

A Lime Example - Features 2/2

```
static bit[[]] flipBits(bit[[]] input) {

  bit[] result = new bit[input.length];

  var flipit =  input.source()
                => task flip
                => result.<bit>sink();

  flipit.finish();

  return new bit[[]](result);
}
local static bit flip(bit b) { return ~b; }
```

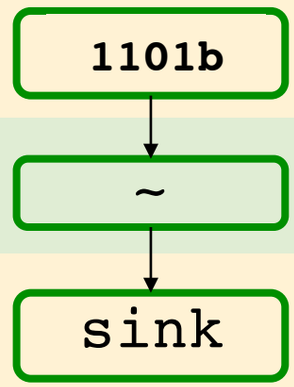
A Lime Example - task, =>

```
static bit[[]] flipBits(bit[[]] input) {

  bit[] result = new bit[input.length];

  var flipit =  input.source()
             => task flip
             => result.<bit>sink();

  flipit.finish();

  return new bit[[]](result);
}
local static bit flip(bit b) { return ~b; }
```



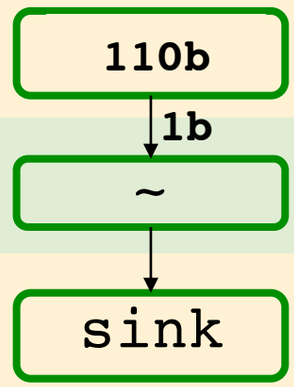
A Lime Example - task, =>

```
static bit[[]] flipBits(bit[[]] input) {

  bit[] result = new bit[input.length];

  var flipit =  input.source()
             => task flip
             => result.<bit>sink();

  flipit.finish();

  return new bit[[]](result);
}
```



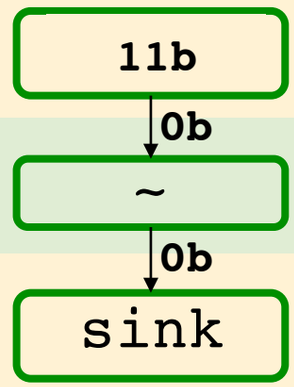
A Lime Example - task, =>

```
static bit[[]] flipBits(bit[[]] input) {

  bit[] result = new bit[input.length];

  var flipit =   input.source()
              => task flip
              => result.<bit>sink();

  flipit.finish();

  return new bit[[]](result);
}
```



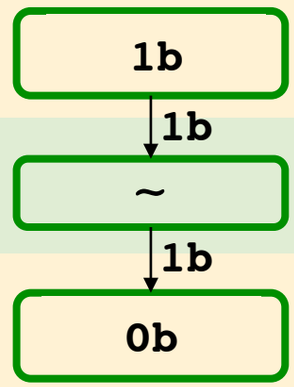
A Lime Example - task, =>

```
static bit[[]] flipBits(bit[[]] input) {

  bit[] result = new bit[input.length];

  var flipit =  input.source()
             => task flip
             => result.<bit>sink();

  flipit.finish();

  return new bit[[]](result);
}
```



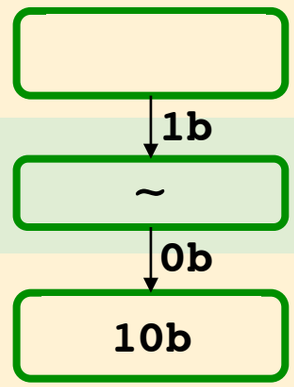
A Lime Example - task, =>

```
static bit[[]] flipBits(bit[[]] input) {

  bit[] result = new bit[input.length];

  var flipit =   input.source()
              => task flip
              => result.<bit>sink();

  flipit.finish();

  return new bit[[]](result);
}
```



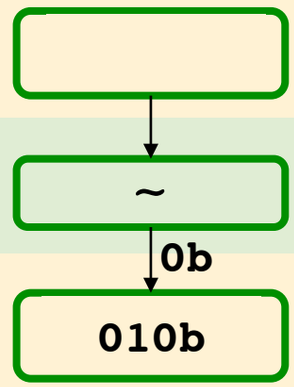
A Lime Example - task, =>

```
static bit[[]] flipBits(bit[[]] input) {

  bit[] result = new bit[input.length];

  var flipit =   input.source()
              => task flip
              => result.<bit>sink();

  flipit.finish();

  return new bit[[]](result);
}
```



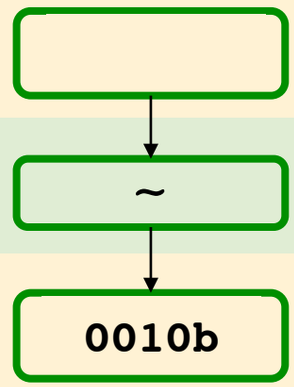
A Lime Example - task, =>

```
static bit[[]] flipBits(bit[[]] input) {

  bit[] result = new bit[input.length];

  var flipit =   input.source()
              => task flip
              => result.<bit>sink();

  flipit.finish();

  return new bit[[]](result);
}
```



A Lime Example - task, =>

```
static bit[[]] flipBits(bit[[]] input) {

  bit[] result = new bit[input.length];

  var flipit =   input.source()
              => task flip
              => result.<bit>sink();

  flipit.finish();

  return new bit[[]](result);
}
```

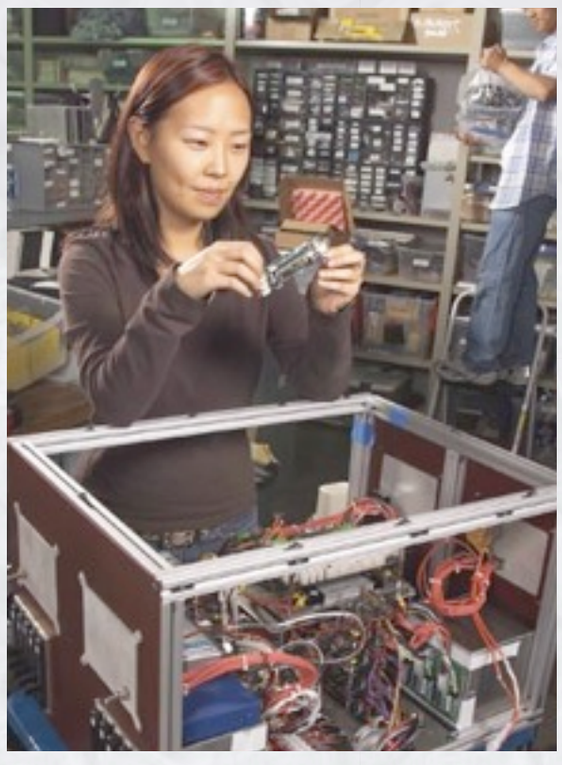


Virtualization of Data Movement

Virtualization of Data Movement

=>

+



x

3

months

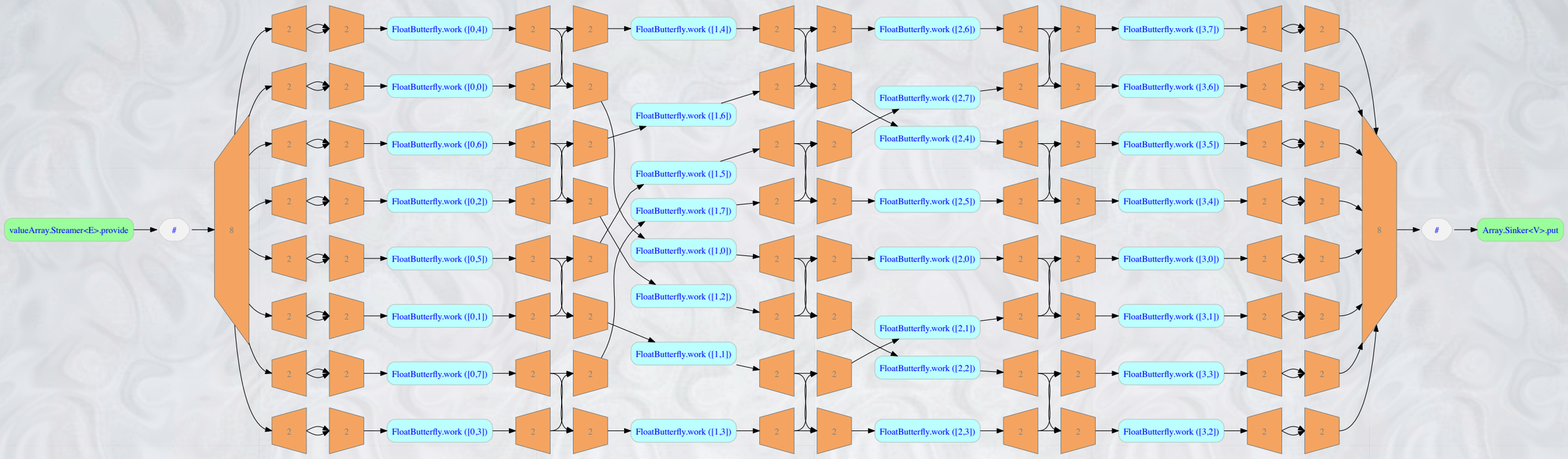
RELOCATION BRACKETS


```
class NBody {

   static void simulate() {
     float[][4] particles = ...; // initial state

     Task nbody =   task NBody(particles).particleGenerator
                 => ([ task forceComputation ])
                 => task NBody(particles).forceAccumulator;
     nbody.finish();
   }

   float[][4] particles;

   NBody(float[][4] particles) { this.particles = particles; }
   ...
}
```

RELOCATION BRACKETS


```
class NBody {

   static void simulate() {
     float[][4] particles = ...; // initial state

     Task nbody =   task NBody(particles).particleGenerator
                 => ([ task forceComputation ])
                 => task NBody(particles).forceAccumulator;
     nbody.finish();
   }

   float[][4] particles;

   NBody(float[][4] particles) { this.particles = particles; }
   ...
}
```

Elaborate Graph Construction



Liquid Metal Summary

- Single set of parallel abstractions for all devices
- Canonical unit of migration and adaptation
- A framework for introspection and adaptation

- No magic
  - parallel programming is not easier
  - one version of a program may not run well on all devices
  - vanilla Java programs will not run on accelerators