

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

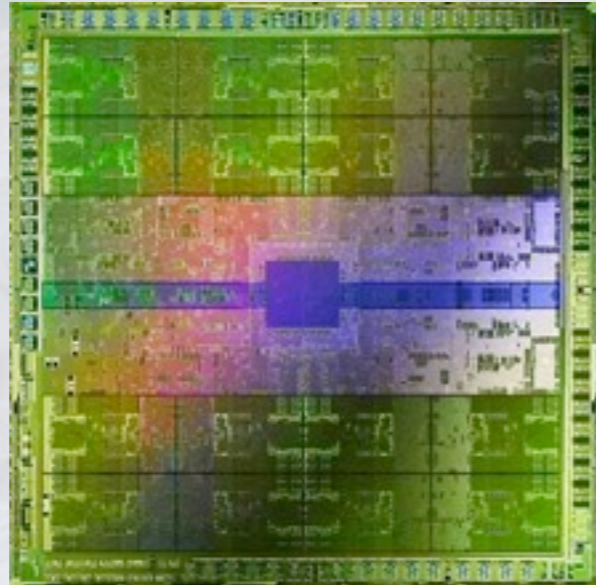
Andrei Hagiescu

Amir Hormati

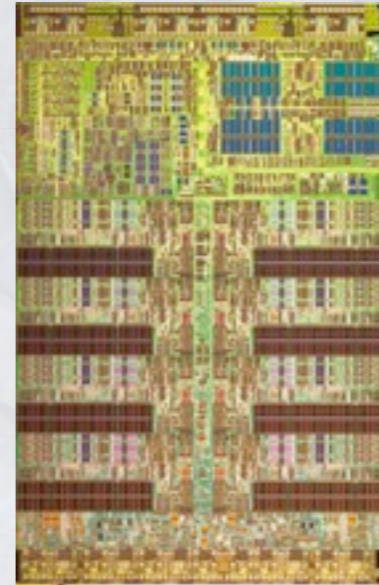
Shan Shan Huang

Myron King

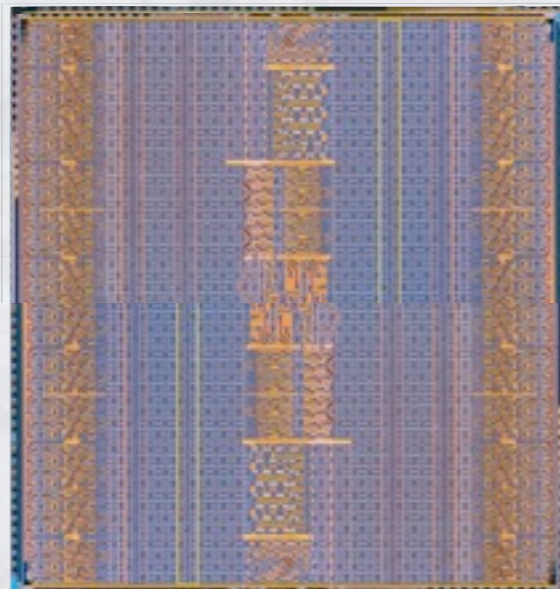
A HETEROGENEOUS REALITY



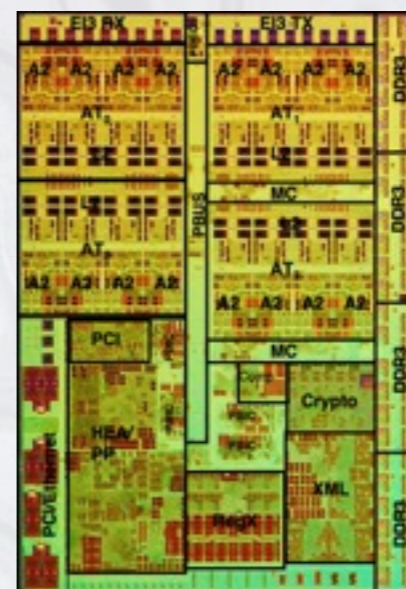
GPU



Multicore

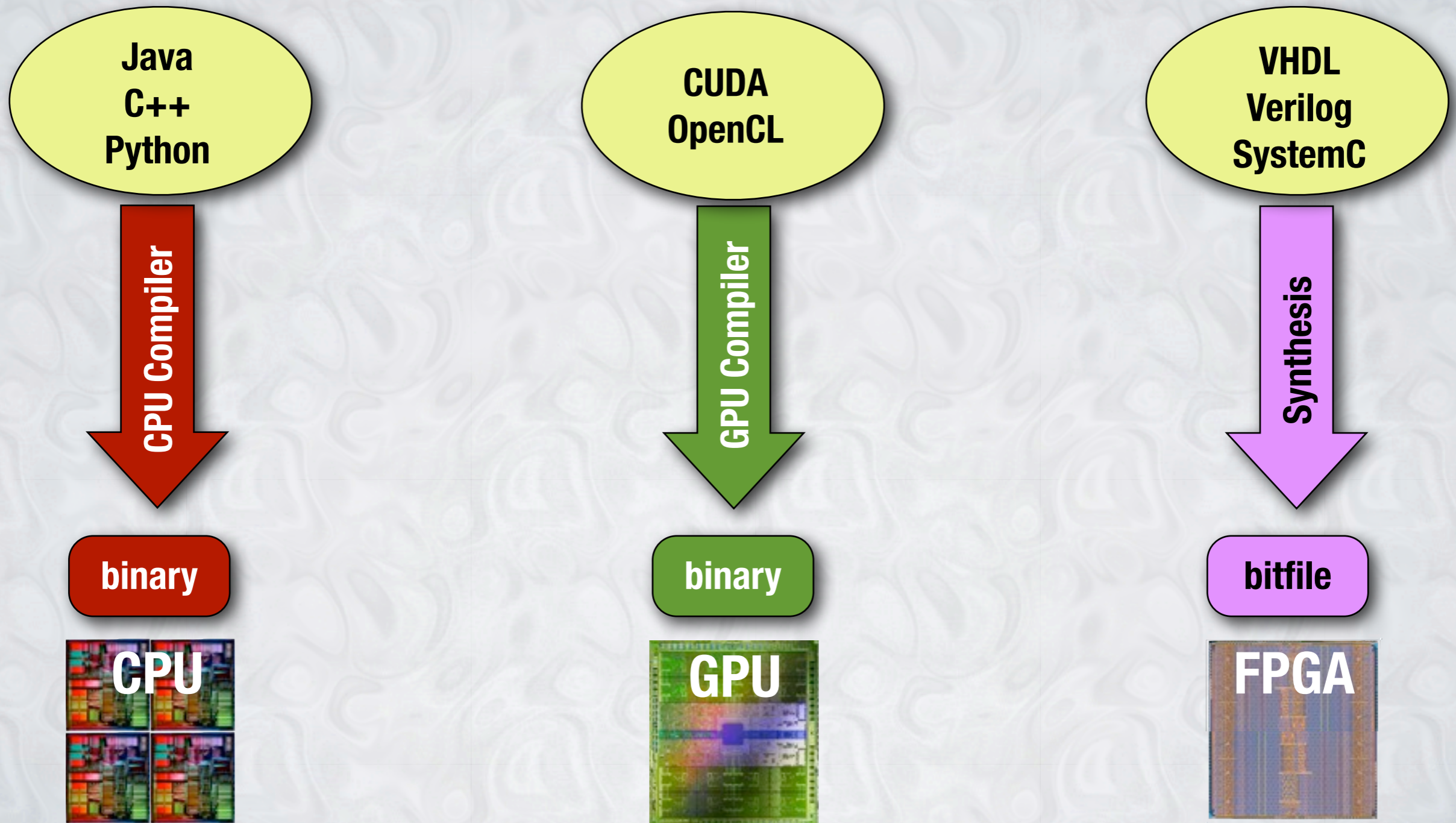


FPGA



ASIP

THE REAL SOFTWARE BURDEN: HETEROGENEOUS PROGRAMMING



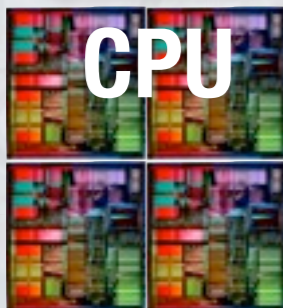
THE LIQUID METAL PROGRAMMING LANGUAGE



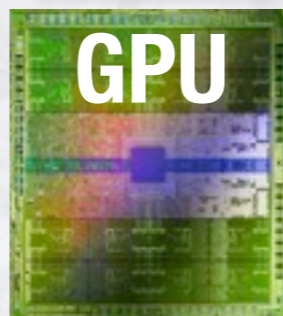
Lime Compiler



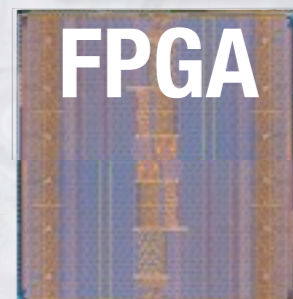
bytecode



binary



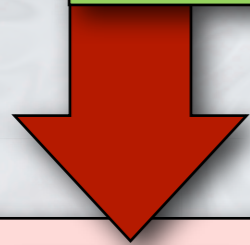
bitfile



LIME ARTIFACT STORE



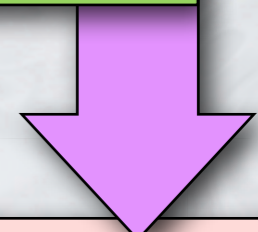
Lime Compiler



bytecode

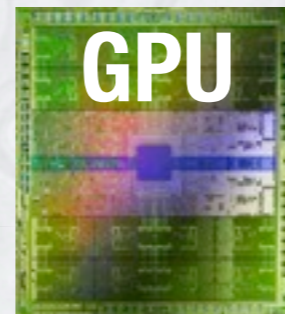
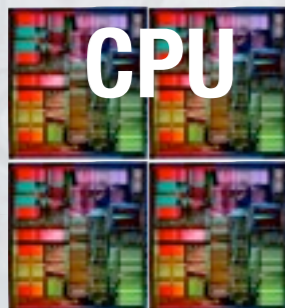


binary



bitfile

Artifact Store



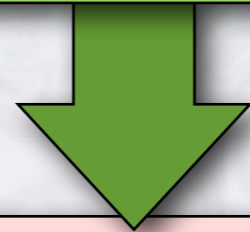
LIME VIRTUAL MACHINE



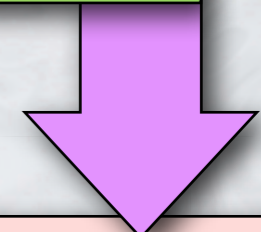
Lime Compiler



bytecode

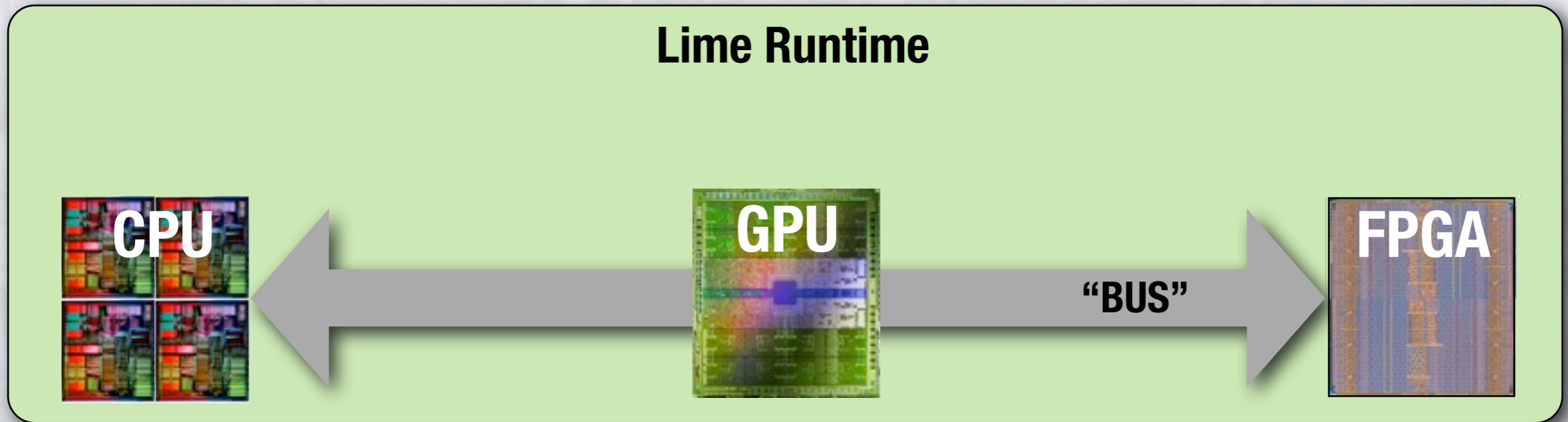


binary

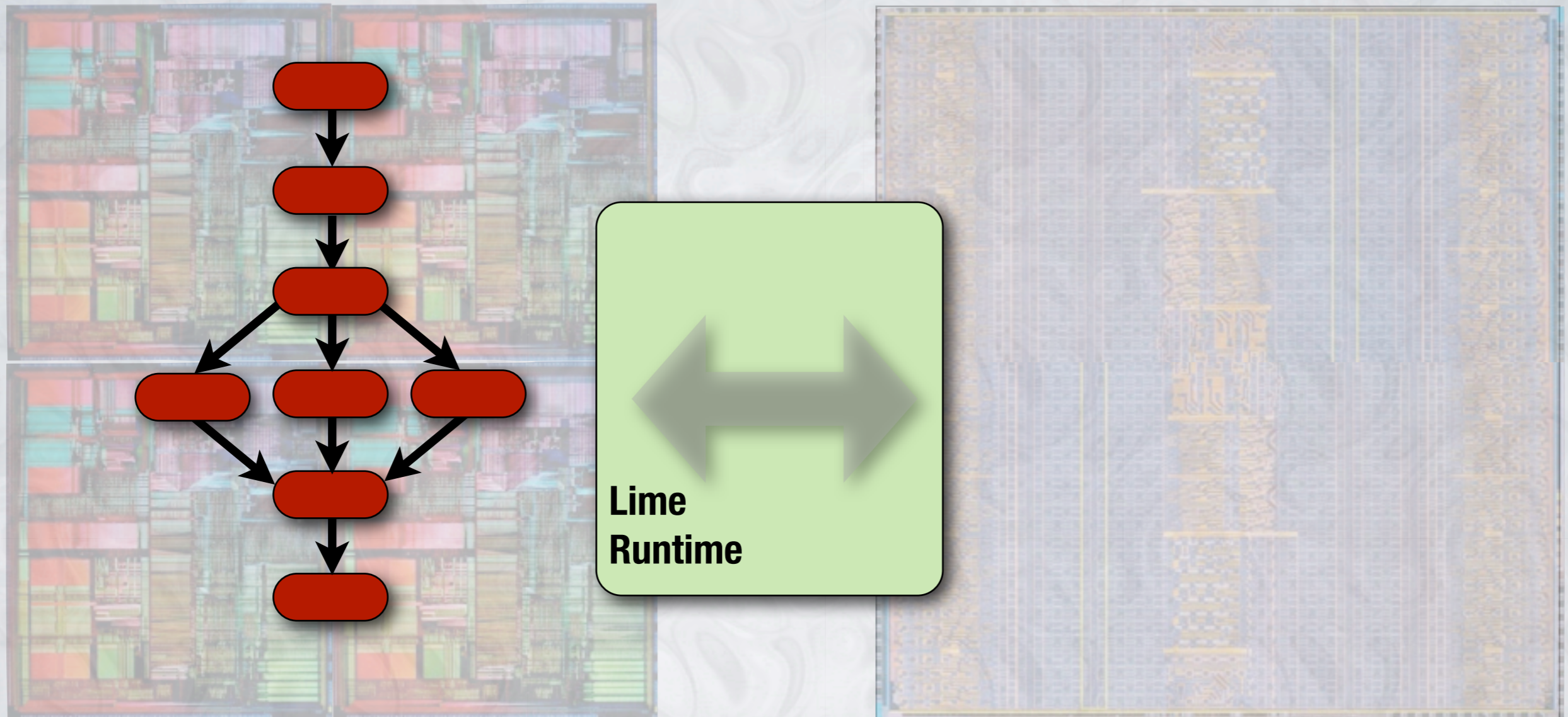


bitfile

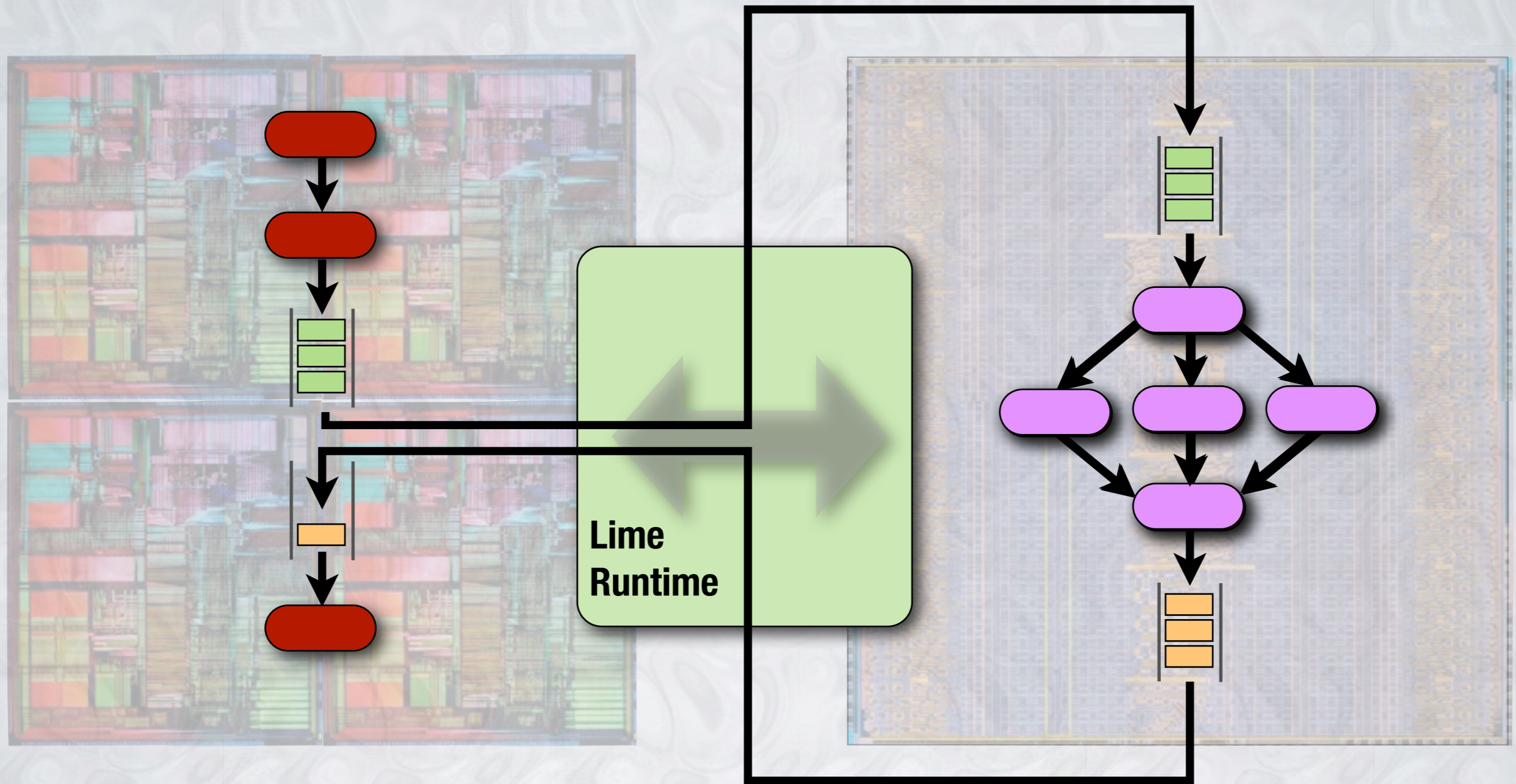
Artifact Store



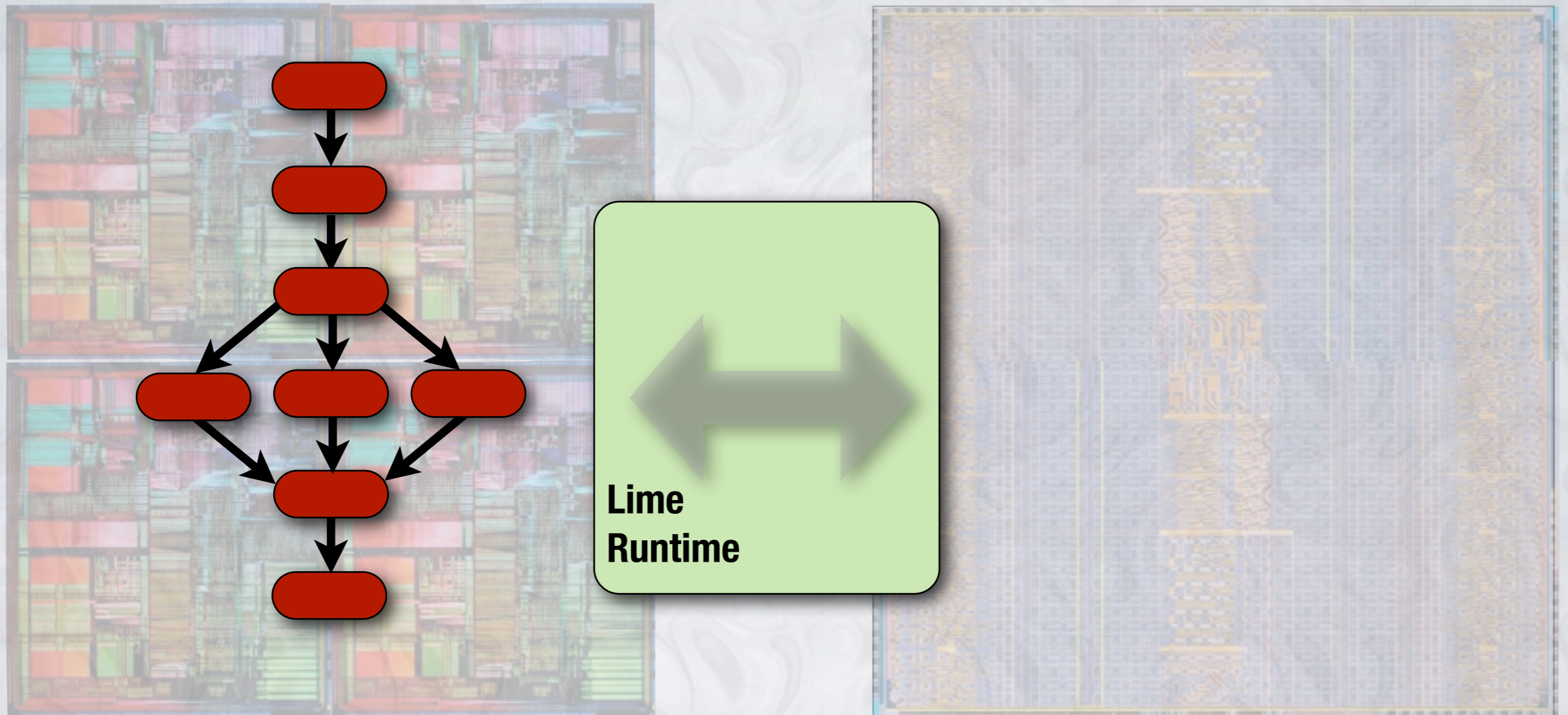
CO-EXECUTION AND MIGRATION



CO-EXECUTION AND MIGRATION



Co-EXECUTION AND MIGRATION



LIVE DEMO

The screenshot displays the Limebada application window, which is divided into several sections:

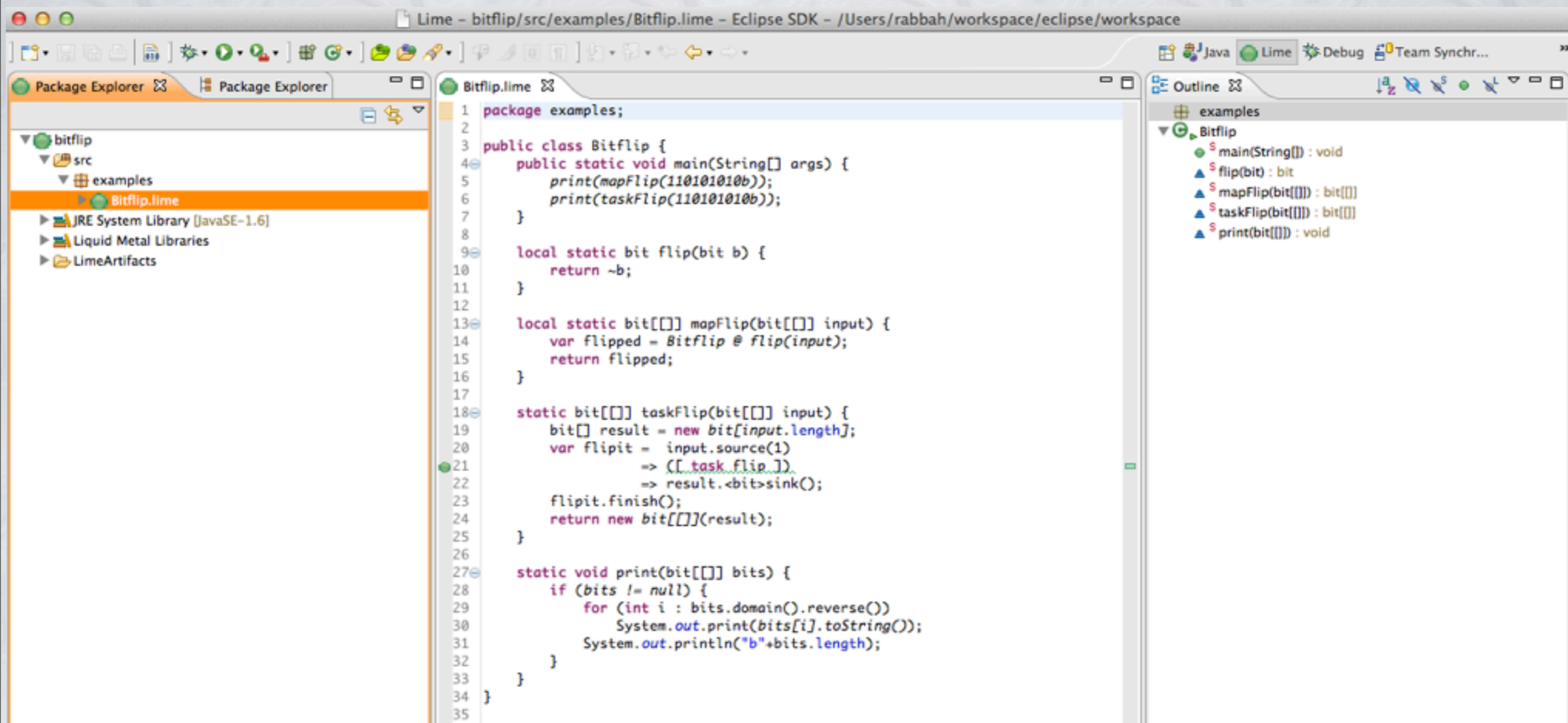
- Computation Panel (Left):** Contains simulation parameters and error metrics.

Computation	
Factory	Euler Galileo:
Model	KeplerDisk (N
Steps/sec	19.2
Flops (avg)	3.55 G
Flops	6.95 G
Step Num	598
Step Size	0.000e+00

Error	
Energy	7.832e-01
Momentum	0.000e+00
Ang. Mom.	0.000e+00
- Visualization (Center):** A dark space filled with a dense cluster of multi-colored particles (red, green, blue, yellow, orange) representing an N-body simulation.
- Lime Runtime System Panel (Right):** A control panel for the simulation.
 - Task:** GalileoSingleCollective.computeForces
 - Migrate:** Buttons for JVM, Native, GPU, and FPGA. The GPU button is highlighted.
 - Platform:** GPU

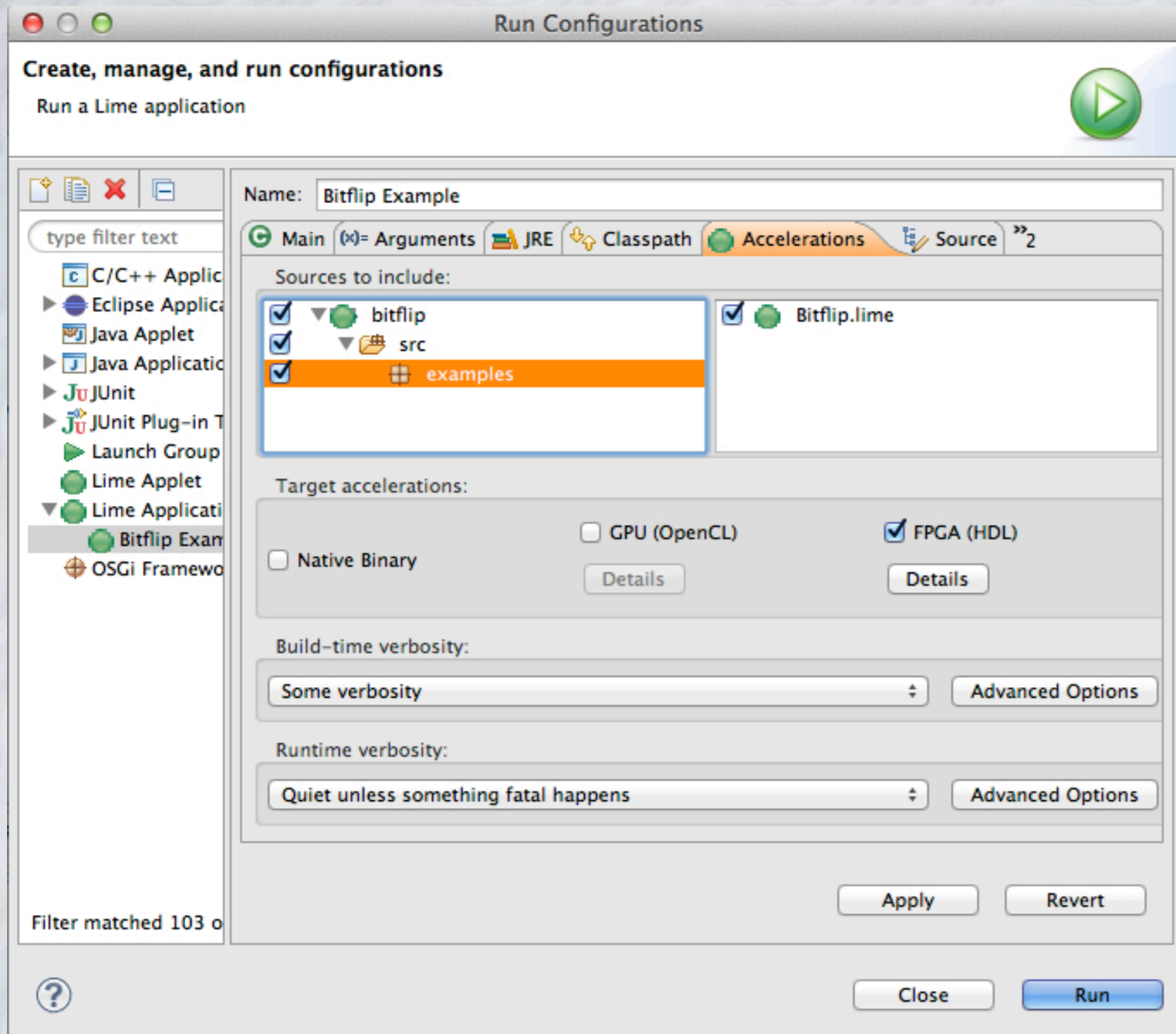
- **N-Body simulation, dynamic migration from CPU to GPU**
- **9x performance improvement (CPU: 1GFLOP, GPU: 9GFLOP)**

ECLIPSE-BASED IDE



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

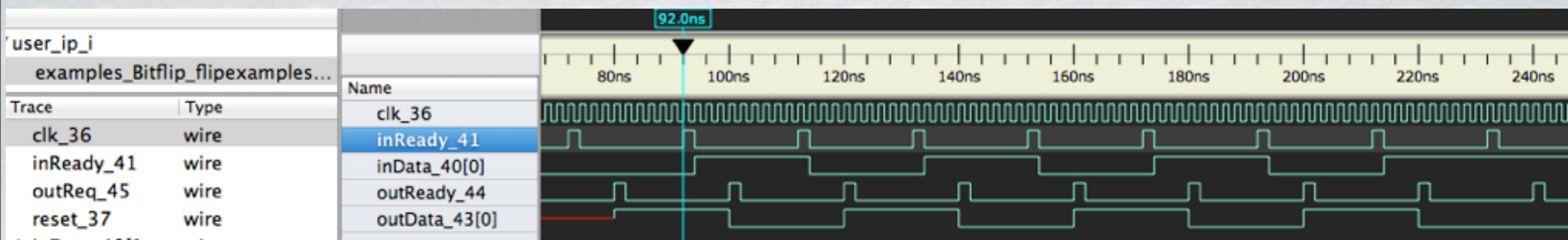
INTEGRATED COMPILE & RUN



- 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();
```

source

bit

flip

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();
```

1101b

~

sink

```
    flipit.finish();
```

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

```
}
```


A LIME EXAMPLE - `task, =>`

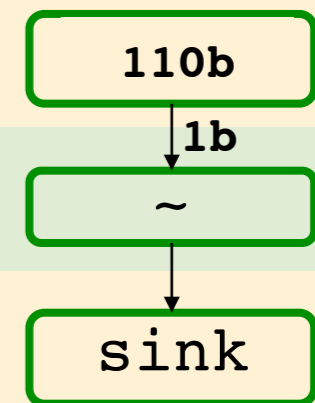
```
static bit[][] flipBits(bit[][] input) {
```

```
    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, =>`

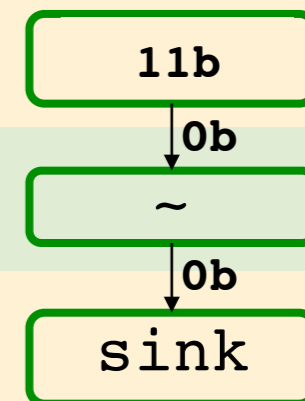
```
static bit[][] flipBits(bit[][] input) {
```

```
    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, =>`

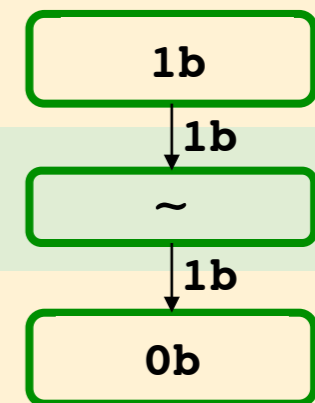
```
static bit[][] flipBits(bit[][] input) {
```

```
    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, =>`

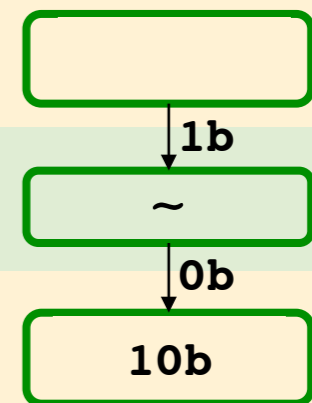
```
static bit[][] flipBits(bit[][] input) {
```

```
    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, =>`

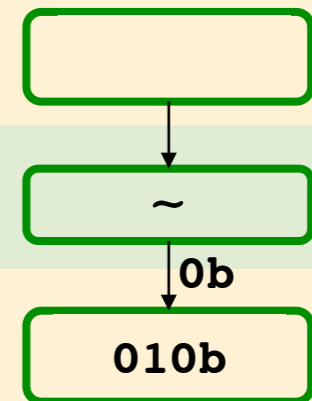
```
static bit[][] flipBits(bit[][] input) {
```

```
    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, =>`

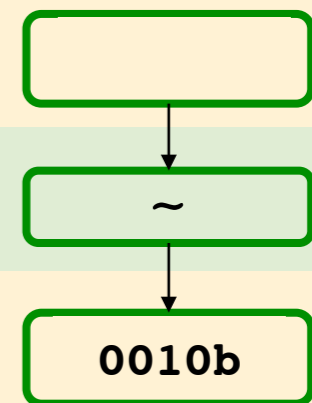
```
static bit[][] flipBits(bit[][] input) {
```

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

```
    var flipit = input.source()
```

```
    => task flip
```

```
    => result.<bit>sink();
```



```
    flipit.finish();
```

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

```
}
```

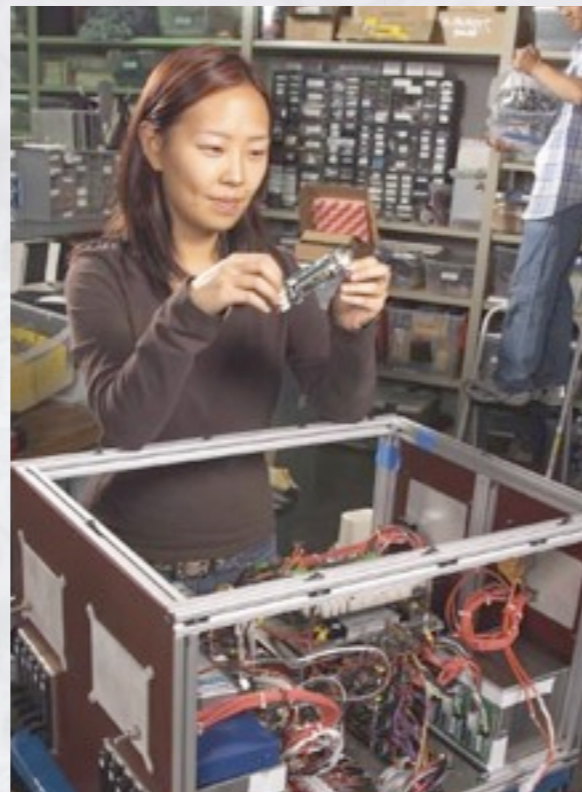

VIRTUALIZATION OF DATA MOVEMENT



VIRTUALIZATION OF DATA MOVEMENT



+

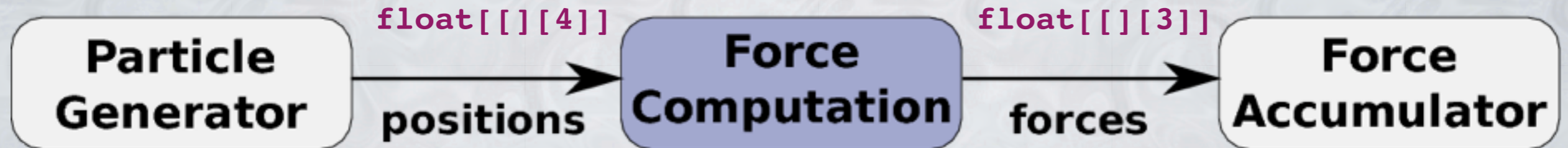


x

3

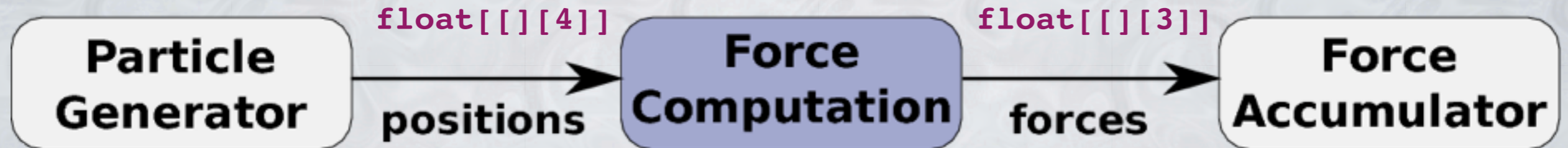
months

RELOCATION BRACKETS



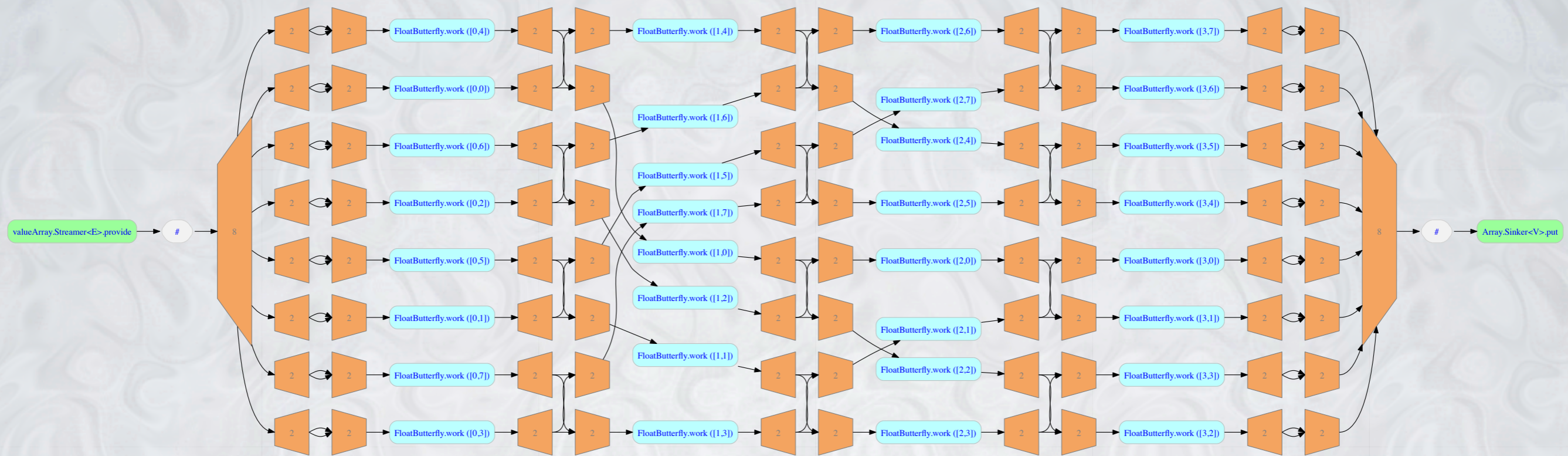
```
class NBody {  
  
    static void simulate() {  
        float[][][4] particles = ...; // initial state  
  
        Task nbody = task NBody(particles).particleGenerator  
                    => ([ task forceComputation ])  
                    => task NBody(particles).forceAccumulator;  
        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**