Liquid Metal

Programming for Heterogeneity

IBM Research

http://www.research.ibm.com/liquidmetal/

Rodric Rabbah

7/1/13

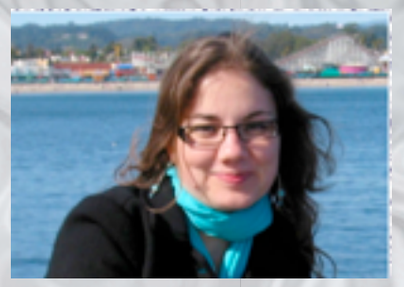
Liquid Metal

Erik Altman

Josh Auerbach

David Bacon

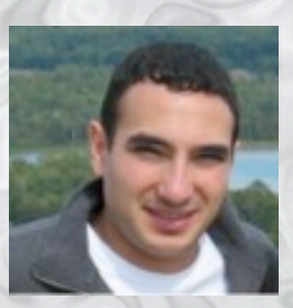
Ioana Baldini



Perry Cheng

Stephen Fink

Rodric Rabbah



Sunil Shukla

The von Neumann Trap

Our thirty year old belief that there is only one kind of computer is the basis of our belief that there is only one kind of programming language, the conventional von Neumann language.

– John Backus 1977 Turing Award Lecture, p. 615

The Path of Least Imagination

*For years, computer architects have been saying that a **big new idea** in computing was needed. Indeed, as transistors have continued to shrink, **rather than continuing to innovate**, computer designers have **simply adopted a so-called "multicore" approach**, where multiple processors are added as more chip real estate became available.*

The New York Times

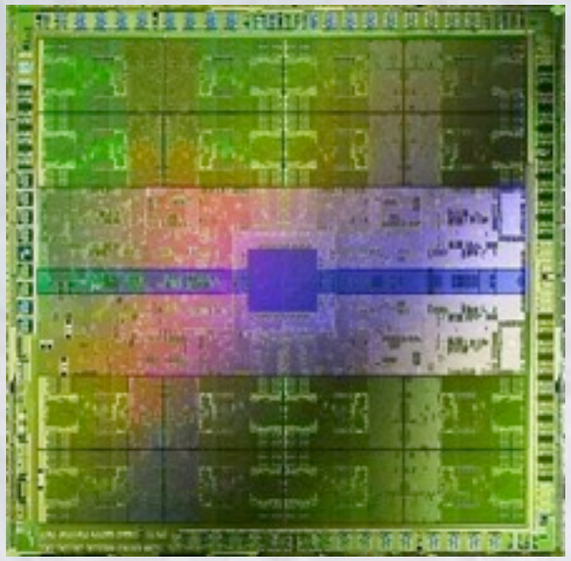
Source: Remapping Computer Circuitry to Avert Impending Bottlenecks, The New York Times, February 28, 2011

HETEROGENEOUS REALITY

Next-gen programmers will have a lot more to think about at the node level than in the past.

– Brad Chamberlain, Yesterday

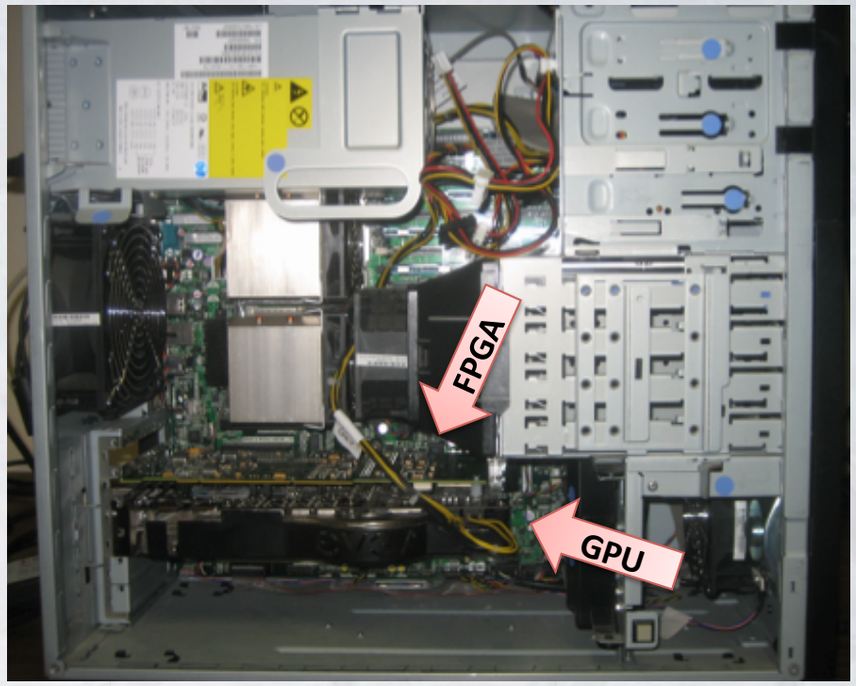
PERFORMANCE AT THE NODE

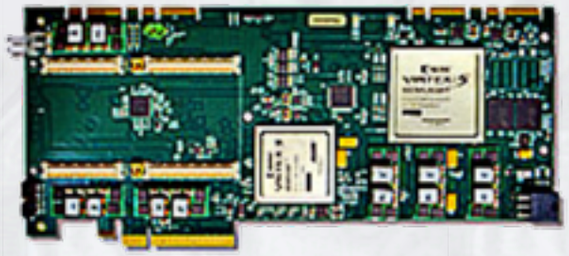


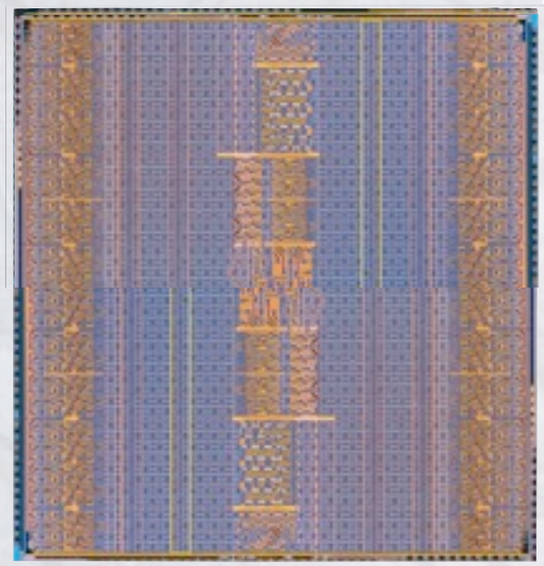
GPU

PCIe attached

x86 workstation with PCIe FPGA and GPU

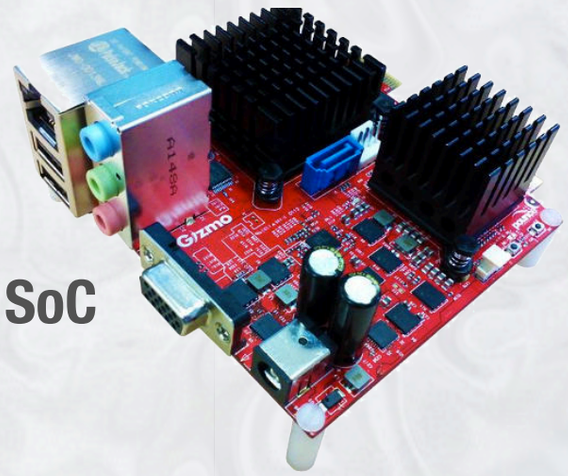






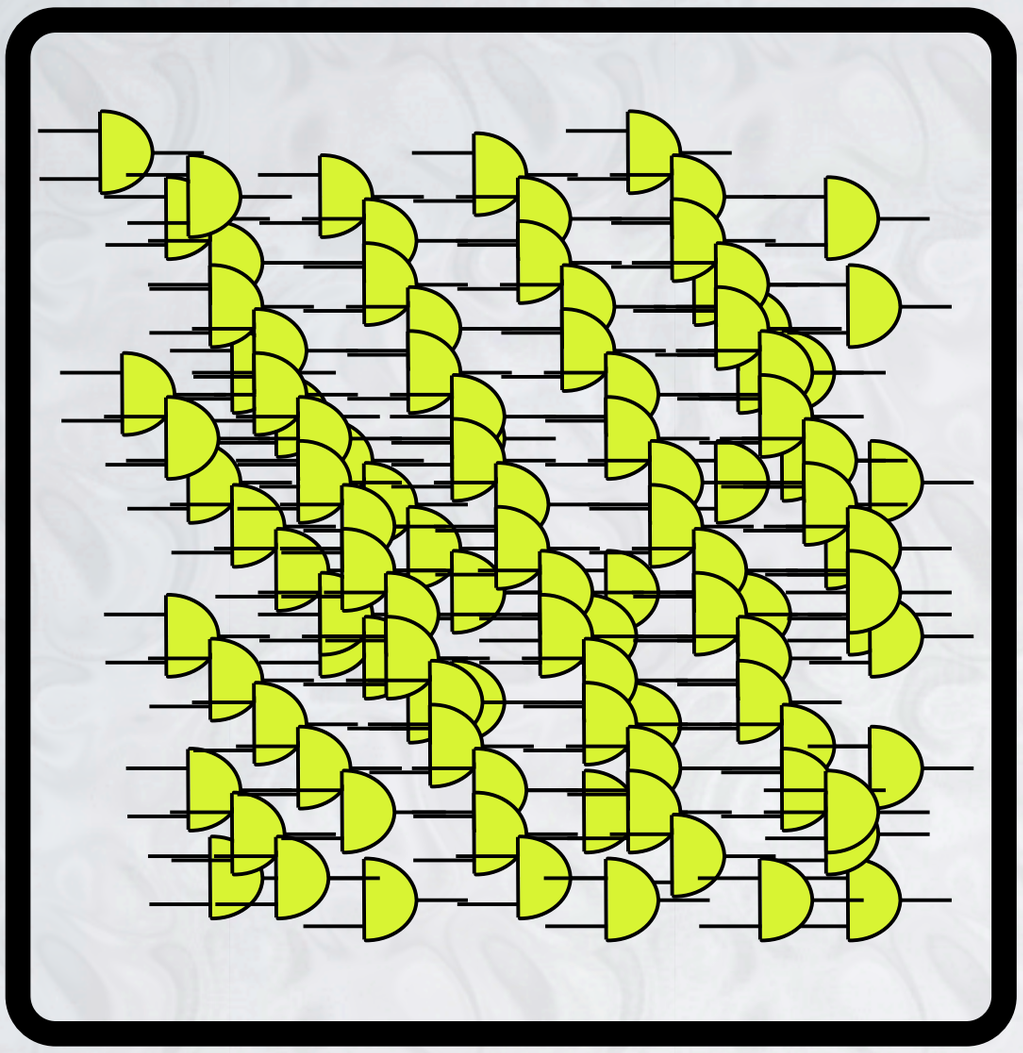
FPGA

Embedded SoC



FPGA - Quick Tutorial

- Programmable logic

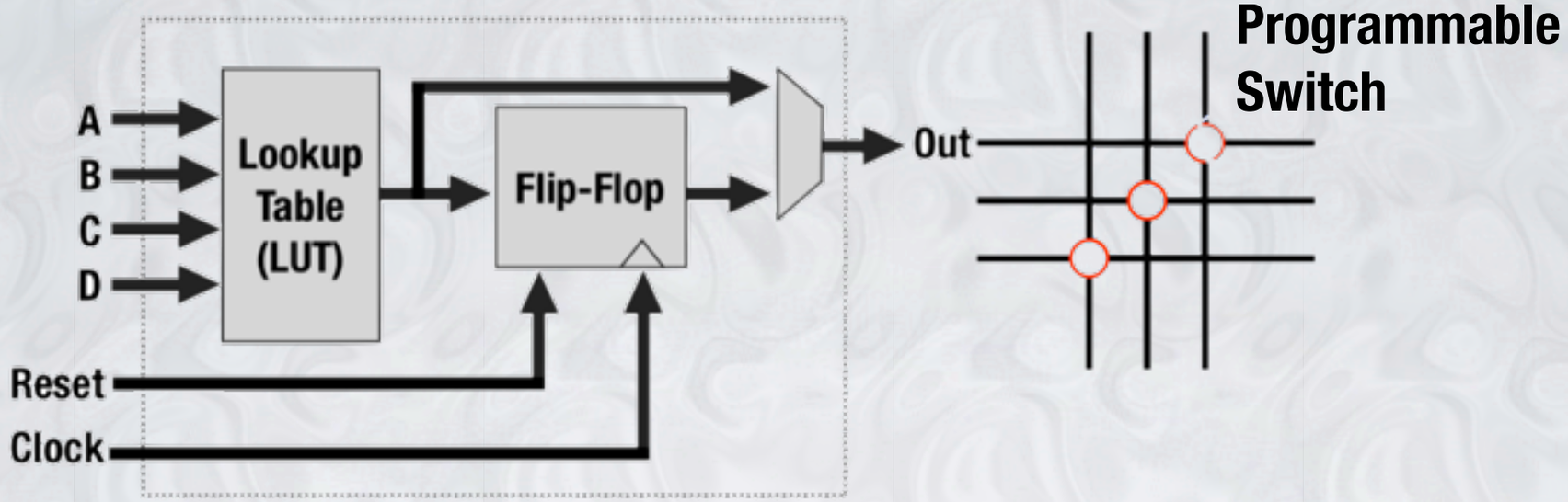


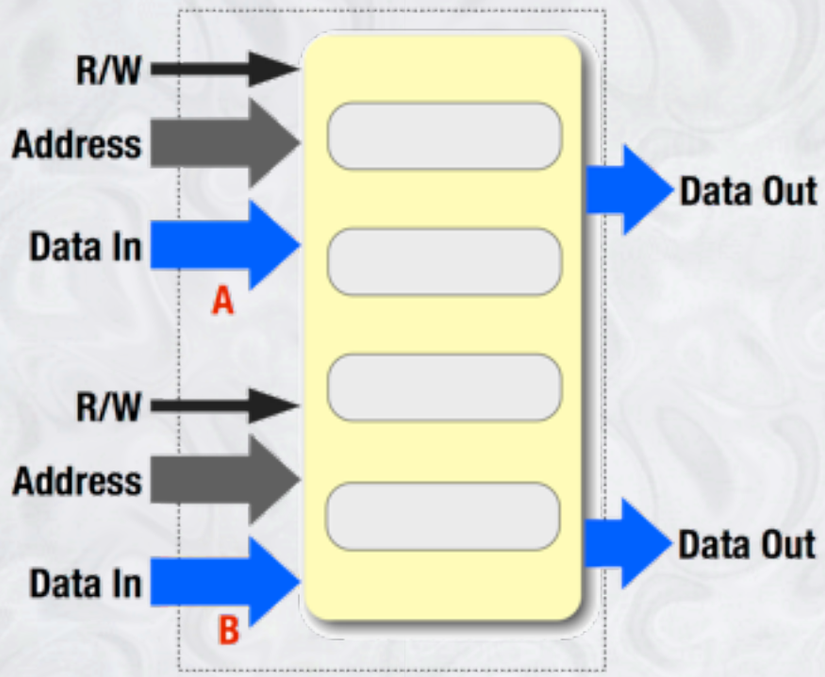
Programming FPGAs

- Programmed at very low level of abstraction
  - hardware description language (HDL)
  - same as designing custom circuits (ASICs)
  - bits and bit arrays are main abstraction

- Synthesis: compilation of HDL to circuit
  - minutes to hours

BUILDING BLOCKS



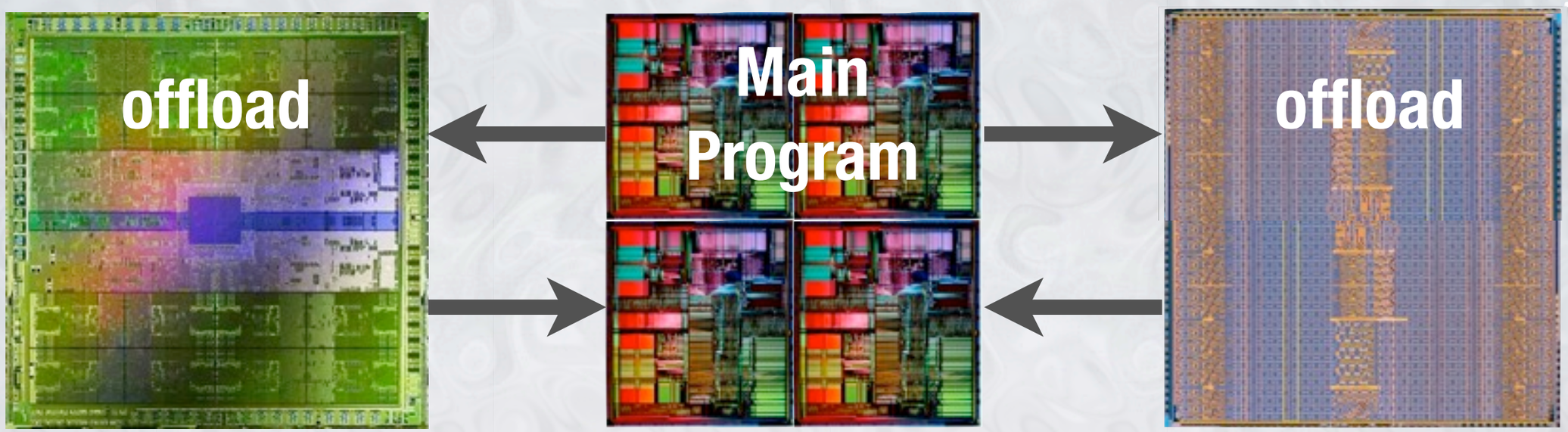


Block RAM (Memory)
- configurable bit-width
- 36Kb as
1K x 36 bits ...
18K x 2 bits
36K x 1 bit

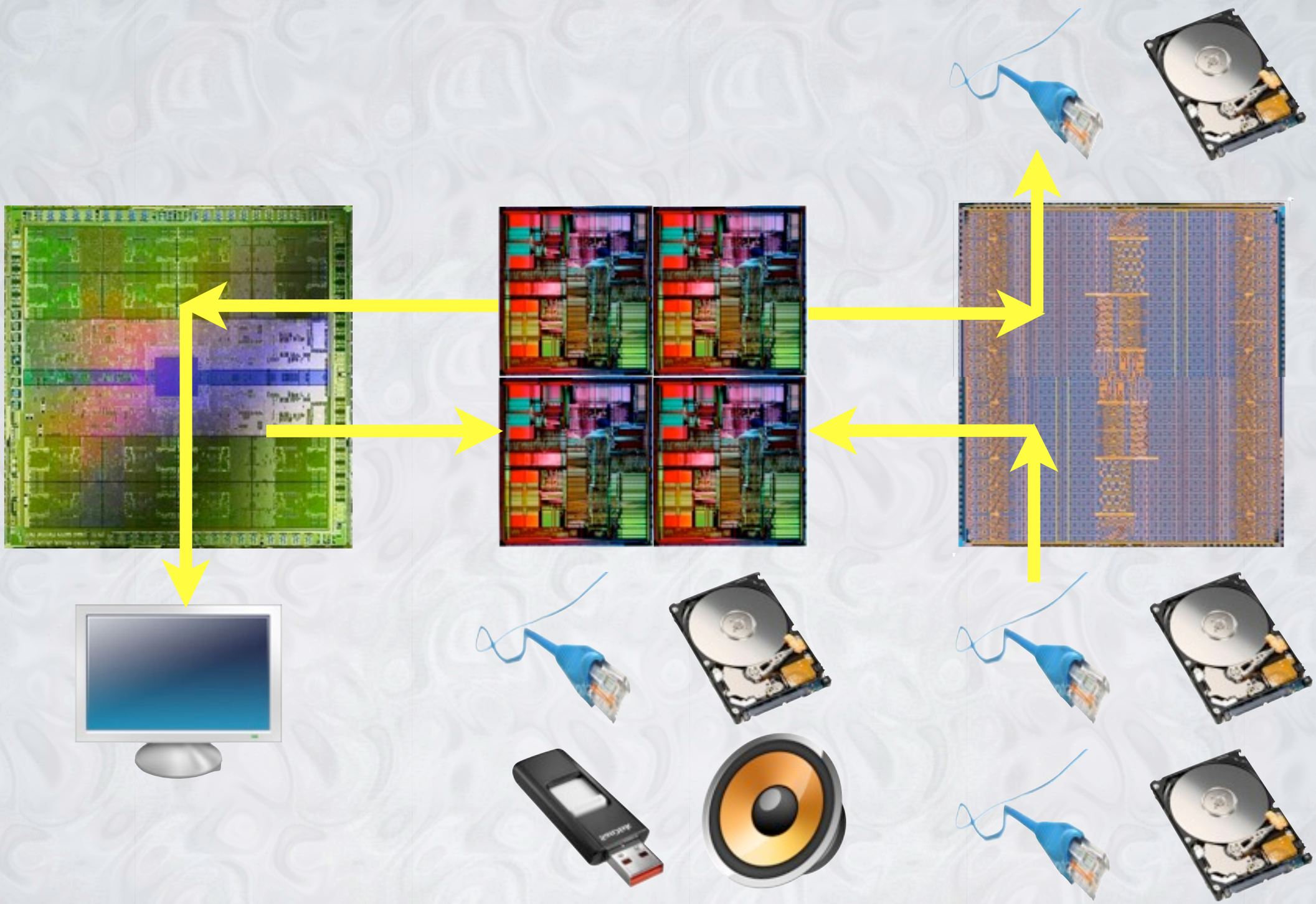
Unique Features

- Dual-ported Memory
- Read-before-Write Memory and Registers
- Simple synchronization
- Determinism
- “A Stall-Free Real-Time Garbage Collector for Reconfigurable Hardware”, Bacon et al. in PLDI 2012

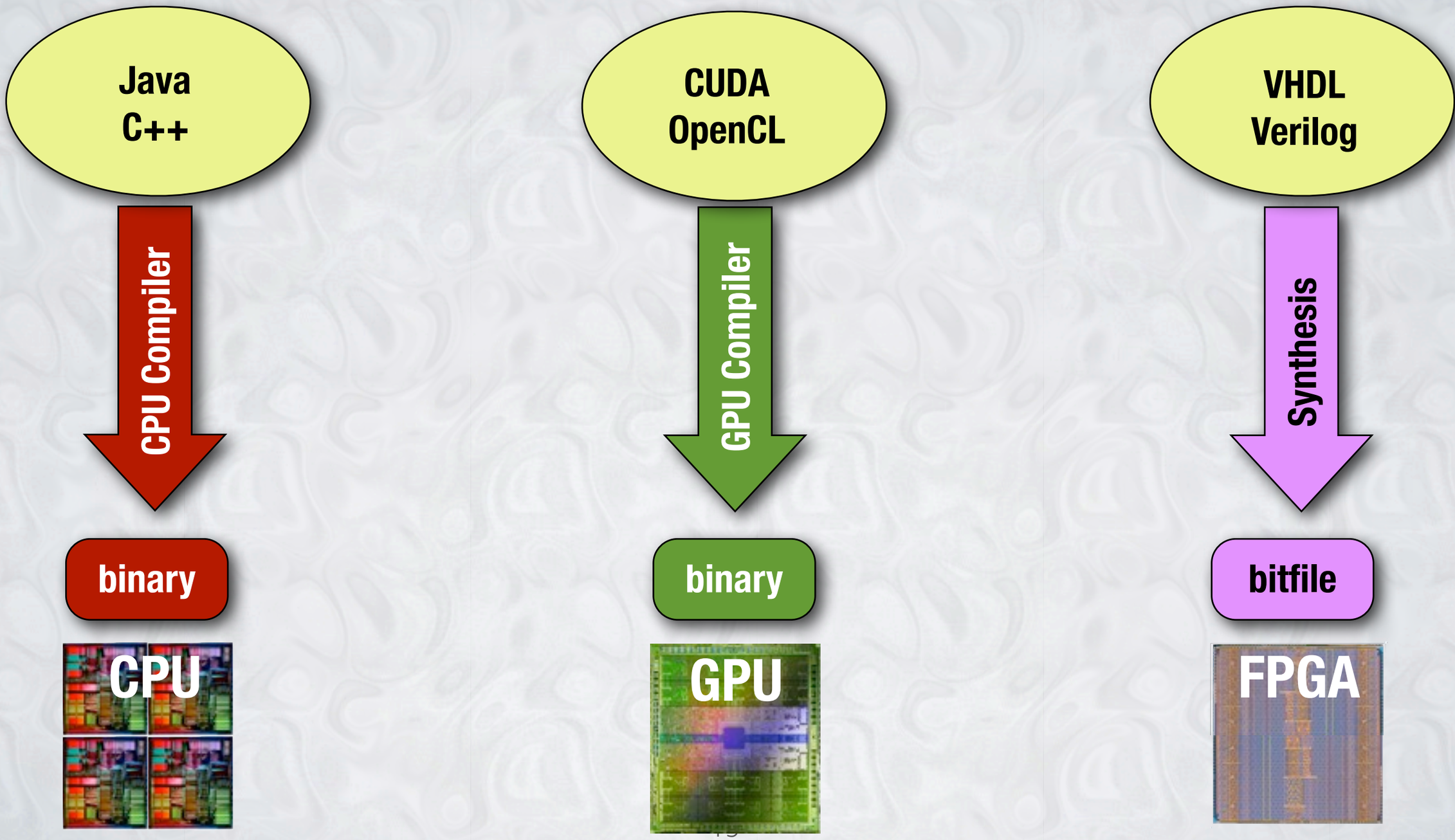
The "Accelerator" Model



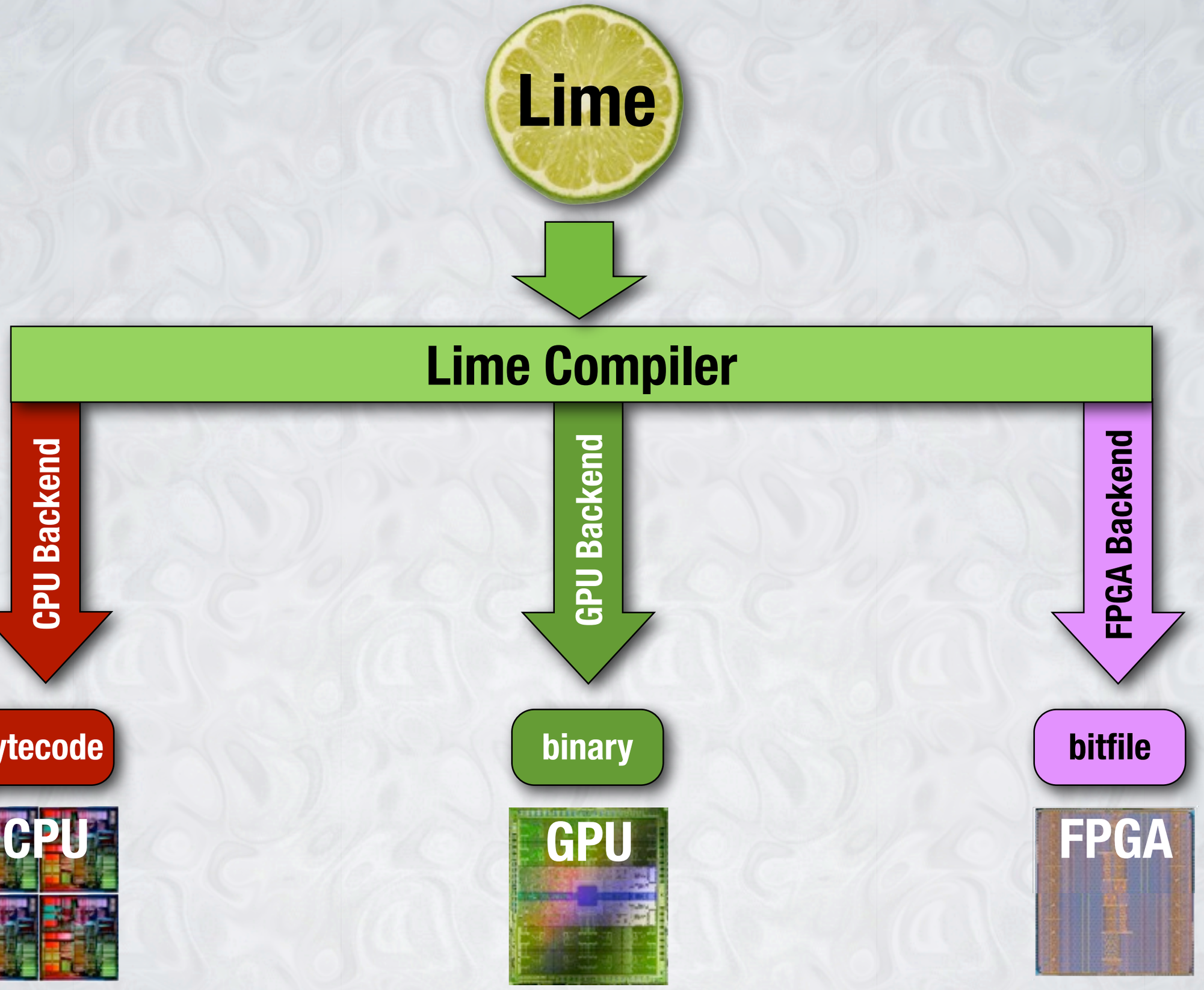
The Appliance Model



HETEROGENEOUS PROGRAMMING



The Liquid Metal Programming Language



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

Benchmarks and Applications

- Parboil, Rodinia, StreamIt, JavaGrande, CHStone
- Encryption, compression, random number generation, video photo mosaic, image processing, network stack

LIME LANGUAGE OVERVIEW

Core Features

Programmable Primitives

Stream Programming

Map/Reduce Operations

BIT-LEVEL PARALLELISM

Immutable Types
Bounded Types
Bounded Arrays
Primitive Supertypes

PIPELINE PARALLELISM

Graph Construction
Isolation Enforcement
Closed World Support
Rate Matching

DATA PARALLELISM

Supporting Features

Reifiable Generics
Ranges, Bounded “for”
User-defined operators

Typedefs
Local type inference
Tuples

Programmable Primitives

BIT-LEVEL PARALLELISM

PROGRAMMABLE PRIMITIVES: BITS

```
public value enum bit {
        zero, one;

}
```

PROGRAMMABLE PRIMITIVES: BITS

```
public value enum bit {
        zero, one;
        public bit ~ this {
            return this == zero ? one : zero;
        }
        public boolean this < (bit that) {
            return this == zero && that == one;
        }
}
```

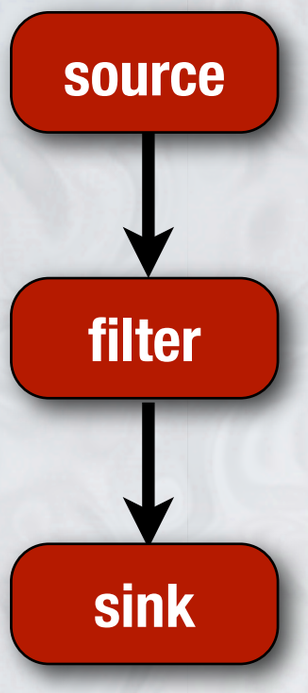
PROGRAMMABLE PRIMITIVES: NUMBERS

```
public final value class unsigned<N extends ordinal<N>> {

   bit[[N]] data;

   public boolean this < (unsigned<N> that) {
      for (N i: N.range.reverse())
         if (this.data[i] != that.data[i])
            return this.data[i] < that.data[i];
      return false;
   }

   public unsigned<N> this & (unsigned<N> that) {
      return new unsigned<N>(this.data &@ that.data);
   }

   public bit parity() {
      return ^! data;
   }
}
```

Stream Programming

PIPELINE PARALLELISM

Stream Programming



computation from methods with local side-effects

communication of values across edges

language support for creating task graphs

`local`: Confines Side-Effects

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

`local`: Confines Side-Effects

```
static bit MUTABLE = 1b; // global variable

local bit flip(bit b) {
  b = b & MUTABLE; // error: access to global var
  return ~b;
}
```

value: Immutable Objects

```
value class aBit {
  bit one = 1b; // bit literal

  bit ok() {
    return one & 1b;
  }
}
```

`value`: IMMUTABLE OBJECTS

```
value class aBit {
  bit one = 1b; // bit literal

  bit ok() {
    return one & 1b;
  }

  bit error() {
    one = 0b; // error: one is immutable
    return one;
  }
}
```

`value`: IMMUTABLE OBJECTS

```
value class aBit {
  bit one = 1b;

  bit ok() {
    return one & 1b;
  }
}

var ones = new aBit[[10]];
```

`value`: IMMUTABLE OBJECTS

```
value class aBit {
  bit one = 1b;

  bit ok() {
    return one & 1b;
  }
}

var ones = new aBit[[10]];
ones[3]  = ...; // error: ones[i] is immutable
```

LIME TASK GRAPH EXAMPLE

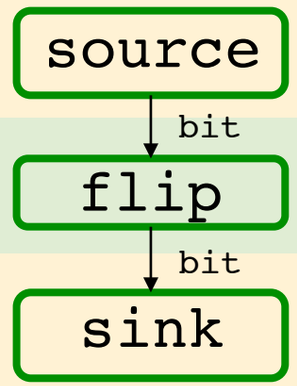
```
flipBits: 1101b ⟶ 0010b
```

Lime task

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

  var result = new bit[input.length];

  var graph  =  Tasks.source(input)
                => task flip
                => Tasks.sink(result);

  graph.finish();

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

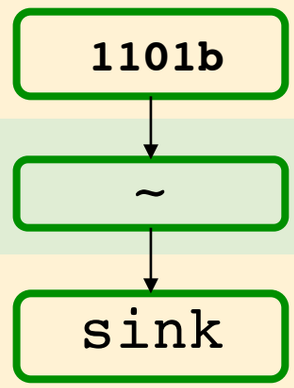
CONNECTING => TASKS

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

  var result = new bit[input.length];

  var graph  =  Tasks.source(input)
             =>  task flip
             => Tasks.sink(result);

  graph.finish();

  return result;
}

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



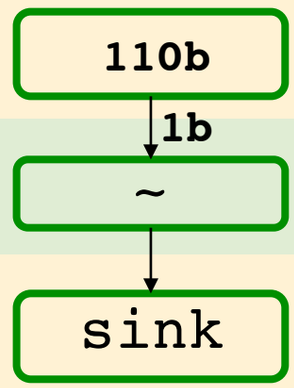
A Lime Example - task, =>

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

  var result = new bit[input.length];

  var graph  =  Tasks.source(input)
             => task flip
             => Tasks.sink(result);

  graph.finish();

  return result;
}
```



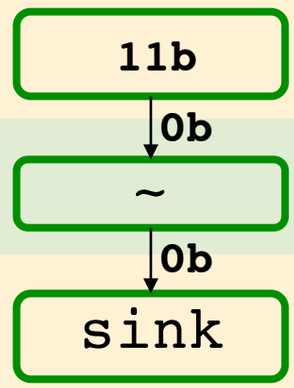
A Lime Example - task, =>

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

  var result = new bit[input.length];

  var graph  =  Tasks.source(input)
             => task flip
             => Tasks.sink(result);

  graph.finish();

  return result;
}
```



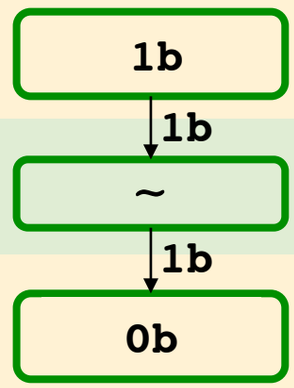
A Lime Example - task, =>

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

  var result = new bit[input.length];

  var graph  =  Tasks.source(input)
             => task flip
             => Tasks.sink(result);

  graph.finish();

  return result;
}
```



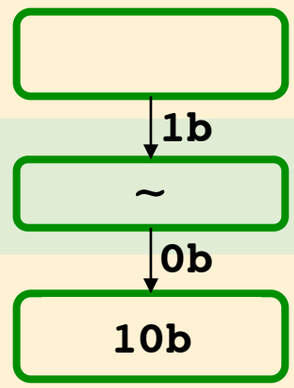
A Lime Example - task, =>

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

  var result = new bit[input.length];

  var graph  =  Tasks.source(input)
             => task flip
             => Tasks.sink(result);

  graph.finish();

  return result;
}
```



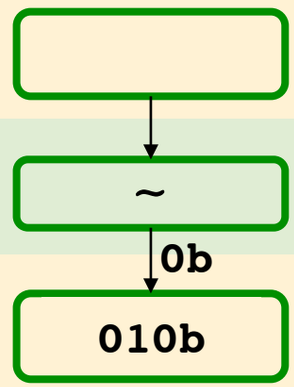
A Lime Example - task, =>

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

  var result = new bit[input.length];

  var graph  =  Tasks.source(input)
             => task flip
             => Tasks.sink(result);

  graph.finish();

  return result;
}
```



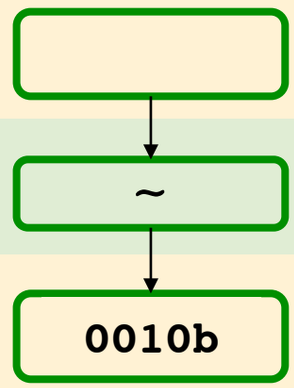
A Lime Example - task, =>

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

  var result = new bit[input.length];

  var graph  =  Tasks.source(input)
             => task flip
             => Tasks.sink(result);

  graph.finish();

  return result;
}
```



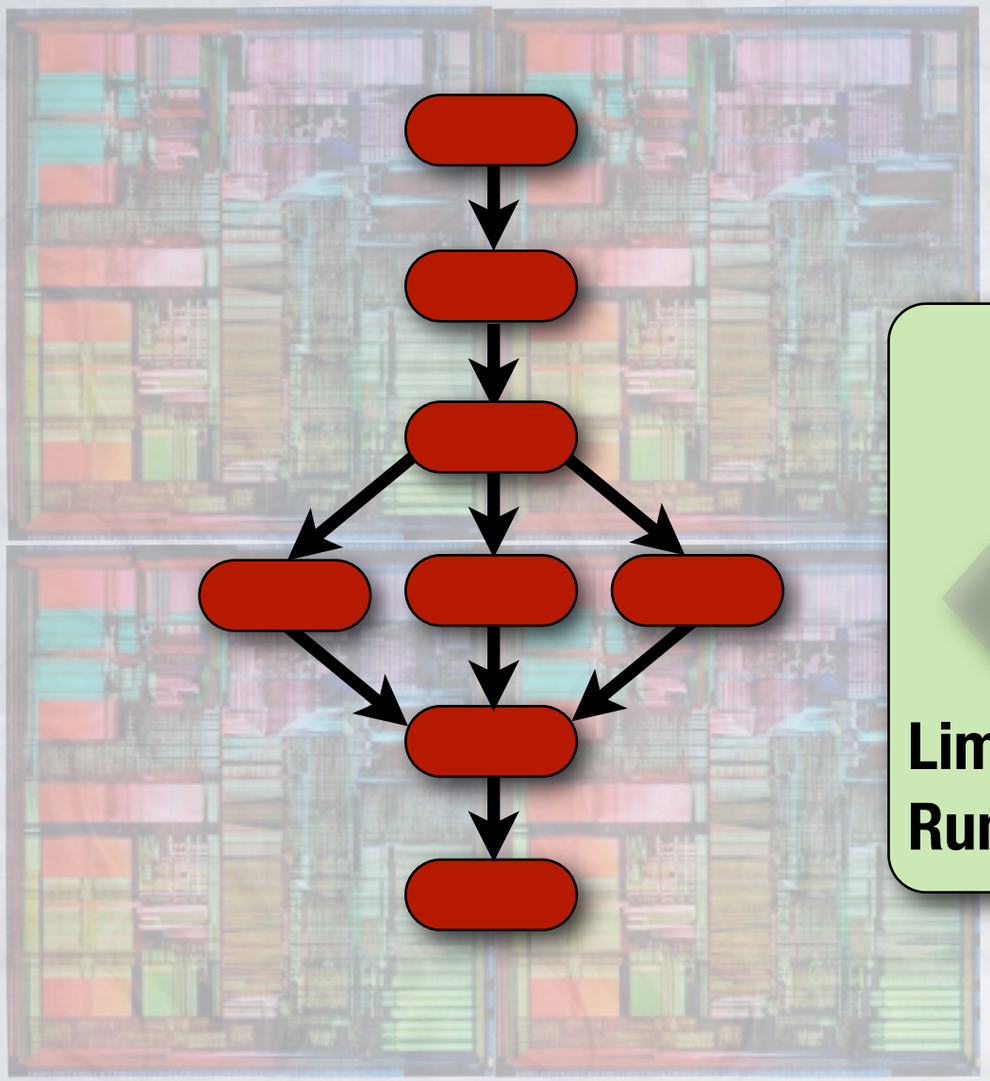
A Lime Example - task, =>

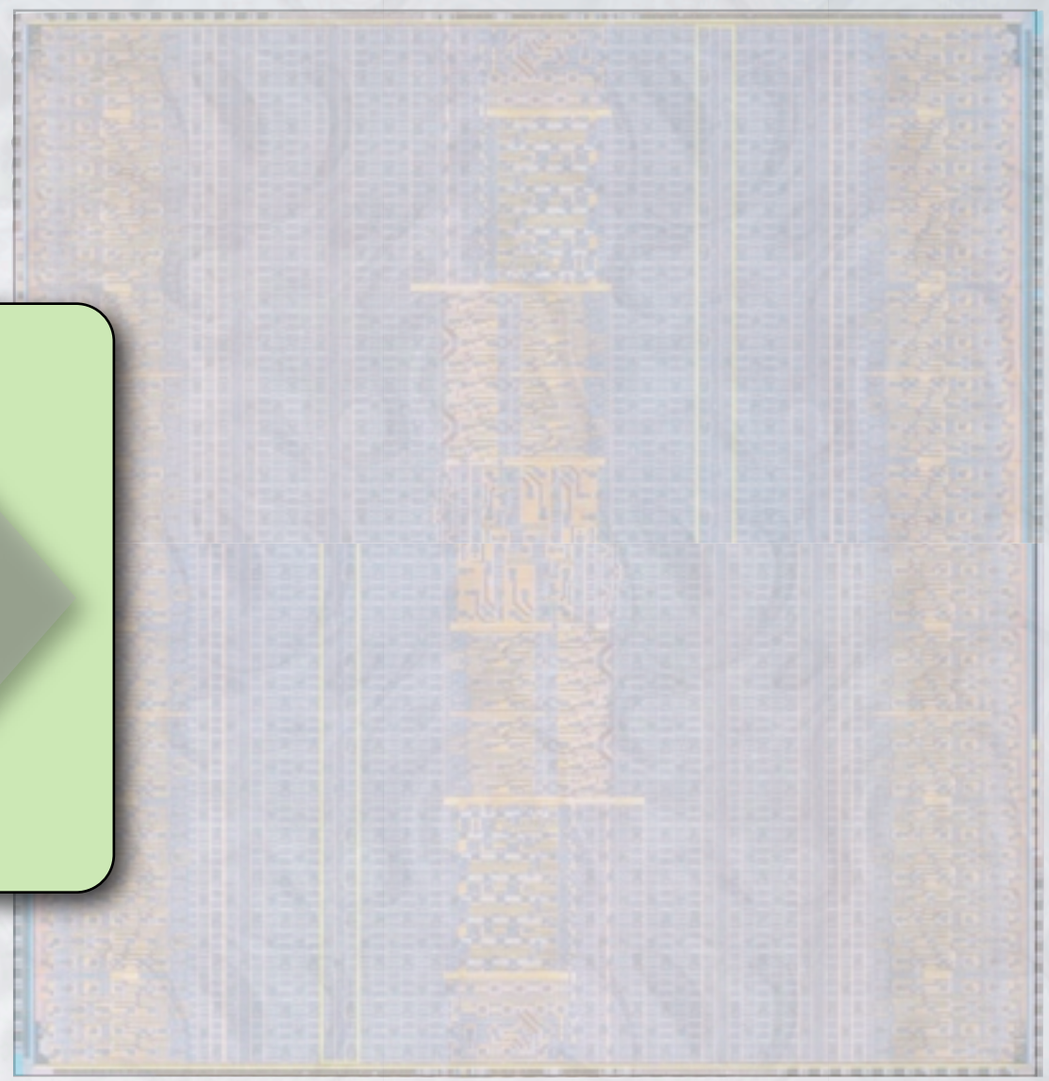
```
static bit[] flipBits(bit[] input) {

  var result = new bit[input.length];

  var graph  =  Tasks.source(input)
             => task flip
             => Tasks.sink(result);

  graph.finish();

  return result;
}
```



Virtualization of Data Movement

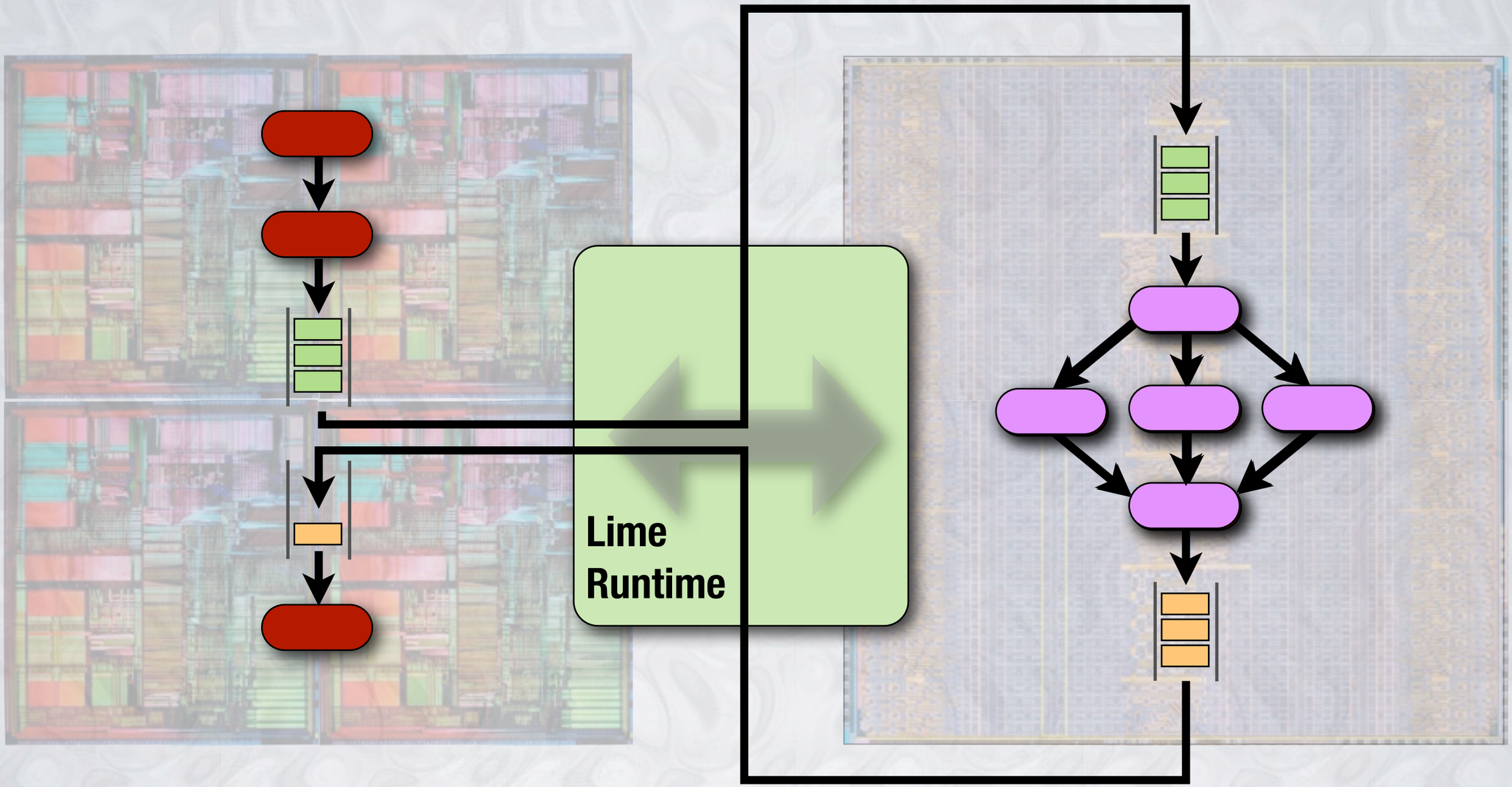
Fluid Task Graph Migration Across Heterogeneous Boundaries



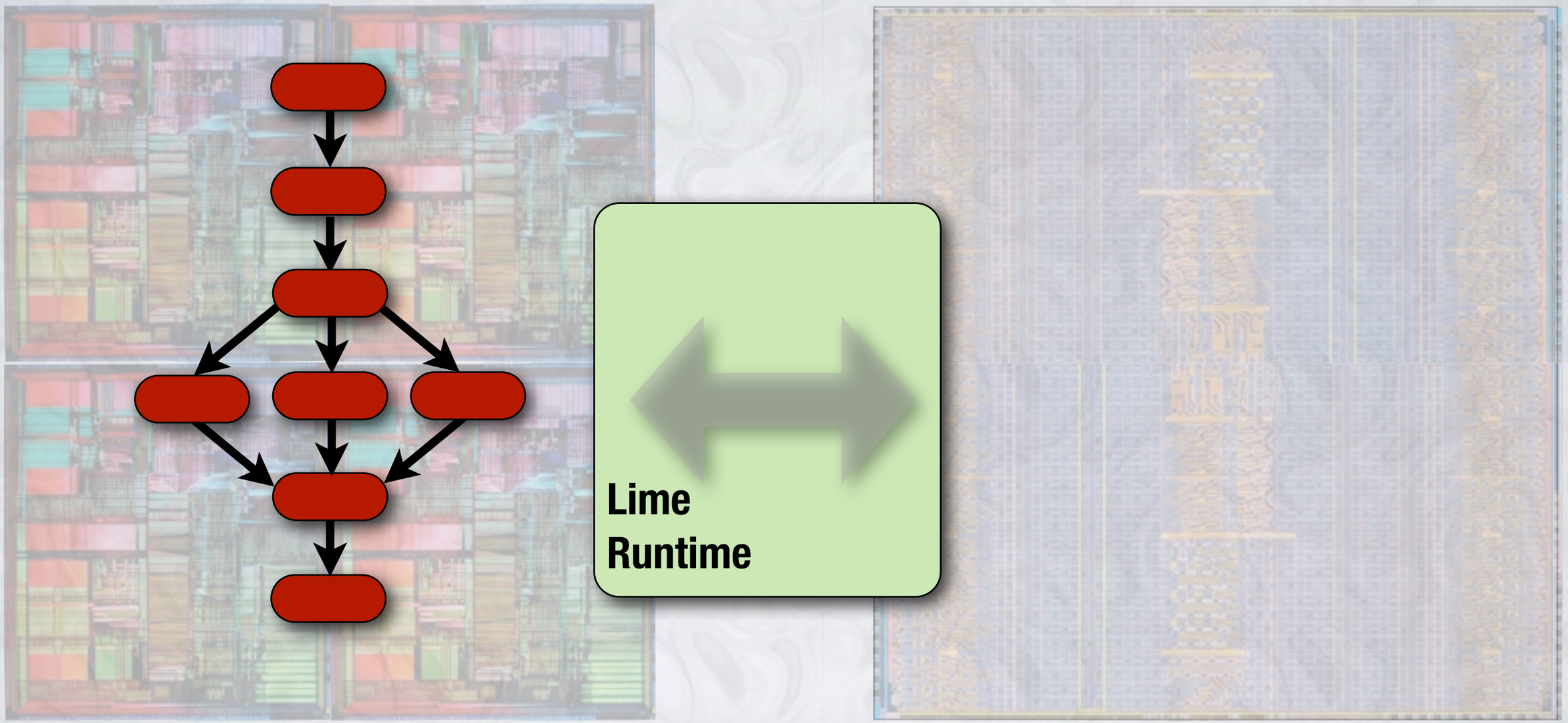


Lime
Runtime

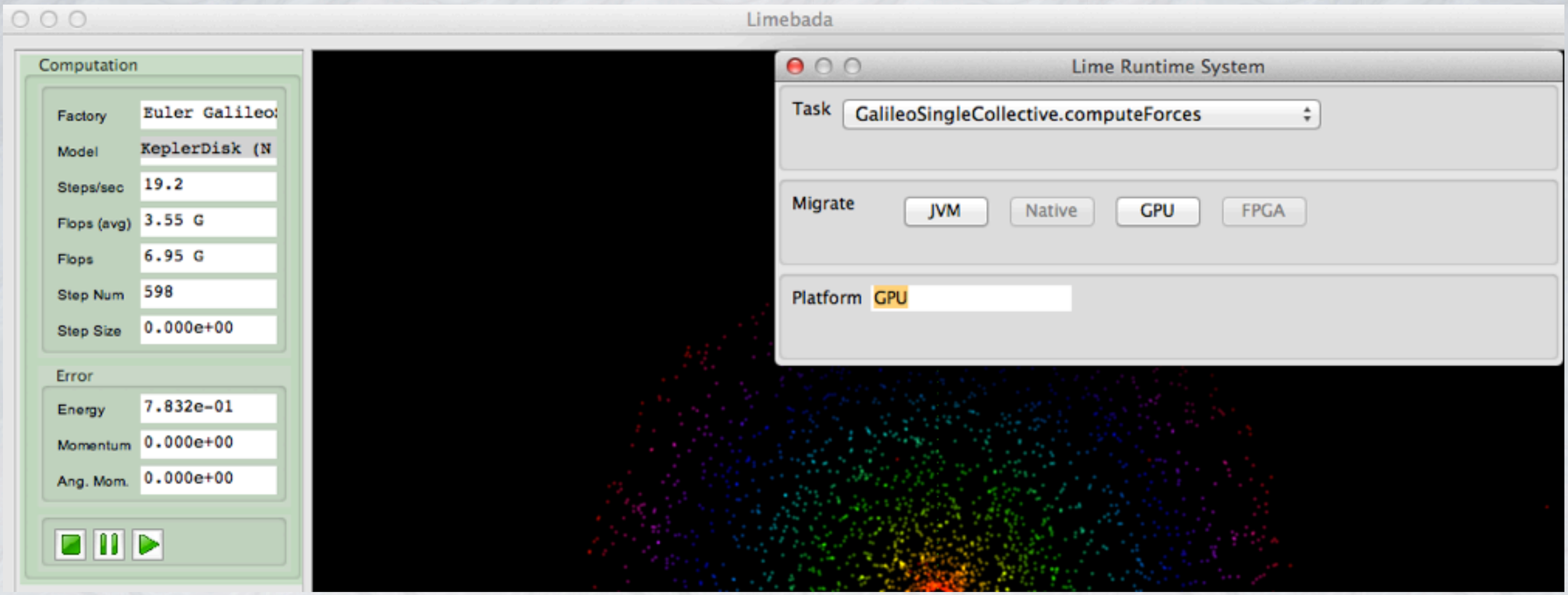
Fluid Task Graph Migration Across Heterogeneous Boundaries



Fluid Task Graph Migration Across Heterogeneous Boundaries



Live Demo



- N-Body simulation, dynamic migration from CPU to GPU
- 7x performance improvement (CPU: 1GFLOP, GPU: 7GFLOP)

THE CODE YOU DON'T HAVE TO WRITE

GalileoSingleAccCalc | EulerIntegrator.java | limebada_calculator_ | limebada_calculator_

```
}

/*
 * Class:     limebada_calculator_OCLAccCalculator
 * Method:    computeAccOCL
 * Signature: (ID[D[D)J
 */
JNIEXPORT jlong JNICALL Java_limebada_calculator_OCLAccCalculator
  (JNIEnv *env, jobject obj, jint deviceIndex, jdouble softenLeng

   struct timeval start, end;

  initializeFIDs(env, obj);

  // Setup various buffers if this is the first time we enter her
  // Note that we are using our own copy via Java-hosted field.
  int numParticles = env->GetArrayLength(partData) / 4;
  cl_float4 * positions = (cl_float4 *) env->GetLongField(obj, po
  float* accs = (float *) env->GetLongField(obj, accMemFID);
  Compute_t *compute = (Compute_t*) env->GetLongField(obj, oclInf

  if (positions == NULL) {
    // It needs to be big enough to contain doubles as we unpack.
    int positionSpace = 4 * numParticles * sizeof(double);
    int accSpace = 3 * numParticles * sizeof(double);  // enough
    positions = (cl_float4 *) malloc(positionSpace);
    accs = (float *) malloc(accSpace);
    compute = setupCompute(numParticles, deviceIndex);
    env->SetLongField(obj, posMemFID, (long) positions);
    env->SetLongField(obj, accMemFID, (long) accs);
    env->SetLongField(obj, oclInfoFID, (long) compute);
  }


  gettimeofday(&start, NULL);
  // Copy data over and then squeeze it down to float
  env->GetDoubleArrayRegion(partData, 0, 4 * numParticles, (doubl
  for (int i=0; i<4*numParticles; i++) {
     ((cl_float*)positions)[i] = (cl_float) ((double *)positions
  }
  gettimeofday(&end, NULL);
  marhsalTime += ((end.tv_sec * 1000000 + end.tv_usec) - (start.t

  float softenLenSquared = softenLength * softenLength;
  calcAcc(compute, numParticles, positions, accs, softenLenSquare

  cl_ulong startTime;
  cl_ulong endTime;

  int retCode = clGetEventProfilingInfo(kernelComputationEvent, (
  checkError(retCode, "clGetEventProfilingInfo");
  retCode = clGetEventProfilingInfo(kernelComputationEvent, CL_PR
  checkError(retCode, "clGetEventProfilingInfo");

  kernelTime += endTime - startTime;

  retCode = clGetEventProfilingInfo(firstTransferEvent, CL_PROFIL
  checkError(retCode, "clGetEventProfilingInfo");
  retCode = clGetEventProfilingInfo(lastTransferEvent, CL_PROFILI
  checkError(retCode, "clGetEventProfilingInfo");

  totalOclTime += endTime - startTime;

  calledNUm++;
```

50 lines of host code

GalileoSingleAccCalc | EulerIntegrator.java | limebada_calculator_

```
// Performance notes:
// (1) rsqrt seems just as fast as native_sqrt
// (2) Using float4 throughout will cause extra
static const char *Source =
"float4 calcAccHelp(float softLenSq, float4 acc,
"{
"       // 3 flops
"       float dx = pj.x - pi.x;
"       float dy = pj.y - pi.y;
"       float dz = pj.z - pi.z;
"       // 10 flops
"       float d2 = dx*dx + dy*dy + dz*dz + softl
"       float id = rsqrt(d2);
"       float f = pj.w*id*id*id;
"       // 6 flops
"       acc.x += f * dx;
"       acc.y += f * dy;
"       acc.z += f * dz;
"       return acc;
"}
"
"__kernel void calcAccKernel(
"    __global float4* positions,
"    __global float4* accs,
"    __local float4* localPos,
"    float softLenSq) {
"
"    int gi = get_global_id(0);
"    int ti = get_local_id(0);
"    int gs = get_global_size(0);
"    int ls = get_local_size(0);
"    int blks = gs / ls;
"    float4 pi = positions[gi];
"    float4 acc = {0,0,0,0};
"    int baseIndex = 0;
"    for (int b=0; b<blks; b++) {
"      localPos[ti] = positions[baseIndex+ti];
"      barrier(CLK_LOCAL_MEM_FENCE);
"      for (int j=0; j<ls; j++) {
"        float4 pj = localPos[j];
"        acc = calcAccHelp(softLenSq, acc, pi, pj
"      }
"      baseIndex += ls;
"      barrier(CLK_LOCAL_MEM_FENCE);
"    }
"    accs[gi] = acc;
"}
  ;
```

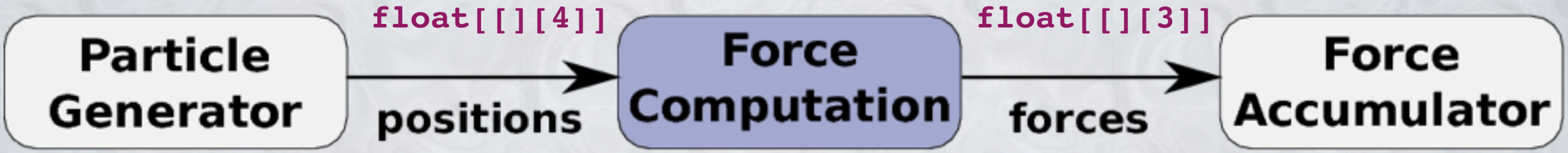
40 lines of OCL kernel code

GalileoSingleAccCalc | EulerIntegrator.java | limebada_calculator_ | limebada_calculator_

```
static Compute_t* setupCompute(int N, int deviceIndex) {
  Compute_t* compute = (Compute_t*) malloc(sizeof(Compute_t));
  // Setup device, compile kernel, and create communication buffers
  compute->info = setupGPU(0, deviceIndex, Source,  "calcAccKernel");
  compute->input = makeBuffer(compute->info, CL_MEM_READ_ONLY,  sizeof(cl_fl
  compute->output = makeBuffer(compute->info, CL_MEM_WRITE_ONLY,  sizeof(cl_
  return compute;
}

static void calcAcc(Compute_t *comp, int N, cl_float4 *pos, cl_float *accs,

    // We can't let softenLen be zero or else we can't let a body self-inter
    // This is not softening in the sense of preventing close-body encounter
    if (softenLenSquared == 0)
        softenLenSquared = 1e-24f; // will compute -1.5 power of this

    int err;
    cl_command_queue queue = comp->info->queue;
    cl_mem input = comp->input;
    cl_mem output = comp->output;
    cl_kernel kernel = comp->info->kernel;

    err = clEnqueueWriteBuffer(queue, input, CL_TRUE, 0, sizeof(cl_float4) *
    checkError(err, "clEnqueueWriteBuffer");

    err  = clSetKernelArg(kernel, 0, sizeof(cl_mem), &input);
    err |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &output);
    err |= clSetKernelArg(kernel, 2, comp->info->maxWorkGroupSize * sizeof(c
    err |= clSetKernelArg(kernel, 3, sizeof(float), &softenLenSquared);
    checkError(err, "clSetKernelArg");



    // one-dimensional
    size_t global = N;
    size_t local = 1;
    err = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &global, NULL, 0, N
    checkError(err, "clEnqueueNDRangeKernel");

    err = clEnqueueReadBuffer(queue, output, CL_TRUE, 0, sizeof(cl_float4) *
    checkError(err, "clEnqueueReadBuffer");

    err = clFinish(queue);
    checkError(err, "clFinish");
}
```

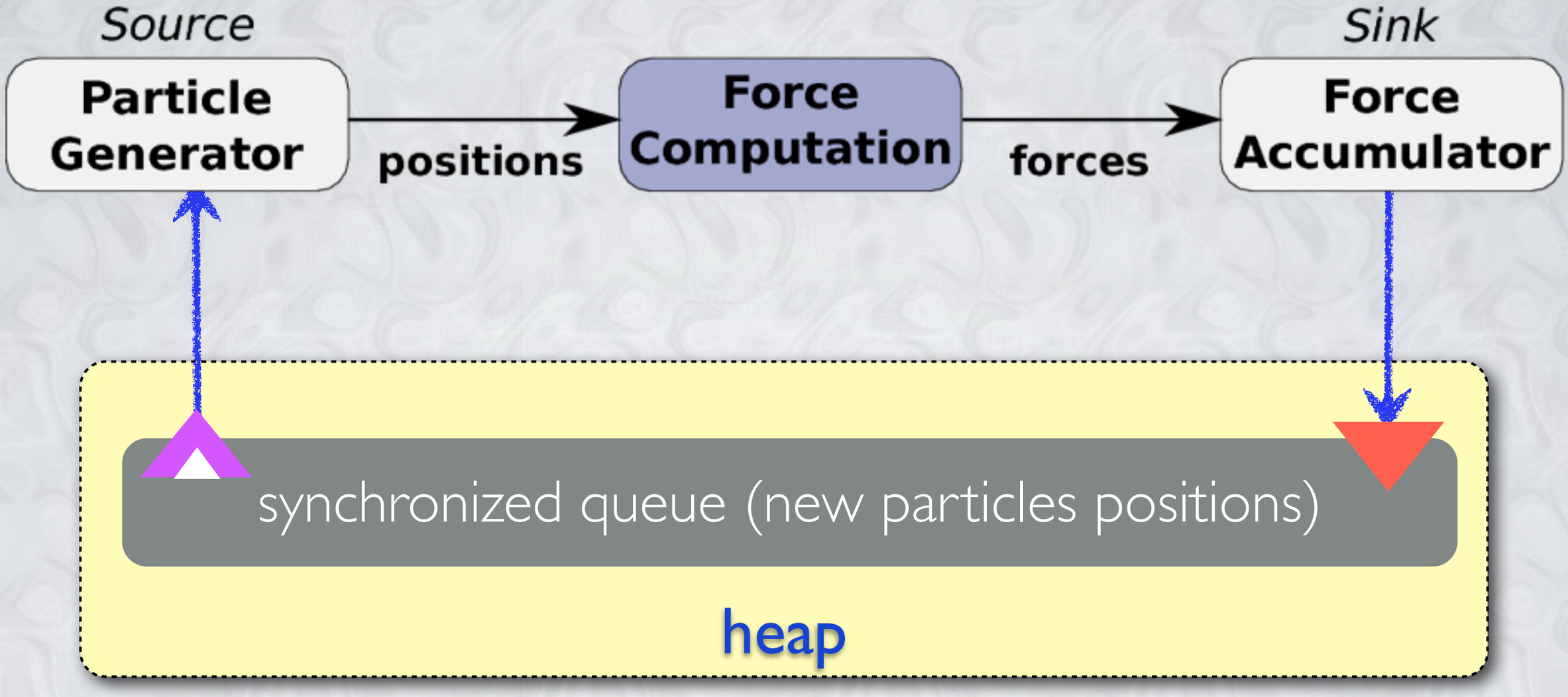
40 lines of OCL driver code

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; }
  ...
}
```

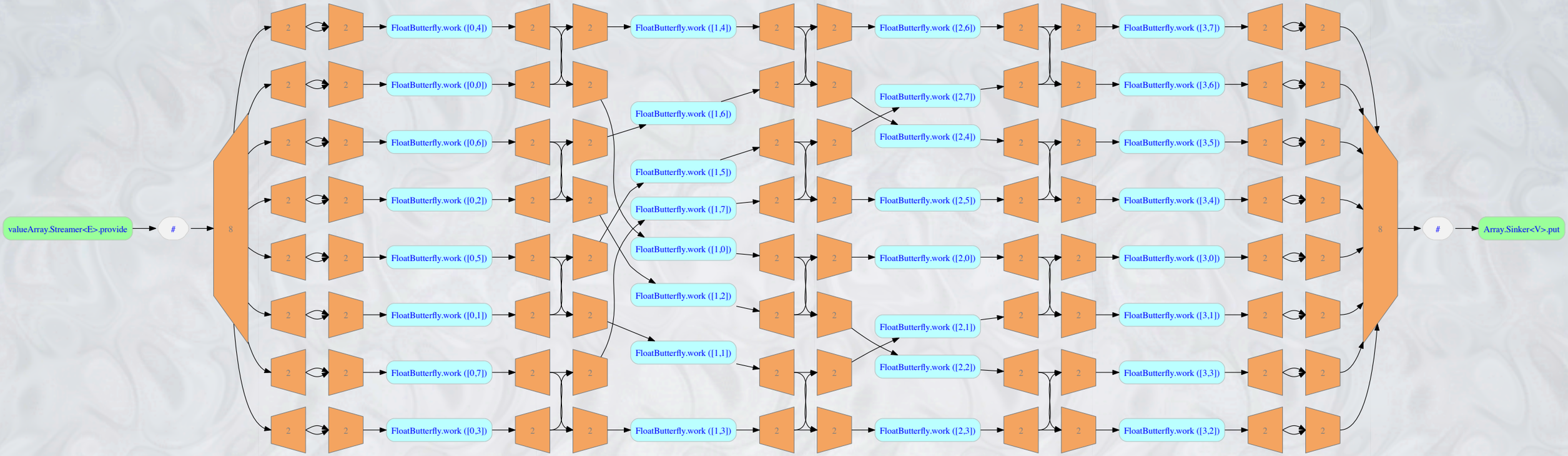
Source and Sink Tasks



- **May have globally observable side-effects (I/O)**

Graph Construction

- Reducible and irreducible graphs



- ([...]) ensure graph can be statically elaborated
- “Compiling Complex Stream Graphs to Reconfigurable Hardware in Lime”, Auerbach et al. in ECOOP 2013

Common Graph Idioms

- Recursive pipeline
- Divide and conquer
- Repeat and map
- Butterfly network
- Systolic array
- Reduction tree

IDIOM EXAMPLES

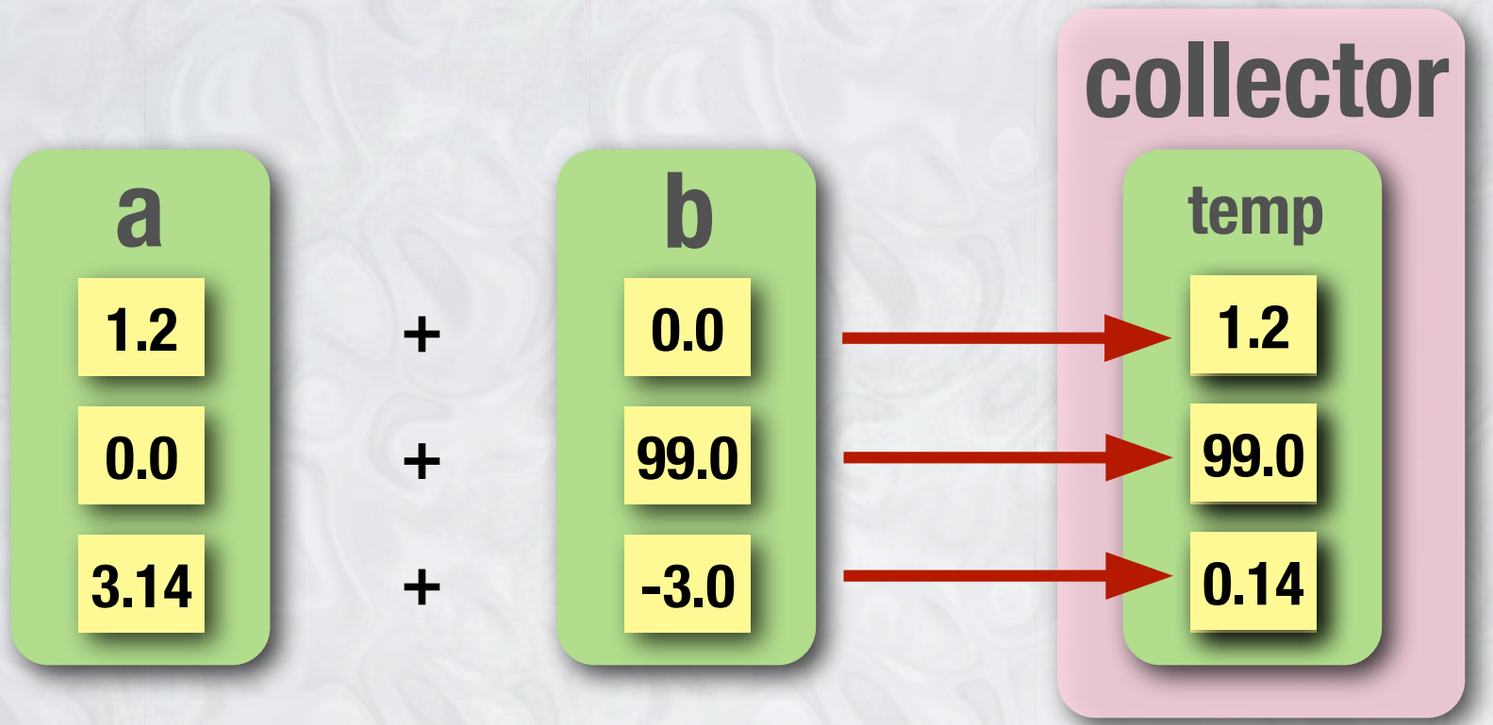
```
25    public class Idioms {
26        static task <V extends Value> LFilter<V,V> pipeline(LFilter<V,V>[[]]filters) {
27            var pipe = filters[0];
28            for (int i = 0; i < filters.length; i++)
29                pipe = pipe => filters[i];
30                return pipe;
31        }
32
33        static task <V extends Value, N extends ordinal<N>> Task dnc(Task t) {
34            final HALF = N.size/2; if (N.size > 2)
35                return (V #) =>
36            task split V[[2]] =>
37            task [ Idioms.<V, HALF>dnc(t), Idioms.<V, HALF>dnc(t) ] => task join V[[2]] =>
38            (# V[[N]]) => t => (# V);
39            else return t; }
40    }
```

MAP/REDUCE OPERATIONS

DATA PARALLELISM

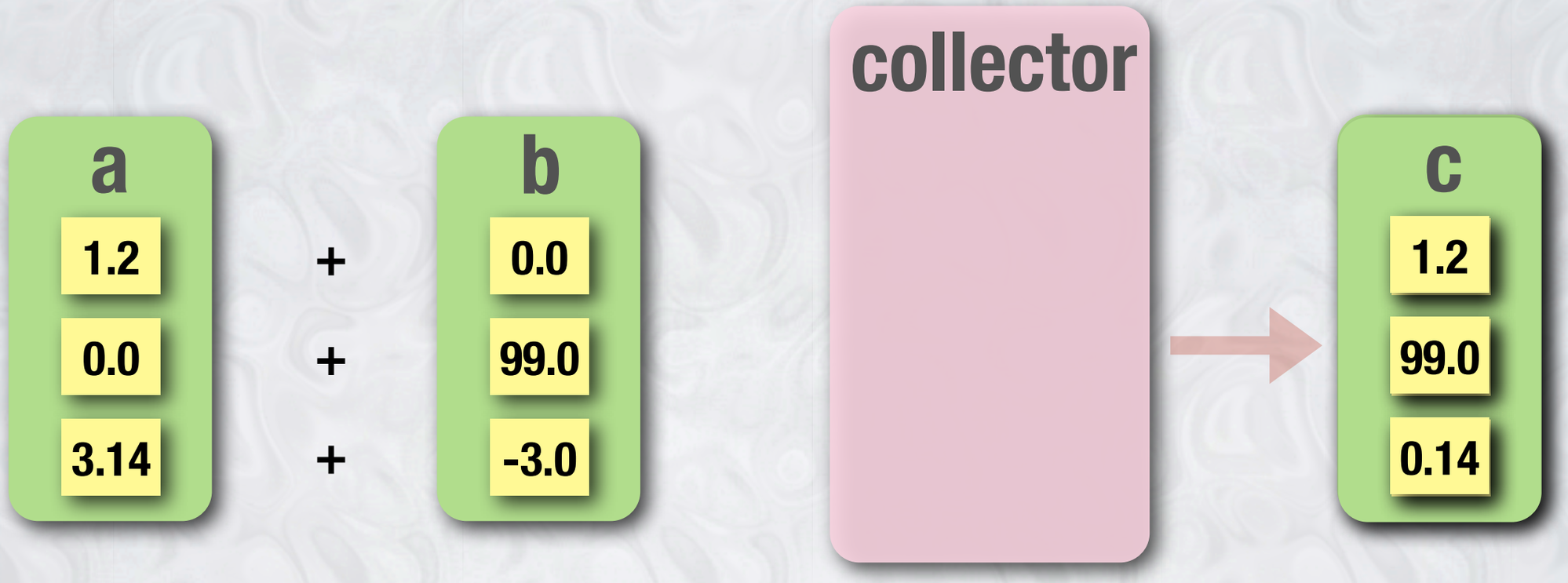
ARRAY PARALLELISM

```
float[ ] a = ...;
float[ ] b = ...;
float[ ] c = a @+ b;
```



ARRAY PARALLELISM

```
float[ ] a = ...;
float[ ] b = ...;
float[ ] c = a @+ b;
```



STREAM TRIAD

Given: m-element vectors A, B, C
Compute: ∀i∈1..m, Ai=Bi+αCi

```
double[m] B = {...};
double[m] C = {...};
double    α =  ...  ;

var A = B @+ (C @* #α);
```

NBody Example

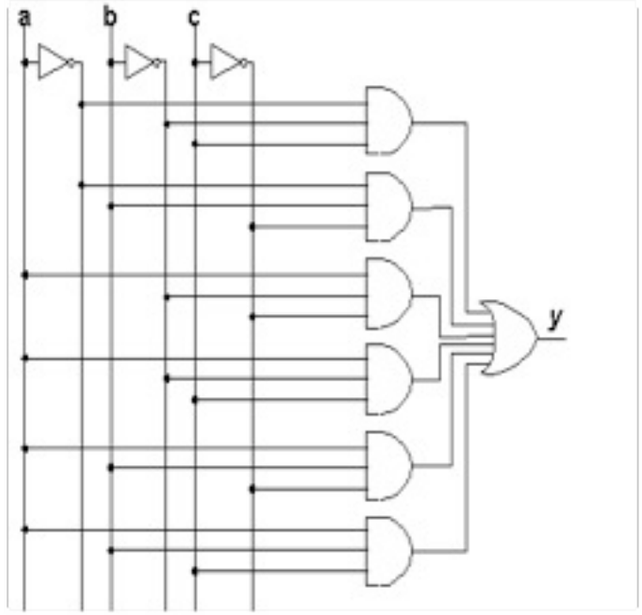
```
static local float[[][3]] forceComputation(float[[][4]] particles) {
   return @force(particles, #particles);
}

static local float[[3]] force(float[[4]] p, float[[][4]] P) {
  float fx = 0, fy = 0, fz = 0;
  for (var Pj : P) {
    var dx = Pj[0] - p[0]; float dy = Pj[1] - p[1]; float dz = ...;
    var F = ...; // scalar code, compute force between p and Pj
    fx += dx + F; fy += dy + F; fz += dz + F;
  }
  return new float[[]] { fx, fy, fz };
}
```

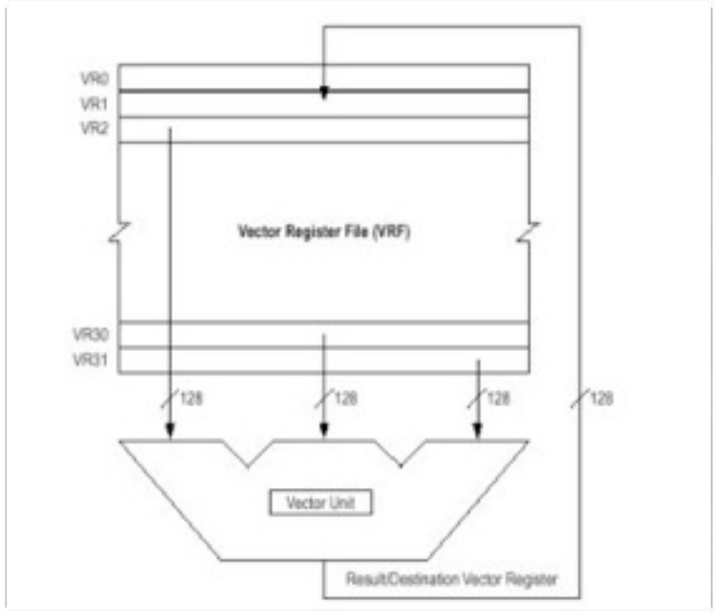
INFERRING PARALLELISM

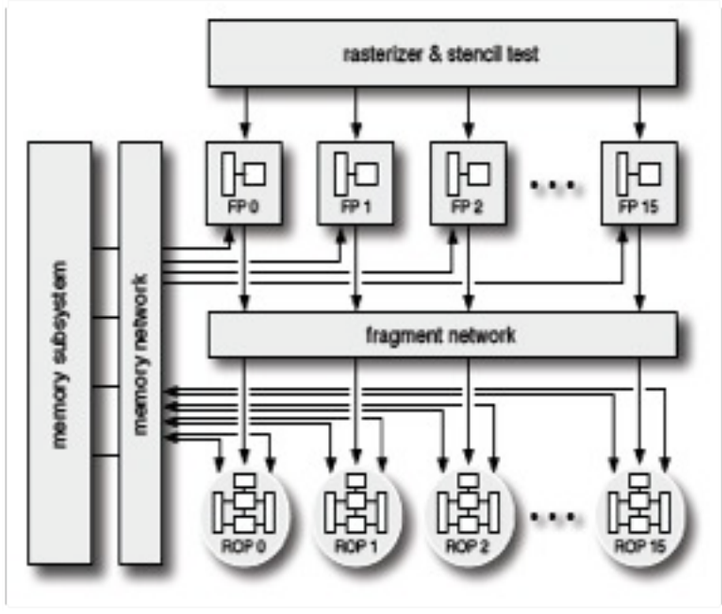
```
PURE FUNCTION = LOCAL + VALUE
```

Data Parallel Operators

@ !

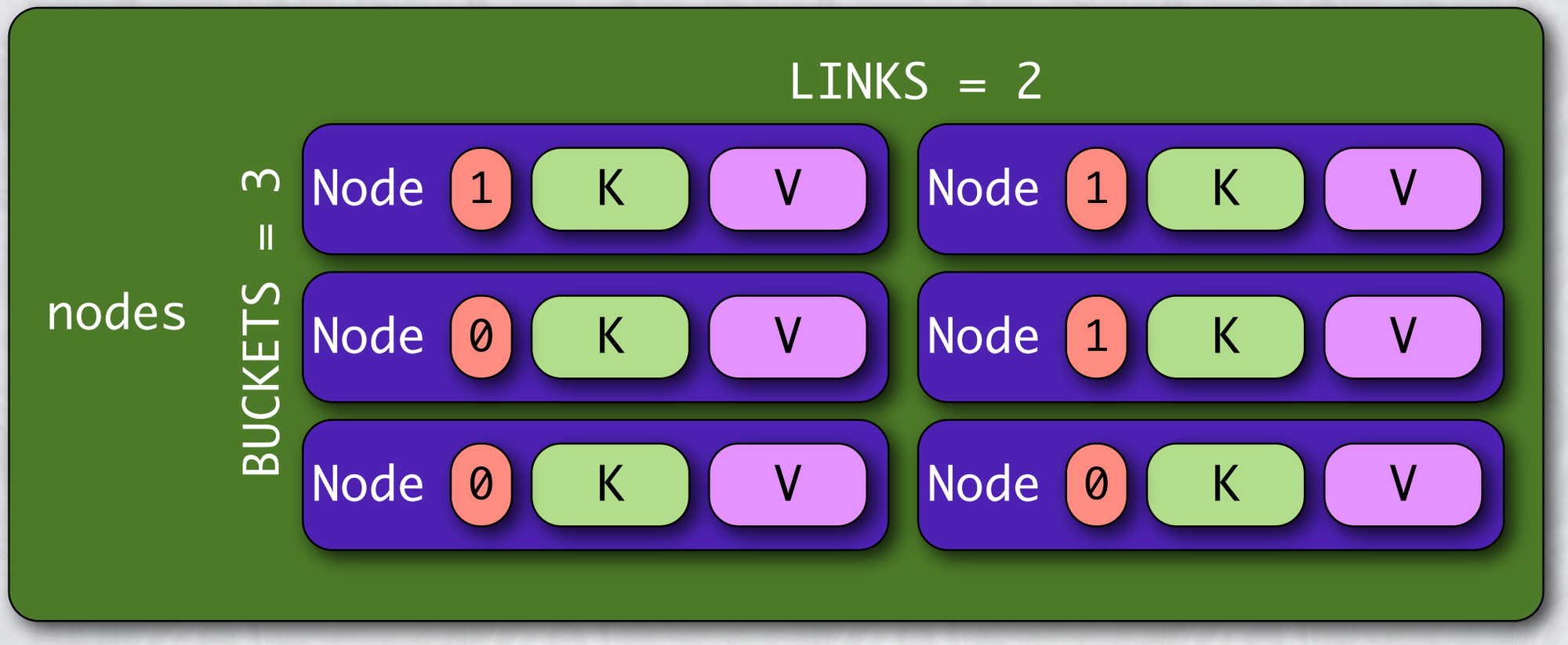






Map/Reduce in Action

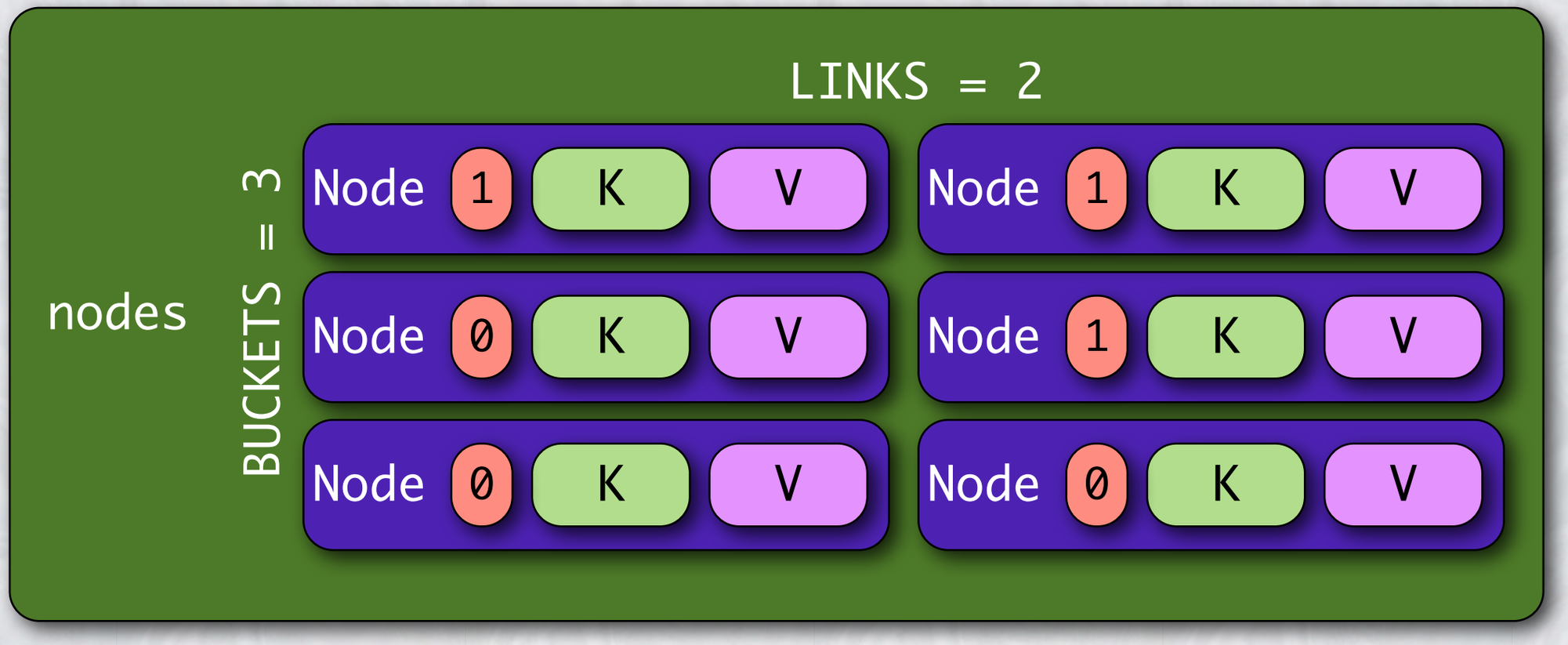
```
package lime.util.synthesizable;

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 @ getValue(selections);
   return | ! vals;
}
```

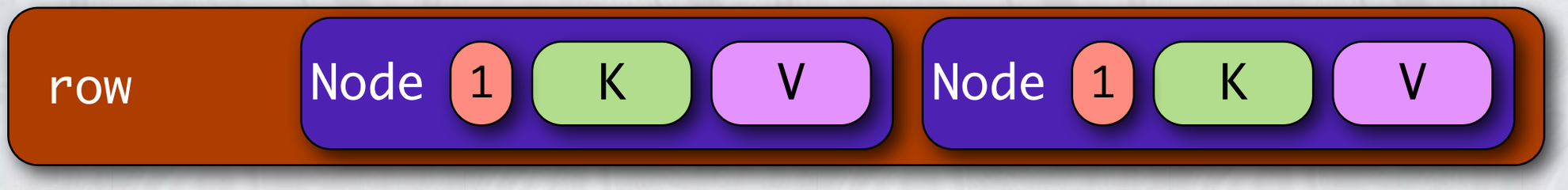
key



GET OPERATION, STEP 1: SELECT ROW

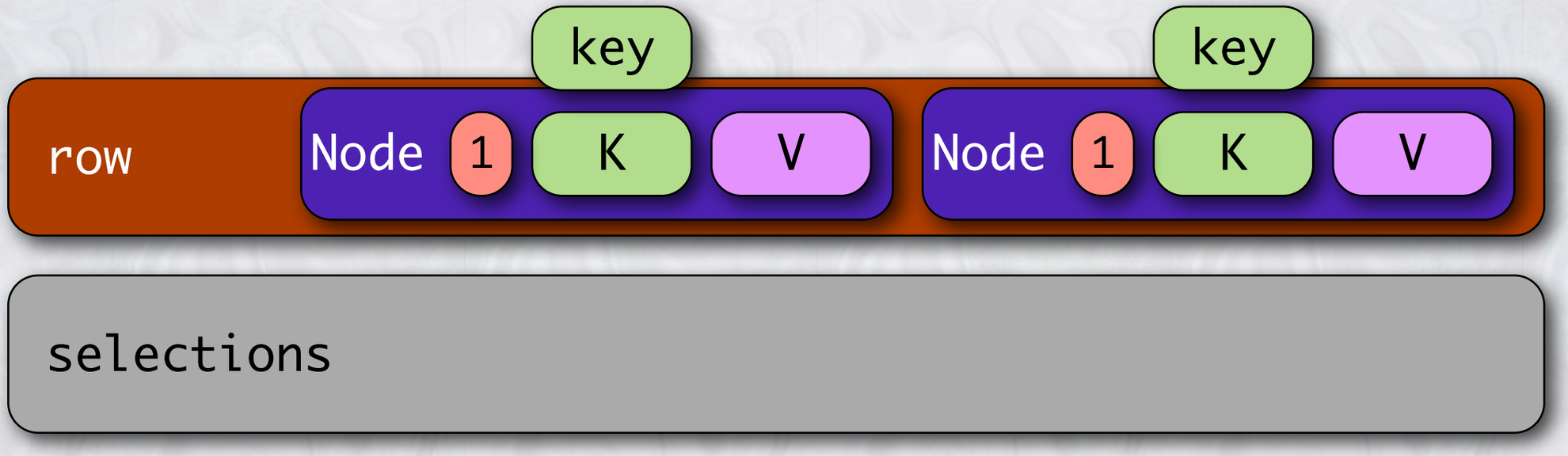
```
public local V get(K key) {
   Node[LINKS] row = nodes[hash(key)];
   boolean[LINKS] selections = row @ compareKey(key);
   V[LINKS] vals = row @ getValue(selections);
   return | ! vals;
}
```

key



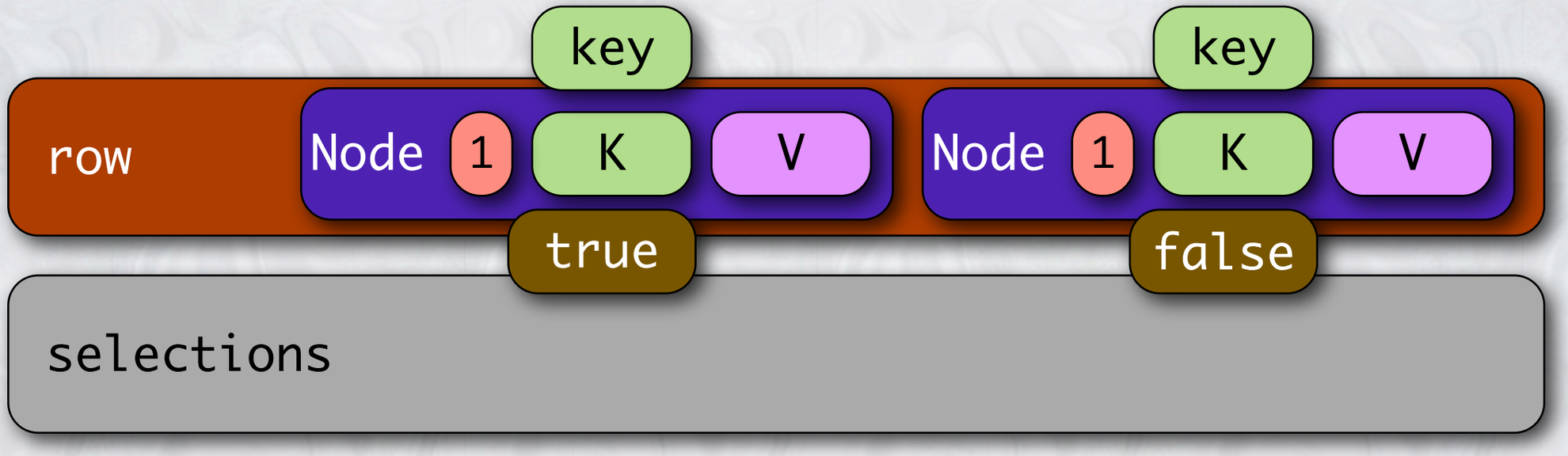
Step 2: Map - Compare All Keys

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



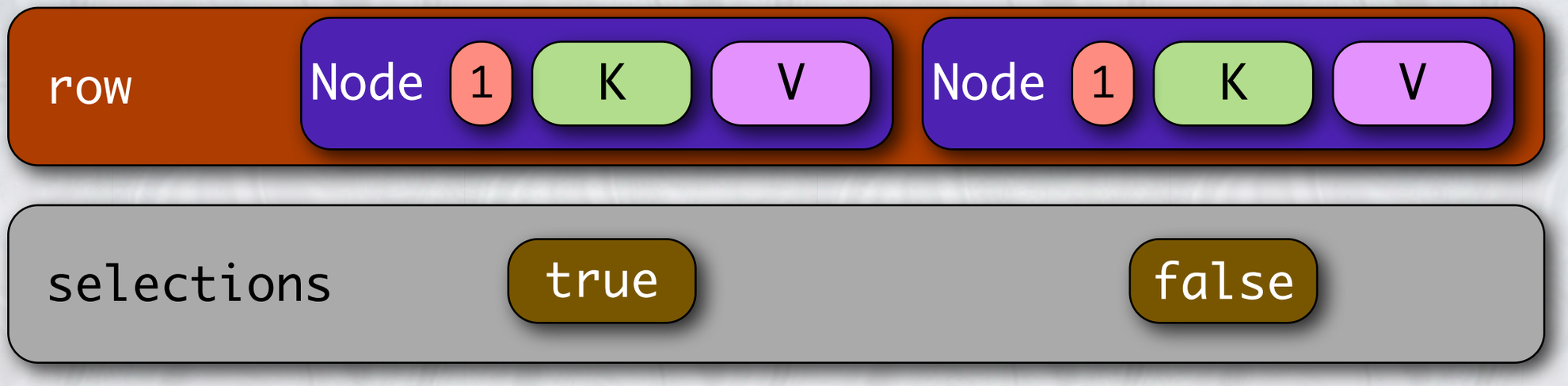
Step 2: Map - Compare All Keys

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



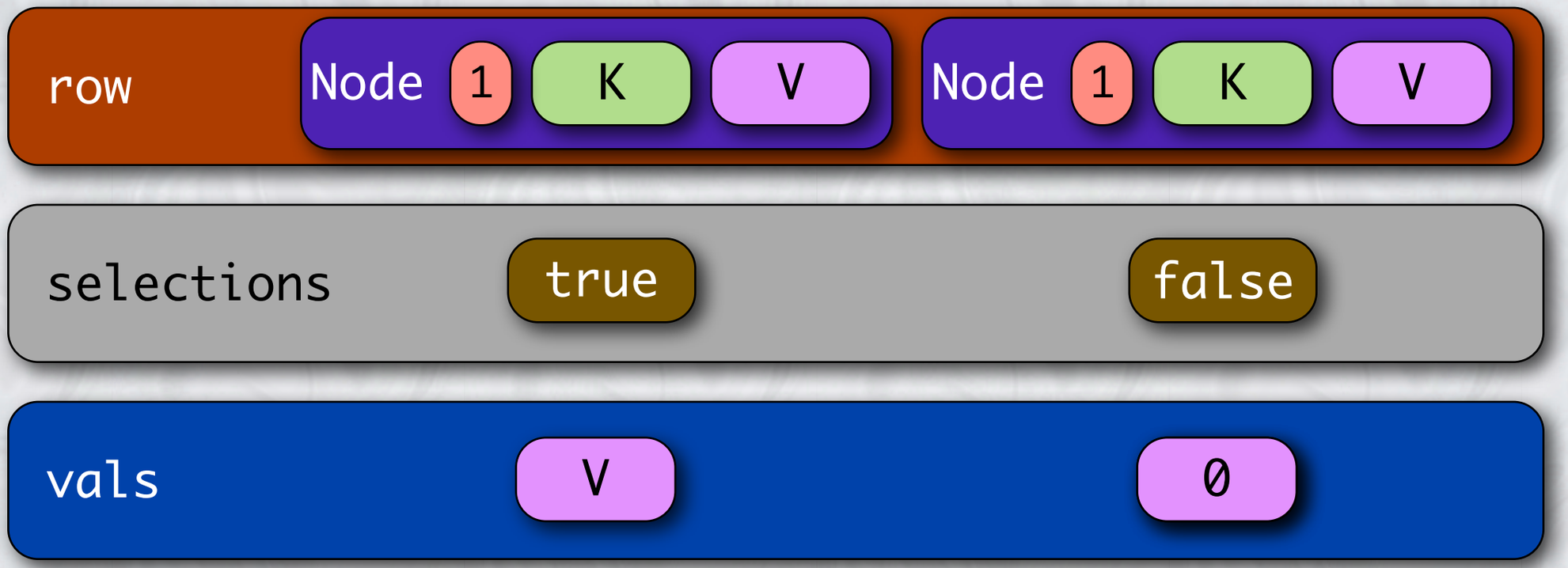
Step 2: Map - Compare All Keys

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



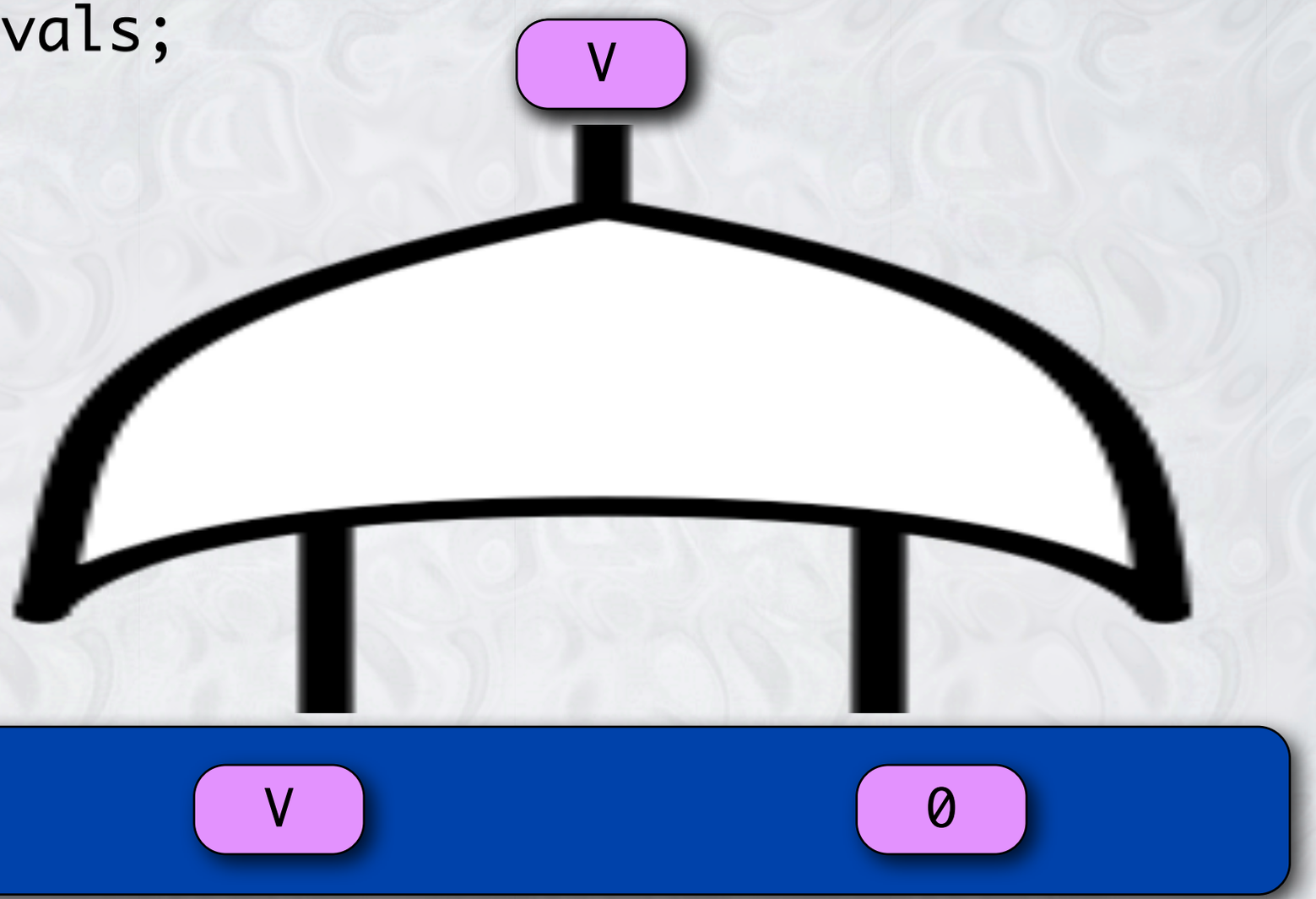
Step 3: Map - Get Values/Zeros

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



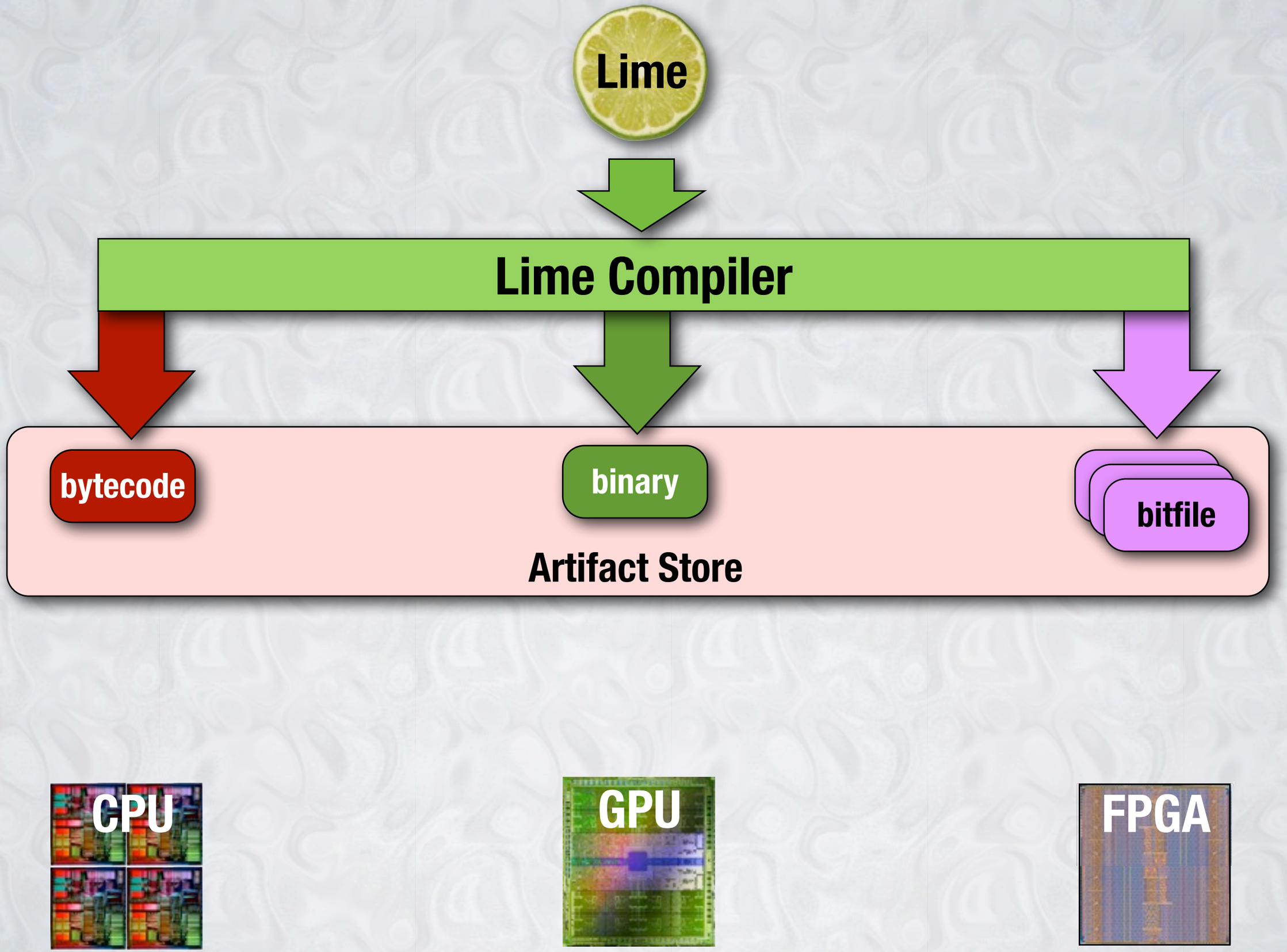
Step 4: Or-Reduce For Result

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

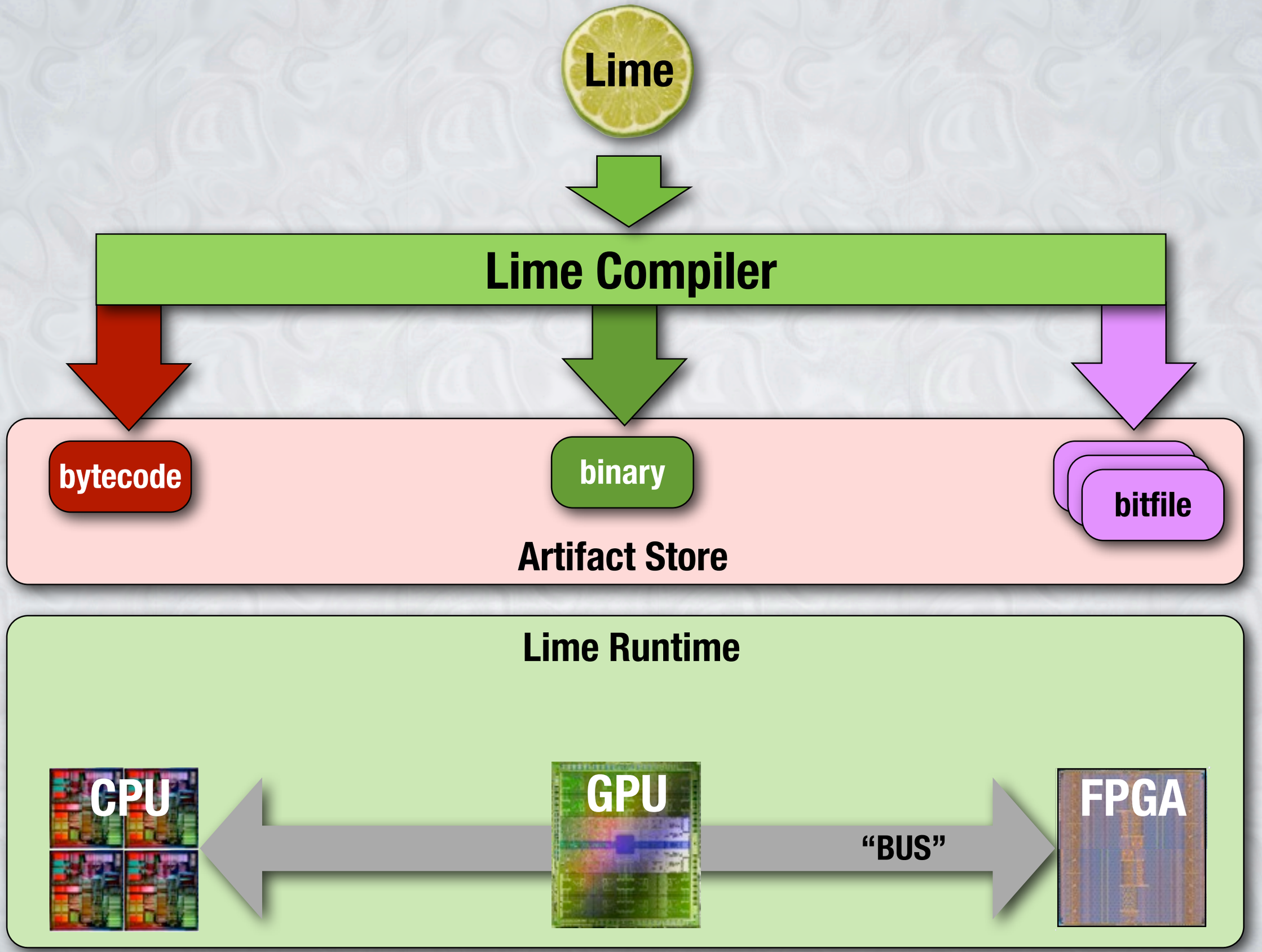


LIME COMPILER

Lime Artifact Store

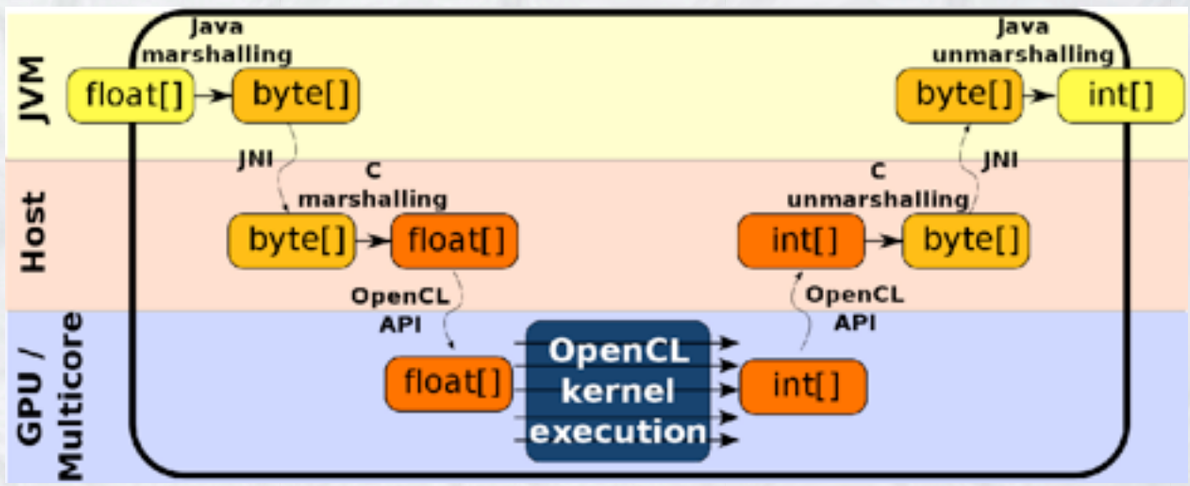


Lime Virtual Machine



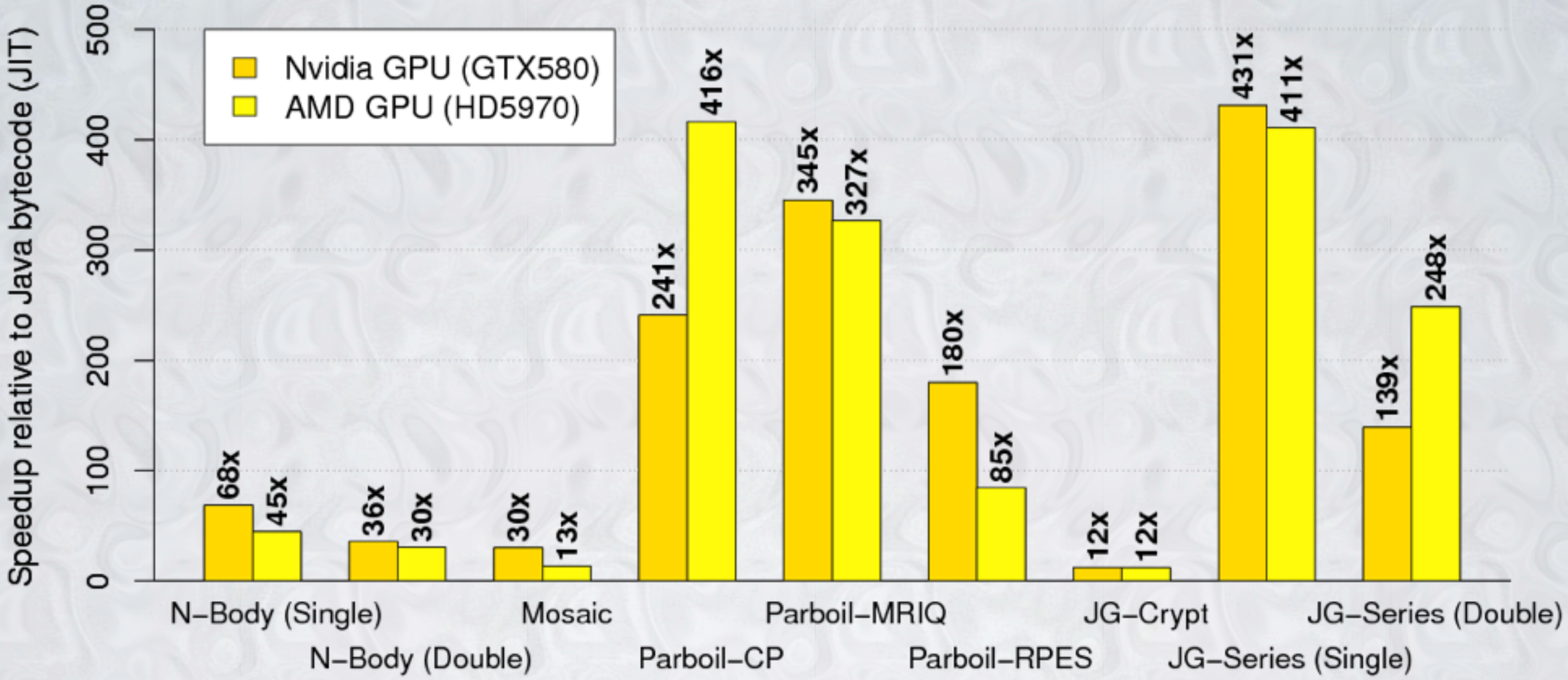
GPU Backend

- Handles large subset of the language (unbounded arrays, objects, GC)
- Generates C++ and OpenCL
- Lime map operations are compiled to OpenCL kernels
- Orchestrates all communication
- DLL as artifact



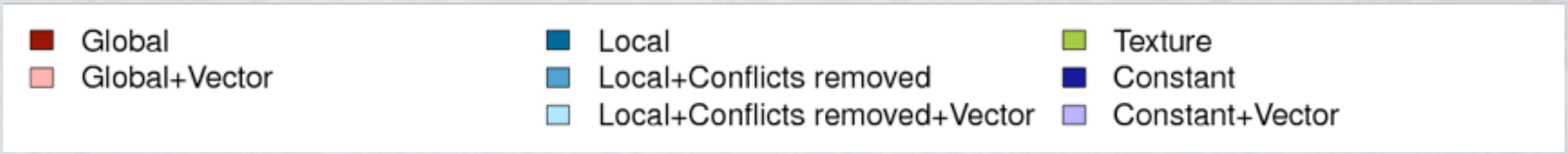
nbody flow of data from JVM to GPU

SPEEDUP VS. JAVA (JIT) [PLDI '12]



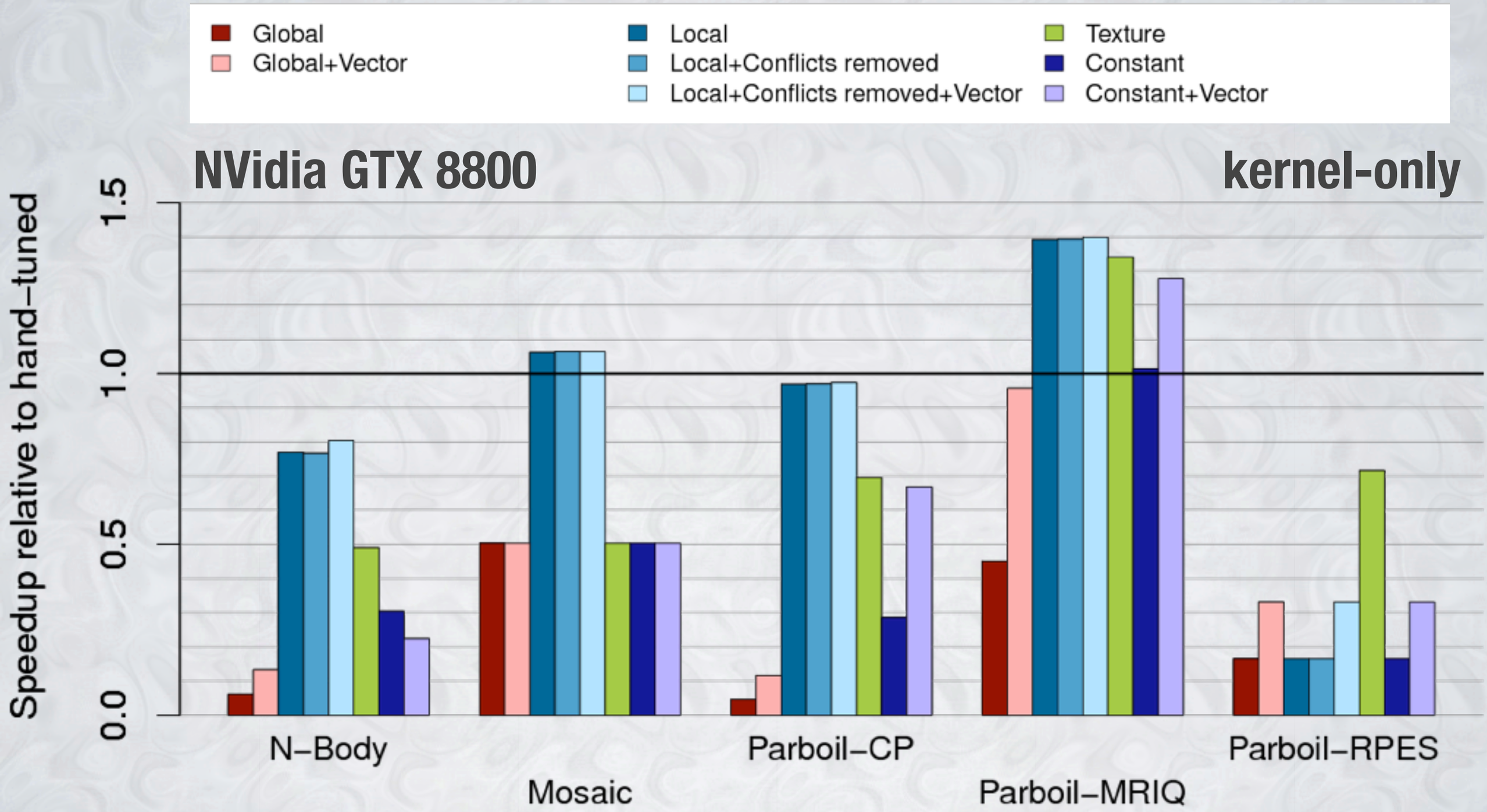
end-to-end performance (all overhead included)

COMPARED TO TUNED OPENCL [PLDI '12]

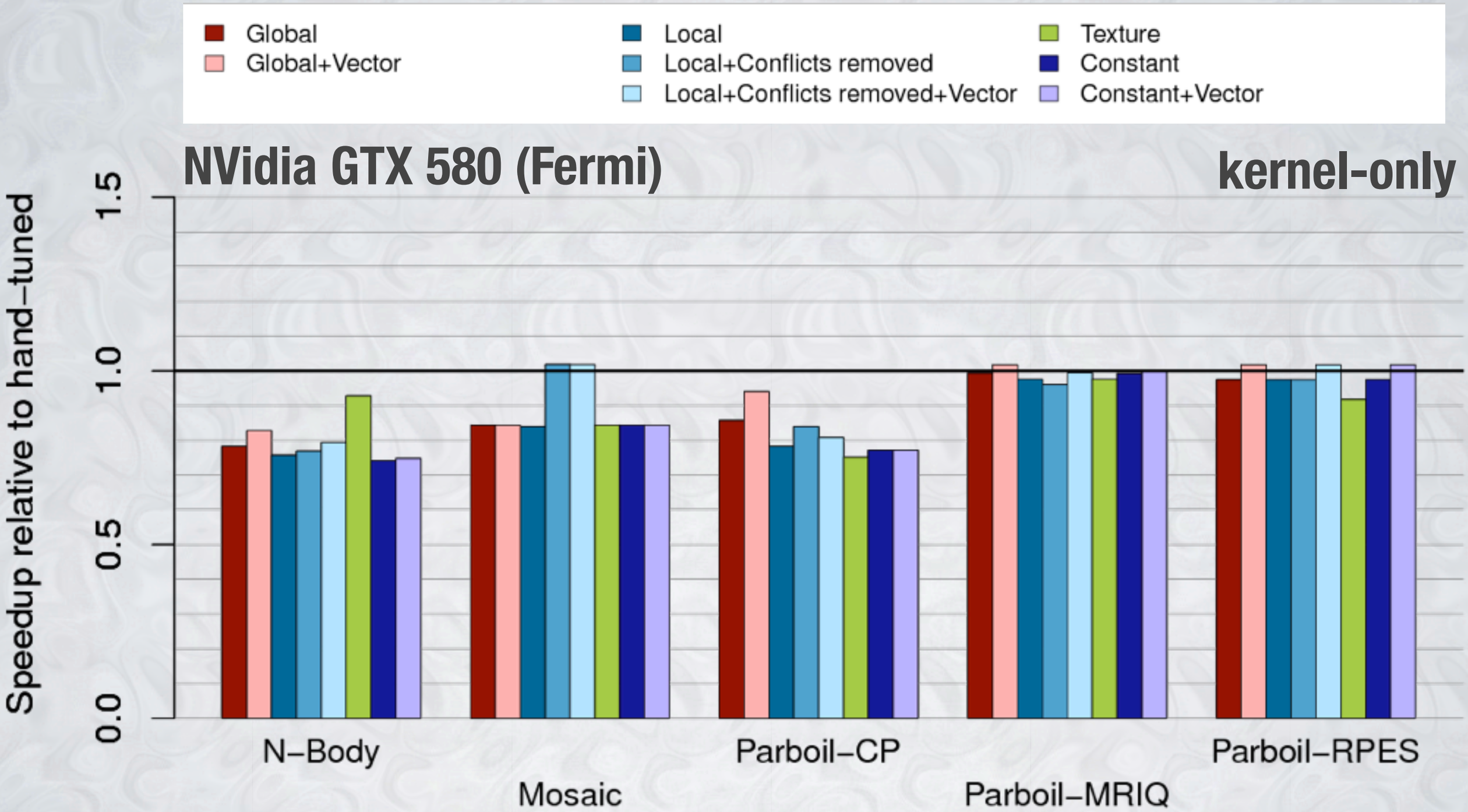


- Simple compiler analysis
- def-use idioms directs data location
- SIMD from @ over bounded arrays of small size

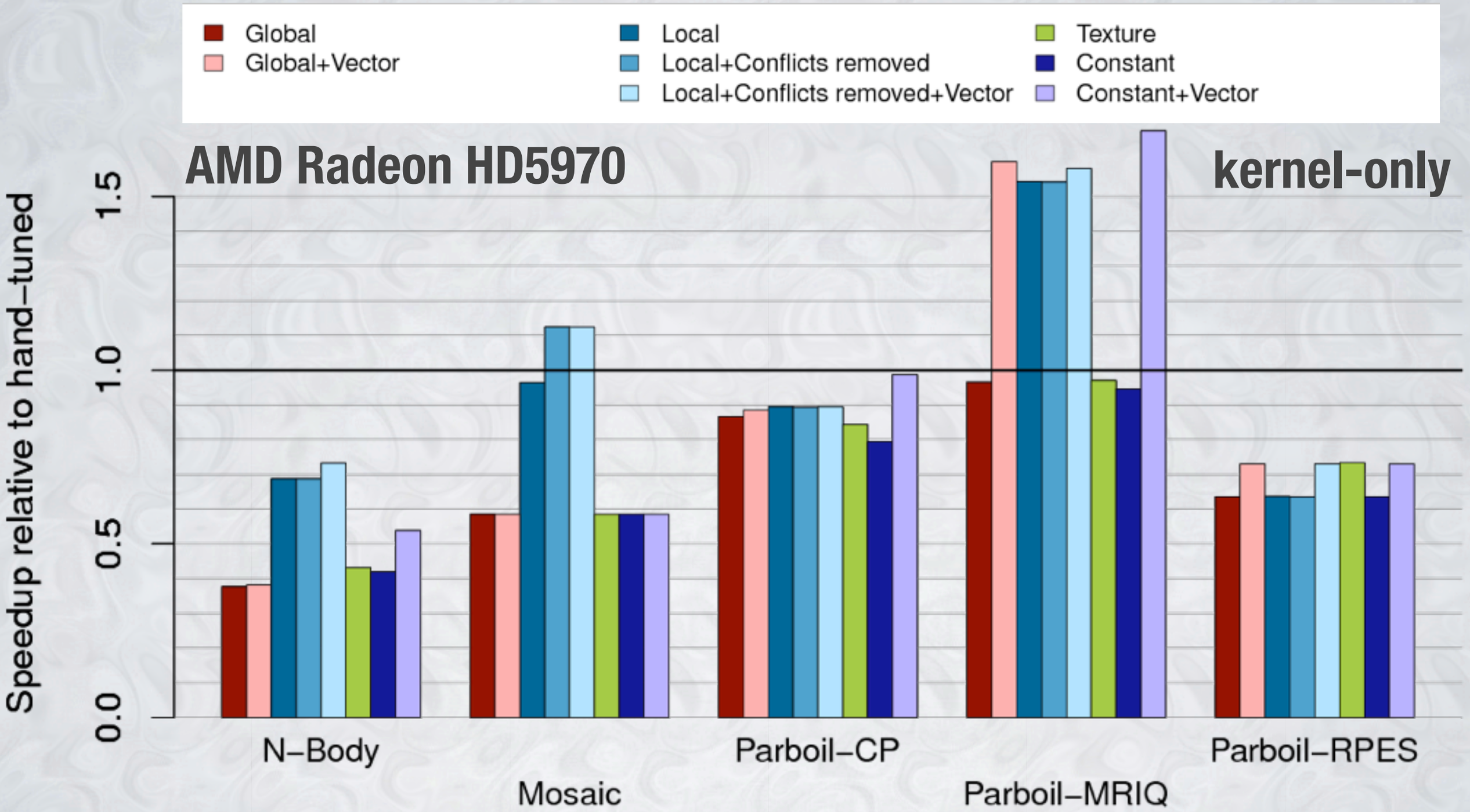
COMPARED TO TUNED OPENCL [PLDI '12]



Compared to Tuned OpenCL [PLDI '12]



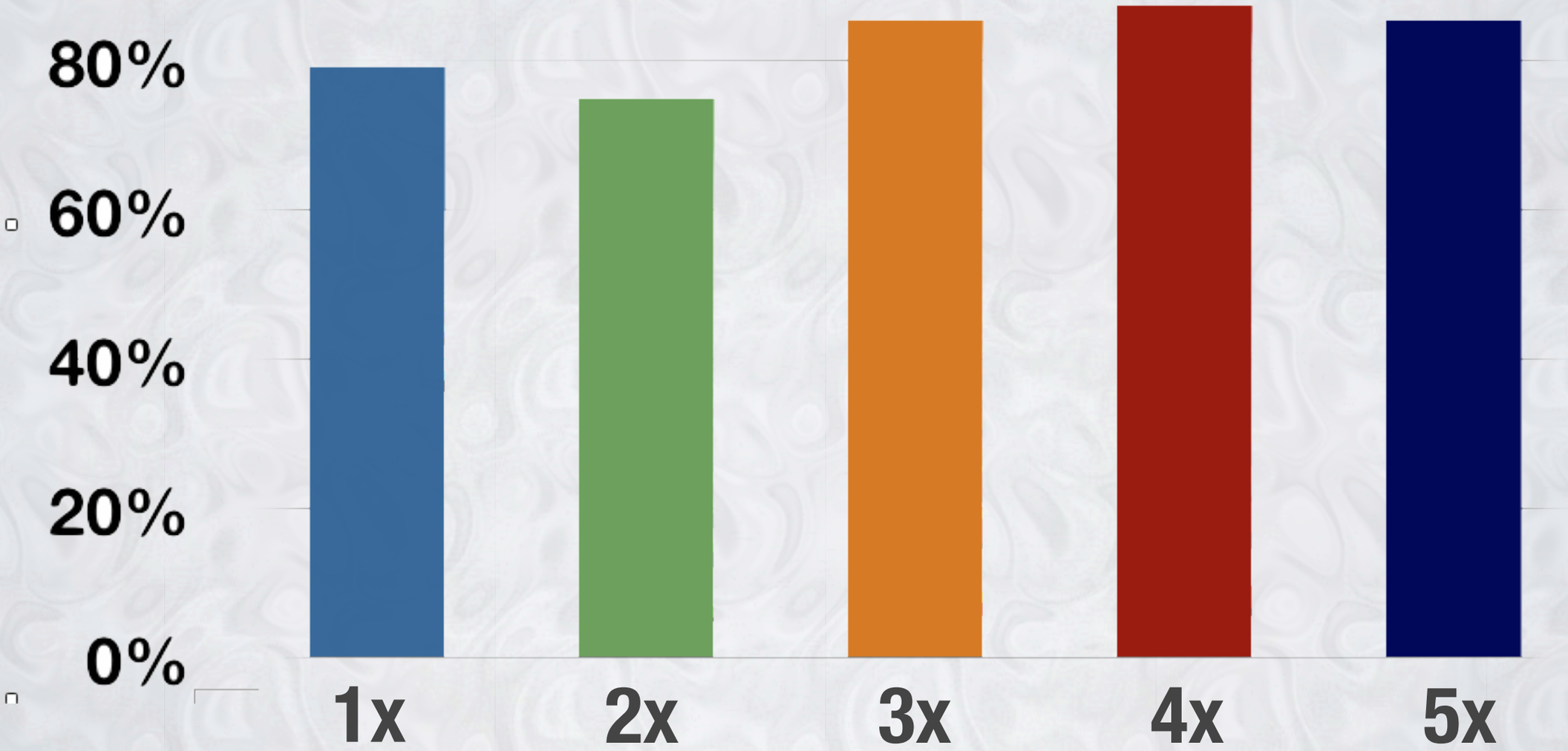
COMPARED TO TUNED OPENCL [PLDI '12]



Performance Oracle

- Prediction accuracy that speedup on GPU is at least 1-5x compared to OpenMP

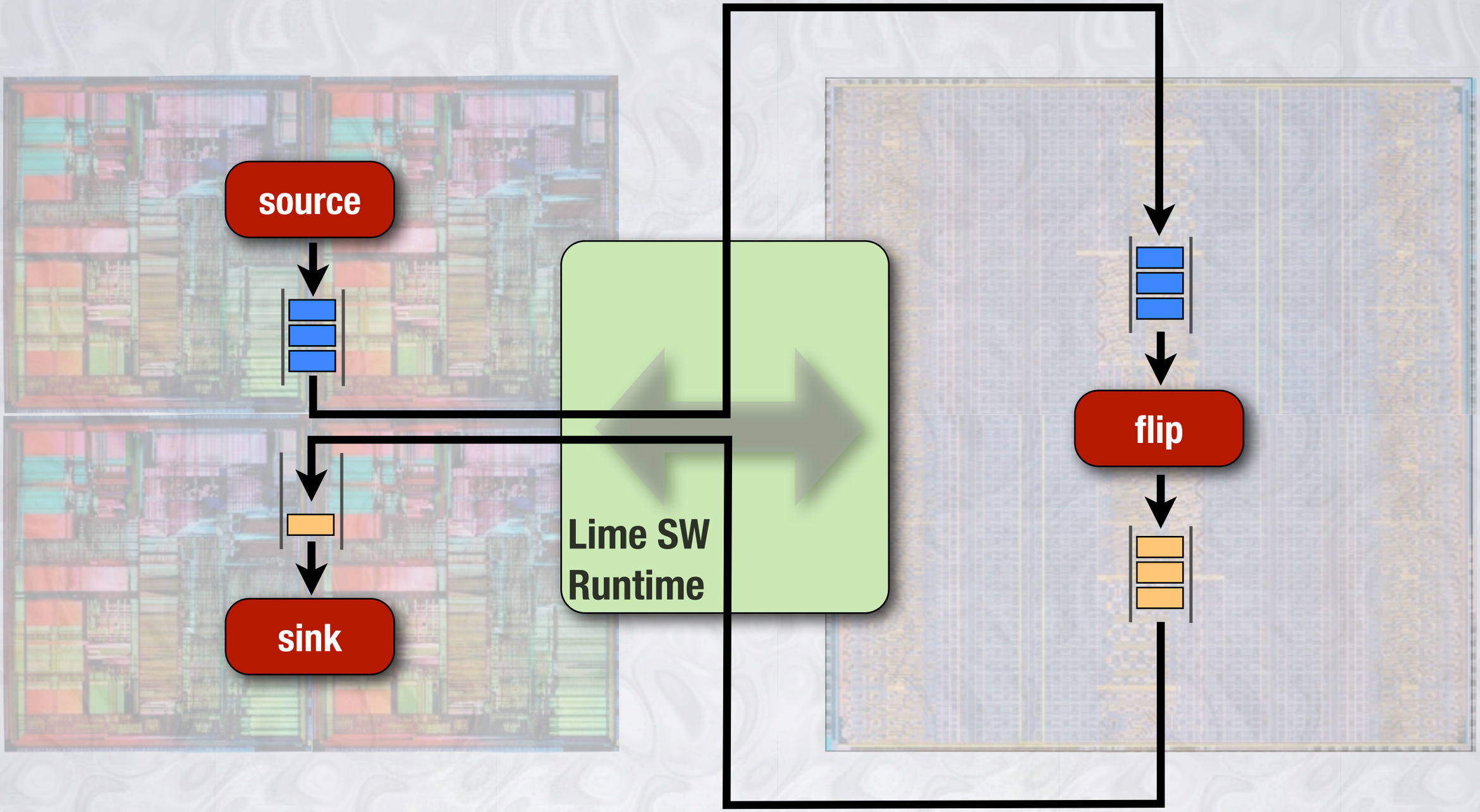
- 18 benchmarks, Tesla C2050 GPU, 12 core 3.5GHz Xeon



FPGA Backend

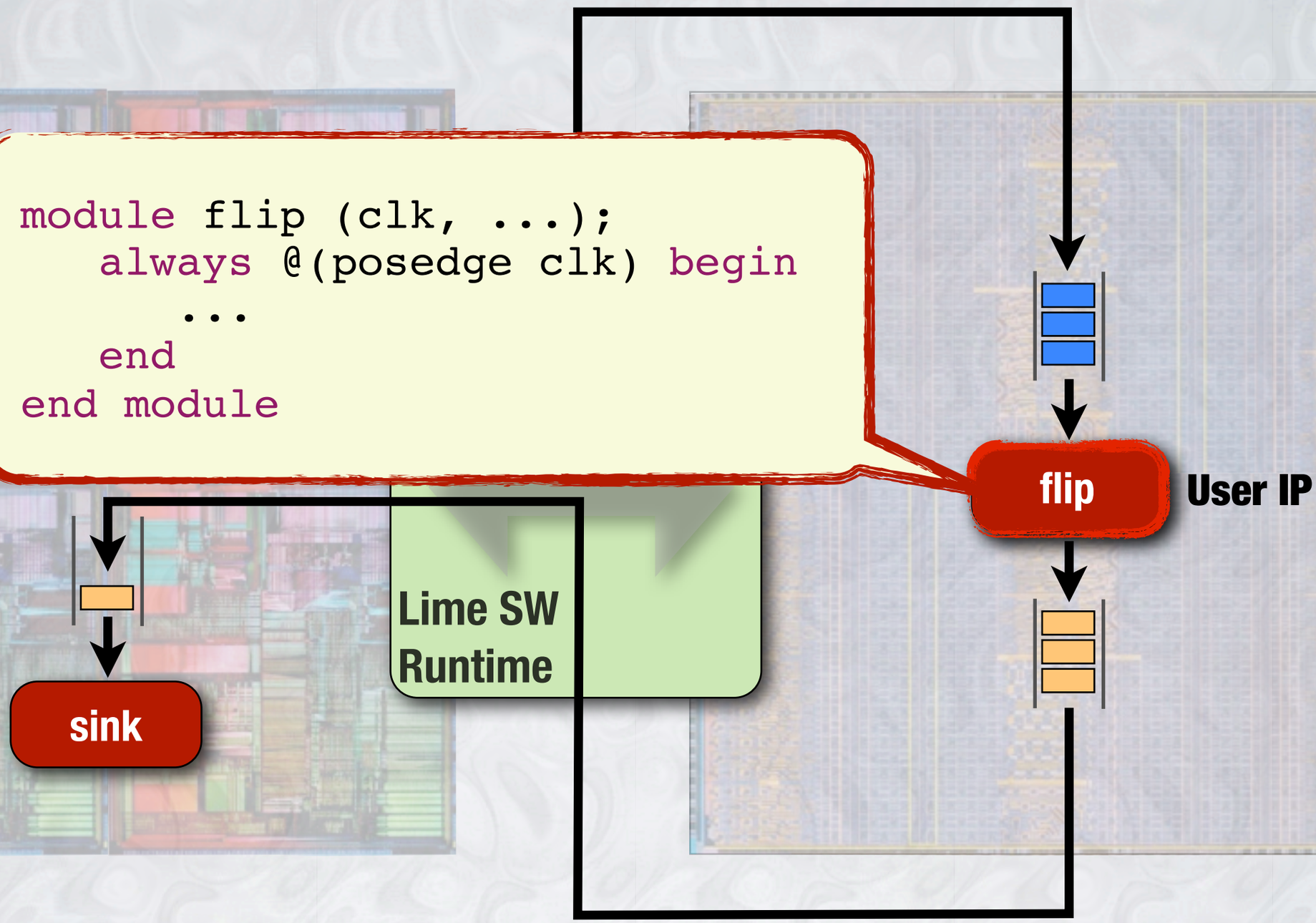
- Handles “bounded” subset of language
- Generates Verilog
- Lime tasks are compiled to modules, => with FIFOs
- Dynamically scheduled modules
- Pure functions as combinational logic
- Stateful methods semi-pipelined; work in progress
- Bitfile as artifact

Canonical IP Packaging

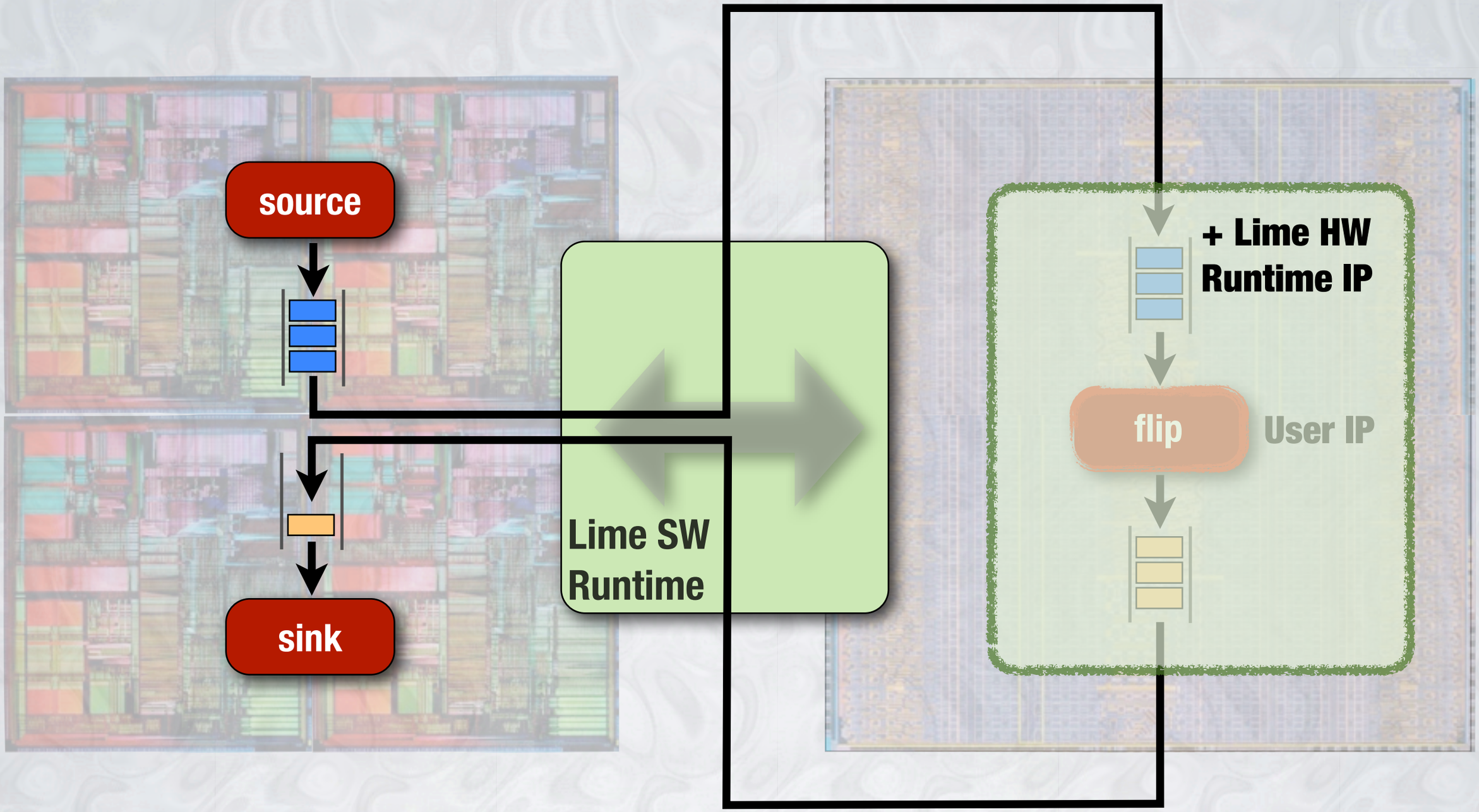


CANONICAL IP PACKAGING

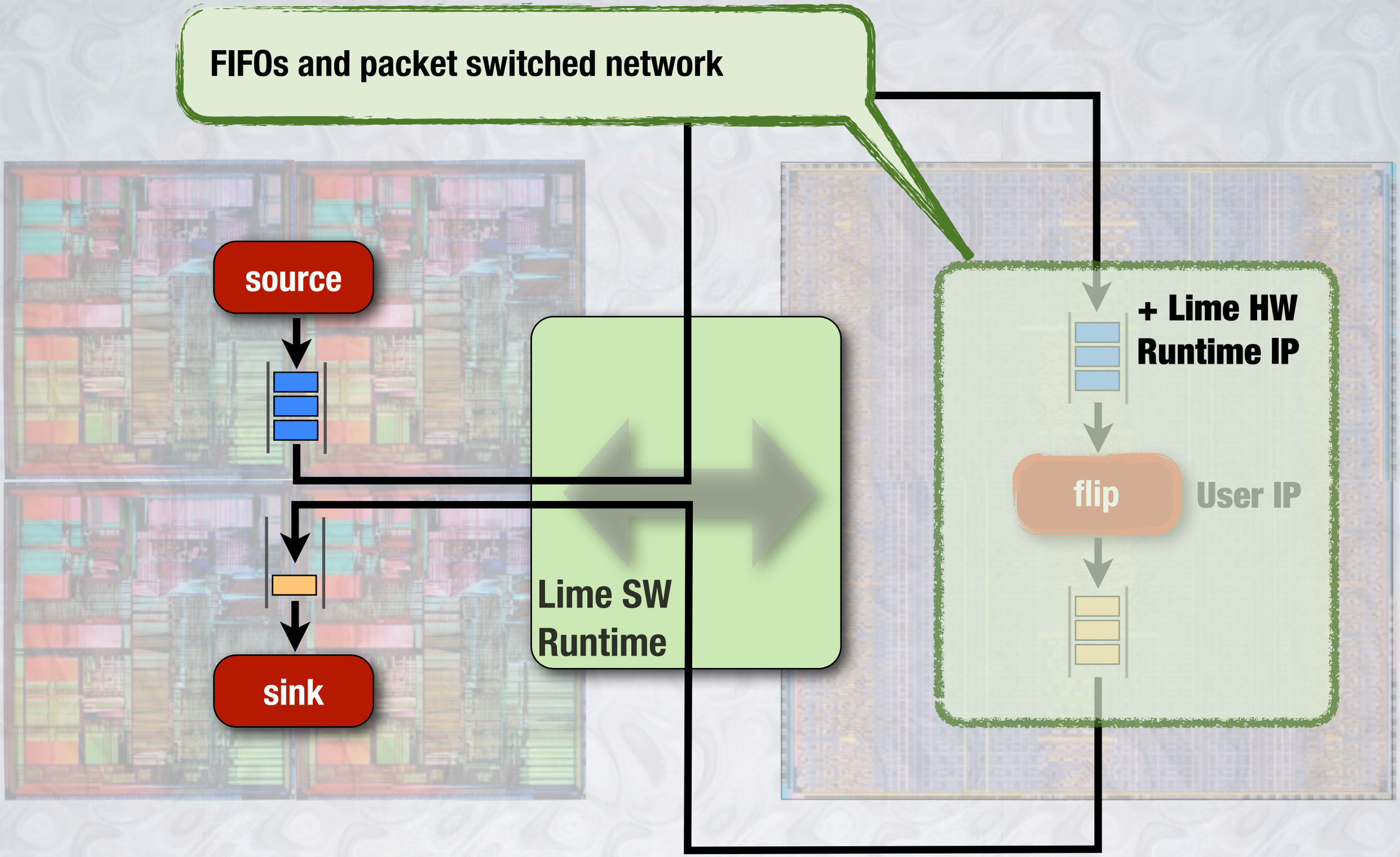
```
module flip (clk, ...);
    always @(posedge clk) begin
        ...
    end
end module
```



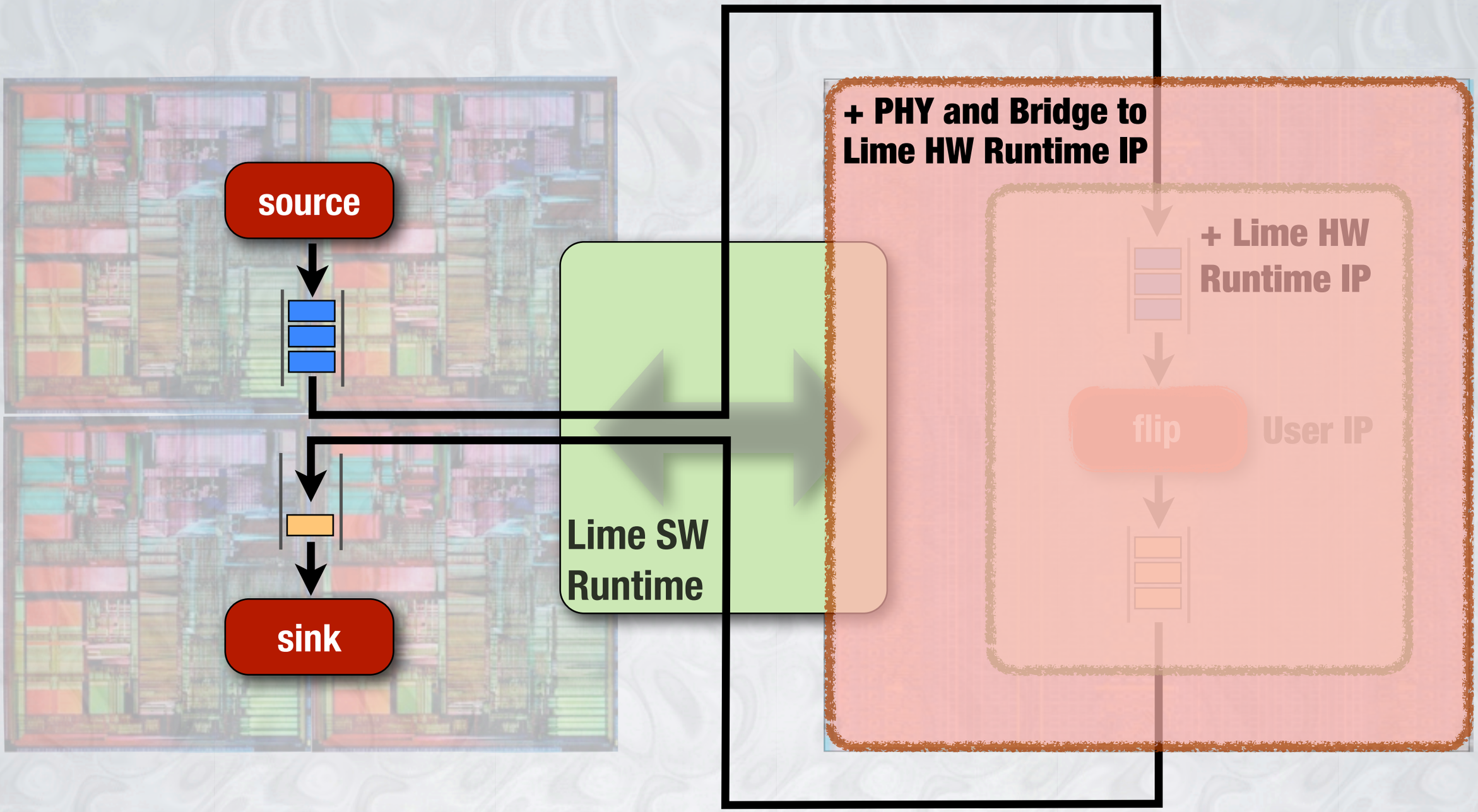
CANONICAL IP PACKAGING



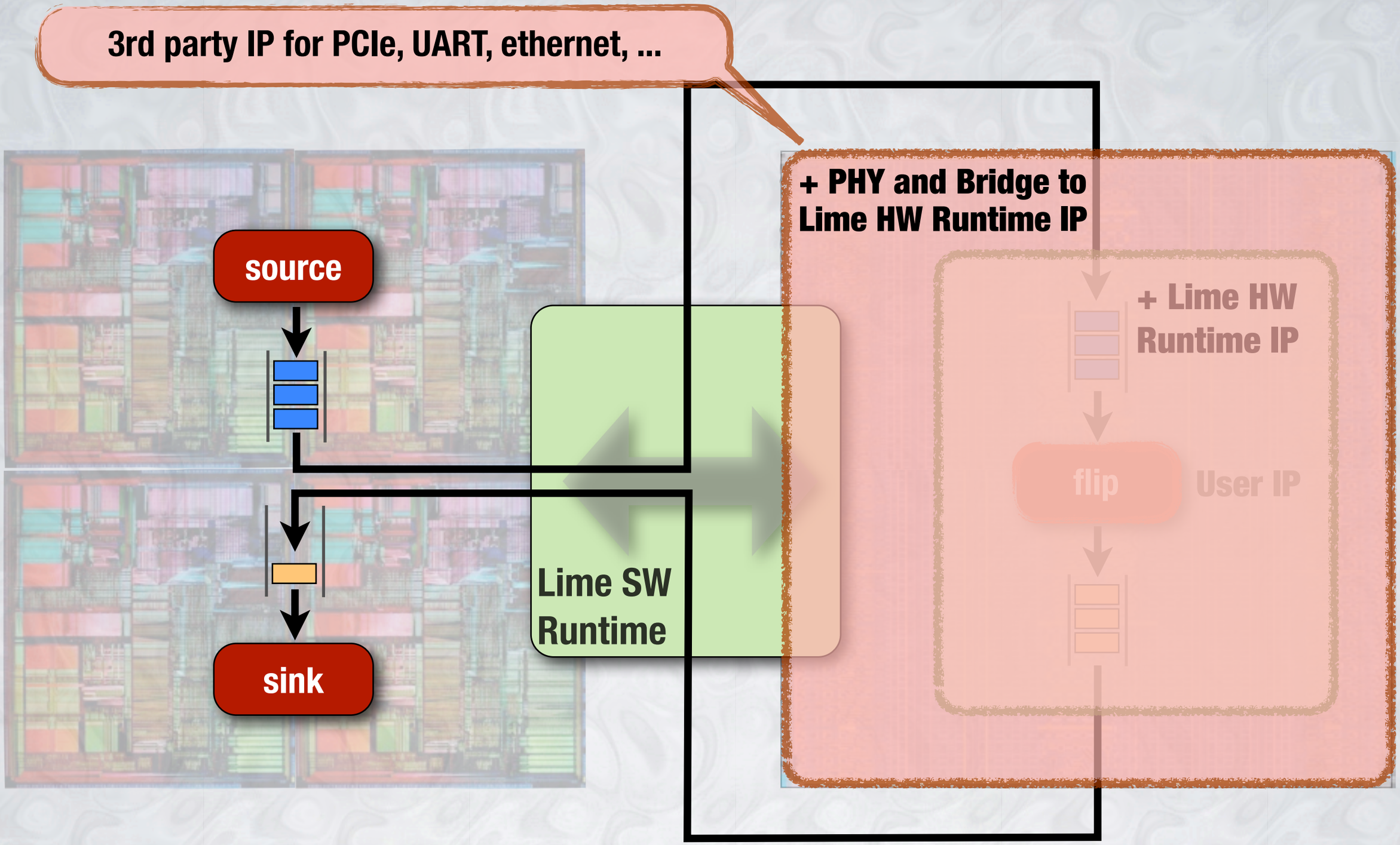
Canonical IP Packaging



CANONICAL IP PACKAGING



Canonical IP Packaging



Platform-Independent Interfaces

- Software interface
  - Lime runtime => device driver
- Hardware interface
  - Lime compiler => physical layer

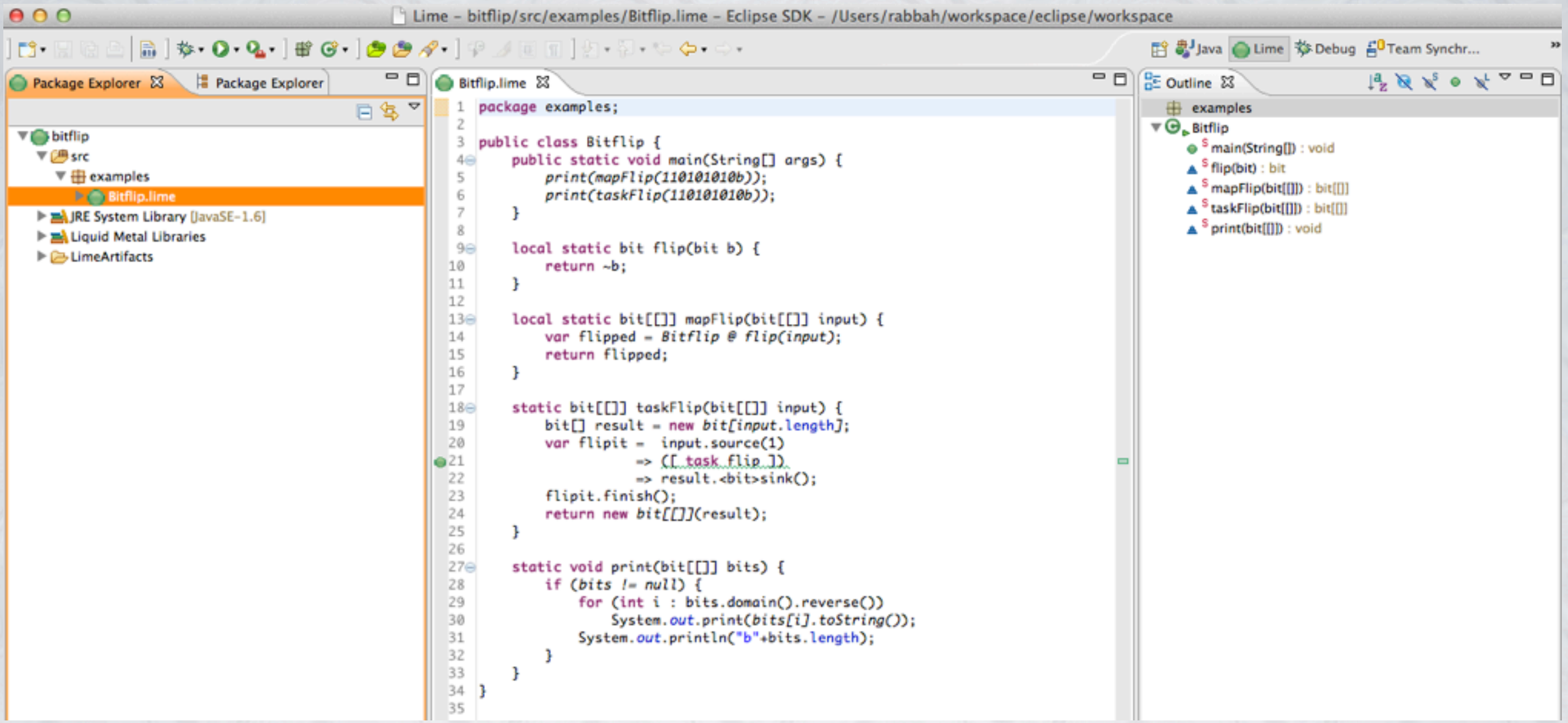
FULLY INTEGRATED LOGIC SYNTHESIS

```
% limec -fpga=nallatech.pcie280 Bitflip.lime
[06:18:44] GraphExtractor: Extracted 1 graphs and 1 tasks.  1 tasks from flipBits.
[06:18:45] HDL Codegen: Processing task Task Bitflip_flip
[06:19:47] Estimated progress: 1%
[06:22:13] Estimated progress: 25%
[06:25:13] Estimated progress: 80%
[06:26:48] Estimated progress: 90%
[06:28:29] HDL codegen build result: Success
[06:28:29] ======= Synthesis Result =======
[06:28:29] FPGA          = PhysicalFPGA[Xilinx-virtex5-xc5vlx330t]
[06:28:29] Clock Freq    = 101 MHz
[06:28:29] LUTs          = 6652 (3.2%)
[06:28:29] FFs           = 6815 (3.3%)
[06:28:29] Slices        = 3373 (6.5%)
[06:28:29] BRAMs         = 22 (3.4%)
[06:28:29] DSPs          = 0 (0.0%)
----------
1. Lime Acceleration in Bitflip.lime (at line 23)
   => ([ task flip ])
      ^^^^^^^^^^^^^^^
Code generated by FPGA backend
```

AUTOMATIC PROGRAMMING OF FPGA

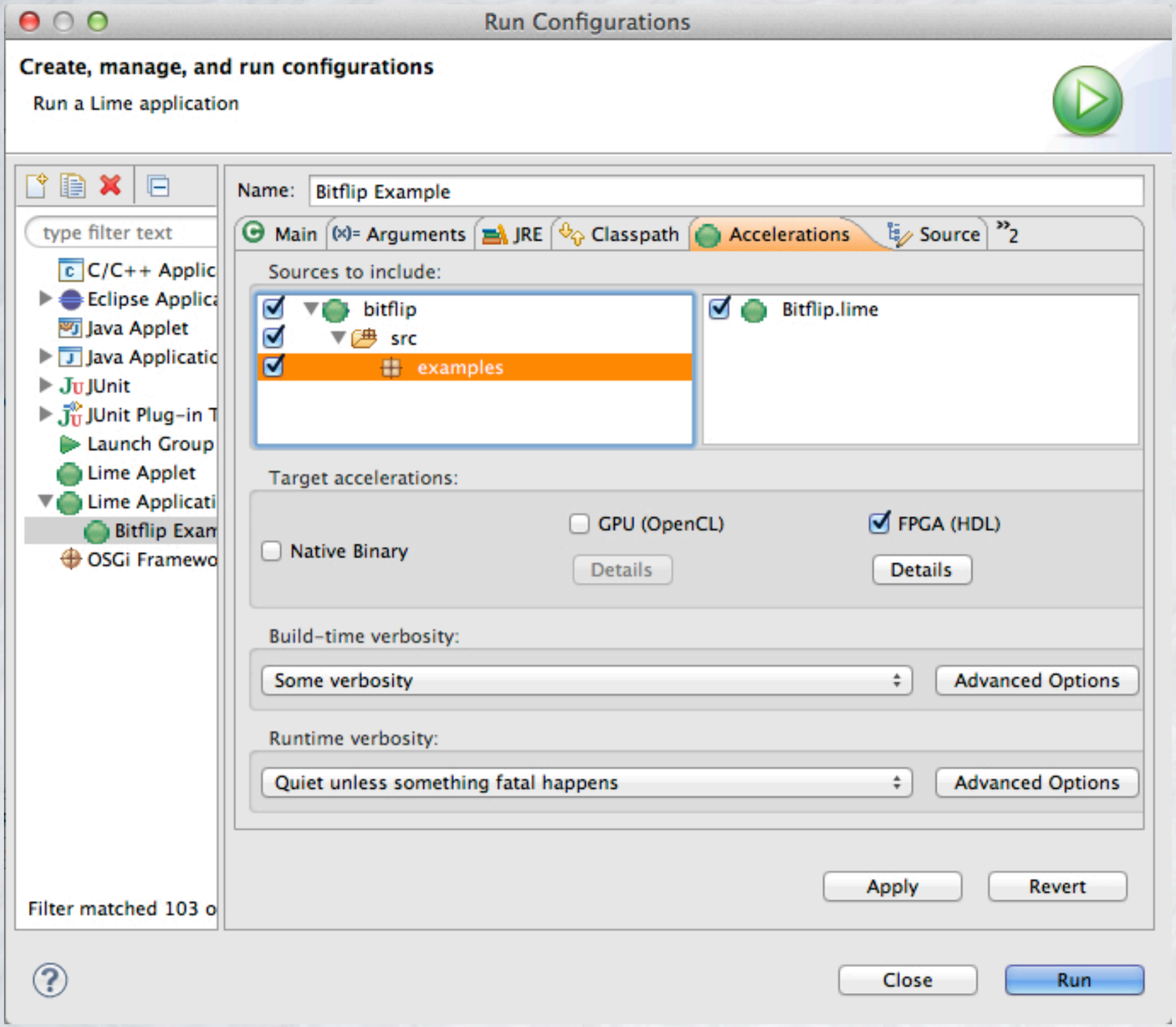
```
% lime -fpga=nallatech.pcie280 Bitflip 110101010

001010101b9

Filter                                  | Information                 |
----------------------------------------------------------------------------------------
NativeRunner (Bitflip.flip-1)           |                             |
    HDL Graph                           | Total Active (cy) =    27   |
                                        |   Number of Tasks =     1   | Number of Fifos   = 2
    Task 0 (Graph = Bitflip.flip-1)     | Total Active (cy) =    27   | Initializing (cy) = 0
        Module = Bitflip_flip           |         Iteration =     9   | Cycle/Iteration   = 3.0
```

ECLIPSE-BASED IDE



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

Integrated Compile & Run



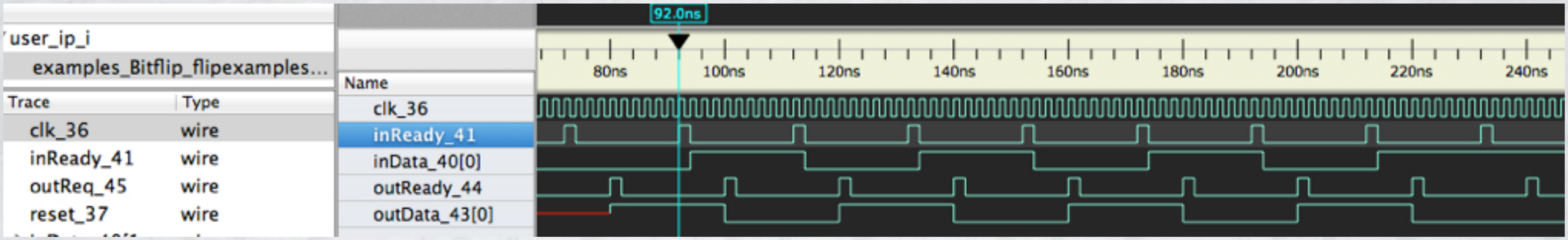
- AMD & NVidia GPUs
- FPGA/RTL simulators
- Xilinx & Altera FPGAs

FPGA Co-Simulation Also

Problems | Javadoc | Declaration | Search | Console | History | Progress

<terminated> Bitflip [Lime Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Jun 4, 2012 1:39:14 PM)

```
001010101b9
[01:39:15.030] [45] >>>>>>>>>> Thread 45 =                    Thread[main,5,main]  <<<<<<<<<<
[01:39:15.030] [45] RPM    : Replacement Plan FPGA
AtomicIds [1]
[01:39:15.033] [45] Executor: Task 1: 1 1

[01:39:15.276] [45] HWAsyncFilter: Made a native hw filter for artifact graphId = examples_Bitflip437_451 replacementUnitId = 1 backend
[01:39:15.279] [45] Executor: performReplacement: Graph examples_Bitflip437_451 replacement-id: 1 Success!
[01:39:26.820] [21] >>>>>>>>>> Thread 21 = Thread[UnmarshalFromNative1-Bitflip.flip-1,5,main]  <<<<<<<<<<
[01:39:26.820] [21] IcarusRunner: simulation log files and wave forms are located in /Users/rabbah/workspace/eclipse/workspace/bitflip/L
001010101b9
```



Lessons Learned

- **Programming FPGA in Lime is not a free lunch**
  - **algorithm = architecture**
  - **performance tuning still hard for skilled SW developers**

LIME VS. COMMERCIAL C TO GATES

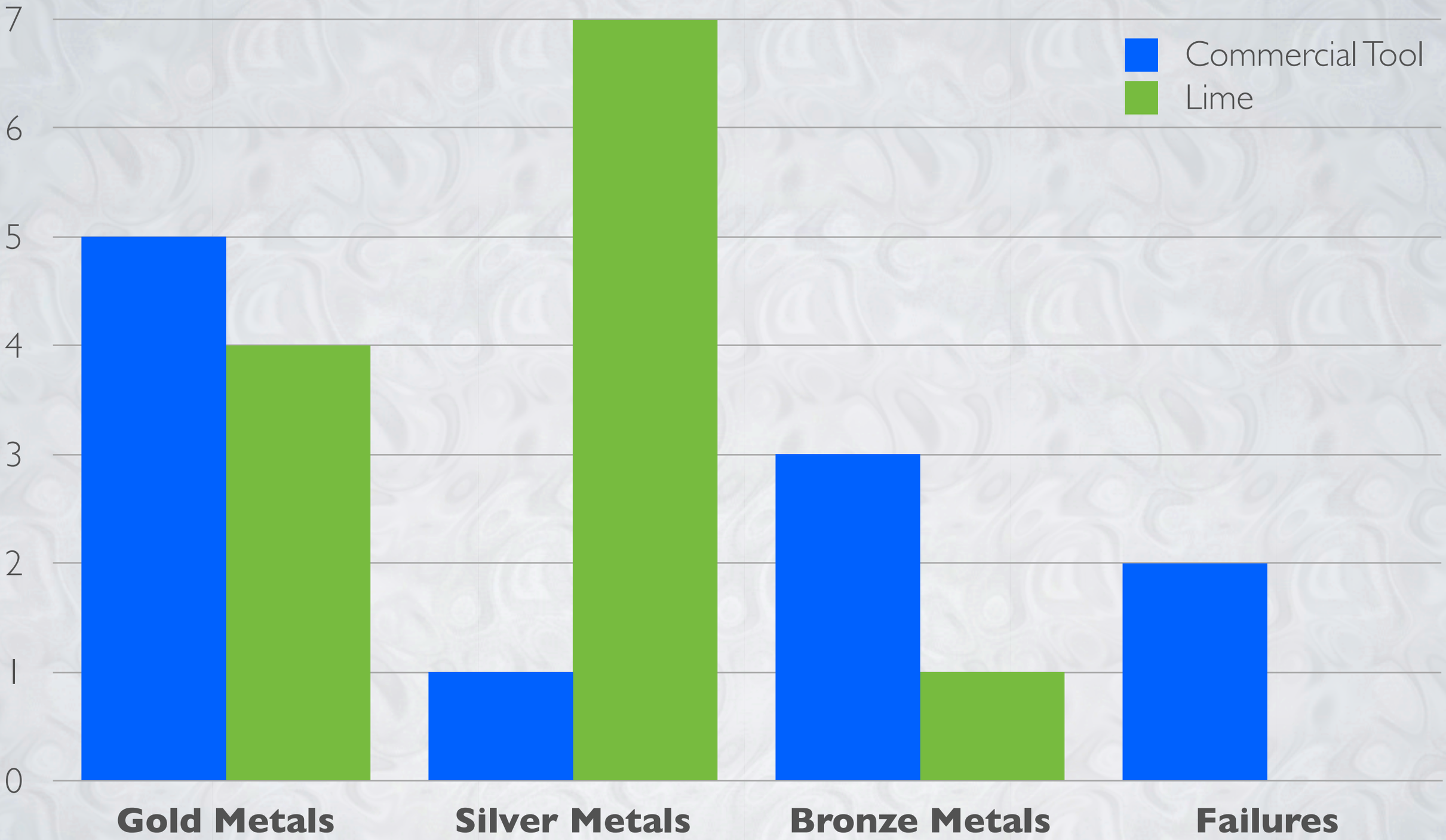
- Compare Lime to commercial C to Gates tool
  - Robustness
  - HDL quality

- CHStone - independent benchmark suite in C
  - encryption, codecs, MIPS processor, FPUs

- Transliterated CHStone to Lime
  - No “Lime” features

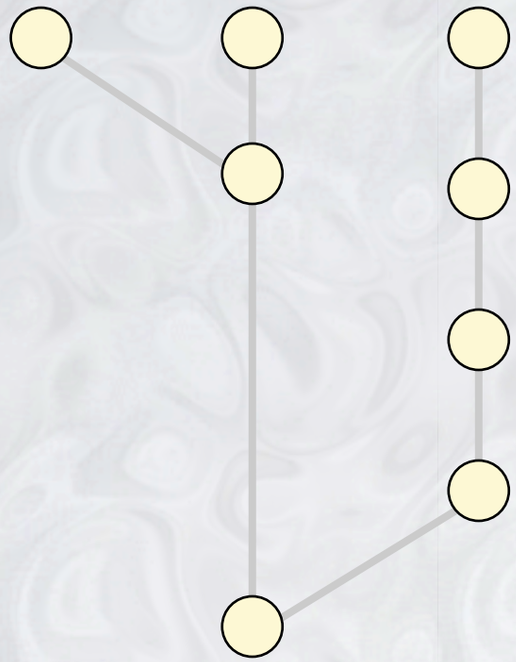
- HDL quality = measured total FPGA cycles (simulation)

HDL Quality Report Card



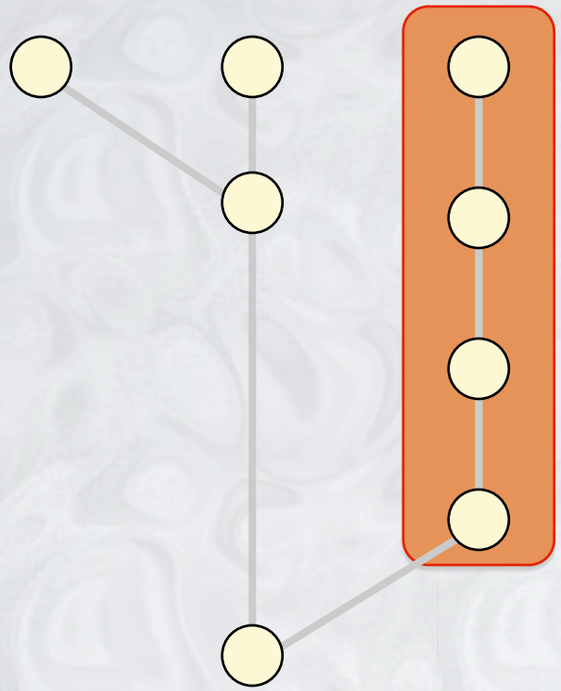
FPGA Backend Optimizations

- Macro (graph-level) and Micro optimizations



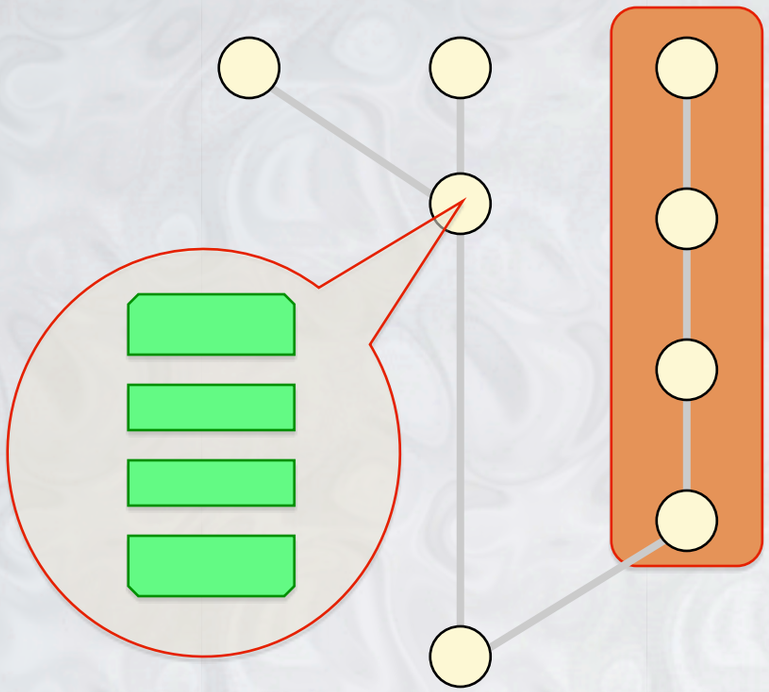
FPGA Backend Optimizations

fusion
- reduce area
- increase latency



FPGA Backend Optimizations

fusion
- reduce area
- increase latency

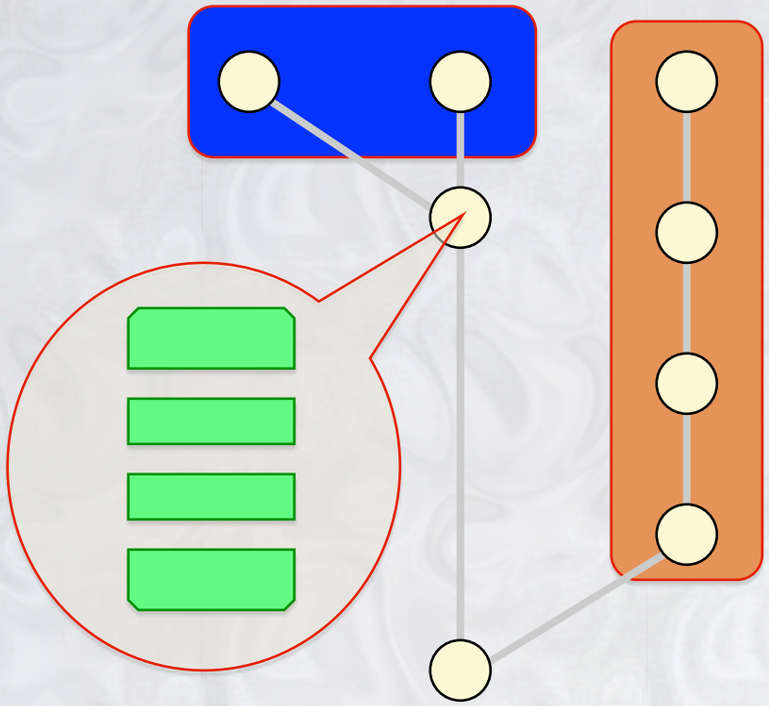


fission
- increase throughput
- increase area, latency

FPGA Backend Optimizations

folding
- share resources

fusion
- reduce area
- increase latency

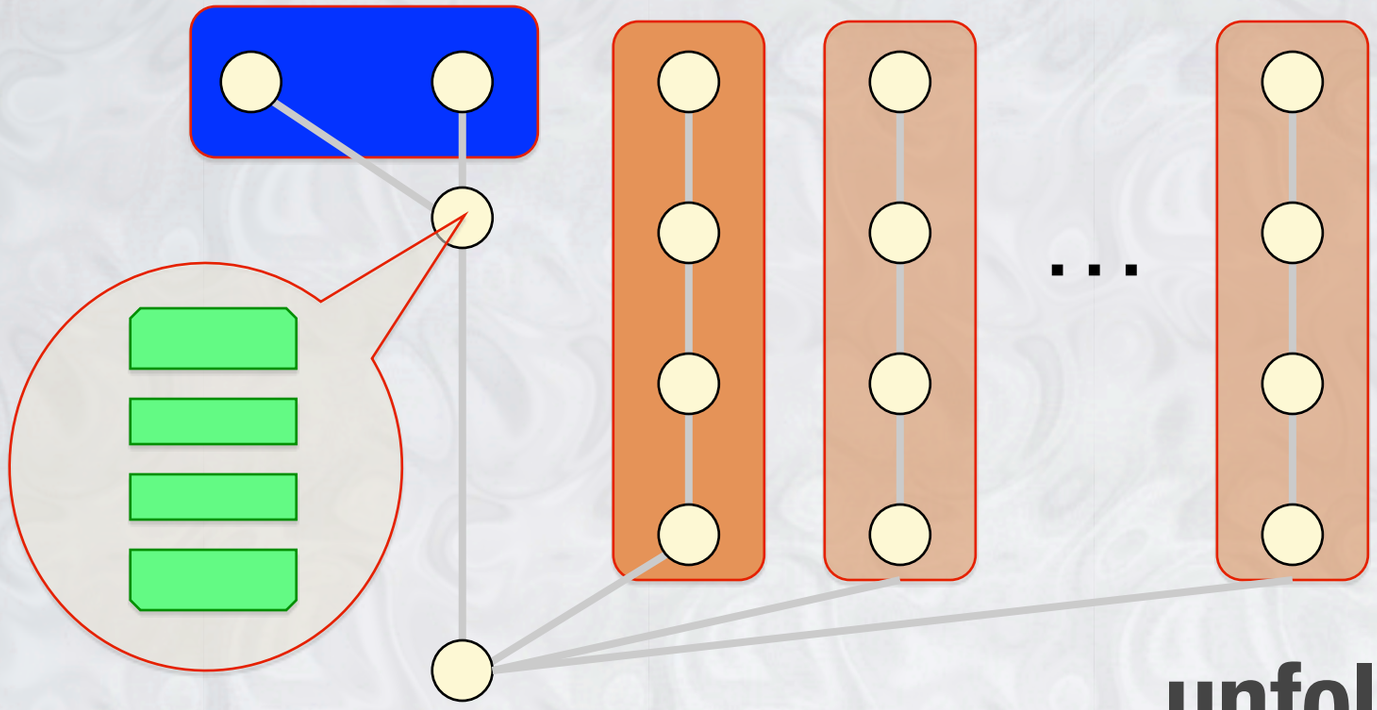


fission
- increase throughput
- increase area, latency

FPGA Backend Optimizations

folding
- share resources

fusion
- reduce area
- increase latency



fission
- increase throughput
- increase area, latency

unfolding
- increase throughput
- and area

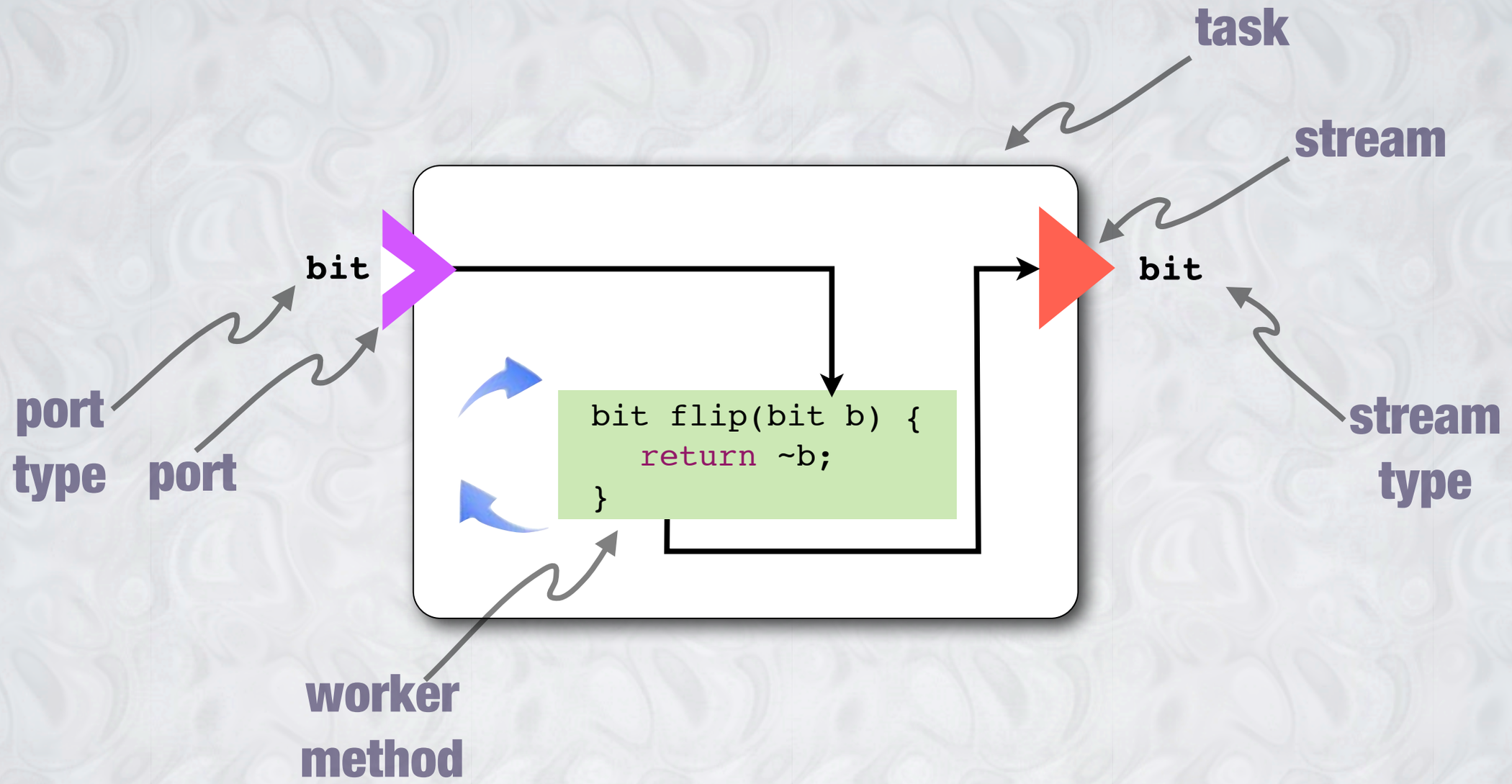
Multi-Objective Optimization

- Throughput
- Latency
- Area

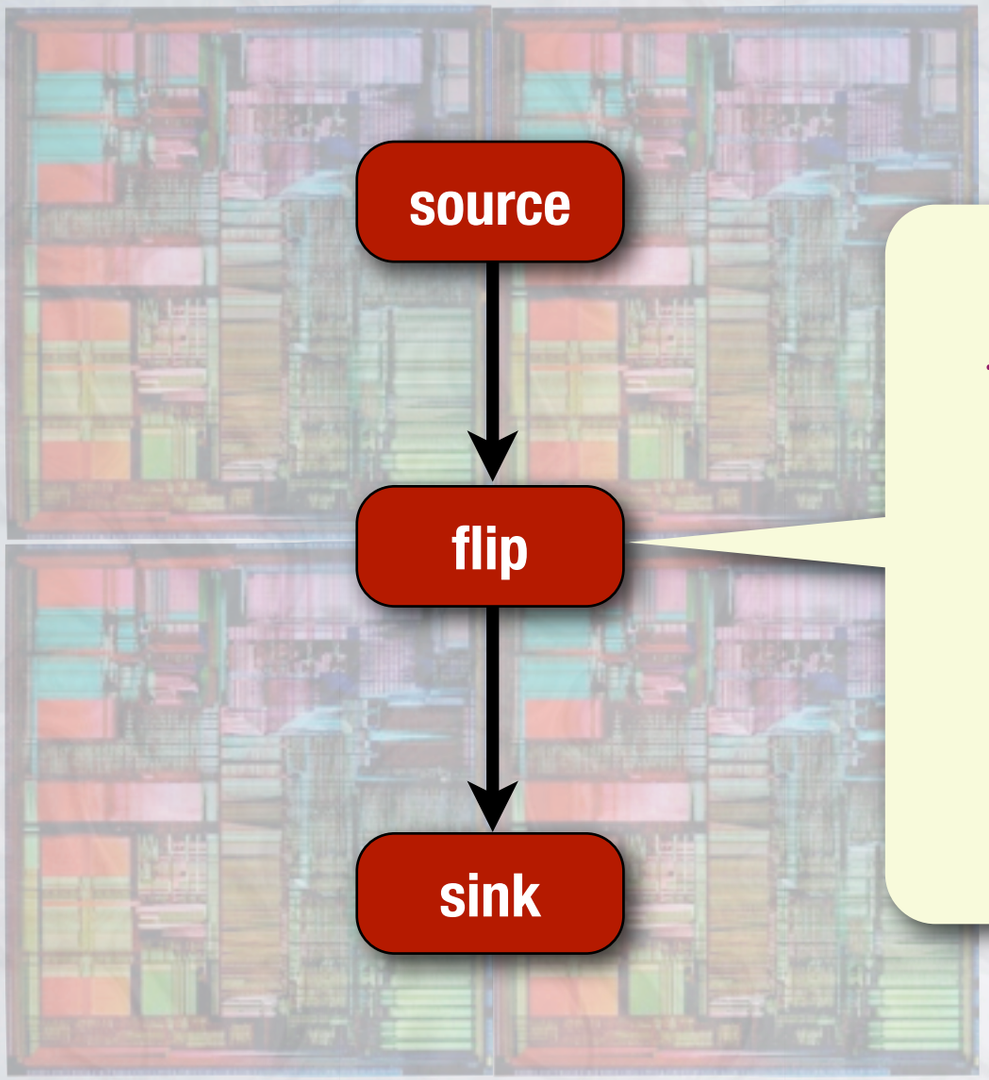
LIME RUNTIME

PORTS AND STREAMS

```
... => task flip => ...
```

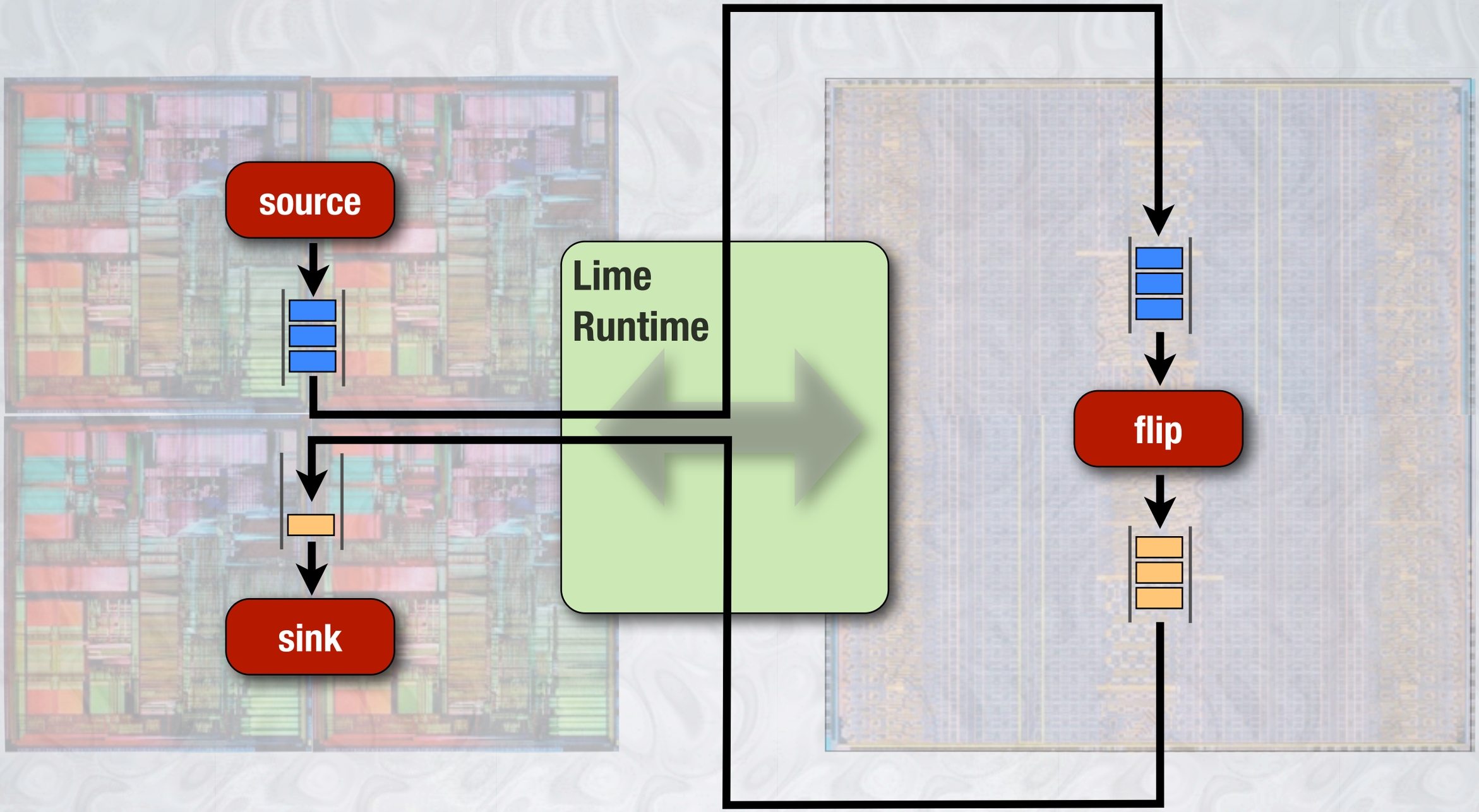


Software (Bytecode) Execution



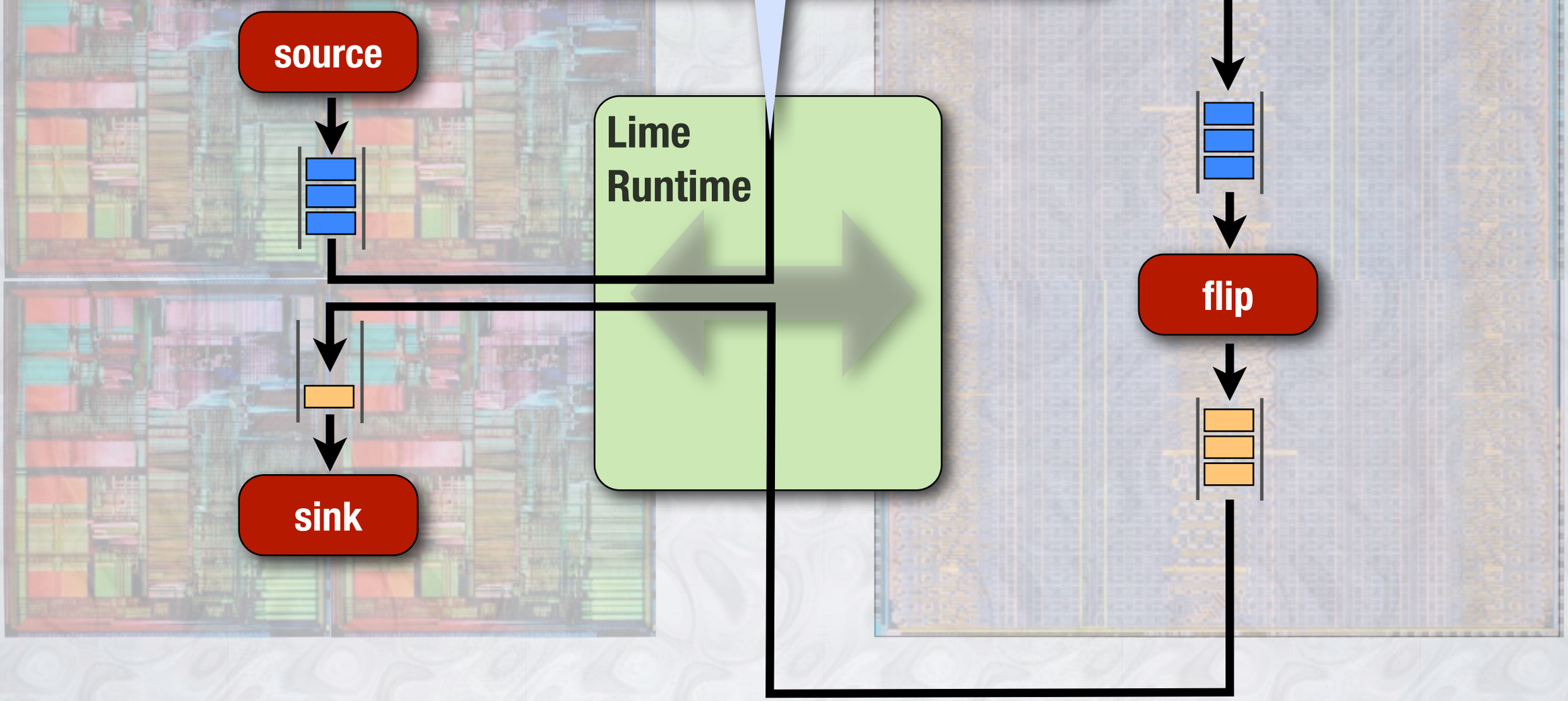

```
while (!endOfStream) {
    bit in  = port.get();
    bit out = flip(in);
    stream.put(out);
}
```

Managed Communication

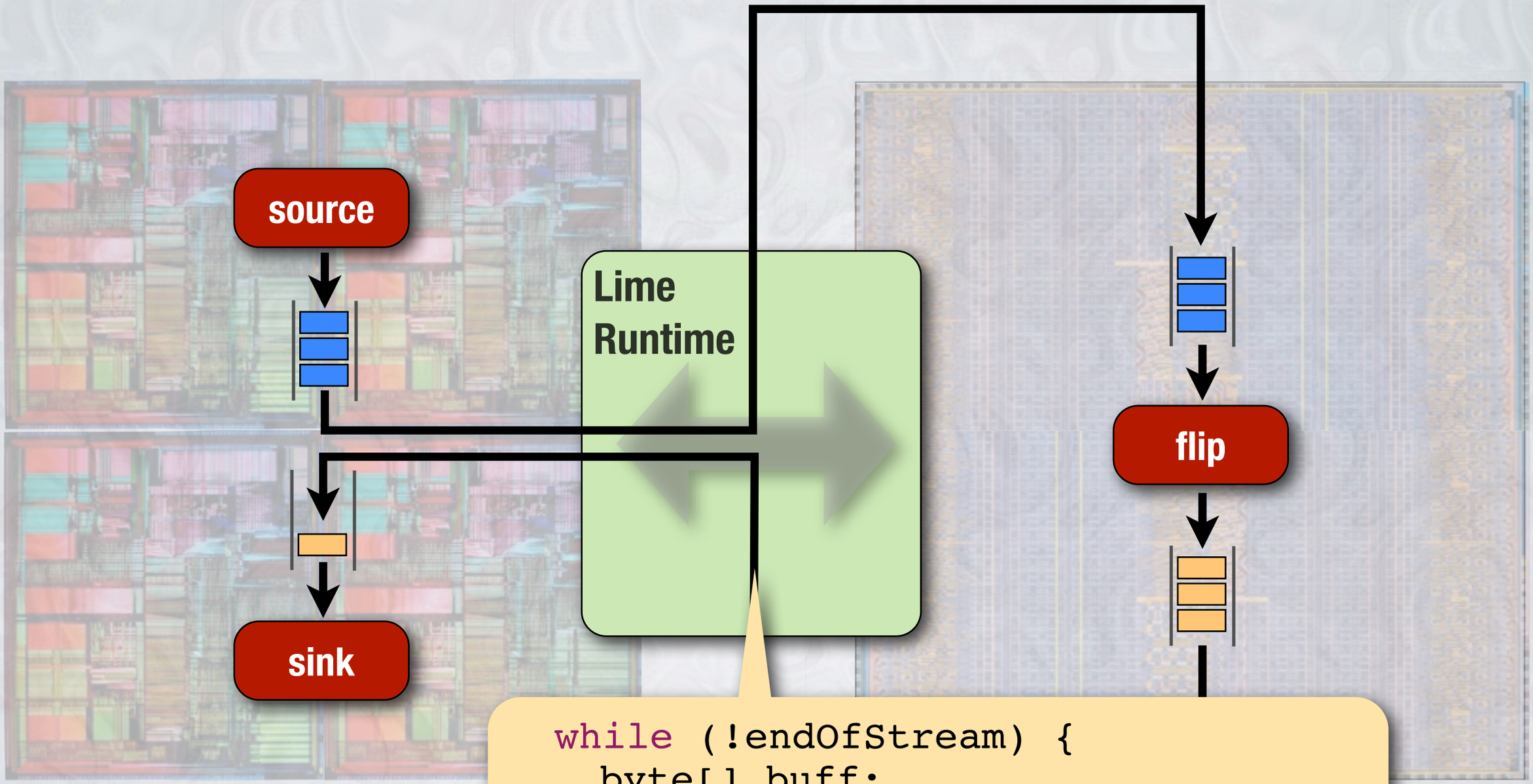


Managed Communication

```
while (!endOfStream) {
  bit in = port.get();
  byte[] buff = Marshal.toBytes(in);
  device.send(buff);
}
```



Managed Communication



Canonical Device Interface

- Synchronous (batch) or asynchronous (streaming) I/O
- Coherent or distributed memory models

```
int count = send(byte[] buffer, int timeout)
```

```
int count = receive(byte[] buffer, int timeout)
```

Measuring => Abstraction Cost

(FPGA)

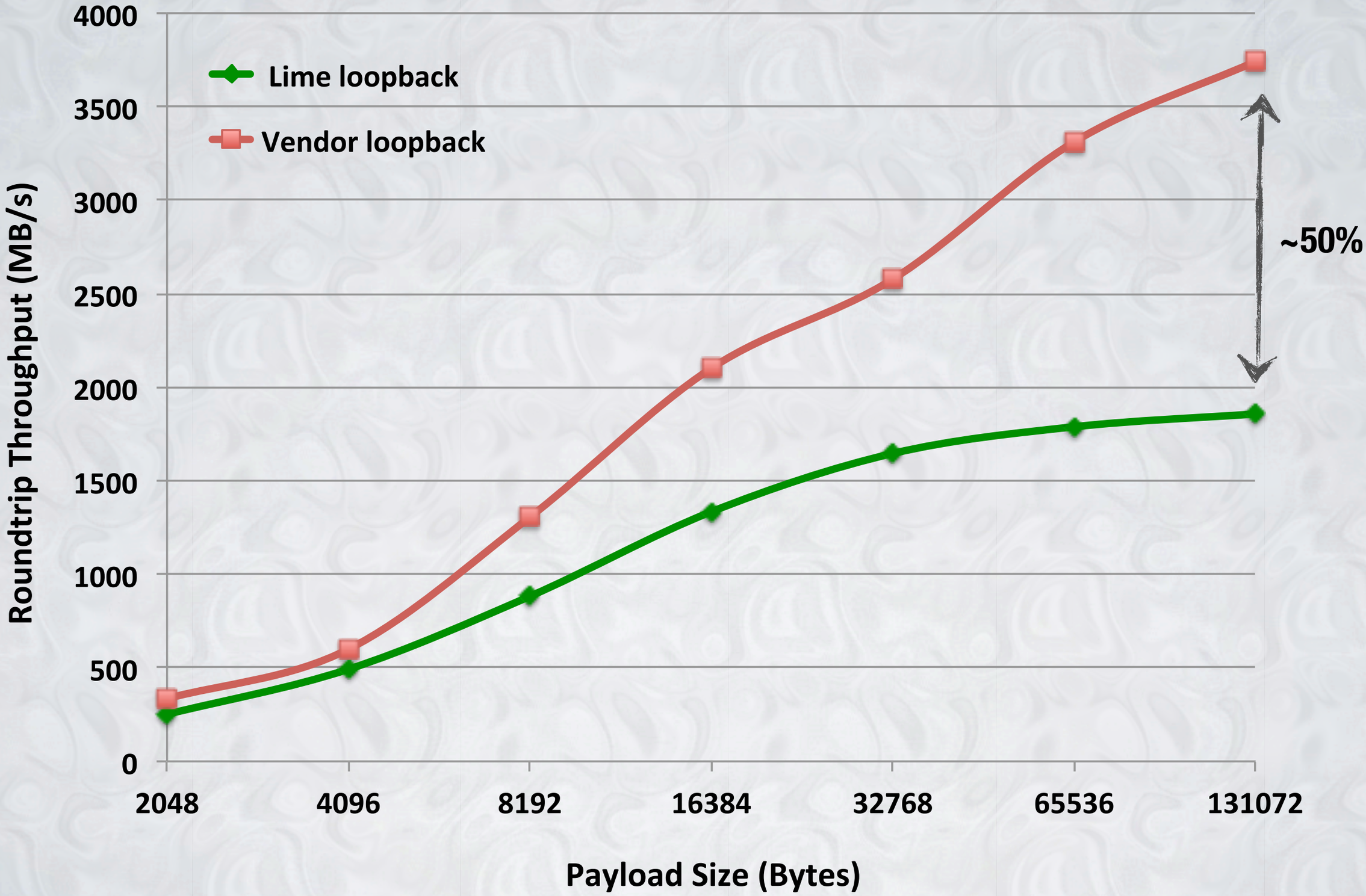
- Measure loopback bandwidth vs. packet size
  - PCIe PHY + bridge + Lime Runtime IP + identity task

- Nallatech PCIe 280 FPGA platform, asynchronous I/O

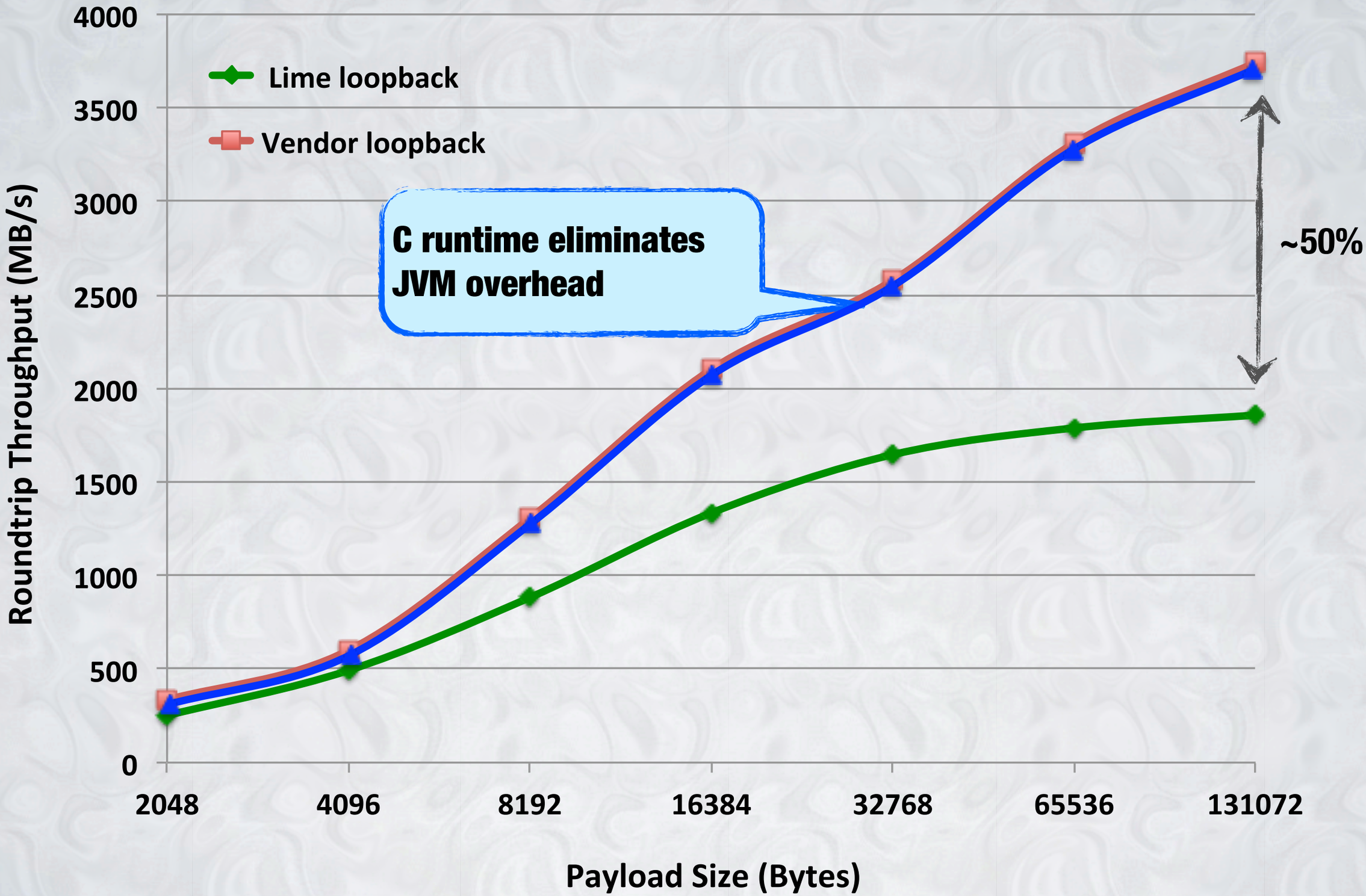
- Lime loopback
  - end-to-end, includes Lime software runtime

- Compare to vendor loopback

Cost of Lime => Abstraction



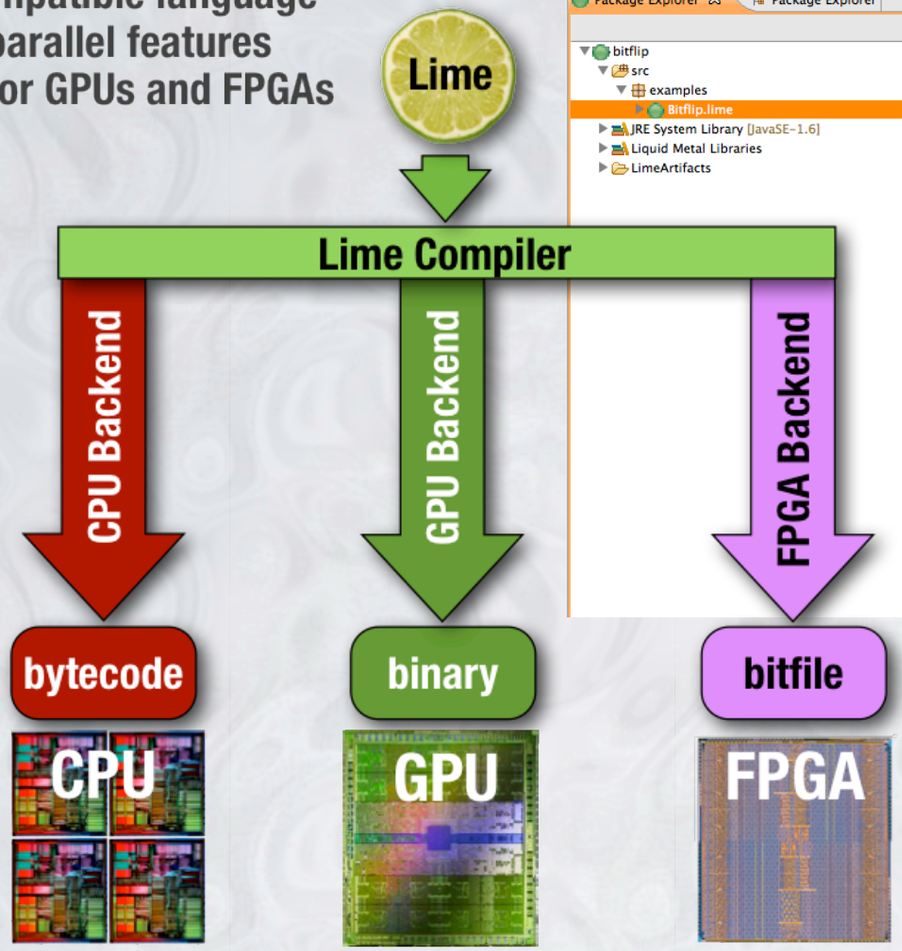
Cost of Lime => Abstraction

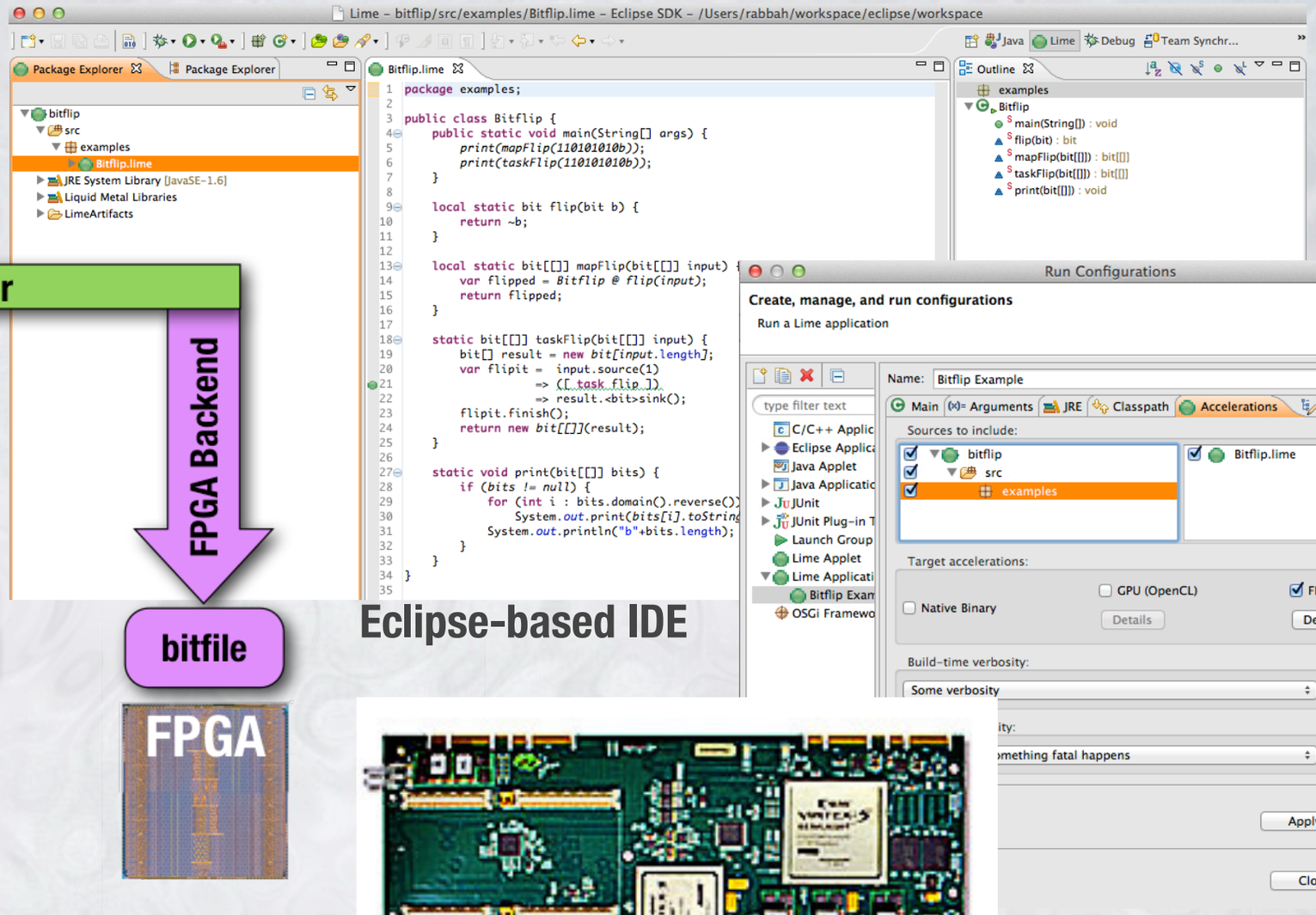


SUMMARY

Liquid Metal Overview

Java-compatible language with parallel features suitable for GPUs and FPGAs





Eclipse-based IDE

Integrated support for EDA tools and PCIe FPGA platforms

Ongoing/Open Problems

- Non-determinism
- Parallelization of stateful map and task
- Multi-objective macro and micro optimizations
- Performance oracle
- Runtime adaptation
- Debugging beyond waveforms