goto; conference aarhus

LIQUID METAL

Taming Heterogeneity

Stephen Fink
IBM Research

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

gotocon.com

Liquid Metal Team (IBM T. J. Watson Research Center)

Josh Auerbach

David Bacon

Ioana Baldini

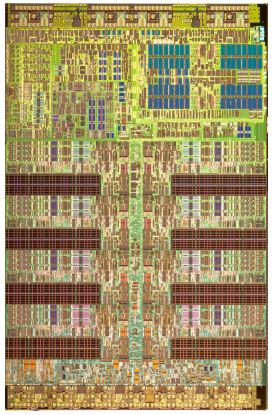
Perry Cheng

Stephen Fink

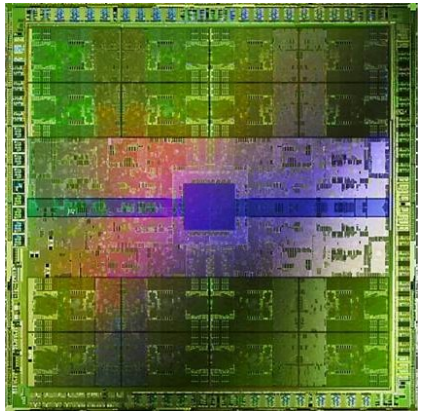
Rodric Rabbah

Sunil Shukla

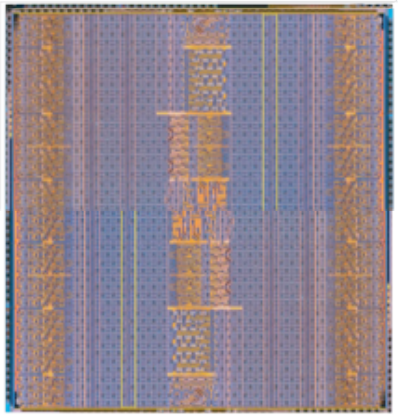
A Heterogeneous Landscape



Multicore



GPU

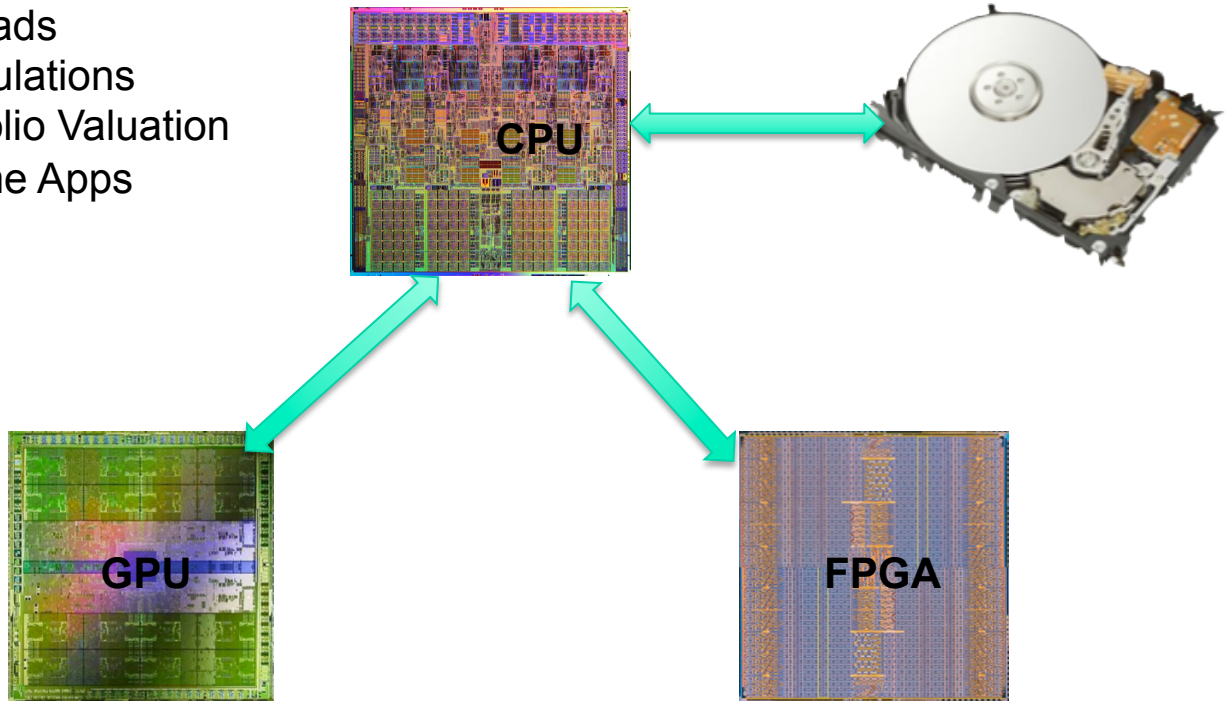


FPGA

Accelerator Platform

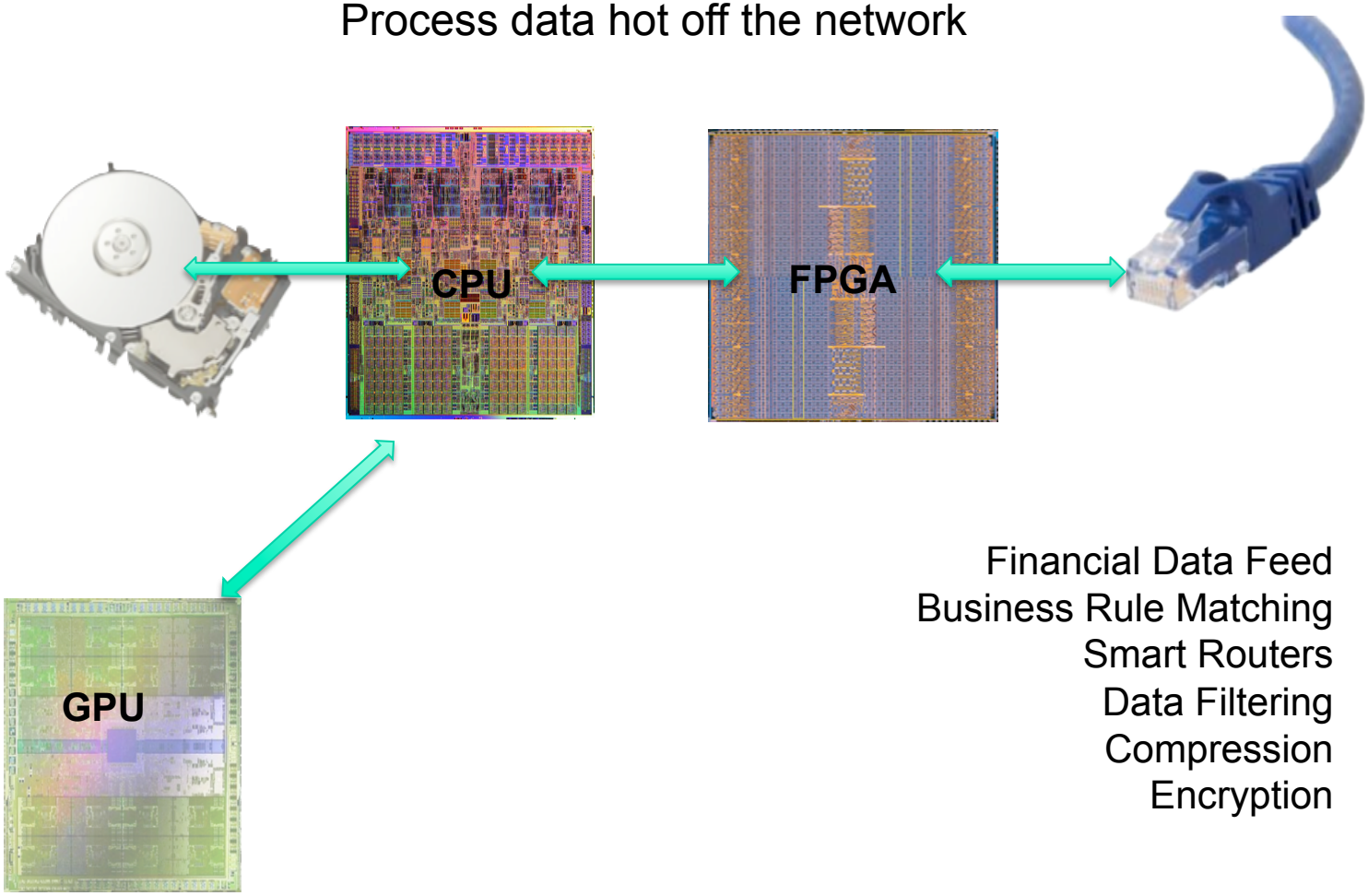
Computation starts from CPU

HPC Workloads
Seismic Simulations
Offline Portfolio Valuation
Non Real-time Apps



Network Appliance Platform

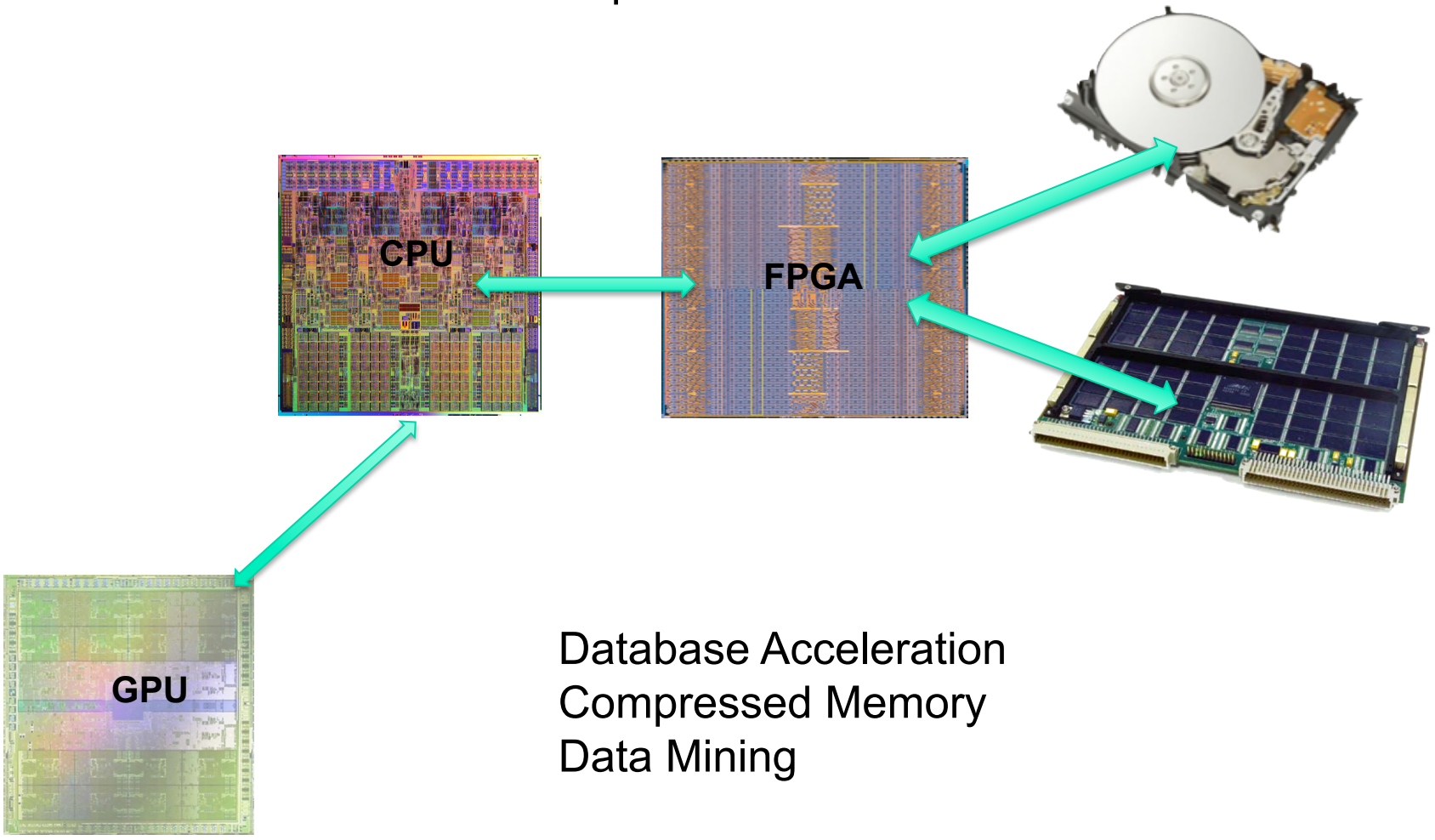
Process data hot off the network



Financial Data Feed
Business Rule Matching
Smart Routers
Data Filtering
Compression
Encryption

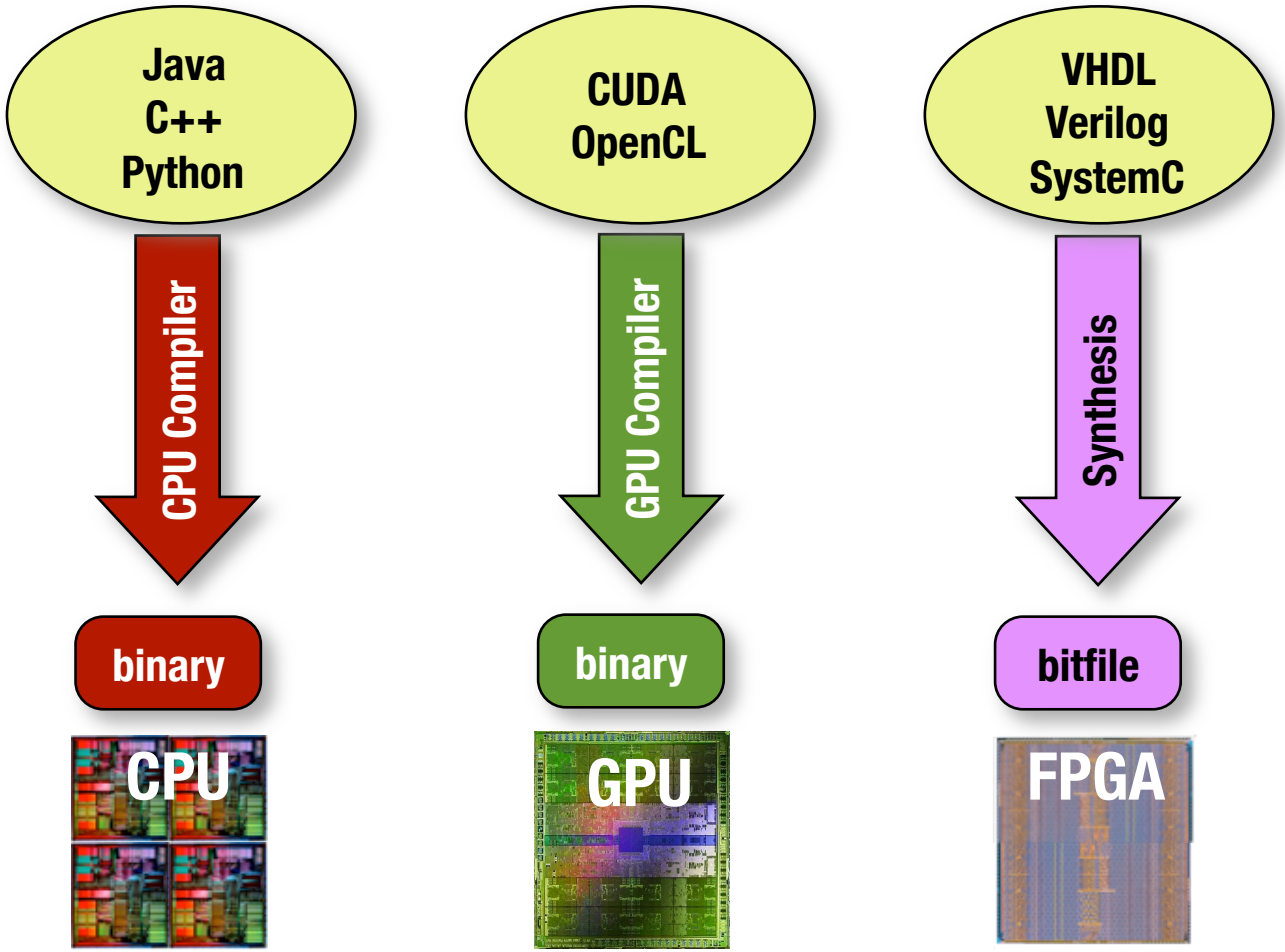
Storage Appliance Platform

Push computation near data



Database Acceleration
Compressed Memory
Data Mining

Heterogeneous Programming: Current Reality

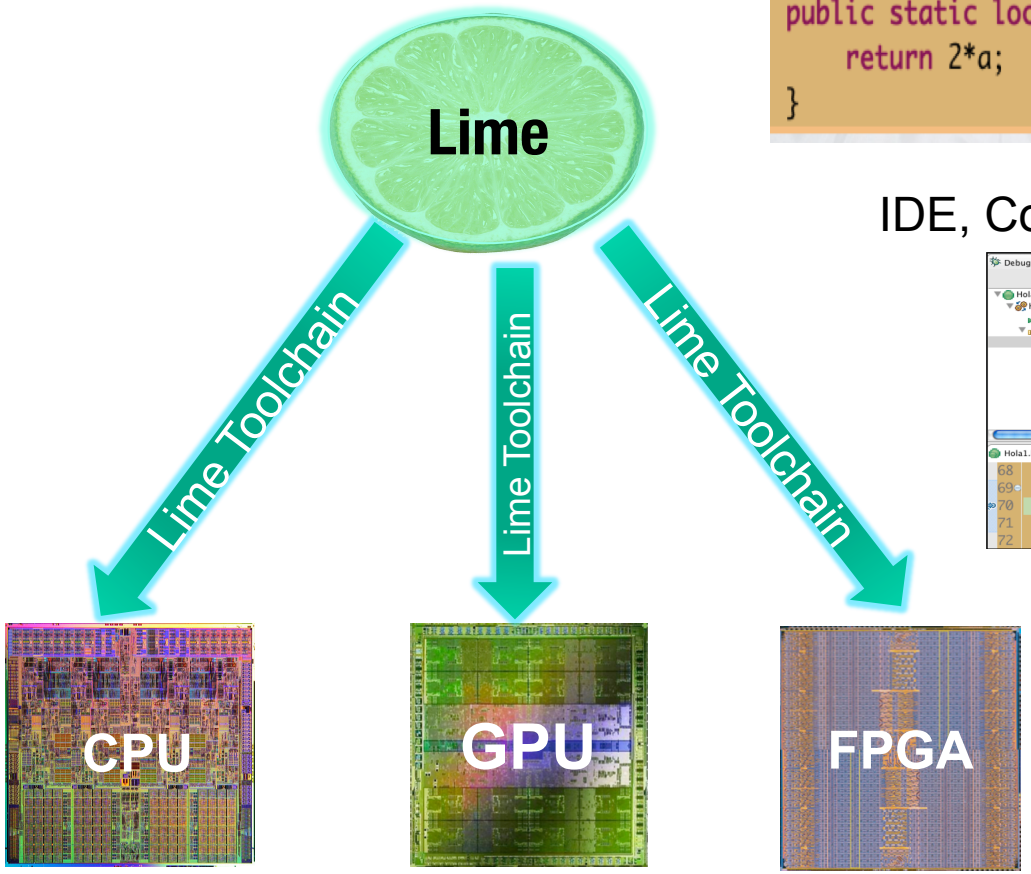


Goal: Make Heterogeneous Platforms Accessible for Mainstream Use

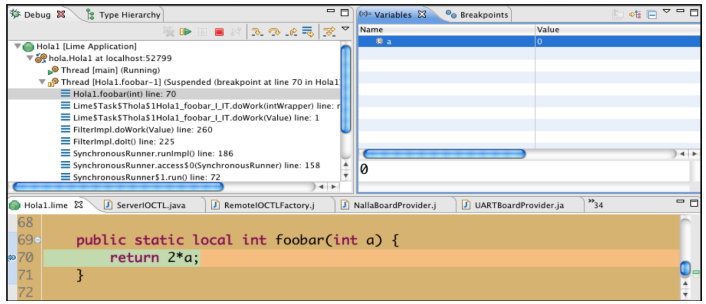
Vision:

High-Level Java-Derived Language

```
public static local int foobar(int a) {
    return 2*a;
}
```



IDE, Compiler, Debugger, Cloud Synthesis



Platform:
Hardware + Runtime

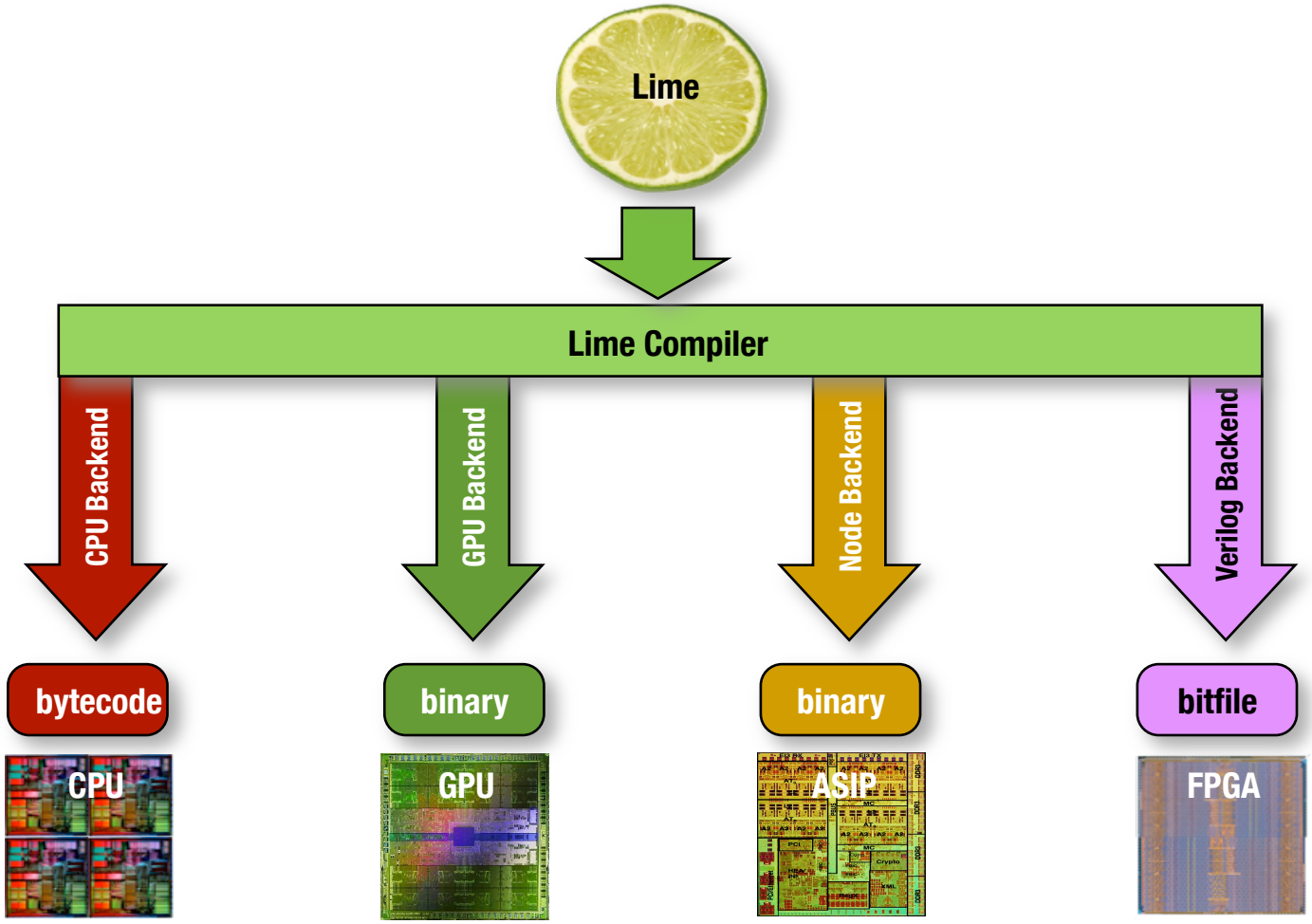
Export Model

Development for customer-owned platform


```
always @(posedge clk_22) begin
    if (reset_23) begin
        pc_30    <=   32'sd0;
        outReady_27    <=   1'd0;
        inReq_25    <=   1'd0;
    end else begin
        case (pc_30)
            0 : begin
                outReady_27    <=   1'd0;
                if (inReady_24) begin
                    inReq_25    <=   1'd1;
                    pc_30    <=   3'd1;
                end
            end
            1 : begin
                inReq_25    <=   1'd0;
                pc_30    <=   3'd2;
            end
            2 : begin
                a_31    <=   inData_26[31:0];
                inReq_25    <=   1'd0;
                pc_30    <=   3'd3;
            end
            3 : begin
                multTmp_32    <=   32'sd2 * a_31;
                pc_30    <=   3'd4;
            end
```

Lime

The Liquid Metal Programming Language



Compiler Philosophy:

- Bytecode backend compiles the full language
- Each device backend compiles a subset of the language
- Compiler gives specific feedback on “exclusions”

What is Lime?

Lime = Java + Isolation + Abstract Parallelism

Isolation: ability to move computation

- **local** keyword
- Immutable types

Abstract Parallelism: freedom to schedule

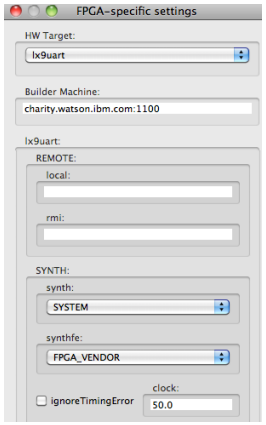
- Stream programming model (**task, =>**)
- Data parallel constructs (**@** map, **!** reduce)

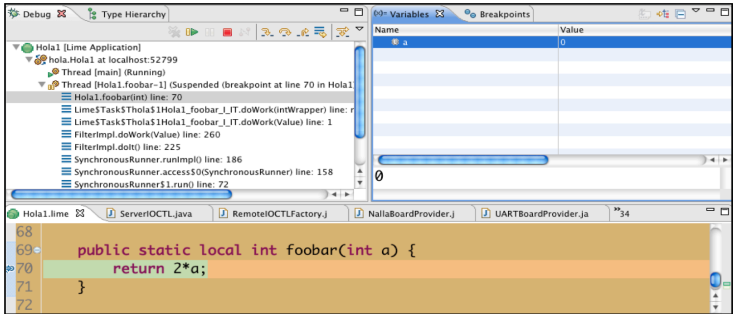
The Lime Development Experience

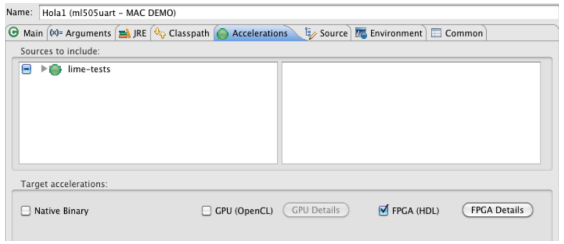
1. **Prototype in standard Java**
   - Lime is a superset of Java
   - **Java-like Eclipse IDE (editors, debugger, navigation)**

2. **Gentle, incremental migration** to parallel Lime code

   Lime language constructs restrict program to safe parallel structures
   - Isolation
   - Immutability
   - Safe Deterministic Parallelism
   - Bounded Space

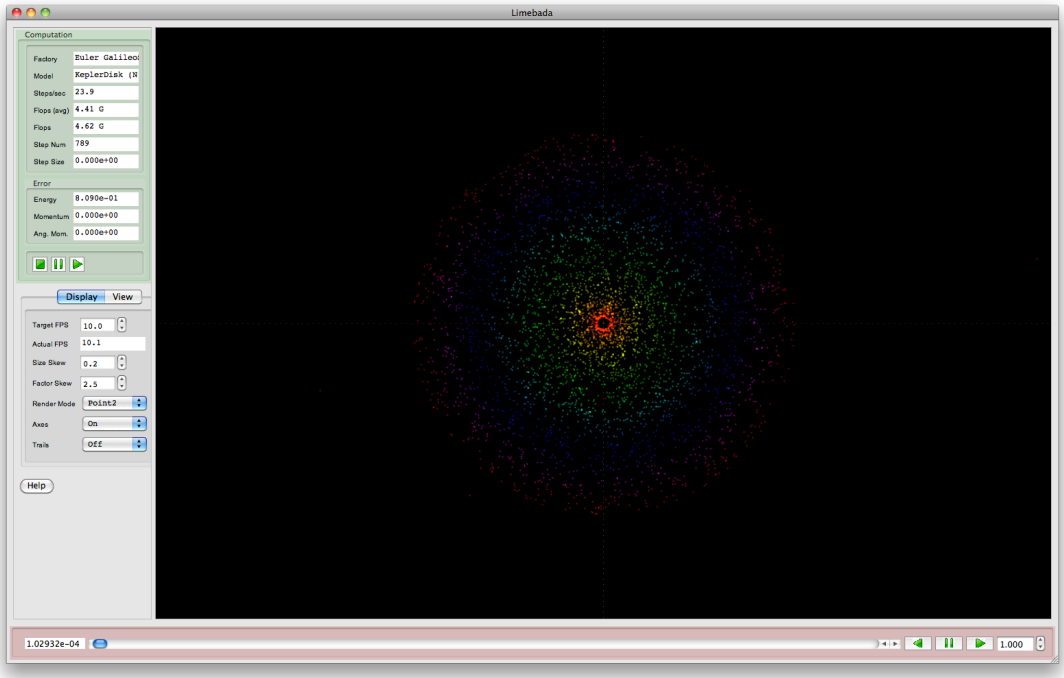






Example Program: N-body simulation

1. Compute forces using particle states
2. Update particles based on forces
3. Periodic checkpoint to disk or display



Task and Connect


```
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; }
   static local float[[][3]] forceComputation(float[[][4]] particles) {…}
   float[[][4]] particleGenerator() {…}
   void forceAccumulator(float[[][3]) {…}
}
```

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; }
   static local float[[][3]] forceComputation(float[[][4]] particles) {…}
   float[[][4]] particleGenerator() {…}
   void forceAccumulator(float[[][3]) {…}
}
```

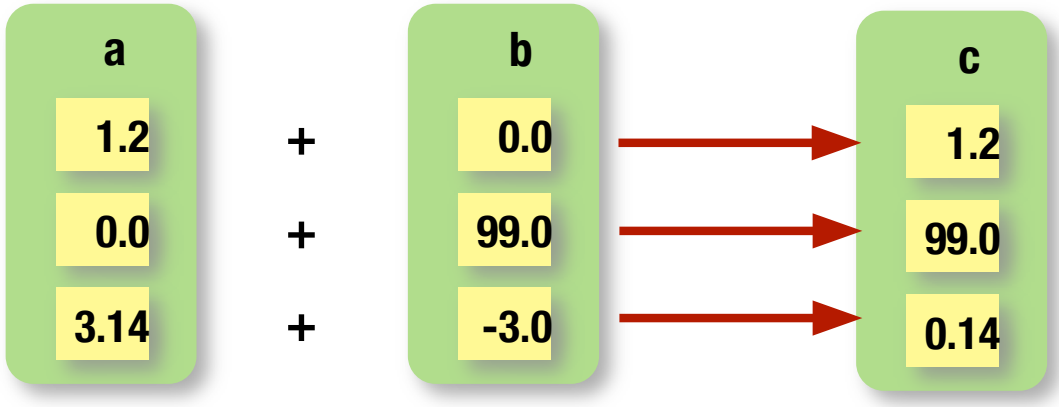
```
// compute the force on each particle
static local float[[][3]] forceComputation(float[[][4]] particles) {
  return @ force(particles, #particles); // map operator @
}

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

Language constructs guarantee safe, deterministic, parallel execution semantics

- Deeply immutable types (e.g. float[[]])
- Isolation (e.g. local)
- Data parallel function application (map, @)

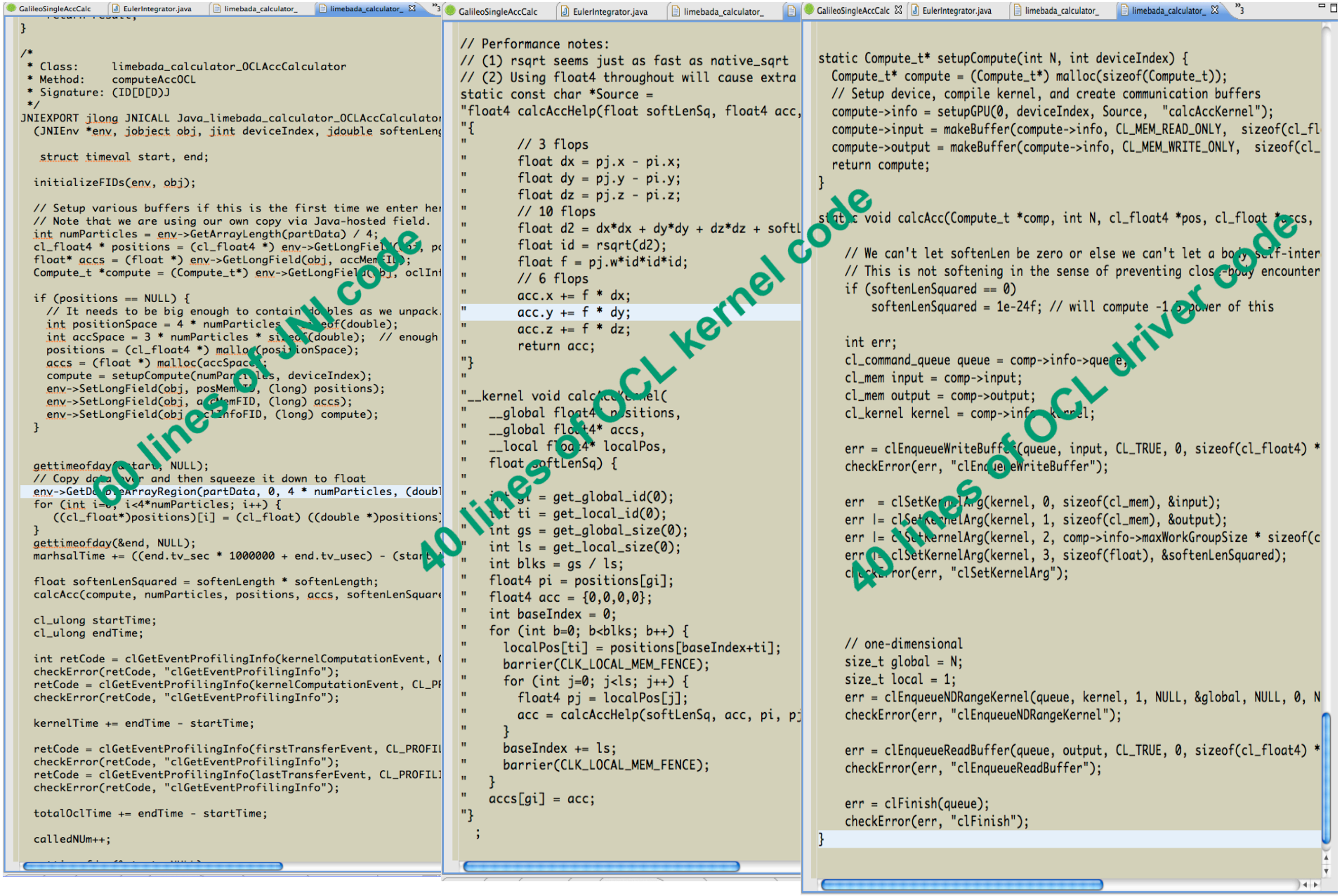
Data Parallelism via map (@)

```
float[] a = ...;
float[] b = ...;
float[] c = a @+ b; // map operator + over 2 arrays
```




```
static local float add(float x, float y) { return x + y; }
float[] c = @add(a, b); // map function add() over 2 arrays
float[] d = @add(a, #1); // map function add() over 1 array, 1 scalar
```

What didn't you write? (GPU)



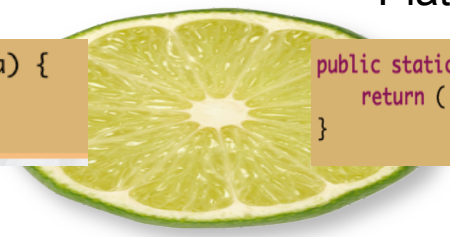
What didn't you write? (FPGA)

User Logic

```
public static local int foobar(int a) {
    return 2*a;
}
```

Platform runtime and communication

```
public static Task makeTask1() {
    return ( task Hola1().source => task Hola1.foobar => task Hola1.printer );
}
```



```
always @(posedge clk_22) begin
    if (reset_23) begin
        pc_30   <=   32'sd0;
        outReady_27   <=   1'd0;
        inReq_25   <=   1'd0;
    end else begin
        case (pc_30)
            0 : begin
                outReady_27   <=   1'd0;
                if (inReady_24) begin
                    inReq_25   <=   1'd1;
                    pc_30   <=   3'd1;
                end
            end
            1 : begin
                inReq_25   <=   1'd0;
                pc_30   <=   3'd2;
            end
            2 : begin
                a_31   <=   inData_26[31:0];
                inReq_25   <=   1'd0;
                pc_30   <=   3'd3;
            end
            3 : begin
                multTmp_32   <=   32'sd2 * a_31;
                pc_30   <=   3'd4;
            end
```

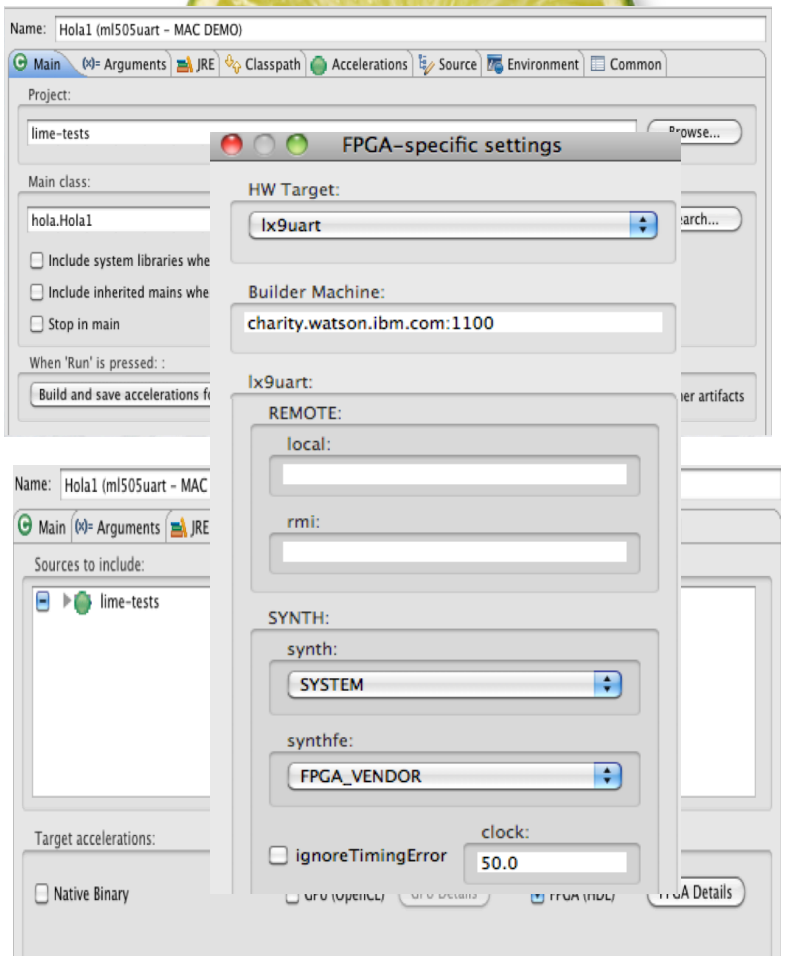
HDL

3000 lines of platform independent HDL
500 lines of platform dependent HDL
3 months of development time
??? months of debugging time

Lime Developer Tools

Compiling the Code (FPGA)

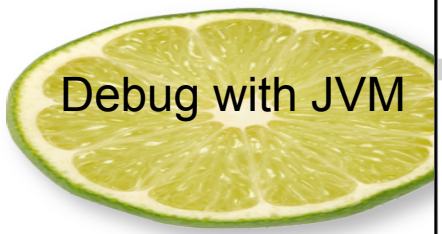
Integrated IDE



Multiple 3rd party tools with long workflow

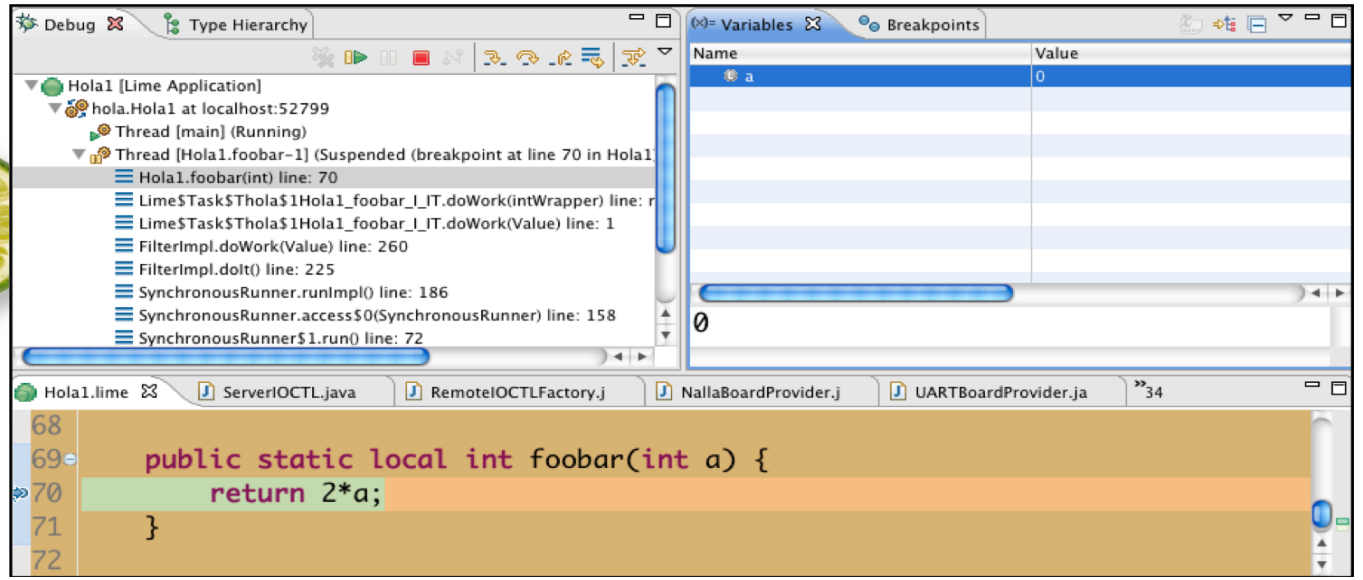
1. Create an ISE® or Quartus® project
2. Write HDL and testbench
3. Simulate design
4. Repeat 2 & 3 until functionality is achieved
5. Implement your top module
6. Follow advanced design guides
7. Modify the design as necessary, simulate, synthesize, and implement until design requirements are met.
8. Run timing simulations, verify timing closures
9. Program your Xilinx® or Altera® device

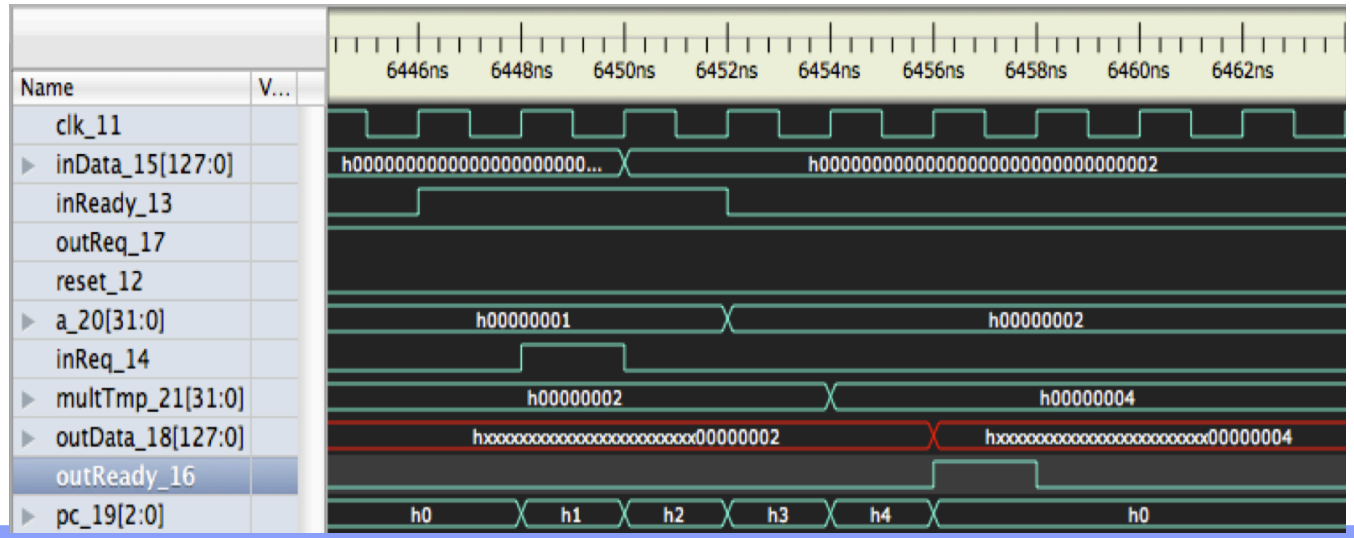
Debugging



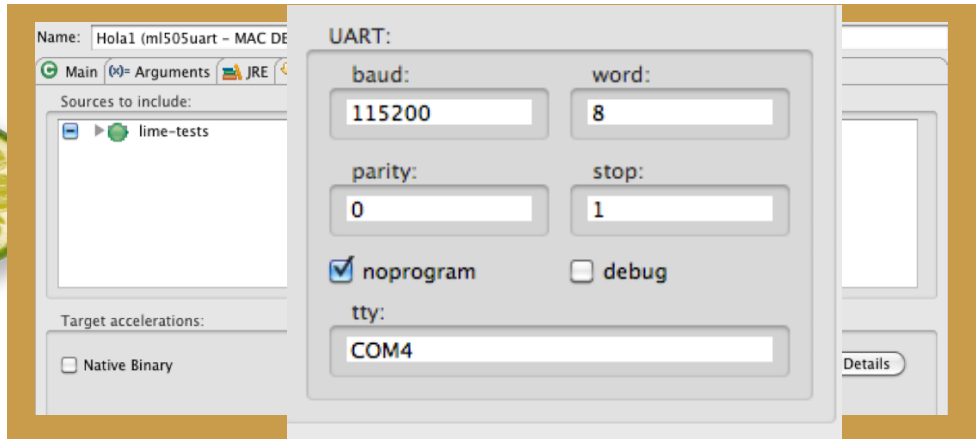
vs.

Low-level Waveforms



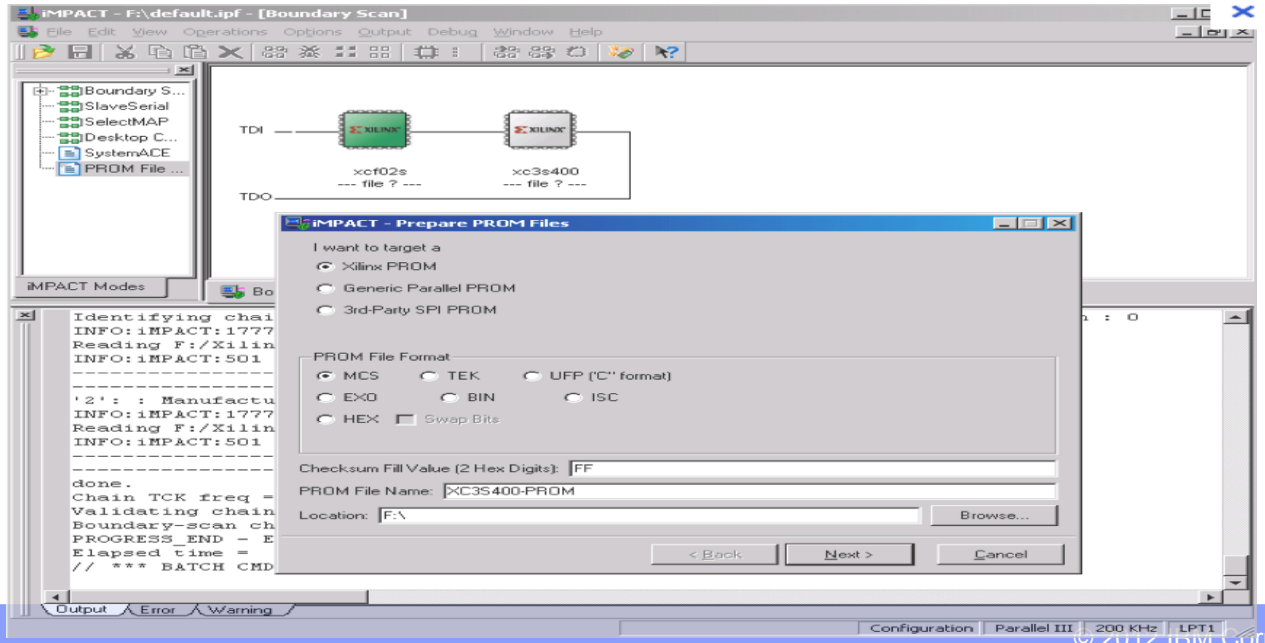


Running the Code

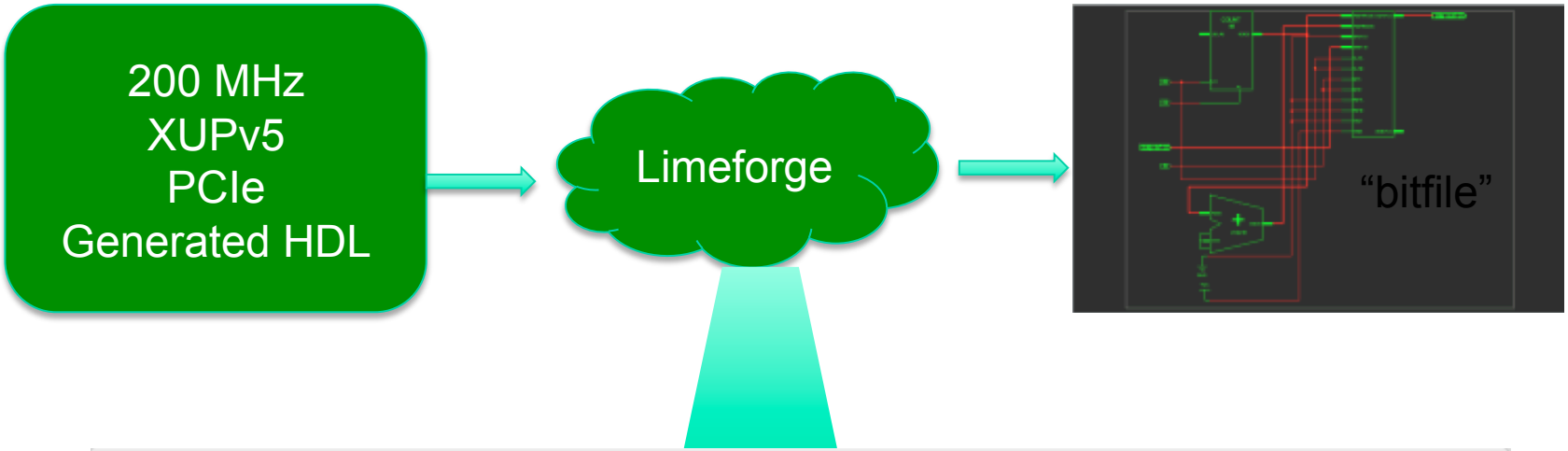


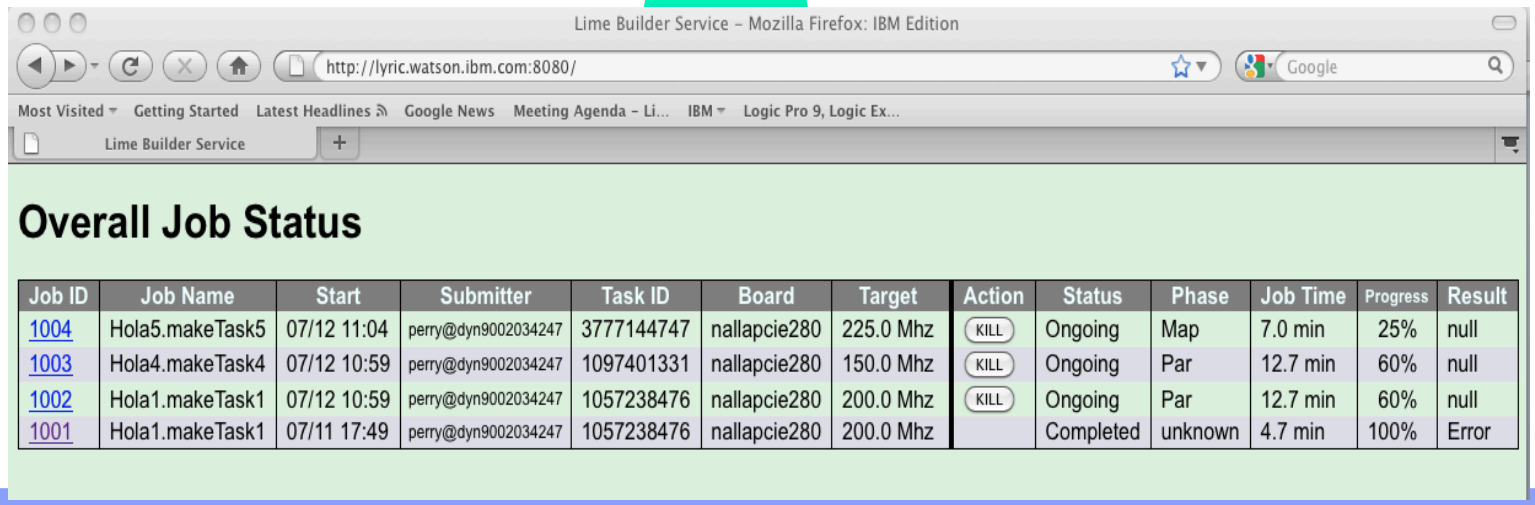
vs.

Yet Another Tool



Limeforge (Synthesis in the Cloud)

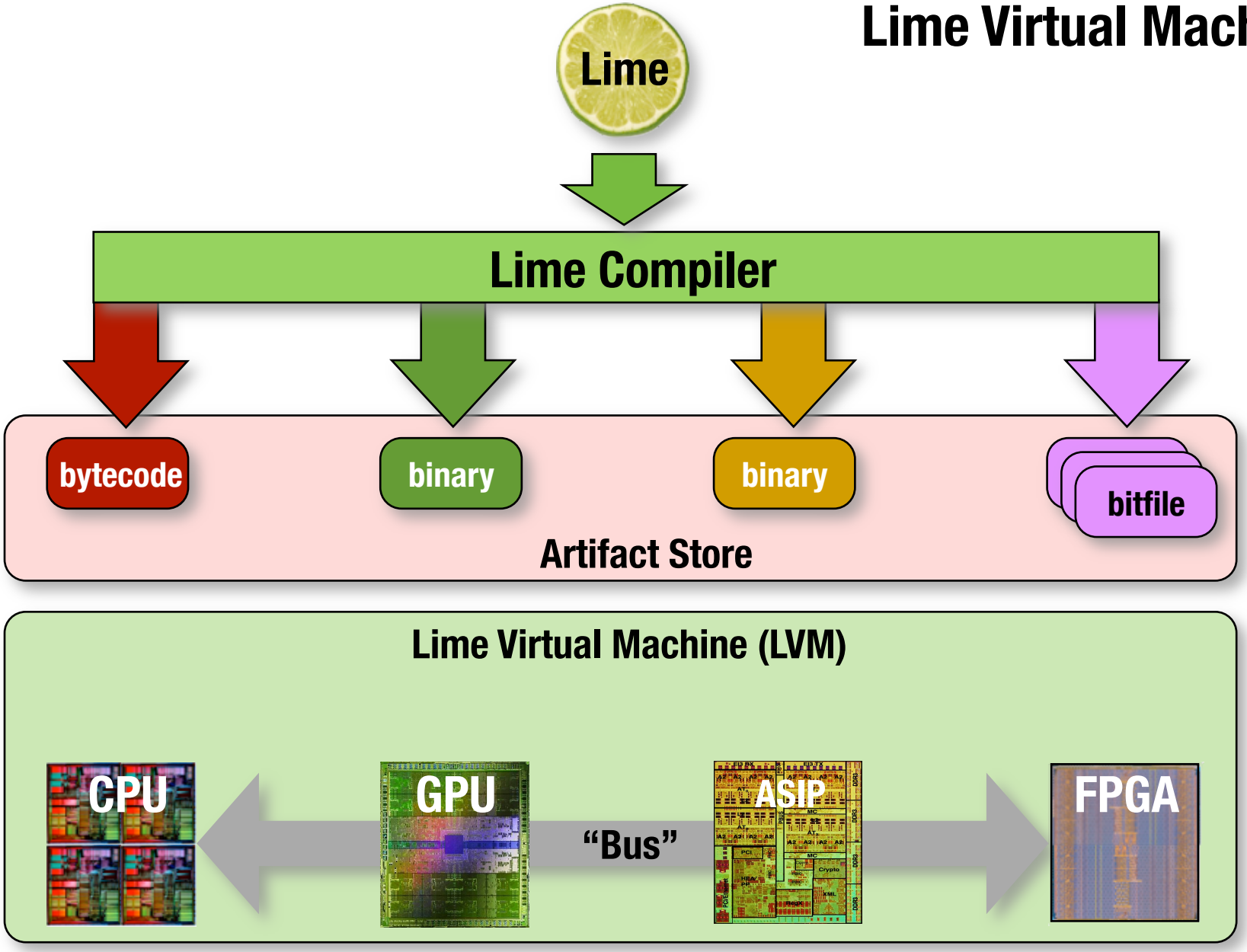




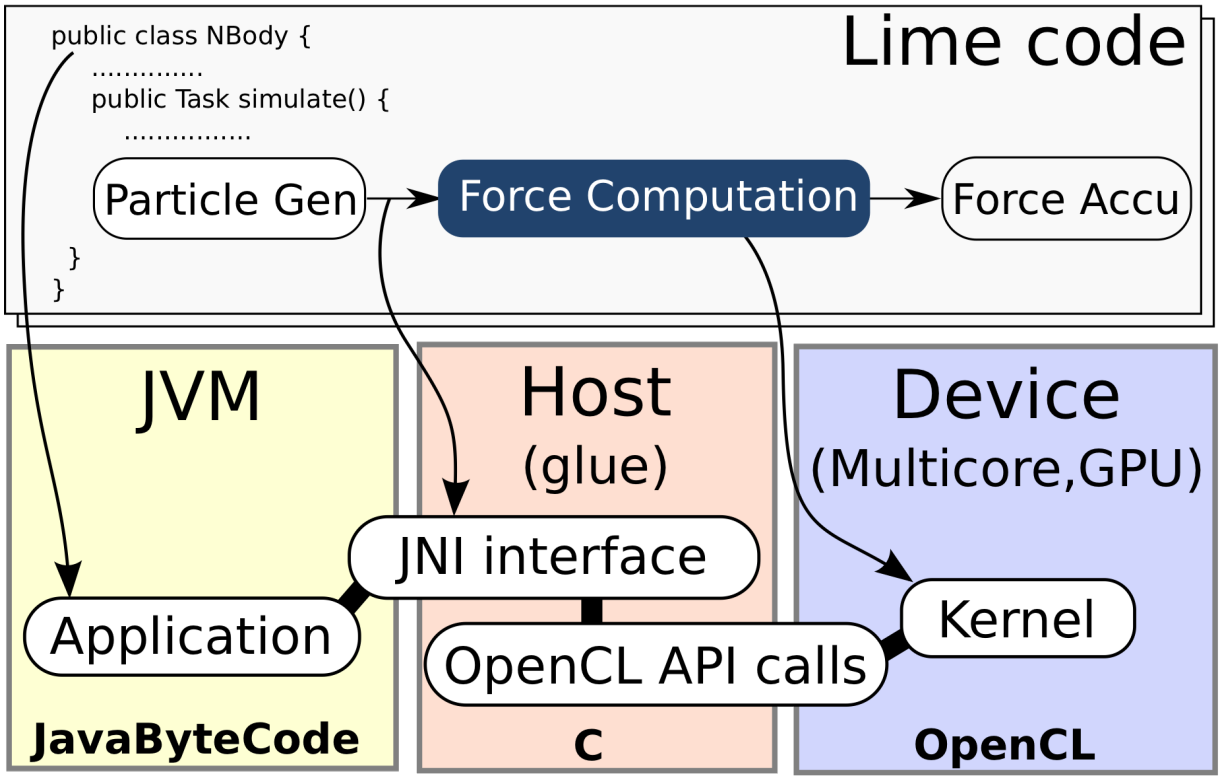
Job ID	Job Name	Start	Submitter	Task ID	Board	Target	Action	Status	Phase	Job Time	Progress	Result
1004	Hola5.makeTask5	07/12 11:04	perry@dyn9002034247	3777144747	nallapcie280	225.0 Mhz	KILL	Ongoing	Map	7.0 min	25%	null
1003	Hola4.makeTask4	07/12 10:59	perry@dyn9002034247	1097401331	nallapcie280	150.0 Mhz	KILL	Ongoing	Par	12.7 min	60%	null
1002	Hola1.makeTask1	07/12 10:59	perry@dyn9002034247	1057238476	nallapcie280	200.0 Mhz	KILL	Ongoing	Par	12.7 min	60%	null
1001	Hola1.makeTask1	07/11 17:49	perry@dyn9002034247	1057238476	nallapcie280	200.0 Mhz		Completed	unknown	4.7 min	100%	Error

Lime Implementation

Lime Virtual Machine



GPU Backend Structure



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?

Current status (bottom line)

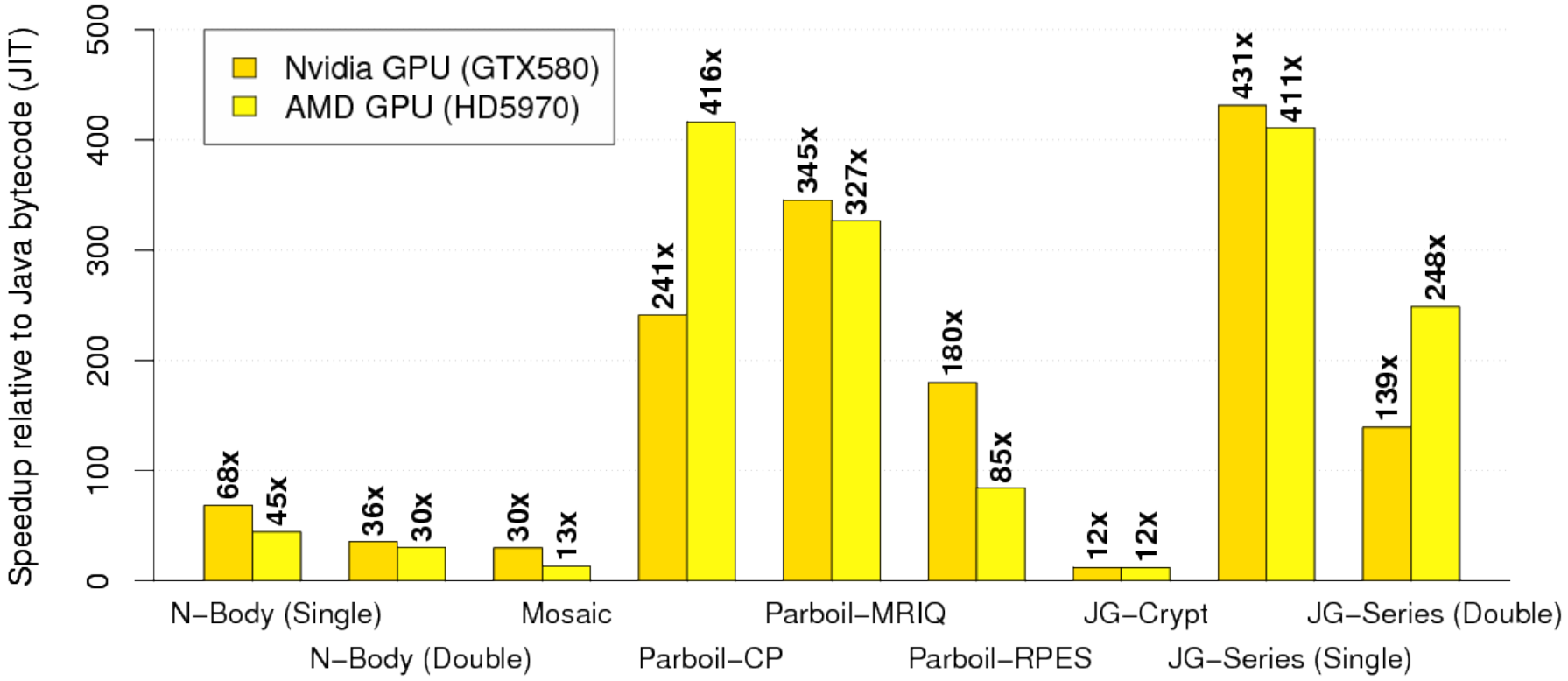
No magical super-compiler

- Some benchmarks Lime performs as well as hand-coded alternative
- Some, not.
- Communication/computation ratio dominates

Ongoing research into better implementation technology

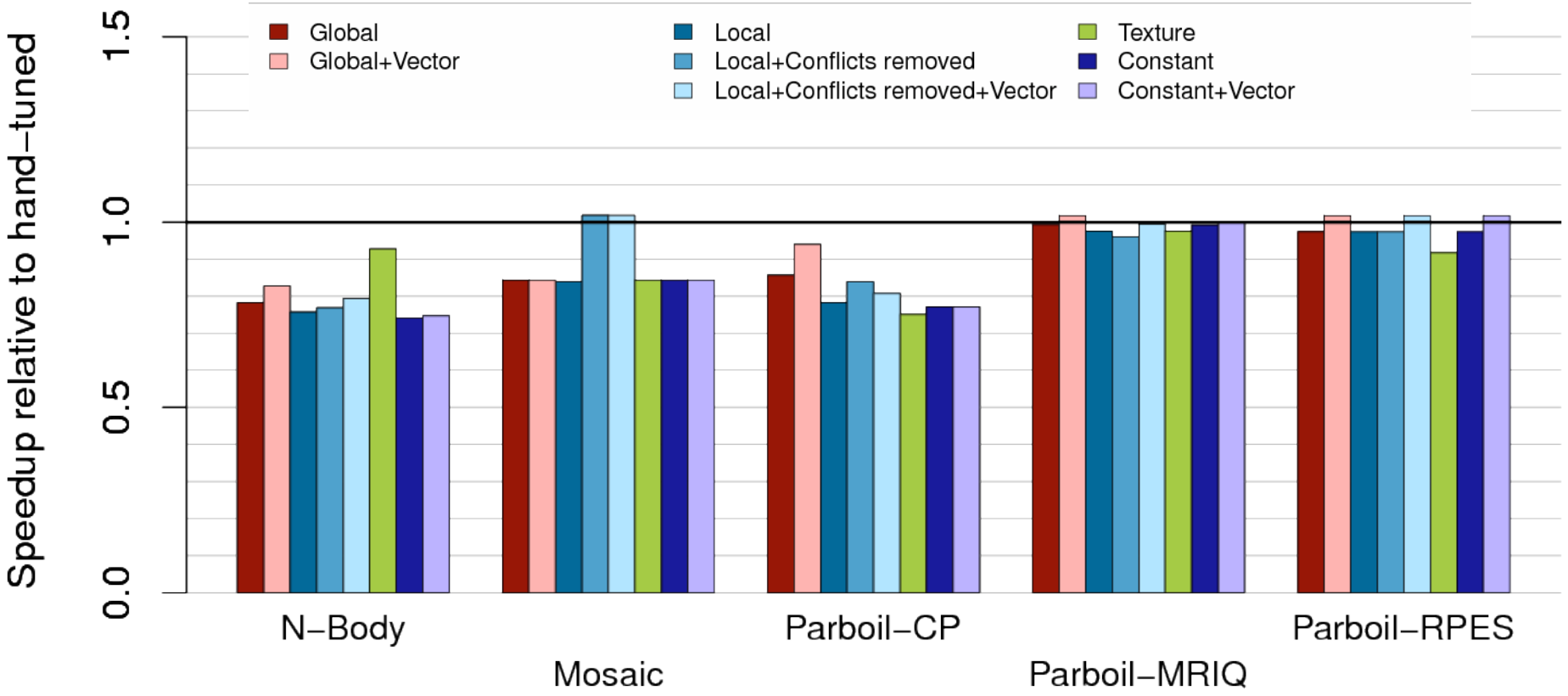
- Big performance improvements in reach
- Good enough for prime time? We're optimistic.

Speedup vs. Java [PLDI '12]



end-to-end performance (all overhead included)

Speedup vs. Hand-Tuned OpenCL [PLDI '12] (GTX 580)



end-to-end performance (all overhead included)

Lime FPGA Results

- **Combinatorial circuits: match hand-coded**
  - pipelining
- **Complex circuits: goal is 2-4x worse, now ≥ 10x**
  - I/O pipelining and classic compiler optimizations
- **Increasing synthesizable language feature set**
  - First of its kind FPGA garbage collector

Liquid Metal Summary

- ✓ High-level, concise abstractions for parallelism
- ✓ Gentle, incremental migration from Java
- ✓ Rich Eclipse-based tool support
- ✓ General solution for programming heterogeneous systems

⚠ No magic
- parallel algorithm design is not easier
- one version of a program may not run well on all devices
- vanilla Java programs will not run on accelerator

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

Backup

Export Model

Development for customer-owned platform


```
always @(posedge clk_22) begin
    if (reset_23) begin
        pc_30    <=   32'sd0;
        outReady_27    <=   1'd0;
        inReq_25    <=   1'd0;
    end else begin
        case (pc_30)
            0 : begin
                outReady_27    <=   1'd0;
                if (inReady_24) begin
                    inReq_25    <=   1'd1;
                    pc_30    <=   3'd1;
                end
            end
            1 : begin
                inReq_25    <=   1'd0;
                pc_30    <=   3'd2;
            end
            2 : begin
                a_31    <=   inData_26[31:0];
                inReq_25    <=   1'd0;
                pc_30    <=   3'd3;
            end
            3 : begin
                multTmp_32    <=   32'sd2 * a_31;
                pc_30    <=   3'd4;
            end
```