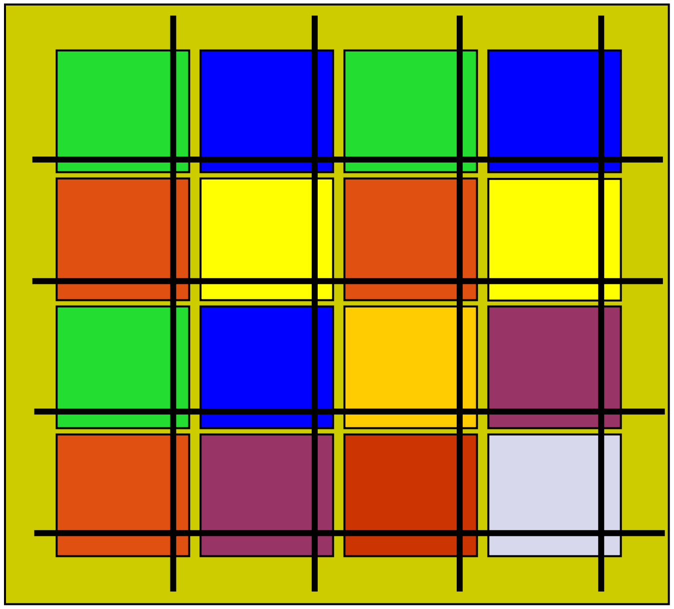
Liquid Metal

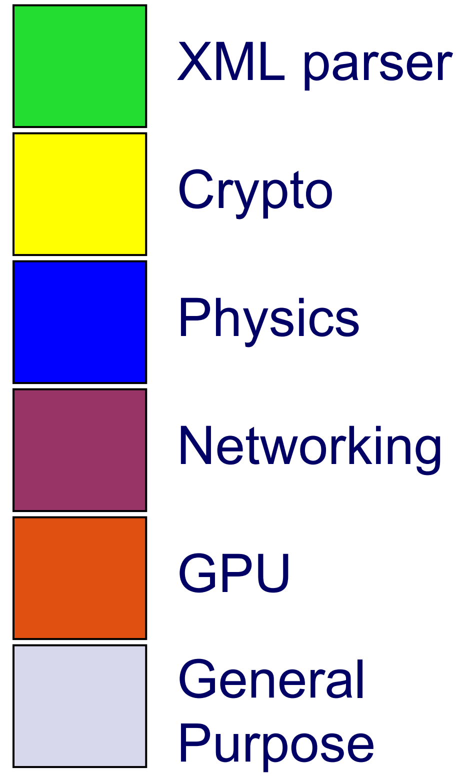
Blurring the Boundary between Software and Hardware for Versatile Parallel Computing

Rodric Rabbah
IBM Research
T. J. Watson

rodric@gmail.com

The Lure of Heterogeneous Architectures



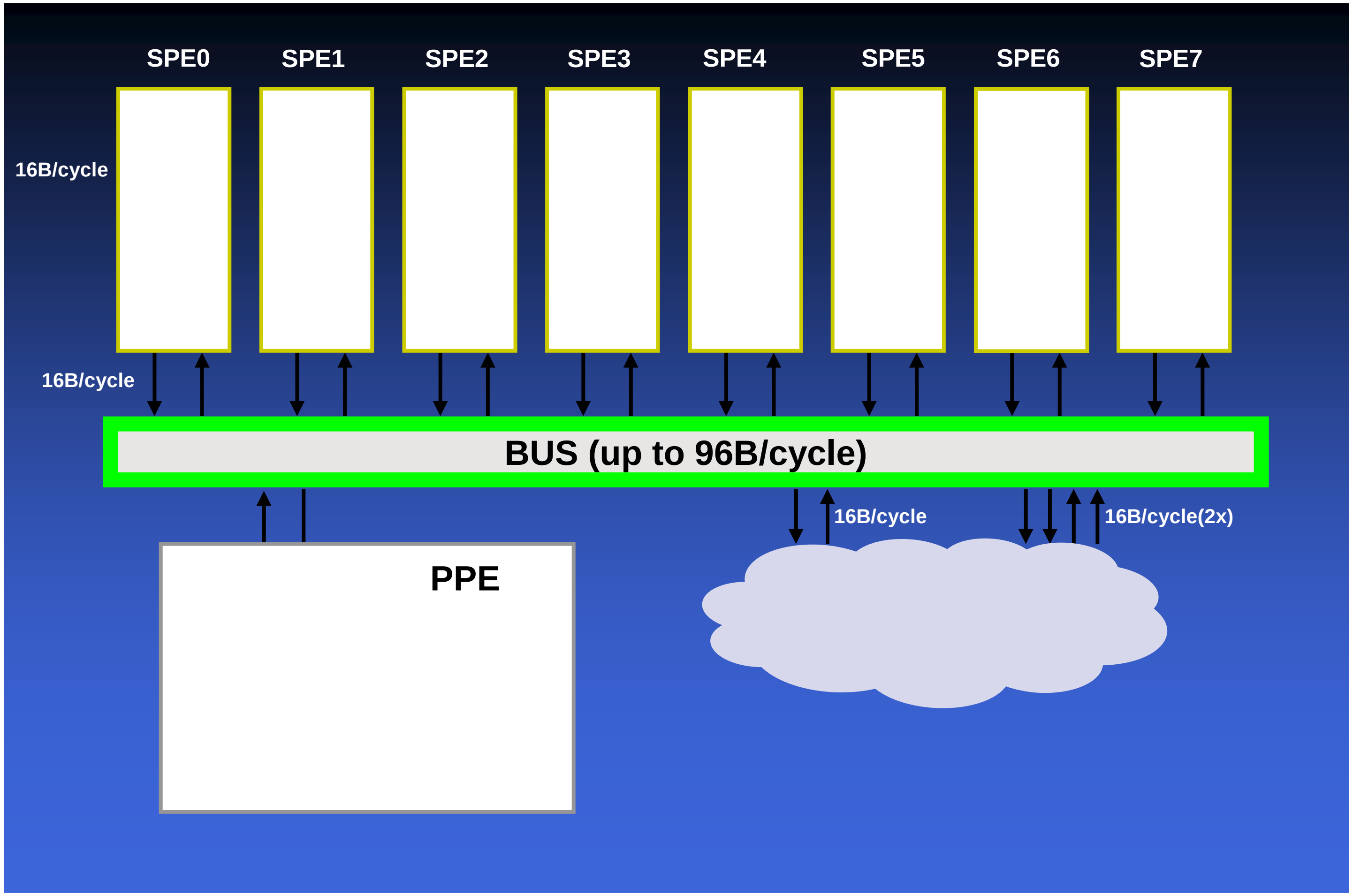


- Transistors are free
  - Many custom cores on a single chip
- Custom IP and fixed function accelerators
  - Lower power and better performance

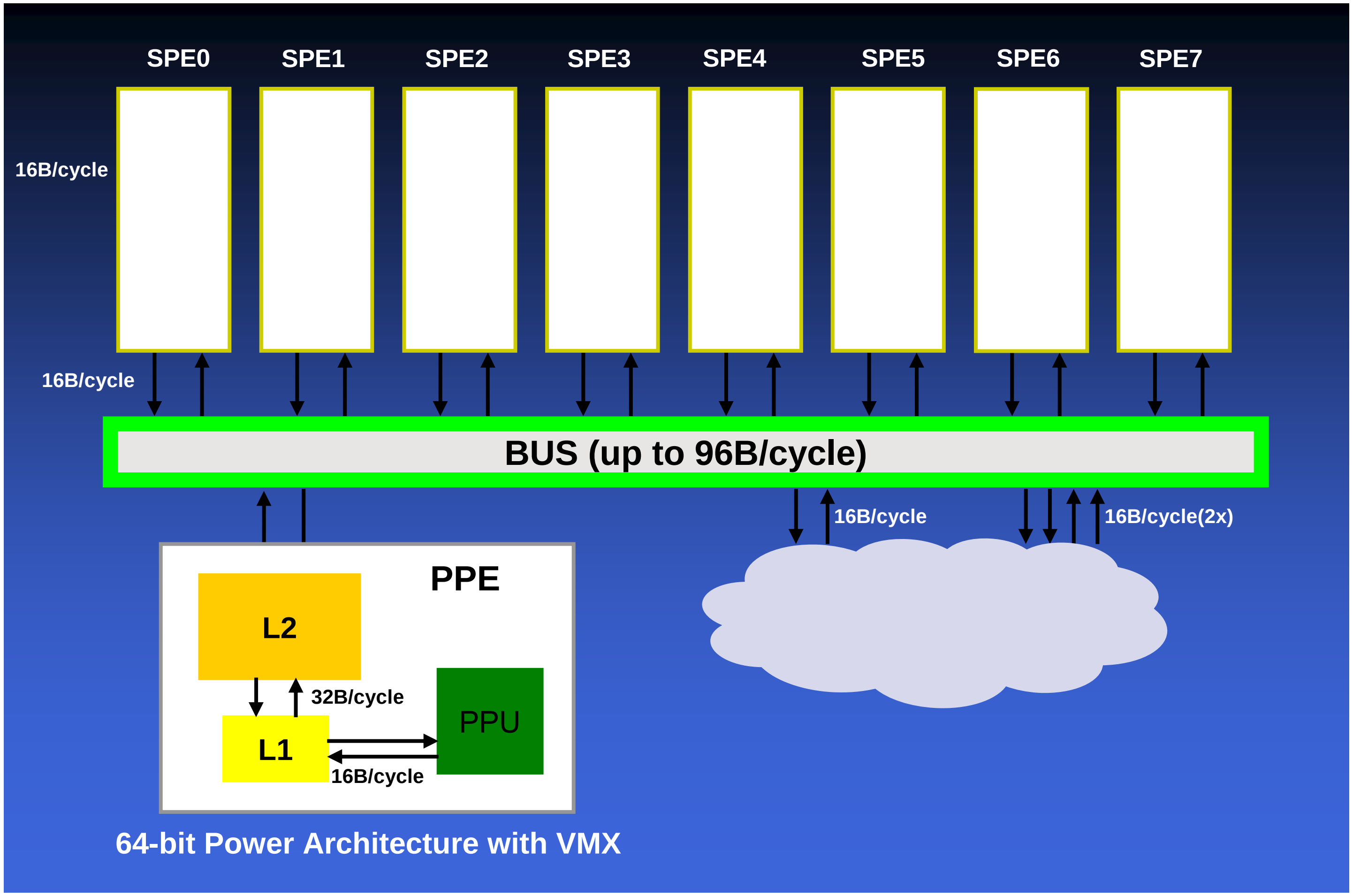
A Look at the Cell Architecture

9-core Heterogeneous Architecture for Streaming, Multimedia, and HPC

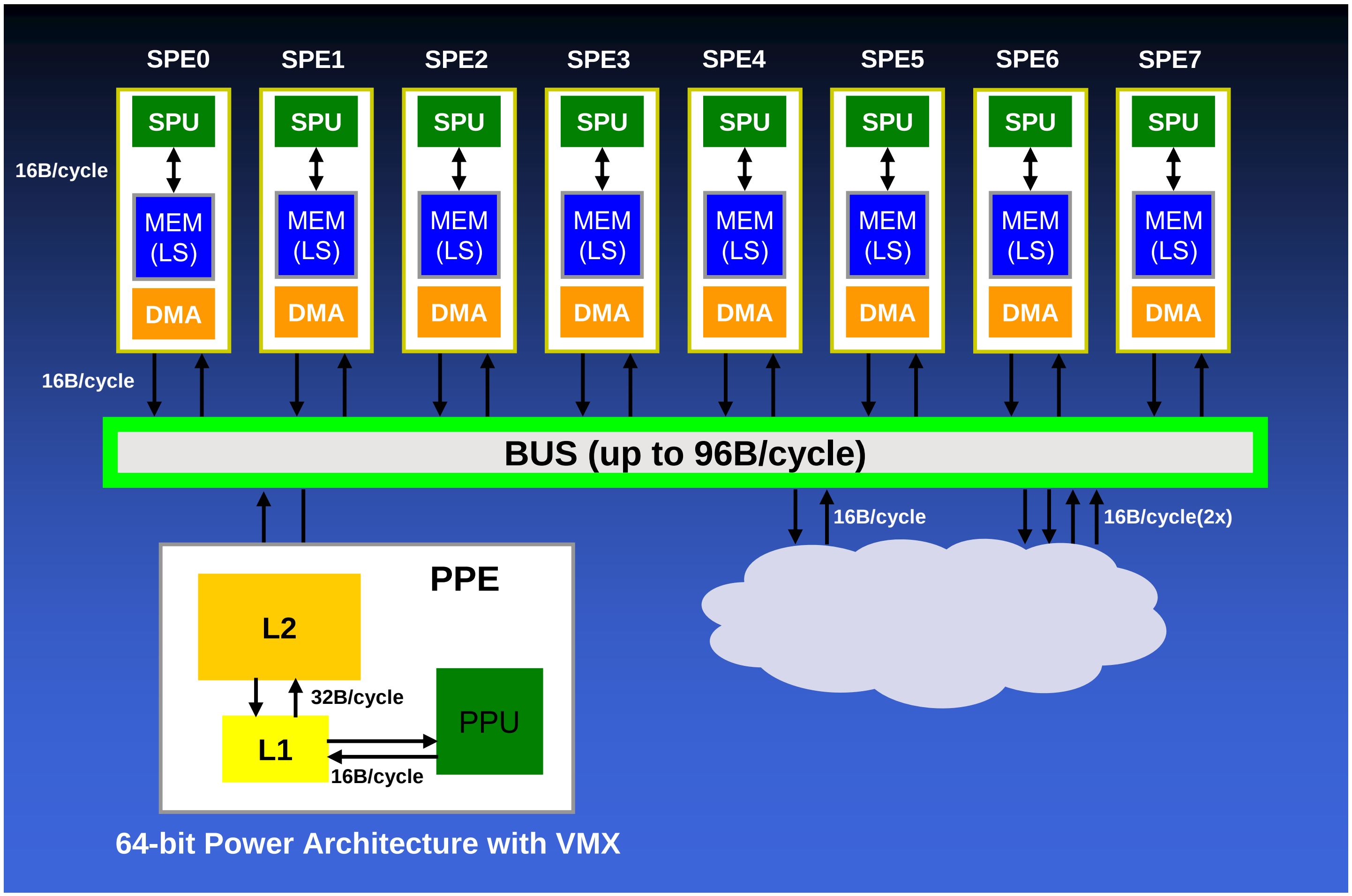
Cell Broadband Engine Architecture



Cell Broadband Engine Architecture



Cell Broadband Engine Architecture



SPU is a dual issue SIMD Processor

LS is a 256KB local mem for code and data

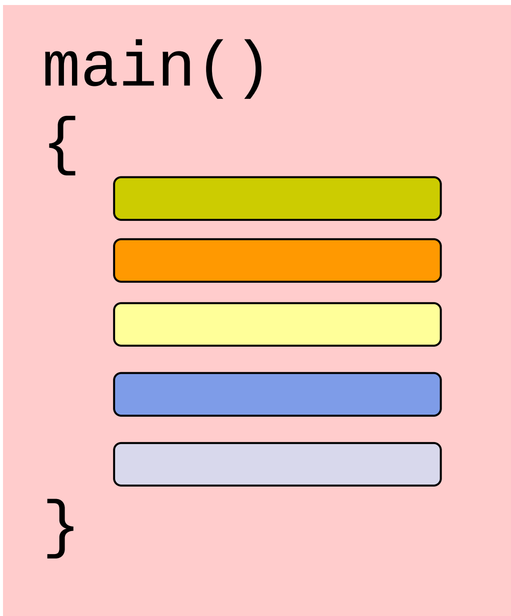
Cell Programming: The Art

Mapping partition an application to run on SPEs vs PPEs
Communication SPE can only directly access its local memory… data is DMA-ed in and out of local memory explicitly
Synchronization coordination between SPEs and PPE
Local Store packing SPE memory is finite, no HW virtualization
SIMD constant factor speedup to single “thread” performance

Cell Programming: The Challenge

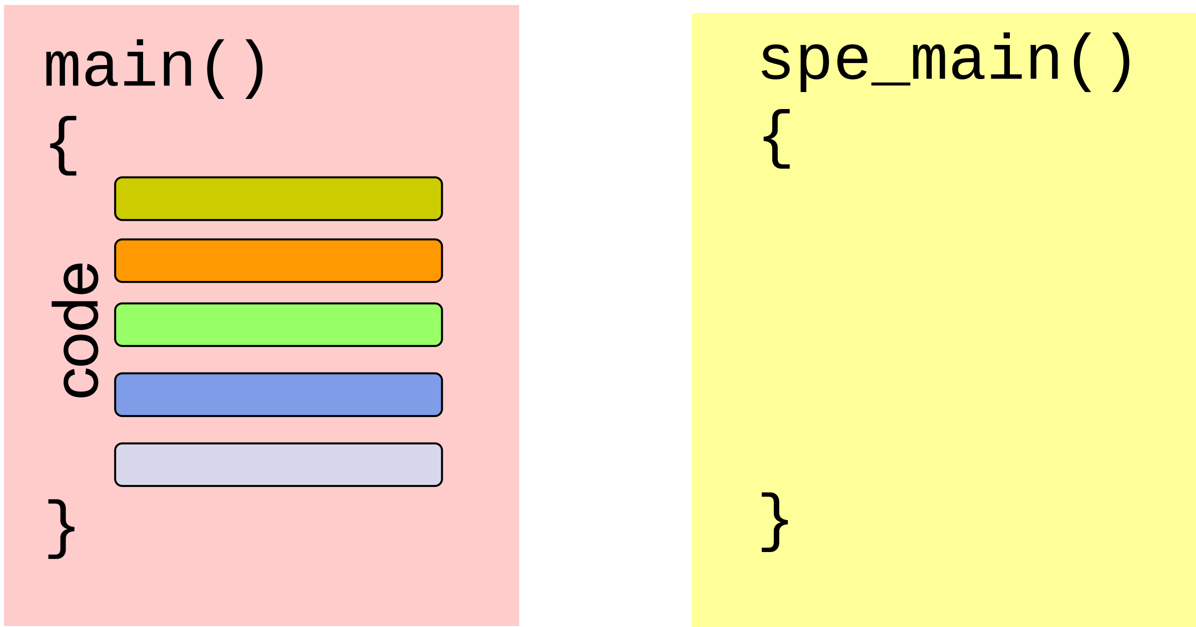
Mapping partition an application to run on SPEs vs PPEs	explicit parallelism, locality, load balancing
Communication SPE can only directly access its local memory… data is DMA-ed in and out of local memory explicitly	compute-DMA concurrency
Synchronization coordination between SPEs and PPE	deadlock, races
Local Store packing SPE memory is finite, no HW virtualization	double buffering, overflow
SIMD constant factor speedup to single “thread” performance	intrinsics, data alignment

Cell Programming Basics



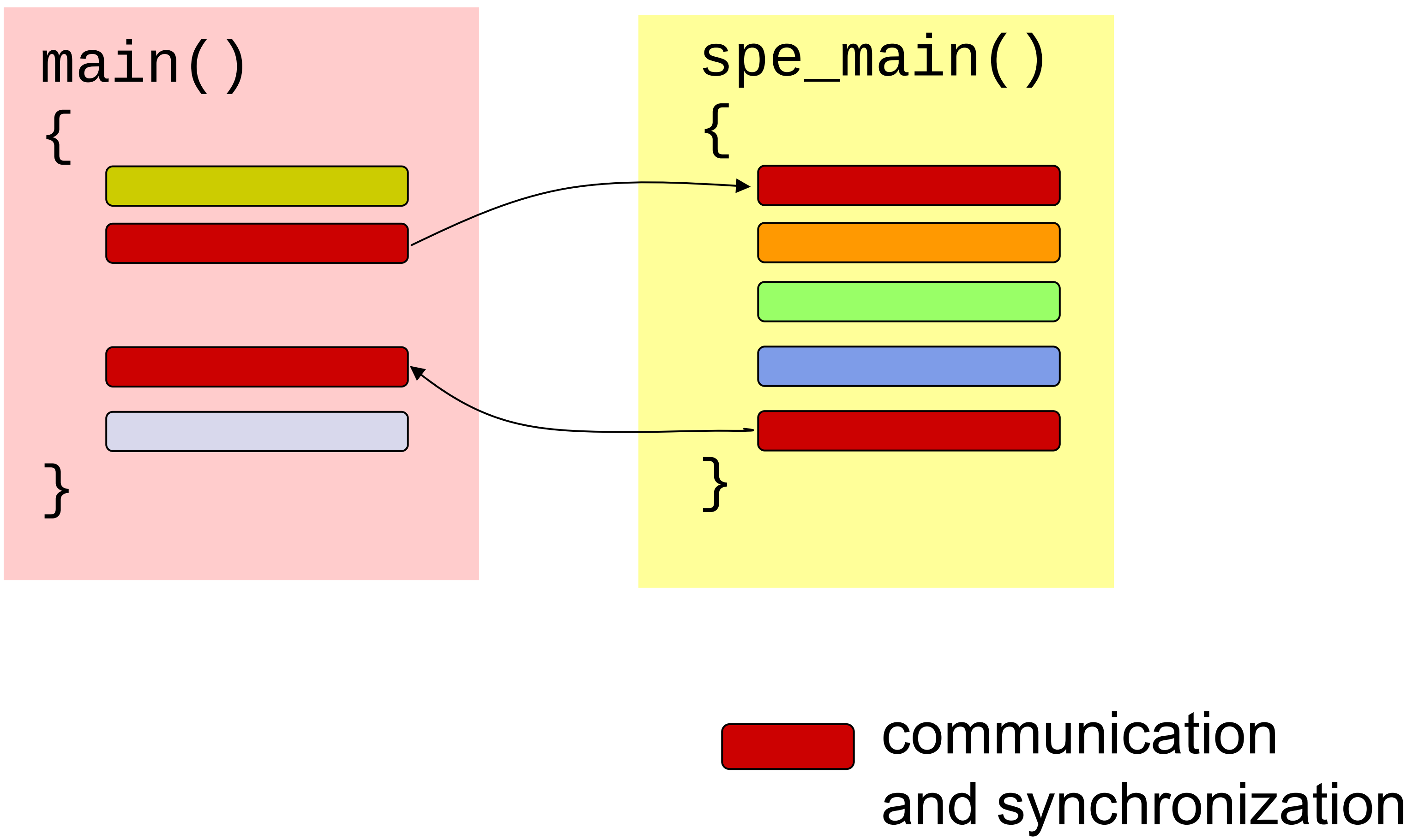
Cell Programming Basics

- Two programs: one for PPE, another for SPEs



Cell Programming Basics

- Two programs: one for PPE, another for SPEs



A Simple Cell Program

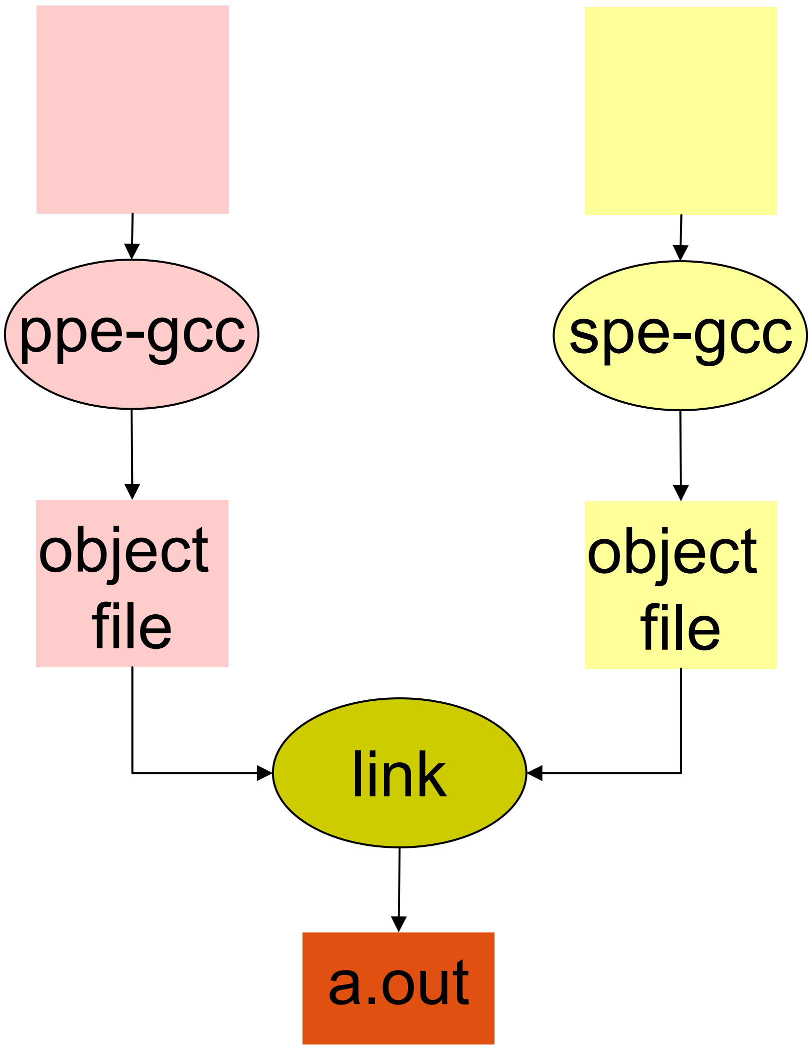
PPE (hello.c)

```
#include <stdio.h>
#include <libspe.h>

extern spe_program_handle_t hello_spe;

int main() {
  speid_t id[8];

  // Create 8 SPU threads
  for (int i = 0; i < 8; i++) {
    id[i] = spe_create_thread(0,
                              &hello_spe,
                              NULL,
                              NULL,
                              -1,
                              0);
  }

  // Wait for all threads to exit
  for (int i = 0; i < 8; i++) {
    spe_wait(id[i], NULL, 0);
  }

  return 0;
}
```

SPE (hello_spe.c)

```
#include <stdio.h>

int
main(unsigned long long speid,
     unsigned long long argp,
     unsigned long long envp)
{
  printf("Hello world! (0x%x)\n", (unsigned int)speid);
  return 0;
}
```

Cell Programming Basics



- Separate tool chains including compilers and debuggers
- Substantial fraction of the code is for orchestration communication and synchronization
- In summary: **not a productive process**

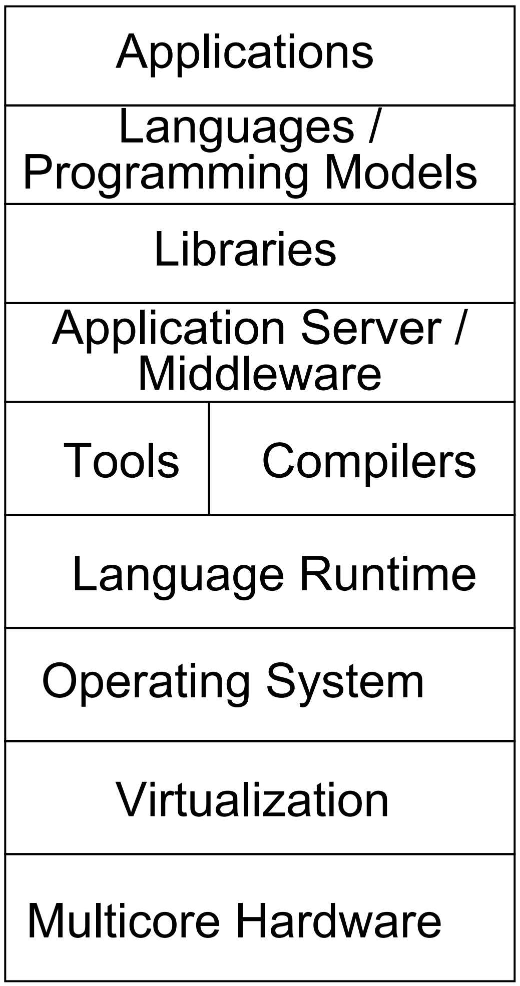
- Experience with Cell has demonstrated that good programming models are no longer optional in the face of ubiquitous parallelism

The Productivity Challenge

- Programmer controls every detail of parallelism

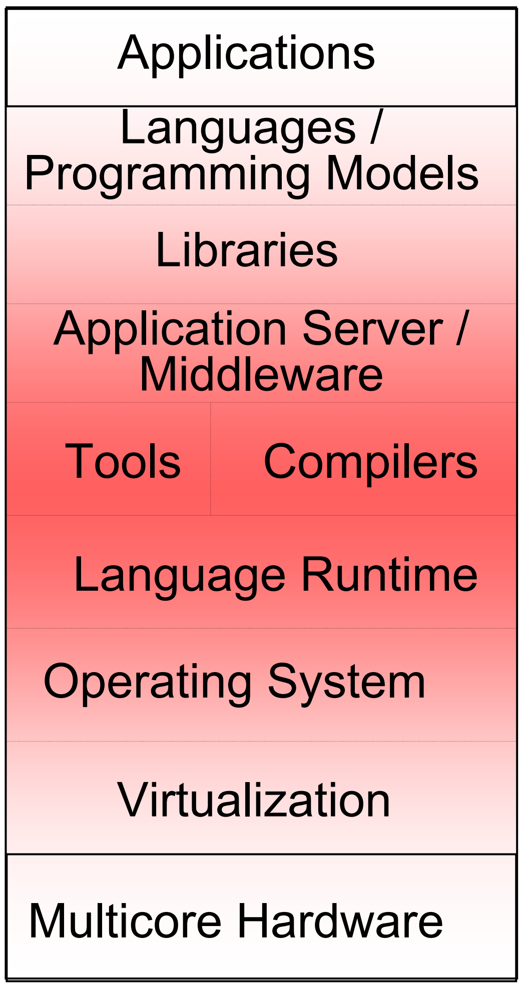
- Granularity decisions
  - If too small, lots of synchronization and thread creation
  - If too large, bad locality
- Load balancing decisions
  - Create balanced parallel sections (not data-parallel)
  - Profiling is a challenge
- Locality decisions
  - Code and data co-partitioning
  - Placement for sharing and optimized communication
- Synchronization decisions
  - Barriers, atomicity, critical sections, order, flushing, races, deadlocks
- Determinism nearly impossible
  - Debugging is heroic

Parallelism Affects Every Layer of the Stack



- Many layers of abstraction facilitated evolution of computation for many years
  - Hide details at each layer
  - Enable componentization
  - Threat of interchanging components in a layer creates healthy incentive for improvements
- Now, the many layers of abstractions are an increasing impediments to innovation
  - Trends to add more layers
    (JVM, App server, OS virtualization)
  - Thin interfaces lead to poor synergy and a lot of redundancy
    (JVM, OS, Virtualization, HW all present a thread abstraction)

Must Blur Boundaries Between Layers



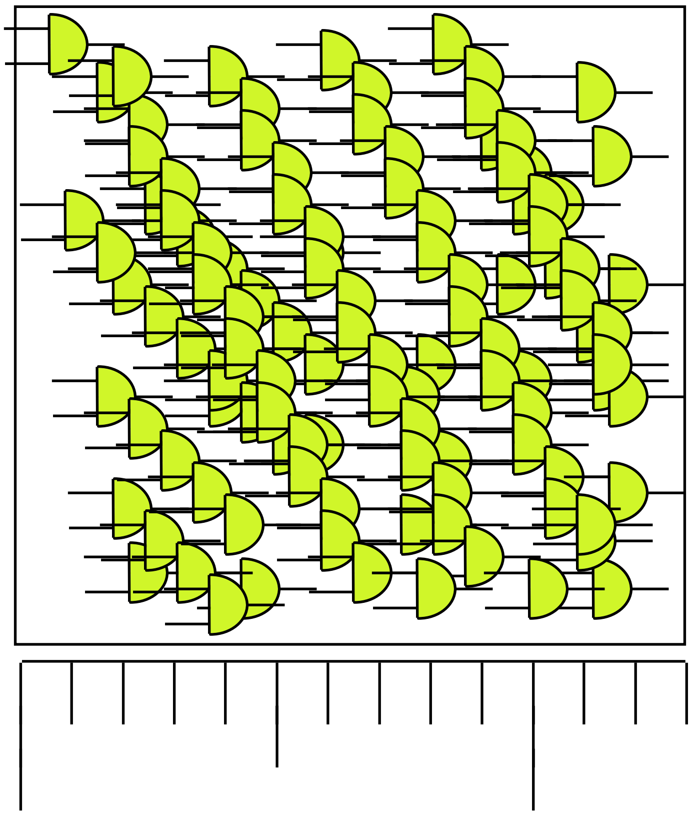
- Provide customization at every level
- Promote cooperation and synergy

- **Lesson from BlueGene playbook:** BlueGene has its own stack with large performance boost from working across layers

Toward Productive Programming for Future Architectures

A Hardware Designer's Perspective

- How is computation coordinated over billions of transistors?



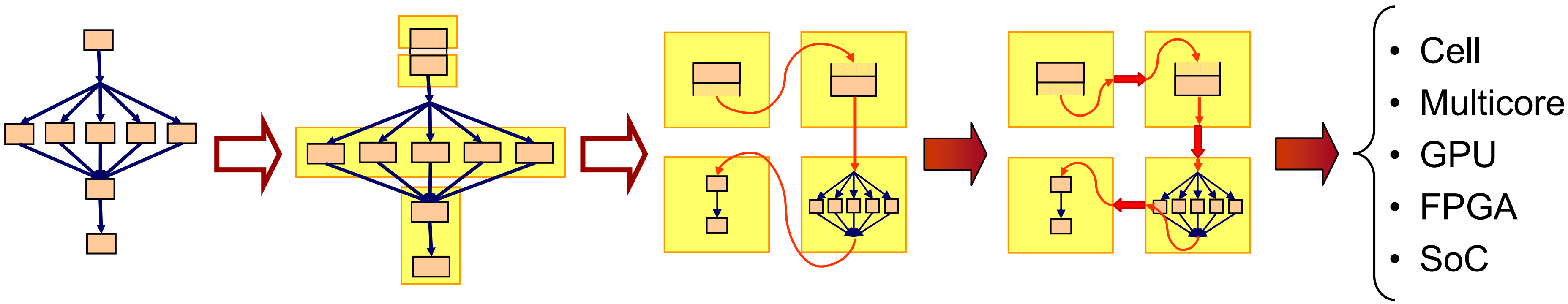
- Impose structure
- Specify behavioral
- Partition
- Place
- Route
- …

The Basics of Programming Multicores

Today's Architectures = Parallel Computers

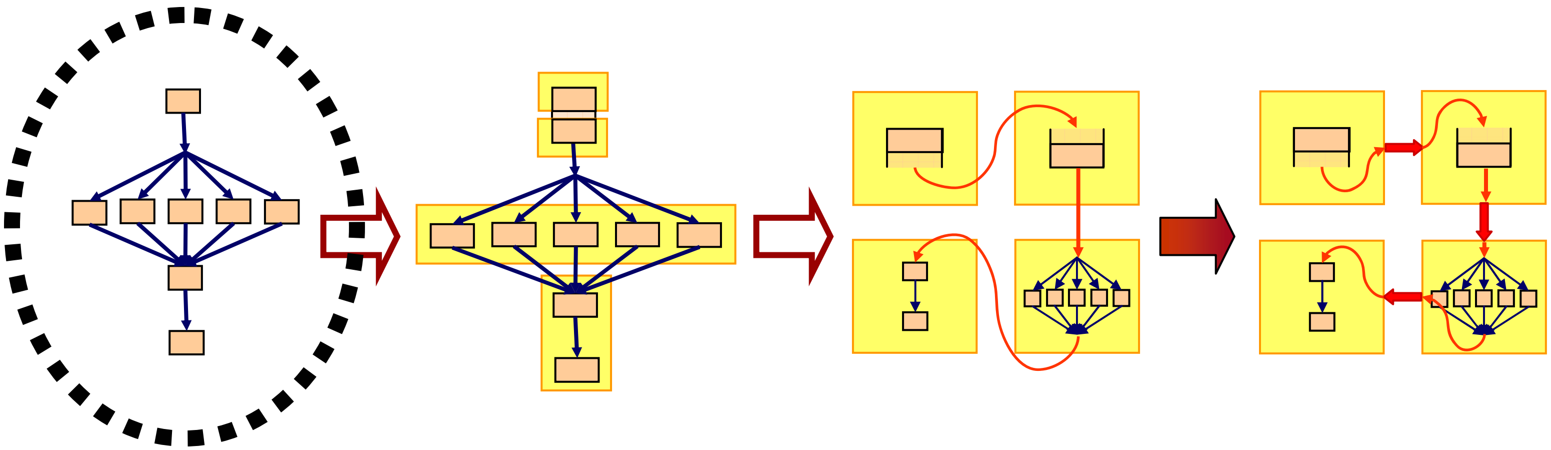
"A parallel computer is a collection of processing elements that cooperate and communicate to solve large problems fast."

- Programming becomes an exercise in partitioning, placement, routing and scheduling



Partition **Place (layout)** **Route and Schedule** **Codegen**

Toward Productive Programming for Future Architectures

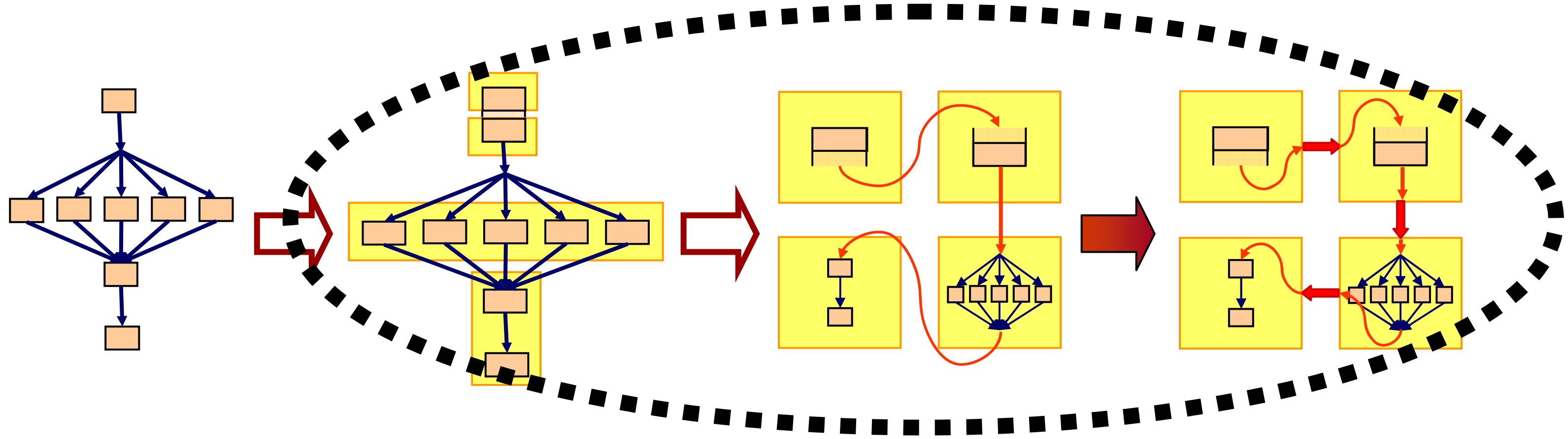


Programming Model Challenges

- Encapsulate computation
  - State updates are explicit
  - No sharing of data except through well **defined** interfaces
- Make communication explicit

- **In a single unified semantically rich programming model** for general purpose, streaming, real time, bit level…

Toward Productive Programming for Future Architectures



Programming Model Challenges

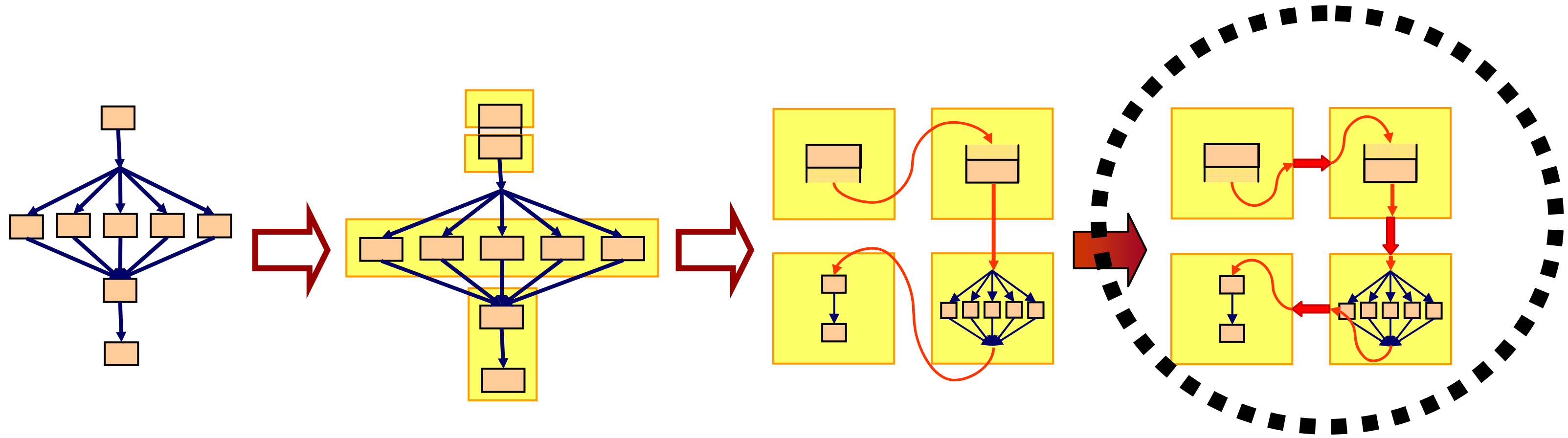
- Encapsulate computation
  - State updates are explicit
  - No sharing of data except through well **defined** interfaces
- Make communication explicit

- **In a single unified semantically rich programming model** for general purpose, streaming, real time, bit level…

Compiler Challenges

- Automate the rest

Toward Productive Programming for Future Architectures



Programming Model Challenges

- Encapsulate computation
  - State updates are explicit
  - No sharing of data except through well **defined** interfaces
- Make communication explicit

- **In a single unified semantically rich programming model** for general purpose, streaming, real time, bit level…

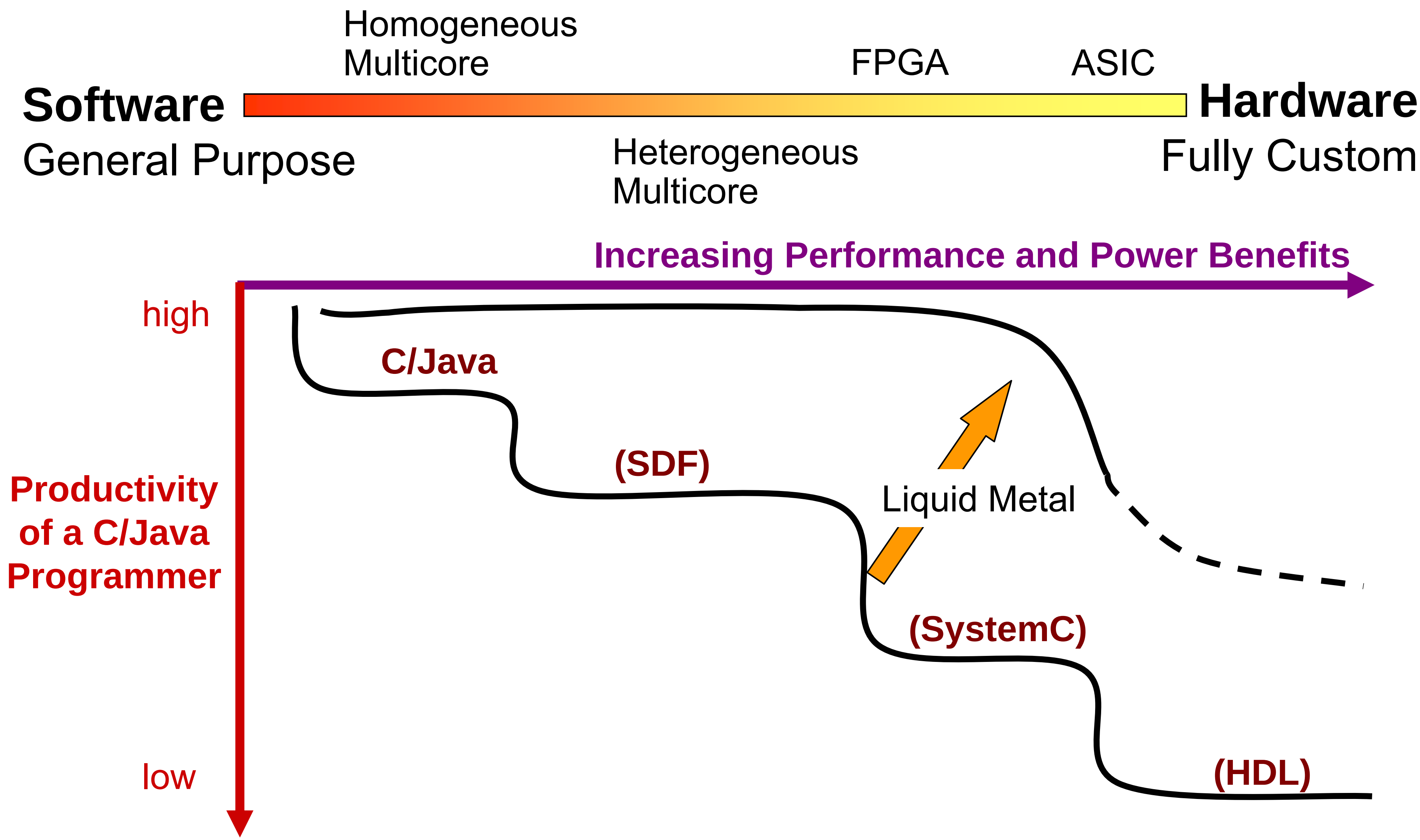
Compiler Challenges

- Automate the rest

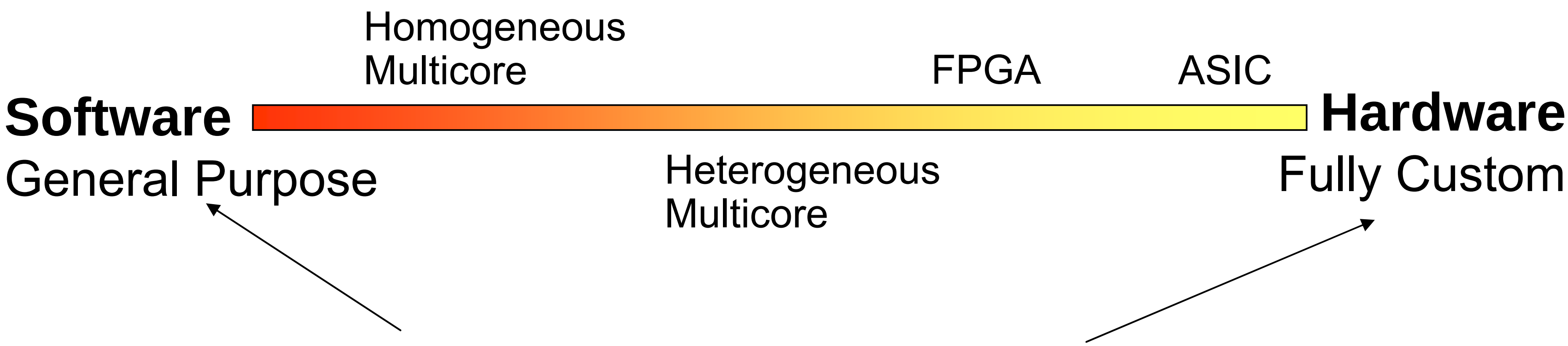
Non trivial issues to solve related to runtime system especially with heterogeneous architectures

- E.g., different clock domains

Toward Productive Programming for Future Architectures



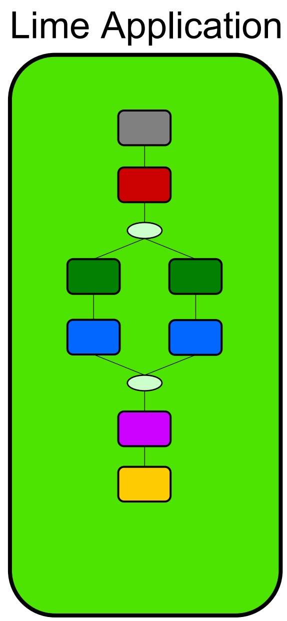
Liquid Metal



- Liquid Metal tackle challenges at the extremes
- Language, Compiler and Runtime for programming software and hardware
- Raise level of abstraction for software/hardware co-design

- Program hardware (with new functionality) at a level of abstraction comparable to Java
- Object Oriented programming across the software/hardware boundary

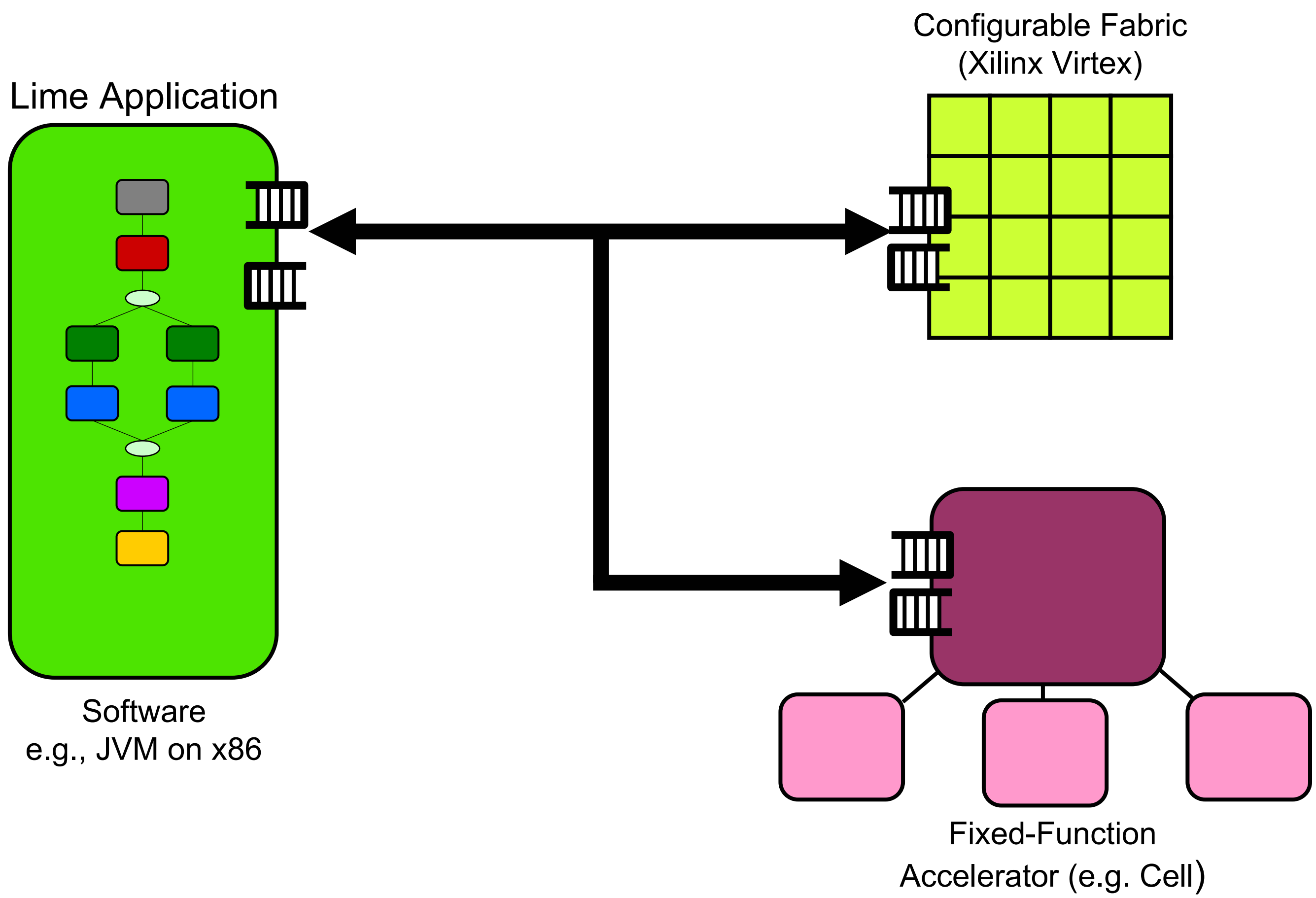
Liquid Metal (Lime)



- Java-based language for programming software or hardware
- Mode-less programming model
- Dataflow driven, real time aware
- Composable and malleable code
- Portable, run anywhere with equivalent semantics

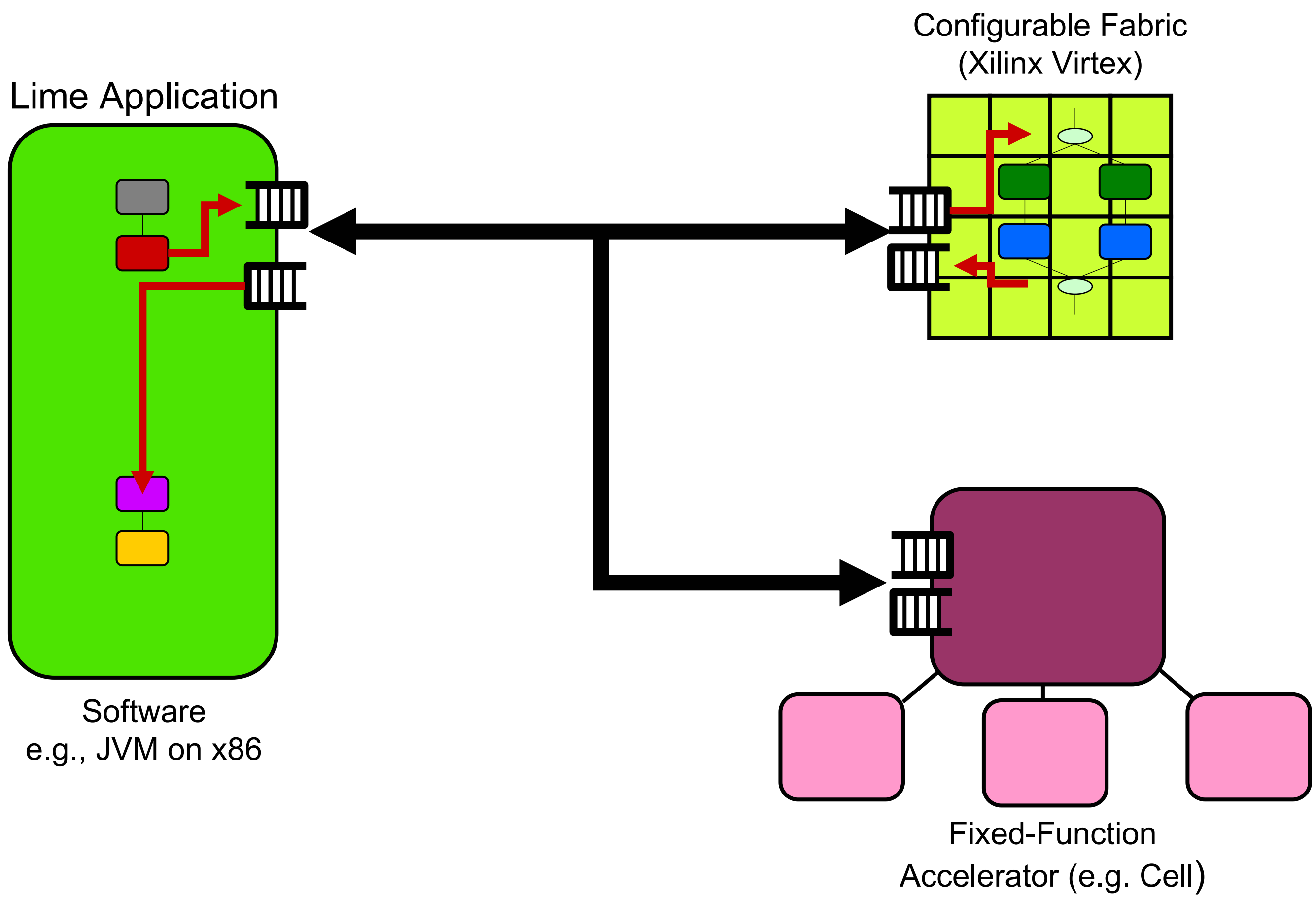
Liquid Metal Runtime

- Run in a JVM or compile to hardware (FPGA)



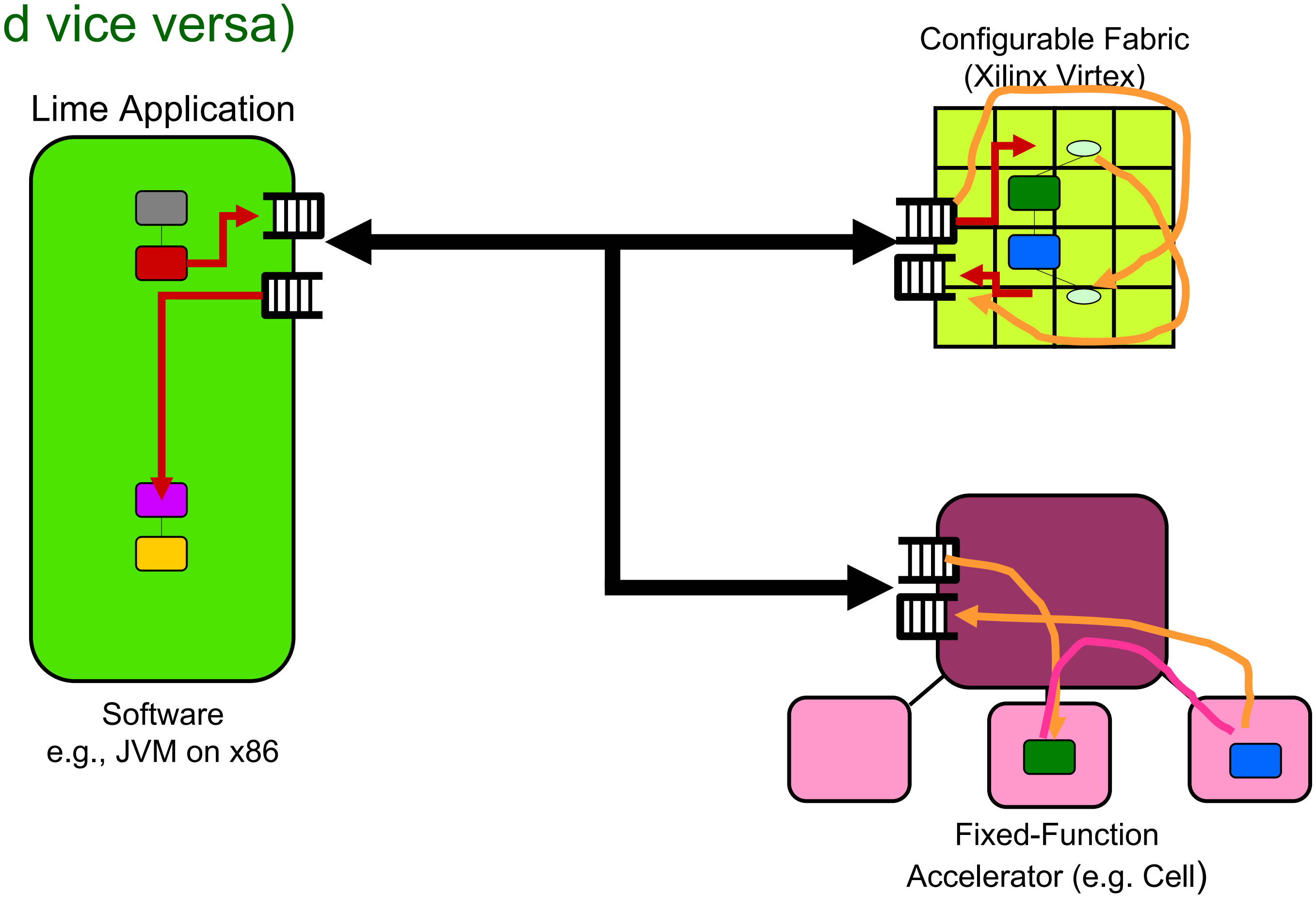
Liquid Metal Runtime

- Run in a JVM or compile to hardware (FPGA)



Liquid Metal Runtime

- Fluidly move computation from hardware to software (and vice versa)



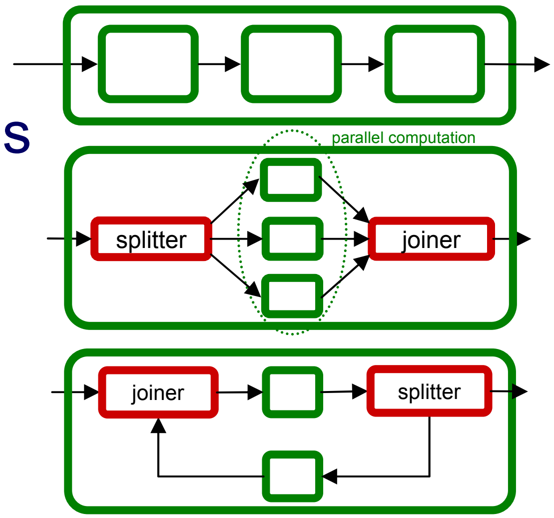
Language at Micro-scale: Functional and Data-parallel Constructs

- Comprehensive value type system
  - All “primitive” types user-defined
  - Efficient, abstract, vectorizable, and synthesizable
- Atomic types
  - Simplified transactional memory
- Parallel Atomics
  - Deterministic, race-free data-parallel construct
  - Easy to express, understand, debug

Macro-scale:
Isolated Classes with Timing

- Lime classes are special classes with actor-like semantics
  - Can not read/write non-final global state
    - Functional in input and current state
  - Can be instantiated in controlled contexts
    - Controlled aliasing allows precise scheduling
  - Mutation of class state is exposed and controlled

- Algorithmic and programmatic assembly of classes into computational dataflow graphs

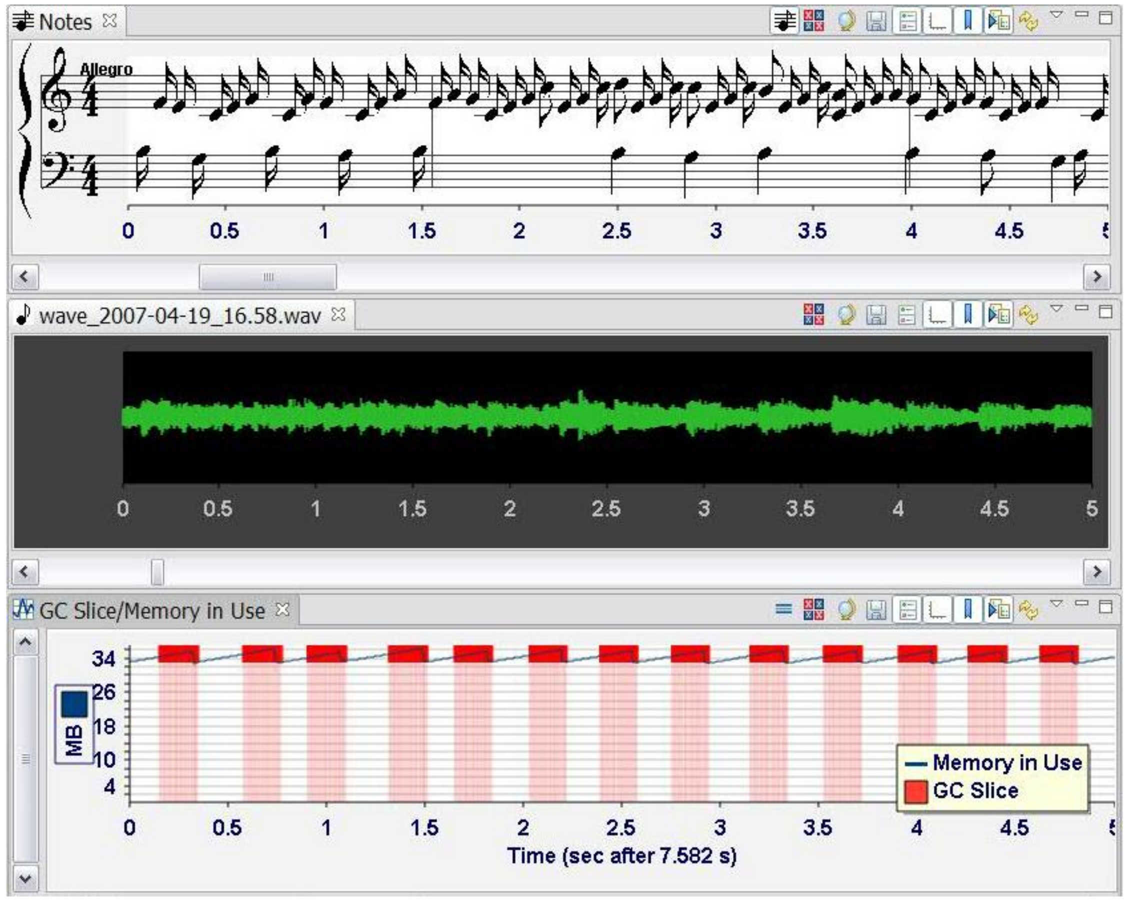


- Portable notion of time
  - Relative (producer/consumer ratio)
  - Absolute (external timing)
  - **Well defined under composition**

The Case for High-Productivity Languages in Embedded Systems: Advances in Real-Time Java

- MIDI synthesizer entirely in Java running on top of IBM WebSphere RT

Human ear can detect latencies of few milliseconds and jitter on even shorter time scale

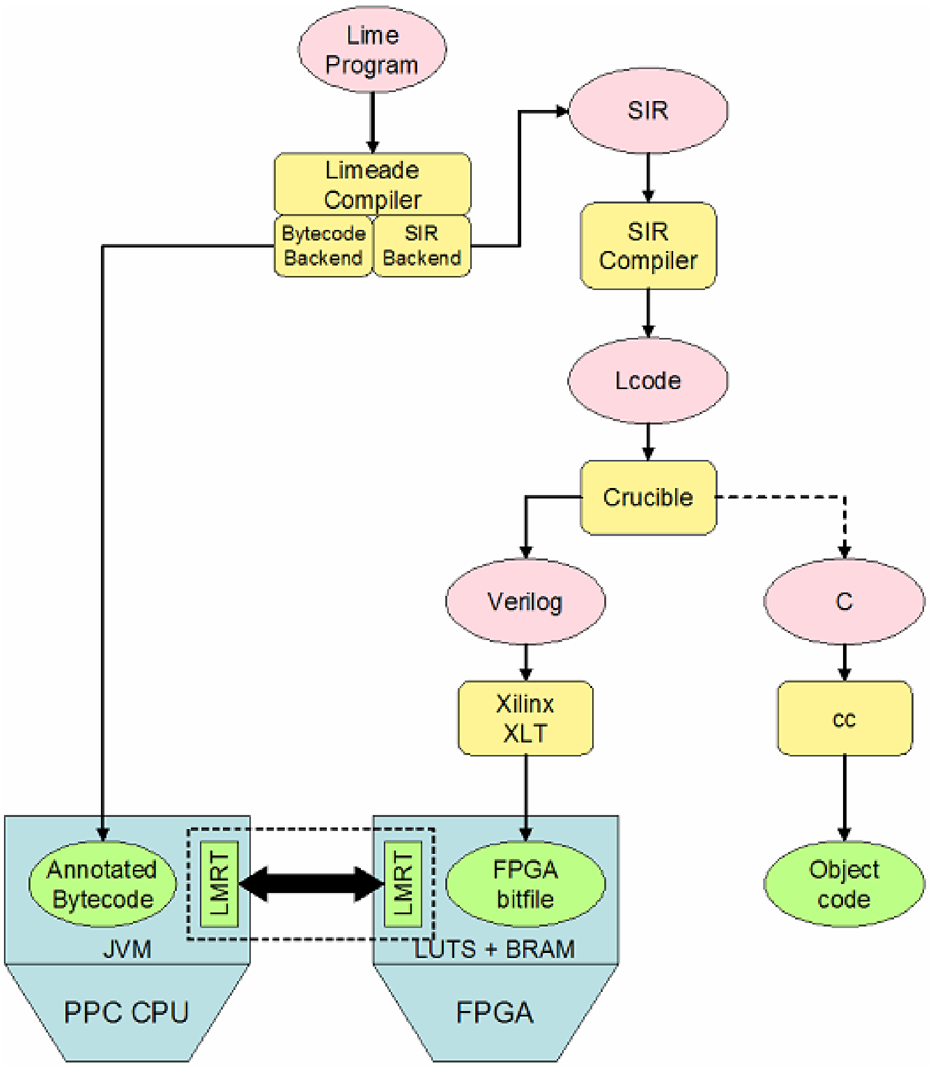


http://www.research.ibm.com/metronome

The Case for High-Productivity Languages in Embedded Systems: Advances in Real-Time Java

- Helicopter Flight Control System in Java running on GumStix

Current Lime Toolchain



- Functional end-to-end toolchain
  - Demonstrated proof of concepts on small kernels
- Lime compiler liquefies OO code
  - Efficiently support OO features in FPGA
  - Provision code to run in software or FPGA
- Compiler has several components
  - Frontend compiles to
    - Standard Java bytecode
    - Lime Spatial (Streaming) IR
  - Backend explores partitioning and scheduling plans
  - Generates Verilog and/or C
- Output can run in
  - Software (standard JVM)
  - Hardware (FPGA)

[To appear ECOOP 2008]

Preliminary Results

- Demonstrated the ability to support OO features in FPGA
  - Inheritance
  - Dynamic dispatch
  - “new”

- Demonstrated performance potential for small kernels with varying properties
  - Data, pipeline parallelism
  - Stateful and stateless computation
  - Different communication to computation ratios
  - Easy to verify output

The Liquid Metal Vision: "JIT the Hardware"

- Lime: high-level Java-based parallel programming model for programming software and hardware
  - Accessible to skilled Java programmers
  - Modular, composable, and malleable components
- Crucible: Lime-to-Hardware JIT compiler
  - Blur existing abstraction layers
  - Allow for application-specific customization throughout
- Lime VM: introspective and pluggable runtime system
  - Fluidly move computation between hardware and software
  - Instantiate on conventional CPUs, FPGA, heterogeneous systems, ...

Liquid Metal-heads

- David Bacon and Rodric Rabbah, IBM Research

- Summer 2007 Interns
  - Amir Hormati, University of Michigan
  - Shan Shan Huang, Georgia Tech