Lecture 16
Fine-Grained Reconfigurable Computing: FPGAs

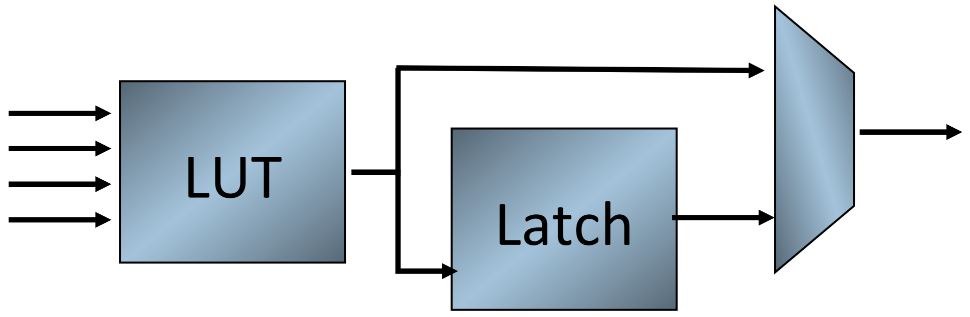
Joel Emer and Daniel Sanchez

6.888 Parallel and Heterogeneous Computer Architecture
Spring 2013

Massachusetts Institute of Technology

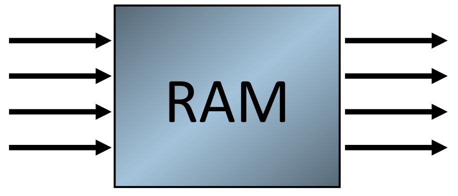
CSAIL

Field Programmable Gate Arrays (FPGA)

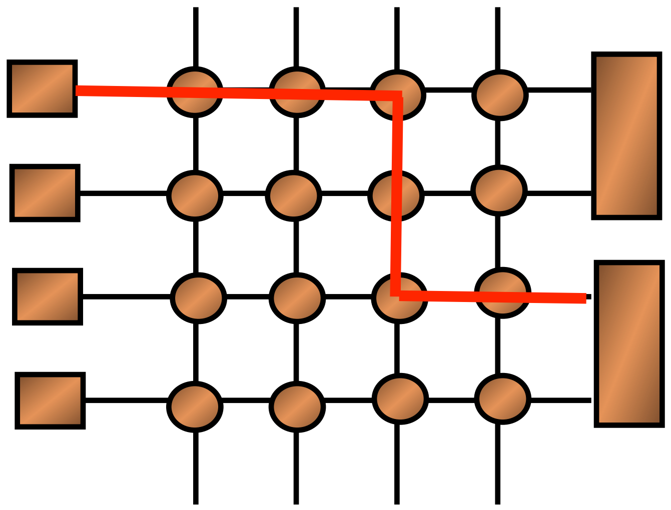


And	
00	0
01	0
10	0
11	1

Or	
00	0
01	0
10	1
11	1



.....



Evolution of FPGA applications

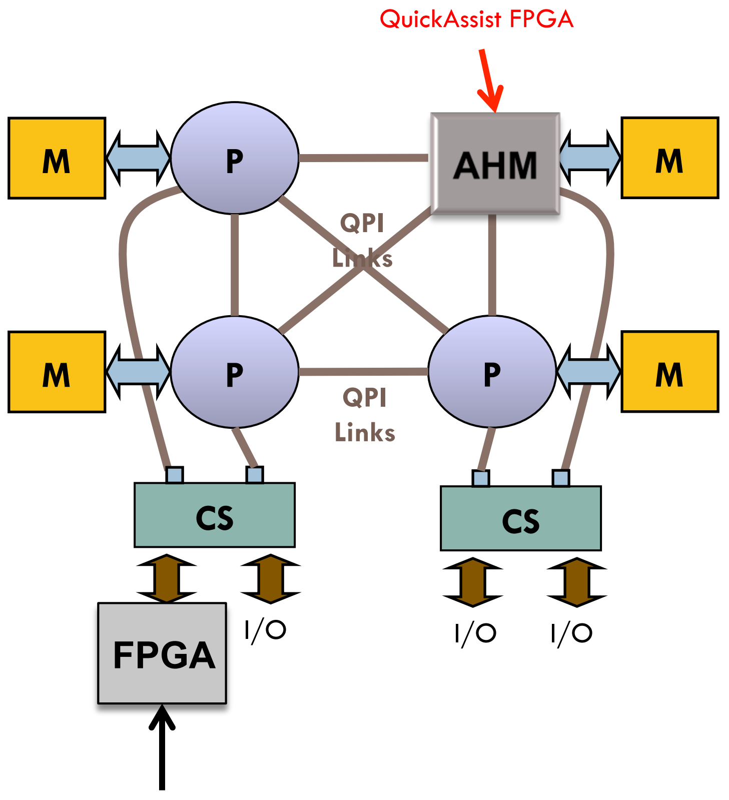
- Logic Replacement
  - Low design cost and effort
  - Low volume applications
  - Often replaced with ASIC as volume increases
- Algorithmic computation
  - Offloads a general purpose processor
  - Used for multiple algorithms
  - ASIC replacement not expected

Benefits of FPGA computation

- Custom operations/data types – custom operations/data types
- Flexible flow control - control flow based on arbitrary state machines
- Local state access - local state elements allows parallel state access
- Fine grain parallelism - replicated logic permits easy parallelism
- Custom communication - explicit direct inter-module communication
- Reduced memory references – more direct reuse of data
- Better power efficiency – more activity directly applied to computation
- Better area efficiency – more area directly applied to computation

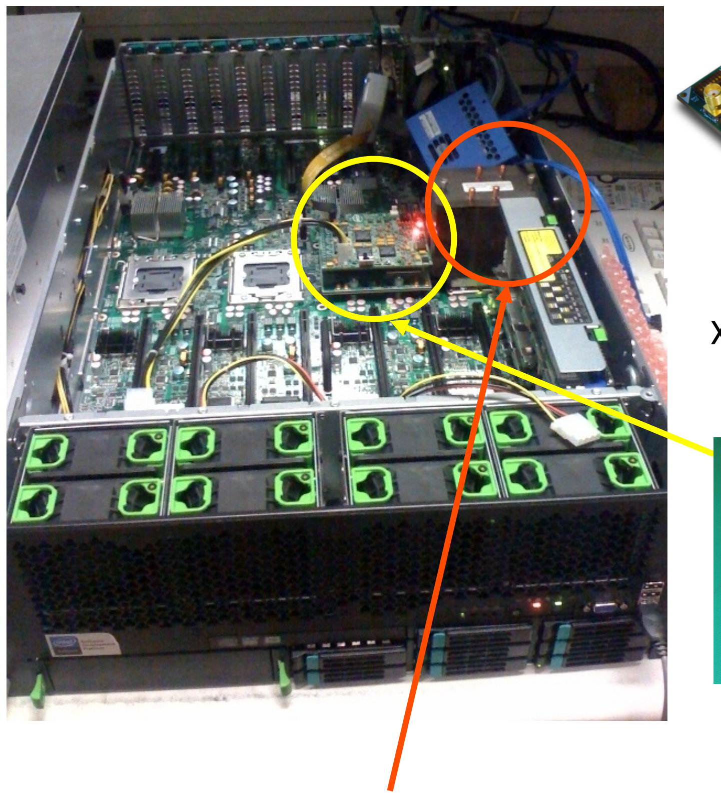
QPI-attached FPGA platform

Four Socket QuickAssist Platform Topology



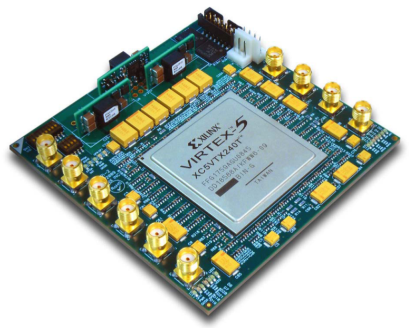
PCIe attached FPGA

Intel QuickAssist QPI-based FPGA Accelerator Platform (QAP)

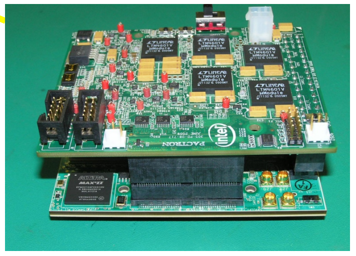


Intel® Xeon processor 7000 series

Accelerator Hardware Module (AHM)



Xilinx Virtex 6 Module



Altera Stratix IV Module

Reed Solomon Results

WiMAX requirement is to support a throughput of 134Mbps

	Xilinx	Catapult-C	Bluespec
Equivalent Gate Count	297,409	596,730	267,741
Frequency (MHz)	145.3	91.2	108.5
Steady State (Cycles/ Block)	660	2073	276
Data rate (Mbps)	392.8	89.7	701.3

Lower is better

Higher is better

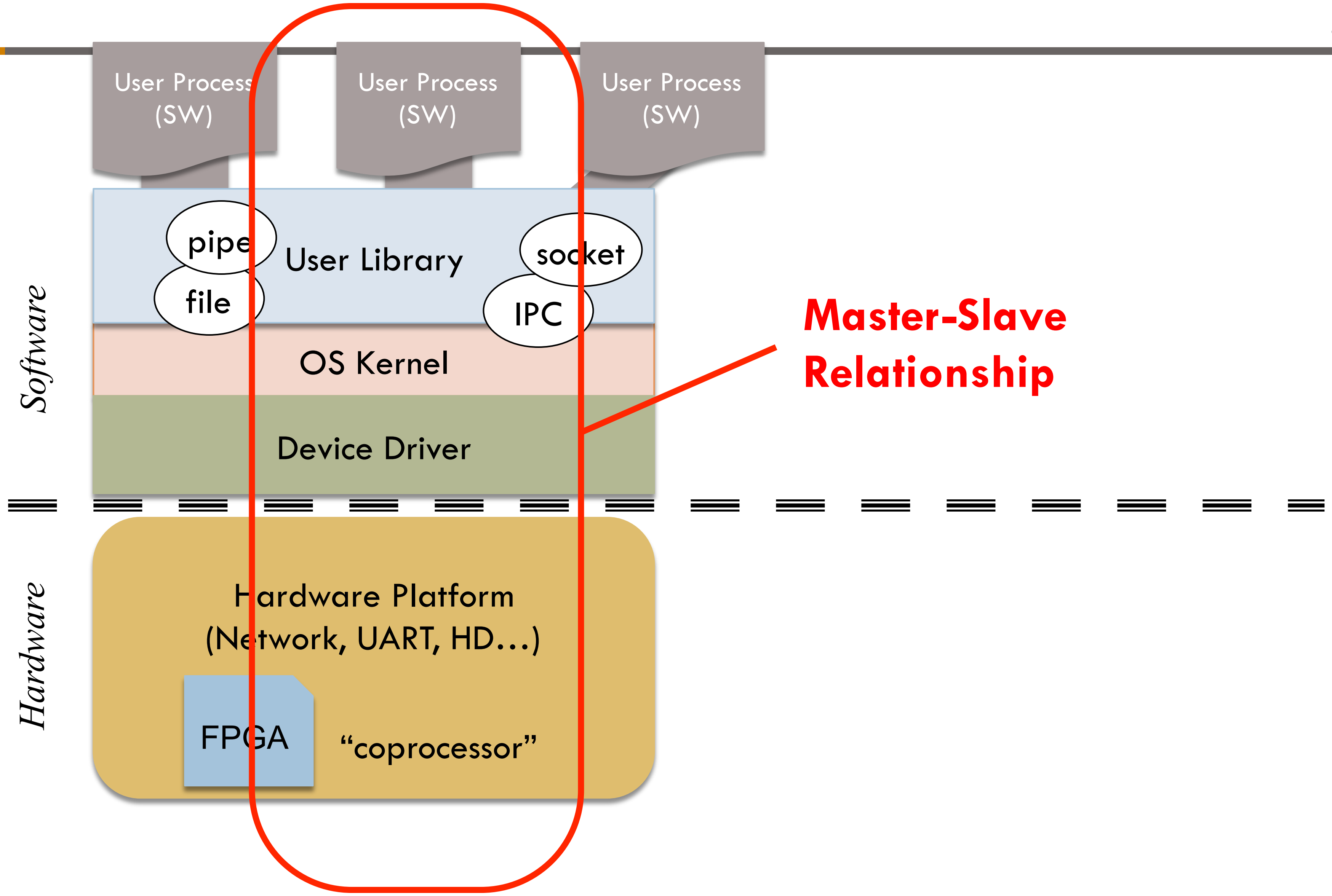
Source: MIT, Abhinav Agarwal, Alfred Ng – CSG

BORPH

- Berkeley Operating system for ReProgrammable Hardware
- OS for reconfigurable computers
  - Treats reconfigurable hardware as computational resources
- UNIX interface to HW designs
  - Familiar to both software and hardware engineers
  - Design language independent
- Goal:

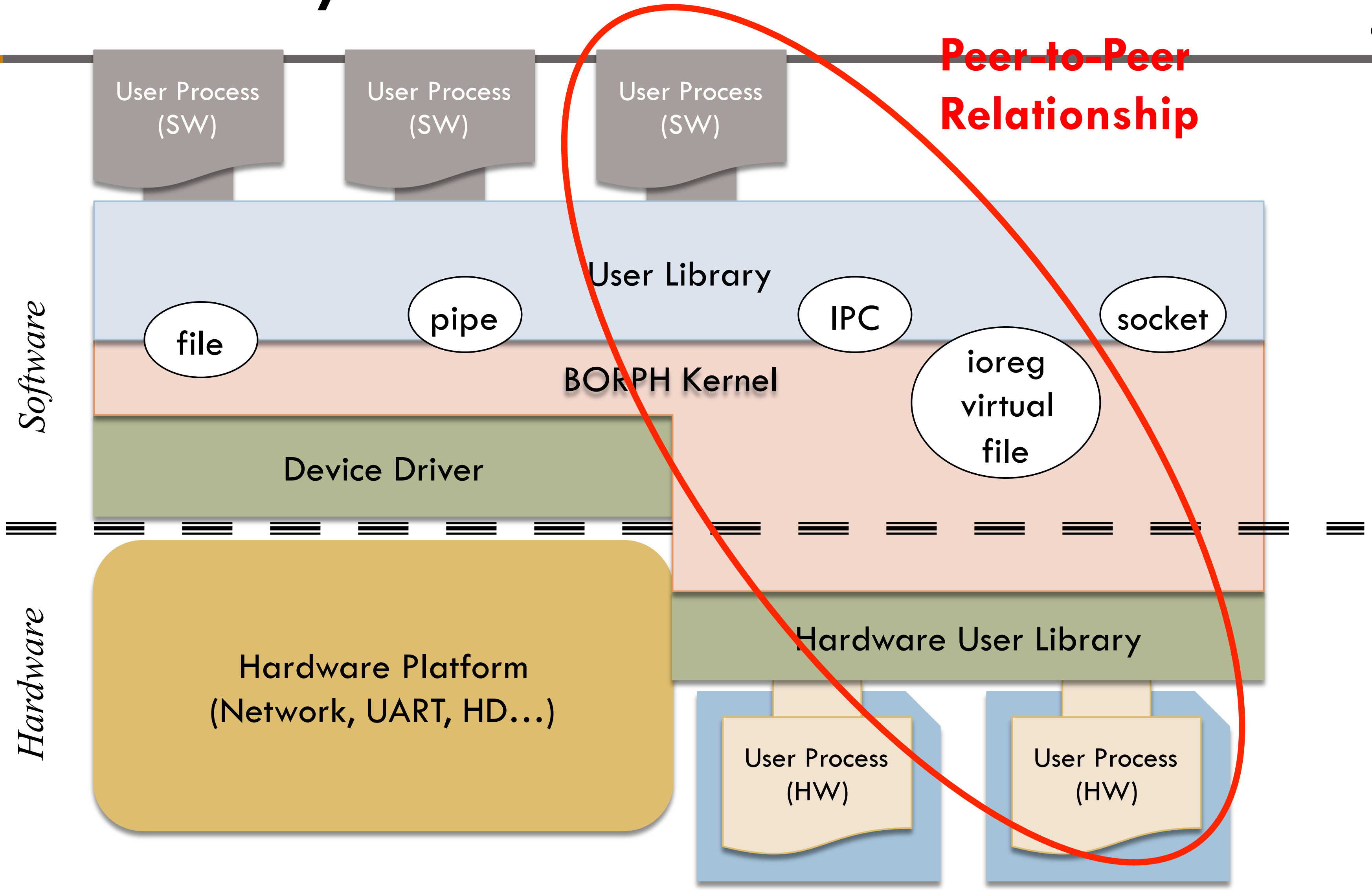
Make FPGA-based reconfigurable computers ***easy to use***

Conventional View of FPGA Systems



6.888 Spring 2013 – Sanchez and Emer – L16

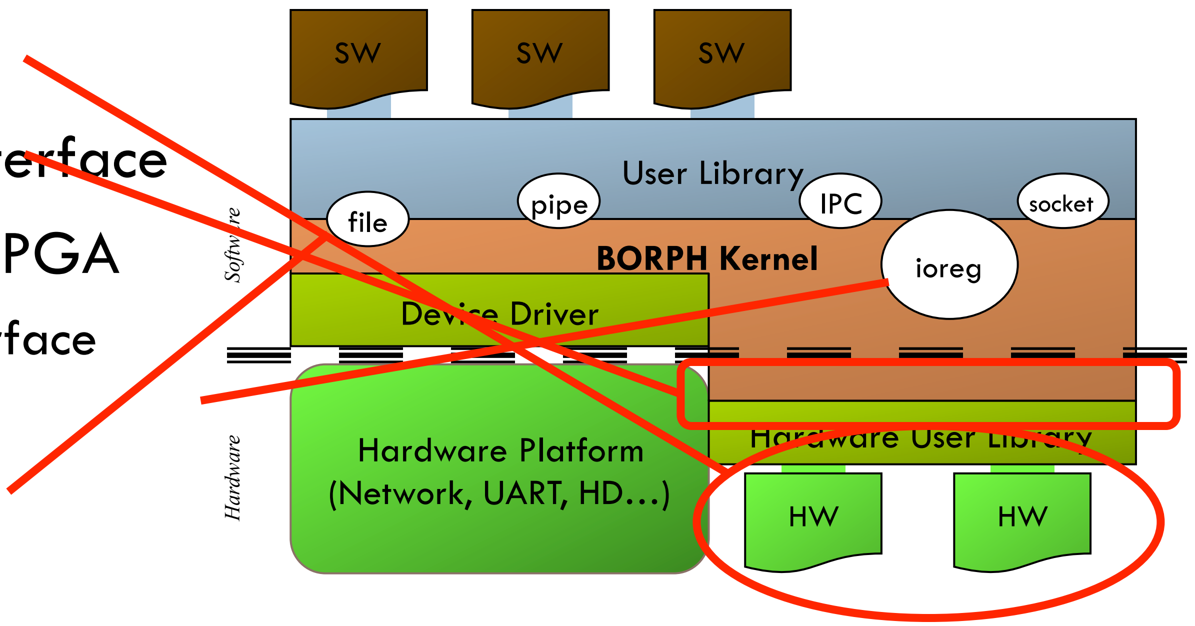
BORPH Layers



6.888 Spring 2013 – Sanchez and Emer – L16

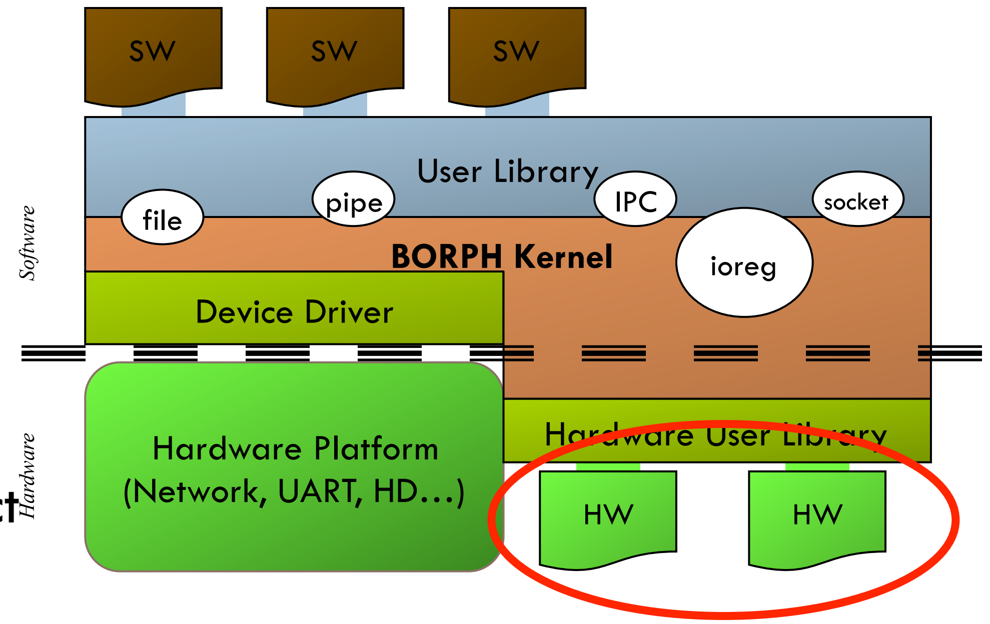
Overview of BORPH Concepts

- Hardware process
- Hardware syscall interface
- Interacting with an FPGA
  - ioreg virtual file interface
  - Hardware file I/O



Hardware Process (1)

- An executing instance of a hardware design
  - SW: An executing instance of a program
- Normal UNIX process
  - Has pid, check status with **`ps`**, **`kill`**, etc
- Unit of management
- Created when a <u>B</u>ORPH <u>O</u>bject <u>F</u>ile (BOF) file is **`exec`**-ed
  - Kernel selects and configure hardware region automatically

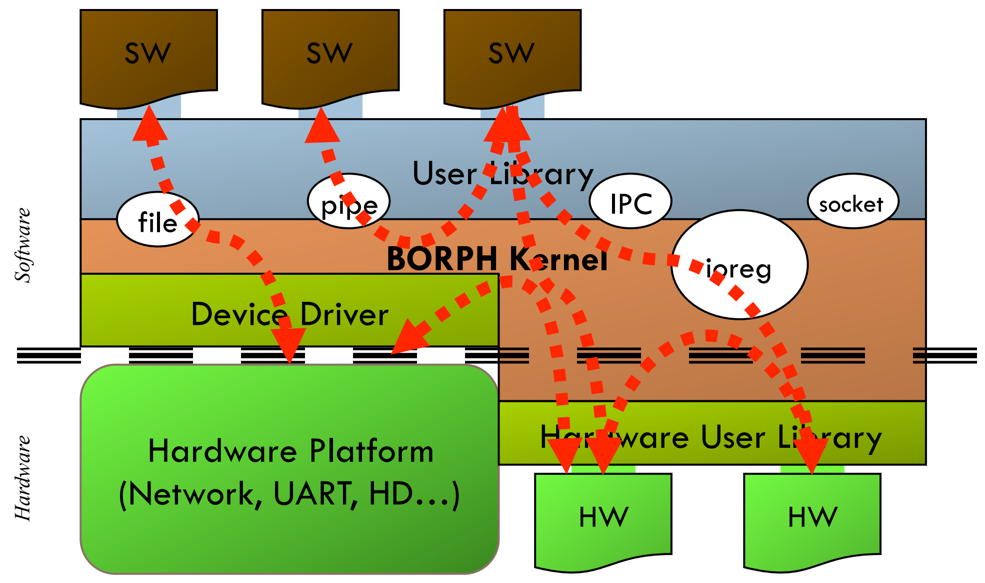


Benefits of UNIX Process Model

- Very easy for user to reason about
- Enable FPGA designs to become active component of the system
  - e.g. an FIR filter:
  - Conventional: a ***passive*** entity where software sends/receives data
  - BORPH: an ***active*** entity that pulls/pushes data as needed
- Enable multiple instances of the same FPGA design running in the system
  - No more fixed accelerator concept
  - Works well in true reconfigurable computing systems

HW Processes I/O

- I/O managed by kernel
  - Similar to SW
- Hide details from users
  - e.g. HW-SW, HW-HW UNIX file pipe
- Standard UNIX I/O mechanism
  - File I/O, pipe, signal
- HW specific service
  - ioreg virtual file system



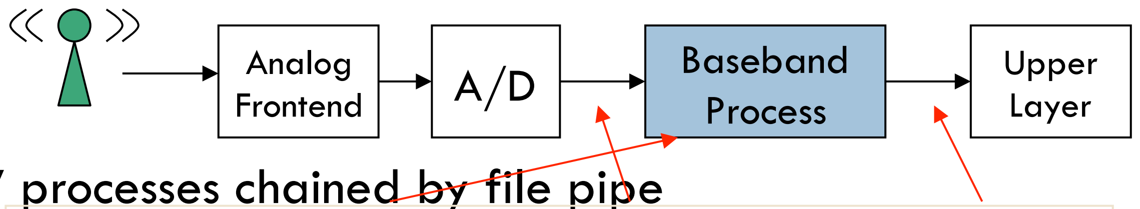
Don’t ask “How do I … in HW”.
Think: “What if it were SW?”

ioreg Virtual File System

- Maps *user defined hardware constructs* as *virtual files* under the process's **`/proc/<pid>/hw/ioreg/`** directory
  - Single word register
  - Memory: On-chip + Off-chip
  - FIFO
- Example:
  - **`/proc/123/hw/ioreg/COUNTERVAL`**
- ioreg information embedded in the executing BOF file
- **`read`** and **`write`** system calls translated to message packet by the ***kernel***
  - Any UNIX program can communicate with hardware processes
    - Shell: **`echo 1 > /proc/123/hw/ioreg/enable`**
    - C: **`MEM_FILE =`**
    - **`    fopen("/proc/123/hw/ioreg/MyMemory", "r");`**
    - **`  fread(swbuf, 1, MEM_SIZE, MEM_FILE);`**
    - Python, Java, etc…

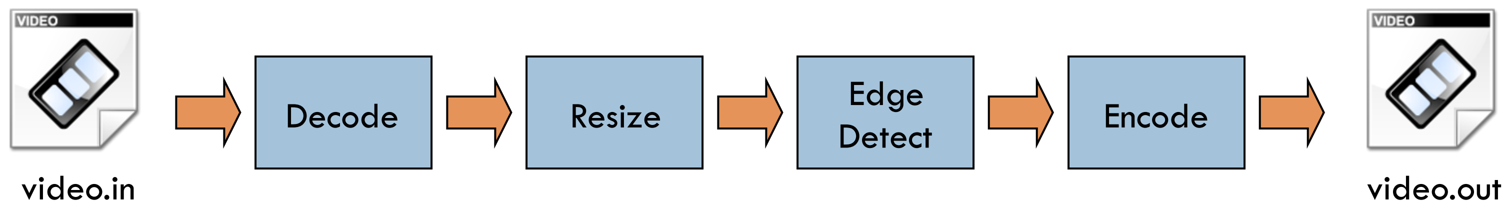
Hardware File I/O

- Access to the general file system from hardware processes
- Debug by printing
  - printf
- Read test vectors, record output



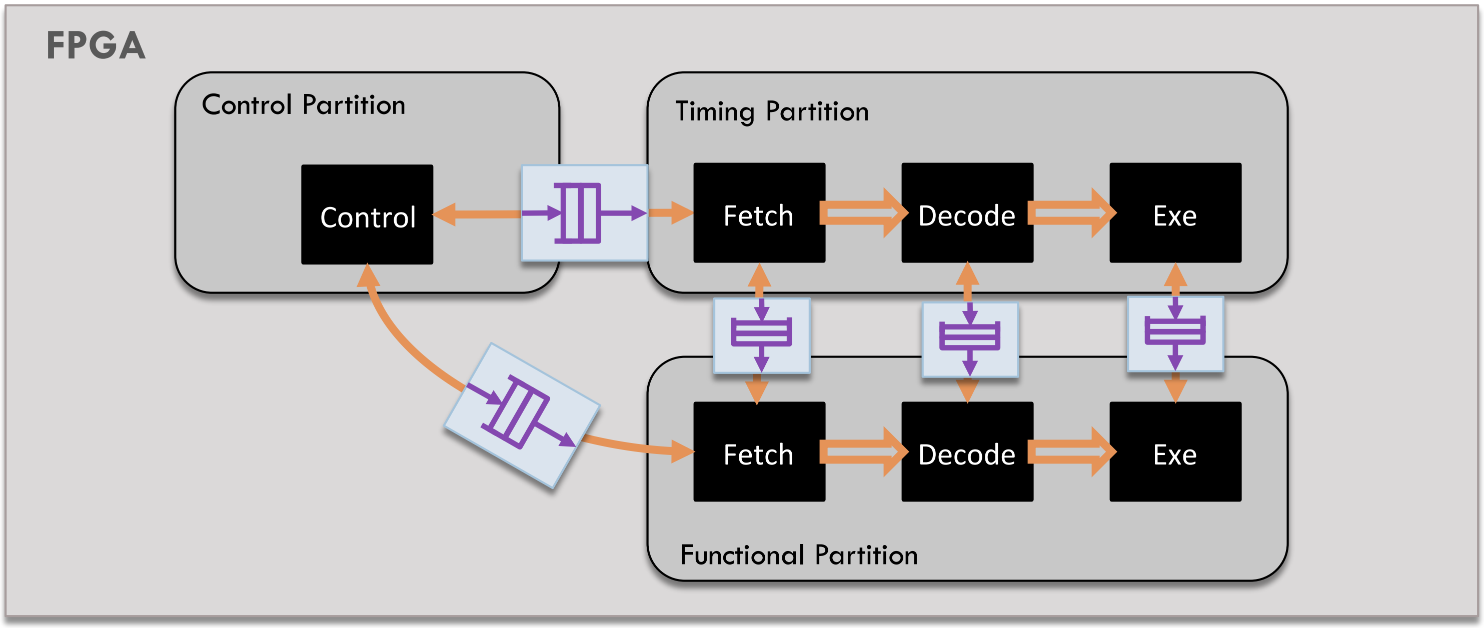
- SW/HW processes chained by file pipe

```
bash$ receiver.bof < file.in > file.out
```



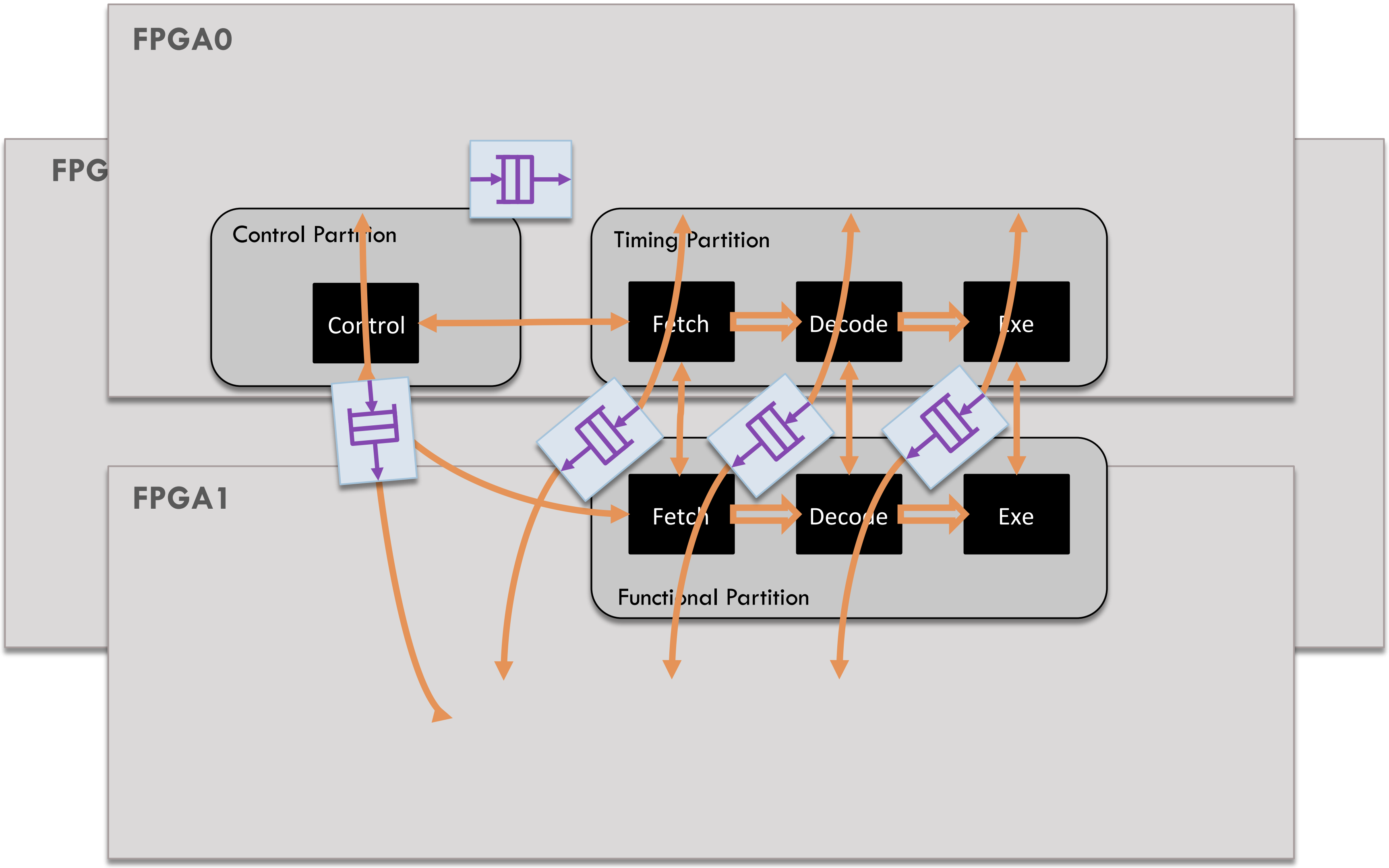
```
bash$ decode video.in | resize | edgdet.bof | encode > video.out
```

Latency-Insensitive Design: A Higher Semantic



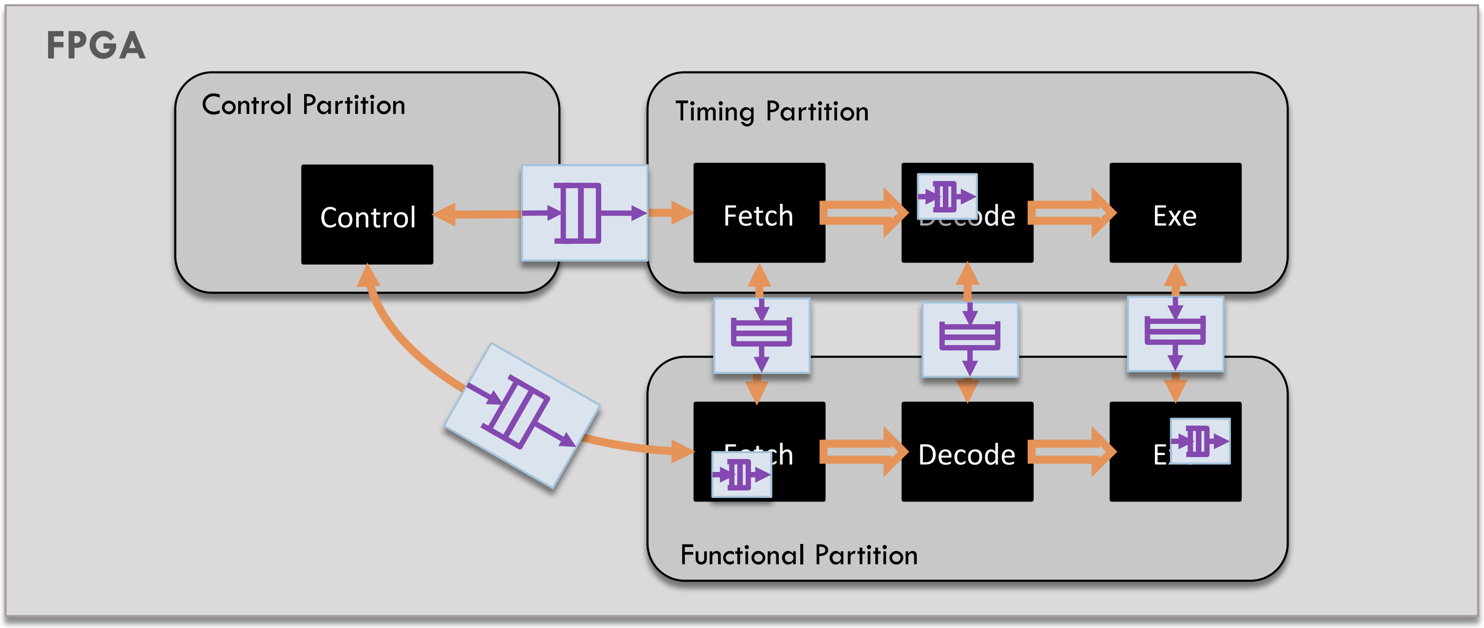
- Inter-module communication by latency insensitive channels
  - Changing the timing behavior of a module does not affect functional correctness of the program
- Many HW designs use this methodology
  - Improved modularity
  - Simplified design-space exploration
- Implemented with guarded FIFOs in current RTLs

Latency-Insensitive Design: A Higher Semantic



Behavior of LI channels does not affect functional correctness.

Latency-Insensitive Design: A Higher Semantic

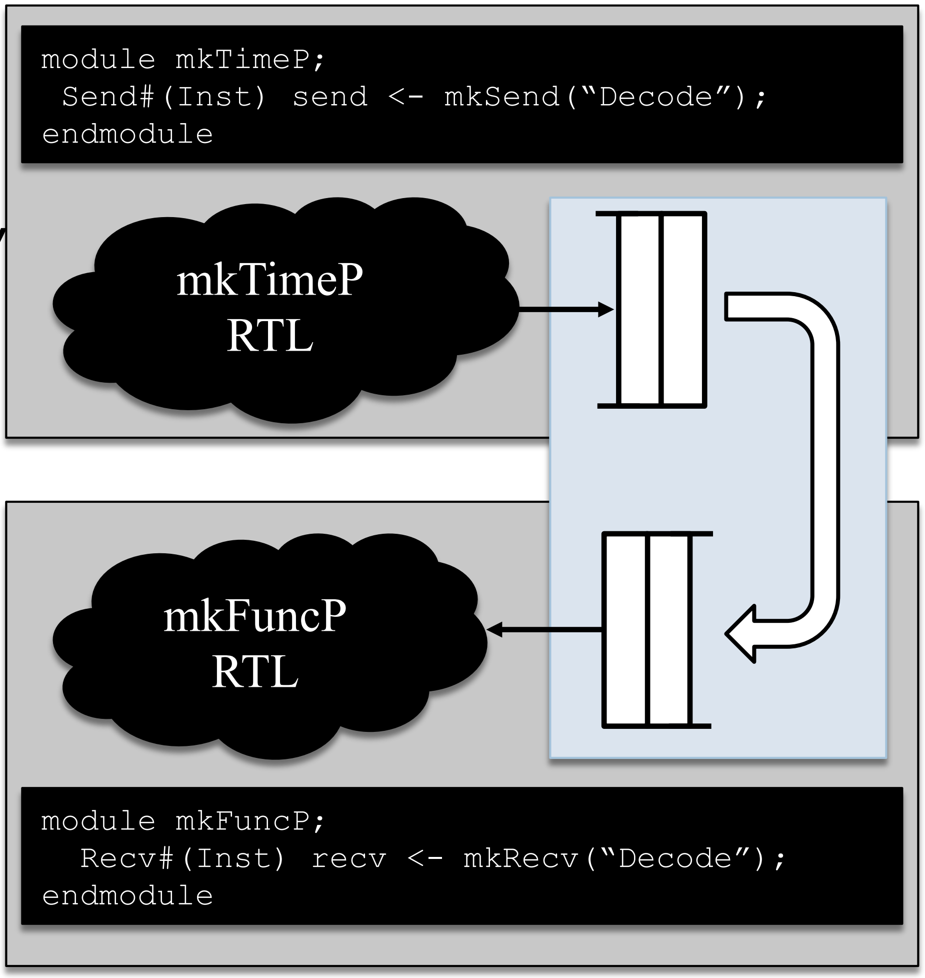


- There are many FIFOs in the design
  - It may not be safe to modify some of them
- Compilers see only wires and registers
  - Reasoning about cycle accuracy is difficult

But the programmer knows about the LI property…

A Syntax for LI Design

- Programmer needs to differentiate LI channels from normal FIFOs
- Latency-Insensitive Send/Recv endpoints
  - Implementation chosen by compiler
  - FIFO order
  - Guaranteed delivery
- Explicit programmer contract
  - Unspecified buffering & unspecified latency
  - Programmer responsible for correct annotation

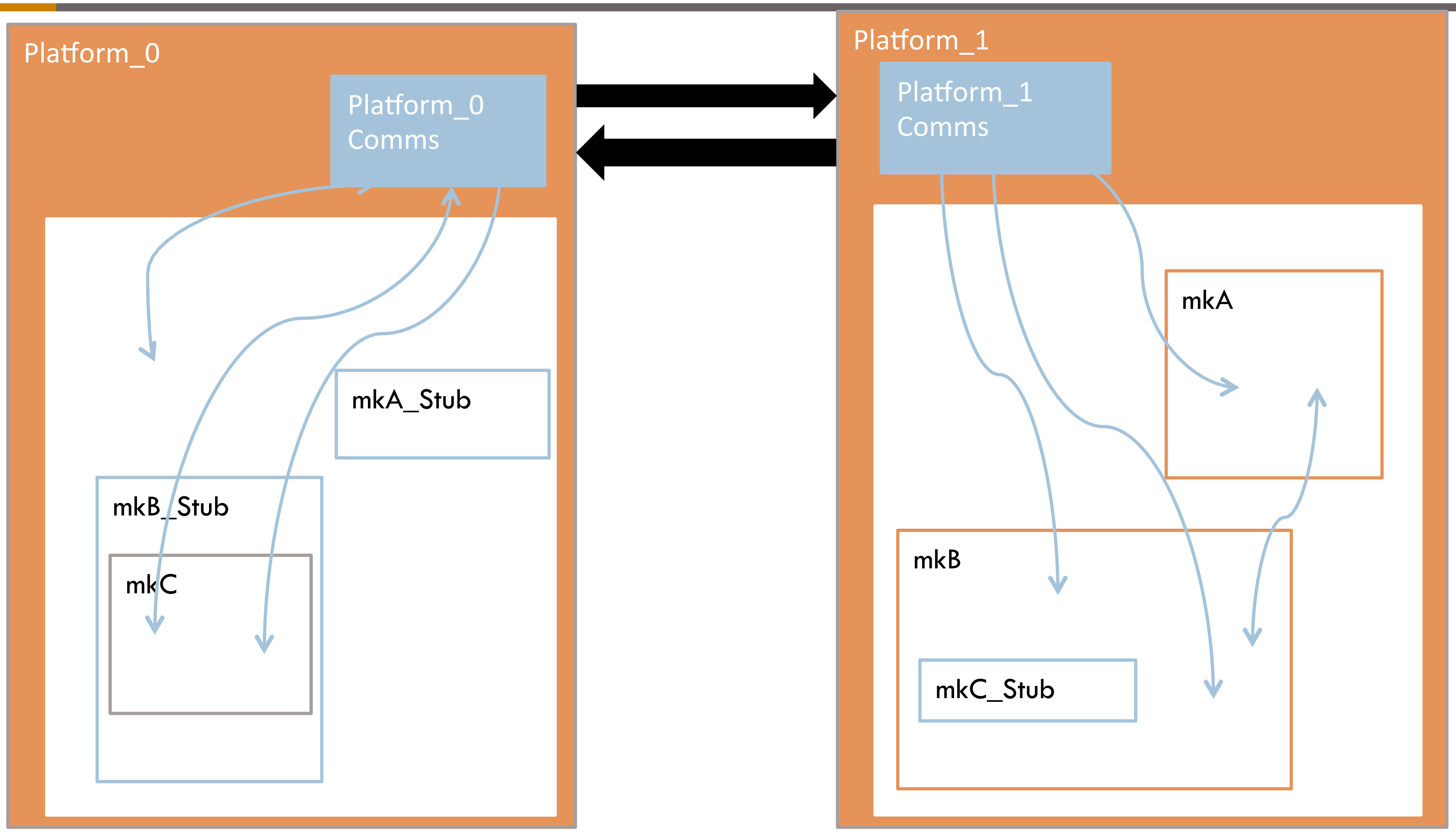


```
module mkTimeP;
 Send#(Inst) send <- mkSend("Decode");
endmodule
```

```
module mkFuncP;
   Recv#(Inst) recv <- mkRecv("Decode");
endmodule
```

Easy to use – often a textual substitution!

Connected User Application



Key: User Library Generated