

# Framework for a Context-Switching Run-Time Reconfigurable System

David I. Lehn

dlehn@vt.edu

May 3, 2002

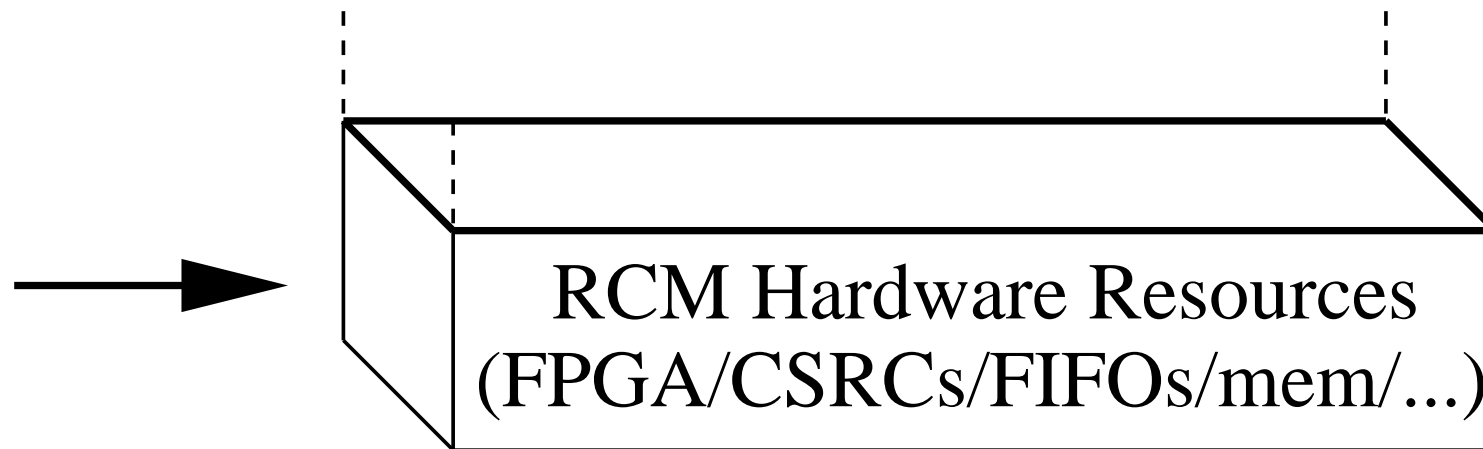
# The Plan

- Goals
- Hardware
- Middleware
- Software
- Applications
- Results & Conclusions
- Q & A

# Goals

- Have complex research hardware
- Want to support various applications
- Need:
  - Completeness
  - Testability
  - Usability
  - Performance

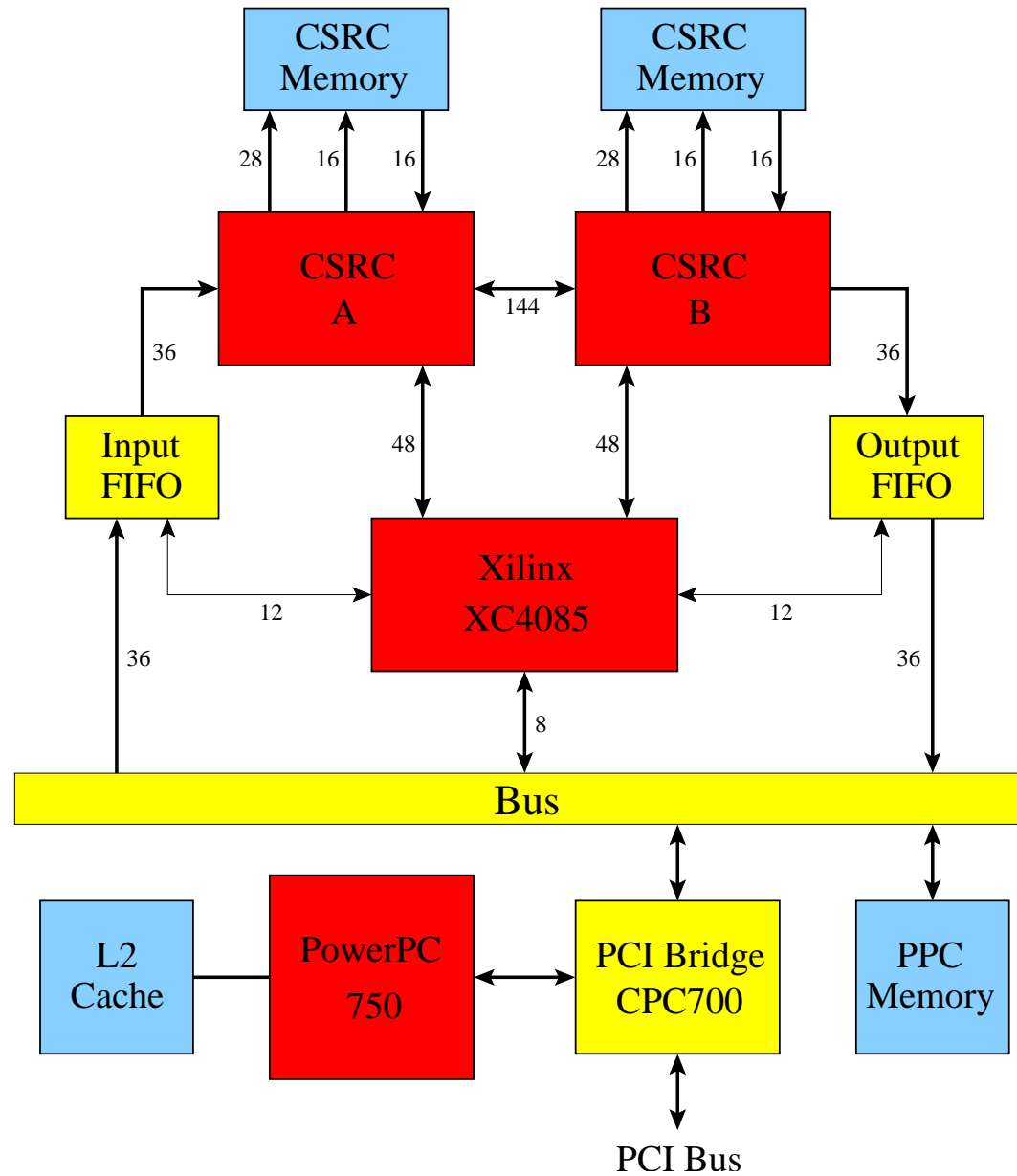
# Hardware Layer



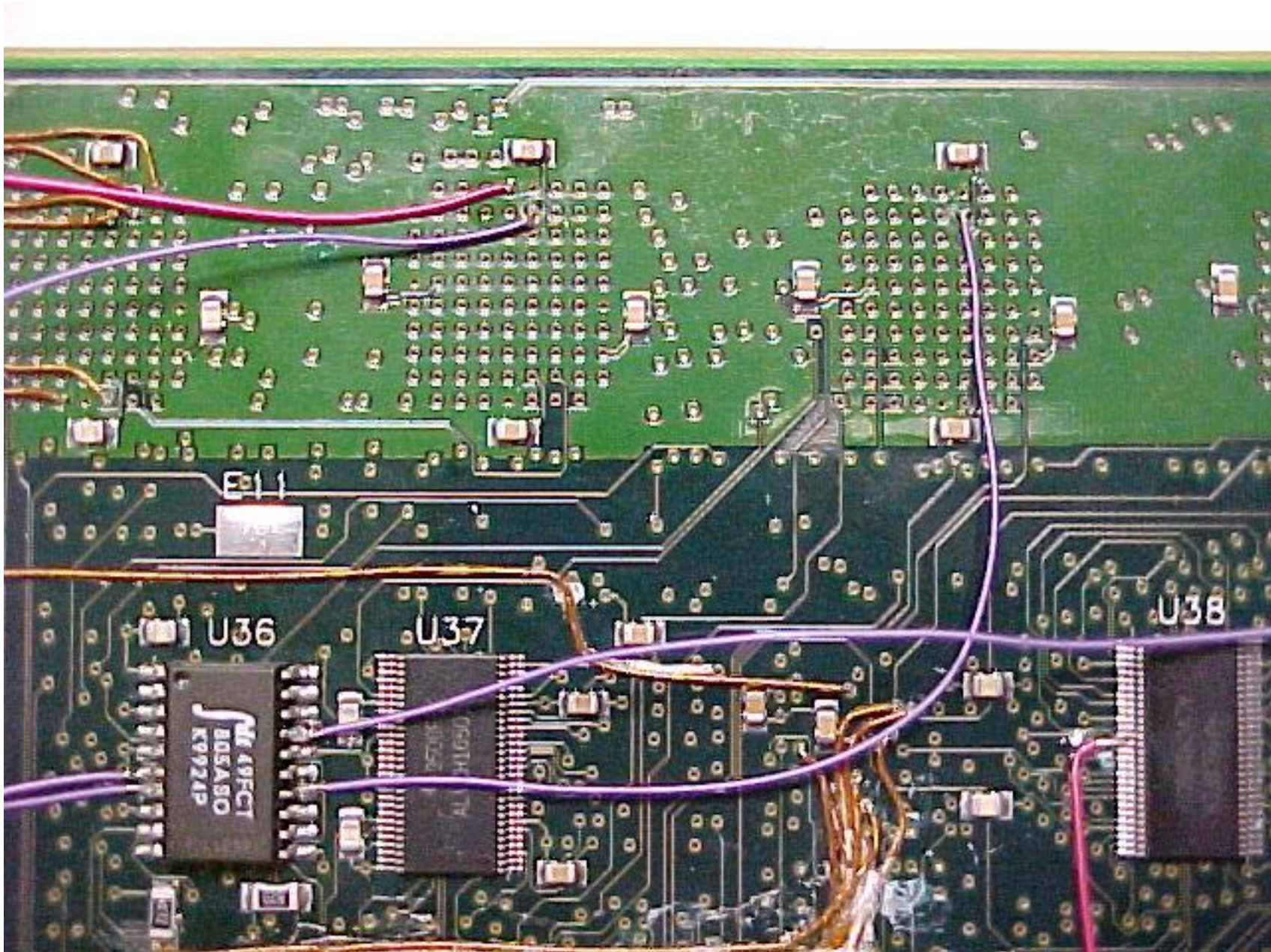
# CSRC

- Context Switching Reconfigurable Computer
- Structure similar to a traditional FPGA
- Multiple configurations (contexts)
- One active executing context
- Shared state between contexts
- Contexts switch *very* fast
- Configurable in the background

# RCM Platform



# Kiran fixing the CSRC memory



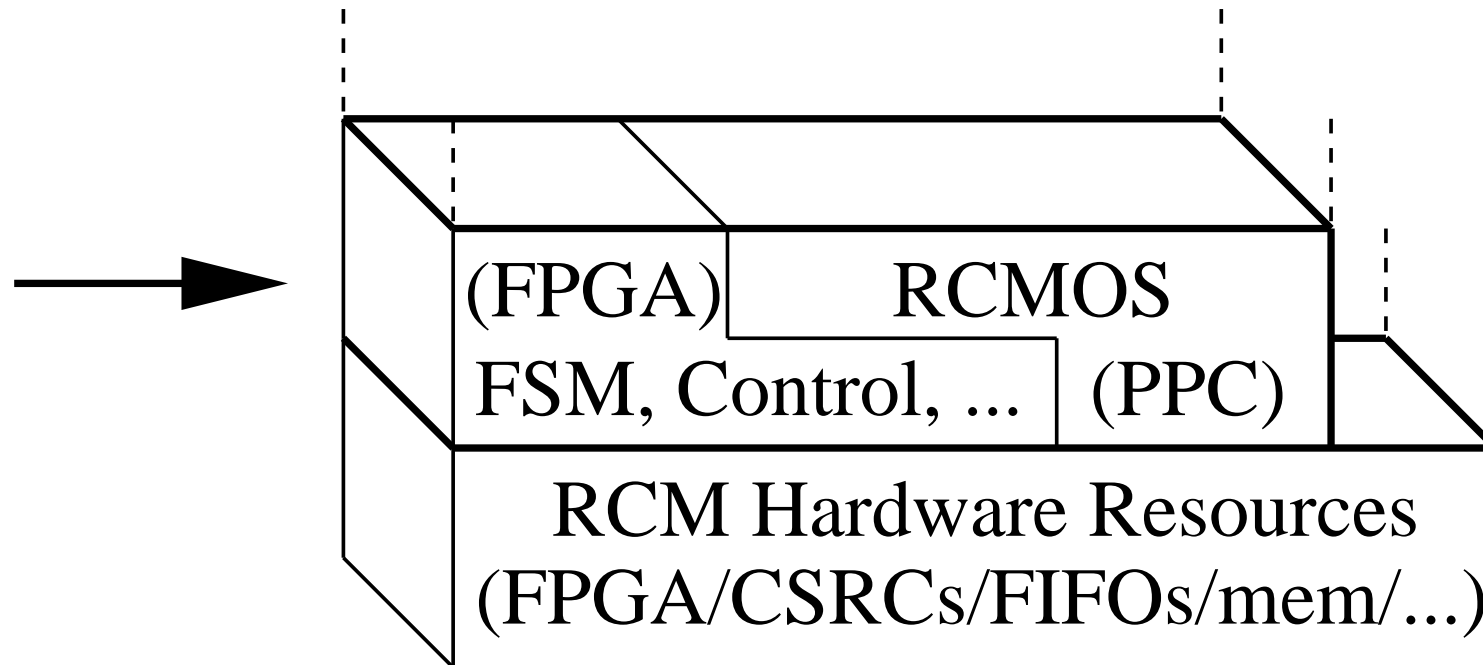
# How to use the hardware?

- A few problems...
  - Hardware “features”
  - Place & route tools
- Hardware + Tool problems = not good
- Had its benefits...

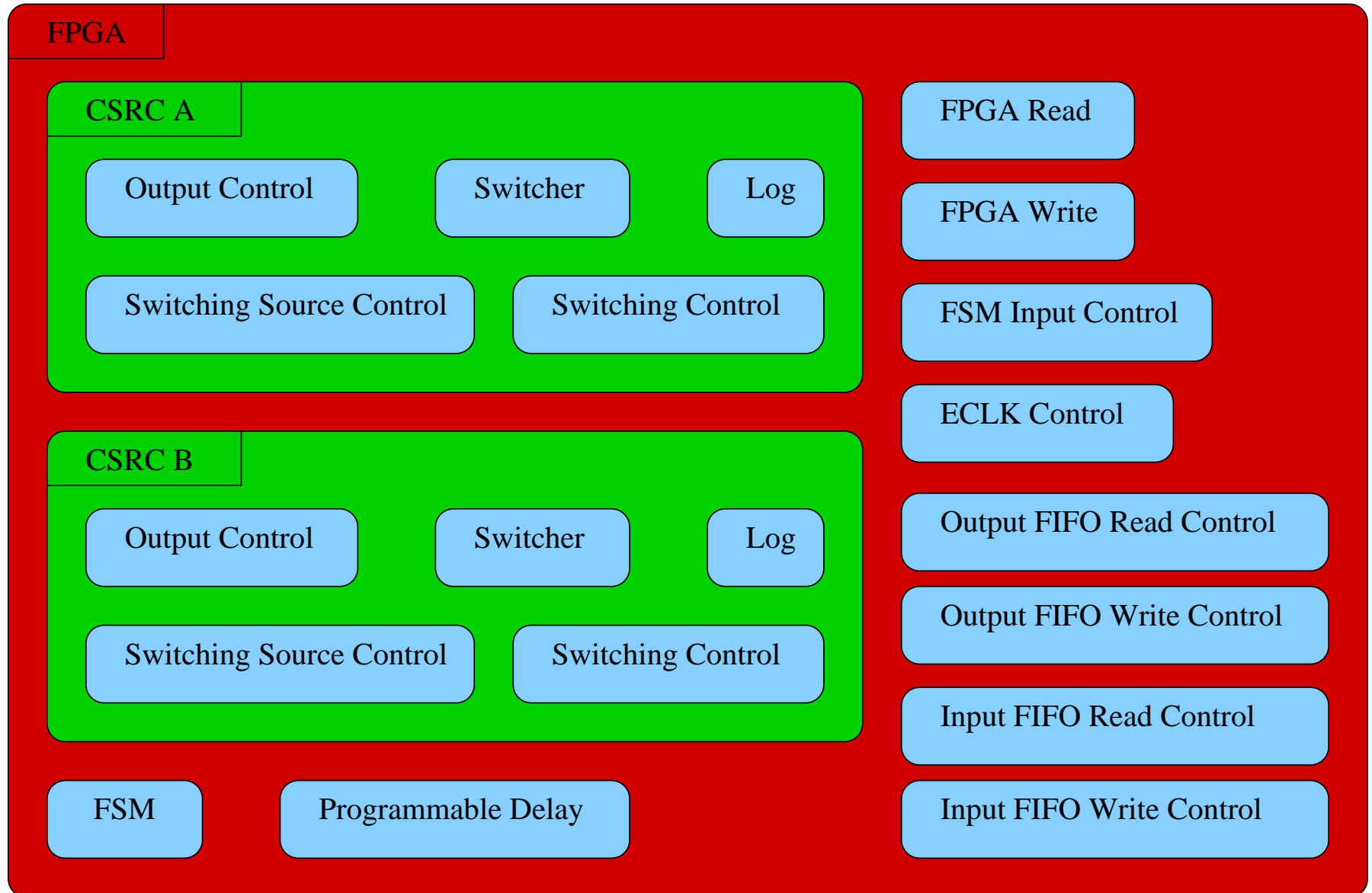
# Testing

- *Lots* of hardware/software tests
- Writing tests good for API improvement
- *Proof* the hardware works
- Easy to check differences between prototype boards
- All done through scripting language: Python

# Middleware Layer

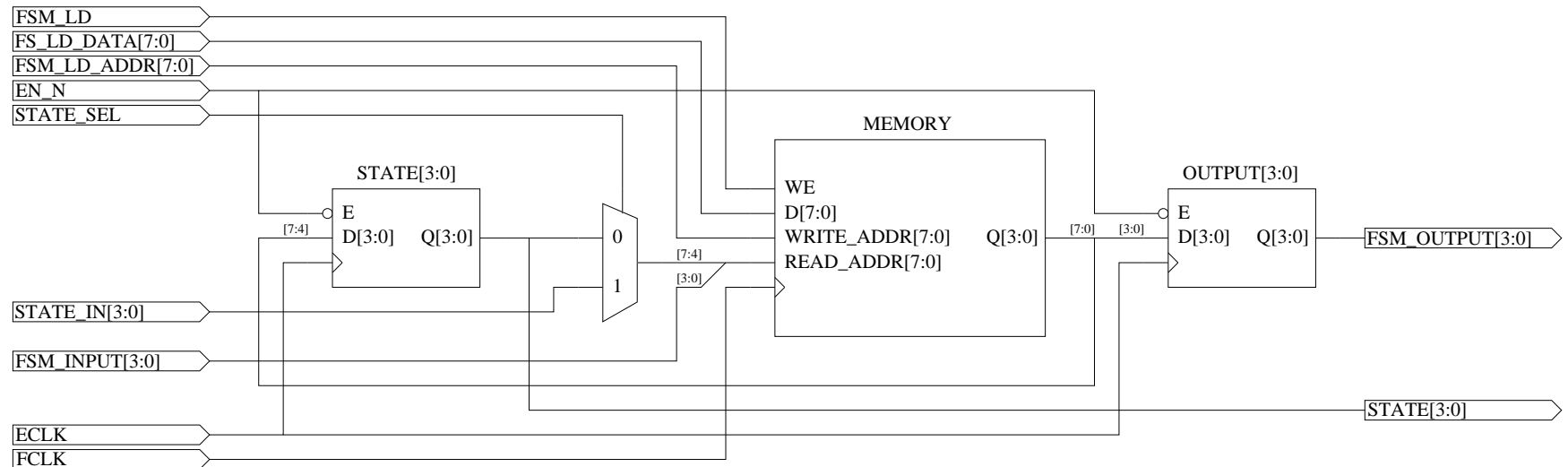


# FPGA Configuration

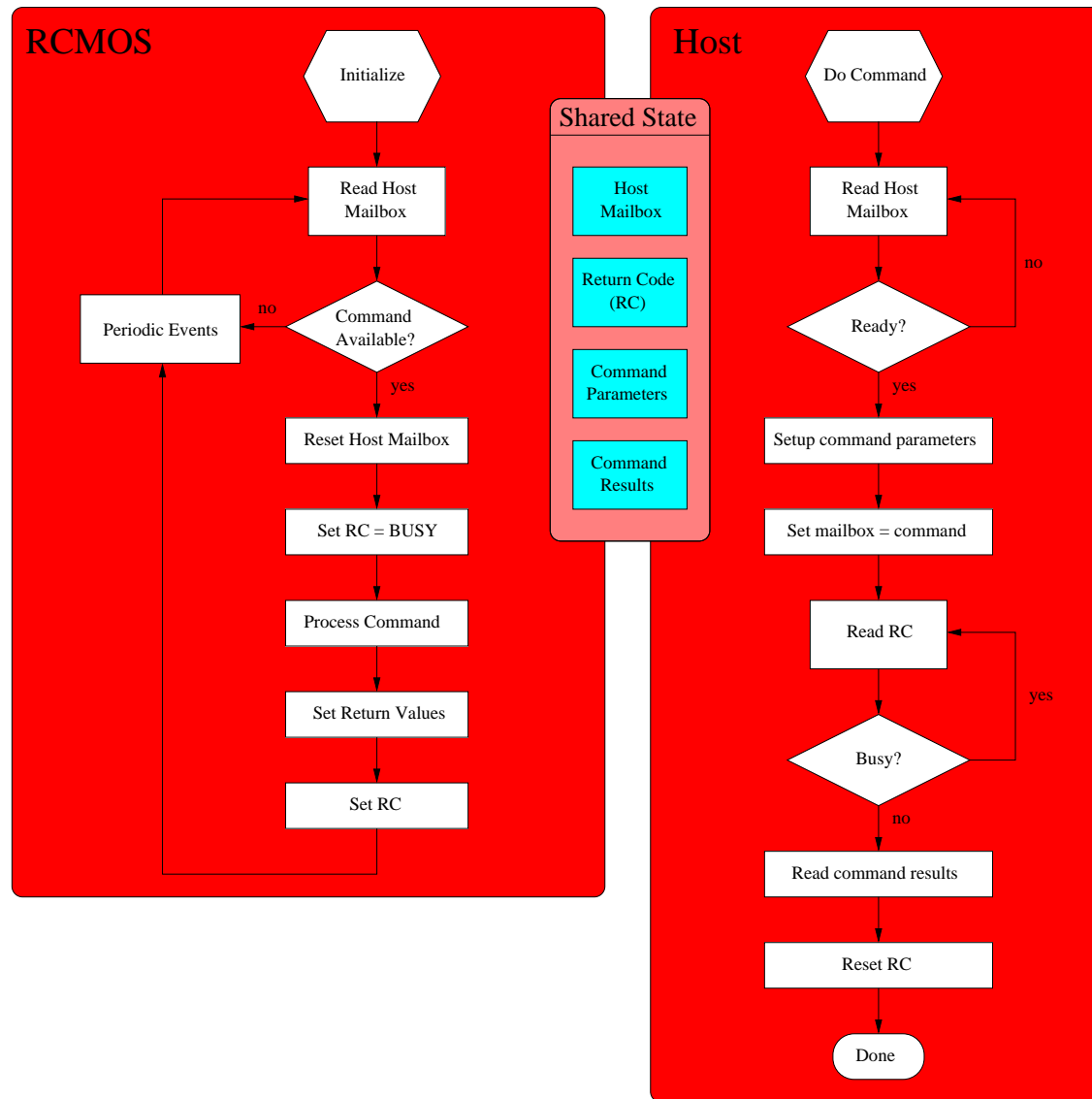


# FSM

- Memory based Finite State Machine
- Uses Xilinx primitives
- Convenience software to simplify programming
- Flexible input and output signal routing

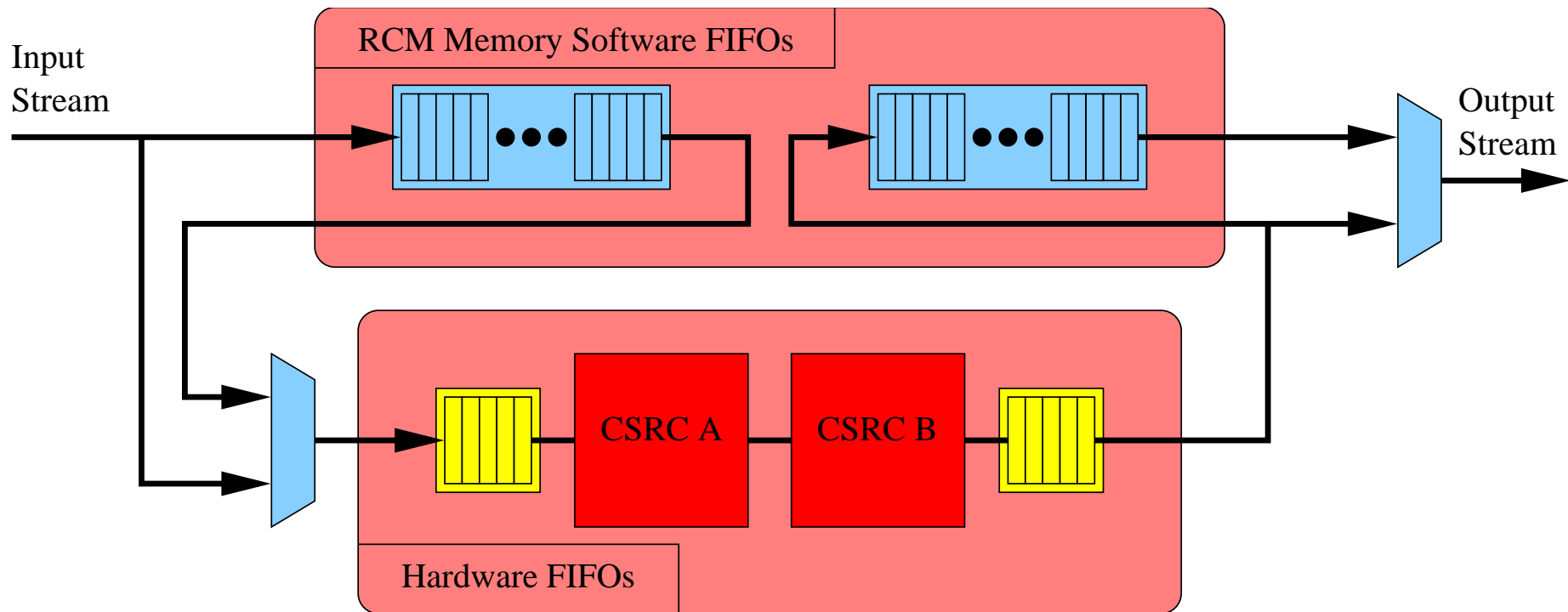


# RCMOS

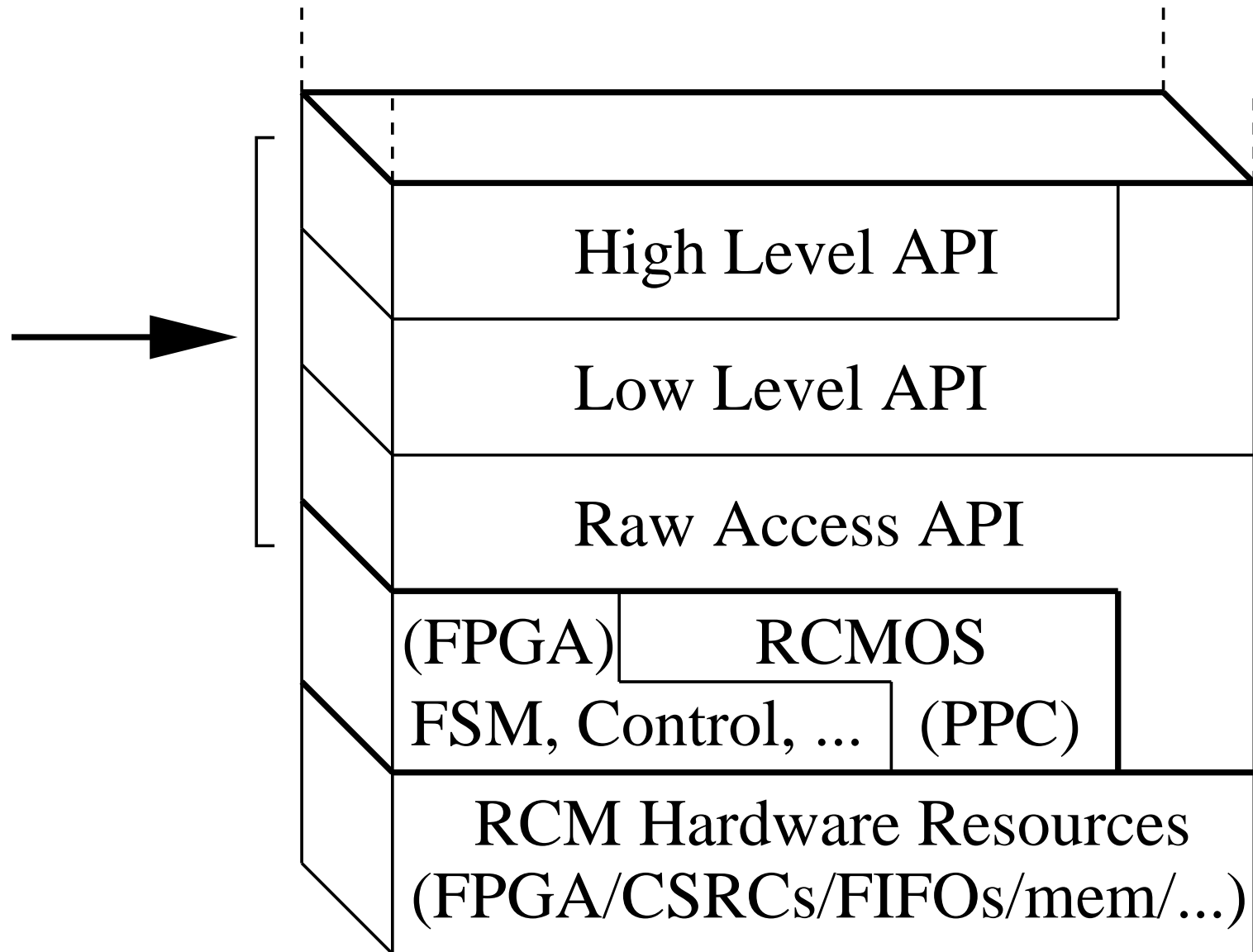


# Software FIFOs

- Hardware FIFOs are 16k by 36-bit
- Software FIFOs can be *much* larger or disabled
- Latency and control issues

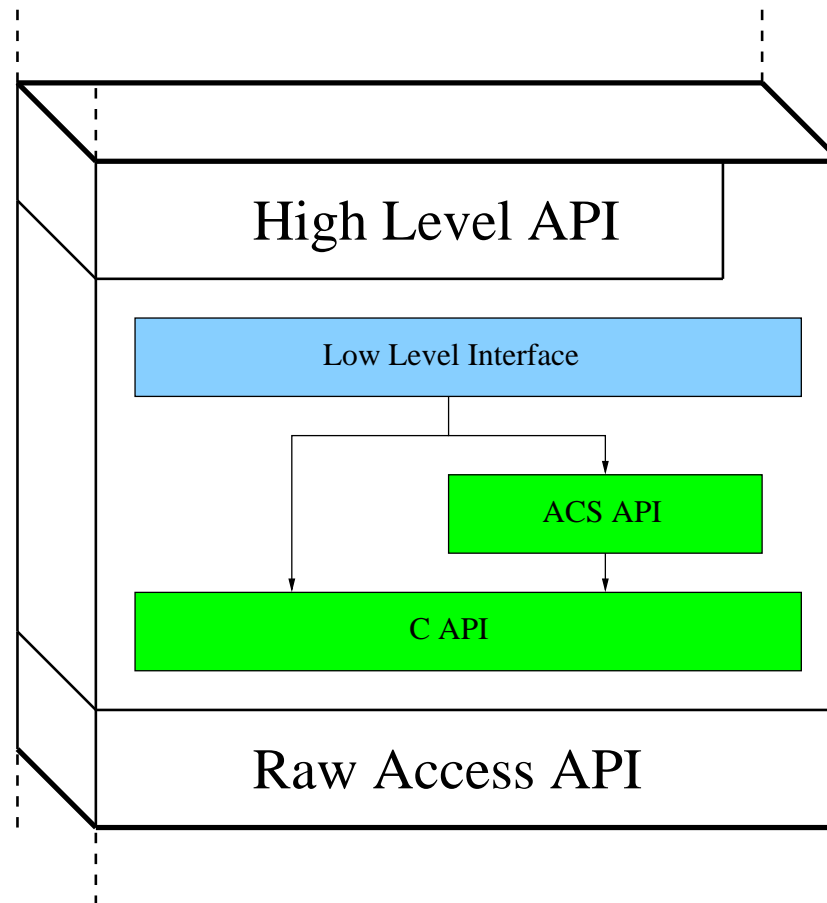


# Software Layer



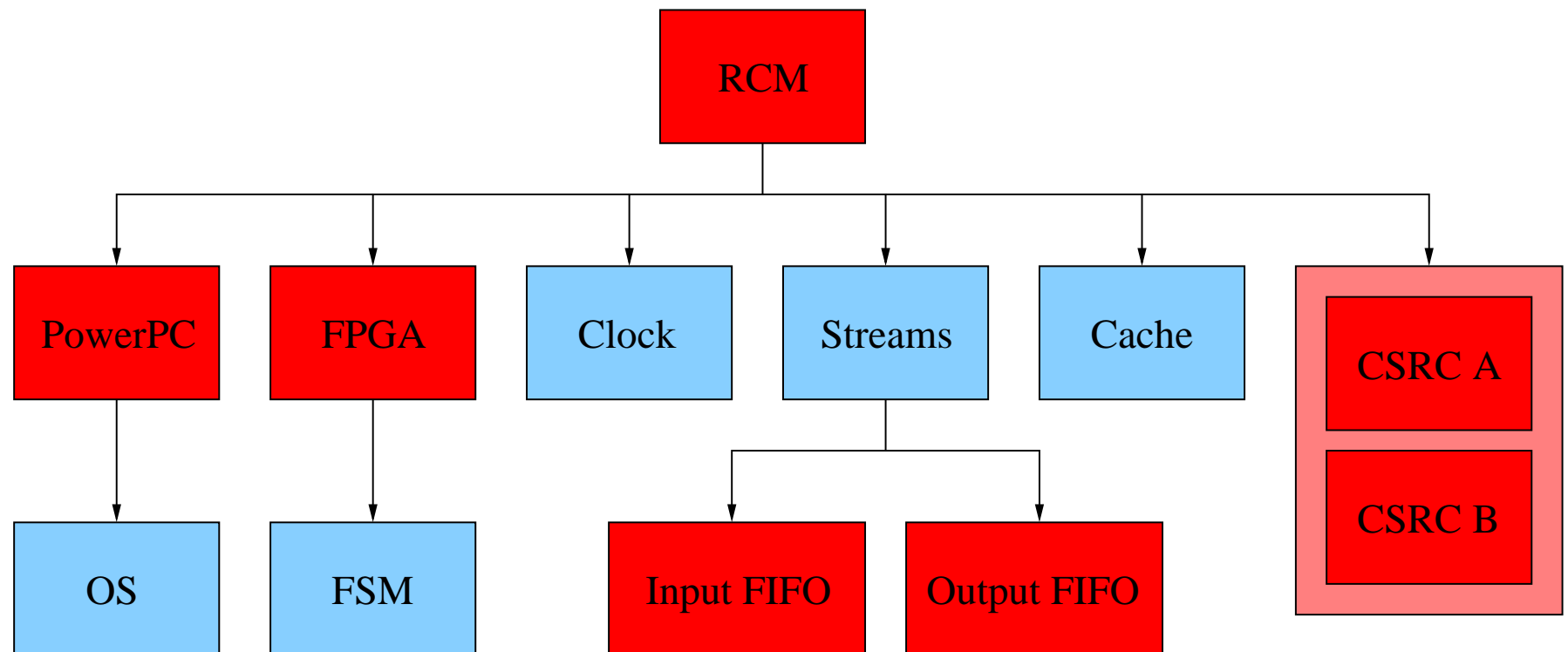
# Low Level Options

- C API for fast direct access
- ACS API for network transparent access and common interface

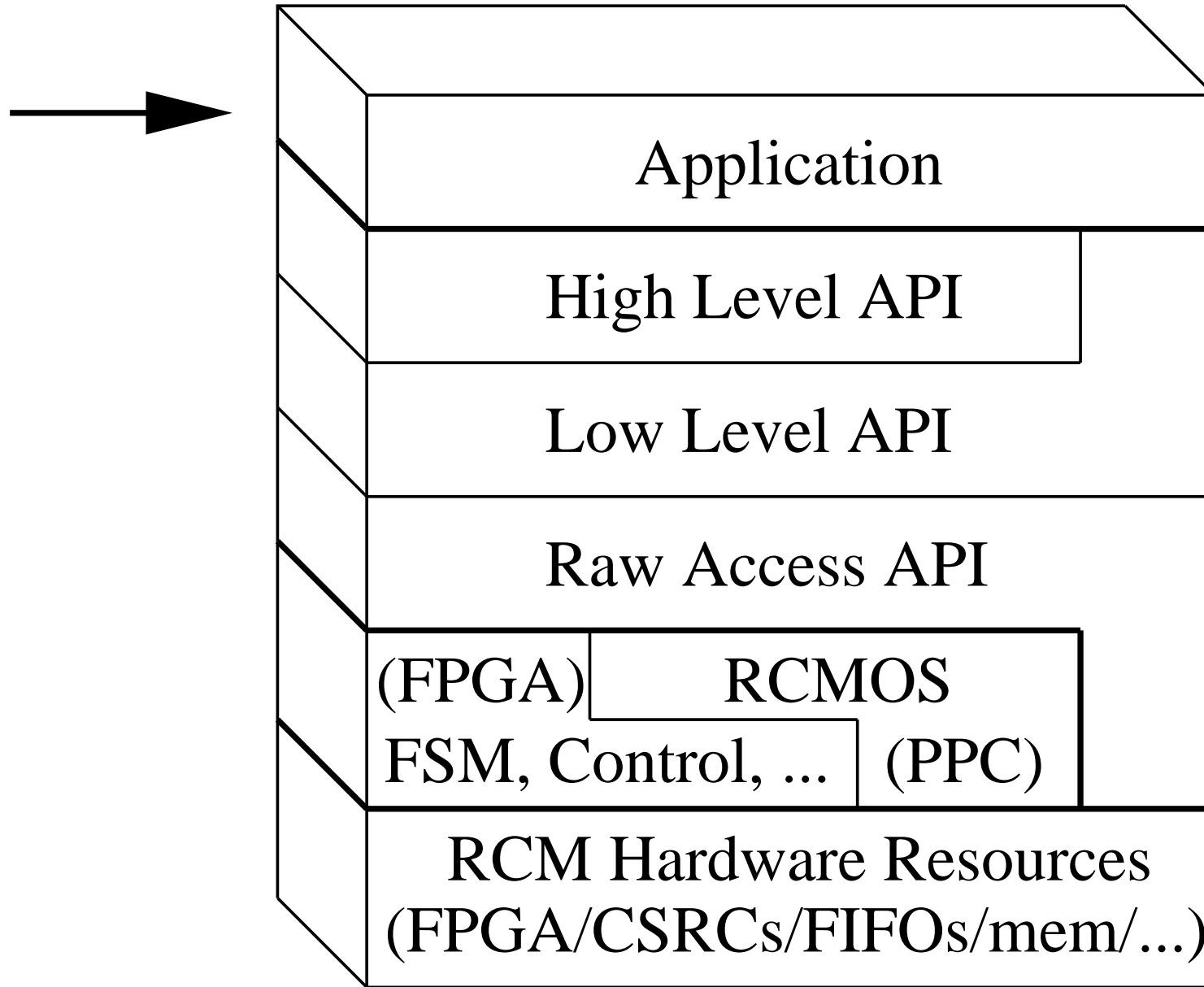


# Object Oriented High Level

- Both physical and virtual components
- Hides bit flipping and direct register access
- Simplified abstract interface

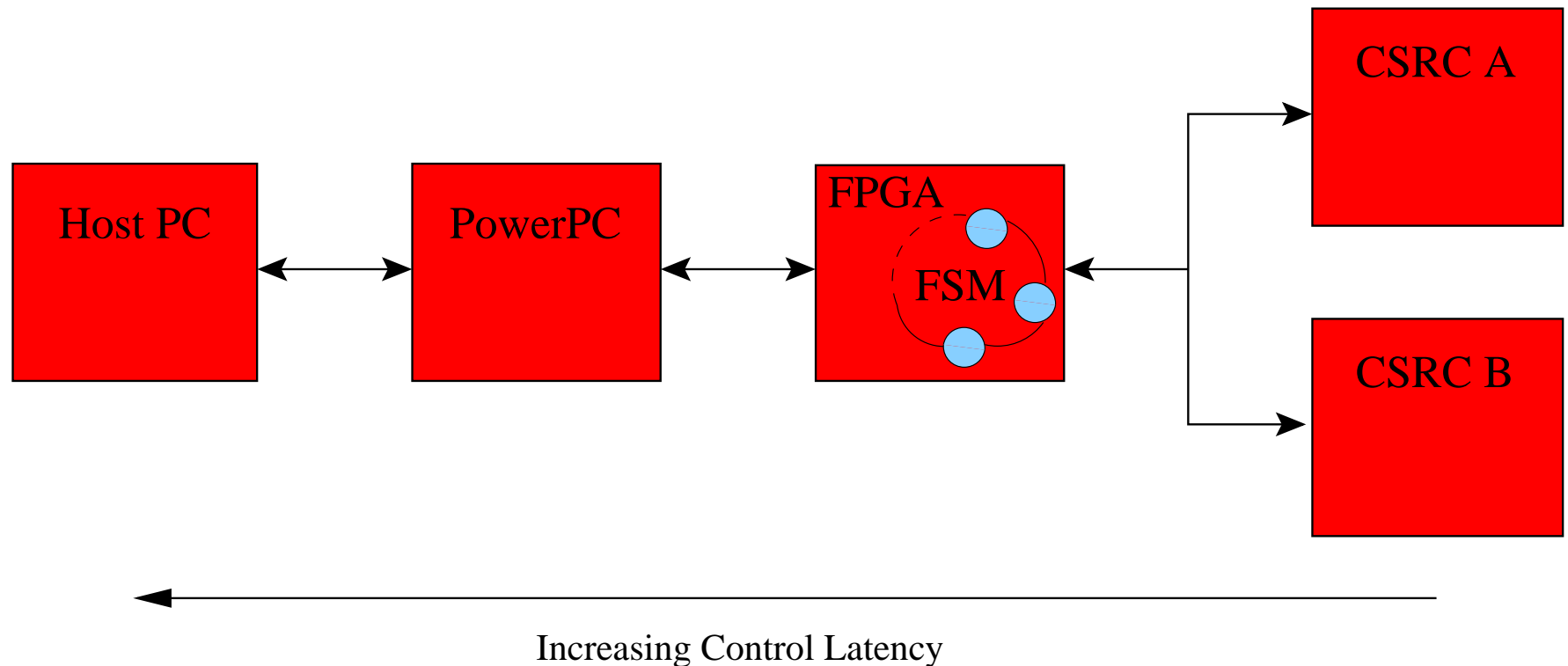


# Application Layer



# Context Switching Control

- Move logic towards CSRCs for high speed apps
- Can move logic towards host for debugging

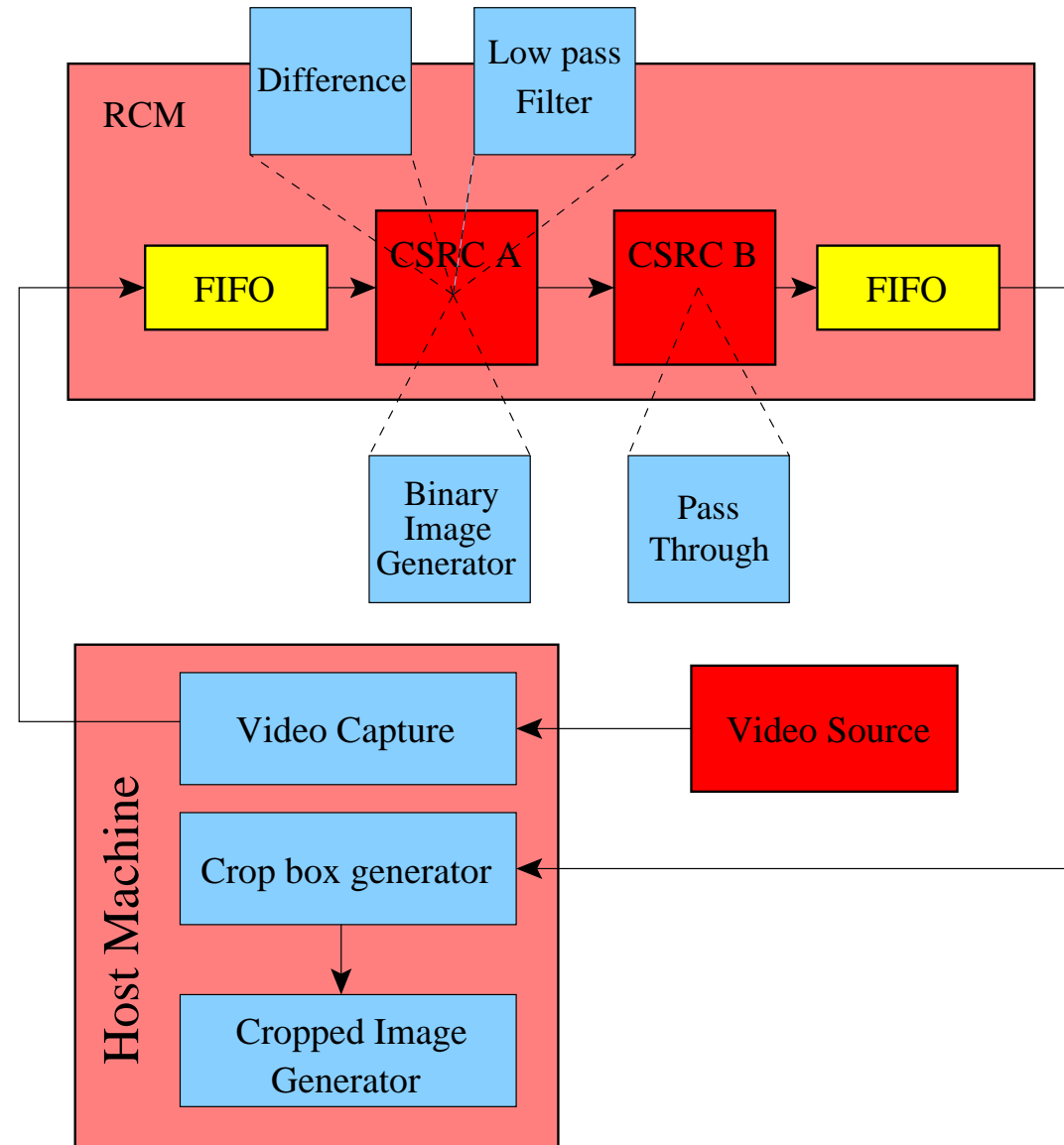


# Video (Host-Driven)

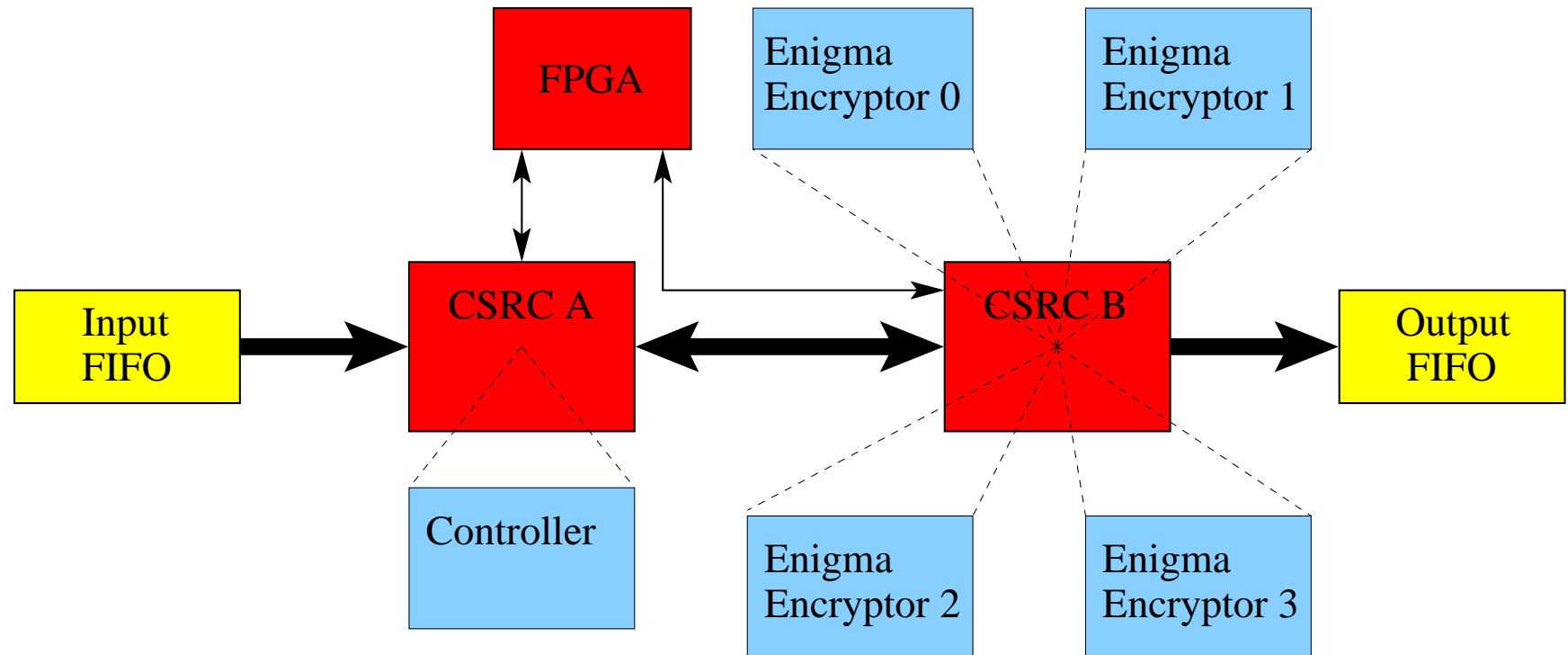
- Host/user controls video processing algorithm
- Each algorithm in its own context
- Single cycle switch (at end of a frame)



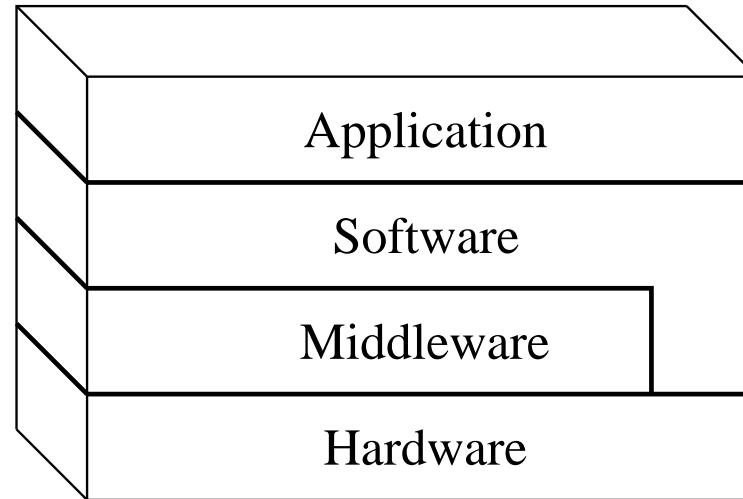
# CA $\mu$ S (Control-Driven)



# Enigma (Data-Driven)



# Results 1/2

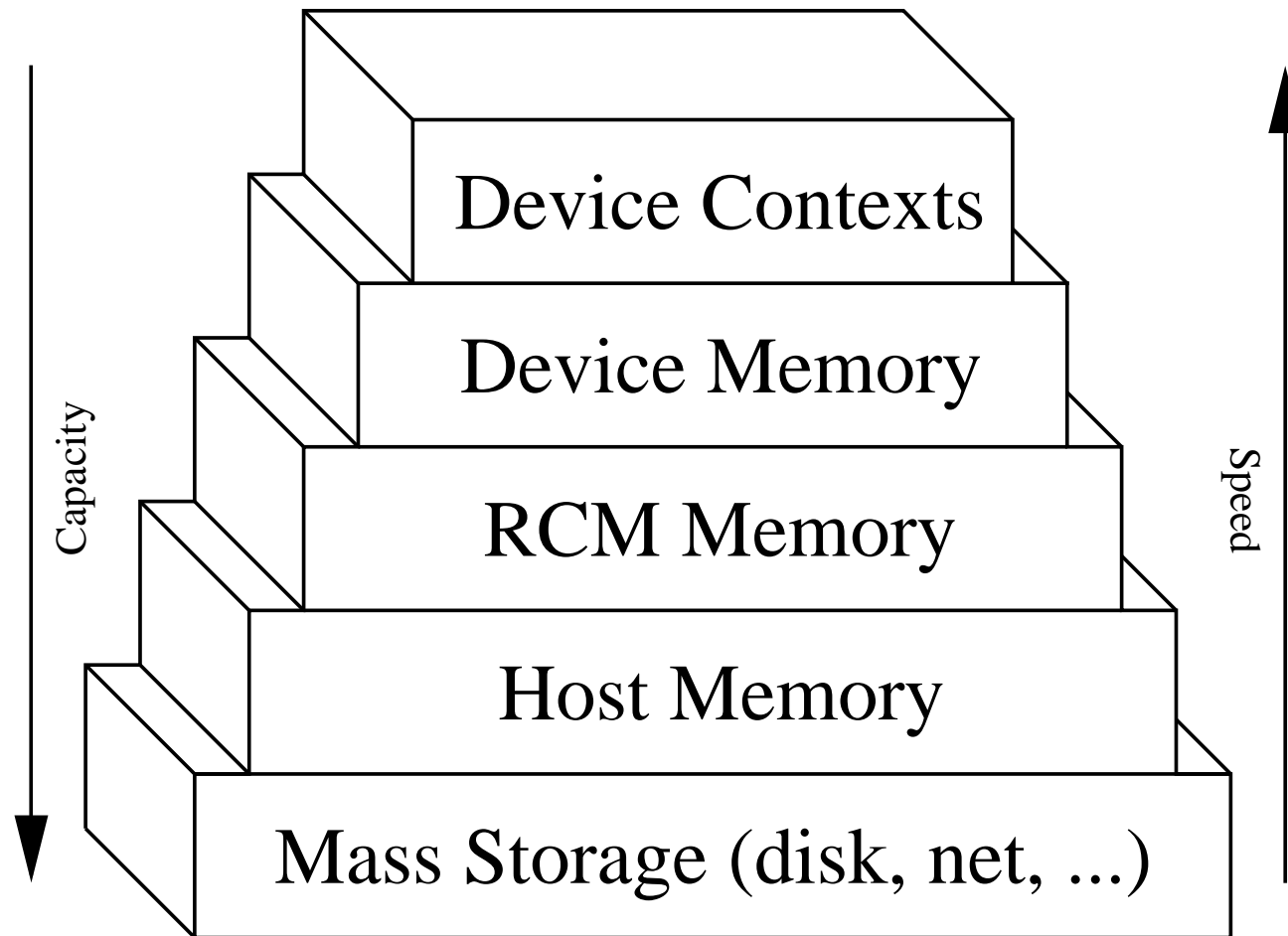


- Layered APIs
- Tests Good
- Scripting Good & works even for applications

# Results 2/2

- Met the goals:
  - Supported various context-switching applications
  - Coverage of RCM features
  - Test suite for hardware
  - Framework hides some complexity from applications
  - Performance... maybe next hardware generation
- Switching & Caching...

# Cache Hierarchy



# Switching Time

$$t_{\text{avg}} = \begin{cases} t_s & \text{if } 1 < n \leq k \\ p_s t_s + p_c t_p & \text{if } n > k \end{cases}$$

$t_{\text{avg}}$  average switching time

$t_s$  context switch time

$t_p$  context program time

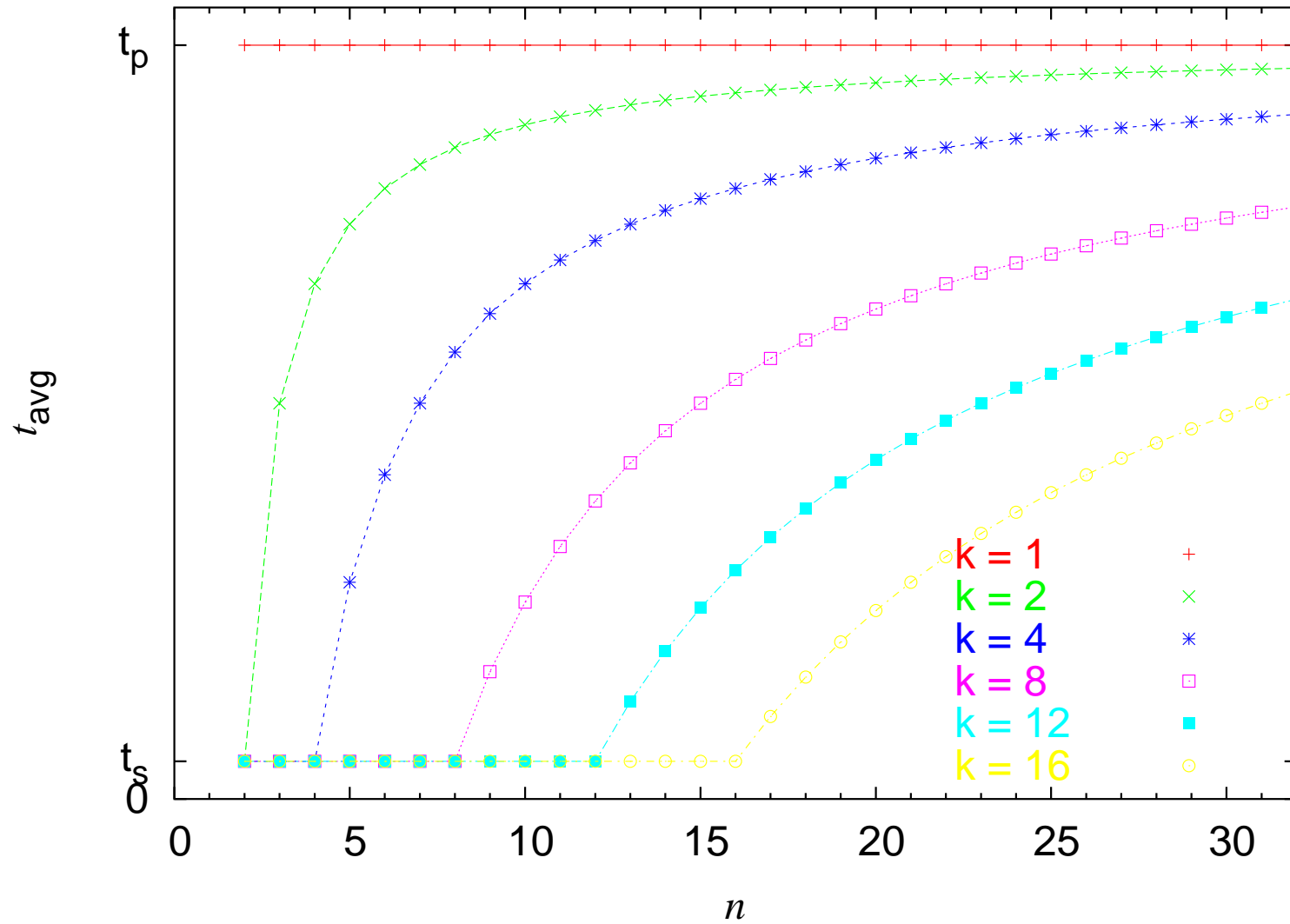
$k$  number of device contexts

$n$  number of application contexts

$p_s$  probability that context is on the device,  $\frac{k-1}{n-1}$

$p_c$  probability of a reconfigure,  $1 - \frac{k-1}{n-1}$

# Switching Time Plot



# Conclusions

- Complexities of context-switching hardware can be simplified
- Layered APIs work well towards this goal
- Applications can be written in scripting languages
- Test suite *very good* thing to have
- Move logic & data close to hardware

# Q & A

