Building systems from actors
From FPGA design to FPGA programming

Jörn W. Janneck

Xilinx Inc.

XILINX®

… as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

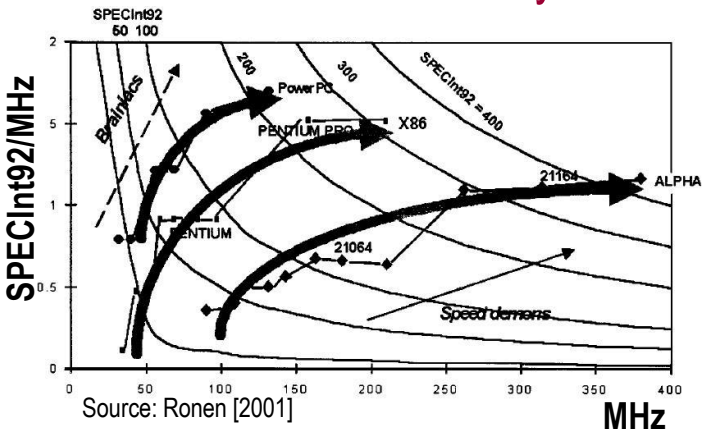
Edsger Dijkstra: The Humble Programmer
Turing Award Lecture, 1972

The Multi-core Imperative

End of roadmap for CPUs

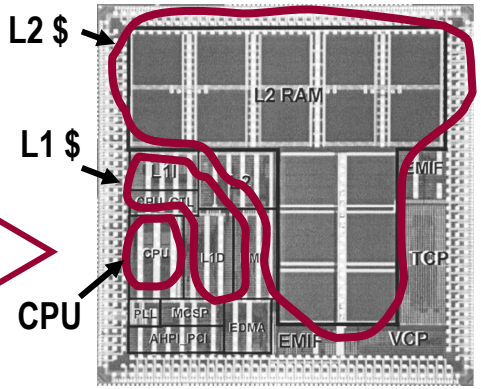
1945-2005
Sequential programming

CPUs are as smart as they can be!



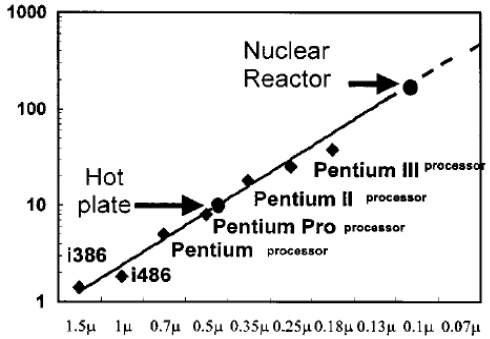
Clock frequency scaling

Spot the CPU!



Source: Agarwala [2002]
TI 6416

With Moore's law you also get leakage!

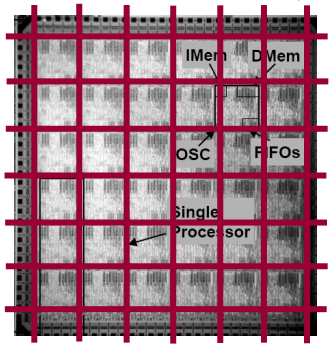


Source: Borkar [1999]

Absolute power limits

Divide and conquer

Multi-core Arrays



Source: Zu & Baas [2006]
6x6 GALS Processor Array

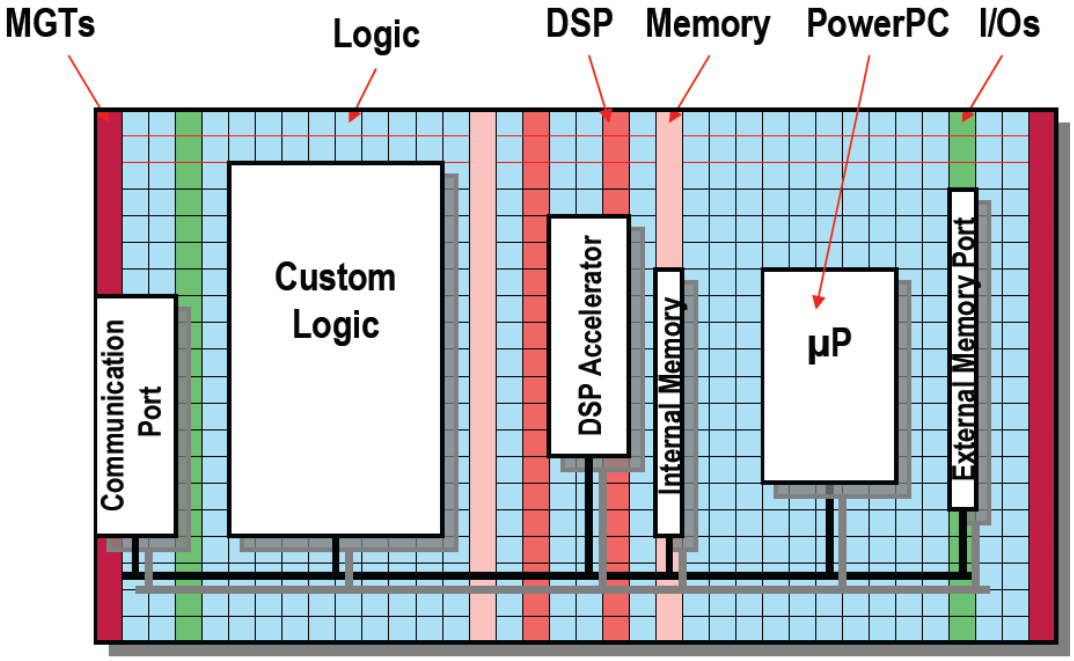
2005 - ????
Concurrent programming

The Platform FPGA

- What's special about a platform FPGA ?
  - It's big.
  - It has a variety of different elements on it.

1985:
128 4-LUTs

2006: [V5-LX]
207360 6-LUTs
10Mbit BRAM
192 ALUs



Platform FPGAs have evolved from glue logic to heavy-duty spatial computing devices.

Size matters, or the spatial software crisis

- lots of horse power, provided...
  - ... the/your problem admits to a **parallel** (-izable) **solution**
  - ... the/a/your solution admits to a **parallel implementation**
- good news / bad news
  - spatial computing devices commercially available
    - large-scale spatial computing is now feasible
  - **FPGAs are programmed by designing circuits**
    - difficult, error-prone, unproductive, not fun, few people know how to do it well

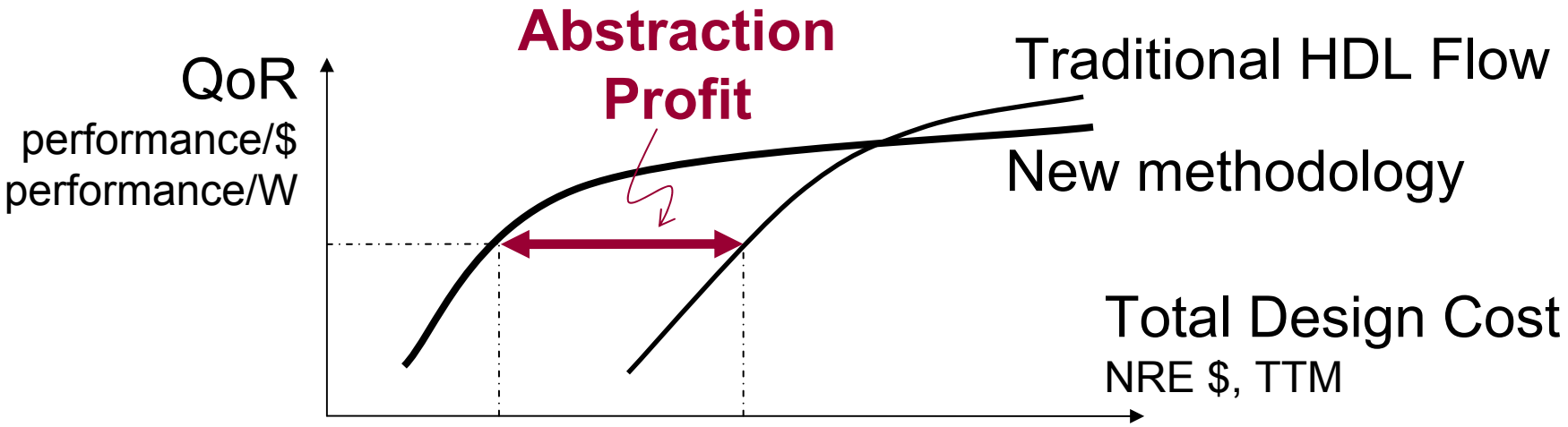
spatial programmers are facing a “spatial software crisis”

What to do?

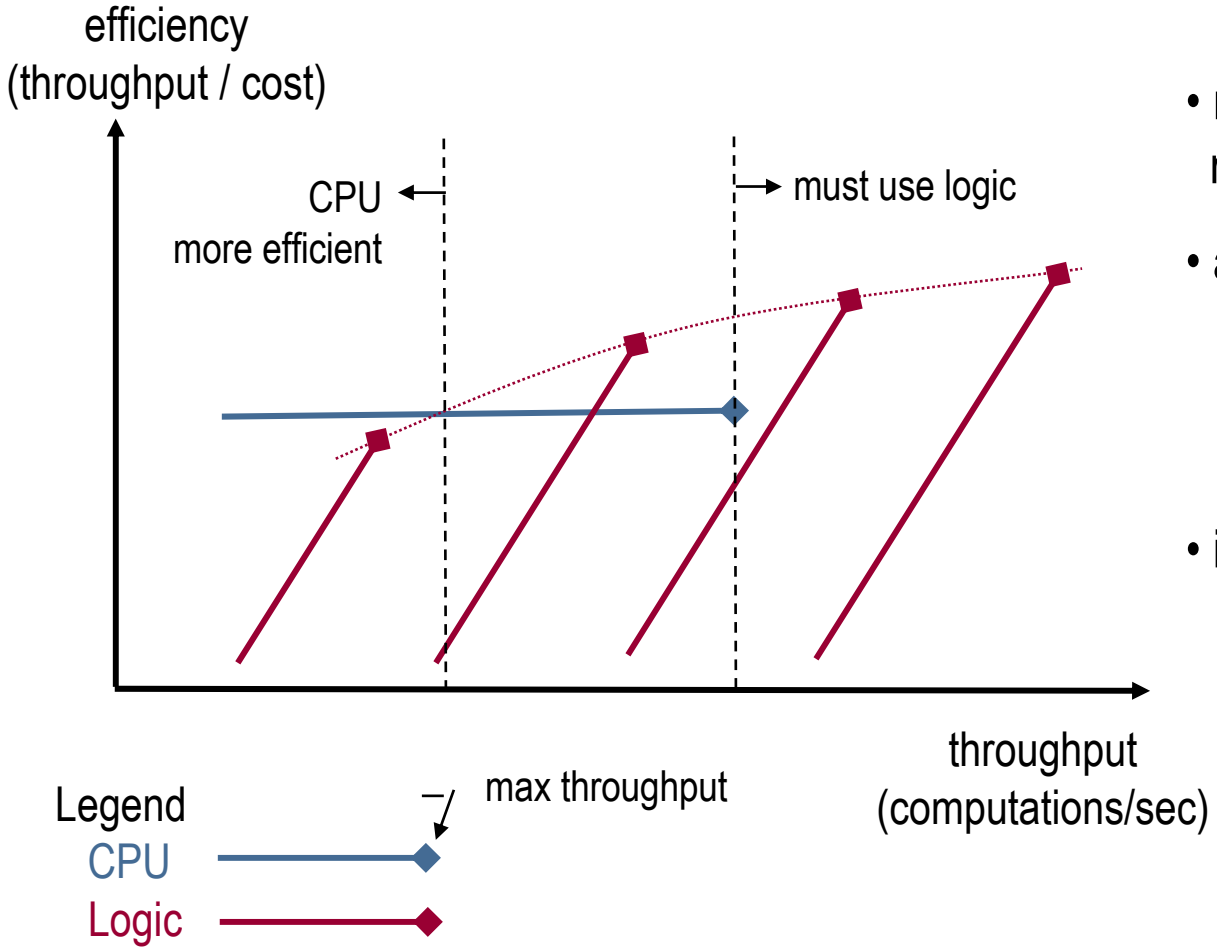
- *No Silver Bullet* [Fred Brooks, 1986]
  - tools, languages, processes
  - eliminate “accidental complexity”
  - recognize “essential complexity”
- dealing with the “essential complexity”
  - concurrency
  - communication
  - control
  - computation
- spatial components
  - should be to FPGAs what objects/classes are to processors

motivating methodology

- Quality of result (QoR) is not a design goal!
  - Performance, power, BOM cost budgets make QoR a design constraint
- The real objective is to meet the QoR target and minimize:
  - Non-recurring engineering costs (NRE)
  - Time-to-market (TTM)
- The new methodology should save on design cost by enabling
  - Design of portable, retargetable, composable IP blocks
  - Rapid design space exploration and system composition



computational efficiency
rating and comparing QoR



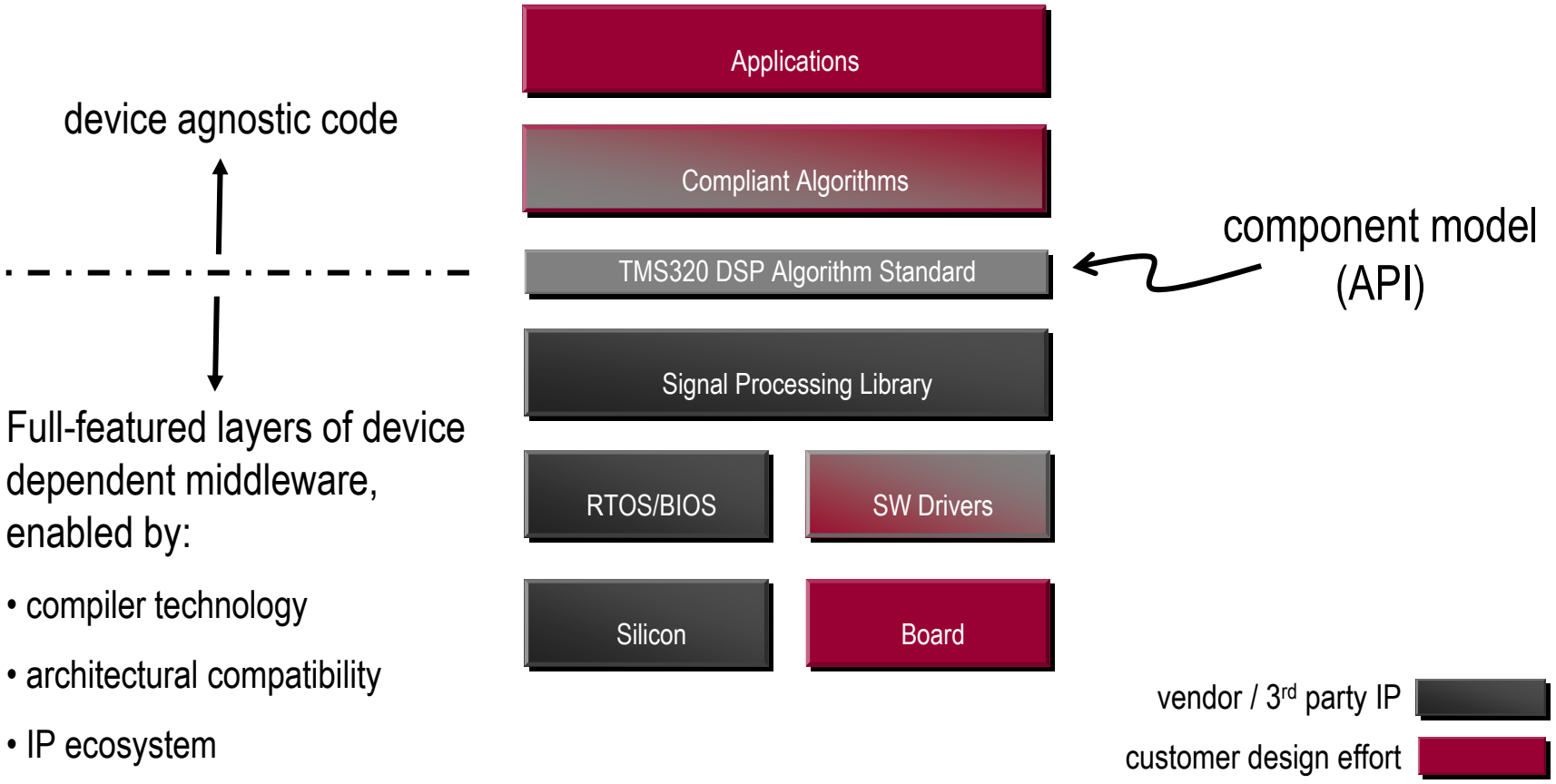
- measure throughput with a meaningful application metric
- application-centric cost
  - units of Watts or mm^2
  - for architecture comparison use only resource consumed
- in embedded apps excess performance is usually useless (hurting efficiency)

What we want to achieve

Make platform FPGAs as easy (or easier) to program than DSPs.

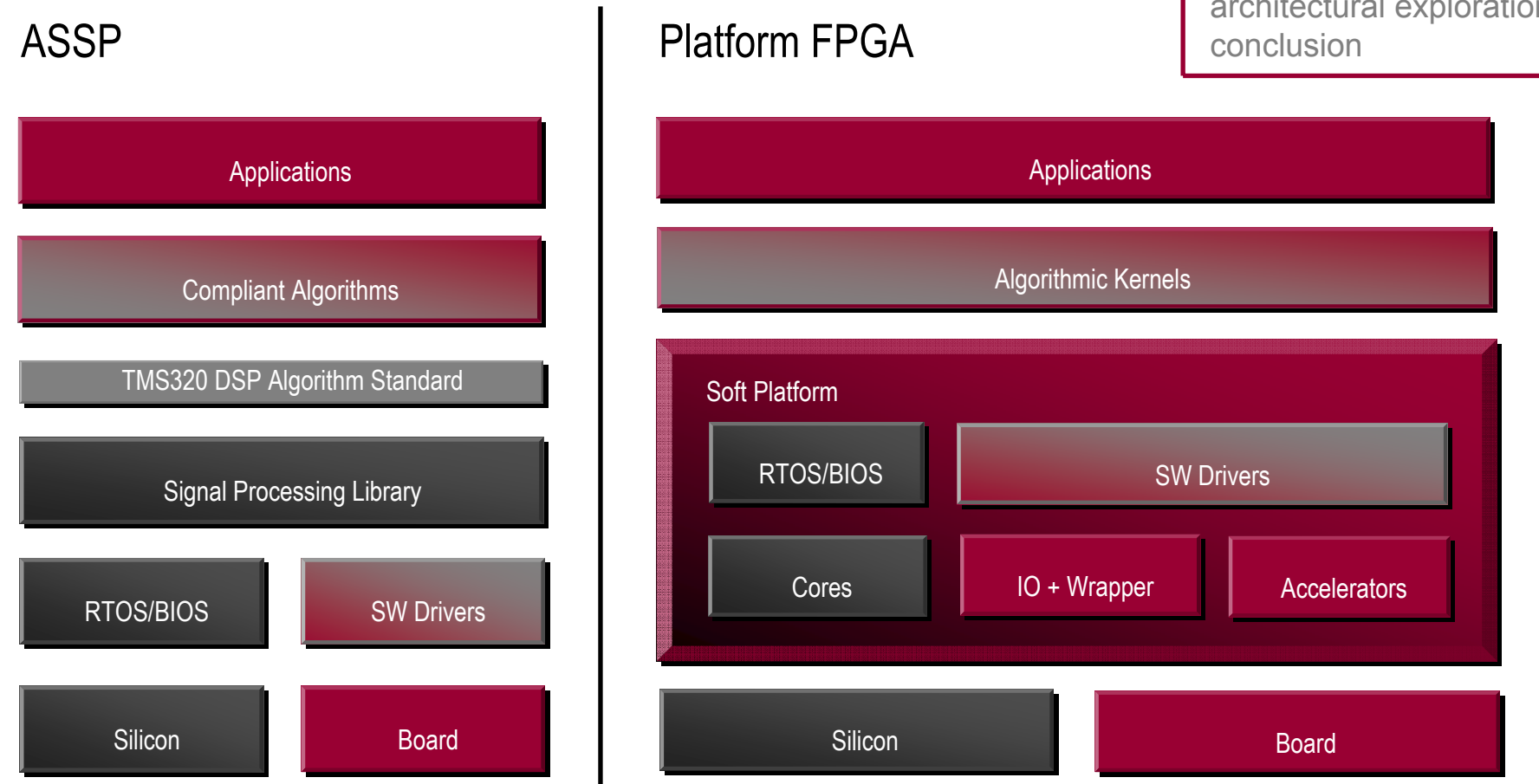
- **Move from FPGA *design* to FPGA *programming***
  - **using FPGAs "the software way"**
- Exploit inherent concurrency of FPGA platform
  - Single core processors nearing end of their roadmap
  - Multi core processors no longer have an automatic programming model advantage over FPGAs.
- Scalable FPGA performance, combined with a concurrent programming model, will enable us to close the gap and even surpass processors in ease-of-use.

ASSP Platform Programming (e.g. TI daVinci)



Platform FPGA Design

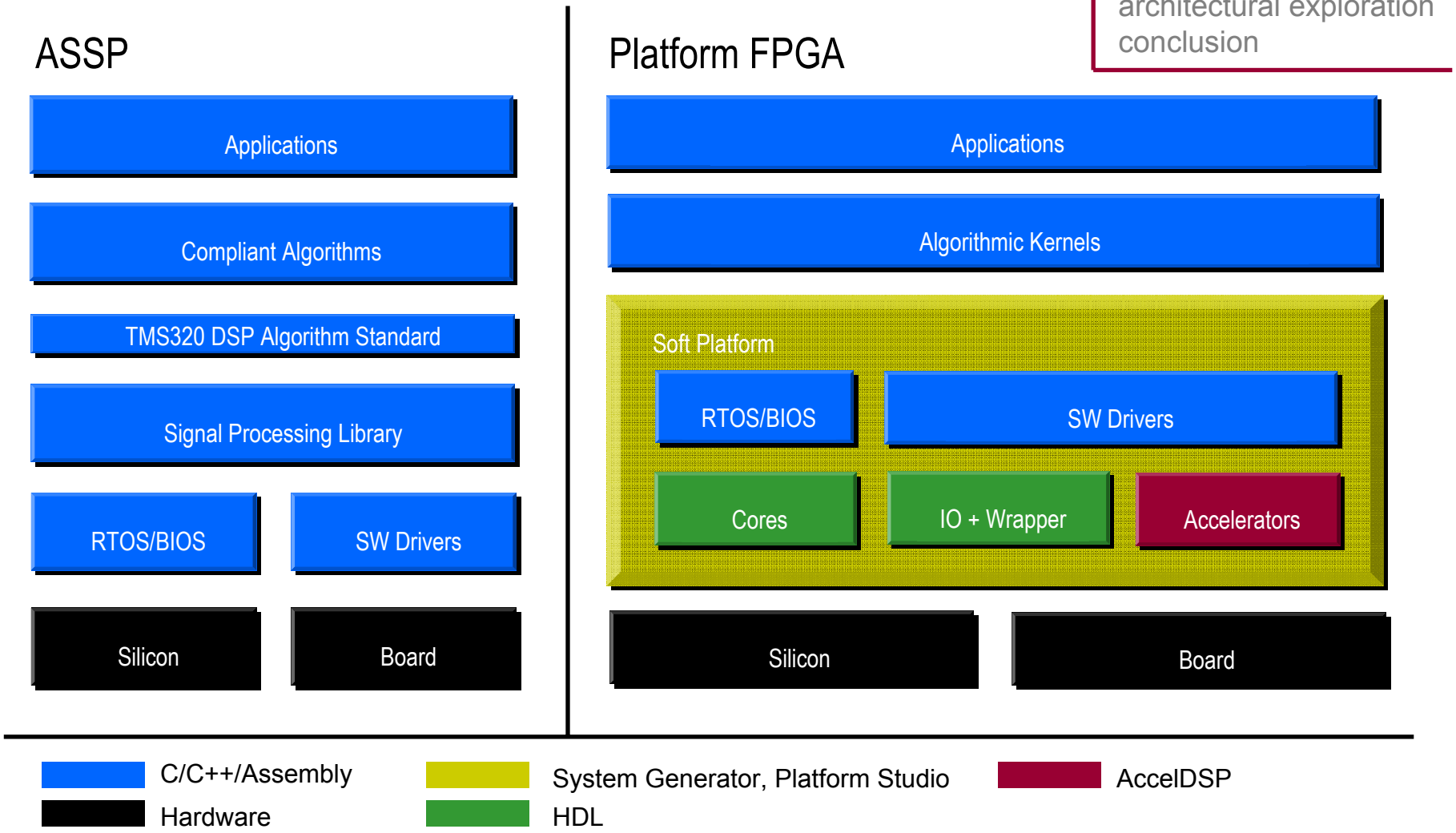
Who creates what



Soft platform approach opens door to multi-processing, but complicates methodology.

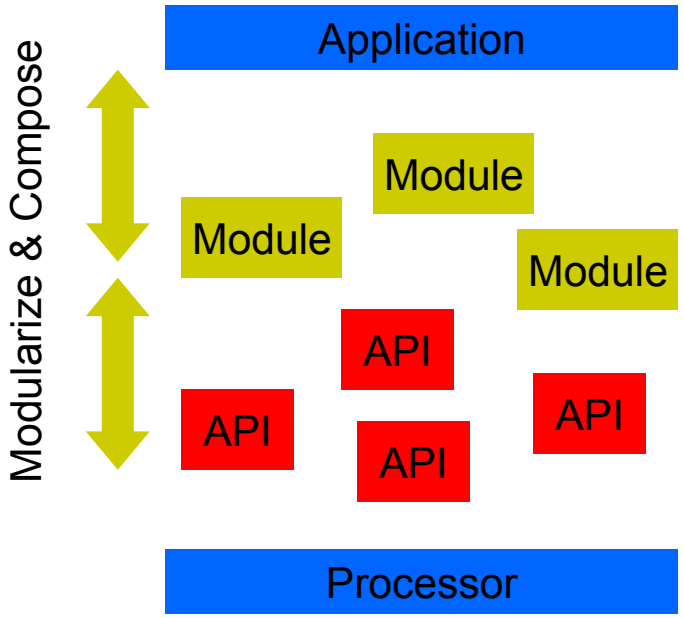
Platform FPGA Design

Languages used

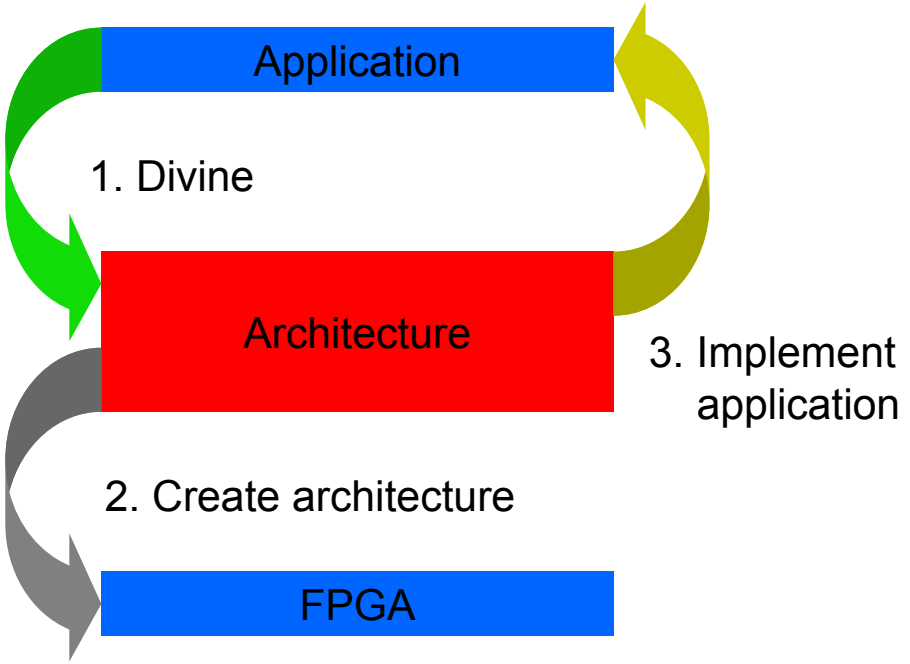


Platform FPGA Design
Implementation process

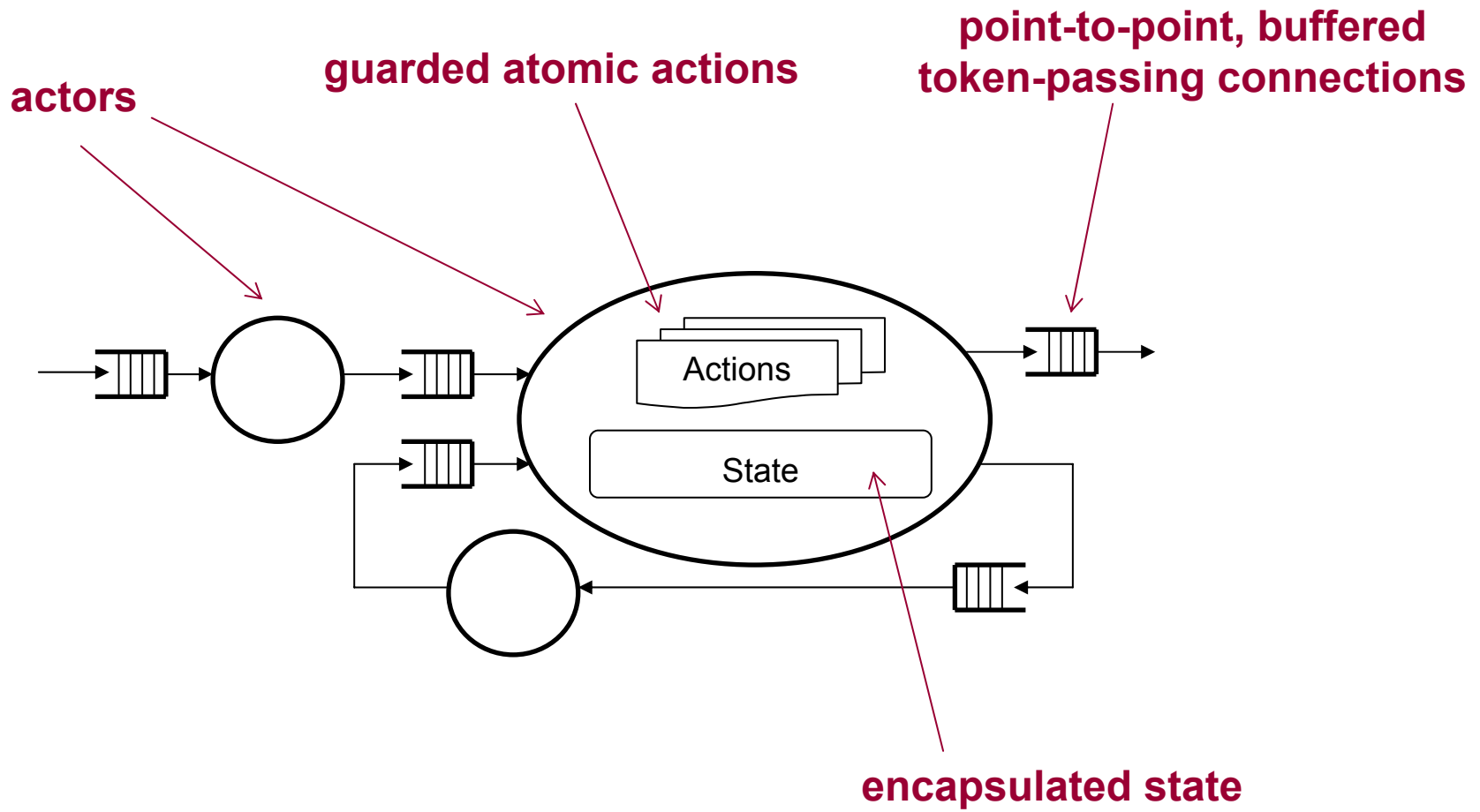
Modularize & Compose



Divine-Create-Implement



actor/dataflow programming model



actors as spatial components

- key characteristics
  - weak coupling
    - small, explicit interfaces
  - exposed parallelism
    - make it available to tools, explicit to programmers
  - encapsulation of state
  - simplicity

actors as spatial components

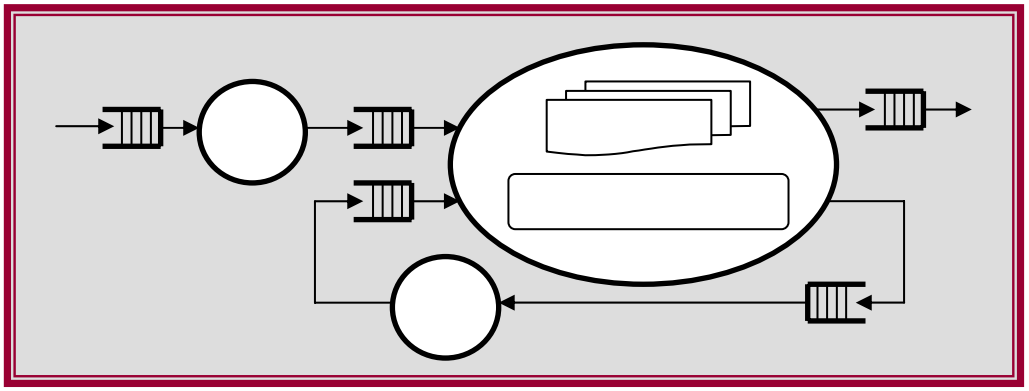
The source of...

- ... reuse
- ... portability
- ... higher-level analyses and optimizations

A conceptual base for new intellectual approaches and tools based on them:

- dataflow design patterns
- actor/dataflow refactoring
- actor/dataflow-based views on computation
  - compute graphs (causation traces)
  - models for token flows, statistical and others

actor/dataflow toolflow



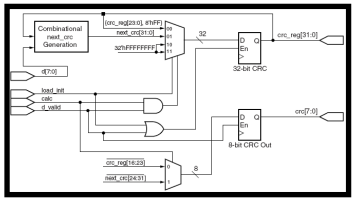
actor source
+ network

simulation

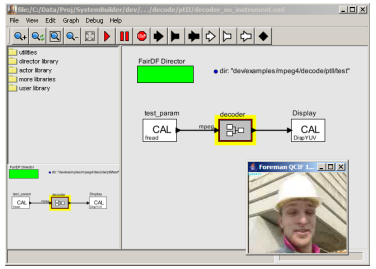
software

```
class MyActor

{ schedule();
  readPort( portNum );
  writePort( portNum );

}
```

high-level synthesis



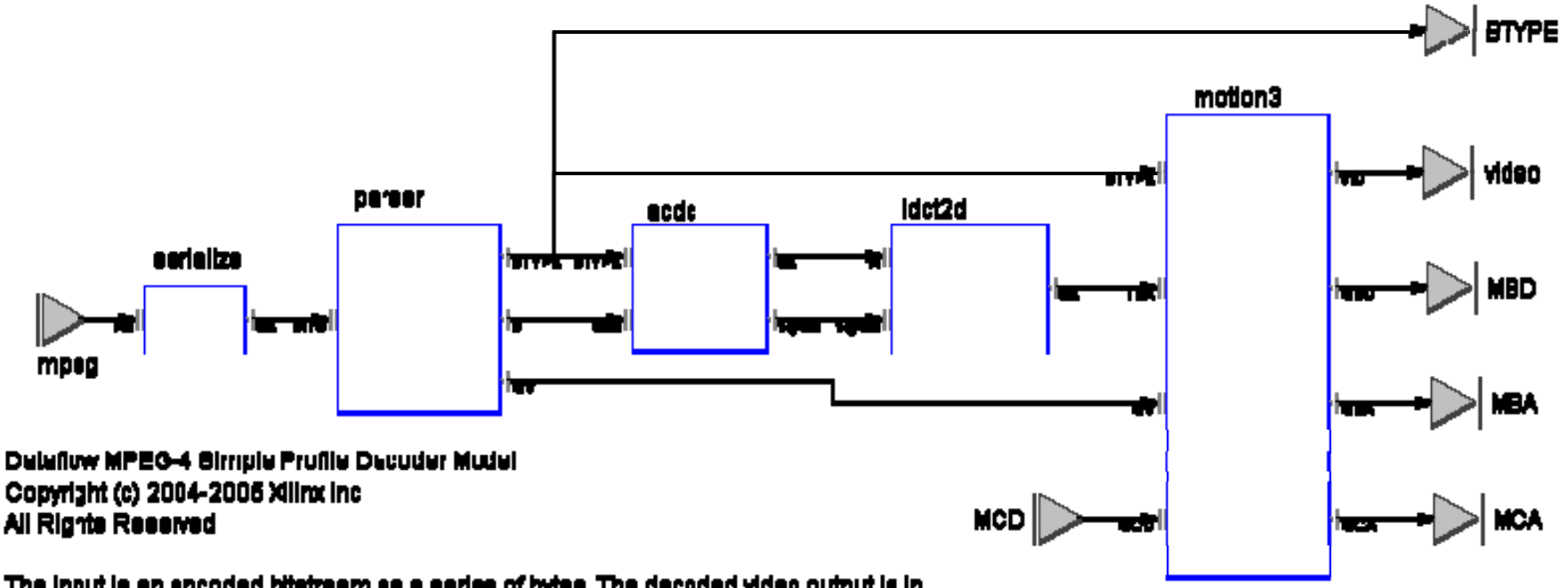
simulation



hardware

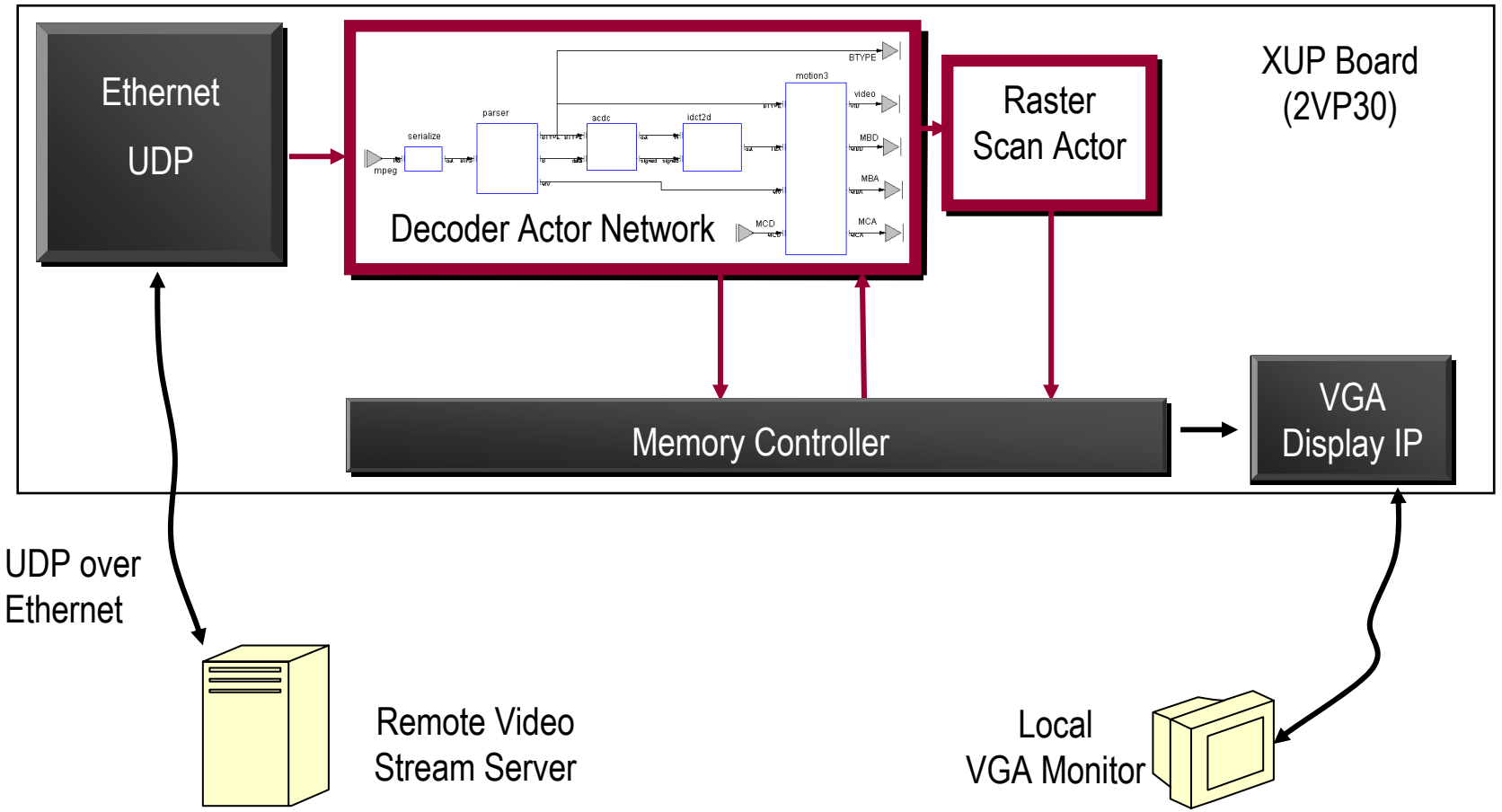
An example

MPEG4 SP decoder



The input is an encoded bitstream as a series of bytes. The decoded video output is in macroblock scan order. There are also connections to an external DRAM and multi-port controller (macroblock data and address - MBD, MBA and motion compensation read-back data and address - MCD, MCA). The BTYPE output contains information about each macroblock type. It also signals start-of-frame and frame size to the video display.

FPGA Programming In Practice
Networked MPEG-4 Viewer



MPEG-4 SP Decoder QoR

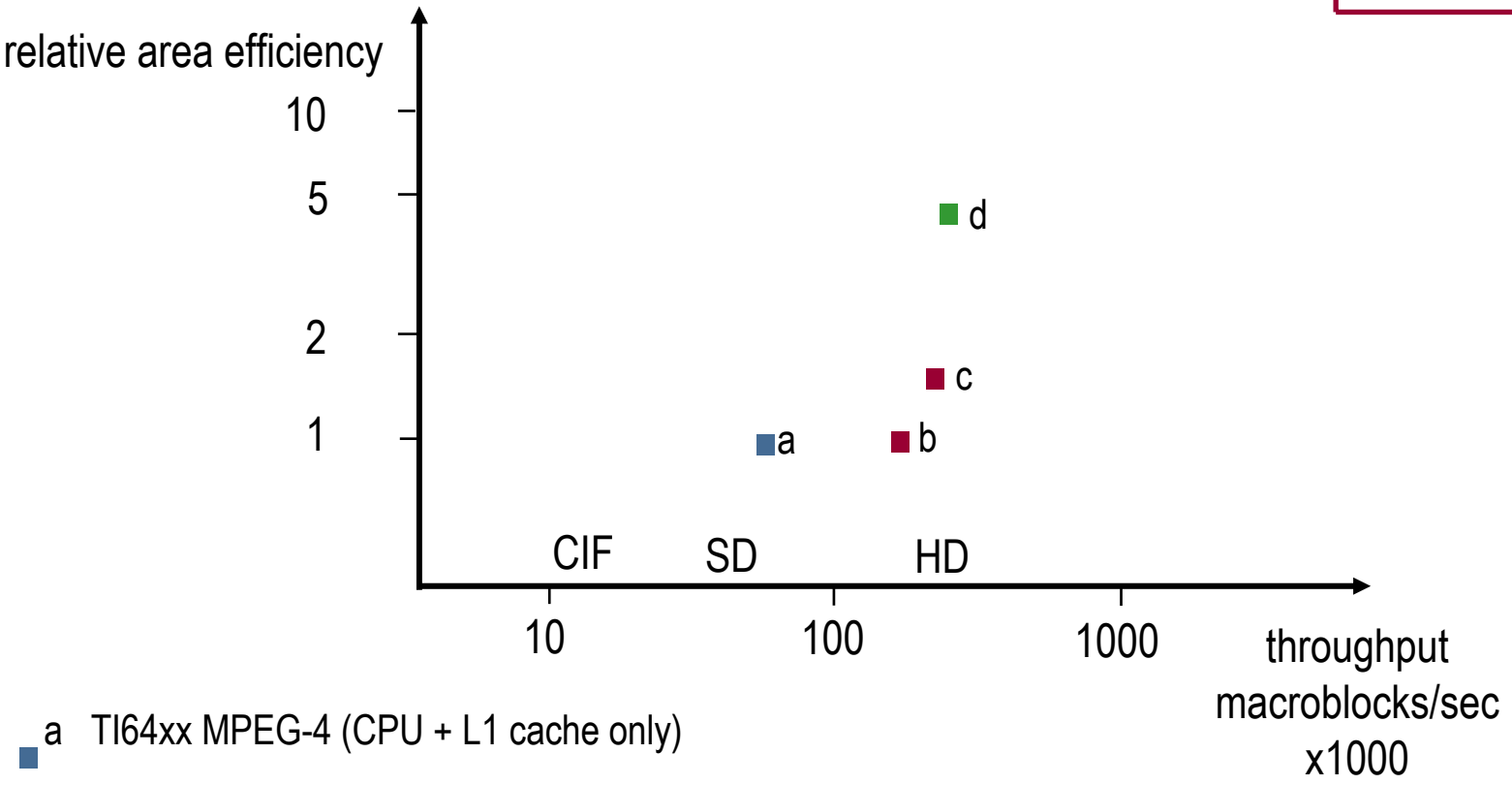
Version	Area					Performance
	Slice	LUT	FF	BRAM	MULT	
VHDL IP [1] (15000 lines)	4637	7923	2637	26 [2]	34	4-CIF image size 180K macroblock/s @ 100MHz Requires ZBT SRAM framebuf
CAL decoder (4000 lines)	3872	7720	3576	22 [3]	7	HD image size 243K macroblock/s @ 120MHz Interfaces to DRAM framebuf I-frame parsing: 50 Mbit/s

[1] http://www.xilinx.com/bvdocs/ipcenter/data_sheet/ds520_prod_brf.pdf

[2] BRAM-limited to 4-CIF image size.

[3] Supports HD image size. Reduces to 16 BRAMs for 4-CIF image size.

Comparing Decoder Solutions



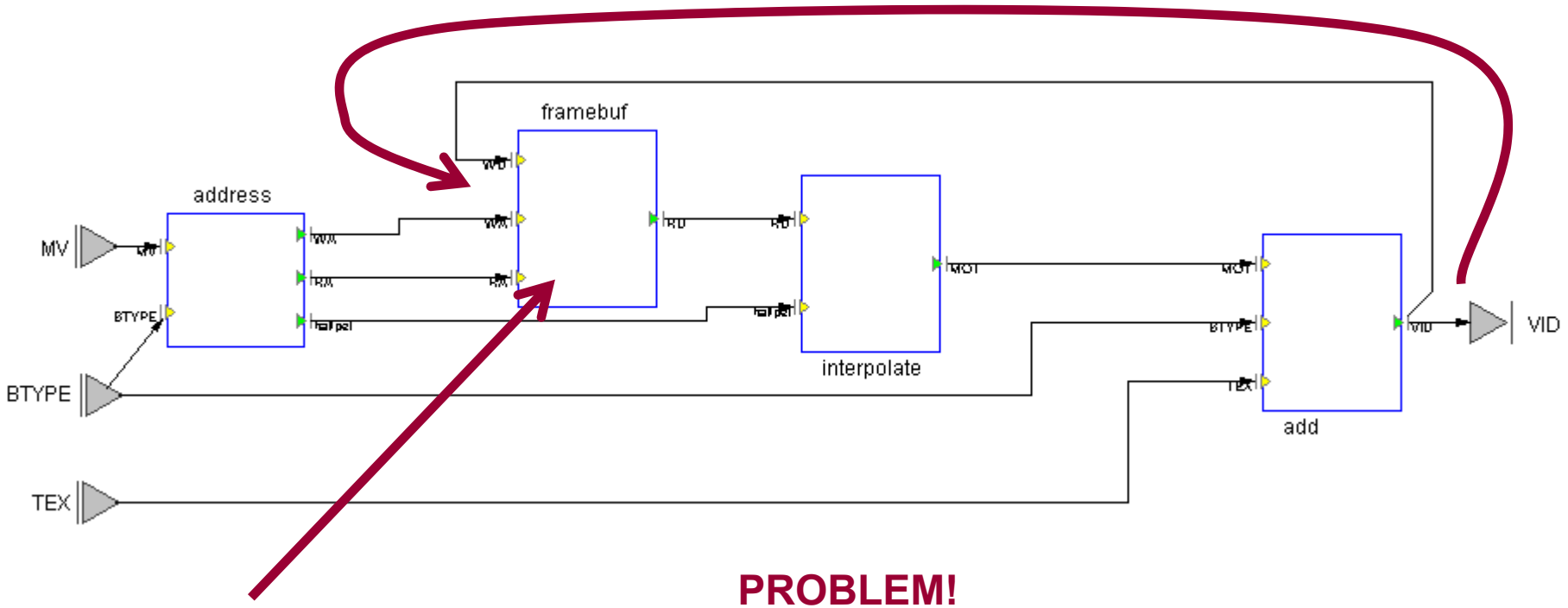
- a TI64xx MPEG-4 (CPU + L1 cache only)
- b FPGA MPEG-4 using traditional HDL flow (12 MM effort)
- c FPGA MPEG-4 using actor/dataflow synthesis (3 MM effort)
- d ISSCC'06 H.264 capable (includes periphery)

Smaller, Faster, Easier
Too good to be true?

- This is what happens when design effort is constrained.
- The key is enabling architectural exploration with rapid turn-around time.
- New decoder architecture incorporates many improvements over original design in motion compensation, AC/DC reconstruction, parser, 2-d IDCT.
- Approximate manpower numbers:
  - VHDL decoder: 12 months
  - Dataflow decoder: 3 months

Architectural Exploration
MPEG4 Motion Compensator

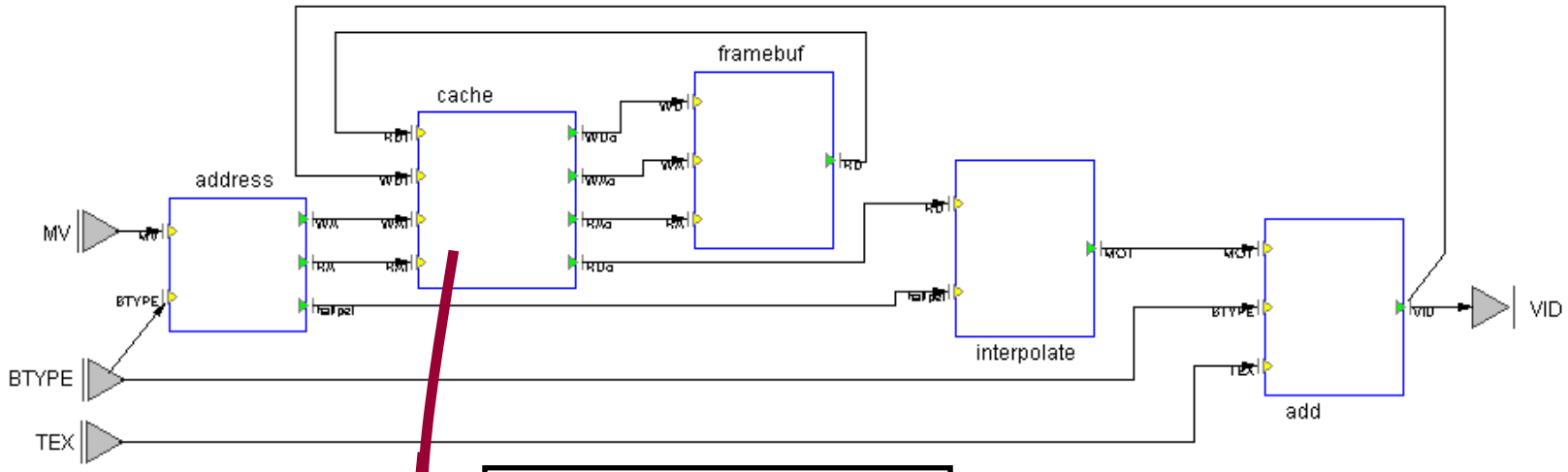
video stream feedback



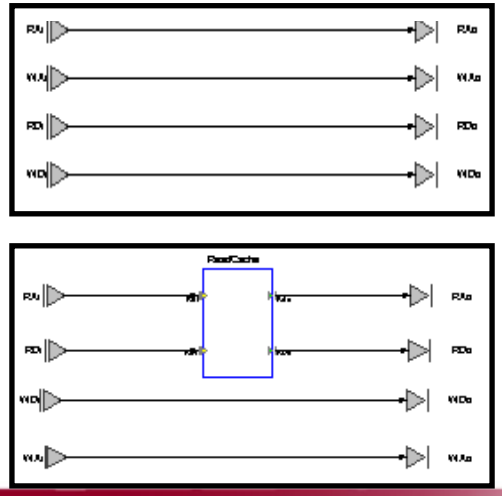
video frame buffer
(off-chip DRAM)

PROBLEM!
Memory latency for random access reads and writes prevents real-world operation at HD rates.

First Step:
Try on-chip cache



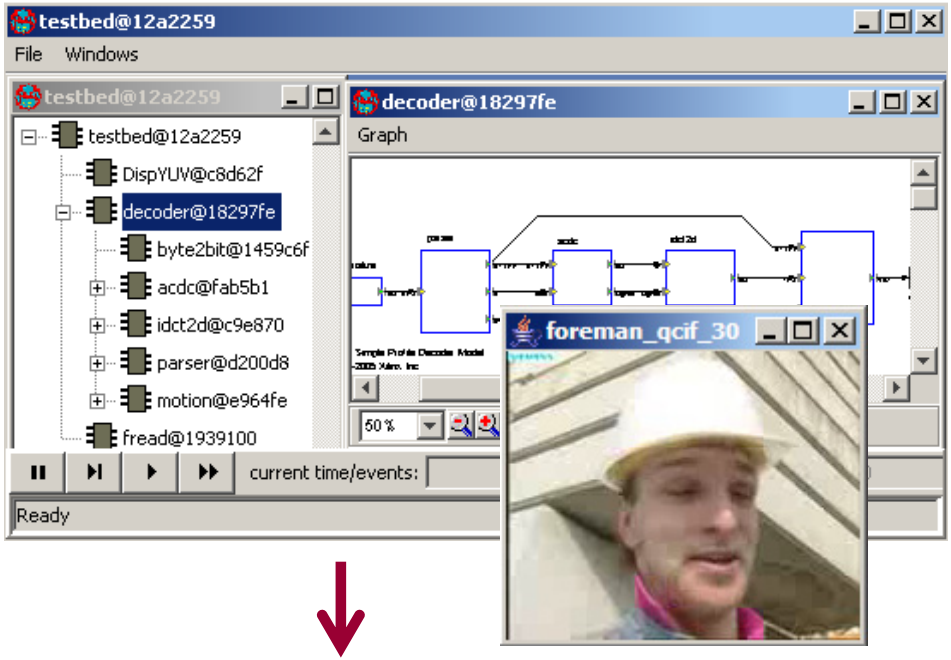
- **Break the address and data streams, insert a cache placeholder.**
- **Insert different policies, see what happens.**



policy1
Pass-through just to make sure model is OK.

policy2
Insert a cache actor in the read path and monitor statistics.

Simulation result with policy2



Monitor console

```
Frame 1 OK time: 28111ms
Frame 2 OK time: 23834ms
Requests: 49456, Hits: 45360
Miss rate: 8.28%
Frame 3 OK time: 27369ms
Requests: 98704, Hits: 90512
Miss rate: 8.30%
```

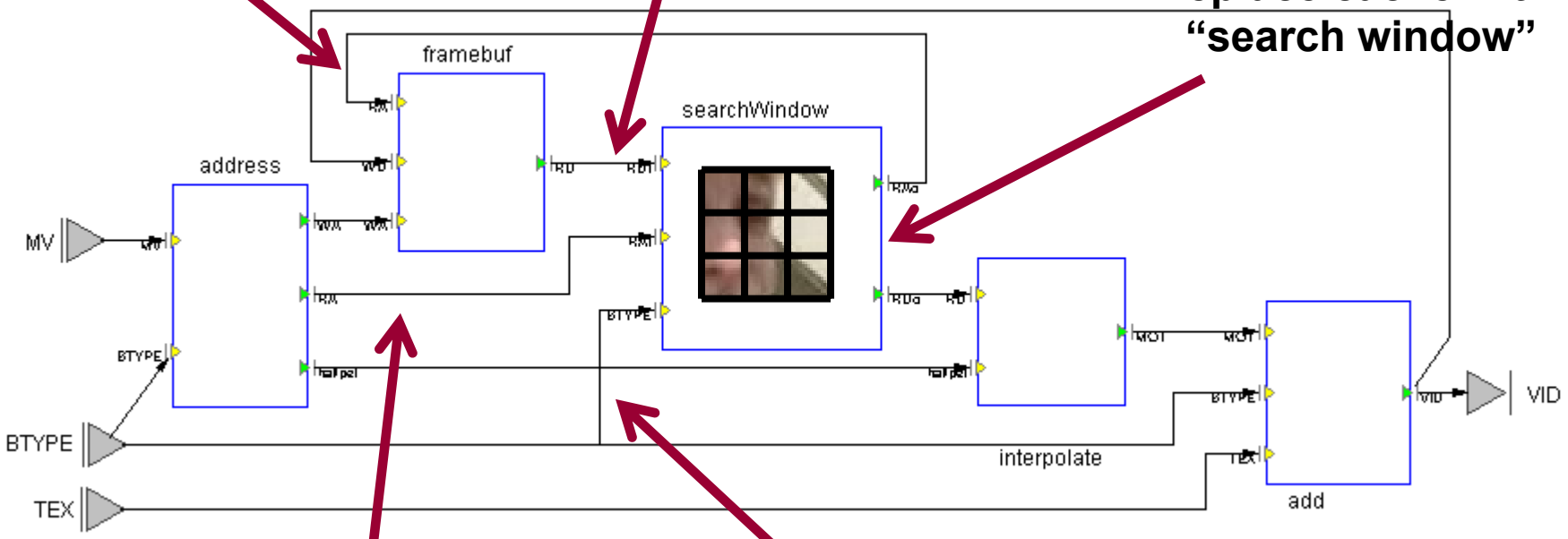
- **Memory controller performance**
  133MHz clock
  32 pixel cache line fill
  in ~18 cycles
- **Worst case compensation is 81 reads for an 8x8 block.**
- **8.3% miss rate implies average read is ~ 2.4 cycles**
- **Rate limit is 44 Mpixel/s**
- **HD (1920p, 4:2:0, 30fps) rate target is 93.3 Mpixel/s**
- **Options for improvement**
  - more expensive controller
  - much better cache policy
  - application-aware prefetch

Step2:
Application-aware prefetch

prefetch requests to frame buffer

prefetch data

replace cache with "search window"



compensation addresses now relative to search window

search window senses block type

Results of prefetch strategy

- Better performance
  - prefetch needs to operate at 3x pixel rate
  - exploits longer burst read with application-awareness (longer cache line did not help policy2 significantly)
  - 64 pixels in 26 cycles → average read is ~ 0.4 cycles
  - peak theoretical performance is 111 Mpixel/s
  - **exceeds HD rate target with cheap DRAM**
- Substantial change to overall model behavior, but
  - impact limited to two actors
  - no refactoring of control in other actors needed

Final remarks

- Several real problems
  - productivity, portability, robustness, ... in spatial programming
- Need to identify and implement
  - the right component model
  - the right language for it
  - a good translation to a variety of computing devices
- High payback
  - Whoever gets them right first wins big.
- Uncertain which language FPGA users will use in 2016

  ... but we know some which they are probably NOT going to use very much: Verilog, VHDL, C, Java

What's the best thing you could be working on, and why aren't you?

Paul Graham, "Good and Bad Procrastination"

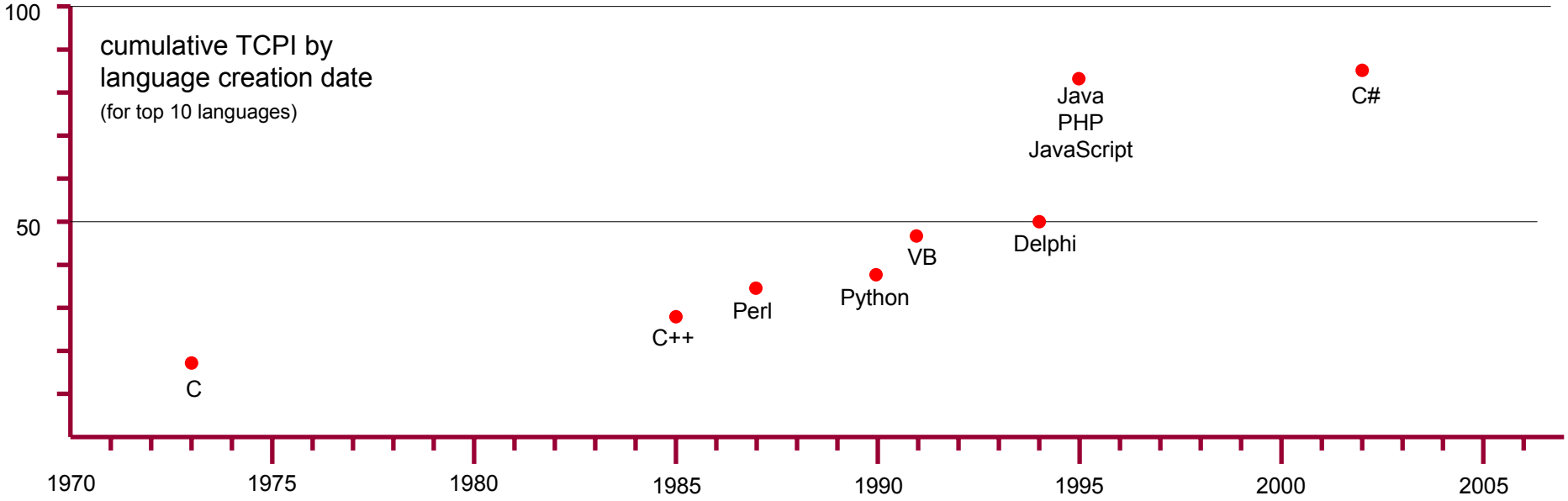
Thank You.

Credits:
Dave B. Parlour, Ian A. Miller, Johan Eker, Edward A. Lee, and many others.

CAL actor language: embedded.eecs.berkeley.edu/caltrop
Ptolemy II: ptolemy.eecs.berkeley.edu

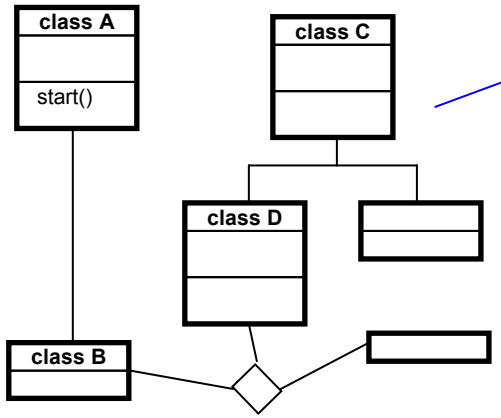
programming language adoption

Name	TPCI	TPCI cum.	Year
C	17.66%	17.66%	1973
C++	11.06%	28.73%	1985
Perl	5.48%	34.20%	1987
Python	3.47%	37.67%	1990
VB	9.73%	47.40%	1991
Delphi	2.15%	49.54%	1994
Java	21.17%	70.72%	1995
PHP	9.86%	80.58%	1995
JavaScript	2.20%	82.78%	1995
C#	3.07%	85.85%	2002

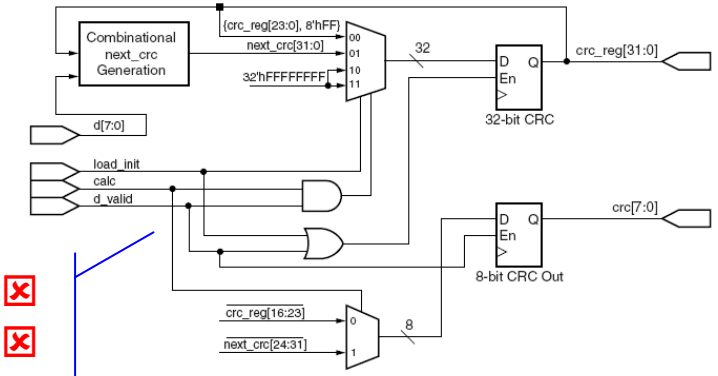


source: TIOBE Programming Community Index, TPCI, October 2006, http://www.tiobe.com/tpci.htm

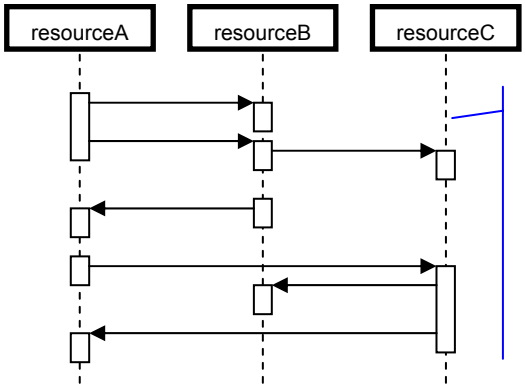
Merging Mindsets:
Software Design vs. Hardware Design



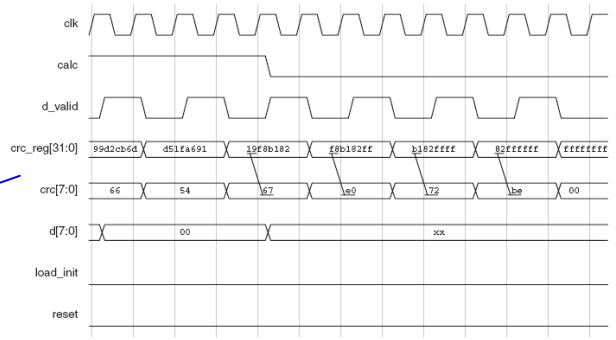
- ☑ Encapsulation
- ☑ Abstraction
- ☑ Portability
- ☑ Re-use



- Implementation Detail ☒
- Control Logic ☒
- Interface Glue ☒
- Concurrency ☑
- Communication ☑
- Architecture ☑



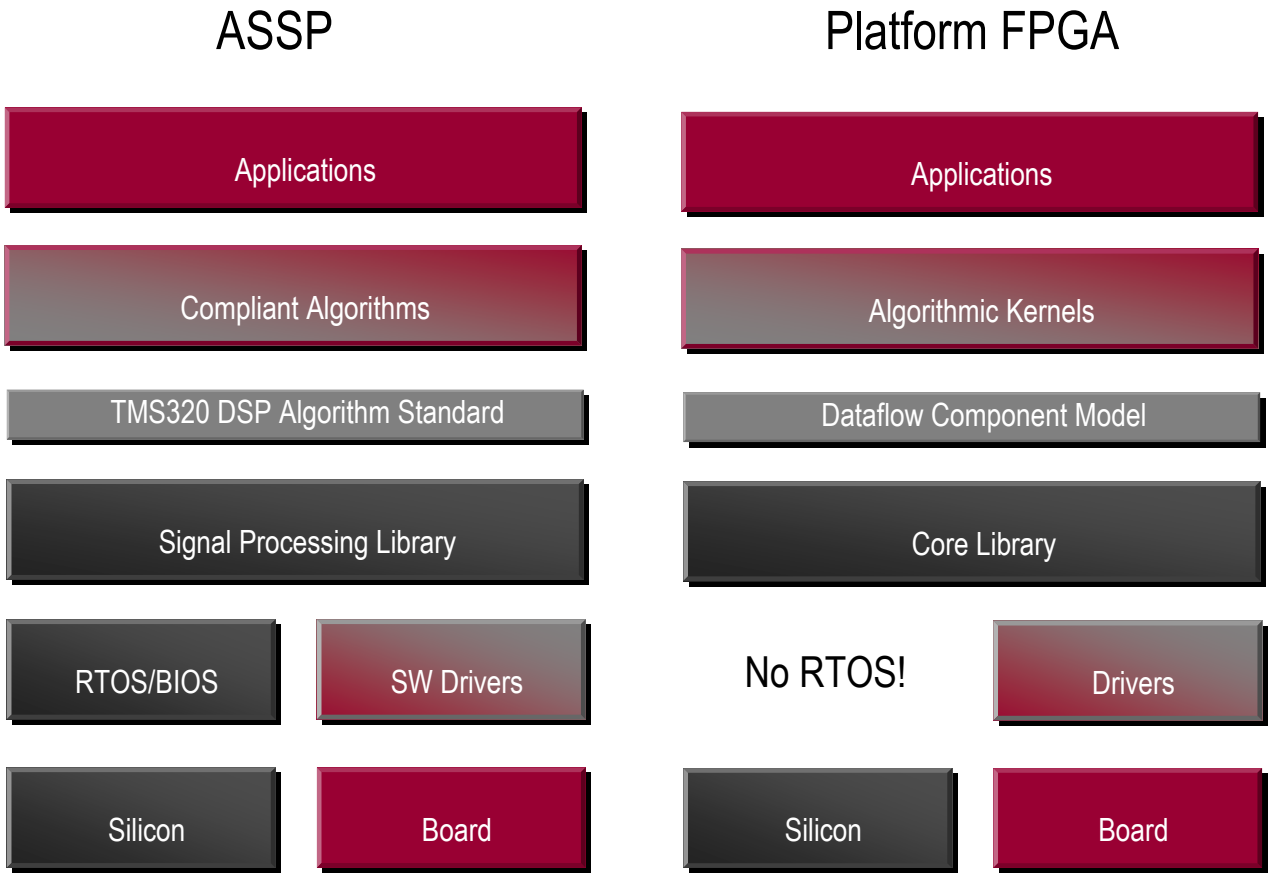
- ☑ Events
- ☑ Protocols
- ☑ Ordering
- ☒ Sequential execution



- Clocks ☒
- Signals ☒
- Timing ☒

Combining the strengths of both paradigms can bring about a radical improvement in hardware/software system design productivity.

Platform FPGA Programming



Actor + dataflow network description

IP in any format with dataflow component model interface, e.g.

- **SystemGenerator**
- **AccelDSP**
- **Platform Studio**
- **Coregen**
- **HDL**

Soft platform + program replaced with a single-source concurrent application.

[A] language that makes source code ugly is maddening to an exacting programmer, as clay full of lumps would be to a sculptor.

At the mention of ugly source code, people will of course think of Perl. But the superficial ugliness of Perl is not the sort I mean. **Real ugliness is not harsh-looking syntax, but having to build programs out of the wrong concepts.** Perl may look like a cartoon character swearing, but there are cases where it surpasses Python conceptually.

Paul Graham, "The Python Paradox"

Designing for programmers

A really good language should be both clean and dirty: cleanly designed, with a small core of well understood and highly orthogonal operators, but dirty in the sense that it lets hackers have their way with it. [...]

A good programming language should have features that make the kind of people who use the phrase "software engineering" shake their heads disapprovingly. [...]

To be attractive to hackers, a language must be good for writing the kinds of programs they want to write. And that means, perhaps surprisingly, that it has to **be good for writing throwaway programs.**

Paul Graham, "Being Popular"