

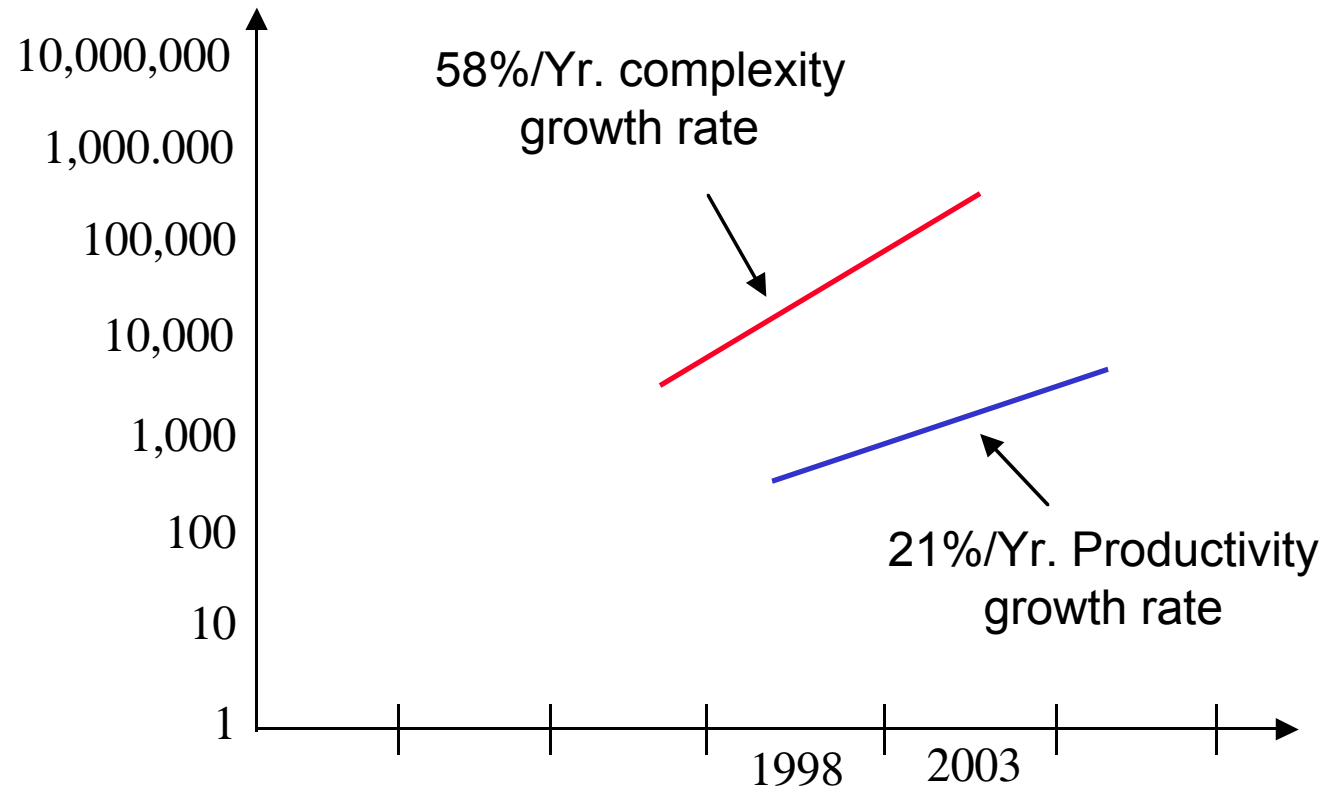
Rapid Application Optimization Using Extensible Processors



Michael Carchia and Albert Wang



Problem: Getting Hot Chips to Market in Time



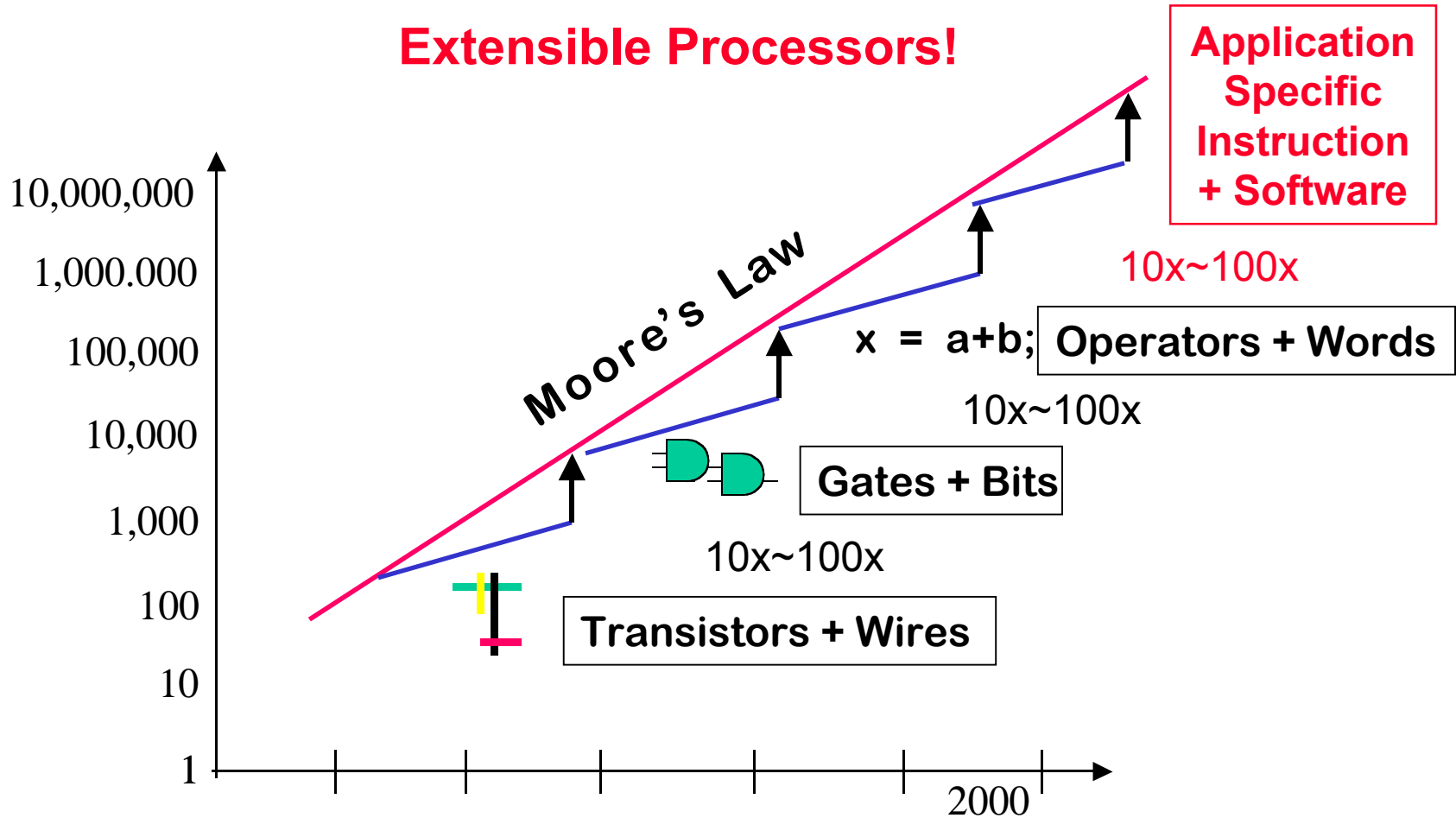
Logic Transistor / Chip (K)

Transistor/Staff-month

Source: NTRS'97

A Brief Tour of History

Extensible Processors!

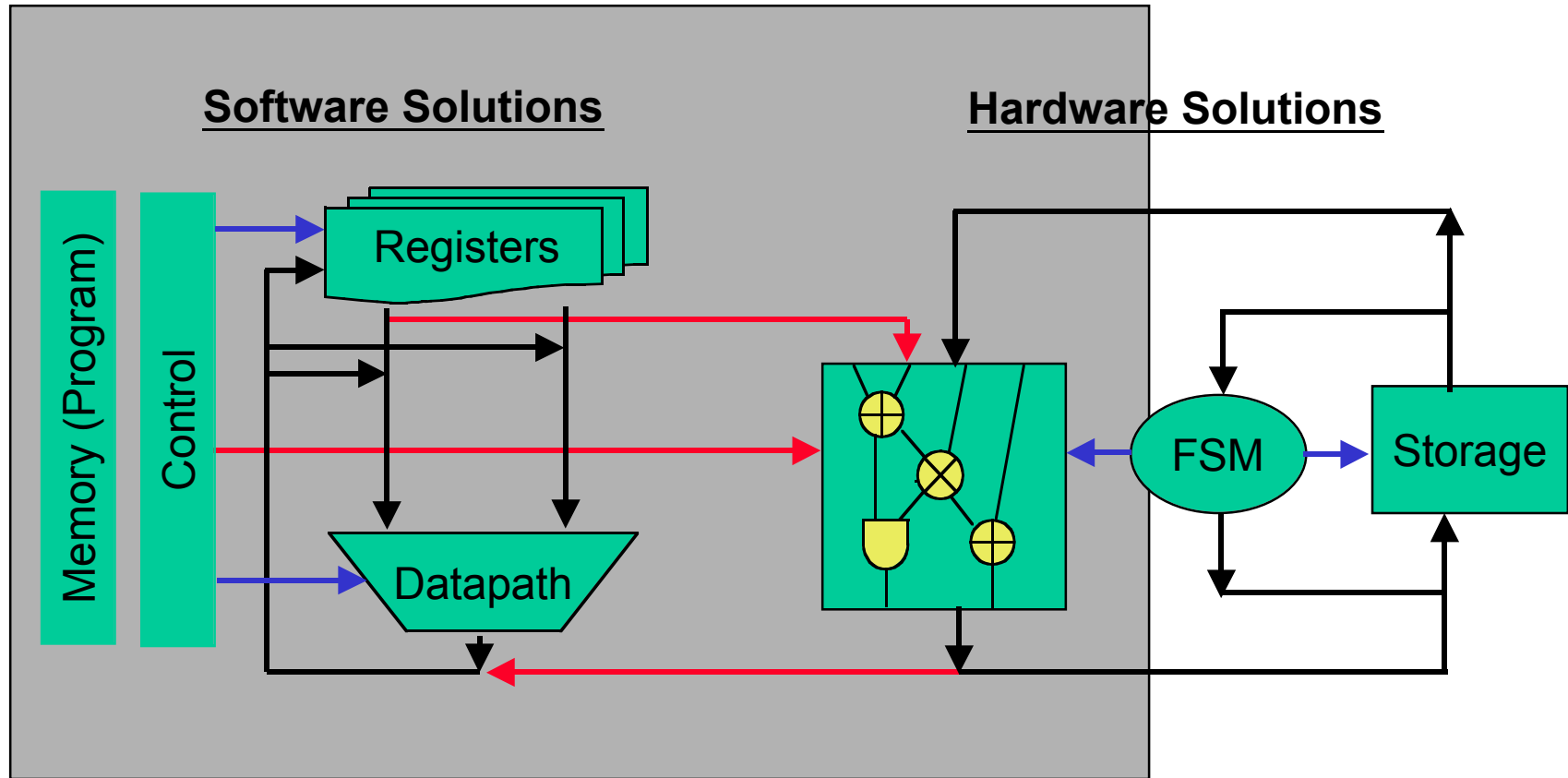


Logic Transistor / Chip (K)

Transistor/Staff-month

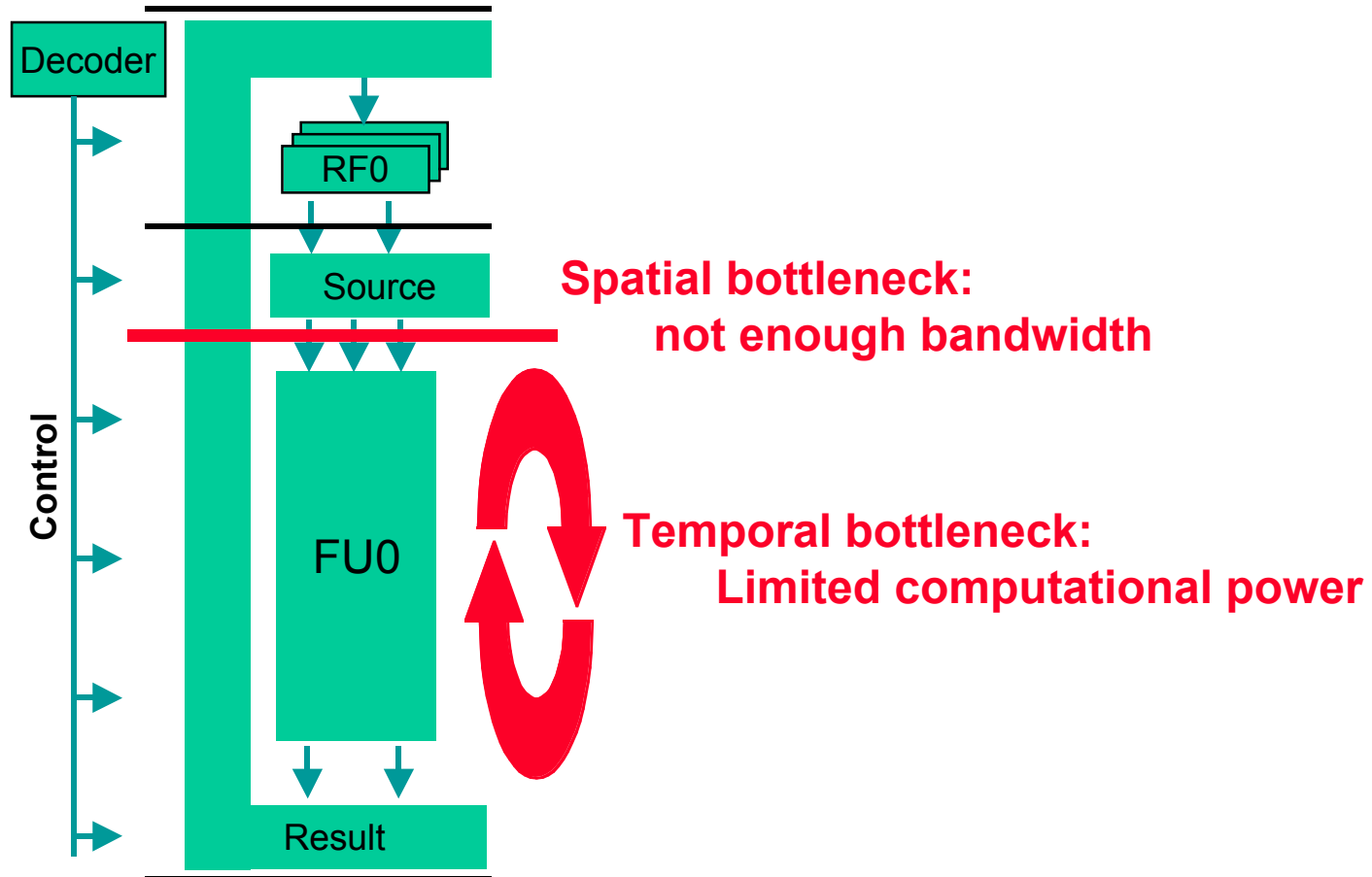
Why Extensible Processors?

Extensible Processor



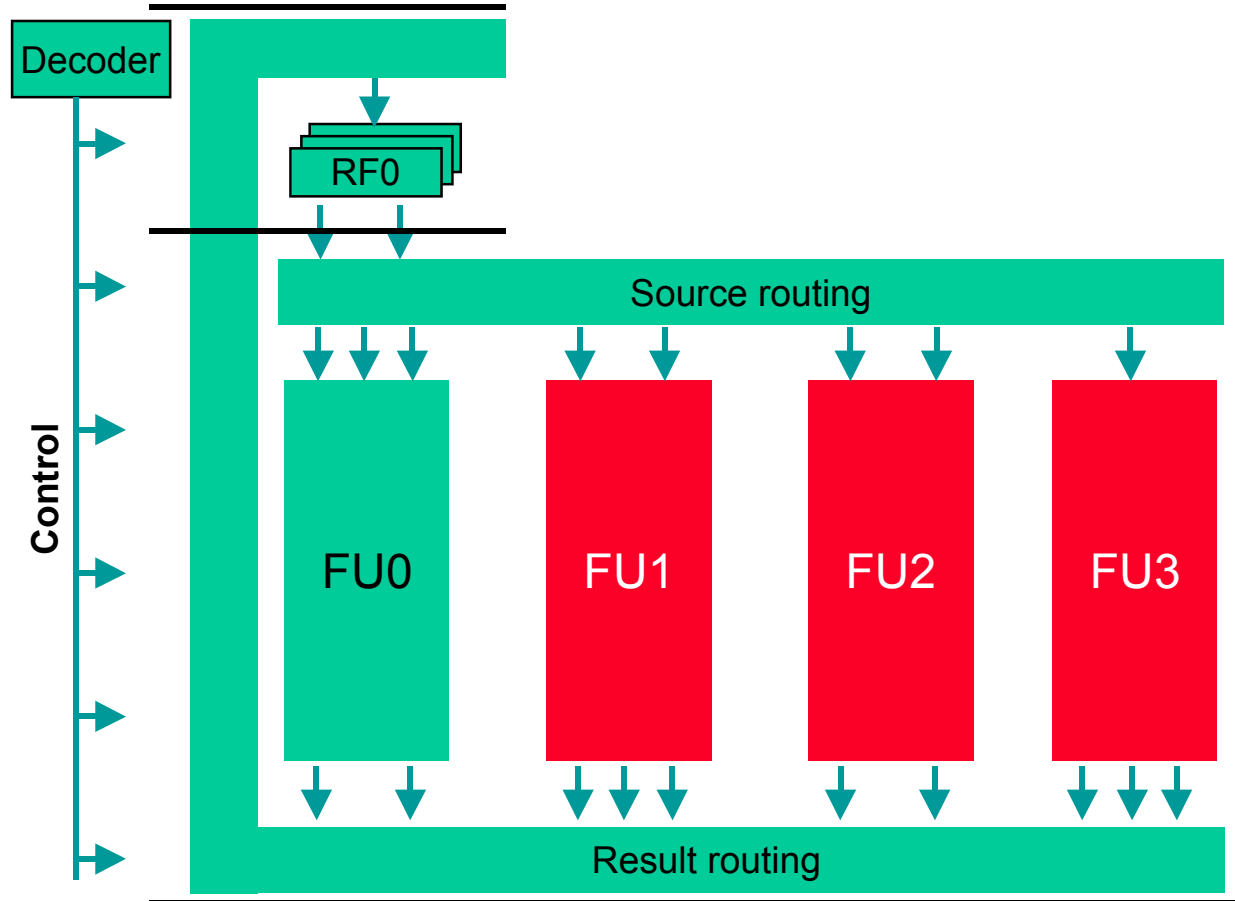
	Correct	Efficient
Software	easier	harder
Hardware	harder	easier

- ❖ **Configurable / Extensible processor solution**
 - **Xtensa architecture**
 - **Instruction extension automation**
 - **A detailed example**
- ❖ **EEMBC benchmarks – a case study**
- ❖ **Summary**





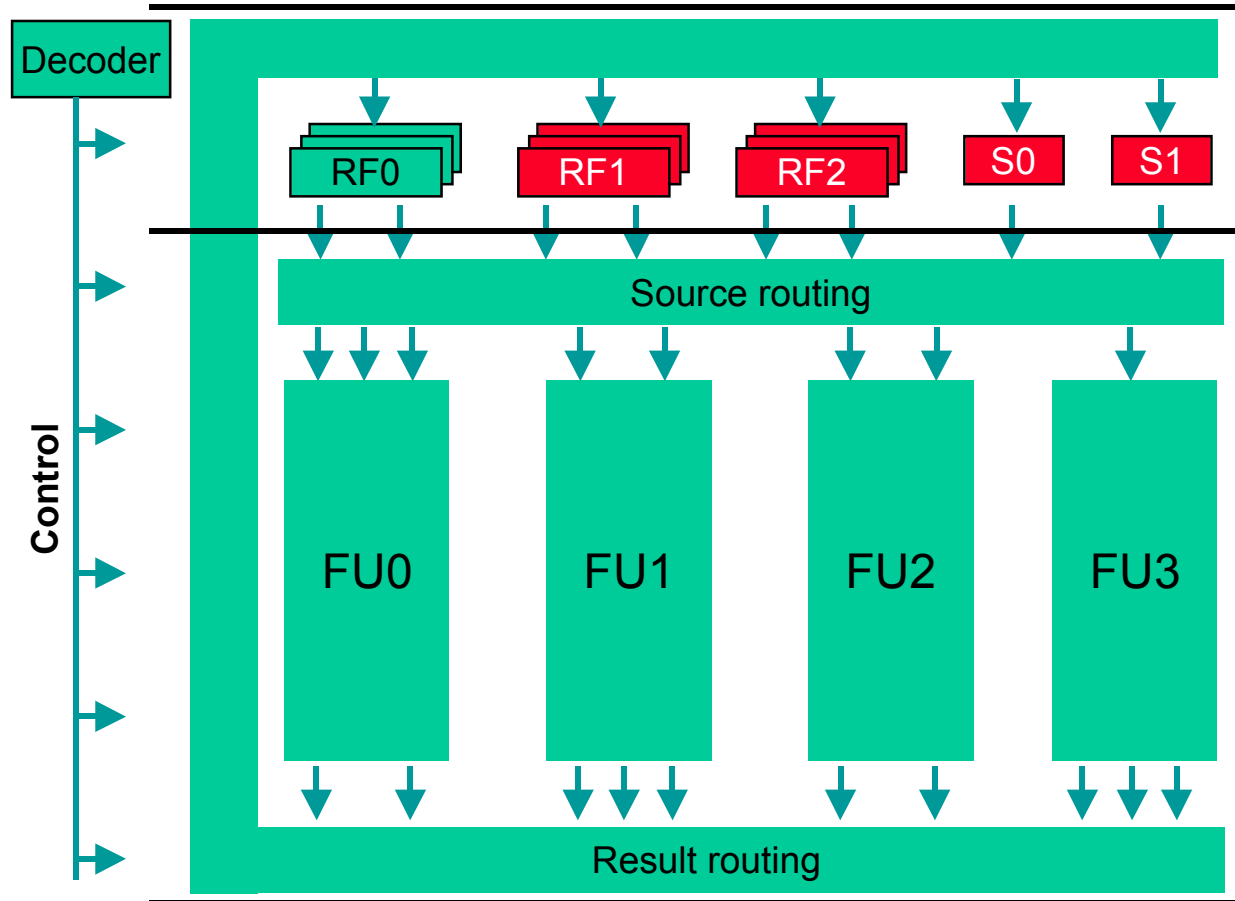
Fixed Architecture - cont.



•More FU's



Fixed Architecture – cont.

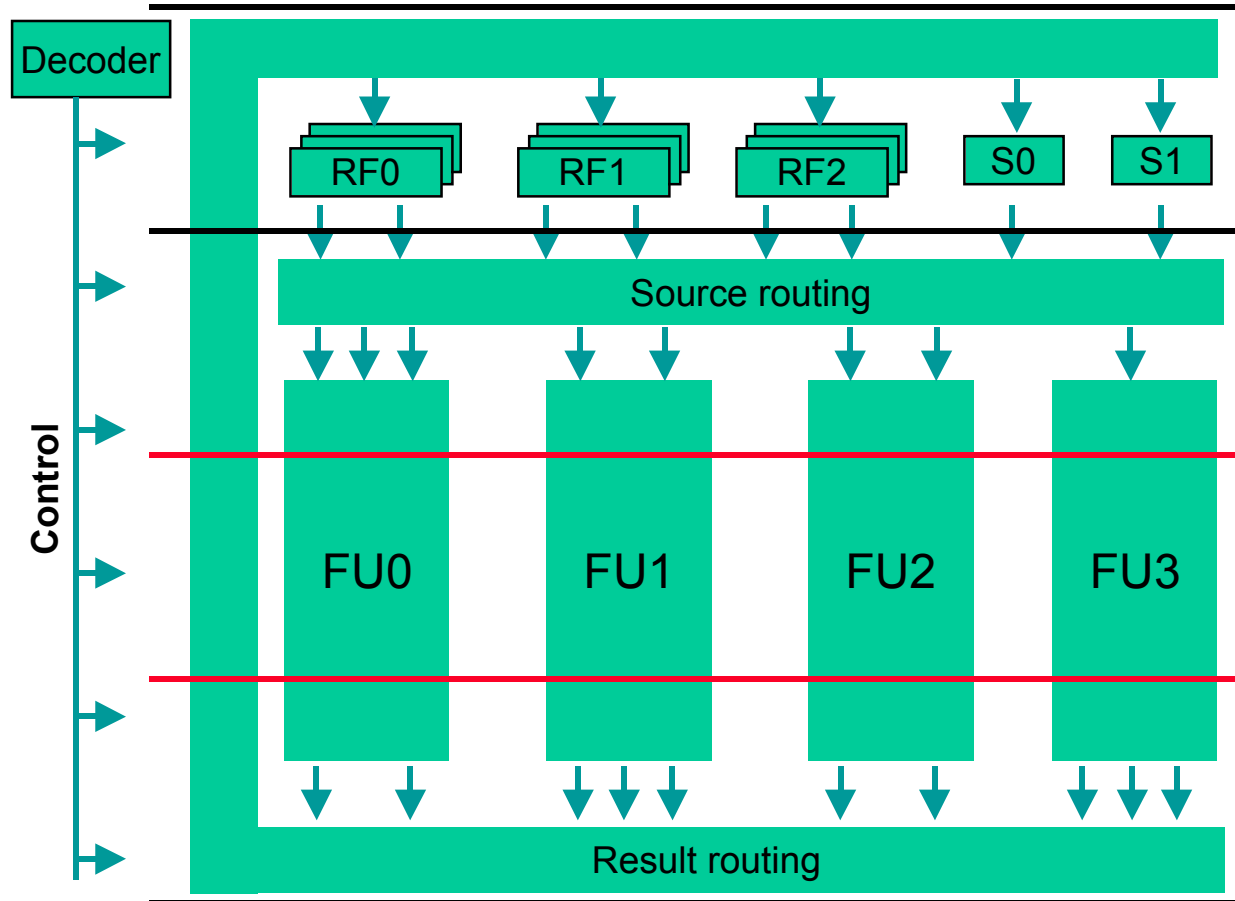


•More FU's

•More registers



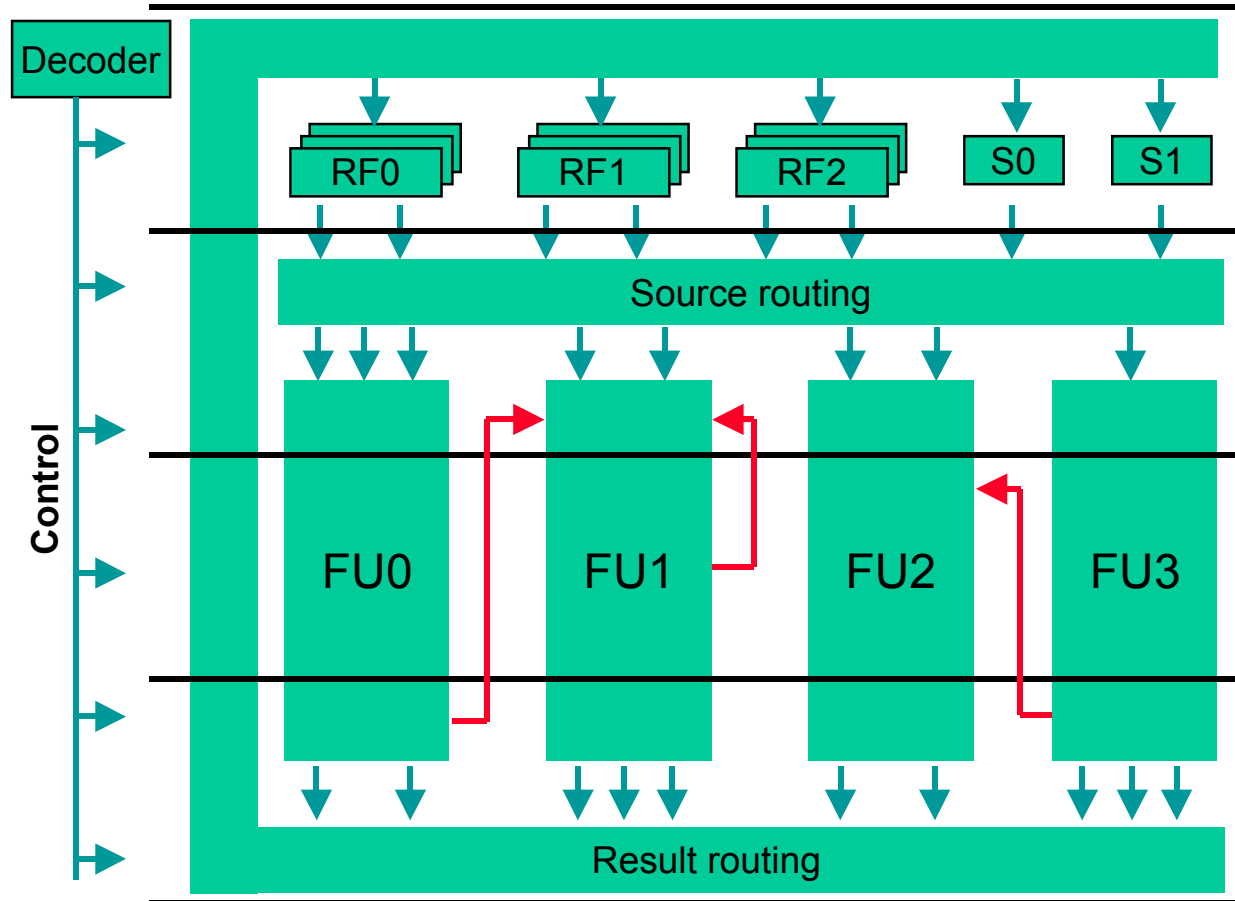
Fixed Architecture – cont.



- More registers
- More FU's
- Deeper pipeline**



Fixed Architecture – cont.



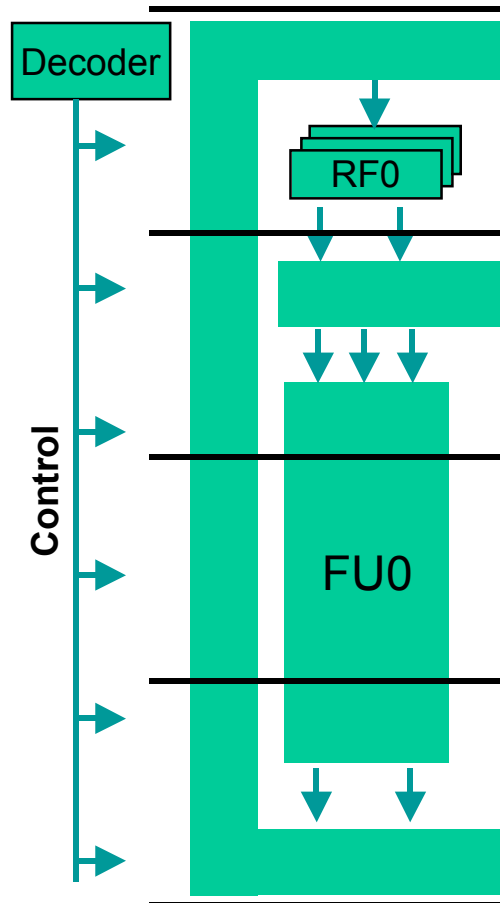
- More registers
- More FU's
- Deeper pipeline
- **Bypass/forward**

❖ Problem with fixed processor:

- Waste silicon
 - No universal extensions, or even one for each application class
- Waste power
- Not fast enough, compared with hardware implementation

❖ Solution:

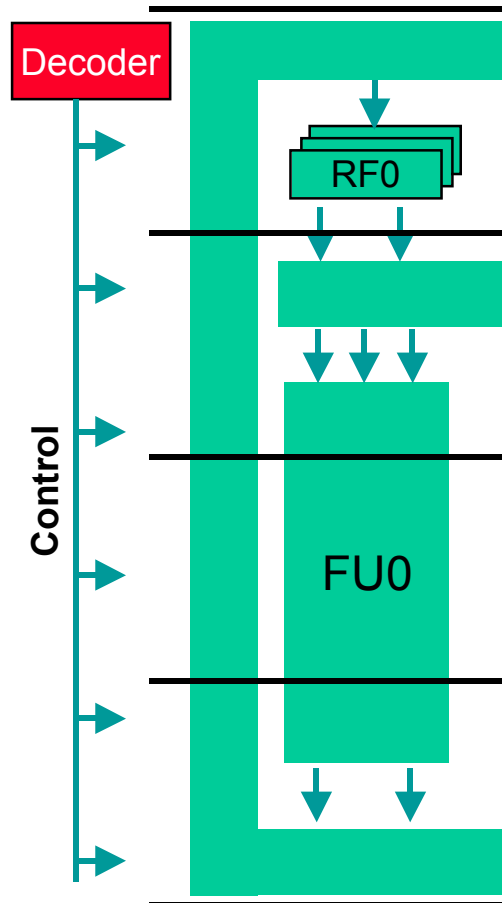
- Moving application-specific datapath into the processor
- Replacing FSM's with software programs



- ❖ Good performance
 - Comparable to any embedded 32-bit RISC
- ❖ Good code density
 - Much better than 32-bit RISC
 - Use 16b/24b instructions
- ❖ Small
 - 0.7mm² in .18_μm
- ❖ Low power
 - 0.4mW / MHz
- ❖ Extensible
 - Allow application-specific extensions
- ❖ Extend with a language
 - Tensilica Instruction Extension (TIE) language
- ❖ Complete HW/SW support for the extension
 - TIE compiler



TIE language - opcode

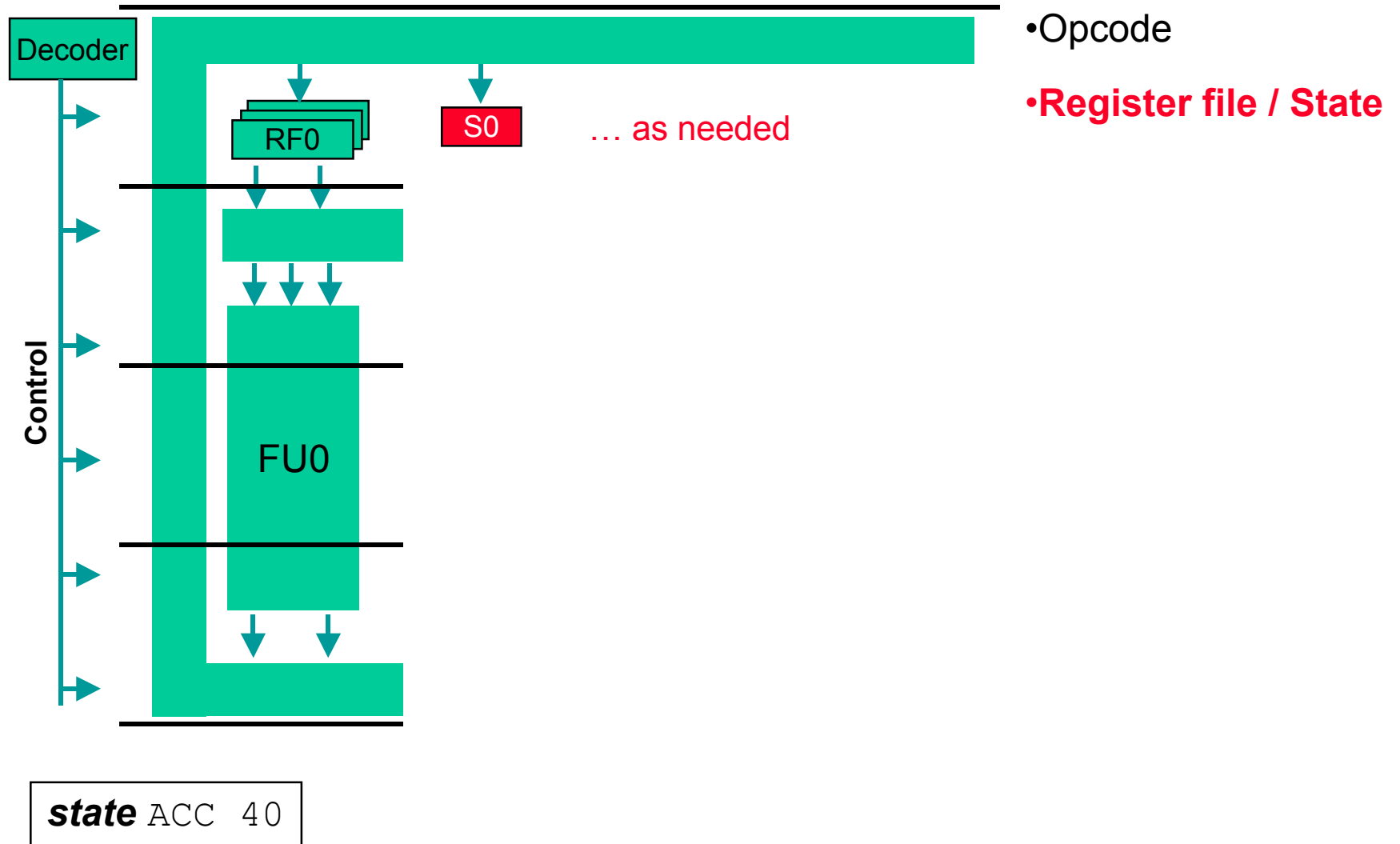


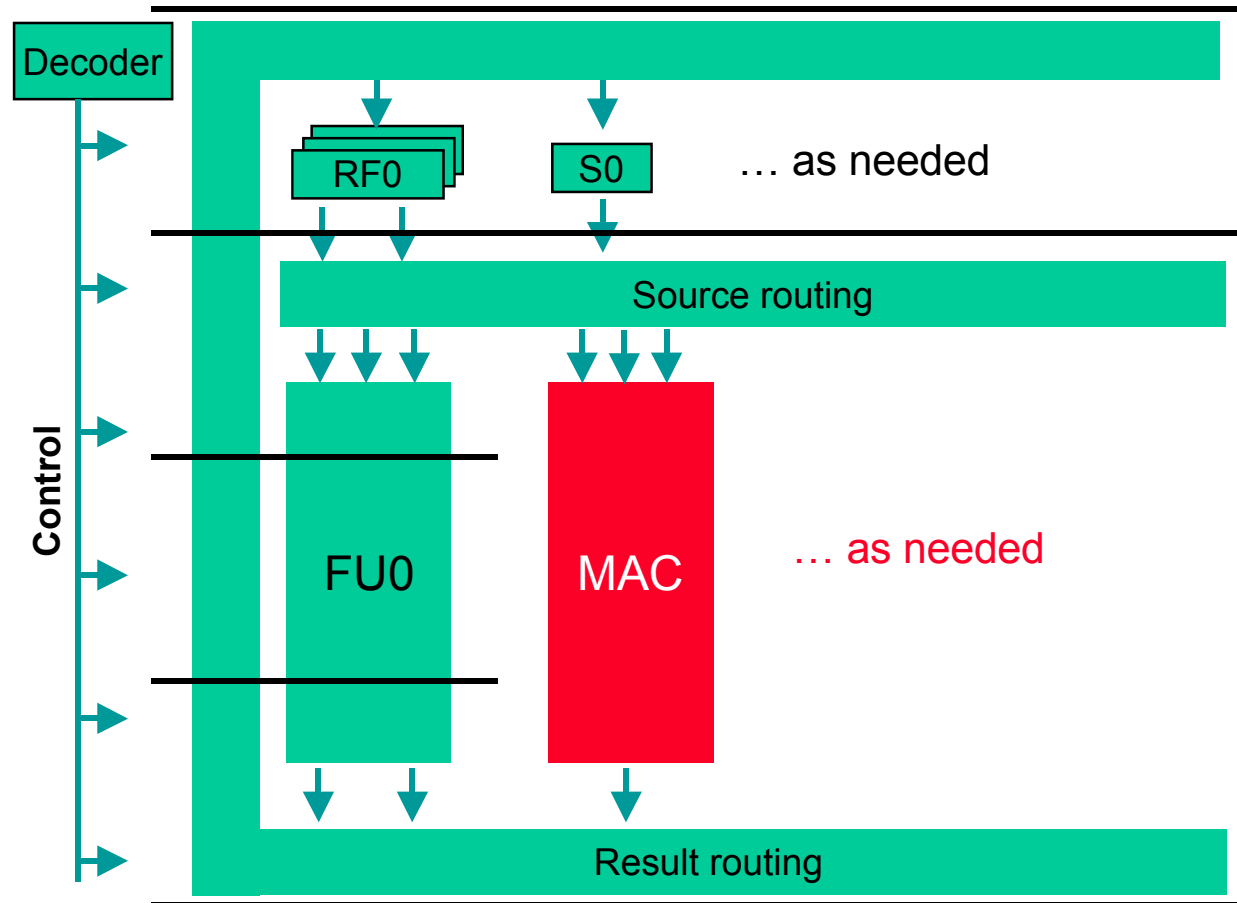
•Opcode

```
opcode MAC op0=4'b1101 CUST0
```



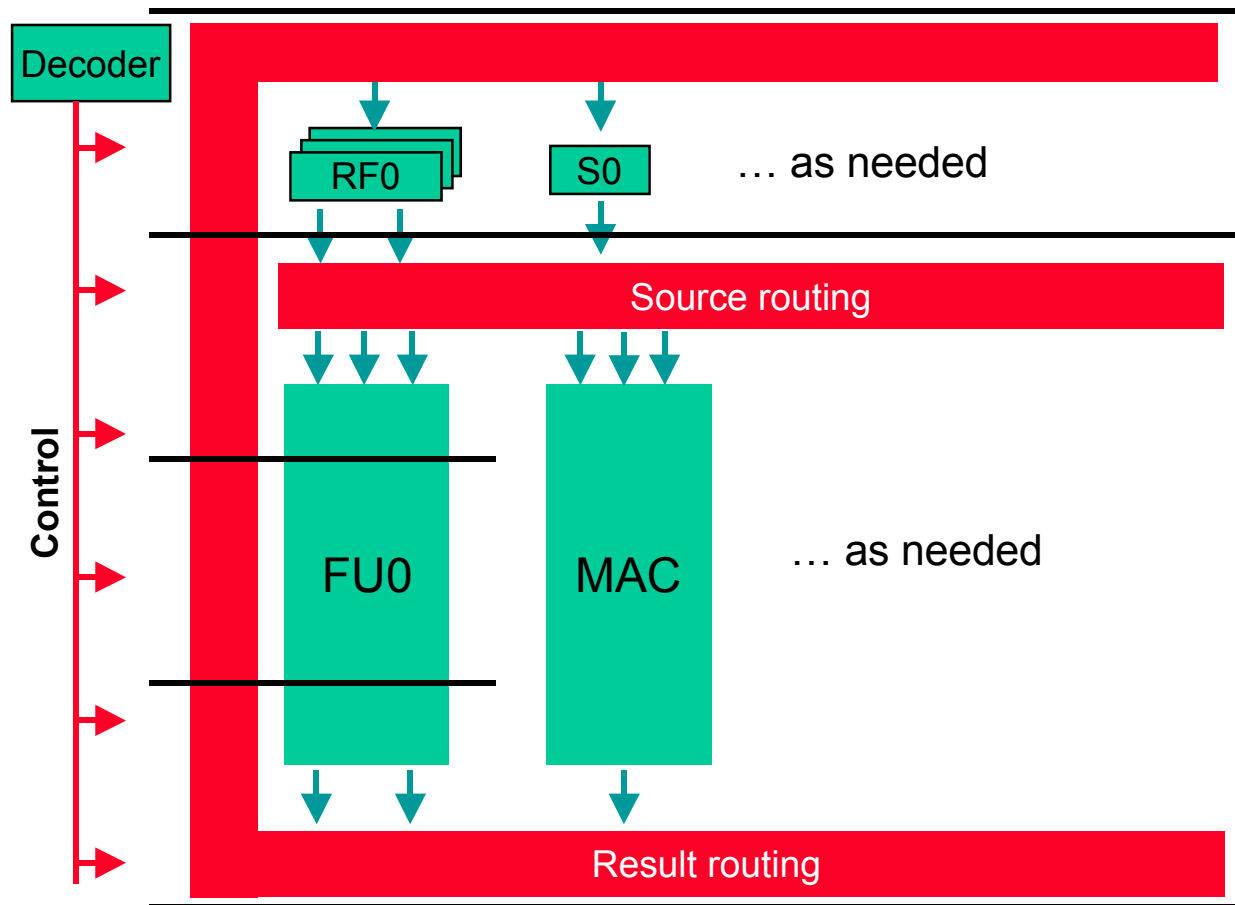
TIE Language – *regfile / state*





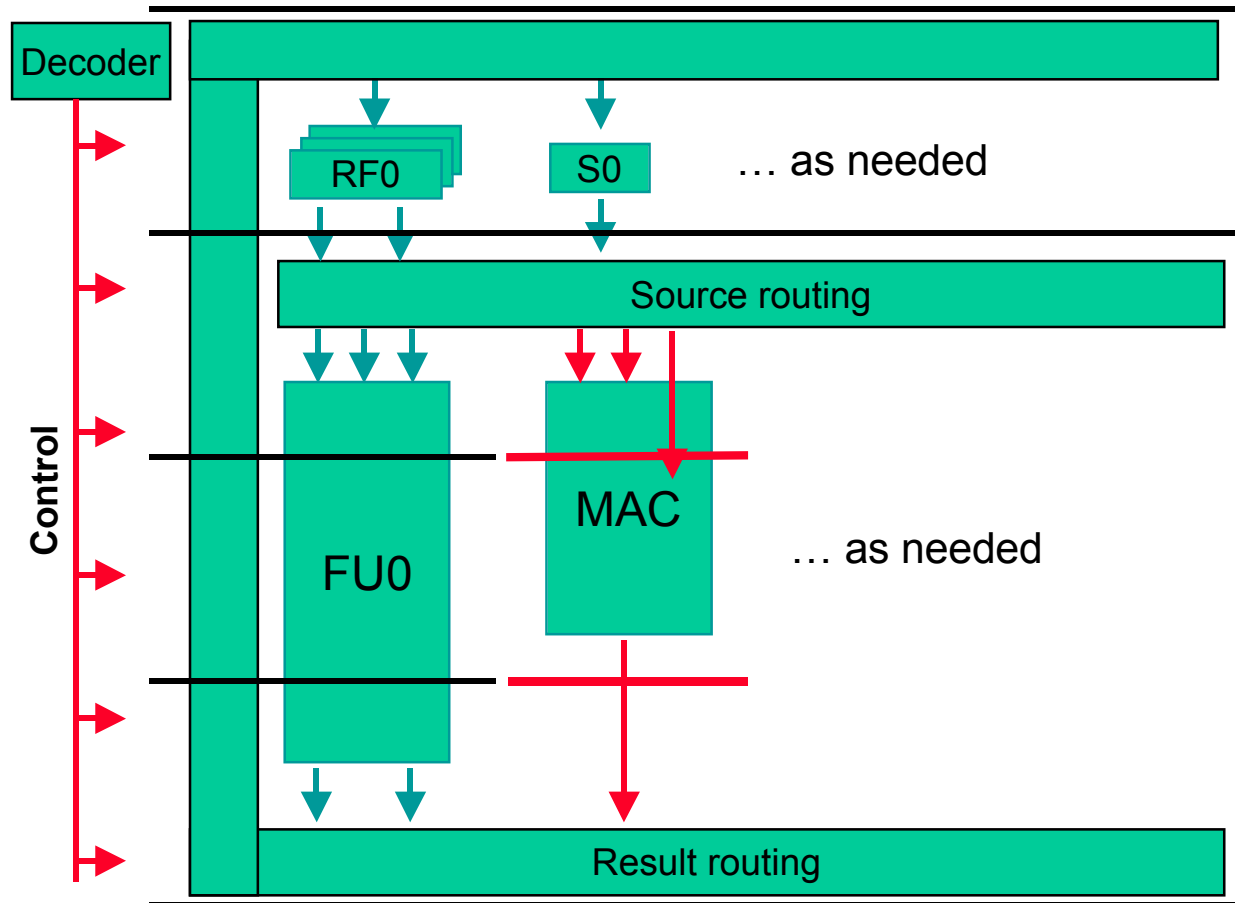
- Opcode
- Register file / state
- **semantics**

```
semantic sem1 {MAC} {assign ACC=ACC+ars[15:0]*art[15:0];}
```



- Opcode
- Register file / state
- semantics
- **Instruction class**

```
iclass c1 {MAC} {in ars, in art} {inout ACC}
```



- Opcode
- Register file / state
- semantics
- **Instruction class**
- **schedule**

```
schedule s1 {MAC} {use ars 1; use art 1; use ACC 2; def ACC 2;}
```

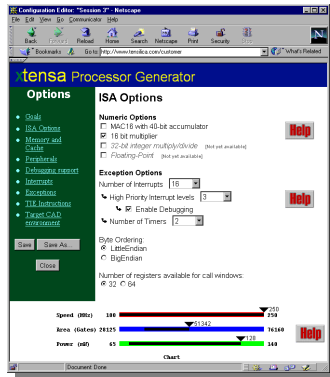


A Complete Example – Parallel MAC

```
opcode PMAC op2=0 CUST0
state ACC1 40
state ACC2 40
iclass rr {PMAC}{in ars, in art}{inout ACC1, inout ACC2}
semantic pmac_sem {PMAC} {
    assign ACC1 = ACC1 + ars[15:0] * art[15:0];
    assign ACC2 = ACC2 + ars[31:16] * art[31:16];
}
schedule pmac_schd {PMAC} {
    use ars 1; use art 1;
    use ACC1 2; use ACC2 2;
    def ACC1 2; def ACC2 2;
}
```



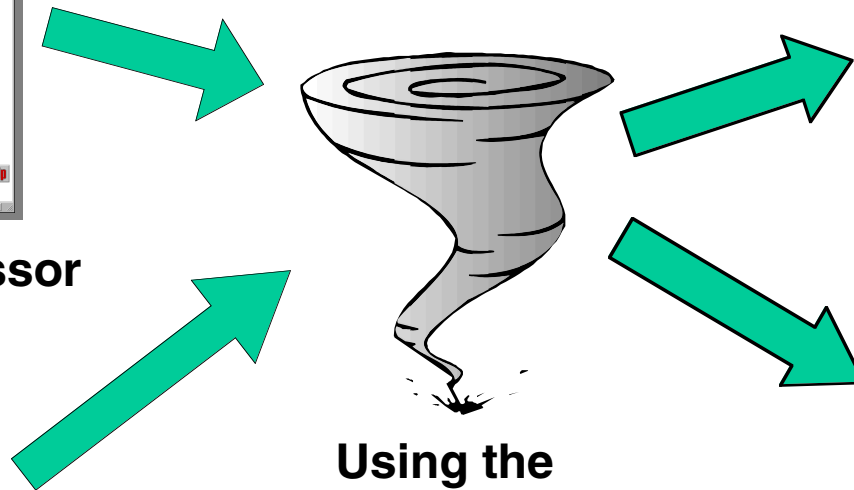
Xtensa Processor Generator



Select processor options

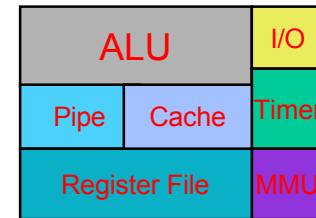


Describe new instructions

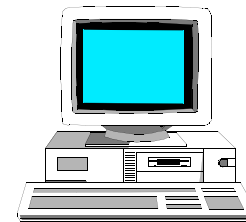


Using the Xtensa processor generator, create...

~ 1 Hour



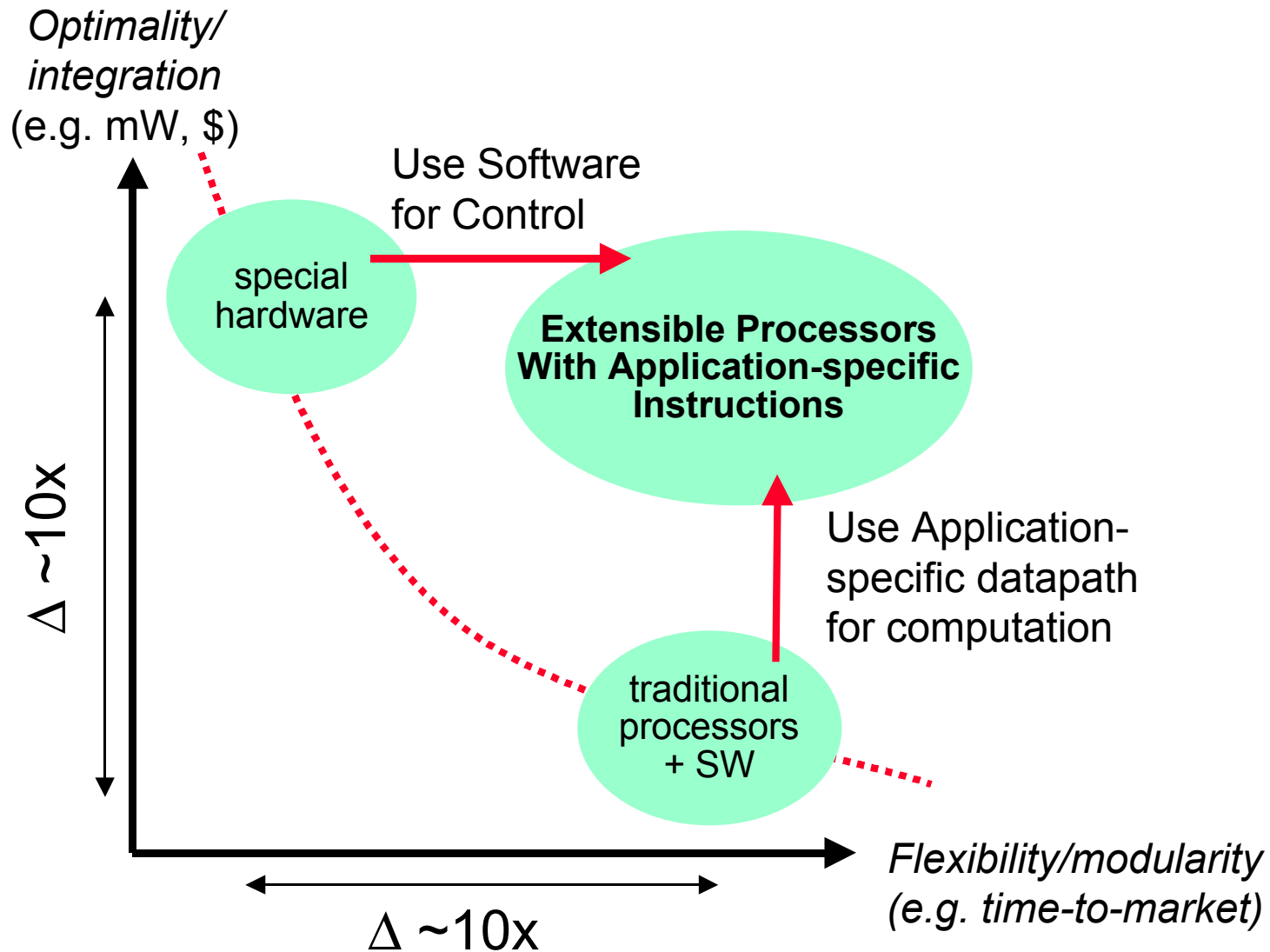
Tailored, synthesizable HDL uP core



Customized Compiler, Assembler, Linker, Debugger, Simulator

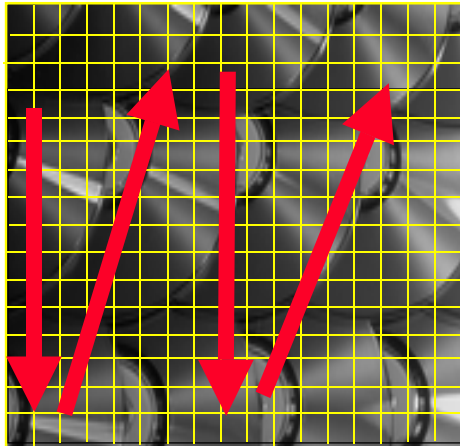


The Benefit of Extensible Processors



- ❖ **Configurable / Extensible processor solution**
 - Xtensa architecture
 - Instruction extension automation
 - **A detailed example**
- ❖ **EEMBC benchmarks – a case study**
- ❖ **Summary**

High Pass Filter: Overview



- ❖ 8 bit grayscale image
- ❖ Variable image size
- ❖ Each filtered pixel is computed from neighboring pixels

P_{11}	P_{12}	P_{13}
P_{21}	P_{22}	P_{23}
P_{31}	P_{32}	P_{33}

- ❖ Pixels are multiplied by constants and added

$$\begin{aligned}
 P'_{22} = & F_{11} * P_{11} + F_{12} * P_{12} + F_{13} * P_{13} + \\
 & F_{21} * P_{21} + F_{22} * P_{22} + F_{23} * P_{23} + \\
 & F_{31} * P_{31} + F_{32} * P_{32} + F_{33} * P_{33}
 \end{aligned}$$

- ❖ Scan down and across the entire image filtering each pixel according to neighboring pixel values



High Pass Filter: Software Implementation

```
for (w = 1; w < (Width - 1); w++) {
    for (h = 1; h < (Height - 1); h++) {
        Center = (Width * h) + w;
        PelValue = (Short) (

            /* top row */
            (F11 * ImageInPtr[Center - Width - 1]) +
            (F21 * ImageInPtr[Center - Width]) +
            (F31 * ImageInPtr[Center - Width + 1]) +

            /* add middle row */
            (F12 * ImageInPtr[Center - 1]) +
            (F22 * ImageInPtr[Center]) +
            (F32 * ImageInPtr[Center + 1]) +

            /* add bottom row */
            (F13 * ImageInPtr[Center + Width - 1]) +
            (F23 * ImageInPtr[Center + Width]) +
            (F33 * ImageInPtr[Center + Width + 1]) );

        ImageOutPtr[Center] = (Byte) (PelValue >> 8);
    }
}
```

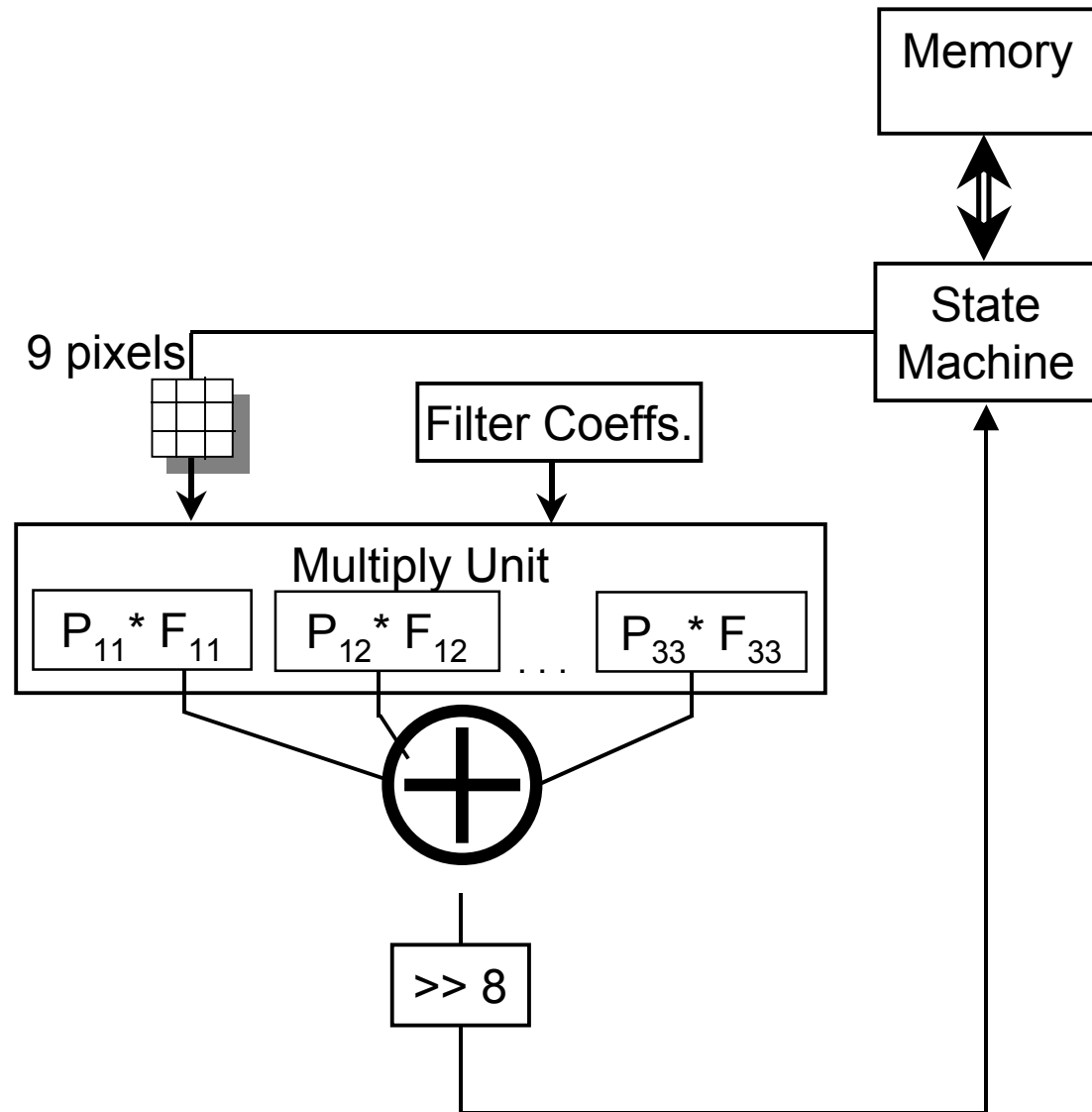


High Pass Filter: Improved Program

```
for (w = 1; w < (Width - 1); w++) {  
  
    LOADROW (ImageInPtr, w, 1);  
    LOADROW (ImageInPtr, w+1, 1);  
  
    for (h = 1; h < (Height - 1); h++) {  
  
        LOADROW (ImageInPtr, w, h);  
        FILTER ( );  
        STOREPIXEL ( & ImageOutPtr[Center] );  
  
    }  
  
}
```



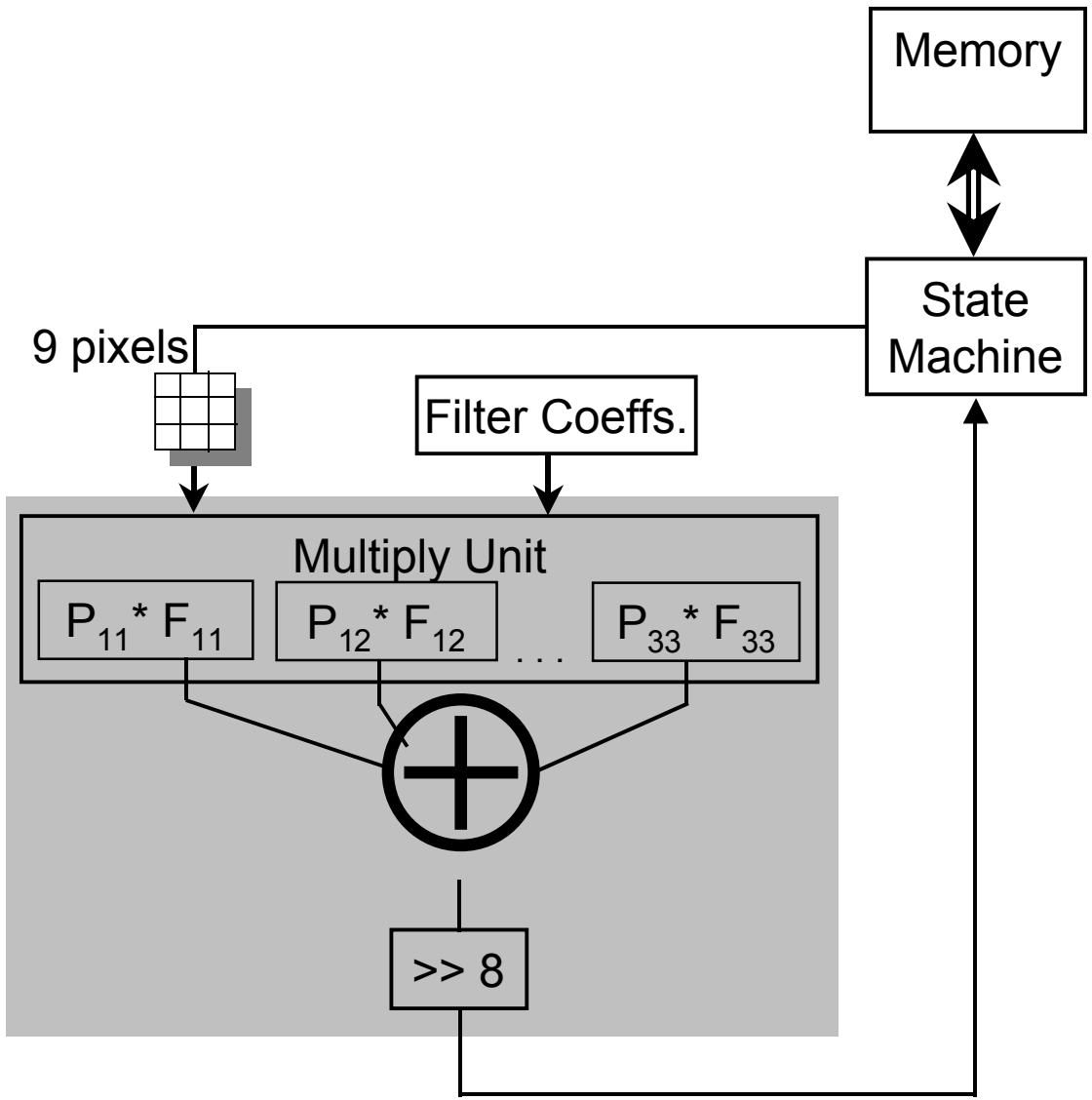
High Pass Filter: Hardware Implementation





High Pass Filter: FILTER Instruction

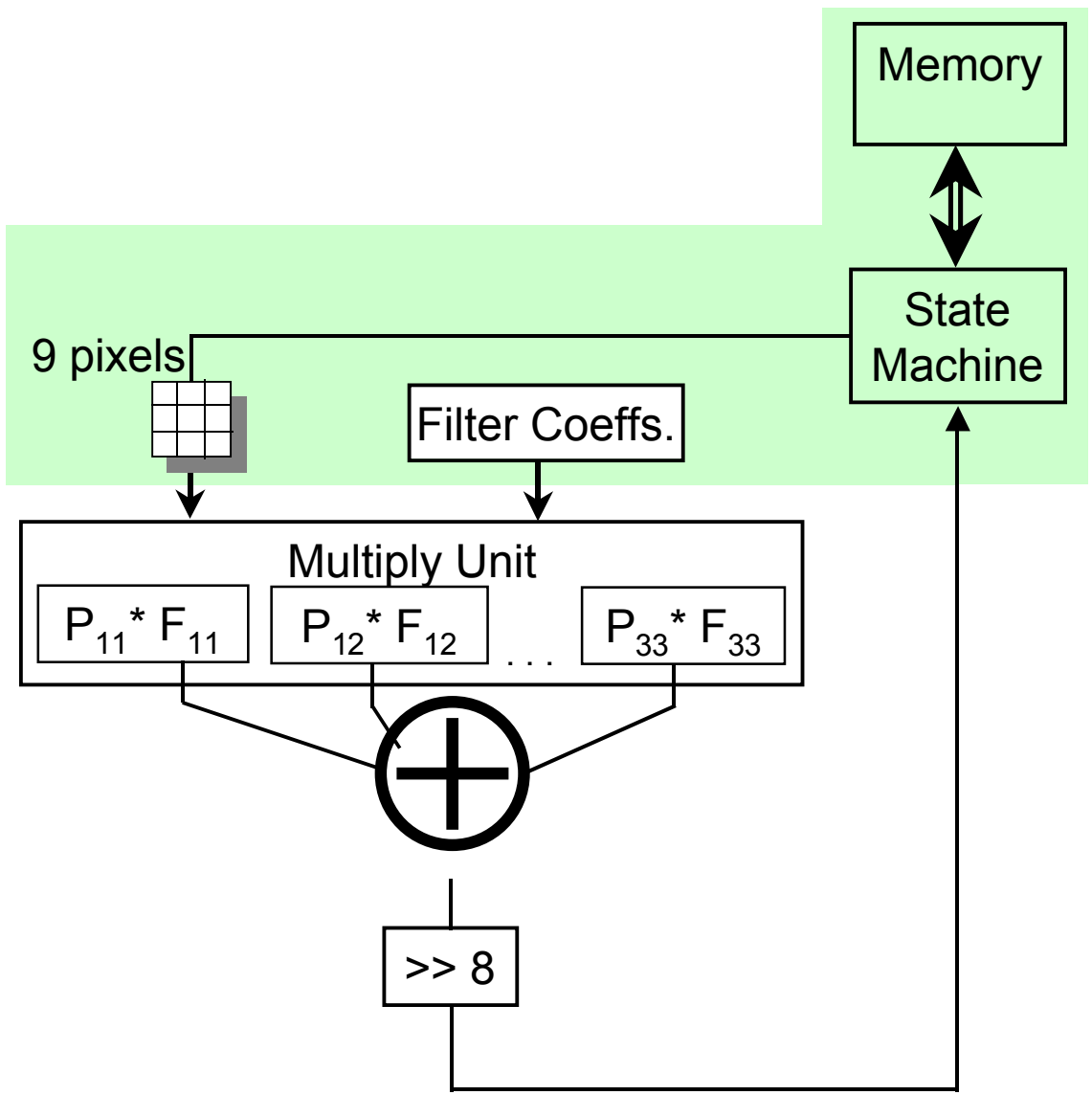
FILTER





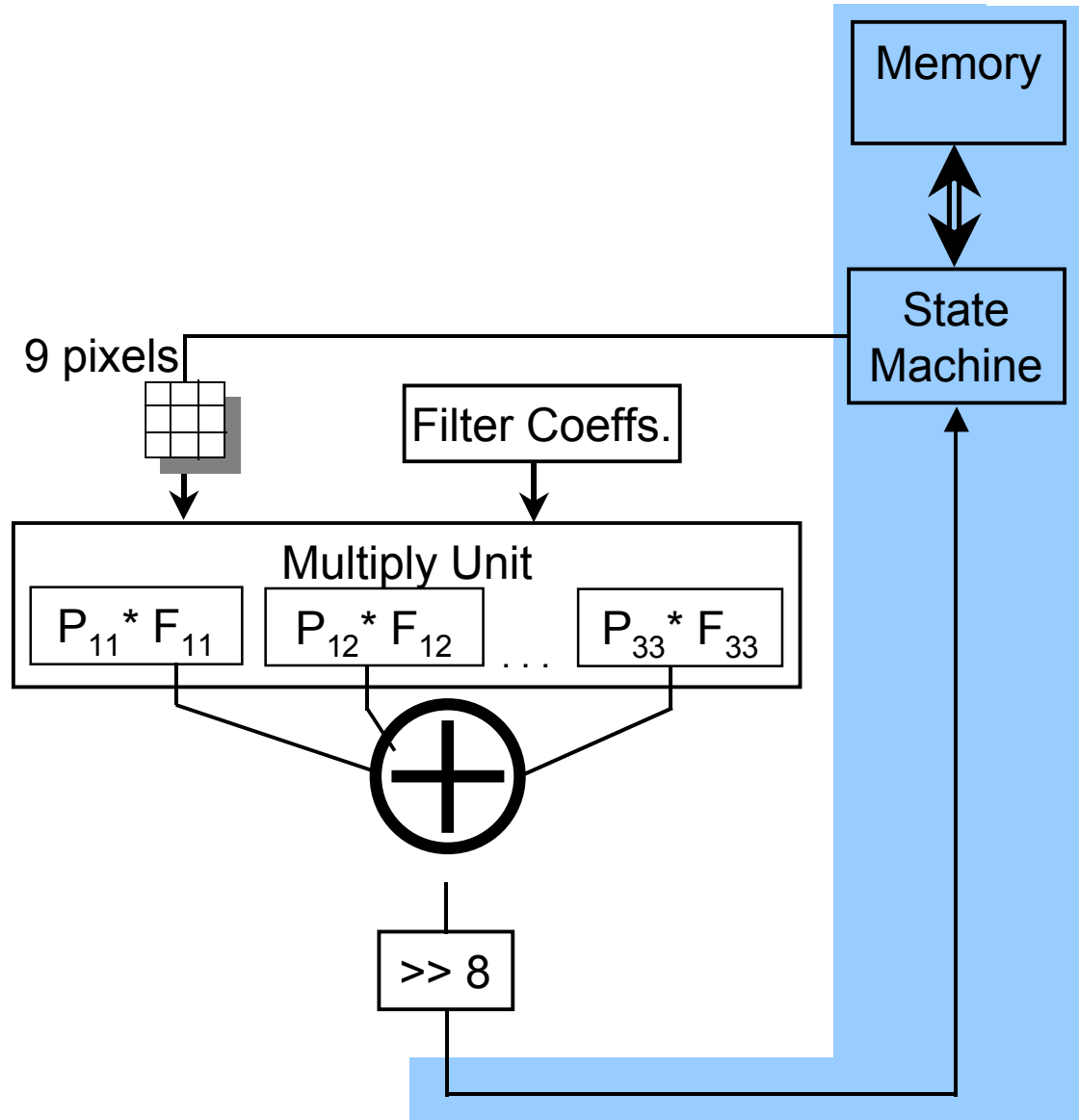
High Pass Filter: LOADROW Instruction

LOADROW arr, ars, art





High Pass Filter: Putting It All Together



STOREPIXEL ars

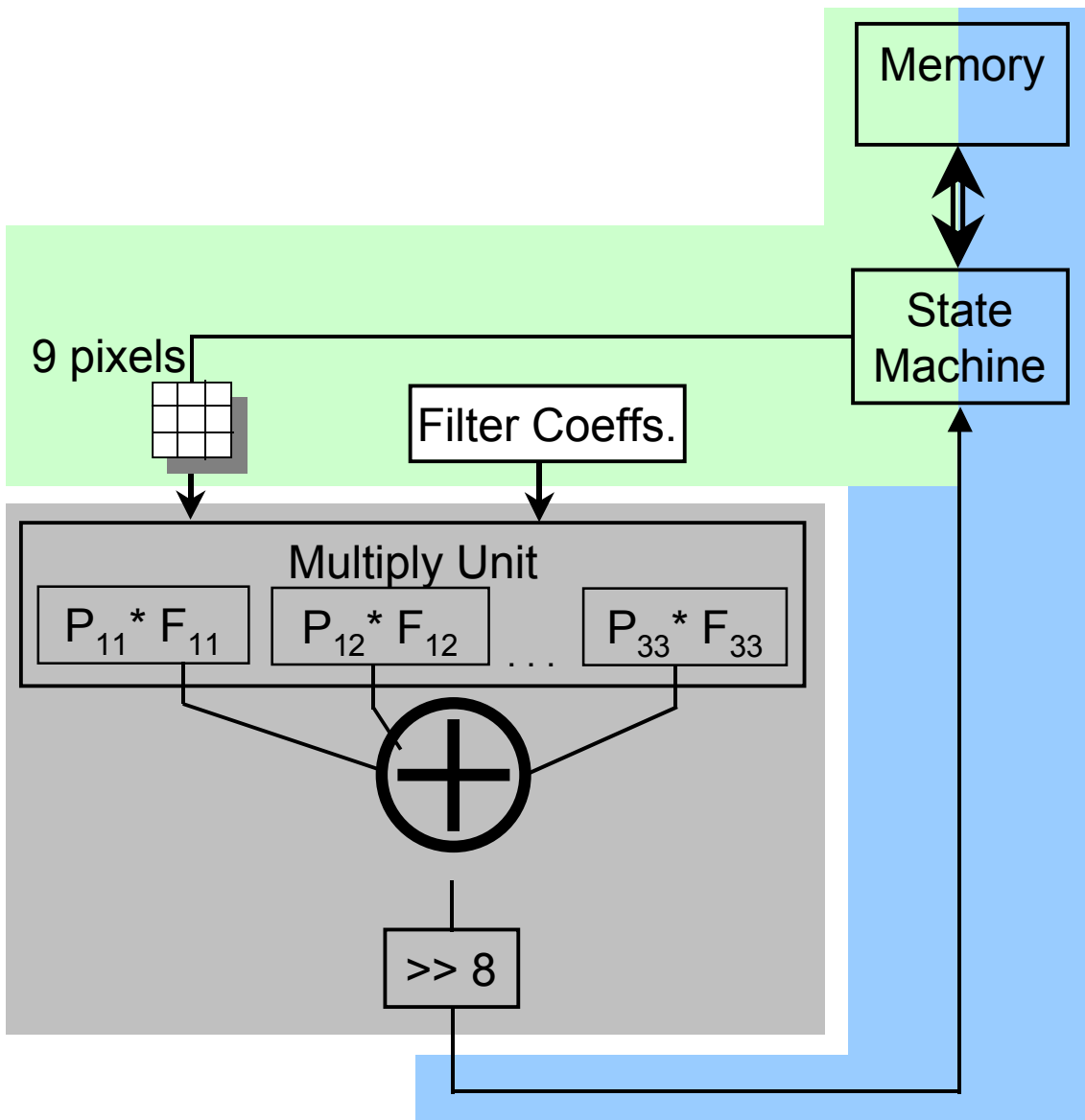


High Pass Filter: Putting It All Together

LOADROW arr, ars, art

FILTER

STOREPIXEL ars





High Pass Filter: Improved Program

```
for (w = 1; w < (Width - 1); w++) {  
  
    LOADROW (ImageInPtr, w, 1);  
    LOADROW (ImageInPtr, w+1, 1);  
  
    for (h = 1; h < (Height - 1); h++) {  
  
        LOADROW (ImageInPtr, w, h);  
        FILTER ( );  
        STOREPIXEL ( & ImageOutPtr[Center] );  
  
    }  
  
}
```

- ❖ **Configurable / Extensible processor solution**
 - Xtensa architecture
 - Instruction extension automation
 - A detailed example
- ❖ **EEMBC benchmarks – a case study**
- ❖ **Summary**



What is “EEMBC”?

- ❖ EDN EEmbedded Microprocessor Benchmark Consortium
- ❖ Pronounced “Embassy”
- ❖ Non-profit consortium, funded by over 40 members
 - Including: ARM, AMD, IBM, Intel, LSI Logic, MIPS, Motorola, National Semi, NEC, TI, Toshiba, Tensilica, and more
- ❖ Objective: Provide independently certified benchmark scores relevant to deeply embedded processor applications
 - Independent laboratory recreates and certifies all benchmark results - no tricks

- ❖ Five different benchmark suites
 - **Consumer**
 - **Networking**
 - **Telecom**
 - *Automotive*
 - *Office Automation*

- ❖ Each suite comprised of a range (five to sixteen) of benchmarks representative of that product category
 - Example: Consumer
 - Image compression, image filtering, color conversion



Two Metrics: Out-of-box vs. Optimized

❖ Out-of-Box

- Benchmark C code, no manual code optimization, no assembly coding

❖ Optimized, or “Full-Fury”

- Conventional Processors
 - Laboriously hand-tuned assembly code
 - Rewriting C code to fit the architecture for VLIW or SIMD machines
 - **Changing Code to Fit the Processor**
- **Xtensa**
 - Optimized processor using Xtensa processor generator and TIE Compiler
 - **Changing Processor to Fit the Application!!**



Xtensa Optimization Process

- ❖ Step #1: Configure processor via generator GUI
 - Compile C-code, evaluate results
 - Modify configuration as needed
 - “Out of Box” results measurement taken here

- ❖ Step #2: Profile Code, Add TIE

- ❖ Step #3: Optimize Code to Utilize TIE instructions
 - “Optimized” results measured on final hardware configuration

Same Path Used by Tensilica Customers!



Optimized Xtensa Configurations for EEMBC

OUT-OF-BOX

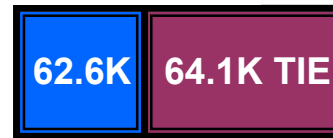
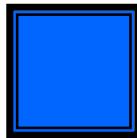
Configured Xtensa
(Using GUI Click box options)
Unmodified C-Code

OPTIMIZED

Configured Xtensa
Plus TIE Gates & Instructions
C-Code optimizations

Consumer Configuration

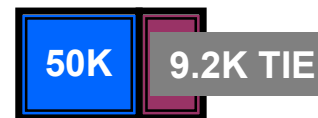
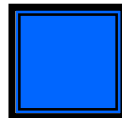
25000 base gates +
37600 config. gates
200MHz



127K total gates
200MHz

Network Configuration

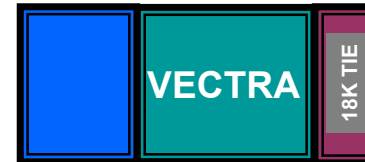
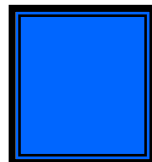
25000 base gates +
25000 config. gates
200MHz



59K total gates
200MHz

Telecom Configuration

25000 base gates +
37000 config Gates
200MHz



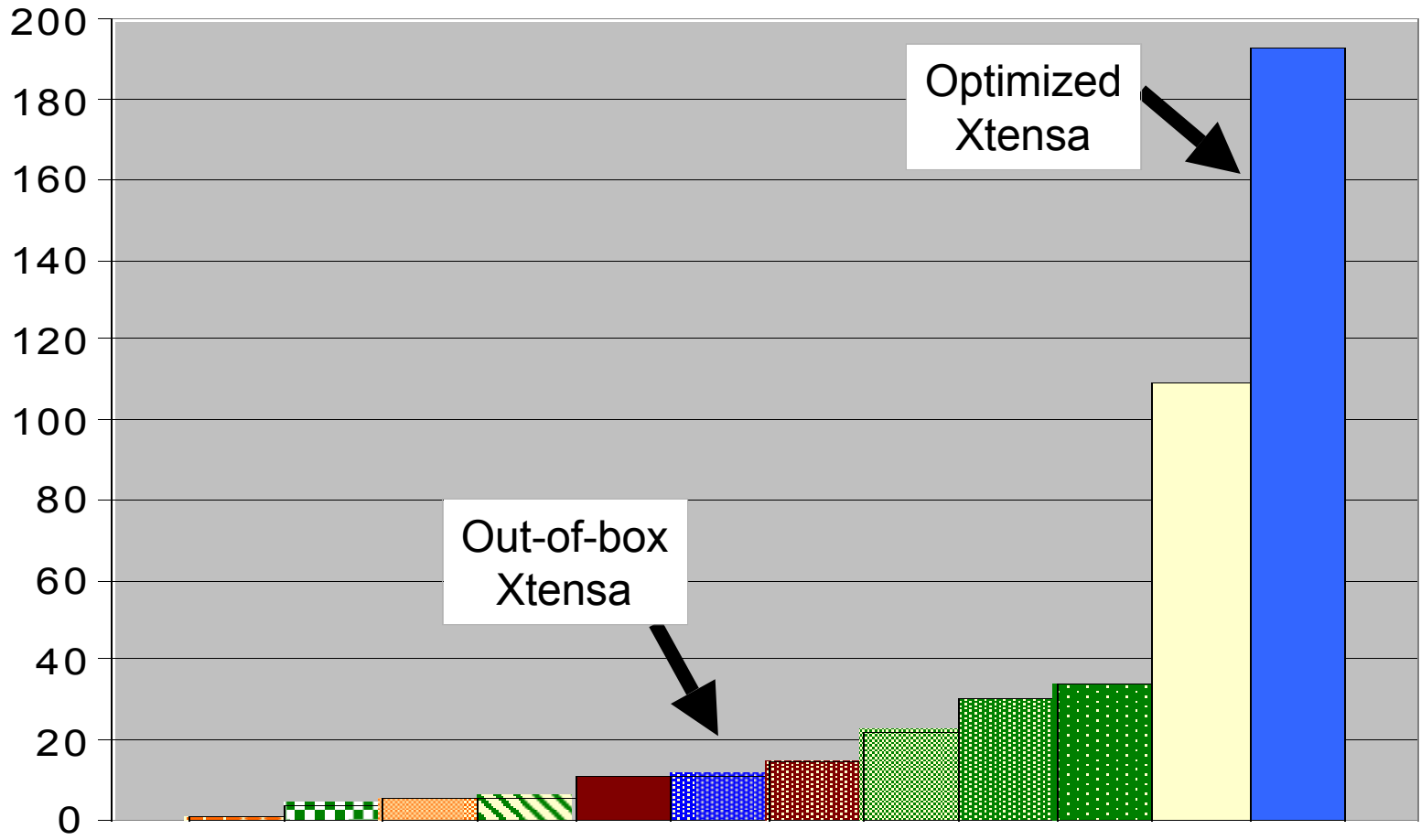
180K total
gates
200MHz

Illustrations conceptual, see EEMBC report for full details



EEMBC Consumer Benchmark

Consumermark

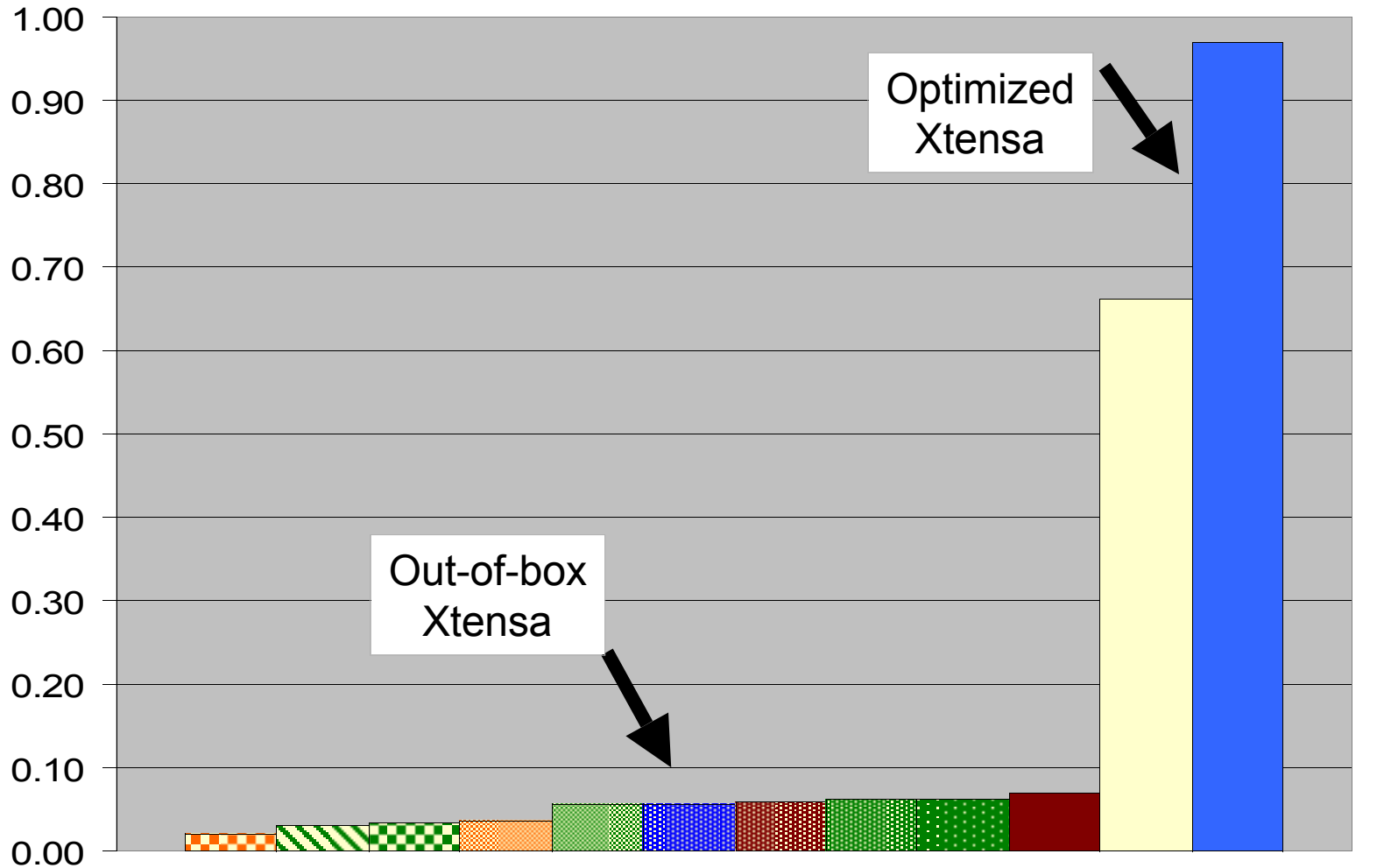


Processors



EEMBC Consumer Benchmark

Consumermark / MHz

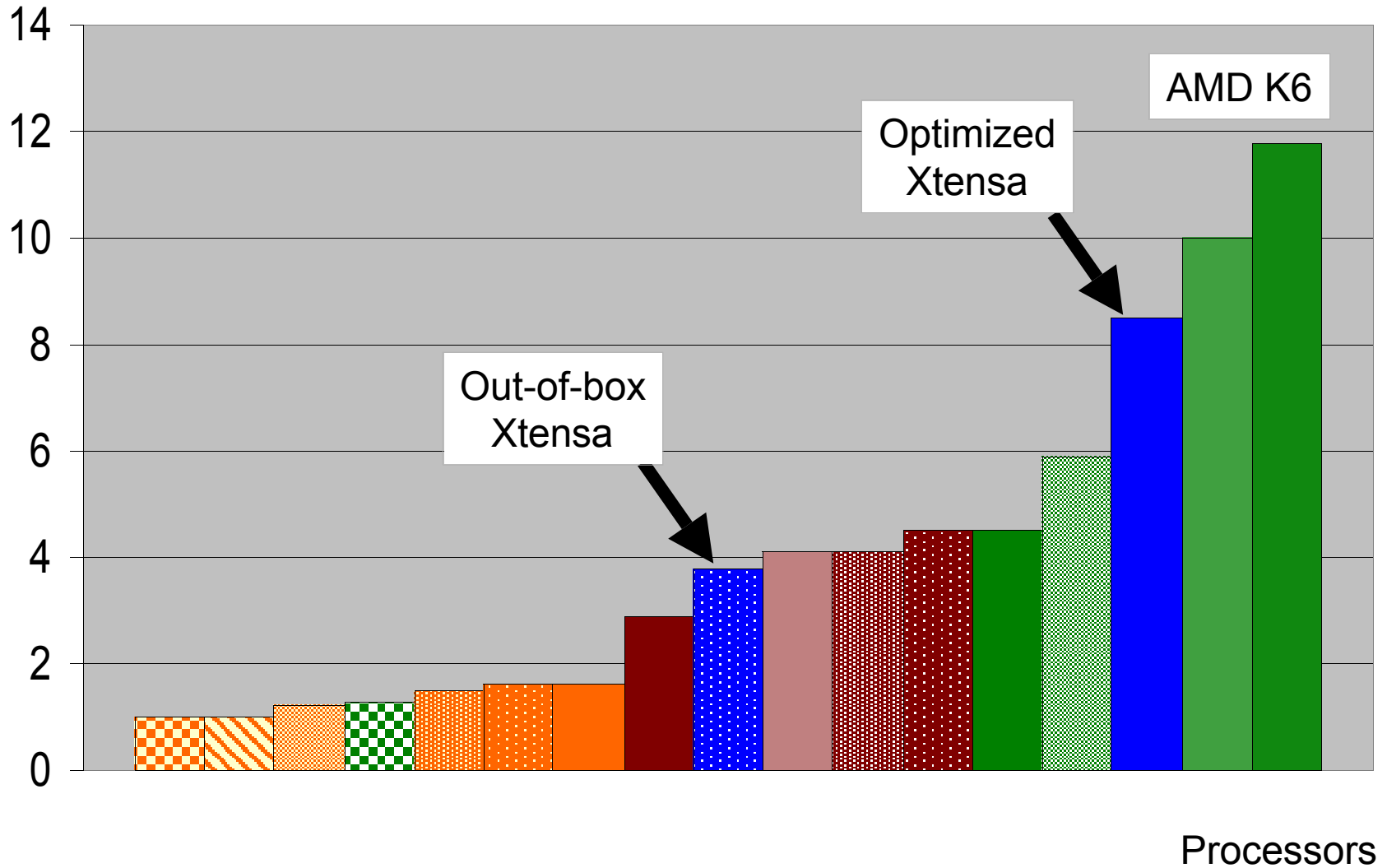


Processors



EEMBC Networking Benchmark

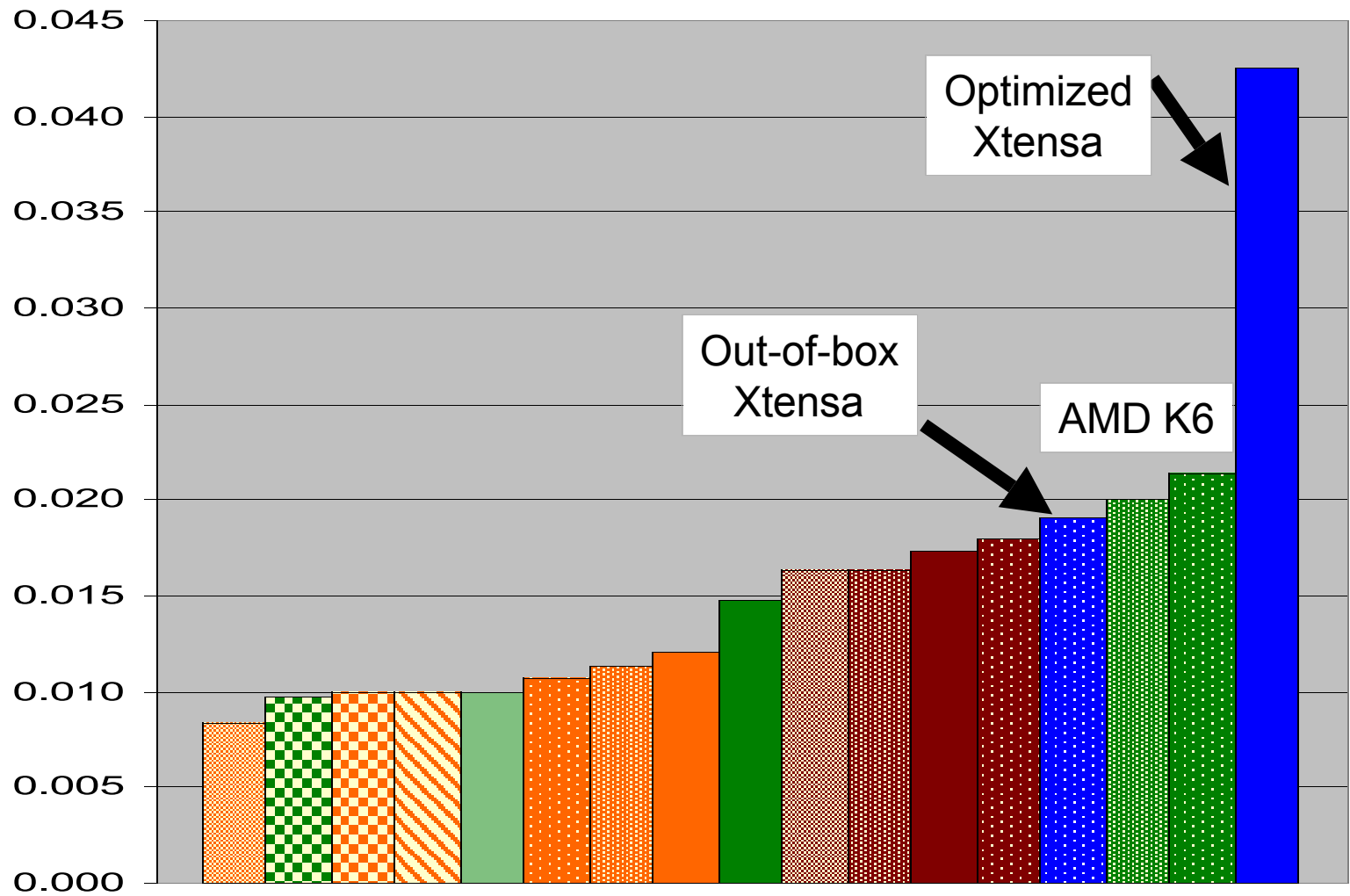
Netmark





EEMBC Networking Benchmark

Netmark / MHz

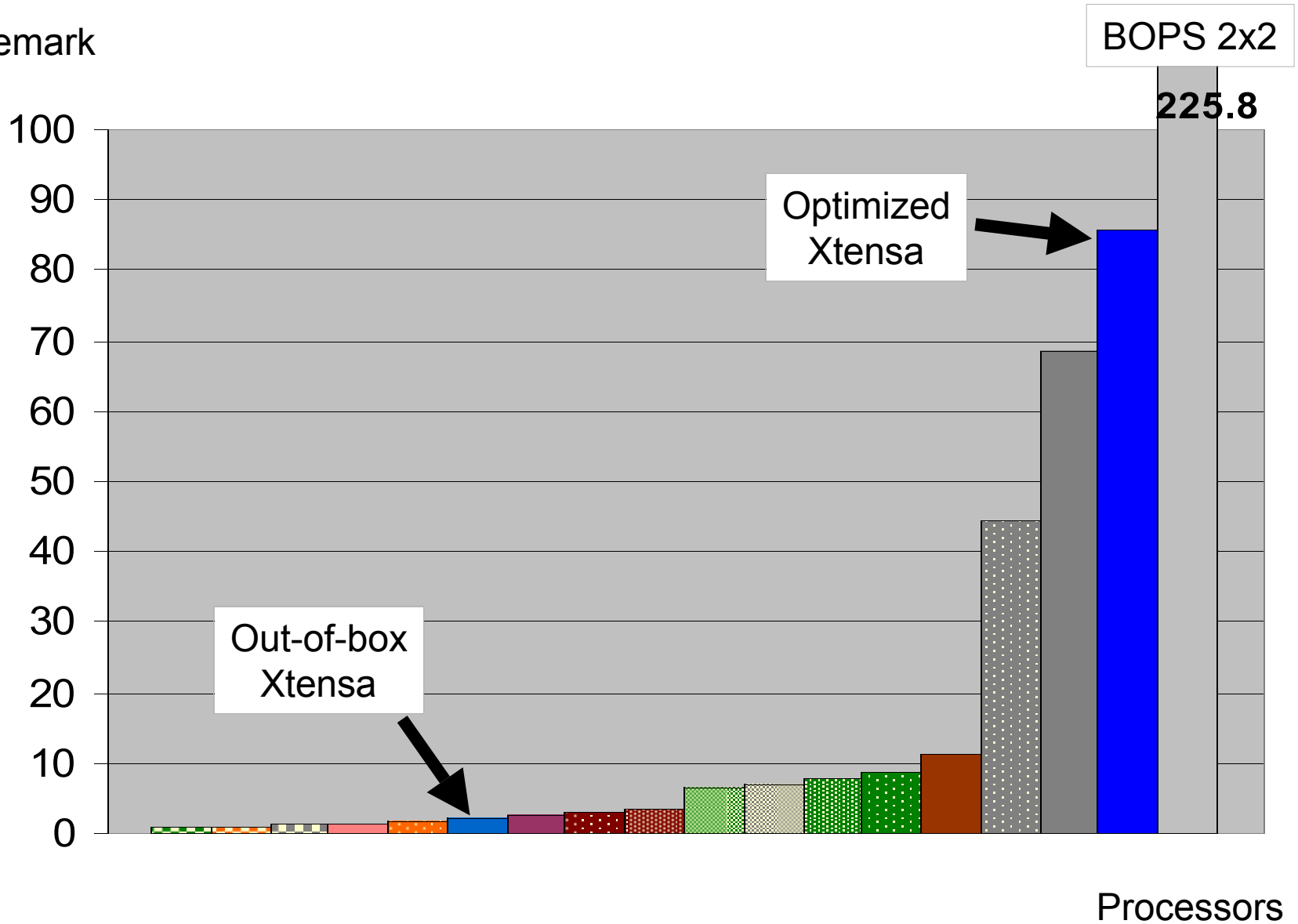


Processors

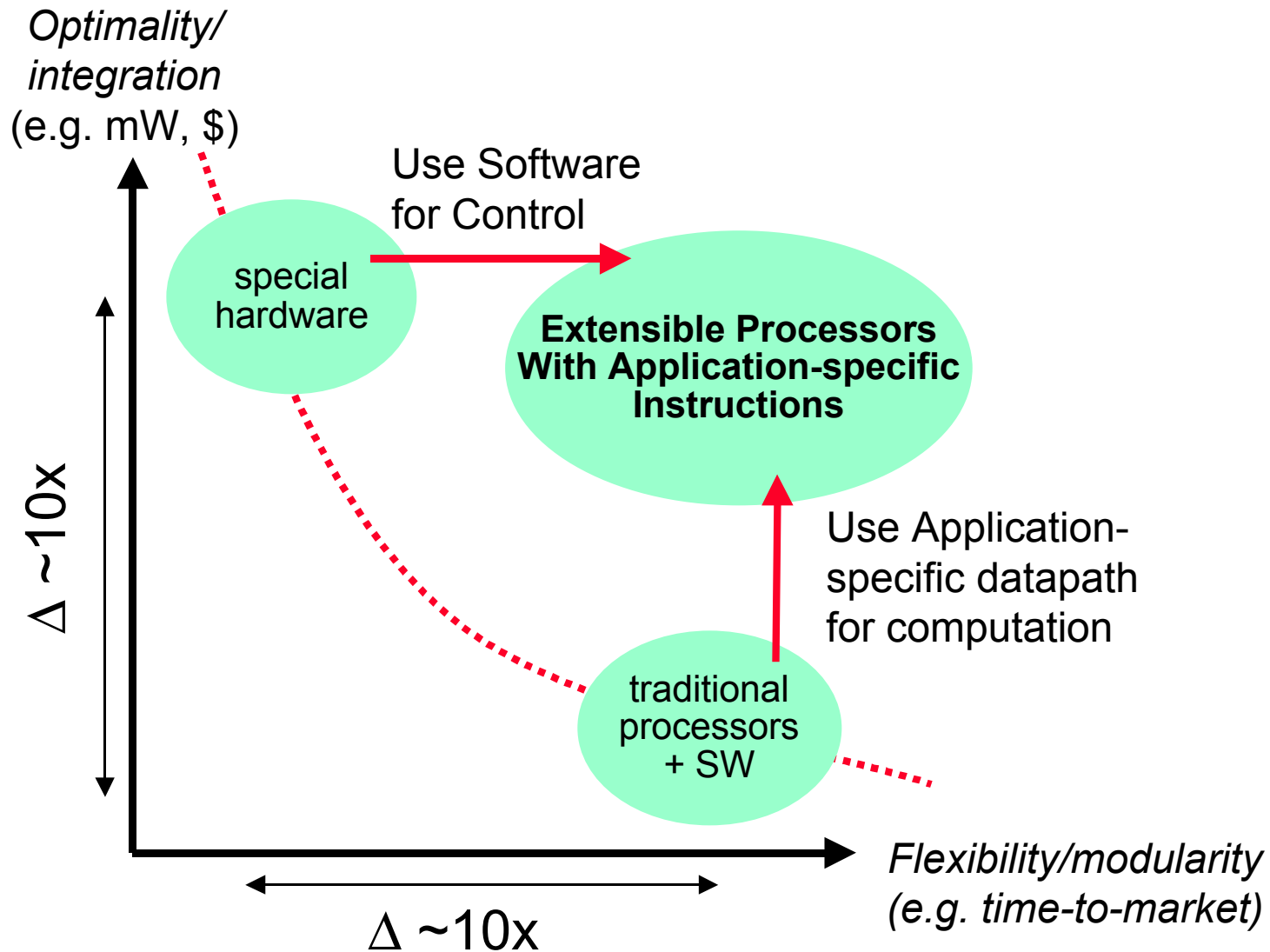


EEMBC Telecom Benchmark

Telemark



- ❖ **Configurable / Extensible processor solution**
 - Xtensa architecture
 - Instruction extension automation
 - A detailed example
- ❖ **EEMBC benchmarks – a case study**
- ❖ **Summary**



Thank You!