

# **Implementation of the Elliptic Curve Method of Factoring in Reconfigurable Hardware**

**Kris Gaj**

**Soonhak Kwon**

**Patrick Baier**

**Paul Kohlbrenner**

**Hoang Le**

**Khaleeluddin Mohammed**

**Ramakrishna Bachimanchi**

**George Mason University**

# GMU Team

Mathematicians/ Cryptographers

Software experiments



**Soonhak Kwon**

Ph.D in Mathematics,  
Johns Hopkins University  
Maryland, U.S  
Visiting professor at GMU  
on leave from  
Sungkyunkwan  
University, Suwon, Korea

**Patrick Baier**

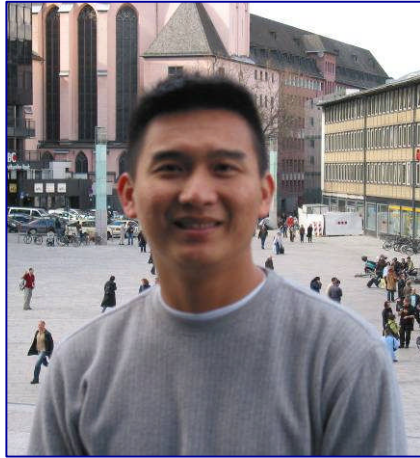
D. Phil. in  
Mathematics,  
Oxford University  
Oxford, U.K  
Affiliated with George  
Washington Univeristy

**Paul Kohlbrenner**

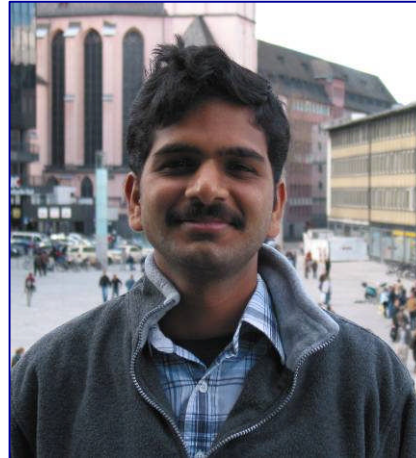
Ph.D student,  
ECE Department  
George Mason University  
Virginia, U.S

# GMU Team

Hardware design



Hoang Le



Ramakrishna Bachimanchi



Khaleeluddin Mohammed

MS in Computer Engineering students  
ECE Department  
George Mason University  
Virginia, U.S.A.

# Objectives

- **Efficient & portable hardware implementation of Elliptic Curve Method** of factoring with an application to the relation collection step of NFS
- **Comparison among** FPGA, ASIC and microprocessor technologies
- Initial study on **porting ECM to** existing general-purpose & special-purpose **reconfigurable computers**

# Outline

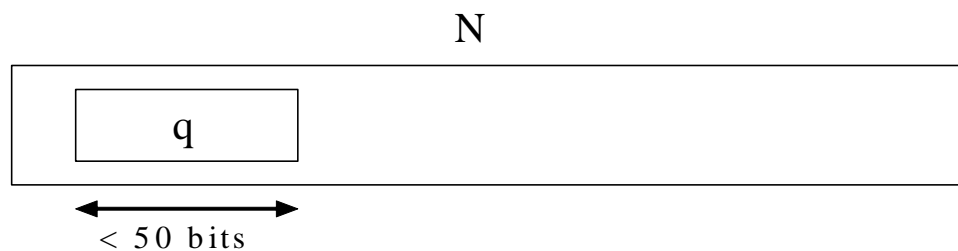
- **ECM 101**
- **Hardware Architecture**
- **Results**
- **Comparison among FPGAs, Microprocessors & ASICs**
- **RCM 101**  
**(Reconfigurable Computing Machines)**
- **Conclusions**

# **ECM 101**

# What is ECM?

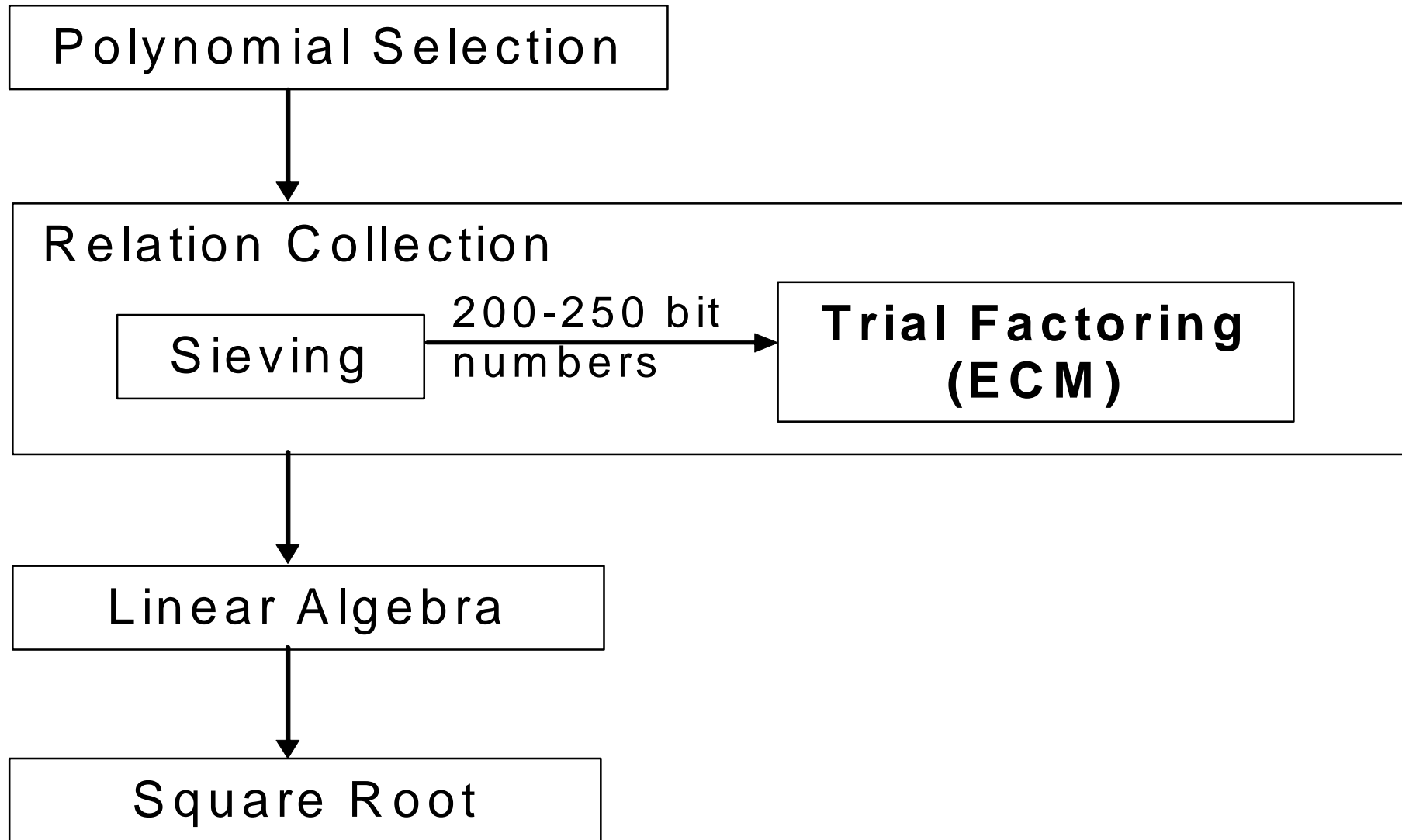
## Elliptic Curve Method of Factoring

Lenstra	1985	Phase 1
Brent, Montgomery	1986-87	Phase 2



Factoring time depends mainly on the size of factor  $q$

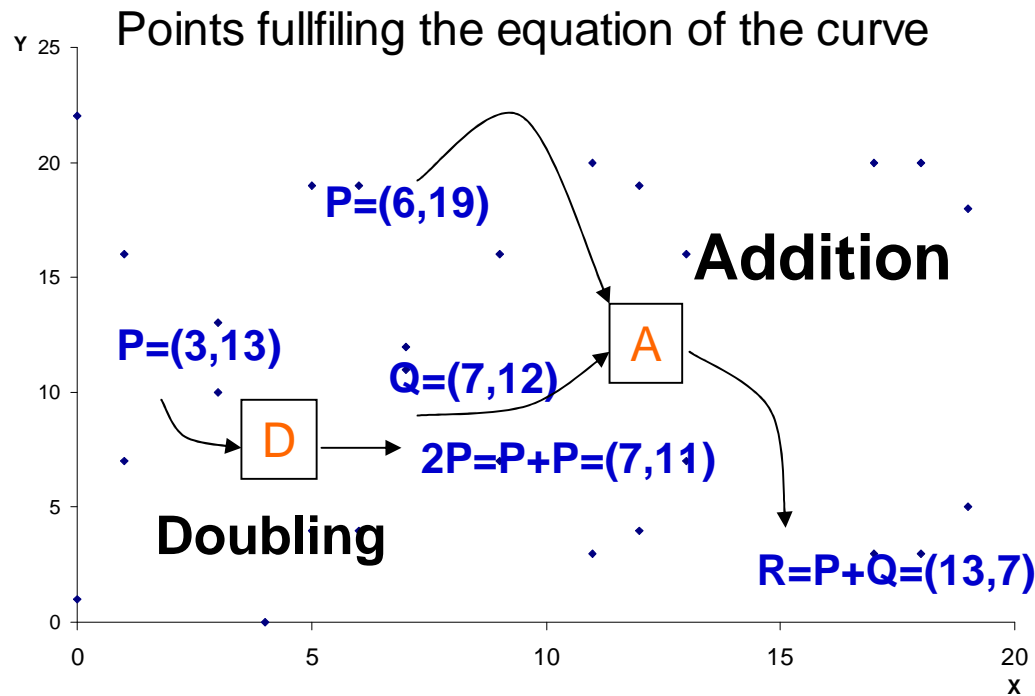
# ECM in the Number Field Sieve (NFS)





# Elliptic Curve

$$Y^2 = X^3 + X + 1 \pmod{p} \quad (p = 23)$$



+ special point  $\mathcal{G}$   
(point at infinity)  
such that:

$$P + \mathcal{G} = \mathcal{G} + P = P$$

$$\exists P: \underbrace{P, 2P, 3P, \dots, nP}_{\text{all points of the curve}} = \mathcal{G}, (n+1)P = P, 2P$$

# Projective vs. Affine coordinates

- affine coordinates

$$P_a = (X_P, Y_P)$$

- addition and doubling **require inversion**

- projective coordinates

$$P_p = (X_P, Y_P, Z_P)$$

- addition and doubling can be done **without inversion**

- projective coordinates for Montgomery form of the curve

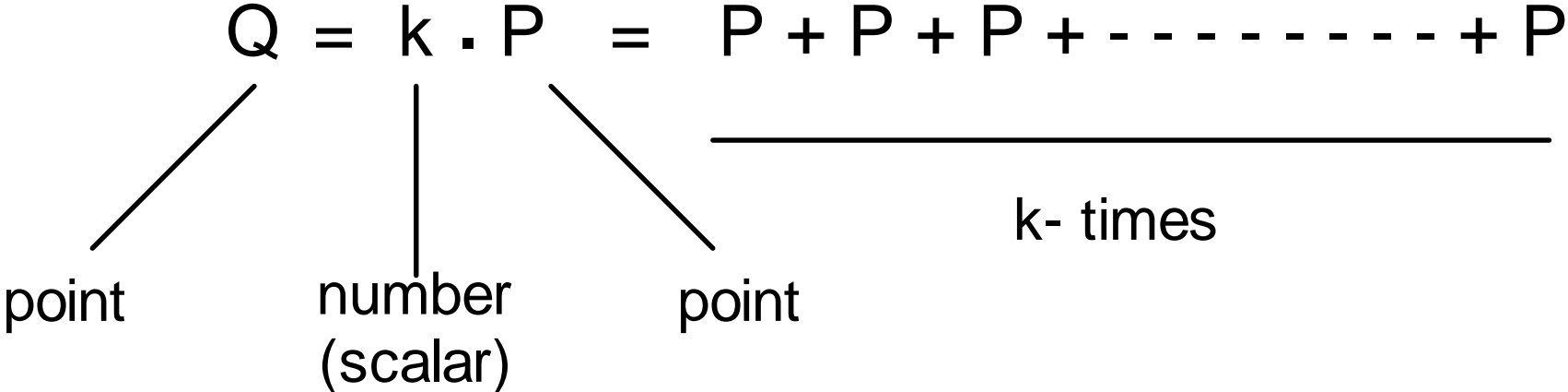
- addition and doubling **do not require y coordinate**

(y coordinate can be recovered from x and z at the end of a long chain of computations)

$$P_{pM} = (X_P : : Z_P)$$

$$\mathcal{O} = (0 : : 0)$$

# Scalar Multiplication



# ECM Algorithm

## Inputs :

- N – number to be factored
- $P_0$  – point of a curve  $E$  : initial point
- $B_1$  – smoothness bound for Phase1
- $B_2$  – smoothness bound for Phase2

## Outputs:

- q - factor of N,  $1 < q \leq N$   
or FAIL

# ECM algorithm – Phase 1

1:  $k \leftarrow \prod_{p_i} p_i^{e_i}$  such that  $p_i$  - consecutive primes  $< B_1$

$e_i$  - largest exponent such that  $p_i^{e_i} \leq B_1$

2:  $Q_0 \leftarrow kP_0 = (x_{Q_0} : : z_{Q_0})$

3:  $q \leftarrow \gcd(z_{Q_0}, N)$

4: if  $q > 1$

5:   return  $q$    (factor of  $N$ )

6: else

7:   go to Phase 2

8: end if

## ECM algorithm – Phase 2

```
09:  $d \leftarrow 1$ 
10: for each prime  $p = B_1$  to  $B_2$  do
11:    $(x_{pQ_0}, y_{pQ_0}, z_{pQ_0}) \leftarrow pQ_0$ 
12:    $d \leftarrow d \cdot z_{pQ_0} \pmod{N}$ 
13: end for
14:  $q \leftarrow \gcd(d, N)$ 
15: if  $q > 1$  then
16:   return  $q$ 
17: else
18:   return FAIL
19: end if
```

# Phase 1 – Numerical example

$$N = 1\,740\,719 = 1279 \cdot 1361$$

$$E : y^2 = x^3 + 14x + 1 \pmod{1\,740\,719}$$

$$P_0 = (5 : : 1)$$

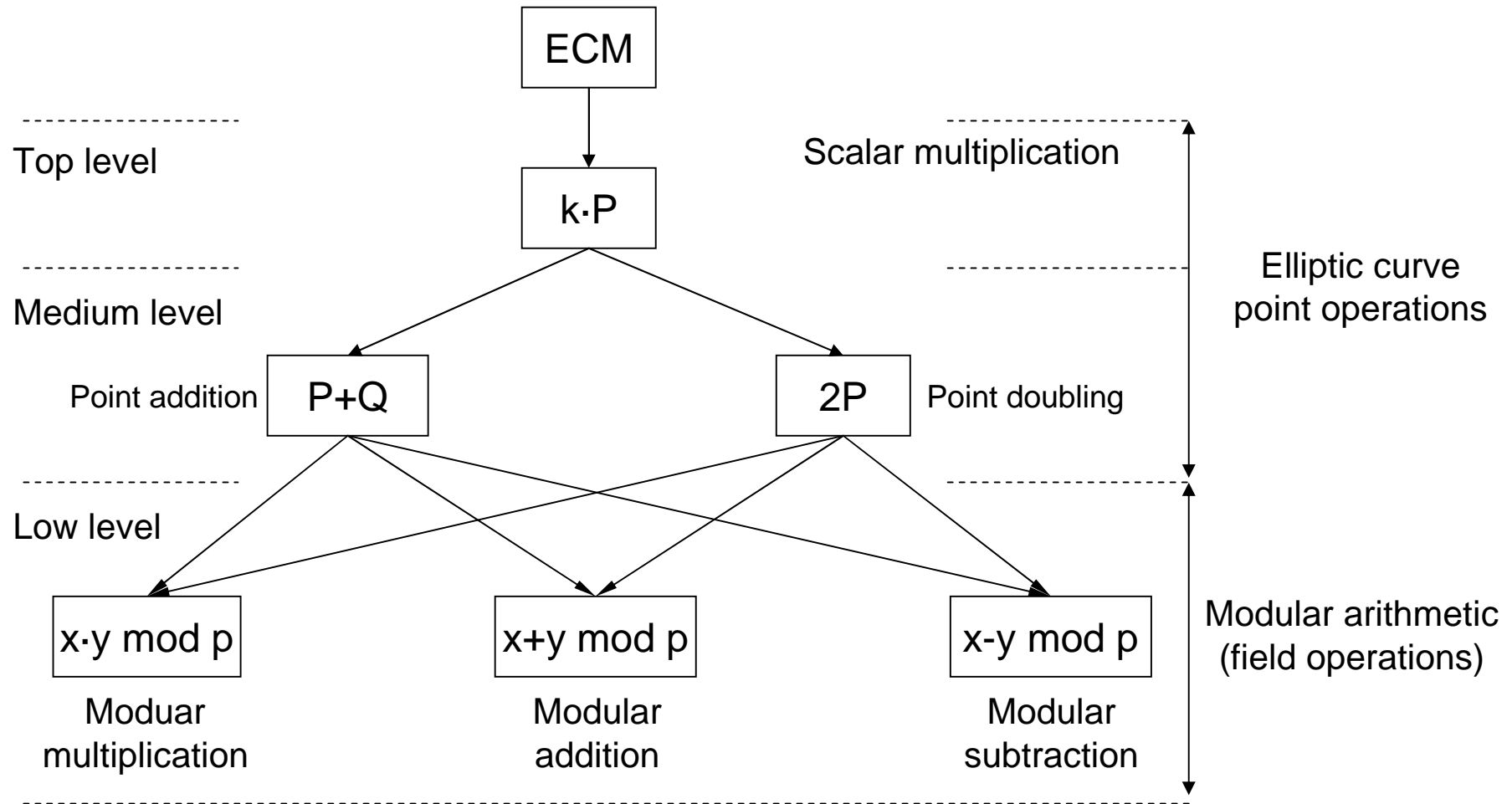
$$B_1 = 20$$

$$k = 2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 = 232\,792\,560$$

$$kP_0 = (707\,838 : : 1\,686\,279)$$

$$\gcd(1\,686\,279, 1\,740\,719) = \mathbf{1361}$$

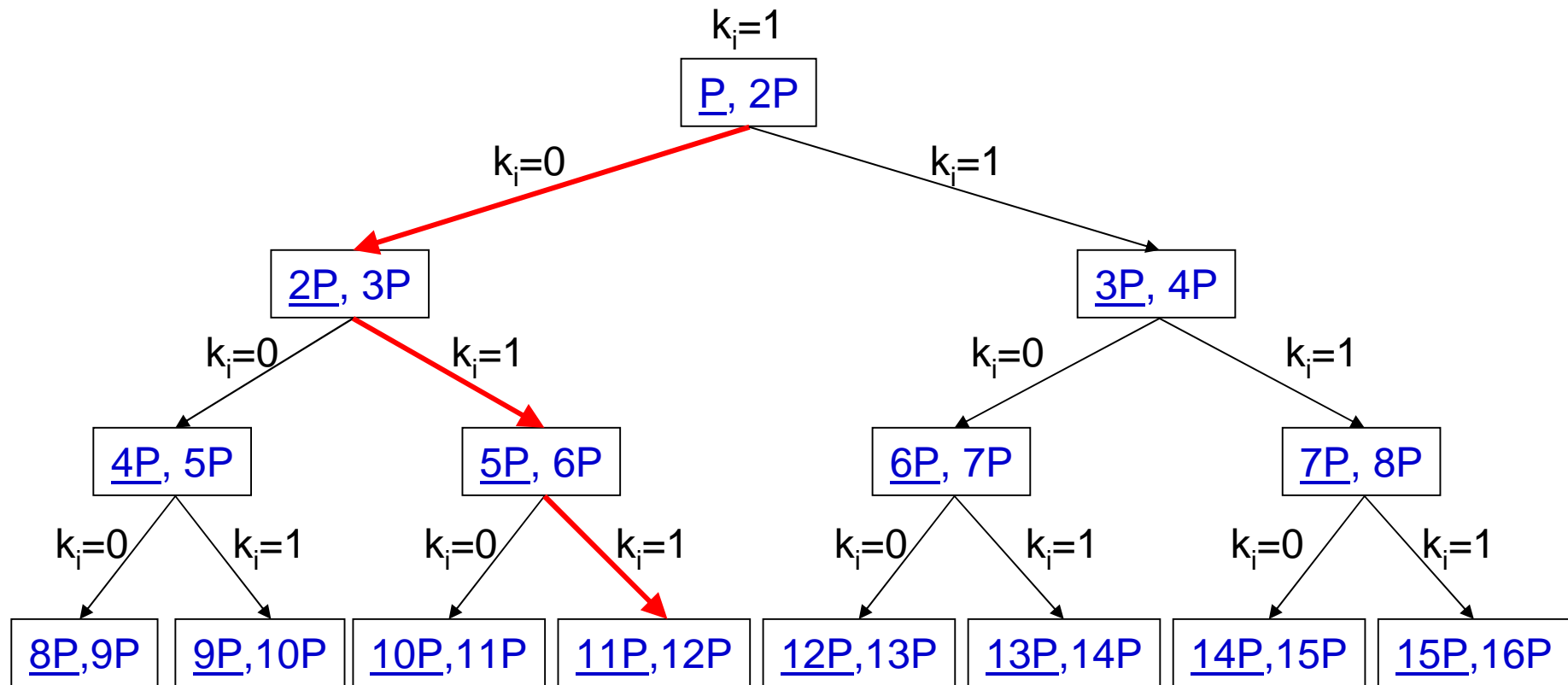
# Hierarchy of Elliptic Curve Operations





# Top level : scalar multiplication

## Montgomery ladder - concept



$$11 = 1011_2$$

# Top Level : Scalar multiplication Montgomery ladder – pseudo code

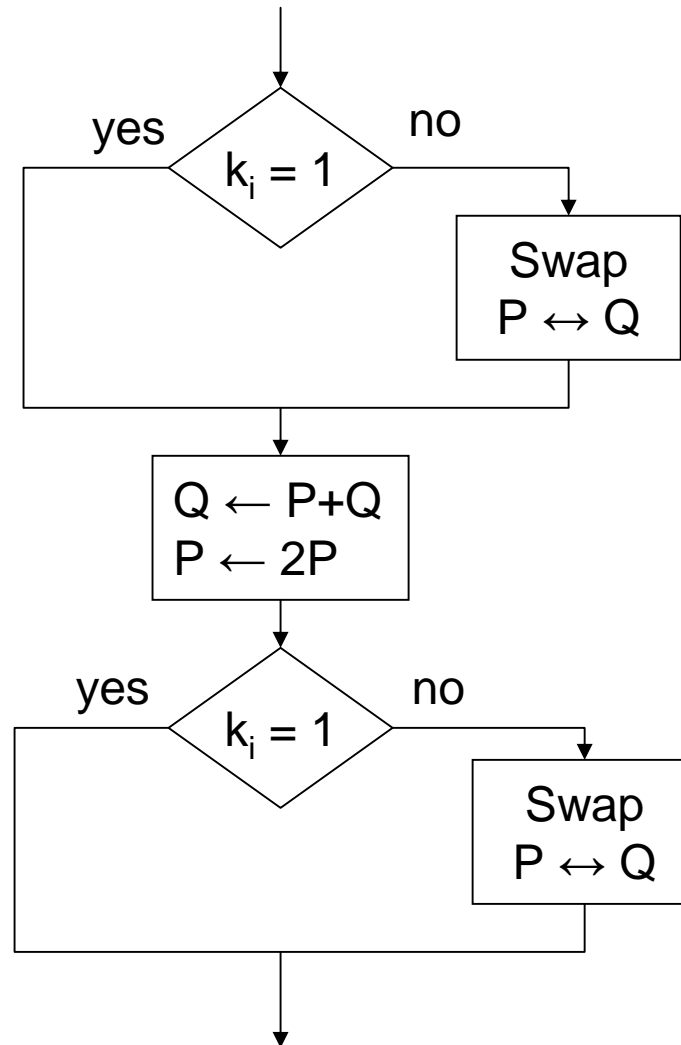
Input :  $P_0 \in E$  ( $x_0 \neq 0$ ),  $k = (k_{s-1}, k_{s-2}, \dots, k_1, k_0)_2$   $k_{s-1} = 1$

Output :  $kP_0$

```
1:  $Q \leftarrow P_0, P \leftarrow 2P_0$ 
2: for  $i = s - 2$  downto 0 do
3:   if  $k_i = 1$  then
4:      $Q \leftarrow P + Q, P \leftarrow 2P$ 
5:   else
6:      $Q \leftarrow 2Q, P \leftarrow P + Q$ 
7:   end if;
8: end for
9: return  $Q$ 
```

# Montgomery ladder algorithm

## Basic step



# Medium level

## Point addition & doubling

Input  $P = (x_P : : z_P)$   $Q = (x_Q : : z_Q)$   $P - Q = (x_{P-Q} : : z_{P-Q})$   $P, Q, P - Q \in E$

$$a_{24} = \frac{a+2}{4} \text{ where } a \text{ is a parameter of the curve } E$$

Output  $P + Q = (x_{P+Q} : : z_{P+Q})$   $2P = (x_{2P} : : z_{2P})$

$$x_{P+Q} = z_{P-Q} \left( (x_P - z_P)(x_Q + z_Q) + (x_P + z_P)(x_Q - z_Q) \right)^2$$

$$z_{P-Q} = x_{P-Q} \left( (x_P - z_P)(x_Q + z_Q) - (x_P + z_P)(x_Q - z_Q) \right)^2$$

$$4x_P z_P = (x_P + z_P)^2 - (x_P - z_P)^2$$

$$x_{2P} = (x_P + z_P)^2 (x_P - z_P)^2$$

$$z_{2P} = 4x_P z_P \left( (x_P - z_P)^2 + a_{24} \cdot (4x_P z_P) \right)$$

**Point addition**

**6 multiplications when  $z_{P-Q} \neq 1$**

**5 multiplications when  $z_{P-Q} = 1$**

**Point doubling**

**5 multiplications**

# Low level : Montgomery Multiplication

$$Z = X \cdot Y \pmod{N} \quad \text{requires division}$$

**Ordinary domain**      **Montgomery domain**

$$X \rightarrow X' = X \cdot 2^n \pmod{N} \quad \text{where } n = \lfloor \log_2 N \rfloor + 2$$

$$Y \rightarrow Y' = Y \cdot 2^n \pmod{N} \quad \text{and } N \text{ odd}$$

$$Z = X \cdot Y \pmod{N} \leftarrow Z' = (X \cdot Y)' = \text{MP}(X', Y', N) = X \cdot Y \cdot 2^n \pmod{N}$$

$$\text{MP}(X', Y', N) \equiv \underline{X' \cdot Y' \cdot 2^{-n} \pmod{N}} \equiv (X \cdot 2^n) (Y \cdot 2^n) \cdot 2^{-n} \pmod{N}$$

$\downarrow$

$$\equiv X \cdot Y \cdot 2^n \pmod{N}$$

does not require division

→ faster than ordinary modular multiplication

In ECM we can do all computations in Montgomery domain because

$$\gcd(z, N) = \gcd(z' = z \cdot 2^n \pmod{N}, N) \text{ for any odd } N$$

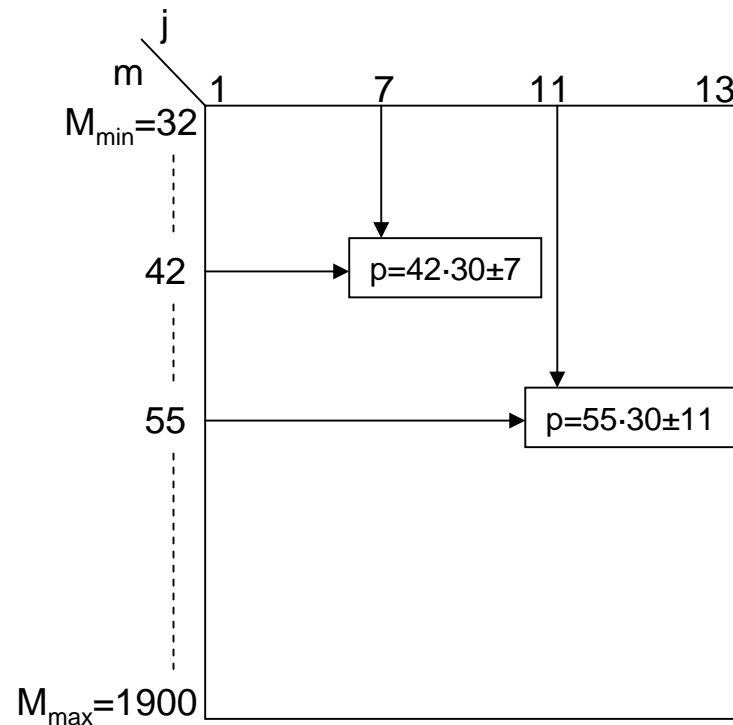
# Phase 2 : Standard continuation algorithm – Basic concept (Step 1)

We express all primes  $B_1 < p < B_2$  in the form

$$p = m \cdot D \pm j \quad \text{where } 1 \leq j \leq \frac{D}{2} \text{ and } \gcd(j, D) = 1$$

For example  $D = 30$

$$B_1 = 960 \quad B_2 = 57000 \rightarrow 1 \leq j \leq 15 \text{ and } \gcd(j, 30) = 1$$



# Phase 2 : Standard continuation algorithm – Basic concept (Step 2)

$$p \cdot Q_0 = \mathcal{G} \pmod{q}$$

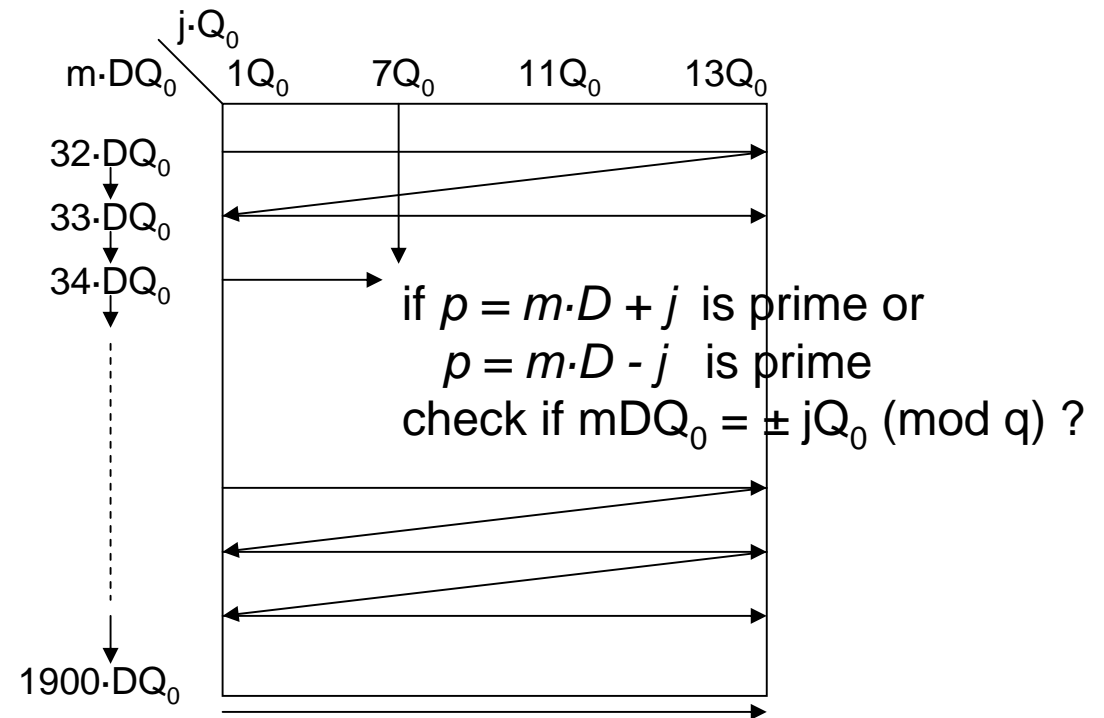
$$\Rightarrow (m \cdot D \pm j) Q_0 = \mathcal{G}$$

$$\Rightarrow m \cdot D Q_0 = \pm j \cdot Q_0$$

We sequentially compute

$m \cdot D Q_0$   
for  $M_{min} \leq m \leq M_{max}$

We pre-compute  $j Q_0$



# Phase 2 : Standard continuation algorithm – Basic concept (Step 3)

$$pQ_0 = \mathcal{G} \pmod{q}$$

$$\Rightarrow mDQ_0 = \pm jQ_0$$

$$(x_{mDQ_0} : : z_{mDQ_0}) = c \cdot (x_{jQ_0} : : z_{jQ_0}) \pmod{q} \quad \text{where} \quad \begin{cases} x_{-jQ_0} = x_{+jQ_0} \\ z_{-jQ_0} = z_{+jQ_0} \end{cases}$$

$$\Rightarrow \frac{x_{mDQ_0}}{x_{jQ_0}} = \frac{z_{mDQ_0}}{z_{jQ_0}} \pmod{q}$$

$$\Rightarrow x_{mDQ_0} \cdot z_{jQ_0} = x_{jQ_0} \cdot z_{mDQ_0} \pmod{q}$$

$$\Rightarrow q \mid (x_{mDQ_0} \cdot z_{jQ_0} - x_{jQ_0} \cdot z_{mDQ_0})$$

$$q \mid d = \prod_{\substack{p = m \cdot D \pm j \\ B_1 < p \leq B_2}} (x_{mDQ_0} \cdot z_{jQ_0} - x_{jQ_0} \cdot z_{mDQ_0}) \wedge q \mid N$$

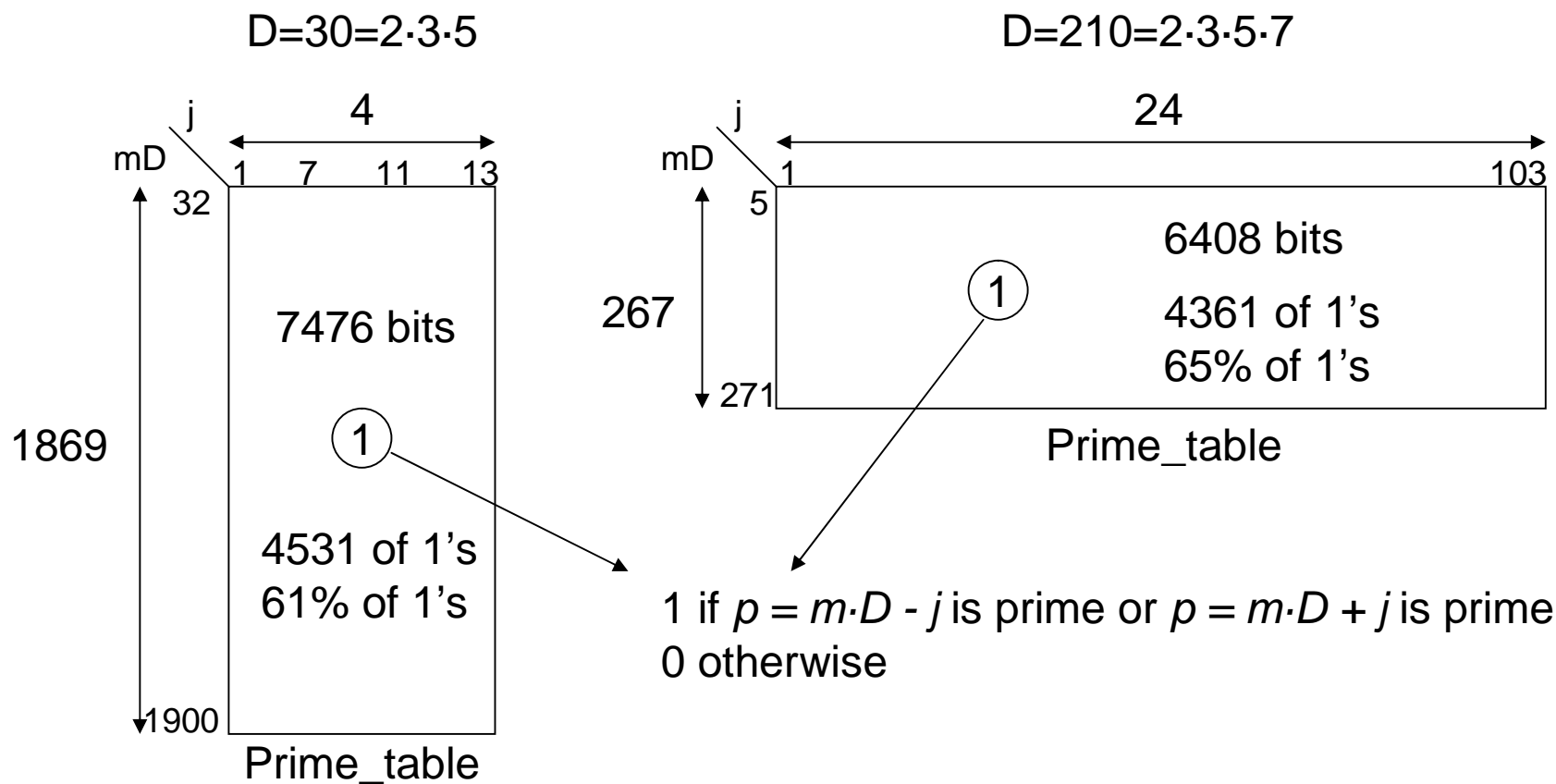
$$\Rightarrow q \mid \gcd(d, N)$$



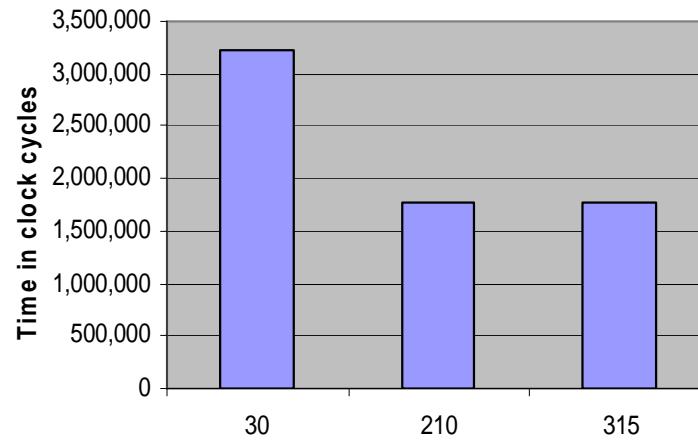
# Choice of D

$$B_1=960$$

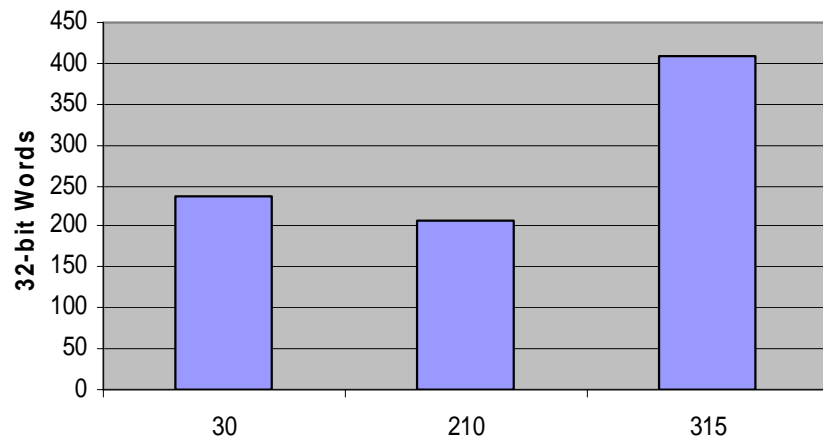
$$B_2=57,000$$



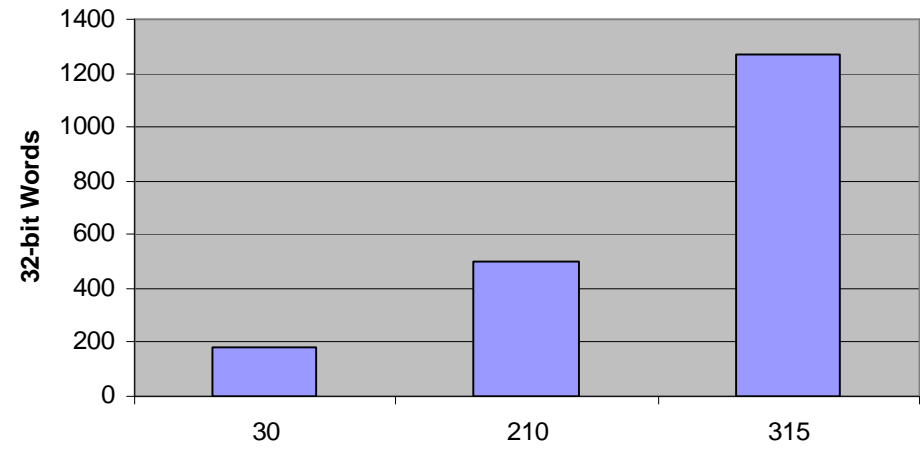
# Phase 2: Time and Memory Usage as a function of D



**Phase 2 Time**



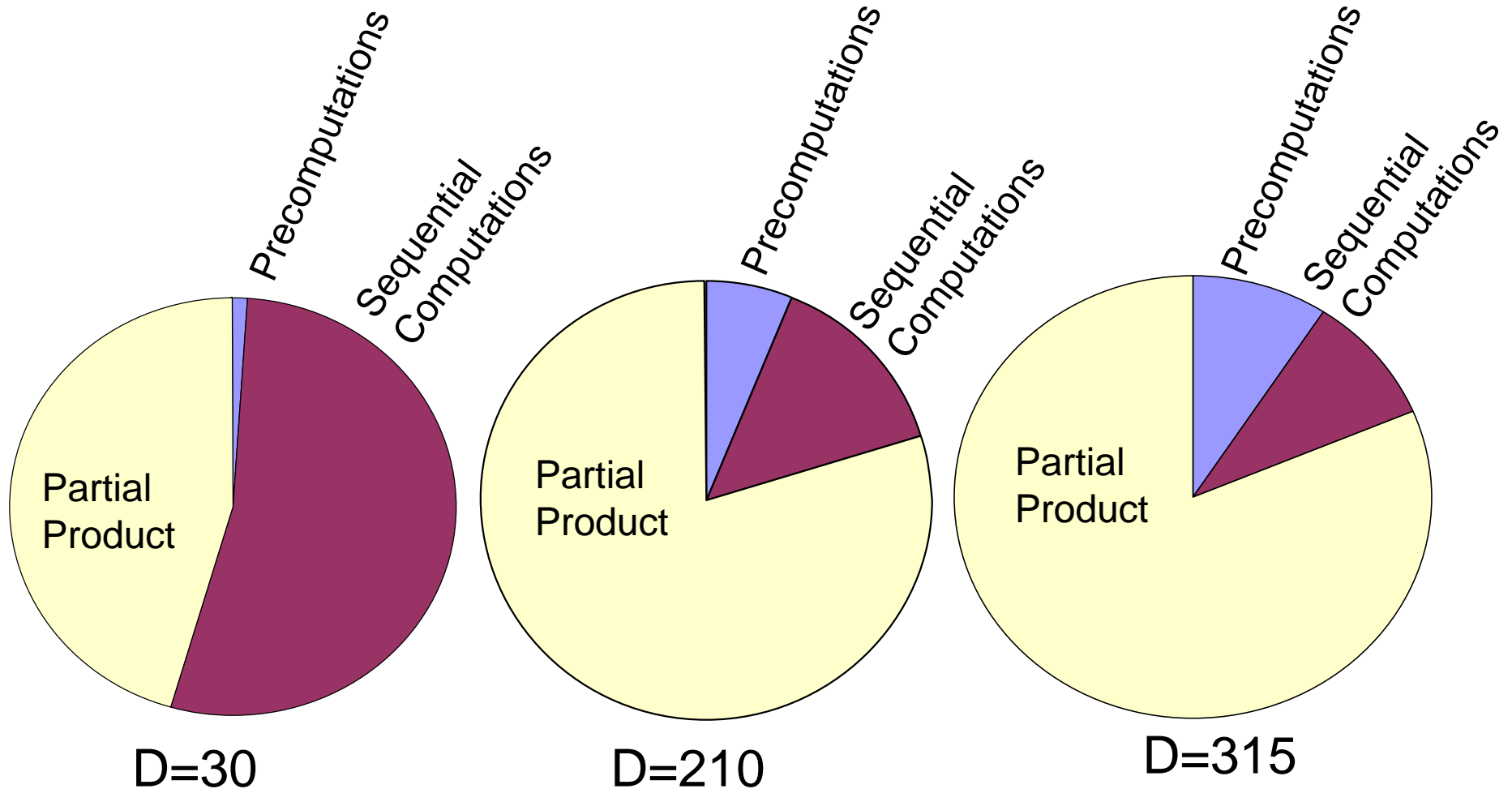
**Phase 2 Global Memory**



**Phase 2 Local Memory**

# Time of Phase 2

## Contributions of Various Phases of Computations



# Phase 2 : Standard continuation algorithm – Basic concept (Step 2)

$$p \cdot Q_0 = \mathcal{G} \pmod{q}$$

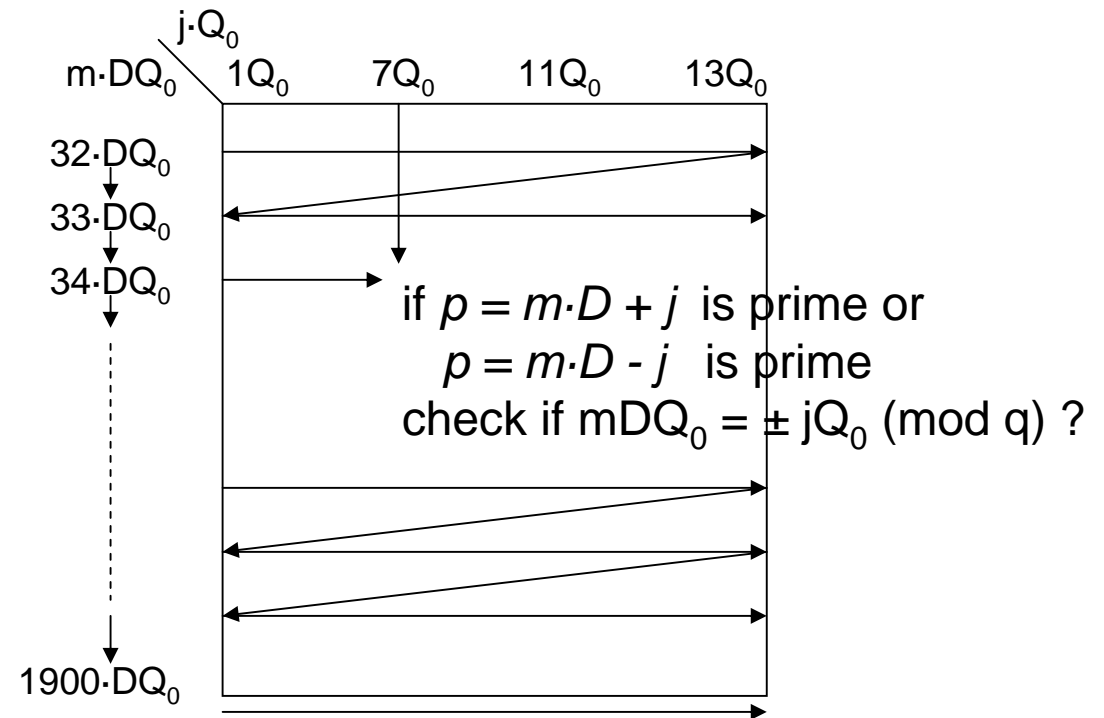
$$\Rightarrow (m \cdot D \pm j) Q_0 = \mathcal{G}$$

$$\Rightarrow m \cdot D Q_0 = \pm j \cdot Q_0$$

We sequentially compute

$m \cdot D Q_0$   
for  $M_{min} \leq m \leq M_{max}$

We pre-compute  $j Q_0$



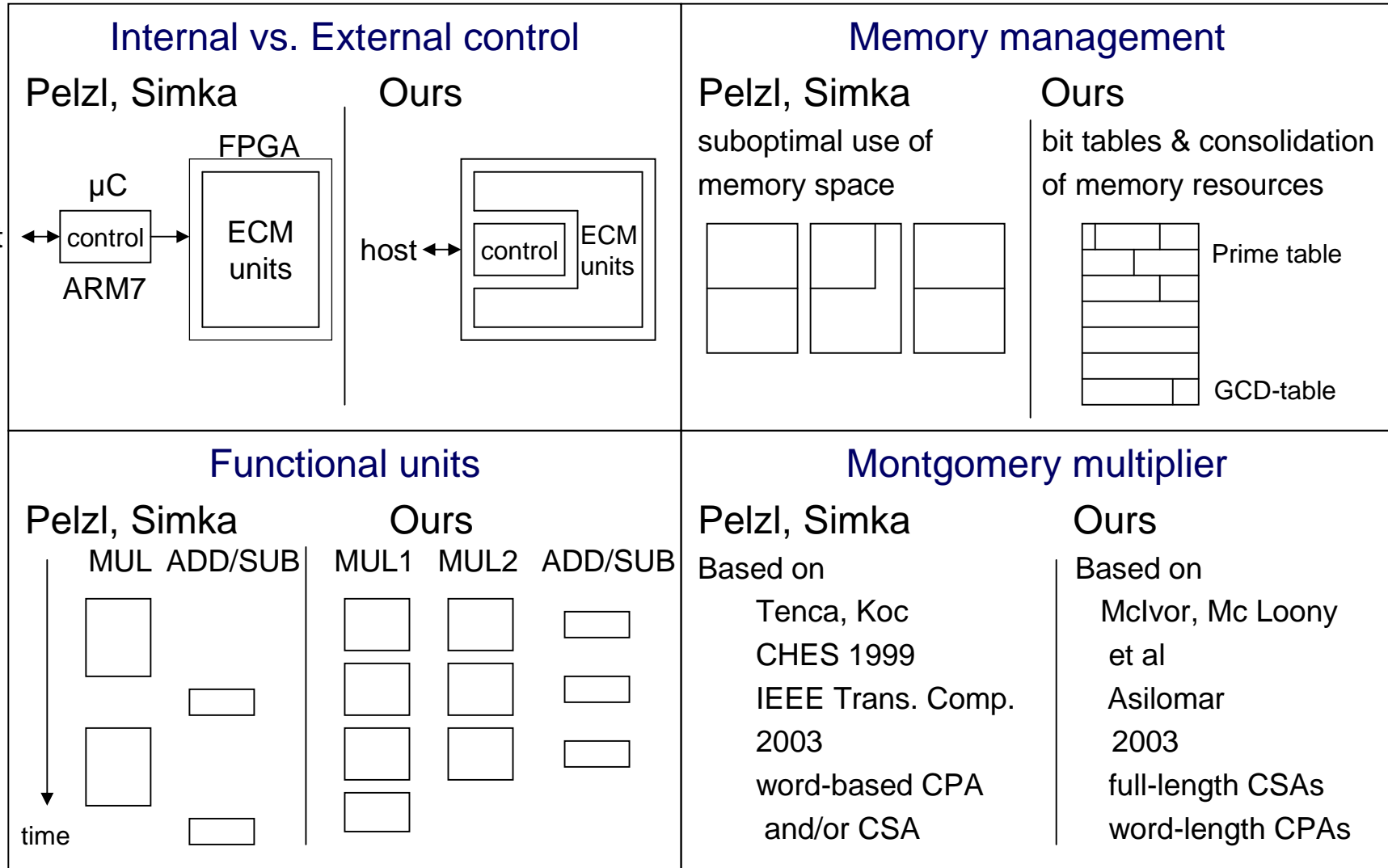
# **ECM IN HARDWARE**

# ECM in Hardware

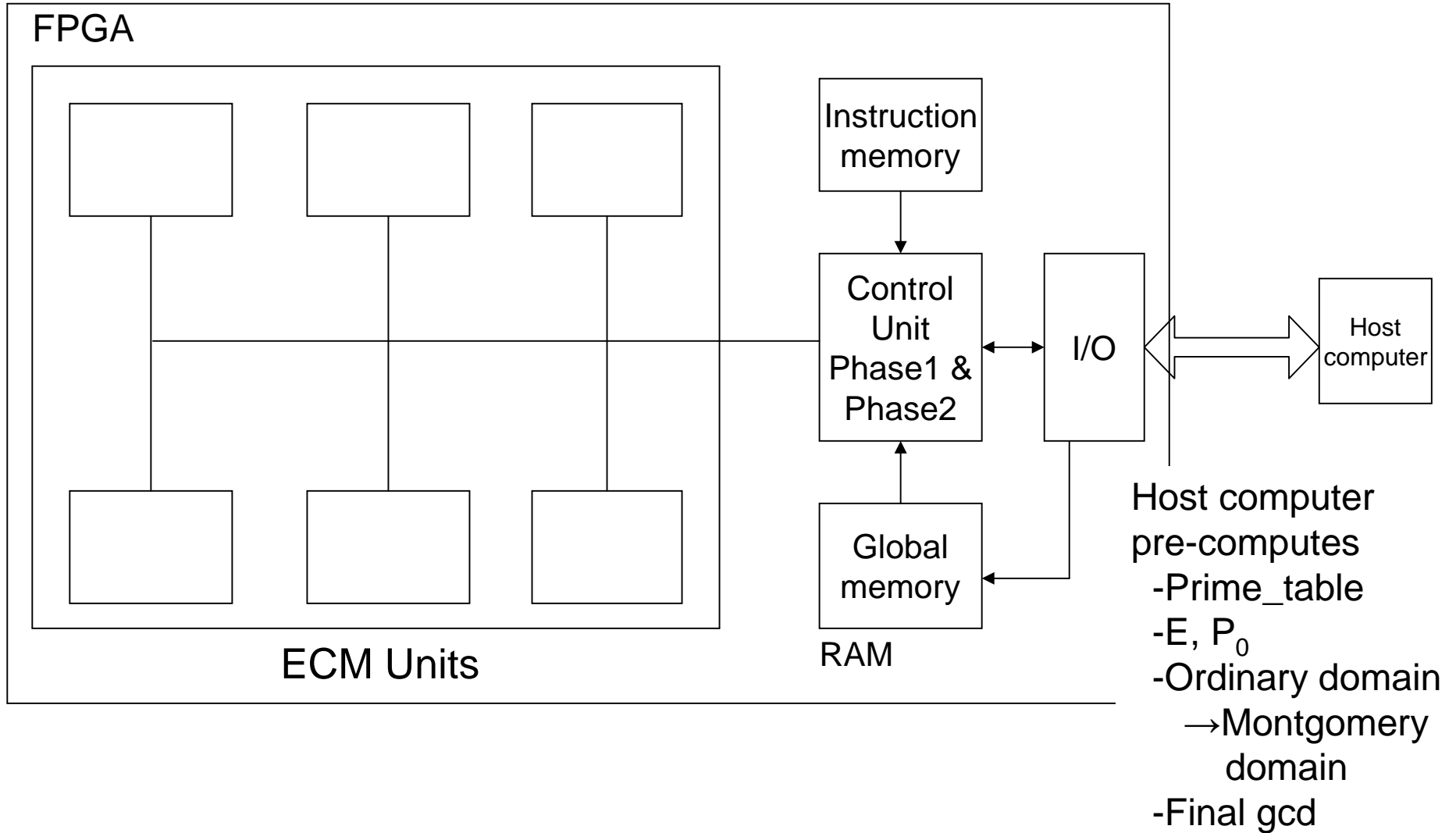
## Previous Proof-of-Concept Design

Pelzl, Šimka,	SHARCS	Feb 2005
Kleinjung, Franke,	FCCM	Apr 2005
Priplata, Stahlke,	IEE Proc.	Oct 2005
Drutarovský, Fischer,		
Paar		

# Modifications compared to Pelzl, Simka

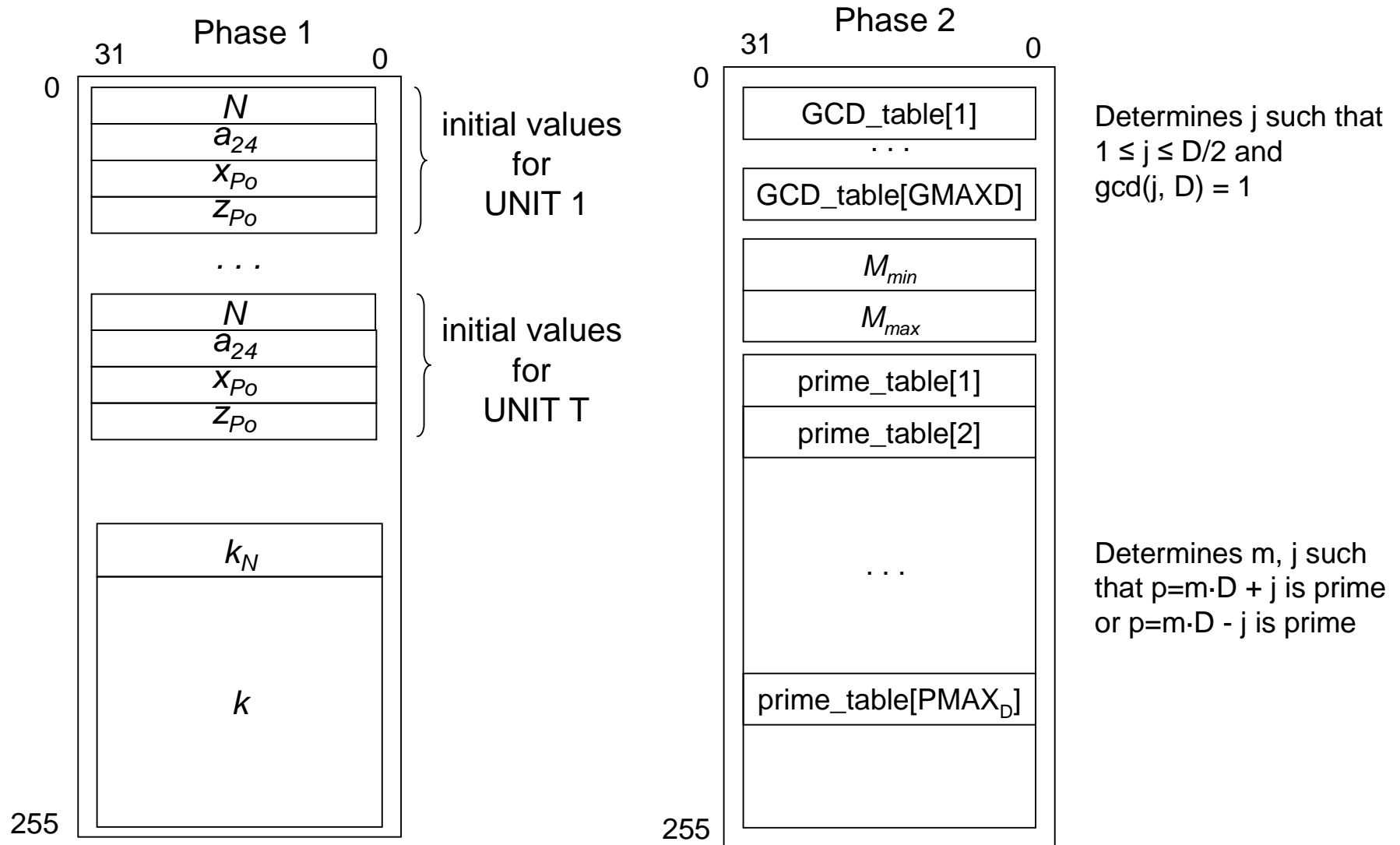


# Our architecture : Top-level view

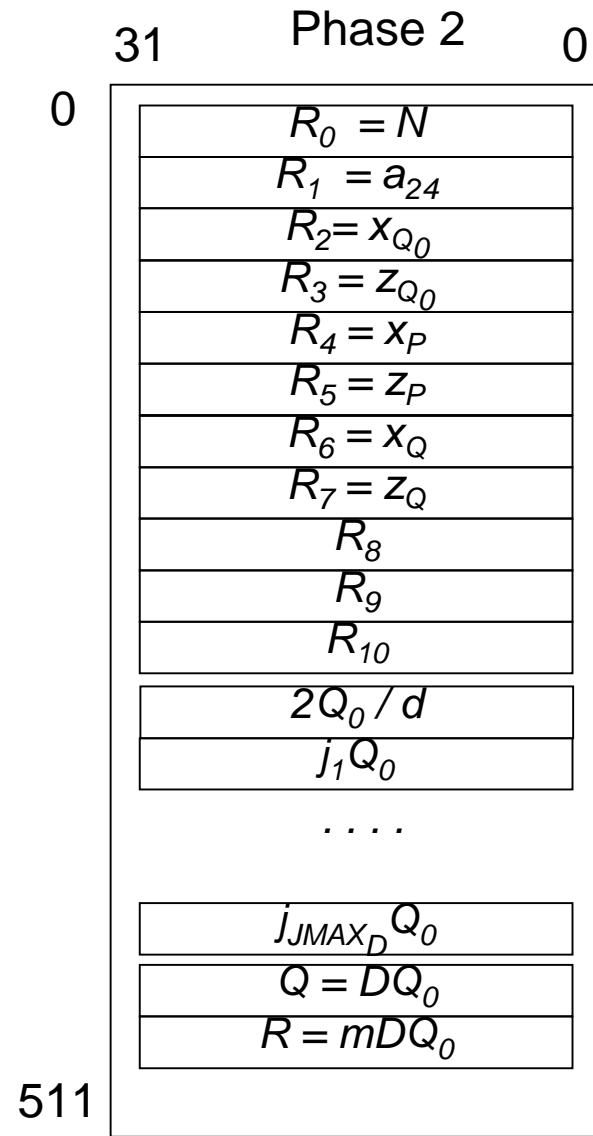
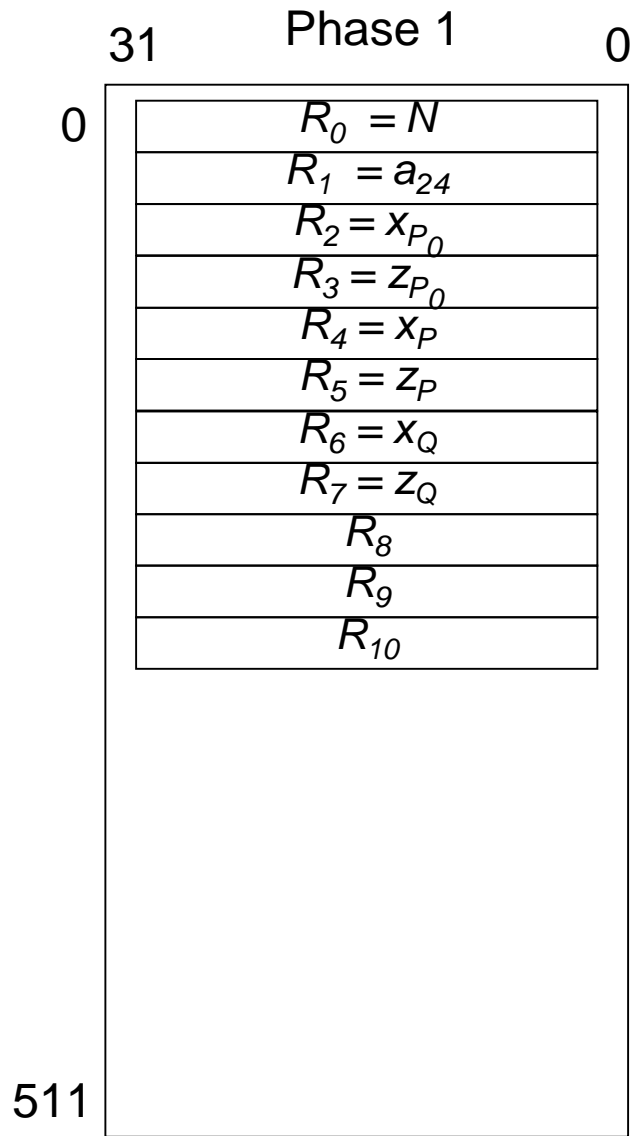




# Global Memory



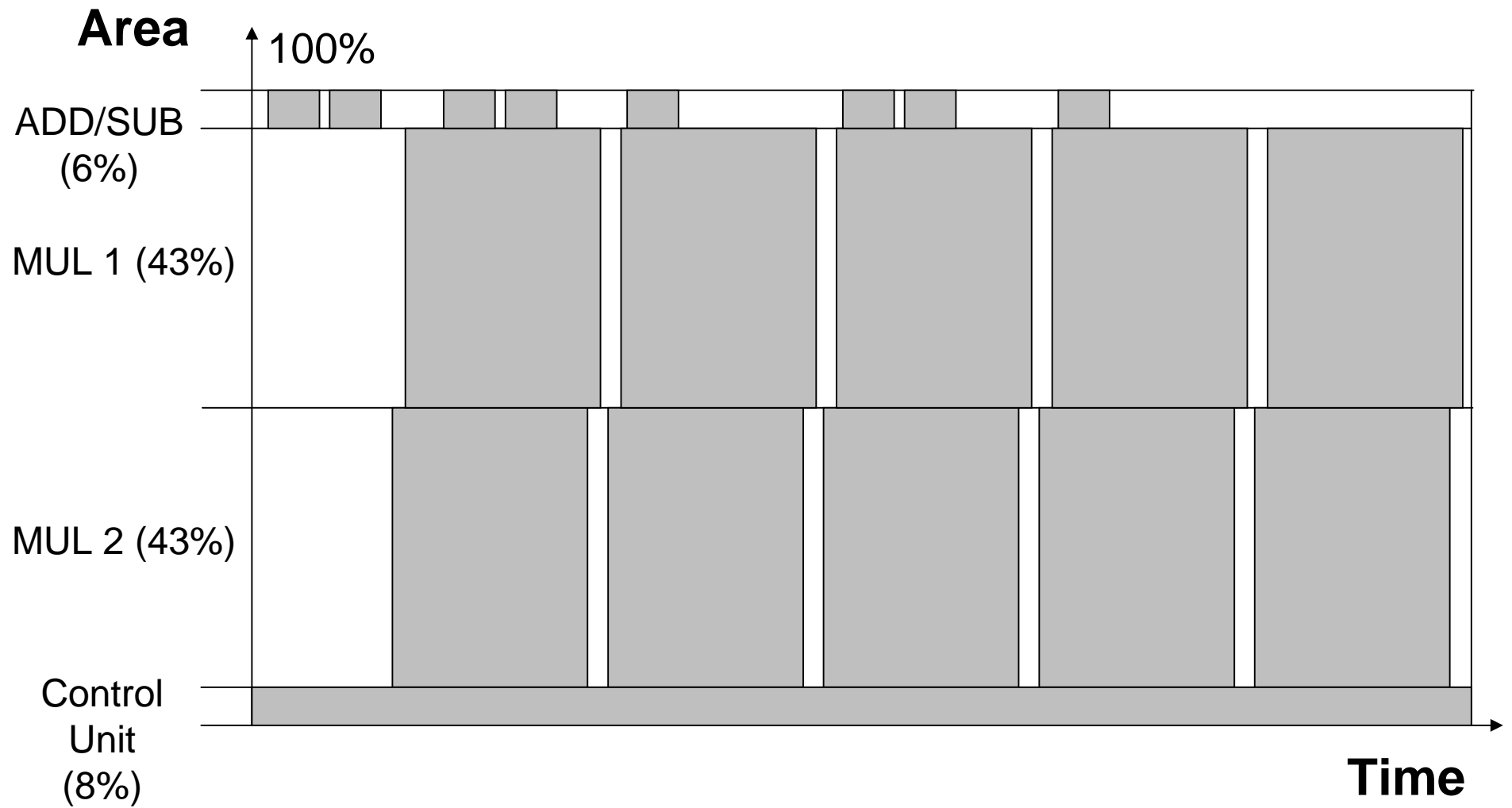
# Local Memory



# Computation Flow for Scalar Multiplication ( $Z_{P-Q} = 1$ )

Adder/Subtractor	Multiplier 1	Multiplier 2
<b>A/D:</b> $a_1 = x_P + z_P$ <b>A/D:</b> $s_1 = x_P - z_P$		
<b>A/D:</b> $a_2 = x_Q + z_Q$ <b>A/D:</b> $s_2 = x_Q - z_Q$	<b>D:</b> $m_1 = s_1^2$	<b>D:</b> $m_2 = a_1^2$
<b>D:</b> $s_3 = m_2 - m_1$	<b>A:</b> $m_3 = s_1 \cdot a_2$	<b>A:</b> $m_4 = s_2 \cdot a_1$
<b>A:</b> $a_3 = m_3 + m_4$ $s_4 = m_3 - m_4$	<b>D:</b> $x_{2P} = m_5 = m_1 \cdot m_2$	<b>D:</b> $m_6 = s_3 \cdot A_{24}$
<b>D:</b> $a_4 = m_1 + m_6$	<b>A:</b> $x_{P+Q} = m_7 = a_3^2$	<b>A:</b> $m_8 = s_4^2$
	<b>A:</b> $z_{P+Q} = m_9 = m_8 \cdot x_{P-Q}$	<b>D:</b> $z_{2P} = m_{10} = s_3 \cdot a_4$

# Resources utilization in time – Phase 1



# RESULTS

# Choice of parameters

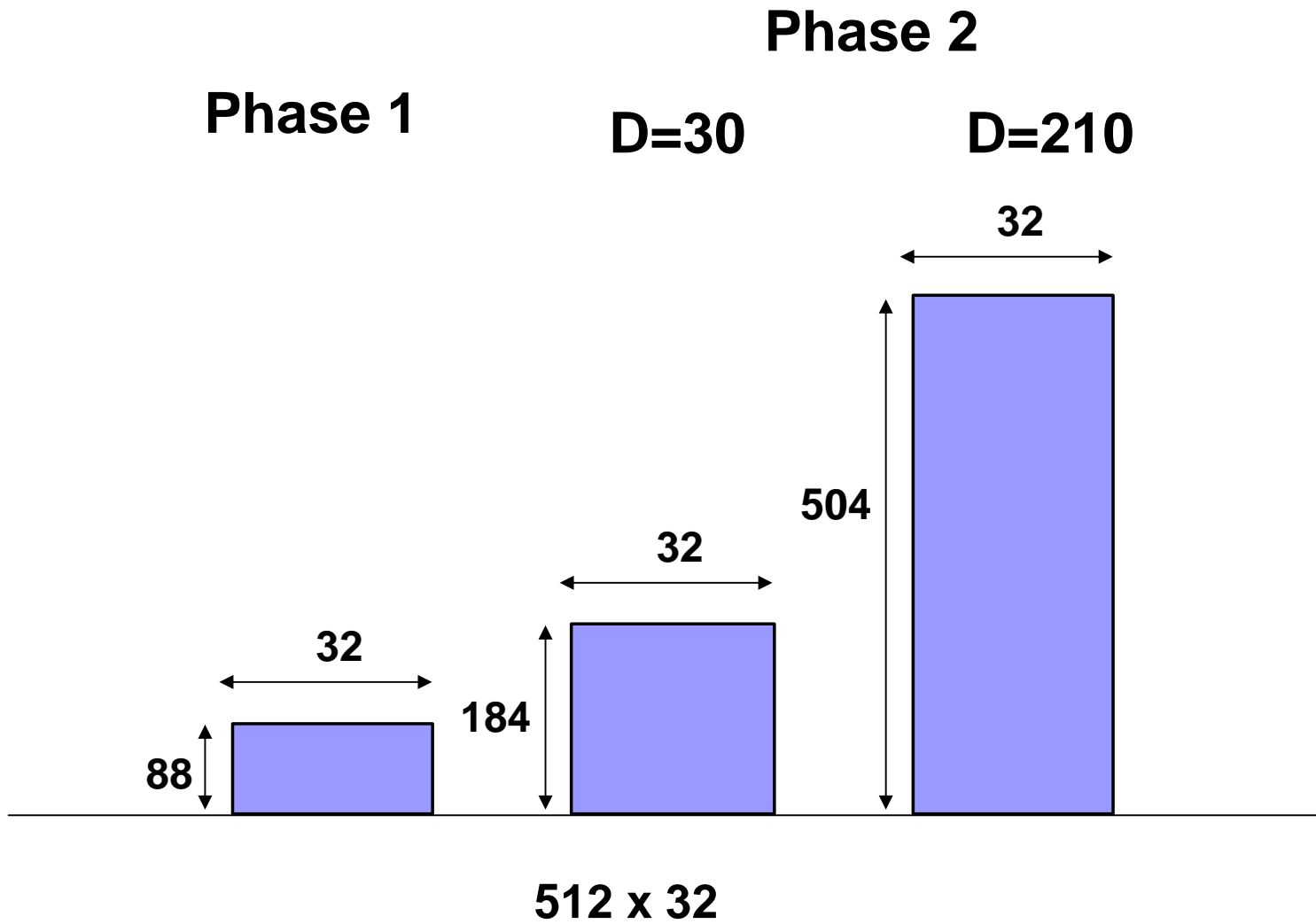
**N: 198 – bit number**

**$B_1 = 960$**

**$B_2 = 57,000$**

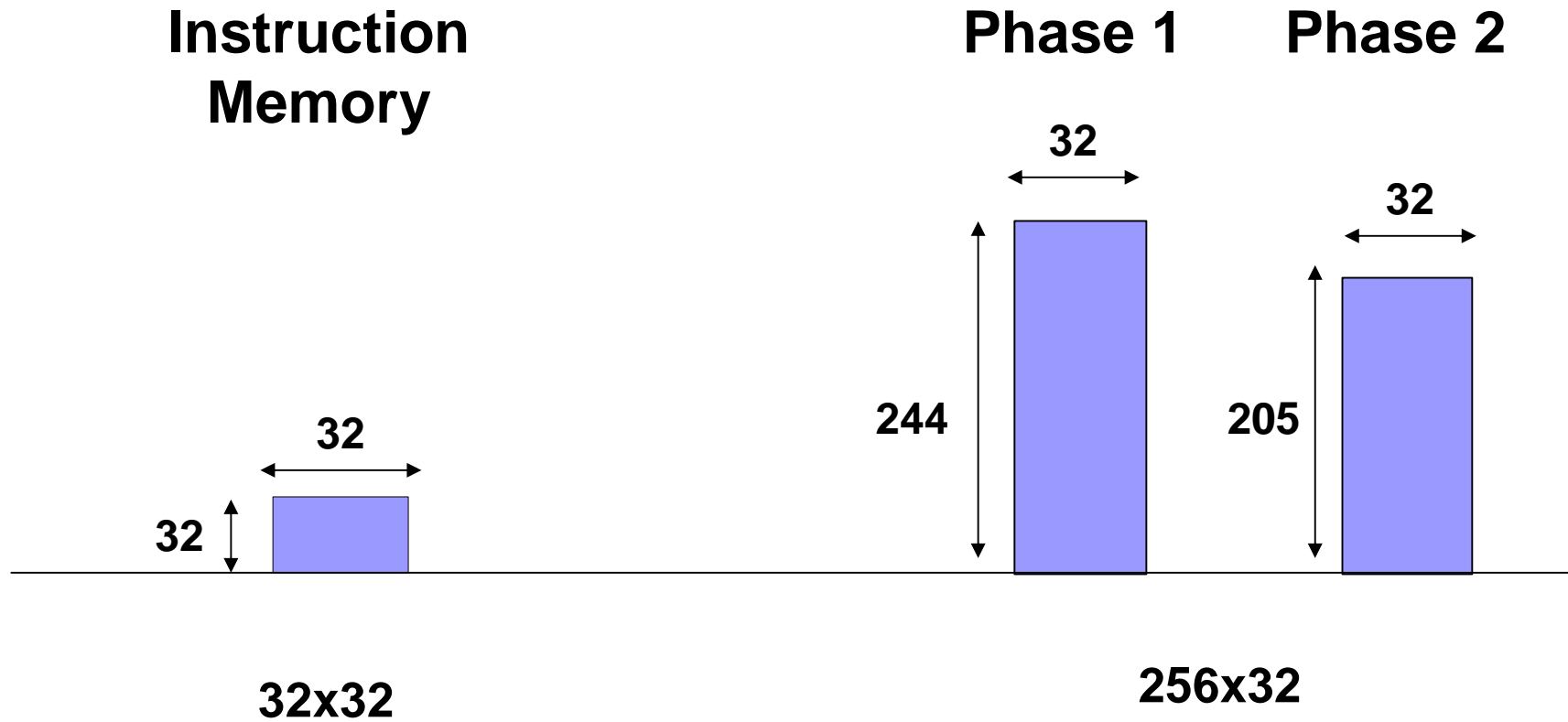
# Memory Requirements

## Local Memory



# Memory Requirements

## Global Memory

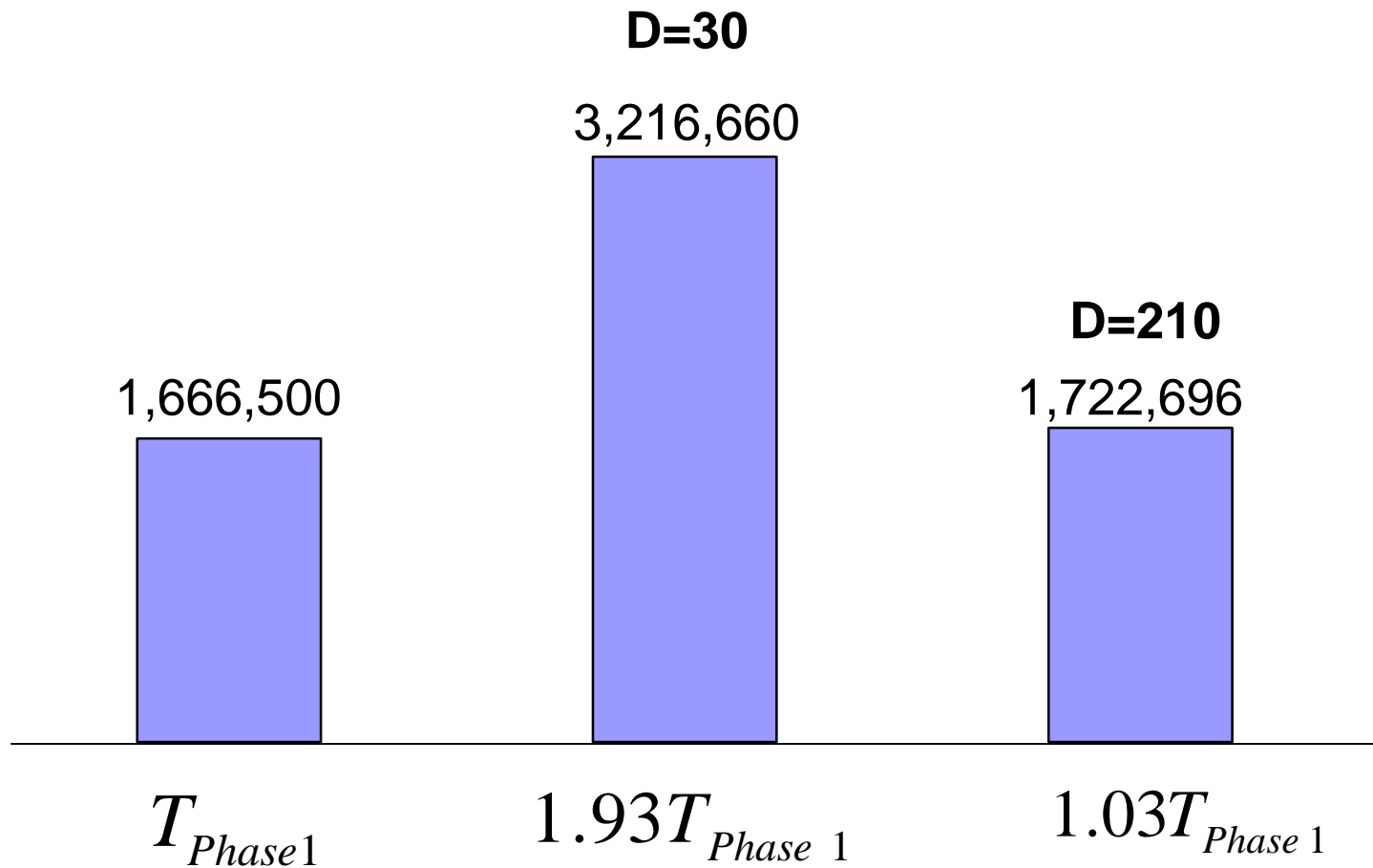




# Execution Time in Clock Cycles

Phase 1

Phase 2

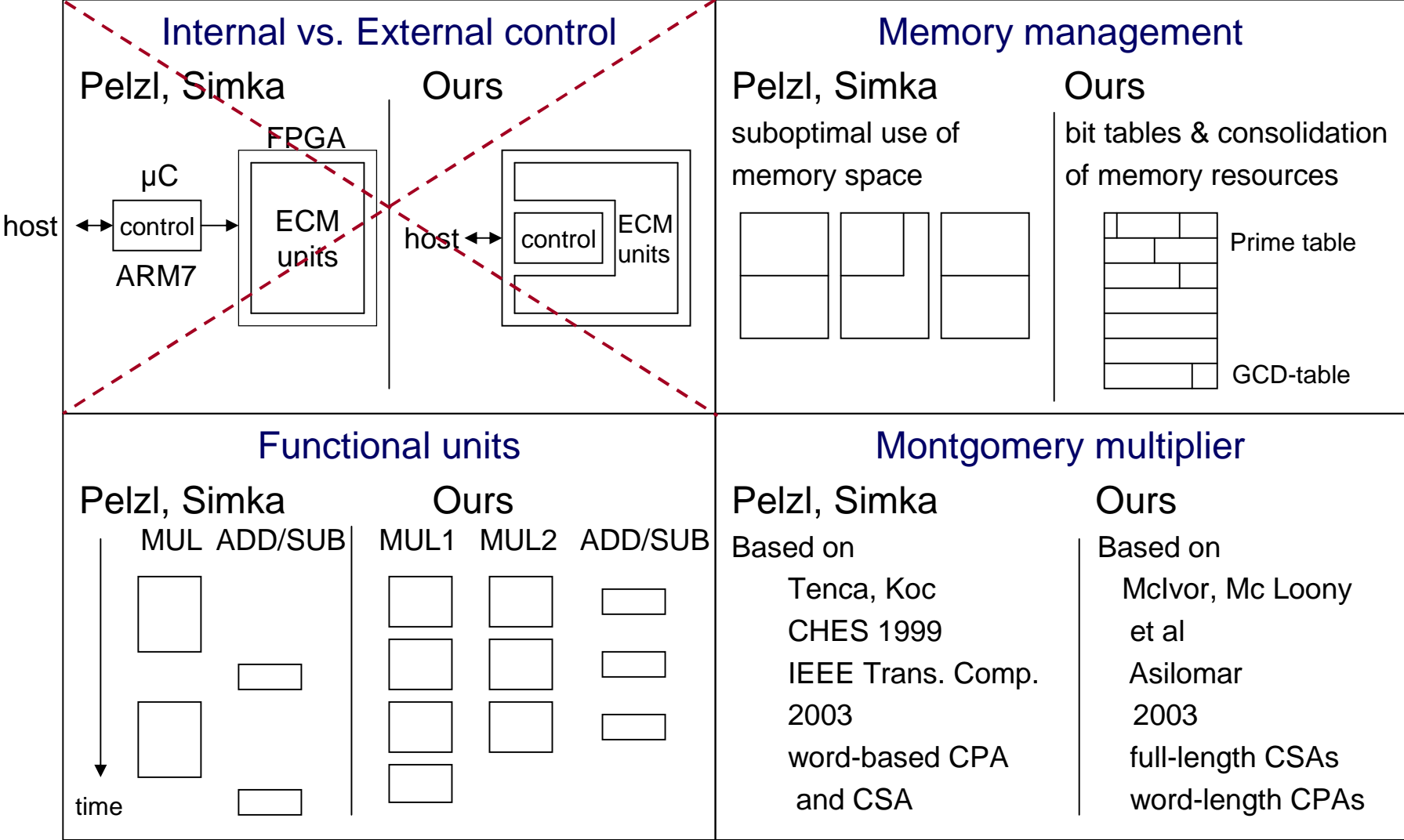


# Comparison with the Proof-of-Concept Design by Pelzl and Šimka Equalizing Measures

- Use the same FPGA device (Xilinx Virtex 2000E-6)
- Pelzl and Šimka design assumed to be redesigned to include an internal controller.

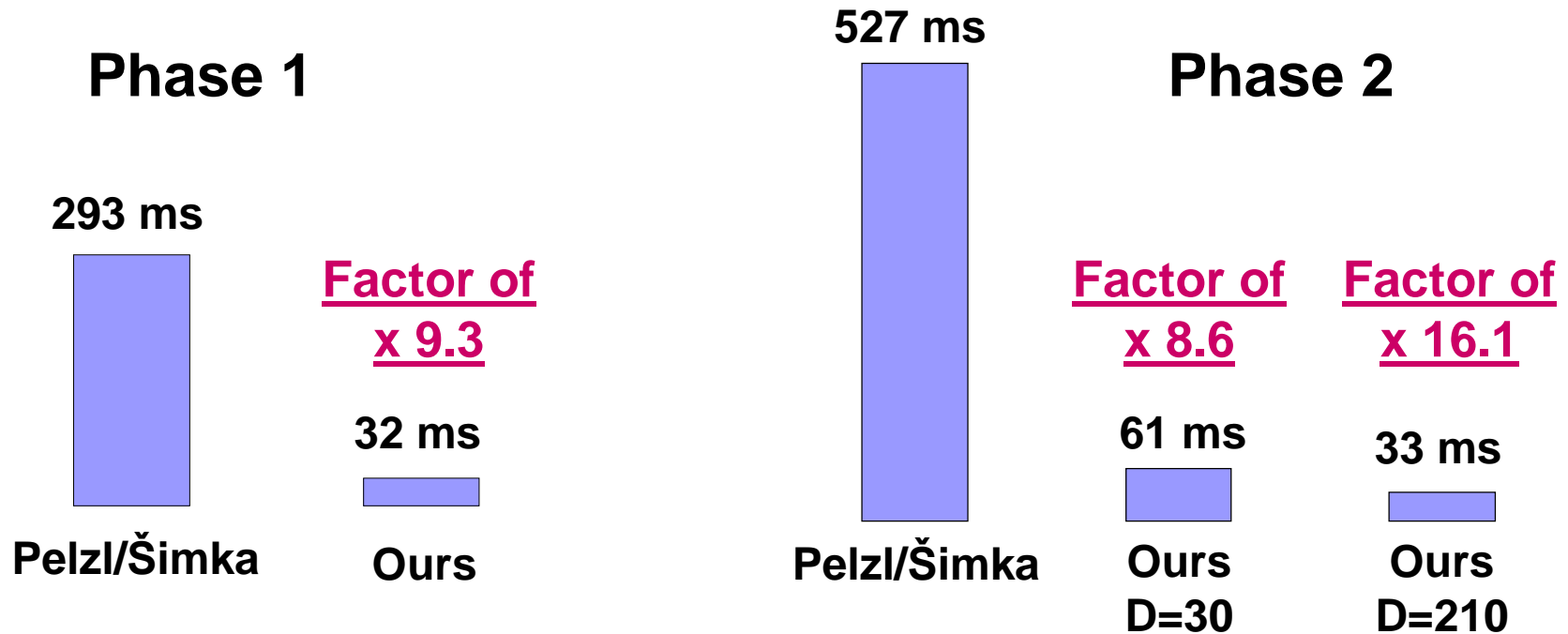
Execution times recalculated based on the limitations of the ECM unit only.

# Modifications compared to Pelzl, Simka



# Comparison with the Proof-of-Concept Design by Pelzl and Šimka

## Timing

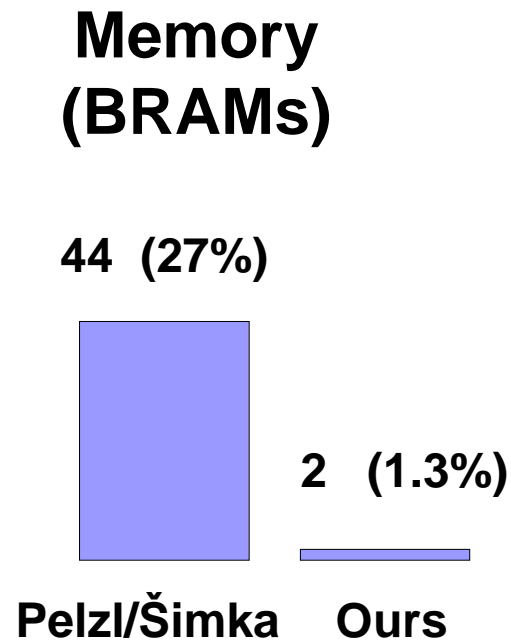


Major Contributors to the speed up:

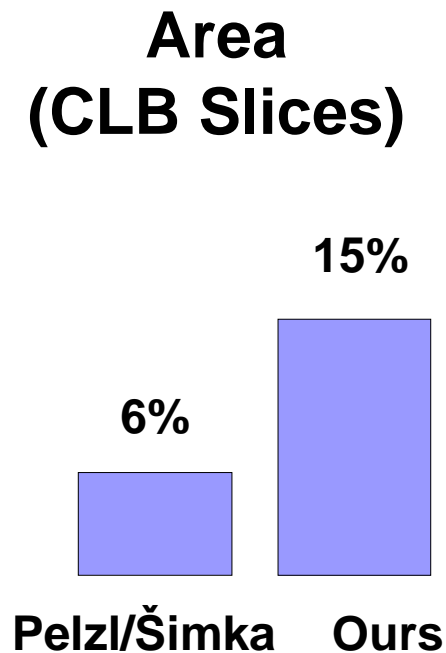
- Different design for the multiplier ( x 5 )
- Two multipliers working in parallel ( x 1.9 )
- Different D ( x 1.9 )

# Comparison with the Proof-of-Concept Design by Pelzl and Šimka

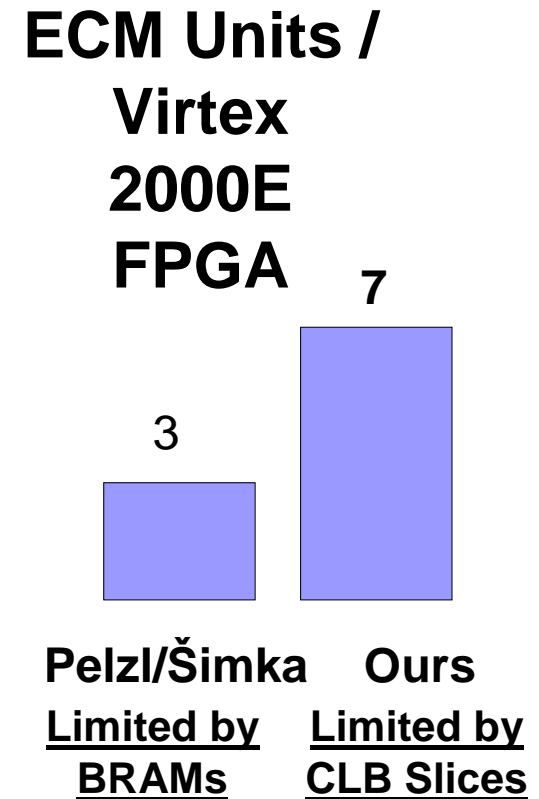
## Resources



Factor of  
x 22

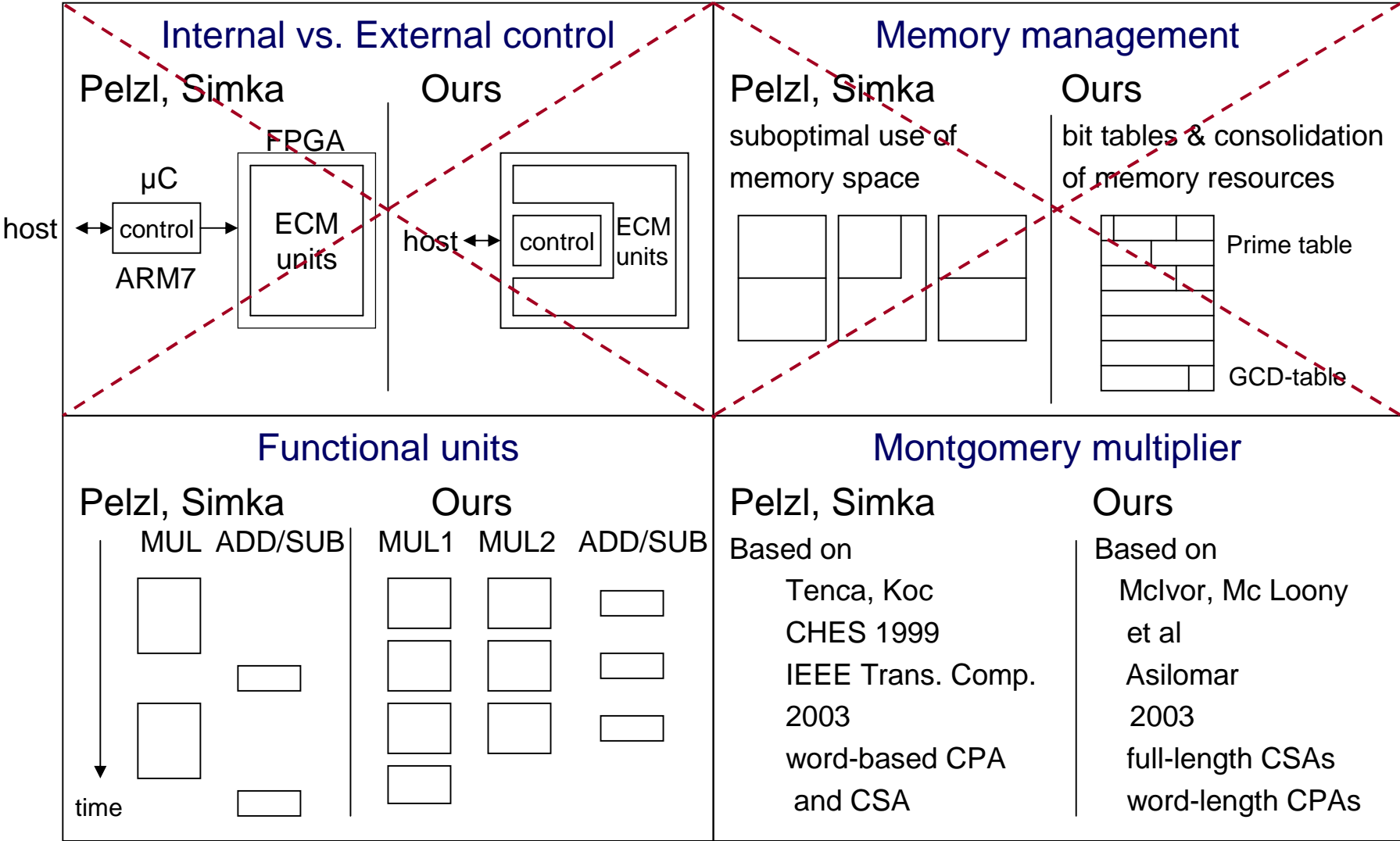


Factor of  
x 2.5



Factor of  
x 2.33

# Modifications compared to Pelzl, Simka



# Comparison with the Proof-of-Concept Design by Pelzl and Šimka

## Time x Area Product

Assuming the same memory management  
(i.e., improved memory management in Pelzl/Šimka):

### Improvement

Phase 1

x 3.7

Phase 2

x 6.4

**COMPARISON  
AMONG  
TECHNOLOGIES**



# Families of Xilinx FPGA Devices

## Low-cost

## High-performance

Spartan 3  
( $< \$130^*$ )

Virtex II  
( $< \$2,700^*$ )

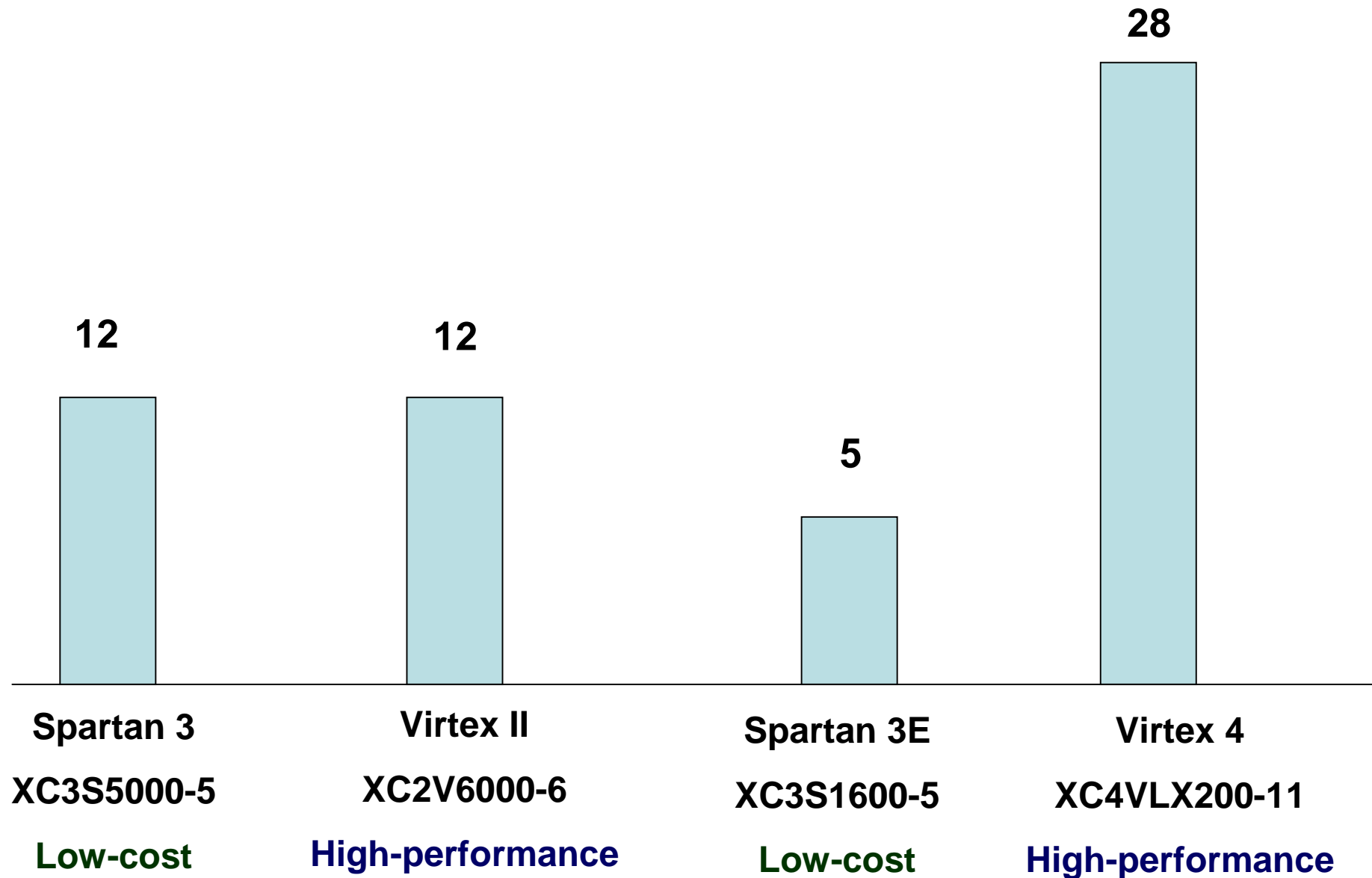
---

Spartan 3E  
( $< \$35^*$ )

Virtex 4  
( $< \$3,000^*$ )

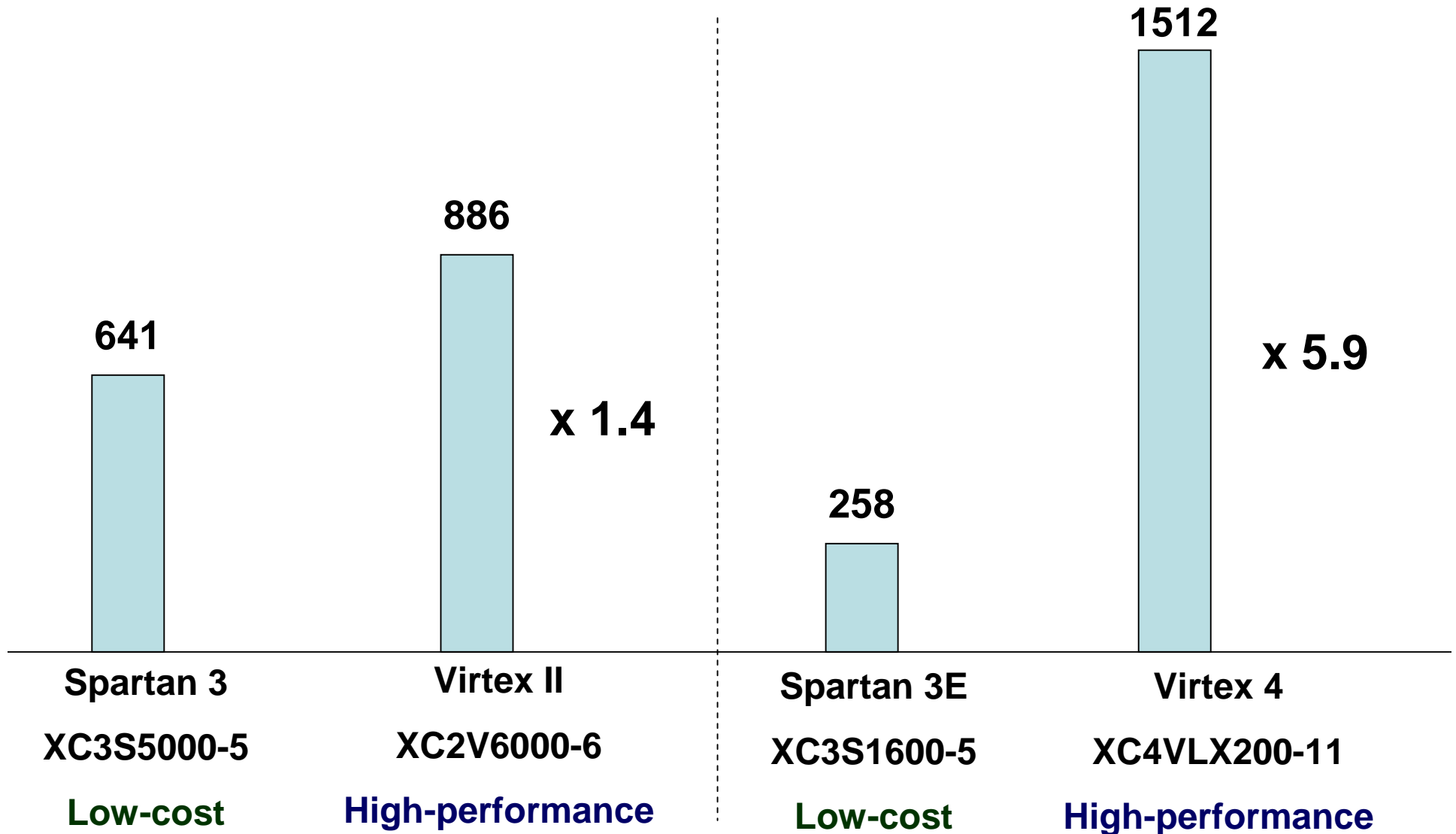
- approximate cost of the largest device per unit for a batch of 10,000 units

# Number of ECM units per FPGA device



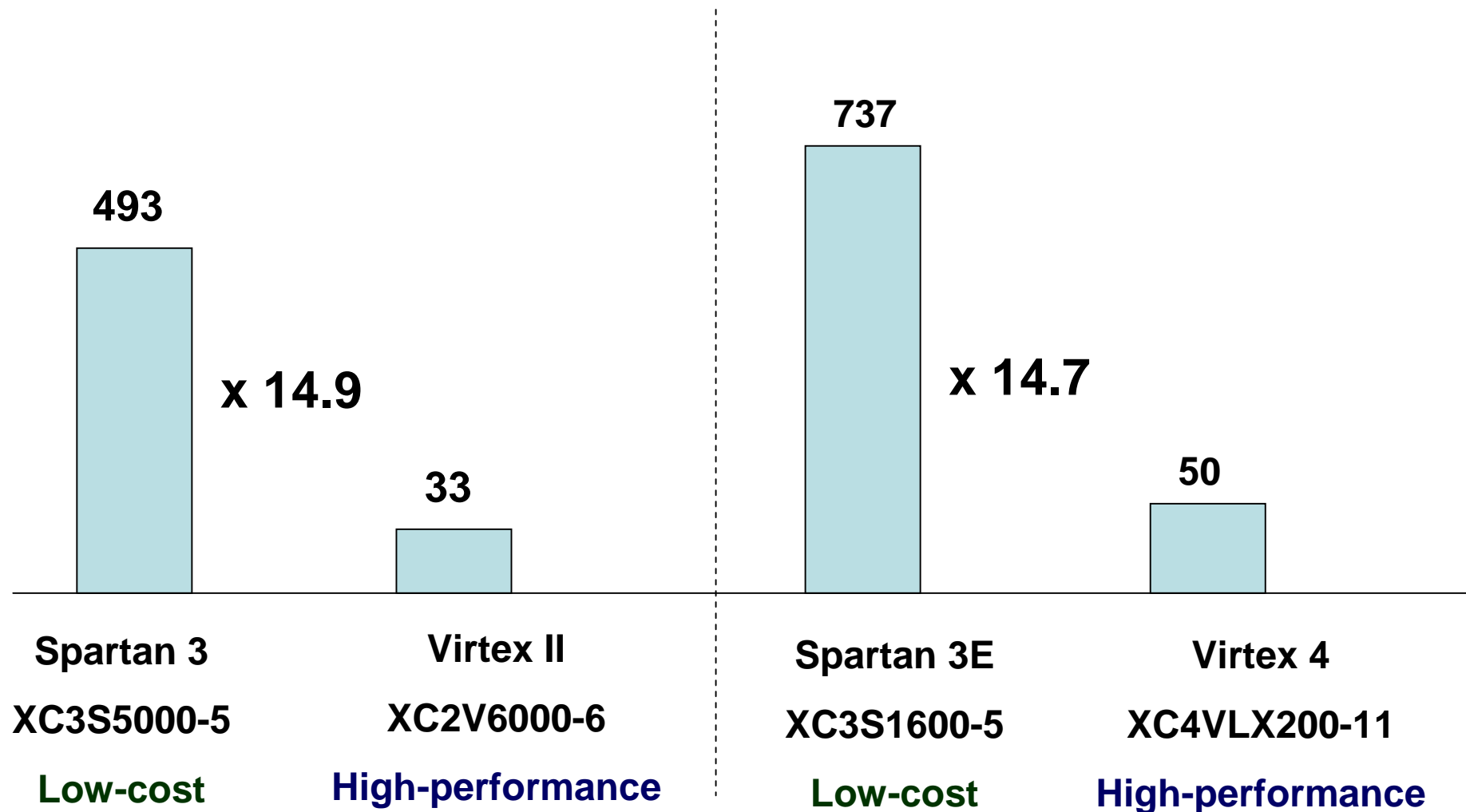
# Performance

## Number of Phase 1 operations per second



# Performance to cost ratio

## Number of Phase 1 operations per second per \$100



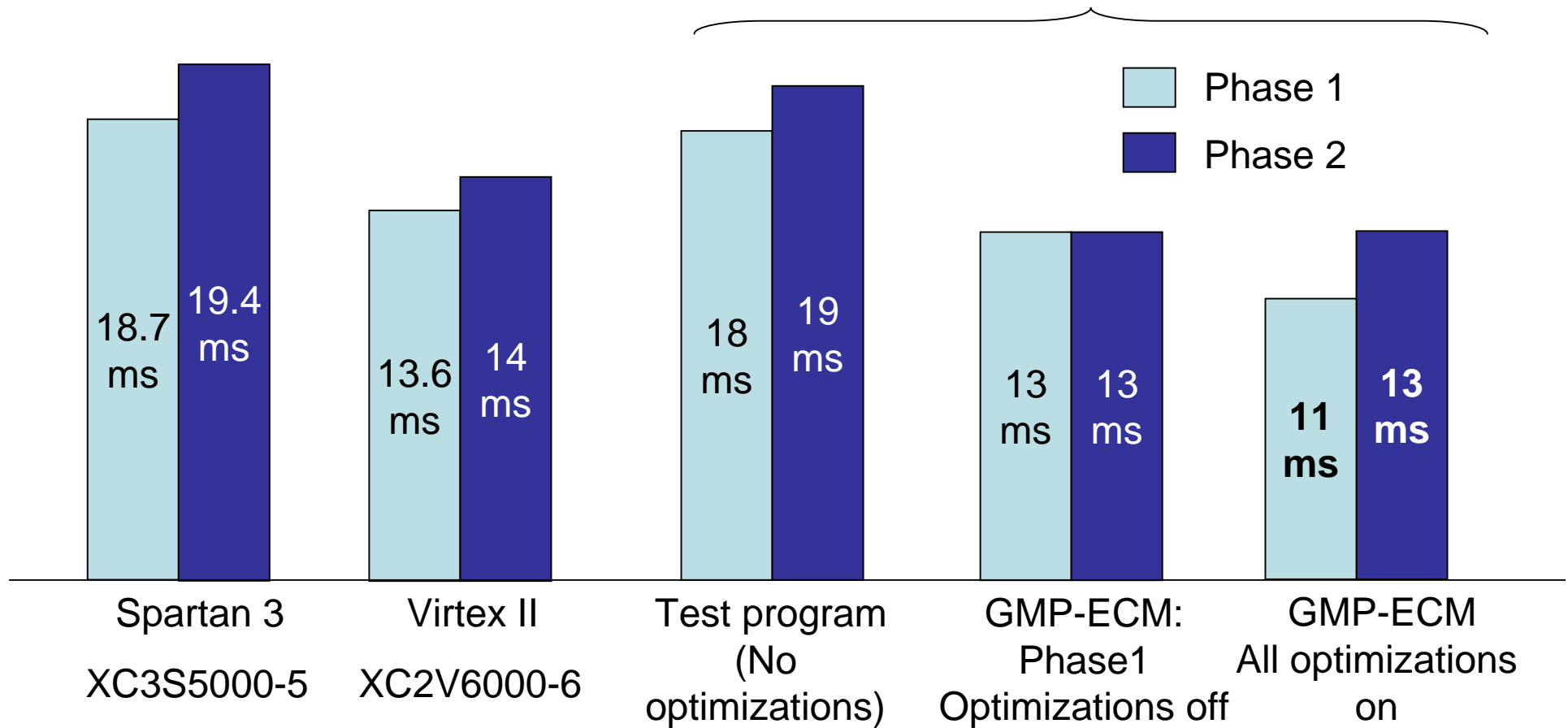
# Software Implementation

## GMP-ECM running on Pentium4 Xeon 2.8GHz

	Phase 1	Phase 2
Elliptic Curve	Montgomery form: $by^2z = x^3 + ax^2z + xz^2$	Weierstrass form: $Y^2 = X^3 + AX + B$
Coordinate	Projective	Affine
Optimization Techniques (Reducing Time)	Lucas chain (PRAC algorithm)	Fast polynomial multiplication Montgomery's $D_1D_2$ method
Optimization Techniques (Increasing Probability)		Brent-Suyama extension
Porting Optimizations To Hardware	Possible with pre-computations in software	Inverter required. Large amounts of memory required

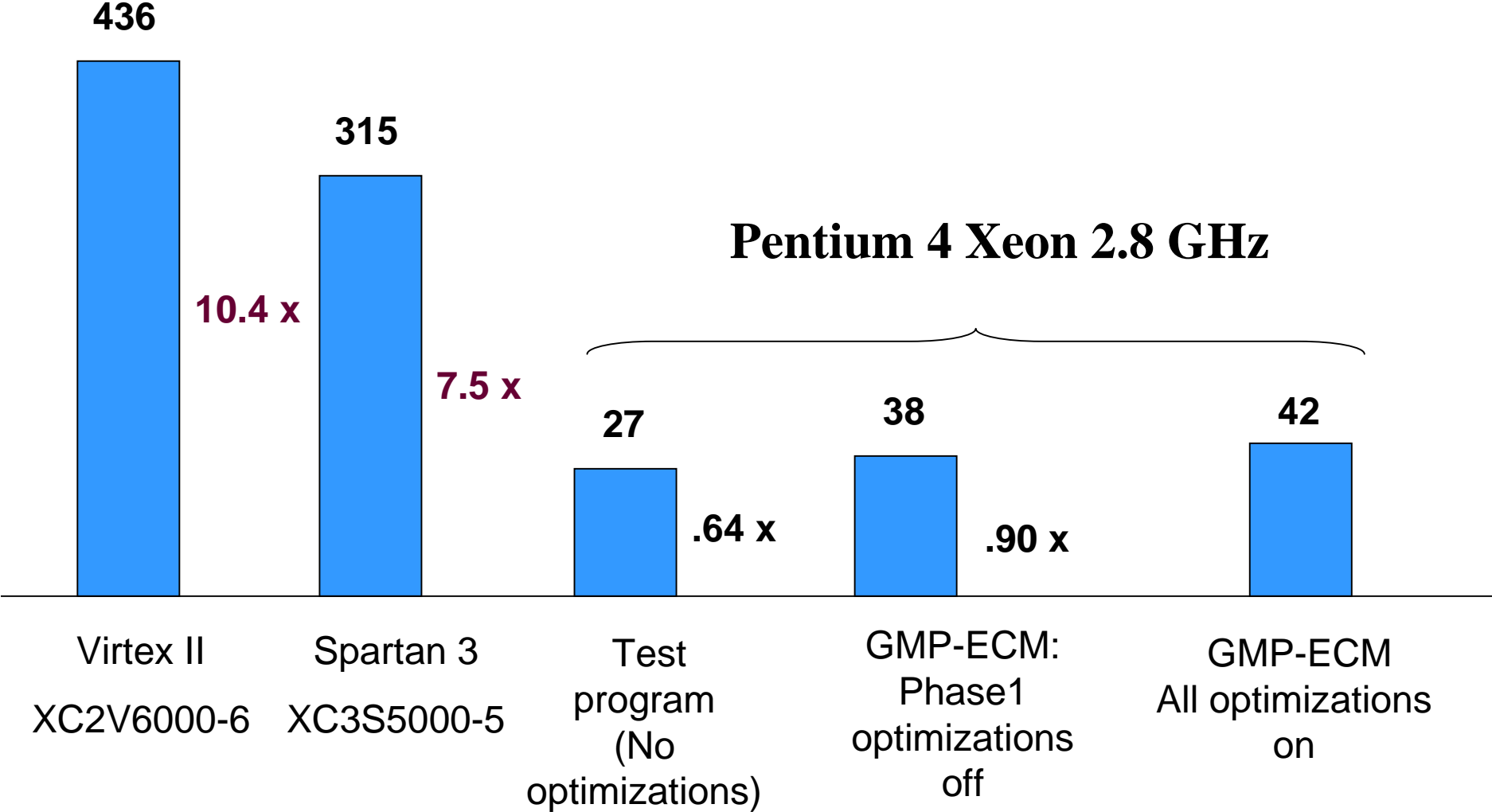
# FPGAs vs. Microprocessor Execution Time

Pentium 4 Xeon 2.8 GHz



# FPGAs vs Microprocessors

## # Phase 1 & Phase 2 computations per second



# ASIC vs. FPGA

- 3–4 times improvement in clock frequency  
(~300 MHz vs. 80-100 MHz)
- 8–10 times improvement in circuit area  
(# ECM units per device (96-120 vs. 12) )
- 2–3 times improvement in recurring fabrication & testing cost  
(~\$40 vs. \$130)

**~50-120 times improvement in performance to recurring  
cost ratio,**

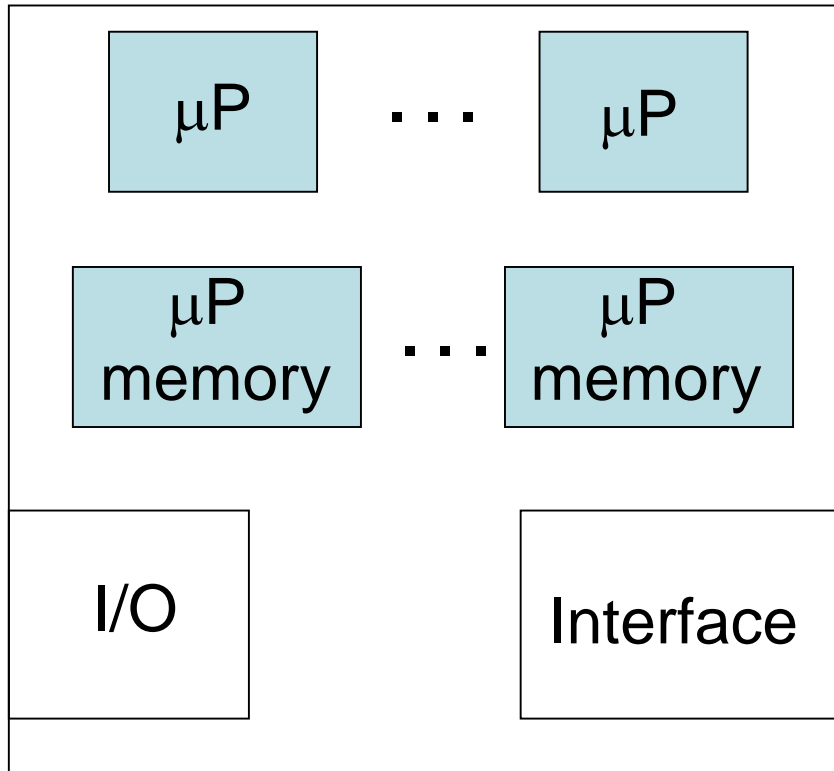
**but about \$1,000,000 of one-time non-recurring costs  
needed for the back-end design & preparation of masks  
for fabrication**



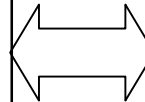
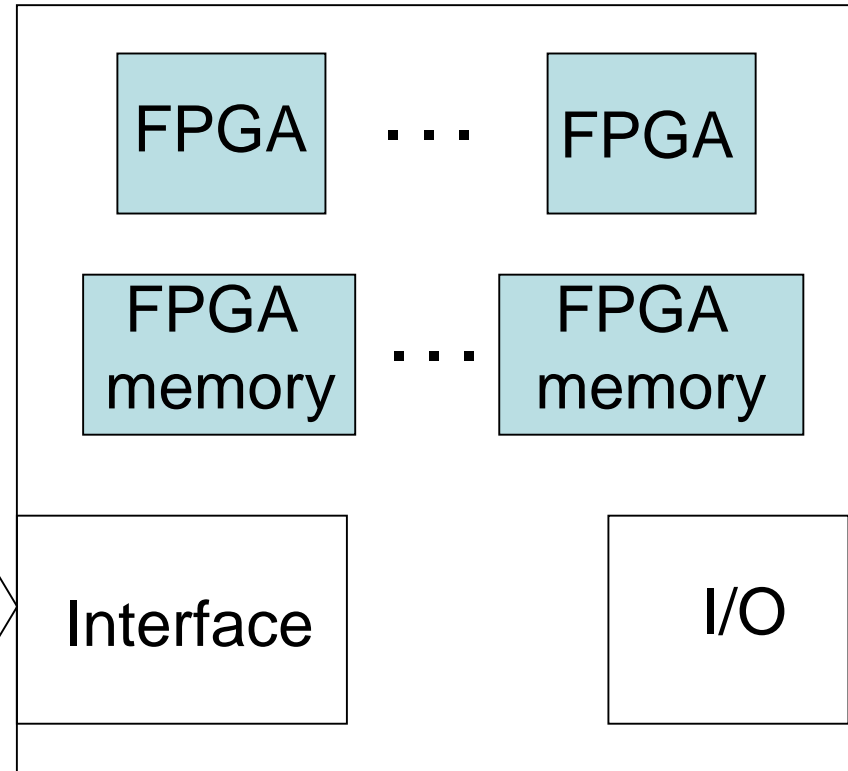
# **RCM 101 & Future work**

# What is a Reconfigurable Computing Machine (RCM)?

## Microprocessor system



## Reconfigurable system



# Prototyping with a general-purpose reconfigurable computing machine

- + can be programmed using high-level programming languages, such as C, by mathematicians & cryptographers themselves
  - + facilitates hardware/software co-design
  - + shortens development time, encourages experimentation and complex optimizations
  - + allows sharing costs among users of various applications
- 
- high entry cost (~\$100,000)
  - hardware aware programming
  - limited portability
  - limited availability of libraries
  - limited maturity of tools.

# Most advanced reconfigurable computing machines currently on the market

Machine	Released
<b>SRC 6</b> from SRC Computers	2002
<b>Cray XD1</b> from Cray	2005
<b>SGI Altix</b> from SGI	2005
<b>SRC 7</b> from SRC Computers, Inc,	2006



# Two major high-level language (HLL) programming models

**SRC 6 & SRC 7** from  
SRC Computers

**SRC MAP C programming model**

---

**Cray XD1** from  
from Cray

**SGI Altix** from  
SGI

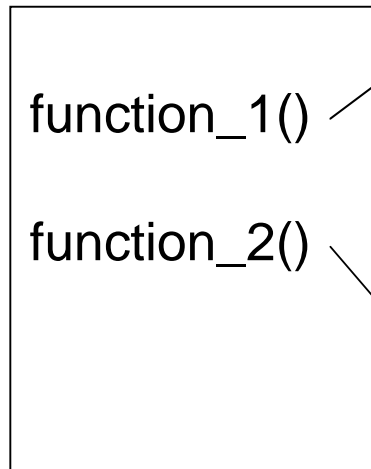
**Mitrion-C programming model**

# SRC Programming Model

**Microprocessor**

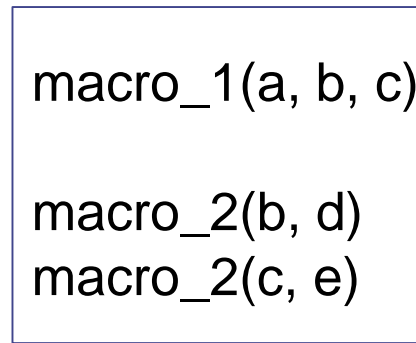
**FPGA**

main.c

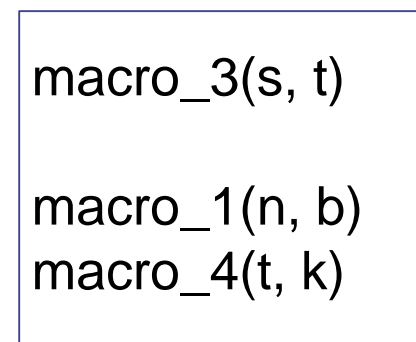


**ANSI C**

function\_1

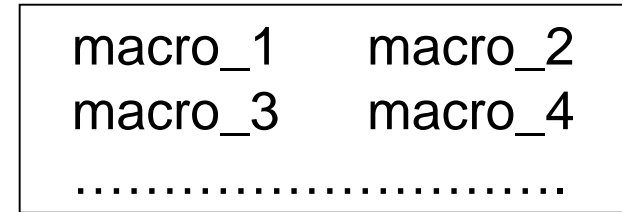


function\_2



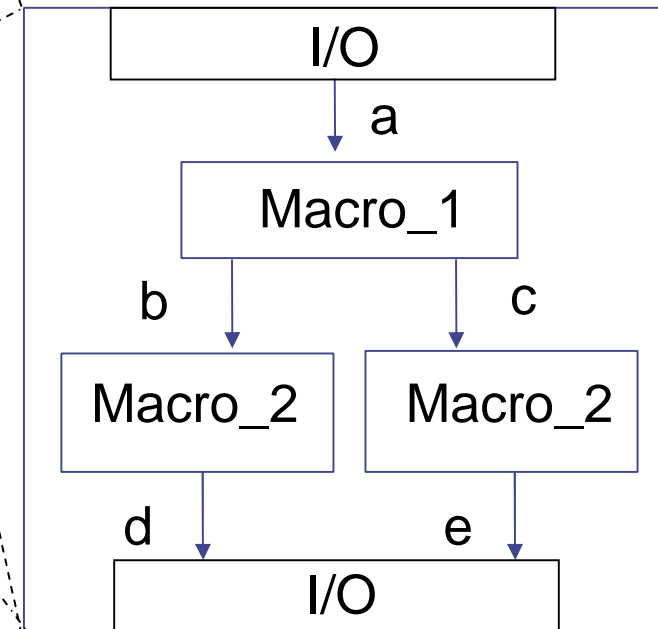
**MAP C**  
**(subset of ANSI C)**

Libraries of macros



**VHDL**

FPGA



# SRC Libraries of Hardware Macros

## Vendor libraries of hardware macros

- basic integer and floating-point arithmetic
- digital signal processing

## User libraries of hardware macros

developed by GWU/GMU/USC 2002-2006

- Secret-key cipher encryption & breaking
  - Binary Galois Field arithmetic  
(polynomial basis & normal basis representation)
  - Elliptic Curve Arithmetic
  - Long integer modular arithmetic (RSA)
  - Sorting
- 
- Image processing
  - Bioinformatics

# SRC Programming Environment

- + very easy to learn and use
  - + standard ANSI C
  - + hides implementation details
  - + very well integrated environment
  - + mature - in production use for over 3 years with constant improvements
- 
- subset of C
  - legacy C code requires rewriting
  - C limitations in describing HW (paralellism, data types)
  - closed environment, limited portability of code to HW platforms other than SRC



# Mitron-C Programming Model for Cray & SGI

**Microprocessor**

**FPGA**

main.c

```
function_1(in1)
start_fpga()

function_1(in2)
start_fpga()
```

**ANSI C  
based on  
Mitron  
API**

Application  
code  
(platform  
independent)

**Mitron-C**

Mitron Distributed  
Processor Architecture  
(platform dependent)

**VHDL**

Mitron Compiler  
& Configurator

**FPGA**

RAM

Input &  
output

I/O

**application  
on the  
distributed  
processor**

# Mitrion-C

high-level parallel programming language  
for high-performance reconfigurable computers

+ arbitrary sizes of operands

e.g.,

int:24

uint:256

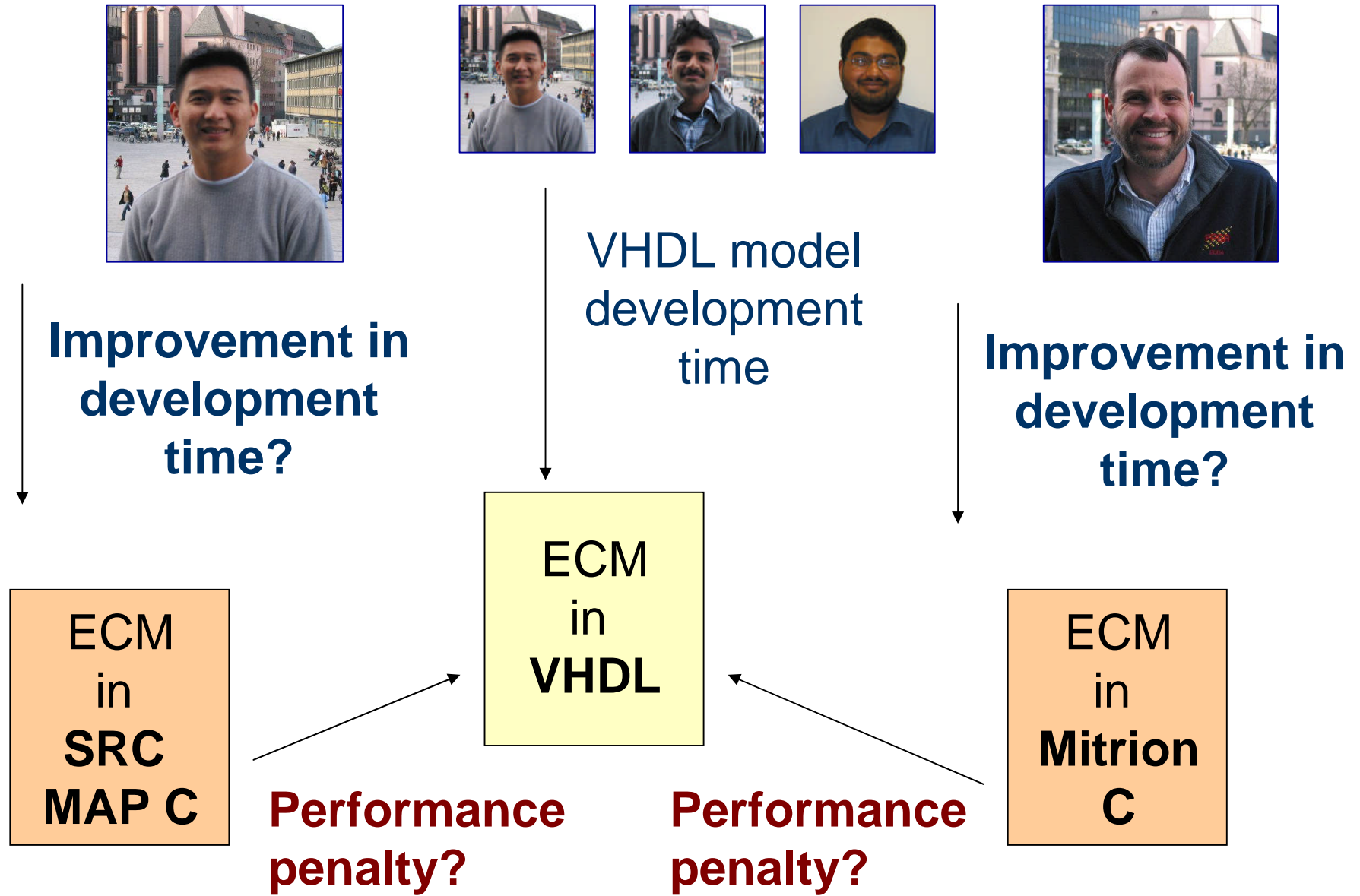
+ parallel execution of instructions limited only by  
data dependencies and limited available area

+ focus on the area \* time space utilization

# Mitrion-C Environment for Cray and SGI

- + easy to learn by high-performance computing (HPC) programmers
- + small amount of Mitrion-C generates large number of lines of HDL code
- + suitable for describing classical complex HPC problems
- + portable application codes
  
- new and yet untested
- non-standard, no support for legacy codes
- language describes only what happens in a single FPGA
- currently, no mechanisms to use HDL macros

# Our near-future experiment



# **Summary & conclusions**

# Summary

## ECM case study

**Hardware implementations provide  
a substantial improvement  
vs. optimized software implementations  
in terms of the performance to cost ratio**

- **low-cost FPGAs vs. microprocessors** > 10 x
- **ASICs vs. low-cost FPGAs** ~ 50-120  
(only for high-volume production > 10,000 chips)

# Conclusions

**Best environment for prototyping  
of hardware implementations of codebreakers**

- **general-purpose reconfigurable computers**

**Best environment for the final design  
of the cost-optimized cipher breaker**

- **special-purpose machines based on**
  - **low-cost FPGAs**  
**(or ASICs \$\$\$)**

# Thank you!



Questions???