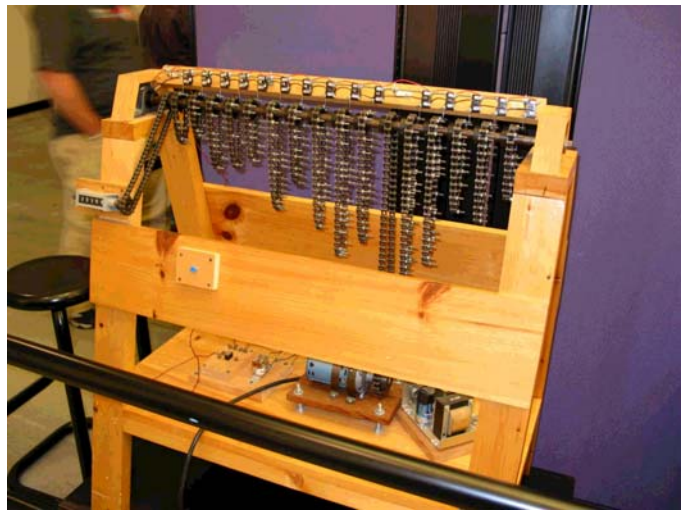
Implementation of the “rho”, p-1 & the Elliptic Curve Methods of Factoring in Reconfigurable Hardware

Kris Gaj
Patrick Baier
Soonhak Kwon

Hoang Le
Ramakrishna Bachimanchi
Khaleeluddin Mohammed

Paul Kohlbrenner
Viswanadham Sanku

George Mason University



GMU Team

Computer Engineer / Cryptographer

Kris Gaj

Ph.D in Electrical Engineering,
Warsaw University of Technology, Poland
Associate Professor at George Mason University

Mathematicians/ Cryptographers

Soonhak Kwon

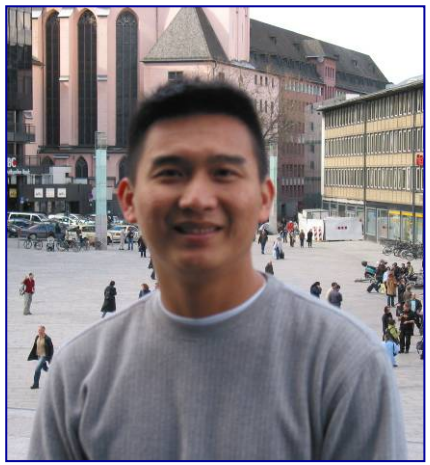
Ph.D in Mathematics,
Johns Hopkins University
Maryland, U.S
Visiting professor at GMU on leave from Sungkyunkwan University, Suwon, Korea

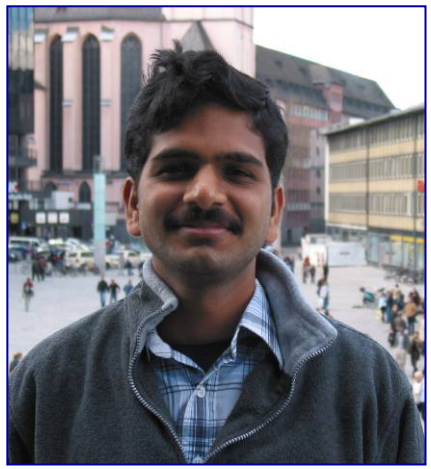
Patrick Baier

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

GMU Team

Hardware design





Hoang Le

Ramakrishna Bachimanchi

Khaleeluddin Mohammed

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

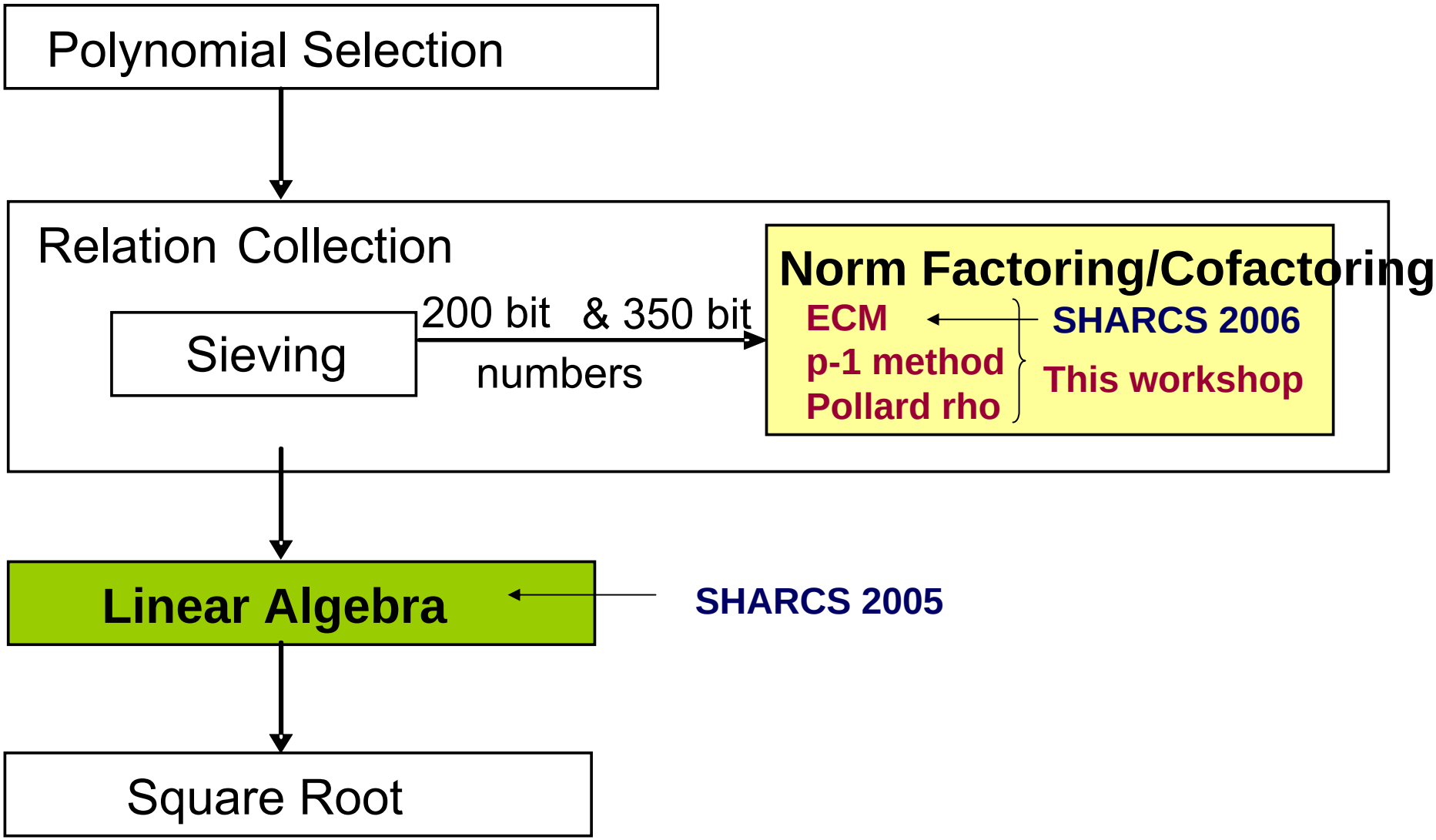
GMU Team

Software experiments

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

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

Factoring 1024-bit RSA keys using Number Field Sieve (NFS)



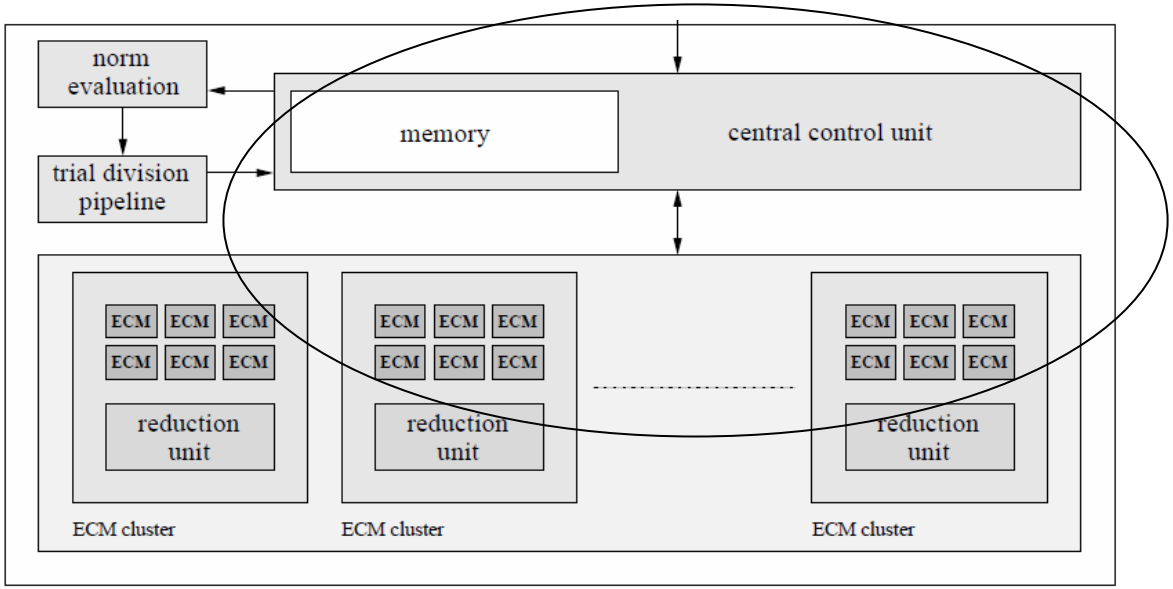
ECM in Hardware

Previous Proof-of-Concept Design

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

Wednesday's talk

Rainer Steinwandt, Willi Geiselmann, Fabian Januszewski, Hubert Köpfer and Jan Pelzl, "Another Attempt to Sieve With Small Chips— Part II: Norm Factorization"

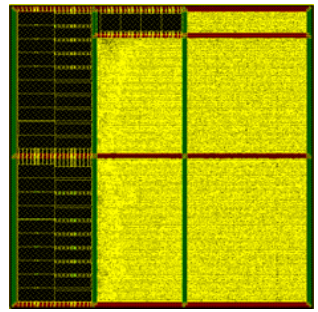


Our contribution:

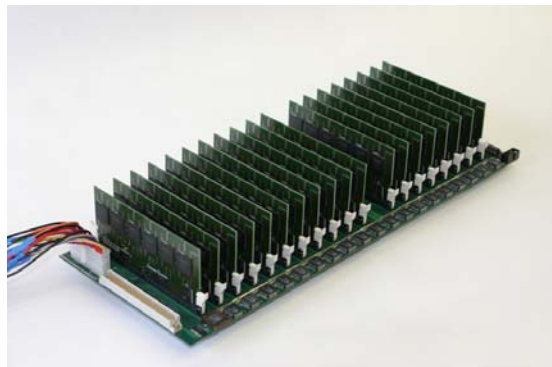
- Independent proposal of a similar architecture
- Software/hardware partitioning
- Detailed design of major units in VHDL
- Verification and experimental testing
- Detailed estimates of speed and area for FPGAs and ASICs

Comparison among technologies

Microprocessors **FPGAs** **ASICs**



SRC



COPACOBANA

FACTORING 101

Special-purpose factoring algorithms

Trial Division
Pollard's rho Method
Pollard's p-1 Method
Elliptic Curve Method

- Typically of exponential complexity as a function of N
- Often superior to more advanced (sub-exponential) methods for finding small factors
- Using more advanced methods to find small divisors is not a good use of resources
- Other special-purpose algorithms, such as
  - Lehman's method,
  - Shanks's Class Group Method
  - Shanks's SQUFOF Method

  less efficient, and thus of historical interest only.

Pollard's Rho Method

Birthday paradox: If more than 23 "random" people are in a room (or even if they aren't) there is a more than 50% probability that the birthdays of two of them fall on the same day of the year.

Pollard's rho method - Example

$N = 97 \cdot 1889 = 183\,233$

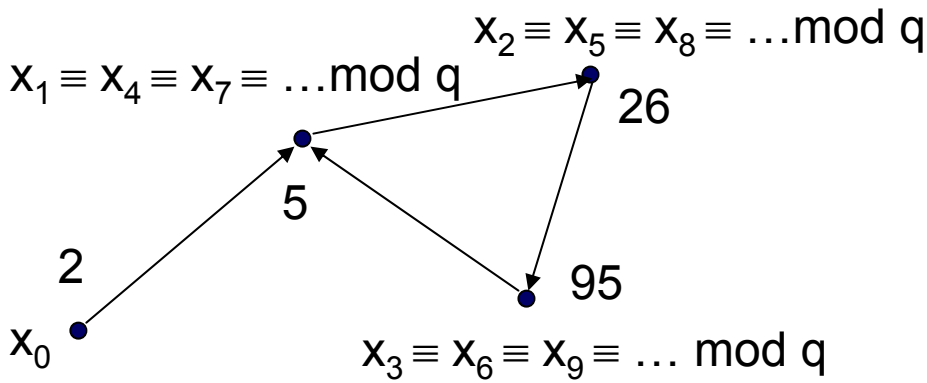
$$x_{i+1} = x_i^2 + 1 \bmod N$$

$$x_0 \to x_1 \to x_2 \to x_3 \to x_4 \to x_5 \to x_6 \to x_7 \to x_8 \to x_9 \ldots$$

$$2 \to 5 \to 26 \to 677 \to 91864 \to 15449 \to 102236 \to 39678 \to 5749 \to 69062 \ldots$$

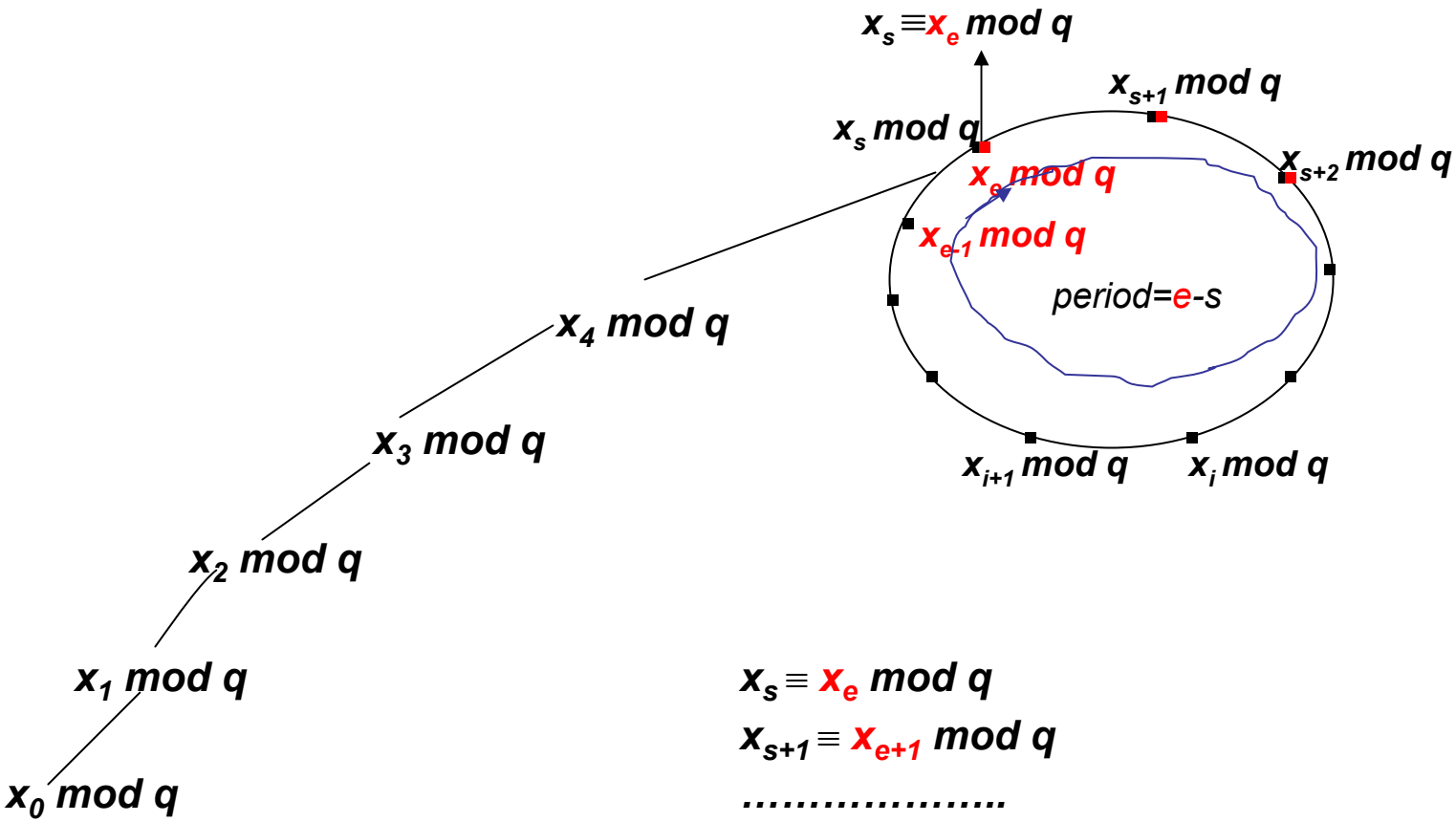
mod 97:

$$2 \to 5 \to 26 \to 95 \to 5 \to 26 \to 95 \to 5 \to 26 \to 95 \ldots$$



$x_1 \equiv x_4 \bmod q$
$q \mid (x_1 - x_4)$
$q \mid N$
$q \mid \gcd(x_1 - x_4, N)$
$q = \gcd(-91\,859, 183\,233)$
$= 97$

Pollard's Rho Method



$$x_s \equiv x_e \bmod q$$
$$x_{s+1} \equiv x_{e+1} \bmod q$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$x_{s+k} \equiv x_{e+k} \bmod q$$

Rho Method - Floyd's Version

x_1-x_2 x_1-x_3 x_1-x_4 x_1-x_5 x_1-x_6 ---------- x_1-x_i

x_2-x_3 x_2-x_4 x_2-x_5 x_2-x_6 x_2-x_7 ---------- x_2-x_i

x_3-x_4 x_3-x_5 x_3-x_6 x_3-x_7 x_3-x_8 ---------- x_3-x_i

x_4-x_5 x_4-x_6 x_4-x_7 x_4-x_8 x_4-x_9 ---------- x_4-x_i

x_5-x_6 x_5-x_7 x_5-x_8 x_5-x_9 x_5-x_{10} ---------- x_5-x_i

x_6-x_7 x_6-x_8 x_6-x_9 x_6-x_{10} x_6-x_{11} x_6-x_{12} ---------- x_6-x_i

x_7-x_8 x_7-x_9 x_7-x_{10} x_7-x_{11} x_7-x_{12} x_7-x_{13} x_7-x_{14} ---------- x_7-x_i

x_8-x_9 x_8-x_{10} x_8-x_{11} x_8-x_{12} x_8-x_{13} x_8-x_{14} x_8-x_{15} x_8-x_{16} ---------- x_8-x_i

---

x_k-x_{k+1} x_k-x_{k+2} x_k-x_{k+3} ---------- x_k-x_{2k} -------- x_k-x_i

Pollard's Rho Algorithm - Floyd's Version

$f(x)=x^2+a$ with $a \notin \{-2,0\}$

iterations $t < 100\sqrt{q_{max}}$ (q_{max} is the maximum factor we expect to find using rho method)

We choose random x_0 in the range(0,N-1) and $x_1=f(x_0)$

	x_0	
	↓	$d=1$
x_2 ←	x_1	$d=d*(x_2-x_1)$
↓$f(f())$	↓$f()$	
x_4	x_2	$d=d*(x_4-x_2)$
↓	↓	
x_6	x_3	$d=d*(x_6-x_3)$
x_t	$x_{t/2}$	$d=d*(x_t-x_{t/2})$
↓	↓	
x_{t+2}	$x_{(t+2)/2}$	$d=d*(x_{t+2}-x_{(t+2)/2})$
x_{2i}	x_i	$d=d*(x_{2i}-x_i)$
↓	↓	
$x_{2(i+1)}$	x_{i+1}	$d=d*(x_{2i+2}-x_{i+1})$
x_{2t}	x_t	$d=d*(x_{2t}-x_t)$
$*x_{2i+2}=f(f(x_{2i})), x_{i+1}=f(x_i)$		$q=gcd(d,N)$

Can be eliminated

Minimization for area and/or memory

Rho Method - Brent's Version

x_1-x_2 x_1-x_3 x_1-x_4 x_1-x_5 x_1-x_6 ---------- x_1-x_i

x_2-x_3 x_2-x_4 x_2-x_5 x_2-x_6 x_2-x_7 ---------- x_2-x_i

x_3-x_4 x_3-x_5 x_3-x_6 x_3-x_7 x_3-x_8 ---------- x_3-x_i

x_4-x_5 x_4-x_6 x_4-x_7 x_4-x_8 x_4-x_9 ---------- x_4-x_i

x_5-x_6 x_5-x_7 x_5-x_8 x_5-x_9 x_5-x_{10} ---------- x_5-x_i

x_6-x_7 x_6-x_8 x_6-x_9 x_6-x_{10} x_6-x_{11} x_6-x_{12} ---------- x_6-x_i

x_7-x_8 x_7-x_9 x_7-x_{10} x_7-x_{11} x_7-x_{12} x_7-x_{13} x_7-x_{14} ---------- x_7-x_i

x_8-x_9 x_8-x_{10} x_8-x_{11} x_8-x_{12} x_8-x_{13} x_8-x_{14} x_8-x_{15} x_8-x_{16} ---------- x_8-x_i

---

x_k-x_{k+1} x_k-x_{k+2} x_k-x_{k+3} ---------- x_2^k-$x_{2^k+2^{k-1}+1}$ ---------- x_2^k-x_2^{k+1}

Rho Method - Brent's Version Sequence of Operations

v_2	d	v_1
x_2	$d=1$	x_2
x_3		
x_4	$d=d*(x_4-x_2)$	x_4
x_5		
x_6		
x_7	$d*(x_7-x_4)$	
x_8	$d*(x_8-x_4)$	x_8
x_9		
x_{10}		
x_{11}		
x_{12}		
x_{13}	$d*(x_{13}-x_8)$	
x_{14}	$d*(x_{14}-x_8)$	
x_{15}	$d*(x_{15}-x_8)$	
x_{16}	$d*(x_{16}-x_8)$	x_{16}

Can be eliminated

Minimization for execution time 24%

p-1 algorithm

Inputs :

- N – number to be factored
- a – arbitrary integer such that gcd(a, N)=1
- B_1 – smoothness bound for Phase1
- B_2 – smoothness bound for Phase2

Outputs:

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

p-1 algorithm – Phase 1

precomputations

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

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

2: $q_0 \leftarrow a^k \bmod N$ *main computations*

3: $q \leftarrow \gcd(q_0 - 1, N)$ *postcomputations*

4: if $q > 1$

5: return q (factor of N)

6: else

7: go to Phase 2

8: end if

p-1 algorithm – Phase 2

09: $d \leftarrow 1$

10: for each prime $p = B_1$ to B_2 do

11: $d \leftarrow d \cdot ({q_0}^p - 1) \pmod{N}$ *main computations*

12: end for

13: $q \leftarrow \gcd(d,\ N)$

14: if $q > 1$ then *postcomputations*

15: return q

16: else

17: return FAIL

18: end if

p-1 Phase 1 – Numerical example

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

$a = 2$

$B_1 = 20$

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

$q_0 = a^k \bmod N = 2^{232\,792\,560} \bmod 1\,740\,719 = 1\,003\,058$

$q = \gcd(1\,003\,058 - 1;\ 1\,740\,719) =$ **1361**

Why did the method work?

$q-1 = 1360 = 2 \cdot 5 \cdot 17 \mid k$

$a^k \bmod q = a^{(q-1) \cdot m} \bmod q = 1$

$q \mid a^k - 1$

ECM Algorithm

Inputs :

- N – number to be factored
- E – elliptic curve
- P_0 – point on the 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

precomputations

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

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

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

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

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

main computations

14: $q \leftarrow \gcd(d, N)$

15: if $q > 1$ then

16: return q

17: else

18: return FAIL

19: end if

postcomputations

ECM Phase 1 – Numerical example

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

$E : y^2 = x^3 + 30x + 1 \pmod{1\,740\,719}$

$P_0 = (4 : : 1)$

$B_1 = 20$

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

$kP_0 = (256\,230 : \ : 1\,242\,593)$

gcd (1 242 593; 1 740 719) = **1361**

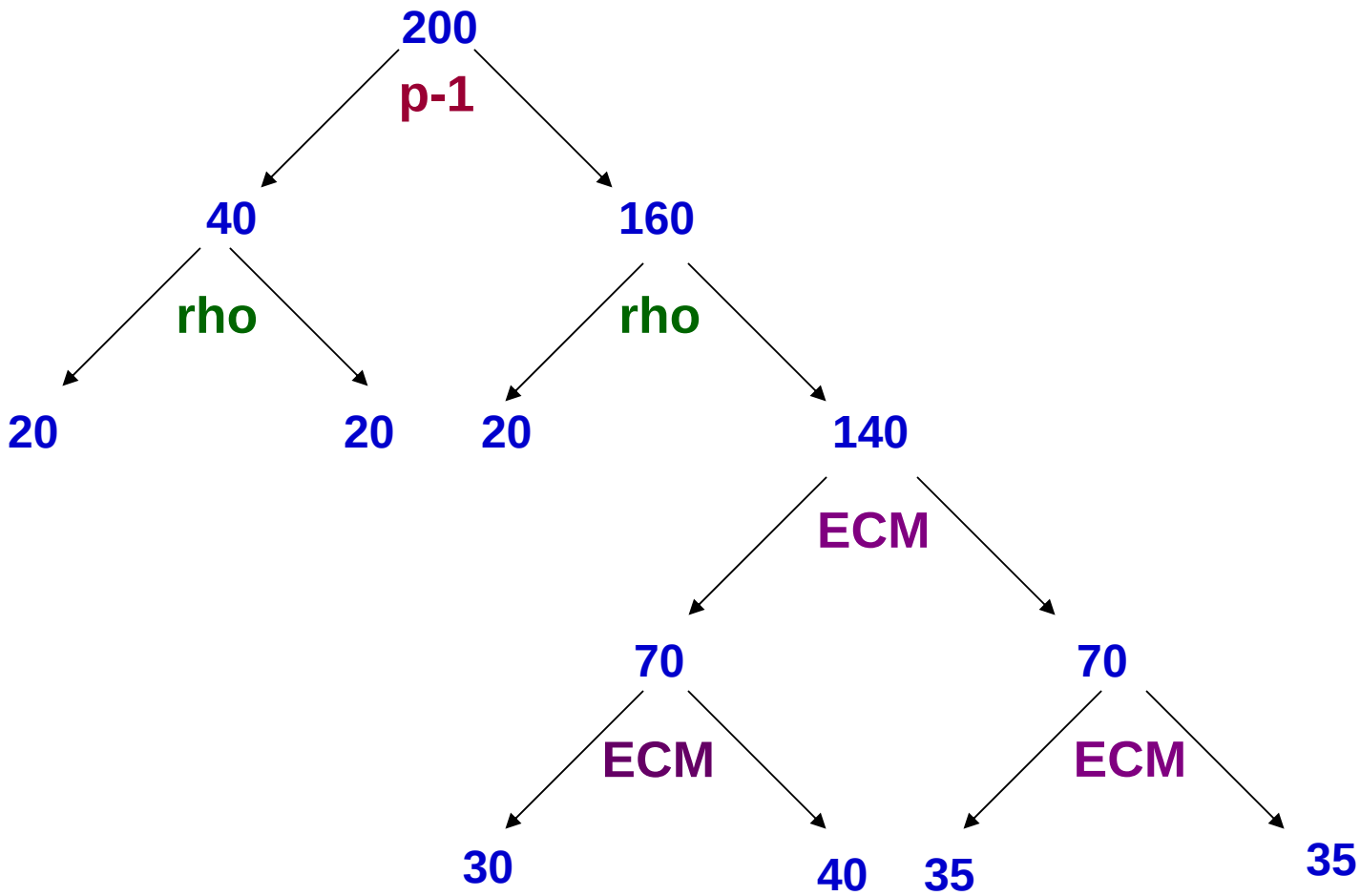
Why does the method work?

#E = 1326 = 2·3·13·17 | k (over GF(1361))

$kP_0 = m \cdot \#E \cdot P_0 = O = (0:1:0)$ (over GF(1361))

$z_{Q0} = 0 \bmod 1361$

Possible use of three special purpose factoring methods in NFS



Choice of parameters

ECM method

B1 ↗ B2 ↗

Probability of success ↗

Execution time ↗

Optimized analytically for the maximum ratio ⇨ $\frac{\text{Probability of success}}{\text{Execution time}}$

⇩

B1 = 960 **B2=57000**

---

p-1 method

The same as in ECM

---

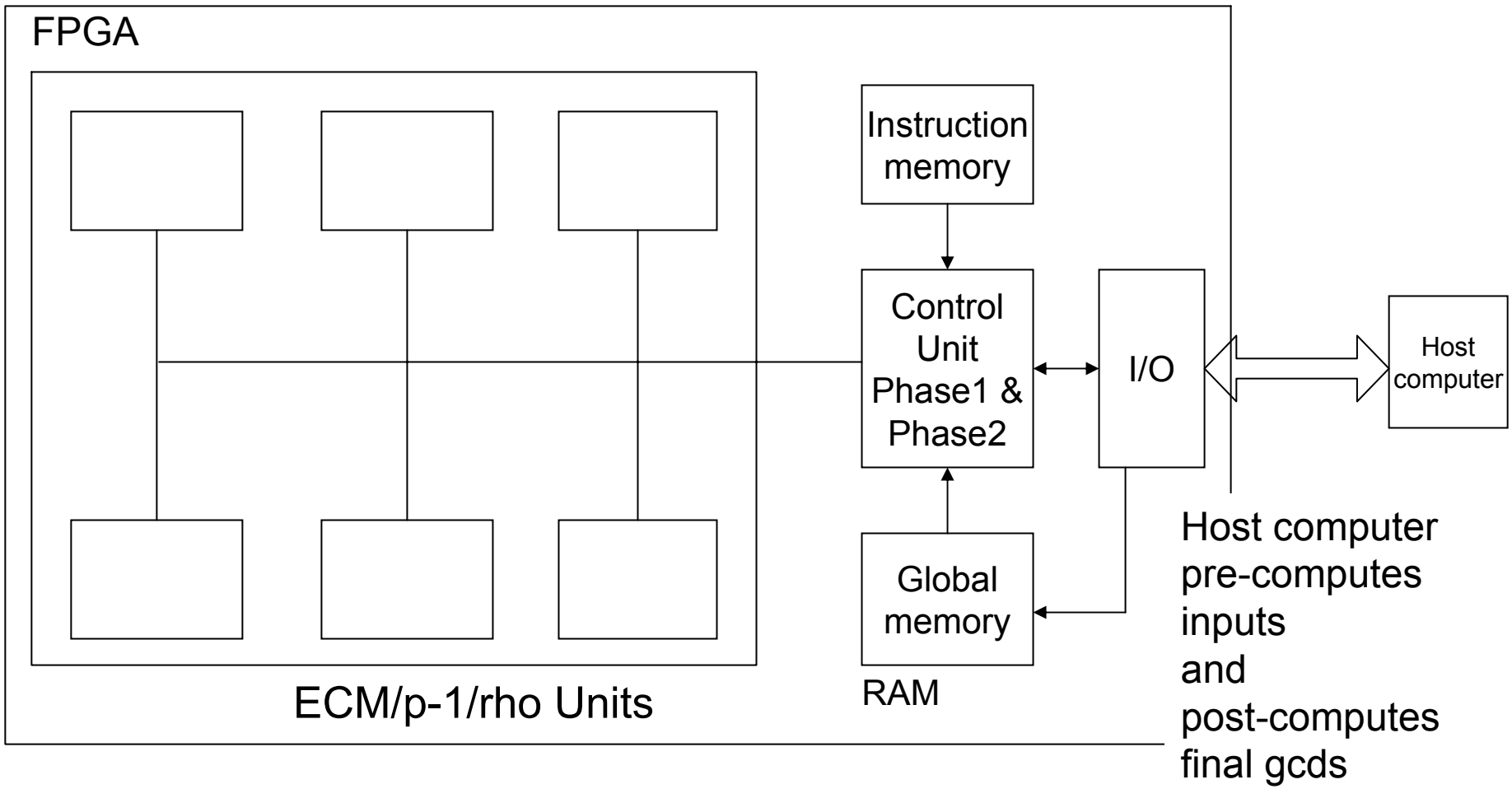
rho method

t = 8192

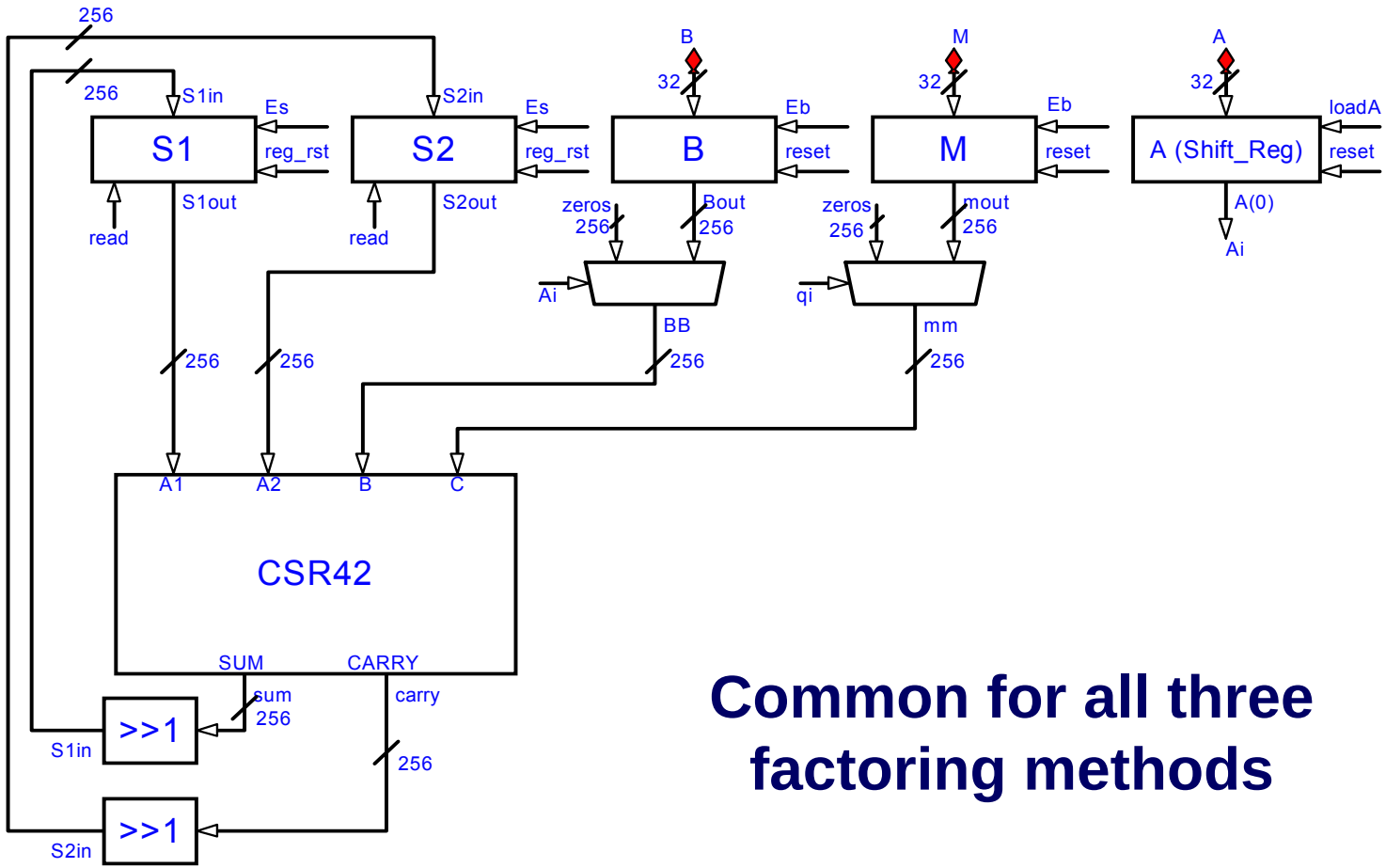
Approximately the same execution time as a single run of ECM.
Likely to find factors up to 2^{26}

ECM IN HARDWARE

Our architecture : Top-level view

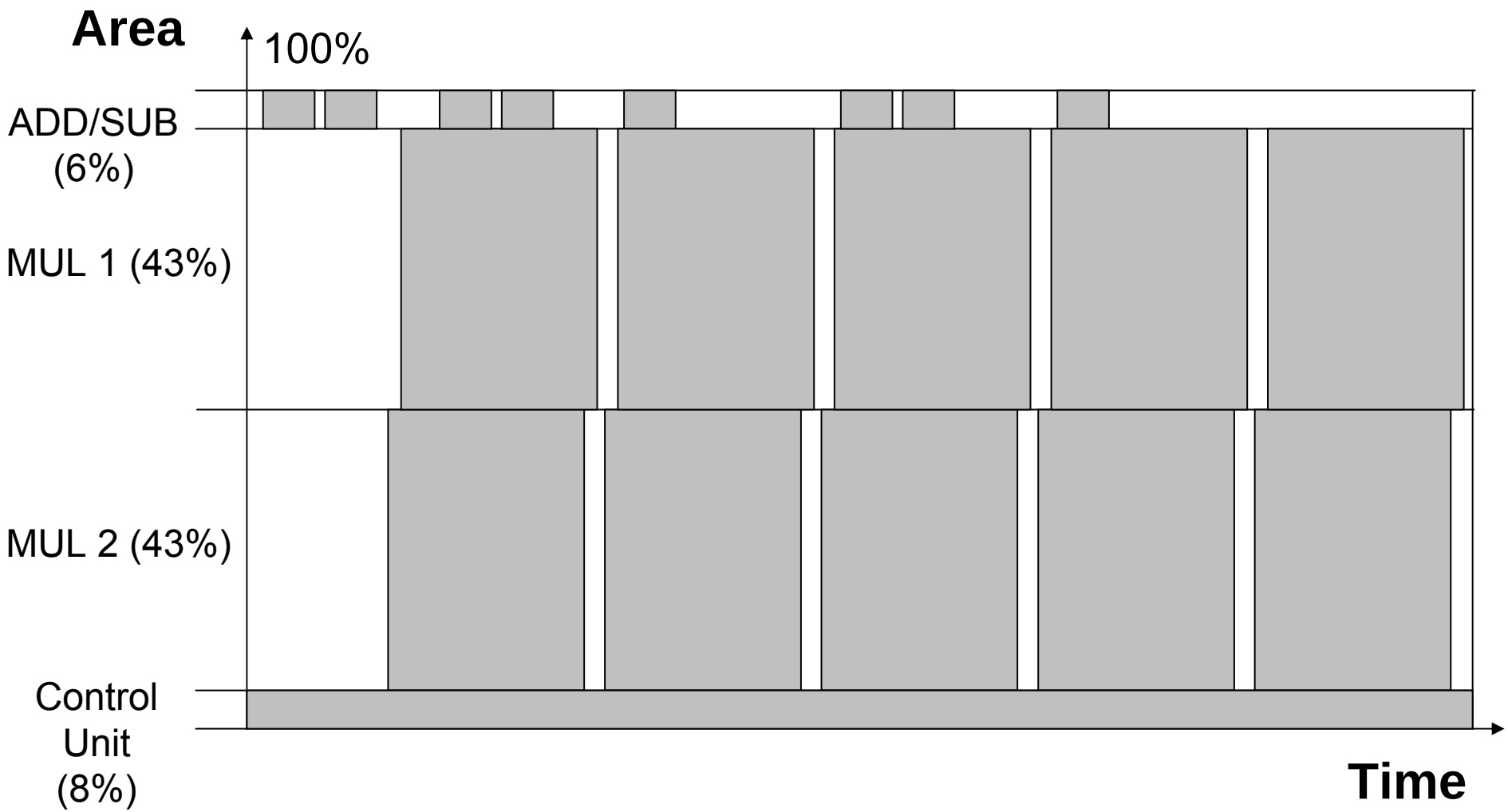


Basic Building Block – Montgomery Multiplier



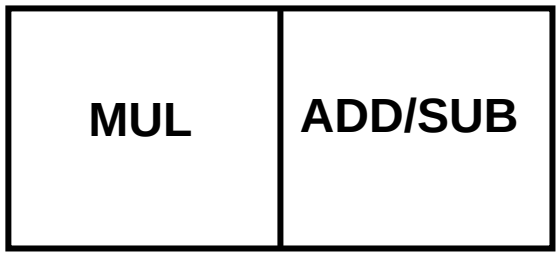
Common for all three factoring methods

Resources utilization in time – ECM Phase 1



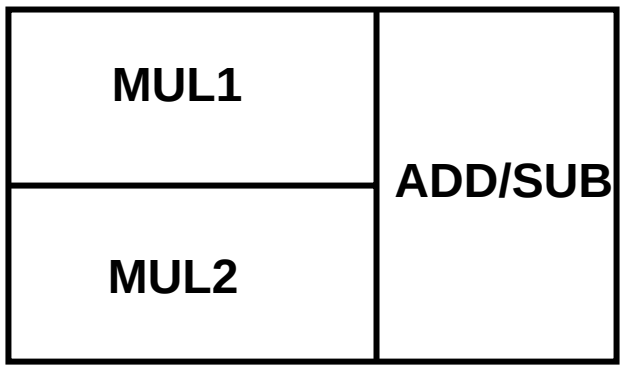
Optimum Execution Unit

rho

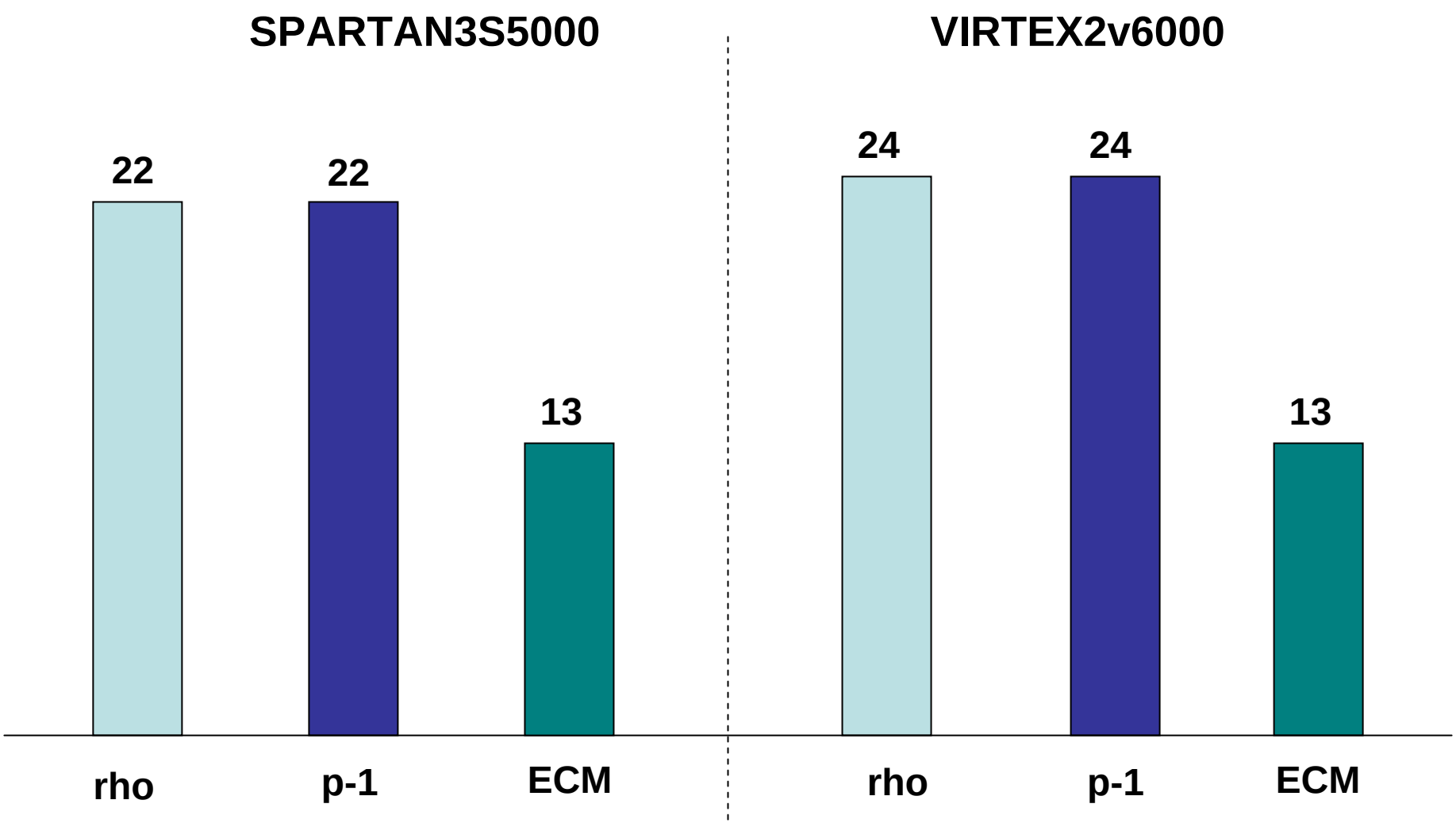


p-1

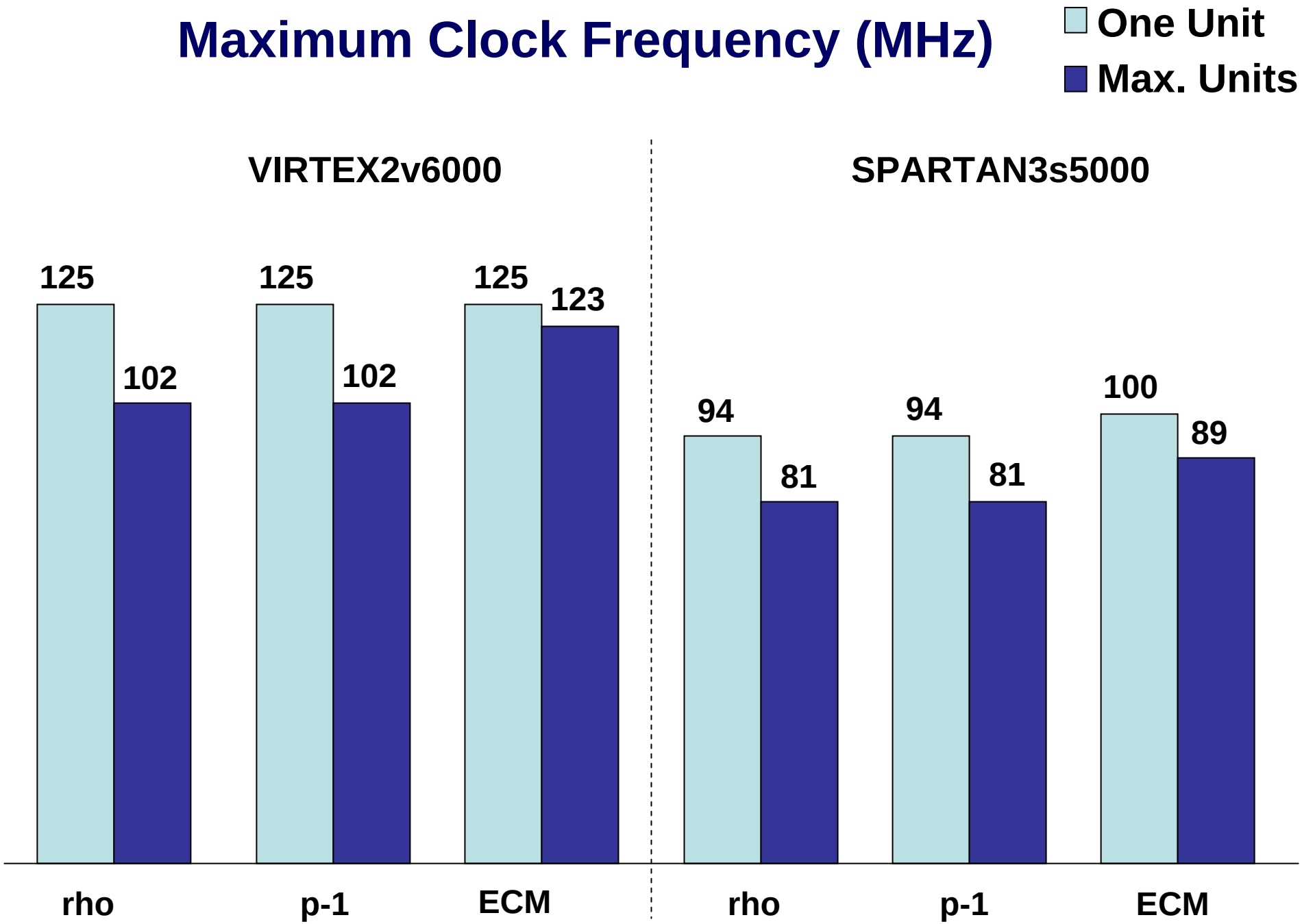
ECM



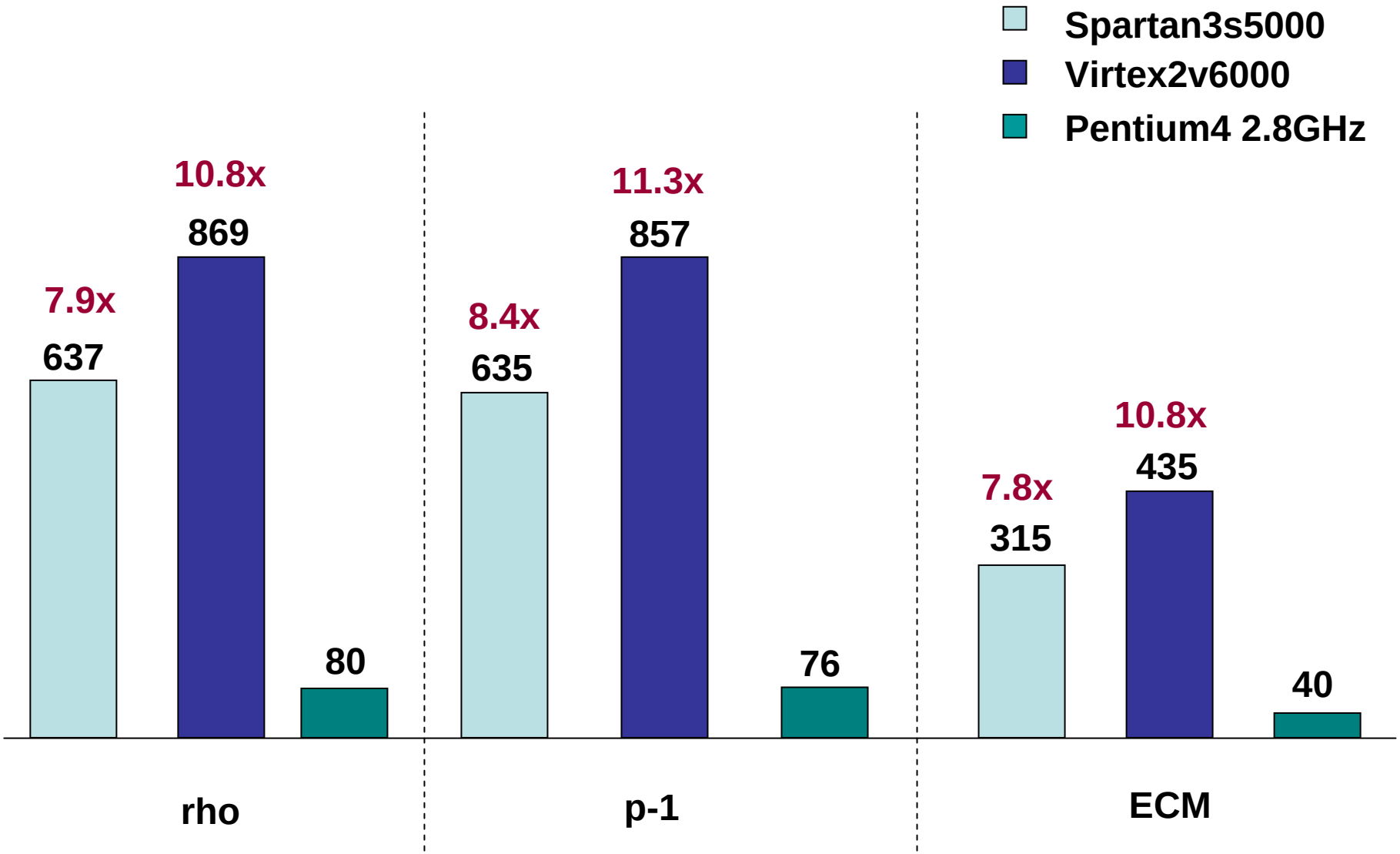
Maximum Number of Units per FPGA Device



Maximum Clock Frequency (MHz)



Runs per Second



Effect of using more aggressive parameters of ECM

Our ECM parameters

B1 = 960
B2 = 57000
D = 2·3·5·7 = 210

Parameters according to Rainer Steinwandt et al. (Wednesday talk)

B1 = **402**
B2 = **9680**
D = $2 \cdot 3^2 \cdot 5 = 90$

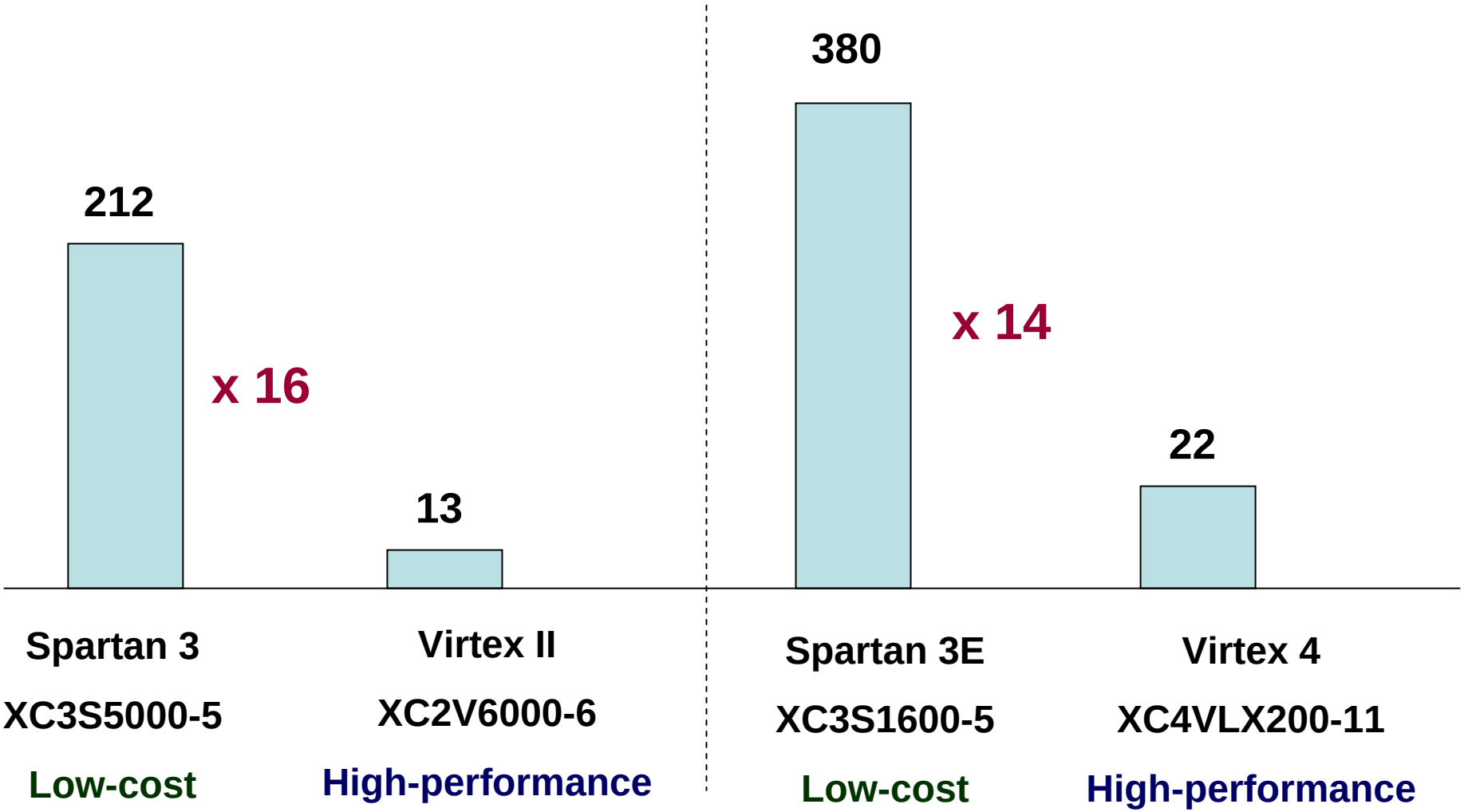
Execution time of Phase 1 & Phase 2 in hardware

36.6 ms → 3.5 x → 10.4 ms

Execution time of Phase 1 & Phase 2 in software (GMP-ECM)

24.8 ms → 2.8 x → 8.8 ms

Performance to cost ratio
Number of Phase 1 & Phase 2 ECM operations per second per $100



ASIC IMPLEMENTATION

ASIC Technology

Process:

0.13 μm

Libraries:

cb13fs120_tsmc_max

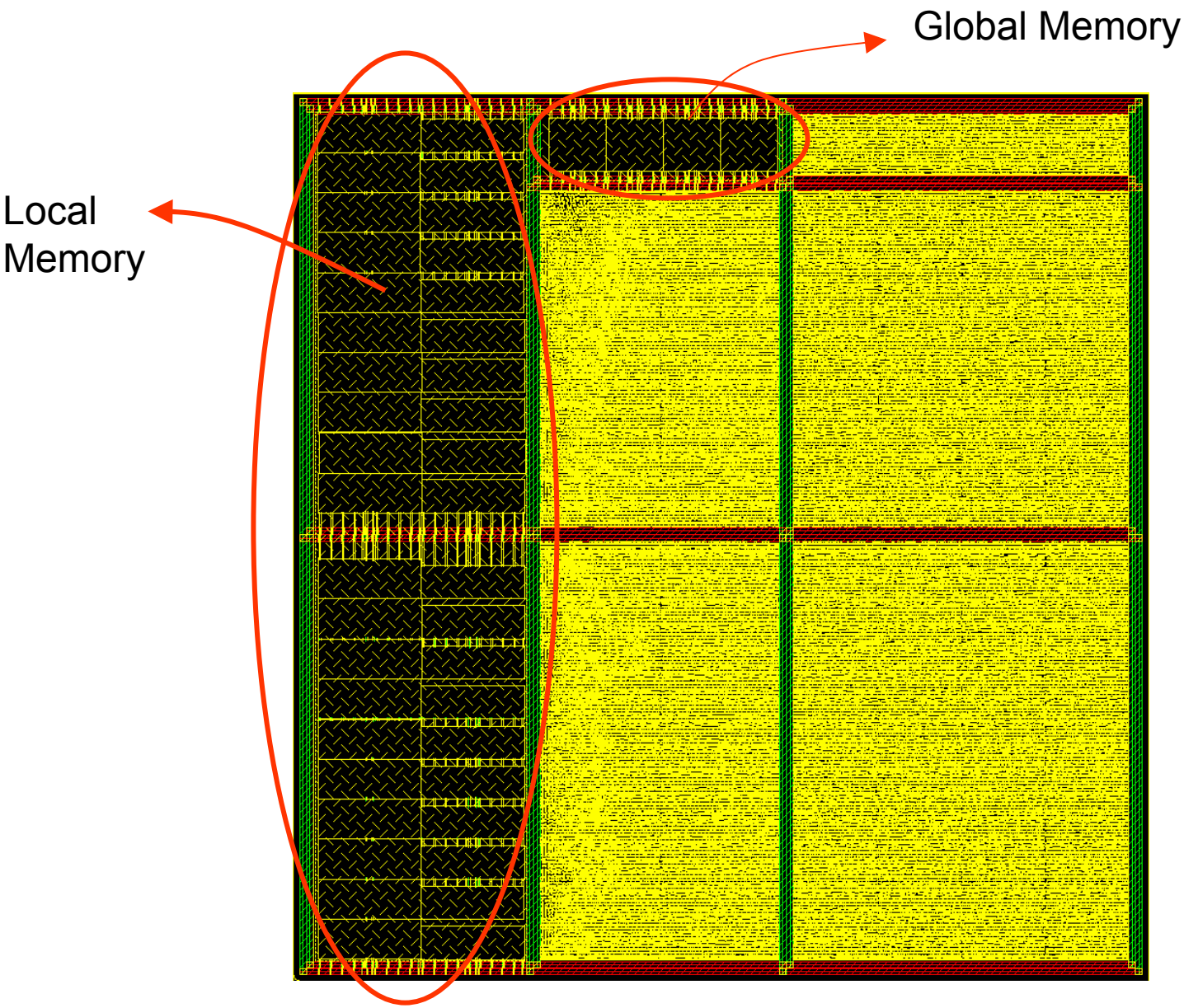
Memory Library:

ram16x128_max, ram32x64_max, ram32x32_max

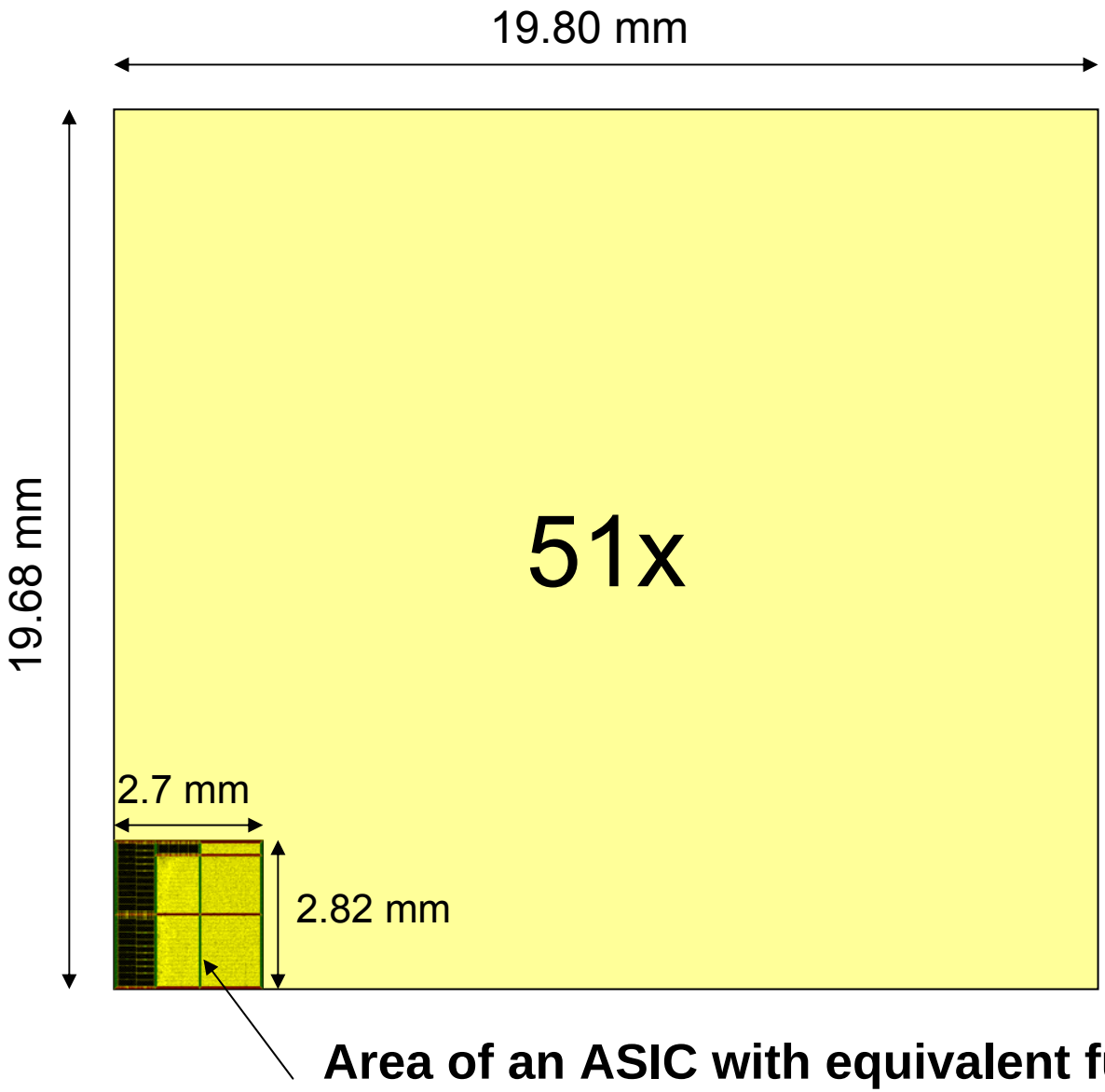
Tools:

Synopsys Design Analyzer, Astro, Primetime

Rho in an ASIC 130 nm



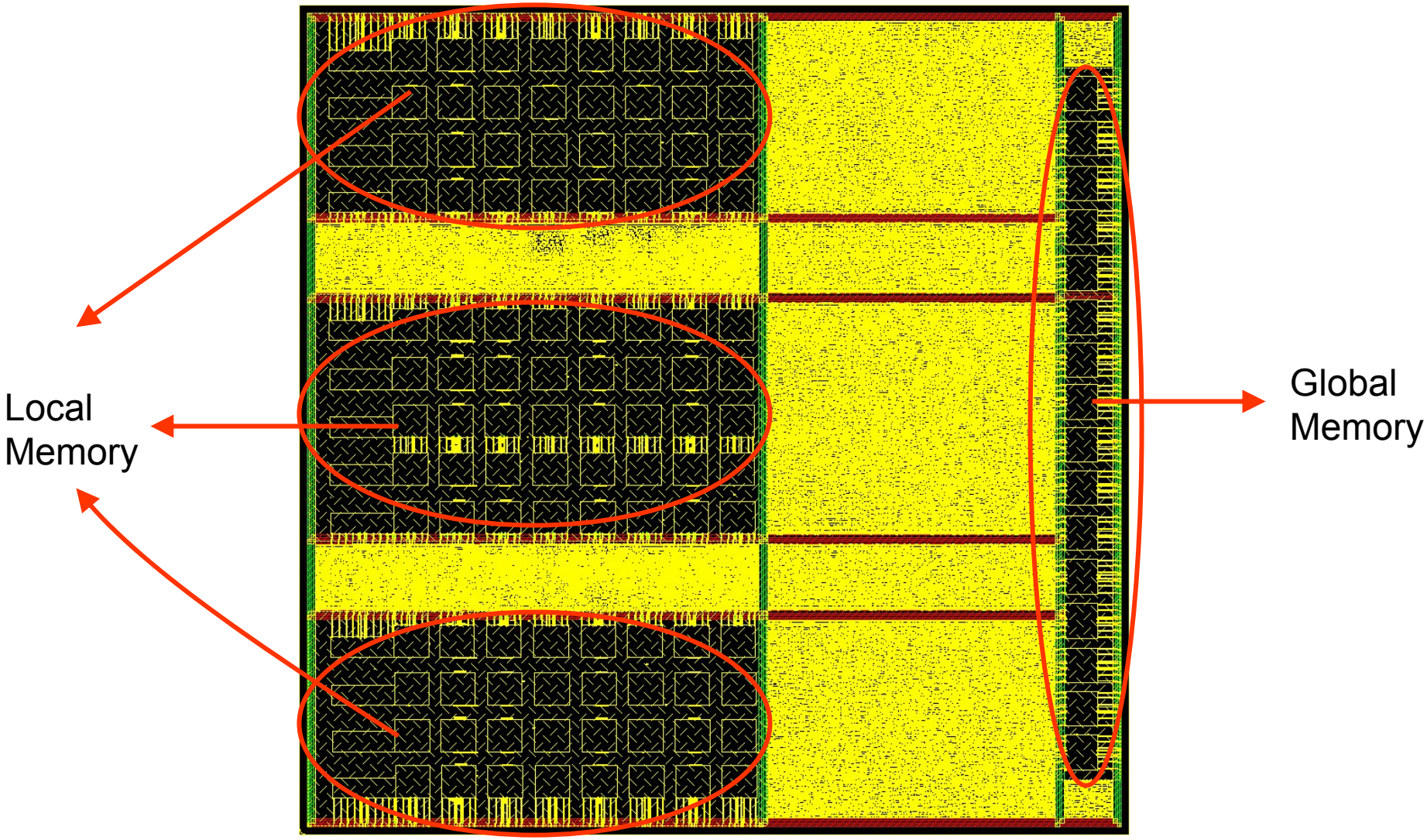
ASIC 130 nm vs. Virtex II 6000 – rho (24 units)



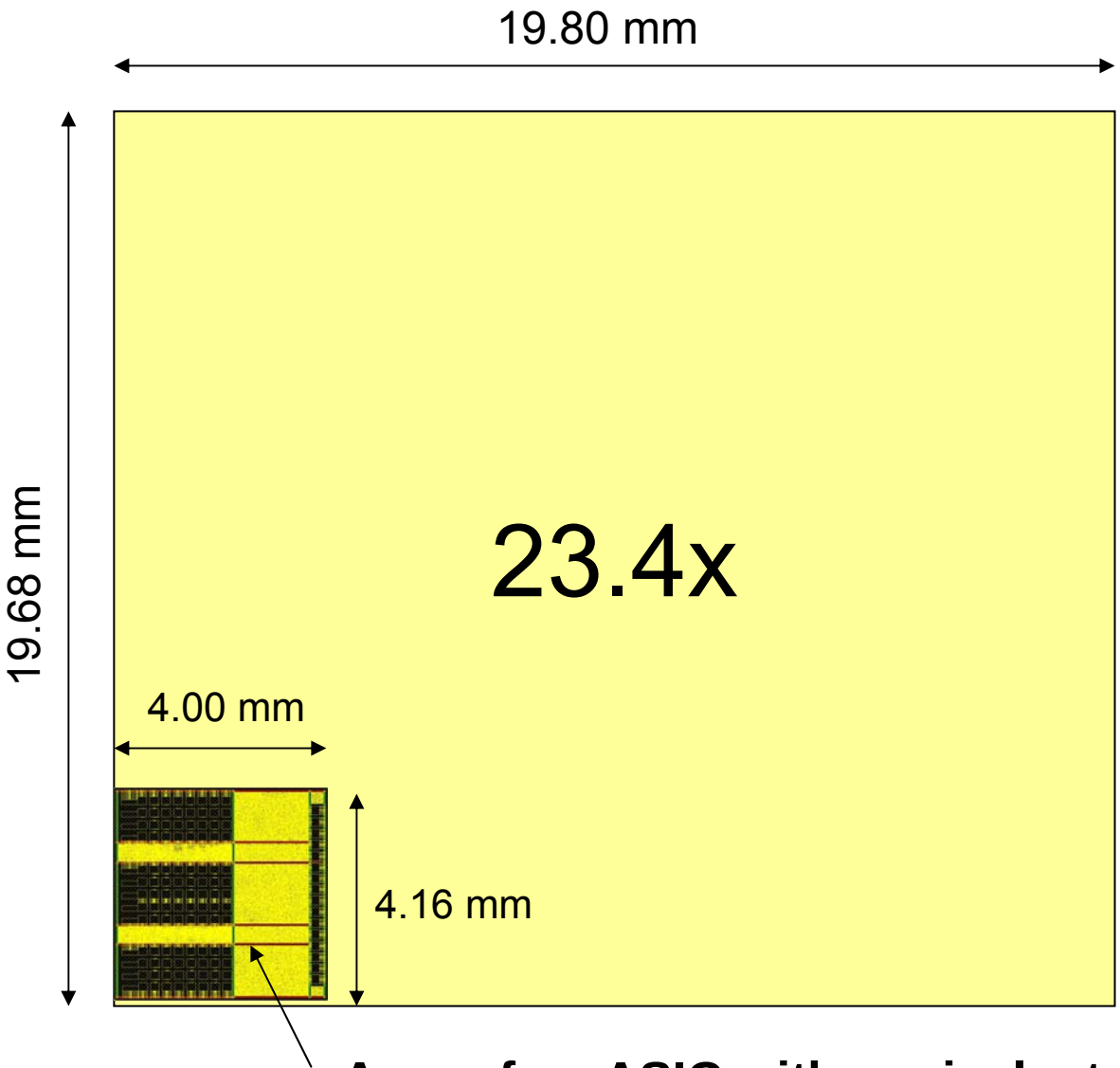
Area of Virtex II 6000
(estimation by R.J. Lim Fong, MS Thesis, VPI, 2004)

Area of an ASIC with equivalent functionality

ECM in an ASIC 130 nm



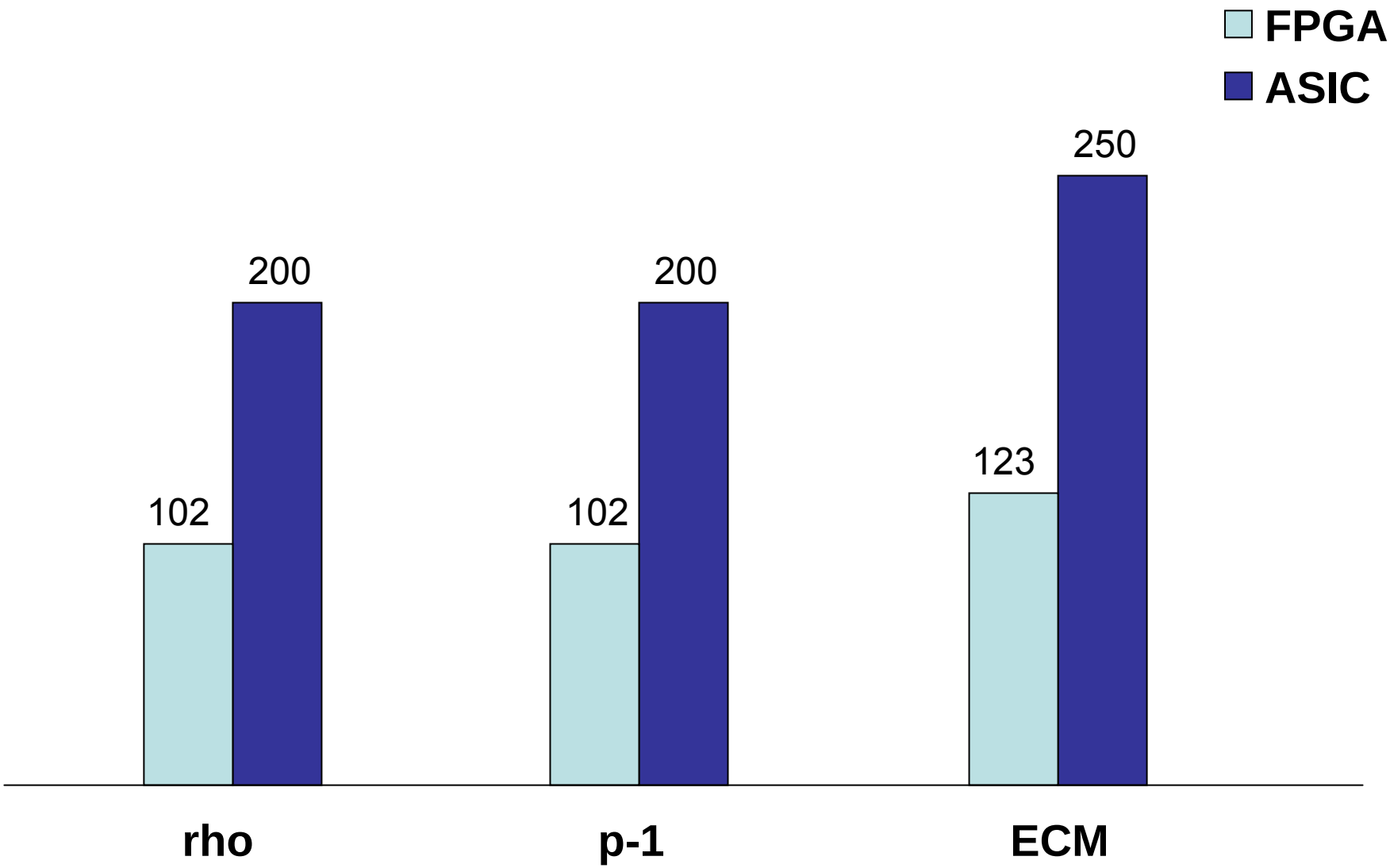
ASIC 130 nm vs. Virtex II 6000 – ECM (13 units)



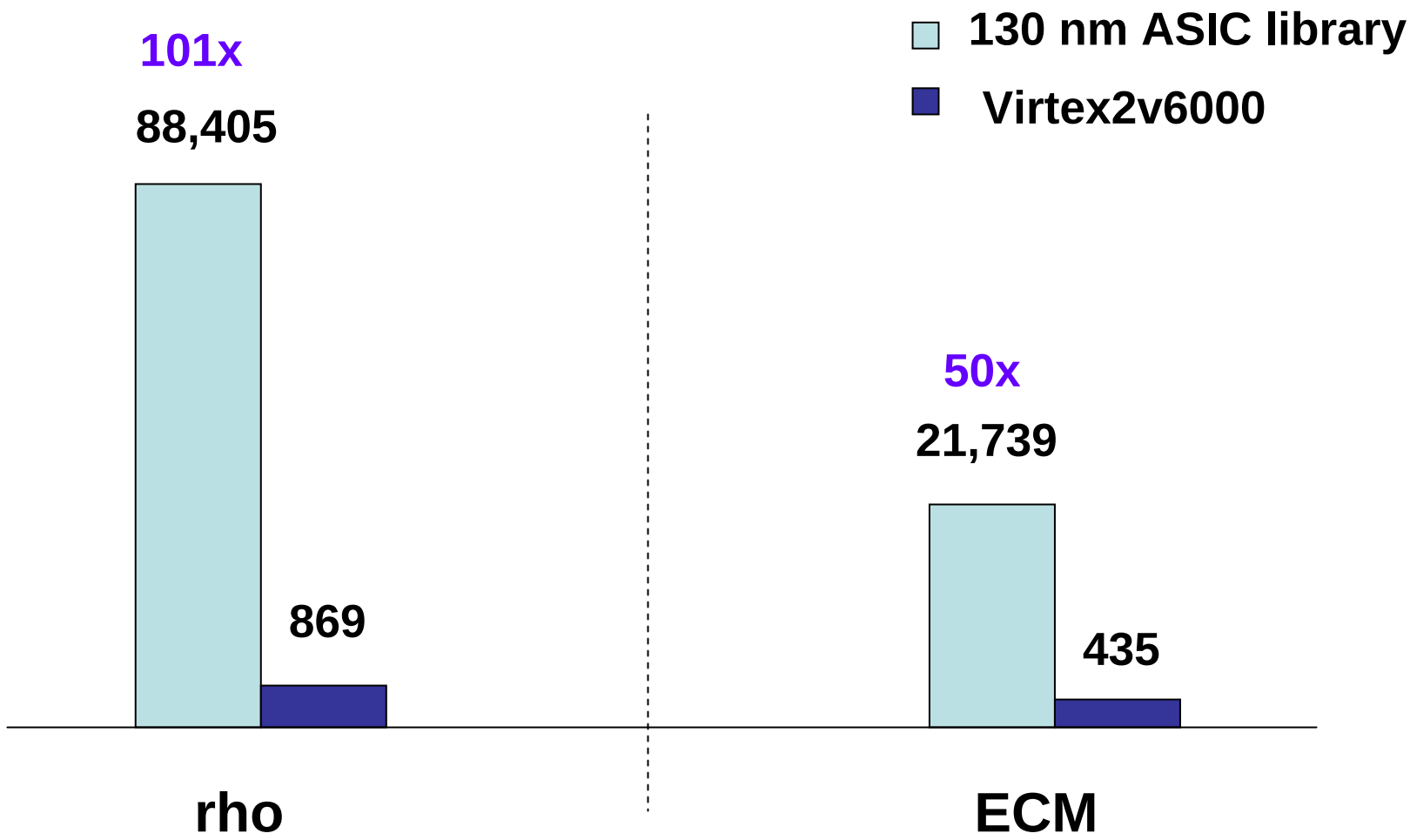
Area of Virtex II 6000
(estimation by R.J. Lim Fong, MS Thesis, VPI, 2004)

Area of an ASIC with equivalent functionality

Clock Frequency in MHz



Number of rho & ECM computations per second using the same chip area



SRC IMPLEMENTATION

SRC 6
reconfigurable computer

SRC 6 from
SRC Computers

Basic unit:

2 x Pentium Xeon 3 GHz

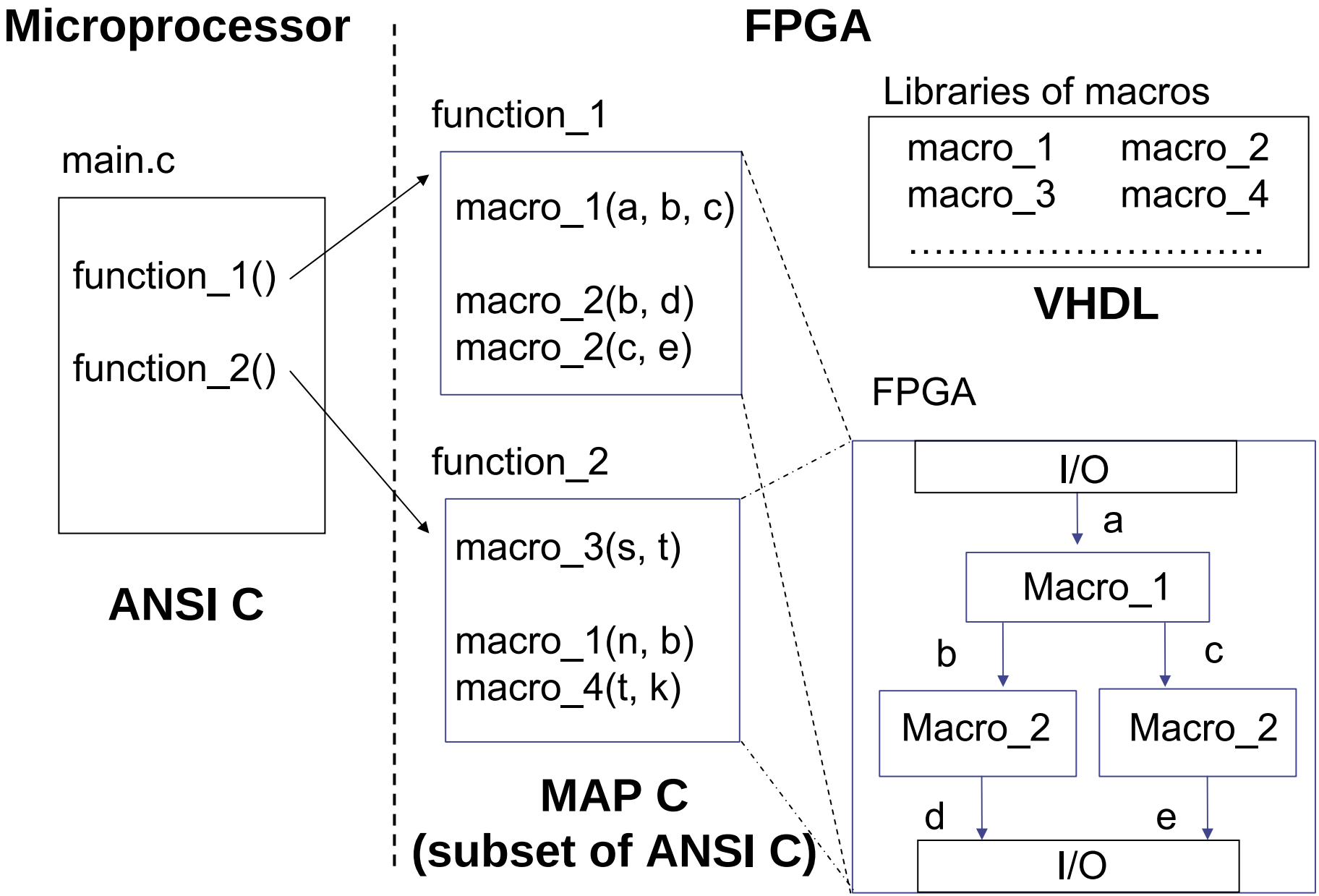
2 x Xilinx Virtex II FPGA
XC2V6000 running at 100 MHz

24 MB of the FPGA-board RAM

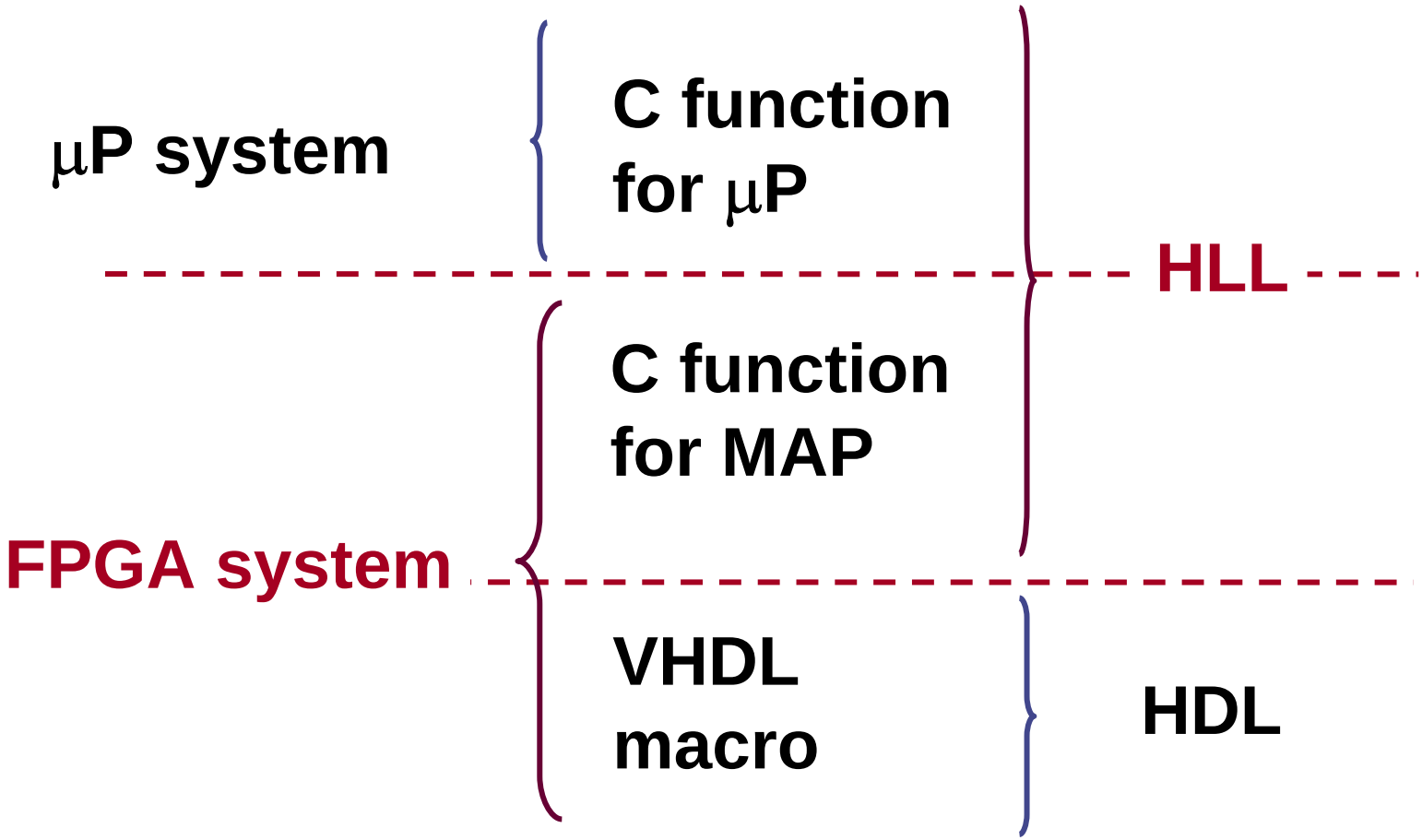
Fast communication interface between the microprocessor board and the FPGA board, 1600 MB/s

Multiple basic units can be connected using Hi-Bar Switch and Global Common Memory

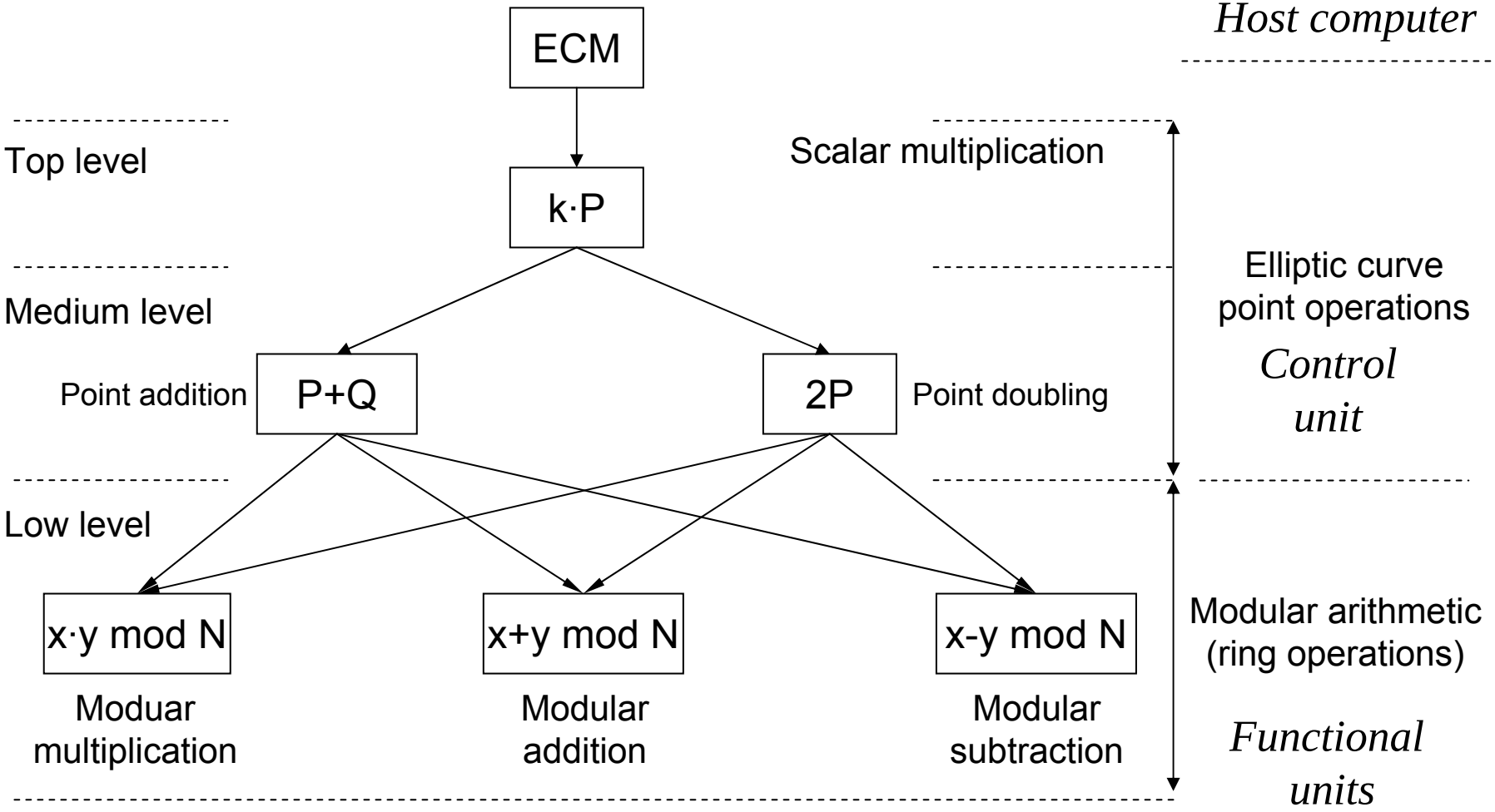
SRC Programming Model



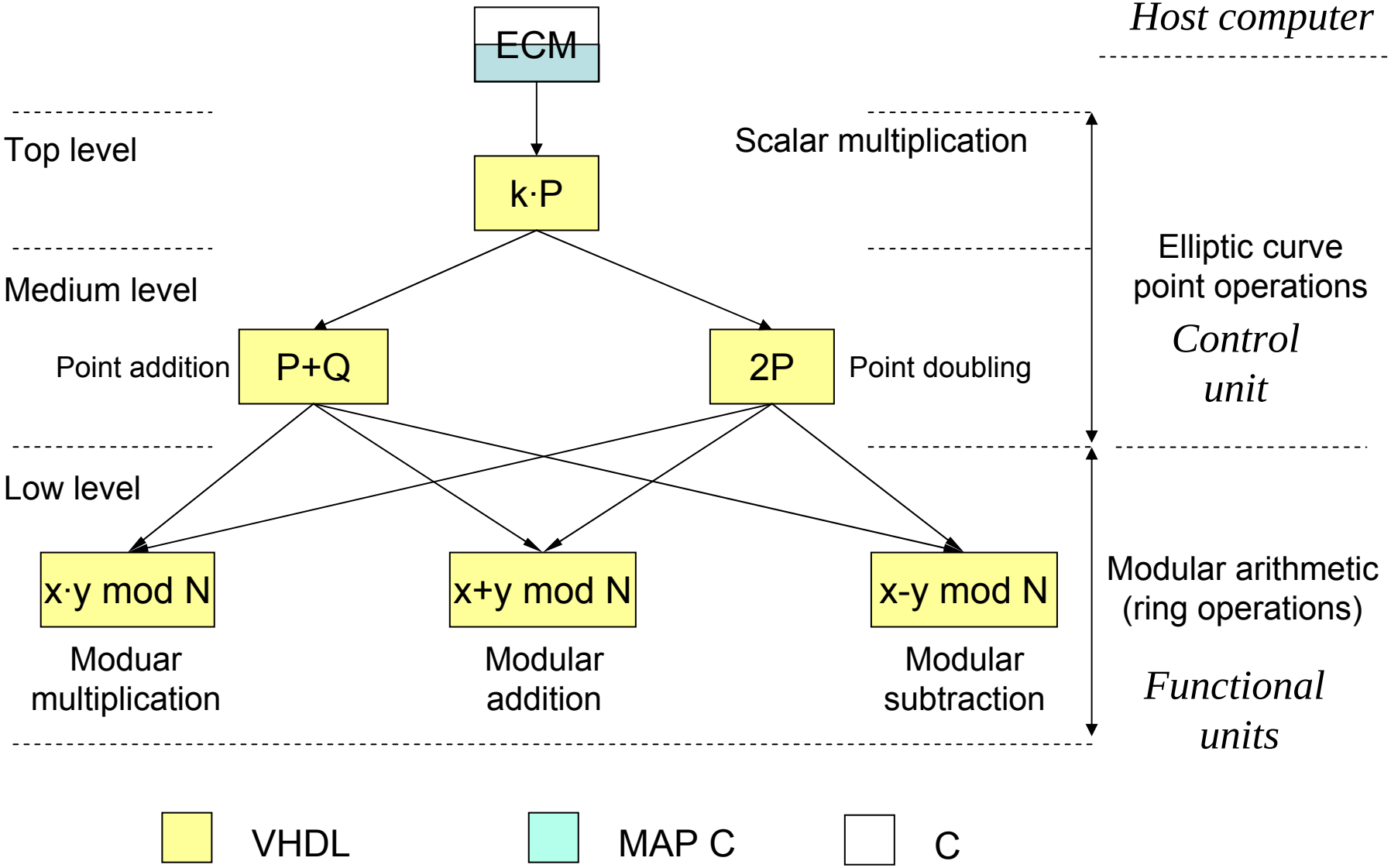
SRC Program Partitioning



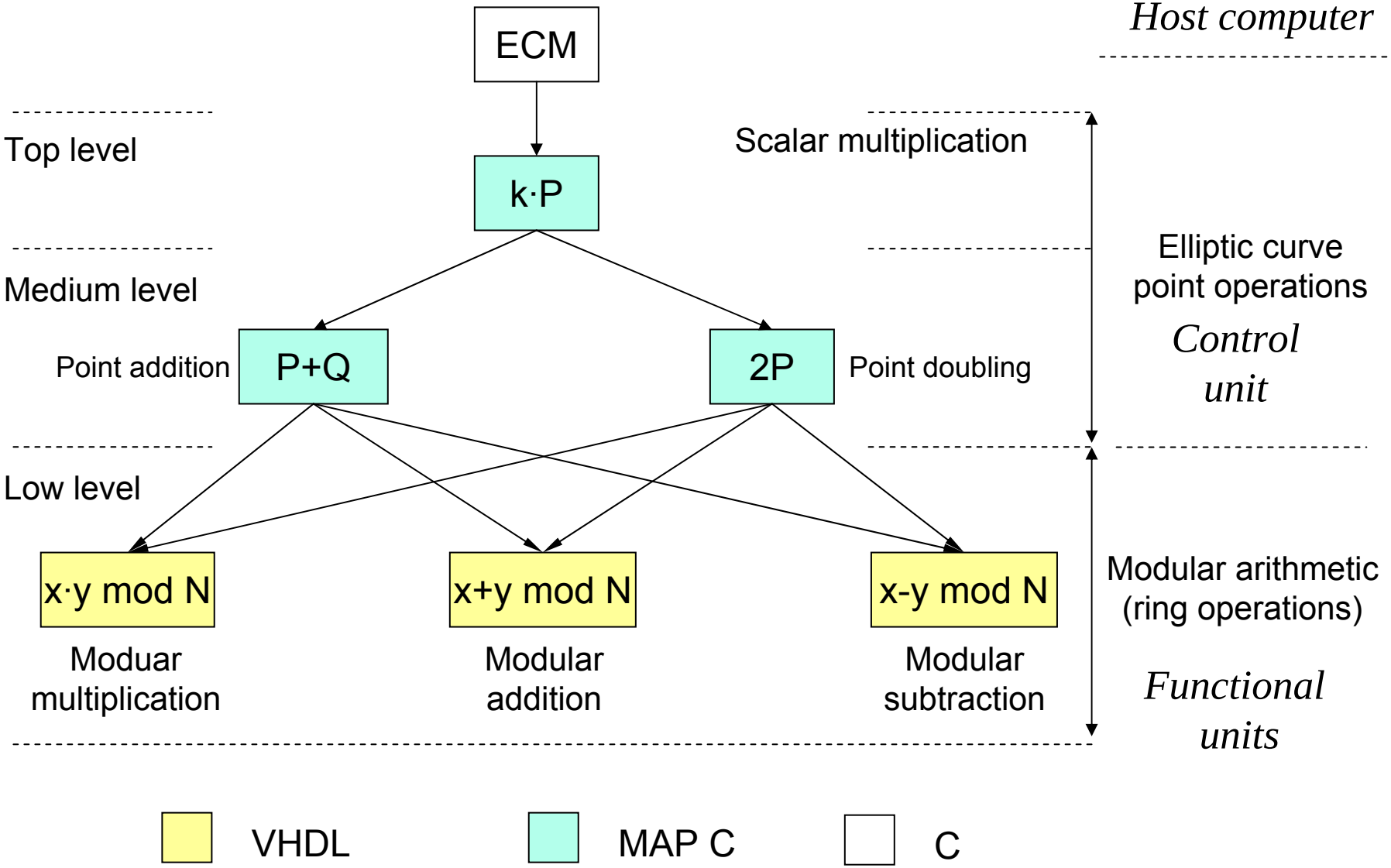
Hierarchy of Elliptic Curve Operations



VHDL-only implementation



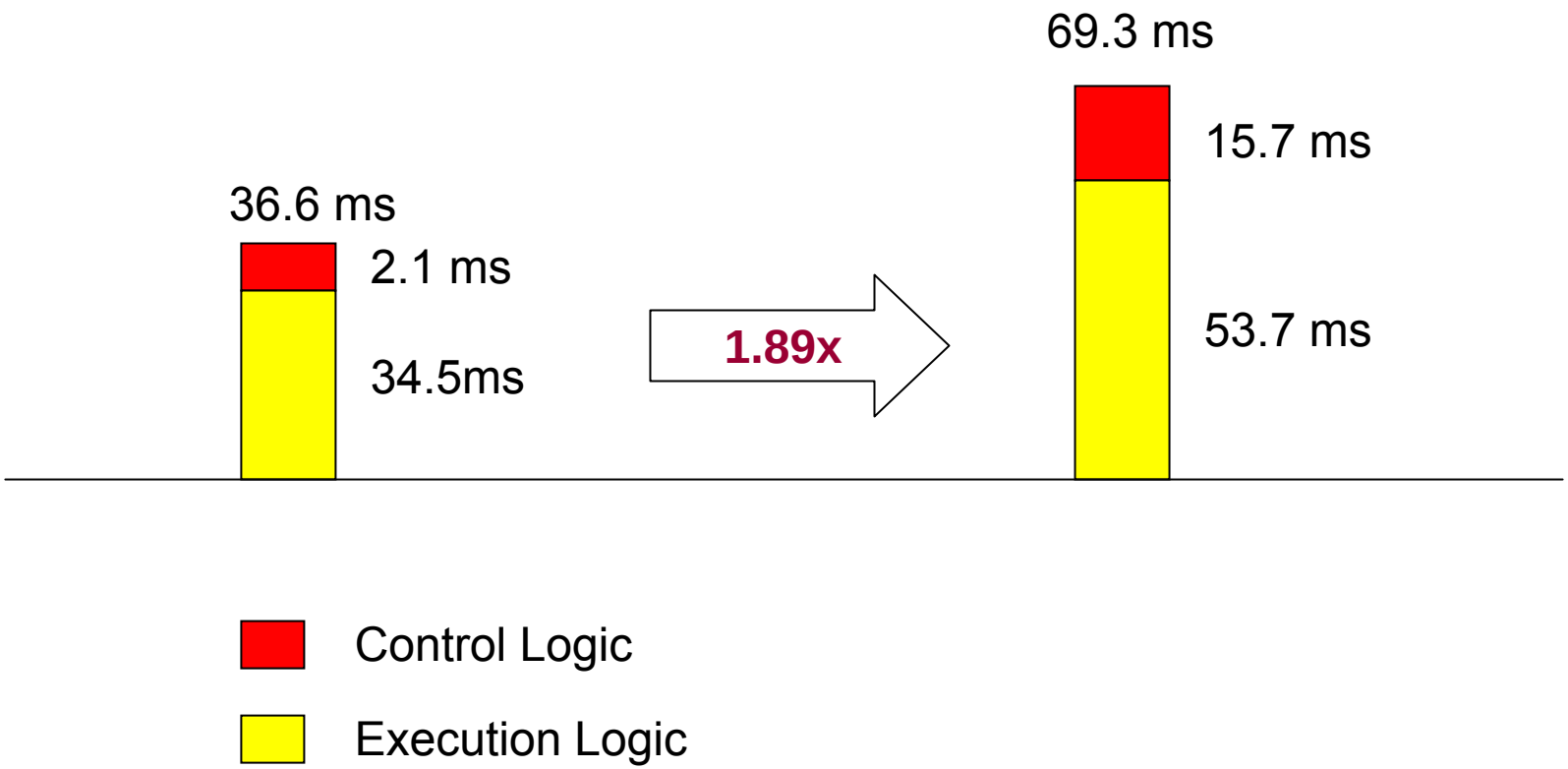
MAP C implementation



MAP C Compiler Performance Penalty

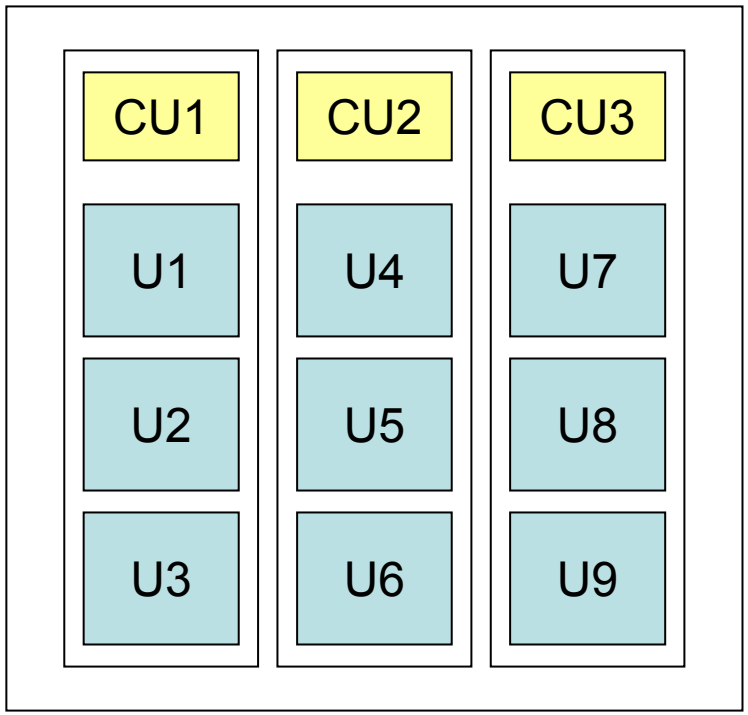
VHDL-only implementation

MAP-C implementation



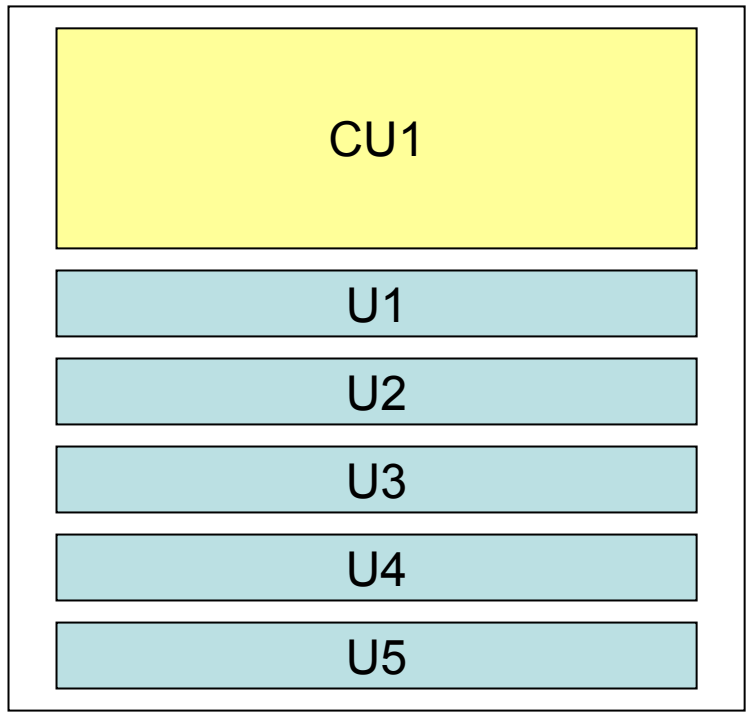
MAP C Compiler Area Penalty

VHDL



9 Units

MAP-C



5 Units

Conversion of MAP C to Hardware (1)

a = src_0

.....

.....

a = src_1

.....

.....

a = src_2

.....

.....

a = src_3

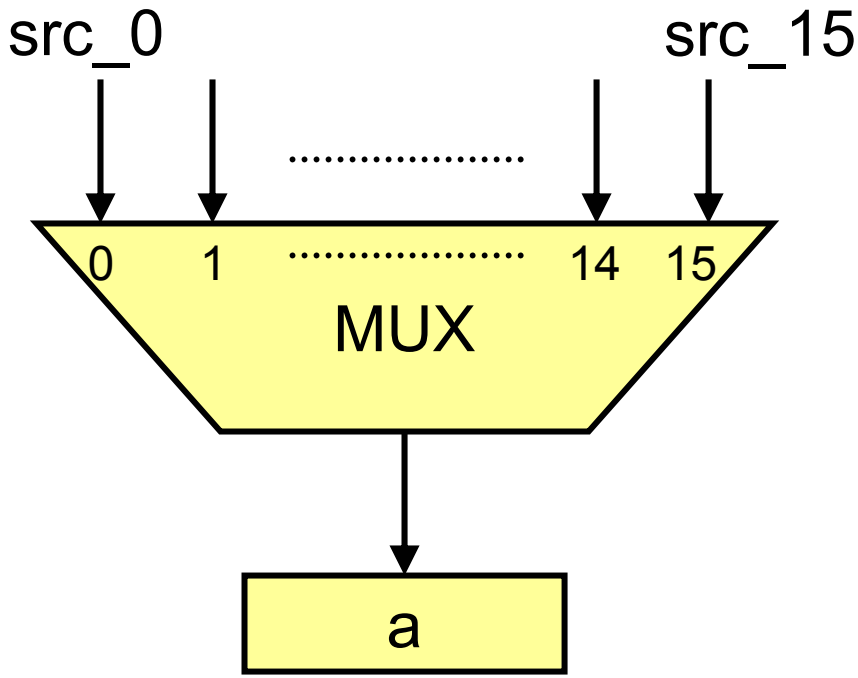
.....

.....

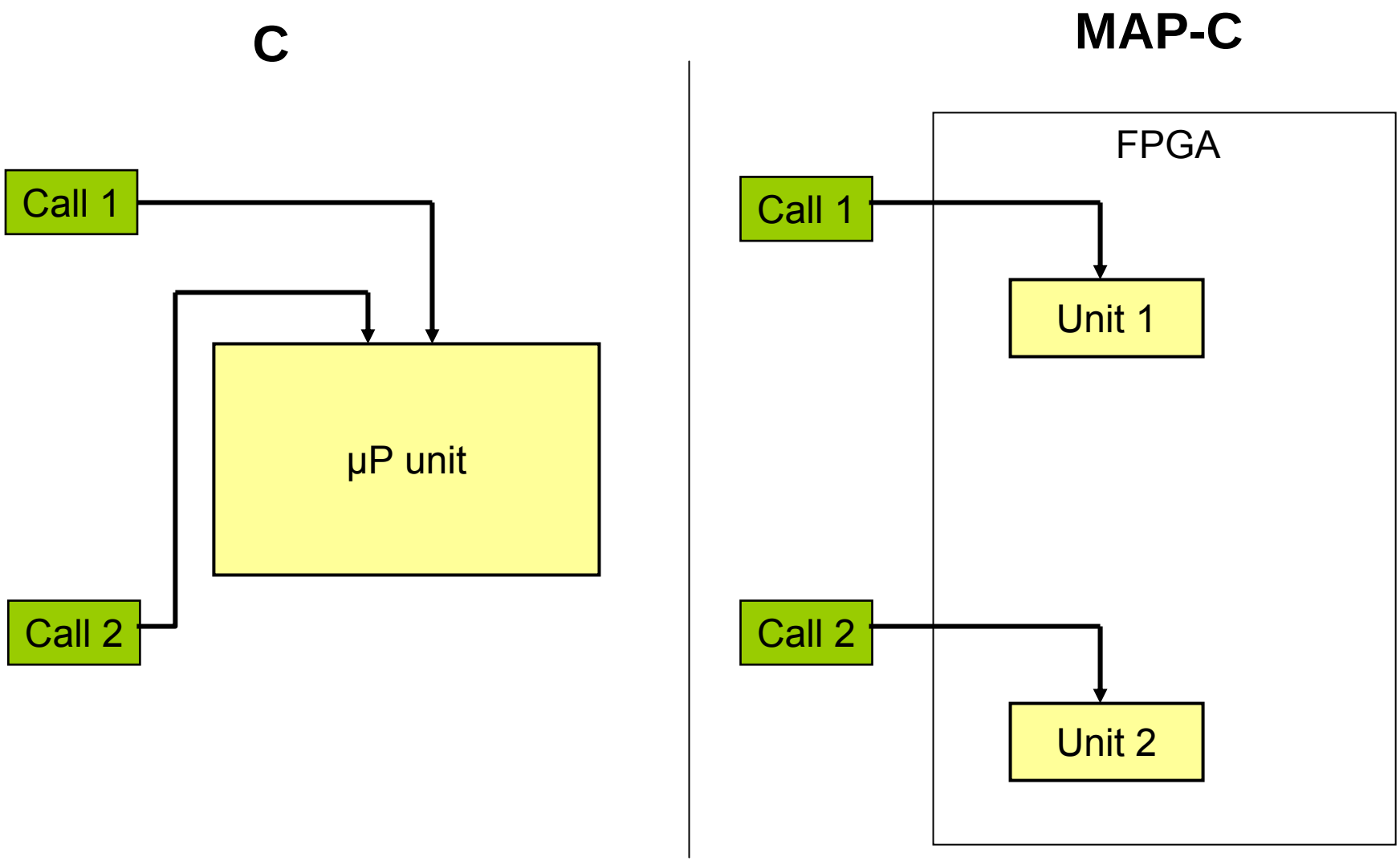
.....

.....

a = src_15



Conversion of MAP C to Hardware (2)

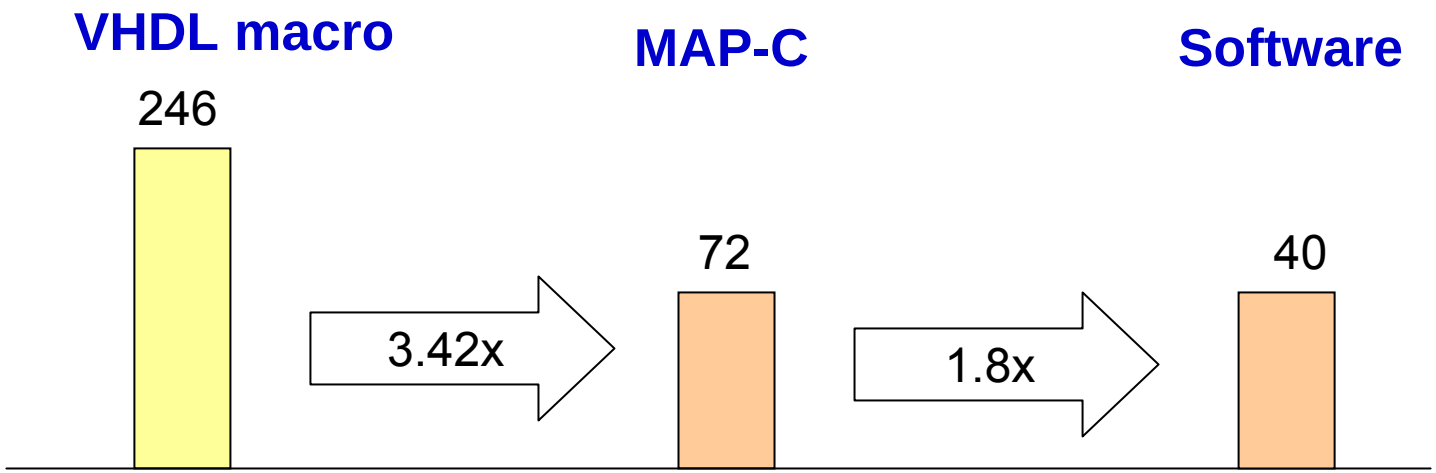


Recommended MAP C programming style

```
while (cond) { ….
if (cond_1) {
    src = src_1;
    dst = dst_1;
}
else if (cond_2) {
    src = src_2;
    dst = dst_2;
}
.
.
else {
    src = src_n;
    dst = dst_n;
}

macro (Mem[src], …, &Mem[dst]);
…}

→ Creates dependency
```

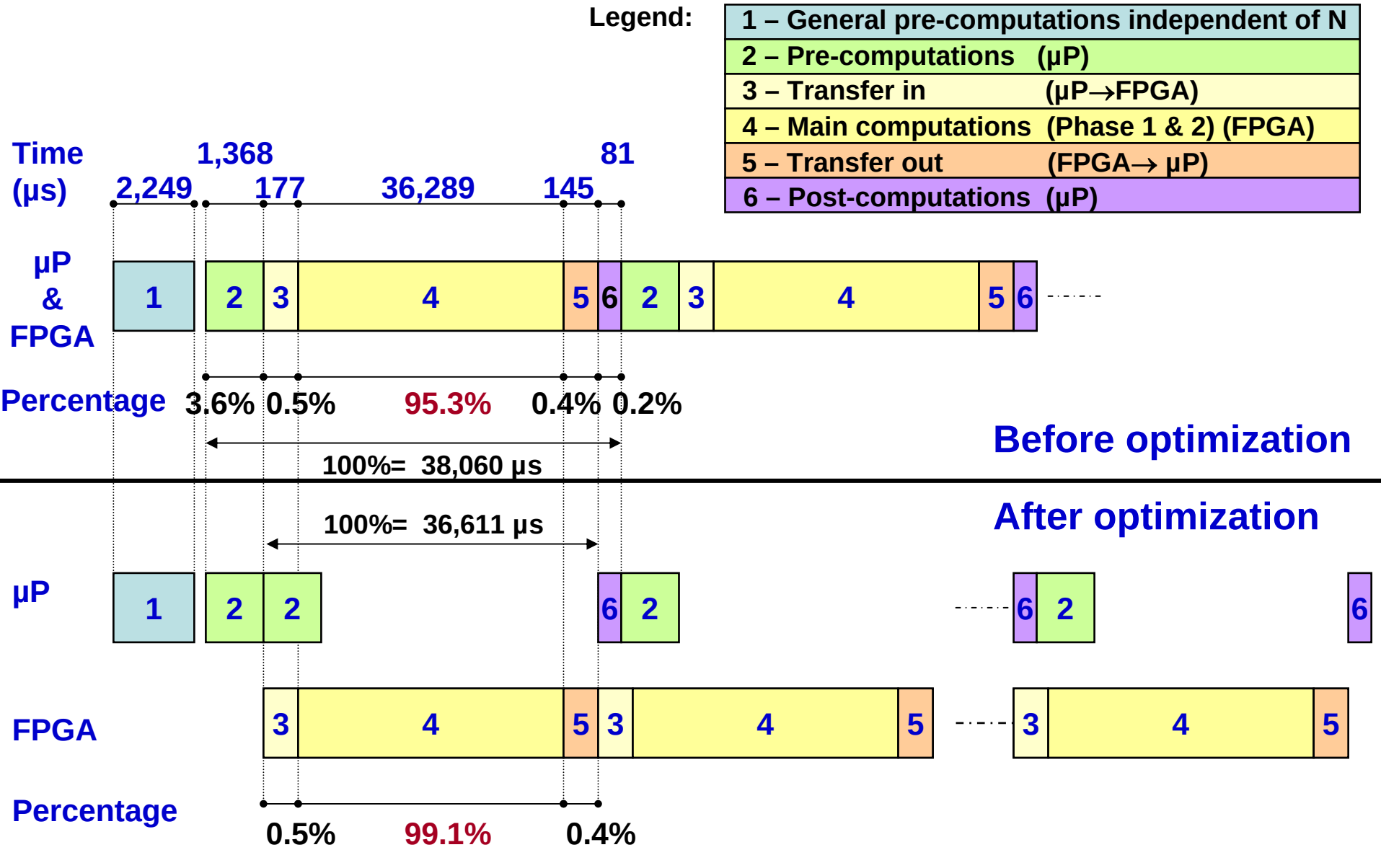
```
while (cond) { ….
If (cond_1) {
    src = src_1;
    dst = dst_1;
}
else if (cond_2) {
    src = src_2;
    dst = dst_2;
}
.
.
else {
    src = src_n;
    dst = dst_n;
}

macro (InMem[src], …, &OutMem[dst]);
…}
while(cond)
   InMem[dst++] = OutMem[dst++];
→ No dependency
```

ECM Operations / sec



Lines of code

Results of experimental testing of the VHDL version using SRC 6 reconfigurable computer



ECM ON COPACOBANA

Timing requirements for COPACOBANA

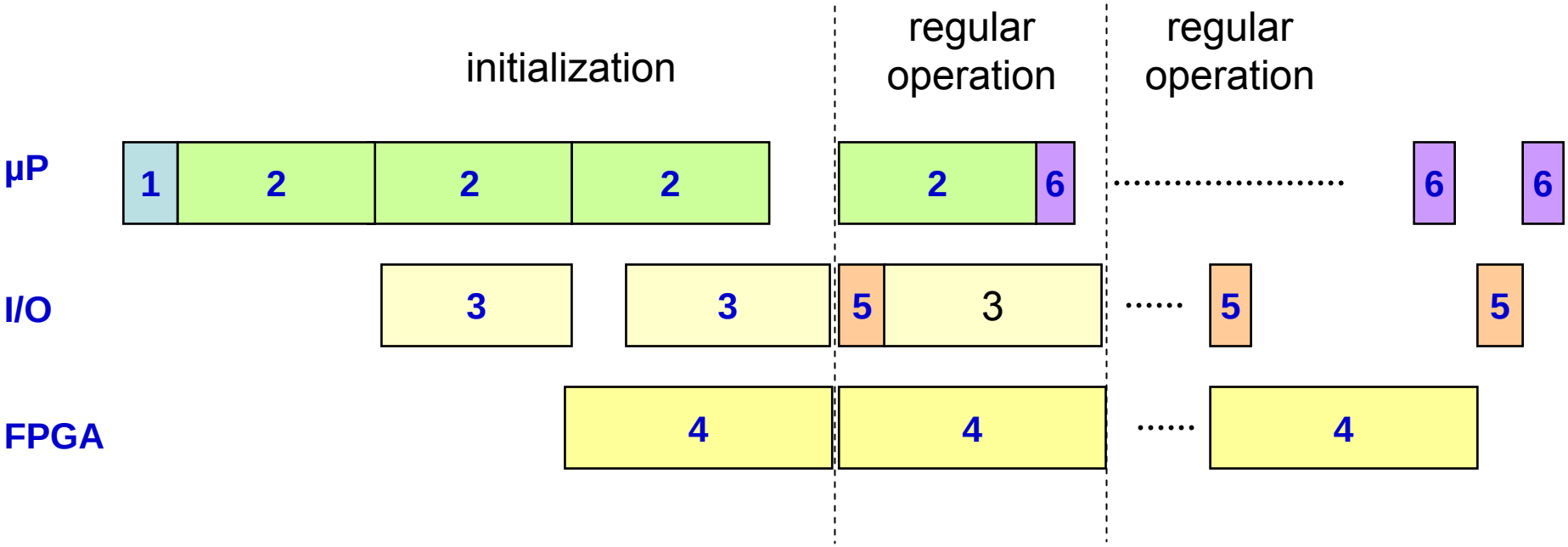
120 FPGAs x 2 ECM units/FPGA = 240 ECM units
Total execution time for Phase 1 & Phase 2 = 36.6 ms

Data transfer in	=	240 x 128 Bytes	=	30720 Bytes
Data transfer out	=	240 x 32 Bytes	=	7680 Bytes
Total			=	38400 Bytes

Required speed = $\frac{\text{Total data transfer}}{\text{Total execution time}}$ = 1.06 MB/s

USB 1.2 speed	=	12 Mb/s	=	1.5 MB/s
USB 2.0 speed	=	480 Mb/s	=	60.0 MB/s

Overlapping of Computations and Data Transfers on COPACOBANA



Legend:		
1 – General pre-computations independent of N		2249 µs
2 – Pre-computations	(µP)	34048 µs
3 – Transfer in	(µP→FPGA)	
4 – Main computations	(Phase 1 & 2) (FPGA)	36611 µs
5 – Transfer out	(FPGA→ µP)	
6 – Post-computations	(µP)	2016 µs

Conclusions

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

- low-cost FPGAs vs. microprocessors x 8-10
- ASICs vs. low-cost FPGAs x 50-100

Best environment for prototyping
of hardware implementations of codebreakers

- general-purpose reconfigurable computers (e.g., SRC)

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

- special-purpose machines based on
  - low-cost FPGAs (or ASICs for very high volumes)

Thank you!

Questions??
?