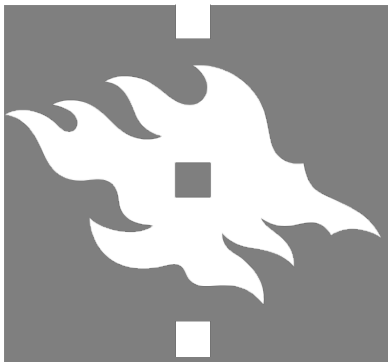


FAST ENDOMORPHISMS IN HARDWARE

Kimmo Järvinen[1,2]

[1] University of Helsinki, Computer Science, Helsinki, Finland
kimmo.u.jarvinen@helsinki.fi

[2] Xiphera Ltd., Espoo, Finland
kimmo.jarvinen@xiphera.com

The 21st Workshop on Elliptic Curve Cryptography

Nijmegen, the Netherlands, Nov. 13–15, 2017

INTRODUCTION

- This talk surveys my work on hardware implementations of ECC with fast endomorphisms
- Particularly: Koblitz curves, Four$\mathbb{Q}$, and GLV/GLS curves
- In software, fast endomorphisms reduce the number of operations and lead to significant speedups
- In hardware, simplicity is often the key to efficiency and the feasibility of fast endomorphisms is less clear

PRELIMINARIES

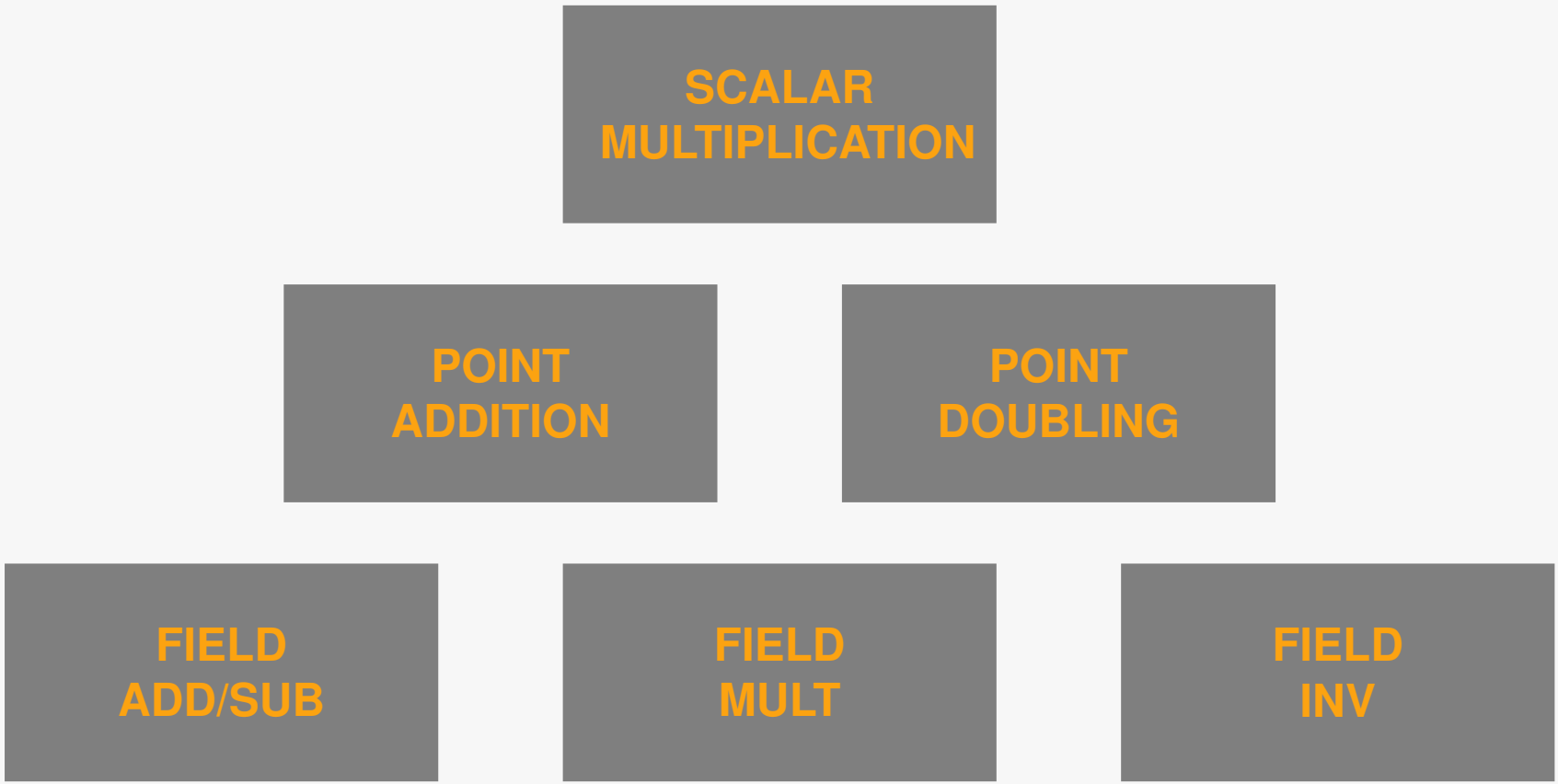
SCALAR MULTIPLICATION

- Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$
- Points on E (together with $\mathcal{O}$) form an additive Abelian group
- Let k be an integer and P be a point on E; then, scalar multiplication is the following operation:

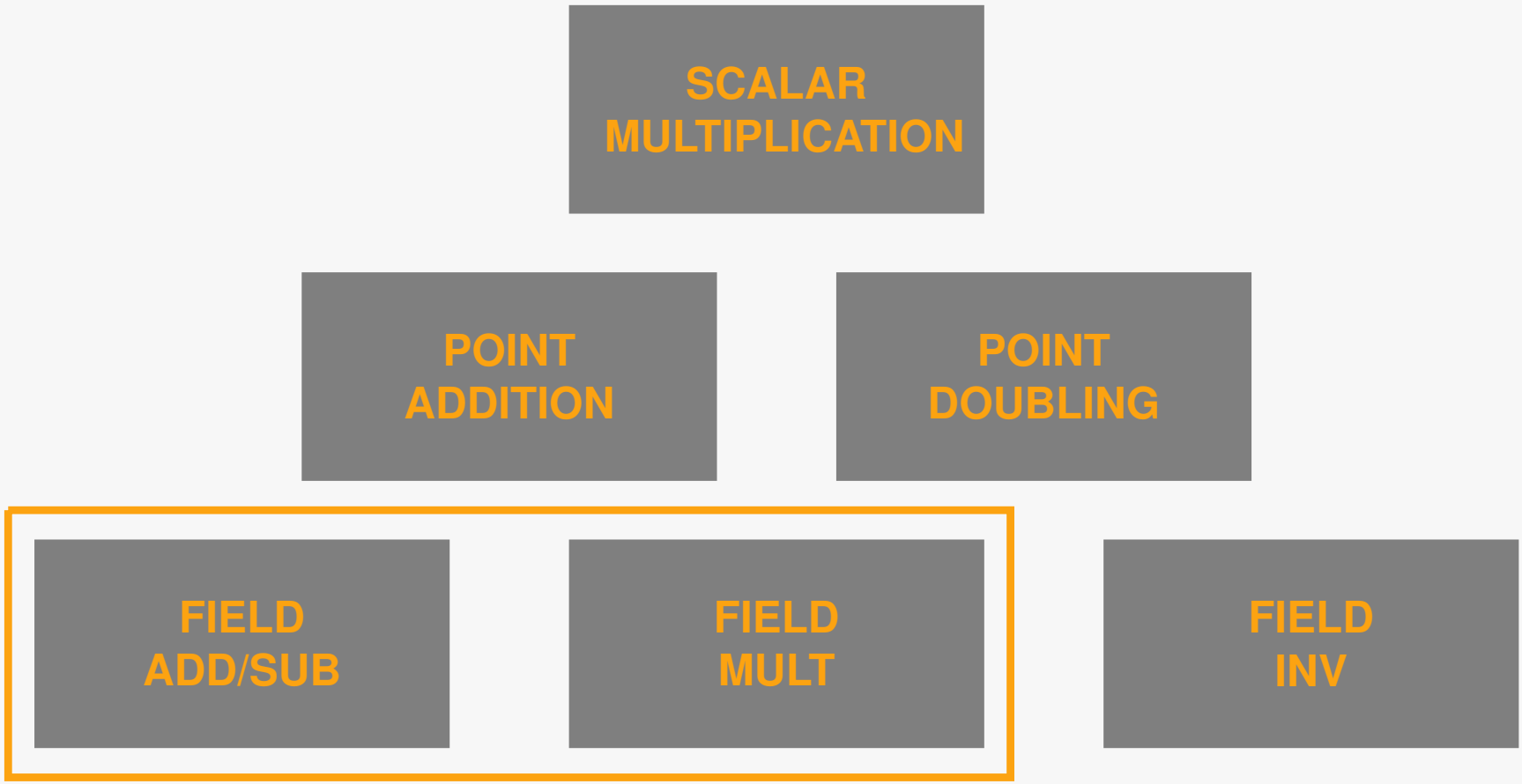
$$[k]P = \underbrace{P + P + \ldots + P}_{k \text{ times}}$$

- Scalar multiplication is the central operation of ECC mostly determining the efficiency of the cryptosystem

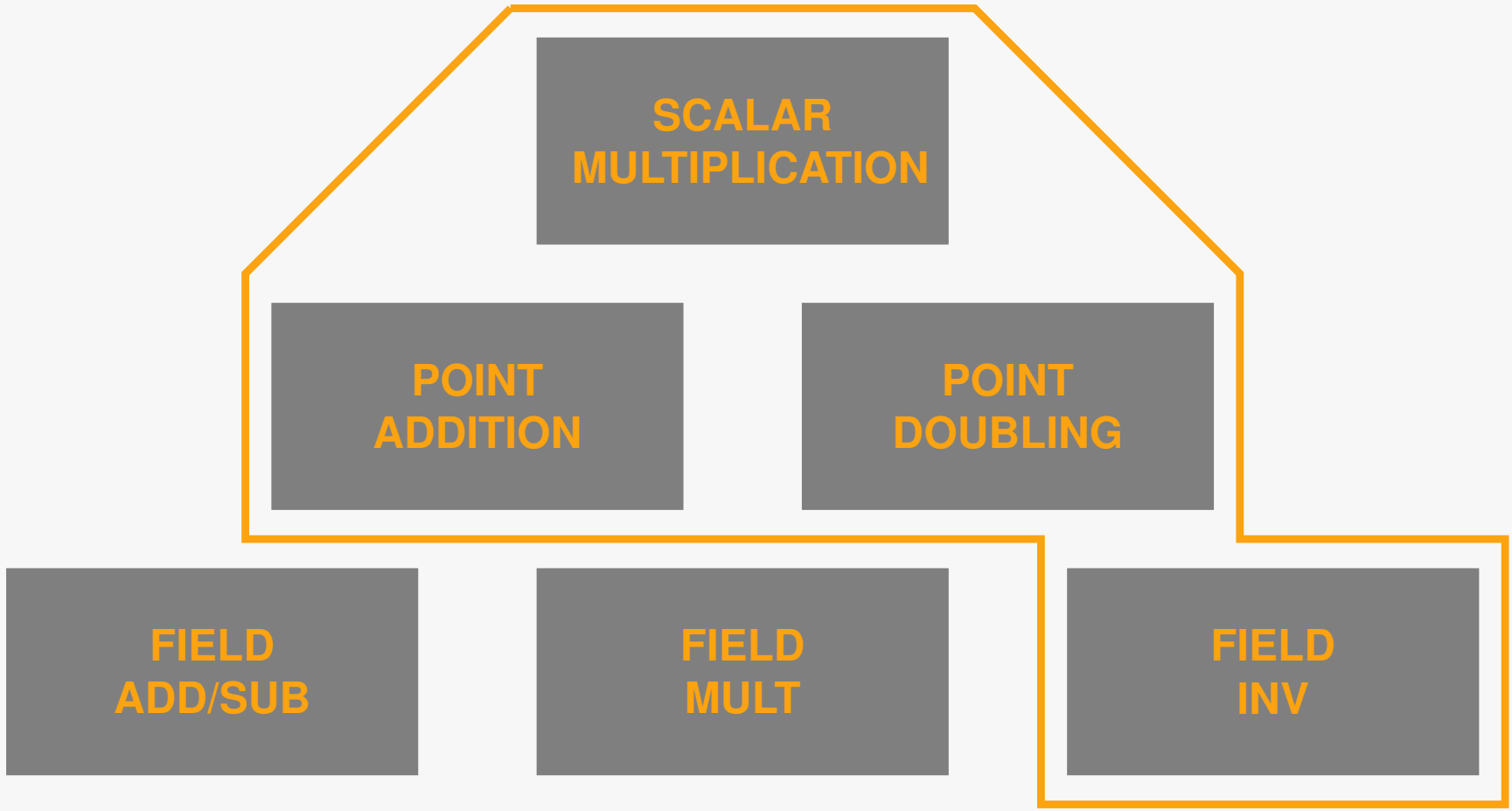
ECC HIERARCHY



ECC HIERARCHY

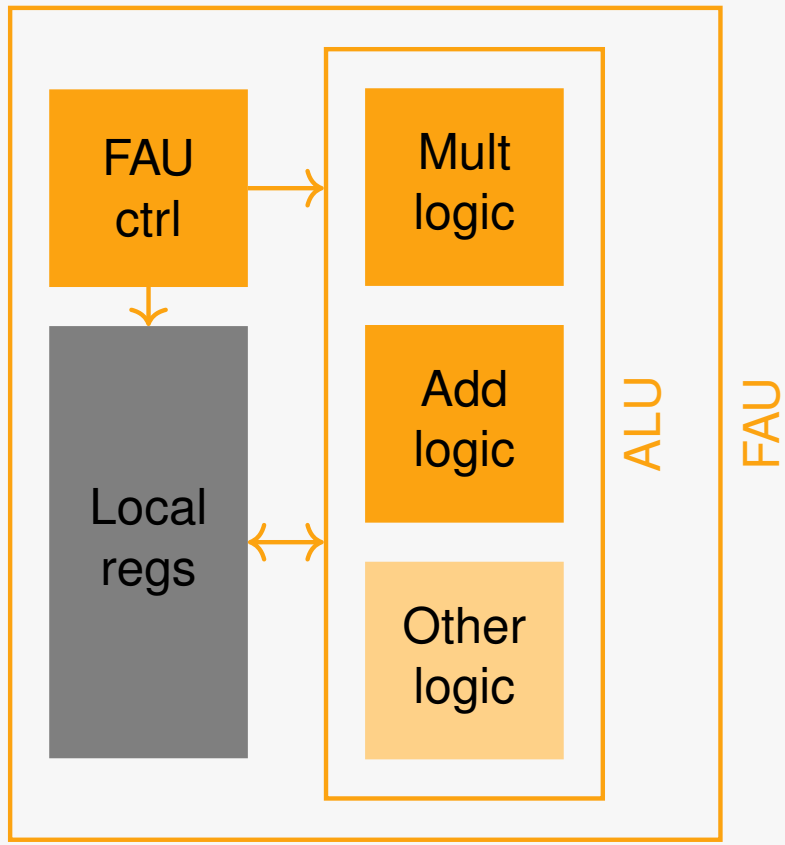


ECC HIERARCHY

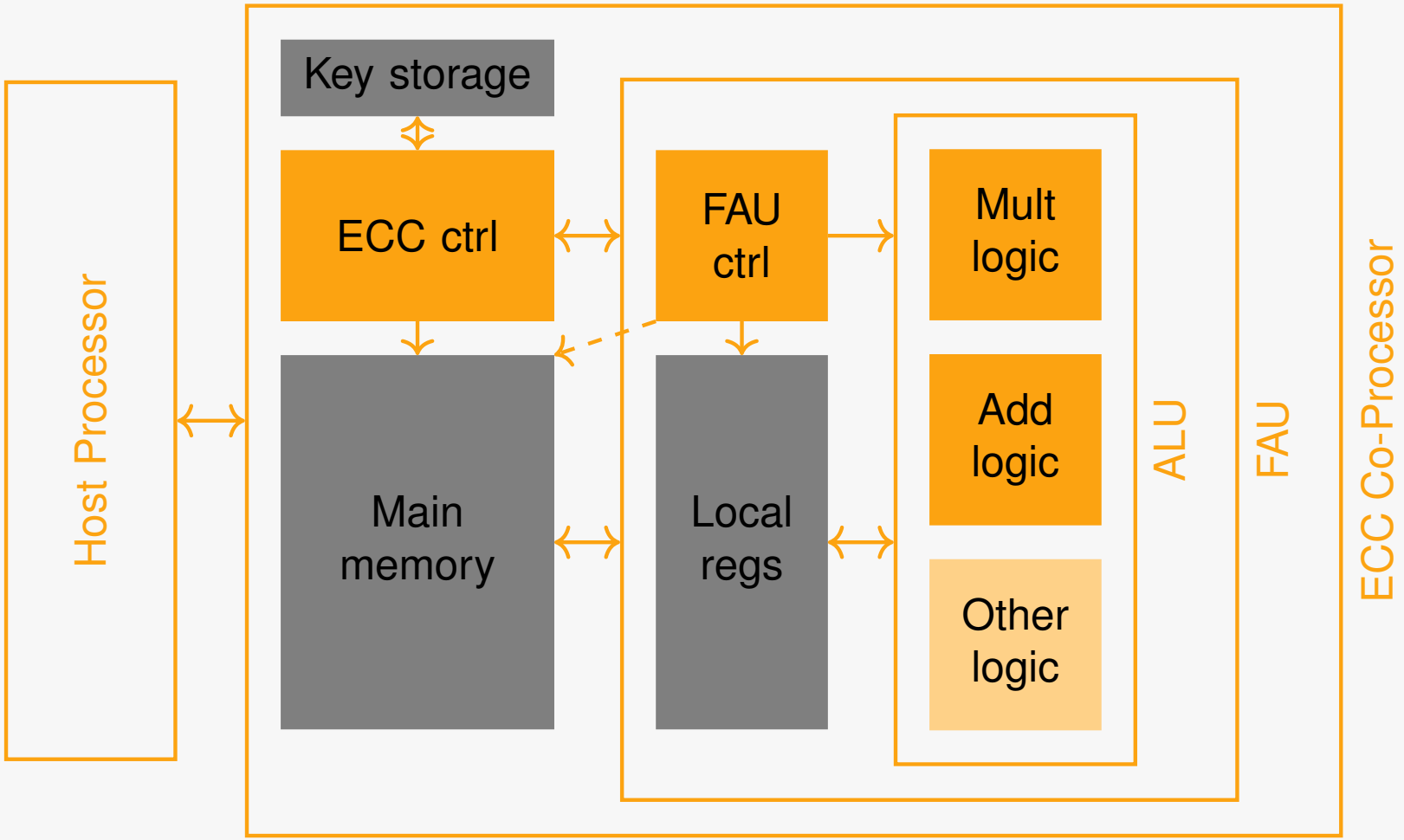


ANATOMY OF ECC HW

ANATOMY OF ECC HW



ANATOMY OF ECC HW



FAST ENDOMORPHISMS

- **GLV/GLS curves** have an efficiently computable endomorphism $\phi(P)$ such that

$$\phi(P) = [\lambda]P$$

Then, scalar multiplication can be computed as:

$$[k]P = [k_0]P + [k_1]\phi(P) \quad \text{where} \quad k_0 + k_1\lambda = k$$

If k_0, k_1 are of the same size, Shamir's trick for double scalar multplication saves about half of the point doublings

- **Koblitz curves** are curves over $\mathbb{F}_{2^m}$ for which $\phi(x, y) = (x^2, y^2)$ is an endomorphism

OVERVIEW OF CHALLENGES

- Fast endomorphisms require recoding of the scalars (e.g., find k_0, k_1)
  $\Rightarrow$ Logic must be added (either a separate converter or FAU instruction set extension)
- The size of the overhead depends on the curve and implementation architecture
  - For binary curves, FAU supports arithmetic over $\mathbb{F}_{2^m}$ but conversions require operations over $\mathbb{Z}$
  - For prime curves, FAU supports arithmetic over $\mathbb{Z}$ but FAU is typically highly optimized for mod$p$ arithmetic

SOFTWARE VS. HARDWARE

Software

+++ Faster scalar multiplications

- Slightly larger program memory and data memory requirements

$\Rightarrow$ Advantages bigger than disadvantages (almost always)

SOFTWARE VS. HARDWARE

Software

+++ Faster scalar multiplications

- Slightly larger program memory and data memory requirements

$\Rightarrow$ Advantages bigger than disadvantages (almost always)

Hardware

++(+) Faster scalar multiplications (almost surely)

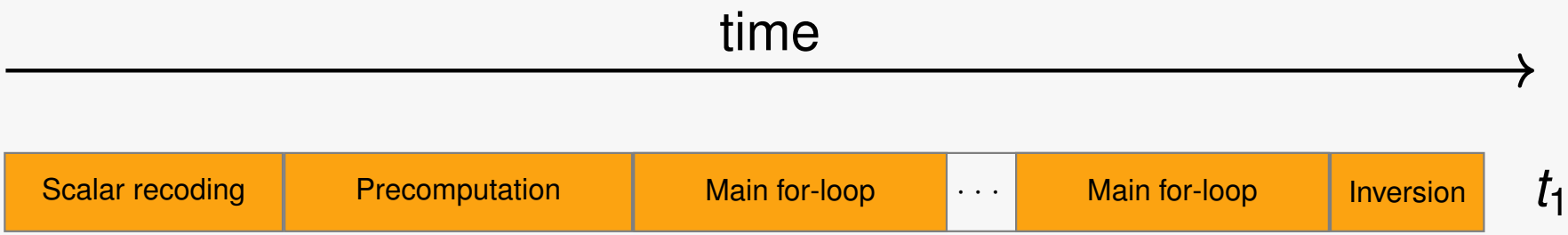
-- More complex control logic

-(-) New instructions needed in FAU

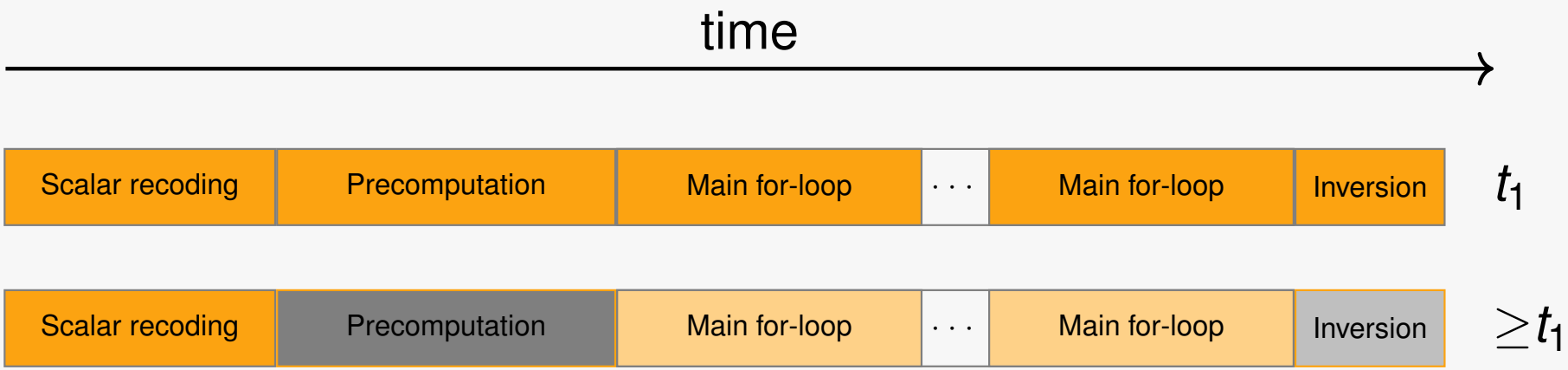
-(--) More memory/registers needed

$\Rightarrow$???

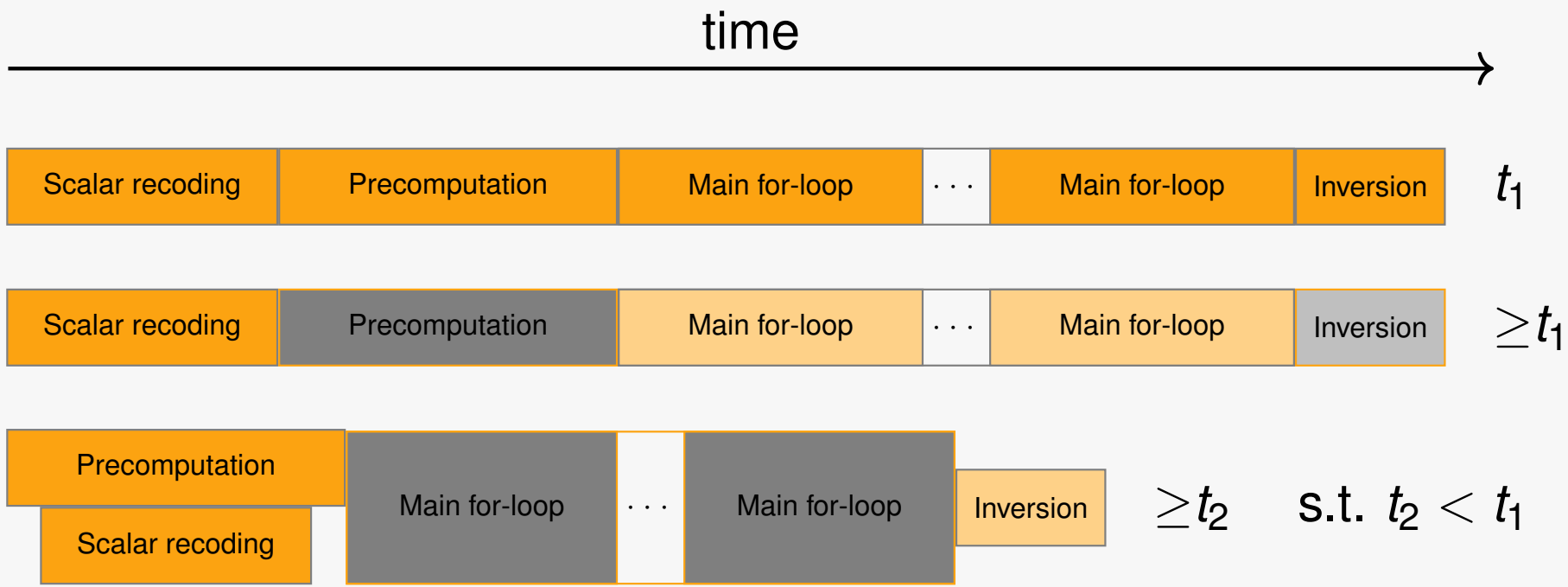
PIPELINING


time
Scalar recoding
Precomputation
Main for-loop
· · ·
Main for-loop
Inversion
t_1

PIPELINING


time
Scalar recoding
Precomputation
Main for-loop
· · ·
Main for-loop
Inversion
t_1
Scalar recoding
Precomputation
Main for-loop
· · ·
Main for-loop
Inversion
$\geq t_1$

PIPELINING


time
Scalar recoding
Precomputation
Main for-loop
· · ·
Main for-loop
Inversion
t_1
Scalar recoding
Precomputation
Main for-loop
· · ·
Main for-loop
Inversion
$\geq t_1$
Precomputation
Scalar recoding
Main for-loop
· · ·
Main for-loop
Inversion
$\geq t_2$
s.t. $t_2 < t_1$

PARALLELISM

- Stages should be balanced because throughput is determined by the slowest stage
- For-loop is by far the slowest stage
- Solutions:
  (a) Make for-loop faster by using more area
  (or make other parts slower and save area)
  (b) Use parallel for-loop units

KOBLITZ CURVES

(Joint work with J. Adikari, B.B. Brumley, V. Dimitrov, S. Sinha Roy, J. Skyttä, and I. Verbauwhede)

KOBLITZ CURVES

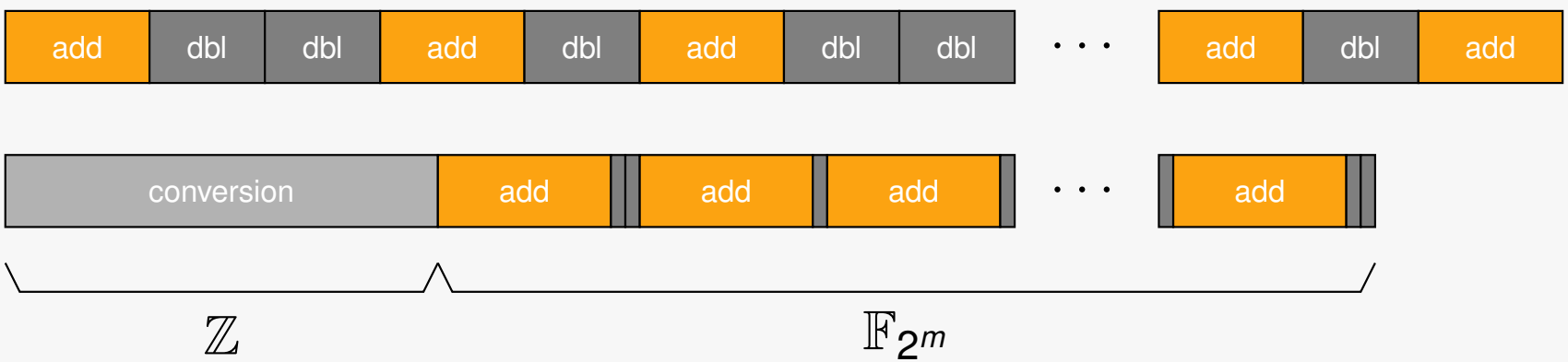
- Binary curves introduced by N. Koblitz already in 1991 and included in many standards (e.g., NIST)

KOBLITZ CURVES

- Binary curves introduced by N. Koblitz already in 1991 and included in many standards (e.g., NIST)
- Cheap **Frobenius maps** $\phi : (x, y) \mapsto (x^2, y^2)$ can be used instead of point doublings

KOBLITZ CURVES

- Binary curves introduced by N. Koblitz already in 1991 and included in many standards (e.g., NIST)
- Cheap **Frobenius maps** $\phi : (x, y) \mapsto (x^2, y^2)$ can be used instead of point doublings
- . . . but first the integer k needs to be given as a **τ-adic expansion** $k = \sum_{i=0}^{\ell-1} k_i \tau^i$ where $\tau = (\mu + \sqrt{-7})/2 \in \mathbb{C}$



SCALAR CONVERSIONS

- Many cryptosystems (e.g., signature schemes) require k also as an integer
  - (a) Select a random integer and find its τ-adic expansion
  - (b) Select a random τ-adic expansion and find its integer equivalent

SCALAR CONVERSIONS

- Many cryptosystems (e.g., signature schemes) require k also as an integer
  - (a) Select a random integer and find its τ-adic expansion
  - (b) Select a random τ-adic expansion and find its integer equivalent
- **Option (a)**
  - Base-τ expansions can be found analogously to finding binary expansions except with divisions by τ instead of 2
  - Straightforward τ-adic expansion of k is twice as long as k
  - Meier and Staffelbach: Because $P = \phi^m(P)$, then $\alpha P = \beta P$ if $\alpha \equiv \beta \pmod{\tau^m - 1}$
  - Solinas: Reduction modulo $(\tau^m - 1)/(\tau - 1)$ gives an expansion of length $m + a$ where $a \in \{0, 1\}$

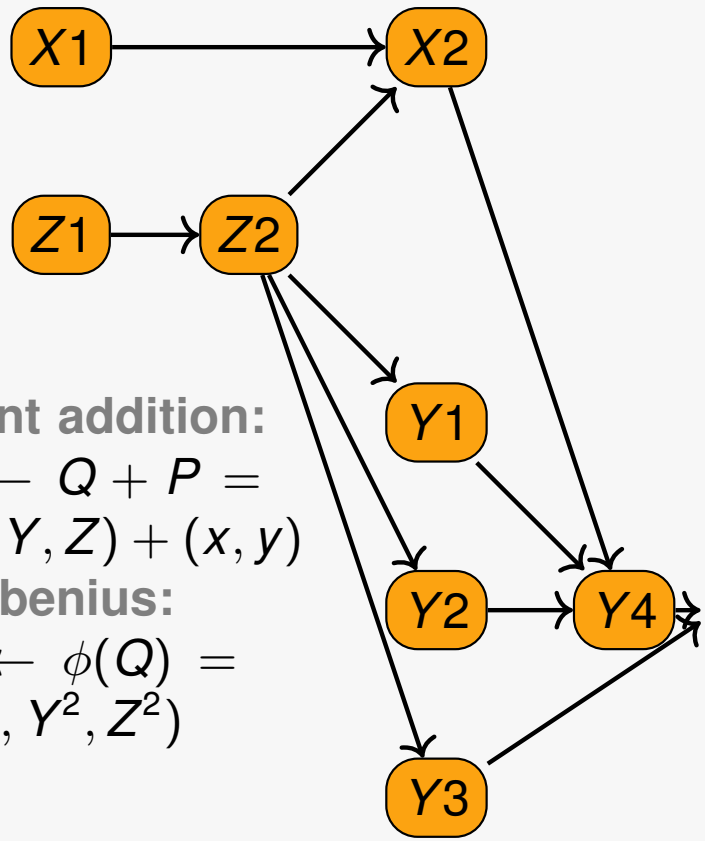
SCALAR CONVERSIONS

- Both require complex operations (e.g., divisions, large multiplications)
- **High-speed implementations:** Avoid conversions from becoming the bottleneck $\Rightarrow$ HW acceleration
- **Lightweight implementations:** Conversions done over $\mathbb{Z}$ $\Rightarrow$ How to combine efficiently with $\mathbb{F}_{2^m}$?
- **Lazy reduction** (repeated divisions by τ) and its many variations (pipelined, word-wise, ...) are commonly used and lead to fast conversions but with an expense in area

HIGH-SPEED IMPLEMENTATION

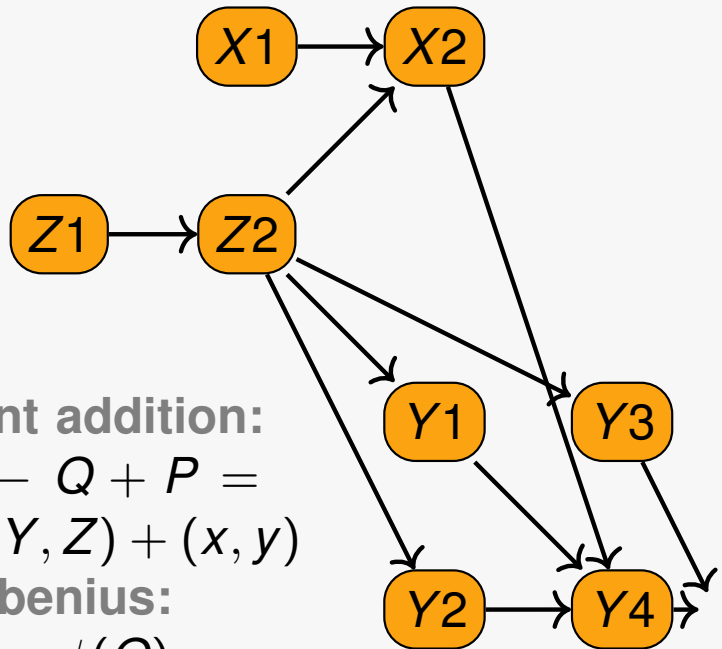
- The key to high speed is to accelerate the main for-loop; other parts can be separated to different pipeline stages
- For-loop consists of point additions and Frobenius maps
- Point additions are dominated by field multiplications (in $\mathbb{F}_{2^m}$)
- Point addition with Lopez-Dahab formulas (SAC'98)
- Frobenius maps $\phi(Q) = (X^2, Y^2, Z^2)$ are cheap and can be computed independently for all coordinates

HIGH-SPEED IMPLEMENTATION



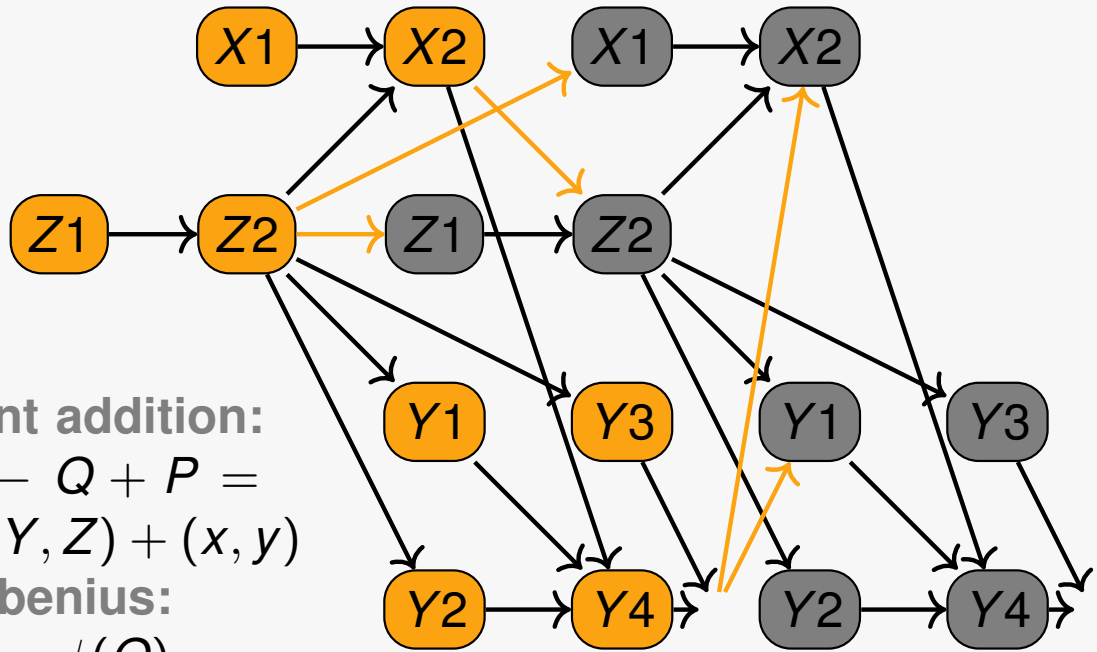
Point addition:
$Q \leftarrow Q + P = (X, Y, Z) + (x, y)$
Frobenius:
$Q \leftarrow \phi(Q) = (X^2, Y^2, Z^2)$

HIGH-SPEED IMPLEMENTATION



Point addition:
$Q \leftarrow Q + P =$
$(X, Y, Z) + (x, y)$
Frobenius:
$Q \leftarrow \phi(Q) =$
(X^2, Y^2, Z^2)

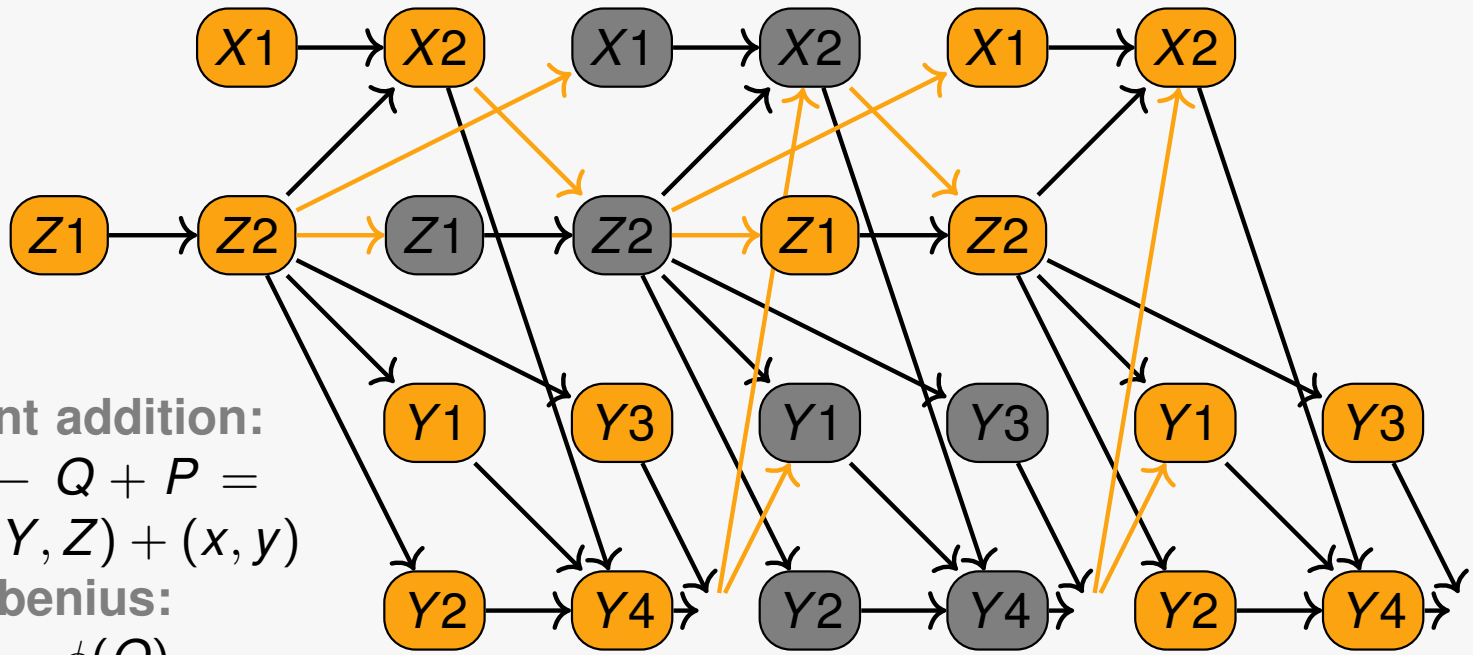
HIGH-SPEED IMPLEMENTATION



Point addition:
$Q \leftarrow Q + P = (X, Y, Z) + (x, y)$

Frobenius:
$Q \leftarrow \phi(Q) = (X^2, Y^2, Z^2)$

HIGH-SPEED IMPLEMENTATION



Point addition:
$Q \leftarrow Q + P = (X, Y, Z) + (x, y)$

Frobenius:
$Q \leftarrow \phi(Q) = (X^2, Y^2, Z^2)$

HIGH-SPEED RESULTS

- The above technique computes the for-loop in less than 5 μs on K-163 or 12 μs on K-283 in a Stratix II FPGA (old)
- One core performs over 200,000 op/s with delay of 11.7 μs
- Multiple cores fit in an FPGA and one device can reach throughputs of several millions
- Delay is not spectacular compared to modern SW but throughput is

COMPACT IMPLEMENTATION

- Koblitz curve K-283
- 16-bit ALU for binary polynomial arithmetic extended with a 16-bit integer adder/subtractor

COMPACT IMPLEMENTATION

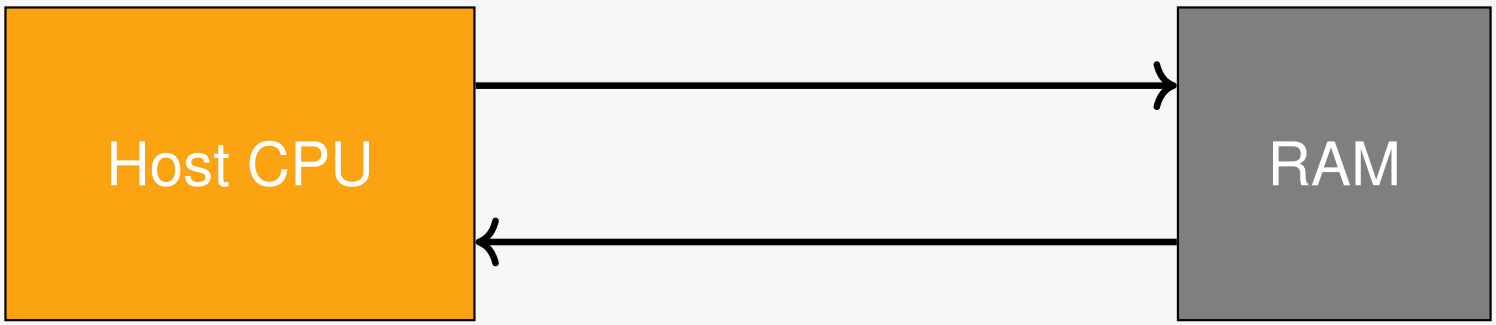
- Koblitz curve K-283
- 16-bit ALU for binary polynomial arithmetic extended with a 16-bit integer adder/subtractor
- Constant-time window algorithm with precomputations
- Conversion (SCA-protected) computed with the lazy reduction

COMPACT IMPLEMENTATION

- Koblitz curve K-283
- 16-bit ALU for binary polynomial arithmetic extended with a 16-bit integer adder/subtractor
- Constant-time window algorithm with precomputations
- Conversion (SCA-protected) computed with the lazy reduction
- Conversion overhead is about 550 GE out of 4323 GE (12.7%)
- Speed-up compared to Montgomery ladder is about 20%
- Memory overhead is significant (about 6000 GE)

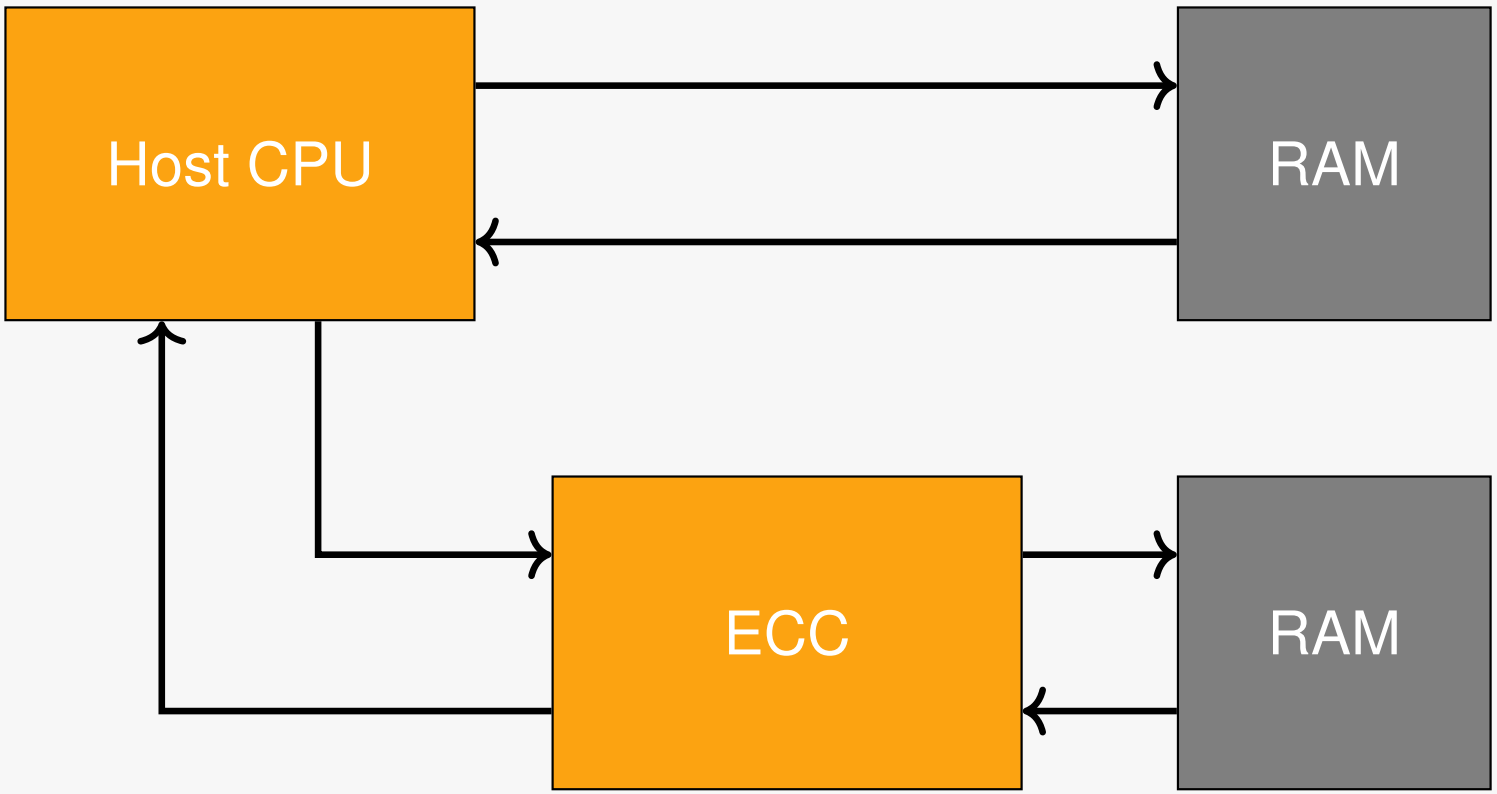
MEMORY SHARING

Wenger, ACNS'13



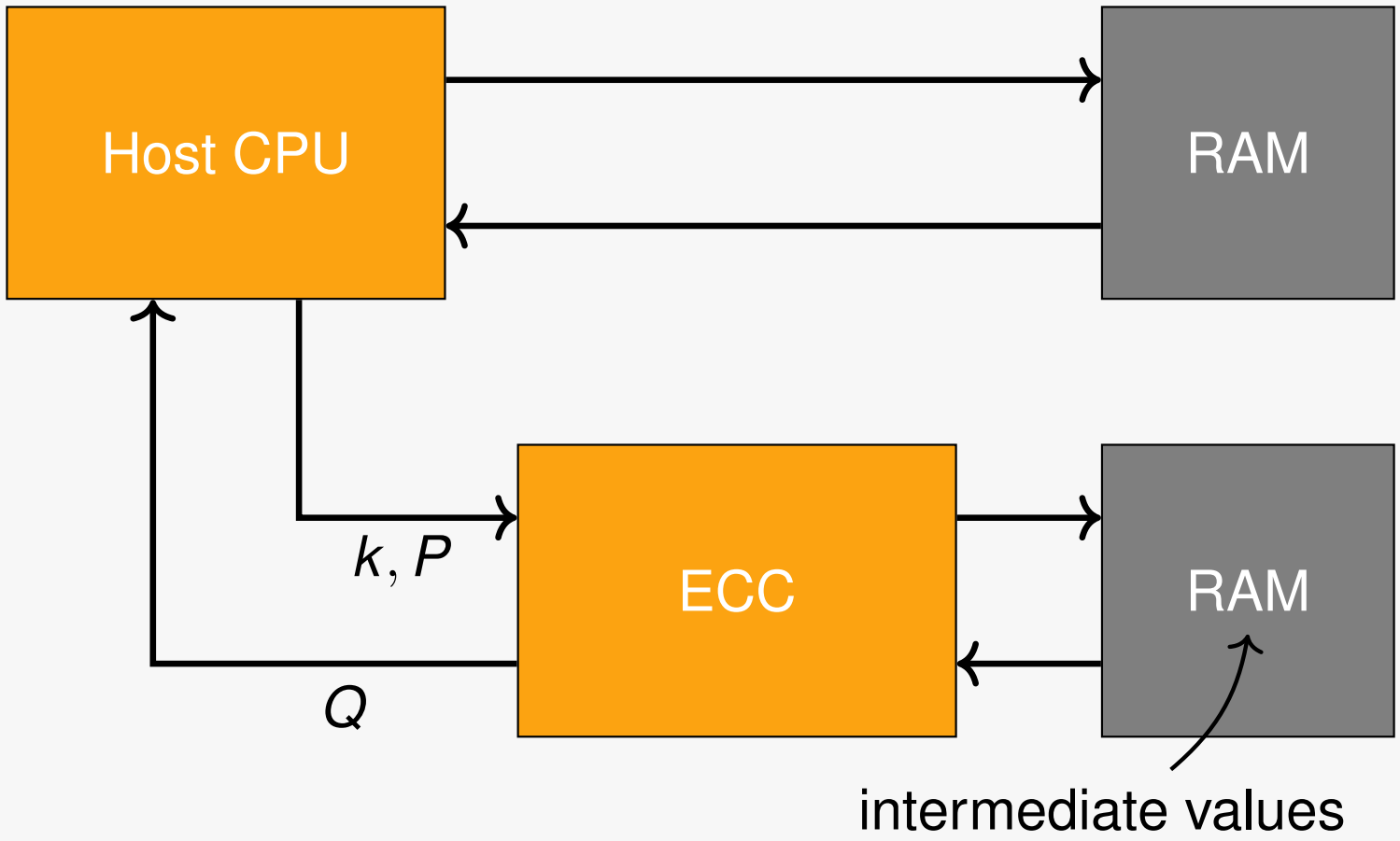
MEMORY SHARING

Wenger, ACNS'13



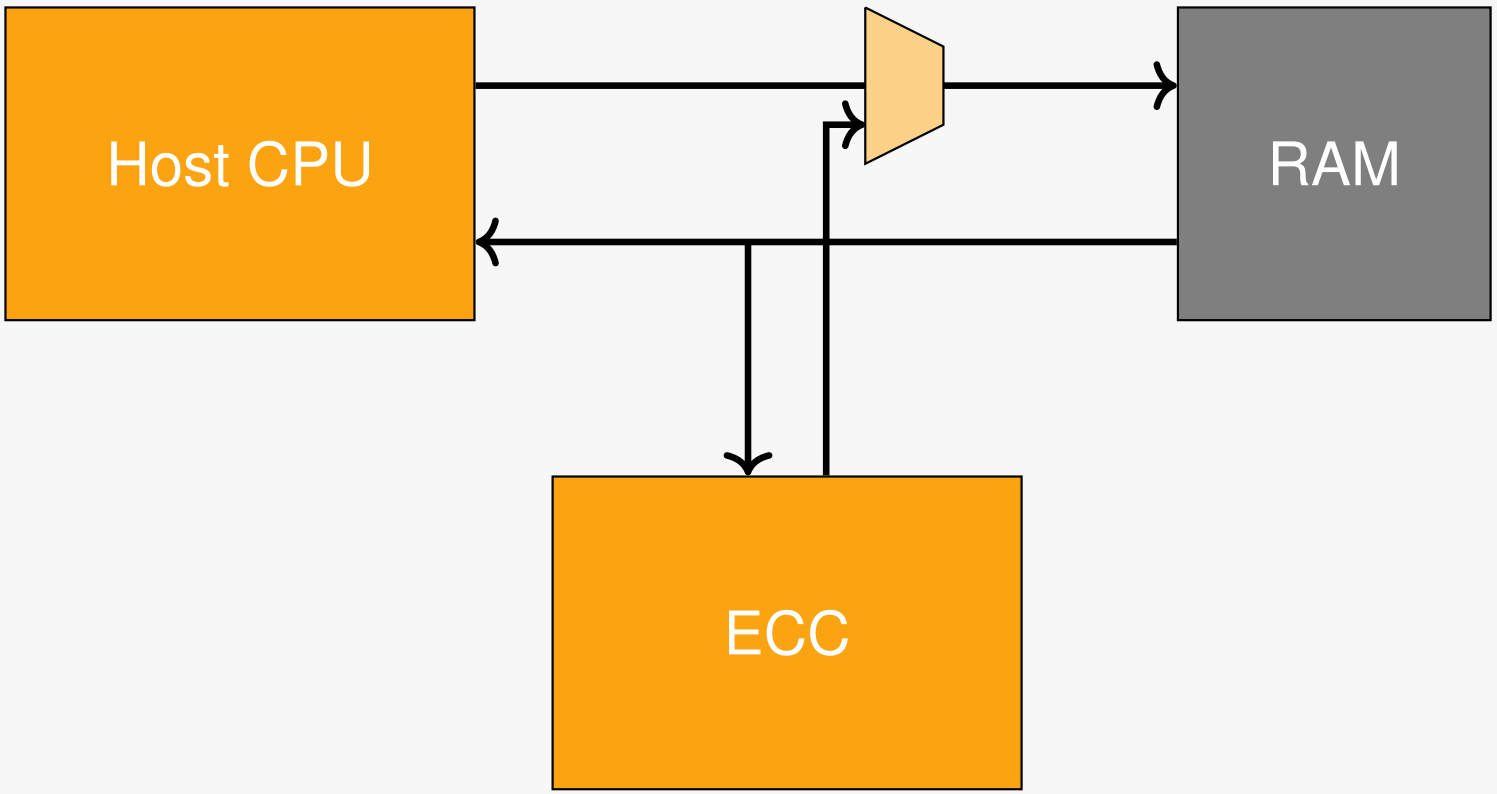
MEMORY SHARING

Wenger, ACNS'13



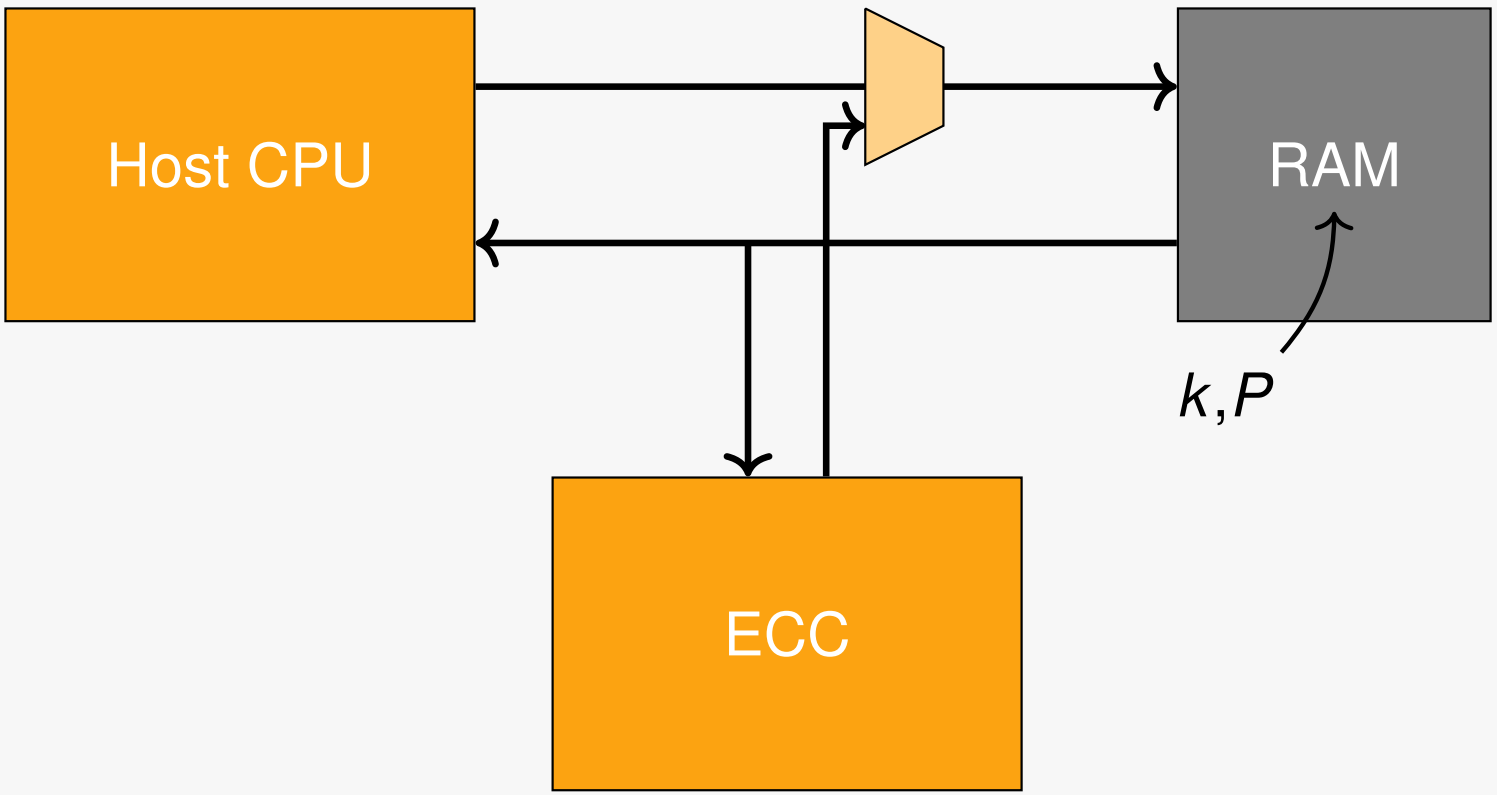
MEMORY SHARING

Wenger, ACNS'13



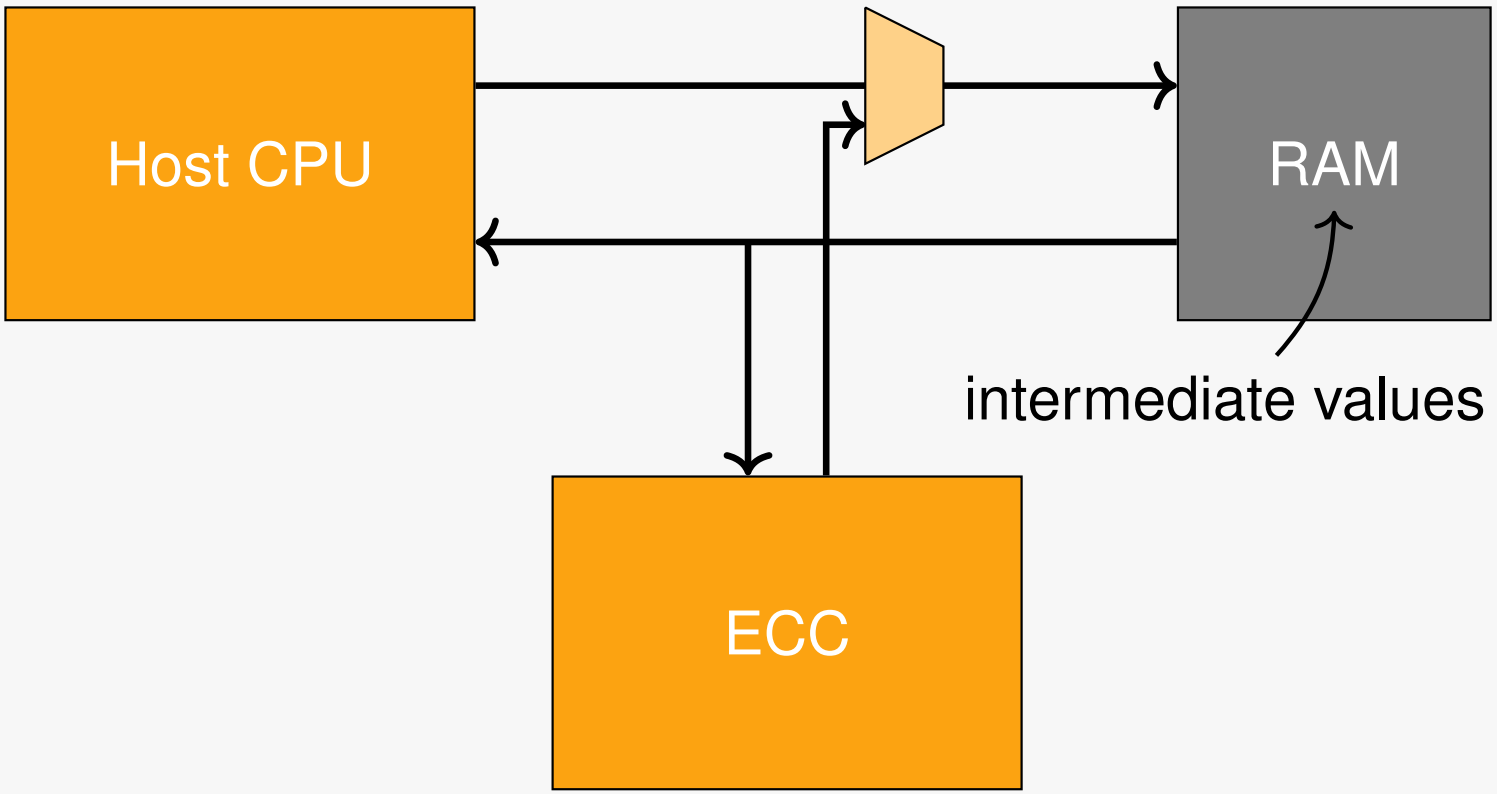
MEMORY SHARING

Wenger, ACNS'13



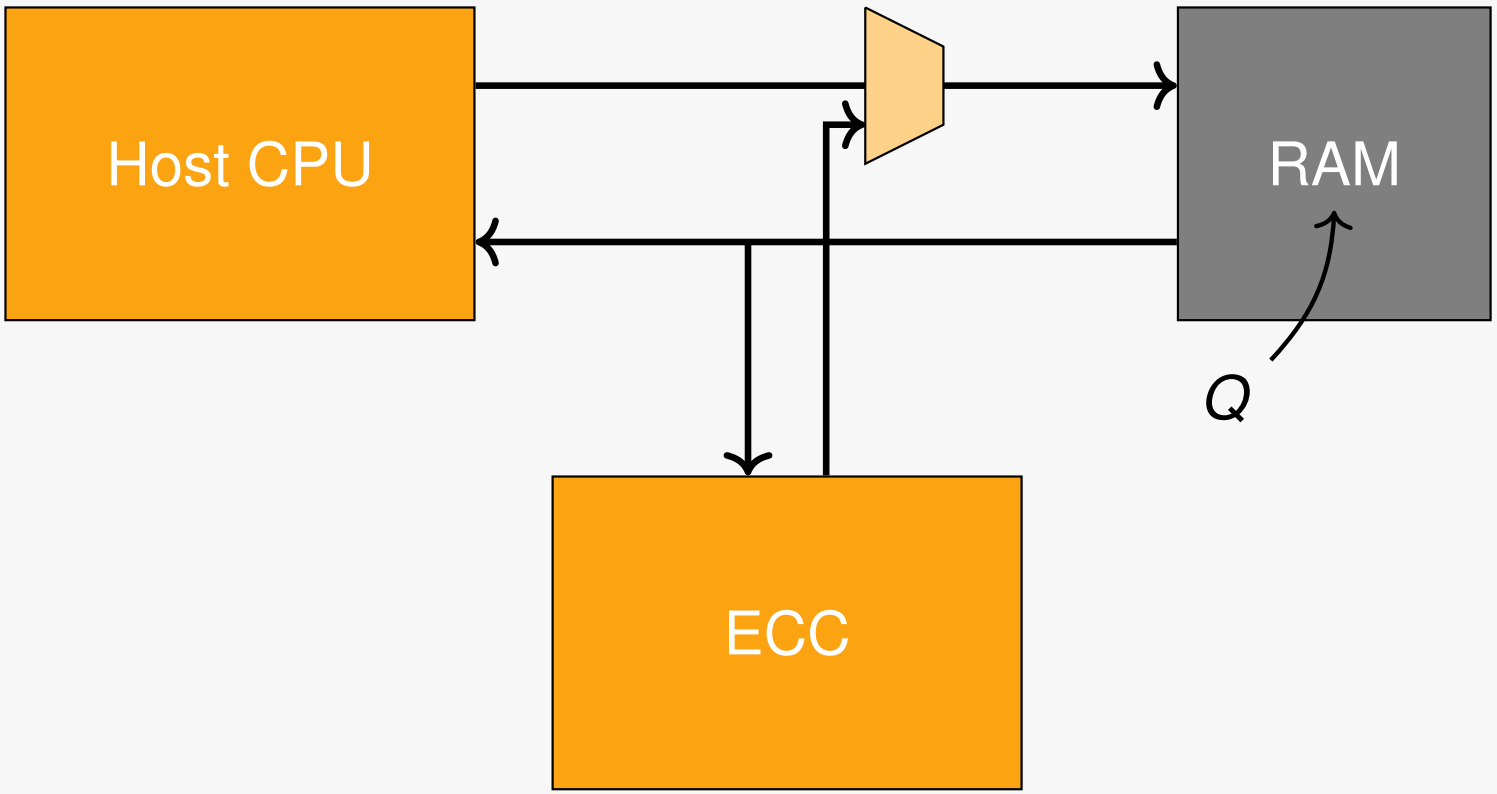
MEMORY SHARING

Wenger, ACNS'13



MEMORY SHARING

Wenger, ACNS'13



COMPACT RESULTS

Work	Curve	RAM	Area (GE)	Latency (cycles)	Latency (ms)	Power (μW)
Batina'06	B-163	no	9,926	95,159	190.32	$<$60
Bock'08	B-163	yes	12,876	–	95	93
Hein'08	B-163	yes	13,250	296,299	2,792	80.85
Kumar'06	B-163	yes	16,207	376,864	27.90	n/a
Lee'08	B-163	yes	12,506	275,816	244.08	32.42
Wegner'11	B-163	yes	8,958	286,000	2,860	32.34
Wegner'13	B-163	no	4,114	467,370	467.37	66.1
Pessl'14	P-160	yes	12,448	139,930	139.93	42.42
Azarderakhsh'14	K-163	yes	11,571	106,700	7.87	5.7
Our, est.	B-163	no	$\approx$3,773	$\approx$485,000	$\approx$30.31	$\approx$6.11
Our, est.	K-163	no	$\approx$4,323	$\approx$420,900	$\approx$26.30	$\approx$6.11
Our, est.	B-283	no	$\approx$3,773	$\approx$1,934,000	$\approx$120.89	$\approx$6.11
Our, est.	K-283	yes*	10,204*	1,566,000	97.89	$>$6.11
Our	**K-283**	**no**	**4,323**	**1,566,000**	**97.89**	**6.11**

$\star$ Estimate for a 256 $\times$ 16-bit RAM, space needed for 252 16-bit words (4032 bits)

SCA COUNTERMEASURE

Okeya, Takagi, Vuillaume ACISP'05

- Montgomery ladder is unavailable for Koblitz curves (without losing the performance advantage of Frobenius)

SCA COUNTERMEASURE

Okeya, Takagi, Vuillaume ACISP'05

- Montgomery ladder is unavailable for Koblitz curves (without losing the performance advantage of Frobenius)
- Constant timing can be achieved with zero-free representations
  - Recode $00\ldots01$ with $1\bar{1}\ldots\bar{1}\bar{1}$
  - Precompute all $\phi^{w-1}(P) + a_{w-2}\phi^{w-2}(P) + \ldots + a_0 P$ with $a_i \in \{-1, 1\}$
  - Scan through the recoded scalar with a w-bit fixed window

SCA COUNTERMEASURE

Okeya, Takagi, Vuillaume ACISP'05

- Montgomery ladder is unavailable for Koblitz curves (without losing the performance advantage of Frobenius)
- Constant timing can be achieved with zero-free representations
  - Recode $00\ldots01$ with $1\bar{1}\ldots\bar{1}\bar{1}$
  - Precompute all $\phi^{w-1}(P) + a_{w-2}\phi^{w-2}(P) + \ldots + a_0 P$ with $a_i \in \{-1, 1\}$
  - Scan through the recoded scalar with a w-bit fixed window

$$1\bar{1}\bar{1}111\bar{1}111\bar{1}\bar{1}\bar{1}\ldots 1\bar{1}11$$
$$\vee$$
$$+P_{-1}$$

$w = 2$:

$$P_{+1} = \phi(P) + P$$
$$P_{-1} = \phi(P) - P$$

SCA COUNTERMEASURE

Okeya, Takagi, Vuillaume ACISP'05

- Montgomery ladder is unavailable for Koblitz curves (without losing the performance advantage of Frobenius)
- Constant timing can be achieved with zero-free representations
  - Recode $00\ldots01$ with $1\bar{1}\ldots\bar{1}\bar{1}$
  - Precompute all $\phi^{w-1}(P) + a_{w-2}\phi^{w-2}(P) + \ldots + a_0 P$ with $a_i \in \{-1, 1\}$
  - Scan through the recoded scalar with a w-bit fixed window

$$1\bar{1}\bar{1}1111\bar{1}111\bar{1}\bar{1}\bar{1}\ldots 1\bar{1}11$$

ϕ^2 (curved arrow over the first four digits)

$+P_{-1}$ (arrow under the last digit)

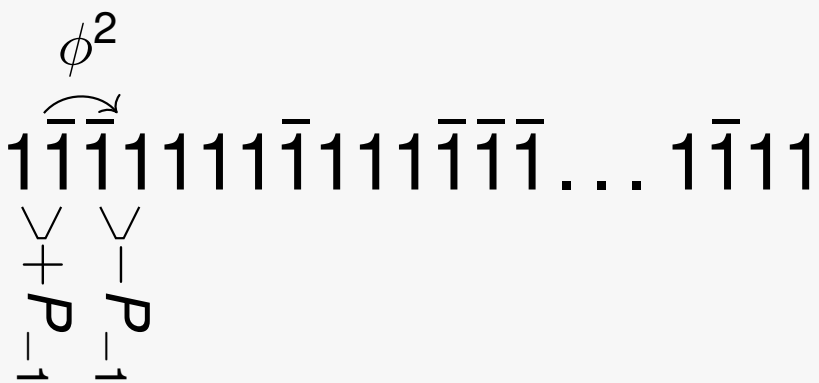
$w = 2$:

$$P_{+1} = \phi(P) + P$$
$$P_{-1} = \phi(P) - P$$

SCA COUNTERMEASURE

Okeya, Takagi, Vuillaume ACISP'05

- Montgomery ladder is unavailable for Koblitz curves (without losing the performance advantage of Frobenius)
- Constant timing can be achieved with zero-free representations
  - Recode $00\ldots01$ with $1\bar{1}\ldots\bar{1}\bar{1}$
  - Precompute all $\phi^{w-1}(P) + a_{w-2}\phi^{w-2}(P) + \ldots + a_0 P$ with $a_i \in \{-1, 1\}$
  - Scan through the recoded scalar with a w-bit fixed window



$w = 2$:

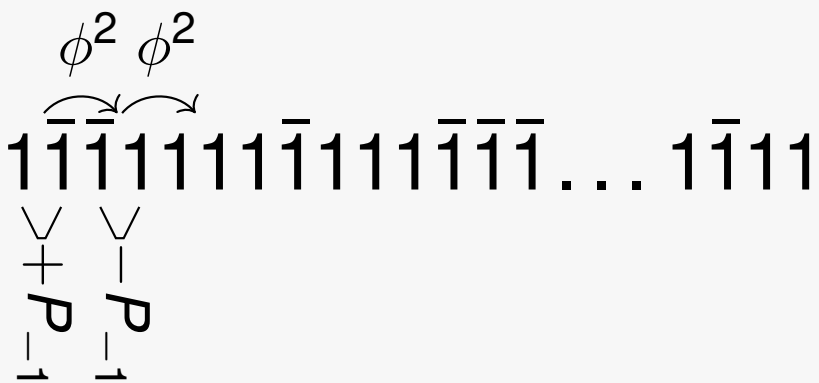
$P_{+1} = \phi(P) + P$

$P_{-1} = \phi(P) - P$

SCA COUNTERMEASURE

Okeya, Takagi, Vuillaume ACISP'05

- Montgomery ladder is unavailable for Koblitz curves (without losing the performance advantage of Frobenius)
- Constant timing can be achieved with zero-free representations
  - Recode $00\ldots01$ with $1\bar{1}\ldots\bar{1}\bar{1}$
  - Precompute all $\phi^{w-1}(P) + a_{w-2}\phi^{w-2}(P) + \ldots + a_0 P$ with $a_i \in \{-1, 1\}$
  - Scan through the recoded scalar with a w-bit fixed window



$w = 2$:

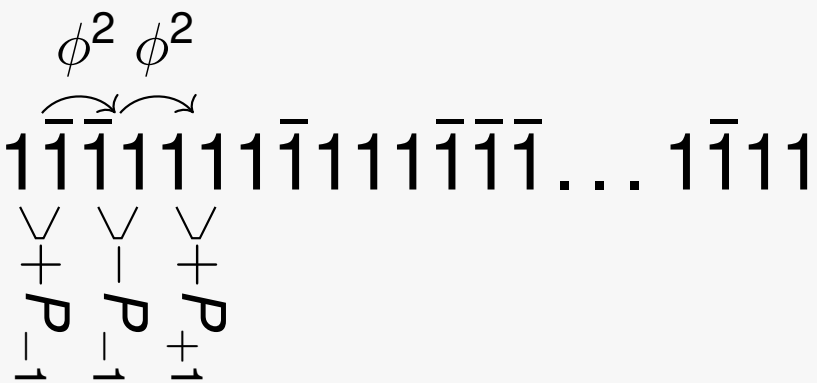
$P_{+1} = \phi(P) + P$

$P_{-1} = \phi(P) - P$

SCA COUNTERMEASURE

Okeya, Takagi, Vuillaume ACISP'05

- Montgomery ladder is unavailable for Koblitz curves (without losing the performance advantage of Frobenius)
- Constant timing can be achieved with zero-free representations
  - Recode $00\ldots01$ with $1\bar{1}\ldots\bar{1}\bar{1}$
  - Precompute all $\phi^{w-1}(P) + a_{w-2}\phi^{w-2}(P) + \ldots + a_0 P$ with $a_i \in \{-1, 1\}$
  - Scan through the recoded scalar with a w-bit fixed window



$w = 2$:

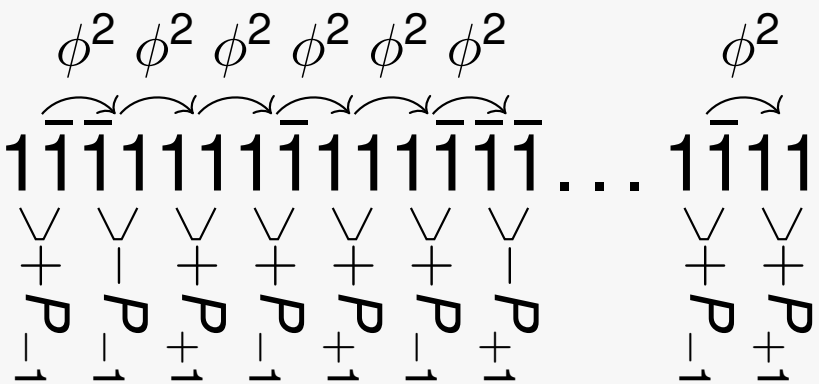
$P_{+1} = \phi(P) + P$

$P_{-1} = \phi(P) - P$

SCA COUNTERMEASURE

Okeya, Takagi, Vuillaume ACISP'05

- Montgomery ladder is unavailable for Koblitz curves (without losing the performance advantage of Frobenius)
- Constant timing can be achieved with zero-free representations
  - Recode $00\ldots01$ with $1\bar{1}\ldots\bar{1}\bar{1}$
  - Precompute all $\phi^{w-1}(P) + a_{w-2}\phi^{w-2}(P) + \ldots + a_0 P$ with $a_i \in \{-1, 1\}$
  - Scan through the recoded scalar with a w-bit fixed window



$w = 2$:

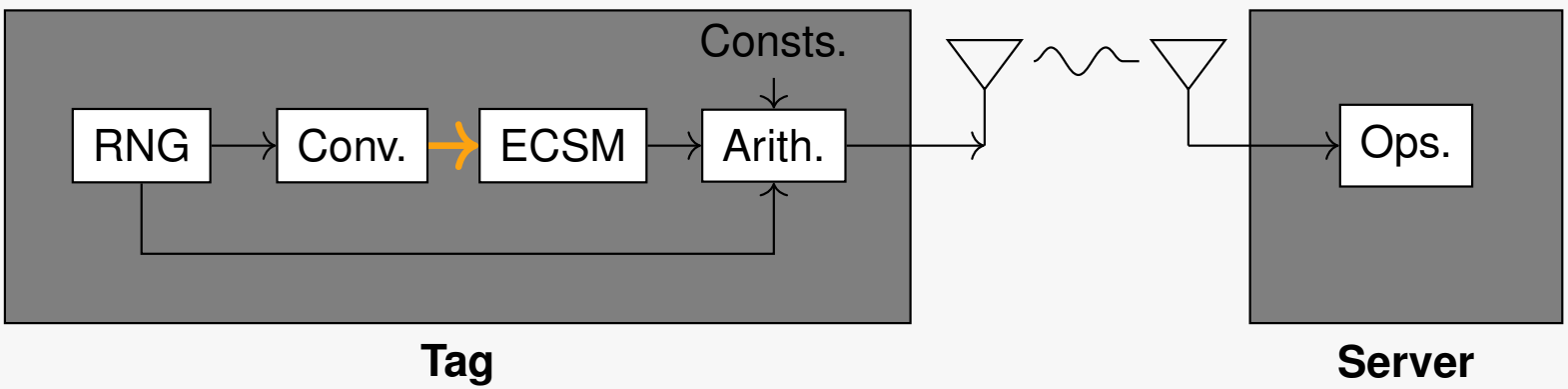
$P_{+1} = \phi(P) + P$

$P_{-1} = \phi(P) - P$

OUTSOURCED CONVERSIONS

- Operations computed with the scalar k are typically simple
- Delegate conversions to a more powerful party so that the weaker party computes operations in the τ-adic domain
- τ-adic operations can be implemented with a small instruction set extension (less than 100GE at minimum)

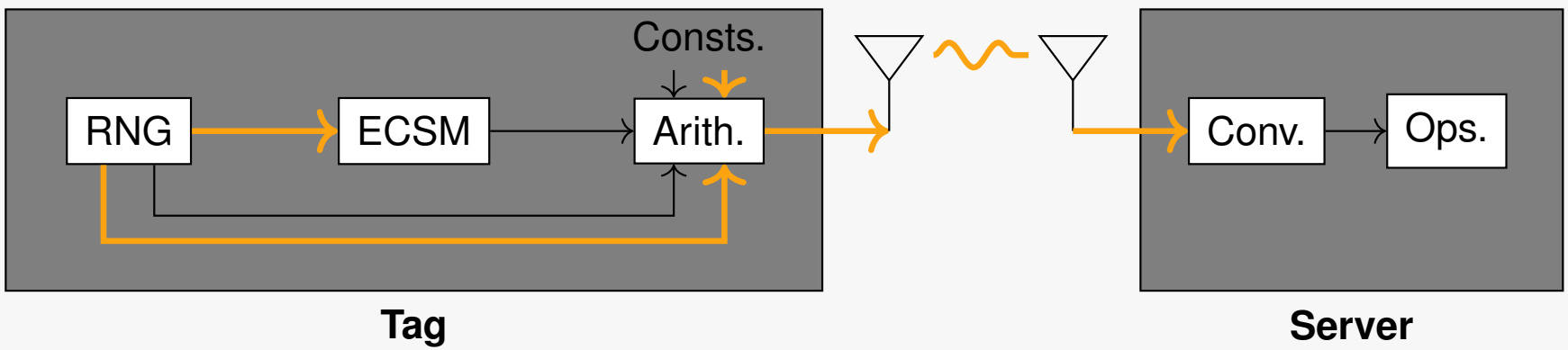
With conversion:



OUTSOURCED CONVERSIONS

- Operations computed with the scalar k are typically simple
- Delegate conversions to a more powerful party so that the weaker party computes operations in the τ-adic domain
- τ-adic operations can be implemented with a small instruction set extension (less than 100GE at minimum)

Outsourced conversions:



FOURQ

(Joint work with R. Azarderakhsh, P. Longa and A. Miele)

FOUR$\mathbb{Q}$

Costello, Longa, ASIACRYPT'15

- **Twisted Edwards curve** with $\#E(\mathbb{F}_{p^2}) = 392 \cdot \xi$ where ξ is a 246-bit prime
- Defined over $\mathbb{F}_{p^2}$ with the **Mersenne prime** $p = 2^{127} - 1$
- **Complete addition formulas** over extended twisted Edwards coordinates (Hisil et al. ASIACRYPT'08)

FOUR$\mathbb{Q}$

Costello, Longa, ASIACRYPT'15

- **Twisted Edwards curve** with $\#E(\mathbb{F}_{p^2}) = 392 \cdot \xi$ where ξ is a 246-bit prime
- Defined over $\mathbb{F}_{p^2}$ with the **Mersenne prime** $p = 2^{127} - 1$
- **Complete addition formulas** over extended twisted Edwards coordinates (Hisil et al. ASIACRYPT'08)
- **Two efficiently-computable endomorphisms** ψ and ϕ
- **Four-dimensional decomposition** for the scalar 256-bit scalar m with (a_1, a_2, a_3, a_4) such that $a_i \in [0, 2^{64})$:

$$[m]P = [a_1]P + [a_2]\psi(P) + [a_3]\phi(P) + [a_4]\psi(\phi(P))$$

SCALAR MULTIPLICATION

Input: Point P,
integer $m \in [0, 2^{256})$
Output: $[m]P$
Decompose and recode m
Precompute lookup table T
$Q \leftarrow T[v_{64}]$
for $i = 63$ **to** 0 **do**
$Q \leftarrow [2]Q$
$Q \leftarrow Q + m_i T[v_i]$

SCALAR MULTIPLICATION

Input: Point P,
integer $m \in [0, 2^{256})$
Output: $[m]P$
Decompose and recode m
Precompute lookup table T
$Q \leftarrow T[v_{64}]$
for $i = 63$ **to** 0 **do**
$Q \leftarrow [2]Q$
$Q \leftarrow Q + m_i T[v_i]$

Scalar decompose and recode

- Decompose to a multi-scalar (a_1, a_2, a_3, a_4) with 65-bit a_i
- Sign-aligned so that $a_1[j] \in \{\pm 1\}$ and $a_i[j] \in \{0, a_1[j]\}$ for $2 \leq j \leq 4$
- Recode to signs $m_i \in \{-1, 1\}$ and values $v_i \in [0, 7]$ (point index)

SCALAR MULTIPLICATION

Input: Point P,
integer $m \in [0, 2^{256})$
Output: $[m]P$
Decompose and recode m
Precompute lookup table T
$Q \leftarrow T[v_{64}]$
for $i = 63$ **to** 0 **do**
$Q \leftarrow [2]Q$
$Q \leftarrow Q + m_i T[v_i]$

Precomputation

- Precompute 8 points: $T[u] = P + [u_0]\phi(P) + [u_1]\psi(P) + [u_2]\psi(\phi(P))$ for $u = (u_2, u_1, u_0) \in [0, 7]$
- Store them with 5 coordinates $(X + Y, Y - X, 2Z, 2dT, -2dT) \Rightarrow$
$+T[u] : (X + Y, Y - X, 2Z, 2dT)$
$-T[u] : (Y - X, X + Y, 2Z, -2dT)$
- $68\mathbf{M} + 27\mathbf{S}$ and several additions

SCALAR MULTIPLICATION

Input: Point P,
integer $m \in [0, 2^{256})$
Output: $[m]P$
Decompose and recode m
Precompute lookup table T
$Q \leftarrow T[v_{64}]$
for $i = 63$ **to** 0 **do**
$Q \leftarrow [2]Q$
$Q \leftarrow Q + m_i T[v_i]$

Main for-loop

- Fully regular and constant-time
- Only 64 double-and-adds
- Hisil et al. (ASIACRYPT '08)
- Doubling:
$(X, Y, Z, T_a, T_b) \leftarrow (X, Y, Z)$
- Addition:
$(X, Y, Z, T_a, T_b) \leftarrow$
$(X, Y, Z, T_a, T_b) \times$
$(X + Y, Y - X, 2Z, 2dT)$

SCALAR MULTIPLICATION

Input: Point P,
integer $m \in [0, 2^{256})$
Output: $[m]P$
Decompose and recode m
Precompute lookup table T
$Q \leftarrow T[v_{64}]$
for $i = 63$ **to** 0 **do**
$Q \leftarrow [2]Q$
$Q \leftarrow Q + m_i T[v_i]$

Scalar Unit

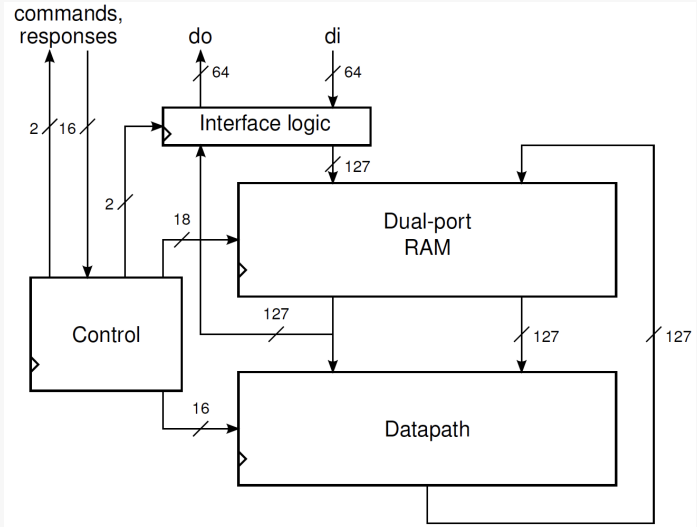
- Decomposes and recodes the scalar
- Mainly multiplications with constants

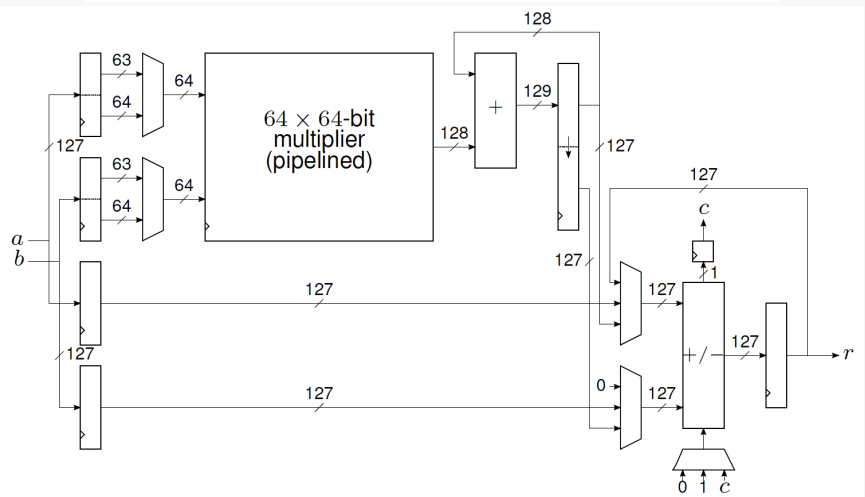
Field Arithmetic Unit

- Precomputation and the main for-loop
- Highly optimized for $\mathbb{F}_p$ with the Mersenne prime

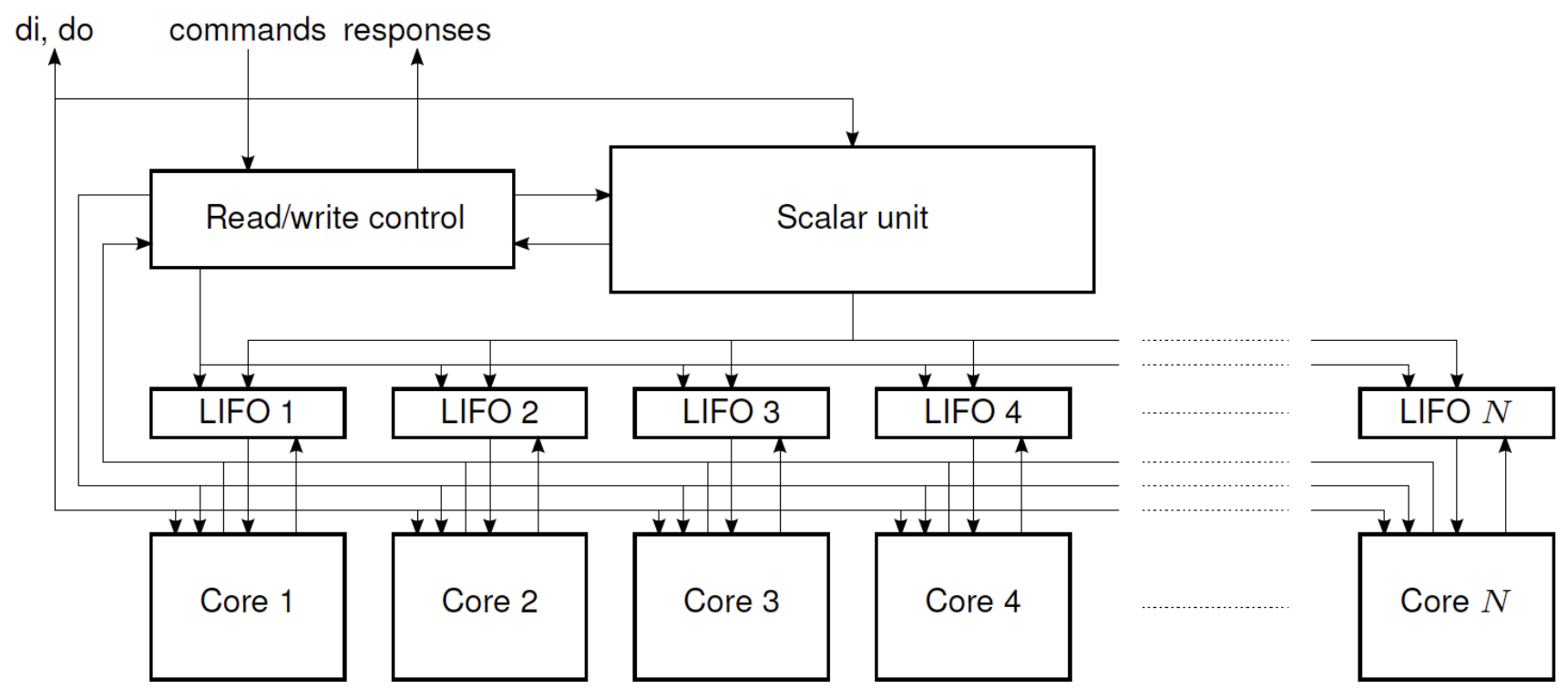
ARCHITECTURE

- Extensive use of FPGAs DSP and BRAM blocks
- Deep pipeline
- Core computes precomputations, for-loop, and inversion
- Multicore architecture shares the converter with 11 cores



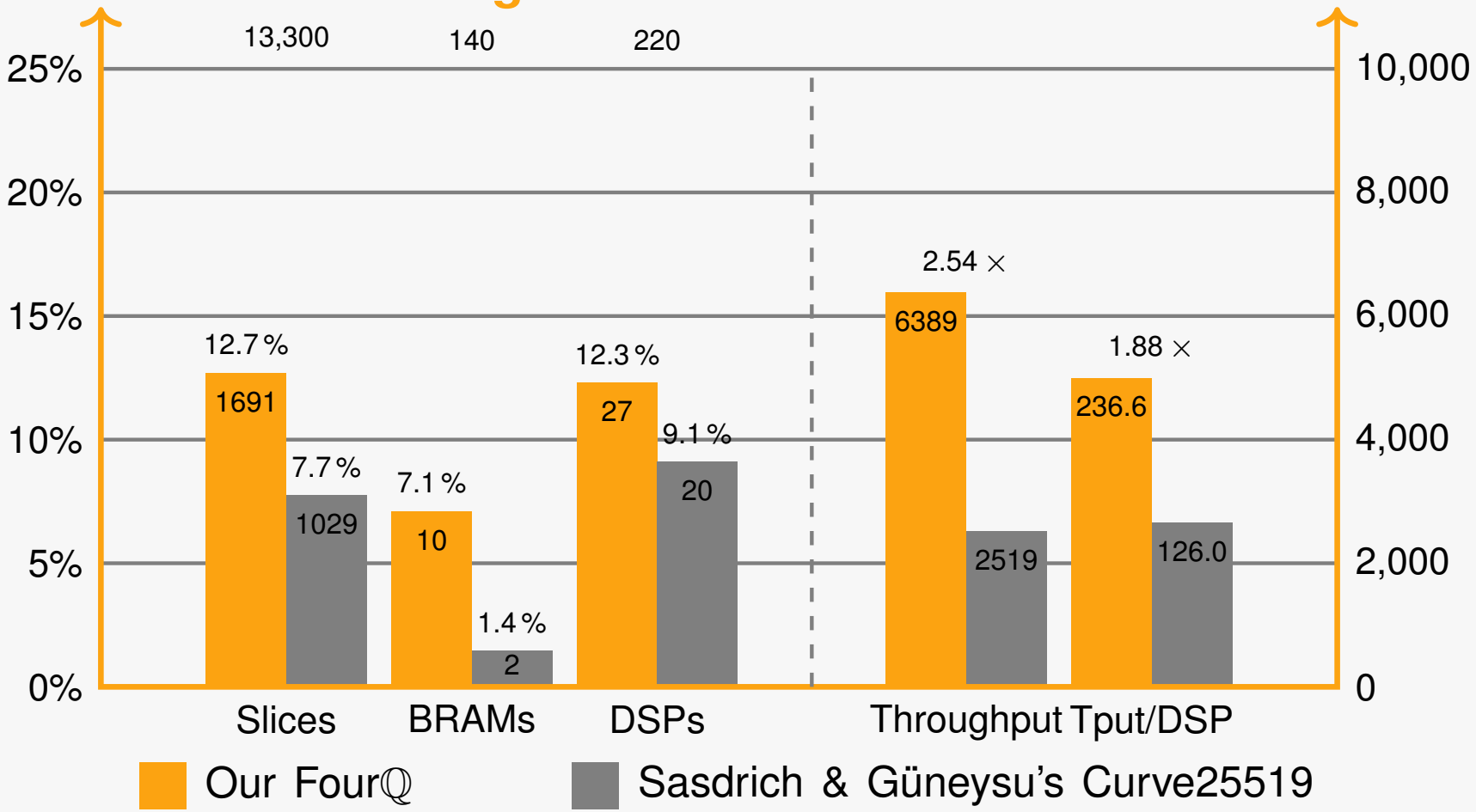


MULTI-CORE ARCHITECTURE



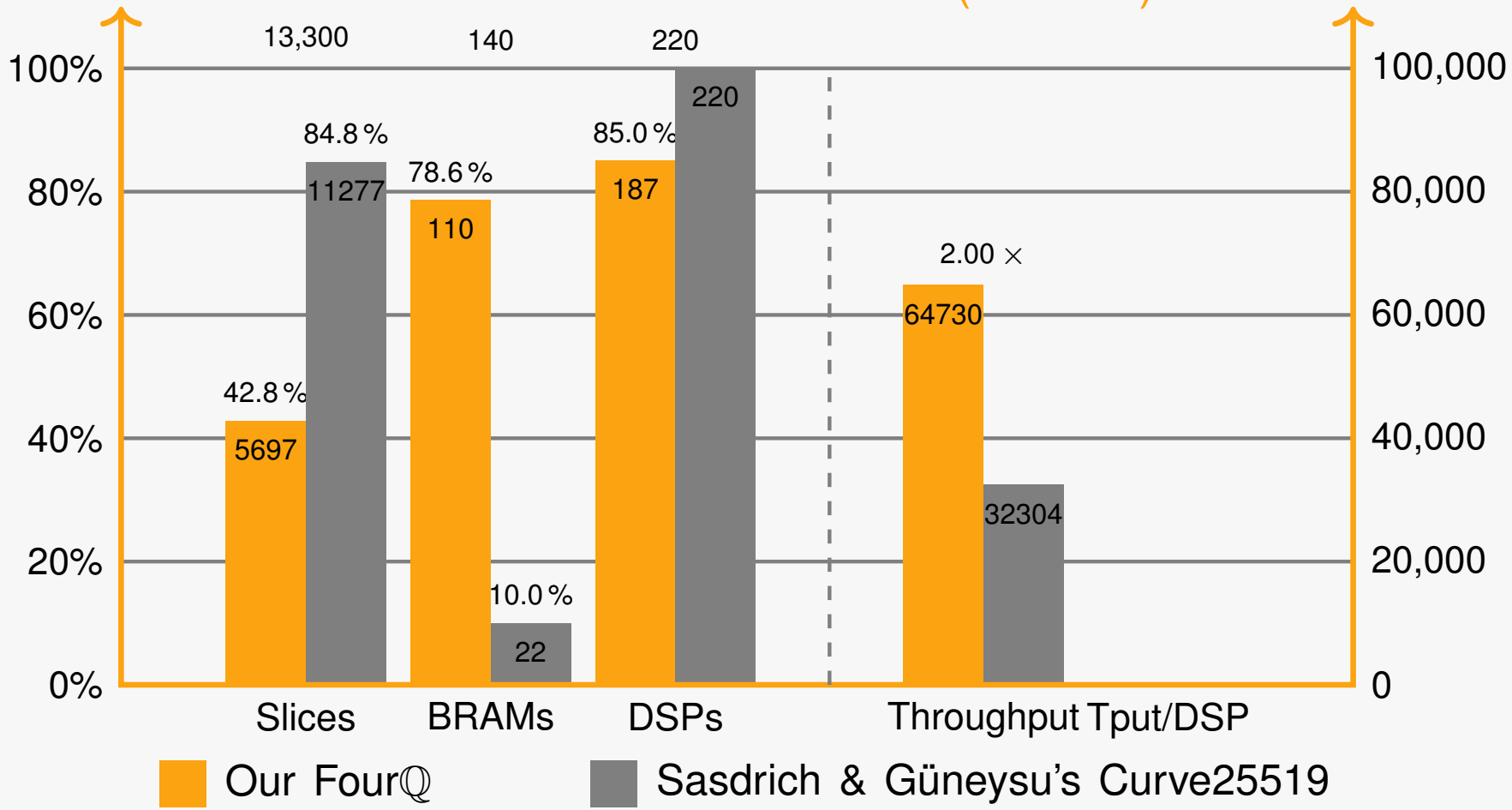
FOURQ vs. CURVE25519

Single-Core Architectures



FOUR$\mathbb{Q}$ vs. CURVE25519

Multi-Core Architectures ($N = 11$)



GLV/GLS

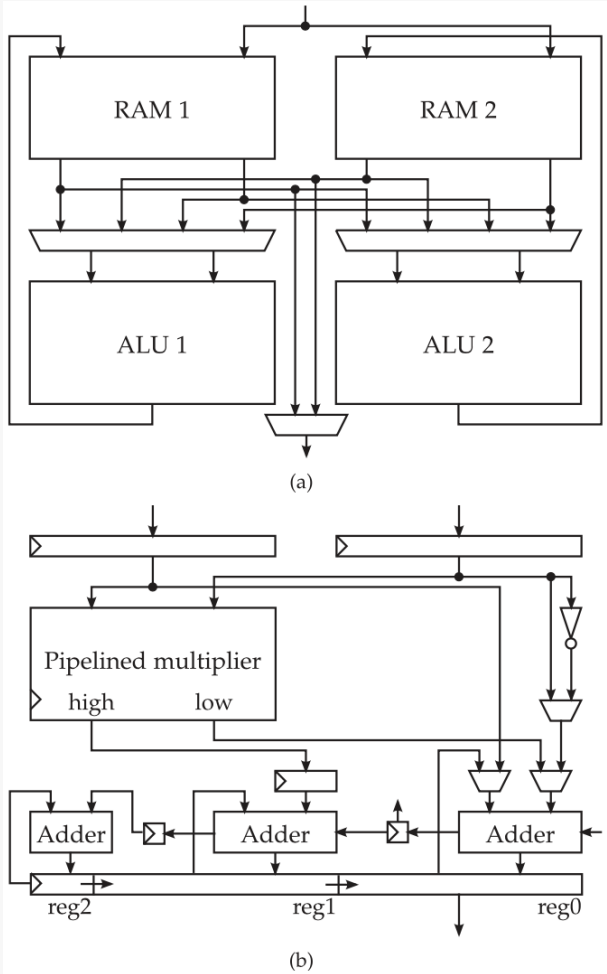
(Joint work with K. Aerts, B. Gövem, J. Großschädl, Z. Hu, Z. Liu, N. Mentens, I. Verbauwhede, H. Wang)

GLS CURVE OVER $\mathbb{F}_{2^{254}}$

- We designed a fast and compact ECC coprocessor for a binary GLS curve
- Operations in $\mathbb{F}_{2^{254}}$ are computed with operations in $\mathbb{F}_{2^{127}}$ and with the λ-coordinates
- Scalar recoding with an 8-bit ALU, field operations with digit-serial ($d = 16$) MALU for $\mathbb{F}_{2^{127}}$
- Recoding unit takes roughly 1/4 of the area

SIGNATURE VERIFICATION

- Signature verification requires $[k]G + [l]Q$ which becomes $[k_0]G + [k_1]\phi(G) + [l_0]Q + [l_1]\phi(Q)$
- Precomputation of 15 points
- Architecture (right) computes Hisil et al.'s point formulas efficiently with two parallel ALUs
- One core computes over 2000 op/s on curve over $\mathbb{F}_p$ with $p = 2^{207} - 5131$ with only 1.8% of DSPs in Virtex-7 $\Rightarrow$ One FPGA reaches even 100,000 op/s



CONCLUSION

- Fast endomorphisms can be used for efficient ECC in hardware also
- Requires careful optimizations
- Slight increase in resource requirements leads to large increase in speed
- Pipelining offers high throughput

CONCLUSION

- Fast endomorphisms can be used for efficient ECC in hardware also
- Requires careful optimizations
- Slight increase in resource requirements leads to large increase in speed
- Pipelining offers high throughput

THANK YOU! QUESTIONS?

LITERATURE

This presentation was based particularly on the following papers:

- Zhe Liu, Johann Großschädl, Zhi Hu, Kimmo Järvinen, Husen Wang, and Ingrid Verbauwhede. ‘”Elliptic Curve Cryptography with Efficiently Computable Endomorphisms and Its Hardware Implementations for the Internet of Things”. IEEE Transactions on Computers 66.5 (May 2017), pp. 773–785. https://doi.org/10.1109/TC.2016.2623609.
- Kimmo Järvinen, Andrea Miele, Reza Azarderakhsh, and Patrick Longa. “Four$\mathbb{Q}$ on FPGA: New Hardware Speed Records for Elliptic Curve Cryptography over Large Prime Characteristic Fields”. In: Proceedings of the IACR Conference on Cryptographic Hardware and Embedded Systems (CHES 2016). Vol. 9813. Lecture Notes in Computer Science. Springer, 2016, pp. 517–537. http://dx.doi.org/10.1007/978-3-662-53140-2_25.

LITERATURE (CONT.)

- Burak Gövem, Kimmo Järvinen, Kris Aerts, Ingrid Verbauwhede, and Nele Mentens. “A Fast and Compact FPGA Implementation of Elliptic Curve Cryptography Using Lambda Coordinates”. In: Progress in Cryptology (AFRICACRYPT 2016). Vol. 9646. Lecture Notes in Computer Science. Springer, 2016, pp. 63–68. http://dx.doi.org/10.1007/978-3-319-31517-1_4.
- Sujoy Sinha Roy, Kimmo Järvinen, and Ingrid Verbauwhede. “Lightweight Coprocessor for Koblitz Curves: 283-bit ECC Including Scalar Conversion with only 4300 Gates”. In: Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES 2015). Vol. 9293. Lecture Notes in Computer Science. Springer, 2015, pp. 102–122. http://dx.doi.org/10.1007/978-3-662-48324-4_6.
- Kimmo Järvinen and Ingrid Verbauwhede. “How to Use Koblitz Curves on Small Devices?” In: Proceedings of the 13th Smart Card Research and Advanced Application Conference (CARDIS 2014), Revised Selected Papers. Vol. 8968. Lecture Notes in Computer Science. Springer, 2014, pp. 154–170. http://dx.doi.org/10.1007/978-3-319-16763-3_10.

LITERATURE (CONT.)

- Kimmo Järvinen. “Optimized FPGA-based Elliptic Curve Cryptography Processor for High-Speed Applications”. Integration, the VLSI Journal 44.4 (2011), pp. 270–279. http://dx.doi.org/10.1016/j.vlsi.2010.08.001.
- Billy Bob Brumley and Kimmo U. Järvinen. “Conversion Algorithms and Implementations for Koblitz Curve Cryptography”. IEEE Transactions on Computers 59.1 (Jan. 2010), pp. 81–92. http://dx.doi.org/10.1109/TC.2009.132.
- Kimmo Järvinen and Jorma Skyttä. “Fast Point Multiplication on Koblitz Curves: Parallelization Method and Implementations”. Microprocessors and Microsystems 33.2 (Mar. 2009), pp. 106–116. http://dx.doi.org/10.1016/j.micpro.2008.08.002.