Lightweight Cryptography for the Cloud: Exploit the Power of Bitslice Implementation

Seiichi Matsuda (Sony Corporation)
Shiho Moriai (NICT)

Outline

- Motivation
  - Lightweight Cryptography in Cyber Physical Systems
- Bitslice Implementations
  - PRESENT
  - Piccolo
- Performance Evaluation
- Conclusion

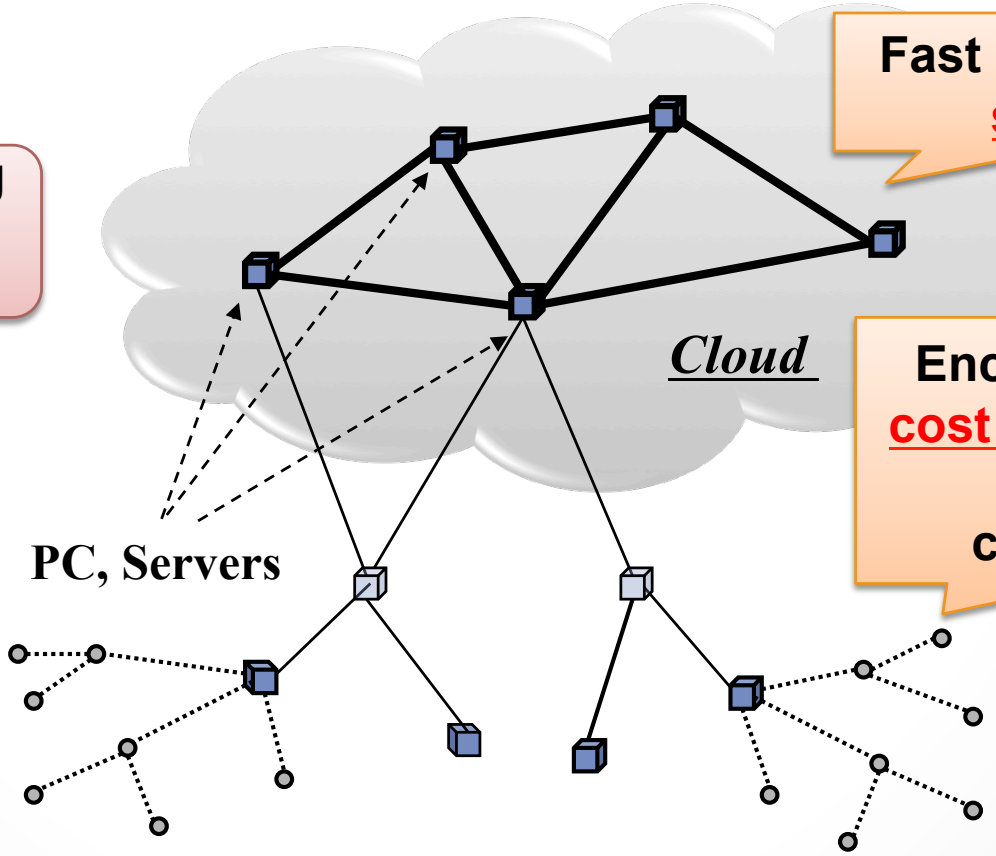
Lightweight cryptography

- Attracting attention in cryptography research area
  - Block ciphers, Stream ciphers, Hash functions, MAC, etc.
- Lightweight block ciphers are standardized in ISO/IEC 29192-2
  - PRESENT(64-bit) and CLEFIA(128-bit)
- Designed for constrained devices
  - Mostly optimized for H/W implementation
    - Gate area, power and energy consumption
  - S/W performance evaluated on low-end platforms
    - Low memory requirements and small code size

Lightweight cryptography in Cyber Physical Systems

Big data processing in a cloud environment

Data collection in sensing networks

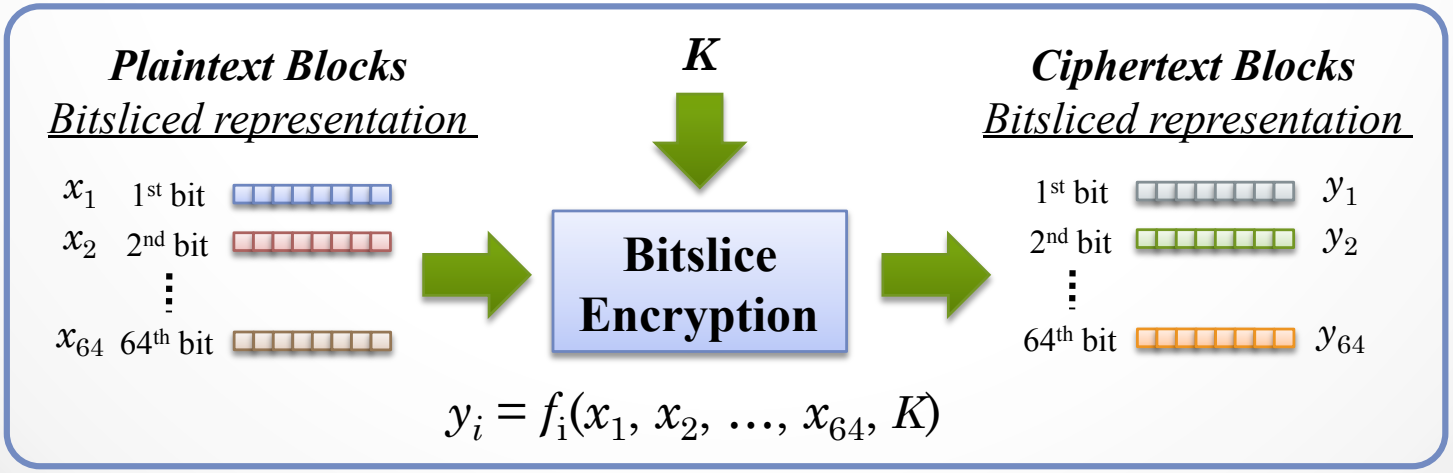


Fast decryption in software

Encryption in low-cost hardware using lightweight cryptography

Motivation

- Most lightweight block ciphers do not show good throughput in software on high-end platforms
  - No use cases identified
  - Low priority in design criteria
- Bitslice implementation (Biham, 1997)
  - Simulates H/W implementation in S/W program
  - Uses logical instructions corresponding to H/W logical gates
    - Expected to improve S/W performance of small ciphers
  - Resistant to cache timing attacks
    - Also prevents cross-VM attacks in multi-tenant cloud



Motivation

- Most lightweight block ciphers do not show good throughput in software on high-end platforms
  - No use cases identified
  - Low priority in design criteria
- Bitslice implementation (Biham, 1997)
  - Simulates H/W implementation in S/W program
  - Uses logical instructions corresponding to H/W logical gates
    - Expected to improve S/W performance of small ciphers
  - Resistant to cache timing attacks
    - Also prevents cross-VM attacks in multi-tenant cloud

Our aim

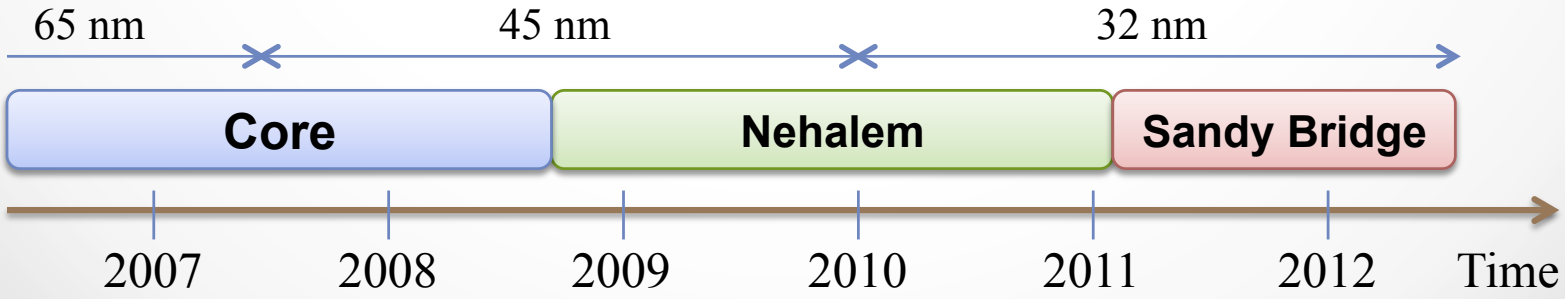
To explore S/W performance of lightweight block ciphers using bitslice implementation

Target ciphers and Approach

- 64-bit lightweight block ciphers
  - PRESENT (Bogdanov et al, CHES2007)
    - Standardized in ISO/IEC 29192-2
    - Substitution-Permutation Network (SPN)
  - Piccolo (Shibutani et al, CHES2011)
    - Generalized Feistel Networks (GFN)
- Our Implementation approach
  - Reduce the degree of parallelism of bitslice implementation for performance and more applications
    - PRESENT 8, 16 and 32 parallel
    - Piccolo 16 parallel

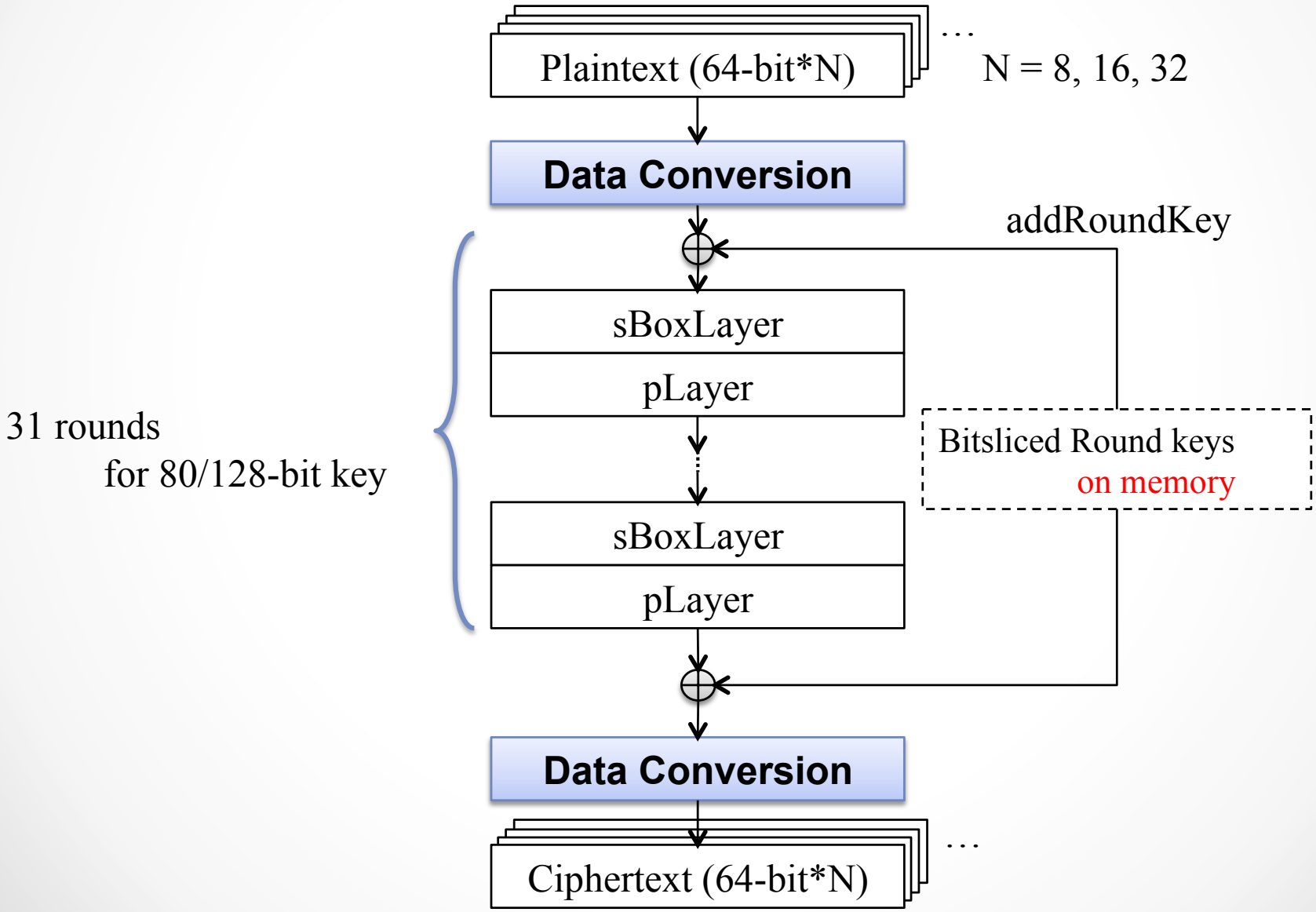
Platforms

- Intel microarchitectures (w/ SIMD instructions)
  - Core (45nm) and Nehalem (SSSE3 and SSE4)
    - 16 128-bit XMM registers
    - Shuffle and unpack instructions
  - Sandy Bridge (AVX)
    - 128-bit XMM registers are extended to 256-bit YMM registers
    - Not fully supports for integer instructions on YMM registers
    - 3-operand syntax
      - SSE: *pxor* xmm1, xmm2 (xmm1 ^= xmm2)
      - AVX: *vpxor* xmm1, xmm2, xmm3 (xmm1 = xmm2 ^ xmm3)
    - In our implementation
      - Using 128-bit AVX working on XMM registers (3-operand syntax)

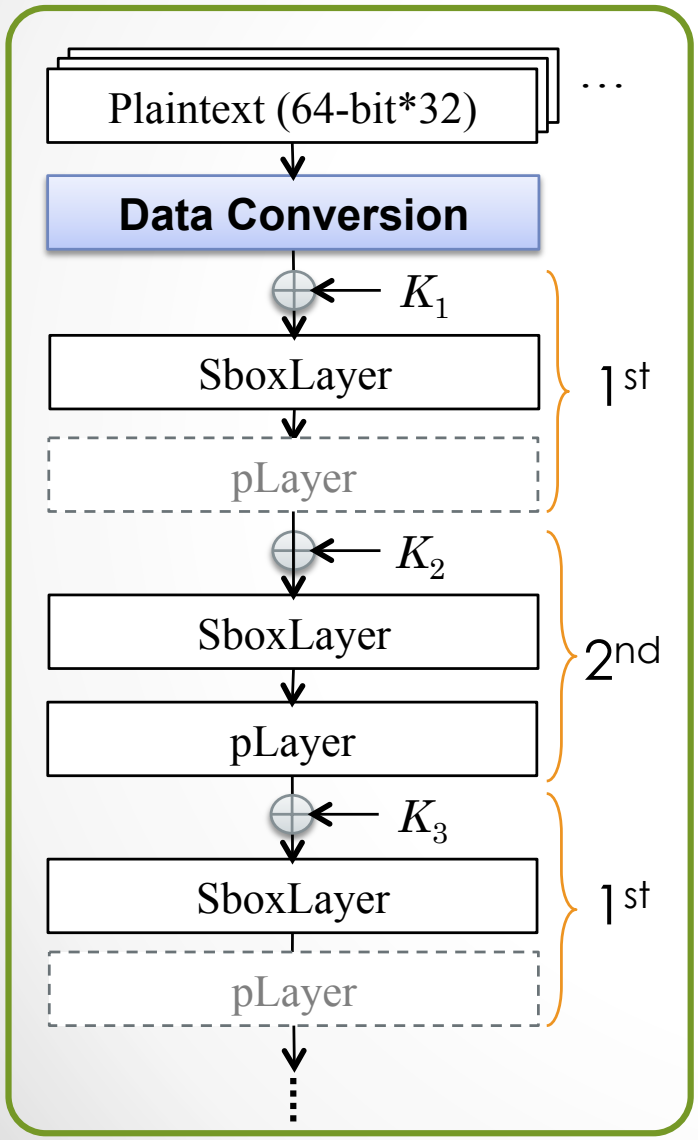


PRESENT

Bitslice implementation of PRESENT

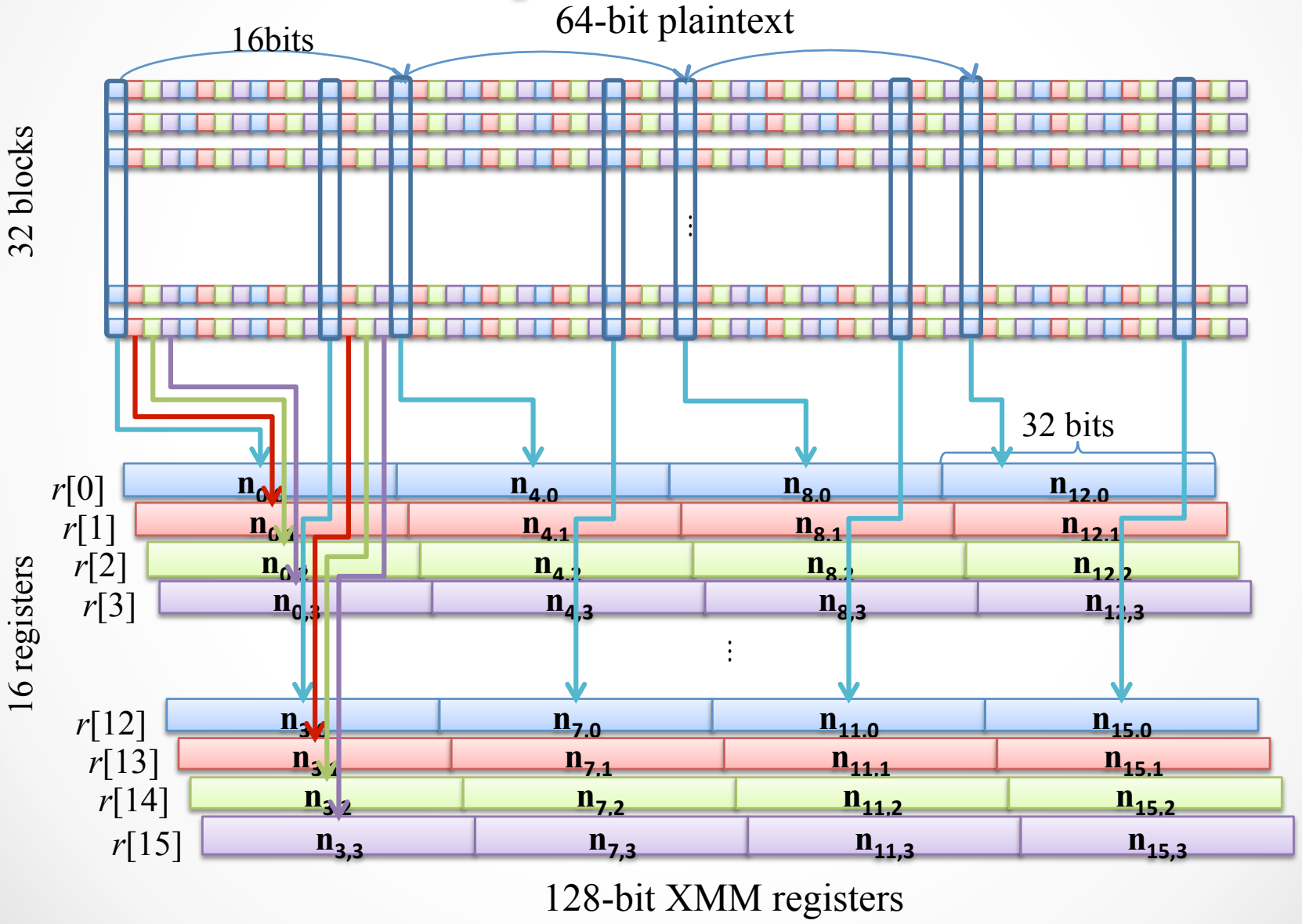


32-block parallel implementation

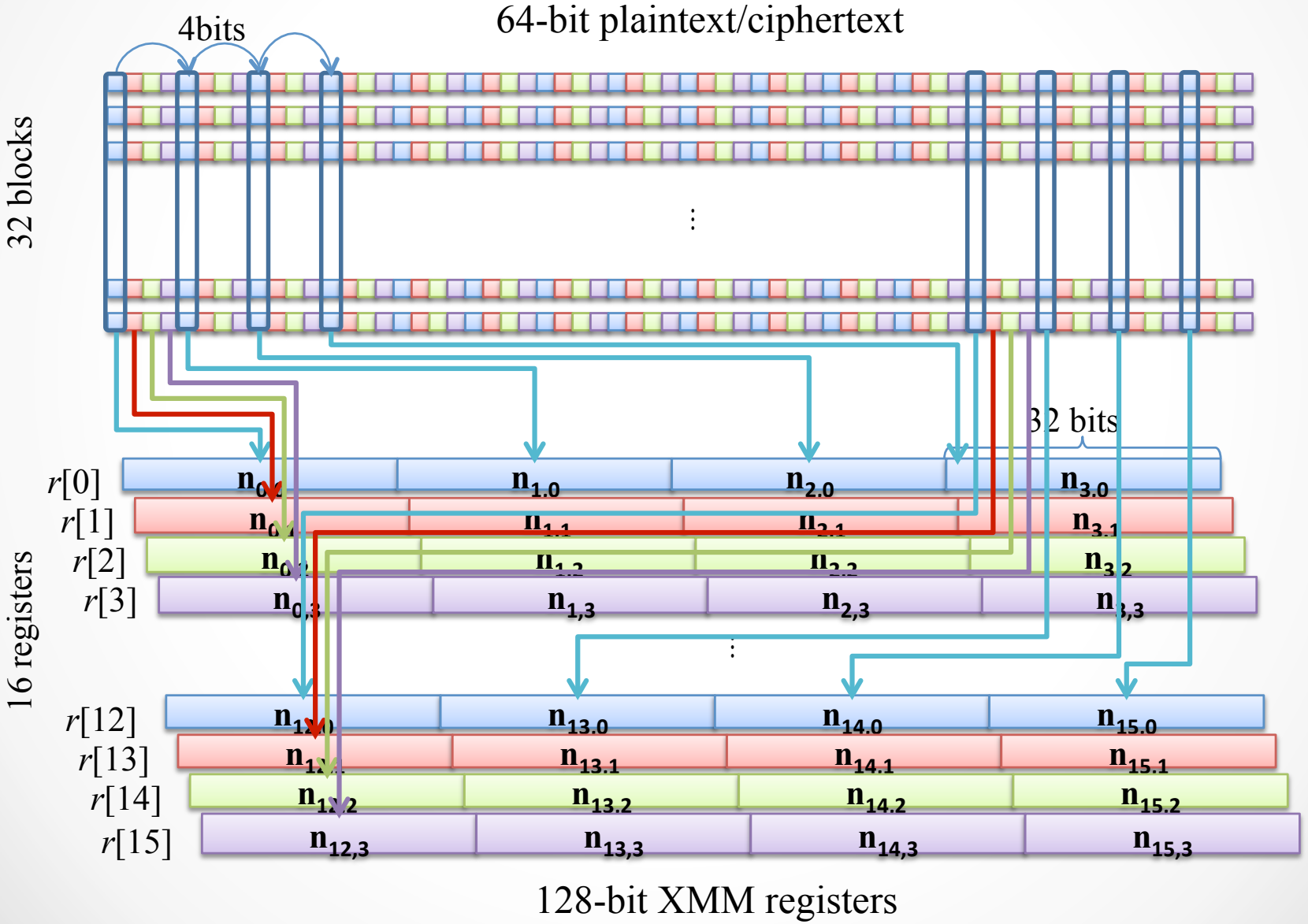


- 2 bitsliced representations (1st/2nd)
  - To skip pLayer every other round
- pLayer for 1st bitslice rep.
  - Conversion from 1st to 2nd rep.
  - Register renaming (**No cost**)
- pLayer for 2nd bitslice rep.
  - Conversion from 2nd to 1st rep.
  - Uses shuffle and unpack instructions
    - 52 instructions with SSE
    - 36 instructions with 128-bit AVX

1st Bitsliced Representation for 32-block

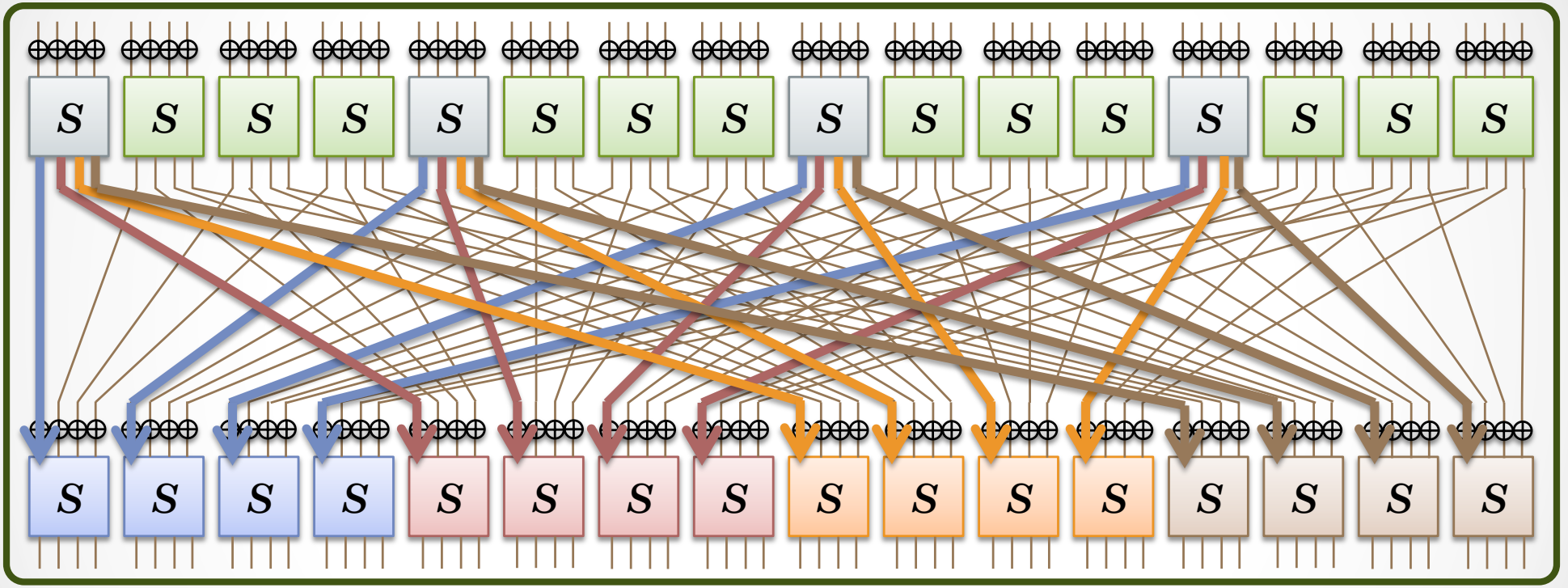


2nd Bitsliced Representation for 32-block



1st Bitslice Representation

$r[0]$	$n_{0,0}$	$n_{4,0}$	$n_{8,0}$	$n_{12,0}$
$r[1]$	$n_{0,1}$	$n_{4,1}$	$n_{8,1}$	$n_{12,1}$
$r[2]$	$n_{0,2}$	$n_{4,2}$	$n_{8,2}$	$n_{12,2}$
$r[3]$	$n_{0,3}$	$n_{4,3}$	$n_{8,3}$	$n_{12,3}$



2nd Bitslice Representation

$r[0]$	$n_{0,0}$	$n_{1,0}$	$n_{2,0}$	$n_{3,0}$
$r[1]$	$n_{4,0}$	$n_{5,0}$	$n_{6,0}$	$n_{7,0}$
$r[2]$	$n_{8,0}$	$n_{9,0}$	$n_{10,0}$	$n_{11,0}$
$r[3]$	$n_{12,0}$	$n_{13,0}$	$n_{14,0}$	$n_{15,0}$

SboxLayer

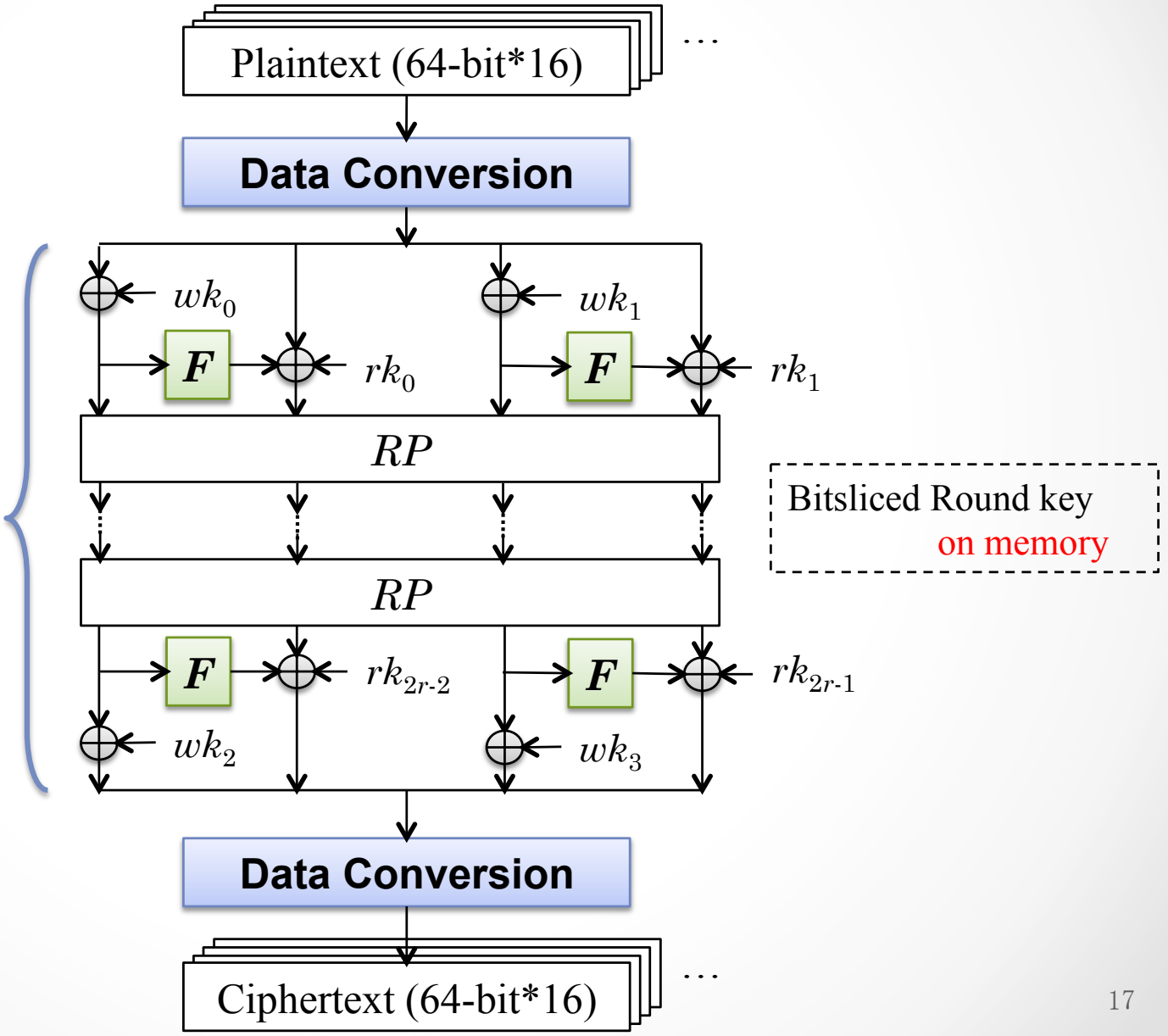
- Using logical representation equivalent to 4-bit S-box
  - 14 instructions (Courtois et al, 2011)
    - Assuming 3-operand syntax with **8** registers
    - Used in 8/16-block parallel implementations with 128-bit AVX
  - 20 instructions (Our result based on Osvik's method)
    - Assuming 2-operand syntax with **5** registers
    - Used in the other implementations
    - Reduced to17 with 128-bit AVX

```
// Input:  r3, r2, r1, r0, tmp
// Output: r3, r2, r1, r0
 1. r2  ^= r1;   r3  ^= r1;
 2. tmp =  r2;   r2  &= r3;
 3. r1  ^= r2;   tmp ^= r0;
 4. r2  =  r1;   r1  &= tmp;
 5. r1  ^= r3;   tmp ^= r0;
 6. tmp |= r2;   r2  ^= r0;
 7. r2  ^= r1;   tmp ^= r3;
 8. r2  = ~r2;   r0  ^= tmp;
 9. r3  =  r2;   r2  &= r1;
10. r2  |= tmp;
11. r2  = ~r2;
```

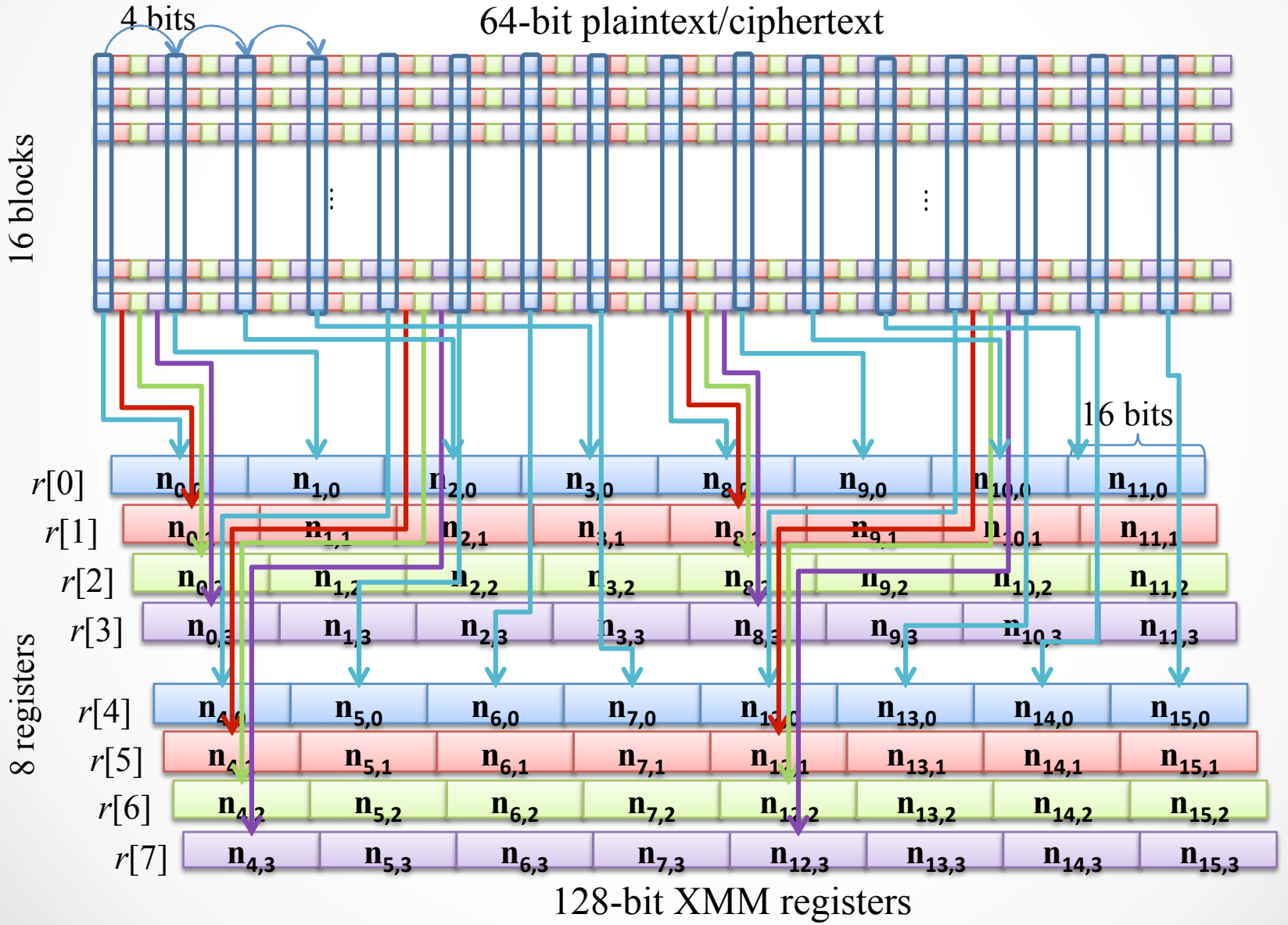
Piccolo

Bitslice implementation of Piccolo

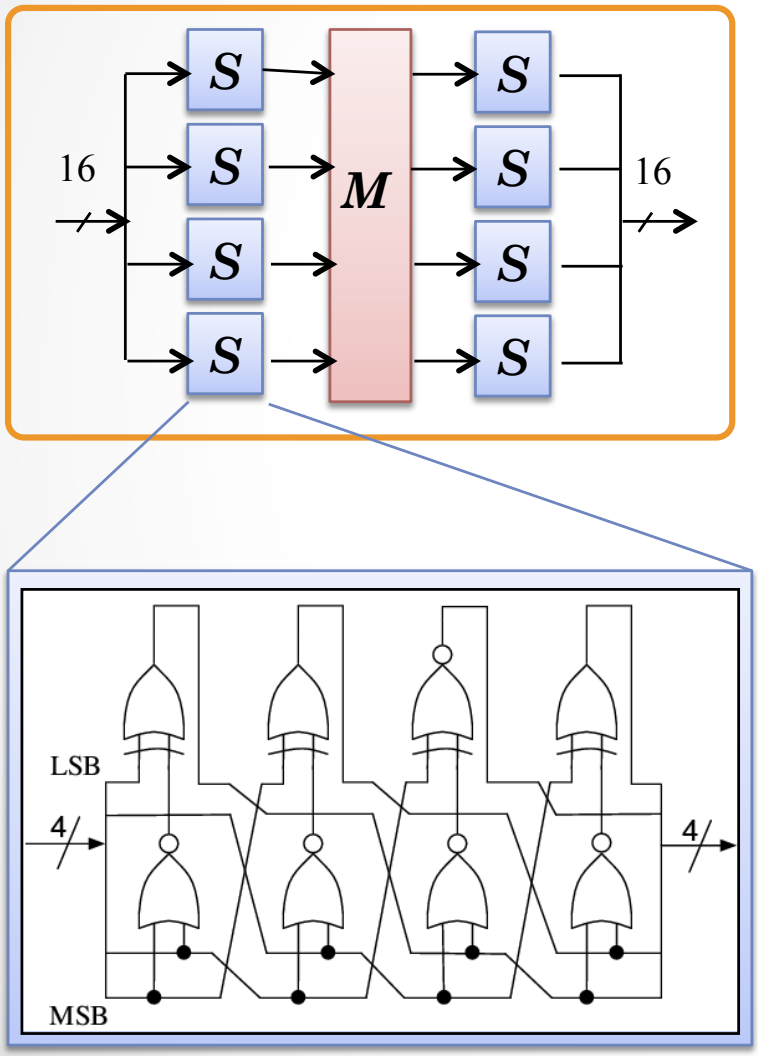
25 (31) rounds for
80-bit (128-bit)
key



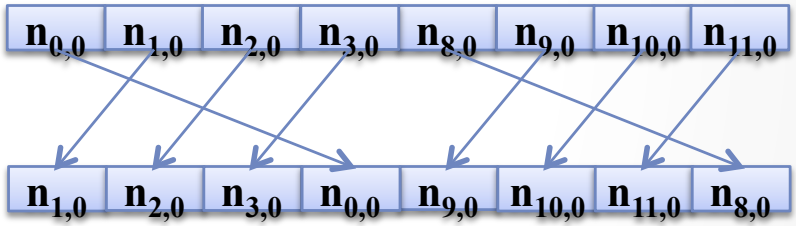
Bitsliced representation for 16-block



F-function

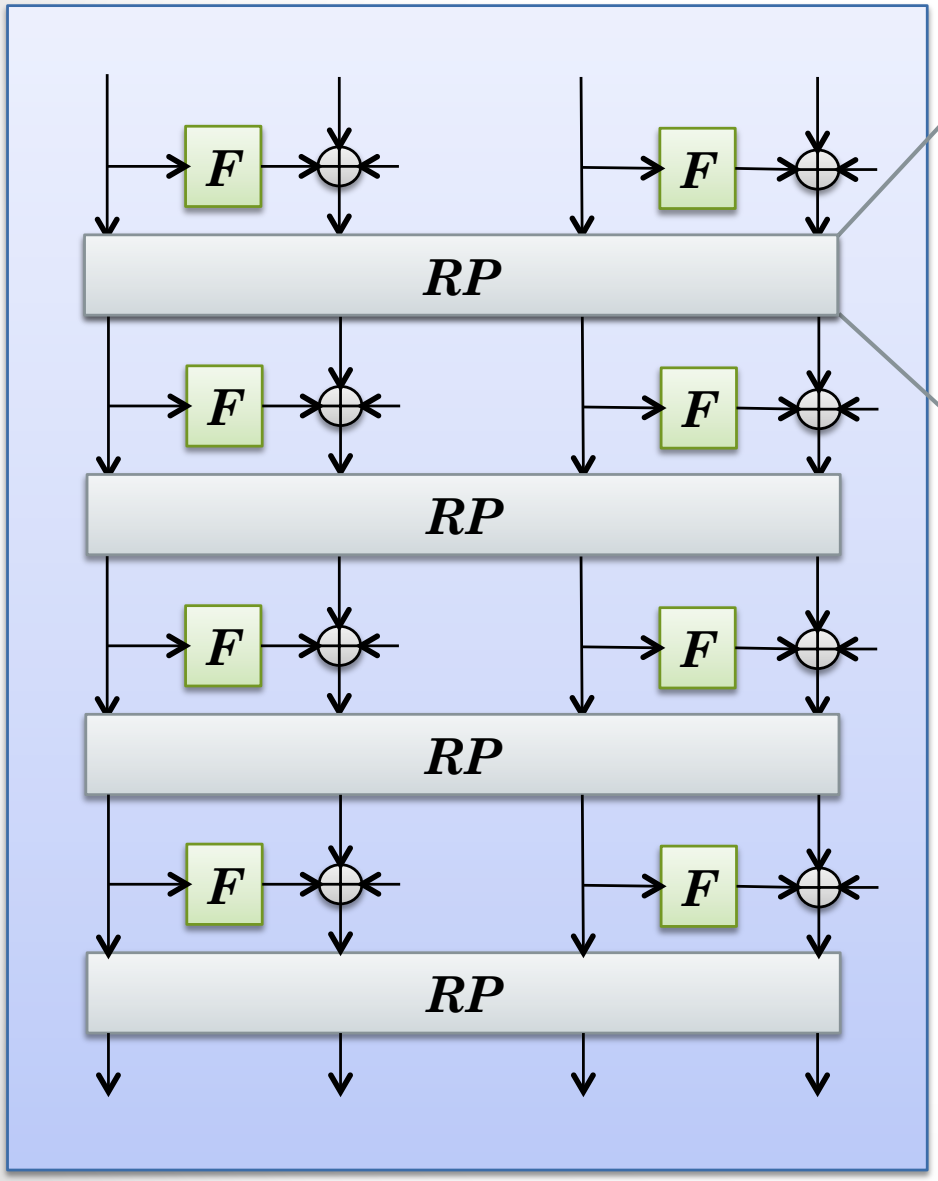


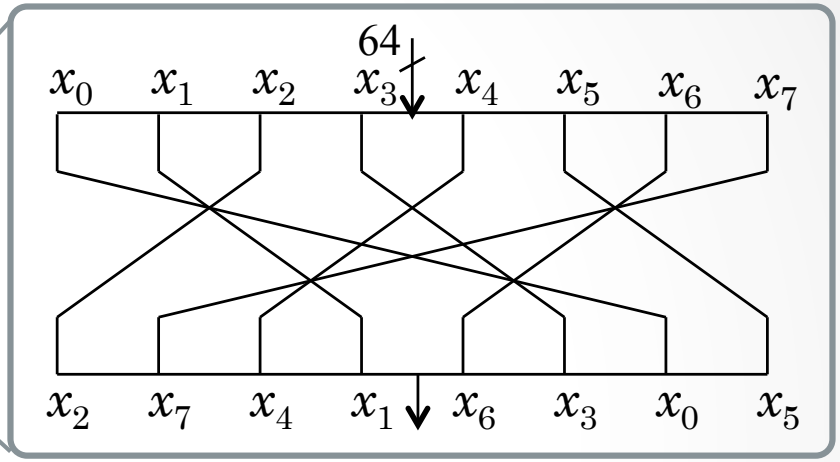
- Diffusion Matrix M
  - 25 instructions (using 8 reg.)
    - Based on Käsper's impl. (CHES2009)
  - 2 matrices can be computed at once
    - Rotation by 16-bit using *pshufb*



- 4-bit S-box S
  - 13 instructions (using 5 reg.)
    - Same approach as PRESENT S-box
    - Reduced to 11 with 128-bit AVX

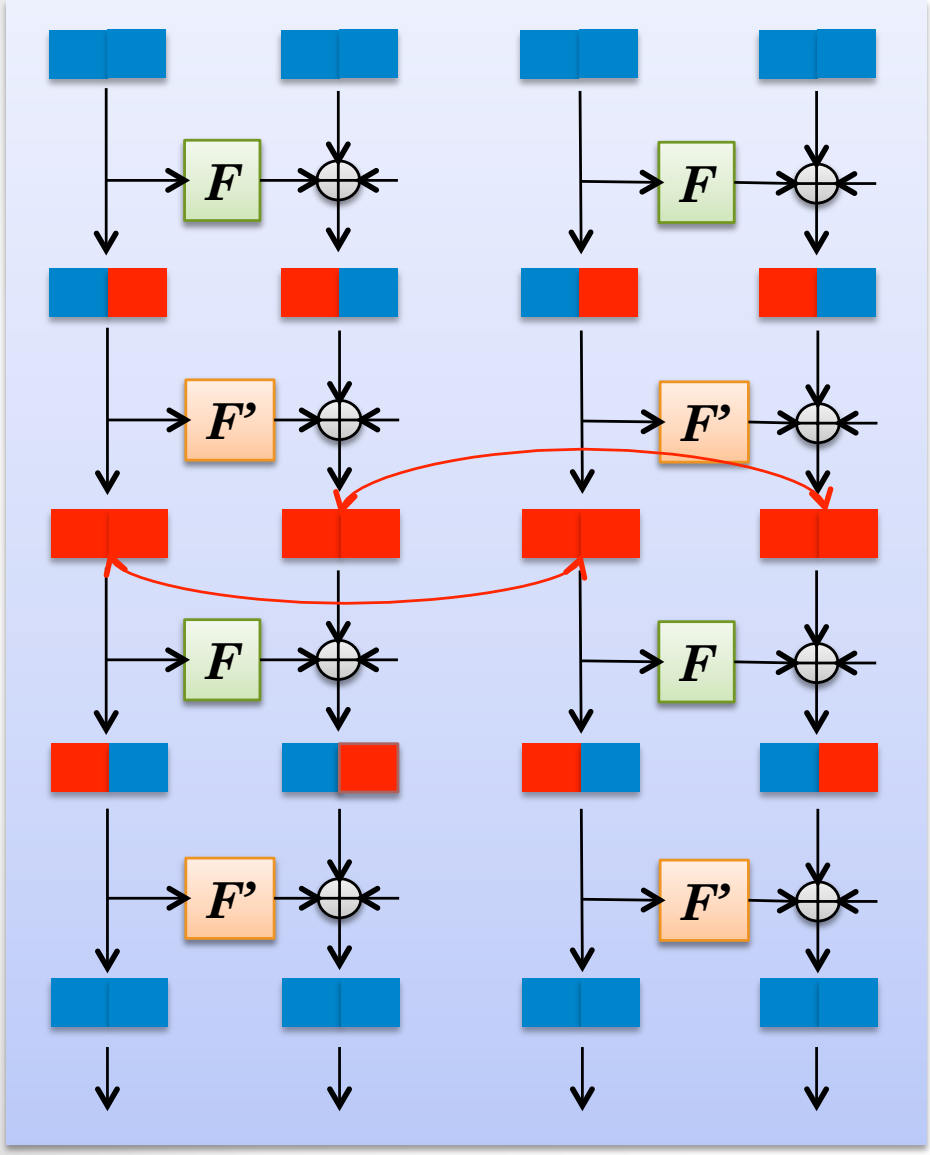
Round Permutation RP





- Standard implementation
  - 8 instructions applying *pshufb*

Round Permutation RP



- Remove RP
  - Saves 200 instructions for 80-bit key
  - Need to modify F-functions
    - F-functions F' on 2nd, 4th round
      - Same S-box calculation
      - New matrix rep.
        - No penalty with SSE
        - 4 more inst. with AVX
    - No effect on 3rd round
  - Need to align round keys

Performance Evaluation

Instruction Counts

PRESENT-80/128 32-block parallel	SSE	128-bit AVX
addRoundKey	512	512
sBoxLayer	2604	2232
pLayer	780	540
Conversion	550	468
Total	*4446*	*3752*

Piccolo -80 16-block parallel	SSE	128-bit AVX
Diffusion matrix	625	573
S-box	650	550
addRK/WK	308	208
Conversion	232	200
Total	*1815*	*1531*

Evaluation Results

Algorithm	PRESENT-80/128			Piccolo-80	Piccolo-128
Number of parallel blocks	8	16	32	16	
Xeon E3-1280 (Sandy Bridge)					
Cycles/byte	8.46	6.52	4.73	4.57	5.52
Instructions/cycle	2.04	2.48	3.10	2.61	2.61
Core i7 870 (Nehalem)					
Cycles/byte	10.88	7.26	5.79	5.69	6.80
Instructions/cycle	2.07	2.93	3.00	2.49	2.52
Xeon E5410 (Core)					
Cycles/byte	13.55	10.98	7.55	6.85	8.23
Instructions/cycle	1.67	1.93	2.30	2.07	2.08

cf. AES-CTR (8-block parallel) 6.92 cycles/byte @ Core i7 920 (Nehalem)
[Käsper et al. CHES2009]

Discussion

- Can lightweight block ciphers with smaller gate size achieve better S/W performance by bitslicing ?
  - Relationship is not trivial and depends on several factors
    - Algorithm of target cipher
    - Implementation approach
    - Target platform
  - Results of other lightweight block ciphers (Non-bitslice implementation)
    - LED-64 (Guo et al., CHES 2011)
      - 57 cycles/byte on Core i7 Q720
    - TWINE (Suzaki et al., SAC 2012)
      - 4.77 cycles/byte on Core i5 U560

Conclusion

- We showed great potential of lightweight cryptography in fast S/W implementation on high-end platforms by exploiting bitslice implementation
  - On Sandy Bridge:
    - PRESENT-80/128 4.73 cycles/byte
    - Piccolo-80 4.57 cycles/byte
  - On Nehalem: Faster results than the S/W record of AES
- Lightweight cryptography is not limited to constrained devices. Open the way to cloud computing!