Computational Survivalism
Compiler(s) for the End of Moore's Law: a case study

Pierre-Évariste Dagand

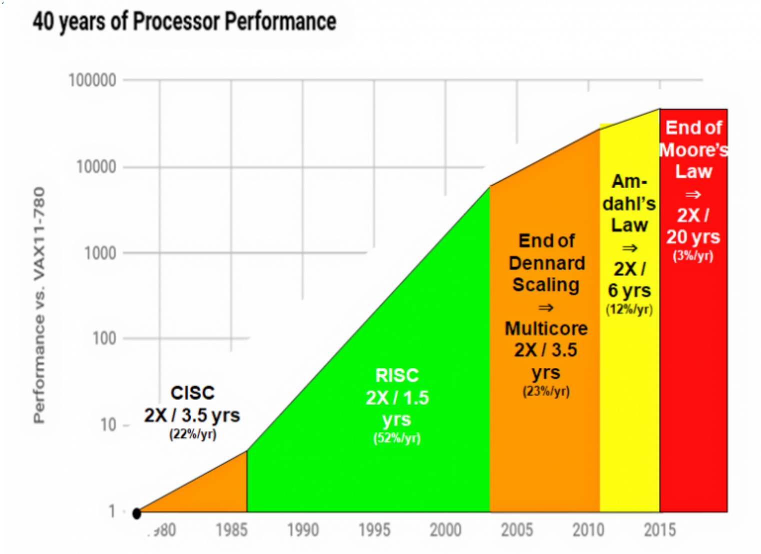
Joint work with Darius Mercadier

Based on an original idea from Xavier Leroy

LIP6 – CNRS – Inria
Sorbonne Université

The End is Coming

(Maybe)



Turing Award Lecture, David Patterson & John Hennessy (2018)

An Escape Hatch

The Way of the Computer Architect:

- Towards domain-specific architectures
- Solving narrow problems
- Delineated by specialized languages
- Gustafson's law: aim for throughput!

What keeps *us* up all night?

- How to organize this diversity?
- Can we retain a "programming continuum"?
- Will PLDI have to go through the next 700 DSLs?

The Usuba Experiment

Setup:

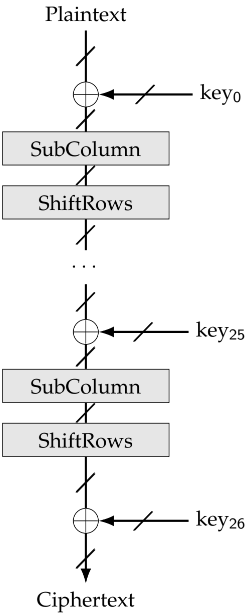
- Domain-specific architecture: SIMD
- Narrow problem: symmetric ciphers
- Specialized language: software circuits

Parameters:

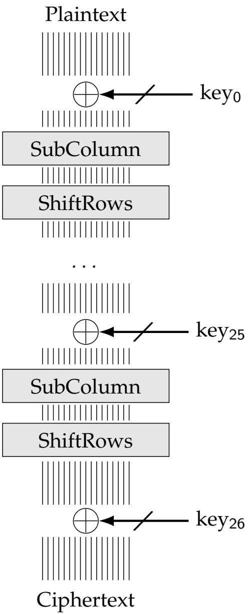
- No runtime, no concurrency
- No memory access *(feature!)*
- Evaluation: optimized reference implementations

The death of optimizing compilers, Daniel J. Bernstein (2015)

Anatomy of a block cipher

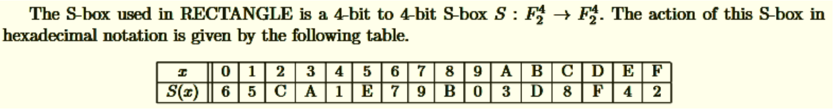


Anatomy of a block cipher



Anatomy of a block cipher

Rectangle/SubColumn



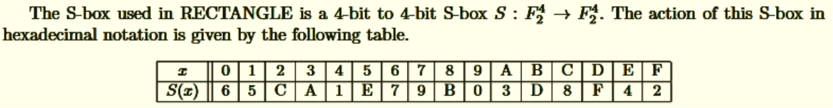
The S-box used in RECTANGLE is a 4-bit to 4-bit S-box $S : F_2^4 \to F_2^4$. The action of this S-box in hexadecimal notation is given by the following table.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	6	5	C	A	1	E	7	9	B	0	3	D	8	F	4	2

Caution: lookup tables are **strictly forbidden**!

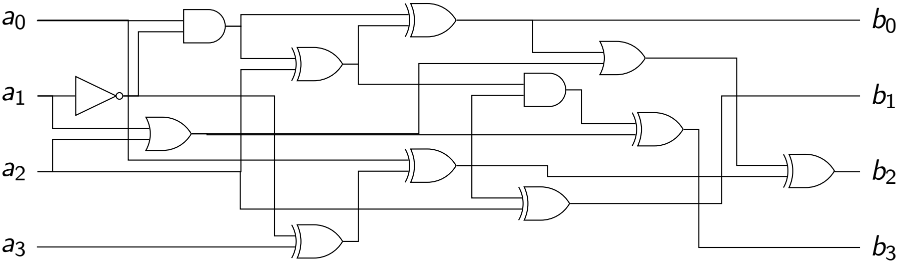
Anatomy of a block cipher

Rectangle/SubColumn



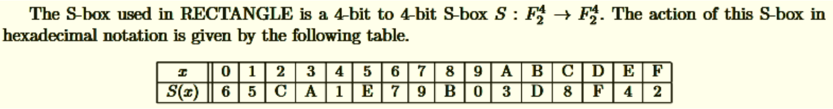
The S-box used in RECTANGLE is a 4-bit to 4-bit S-box $S : F_2^4 \to F_2^4$. The action of this S-box in hexadecimal notation is given by the following table.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	6	5	C	A	1	E	7	9	B	0	3	D	8	F	4	2



Anatomy of a block cipher

Rectangle/SubColumn



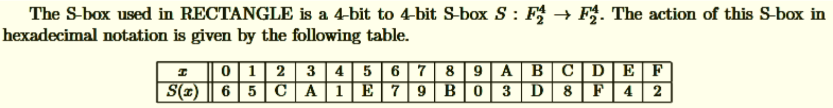
The S-box used in RECTANGLE is a 4-bit to 4-bit S-box $S : F_2^4 \to F_2^4$. The action of this S-box in hexadecimal notation is given by the following table.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	6	5	C	A	1	E	7	9	B	0	3	D	8	F	4	2

```
void SubColumn(__m128i *a0, __m128i *a1,
               __m128i *a2, __m128i *a3) {
    __m128i t1, t2, t3, t5, t6, t8, t9, t11;
    __m128i a0_ = *a0; __m128i a1_ = *a1;
    t1 = ~*a1;      t2 = *a0 & t1;  t3 = *a2 ^ *a3;
    *a0 = t2 ^ t3;  t5 = *a3 | t1;  t6 = a0_ ^ t5;
    *a1 = *a2 ^ t6; t8 = a1_ ^ *a2; t9 = t3 & t6;
    *a3 = t8 ^ t9;  t11 = *a0 | t8; *a2 = t6 ^ t11;
}
```

Anatomy of a block cipher

Rectangle/SubColumn



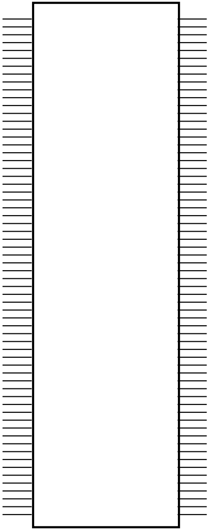
The S-box used in RECTANGLE is a 4-bit to 4-bit S-box $S : F_2^4 \rightarrow F_2^4$. The action of this S-box in hexadecimal notation is given by the following table.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	6	5	C	A	1	E	7	9	B	0	3	D	8	F	4	2

```
table SubColumn (a:v4) returns (b:v4) {
    6, 5, 12, 10, 1, 14, 7, 9, 11, 0,  3, 13, 8, 15, 4, 2
}
```

Anatomy of a block cipher

Rectangle/ShiftRows

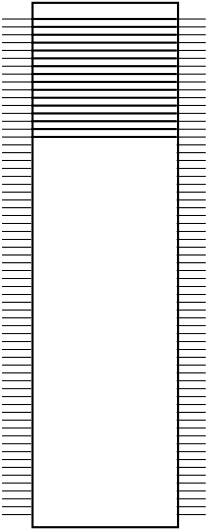


ShiftRows

```
node ShiftRows (input:u16x4)
   returns     (out:u16x4)
```

Anatomy of a block cipher

Rectangle/ShiftRows

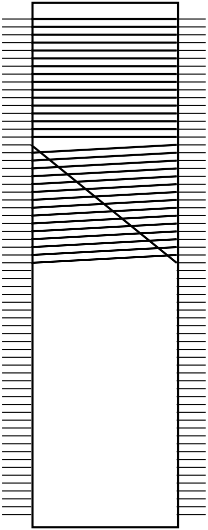


ShiftRows

```
node ShiftRows (input:u16x4)
   returns     (out:u16x4)
let
    out[0] = input[0];



tel
```

Anatomy of a block cipher

Rectangle/ShiftRows

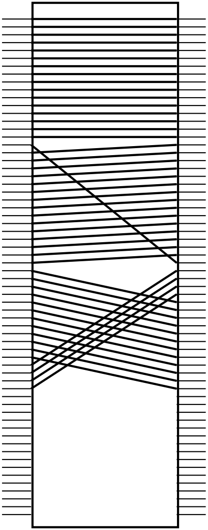


ShiftRows

```
node ShiftRows (input:u16x4)
   returns     (out:u16x4)
let
    out[0] = input[0];
    out[1] = input[1] <<< 1;


tel
```

Anatomy of a block cipher

Rectangle/ShiftRows

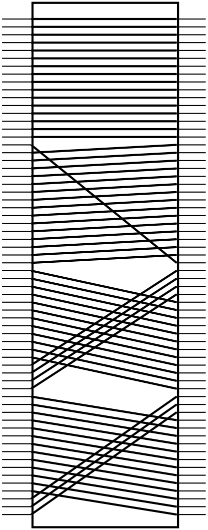


ShiftRows

```
node ShiftRows (input:u16x4)
   returns     (out:u16x4)
let
    out[0] = input[0];
    out[1] = input[1] <<< 1;
    out[2] = input[2] <<< 12;

tel
```

Anatomy of a block cipher

Rectangle/ShiftRows

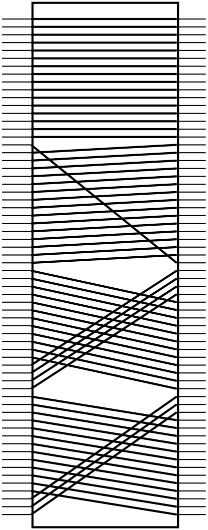


ShiftRows

```
node ShiftRows (input:u16x4)
   returns     (out:u16x4)
let
    out[0] = input[0];
    out[1] = input[1] <<< 1;
    out[2] = input[2] <<< 12;
    out[3] = input[3] <<< 13;
tel
```

Anatomy of a block cipher

Rectangle/ShiftRows



ShiftRows

```
void ShiftRows(__m128i a[64]) {
  int rot[] = { 0, 1, 12, 13 };
  for (int k = 1; k < 4; k++) {
    __m128i tmp[16];
    for (int i = 0; i < 16; i++)
      tmp[i] = a[k*16+(16+rot[k]+i)%16];
    for (int i = 0; i < 16; i++)
      a[k*16+i] = tmp[i];
  }
}
```

Anatomy of a block cipher

Rectangle, naïvely

```
void Rectangle(__m128i plain[64], __m128i key[26][64],
               __m128i cipher[64]) {

  for (int i = 0; i < 25; i++) {
    for (int j = 0; j < 64; j++)
      plain[j] ^= key[i][j];
    for (int j = 0; j < 16; j++)
      SubColumn(&plain[j], &plain[j+16],
                &plain[j+32], &plain[j+48]);
    ShiftRows(plain);
  }
  for (int i = 0; i < 64; i++)
    cipher[i] = plain[i] ^ key[25][i];
}
```

Anatomy of a block cipher

Rectangle, our way

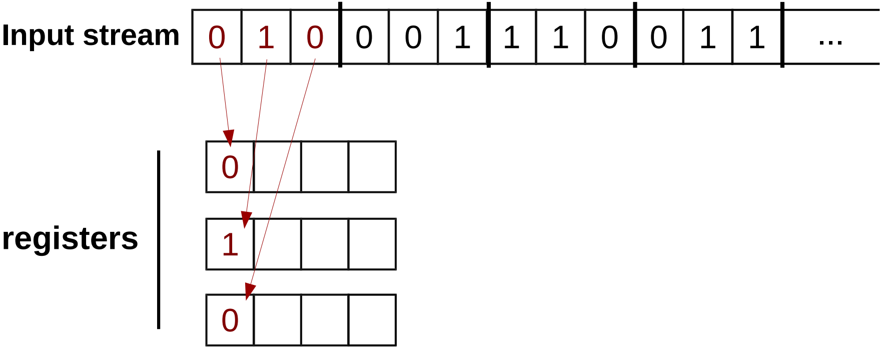
```
node ShiftRows (input:u16x4)
       returns (out:u16x4)
vars
let
    out[0] = input[0];
    out[1] = input[1] <<< 1;
    out[2] = input[2] <<< 12;
    out[3] = input[3] <<< 13;
tel
```

```
table SubColumn (input:v4)
       returns  (out:v4) {
    6, 5, 12, 10, 1, 14, 7, 9,
    11, 0, 3, 13, 8, 15, 4, 2
}
```

```
node Rectangle (plain:u16x4,
                key  :u16x4[26])
       returns (cipher:u16x4)
vars
    round : u16x4[26]
let
    round[0] = plain;
    forall i in [0,24] {
      round[i+1] =
         ShiftRows(
           SubColumn(
             round[i] ^ key[i]
           )
         )
    }
    cipher = round[25] ^ key[25]
tel
```

Bitslicing

High-throughput software circuits



Input stream

0 1 0 | 0 0 1 | 1 1 0 | 0 1 1 | ...

registers

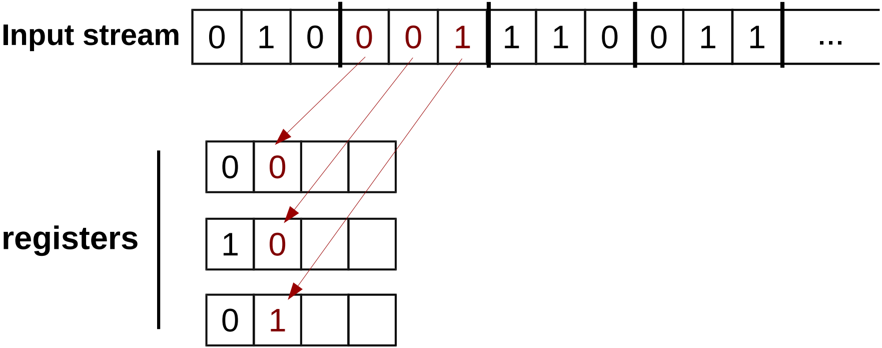
0

1

0

Bitslicing

High-throughput software circuits



Input stream

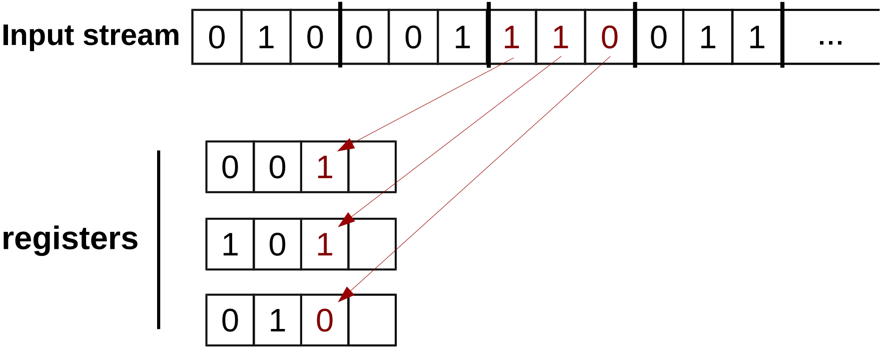
0	1	0	0	0	1	1	1	0	0	1	1	...

registers

0	0		
1	0		
0	1		

Bitslicing

High-throughput software circuits



Input stream | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | ...

registers

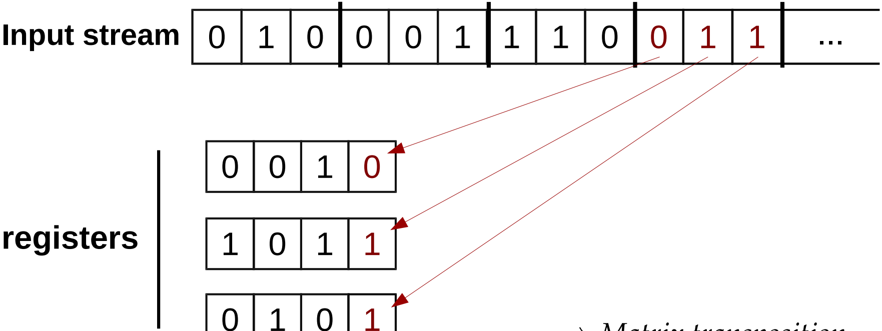
0 | 0 | 1 |

1 | 0 | 1 |

0 | 1 | 0 |

Bitslicing

High-throughput software circuits



Input stream | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | …

registers

0 | 0 | 1 | 0

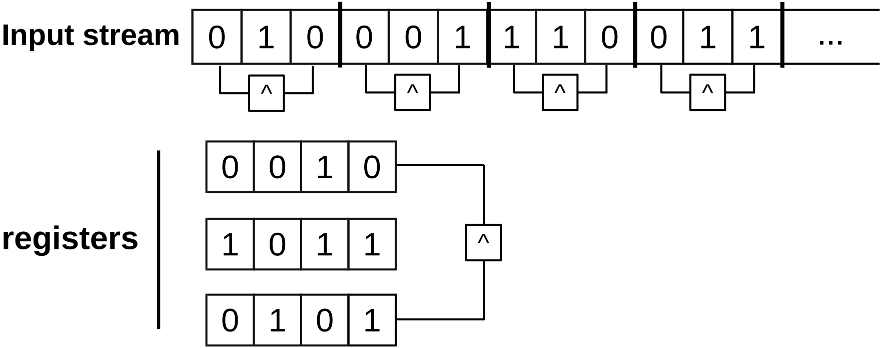
1 | 0 | 1 | 1

0 | 1 | 0 | 1

⇒ *Matrix transposition*

Bitslicing

High-throughput software circuits



Input stream 0 1 0 | 0 0 1 | 1 1 0 | 0 1 1 | ...

^ ^ ^ ^

registers

0 0 1 0

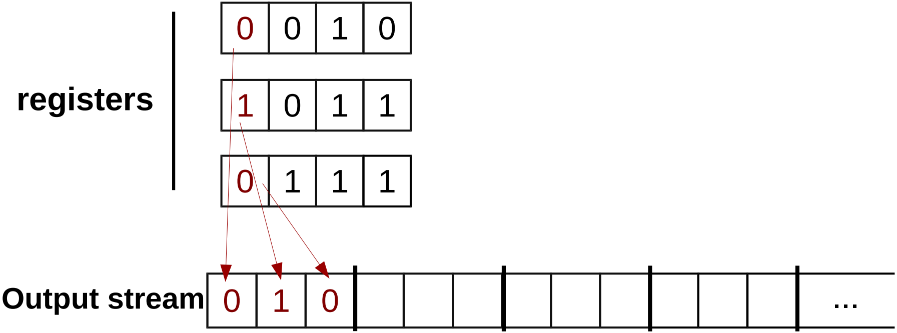
1 0 1 1

0 1 0 1

^

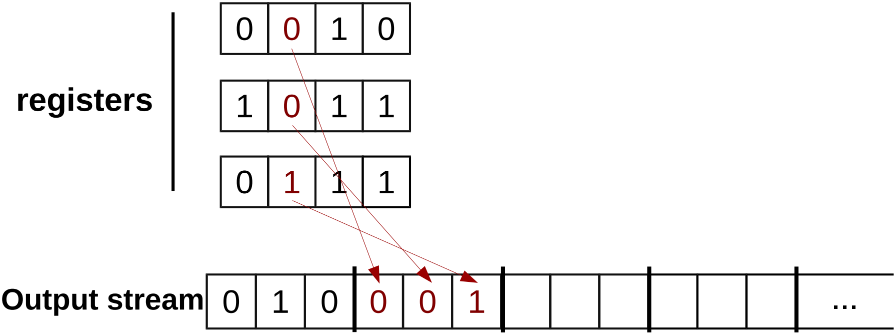
Bitslicing

High-throughput software circuits



Bitslicing

High-throughput software circuits



registers

0 0 1 0

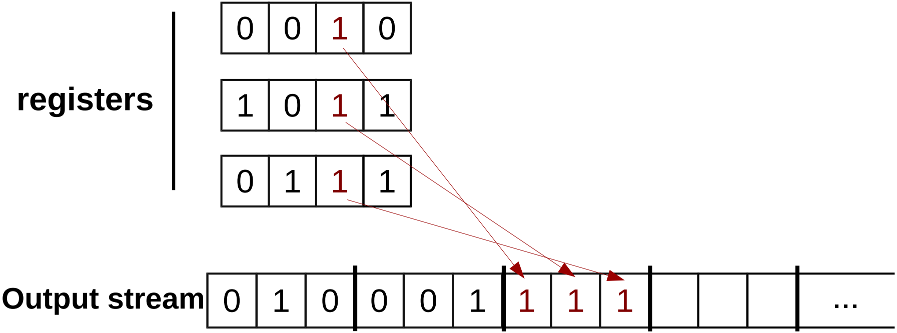
1 0 1 1

0 1 1 1

Output stream 0 1 0 0 0 1 …

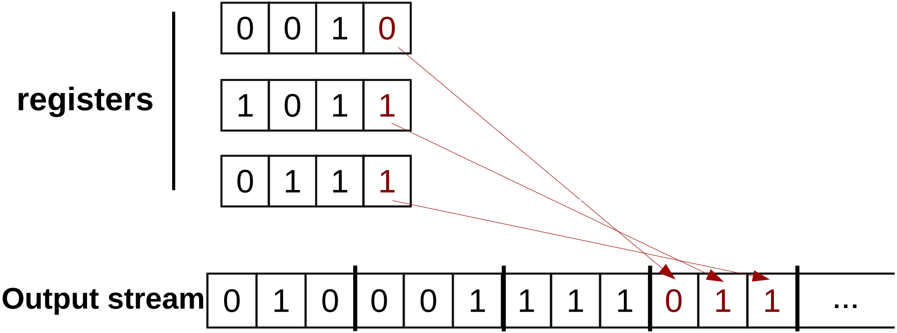
Bitslicing

High-throughput software circuits

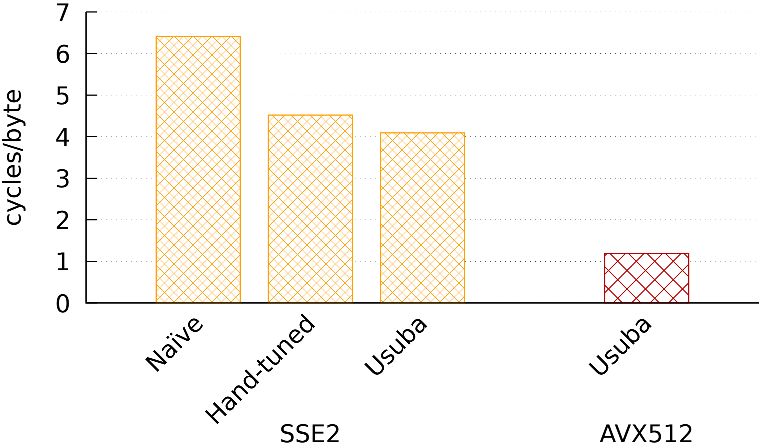


Bitslicing

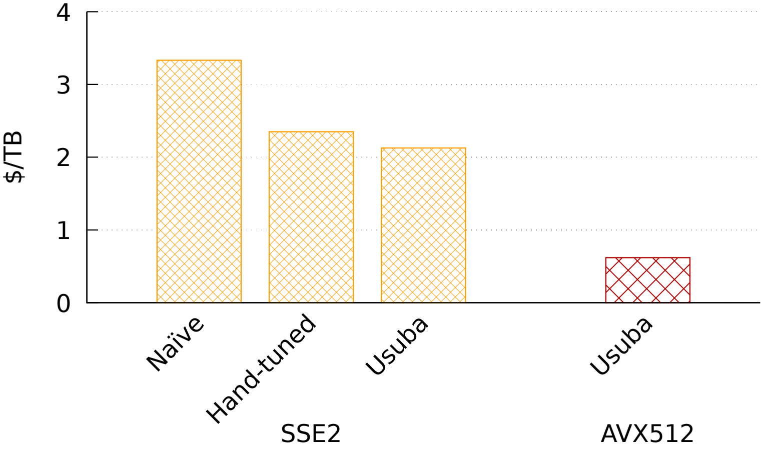
High-throughput software circuits



Man vs. Machine

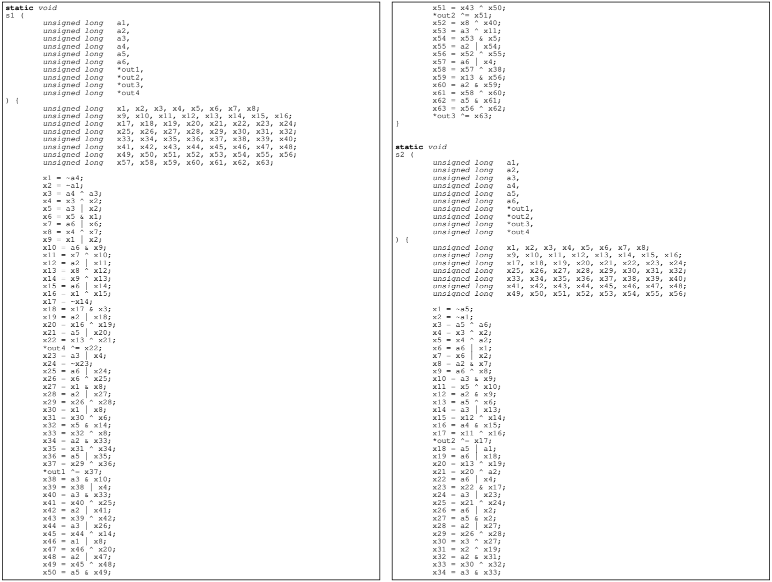


Man vs. Machine



Anatomy of a block cipher

The Real Thing



```
static void
s1 (
        unsigned long   a1,
        unsigned long   a2,
        unsigned long   a3,
        unsigned long   a4,
        unsigned long   a5,
        unsigned long   a6,
        unsigned long   *out1,
        unsigned long   *out2,
        unsigned long   *out3,
        unsigned long   *out4
) {
        unsigned long   x1, x2, x3, x4, x5, x6, x7, x8;
        unsigned long   x9, x10, x11, x12, x13, x14, x15, x16;
        unsigned long   x17, x18, x19, x20, x21, x22, x23, x24;
        unsigned long   x25, x26, x27, x28, x29, x30, x31, x32;
        unsigned long   x33, x34, x35, x36, x37, x38, x39, x40;
        unsigned long   x41, x42, x43, x44, x45, x46, x47, x48;
        unsigned long   x49, x50, x51, x52, x53, x54, x55, x56;
        unsigned long   x57, x58, x59, x60, x61, x62, x63;

        x1 = ~a4;
        x2 = ~a1;
        x3 = a4 ^ a3;
        x4 = x3 ^ x2;
        x5 = a3 | x2;
        x6 = x5 & x1;
        x7 = a6 | x6;
        x8 = x4 ^ x7;
        x9 = x1 | x2;
        x10 = a6 & x9;
        x11 = x7 ^ x10;
        x12 = a2 | x11;
        x13 = x8 ^ x12;
        x14 = x9 ^ x13;
        x15 = a6 | x14;
        x16 = x1 ^ x15;
        x17 = ~x14;
        x18 = x17 & x3;
        x19 = a2 | x18;
        x20 = x16 ^ x19;
        x21 = a5 | x20;
        x22 = x13 ^ x21;
        *out4 ^= x22;
        x23 = a3 | x4;
        x24 = ~x23;
        x25 = a6 | x24;
        x26 = x6 ^ x25;
        x27 = x1 & x8;
        x28 = a2 | x27;
        x29 = x26 ^ x28;
        x30 = x1 | x8;
        x31 = x30 ^ x6;
        x32 = x5 & x14;
        x33 = x32 ^ x8;
        x34 = a2 & x33;
        x35 = x31 ^ x34;
        x36 = a5 | x35;
        x37 = x29 ^ x36;
        *out1 ^= x37;
        x38 = a3 & x10;
        x39 = x38 | x4;
        x40 = a3 & x33;
        x41 = x40 ^ x25;
        x42 = a2 | x41;
        x43 = x39 ^ x42;
        x44 = a3 | x26;
        x45 = x44 ^ x14;
        x46 = a1 | x8;
        x47 = x46 ^ x20;
        x48 = a2 | x47;
        x49 = x45 ^ x48;
        x50 = a5 & x49;
```

```
        x51 = x43 ^ x50;
        *out2 ^= x51;
        x52 = x8 ^ x40;
        x53 = a3 ^ x11;
        x54 = x53 & x5;
        x55 = a2 | x54;
        x56 = x52 ^ x55;
        x57 = a6 | x4;
        x58 = x57 ^ x38;
        x59 = x13 & x56;
        x60 = a2 & x59;
        x61 = x58 ^ x60;
        x62 = a5 & x61;
        x63 = x56 ^ x62;
        *out3 ^= x63;
}


static void
s2 (
        unsigned long   a1,
        unsigned long   a2,
        unsigned long   a3,
        unsigned long   a4,
        unsigned long   a5,
        unsigned long   a6,
        unsigned long   *out1,
        unsigned long   *out2,
        unsigned long   *out3,
        unsigned long   *out4
) {
        unsigned long   x1, x2, x3, x4, x5, x6, x7, x8;
        unsigned long   x9, x10, x11, x12, x13, x14, x15, x16;
        unsigned long   x17, x18, x19, x20, x21, x22, x23, x24;
        unsigned long   x25, x26, x27, x28, x29, x30, x31, x32;
        unsigned long   x33, x34, x35, x36, x37, x38, x39, x40;
        unsigned long   x41, x42, x43, x44, x45, x46, x47, x48;
        unsigned long   x49, x50, x51, x52, x53, x54, x55, x56;

        x1 = ~a5;
        x2 = ~a1;
        x3 = a5 ^ a6;
        x4 = x3 ^ x2;
        x5 = x4 ^ a2;
        x6 = a6 | x1;
        x7 = x6 | x2;
        x8 = a2 & x7;
        x9 = a6 ^ x8;
        x10 = a3 & x9;
        x11 = x5 ^ x10;
        x12 = a2 & x9;
        x13 = a5 ^ x6;
        x14 = a3 | x13;
        x15 = x12 ^ x14;
        x16 = a4 & x15;
        x17 = x11 ^ x16;
        *out2 ^= x17;
        x18 = a5 | a1;
        x19 = a6 | x18;
        x20 = x13 ^ x19;
        x21 = x20 ^ a2;
        x22 = a6 | x4;
        x23 = x22 & x17;
        x24 = a3 | x23;
        x25 = x21 ^ x24;
        x26 = a6 | x2;
        x27 = a5 & x2;
        x28 = a2 | x27;
        x29 = x26 ^ x28;
        x30 = x3 ^ x27;
        x31 = x2 ^ x19;
        x32 = a2 & x31;
        x33 = x30 ^ x32;
        x34 = a3 & x33;
```

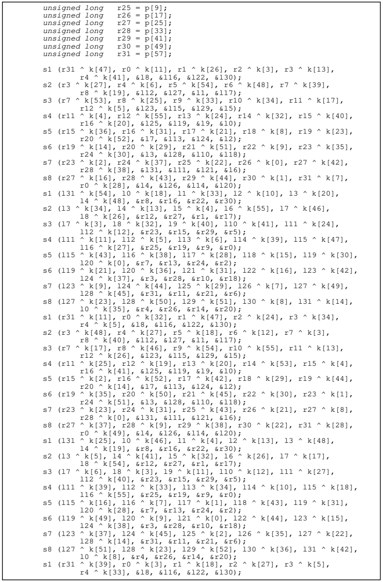
Anatomy of a block cipher

The Real Thing

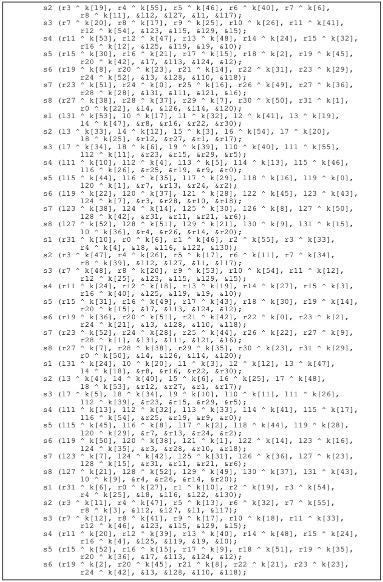
(follows 10 pages of the same…)

Anatomy of a block cipher

The Real Thing



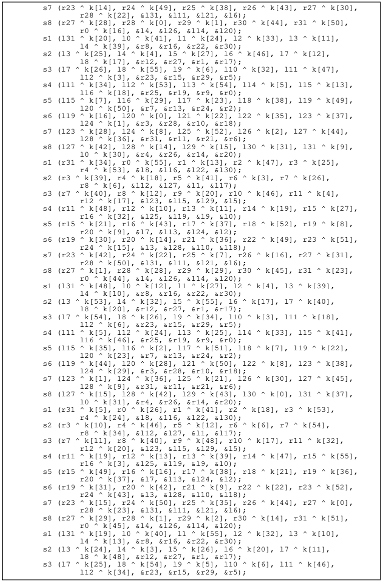
```
unsigned long   r25 = p[9];
unsigned long   r26 = p[17];
unsigned long   r27 = p[25];
unsigned long   r28 = p[33];
unsigned long   r29 = p[41];
unsigned long   r30 = p[49];
unsigned long   r31 = p[57];

s1 (r31 ^ k[47], r0 ^ k[11], r1 ^ k[26], r2 ^ k[3], r3 ^ k[13],
        r4 ^ k[41], &l8, &l16, &l22, &l30);
s2 (r3 ^ k[27], r4 ^ k[6], r5 ^ k[54], r6 ^ k[48], r7 ^ k[39],
        r8 ^ k[19], &l12, &l27, &l1, &l17);
s3 (r7 ^ k[53], r8 ^ k[25], r9 ^ k[33], r10 ^ k[34], r11 ^ k[17],
        r12 ^ k[5], &l23, &l15, &l29, &l5);
s4 (r11 ^ k[4], r12 ^ k[55], r13 ^ k[24], r14 ^ k[32], r15 ^ k[40],
        r16 ^ k[20], &l25, &l19, &l9, &l0);
s5 (r15 ^ k[36], r16 ^ k[31], r17 ^ k[21], r18 ^ k[8], r19 ^ k[23],
        r20 ^ k[52], &l7, &l13, &l24, &l2);
s6 (r19 ^ k[14], r20 ^ k[29], r21 ^ k[51], r22 ^ k[9], r23 ^ k[35],
        r24 ^ k[30], &l3, &l28, &l10, &l18);
s7 (r23 ^ k[2], r24 ^ k[37], r25 ^ k[22], r26 ^ k[0], r27 ^ k[42],
        r28 ^ k[38], &l31, &l11, &l21, &l6);
s8 (r27 ^ k[16], r28 ^ k[43], r29 ^ k[44], r30 ^ k[1], r31 ^ k[7],
        r0 ^ k[28], &l4, &l26, &l14, &l20);
s1 (l31 ^ k[54], l0 ^ k[18], l1 ^ k[33], l2 ^ k[10], l3 ^ k[20],
        l4 ^ k[48], &r8, &r16, &r22, &r30);
s2 (l3 ^ k[34], l4 ^ k[13], l5 ^ k[4], l6 ^ k[55], l7 ^ k[46],
        l8 ^ k[26], &r12, &r27, &r1, &r17);
s3 (l7 ^ k[3], l8 ^ k[32], l9 ^ k[40], l10 ^ k[41], l11 ^ k[24],
        l12 ^ k[12], &r23, &r15, &r29, &r5);
s4 (l11 ^ k[11], l12 ^ k[5], l13 ^ k[6], l14 ^ k[39], l15 ^ k[47],
        l16 ^ k[27], &r25, &r19, &r9, &r0);
s5 (l15 ^ k[43], l16 ^ k[38], l17 ^ k[28], l18 ^ k[15], l19 ^ k[30],
        l20 ^ k[0], &r7, &r13, &r24, &r2);
s6 (l19 ^ k[21], l20 ^ k[36], l21 ^ k[31], l22 ^ k[16], l23 ^ k[42],
        l24 ^ k[37], &r3, &r28, &r10, &r18);
s7 (l23 ^ k[9], l24 ^ k[44], l25 ^ k[29], l26 ^ k[7], l27 ^ k[49],
        l28 ^ k[45], &r31, &r11, &r21, &r6);
s8 (l27 ^ k[23], l28 ^ k[50], l29 ^ k[51], l30 ^ k[8], l31 ^ k[14],
        l0 ^ k[35], &r4, &r26, &r14, &r20);
s1 (r31 ^ k[11], r0 ^ k[32], r1 ^ k[47], r2 ^ k[24], r3 ^ k[34],
        r4 ^ k[5], &l8, &l16, &l22, &l30);
s2 (r3 ^ k[48], r4 ^ k[27], r5 ^ k[18], r6 ^ k[12], r7 ^ k[3],
        r8 ^ k[40], &l12, &l27, &l1, &l17);
s3 (r7 ^ k[17], r8 ^ k[46], r9 ^ k[54], r10 ^ k[55], r11 ^ k[13],
        r12 ^ k[26], &l23, &l15, &l29, &l5);
s4 (r11 ^ k[25], r12 ^ k[19], r13 ^ k[20], r14 ^ k[53], r15 ^ k[4],
        r16 ^ k[41], &l25, &l19, &l9, &l0);
s5 (r15 ^ k[2], r16 ^ k[52], r17 ^ k[42], r18 ^ k[29], r19 ^ k[44],
        r20 ^ k[14], &l7, &l13, &l24, &l2);
s6 (r19 ^ k[35], r20 ^ k[50], r21 ^ k[45], r22 ^ k[30], r23 ^ k[1],
        r24 ^ k[51], &l3, &l28, &l10, &l18);
s7 (r23 ^ k[23], r24 ^ k[31], r25 ^ k[43], r26 ^ k[21], r27 ^ k[8],
        r28 ^ k[0], &l31, &l11, &l21, &l6);
s8 (r27 ^ k[37], r28 ^ k[9], r29 ^ k[38], r30 ^ k[22], r31 ^ k[28],
        r0 ^ k[49], &l4, &l26, &l14, &l20);
s1 (l31 ^ k[25], l0 ^ k[46], l1 ^ k[4], l2 ^ k[13], l3 ^ k[48],
        l4 ^ k[19], &r8, &r16, &r22, &r30);
s2 (l3 ^ k[5], l4 ^ k[41], l5 ^ k[32], l6 ^ k[26], l7 ^ k[17],
        l8 ^ k[54], &r12, &r27, &r1, &r17);
s3 (l7 ^ k[6], l8 ^ k[3], l9 ^ k[11], l10 ^ k[12], l11 ^ k[27],
        l12 ^ k[40], &r23, &r15, &r29, &r5);
s4 (l11 ^ k[39], l12 ^ k[33], l13 ^ k[34], l14 ^ k[10], l15 ^ k[18],
        l16 ^ k[55], &r25, &r19, &r9, &r0);
s5 (l15 ^ k[16], l16 ^ k[7], l17 ^ k[1], l18 ^ k[43], l19 ^ k[31],
        l20 ^ k[28], &r7, &r13, &r24, &r2);
s6 (l19 ^ k[49], l20 ^ k[9], l21 ^ k[0], l22 ^ k[44], l23 ^ k[15],
        l24 ^ k[38], &r3, &r28, &r10, &r18);
s7 (l23 ^ k[37], l24 ^ k[45], l25 ^ k[2], l26 ^ k[35], l27 ^ k[22],
        l28 ^ k[14], &r31, &r11, &r21, &r6);
s8 (l27 ^ k[51], l28 ^ k[23], l29 ^ k[52], l30 ^ k[36], l31 ^ k[42],
        l0 ^ k[8], &r4, &r26, &r14, &r20);
s1 (r31 ^ k[39], r0 ^ k[3], r1 ^ k[18], r2 ^ k[27], r3 ^ k[5],
        r4 ^ k[33], &l8, &l16, &l22, &l30);
```



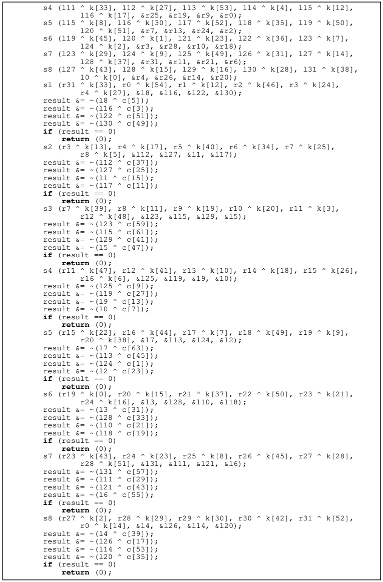
```
s2 (r3 ^ k[19], r4 ^ k[55], r5 ^ k[46], r6 ^ k[40], r7 ^ k[6],
        r8 ^ k[11], &l12, &l27, &l1, &l17);
s3 (r7 ^ k[20], r8 ^ k[17], r9 ^ k[25], r10 ^ k[26], r11 ^ k[41],
        r12 ^ k[54], &l23, &l15, &l29, &l5);
s4 (r11 ^ k[53], r12 ^ k[47], r13 ^ k[48], r14 ^ k[24], r15 ^ k[32],
        r16 ^ k[12], &l25, &l19, &l9, &l0);
s5 (r15 ^ k[30], r16 ^ k[21], r17 ^ k[15], r18 ^ k[2], r19 ^ k[45],
        r20 ^ k[42], &l7, &l13, &l24, &l2);
s6 (r19 ^ k[8], r20 ^ k[23], r21 ^ k[14], r22 ^ k[31], r23 ^ k[29],
        r24 ^ k[52], &l3, &l28, &l10, &l18);
s7 (r23 ^ k[51], r24 ^ k[0], r25 ^ k[16], r26 ^ k[49], r27 ^ k[36],
        r28 ^ k[28], &l31, &l11, &l21, &l6);
s8 (r27 ^ k[38], r28 ^ k[37], r29 ^ k[7], r30 ^ k[50], r31 ^ k[1],
        r0 ^ k[22], &l4, &l26, &l14, &l20);
s1 (l31 ^ k[53], l0 ^ k[17], l1 ^ k[32], l2 ^ k[41], l3 ^ k[19],
        l4 ^ k[47], &r8, &r16, &r22, &r30);
s2 (l3 ^ k[33], l4 ^ k[12], l5 ^ k[3], l6 ^ k[54], l7 ^ k[20],
        l8 ^ k[25], &r12, &r27, &r1, &r17);
s3 (l7 ^ k[34], l8 ^ k[6], l9 ^ k[39], l10 ^ k[40], l11 ^ k[55],
        l12 ^ k[11], &r23, &r15, &r29, &r5);
s4 (l11 ^ k[10], l12 ^ k[4], l13 ^ k[5], l14 ^ k[13], l15 ^ k[46],
        l16 ^ k[26], &r25, &r19, &r9, &r0);
s5 (l15 ^ k[44], l16 ^ k[35], l17 ^ k[29], l18 ^ k[16], l19 ^ k[0],
        l20 ^ k[1], &r7, &r13, &r24, &r2);
s6 (l19 ^ k[22], l20 ^ k[37], l21 ^ k[28], l22 ^ k[45], l23 ^ k[43],
        l24 ^ k[7], &r3, &r28, &r10, &r18);
s7 (l23 ^ k[38], l24 ^ k[14], l25 ^ k[30], l26 ^ k[8], l27 ^ k[50],
        l28 ^ k[42], &r31, &r11, &r21, &r6);
s8 (l27 ^ k[52], l28 ^ k[51], l29 ^ k[21], l30 ^ k[9], l31 ^ k[15],
        l0 ^ k[36], &r4, &r26, &r14, &r20);
s1 (r31 ^ k[10], r0 ^ k[6], r1 ^ k[46], r2 ^ k[55], r3 ^ k[33],
        r4 ^ k[4], &l8, &l16, &l22, &l30);
s2 (r3 ^ k[47], r4 ^ k[26], r5 ^ k[17], r6 ^ k[11], r7 ^ k[34],
        r8 ^ k[39], &l12, &l27, &l1, &l17);
s3 (r7 ^ k[48], r8 ^ k[20], r9 ^ k[53], r10 ^ k[54], r11 ^ k[12],
        r12 ^ k[25], &l23, &l15, &l29, &l5);
s4 (r11 ^ k[24], r12 ^ k[18], r13 ^ k[19], r14 ^ k[27], r15 ^ k[3],
        r16 ^ k[40], &l25, &l19, &l9, &l0);
s5 (r15 ^ k[31], r16 ^ k[49], r17 ^ k[43], r18 ^ k[30], r19 ^ k[14],
        r20 ^ k[15], &l7, &l13, &l24, &l2);
s6 (r19 ^ k[36], r20 ^ k[51], r21 ^ k[42], r22 ^ k[0], r23 ^ k[2],
        r24 ^ k[21], &l3, &l28, &l10, &l18);
s7 (r23 ^ k[52], r24 ^ k[28], r25 ^ k[44], r26 ^ k[22], r27 ^ k[9],
        r28 ^ k[1], &l31, &l11, &l21, &l6);
s8 (r27 ^ k[7], r28 ^ k[38], r29 ^ k[35], r30 ^ k[23], r31 ^ k[29],
        r0 ^ k[50], &l4, &l26, &l14, &l20);
s1 (l31 ^ k[24], l0 ^ k[20], l1 ^ k[3], l2 ^ k[12], l3 ^ k[47],
        l4 ^ k[18], &r8, &r16, &r22, &r30);
s2 (l3 ^ k[4], l4 ^ k[40], l5 ^ k[6], l6 ^ k[25], l7 ^ k[48],
        l8 ^ k[53], &r12, &r27, &r1, &r17);
s3 (l7 ^ k[5], l8 ^ k[34], l9 ^ k[10], l10 ^ k[11], l11 ^ k[26],
        l12 ^ k[39], &r23, &r15, &r29, &r5);
s4 (l11 ^ k[13], l12 ^ k[32], l13 ^ k[33], l14 ^ k[41], l15 ^ k[17],
        l16 ^ k[54], &r25, &r19, &r9, &r0);
s5 (l15 ^ k[45], l16 ^ k[8], l17 ^ k[2], l18 ^ k[44], l19 ^ k[28],
        l20 ^ k[29], &r7, &r13, &r24, &r2);
s6 (l19 ^ k[50], l20 ^ k[38], l21 ^ k[1], l22 ^ k[14], l23 ^ k[16],
        l24 ^ k[35], &r3, &r28, &r10, &r18);
s7 (l23 ^ k[7], l24 ^ k[42], l25 ^ k[31], l26 ^ k[36], l27 ^ k[23],
        l28 ^ k[15], &r31, &r11, &r21, &r6);
s8 (l27 ^ k[21], l28 ^ k[52], l29 ^ k[49], l30 ^ k[37], l31 ^ k[43],
        l0 ^ k[9], &r4, &r26, &r14, &r20);
s1 (r31 ^ k[6], r0 ^ k[27], r1 ^ k[10], r2 ^ k[19], r3 ^ k[54],
        r4 ^ k[25], &l8, &l16, &l22, &l30);
s2 (r3 ^ k[11], r4 ^ k[47], r5 ^ k[13], r6 ^ k[32], r7 ^ k[55],
        r8 ^ k[3], &l12, &l27, &l1, &l17);
s3 (r7 ^ k[12], r8 ^ k[41], r9 ^ k[17], r10 ^ k[18], r11 ^ k[33],
        r12 ^ k[46], &l23, &l15, &l29, &l5);
s4 (r11 ^ k[20], r12 ^ k[39], r13 ^ k[40], r14 ^ k[48], r15 ^ k[24],
        r16 ^ k[4], &l25, &l19, &l9, &l0);
s5 (r15 ^ k[52], r16 ^ k[15], r17 ^ k[9], r18 ^ k[51], r19 ^ k[35],
        r20 ^ k[36], &l7, &l13, &l24, &l2);
s6 (r19 ^ k[2], r20 ^ k[45], r21 ^ k[8], r22 ^ k[21], r23 ^ k[23],
        r24 ^ k[42], &l3, &l28, &l10, &l18);
```

Anatomy of a block cipher

The Real Thing



```
        s7 (r23 ^ k[14], r24 ^ k[49], r25 ^ k[38], r26 ^ k[43], r27 ^ k[30],
                r28 ^ k[22], &l31, &l11, &l21, &l6);
        s8 (r27 ^ k[28], r28 ^ k[0], r29 ^ k[1], r30 ^ k[44], r31 ^ k[50],
                r0 ^ k[16], &l4, &l26, &l14, &l20);
        s1 (l31 ^ k[20], l0 ^ k[41], l1 ^ k[24], l2 ^ k[33], l3 ^ k[11],
                l4 ^ k[39], &r8, &r16, &r22, &r30);
        s2 (l3 ^ k[25], l4 ^ k[4], l5 ^ k[27], l6 ^ k[46], l7 ^ k[12],
                l8 ^ k[17], &r12, &r27, &r1, &r17);
        s3 (l7 ^ k[26], l8 ^ k[55], l9 ^ k[6], l10 ^ k[32], l11 ^ k[47],
                l12 ^ k[3], &r23, &r15, &r29, &r5);
        s4 (l11 ^ k[34], l12 ^ k[53], l13 ^ k[54], l14 ^ k[5], l15 ^ k[13],
                l16 ^ k[18], &r25, &r19, &r9, &r0);
        s5 (l15 ^ k[7], l16 ^ k[29], l17 ^ k[23], l18 ^ k[38], l19 ^ k[49],
                l20 ^ k[50], &r7, &r13, &r24, &r2);
        s6 (l19 ^ k[16], l20 ^ k[0], l21 ^ k[22], l22 ^ k[35], l23 ^ k[37],
                l24 ^ k[1], &r3, &r28, &r10, &r18);
        s7 (l23 ^ k[28], l24 ^ k[8], l25 ^ k[52], l26 ^ k[2], l27 ^ k[44],
                l28 ^ k[36], &r31, &r11, &r21, &r6);
        s8 (l27 ^ k[42], l28 ^ k[14], l29 ^ k[15], l30 ^ k[31], l31 ^ k[9],
                l0 ^ k[30], &r4, &r26, &r14, &r20);
        s1 (r31 ^ k[34], r0 ^ k[55], r1 ^ k[13], r2 ^ k[47], r3 ^ k[25],
                r4 ^ k[53], &l8, &l16, &l22, &l30);
        s2 (r3 ^ k[39], r4 ^ k[18], r5 ^ k[41], r6 ^ k[3], r7 ^ k[26],
                r8 ^ k[6], &l12, &l27, &l1, &l17);
        s3 (r7 ^ k[40], r8 ^ k[12], r9 ^ k[20], r10 ^ k[46], r11 ^ k[4],
                r12 ^ k[17], &l23, &l15, &l29, &l5);
        s4 (r11 ^ k[48], r12 ^ k[10], r13 ^ k[11], r14 ^ k[19], r15 ^ k[27],
                r16 ^ k[32], &l25, &l19, &l9, &l0);
        s5 (r15 ^ k[21], r16 ^ k[43], r17 ^ k[37], r18 ^ k[52], r19 ^ k[8],
                r20 ^ k[9], &l7, &l13, &l24, &l2);
        s6 (r19 ^ k[30], r20 ^ k[14], r21 ^ k[36], r22 ^ k[49], r23 ^ k[51],
                r24 ^ k[15], &l3, &l28, &l10, &l18);
        s7 (r23 ^ k[42], r24 ^ k[22], r25 ^ k[7], r26 ^ k[16], r27 ^ k[31],
                r28 ^ k[50], &l31, &l11, &l21, &l6);
        s8 (r27 ^ k[1], r28 ^ k[28], r29 ^ k[29], r30 ^ k[45], r31 ^ k[23],
                r0 ^ k[44], &l4, &l26, &l14, &l20);
        s1 (l31 ^ k[48], l0 ^ k[12], l1 ^ k[27], l2 ^ k[4], l3 ^ k[39],
                l4 ^ k[10], &r8, &r16, &r22, &r30);
        s2 (l3 ^ k[53], l4 ^ k[32], l5 ^ k[55], l6 ^ k[17], l7 ^ k[40],
                l8 ^ k[20], &r12, &r27, &r1, &r17);
        s3 (l7 ^ k[54], l8 ^ k[26], l9 ^ k[34], l10 ^ k[3], l11 ^ k[18],
                l12 ^ k[6], &r23, &r15, &r29, &r5);
        s4 (l11 ^ k[5], l12 ^ k[24], l13 ^ k[25], l14 ^ k[33], l15 ^ k[41],
                l16 ^ k[46], &r25, &r19, &r9, &r0);
        s5 (l15 ^ k[35], l16 ^ k[2], l17 ^ k[51], l18 ^ k[7], l19 ^ k[22],
                l20 ^ k[23], &r7, &r13, &r24, &r2);
        s6 (l19 ^ k[44], l20 ^ k[28], l21 ^ k[50], l22 ^ k[8], l23 ^ k[38],
                l24 ^ k[29], &r3, &r28, &r10, &r18);
        s7 (l23 ^ k[1], l24 ^ k[36], l25 ^ k[21], l26 ^ k[30], l27 ^ k[45],
                l28 ^ k[9], &r31, &r11, &r21, &r6);
        s8 (l27 ^ k[15], l28 ^ k[42], l29 ^ k[43], l30 ^ k[0], l31 ^ k[37],
                l0 ^ k[31], &r4, &r26, &r14, &r20);
        s1 (r31 ^ k[5], r0 ^ k[26], r1 ^ k[41], r2 ^ k[18], r3 ^ k[53],
                r4 ^ k[24], &l8, &l16, &l22, &l30);
        s2 (r3 ^ k[10], r4 ^ k[46], r5 ^ k[12], r6 ^ k[6], r7 ^ k[54],
                r8 ^ k[34], &l12, &l27, &l1, &l17);
        s3 (r7 ^ k[11], r8 ^ k[40], r9 ^ k[48], r10 ^ k[17], r11 ^ k[32],
                r12 ^ k[20], &l23, &l15, &l29, &l5);
        s4 (r11 ^ k[19], r12 ^ k[13], r13 ^ k[39], r14 ^ k[47], r15 ^ k[55],
                r16 ^ k[3], &l25, &l19, &l9, &l0);
        s5 (r15 ^ k[49], r16 ^ k[16], r17 ^ k[38], r18 ^ k[21], r19 ^ k[36],
                r20 ^ k[37], &l7, &l13, &l24, &l2);
        s6 (r19 ^ k[31], r20 ^ k[42], r21 ^ k[9], r22 ^ k[22], r23 ^ k[52],
                r24 ^ k[43], &l3, &l28, &l10, &l18);
        s7 (r23 ^ k[15], r24 ^ k[50], r25 ^ k[35], r26 ^ k[44], r27 ^ k[0],
                r28 ^ k[23], &l31, &l11, &l21, &l6);
        s8 (r27 ^ k[29], r28 ^ k[1], r29 ^ k[2], r30 ^ k[14], r31 ^ k[51],
                r0 ^ k[45], &l4, &l26, &l14, &l20);
        s1 (l31 ^ k[19], l0 ^ k[40], l1 ^ k[55], l2 ^ k[32], l3 ^ k[10],
                l4 ^ k[13], &r8, &r16, &r22, &r30);
        s2 (l3 ^ k[24], l4 ^ k[3], l5 ^ k[26], l6 ^ k[20], l7 ^ k[11],
                l8 ^ k[48], &r12, &r27, &r1, &r17);
        s3 (l7 ^ k[25], l8 ^ k[54], l9 ^ k[5], l10 ^ k[6], l11 ^ k[46],
                l12 ^ k[34], &r23, &r15, &r29, &r5);
```

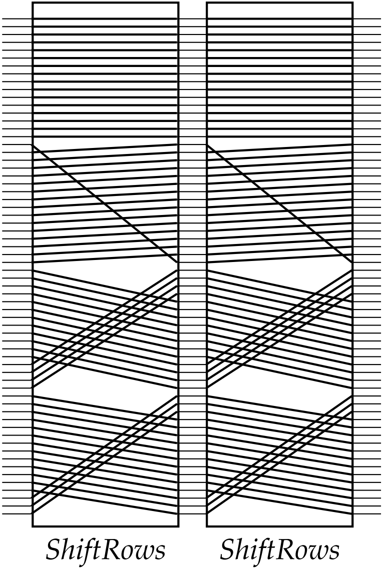


```
        s4 (l11 ^ k[33], l12 ^ k[27], l13 ^ k[53], l14 ^ k[4], l15 ^ k[12],
                l16 ^ k[17], &r25, &r19, &r9, &r0);
        s5 (l15 ^ k[8], l16 ^ k[30], l17 ^ k[52], l18 ^ k[35], l19 ^ k[50],
                l20 ^ k[51], &r7, &r13, &r24, &r2);
        s6 (l19 ^ k[45], l20 ^ k[1], l21 ^ k[23], l22 ^ k[36], l23 ^ k[7],
                l24 ^ k[2], &r3, &r28, &r10, &r18);
        s7 (l23 ^ k[29], l24 ^ k[9], l25 ^ k[49], l26 ^ k[31], l27 ^ k[14],
                l28 ^ k[37], &r31, &r11, &r21, &r6);
        s8 (l27 ^ k[43], l28 ^ k[15], l29 ^ k[16], l30 ^ k[28], l31 ^ k[38],
                l0 ^ k[0], &r4, &r26, &r14, &r20);
        s1 (r31 ^ k[33], r0 ^ k[54], r1 ^ k[12], r2 ^ k[46], r3 ^ k[24],
                r4 ^ k[27], &l8, &l16, &l22, &l30);
        result &= ~(l8 ^ c[5]);
        result &= ~(l16 ^ c[3]);
        result &= ~(l22 ^ c[51]);
        result &= ~(l30 ^ c[49]);
        if (result == 0)
            return (0);
        s2 (r3 ^ k[13], r4 ^ k[17], r5 ^ k[40], r6 ^ k[34], r7 ^ k[25],
                r8 ^ k[5], &l12, &l27, &l1, &l17);
        result &= ~(l12 ^ c[37]);
        result &= ~(l27 ^ c[25]);
        result &= ~(l1 ^ c[15]);
        result &= ~(l17 ^ c[11]);
        if (result == 0)
            return (0);
        s3 (r7 ^ k[39], r8 ^ k[11], r9 ^ k[19], r10 ^ k[20], r11 ^ k[3],
                r12 ^ k[48], &l23, &l15, &l29, &l5);
        result &= ~(l23 ^ c[59]);
        result &= ~(l15 ^ c[61]);
        result &= ~(l29 ^ c[41]);
        result &= ~(l5 ^ c[47]);
        if (result == 0)
            return (0);
        s4 (r11 ^ k[47], r12 ^ k[41], r13 ^ k[10], r14 ^ k[18], r15 ^ k[26],
                r16 ^ k[6], &l25, &l19, &l9, &l0);
        result &= ~(l25 ^ c[9]);
        result &= ~(l19 ^ c[27]);
        result &= ~(l9 ^ c[13]);
        result &= ~(l0 ^ c[7]);
        if (result == 0)
            return (0);
        s5 (r15 ^ k[22], r16 ^ k[44], r17 ^ k[7], r18 ^ k[49], r19 ^ k[9],
                r20 ^ k[38], &l7, &l13, &l24, &l2);
        result &= ~(l7 ^ c[63]);
        result &= ~(l13 ^ c[45]);
        result &= ~(l24 ^ c[1]);
        result &= ~(l2 ^ c[23]);
        if (result == 0)
            return (0);
        s6 (r19 ^ k[0], r20 ^ k[15], r21 ^ k[37], r22 ^ k[50], r23 ^ k[21],
                r24 ^ k[16], &l3, &l28, &l10, &l18);
        result &= ~(l3 ^ c[31]);
        result &= ~(l28 ^ c[33]);
        result &= ~(l10 ^ c[21]);
        result &= ~(l18 ^ c[19]);
        if (result == 0)
            return (0);
        s7 (r23 ^ k[43], r24 ^ k[23], r25 ^ k[8], r26 ^ k[45], r27 ^ k[28],
                r28 ^ k[51], &l31, &l11, &l21, &l6);
        result &= ~(l31 ^ c[57]);
        result &= ~(l11 ^ c[29]);
        result &= ~(l21 ^ c[43]);
        result &= ~(l6 ^ c[55]);
        if (result == 0)
            return (0);
        s8 (r27 ^ k[2], r28 ^ k[29], r29 ^ k[30], r30 ^ k[42], r31 ^ k[52],
                r0 ^ k[14], &l4, &l26, &l14, &l20);
        result &= ~(l4 ^ c[39]);
        result &= ~(l26 ^ c[17]);
        result &= ~(l14 ^ c[53]);
        result &= ~(l20 ^ c[35]);
        if (result == 0)
            return (0);
```

Bitsliced optimization

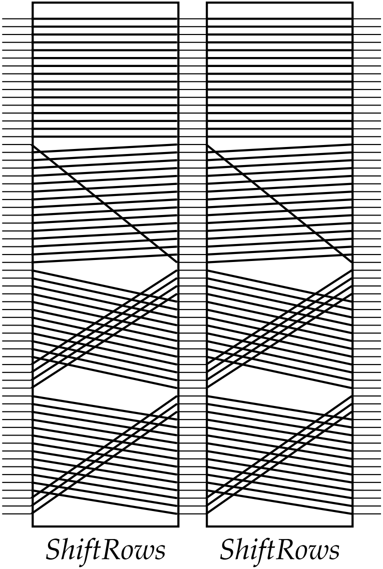
Unrolling & Inlining

```
node ShiftRows_x2 (plain:b64)
            returns (cipher:b64)
let
    forall i in [0,1] {
      plain = ShiftRows(plain)
    }
    cipher = plain
tel
```



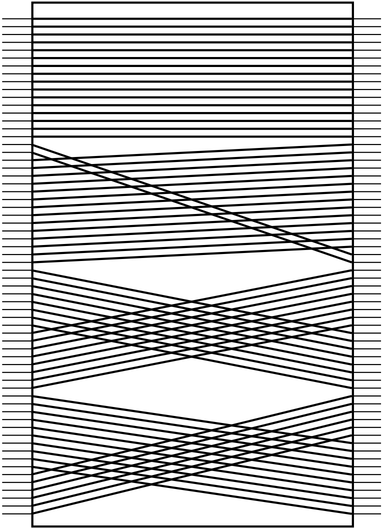
Unrolling & Inlining

```
node ShiftRows_x2 (plain:b64)
            returns (cipher:b64)
let
    forall i in [0,1] {
      tmp[0] = plain[0];
      tmp[1] = plain[1];
      ...
      tmp[16] = plain[17];
      tmp[17] = plain[18];
      ...
      tmp[63] = plain[60];
      plain = tmp;
    }
    cipher = plain
tel
```



Unrolling & Inlining

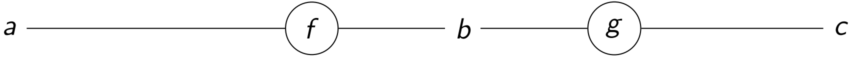
```
node ShiftRows_x2 (plain:b64)
            returns (cipher:b64)
let
    cipher[0] = plain[0];
    cipher[1] = plain[1];
    ...
    cipher[16] = plain[18];
    cipher[17] = plain[19];
    ...
    cipher[63] = plain[57];
tel
```



ShiftRows (x2)

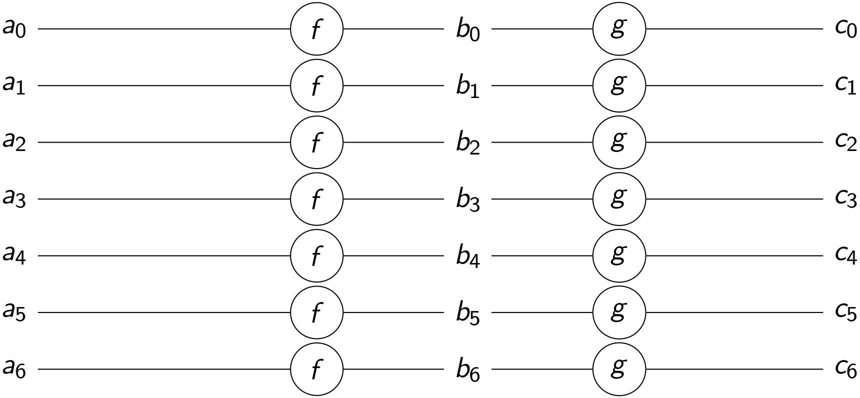
Scheduling bitsliced code

```
// Suppose f: b1 -> b1 and g: b1 -> b1
node my_cipher (a:b7) returns (c:b7)
let     b = f(a);
        c = g(b);    tel
```



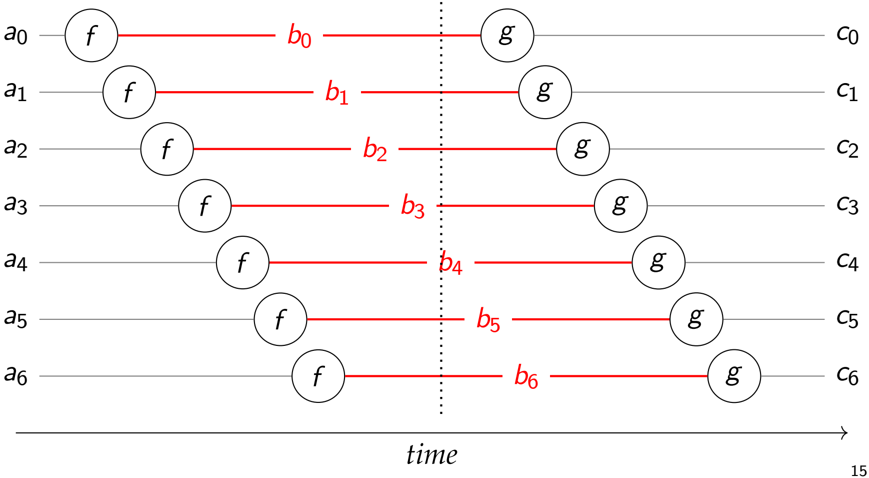
Scheduling bitsliced code

```
// Suppose f: b1 -> b1 and g: b1 -> b1
node my_cipher (a:b7) returns (c:b7)
let     b = f(a);
        c = g(b);    tel
```



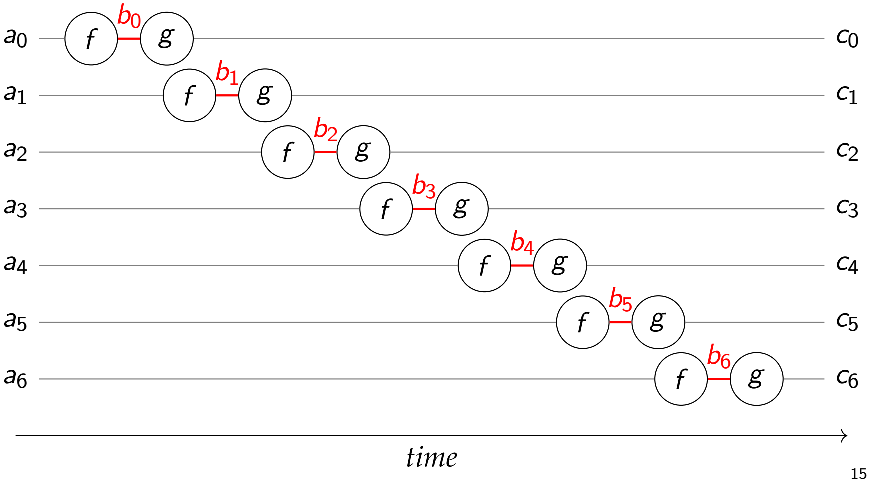
Scheduling bitsliced code

```
// Suppose f: b1 -> b1 and g: b1 -> b1
node my_cipher (a:b7) returns (c:b7)
let     b = f(a);
        c = g(b);    tel
```



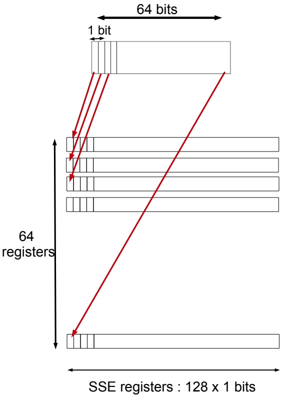
Scheduling bitsliced code

```
// Suppose f: b1 -> b1 and g: b1 -> b1
node my_cipher (a:b7) returns (c:b7)
let     b = f(a);
        c = g(b);    tel
```



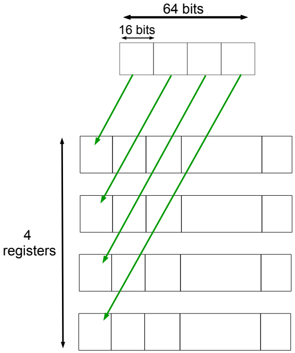
Making larger slices

Parallelization strategies



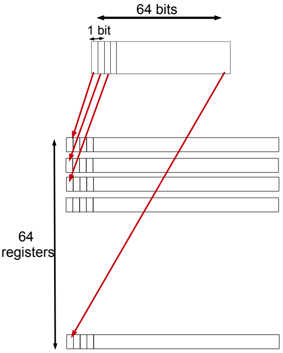
**u1x64
(bitslicing)**

Parallelization strategies



SSE registers : 8 x 16 bits

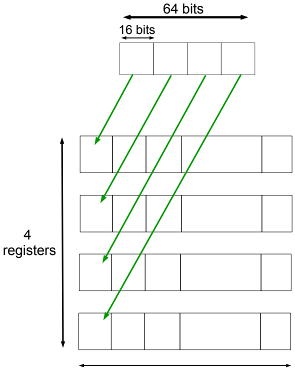
u_V16x4
(V-slicing)



SSE registers : 128 x 1 bits

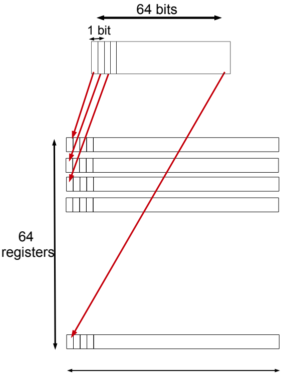
u1x64
(bitslicing)

Parallelization strategies



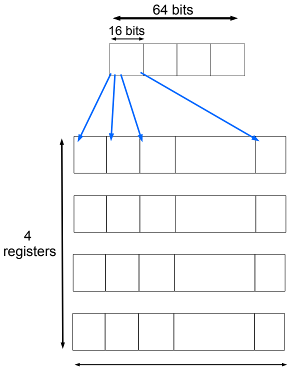
SSE registers : 8 x 16 bits

u_V16x4
(V-slicing)



SSE registers : 128 x 1 bits

u1x64
(bitslicing)



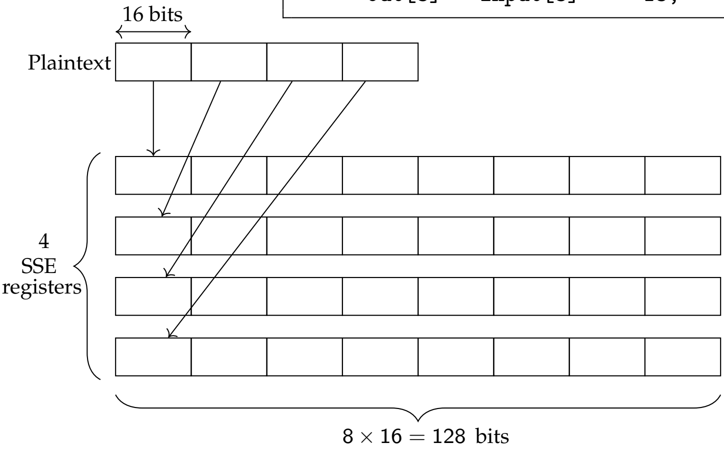
SSE registers : 16 x 8 bits

u_H16x4
(H-slicing)

V-slicing

ShiftRows in Vertical mode

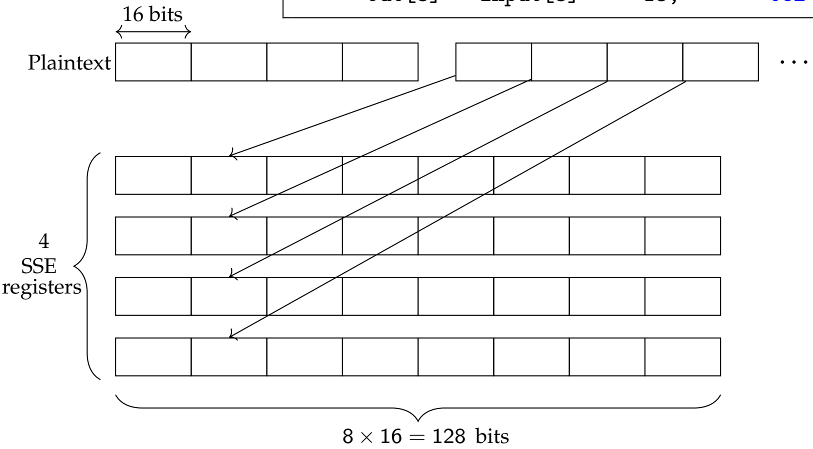
```
node ShiftRows (input:uV16x4) : (out:uV16x4)
let   out[0] = input[0];
      out[1] = input[1] <<< 1;
      out[2] = input[2] <<< 12;
      out[3] = input[3] <<< 13;        tel
```



V-slicing

ShiftRows in Vertical mode

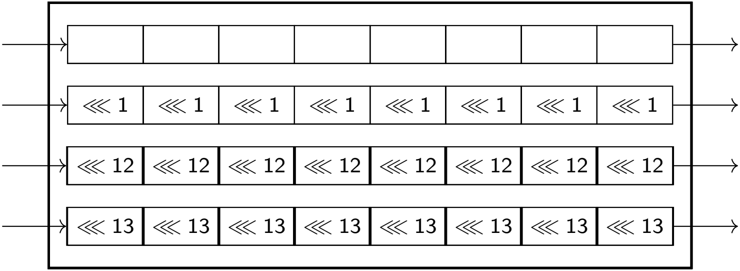
```
node ShiftRows (input:uV16x4) : (out:uV16x4)
let   out[0] = input[0];
      out[1] = input[1] <<< 1;
      out[2] = input[2] <<< 12;
      out[3] = input[3] <<< 13;         tel
```



V-slicing

ShiftRows in Vertical mode

```
node ShiftRows (input:uV16x4) : (out:uV16x4)
let   out[0] = input[0];
      out[1] = input[1] <<< 1;
      out[2] = input[2] <<< 12;
      out[3] = input[3] <<< 13;        tel
```



⋘ 1	⋘ 1	⋘ 1	⋘ 1	⋘ 1	⋘ 1	⋘ 1	⋘ 1
⋘ 12	⋘ 12	⋘ 12	⋘ 12	⋘ 12	⋘ 12	⋘ 12	⋘ 12
⋘ 13	⋘ 13	⋘ 13	⋘ 13	⋘ 13	⋘ 13	⋘ 13	⋘ 13

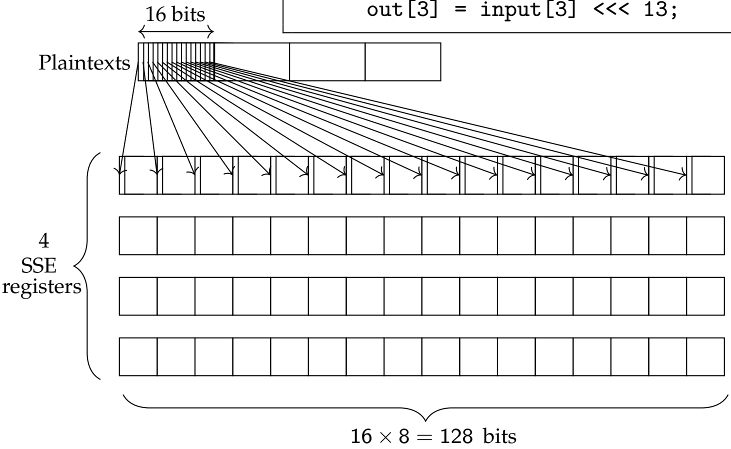
ShiftRows

```
__m128i _mm_sll_epi16 (__m128i a, __m128i count)
```

H-slicing

ShiftRows in Horizontal mode

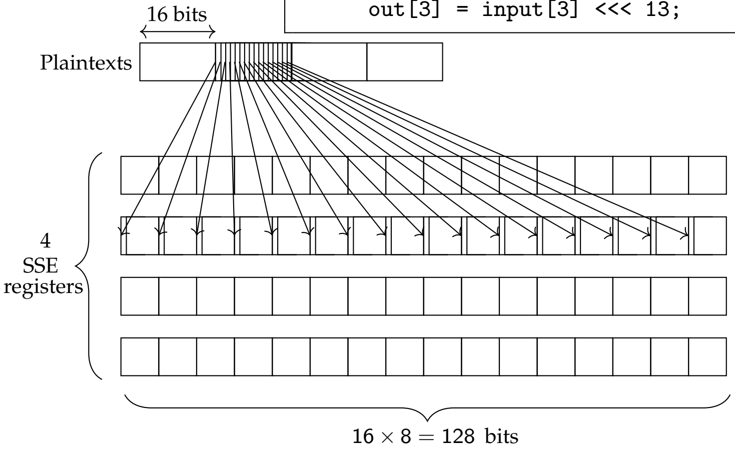
```
node ShiftRows (input:u_H16x4) : (out:u_H16x4)
let   out[0] = input[0];
      out[1] = input[1] <<< 1;
      out[2] = input[2] <<< 12;
      out[3] = input[3] <<< 13;        tel
```



H-slicing

ShiftRows in Horizontal mode

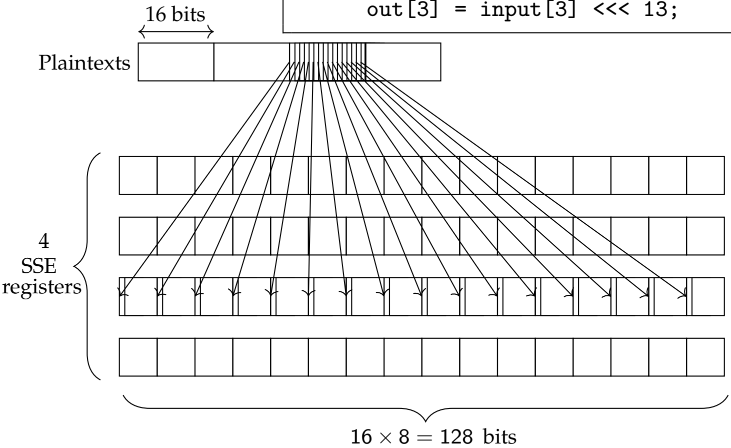
```
node ShiftRows (input:uH16x4) : (out:uH16x4)
let   out[0] = input[0];
      out[1] = input[1] <<< 1;
      out[2] = input[2] <<< 12;
      out[3] = input[3] <<< 13;        tel
```



H-slicing

ShiftRows in Horizontal mode

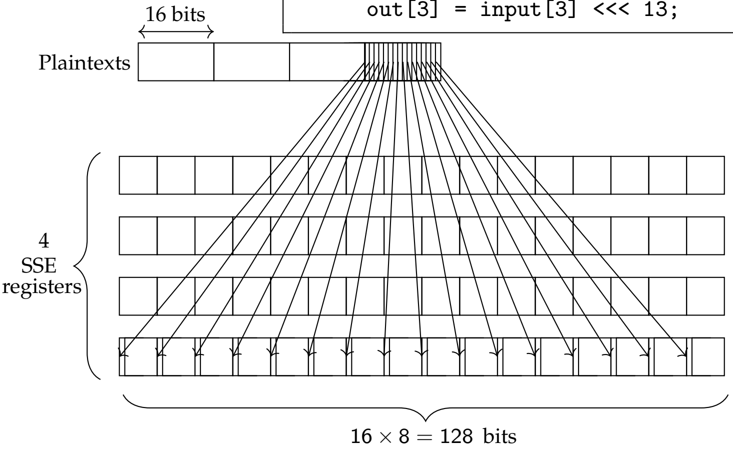
```
node ShiftRows (input:uH16x4) : (out:uH16x4)
let   out[0] = input[0];
      out[1] = input[1] <<< 1;
      out[2] = input[2] <<< 12;
      out[3] = input[3] <<< 13;        tel
```



H-slicing

ShiftRows in Horizontal mode

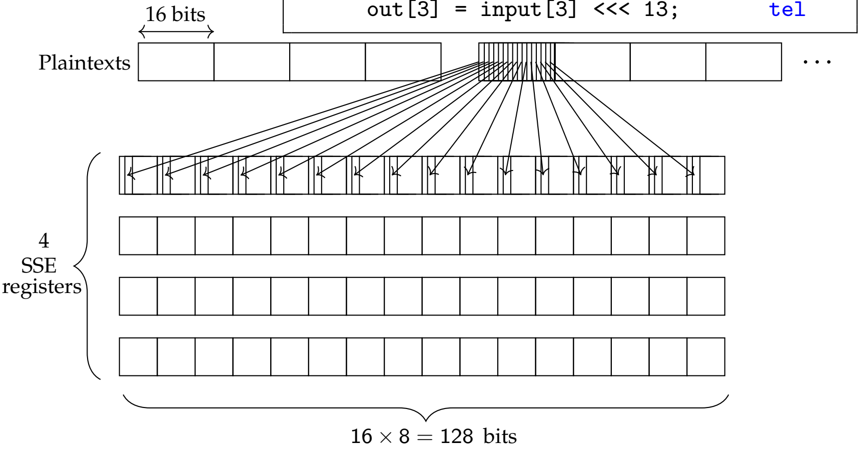
```
node ShiftRows (input:u_H16x4) : (out:u_H16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



H-slicing

ShiftRows in Horizontal mode

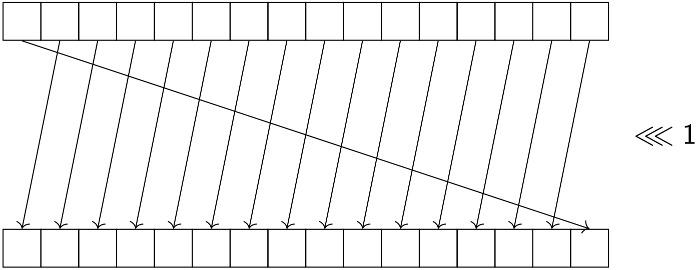
```
node ShiftRows (input:u_H16x4) : (out:u_H16x4)
let   out[0] = input[0];
      out[1] = input[1] <<< 1;
      out[2] = input[2] <<< 12;
      out[3] = input[3] <<< 13;        tel
```



H-slicing

ShiftRows in Horizontal mode

```
node ShiftRows (input:u_H16x4) : (out:u_H16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;         tel
```

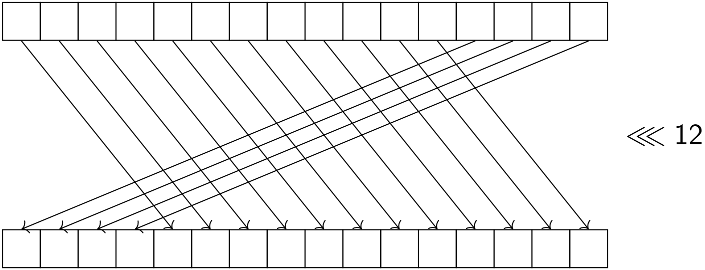


```
__m128i _mm_shuffle_epi8 (__m128i a, __m128i b)
```

H-slicing

ShiftRows in Horizontal mode

```
node ShiftRows (input:uH16x4) : (out:uH16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```

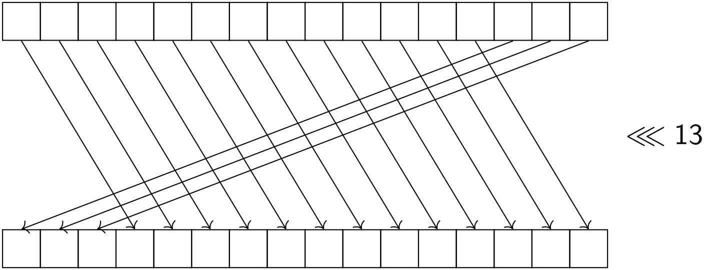


```
__m128i _mm_shuffle_epi8 (__m128i a, __m128i b)
```

H-slicing

ShiftRows in Horizontal mode

```
node ShiftRows (input:u_H16x4) : (out:u_H16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



```
__m128i _mm_shuffle_epi8 (__m128i a, __m128i b)
```

Quick Peek at the Language

```
node ShiftRows (input:u16x4)
       returns (out:u16x4)
vars
let
    out[0] = input[0];
    out[1] = input[1] <<< 1;
    out[2] = input[2] <<< 12;
    out[3] = input[3] <<< 13;
tel
```

```
table SubColumn (input:v4)
       returns  (out:v4) {
    6, 5, 12, 10, 1, 14, 7, 9,
    11, 0, 3, 13, 8, 15, 4, 2
}
```

```
node Rectangle (plain:u16x4,
                key  :u16x4[26])
       returns (cipher:u16x4)
vars
    round : u16x4[26]
let
    round[0] = plain;
    forall i in [0,24] {
      round[i+1] =
         ShiftRows(
           SubColumn(
             round[i] ^ key[i]
           )
         )
    }
    cipher = round[25] ^ key[25]
tel
```

Quick Peek at the Language

```
node ShiftRows (input:u'D16x4)
       returns (out:u'D16x4)
vars
let
    out[0] = input[0];
    out[1] = input[1] <<< 1;
    out[2] = input[2] <<< 12;
    out[3] = input[3] <<< 13;
tel
```

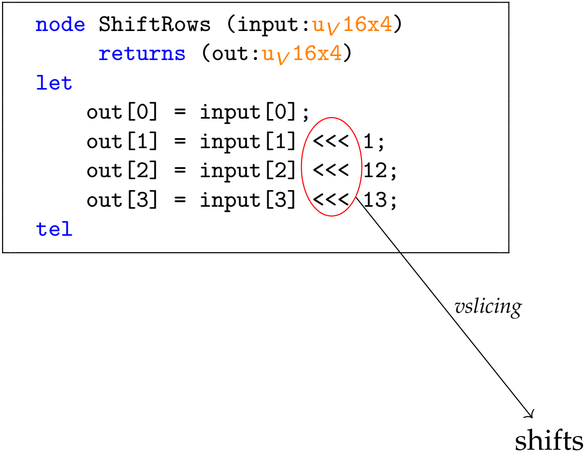
```
table SubColumn (input:u'D'mx4)
       returns  (out:u'D'mx4) {
    6, 5, 12, 10, 1, 14, 7, 9,
    11, 0, 3, 13, 8, 15, 4, 2
}
```

```
node Rectangle (plain:u'D16x4,
                key  :u'D16x4[26])
       returns (cipher:u'D16x4)
vars
    round : u'D16x4[26]
let
    round[0] = plain;
    forall i in [0,24] {
      round[i+1] =
         ShiftRows(
           SubColumn(
             round[i] ^ key[i]
           )
         )
    }
    cipher = round[25] ^ key[25]
tel
```

Quick Peek at the Language

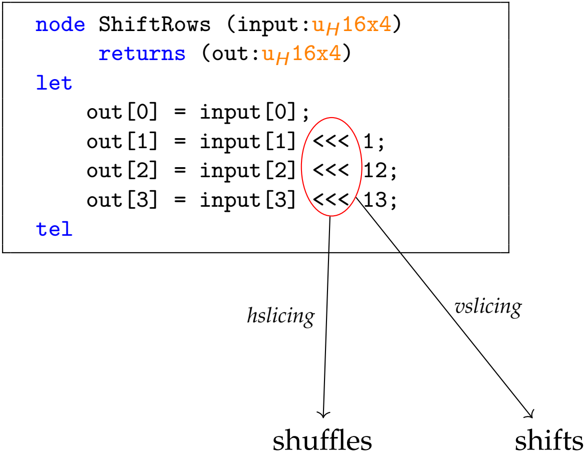
```
node ShiftRows (input:u16x4)
     returns (out:u16x4)
let
    out[0] = input[0];
    out[1] = input[1] <<< 1;
    out[2] = input[2] <<< 12;
    out[3] = input[3] <<< 13;
tel
```

Quick Peek at the Language



```
node ShiftRows (input:uV16x4)
     returns (out:uV16x4)
let
    out[0] = input[0];
    out[1] = input[1] <<< 1;
    out[2] = input[2] <<< 12;
    out[3] = input[3] <<< 13;
tel
```

Quick Peek at the Language



```
node ShiftRows (input:u_H16x4)
     returns (out:u_H16x4)
let
    out[0] = input[0];
    out[1] = input[1] <<< 1;
    out[2] = input[2] <<< 12;
    out[3] = input[3] <<< 13;
tel
```

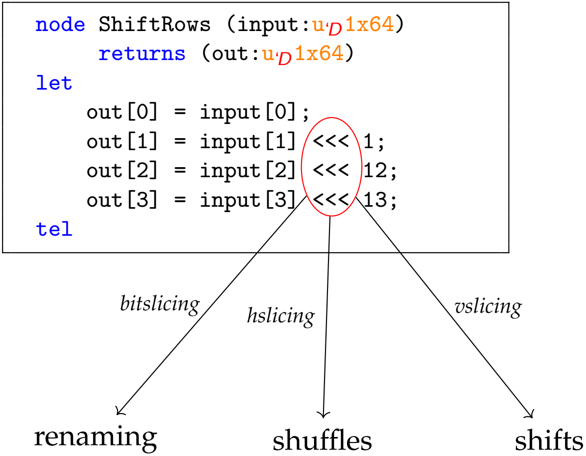
hslicing

vslicing

shuffles

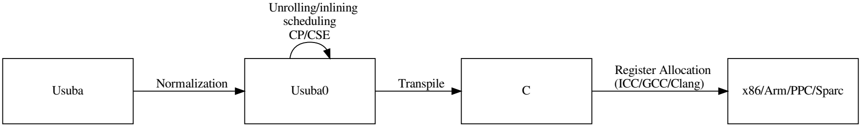
shifts

Quick Peek at the Language

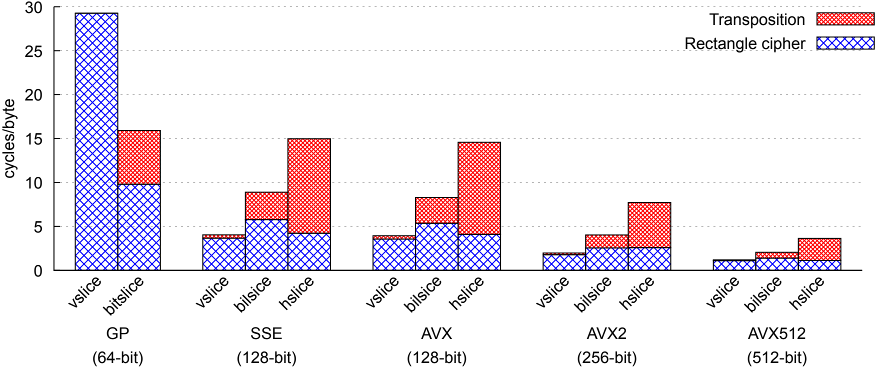


```
node ShiftRows (input:u'D1x64)
     returns (out:u'D1x64)
let
    out[0] = input[0];
    out[1] = input[1] <<< 1;
    out[2] = input[2] <<< 12;
    out[3] = input[3] <<< 13;
tel
```

m-sliced optimization



Monomorphization

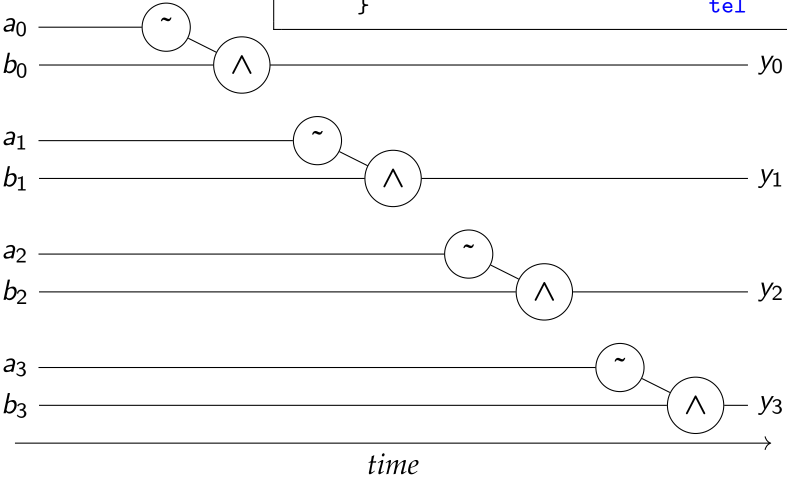


```
node Rectangle (plain  : u16x4, key : u16x4[26],
                cipher : u16x4)
```

```
void RectangleV (__m256i plain[4],  __m256i key[26][4],
                 __m256i cipher[4])

void RectangleB (__m128i plain[64], __m128i key[26][64],
                 __m128i cipher[64])
```

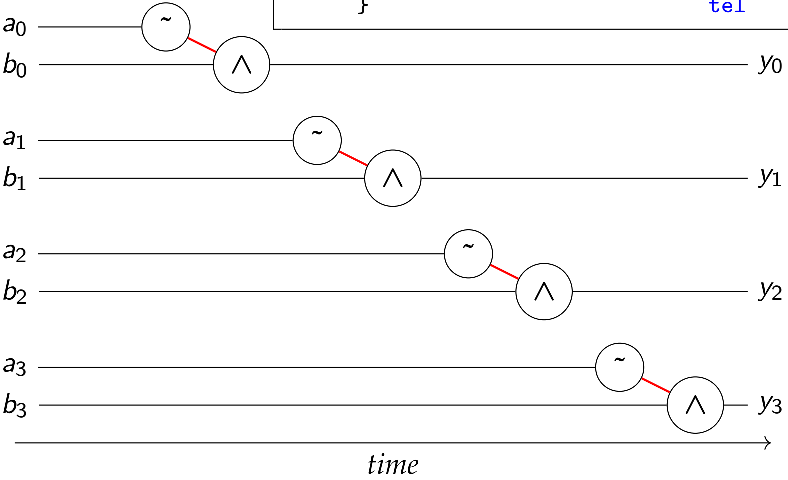
Scheduling *m*-sliced code

```
node my_cipher (a,b:b4) returns (y:b4)
let   forall i in [0, 3] {
          tmp  = ~ a[i];
          y[i] = tmp  ^ b[i];
      }                               tel
```



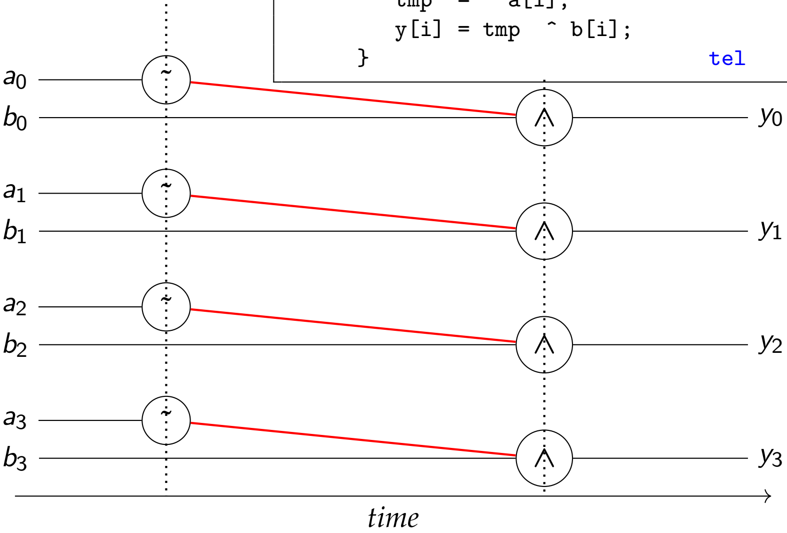
Scheduling *m*-sliced code

```
node my_cipher (a,b:b4) returns (y:b4)
let   forall i in [0, 3] {
         tmp  = ~ a[i];
         y[i] = tmp  ^ b[i];
      }                         tel
```



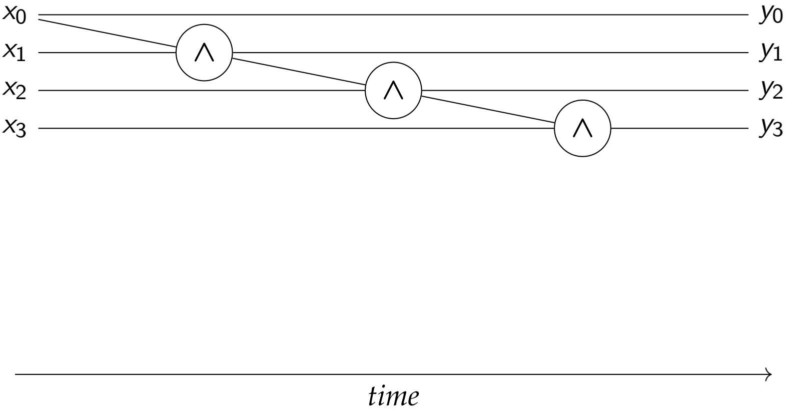
Scheduling *m*-sliced code

```
node my_cipher (a,b:b4) returns (y:b4)
let   forall i in [0, 3] {
          tmp  = ~ a[i];
          y[i] = tmp  ^ b[i];
      }                           tel
```



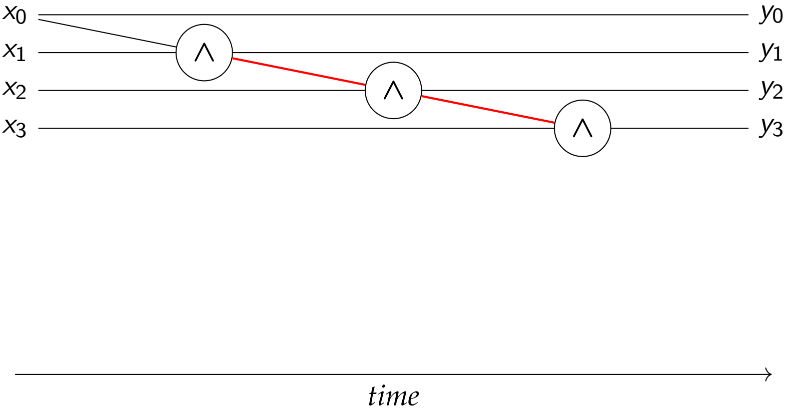
Interleaving

```
node my_cipher (x:b4) returns (y:b4)
let     y[0] = x[0];
        forall i in [1, 3] {
            y[i] = y[i-1] ^ x[i];
        }                                  tel
```



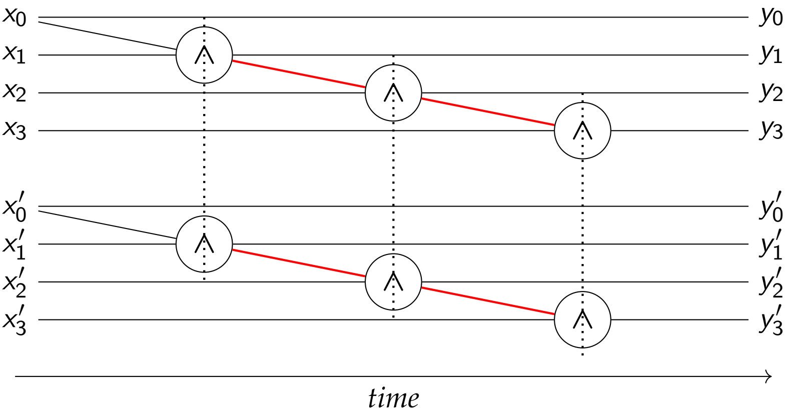
Interleaving

```
node my_cipher (x:b4) returns (y:b4)
let    y[0] = x[0];
       forall i in [1, 3] {
           y[i] = y[i-1] ^ x[i];
       }                             tel
```



Interleaving

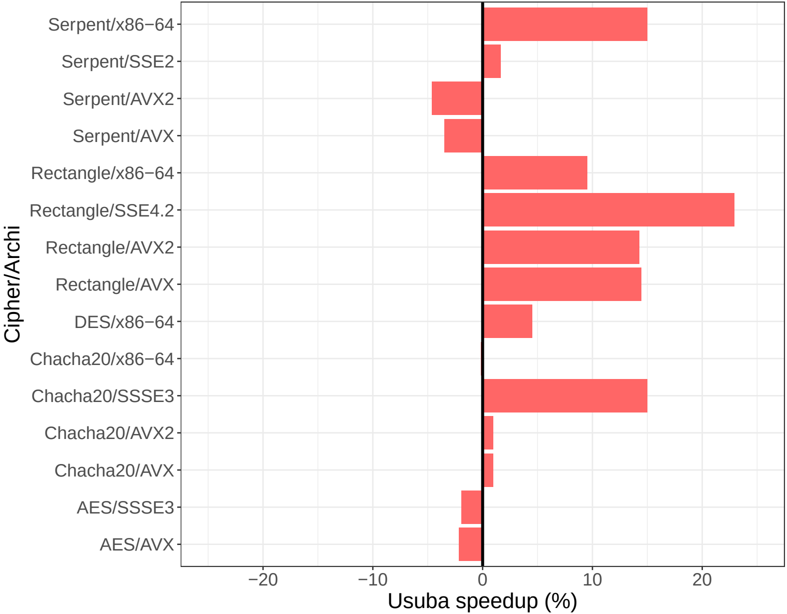
```
node my_cipher (x:b4) returns (y:b4)
let     y[0] = x[0];
        forall i in [1, 3] {
            y[i] = y[i-1] ^ x[i];
        }                             tel
```



Evaluation & Conclusion

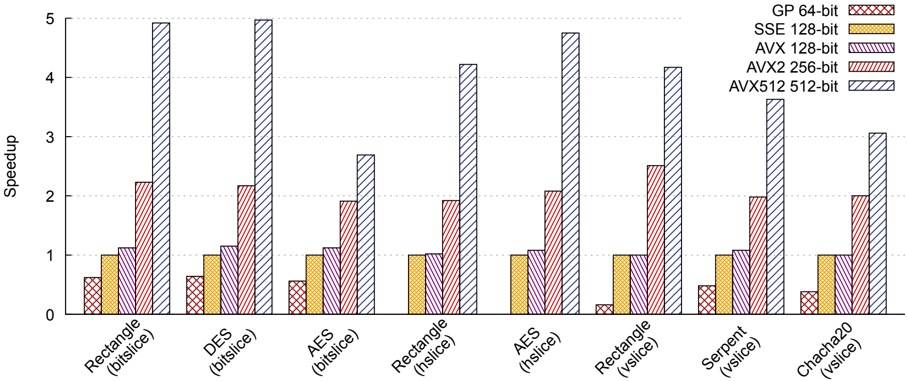
Evaluation

Usuba *vs.* Reference



Evaluation

Scalability



Conclusion

Usuba:

- High-level description of combinational circuits
- General model of *m*-slicing
- Generates optimized C code

Cipher	Mode	CC	Inline	Unroll	Interleave	Schedule
DES	bitslice	`Clang`	✓	✓		✓
AES	bitslice	`Clang`	✓	✓		✓
	hslice	`Clang`	✓	✓		✓
Rectangle	bitslice	`ICC`	✓	✓		✓
	hslice	`GCC`			✓	✓
	vslice	`Clang`			✓	✓
Chacha20	vslice	`ICC`	✓	✓		✓
Serpent	vslice	`Clang`		✓	✓	

Take-aways

Satisfying:

- Simple programming model / language
- Compiler exploits these invariants
- Correctness: equivalence of combinational circuits
- Itself a back-end for further transformations

(aggregated bitslice model)

Disappointing:

- Do we really need a language for that?
- How to achieve economies of scale?
- How to interact with sequential code?

(e.g.: crypto runtime)

Our roadmap

1. Develop bitslicing as a programming model
    - Protection against faults
    - Protection against side-channels
2. Take back control!
    - Custom register allocation
    - Bypass C / target Jasmin
    - End-to-end correctness proof *(without the chains)*
3. Beyond data parallelism
    - Factor in the crypto runtime
    - Target embedded devices
    - Find a suitable host
4. ???
5. Turing award *(see Patterson & Hennessy's lecture)*