# Usuba
## High-Throughput and Constant-Time Ciphers, by Construction

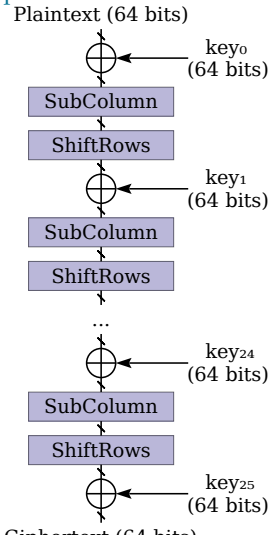
Darius Mercadier  Pierre-Évariste Dagand

LIP6 – CNRS – Inria
Sorbonne Université

June 24, 2019

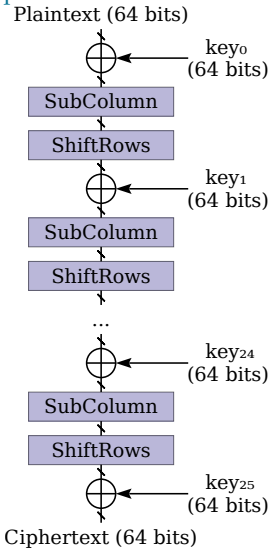
# Anatomy of a block cipher

## The Rectangle cipher



# Anatomy of a block cipher

## The Rectangle cipher

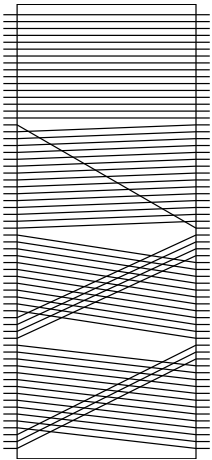



USUBA

```
node Rectangle (plain:b64,
                key  :b64[26])
       returns (cipher:b64)
vars
    round : b64[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
```

# Anatomy of a block cipher

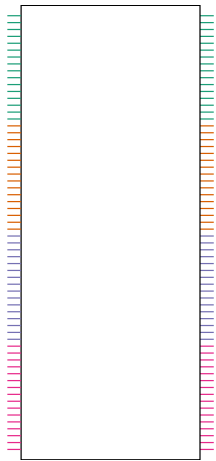
## Rectangle/ShiftRows



```
void ShiftRows(bool a[64], bool b[64]) {
  b[0]  = a[0];
  b[1]  = a[1];
  b[2]  = a[2];
  b[3]  = a[3];
  b[4]  = a[4];
  b[5]  = a[5];
  ...
  b[59] = a[56];
  b[60] = a[57];
  b[61] = a[58];
  b[62] = a[59];
  b[63] = a[60];
}
```

# Anatomy of a block cipher

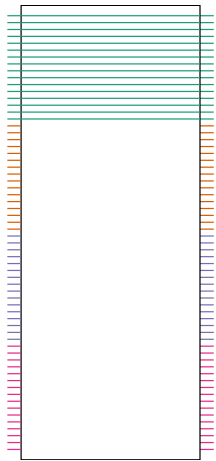
## Rectangle/ShiftRows



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

# Anatomy of a block cipher

## Rectangle/ShiftRows



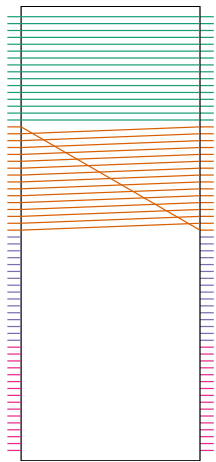
```
node ShiftRows (input:u16x4)
   returns     (out:u16x4)
let
    out[0] = input[0];



tel
```

# Anatomy of a block cipher

## Rectangle/ShiftRows

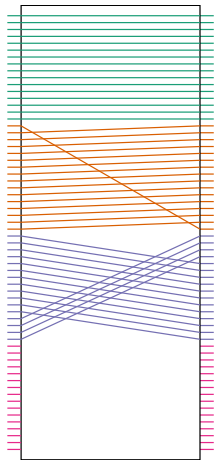


```
node ShiftRows (input:u16x4)
   returns     (out:u16x4)
let
    out[0] = input[0];
    out[1] = input[1] <<< 1;


tel
```

# Anatomy of a block cipher

## Rectangle/ShiftRows

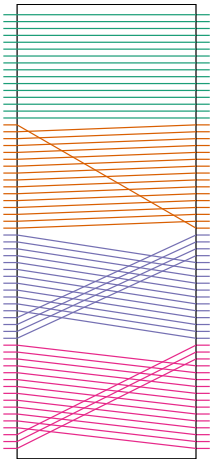


```
node ShiftRows (input:u16x4)
   returns     (out:u16x4)
let
    out[0] = input[0];
    out[1] = input[1] <<< 1;
    out[2] = input[2] <<< 12;

tel
```

# Anatomy of a block cipher

## Rectangle/ShiftRows



```
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/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 |

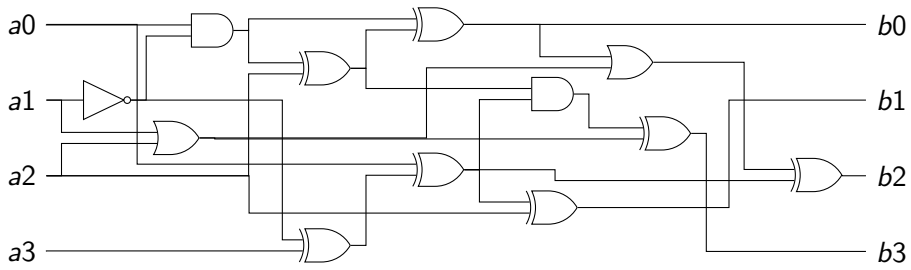
Caution: lookup tables are **strictly forbidden**!

# Anatomy of a block cipher

## Rectangle/SubColumn

The S-box used in RECTANGLE is a 4-bit to 4-bit S-box $S : F_2^4 \rightarrow F_2^4$. 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 |



# 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 |

```
void SubColumn(bool *a0, bool *a1,
               bool *a2, bool *a3) {
    bool t1, t2, t3, t5, t6, t8, t9, t11;
    bool a0_ = *a0;       bool 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

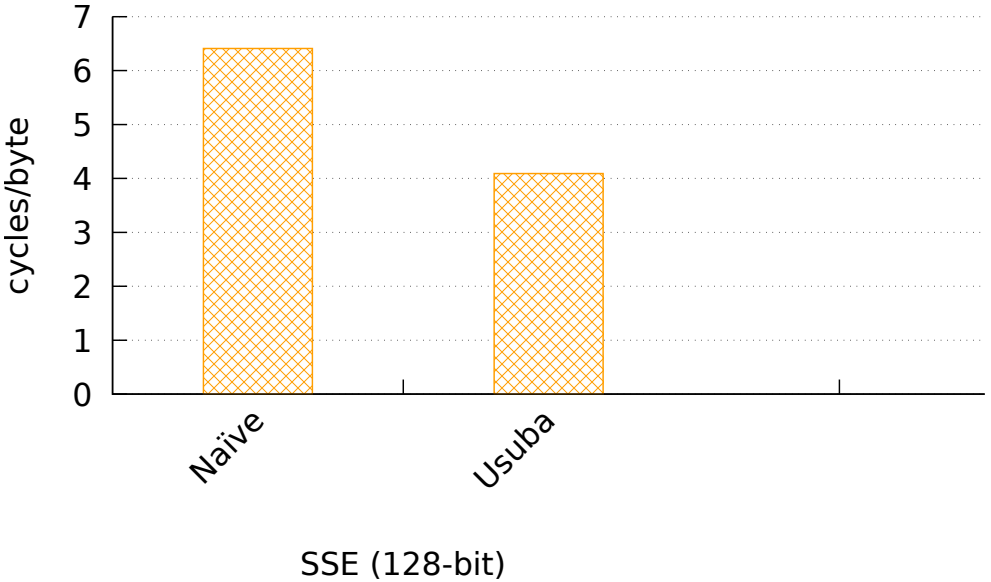
## Rectangle, our way

```
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
```

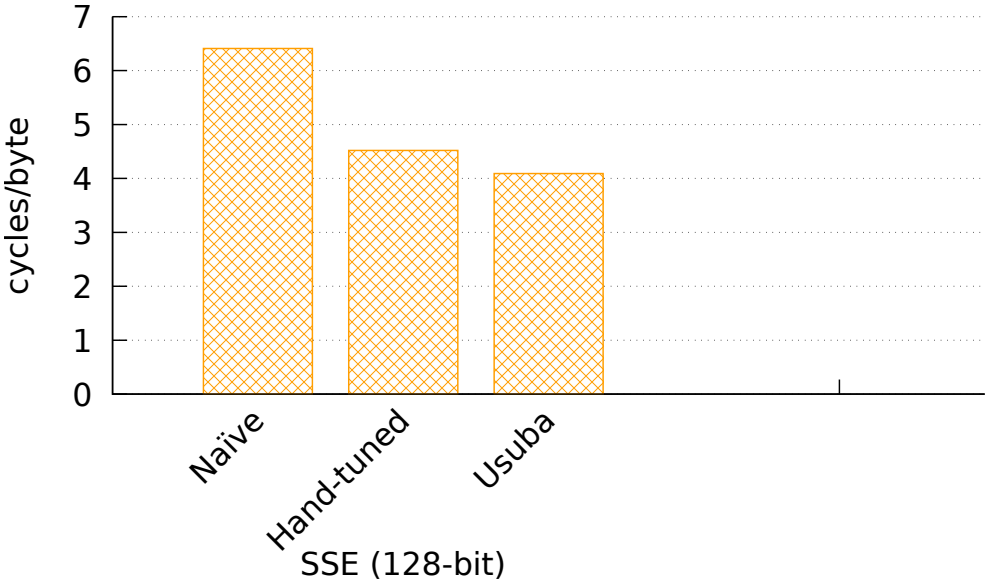
```
table SubColumn (input:v4)
       returns  (out:v4) {
    6, 5, 12, 10, 1, 14, 7, 9,
    11, 0, 3, 13, 8, 15, 4, 2
}
```

```
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
```

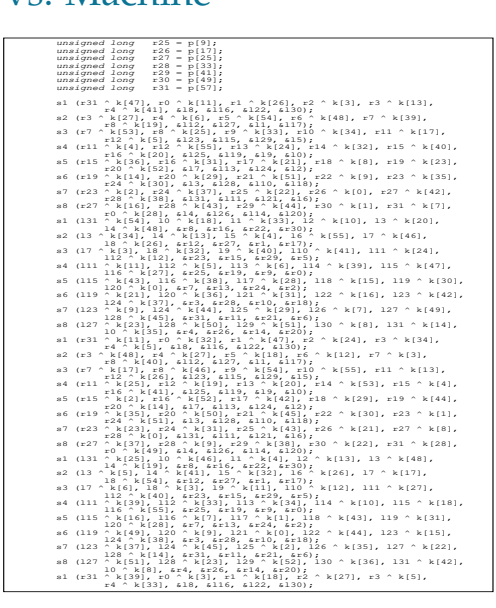
# Man vs. Machine



# Man vs. Machine



# Man vs. Machine

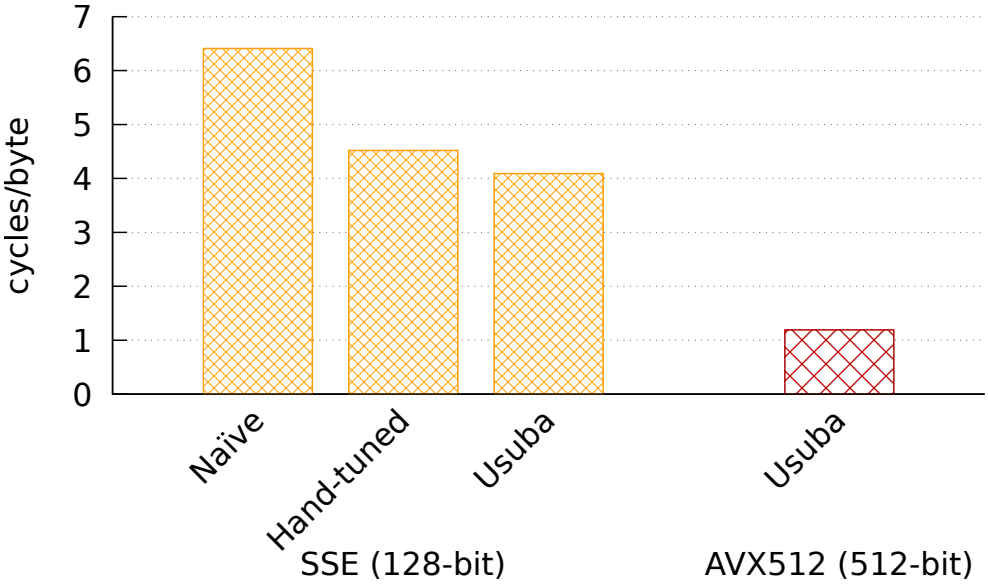


```
unsigned long   r25 = p[9];
unsigned long   r26 = p[17];
unsigned long   r27 = p[25];
unsigned long   r28 = p[33];
unsigned long   r29 = p[41];
unsigned long   r30 = p[49];
unsigned long   r31 = p[57];

s1 =r31 ^ k[47], r0 ^ k[11], r1 ^ k[26], r2 ^ k[3], r3 ^ k[13],
    r4 ^ k[41], &i8, &i16, &i22, &i30;
s2 =r3 ^ k[27], r4 ^ k[6], r5 ^ k[54], r6 ^ k[48], r7 ^ k[39],
    r8 ^ k[19], &i12, &i27, &i51, &i17;
s3 =r7 ^k[53], r8 ^ k[25], r9 ^ k[33], r10 ^ k[34], r11 ^ k[17],
    r12 ^ k[5], &i23, &i15, &i29, &i5);
s4 =r11 ^ k[4], r12 ^ k[55], r13 ^ k[24], r14 ^ k[32], r15 ^ k[40],
    r16 ^ k[20], &i25, &i19, &i9, &i10);
s5 =r15 ^k[36], r16 ^ k[31], r17 ^ k[21], r18 ^ k[8], r19 ^ k[23],
    r20 ^ k[52], &i7, &i13, &i24, &i2);
s6 =r19 ^ k[14], r20 ^ k[29], r21 ^ k[51], r22 ^ k[9], r23 ^ k[35],
    r24 ^ k[30], &i3, &i28, &i10, &i18);
s7 =r23 ^ k[2], r24 ^ k[37], r25 ^ k[22], r26 ^ k[0], r27 ^ k[42],
    r28 ^ k[38], &i31, &i11, &i26, &i6);
s8 =r27 ^ k[16], r28 ^ k[43], r29 ^ k[44], r30 ^ k[1], r31 ^ k[7],
    r0 ^ k[28], &i4, &i26, &i14, &i20);
s1 =r31 ^k[54], r0 ^ k[18], i1 ^ k[33], i2 ^ k[10], i3 ^ k[20],
    i4 ^ k[48], &r8, &r16, &r22, &r30;
s2 =i3 ^k[34], i4 ^ k[13], i5 ^ k[4], i6 ^ k[55], i7 ^ k[46],
    i8 ^ k[26], &i12, &i27, &i1, &i17);
s3 =i7 ^k[3], i8 ^ k[32], i9 ^k[40], i10 ^ k[41], i11 ^ k[24],
    i12 ^ k[12], &i23, &i15, &i29, &i5);
s4 =i11 ^k[11], i12 ^ k[5], i13 ^ k[6], i14 ^ k[39], i15 ^ k[47],
    i16 ^ k[27], &r25, &r19, &r9, &r10;
s5 =i15 ^k[43], i16 ^ k[38], i17 ^ k[28], i18 ^ k[15], i19 ^ k[30],
    i20 ^k[0], &r7, &i13, &r24, &r2;
s6 =i19 ^k[21], i20 ^ k[36], i21 ^ k[31], i22 ^ k[16], i23 ^ k[42],
    i24 ^ k[37], &r3, &r28, &r10, &r18;
s7 =i23 ^ k[9], i24 ^ k[44], i25 ^ k[22], i26 ^ k[7], i27 ^ k[49],
    i28 ^ k[45], &r31, &i11, &r21, &r6;
s8 =i27 ^k[23], i28 ^ k[50], i29 ^ k[53], i30 ^ k[8], i31 ^ k[14],
    i0 ^ 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], &i8, &i16, &i22, &i30;
s2 =r3 ^k[48], r4 ^ k[27], r5 ^ k[18], r6 ^ k[12], r7 ^ k[3],
    r8 ^ k[40], &i12, &i27, &i1, &i17);
s3 =r7 ^k[17], r8 ^ k[46], r9 ^ k[10], r10 ^ k[55], r11 ^ k[13],
    r12 ^ k[26], &i23, &i15, &i29, &i5);
s4 =r11 ^k[25], r12 ^ k[19], r13 ^ k[20], r14 ^ k[53], r15 ^ k[4],
    r16 ^ k[41], &i25, &i19, &i9, &i10);
s5 =r15 ^k[2], r16 ^ k[52], r17 ^ k[42], r18 ^ k[29], r19 ^ k[44],
    r20 ^k[14], &i7, &i13, &i24, &i2);
s6 =r19 ^k[35], r20 ^ k[50], r21 ^ k[45], r22 ^ k[30], r23 ^ k[1],
    r24 ^ k[51], &i3, &i28, &i10, &i18);
s7 =r23 ^k[23], r24 ^k[31], r25 ^ k[43], r26 ^ k[21], r27 ^ k[8],
    r28 ^ k[0], &i31, &i11, &i26, &i6);
s8 =r27 ^ k[37], r28 ^ k[9], r29 ^ k[38], r30 ^ k[22], r31 ^ k[28],
    r0 ^ k[49], &i4, &i26, &i14, &i20);
s1 =i31 ^k[25], i0 ^ k[46], i1 ^ k[4], i2 ^ k[13], i3 ^ k[48],
    i4 ^k[30], &r8, &r16, &r22, &r30;
s2 =i3 ^ k[5], i4 ^ k[41], i5 ^ k[32], i6 ^ k[26], i7 ^ k[17],
    i8 ^ k[6], &i8, &i3, &i9 ^k[11], i10 ^ k[12], i11 ^ k[27],
    i12 ^ k[40], &i23, &r15, &r29, &r5);
s4 =i11 ^k[39], i12 ^k[33], i13 ^ k[34], i14 ^ k[10], i15 ^ k[18],
    i16 ^ k[55], &r25, &r19, &r9, &r10;
s5 =i15 ^k[16], i16 ^ k[7], i17 ^ k[11], i18 ^ k[43], i19 ^ k[31],
    i20 ^k[28], &r7, &r13, &r24, &r2;
s6 =i19 ^k[49], i20 ^k[19], i21 ^ k[0], i22 ^ k[44], i23 ^ k[15],
    i24 ^k[38], &r3, &r28, &r10, &r18;
s7 =i23 ^ k[37], i24 ^ k[45], i25 ^ k[2], i26 ^ k[35], i27 ^ k[22],
    i28 ^ k[14], &r31, &i11, &r21, &r6;
s8 =i27 ^k[51], i28 ^ k[23], i29 ^ k[53], i30 ^ k[36], i31 ^ k[42],
    i0 ^ 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], &i8, &i16, &i22, &i30;
```

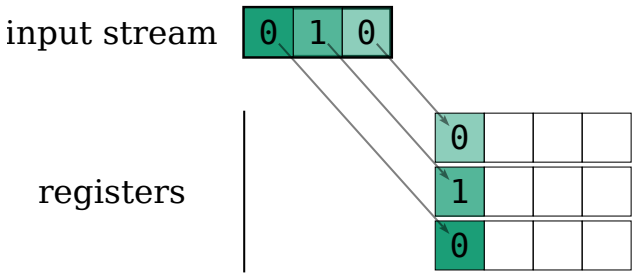
```
s2 =r3 ^ k[19], r4 ^ k[55], r5 ^ k[46], r6 ^ k[40], r7 ^ k[6],
    r8 ^ k[11], &i12, &i27, &i1, &i17);
s3 =r7 ^ k[20], r8 ^ k[17], r9 ^ k[25], r10 ^ k[26], r11 ^ k[41],
    r12 ^ k[54], &i23, &i15, &i29, &i5);
s4 =r11 ^k[53], r12 ^ k[47], r13 ^ k[48], r14 ^ k[24], r15 ^ k[32],
    r16 ^ k[12], &i25, &i19, &i9, &i10);
s5 =r15 ^ k[30], r16 ^ k[21], r17 ^ k[15], r18 ^ k[2], r19 ^ k[45],
    r20 ^ k[42], &i7, &i13, &i24, &i2);
s6 =r19 ^ k[8], r20 ^ k[23], r21 ^ k[14], r22 ^ k[31], r23 ^ k[29],
    r24 ^k[52], &i3, &i28, &i10, &i18);
s7 =r23 ^k[50], r24 ^ k[10], r25 ^ k[36], r26 ^ k[49], r27 ^ k[36],
    r28 ^ k[28], &i31, &i11, &i26, &i6);
s8 =r27 ^k[34], r28 ^k[35], r29 ^ k[33], r30 ^ k[50], r31 ^ k[1],
    r0 ^ k[22], &i4, &i26, &i14, &i20);
s1 =i31 ^k[53], i0 ^ k[17], i1 ^ k[32], i2 ^k[41], i3 ^ k[19],
    i4 ^k[47], &r8, &r16, &r22, &r30;
s2 =i3 ^k[33], i4 ^ k[12], i5 ^ k[3], i6 ^ k[54], i7 ^ k[20],
    i8 ^ k[26], &i12, &i27, &i1, &i17);
s3 =i7 ^k[34], i8 ^ k[6], i9 ^k[39], i10 ^k[40], i11 ^ k[55],
    i12 ^ k[4], &i23, &i15, &i29, &i5);
s4 =i11 ^ k[10], i12 ^k[4], i13 ^ k[5], i14 ^ k[13], i15 ^ k[46],
    i16 ^ k[24], &r25, &r19, &r9, &r10;
s5 =i15 ^ k[44], i16 ^k[35], i17 ^ k[29], i18 ^ k[16], i19 ^ k[0],
    i20 ^k[31], &r7, &i13, &r24, &r2;
s6 =i19 ^ k[22], i20 ^ k[37], i21 ^ k[28], i22 ^ k[45], i23 ^ k[43],
    i24 ^ k[7], &r3, &r28, &r10, &r18;
s7 =i23 ^k[38], i24 ^ k[14], i25 ^ k[30], i26 ^ k[8], i27 ^ k[50],
    i28 ^ k[52], &r31, &i11, &r21, &r6;
s8 =i27 ^k[52], i28 ^ k[51], i29 ^ k[21], i30 ^ k[9], i31 ^ k[15],
    i0 ^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], &i8, &i16, &i22, &i30;
s2 =r3 ^k[47], r4 ^ k[28], r5 ^ k[17], r6 ^ k[11], r7 ^ k[34],
    r8 ^k[39], &i12, &i27, &i1, &i17);
s3 =r7 ^k[48], r8 ^ k[29] ^k[53], r9 ^k[54], r10 ^ k[12],
    r11 ^ k[18], &i23, &i15, &i29, &i5);
s4 =r11 ^ k[24], r12 ^ k[41], r13 ^ k[19], r14 ^ k[27], r15 ^ k[3],
    r16 ^ k[40], &i25, &i19, &i9, &i10);
s5 =r15 ^ k[14], r16 ^ k[49], r17 ^ k[43], r18 ^k[30], r19 ^k[14],
    r20 ^ k[15], &i7, &i13, &i24, &i2);
s6 =r19 ^ k[36], r20 ^ k[51], r21 ^ k[42], r22 ^ k[0], r23 ^ k[21],
    r24 ^ k[21], &i3, &i28, &i10, &i18);
s7 =r23 ^k[23], r24 ^k[31], r25 ^ k[44], r26 ^k[22], r27 ^ k[9],
    r28 ^ k[1], &i31, &i11, &i21, &i6);
s8 =r27 ^k[7], r28 ^k[38], r29 ^ k[30] ^k[23], r31 ^k[29],
    r0 ^k[50], &i4, &i26, &i14, &i20);
s1 =i31 ^k[22], i0 ^ k[20], i1 ^ k[3], i2 ^ k[12], i3 ^ k[47],
    i4 ^ k[18], &r8, &r16, &r22, &r30;
s2 =i3 ^k[4], i4 ^k[40], i5 ^ k[6], i6 ^k[25], i7 ^k[48],
    i8 ^k[31], &i12, &i27, &i1, &i17);
s3 =i7 ^k[5], i8 ^k[34], i9 ^k[10], i10 ^ k[11], i11 ^ k[26],
    i12 ^ k[39], &i23, &i15, &i29, &i5);
s4 =i11 ^ k[13], i12 ^k[32], i13 ^ k[33], i14 ^ k[41], i15 ^ k[17],
    i16 ^ k[54], &r25, &r19, &r9, &r10;
s5 =i15 ^ k[45], i16 ^ k[8], i17 ^ k[2], i18 ^ k[44], i19 ^k[28],
    i20 ^k[53], &r7, &i13, &r24, &r2;
s6 =i19 ^k[50], i20 ^k[38], i21 ^ k[1], i22 ^ k[14], i23 ^ k[16],
    i24 ^ k[35], &r3, &r28, &r10, &r18;
s7 =i23 ^ k[7], i24 ^k[42], i25 ^ k[31], i26 ^ k[36], i27 ^ k[23],
    i28 ^ k[37], &r31, &i11, &r21, &r6;
s8 =i27 ^ k[21], i28 ^k[52], i29 ^k[49], i30 ^ k[37], i31 ^ k[43],
    i0 ^ k[9], &r4, &r26, &r14, &r20;
s1 =r31 ^k[6], r0 ^k[28], r1 ^k[10], r2 ^ k[19], r3 ^ k[54],
    r4 ^k[25], &i8, &i16, &i22, &i30;
s2 =r3 ^k[11], r4 ^k[47], r5 ^k[13], r6 ^k[32], r7 ^k[55],
    r8 ^k[3], &i12, &i27, &i1, &i17);
s3 =r7 ^k[12], r8 ^k[41], r9 ^k[17], r10 ^k[18], r11 ^k[33],
    r12 ^k[46], &i23, &i15, &i29, &i5);
s4 =i11 ^k[20], r12 ^k[39], r13 ^k[40], r14 ^k[48], r15 ^k[24],
    r16 ^k[4], &i25, &i19, &i9, &i10);
s5 =i15 ^k[27], r16 ^k[37], r17 ^k[9], r18 ^k[51], r19 ^k[35],
    r20 ^k[36], &i7, &i13, &i24, &i2);
s6 =i19 ^k[24], r20 ^k[21], r21 ^k[8], r22 ^k[21], r23 ^k[23],
    r24 ^k[42], &i3, &i28, &i10, &i18);
```

# Man vs. Machine



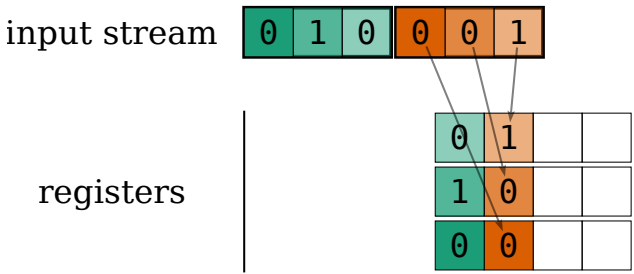
# Bitslicing

## High-throughput software circuits



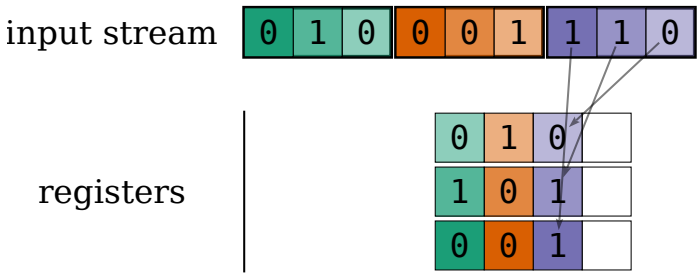
# Bitslicing

## High-throughput software circuits



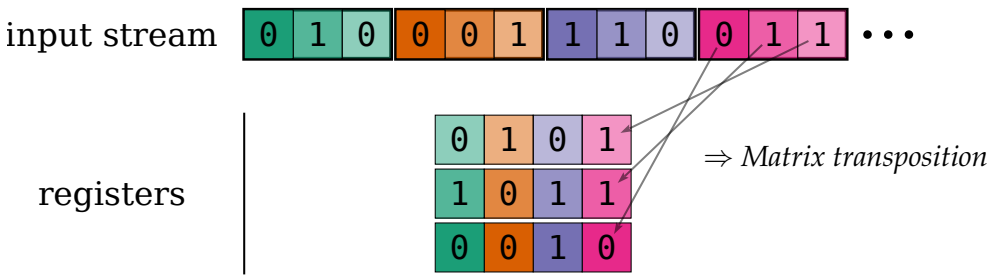
# Bitslicing

## High-throughput software circuits



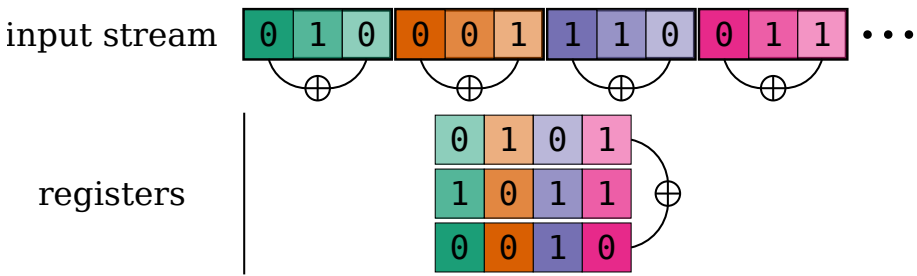
# Bitslicing

## High-throughput software circuits



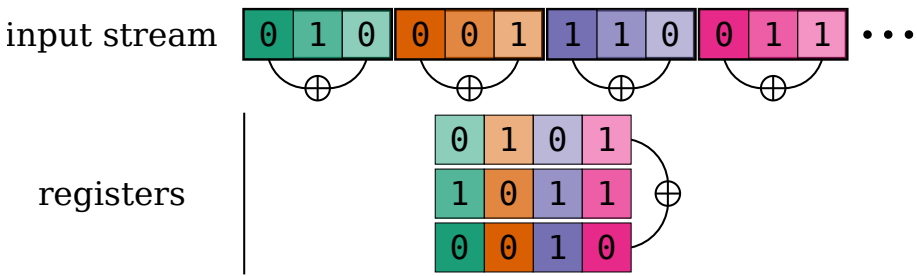
# Bitslicing

## High-throughput software circuits



# Bitslicing

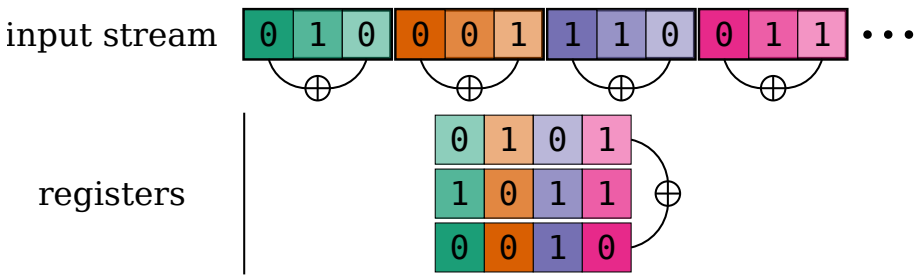
## High-throughput software circuits




*Wider registers ⇒ More parallelism*
*(SSE, AVX, AVX2, AVX-512, . . . )*

# Bitslicing

## High-throughput software circuits

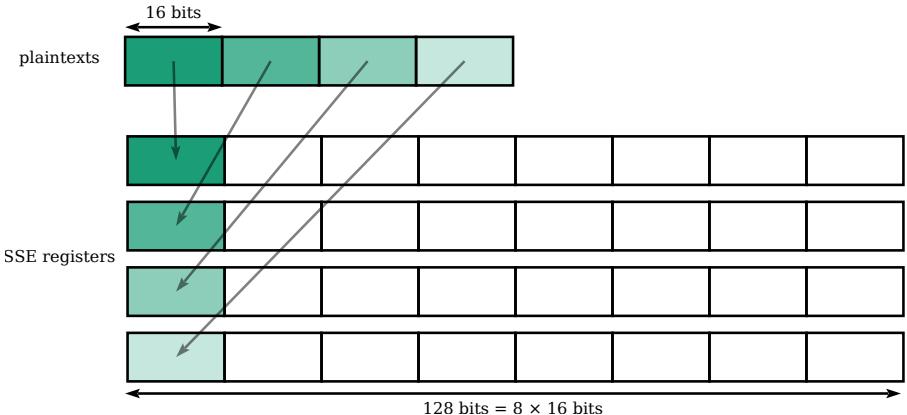



*Wider registers ⇒ More parallelism*
*(SSE, AVX, AVX2, AVX-512, . . . )*
*Register pressure?*

# V-slicing

## ShiftRows in Vertical mode

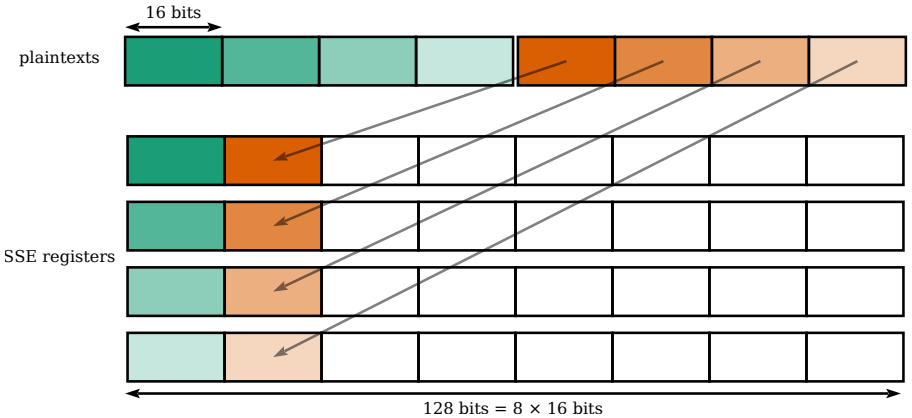
```
node ShiftRows (input:u16x4) : (out:u16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



# V-slicing

## ShiftRows in Vertical mode

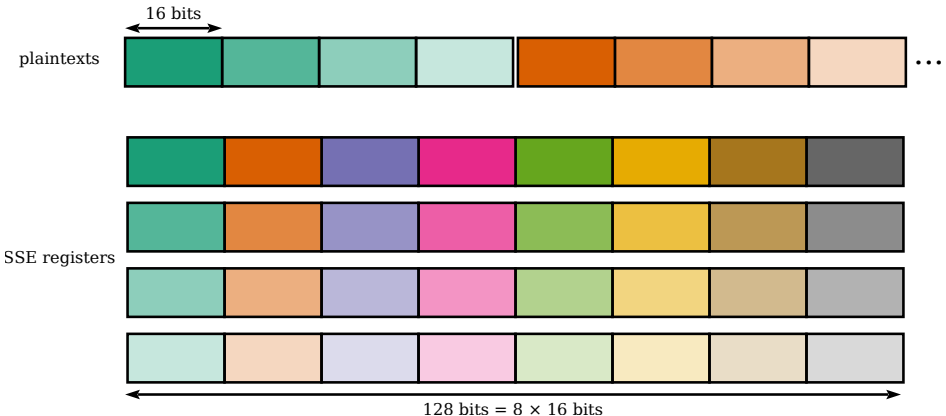
```
node ShiftRows (input:u16x4) : (out:u16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



# V-slicing

## ShiftRows in Vertical mode

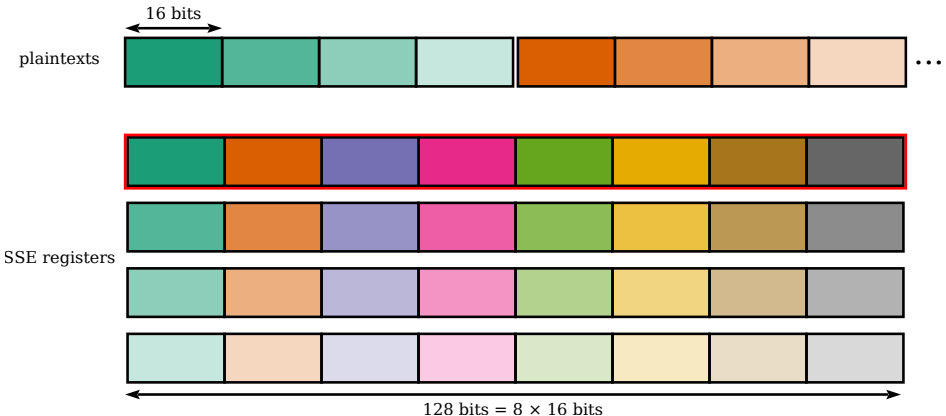
```
node ShiftRows (input:u16x4) : (out:u16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



# V-slicing

## ShiftRows in Vertical mode

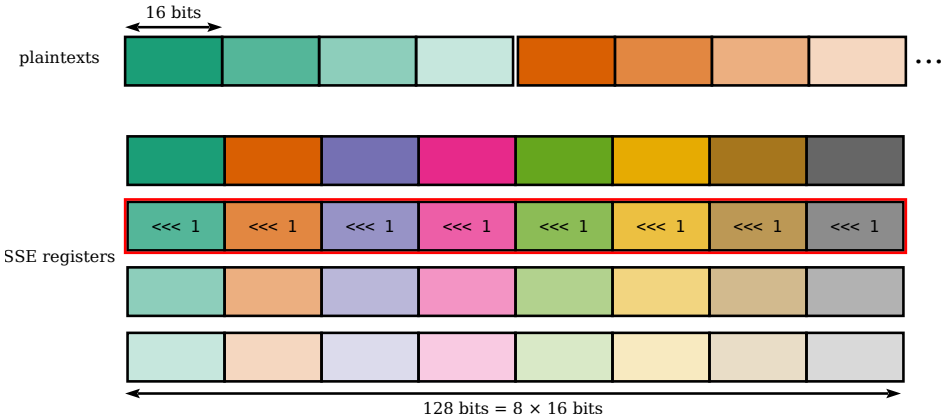
```
node ShiftRows (input:u16x4) : (out:u16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



# V-slicing

## ShiftRows in Vertical mode

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

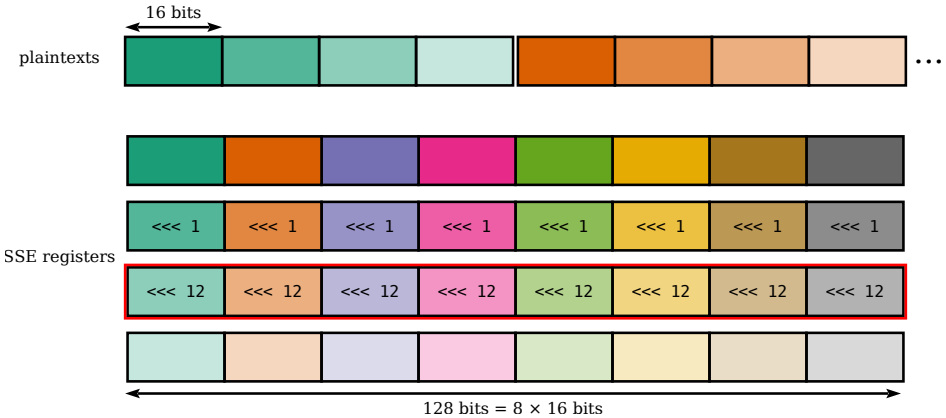



`_mm_slli_epi16, _mm_or_si128, __mm_srli_epi16`

# V-slicing

## ShiftRows in Vertical mode

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

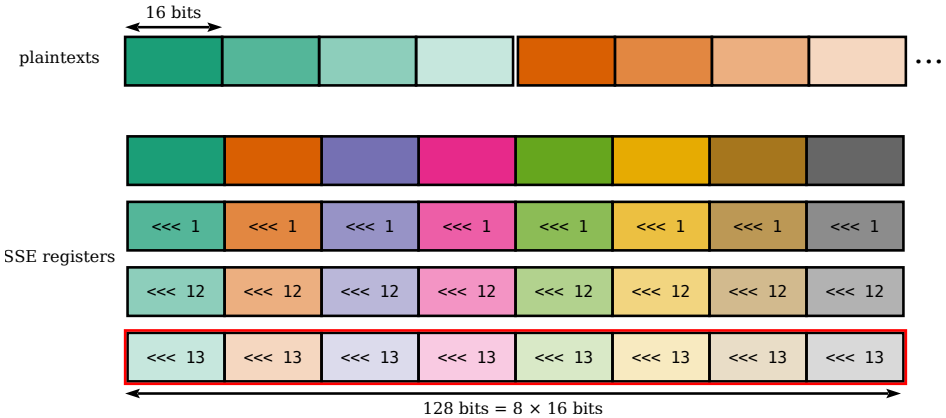



`_mm_slli_epi16, _mm_or_si128, __mm_srli_epi16`

# V-slicing

## ShiftRows in Vertical mode

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

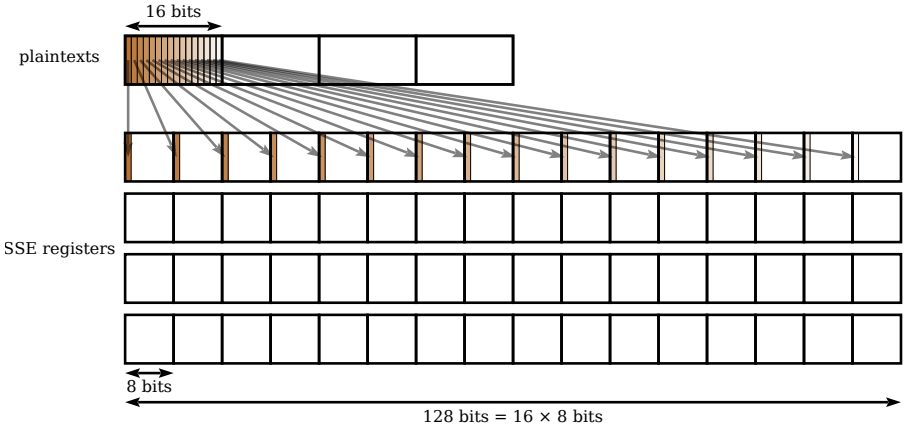



`_mm_slli_epi16, _mm_or_si128, __mm_srli_epi16`

# H-slicing

## ShiftRows in Horizontal mode

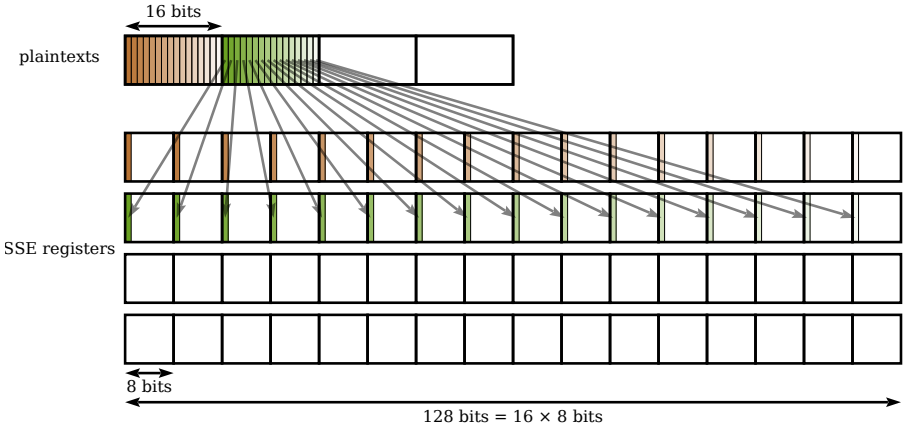
```
node ShiftRows (input:u16x4) : (out:u16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



# H-slicing

## ShiftRows in Horizontal mode

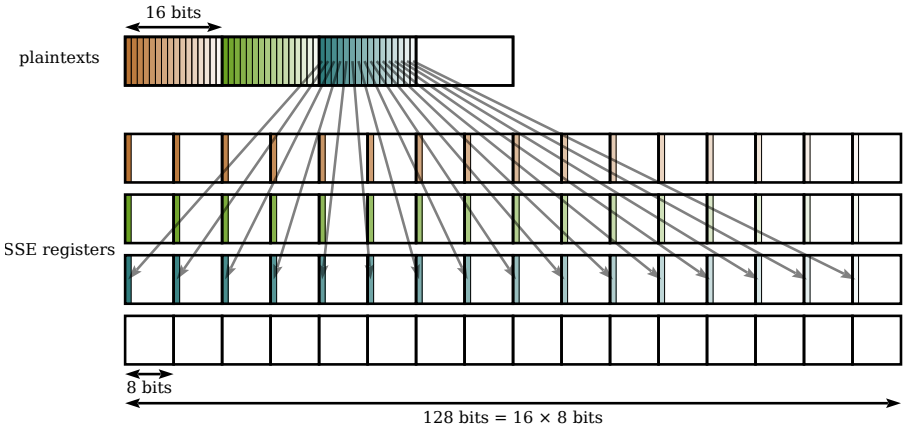
```
node ShiftRows (input:u16x4) : (out:u16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



# H-slicing

## ShiftRows in Horizontal mode

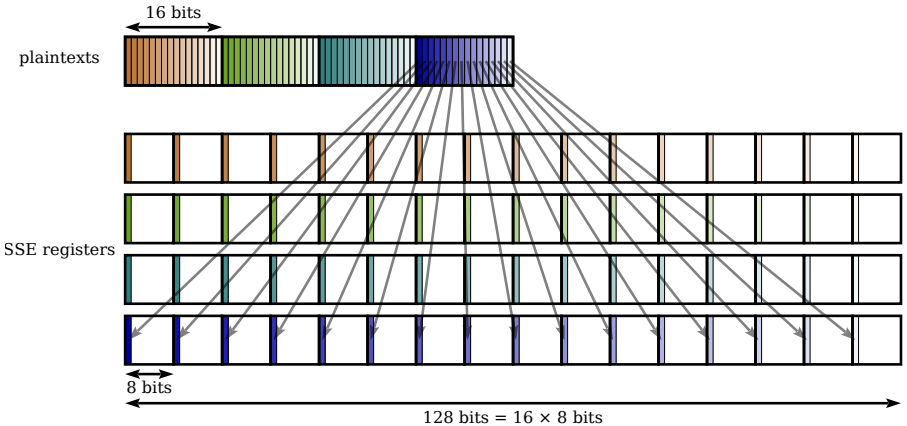
```
node ShiftRows (input:u16x4) : (out:u16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



# H-slicing

## ShiftRows in Horizontal mode

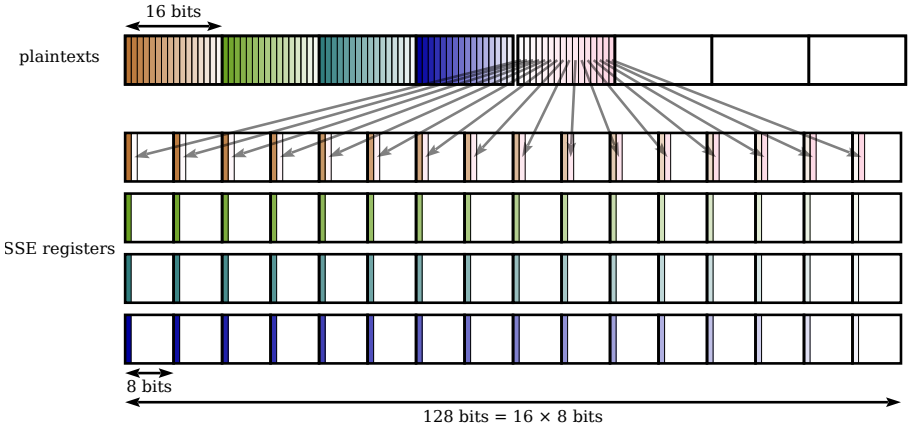
```
node ShiftRows (input:u16x4) : (out:u16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



# H-slicing

## ShiftRows in Horizontal mode

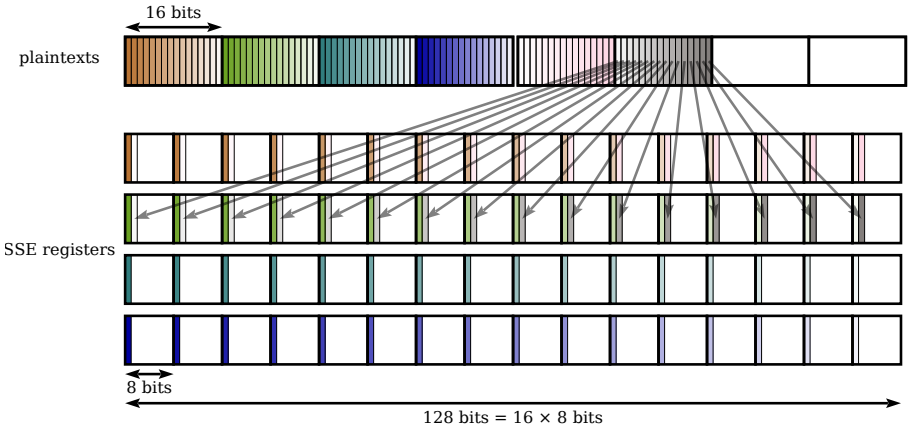
```
node ShiftRows (input:u16x4) : (out:u16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



# H-slicing

## ShiftRows in Horizontal mode

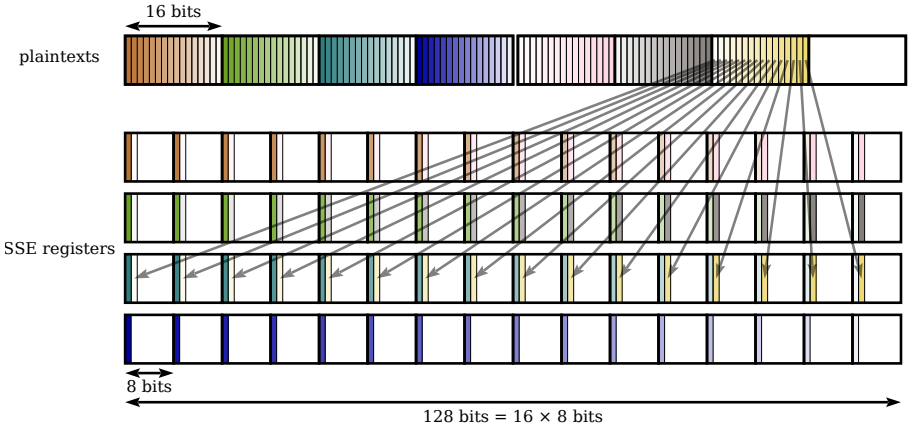
```
node ShiftRows (input:u16x4) : (out:u16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



# H-slicing

## ShiftRows in Horizontal mode

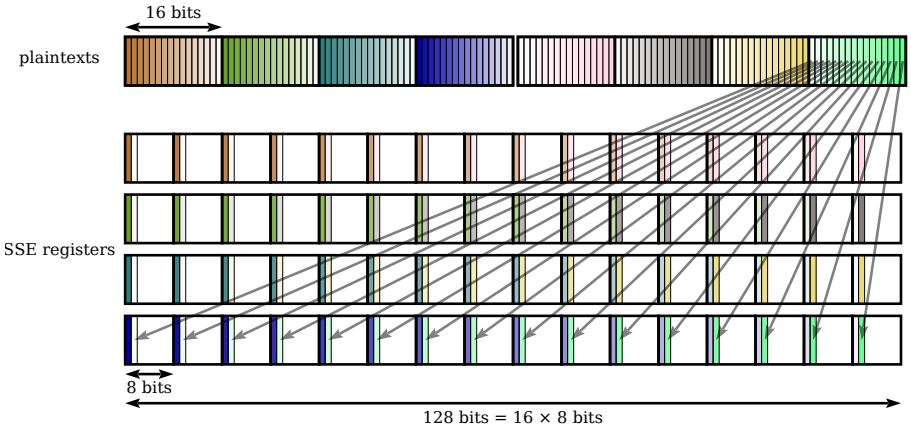
```
node ShiftRows (input:u16x4) : (out:u16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



# H-slicing

## ShiftRows in Horizontal mode

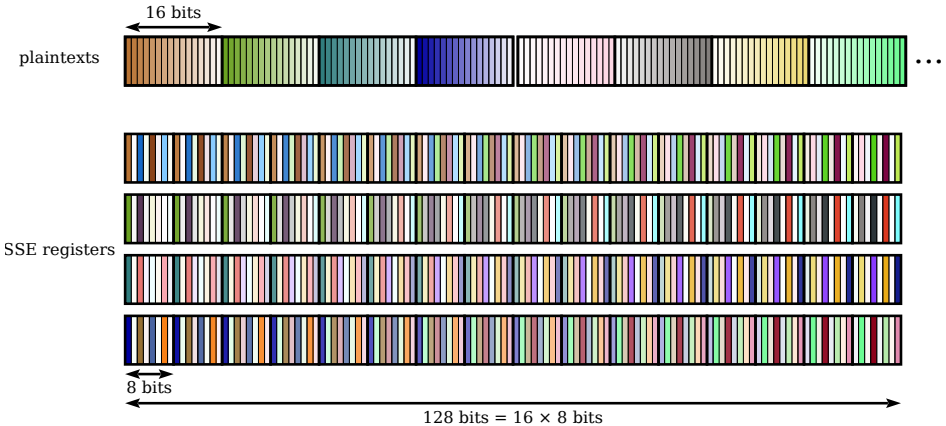
```
node ShiftRows (input:u16x4) : (out:u16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



# H-slicing

## ShiftRows in Horizontal mode

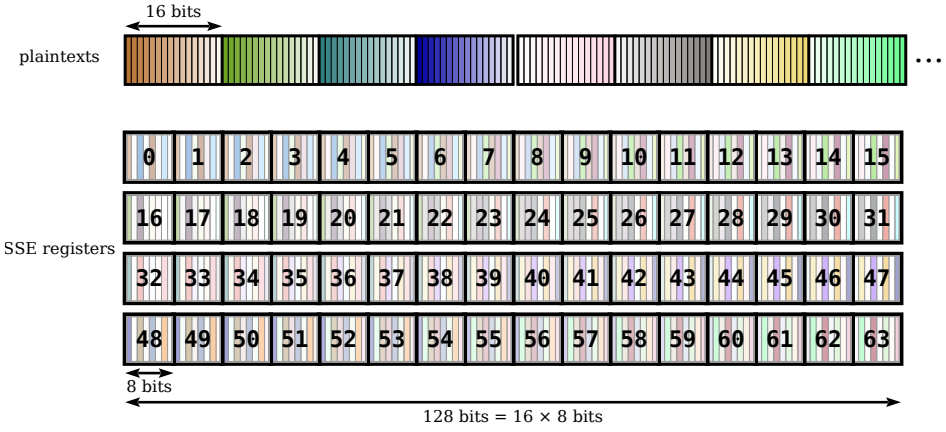
```
node ShiftRows (input:u16x4) : (out:u16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



# H-slicing

## ShiftRows in Horizontal mode

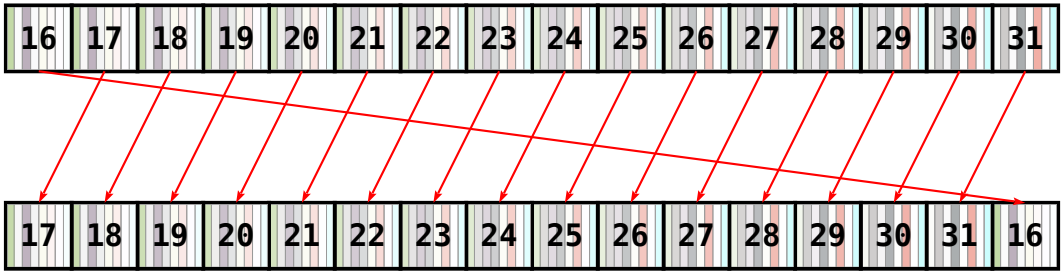
```
node ShiftRows (input:u16x4) : (out:u16x4)
let    out[0] = input[0];
       out[1] = input[1] <<< 1;
       out[2] = input[2] <<< 12;
       out[3] = input[3] <<< 13;        tel
```



# H-slicing

## ShiftRows in Horizontal mode

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

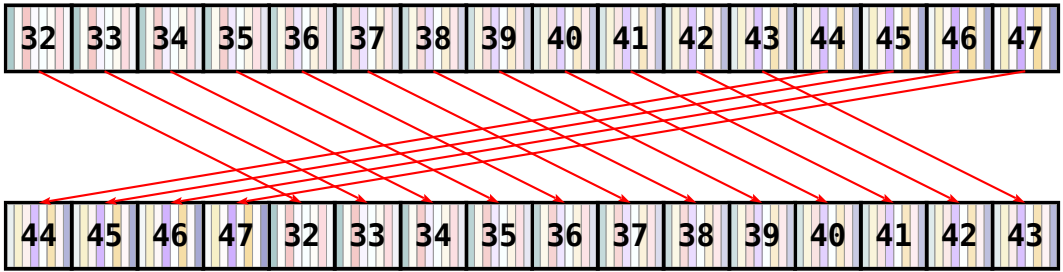



```
__m128i _mm_shuffle_epi8 (__m128i a, __m128i b)
```

# H-slicing

## ShiftRows in Horizontal mode

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

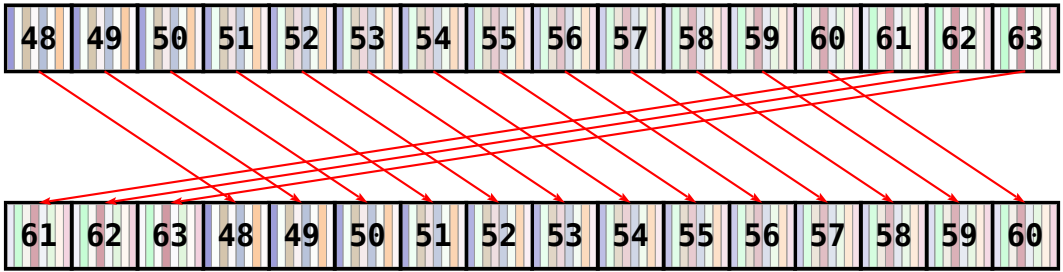



```
__m128i _mm_shuffle_epi8 (__m128i a, __m128i b)
```

# H-slicing

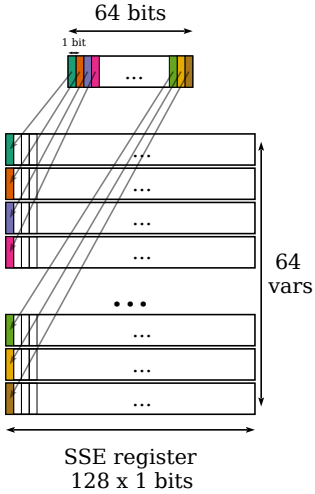
## ShiftRows in Horizontal mode

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

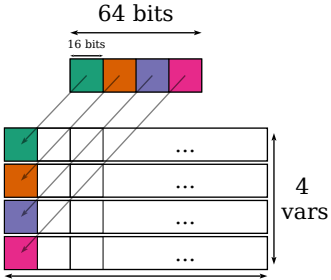
# Parallelization strategies




bitslicing

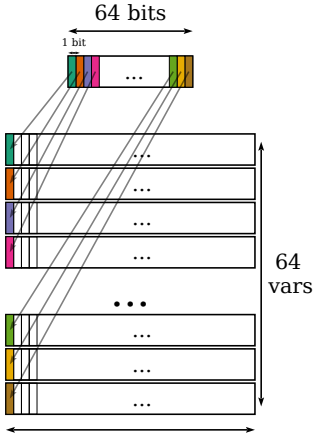
b64

# Parallelization strategies




vslicing

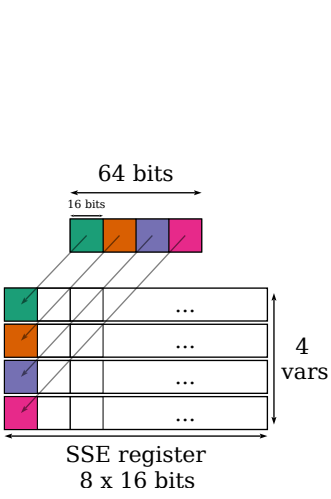
u$_V$16x4




bitslicing

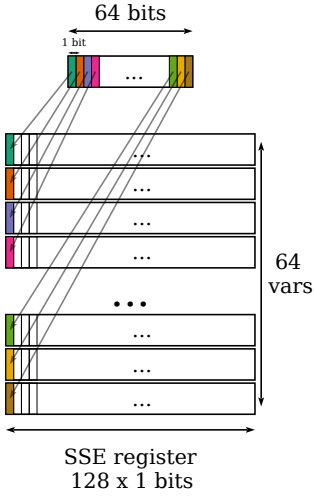
b64

# Parallelization strategies



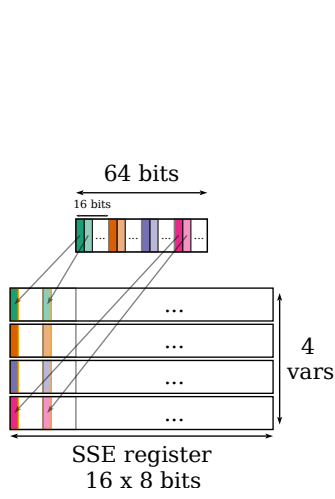

vslicing

$u_V16x4$




bitslicing

b64




hslicing

$u_H16x4$

# Quick Peek at the Language

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

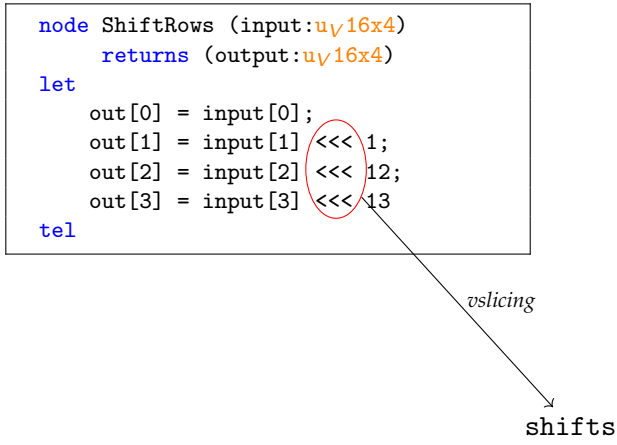
# Quick Peek at the Language

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

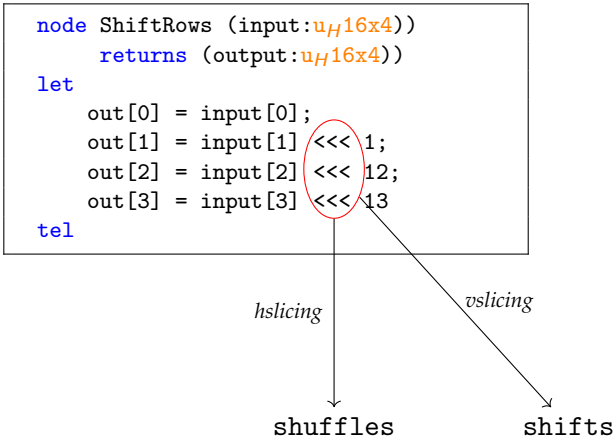
# Quick Peek at the Language

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

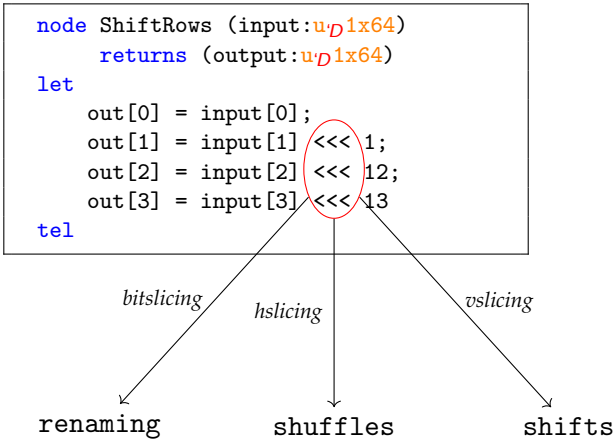
# Quick Peek at the Language



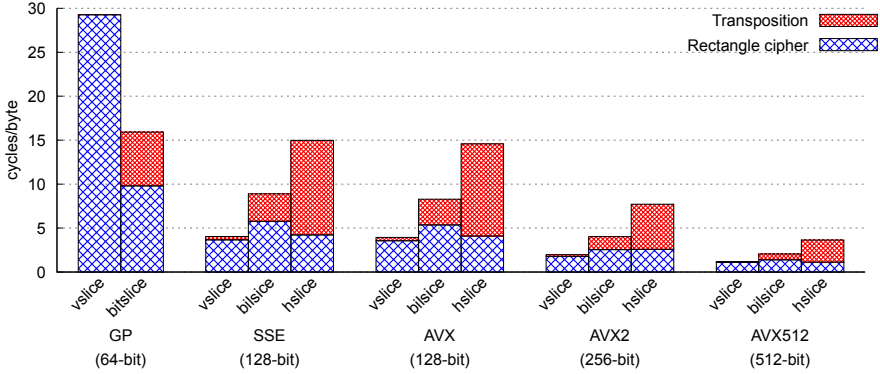
# Quick Peek at the Language



# Quick Peek at the Language



# Monomorphization



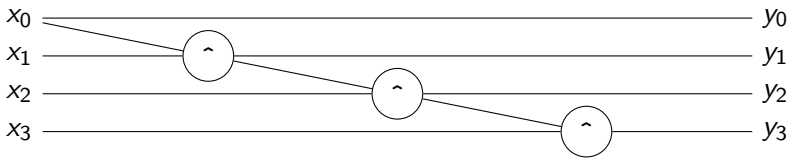

```
node Rectangle (plain : u16x4, key : u16x4[26]) returns (cipher : u16x4)
```

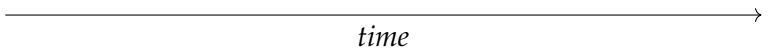
```
void RectangleV (__m256i plain[4],  __m256i key[26][4], __m256i cipher[4])
void RectangleB (__m128i plain[64], __m128i key[26][64], __m128i cipher[64])
```

# Interleaving

## Exploiting superscalar CPUs

```
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
```

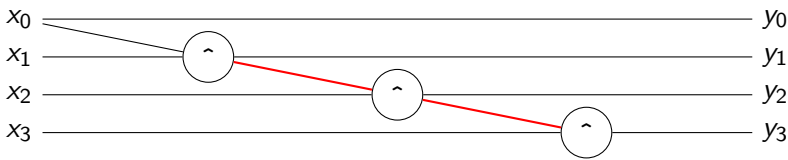


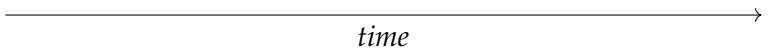



# Interleaving

## Exploiting superscalar CPUs

```
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
```

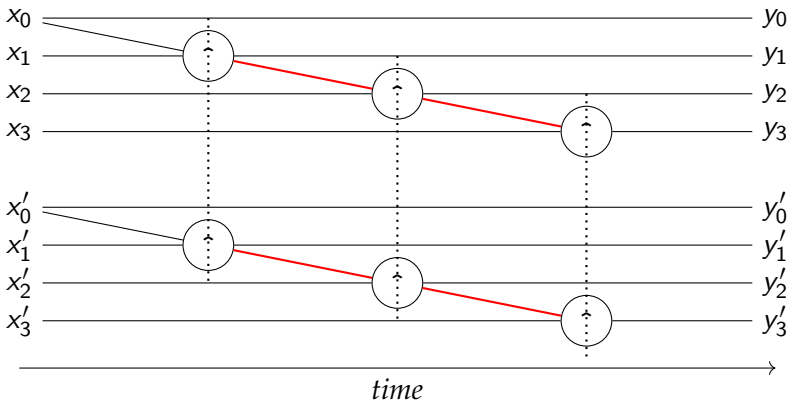


# Interleaving

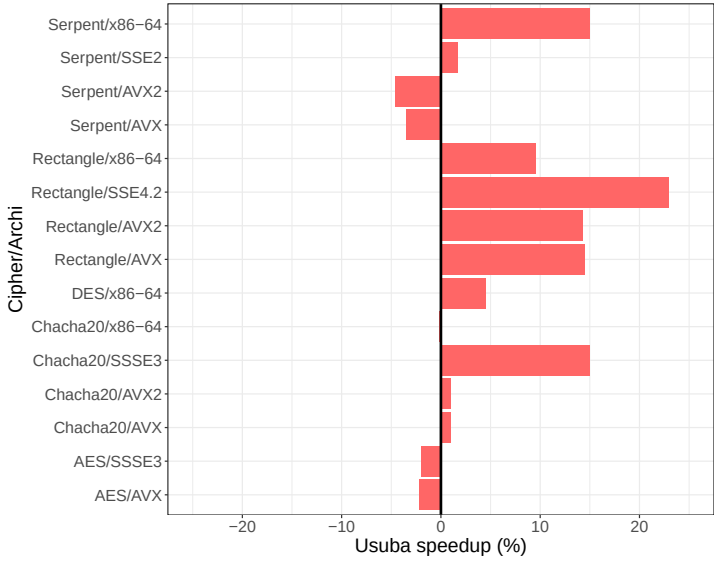
## Exploiting superscalar CPUs

```
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
```

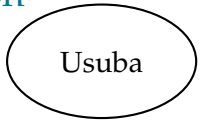


# Evaluation

## Usuba *vs.* Reference



# Conclusion

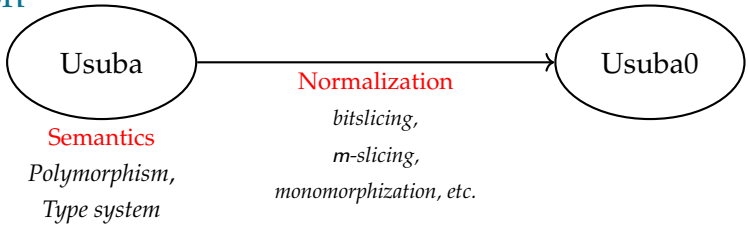



Semantics

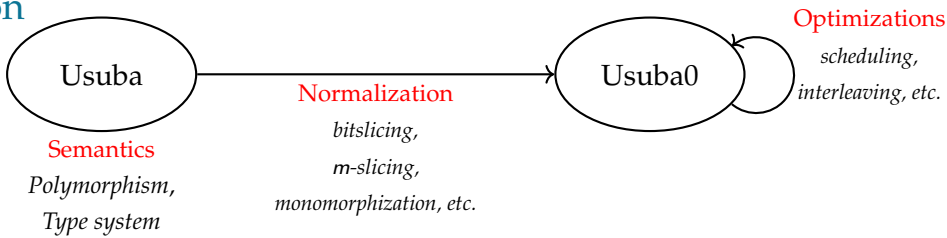
*Polymorphism,*

*Type system*

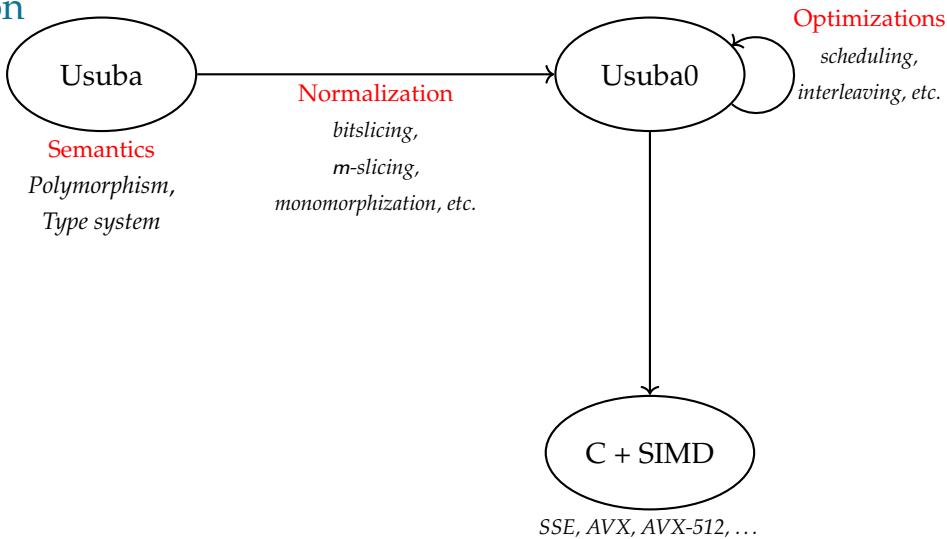
# Conclusion


Usuba
Usuba0
Normalization
bitslicing,
m-slicing,
monomorphization, etc.
Semantics
Polymorphism,
Type system

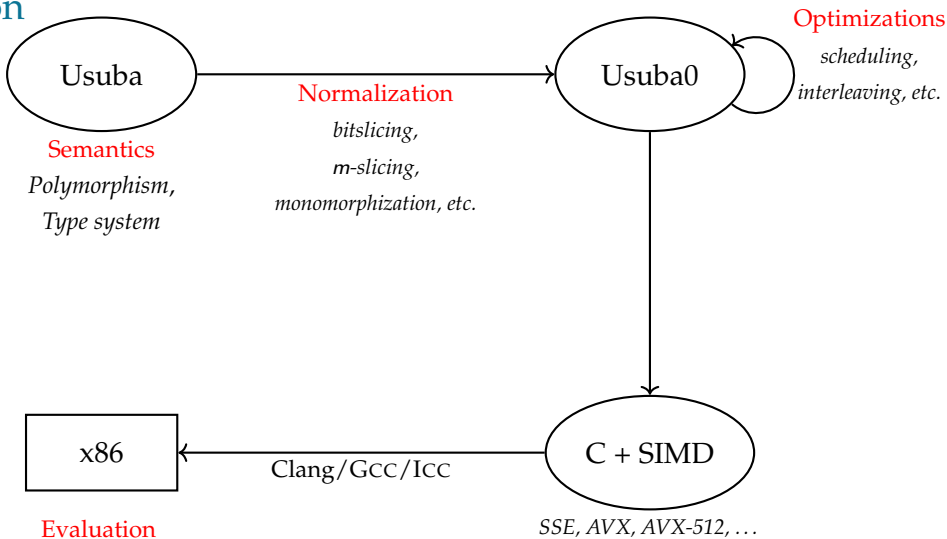
# Conclusion


Usuba
Usuba0
Optimizations
scheduling,
interleaving, etc.
Normalization
bitslicing,
m-slicing,
monomorphization, etc.
Semantics
Polymorphism,
Type system

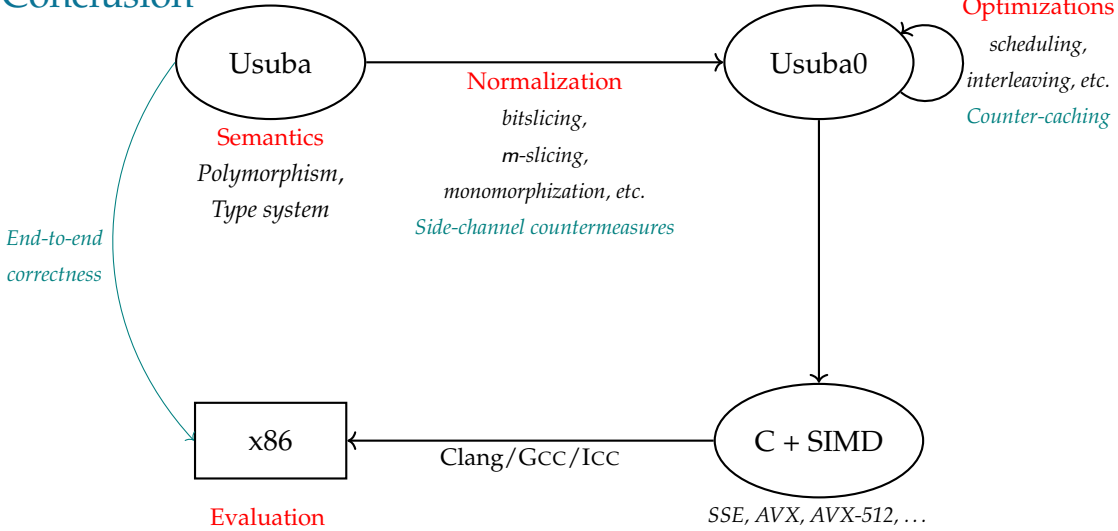
# Conclusion


Usuba
Semantics
Polymorphism,
Type system
Normalization
bitslicing,
m-slicing,
monomorphization, etc.
Usuba0
Optimizations
scheduling,
interleaving, etc.
C + SIMD
SSE, AVX, AVX-512, ...

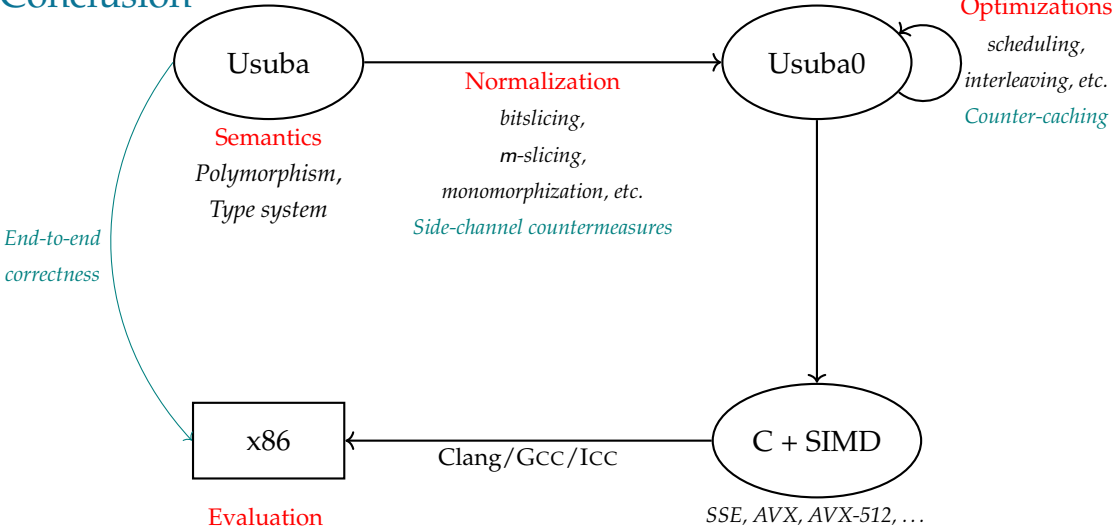
# Conclusion


Usuba
Semantics
Polymorphism,
Type system
Normalization
bitslicing,
m-slicing,
monomorphization, etc.
Usuba0
Optimizations
scheduling,
interleaving, etc.
x86
Evaluation
Clang/GCC/ICC
C + SIMD
SSE, AVX, AVX-512, ...

# Conclusion


Usuba
Usuba0
Optimizations
scheduling,
interleaving, etc.
Counter-caching
Normalization
bitslicing,
m-slicing,
monomorphization, etc.
Side-channel countermeasures
Semantics
Polymorphism,
Type system
End-to-end
correctness
x86
Clang/GCC/ICC
C + SIMD
SSE, AVX, AVX-512, ...
Evaluation

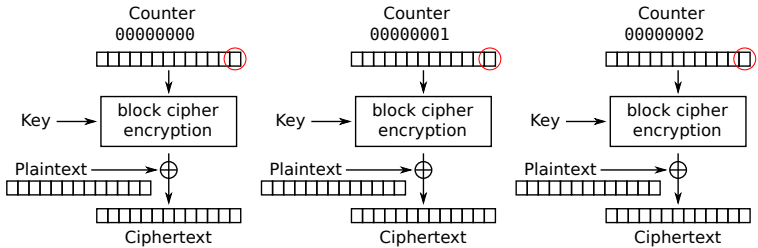
# Conclusion




`github.com/DadaIsCrazy/Usuba`

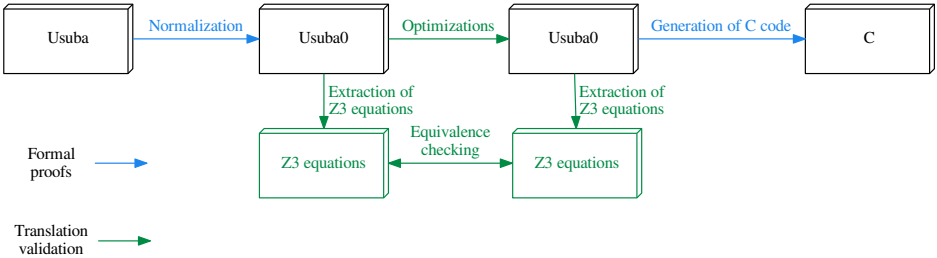
# Future work

## Optimization: counter caching



# Future work

## Verification



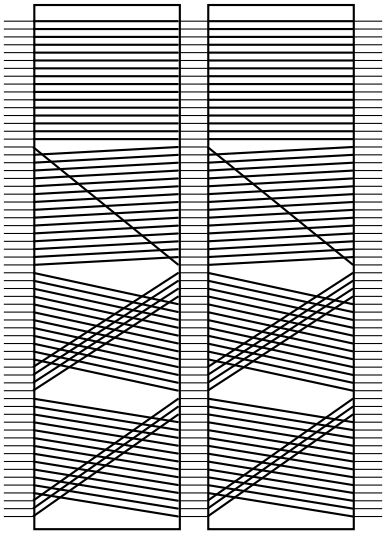

- Validate compilation
- Validate manual modifications

# Design space

| 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` | | ✓ | ✓ | |

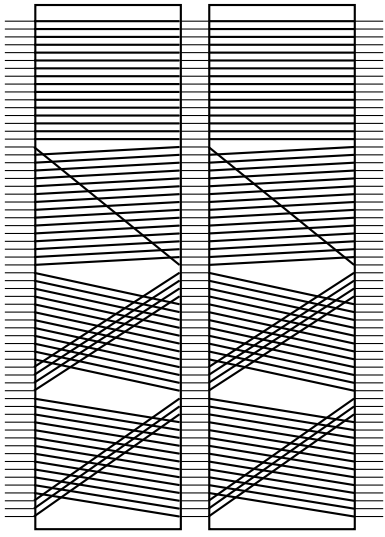
# Unrolling & Inlining

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



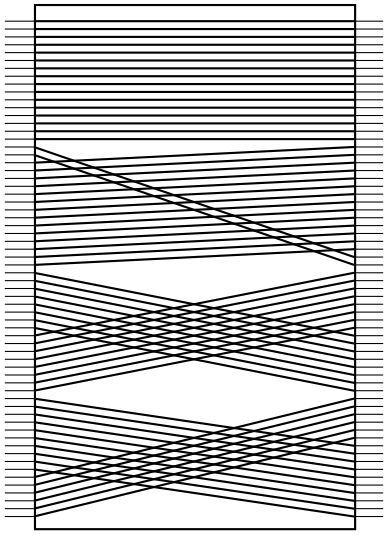
# Unrolling & Inlining

```
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
```



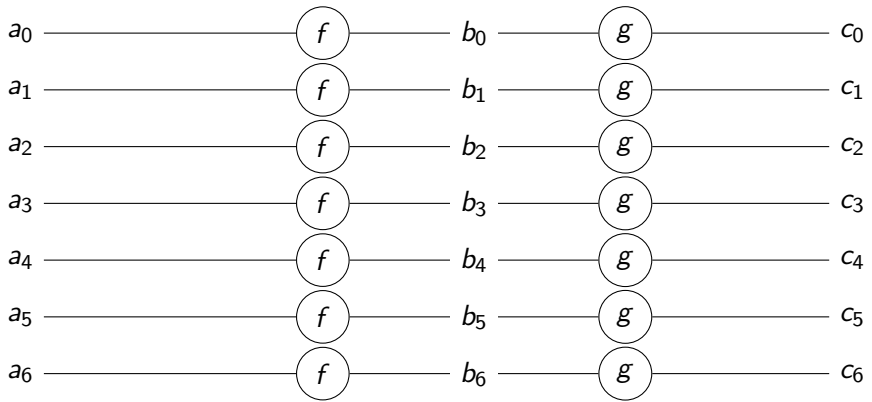
*ShiftRows (x2)*

# 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
```

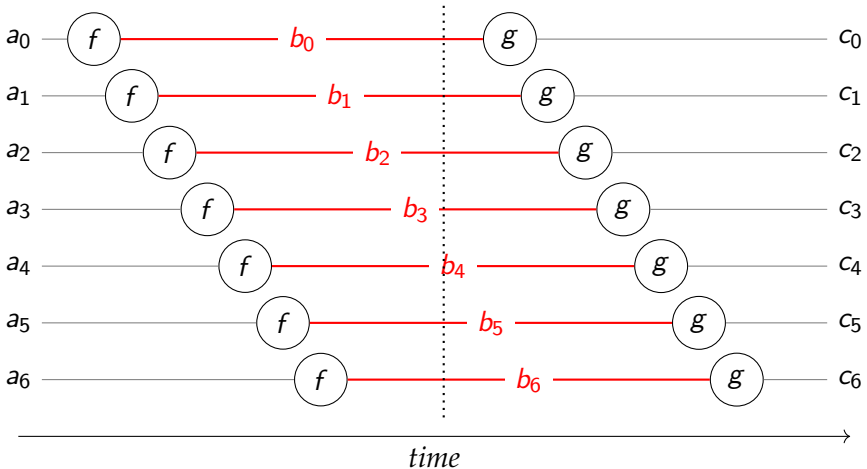
# Scheduling bitsliced code

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



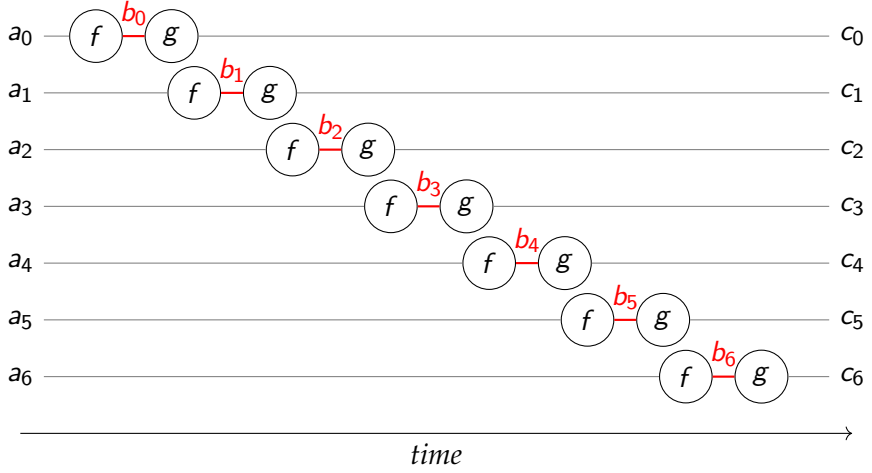
# Scheduling bitsliced code

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



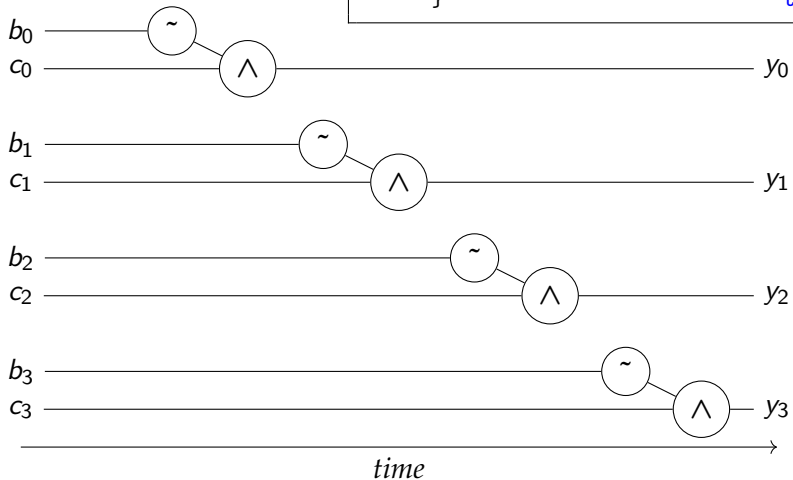
# Scheduling bitsliced code

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



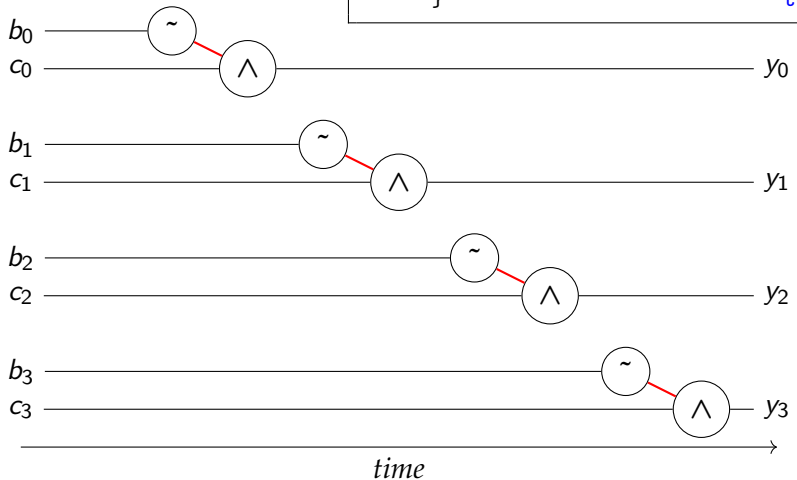
# Scheduling *m*-sliced code

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



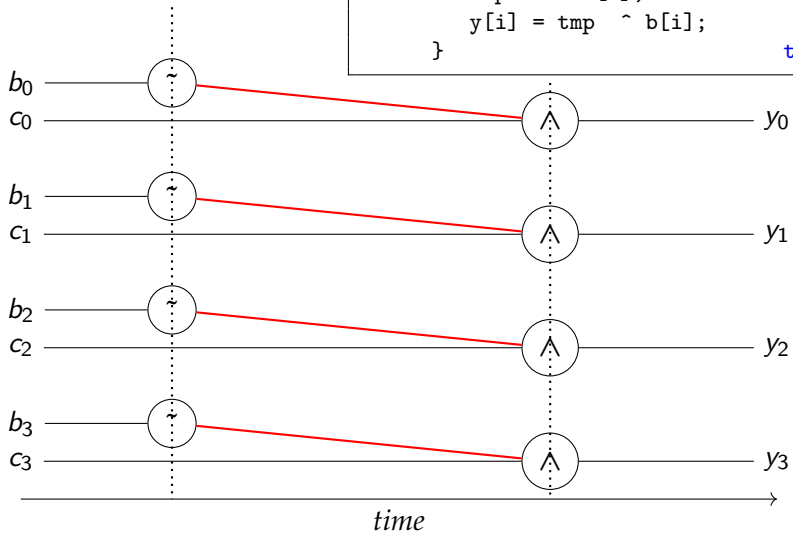
# Scheduling *m*-sliced code

```
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
```



# Evaluation

## Scalability