Loop unrolling

```
for (int i = 0; i < 8; i++) {
    a[i] = b[i] * x;
}
```

```
for (int i = 0; i < 2; i+=4) {
    a[i + 0] = b[i + 0] * x;
    a[i + 1] = b[i + 1] * x;
    a[i + 2] = b[i + 2] * x;
    a[i + 3] = b[i + 3] * x;
}
```

Loop overhead reduced

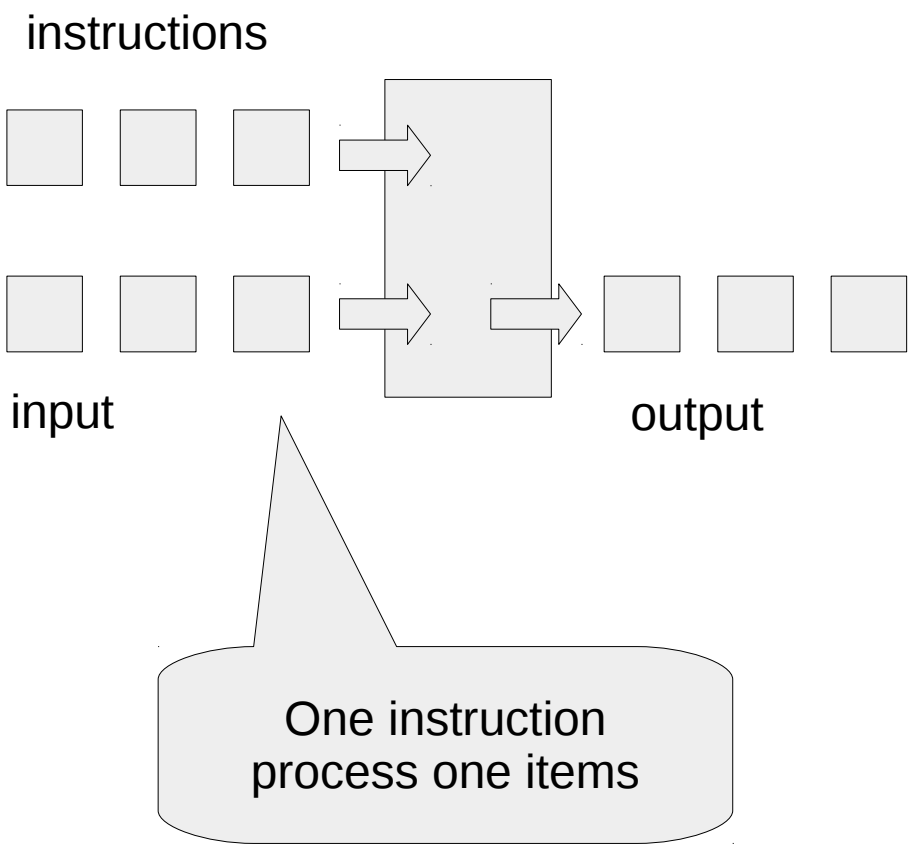
More opportunity to keep the pipeline busy

POLYTECHNIQUE MONTRÉAL

ERICSSON

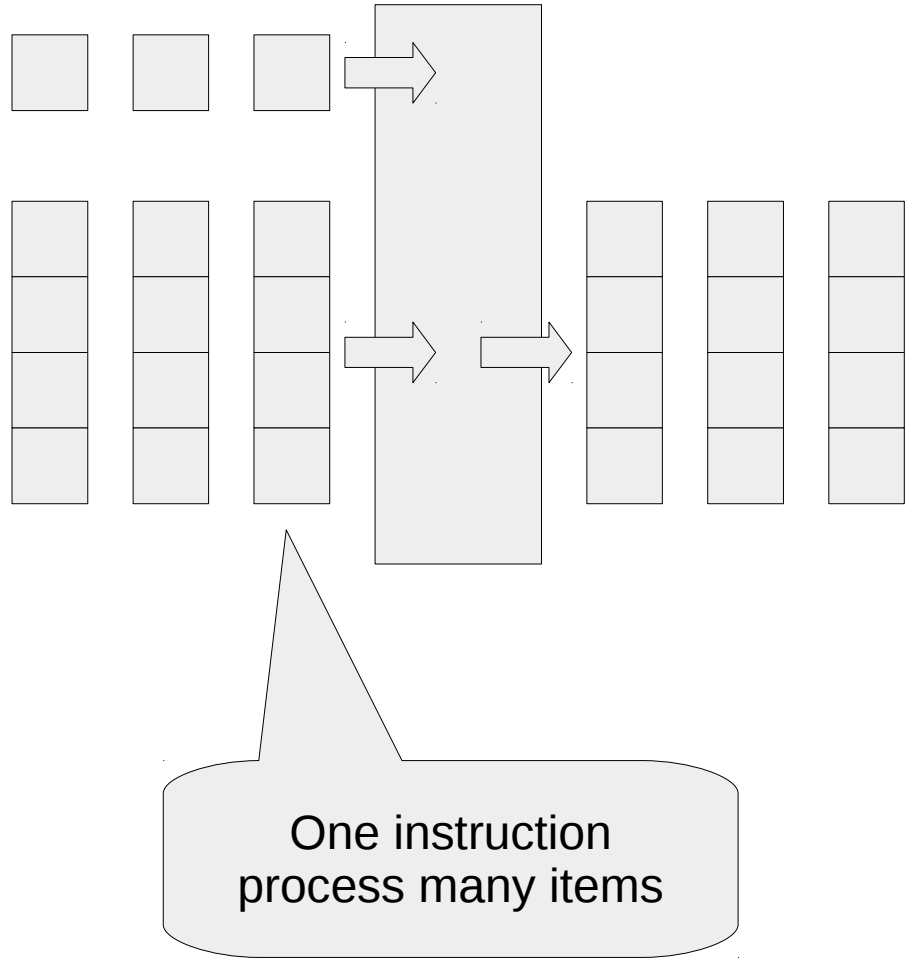
SISD

Single instruction on single data

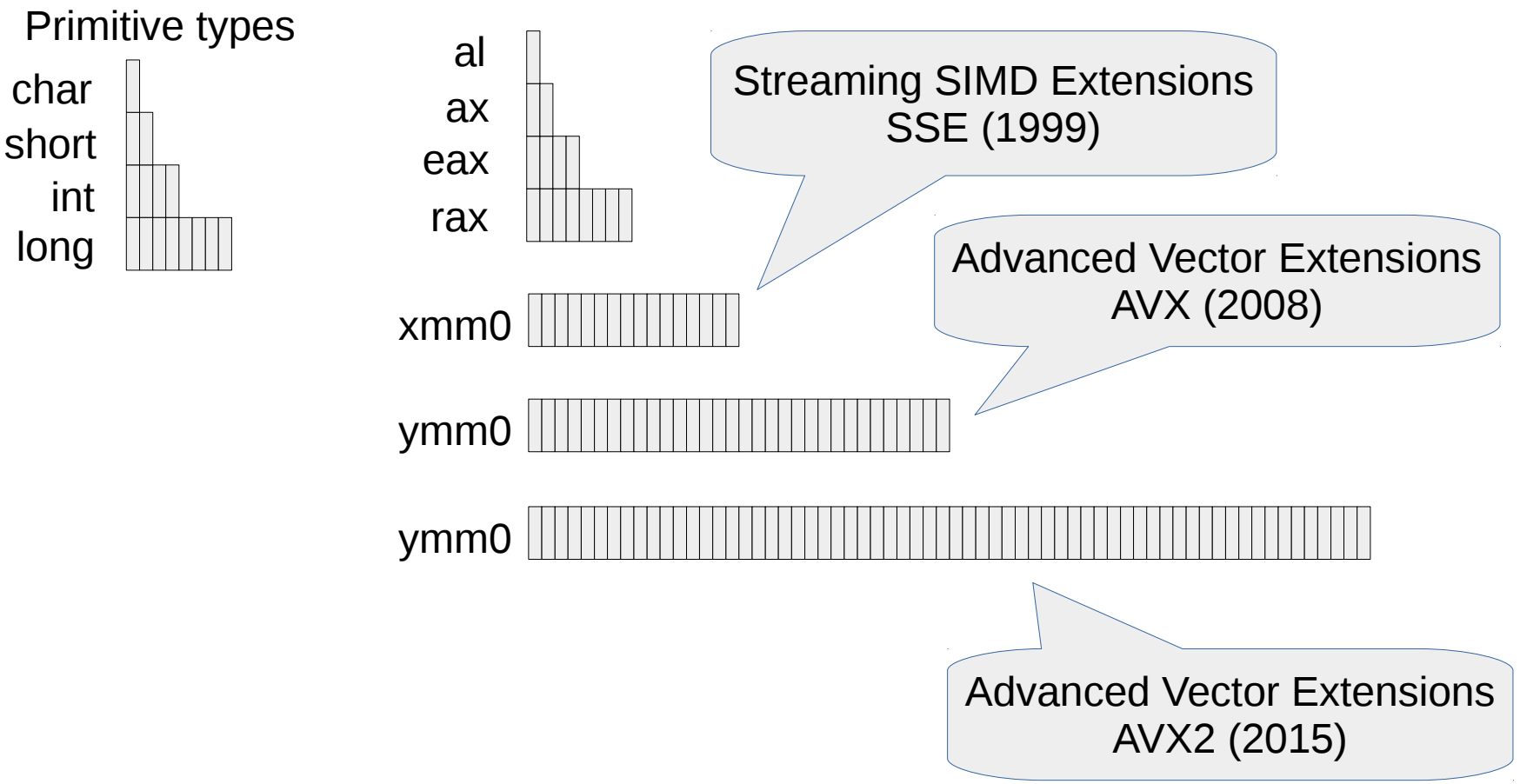


SIMD

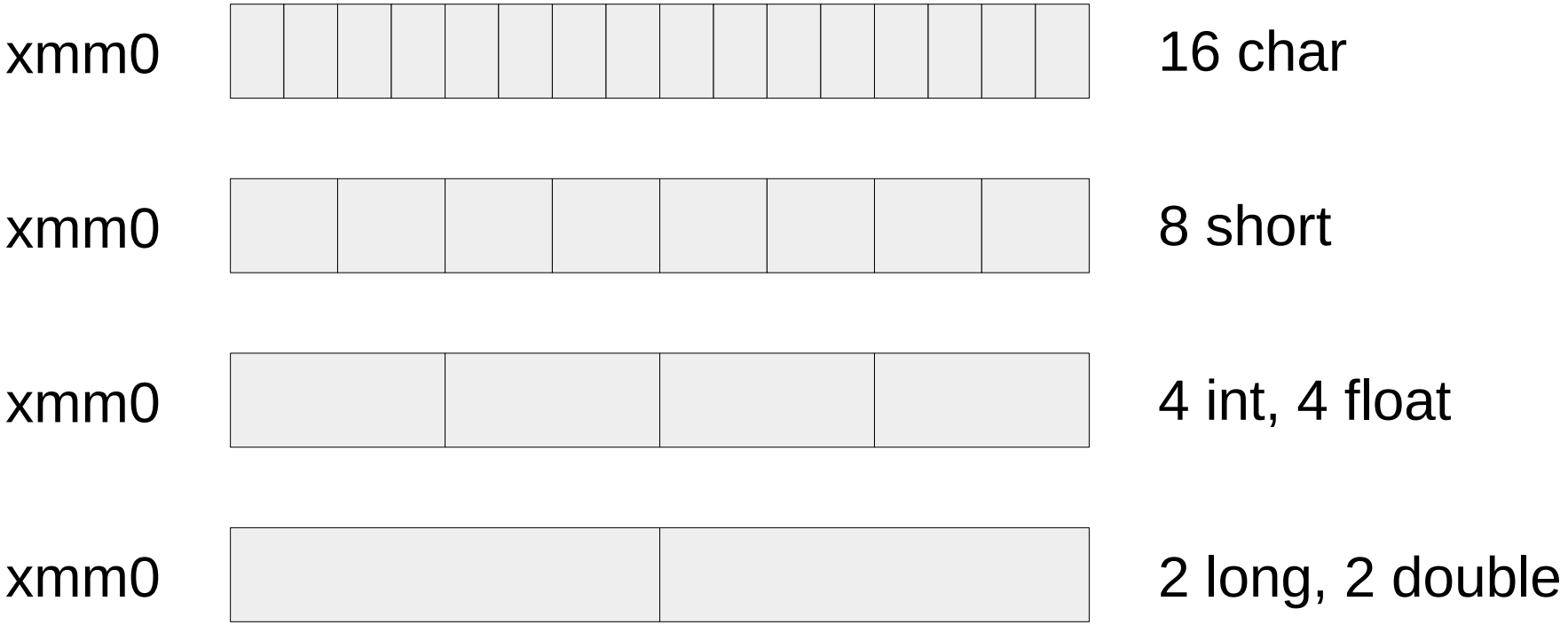
Single instruction on multiple data



SIMD registers



SIMD registers



mov instructions

- Copy mem -> reg, reg -> mem, reg -> reg
- mov %src, %dst (SISD)
- mov**ss** : Move **S**calar **S**ingle-Precision Floating-Point Values (SISD, use only first float of xmm register)
- mov**aps**: Move **A**ligned **P**acked **S**ingle-Precision Floating-Point Values (SIMD)
- mov**ups**: Move **U**naligned **P**acked **S**ingle-Precision Floating- Point Values (SIMD)
- mov**upd**: Move **U**naligned **P**acked **D**ouble-Precision Floating- Point Values (SIMD)
- mov**apd**: Move **A**ligned **P**acked **D**ouble-Precision Floating-Point Values (SIMD)

Référence: http://x86.renejeschke.de/

movups example

Move **U**naligned **P**acked **D**ouble-Precision Floating- Point Values (SIMD)

```
movss %xmm0, %xmm1
```

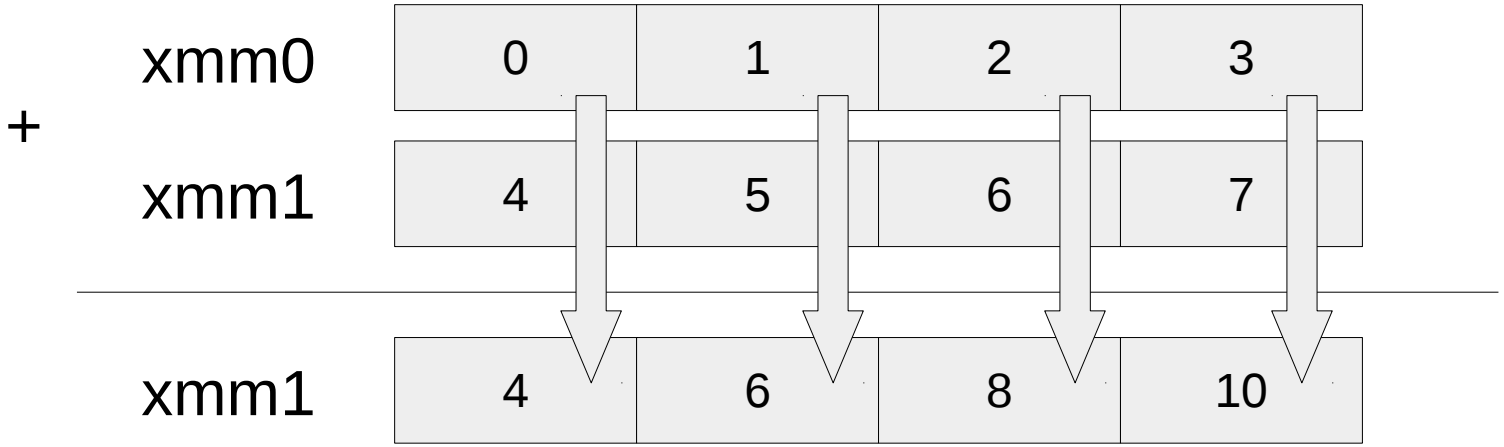
First element of xmm0 is copied to xmm1 (4 bytes)

```
movups %xmm0, %xmm1
```

Copies 4 floats of xmm0 to xmm1 (16 bytes)

Addition

- add $1, %rax (SISD)
- addss: Add Scalar Single-Precision Floating-Point Values (SIMD, use only first float of xmm register)
- addps: Add Packed Single-Precision Floating-Point Values
- addps %xmm0, %xmm1



Scratchpad 25-simd-base ops.S

- Assembly overview
  - GCC Assembler Syntax (GAS)
  - isnt %src, %dst
- Add integers (add2, add10)
- Add floating points (fadd2)
- Replace conditional branches with SIMD (min3)

Scratchpad 25-simd-base vec.S

- Vector operations
- Scalar copy with xmm0 (movss_ex1)
- Vector copy xmm0 (movups_ex1)
- Scalar add (array_addss_iter)
- Vector add (array_addps_vect)

API intrin

- Explicit SIMD instructions in C++
- Use low-level assembly without actually writing assembly
- #include <xmmintrin.h>
- Header only (no runtime shared library)

API intrin

- Data type
  - __m128 => float x[4]
- Init
  - __m128 A = _mm_setzero_ps(); // { 0.0f, 0.0f, 0.0f, 0.0f }
  - __m128 A = _mm_set_ps(v1, v2, v3, v4);
- Load
  - __m128 A = _mm_loadu_ps(float *p)
- Add
  - __m128 C = _mm_add_ps(A, B)
- Store
  - _mm_store_ps(float *p, C)

Scratchpad 25-simd-base

- Fonction array_addps_cpp_vect() with API SIMD intrin
- Should do the same as array_addss_cpp()
- Must accept tables of any size

Automatic vectorization

- In certain simple cases, compiler can automatically use vector instruction
- gcc -O3 (-ftree-vectorize)

```
3  __attribute__((noinline))
4  void array_add(float *a, float *b, float *c, long length)
5  {
6      for (long i = 0; i < length; i++){
7          a[i] = b[i] + c[i];
8      }
9  }
```

1) must be inner loop
2) bounded loops
3) independent data
4) contiguous data (array)

Problematic code for auto vector

```
for (long i = 0; i < n; i++) {
    a[i] = b[i] + c[i];
    if (a[i] == 0)
        break;
}
```

Branch in the loop

```
for (int i = 0; b[i] != 0; i++) {
    a[i] = b[i] + c[i];
}
```

Unkown number of iterations

```
for (int i = 0; i < n - 1; i++) {
    a[i] = a[i + 1] + b[i];
}
```

Data dependency

```
for (int i = 0; i < n; i++) {
    a[i] = b[i] + c[i];
}
```

```
for (int i = 0; i < n; i++) {
    a[i] = b[i] + c[i];
    for (int j = 0; j < n; j++){
        x[i][j] = y[i][j] * a[i];
    }
}
```

Only inner loop can be vectorized

POLYTECHNIQUE MONTRÉAL

ERICSSON

Advanced SIMD

1 - Vectorize moving average 3

```
void mean3(QVector<float> &m, const QVector<float> &x)
{
    for (int i = 1; i < x.size() - 1; i++) {
        m[i] = (x[i-1] + x[i] + x[i+1]) * (1/3.0f);
    }
}
```

1 - Solution

```
void mean3(QVector<float> &m, const QVector<float> &x)
{
    for (int i = 1; i < x.size() - 1; i++) {
        m[i] = (x[i-1] + x[i] + x[i+1]) * (1/3.0f);
    }
}
```

```
X = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

0 + 1 + 2 = 3
1 + 2 + 3 = 6
2 + 3 + 4 = 9
3 + 4 + 5 = 12

{ 0, 1, 2, 3 }
{ 1, 2, 3, 4 }
{ 2, 3, 4, 5 }
===============
{ 3, 6, 9, 12 }
```

```
float f = 1/3.0f;
__m128 F = _mm_set_ps(f, f, f, f);

__m128 V1 = _mm_loadu_ps(x.data() + 0 + i * 4);
__m128 V2 = _mm_loadu_ps(x.data() + 1 + i * 4);
__m128 V3 = _mm_loadu_ps(x.data() + 2 + i * 4);
__m128 A = _mm_add_ps(V1, V2);
__m128 B = _mm_add_ps(A, V3);
__m128 C = _mm_mul_ps(B, F);
_mm_storeu_ps(m.data() + 1 + i * 4, C);
```

2 – Vectorize a branch

```
void sature_serial(QVector<float> &in,
                   QVector<float> &out,
                   float max)
{
    for (int i = 0; i < in.size(); i++) {
        if (in[i] < max) {
            out[i] = in[i];
        } else {
            out[i] = max;
        }
    }
}
```

_mm_cmplt_ps()
_mm_and_ps()
_mm_andnot_ps()
_mm_or_ps()

POLYTECHNIQUE MONTRÉAL

ERICSSON

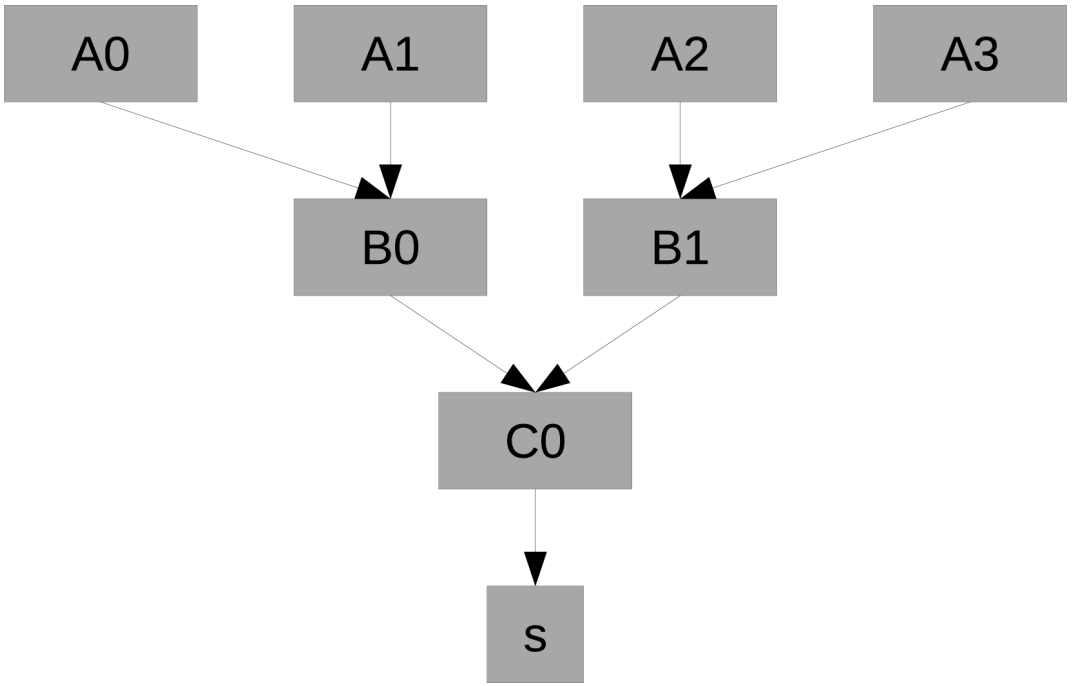
2 - Solution

```
1  void sature_simd(QVector<float> &in,
2                   QVector<float> &out, float max)
3  {
4      __m128 V_max = _mm_set1_ps(max);
5      for (int i = 0; i < in.size(); i += 4) {
6          __m128 V_in = _mm_loadu_ps(in.data() + i);
7          __m128 mask = _mm_cmplt_ps(V_in, V_max);
8          __m128 V_true = _mm_and_ps(mask, V_in);
9          __m128 V_false = _mm_andnot_ps(mask, V_max);
10         __m128 result = _mm_or_ps(V_true, V_false);
11         _mm_storeu_ps(out.data() + i, result);
12     }
13 }
14
15 Exemple de variables:
16       V_in {  2,  3,  4,  5 }
17       mask { -1, -1,  0,  0 }
18     V_true {  2,  3,  0,  0 }
19    V_false {  0,  0,  4,  4 }
20     result {  2,  3,  4,  4 }
21
22 Performance (size = 1E6):
23 Serie: 94 ms
24 SIMD: 21 ms (acceleration de 4.5x, Intel i7-4600U)
```

-1 == 0xFFFFFFFF

3 – Tree reduction

```
1  float reduce_serial(QVector<float> &in)
2  {
3      float sum = 0.0;
4      for (int i = 0; i < in.size(); i++) {
5          sum += in[i];
6      }
7      return sum;
8  }
```



3 - Solution

```
1  float reduce_simd(QVector<float> &in)
2  {
3      float sum = 0;
4      for (int i = 0; i < in.size() / 16; i++) {
5          int x = i * 16;
6          /* niveau 2 */
7          __m128 A0 = _mm_loadu_ps(in.data() + x + 0);
8          __m128 A1 = _mm_loadu_ps(in.data() + x + 4);
9          __m128 A2 = _mm_loadu_ps(in.data() + x + 8);
10         __m128 A3 = _mm_loadu_ps(in.data() + x + 12);
11
12         /* niveau 1 */
13         __m128 B0 = _mm_add_ps(A0, A1);
14         __m128 B1 = _mm_add_ps(A2, A3);
15         __m128 C0 = _mm_add_ps(B0, B1);
16
17         /* niveau 0 */
18         float* s = ((float *) &C0);
19         sum = sum + s[0] + s[1] + s[2] + s[3];
20     }
21     // restant
22     int begin = in.size() - (in.size() % 16);
23     for (int i = begin; i < in.size(); i++) {
24         sum += in[i];
25     }
26     return sum;
27 }
28
29 Performance (size = 1E6)
30 Serie: 42 ms
31 SIMD: 10 ms (acceleration 4.2x, Intel i7-4600U)
```

Leaves (16)

Reduction 16 -> 8

Reduction 8 -> 4

Reduction 4 -> 1

POLYTECHNIQUE MONTRÉAL

ERICSSON

4 – What does the following code?

```
__m128 v       = _mm_set_ps(40, 30, 20, 10);
__m128 movehl  = _mm_movehl_ps(v, v);
__m128 add     = _mm_add_ps(v, movehl);
__m128 shuffle = _mm_shuffle_ps(add, add,
                                _MM_SHUFFLE(0, 0, 0, 1));
__m128 result  = _mm_add_ss(add, shuf);
```

{ 10, 20, 30, 40 }

; m1 =

; m2 =

MM_SHUFFLE(1, 0, 3, 2)

Selector m2

Selector m1

```
m3 = _mm_shuffle_ps(m1, m2, _MM_SHUFFLE(1,0,3,2))
```

; m3 =

Source: https://msdn.microsoft.com/en-us/library/4d3eabky(v=vs.90).aspx

4 - Solution

```
          v {  10, 20, 30, 40 }
    movehl {  30, 40, 30, 40 }
       add {  40, 60, 60, 80 }
   shuffle {  60, 40, 40, 40 }
    result { 100, 60, 60, 80 }

Exemples d'operations shuffle:
                          v { 10, 20, 30, 40 }
shuffle(v, v, {0,0,0,0}) { 10, 10, 10, 10 }
shuffle(v, v, {1,1,1,1}) { 20, 20, 20, 20 }
shuffle(v, v, {0,1,2,3}) { 40, 30, 20, 10 }
shuffle(v, v, {3,2,1,0}) { 10, 20, 30, 40 }
shuffle(v, v, {3,0,0,0}) { 10, 10, 10, 40 }
shuffle(v, v, {0,3,0,0}) { 10, 10, 40, 10 }
shuffle(v, v, {0,0,3,0}) { 10, 40, 10, 10 }
shuffle(v, v, {0,0,0,3}) { 40, 10, 10, 10 }
shuffle(v, v, {0,0,0,1}) { 20, 10, 10, 10 }
```

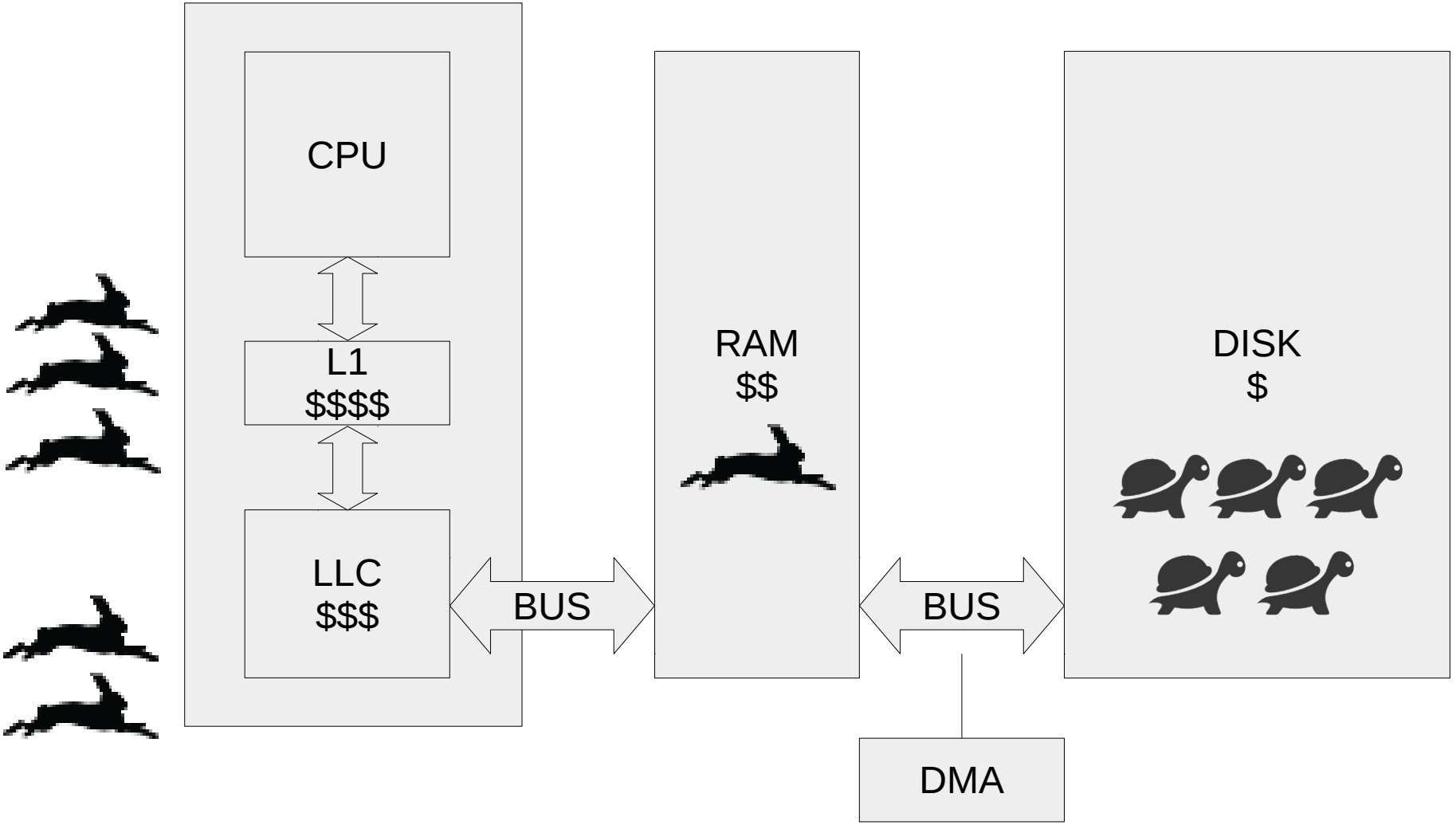
MM_SHUFFLE(0, 0, 0, 1)

6 – Search for End Of Line

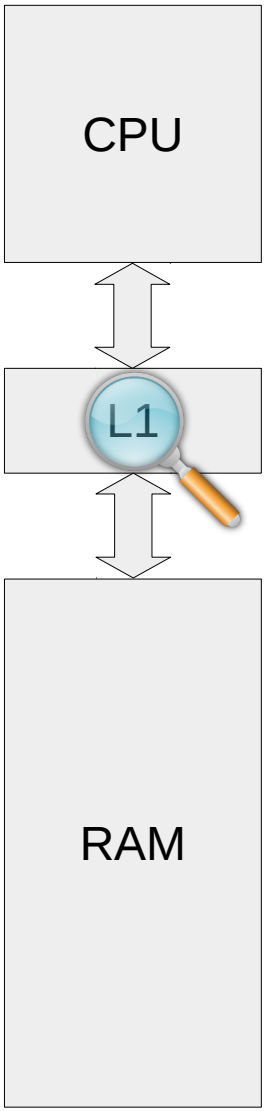
```
const char *buf = "abc\ndef\n~~~~~~~";
__m128i vec = _mm_loadu_si128((__m128i *) buf);
__m128i eol = _mm_set1_epi8('\n');
__m128i cmpeq = _mm_cmpeq_epi8(vec, eol);
uint res = _mm_movemask_epi8(cmpeq);
int pos = __builtin_ctz(res); // ou _BitScanReverse()
    sur MS Windows
```

```
Sortie:
vec 61 62 63 0A 64 65 66 0A 7E 7E 7E 7E 7E 7E 7E 00
eol 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A
cmp 00 00 00 FF 00 00 00 FF 00 00 00 00 00 00 00 00
res  0  0  0  1  0  0  0  1  0  0  0  0  0  0  0  0
pos 3
```

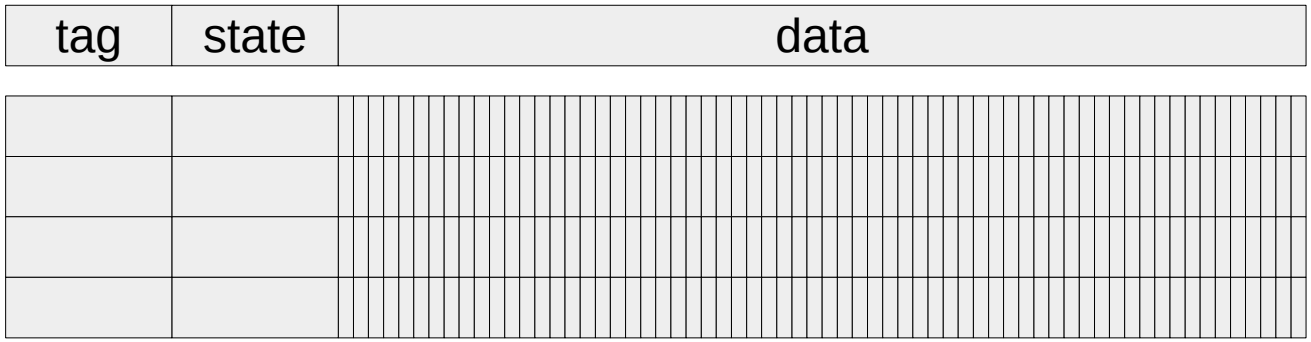
Memory hierarchy



Cache line anatomy



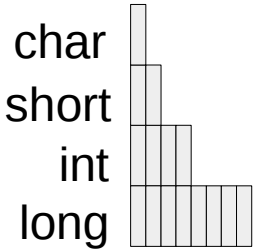
Intel cache line:
64 bytes



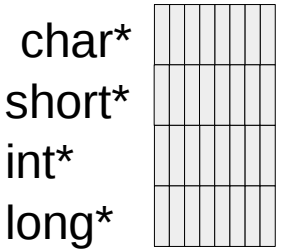
...

Layout in memory

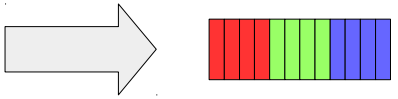
Built-in types



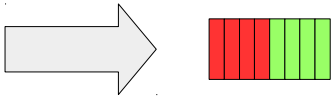
Pointers



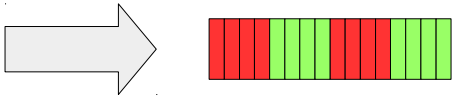

```
vector<int> a = {1, 2, 3};
```



```
typedef struct point {
    int x;
    int y;
} point_t;
```

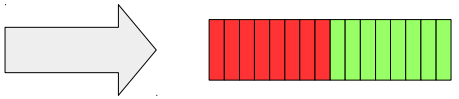


```
vector<point_t> a = {
  { 1, 2 },
  { 3, 4 },
};
```



Array of Structs (AoS)

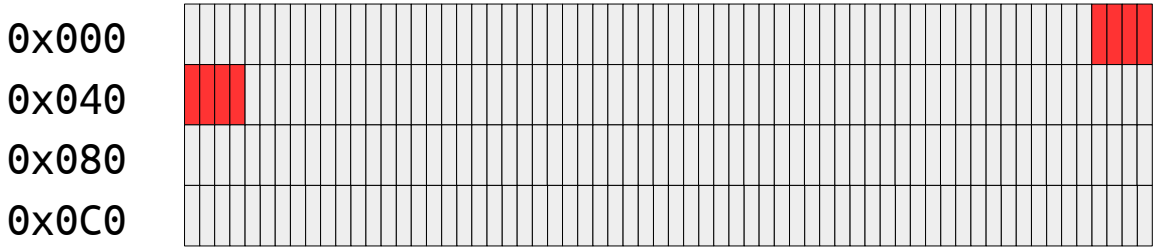
```
typedef struct data {
    vector<int> x = { 1, 2 };
    vector<int> y = { 3, 4 };
} data_t;
```



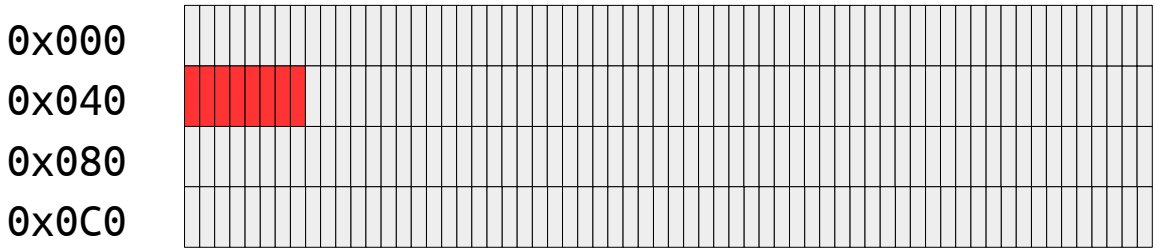
Struct of Arrays (SoA)

Data location in the cache

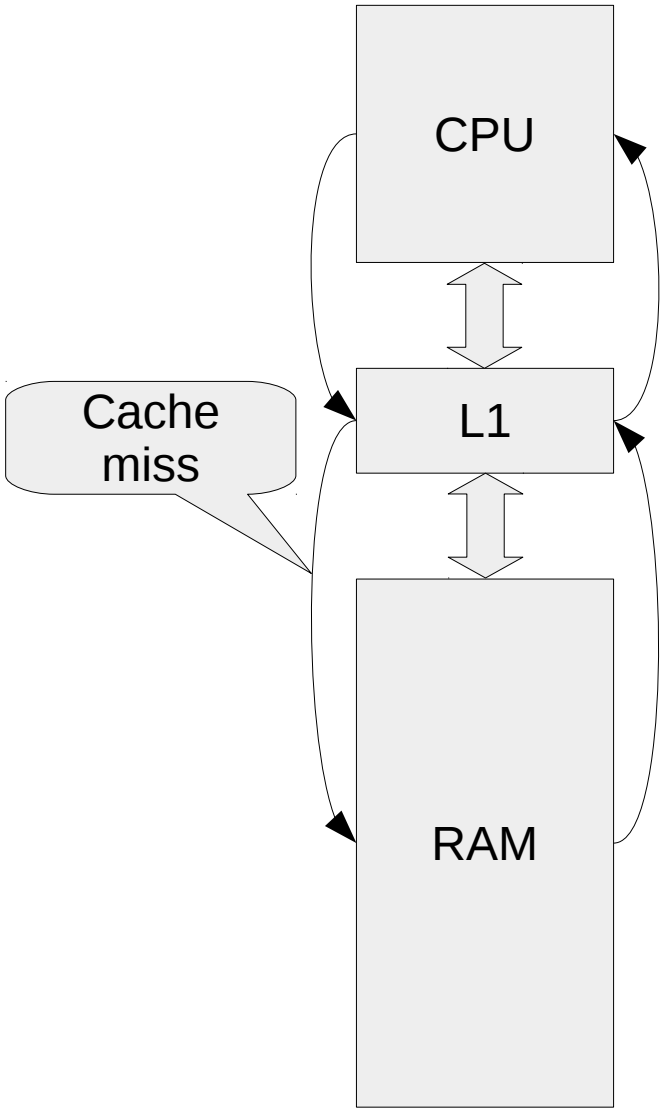
```
long x = 42;
long *p_x = &x;
// p_x == 0x03C (60)
```



```
long __attribute__((__aligned__(64))) x = 42;
long *p_x = &x;
// p_x == 0x040 (64)
```



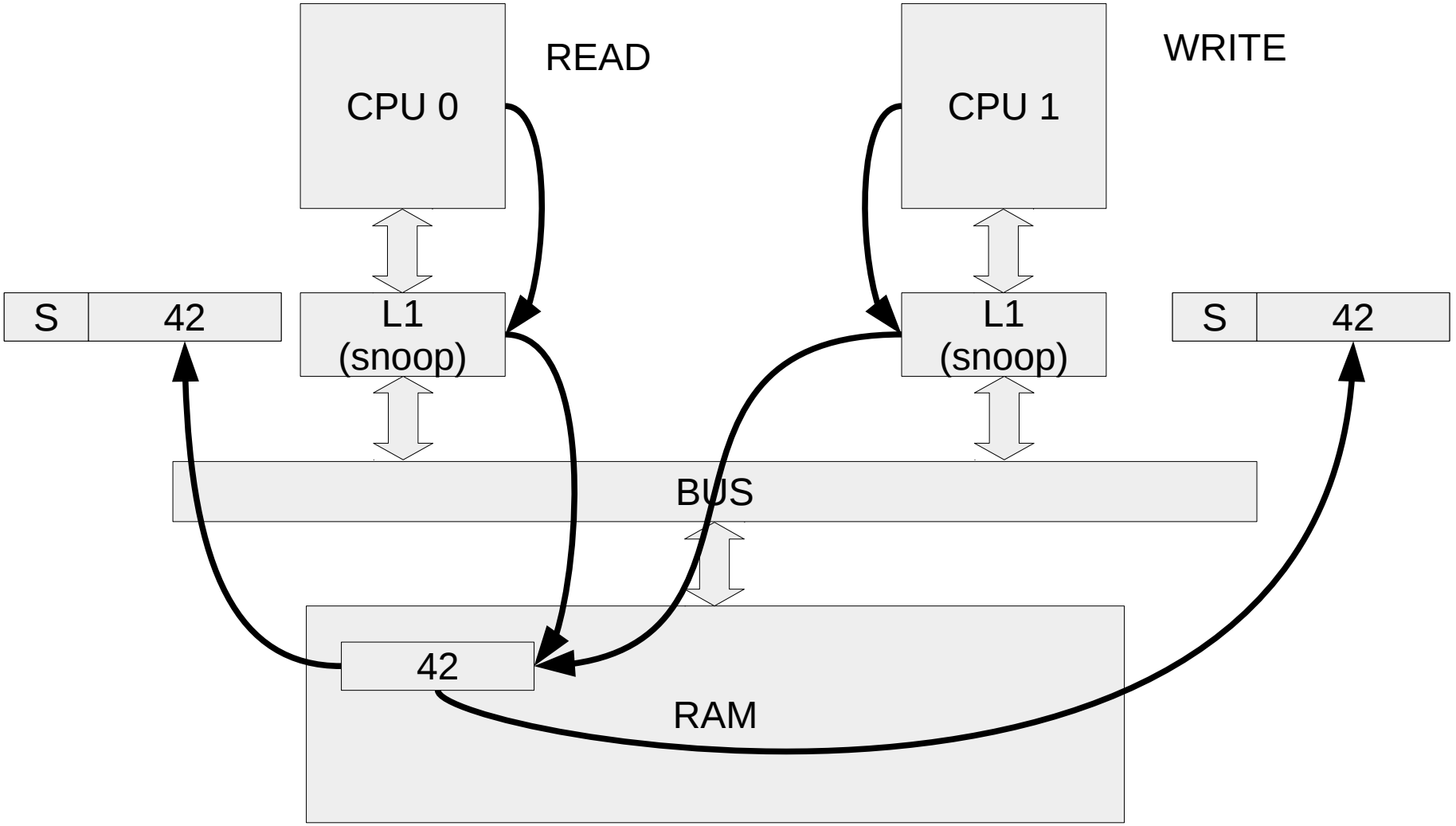
Cache access for uniprocessors



- Reading
  - If the cache line is present, use it
  - If not present, fetch the line from the main memory (cache miss)
  - No free cache line: eviction
- Write-through
  - Each write goes to main memory
  - Simple solution, but uses a lot of bandwidth
- Write-back
  - Fetch the line
  - Modify the line
  - Change the state to dirty=true
  - Write to main memory only in case of eviction and dirty=true

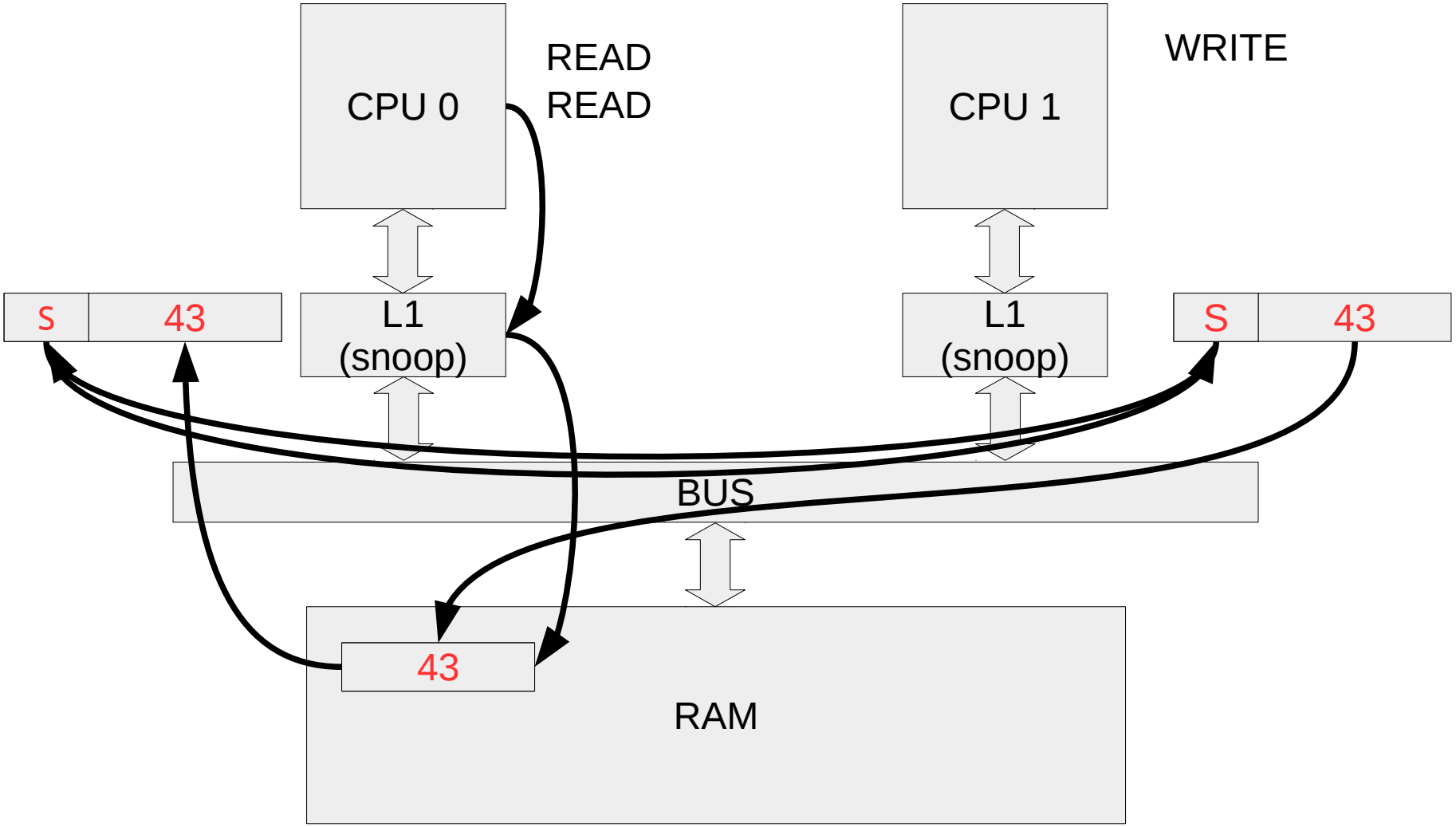
Multicore: cache coherency

Modified Shared Invalid (MSI)



Multicoeurs: cohérence de cache

Modified Shared Invalid (MSI)



Cache coherency protocoles

- MSI
  - Simple to implement
  - High replication rate
- MESI
  - Add state “**E**xclusive”
  - Reduces useless replication
- MOSI
  - Add state “**O**wned”
  - Reduces write-back
- MOESI
  - Combines de MESI + MOESI
- MESIF
  - Add “**F**orward” to replicate cache lines between caches directly (without going through main memory)

Memory barrier

```
struct personne {
  Char *birthday;
  char *email;
  char *name;
}
struct personne boss = NULL;
```

Modification (CPU 0)

```
struct *new_boss = malloc(1,
sizeof(struct personne));

new_boss->birthday = “1970-01-01”;
new_boss->email = “foo@bar”;
new_boss->name = “foo bar”;

// set the new boss
boss = new_boss;
```

The compiler can reorder the assignments: the new boss structure may be visible to CPU1 before all fields are set

Read (CPU 1)

```
// send a mail to my boss
// for it's birthday

char *msg = “Happy Birthday!”;

if (boss != NULL) {
  if (boss->birthday == today()) {
    send_email(boss->email, msg);
  }
}
```

Memory barrier

```
struct person {
  Char *birthday;
  char *email;
  char *name;
}
struct person *boss = NULL;
```

Modification (CPU 0)

```
struct person *new_boss =
malloc(1, sizeof(struct person));

new_boss->birthday = “1970-01-01”;
new_boss->email = “foo@bar”;
new_boss->name = “foo bar”;

// set the new boss
__sync_synchronize();
boss = new_boss;

// free() de l'ancien boss
// quand on est certain
// qu'il n'y a plus d'utilisateurs
// ex: refcount
```

Read (CPU 1)

```
// send a mail to my boss
// for it's birthday

struct person *my_boss;
char *msg = “Happy Birthday!”;

my_boss = ACCESS_ONCE(boss);
if (my_boss != NULL) {
  if (my_boss->birthday == today()) {
    send_email(my_boss->email, msg);
  }
}
```