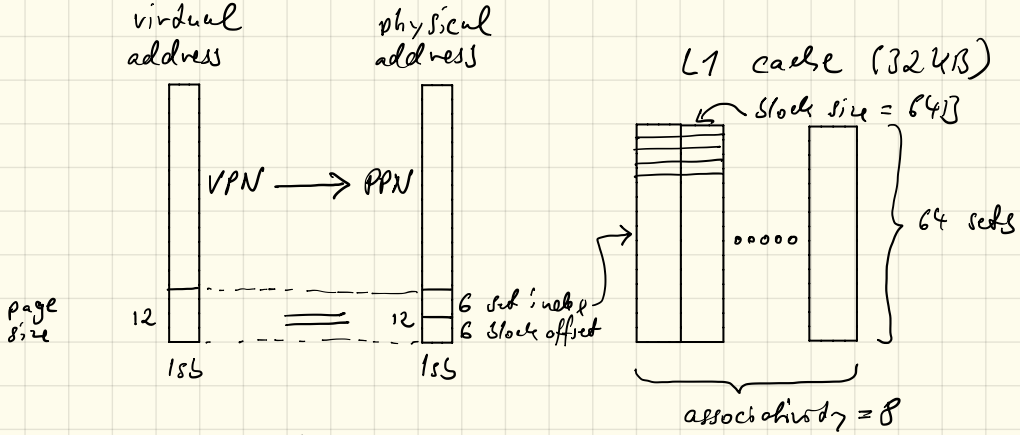
Virtual Memory System (Core Family)

- the processor works with virtual addresses
- all caches work with physical addresses
- both address spaces are organized in pages

page size: 4kB (can be changed to 2MB and even 1GB on the latest processor; change in OS settings)

- address translation: virtual page number (VPN) → physical page number (PPN)



⇒ L1 cache lookup can start concurrently with address translation

Address translation

- uses a cache called translation lookaside buffer (TLB)
- Core Sandybridge and later:

ITLB (instructions): 128 entries DTLB (data): 64 entries

STLB (shared level 2): 512 – 1536 entries
(Sandy Bridge – Sky Lake)

DTLB hit: no penalty
DTLB miss, STLB hit: few cycles penalty
STLB miss: can be very expensive

Impact on performance

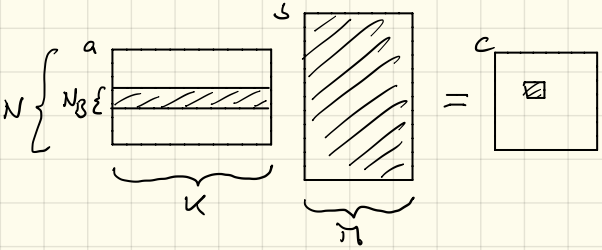
Repeatedly accessing a working set spread over too many pages (uncached) yields TLB misses and can yield significant slow down.

Example: STLB = 1024 (Haswell)

accessing repeatedly 2048 doubles spread out in memory = working set of $2048 \times 64 = 128\,kB$
(64: cache block)

⇒ fits into L2 cache
but TLB misses

Example MMM



working set at highest level is shaded

We look for parts in working set spread in memory
- block row of a: contiguous
- all of b: contiguous
- block of c: if $M > 512$ (512 doubles = 4KB) then spread over $\geq NB$ pages

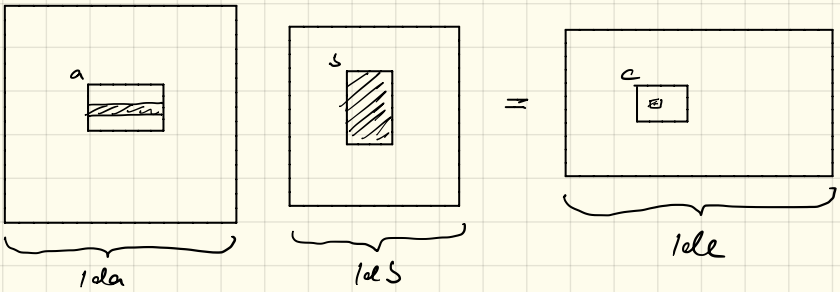
Typically NB is in the 10s, so no big problem

But: the BLAS function dgemm has this interface:

dgemm (a, b, c, N, K, M, lda, ldb, ldc)

matrices — sizes — leading dimensions

Leading dimensions: enable use on matrices inside matrices



assume $lda, ldb, ldc > 512$

- block row of a: spread over $\geq N_B$ pages
- all of b: spread over $\geq K$ pages
- block of c: spread over $\geq N_B$ pages

So copying to contiguous memory may pay off!

```
// all of b reused: possibly copy
for i = 0: NB: N-1
    // block row of a reused: possibly copy
    for j = 0: NB: M-1
        // block of c reused: possibly copy
        for k = 0: NB: K-1
            ....
```