# Amortized modification cost (cont.)

**Thm:** Any sequence of $m$ INSERTs / DELETEs on initially empty $(a, 2a)$-tree does $O(m)$ modifications.

Pf: goal: find $\Phi$ s.t. $A_{\substack{\text{add}\\\text{remove}\\\text{move}}} = O(1)$, $A_{\substack{\text{split}\\\text{merge}}} \le 0$ w.r.t. $\Phi$.

$(\sum R_i = \sum A_i - \Delta\Phi)$

$\Phi := \sum_v f(|v|)$ where $f: \{a-2, a-1, \dots, 2a-1, 2a\} \to \mathbb{N}$ properly chosen:

(#keys in $v$; $a-2$ undersized, $2a$ oversized)

1) $|f(i) - f(i+1)| \le c$ for some const. $c$

2) $f(2a) \ge f(a) + f(a-1) + c + 1$ for $A_{split} = O(1) + \Delta\Phi \le 0$

($2a$ oversized; $a$, $a-1$ new nodes; $c$ add to parent; $+1$ rescaling to cover real cost)

3) $f(a-2) + f(a-1) \ge f(2a-2) + c + 1$ for $A_{merge} = O(1) + \Delta\Phi \le 0$

($a-2$ undersized; $a-1$ sibling; $2a-2$ new node; $c$ remove from parent; $+1$ rescaling)

For $f$ we can take e.g.:

| $k$ | $a-2$ | $a-1$ | $\dots$ | $2a-1$ | $2a$ |
|---|---|---|---|---|---|
| $f(k)$ | 2 | 1 | 0 … 0 | 2 | 4 |

✓

⇒ INSERT / DELETE $O(1)$ amortized modification cost

⇒ total real cost $\sum R_i = \sum A_i - \underbrace{\Phi_m}_{\ge 0} + \underbrace{\Phi_0}_{=0} = O(m)$. □

## Top-down $(a,b)$-trees

(median to parent)

INSERT: if the current node is full ($b-1$ keys), split it

⇒ parent always has a free space ⇓ needs $\lfloor \frac{b-2}{2} \rfloor \ge a-1$

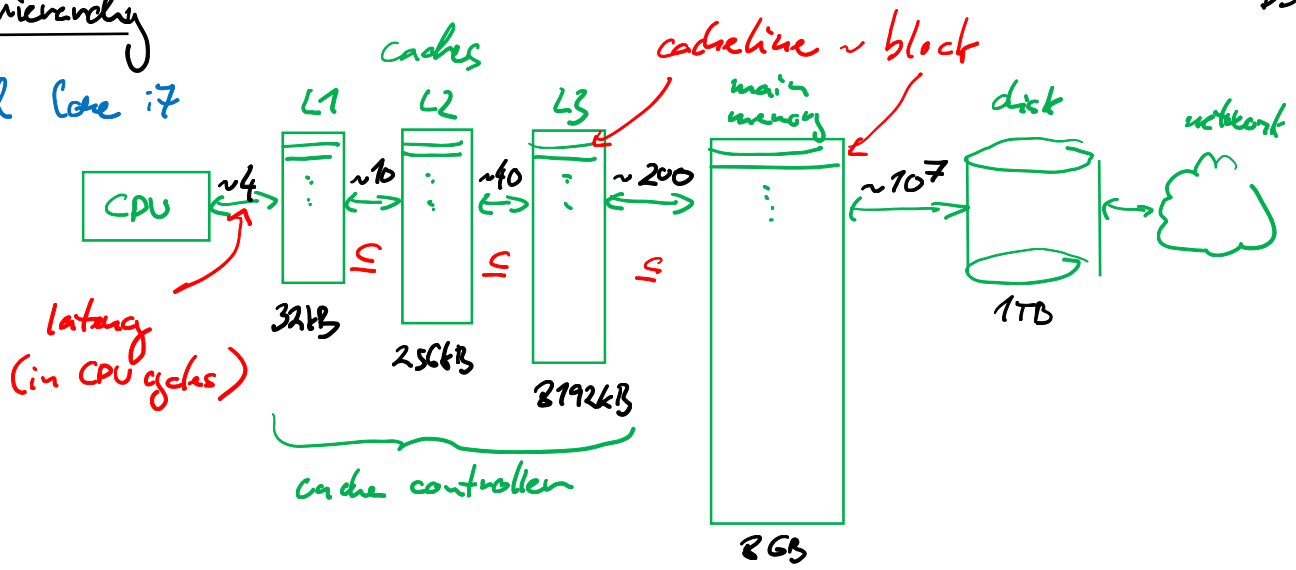
(preemptive splitting) $b \ge 2a$

DELETE: similarly, preemptive merging / moving (exercise)

⇒ useful for parallel $(a,b)$-trees: simple locking mechanism (to be discussed later)

# Memory hierarchy

Ex. Intel Core i7

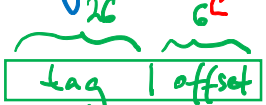



cache line (block) of size B ~ 32 / 64 B (unit of transfer)

cache hit / miss ⇒ eviction (replacement strategy)

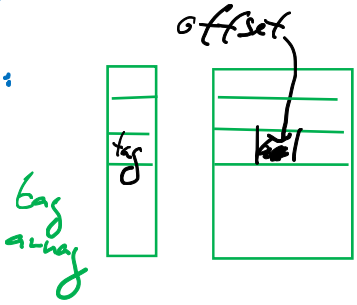
## cache mapping

1) fully associative: every block can be in any cache line



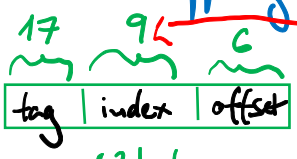

memory address (4GB addressable space)

cache:

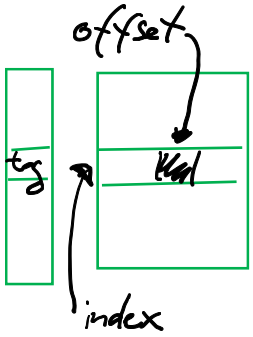



⇒ need many tag comparisions

2) direct mapping: every block can be in only one cache line

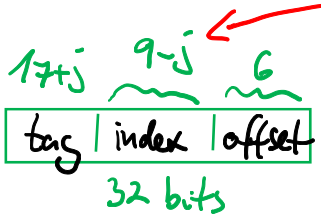



32 kB cache ⇒ 512 cachelines



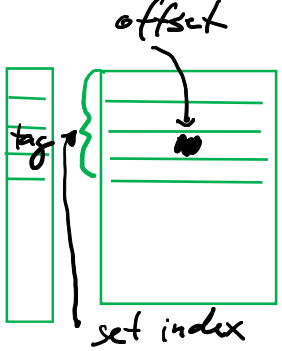

⇒ no need for a replacement strategy

3) set associative: every block can be in a set of $2^j$ cache lines



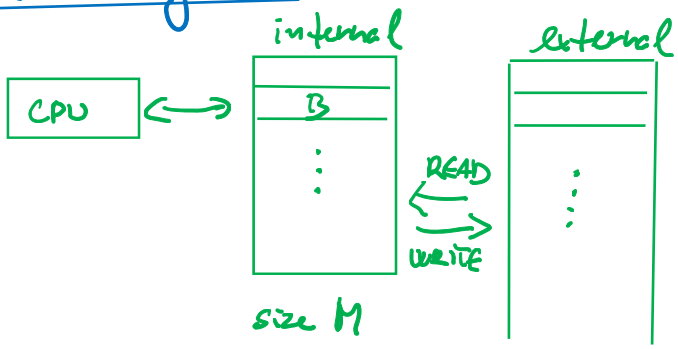

($2^j$-way associative)

(compromise between 1) and 2))



# Models of computation

## 1) External memory model (I/O model)

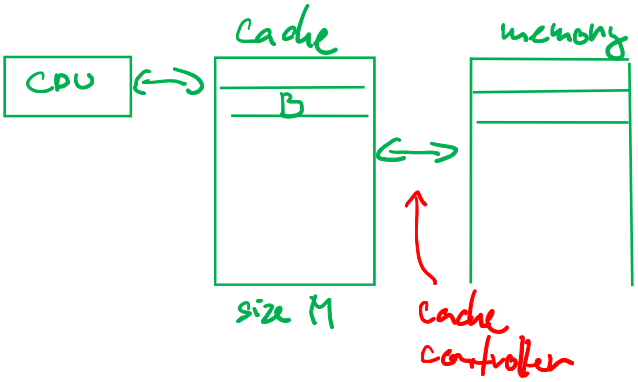



block size $B$ (# items / Byte / ...)
internal memory of size $M$
(alg.) ($M/B$ blocks)
CPU controls READs / WRITEs to the external memory

**Def:** An algorithm has **I/O complexity** $f(n, B, M)$ if for any parameters $B, M$ any input of size $n$ the number of I/O operations (read / write) is $\leq f(n, B, M)$.

**Note:** Typically, # writes = $O$(# reads) (if $|\text{output}| = O(|\text{input}|)$).
$\Rightarrow$ analysis reduces only to # reads.

## 2) cache-aware model




Alg. knows the parameters $B, M$.
Alg has no control of block transfers (it is handled by the controller).

## 3) cache-oblivious model

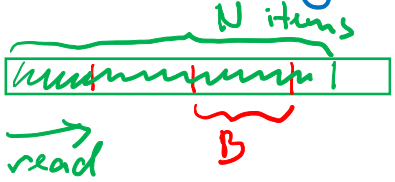
Alg. does not know $B, M$. (Maybe assumes $M \in \Omega(B^2)$.)
$\Rightarrow$ needs a strategy that works for any cache
$\Rightarrow$ more complex (overhead) but more general (good for cache hierarchies)

**Assumptions for analysis:** (offline, knows future accesses to memory)

1) cache controller has optimal replacement strategy: „evict the cache line whose block is needed furthest"

2) fully associative cache

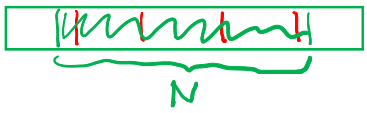
## Scan of an array (example)




1) I/O model: aligned array start
1 interval / block suffices
$\Rightarrow$ I/O complexity $= \lceil N/B \rceil$

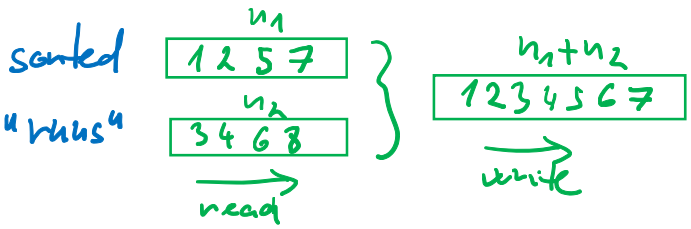
2) cache-aware: also aligned (since we know B)
I/O complexity = # cache misses (reads) in OPT strategy $\leq$ # cache misses in "use 1 cacheline" strategy $= \lceil N/B \rceil$
maybe M/B blocks already cached

3) cache oblivious: no alignment guaranteed $\Rightarrow$ I/O complexity $\leq \lceil N/B \rceil + 1$ (extra read)

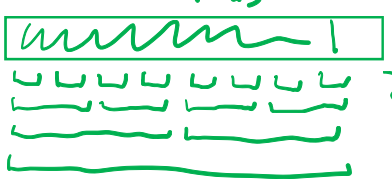



$\Rightarrow O(N/B+1)$ in all models
cannot be omitted!

## Merge sort (for sorting in external memory)




3x sequential scans $\Rightarrow$
3 blocks / cache lines suffice
$\Rightarrow$ single merge in $O(n_1+n_2)$ time
I/O complexity $O((n_1+n_2)/B+1)$ (in all models)




$\lceil \log N \rceil$ passes
switch 2 input / output arrays

single pass in $O(N)$ time
$O(N/B+1)$ block transfers
runs are consecutive

$\Rightarrow$ Merge sort in $\Theta(N \log N)$ time and I/O complexity
$O(N/B \log N + \cancel{\log N})$ $1 \leftarrow$ improved since for $N \geq B$ we have $O(N/B+1) = O(N/B)$
and for $N < B$ entire array fits in 1 block $\Rightarrow O(1)$ block transfers