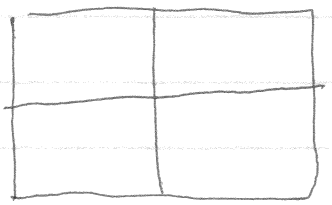
7-1

Caches — temporal + spatial locality

use the same data many times
+ use neighboring data



a - 5

a b c d e a b c d a

matrix transpose

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}^T = \begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix} \rightarrow (1\ 5\ 2\ 6\ 3\ 7\ 4\ 8)$$

in (1 2 3 4 5 6 7 8) [2,3] [3,5] [4,7] [7,6]
out (1 5 2 6 3 7 4 8)

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

```
for n = 0 to N-2
  for m = n+1 to N-1
    swap A(n,m) with A(m,n)
```

stride-length — gap between access, k => A(n), A(n+k)
if k > cache size (block), then not spatially locally

cache-oblivious — optimized regardless of cache size
(almost always divide & conquer)

NxN

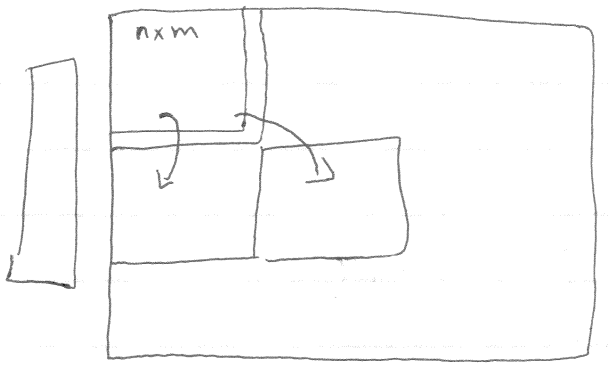
recur	manual
manual	recur

blocking — not cache-oblivious

all
~~extra~~ writes are non-temporal

17-2 / Blocking

NxM

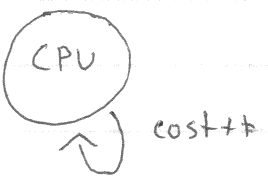


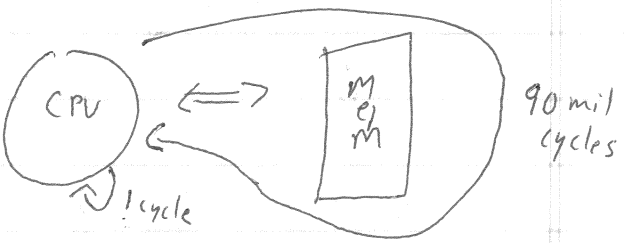
nxm - should fit M cache ↗ k

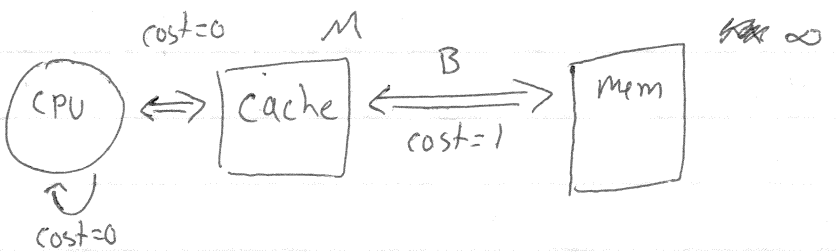
m - should be a multiple of cache line size

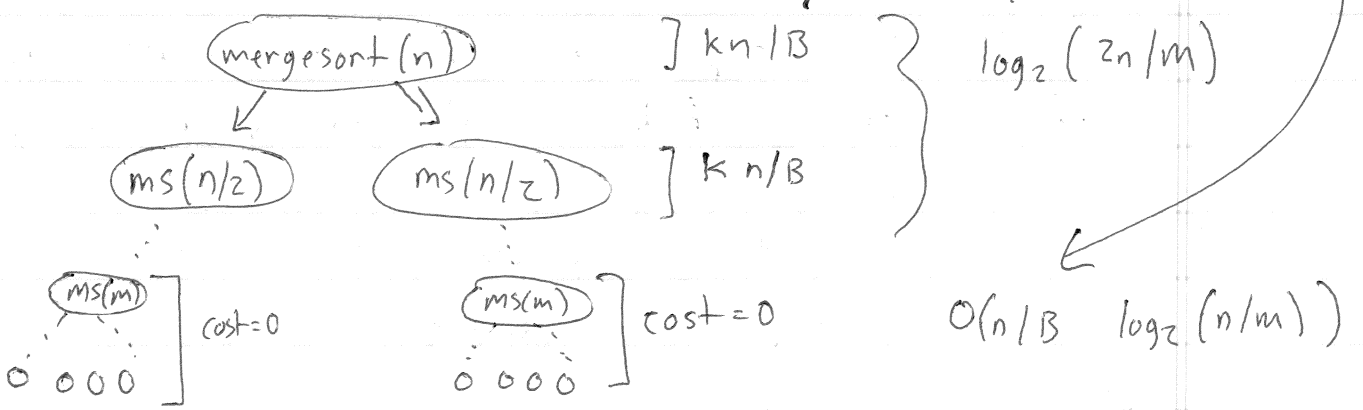
Choose n, k to fill cache

for i = 0 to N — N
op (i) — O(1)
} — N * O(1) = O(N)







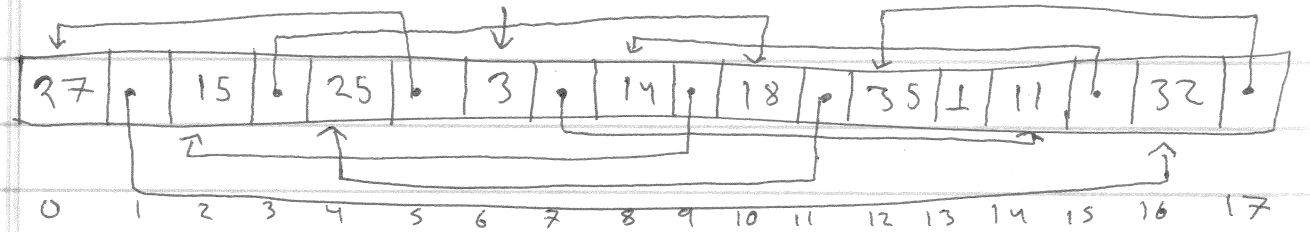


7-3/

Linked-List

struct llnode { int data; struct llnode *next };

(A)

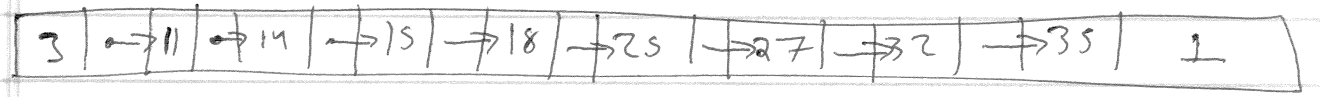


B = 4 assume M = 2

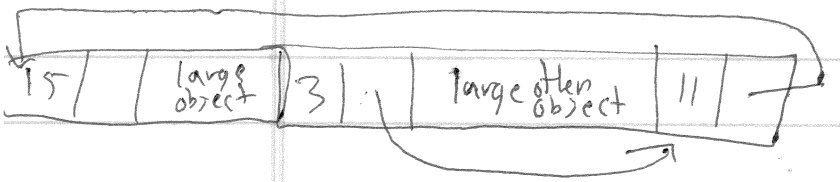
6 7 14 15 8 9 2 3 10 11 4 5 0 1 16 17 12 13

load 6,7,8,9 load [14,17] load [2,5] evict [6,9] load [10,13] evict [14,17]

(B)



~~hits~~ ~~misses~~ misses n/B



(C)

(C) < (A) < (B)

Idea 1: Separate static mem + dynamic mem (i.e. malloc)

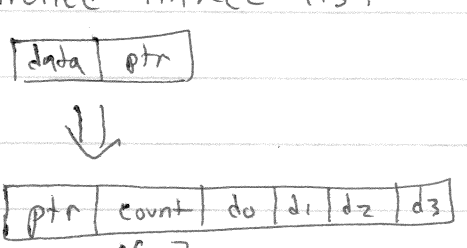
Idea 2: Slab Allocation

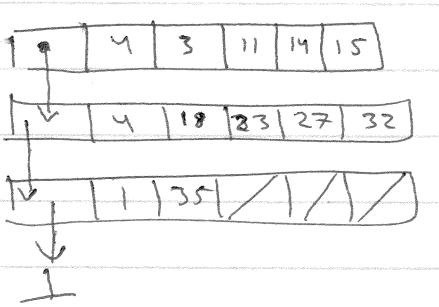
regions per type (C) → (A)

Idea 3: want (A) → (B)

1) use a different datastructure
2) rearrange things dynamically (Stop & Copy)

unrolled linked list





data = 14 ints
ovr = 7 ptrs