Merge based Sorting

» Very similar to the standard merge sort technique.

» Phase I: Called the Run-Formation Phase

- Scan one buffer of data of size B
- Sort the buffer internally

Transfer B^1 B^2 B^3

- Write back the sorted buffer

compute B^1 B^2

» Phase II: Merge Phase

- Scan two buffers of data
- Merge them

» In both phases, one can use double buffering to overlap computation with disk I/O.

Merge Based Sorting

- So, the number of passes is $O(\log_{M/B} N/B)$.

- Using software prefetching can help sorting algorithms.

- As an aside, forming a permutation of n numbers is a special case of sorting.
  - So, similar bounds apply to permuting.

Matrix Multiplication

- A good case study of how to tune algorithms to a cache size.

```
MatrixMultiply(A, B, C, n)
Begin
    for i = 1 to n do
        for j = 1 to n do
            for k = 1 to n do
                C[i,j]  += A[i.k]*B[k,j]
            end-for
        end-for
    end-for
End.
```

Matrix Multiplication

- A good case study of how to tune algorithms to a cache size.

```
MatrixMultiply(A, B, C, n)
Begin
    for i = 1 to n do
        for j = 1 to n do
            for k = 1 to n do
                C[i,j]  += A[i.k]*B[k,j]
            end-for
        end-for
    end-for
End.
```

- Suffers from lots of cache misses.
- Exercise: Estimate the number of cache misses.

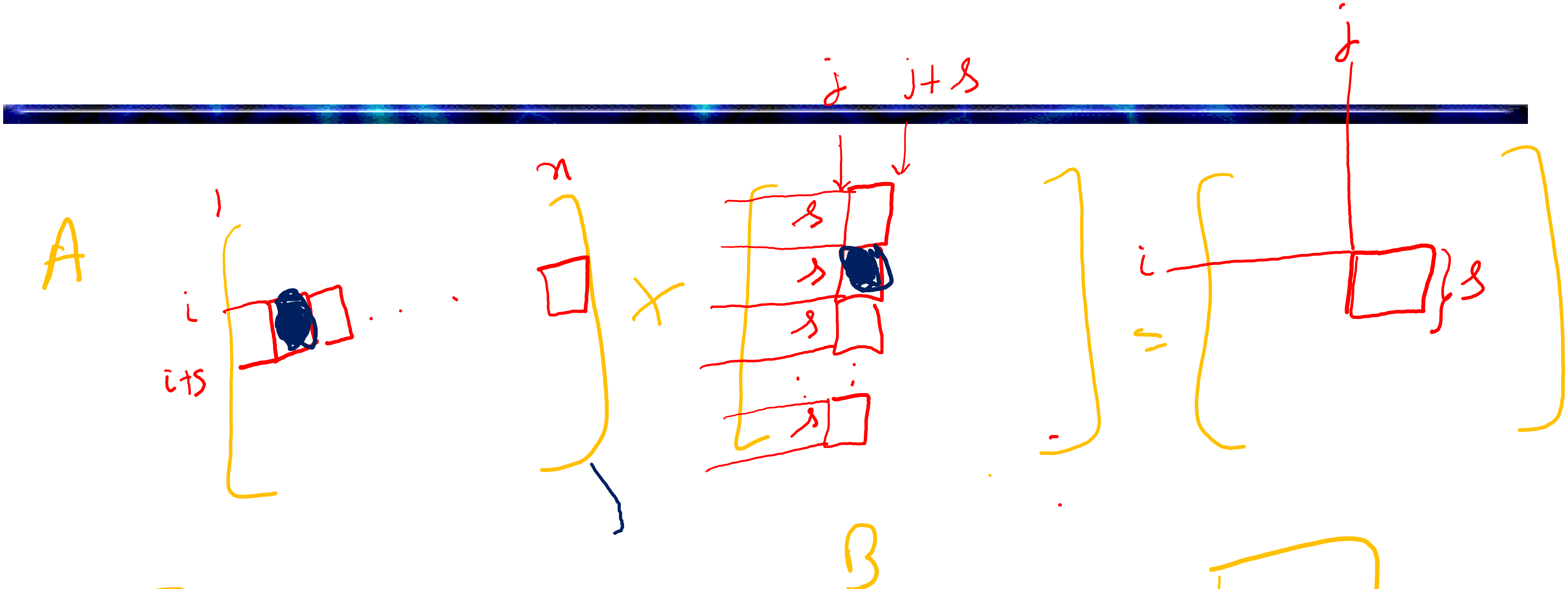
Matrix Multiplication – Naive

- To compute each element of C:
  - Read/Scan the entire row of A – O(N/B) cache misses
  - Read/Scan the column of B – O(N) cache misses.
    - Why?
  - Cache misses per element of C = O(N) + O(N/B) = O(N).
- Overall cache misses = $O(N^3)$

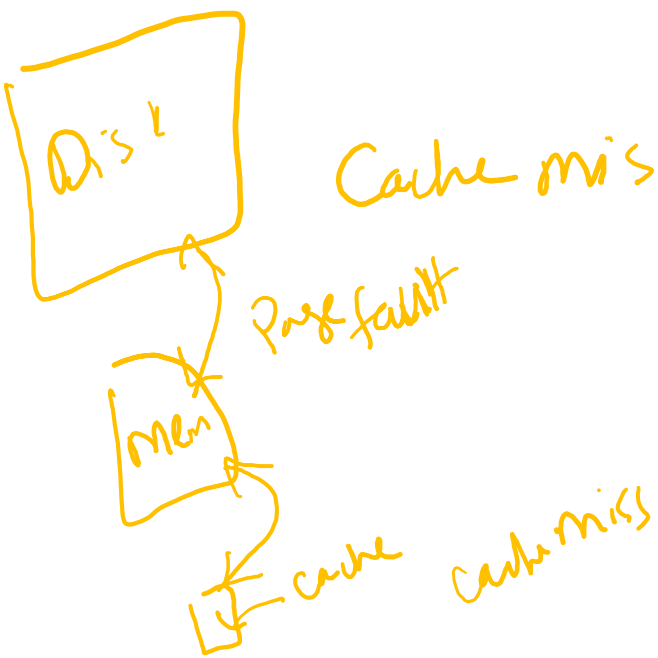
Matrix Multiplication

- It is however better to imagine that the computation be broken into blocks of sxs submatrices.

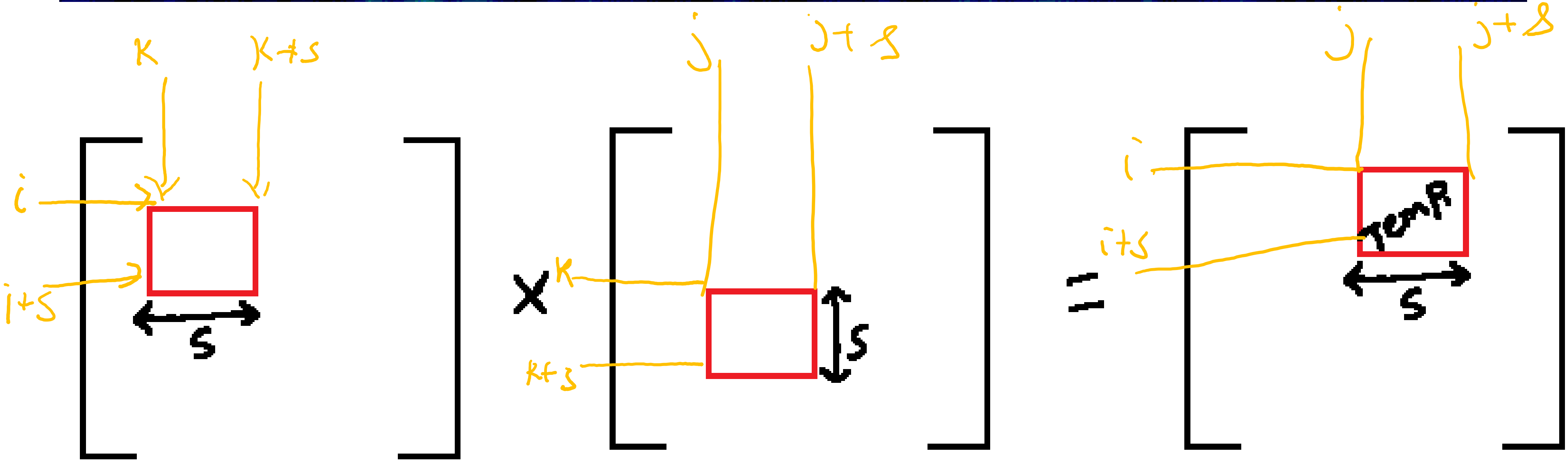
```
MatrixMultiply(A, B, C, n)
Begin
    for i = 1 to n/s do
        for j = 1 to n/s do
            for k = 1 to n/s do
                C = C + Mul(A_ik, B_kj, s)
            end-for
        end-for
    end-for
End.
```



Cache miss ≈ page fault



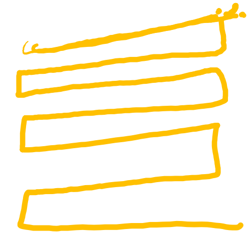
Matrix Multiplication in Pictures



$C_{ij} = A[i .. i+s, 1 : 1+s] \times B[1 .. 1+s ; j .. j+s] +$

$A[i+s+1 .. i+2s, 1 : 1+s] \times B[1 .. 1+s ; j+s+1 .. j+2s] +$

Improved Version



- Estimate the number of cache misses again.

for $i = 1$ to $\frac{n}{s}$ do

for $j = 1$ to $\frac{n}{s}$ do

for $k = 1$ to $\frac{n}{s}$ do

$C_{ij} += \text{Mul}(A_{ik}, B_{kj}, s)$ // s^3 comp

for one C_{ij} # cache misses $= O\left(\frac{s^2}{B}\right)$

$\# C_{ij} = \left(\frac{n}{s}\right)^3$

Total $= \left(\frac{n}{s}\right)^3 \times \frac{s^2}{B}$

$= O\left(\frac{n^3}{sB}\right)$

Improved Version

- Estimate the number of cache misses again.

- To compute an sxs submatrix of C, we need to bring in $O(s^2)$ elements into the cache.
- Results in $O(s^2/B)$ cache misses.

Improved Version

- Estimate the number of cache misses again.

- To compute an sxs submatrix of C, we need to bring in $O(s^2)$ elements into the cache.
- Results in $O(s^2/B)$ cache misses.

- For the entire C matrix, the number of cache misses = $O(s^2/B) \times (n/s)^3 = O(n^3/sB)$.
  - Why?

iterations

Improved Version

- Estimate the number of cache misses again.

- To compute an sxs submatrix of C, we need to bring in $O(s^2)$ elements into the cache.
- Results in $O(s^2/B)$ cache misses.

n^2 in C

n^2

s^2

- For the entire C matrix, the number of cache misses = $O(s^2/B)x(n/s)^3 = O(n^3/sB)$.
  - Why?
  - The program has a three way nested loop of n/s iterations in each loop.

Improved Version

- Estimate the number of cache misses again.

- To compute an sxs submatrix of C, we need to bring in $O(s^2)$ elements into the cache.
- Results in $O(s^2/B)$ cache misses.

- For the entire C matrix, the number of cache misses = $O(s^2/B)x(n/s)^3 = O(n^3/sB)$.

good

Cache size

choice of s

- Tall cache Assumption: M = $\Omega(B^2)$, i.e., many more cache lines fit in the cache.

- Number of cache misses = $O(n^3/BM^{1/2})$ at s = B.

Improved Version

- Estimate the number of cache misses again.

- To compute an sxs submatrix of C, we need to bring in $O(s^2)$ elements into the cache.
- Results in $O(s^2/B)$ cache misses.

- For the entire C matrix, the number of cache misses = $O(s^2/B)x(n/s)^3 = O(n^3/sB)$.

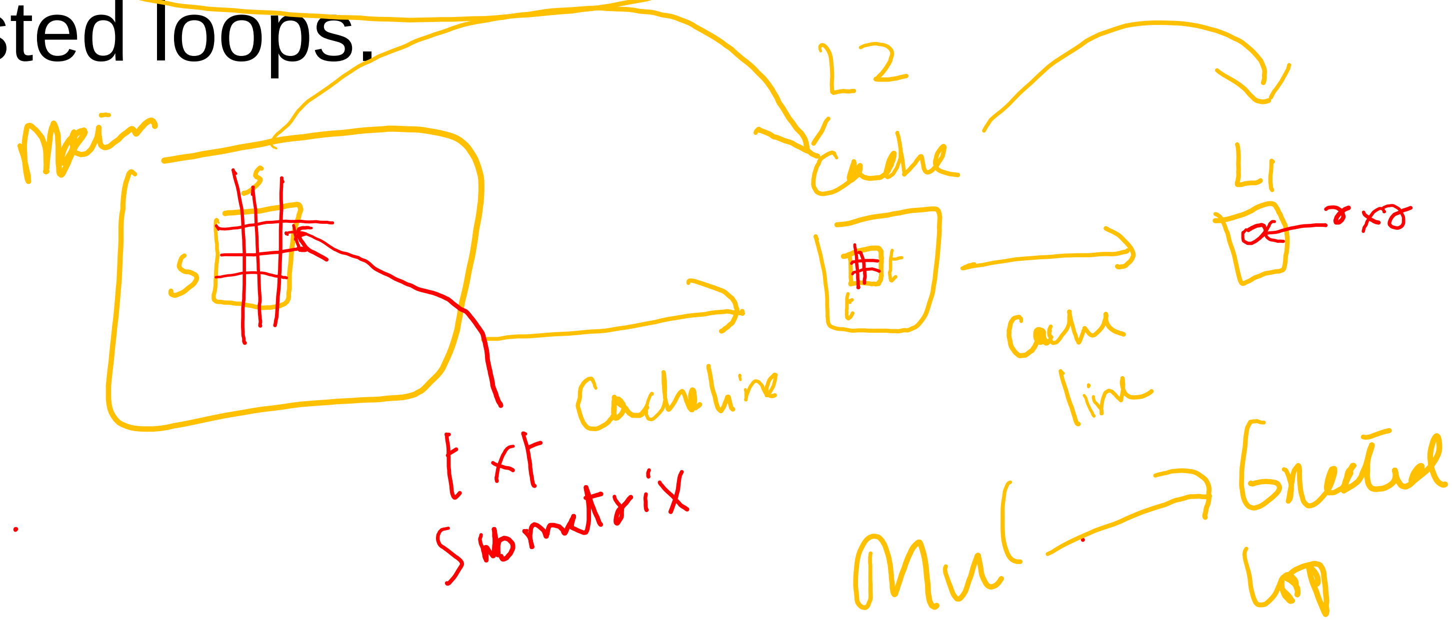
- Tall cache Assumption: $M = \Omega(B^2)$, i.e., many more cache lines fit in the cache.

- Number of cache misses = $O(n^3/BM^{1/2})$ at s = B.

Further Tuning

- May be required if there are more cache levels.
- Each sxs submatrix has to be further blocked for the next level of the memory.
    - Number of such blocking parameters increases with the number of levels in the memory hierarchy.
- Increases the program complexity because of multiple nested loops.

Intel MKL
- subroutine for matrix multiply



Summary

- Blocking is a good technique for making an implementation cache-aware.
- There are some lower bounds too.