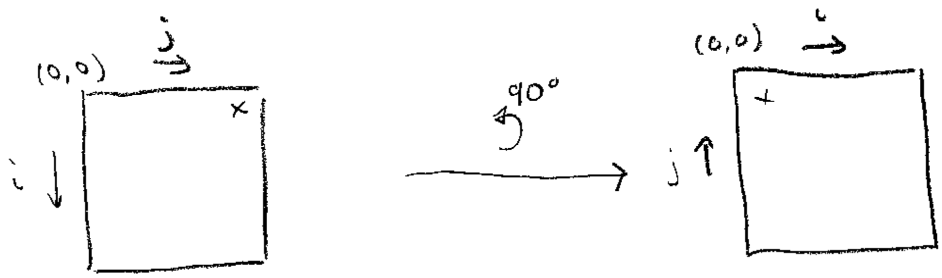
ECE 454S Tutorial – Lab 2 Spring 02 Marek Olszewski

- Warm up students
  - Do they like the class. Tell them that they are lucky.
  - Do they like C, C++, Java? Would they prefer Matlab, H-spi

- Introduce Assn 2.

Want to write efficient code that can rotate an arbitrarily sized image by 90° counter-clockwise.



Two basic operations:
- Transpose: M_{ij} interchanged with M_{ji}
- Exchange rows: Row i is exchanged with row $N-1-i$

Implementation:

```
void naive_rotate(int dim, pixel *src, pixel *dst) {
    int i, j;
    for (i=0; i<dim; i++)
      for (j=0; j<dim; j++)
        dst[RIDX(dim-1-j, i, dim)] =
                  src[RIDX(i, j, dim)];


    return;
}
```

Where RIDX is defined as:

```
#define RIDX(i, j, n) ((i)*(n) + (j)).
```

- Your job is to optimize this code using a number of well known optimizations.

* Note: All these optimizations *could* be performed by a compiler automatically. However, not all compilers are created equal. gcc for example is known to be weak at these kinds of things

+ Furthermore, many of these are very specific to the* architecture that you are running on

+ Therefore, do not think that you will be performing such optimizations on a day to day basis. Doing so would make your code very hard to understand!

+ A good programmer must be aware of what the compiler is doing with his or her code!

+ If the compiler does not optimize your code sufficiently, and it appears to be - both weak, then you will a in - take care of it

eg. I spent a whole summer hand optimi certain C code in the MS .Net frame

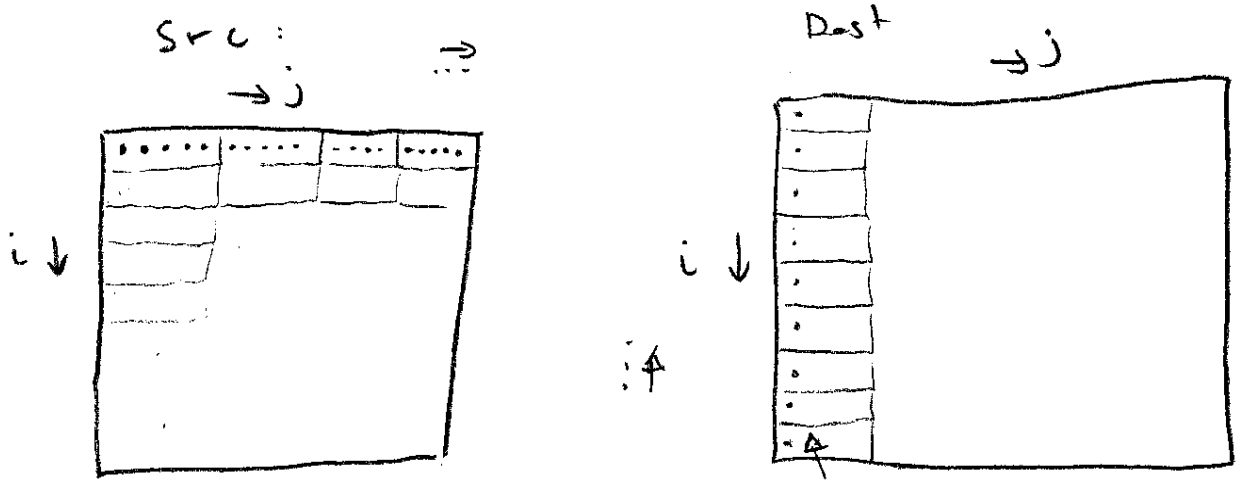
+ I was actually rewriting them in AMD 64 assembly code

+ The code was called so often, th they couldn't afford not to.

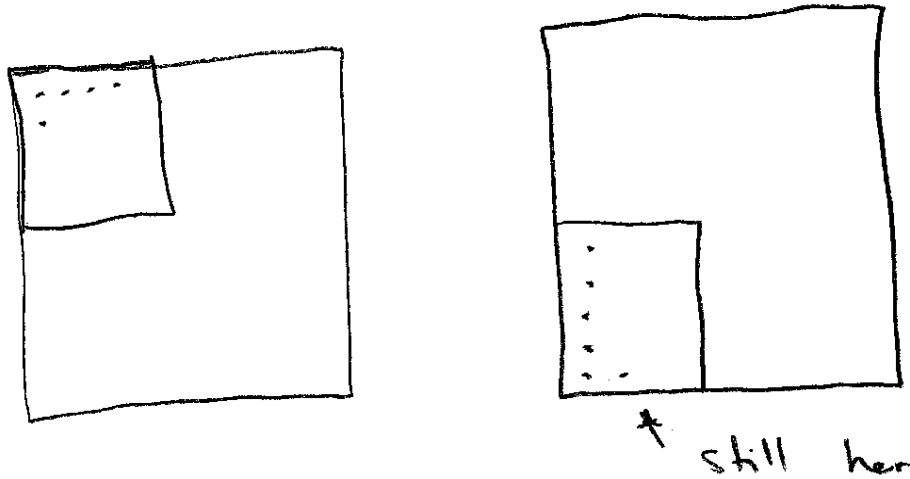
Blocking: Start with this optimization.

We want to improve **temporal locality**.

Lets look at what happens when we run the naive implementation.



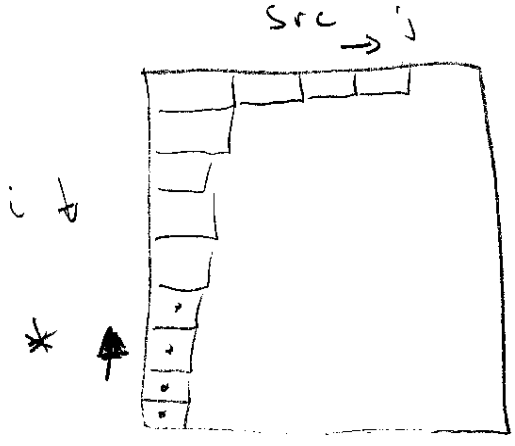
Solution: work on small sub matrices or "blocks" at a time

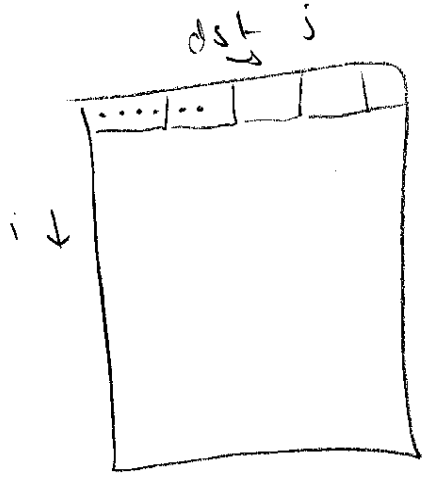


To do this, we have to add two more loops to our code, that iterate through all the blocks in the matrix. eg.

1	2
3	4

Loop reordering: A common optimization is switching the order or nesting of loops to ensure that accesses benefit from spatial locality. In this case we actually can't improve spatial locality. Lets see why.





* start on dst side

Total number of misses stays the same!
But! We do change the number of write and read misses.

Turns out on the Pentium 4 architecture, there one is cheaper.

* This for you to find out!

~~The difference on pentium 4 due to the out-of-order instruction scheduler~~

The difference is due to the way store and load instructions are scheduled by the out-of-order pentium 4 scheduler.