## Optimizing loops for the memory hierarchy

A typical loop walks over lots of data (long arrays)

Enclosing loops lead to repeated traversals of same data

Want to restructure loops to revisit data
while they are in cheap memory

Example: matrix multiply

```
for i := 1 to N
    for j := 1 to M
        for k := 1 to L
            c[i,j] += a[i,k] * b[k,j]
```

`a[i,*]` visited repeatedly, but usually flushed from cache

`b[*,j]` similar, though less extreme

`c[i,j]` not critical: the same element accessed multiple times



## **Blocking** a.k.a. **Tiling**

Restructure loop so that it revisits a chunk that
fits in register/cache/page before moving on

```
for k' := 1 to L by ChunkSize
    for i := 1 to N
        for j := 1 to M
            for k := k' to k'+ChunkSize-1
                c[i,j] += a[i,k] * b[k,j]
```

Implement transformation with combination of
strip mining + loop interchange



## Strip mining

Strip mining: turn single loop into 2 loops

- inner loop up to some chunk size
- outer loop repeats chunks till done

Before:

```
for k := 1 to L
    c[i,j] += a[i,k] * b[k,j]
```

After:

```
for k' := 1 to L by ChunkSize
    for k := k' to k'+ChunkSize-1
        c[i,j] += a[i,k] * b[k,j]
```



## Loop interchange

Apply loop interchange to move outer strip-mined loop outside
of enclosing loop(s)

Before:

```
for i := 1 to N
    for j := 1 to M
        for k' := 1 to L by ChunkSize
            for k := k' to k'+ChunkSize-1
                c[i,j] += a[i,k] * b[k,j]
```

After:

```
for k' := 1 to L by ChunkSize
    for i := 1 to N
        for j := 1 to M
            for k := k' to k'+ChunkSize-1
                c[i,j] += a[i,k] * b[k,j]
```

## Do again

Block the `j` loop, too

```
for k' := 1 to L by ChunkSize
    for j' := 1 to M by ChunkSize
       for i := 1 to N
          for j := j' to j'+ChunkSize-1
             for k := k' to k'+ChunkSize-1
                c[i,j] += a[i,k] * b[k,j]
```