High Performance Computing Systems

- Memory Mountain

Doug Shook via Roger Chamberlain

Writing Cache Friendly Code

- Repeated references to variables are good (locality)
- Example: 4-byte words, 4-word cache blocks

```
int sumarrayrows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Miss Rate: ?

```
int sumarraycols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Miss Rate: ?

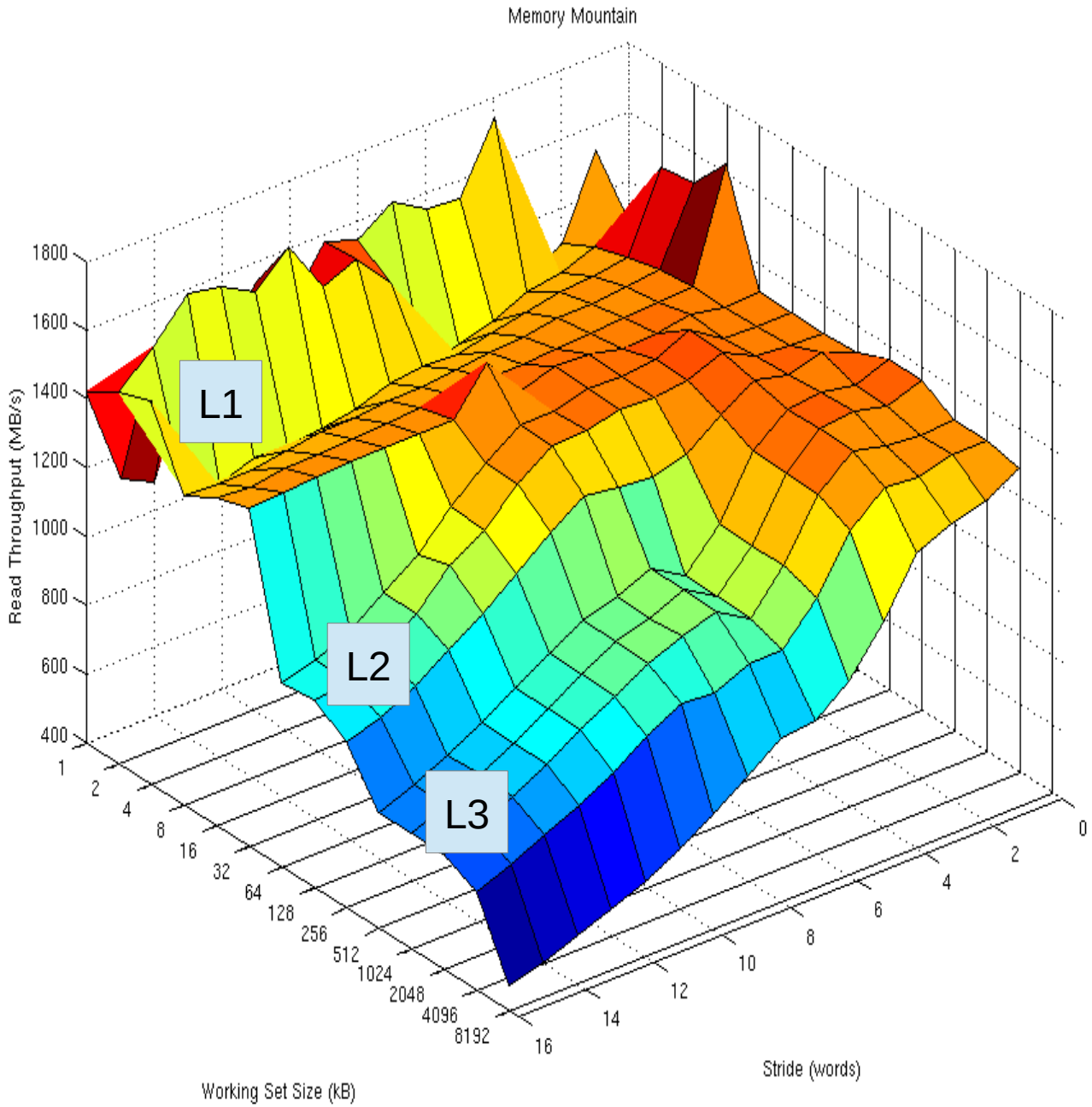
The Memory Mountain

- Read Throughput
  - Number of bytes read from memory per second
- Memory Mountain
  - Read throughput as a function of locality

Memory Mountain Test Function

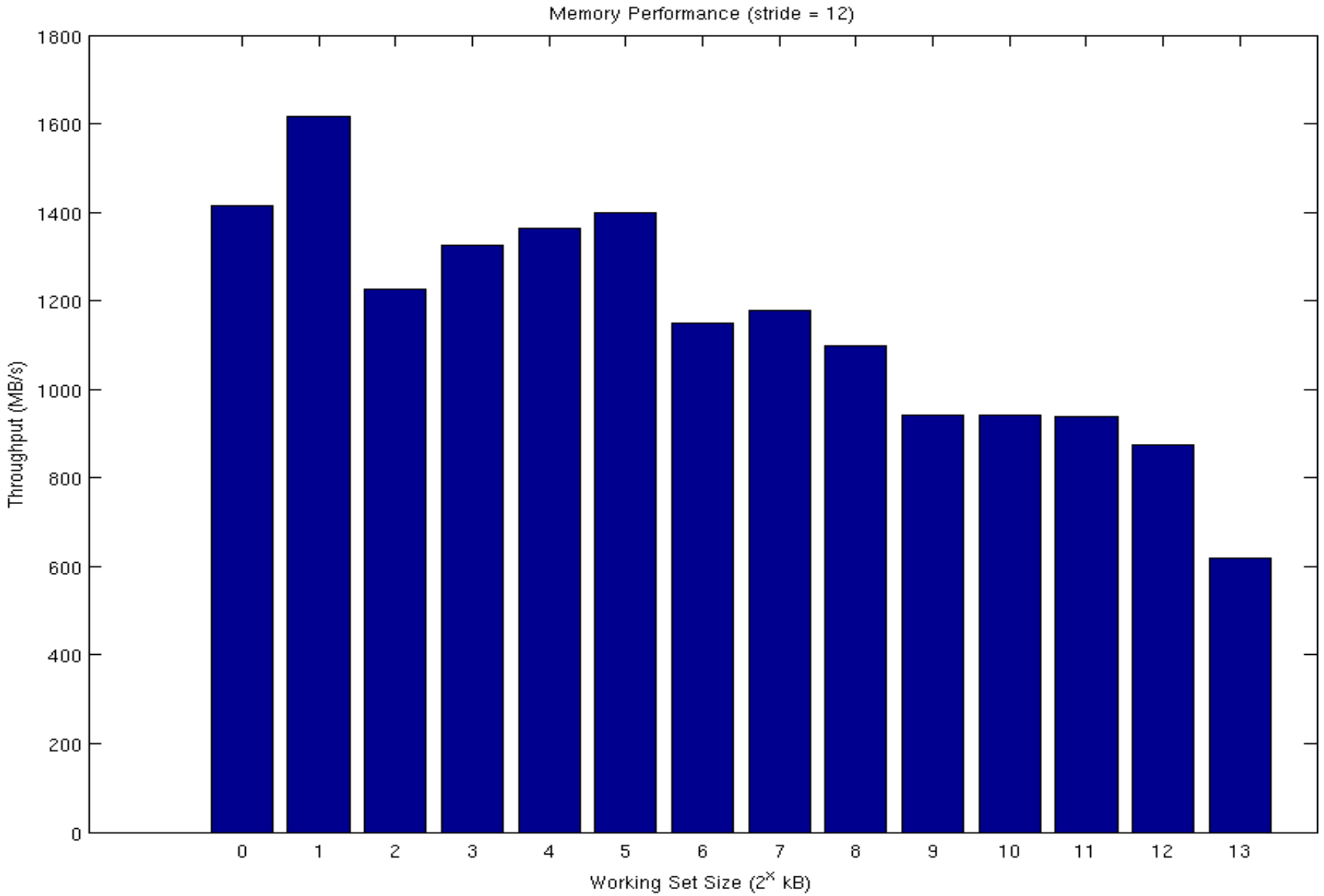
```
/* The test function */
void test(int elems, int stride) {
    int i, result = 0;
    volatile int sink;

    for (i = 0; i < elems; i += stride)
        result += data[i];
    sink = result; /* So compiler doesn't optimize away the loop */
}
```

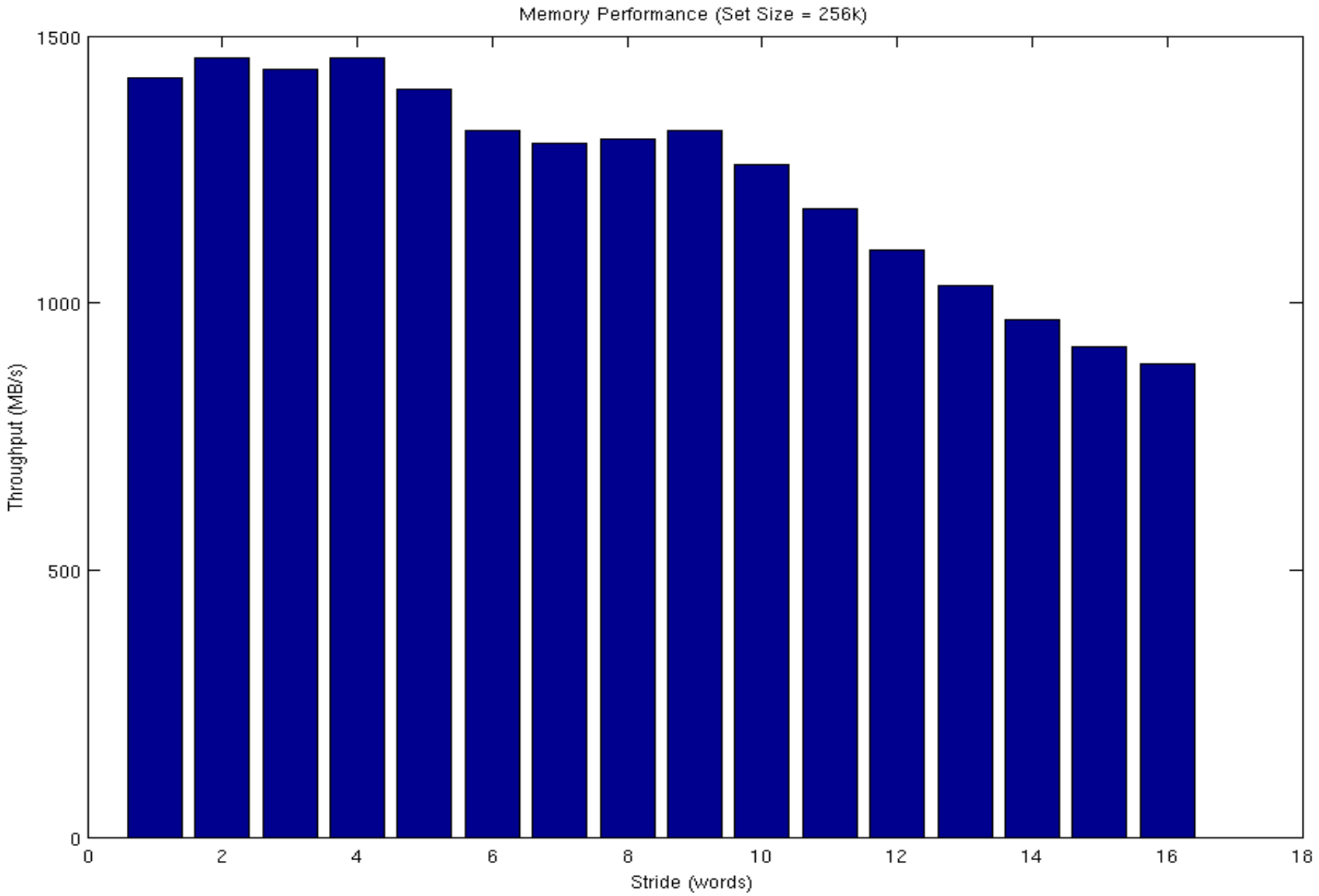


Core i7 Q720
1.6 MHz
32kB L1 d-cache
32kB L1 i-cache
256kB L2 cache
6MB L3 cache

Temporal Locality



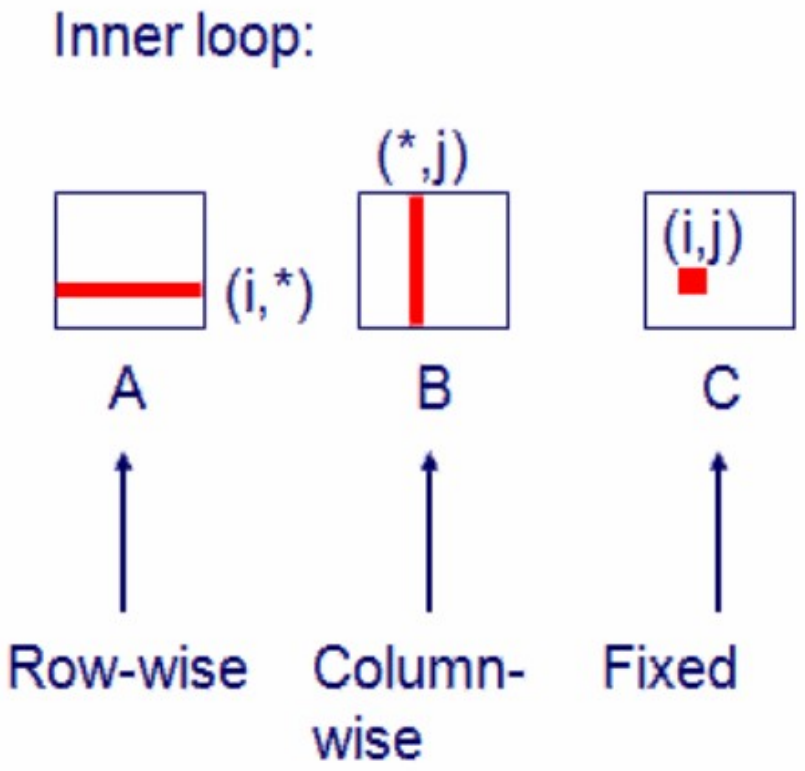
Spatial Locality



Blocking

- Say you are performing a matrix multiplication
  - Traditional method would involve nested loops:

```
/* ijk */
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```



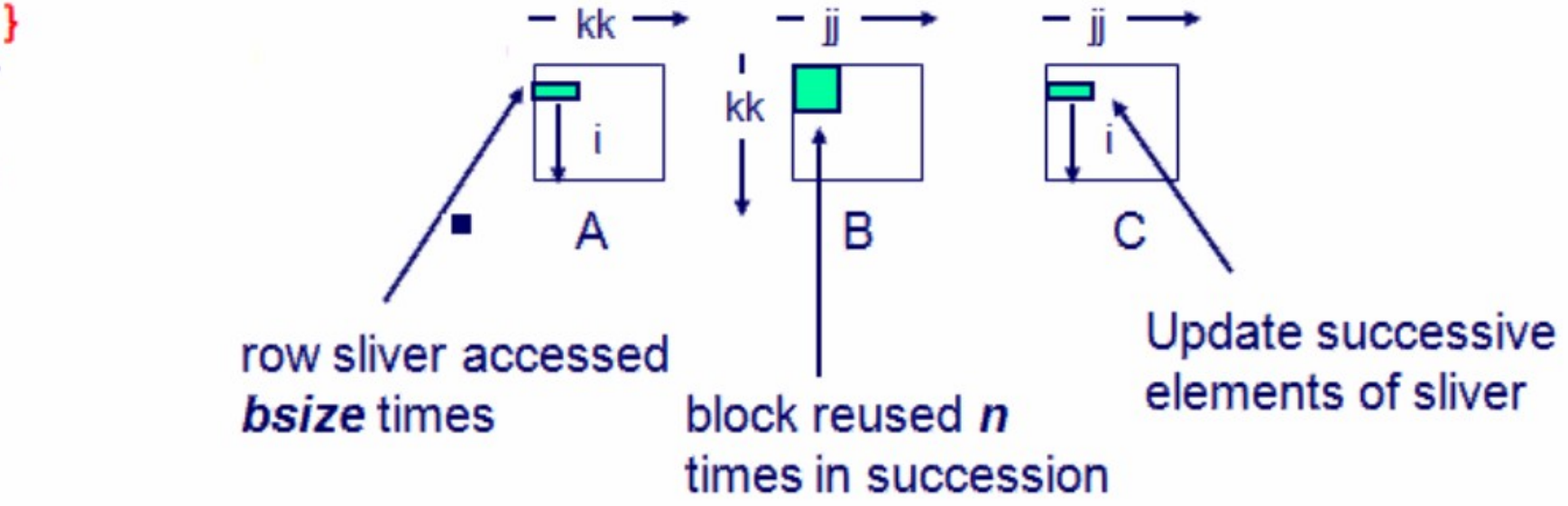
- This method completely ignores ….?

Blocking

- What if we choose to split the work up differently?

```
for (i=0; i<n; i++) {
  for (j=jj; j < min(jj+bsize,n); j++) {
    sum = 0.0
    for (k=kk; k < min(kk+bsize,n); k++) {
      sum += a[i][k] * b[k][j];
    }
    c[i][j] += sum;
  }
```

Innermost Loop Pair



Conclusion

- Programmers can optimize for cache performance

- All systems favor “cache friendly code”