Reuse

Small example:

Cache: - direct mapped (1-way)
- cacheline = 2 doubles
- 2 cachelines

Array $x = x[0], \ldots, x[7]$ accessed twice, cold cache

1.)
```
for j = 0:1
    for i = 0:7
        access(x[i])
```

8 misses, 8 hits
neighbor use ✓
data reuse ↯

access pattern: 0 1 2 ... 7 0 1 2 ... 7

2.)
```
for j = 0:1
    for i = 0:2:7
        access(x[i])
    for i = 1:2:7
        access(x[i])
```

16 misses, 0 hits
neighbor use ↯
data reuse ↯

access pattern: 0 2 4 6 1 3 5 7 0 2 4 6 1 3 5 7

3.)
```
for j = 0:1
    for k = 0:1
        for i = 0:3
            access(x[i+4j])
```

4 misses, 12 hits
(optimal)
neighbor use ✓
data reuse ✓

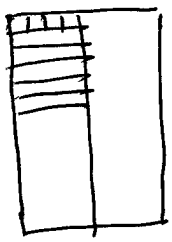
access pattern: 0 1 2 3 0 1 2 3 4 5 6 7 4 5 6 7

Types of reuse

Data reuse: reuse of data brought into cache before it is evicted

Neighbor use: use of data in the same cacheline of the requested data before it is evicted

Accessing a vector



Cache: - 2-way
- cacheline = 4 doubles

Array: x[0], ..., x[n-1] , n >> cache size
accessed once (=> no data reuse)

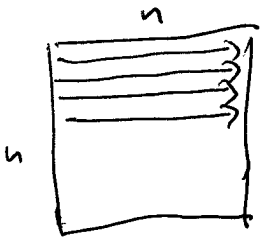
1.) sequential (stride = 1): 0 1 2 ...
- neighbor use, $\frac{n}{4}$ misses

2.) stride = 2: 0 2 4 ... (n-2) 1 3 5 ... (n-1)
- some neighbor use, $\frac{n}{2}$ misses

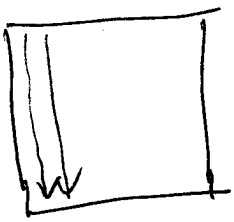
3.) stride ≥ 4: 0 4 8 ... (n-4) 1 5 9 ... (n-3) ...
- no neighbor use, n misses

Where does this occur?

2-D arrays (e.g. images) are stored in row-major order (in C):



Access by column-wise + n divisible by power of 2
=> no neighbor use



Typical pattern in image processing (n = 256, 512, ...)