Reality check on Cache Design

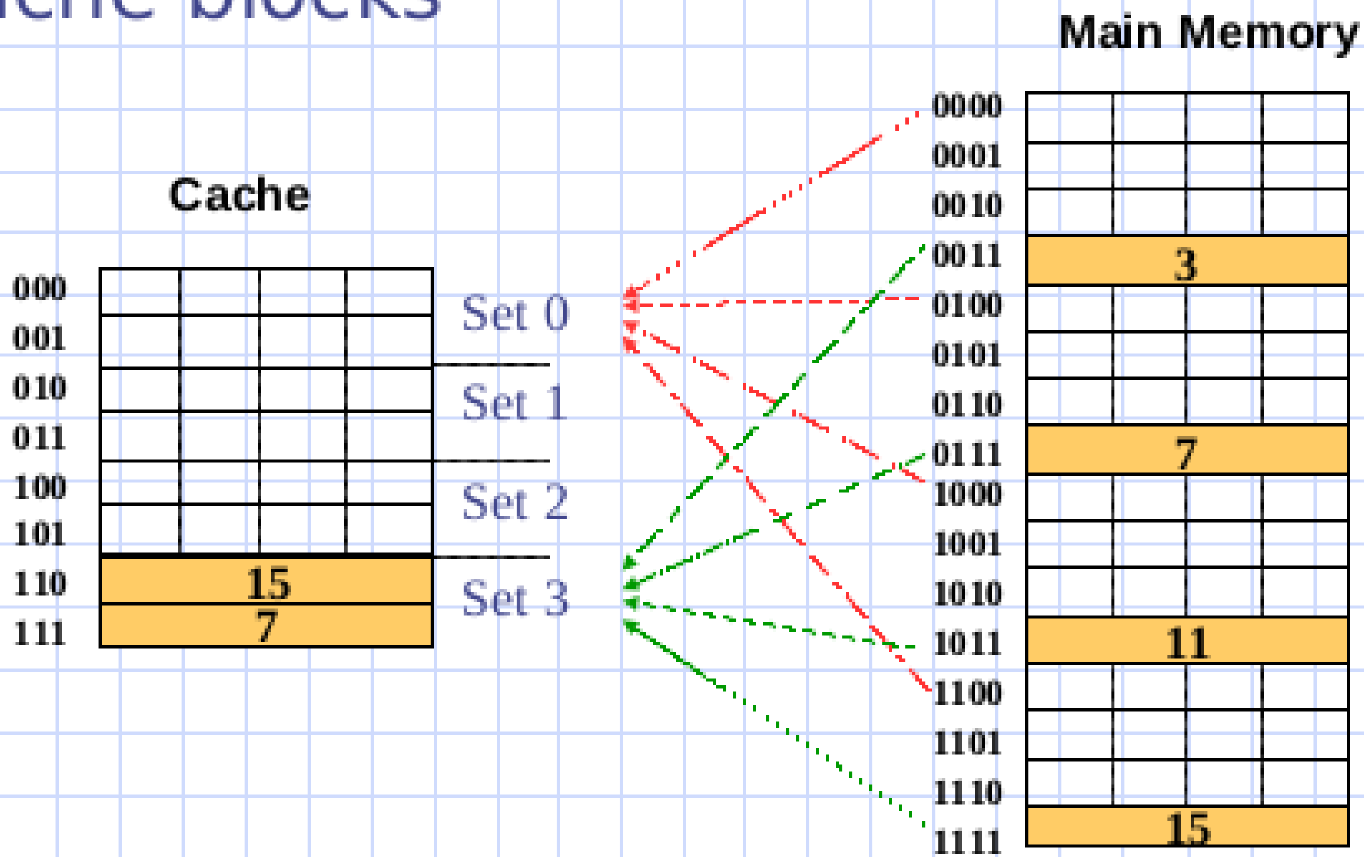
1. Alternatives to direct mapping
2. Enhanching spatial locality

1. Alternative to Direct Mapping

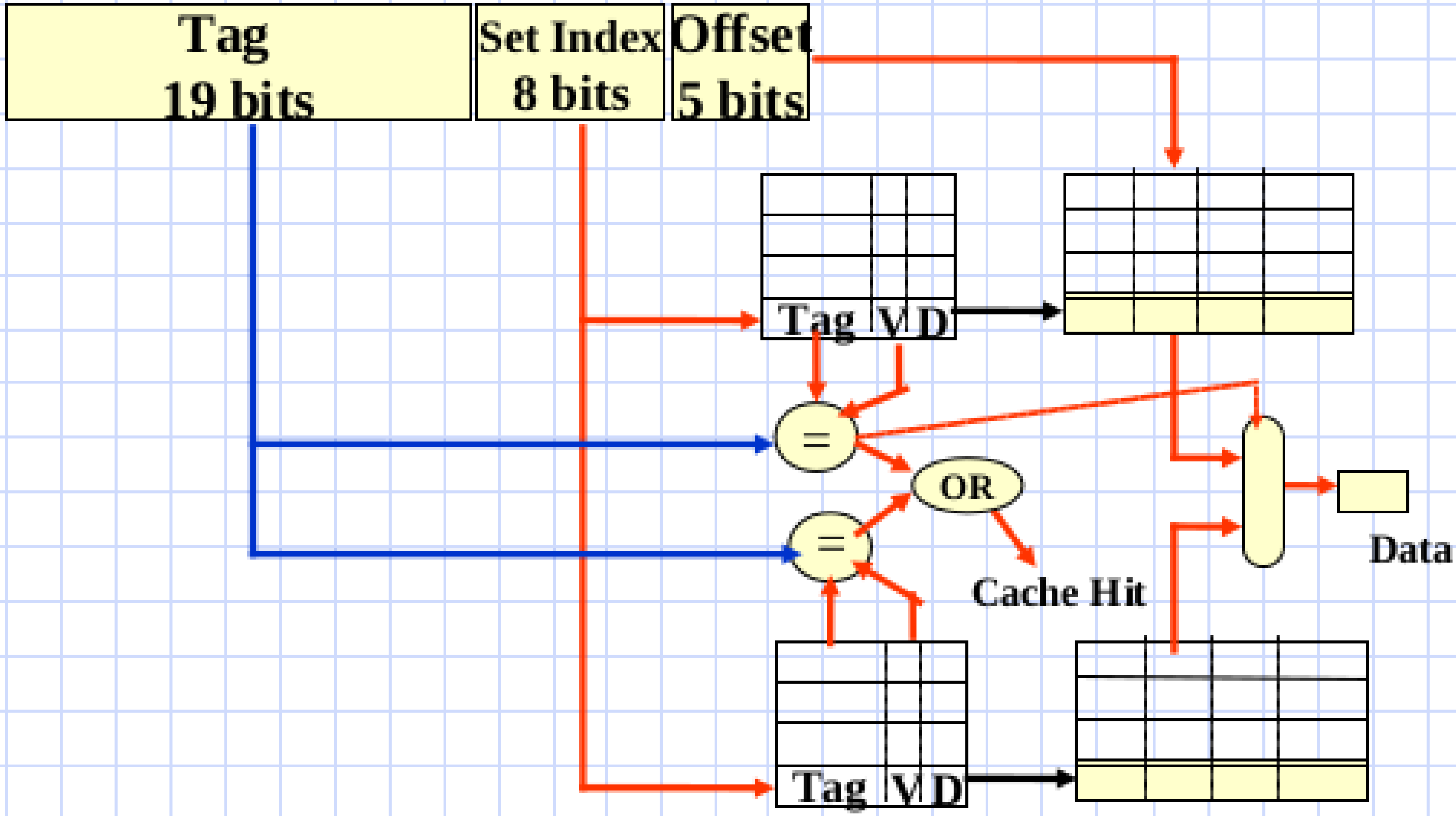
- Why?
  - Conflict misses: Difficult to handle all of them with programming tricks like packing
- Set associative mapping
  - e.g., 2 way set associative
  - Idea: A given memory block can be present in either of 2 blocks of the cache

Set Associative Cache

- A memory block can to be loaded into any cache block within a unique set of cache blocks



e.g., 2-way Set Associative Cache



e.g., 4-way Set Associative Cache

- Assume 16KB cache, 32B block size
- 16KB/32B = 512 blocks
- 512/4 = 128 sets of blocks
- $\log_2 128 = 7$ set index bits
- $\log_2 32 = 5$ offset bits

Tag 20 bits	**Set Index 7 bits**	**Offset 5 bits**

2. Enhancing Spatial Locality

- Recall: Our prefetching loop for Ex 1
- Hardware prefetcher: Hardware that initiates prefetch into the cache of data that might be required by the CPU soon
  - e.g., Many current Intel cores have 4 hardware prefetchers, 2 for L1 data cache and 2 for L2 cache
- So, your programs may experience more hits due to spatial locality than we have calculated

Caches and Programming

- Look critically at the “important parts of your program”
  - “Hot” functions, where a significant part of program execution time is spent
  - e.g., In the LINPACK benchmark, 70% of the time is spent in Daxpy()
  - Hot loops within hot functions
- Hot functions can be identified using tools called profilers
  - e.g., gprof in Linux/UNIX systems
  - With gcc: use -pg flag

Measuring cache performance

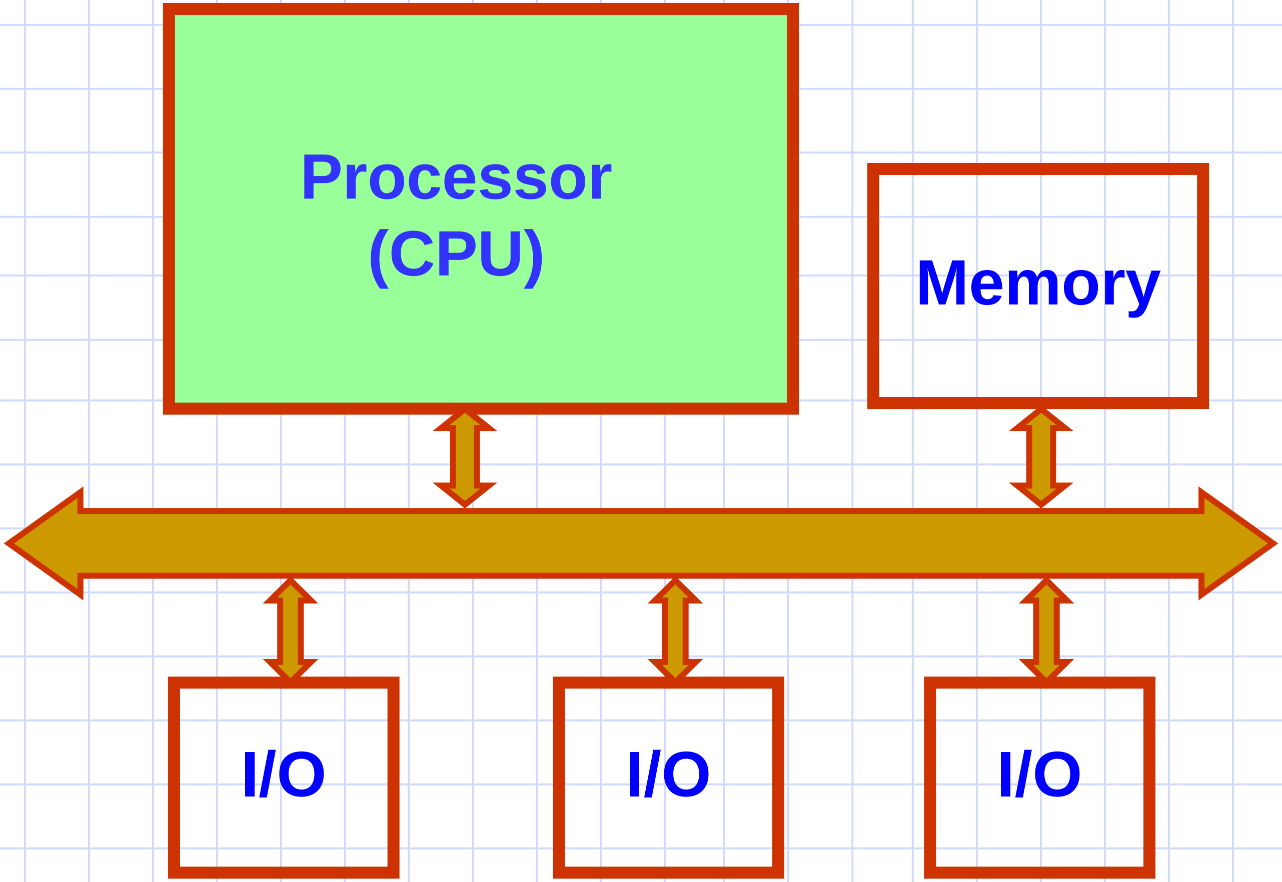
- All modern processors have hardware performance monitoring counters (PMC)
  - They enable counting of numerous “events” during program execution
  - Intel: Separate cycle counter, TSC (time stamp counter), readable with “rdtsc” instruction
  - Other counters require use of privileged instructions

Measuring cache performance

- Related tools
  - VTune amplifier (Intel)
  - PAPI library
  - Perf performance analyzing tool (Linux)
    - You can get privileged access to performance counters on your laptop using “sudo”

```
$ sudo perf stat -e cycles,instructions,cache-references,cache-misses ./a.out
```

Basic Computer Organization



Example: Vector Sum

```
double A[2048], B[2048], C[2048];
for (i=0; i<2048, i++) C[i] = A[i] + B[i];
```

- What if a CPU has 4 adders?
- It can be designed to support an instruction to do 4 iterations of the Vector Sum loop at a time

```
VADD v1_A[0:3], v2_B[0:3], v3_C[0:3]
```

- Called a vector instruction

Multimedia Extensions

- Hardware support for operations on "short vectors" is provided in existing microprocessors
- Example: 256 bit registers, each split into 4x64b (or 8x32b)
  - Maximum vector length
- Example: Intel "x86" processors
  - SSE (Streaming SIMD Extension)
  - AVX (Advanced Vector Extension)

Vectorization of Loops

We will use a generic notation

Instead of

VADD C[0:3], A[0:3], B[0:3]

C[0:3] = A[0:3] + B[0:3]

An example of vectorization

- Given maximum vector length, VL

```
for (i=0; i < N; i++)
    A[i] = A[i] + B[i];
for (i=0; i < N; i+=VL)
    A[i:i+VL-1] = A[i:i+VL-1] + B[i:i+VL-1];
```

What if N is not divisible by VL?

```
for (i=0; i < (N – N%VL); i+=VL)
    A[i:i+VL-1] = A[i:i+VL-1] + B[i:i+VL-1];
for (; i<N; i++) A[i] = A[i] + B[i];
```

- This technique is called Stripmining

Auto-vectorization with gcc

- Auto-vectorization compiler feature
  - Compiler will analyze loops of your program and try to use vector instructions
- e.g., gcc command line options
  - -ftree-vectorize for autovectorization
  - -fopt-info-vec to get feedback on autovectorization (which loops were vectorized, etc)
  - Note: These require optimization level of at least -O2

Auto-vectorization with gcc

```
gcc -O2 -ftree-vectorize -fopt-info-vec prog.c
```

```
prog.c:18:1 note: loop vectorized
```

Possible complications

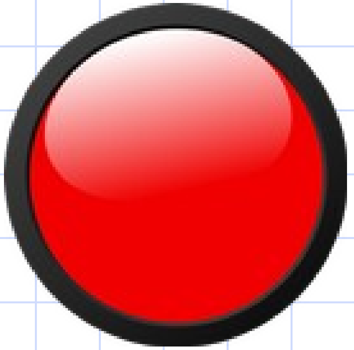
- Dependences between statements within the loop

Example 1

```
for (i=0; i < N; i++) {
    A[i] = B[i] + C[i];
    D[i] = (A[i] + A[i+1])/2;
}
for (i=0; i < (N – N%VL); i+=VL){
    A[i:i+VL-1] = B[i:i+VL-1] + C[i:i+VL-1];
    D[i: ... will get wrong value of A[i+1], etc
```

Example 1

```
for (i=0; i < N; i++) {
    A[i] = B[i] + C[i];
    D[i] = (A[i] + A[i+1])/2;
 }
```

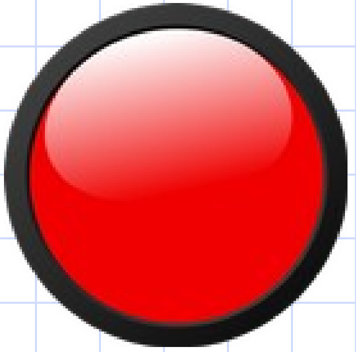


```
for (i=0; i < N; i++) {
    temp[i] = A[i+1];
    A[i] = B[i] + C[i];
    D[i] = (A[i] + temp[i])/2;
}
```

- This loop transformation, through copying of data, is called Node Splitting

Example 2

```
for (i=0; i < N; i++) {
     X = A[i] + 1;
     B[i] = X + C[i];
 }
```

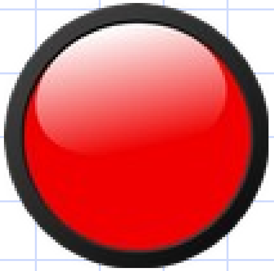


```
for (i=0; i < N; i++) {
     temp[i] = A[i] + 1;
     B[i] = temp[i] + C[i];
}
```

- Scalar expansion

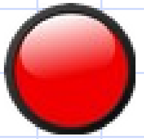
Example 3

```
for (i=0; i < N; i++) {
    A[i] = B[i];
    C[i] = C[i-1] + 1;
 }
```



```
for (i=0; i < N; i++) A[i] = B[i];
```

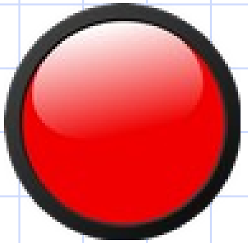
```
for (i=0; i<N; i++) C[i] = C[i-1] +1;
```



- Loop fission

Example 4

```
for (j=1; i < N; j++)
    for (i=2; i < N; i++)
        A[i,j] = A[i-1, j] + B[i];
```



```
for (i=2; i < N; i++)
    for (j=1; j<N; j++)
        A[i,j] = A[i-1,j] + B[i];
```

- Loop interchange