Lecture 21: Memory Allocation Implementation

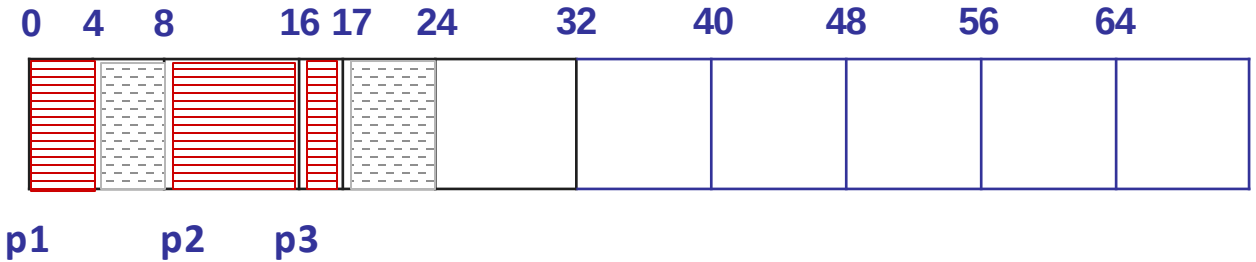
CSE 29: Systems Programming and Software Tools

Aaron Schulman (Shalev)

UCSDCSE

Computer Science and Engineering

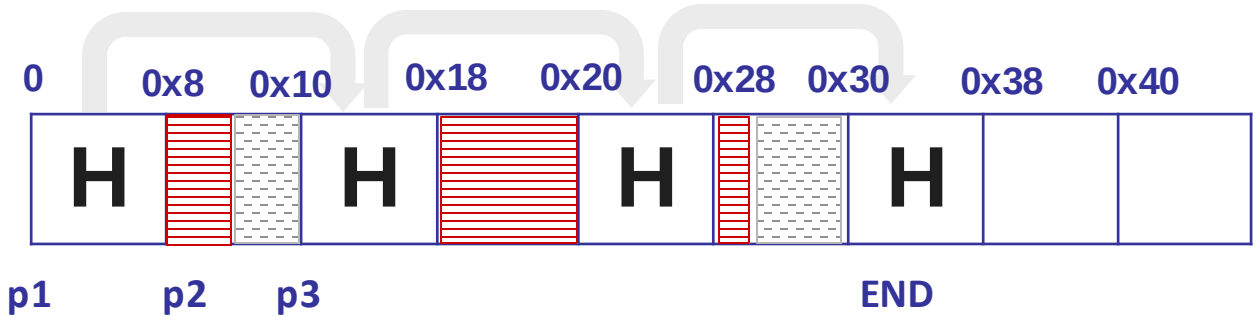
Malloc Implementation: Heap structure




```
int* p1 = malloc(4);
int* p2 = malloc(8);
int* p3 = malloc(1);
```

Alignment Requirement for 64-bit machine:
8 bytes (every heap allocation should be aligned to 8 bytes)

How do we use the heap itself to find free area in the heap?



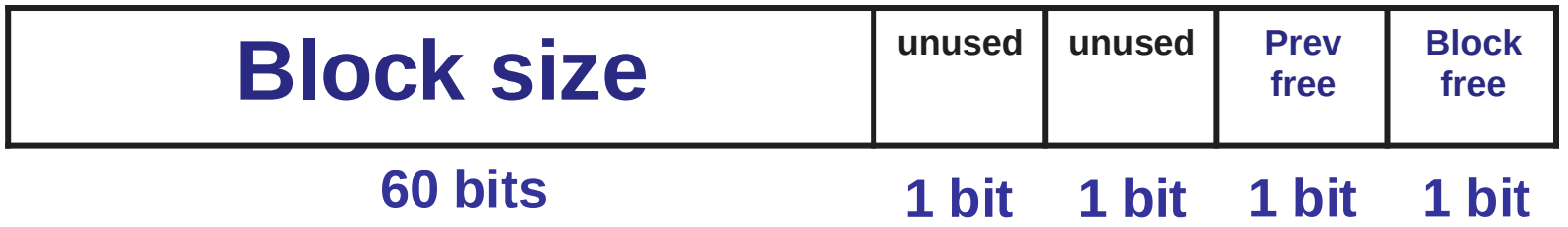

```
int* p1 = malloc(4); // 0x08
int* p2 = malloc(8); // 0x18
int* p3 = malloc(1); // 0x28
```

H = Heap header

Block header struct

```
struct block_header {
    /**
     * The size of a block is always a multiple of 16. This means that
     * the last four bits are always zero. We use the last two bits of
     * this variable to store other information.
     *
     * LSB: Least Significant Bit (Last Bit)
     * SLB: Second to Last Bit
     *
     * LSB = 0 <=> This block is free
     * LSB = 1 <=> This block is allocated
     * SLB = 0 <=> Previous block is free
     * SLB = 1 <=> Previous block is allocated
     *
     * When used as End Mark:
     *   size_status should be 1 (i.e. zero sized busy block). see VM_ENDMARK
     * macro.
     *
     * When we want to read the size, we should ignore the last two bits.
     */
    size_t size_status;
};
```

Header structure



Alignment Requirement (because header + block):
16 bytes (every heap allocation should be aligned to 16 bytes)

Special cases for

- Special cases of block size have a particular meaning
  - Block size **0** and free is **1** means this block is the end of the heap
  - Prev free is **1** and free is **0** means there is some heap fragmentation