- Referent object approach
- Similar to Jones & Kelly with a different data structure

**Invariant:** Pointers are always in bounds
⇒ Find bounds by looking up the enclosing interval

**BBC Representation**

**Tricks:**
- Region of size \( l \) is aligned to \( 2^{\log_2 l} \) bytes
- Every allocation is a power of 2 ⇒ Easy alignment
  E.g. – If the allocation is aligned to 128 bytes, address of start region is divisible by 128
- To store bounds, only need \( \log_2 \) of allocation size

“Aligned to \( 2^l \) bits” ----> “l LSBs are 0”

![Diagram of low and hi pointers](image)

**Fragmentation**

If an object requires 33 bytes and the slot size is 64 bytes, we allocate 64 bytes for the object.

![Fragmentation Diagram](image)

- Could waste up to half of RAM
- Unused memory (marked by X in the figure above) is considered allocated memory but system can’t use it
Storing & looking up \( l \)

- Map \( p \to l \)
- Implemented as a special purpose hash table

### Bytes \( L \left[ 2^{32}/\text{slot\_size} \right] \)

\[
l = L \left[ \frac{p}{\text{slot\_size}} \right]
\]

Slot size = 16

Looking up \( l \), given \( p \)

#### Updating \( l \)

\[
p = \text{malloc}(16);
\]

\[
L \left[ \frac{p}{\text{slot\_size}} \right] = 4;
\]

\[
p = \text{malloc}(32);
\]

for \( i = 0; i < 32/\text{slot\_size}; i++ \)

\[
L \left[ \left( \frac{p}{\text{slot\_size}} + i \right) \right] = 5;
\]

### Bounds checks:

\[
q = p + i;
\]

assert(((\( p \land q \)) >> L[p/\text{slot\_size}]) == 0);

Here we care only the MSBs of the pointers \( p \) and \( q \).

If they are the same, the assert succeeds & \( q \) is within bounds.

Before dereferencing \( q \), we have to perform the following assert

\[
\text{assert}(((q \land (q + \text{sizeof}(*q) - 1)) >> L[q/\text{slot\_size}]) == 0);
\]

\[
*q = 0;
\]

### OOB Pointers

When a pointer is out of bounds, set \( p \) to point to inaccessible region of the memory.

Set \( q = q | 0x80000000 \)

Virtual Memory

4 GB

Inaccessible (1xxxxxxx)

2 GB
Entire upper 2GB of VM is system memory reserved for OS (Windows).

In Linux, upper 1GB is reserved for system memory.

If the assert fails i.e. q is not within bounds, we set it to point to inaccessible memory:

\[ q = q | 0x80000000 \]

if (q & 0x80000000)
    \[ q = q | 0x7fffffff \]
    // Recover the original pointer
    // by setting the MSB to 0

if(q % 16 < 8) // if (q % slot_size < slot_size/2)
    l = L [(q[slot_size]) - 1]
else
    \[ l = L [(q[slot_size]) + 1] \]

We can change the assert to check if the pointer is more than half slot size out of bound and abort the program if it is.

\[ \text{If}((q^p) >> \text{L}[p/slot\_size] \neq 0) \]
\[ q = \text{oob}(p, q); \]

\text{void} *\text{oob}(\text{void} *p, \text{void} *q) \{
q = q | 0x7fffffff;
lo = \text{lookup\_lo}(p);
hi = \text{lookup\_hi}(p);
\text{if}(q >= lo \&\& q <= hi) // q is either in lower half of top
    \text{return} q; // slot or upper half of bottom slot
\text{if}((q > lo - (slot\_size/2) \&\& q < lo) ||
    (q < hi + (slot\_size/2) \&\& q >= hi))
    \{
q = q | 0x80000000;
    \text{return} q;
\}
\}
\text{abort}();