Memory Safety

- Store by pointer
- Store by region
  - Soft bound (hash table)
  - CQuare approach

```c
Struct list {
    Struct list * next;
    Bounds b_next;
    Int * j; }
```

CQUAR ->

```c
Struct list_shadow {
    Bounds b_next;
    Struct list_shadow * next;}
```

**Latest technology**

We will keep a shadow stack that checks the bounds for memory

- Every time original program allocates the list the modified program has same code with the bounds

Original list

- We always make a shadow of the struct with bounds as payload

```
<table>
<thead>
<tr>
<th></th>
<th>next</th>
<th>next</th>
<th>next</th>
<th>null</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>k</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- we update the pointer and stuff it points to for both the program list and shadow list

Problem: library can insert into data stricter and shadow won’t match with the program
The other side is storing with the pointer value

Junes & Kelly

- Pointer bound are associated w/ regions
  - \texttt{Bounds\_lookup(p);} // not address of p, but p itself
  - \texttt{Int * p = malloc (n);};
  - We put a check here to make sure p++ wont step in another memory
    - P++
    - \texttt{Bounds\_store (p,p+n);};
    - \texttt{Bounds t = bounds\_lookup(p);};
    - \texttt{B\_check(t,p,p+1)}
    - *\texttt{p} = 0

How do we store the address?

- We use internal tree
  - Look up and updates are Olog(n)
    - Lookup on increment and decrement so pointer don’t step on others
- Problems:
  - Moving out of memory space temporary and back not accepted
    - P+n
    - Check fail
    - P-n
  - Internal tree are too slow
    - Overhead 10x to 100x