Deallocation Models:

Explicit: (C++/C++)
- programmer deletes
- may result in leaks, if not careful

Implicit: (Java, Python)
- runtime system deletes
- Garbage collection
- Slower runtime
- Better memory compaction

What happens when you do
- new (Java)
- malloc/free (C)
- new/delete (C++)?

Runtime System Mem. Mgr.
- Stack - local vars, recursion
- Heap - for "new" objects

Guide:
prevInUse: 1 if prev. contig. block is allocated
prev/next: links in avail list
size/size2: total block size (includes headers)

How to select from available blocks?
-First-fit: Take first block from avail. list that is large enough
-Best fit: Find closest fit from avail list

Surprise: First-fit is usually better -faster & avoids small fragments

Block Structure:

Available:

Explicit Allocation/Deallocation
- Heap memory is split into blocks whenever requests made
- Available blocks:
  - merged when contiguous
  - stored in available block list

Memory Management

Fragmentation:
- Results from repeated allocation + deallocation
(Swiss-cheese effect)

Guide:
prevInUse: 1 if prev. contig. block is allocated
prev/next: links in avail list
size/size2: total block size (includes headers)

Available:

How to select from available blocks?
-First-fit: Take first block from avail. list that is large enough
-Best fit: Find closest fit from avail list

Surprise: First-fit is usually better -faster & avoids small fragments

Memory Management
Example: Alloc b=59

**Allocation:**
- `malloc(b)`
- Search avail. list for block of size `b' ≥ b+1`
- If `b'` close to `b`: alloc entire block (unlink from avail list)
- Else: split block

```
(b)
```

```
dealoc
```

**Deallocation:**
- If prev+next contiguous blocks are allocated → add this to avail
- Else: merge with either/both to make max. avail block

```
Example:
```

Some C-style pointer notation:
- `void*` - pointer to generic word of memory
- `p+10` - 10 words beyond `p`
- `*(p+10)` - contents of this word
- `p` - point to head of block:
  - `p->inuse`, `p->prevInUse`, `p->size`
  - We omit bit manipulation
  - `*(p+p->size-1)` - references last word in this block

```
(continues)
```
Buddy System:
- Block sizes (including headers) are power of 2
- Requests are rounded up (internal fragmentation)
- Block size $2^k$ starts at address that is multiple of $2^k$
- $k =$ level of a block

Structure:

In practice: There is a minimum allowed block size

Buddy system only allows allocations aligning with these blocks

Coping with External Fragmentation
- Unstructured allocation can result in severe external fragmentation
- Can we compress? Problem of pointers
- By adding more structure we can reduce extern frag. at cost of internal frag.

Memory Management

 allocation: $\text{alloc}(b)$
- $k = \lceil \log (b+1) \rceil$ add 1 for header
- if $\text{avail}[k]$ non-empty
  return entry + delete
- else: find $\text{avail}[j] \neq \emptyset$
  for $j > k$
  - split this block

Big Picture:
- $\text{Avail list}$ is organized by level: $\text{avail}[k]$
- $\text{Block header}$ structure same as before except:
  - `prevInUse` NOT needed

Def: $\text{buddy}_k(x) = \begin{cases} 
  x + 2^k & \text{if } 2^k \text{ divides } x \\
  x - 2^k & \text{otherwise}
\end{cases}$

$\text{buddy}_k(x) = (1 \ll k) \oplus x$ [Bit manipulation]
Announcements:

① Prog assign 3 - Due, Mon, May 9, 11:59 pm
   - Optional Prog Assign 0+ln+1b \{drop 2 lowest\}

② HW 4 - Due, Tue, May 10, 11:00 am
   - Optional - Drop lowest HW
   - HW grades - more variation
   - Questions on final exam - mod from HW4

③ Final Exam
   - Fri, May 13, 4-6 pm, Tydings Bldg 0130
Example: \( \text{alloc}(2) \xrightarrow{\text{round up}} \text{alloc}(4) \)