Virtual Memory:
malloc: Measuring performances

Performance Goals

- **Given some sequence of `malloc` and `free` requests:**
  - $R_0, R_1, ..., R_k, ..., R_{n-1}$

- **Goals: maximize throughput and peak memory utilization**
  - These goals are often conflicting

Performance Goal: Throughput

- **Given some sequence of `malloc` and `free` requests:**
  - $R_0, R_1, ..., R_k, ..., R_{n-1}$

Performance Goal: Throughput

- **Given some sequence of `malloc` and `free` requests:**
  - $R_0, R_1, ..., R_k, ..., R_{n-1}$

- **Throughput:**
  - Number of completed requests per unit time (n / *time-between-*R_0,R_{n-1})
  - Example:
    - 5,000 **`malloc`** calls and 5,000 **`free`** calls in 10 seconds
    - Throughput is 1,000 operations/second

Performance Goal: Peak Memory Utilization

- **Given some sequence of `malloc` and `free` requests:**
  - $R_0, R_1, ..., R_k, ..., R_{n-1}$

Performance Goal: Peak Memory Utilization

- **Given some sequence of `malloc` and `free` requests:**
  - $R_0, R_1, ..., R_k, ..., R_{n-1}$
- ***Def:* Aggregate payload P_k**
  - **`malloc(p)`** results in a block with a ***payload*** of **`p`** bytes
  - After request R_k has completed, the ***aggregate payload*** P_k is the sum of currently allocated payloads

Performance Goal: Peak Memory Utilization

- **Given some sequence of `malloc` and `free` requests:**
  - $R_0, R_1, ..., R_k, ..., R_{n-1}$
- ***Def:* Aggregate payload P_k**
  - **`malloc(p)`** results in a block with a ***payload*** of **`p`** bytes
  - After request R_k has completed, the ***aggregate payload*** P_k is the sum of currently allocated payloads
- ***Def:* Current heap size H_k**
  - Assume H_k is monotonically nondecreasing
    - i.e., heap only grows when allocator uses **`sbrk`** (increments **`brk`**)

Performance Goal: Peak Memory Utilization

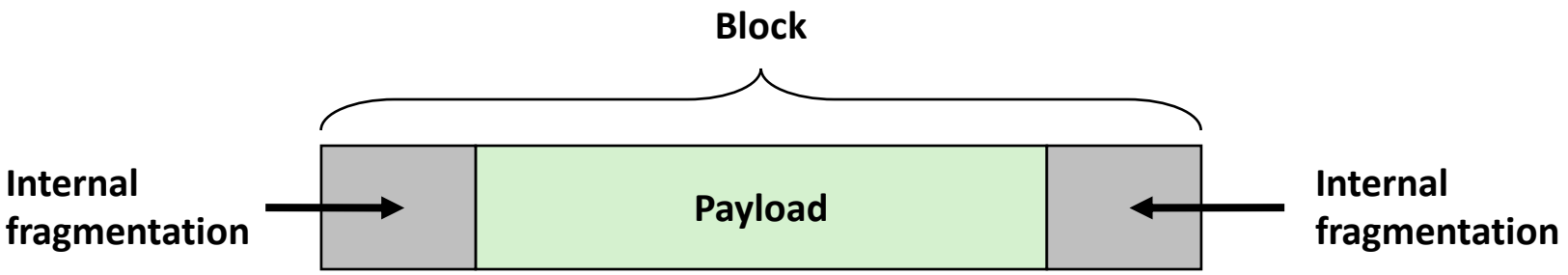
- **Given some sequence of `malloc` and `free` requests:**
  - $R_0, R_1, ..., R_k, ..., R_{n-1}$
- ***Def:* Aggregate payload P_k**
  - **`malloc(p)`** results in a block with a ***payload*** of **`p`** bytes
  - After request R_k has completed, the ***aggregate payload*** P_k is the sum of currently allocated payloads
- ***Def:* Current heap size H_k**
  - Assume H_k is monotonically nondecreasing
    - i.e., heap only grows when allocator uses **`sbrk`** (increments **`brk`**)
- ***Def:* Peak memory utilization after k+1 requests**
  - $U_k = (max_{i<=k} P_i) \ / \ H_k$

Fragmentation

- **Poor memory utilization caused by *fragmentation***
  - ***internal*** fragmentation
  - ***external*** fragmentation

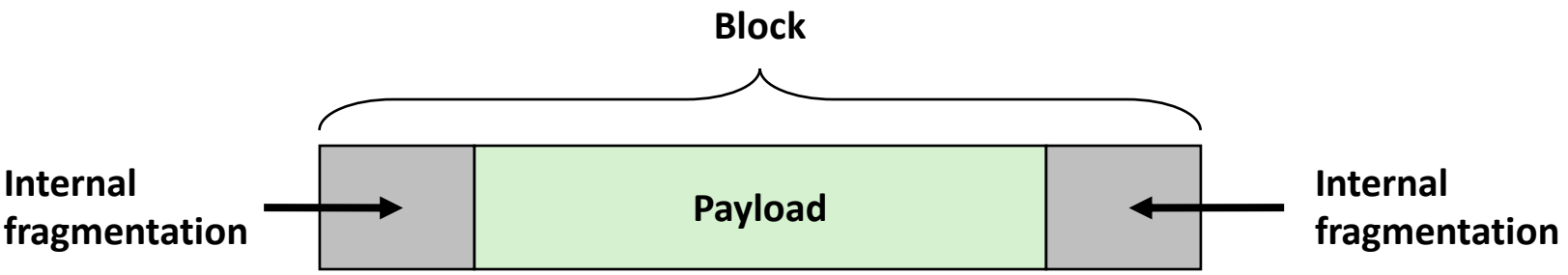
Internal Fragmentation

- For a given block, *internal fragmentation* occurs if payload is smaller than block size



Internal Fragmentation

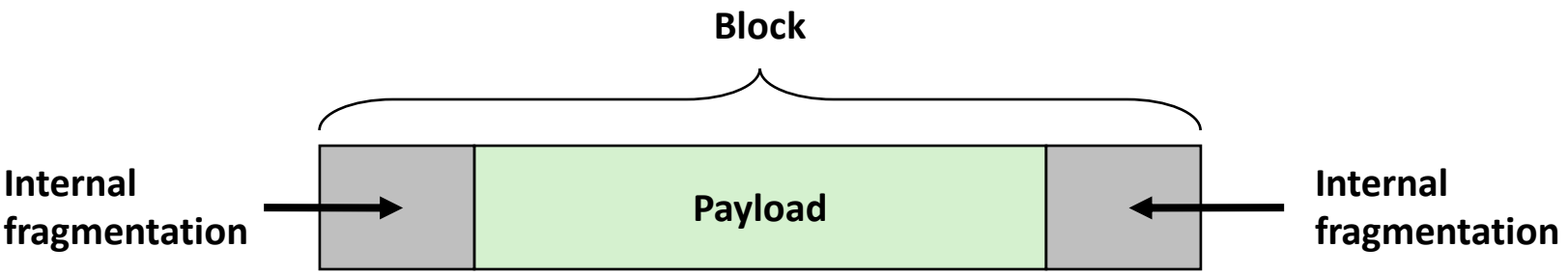
- **For a given block, *internal fragmentation* occurs if payload is smaller than block size**



- **Caused by**
  - Overhead of maintaining heap data structures
  - Padding for alignment purposes
  - Explicit policy decisions
    (e.g., to return a big block to satisfy a small request)

Internal Fragmentation

- **For a given block, *internal fragmentation* occurs if payload is smaller than block size**



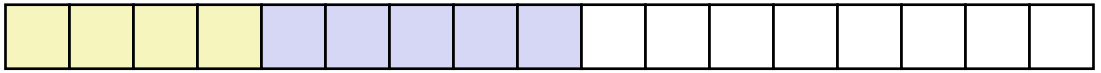
- **Caused by**
  - Overhead of maintaining heap data structures
  - Padding for alignment purposes
  - Explicit policy decisions
    (e.g., to return a big block to satisfy a small request)
- **Depends only on the pattern of *previous* requests**
  - Thus, easy to measure

External Fragmentation

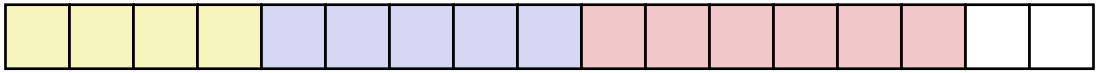
- Occurs when there is enough aggregate heap memory, but no single free block is large enough

```
p1 = malloc(4)
```

```
p2 = malloc(5)
```



```
p3 = malloc(6)
```



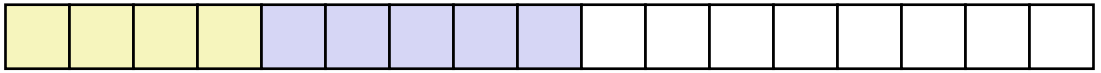
```
free(p2)
```

External Fragmentation

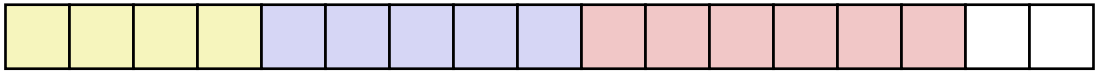
- Occurs when there is enough aggregate heap memory, but no single free block is large enough

```
p1 = malloc(4)
```

```
p2 = malloc(5)
```



```
p3 = malloc(6)
```



```
free(p2)
```

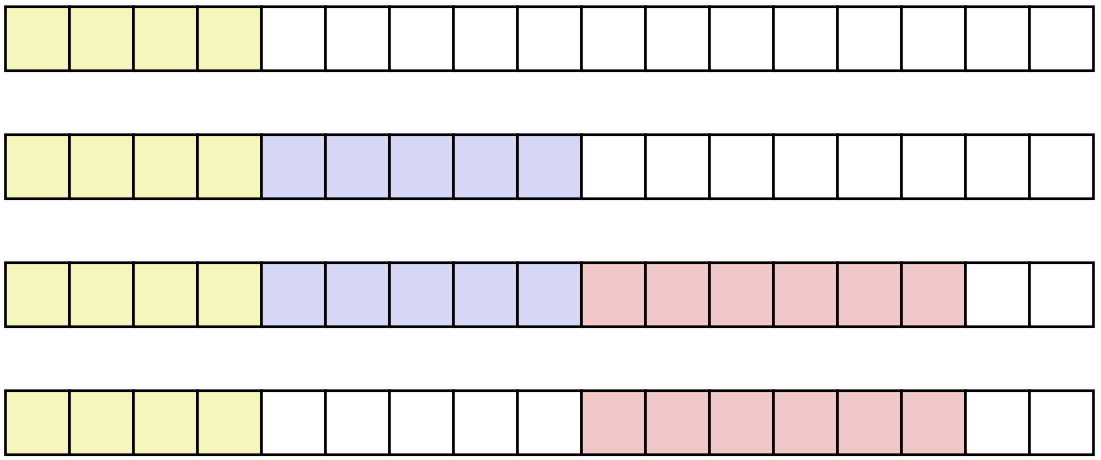
```
p4 = malloc(6)
```

Oops! (what would happen now?)

External Fragmentation

- Occurs when there is enough aggregate heap memory, but no single free block is large enough

```
p1 = malloc(4)
p2 = malloc(5)
p3 = malloc(6)
free(p2)
p4 = malloc(6)
```



Oops! (what would happen now?)

- Depends on the pattern of future requests
  - Thus, difficult to measure