Process

| **Process** | **Process Size** |
| --- | --- |
| Process 1 | 90 |
| Process 2 | 50 |
| Process 3 | 30 |
| Process 4 | 40 |

Let the free space memory allocation blocks be:

Blocks

| **Block** | **Block Size** |
| --- | --- |
| Block 1 | 20 |
| Block 2 | 100 |
| Block 3 | 40 |
| Block 4 | 200 |
| Block 5 | 10 |

Explanation

Process P1

* All Blocks 1 – 5 are available
* P1 checks for block 1, Block 1 Size (20K) < P1 size (90K), can’t fit
* P2 checks for block 2, Block 2 Size (100 k) > p1 size (90K)
* P2 goes to block 2

Process 2

* Block 1 size (20K) < P2 size (50K), can’t fit
* Block 2 is already occupied by P1, unavailable
* Block 3 size (40K) < p2 size (50K), can’t fit
* Block 4 size (200K) > p2 size (50K)
* Block 4 allocated to P2

Process 3

* Block 1 size (20K) < p3 size (30K)
* Block 2 unavailable
* Block 3 size (40K) > p3 size (30K)
* Block 3 allocated to p3

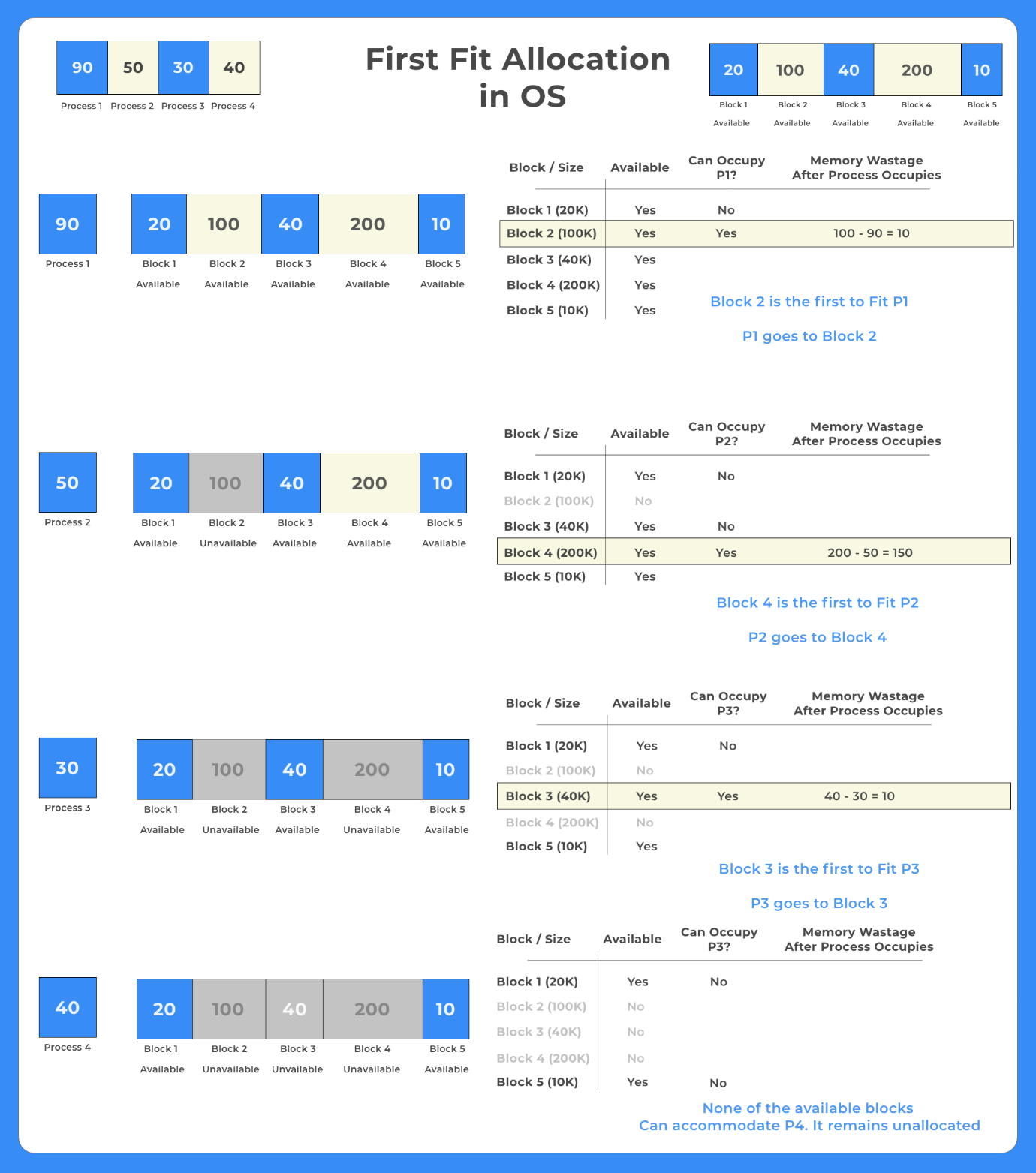
Process 4

* Block 1 size (20K) < p4 size (40K)
* Block 2, 3, 4 are unavailable
* Block 5 size (10K) < p4 size (40K)
* Process 4 remains unallocated

Result

| **Process** | **Size** | **Allocated to** | **Block size** | **Memory Wastage** |
| --- | --- | --- | --- | --- |
| Process 1 | 90 | Block 2 | 100 | 10 |
| Process 2 | 50 | Block 4 | 200 | 150 |
| Process 3 | 30 | Block 3 | 40 | 10 |
| Process 4 | 40 | Unallocated | – | – |

Let’s have a look at the detailed explanation in the image below –



Example of Best Fit Method

This method works as for any process Pn, the OS searches from starting block again and again and allocates a block to process Pn such that –

* Block can accommodate process
* Memory wastage is minimum



In the given example, let us assume the jobs and the memory requirements as the following:

Process

| **Process** | **Size** |
| --- | --- |
| Process 1 | 40 |
| Process 2 | 10 |
| Process 3 | 30 |
| Process 4 | 60 |

Blocks

Let the free space memory allocation blocks be:

| **Blocks** | **Size** |
| --- | --- |
| Block 1 | 100 |
| Block 2 | 50 |
| Block 3 | 30 |
| Block 4 | 120 |
| Block 5 | 35 |

**Job 1 (size – 40K)**

* Block 3 and 5 can not place as their size is lesser than 40K
* Block 1 Memory wastage : 100 – 40 = 60K
* Block 2 Memory wastage : 50 – 40 = 20K
* Block 4 Memory wastage : 120 – 40 = 80K
* Thus Block 2 is best, P1 placed in block 2

Job 2 (size – 10K)

* Block 2 is unavailable now
* Block 1 memory wastage: 100 – 10 = 90
* Block 3 memory wastage: 10 – 10 = 20
* Block 4 memory wastage : 120 – 10 = 110
* Block 5 memory wastage: 35 – 10 = 25
* Thus, block 3 is the best, P2 placed in block 3

Job 3 (size – 30K)

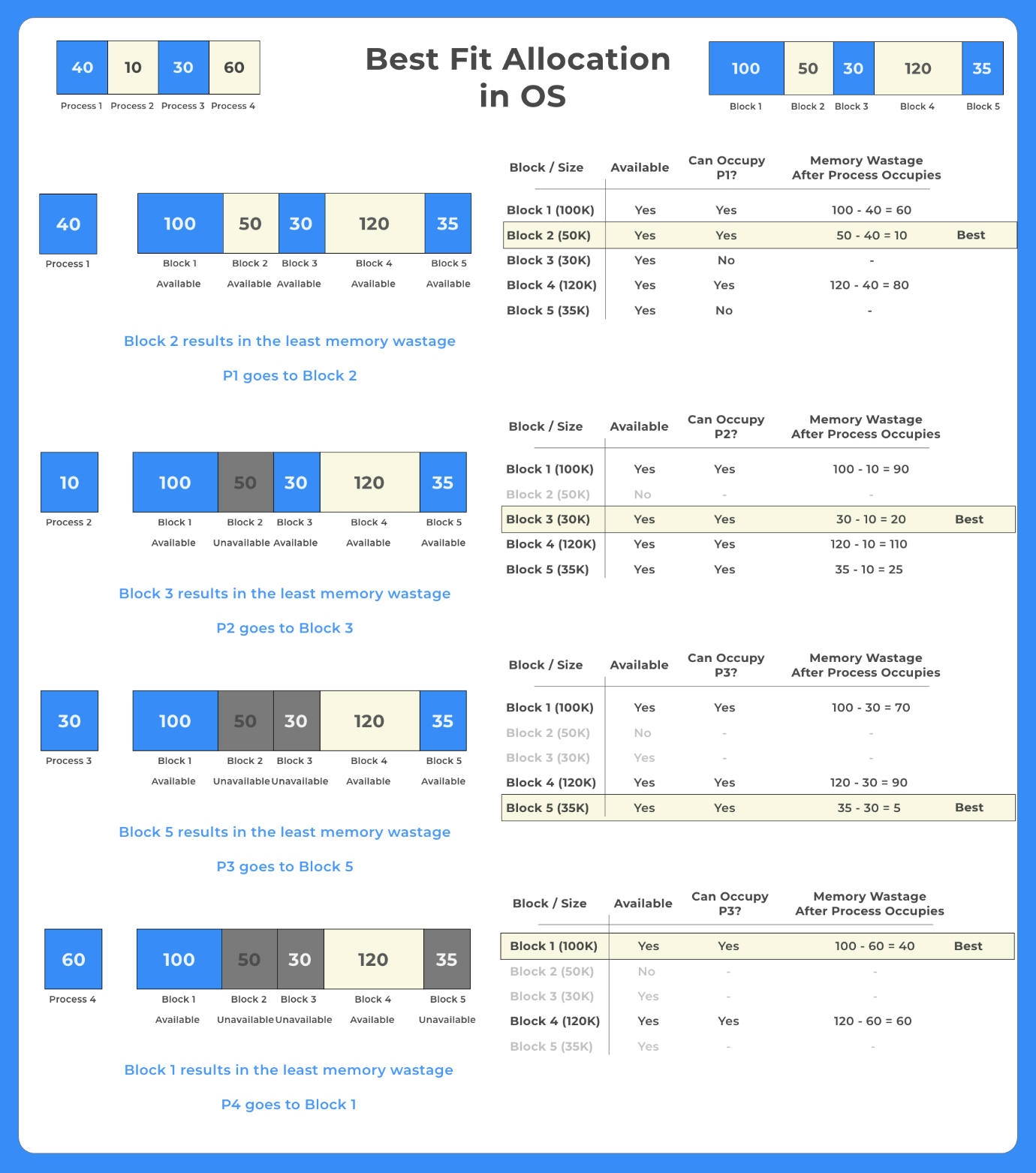
* Block 2, 3 unavailable
* Block 1 memory wastage : 100 – 30 = 70
* Block 4 memory wastage : 120 – 30 = 90
* Block 5 memory wastage : 35 – 30 = 5
* Thus, block is the best, P3 placed in block 5

Job 4 (size 60K)

* Block 2, 3 and 5 are unavailable
* Block 1 memory wastage : 100 – 60 = 40
* Block 4 memory wastage : 120 – 60 = 60
* Thus, block 1 is the best P4 goes to block 1

Results

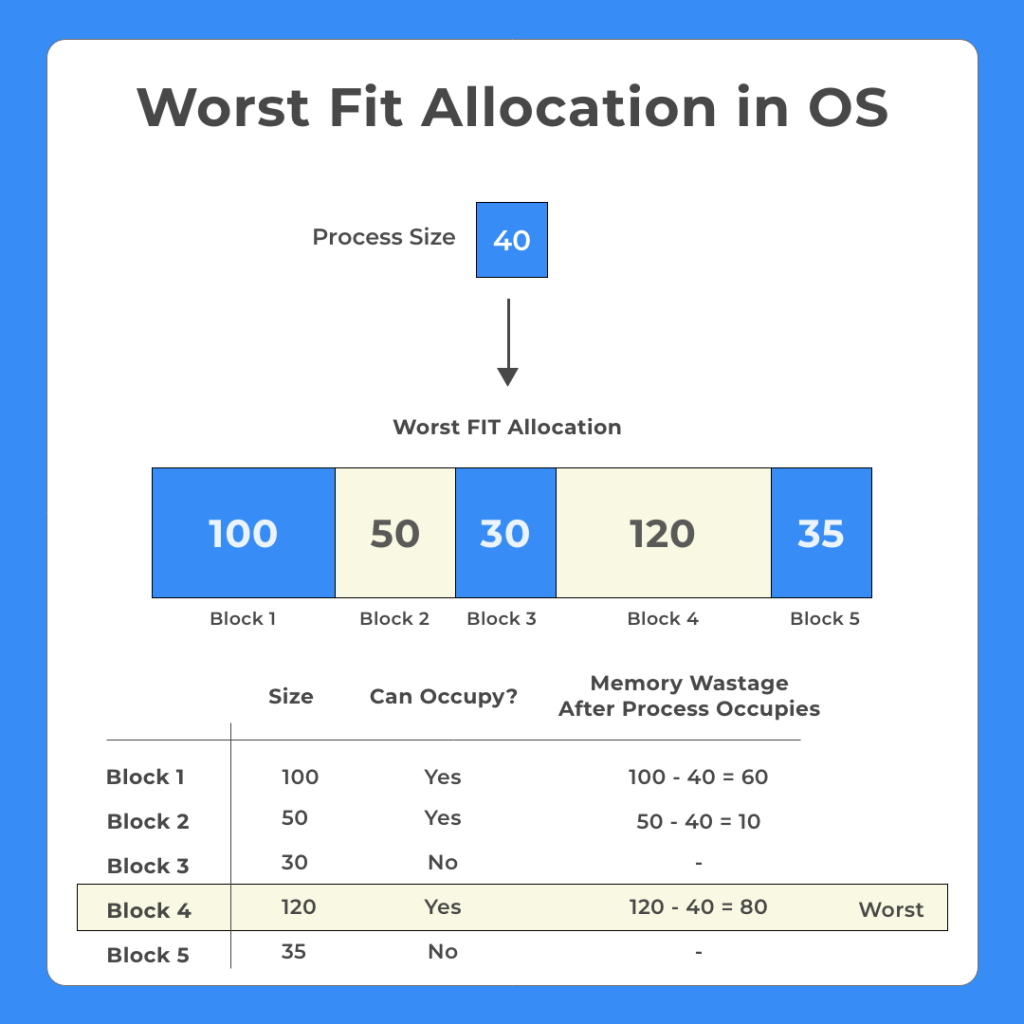
| **Process** | **Size** | **Allocated to** | **Block size** | **Wastage** |
| --- | --- | --- | --- | --- |
| Process 1 | 40 | Block 2 | 50 | 10 |
| Process 2 | 10 | Block 3 | 30 | 20 |
| Process 3 | 30 | Block 5 | 35 | 5 |
| Process 4 | 60 | Block 1 | 100 | 40 |



Example of Best Fit Method

This method works as for any process Pn, the OS searches from starting block again and again and allocates a block to process Pn such that –

* Block can accommodate process
* Memory wastage is maximum



In the given example, let us assume the jobs and the memory requirements as the following:

Process

| **Process** | **Size** |
| --- | --- |
| Process 1 | 40 |
| Process 2 | 10 |
| Process 3 | 30 |
| Process 4 | 60 |

Blocks

Let the free space memory allocation blocks be:

| **Blocks** | **Size** |
| --- | --- |
| Block 1 | 100 |
| Block 2 | 50 |
| Block 3 | 30 |
| Block 4 | 120 |
| Block 5 | 35 |

Explanation

Process 1 (size 40K)

* Block 2 and 5 are smaller than P1 thus can not accommodate
* Block1 memory wastage : 100 – 40 = 60
* Block2 memory wastage: 50 – 40 = 10
* Block3 memory wastage 120 – 40 = 80
* Thus Block 3 is worst and P1 goes to block 3

Process 2 (size: 10)

* Block 4 is occupied
* Block 1 memory wastage: 100 – 10 = 90
* Block 2 memory wastage : 50 – 10 = 40
* Block 3 memory wastage: 30 – 10 = 20
* Block 5 memory wastage: 35 – 10 = 25
* Thus, block 1 is the worst, and Process 2 goes to block 1

Process 3 (size: 30K)

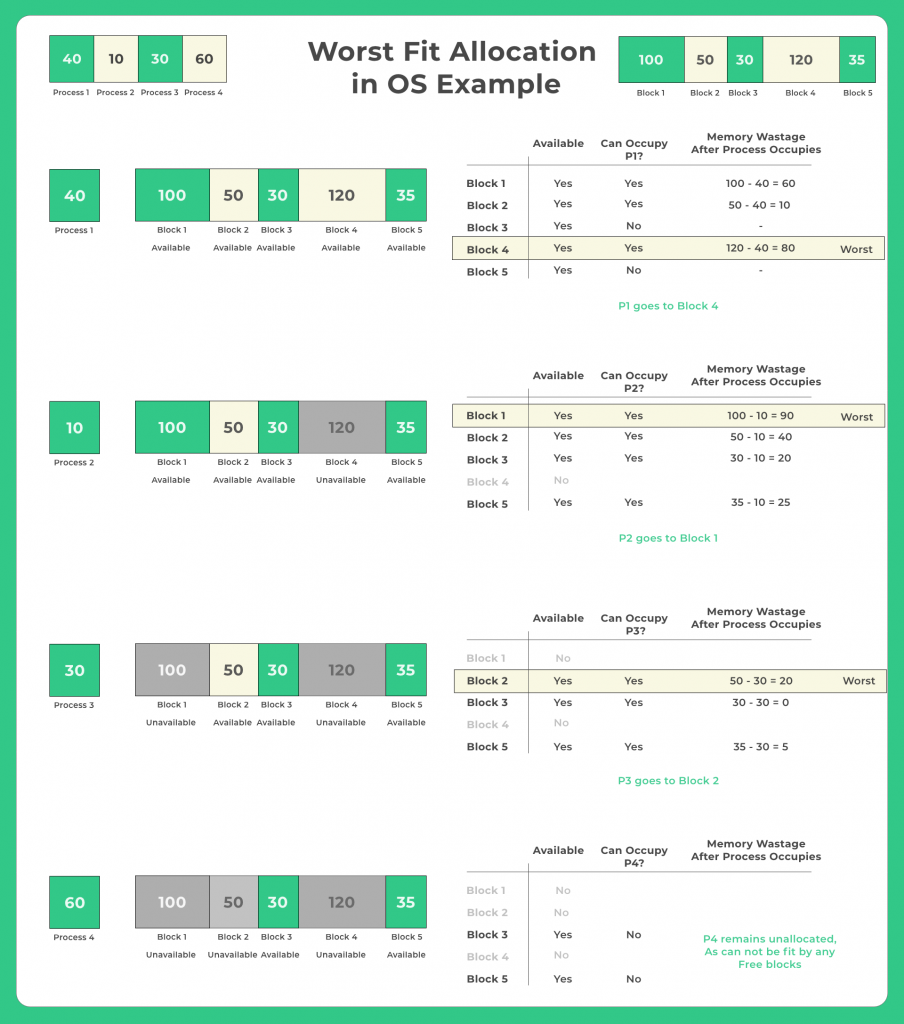
* Block 1 and 4 are occupied
* Block 2 memory wastage : 50 – 30 = 20
* Block 3 memory wastage : 30 – 30 = 0
* Block 5 memory wastage: 35 – 30 = 5
* Thus block 2 is the worst, process 3 goes to block 2

Process 4 (size : 60K)

* Block 1, 2 and 3 are occupied
* Block 3 and 4 can not accommodate the process p4
* Thus, process 4 will remain unallocated

Results

| **Process** | **Size** | **Allocated to** | **Block size** | **Wastage** |
| --- | --- | --- | --- | --- |
| Process 1 | 40 | Block 4 | 120 | 80 |
| Process 2 | 10 | Block 1 | 100 | 90 |
| Process 3 | 30 | Block 2 | 50 | 20 |
| Process 4 | 60 | Unallocated | – | – |



Firstfit

void implimentFirstFit(int blockSize[], int blocks, int processSize[], int processes)

{

// This will store the block id of the allocated block to a process

int allocate[processes];

int occupied[blocks];

// initially assigning -1 to all allocation indexes

// means nothing is allocated currently

for(int i = 0; i < processes; i++)

{

allocate[i] = -1;

}

for(int i = 0; i < blocks; i++){

occupied[i] = 0;

}

// take each process one by one and find

// first block that can accomodate it

for (int i = 0; i < processes; i++)

{

for (int j = 0; j < blocks; j++)

{

if (!occupied[j] && blockSize[j] >= processSize[i])

{

// allocate block j to p[i] process

allocate[i] = j;

occupied[j] = 1;

break;

}

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

for (int i = 0; i < processes; i++)

{

printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);

if (allocate[i] != -1)

printf("%d\n",allocate[i] + 1);

else

printf("Not Allocated\n");

}

}

void main()

{

int blockSize[] = {30, 5, 10};

int processSize[] = {10, 6, 9};

int m = sizeof(blockSize)/sizeof(blockSize[0]);

int n = sizeof(processSize)/sizeof(processSize[0]);

implimentFirstFit(blockSize, m, processSize, n);

}

### Output

Process No. Process Size Block no.

1 10 1

2 6 3

3 9 Not Allocated

## Best Fit

#include <stdio.h>

void implimentBestFit(int blockSize[], int blocks, int processSize[], int proccesses)

{

// This will store the block id of the allocated block to a process

int allocation[proccesses];

int occupied[blocks];

// initially assigning -1 to all allocation indexes

// means nothing is allocated currently

for(int i = 0; i < proccesses; i++){

allocation[i] = -1;

}

for(int i = 0; i < blocks; i++){

occupied[i] = 0;

}

// pick each process and find suitable blocks

// according to its size ad assign to it

for (int i = 0; i < proccesses; i++)

{

int indexPlaced = -1;

for (int j = 0; j < blocks; j++) {

if (blockSize[j] >= processSize[i] && !occupied[j])

{

// place it at the first block fit to accomodate process

if (indexPlaced == -1)

indexPlaced = j;

// if any future block is smalller than the current block where

// process is placed, change the block and thus indexPlaced

// this reduces the wastage thus best fit

else if (blockSize[j] < blockSize[indexPlaced])

indexPlaced = j;

}

}

// If we were successfully able to find block for the process

if (indexPlaced != -1)

{

// allocate this block j to process p[i]

allocation[i] = indexPlaced;

// make the status of the block as occupied

occupied[indexPlaced] = 1;

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

for (int i = 0; i < proccesses; i++)

{

printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);

if (allocation[i] != -1)

printf("%d\n",allocation[i] + 1);

else

printf("Not Allocated\n");

}

}

// Driver code

int main()

{

int blockSize[] = {100, 50, 30, 120, 35};

int processSize[] = {40, 10, 30, 60};

int blocks = sizeof(blockSize)/sizeof(blockSize[0]);

int proccesses = sizeof(processSize)/sizeof(processSize[0]);

implimentBestFit(blockSize, blocks, processSize, proccesses);

return 0 ;

}

#### Output

Process No. Process Size Block no.

1 10 2

2 30 1

3 60 4

4 30 4

**Worst Fit**

#include <stdio.h>

void implimentWorstFit(int blockSize[], int blocks, int processSize[], int processes)

{

// This will store the block id of the allocated block to a process

int allocation[processes];

int occupied[blocks];

// initially assigning -1 to all allocation indexes

// means nothing is allocated currently

for(int i = 0; i < processes; i++){

allocation[i] = -1;

}

for(int i = 0; i < blocks; i++){

occupied[i] = 0;

}

// pick each process and find suitable blocks

// according to its size ad assign to it

for (int i=0; i < processes; i++)

{

int indexPlaced = -1;

for(int j = 0; j < blocks; j++)

{

// if not occupied and block size is large enough

if(blockSize[j] >= processSize[i] && !occupied[j])

{

// place it at the first block fit to accomodate process

if (indexPlaced == -1)

indexPlaced = j;

// if any future block is larger than the current block where

// process is placed, change the block and thus indexPlaced

else if (blockSize[indexPlaced] < blockSize[j])

indexPlaced = j;

}

}

// If we were successfully able to find block for the process

if (indexPlaced != -1)

{

// allocate this block j to process p[i]

allocation[i] = indexPlaced;

// make the status of the block as occupied

occupied[indexPlaced] = 1;

// Reduce available memory for the block

blockSize[indexPlaced] -= processSize[i];

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

for (int i = 0; i < processes; i++)

{

printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);

if (allocation[i] != -1)

printf("%d\n",allocation[i] + 1);

else

printf("Not Allocated\n");

}

}

// Driver code

int main()

{

int blockSize[] = {100, 50, 30, 120, 35};

int processSize[] = {40, 10, 30, 60};

int blocks = sizeof(blockSize)/sizeof(blockSize[0]);

int processes = sizeof(processSize)/sizeof(processSize[0]);

implimentWorstFit(blockSize, blocks, processSize, processes);

return 0;

}

## ****Output****

Process No. Process Size Block no.

1 40 4

2 10 1

3 30 2

4 60 Not Allocated