

Mr C plays Ring'a- Ring'o Roses!

ESC101: Fundamentals of Computing

Purushottam Kar

Lab Exam (Sun, 04 Nov)

- Morning exam (Mon, Tue batches)
 - 10:30 AM - 2 PM – starts 10:30 AM sharp
 - **CC-01**: B2, {B1 even roll numbers}
 - **CC-02**: B4, B5, B6
 - **CC-03**: B3
 - **MATH-LINUX**: B13, {B1 odd roll numbers}
- Go see your room during this week's lab
- Be there 15 minutes before your exam 10:15AM
- Cannot switch to afternoon session



Lab Exam (Sun, 04 Nov)

- Evening exam (Wed, Thu batches)
 - 2:30 PM - 6 PM – starts 2:30 PM sharp
 - **CC-01**: B9, {B8 odd roll numbers}
 - **CC-02**: B7, B10, B11
 - **CC-03**: B12
 - **MATH-LINUX**: B14, {B8 even roll numbers}
- Go see your room during this week's lab
- Be there 15 minutes before your exam 2:15 PM
- Cannot switch to morning session



Lab Exam (Sun, 04 Nov)

- **Syllabus** – till structures
- Open handwritten notes – However, **NO** printouts, photocopies, slides, websites, mobile phone, Ipad
- **USE OF ANY OF ABOVE WILL BE CONSIDERED CHEATING**
- Prutor CodeBook will be unavailable during lab exam
- Exam will be like labs - marks for passing test cases
- Marks for writing clean indented code, proper variable names, a few comments – illegible code poor marks



Are Arrays the Best?

5



Are Arrays the Best?

ADVANTAGES

5



Are Arrays the Best?

5

ADVANTAGES

Allow us to create several variables of a given type



Are Arrays the Best?

5

ADVANTAGES

Allow us to create several variables of a given type

Allow us to give them very convenient names e.g. `arr[i]`



Are Arrays the Best?

5

ADVANTAGES

Allow us to create several variables of a given type

Allow us to give them very convenient names e.g. `arr[i]`

Can access n-th element very very easily – just use `arr[n-1]`



Are Arrays the Best?

5

ADVANTAGES

Allow us to create several variables of a given type

Allow us to give them very convenient names e.g. `arr[i]`

Can access n-th element very very easily – just use `arr[n-1]`

Very easy to set up, can also change size (realloc)



Are Arrays the Best?

5

ADVANTAGES

- Allow us to create several variables of a given type
- Allow us to give them very convenient names e.g. `arr[i]`
- Can access n-th element very very easily – just use `arr[n-1]`
- Very easy to set up, can also change size (realloc)
- Can have arrays of structures as well



Are Arrays the Best?

5

ADVANTAGES

Allow us to create several variables of a given type

Allow us to give them very convenient names e.g. `arr[i]`

Can access n-th element very very easily – just use `arr[n-1]`

Very easy to set up, can also change size (realloc)

Can have arrays of structures as well

Inserting a new element at the end of the array simple



Are Arrays the Best?

5

ADVANTAGES

- Allow us to create several variables of a given type
- Allow us to give them very convenient names e.g. `arr[i]`
- Can access n-th element very very easily – just use `arr[n-1]`
- Very easy to set up, can also change size (realloc)
- Can have arrays of structures as well
- Inserting a new element at the end of the array simple

DISADVANTAGES



Are Arrays the Best?

5

ADVANTAGES

- Allow us to create several variables of a given type
- Allow us to give them very convenient names e.g. `arr[i]`
- Can access n-th element very very easily – just use `arr[n-1]`
- Very easy to set up, can also change size (realloc)
- Can have arrays of structures as well
- Inserting a new element at the end of the array simple

DISADVANTAGES

- Inserting in the middle/beginning of array tedious ☹ - need to shift elements one location to make space – can be time consuming too!



Are Arrays the Best?

5

ADVANTAGES

- Allow us to create several variables of a given type
- Allow us to give them very convenient names e.g. `arr[i]`
- Can access n-th element very very easily – just use `arr[n-1]`
- Very easy to set up, can also change size (realloc)
- Can have arrays of structures as well
- Inserting a new element at the end of the array simple

DISADVANTAGES

- Inserting in the middle/beginning of array tedious ☹ - need to shift elements one location to make space – can be time consuming too!
- Realloc is an expensive procedure – Mr C has to find new space for the enlarged array, allocate that space and then copy all old elements one by one ☹



Are Arrays the Best?

5

ADVANTAGES

- Allow us to create several variables of a given type
- Allow us to give them very convenient names e.g. `arr[i]`
- Can access n-th element very very easily – just use `arr[n-1]`
- Very easy to set up, can also change size (realloc)
- Can have arrays of structures as well
- Inserting a new element at the end of the array simple

DISADVANTAGES

- Inserting in the middle/beginning of array tedious ☹ - need to shift elements one location to make space – can be time consuming too!
- Realloc is an expensive procedure – Mr C has to find new space for the enlarged array, allocate that space and then copy all old elements one by one ☹
- Sometimes if there is not enough memory, realloc may just fail and return a NULL pointer ☹

Realloc can fail!

6



Realloc can fail!

000000								
000001								
000002								
000003								
000004								
000005								
000006								
000007								
000008								
000009								
000010								
000011								
000012								
000013								
000014								
000015								
000016								
000017								
000018								
000019								
000020								
000021								
000022								
000023								
000024								
000025								

Realloc can fail!

000000									
000001									
000002									
000003									
000004									
000005									
000006									
000007									
000008									
000009									
000010									
000011									
000012									
000013									
000014									
000015									
000016									
000017									
000018									
000019									
000020									
000021									
000022									
000023									
000024									
000025									

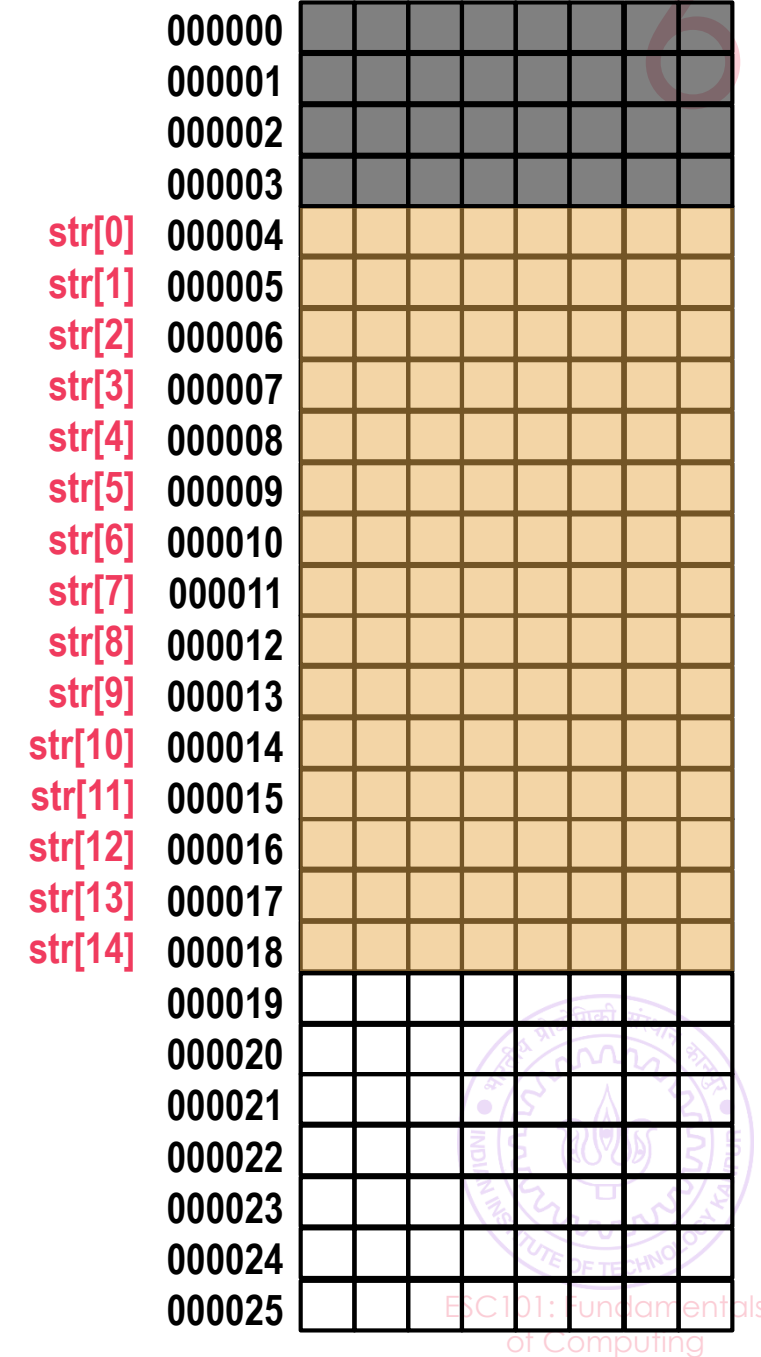
Realloc can fail!

```
char *str = (char*)malloc(15 * sizeof(char));
```

000000						
000001						
000002						
000003						
000004						
000005						
000006						
000007						
000008						
000009						
000010						
000011						
000012						
000013						
000014						
000015						
000016						
000017						
000018						
000019						
000020						
000021						
000022						
000023						
000024						
000025						

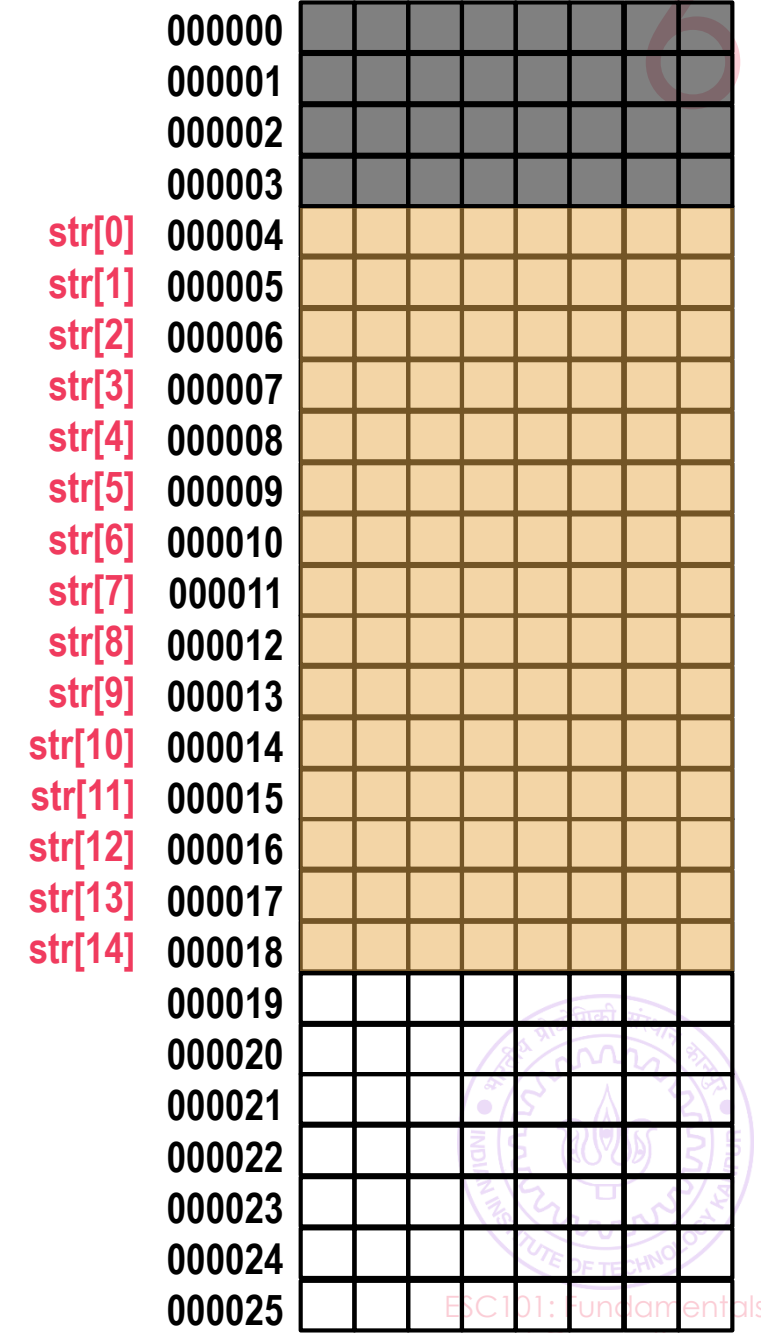
Realloc can fail!

```
char *str = (char*)malloc(15 * sizeof(char));
```



Realloc can fail!

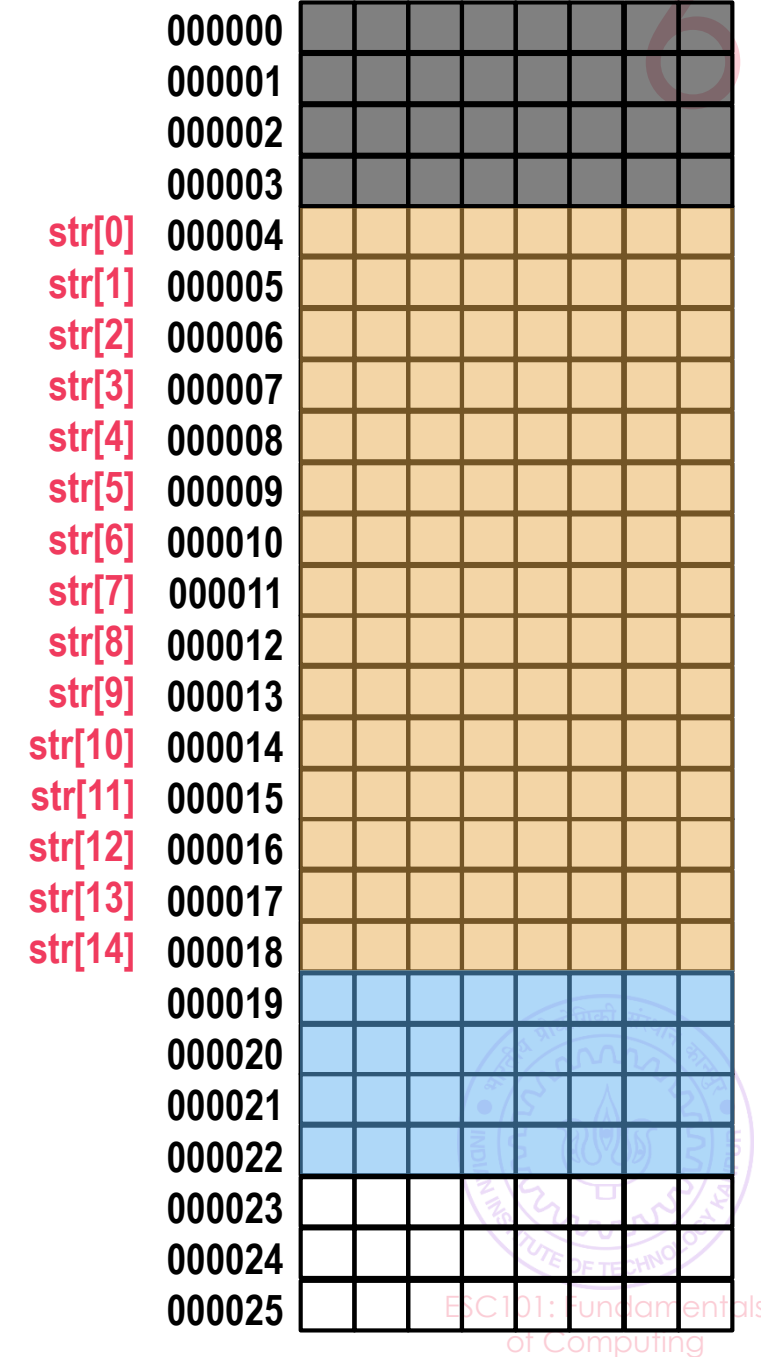
```
char *str = (char*)malloc(15 * sizeof(char));  
int a;
```



Realloc can fail!

```
char *str = (char*)malloc(15 * sizeof(char));  
int a;
```

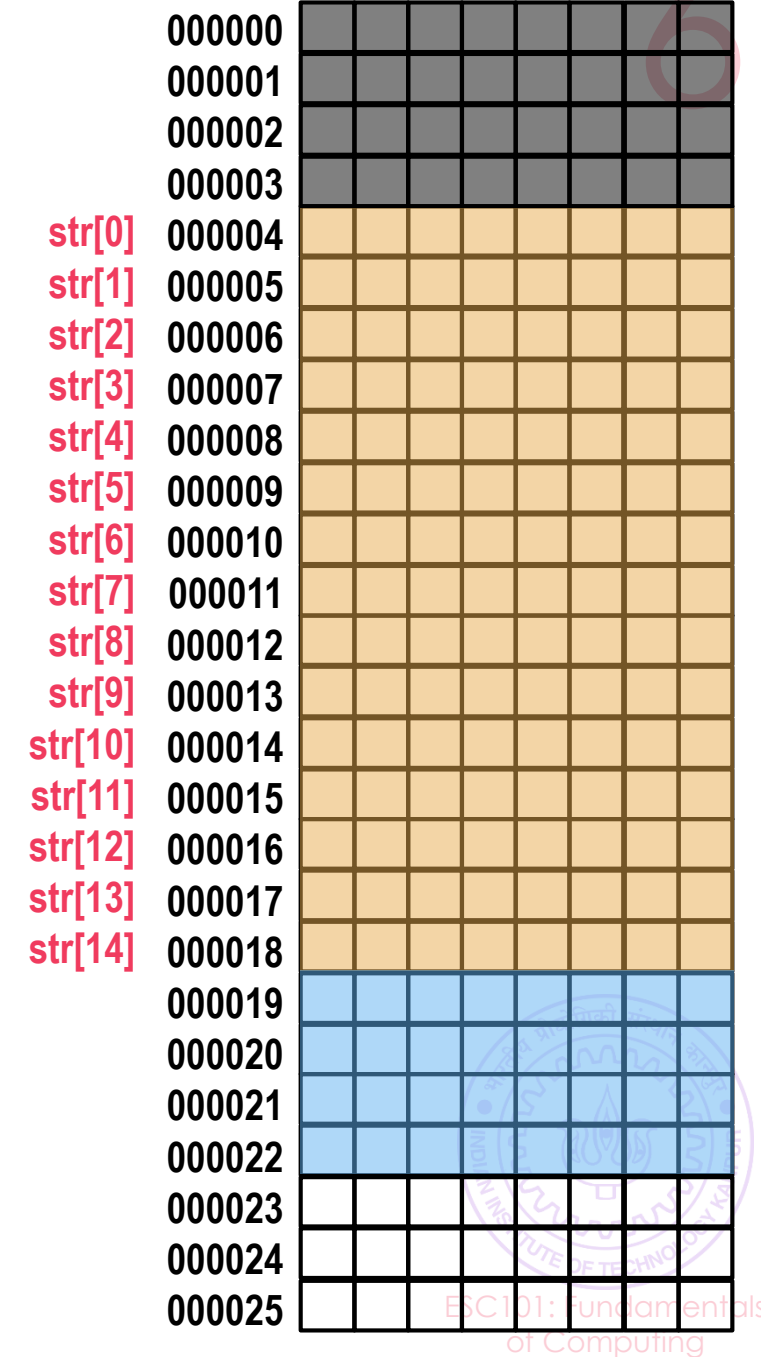
a



Realloc can fail!

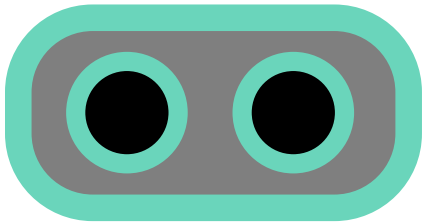
```
char *str = (char*)malloc(15 * sizeof(char));  
int a;  
char *ptr = (char*)realloc(str, 18 * sizeof(char));
```

a



Realloc can fail!

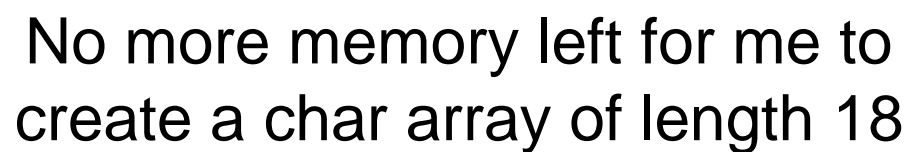
```
char *str = (char*)malloc(15 * sizeof(char));  
int a;  
char *ptr = (char*)realloc(str, 18 * sizeof(char));
```



a

	000000						
	000001						
	000002						
	000003						
str[0]	000004						
str[1]	000005						
str[2]	000006						
str[3]	000007						
str[4]	000008						
str[5]	000009						
str[6]	000010						
str[7]	000011						
str[8]	000012						
str[9]	000013						
str[10]	000014						
str[11]	000015						
str[12]	000016						
str[13]	000017						
str[14]	000018						
	000019						
	000020						
	000021						
	000022						
	000023						
	000024						
	000025						

```
int a;
```



ESC101: Fundamentals of Computing

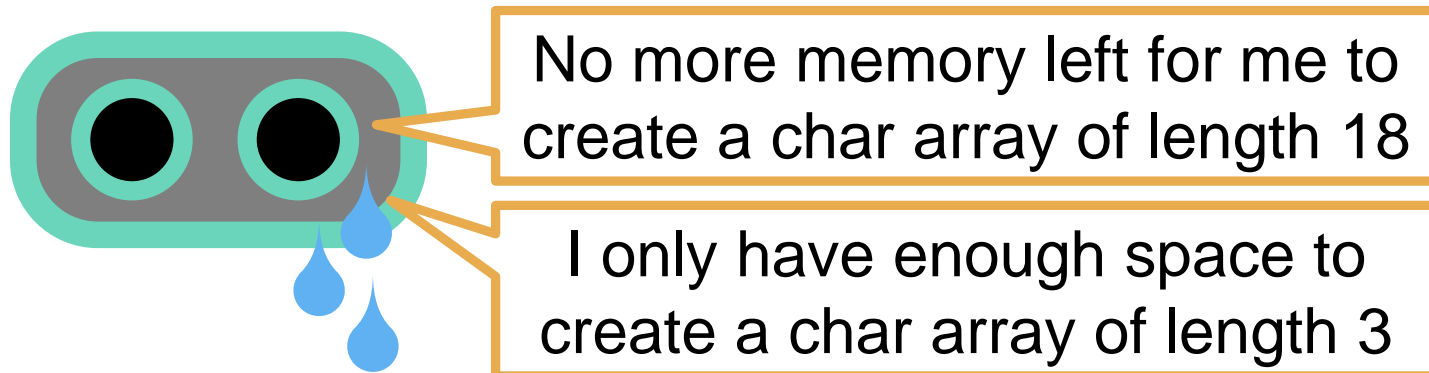
```
int a;
```



ESC101: Fundamentals
of Computing

Realloc can fail!

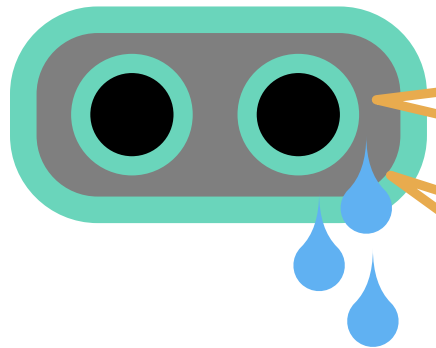
```
char *str = (char*)malloc(15 * sizeof(char));
int a;
char *ptr = (char*)realloc(str, 18 * sizeof(char));
if(ptr != NULL)
    str = ptr;
```



	000000						
	000001						
	000002						
	000003						
str[0]	000004						
str[1]	000005						
str[2]	000006						
str[3]	000007						
str[4]	000008						
str[5]	000009						
str[6]	000010						
str[7]	000011						
str[8]	000012						
str[9]	000013						
str[10]	000014						
str[11]	000015						
str[12]	000016						
str[13]	000017						
str[14]	000018						
	000019						
	000020						
	000021						
	000022						
	000023						
	000024						
	000025						

Realloc can fail!

```
char *str = (char*)malloc(15 * sizeof(char));  
int a;  
char *ptr = (char*)realloc(str, 18 * sizeof(char)),  
if(ptr != NULL)  
    str = ptr;
```



No more memory left for me to create a char array of length 18

I only have enough space to create a char array of length 3

	000000	
	000001	
	000002	
	000003	
str[0]	000004	
str[1]	000005	
str[2]	000006	
str[3]	000007	
str[4]	000008	
str[5]	000009	
str[6]	000010	
str[7]	000011	
str[8]	000012	
str[9]	000013	
str[10]	000014	
str[11]	000015	
str[12]	000016	
str[13]	000017	
str[14]	000018	
	000019	
	000020	
	000021	
	000022	
	000023	
	000024	
	000025	

Real

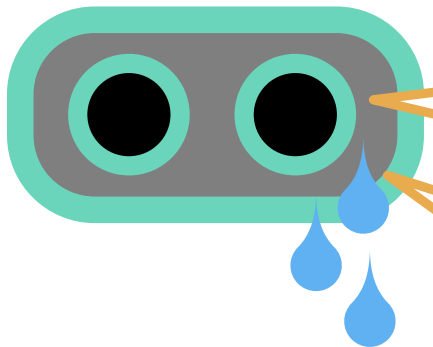
So close! We did have space for 18 characters, just not contiguous space

```
char *str = (char*)malloc(15 * sizeof(char));
int a;
char *ptr = (char*)realloc(str, 18 * sizeof(char));
if(ptr != NULL)
    str = ptr;
```



str[0]	000004
str[1]	000005
str[2]	000006
str[3]	000007
str[4]	000008
str[5]	000009
str[6]	000010
str[7]	000011
str[8]	000012
str[9]	000013
str[10]	000014
str[11]	000015
str[12]	000016
str[13]	000017
str[14]	000018

000000							
000001							
000002							
000003							
000004							
000005							
000006							
000007							
000008							
000009							
000010							
000011							
000012							
000013							
000014							
000015							
000016							
000017							
000018							
000019							
000020							
000021							
000022							
000023							
000024							
000025							



No more memory left for me to create a char array of length 18

I only have enough space to
create a char array of length 3

So close! We did have space for 18 characters, just not contiguous space

```
(char*)malloc(15 * sizeof(char));
```

Can't we create a 3 length array and link the two arrays together?

```
char ptr = (char *)malloc(3 * sizeof(char));  
if(ptr != NULL)  
    str = ptr;
```

No more memory left for me to create a char array of length 18

I only have enough space to create a char array of length 3

	000000	
	000001	
	000002	
	000003	
str[0]	000004	
str[1]	000005	
str[2]	000006	
str[3]	000007	
str[4]	000008	
str[5]	000009	
str[6]	000010	
str[7]	000011	
str[8]	000012	
str[9]	000013	
str[10]	000014	
str[11]	000015	
str[12]	000016	
str[13]	000017	
str[14]	000018	

	000019	
	000020	
	000021	
	000022	
	000023	
	000024	
	000025	

So close! We did have space for 18 characters, just not contiguous space

```
(char*)malloc(15 * sizeof(char));
```

Can't we create a 3 length array and link the two arrays together?

```
char ptr = (char *)malloc(3 * sizeof(char)),
```

```
if(ptr != NULL)
```

```
str = ptr;
```

Yes, you can – using structures and linked lists

No more memory left for me to create a char array of length 18

I only have enough space to create a char array of length 3

str[0]	000004
str[1]	000005
str[2]	000006
str[3]	000007
str[4]	000008
str[5]	000009
str[6]	000010
str[7]	000011
str[8]	000012
str[9]	000013
str[10]	000014
str[11]	000015
str[12]	000016
str[13]	000017
str[14]	000018

000000
000001
000002
000003
000004
000005
000006
000007
000008
000009
000010
000011
000012
000013
000014
000015
000016
000017
000018
000019
000020
000021
000022
000023
000024
000025

Linked Lists

7



Linked Lists

000000									
000001									
000002									
000003									
000004									
000005									
000006									
000007									
000008									
000009									
000010									
000011									
000012									
000013									
000014									
000015									
000016									
000017									
000018									
000019									
000020									
000021									
000022									
000023									
000024									
000025									

Linked Lists

000000									
000001									
000002									
000003									
000004									
000005									
000006									
000007									
000008									
000009									
000010									
000011									
000012									
000013									
000014									
000015									
000016									
000017									
000018									
000019									
000020									
000021									
000022									
000023									
000024									
000025									

Linked Lists

000000								
000001								
000002								
000003								
000004								
000005								
000006								
000007								
000008								
000009								
000010								
000011								
000012								
000013								
000014								
000015								
000016								
000017								
000018								
000019								
000020								
000021								
000022								
000023								
000024								
000025								

Linked Lists

000000								
000001								
000002								
000003								
000004								
000005								
000006								
000007								
000008								
000009								
000010								
000011								
000012								
000013								
000014								
000015								
000016								
000017								
000018								
000019								
000020								
000021								
000022								
000023								
000024								
000025								

Linked Lists

000000								
000001								
000002								
000003								
000004								
000005								
000006								
000007								
000008								
000009								
000010								
000011								
000012								
000013								
000014								
000015								
000016								
000017								
000018								
000019								
000020								
000021								
000022								
000023								
000024								
000025								

Linked Lists

000000								
000001								
000002								
000003								
000004								
000005								
000006								
000007								
000008								
000009								
000010								
000011								
000012								
000013								
000014								
000015								
000016								
000017								
000018								
000019								
000020								
000021								
000022								
000023								
000024								
000025								

Linked Lists

000000								
000001								
000002								
000003								
000004								
000005	0	0	0	0	0	0	0	6
000006								
000007								
000008								
000009								
000010								
000011								
000012								
000013								
000014								
000015								
000016								
000017								
000018								
000019								
000020								
000021								
000022								
000023								
000024								
000025								

Linked Lists

000000								
000001								
000002								
000003								
000004								
000005	0	0	0	0	0	0	0	6
000006								
000007								
000008								
000009								
000010								
000011								
000012								
000013								
000014								
000015								
000016								
000017								
000018								
000019								
000020								
000021								
000022								
000023								
000024								
000025								

ESC101: Fundamentals
of Computing

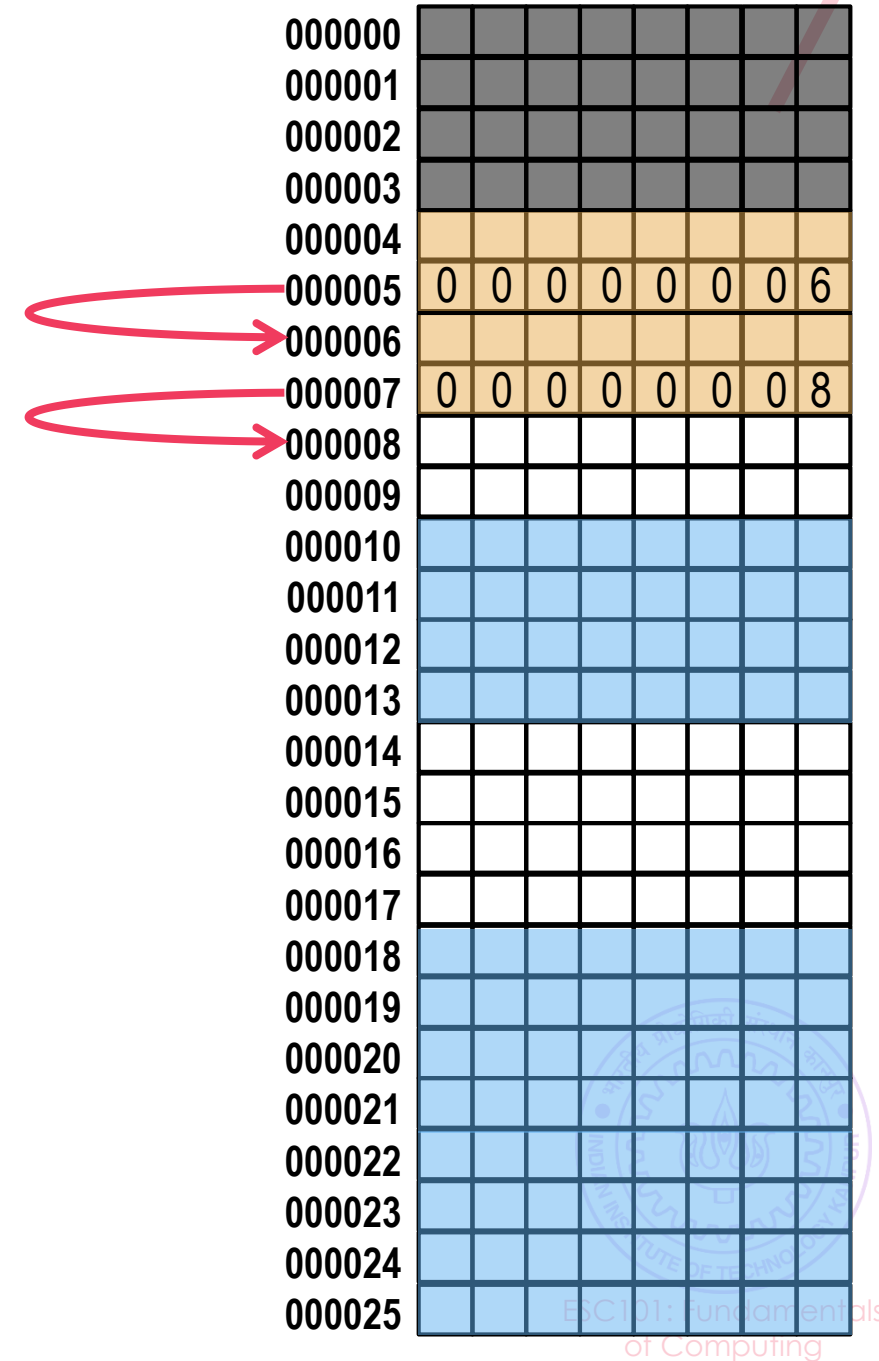
Linked Lists

000000								
000001								
000002								
000003								
000004								
000005	0	0	0	0	0	0	0	6
000006								
000007								
000008								
000009								
000010								
000011								
000012								
000013								
000014								
000015								
000016								
000017								
000018								
000019								
000020								
000021								
000022								
000023								
000024								
000025								

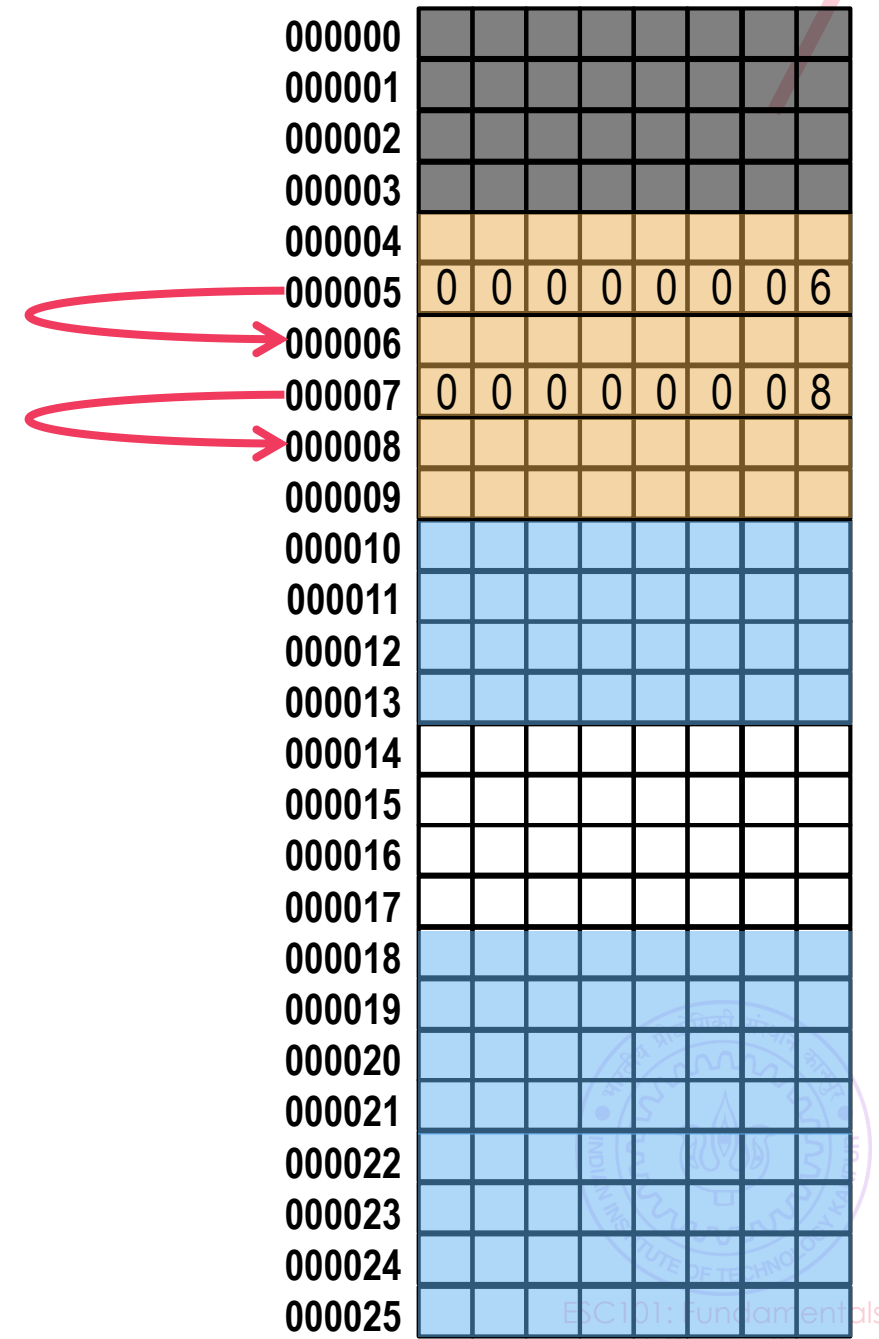
Linked Lists

000000								
000001								
000002								
000003								
000004								
000005	0	0	0	0	0	0	0	6
000006								
000007	0	0	0	0	0	0	0	8
000008								
000009								
000010								
000011								
000012								
000013								
000014								
000015								
000016								
000017								
000018								
000019								
000020								
000021								
000022								
000023								
000024								
000025								

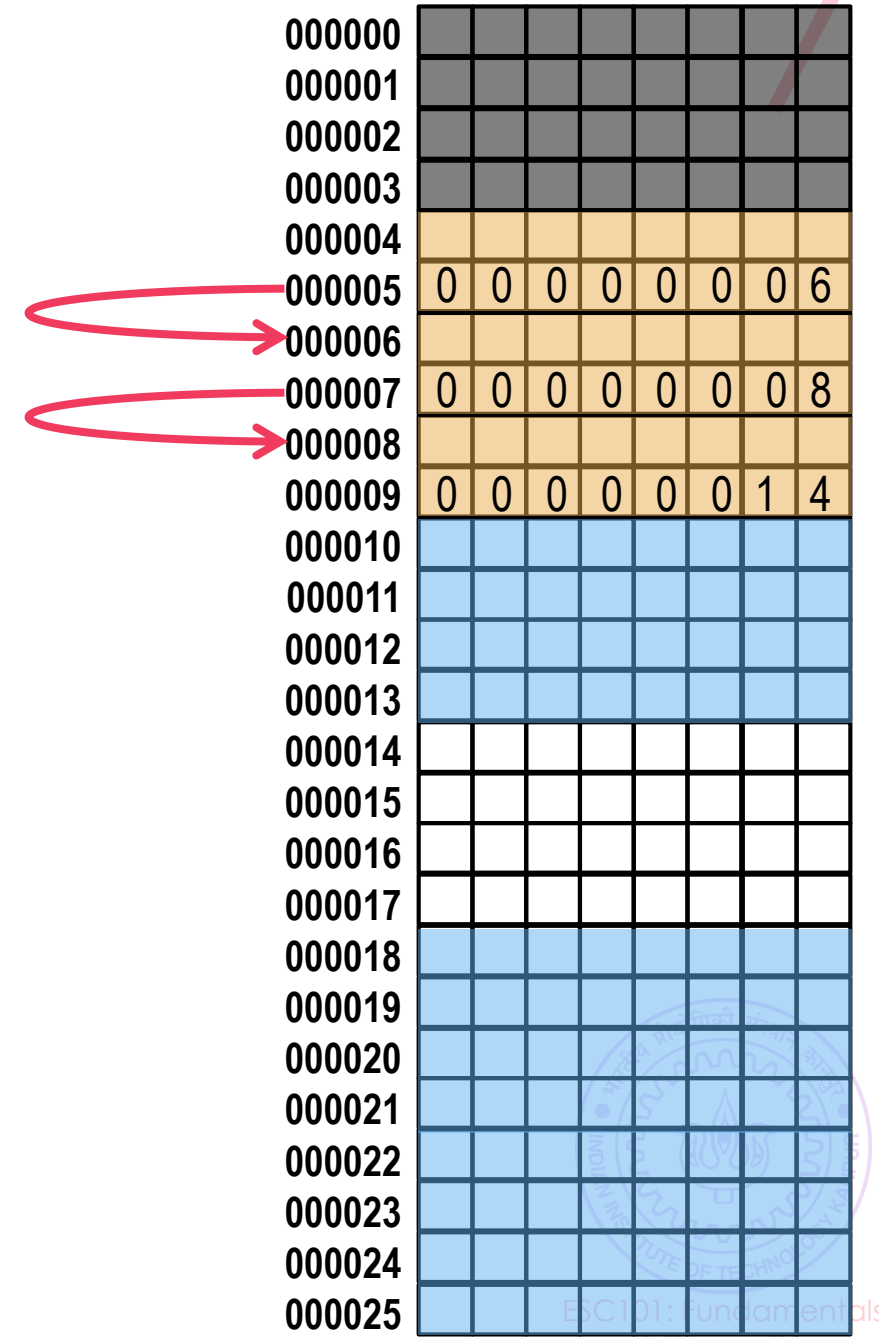
Linked Lists



Linked Lists

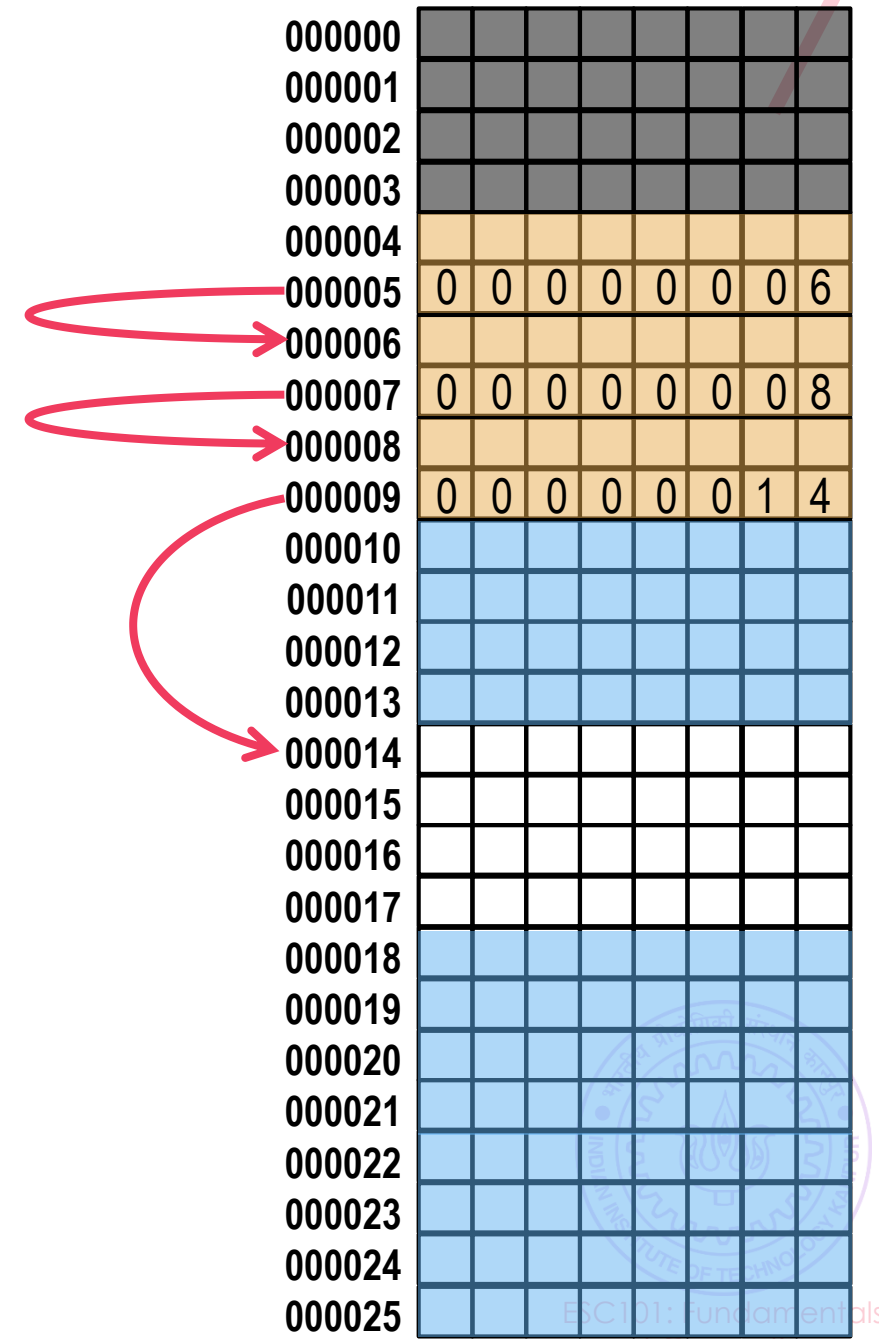


Linked Lists

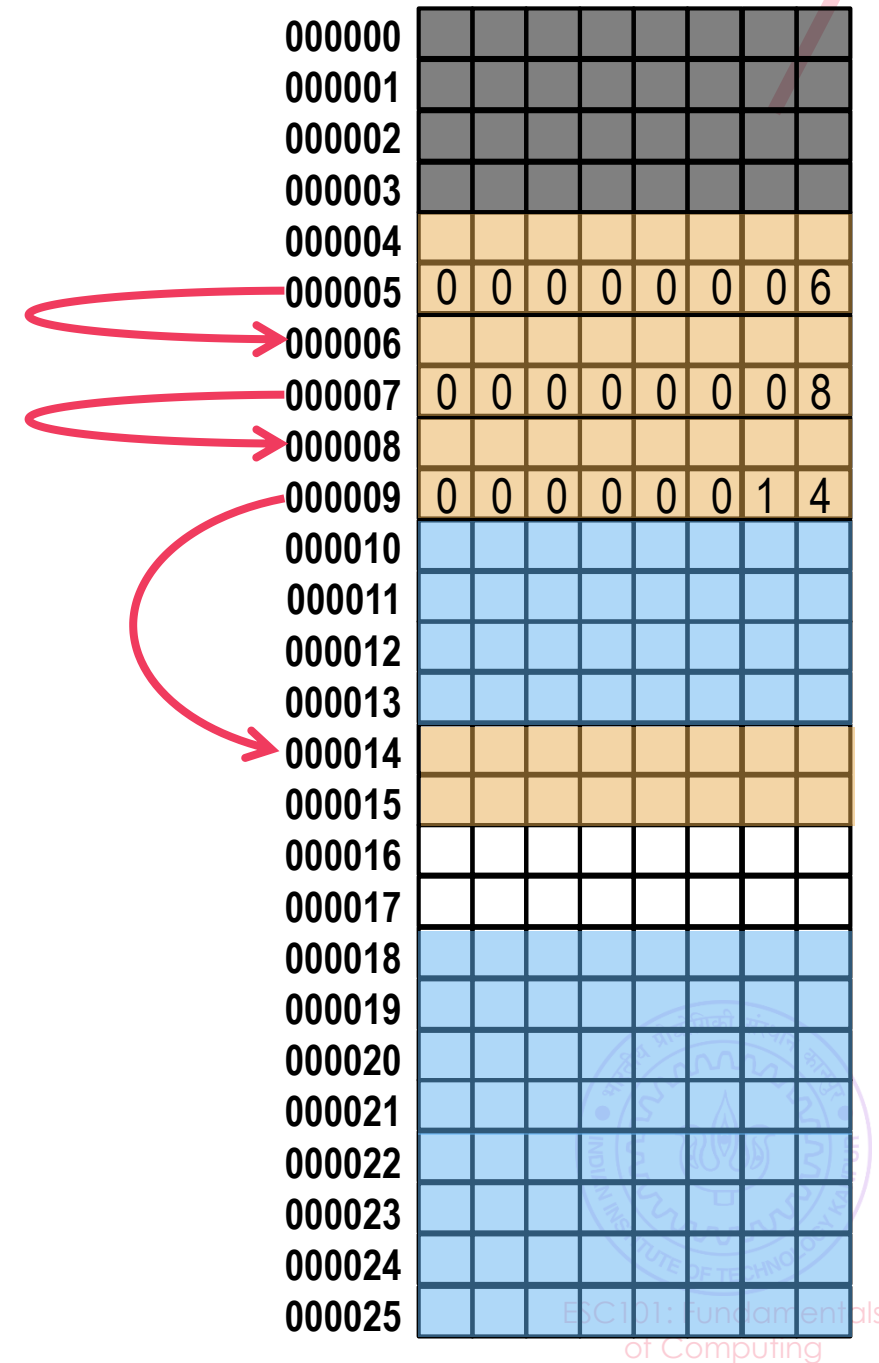


000000								
000001								
000002								
000003								
000004								
000005	0	0	0	0	0	0	0	6
000006								
000007	0	0	0	0	0	0	0	8
000008								
000009	0	0	0	0	0	0	1	4
000010								
000011								
000012								
000013								
000014								
000015								
000016								
000017								
000018								
000019								
000020								
000021								
000022								
000023								
000024								
000025								

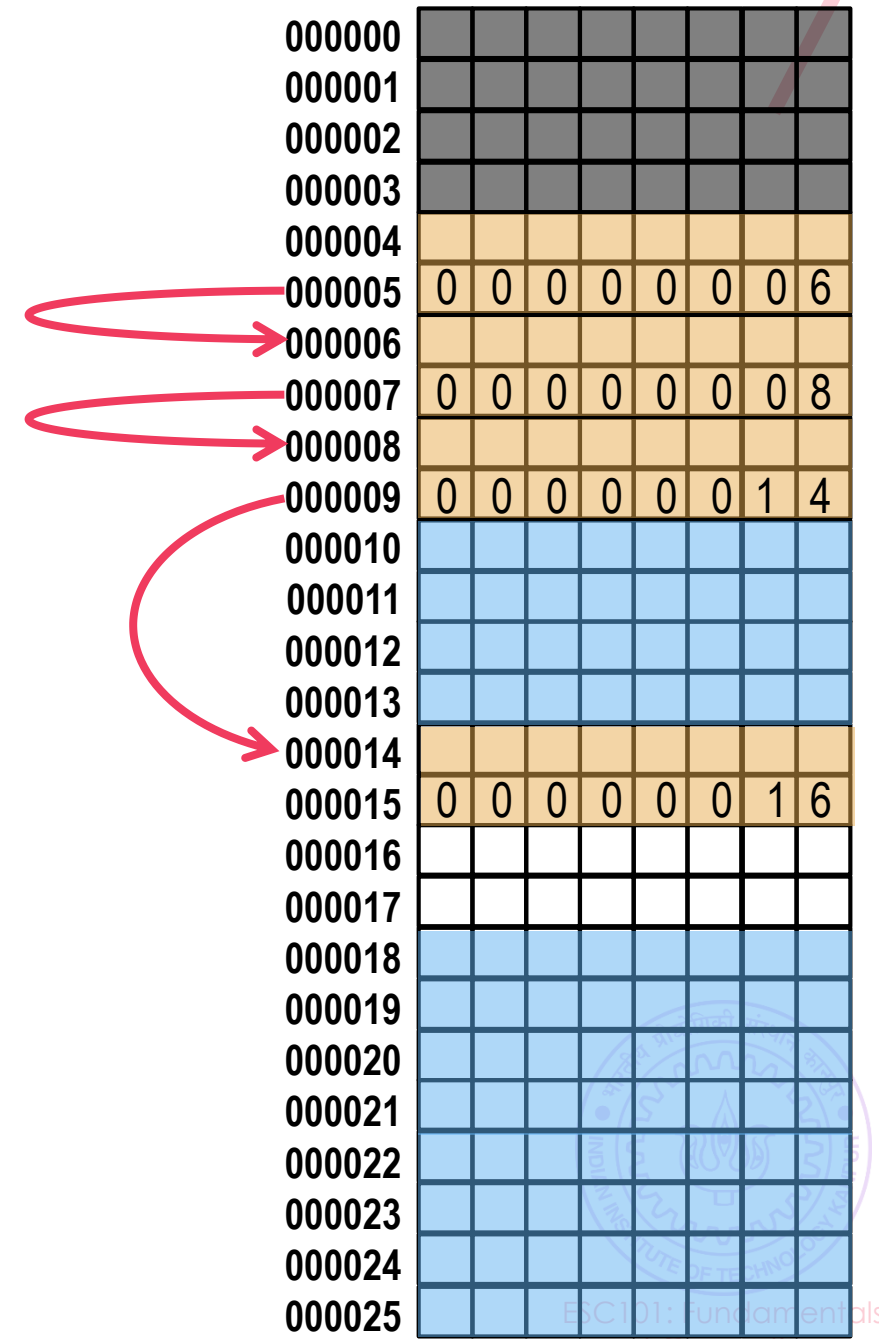
Linked Lists



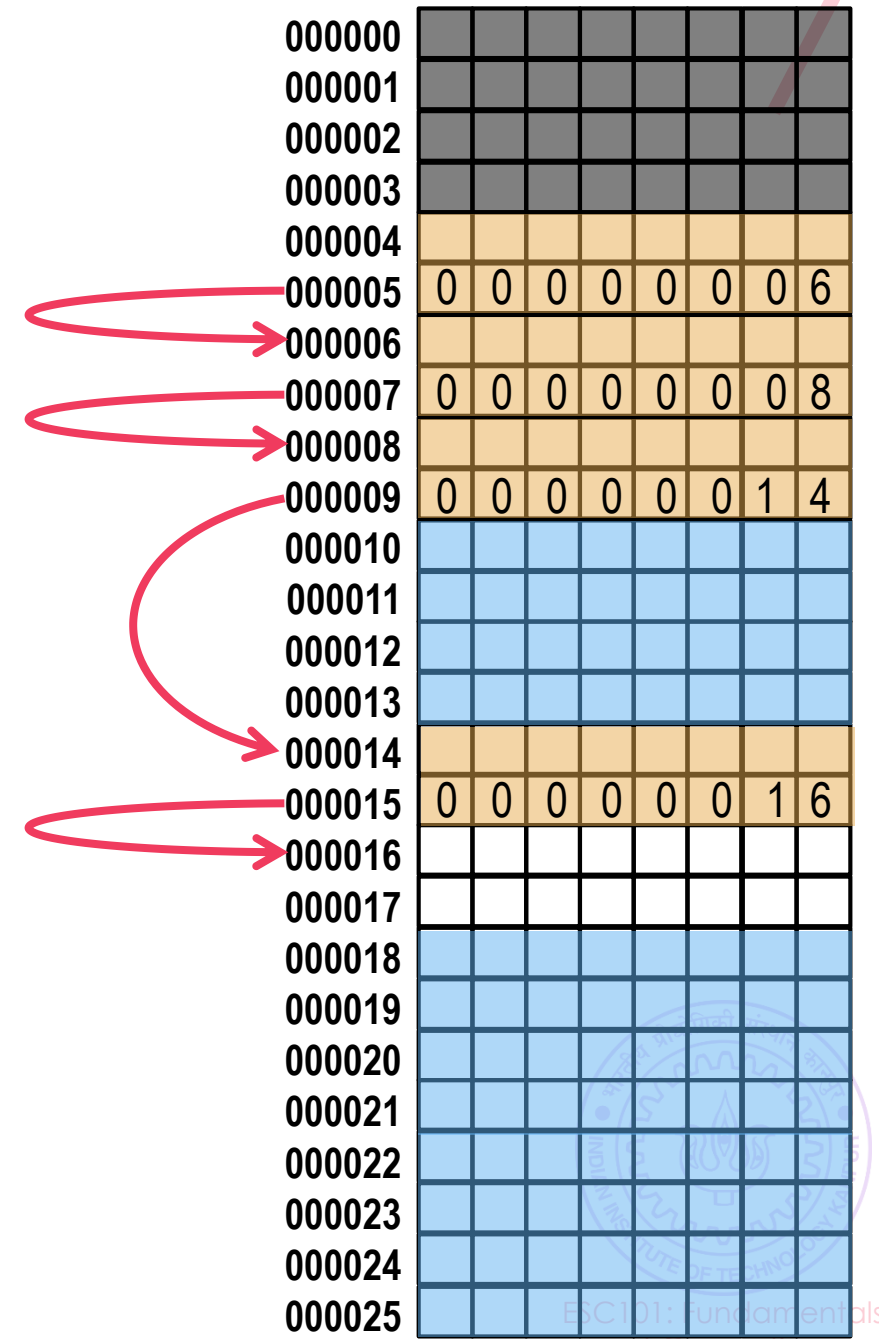
Linked Lists



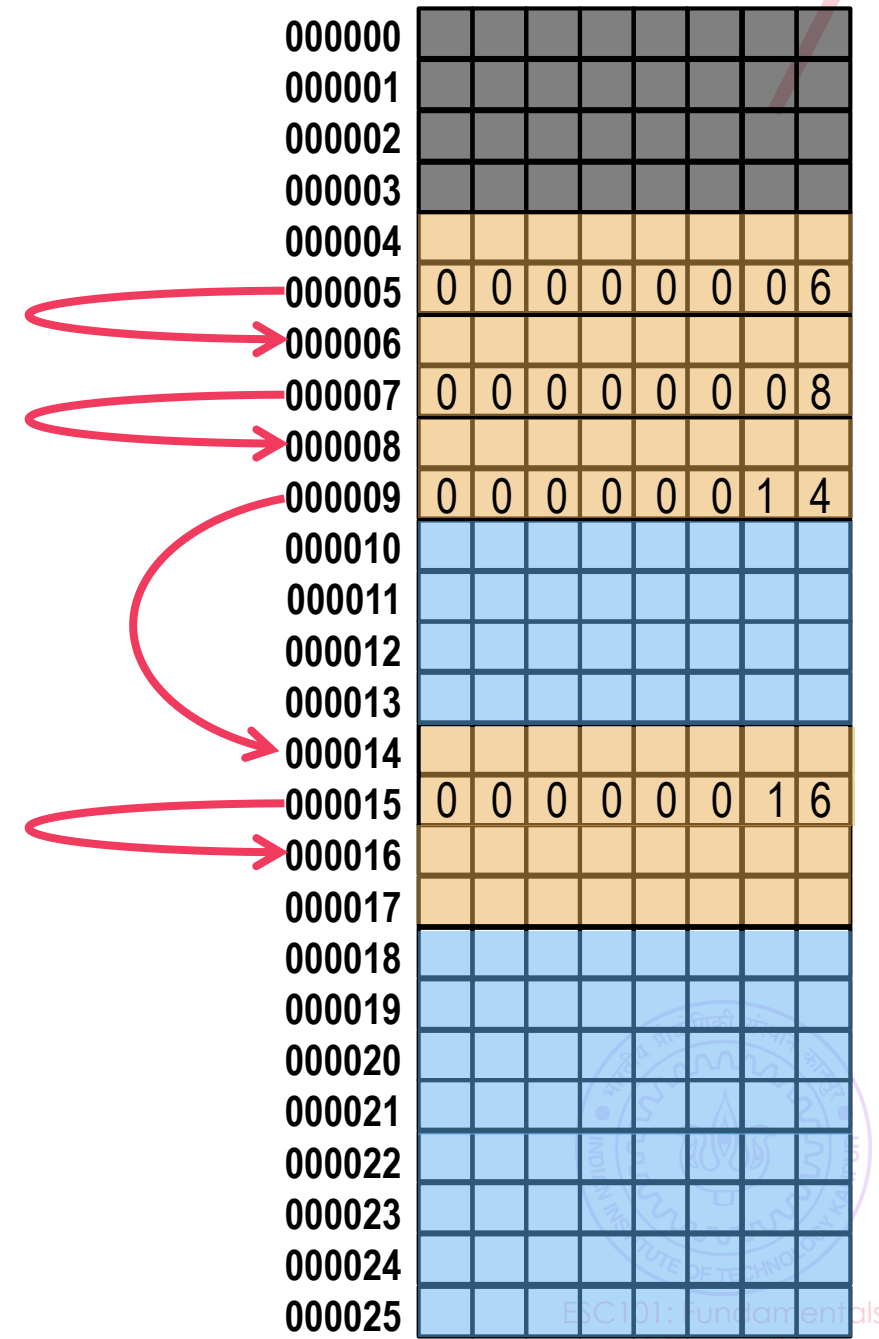
Linked Lists



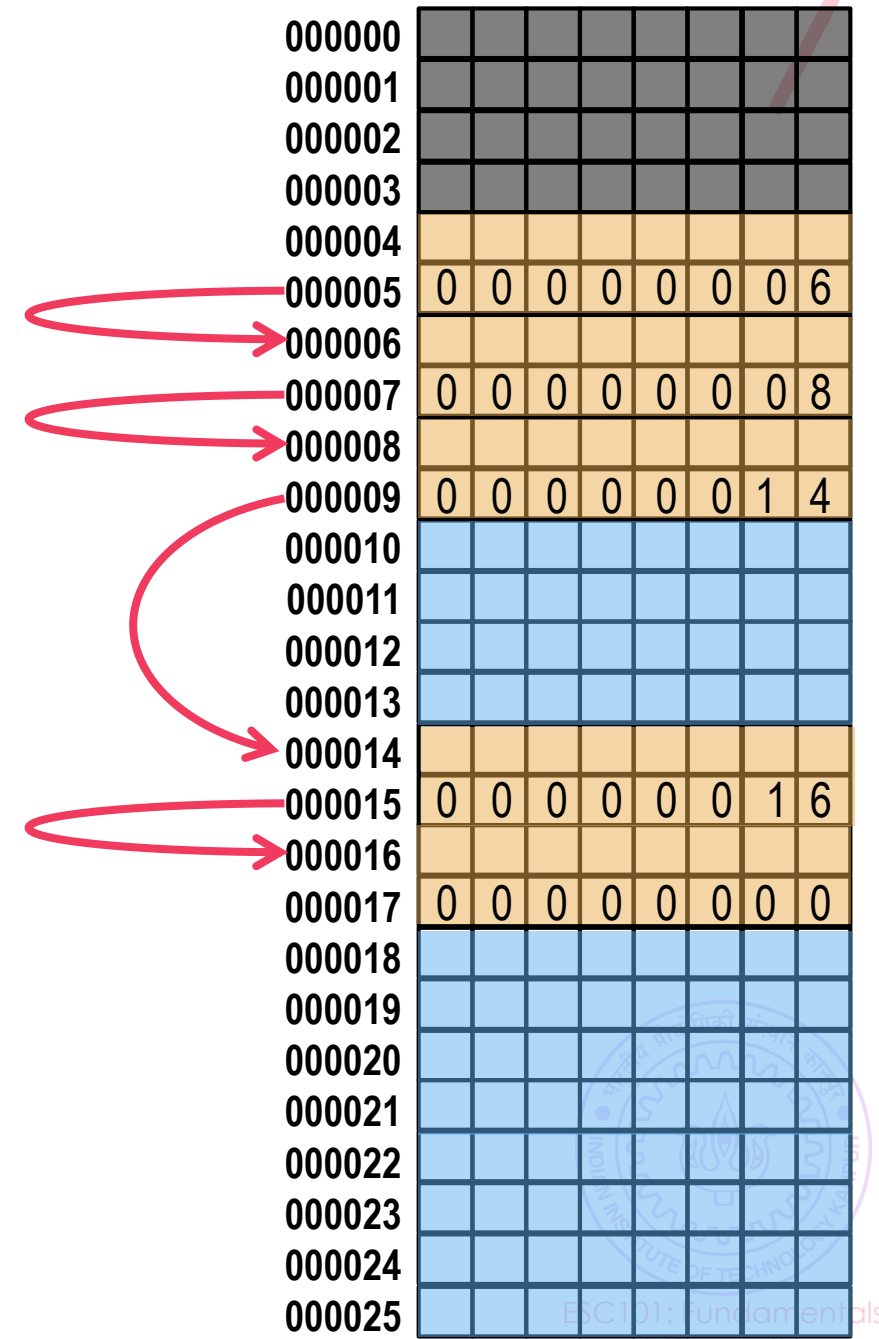
Linked Lists



Linked Lists

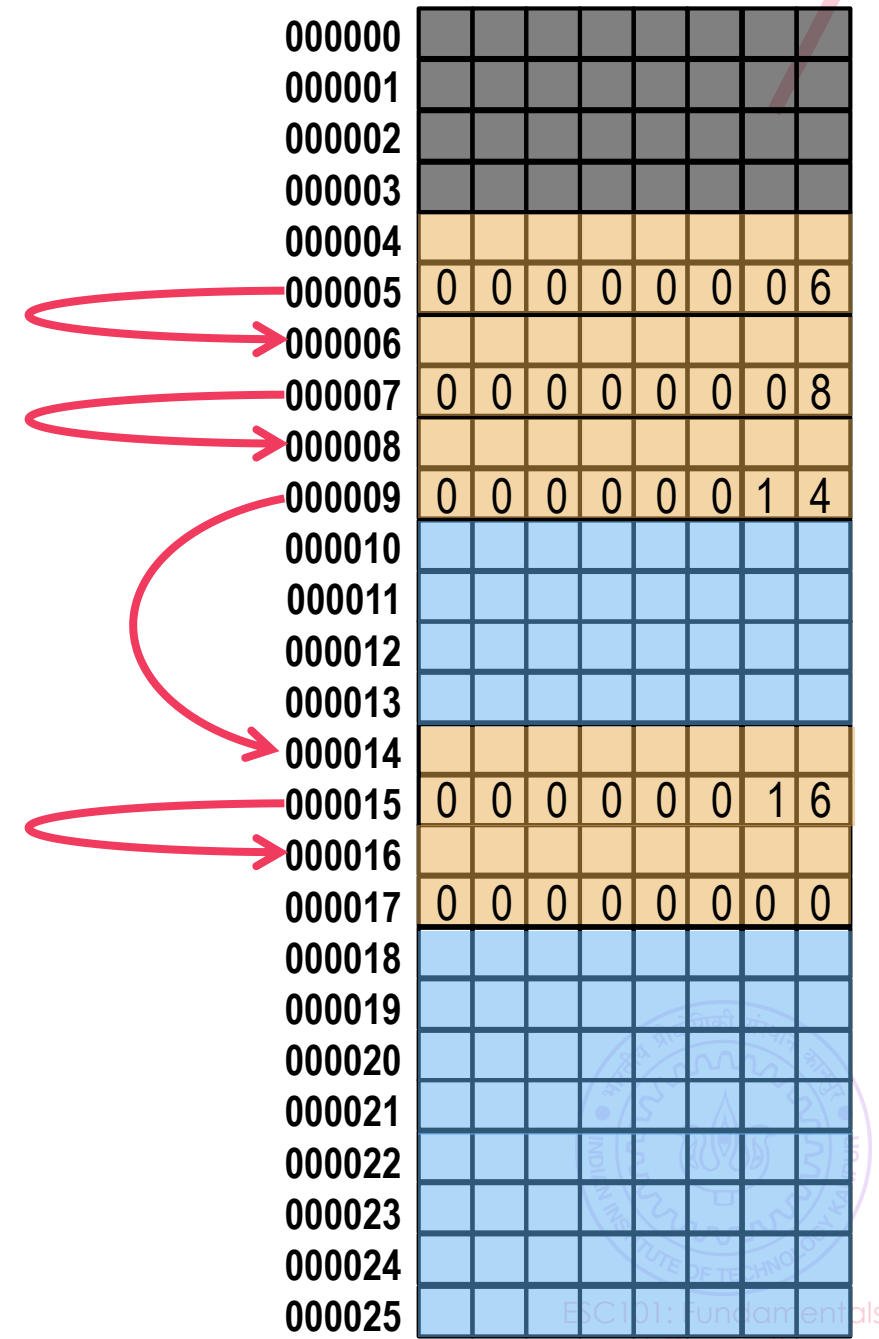


Linked Lists



Linked Lists

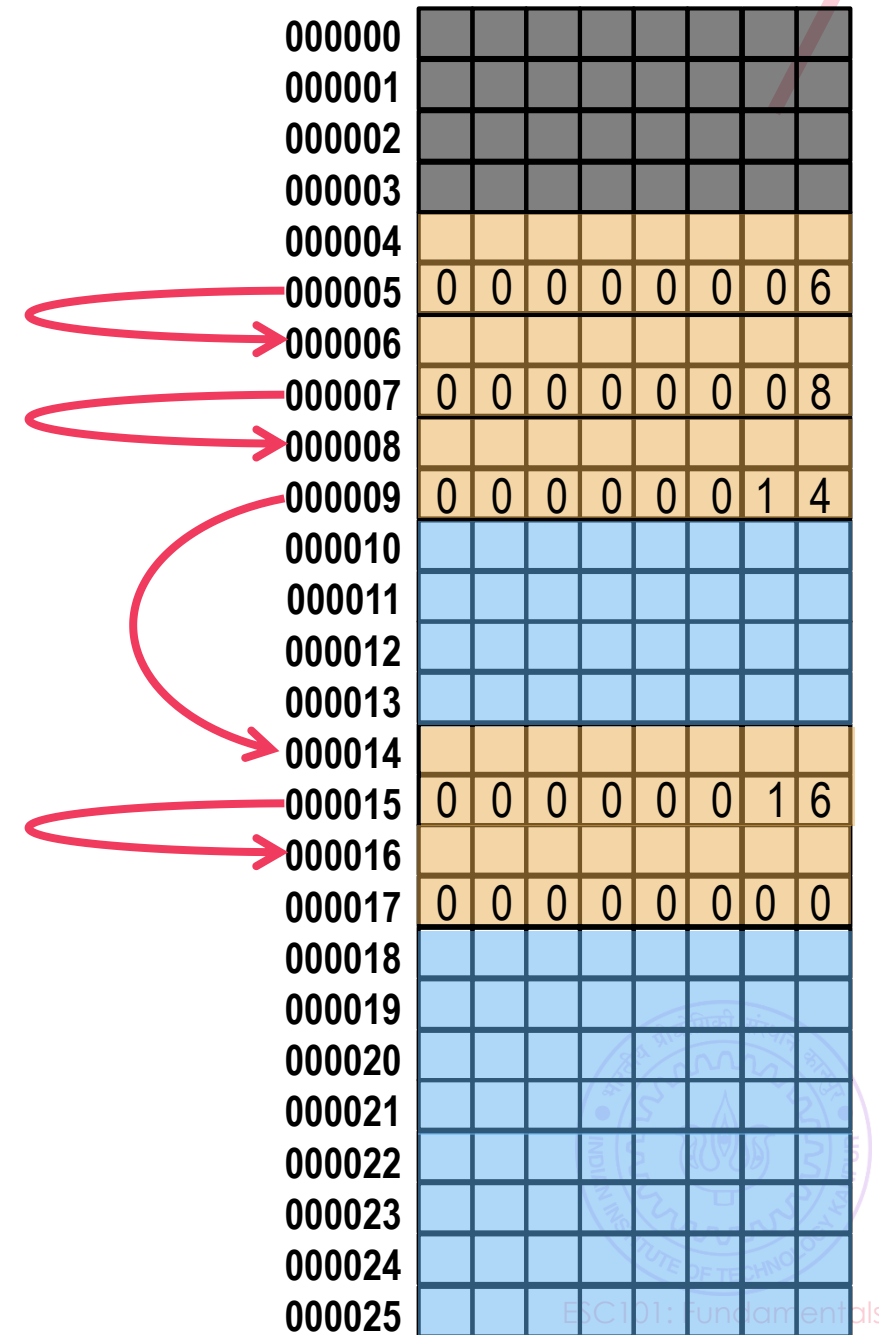
Allow for more efficient
usage of space



Linked Lists

Allow for more efficient
usage of space

ADVANTAGES

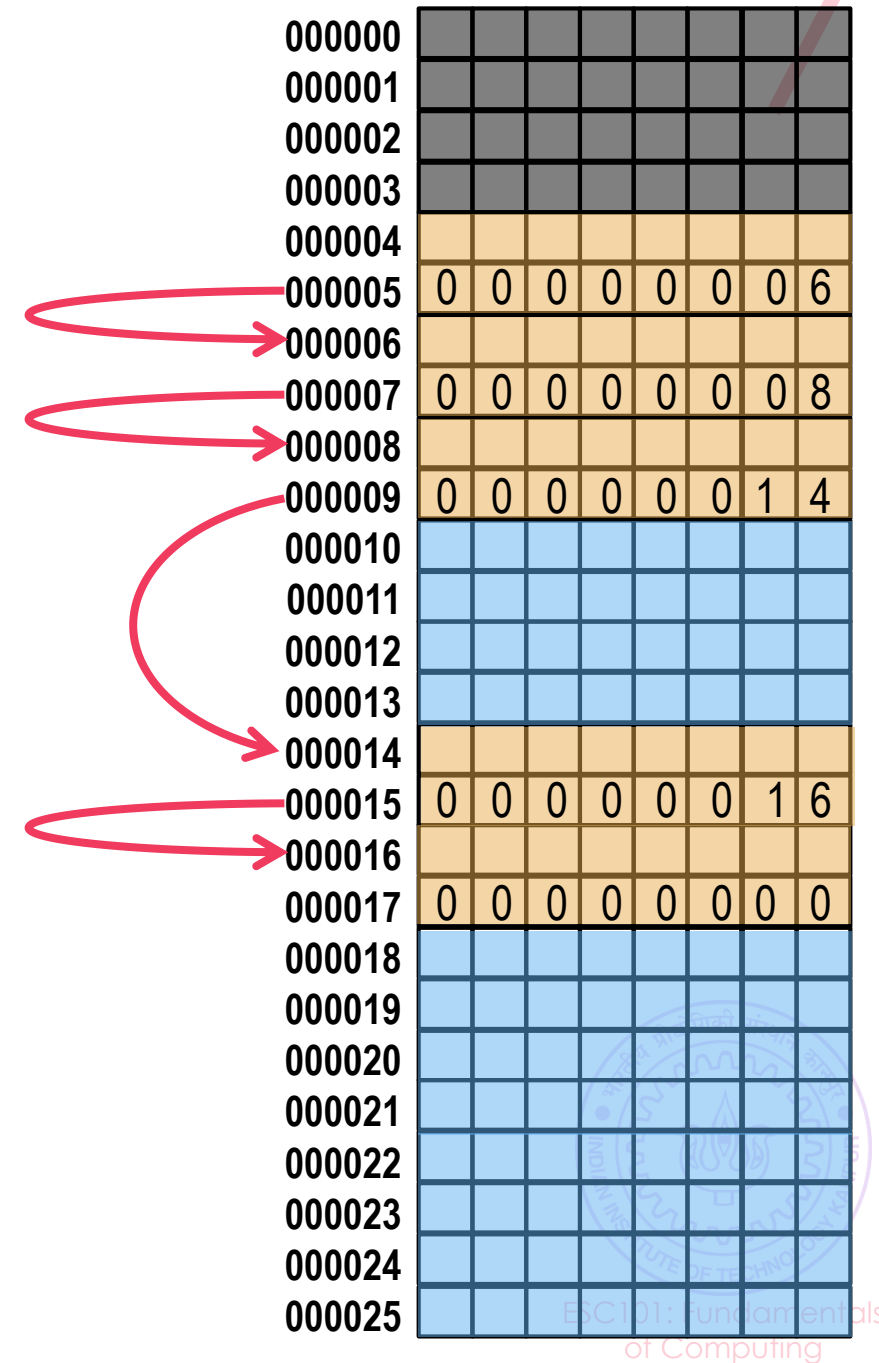


Linked Lists

Allow for more efficient
usage of space

ADVANTAGES

Allow as many elements as you want

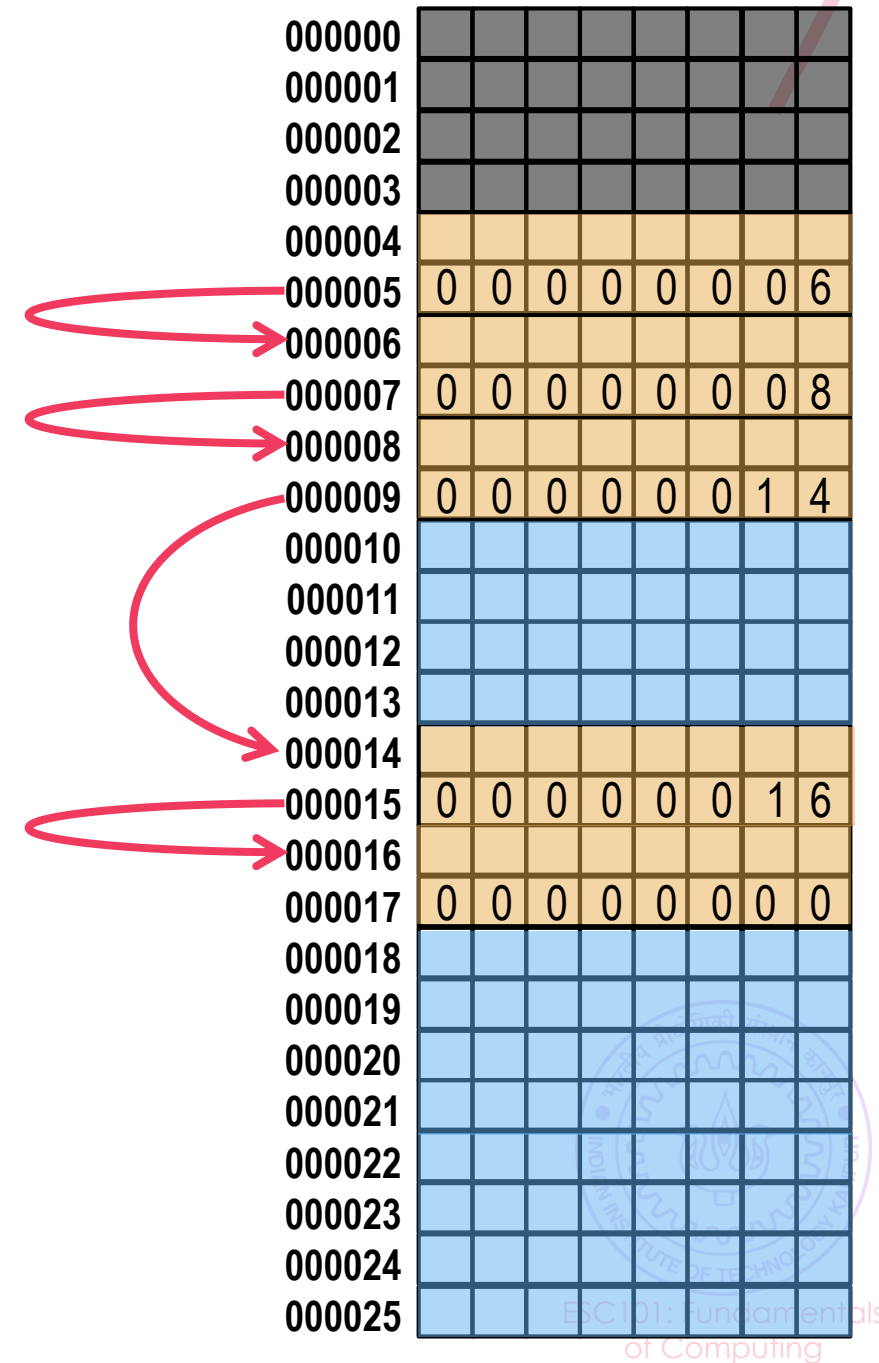


Linked Lists

Allow for more efficient
usage of space

ADVANTAGES

Allow as many elements as you want
Do not require contiguous space to
be available – pack things better

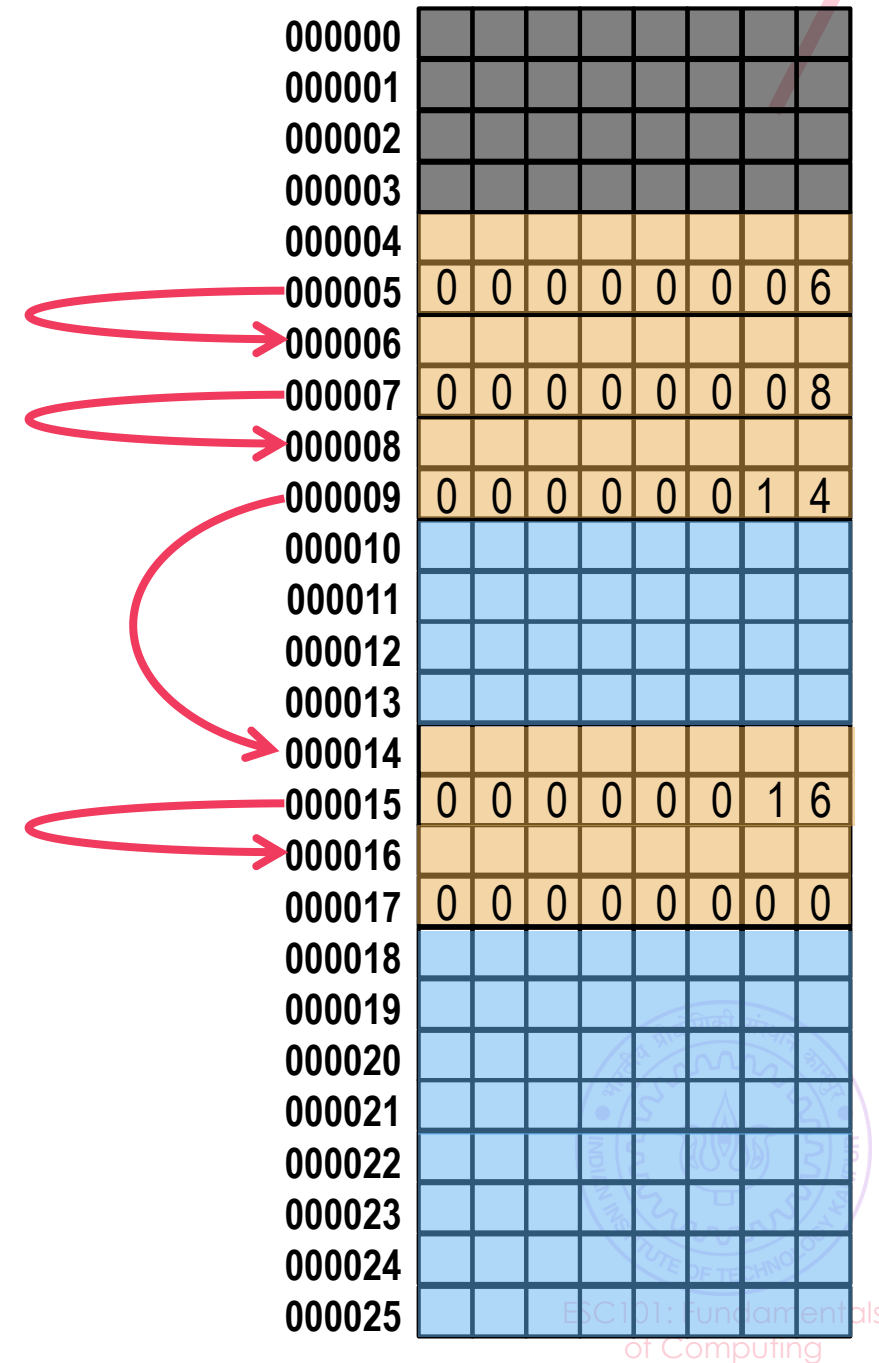


Linked Lists

Allow for more efficient
usage of space

ADVANTAGES

- Allow as many elements as you want
- Do not require contiguous space to be available – pack things better
- Can expand without calling realloc

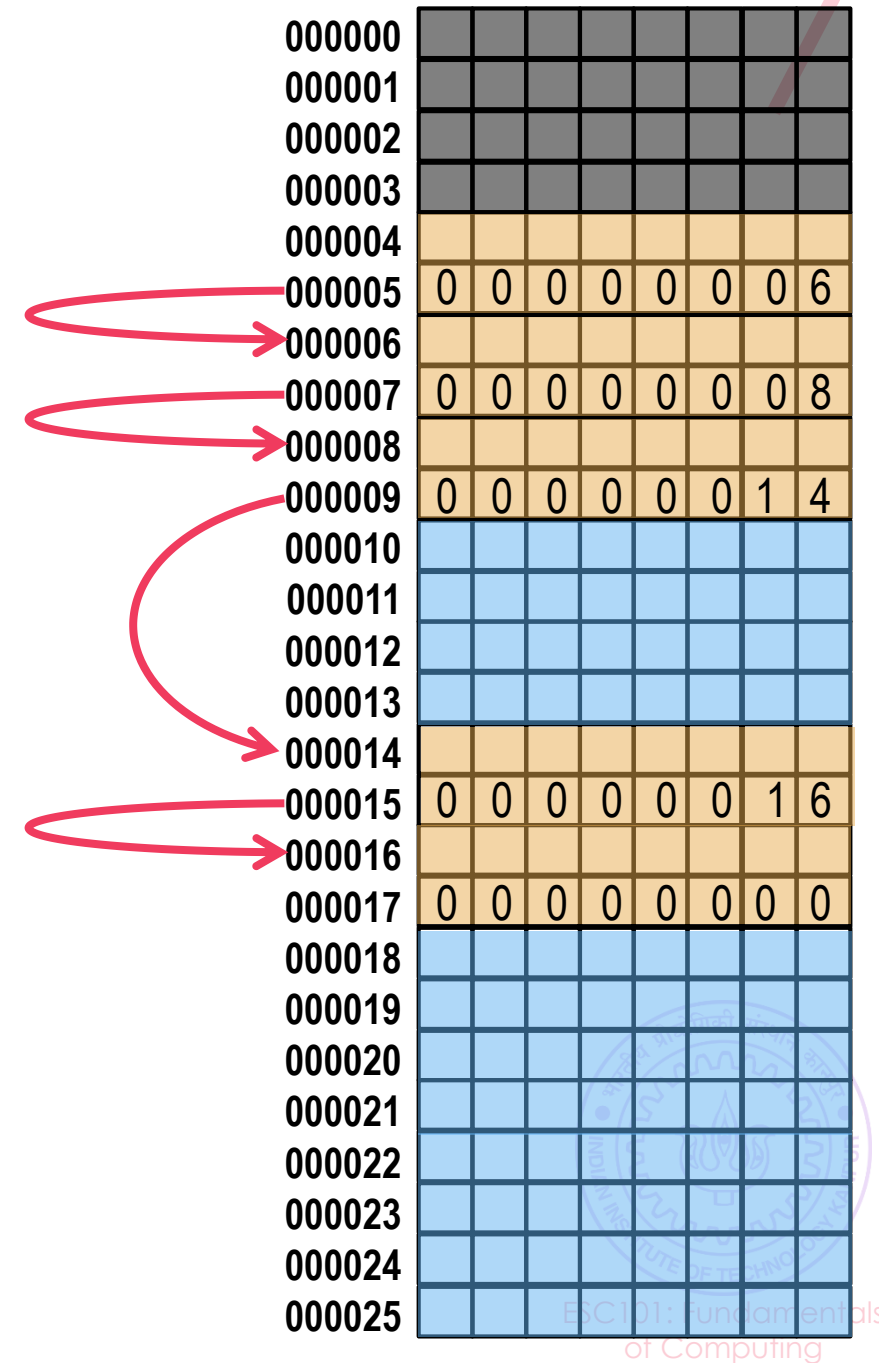


Linked Lists

Allow for more efficient usage of space

ADVANTAGES

- Allow as many elements as you want
- Do not require contiguous space to be available – pack things better
- Can expand without calling realloc
- Inserting in the middle very simple



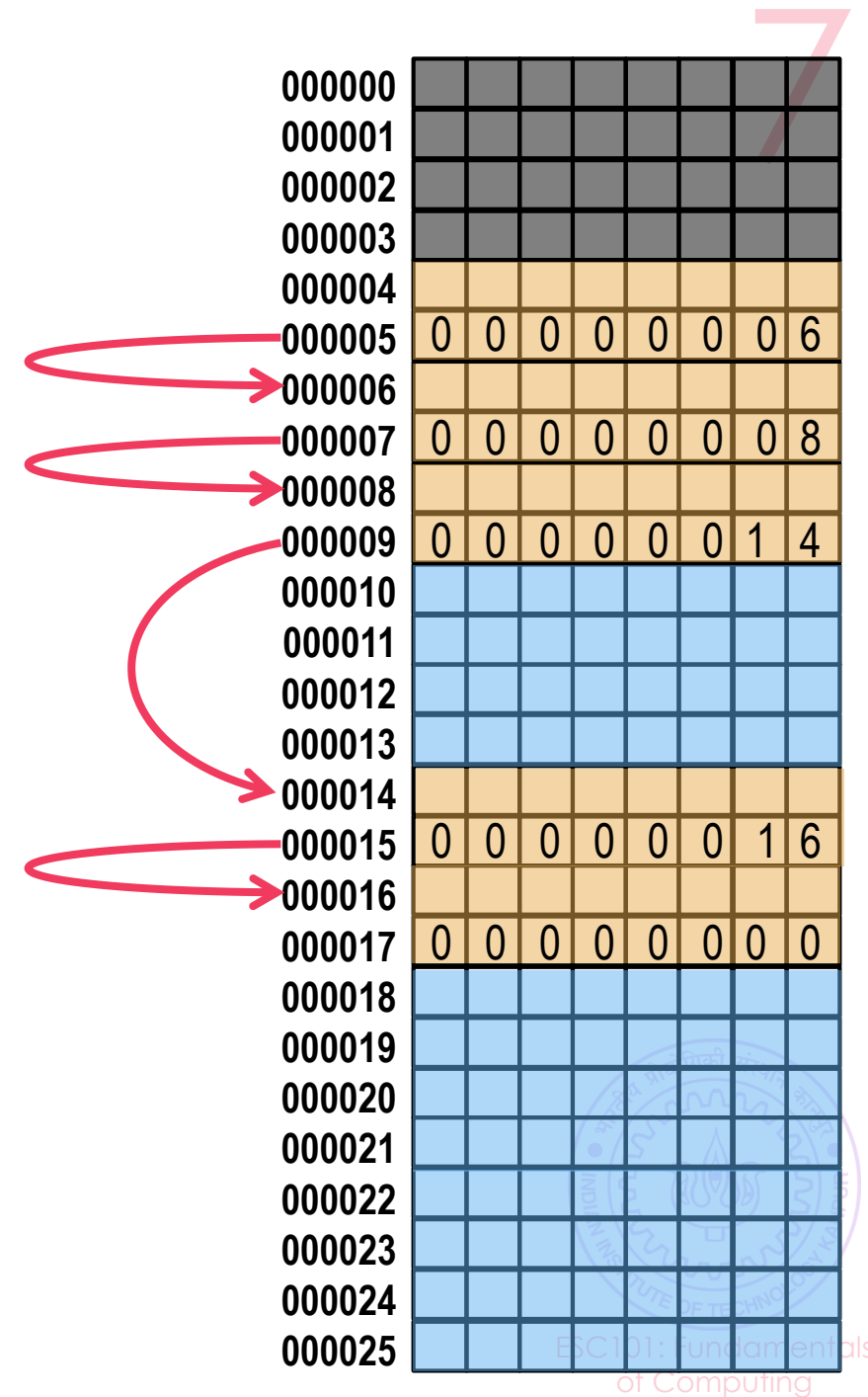
Linked Lists

Allow for more efficient usage of space

ADVANTAGES

- Allow as many elements as you want
- Do not require contiguous space to be available – pack things better
- Can expand without calling realloc
- Inserting in the middle very simple

DISADVANTAGES



Linked Lists

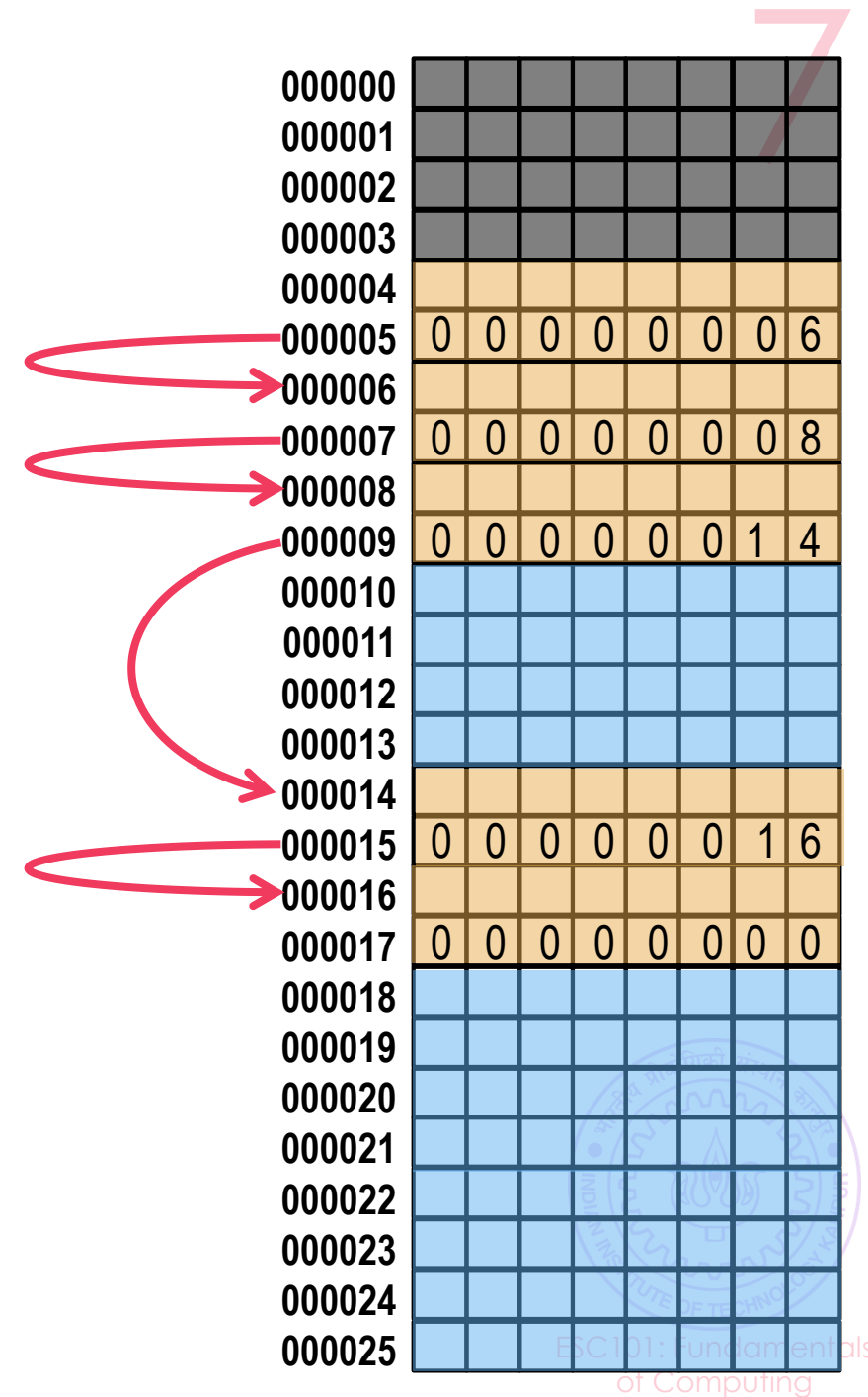
Allow for more efficient usage of space

ADVANTAGES

- Allow as many elements as you want
- Do not require contiguous space to be available – pack things better
- Can expand without calling realloc
- Inserting in the middle very simple

DISADVANTAGES

- No convenient “names” for elements



Linked Lists

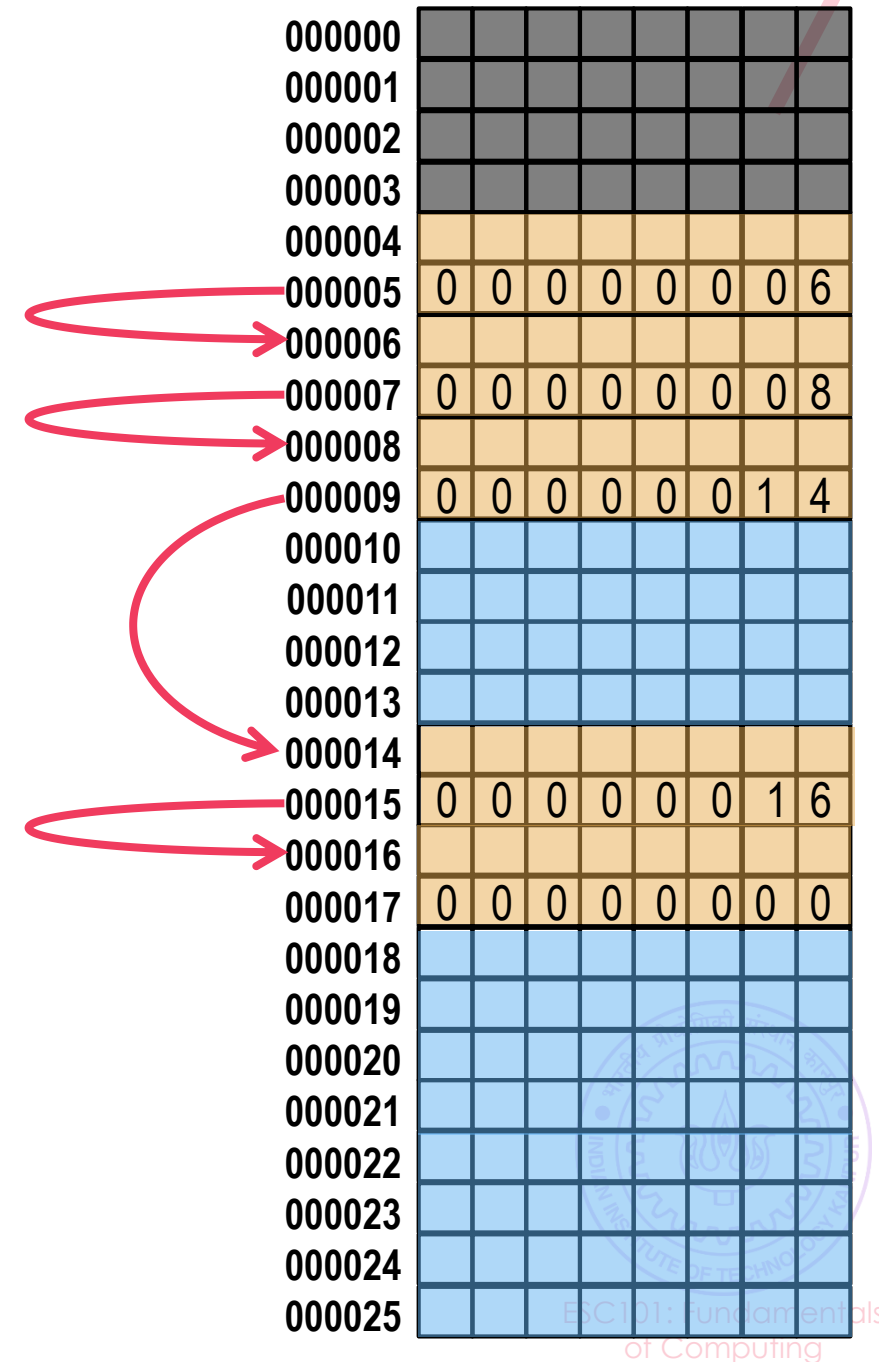
Allow for more efficient usage of space

ADVANTAGES

- Allow as many elements as you want
- Do not require contiguous space to be available – pack things better
- Can expand without calling realloc
- Inserting in the middle very simple

DISADVANTAGES

- No convenient “names” for elements
- Accessing n-th element slow – require going through first n-1 elements



Linked Lists

Allow for more efficient usage of space

ADVANTAGES

- Allow as many elements as you want
- Do not require contiguous space to be available – pack things better
- Can expand without calling realloc
- Inserting in the middle very simple

DISADVANTAGES

- No convenient “names” for elements
- Accessing n-th element slow – require going through first n-1 elements
- Setting them up requires more work

