Embedded Operating Systems

Circular buffer

protoCore 0.00001

Prof. Otávio Gomes

otavio.gomes@unifei.edu.br

in /otavio-gomes

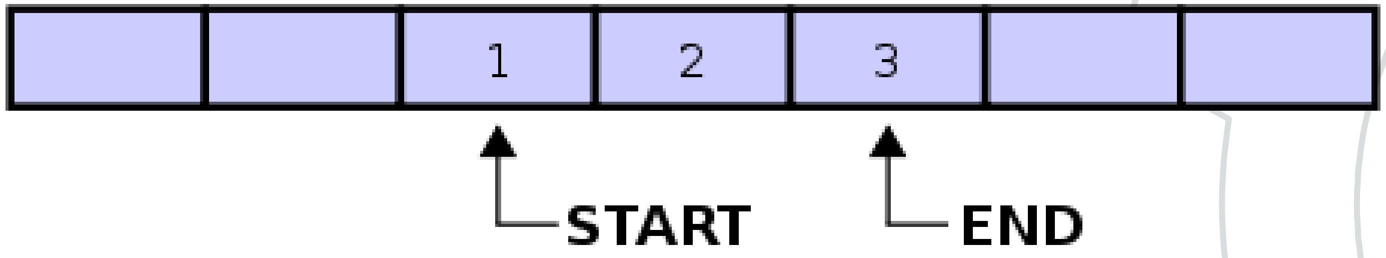
Circular buffers

Circular Buffers

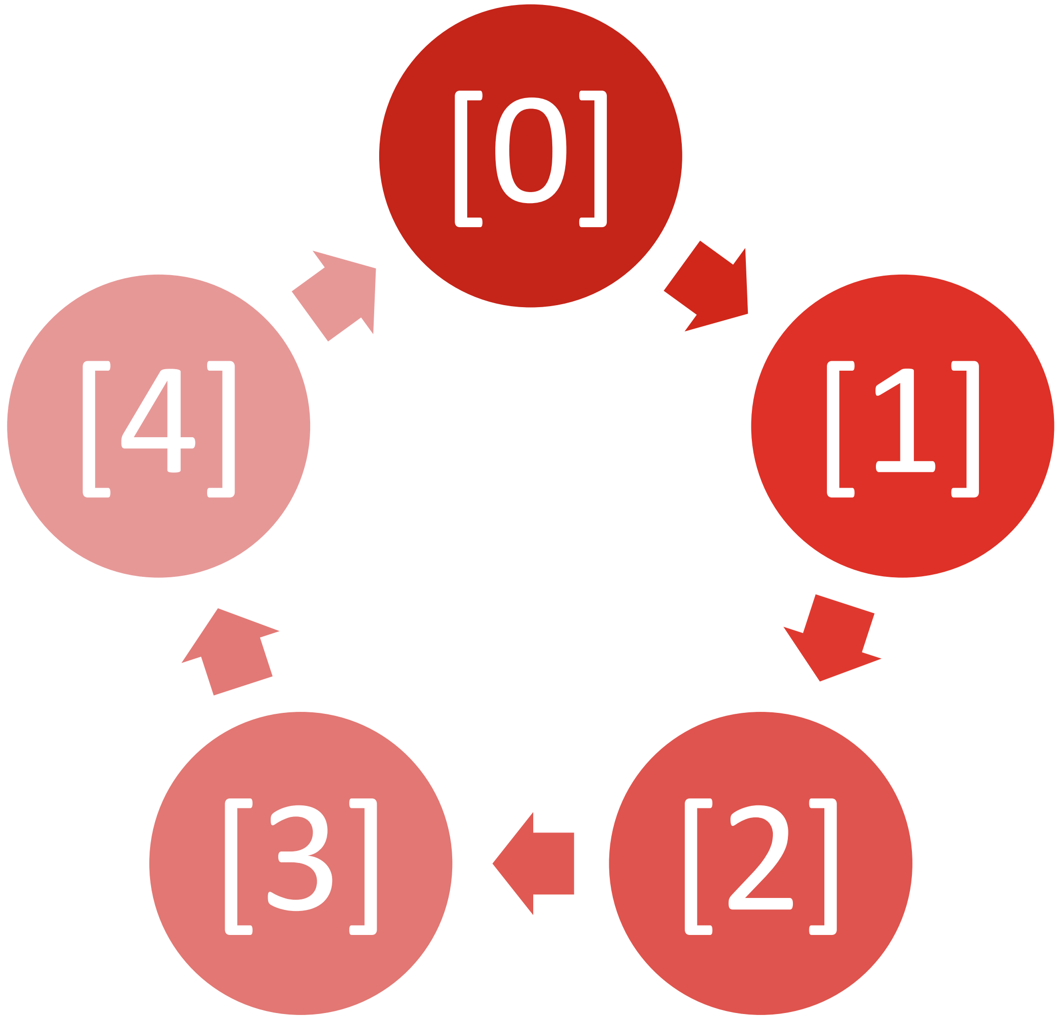
- “Endless” memory spaces
- Use FIFO aproach
- Store temporary data
- Can implemented using vectors or linked-lists

Circular Buffers

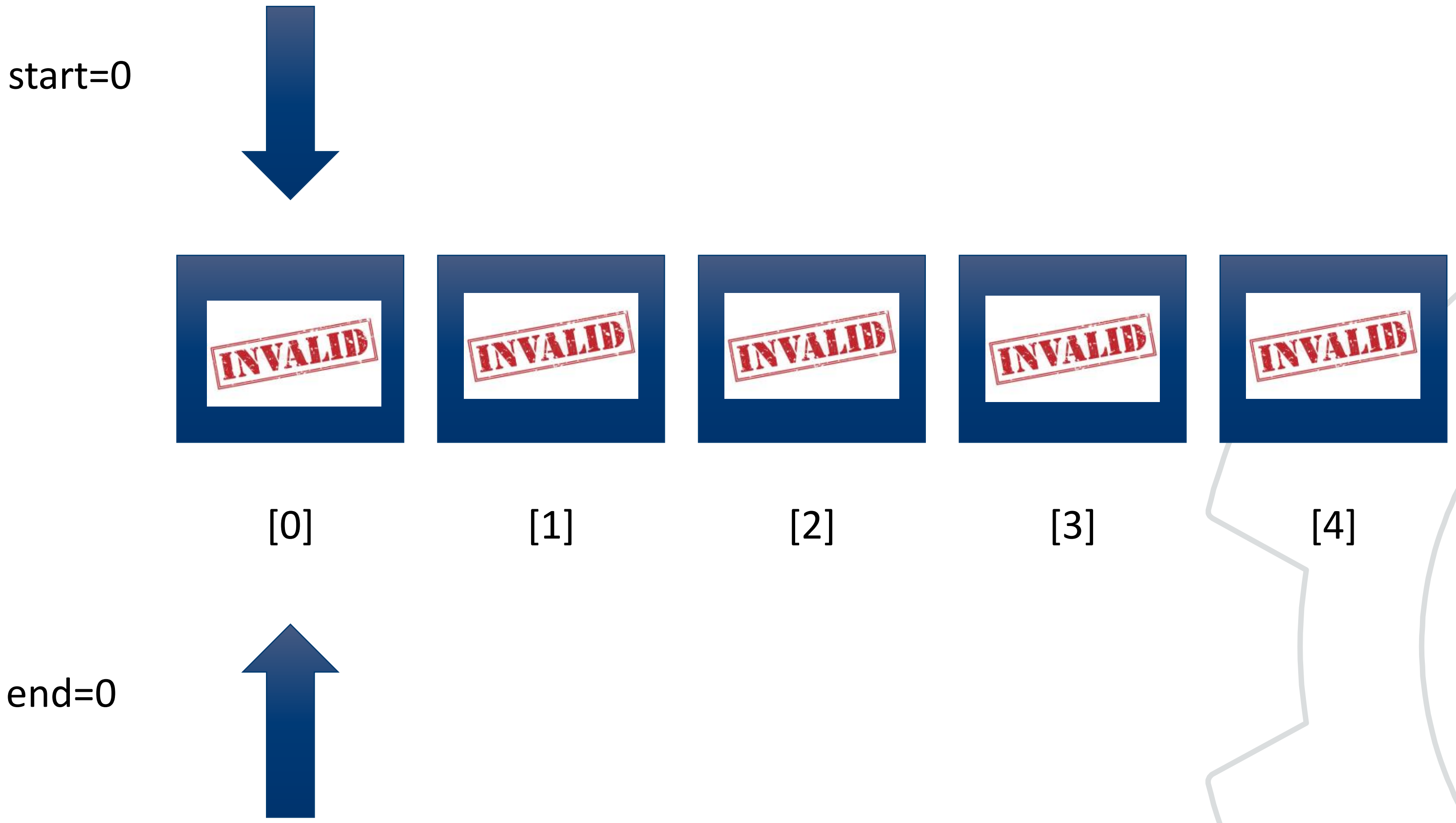
- Vector implementation
  - Uses less space
  - Need special caution when cycling
  - Problem to differentiate full from empty



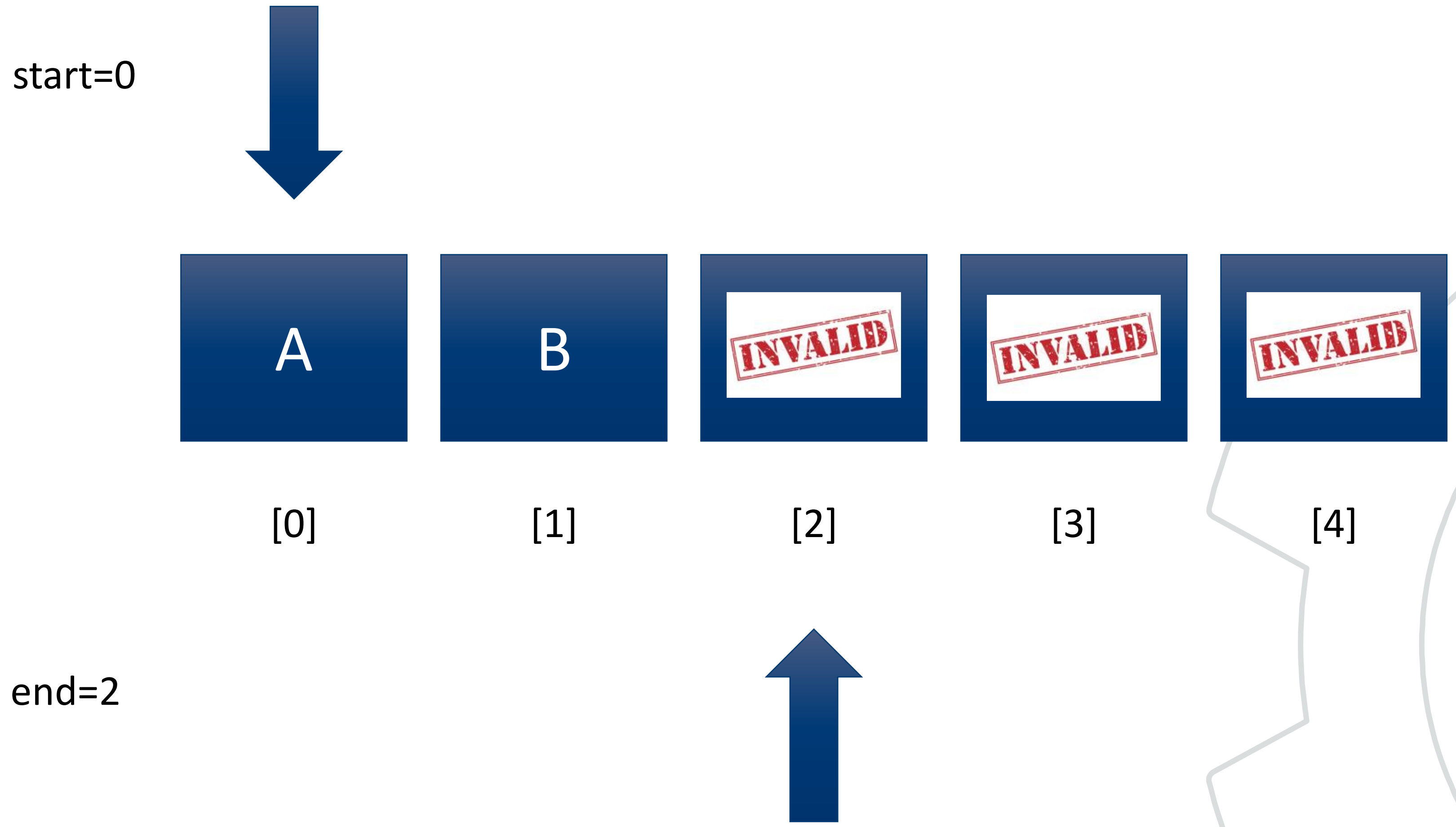
Circular Buffers



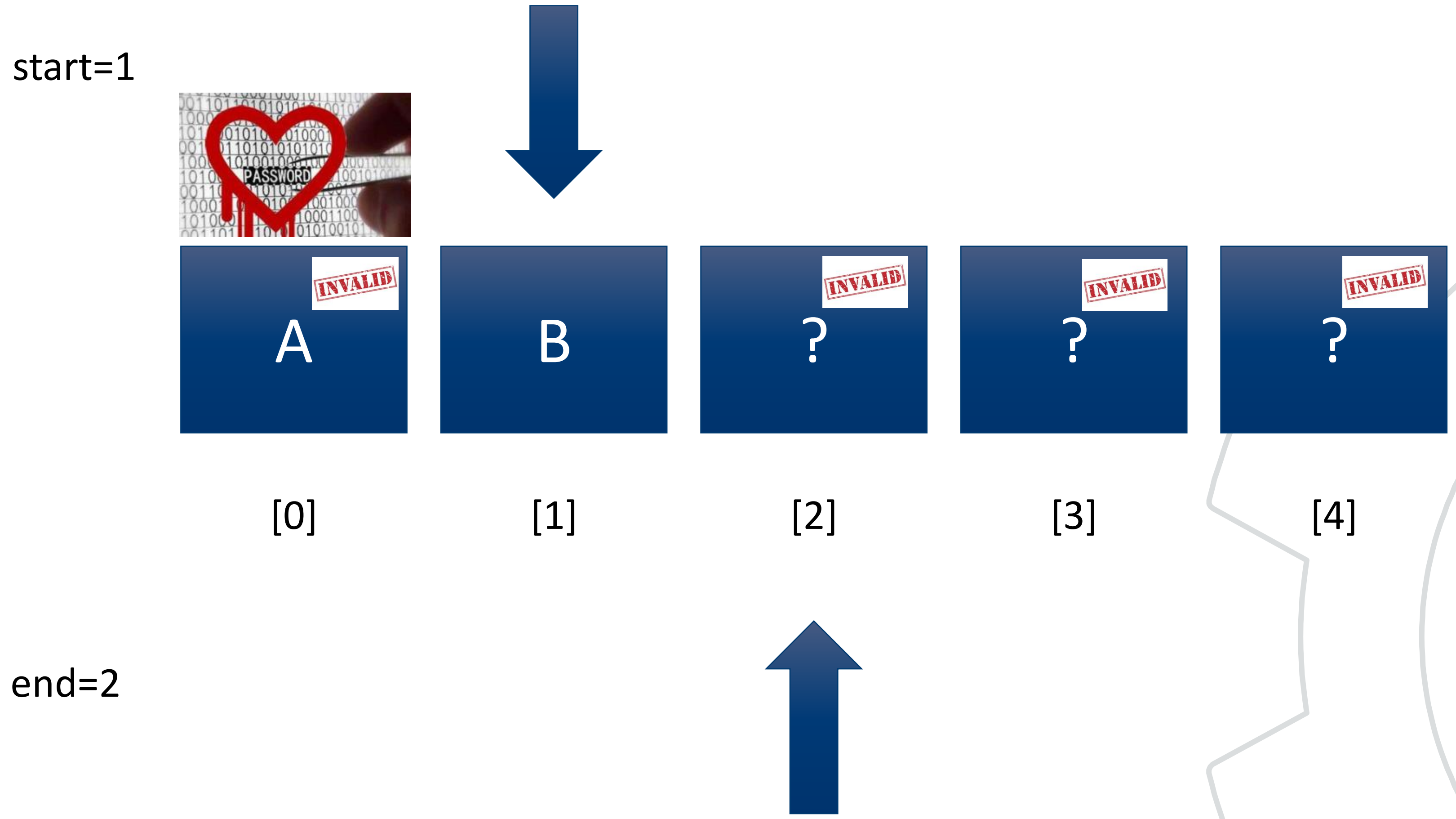
Empty Buffer



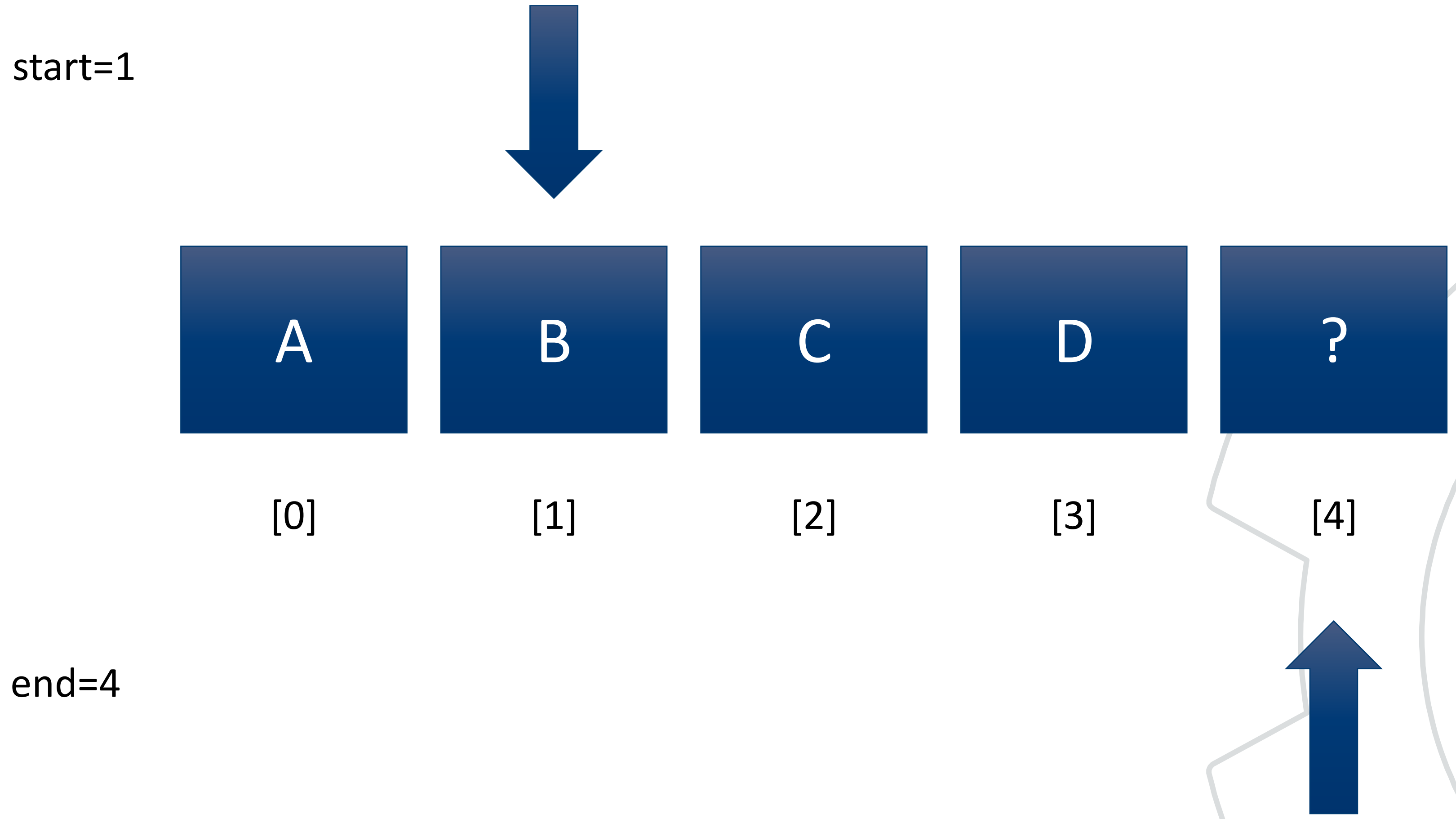
Adding 2 elements



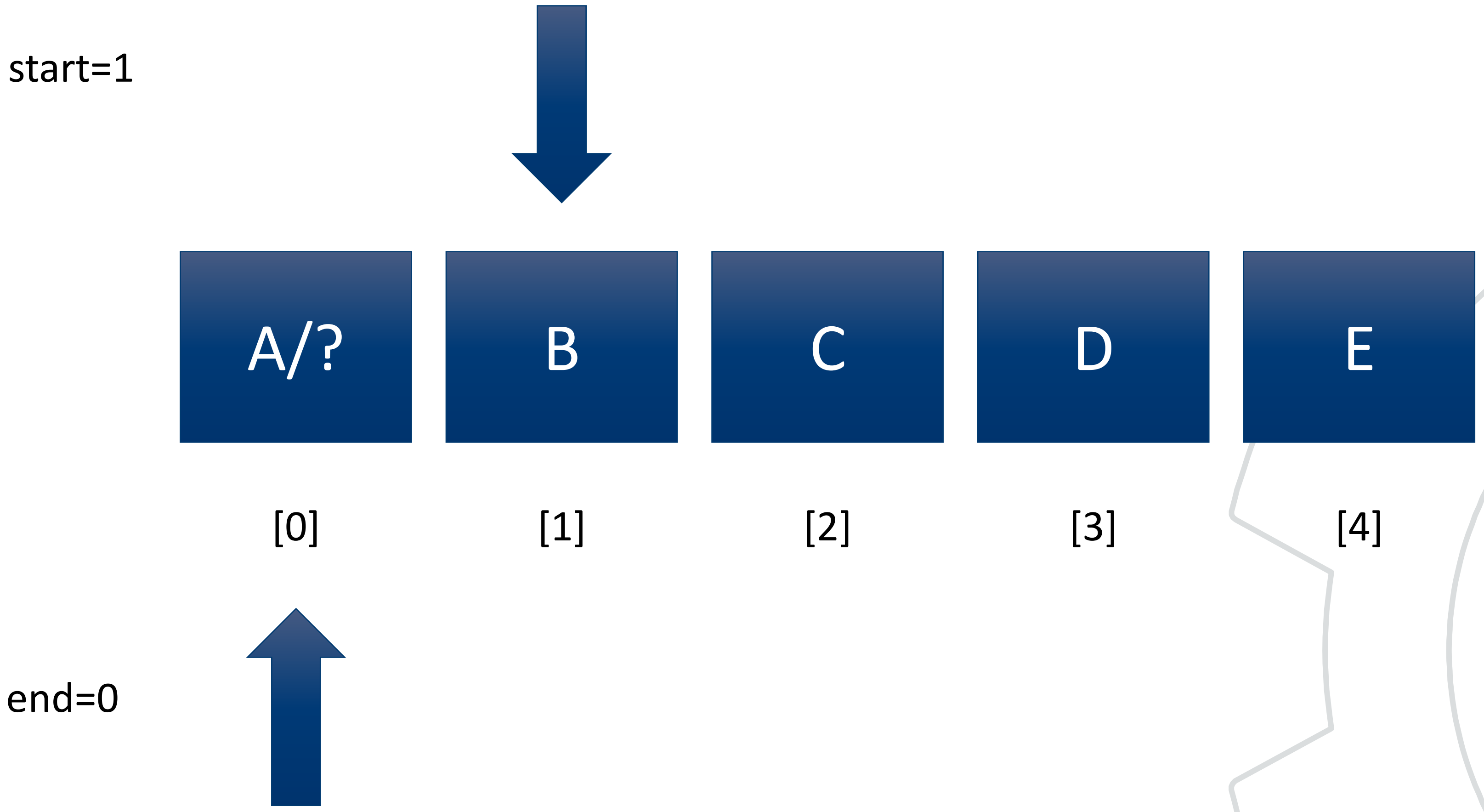
Removing 1 element



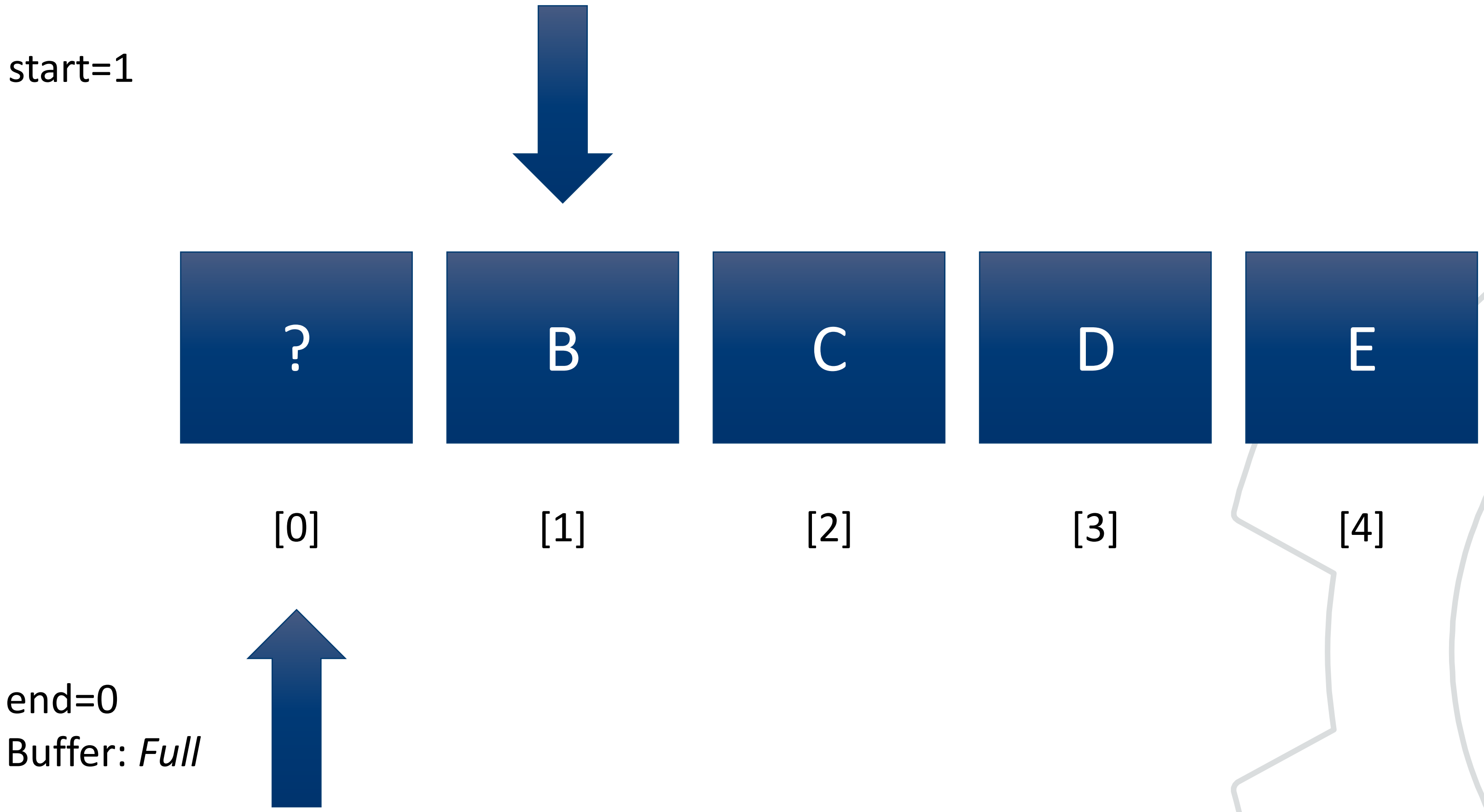
Adding 2 elements



Adding 1 element



Full Buffer



```
#define CB_SIZE 10

int circular_buffer[CB_SIZE];

int index=0;

for(;;){
    //do anything with the buffer
    circular_buffer[index] = index;
    //increment the index
    index = (index+1)%CB_SIZE;
}
```

Circular buffers

```c
#define CB_SIZE 10

int circular_buffer[CB_SIZE];

int index=0;

for(;;){
    //do anything with the buffer
    circular_buffer[index] = index;
    //increment the index
    index = (index+1)%CB_SIZE; //index++;
}
```

Circular buffers

```c
#define CB_SIZE 10
int circular_buffer[CB_SIZE];
int start=0, end=0;

char AddBuff(int newData)
{
  //check if there is space to add a number
  if ( ((end+1)%CB_SIZE) != start)
  {
    circular_buffer[end] = newData;
    end = (end+1)%CB_SIZE;
    return SUCCESS;
  }
  return FAIL;
}
```

Circular buffers

```
#define CB_SIZE 10
int circular_buffer[CB_SIZE];
int start=0, end=0;

char AddBuff(int newData)
{
  //check if there is space to add a number
  if ( ((end+1)%CB_SIZE) != start)
  {
    circular_buffer[end] = newData;
    end = (end+1)%CB_SIZE;
    return SUCCESS;
  }
  return FAIL;
}
```

Exercise

- Implement a circular buffer
  - Use a 10-position vector
- Each element of the vector is a structure with two variables
  - `char * ProcessName;`
  - `int Time;`

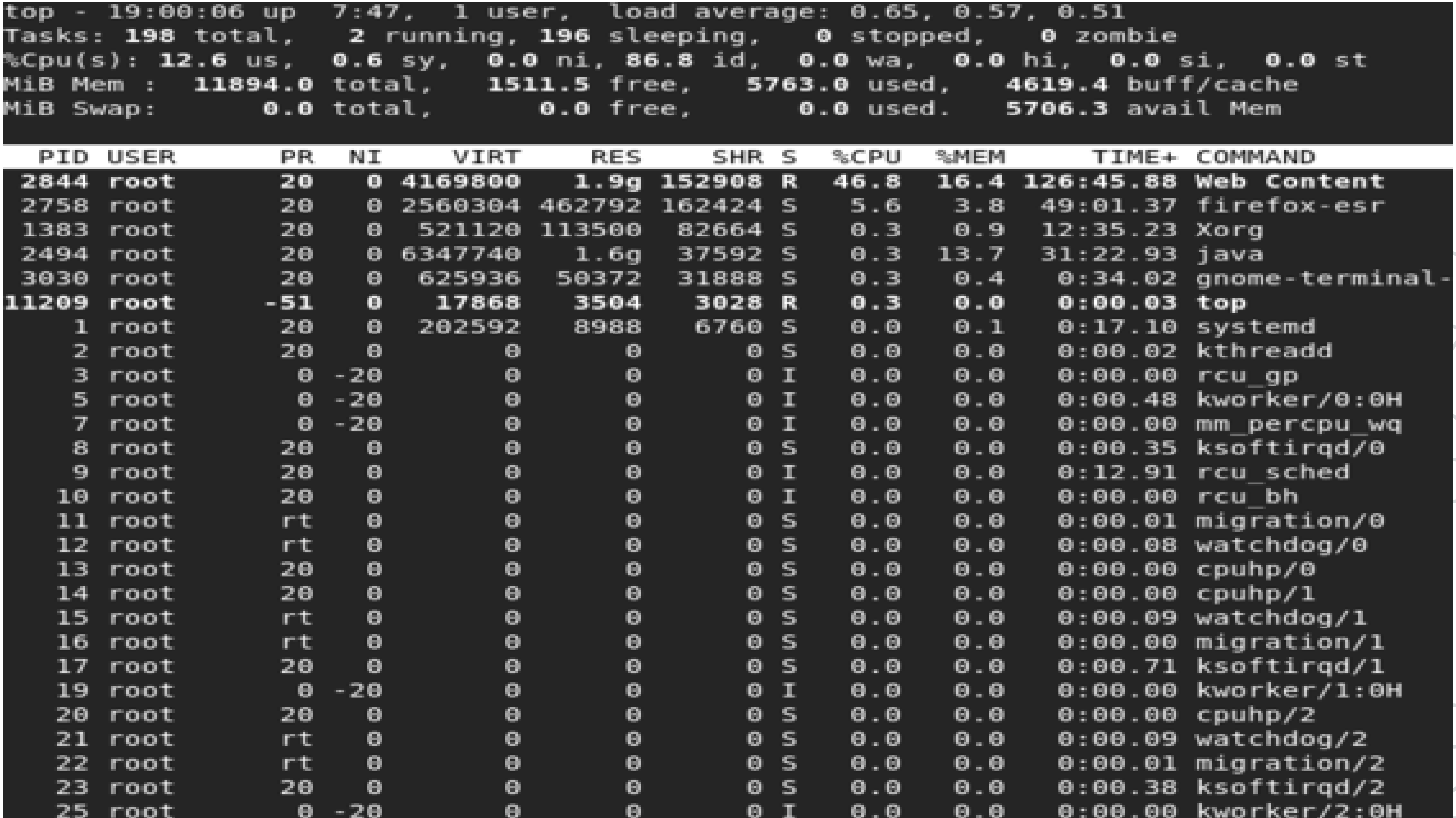
- Create one function to add new elements and one to remove the oldest elements.

```c
typedef struct {
    char* processName;
    int time;
}process;

//circular buffer declaration
#define BUFFERSIZE  10
process buffer[BUFFERSIZE];

//Declaration of access pointers
int start=0, end=0;
```

Exercise



```
top - 19:00:06 up  7:47,  1 user,  load average: 0.65, 0.57, 0.51
Tasks: 198 total,   2 running, 196 sleeping,   0 stopped,   0 zombie
%Cpu(s): 12.6 us,  0.6 sy,  0.0 ni, 86.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  11894.0 total,   1511.5 free,   5763.0 used,   4619.4 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   5706.3 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 2844 root      20   0 4169800   1.9g 152908 R  46.8  16.4 126:45.88 Web Content
 2758 root      20   0 2560304 462792 162424 S   5.6   3.8  49:01.37 firefox-esr
 1383 root      20   0  521120 113500  82664 S   0.3   0.9  12:35.23 Xorg
 2494 root      20   0 6347740   1.6g  37592 S   0.3  13.7  31:22.93 java
 3030 root      20   0  625936  50372  31888 S   0.3   0.4   0:34.02 gnome-terminal-
11209 root     -51   0   17868   3504   3028 R   0.3   0.0   0:00.03 top
    1 root      20   0  202592   8988   6760 S   0.0   0.1   0:17.10 systemd
    2 root      20   0       0      0      0 S   0.0   0.0   0:00.02 kthreadd
    3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
    5 root       0 -20       0      0      0 I   0.0   0.0   0:00.48 kworker/0:0H
    7 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
    8 root      20   0       0      0      0 S   0.0   0.0   0:00.35 ksoftirqd/0
    9 root      20   0       0      0      0 I   0.0   0.0   0:12.91 rcu_sched
   10 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_bh
   11 root      rt   0       0      0      0 S   0.0   0.0   0:00.01 migration/0
   12 root      rt   0       0      0      0 S   0.0   0.0   0:00.08 watchdog/0
   13 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
   14 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
   15 root      rt   0       0      0      0 S   0.0   0.0   0:00.09 watchdog/1
   16 root      rt   0       0      0      0 S   0.0   0.0   0:00.00 migration/1
   17 root      20   0       0      0      0 S   0.0   0.0   0:00.71 ksoftirqd/1
   19 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/1:0H
   20 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/2
   21 root      rt   0       0      0      0 S   0.0   0.0   0:00.09 watchdog/2
   22 root      rt   0       0      0      0 S   0.0   0.0   0:00.01 migration/2
   23 root      20   0       0      0      0 S   0.0   0.0   0:00.38 ksoftirqd/2
   25 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/2:0H
```

```
//Function to add process to buffer
void addProc(char *nname, int ntime){

  //Verification of position (full?)
  if ( ((end+1)%BUFFERSIZE) != start){
    //Current position update
     buffer[end].processName = nname;
     buffer[end].time = ntime;
    //increment circular buffer position
     end = (end+1)%(BUFFERSIZE);
  }
}
```

Exercise

```c
//Function to add process to buffer
void addProc(char *nname, int ntime){

  //Verification of position (full?)
  if ( ((end+1)%BUFFERSIZE) != start){
    //Current position update
    buffer[end].processName = nname;
    buffer[end].time = ntime;
    //increment circular buffer position
    end = (end+1)%(BUFFERSIZE);
  }
}
```

Exercise

```c
//Function to add process to buffer
void addProc(char *nname, int ntime){

  //Verification of position (full?)
   if ( ((end+1)%BUFFERSIZE) != start){
    //Current position update
     buffer[end].processName = nname;
     buffer[end].time = ntime;
    //increment circular buffer position
     end = (end+1)%(BUFFERSIZE);
   }
}
```

Exercise

TO DO: Try/Catch or SUCCESS/FAIL

```
//Function to remove process from buffer
void removeProc (void){

  //Verification of position (empty?)
   if (end != start){
     //increment circular buffer start position
       start = (start +1)%(BUFFERSIZE);
   }

}
```

Exercise

```c
#include "stdio.h"
void main (void){
  addProc("proc1", 0);
  addProc("proc2", 1);
  addProc("proc3", 2);
  removeProc();
  removeProc();
  removeProc();
}
```

Exercise

Software engines

Kernel – abstraction and management

Software engines

- **Goal:**
  - Make an image editor that can choose the right function to call
- 1st Implementation
  - Use a option parameter as a switch operator

```
image Blur(image nImg){}
image Sharpen(image nImg){}

image imageEditorEngine(image nImg, int opt){
  image temp;
  switch(opt){
    case 1:
      temp = Sharpen(nImg);
      break;
    case 2:
      temp = Blur(nImg);
      break;
  }
  return temp;
}
```

Software engines

Software engines

```
image Blur(image nImg){}
image Sharpen(image nImg){}

image imageEditorEngine(image nImg, int opt){
  image temp;
  switch(opt){
    case 1:
      temp = Sharpen(nImg);
      break;
    case 2:
      temp = Blur(nImg);
      break;
  }
  return temp;
}
```

Why not?

```
Sharpen(nImg);
      or
Blur(nImg);
```

```
image Blur(image nImg){}
image Sharpen(image nImg){}

image imageEditorEngine(image nImg, int opt){
  image temp;
  switch(opt){
    case 1:
      temp = Sharpen(nImg);
      break;
    case 2:
      temp = Blur(nImg);
      break;
  }
  return temp;
}
```

Software engines

GET and **SET** methods

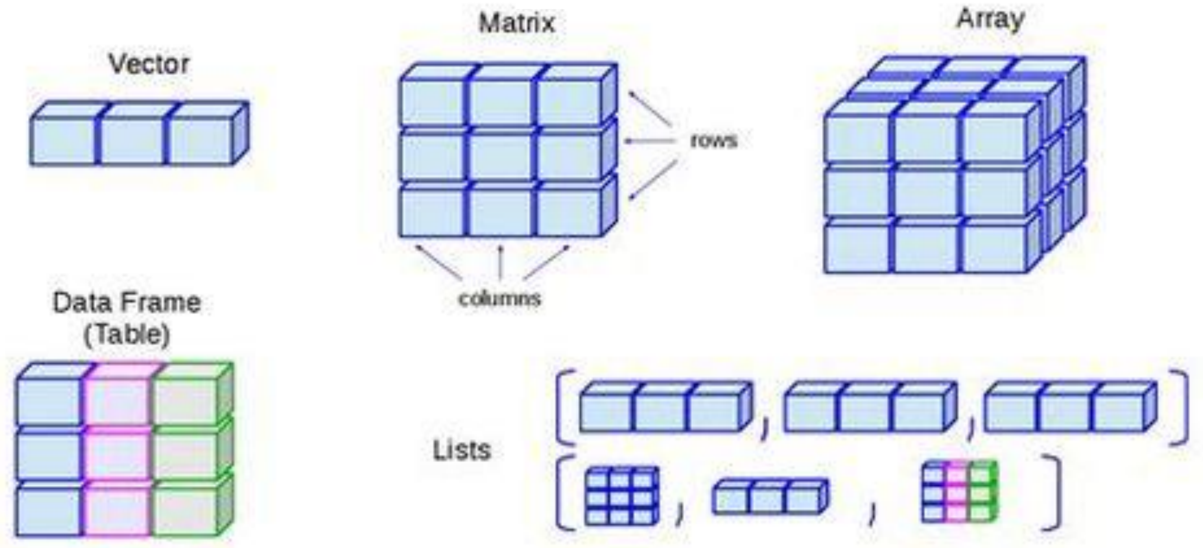
- Permissions
- Img Resolution
- File Type

Variables
Versus
Circular buffer

```
image Blur(image nImg){}
image Sharpen(image nImg){}

image imageEditorEngine(image nImg, int opt){
  image temp;
  switch(opt){
    case 1:
      temp = Sharpen(nImg);
      break;
    case 2:
      temp = Blur(nImg);
      break;
  }
  return temp;
}
```

Variables
Versus



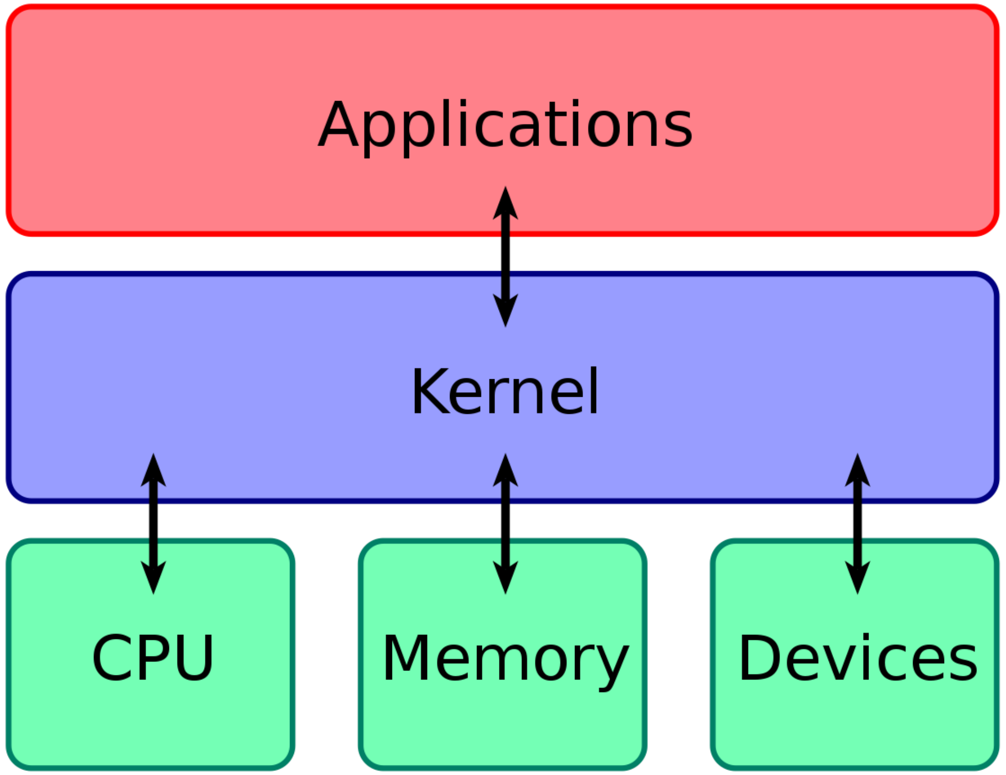
Easy to handle with a lot of data

Software engines

Function pointers

Problem

How to execute a function that **is not known at compile time**?



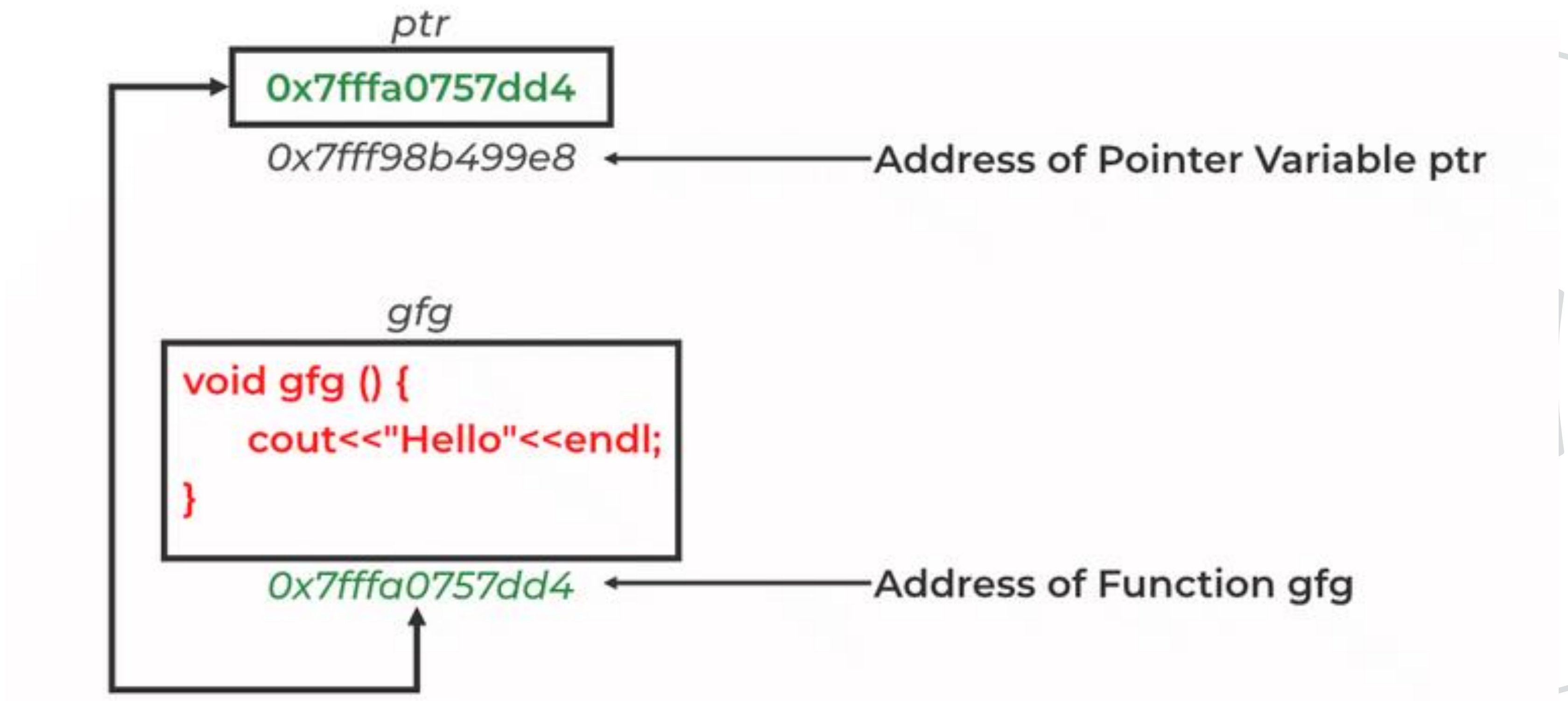
Problem

How to execute a function that **is not known at compile time**?

- Know the address of the function at runtime.
- Stack the parameters correctly that the function needs
- Make a function call to this address

Function pointers

- Work *almost* as a normal pointer
- Its manipulation obeys all pointer manipulation rules
- Hold the address of a function start point instead the address of a variable



```
void gfg () {
    cout<<"Hello"<<endl;
}
```

Function pointers

- Work *almost* as a normal pointer
- Its manipulation obeys all pointer manipulation rules
- Hold the address of a function start point instead the address of a variable
- The compiler <u>need no known the function signature</u> to pass the correct parameters and the return value.
- Awkard declaration (it is best to <u>**use a typedef**</u>)

Function pointers

```c
//defining the type pointerTest
//it is a pointer to function that:
//   receives no parameter
//   returns no parameter
void (*pointerTest)(void);
```

Function pointers

```
//defining the type pointerTest
//it is a pointer to function that:
//   receives no parameter
//   returns no parameter
void (*pointerTest)(void);

//Function to be called
void nop (void){ __asm NOP __endasm }

//creating an pointerTest variable;
(*pointerTest)(void) foo;
foo = nop; //foo receives the address of nop
(*foo)(); //calling the function via pointer
// or foo();
```

Function pointers

```
//defining the type pointerTest
//it is a pointer to function that:
//   receives no parameter
//   returns no parameter
// ~~void (*pointerTest)(void);~~
typedef void (*pointerTest)(void);
```

Function pointers

```
//defining the type pointerTest
//it is a pointer to function that:
//   receives no parameter
//   returns no parameter
typedef void (*pointerTest)(void);

//Function to be called
void nop (void){ __asm NOP __endasm }

//creating an pointerTest variable;
pointerTest foo;
foo = nop; //foo receives the address of nop
(*foo)(); //calling the function via pointer
// or foo();
```

Function pointers

Re-code the image editor engine using function pointers

Function pointers

```
image Blur(image nImg){}

image Sharpen(image nImg){}

typedef image (*ptrFunc)(image nImg);

//image editor engine
image imageEditorEngine(ptrFunc function,
                        image nImg){
    image temp;
    temp = (*function)(nImg);
    return temp;
}
```

Function pointers

```
image Blur(image nImg){}

image Sharpen(image nImg){}

typedef image (*ptrFunc)(image nImg);

//image editor engine
image imageEditorEngine(ptrFunc function,
                        image nImg){
    image temp;
    temp = (*function)(nImg);
    return temp;
}
```

Receive:
functionPtr & image

It executes the function with the image
Function returns the result

Returns:
image

Function pointers

- **Good**
  - New function additions do **not alter the engine**
  - The engine only needs to be tested once
  - Can change the function implementations **dynamically**
- **Bad**
  - **More complex** code (function pointers are not easy to understand for beginners)
  - Probable *bugs*
  - Lack of **compile time guarantees** (function signature)

Function pointers

```
int plus(int a, int b) { return a+b; }
int minus(int a, int b) { return a-b; }




int main() {
  int (*func)(int, int);
  func = plus;
  printf("%d\n", func(2,5));
  return 0;
}
```

Function pointers

```
int plus(int a, int b) { return a+b; }
int minus(int a, int b) { return a-b; }
int times(int a, int b) { return a*b; }
int divide(int a, int b) { return a/b; }

int main() {
  int (*func)(int, int);
  func = plus;
  printf("%d\n", func(2,5));
  return 0;
}
```

Function pointers

```c
int plus(int a, int b) { return a+b; }
int minus(int a, int b) { return a-b; }
int times(int a, int b) { return a*b; }
int divide(int a, int b) { return a/b; }

int main() {
  int (*func)(int, int);
  func = plus;
  printf("%d\n", func(2,5));
  return 0;
}
```

Only new features/functions must to be tested

Exercise

Using function pointers

Exercise

- Update last class structure to include a function pointer as one of its members.
- Create a function (ExecProc) that executes the pointer stored in the "first" filled position of the circular buffer.
- Create a main that executes the commands to the side:
- Create the three different functions, each printing a different phrase.

```
#include "stdio.h"
void main (void){
    addProc(p1);
    addProc(p2);
    addProc(p3);
    ExeProc();
    RemoveProc();
    ExeProc();
    RemoveProc();
    ExeProc();
    RemoveProc();
}
```

```
#include "stdio.h"
void main (void){
   addProc("proc1", 0);
   addProc("proc2", 1);
   addProc("proc3", 2);
   removeProc();
   removeProc();
   removeProc();
}
```

Exercise

Exercise

- Update last class structure to include a function pointer as one of its members.
- Create a function (ExecProc) that executes the pointer stored in the "first" filled position of the circular buffer.
- Create a main that executes the commands to the side:
- Create the three different functions, each printing a different phrase.

```
#include "stdio.h"
void main (void){
    addProc(p1);
    addProc(p2);
    addProc(p3);
    ExeProc();
    RemoveProc();
    ExeProc();
    RemoveProc();
    ExeProc();
    RemoveProc();
}
```

```c
typedef int (*ptrFunc)(void);

typedef struct {
    char name;
    int time;
    ptrFunc func;
}process;

#define BUFFERSIZE  10
process buffer[BUFFERSIZE];

int start=0, end=0;
```

Exercise

```c
void addProc(process *nProcess, int nTime, ptrFunc
fPointer){
  if ( ((end+1)%BUFFERSIZE) != start){
    buffer[end].name = nProcess;
    buffer[end].time = nTime;
    buffer[end].pFunc = fPointer;
    end = (end+1)%(BUFFERSIZE);
  }
}
```

Exercise

```
void removeProc (void){
  if ( start != end){
    start = (start +1)%(BUFFERSIZE);
  }
}

void exec(void){
  if (start != end){
    buffer[start].func();
  }
}
```

Exercise

```
void func1(void){printf("f1 \n");}

void func2(void){printf("f2 \n");}

void func3(void){printf("f3 \n");}
```

Exercise

```c
#include "stdio.h"
void main (void){
   addProc("Proc1", 1, func1);
   addProc("Proc2", 2, func2);
   addProc("Proc3", 3, func3);
   exec();
   removeProc();
   exec();
   removeProc();
   exec();
   removeProc();
}
```

Exercise

```c
#include "stdio.h"
void main (void){
   addProc("Proc1", 1, func1);
   addProc("Proc2", 2, func2);
   addProc("Proc3", 3, func3);
   exec();
   removeProc();
   exec();
   removeProc();
   exec();
   removeProc();
}
```

```c
/* Similar to:
  f1();
  f2();
  f3();
 */
```

Exercise

Exercise

Step-by-step

```c
typedef int (*ptrFunc)(void);

typedef struct {
    char name;
    int time;
    ptrFunc func;
}process;

#define BUFFERSIZE  10
process buffer[BUFFERSIZE];

int start=0, end=0;
```

Exercise

buffer

	name	**time**	**func**
	char	int	ptrFunc
0	?	?	?
1	?	?	?
2	?	?	?
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	0
end	0

```
#include "stdio.h"
void main (void){
   addProc("Proc1", 1, func1);
   addProc("Proc2", 2, func2);
   addProc("Proc3", 3, func3);
   exec();
   removeProc();
   exec();
   removeProc();
   exec();
   removeProc();
}
```

Exercise

buffer

	name char	time int	func ptrFunc
0	?	?	?
1	?	?	?
2	?	?	?
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	0
end	0

```
void addProc(process *nProcess, int nTime,
ptrFunc fPointer){
  if ( ((end+1)%BUFFERSIZE) != start){
    buffer[end].name = nProcess;
    buffer[end].time = nTime;
    buffer[end].func = fPointer;
    end = (end+1)%(BUFFERSIZE);
  }
}
```

Exercise

buffer

	name char	**time** int	**func** ptrFunc
0	?	?	?
1	?	?	?
2	?	?	?
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	0
end	0

```
void addProc(process *nProcess, int nTime,
ptrFunc ponteiro){
  if ( ((end+1)%BUFFERSIZE) != start){
    buffer[end].name = nProcess;
    buffer[end].time = nTime;
    buffer[end].func = ponteiro;
    end = (end+1)%(BUFFERSIZE);
  }
}
```

Exercise

buffer

	name char	**time** int	**func** ptrFunc
0	**Proc1**	**1**	**func1**
1	?	?	?
2	?	?	?
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	0
end	1

```c
void addProc(process *nProcess, int nTime,
ptrFunc ponteiro){
  if ( ((end+1)%BUFFERSIZE) != start){
    buffer[end].name = nProcess;
    buffer[end].time = nTime;
    buffer[end].func = ponteiro;
    end = (end+1)%(BUFFERSIZE);
  }
}
```

Exercise

buffer

	name char	time int	func ptrFunc
0	**Proc1**	**1**	**func1**
1	**Proc2**	**2**	**func2**
2	?	?	?
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	0
end	2

```
void addProc(process *nProcess, int nTime,
ptrFunc ponteiro){
  if ( ((end+1)%BUFFERSIZE) != start){
    buffer[end].name = nProcess;
    buffer[end].time = nTime;
    buffer[end].func = ponteiro;
    end = (end+1)%(BUFFERSIZE);
  }
}
```

Exercise

buffer

	name char	**time** int	**func** ptrFunc
0	**Proc1**	**1**	**func1**
1	**Proc2**	**2**	**func2**
2	**Proc3**	**3**	**func3**
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	0
end	3

```c
#include "stdio.h"
void main (void){
   addProc("Proc1", 1, func1);
   addProc("Proc2", 2, func2);
   addProc("Proc3", 3, func3);
   exec();
   removeProc();
   exec();
   removeProc();
   exec();
   removeProc();
}
```

Exercise

buffer

	name char	time int	func ptrFunc
0	**Proc1**	**1**	**func1**
1	**Proc2**	**2**	**func2**
2	**Proc3**	**3**	**func3**
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	0
end	3

```
#include "stdio.h"
void main (void){
   addProc("Proc1", 1, func1);
   addProc("Proc2", 2, func2);
   addProc("Proc3", 3, func3);
   exec();
   removeProc();
   exec();
   removeProc();
   exec();
   removeProc();
}
```

Exercise

buffer

	name char	time int	func ptrFunc
0	**Proc1**	**1**	**func1**
1	**Proc2**	**2**	**func2**
2	**Proc3**	**3**	**func3**
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	0
end	3

```c
void exec(void){
  if (start != end){
    buffer[start].func();
  }
}
```

1

```c
void removeProc (void){
  if ( start != end){
     start = (start +1)%(BUFFERSIZE);
  }
}
```

```c
void func1(void){printf("f1 \n");}
```

2

```c
void func2(void){printf("f2 \n");}

void func3(void){printf("f3 \n");}
```

Exercise

buffer

	name char	**time** int	**func** ptrFunc
0	**Proc1**	**1**	**func1**
1	**Proc2**	**2**	**func2**
2	**Proc3**	**3**	**func3**
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	0
end	3

```c
void exec(void){
  if (start != end){
    buffer[start].func();
  }
}

void removeProc (void){
  if ( start != end){
      start = (start +1)%(BUFFERSIZE);
  }
}

void func1(void){printf("f1 \n");}

void func2(void){printf("f2 \n");}

void func3(void){printf("f3 \n");}
```

3

buffer

	name char	time int	func ptrFunc
0	**Proc1**	**1**	**func1**
1	**Proc2**	**2**	**func2**
2	**Proc3**	**3**	**func3**
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	1
end	3

Exercise

```c
#include "stdio.h"
void main (void){
   addProc("Proc1", 1, func1);
   addProc("Proc2", 2, func2);
   addProc("Proc3", 3, func3);
   exec();
   removeProc();
   exec();
   removeProc();
   exec();
   removeProc();
}
```

buffer

	name char	**time** int	**func** ptrFunc
0	**Proc1**	**1**	**func1**
1	**Proc2**	**2**	**func2**
2	**Proc3**	**3**	**func3**
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	1
end	3

Exercise

1

```c
void exec(void){
  if (start != end){
    buffer[start].func();
  }
}

void removeProc (void){
  if ( start != end){
     start = (start +1)%(BUFFERSIZE);
  }
}


void func1(void){printf("f1 \n");}

void func2(void){printf("f2 \n");}
```

2

```c
void func3(void){printf("f3 \n");}
```

Exercise

buffer

	name char	**time** int	**func** ptrFunc
0	**Proc1**	**1**	**func1**
1	**Proc2**	**2**	**func2**
2	**Proc3**	**3**	**func3**
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	1
end	3

```
void exec(void){
  if (start != end){
     buffer[start].func();
   }
}

void removeProc (void){
   if ( start != end){
       start = (start +1)%(BUFFERSIZE);
   }
}


void func1(void){printf("f1 \n");}

void func2(void){printf("f2 \n");}

void func3(void){printf("f3 \n");}
```

3

Exercise

buffer

	name char	time int	func ptrFunc
0	**Proc1**	**1**	**func1**
1	**Proc2**	**2**	**func2**
2	**Proc3**	**3**	**func3**
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	2
end	3

Exercise

```c
#include "stdio.h"
void main (void){
   addProc("Proc1", 1, func1);
   addProc("Proc2", 2, func2);
   addProc("Proc3", 3, func3);
   exec();
   removeProc();
   exec();
   removeProc();
   exec();
   removeProc();
}
```

buffer

	name char	time int	func ptrFunc
0	**Proc1**	**1**	**func1**
1	**Proc2**	**2**	**func2**
2	**Proc3**	**3**	**func3**
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	2
end	3

```
void exec(void){
  if (start != end){
     buffer[start].func();
   }
}
```

1

```
void removeProc (void){
   if ( start != end){
        start = (start +1)%(BUFFERSIZE);
   }
}


void func1(void){printf("f1 \n");}

void func2(void){printf("f2 \n");}

void func3(void){printf("f3 \n");}
```

2

Exercise

buffer

	name char	**time** int	**func** ptrFunc
0	**Proc1**	**1**	**func1**
1	**Proc2**	**2**	**func2**
2	**Proc3**	**3**	**func3**
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	2
end	3

```
void exec(void){
  if (start != end){
     buffer[start].func();
   }
}

void removeProc (void){
   if ( start != end){
       start = (start +1)%(BUFFERSIZE);
   }
}


void func1(void){printf("f1 \n");}

void func2(void){printf("f2 \n");}

void func3(void){printf("f3 \n");}
```

Exercise

buffer

	name char	**time** int	**func** ptrFunc
0	**Proc1**	**1**	**func1**
1	**Proc2**	**2**	**func2**
2	**Proc3**	**3**	**func3**
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	3
end	3

Exercise

```c
#include "stdio.h"
void main (void){
   addProc("Proc1", 1, func1);
   addProc("Proc2", 2, func2);
   addProc("Proc3", 3, func3);
   exec();
   removeProc();
   exec();
   removeProc();
   exec();
   removeProc();
}
```

buffer

	name char	time int	func ptrFunc
0	Proc1	1	func1
1	Proc2	2	func2
2	Proc3	3	func3
3	?	?	?
4	?	?	?
5	?	?	?
6	?	?	?
7	?	?	?
8	?	?	?
9	?	?	?

start	3
end	3

Exercise

Options / Changeovers

Exercise – Option 01

```
void exec(void){
  if (start != end){
    printf("Process - Current ID %d\n", start);
    printf("Process - Last ID %d [null]\n", end);
    printf("<----------------------\n", end);
    buffer[start].func();
    printf("---------------------->\n", end);
  }
}
```

Exercise – Option 02

```c
#include "stdio.h"
void main (void){
   addProc("Proc1", 1, func1);
   addProc("Proc2", 2, func2);
   addProc("Proc3", 3, func3);
   exec();
   exec();
   removeProc();
   exec();
   removeProc();
   exec();
   removeProc();
}
```

Exercise – Option 03

```c
#include "stdio.h"
void main (void){
   addProc("Proc1", 1, func1);
   addProc("Proc2", 2, func2);
   addProc("Proc3", 3, func3);
   exec();
   removeProc();
   addProc("Proc4", 4, func1);
   addProc("Proc5", 5, func2);
   exec();
   removeProc();
   exec();
   addProc("Proc5", 5, func2);
   removeProc();
}
```

Exercise – Option 04

```
#include "stdio.h"
void main (void){
  addProc("Proc1", 1, func1);
  addProc("Proc2", 2, func2);
  addProc("Proc3", 3, func3);
  exec();
  removeProc();
  if (<condition>){
    removeProc();
    addProc("Proc4", 4, func1);
  else{
    addProc("Proc5", 5, func2);
  }
  exec();
  removeProc();
  exec();
  removeProc();
}
```

Linus Torvalds

core 0.01 (1991)

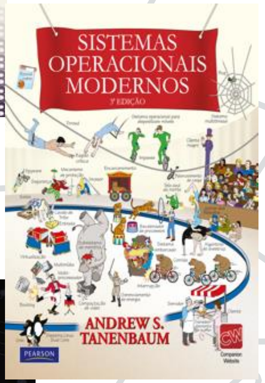
Bibliography

- Denardin, G. B.; Barriquello, C. H. **Sistemas operacionais de tempo real e sua aplicação em sistemas embarcados**. 1ª ed. Editora Blucher. ISBN: 9788521213970. https://plataforma.bvirtual.com.br/Acervo/Publicacao/169968

- Tanenbaum, A.S. **Sistemas Operacionais Modernos**. 3ª ed. 674 páginas. São Paulo: Pearson. ISBN: 9788576052371.
  https://plataforma.bvirtual.com.br/Acervo/Publicacao/1233

- Almeida, Moraes, Seraphim e Gomes. **Programação de Sistemas Embarcados**. 2ª ed. Editora GEN LTC. ISBN: 9788595159105.
  https://cengagebrasil.vitalsource.com/books/9788595159112

SISTEMAS OPERACIONAIS DE TEMPO REAL E SUA APLICAÇÃO EM SISTEMAS EMBARCADOS
GUSTAVO WEBER DENARDIN
CARLOS HENRIQUE BARRIQUELLO
Blucher



RODRIGO MAXIMIANO A. ALMEIDA
CARLOS HENRIQUE V. MORAES
THATYANA F. PIOLA SERAPHIM
OTÁVIO DE SOUZA M. GOMES
PROGRAMAÇÃO DE
SISTEMAS EMBARCADOS
DESENVOLVENDO SOFTWARE PARA MICROCONTROLADORES EM LINGUAGEM C
2ª EDIÇÃO
LTC

Available at: https://unifei.edu.br/ensino/bibliotecas/

Embedded Operating Systems

Circular buffer

protoCore 0.00001

Prof. Otávio Gomes

otavio.gomes@unifei.edu.br

in /otavio-gomes