



Systems Programming

Lecture 5:

Programming in C using Linux System Calls. Input and Output

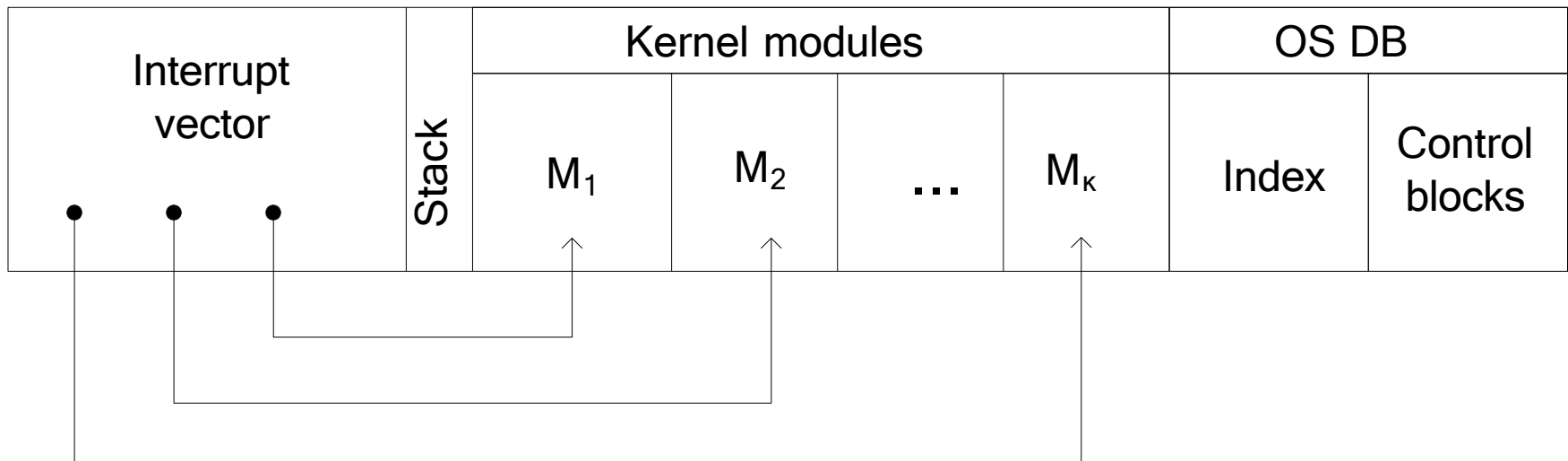
Dmitry A. Zaitsev

<https://dimazaitsev.github.io>

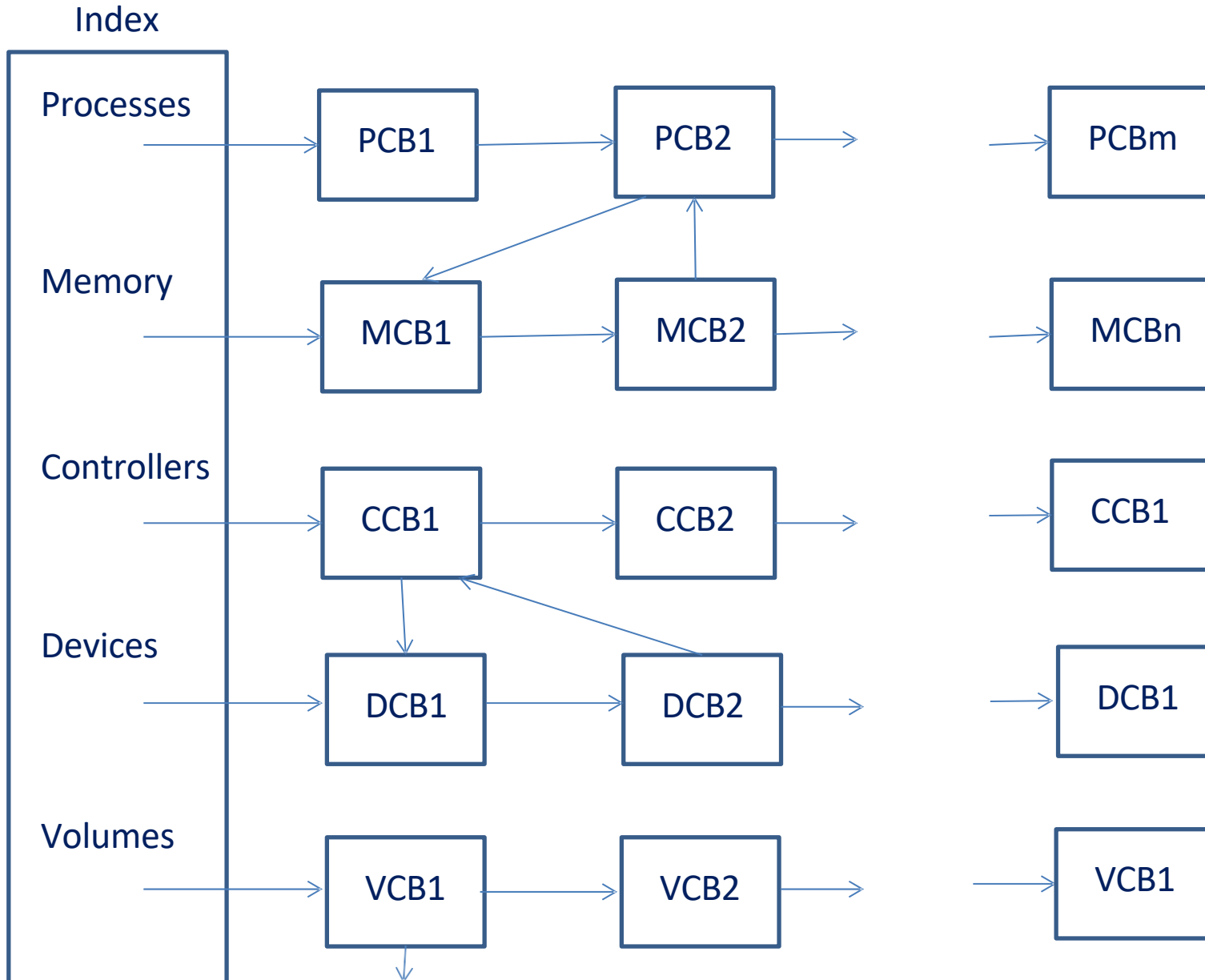
Kernel

A part of OS which is not a process.

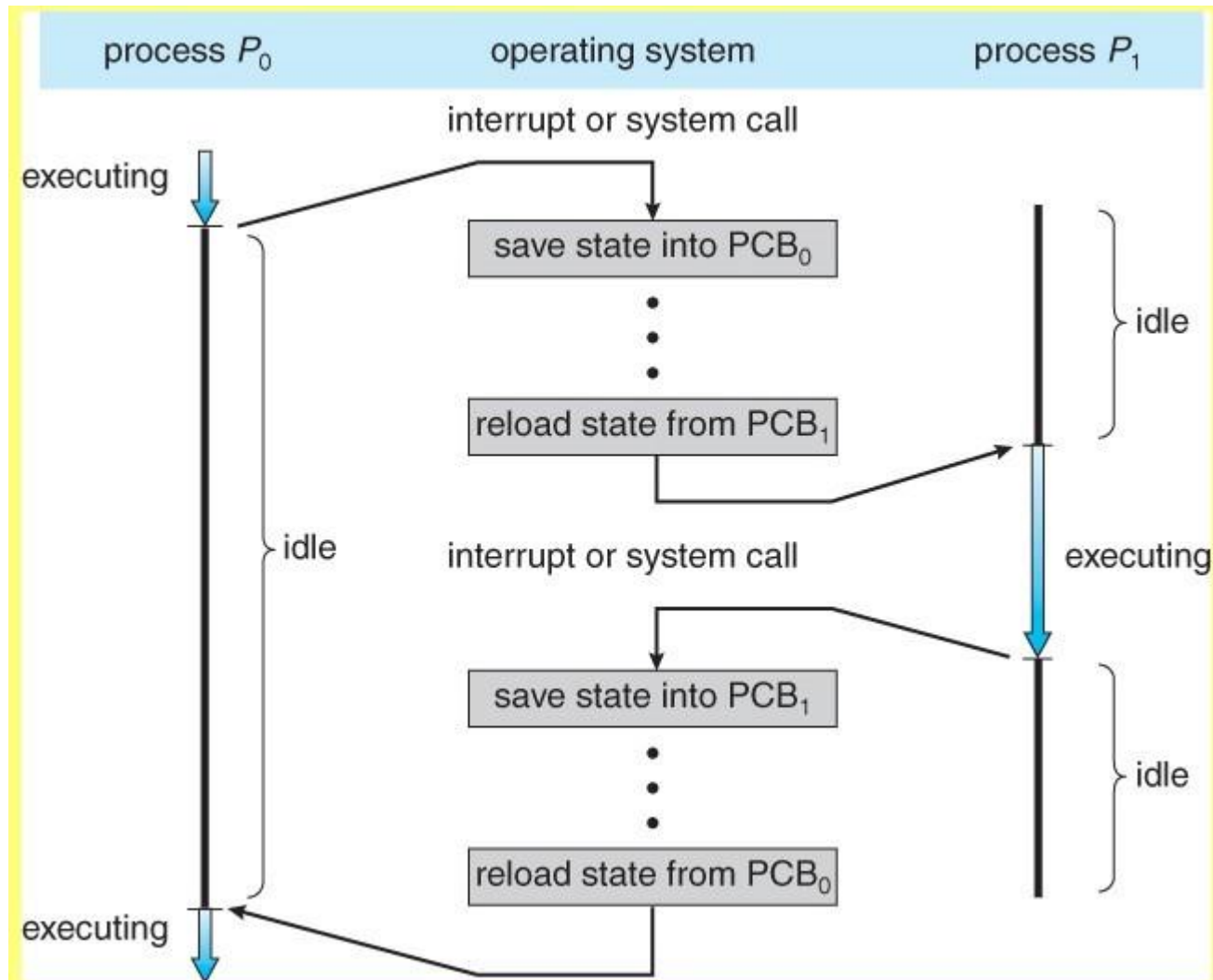
A set of interrupt handlers, stack, data base, and pool.



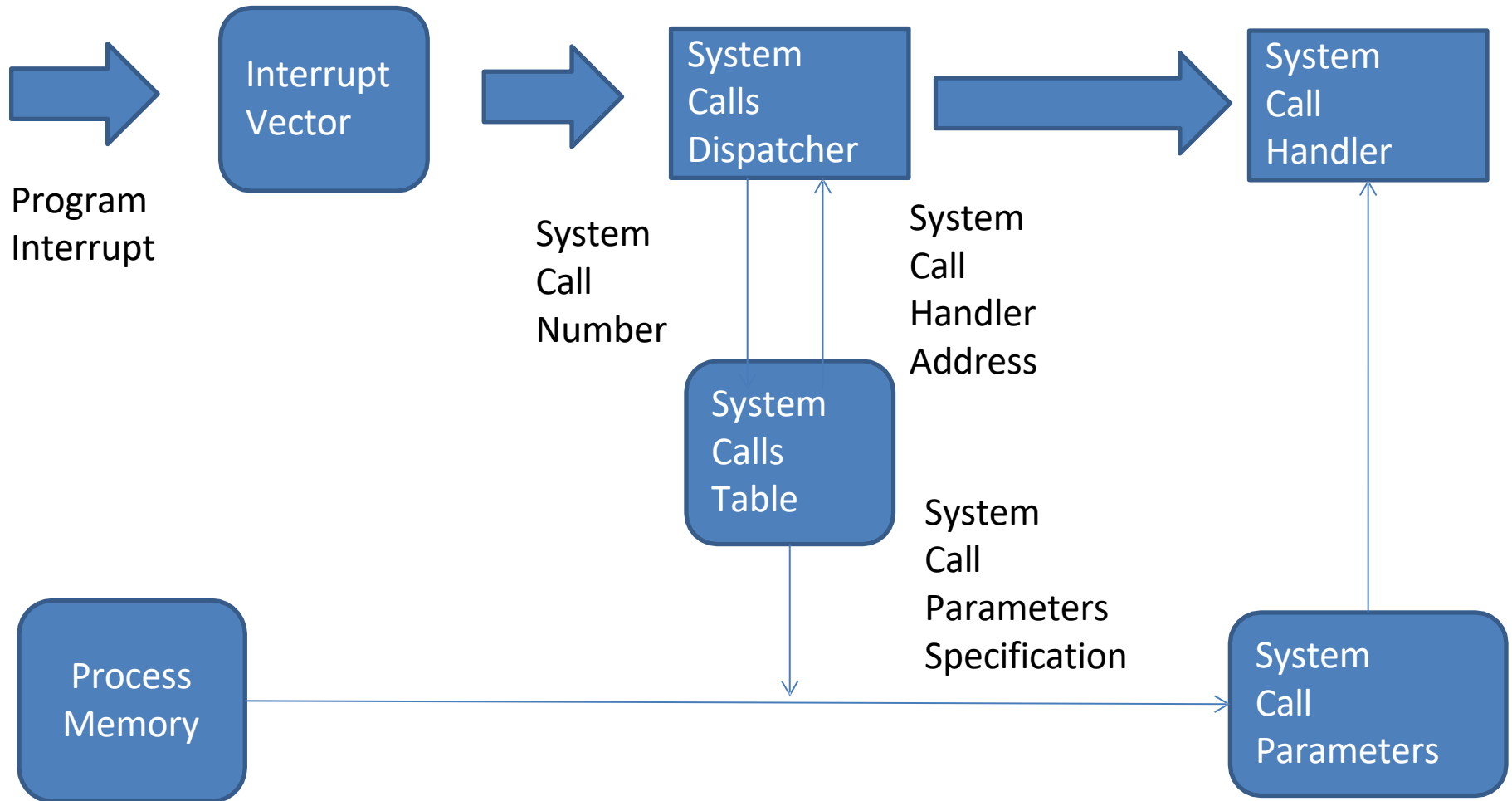
Data base of OS



Switching processes



Linux System Call Scheme



List of Linux System Calls

%eax	Name	Source	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	kernel/exit.c	int	-	-	-	-
2	sys_fork	arch/i386/kernel/process.c	struct pt_regs	-	-	-	-
3	sys_read	fs/read_write.c	unsigned int	char *	size_t	-	-
4	sys_write	fs/read_write.c	unsigned int	const char *	size_t	-	-
5	sys_open	fs/open.c	const char *	int	int	-	-
6	sys_close	fs/open.c	unsigned int	-	-	-	-
7	sys_waitpid	kernel/exit.c	pid_t	unsigned int *	int	-	-
8	sys_creat	fs/open.c	const char *	int	-	-	-
9	sys_link	fs/namei.c	const char *	const char *	-	-	-
10	sys_unlink	fs/namei.c	const char *	-	-	-	-
11	sys_execve	arch/i386/kernel/process.c	struct pt_regs	-	-	-	-
12	sys_chdir	fs/open.c	const char *	-	-	-	-
13	sys_time	kernel/time.c	int *	-	-	-	-
14	sys_mknod	fs/namei.c	const char *	int	dev_t	-	-
15	sys_chmod	fs/open.c	const char *	mode_t	-	-	-
16	sys_lchown	fs/open.c	const char *	uid_t	gid_t	-	-
18	sys_stat	fs/stat.c	char *	struct __old_kernel_stat *	-	-	-
19	sys_lseek	fs/read_write.c	unsigned int	off_t	unsigned int	-	-

System Calls in C Program

- Direct System Call:

```
rc = syscall(SYS_chmod, "/etc/passwd", 0444);
```

```
if (rc == -1)
```

```
    fprintf(stderr, "chmod failed, errno = %d\n", errno);
```

- Using a Dedicated libc function:

```
rc = chmod("/etc/passwd", 0444);
```

```
if (rc == -1)
```

```
    fprintf(stderr, "chmod failed, errno = %d\n", errno);
```

An overview of libc functions

- Creation and termination of processes
- Inter Process Communication
- Input/Output: Streams & Low-Level
- Dynamic memory & shared memory
- Sockets
- Mathematics
- Searching, Sorting, and Pattern Matching
- Date & Time
- etc

Building and running a program

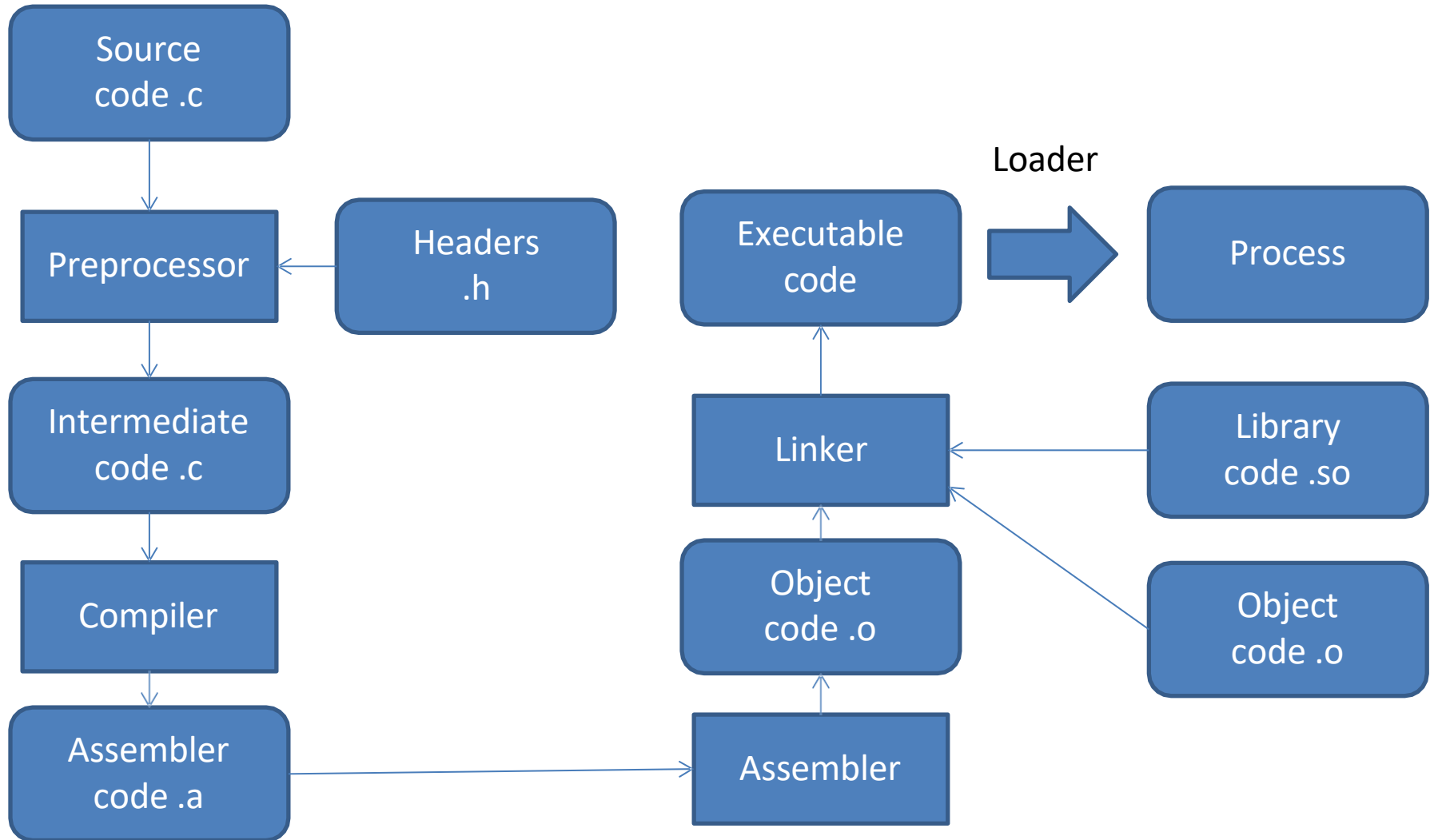
```
>gedit name.c
```

```
>gcc -o name name.c
```

```
>./name
```

Keys: -lm maths; -lrt real time; -lpthread threads

Stages of gcc work



Saving intermediate results

- Preprocessor

`gcc -E`

- Compile (to Assembler)

`gcc -S`

- Compile (to Object)

`gcc -c`

An example of C program

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#define min2(x,y) (((x)<(y))? (x): (y))
```

```
int main(int argc, char * argv[])
```

```
{
```

```
    int a=atoi(argv[1]), b=atoi(argv[2]);
```

```
    printf("minimum of %d and %d is %d\n", a, b, min2(a,b));
```

```
}
```

Basic headers

- `stdio.h` – stream input/output
- `stdlib.h` – frequently used functions
- `unistd.h` – Unix standard
- `fcntl.h` – control flags
- `semaphore.h` – semaphores
- `mqueue.h` – message queue

I/O on Streams

- FILE
- FILE * stdin; FILE * stdout; FILE * stderr;
- Opening Streams

FILE * fopen (const char *filename, const char *opentype)

- Closing Streams

int fclose (FILE *stream)

Simple I/O by Characters or Lines

- `int fputc (int c, FILE *stream)`
- `int putchar (int c)`
- `int fputs (const char *s, FILE *stream)`

- `int fgetc (FILE *stream)`
- `int getchar (void)`
- `char * fgets (char *s, int count, FILE *stream)`

Block Input/Output

- `size_t fread (void *data, size_t size, size_t count, FILE *stream)`
- `size_t fwrite (const void *data, size_t size, size_t count, FILE *stream)`

Formatted Output

- `int printf (const char *template, . . .)`
- `int fprintf (FILE *stream, const char *template, . . .)`
- `int sprintf (char *s, const char *template, . . .)`
- `int scanf (const char *template, . . .)`
- `int fscanf (FILE *stream, const char *template, . . .)`
- `int sscanf (const char *s, const char *template, . . .)`

I/O formats

- Output

% [flags width [.precision] type] conversion

%c, %s, %d, %f

- Input

% [flags width type] conversion

- Flags: +, -, 0, #

- Type: h, l

Low-Level I/O

- `#include <unistd.h>`
`#include <sys/types.h>`
`#include <sys/stat.h>`
`#include <fcntl.h>`
- Given a pathname for a file, `open()` returns a file descriptor, a small, nonnegative integer.
- A call to `open()` creates a new open file description, an entry in the system-wide table of open files.
- The open file description records the file offset and the file status flags (see below).

Open and possibly create a file

- `int open(const char *pathname, int flags);`
- `int open(const char *pathname, int flags, mode_t mode);`
- Flags: access mode `O_RDONLY`, `O_WRONLY`, `O_RDWR`; file creation `O_CLOEXEC`, `O_CREAT`, `O_DIRECTORY`, `O_EXCL`, `O_NOCTTY`, `O_NOFOLLOW`, `O_TMPFILE`, `O_TRUNC`
- Mode: `S_IRWXU`, `S_IRUSR`, `S_IXUSR`,...

Read, Write, and Close

- Read from a file descriptor

```
ssize_t read(int fd, void *buf, size_t count);
```

- write to a file descriptor

```
ssize_t write(int fd, const void *buf, size_t count);
```

- close a file descriptor

```
int close(int fd);
```

Reposition read/write file offset

- `off_t lseek(int fd, off_t offset, int whence);`
- Whence: `SEEK_SET`, `SEEK_CUR`, `SEEK_END`
- `SEEK_DATA`, `SEEK_HOLE`