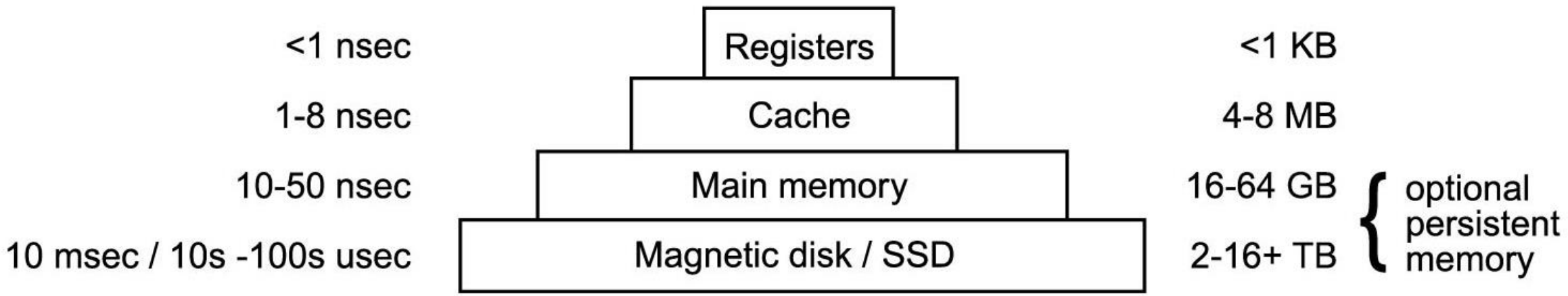
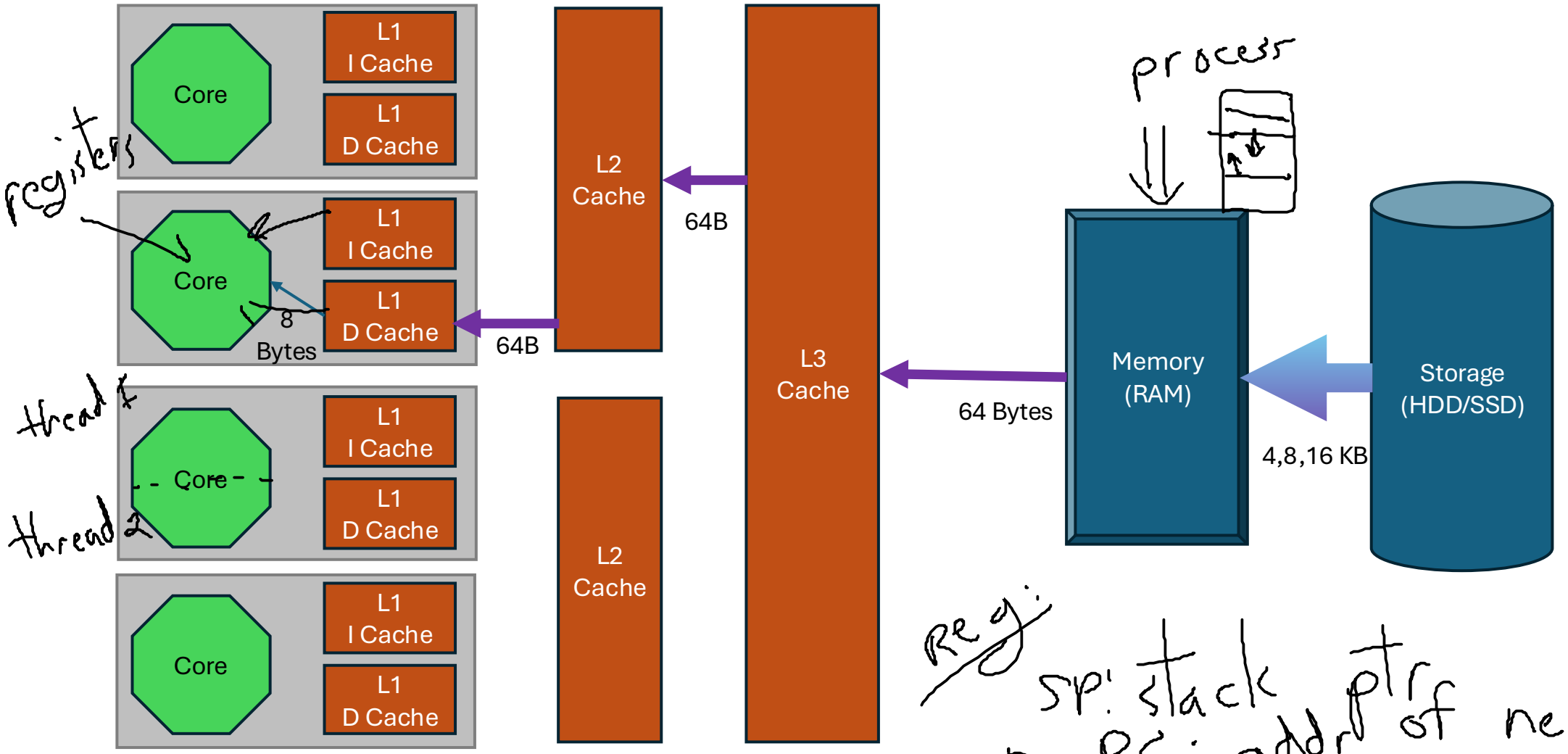
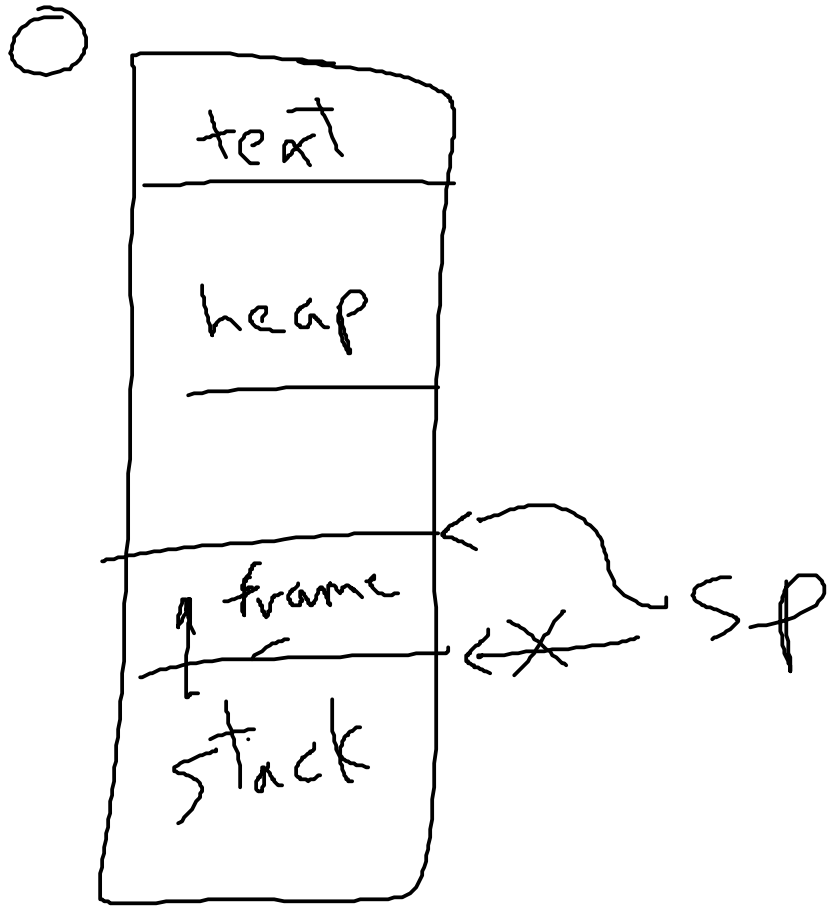
Memory Hierarchy

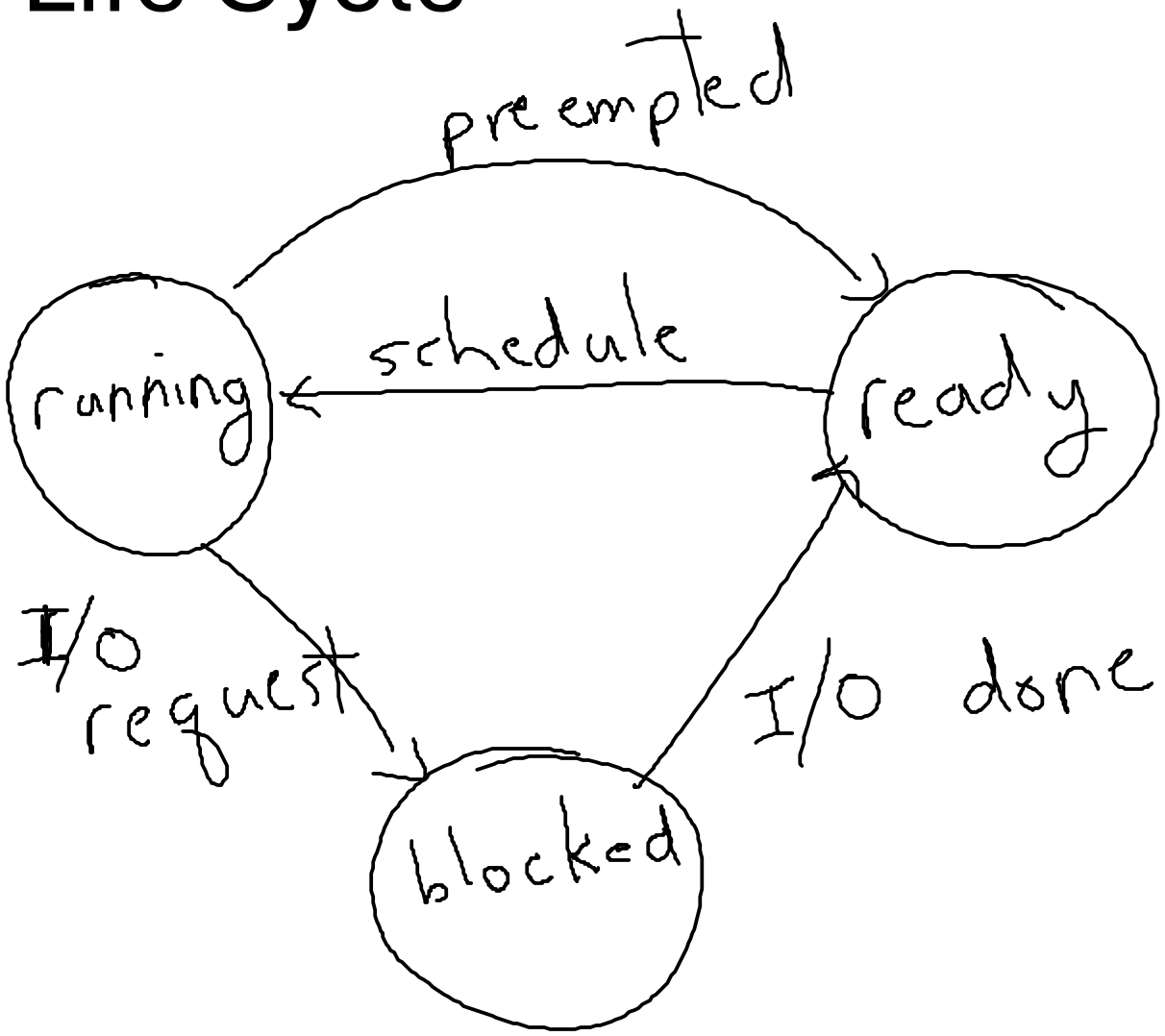


Multi-core CPU – Cache and Memory



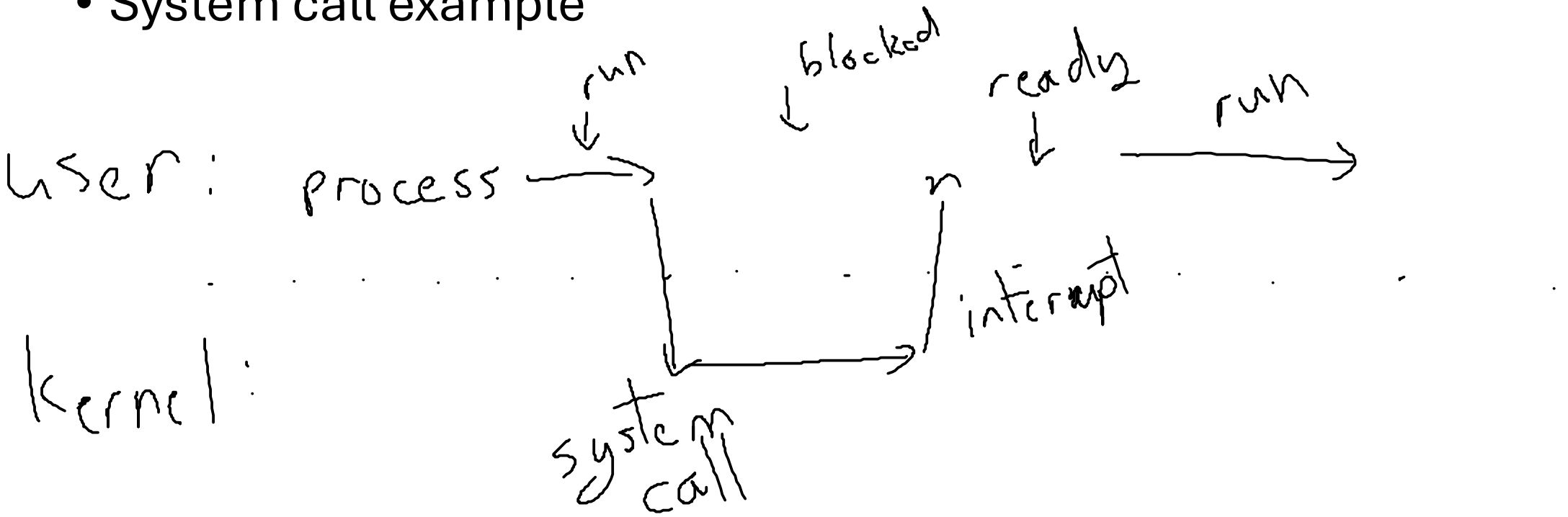

text
heap
frame
sp
stack

Process Life Cycle



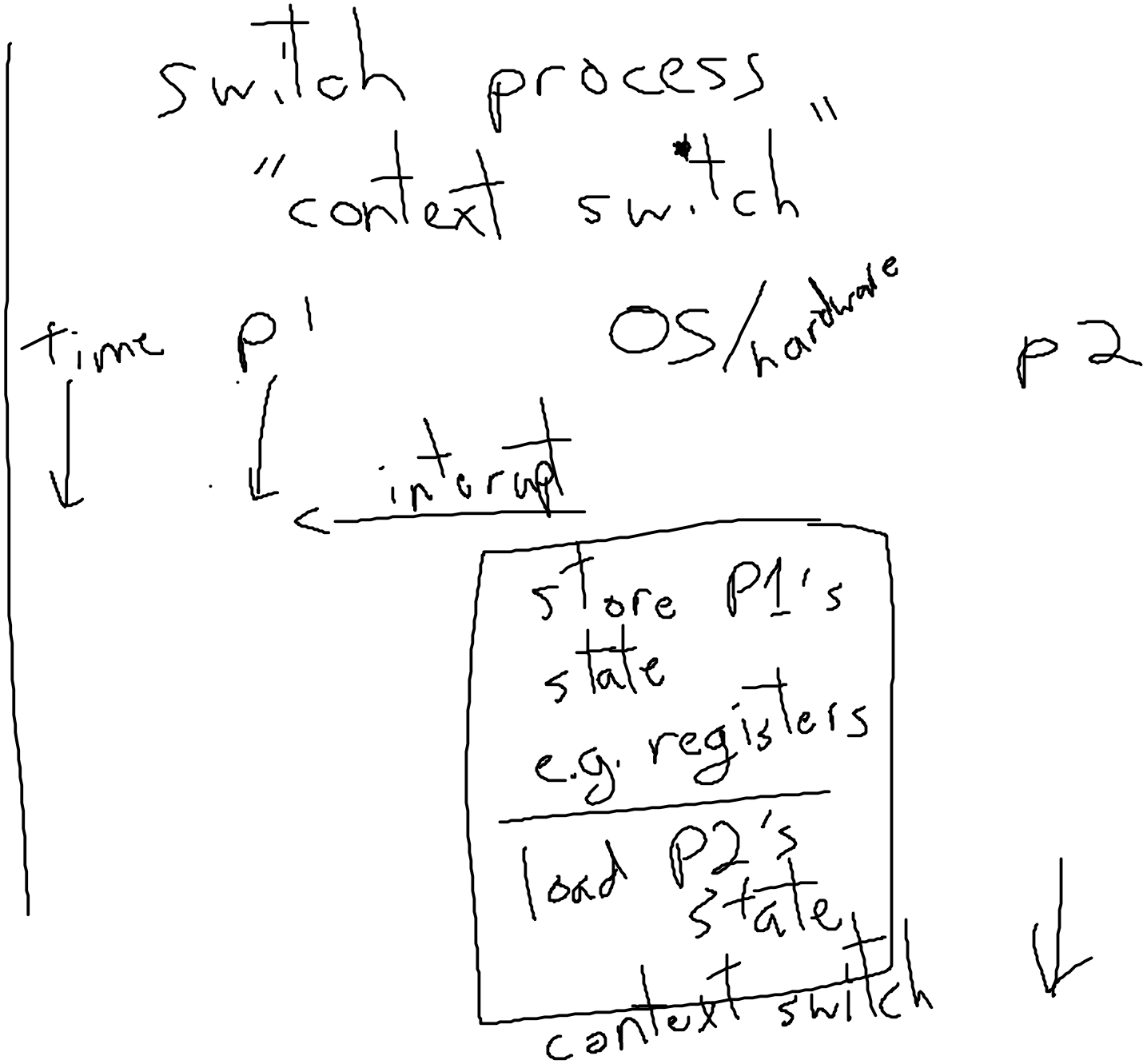
Modes and System Calls

- User mode
- Kernel mode — access to OS instr., resources
- System call example



Interrupts

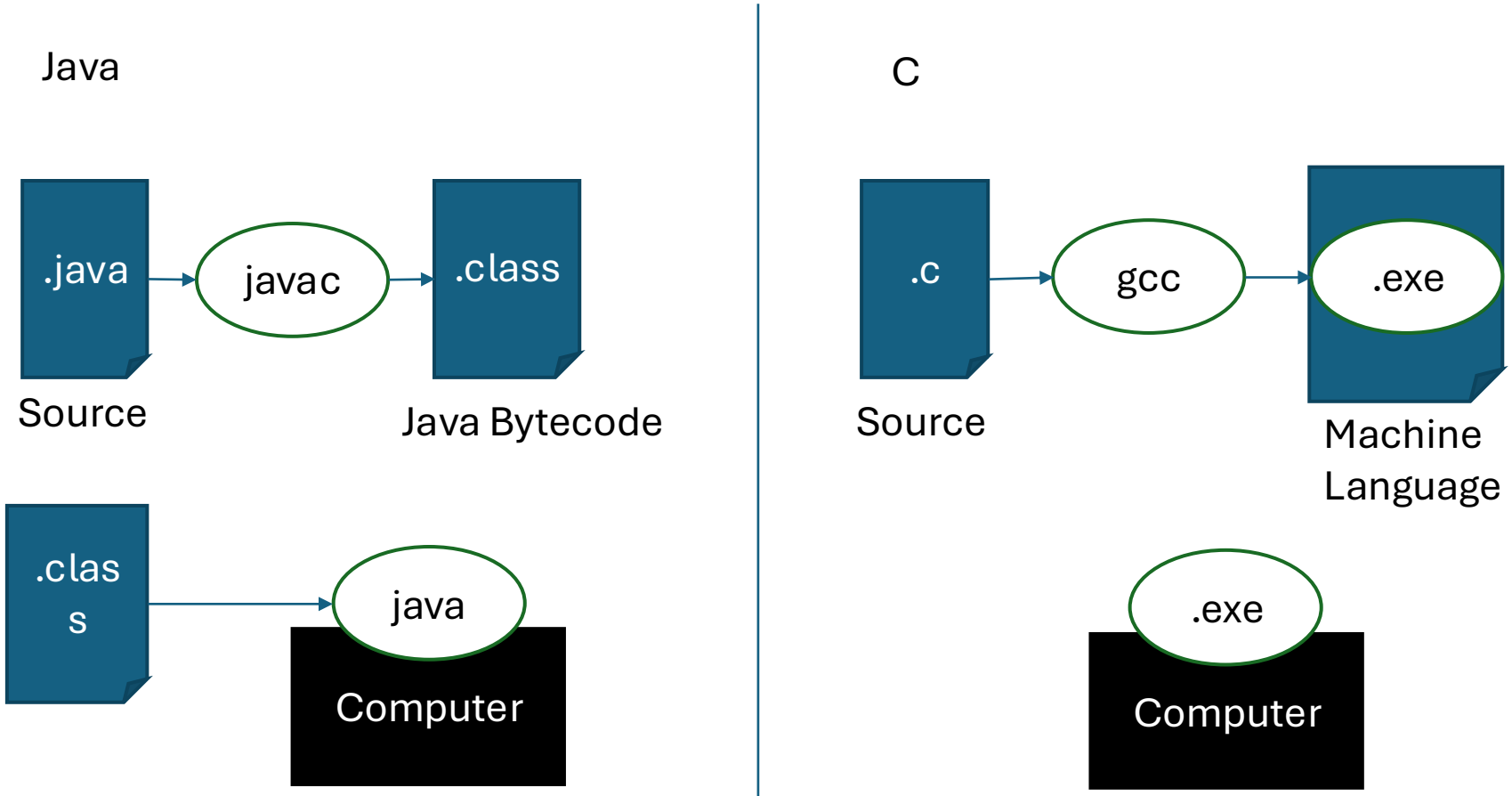
- I/O Device Interaction
- System Calls



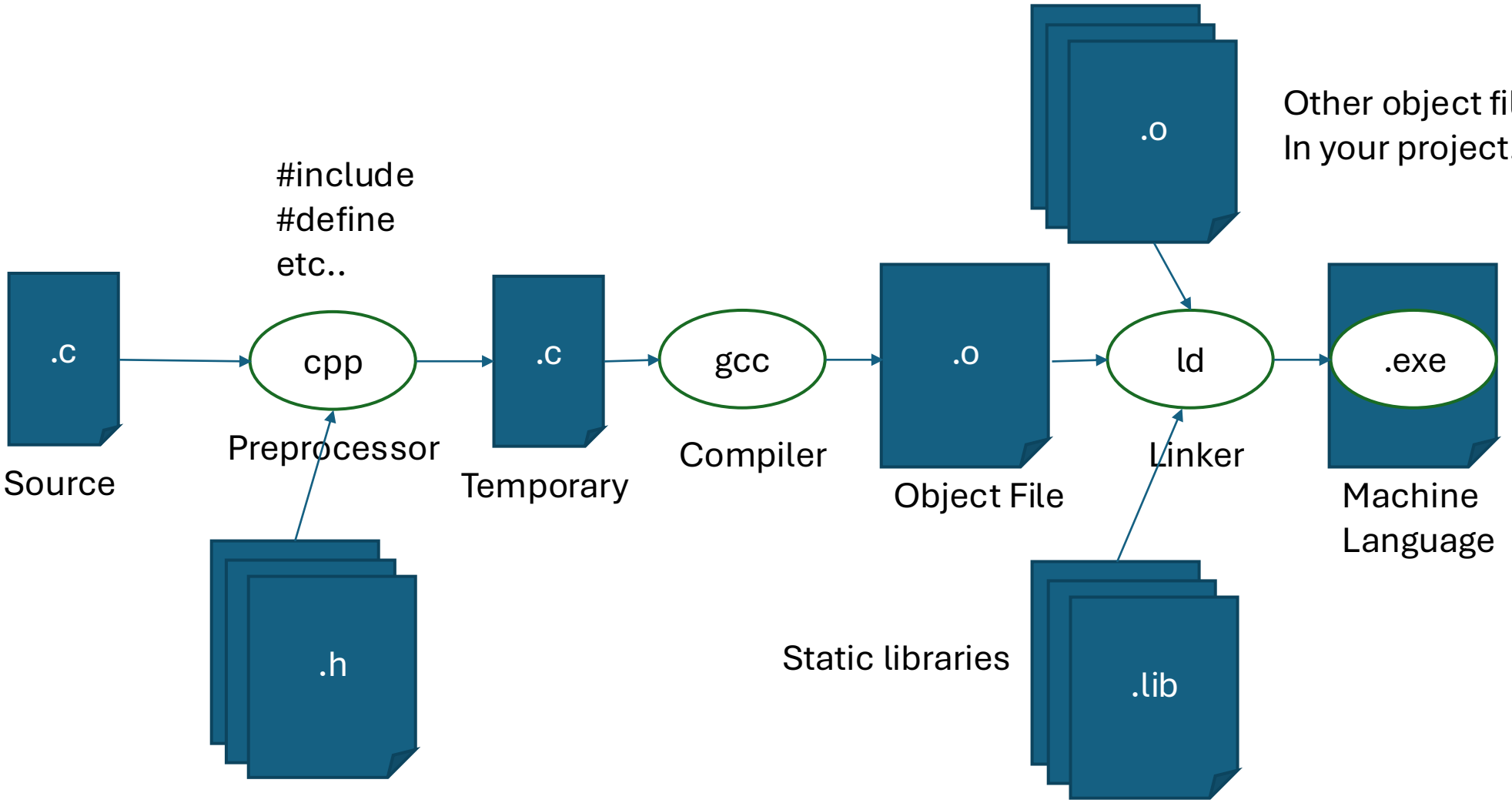
Units

Prefix	Base 10	Value	Base 2	Value
Kilo	10^3	1,000	2^{10}	1,024
Mega	10^6	1,000,000	2^{20}	1,048,576
Giga	10^9	1,000,000,000	2^{30}	1,073,741,824
Tera	10^{12}	1,000,000,000,000	2^{40}	1,099,511,627,776
Peta	10^{15}	1,000,000,000,000,000	2^{50}	1,125,899,906,842,624
Exa	10^{18}	1,000,000,000,000,000,000	2^{60}	1,152,921,504,606,846,976

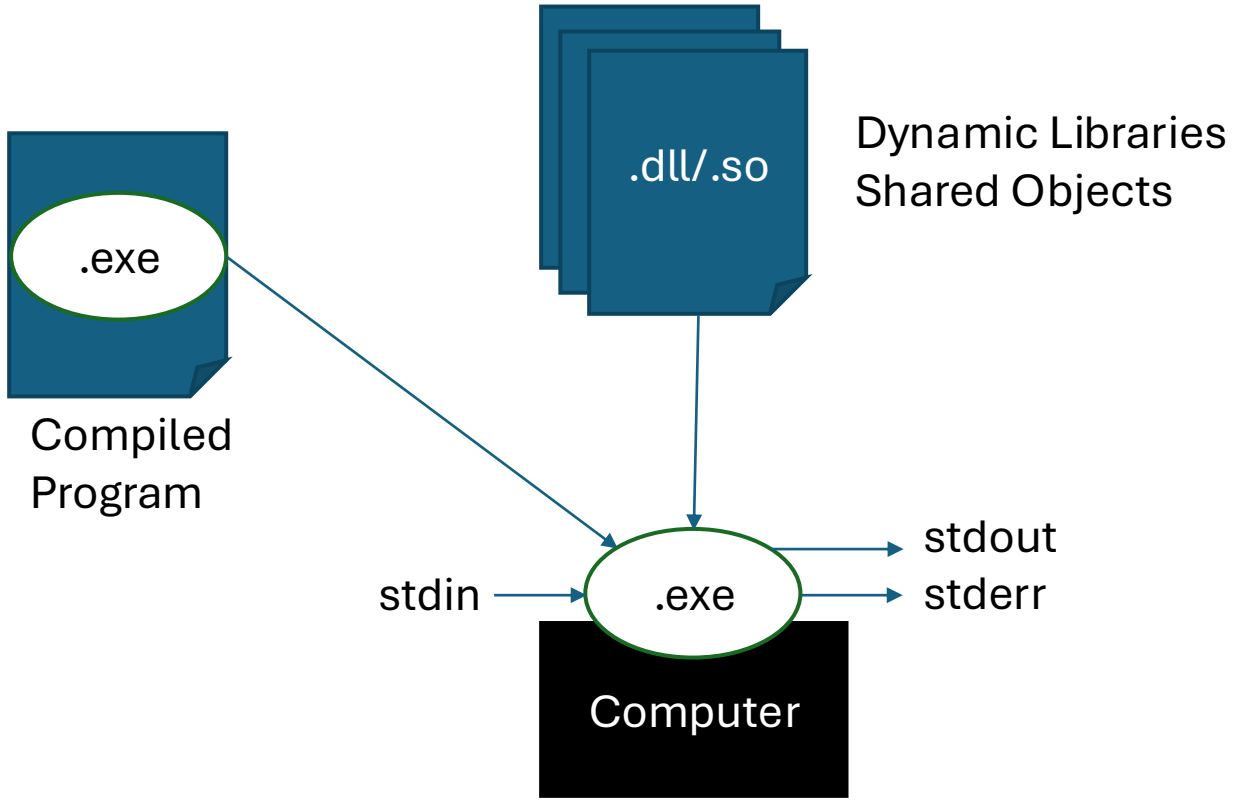
Compile and Run



C Build Process



C Loading/Execution



C main function

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

int main() {
  printf("Hello World\n");
  return EXIT_SUCCESS;
}
```

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

int main(int argc, char **argv) {
  printf("Hello World\n");
  return EXIT_SUCCESS;
}
```

Data Types

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

int main(){
        int i = 0;
        long l = 56;
        float f = 1.5;
        double d = 2.5;
        char c = 'g';
        short s = 45;
        char *str = "Hello There";
        printf("int    %d bytes.\n",sizeof(i));
        printf("long   %d bytes.\n",sizeof(l));
        printf("float  %d bytes.\n",sizeof(f));
        printf("double %d bytes.\n",sizeof(d));
        printf("char   %d bytes.\n",sizeof(c));
        printf("short  %d bytes.\n",sizeof(s));
        printf("char * %d bytes.\n",sizeof(str));
}
```

```
int    4 bytes.
long   8 bytes.
float  4 bytes.
double 8 bytes.
char   1 bytes.
short  2 bytes.
char * 8 bytes.
```

fork

Figure 5.2

```
6  int main(int argc, char *argv[]) {
7    printf("hello (pid:%d)\n", (int) getpid());
8    int rc = fork();
9    if (rc < 0) {           // fork failed; exit
10     fprintf(stderr, "fork failed\n");
11     exit(1);
12   } else if (rc == 0) { // child (new process)
13     printf("child (pid:%d)\n", (int) getpid());
14   } else {                // parent goes down this path
15     int rc_wait = wait(NULL);
16     printf("parent of %d (rc_wait:%d) (pid:%d)\n",
17            rc, rc_wait, (int) getpid());
18   }
19   return 0;
20 }
21
```

exec

Figure 5.3

```
7  int main(int argc, char *argv[]) {
8    printf("hello (pid:%d)\n", (int) getpid());
9    int rc = fork();
10   if (rc < 0) {           // fork failed; exit
11     fprintf(stderr, "fork failed\n");
12     exit(1);
13   } else if (rc == 0) { // child (new process)
14     printf("child (pid:%d)\n", (int) getpid());
15     char *myargs[3];
16     myargs[0] = strdup("wc");   // program: "wc"
17     myargs[1] = strdup("p3.c"); // arg: input file
18     myargs[2] = NULL;           // mark end of array
19     execvp(myargs[0], myargs);  // runs word count
20     printf("this shouldn't print out");
21   } else {                  // parent goes down this path
22     int rc_wait = wait(NULL);
23     printf("parent of %d (rc_wait:%d) (pid:%d)\n",
24            rc, rc_wait, (int) getpid());
25   }
26   return 0;
27 }
28
```