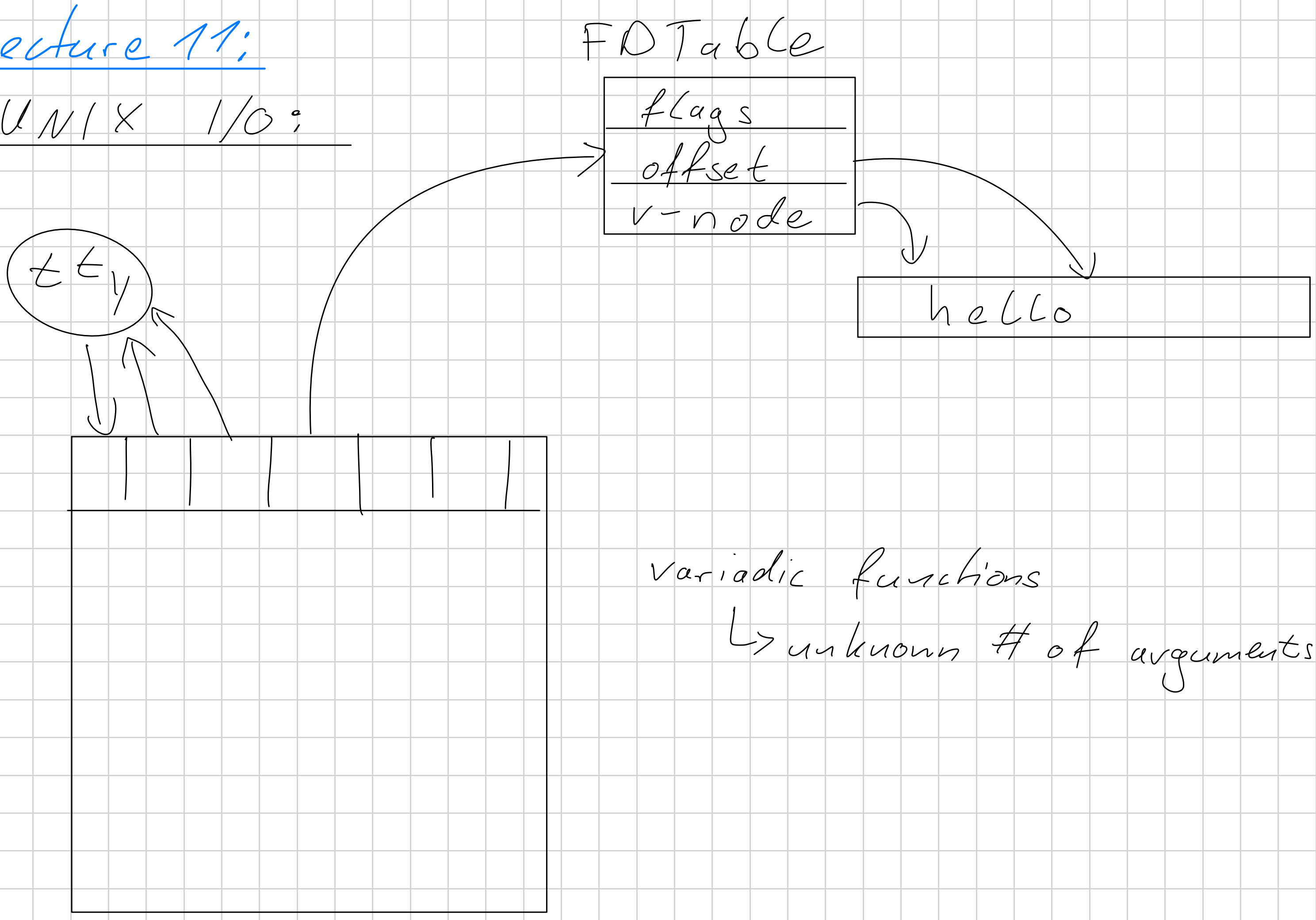
Lecture 11:

UNIX I/O:



variadic functions
↳ unknown # of arguments

```
int open(const char *path, int flags)

int open(const char *path, int flags, mode_t mode);
```

mode_t ↓ permissions to give if created

bitwise OR (see stat(2))

S_IRUSR
S_IWUSR
S_IXUSR
'' GRP
'' OTH

flags: how to open it:

one of

O_RDONLY
O_WRONLY
O_RDWR

possible ones:

O_CREAT
O_TRUNC
O_APPEND

returns: valid file descriptor on success, -1 on failure

```
int creat(const char *path, mode_t mode);

int close(int fd);

ssize_t read(int fd, void *buf, ssize_t size);
```
↳ read up to size bytes from fd;

↳ returns # bytes actually read or -1 on error.

write(2) is read backwards

```
int main (int argc, char *argv[])
{
    int in, out;
    if(argc != 3)
    {
        fprintf(stderr, "usage: %s in out \n" argv[0]);
        exit(EXIT_FAILURE);
    }
    if(-1 == (in=open(argv[ 1], O_RDONLY)))
    {
      perror(argv[1]);
      exit(EXIT_FAILURE);
    }
  if(-1 == (out =open(argv[2], O_WRONLY|O_TRUNC|
                               O_CREAT, S_IRUSR|
                               S_IWUSR)))
  {
    perror(argv[2]);
    exit(EXIT_FAILURE);
  }
  copy(in, out);
  close(in);
  close(out);
  return 0;
}
#define SIZE 1024
void copy (int in, int out)
    size_t num;
```

```
char buff[SIZE];

while((num=read(in, buff, SIZE)) > 0)
{ if (write(out, buff, num)) > ∅
  {
      perror("write");
      exit(EXIT_FAILURE);
  }
}
if(num==-1)
 { perror("read");
   exit(EXIT_FAILURE);
 }
}
```

h-table : [phe0, phe1, phe2, phe3, phe4]
$\uparrow$
*h-table-entry

struct h-table-entry {

byte
encoding