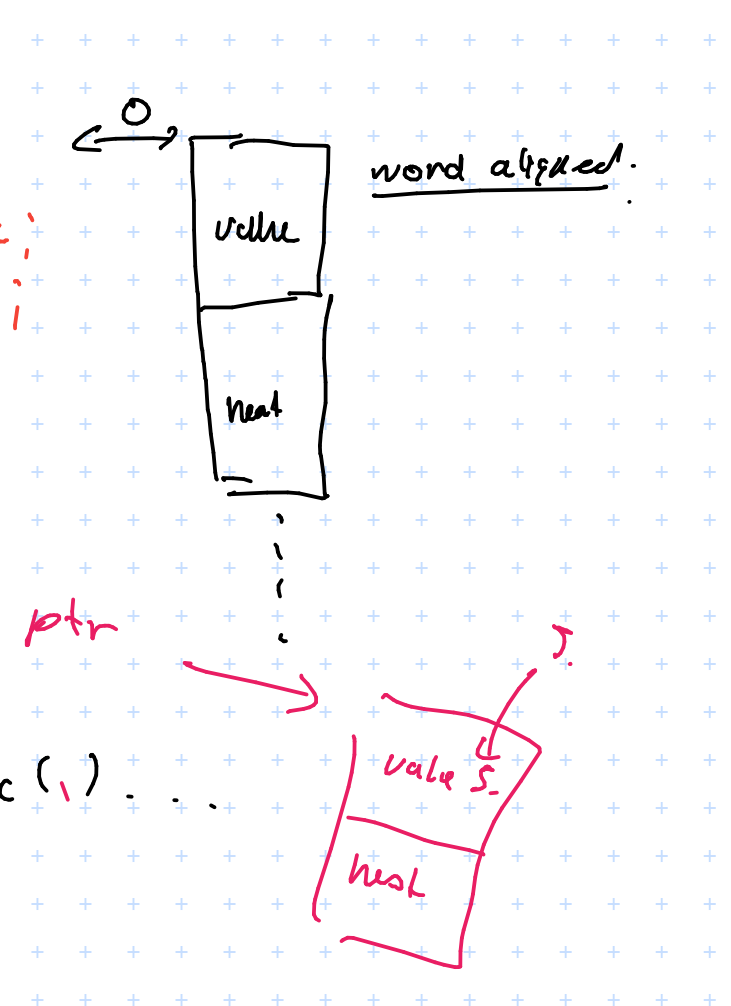
1) Structs
2) Files
3) L.L.
4) Stack + Queues

```
struct my_struct {
    type name;
    :
};
```

int value;
node* next;

word aligned.

init
acess

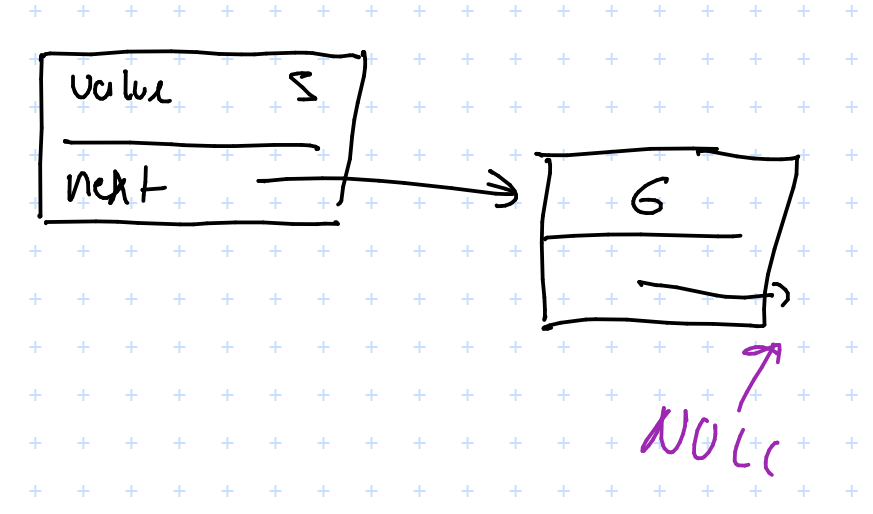



```
struct mystruct* ptr = malloc(..) ...
ptr -> val = 5;

struct mystruct s = {
    .value = 5;
    .next = &another_s;
}

struct dog {
    int age;
}

struct dog spot;
spot.age = 5;
struct dog * spot_ptr = &spot;
```

" -> "

spot_ptr -> age = 5.
(*spot_ptr).age
— they mean the same thing.

```
struct node {
    int val;
    struct node* next;
};
```



① → ② → ③ → NULL

typedef struct node node;

Iteratively

```
void print_iter(node* n){
    while (n != NULL)
        print("%, ", n->val;
        n = n->next;
    }
}
```

Recursive

① → ② → ③ → ④ ..

```
void print_rec(node* n){   // idx
    if (n == NULL){
        return;
    }
    printf("%d, ", n->val);
    print_rec(n->next);
}
```

Indexing LL

```
int idx_ll (node* n, int idx) {
    int count = 0;

    while (count ! idx) {
        n = n->next;
        count++;
    }
    return n->val;
}
```

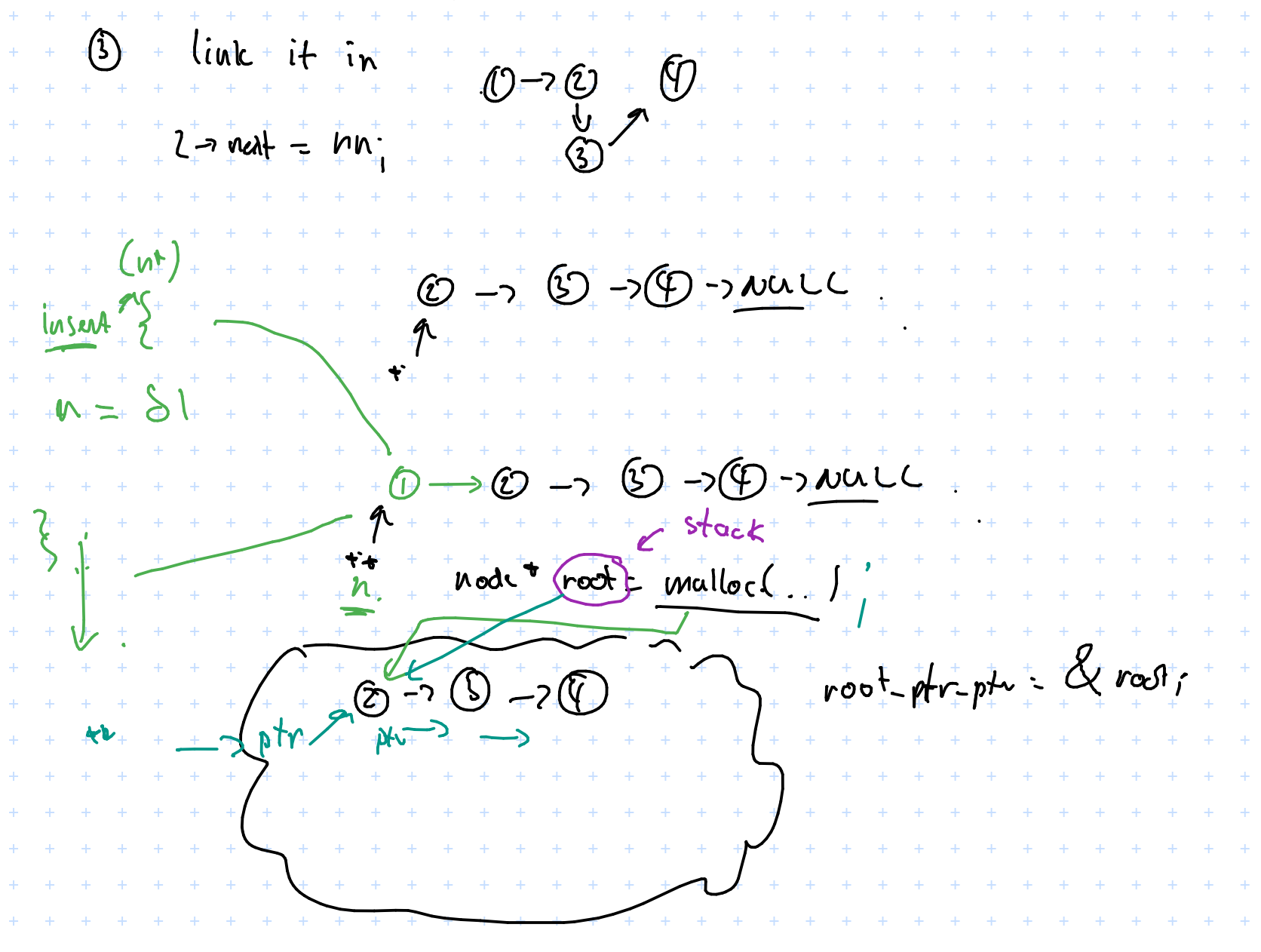
Inserting   append, prepend.

① → ② → ④

① create a new node.   ① → ② → ④
node* nn = malloc(sizeof(node));

② put in values.   ① → ② → ④, ③ ↗
nn->val = 3;
nn->next = &4;

③ link it in   ① → ② ↓ ③ ↗ ④
2->next = nn;



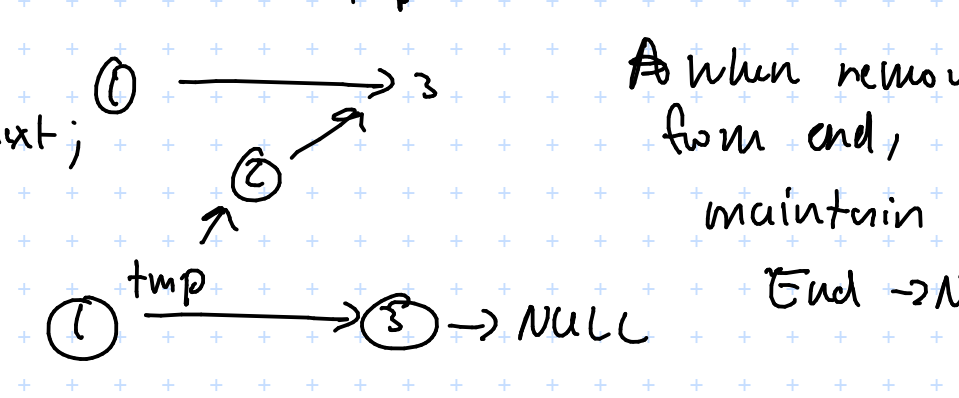
Removing an item

① → ② → ③   remove ②

1) create a tmp ref to ②
node* tmp = &②;   ① → ② → ③ (tmp)

2) unlink ②
①->next = ②->next;

③ free ②
free(tmp);



when removing from end, maintain End -> NULL.

Free a LL.

1) Iter

PREV
CURR

```
void free_iter(node** n){
    node* prev = n;
    while (n->next != NULL {
        n = n->next;
        free(prev);
        prev = n;
    }
```

2) Recursive

```
void free_rec (node** n){
    if (n==NULL){ return;}
    free_rec(n->next);
    free (n);
}
```

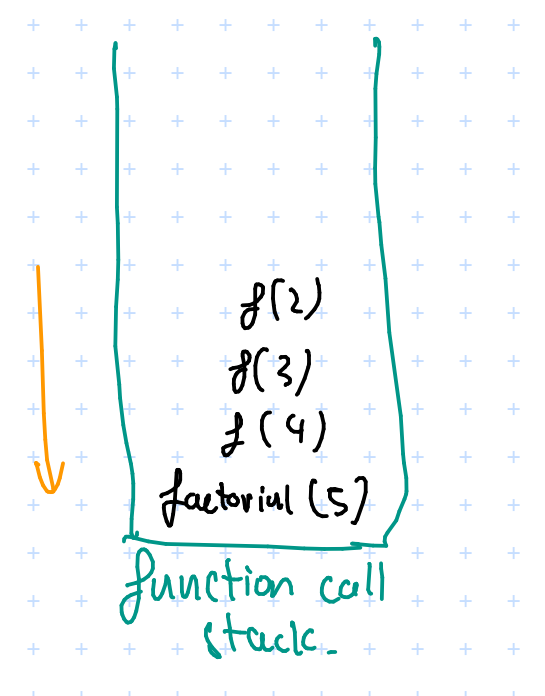
	Array	LL	
Indexing	O(1)	O(n)	
Inserting	O(n) + whatever for realloc	O(N)	to avoid realloc time/int.
Removing	"	"	

Stack
FILO
think of a can of pringles

Queue
FIFO
think of supermarket line.

```
def factorial(n):
    if n<1
        return 1
    else
        return n*factorial(n-1)
```

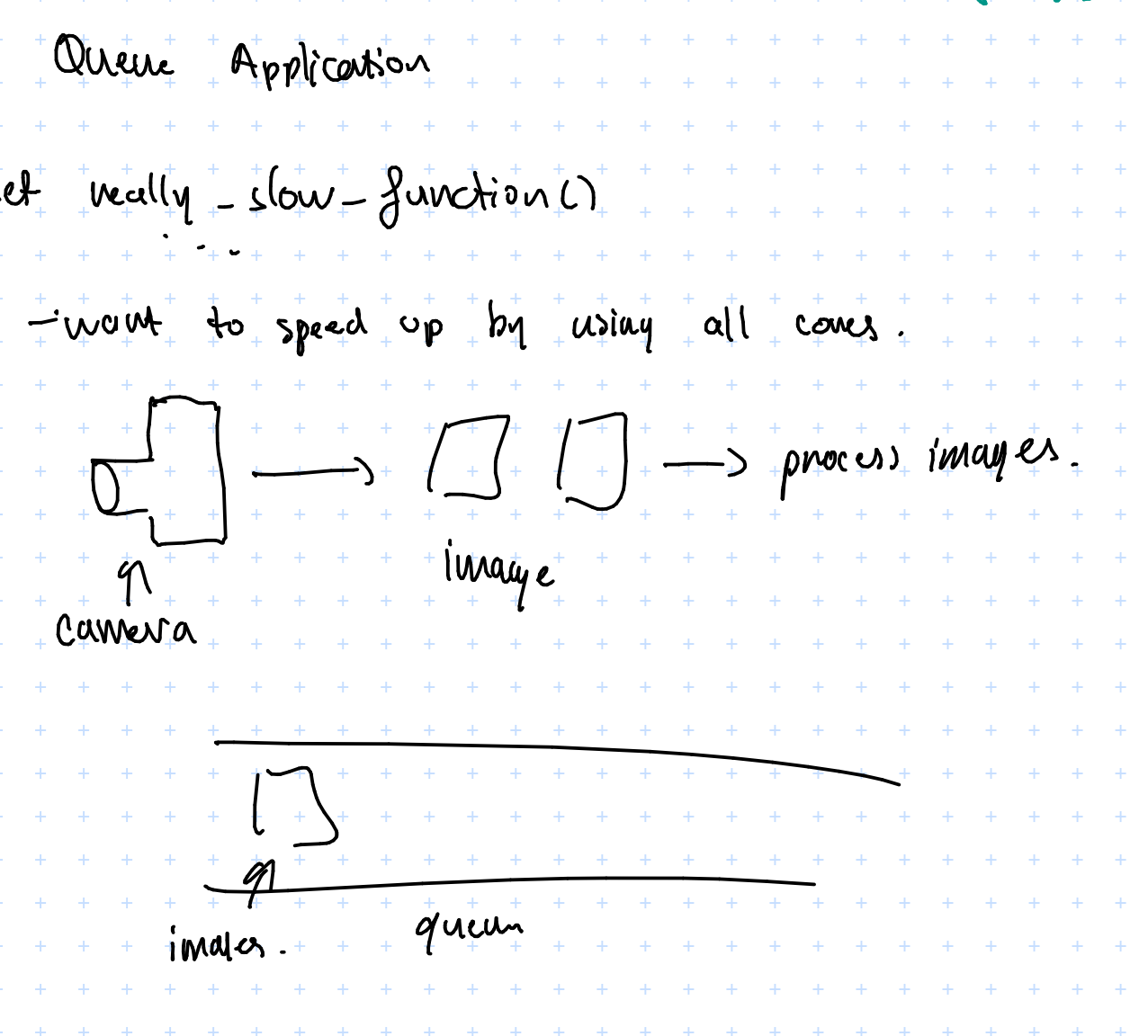
factorial(5)
↳ f(4)
↳ f(3)
↳ f(2)



Queue Application

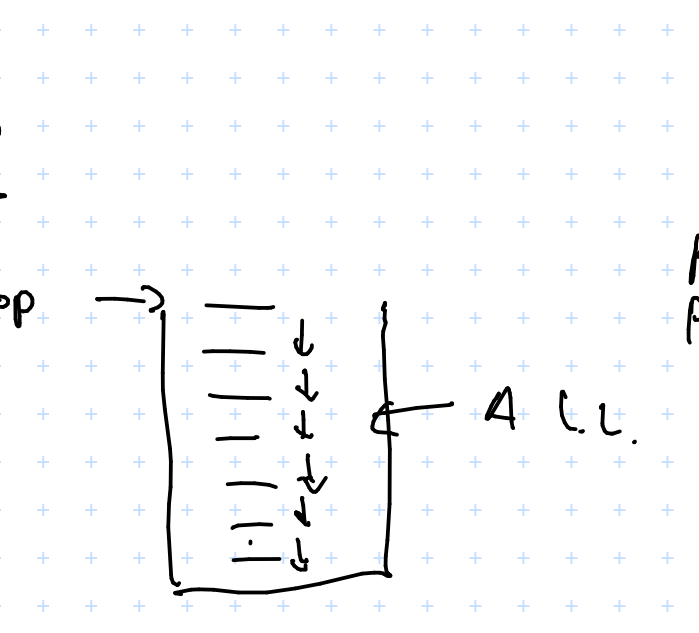
def really_slow_function()

- want to speed up by using all cores.



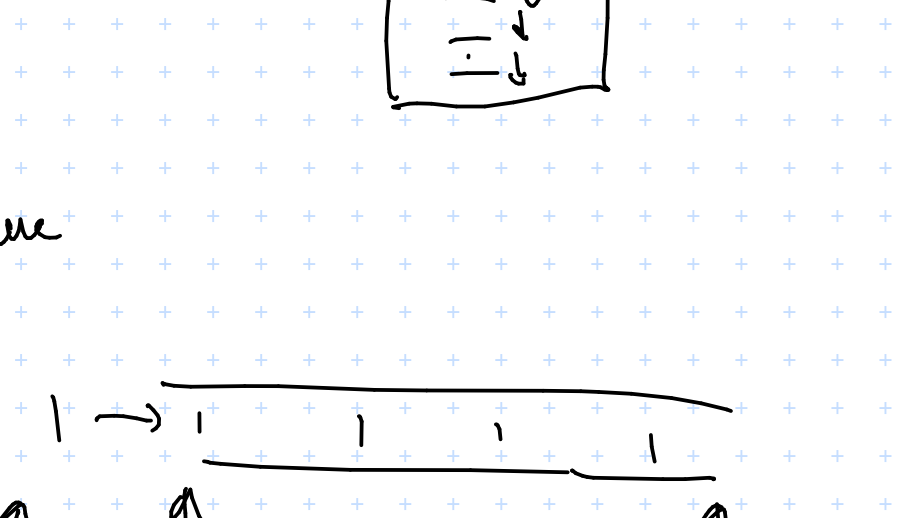
Implement?

Stack



pop() -> remove(0)
push() -> insert(0)

queue



push() -> add elem to tail
pop() -> remove item from head.

FILEIO   read: fgets getline fscanf
write: fprintf

① : open   "w" "r" "rb" "rw" ...
FILE* fp = fopen("filename", spec);

② read/write from it
read: fgets getline fscanf
write: fprintf

③ close it.
fclose(ptr);

size_t getline (char ** lineptr, size_t *n, FILE* fptr)
- return 0 @ EOF
- otherwise reads in line (\n term) from fptr & puts in line ptr.

hello world\n — File
12

```
char * line = malloc(sizeof(char) * 5);
int n = 0;
FILE* ptr = fopen("file.txt", "r");
getline (&line, &n; ptr)
```

"hello world\n"

① read things in
② try to put into line
↓
realloc if necessary
③ update n so that we know how much new memory it uses.

line
realloc(line) ← make it fit
-> then, puts "hello world\n" -> line.
*n = 12;

fprintf(target, "_", var...)
FILE*

fprintf(FILE* ptr, char* format, ...)

char * hello_world "hello world"