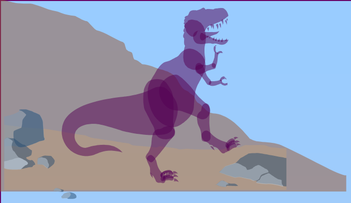
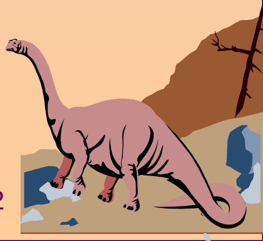
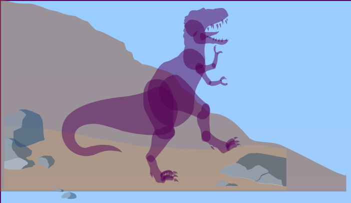


SO 1: List the characteristics of segmentation

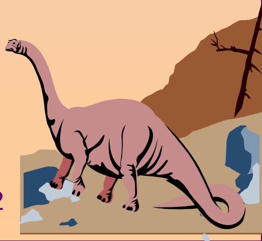


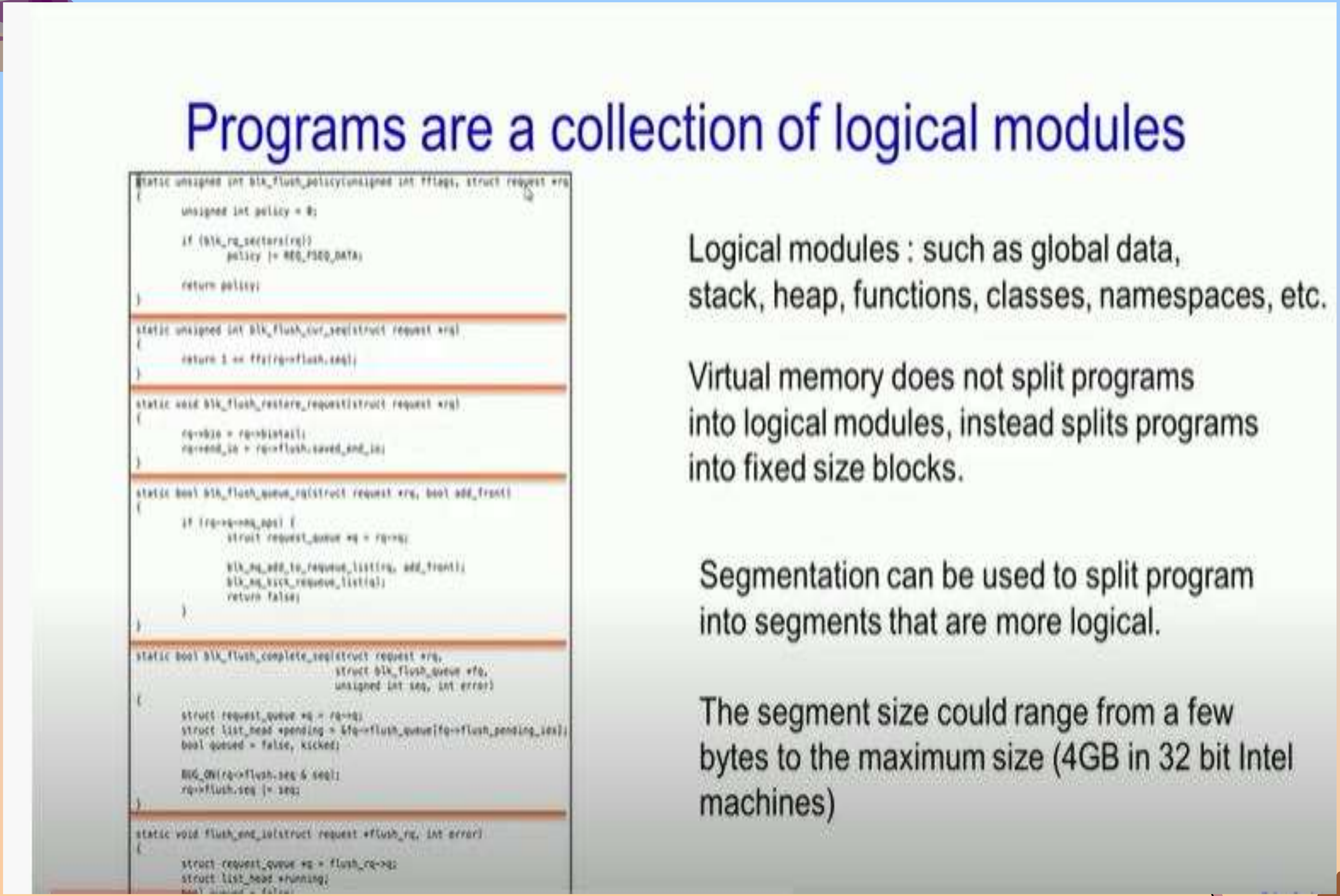


Segmentation

- Memory-management scheme that supports user view of memory.
- A program is a collection of segments. A segment is a logical unit such as:

  main program,
  procedure,
  function,
  method,
  object,
  local variables, global variables,
  common block,
  stack,
  symbol table, arrays





Programs are a collection of logical modules

```
static unsigned int blk_flush_policy(unsigned int fflags, struct request *rq)
{
        unsigned int policy = 0;

        if (blk_rq_sectors(rq))
                policy |= REQ_FSEQ_DATA;

        return policy;
}
```

```
static unsigned int blk_flush_cur_seq(struct request *rq)
{
        return 1 << ffz(rq->flush.seq);
}
```

```
static void blk_flush_restore_request(struct request *rq)
{
        rq->bio = rq->biotail;
        rq->end_io = rq->flush.saved_end_io;
}
```

```
static bool blk_flush_queue_rq(struct request *rq, bool add_front)
{
        if (rq->q->mq_ops) {
                struct request_queue *q = rq->q;

                blk_mq_add_to_requeue_list(rq, add_front);
                blk_mq_kick_requeue_list(q);
                return false;
        }
}
```

```
static bool blk_flush_complete_seq(struct request *rq,
                                   struct blk_flush_queue *fq,
                                   unsigned int seq, int error)
{
        struct request_queue *q = rq->q;
        struct list_head *pending = &fq->flush_queue[fq->flush_pending_idx];
        bool queued = false, kicked;

        BUG_ON(rq->flush.seq & seq);
        rq->flush.seq |= seq;
}
```

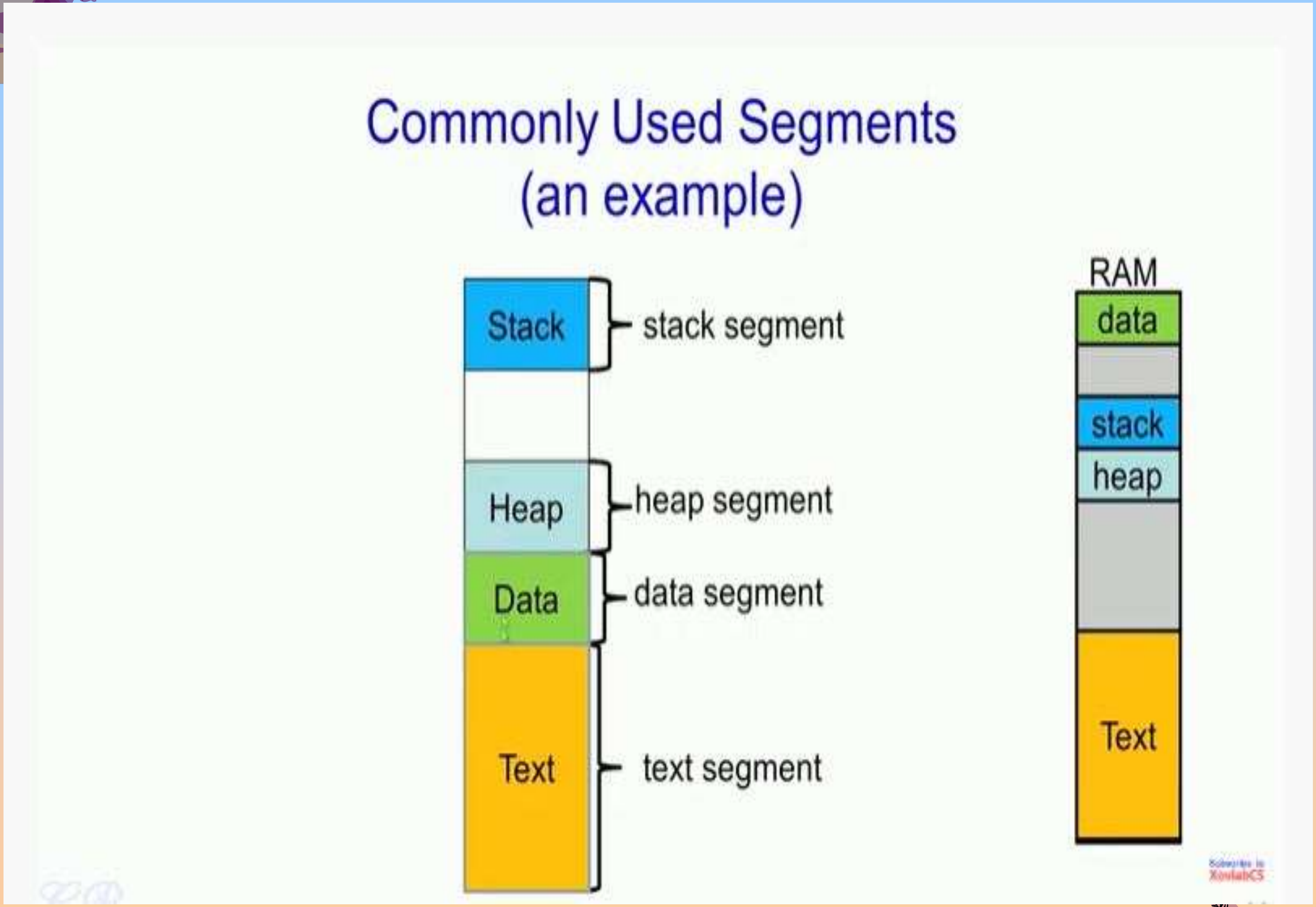
```
static void flush_end_io(struct request *flush_rq, int error)
{
        struct request_queue *q = flush_rq->q;
        struct list_head *running;
        bool queued = false;
```

Logical modules : such as global data, stack, heap, functions, classes, namespaces, etc.

Virtual memory does not split programs into logical modules, instead splits programs into fixed size blocks.

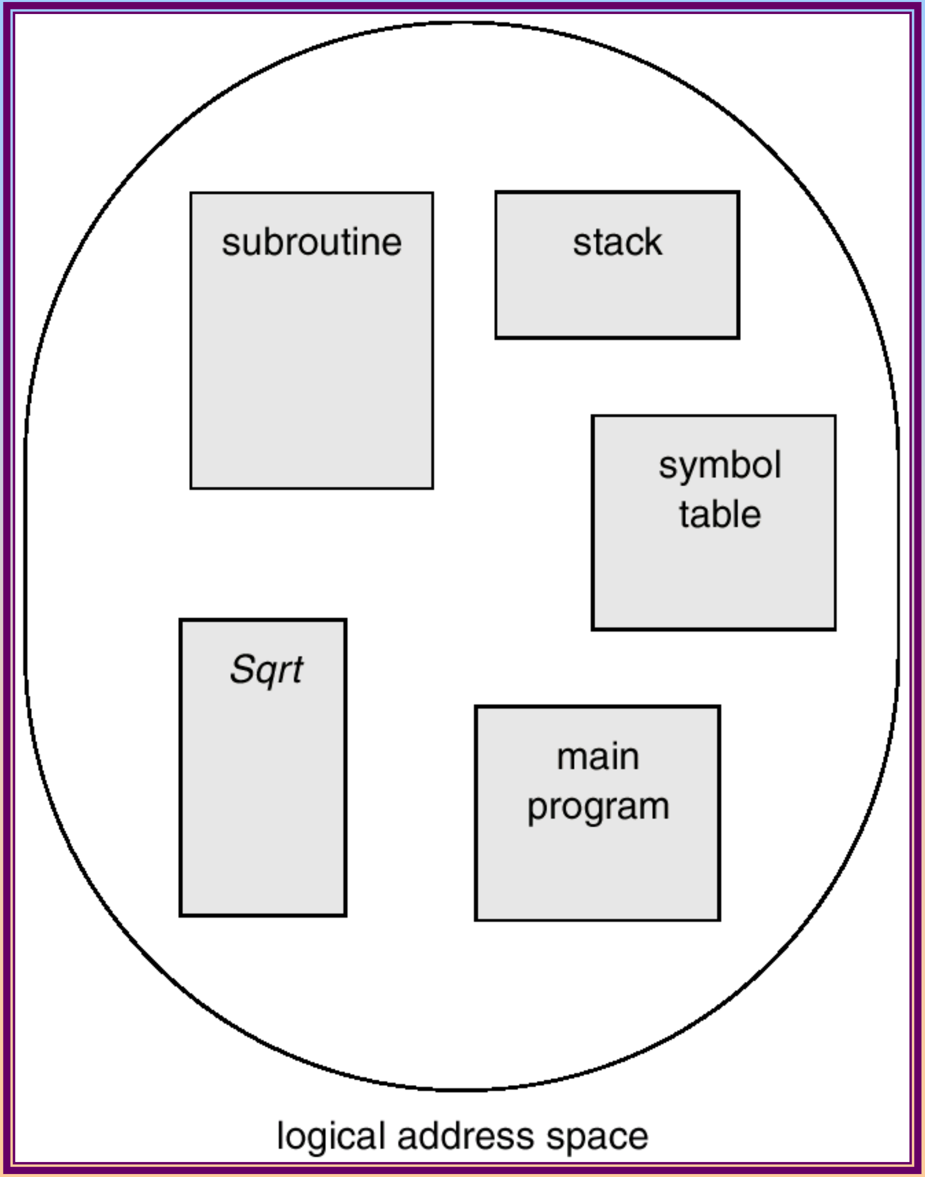
Segmentation can be used to split program into segments that are more logical.

The segment size could range from a few bytes to the maximum size (4GB in 32 bit Intel machines)

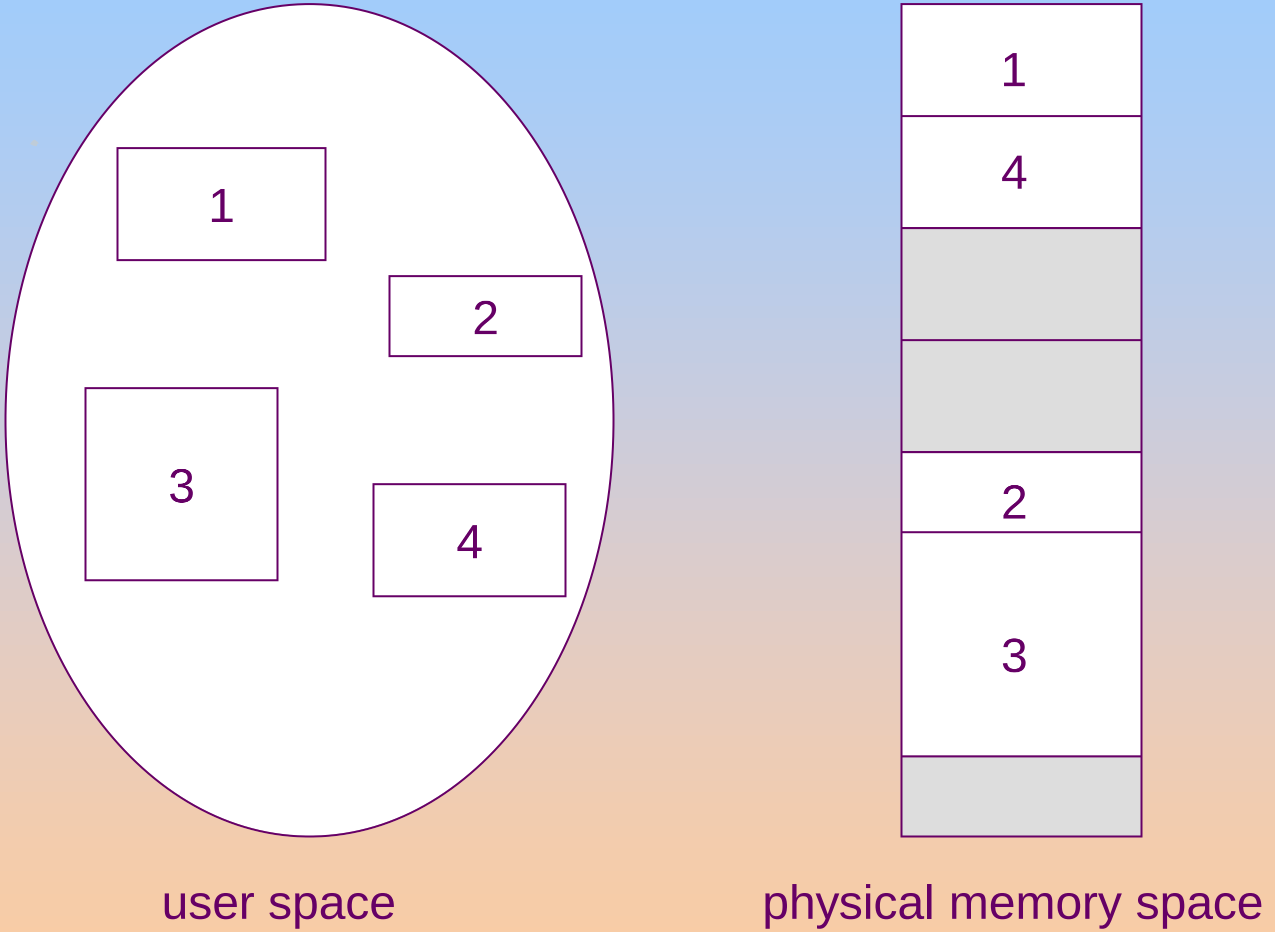


Commonly Used Segments (an example)

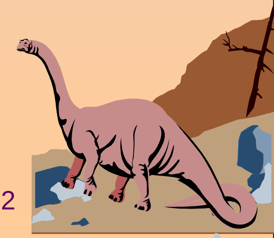
User’s View of a Program

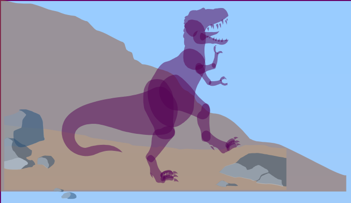


Logical View of Segmentation



SO 2: Illustrate the segmentation architecture

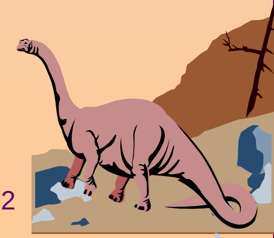


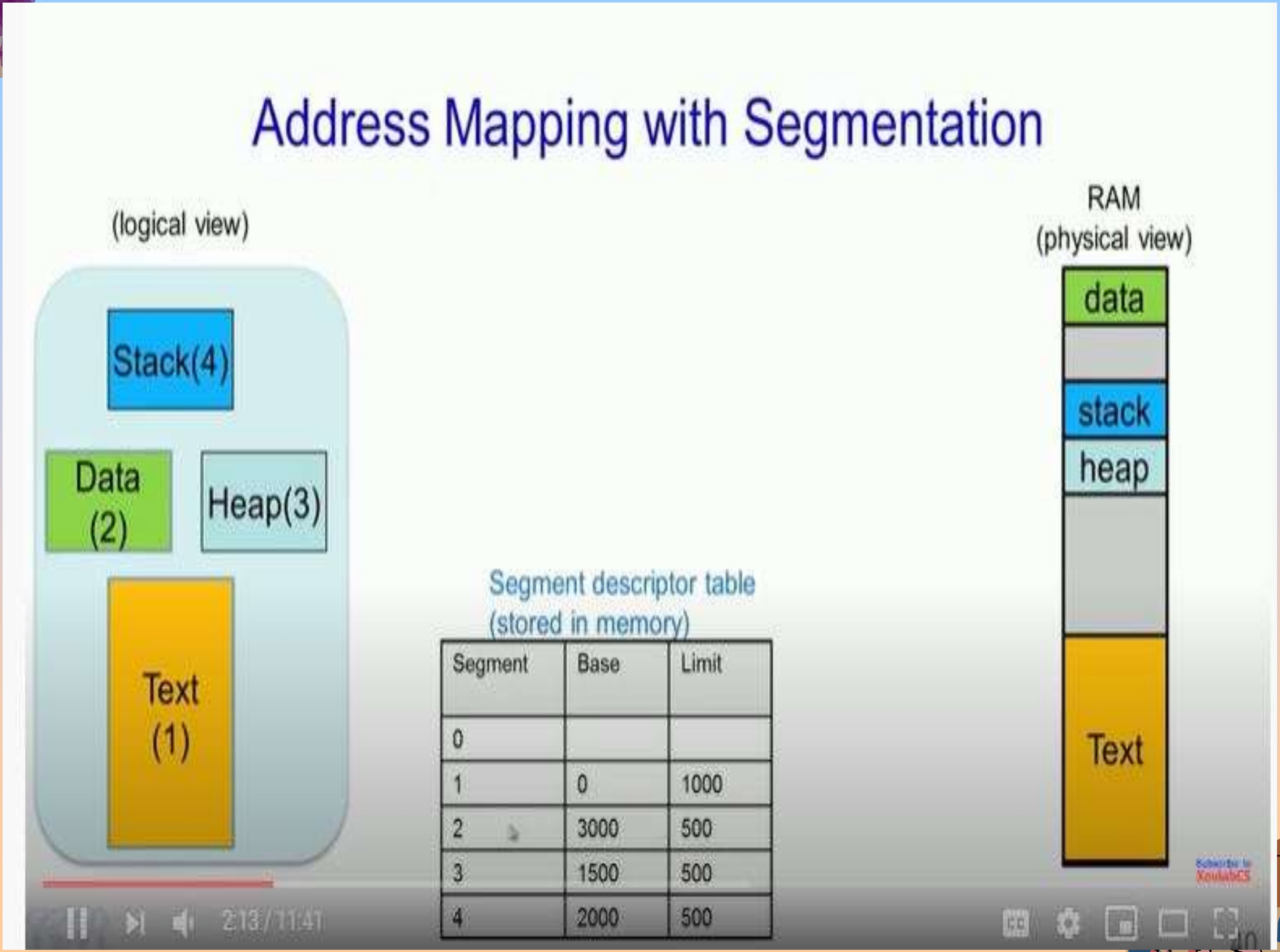


Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,
- Segment table – maps two-dimensional physical addresses; each table entry has:
  - base – contains the starting physical address where the segments reside in memory.
  - limit – specifies the length of the segment.
- Segment-table base register (STBR) points to the segment table's location in memory.

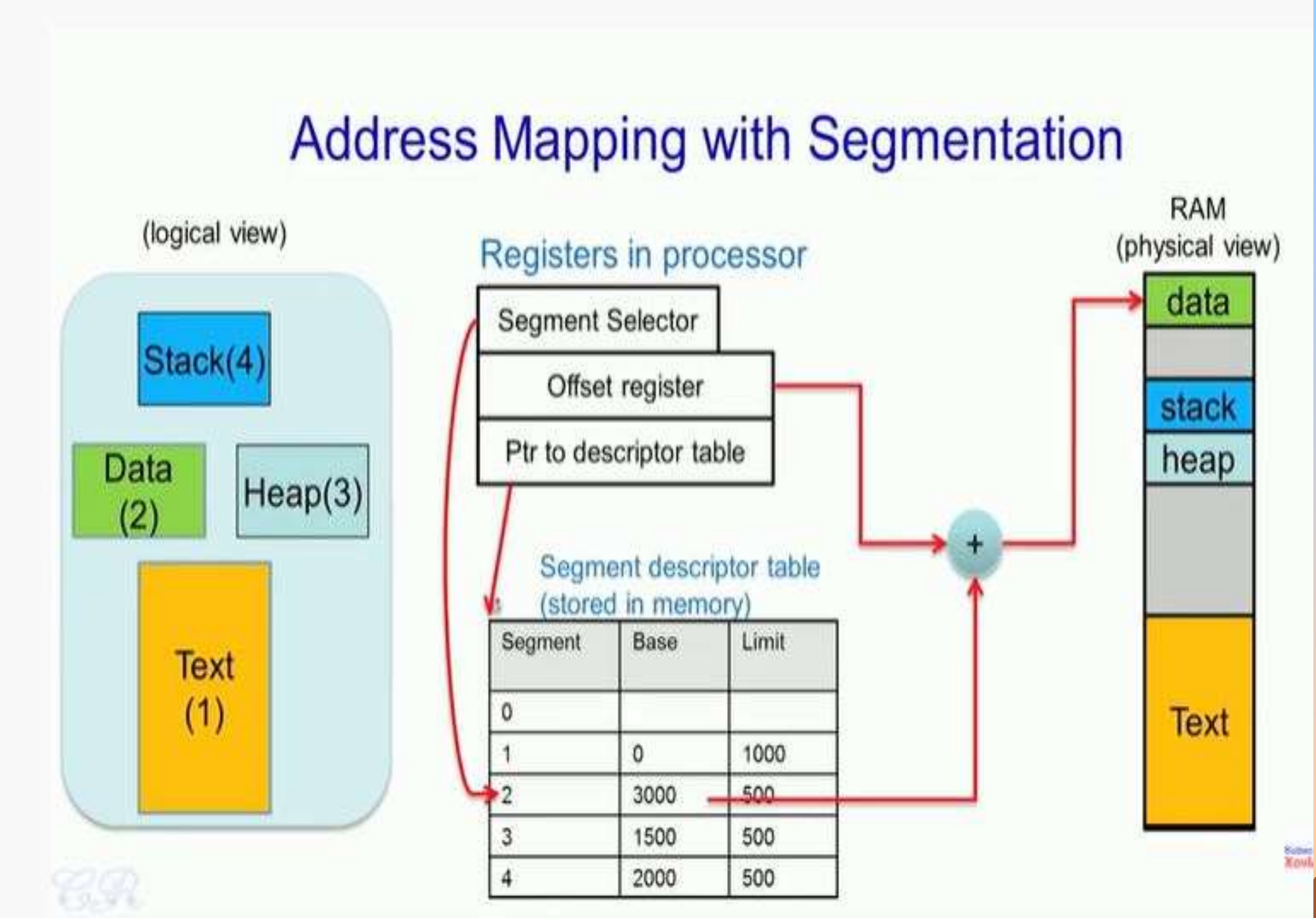




Address Mapping with Segmentation

Segment descriptor table
(stored in memory)

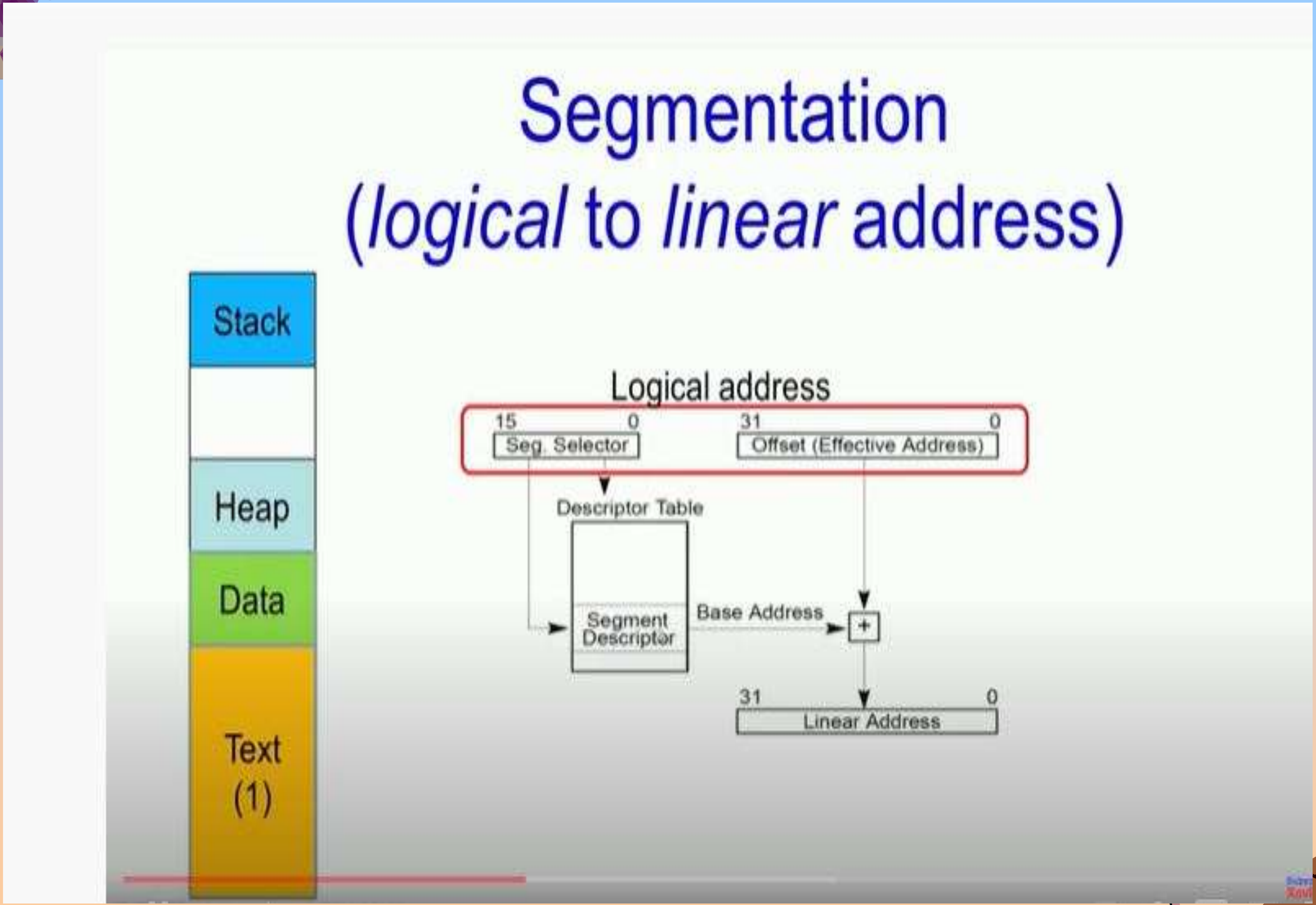
Segment	Base	Limit
0		
1	0	1000
2	3000	500
3	1500	500
4	2000	500



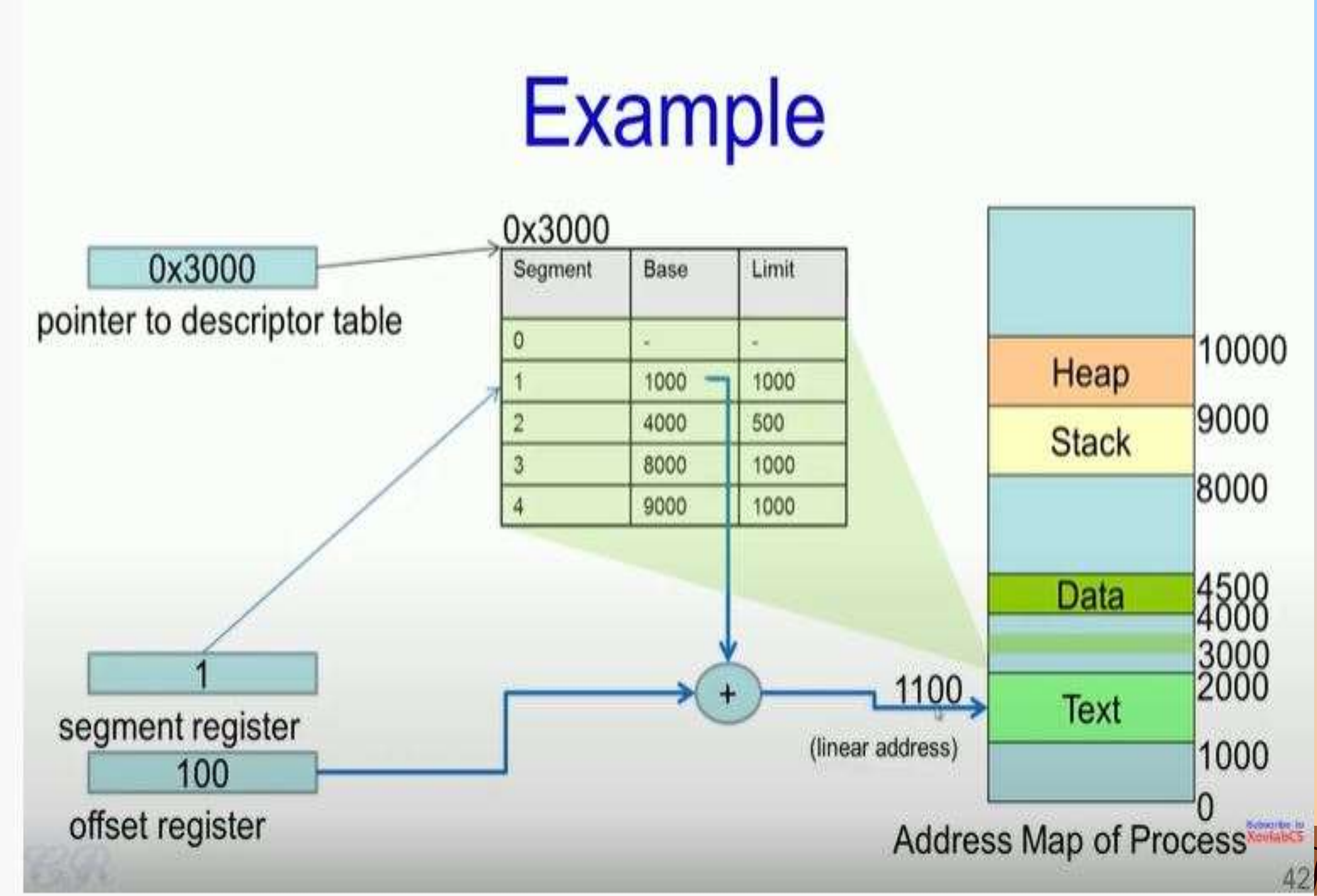
Address Mapping with Segmentation

Segment descriptor table (stored in memory)

Segment	Base	Limit
0		
1	0	1000
2	3000	500
3	1500	500
4	2000	500



Segmentation
(*logical* to *linear* address)

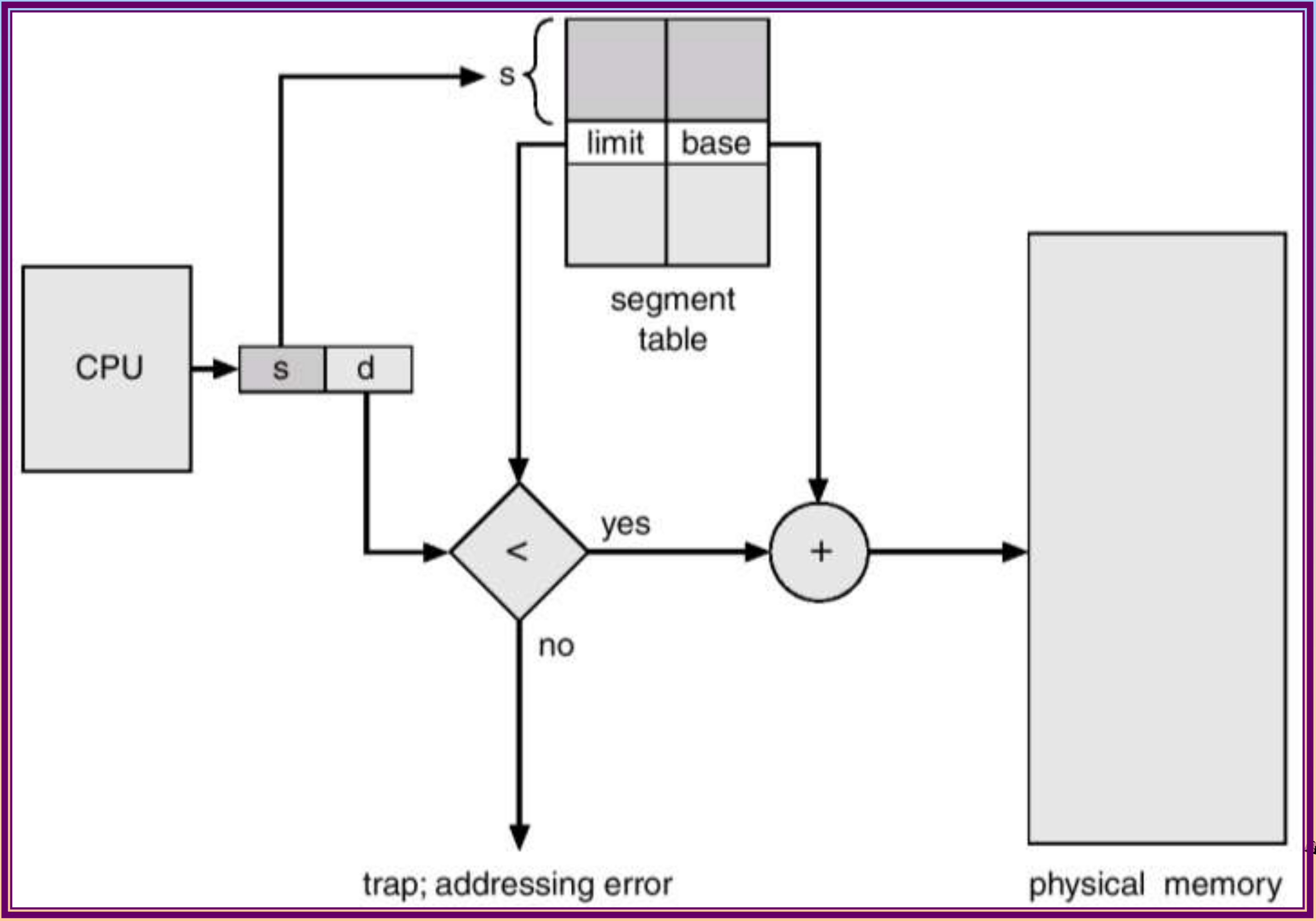


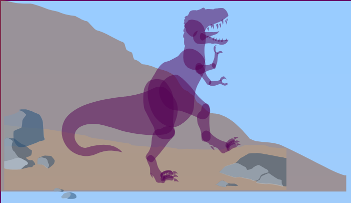
Example

0x3000

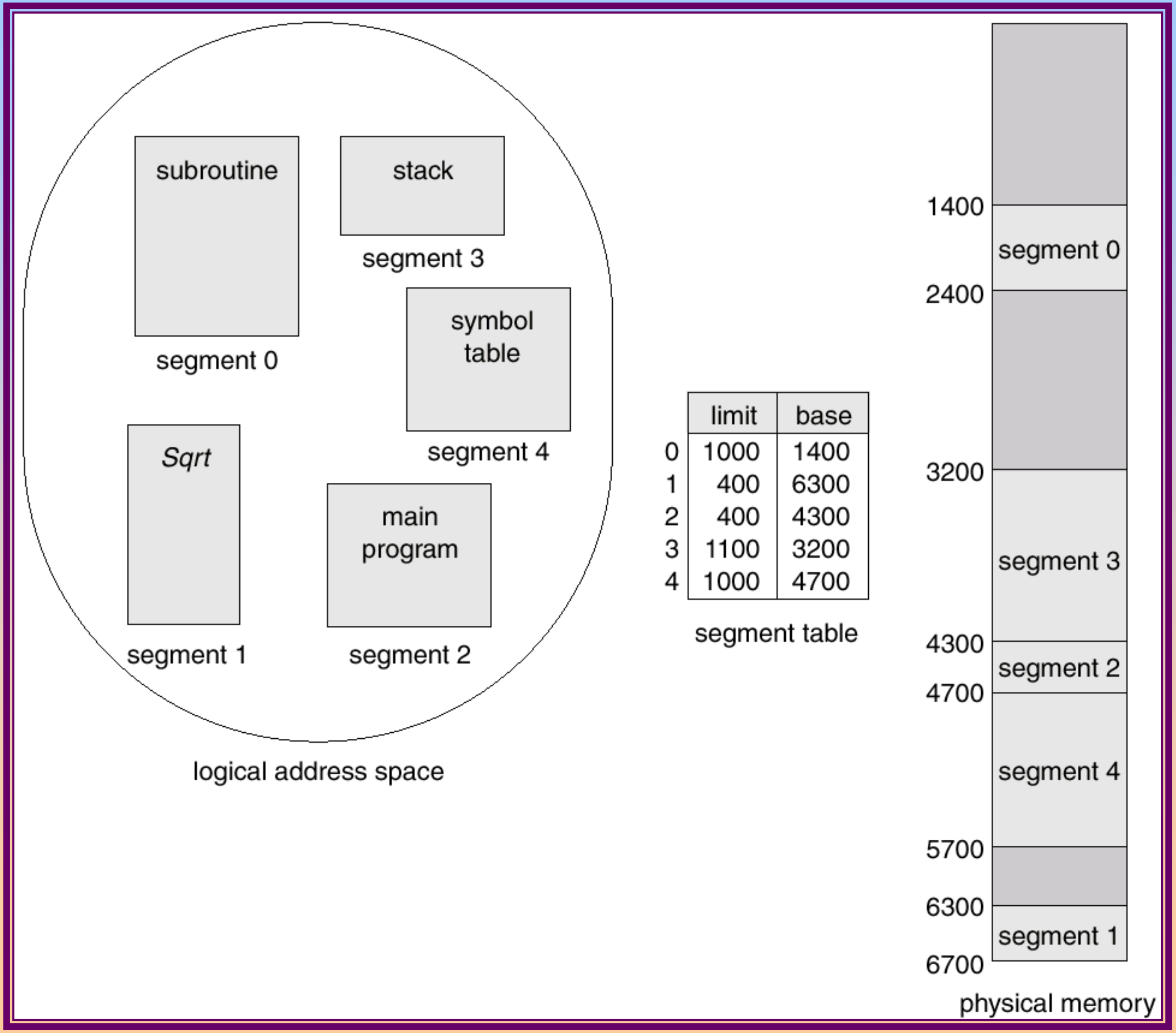
Segment	Base	Limit
0	-	-
1	1000	1000
2	4000	500
3	8000	1000
4	9000	1000

Segmentation Hardware

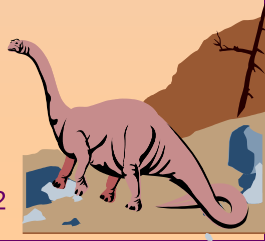


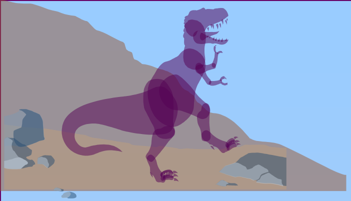


Example of Segmentation

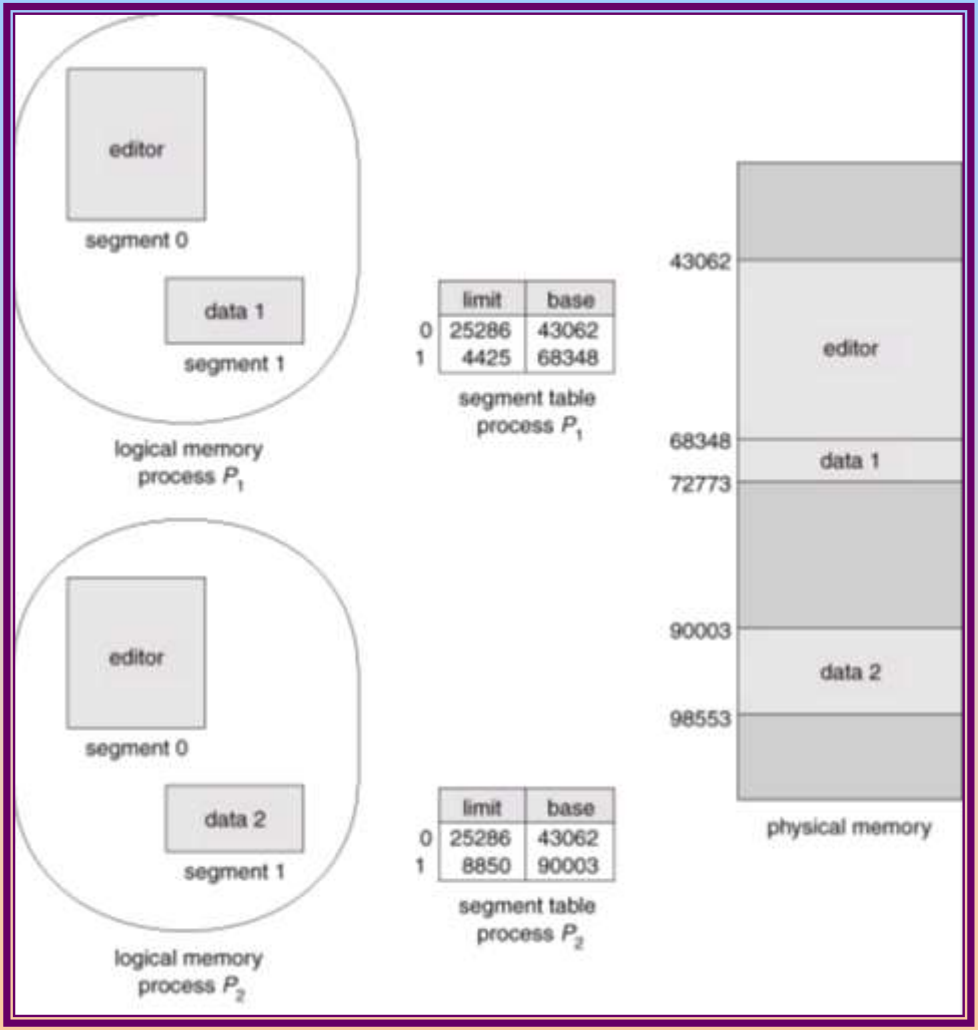


	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700





Sharing of Segments

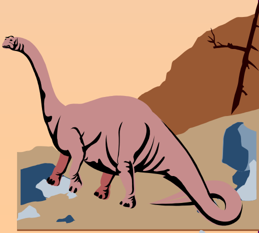


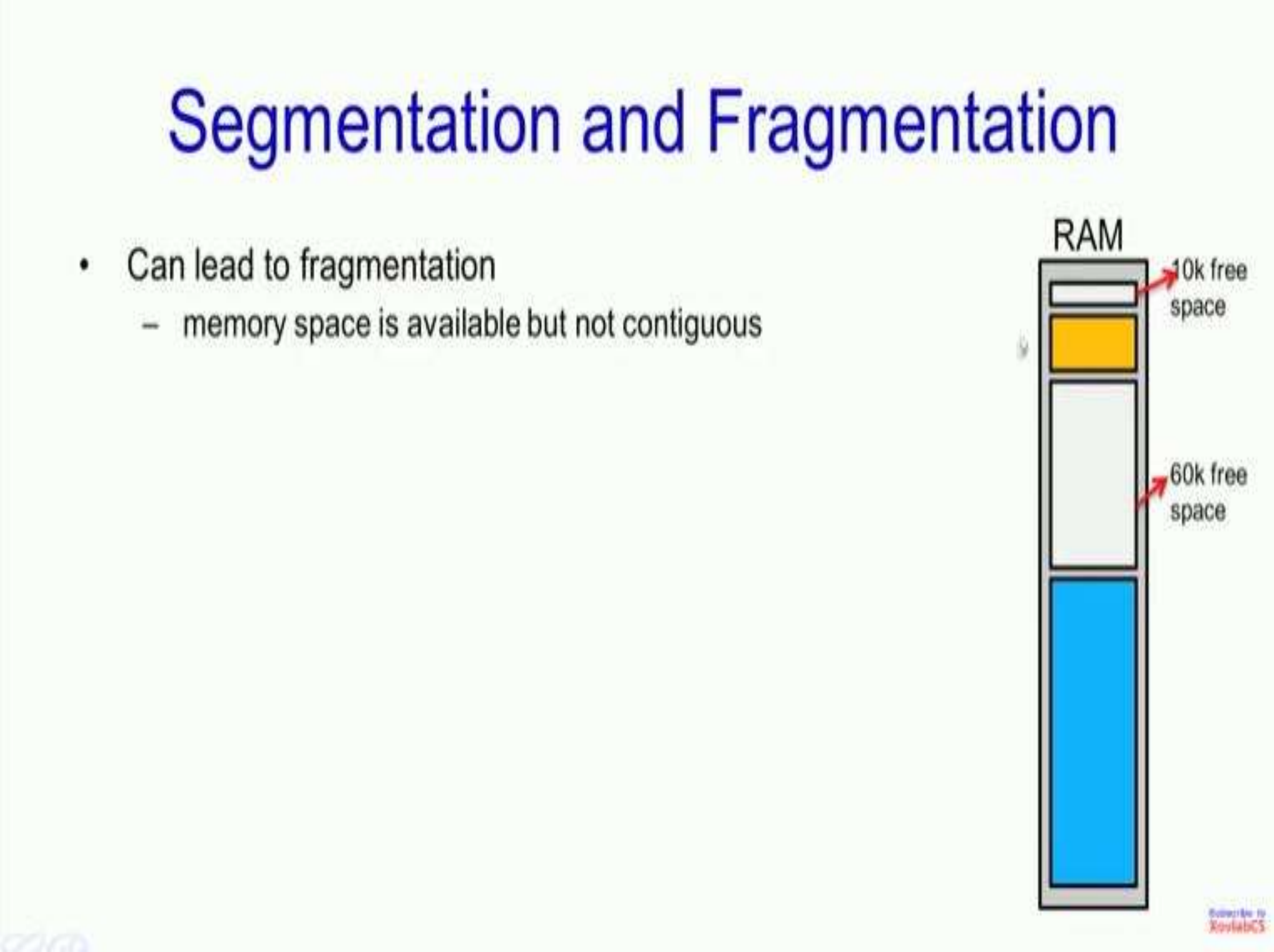
	limit	base
0	25286	43062
1	4425	68348

segment table
process P_1

	limit	base
0	25286	43062
1	8850	90003

segment table
process P_2





Segmentation and Fragmentation

- Can lead to fragmentation
  - memory space is available but not contiguous