Multics

Background

- Multics
  - Multiplexed Information and Computing Service
- Unix
  - Uniplexed Information and Computing Service
- Many great inventions
  - Virtual memory
  - Security protection
  - File system

Goals

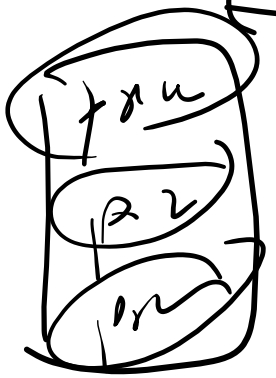
- “so the system would effectively serve the computing needs of a large community of users with diverse interests, operating principally from remote terminals”

(1) To provide the user with a large machine-independent virtual memory, thus placing the responsibility for the management of physical storage with the system software. By this means the user is provided with an address space large enough to eliminate the need for complicated buffering and overlay techniques. Users, therefore, are relieved of the burden of preplanning the transfer of information between storage levels, and user programs become independent of the nature of the various storage devices in the system.

Goal

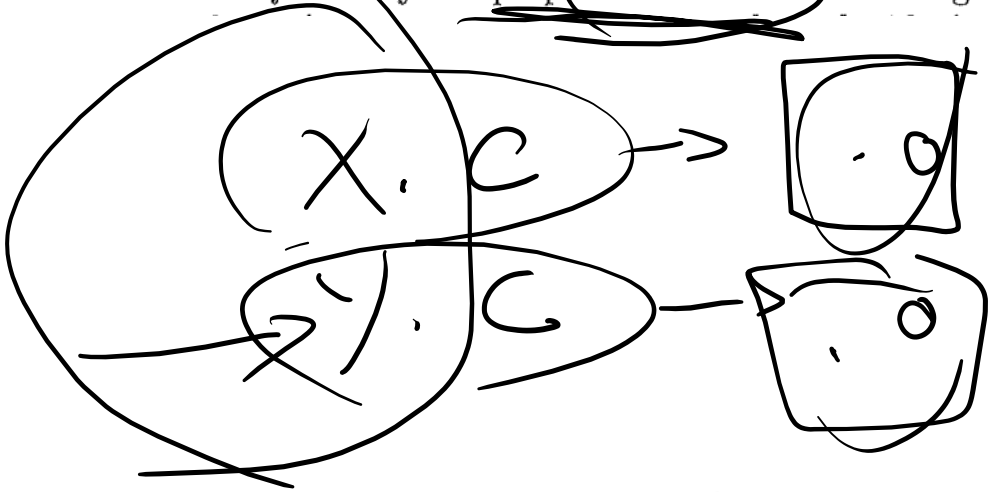
printf( ); dynamic linking

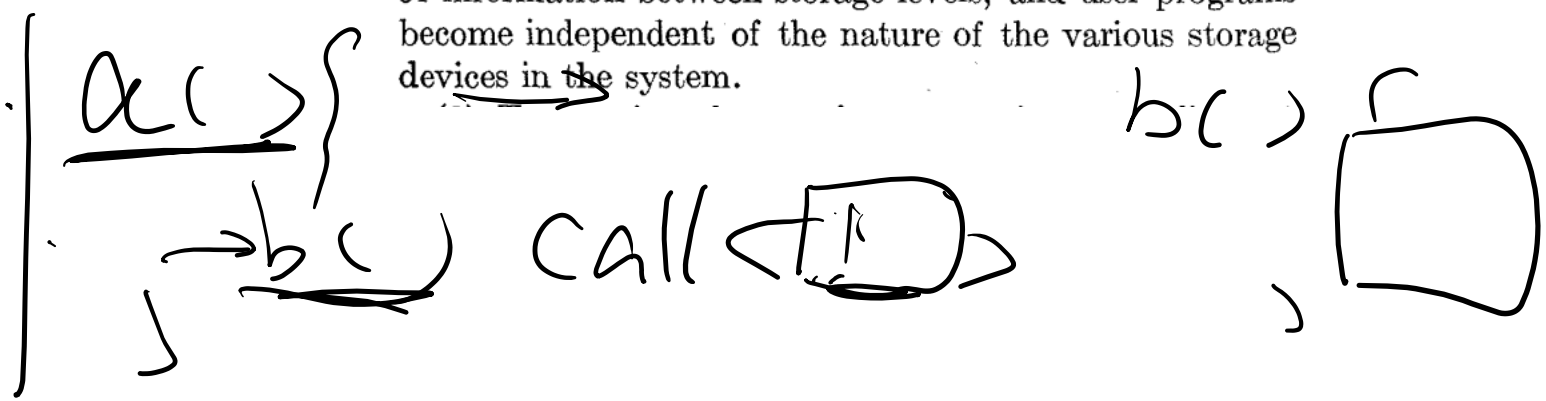
(2) To permit a degree of programming generality not previously practical. This includes the ability of one procedure to use another procedure knowing only its name, and without knowledge of its requirements for storage, or the additional procedures upon which it may in turn call.



(1) To provide the user with a large machine-independent virtual memory, thus placing the responsibility for the management of physical storage with the system software. By this means the user is provided with an address space large enough to eliminate the need for complicated buffering and overlay techniques. Users, therefore, are relieved of the burden of preplanning the transfer of information between storage levels, and user programs become independent of the nature of the various storage devices in the system.

(3) To permit sharing of procedures and data among users subject only to proper authorization. Sharing of





Outline

- Virtual memory background
- Virtual memory in Multics
- Data sharing
- Code sharing

Backround

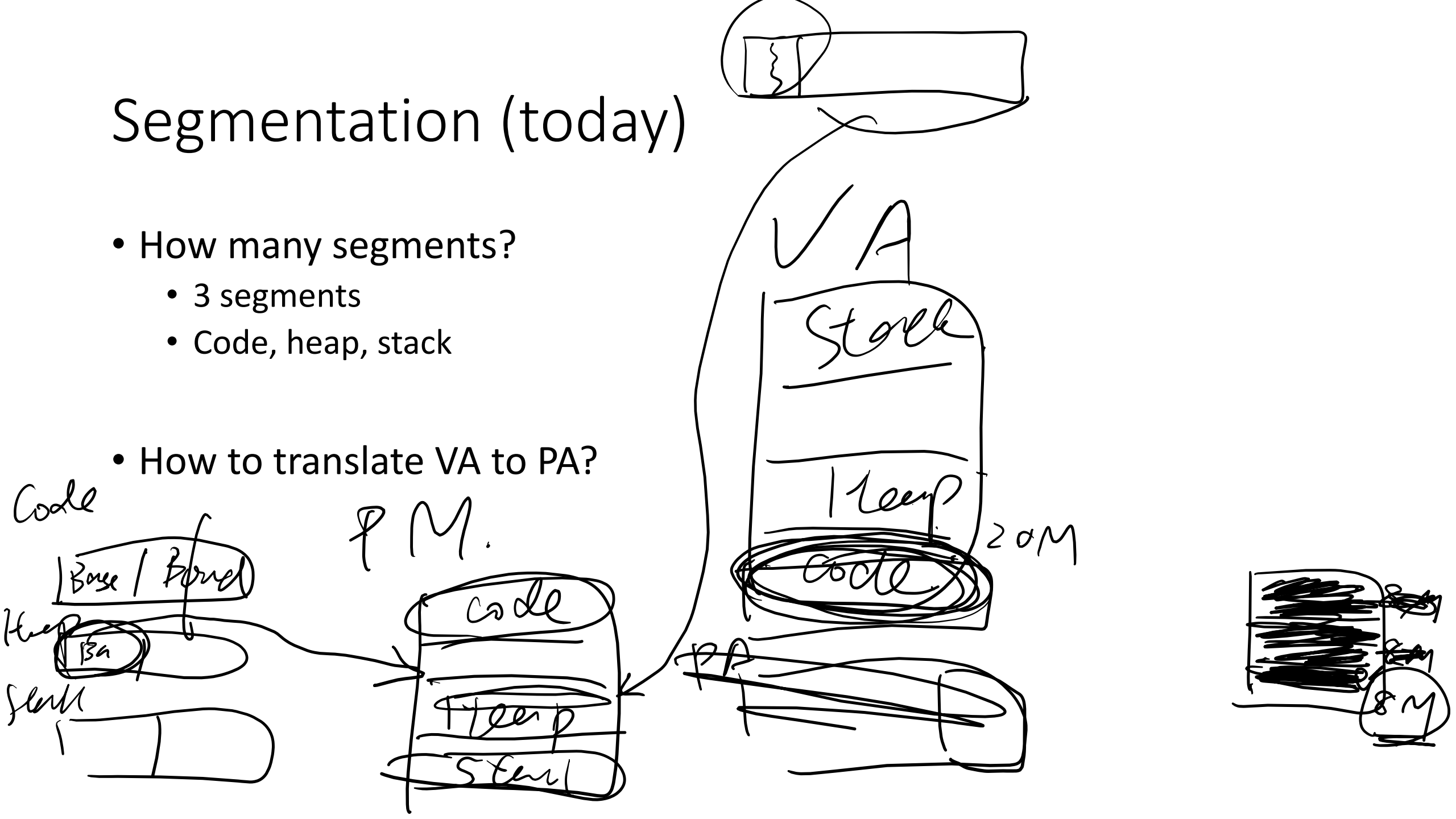
Virtual address, physical address, address space

Benefits of virtual memory

- Programming becomes easy
- More portable across machines
- Easier physical memory management
  - No need of contiguous allocation
  - Large virtual address space on small physical memory
- Benefit for MULTICS particularly
  - Code sharing, data sharing (dynamic link)

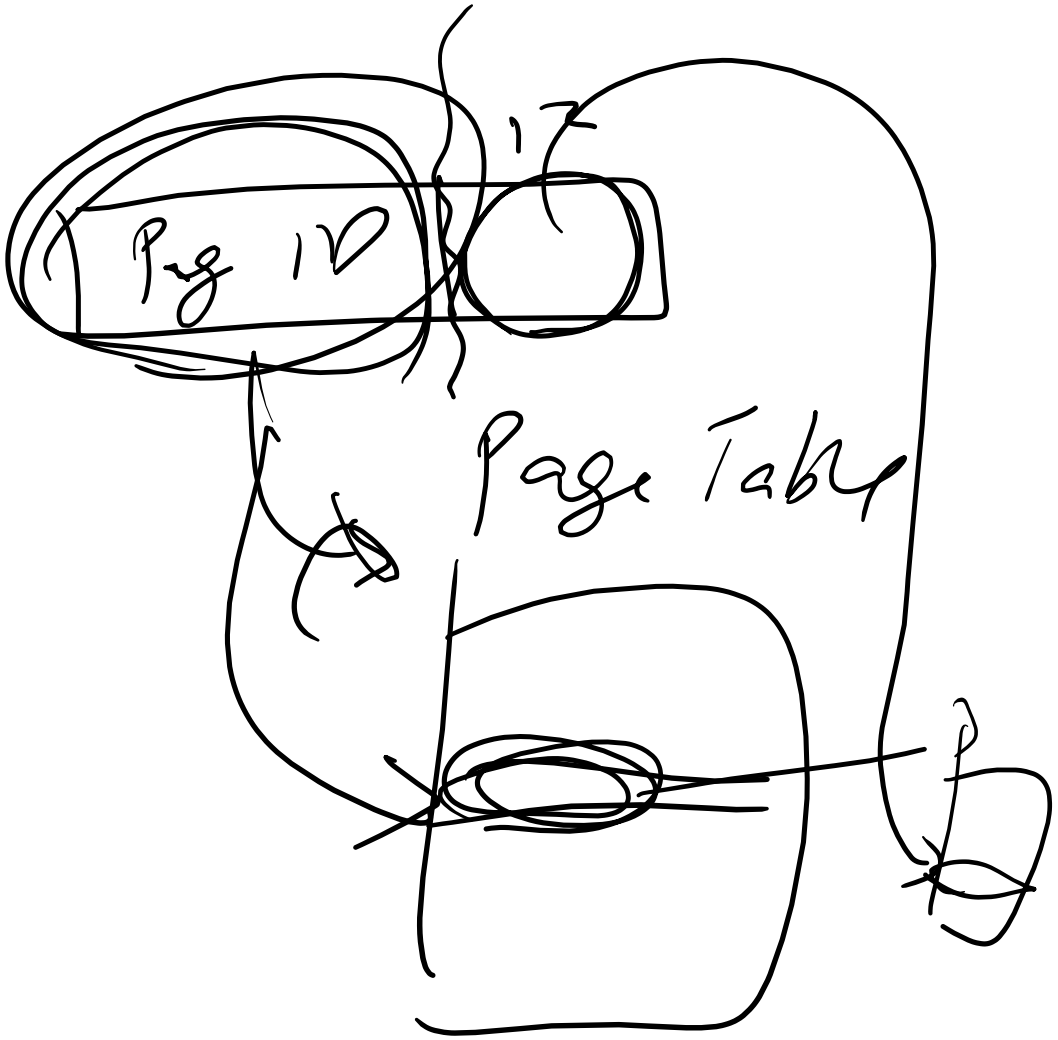
Segmentation (today)

- How many segments?
  - 3 segments
  - Code, heap, stack
- How to translate VA to PA?



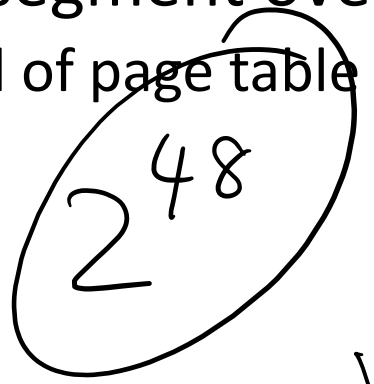
Paging (today)

- Page frame
- Page
- Page table



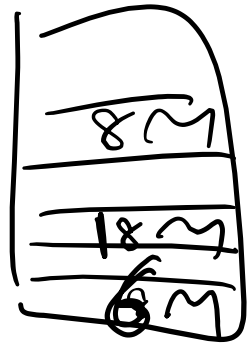
Segment vs. Paging

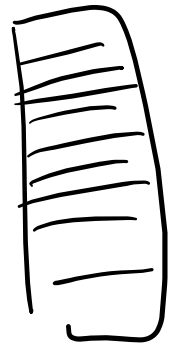
- What is the benefit of paging over segment?
  - Make better use of physical memory
    - Address external fragmentation
  - Lower the amount of needed physical memory to run …
- What is the benefit of segment over paging?
  - Eliminate the overhead of page table



2^{12}

2^{36}

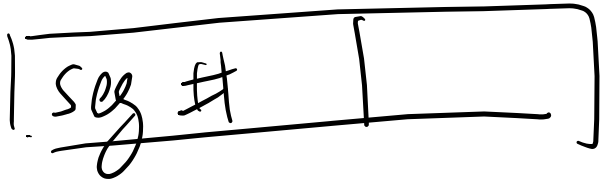




Virtual Memory in Multics

Virtual address

- What are the two components of VA in Multics?

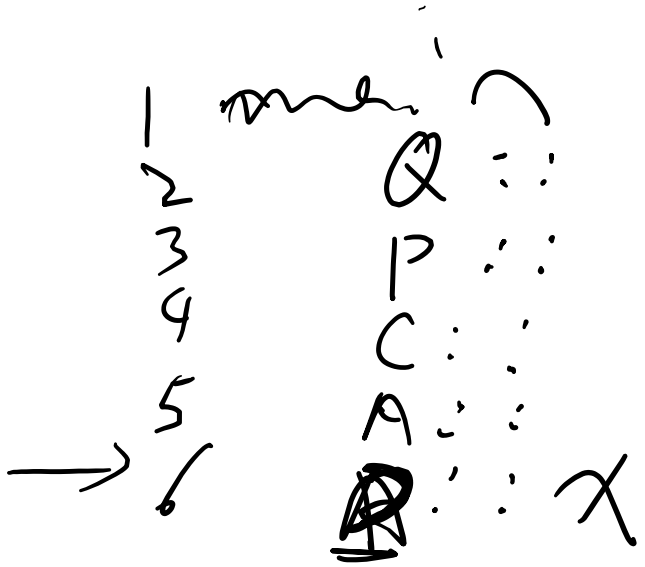


Segment #	Word #
14 bits	18 bits

2^{14}

1 — main

2 — D::X



Where does the GA (VA) come from?

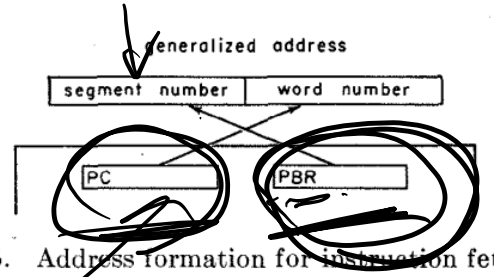


FIG. 5. Address formation for instruction fetch

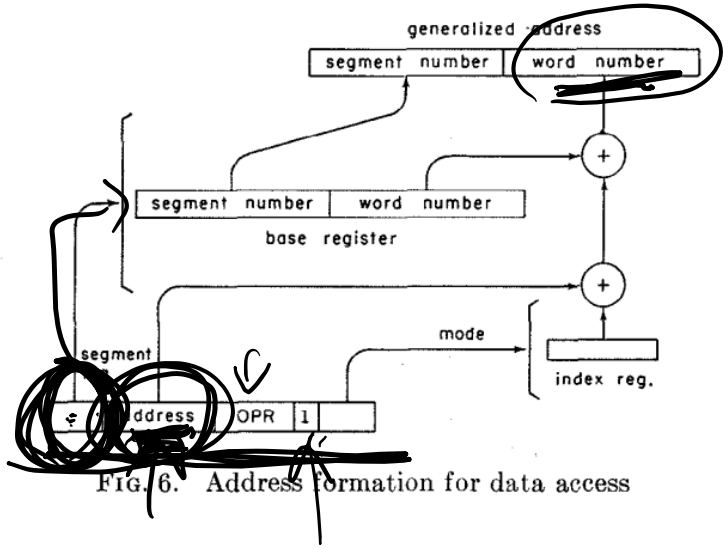
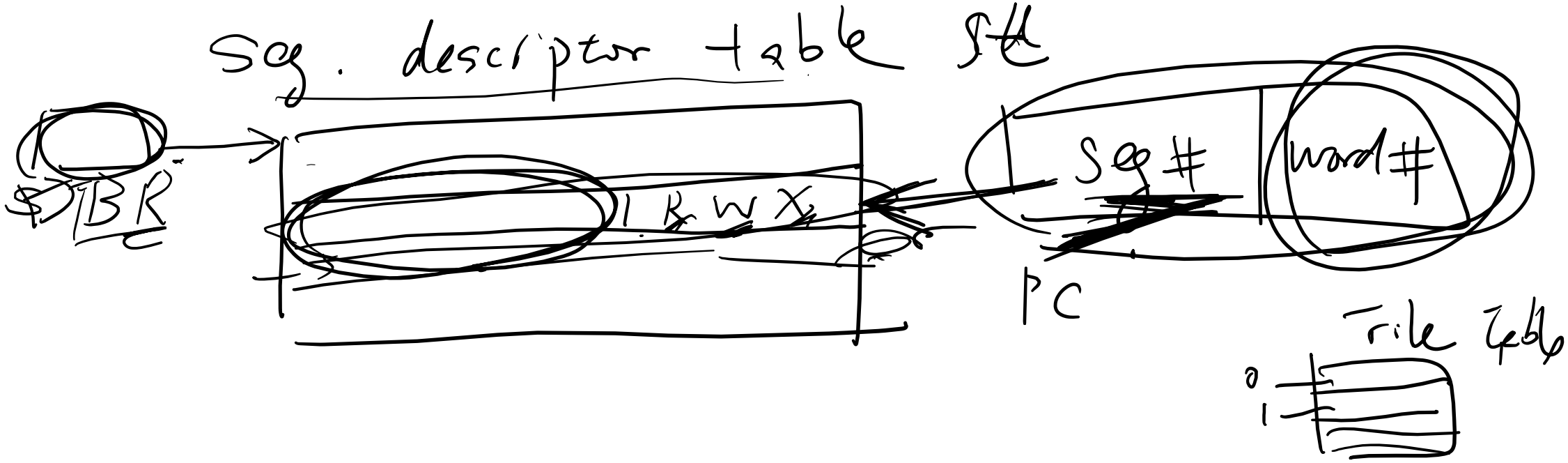


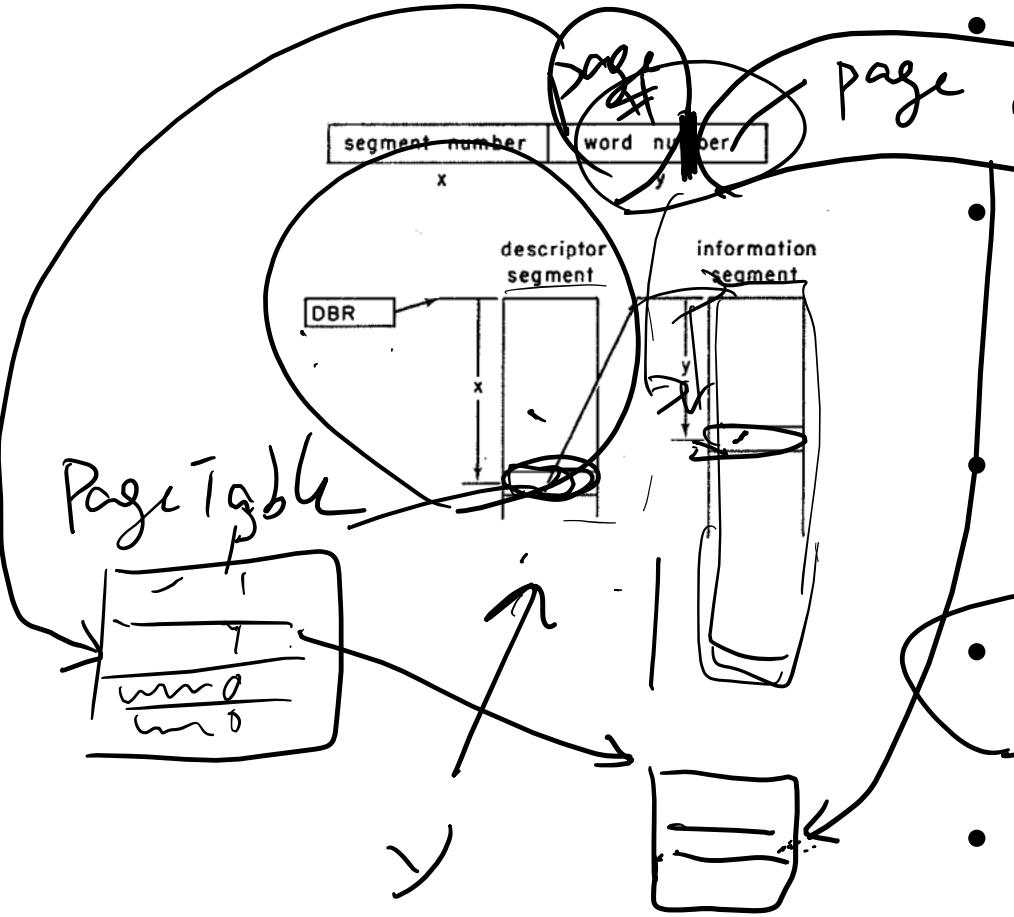
FIG. 6. Address formation for data access

How to translate VA → PA?

- Where is each segment's information stored?

- How does the machine locate that place?





- What else is inside a segment descriptor?
- Does every process have its own descriptor segment?
- What set up the descriptor segment?
- What if the segment is very long?
- What needs to change at context switch?

Data & Code sharing

dymanic linking

Requirements / Goals

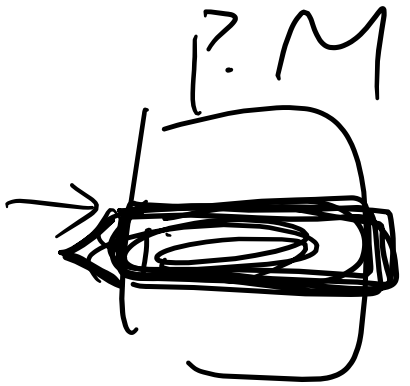
$D_A :: X$

(1) Procedure segments must be *pure*; that is, their execution must not cause a single word of their content to be modified.

read only

(2) It must be possible for a process to call a routine by its symbolic name without having made prior arrangements for its use.

(3) Segments of procedure must be invariant to the recompilation of other segments.



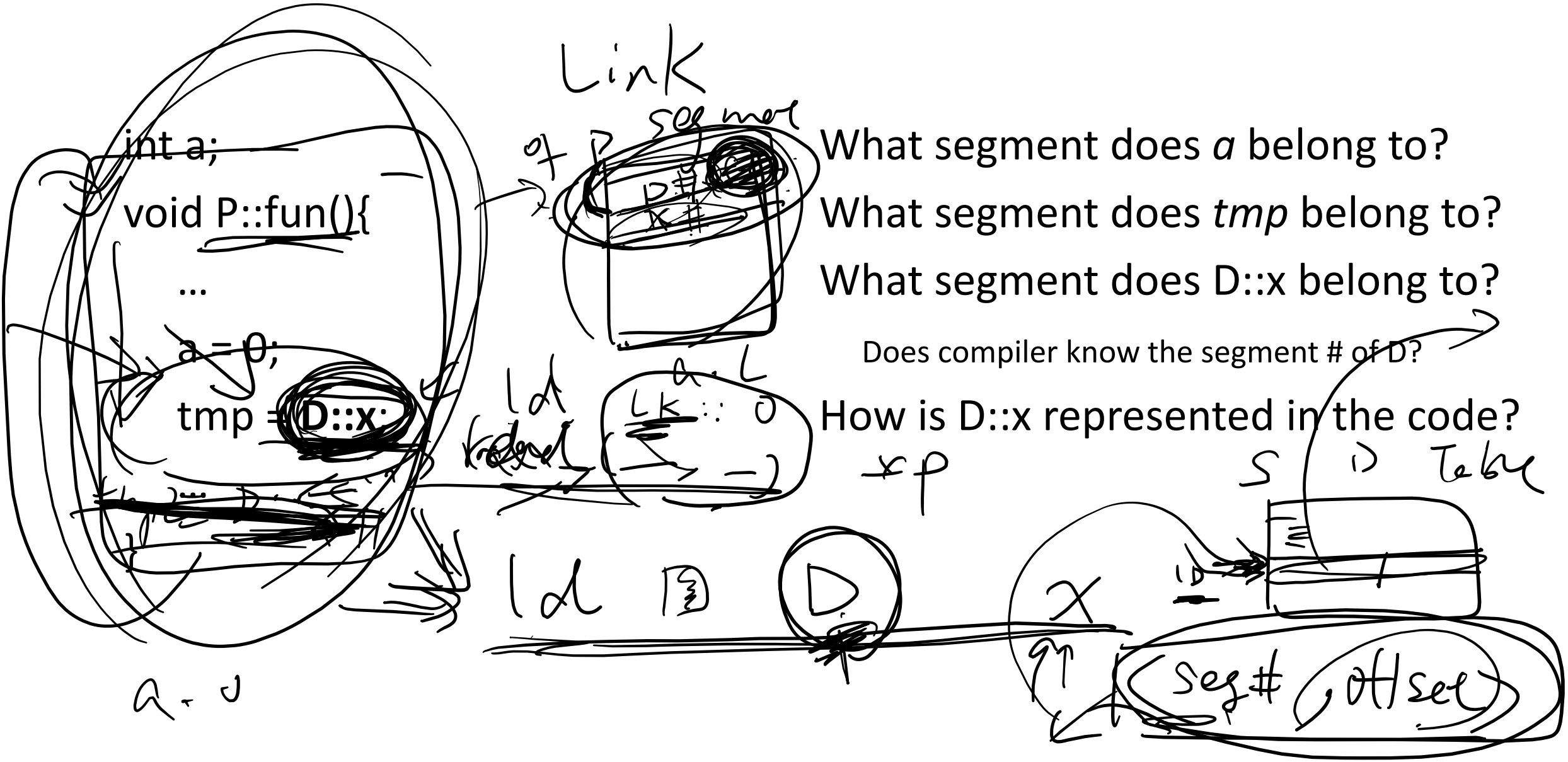
```
int a;
void P::fun(){
    int tmp;
    a = 0;
    tmp = D::x;
    …
}
```

What segment does *a* belong to?

What segment does *tmp* belong to?

What segment does D::x belong to?

Does compiler know the segment # of D?



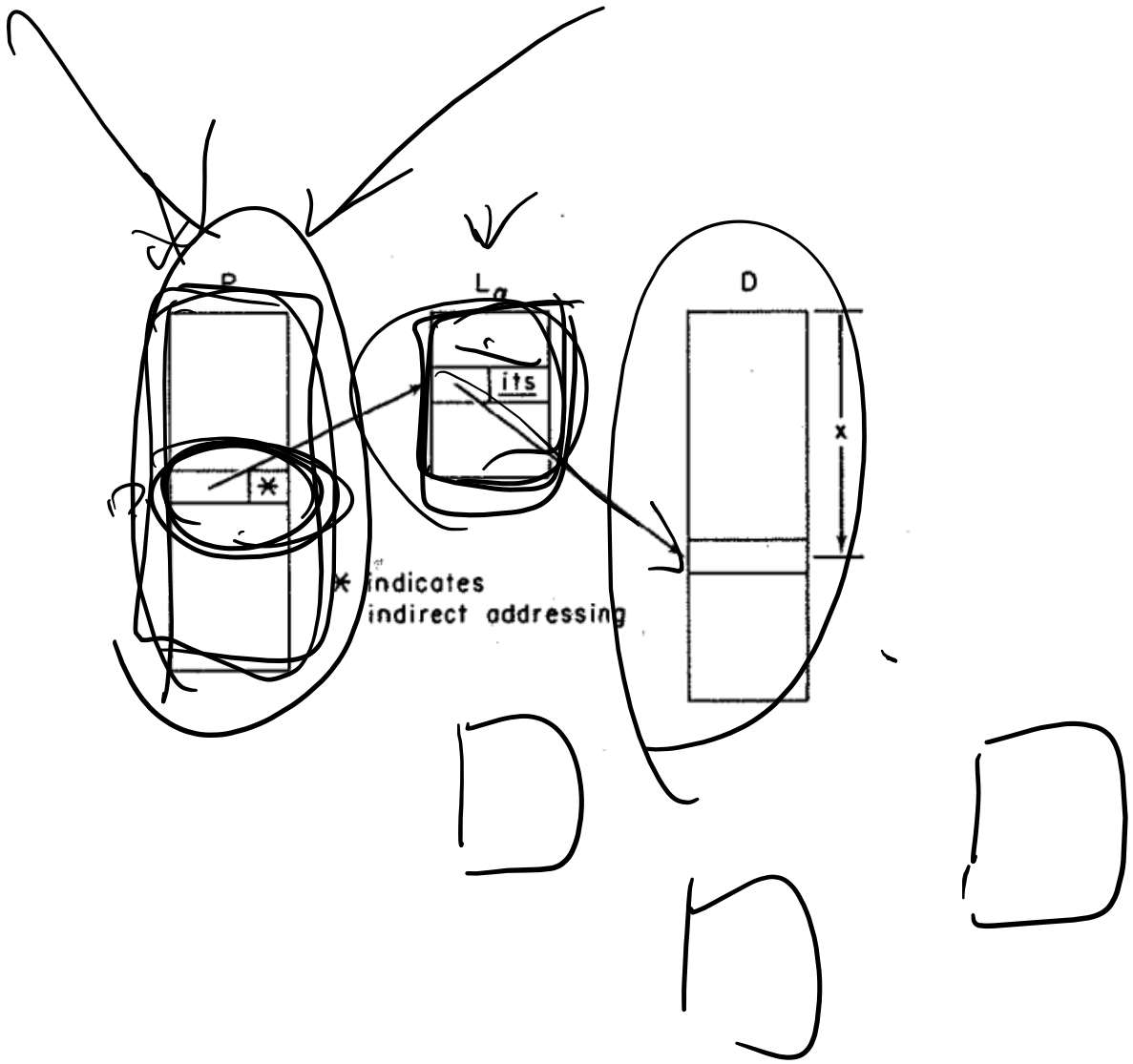
What segment does *a* belong to?

What segment does *tmp* belong to?

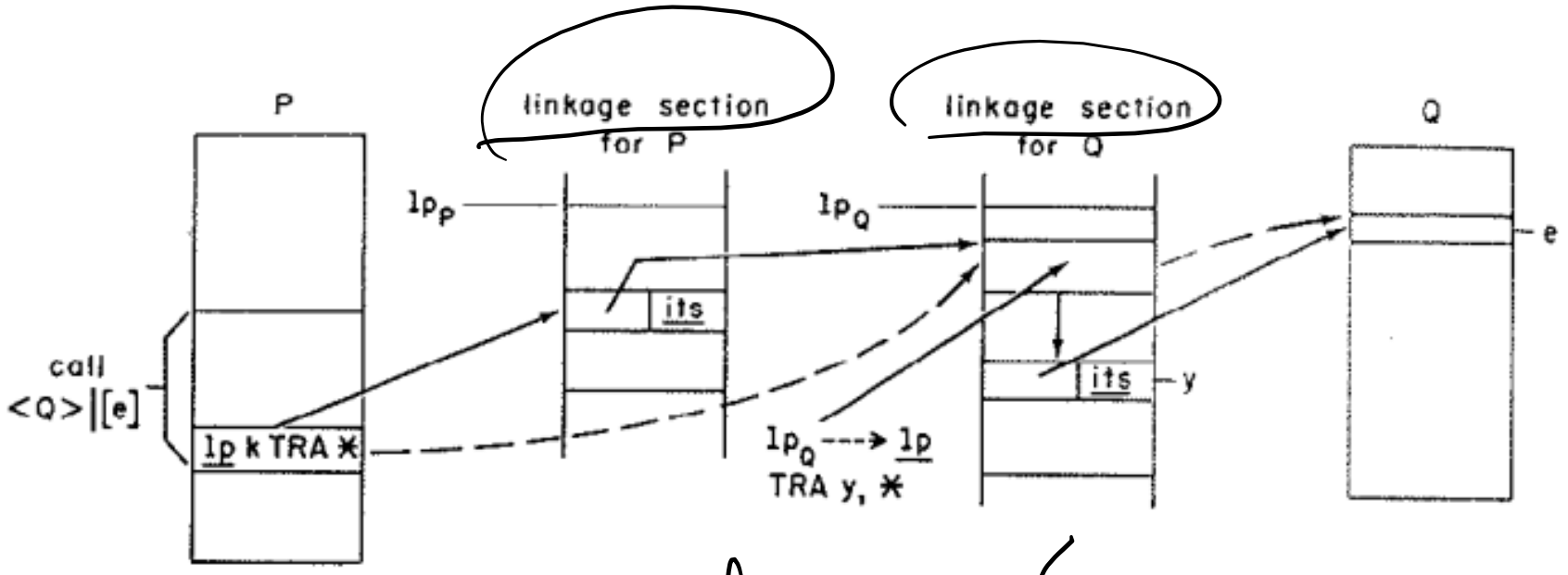
What segment does D::x belong to?

Does compiler know the segment # of D?

How is D::x represented in the code?


P
L_a
D
c
its
*
x
* indicates
indirect addressing

What are the challenges of code sharing?



P::func(){

Q::bar();

}

CBR