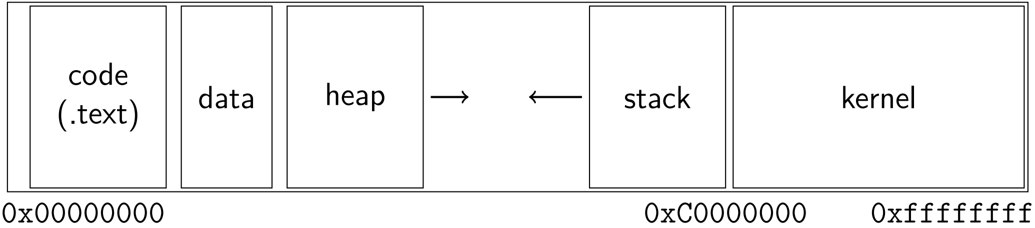
Memory

Johan Montelius

KTH

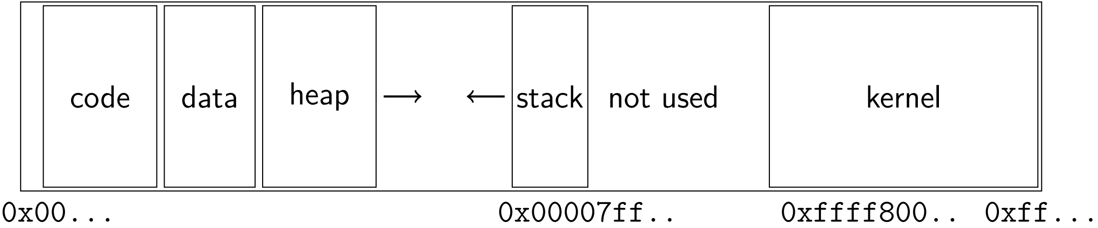
2020

The process



Memory layout for a 32-bit Linux process

64-bit Linux on a x86_64 architecture



Memory virtualization

Every process has an address space from zero to some maximal address.

A program contains instructions that of course rely on that code and data can be found at expected addresses.

We only have one physical memory.

IBM System 360

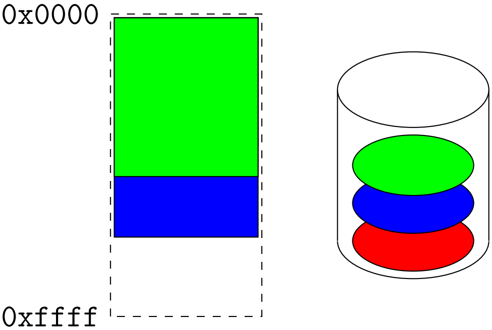
IBM System 360

- 1964, 8-64 Kbyte memory
- 12+12 bit address space
- batch operating system

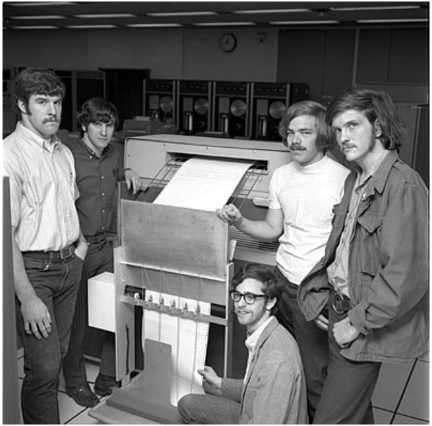
Chief architect: Gene Amdahl

when things were simple

Batch processing:



The Dartmouth Time-Sharing System



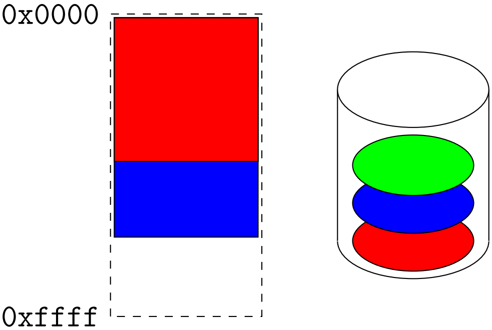
GE-235

- 1964
- 20-bit word
- 8 Kword address space

Arnold Spielberg was in the team that designed the GE-235

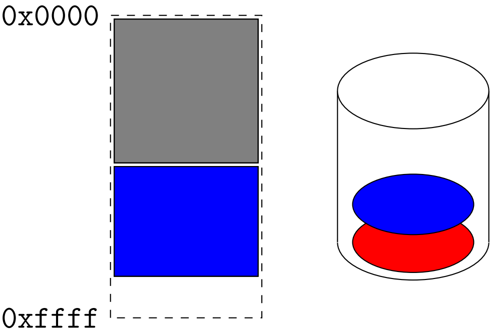
time-sharing

Time-sharing:



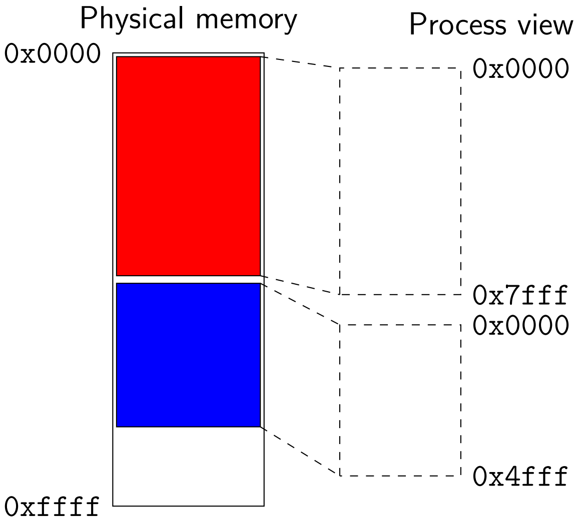
why not switch between two programs

If both programs will fit in memory:



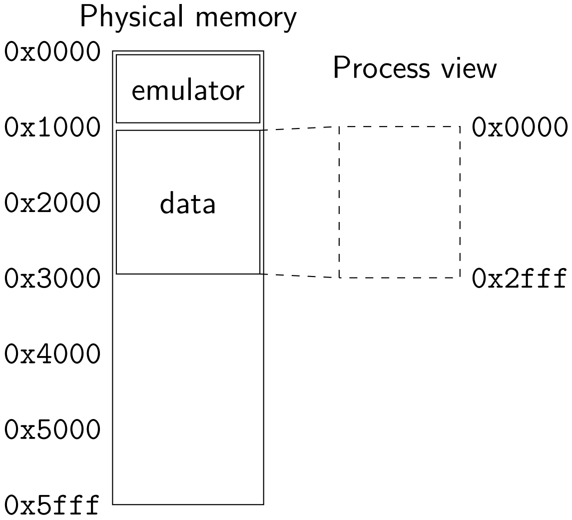
What is the problem?

Virtual memory



- Transparent: processes should be unaware of virtualization.
- Protection: processes should not be able to interfere with each other.
- Efficiency: execution should be as close to real execution as possible.

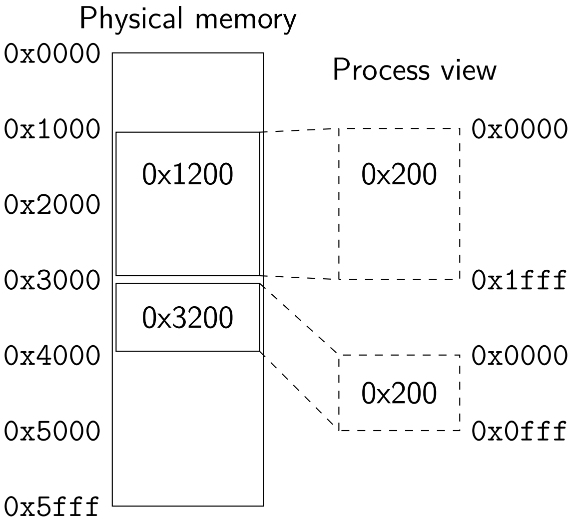
Emulator - simple but slow



Let the operating system run an *emulator* that interprets the operations of the process and changes the memory addresses as needed.

This is similar to how the JVM works

Static relocation - ehh, static

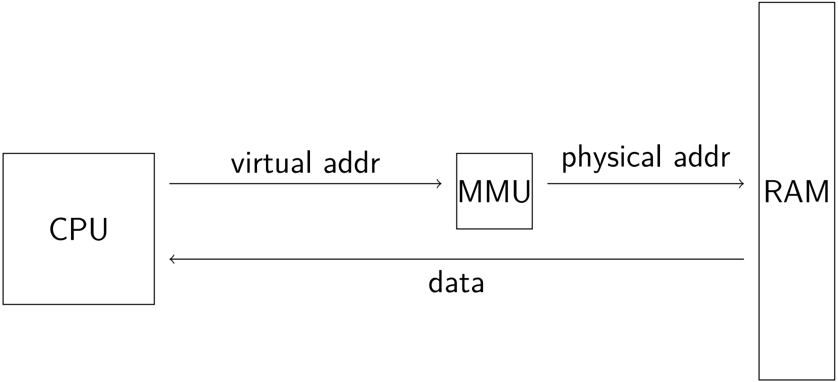


When a program is loaded, all references to memory locations are changed so that they correspond to the actual location in RAM where the program is loaded.

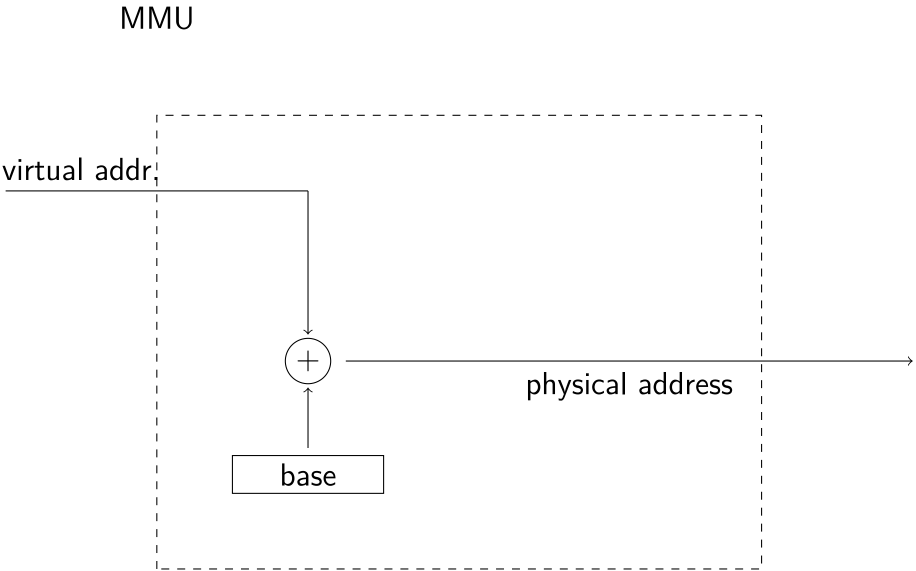
How do we know we have changed all addresses?

Dynamic relocation

Change every memory reference, on the fly, to a region in memory allocated for the process.

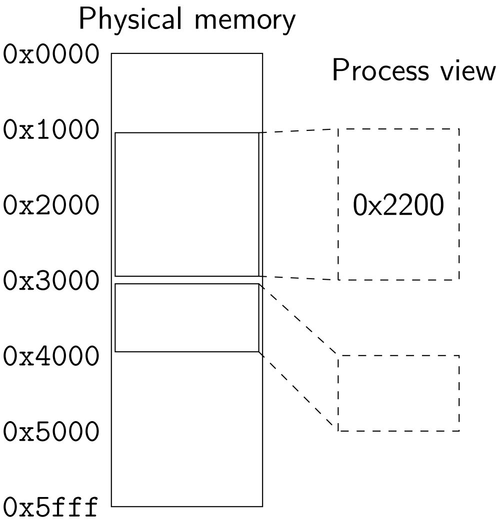


Base register



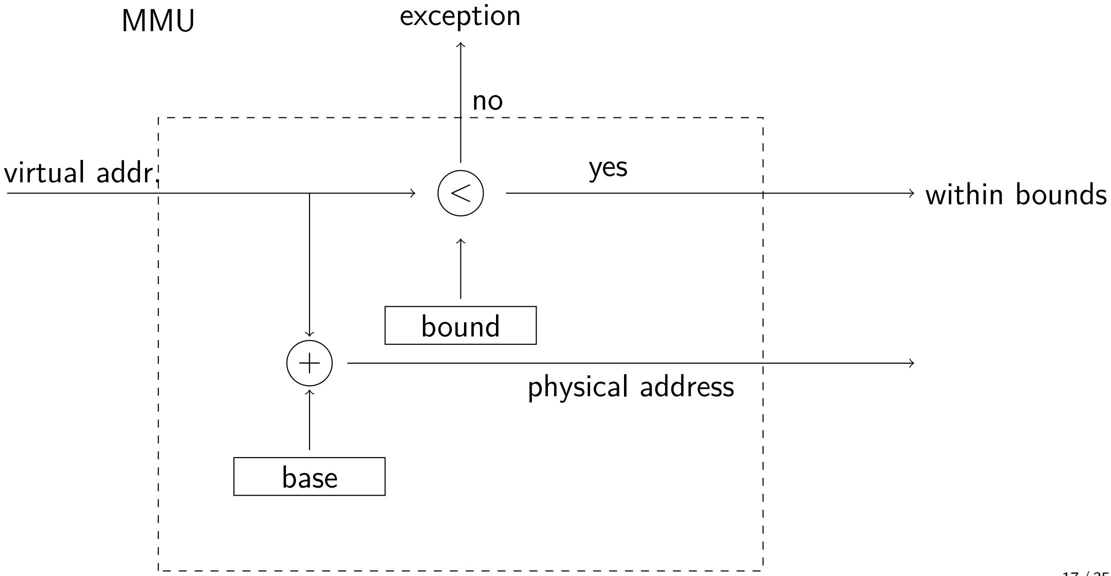
Base problem

- Who is allowed to change the base register?
- How do we prevent one process from overwriting another process?



Can we prevent this at compile or load time?

Base and bound



Base and bound

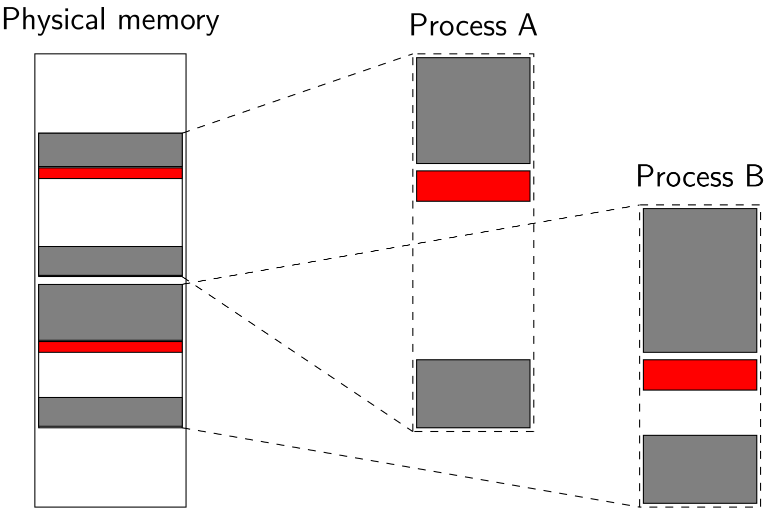
Pros:

- Transparent to a process.
- Simple to implement.
- Easy to change process.

Cons:

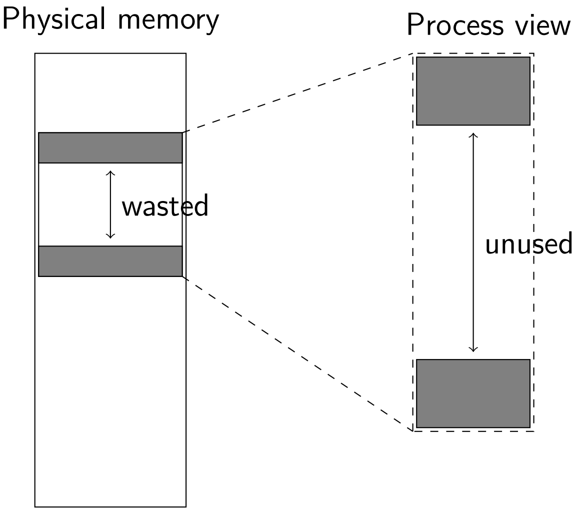
- How do we share data?
- Wasted memory.

shared read-only segments

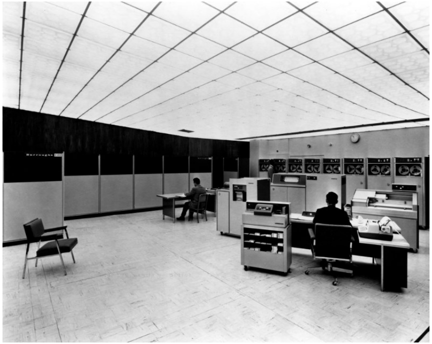


How do we write code that can be shared?

Internal fragmentation



Burroughs B5000



- 1961
- Designed for high-level languages: ALGOL-60
- Memory access through a set of segment *descriptors* i.e. the view of a process is not a consecutive memory rather a set of individual memory segments.

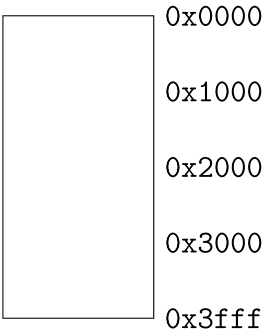
Donald Knuth was part of the design team.

ALGOL 60

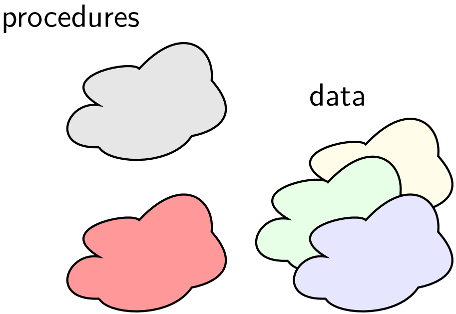
```
procedure Absmax(a) Size:(n, m) Result:(y) Subscripts:(i, k);
    value n, m; array a; integer n, m, i, k; real y;

comment The absolute greatest element of the matrix a ...

begin
    integer p, q;
    y := 0; i := k := 1;
    for p := 1 step 1 until n do
        for q := 1 step 1 until m do
            if abs(a[p, q]) > y then
                begin y := abs(a[p, q]);
                    i := p; k := q
                end
end Absmax
```

Process view

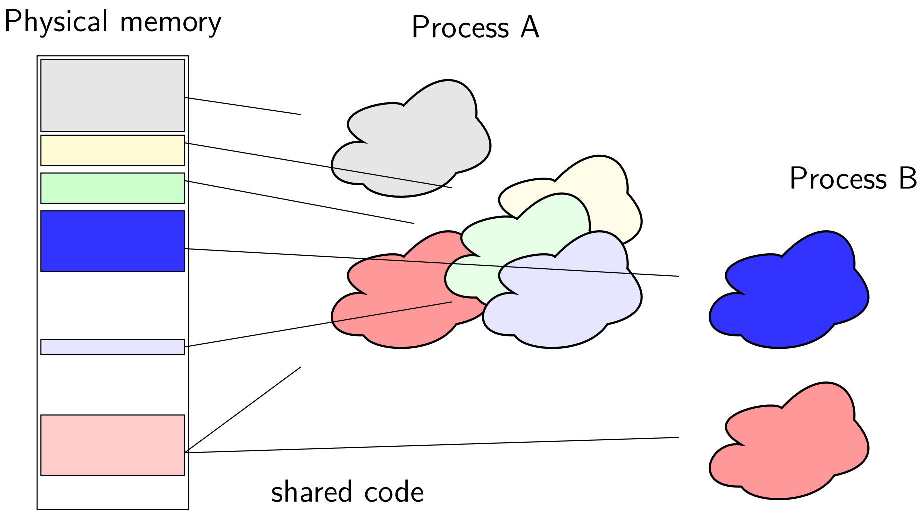
The view of the assembler programmer.



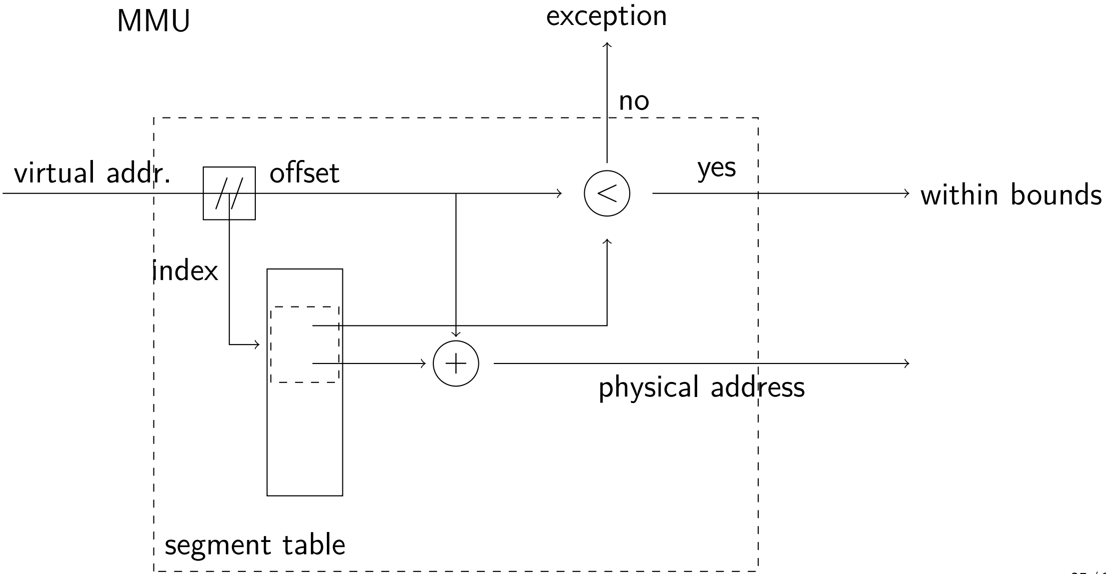
The view of the ALGOL programmer.



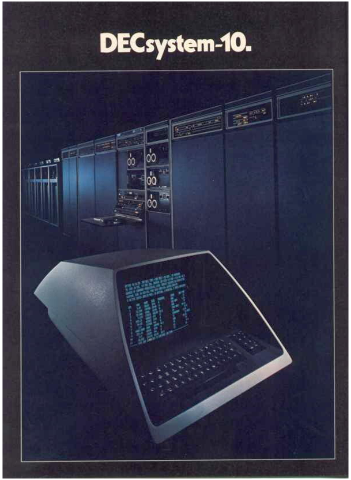
Segmented architecture



Segmented MMU



DECsystem10



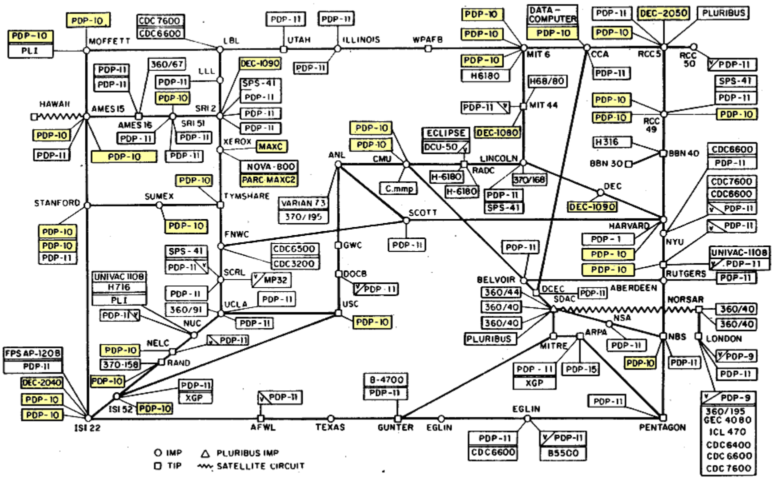
PDP-10

- 1966, 1 MHz
- 36 bit words
- 16 bit process address space (64Kword)
- 18 bit physical address (256 Kword)
- base and bound

The PDP10 had two segments per process, one read only code segment and one read/write for data.

ARPANET 1977

ARPANET LOGICAL MAP, MARCH 1977

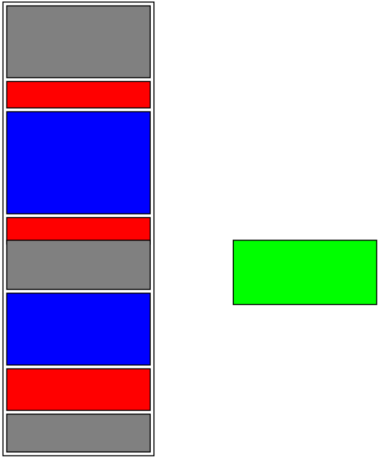


(PLEASE NOTE THAT WHILE THIS MAP SHOWS THE HOST POPULATION OF THE NETWORK ACCORDING TO THE BEST INFORMATION OBTAINABLE, NO CLAIM CAN BE MADE FOR ITS ACCURACY)

NAMES SHOWN ARE IMP NAMES, NOT (NECESSARILY) HOST NAMES

Segmentation: the solution - **not**

- Segments have variable size.
- Reclaiming segments will cause holes (external fragmentation).
- Compaction needed.



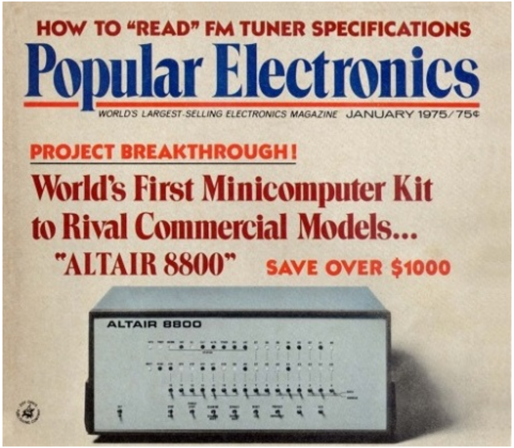
Is it possible to do compaction?

large grain vs fine grain segments

Using few large segments is easier to implement.

Using many small segments would allow the compiler and operating system to do a better job.

The Altair 8800



Intel 8080

- 1972
- 2 MHz
- 16 bit address space (64 Kbyte)

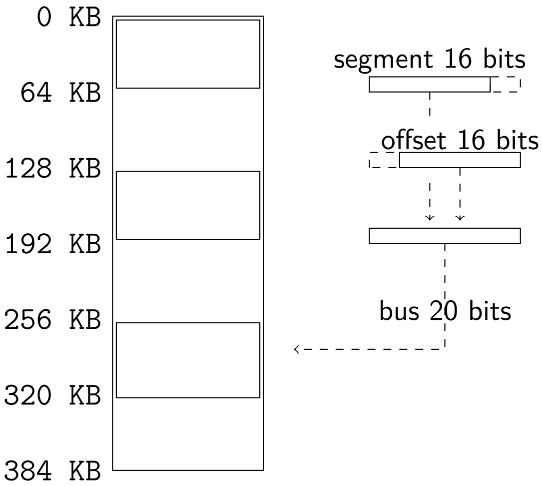
Altair 8800 would have 4 or 8 Kbytes of memory.

The workhorse: 8086

Intel 8086

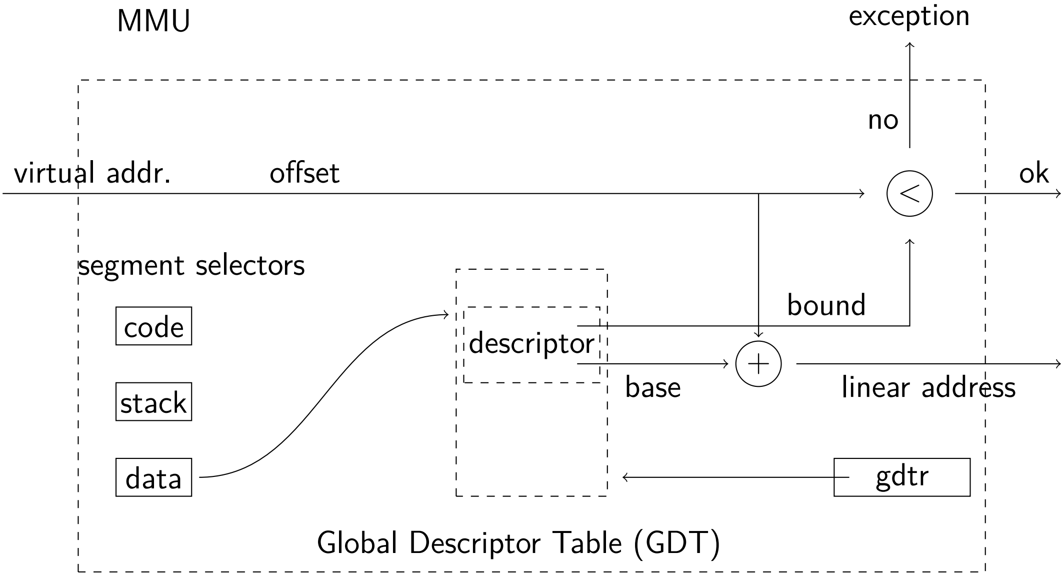
- 1978, 5 MHz
- 16 bit address space (64 Kbyte)
- 20 bit memory bus (1 Mbyte)
- no protection of segments
- segments for: code, data, stack, extra

Segment addressing in 8086 - real mode



- Segment register chosen based on instruction: *code segment*, *stack segment*, *data segment* (and the *extra segment*.
- The segment architecture available still today in *real mode* i.e. the 16-bit mode that the CPU is initally in.

Segment addressing in 80386 - protected mode



Linux and segmentation

- The segments descriptors of code, data and stack all have base address set to `0x0` and limit to `0xffffffff` i.e. they all referre to the same 4 Gibyte linear address space.
- In x86_64 long mode (64 bit mode) Intel removed some support for segments and enforce that these segments are set to `0x0` and `0xff..ff`.
- Segmentation is still used to refere to memory that belongs to a *specific core* or to *thread specific memory*.

Summary

Virtual address space: provide a process with a view of a private address space.

- Transparent: processes should be unaware of virtualization.
- Protection: processes should not be able to interfere with each other.
- Efficiency: execution should be as close to real execution as possible.

- Emulator - two slow.
- Static relocation - not flexible.
- Dynamic relocation:
  - base and bound - simple to implement
  - segmentation - more flexible
  - problems: fragmentation, sharing of code

Cliffhanger - paging, the solution.