

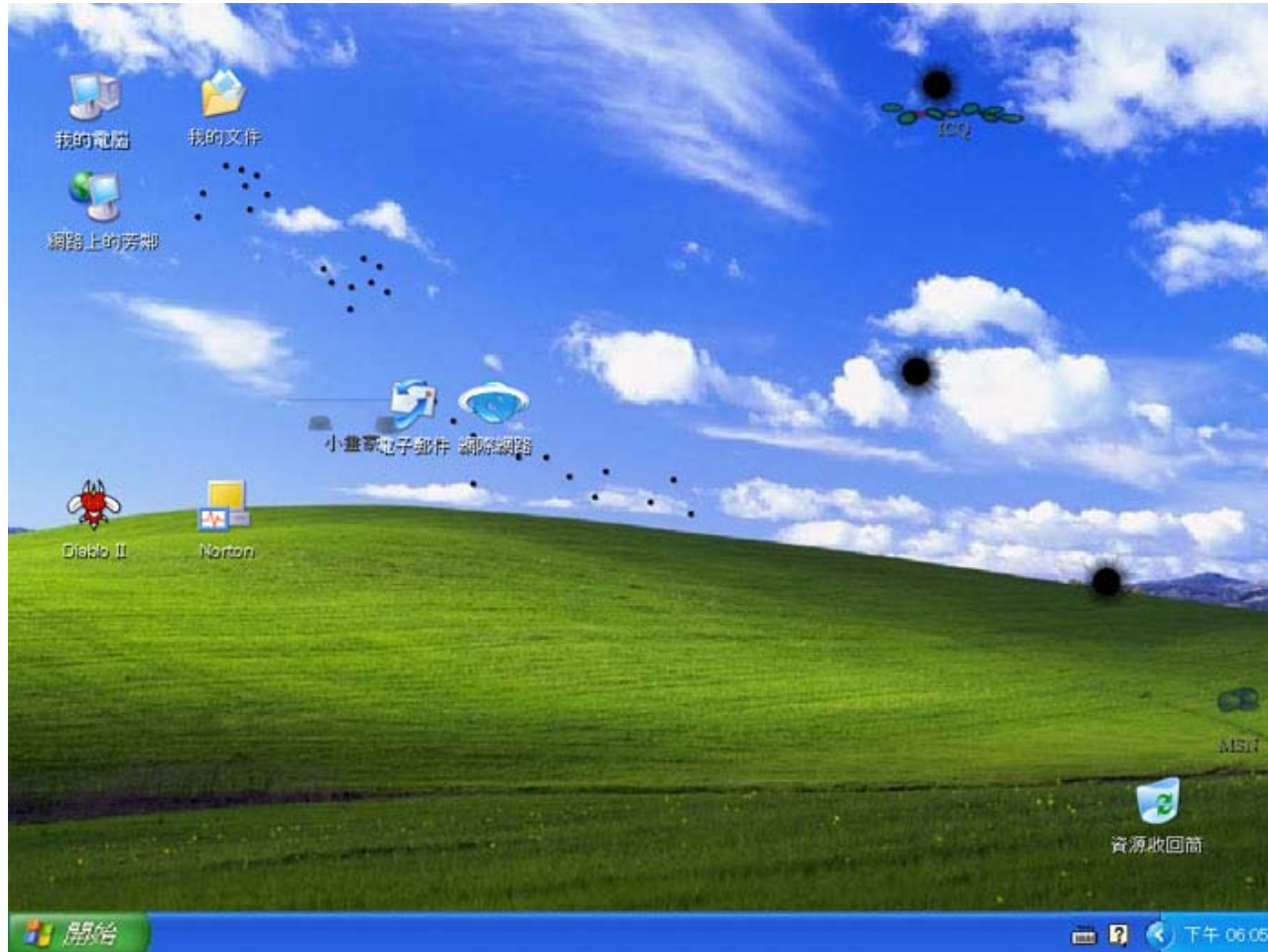
Protection via Separation

Lujo Bauer

18-732

Fall 2010

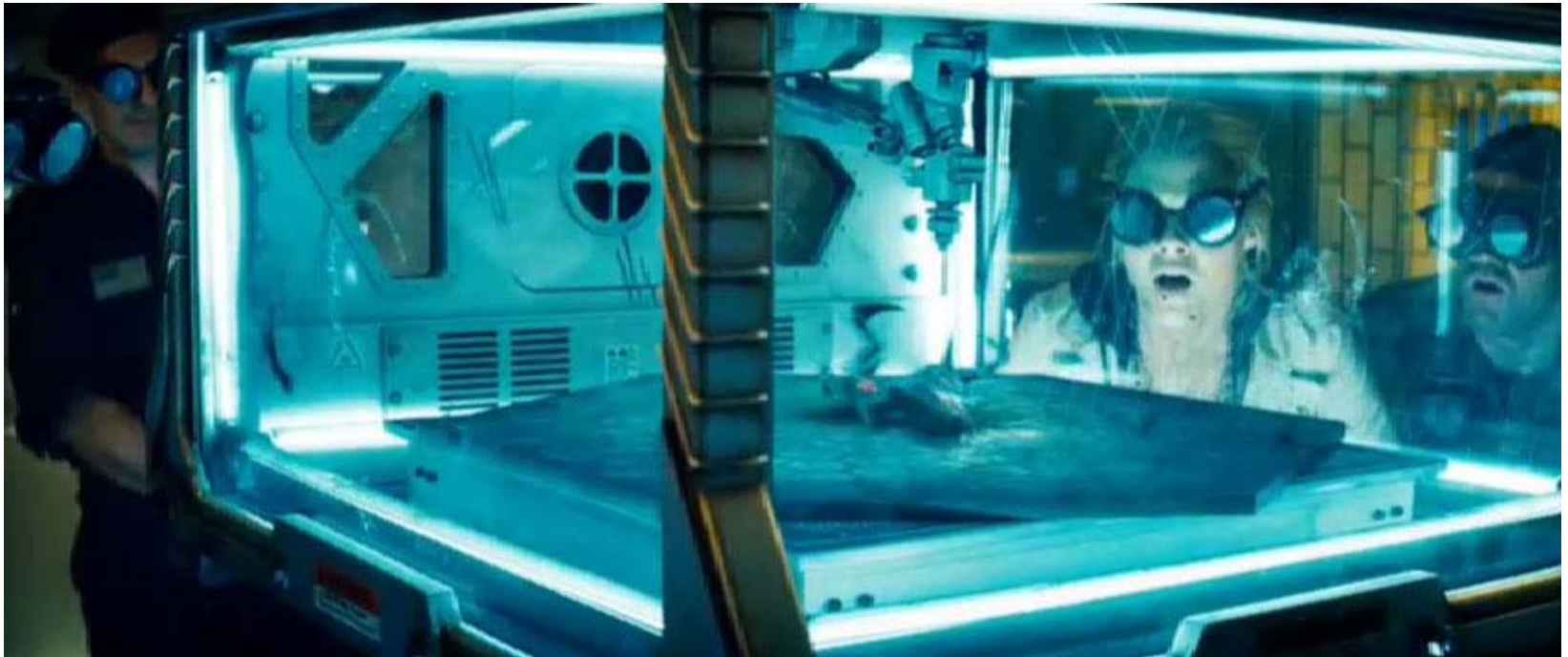
When There Is No Protection...



Resources That Need Protection

- Multiprogramming caused/increased the need for protection
- Resources that need protection
 - Memory
 - Shared IO devices
 - Disks (simultaneously sharable)
 - Printers (serially sharable)
 - Shared programs and modules
 - Networks
 - Shared data

Separation



Methods of Separation

- Physical
 - Multiple printers, disks, ...
- Temporal
 - Insecure before 12pm; secure after 12pm
- Logical
 - OS/environment provides isolation
- Cryptographic
 - Data and computation concealed cryptographically

Degrees of Separation

- None
 - All resources are fully shared
- Complete isolation
 - Programs are completely oblivious to each other
- Binary sharing
 - Resources are either public or private
- Limited sharing
 - Fine-grained control
 - Authorization checked on a per-access basis
 - E.g., ACLs, capabilities
- Usage control
 - Control not just access to resources, but also how they're used

Mechanisms for Separation

- Hardware
 - Memory protection
- Software
 - Sandboxing
- Hardware + software
 - Virtual machines

Granularity of Separation

- What are the objects that are being separated from each other?
- Programs
- Sets of programs
- Parts of a program
- OSes

Separation

- In what dimension should objects be separated?
- To what degree should objects be kept separate?
 - And how should crossing the separation boundary be mediated?
- What are the objects that will be kept apart?
- What mechanisms will provide separation?

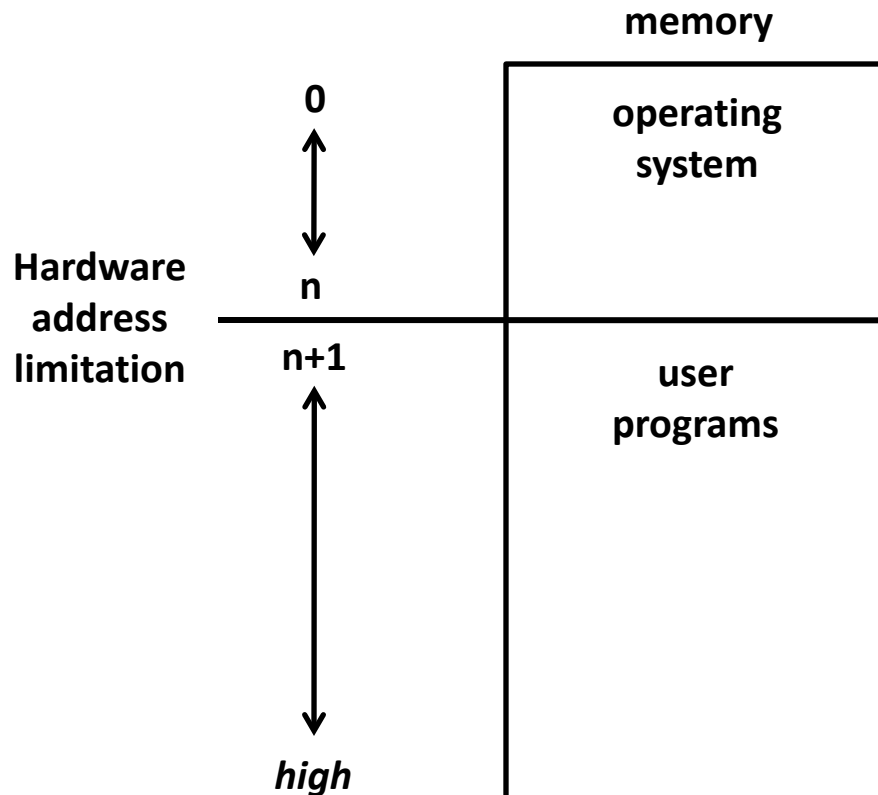
- Big design space
 - We will learn about various points in this space

Memory Protection

- An oldie but goodie
- Logical separation, between programs, using hardware (and software)

Memory Protection – Fence

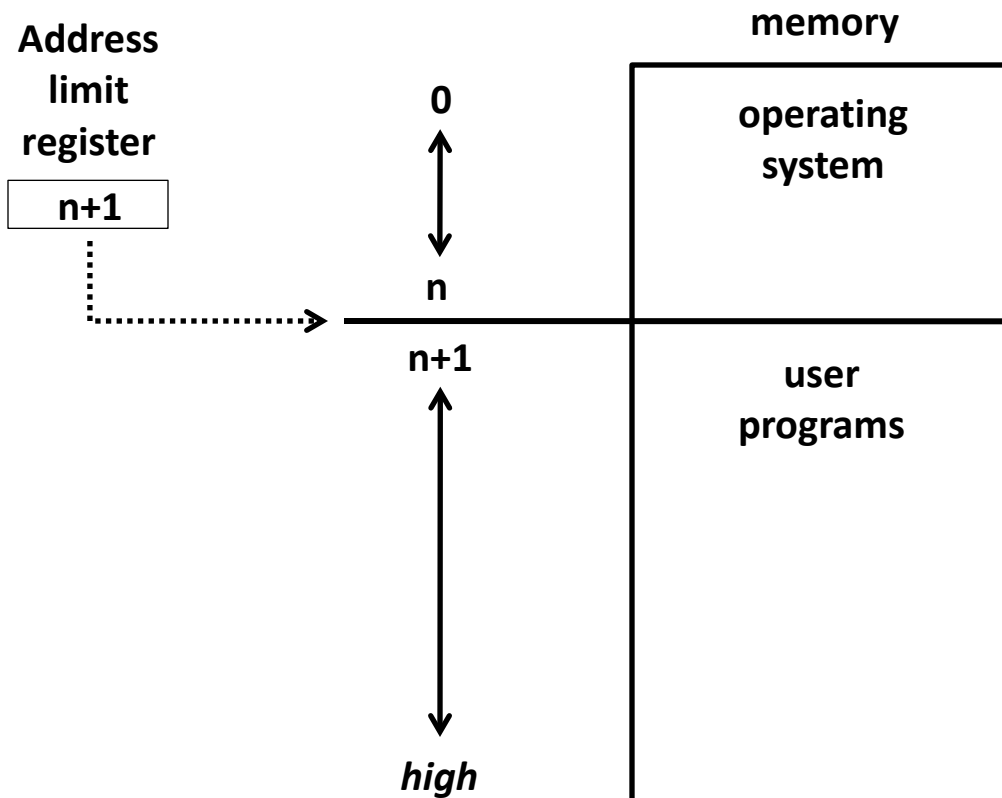
- Boundary between user programs and operating system



- Fence fixed for all programs and OSeS

Memory Protection – Fence

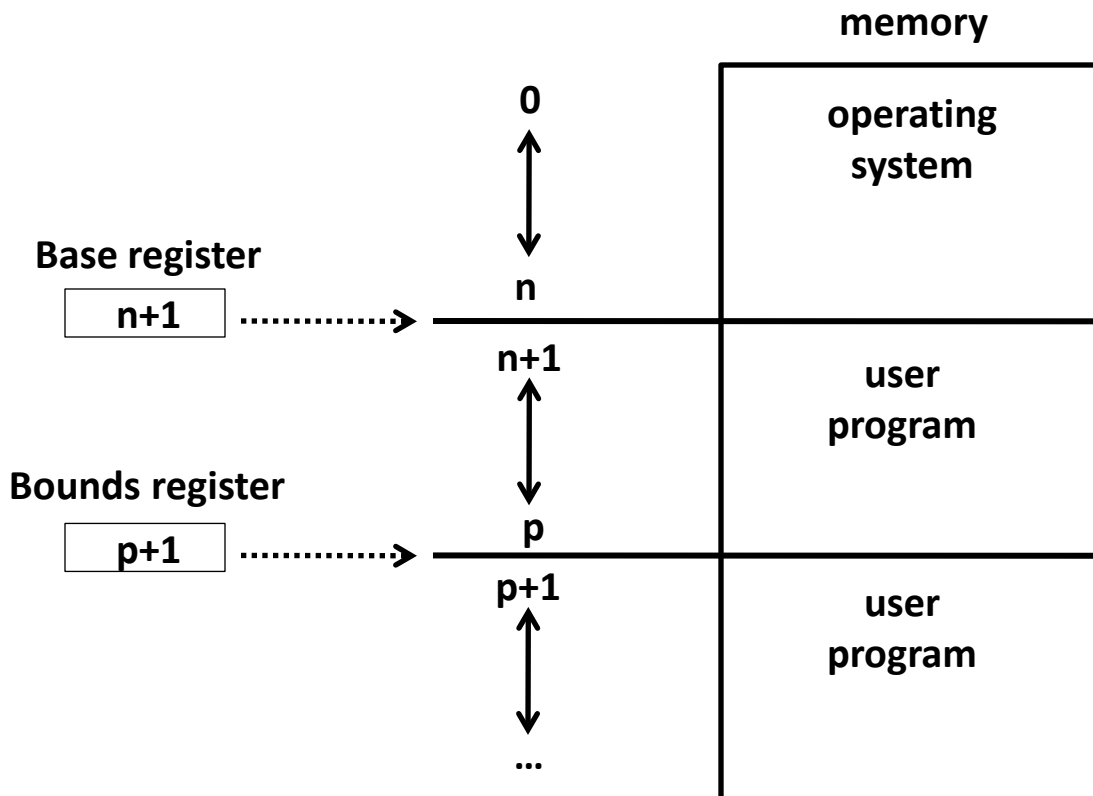
- A slight refinement



- **One-way protection**
- **No protection within user program space**

Memory Protection – Base/Bounds Registers

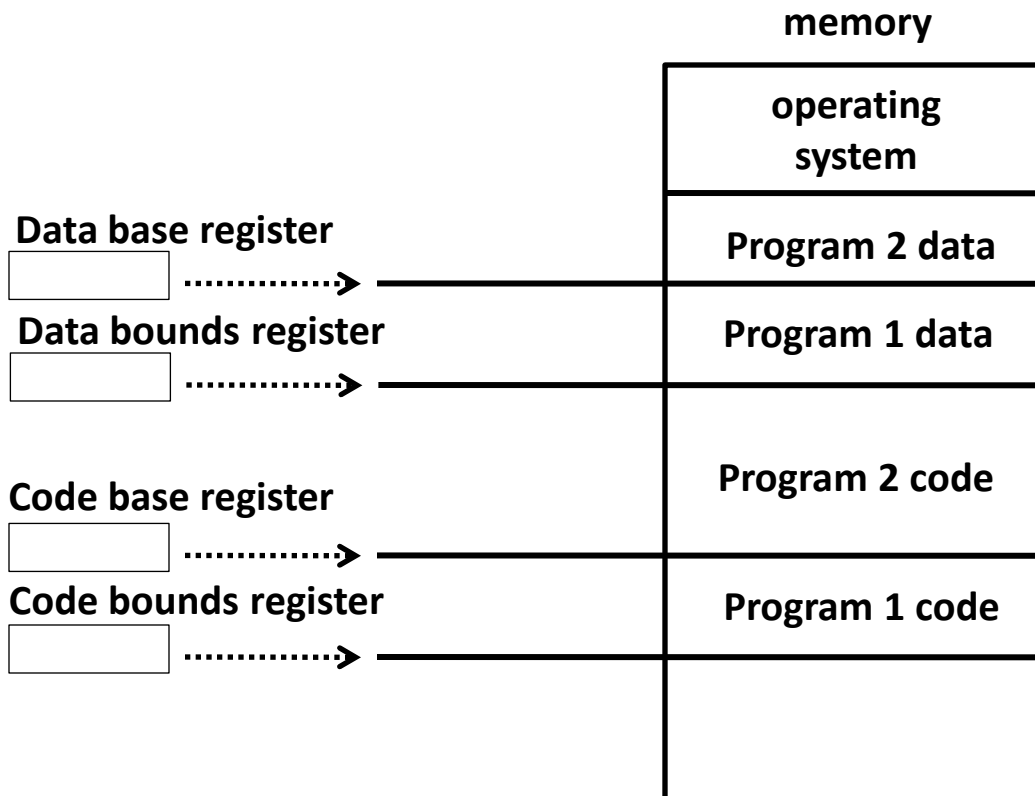
- If one fence is good, two are better



- Programs protected from each other
- Context switch exchanges registers
- Programs can still hurt themselves

Memory Protection – Base/Bounds Registers

- If two good, four are better



Memory Protection – Base/Bounds Registers

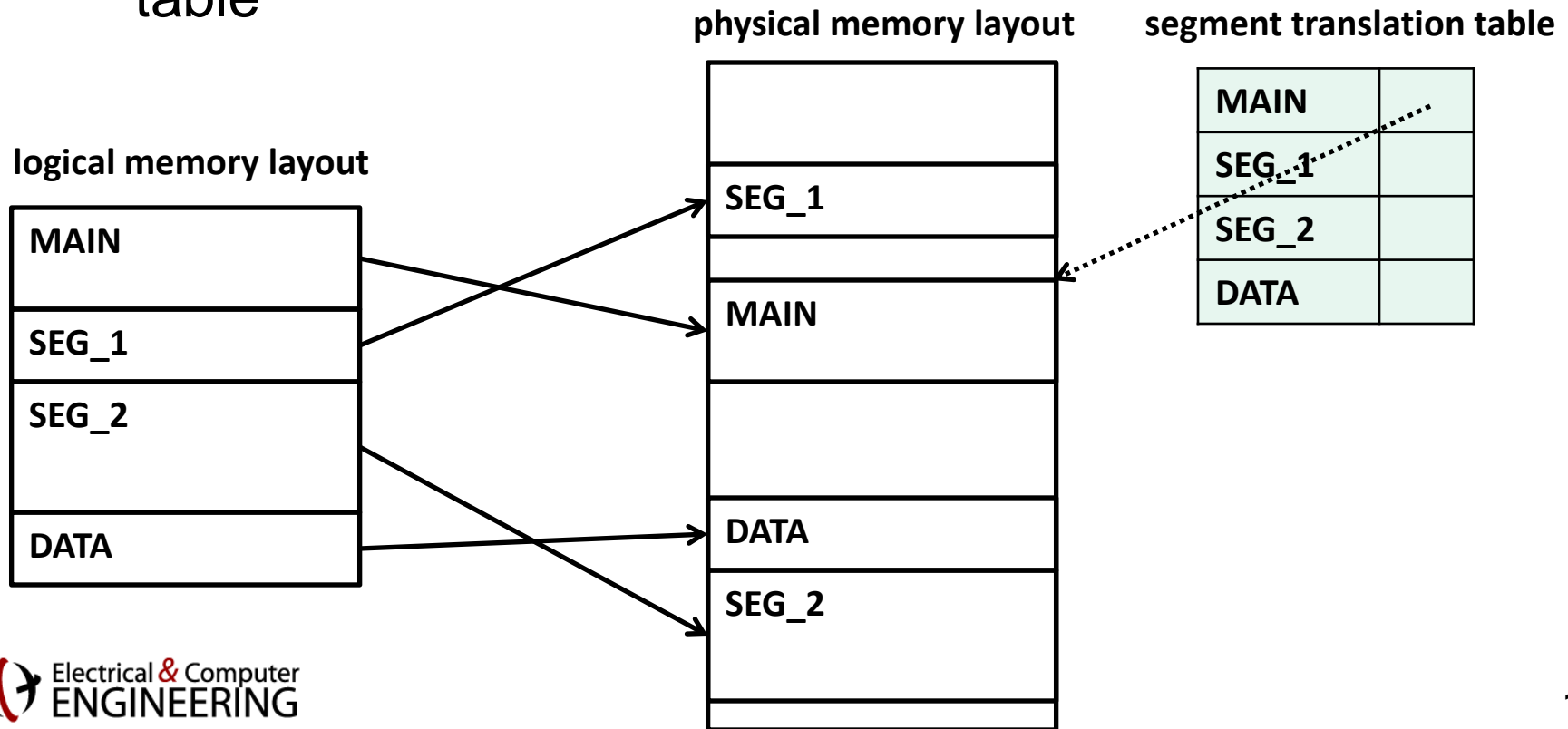
- If four are good, ... ?
- E.g., separate data into read-only and read-write
 - Instructions have to include pointer to data space

Memory Protection – Tagging

- If four are good, ... ?
- E.g., separate data into read-only and read-write
 - Instructions have to include pointer to data space
- In the limit, each word of memory can be in own domain
 - E.g., labeled as read-only, read-write, or execute

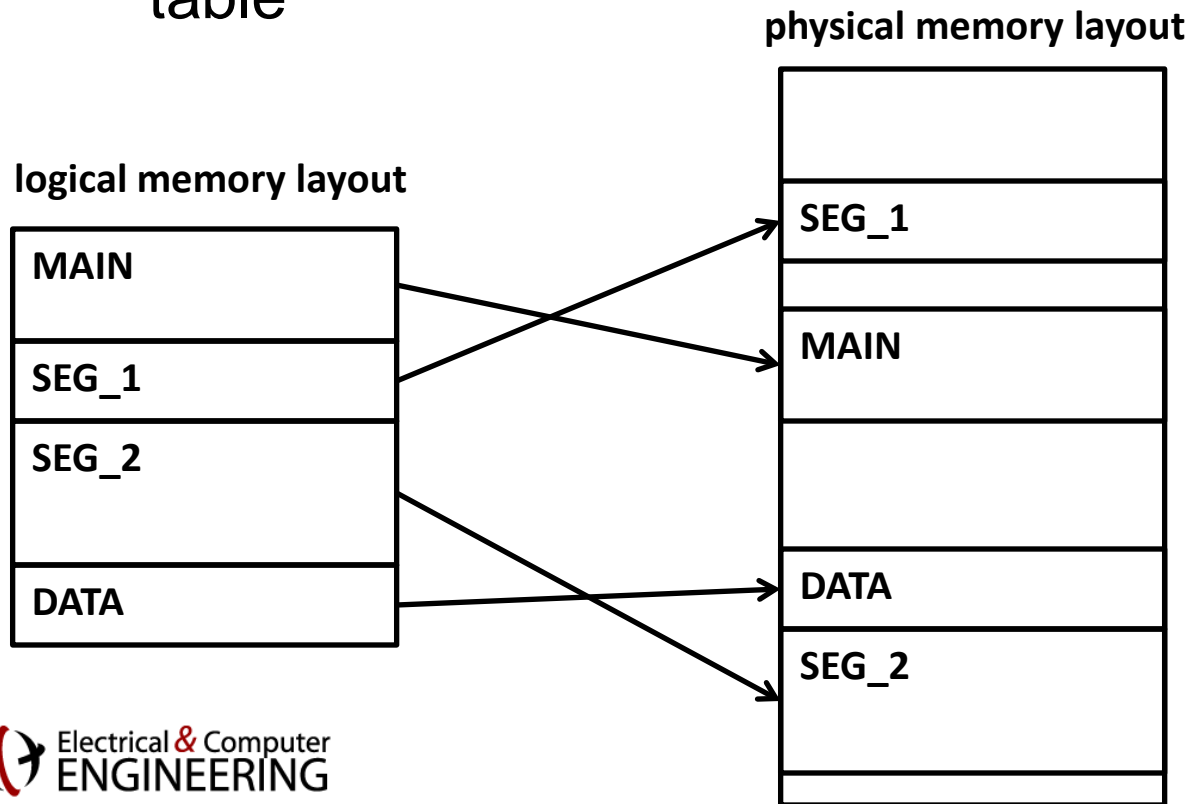
Memory Protection – Segmentation

- Program divided into many logical, named pieces
- Addresses have the form $\langle name, offset \rangle$
- OS maintains translation table



Memory Protection – Segmentation

- Program divided into many logical, named pieces
- Addresses have the form $\langle name, offset \rangle$
- OS maintains translation table

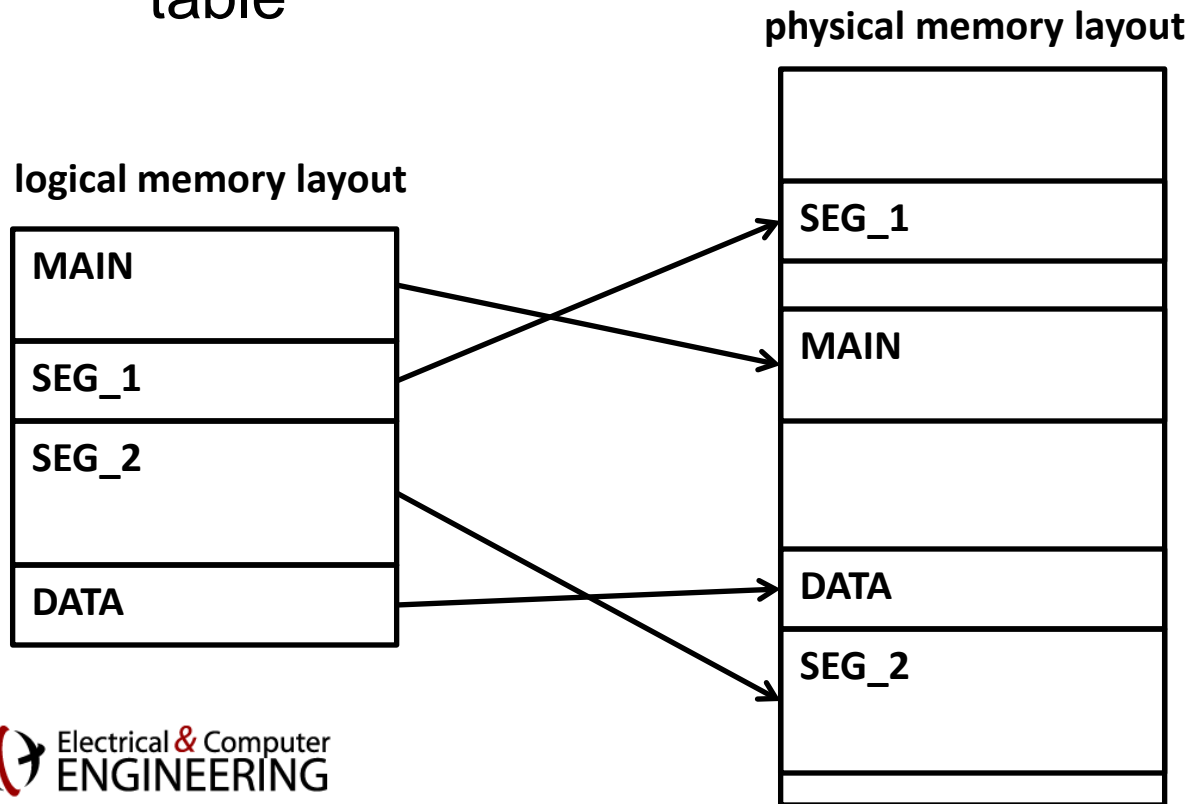


Advantages

- All references by name – OS mediates each access
- Segments can be moved around or swapped to disk
- Segments can be shared

Memory Protection – Segmentation

- Program divided into many logical, named pieces
- Addresses have the form $\langle name, offset \rangle$
- OS maintains translation table



Disadvantages

- Fragmentation
- Inefficient bounds checking
 - ▼ Offset may be beyond end of segment

Memory Protection – Paging

- Program divided into equal-sized *pages*
- Addresses have the form $\langle page, offset \rangle$

- Fragmentation isn't a problem
 - All pages (segments) have the same size
- Addressing ensures offset is within page
 - E.g., 1024-byte pages use 10 bytes for the offset
- Logical segmentation isn't preserved
 - Pages don't correspond to logical program segments

Memory Protection – Segmentation & Paging

- Combine efficiency of paging with protection of segmentation
- First divide into segments, then each segment into pages
- Intel x86 architecture implements paged segmentation ...
- ... but no widely used OS takes advantage

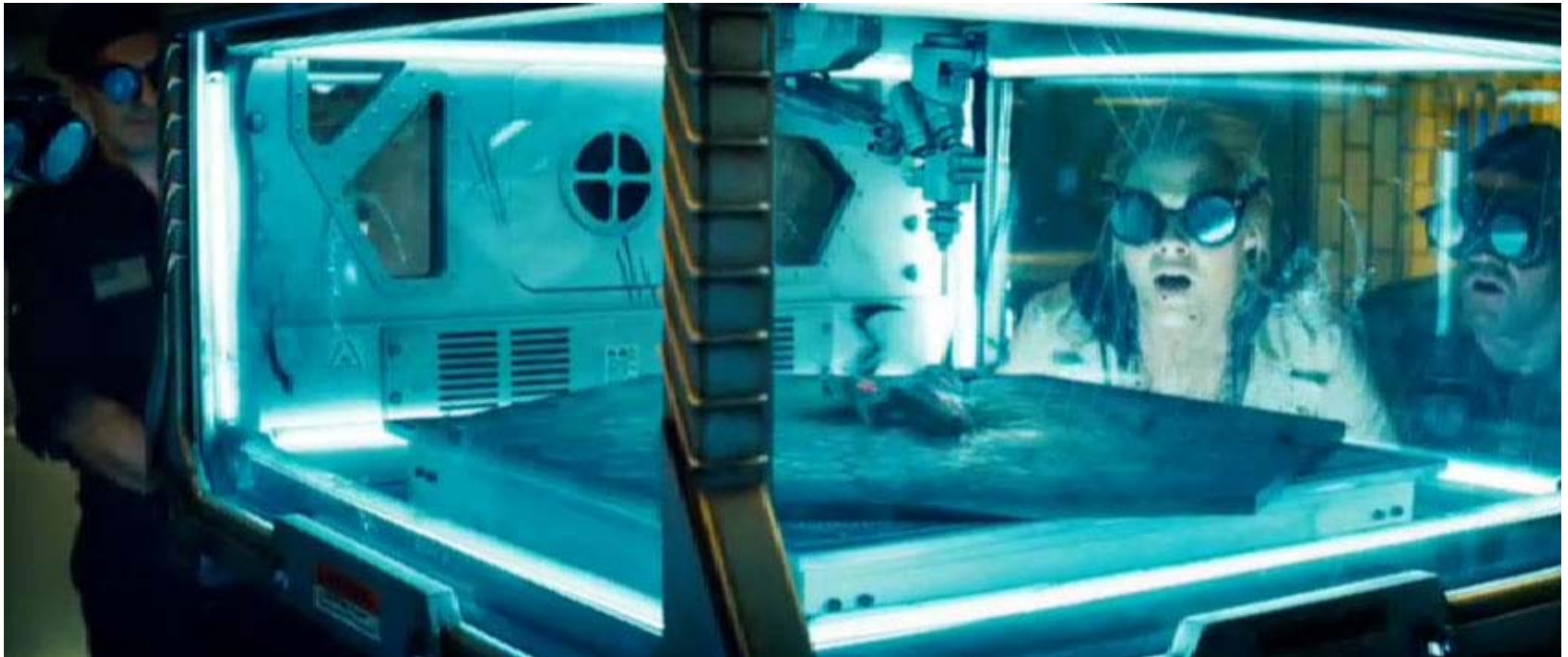
Memory Protection in Software

- Hardware memory protection can be expensive
 - Context switches are expensive
 - Each cross-module call requires a context switch
- Software memory protection
[Wahbe, Lucco, Anderson, Graham 1993]
 - Allocate a contiguous memory space to an untrusted program
 - Modify untrusted program so that every memory access has to be to memory within that space

Memory Protection in Software

- Code divided into code and data segments
 - Jump targets restricted to code segment
 - Data addresses restricted to data segment
- Code and data segment addresses stored in dedicated registers
- Before executing any instruction that references memory
 - Compare segment of memory reference to stored segment addresses
 - Stop execution if addresses aren't equal
- Applications
 - User-level file systems, user-programmable IO systems, etc.

Separation



Administrative

- Be nice to the TA

Sources

- Chp. 4. Pfleger & Pfleger. *Security in Computing*. 4th ed. Prentice Hall, 2006.
- R. Wahbe, S. Lucco, T. E. Anderson, S. L. Graham. Efficient software-based fault isolation. In *Proceedings of the 14th ACM symposium on Operating systems principles*, 1994.