Project VM

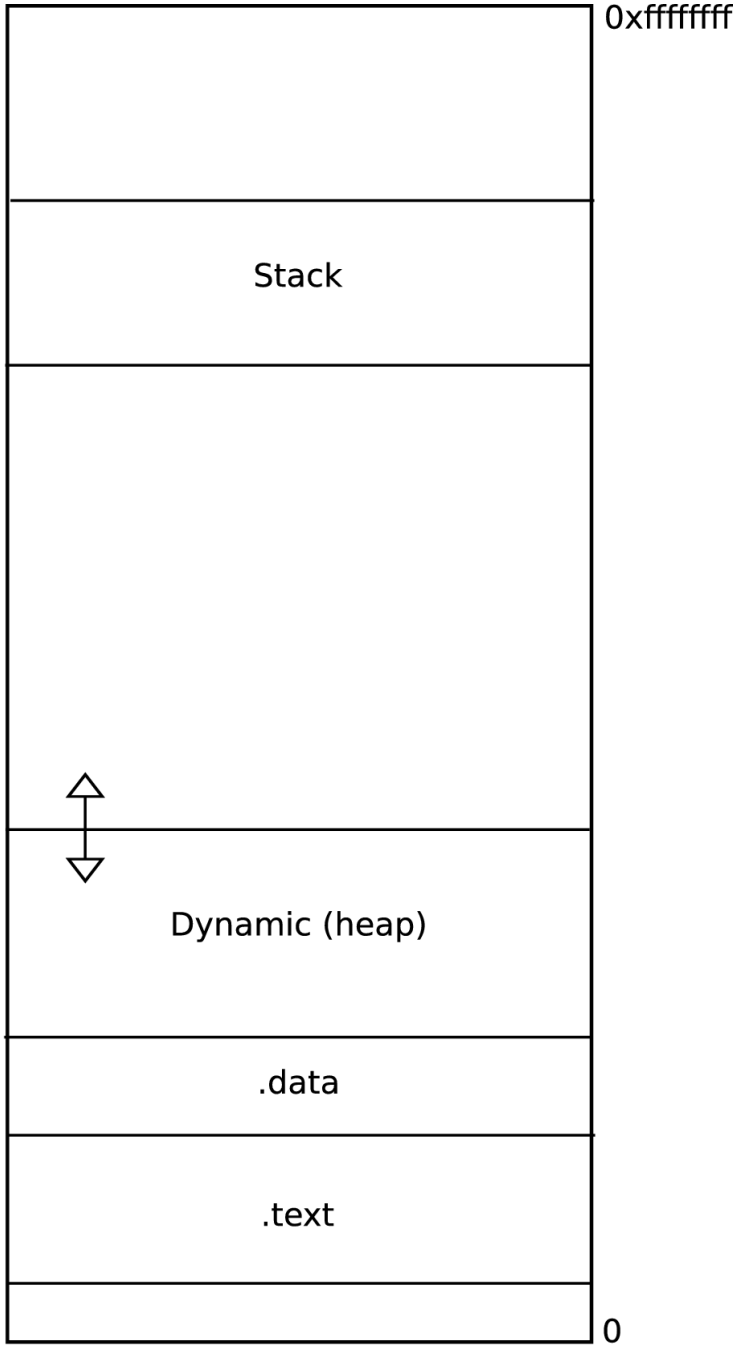
(or: Project Address)

Help Session

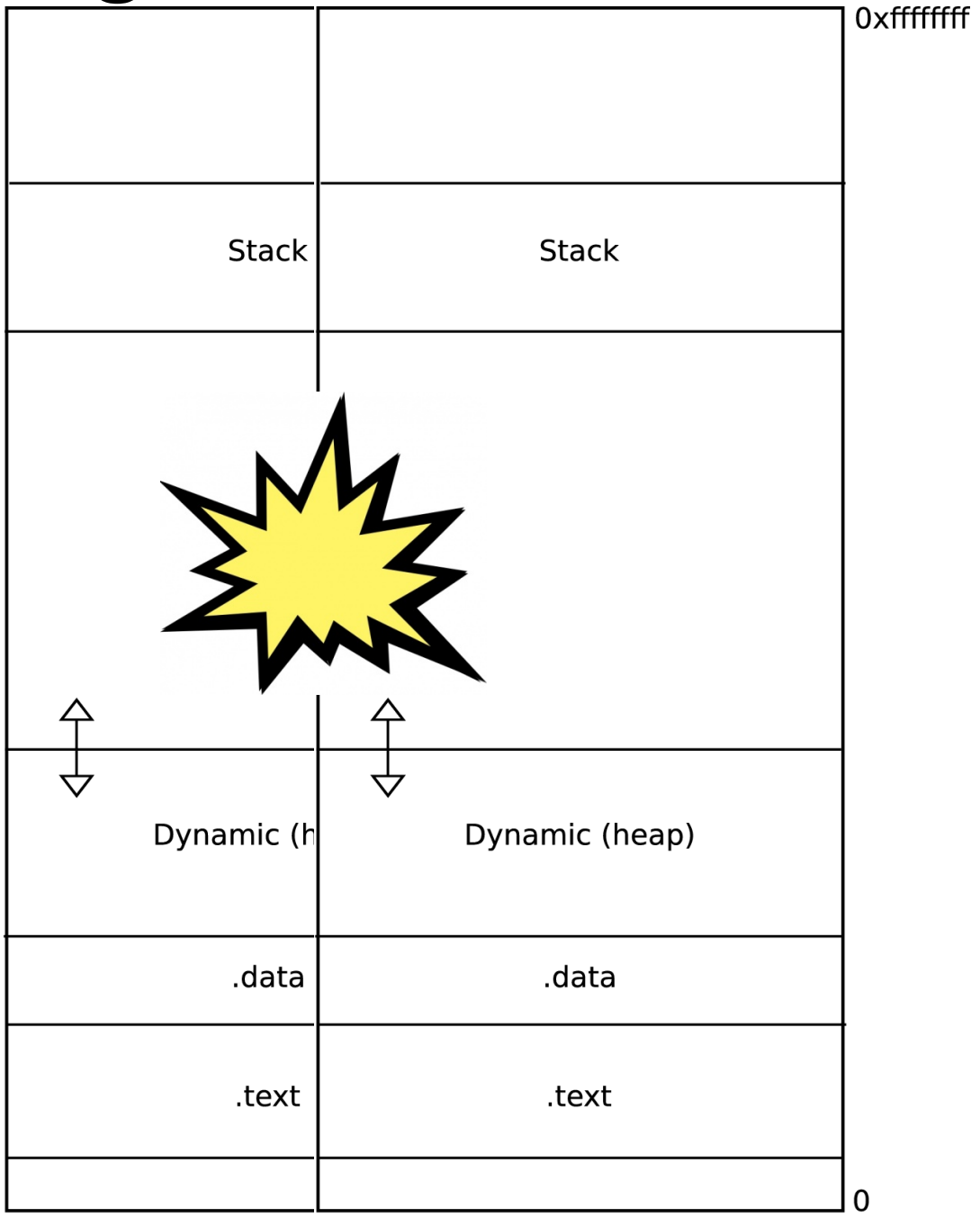
Motivation

Programs Like Predictability

Typical Address Space

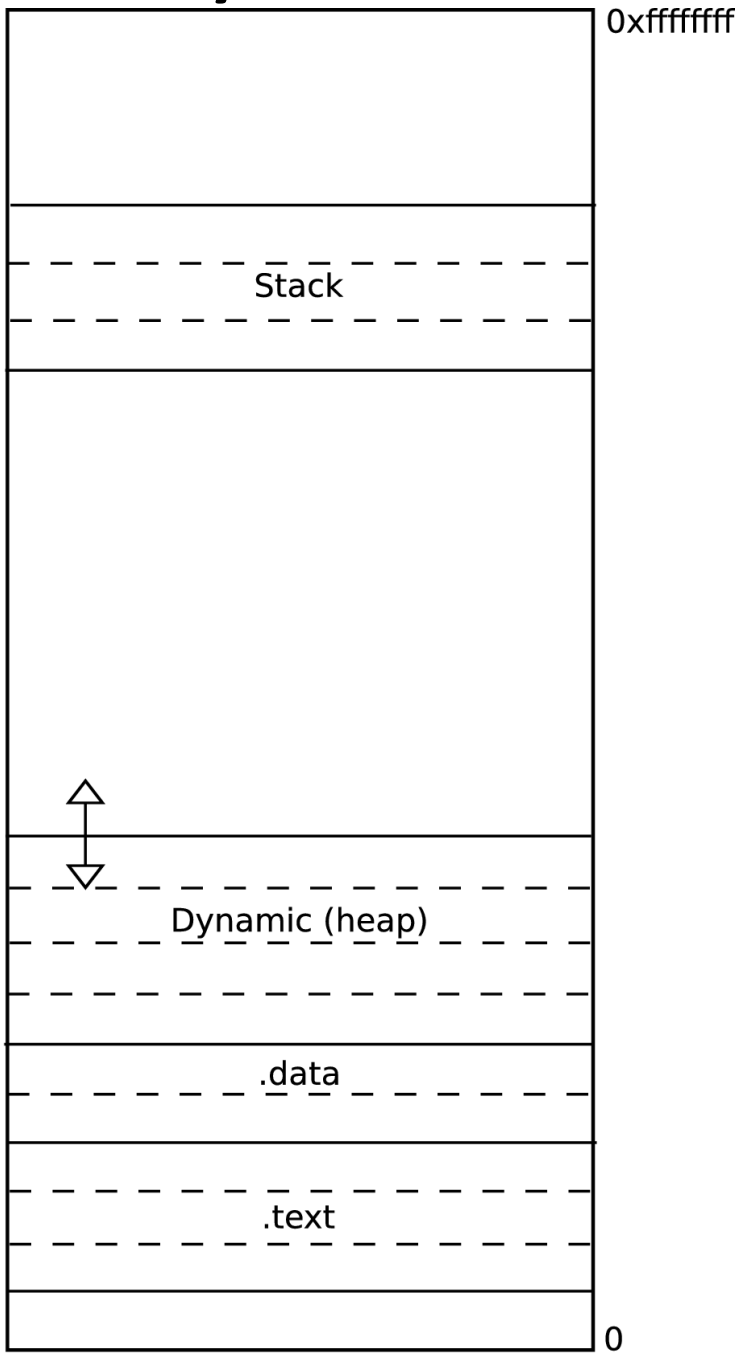


Two Programs – Now What???

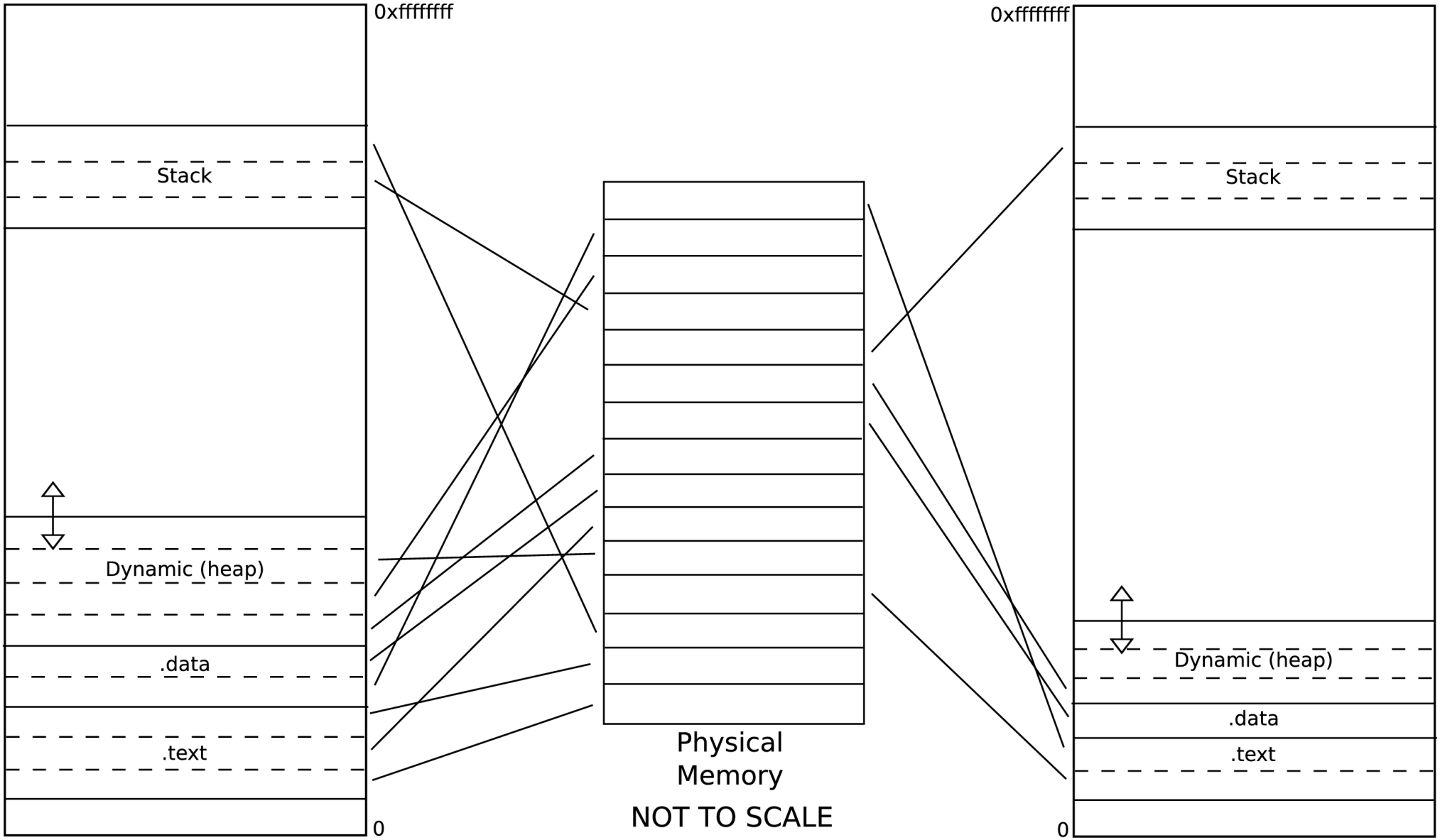


Virtual Memory To The Rescue!

Step 1:
Divide
Memory
Into
4096 byte
“Pages”



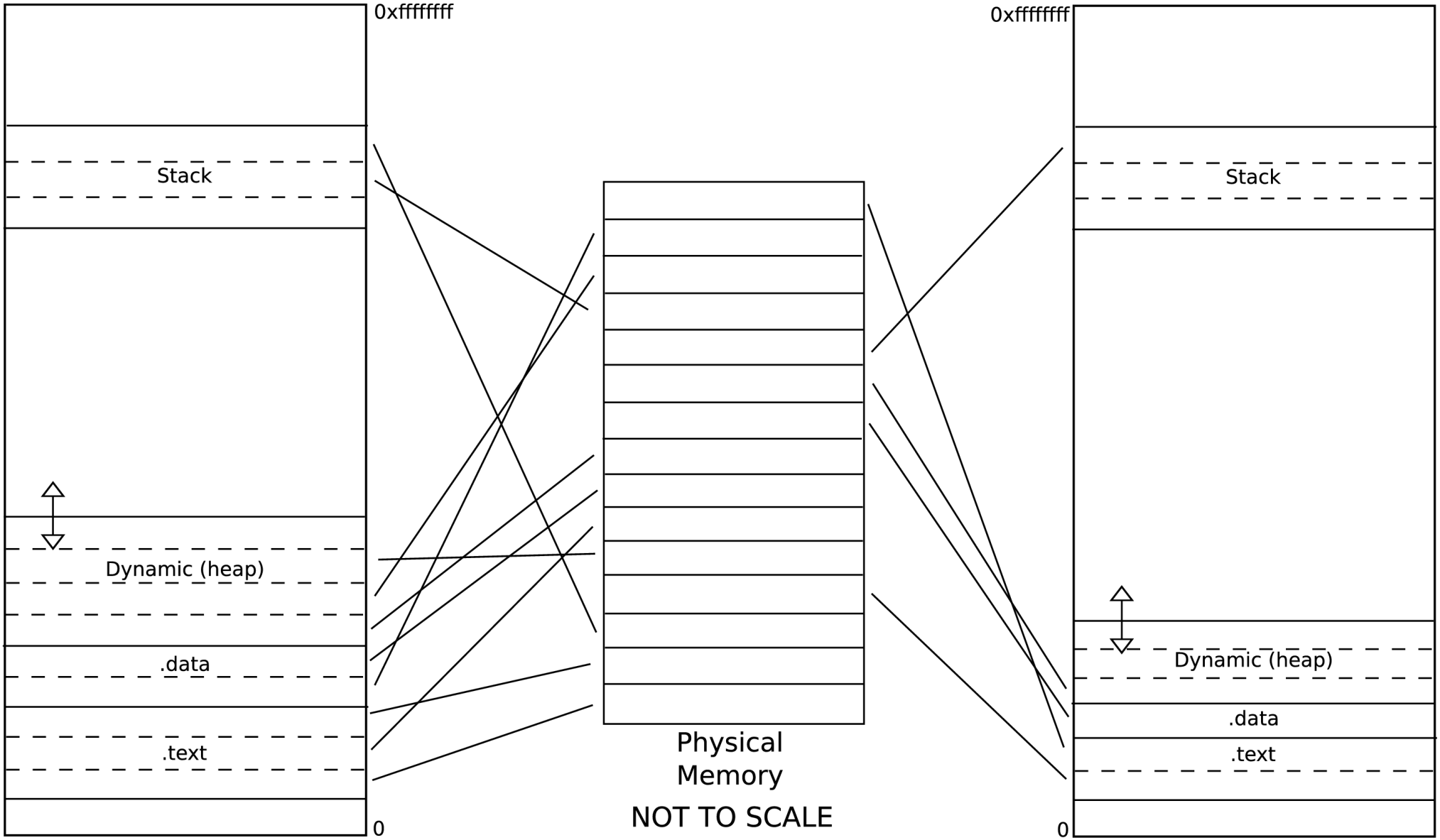
Virtual Pages are Mapped to Physical Frames



Another Motivation: Sometimes Memory Isn't Big Enough

- Virtual Pages are “backed” by the hard disk
- If a page isn't found in physical memory, the hardware generates a **page fault**
- Upon a page fault, the operating system finds the page on disk and **pages it in** to physical memory
- **Demand paging**: memory starts empty, and pages are paged in only when they are first accessed

What if Physical Memory is Full?



Have to Evict a Page: But Which One?

- Random
- LRU: Least Recently Used
- LFU: Least Frequently Used

The Page Table

- Linear page table (i.e. not segmented, not inverted)
- One row per virtual page number
- Columns
  - 2 words (32 bit)
  - 4 bits
  - You decide how to use these
    - You don't need all of them
    - DOCUMENT IN YOUR README!!!!

Assumptions

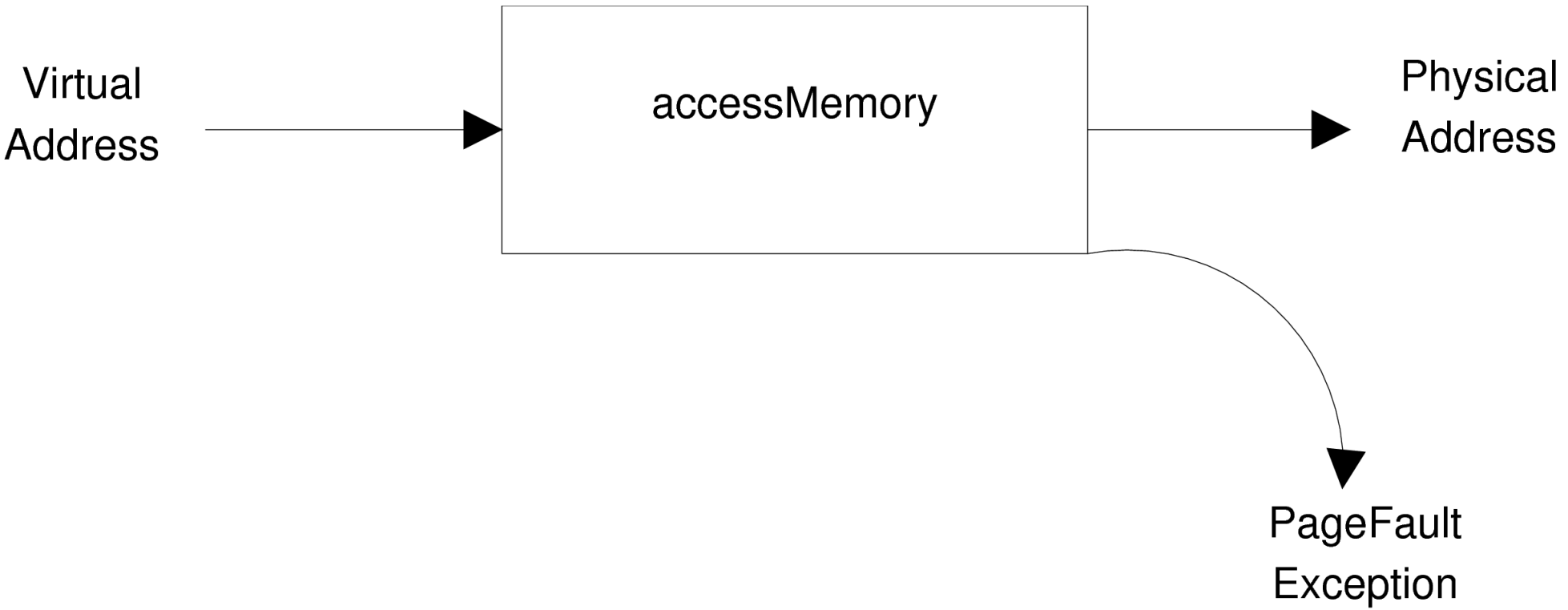
- 32-bit address space
- 4k pages
  - should work with any page size – use `getPageSize()`
- Only one program
- No TLB
- No cache
- Page table doesn't take up memory
  - No paging the page table

Methods To Implement

- long accessMemory (MIPSMachine m, long vaddr, boolean isStore)
- void handlePageFault (MIPSMachine m, long vaddr)
- void startup (MIPSMachine m)
- void shutdown (MIPSMachine m)

Note: they all take MIPSMachine

accessMemory



accessMemory Rules

- Usually done in hardware with gates
- Keep the code short and simple
- Obey the rules in the stencil code
- This is the **easy** method

handlePageFault

- Input: the virtual *address* that wasn't found in physical memory
- Post-condition:
  - The virtual address is in physical memory
  - accessMemory will be called again

handlePageFault

1. Choose a location in physical memory
   - If there's an unused frame, use it
   - If physical memory is full, choose a page to evict
2. Evict the page (if applicable)
3. Page in the requested page
4. Update the page table

PageTable Methods

- `void setWord0 (long vpn, long l)`
- `void setWord1 (long vpn, long l)`
- `void setBit0 (long vpn, boolean b)`
- `void setBit1 (long vpn, boolean b)`
- `void setBit2 (long vpn, boolean b)`
- `void setBit3 (long vpn, boolean b)`

- `long getWord0 (long vpn)`
- `long getWord0 (long vpn)`
- `boolean getBit0 (long vpn)`
- `boolean getBit1 (long vpn)`
- `boolean getBit2 (long vpn)`
- `boolean getBit3 (long vpn)`

pageIn/pageOut (MIPSMachine)

- boolean pageIn (long vpn, long physicalAddr)
  boolean pageOut (long vpn, long physicalAddr)
- Where is the page stored on disk?
  - It's a black box – don't worry about it
- What if it's not found on disk?
  - pageIn/pageOut return false
  - you throw a SegmentationFault exception
- Expensive – don't page out if you don't need to

MIPSMachine

- `long getInstructionCount ()`
  - Strictly increasing – where might you need this?
- `int getMemorySize ()`
- `int getPageSize ()`
  - Don't hard code 4096!
- `PageTable getPageTable ()`
- `void setPageTable (PageTable pt)`
  - Only needs to be called once
- PageTable newPageTable ( )

A Question

The very first instruction in a program is:

```
lw $t0, freelist
```

Does this instruction cause a page fault?

A Question

The very first instruction in a program is:

```
lw $t0, freelist
```

Does this instruction cause a page fault?

Yes

(Remember, physical memory starts out empty)

Another Question

The very first instruction in a program is:

`add $s0, $s0, 10`

Does this instruction cause a page fault?

Another Question

The very first instruction in a program is:

`add $s0, $s0, 10`

Does this instruction cause a page fault?

Yes

(Remember, instructions come from memory too)

Back to the First Question

The very first instruction in a program is:

```
lw $t0, freelist
```

How many times does this instruction page fault?

Back to the First Question

The very first instruction in a program is:

```
lw $t0, freelist
```

How many times does this instruction page fault?

Twice

(Once for instruction, once for freelist)

Running the Emulator

- If you're using ant:
  - ./run-vm arguments program
- Arguments (all optional)
  - -lfu, -lru, -random — Sets the replacement strategy
  - -pages n — Sets the size of physical memory
  - -input filename — Read input from the given file
  - -debug — Enable debug mode
  - -v — Enable visualizer

Programs

- TA-provided programs
  - tests/primes — Prime number generator
  - tests/life — Life (doesn't use much memory)
  - tests/knapsack — Knapsack problem
    - Use -input with knapsack.5, knapsack.100, or knapsack.1000
- Write your own!
  - mips-as MIPSFILE EXEFILE

Demo