

CS 61C: Great Ideas in Computer Architecture (Machine Structures)

Lecture 37: IO Basics

Instructor: Dan Garcia

<http://inst.eecs.Berkeley.edu/~cs61c/>

Review

- Manage memory to disk? Treat as cache
 - Included protection as bonus, now critical
 - Use Page Table of mappings for each process vs. tag/data in cache
 - TLB is cache of Virtual \Rightarrow Physical addr trans
- Virtual Memory allows protected sharing of memory between processes
- Temporal and Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well

Peer Instruction: True or False

A program tries to load a word X that causes a TLB miss but not a page fault. Which are True or False:

1. A TLB miss means that the page table does not contain a valid mapping for virtual page corresponding to the address X
2. There is no need to look up in the page table because there is no page fault
3. The word that the program is trying to load is present in physical memory.

	1	2	3
A:	F	F	T
B:	F	T	F
C:	F	T	T
D:	T	F	F
E:	T	F	T

Peer Instruction: True or False

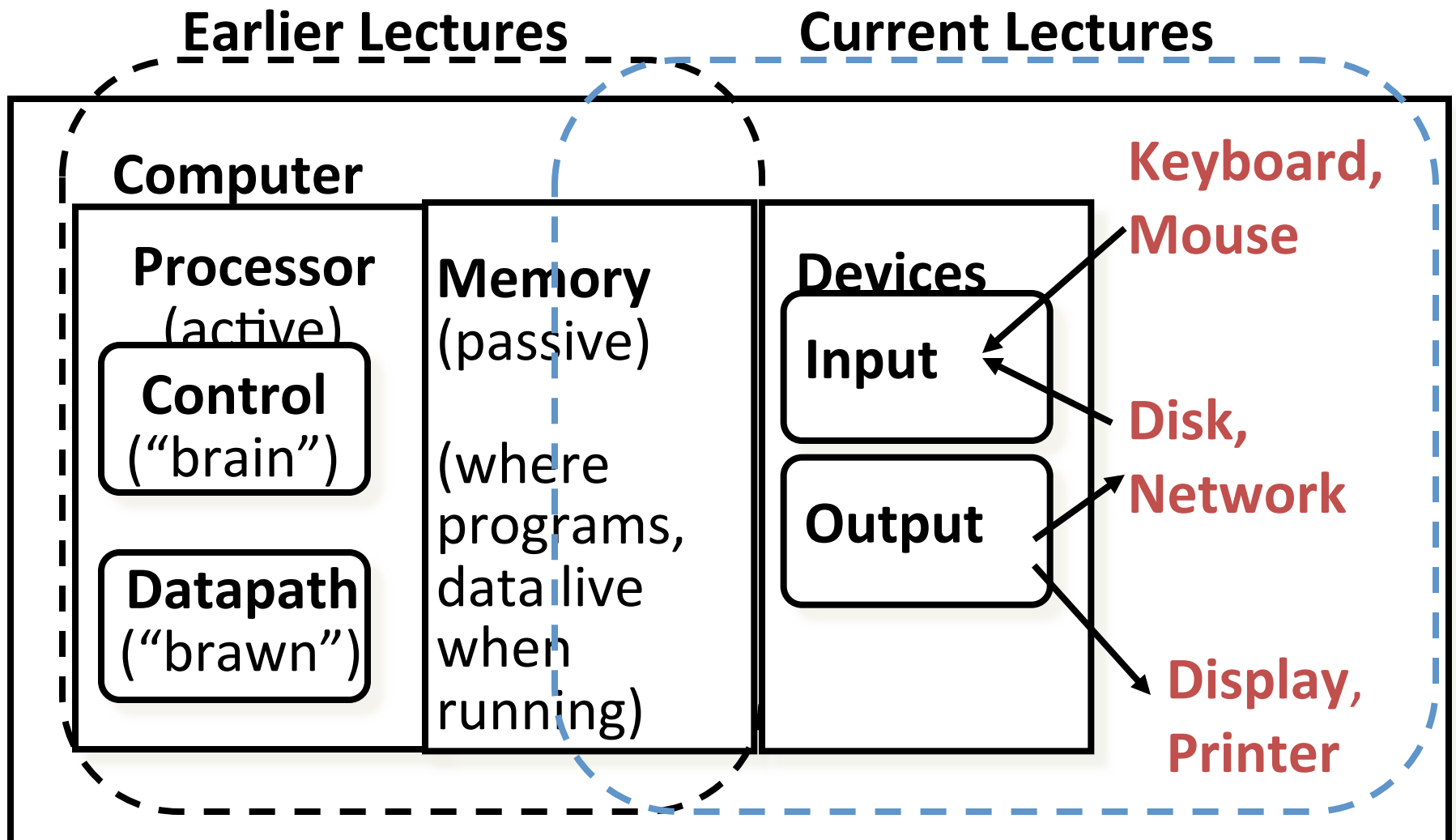
A program tries to load a word X that causes a TLB miss but not a page fault. Which are True or False:

1. A TLB miss means that the page table does not contain a valid mapping for virtual page corresponding to the address X
2. There is no need to look up in the page table because there is no page fault
3. The word that the program is trying to load is present in physical memory.

F A L S E
F A L S E
T R U E

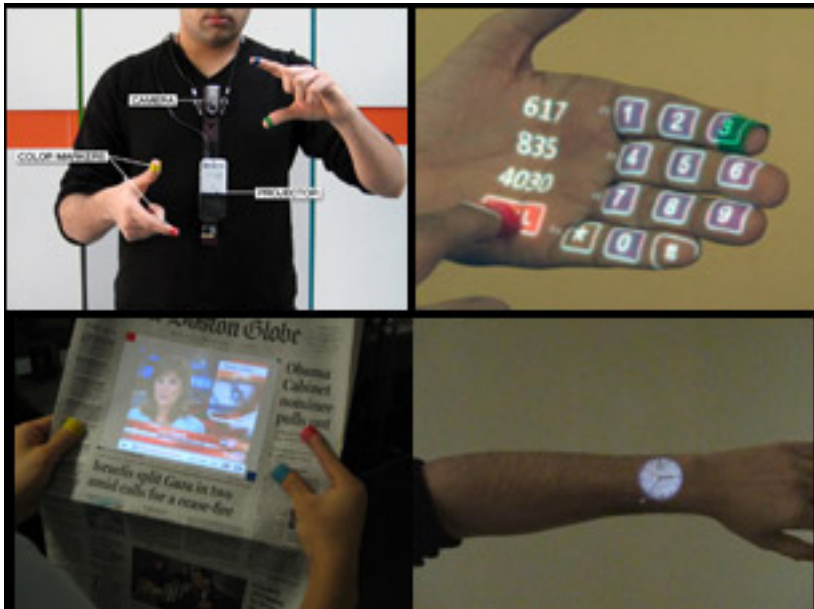
	1	2	3
A:	F	F	T
B:	F	T	F
C:	F	T	T
D:	T	F	F
E:	T	F	T

Recall : 5 components of any Computer



Motivation for Input/Output

- I/O is how humans interact with computers
- I/O gives computers long-term memory.
- I/O lets computers do amazing things:



MIT Media Lab

“Sixth Sense”

<http://youtu.be/ZfV4R4x2SK0>

- Computer without I/O like a car w/no wheels; great technology, but gets you nowhere

www.technologyreview.com/lists/breakthrough-technologies/2013/

Deep Learning

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart. →

Temporary Social Media

Messages that quickly self-destruct could enhance the privacy of online communications and make people freer to be spontaneous. →

Prenatal DNA Sequencing

Reading the DNA of fetuses will be the next frontier of the genomic revolution. But do you really want to know about the genetic problems or musical aptitude of your unborn child? →

Additive Manufacturing

Skeptical about 3-D printing? GE, the world's largest manufacturer, is on the verge of using the technology to make jet parts. →

Baxter: The Blue-Collar Robot

Rodney Brooks's newest creation is easy to interact with, but the complex innovations behind the robot show just how hard it is to get along with people. →

Memory Implants

A maverick neuroscientist believes he has deciphered the code by which the brain forms long-term memories. Next: testing a prosthetic implant for people suffering from long-term memory loss. →

Smart Watches

The designers of the Pebble watch realized that a mobile phone is more useful if you don't have to take it out of your pocket. →

Ultra-Efficient Solar Power

Doubling the efficiency of a solar cell would completely change the economics of renewable energy. Nanotechnology just might make it possible. →

Big Data from Cheap Phones

Collecting and analyzing information from simple cell phones can provide surprising insights into how people move about and behave – and even help us understand the spread of diseases. →

Supergrids

A new high-power circuit breaker could finally make highly efficient DC power grids practical. →

I/O Device Examples and Speeds

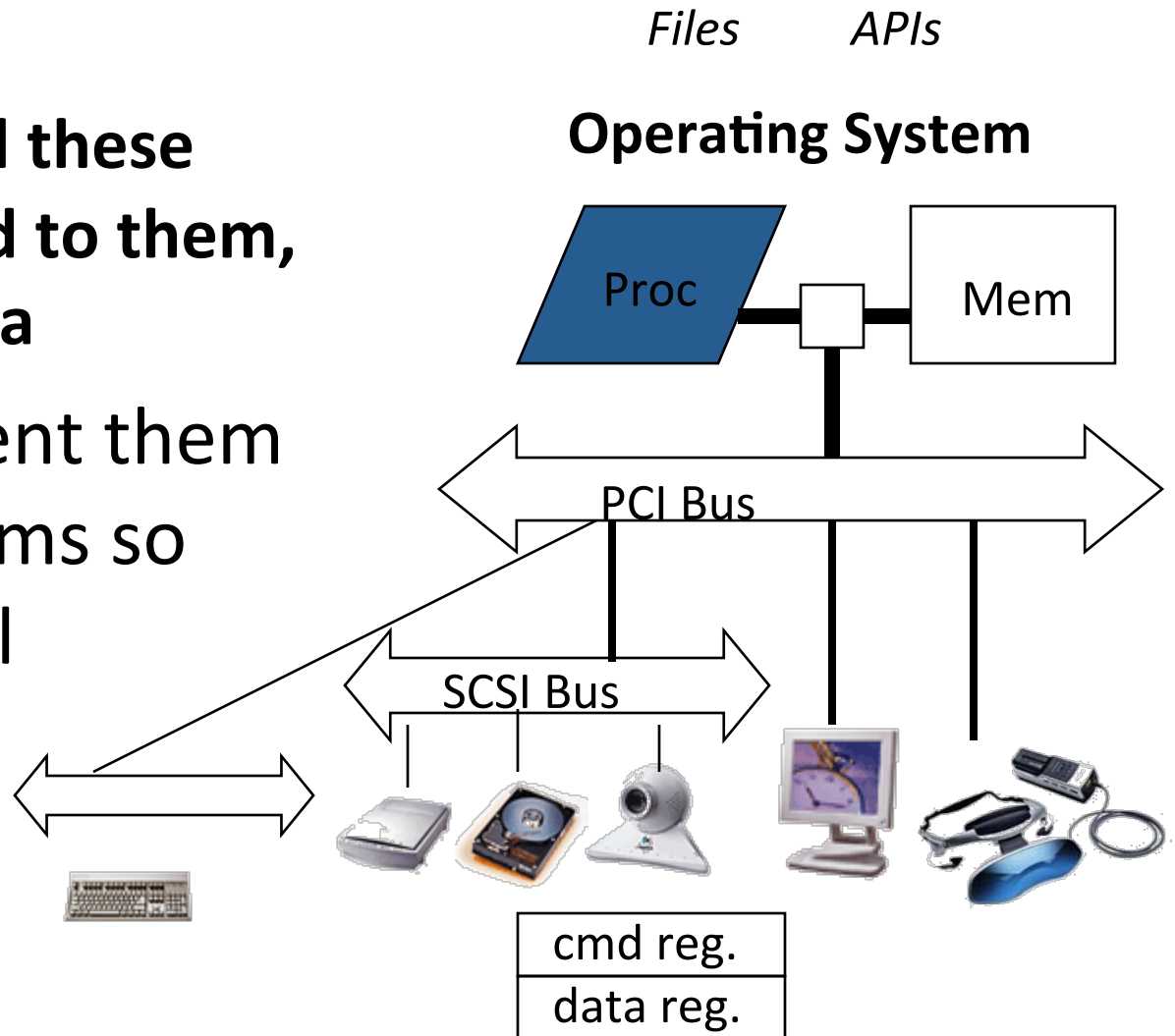
- I/O Speed: bytes transferred per second
(from mouse to Gigabit LAN: 7 orders of magnitude!)

Device	Behavior	Partner	Data Rate (KBytes/s)
Keyboard	Input	Human	0.01
Mouse	Input	Human	0.02
Voice output	Output	Human	5.00
Floppy disk	Storage	Machine	50.00
Laser Printer	Output	Human	100.00
Magnetic Disk	Storage	Machine	10,000.00
Wireless Network	I or O	Machine	10,000.00
Graphics Display	Output	Human	30,000.00
Wired LAN Network	I or O	Machine	125,000.00

When discussing transfer rates, use 10^x

What do we need to make I/O work?

- A way to connect many types of devices
- A way to control these devices, respond to them, and transfer data
- A way to present them to user programs so they are useful

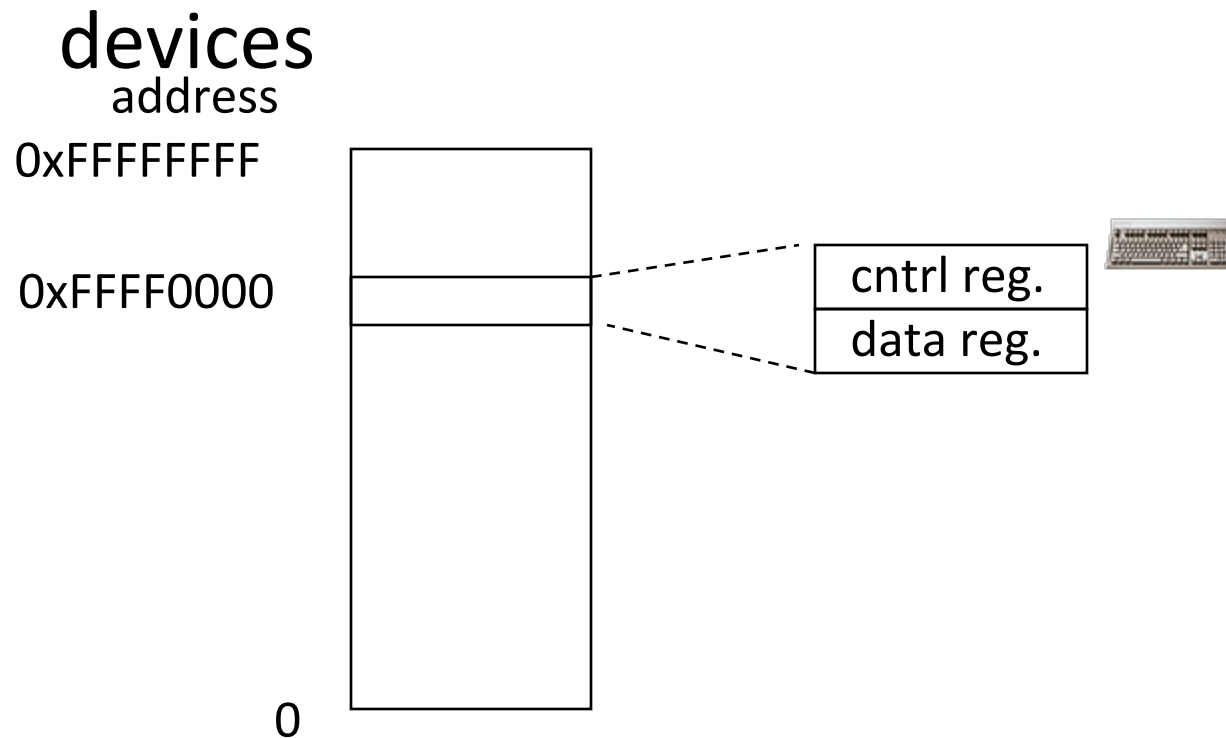


Instruction Set Architecture for I/O

- What must the processor do for I/O?
 - Input: reads a sequence of bytes
 - Output: writes a sequence of bytes
- Some processors have special input and output instructions
- Alternative model (used by MIPS):
 - Use loads for input, stores for output (in small pieces)
 - Called **Memory Mapped Input/Output**
 - A portion of the address space dedicated to communication paths to Input or Output devices (no memory there)

Memory Mapped I/O

- Certain addresses are not regular memory
- Instead, they correspond to registers in I/O devices



Processor-I/O Speed Mismatch

- 1GHz microprocessor can execute 1 billion load or store instructions per second, or 4,000,000 KB/s data rate
 - I/O devices data rates range from 0.01 KB/s to 125,000 KB/s
- Input: device may not be ready to send data as fast as the processor loads it
 - Also, might be waiting for human to act
- Output: device not be ready to accept data as fast as processor stores it
- What to do?

Processor Checks Status before Acting

- Path to a device generally has 2 registers:
 - **Control Register**, says it's OK to read/write (I/O ready) [think of a flagman on a road]
 - **Data Register**, contains data
- Processor reads from Control Register in loop, waiting for device to set **Ready** bit in Control reg (0 \Rightarrow 1) to say its OK
- Processor then loads from (input) or writes to (output) data register
 - Load from or Store into Data Register resets Ready bit (1 \Rightarrow 0) of Control Register
- This is called "**Polling**"

I/O Example (polling)

- Input: Read from keyboard into \$v0

```
Waitloop:      lui    $t0, 0xffff #ffff0000
               lw     $t1, 0($t0) #control
               andi   $t1, $t1, 0x1
               beq    $t1, $zero, Waitloop
               lw     $v0, 4($t0) #data
```

- Output: Write to display from \$a0

```
Waitloop:      lui    $t0, 0xffff #ffff0000
               lw     $t1, 8($t0) #control
               andi   $t1, $t1, 0x1
               beq    $t1, $zero, Waitloop
               sw    $a0, 12($t0) #data
```

“Ready” bit is from processor’s point of view!

Cost of Polling?

- Assume for a processor with a 1GHz clock it takes 400 clock cycles for a polling operation (call polling routine, accessing the device, and returning). Determine % of processor time for polling
 - Mouse: polled 30 times/sec so as not to miss user movement
 - Floppy disk (Remember those?): transferred data in 2-Byte units and had a data rate of 50 KB/second. No data transfer can be missed.
 - Hard disk: transfers data in 16-Byte chunks and can transfer at 16 MB/second. Again, no transfer can be missed. (we'll come up with a better way to do this)

% Processor time to poll

- Mouse Polling [clocks/sec]
= 30 [polls/s] * 400 [clocks/poll] = 12K [clocks/s]
- % Processor for polling:
 $12 * 10^3$ [clocks/s] / $1 * 10^9$ [clocks/s] = 0.0012%
⇒ Polling mouse little impact on processor

% Processor time to poll hard disk

- Frequency of Polling Disk
= $16 \text{ [MB/s]} / 16 \text{ [B/poll]} = 1\text{M [polls/s]}$
- Disk Polling, Clocks/sec
= $1\text{M [polls/s]} * 400 \text{ [clocks/poll]}$
= 400M [clocks/s]
- % Processor for polling:
 $400 * 10^6 \text{ [clocks/s]} / 1 * 10^9 \text{ [clocks/s]} = 40\%$
⇒ Unacceptable

(Polling is only part of the problem – main problem is that accessing in small chunks is inefficient)

What is the alternative to polling?

- Wasteful to have processor spend most of its time “spin-waiting” for I/O to be ready
- Would like an unplanned procedure call that would be invoked only when I/O device is ready
- Solution: use **exception mechanism** to help I/O. **Interrupt** program when I/O ready, return when done with data transfer

Peer Instruction

- 1) A faster CPU will result in faster I/O.
- 2) Hardware designers handle mouse input with interrupts since it is better than polling in almost all cases.
- 3) Low-level I/O is actually quite simple, as it's really only reading and writing bytes.

	123
A:	FFF
B:	FFT
B:	FTF
C:	FTT
C:	TFF
D:	TFT
D:	TTF
E:	TTT

Peer Instruction Answer

- 1) Less sync data idle time
- 2) Because mouse has low I/O rate polling often used
- 3) Concurrency, device requirements vary!

- 1) A faster CPU will result in faster I/O. **TRUE**
- 2) Hardware designers handle mouse input with interrupts since it's better than polling in almost all cases. **FALSE**
- 3) Low level I/O is actually quite simple, as it's really only reading and writing bytes. **FALSE**

	123
A:	FFF
B:	FFT
B:	FTF
C:	FTT
C:	TFE
D:	TFT
D:	TFE
E:	TTT

“And in conclusion...”

- I/O gives computers their 5 senses + long term memory
- I/O speed range is 7 Orders of Magnitude (or more!)
- Processor speed means must synchronize with I/O devices before use
- Polling works, but expensive
 - processor repeatedly queries devices
- Interrupts work, more complex
 - we’ll talk about these next
- I/O control leads to **Operating Systems**

Memory Management Today

- Reality: Slowdown too great to run much bigger programs than memory
 - Called *Thrashing*
 - Buy more memory or run program on bigger computer or reduce size of problem
- Paging system today still important for
 - *Translation* (mapping of virtual address to physical address)
 - *Protection* (permission to access word in memory)
 - Sharing of memory between independent tasks

Impact of Paging on AMAT

- Memory Parameters:
 - L1 cache hit = 1 clock cycles, hit 95% of accesses
 - L2 cache hit = 10 clock cycles, hit 60% of L1 misses
 - DRAM = 200 clock cycles (~100 nanoseconds)
 - Disk = 20,000,000 clock cycles (~ 10 milliseconds)
- Average Memory Access Time (with no paging):
 - $1 + 5\% * 10 + 5\% * 40\% * 200 = 5.5$ clock cycles
- Average Memory Access Time (with paging) =
 - 5.5 (AMAT with no paging) + ?

Impact of Paging on AMAT

- Memory Parameters:
 - L1 cache hit = 1 clock cycles, hit 95% of accesses
 - L2 cache hit = 10 clock cycles, hit 60% of L1 misses
 - DRAM = 200 clock cycles (~100 nanoseconds)
 - Disk = 20,000,000 clock cycles (~ 10 milliseconds)
- Average Memory Access Time (with paging) =
 - $5.5 + 5\% * 40\% * (1 - \text{Hit}_{\text{Memory}}) * 20,000,000$
- AMAT if $\text{Hit}_{\text{Memory}} = 99\%$?
 - $5.5 + 0.02 * 0.01 * 20,000,000 = 4005.5$ (~728x slower)
 - 1 in 20,000 memory accesses goes to disk: 10 sec program takes 2 hours
- AMAT if $\text{Hit}_{\text{Memory}} = 99.9\%$?
 - $5.5 + 0.02 * 0.001 * 20,000,000 = 405.5$
- AMAT if $\text{Hit}_{\text{Memory}} = 99.9999\%$
 - $5.5 + 0.02 * .000001 * 20,000,000 = 5.9$

Impact of TLBs on Performance

- Each TLB miss to Page Table ~ L1 Cache miss
- Page sizes are 4 KB to 8 KB (4 KB on x86)
- TLB has typically 128 entries
 - Set associative or Fully associative
- *TLB Reach*: Size of largest virtual address space that can be simultaneously mapped by TLB:
- $128 * 4 \text{ KB} = 512 \text{ KB} = 0.5 \text{ MB!}$
- What can you do to have better performance?

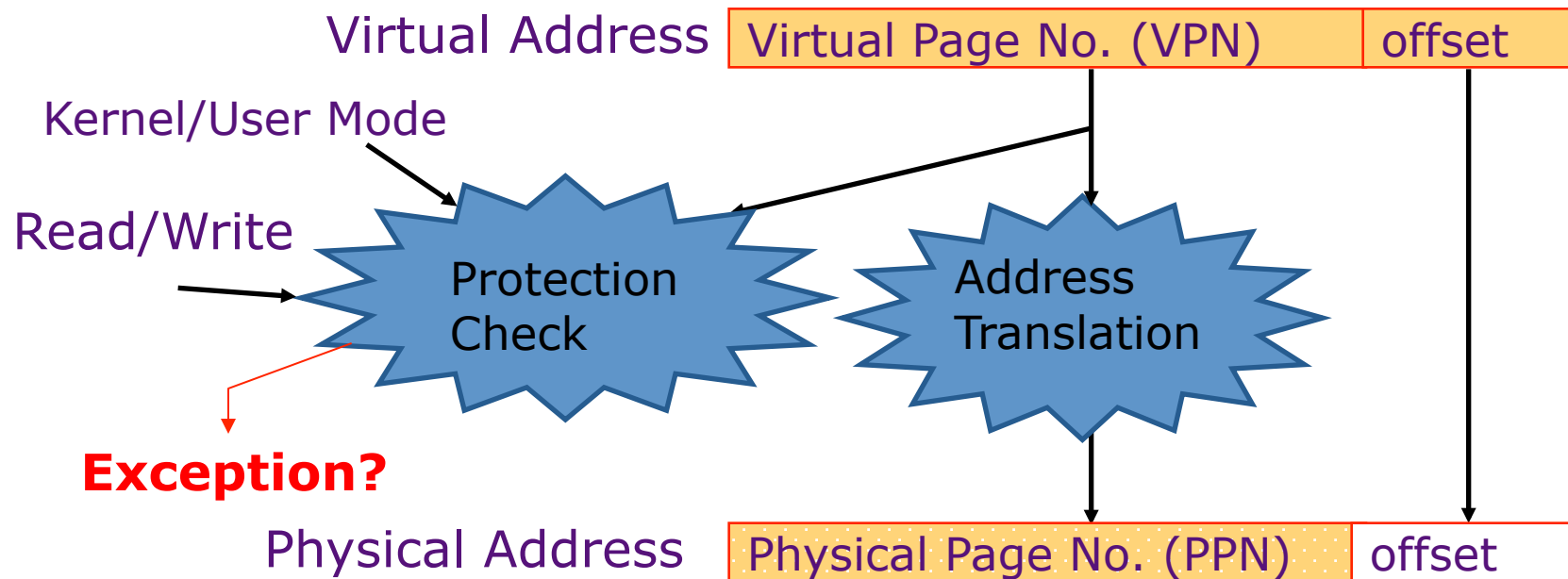
Improving TLB Performance

- Have 2 Levels of TLB just like 2 levels of cache instead of going directly to Page Table on L1 TLB miss
- Add larger page size that operating system can use in situations when OS knows that object is big and protection of whole object OK
 - X86 has 4 KB pages + 2 MB and 4 MB “superpages”

Using Large Pages from Application?

- Difficulty is communicating from application to operating system that want to use large pages
- Linux: “Huge pages” via a library file system and memory mapping; beyond 61C
 - See <http://lwn.net/Articles/375096/>
 - <http://www.ibm.com/developerworks/wikis/display/LinuxP/libhuge+short+and+simple>
- Max OS X: no support for applications to do this (OS decides if should use or not)

Address Translation & Protection



- Every instruction and data access needs address translation and protection checks

A good VM design needs to be fast (~ one cycle) and space efficient

Nehalem Virtual Memory Details

- 48-bit virtual address space, 40-bit physical address space
- Two-level TLB
- I-TLB (L1) has shared 128 entries 4-way associative for 4KB pages, plus 7 dedicated fully-associative entries per SMT thread for large page (2/4MB) entries
- D-TLB (L1) has 64 entries for 4KB pages and 32 entries for 2/4MB pages, both 4-way associative, dynamically shared between SMT threads
- Unified L2 TLB has 512 entries for 4KB pages only, also 4-way associative
- Data TLB Reach
(4 KB only) L1: $64 * 4 \text{ KB} = 0.25 \text{ MB}$, L2: $512 * 4 \text{ KB} = 2 \text{ MB}$
(superpages) L1: $32 * 2\text{-}4 \text{ MB} = 64\text{-}128 \text{ MB}$