CE/CZ2005 and CPE205/CSC205: Operating Systems – Lab Experiment 3

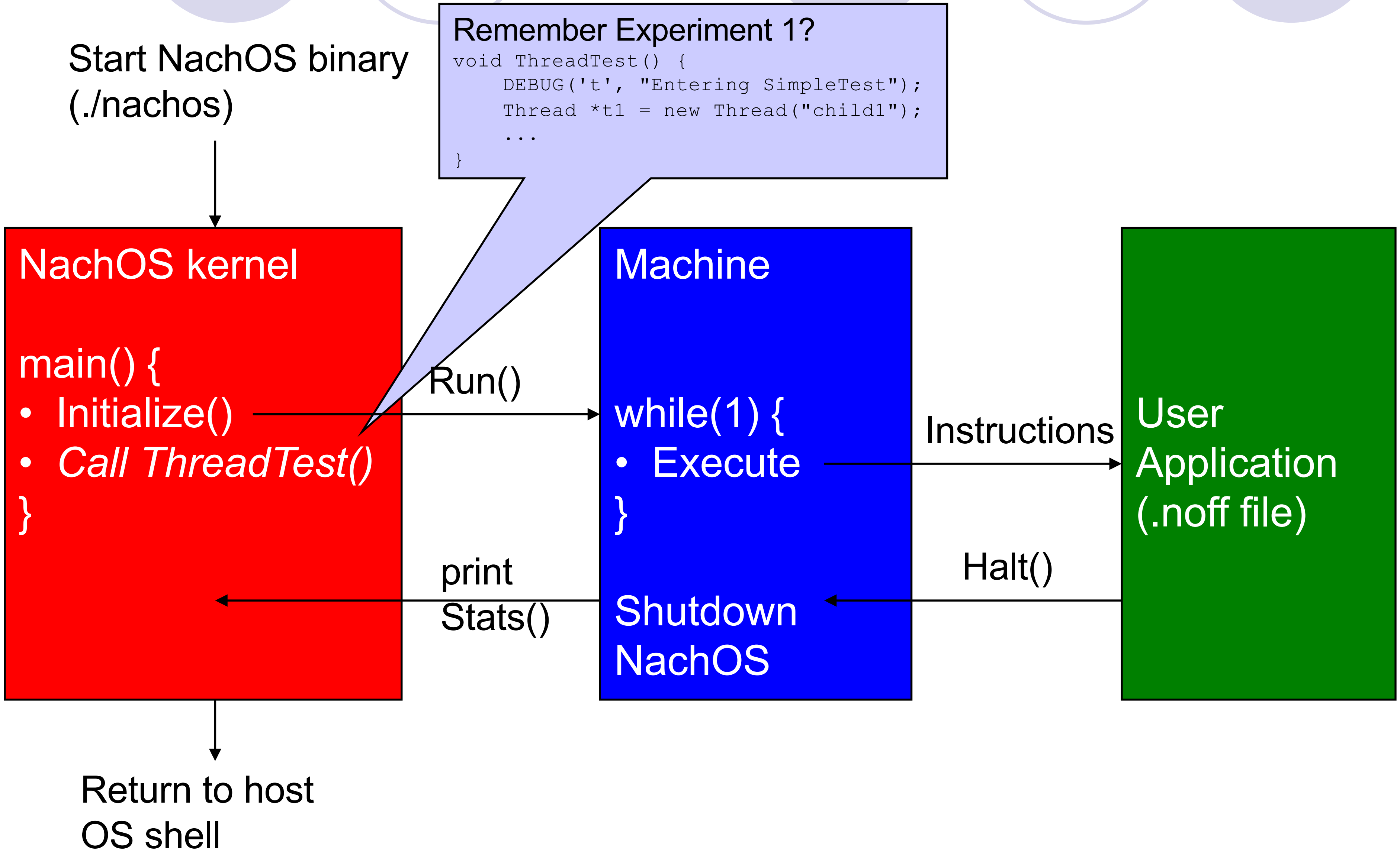
Contact: Bingsheng He (bshe@ntu.edu.sg)

Outline

- Difference between Lab 1 and 2 to Lab 3
- Address translation in NachOS
- Discussion of Experiment 3

Difference between Labs 1 and 2 to Lab 3

NachOS – Remember how it works?

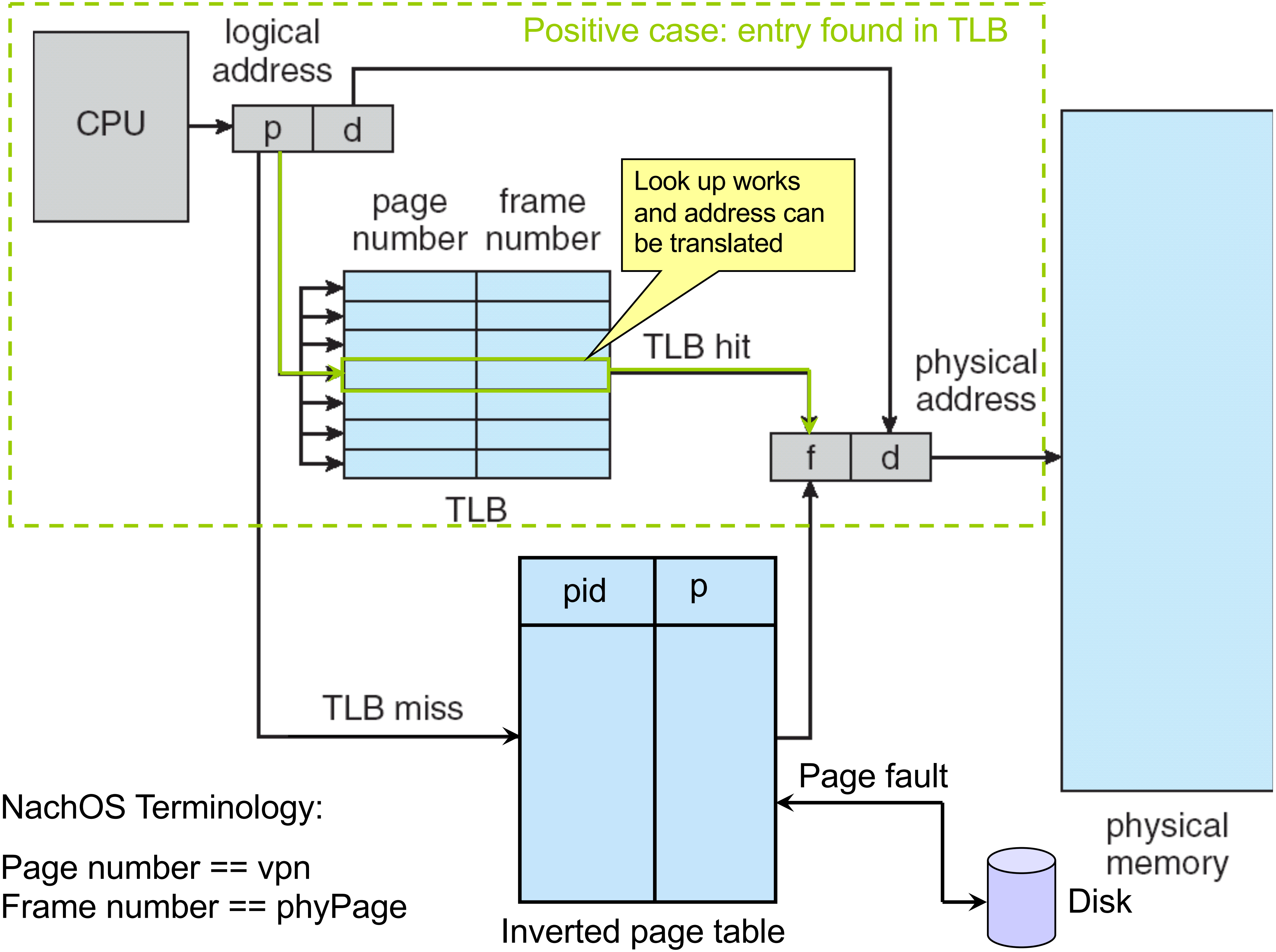


Address Translation in NachOS

Address Translation in NachOS

- How to translate a virtual address into a physical address?
  - Translation Look-aside Buffer (TLB): one per process
  - Inverted Page Table (IPT): one for the entire system

- Function: `Translate(vAddress, ...)`
  - Calculate VPN (Virtual Page Number) and offset
  - Lookup VPN in TLB
  - If lookup was successful:
    - Calculate physical address
  - If lookup was not successful (i.e., TLB miss)
    - Update TLB
    - Try same lookup again (this time it should work)



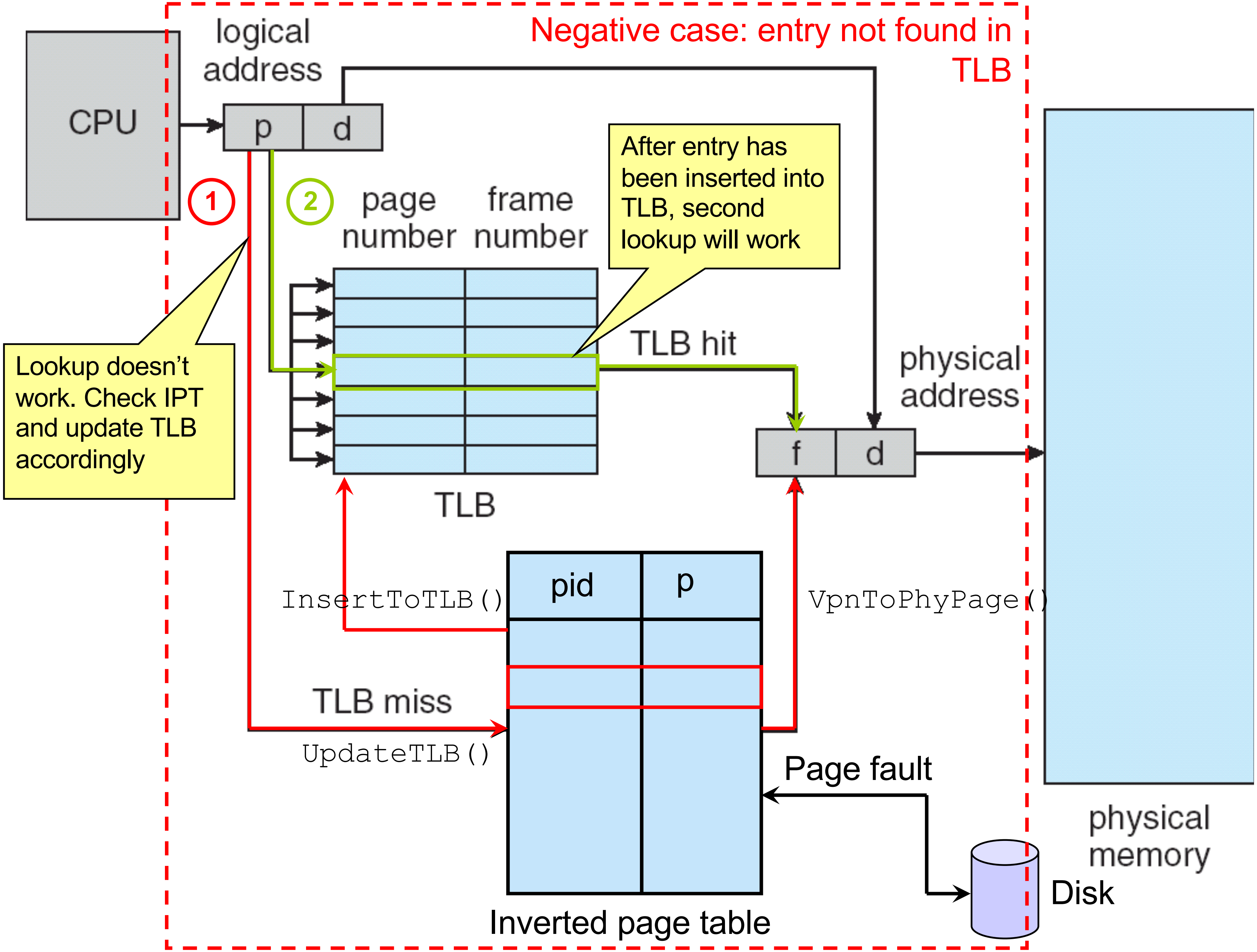
NachOS Terminology:

Page number == vpn
Frame number == phyPage

Experiment 3 – Update TLB

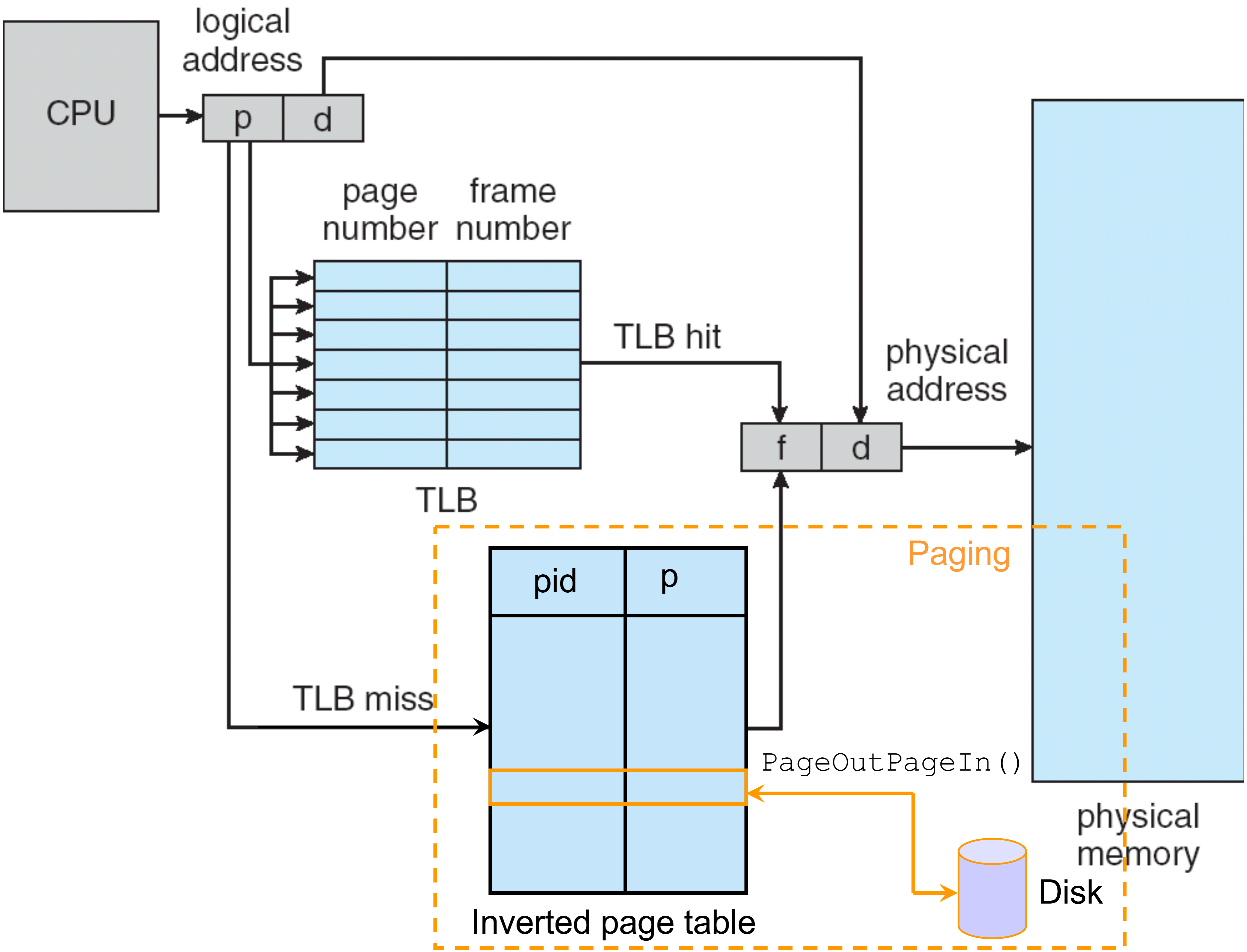
- In case the lookup was not successful (i.e., in case of a TLB miss), how to update the TLB?

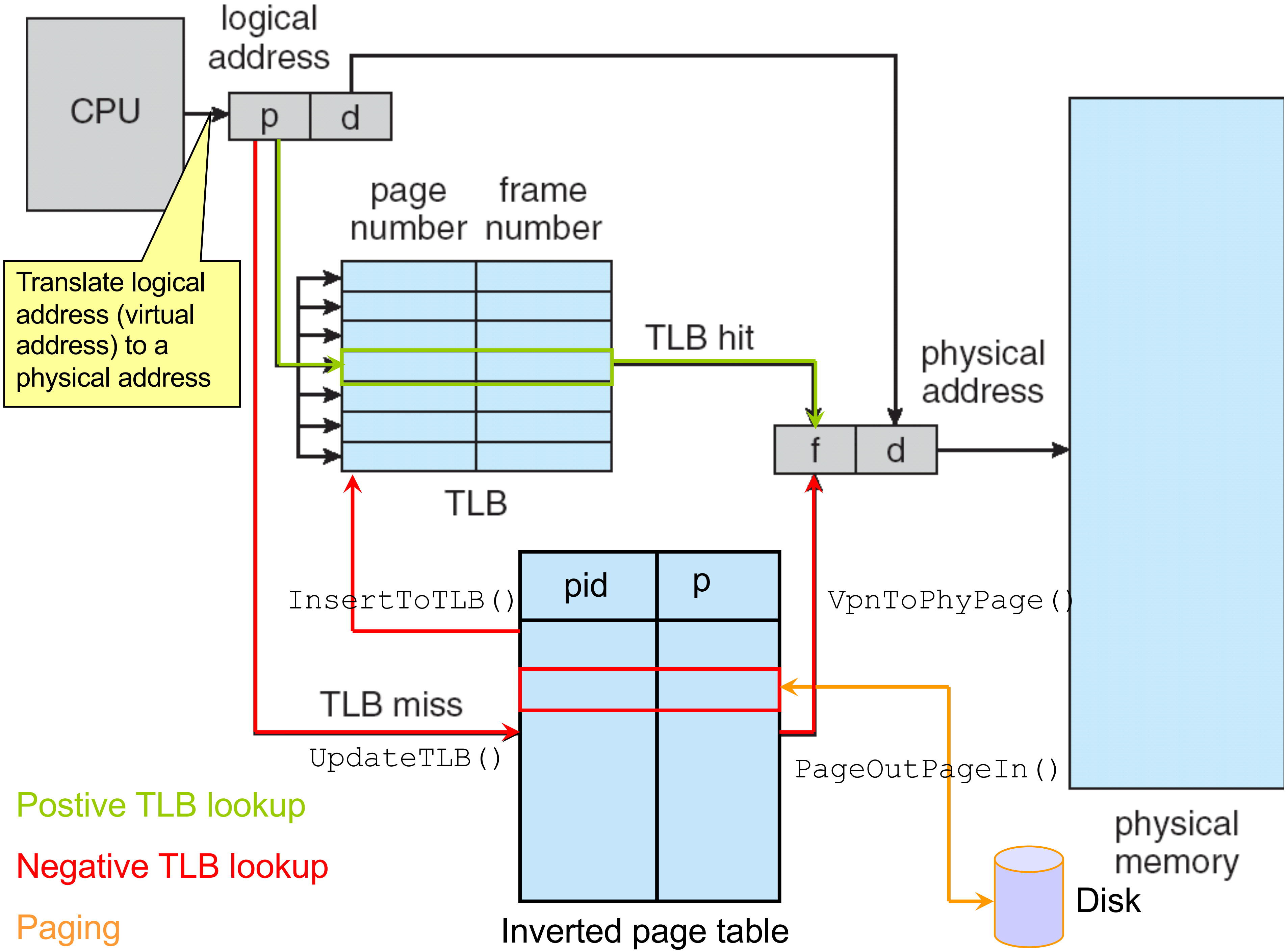
- Function: `UpdateTLB(vAddress)`
  - Determine VPN based on virtual address
  - Check whether VPN can be found in IPT
  - If VPN can be found:
    - Insert the VPN/PhyPage into TLB
  - If not:
    - Perform paging


Negative case: entry not found in TLB
CPU
logical address
p
d
1
2
page number
frame number
After entry has been inserted into TLB, second lookup will work
Lookup doesn't work. Check IPT and update TLB accordingly
TLB hit
physical address
f
d
TLB
pid
p
InsertToTLB()
VpnToPhyPage()
TLB miss
UpdateTLB()
Page fault
physical memory
Disk
Inverted page table

Experiment 3 – Update TLB

- If VPN cannot be found in IPT, how to perform paging?
- Function: `PageOutPageIn(vpn)`
  - Determine the victim frame using the clock algorithm
  - Page out victim page
    - Write victim page to swap file
  - Page in the new page
    - Load new physical page from swap file
    - Update the IPT table with VPN/PhyPage combination


CPU
logical address
p
d
page number
frame number
TLB hit
physical address
f
d
TLB
Paging
pid
p
TLB miss
PageOutPageIn()
physical memory
Disk
Inverted page table


CPU
logical address
p
d
page number
frame number
Translate logical address (virtual address) to a physical address
TLB hit
physical address
f
d
TLB
pid
p
InsertToTLB()
VpnToPhyPage()
TLB miss
UpdateTLB()
PageOutPageIn()
Postive TLB lookup
Negative TLB lookup
Paging
physical memory
Disk
Inverted page table

Discussion of Experiment 3

Experiment 3 – Overview

- Objective
  - Understand why TLB can provide fast address translation
  - Know how address translation is done by using IPT
  - Understand how page replacement is essential to virtual memory
  - Understand how to implement a clock replacement algorithm

- Tasks
  - Implement missing functionality for address translation
  - Run test program and analyse output

Directory Structure

bin	For generating NachOS format files, DO NOT CHANGE!
filesys	NachOS kernel related to file system, DO NOT CHANGE!
lab1	**Experiment 1**, no coding is required.
lab2	Experiment 2, process synchronization.
machine	MIPS H/W simulation, DO NOT CHANGE unless asked.
Makefile.common Makefile.dep	For compilation of NachOS, DO NOT CHANGE!
network	NachOS kernel related to network, DO NOT CHANGE!
port	Additional experiment for students registered with CPE205/CSC205
readme	Short description of OS labs and assessments
test	NachOS format files for testing virtual memory, DO NOT CHANGE!
threads	NachOS kernel related to thread management, DO NOT CHANGE!
userprog	NachOS kernel related to running user applications, DO NOT CHANGE!
vm	Experiment 3, coding virtual memory (TLB, page replacement)

Experiment 3 – Overview

- Incomplete VM code can be found in
  - `vm/tlb.cc`
- You need to add your code to these 3 functions:
  - `int VpnToPhyPage(int vpn)`
  - `void InsertToTLB(int vpn, int phyPage)`
  - `int clockAlgorithm(void)`
- Binary test program for Experiment 3 is provided
  - Execute the test program:
    - `./nachos -x ../test/vmtest.noff -d`

Experiment 3 – InsertToTLB()

- Put a virtual page number and its associated physical page into the TLB

```
void InsertToTLB(int vpn, int phyPage) {
    int i = 0; //entry in the TLB

    //your code to find an empty in TLB or to
    //replace the oldest entry if TLB is full
    ???

    //copy dirty data to memoryTable
    ...
    memoryTable[phyPage].clockCounter = 0
}
```

This is the reset to 0, mentioned in the lab manual. It's already done for you, so you don't need to worry about it

- What do you need to do?
  - Replace the ??? with your code, don't touch the rest
  - Your code has to make sure that *i* is set correctly
    - i.e., the *i*-th entry in the TLB is either *empty* or the *oldest*

Experiment 3 – InsertToTLB()

- Check all entries in the TLB whether there are any invalid entries

- How to do that?
  - The TLB is an array of `TranslationEntry` objects:
    - `TranslationEntry *tlb;`
    - You can access the TLB in the following way:
      - `machine->tlb[i]`
  - A translation entry has several flags
    - For example: you can check whether an entry is valid:
      - `machine->tlb[i].valid`
    - Check `machine/translate.h` for more details
  - The size of that array is defined by the constant `TLBSize`

Experiment 3 – InsertToTLB()

- If there is an invalid TLB entry, then *i* should point to it

- For example, if the 2nd entry is invalid then *i=1*
  - The new VPN/PhyPage value will be inserted into the 2nd entry of the table

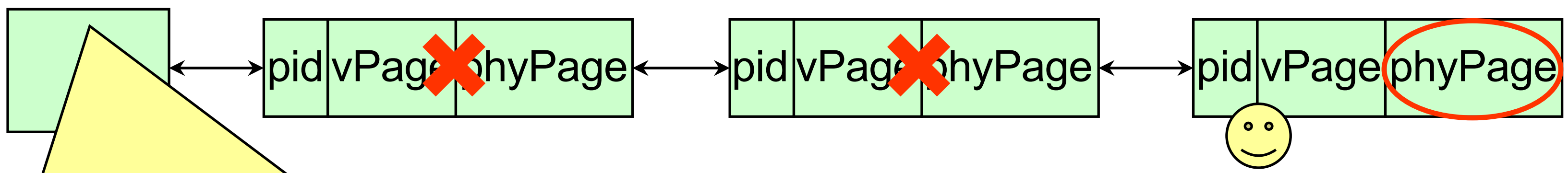
# Entry	VPN	PhyPage	Valid	ReadOnly	Use	Dirty
0	...	...	TRUE	...	...	...
1	...	...	FALSE	...	...	...
2	...	...	TRUE	...	...	...

Experiment 3 – InsertToTLB()

- If there is no invalid entry, then *i* should point to the oldest entry
  - The oldest entry will be replaced by the new VPN/PhyPage values
  - You need to keep track of the oldest entry
  - C++ hint: use a static variable for that purpose
    - `static int FIFOPointer = 0;`
    - Once a static variable is initialised, it remains in the memory
    - No re-initialisation afterwards
  - Make sure FIFOPointer is always correctly pointing to the oldest entry
    - Simple FIFO: if an entry is just inserted, then the entry next to it is the oldest entry
    - `FIFOPointer = (i + 1) % TLBSize`

Experiment 3 – VpnToPhyPage()

- Return the physical page for a VPN (if it exists in the IPT)
  - `int VpnToPhyPage(int vpn) { ... }`
- IPT is realised as hash table:



Access to the first entry:
`IptEntry* iptPtr =hashIPT(vpn, currentThread->pid)`

- Multiple entries are chained in a linked list
  - To access the next element use `iptPtr = iptPtr->next`
- Return the `iptPtr->phyPage` value from the IPT entry for which the following condition is true:
  - `iptPtr->vPage==vpn && pid==currentThread->pid`
- Return -1 if no entry can be found that matches the above condition

Experiment 3 – clockAlgorithm()

- Determines which physical page should be paged out
  - `int clockAlgorithm(void)`

- A memory table is used by the clock algorithm
  - Can be accessed by using `memoryTable[i]`

- This memory table has as many entries as there are physical pages
  - A constant is used for that purpose: `NumPhysPages`

- Three things are of importance
  - `memoryTable[i].valid` – Is the *i*-th entry valid?
  - `memoryTable[i].dirty` – Is the *i*-th entry dirty?
  - `memoryTable[i].counter` – How often has the page been used?

Experiment 3 – clockAlgorithm()

- A `clockPointer` is used which has to remember its position
  - Use a static variable for this purpose
    - `static int clockPointer = 0;`

- Repeatedly loop over all entries until you find a page that fulfills one of these conditions:
  - Is invalid:
    - `memoryTable[clockPointer].valid == false`
  - Is not dirty and old enough:
    - `!memoryTable[clockPointer].dirty && memoryTable[clockPointer].clockCounter == OLD_ENOUGH`
  - Is dirty, old enough, and already got a second chance
    - `memoryTable[clockPointer].dirty && memoryTable[clockPointer].clockCounter == OLD_ENOUGH + DIRTY_ALLOWANCE`

Experiment 3 – clockAlgorithm()

- After you have checked these conditions for one entry, increase the clock counter
  - `memoryTable[clockPointer].clockCounter++`
  - This is to ensure that eventually one page will be old enough
- Also, in order to check the next page, advance the clock pointer
  - `clockPointer = (clockPointer + 1) % NumPhysPages`
- Once a victim has been found
  - Return the page number (i.e., `return clockPointer`)
  - Advance the clock pointer before return (important because the page that is currently being pointed at will be replaced)

Experiment 3 – Analysis of Output

VPN	TLB Miss/ Page Fault	Page In	Page Out	PhyPage	TLBEntry Inserted
0	Page Fault	0	-	0	0
9	Page Fault	9	-	1	1
...	...	...	...	...	...
0	TLB Miss	-	-	0	1
...	...	...	...	...	...
9	Page Fault	9	27	2	2

- Analyse the output of your program and complete the table
- Write down the following
  - Page size (defined in NachOS)
  - Number of physical frames (defined in NachOS)
  - TLB size (defined in NachOS)
  - Number of pages used by the test program
  - Number of page faults that occurred during execution of the test program

Experiment 3 – Summary

- No group effort
- Leave a working copy of your code in your account
- Report:
  - Include analysis of test program output which verifies that your solution works
  - Cleary explain which TLB entry or physical frame should be used whenever there is a TLB miss or page fault
  - Do not attach any debug output
  - Due ONE WEEK AFTER your lab session

Acknowledgement

- The slides are revised from the previous versions created by Dr. Heiko Aydt.