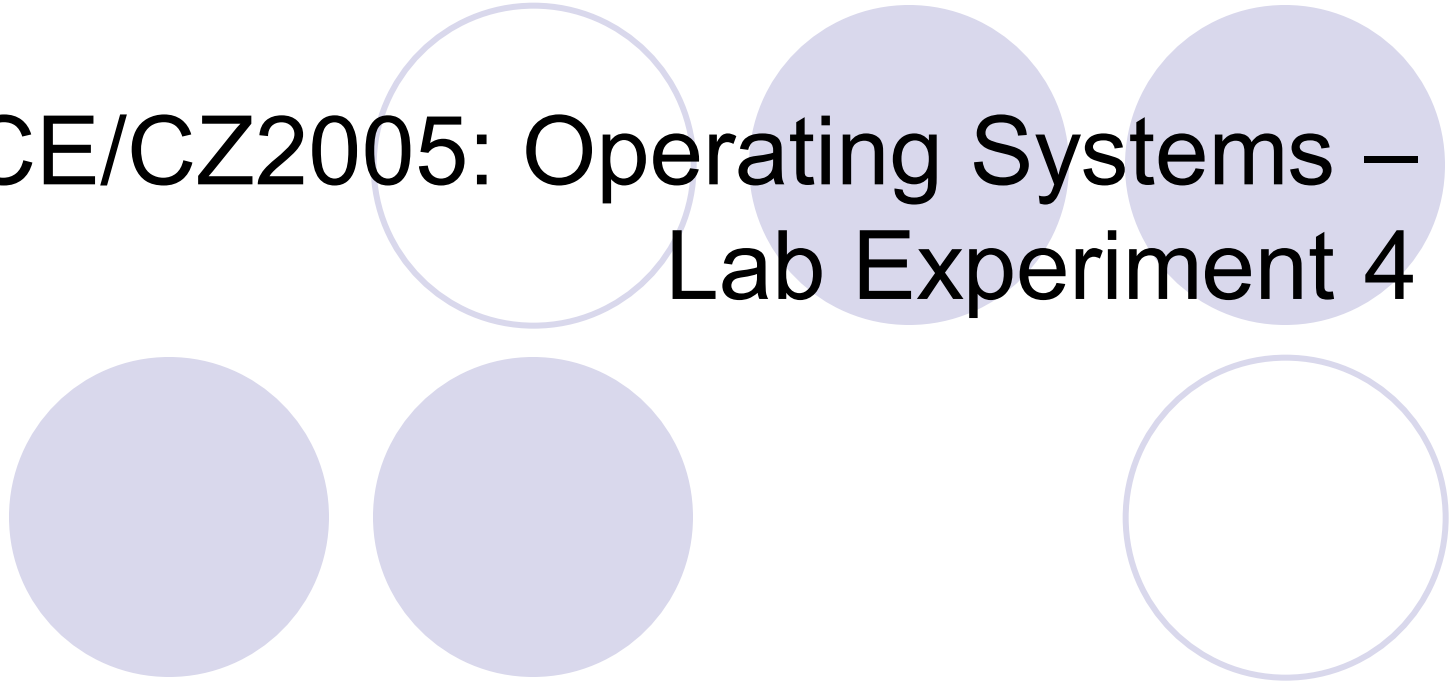
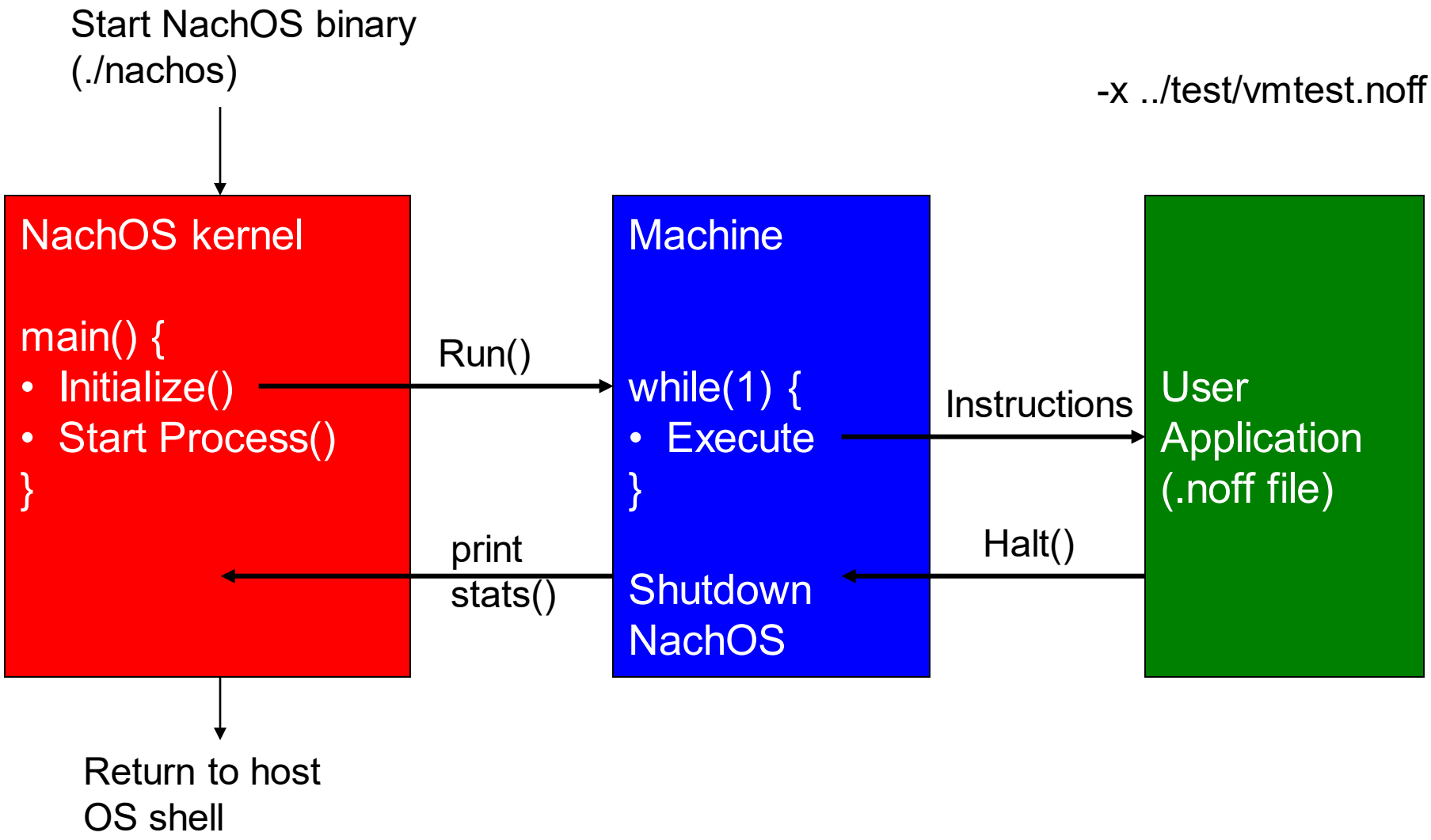
CE/CZ2005: Operating Systems – Lab Experiment 4



Outline

- Address translation in NachOS
- Discussion of Experiment 4

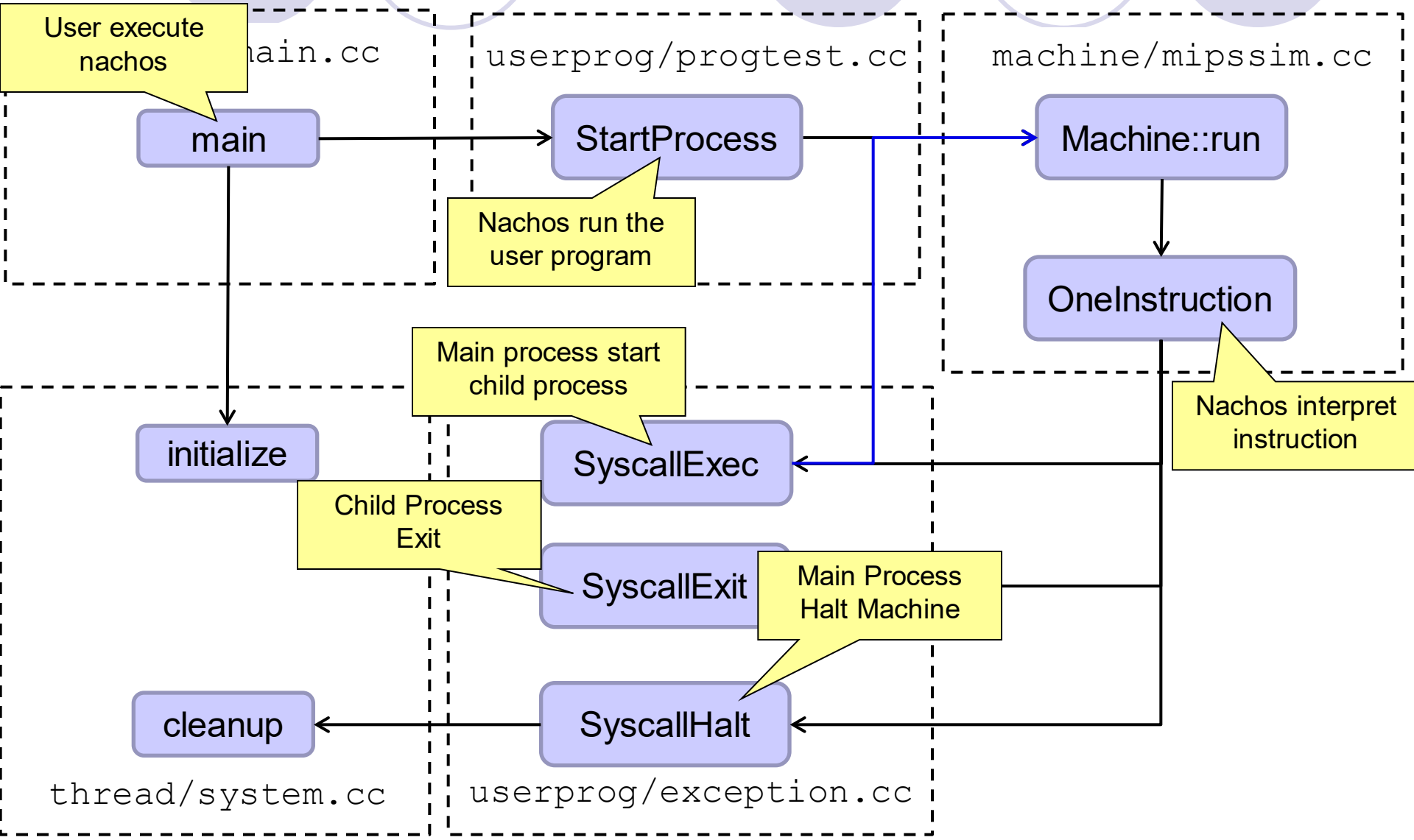
NachOS – Remember how it works?



User Application

- Source code for the user application (`test/vmtest.noff`) can be found in `test/vmtest.c`.
- `vmtest` executes 1 child processes (`test/vm.noff`).

Execution Flow of User Application



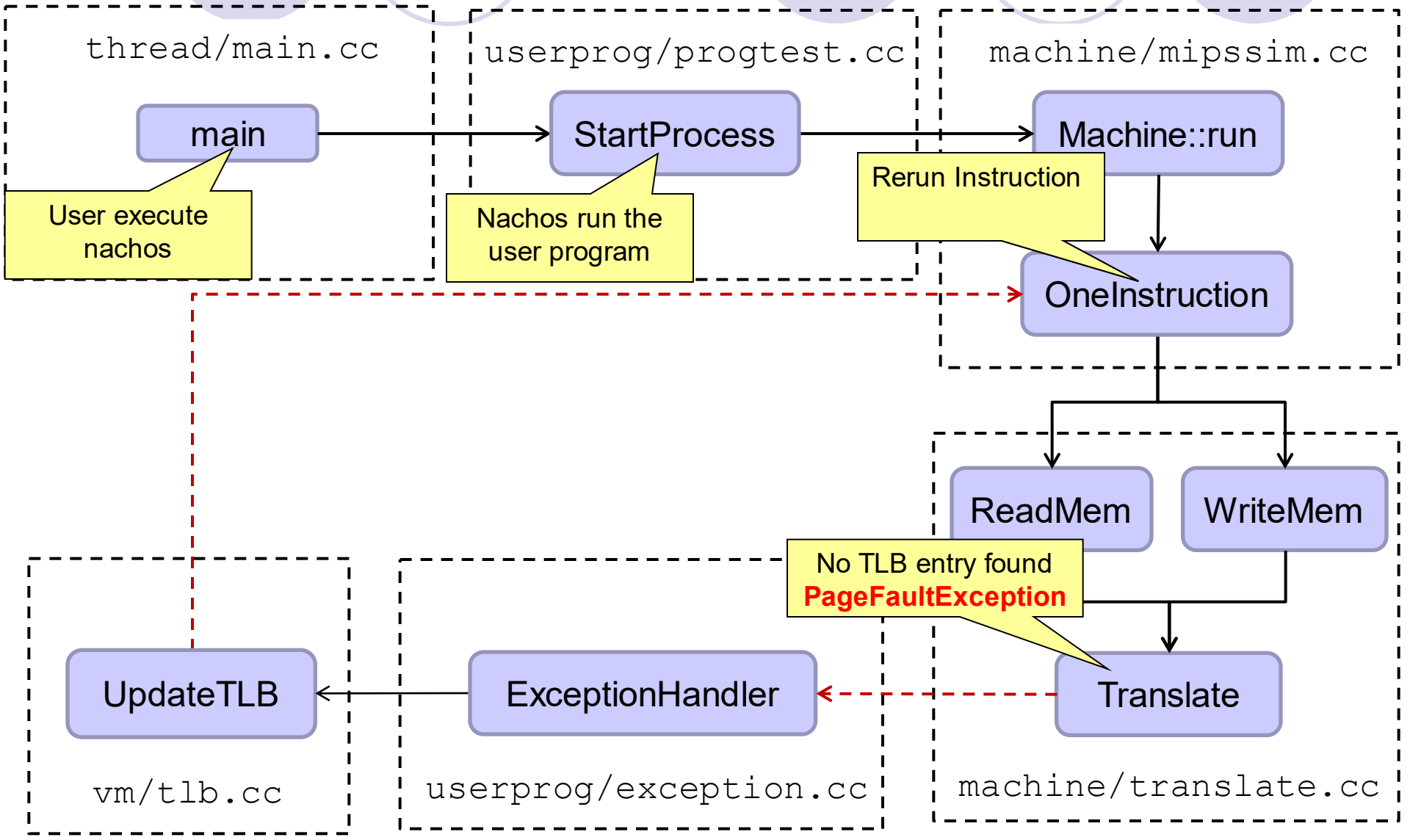
Address Translation in NachOS

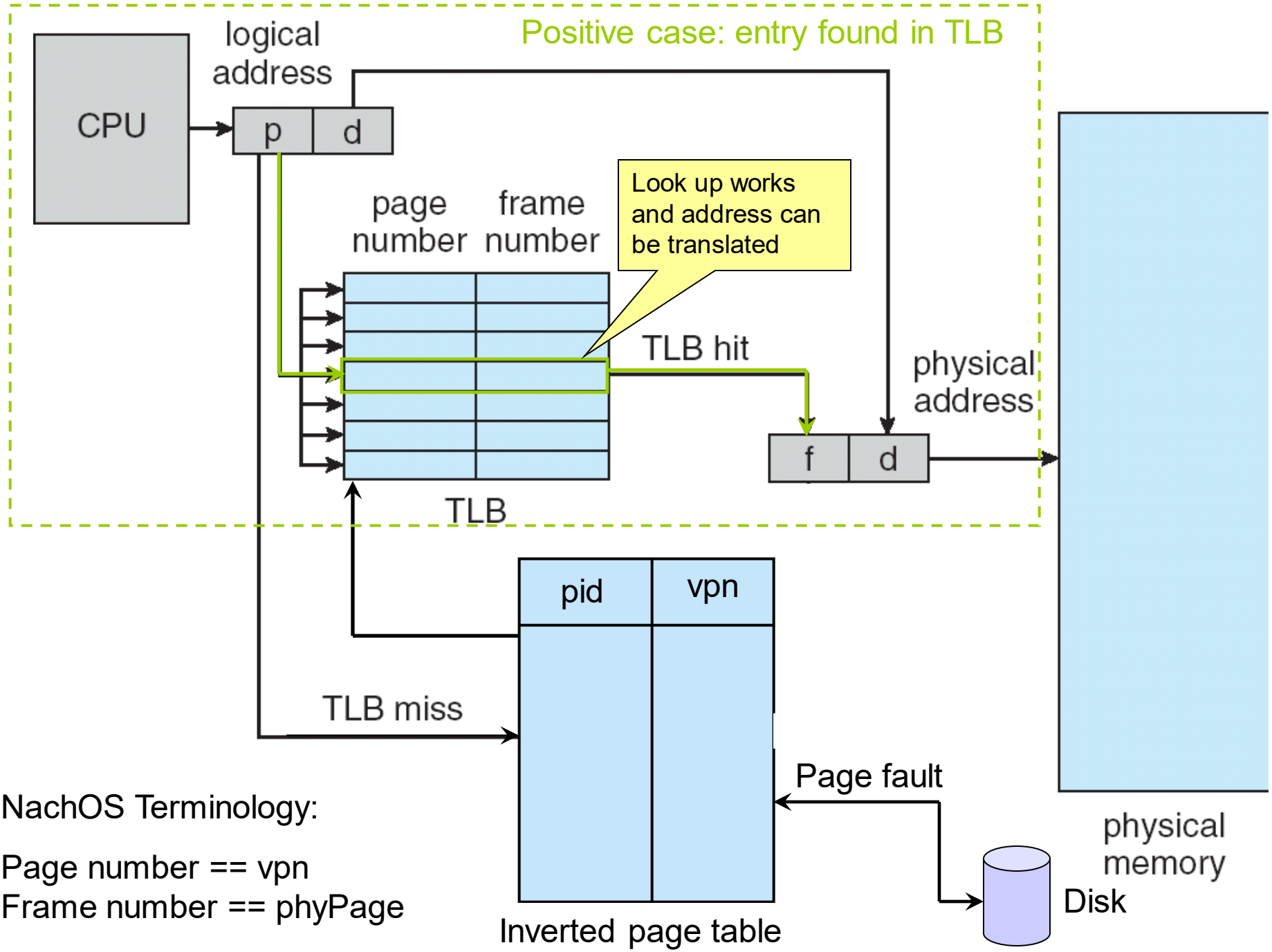
Address Translation in NachOS

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

- Call `Translate(vpn,&phyAddr ...)` (defined under `machine/translate.cc`)
  - Calculate VPN (Virtual Page Number)
  - Lookup VPN in TLB
  - If lookup was successful
    - Calculate physical address
  - If lookup was not successful (i.e., TLB miss)
    - Generate an exception
- If an exception is generated
  - Call `updateTLB(vpn)`
  - Call `Translate(vpn,&phyAddr ...)` again

Virtual Address Translation

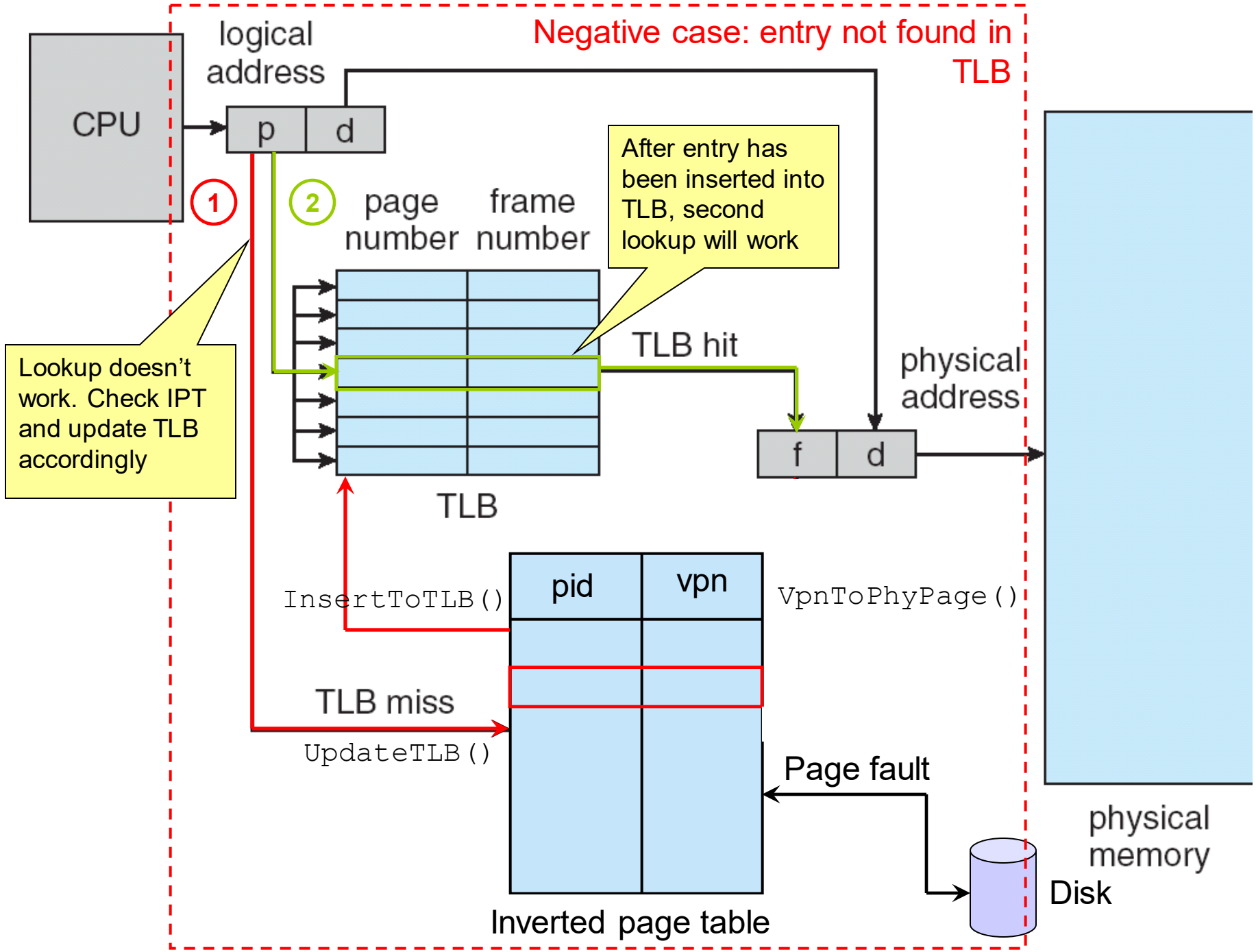



Positive case: entry found in TLB
CPU
logical address
p
d
page number
frame number
Look up works and address can be translated
TLB hit
physical address
f
d
TLB
pid
vpn
TLB miss
Page fault
physical memory
Disk
Inverted page table
NachOS Terminology:
Page number == vpn
Frame number == phyPage

Experiment 4 – Update TLB

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

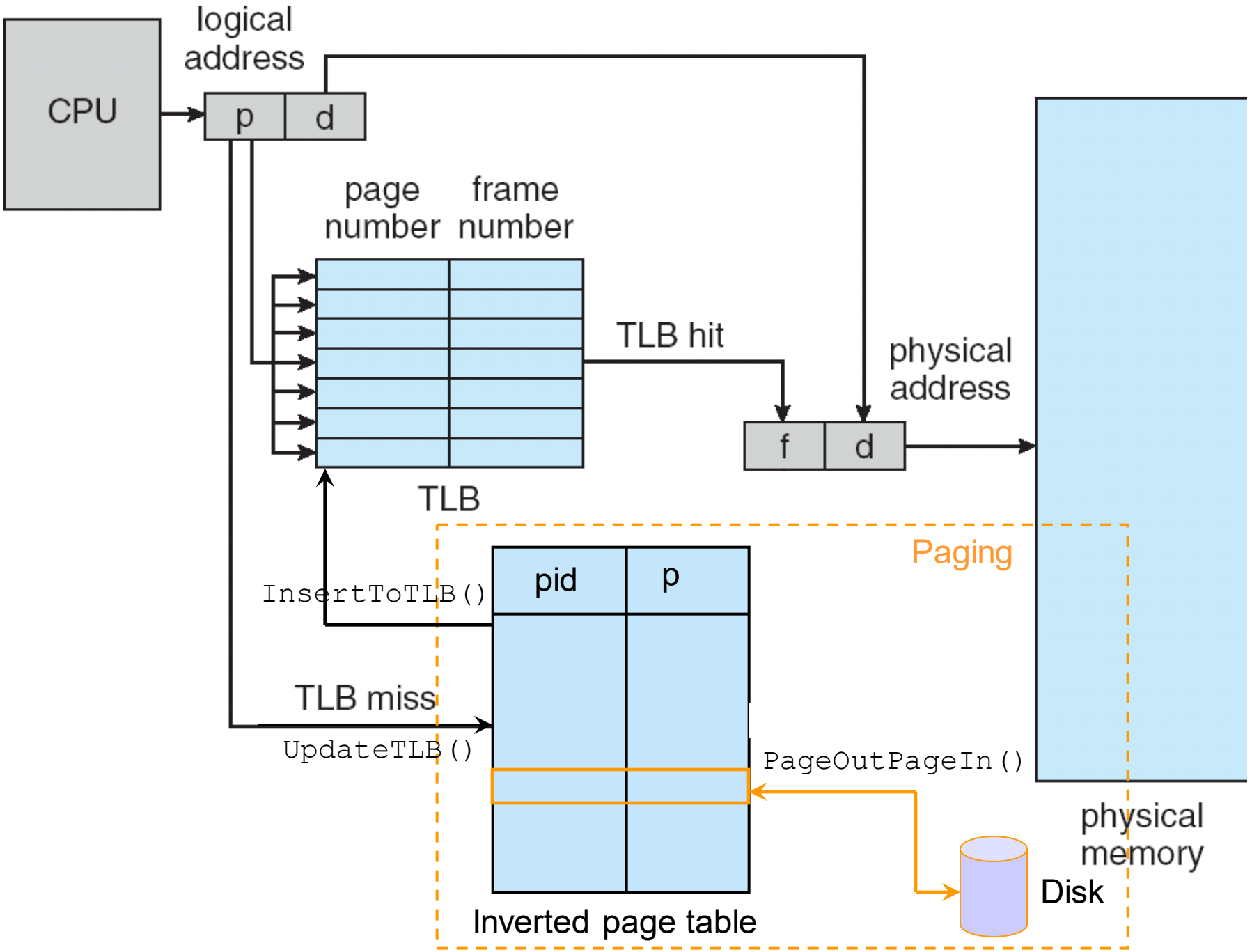
- Function: `UpdateTLB(vpn)` (defined under `vm/tlb.cc`)
  - Call `vpnToPhyPage(vpn)`
  - If `vpn` can be found in IPT (i.e., `vpnToPhyPage(vpn)` returns a valid `phyPage`), update TLB
    - Call `insertToTLB(vpn, phyPage)`
  - Otherwise, perform paging and update TLB
    - Call `insertToTLB(vpn, PageOutPageIn(vpn))`

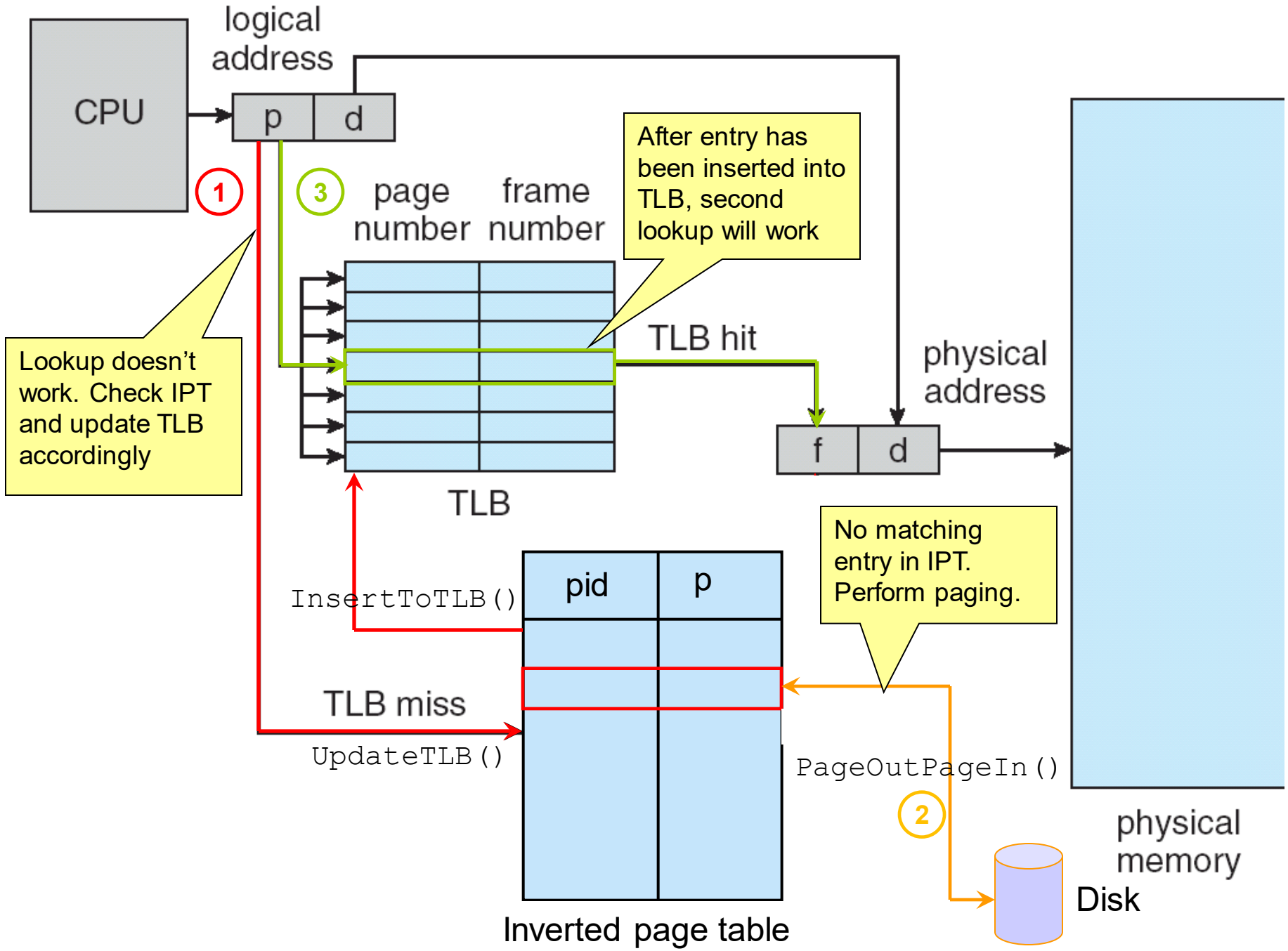

Negative case: entry not found in TLB
CPU
logical address
p
d
1
2
page number
frame number
After entry has been inserted into TLB, second lookup will work
Lookup doesn't work. Check IPT and update TLB accordingly
TLB hit
physical address
f
d
TLB
InsertToTLB()
pid
vpn
VpnToPhyPage()
TLB miss
UpdateTLB()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

Experiment 4 – Update TLB

- If VPN cannot be found in IPT, how to perform paging?

- Function: `PageOutPageIn(vpn)`
  - Determine the victim frame using the least recently used 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
InsertToTLB()
pid
p
TLB miss
UpdateTLB()
PageOutPageIn()
physical memory
Disk
Inverted page table


CPU
logical address
p
d
1
3
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
No matching entry in IPT. Perform paging.
InsertToTLB()
pid
p
TLB miss
UpdateTLB()
PageOutPageIn()
2
physical memory
Disk
Inverted page table

Discussion of Experiment 4

Experiment 4 – 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 least recently used 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!
exp3	Experiment 3, 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	NachOS kernel related to port, DO NOT CHANGE!
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 4, coding virtual memory (TLB, page replacement)**

Experiment 4 – 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 lruAlgorithm(void)`
- Binary test program for Experiment 4 is provided
  - Execute the test program:
    - `./nachos -x ../test/vmtest.noff -d`

Experiment 4 – 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].lastUsed = stats->totalTicks;
}
```

This is the set to the current tick, 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 4 – 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 4 – 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 4 – 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 4 – VpnToPhyPage()

- Return the physical page for a VPN (if it exists in the IPT)
  - `int VpnToPhyPage(int vpn) { ... }`
- IPT is realised as memory table (see `vm/ipt.h`).

- A memory table is used by the 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`
- Iterate the memory table to find the corresponding physical page entry.
- Return the index `i` for the physical page entry for which the following conditions are all true:
  - `memoryTable[i].valid`
  - `memoryTable[i].pid == currentThread->pid`
  - `memoryTable[i].vPage == vpn`
- Return -1 if no entry can be found that matches the above condition

Experiment 4 – lruAlgorithm()

- Determines which physical page should be paged out
  - `int lruAlgorithm(void)`
- Need to find the least recently used entry in the `memoryTable`.
- The last tick that the physical page is accessed, is stored in `memoryTable[i].lastUsed`.
- Search for the invalid entry from the beginning of the `memoryTable`. If there is an invalid entry, return that to be used by the virtual page.
  - `!memoryTable[i].valid`
- Otherwise, find a victim (entry with smallest `lastUsed`)
  - Return the page number

Experiment 4 – Analysis of Output

tick	vpn	pid	IPT[0]	IPT[1]	IPT[2]	IPT[3]	TLB[0]	TLB[1]	TLB[2]	Page Out
			pid, vpn, last used, valid				vpn, phy, valid			
10	0	0	0,0,0,0	0,0,0,0	0,0,0,0	0,0,0,0	0,0,0	0,0,0	0,0,0	N
13	9	0	0,0,12,1	0,0,0,0	0,0,0,0	0,0,0,0	0,0,1	0,0,0	0,0,0	N
15	26	0	0,0,12,1	0,9,15,1	0,0,0,0	0,0,0,0	0,0,1	9,1,1	0,0,0	N
20	1	0	0,0,12,1	0,9,19,1	0,26,17,1	0,0,0,0	0,0,1	9,1,1	26,2,1	N

- Analyse the output of your program and complete **Table1.csv** (record down the entries of the IPT and TLB **before** replacement and **highlight** the entry that is selected to be updated in **Table1.pdf**. **Do not highlight cells in Table1.csv and do not add any new column header to Table1.csv**)
  - For IPT entries, record:
    - pid, vpn, last used, valid
  - For TLB entries, record:
    - vpn, phy, valid
- Write down the following (By completing **Table2.csv**)
  - 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 4 – Summary

- No group effort
- **Assessment:**
  - Assessment of your implementation. Please leave your code, the output file **output.txt** as well as **Table1.csv**, **Table2.csv** and **Table1.pdf** files containing the above information in the **vm** folder for TA/Supervisor to review. Deadline is 1 week after your lab session (e.g., if lab session is from 10AM-12PM on a Monday, then deadline is 9:59AM on the next Monday).
  - Lab Quiz 2, which is an online multiple-choice quiz, will be administered through NTULearn in Week 14.

Acknowledgement

- The slides are revised from the previous versions created by Dr. Heiko Aydt and Tan Wen Jun.