Anthony Ventresque
anthony.ventresque@ucd.ie

Virtual Memory Management

School of Computer Science, UCD

Scoil na Ríomheolaíochta, UCD

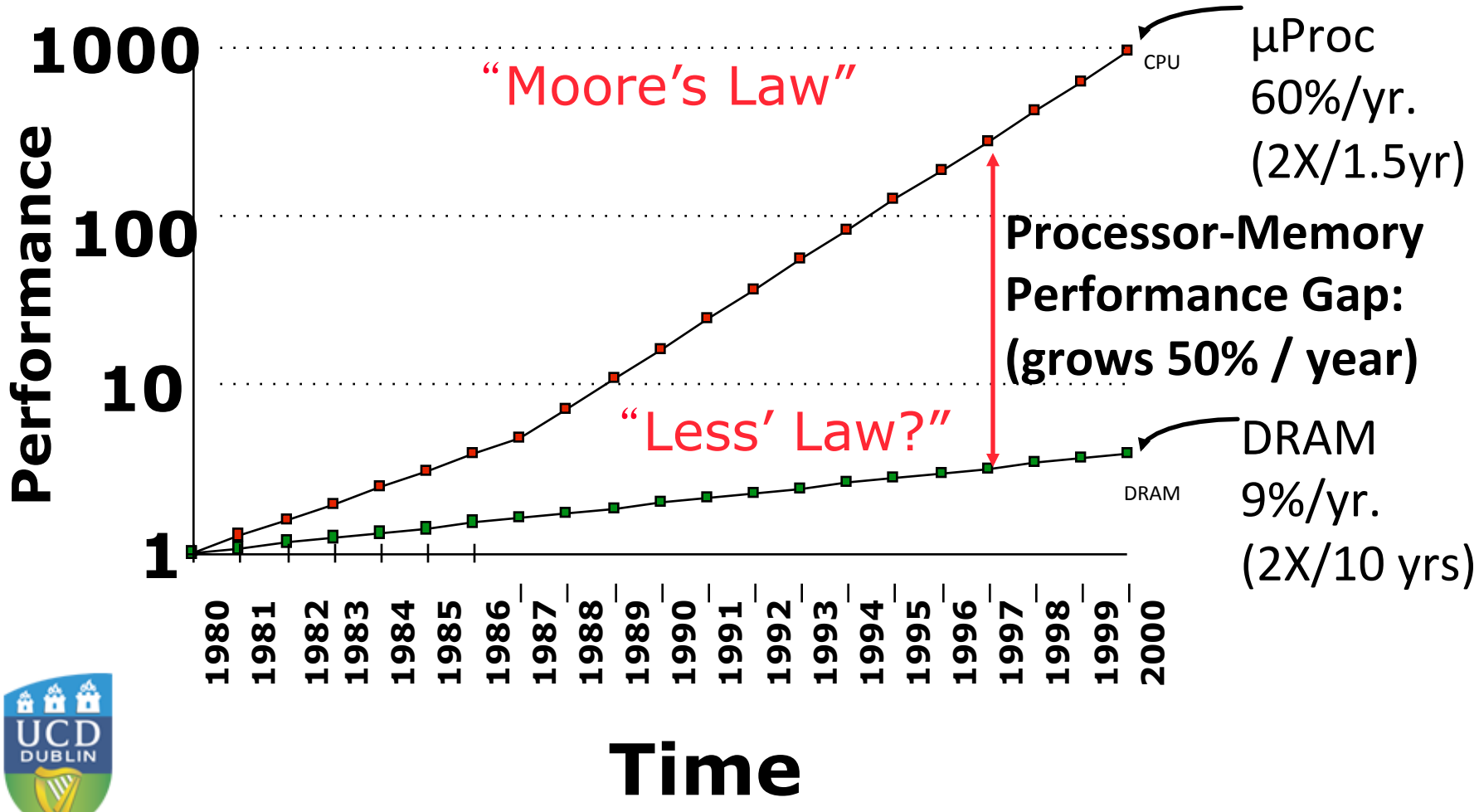
Learning Outcomes

- Understand the technique of caching and its implementation in Virtual Memory
- Understand how the principle of locality and its role in VM
- Understand the techniques of demand paging, page faults and trashing
- Understand replacement policies for pages in memory
- Understand multi-level tables and inverted page tables

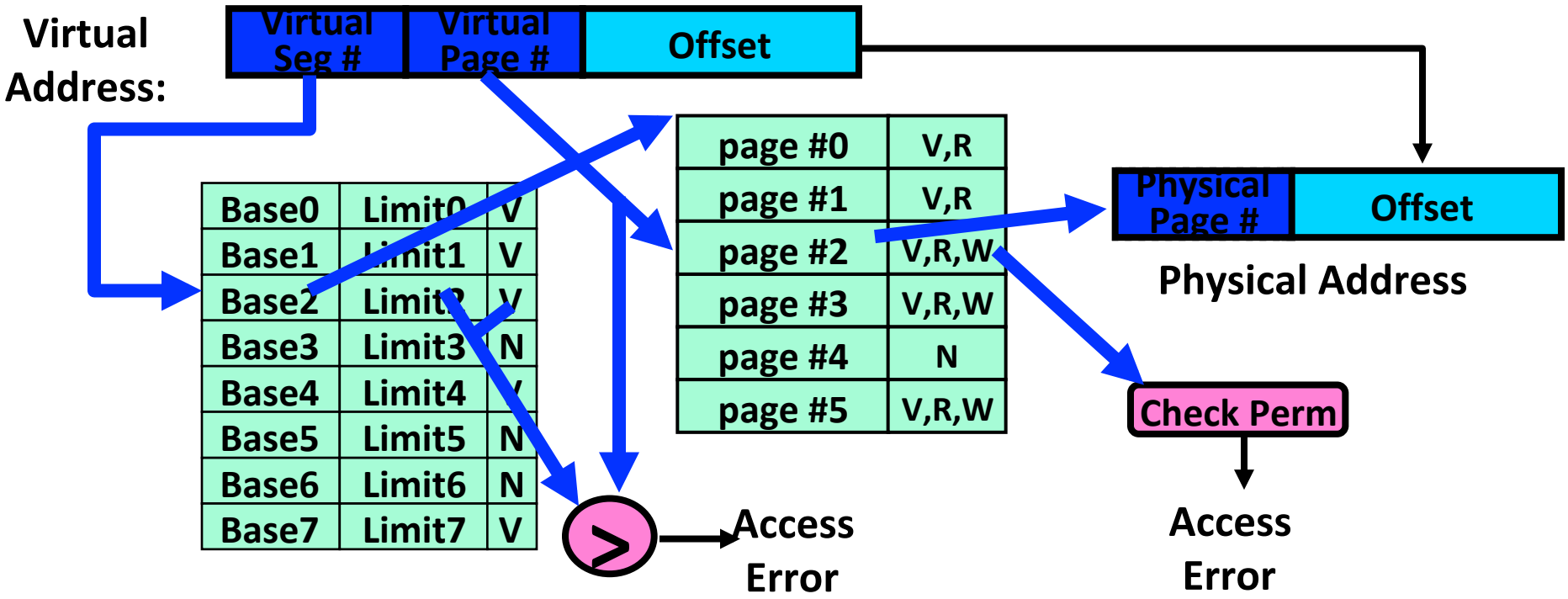
Caching Concept

- ***Cache*:** a repository for copies that can be accessed more quickly than the original
  - Make frequent case fast and infrequent case less dominant
- Caching underlies many of the techniques that are used today to make computers fast
  - Can cache: memory locations, address translations, pages, file blocks, file names, network routes, etc…
- Only good if:
  - Frequent case frequent enough and
  - Infrequent case not too expensive
- Important measure: Average Access time =
  (Hit Rate x Hit Time) + (Miss Rate x Miss Time)

Why Bother with Caching

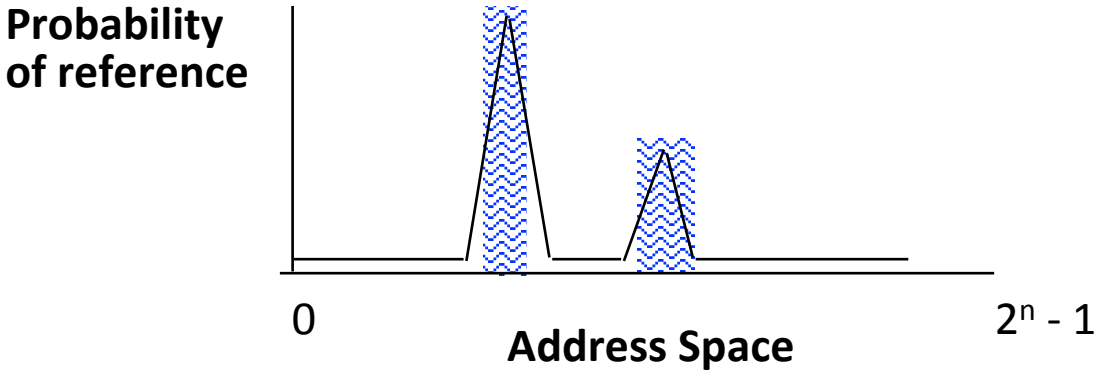


Another Major Reason to Deal with Caching

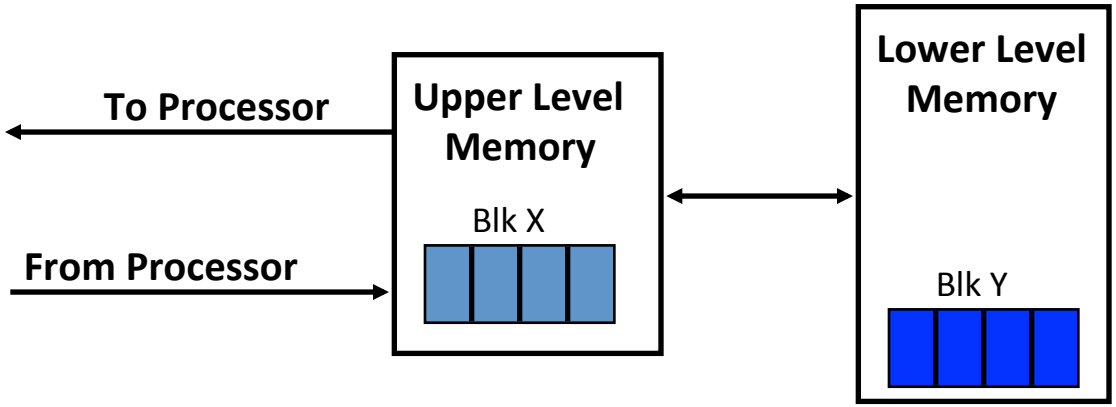


- Cannot afford to translate on every access
  - At least three DRAM accesses per actual DRAM access
  - Or: perhaps I/O if page table partially on disk!
- Even worse: What if we are using caching to make memory access faster than DRAM access???
- Solution? Cache translations!
  - **Translation Cache: TLB ("Translation Lookaside Buffer")**

Why Does Caching Help? Locality!

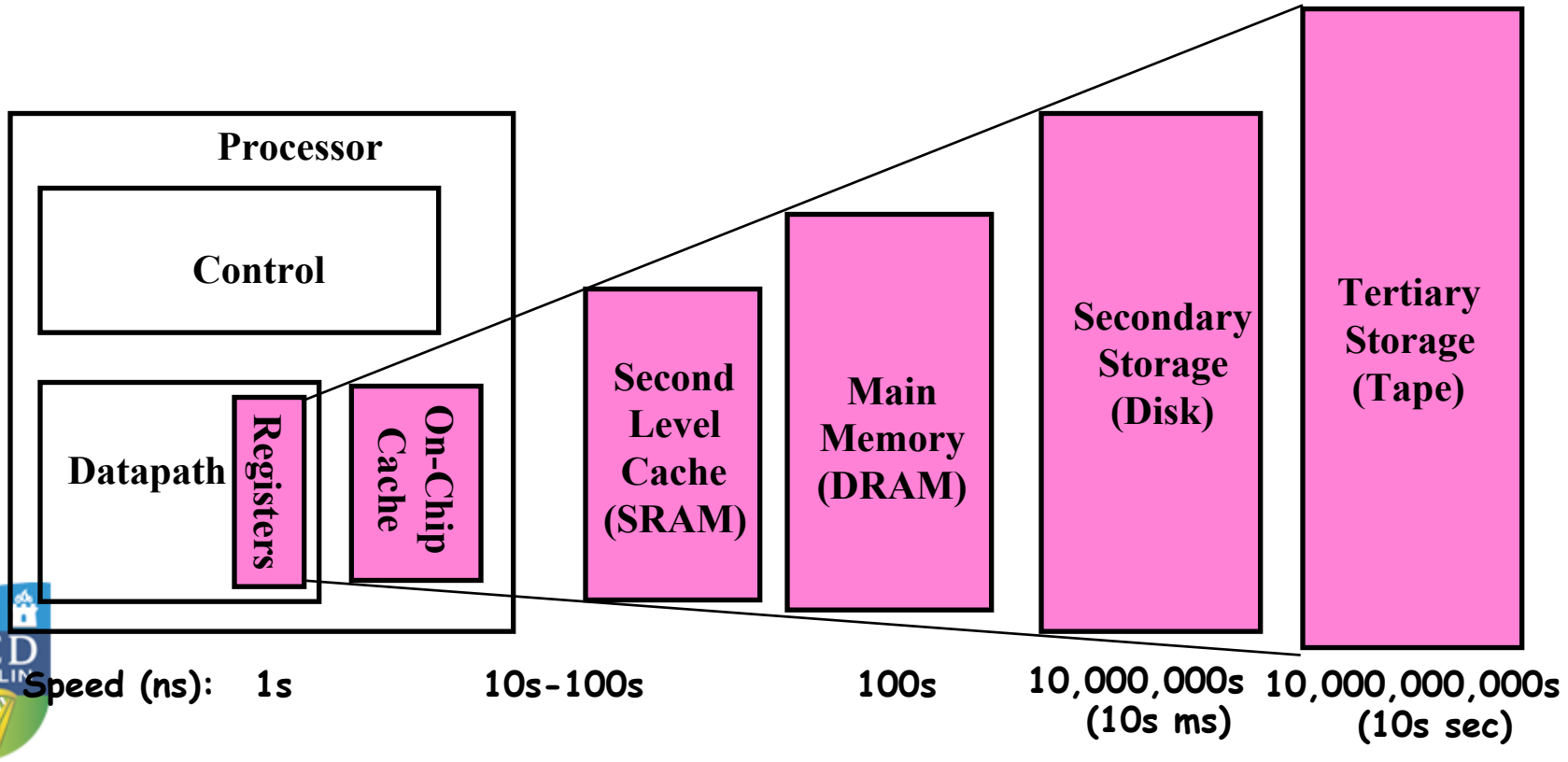


- ***Temporal Locality*** (Locality in Time):
  - Keep recently accessed data items closer to processor
- ***Spatial Locality*** (Locality in Space):
  - Move contiguous blocks to the upper levels



Memory Hierarchy in a Modern Conputer System

- Take advantage of the principle of locality to:
  - Present as much memory as in the cheapest technology
  - Provide access at speed offered by the fastest technology



Virtual Memory

- **Characteristics of paging and segmentation:**
  - The address space of a process may be broken up into pieces (pages or segments)
  - Memory references within a process are logical addresses, dynamically translated into physical addresses
  - By virtue of the page/segment table, pages/segments of a process need not be contiguously located in main memory
  - Last but not least: it is not necessary that all pages/ segments of a process be simultaneously in main memory during execution: part of the process may be disk

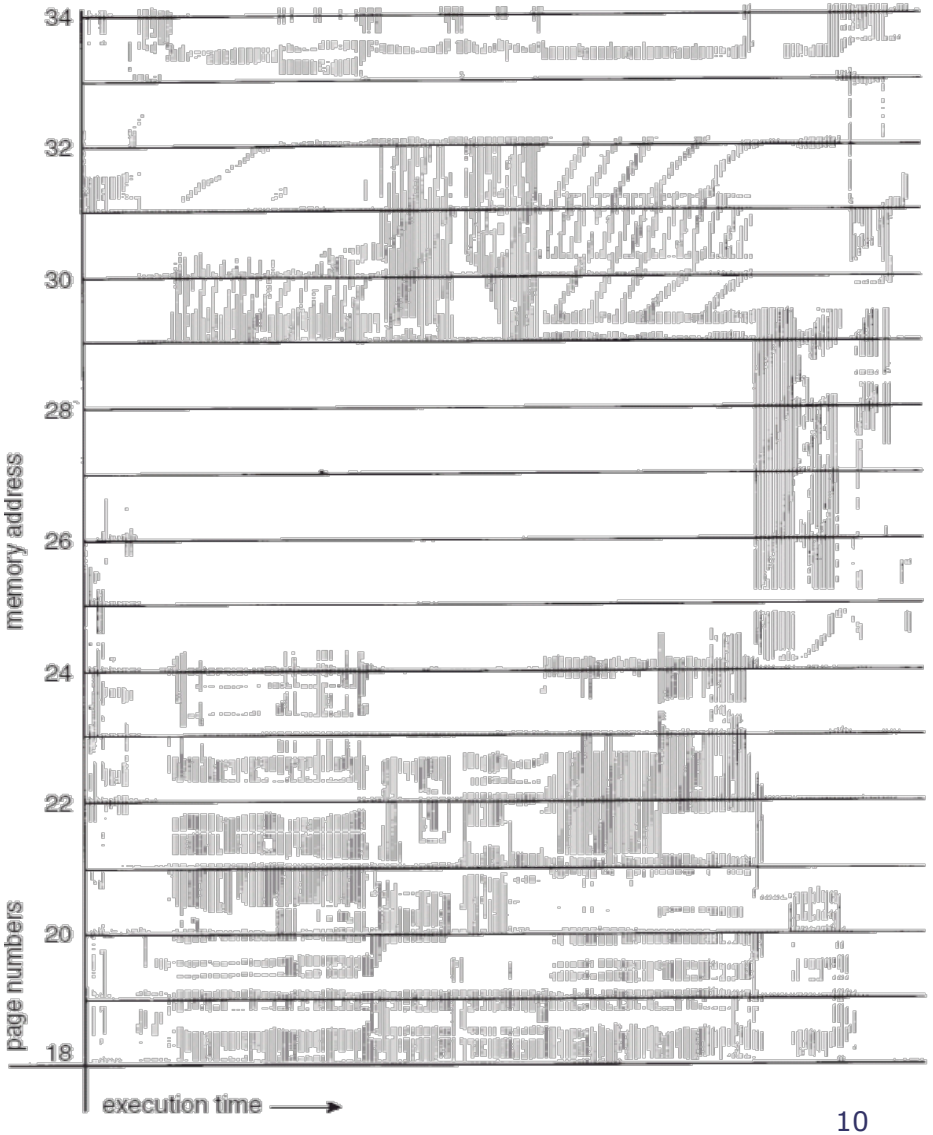
- ***Virtual memory (VM):*** illusion supported by system hardware and software that a process has a vast and linear expanse of available memory (much bigger than the main memory unit)
  - Main memory can be seen as a cache for the disk
  - Logical addresses are also called ***virtual addresses*** in the context of VM

Real Programs and VM

- The VM scheme is appealing: is it practical as well?
- An examination of a real program shows that it can usually be divided into two parts:
  1. Parts frequently needed
  2. Parts rarely or never needed:
     - Code handling unusual error conditions
     - Code handling certain options and features rarely used
     - Allocation of more memory than strictly needed (arrays, tables. . .)
- Also, memory references within a program tend to be clustered (principle of ***locality of references***)

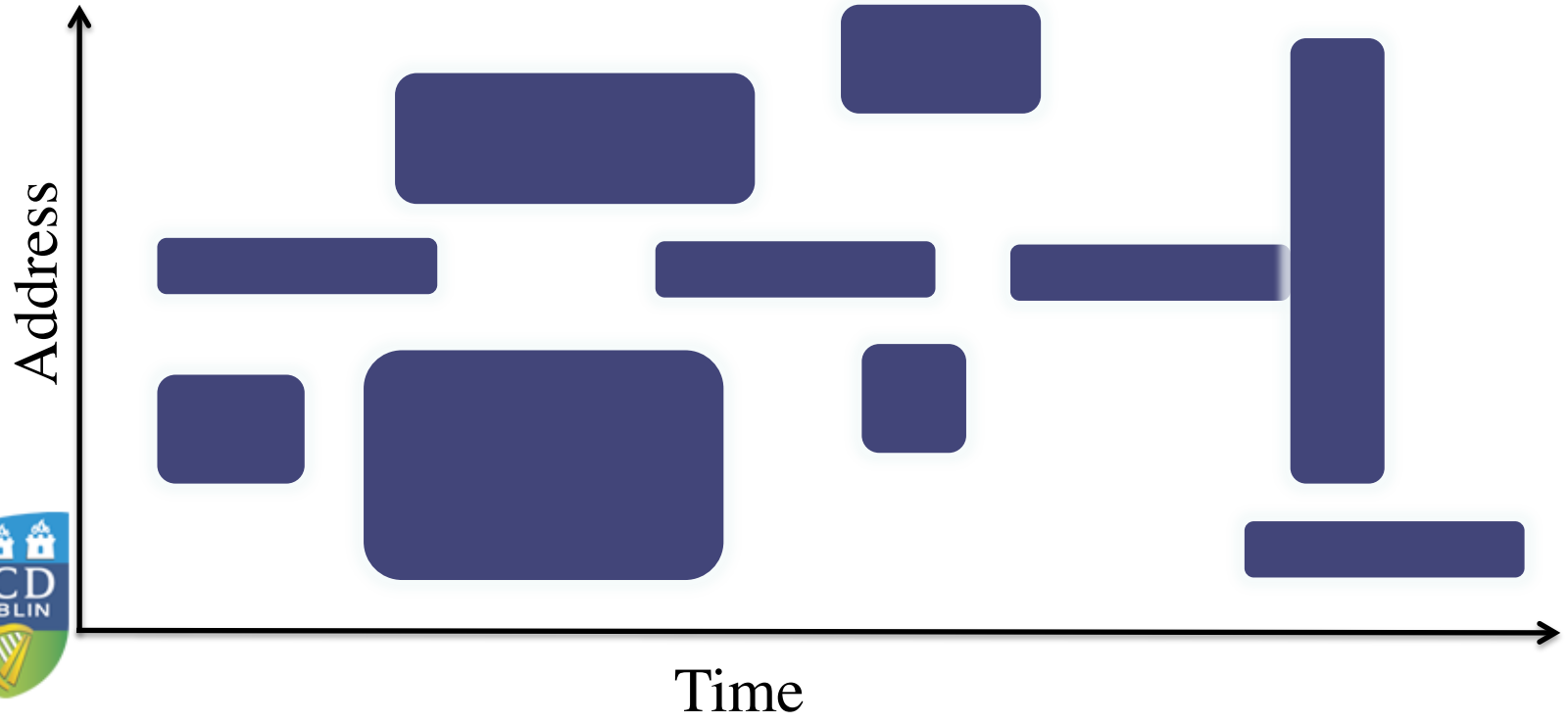
Locality in a Memory-reference Pattern

- Program Memory Access Patterns have temporal and spatial locality
  - Group of Pages accessed along a given time slice called the "Working Set"
  - Working Set defines minimum number of pages needed for process to behave well
- Not enough memory for Working Set⇒Thrashing
  - Better to swap out process?



Working Set Model

- As a program executes it transitions through a sequence of "working sets" consisting of varying sized subsets of the address space

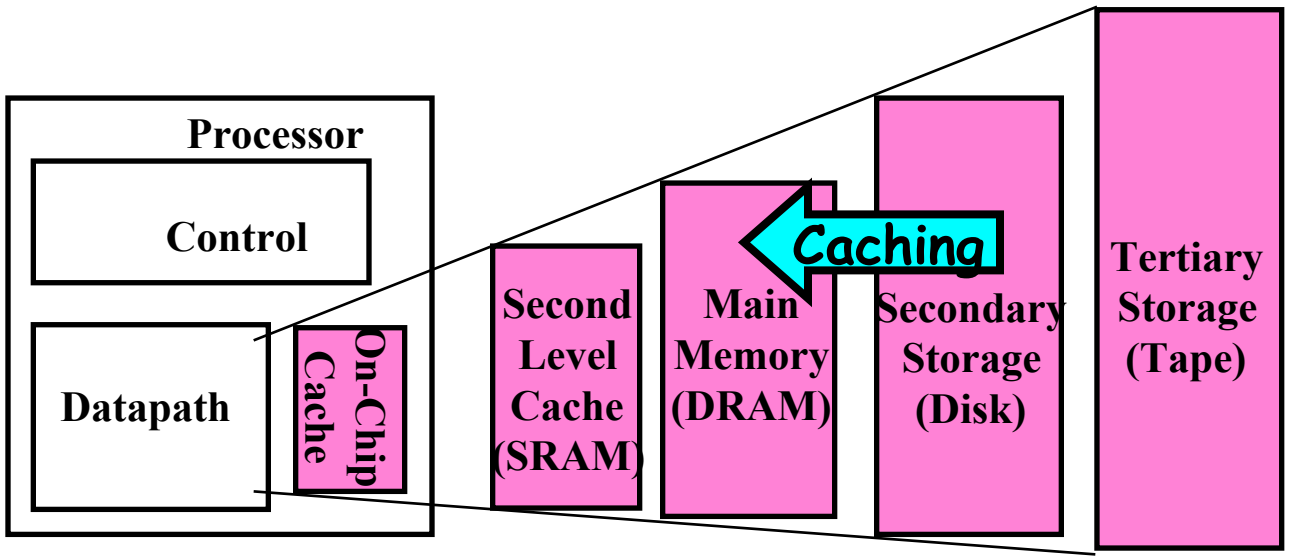


Real Programs and VM

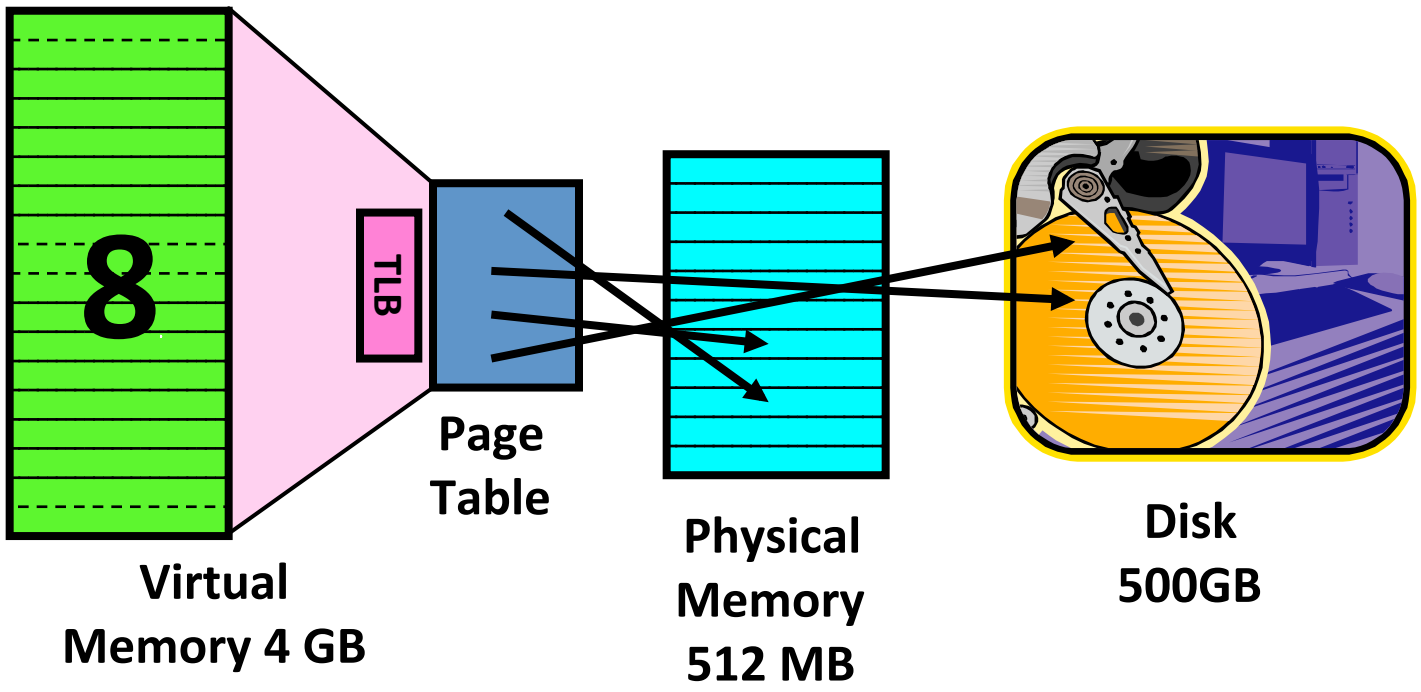
- Therefore, it is possible to make intelligent guesses about which pieces of the address space of the process will most likely be needed in the near future
  - If guesses are good, then a system using VM will perform efficiently
- If guesses are poor, the system will suffer from ***thrashing***
  - It will spend too much time swapping processes' pieces between memory and disk, rather than executing instructions
  - Disk accesses are much slower than memory accesses

Demand Paging

- Modern programs require a lot of physical memory
  - Memory per system growing faster than 25%-30%/year
- But they don't use all their memory all of the time
  - 90-10 rule: programs spend 90% of their time in 10% of their code
  - Wasteful to require all of user's code to be in memory
- Solution: use main memory as cache for disk



Illusion of Infinite Memory



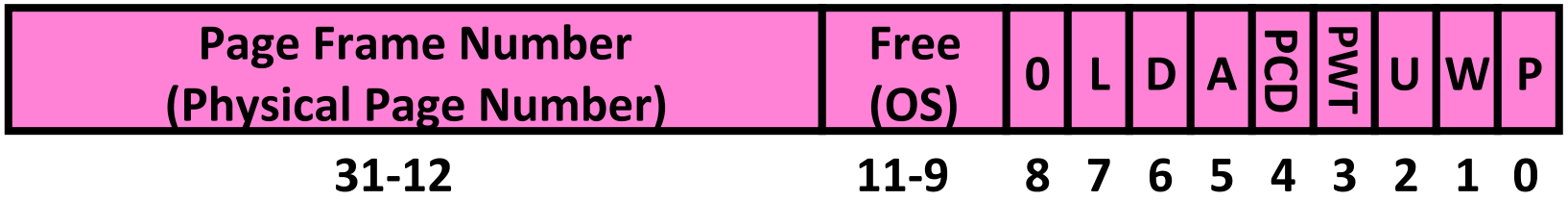
- Disk is larger than physical memory ⇒
  - In-use virtual memory can be bigger than physical memory
  - Combined memory of running processes much larger than physical memory
    - More programs fit into memory, allowing more concurrency
- Principle: ***Transparent Level of Indirection*** (page table)
  - Supports flexible placement of physical data
    - Data could be on disk or somewhere across network
  - Variable location of data transparent to user program
    - Performance issue, not correctness issue

Virtual Memory Features

- VM is commonly implemented by ***demand paging***
  - When a program is loaded, the OS brings into main memory only a few pages of it (including its starting point)
  - Further pages are then brought to memory or swapped to disk as needed
  - The ***resident set*** is the portion of the process that is in main memory at a given time
- VM system supported by
  - **Hardware:** paging mechanism, which generates ***page faults*** when pages in disk are referenced
  - **Software:** page swapping management (OS algorithm)

What is in a Page Table Entry?

- What is in a Page Table Entry (or PTE)?
  - Pointer to next-level page table or to actual page
  - Permission bits: present (sometimes "valid"), read-only, read-write, write-only
- Example: Intel x86 architecture PTE:



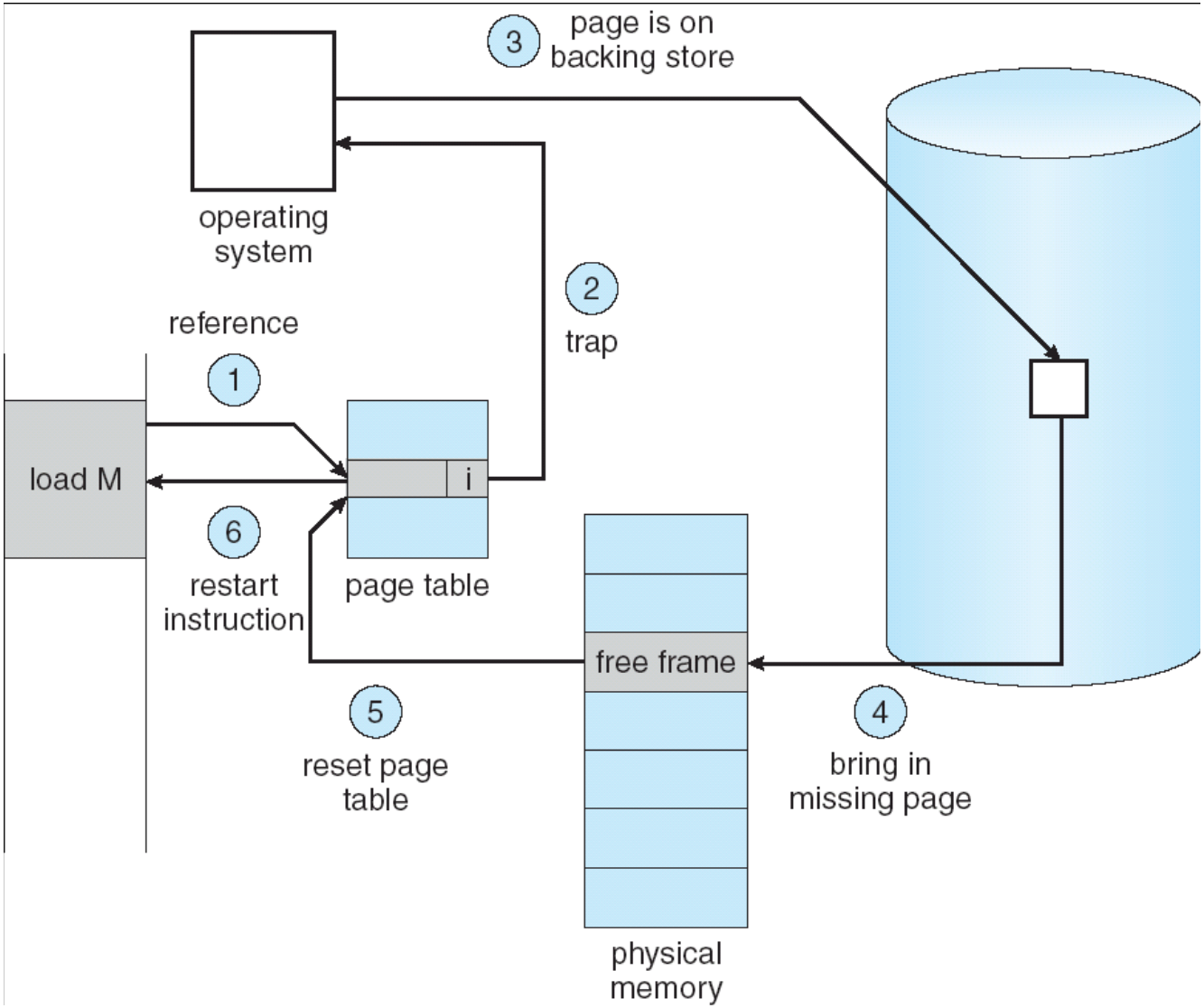
P: Present: Set to 1 if piece in main memory. **Page fault** when not
W: Writeable
U: User accessible
PWT: Page write transparent: external cache write-through
PCD: Page cache disabled (page cannot be cached)
A: Accessed: page has been accessed recently
D: Dirty (PTE only): page has been modified recently
L: L=1⇒4MB page (directory only).

Main entry: **if page in memory:** frame number
if page in disk: address in disk, or index to a table
Bottom 22 bits of virtual address serve as offset

Demand Paging Mechanisms

- PTE helps us implement demand paging
  - Present (or "Valid") ⇒ Page in memory, PTE points at physical page
  - Not Present (or "Not Valid") ⇒ Page not in memory; use info in PTE to find it on disk when necessary
- Suppose user references page with invalid PTE?
  - Memory Management Unit (MMU) traps to OS
    - Resulting trap is a "Page Fault"
  - What does OS do on a Page Fault?:
    - Choose an old page to replace
    - If old page modified ("D=1"), write contents back to disk
    - Change its PTE and any cached TLB to be invalid
    - Load new page into memory from disk
    - Update page table entry, invalidate TLB for new entry
    - Continue thread from original faulting location
  - TLB for new page will be loaded when thread continued!
  - While pulling pages off disk for one process, OS runs another process from ready queue
    - Suspended process sits on wait queue

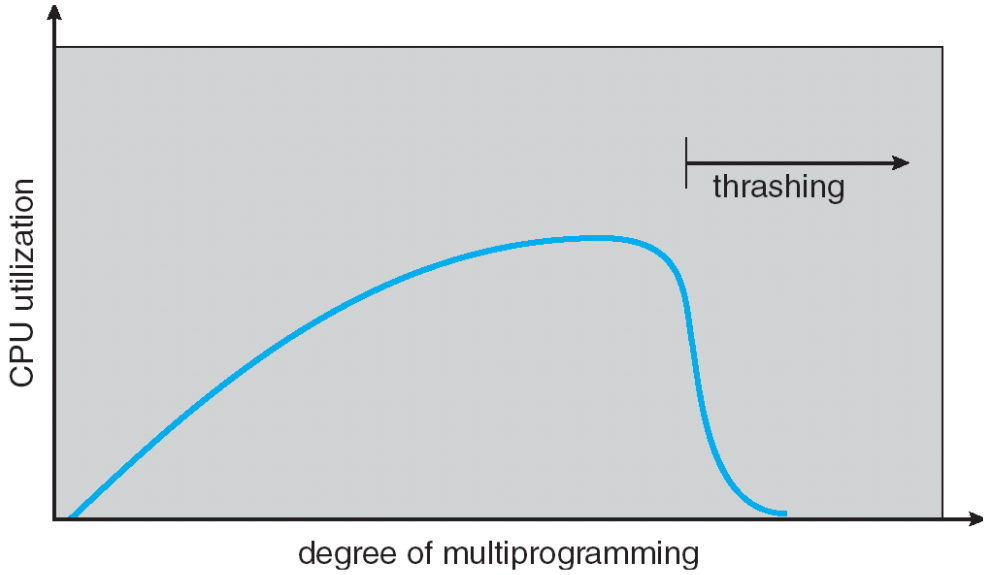
Steps in Handling a Page Fault



VM Advantages

- Programs ***not constrained by the physical memory space***
  - Can be as large as the virtual address space will allow
  - Timeline:
    - 1970s: 32-bit virtual addresses → 4 GB of memory
    - 2000s: 64-bit virtual addresses → 16 EB of memory
  - In both periods, that amount of real memory would cost millions of euro
- ***Better multiprogramming:*** more processes can be maintained in memory at any given time
  - More likely that one of these processes will be in ready state
- ***Less I/O*** is needed to load a program or to swap it
  - increase in CPU utilisation and throughput

Thrashing

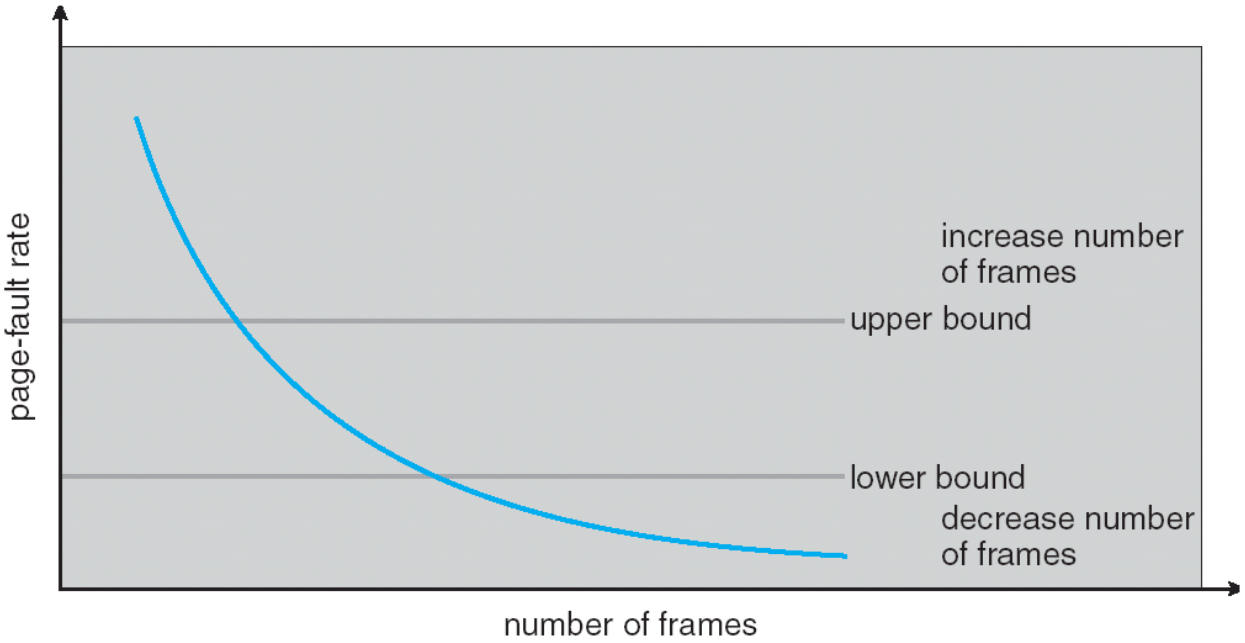


- If a process does not have "enough" pages, the page-fault rate is very high. This leads to:
  - Low CPU utilization
  - Operating system spends most of its time swapping to disk
- ***Thrashing*** ≡ a process is busy swapping pages in and out

- Questions:
  - How do we detect Thrashing?
  - What is best response to Thrashing?

Page Faults and Thrashing

- The page fault rate is a good indicator of thrashing
- In order to minimise thrashing, the OS should vary the size of a process's resident set (resident set management) according to some rules:
  - If a process is faulting heavily, allocate more frames to it
  - If faulting very little, take away some frames from it



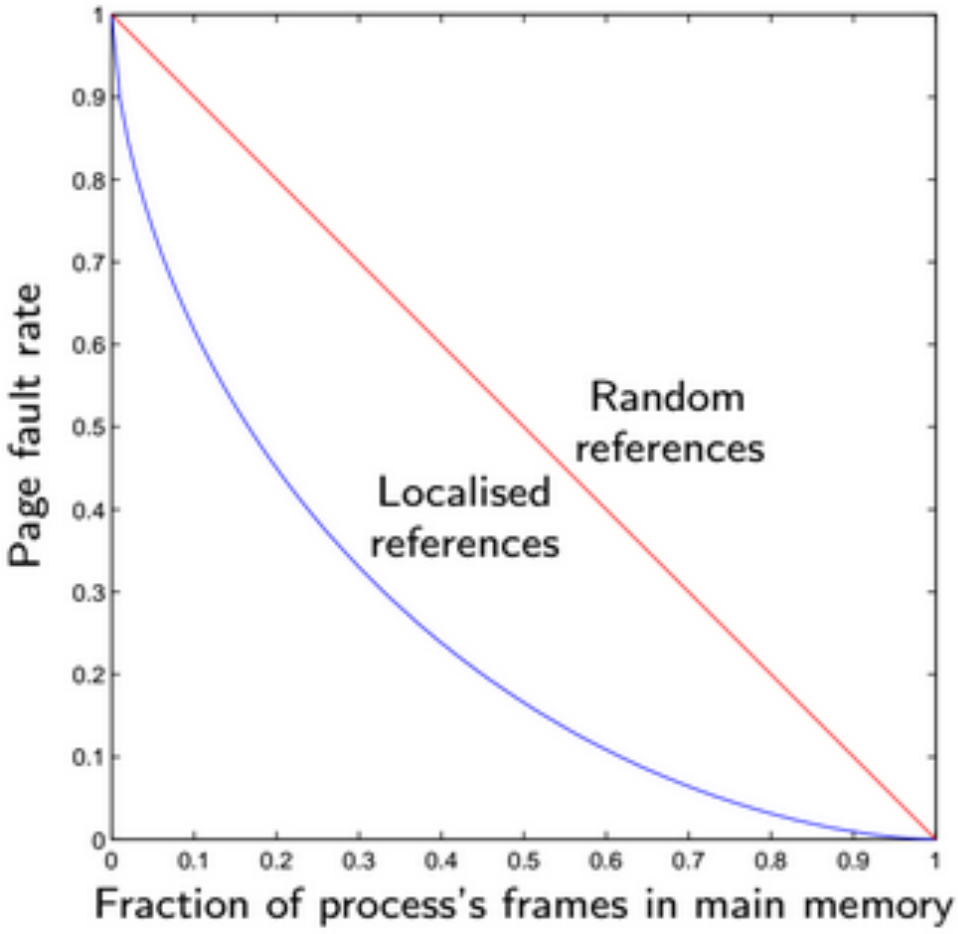
Memory Management in VM

In a system with VM, the OS must mainly deal with two memory management issues:

1. ***Replacement policy***
   - What happens when a page fault occurs and there is no free frame to swap a page in?
   - The OS must select a frame for replacement (to be swapped out to disk) when a new page must be brought in
   - Restrictions are usually placed on the page replacement policy:
     - Much of the kernel and control structures of the OS are held on locked frames (cannot be replaced)
2. ***Resident set management***
   - Choice of dynamic or static number of frames for each active process
   - Choice of replacement type allowed; examples:
     - Limited to pages of the process that caused the page fault
     - Encompassing any frame in main memory

Replacement Policy and Locality

- A good replacement policy should exploit the principle of locality of references
- If memory references were random rather than localised, we would not be able to pin down the working set efficiently
- The cost of being wrong is high: must go to disk
- Must keep important pages in memory, not toss them out



Optimal/Minimum (Min)

- Replaces page that will not be referenced forlongest period of time
- Minimum number of page faults, but impossible to implement (knowledge of future events required)
- Standard yardstick used to gauge other algorithms
- Example Suppose we have the same reference stream:
  - **A B C A B D A D B C B**

Ref:	A	B	C	A	B	D	A	D	B	C	B
Page: 1	A									C	
2		B									
3			C			D					

- Where will D be brought in? Look for page not referenced farthest in future
- MIN: 5 faults

LRU (Last Recently Used)

- Replaces the page that has not been referenced for the longest period of time
- By the principle of locality: it is likely that this page will not be referenced in the near future either
- Almost as good as the optimal policy, but difficult to implement (overheads associated to time keeping)
- Example: A B C D A B C D A B C D

Ref: / Page:	A	B	C	D	A	B	C	D	A	B	C	D
1	A			D			C			B		
2		B			A			D			C	
3			C			B			A			D

First In First Out (FIFO)

- Frames traversed as a circular buffer, triggered by replacements
- Pages are removed in round-robin style
- Rationale: a page fetched long ago may be now out of use (when main memory is composed by many frames)
- Simple to implement, but some replacements will not be good
- Example: Suppose we have 3 page frames, 4 virtual pages, and following reference stream:
  - A B C A B D A D B C B

Ref: / Page:	A	B	C	A	B	D	A	D	B	C	B
1	A					D				C	
2		B					A				
3			C						B		

- FIFO: 7 faults.
- When referencing D, replacing A is bad choice, since need A again right away

Page Size Considerations

- Page size is invariably a power of 2, but how do we select it?
- No single best answer:
  - Small page size leads to large page table
    - Example: for a VM of 4 MB (2^{22}) we would have 4096 pages of 1024 bytes, but only 512 pages of 8192 bytes
  - Memory is better used with smaller page sizes
    - On average, half of the last page is wasted
  - I/O transfer time is small compared to seek and latency, which favours larger pages
    - However locality improves with smaller page size (higher resolution)
- The historical trend is towards larger pages
  - 1990: typically around 4096 bytes; currently: 4 MB and higher

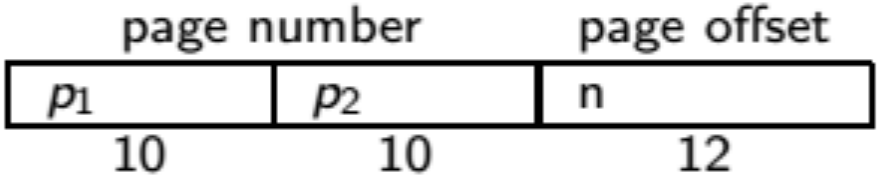
Page Table Structure

- Issue: page tables can take a big portion of memory
  - 2^{m-n} entries required for an m-bit long virtual address with n page offset bits
  - Example: with m = 32 and 4 KB (2^{12}) page size → 2^{20} = (1,048,576) entries
  - This problem is even more severe with modern 64-bit addresses

- Solutions:
  - Hierarchical paging
  - Inverted page table

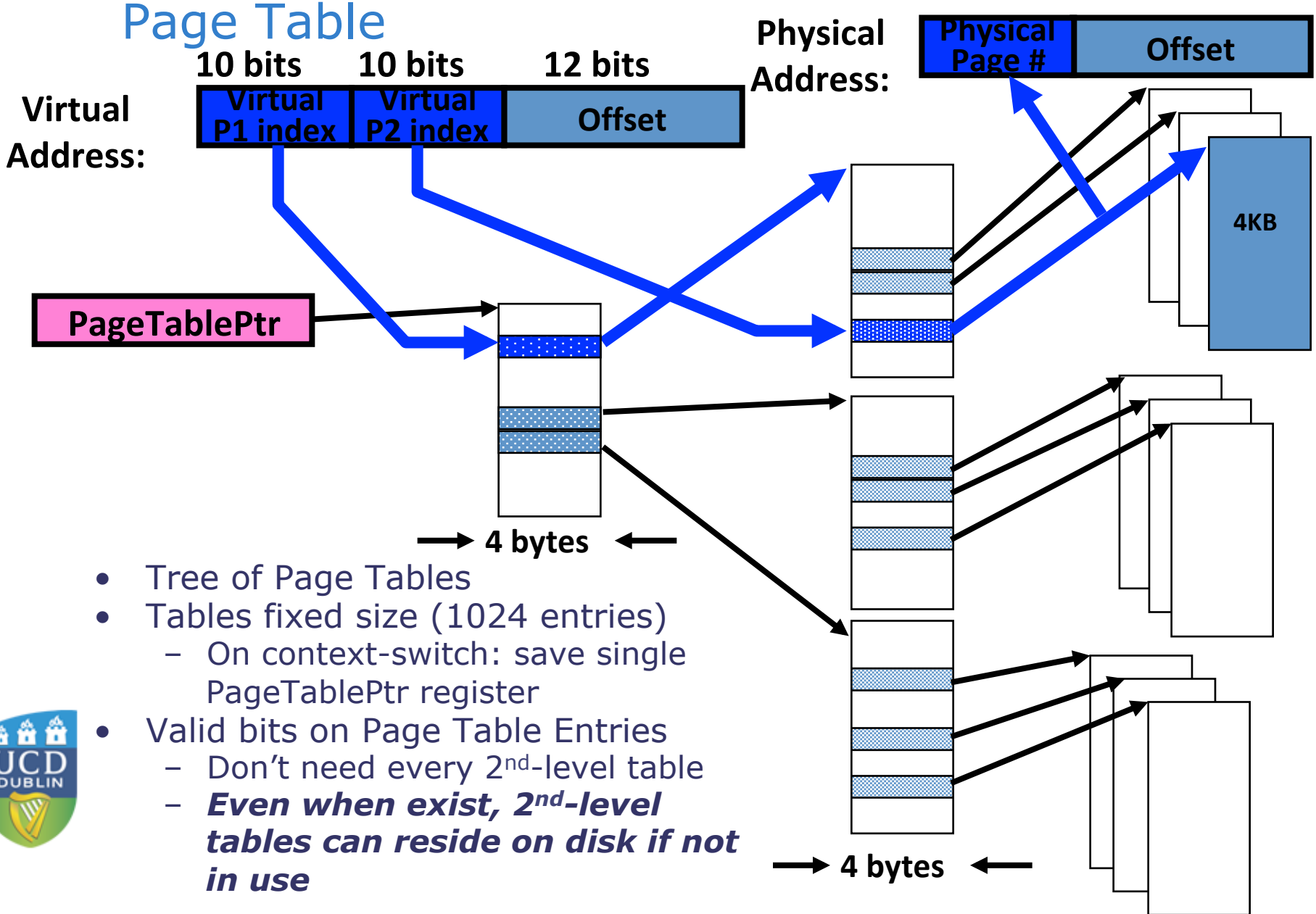
Hierarchical Paging (Two Levels)

- **Two-level page table:** the $m - n$ page number bits are divided into two sections
  1. The first points at an entry in the outer page table, which gives a frame corresponding to a page of the page table proper
  2. The second points at an entry within that page of the page table
- Example (using same values as in previous slide): $m - n = 20$ can be divided into two 10-bit numbers
  - Virtual address:



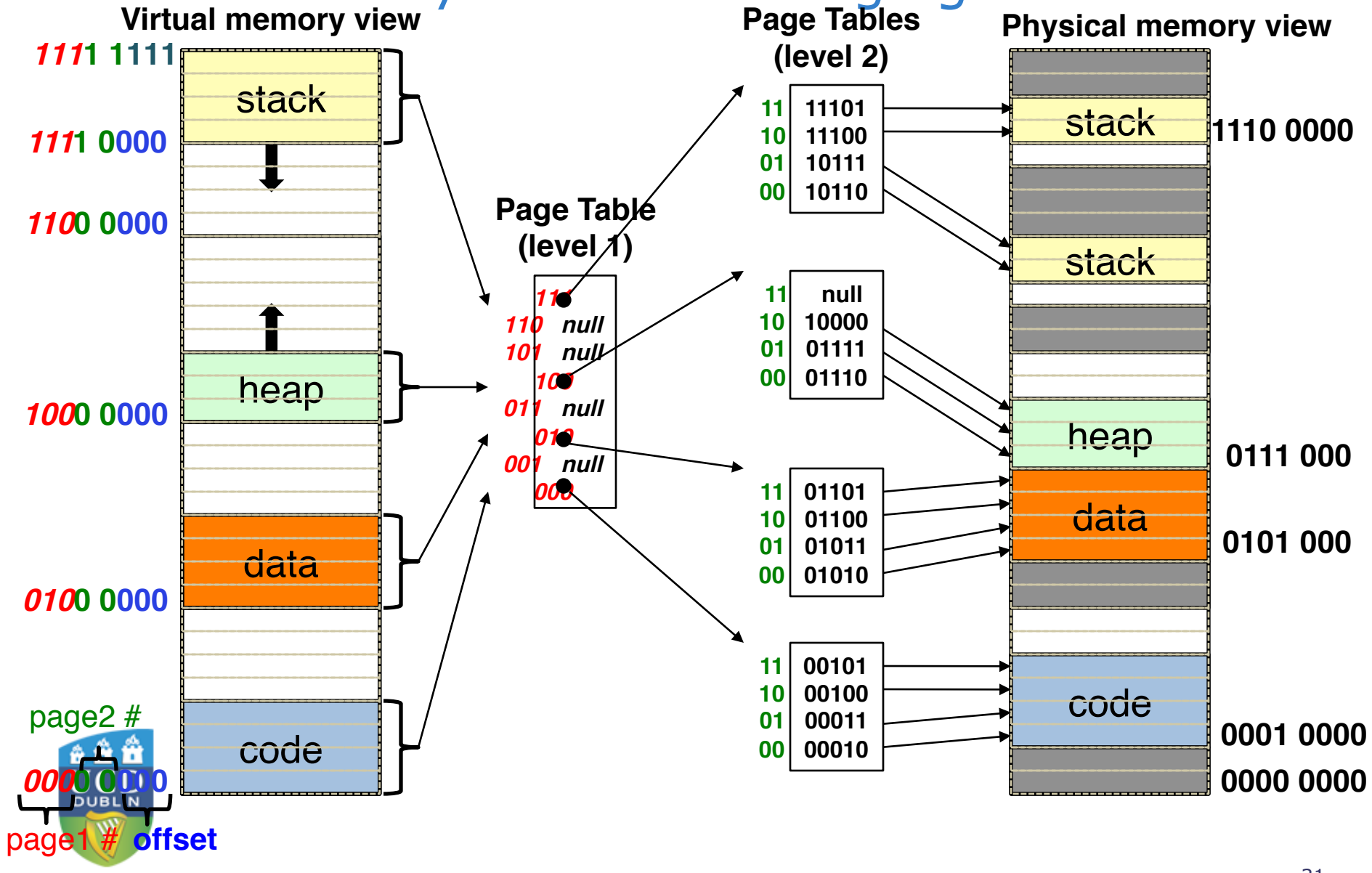
  - The 2^{10} outer page table entries fit within one 4 KB page
  - We only need one frame for the outer page table, and enough frames to reference the pages actually used by the process

Fix for Sparse Address Space: The Two-level Page Table

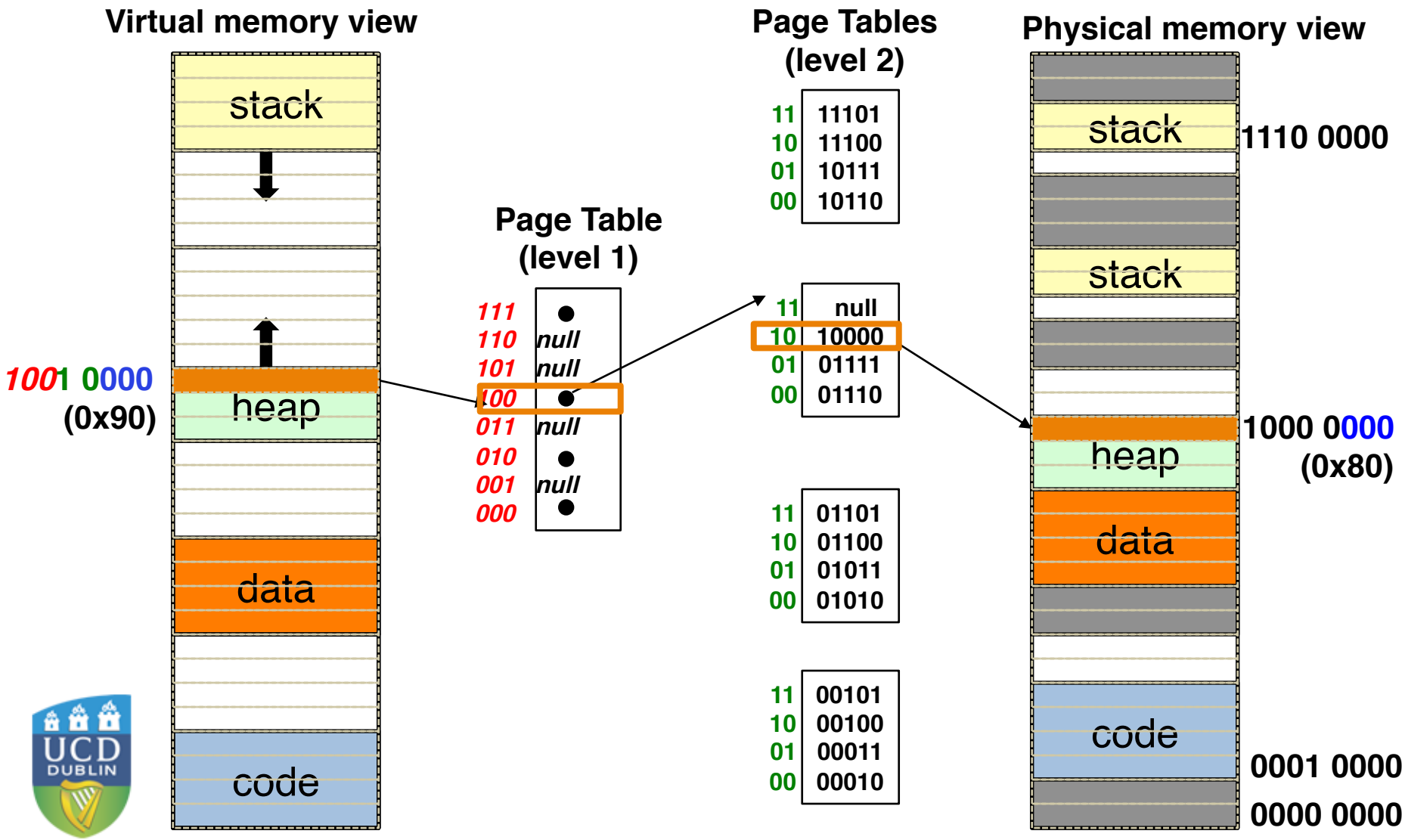


- Tree of Page Tables
- Tables fixed size (1024 entries)
  - On context-switch: save single PageTablePtr register
- Valid bits on Page Table Entries
  - Don't need every 2nd-level table
  - ***Even when exist, 2nd-level tables can reside on disk if not in use***

Summary: Two-level Paging

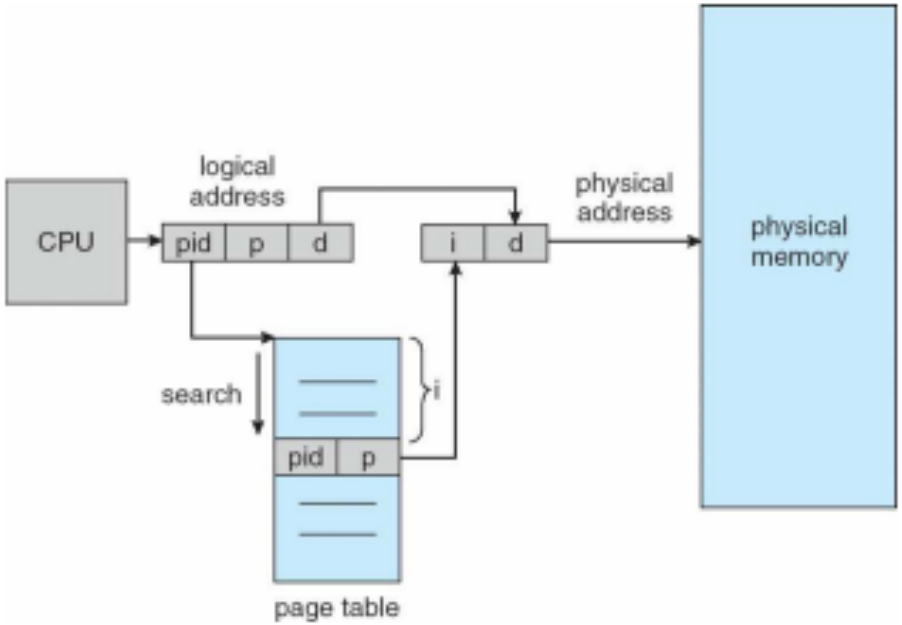


Summary: Two-level Paging



Inverted Page Table

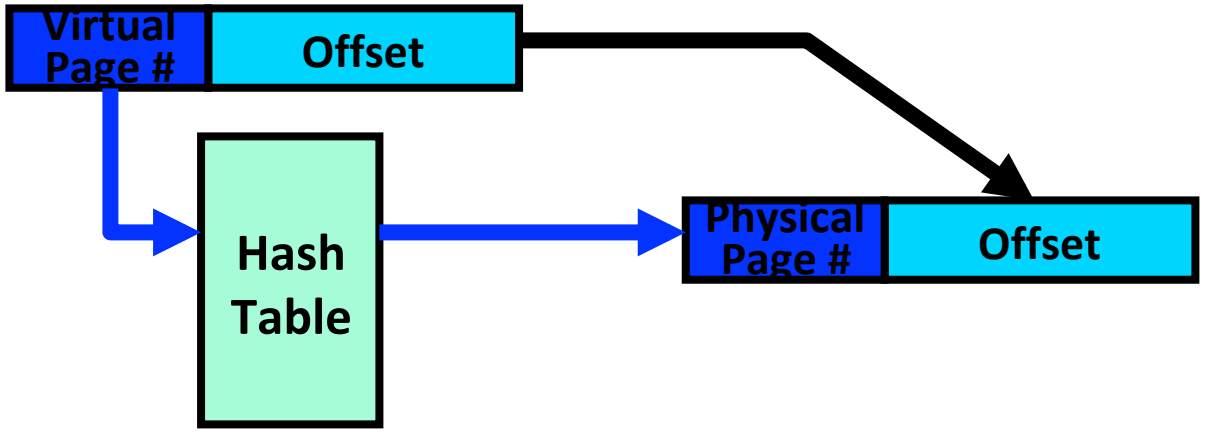
- For large *m* a good solution is an ***inverted page table***
  - the inverted page table has ***one entry per frame of physical memory*** (so there is only one)
  - each entry includes a PID and page number



- Issue: sequential search (linear inverted page table)
  - for this reason it is usually hashed to speed up access, using linked lists for collisions
  - If no match found: page fault

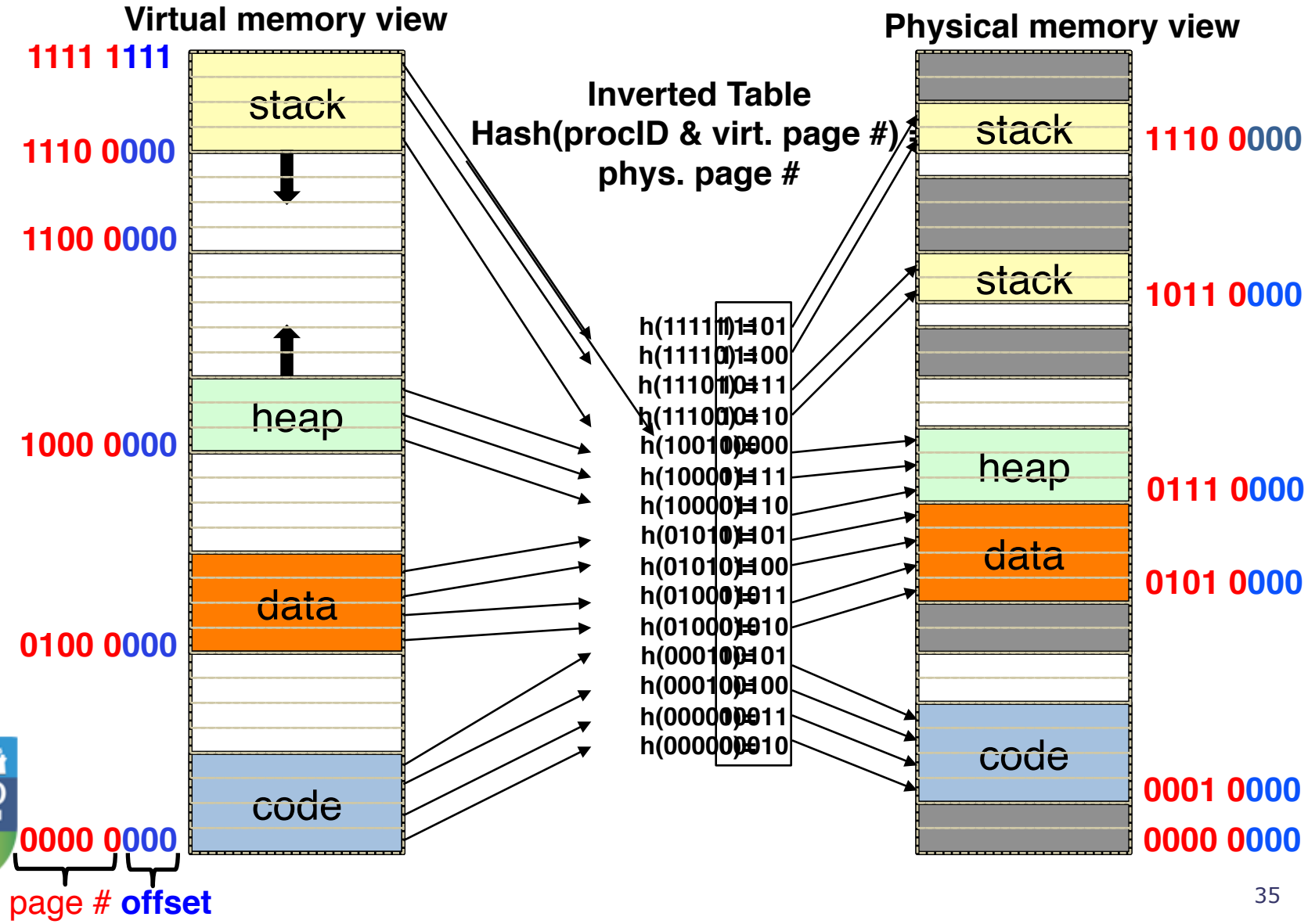
Inverted Page Table

- With all previous examples ("Forward Page Tables")
  - Size of page table is at least as large as amount of virtual memory allocated to processes
  - Physical memory may be much less
    - Much of process space may be out on disk or not in us



- Answer: use a hash table
  - Called an "Inverted Page Table"
  - Size is independent of virtual address space
  - Directly related to amount of physical memory
  - Very attractive option for 64-bit address spaces
- Cons: Complexity of managing hash changes
  - Often in hardware!

Summary: Inverted Table



Address Translation Comparison

	Advantages	**Disadvantages**
Simple Segmentation	Fast context switching: Segment mapping maintained by CPU	External fragmentation
Paging (single-level page)	No external fragmentation, fast easy allocation	Large table size ~ virtual memory Internal fragmentation
Paged segmentation	Table size ~ # of pages in **virtual memory**, fast easy allocation	Multiple memory references per page access
Two-level pages		
Inverted Table	Table size ~ # of pages in **physical memory**	Hash function more complex

Conclusion

- **Cache:** A repository for copies that can be accessed more quickly than original
- **Virtual Memory:** Illusion supported by system hardware and software that a process has a vast and linear expanse of available memory
- **Principle of Locality:** Program likely to access a relatively small portion of the address space at any instant of time.
  - **Temporal Locality, *Spatial Locality***
- VM is commonly implemented by ***demand paging***
- **Working Set:** Set of pages touched by a process recently
- **Resident Set:** Portion of process that is in main memory at a given time
- **Thrashing:** A process is busy swapping pages in and out
  - Process will thrash if working set doesn't fit in memory
  - Need to swap out a process

Conclusion (cont'd)

- A good replacement policy should exploit the principle of locality of references

- Replacement policies
  - **FIFO:** Place pages on queue, replace page at end
  - **MIN:** Replace page that will be used farthest in future (optimal)
  - **LRU:** Replace page used farthest in past
- Multi-Level Tables
  - Virtual address mapped to series of tables
  - Permit sparse population of address space
- Inverted Page table
  - Size of page table related to physical memory size