# High Performance Computing
# Lecture 14

Matthew Jacob

Indian Institute of Science

# Page Fault

- Situation where virtual address generated by processor is not available in main memory
- Detected on attempt to translate address
  - Page Table entry is invalid
- Must be `handled' by operating system
  1. Identify slot in main memory to be used
  2. Get page contents from disk
  3. Update page table entry
- Data can then be provided to the processor

# Page Fault Handler

1. Identify slot in main memory to be used
2. Get page contents from disk
3. Update page table entry

- It must keep track of the available, unused physical pages, maybe in a free list
- What if the free list is empty?
  - i.e., all main memory physical pages are already mapped to virtual pages
  - The page fault handler must then identify a page to be replaced (evicted) from main memory

# Page Replacement Policies

- Question: How does the page fault handler decide which main memory page to replace when there is a page fault?
  - How important is this decision?
  - In the worst case, the policy could always replace the page that is going to be accessed by the processor next
    - Each of these would require copying the virtual page from hard disk to main memory

# Aside: Disk Access Speed

- We saw that there is a speed disparity of about 2 orders of magnitude between Processor (nsec) and Main Memory (~100 ns)
  - Recall: nano $10^{-9}$
- Hard disk
  - Remembers things by the state of magnetic material
  - Disk is a mechanical device: motors rotating a firm plate coated with magnetic material
  - Aside: Computer noises
  - Reading a page from hard disk could take -msecs (milli:$10^{-3}$) if not longer
  - i.e., $10^4$ times slower than main memory!

# Page Replacement Policies

- Question: How does the page fault handler decide which main memory page to replace when there is a page fault?
  - How important is this decision?
  - In the worst case, the policy could always replace the page that is going to be accessed by the processor next
  - So, the OS page fault handler code must be written based on a realistic model of how programs behave with respect to memory

# Page Replacement Policies

## Principle of Locality of Reference

- A commonly believed/seen program property
- If memory address ***A*** is referenced at time ***t***, then it and its neighbouring memory locations are likely to be referenced in the near future

Temporal Locality of reference

Spatial Locality of reference

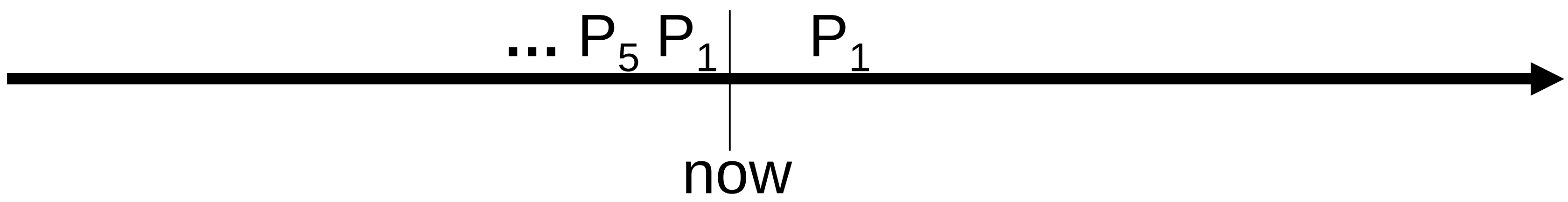
# Locality of Reference

- Based on your experience, why do you expect that programs will display locality of reference?

| | **Same address (temporal)** | **Neighbours (spatial)** |
|---|---|---|
| **Instructions** | Small loop<br>Function | Sequential code<br>Loop |
| **Data** | Local variable<br>Loop index | Stepping through array |

# Page Replacement Policies

- For a program that displays good locality of reference what would be a good page replacement policy?




A page fault occurs on reference to page $P_x$

Which page should be replaced from memory to make space for page $P_x$?

Candidates: $P_1$, $P_2$, $P_3$,…$P_{n,}$ all the pages in main memory

Pick from them the page that was referenced least recently

# Least Recently Used (LRU) Policy

- Keep track of when each page was last used
  - With a timestamp
  - LRU page: the one with the smallest timestamp
  - Requires a large number of comparisons
- Or, keep track of the stack of recently used pages
  - LRU page: at the bottom of the stack
  - Stack must be updated on every memory access
- So, LRU might be too expensive in practise