

ECE7995

(3) Basis of Caching and Prefetching --- Locality

What's locality?

- **Temporal locality** is a property **inherent** to programs and independent of their execution environment.
 - Temporal locality: the data that be requested in a program is likely to be requested again soon.
 - This locality can be improved by restructuring programs or using a locality-aware compiler.
 - This locality serves as a fundamental principle for caching.
 - ✓ This locality was born from efforts to make virtual memory systems work.
 - ✓ But the locality **alone** cannot lead to a high performance (details in the next slide)
 - ✓ We need to be able to manage the locality.

What's locality? (cont'd)

- **Spatial locality** is determined by both data layout in both logical and physical levels.
 - Spatial locality: a data is likely to be requested soon after the data close to it is accessed (“close” can be in logical or physical sense).
 - This locality can be improved by re-organizing data according to their access patterns, at logical level in a program or at physical level in their storage.
 - This locality serves as an essential basis for prefetching.
 - ✓ This locality should be considered together with temporal locality (Details in other lectures)

Virtual Memory: a Story about the discovery of Locality Principle*

*: locality refers to temporal specifically in the rest of the lecture.

- A **hierarchy** that consists of main memory (RAM) and secondary memory (hard disk)
- Page: **the unit** of data storage and transfer (A page is transferred to memory from disk when any data contained in the page is requested).
- By automating the page transfers, system designers can **simulate** a (much) larger (virtual) main memory within a small real one.
 - Programs no longer need to specify **physical address** of a datum in order to access it (they actually don't know where the datum is, in memory, in disk, or both).
 - Instead, programs give data name (a logical address in its program space). System is responsible to **translate** the logical address of a datum into physical address, **check** page table to see if the page that contains the datum is memory, and if not (a page fault), automatically **transfer** the page into the memory.
 - However, if memory is full, systems have to **decide** which in-memory page should be replaced to make room for the faulted → **Page replacement algorithm**.
 - “Virtual memory” dramatically improved programming productivity, but **performance is at risk!**

Making Virtual Memory Efficient

- Disk is **way too much lower** than disk (more than 1000x), imagine how slow a virtual memory is if most data requests incur page fault!
- Page hit: a requested data is found in memory; otherwise, an **expensive** page fault (also called miss)!
- Page fault ratio is determined by program's access pattern and system's **replacement decision**.
 - **Page fault ratio** is the ratio between number of page fault and number of accesses.
 - On system side, page replacement algorithm is the key to **minimizing page faults**.
 - An **optimal** algorithm is to replace page that will not be used again for the longest time. (Can you prove this?!)
 - You never know what happens in the future (which pages will be accessed sooner than others), BUT you can **guess**.
 - It was observed that algorithms that favor recently used pages perform better than others → **Least Recently Used (LRU)** replacement algorithms.
 - This observation hinted that **recently used pages are likely to be used again soon**. (Could you conceptualize the phenomenon?)

Thrashing: Another Painful Challenge with VM

- Multiprogramming VM: multiple programs **concurrently** run in a virtual memory.
 - **Multiprogramming level**: the number of concurrently running programs.
- Thrashing is manifested as a sudden **collapse** of throughput as the multiprogramming level rose to a certain **threshold**.
 - A thrashing system spent most of its time resolving page faults and **little** running the CPU.
 - Thrashing *kills* the entire system.
 - Thrashing seems to have nothing to do with the choice of replacement algorithm.
 - Thrashing is an outcome of fierce competition of **physical** memory.
- So, is virtual memory still a viable technique??

Discovery and Adoption of Locality Principle

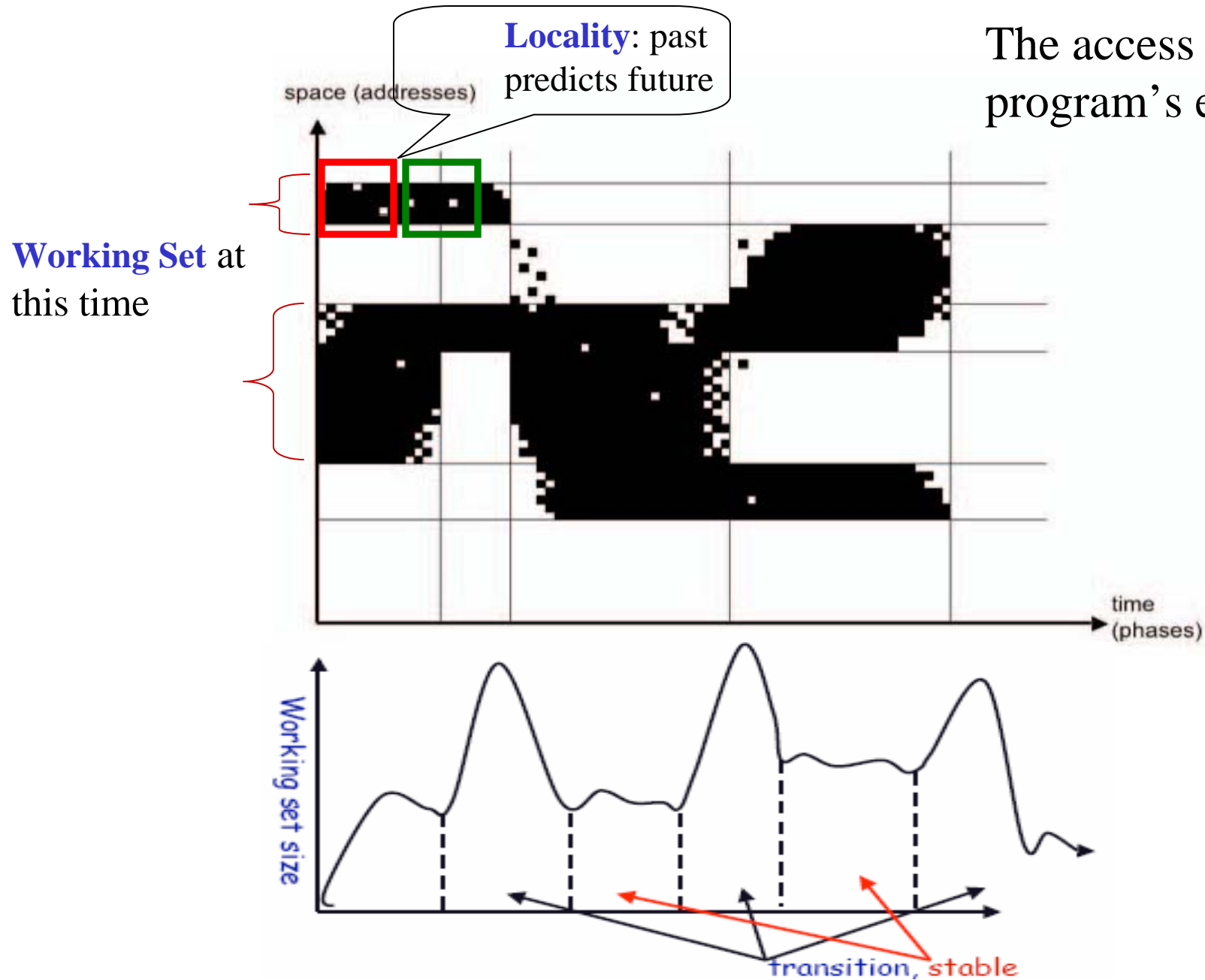
- A speculation: is it possible to make variable memory partitions, each for one process, to avoid the **fighting** among multiple memory-intensive processes?
- But who dictates how much memory a particular process should receive?
 - Program's **access behaviors** determines the number of pages it needs for a **smooth** run at a particular time.
 - This memory demand is **intrinsic** to programs, and may **change** with time.
 - If we allocate this number of **demanded page** to hold the pages in memory, then the job is done.
 - This set of pages is called **working set** (WS).

Discovery and Adoption of Locality Principle (Cont'd)

- But how to measure a program's working set?
 - A more formal definition: WS is the set of pages used during a fixed-length **sampling window** in the immediate past;
 - The window has to be in the virtual time of a process --- time as measured by the number of memory references made by the process
 - WS at time t with a sampling window size T is $W(t, T)$.
 - Questions: if a process is not scheduled in the last T , either because other process is running on the processor, or because this process is stuck on the I/O operations, is it true that $W(t, T) = 0$? (Hint: WS is a program's intrinsic property, and independent of scheduling policy or congested I/O path.)
 - The WS idea is based on the locality principle (The pages seen in the backward window are highly likely to be used again in the immediate future.)

Locality and Working Set

The access in a program's execution



Why there Exists Locality

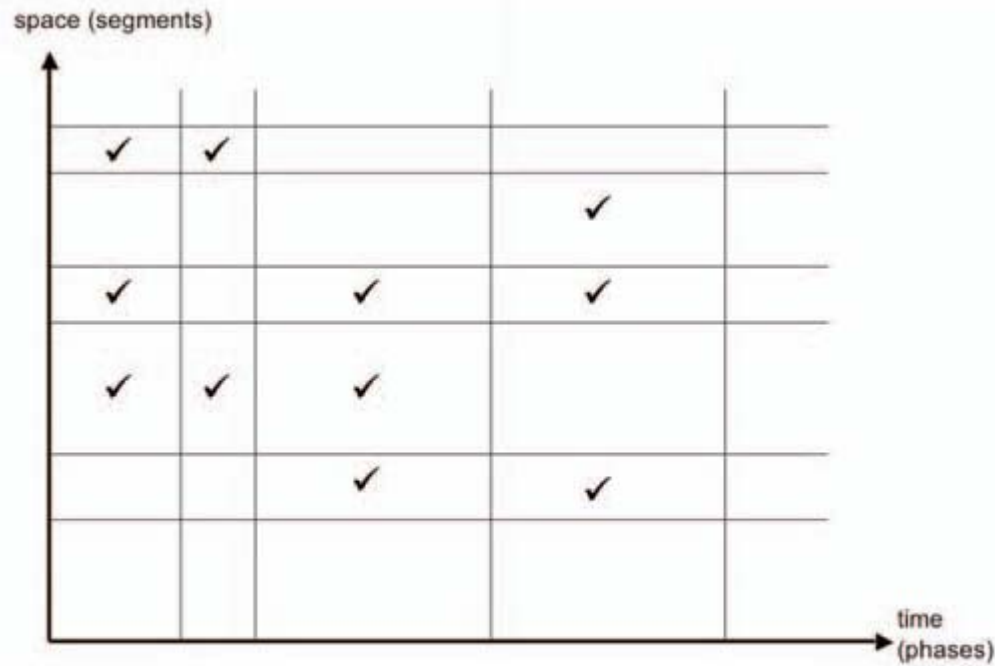
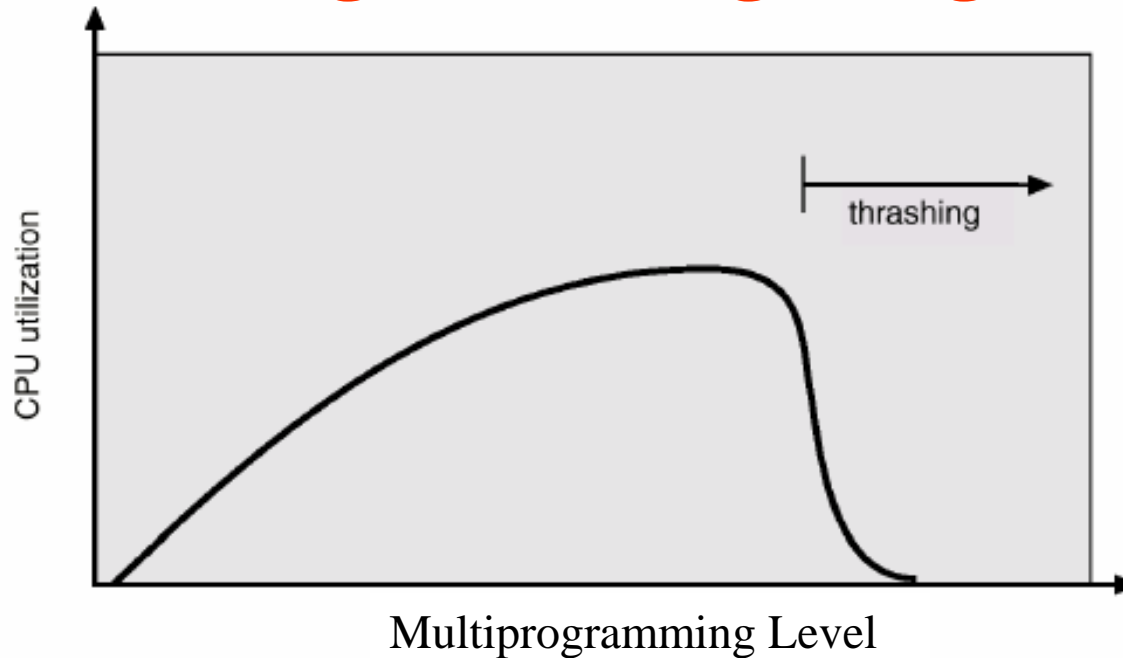


Figure 1. Locality-sequence behavior diagrammed by programmer during overlay planning.

- (1) Temporal clustering due to looping and executing within modules
- (2) Spatial clustering due to related data grouped together, such as array.

Avoiding Thrashing using WS



- Increasing the level is supposed to increase CPU utilization, but being too high makes the memory unable to hold the aggregate WS, which leads to thrashing.

- Solutions:

- ✓ Admission control: accept a new program only when its expected WS is larger than currently available memory
- ✓ Load control: swapping out and suspend program to virtual memory when thrashing happens

Applications of Locality Principle

- The locality is applied in ever-widening circles:
 - Processor cache
 - TLB for accelerating address translation
 - memory buffer
 - Web browser cache
 - Search engine for relevant response to queries
 - Content Delivery Network
-

Acknowledgements: many contents covered in the lecture are adapted from “*The Locality Principle*” by Peter Denning

Further reading for your presentations:

Song Jiang, Xiaodong Zhang. **Token-Ordered LRU: An Efficient Page Replacement Policy and Implementation in Linux Systems.** In *Performance Evaluation*, 60 (1-4): 5--29, May 2005.

Song Jiang, Xiaodong Zhang. **TPF: a System Thrashing Protection Facility.** In *Software - Practice & Experience*, 32 (3): 295--318, March 2002.