A Distributed Deep Memory Hierarchy System for Content-based Image Retrieval of Big Whole Slide Image Datasets

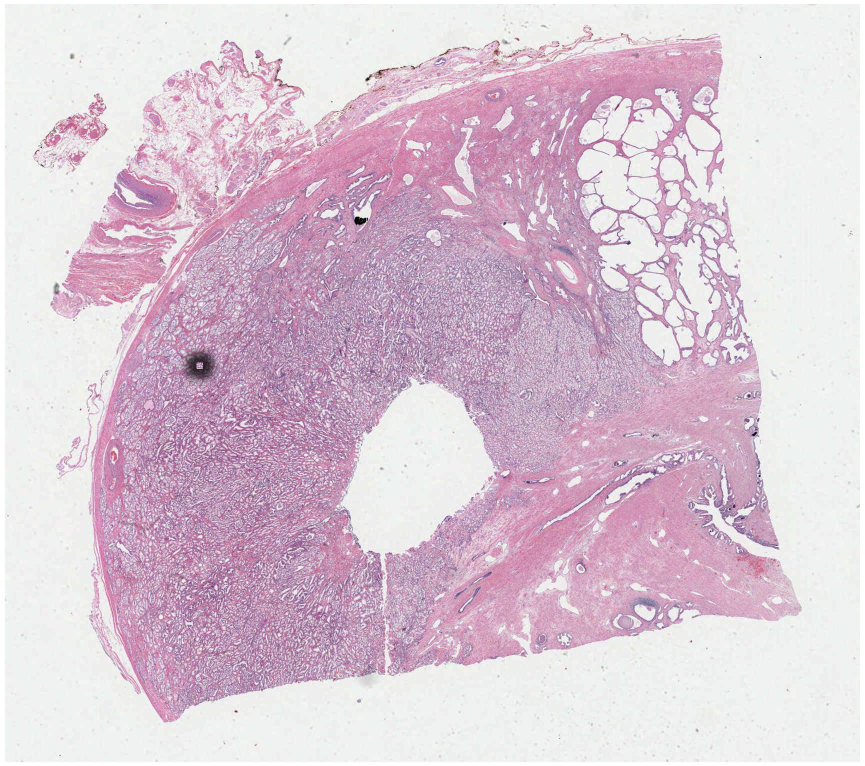
Esma Yildirim, PhD
Department of Mathematics and Computer Science
Queensborough Community College of CUNY

November 18 , **2019**

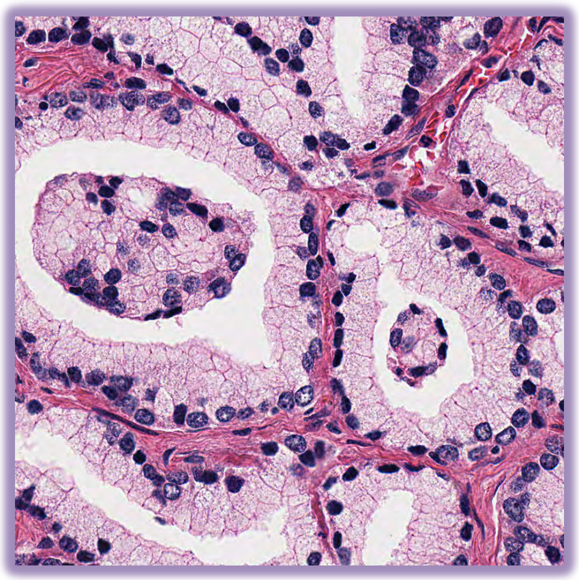
Histological Big Data: Whole Slide Images

- A whole slide image (WSI) is a multiple-resolution multi-Giga-pixel image produced by a whole slide scanner used in pathological diagnostics

Highest resolution(Level 0): 100912 x 94774 pixel image



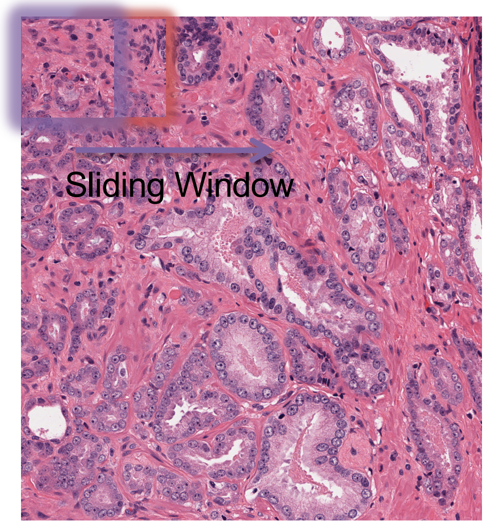
A 1024x1024 pixel tile among 9206 others



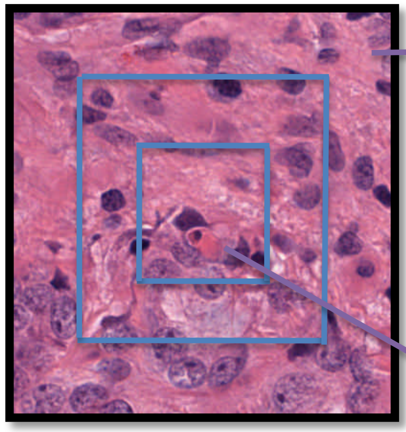
CBIR's Role in Cancer Treatment

- *Content-based Image Retrieval (CBIR)* methods allow pathologists search for *region of interests (ROIs)* containing cancerous patterns efficiently in a vast amount of whole slide image dataset.
  - Identification of regions exhibiting similar characteristics in either the same specimen or across disparate specimens
  - Draw comparisons among patient samples in order to make informed decisions for likely prognosis and most appropriate treatment regimens.

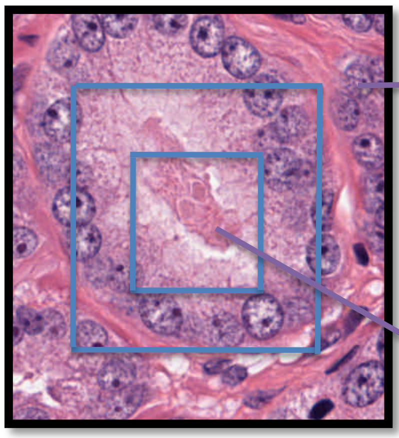
CBIR Workflow: *Step 1 – Coarse Searching*



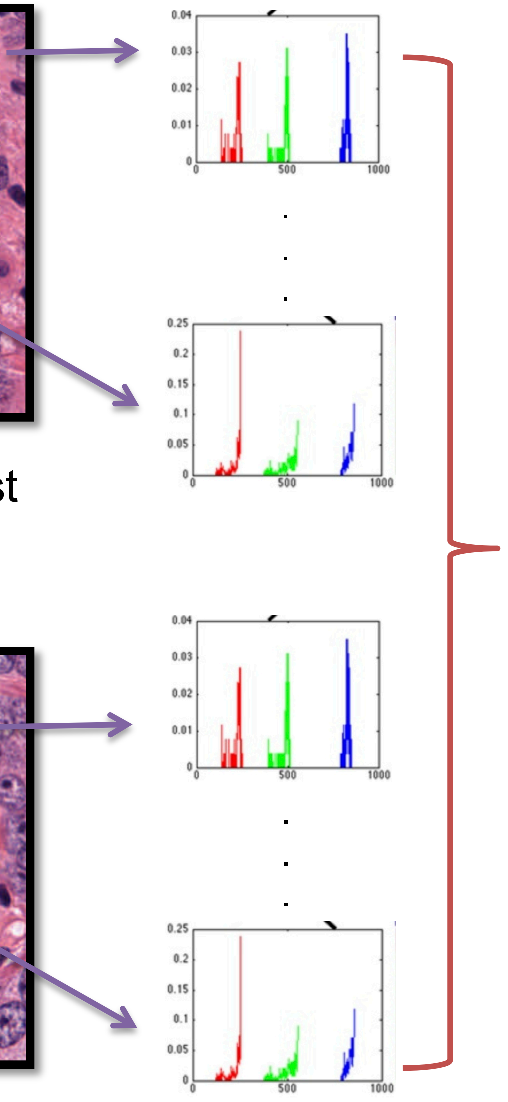
Example tile



Region of Interest



Query patch



Color histograms of rings

Distance between feature sets

↓

Sort by distances

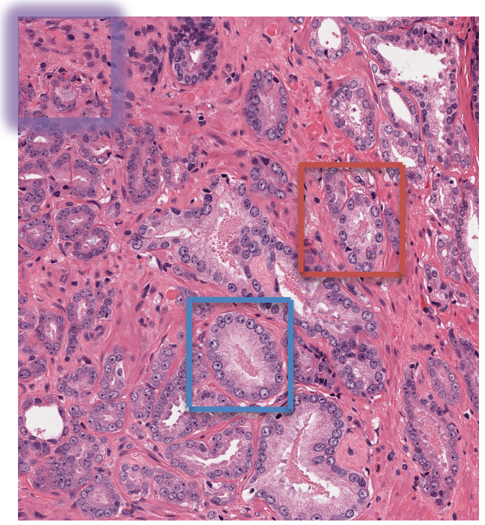
↓

Crop 10% of best results

↓

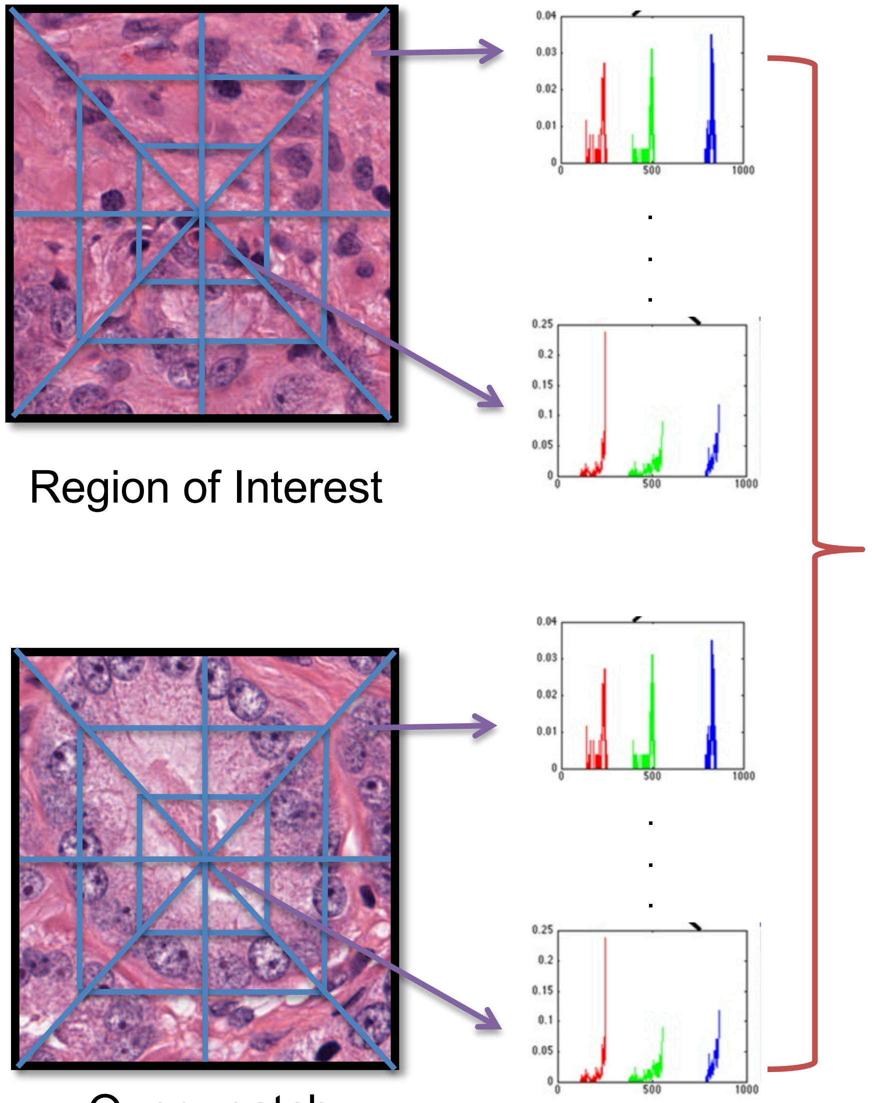
Output coordinates of Top 10% ROIs

CBIR Workflow: *Step 2 – Fine Searching*



Example tile

Out of 10% top results of Coarse Searching



Region of Interest

Query patch

Color histograms of segments

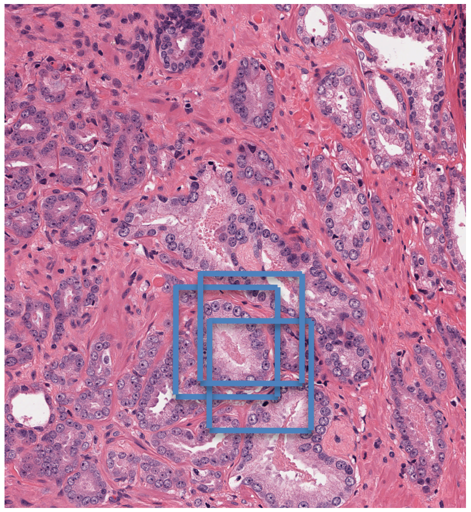
Distance between feature sets

Sort by distances

Crop 10% of best results

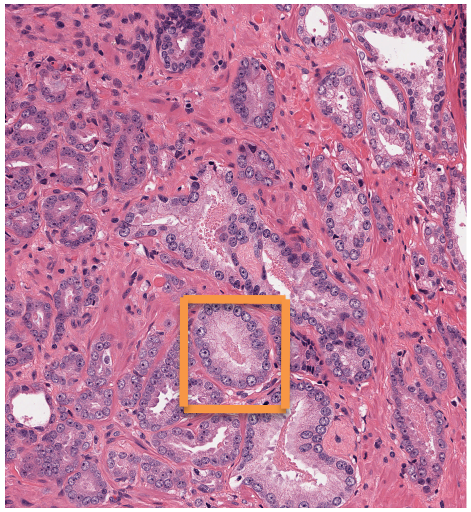
Output coordinates of Top 10% ROIs

CBIR Workflow: *Step 3 – Clustering*



Example tile

Out of 10% top results of Fine Searching



Overlapping patches are combined

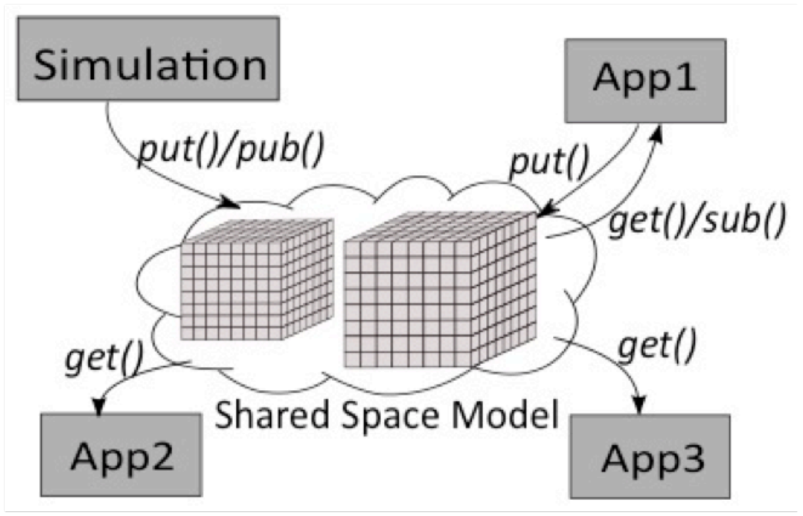
$$\begin{tikzcd}\text{Fine searching} \arrow[d] \\ \text{Sort by distance} \arrow[d] \\ \textbf{Output coordinates of resultant patches}\end{tikzcd}$$

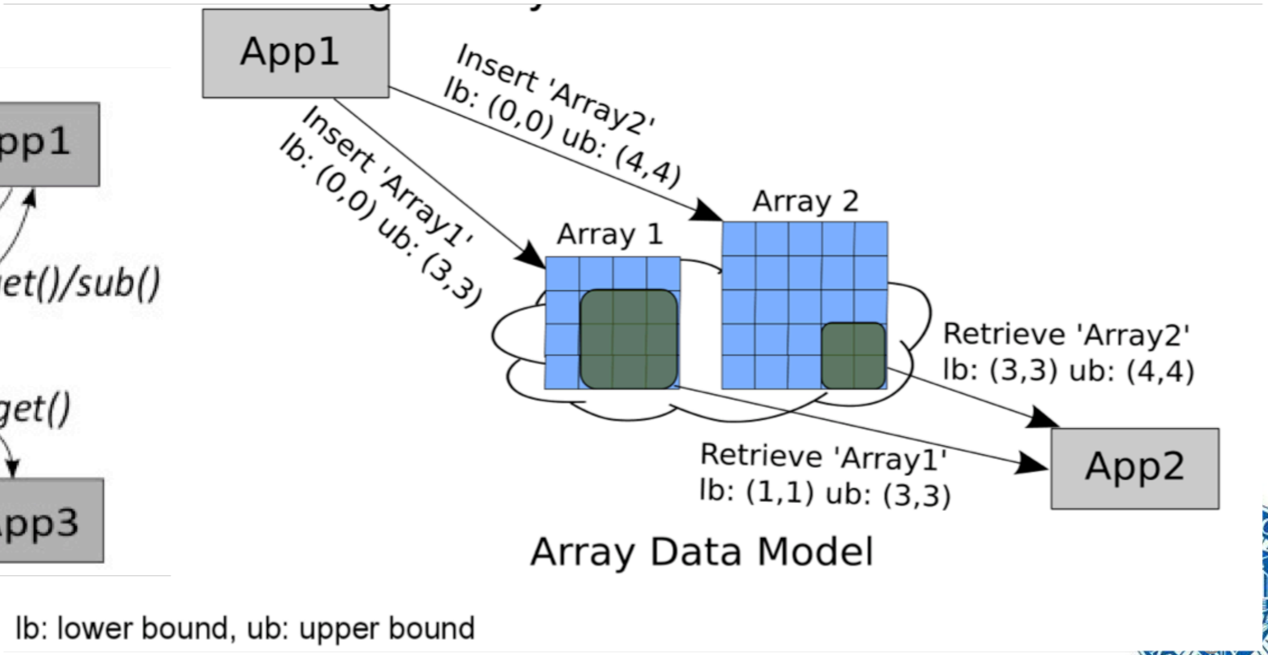
Resource Management

- Storage System
  - Exploit parallel file systems (e.g.Lustre, GPFS) through parallel disk accesses
  - Distribute tile coordinates based on the tile size and WSI path information among MPI processes
  - Let each process access the parallel file system to bring the assigned list of tiles into memory
- Processing Power
  - Hybrid + Nested parallel programming
    - Addition of parallel threads (OpenMP) into communicating sequential processes(MPI) to calculate feature sets on tiles
    - Increase thread-level parallelism through nested threads
- Memory Management
  - Keep track of intermediate data size
    - Spill to disk if necessary and do a parallel external sort
  - Stage input data in memory a distributed memory staging system

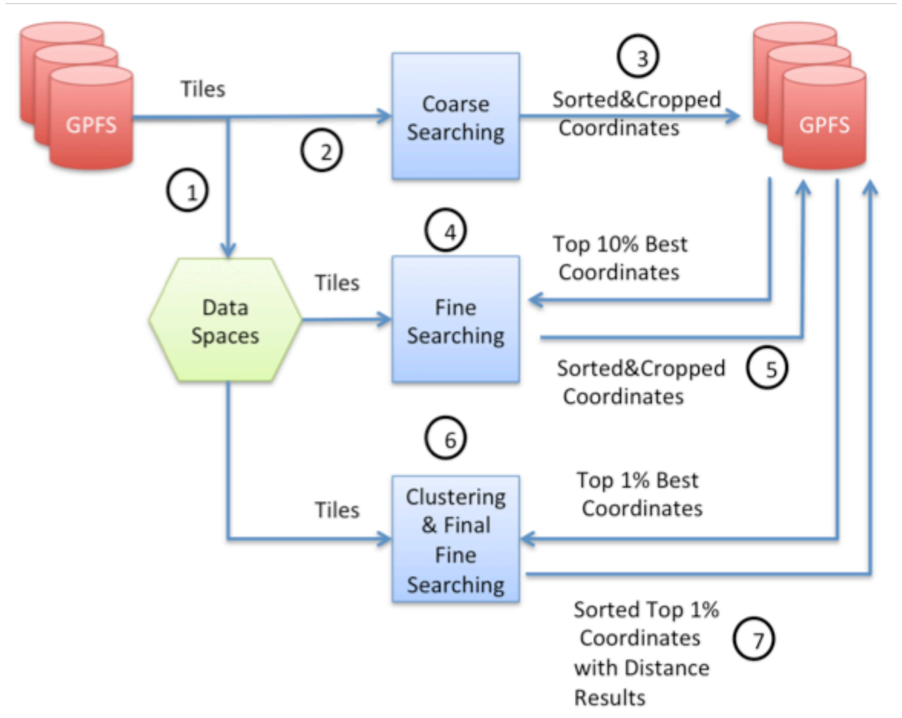
Memory Management: *Staging Input Data*

- Repeated data accesses to input data in each stage of the workflow
- Keep data in a shared distributed memory area
- DataSpaces being developed at RDI2
- Parallel applications can have synchronized access to tensor objects by specifying coordinates as bounding boxes





Memory Management: *Staging Input Data*



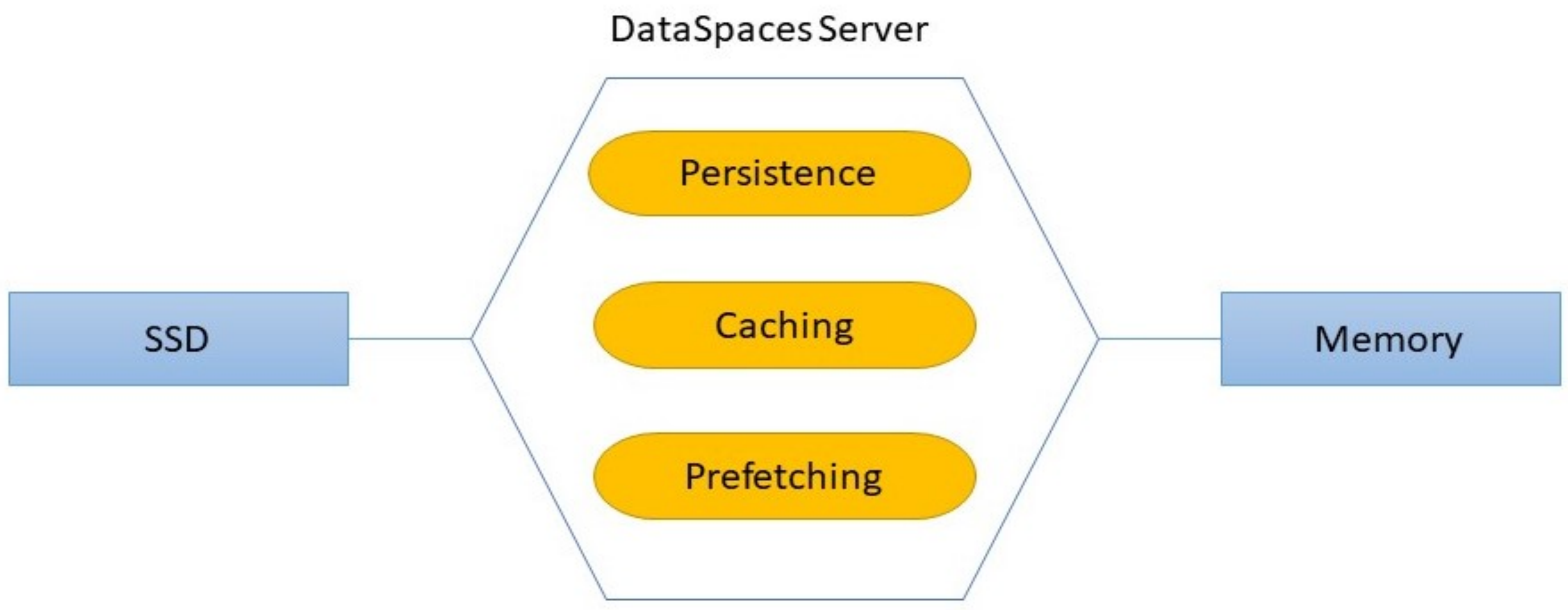
GPFS
Tiles
2
1
Coarse Searching
3
Sorted&Cropped Coordinates
GPFS
4
Top 10% Best Coordinates
Data Spaces
Tiles
Fine Searching
Sorted&Cropped Coordinates
5
6
Top 1% Best Coordinates
Tiles
Clustering & Final Fine Searching
Sorted Top 1% Coordinates with Distance Results
7

- Store tiles as objects in dataspaces
- Let the following stages access data from Dataspaces instead of file system

Deep Memory Hierarchies

- Memory space of each node is limited
  - Require large number of data staging nodes to accommodate the WSI data
- *Solution:* Use of deep memory hierarchies including Solid-state drives (SSDs) into the system
  - SSDs are faster than disk systems
  - NVMe SSDs perform even better

Deep Memory Hierarchies



Proposed framework consists of caching, prefetching and persistence modules implemented into DataSpaces server code

Modules: *Persistence*

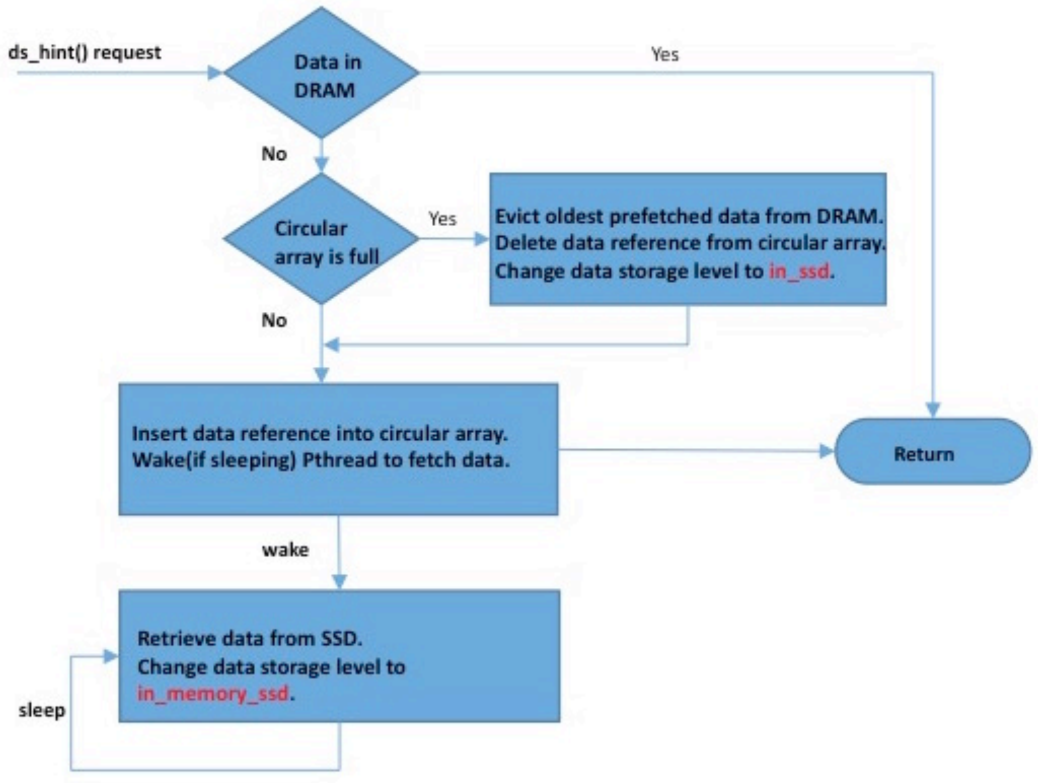
- Allows catching and prefetching modules allocate space on SSD
- Posix mmap() interface is used to allocate a large memory space on SSD
- Files in SSD can be accessed as if they were in memory
- It manages allocation/deallocation and fragmentation of space through doubly linked lists

Modules: *Caching*

- Uses a Least-recently Used algorithm to keep recently requested data in DRAM memory and evict less recently used data back to SSD.
- Data could be located in memory, in SSD or both
  - Status: in_memory, in_SSD, in_memory_SSD

Modules: *Prefetching*

- Makes the data available in DRAM memory before the application requests it
  - Masks the latency of SSD memory
- Operates as a concurrent thread under the main Dataspaces server thread
  - Pipelines the prefetching I/O operations between SSD and DRAM memory and I/O operations between client application and Dataspaces server



ds_hint() request
Data in DRAM
Yes
No
Circular array is full
Yes
Evict oldest prefetched data from DRAM. Delete data reference from circular array. Change data storage level to in_ssd.
No
Insert data reference into circular array. Wake(if sleeping) Pthread to fetch data.
Return
wake
Retrieve data from SSD. Change data storage level to in_memory_ssd.
sleep

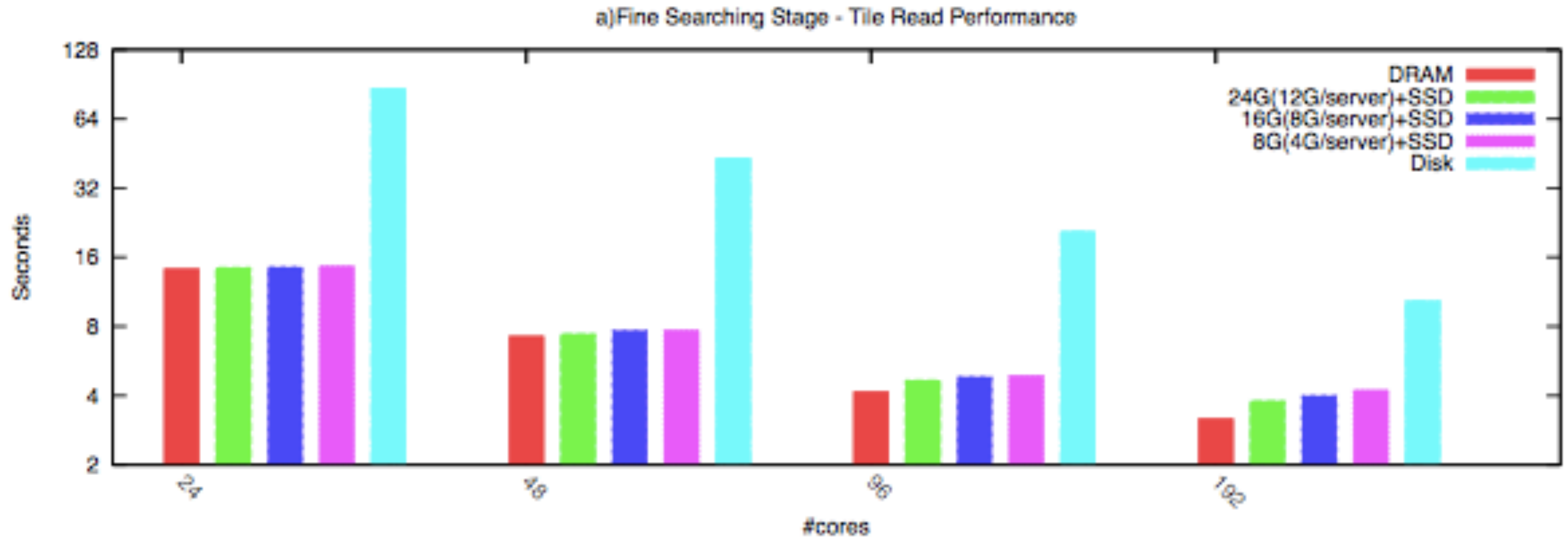
Updated and Newly Introduced Interfaces

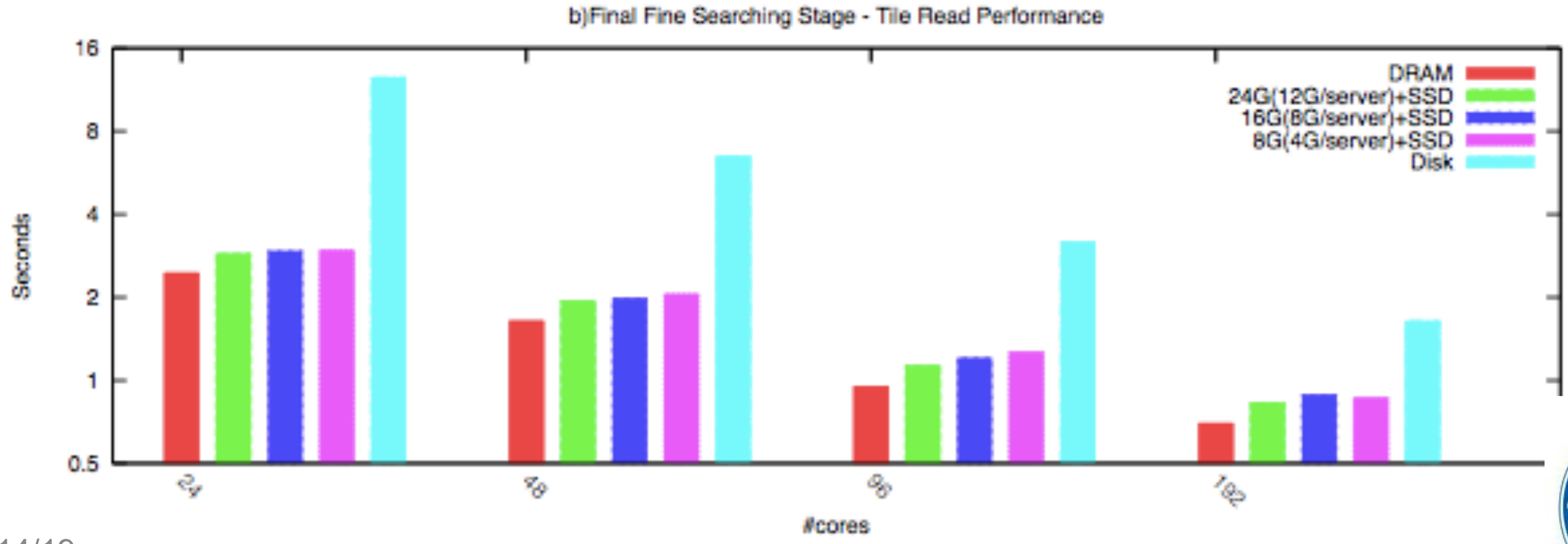
- dspaces init() calls persistence module to create the mapped file in SSD and launches the Prefetching thread.
- dspaces get() queries DataSpaces server to retrieve data.
  - If the data for the requested coordinates is in DRAM memory, the server returns it to the client.
  - If it is in SSD, it caches it into DRAM and changes data storage status into In memory ssd.
  - If DRAM has no space, then it calls the cache replacement algorithm in the Caching Module to evict some data into SSD before bringing requested data into DRAM.

Updated and Newly Introduced Interfaces

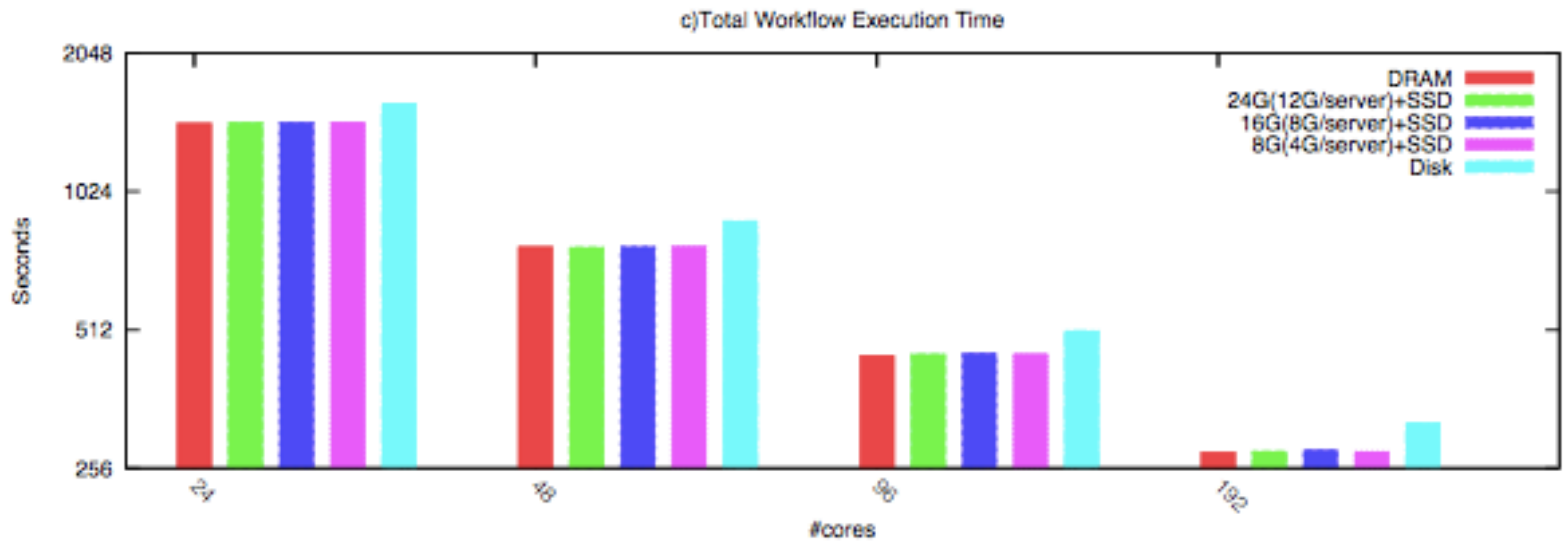
- dspaces put() inserts data in DRAM and changes data storage status into In memory.
  - If DRAM has no space then it calls the cache replacement algorithm before inserting data into DRAM.
- dspaces hint() queries DataSpaces servers to check if data is in SSD.
  - If so, the hint is inserted into prefetching circular array. It wakes prefetching thread if the thread is sleeping to fetch data from SSD to DRAM. After the operation is complete data storage status is changed into In_memory_ssd.

Performance Results: *Overhead*

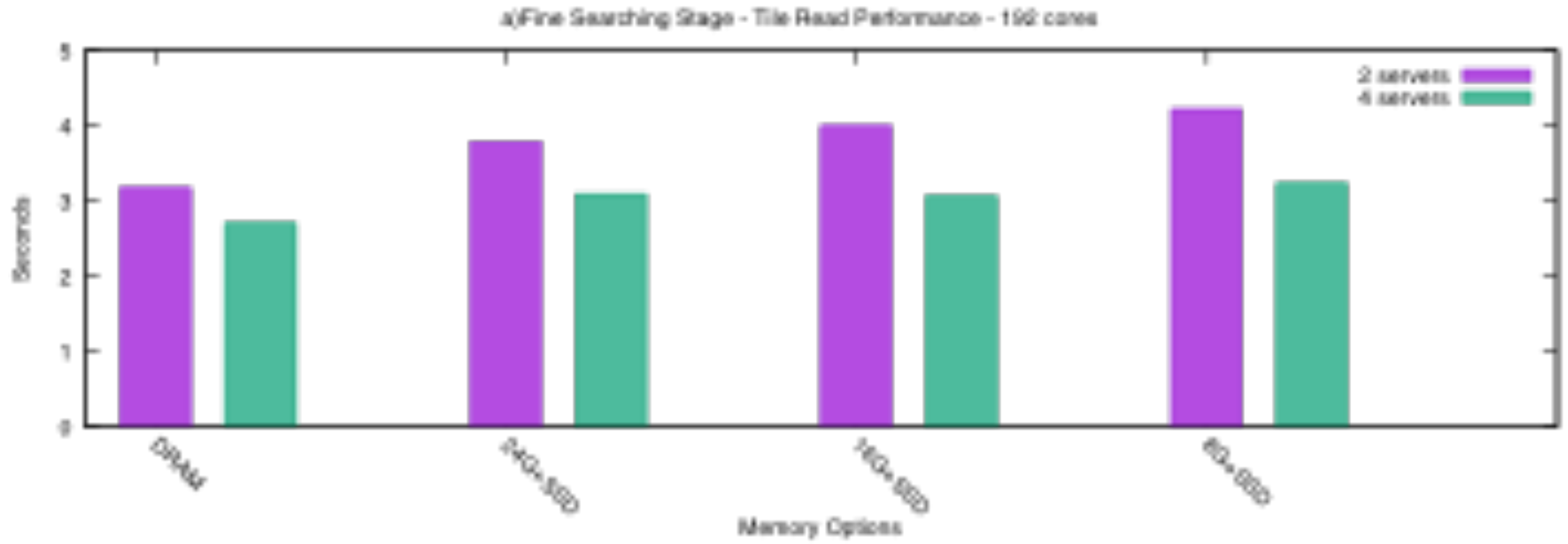


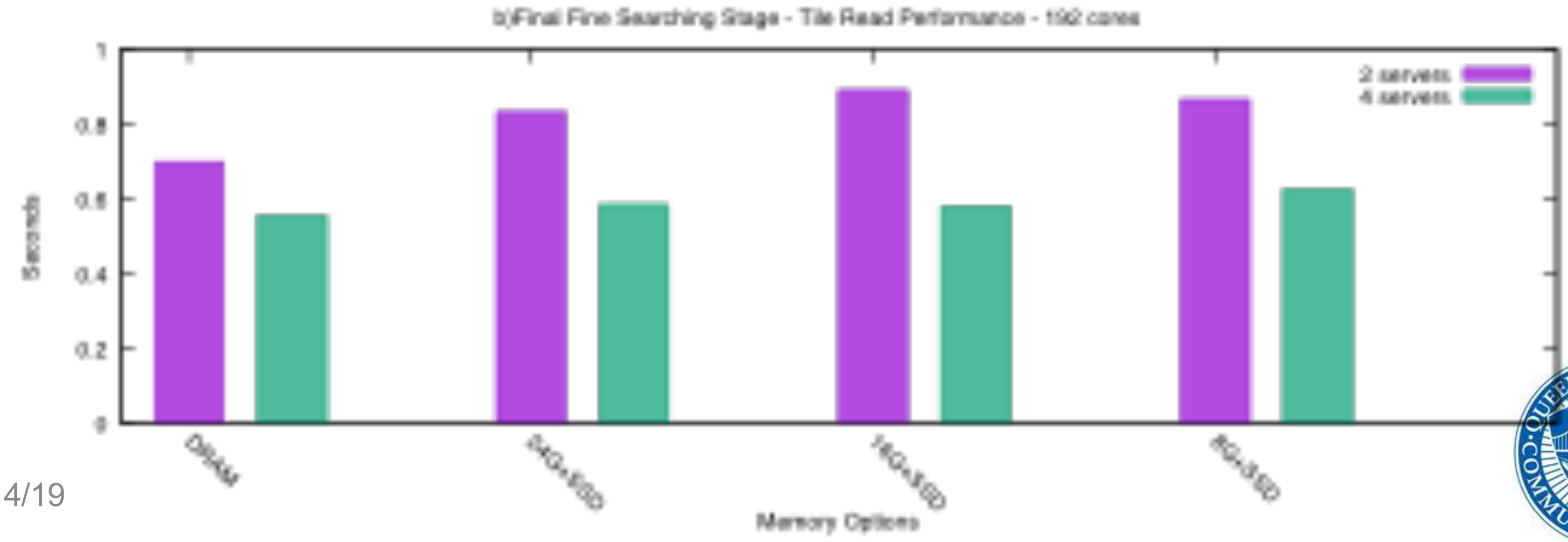


Performance Results: *Total Workflow Execution Time*



Performance Results: *Effect of #Staging Nodes*





Conclusions

- An extreme-scale multi-stage CBIR implementation is possible without preprocessing of input data
- Staging data for subsequent stages of the workflow reduce execution time
- A novel deep memory hierarchy approach can remove memory limit of the nodes by hiding latencies introduced by SSDs.
- The overhead of the deep memory hierarchy system is negligible