BITS Pilani
Pilani Campus

DSECL ZG517 - Systems for Data Analytics

Session #1 – Systems Attributes for Data Analytics – Single System

Murali P
muralip@wilp.bits-pilani.ac.in

[Saturday – 04:30 PM]

Agenda

- Review of Systems Attributes for Data Analytics - Single System
  - Memory Heirarchy
- Locality of Reference

innovate achieve lead

BITS Pilani
Pilani Campus

Review: Systems Attributes for Data Analytics - Single System

Courtesy: Prof Sundar B slides

Storage for Data

- **Structured** Data
  - E.g. Relational Data
- **Semi-structured** Data
  - E.g. HTML pages, XML data, JSON, CSV files, Email, NoSQL DB, etc.
- **Un-structured Data**
  - E.g. X-ray images, audio/video/photo files, word processing docs, books, journals, health records, metadata, etc.

Anecdotal Evidence

- *Most of the data today is semi-structured / unstructured.*

Kinds of Data and Forms of Storage [3]

- Today,

Kind of Data	Form of Storage	Example (Products)
Structured (Relational)	Relational Databases	Oracle, MSSQL, and MySQL; SimpleDB (Amazon)
Semi-structured / Unstructured	File Systems or Object Storages or NOSQL databases (including XML databases)	MongoDB; S3, Elastic FS (Amazon)

Discussion/Assignment:

How do we access data in relational vs semi-structured vs unstructured data?

Compare with real examples

Data Location: Memory vs. Storage

- Computational Data is stored in
  - ***Primary Memory*** (a.k.a. Memory)

vs.

- Persistent Data is stored in
  – i.e. ***Secondary Memory*** (a.k.a Storage)

Use and Throw

Multiple runs

Questions / Exercises

1. What does "persistent" refer to? Is it same as non-volatile?
2. Identify examples of these two kinds of data
3. Identify technologies suitable for the two kinds of data

Data Location: Memory vs. Storage vs. Network

- Data accessed from a ***local store***
  - i.e. storage attached to a computer

vs.

- Data accessed from a ***remote store*** / remote processor
  - i.e. <u>storage hosted </u>on the network (or *storage attached to a computer hosted on the network*)

<u>Question</u>

- What is the difference in the form of access?

innovate achieve lead

There is a link between

form of data and form of storage

Cost of Access: *Memory vs. Storage vs. Network*

- Exercise:
  - What are the typical access times?
    1. RAM
    2. Hard Disk
    3. Ethernet LAN
  - Access Time Parameters: ***Latency*** and ***Bandwidth***
    - When and how do these parameters matter?
  - Identify mechanisms used to alleviate the access time delays in each case.

Memory Bandwidth Requirement [3]

- Total bandwidth to constantly feed processor:
  - 40GBps

How to give a typical 4-core processor about 40GB of data every second?

Memory Hierarchy – Motivation

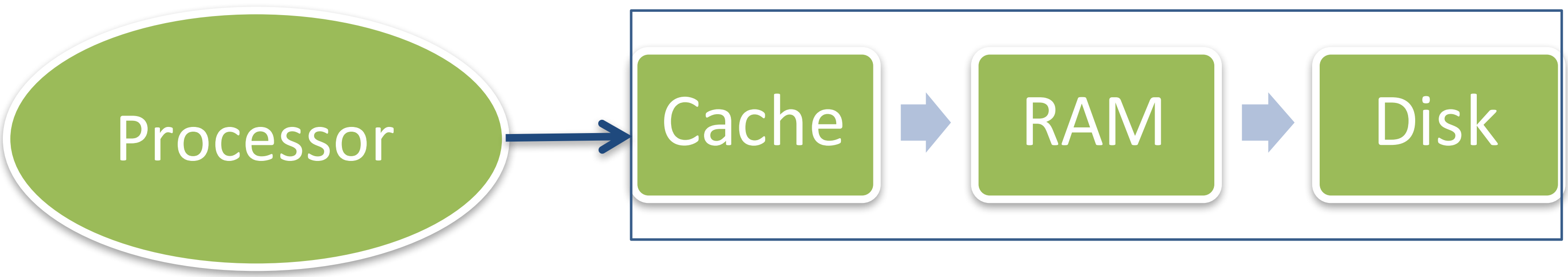
- A **Memory Hierarchy** amortizes cost in computer architecture:
  - fast (*and therefore costly*) but small-sized memory

to

  - large-sized but slow (*and therefore cheap*) memory

Memory Hierarchy

- Original:



- Modern:

Discussion/Assignment:

How do we
Reconcile Memory Bandwidth Requirement
with the Memory Hierarchy?

innovate achieve lead

BITS Pilani
Pilani Campus

Locality of Reference

Courtesy: Prof Sundar B slides

Locality of Reference(s)

- The Principle of **Locality of Reference**(s)
  - **The locus of data access** – and hence that *of memory references* – *is small at any point during program execution.*

- ***more like an Observation***

- **Temporal Locality**
- **Spatial Locality**

Locality of References - Temporal Locality

- Temporal Locality
  - Data that is accessed (at a point in program execution) is *likely to be accessed again in the near future:*
    - i.e. data is likely to be repeatedly accessed in a short span of time during execution

Locality of References - Temporal Locality

- Temporal Locality
  - Data that is accessed (at a point in program execution) is *likely to be accessed again in the near future*:
    - i.e. data is likely to be repeatedly accessed in a short span of time during execution
- Examples (*of manifestation of Temporal Locality*)
  1. Instructions in the body of a loop
  2. Parameters / Local variables of a function / procedure
  3. Data (or a variable) that is computed iteratively
     - e.g. a cumulative sum or product

Locality of References - Spatial Locality

- Spatial Locality
  - Data accessed (at a point in program execution) is likely *located adjacent to data* that is to be *accessed in near future*:
    - i.e. data accessed in a short span during execution is likely to be within a small region (in memory)

Locality of References - Spatial Locality

- Spatial Locality
  - Data accessed (at a point in program execution) is likely *located adjacent to data* that is to be *accessed in near future*:
    - i.e. data accessed in a short span during execution is likely to be within a small region (in memory)
- Examples (*of manifestation of Spatial Locality*)
  - Linear sequences of instructions
  - Elements of Arrays (accessed sequentially)

Locality of References - Spatial Locality

- Spatial Locality
  - Data accessed (at a point in program execution) is likely *located adjacent to data* that is to be *accessed in near future*:
    - i.e. data accessed in a short span during execution is likely to be within a small region (in memory)
- Examples (*of manifestation of Spatial Locality*)
  - Linear sequences of instructions
  - Elements of Arrays (accessed sequentially)

Question: *Accessing nodes in a linked list sequentially may violate this principle*. Why?

Memory Hierarchy and Locality

- A memory hierarchy is effective only due to **Locality** exhibited by programs (and the data they access)!
  - Longer the range of execution time of the program, larger is the locus of data accesses:
    - this aligns with the memory hierarchy:
      - *increasing size with increasing access time of memory levels*

Locality Example 1: Matrices

- Consider matrices and an operation such as the *addition of two matrices*:
  - elements M[i,j] may be accessed either *row by row* or *column by column*

Locality Example 1: Matrices

- Consider matrices and an operation such as the *addition of two matrices*:
  - elements M[i,j] may be accessed either *row by row* or *column by column*
- i.e. one can write the addition algorithm as:

```
for (i=0; i<N; i++)
   for(j=0; j<N; j++)
      M3[i,j] = M2[i,j] + M1[i,j]
```

1	2	3
4	5	6
7	8	9

Locality Example 1: Matrices

- Consider matrices and an operation such as the *addition of two matrices*:
  - elements M[i,j] may be accessed either *row by row* or *column by column*
- i.e. one can write the addition algorithm as:

```
for (i=0; i<N; i++)
   for(j=0; j<N; j++)
      M3[i,j] = M2[i,j] + M1[i,j]
```

1	2	3
4	5	6
7	8	9

- or as

```
for (j=0; j<N; j++)
    for(i=0; i<N; i++)
      M3[i,j] = M2[i,j] + M1[i,j]
```

1	4	7
2	5	8
3	6	9

Locality Example 1: Matrices

- Consider matrices and an operation such as the *addition of two matrices*:
  - elements M[i,j] may be accessed either *row by row* or *column by column*
- i.e. one can write the addition algorithm as:

```
for (i=0; i<N; i++)
  for(j=0; j<N; j++)
   M3[i,j] = M2[i,j] + M1[i,j]
```

- or as

```
for (j=0; j<N; j++)
   for(i=0; i<N; i++)
     M3[i,j] = M2[i,j] + M1[i,j]
```

Time Complexity of both these algorithms is the same but:
- locality varies (Why?)

and
- so does performance!

Locality Example 1: Matrices [*contd*.]

- Matrices are 2-dimensional but memory is 1-dimensional!
  - i.e. matrices are stored in row-major order or in column-major order.
- Impact?

Locality Example 1: Matrices [*contd.*]

- Matrices are 2-dimensional but memory is 1-dimensional!
  – i.e. matrices are stored in row-major order or in column-major order.
- Impact?

#rows	#cols	# elements	rowTime	colTime
512	512	262144	1000	1000
1024	1024	1048576(1M)	3999	5999
2048	2048	4M	15997	32995
4096	4096	16M	62990	141978
8192	8192	64M	253961	670898
16384	16384	256M	1014846	3013541

Locality Example 2: Partitioning in QuickSort

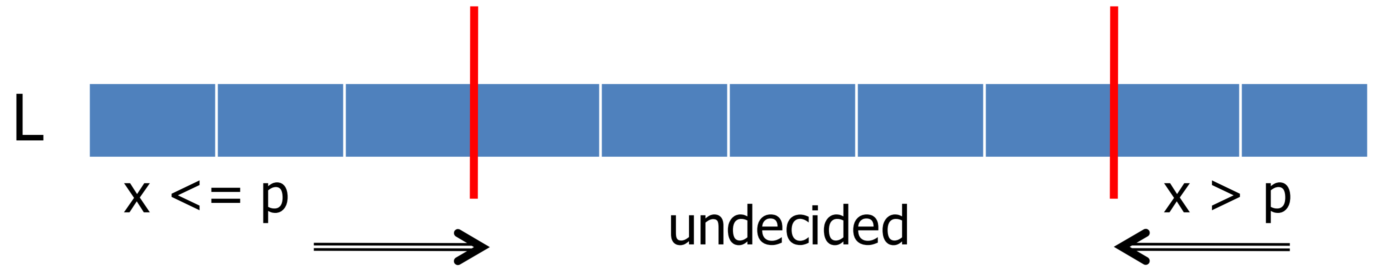
- Partitioning is a step in QuickSort:
  - Given a *pivot* ***p***, partition a list ***L*** such that:
    - **p= L[i]** for some ***i*** where **0 <= i < n** and
    - for all ***j*** where **0 < j < n** and **n = length(L)**
      - **j < i** implies **L[j] <= p** and
      - **j > i implies L[j] > p**

Locality Example 2: Partitioning in QuickSort

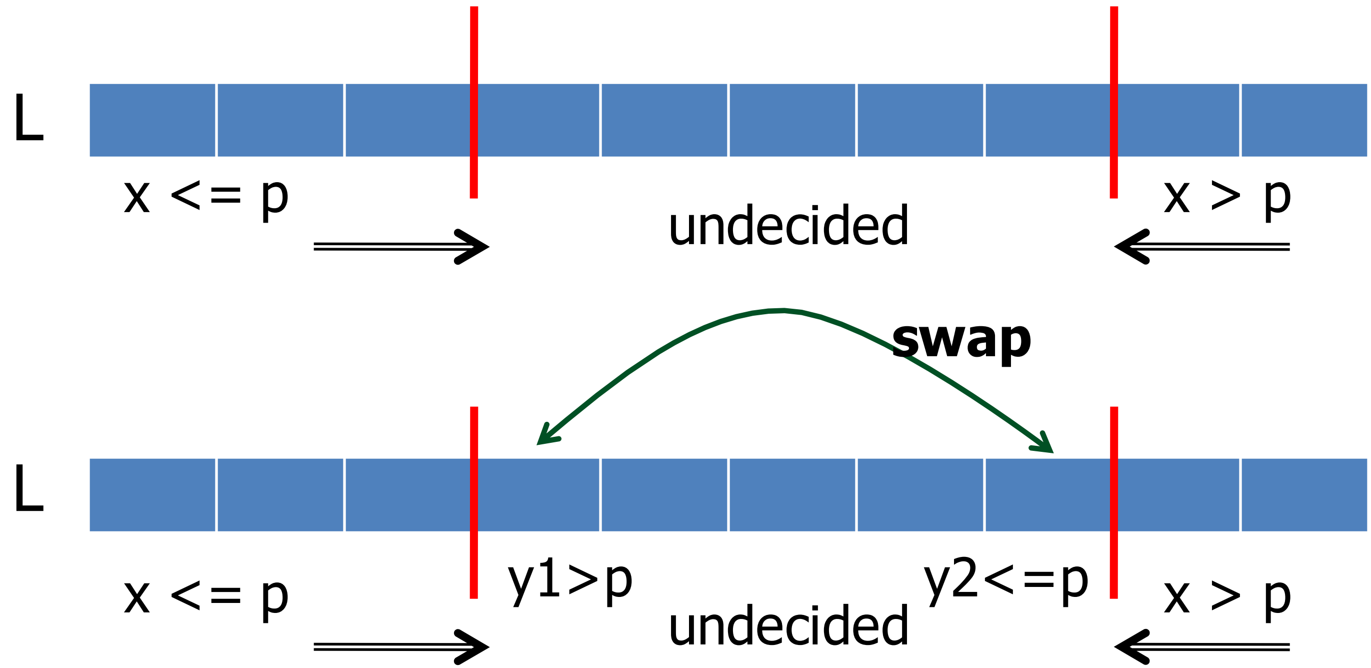
- Partitioning is a step in QuickSort:
  - Given a *pivot* ***p***, <u>partition a list ***L*** such that</u>:
    - **p= L[i]** for some ***i*** where **0 <= i < n** and
    - for all ***j*** where **0 < j < n** and **n = length(L)**
      - **j < i** implies **L[j] <= p** and
      - **j > i implies L[j] > p**
- Hoare's Partitioning Algorithm:

```
/* assume L[lo] = p */
i=lo+1;  j=hi;
while (i <= j) {
   while (L[i]<=p) i++;   while (L[j]>p) j--;
   if (i<j) swap(L[i], L[j]);
}
```

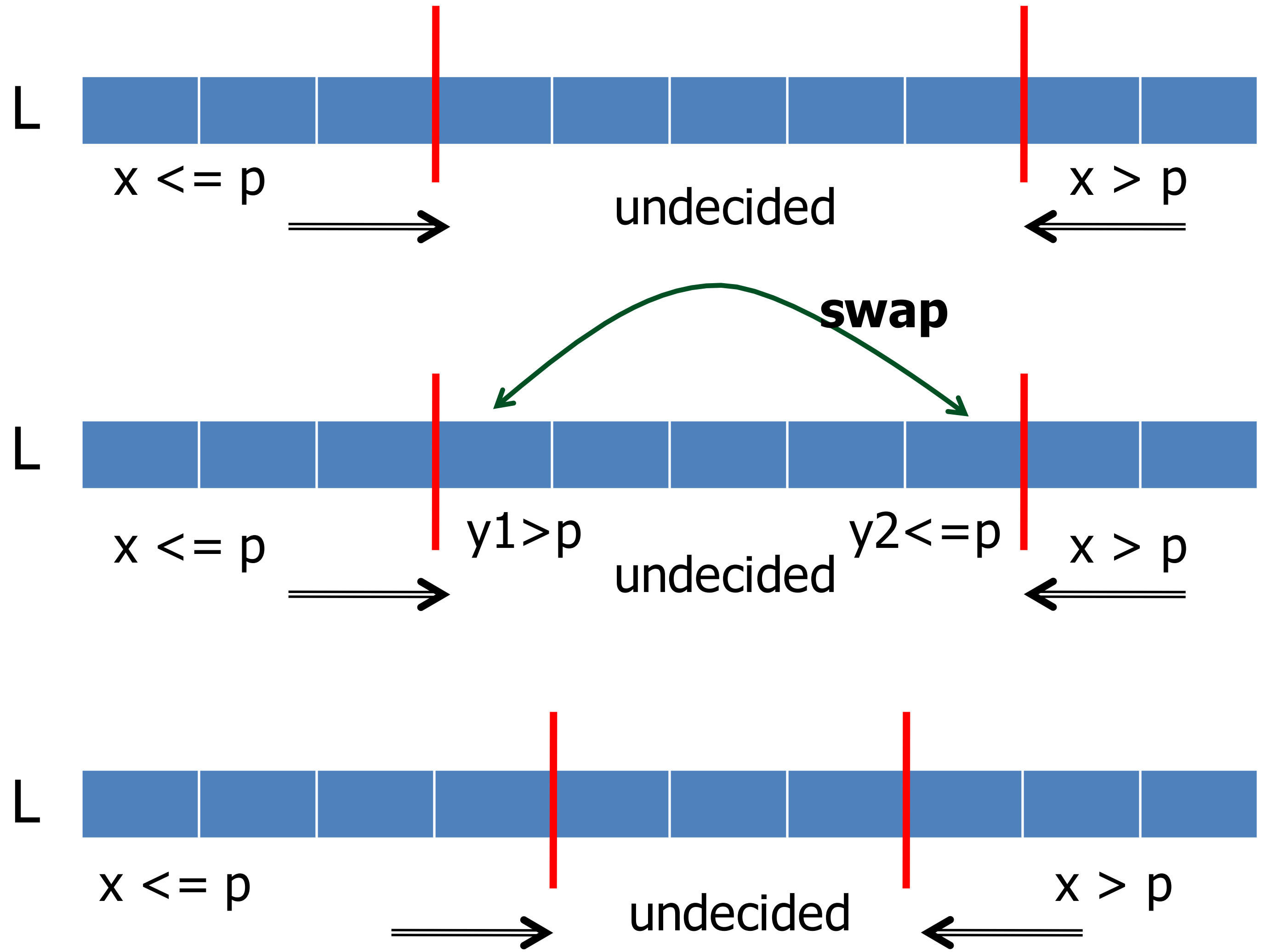
(Hoare's) Partitioning : (pivot p in L)



(Hoare’s) Partitioning : (pivot p in L)



(Hoare's) Partitioning : (pivot p in L)



Partitioning and Locality

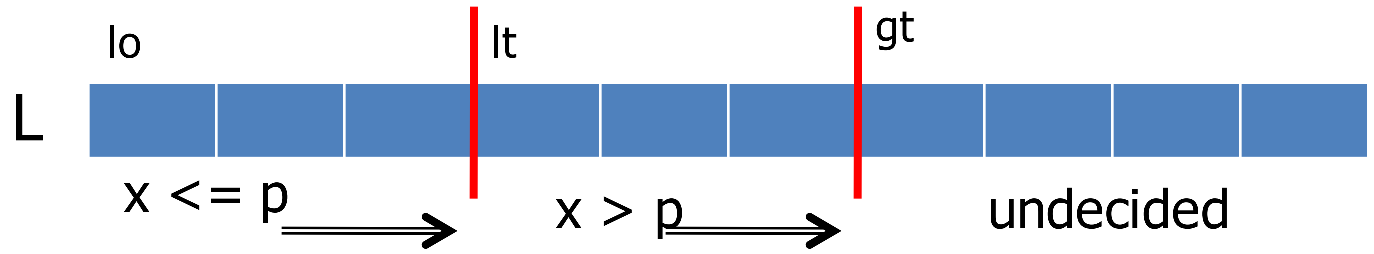
- Recall the inner loops of the Partition algorithm:
  - Array is accessed left-to-right (**L-R**) in one loop, and right-to-left (**R-L**) in the next.
- And these two loops are repeated in the outer loop
  - i.e. the ***locus*** alternates within each iteration of the outer loop, and from the end of one iteration to the start of the next

I1: L-R	X	X								
I1:R-L									Y	Y
I2: L-R			X	X						
I2: R-L								Y		
I3: L-R					X					
I3: R-L							Y			

Back to the Partitioning Algorithm:

- Can you access the array elements from one end instead of both ends?

Locality-Aware Partitioning : (pivot p in L)



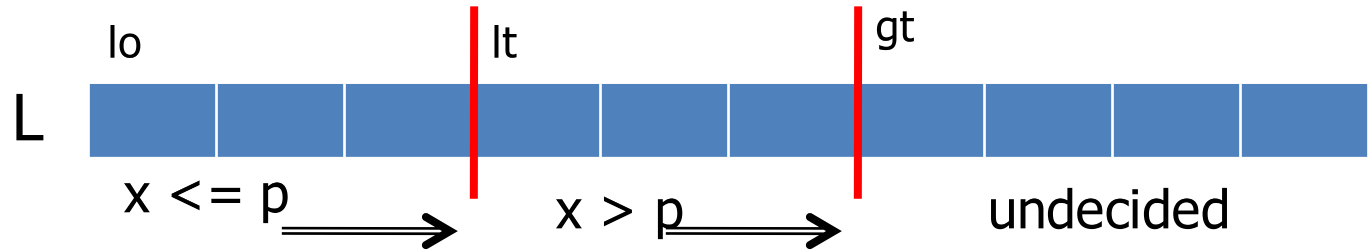
Maintain these sub-lists (and *the invariants*) i.e.

forall j: lo<=j<=lt --> L[j]<=piv

and

forall j: lt<j<=gt --> L[j]>piv

Locality-Aware Partitioning : (pivot p in L)



Lomuto's partitioning

Maintain these sub-lists (and *the invariants*)

i.e.

forall j: lo<=j<=lt --> L[j]<=piv

and

forall j: lt<j<=gt --> L[j]>piv

<u>Exercise</u>:

- Code the locality-aware partitioning algorithm.
- Determine the impact of locality by measuring the time taken – for different data sizes – the classic algorithm and this one.

Thank you !