Very Large Dataset Access and Manipulation: Active Data Repository (ADR) and DataCutter

Joel Saltz

University of Maryland, College Park

and

Johns Hopkins Medical Institutions

http://www.cs.umd.edu/projects/adr

Data Intensive Research Group

University of Maryland/Johns Hopkins

- Mike Beynon
- Umit Catalyurek
- Chialin Chang
- Renato Ferreira
- Tahsin Kurc
- Alan Sussman

Tools to Manage Storage Hierarchy

- **Mass Storage:**
  - Load subset of data from tertiary storage into disk cache or client
  - Access data from distributed data collections
  - Preprocess close to data sources
- **Fast secondary storage**
  - Tools for on-demand data product generation, interactive data exploration, visualization
  - Target closely coupled sets of processors/disks

Irregular Multi-dimensional Datasets

- **Spatial/multi-dimensional multi-scale, multi-resolution datasets**
- **Applications select portions of one or more datasets**
- **Selection of data subset makes use of spatial index (e.g., R-tree, quad-tree, etc.)**
- **Data not used “as-is”, generally preprocessing is needed - often to reduce data volumes**

DataCutter

- **A suite of Middleware for subsetting and filtering multi-dimensional datasets stored on archival storage systems**
- **Subsetting through Range Queries**
  - a hyperbox in dataset's multi-dimensional space
  - retrieve items with multi-dimensional coordinates in box
- **Processing (filtering/aggregations) through *Filters***
  - Carry out processing near data, compute servers

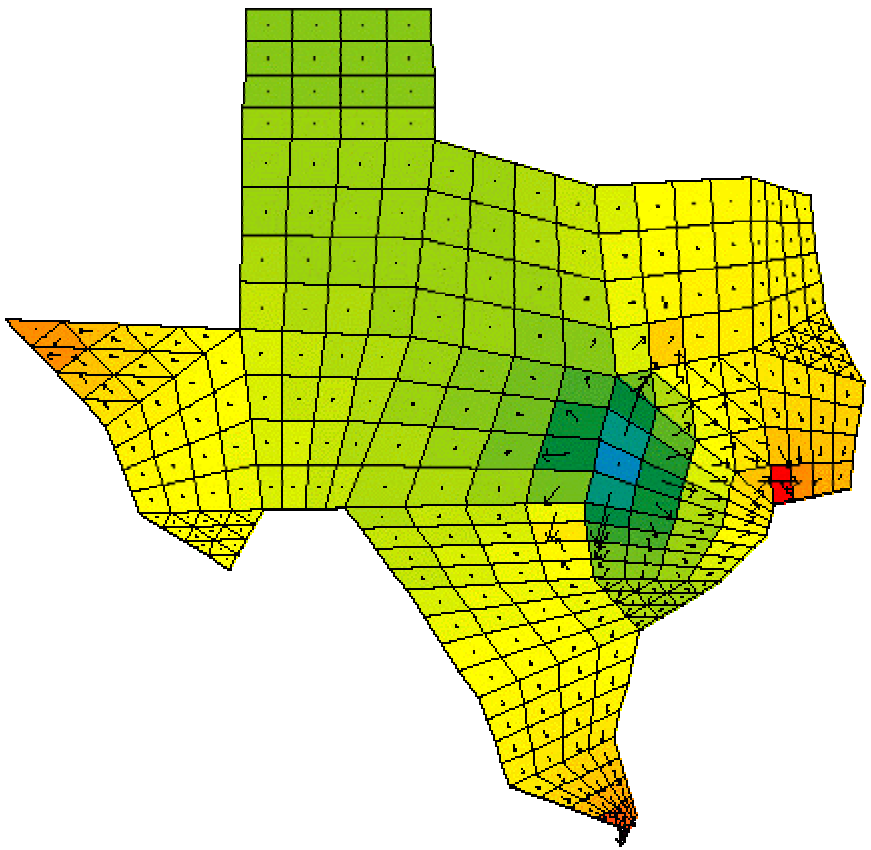
Active Data Repository (ADR)

- **Set of services for building parallel databases of multi-dimensional datasets**
  - enables integration of storage, retrieval and processing of multi-dimensional datasets on parallel machines.
  - can maintain and jointly process multiple datasets.
  - provides support and runtime system for common operations such as
    - data retrieval,
    - memory management,
    - scheduling of processing across a parallel machine.
  - customizable for various application specific processing.

Querying Irregular Multi-dimensional Datasets

- **Irregular datasets**
  - Think of disk-based unstructured meshes, data structures used in adaptive multiple grid calculations, sensor data
    - indexed by spatial location (e.g., position on earth, position of microscope stage)
  - Spatial query used to specify iterator
    - computation on data obtained from spatial query
    - computation aggregates data - resulting data product size significantly smaller than results of range query

Dataset Structure

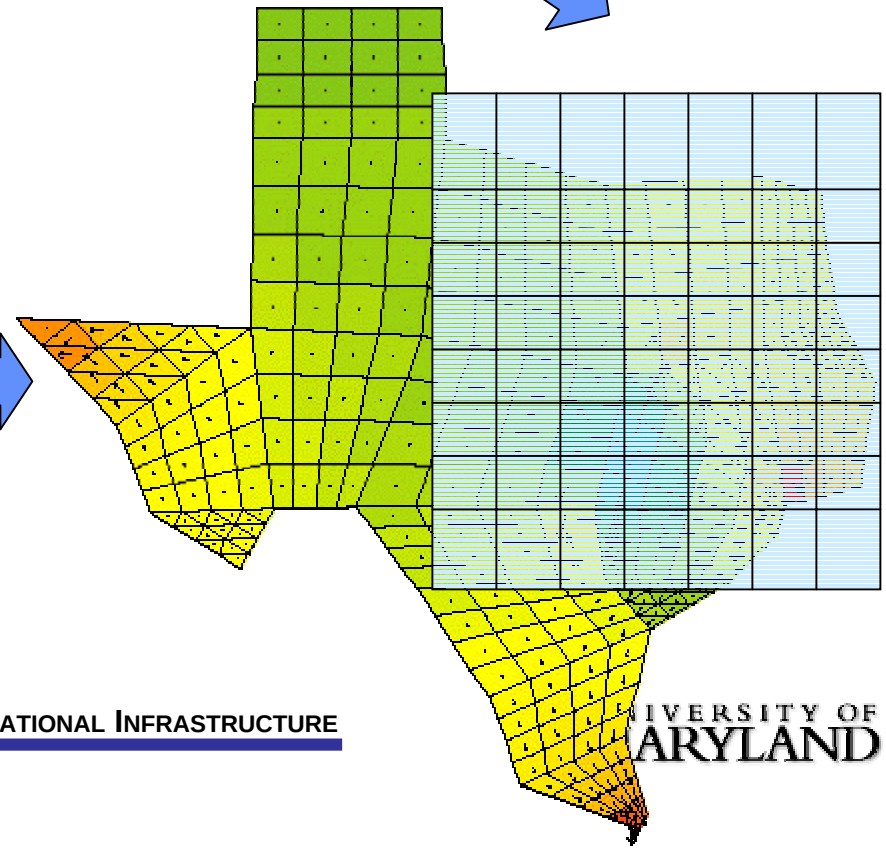


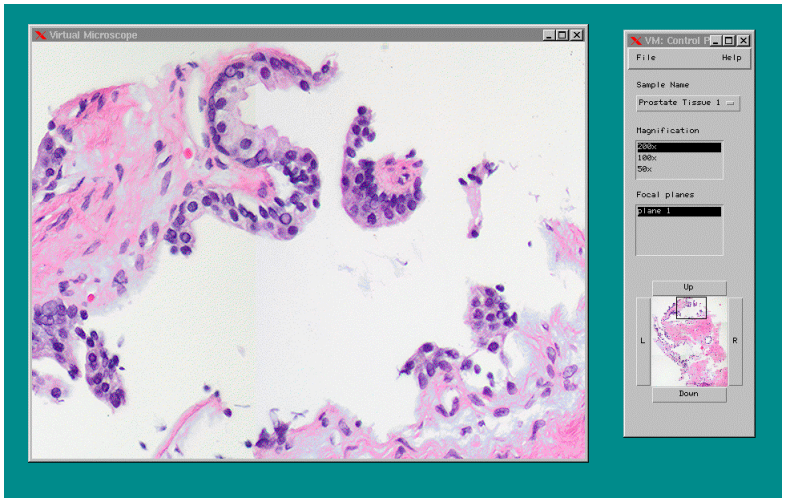
- **Spatial and temporal resolution may depend on spatial location**
- **Physical quantities computed and stored vary with spatial location**

Processing Irregular Datasets
Example -- Interpolation

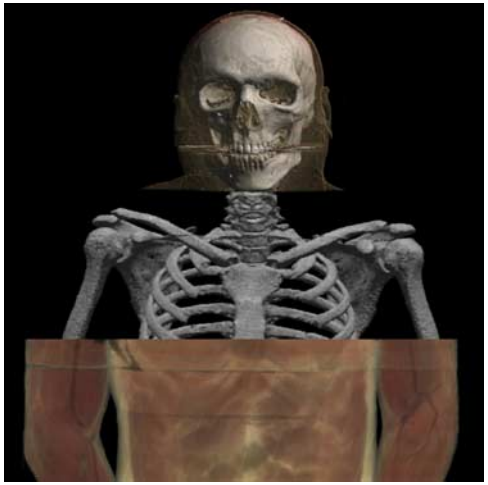
Output grid onto which a projection is carried out

Specify portion of raw sensor data corresponding to some search criterion



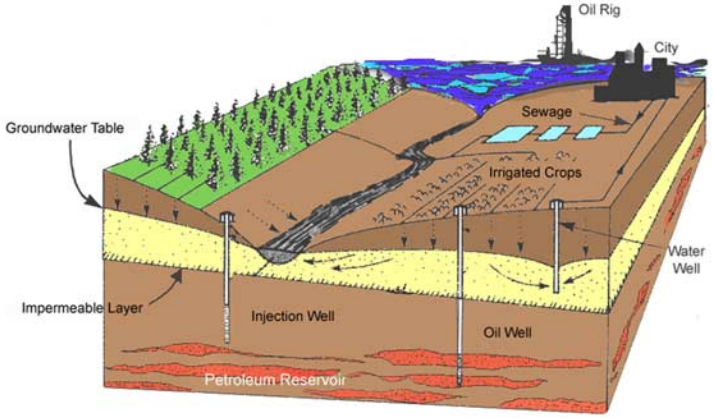


Pathology

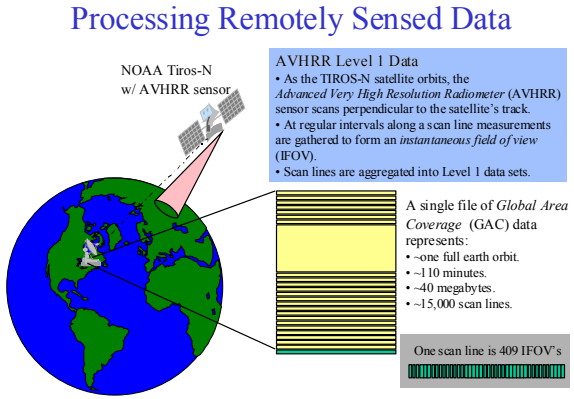


Volume Rendering

Applications



Surface/Groundwater Modeling



Satellite Data Analysis

Application Scenarios

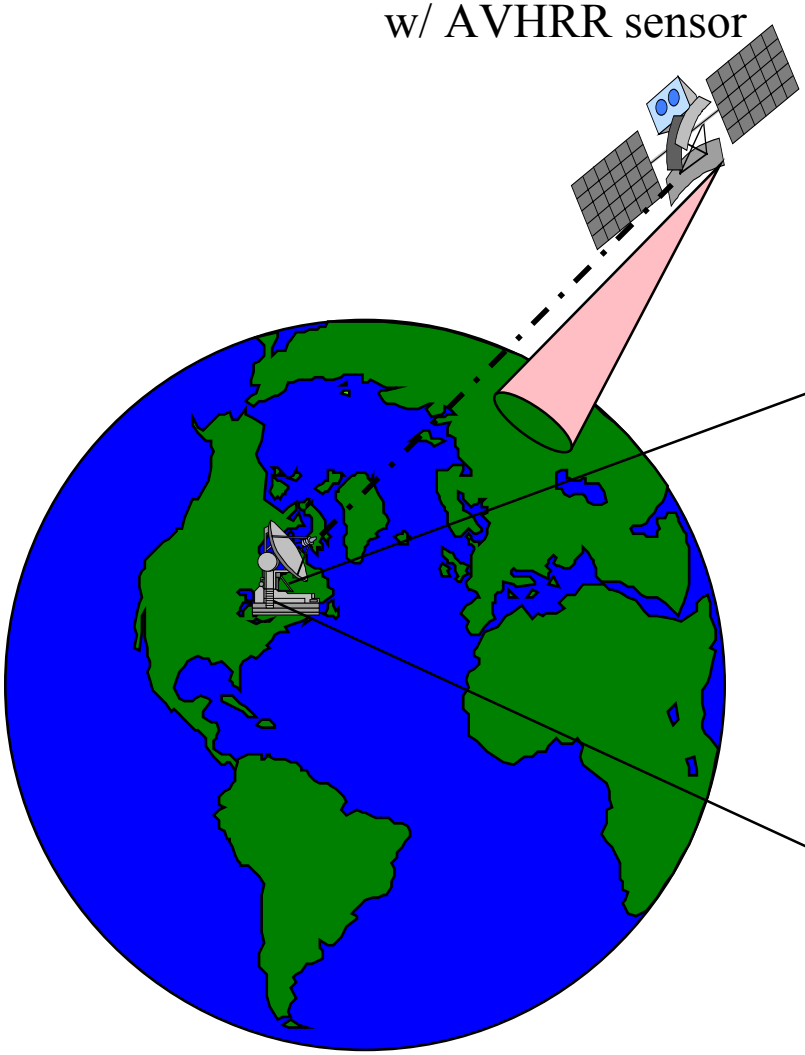
- **Locate TB spatio-temporal region in multi-scale, multi-resolution PB dataset, project data onto new spatio-temporal grid**
  - Ad-hoc queries, data products from satellite sensor data
  - Browse or analyze (multi-resolution) digitized slides from high power light or electron microscopy
    - 1-50 GBytes per digitized slide, 5-50 slides per case, 100's of cases per day per hospital

Application Scenarios (cont.)

- **Sensor data, fluid dynamics and chemistry codes to predict condition of waterways (e.g. Chesapeake bay simulation) and to carry out petroleum reservoir simulation**
- **Predict materials properties using electron microscope computerized tomography sensor data**
- **Post-processing, analysis and visualization of data generated by large scientific simulations**

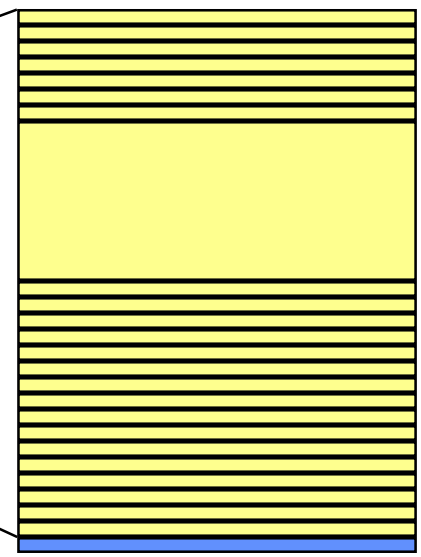
Processing Remotely Sensed Data

NOAA Tiros-N
w/ AVHRR sensor



AVHRR Level 1 Data

- As the TIROS-N satellite orbits, the *Advanced Very High Resolution Radiometer* (AVHRR) sensor scans perpendicular to the satellite's track.
- At regular intervals along a scan line measurements are gathered to form an *instantaneous field of view* (IFOV).
- Scan lines are aggregated into Level 1 data sets.



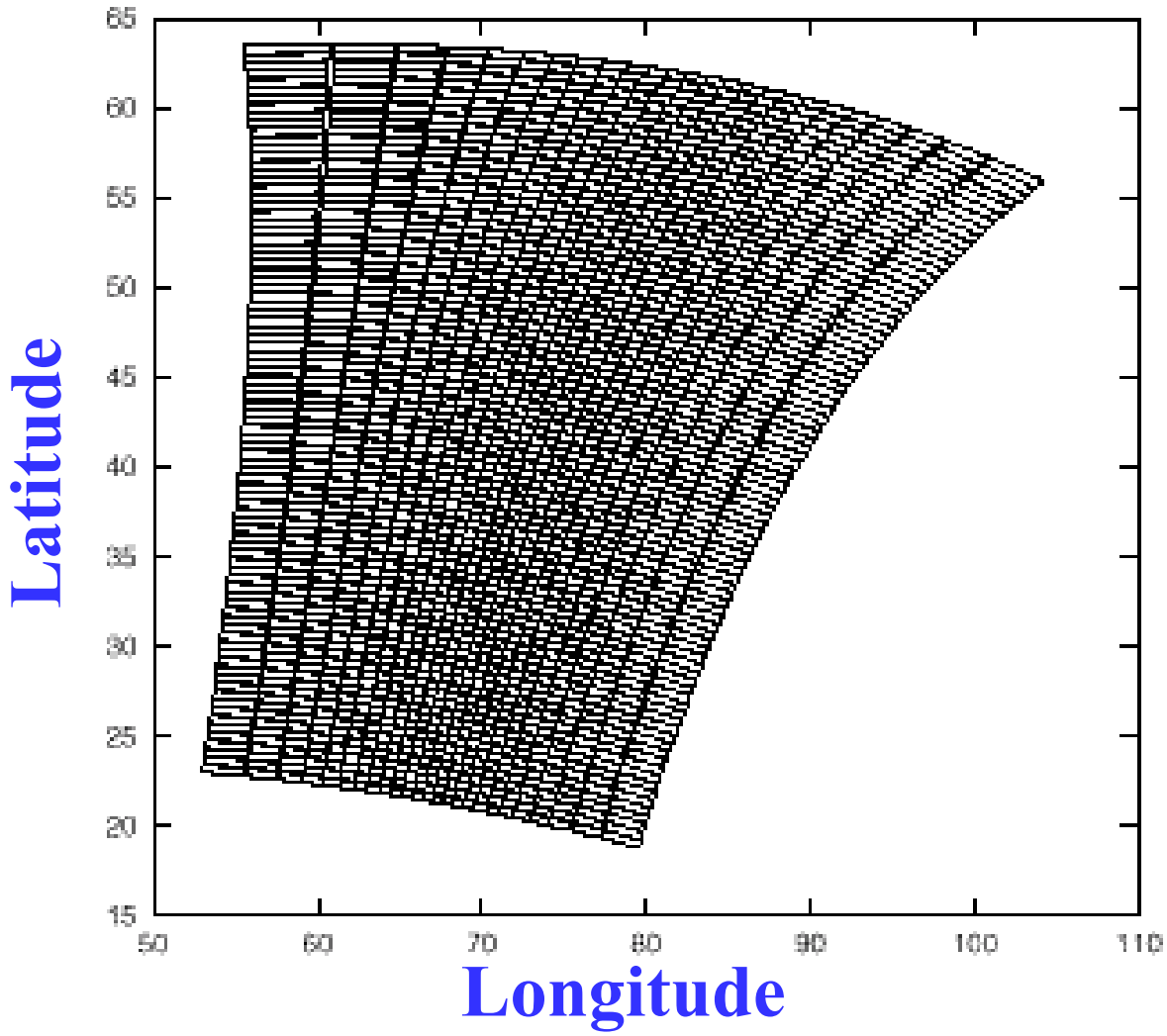
A single file of *Global Area Coverage* (GAC) data represents:

- ~one full earth orbit.
- ~110 minutes.
- ~40 megabytes.
- ~15,000 scan lines.

One scan line is 409 IFOV's

Spatial Irregularity

AVHRR Level 1B NOAA-7 Satellite 16x16 IFOV blocks.

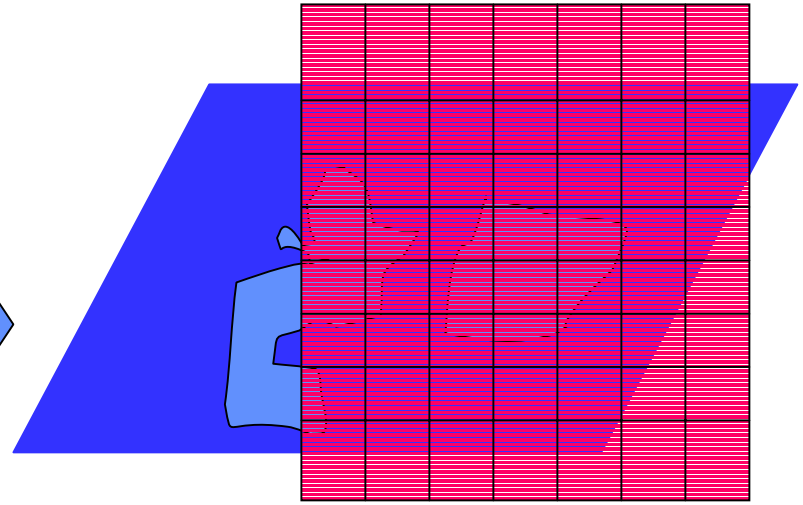


Active Data Repository

Typical Query

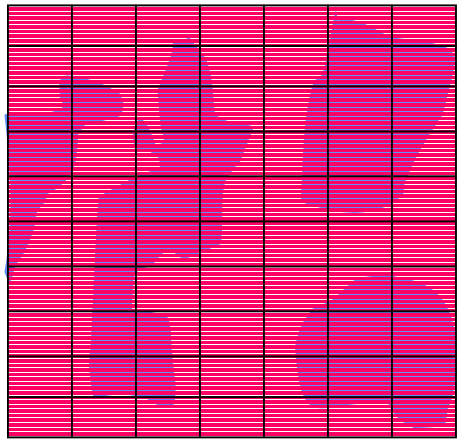
Output grid onto which a projection is carried out

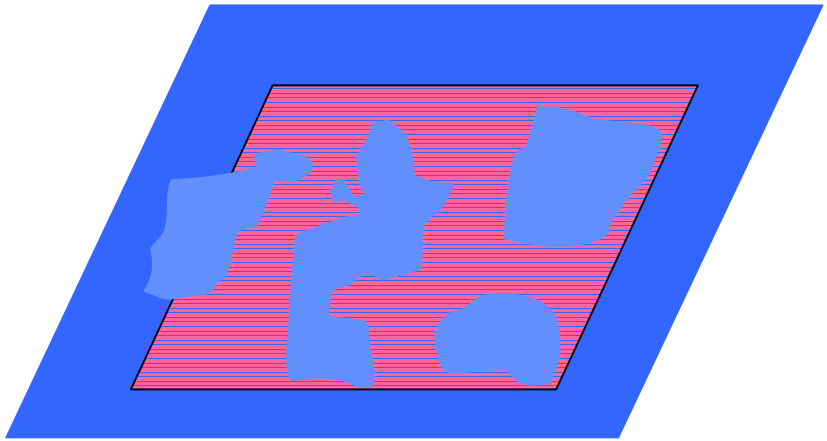
Specify portion of raw sensor data corresponding to some search criterion



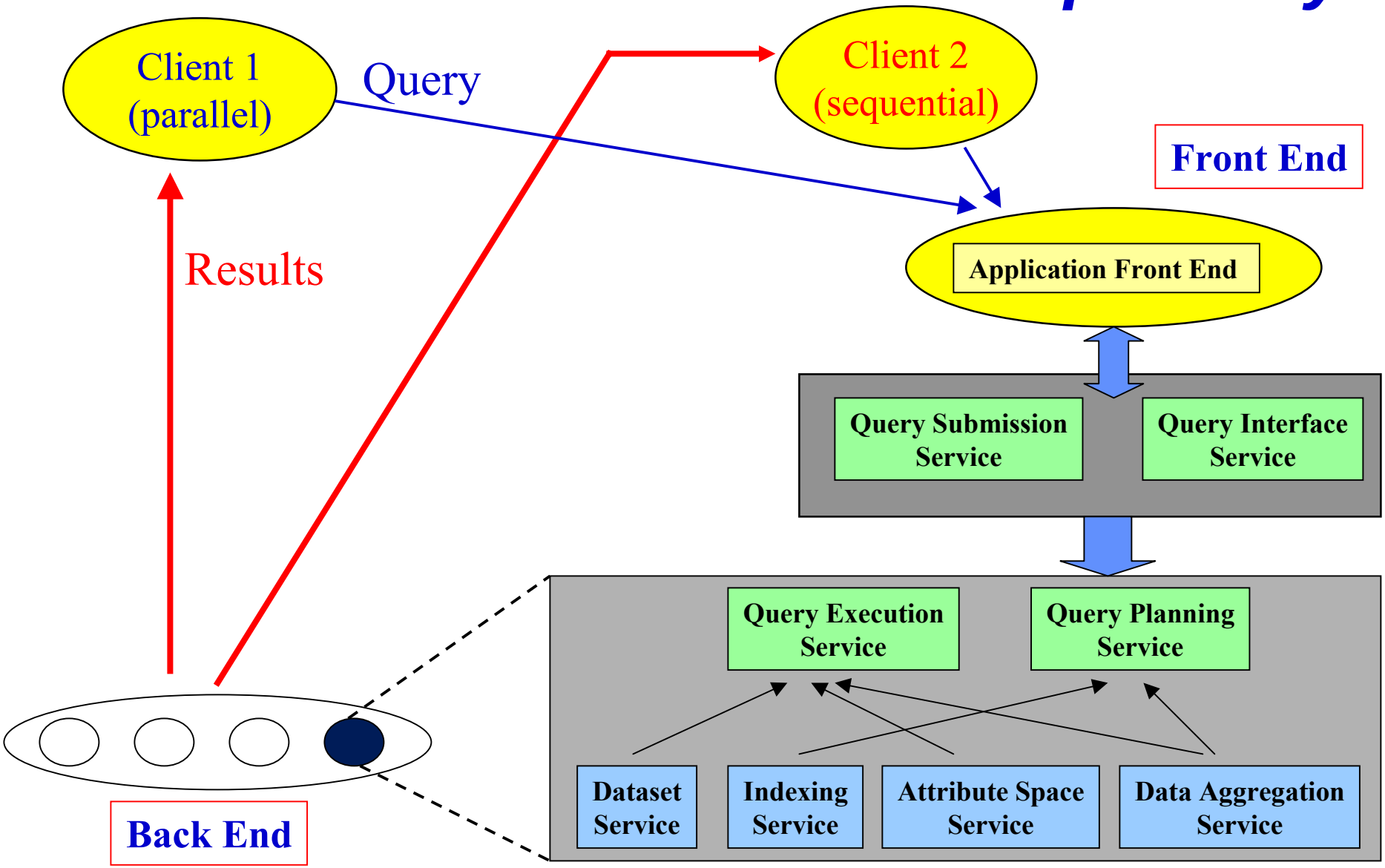
Application Processing Loop

```
O ← Output dataset, I ← Input dataset
A ← Accumulator (intermediate results)
[S_I, S_O] ← Intersect(I, O, R_query)
foreach o_e in S_O do
        read o_e
        a_e ← Initialize(o_e)
foreach i_e in S_I do
        read i_e
        S_A ← Map(i_e) ∩ S_O
        foreach a_e in S_A do
                a_e ← Aggregate(i_e, a_e)
foreach a_e in S_O do
        o_e ← Output(a_e)
        write o_e
```





Architecture of Active Data Repository

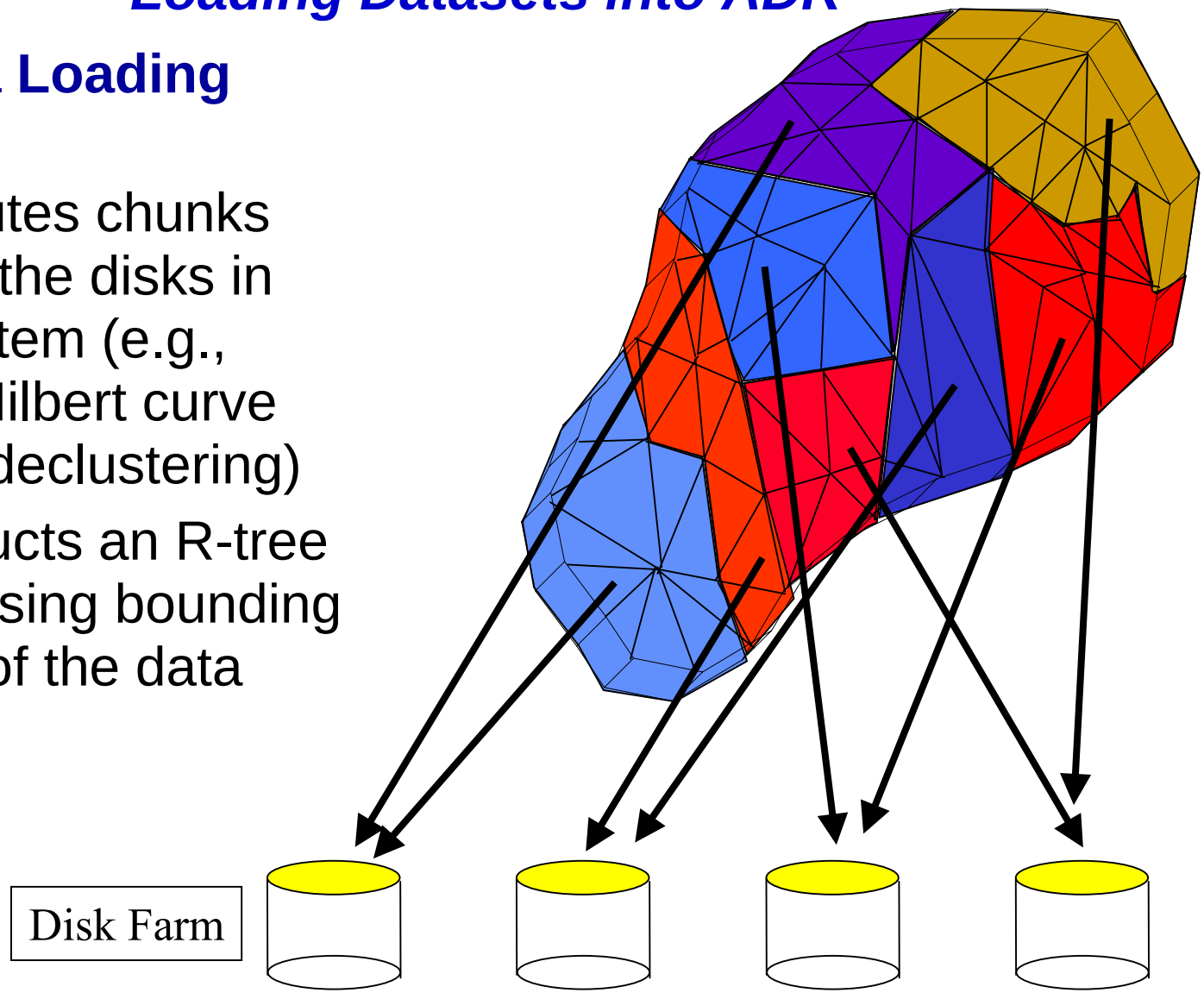


Loading Datasets into ADR

- **A user**
  - should decompose dataset into data chunks
  - optionally can distribute chunks across the disks, and provide an index for accessing them
- **ADR, given data chunks and associated minimum bounding rectangles in a set of files**
  - can distribute data chunks across the disks using a Hilbert-curve based declustering algorithm,
  - can create an R-tree based index on the dataset.

Loading Datasets into ADR

- **ADR Data Loading Service**
  - Distributes chunks across the disks in the system (e.g., using Hilbert curve based declustering)
  - Constructs an R-tree index using bounding boxes of the data chunks



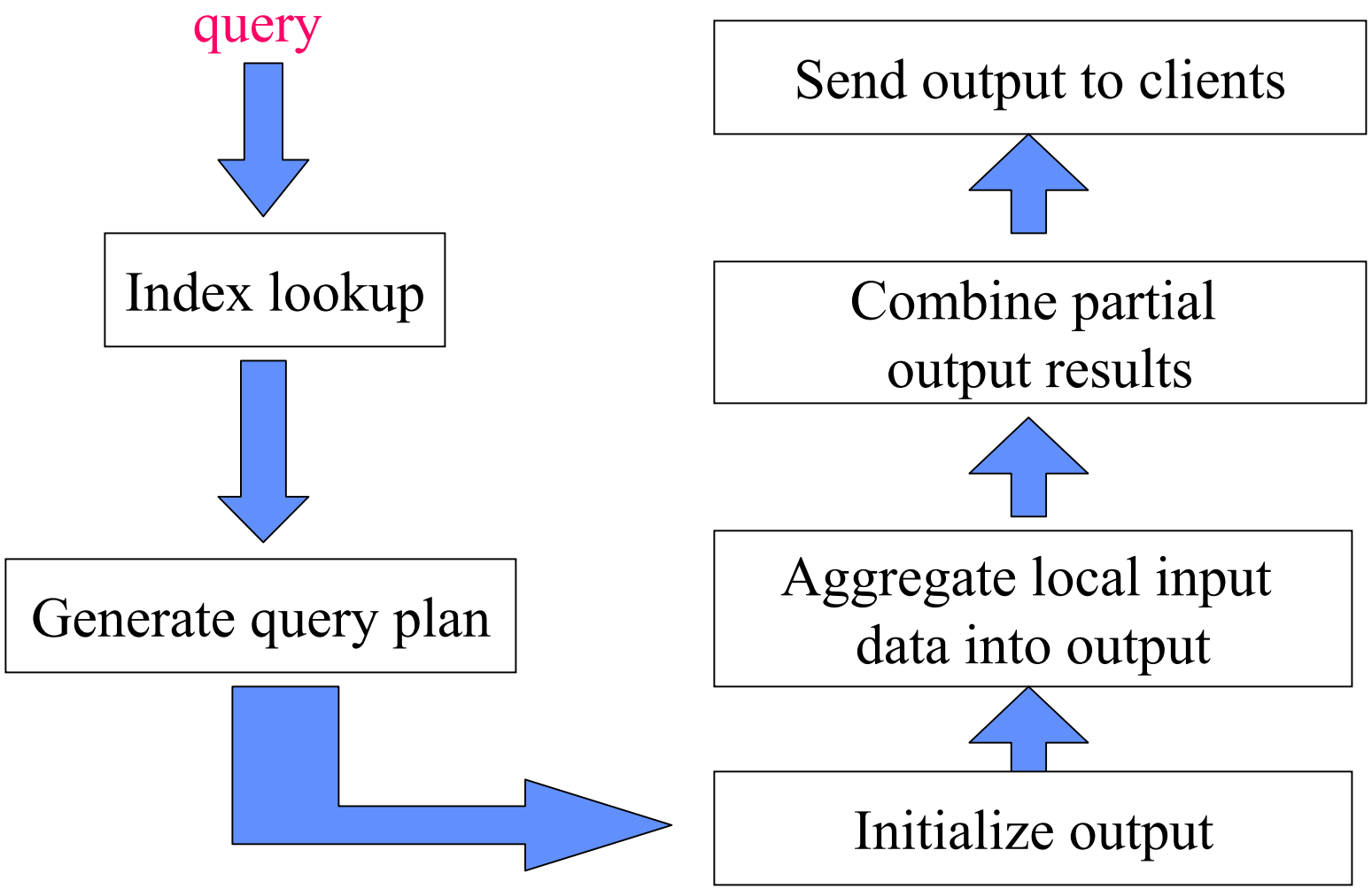
Data Loading Service

- **User must decompose the dataset into chunks**
- **For a fully cooked dataset, User**
  - moves the data and index files to disks (via ftp, for example)
  - registers the dataset using ADR utility programs
- **For a half cooked dataset, ADR**
  - computes placement information using a Hilbert curve-based declustering algorithm,
  - builds an R-tree index,
  - moves the data chunks to the disks
  - registers the dataset

Query Execution in Active Data Repository

- **An ADR Query contains a reference to**
  - the data set of interest,
  - a query window (a multi-dimensional bounding box in input dataset's attribute space),
  - default or user defined index lookup functions,
  - user-defined accumulator,
  - user-defined projection and aggregation functions,
  - how the results are handled (write to disk, or send back to the client).
- **ADR handles multiple simultaneous active queries**

ADR Query Execution



Query Execution

- **For each accumulator tile:**
  - Initialization -- allocate space and initialize
  - Local Reduction -- input data chunks on each processor's local disk -- aggregate into accumulator chunks
  - Global Combine -- partial results from each processor combined
  - Output Handling -- create new dataset, update output dataset or serve to clients

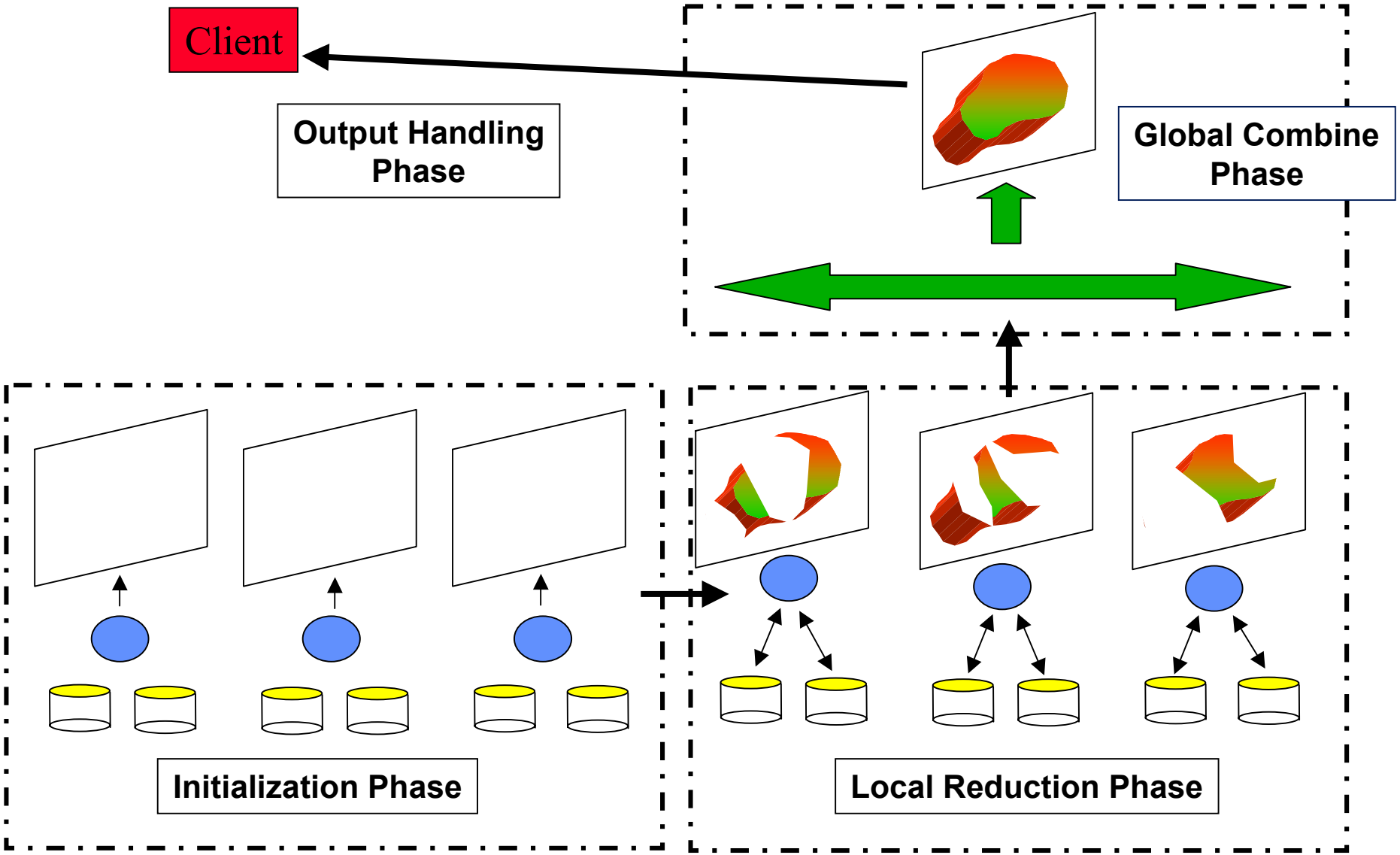
Query Planning Strategies

- **Fully replicated accumulator strategy**
  - Partition accumulator into tiles
  - Each tile is small enough to fit into single processor's memory
  - Accumulator tile is replicated across processors
  - Input chunks living on disk attached to processor P is accumulated into tile on P
  - Global combine employs accumulation function to merge data from replicated tiles

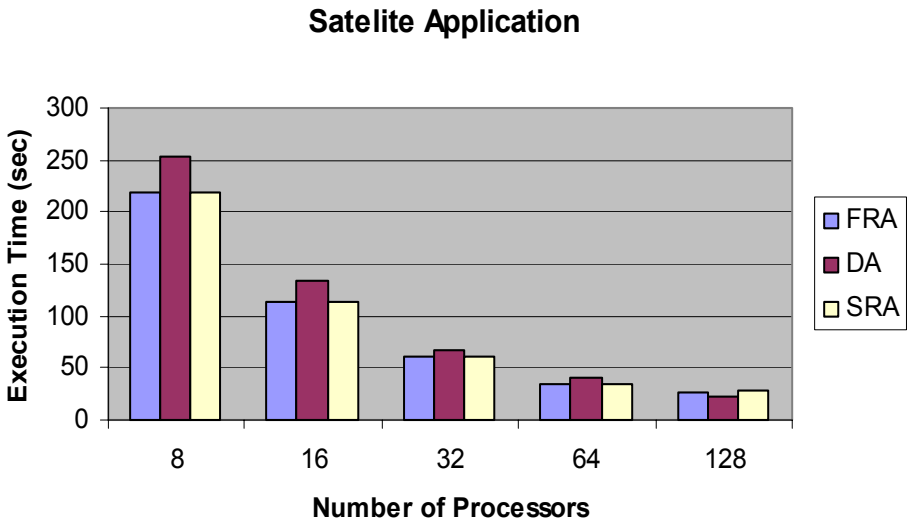
Query Planning Strategies

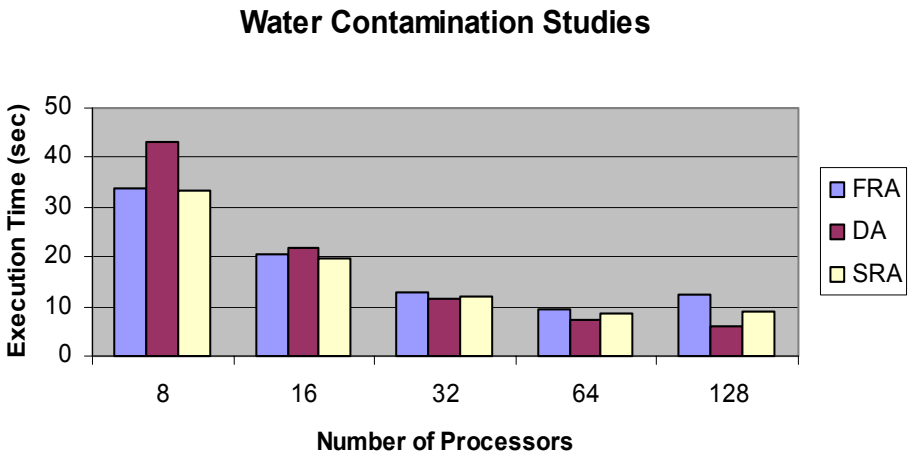
- **Sparsely replicated accumulator strategy**
  - Sparse data structures are used in chunk accumulation
- **Distributed Accumulator Strategy**
  - Partition accumulator between processors
  - Single processor “owns” accumulator chunk
  - Carry out all accumulations on processor that owns chunk

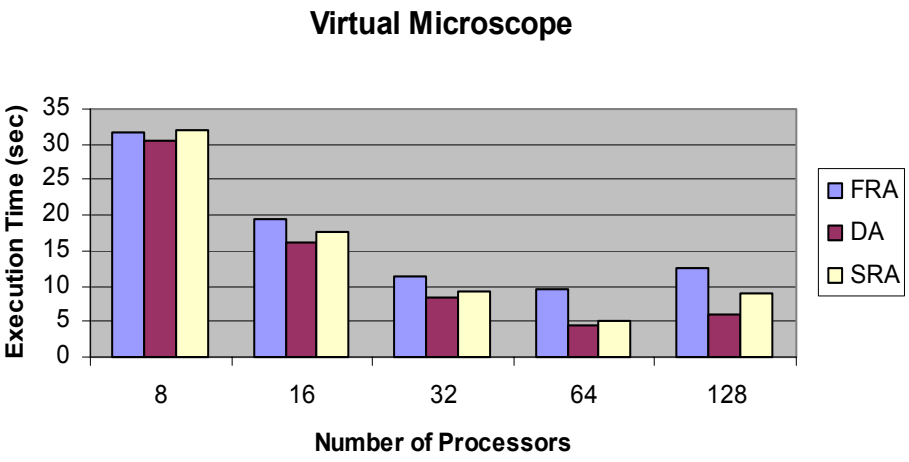
ADR Query Execution



Comparing Execution Strategies





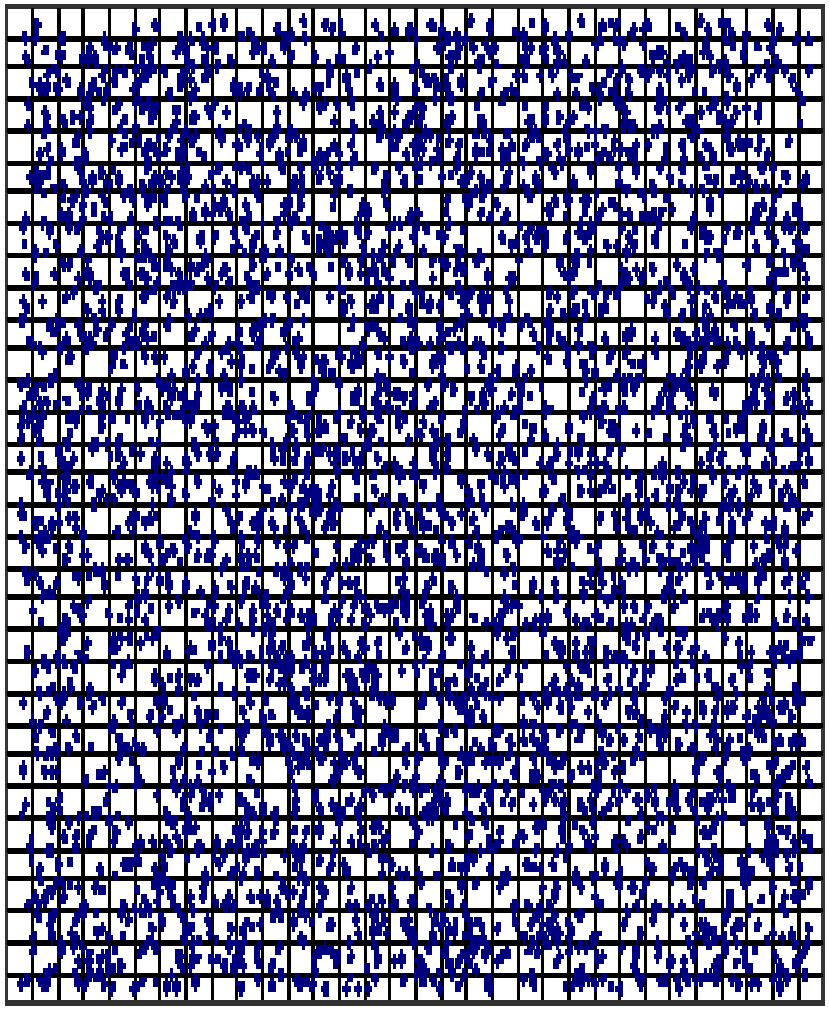


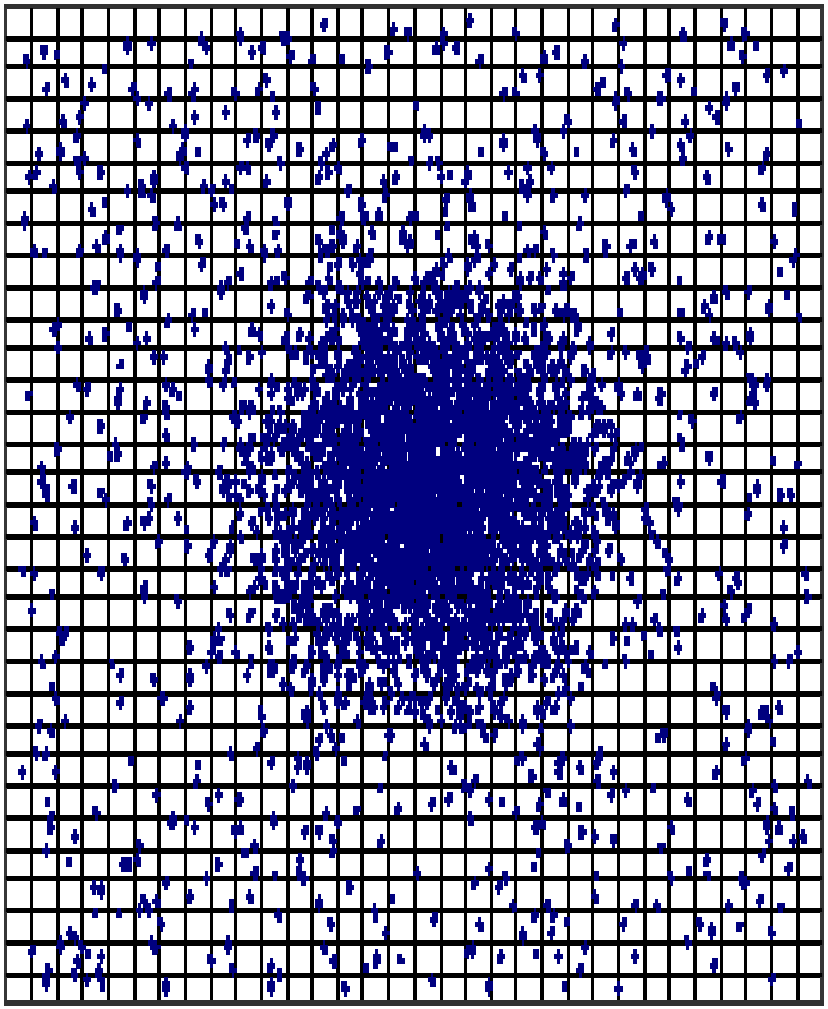
NPACI NATIONAL PART

UNIVERSITY OF MARYLAND

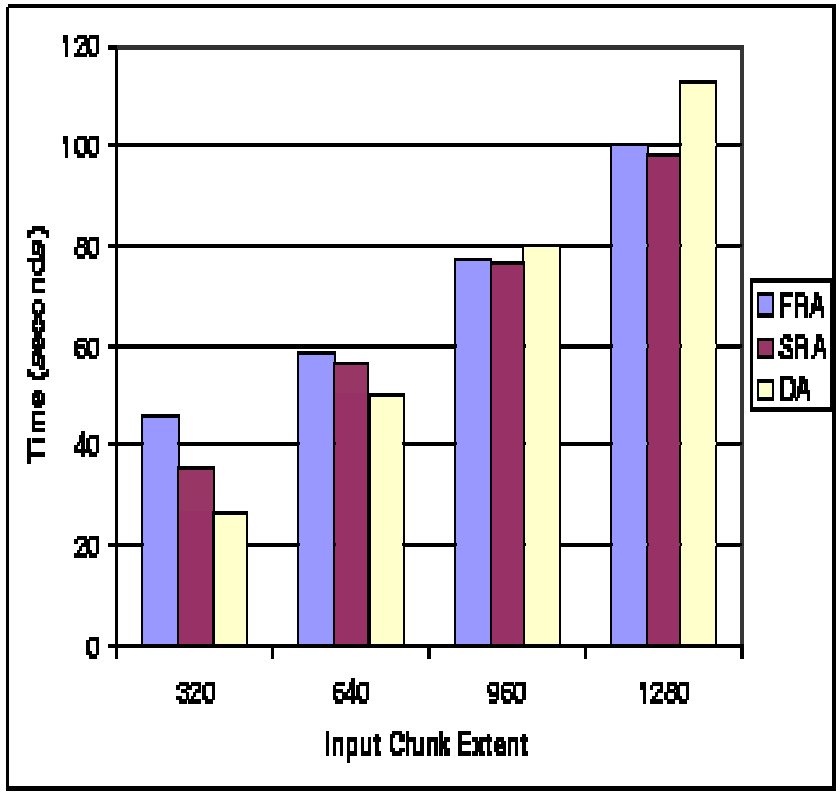
Studies to evaluate query processing strategies

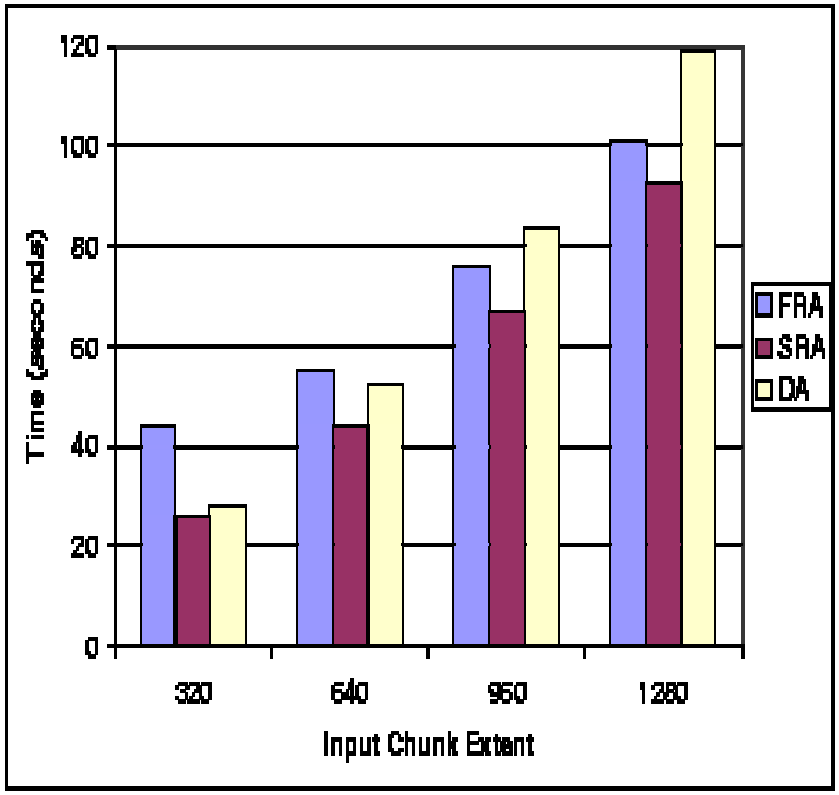
- **Projection of 3-D datasets onto 2-D grid**
- **Query windows of various sizes directed at synthetic datasets with uniform, skewed data distributions**
- **Sparse replicated accumulator wins when there is a high degree of fan-in -- communication can be saved by local accumulation of multiple chunks**
- **Distributed accumulator wins when there is a low degree of fan-in**
  - avoids overhead arising from computation and datastructure manipulations arising from both local accumulation and subsequent combining stage
  - minor decrease in I/O due to bigger tiles





Effect of Accumulator Strategy on Performance





Current Status

- **ADR distributed**
  - supported at NPACI and available from www.cs.umd.edu/projects/adr
- **APIs for customization and among services defined**
  - Visual ADR
- **ADR customizations for applications**
  - done for remote sensing (Titan), Virtual Microscope, bay/estuary simulation, large scale data visualization (ray casting), system analysis of CT data in material science
  - HUBS project will fund support of CBIR and other image classification algorithms

DataCutter

DataCutter

- **A suite of Middleware for subsetting and filtering multi-dimensional datasets stored on archival storage systems**
- **Integrated with NPACI Storage Resource Broker (SRB)**
- **Standalone Prototype**
- **To be layered on Globus (via SRB and standalone version)**

DataCutter

- **Spatial Subsetting using Range Queries**
  - a hyperbox defined in the multi-dimensional space underlying the dataset
  - items whose multi-dimensional coordinates fall into the box are retrieved.
  - Two-level hierarchical indexing -- summary and detailed index files
  - Customizable --
    - Default R-tree index
    - User can add new indexing methods

Processing

- **Processing (filtering/aggregations) through Filters**
  - to reduce the amount of data transferred to the client
  - filters can run anywhere, but intended to run near (i.e., over local area network) storage system
- **Standalone system allows multiple filters placed on different platforms**
- **SRB release allows only a single filter which can be placed anywhere**
- **Motivated by Uysal's disklet work**

Filter Framework

```
class MyFilter : public AS_Filter_Base {
public:
    int init(int argc, char *argv[ ]) { … };
    int process(stream_t st) { … };
    int finalize(void) { … };
}
```

DataCutter -- Subsetting

- **Datasets are partitioned into segments**
  - used to index the dataset, unit of retrieval

- **Indexing very large datasets**
  - Multi-level hierarchical indexing scheme
  - Summary index files -- to index a group of segments or detailed index files
  - Detailed index files -- to index the segments

Placement

- **The dynamic assignment of filters to particular hosts for execution is placement (mapping)**
- **Optimization criteria:**
  - Communication
    - leverage filter affinity to dataset
    - minimize communication volume on slower connections
    - co-locate filters with large communication volume
  - Computation
    - expensive computation on faster, less loaded hosts

Integration of DataCutter with the Storage Resouce Broker

Storage Resource Broker (SRB)

- **Middleware between clients and storage resources**
- **Remote Access to storage resources.**
  - Various types :
    - File Systems - UNIX, HPSS, UniTree, DPSS (LBL).
    - DB large objects - Oracle, DB2, Illustra.
  - Uniform client interface (API).

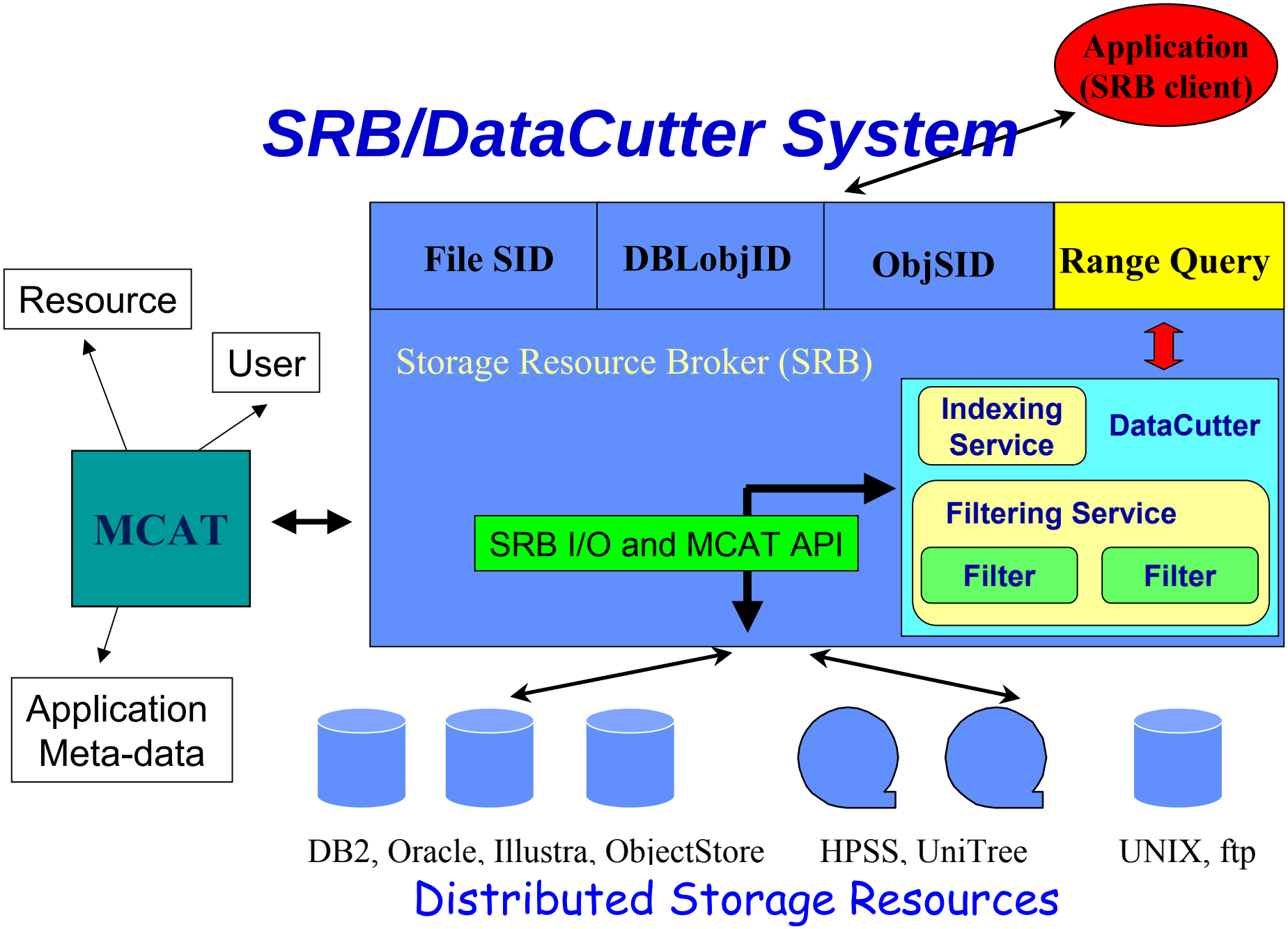
Storage Resource Broker (SRB)

- **MCAT - MetaData Catalog**
  - Datasets (files) and Collections (directories) - inodes and more.
  - Storage resources
  - User information - authentication, access privileges, etc.
- **Software package**
  - Server, client library, UNIX-like utilities, Java GUI
  - Platforms - Solaris, Sun OS, Digital Unix, SGI Irix, Cray T90.

SRB/DataCutter

- **Support for Range Queries**
  - Creation of indices over data sets (composed set of data files)
  - Subsetting of data sets
    - Search for files or portions of files that intersect a given range query
  - Restricted filter operations on portions of files (data segments) before returning them to the client (to perform filtering or aggregation to reduce data volume)

SRB/DataCutter System



SRB/DataCutter Client Interface

- **Creating and Deleting Index**

```
int sfoCreateIndex(srbConn *conn, sfoClass class, int catType,
                   char *inIndexName, char *outIndexName,
                   char *resourceName)
```

```
int sfoDeleteIndex(srbConn *conn, sfoClass class, int catType,
                   char *indexName)
```

SRB/DataCutter Client Interface

- **Searching Index -- R-tree index**

```
int sfoSearchIndex(srbConn *conn, sfoClass class,
                   char *indexName, void *query,
                   indexSearchResult *myresult,
                   int maxSegCount)
```

```
typedef struct {
    int    dim;
    double *min, *max;
} rangeQuery;
```

```
int sfoGetMoreSearchResult(srbConn *conn, int continueIndex,
                           indexSearchResult *myresult,
                           int maxSegCount)
```

Applying Filters

```
int sfoApplyFilter(srbConn *conn, sfoClass class, char *hostName,
                   int filterID, char *filterArg,
                   int numOfInputSegments,
                   segmentInfo *inputSegments,
                   filterDataResult *myresult,
                   int maxSegCount)
```

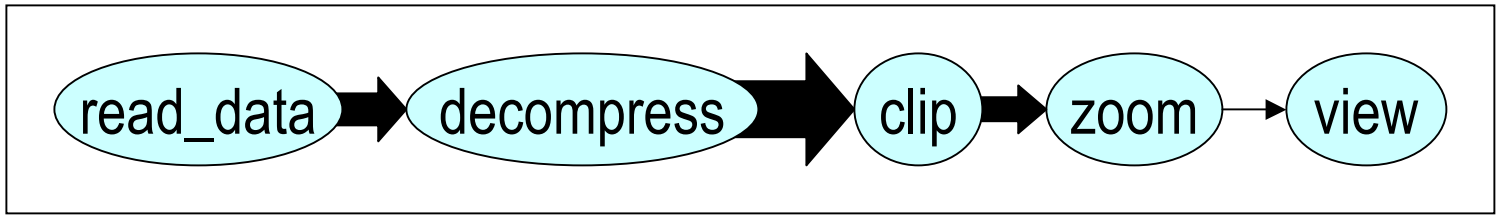
```
int sfoGetMoreFilterResult(srbConn *conn, int continueIndex,
                           filterDataResult *myresult,
                           int maxSegCount)
```

Applying Filters

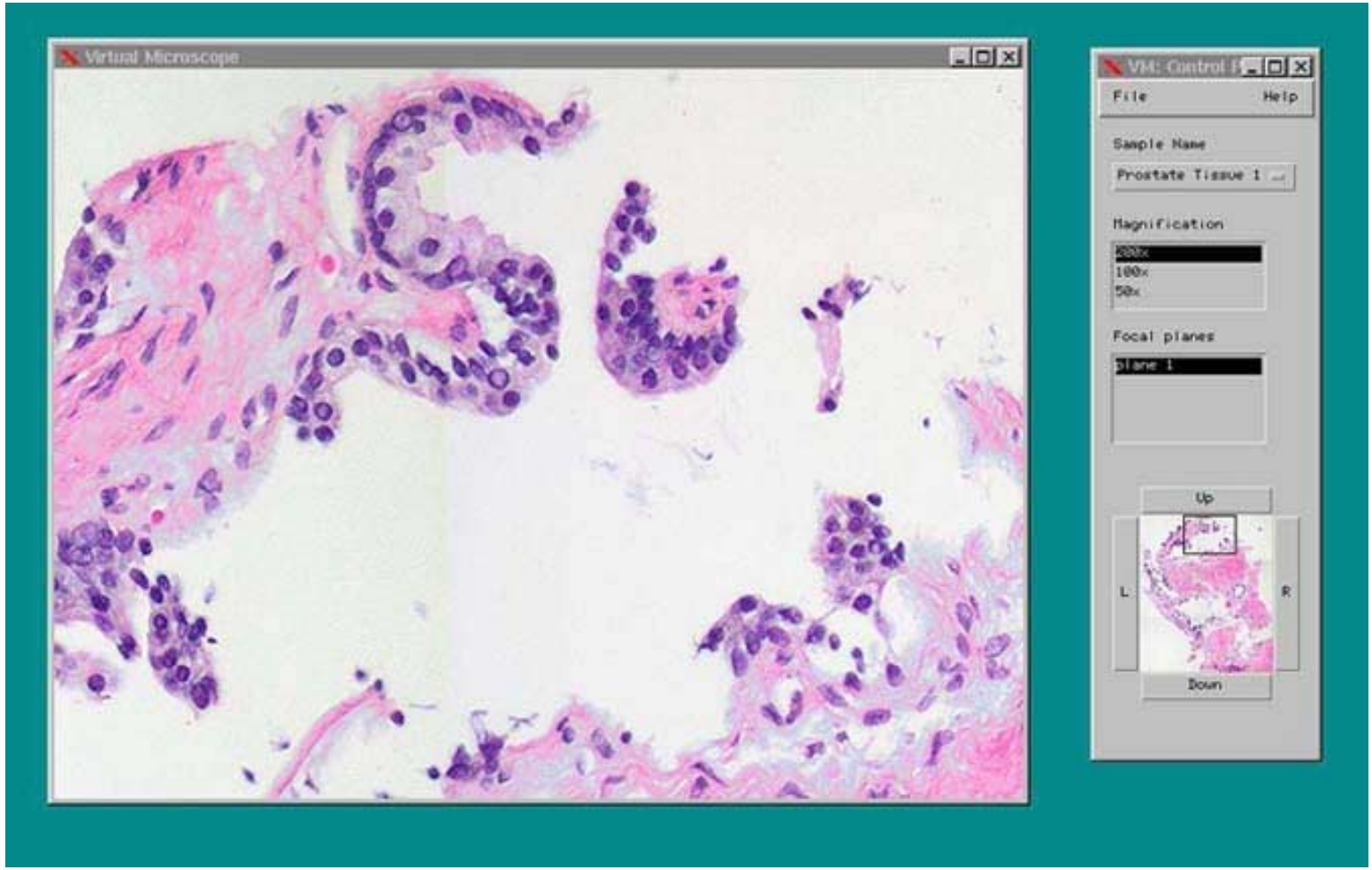
```
typedef struct {
  segmentInfo segInfo;  /* info on segment data buffer after filter oper. */
  char        *segment;     /* segment data buffer after filter is applied     */
} segmentData;

typedef struct {
   int        segmentDataCount;      /* #segments in segmentData array */
   segmentData *segments;            /* segmentData array */
   int        continueIndex;         /* continuation flag */
} filterDataResult;
```

Application: Virtual Microscope

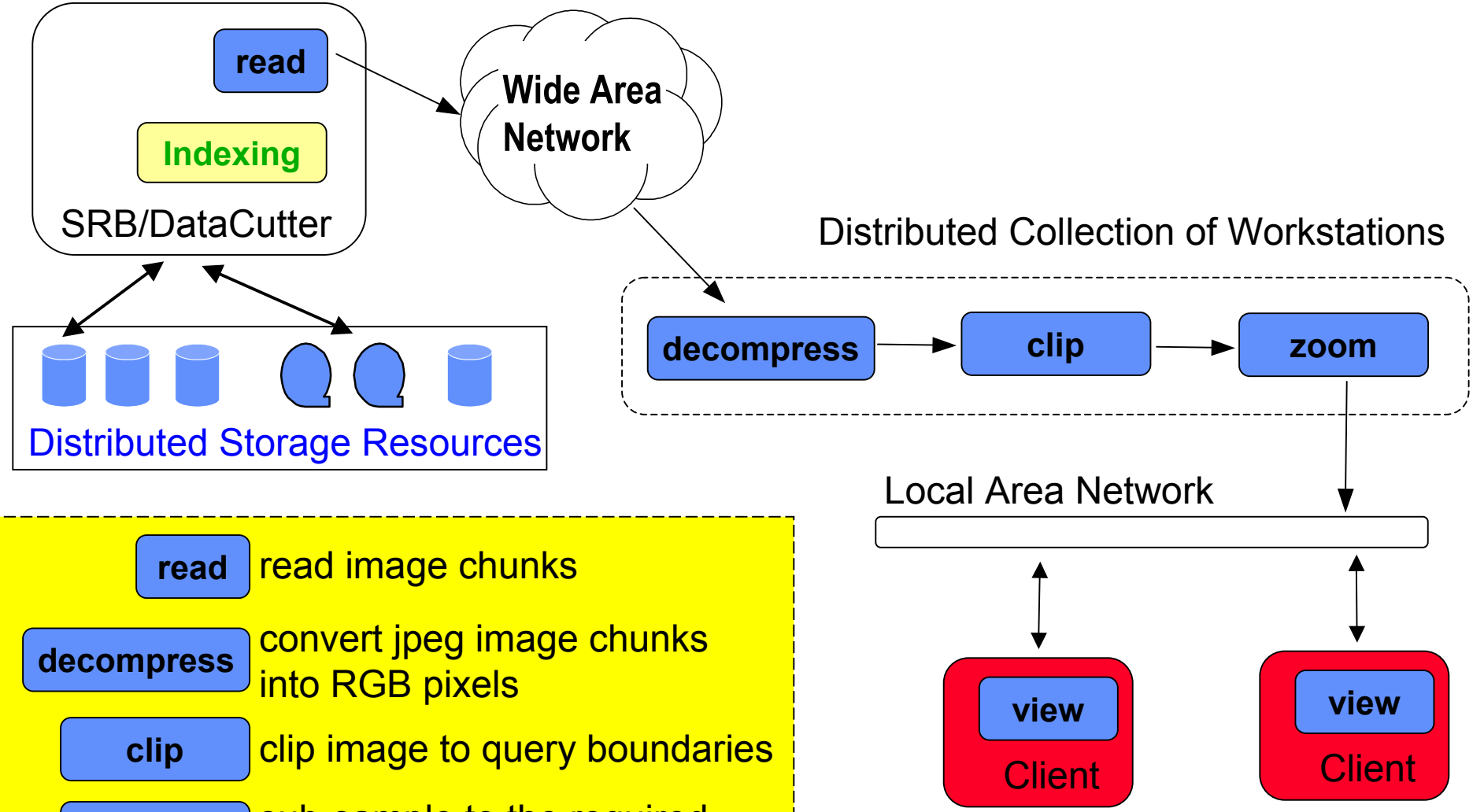
- **Interactive software emulation of high power light microscope for processing/visualizing image datasets**
- **3-D Image Dataset (100MB to 5GB per focal plane)**
- **Client-server system organization**
- **Rectangular region queries, multiple data chunk reply**
- **pipeline style processing**



Virtual Microscope Client



VM Application using SRB/DataCutter



read	read image chunks
decompress	convert jpeg image chunks into RGB pixels
clip	clip image to query boundaries
zoom	sub-sample to the required magnification
view	stitch image pieces together and display image

Experimental Setup

- **UMD 10 node IBM SP (1 4CPU, 3 2CPU, 6 1CPU)**
- **HPSS system (10TB tape storage, 500GB disk cache)**
- **4GB JPEG compressed dataset (90GB uncompressed), 180k x 180k RGB pixels (200 x 200 jpeg blocks of 900x900 pixels each)**
- **250GB JPEG compressed dataset (5.6TB uncompressed), 1.44Mx1.44M RGB pixels (1600x1600 jpeg blocks)**
- **Rtree index based query lookups**
- **server host = SP 2CPU node**
- **Read, Decompress, Clip, Zoom, View distributed between client and server**

Dataset --250 GB (Compressed)
All Computation on Server

Query Size	Cold Disk Cache (Sec)	Warm Disk Cache (Sec)
4500x4500	131	15
9000x9000	244	48
18000x18000	416	100

Breakdown of DataCutter Costs 250 GB dataset, 9600x9600 query

Operation	Cold Cache (Sec)	Warm Cache (Sec)
Total Query+ Compute	244	48
Index lookup	107	3
Data Lookup	115	25

Effect of Filter Placement 9600x9600 Query Warm Cache

	Everything but View on Server (Seconds)	Server:Read Decompress, Clip (Seconds)	Server just reads, client does all else (Seconds)
4.5Kx 4.5K	15	66	14
9.6Kx 9.6K	48	251	46
18Kx 18K	180	991	186

Effect of Dataset Size
4.5Kx4.5K Query
Server does Everything but View
Warm Cache

Dataset Size	Size Uncompressed	Total Time (Sec)	DataCutter Indexing (Sec)	DataCutter Data Retrieval (Sec)
4GB	90GB	49	4	10
250GB	5.6TB	75	5	10

Compiler Techniques for Data Parallel Applications using Very Large Multi-Dimensional Datasets

Department of Computer Science
University of Maryland at College Park

Programming Interface

- **Multi-dimensional collections**
  - Domain
  - RectDomain
- **Foreach loop**
  - iterates over collections
- **Reduction interface**
  - defining reduction variables
    - updated within foreach loops
    - associative and commutative operations
    - only used for self-updates

Canonical Loop

Foreach($r \in R$) {

$O_1\ [S_L(r)]\ \ op_1 = A_1(I_1[S_R(r)], \ldots, I_m[S_R(r)]);$

...

$O_n\ [S_L(r)]\ \ op_n = A_n(I_1[S_R(r)], \ldots, I_m[S_R(r)]);$

}

Virtual Microscope Example

```
public class VMPixel {
  char[3] colors;
  void Initialize(){
    colors[0] = colors[1] = colors[2] = 0;
  }
  void Accum(VMPixel p, int avg) {
    colors[0] += p.colors[0]/avg;
    colors[1] += p.colors[1]/avg;
    colors[2] += p.colors[2]/avg;
  }
}

public class VMPixelOut extends VMPixel
        implements Reducinterface;

Public Class VMScope {
  static Point[2] lowpoint [0,0];
  static Point[2] hipoint[MaxX-1, MaxY-1];
  static RectDomain[2] VMSlide = [lowpoint:hipoint];
  static VMPixel[2d] Vscope = new VMPixel[VMSlide];
```

```
public static void main(String[] args) {
    Point[2] lowend = [args[0], args[1]];
    Point[2] hiend = [args[2],args[3]];
    RectDomain[2] querybox = [lowend:hiend];
    int subsamp = args[4];
    RectDomain[2] OutDomain =
                        [[0,0]:(hiend-lowend)/subsamp];
    VMPixelOut[2d] Output = new VMPixelOut[OutDomain];
    Point[2] p;
    foreach (p in OutDomain)
      Output[p].Initialize();
    foreach (p in querybox) {
      Point[2] q = (p - lowend)/subsamp;
      Output[q].Accum(Vscope[p], subsamp*subsamp);
    }
  }
}
```

Program Slicing

- **Given a *slicing criterion* (s, x), computes the portion of program that computes the value of *x* at statement *s***
- **Extracting customization:**
  - Range Function: loop iterator
  - Subscript Function: index used to access collection
  - Aggregation Function: value assigned to the collection element

Range Function

```
public class VMPixel {
  char[3] colors;
  void Initialize(){
    colors[0] = colors[1] = colors[2] = 0;
  }
  void Accum(VMPixel p, int avg) {
    colors[0] += p.colors[0]/avg;
    colors[1] += p.colors[1]/avg;
    colors[2] += p.colors[2]/avg;
  }
}

public class VMPixelOut extends VMPixel
        implements Reducinterface;

Public Class VMScope {
  static Point[2] lowpoint [0,0];
  static Point[2] hipoint[MaxX-1, MaxY-1];
  static RectDomain[2] VMSlide = [lowpoint:hipoint];
  static VMPixel[2d] Vscope = new VMPixel[VMSlide];
```

```
public static void main(String[] args) {
    Point[2] lowend = [args[0], args[1]];
    Point[2] hiend = [args[2],args[3]];
    RectDomain[2] querybox = [lowend:hiend];
    int subsamp = args[4];
    RectDomain[2] OutDomain =
                        [[0,0]:(hiend-lowend)/subsamp];
    VMPixelOut[2d] Output = new VMPixelOut[OutDomain];
    Point[2] p;
    foreach (p in OutDomain)
      Output[p].Initialize();
    foreach (p in querybox) {
      Point[2] q = (p - lowend)/subsamp;
      Output[q].Accum(Vscope[p], subsamp*subsamp);
    }
  }
}
```

Subscript Function

```
public class VMPixel {
  char[3] colors;
  void Initialize(){
    colors[0] = colors[1] = colors[2] = 0;
  }
  void Accum(VMPixel p, int avg) {
    colors[0] += p.colors[0]/avg;
    colors[1] += p.colors[1]/avg;
    colors[2] += p.colors[2]/avg;
  }
}

public class VMPixelOut extends VMPixel
        implements Reducinterface;

Public Class VMScope {
  static Point[2] lowpoint [0,0];
  static Point[2] hipoint[MaxX-1, MaxY-1];
  static RectDomain[2] VMSlide = [lowpoint:hipoint];
  static VMPixel[2d] Vscope = new VMPixel[VMSlide];
```

```
public static void main(String[] args) {
    Point[2] lowend = [args[0], args[1]];
    Point[2] hiend = [args[2],args[3]];
    RectDomain[2] querybox = [lowend:hiend];
    int subsamp = args[4];
    RectDomain[2] OutDomain =
                        [[0,0]:(hiend-lowend)/subsamp];
    VMPixelOut[2d] Output = new VMPixelOut[OutDomain];
    Point[2] p;
    foreach (p in OutDomain)
      Output[p].Initialize();
    foreach (p in querybox) {
      Point[2] q = (p - lowend)/subsamp;
      Output[q].Accum(Vscope[p], subsamp*subsamp);
    }
  }
}
```

Aggregation Function

```
public class VMPixel {
  char[3] colors;
  void Initialize(){
    colors[0] = colors[1] = colors[2] = 0;
  }
  void Accum(VMPixel p, int avg) {
    colors[0] += p.colors[0]/avg;
    colors[1] += p.colors[1]/avg;
    colors[2] += p.colors[2]/avg;
  }
}

public class VMPixelOut extends VMPixel
        implements Reducinterface;

Public Class VMScope {
  static Point[2] lowpoint [0,0];
  static Point[2] hipoint[MaxX-1, MaxY-1];
  static RectDomain[2] VMSlide = [lowpoint:hipoint];
  static VMPixel[2d] Vscope = new VMPixel[VMSlide];
```

```
public static void main(String[] args) {
    Point[2] lowend = [args[0], args[1]];
    Point[2] hiend = [args[2],args[3]];
    RectDomain[2] querybox = [lowend:hiend];
    int subsamp = args[4];
    RectDomain[2] OutDomain =
                        [[0,0]:(hiend-lowend)/subsamp];
    VMPixelOut[2d] Output = new VMPixelOut[OutDomain];
    Point[2] p;
    foreach (p in OutDomain)
      Output[p].Initialize();
    foreach (p in querybox) {
      Point[2] q = (p - lowend)/subsamp;
☞     Output[q].Accum(Vscope[p], subsamp*subsamp);
    }
  }
}
```

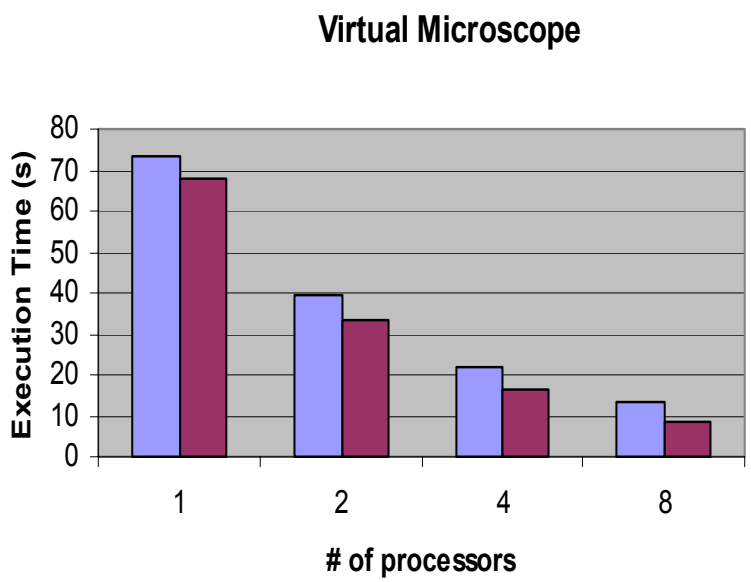
Experimental Compiler

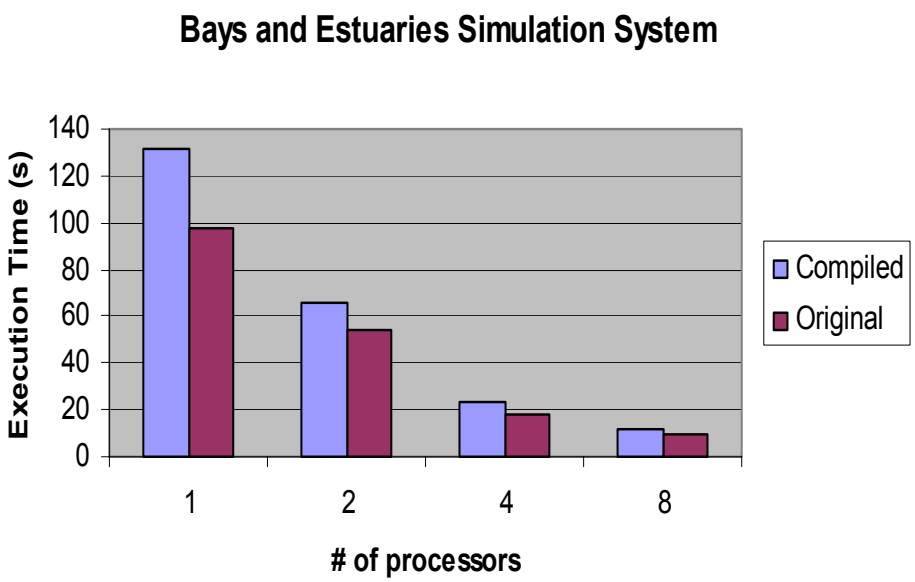
- **Based on Titanium from Berkeley**
  - has extensions suitable to our purposes
  - applications can be defined as objects and operations on objects
  - a number of projects on high performance computing using Java
  - language/programming paradigm becoming popular among application programmers

Experimental Evaluation

- **Compare the performance of compiler generated code with hand-customization**
- **Two different applications**
  - Virtual Microscope
  - Bays and Estuaries Simulation System
- **Experiments on cluster of 400MHz pentiums connected by gigabit ethernet**

Experimental Results





Performance 21% lower on average

The Future

- **Integrated suite of tools for handling very deep memory hierarchies**
  - Common set of tools for grid and disk cache computations
- **Programmability**
  - Use XML metadata
  - Ongoing data parallel compiler project -- uses Java based user defined functions
  - Applications development toolkit (Visual DataCutter)
- **Implementation**
  - NPACI
  - Private sector (?)