Storing and Processing Multi-dimensional Scientific Datasets

Alan Sussman

UMIACS & Department of Computer Science

UNIVERSITY OF MARYLAND

http://www.cs.umd.edu/~als

Data Exploration and Analysis

- Large data collections emerge as important resources
  - Data collected from sensors and large-scale simulations
  - Multi-resolution, multi-scale, multi-dimensional
    - data elements often correspond to points in multi-dim attribute space
    - medical images, satellite data, hydrodynamics data, etc.
  - Terabytes to petabytes today
- Low-cost, high-performance, high-capacity commodity hardware
  - 5 PCs, 5 Terabytes of disk storage for << $10,000

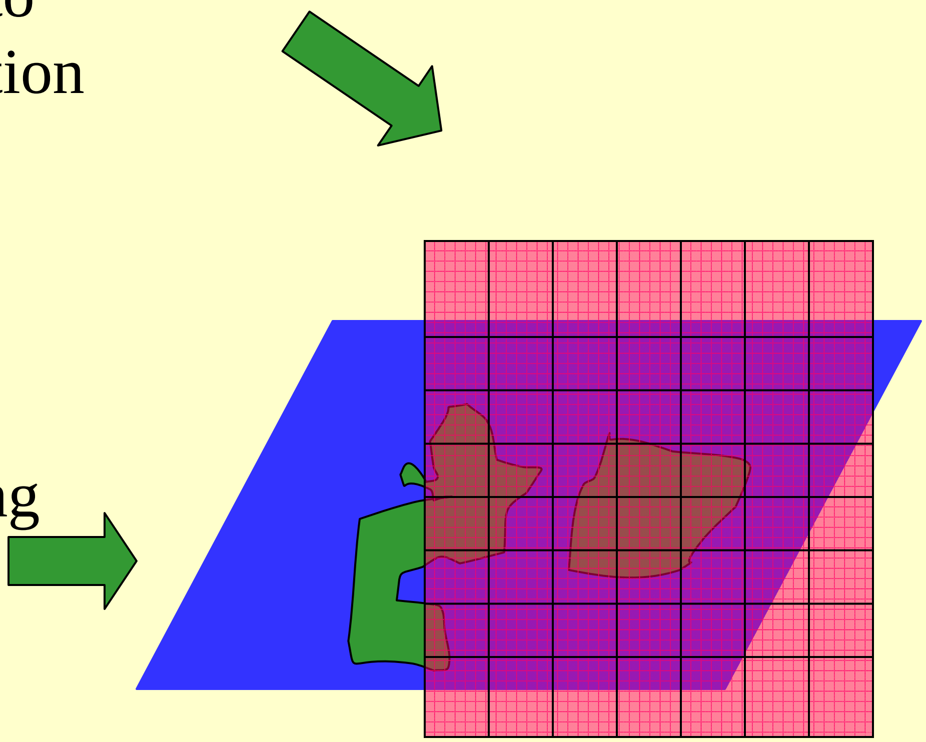
Large Data Collections

- Scientific data exploration and analysis
  - To identify trends or interesting phenomena
  - Only requires a portion of the data, accessed through spatial index
    - e.g., Quad-tree, R-tree
- *Spatial* (range) query often used to specify iterator
  - computation on data obtained from spatial query
  - computation *aggregates* data (MapReduce) - resulting data product size significantly smaller than results of range query

Typical Query

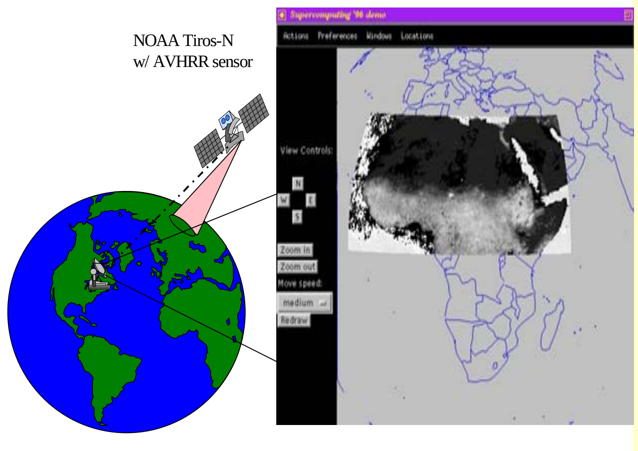
Output grid onto which a projection is carried out

Specify portion of raw sensor data corresponding to some search criterion



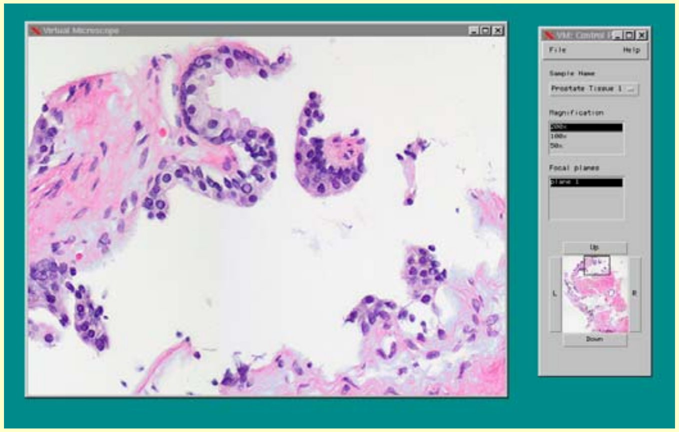
Target example applications

Processing Remotely-Sensed Data

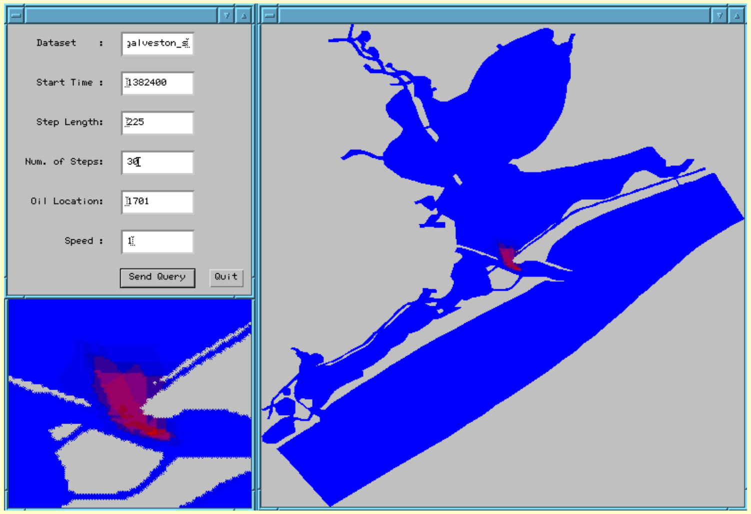


Satellite
Data
Processing

Pathology



Water Contamination Study



Multi-perspective
volume reconstruction

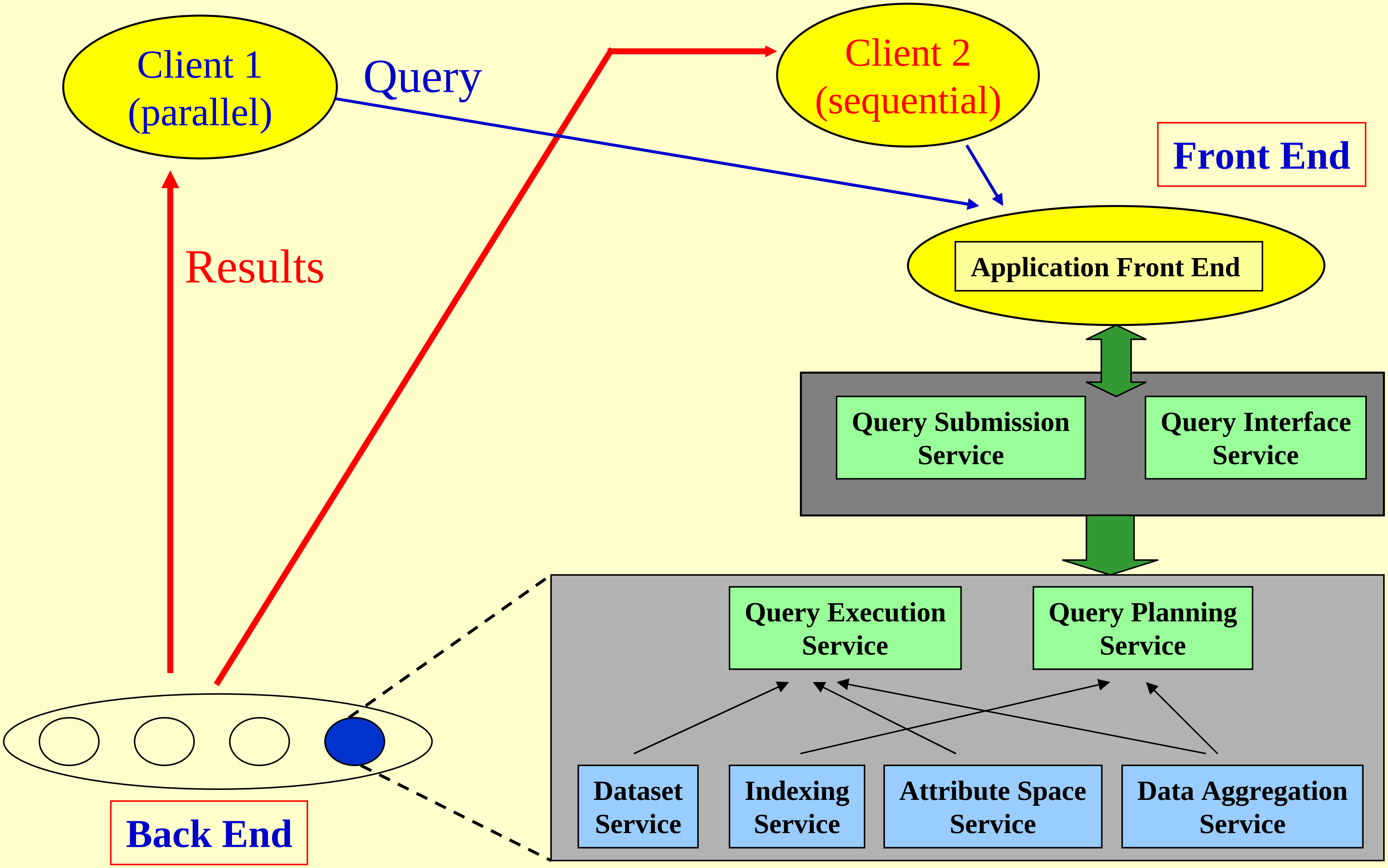
Outline

- Active Data Repository
  - Overall architecture
  - Query planning
  - Query execution
  - Experimental Results
- DataCutter

Active Data Repository (ADR)

- An object-oriented framework (class library + runtime system) for building parallel databases of multi-dimensional datasets
  - enables integration of storage, retrieval and processing of multi-dimensional datasets on distributed memory parallel machines.
  - can store and 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 application specific processing.

ADR Architecture



Active Data Repository (ADR)

- Dataset is collection of user-defined data chunks
  - a data chunk contains a set of data elements
  - multi-dim bounding box (MBR) for each chunk, used by spatial index
  - chunks declustered across disks to maximize aggregate I/O bandwidth
- Separate planning and execution phases for queries
  - Tile output if too large to fit entirely in memory
  - Plan each tile's I/O, data movement and computation
    - Identify all chunks of input that map to tile
    - Distribute processing for chunks among processors
  - All processors work on one tile at a time

Query Planning

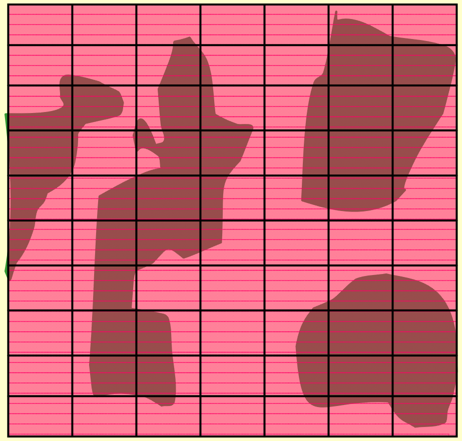
- Index lookup
  - Select data chunks of interest
  - Compute mapping between input and output chunks
- Tiling
  - Partition output chunks so that each tile fits in memory
  - Use Hilbert curve to minimize total length of tile boundaries
- Workload partitioning
  - Each aggregation operation involves an input/output chunk pair
  - Want good load balance and low communication overhead

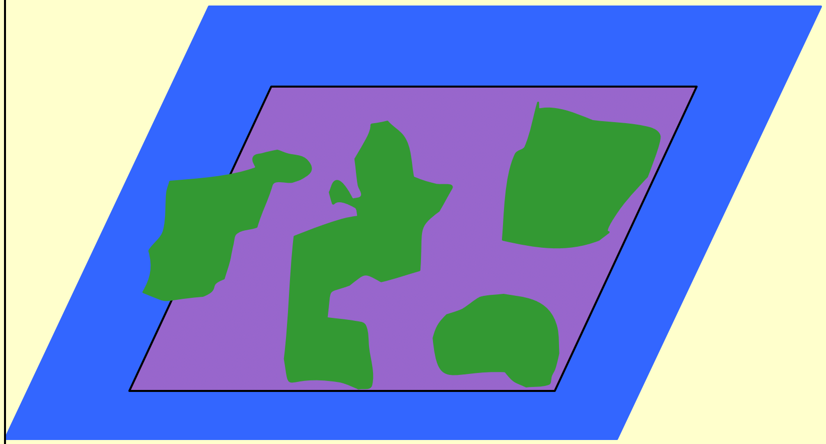
Query Execution

- Broadcast query plan to all processors
- For each output tile:
  - Initialization phase
    - Read output chunks into memory, replicate if necessary
  - Reduction phase
    - Read and process input chunks that map to current tile
  - Combine phase
    - Combine partial results in replicated output chunks, if any
  - Output handling
    - Compute final output values

ADR Processing Loop

```
O ← Output dataset, I ← Input dataset
A ← Accumulator (for 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
```





Query Execution Strategies

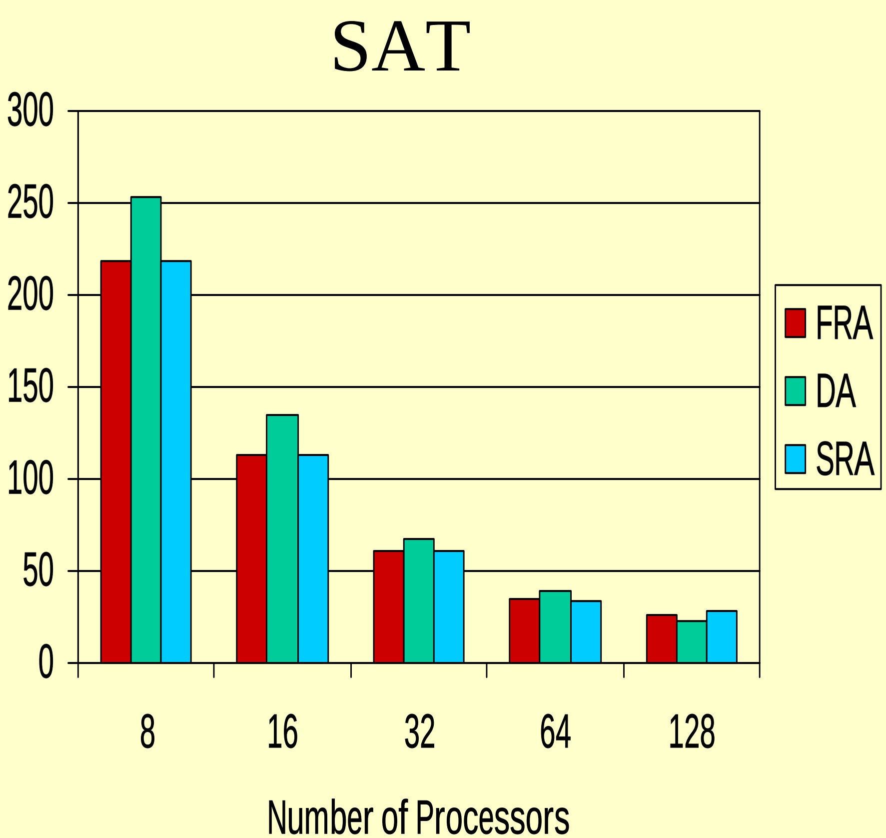
- Distributed Accumulator (DA)
  - Assign aggregation operation to owner of output chunk
- Fully Replicated Accumulator (FRA)
  - Assign aggregation operation to owner of input chunk
  - Requires *combine* phase
- Sparsely Replicated Accumulator (SRA)
  - similar to FRA, but only replicate output chunk when needed

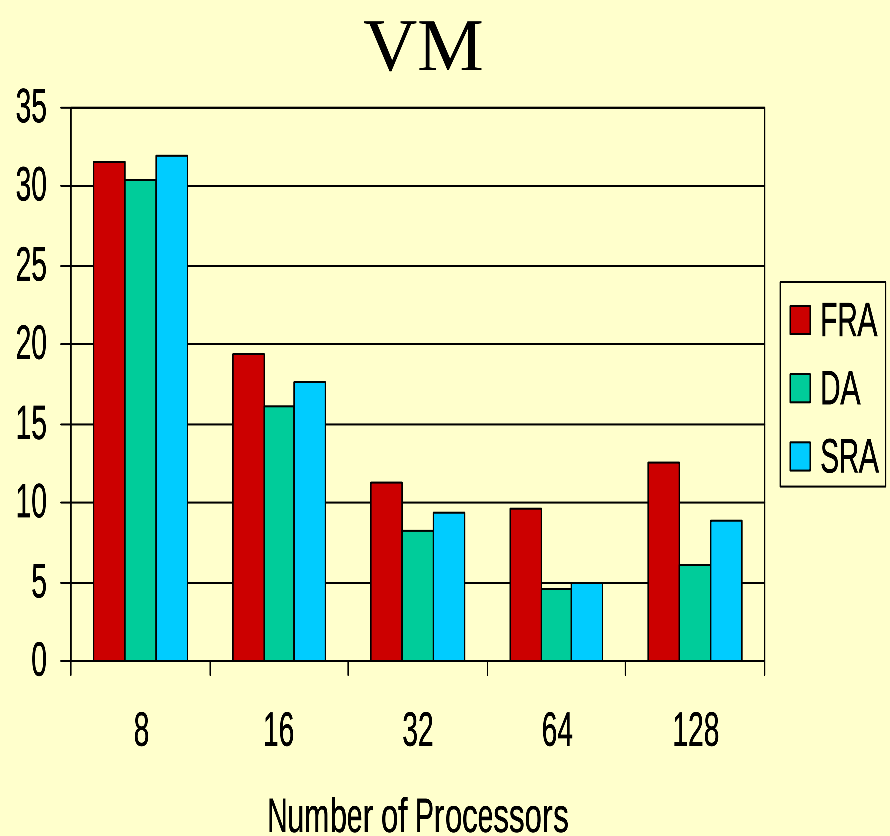
Performance Evaluation

- 128-node IBM SP, with 256MB memory per node
- Datasets generated by Application Emulators
  - Satellite Data Processing (SAT) – *non-uniform mapping*
  - Virtual Microscope (VM)

App	Input	Output	Fan-in	Fan-out (avg)	Comp (ms) t_{init}-t_{red}-t_{comb}
SAT	1.6-26GB	25MB	161-1307	4.6	1-40-20
VM	1.5-24GB	192MB	16-128	1.0	1-5-1

Query Execution Time (sec)





(Fixed input size)

Summary of Experimental Results

- Communication volume
  - Comm. $\text{Volume}_{DA} \propto$ fan-out
  - Comm. $\text{Volume}_{FRA/SRA} \propto$ fan-in
- DA may have computational load imbalance due to non-uniform mapping
- Relative performance depends on
  - Query characteristics (e.g., fan-in, fan-out)
  - Machine configurations (e.g., number of processors)
- No strategy always outperforms the others

ADR queries vs. Other Approaches

- Similar to out-of-core reductions (more general MapReduce)
  - Commutative & associative
  - Most reduction optimization techniques target in-core data
  - Out-of-core techniques require data redistribution
- Similar to relational group-by queries
  - Distributive & algebraic [Gray96]
  - spatial-join + group-by
  - For ADR, output data items and extents known prior to processing

```
double x[max_nodes],
          y[max_nodes];
integer ia[max_edges],
          ib[max_edges];
for (i=0; i<max_edges; i++)
   x[ia[i]] += y[ib[i]];
```

```
Select      Dept, AVG(Salary)
From        Employee
Group By    Dept
```

Outline

- Active Data Repository
- DataCutter
  - Architecture
  - Filter-stream programming
  - Group Instances
  - Transparent copies

Distributed Grid Environment

Heterogeneous Shared Resources:

- Host level: machine, CPUs, memory, disk storage
- Network connectivity

Many Remote Datasets:

- Inexpensive archival storage
- *Islands* of useful data
- Too large for replication

DataCutter

- Target same classes of applications as ADR

Indexing Service

- Multi-level hierarchical indexes based on spatial indexing methods – e.g., R-trees
  - Relies on underlying multi-dimensional space
  - User can add new indexing methods

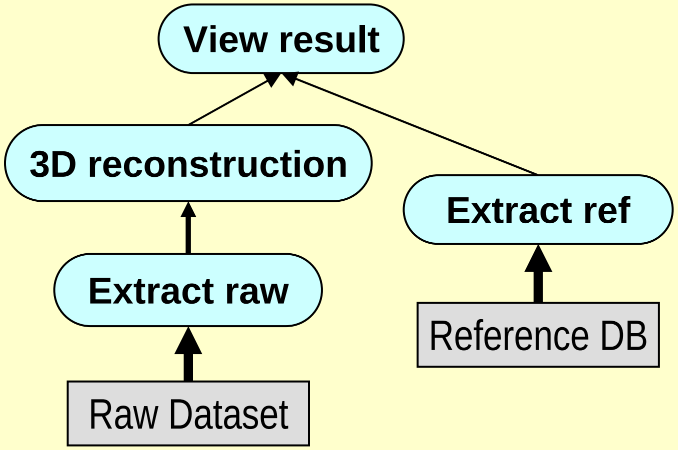
Filtering Service

- Distributed C++ (and Java) *component* framework
- Transparent tuning and adaptation for heterogeneity
- Filters implemented as threads – 1 process per host

Filter-Stream Programming (FSP)

Purpose: Specialized components for processing data

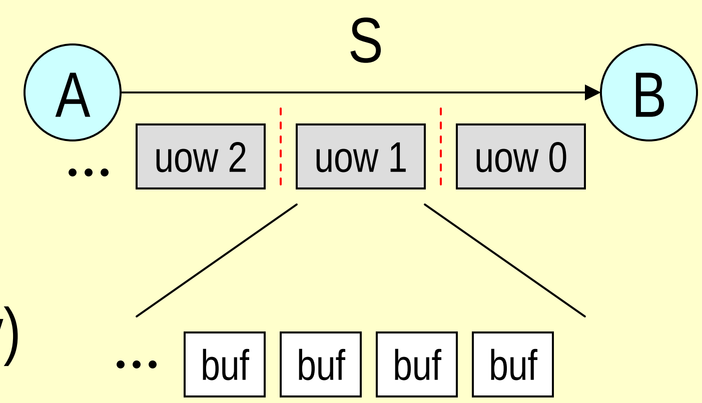
- based on Active Disks research [Acharya, Uysal, Saltz: ASPLOS'98], macro-dataflow, functional parallelism
- filters – logical unit of computation
  - high level tasks
  - init,process,finalize interface
- streams – how filters communicate
  - unidirectional buffer pipes
  - uses fixed size buffers (min, good)
- users specify filter connectivity and filter-level characteristics



FSP: Abstractions

Filter Group

- logical collection of filters to use together
- application starts filter group *instances*



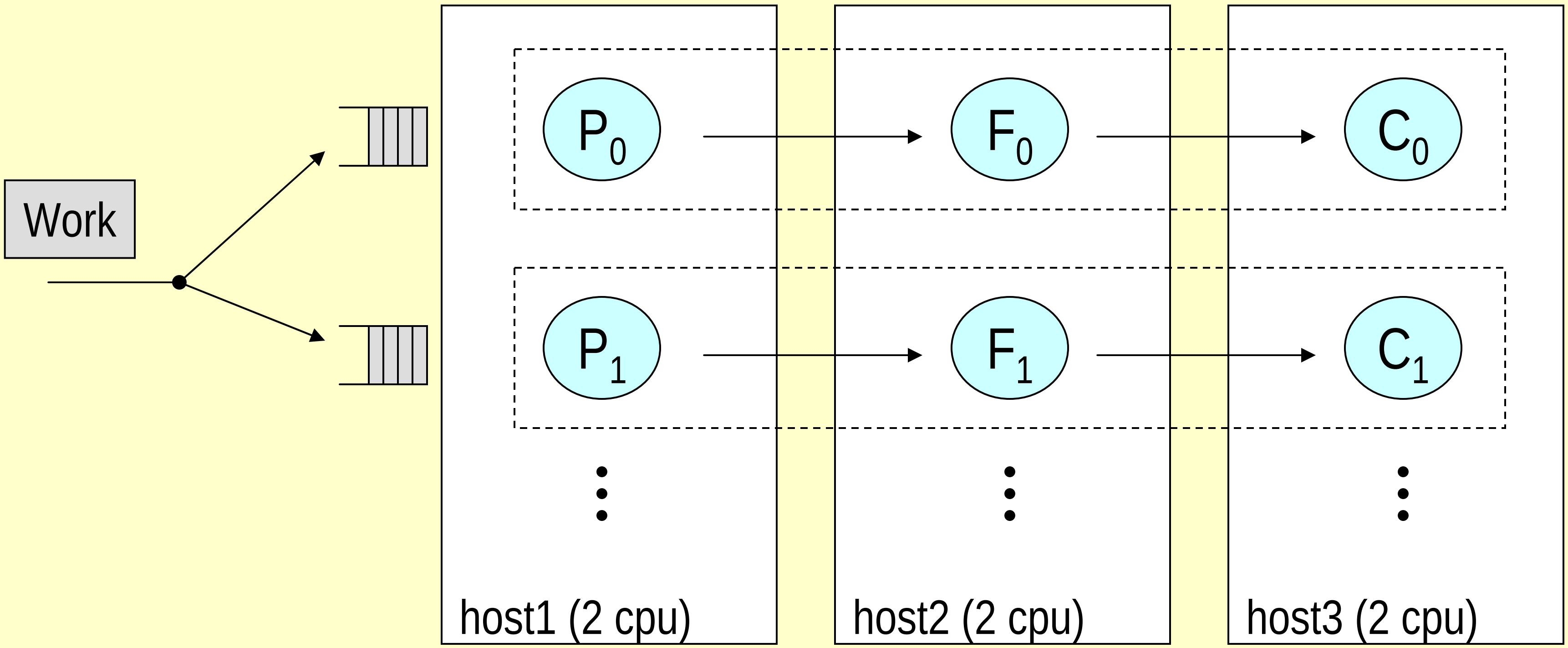
Unit-of-work cycle

- "work" is application defined (ex.: a query)
- work is appended to running instances
- init(), process(), finalize() called for each uow
- process() returns { EndOfWork | EndOfFilter }
- allows for adaptivity

Optimization Techniques

- Mapping filters to hosts
  - allow components to execute concurrently
- Multiple filter group instances
  - allow work to be processed concurrently
- Transparent copies
  - keep pipeline full by avoiding filter processing imbalance and use write policies to deal with dynamic buffer distribution
- Application memory tuning
  - minimize resource usage to allow for copies

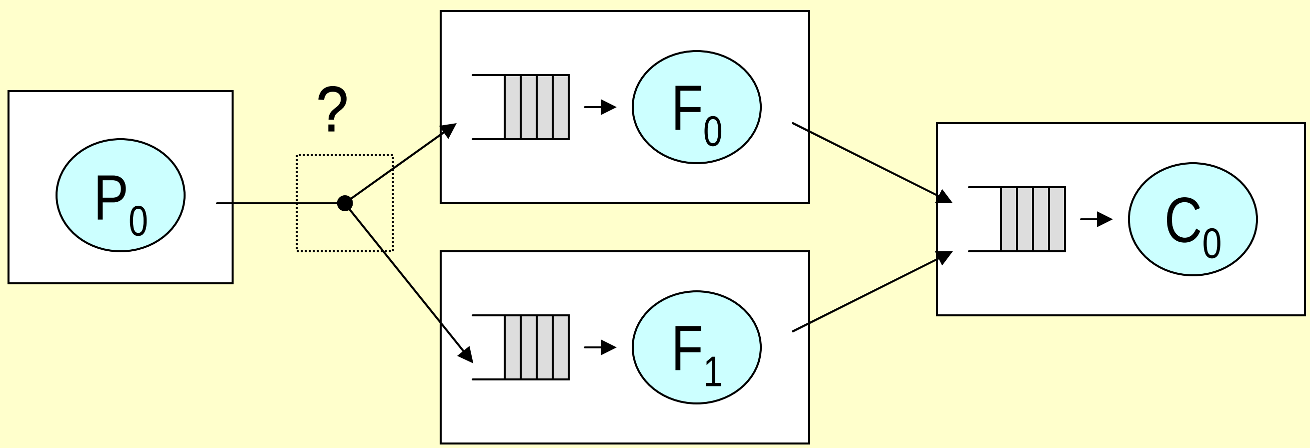
Optimization - Group Instances



Match # instances to environment (CPU capacity, network)

Transparent Copies

- replicate filters *within* an instance (intra-work)
- *write policy* to distribute work buffers to copies
  - shared queue within host
  - across hosts - round robin (RR), weighted RR (WRR), demand-driven (DD), user-defined (UD)
- single stream illusion, $UOW_i < UOW_{i+1}$
- state consistency problems addressed by a **merge** step



Runtime Pipeline Balancing

Use local information:

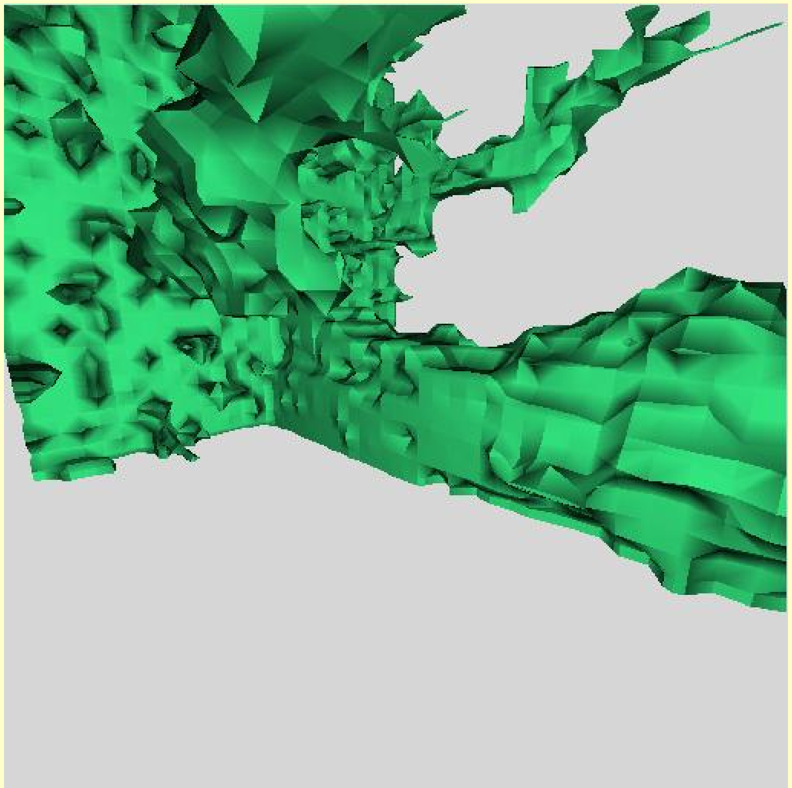
- queue size, send time / receiver acks

- Adjust number of transparent copies
- Demand based dataflow (choice of consumer)
  - Within a host – perfect shared queue among copies
  - Across hosts
    - Round Robin (RR)
    - Weighted Round Robin (WRR)
    - Demand-Driven (DD) sliding window (buffer consumption rate)
    - User-defined

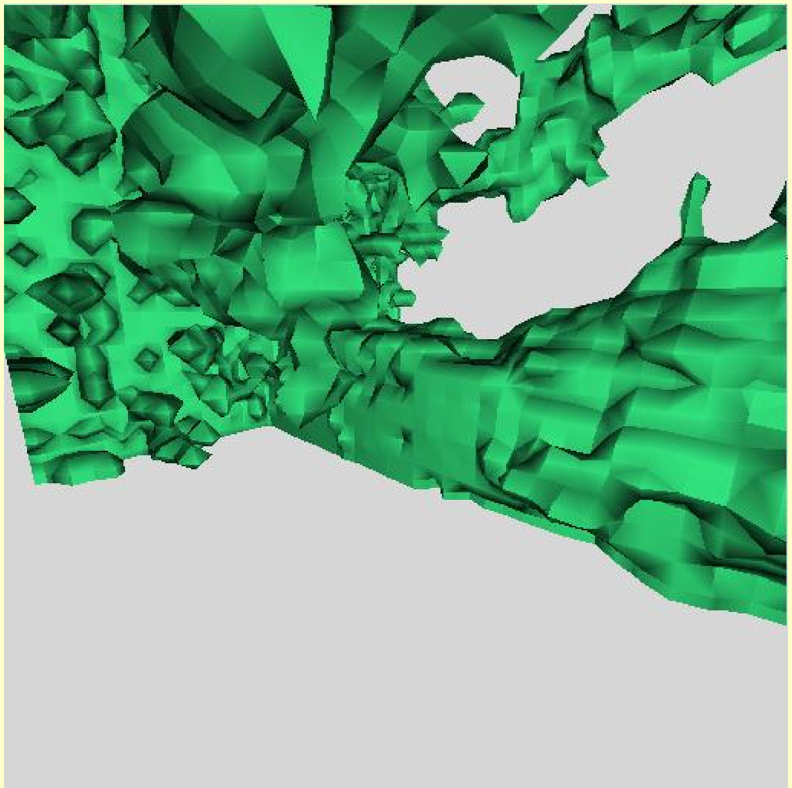
Experiment – Isosurface Rendering

- Isosurface rendering on Red/Blue Linux cluster at Maryland
  - Red – 16 2-processor PII-450, 256MB, 18GB SCSI disk
  - Blue – 12 2-processor PIII-550, 1GB, 2-8GB SCSI disk + 1 8-processor PIII-550, 4GB, 2-18GB SCSI disk
  - Connected via Gigabit Ethernet
- UT Austin ParSSim chemical species transport simulation
  - Single time step 3D visualization, read all data for 1 time step
- Two implementations of Raster filter – z-buffer and active pixels

Sample Isosurface Visualization



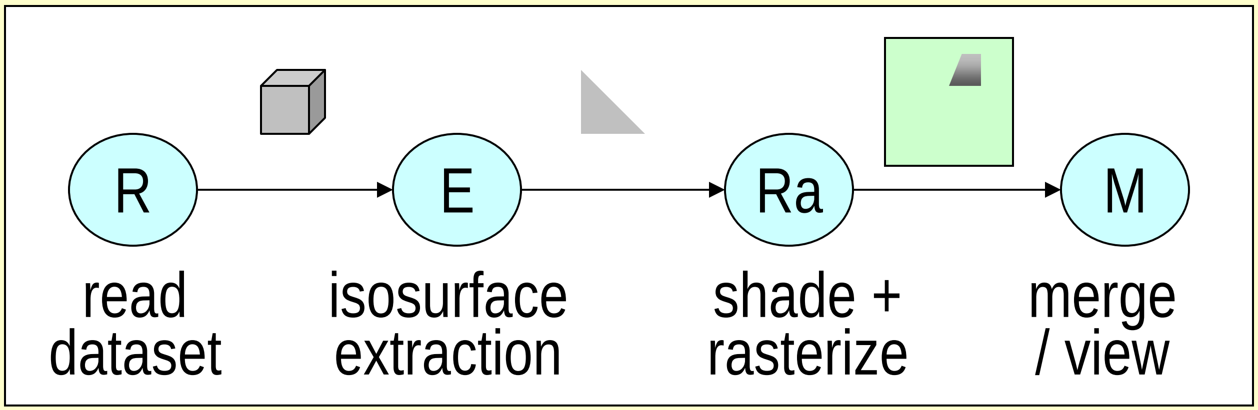
V = 0.35



V = 0.7

Experimental setup

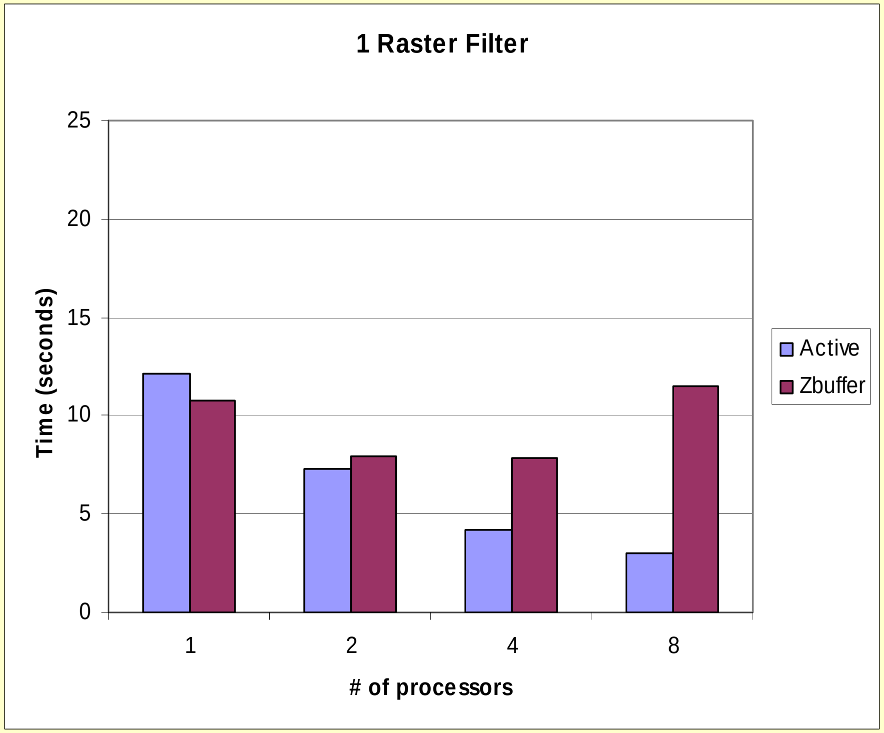
			32.0MB	Z-buffer	
150 MB	38.6 MB	11.8 MB	28.5 MB	Active Pixel	

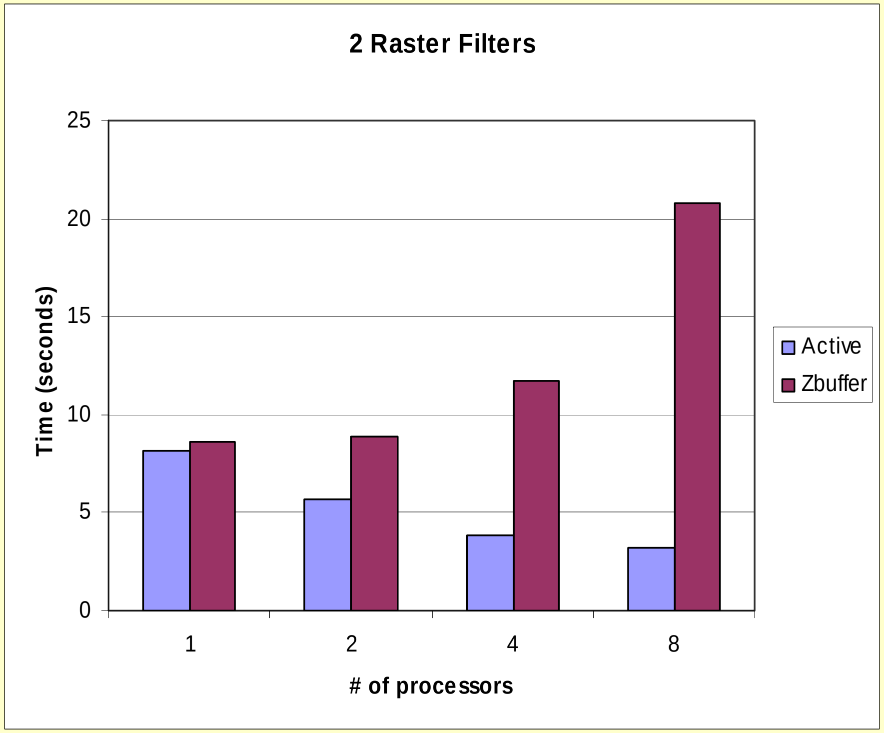


Active Pixel	0.64s	1.64s	11.67s	0.73s	= 14.68s
Z-buffer	0.68s	1.65s	9.43s	0.90s	= 12.66s

Experiment to follow combines R and E filters, since that showed best performance in experiments not shown

Active Pixel vs. Z-Buffer

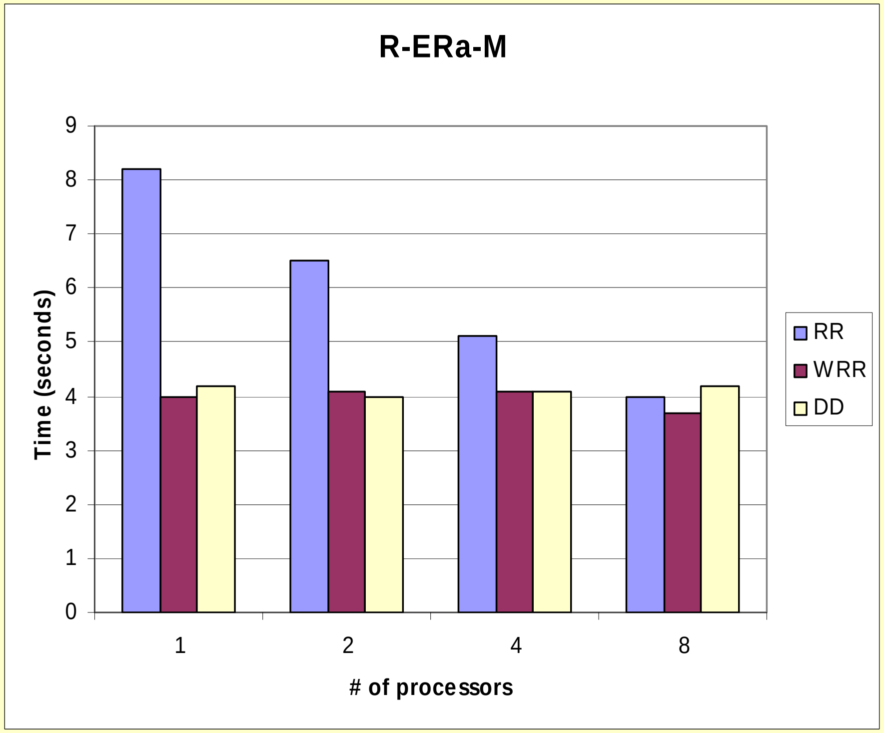


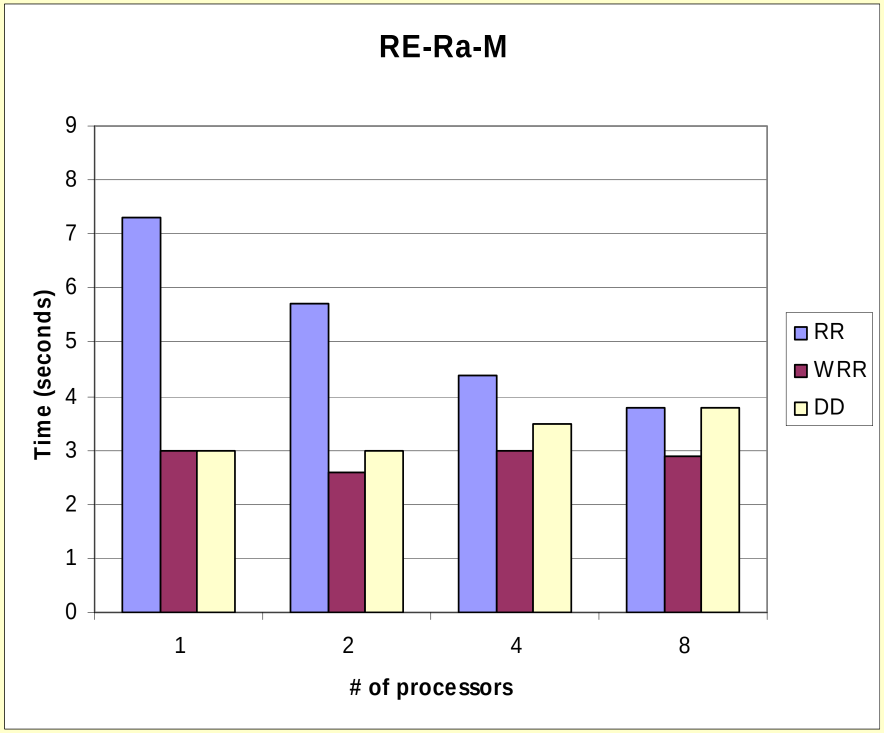


Configuration: RE-Ra-M

Only Red nodes used – each one runs 1 RE, 1 or 2 RA, and one node runs M

Heterogeneous Nodes



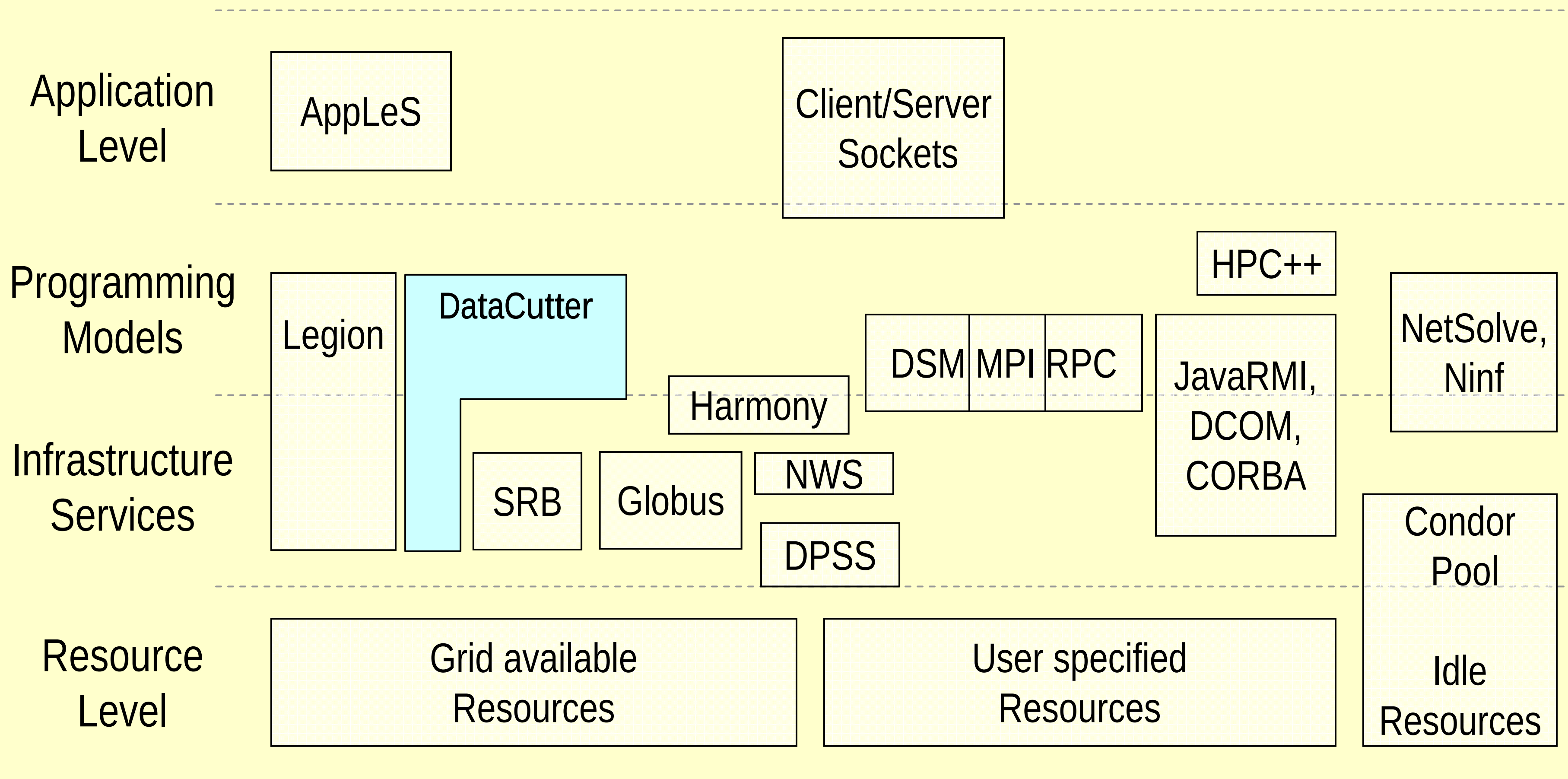


Active Pixel algorithm on 8-processor Blue node + Red data nodes
Blue node runs 7 Ra or ERa copies and M, Red nodes each run 1 of each except M

Summary of Results

- Placement matters
  - Heterogeneity of shared resources, data volume
- More instances and transparent copies
  - Balance applications for heterogeneity
- No static choice will work
  - Runtime heterogeneity and dynamic shared resources

DataCutter as a Grid Service



Acknowledgments

- Students
  - Chialin Chang – ADR
  - Michael Beynon, Renato Ferreira – DataCutter
- Other faculty and postdocs (now at Ohio State)
  - Joel Saltz
  - Tahsin Kurc
  - Umit Catalyurek