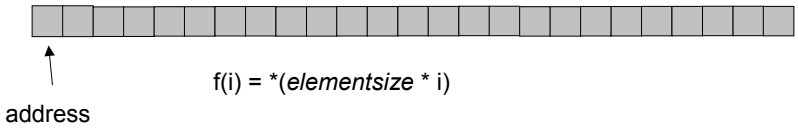
Querying Multidimensional Arrays

Outline

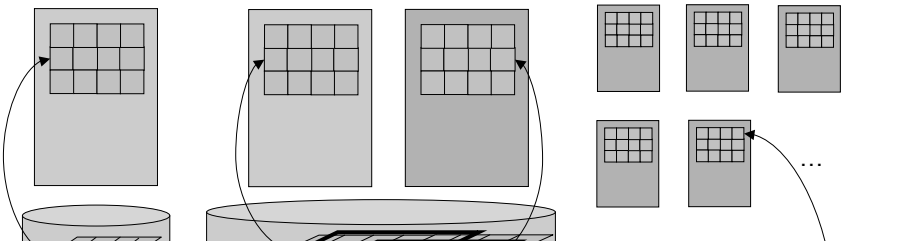
- Models and Languages for Querying Arrays
- Efficient Array Storage and Access

Ordinary Arrays

- Usually a language feature implying at least two things:
  - A **function** `f : Int -> Value`
  - A **performance contract**
    O(1) access to read/write any element
- In C?
- An address, an element type, pointer deref



Large, Shared Arrays



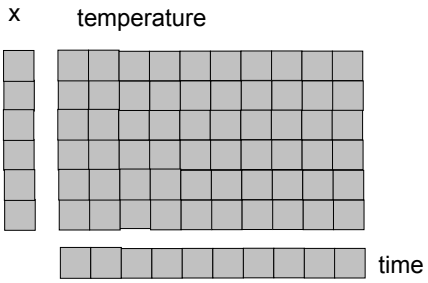

```
for (i=0; i<n; i++) {
  for (j=0; j<m; j++) {
     out[i][j] = in[i][j] * 5;
  }
}
```

Managing Arrays

- File formats (with an API)
  - netCDF, HDF, FITS
- Languages with persistence features
  - MATLAB, APL, others
- Database Extensions

```
SELECT img[23:45, 100:150].g * 20
  FROM SatelliteImages s
 WHERE img[10:20, 40:50].b > 13.4
```

Arrays in Files: netCDF



Arrays in Files: netCDF

```
netcdf temp.cdf {
dimensions:
        x_coord = 3 ;
        time = UNLIMITED ;
variables:
        float time(time) ;
                time:units = “seconds” ;
        float x_coord(x_coord) ;
                x_coord:units = “meters” ;
                x_coord:attribute2 = 1.003f ;
        float temperature(time, x_coord) ;
                 temperature:units = “celsius” ;
// global attributes
        :name = “temperature measurements” ;
        :calibration date = 1/23/2006 ;
data:
        x_coord       = 2.34, 2.36, 2.37 ;
        time           = 1.0, 2.5, 3.7, 7.0 ;
        temperature = 34.5, 31.2, 23.7, 19.6, 18.5, 17.1, 22,9,
29.9, 31.3, 34.5, 34.3, 33.7 ;
}
```

dimension names

variables

by convention, one variable per dimension has the same name as the dimension.

type

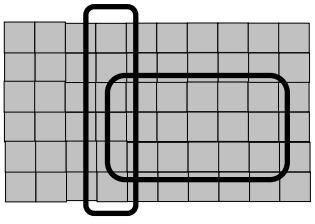
v(x,t) means v is a function of x, t

metadata

not actually in ASCII!

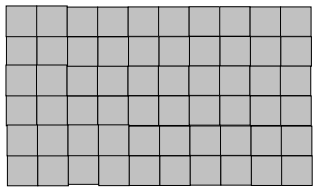
netCDF and HDF APIS

- Read/Write metadata
- Read/Write whole datasets
- Read/Write element
- Read/Write slices
- Min/Max over dims



- Is this enough?

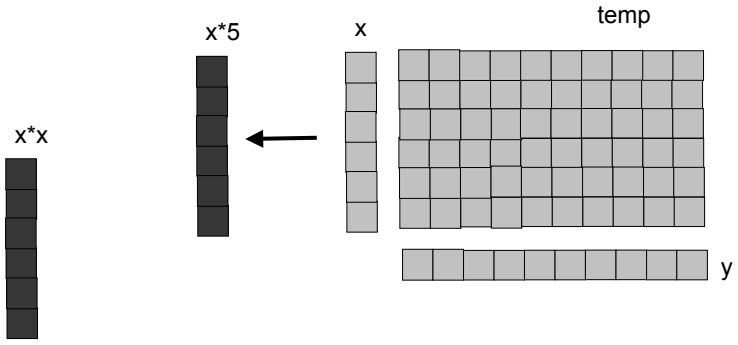
Design an Array Algebra



?

- Possible design goals:
  - small number of operators
  - closed (operators return arrays)
  - write down operator signatures:
    Ex: *item(A, i)* =
    the value of A at position i

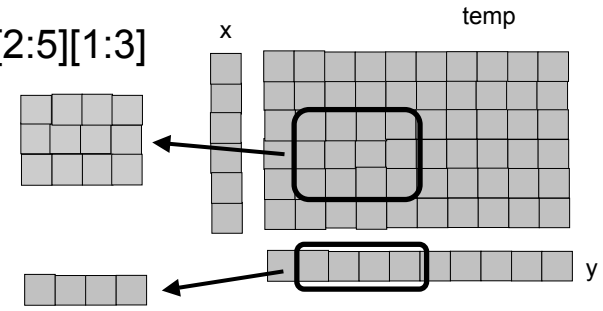
Arithmetic



Query-by-Structure

temp[2:5][1:3]

y[1:4]



“slice”, “subslab”, “section”, …

Query-by-value?

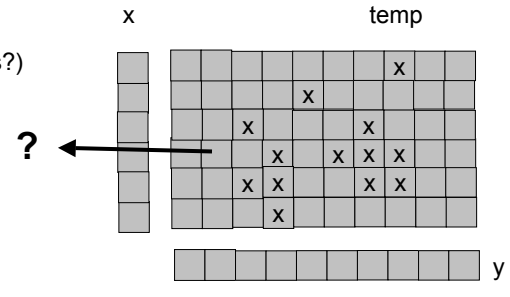
where temp[i,j] < 10

Array of elements?
(with what dimensions?)

List of elements?
(In what order?)

Bag of elements?

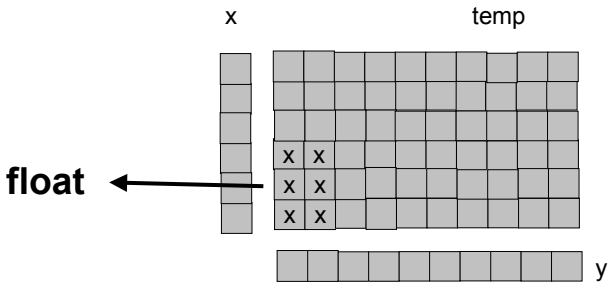
Set of tuples (i, j, t)?



query-by-value operations over arrays are not closed

Aggregation

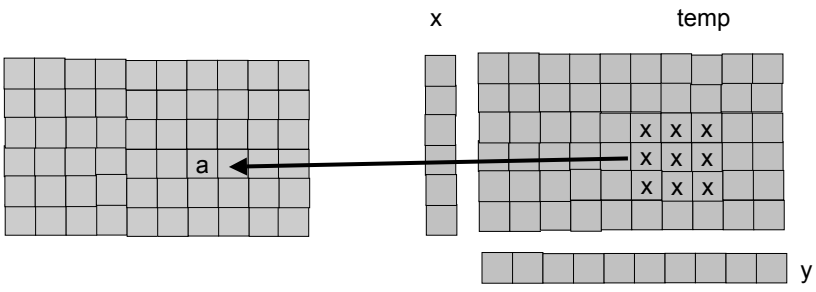
max(temp[0:2,0:3])



Neighborhoods

in “comprehension” syntax:

[avg([temp[x,y] | i-1 < x < i+1, j-1 < y < j+1]) | i < N, j < M]



Modeling Arrays

as collections…

$\{ (0, 0, \ldots, 0, x_{00\ldots0}),$
$(0, 0, \ldots, 1, x_{00\ldots1}),$
$:$
$(s_0, s_1, \ldots, s_d, x_{s0s1sd}) \}$

Beeri, Chan 96
Fegaras, Maier 95
Object algebras with order

Modeling Arrays

as functions…

shape = $(s_0, s_1, \ldots, s_d)$
$I_0 = \{0.. s_0\}$
$I_1 = \{0.. s_1\}$
:
$I_d = \{0.. s_d\}$

$f : I_0 x\ I_1 x \ldots I_d \rightarrow T$

Libkin, Machlin, Wong 96
Baumann 99
APL, functional languages

Comprehension Syntax

slice … = [A[i, j]) | 5 < i < 10, 10 < j < 15]
map f A = [f(A[i, j]) | i < N, j < M]
transpose A = [A[j, i] | | i < N, j < M]

reverse B = [B[N - i - 1] | | i < N]

Baumann 99
Libkin, Machlin, Wong 96

Optimizing Comprehensions

β: [e1 | i < e2][e3] →
if e3 < e2 *then* e1{i/e2} *else error*

η: [e[i] | i < len(e)] → e

δ: len([e1 | i < e2]) → e2 *see Limsoon Wong's dissertation*

Comprehensions are a syntax for the Nested Relational Calculus w/ Arrays
NRCA sufficient for complex objects: arrays, bags, sets, lists
Strong theoretical results mostly borrowed from functional programming
A negative result: bounds checking is undecidable

Query Languages for Arrays

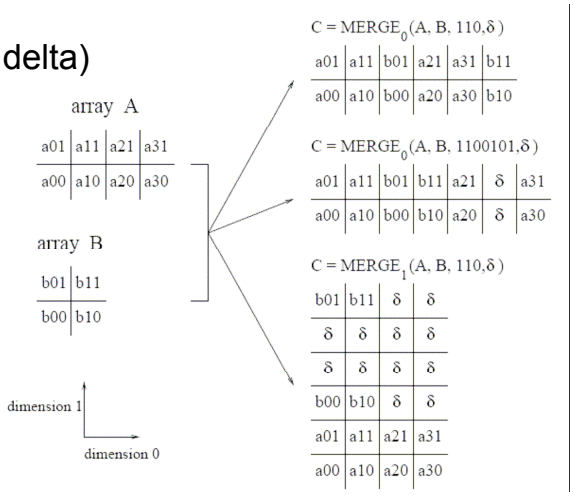
- AQL *Libkin 96*
  - comprehensions
- RasDaMan *Baumann 99*
  - comprehensions, condense, sort
- AML
  - subsample, merge, apply
- RAM *Ballegooj, Cornacchia, de Vries 2005*
  - map, transform, aggregate

AML: MERGE

$MERGE_i(A, B, P, delta)$

P is a *bit pattern* indicating columns along dimension i

delta is a filler value



Access and Storage

Paradise, Dewitt et al, VLDB 1994

Active Data Repository, Saltz 1999 – 2001

RasDaMan, Baumann 1999 – 2005

Granite DB, Rhodes, Bergeron 2002 – 2005

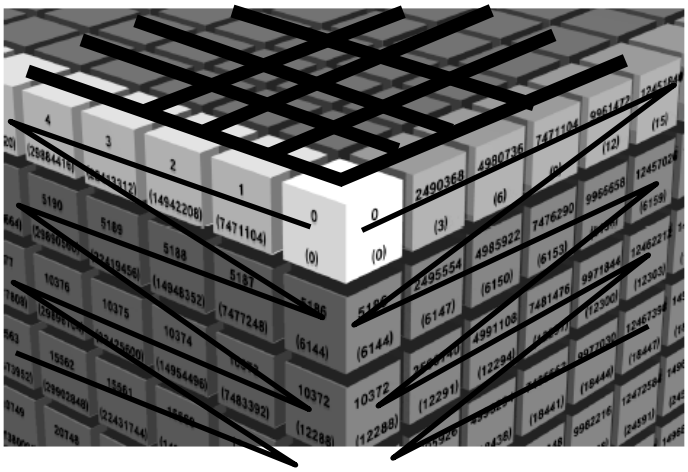
HDFFastQuery, Gosink et al, SSDBM 2005

Iteration-aware Prefetching

- Rhodes, Bergeron, SSDBM 2005

Idea: use cache block shapes and prefetching that match the access pattern of the query

Iteration Order vs. Storage Order



Iteration Order vs. Storage Order

```
for x in xs:
  for y in ys:
    for z in zs:
      read(&v[x][y][z], datum_size);
```

```
for x in xs:
  for y in ys:
    read(&v[x][y][0], |zs|*datum_size);
```

fewer read calls, every datum read once only

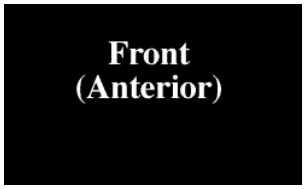
Iteration Order vs. Storage Order

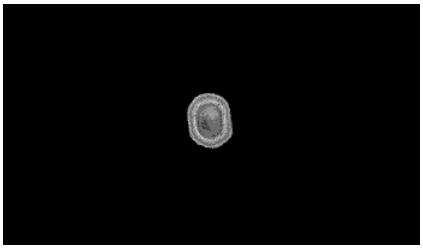
```
for x in xs:
  for y in ys:
    for z in zs:
      read(&v[x][y][z], datum_size);
```

```
for z in zs:
  for x in xs:
    for y in ys:
      read(&v[x][y][z], datum_size);
```

every datum read once only, but too many read calls

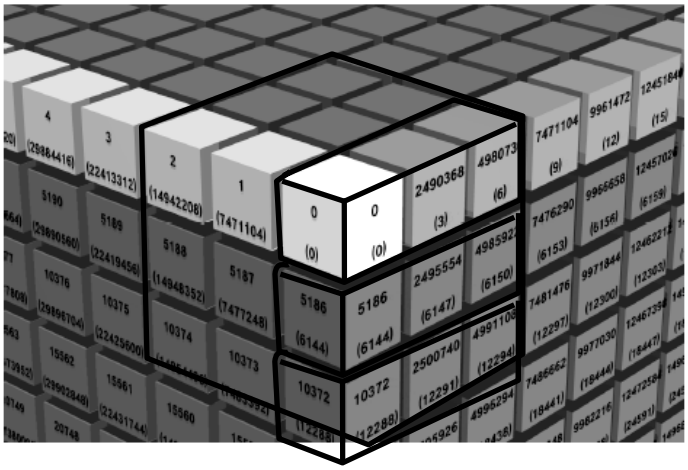
Application: Visible Human Animations





Subblock Query

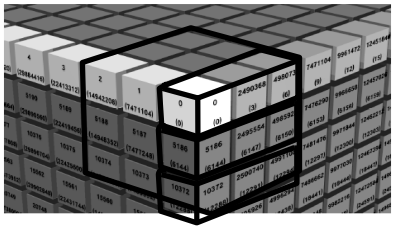
query region



rods

Subblock Query

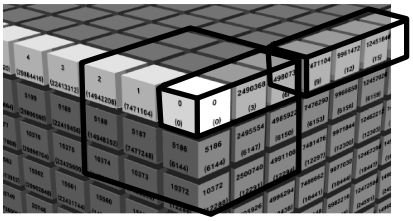
query region



rods

"Spatial Prefetching"

query region



Ordinary prefetching fetches wrong data

Results

	Datum Iteration over native file				Datum Iteration over chunked file			64^3 Block Iteration over native file			64^3 Block Iteration over chunked file		
	a	b	c	d	e	f	g	h	i	j	k	l	m
Ordering	File System Cache	128MB SP Cache	512MB SP Cache	Max Speed up	512MB LRU Cache	512MB SP Cache	Speed up	File System Cache	512MB SP Cache	Speed up	512MB LRU Cache	512MB SP Cache	Speed up
{0, 1, 2}	9963	868	961	11.5	4628	3634	1.27	705	304	2.3	3288	342	9.6
{1, 2, 0}	16786	1311	1294	13.0	9175	3880	2.4	664	279	2.4	3081	342	9.0
{2, 1, 0}	360000 (est)	11349	3719	96.8 (est)	9607	4485	2.14	7847	2777	2.8	3736	966	3.7

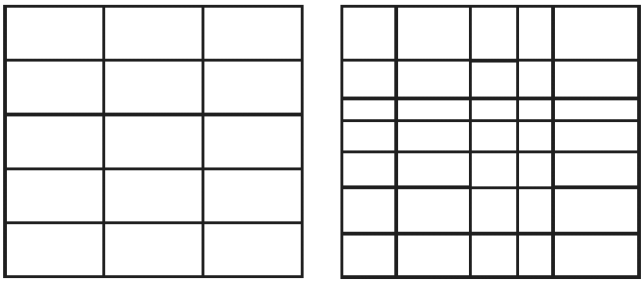
Table 1. Results for a complete traversal of an 8GB file of 1024x1024x2048 floats. Native files are in plane-row-column order, while chunked files consist of 4K chunks. All execution times are in seconds.

They don't show results for iteration order (2,0,1)…

Arbitrary Tiling

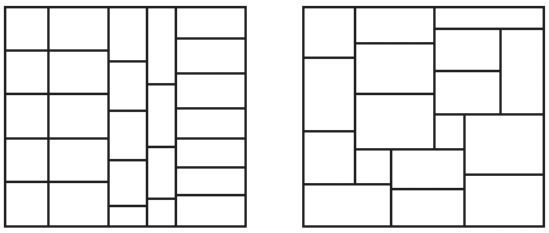
- Baumann, 1999
- used in the *RasDaMan* raster database management system

Tiling Arrays



a) Regular b) Irregular

Arbitrary Tiling



a) Partially aligned b) Totally nonaligned

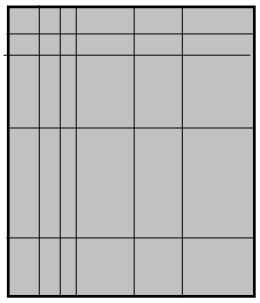
Access Patterns

- whole object
- subslab, same dimension
- subslab, lower dimension
- section, 1-dimensional

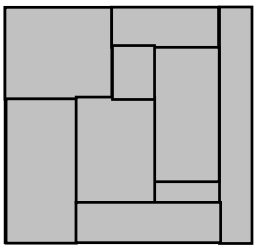
Idea: fit the tiling scheme to the query workload

Tiling Strategies

- Directional Tiling
  (Dimension Partition)



- Areas of Interest

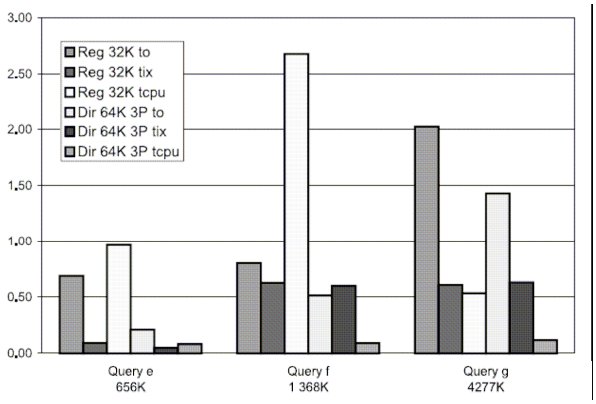


Tested Queries

	Query Region	Size (KB)	Selected (Months, Product classes, Country Districts)
a	[32:59,28:42,28:35]	13	1,1,1
b	[32:59,*:*,28:35]	52.5	1,all,1
c	[32:59,28:42,*:*]	164	1,1,all
d	[*:*,28:42,28:35]	342	all,1,1
e	[32:59,*:*,*:*]	656	1,all,all
f	[*:*,*:*,28:35]	1400	all,all,1
g	[*:*,28:42,*:*]	4300	all,1,all
h	[182:365,*:*,*:*]	4300	6,all,all
i	[32:396,*:*,*:*]	8500	12,all,all
j	[28:34,*:*,*:*]	164	1 week,all,all

Table 3. Queries for the directional tiling test.

Results: Directional Tiling



t0 = time to retrieve tiles

tix = time to access index

tcpu = time to compose tiles and form the result

A Different Problem

- When are two arrays similar?

Q =

DB =

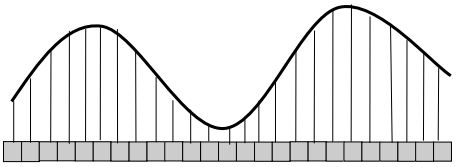
Applications

- Image
- Timeseries
- Sound
  - music (Query-by-humming)
  - sonar signatures
- High-dimensional data
- Financial Analysis
- Feature Tracking…

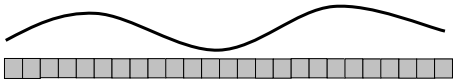
Raster → Features

Timeseries Indexing

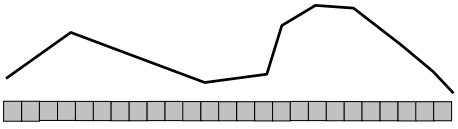
Q =



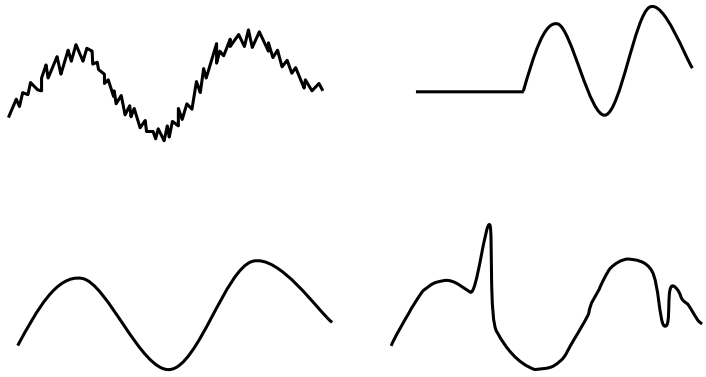
A =



B =



Timeseries Indexing



Timeseries Indexing

- Euclidean distance
- Dynamic Time Warping
  - Jagadish, Faloutsos 1998, Keogh 2002
- Wavelets
  - Miller 2003
- LCSS
  - Vlachos, Kollios, Gunopolos 2002
- EDR
  - Chen, Ozsu, Oria 2005