Lysator Upplysning

High-Performance Database System for Weather and Water data

Dr Esa Falkenroth, SMHI
Datalager och -åtkomst
Esa.Falkenroth@smhi.se
Phone: +46 (0)702-104028

SMHI

Synopsis

- What is weather data
- Extreme performance (Unofficial Record?)
- Cross-enterprise retrieval interface
- Experience of building a large-scale high-performance weather database system

Who I Am

- Dr Esa Falkenroth, Database architect
- SMHI MHO Datalager och åtkomst
  - 7 person database unit
  - Responsible for the central weather databases

Who are you?

- **SURVEY: Please raise your hands…**
  - **Who used a database system ?**
  - **Who has written any computer program ?**
  - **Who has written an SQL-query ?**
  - **Who knows what a B-tree is ?**
  - **Who has written stored procedures ?**
  - **Who has written spatial indexing methods ?**

Swedish Meteorological and Hydrological Institute (SMHI)

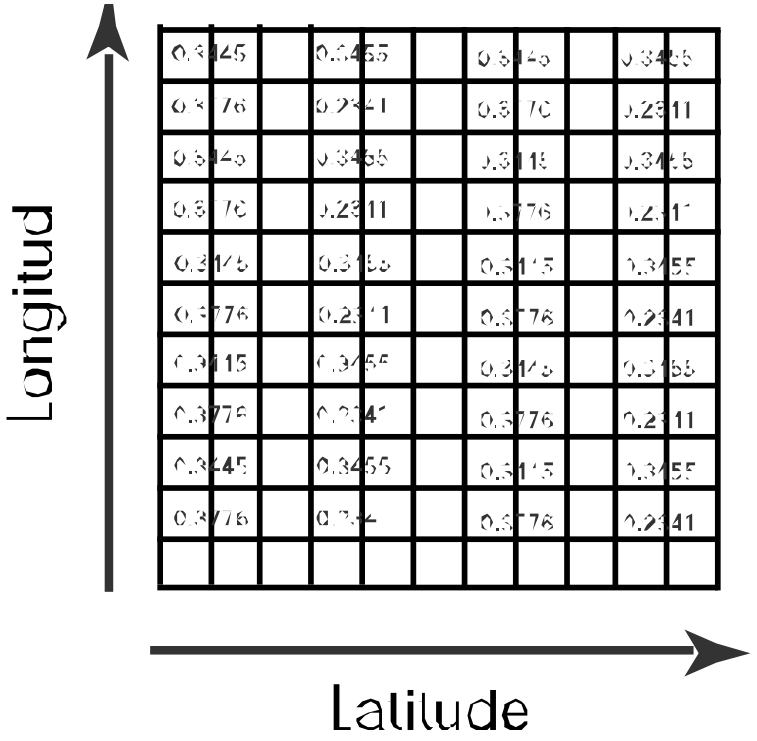
-- An IT-company started in 1873

- SMHI provides planning and decision support for businesses and activities that dependent on weather or water.
- Competence in meteorology,hydrology, and oceanography.
- Customers are swedish and international businesses in transport, environment, energy, as well as commerce and governments.

SMHI Customers

WHAT IS WEATHER DATA ?

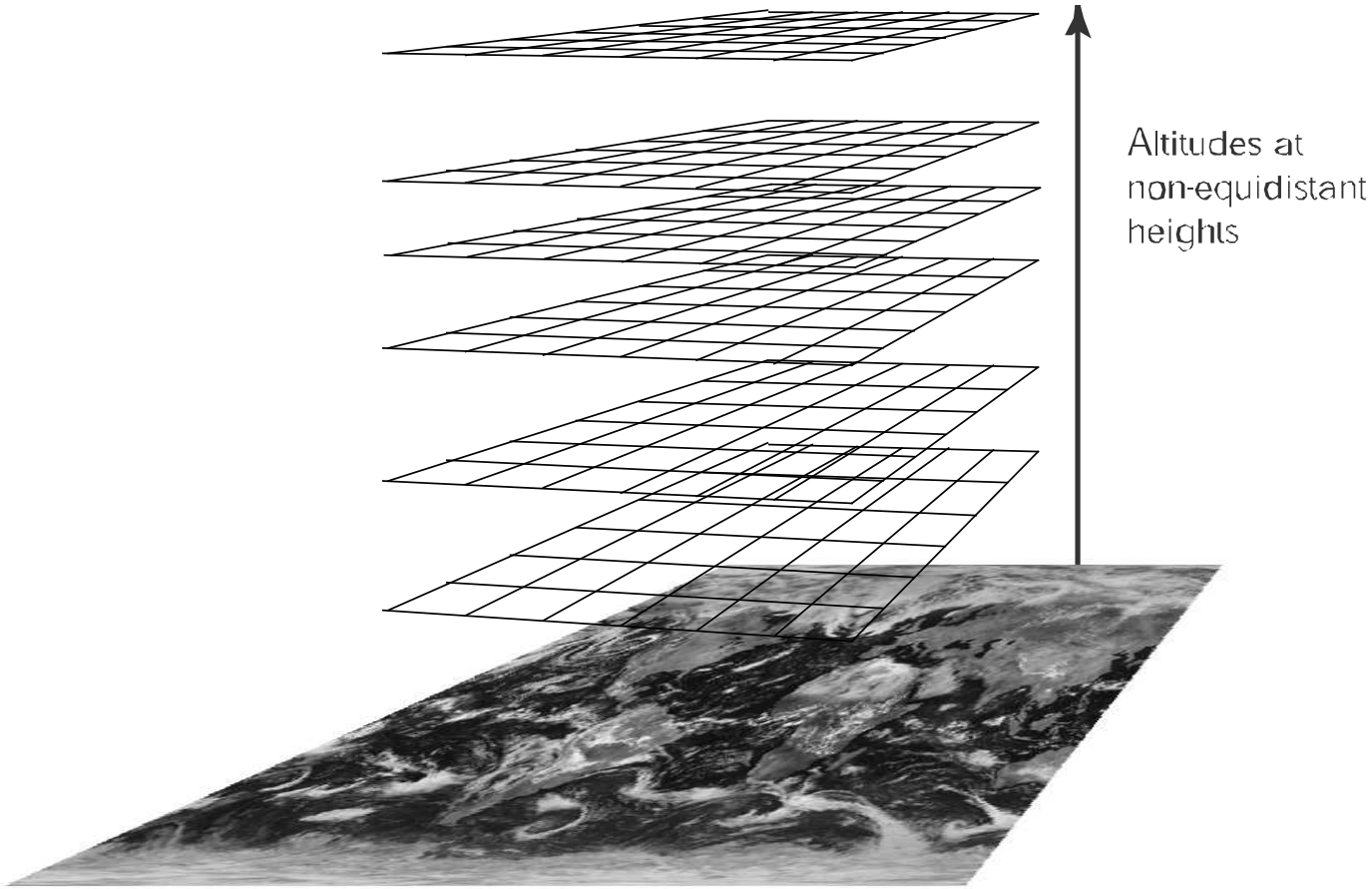
What is weather data ?



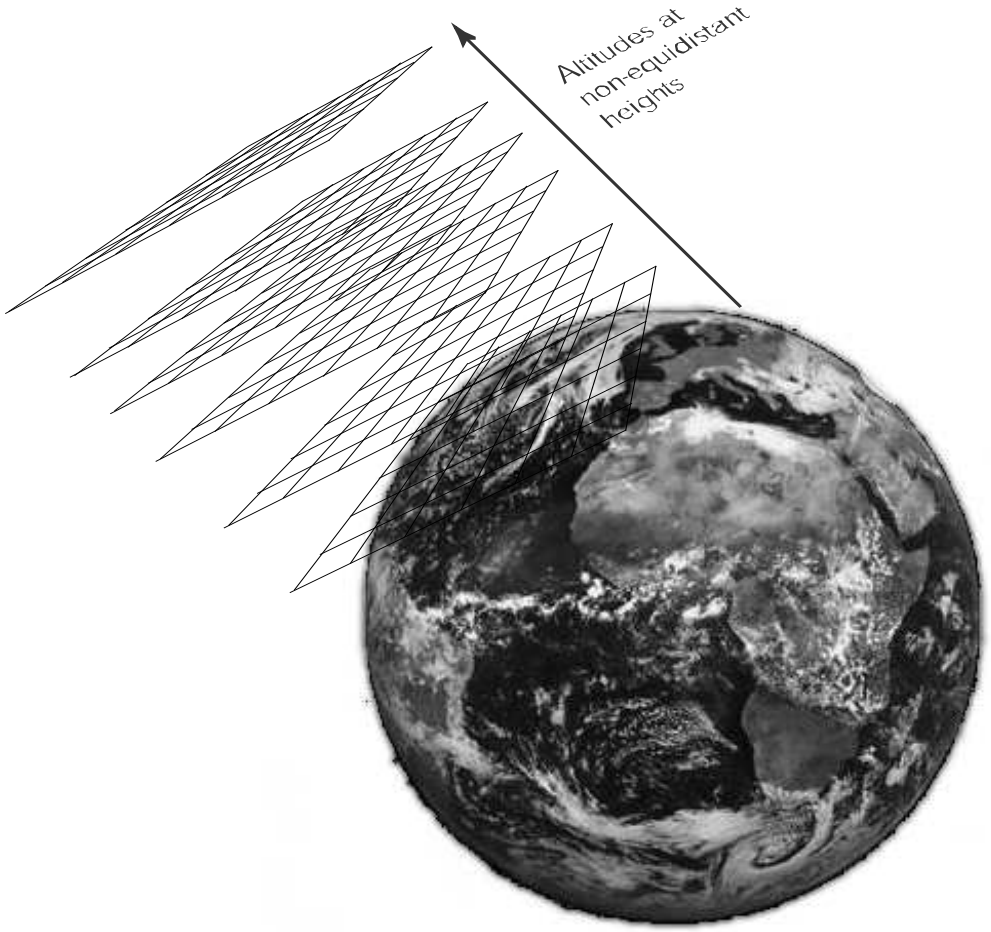
FORECASTS
=GRID DATA

Spacefilling scanmode

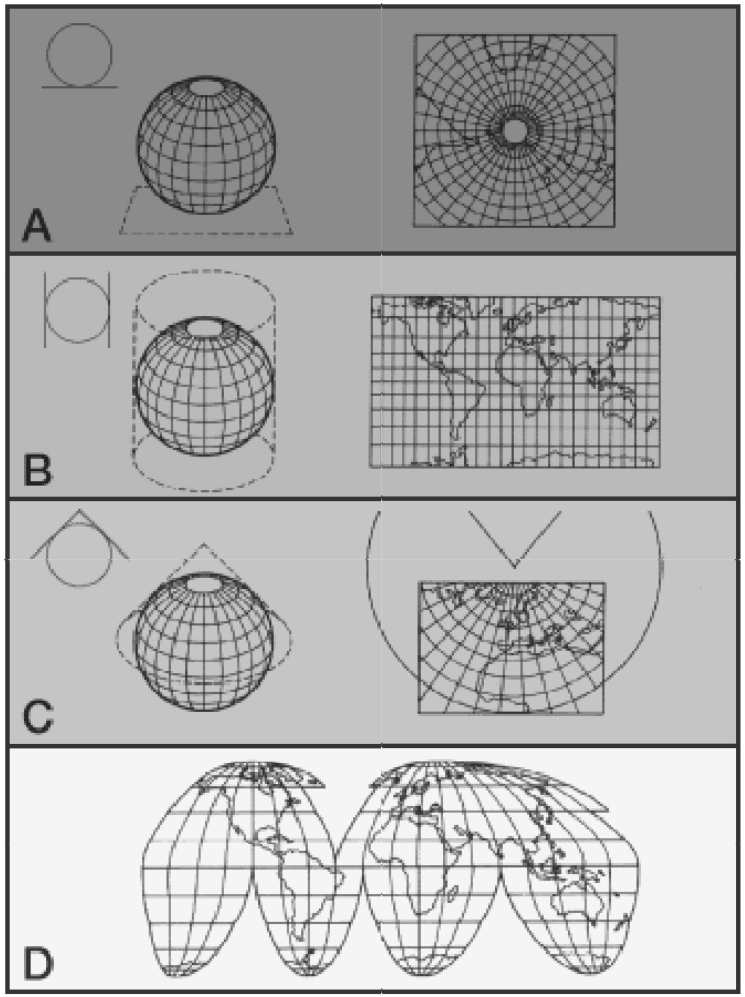
What is weather data ?



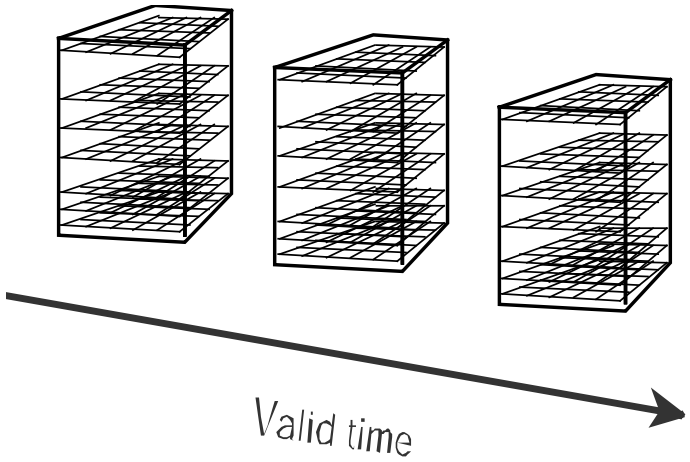
Geodetic Columbus model



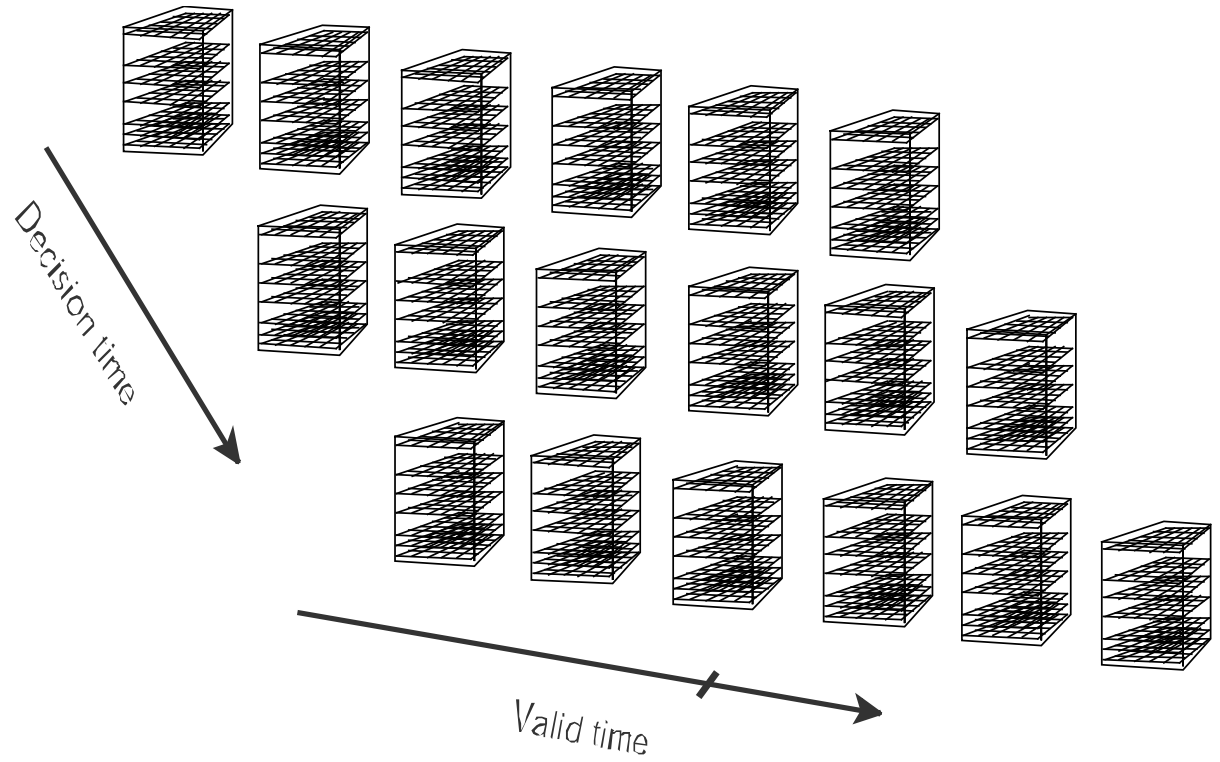
Earth can be flattened in many ways



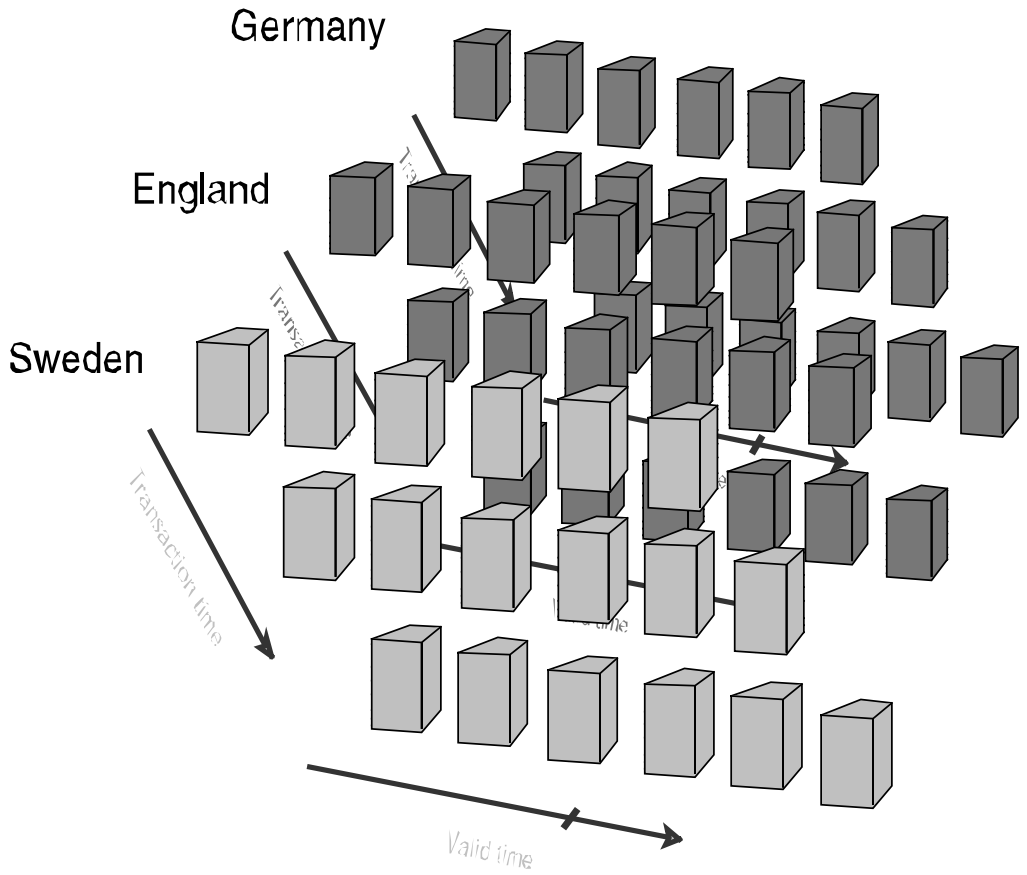
Temporal dimension



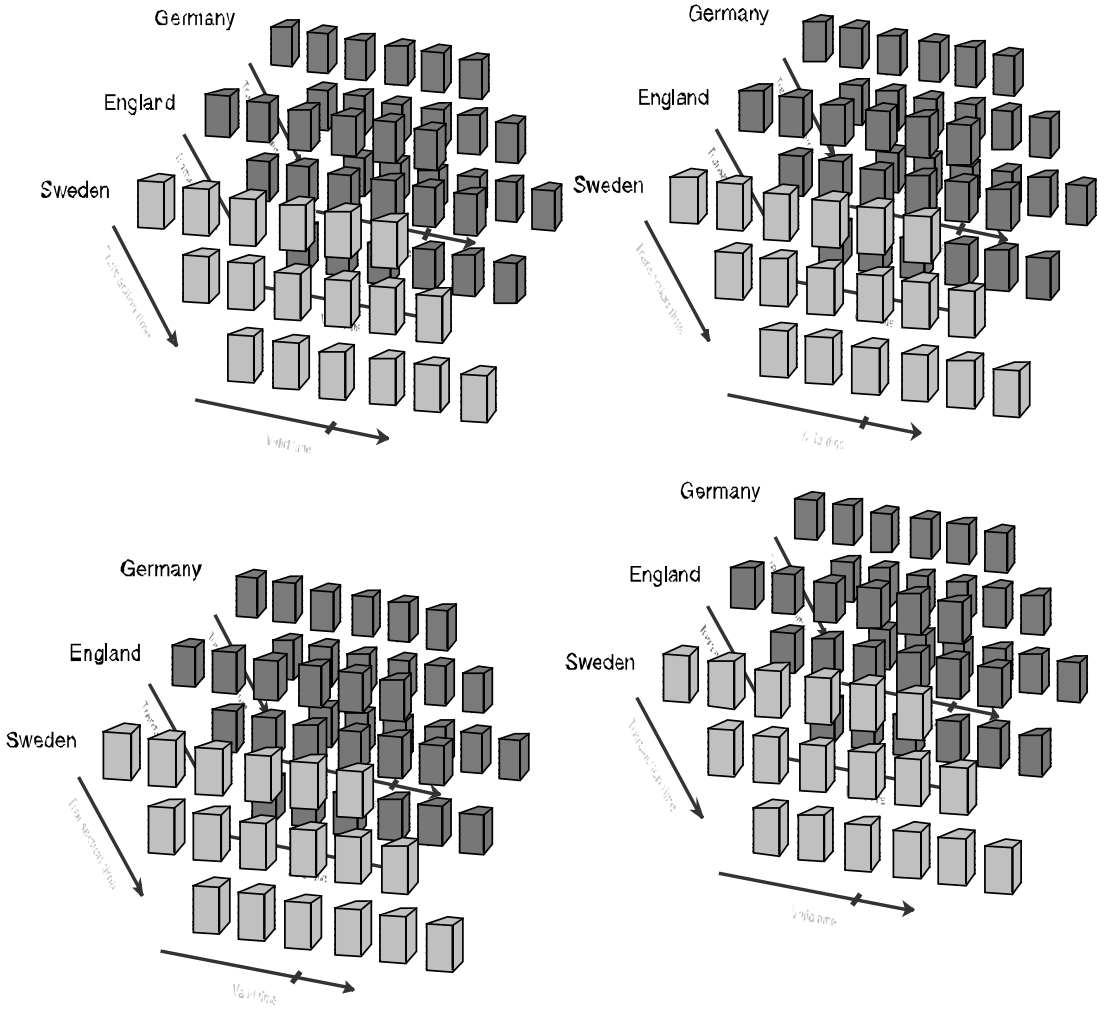
Bitemporal database



Multiple sources



Multiple parameters



PROBLEM STATEMENT

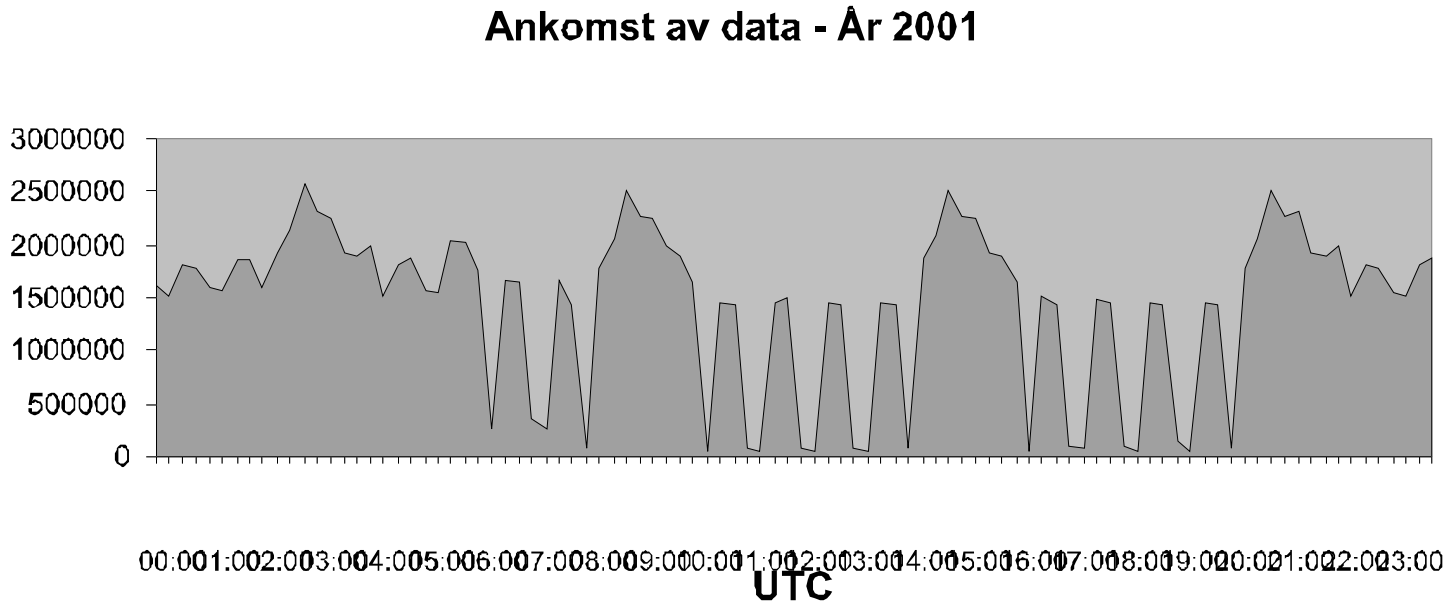
Information overload problem

- Too much information...
- Customers and meteorologists have problems interpreting 13-dimensional data
- Earlier data was stored in a separate file servers :-(~~
- Different data formats, different units, different meta data, different everything
- Inconsistencies in data

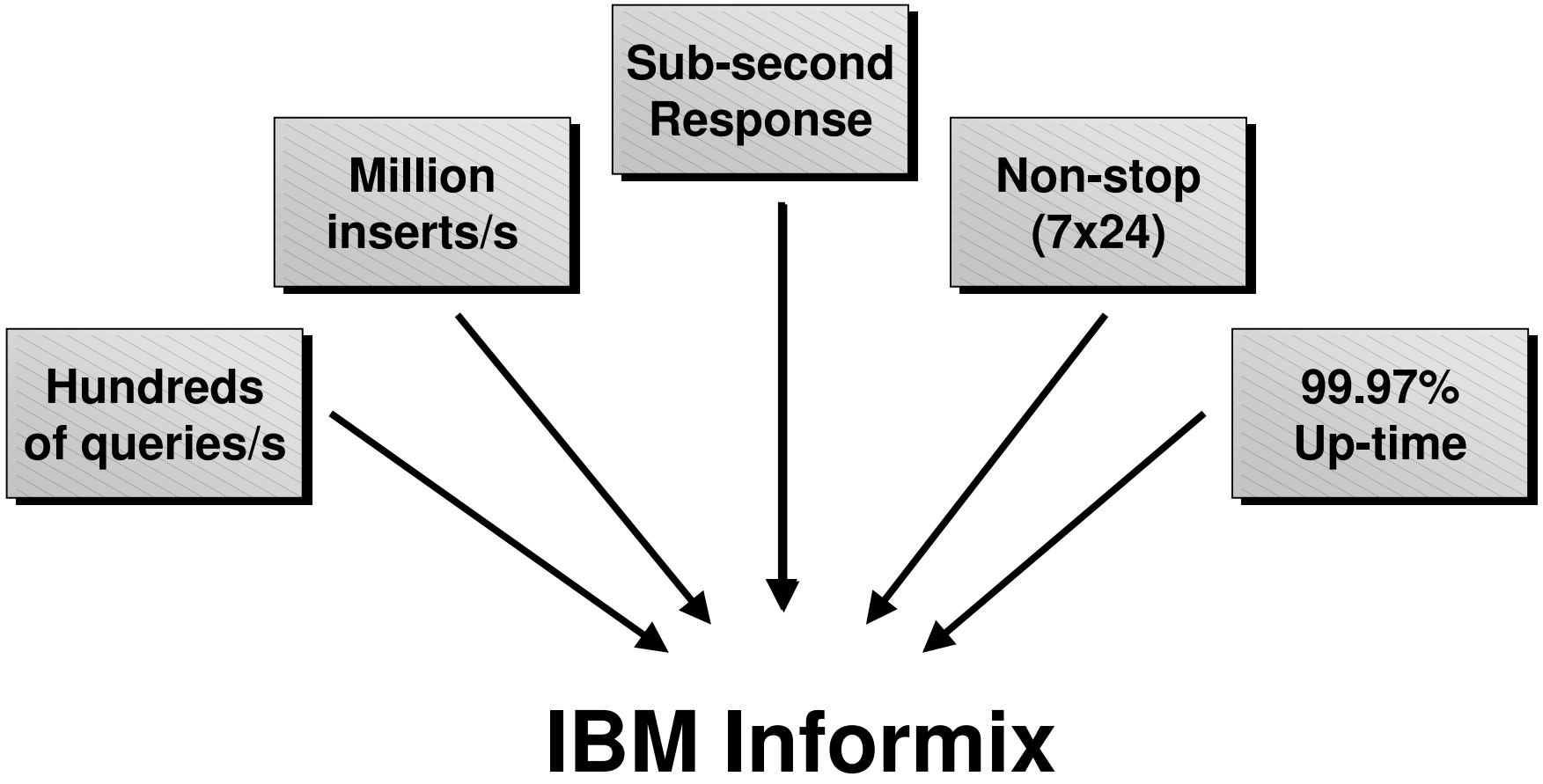
Large volumes of data

- Each day, SMHI receive in excess of 50 GB of structured data from various sources
- Corresponds to a 1km stack of printed paper

Peak-hour problem

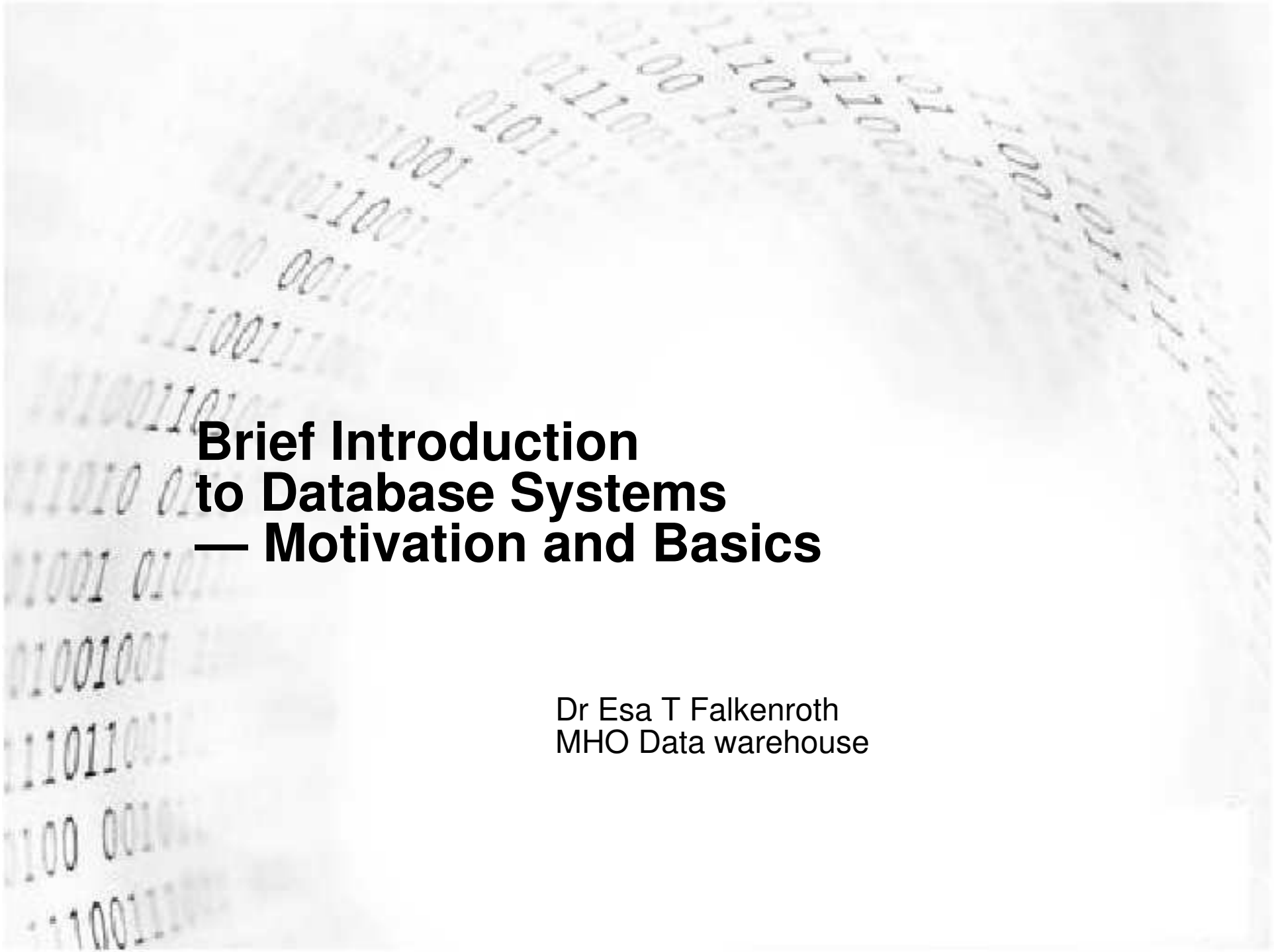


Requirements on IBM IDS



Mission Impossible

- Given a midrange Sun server (E450R)...
- How to insert 1000000 geographically referenced floats/s ?
- How to retrieve 1000 rows per second ?
- How to build cross-platform APIs that support access from all platforms and programming languages ?
- How to make this work almost always ?



Brief Introduction to Database Systems — Motivation and Basics

Dr Esa T Falkenroth
MHO Data warehouse

Early data management

- Access time increase as data volume grows
  efficient access to large data sets was cumbersome
- Recovering data after system crashes was difficult
- Handling concurrent users/applications was difficult
- Changes of file format was extremely difficult
  Assumptions on structure of data are spread in many different applications
- >50% of programming effort was spent on data management: creating, manipulating, searching data
- Basically, each program reinvented the ”wheel”

BIRTH of DBMS

- **Solution was**

  (1) Extract the data (and the handling of data) from programs and move them to a separate database

  (2) Create a schema that defines structure of database

  (3) Create a general-purpose program that allows users and applications to store, organise, manipulate, and retrieve data the database:

  DATABASE MANAGEMENT SYSTEM (DBMS)

PROPERTIES OF DBMS

- Near-constant performance independent of data size
- Automated recovery and repair after crashes
- Concurrent users (efficient & correct interleaving)
- Structure for data
- Access independent of file formats and physical layout of data on disks
- ….and flexibility in search

TERMINOLOGY

- Data are known facts that can be recorded and have an implicit meaning
- Database is an interrelated collection of data that represent a specific aspect of the real world. Databases must have a regular recurring structure to facilitate retrieval and manipulation.
- Database management system (DBMS) is a set of programs that allows users and applications to create, manipulate, search, and maintain databases.

TERMINOLOGY

- Database system includes a database and a database management system
- A schema defines the structures of data (a set of tables with several columns)

Money transfer example

- Consider repeated transfers of X$ between two bank accounts: A and B (no database involved)

- Algorithm:
  Read balance for acct A
  Subtract X$
  Write back balance for A
  Read balance for acct B
  Add X$
  Write back balance for B

Case: Disappearing money

- Customer B is upset and calls his bank
- He received10$ too much
- What happened ?

Case of the extra money

- Customer B is upset and calls his bank
- He received10$ too much
- What happened ?
  - Concurrent interleaved manipulations
  - Communication failure during update
  - Media failure after update

Solution is ACID transactions

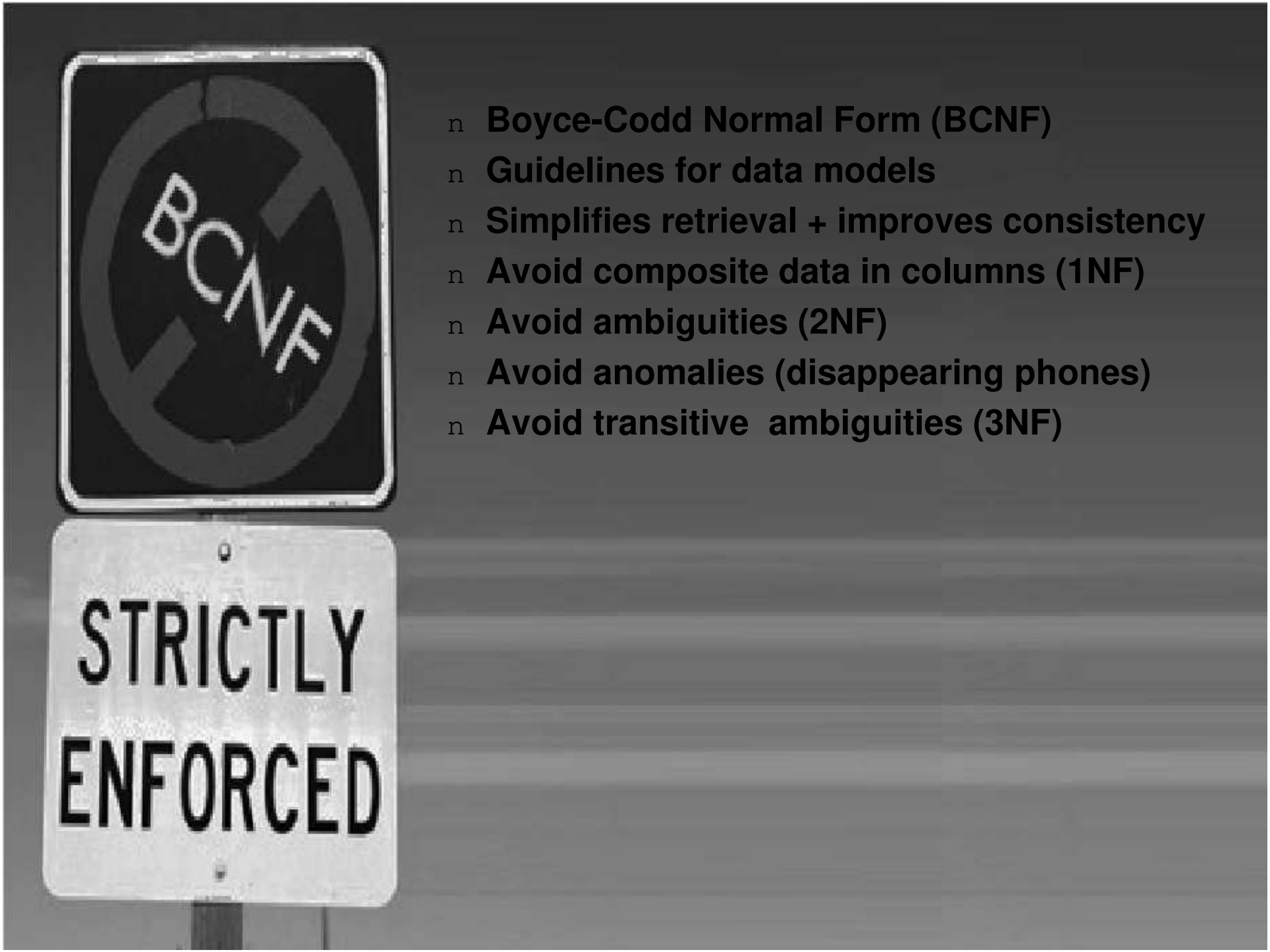
- Atomicity (All or nothing property)
- Consistency (Leave the database in a consistent state)
- Isolation (Ongoing change is hidden from other users
- Durability (changes written to both disk and logfile)

Relational Data Model

- Database is a collection of ”tables” [relation]
- Each table contains a set of rows [tuples]
- Each row contains an ordered set of columns [attrib.]
- Columns contain atoms (indivisible facts)

- PERSON_TABLE

Name	*Phone_column*	*Room*	*Building*
Esa	4958	486 + 155 23	
Jim	2342	512	11
Airi	6661	122	21
Anna	6461	123	21
Ivan	7657	122	22
Miguel	2342	445	11



- **Boyce-Codd Normal Form (BCNF)**
- **Guidelines for data models**
- **Simplifies retrieval + improves consistency**
- **Avoid composite data in columns (1NF)**
- **Avoid ambiguities (2NF)**
- **Avoid anomalies (disappearing phones)**
- **Avoid transitive ambiguities (3NF)**

NORMALISATION

- **PERSON_TABLE**

Name	*Phone_column*	*Room*	*Building*
Esa	4958	486 + 155 23	
Airi	6661	122	21
Ivan	7657	122	22

- **PERSON_TABLE** **ROOM_TABLE**

Name	*Room*
Esa	486
Esa	155
Airi	122
Ivan	122

Room	*Phone*	*Building*
486	4958	23
155	9821	22
122	7777	22
999	9998	11

Data retrieval

- Easy retrieval
- Specify what not how
- No programming

SQL

- **ANSI-standard query language for interacting with a database**
- **Creating structures (relational tables)**
- **Storing data into tables**
- **Powerful retrieval from tables**
- **Improving performance through indices**

CREATE TABLE

- **Create table person_table**
  (name varchar(80),
  room varchar(80));

- **Create table room_table**
  (room varchar(80) primary key,
  phone varchar(80) default ‘009’
  building integer not null);

INSERT

- **Insert into person_table values ('Esa Falkenroth', '348');**

SELECT

- **Who works in office ‘348’?**

  Select name, phone from person_table where room=‘348’;

SELECT

- **Does anybody share her/his room?**

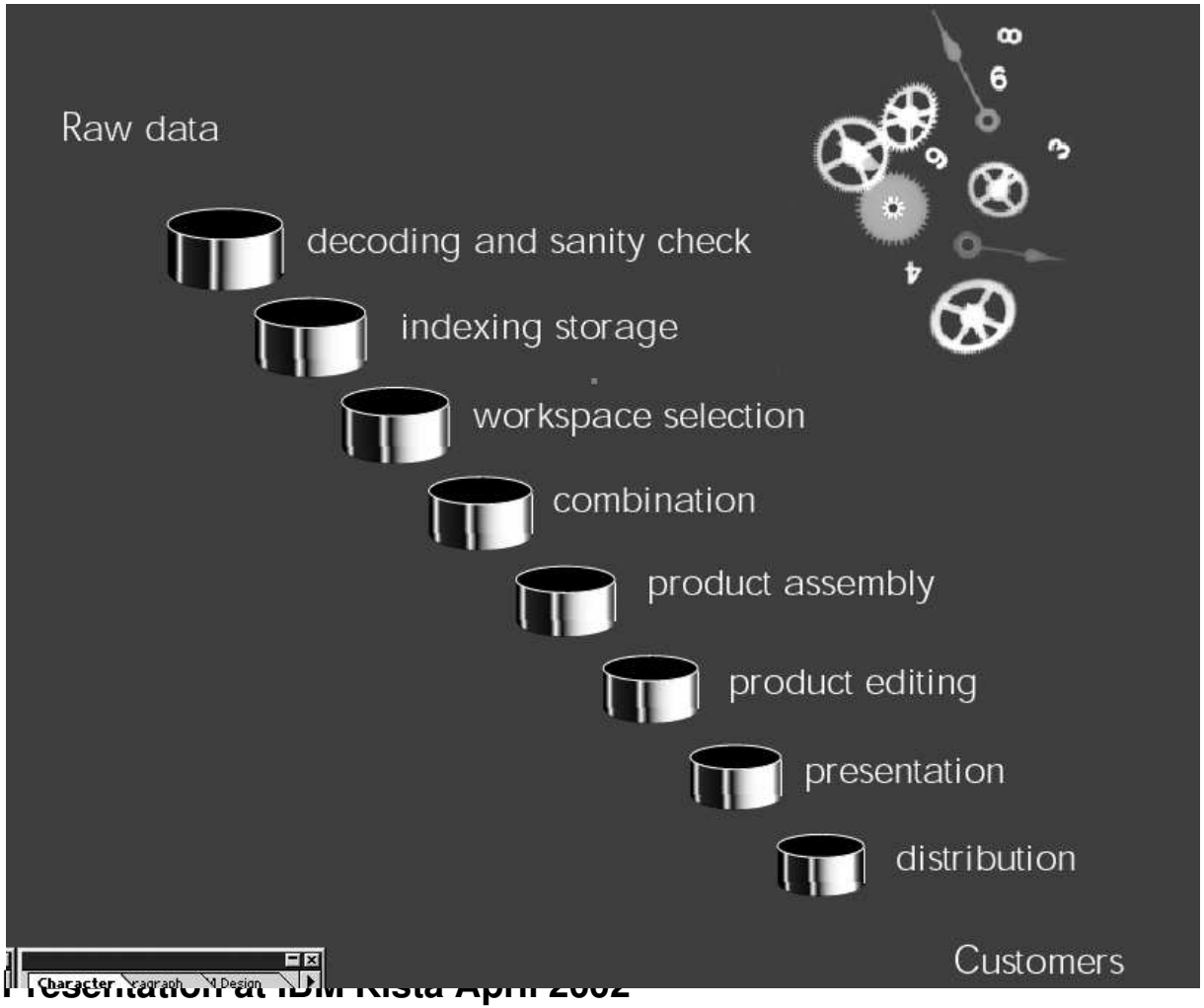
```
Select distinct p1.name
from person_table p1, person_table p2,
     where p1.room=p2.room
     and not p1.name=p2.name;
```

ARCHITECTURE WALKTHROUGH

Refining raw data to products

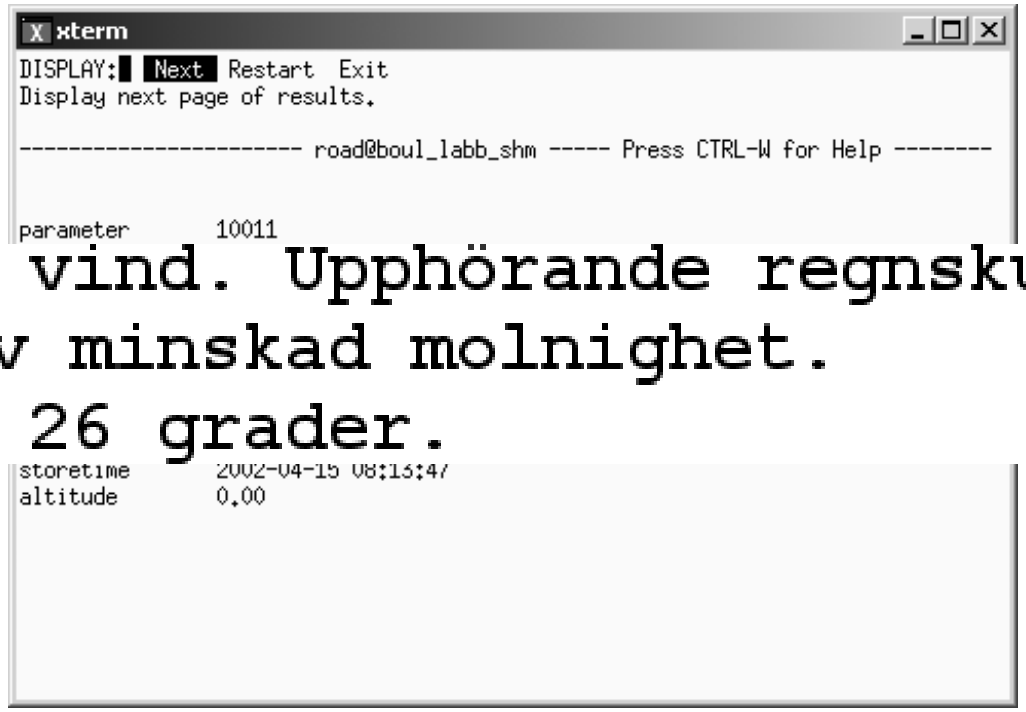
- Manage volume and complexity of data
- Turning raw data to customer products
- Need to analyse and process the data and build products

SMHI Information factory



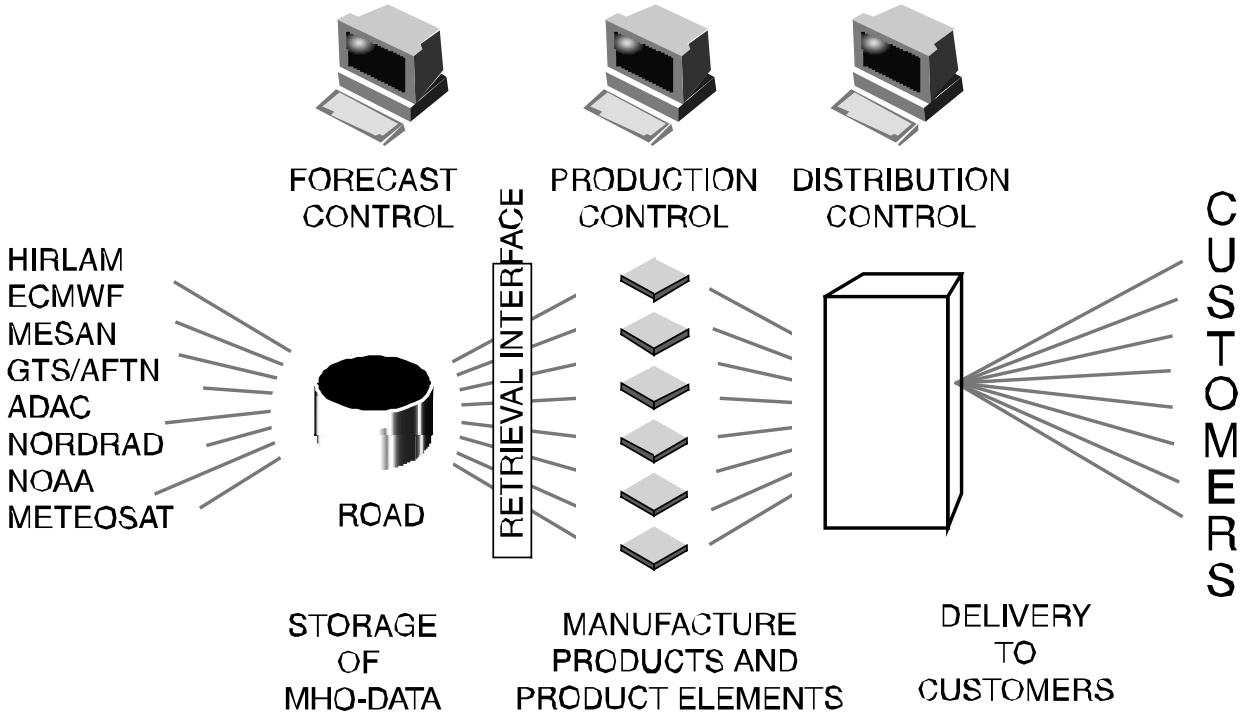
Raw data to products

```
LQTZ 150555Z VRB01KT 0200 R09/RVRNO FG SCT000 SCT100 BKN170 08/
08 A29 80 RMK PK WND 29003/08 WND DATA ESTMD FG SCT000 SLPNO 70
006 8/070 9/02 0 53010 90501 WR// RED=
```



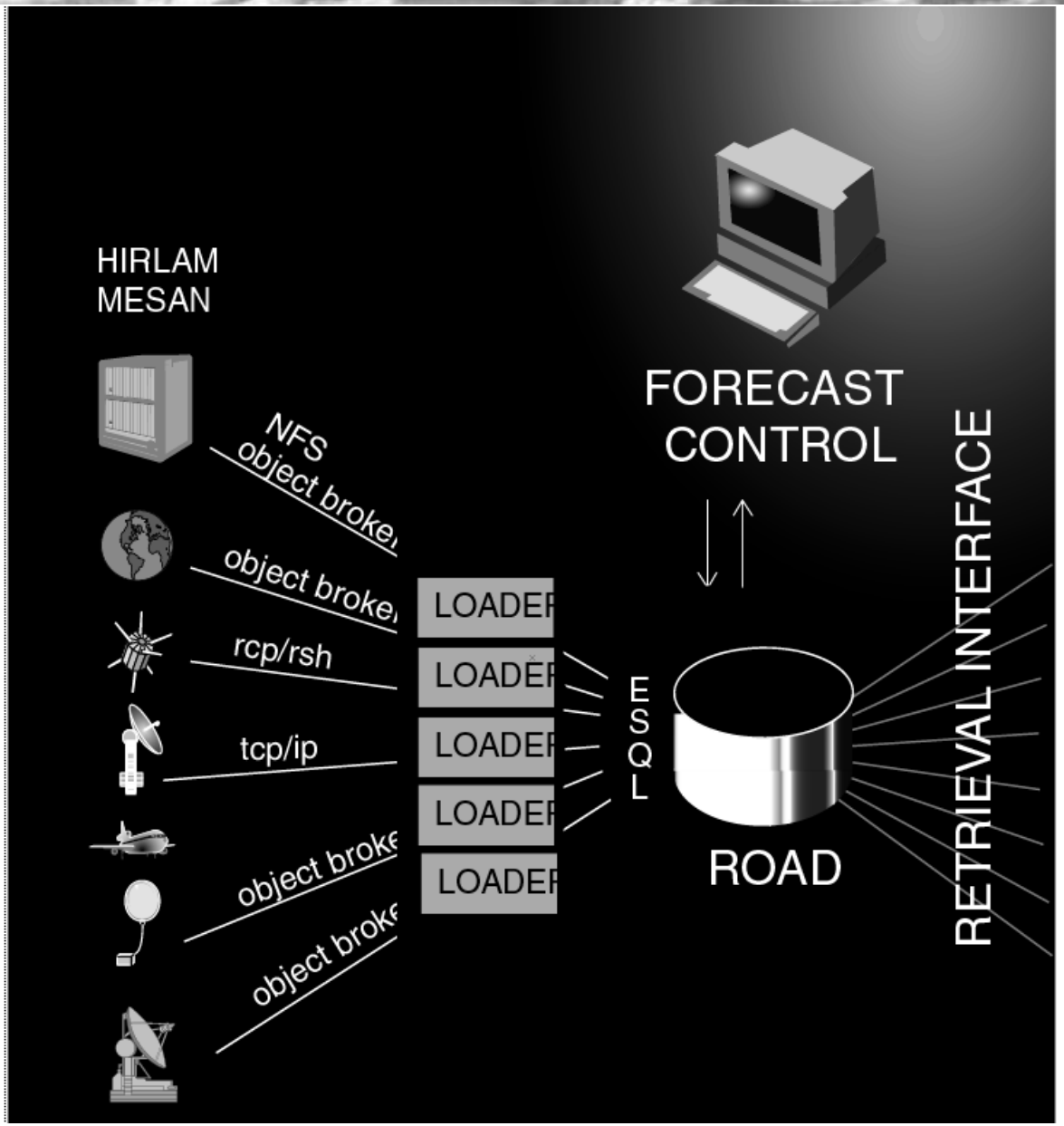
Västlig vind. Upphörande regnskurar följt av minskad molnighet.
22 till 26 grader.

System architecture

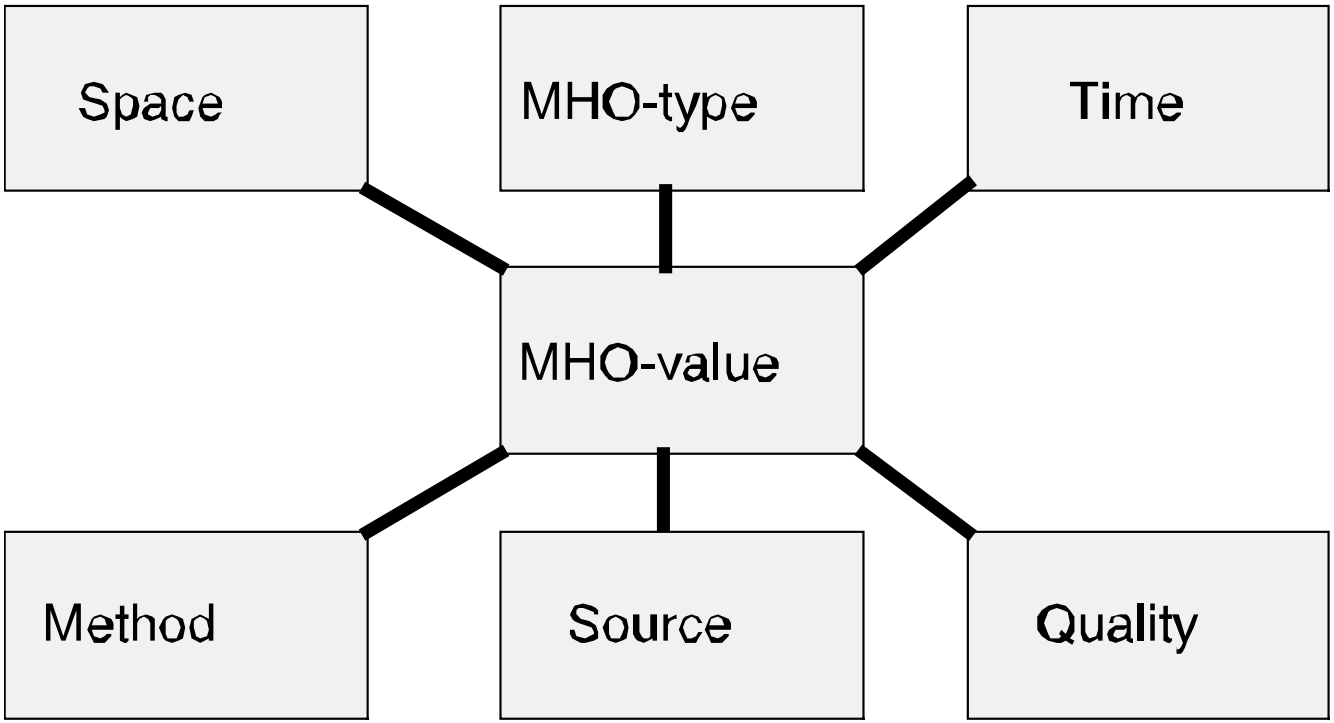


System architecture ZOOM

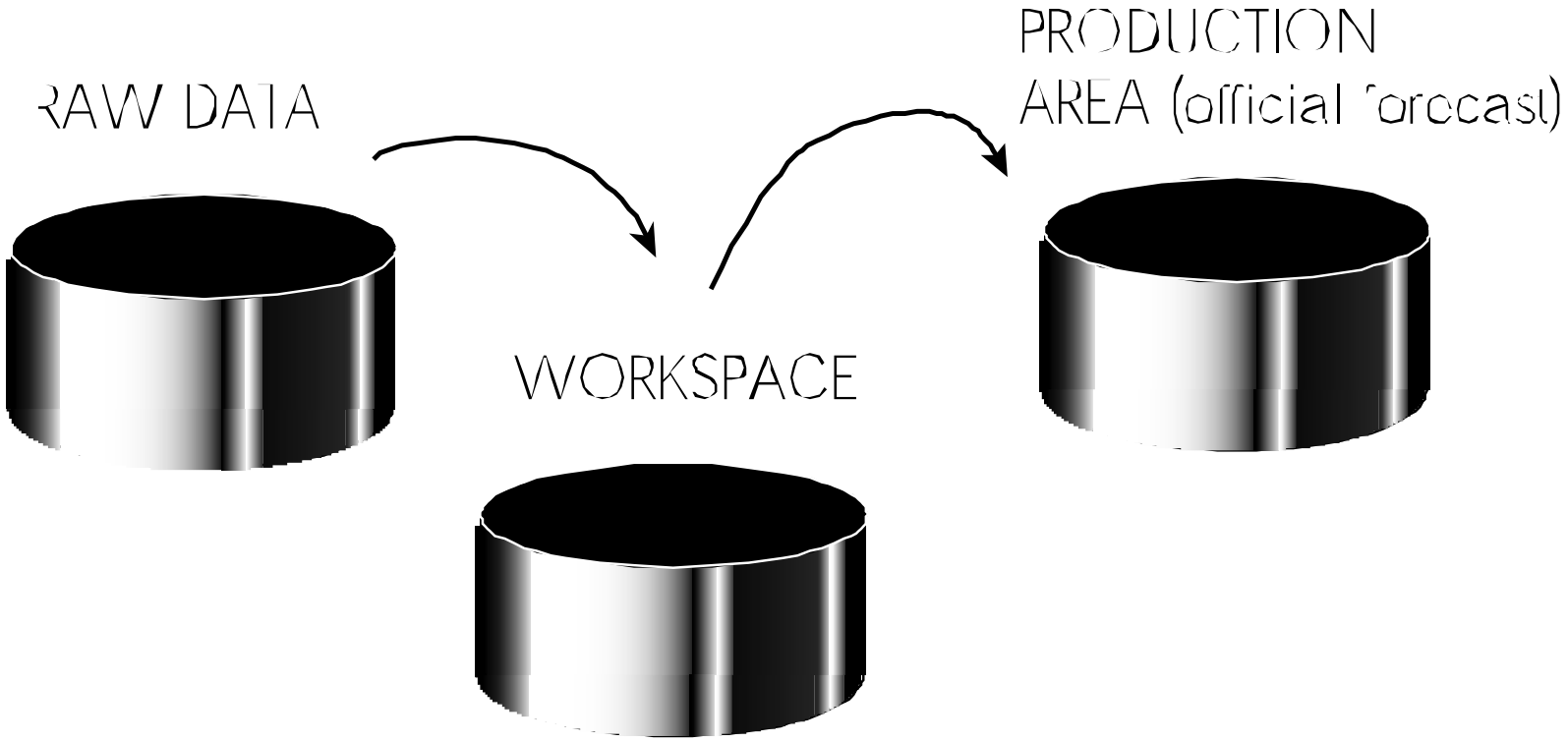
Similar to realtime loader of timeseries

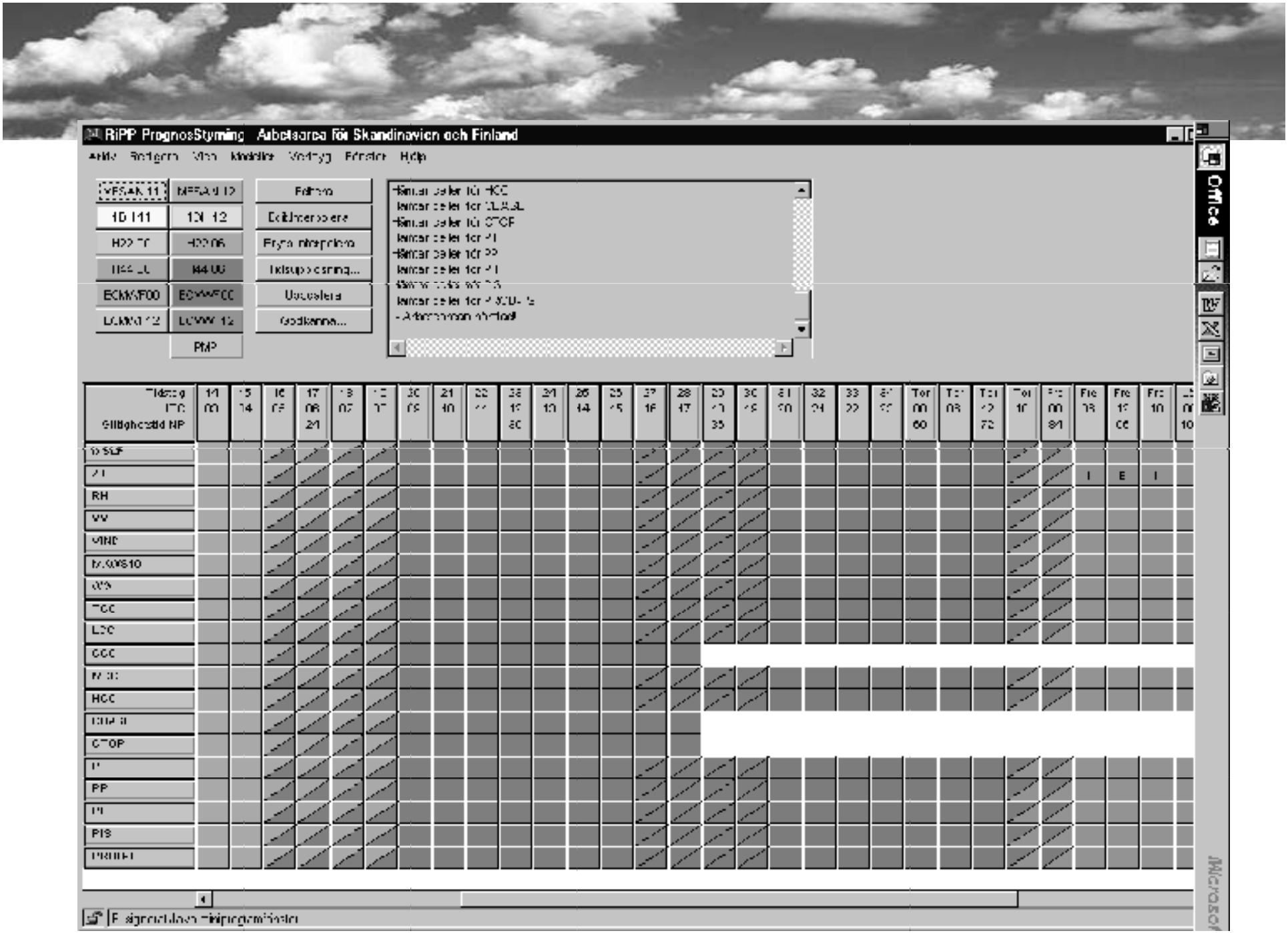


Data model

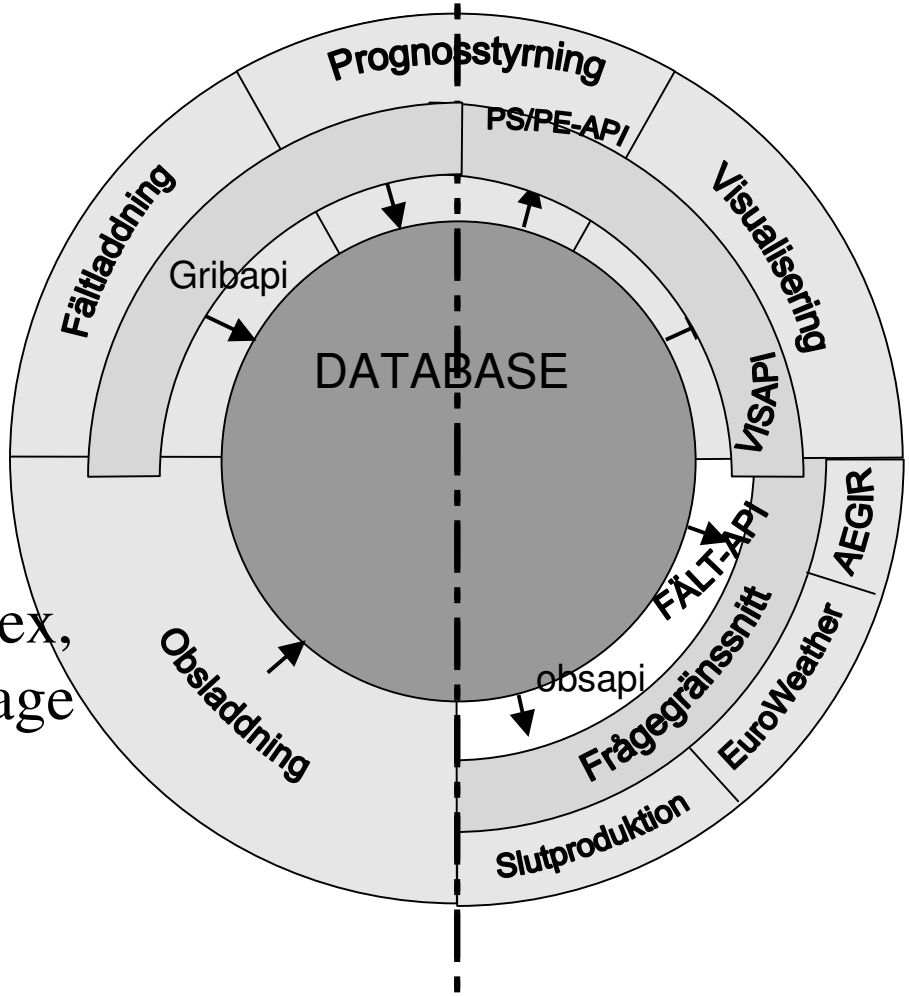


Official ackredited forecast




RiPP PrognosStyming Arbetsarea för Skandinavien och Finland
Office
RH
VV
VIND
HCC
CTOP
PP
PIS

System architecture (retrieval)



IN

OUT

Clumsy, complex, platform/language specific APIs

Retrieval volumes/intensity

- SMHI volumes
  - 2000 deliveries each day
  - >5 products per delivery
  - 10-100 elements per product
  - 10-100 symbols per element
  - 1-15 queries per symbol
  - 1-100 rows per query
- Peak intensity
  - 70-150 queries per second
  - delivers ~1000 rows per second (4 CPU)
- Diskvolume (72 GB -> 400 GB)

SUMMARY OF ARCHITECTURE

Recipe for real-time database

- Collect all MHO data in a single database system
- Standardised cross-enterprise interfaces to MHO data
- One parameter system for MHO data
- One official accredited forecast
- Platform-independent access

Enabling technologies
IBM IDS 9.21

Mission Impossible API

- Solution to Mission Impossible
- …is extending database functionality
- PostgreSQL provides C-routines in engine
- IBM/IDS provides milib in engine
- Oracle provides stored functions (outside engine)
- Sybase provides Snap-ins

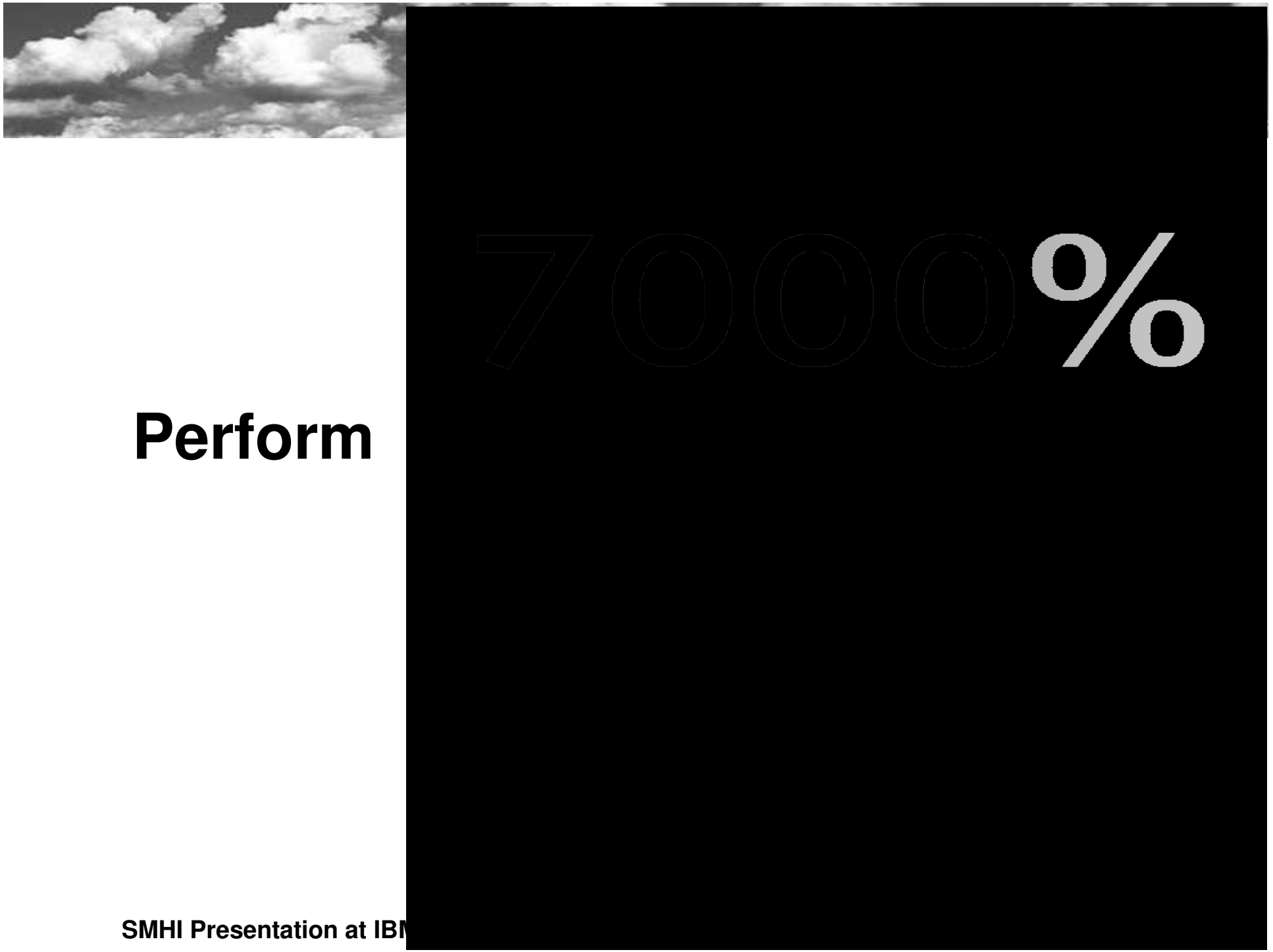
TAKE IT EASY.
IT'S JUST ZEROS
AND ONES.

Initial performance

- ~ 1 hour to load forecast data
- barely capacity to manage incoming weather observations

IBM IDS Extensibility

- What do we me an by “extensible”?
  - **Data Types (Distinct, Row, Opaque)**
  - **Built-in Routines (UDRs)**
  - **Access Methods (Applicationsspecific indices)**

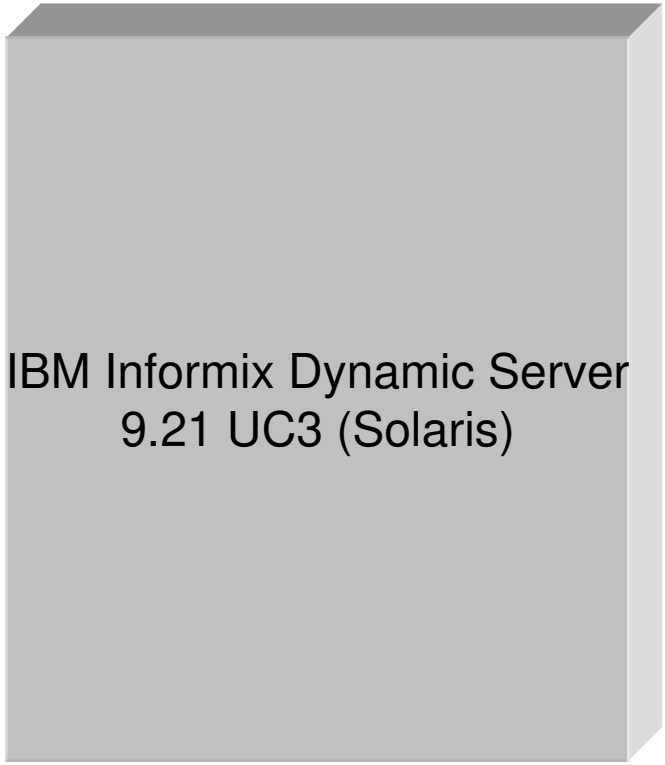


Perform

Based on commercial DBMS IBM/IDS9.21 (aka Informix)

- IBM IDS 9.21 UC3
- ESQL/C, JDBC, ODBC, OLE-DB, milib
- SMHI Datablades: functional indices, geographic indices, retrieval, meta data
- Smart BLOB for radar, satellite, forecasts
- Shared memory communication
- Binary client communication
- Extensible types (distinct, row, opaque types)
- Geodetic 3.0X1, Rtree (3?)
- Statement cache, fuzzy checkpoint

How SMHI uses IBM IDS



- **DataBlade Developer Kit**
- **User Defined Routines**
- **User Defined Datatypes**
- **User Defined Indexing**
- **R-Tree Indexing**
- **Extended B-Tree Support**
- **Row Types**
- **Collections (sets, multiset, lists)**
- **Inheritance**
- **Polymorphism**

- **We use it all...**

Extreme performance oversimplified

- **Basic tuning** **100 %**
- **High performance architecture** **1000%**
- **Extensions to DBMS** **10000%**

Way to high performance

- CPU-bound, Disk-bound, IOPS-bound
- Do as much parallell as possible
  - Large continuous parallel I/O (100 kIO minimum)
  - Parallel sources
  - Parallel loader processes
  - Parallel CPU (SMP)
- Gigantic buffers 99,97% cached reads (85%writes)
- Pipeline production process
- Use datablade technology
  - Ship computations to data rather than data to computations
  - Faster communication inside DBMS

7000% better performance

- **>100x Exploit computational indices instead of B-trees/R-trees**
- **7x Shm-communication (unless you have linked with Fortran subroutines containing COMMON…)**
- **5x Always reduce number of database calls (Essential)**
- **5x Using binary transport-format for complex objects (geodetic)**
- **5x Normalise all tables with object-columns (geodetic, LOs etc.)**
- **5x Ship operations to data instead of data to operations**
- **5x Replace r-trees with functional indices on accessor-UDRs for geo-objects (Geox is great!)**
- **5x Run ISPY on thy SQL-clients. They tend to do unexpected things**
- **4x Write your UDRs in C instead of SPL**
- **4x Continuous I/O by writing data to a single very large smart-BLOB**
- **3x Reduced frequency of meta-data updates (bundle)**
- **2x Avoid ifx_lo_write (Filetolo from /tmp is a slow starter but uses 100kIO instead of 2kIO. Faster for BLOB >5kB**
- **2x Prepared statements everywhere**
- **2x Main-memory buffer for RAID-system (Sun T3-array has 512 MB)**
- **2x Removing printf, debugging, unnecessary logging in production code**
- **2x Combining several queries into one to eliminate database calls**
- **2x Remove triggers on heavy traffic tables (infrequently accessed tables are ok)**
- **2x Nonatomic data (generally a bad thing but it improves performance)**
- **1.5x for non-ordered access use checksum-indices instead of LVARCHAR**
- **1.5x Eliminate indices (use composite indices)**
- **1.5x Concatenate transactions (tricker recovery)**
- **1.5x Let applications cache BLOB-handles to reduce selects of blob-columns (140 bytes identifier)**
- **1.5x Remove unnecessary columns**
- **1.5x Replace LVARCHAR-indices with functional index on hash(LVARCHAR) (not for range queries)**
- **1.3 Geodetic 3.0 speedup (good work)**
- **1.2x LRU-cleaner setting using fuzzy ckpt**
- **1.2x Host-files for clients**
- **1.2x Connection pooling (prepare, set isolation, lock modes etc. **once**)**
- **1.2x SDK2.60 upgrade (from SDK2.10)**
- **1.2x Remove inheritance hierarchy**
- **1.2x Look actively for sequential scans/hotspots (sysptprof in sysmaster)**
- **1.17 ExecToSet to avoid iterator-return with multiple network-msgs**
- **1.1x Select distinct if you know your retrieving a single row**
- **1.1x Cache BLOB-data within datablade statics (no use, mi_lo_readwithseek is fast!)**
- **1.1x Key only selects**
- **1.1x Use one large table instead of several small**
- **1.08 Fragment index pages**
- **1.00 Fill factors,**
- **1.0 Truncated time-columns (no gain)**
- **0.8 Optimiser hints (Informix query opt. does a better job)**
- **0.5 OPTOFC/OPTMSG (FETBUFSIZE-bug)**

Domain-specific indexing extension

- Computational Indexing
- Postpone parts of indexing at insert
- Run-time indexed when query is issued
- Outperforms IBM IDS R-trees with a factor of 200 (in our applications)

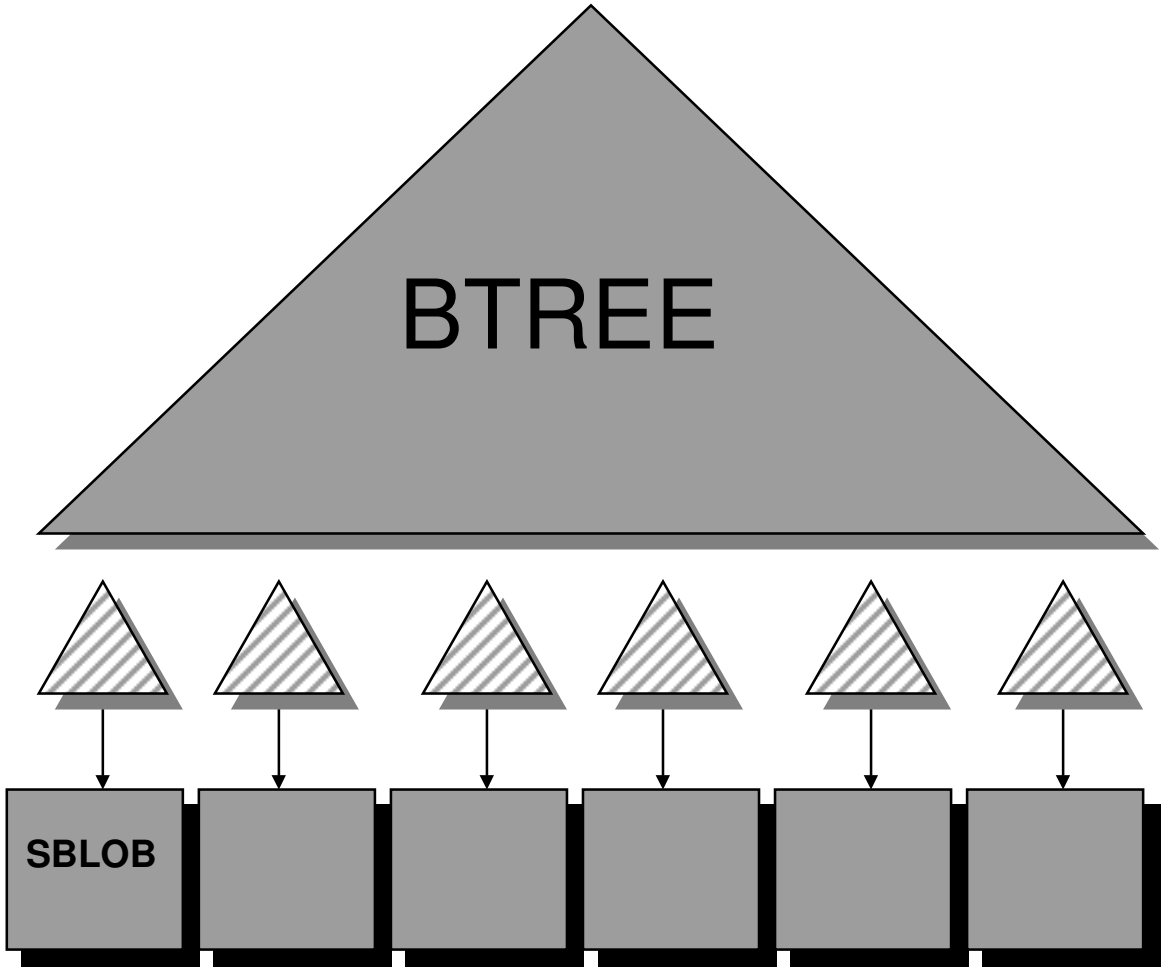
Rationale for Computational Indices

- Freshness is important
  - Must load data in (near) real-time
  - No time to index 1000000 floats during insertion
- Solution is computational indices
  - Postpone parts of the indexing built at insert time
  - Remaining index built in main-memory at run-time when doing retrieval (very fast operation)
  - Exploits key-monotonicity of inserted data
  - Example: Time-series have irregular time-stamps but the values are monotonically increasing during insertion
  - Chunks of nominal non-monotonic keys put into functional B-tree index
- Technique useful when insert flow exhibits monotonic patterns on one or more keys
- Also works when insert flow contains subsequences that exhibit monotonic patterns

Ultra-performance Spatiotemporal Index

Btree keys for nominal (non-monotonic) dimensions

Computational index

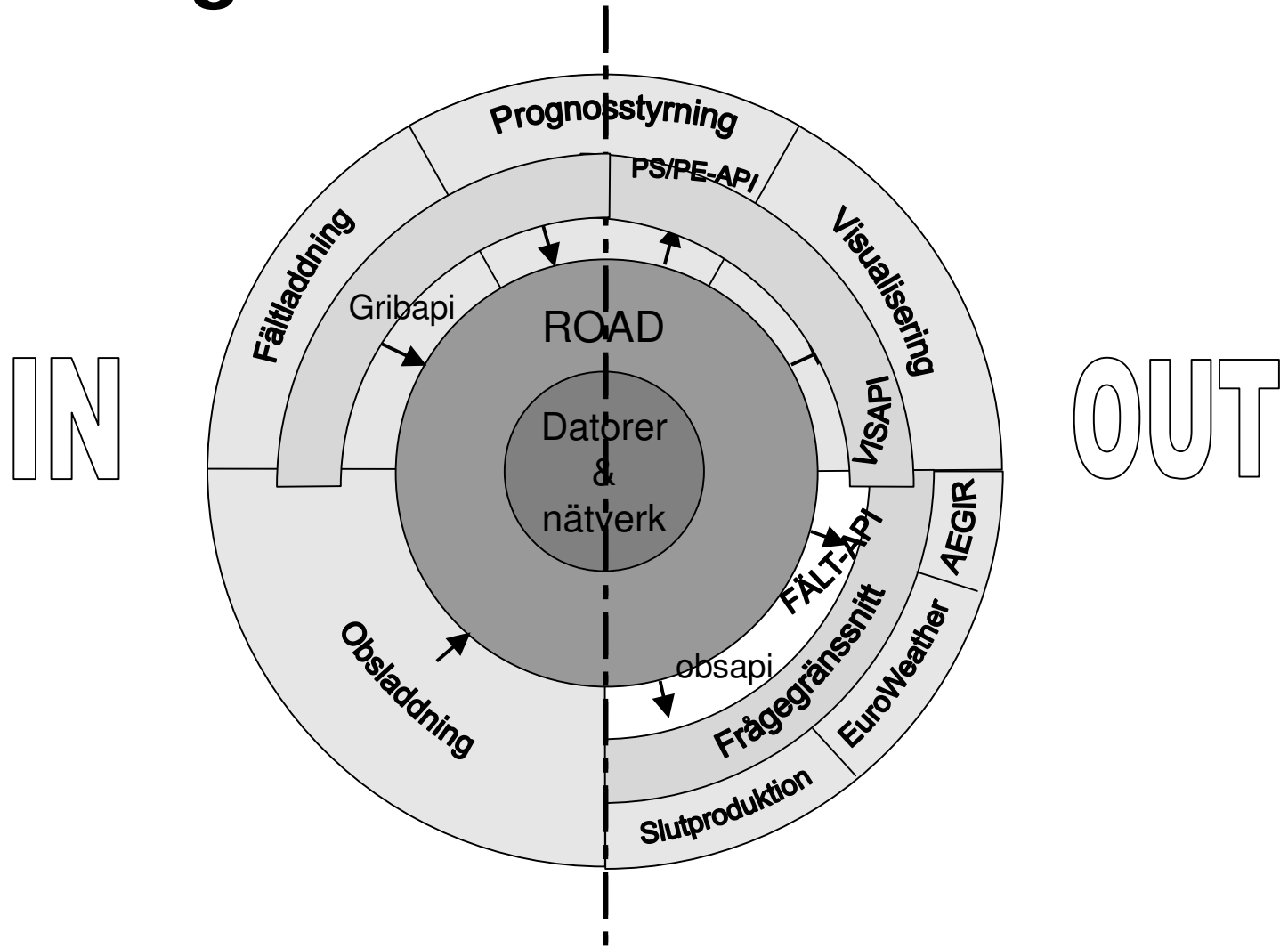


Performance of computational indices vs R-tree

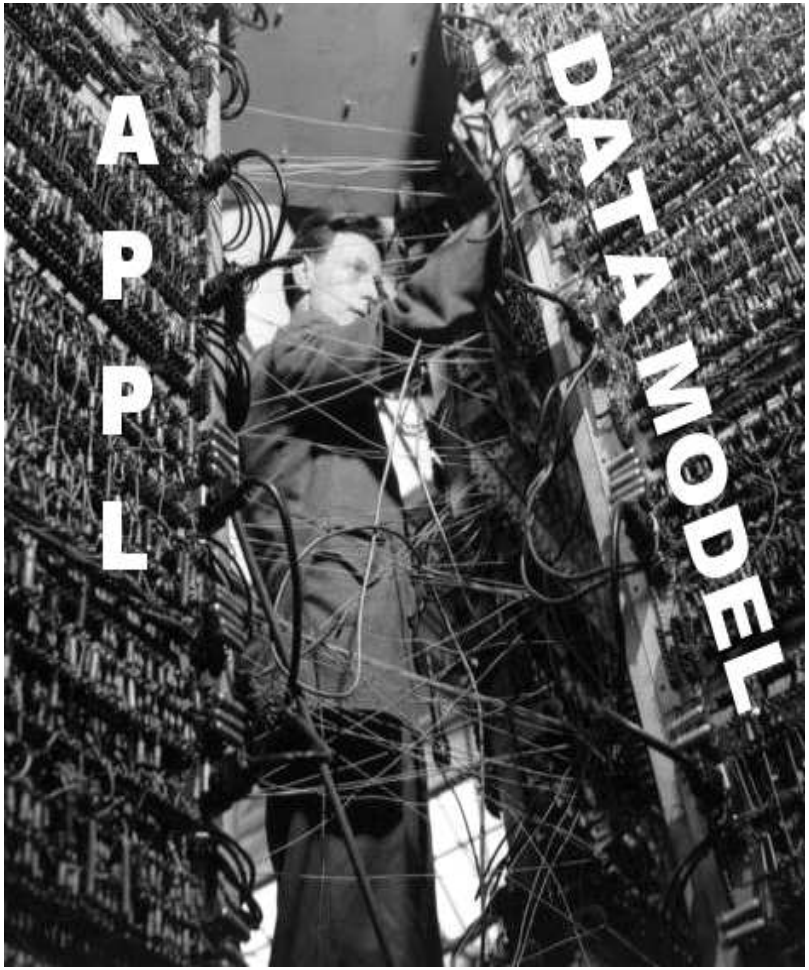
- For our applications:
- 200 times faster than R-tree at insert
- 1000 times faster than R-tree at retrieval
- Receive, store, and index 1000000 floats per seconds

Cross-enterprise retrieval

Existing APIs are hard to maintain



Entangled models

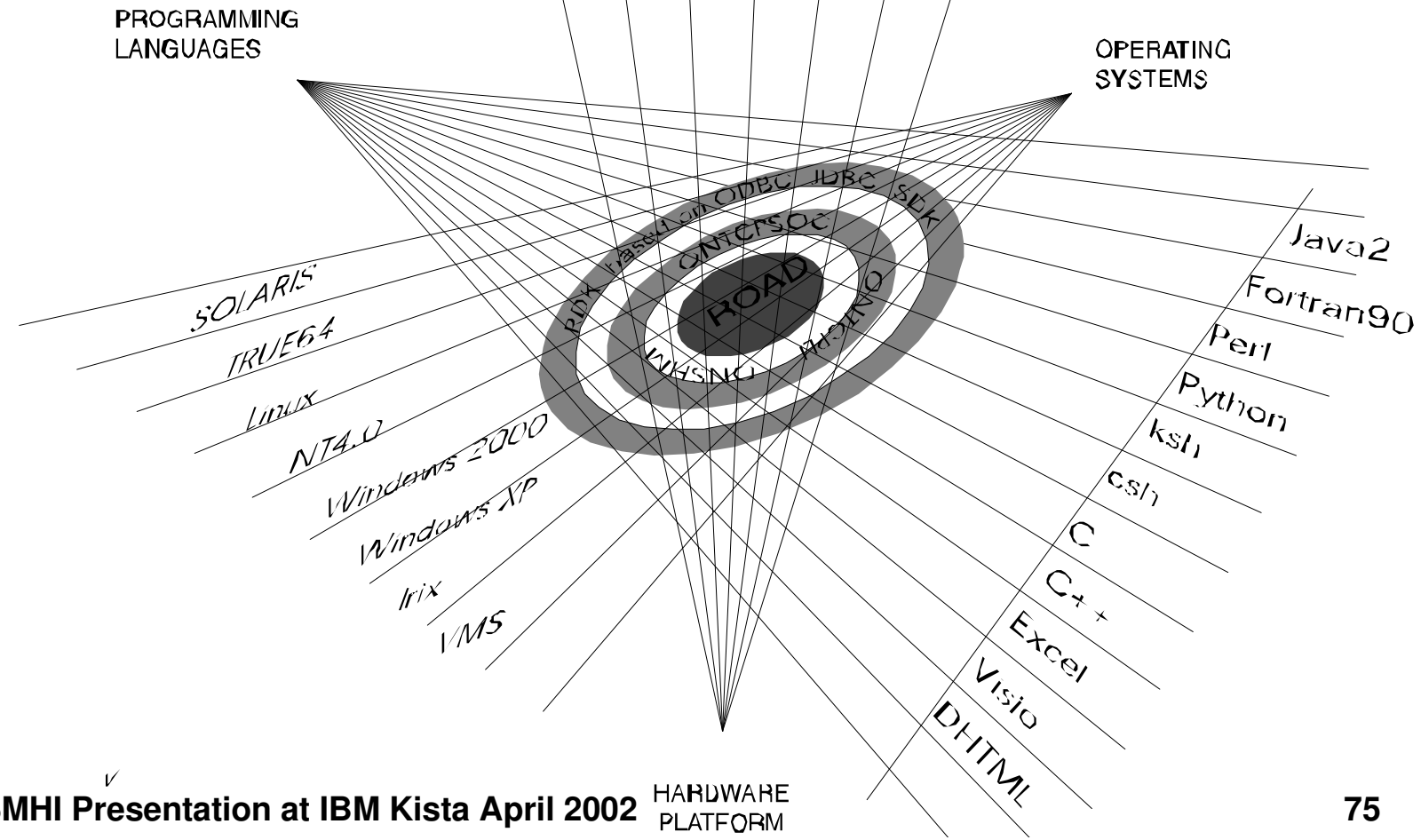


- An enterprise database is a shared resource
- Each application build their own API for accessing the information they are interested in
- Diluted competence
- Expensive maintenance
- Application and data model become entangled
- Development of database system is effectively halted
- Integration testing of change and new applications become prohibitivly tedious

Cross-enterprise retrieval of weather data

- Generation 1: C++ classes for forecasts and observations map to ESQL/C-queries (Sun/Solaris environment)
  - Generation 2: Java classes for forecasts and observations map to JDBC queries
    - Generation 3: Python interface to forecasts
      - Generation 4:
        - Generation 5:
        - Hmm…. Not a good idea….

Heterogeneous environment at SMHI



How many APIs are necessary ?

- Java/JDBC2.20, Sun Solaris
- Fortran 77, Fortran 90, Sun Solaris
- SQL (dbaccess), Sun Solaris
- Python, Sun Solaris
- Java, JDBC, Alpha True64
- ESQL/C, Alpha True64
- Fortran 77, Fortran 90, Alpha True64
- Python, Alpha True64
- Java, OpenVMS/Alpha
- ESQL/C, HP, HPUX
- Fortran 77/90, OpenVMS/Alpha
- Python, OpenVMS/Alpha
- Java, Linux/intel
- ESQL/C, Linux/intel
- Fortran 77/90, Linux/intel
- Python, Linux/intel
- Java, Windows NT/2000
- ESQL/C, Windows NT/2000
- VB6, OLE-DB, Windows NT/2000
- Python, Windows NT/2000

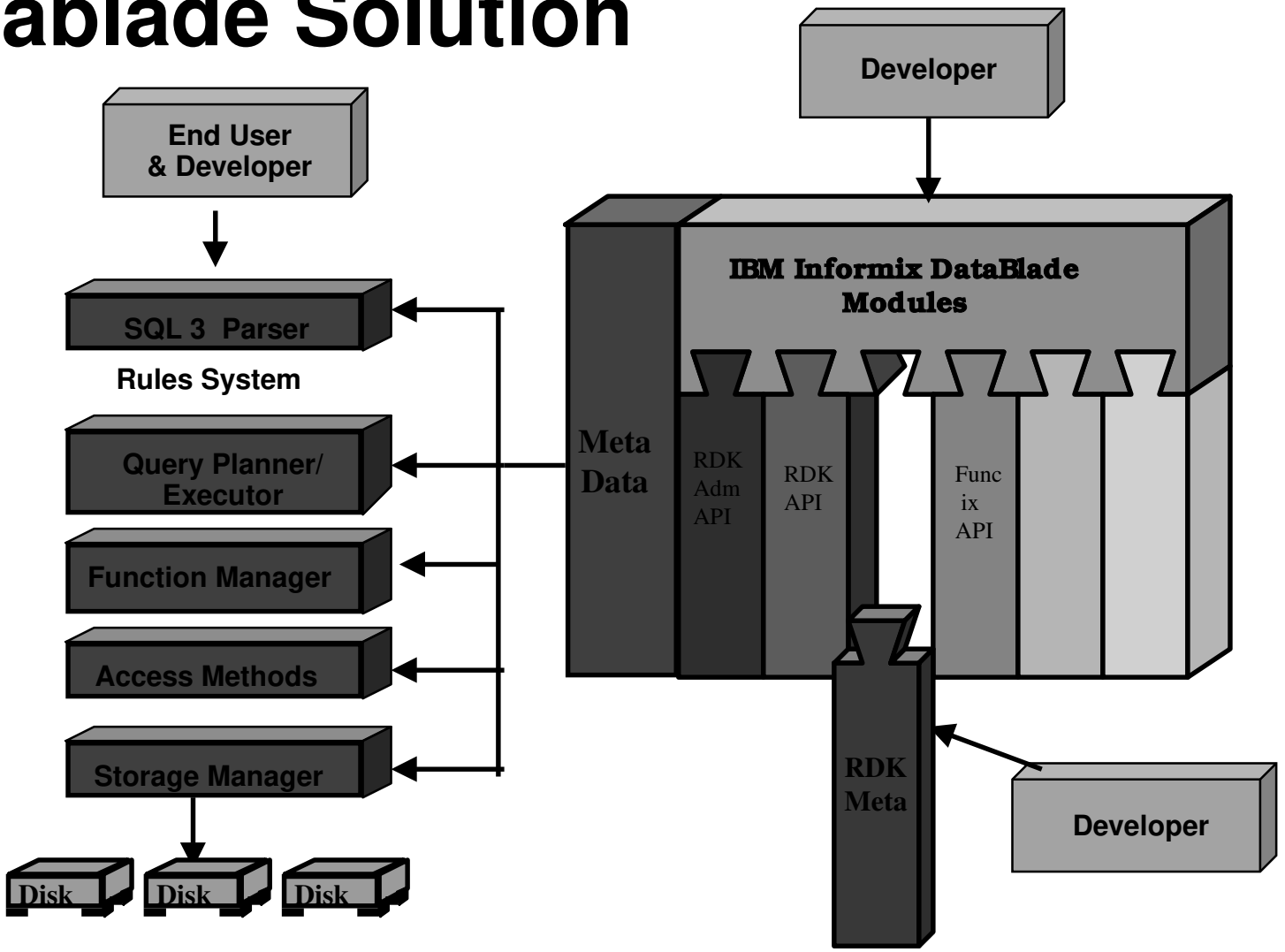
Simple/efficient access

- Goal is simple, efficient, maintable solution for access to MHO-data
- Access for non-expert
- Less than 1/2 page code for retrieval
- Support all primary platforms/languages

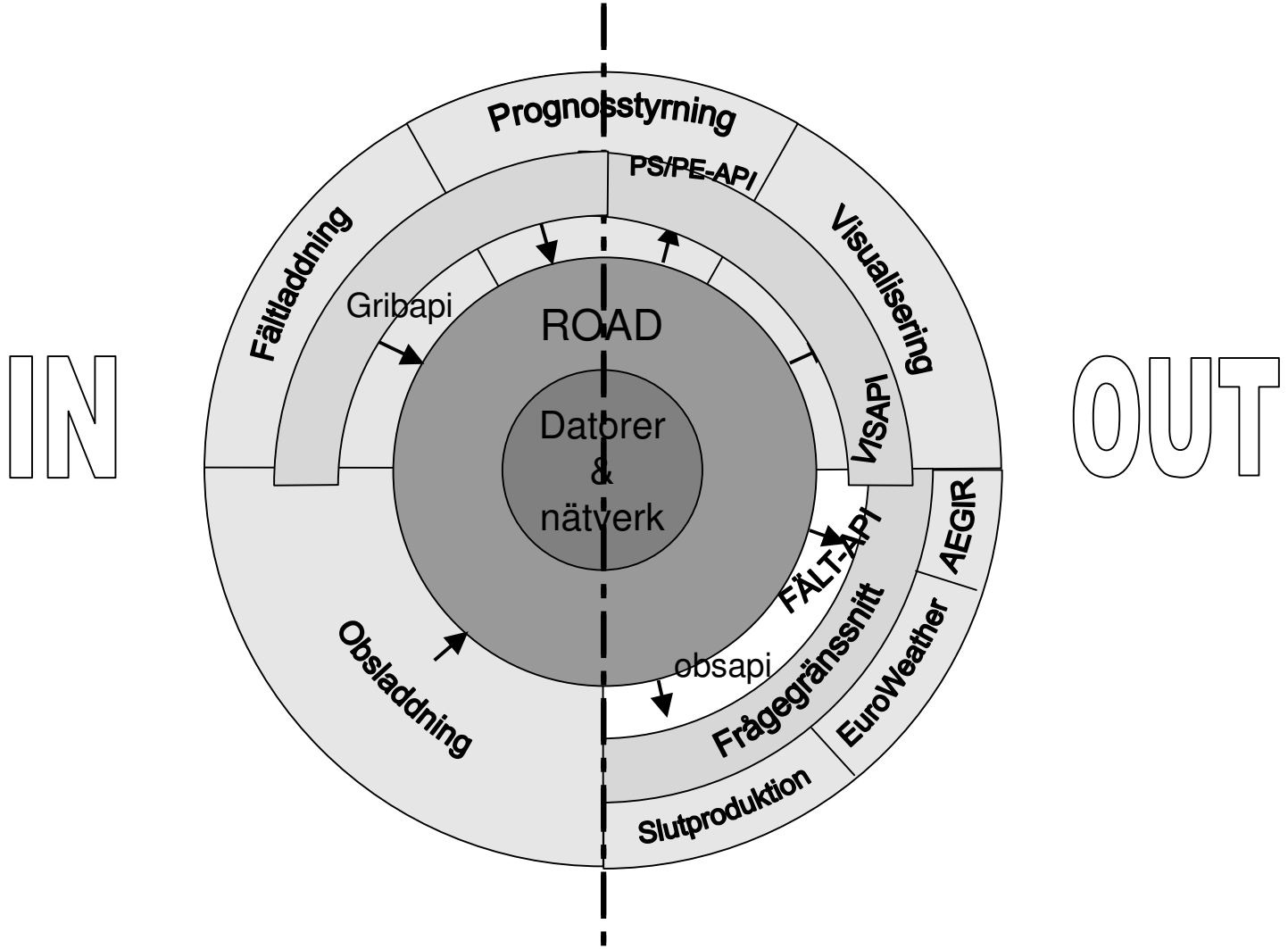
Additional requirements API

- **Maintainable**
  - **Support several API-version at the same time**
  - **Controlled access**
- **Future safe**
  - **Data model may be changed**
  - **VTI to import external data sources**
- **Extendable**
  - **New functionality can be added without affecting existing client applications**

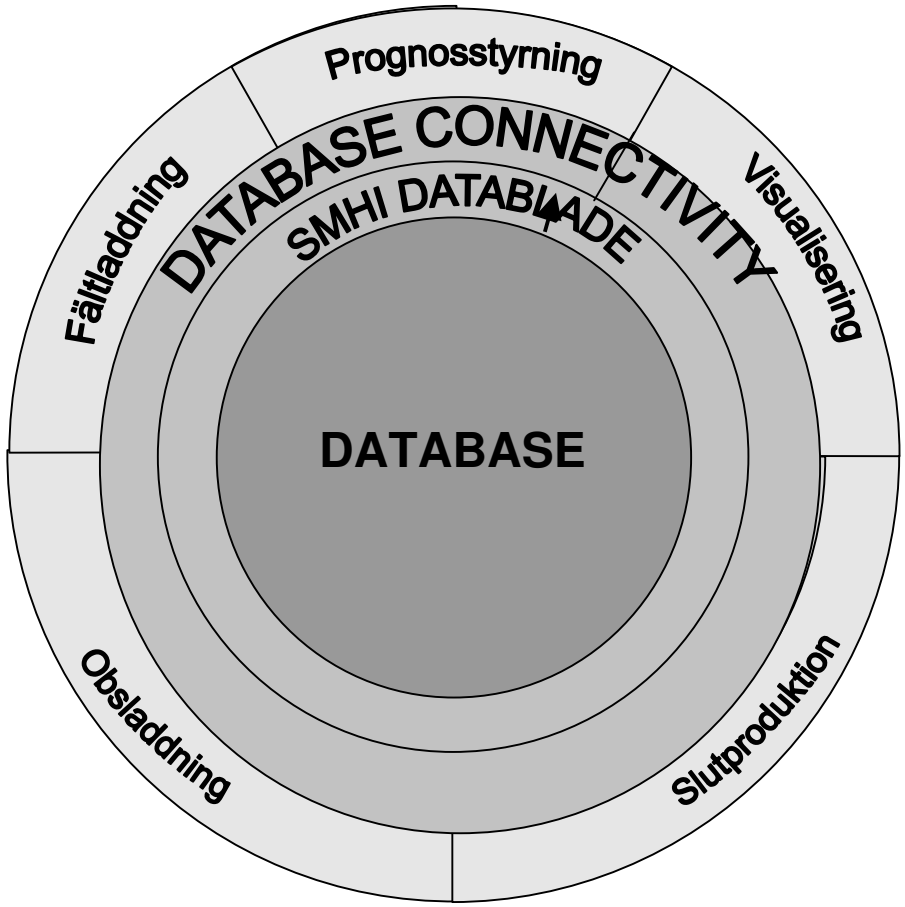
Datablade Solution



Old retrieval architecture



New retrieval architecture based on Datablade technology



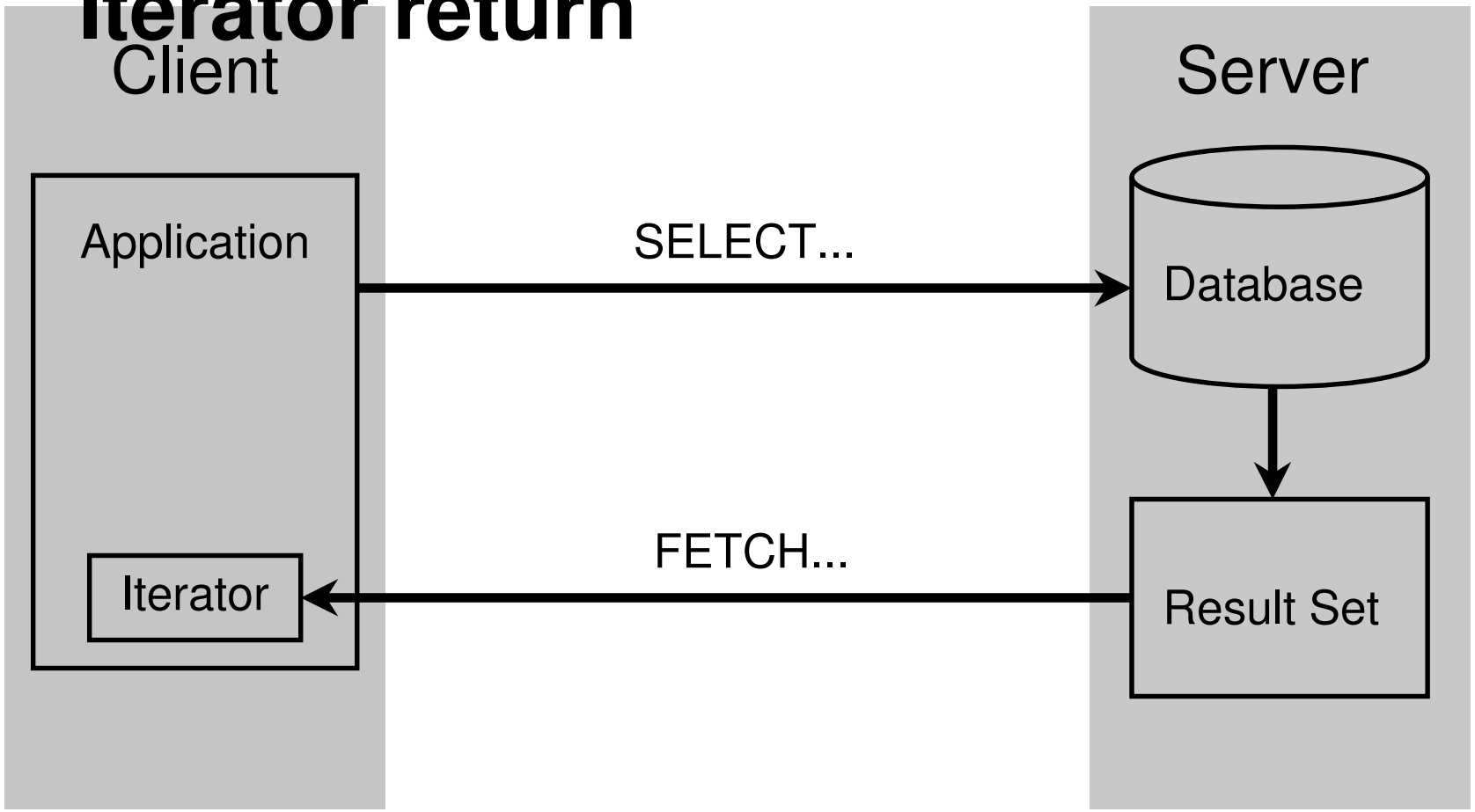
Supported database connectivity IBM Informix working for us

- IBM Informix JDBC2.20 Type 4
- Object Interface gives C++ classes for Connections, cursors, and queries
- ODBC3.51
- OLEDB version 2.0
- ESQL/C

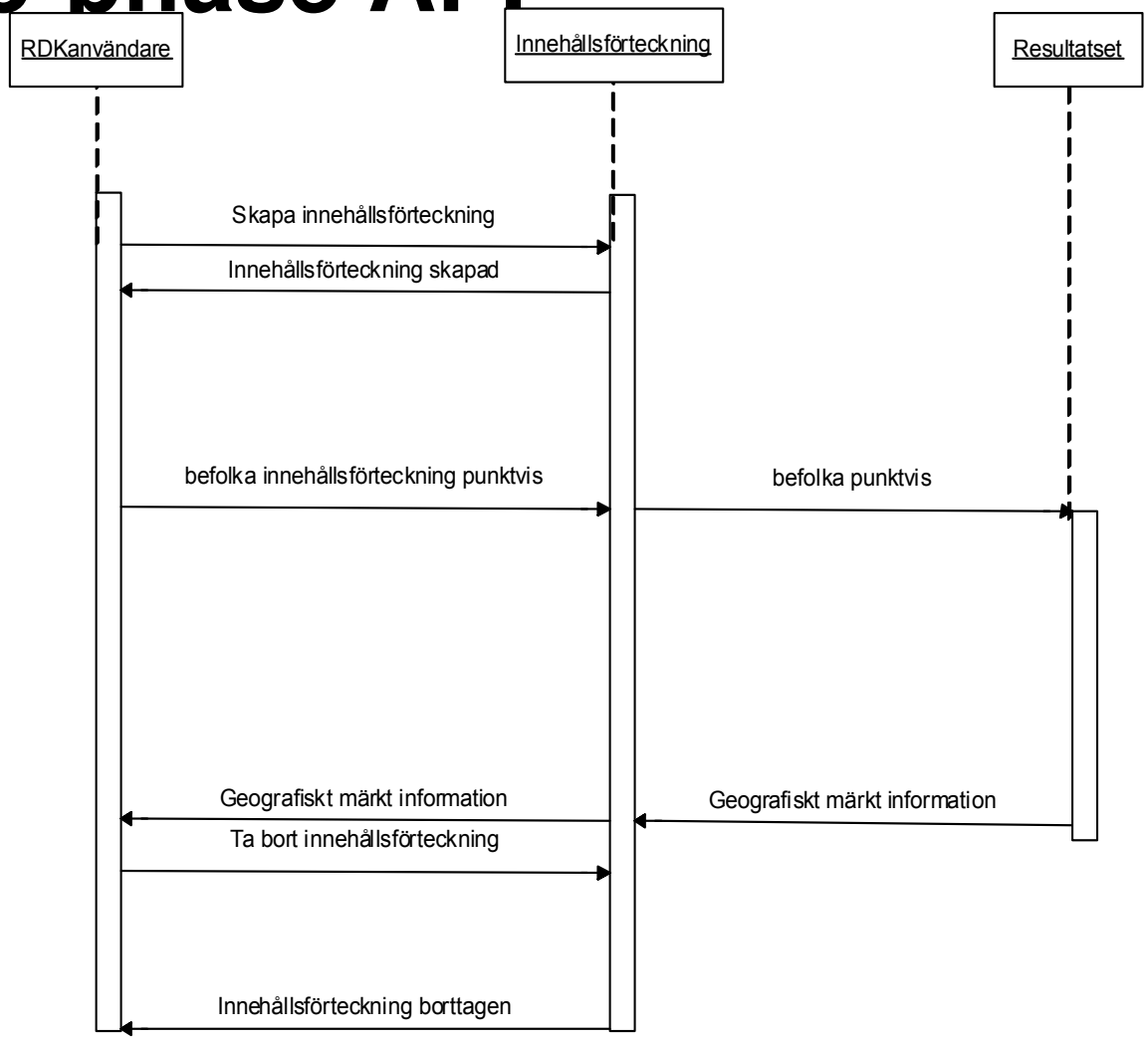
Benefits with datablade approach

- Single uniform API for all platforms
- Single uniform API for all progr langs.
- Run-time deploy (7x24)
- Single code-base for all environments
- Isolates applications from data model
- Lowered technical barrier
- RAD (rapid application development)
- Higher security
- No recompilation of client apps
- Opens access to previous isolated envs

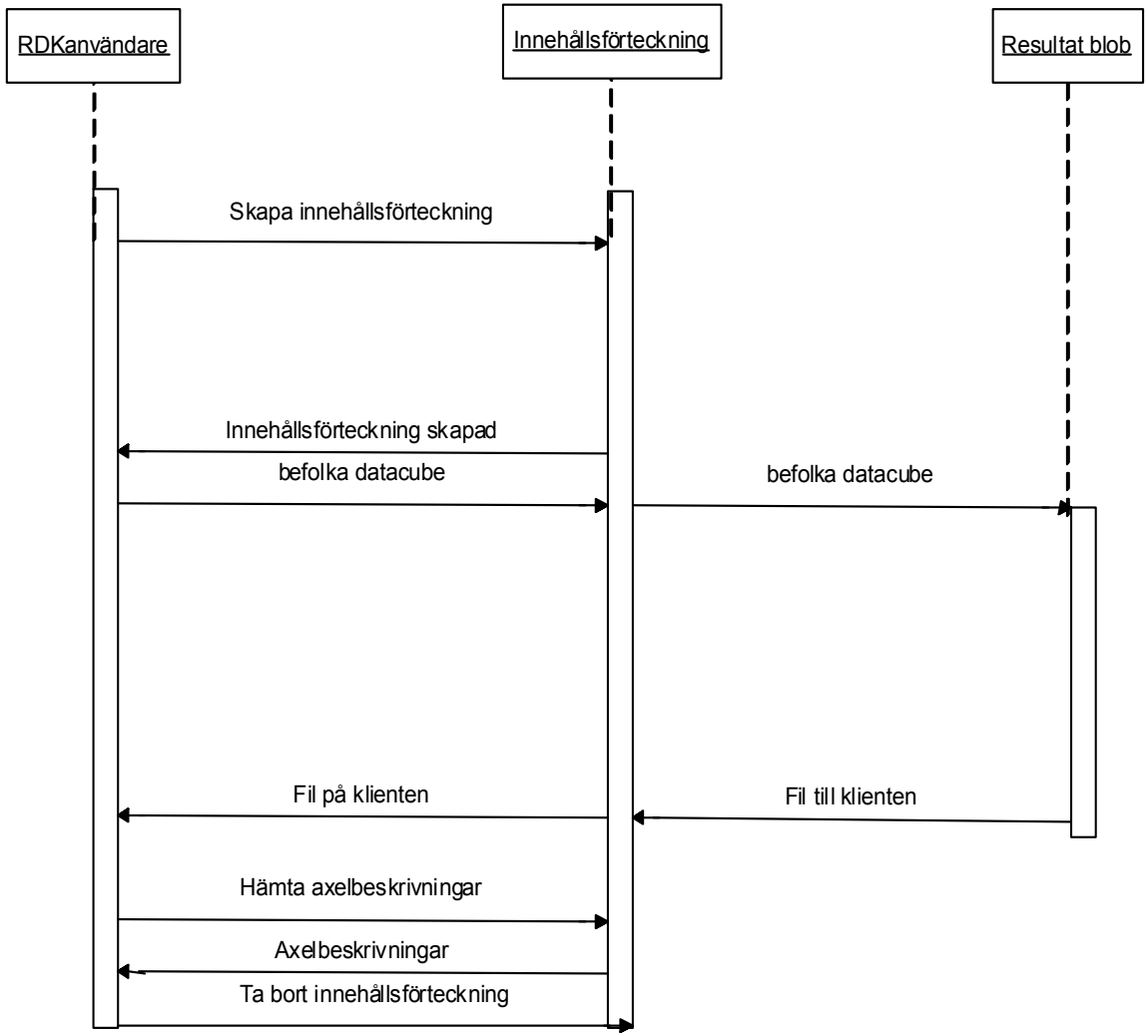
Iterator return



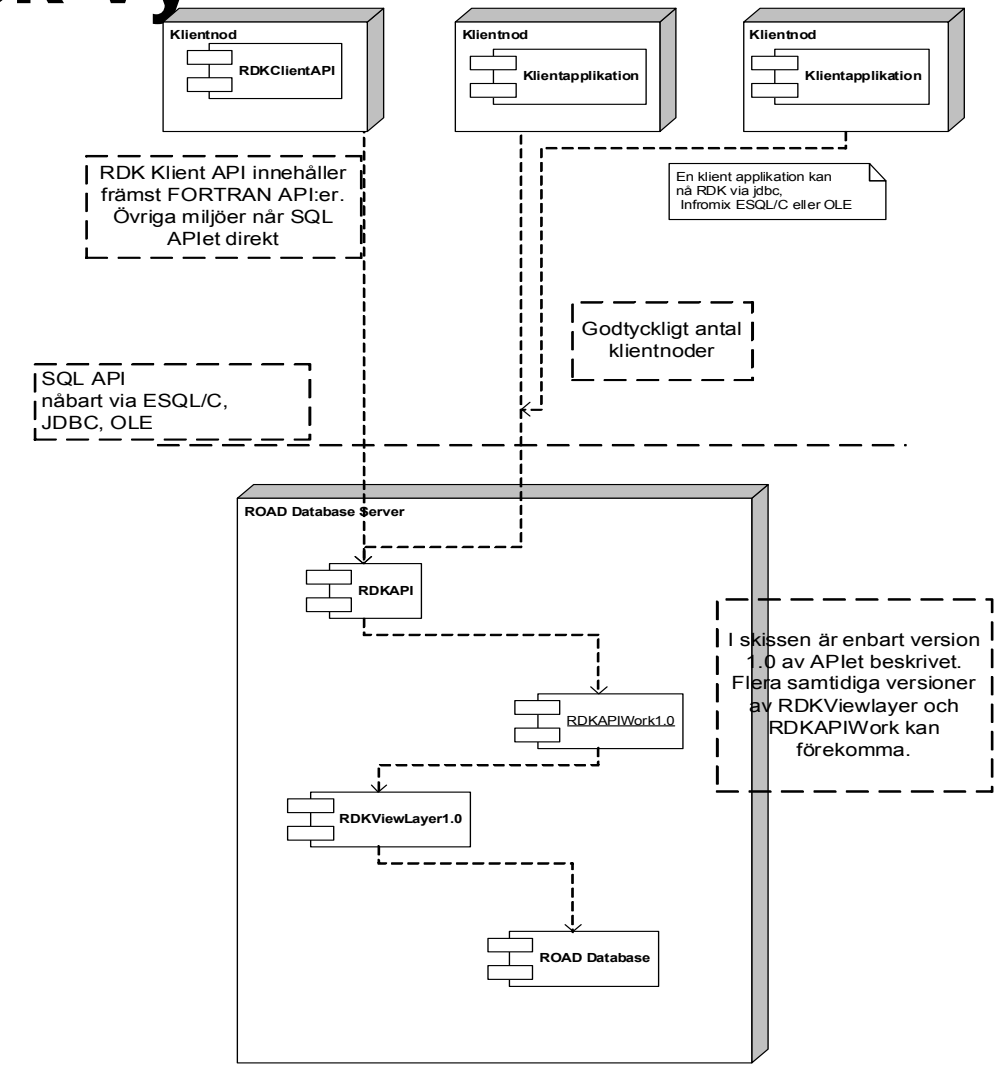
Two-phase API



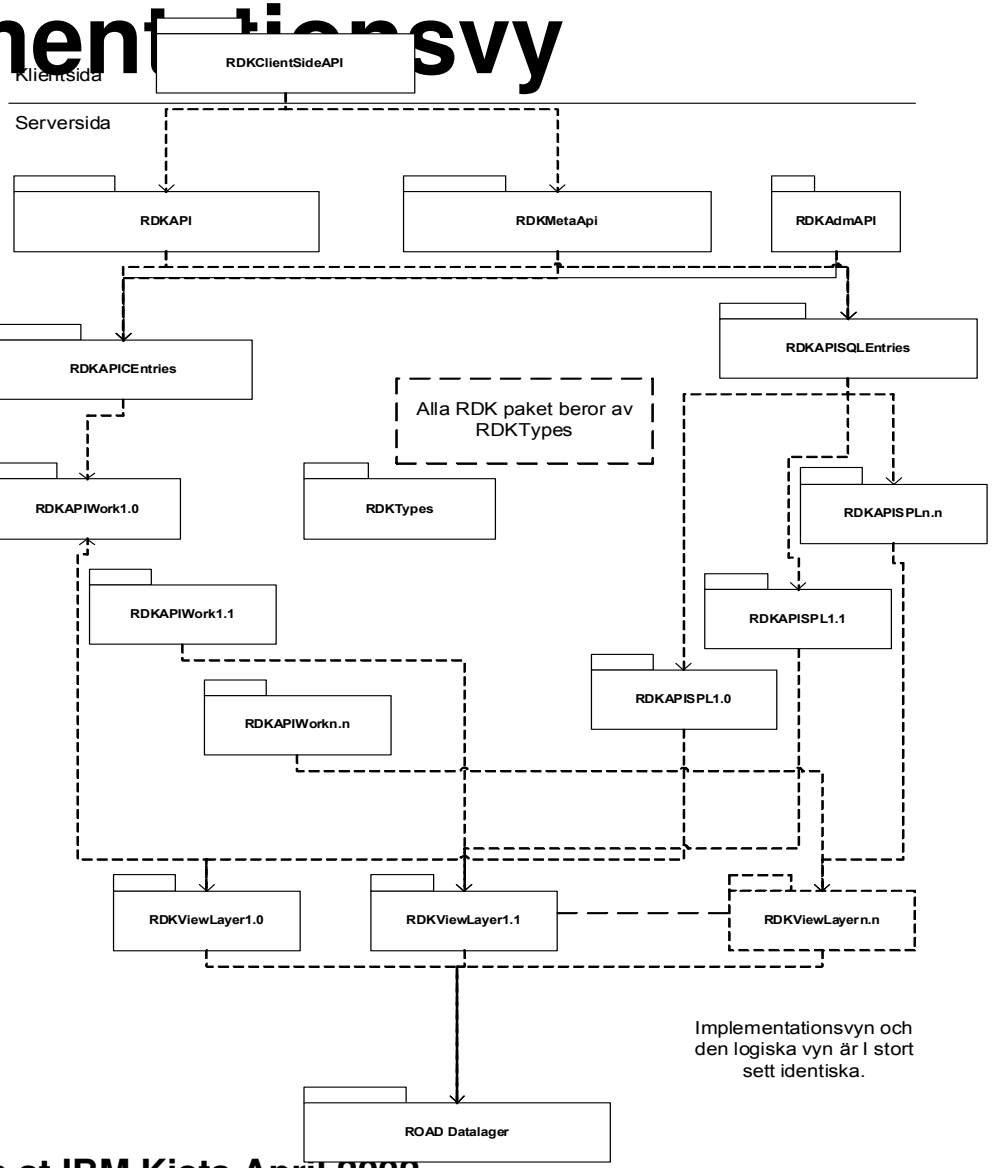
Large volumes delivered as BLOBs



Fysisk vy


Klientnod
RDKClientAPI
Klientnod
Klientapplikation
Klientnod
Klientapplikation
RDK Klient API innehåller främst FORTRAN API:er. Övriga miljöer når SQL APIet direkt
En klient applikation kan nå RDK via jdbc, Infromix ESQL/C eller OLE
Godtyckligt antal klientnoder
SQL API nåbart via ESQL/C, JDBC, OLE
ROAD Database Server
RDKAPI
RDKAPIWork1.0
RDKViewLayer1.0
ROAD Database
I skissen är enbart version 1.0 av APIet beskrivet. Flera samtidiga versioner av RDKViewlayer och RDKAPIWork kan förekomma.

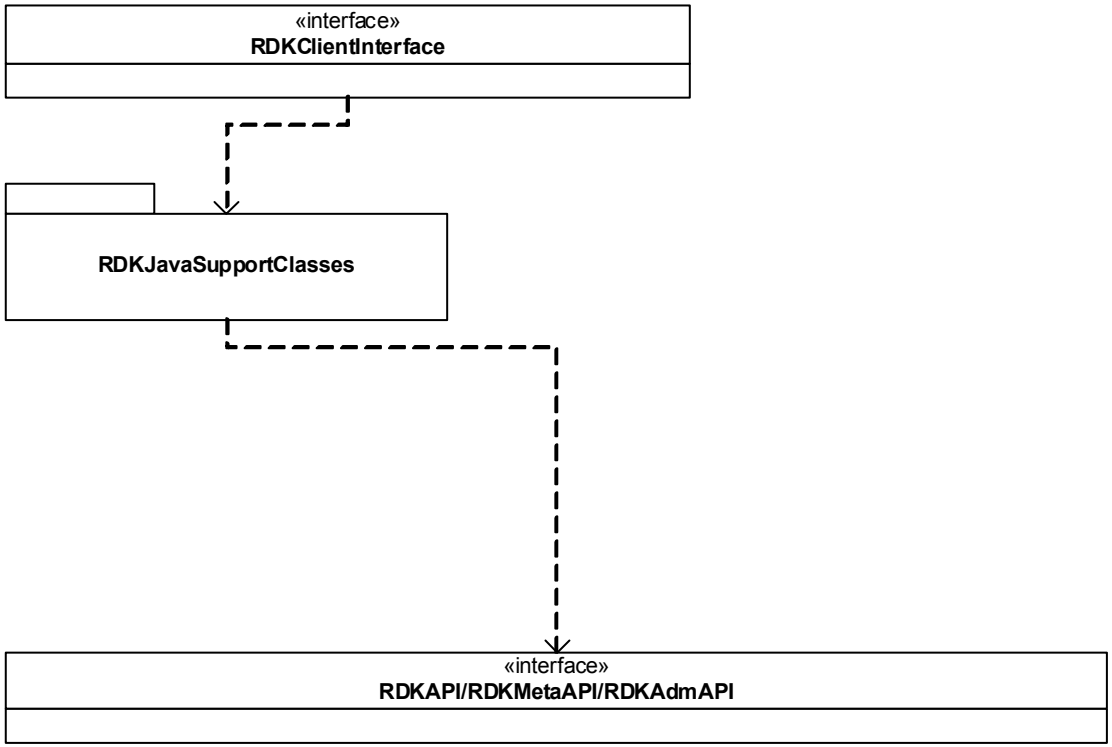
Implementationsvy



Alas, some environments require additional client code

- For imperative languages like Fortran
- For platformar not covered by database APIs
- Client mirror of server-functions
- Much like libDMI

Fortran connectivity



JNI-bridge to IBM Informix

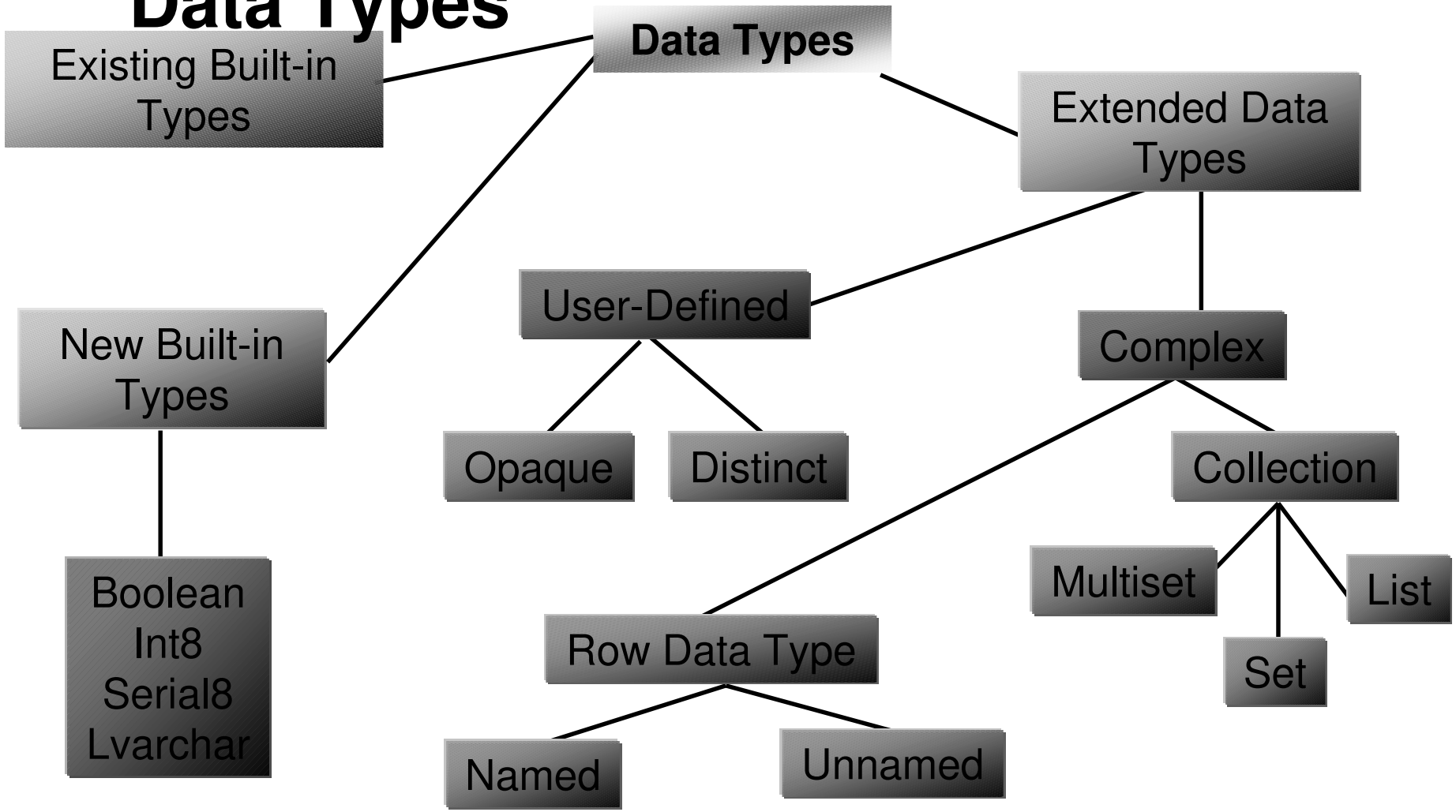
- Client invokes RDK function wrapper
- Client instansiate a Java Virutal machine
- JNI, Java Native Interface utnyttjas för att anropa javakod
- Jdbc- kommunikation med RDK-serverkomponenter

Dimensions become UDR arguments

- **källtyp**
- **källa**
- **parameter**
- **nivåparameter**
- **nivåinformation**
- **geografi, geo (x,y, höjd, tidsplanet och srid). Anm. srid är anger vilket koordinatsystem som den geografiska informationen är given i.**
- **referenstid (referenstid = analystid för prognosfält och observationstid för observationer).**
- **Lagringstid i datakällan**
- **version, dataversion (typiskt för så kallade ensembleprognoser)**
- **Kvalitetsmask**
- **Ytterligare dimensioner kan tillkomma i kommande versioner…**

IBM IDS Extensibility -- use at SMHI

Complex and User-Defined Data Types



IBM IDS Extensible Type System

Mechanism	Example	Strengths and Weaknesses
Built-In Types	INTEGER, VARCHAR, DATE etc. These are standardized in the SQL-92 language specification.	Mature and high performance because they are compiled into the ORDBMS. But they are very simple. Good building-blocks for other types.
DISTINCT	CREATE DISTINCT TYPE String AS VARCHAR(32);	Simple to create, and useful when what you want is something very close to another type.
ROW TYPES	CREATE ROW TYPE Address (Address_Line_One String NOT NULL, Address_Line_Two String NOT NULL, City String NOT NULL, State String, ZipCode PostCode, Country String NOT NULL);	Relatively easy to use means of combining pre-existing types into a more complex objects, and enforcing rules about contents. ROW TYPEs have several drawbacks that makes them a poor choice for types to define columns.
Java Classes	Combination of Java UDRs with opaque data storage.	More complex to develop, but an excellent choice when you want code that runs in both the outside, and inside, the DBMS.
OPAQUE TYPES	CREATE OPAQUE TYPE GeoPoint (internallength = 16);	Most complex to develop, but these are the most powerful in terms of performance, scalability and the range of object sizes that can be supported.

SMHI Extended types

- **Distinct types**
  - **create distinct type 'informix'.rdksource as integer;**
- **Opaque types**
  - **create opaque type 'informix'.rdkdimension (internallength=4, alignment=4);**
- **Row type**
  - **create row type 'informix'.rdkfloatpoint**
    (ibtype rdkibtype, source rdksource, parameter rdkparameter,
    levelparameter rdklevelparameter,
    reftimebegin rdkreftimebegin,
    reftimeend rdkreftimeend,
    value decimal(16),
    qualitymask rdkqualitymask,
    geo geoobject,
    storetime rdkstoretimeend);

Create function (SPL-prototype)

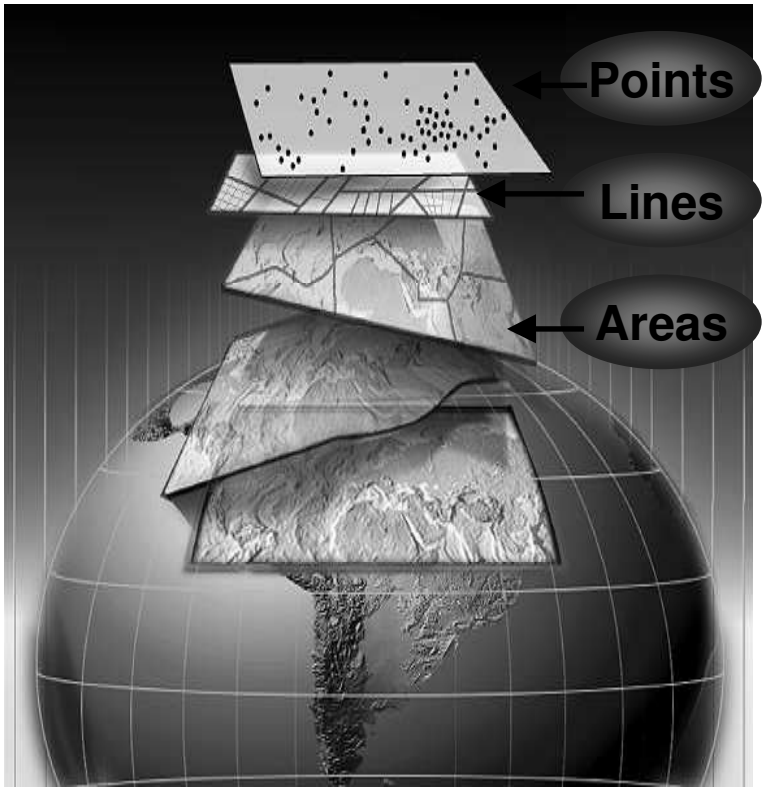
```
create function "informix".rdkpopulatefloatpointwise(toc RDKTocHandle,authToken RDKAuthToken,qualityMask RDKQualityMask,debug RDKDebugFlag)
returns RDKFloatPointwise
define result RDKFloatPointwise;
define v_geo geoobject;
….
foreach cursor for
  select ibtypeid, source, parameter, levelparameter, levelinfo, reftime::RDKReftimeBegin,
       storetime::RDKStoreTimeEnd, quality, image::lvarchar, tableid, key,
       origgeoobject, usergeoobject::lvarchar, nrx, nry, xincr, yincr, startlat, startlong, polelat, polelong, projection
  into result.ibtype, result.source, result.parameter, result.levelparameter,
       result.levelinfo, result.reftimebegin, result.storetime, result.levelinfo, result.reftimebegin, result.storetime,
       result.qualitymask, v_blob, tid, v_key,
       v_geo, v_usergeo, v_nrx, v_nry, v_xincr, v_yincr, v_startlat, v_startlong, v_polelat, v_polelong, v_projection
  from tocrows where ….
…..
return result with resume;
       end foreach
     else
       raise exception -999;      end if;
   end if

end foreach

end function;
```

Create function (C-routine)

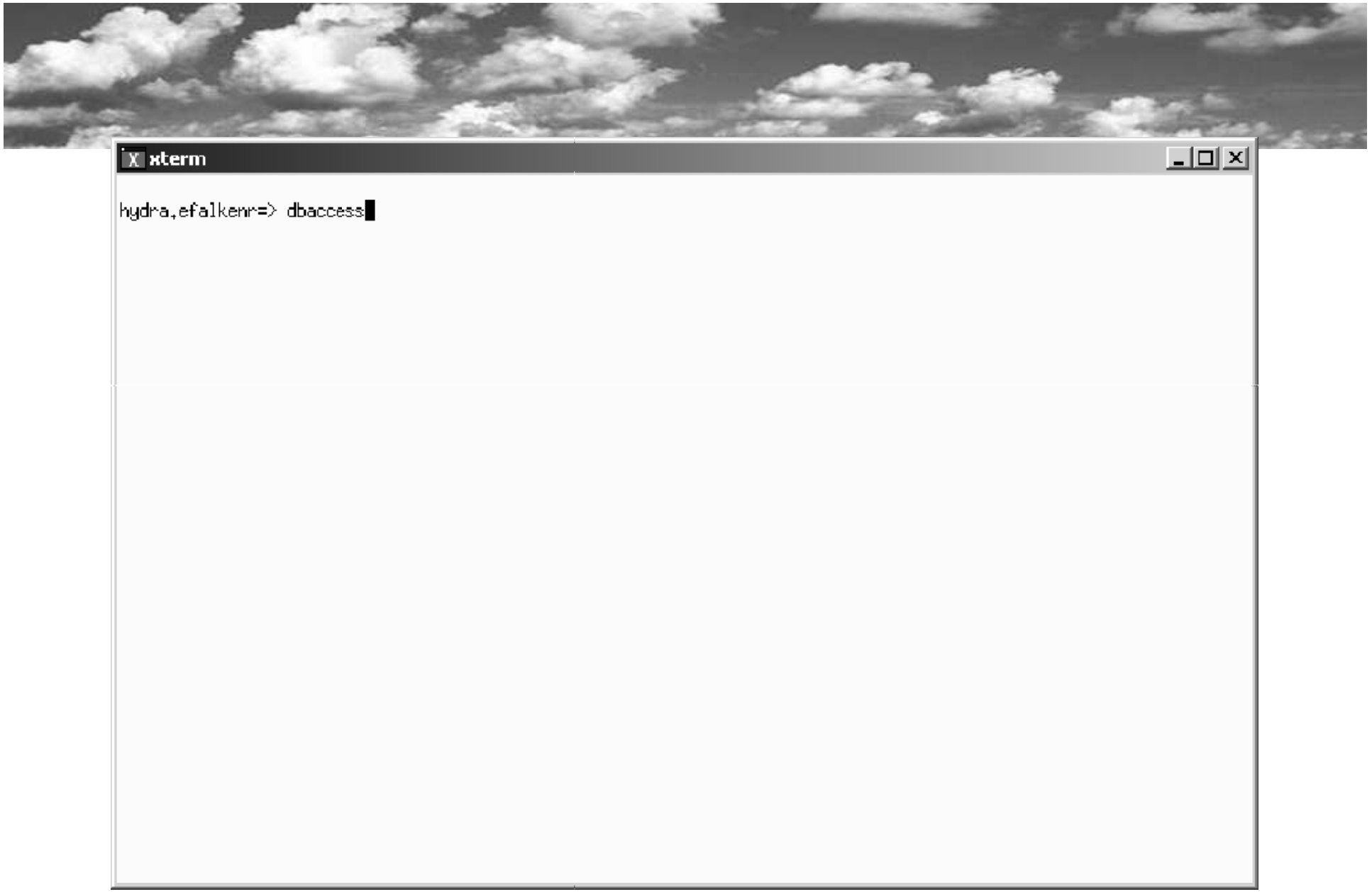
```
create function "informix".lon(GeoPoint) returns GeoLongitude
external name
    "$INFORMIXDIR/extend/RoadIndexFunctions.1.0/RoadIndexFunctions.bld(lo
    n)" language c;

alter routine  "informix".lon (GeoPoint)
        with (add parallelizable);
alter routine  "informix".lon (GeoPoint)
        with (add not variant);
```

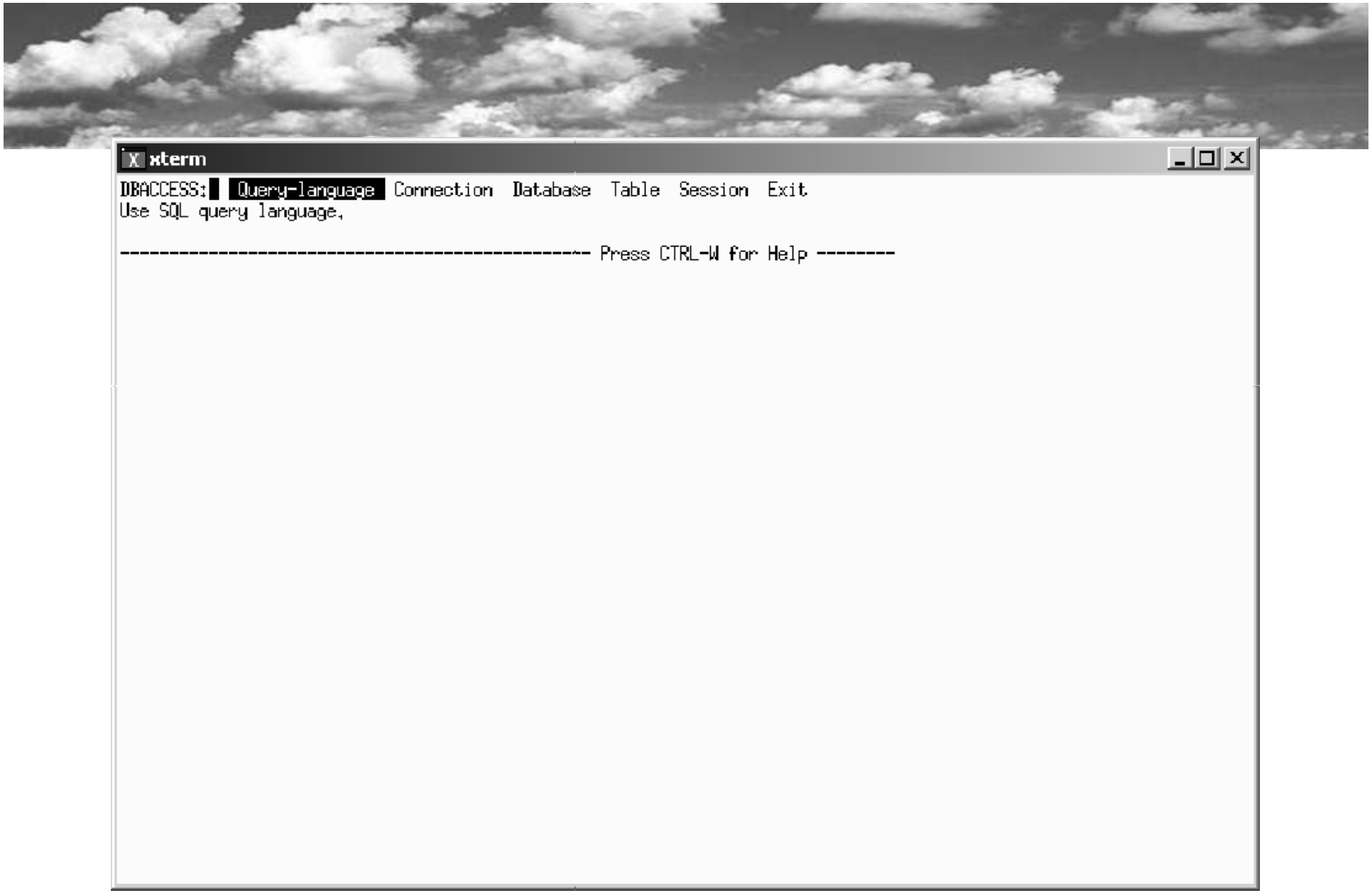
DEMO

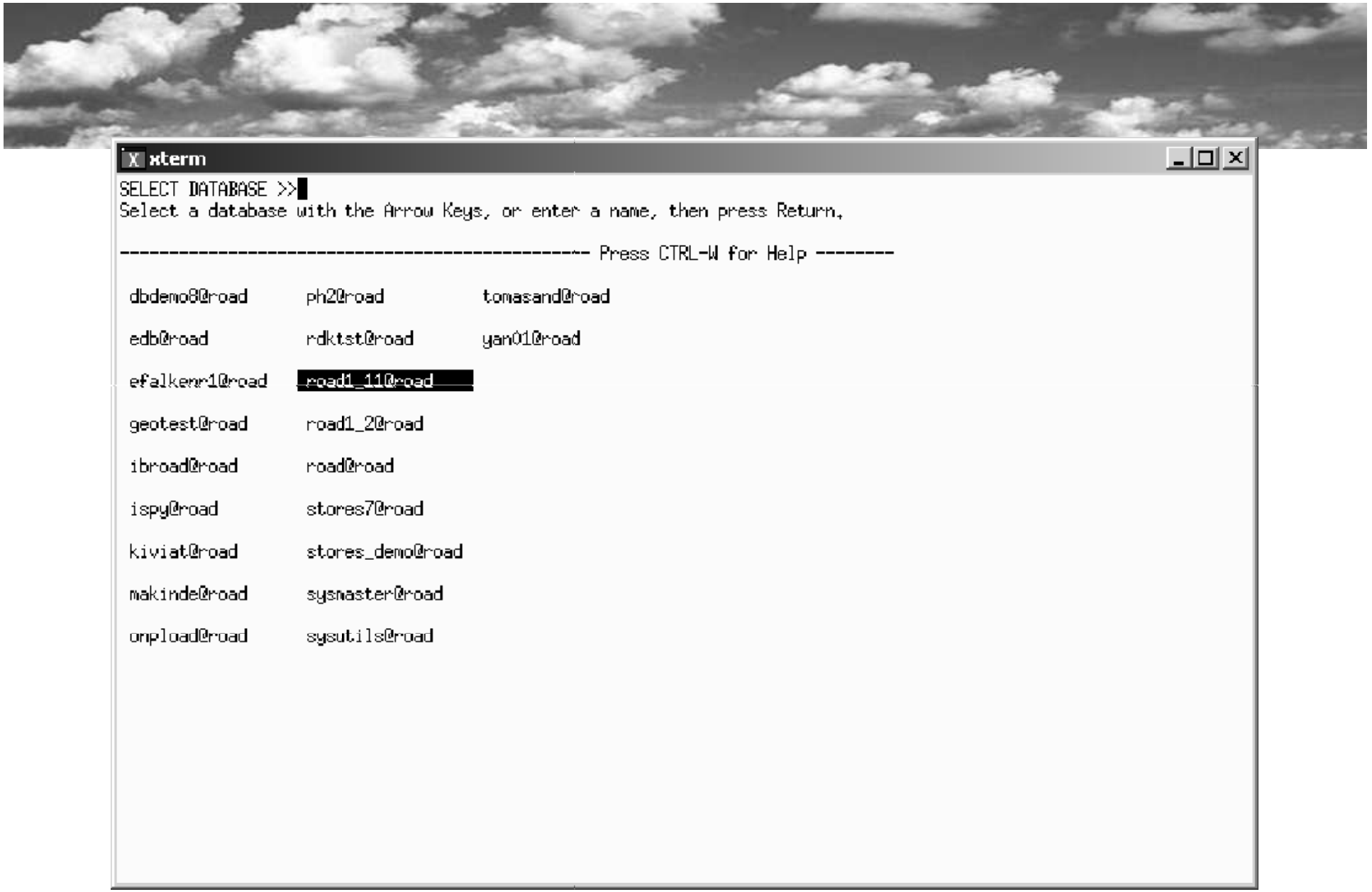
DEMO Weather in Stockholm

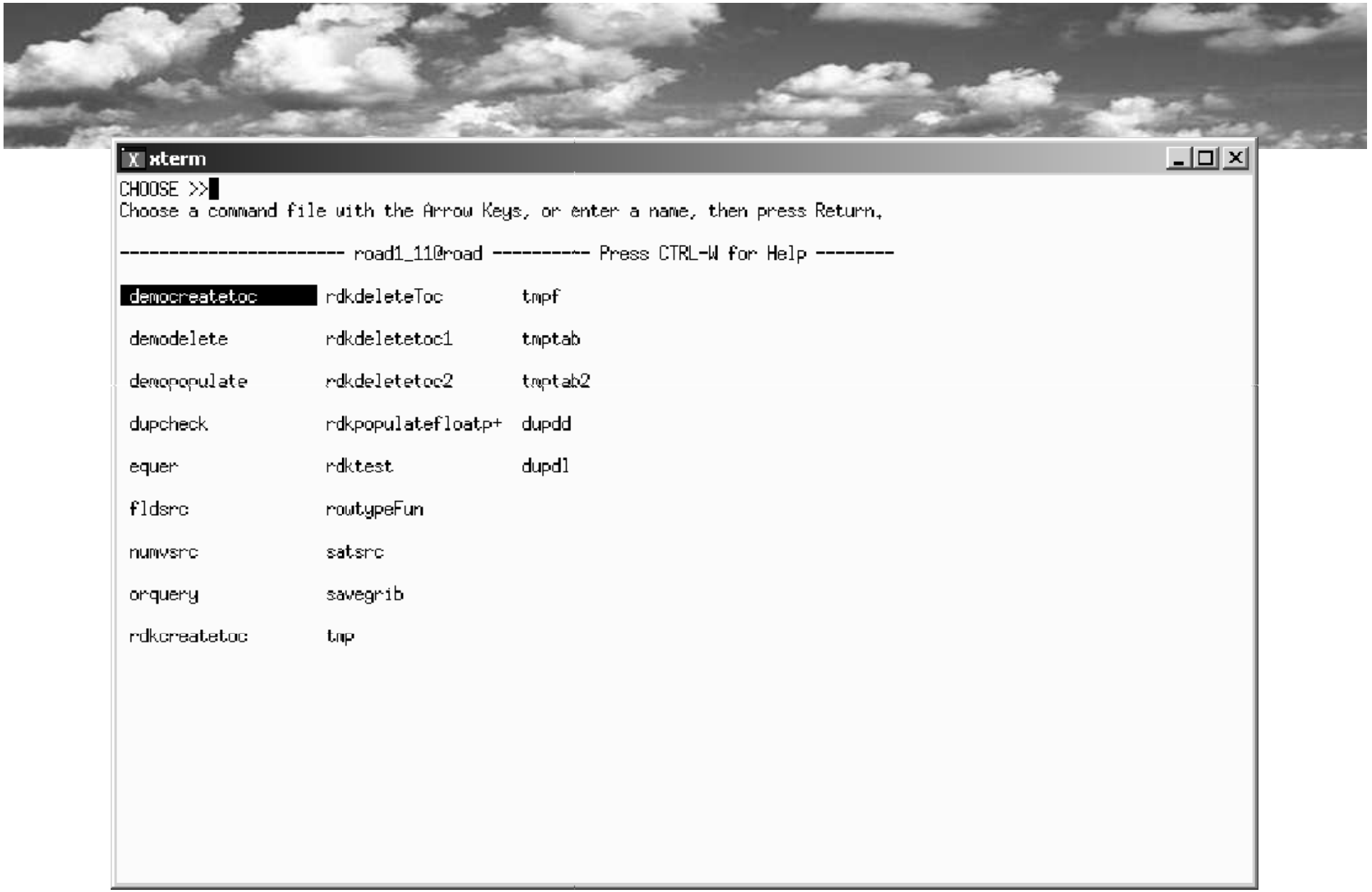


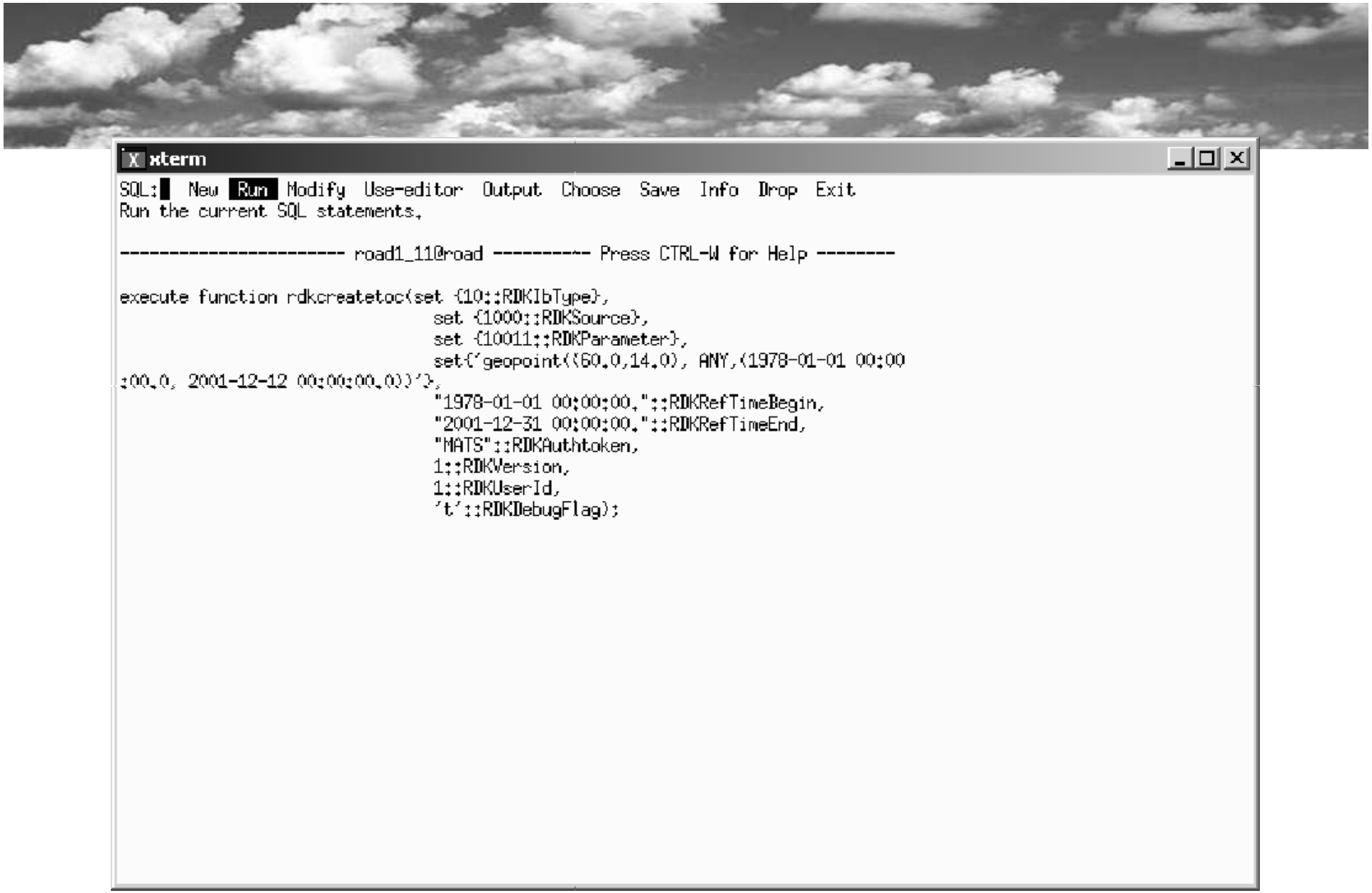
- Specify area as point, circle, box, polygon
- Specify time interval
- Specify type product
  - Text
  - Probability
  - Symbol
  - Numerical values
  - etc.

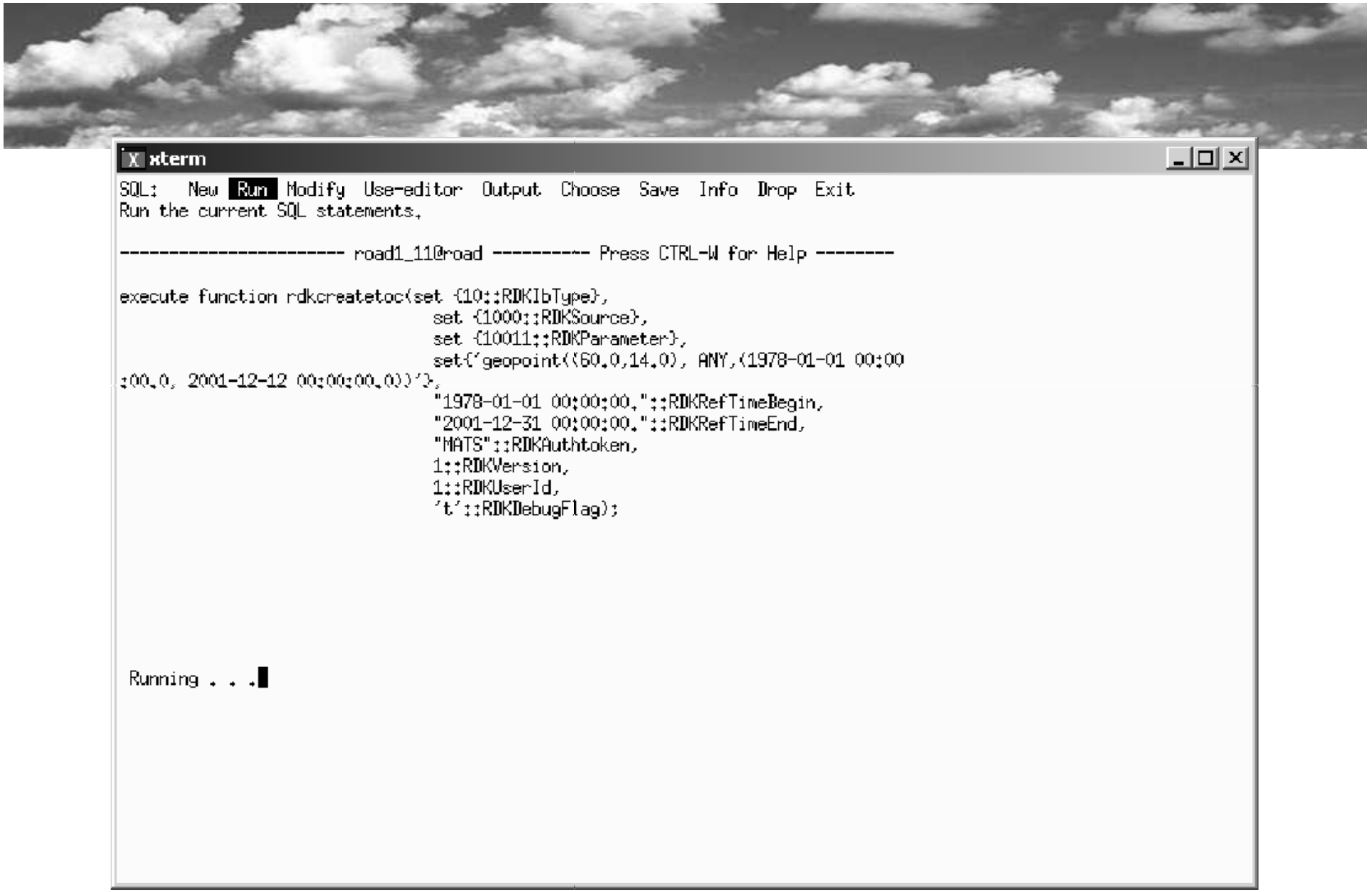

xterm
hydra,efalkenr=> dbaccess

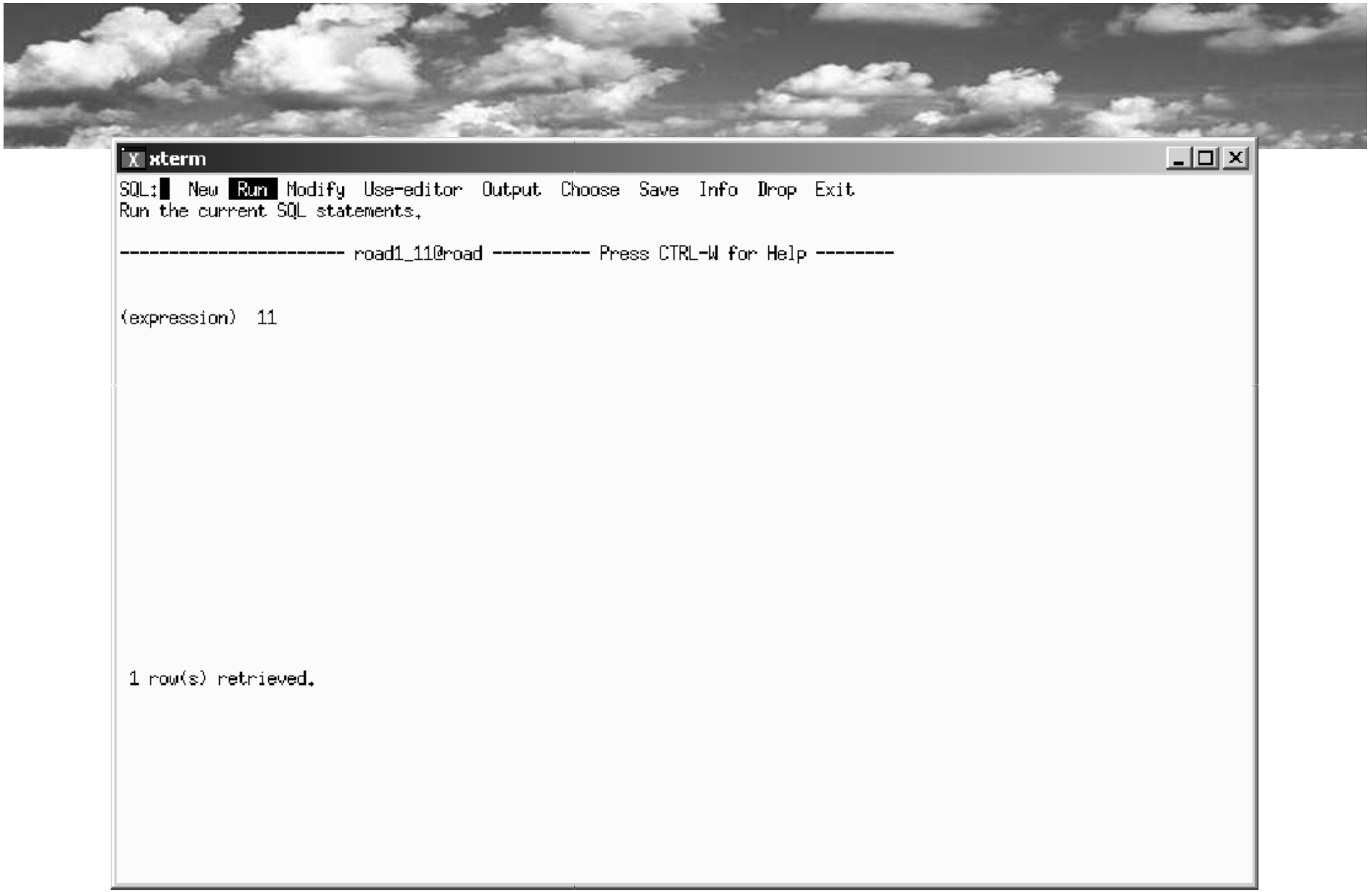

xterm
DBACCESS: Query-language Connection Database Table Session Exit
Use SQL query language.
---------- Press CTRL-W for Help --------

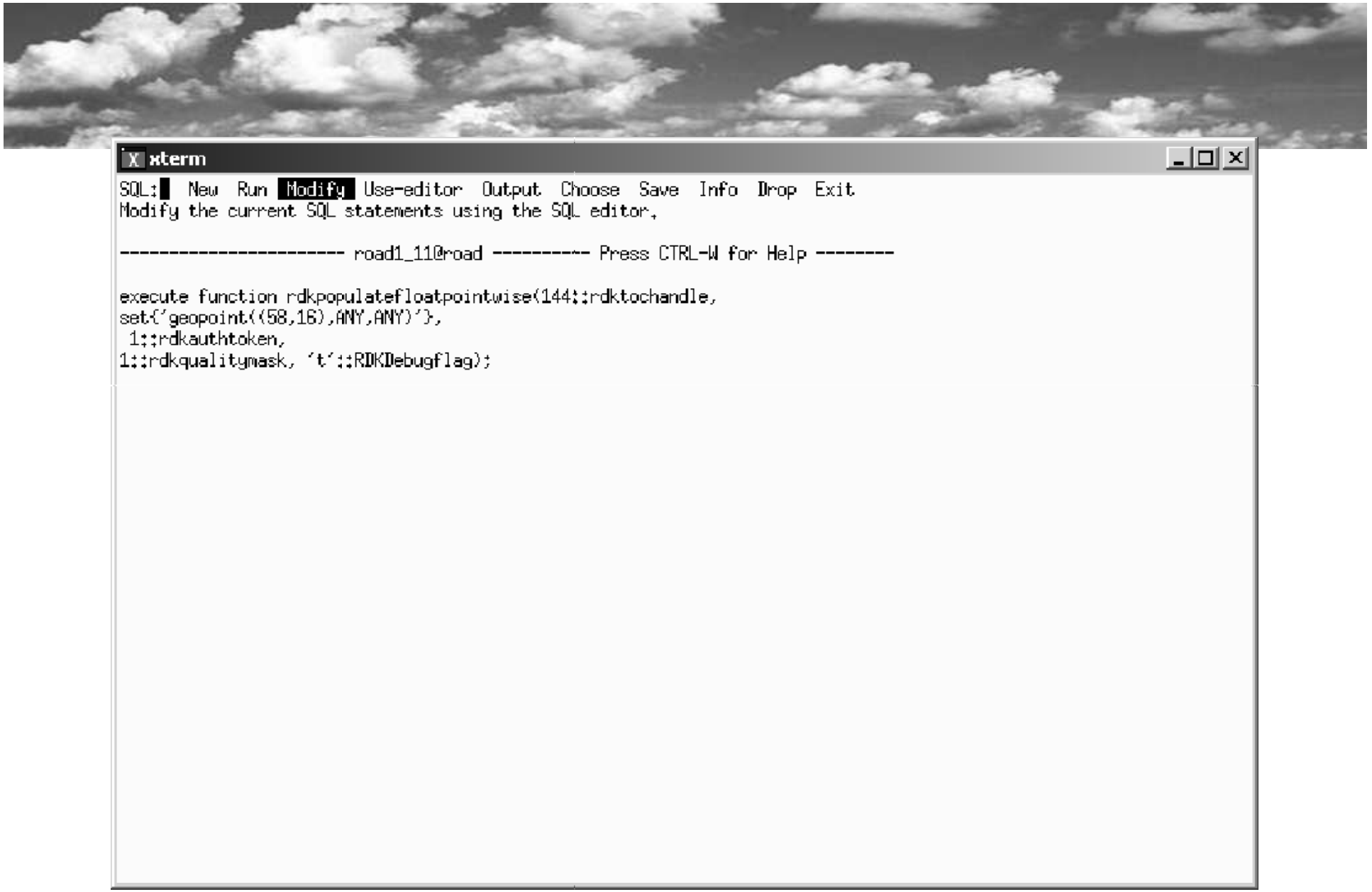

xterm
SELECT DATABASE >>
Select a database with the Arrow Keys, or enter a name, then press Return.
---------- Press CTRL-W for Help --------
dbdemo8@road ph2@road tomasand@road
edb@road rdktst@road yan01@road
efalkenr1@road road1_11@road
geotest@road road1_2@road
ibroad@road road@road
ispy@road stores7@road
kiviat@road stores_demo@road
makinde@road sysmaster@road
onpload@road sysutils@road

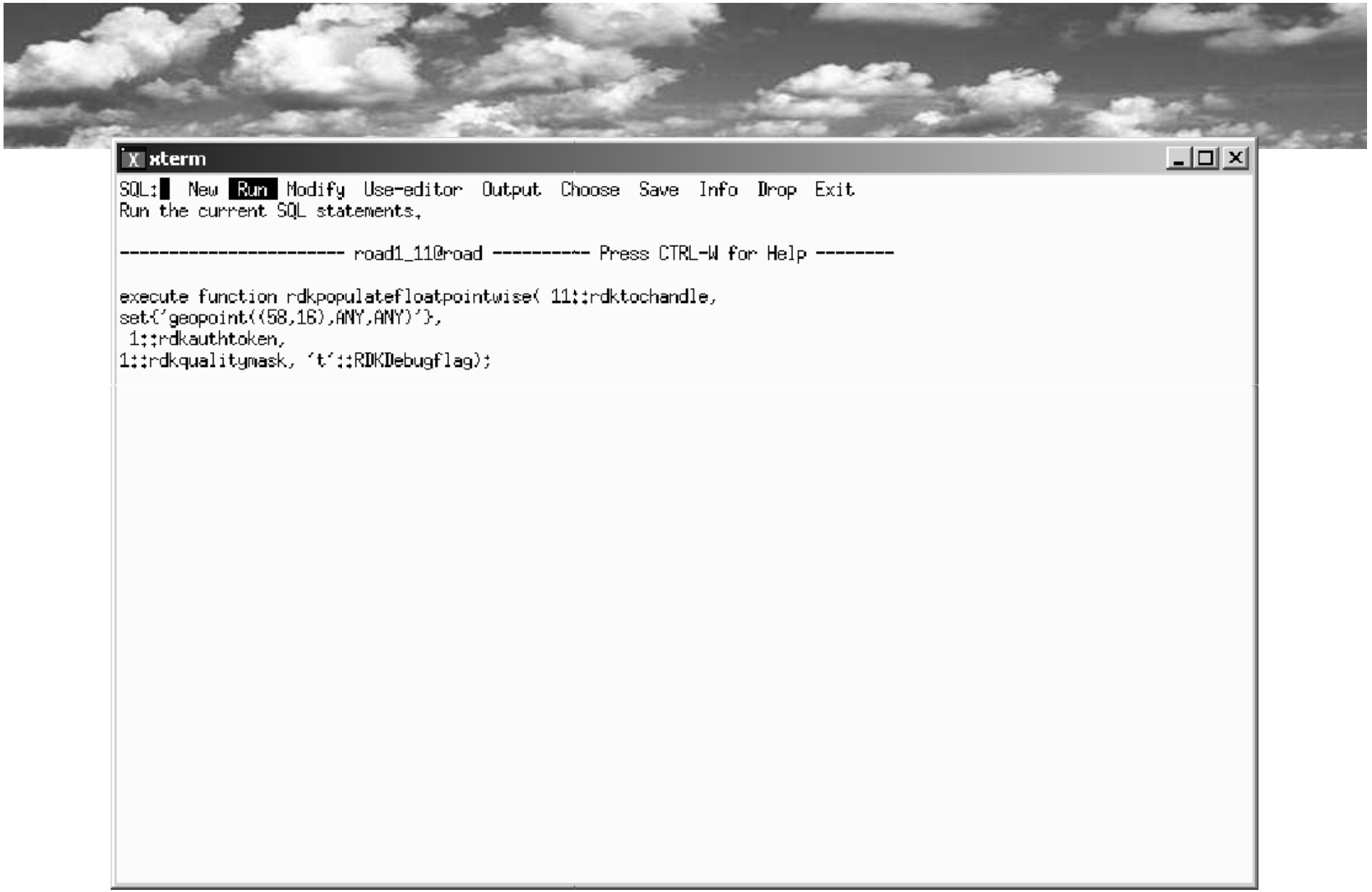

xterm
CHOOSE >>
Choose a command file with the Arrow Keys, or enter a name, then press Return.
---------- road1_11@road ---------- Press CTRL-W for Help --------
democreatetoc
demodelete
demopopulate
dupcheck
equer
fldsrc
numvsrc
orquery
rdkcreatetoc
rdkdeleteToc
rdkdeletetoc1
rdkdeletetoc2
rdkpopulatefloatp+
rdktest
routypeFun
satsrc
savegrib
tmp
tmpf
tmptab
tmptab2
dupdd
dupdl


xterm
SQL: New Run Modify Use-editor Output Choose Save Info Drop Exit
Run the current SQL statements.
---------- road1_11@road ---------- Press CTRL-W for Help --------
execute function rdkcreatetoc(set {10::RDKIbType},
set {1000::RDKSource},
set {10011::RDKParameter},
set{'geopoint((60.0,14.0), ANY,(1978-01-01 00:00
:00.0, 2001-12-12 00:00:00.0))'},
"1978-01-01 00:00:00."::RDKRefTimeBegin,
"2001-12-31 00:00:00."::RDKRefTimeEnd,
"MATS"::RDKAuthtoken,
1::RDKVersion,
1::RDKUserId,
't'::RDKDebugFlag);


xterm
SQL: New Run Modify Use-editor Output Choose Save Info Drop Exit
Run the current SQL statements.
---------- road1_11@road ---------- Press CTRL-W for Help --------
execute function rdkcreatetoc(set {10::RDKIbType},
set {1000::RDKSource},
set {10011::RDKParameter},
set{'geopoint((60.0,14.0), ANY,(1978-01-01 00:00
:00.0, 2001-12-12 00:00:00.0))'},
"1978-01-01 00:00:00."::RDKRefTimeBegin,
"2001-12-31 00:00:00."::RDKRefTimeEnd,
"MATS"::RDKAuthtoken,
1::RDKVersion,
1::RDKUserId,
't'::RDKDebugFlag);
Running . . .


xterm
SQL: New Run Modify Use-editor Output Choose Save Info Drop Exit
Run the current SQL statements.
---------- road1_11@road ---------- Press CTRL-W for Help --------
(expression) 11
1 row(s) retrieved.


xterm
SQL: New Run Modify Use-editor Output Choose Save Info Drop Exit
Modify the current SQL statements using the SQL editor.
---------- road1_11@road ---------- Press CTRL-W for Help --------
execute function rdkpopulatefloatpointwise(144::rdktochandle,
set{'geopoint((58,16),ANY,ANY)'},
1::rdkauthtoken,
1::rdkqualitymask, 't'::RDKDebugflag);


xterm
SQL: New Run Modify Use-editor Output Choose Save Info Drop Exit
Run the current SQL statements.
---------- road1_11@road ---------- Press CTRL-W for Help --------
execute function rdkpopulatefloatpointwise(11::rdktochandle,
set('geopoint((58,16),ANY,ANY)'),
1::rdkauthtoken,
1::rdkqualitymask, 't'::RDKDebugflag);

xterm

```
DISPLAY:  Next  Restart  Exit
Display next page of results.

----------------------- road1_2@road ----------- Press CTRL-W for Help --------


parameter       10002
source          13682
geo             GeoPoint((40.55,65.683333),(0,0),(1978-12-11 06:00:00,1978-12-1
                1 06:00:00))
qualityflag     1026
levelparameter  0
levelinfo       700
value           102940.0000000
ibtypeid        1
storetime       1978-12-11 12:06:04
altitude        0.00
```

XML

```
<?xml version="1.0"  encoding="ISO-8859-1"?>
<!DOCTYPE PRODUCT SYSTEM "http://www-media.smhi.se/SMHI/smhi-
product.dtd" >

<PRODUCT PRODUCTID="99" DESCRIPTION="Tvådygnsprognos till SMHI-
kund" CUSTOMER="Företaget AB" GENERATOR="BESTWEATHER">

   <USER_DEF>2000-06-07</USER_DEF>

   <PRODUCTELEMENT TYPE="Tvådygnsprognos" ELEMENTID="1">
      <PARAMETER NR="1000022" CONTENT-DESCRIPTION="Place name"
         CONTENT-TYPE="TEXT/PLAIN; charset=ISO-8859-1"
         UNIT="String">Stockholm</PARAMETER>
      <PARAMETER NR="1000115" CONTENT-DESCRIPTION="Ortskategori"
         CONTENT-TYPE="TEXT/PLAIN; charset=ISO-8859-1"
         UNIT="String">storstad</PARAMETER>
      <PARAMETER NR="1000110" CONTENT-DESCRIPTION="GällandeTid"
         CONTENT-TYPE="TEXT/PLAIN; charset=ISO-8859-1"
         UNIT="Date(UTC)">2000-06-07 12:00

         <PARAMETER NR="1000101" CONTENT-DESCRIPTION="Vindhast"
            CONTENT-TYPE="TEXT/PLAIN; charset=ISO-8859-1"
            UNIT="m/s">2</PARAMETER>
         <PARAMETER NR="1000102" CONTENT-DESCRIPTION="Vindriktn"
            CONTENT-TYPE="TEXT/PLAIN; charset=ISO-8859-1"
            UNIT="String">W</PARAMETER>
         <PARAMETER NR="1000103" CONTENT-DESCRIPTION="Temp_C"
            CONTENT-TYPE="TEXT/PLAIN; charset=ISO-8859-1"
            UNIT="Celsius">17</PARAMETER>
         <PARAMETER NR="1000104" CONTENT-DESCRIPTION="Temp_F"
            CONTENT-TYPE="TEXT/PLAIN; charset=ISO-8859-1"
            UNIT="Fahrenheit">63</PARAMETER>
         <PARAMETER NR="1000105" CONTENT-DESCRIPTION="Vädersymbol"
            CONTENT-TYPE="TEXT/PLAIN; charset=ISO-8859-1"
            UNIT="String">4</PARAMETER>
      </PARAMETER>

      <PARAMETER NR="1000110" CONTENT-DESCRIPTION="GällandeTid"
         CONTENT-TYPE="TEXT/PLAIN; charset=ISO-8859-1"
         UNIT="Date(UTC)">2000-06-08 12:00
               .                                  .
               .                                  .
               .                                  .
      </PARAMETER>
   </PRODUCTELEMENT>

   <PRODUCTELEMENT TYPE="Tvådygnsprognos" ELEMENTID="2">
               .                                  .
               .                                  .
               .                                  .
   </PRODUCTELEMENT>

</PRODUCT>
```

1.

2.

Figur 2. Exempel på XML-fil baserad på SMHIs generella DTD-fil

Hardware

- Production server
  - Sun E3000 with 6 CPUs (1 GB/250 MHz/1996)
  - Solaris 2.6 (moving to Solaris8 soon)
  - Dual A5000 Diskarray
- Production test server
  - Sun E450R with 4 CPUs (2GB/450 MHz)
  - Solaris 2.6 (moving to Solaris8 soon)
  - T3 Diskarray (RAID5) with 512 MB battery-backup diskcache

Experience SCALABILITY

- What is scalability problem?
  - You add CPUs and disks/controller but throughput does not increase
  - You have spare capacity (CPU/Disk) and you increase the load but the utilisation does increase (something serialises)
- 9.20 on E4500 did not scale (iops-bound?)
- 9.21 scalability worse than 9.20 (more mutexes)
- Most datablades scale linearly
  - Memory allocation (mi_alloc) is expensive and requires mutex -> scalability problems

PLUS MINUS

Minus

- **IDS issues**
  - **B-tree cleaning problems with skewed data distributions**
  - **Datablades brings you back to printf debugging**
  - **Complex memory allocation**
  - **Support do not understand...**
  - **Full SMP exploitation is hard: mi_alloc requires mutex (serialises fast udrs)**
  - **Rather high threshold >1 month to be productive**
  - **Extensive testing required to maintain engine stability**
  - **No profiling of performance**
  - **Locked into IBM IDS. Similar technology only exists in PostgreSQL, WS-Iris, AMOS.**

Minus

- **Bladesmith issues**
  - **DBDK single developers environment**
  - **Careful planning necessary to avoid collisions**
  - **NT-only tool for auto-generation of datablade code (although generated code can be moved to other environments)**
  - **Functions with multiple results not supported by Bladesmith**

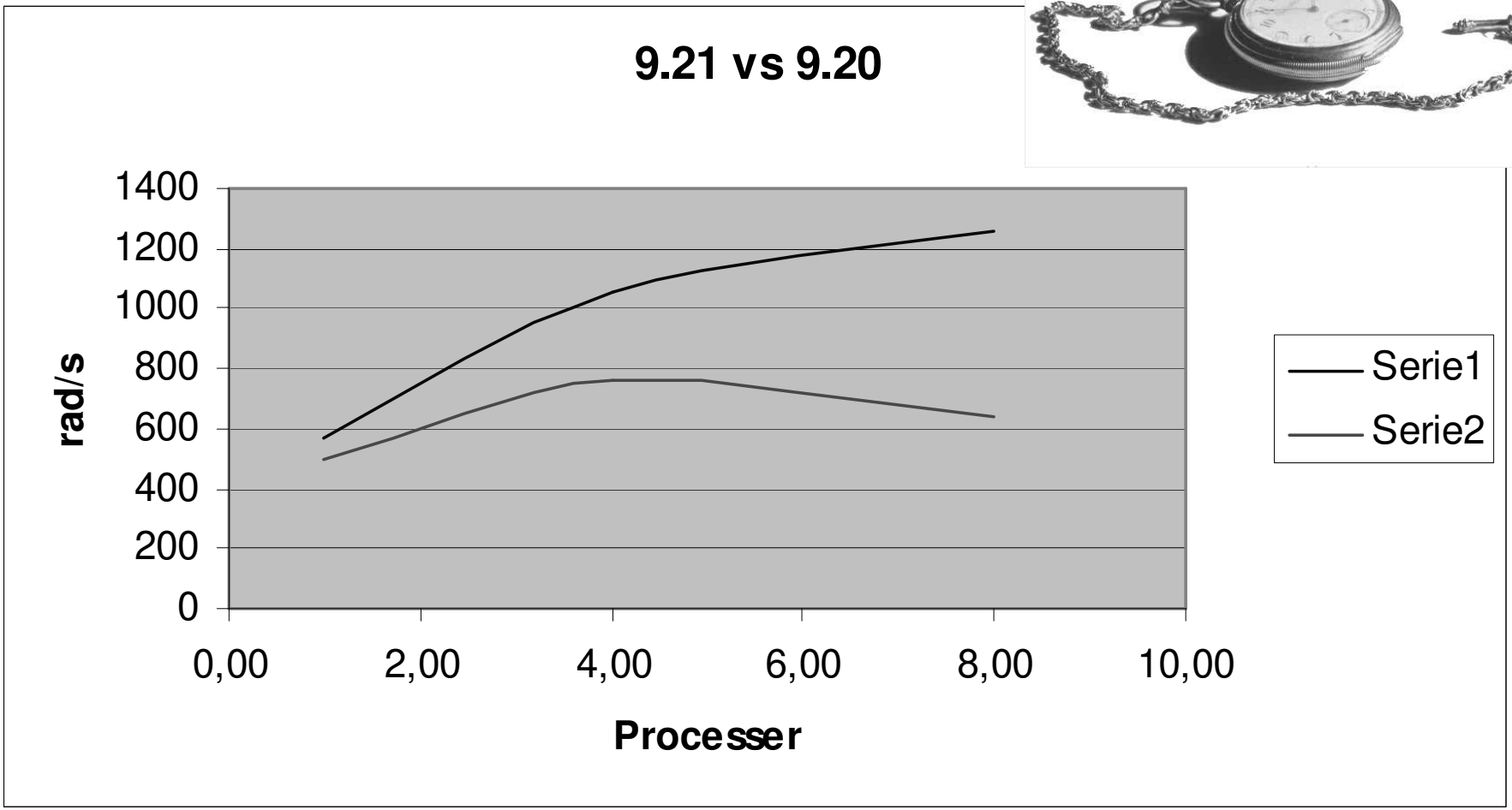
Minus

- **IDS issues**
  - **SDK not threadsafe in Solaris (is threadsafe in NT4!!)**
  - **Collection iterator in server crashes after 11 retone**
  - **Limit of 1000 grants**
  - **Multiset limit 32k is limiting**
  - **Client-side mem leak ifx_var_flag(&binP,0); Ifx_var_alloc(&binP,sizeof.. Ifx_var_dealloc(&binP);**
  - **Fix? Free(binP) which is an nullpointer frees memory…**
  - **R-tree not stable...**

Minus

- **BUG/FEATURE DANCE**
  - **Que? What is a datablade?**
  - **It's a bug**
  - **It's a feature**
  - **It's a bug**
  - **It's a feature**
  - **Ohh…. I get it… It's a bug**
  - **No… It's a feature**
  - **It's a bug**
  - **It's a feature**
  - **Ahaa… It's a bug**
  - **Sorry too hard to fix**
  - **We have a workaround for you**

Insert scalability



Datablade Benefits

- Simple
  - Use standard SQL DB-APIs
  - Use standard SQL tools
  - Ensures data integrity
  - Share central business logic
  - Implement once, use everywhere
  - Improved portability of apps
- Improves performance
  - Reduces client-server I/O
  - Reduces internal processing
  - Function shipping
- 7/24
  - Runtime deployment
  - No need to recompile clients
- Free services
  - Multithreading,transactions, backup/restore, etc.

Benefits IDS

- Performance Insertions
  - 1000000 floats inserted/s (86 transactions per second)
  - Not bulk updates!
  - 1600 rows inserted per second
  - Outperforms geriatric dedicated solution based on files and specific Fortran APIs
- Performance I/O
  - 90 MB per second
  - IOPS-bound
  - Faster than 100 Mbit network
  - Twice as fast as filesystem
- Performance Retrieval
  - 500 rows retrieved per second
  - 150 queries per second

Conclusion

- Operational since 1999
- IBM IDS 9.21UC3 very stable and very good performance with our datablades.
- Good support from Development team, Informix Sweden (especially Rickard), Advanced Technology Group, Geodetic (Robert Uleman)
- Improved UK-support after IBM acquisition

Future trends

- Database systems provide a fixed set of services. The services has been carefully selected to provide adequate functionality for target users. There are always applications where the DBMS does not provide adequate functionality.
- There are two remedies for this: extend inside or simulate with a wrapper. Much better performance can be achieved if extension is made inside the engine.
- If the DBMS can be tailored for the application the complexity is ultimately reduced. Complex data types become natural. Complex access patterns become easier to handle.
- Performance is crucial. Engineers are always trying to cut cycle times. A major villain is communication cost. Datablade technology allows you to reduce communication costs and hence improve performance.

Inspiration technology

- Datablades are inspiration technology
  - Elegance, Modern sw architecture
  - Performance increase when operating near data
  - Logic in server improves adaptability
  - Encapsulates domain-specific knowledge
- Application are different but..
- I hope you have been inspired...
- Mission impossible only takes a bit longer

Resources

- **Object-Relational Datablade Development**
  - **A Plumbers Guide (by Paul Brown)**
    ISBN 0130194603
  - **Extending IDS2000 (Informix manual)**
  - **Datablade API (Informix manual)**
  - **Database Technology for Control and Simulation (PhD thesis by Esa Falkenroth)**

CONCLUSIONS

- Database technology simplifies development and maintenance of data-intensive applications
- Use database systems when:
  - data volumes are large
  - data have complex inherent structure
  - flexibility is needed (structure and access patterns)
  - concurrent access from several users/appl
  - data are valuable
- Economy of scale: More information in the database increases its value

Commercial DBMS

- **Oracle 9i <http://www.oracle.com>**
- **IBM DB2 <http://www.ibm.com>**
- **Informix IDS2000 <http://www.ibm.com>**
- **Sybase Adaptive Server <http://www.sybase.com>**
- **Microsoft Access (not for large data volumes)**

FREE LINUX DBMS

- **SAPDB http://www.sap.com/**
  Internal DBMS of SAP erp-software (GPL)
- **PostgreSQL <http://www.postgresql.org/>**
  Pioneer object-relational database system (GPL)
- **MySQL <http://www.mysql.com>**
  Originally lightweight webdb. No transactions in early versions (GPL)
- **Many more at <http://linas.org/linux/db.html>**

FURTHER DB-READING

- **Fundamentals of Database Systems (Elmasri/Navathe)**
- **An Introduction to Database Systems (Date)**
- **Climate and Environmental Database Systems (Lautenschlager and Reinke eds.)**

EXJOBB and Project employment

- SMHI has many opportunities for exjobb and project employment.
- Past and ongoing exjobb in meta-data representation and harvesting
- Contact us for master thesis work (exjobb)
- Contact us for hints on research problems in database systems

THANK YOU !
Dr Falkenroth
SMHI