Efficient Data Dissemination through a Storageless Web Database

Thomas Hammel

prepared for DARPA SensIT Workshop

19 April 2001

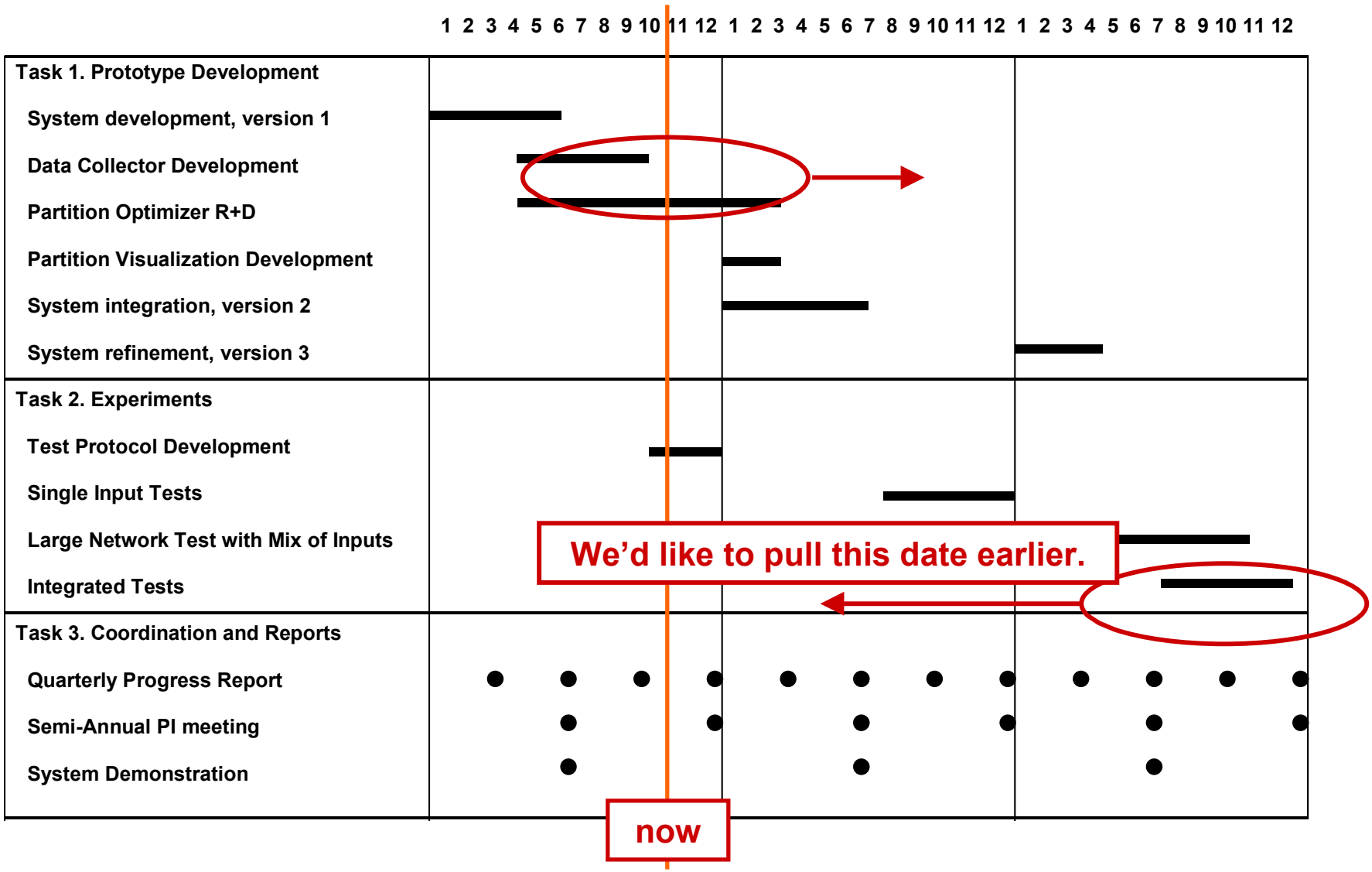
Fantastic Data

207 Prospect Avenue, San Francisco, California 94110, USA • (415) 643-9555 • (415) 643-9554 • www.fandat.com

Web Database System

- **Automatically establish redundant data caches throughout the network based on**
  - **data usage patterns, transactions and queries**
  - **optimize cost function based on power consumption, latency, and survivability**
  - **no permanent storage**
- **Disseminate data and maintain redundant caches**
  - **reliable delivery on top of an unreliable channel**
  - **retries mitigated by**
    - **data expiration**
    - **obsolescence detection**
    - **priority**
  - **supports dynamic filter changes**
  - **cooperative repair**

Schedule



Fantastic Data

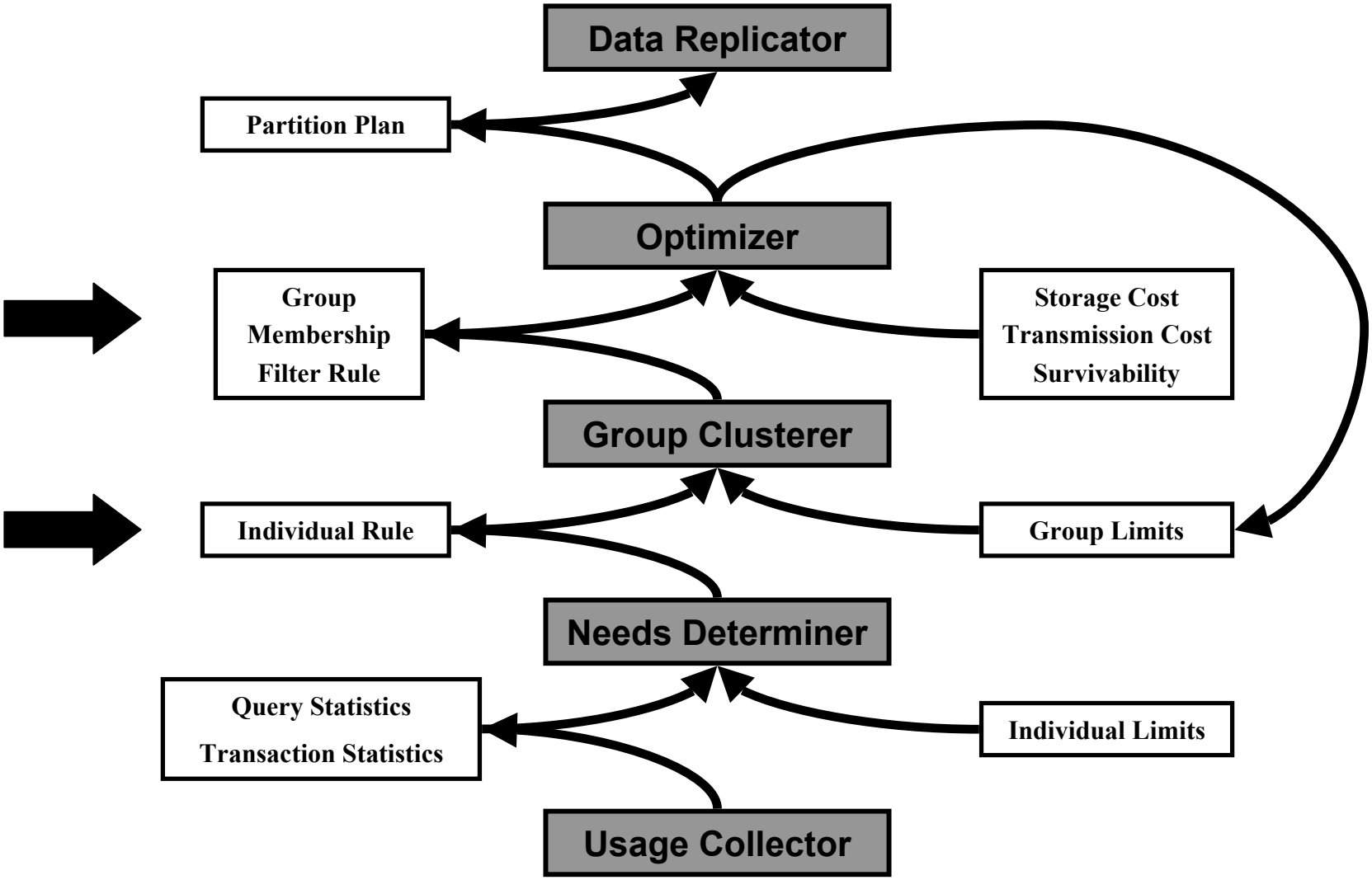
Major focus of last period

- Preparing to support experiments on the SensIT nodes
- Implementation of the in-memory cache
  - higher speed, smaller code size
  - allows easy creation of special data types, analysis functions
  - better integration of consistency bookkeeping fields with regular data fields
  - implemented distributed table creation
- Establish API
  - reduce complexity
    - eliminate features we don't really want to support
  - streamline the socket interface

Determining dissemination filters

- **Persistent queries from application programs are used to determine dissemination filters**
  - **easy to “or” each application filter together**
  - **want to reduce composite condition complexity**
  - **dissemination filters are evaluated on every record change**
  - **reducing complexity is good, maybe even if it increases the data load**
    - **(a>=25 & a<=45) | (a>=30 & a<=50) is clearly equivalent to (a>=25 and a<=50), but what about**
    - **(power>=3.0 & lat>=39.345678 & lon>=-120.342780 & lat<=39.346001 & lon<=-120.342678 & _t>=900000000.0) | (power>=2.7 & lat>=39.345670 & lon>=-120.342778 & lat<=39.346000 & lon<=-120.342676)**
  - **clustering code has been developed, needs to be integrated**
- **Need to factor in 1-time queries**
  - **how often are they done?**
  - **how closely do they match the persistent queries?**
  - **how large is the remote load required to satisfy the query?**

Partition Determination



Fantastic Data

207 Prospect Avenue, San Francisco, California 94110, USA • (415) 643-9555 • (415) 643-9554 • www.fandat.com

Very near term plans

- **Finish version 1**
  - **add single precision float data type**
  - **add some functions in where clause evaluator**
    - **min, max, mean, abs, sin, cos, tan, atan, exp, log, sqrt, …**
  - **descriptive error codes**
  - **testing and packaging**
- **Need to select a distribution method that can be easily used by others and can simulate need for multihop**
  - **suggest broadcast to different UDP ports**
    - **each server broadcasts to a particular port**
    - **each server reads from whichever set of ports represent its neighbors**
- **Deliver to BBN and BAE about June 2001**
  - **Wider distribution later**

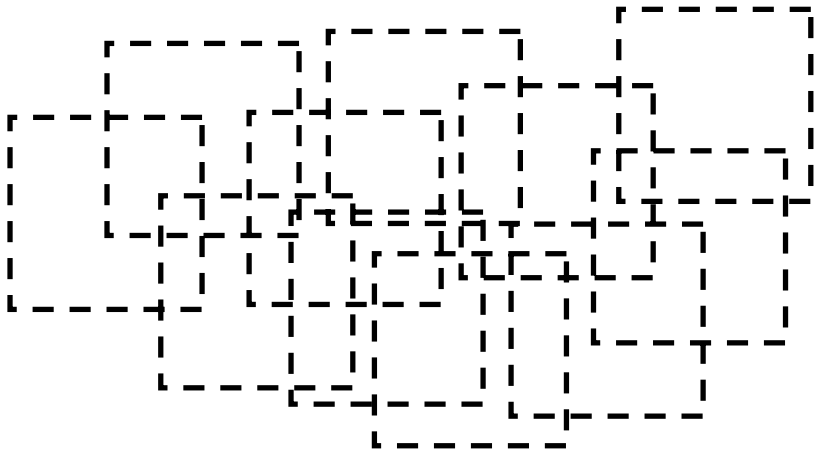
Near term plans (next 6 months)

- **Support use of system**
  - **either in operational demo, or**
  - **in field/lab demos as appropriate**
- **Integrate dynamic filter support into production version**
  - **code exists in larger, slower disk-resident version**
- **Improve filtering performance**

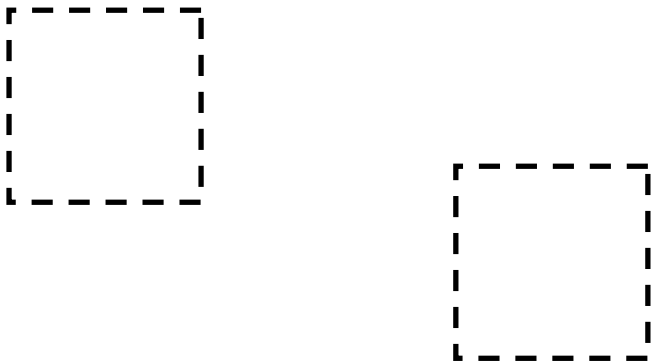
Longer term plan (6 months +)

- **Support additional users**
- **Implement configuration options**
  - **data criticality**
  - **latency requirements**
  - **excess data holdback (don't need it more frequently than ...)**
- **Investigate relationship of Fantastic Data caches to ISI routing**
  - **Should we place a filtering module inside the routing layer?**
  - **What are the similarities/differences between our filtering approach and ISI's.**

2 potentially different filtering problems

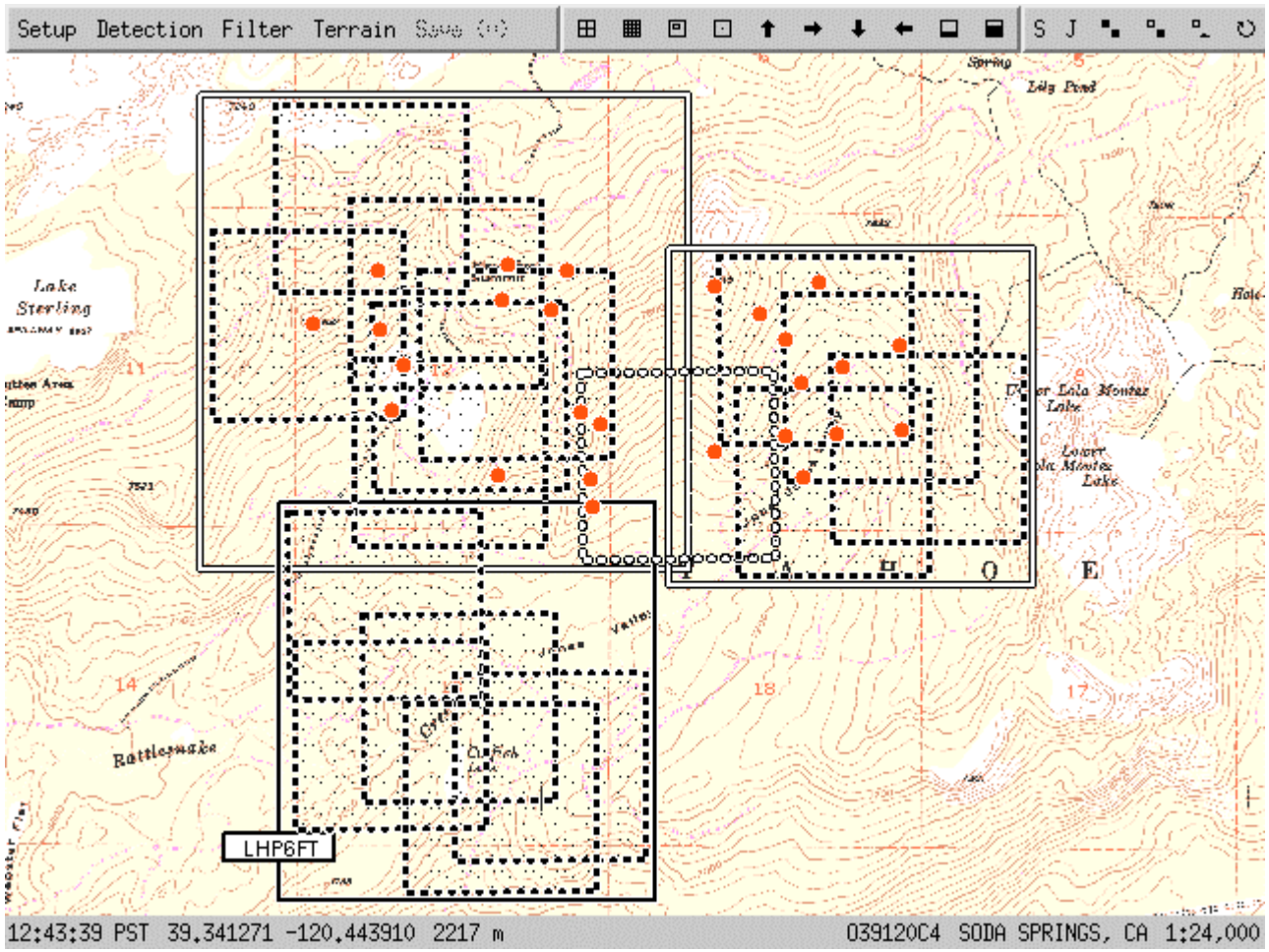


- Dense, connected interests
- Well served by local broadcast
  - flooding can be mitigated by knowledge of link state
- Data dissemination decision made by knowledge of neighbors' interest
  - can be approximated by own
- Is this the results formation problem?



- Sparse, disjoint interests
- Routing required
- Data moves across network through many uninterested nodes
- Is this the results extraction problem?

Clustering



Fantastic Data

207 Prospect Avenue, San Francisco, California 94110, USA • (415) 643-9555 • (415) 643-9554 • www.fandat.com

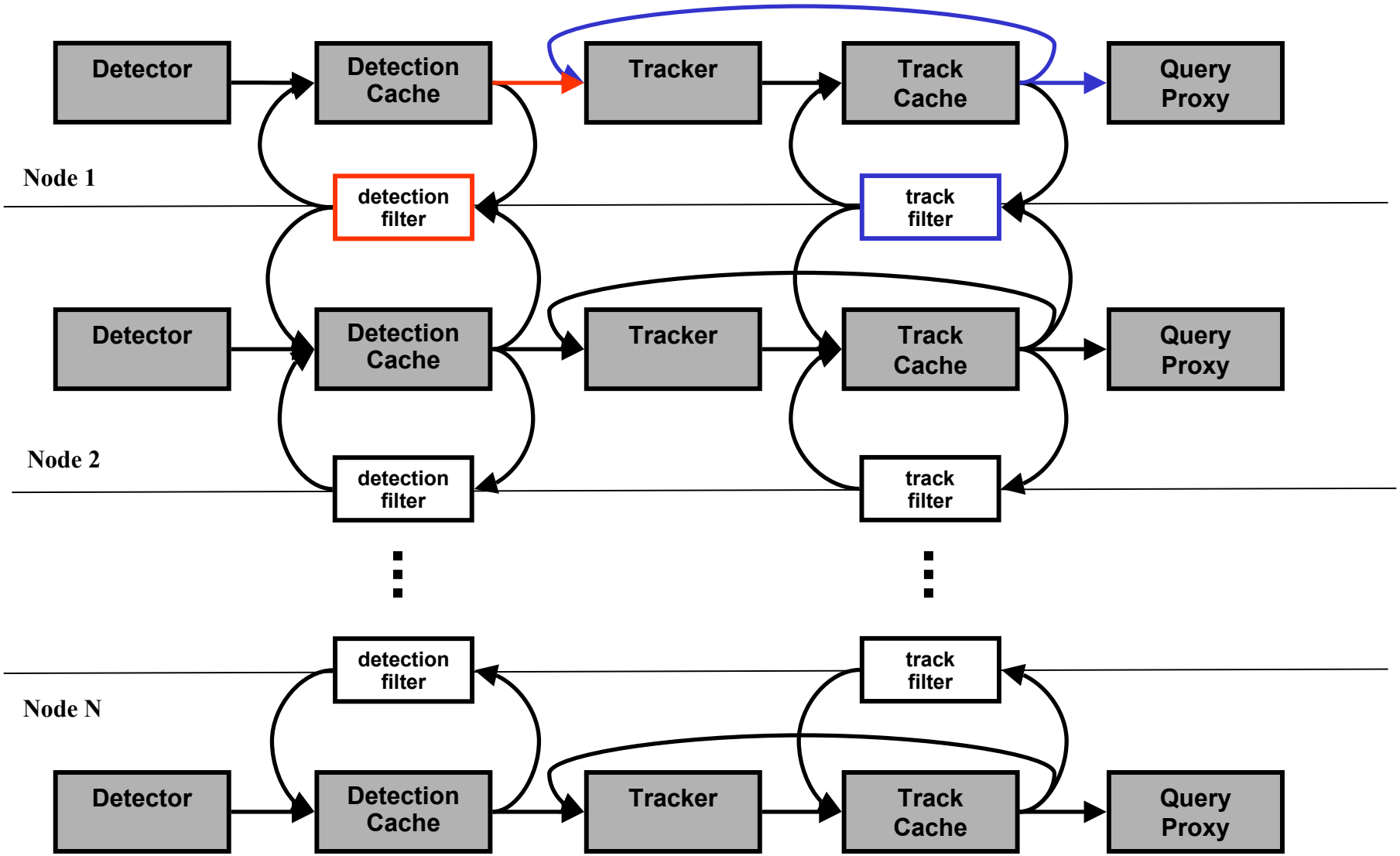
Clustering philosophy

- **locally determined**
  - **not globally optimized**
  - **minimize interaction between nodes required to setup filters**
- **incremental**
  - **try to disturb existing situation as little as possible**
- **filter tolerance**
  - **a little too big, a little too small, that's ok**
- **maintain cluster quality information**
  - **mean coverage of individual needs (percent, record count, bandwidth)**
  - **excess coverage (percent, record count, bandwidth)**
  - **number of members in group**
  - **mean age of member's input data (seconds)**

Automatic clustering rules

- **Individual rules**
  - **specific value, integer or character**
    - **mode='acoustic'**
    - **type='tank'**
    - **code=347**
  - **range of values (allow 1 side to be unbounded), integer or float**
    - **value>=3 and value<=14**
    - **snr>=3.5**
    - **power>=0.0 and power<=10.0**
  - **area, integer or float**
    - **latitude>=39.342893 and latitude<=39.358214 and longitude>=-120.451740 and longitude<=-120.430266**
- **Combination rules**
  - **"and" and "or"**
    - **mode='acoustic' and snr>3.5 and (latitude>=39.342893 and latitude<=39.358214 and longitude>=-120.451740 and longitude<=-120.430266)**

Detection and Tracking Example



Interface options

- **Communicate over your own socket**
  - **open connection to TCP localhost:TBD or local domain /tmp/.fdb**
  - **send requests**
    - **[id] [statement];**
    - **where [id] is a unique integer and [statement] is any valid statement**
    - **new line characters may be embedded in the statement**
    - **it must end with a semi-colon followed by a new line character**
  - **cache responds with**
    - **[id] [status] [data]**
- **Use our C library functions**
  - **library creates and manages the socket communications**
  - **library provides callback functions when data arrives**
  - **library provides an event loop or you can use your own**
    - **call library function when there is activity on the socket**

Cache: How to use

- Create the tables you need
  - another application or another node may have already done it
  - but doing it twice does no harm
    - unless you are first and you get it wrong
- Perform queries for configuration information
  - for example, to find out the capabilities of the node
- Perform watch operations to monitor the data you need
  - data is provided as it changes
    - you get the data before the other application gets confirmation
  - watch operations are the primary determinant of distribution filters
- Sit back and relax
  - data will come to you
- Wake up
  - process the incoming data
  - create some new data of your own

API: Communicating with the server

```
/*
 * Connect to the server. This is automatically done by CachePerform()
 * if you don't do it first. If a function is provided it is called
 * whenever the connection status changes.
 *
 * If socket>=0, the connection is up and this is the socket number.
 * You may use this in select() or poll().
 * If socket<0, the connection is down, and this is an error code.
 *
 * If the connection should fail, it is automatically restored and
 * any operations in progress are automatically resubmitted unless
 * CacheDisconnect() is called.
 */
extern int CacheConnect(void (*function)(int socket));


/*
 * Disconnect from the server. If a function was provided to CacheConnect(),
 * it is called when the connnection status changes.
 */
extern void CacheDisconnect();
```

API: performing an operation

```
/*
 * Ask to have an operation performed. Automatically connects
 * to the server if that has not already been done.
 *
 * Returns code assigned to this operation. Code is a positive integer.
 * A negative return indicates an error.
 *
 * Callback function is called once for each row (status>0), error (status<0),
 * and when done (status==0).
 */
extern int CachePerform(char *statement,
    void (*function)(void *ptr, void *arg), void *arg);
```

API: Statements that maintain tables

```
/*
 * Statements may be any of the following:
 *    CREATE TABLE table (field type, ... , field type, PRIMARY(field list));
 *        Create a new table. The primary key is required.
 *    DROP TABLE table;
 *        Delete a table and all of its contents.
   DESCRIBE TABLE table;
 *        Return the specification of a table.
 */
```

API: Statements that change data

```
/*
 * Statements may be any of the following:
 *    INSERT INTO table field list VALUES value list;
 *        Insert a row into the table. The row must not exist.
 *    PUT INTO table field list VALUES value list;
 *        Insert a row into the table. If the row already exists, this
 *        operation is treated as the equivalent update.
 *    UPDATE table SET field list = value list WHERE condition;
 *        Update all existing rows of the table that meet the condition.
 *    DELETE FROM table WHERE condition;
 *        Delete all existing rows that meet the condition. Caution: if no
 *        condition is specified, deletes all rows of the table.
 *    UNDELETE FROM table WHERE condition;
 *        Restore all deleted rows that meet the condition.
 *    PURGE FROM table WHERE condition;
 *        Permanently removes all rows of the table that meet the condition.
 *        Caution: The presence of deleted data is necessary to ensure
 *        consistency of the redundant caches. Data should be purged only
 *        if some other mechanism makes sure that all nodes purge the same
 *        data at the same time. For example, a data expiration feature
 *        based on time of day and time of data creation is safe.
 */
```

API: Statements that query data

```
/*
 * Statements may be any of the following:
 *    SELECT field list FROM table WHERE condition;
 *        Return the rows of the table that match the condition.
   WATCH field list FROM table WHERE condition;
 *        Return the rows of the table that match the condition as they are
 *        inserted, updated, or deleted.
   SELECT AND WATCH field list FROM table WHERE condition;
 *        Do both select and watch operations. Guaranteed to return all of
 *        the existing rows before any changes and to not miss any changes.
   CANCEL operation;
 *        Cancel a previously specified operation, for example, an
 *        ongoing watch operation.
 */
```

API: Interpreting the results

```
/*
 * Returns the operation code associated with the given result pointer.
 * Returns a negative error code if something is wrong.
 */
extern int CacheResultCode(void *ptr);


/*
 * Returns the status associated with the given result pointer.
 * See enum CacheStatus. 0 means the operation is done.
 * A positive status means that there is valid data.
 * A negative return is an error.
 */
extern int CacheResultStatus(void *ptr);


/*
 * Return the number of rows changed or returned by the operation.
 * A negative return is an error code.
 */
extern int CacheResultImpact(void *ptr);
```

API: Interpreting the results

```
/*
 * Returns a pointer to the data associated with the result pointer.
 * If the result status is positive, this is the row data in text form.
 * If the result status is negative, this is an error message.
 * Returns 0 if there is something wrong.
 */
extern char *CacheResultMessage(void *ptr);

/*
 * Returns the number of fields in the result. Valid only if the
 * associated operation was a query.
 *
 * A negative return is an error code.
 */
extern int CacheResultNvalue(void *ptr);

/*
 * Returns the value of the specified field in the specified result
 * in text form.
 * Returns 0 if there is something wrong.
 */
extern char *CacheResultValue(void *ptr, int it);
```

API: data field types

```
/*
 * Data types include:
 *    integer (32 bits),
 *    short (16 bits),
 *    byte (8 bits),
 *    char (variable length),
 *    double (64 bits),
 *    blob (variable length, uninterpreted binary data).
 */
```

API: where clauses

```
/*
 * Conditions may be composed of any field, constants, and the operators
 *
 *     +, -, *, /, &, |, !, ^, * <, <=, =, >=, >, !=.
 *
 * The absence of a WHERE clause is interpreted as whatever portion of
 * the table is maintained on the local node. This is not the same as
 * all data in the table throughout the system.
 *
 * Where clauses on queries (especially watch queries) are the major
 * determinant of data distribution filters.
 */
```

API: special data fields

```
/*
 * All tables are automatically supplied with several special fields. The
 * values of these fields may not be set with insert, update, or put
 * statements, but they may be returned by queries and used in conditions.
 * The special fields are the following:
 *    _b integer, the node that changed the record.
 *    _s integer, the series of the record change.
 *    _n integer, the sequence number of the record change.
 *    _t double, the time at which the record change was originated.
 *    _pb integer, the node that previously changed the record.
 *    _ps integer, the series of the previous record change.
 *    _pn integer, the sequence number of the previous record change.
 *    _pt double, the time at which the previous record change was originated.
 *    _lt double, the time at which the record change was performed on
 *        the local node.
 */
```

Common Table Definitions

- **Location of nodes**

```
create table node (id integer, latitude double, longitude double,
  primary(id));
```

- **Capabilities of nodes**

```
create table capability (id integer, type char, …, primary(id));
```

- **Detections**

```
create table detection (id integer, latitude double, longitude double,
  start integer, end integer, cpa integer, power double, primary(id));
```

- **Tracks**

```
create table track (id integer, type char, latitude double, longitude
  double, confidence double, primary(id));
```