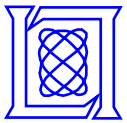


MySQL Users Conference 2007

Using MySQL as Active Database for Monitoring Applications

Jacob Nikom

MIT Lincoln Laboratory

This work was sponsored by the U.S. Government under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

Outline

MySQL Users Conference 2007

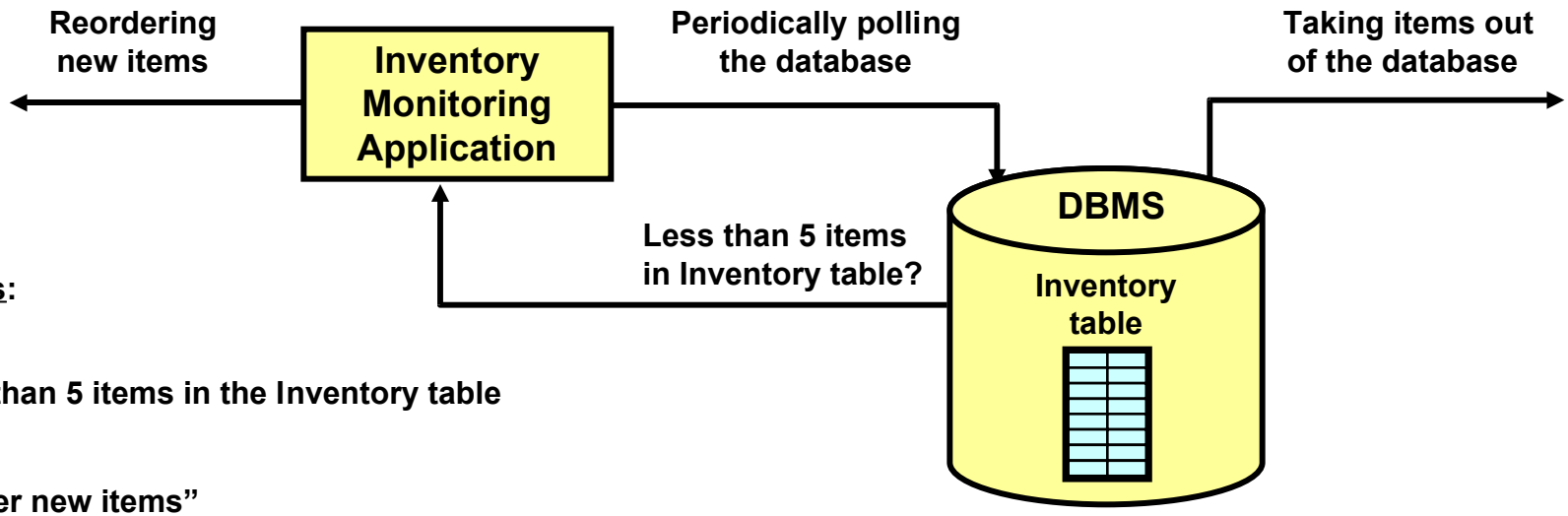
- Introduction
- Preventing Monitoring System Architecture
- Building a Rules Engine for an Active Database
- MySQL as an Active Database
- Summary

Outline

MySQL Users Conference 2007

- Introduction
  - Simple inventory monitoring system
  - Inventory monitoring with Active Database (ADBMS)
  - Railroad tickets inventory monitoring
  - Video monitoring
  - Preventive monitoring
  - Why use ADBMS instead of applications?
- Preventing Monitoring System Architecture
- Building a Rules Engine for ADBMS
- MySQL as ADBMS
- Summary

Simple Inventory Monitoring System



Rules:

If

less than 5 items in the Inventory table

then

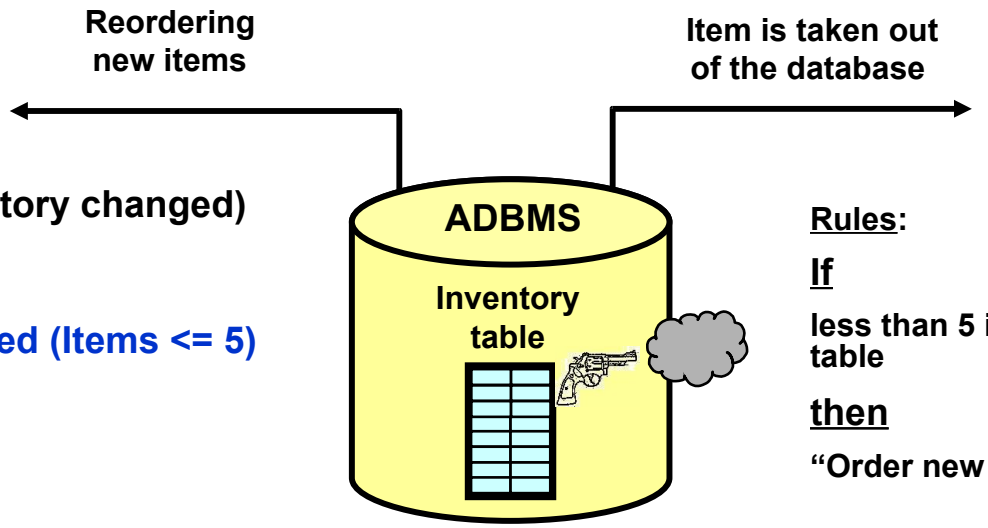
"Order new items"

- Need to poll periodically the database state to detect changes
- High polling rate wastes resources
- Low polling rate reduces responsiveness and accuracy
- Common polling functionality is replicated across many applications
- Difficult to change and maintain features embedded into application code

Monitoring with conventional databases cannot be implemented efficiently!

Cause: conventional DBMS does not know that an application is polling it

Inventory Monitoring with Active DBMS



- If event happened (inventory changed)
- Trigger fires AND
- Condition check happened (Items <= 5)
- If condition is satisfied
- Rules applied AND
- Action executed

- An active database is a database in which some operations are automatically executed once a given situation arises
- An active rule (trigger) is a language construct for defining the system reaction to the situation
- The situation may correspond to the fact that:
  - Some specified events arise
  - Specific conditions or state transitions are detected

Active DBMS Monitoring Architecture Analysis

MySQL Users Conference 2007

Monitoring architecture improvements due to Active DBMS usage

- **Better efficiency**
- **Less components**
- **Better integration with other DBMS features**
- **Better modularity**
  - **Change detection code is isolated from application code**
  - **Uniformity of rules and data interpretation**

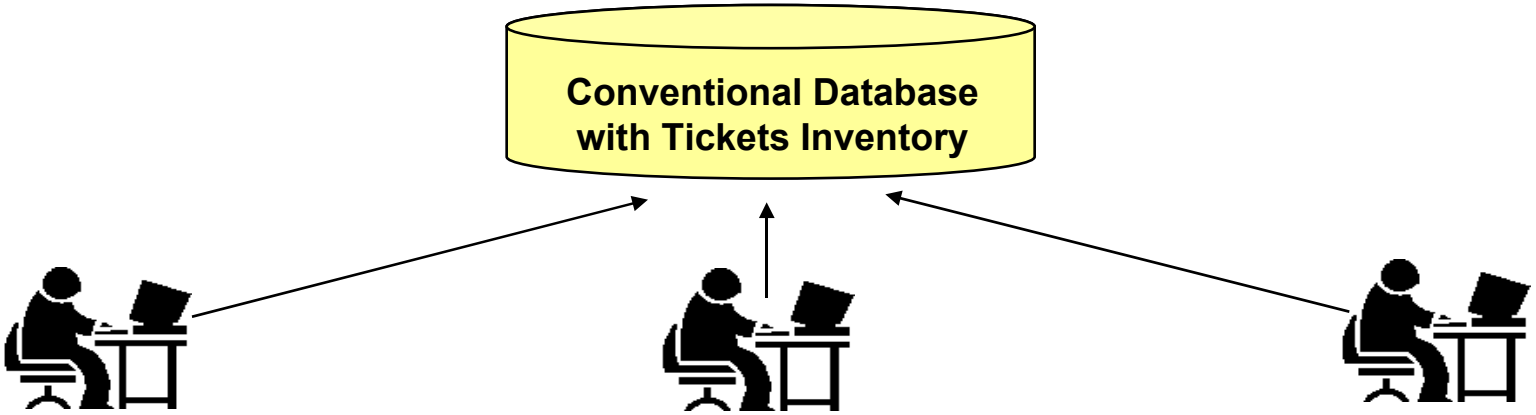
Knowledge about data changes <u>does not</u> belong to application

Consequence of centralization—knowledge sharing

- One application could be aware about the data changes in another one
- Why one application needs to know about changes in another one?

Railroad Tickets Inventory Monitoring

- There are multiple small local travel offices
- Customers are buying tickets to go to city



Travel Agent 1

- Customer 1 wants to buy a ticket for the train between 9 and 10 AM
- The agent 1 checks and returns "No ticket for this time"

Travel Agent 2

- Customer 2 wants to buy a ticket for the train between 9 and 10 AM
- The agent 2 checks and returns "No ticket for this time"

Travel Agent N

- Customer N wants to buy a ticket for the train between 9 and 10 AM
- The agent N checks and returns "No ticket for this time"

One failed request is bad. The group (pattern) of failed requests is really bad!

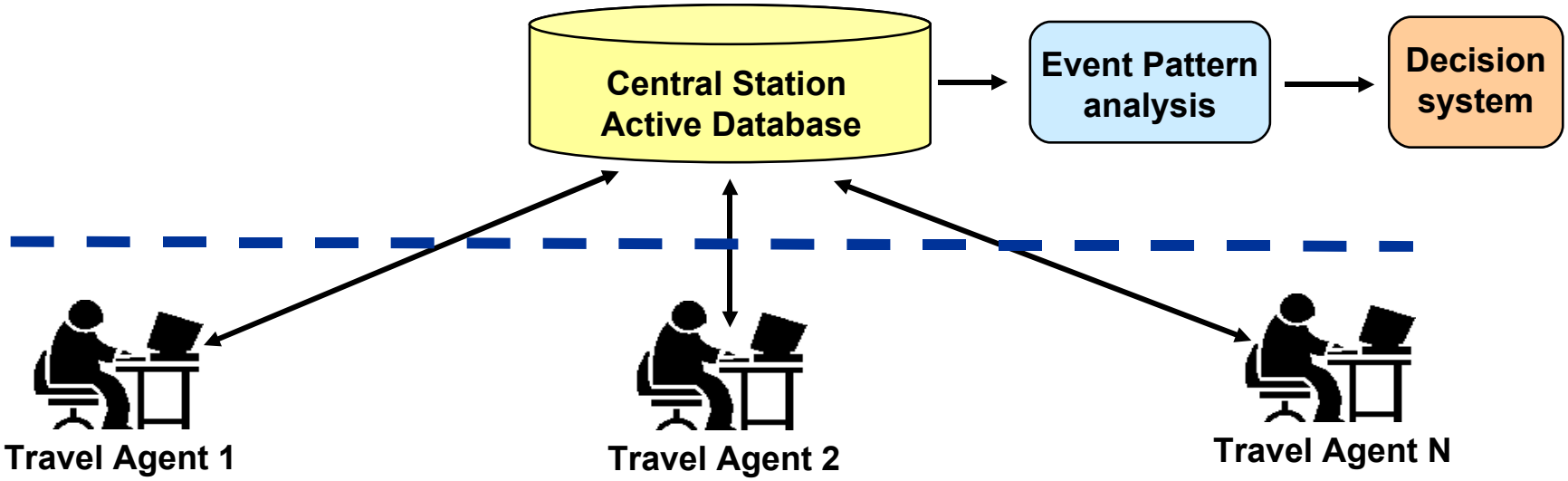
Hey! Maybe we should increase the number of trains available!

Shared knowledge facilitates non-obvious action

ADBMS Railroad Tickets Inventory Monitoring

- Notifies monitoring applications when inventory changes
- Stores monitoring events in the events table
- Starts event pattern analysis when events table changes

Global and local levels of interaction



- Receives the latest ticket inventory state automatically
- Receives the latest ticket inventory state automatically
- Receives the latest ticket inventory state automatically

Video monitoring

- **Video monitoring in the UK**
  - There are 14,000 CCTV cameras in London Underground and 400,000 in London
  - Total number of cameras in the UK is 4,000,000 (14 people per camera)
  - 24 x 7 video monitoring requires a lot of human resources for tape analysis
  - Only an attack justifies the number of people necessary to analyze these tapes
- **Questions**
  - Is video surveillance only useful after an attack?
  - Is it possible to identify an attack before it happens?
  - Does it require tracking all suspicious individuals at all times?

Preventive Video Monitoring with ADBMS

MySQL Users Conference 2007

- **Principles**
  - Don't track individuals, track the activity
  - Activity is an ordered sequence of events
  - Suspicious activity is made up of seemingly unsuspicious events
  - Only the relations associate those events with particular activity
- **Design**
  - Suspicious activities are well known in advance
  - Usually all events that make up the suspicious activity are known
  - Participation of the individual in one event is not bad
  - Participation of the individual in the suspicious sequence of events is bad
- **Implementation**
  - New events are stored in the ADBMS
  - Each event insertion triggers ADBMS to start-up the Rules Engine to scan all events in search of a pattern
  - Rules Engine notifies decision system about patterns found

Why Use ADBMS Instead of Applications?

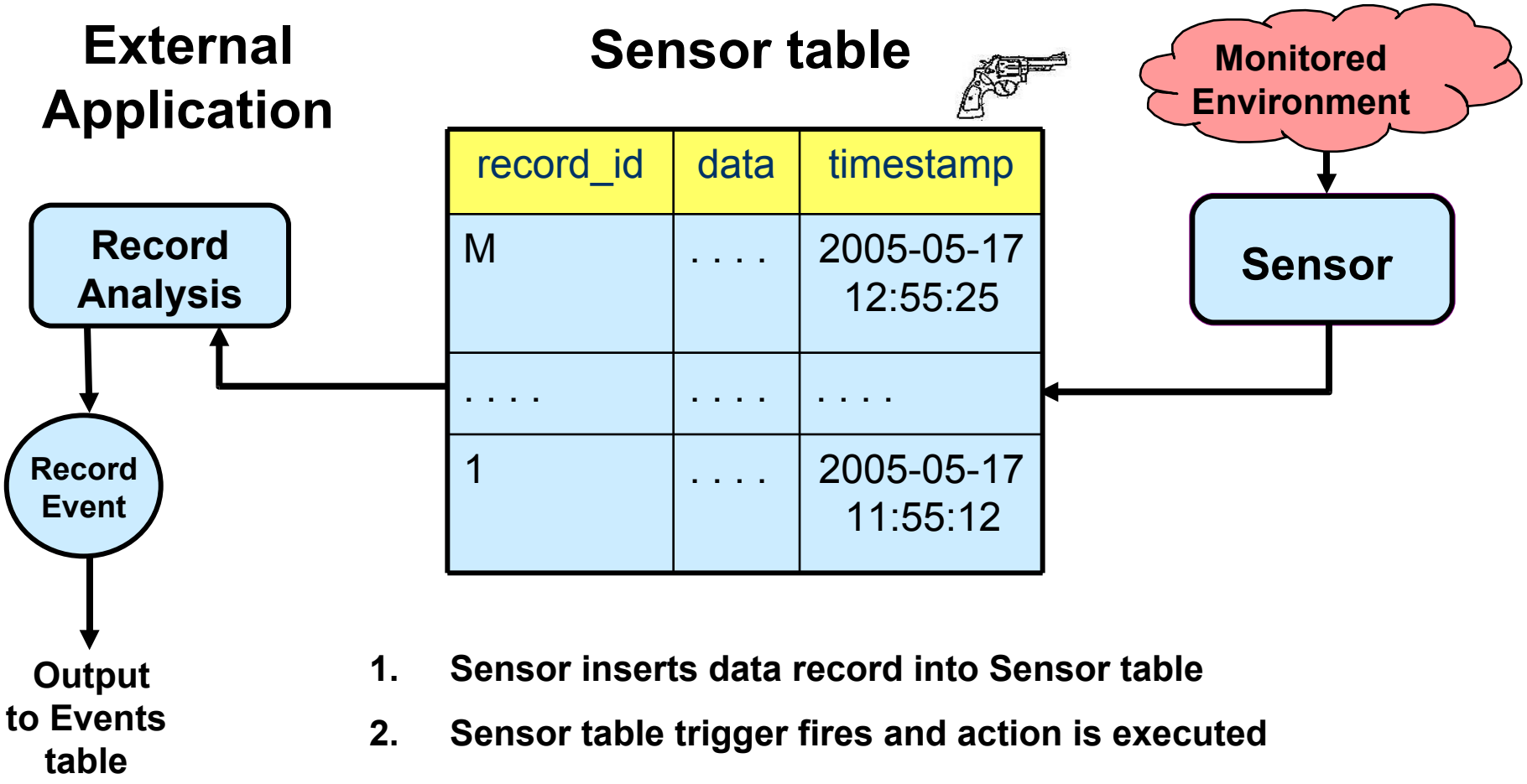
MySQL Users Conference 2007

- Ready available powerful, reliable and flexible ADBMS framework with little need for additional programming
- Easy shared knowledge between multiple applications
- One implementation enforces uniform, consistent behaviour for all monitoring applications
- Monitoring events could be described with standard SQL
- ADBMS has full and quick access to all data on the server
- ADBMS has full access to functions and store procedures

Outline

- Introduction
- ADBMS Monitoring System Architecture
  - Processing Single Event
  - Populating Events Table
  - Running Rules Engine
  - Full Monitoring System
- Building a Rules Engine for ADBMS
- MySQL as ADBMS
- Summary

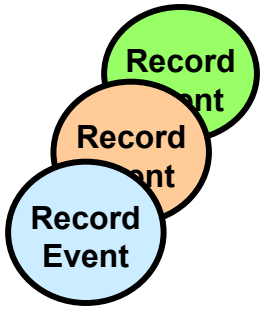
Processing Sensor Events



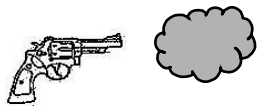
1. Sensor inserts data record into Sensor table
2. Sensor table trigger fires and action is executed
3. Trigger firing starts External record analysis application
4. It reads and analyzes the inserted record
5. It creates the event record based on the analysis

Events Table and Rules Engine

Record events from multiple sensors

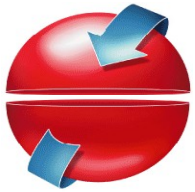


Events table



event_id	eventSrc	eventParams	timestamp
N	Sensor_N	……	2005-05-17 11:55:43
…….	…….	…….	…….
2	Sensor_2	……	…….
1	Sensor_1	SensorType, config, etc.	2005-05-17 11:55:12

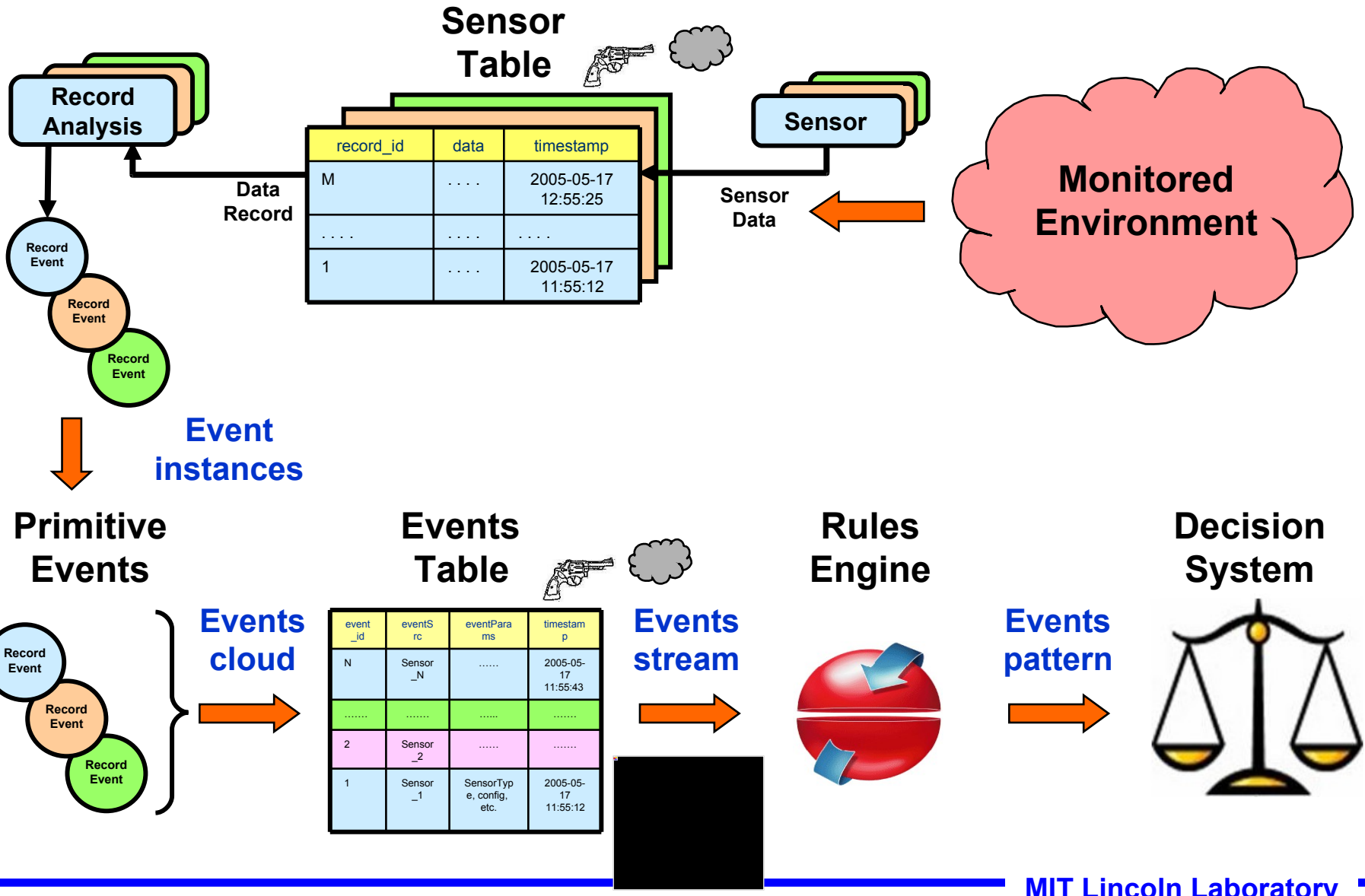
Rules Engine



Decision System

1. Each event record is inserted into Events table
2. Once the record is inserted the Events table trigger fires
3. Trigger firing launches Rules Engine Events table scanning
4. Rules Engines searches for Events pattern in the Events table
5. Once Events pattern found, the message is sent to Decision System

Running Rules Engine



Outline

MySQL Users Conference 2007

- Introduction
- Preventing Monitoring System Architecture
- Building a Rules Engine for ADBMS
  - Theory of Events
  - Primitive Events
  - Composite Events
  - Composite Event Generation
  - Triggers
  - Event-Condition-Action (ECA) Rules
- MySQL as ADBMS
- Summary

Theory of Events

- Event definition (monitoring systems)—recorded environment change
- Event definition (ADBMS)—change in the database state

Event Aspects

- Event form: how an event is represented
  - Event could be represented (recorded) as tuple of data components
- Event significance: how an event signifies activity
  - Event is a sign of activity. Analysis of events leads to activity understanding
- Event relativity: how an event relates to other events
  - Causality: which event caused the event to occur?
  - Time: when did the event happen?
- Event aggregation: does the event contain other events
  - Primitive events don't contain other events
  - Composite events are built from primitive events
- Event recurrence: do the events belong to the same type
  - Event classes
  - Event instances

Primitive Events

MySQL Users Conference 2007

- **Data modification**: raised on insert, update, or delete
- **Data reference**: raised on select
- **Stored procedure invocation**: raised before call, or after return
- **Message send/receive**: raised on send/receive of message
- **Transaction**: raised on transaction start, rollback, or commit
- **Exception**: raised on error (e.g., authorization failure)
- **Relative Timer**: raised after another specified event
- **Absolute Timer**: raised at a specified absolute time
- **Repetitive Timer**: raised periodically (e.g., every hour)
- **User-defined**: raised by an external application/device or another rule

Composite Events

MySQL Users Conference 2007

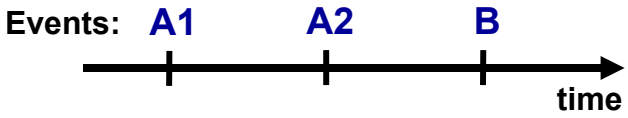
Composite events are built from primitive events, or other composite events using Event Algebra

- Sequence: E=(E1 ; E2) {E2 occurs after E1 (E1.time < E2.time), E.time=E2.time}
- Disjunction: E=(E1 | E2) {E2 occurs after E1 (E1.time < E2.time), E.time=E2.time}
- Conjunction: E=(E1, E2) {E2 occurs after E1 (E1.time < E2.time), E.time=E2.time}
- Negation: ¬E=[E1, E2] {E did not occur within [E1,time < E2.time], E.time=E2.time}
- Periodic: E=P(E1, T, E2) {E occurs every T=[E1.time,E2.time], E.time=E2.time}
- Cumulative periodic: {P* cumulates all occurrences and occurs only one time at E2}
- Aperiodic: E=A(E1, E2, E3) {E occurs when E2 occurs after E1, but before E3}
- Cumulative aperiodic: {A* cumulates all occurrences and occurs only one time at E3}
- ANY operator: ANY (k, E1,...,En) {E occurs when k < n distinct events occur}

Composite Event Generation

- **Primitive Event consumption policy**
  - Defines how primitive events are “consumed” by the composite event
  - Defines how primitive events are removed from further consideration

- **Consumption policy types**



E = {A;B}

  - **Recent:** only the most recent instances of any event {**A2** and **B**} are considered; older events are discarded
  - **Chronicle:** the oldest instances {**A1** and **B**} are considered and then discarded; i.e. events are consumed in chronological order
  - **Continuous:** all possible combinations raise separate events; {**A1** and **B**} as well as {**A2** and **B**}
  - **Cumulative:** all possible combinations of the primitive events are collected into one event {**A1**, **A2**, and **B**}

Triggers and ECA Rules

MySQL Users Conference 2007

- Trigger (active rule) is a language construct for defining the database reactions
- Defined by using Event-Condition-Action (ECA) rule language
- Active Rule Syntax:
  - ON *<event>* IF *<condition>* THEN *<action>*
  - If the event arises, the condition is evaluated
  - If the condition is satisfied, the action is executed
- Active rules originated from production rules of Artificial Intelligence or Expert Systems
- AI Production rules are executed for every request
  - IF *<condition>* THEN *<action>*
  - Active rules are executed only in case of events
- Events are recognized by the application or the database (in case of database event, the database is the application)

Outline

MySQL Users Conference 2007

- Introduction
- Preventing Monitoring System Architecture
- Building a Rules Engine for ADBMS
- MySQL as ADBMS
  - Triggers
    - Trigger syntax
    - Event and EventLog tables
  - Messaging
    - Servers, Daemons, and Applications
  - Events
    - Event syntax and usage
  - User Defined Functions (UDFs)
    - UDF Creation and Installation (Linux)
- Summary

MySQL Triggers

MySQL Users Conference 2007

Represent a reaction of a database to a change of its state

- **MySQL trigger features:**
  - Primitive event type—insert, delete, update
  - Activation time—before, after
  - Granularity—for each row
  - Transition variables—old, new
  - Actions—SQL statements, control flow, procedures, and UDF calls
- **What triggers are used for?**
  - Maintain the data constraints
  - Compute (update) materialized derived data
  - Maintain consistency across system catalogs or other metadata
  - Replicate, migrate, or log data from one table (database) to another
  - Manage new types of data (validate input) and keep external repositories consistent with internal data
  - Implement business rules, scheduling, workflow, and other kinds of application functionality
  - Notify users about changes in the database state usually in form of messages

Trigger syntax (MySQL version 5.1)

```
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
1. Define the trigger:
DELIMITER |
CREATE TRIGGER testref AFTER INSERT ON test1
   FOR EACH ROW BEGIN
      INSERT INTO test2 SET a2 = NEW.a1;
   END;
|
DELIMITER ;

2. Fire the trigger:
INSERT INTO test1 VALUES (4);
```

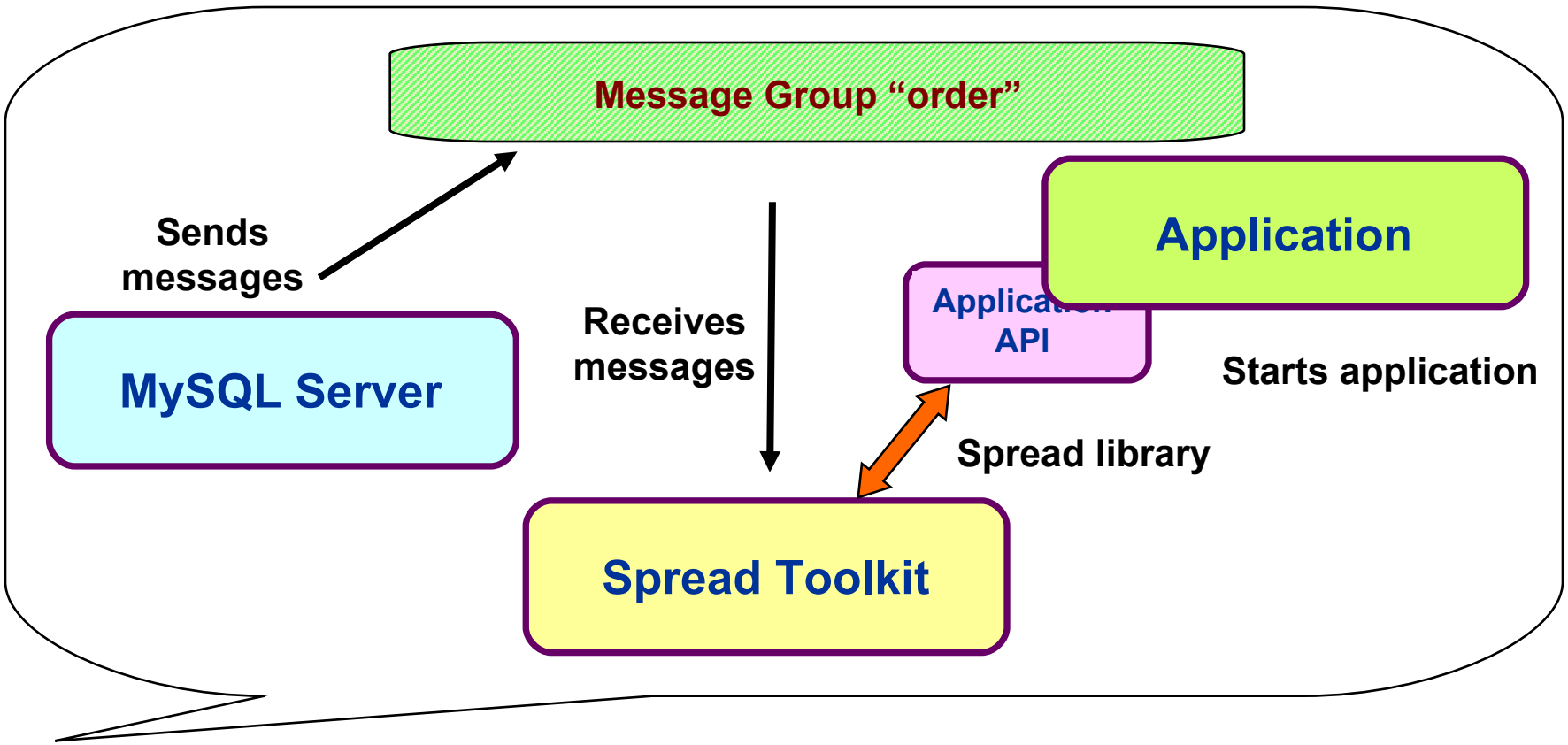
MySQL Messaging

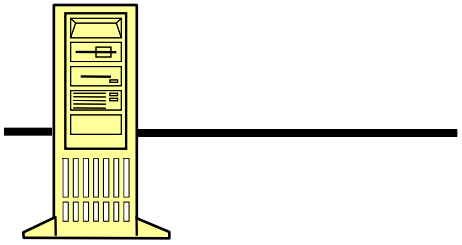
MySQL Users Conference 2007

MySQL servers can send and receive messages across the network using simple SQL queries

- **MySQL Message features:**
  - Messages are sent by the calls to User Defined Functions from SQL query
  - Messages are delivered by the Spread Toolkit
  - Messages are sent to the members of message group
  - Group members could be applications written in C, PHP, Perl, Java, etc.
- **Spread Toolkit**
  - Open source project developed by Johns Hopkins University
  - Backed up by the commercial license from Spread Concepts LLC
  - Provides messaging service across local and wide area networks
  - Capable to deliver up to 8 MB/sec with 10 K messages/sec
  - Support multicast, group communication, and point-to-point message delivery
  - Simple API for C and Java, easy to install, use, and maintain
- **Implementation**
  - The UDFs must be linked against the thread-safe Spread library
  - The Message APIs require a Spread daemon to be running on each MySQL server
  - The Spread daemons must be configured to define the domain for group membership

Servers, Daemons, and Applications





SELECT send_mesg("orders", byte_array)

- “orders”—message group
- Byte_array—message payload

MySQL Events

MySQL Users Conference 2007

MySQL servers can schedule and execute tasks at specified time with specified periodicity (temporal triggers)

- **Features:**
  - Temporal triggers are triggered by the passage of time, not the change of database state
  - Scheduled event is essentially a stored procedure with known start time
  - Scheduled event is a first class MySQL object with its own table, privilege, and log
  - One-time scheduled event—executes one time only
  - Recurrent scheduled event—repeats its action at a regular interval
- **What ADBMS functionality could those features be used for?**
  - Absolute Timer: raised at a specified absolute time
  - Repetitive Timer: raised periodically

MySQL Events (cont.)

MySQL Users Conference 2007

MySQL Event Syntax:

1. Create periodic scheduled event

```
CREATE EVENT my_event
   ON SCHEDULE
   EVERY 1 WEEK
   DO
      INSERT INTO t VALUES (9);
```

2. Turn on event_scheduler

```
SET GLOBAL event_scheduler = 1;
```

1. Create one-time scheduled event

```
CREATE EVENT my_event
   ON SCHEDULE
    AT TIMESTAMP '2006-01-20 12:00:00'
   DO
      INSERT INTO t VALUES (0);
```

2. Turn on event_scheduler

```
SET GLOBAL event_scheduler = 1;
```

Alter scheduled event

```
ALTER EVENT my_event
   ON SCHEDULE
   AT CURRENT_TIMESTAMP
   DO
      INSERT INTO t VALUES (7);
```

This event fires a trigger NOW

Drop scheduled event

```
DROP EVENT IF EXISTS  my_event
```

MySQL User Defined Functions (UDF)

MySQL Users Conference 2007

- **Purpose**
  - Implement functionality which does not exists in MySQL
  - Provide interface to existing libraries
  - Increase database performance
- **What ADBMS functionality should UDF implement?**
  - Start up the external program

    External program could be the Rule Engine to generate Composite Events
  - Send notification to external programs

    Message API UDFs already implements this functionality using Spread Toolkit
  - Efficient Events table scanning in search of Composite Events

    Composite Event generation could be done more efficiently without SQL

UDF Creation and Installation (Linux)

MySQL Users Conference 2007

1. Create the file 'so_system.c'

Make sure that "UDFs should have at least one symbol defined in addition to the xxx symbol that corresponds to the main xxx() function. These auxiliary symbols correspond to the xxx_init(), xxx_deinit(), xxx_reset(), xxx_clear(), and xxx_add() functions".

2. Compile the file 'so_system.c'

$ gcc -g -c so_system.c

3. Run linker with the file to create shared library

$ gcc -g -shared -W1,-soname,so_system.so.0 -o so_system.so.0.0 so_system.o -lc

4. Copy 'so_system.so.0' file into /usr/lib directory

cp so_system.so.0 /usr/lib

5. Create softlink with shared file to the real file name

ln -s so_system.so.0.0 so_system.so

6. Start up MySQL client

7. Run the MySQL SQL command

mysql> CREATE FUNCTION do_system RETURNS INTEGER soname 'so_system.so';

Query OK, 0 rows affected (0.00 sec)

UDF Creation and Installation (cont.)

8. Verify that the function is installed

The 'mysql.func' table then looks like this (you can also do the update manually):

mysql> select * from mysql.func;

```
+-----------+-----+-------------+----------+
| name      | ret | dl          |   type   |
+-----------+-----+-------------+----------+
| do_system |   2 | so_system.so | function |
+-----------+-----+-------------+----------+
```

1 row in set (0.00 sec)

9. Call the function with system command

The function can be called like this:

mysql> select do_system('ls > /tmp/test.txt');

```
+---------------------------------+
| do_system('ls > /tmp/test.txt') |
+---------------------------------+
|              -4665733612002344960|
+---------------------------------+
```

1 row in set (0.02 sec)

Summary

MySQL Users Conference 2007

- An active DBMS improves the efficiency of the monitoring applications
- Centralized and shared event knowledge between applications allows monitoring complex events
- Preventive monitoring could be implemented using the theory of events and active databases
- MySQL has all necessary features to be used as an active database for preventive monitoring applications