Oracle Berkeley DB
Data Store

A Use-Case Based Tutorial

Part I: Data Store (DS)

- Overview
  - Creating and removing databases
  - Getting and putting data
- Case Studies
  - Telecom use case
  - Customer account management

Berkeley DB Data Store

- Simple indexed data storage
- Attributes
  - Blindingly fast
  - Easy to use
  - APIs in the languages of your choice
- Limitations
  - No concurrency
  - No recovery

Blindingly Fast

- Platform Description
  - 2.33 GHz Intel Core 2 Duo
  - 2 GB 667 MHz DDR2 SDRAM
  - Mac OSX 10.4.10
  - Window manager and apps all up and running
- Insert 1,000,000 8-byte keys with 25-byte data
  - 180,000 inserts/second
- Retrieve same keys/data
  - 750,000 gets/second
- Bulk get of same key/data pairs
  - 2,500,000 pairs/second

Easy to Use

- Code for benchmark program
  - Main loop
  - Create/open database
  - Put keys
  - Get keys
  - Close/remove database
  - (Omitted) Command line processing, key generation

Main Program

```
•    main(argc, argv)
•            int argc;
•            char *argv[];
•    {
•            DB *dbp;
•            int do_n, rm_db, skip_put;

•            do_n = DEFAULT_N;
•            rm_db = 0;
•            skip_put = 0;

•            do_cmdline(argc, argv, &do_n, &rm_db, &skip_put);
•            create_database(&dbp);
•            if (!skip_put)
•                    do_put(dbp, do_n);
•            do_bulk_get(dbp, do_n);
•            do_get(dbp);
•            close_database(dbp, rm_db);
•    }
```

ORACLE

Create/Open Database

```
static void
create_database(dbpp)
        DB **dbpp;
{
        DB *dbp;

        /* Create database handle. */
        assert(db_create(&dbp, NULL, 0) == 0);

        /* Open/Create btree database. */
        assert(dbp->open(dbp,
            NULL, DBNAME, NULL, DB_BTREE, DB_CREATE, 0644) == 0);

        *dbpp = dbp;
}
```

Put Data

```
static void
do_put(dbp, n)
        DB *dbp;
        int n;
{
        char key[KEYLEN];
        DBT datadbt, keydbt;
        int i;

        /* Initialize key DBT (repeat with Data, omitted) */
        strncpy(key, FIRST_KEY, KEYLEN);
        memset(&keydbt, 0, sizeof(keydbt));
        keydbt.data = key;
        keydbt.size = KEYLEN;
        for (i = 0; i < n; i++) {
                assert(dbp->put(dbp,
                    NULL, &keydbt, &datadbt, 0) == 0);
                next_key(key);
        }
}
```

Get Data

```
static void
do_get(dbp)
        DB *dbp;
{
        DBC *dbc;
        DBT keydbt, datadbt;
        int n, ret;

        assert(dbp->cursor(dbp, NULL, &dbc, 0) == 0);
        memset(&keydbt, 0, sizeof(keydbt));
        memset(&datadbt, 0, sizeof(datadbt));
        n = 0;
        TIMER_START;
        while ((ret = dbc->get(dbc,
            &keydbt, &datadbt, DB_NEXT)) == 0) {
                n++;
        }
        TIMER_STOP;
        TIMER_DISPLAY(n);
}
```

Close and Remove Database

```
•    static void
•    close_database(dbp, remove)
•            DB *dbp;
•            int remove;
•    {
•            assert(dbp->close(dbp, 0) == 0);
•            if (remove) {
•                    assert(db_create(&dbp, NULL, 0) == 0);
•                    assert(dbp->remove(dbp,
•                        DBNAME, NULL, 0) == 0);
•            }
•    }
```

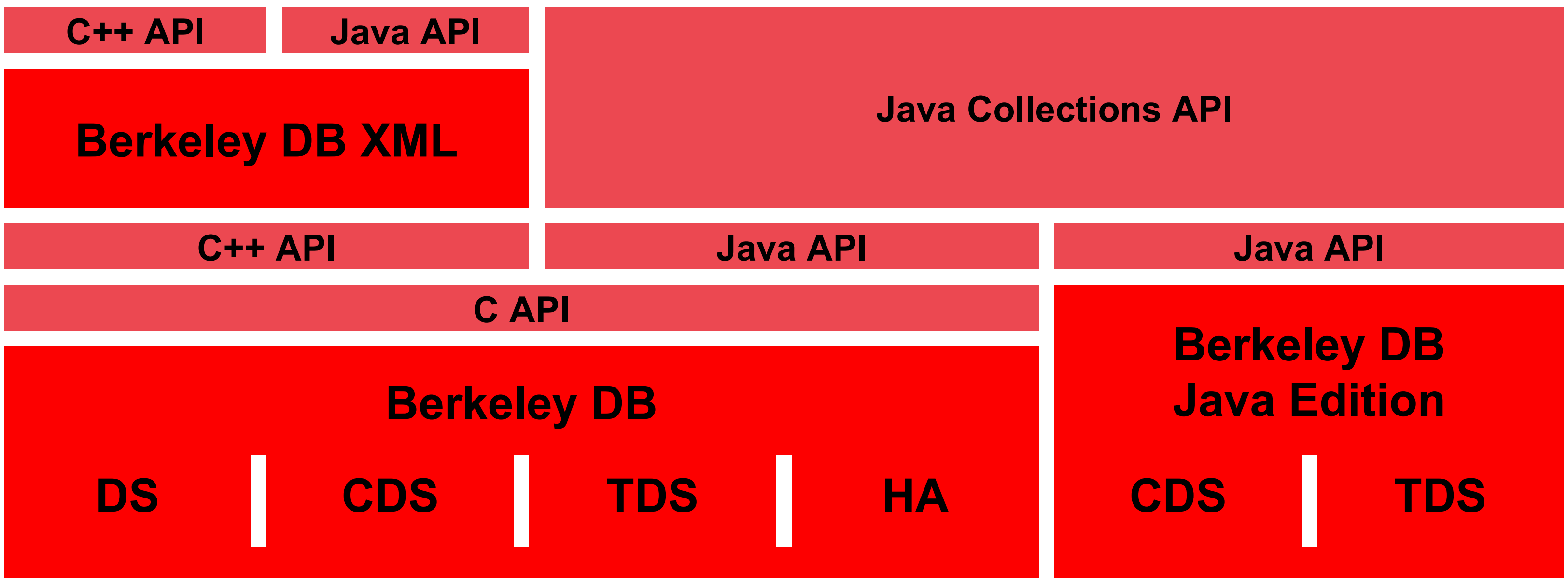
ORACLE

APIs in many Languages

- C/C++
- Java
- Python
- Perl
- Tcl
- Ruby
- Etc.

Berkeley DB Product Family

APIs for C, C++, Java, Perl, Python, PHP, Ruby, Tcl, Eiffel, etc.



Runs on UNIX, Linux, MacOS X, Windows, VxWorks, QNX, etc.

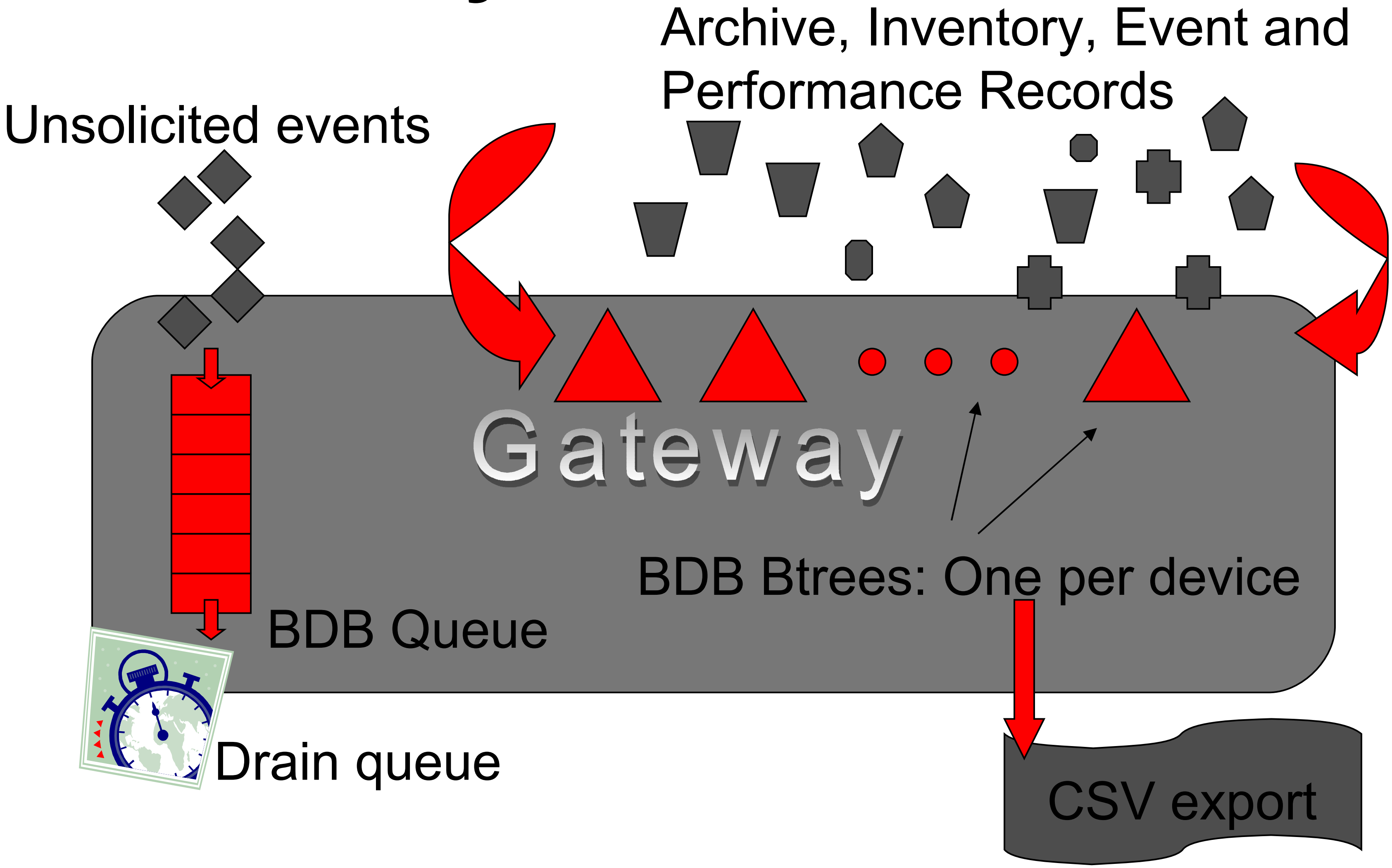
Part I: Data Store

- Overview
  - Creating and removing databases
  - Getting and putting data
- Case Studies
  - Telecom Use Case
    - Creating/using an environment
    - Filling/draining a queue
    - Btrees: storing different record types in a single database
    - Cursors
  - Customer account management

Telecommunications Use Case

- Radio access network optimization system
- Vendor-independent multi-service multi-protocol RAN aggregation plus optimization
- Interoperable with all major base stations
- Proven, world-wide deployment
- Functional over range of operating conditions
- Low TCO

Berkeley DB in Telecom



BDB Access Methods

- DB_QUEUE/DB_BTREE:
  - Berkeley DB supports a variety of different indexing mechanisms.
  - Applications select the appropriate mechanism for the data management task at hand.
  - For example:
    - DB_QUEUE: FIFO order, fixed size data, numerical keys
    - DB_BTREE: Clustered lookup, variable length keys/data
- Other index types (access methods)
  - Hash: truly large unordered data
  - Recno: data with no natural key; numerical keys assigned

Databases & Environments

- Each different data collection is a *database*.
- Databases may be gathered together into *environments*.
- An environment is a logically related collection of databases.
- Example:
  - The NMS queue and per-device btrees all reside in a single environment for a single instance of AccessGate.

Creating Environments

- Create the handle using `db_env_create()`
- Open using `DB_ENV->open()`
  - Use `DB_CREATE` flag to create a new environment.
  - Use `DB_INIT_MPOOL` for a shared memory buffer pool
  - Use `DB_PRIVATE` to keep environment in main memory.
  - Use `DB_THREAD` to allow threaded access to the DB_ENV handle.

Environment Example

```
/* Create handle. */
ret = db_env_create(&dbenv, 0);
    … error_handling …

ret = dbenv->set_cachesize(dbenv, 1, 0, 1);
    … error_handling …

/* Other configuration here. */

/* Create and open the environment. */
flags = DB_CREATE |DB_INIT_MPOOL;

ret = dbenv->open(dbenv, HOME, flags, 0);
 … error_handling …
```

ORACLE

Filling and Draining a Queue

- Creating a queue database.
- Appending into a queue.
- Draining a queue
- Important attributes:
  - Queue supports only fixed-length records
  - Queue “keys” are integer record numbers
  - Queues *do* support random access

Creating a Queue Database

- Create the handle using `db_create()`
- Open using `DB->open()`
  - Use `DB_CREATE` flag to create a new database
- Close using `DB->close()`
  - Never close a database with open cursors or transactions.

Creating a Queue

```
/* Create a database handle. */
ret = db_create(&dbp, dbenv, 0);
    … error_handling …

/* Set the record length to 256 bytes. */
ret = dbp->set_relen(dbp, 256);
 … error_handling …
ret = dbp->set_repad(dbp, (int)’ ‘);
 … error_handling …

/* Create and open the database. */
ret = dbp->open(dbp, NULL, file, database,
    DB_QUEUE,DB_CREATE, 0666);
     … error_handling …
```

Enqueuing Data

- `DB->put()` inserts data into a database.
- Appending onto the end of a queue:

```
DBT key, data;
memset(&key, 0, sizeof(key));
memset(&data, 0, sizeof(data));
data->data = "fill in the record here";
data->size = DATASIZE;

ret = dbp->put(dbp, NULL, &key, &data, DB_APPEND);
if (ret != 0)
    … error_handling …

/* Key->data contains the new record number. */
```

Draining the Queue

- `DB->get()` retrieves records

```
memset(&key, 0, sizeof(key));
memset(&data, 0, sizeof(data));

While ((ret = dbp->get(dbp,
        NULL, &key, &data, DB_CONSUME) == 0) {
      /* Do something with record. */
}

if (ret != 0)
        … error_handling …
```

ORACLE

Btrees: Different record types in a single database

- One btree per device
- Each btree contains four record types
  - Archive
  - Inventory
  - Event
  - Performance
- Not necessary to have N databases for N record types.

Record Types

- Different records (with different fields/sizes):

```
struct archive {
        int32_t   field1;
        … other fields here
};

struct inventory {
        double    field1;
        … other fields here
};
```

```
struct event {
            int64_t   field1;
            … other fields here
            };

struct perf {
            char      field1[16];
            … other fields here

};
```

- Assumptions:
  - Every record has a timestamp
  - May need to encode types of different records

Data Design 1

- Index database by timestamp
  - Add “type” field to every record.
  - Allow duplicate data records for any timestamp
- Advantages:
  - Groups data temporally
- Disadvantages:
  - Difficult to get records of a particular type

Allowing Duplicates

```
int create_database (DB_ENV *dbenv, DB *dbpp, char *name)
{
    DB *dbp;
    int ret;

    if ((ret = db_create(&dbp, dbenv, 0)) != 0)
           return (ret);

    if ((ret = dbp->set_flags(dbp, DB_DUP)) != 0 ||
         ((ret = dbp->open(dbp, NULL, name, NULL, DB_BTREE, 0, 0644)) != 0)
           (void)dbp->close(dbp, 0);
           return (ret);

    *dbpp = dbp;
    return (ret);
}
```

ORACLE

Inserting a Record (design 1)

```
int insert_archive_record(DB *dbp, timestamp_t ts, char *buf)
{
   DBT  key, data;
   struct archive a;

   memset(&key, 0, sizeof(key));
   memset(&data, 0, sizeof(data));

   key->data = &ts;
   key->size = sizeof(ts);

   a->type = ARCHIVE_TYPE;
   BUFFER_TO_ARCHIVE(buf, a);/
   data->data = &a;
   data->size = sizeof(a);

   return (dbp->put(dbp, &key, &data, 0));
}
```

Data Design 2

- Index database by type
  - Add “timestamp” field to every record.
  - Allow duplicate data records for any type.
- Advantages:
  - Groups data by type
- Disadvantages:
  - Difficult to get records for a given time
  - Many records will have the same type

Inserting a Record (design 2)

```
int insert_archive_record(DB *dbp, timestamp_t ts, char *buf)
{
   DBT  key, data;
   RECTYPE type;
   struct archive a;

   memset(&key, 0, sizeof(key));
   memset(&data, 0, sizeof(data));

   type = ARCHIVE_TYPE;
   key->data = &type;
   key->size = sizeof(type);

   a->timestamp = ts;
   BUFFER_TO_ARCHIVE(buf, a);
   data->data = &a;
   data->size = sizeof(a);

   return (dbp->put(dbp, &key, &data, 0));
}
```

Data Design 3

- Index database by type and timestamp
  - May not need to allow duplicates
- Advantages:
  - Groups data by type and by timestamp within type
  - Fast retrieval by type
  - Reasonable retrieval by timestamp
- Disadvantages:
  - Nothing obvious

Inserting a Record (design 3)

```
int insert_archive_record(DB *dbp, timestamp_t ts, char *buf)
{
   DBT  key, data;
   struct archive a;
   struct akey {
        timestamp_t ts;
        RECTYE type;
   } archive_key;

   memset(&key, 0, sizeof(key));
   memset(&data, 0, sizeof(data));

   archive_key.ts = ts;
   archive_key.type = ARCHIVE_TYPE;
   key->data = &archive_key;
   key->size = sizeof(archive_key);
   BUFFER_TO_ARCHIVE(buf, a);
   data->data = &a;
   data->size = sizeof(a);
   return (dbp->put(dbp, &key, &data, 0));
}
```

ORACLE

Data Design 4

- Create four databases in a single file
  - Key each database by timestamp
- Advantages:
  - Groups data by type
  - Fast retrieval by type
  - Reasonable retrieval by timestamp
- Disadvantages:
  - Nothing obvious

Multiple Database in a File

```
int create_database (DB_ENV *dbenv, DB *db_array, char **names)
{
    DB *dbp;
    int i, ret;

    for (i = 0; i < 4; i++) {
          db_array[i] = dbp = NULL;
          if ((ret = db_create(&dbp, dbenv, 0)) != 0 ||
             ((ret = dbp->open(dbp,
              NULL, “DBFILE”, names[i], DB_BTREE, 0, 0644)) != 0)
                    if (dbp != NULL)
                              (void)dbp->close(dbp, 0);
                    break;

          db_array[i] = dbp;
    }
    if (ret != 0)
          /* Close all non-Null entries in db_array. */
    return (ret);
```

ORACLE

Inserting a Record (design 4)

```
int insert_record(DB *db_array, timestamp_t ts, RECTYPE type, char *buf, size_t buflen)
{
    DBT   key, data;
    memset(&key, 0, sizeof(key));
    memset(&data, 0, sizeof(data));
    key->data = &ts;
    key->size = sizeof(ts);
    data->data = buf;
    data->size = buflen;

    if (type == ARCHIVE_TYPE)
          ret = put_rec(db_array[ARCHIVE_NDX], &key, &data);
    else if (type == EVENT_TYPE)
          ret = put_rec(db_array[EVENT_NDX], &key, &data);
    else if (type == INVENTORY_TYPE)
          ret = put_rec(db_array[INVENTORY_NDX], &key, &data);
    else if (type == PERF_TYPE)
          ret = put_rec(db_array[PERF_NDX], &key, &data);
    else  /* Signal invalid record type */

    return (ret);
}
```

ORACLE

Inserting a Record (cont)

```
int put_rec(DB *dbp, DBT *key, DBT *data)
{
    return (dbp->put(dbp, NULL, &key, &data, 0));
}
```

Data Design:Summary

- Berkeley DB’s schemaless design provides applications flexibility.
- Tune data organization for performance, ease of programming, or other criteria.
- Create indexes based upon query needs.

Exporting to CSV

- Assume that we used design 4
  - Four databases in a single file
  - One database per record type
- Let's say that we want one CSV file per database.
- Each CSV export will look very similar:
  - Use a cursor to iterate over the database.
  - For each record, output the CSV entry.

Iteration with cursors

- A cursor represents a position in the database.
- The DB->cursor method creates a cursor.
- Cursors have methods to get/put data.
  - DBC->del
  - DBC->get
  - DBC->put
- Close cursors when done.

Iteration (code)

```
int archive_to_csv(DB *dbp)
{
    DBC dbc;
    DBT key, data;
    struct archive *a;
    int ret;

    if ((ret = dbp->cursor(dbp, NULL, &dbc, 0)) != 0)
          return (ret);
    memset(&key, 0, sizeof(key));
    memset(&data, 0, sizeof(data));
    while ((ret = dbc->get(dbc, &key, &data, DB_NEXT)) == 0) {
          a = (struct archive *)data->data;
          /* Output archive structure into CSV */
    }
    if (ret == DB_NOTFOUND)
          ret = 0;
    return (ret);
}
```

Cleaning out the Database

- After creating the CSV, we want to remove the data from the Berkeley DB databases.
- Two options:
  - Delete and recreate the database.
  - Truncate the database (leave the database and remove all the data).

Removing a Database

- Removal with a DB handle (un-opened)

```
ret = db_create(&dbp, NULL, 0);
    … error_handling …
ret = dbp->remove(dbp, file, database, 0);
    … error handling …
```

- Removal with a DB_ENV (no DB handle)

```
ret = dbenv->dbremove(dbenv, NULL,
    file, database, 0);
        … error handling …
```

Truncating a Database

- Truncate with an open DB handle

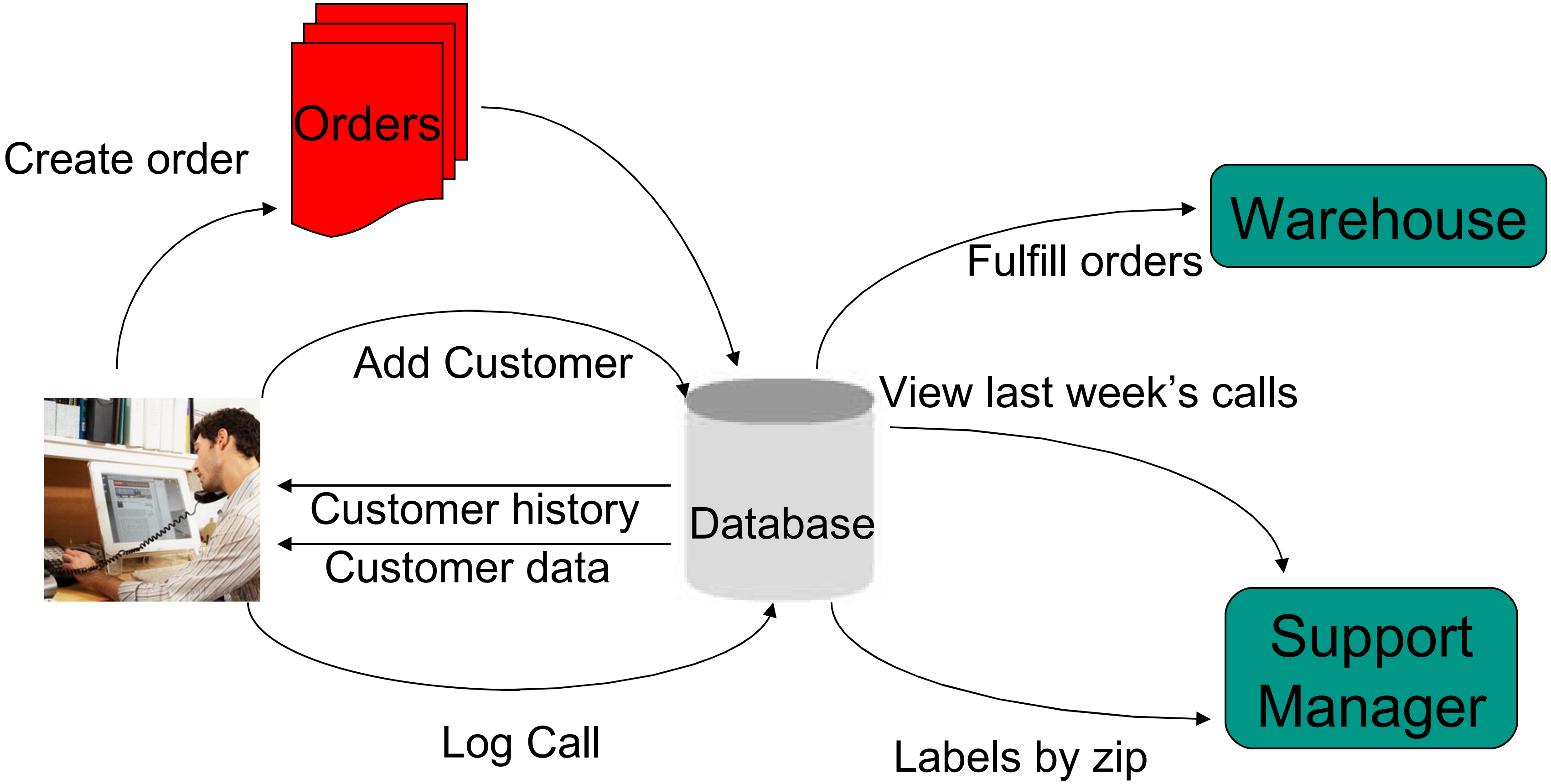
```
ret = dbp->truncate(dbp, NULL, &nrecords, 0);
    … error handling …
```

- Nrecords returns the number of records removed from the database during the truncate.

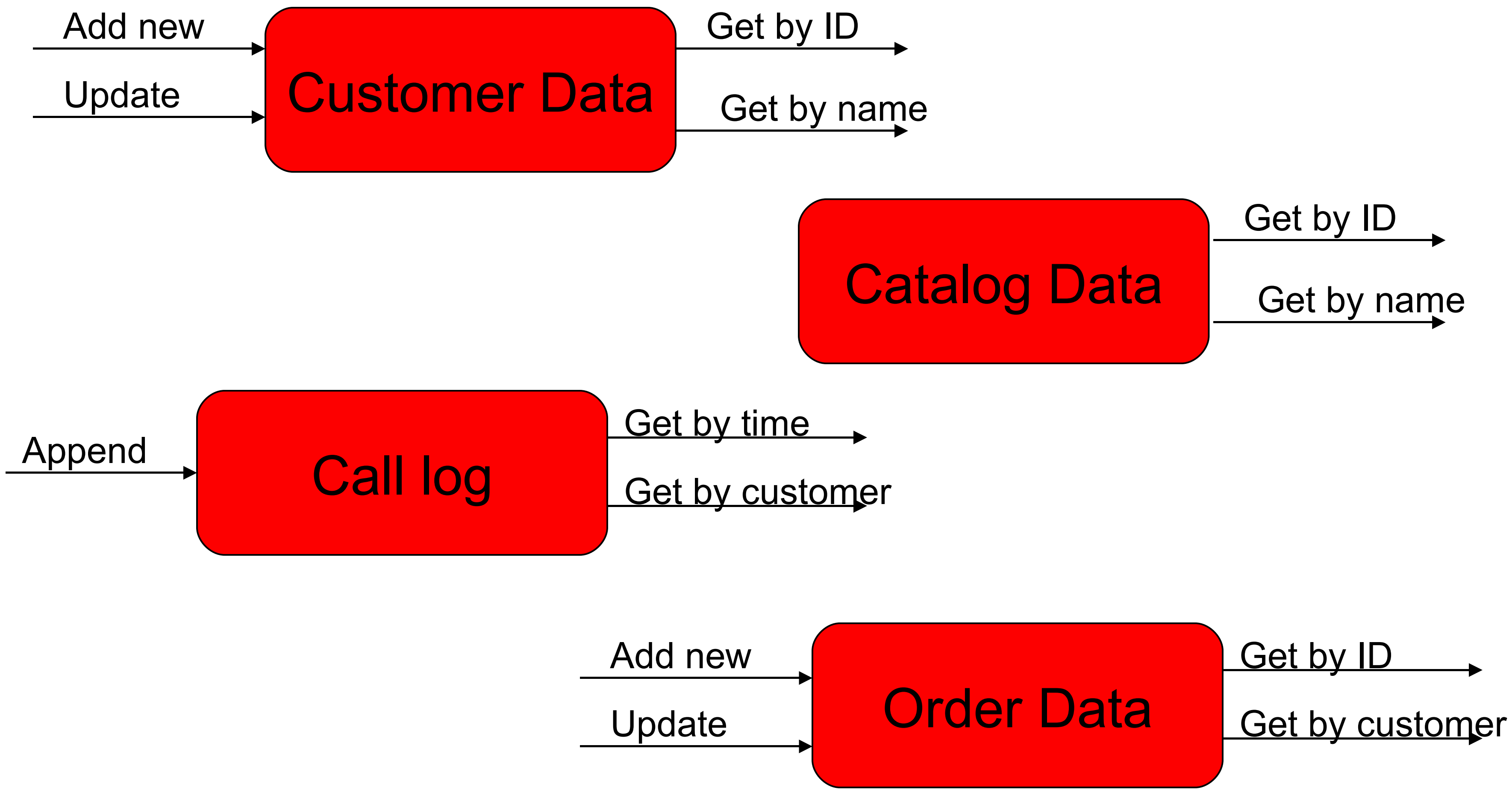
Part I: Data Store

- Overview
  - Creating and removing databases
  - Getting and putting data
- Case Studies
  - Telecom Use Case
  - Customer account management
    - Secondary indexes
    - Cursor-based manipulation
    - Configuration and tuning

Customer Account Management



The Schema


Add new
Update
Customer Data
Get by ID
Get by name
Catalog Data
Get by ID
Get by name
Append
Call log
Get by time
Get by customer
Add new
Update
Order Data
Get by ID
Get by customer

Secondary Indexes

- Note that most of the data objects require lookup in (at least) two different ways.
- Berkeley DB keys are primary (clustered) indexes.
- Berkeley DB also supports secondary (unclustered) indexes.
  - Automatic maintenance in the presence of insert, delete, update

Selecting keys and indexes

Database	Primary Key	Secondary Key	Duplicates?
Customer	Customer ID	Name	No
Catalog	Product ID	Name	No
Order	Order ID	Customer	Maybe
Calls	Timestamp	Customer	No

Secondaries internally

Customer Primary Database	
Key	Data
01928374	IBM; Hawthorne, NY …
19283746	HP; Cupertino, CA …
28910283	Dell; Austin, TX …
…	…

Customer Secondary	
Key	Data
Dell	28910283
IBM	01928374
HP	19283746
…	…

Secondaries Programmatically

- Create new database to contain secondary.
- Define callback function that extracts secondary key from primary key/data pair.
- Associate secondary with (open) primary.

Customer Secondary (1)

- Assume our customer data item contains a structure:

```
struct customer {
        char name[20];
        char address1 [20];
        char address2 [20];
        char state[2];
        int zip;
};
```

Customer Secondary (2)

- Need callback function to extract secondary key, given a primary key/data pair:

```
int customer_callback(DB *dbp, DBT *key, DBT *data, DBT *result)
{
        struct customer rec;

        rec = data.data;
        result.data = rec.name;
        result.size = sizeof(rec.name);
        return (0);
}
```

Customer Secondary (3)

- Associate the (open) primary with the (open) secondary:

```
ret = pdbp->open(pdbp, NULL, "customer.db", NULL, DB_BTREE, 0);
if (ret != 0)
    return (ret);
ret = sdbp->open(sdbp, NULL, "custname.db", NULL, DB_BTREE, 0);
if (ret != 0){
    (void)pdbp->close(pdbp, 0);
    return (ret);
}
ret = pdbp->associate(pdbp, NULL, sdbp, customer_callback, 0);
if (ret != 0)
    return (ret);
```

Representing Orders

- What is an order?
  - An order number
  - A customer
  - A date/time
  - A collection of items
- How do we represent orders?
  - Key by order number with items in a single data item
  - Key by order number with duplicates for items
  - Key by order number/sequence number

Orders: Multiple Items in Data

```
typedef int       order_id;         /* Primary key */
struct order_info {
    int           customer_id;
    time_t        date;
    float         total;
    time_t        shipdate;
    int           nitems;           /* Number of items to follow. */
    char[1]       order_items;      /* Must martial all items into here */
};

Struct order_item {
    int           item_id;
    int           nitems;           /* per-item price stored elsewhere */
    float         total;
};
```

ORACLE

Orders: Multiple Items in Data -- Secondaries

- Secondary on customer_id
  - Just like what we did on customer
  - Create a callback that extracts the customer_id field from the data field.
  - Associate secondary with primary.
- Secondary on items
  - Need to return multiple secondary keys for a single primary record.

Secondaries: Return multiple keys per key/data pair

```
int item_callback(DB *dbp, DBT *key, DBT *data, DBT *result)
{
        struct order_info rec;
        DBT **dbta;
        int i;

        order = data.data;
        result.flags = DB_DBT_MULTIPLE | DB_DBT_APPMALLOC;
        result.size = order->nitems;
        result.data = calloc(sizeof(DBT), order->nitems);
        if (result.data == NULL)
                return (ENOMEM);
        dbta = result.data;
        for (i = 0; i < order->nitems; i++) {
                dbta[i].size = sizeof(order_item);
                dbta[I].data = order->order_items[i];
        }
        return (0);
}
```

ORACLE

Secondaries: Example of multiple keys per key/data pair

Order Primary Database	
Key	Data
9876543	19283746;3/17/08;138.50;3/18/09;3 0003;2;130.00;0019;1;8.50
8769887	19283877;3/16/08;1024.00;3/16/0810 0092;10;1024.00
5430987	90867654;3/16/08;564.98;;5 0003;3;195.00;0092;1;102.40; 9902;1;267.58
…	…

Order/Item Secondary	
Key	Data
0003	9876543 **5439087**
0019	9876543
0092	8769887 **5430987**
9902	**5430987**
…	…

Orders: Duplicates for Items

- Modify previous structures slightly:

```
typedef int          order_id; /* Primary key */
struct order_info {
    int              customer_id;
    time_t date;
    float            total;
    time_t shipdate;
    int              nitems;           /* Number of items to follow. */
};

struct order_item {
    int              item_id;
    int              nitems;           /* per-item price stored elsewhere */
    float            total;
};
```

- First duplicate is order; rest are items

Orders: Items as duplicates Example

Order Primary Database	
Key	Data
9876543	19283746;3/17/08;138.50;3/18/09
	0003;2;130.00
	0019;1;8.50
8769887	19283877;3/16/08;1024.00;3/16/0810
	0092;10;1024.00
5430987	90867654;3/16/08;564.98;;5
	0003;3;195.00
	0092;1;102.40
	9902;1;267.58

ORACLE

Orders: Duplicates for Items
Inserting an order

```
int insert_order(int order_id, struct orderer_info *oi;
    struct order_items **oip)
{
        DBT keydbt, datadbt;
        int I;

        keydbt.data = &order_id;
        keydbt.size = sizeof(order_id);
        datadbt.data = oi;
        datadbt.size = sizeof(order_info);
        dbp->put(dbp, NULL, &keydbt, &datadbt, DB_NODUPDATA);
        for (i = 0; i < oi->nitems; i++) {
                datadbt.data = *oip++;
                datadbt.size = sizeof(order_item);
                dbp->put(dbp, NULL, &keydbt, &datadbt, 0);
        }
}
```

ORACLE

Orders: Duplicates for Items Trade-offs

- Must configure database for duplicates.
- - No automatic secondaries (must hand-code).
- + Easy to add/remove items from order.

Orders: Key by order_id/seqno

```
struct pkey {     /* Primary key */
    int   order_id;
    int   seqno;
};
struct order_info {
    int            customer_id;
    time_t         date;
    float          total;
    time_t         shipdate;
    int            nitems;          /* Number of items to follow. */
};

struct order_item {
    int            item_id;
    int            nitems;          /* per-item price stored elsewhere */
    float          total;
};
```

Orders: key by order/seqno Example

Order Primary Database	
Key	**Data**
9876543/00	19283746;3/17/08;138.50; 3/18/09
9876543/01	0003;2;130.00
9876543/02	0019;1;8.50
8769887/00	19283877;3/16/08;1024.00; 3/16/0810
8769887/01	0092;10;1024.00
5430987/00	90867654;3/16/08;564.98;;5
5430987/01	0003;3;195.00
5430987/02	0092;1;102.40
5430987/03	9902;1;267.58

Order/Item Secondary	
Key	**Data**
0003	9876543/01
0003	5439087/01
0019	9876543/02
0092	8769887/01
0092	5430987/02
9902	5430987/03
…	…

Orders: Key by order/seqno Inserting an order

```
int insert_order(int order_id, struct orderer_info *oi;
    struct order_items **oip)
{
        DBT keydbt, datadbt;
        struct pkey;
        int i;

        pkey.order_id = order_id;
        pkey.seqno = 0;
        keydbt.data = &pkey;
        keydbt.size = sizeof(pkey);
        datadbt.data = oi;
        datadbt.size = sizeof(order_info);
        dbp->put(dbp, NULL, &keydbt, &datadbt, DB_NODUPDATA);
        for (i = 0; i < oi->nitems; i++) {
                pkey.seqno = i + 1;
                datadbt.data = *oip++;
                datadbt.size = sizeof(order_item);
                dbp->put(dbp, NULL, &keydbt, &datadbt, 0);
        }
}
```

ORACLE

Order Trade-offs

- ID Key w/multiple items in data
  - One get/put per order
  - No duplicates means automatic secondary support
  - Add/delete/update item is somewhat messier
- ID key w/duplicate data items
  - Easy to add/delete/update order items
  - Must implement secondaries manually
  - Where do you place per-order info (CID, data) (first dup?)
- Key is ID/sequence number
  - Easy to add/delete/update order items
  - Can support secondaries automatically
  - Where do you place per-order info (data item 0?)

End of Part I