Falcon詳細解説

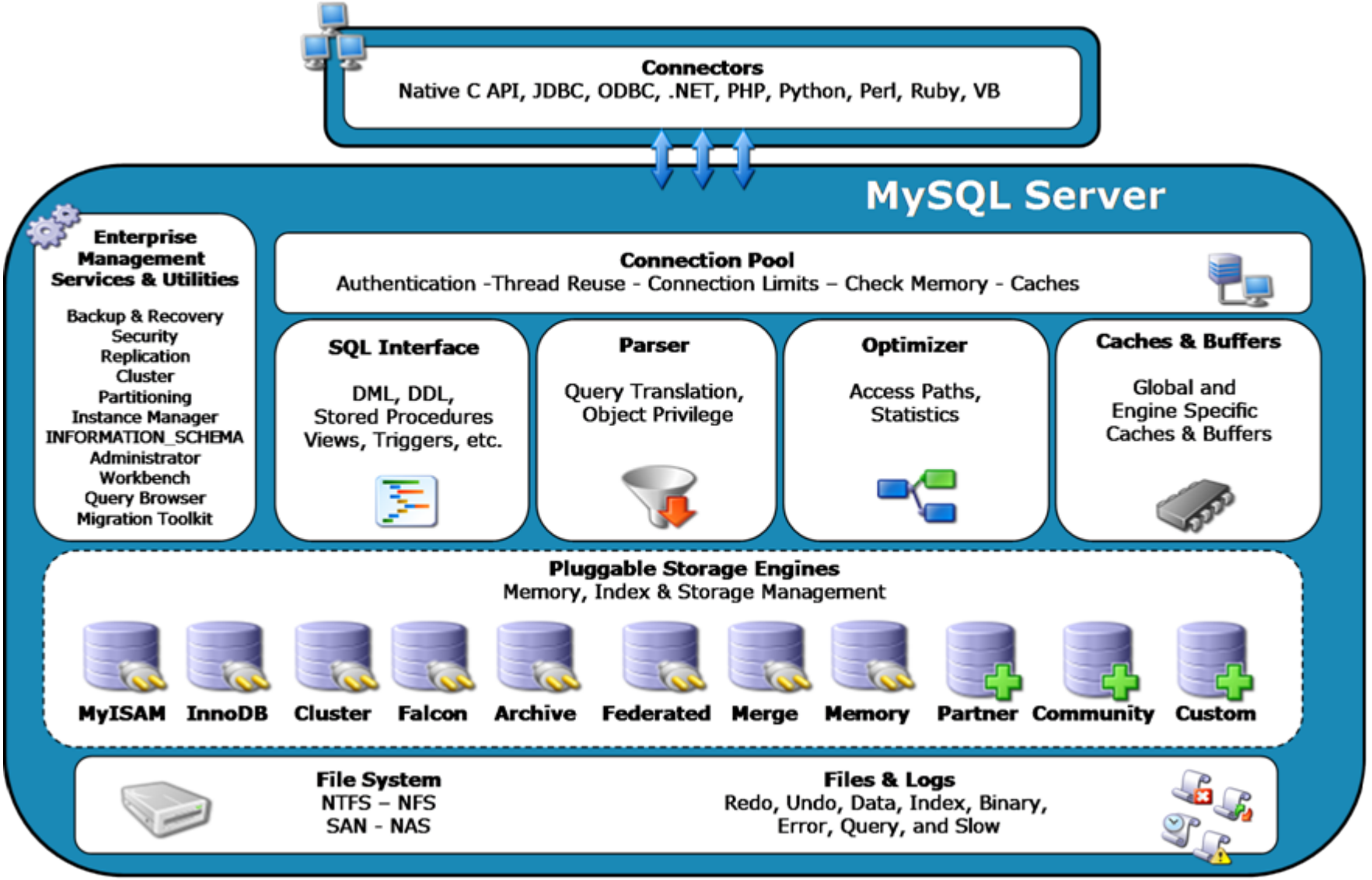
松信 嘉範(MATSUNOBU Yoshinori)
MySQL株式会社
シニアコンサルタント
ymatsunobu@mysql.com

セッション内容

- MySQLのアーキテクチャ概要
- Falconとは
- InnoDBとの機能差異

- プロセス、スレッド、メモリ構成
- インデックス構成
- レコード取得の流れ
- データ型と消費サイズ
- トランザクション制御

MySQLのアーキテクチャ



Falconとは

- MySQL ABにより現在開発中の、トランザクション対応のストレージエンジン
- Jim Starkey氏を中心に開発
  - Interbaseの開発者。BLOBやMVCCの発明者でもある。
- InnoDBをほぼ全ての点で上回ることを目指している
- 現在はアルファ版、間もなくベータ版が登場
- MySQL 6.0で安定版を搭載予定

- 思想：現代的なハードウェア環境をフル活用できるRDBMSを目指す
  - マルチコアCPU
  - 大容量メモリ
  - 低速なディスク、RAID

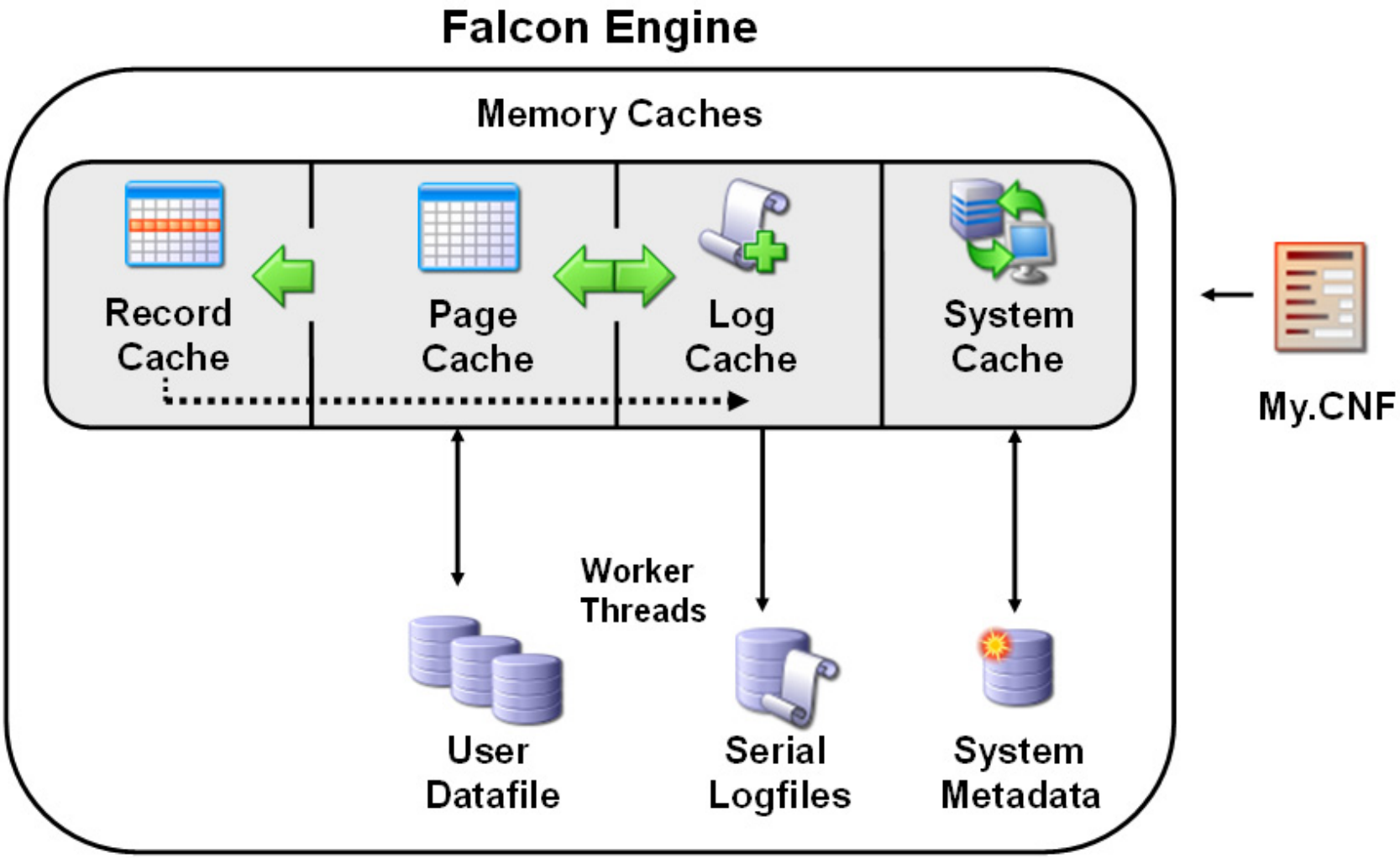
InnoDBとの主な差異

- クラスタ索引を採用していない
- 行ベースのレプリケーション(Binary Logging)のみをサポートし、文ベースのレプリケーションをサポートしない
- 分離レベル「Read Uncommitted」をサポートしない

ほかのストレージエンジンとの比較

特徴	Falcon	InnoDB	NDB	MyISAM
クラッシュリカバリ	☑	☑	☑	
外部キー	予定	☑		
テーブルスペース	☑	☑	☑	
領域の自動拡張	☑	☑	☑	☑
専用のキャッシュ領域(データ)	☑	☑	☑	
専用のキャッシュ領域(インデックス)	☑	☑	☑	☑
トランザクション	☑	☑	☑	
セーブポイント	☑	☑		
MVCC	☑	☑		
行レベルロック	☑	☑	☑	
デッドロックの検知	☑	☑	☑	
GIS型のサポート	☑	☑		☑
B-Tree索引	☑	☑	☑	☑
クラスタ索引		☑		
レプリケーション	Row	Stmt/Row	Row	Stmt/Row

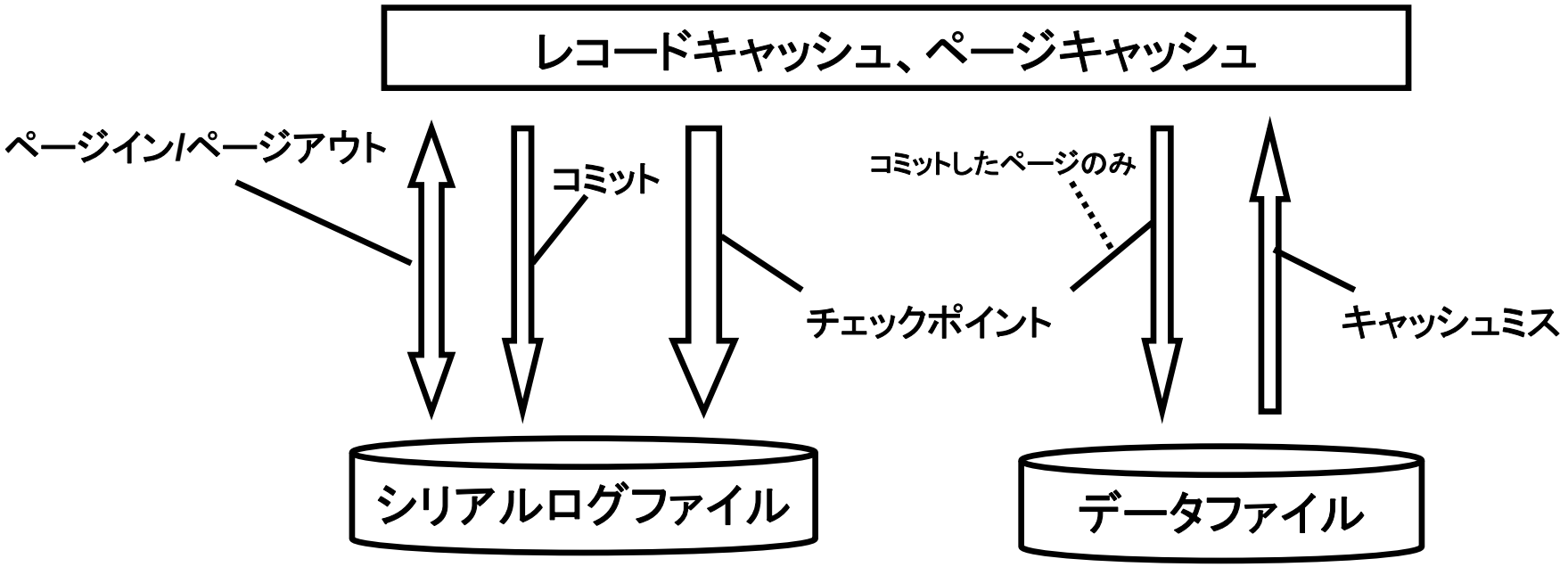
Falconのアーキテクチャ概略



プロセス/スレッド

- MySQLはマルチスレッド型アーキテクチャ
  - 接続1個に対してスレッドを1個割り当てて動作
- Falcon専属の、バックグラウンドで動作するスレッドがある
  - キャッシュ管理 (古いレコードの退避など)
  - シリアルログの中身をデータファイル等に反映 (後述)
- mutexをラッピングして排他制御 (SyncObjects)
  - Read/Writeロックを導入
  - Readは他のReadをブロックしない (※行レベルロックとは別物)
  - 同時実行性を大きく高める効果がある
  - マルチCPUコア環境で重要な技術

ファイルの読み書き



- シリアルログファイルはREDOログに相当。ただし、サイズは固定ではない
- 未コミットの情報はデータファイルに書かれない
  - -ロールバックの高速化
  - -ランダムI/Oの回数が減る
- コミットした情報は最終的にデータファイルに反映される
- Falconの「グループコミット」機能は極めて高性能

ファイル書き込み方式

	InnoDB	Falcon
REDO ログ	コミット/チェックポイント - O_SYNC + write - Write + fsync - Write (innodb_flush_log_trx_commit=0 or 2)	コミット/チェックポイント -O_SYNC + write -Write + fsync (サポート予定) -Write (サポート予定)
データファイル	チェックポイント - Write + fsync - O_DIRECT + write + fsync	チェックポイント -Write + fsync -非同期I/Oをサポート予定

表領域(テーブルスペース)

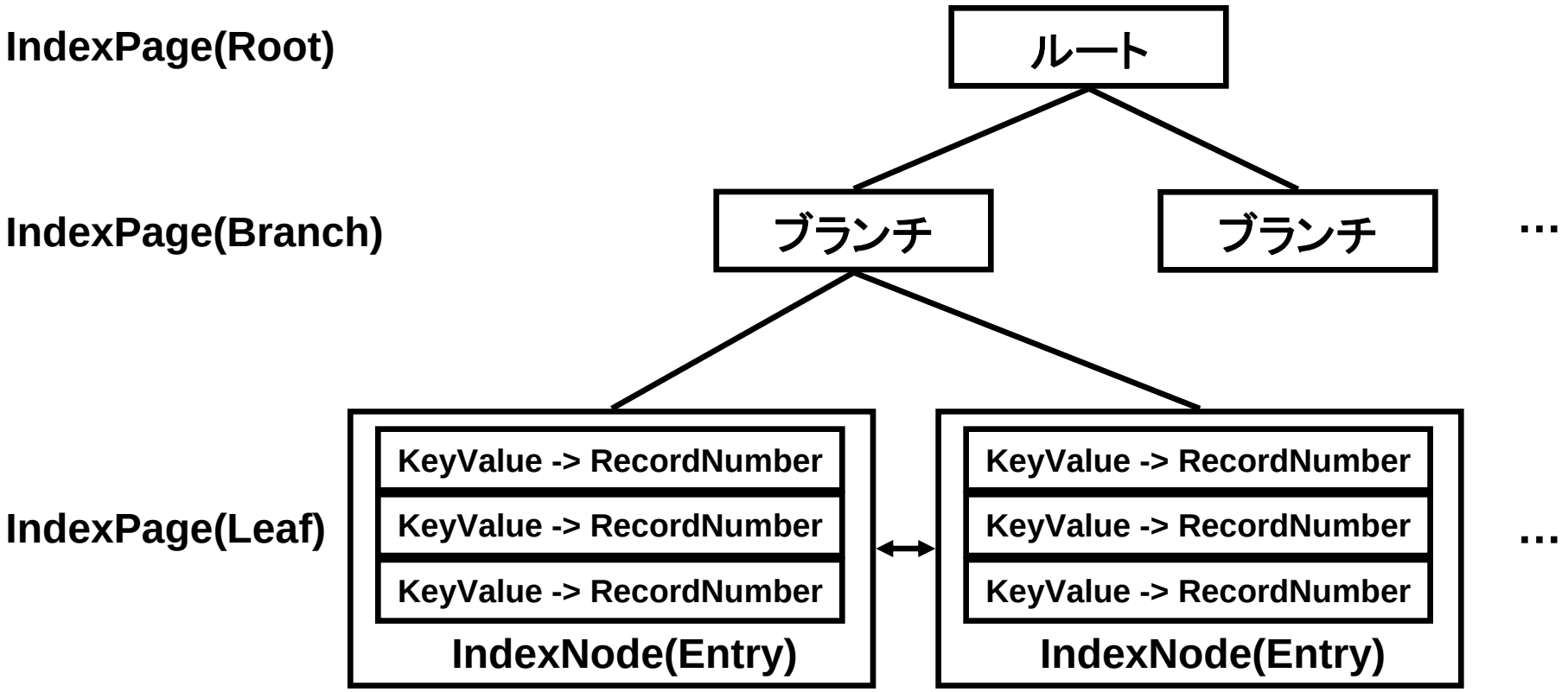
- Oracleの表領域と類似
- 任意のファイルを割り当てられるため、I/O分散が可能
- Falconのほかに、5.1以降でMySQL Cluster (NDB) がサポート

```
mysql> CREATE TABLESPACE space1 ADD DATAFILE 'file1'
ENGINE=Falcon;

mysql> CREATE TABLE t1 (c1 BIGINT PRIMARY KEY, c2 INT,
c3 VARCHAR(100)) TABLESPACE space1 ENGINE=Falcon;

mysql> select * from information_schema.falcon_tables;
+-------------+------------+------------+
| SCHEMA_NAME | TABLE_NAME | TABLESPACE |
+-------------+------------+------------+
| TEST        | T1         | SPACE1     |
+-------------+------------+------------+
1 row in set (0.00 sec)
```

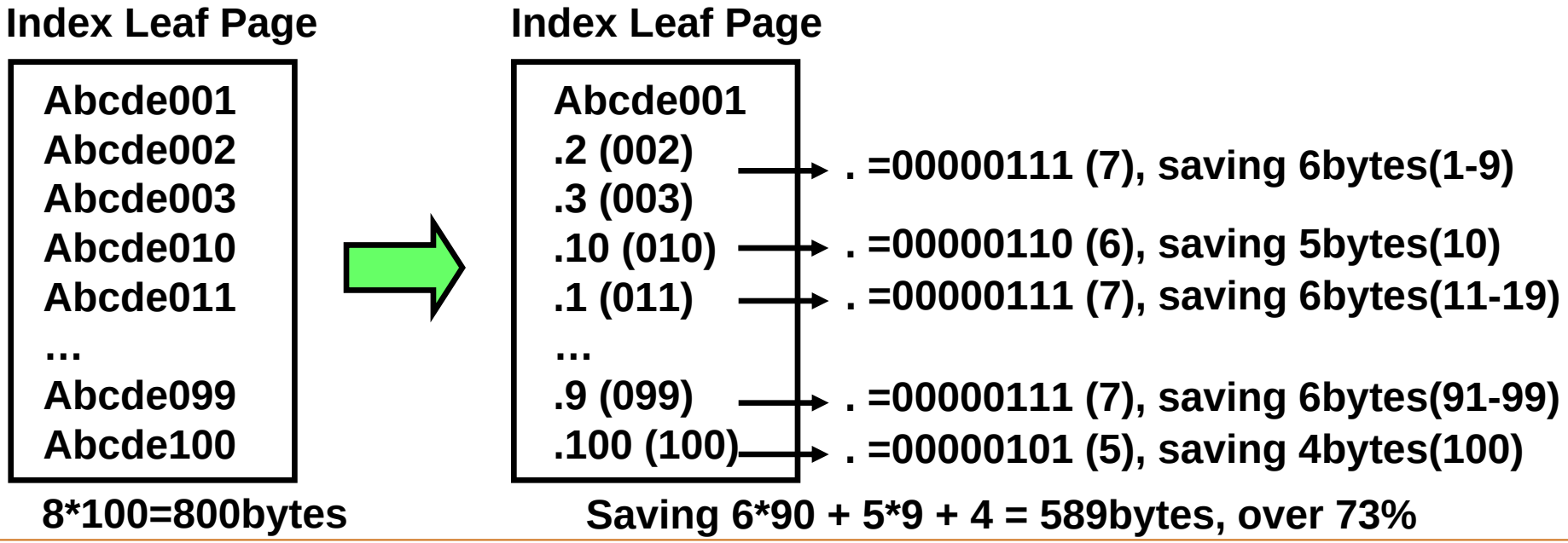
インデックス構成



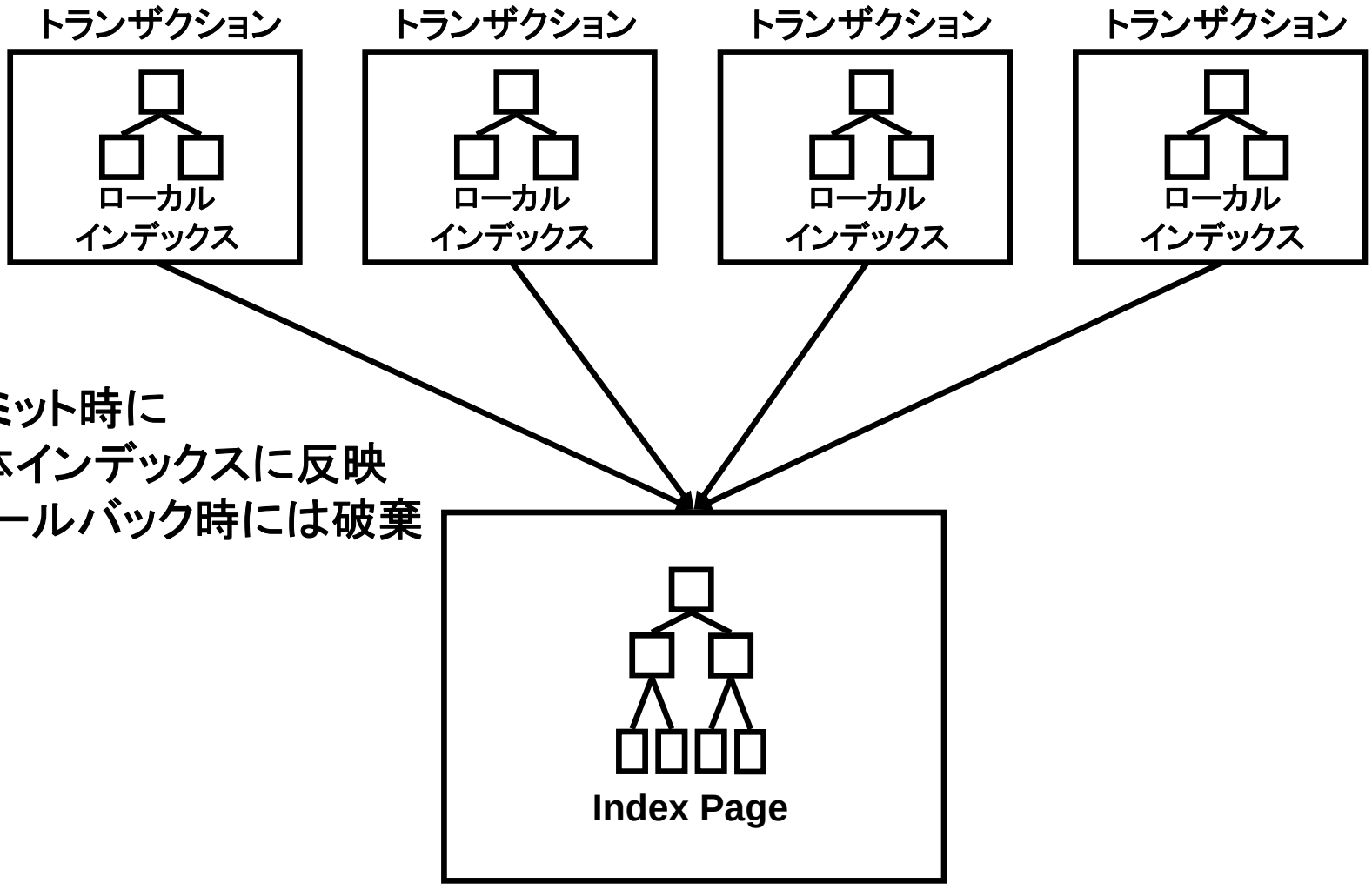
- クラスタ索引/セカンダリ索引という区別はない。全部この形
- RecordNumber(レコード番号)は現在4バイト固定(安定版までに変更予定)。
- ゆえにインデックスサイズは(クラスタ索引に比べて)さほど大きくならない
- ページはI/Oの最小単位。サイズは可変(2KB – 32KB予定)

インデックス値の圧縮

- 接尾辞の圧縮
  - 数値型：末尾のゼロを圧縮
  - 文字列型：末尾の空白を圧縮
- 接頭辞の圧縮
  - 各ページの先頭のインデックス値は圧縮しない
  - 2番目以降のインデックス値は、開始何バイトが
    直前のインデックス値と一致するかを調べ、その分を圧縮する

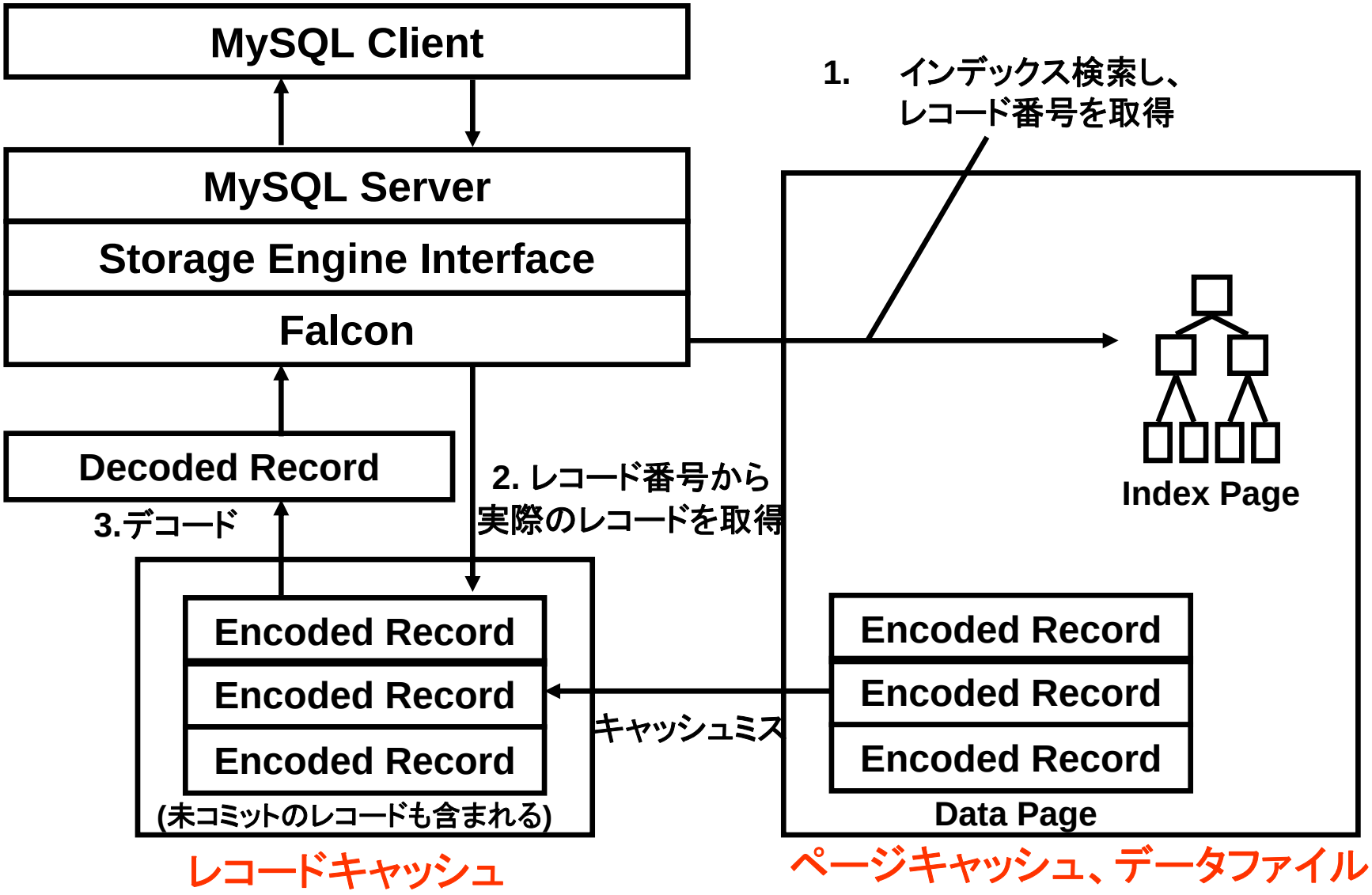


インデックスアクセラレータ



・コミット時に
本体インデックスに反映
・ロールバック時には破棄

レコードの取得


MySQL Client
MySQL Server
Storage Engine Interface
Falcon
Decoded Record
3.デコード
1. インデックス検索し、
レコード番号を取得
2. レコード番号から
実際のレコードを取得
Index Page
Encoded Record
Encoded Record
Encoded Record
(未コミットのレコードも含まれる)
キャッシュミス
Encoded Record
Encoded Record
Encoded Record
Data Page
レコードキャッシュ
ページキャッシュ、データファイル

データ型と消費サイズ

- GIS型を含む全MySQLデータ型をサポート
- 整数型を含む全データ型が可変長として扱われる
- 消費サイズはデータ型ではなく、実際の値に応じて変わる
  - 例： 値「5」の消費サイズは、BIGINT型であろうとTINYINT型であろうと同じ

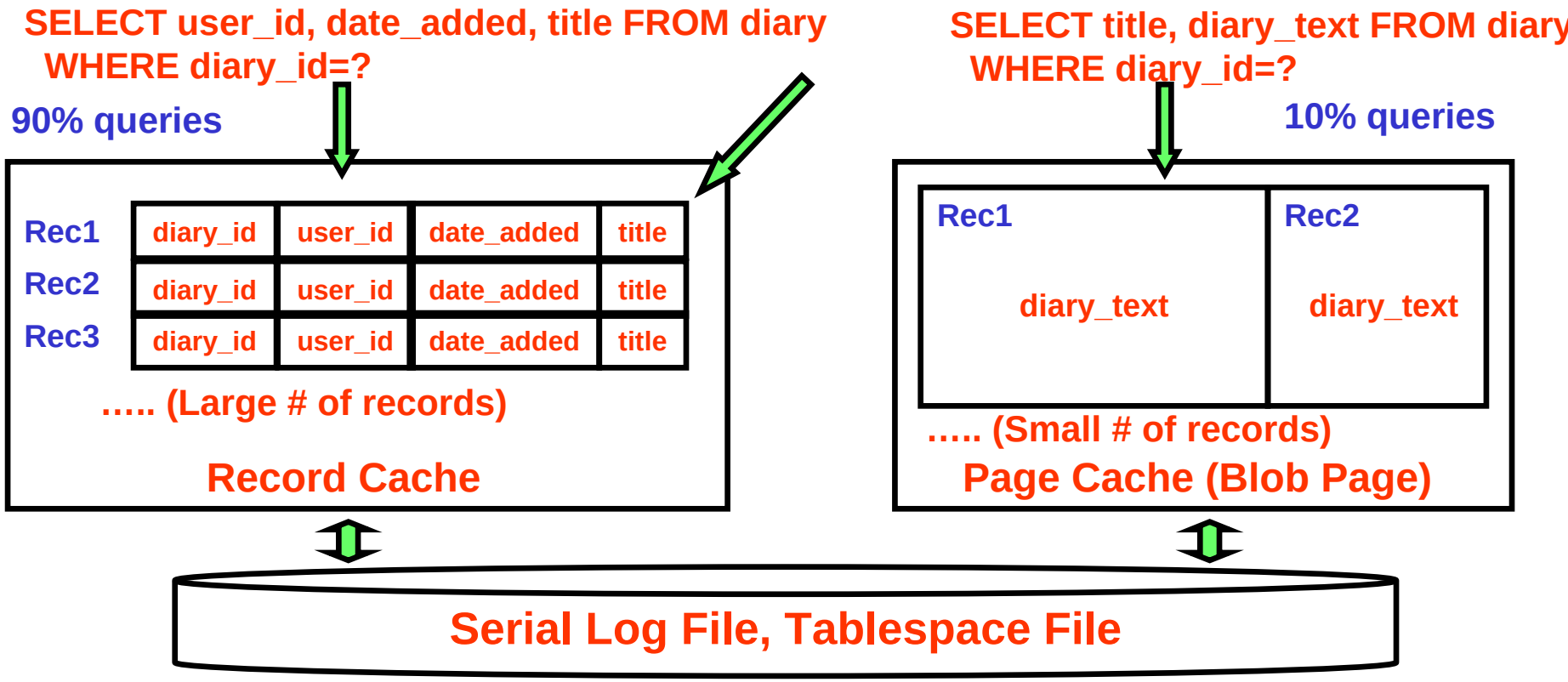
1レコードあたりのメモリ消費量(現在の動作)

sizeof(Record) (現状40バイト)
+ カラムの数 * 2バイト
+ エンコードされた値(下記参照)
+ オーバーヘッド (通常2バイト)

データ型と値	レコードキャッシュ上で消費されるバイト数
1 INT, value(1)	45 (42 + 2+1)
1 INT, value(1000)	47 (42 + 2+1+2)
1 BIGINT, value(1)	45 (42 + 2+1)
1 BIGINT, value(1000)	47 (42 + 2+1+2)
1 BIGINT, value(100,000,000,000)	50 (42 + 2+1+5)
2 INT, value(1,1)	48 (42 + 2+1 + 2+1)
3 INT, value(1,1,1)	51 (42 + 2+1 + 2+1 + 2+1)
1 VARCHAR, value(‘’)	45 (42 +2+1)
1 VARCHAR, value(null)	45 (42 +2+1)
1 VARCHAR, value(‘a’)	46 (42 +2+1+1)
1 VARCHAR, value(‘ABCDEFGHIJ’)	55 (42 +2+1+10)

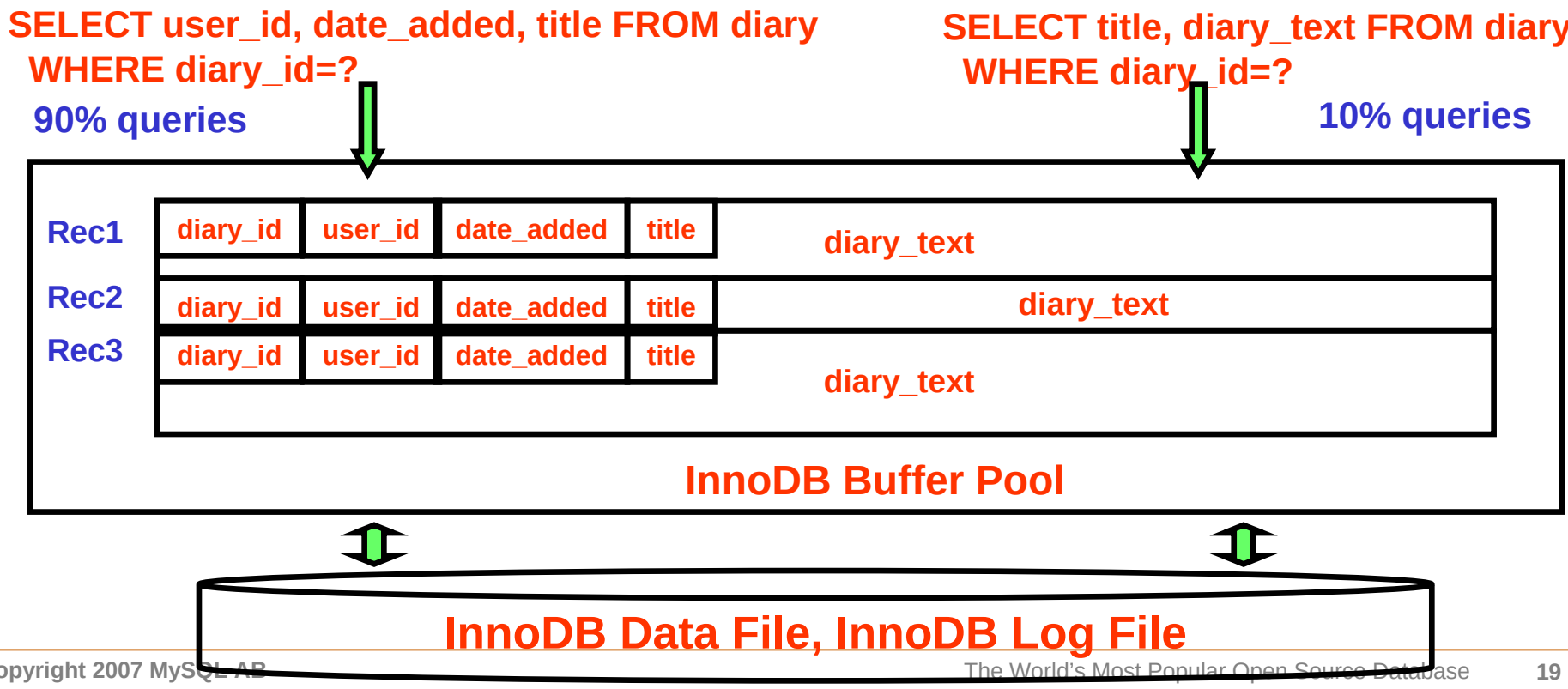
FalconはTEXT/BLOB型を別メモリ領域に管理する

diary_id (PK)	user_id (INDEX)	date_added (DATETIME, INDEX)	title (VARCHAR(100), INDEX(10)	diary_text (TEXT)
1	5544321	2007/06/13 21:10:14	Talking about Falcon	…….(2000bytes)
2	5544321	2007/06/13 22:13:34	UEFA Champions League	…….(700bytes)
3	2345	2007/06/14 01:12:23	Red Sox vs NYY	…….(3000bytes)



TEXT/BLOB (InnoDBの場合)

diary_id (PK)	user_id (INDEX)	date_added (DATETIME, INDEX)	title (VARCHAR(100), INDEX(10)	diary_text (TEXT)
1	5544321	2007/06/13 21:10:14	Talking about Falcon	…….(2000bytes)
2	5544321	2007/06/13 22:13:34	UEFA Champions League	…….(700bytes)
3	2345	2007/06/14 01:12:23	Red Sox vs NYY	…….(3000bytes)



AUTO_INCREMENTの動作の違い

```
mysql> CREATE TABLE tbl1(id INT AUTO_INCREMENT PRIMARY KEY);
mysql> INSERT INTO tbl1(id) VALUES(null);
mysql> INSERT INTO tbl1(id) VALUES(null);
mysql> INSERT INTO tbl1(id) VALUES(null);
mysql> INSERT INTO tbl1(id) VALUES(100);
mysql> DELETE FROM tbl1 WHERE id=100;
mysql> SELECT id FROM tbl1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
+----+
```

mysqldの再起動

```
mysql> INSERT INTO tbl1(id) VALUES(null);
```

```
mysql> SELECT id FROM tbl1;
+-----+
| id  |
+-----+
|   1 |
|   2 |
|   3 |
| 101 |
+-----+
```

Falcon,MyISAM

```
mysql> SELECT id FROM tbl1;
+-----+
| id  |
+-----+
|   1 |
|   2 |
|   3 |
|   4 |
+-----+
```

InnoDB

トランザクション、ロック、MVCC

- 行レベルロッキング
- SELECTはロックをかけない。更新系処理と競合しない
  - ブロックするSELECT FOR UPDATEもサポート
- AUTO_INCREMENTの割当にテーブルロックをかけない

- 分離レベルとしてRead Committed, Repeatable Readをサポート
  - Serializableもサポート予定
- ロックエスカレーションは発生しない
- デッドロックの検知は自動で行う

- ロックをかけないカラム、インデックスの追加/削除
  - MySQL側でストレージエンジンのインターフェースとして6.0から搭載予定

- ロストアップデートの自動検知
- Next Keyロッキングは発生しない

ロストアップデートの自動検知

```
CREATE TABLE tbl2 (id INTEGER AUTO_INCREMENT PRIMARY KEY,
value INTEGER) ENGINE=Falcon/InnoDB;
INSERT INTO tbl2 VALUES(1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,7),(8,8),(9,9),(10,10);
```

	Transaction 1	Transaction 2
1)	START TRANSACTION;	START TRANSACTION;
2)	SELECT value FROM tbl2 WHERE id=1;	
3)	UPDATE tbl2 SET value=100 WHERE id=1;	
4)		SELECT value FROM tbl2 WHERE id=1;
5)		UPDATE tbl2 SET value=10 WHERE id=1;
6)	COMMIT;	
7)		COMMIT;

```
mysql> UPDATE tbl2 SET value=10 WHERE id=1;
ERROR 1020 (HY000): Record has changed since last read in table 'tbl2‘ # at 6)
```

最終的な値
InnoDB: 10
Falcon: 100

Falconの動作を、設定でInnoDBと同じにすることもできる
(デフォルトで同一にする方向)

Next-Key Locking (InnoDBのロック制御)

- InnoDBログファイルとバイナリログの不整合を防ぐために必要な実装

- INSERT INTO t1 … SELECT … FROM t2
  - t2に対して共有ロックをかける

- UPDATE t1 SET xx WHERE non_index_column=x;
  - スキャンしたレコード全体(この場合はt1全体)に対して排他ロックをかける

- UPDATE t1 SET xx WHERE non_unique_index_column=x;
  - 当該インデックスと、その前後に対して排他ロックをかける

- セッション1：INSERT INTO t1 SELECT * FROM t2
- セッション2：INSERT INTO t2 VALUES(…)
  - セッション2がセッション1よりも前に終わったとする
  - バイナリログの中身は、以下の順番になり逆転する
    - INSERT INTO t2 VALUES(…)
    - INSERT INTO t1 SELECT * FROM t2
- Next-Key Lockingは、これを防ぐための実装。
- 副作用として、同時実行性が下がる

FalconではNext Key Lockingは発生しない

- INSERT INTO t1 SELECT * FROM large_table;
- UPDATE t1 SET xx WHERE no_index_column=x;

T1	T2	T3
create table t1 (c1 int auto_increment primary key , c2 int , index(c2)) engine=Falcon/InnoDB; insert into t1 values(1,1),(2,2),(3,3);		
set autocommit=0; update t1 set c2=2000 where c2=2;		
	set autocommit=0; insert into t1 values(null,2);	
		set autocommit=0; insert into t1 values(null,6);

1	1
2	2
3	3

←InnoDBでは待たされる (next_key locking)

←InnoDBでは待たされる (AUTO_INC table locking)

InnoDBでも、5.1以降ではNext Key Lockingを無効化できる

参考資料

- 英語
  - MySQL Developer Zone: Falcon in depth
    ベータ版リリース後に公開
  - MySQL Online Manual
    http://dev.mysql.com/doc/falcon/en/index.html
  - MySQL Forge
    http://forge.mysql.com/wiki/Falcon
- 日本語
  - マイコミジャーナル:　Falcon徹底リサーチ
    http://journal.mycom.co.jp/special/2007/falcon/