Optimizing Large Databases Using InnoDB Clustered Indexes

- *Clustered Indexes*, a main optimisation technique for large OLTP databases.
- When DB cannot be fully cached in RAM.
- Eg. web application with lots of accounts.
- Description & benchmarks.

Kristian Nielsen, independent software professional.
Free Software since 1990: Wine, MySQL, McStas, and others.
MySQL AB software engineer 2005-2008.

Problem: Disk Latency

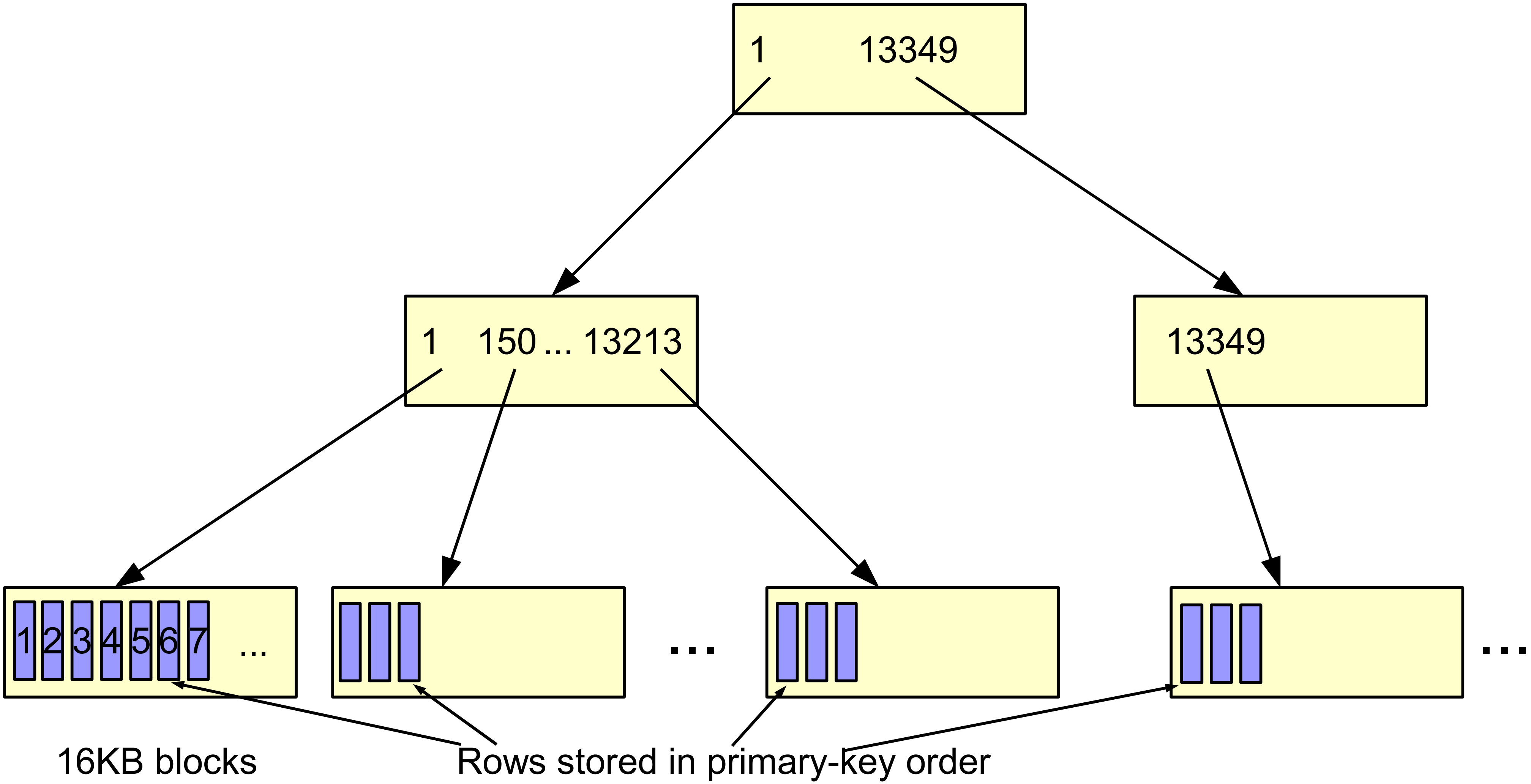
Typical HDD performance (non-SSD):

- Sequential ~100MB/sec (1M rows @ 100B). Great!
- Random access ~100 IO/sec (100 rows/sec). That's bad :-(

Possible solutions:

- DB block cache -> fails when working set > RAM.
- LOTS of disks -> works, but expensive.
- Data locality -> clustered indexes.

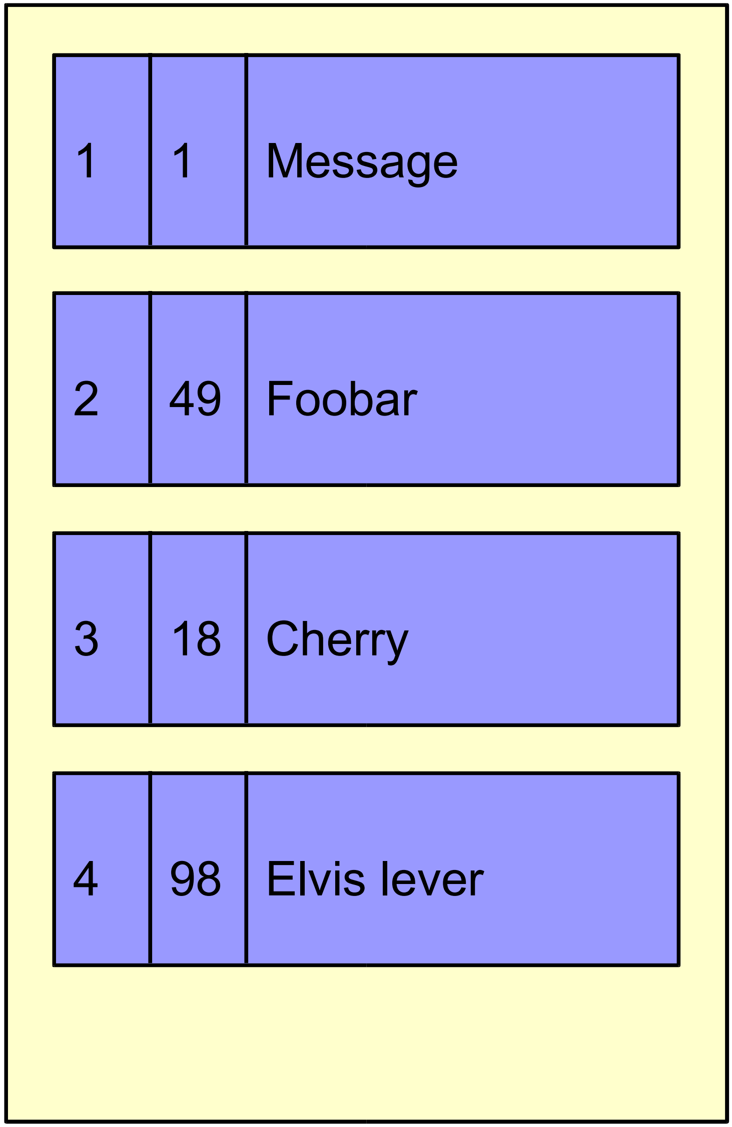
InnoDb Data Storage: BTree



Autoincrement Primary Key

```
CREATE TABLE message (
  message_id INT AUTO_INCREMENT,
  user_id INT,
  mtext VARCHAR(1000),
  PRIMARY KEY(message_id),
  KEY (user_id))
```

- Rows stored on disk in insert order.
- Insert multiple rows fast (if secondary index fits in RAM).
- Select multiple rows slow.

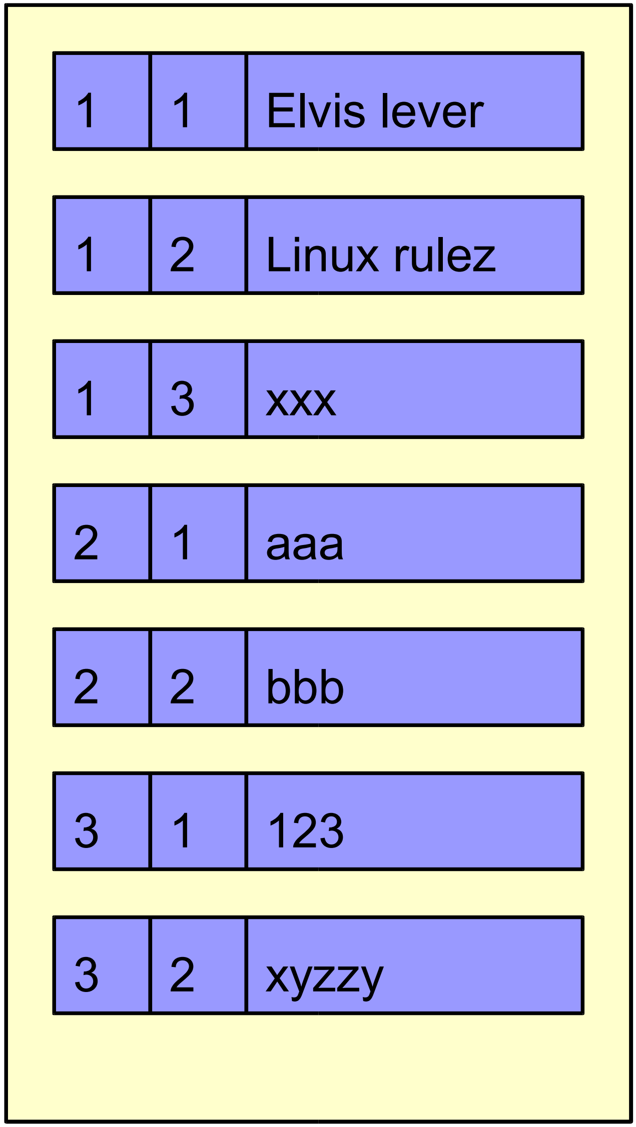


1	1	Message
2	49	Foobar
3	18	Cherry
4	98	Elvis lever

Natural Primary Key

```
CREATE TABLE message (
  user_id INT,
  message_id INT,
  mtext VARCHAR(1000),
  PRIMARY KEY(user_id, message_id)
)
```

- Selecting all messages from one account is fast (~1 I/O).
- Inserting multiple messages is slower.
- Can use AUTO_INCREMENT for message_id, or `SELECT 1+MAX(message_id)` ...



1	1	Elvis lever
1	2	Linux rulez
1	3	xxx
2	1	aaa
2	2	bbb
3	1	123
3	2	xyzzy

AUTO_INCREMENT is Evil!

- ... or at least using them blindly is.
- Often, there is a natural primary key.

Examples:

- Social networking (user_id, other_user_id).
- Forum, (thread_id, message_id).
- Wikipedia revisions (rev_page, rev_id).

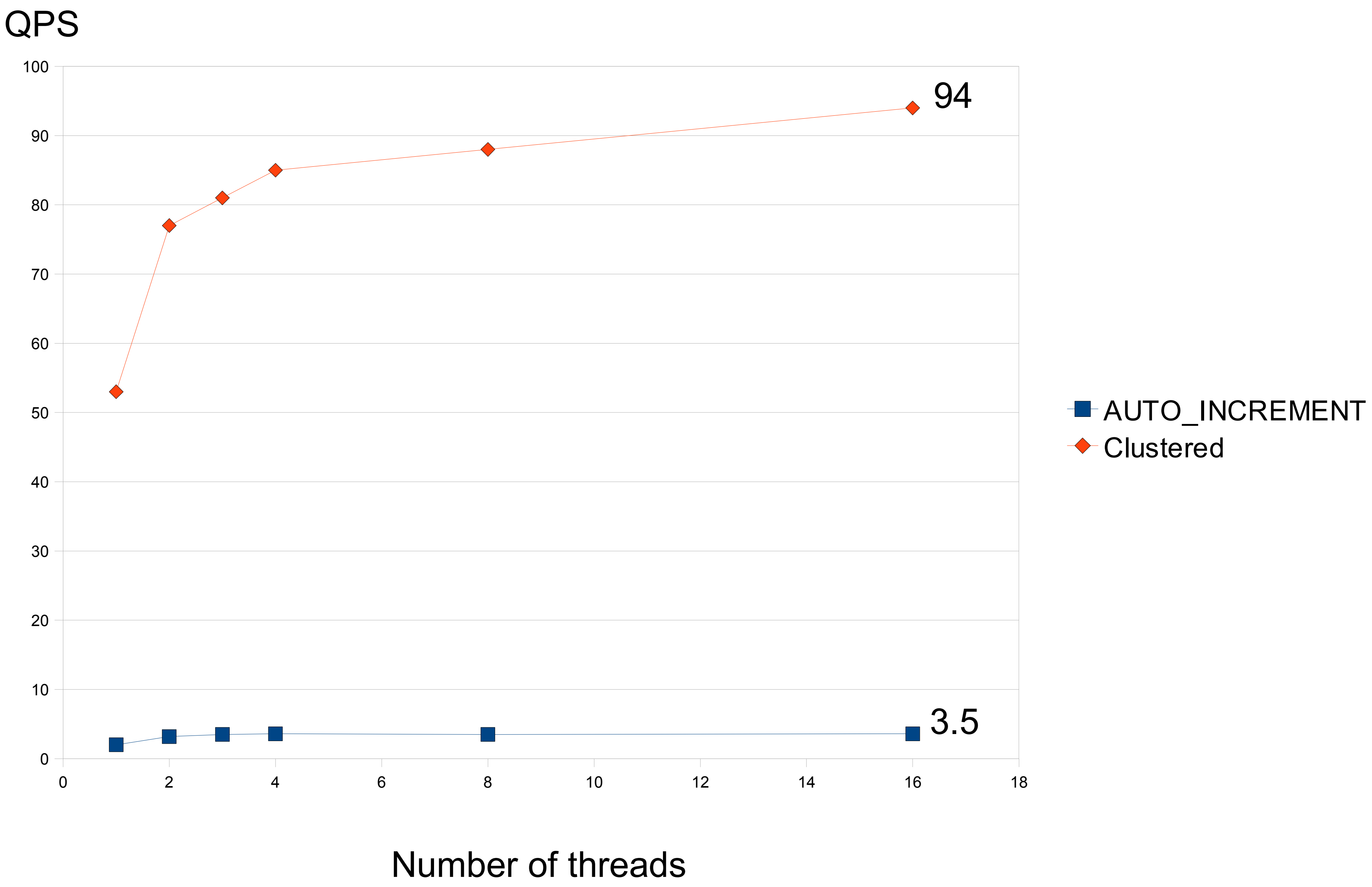
Benchmark

- Message database.
- 1M users, 50M messages, ~300Byte row size.
- ~20GByte database, 3GByte buffer pool.
- 2 x Western Digital Raptor 10k rpm.
- Striped tablespace.
- Intel Core 2 Quad 2.4GHz, 4GByte ram.

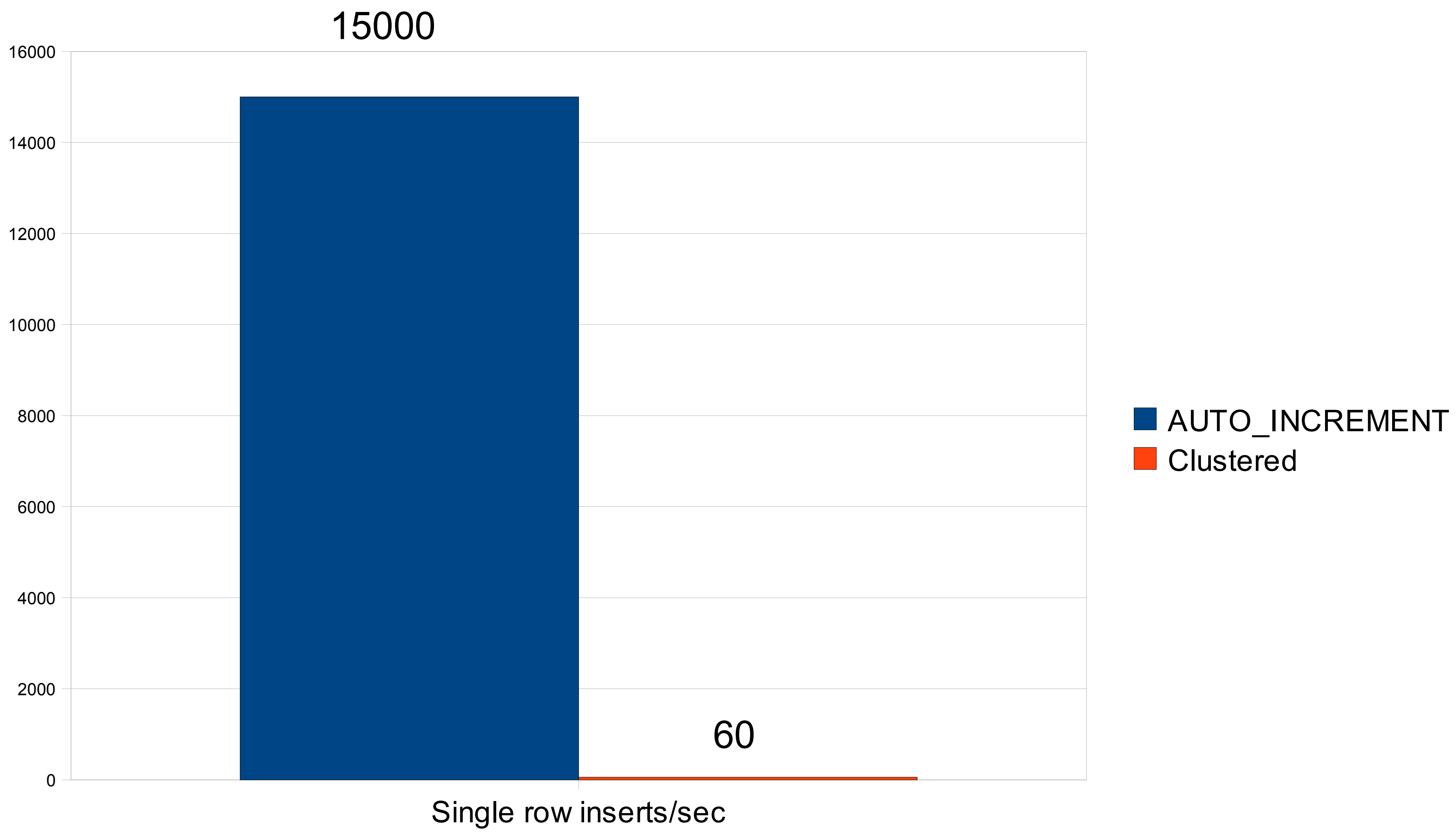
```
CREATE TABLE message_auto (
   message_id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
   user_id INT NOT NULL,
   subject VARCHAR(256),
   mtext VARCHAR(4000),
   create_dt TIMESTAMP NOT NULL,
   KEY(user_id)
) ENGINE=innodb

CREATE TABLE message_cluster (
   user_id INT NOT NULL,
   message_id INT NOT NULL,
   subject VARCHAR(256),
   mtext VARCHAR(4000),
   create_dt TIMESTAMP NOT NULL,
   PRIMARY KEY(user_id, message_id)
) ENGINE=innodb
```

AUTO_INCREMENT vs. Clustered

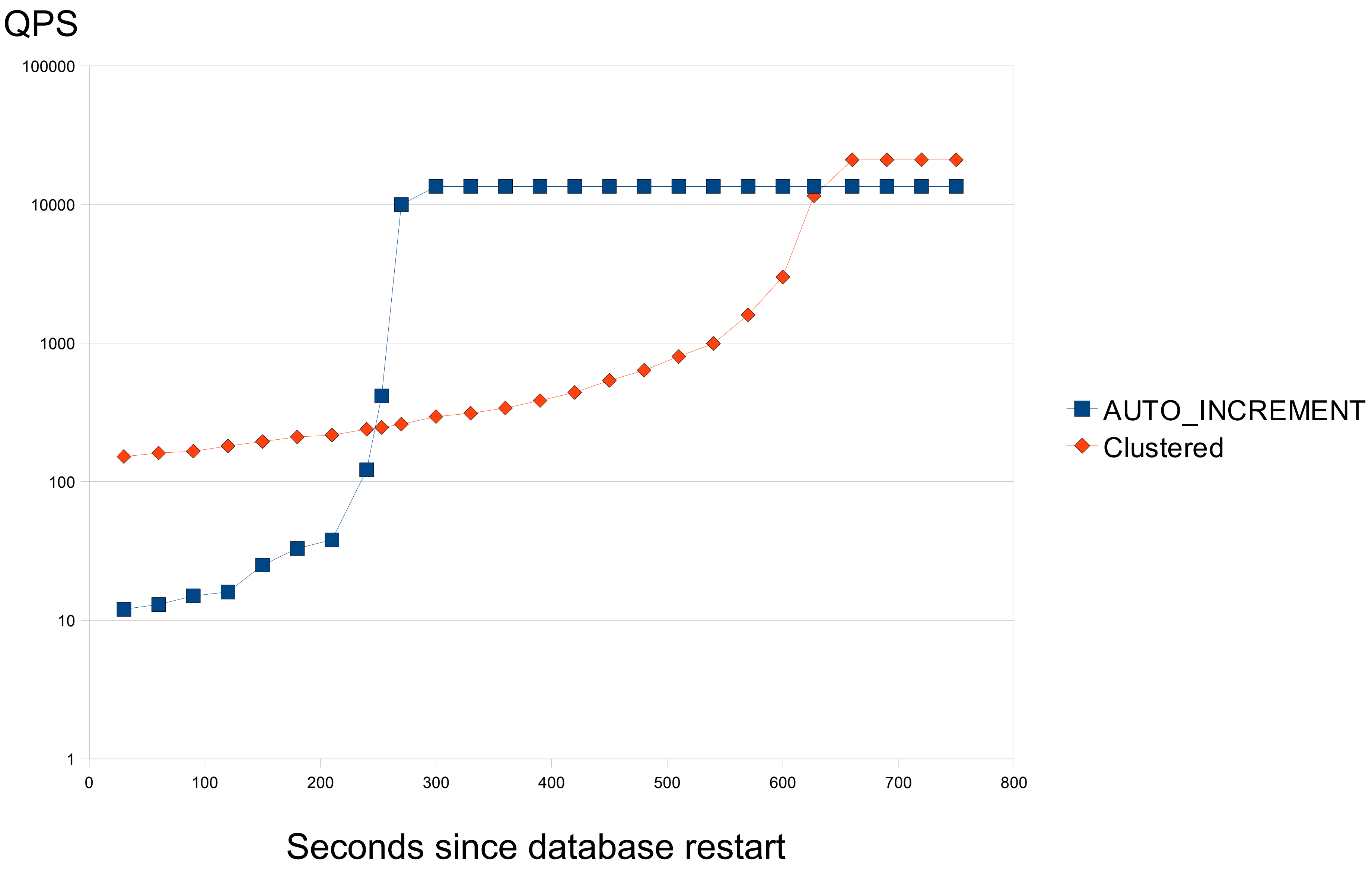
```
SELECT message_id,subj,mtext,create_dt FROM message WHERE user_id=? LIMIT 50
```



BUT! Insert slower with Clustered



Cold Start


QPS
100000
10000
1000
100
10
1
0
100
200
300
400
500
600
700
800
AUTO_INCREMENT
Clustered
Seconds since database restart

Covering Indexes: Poor Mans Clustered Indexes

- Index that includes all columns in the `SELECT`.
- `SELECT` can use index (only) for queries, similar to InnoDB primary key.
- Inserts and deletes are penalised.
- Possible alternative if clustered index cannot be used (Legacy application, MyISAM/Maria, ...).
- Case story: CBB Mobilsvar.

Conclusions

- Random access I/O is a performance killer for high-trafic bigger-than-mem OLTP (or cold start).
- InnoDB clusters on the primary key -> USE it, don't blindly add an AUTO_INCREMENT key.
- Order-of-magnitude speedups.
- Not a magic silver bullet.