

An Introduction to InnoDB Internals

Date, conference, place:

February 16th,
Percona Live,
San Francisco, CA

Presenter:

Justin Swanhart Percona

Introduction

- ★ Justin Swanhart
- ★ Principal Consultant – Percona, Inc.
 - ♦ Also a trainer
- ★ Thanks to Morgan Tocker, Peter Zaitsev, Baron Schwartz and the other Percona employees
- ★ Percona Live, February 16th, 2011

The InnoDB plugin

The InnoDB pluggable SE

- ★ Until recently(MySQL 5.1) the InnoDB version has been tied closely to the MySQL release.
- ★ MySQL 5.1's pluggable storage engine API -
 - ◆ Developers have increased freedom to make improvements independent of MySQL.
- ★ The InnoDB plugin is the default InnoDB engine in MySQL 5.5
- ★ The InnoDB storage engine is the default in 5.5

The plugin also brings

- ★ New features!
 - ♦ Improved CPU scalability, fast index creation, buffer pool tablescan resistance, fast crash recovery, multiple buffer pools (5.5 only)...
- ★ The 5.5 plugin is the main focus of development.
- ★ The 5.5 plugin is version 1.1.x The 5.1 plugin is version 1.0.x

InnoDB / MySQL Relationship

The two systems communicate mostly on a row-based relationship, a.k.a:

- ♦ The storage engine API.
- ♦ The two systems communicate mostly on a row-based relationship, a.k.a:
- ♦ Some functions like foreign key constraint parsing and enforcement are delegated to the storage engine

Today's Focus

- ★ We're concerned mostly about what happens below that storage engine API.
 - ♦ aka "**What's Inside InnoDB**".
- ★ General InnoDB plugin
 - ♦ I may mention XtraDB but it is not the focus
 - ♦ These topics are geared toward 5.5 when there are differences in the newer version
 - ♦ Almost everything applies to MySQL 5.0 and greater

Today's Focus (cont.)

- ★ Remember that there are other moving pieces sitting above InnoDB that we will not be talking about.
- ★ That means that if by slip of the tongue we refer to *log files* - you can assume we mean ib_logfile0 and ib_logfile1, **and not** the binary log.

High Level Features

- ★ InnoDB has numerous modern RDBMS features that make it possible to meet the operational requirements of large scale database applications
- ★ Crash safe (ACID compliance)
- ★ MVCC – readers don't block writers
- ★ Index only scans
- ★ Change buffering

High Level Features (cont)

- ★ Caches indexes and data
- ★ Clustered index
- ★ Compression
- ★ Hot backup

Visibility into the layer below

- ★ There are very few diagnostic commands to know.
 - ♦ `SHOW ENGINE INNODB STATUS;`
 - ♦ `SHOW ENGINE INNODB MUTEX;`
 - ♦ `information_schema` tables (5.1 plugin).
 - ♦ slow query log (Percona Server)

SHOW ENGINE INNODB STATUS

- ★ This is the command that will likely be the most useful.
 - ♦ Lets look at some of the basics now.
 - ♦ We will cover diagnostics *if we have time*

InnoDB status (cont.)

```
mysql> SHOW ENGINE INNODB STATUS\G***** 1.
row *****      Type: InnoDB  Name: Status:
=====100618 16:25:26 INNODB
MONITOR OUTPUT=====
averages calculated from the last 4 seconds
```

Make sure the average you
are reading is at least 20-
30 seconds.

InnoDB status (cont.)

★

```
-----
LATEST DETECTED DEADLOCK
-----
060717  4:16:48
*** (1) TRANSACTION:
TRANSACTION 0 42313619, ACTIVE 49 sec, process no 10099, OS thread id 3771312
starting index read
mysql tables in use 1, locked 1
LOCK WAIT 3 lock struct(s), heap size 320
MySQL thread id 30898, query id 100626 localhost root Updating
update iz set pad='a' where i=2
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 16403 n bits 72 index `PRIMARY` of table `test/iz`
trx id 0 42313619 lock_mode X locks rec but not gap waiting
Record lock, heap no 5 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 4; hex 80000002; asc      ;; 1: len 6; hex 00000285a78f; asc      ;; 2: len
7; hex 00000040150110; asc    @    ;; 3: len 10; hex 61202020202020202020; asc a
;;
```

The order of each section changes between versions. Sections may only appear if problems are encountered - such as deadlocks and foreign key errors.

InnoDB status (cont.)

★

```
..
-----
TRANSACTIONS
-----
Trx id counter 0 80157601
Purge done for trx's n:o <0 80154573 undo n:o <0 0
History list length 6
Total number of lock structs in row lock hash table 0
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0 0, not started, process no 3396, OS thread id 1152440672
MySQL thread id 8080, query id 728900 localhost root
show innodb status
---TRANSACTION 0 80157600, ACTIVE 4 sec, process no 3396, OS thread id 1148250464,
thread declared inside InnoDB 442
mysql tables in use 1, locked 0
MySQL thread id 8079, query id 728899 localhost root Sending data
select sql_calc_found_rows * from b limit 5
Trx read view will not see trx with id>= 0 80157601, sees <0 80157597
---TRANSACTION 0 80157599, ACTIVE 5 sec, process no 3396, OS thread id 1150142816
fetching rows, thread
declared inside InnoDB 166
mysql tables in use 1, locked 0
MySQL thread id 8078, query id 728898 localhost root Sending data
select sql_calc_found_rows * from b limit 5
Trx read view will not see trx with id>= 0 80157600, sees <0 80157596
```

Whenever you see two numbers together, it may actually be one number. i.e. 0 80157601 is two 32-bit integers, where 0 will increment when 80157601 wraps.

InnoDB status (cont.)

★

```
..
-----
TRANSACTIONS
-----
Trx id counter 0 80157601
Purge done for trx's n:o <0 80154573 undo n:o <0 0
History list length 6
Total number of lock structs in row lock hash table 0
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0 0, not started, process no 3396, OS thread id 1152440672
MySQL thread id 8080, query id 728900 localhost root
show innodb status
---TRANSACTION 0 80157600, ACTIVE 4 sec, process no 3396, OS thread id 1148250464,
thread declared inside InnoDB 442
mysql tables in use 1, locked 0
MySQL thread id 8079, query id 728899 localhost root Sending data
select sql_calc_found_rows * from b limit 5
Trx read view will not see trx with id>= 0 80157601, sees <0 80157597
---TRANSACTION 0 80157599, ACTIVE 5 sec, process no 3396, OS thread id 1150142816
fetching rows, thread
declared inside InnoDB 166
mysql tables in use 1, locked 0
MySQL thread id 8078, query id 728898 localhost root Sending data
select sql_calc_found_rows * from b limit 5
Trx read view will not see trx with id>= 0 80157600, sees <0 80157596
```

This is the corresponding thread number inside SHOW PROCESSLIST in MySQL.

InnoDB status (cont.)

★

```
..
-----
TRANSACTIONS
-----
Trx id counter 0 80157601
Purge done for trx's n:o <0 80154573 undo n:o <0 0
History list length 6
Total number of lock structs in row lock hash table 0
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0 0, not started, process no 3396, OS thread id 1152440672
MySQL thread id 8080, query id 728900 localhost root
show innodb status
---TRANSACTION 0 80157600, ACTIVE 4 sec, process no 3396, OS thread id 1148250464,
thread declared inside InnoDB 442
mysql tables in use 1, locked 0
MySQL thread id 8079, query id 728899 localhost root Sending data
select sql_calc_found_rows * from b limit 5
Trx read view will not see trx with id>= 0 80157601, sees <0 80157597
---TRANSACTION 0 80157599, ACTIVE 5 sec, process no 3396, OS thread id 1150142816
fetching rows, thread
declared inside InnoDB 166
mysql tables in use 1, locked 0
MySQL thread id 8078, query id 728898 localhost root Sending data
select sql_calc_found_rows * from b limit 5
Trx read view will not see trx with id>= 0 80157600, sees <0 80157596
```

Some numbers are not well self-documented. For example 442 is “the number of concurrency tickets left for this thread.”

Basic Operation

First, some prerequisites.

ACID

- ★ **A**tomicity

- ♦ When you change multiple rows, all of the rows are changed as a group.

- ★ **C**onsistency

- ♦ When I read data multiple times, do I see the same data each time?

- ★ **I**solation

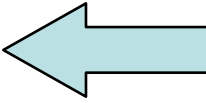
- ♦ Row level locking (prevent conflicting updates)

- ★ **D**urability

- ♦ Committed data remains committed even after a crash

“Numbers everyone should know”

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns



See: <http://www.linux-mag.com/cache/7589/1.html> and Google
<http://www.cs.cornell.edu/projects/ladis2009/talks/dean-keynote-ladis2009.pdf>

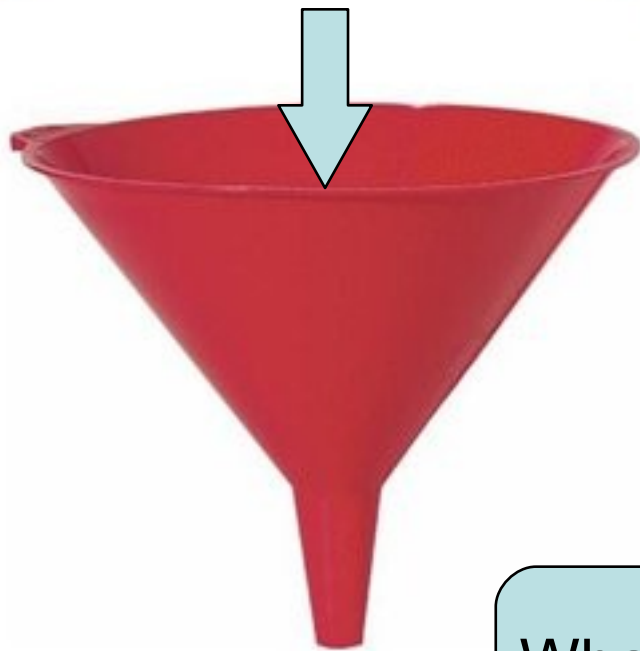
About Disks.

- ★ $10,000,000 \text{ ns} = 10\text{ms} = 100$ operations/second.
 - ♦ This is about the **average** for a 7200RPM drive.
- ★ The actual time has dramatic variation ($\sim 0.2\text{ms}$ - $\sim 20\text{ms}$)
 - ♦ The variation is because disks are mechanical.
 - ♦ We can write *much* faster sequentially than randomly.

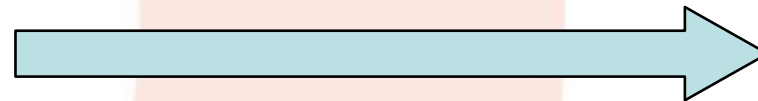
[default] Everything is buffered!

- ★ When you write to a file, here's what happens in the Operating System:

Block 9, 10, 1, 4, 200, 5.



Block 1, 4, 5, 9, 10, 200



What happens to this buffer if we loose power?

The OS provides a way!

★ \$ man fsync

Synopsis

```
#include <unistd.h>
int fsync(int fd);
int fdatasync(int fd);
```

Description

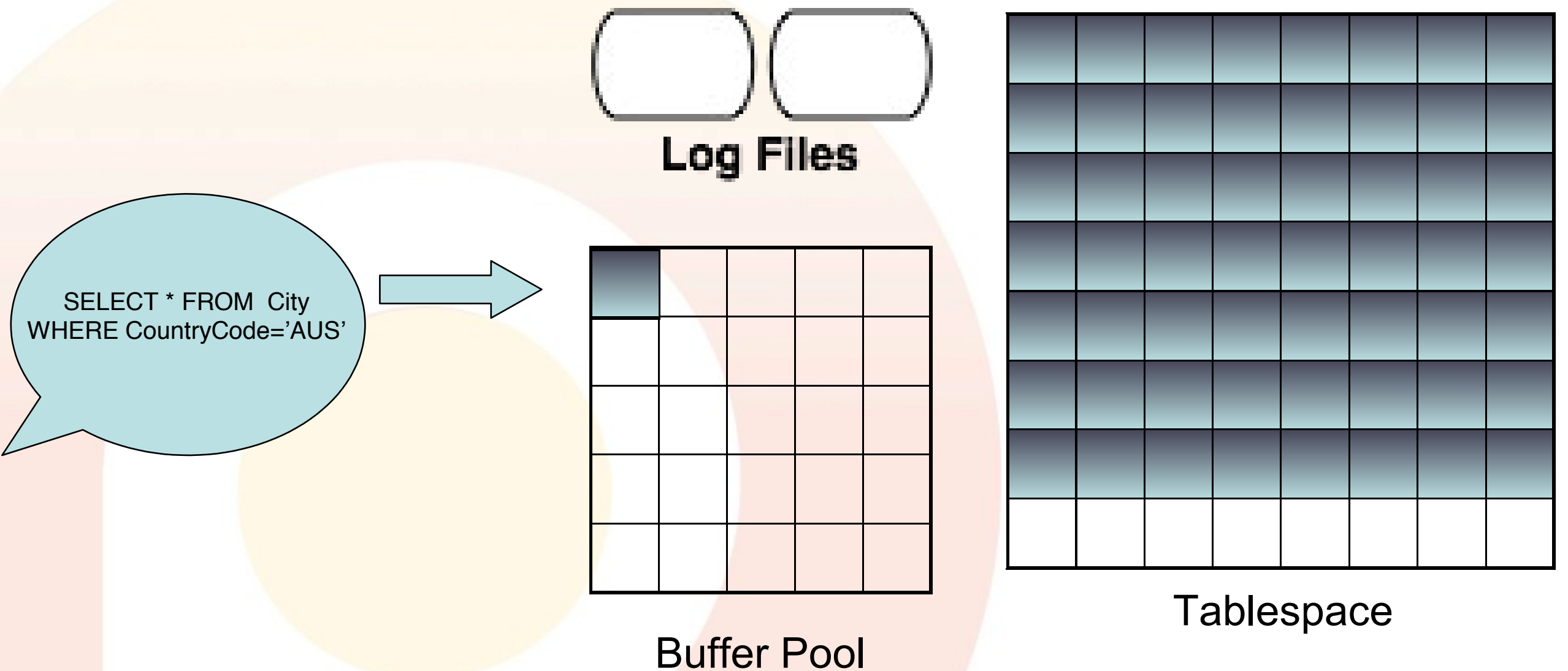
`fsync()` transfers ("flushes") all modified in-core data of (i.e., modified buffer cache pages for) the file referred to by the file descriptor `fd` to the disk device (or other permanent storage device) where that file resides. The call blocks until the device reports that the transfer has completed. It also flushes metadata information associated with the file (see `stat(2)`).

Hint: MyISAM just writes to the OS buffer and has no durability.

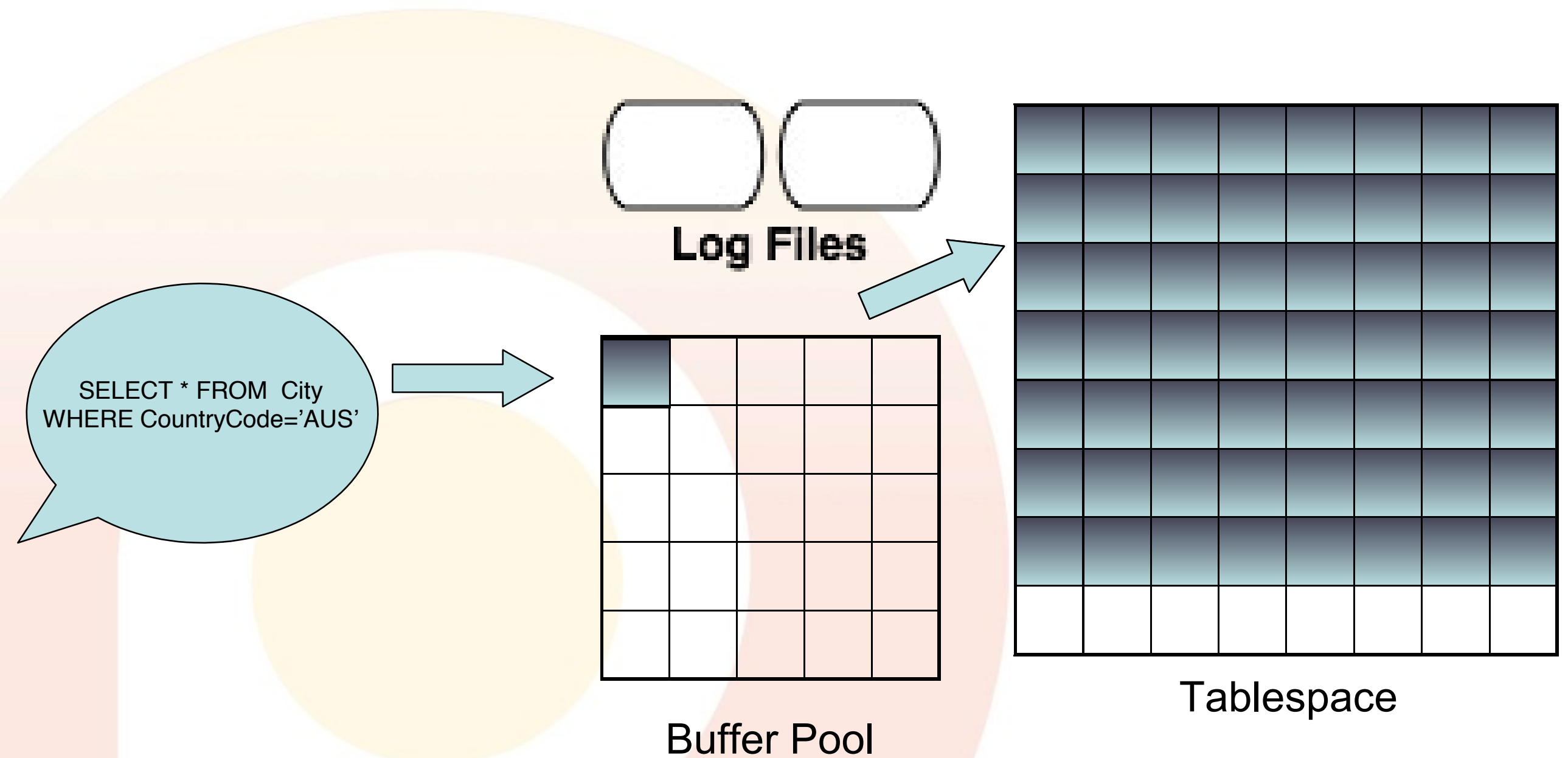
Knowing this:

- ★ InnoDB wants to try and reduce random IO.
- ★ It can not (safely) rely on the operating system's write buffering and be ACID compliant.
- ♦ .. and InnoDB algorithms have to compensate.

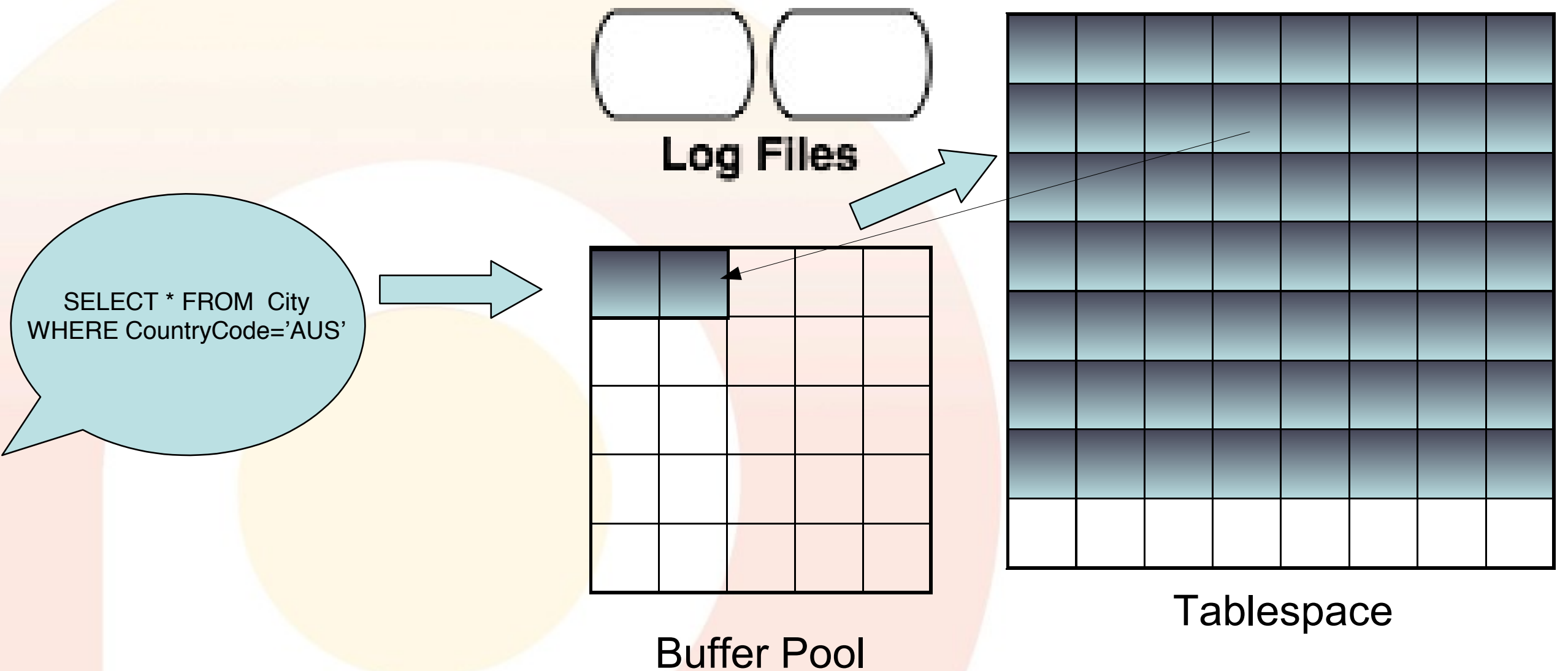
Basic Operation (High Level)



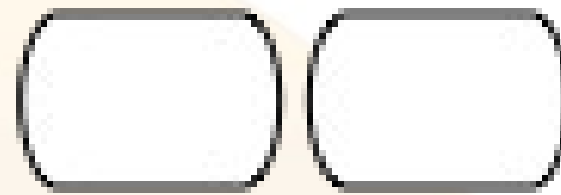
Basic Operation (High Level)



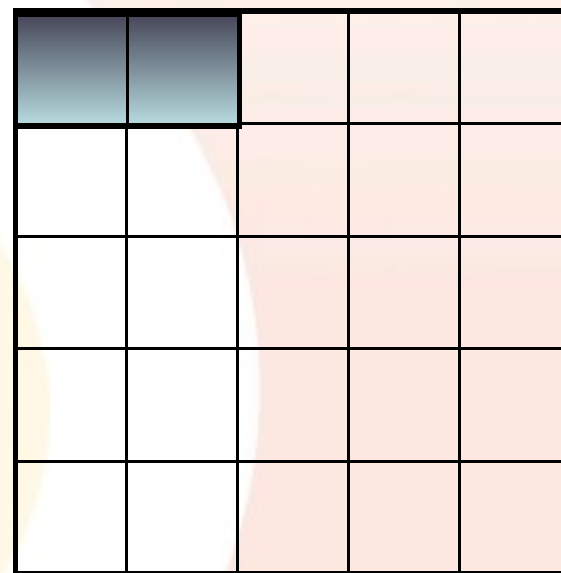
Basic Operation (High Level)



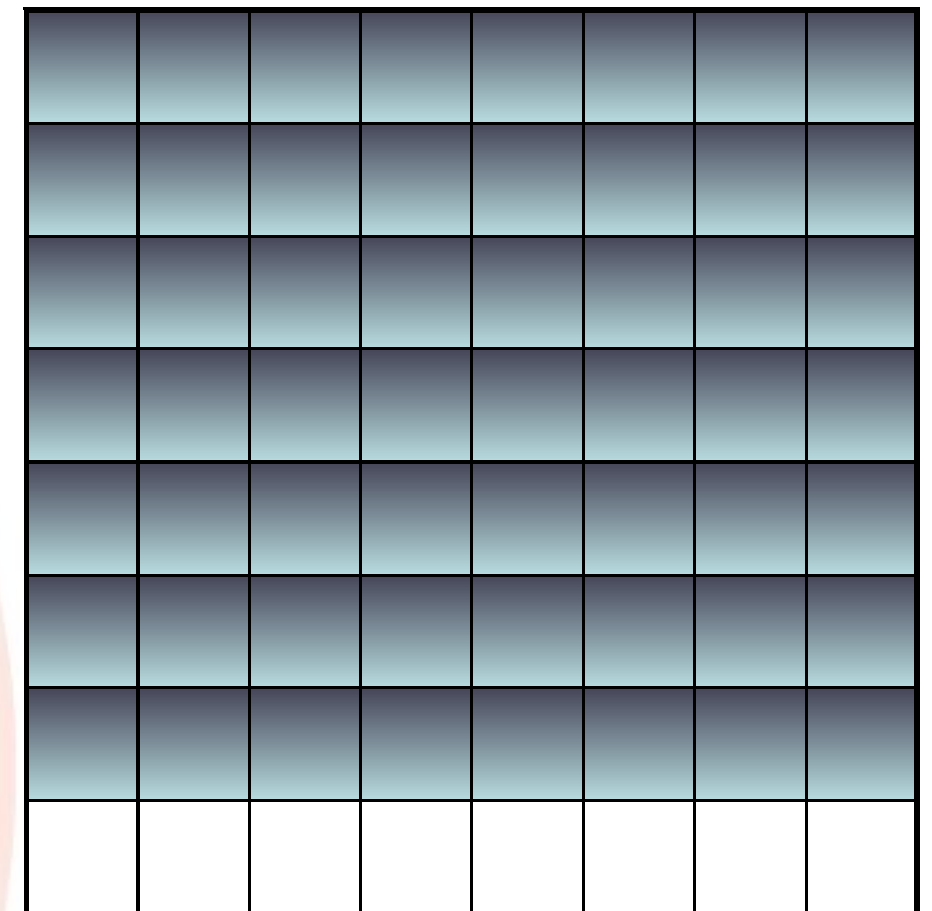
Basic Operation (High Level)



Log Files

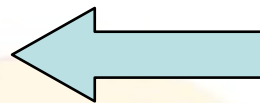


Buffer Pool



Tablespace

SELECT * FROM City
WHERE CountryCode='AUS'



Basic Operation (cont.)

UPDATE City SET
name = 'Morgansville'
WHERE name = 'Brisbane'
AND CountryCode='AUS'

01010

Log Files

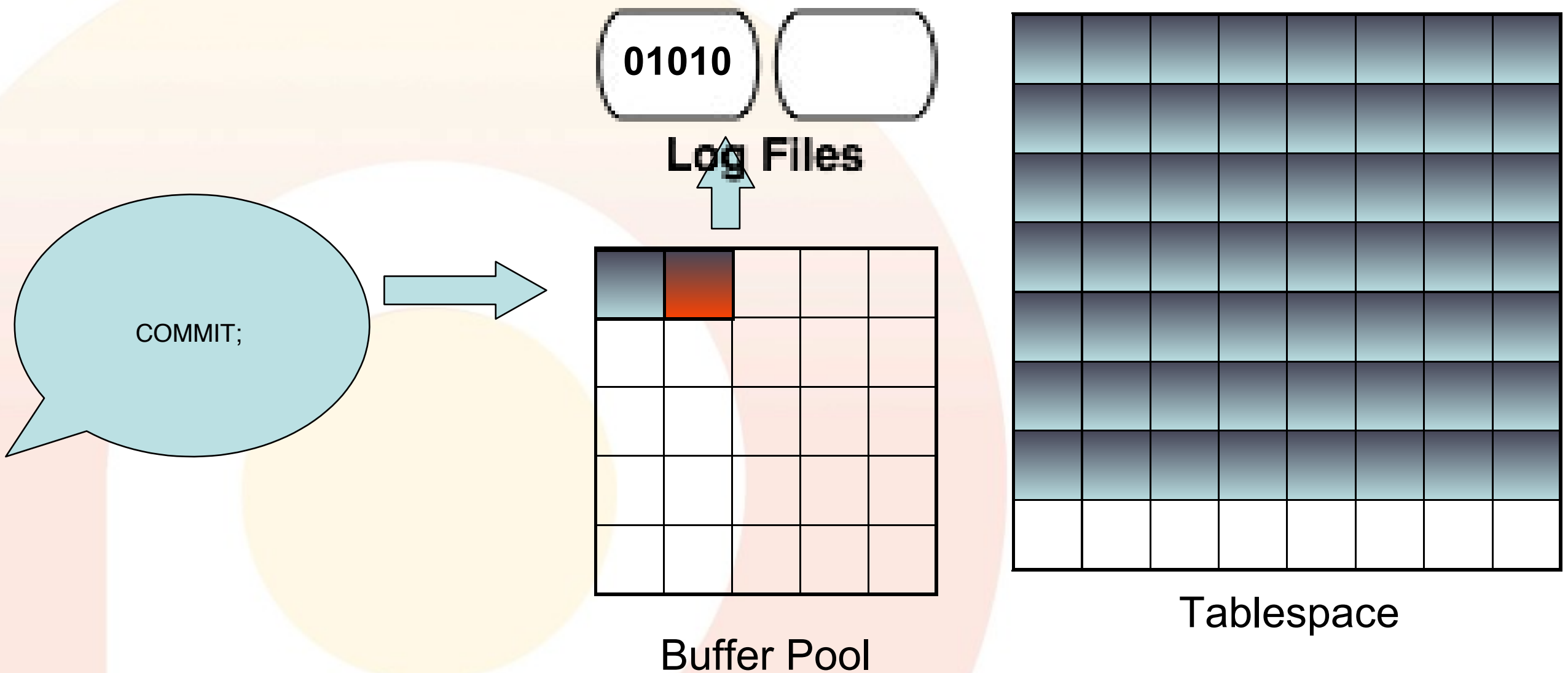
The diagram shows a Buffer Pool and Log Files. The Buffer Pool is a 6x5 grid of cells. The top-left cell is dark blue, and the cell immediately to its right is red. The other cells are light pink. An arrow points from the Buffer Pool to the Log Files. The Log Files consist of two rounded rectangular boxes. The first box contains the binary string '01010', and the second box is empty.

Buffer Pool

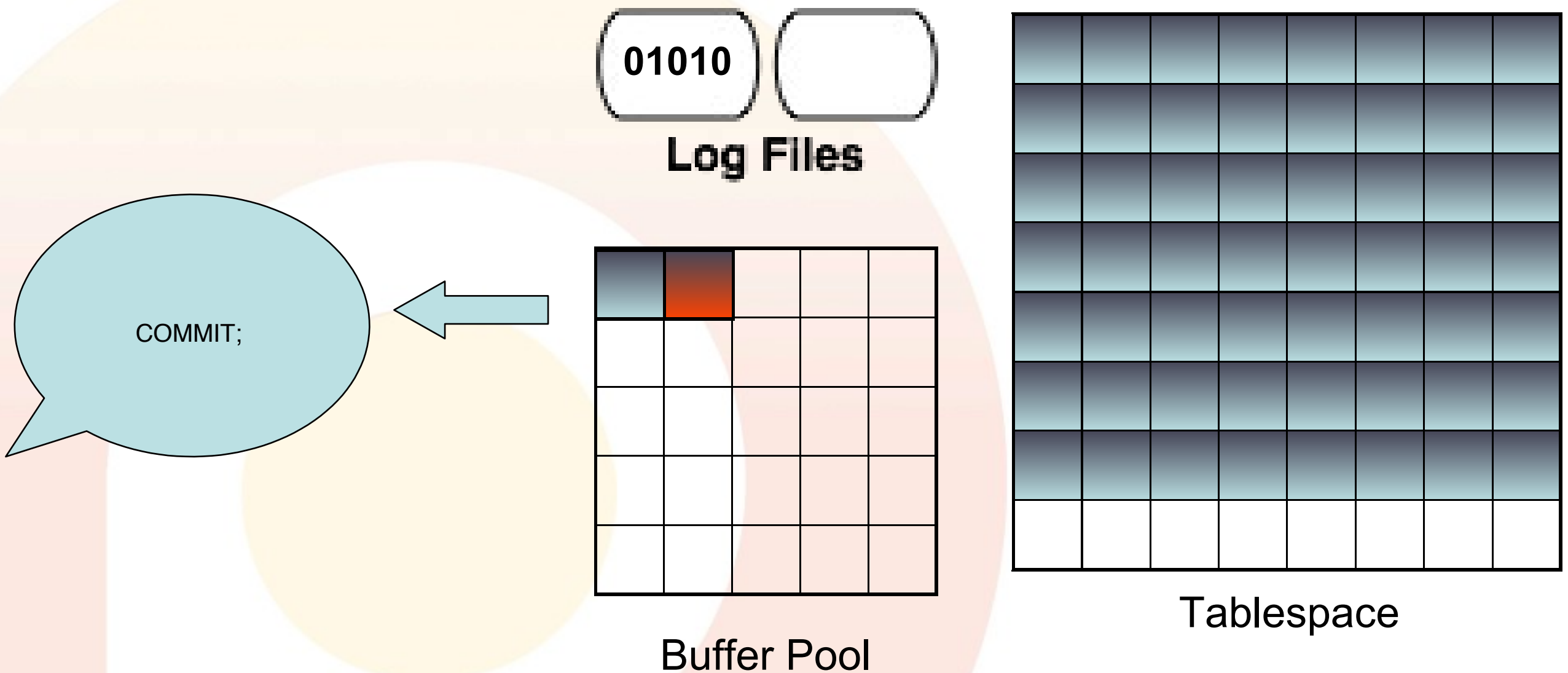
The diagram shows a Tablespace, which is a large grid of 8 rows and 8 columns of cells. The top 7 rows are dark blue, and the bottom row is light pink.

Tablespace

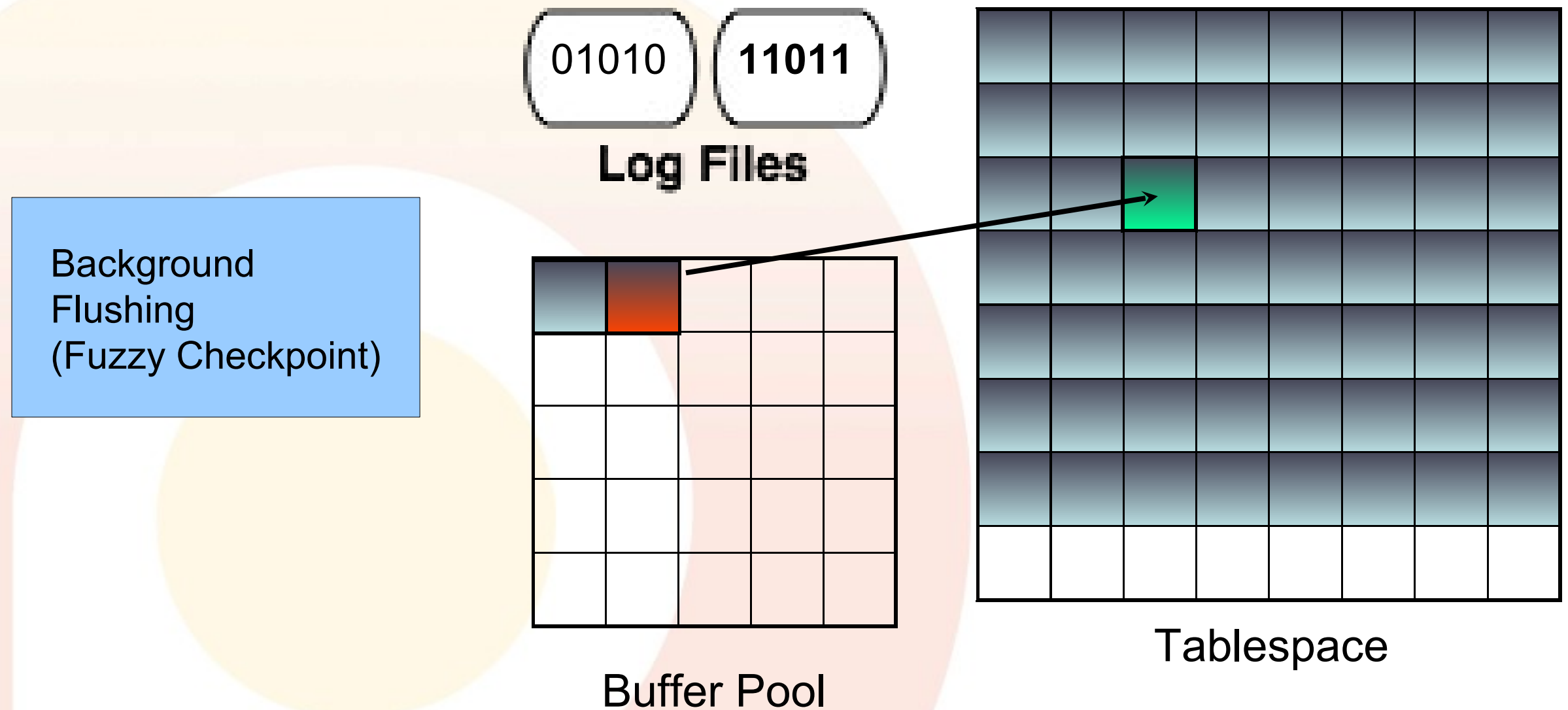
Basic Operation (cont.)



Basic Operation (cont.)



Basic Operation (cont.)



Basic Operation (cont.)

01010 11011

Log Files

Background
Flushing
(Fuzzy Checkpoint)

Buffer Pool

Tablespace

Why delay writing page?

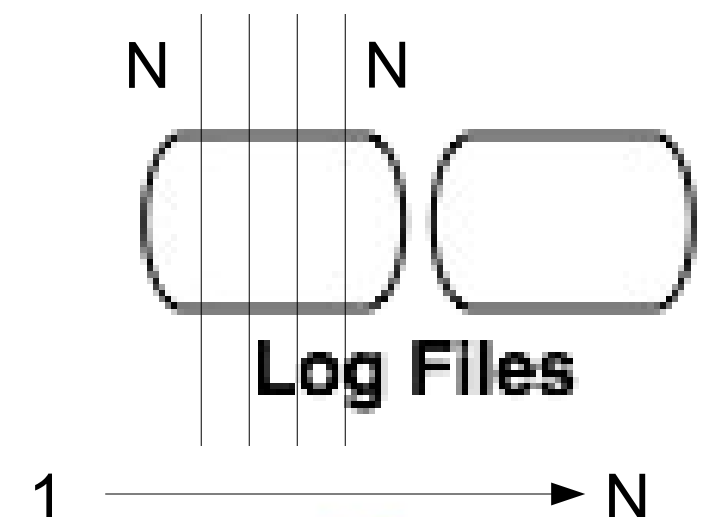
- ★ This is an optimization!
 - ♦ The log file IO is sequential and much cheaper than live updates.
 - ♦ The IO for the eventual updates to the tablespace can be optimized as well.
- ★ This works provided that there is enough transaction log data to *recover those pages which have been modified, but not yet flushed to disk.*

It turns out this is one of the main tuning points for InnoDB.

Durability

More on Logs...

- ★ Logs are only used during recovery.
 - ✦ Not even read when we need to write down dirty pages!
- ★ To figure out which pages need to be evicted we have two lists - **the flush list** and the **LRU**.
- ★ Log activities are all assigned a LSN (log sequence number).



Log Files, Checkpoints, etc.

- ★ Most database systems work this way:
 - ♦ In Oracle the transaction logs are called “Redo Logs”.
- ★ The background process of syncing dirty pages is normally referred to as a “Checkpoint”.
- ★ InnoDB has *fuzzy* checkpointing.

Log Writing

- ★ You can change increase `innodb_log_file_size`. This will allow InnoDB to “smooth out” background IO for longer.
- ♦ **Tip:** Optionally you could change `innodb_log_files_in_group` as well. Be aware that your effective log file is $\text{innodb_log_file_size} * \text{innodb_log_files_in_group}$

Log Writing (cont.)

- ★ You can also change **`innodb_flush_log_at_trx_commit`** to 0 or 2 to reduce the durability requirements of this write.
 - ♦ Requires less flushing - particularly helpful on systems without writeback caches!
- ★ **`innodb_log_buffer_size`** may also help buffer changes for longer before writing to the logs.
 - ♦ Very workload dependent - tends to be more helpful for writing big TEXT/BLOB changes.

Recovery Process (simplified)

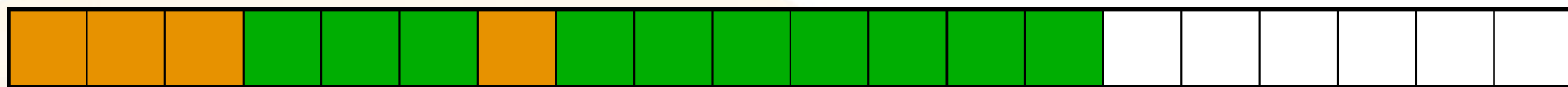
- ★ We replay through the log files, and figure out what changes need to be applied starting from the oldest LSN.
- ★ Since dirty pages are not flushed in order, we may discover some changes are already made.
- ★ We can tell if a pages is already up to date, because each page internally stores its LSN.

What's the best log file size?

- ★ *[Simple Answer]* It's a trade off between performance and recovery time - so it may depend on your *recovery time objective*.
- ★ It is **not possible** to make generic statements such as "a 128M log file will recovery in 10 minutes".

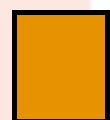
Why is that?

- ★ Picture the log visually:

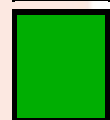


Recoverability is the difference between these two points. If flushing dirty pages is behind, it will take longer.

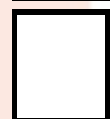
Not all flushing to the tablespace is done in order. By default request merging is done to neighbor pages.



Change for which page has been modified in tablespace



Change written to log file



Unused space in the log file

Those numbers in INNODB STATUS

```
★ ---  
LOG  
---  
Log sequence number 43921309413  
Log flushed up to   43921308508  
Last checkpoint at  43497448671  
..
```

Last modification made -
which has not necessarily
been flushed.

Last successful operation
that has been flushed to
the log files.

Last successful point for
which all background
operations have been
completed.

★ Rough sizing rule of thumb is to track
change in “log sequence number” over
an hour. i.e.

★ 3951840903-3836410803
= 115430100 bytes
= 110M or (55M * 2 log files)

<http://www.mysqlperformanceblog.com/2008/11/21/how-to-calculate-a-good-innodb-log-file-size/>

Frequently Asked Questions

- ★ **Q:** What do we write to the log - is it committed data only, or can we write uncommitted data as well?

Frequently Asked Questions

- ★ **Q:** What do we write to the log - is it committed data only, or can we write uncommitted data as well?
- ★ **A:** Both.

Frequently Asked Questions (cont.)

- ★ **Q:** Doesn't that mean that you have to remember how to apply uncommitted changes as part of crash recovery as well?
- ★ **A:** Yes it does. InnoDB stores UNDO information to be able to do this.

SELECT (more detail)

SELECT * FROM
City WHERE
CountryCode='Aus'

Do the pages
exist in
memory?

Log files (on disk)



\$datadir/ib_logfile0 and
\$datadir/ib_logfile1 as one logical
file.

Log Buffer (in memory)

Tablespace (on disk)

Buffer Pool (in memory)

Additional in
memory items

Flush List

LRU

SELECT (more detail)

SELECT * FROM
City WHERE
CountryCode='Aus'

Do the pages
exist in
memory?

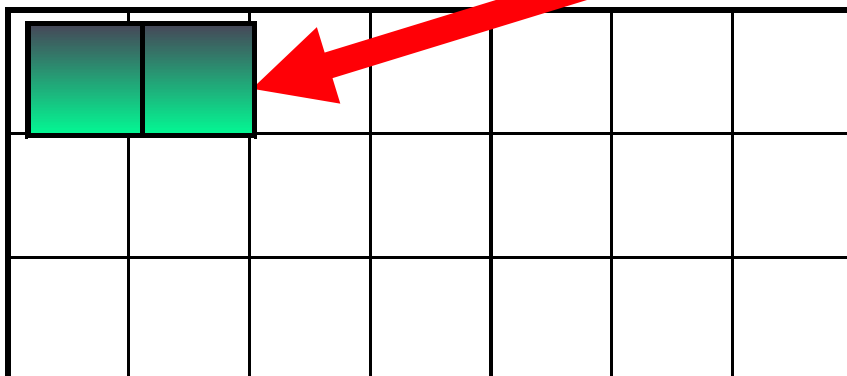
Log files (on disk)

\$datadir/ib_logfile0 and
\$datadir/ib_logfile1 as one logical
file.

Log Buffer (in memory)

No? Do physical
IO to load them
into the buffer
pool.

Buffer Pool (in memory)

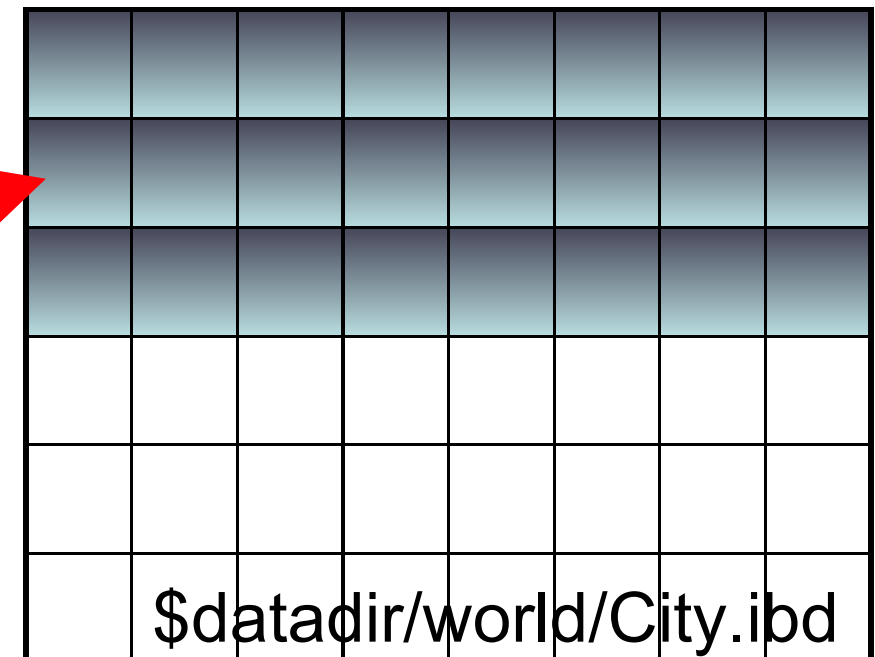
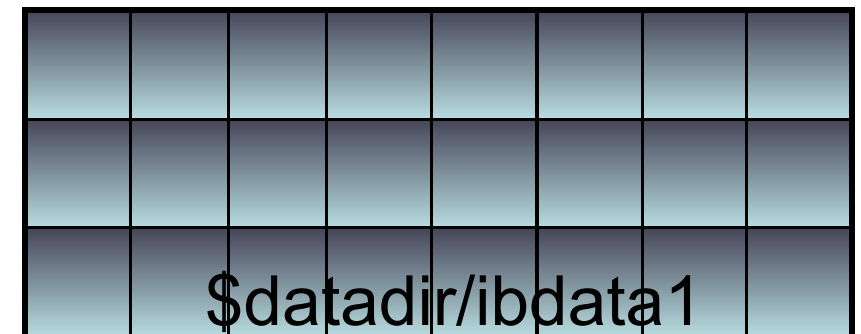


Additional in
memory items

Flush List

LRU

Tablespace (on disk)



SELECT (more detail)

```
SELECT * FROM  
City WHERE  
CountryCode='Aus'
```

Log files (on disk)



\$datadir/ib_logfile0 and
\$datadir/ib_logfile1 as one logical
file.

Tablespace (on disk)

\$datadir/ibdata1

\$datadir/world/City.ibd

Log P

**innodb_read_
io_threads**

Buffer Pool (in memory)

As they are loaded,
remember when these
pages are created (LRU)

ash List

LRU

SELECT (more detail)

SELECT * FROM
City WHERE
CountryCode='Aus'

Return
results to
the client.

Log files (on disk)

\$datadir/ib_logfile0 and
\$datadir/ib_logfile1 as one logical
file.

Log Buffer (in memory)

Tablespace (on disk)

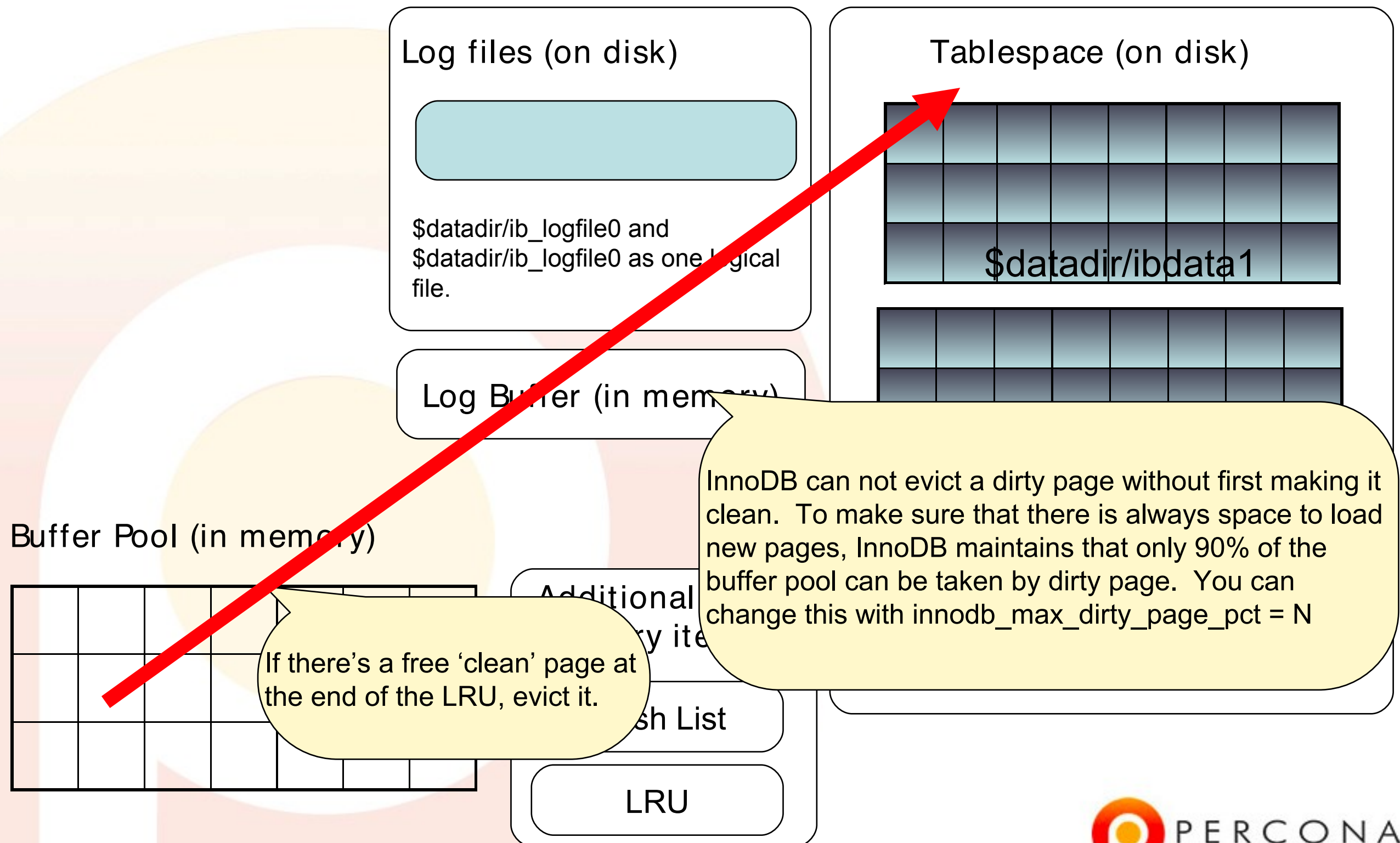
Buffer Pool (in memory)

Additional in
memory items

Flush List

LRU

No free pages in Buffer Pool



Flushing, synching and buffers

- ★ When you write to a file you are writing to a buffer.
- ★ The buffer exists in your process..
- ★ Flushing a buffer moves it to the OS cache
- ★ Anything flushed will survive the process crashing
- ★ Data synced (fsync) will survive power outage

innodb_flush_log_at_trx_commit=1

- ★ At every commit the innodb log buffer is synched
- ★ This is the most expensive mode of operation but there will be no loss of committed transactions after a crash or power loss.
- ★ Battery backed write cache or SSD important
- ★ This is the only mode in which InnoDB is fully ACID compliant.

innodb_flush_log_at_trx_commit=2

- ★ At every commit (or each second) InnoDB will flush (but not sync) the InnoDB log buffer.
- ★ Each second InnoDB will sync the log buffer.
- ★ Implies that InnoDB could lose some transactions (usually up to one second) if there is an OS crash or power loss.
- ★ Not fully ACID compliant
- ★ Slaves might receive rows that are rolled back if the master crashes.

innodb_flush_log_at_trx_commit=0

- ★ At commit InnoDB does not do anything with the log buffer
- ★ Once per second the log buffer is synced.
- ★ Implies that InnoDB could lose some transactions (usually up to one second) if MySQL crashes or if there is an OS crash or power loss.
- ★ Not ACID compliant.
- ★ Slaves are very likely to receive rows that are rolled back when the master crashes.
- ★ Don't use this mode.

Undo Information

- ★ The transaction logs store **REDO** information - and data is written whether committed or not. InnoDB also has to be able to **UNDO** changes;
 - ♦ It has to be able to show old versions of the data to older transactions that are still active.
 - ♦ It has to be able to reverse a transaction.
- ★ The undo information is stored in the InnoDB tablespace.

Internals

On Disk Format

- ★ Everything is stored in an InnoDB tablespace.
- ★ (Default Mode) Tablespace is one contiguous logical file called **ibdata1**.
- ♦ It is possible to configure this to be many physical files - i.e. ibdata1 (20G), ibdata2 (20G), ibdata3 (20G).
- ♦ The last file (ibdata3) can autoextend, but every other file is fixed in length.

On Disk Format (cont.)

- ★ **Q:** Why would you have ibdata1, ibdata2, ibdata3 and not just ibdata1?
- ★ **A:** From InnoDB's perspective it makes no difference. From your perspective it may be easier to backup split files.
- ★ *[Advanced]* Some filesystems may fragment / have different performance characteristics with very large files. Don't assume, test!

On Disk Format (cont.)

- ★ The first tablespace file is filled before, the next file is started.
- ♦ There's no advanced features such as mirroring, striping or round robin writing to the physical files.
- ♦ InnoDB wants you to use the Operating System to do this.
- ♦ Each page is filled 93% (15/16) leaving a small gap. This is also not configurable.

“Advanced” options (1/2)

- ★ You can bypass the Filesystem and configure InnoDB raw partitions.
 - ♦ Not many people are doing this - and it's no longer as big as an advantage as it seems. *Here be dragons.*
 - ♦ Might be an alternative if you have many cores but you do not have the ability to use XFS
 - ♦ Operational tasks like backup are much more difficult

“Advanced” options (2/2)

- ★ You can configure **--innodb-file-per-table**
 - ♦ Each table essentially has its own tablespace.
 - ♦ It may be composed of only one physical file *per table*.
- ★ Yes, there are **no other** tablespace management options!

Is innodb-file-per-table better?

- ★ A difficult question to answer.
- ★ For manageability, **“yes”**.
 - ♦ The biggest win is that you can reclaim space whenever you TRUNCATE/DROP a table, and therefor *reduce the size of your backup*.
 - ♦ With the single tablespace option, the space is marked as free, but the tablespace files will never contract.
 - ♦ Must be selected if you want to use the new InnoDB file format (barracuda)

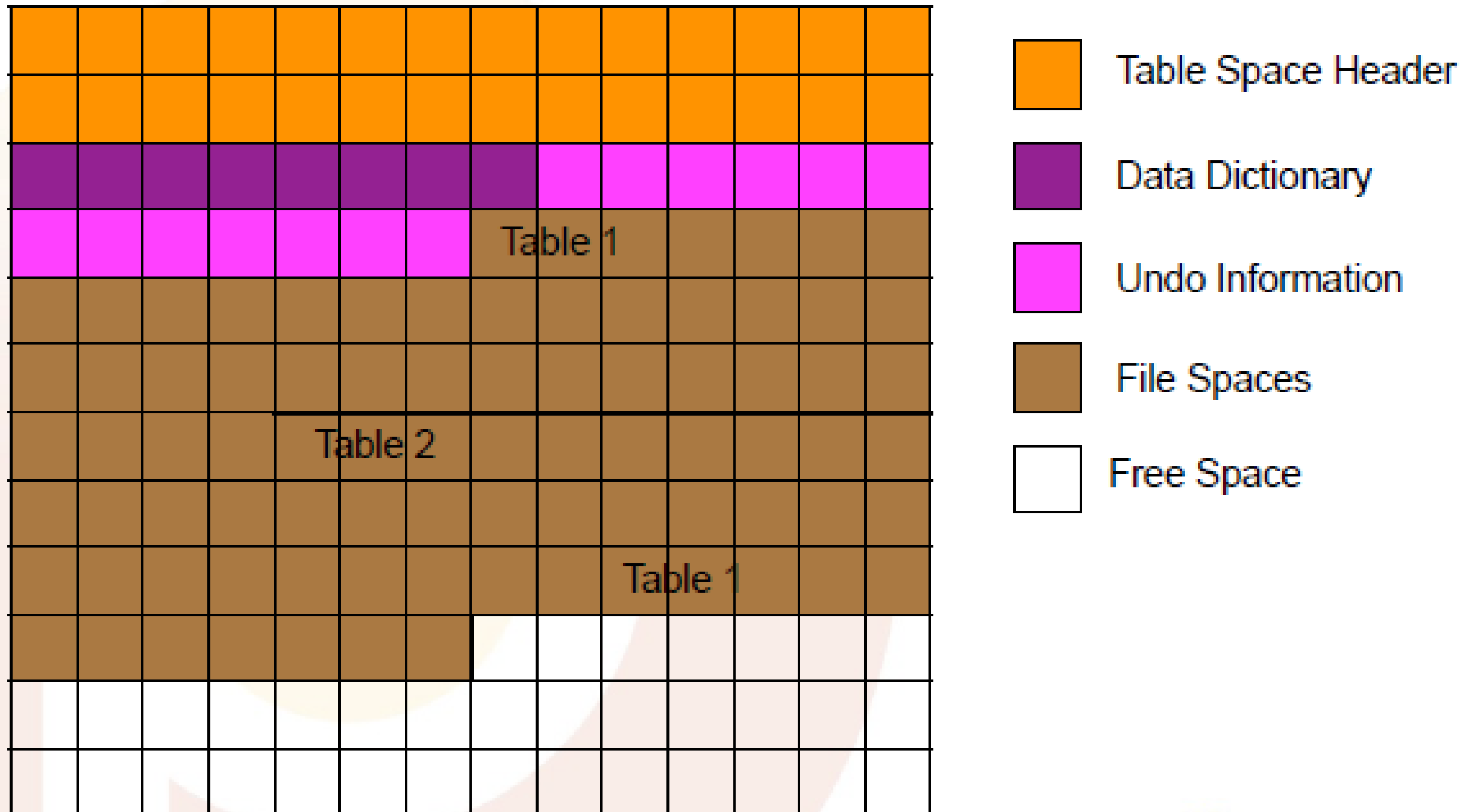
innodb-file-per-table better (cont.)

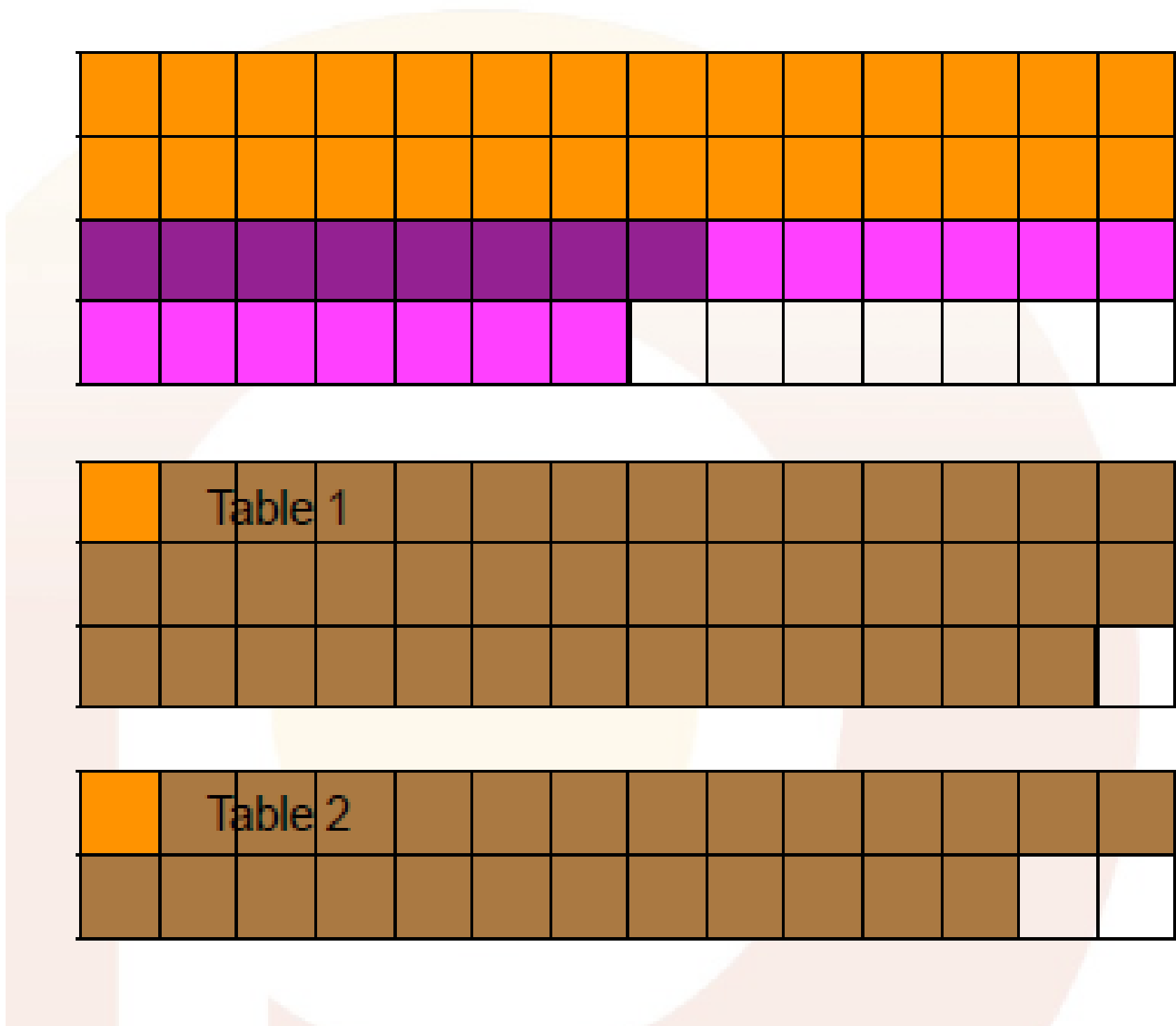
- ★ For performance, it should be *roughly* comparable in most cases.
- ♦ **[Con]** It incurs additional file descriptors.
- ♦ **[Con]** Files on the filesystem may fragment more as they only grow 4MB at a time (not configurable with file-per-table).
- ♦ **[Con]** More complicated internal synchronization. For example, on startup checking 100K tables exist takes some time.
- ♦ **[Pro]** Works better with ext3 and O_DIRECT, since only one thread can write to any file at a time.
 - Not true for xfs filesystem.

Contents

- ★ As well as table data, the table space contains:
 - ♦ A tablespace header
 - ♦ InnoDB's own data dictionary
 - ♦ Undo Information
 - ♦ The Doublewrite buffer
 - ♦ The Insert buffer

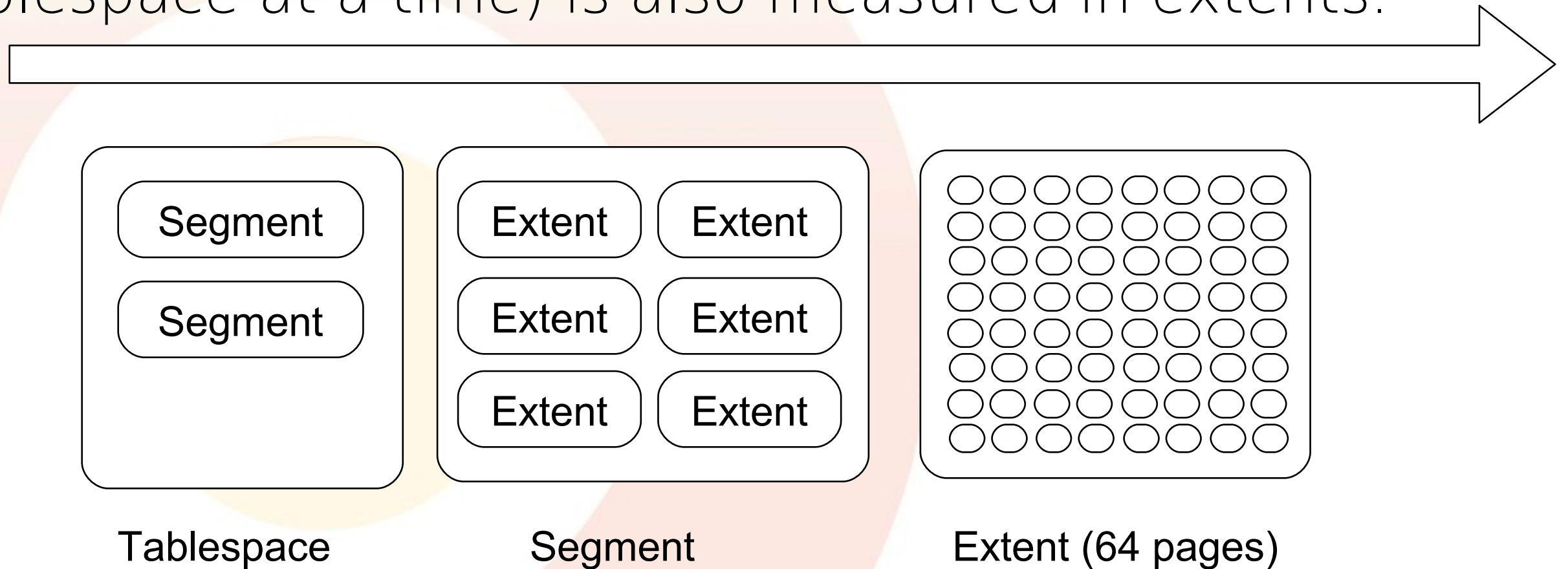
Contents, *visually**:





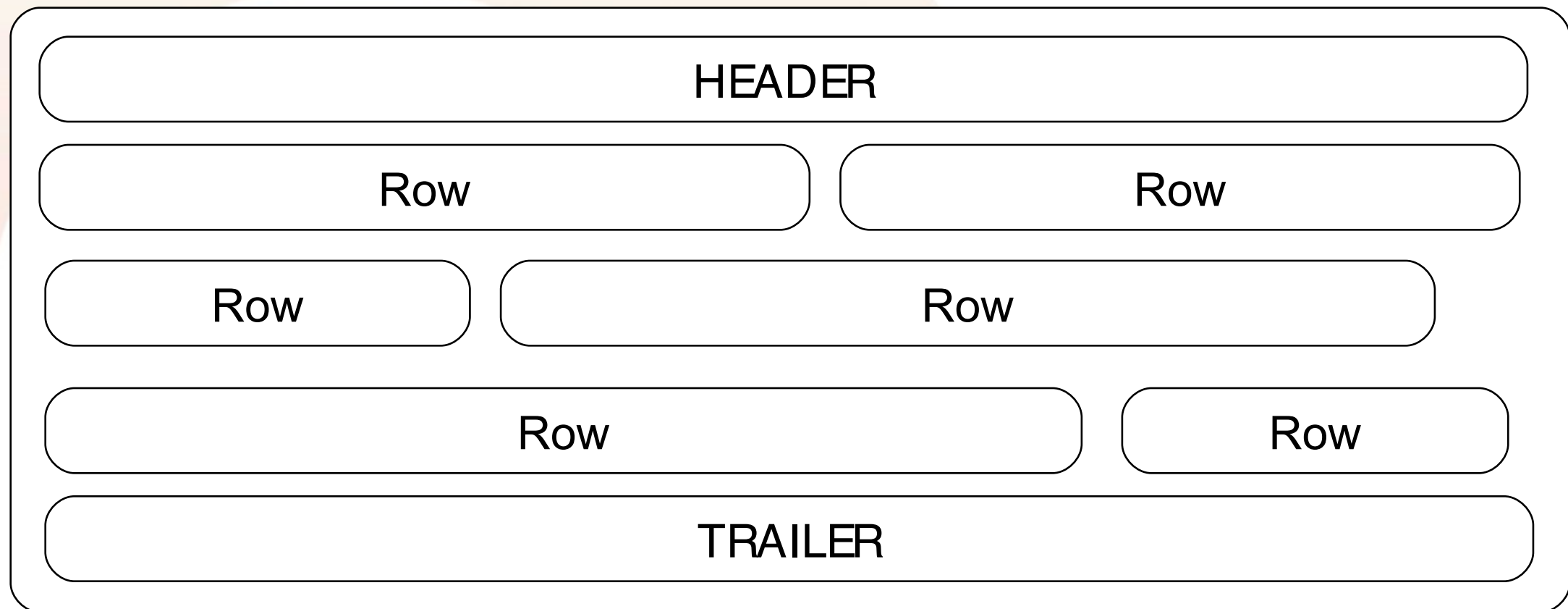
Expanding on Storage

- ♦ Most internal allocations are based on one extent at a time.*
- ♦ **innodb_autoextend_increment** (how big to expand the tablespace at a time) is also measured in extents.



Expanding on Storage (cont.)

- ★ Everything fits into **pages**, which are 16 KiB*.
- ★ A data page might look something like this:



Expanding on Storage (cont.)

★ **Some of the meta data in the header includes:**

- ♦ Information on what page type it is.
- ♦ A page checksum.
- ♦ The Log Sequence Number.
- ♦ The last value of the Log Sequence number as flushed to disk.

★ **Some of the information in the trailer includes:**

- ♦ The last 4 digits of the page LSN repeated.
- ♦ A redundant checksum (obsolete).

Of note: page checksum

- ★ A defensive feature of InnoDB.
 - ♦ On read from storage into buffer pool a check is done to detect silent corruption.
 - ♦ On failure, InnoDB intentionally crashes the server*
- ★ Checksums are updated when writing pages back to disk.
 - ♦ Cost “should be” low relative to the cost of the IO.
 - ♦ “Fast checksum” is available in XtraDB.

For example, “numbers everyone should know” says compressing 1K of data takes 3us, while an IO takes 10ms.

This may be important for workloads with a high page turnover rate, and fast storage (SSDs).

What does a row look like?

- ★ Exact format will differ between versions.
 - ♦ In 5.0 the COMPACT row format was introduced - where NULL values are stored more efficiently.
- ★ Most important features to note here are:
 - ♦ Storage of Transaction ID.
 - ♦ Storage of pointer to older version of row in UNDO space.

Transaction ID

Roll Pointers

Field Pointers

Field #1

Field #2

Field #3

Field -> Row -> Page

- ★ All fields are stored inline in the row - except for Blobs which are “a special case”.
- ★ Rows can not be split across pages.
- ★ InnoDB sets a restriction that **two rows** must fit within one page.*

Blob/Long Text Storage

- ★ VARCHAR/TEXT/BLOB are the special case. They are all handled the same internally -
 - ♦ They default to, but are not required to be stored in the same page as the row.
 - ♦ The rules to how blobs are stored is very specific and changes between versions of InnoDB.

Blob/Long Text (cont.)

- ★ "Small blobs"
 - ♦ When whole row fits in ~8000 bytes, the blob is always stored on the page.
- ★ "Large blobs"
 - ♦ First 768 bytes are stored on the same page, remainder is stored on overflow page(s) (Antelope).
 - ♦ Will be stored full on external pages, with 20 byte pointer to overflow page (Barracuda).

What does the log file look like?

- ★ These records are 512 byte aligned.
- ★ The log file does not store a complete page - just the changes that need to be made to recreate the page. i.e.
- ♦ Space ID + Page ID + Offset + Payload.

Space ID + Page ID is the internal addressing system.

ibdata1 is always Space ID 0. Page ID is the page number inside that space.

What does Undo look like?

- ★ Undo information is stored in the *rollback segment*.
 - ♦ The default is one segment comprised of 1024 undo slots.
 - ♦ Each open transaction requires at least 1 undo slot.
- ★ The* rollback segment is *always* in the global tablespace.

*XtraDB supports more than one rollback segment

In Memory Structures

- ★ These consume memory in addition to the InnoDB buffer pool
 - ♦ Meta Data for accessed tables
 - ♦ Lock information
 - ♦ Adaptive Hash Index
 - ♦ SQL Statements
 - ♦ Thread Stacks

Additional Memory Pool

- ★ This feature is becoming obsoleted by scalable OS implementations of malloc:
 - ★ In InnoDB plugin, the setting `innodb_use_sys_malloc` (default: 1) disables the additional memory pool.
- ★ Whether you are using built-in or plugin, leaving `innodb_additional_mem_pool_size` at the default setting is usually fine.

InnoDB Buffer Pool

- ★ Configured by **`--innodb-buffer-pool-size`**
- ★ The main setting for InnoDB caching - responsible for all pages types (data, indexes, undo, insert buffer..)
- ★ You may be surprised how much insert buffer and undo pages take up!

InnoDB Buffer Pool (cont.)

- ★ Recommended size is “about 50-80% of memory”, but there are better ways of calculating.
- ♦ **[Warning]** Meta data always consumes an additional 5-10% more space on top of that.
- ♦ **[Warning]** Some subsystems in MySQL rely on healthy OS caches - i.e. binary logs, relay logs.

Other Settings

- ★ Maximum percentage of the buffer pool that can contain dirty pages:
`innodb_max_pct_dirty_pages = N` (default: 90 or 75)
- ★ You could decrease this if you are concerned that you are evicting clean pages too frequently to service reads, because the dirty pages can not be made free.
- ★ Previously, this setting was the only way to increase the rate at which the background thread flushed dirty pages.

In InnoDB plugin/XtraDB,
likely better way to set this
is `innodb_io_capacity=N`

Other Settings (cont.)

- ★ **`innodb_log_buffer_size = N`** (default 1M / 8M)
 - ♦ Size of buffer to spool changes to before writing to logs.
- ★ **SHOW GLOBAL STATUS LIKE**
 - ♦ **`'innodb_log_waits'`** - indicates the number of times the buffer was full and needed to be synchronously flushed.
 - ♦ Default is normally fine. Usually only an issue when writing large TEXT/BLOBs.

Locking

- ★ Locks are held for the duration of a transaction.
- ★ It is critical that UPDATE/DELETE queries are well indexed.
 - ♦ More rows than you may expect may be locked.
 - ♦ This additional locking is performed by InnoDB to ensure consistency with statement based replication...

More on locking

- ★ Two common types of locks you may be familiar with:
 - ♦ SHARED (S) aka READ LOCKs.
 - ♦ EXCLUSIVE (X) aka WRITE LOCKs.
- ★ In InnoDB most of the time when we talk about “row locking” rows, it is X locks:
 - ♦ An S lock never occurs for reads because of MVCC.
 - ♦ S locks are normally only visibly with foreign key constraints.

Clustered Index

- ★ Everything in InnoDB is an index:
 - ♦ Data is stored in a clustered index organized by the primary key. In the absence of a primary key, the first unique not null key is selected*.
 - ♦ Other indexes are stored in secondary indexes.

What is a clustered index?

- ★ First lets look at how MyISAM stores data*:

Staff.MYI

8															
4				12											
2		6		10		14									
1	3	5	7	9	11	13	15								

Data is stored “roughly” in insertion order, with no guarantees, i.e.

Deleted rows may be filled with newer records.

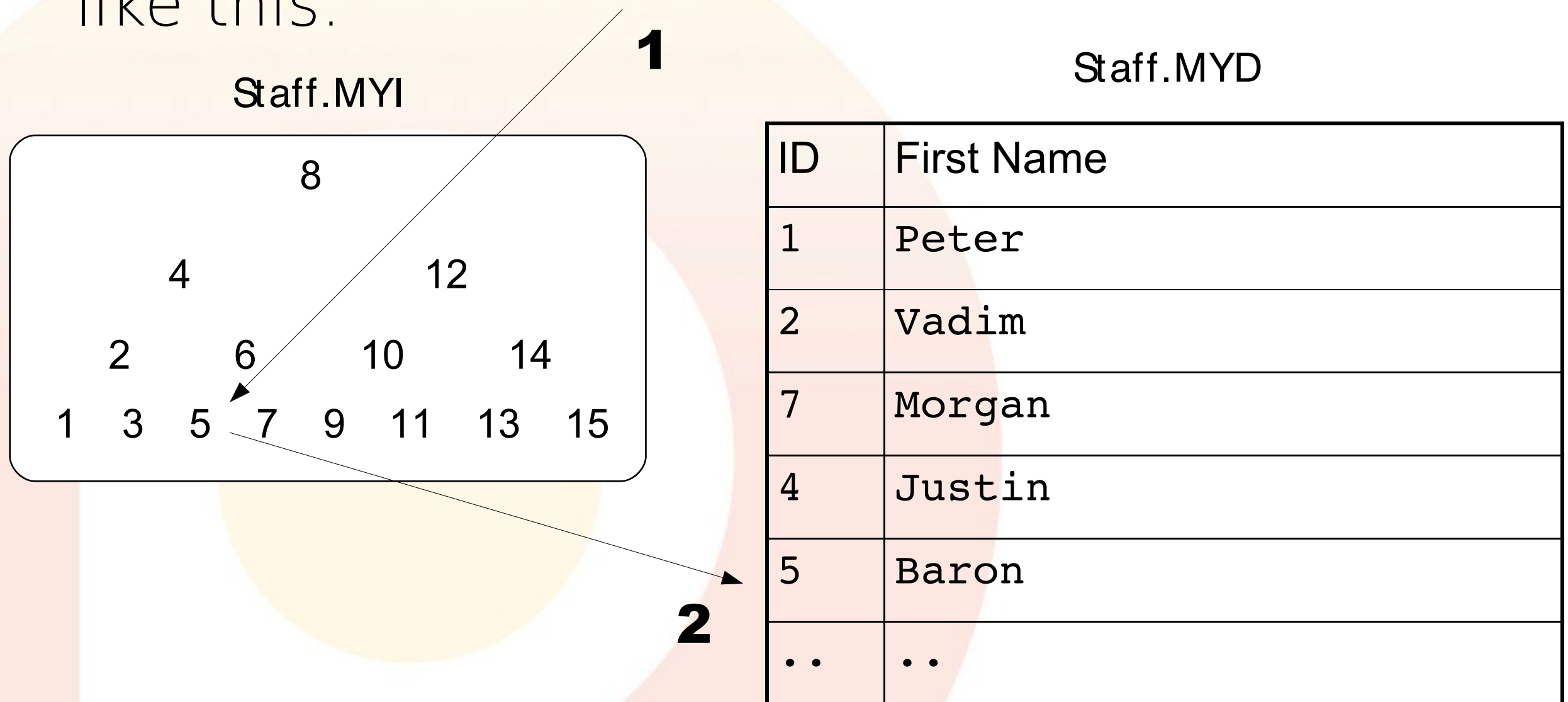
Staff.MYD

ID	First Name
1	Peter
2	Vadim
7	Morgan
4	Justin
5	Baron
..	..

* Illustrating B-Tree as Binary Tree for simplicity

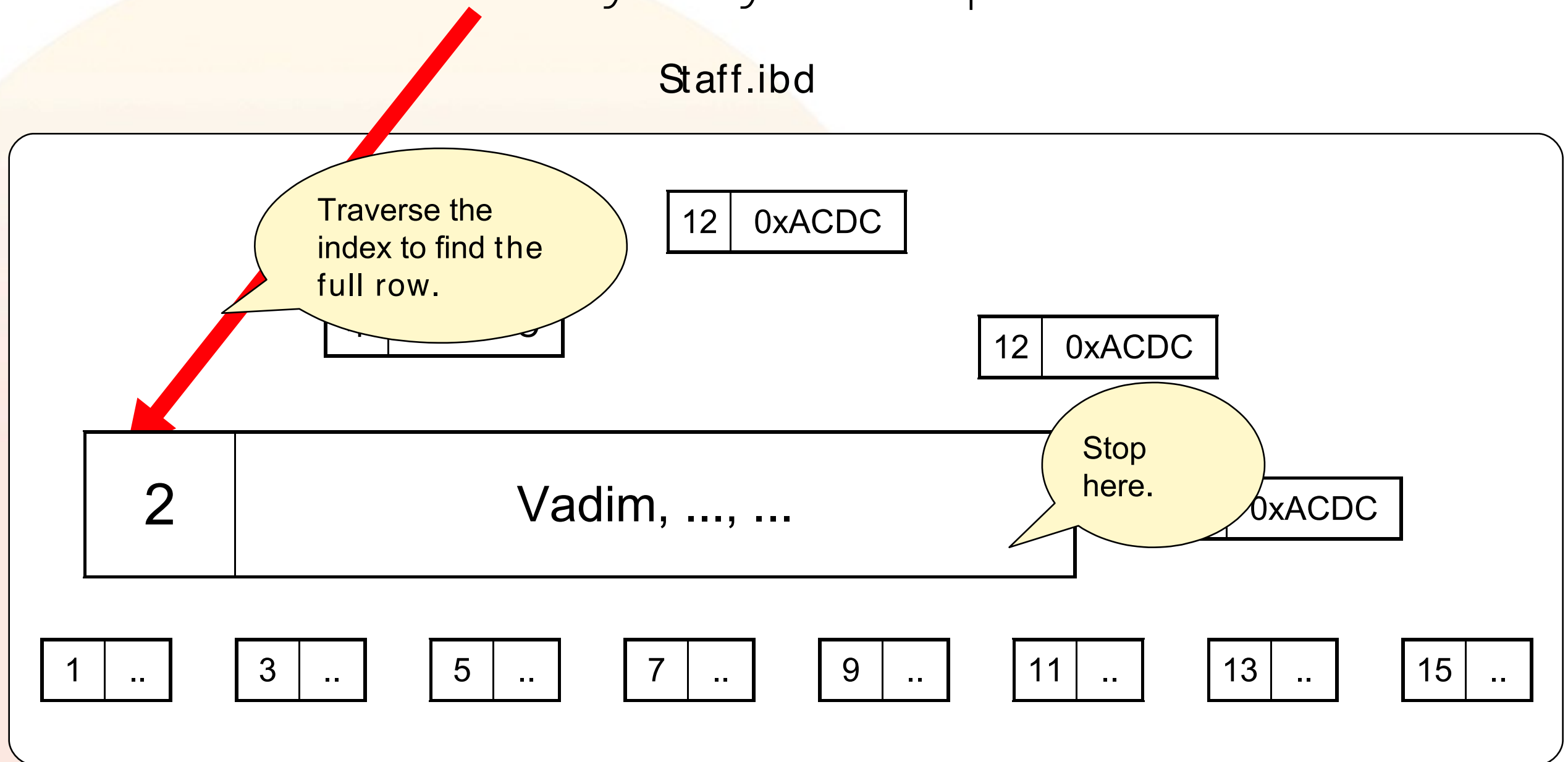
What is a clustered index (cont.)

- ★ A MyISAM primary key lookup looks something like this:



What is a clustered index (cont.)

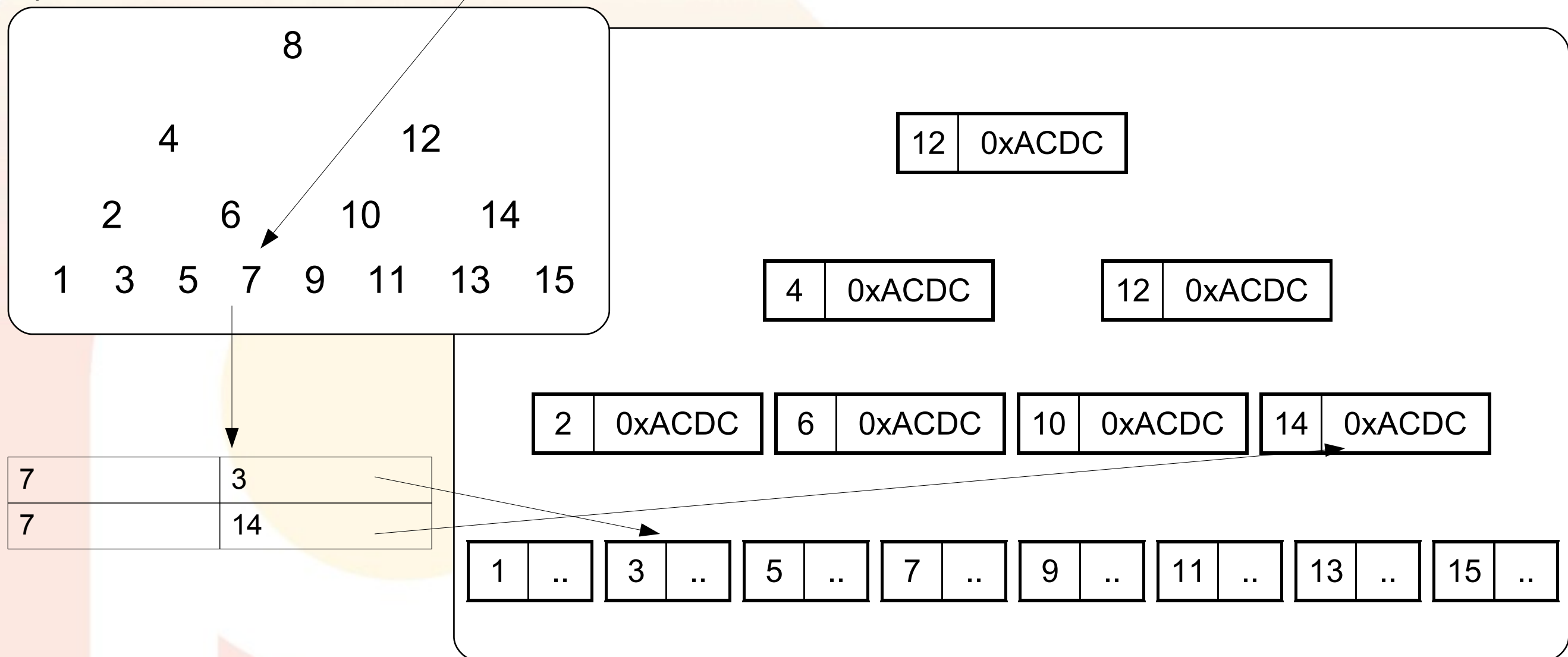
- ★ An InnoDB Primary Key lookup looks like this:



What is a clustered index (cont.)

- ★ A secondary key lookup looks like this:

department_number



Clustered Index (cont.)

- ★ This design has some interesting consequences:
 - ♦ Primary key lookups are very fast.
 - ♦ Inserting data in order is fast - out of order can be very slow, and cause fragmentation.
 - ♦ Secondary indexes can become very large if you have a large primary key.

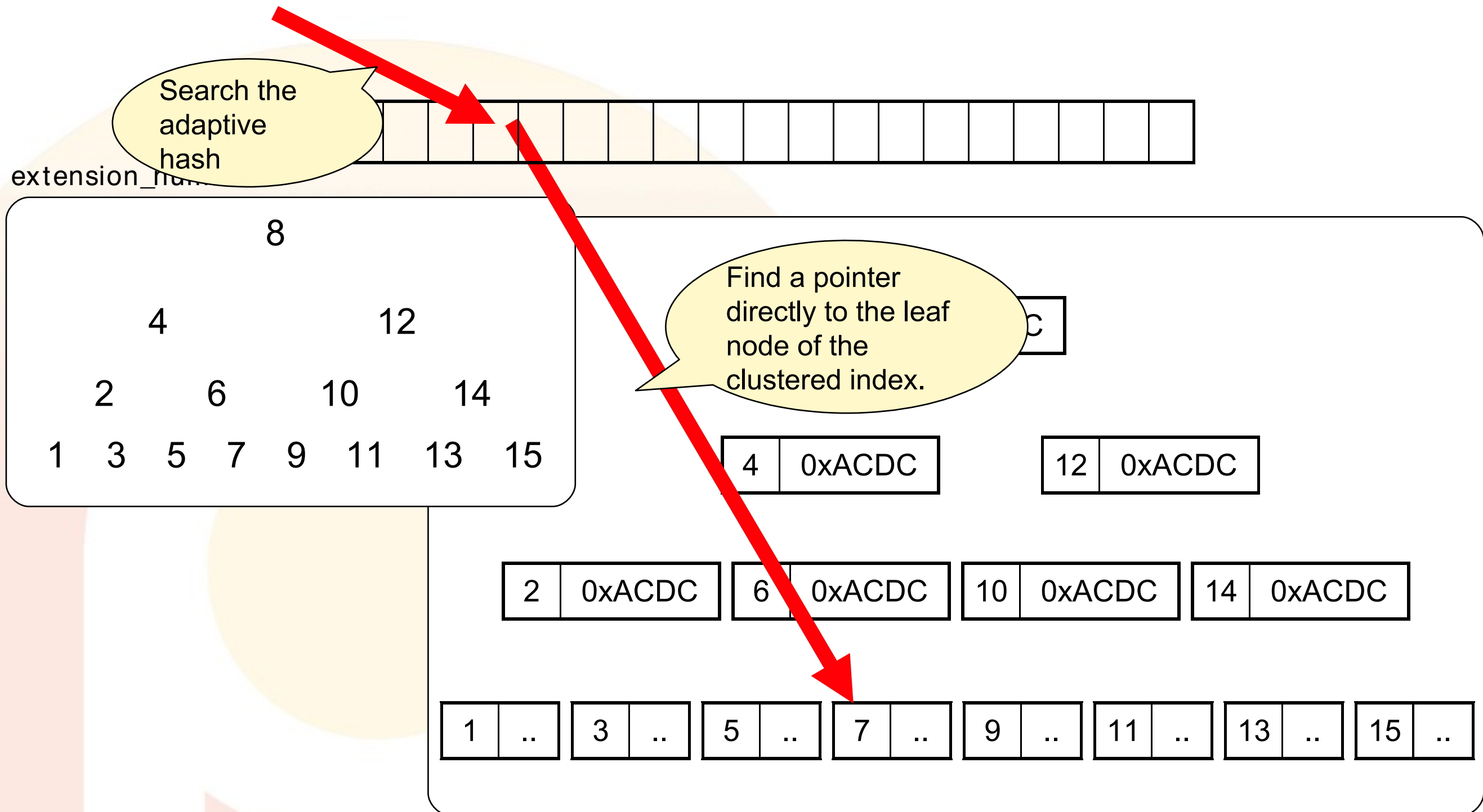
Clustered Index (cont.)

- ★ In practical terms this means:
 - ♦ Don't use GUIDs for InnoDB tables!
 - ♦ Never piggy-back the primary key index into the end of a composite index or covering index - it is already included for free.

Adaptive Hash

- ★ Secondary index lookups are slower in InnoDB;
 - ♦ First you need to scan the secondary index.
 - ♦ Then you can scan the primary key index.
- ★ Most workloads have **hotspots**.
 - ♦ InnoDB monitors index usage. Frequently accessed values are inserted into an in-memory hash table to accelerate lookups.
 - ♦ The hash table does not cover the whole index.

Adaptive Hash (cont.)



Adaptive Hash (cont.)

- ★ Not much transparency into its operation:

- ★ -----
INSERT BUFFER AND ADAPTIVE HASH INDEX

..

Hash table size 31874747, node heap has 9526 buffer(s)
25448.49 hash searches/s, 54424.21 non-hash searches/s

How many spaces in
the hash table.

Number of pages that the
hash table takes up.
i.e. 9526 = 126M

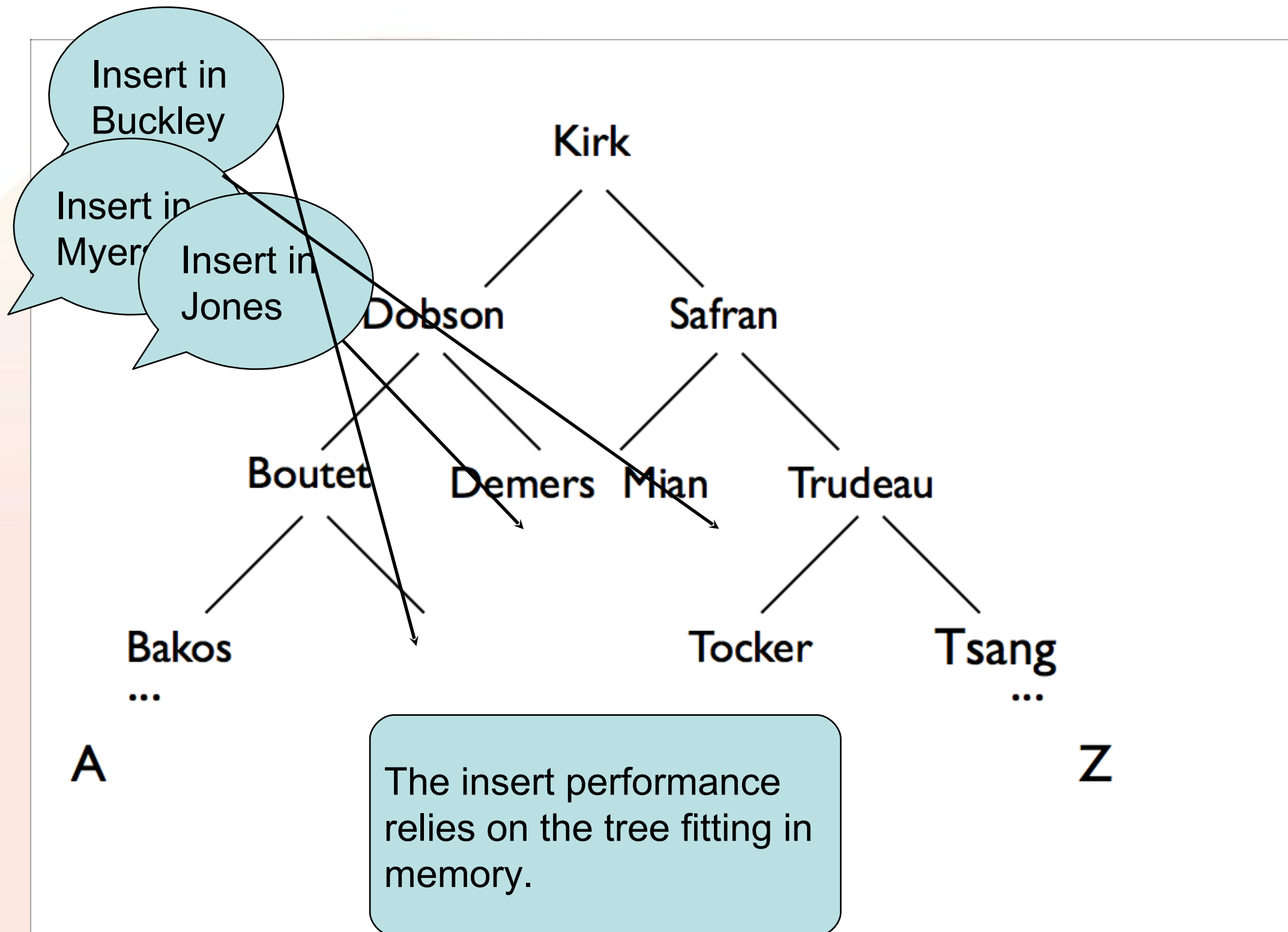
Insert Buffer / Change Buffer

- ★ Very useful for non-unique secondary indexes.
 - ♦ Reduces random IO by delaying index creation and first writing to a buffer.
 - ♦ Potential for merging update requests.
 - ♦ InnoDB plugin 1.1+ (MySQL 5.5) buffers changes to secondary index for INSERT/UPDATE/DELETE, not just INSERT/LOAD DATA.

Change Buffer (cont.)

- ★ Builds the index pages just-in-time if destination page is loaded into buffer pool -
 - ♦ Entirely safe - never returns wrong data.

Insert Buffer



Insert Buffer Concerns

- ★ No configuration possible in the built-in InnoDB.
 - ♦ Potential to take up to 1/2 the buffer pool's memory!
- ★ May want to disable it on SSDs.
 - ♦ Still has the advantage that requests are reduced for merging, but random IO reduction is of no help.
 - ♦ Buffer pool may be better spent on other pages.

Insert Buffer efficiency

★ In 5.1 plugin and great

Current size of used memory in the insert buffer in pages.
i.e. 1 page = 16K

★

INSERT BUFFER AND ADAPTIVE HASH INDEX

Ibuf: size 1, free list len 16829, seg size 16831,
0 inserts, 0 merged recs, 1 merges
..

Total amount of space allocated, but unused. 16829 pages = 262 MB.

Total amount of space that has been allocated for the insert buffer. 16831 pages = 262 MB.

Insert Buffer efficiency (cont.)

- ★ In 5.1 plugin and greater:

- ★ -----
INSERT BUFFER AND ADAPTIVE HASH INDEX

Ibuf: size 1, free list len 16829, seg size 16831,
0 inserts, 0 merged recs, 1 merges

Number of row inserts
into the insert buffer
since server startup.

How many merged
records the inserts have
resulted in.

A “merge” indicates
success at eliminating a
physical IO operation.

Double Write Buffer

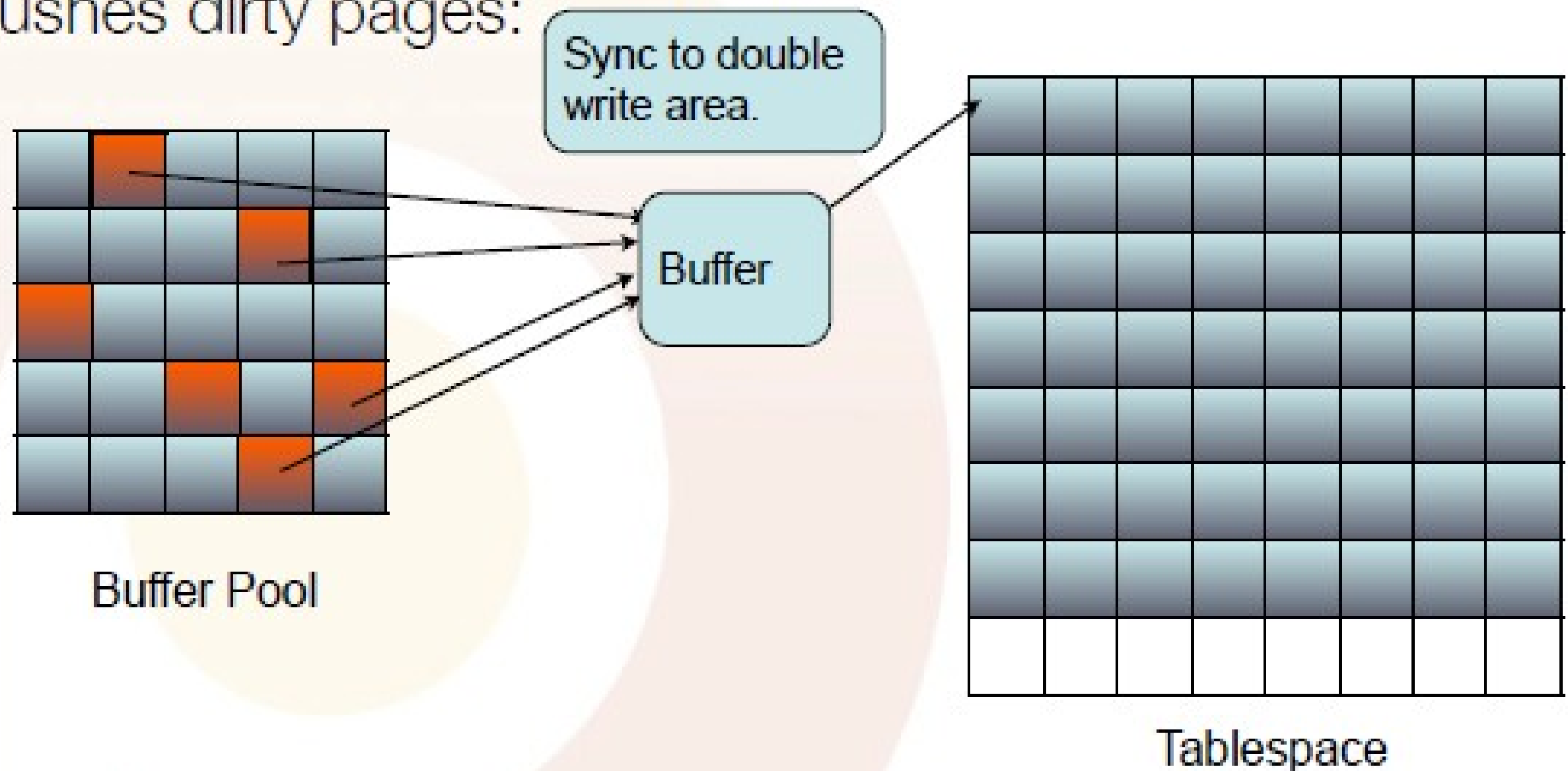
- ★ Changes to table space are “double written” to a section of the main tablespace by default. This prevents partially written/corrupt pages.
- ★ Filesystem journals prevent meta data corruption, but data can still be problematic.

Actual Implementation

- ★ The double-write area is 2MB and consists of 2 parts.
 - ♦ i.e. 2x64 pages.
- ★ Writing to this buffer is serialized. InnoDB synchronously confirms that data is written.
- ★ Then individual pages are asynchronously written to destination locations using write threads*.

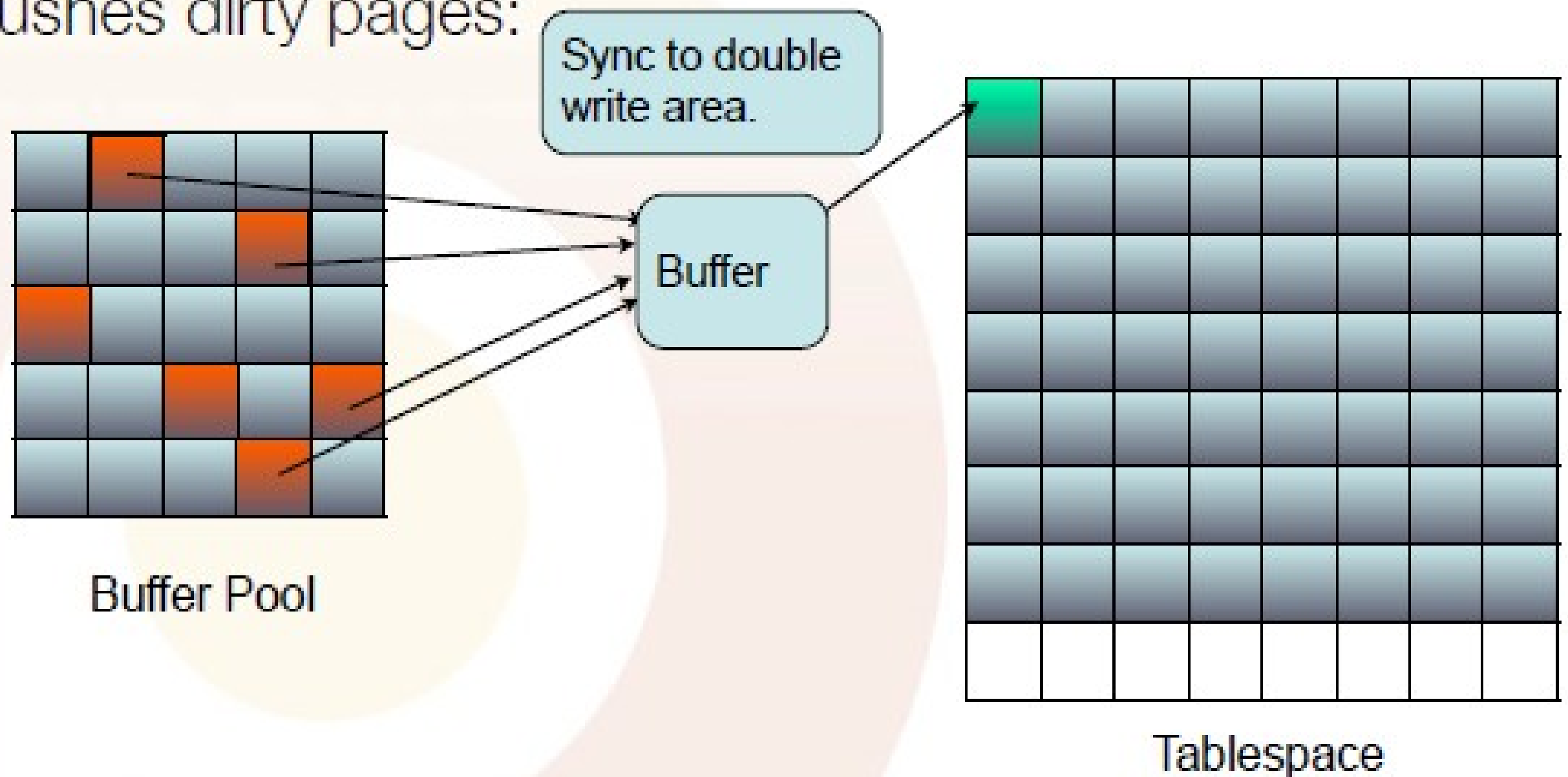
Double Write Buffer (cont.)

- ★ The extended description of how the background thread flushes dirty pages:



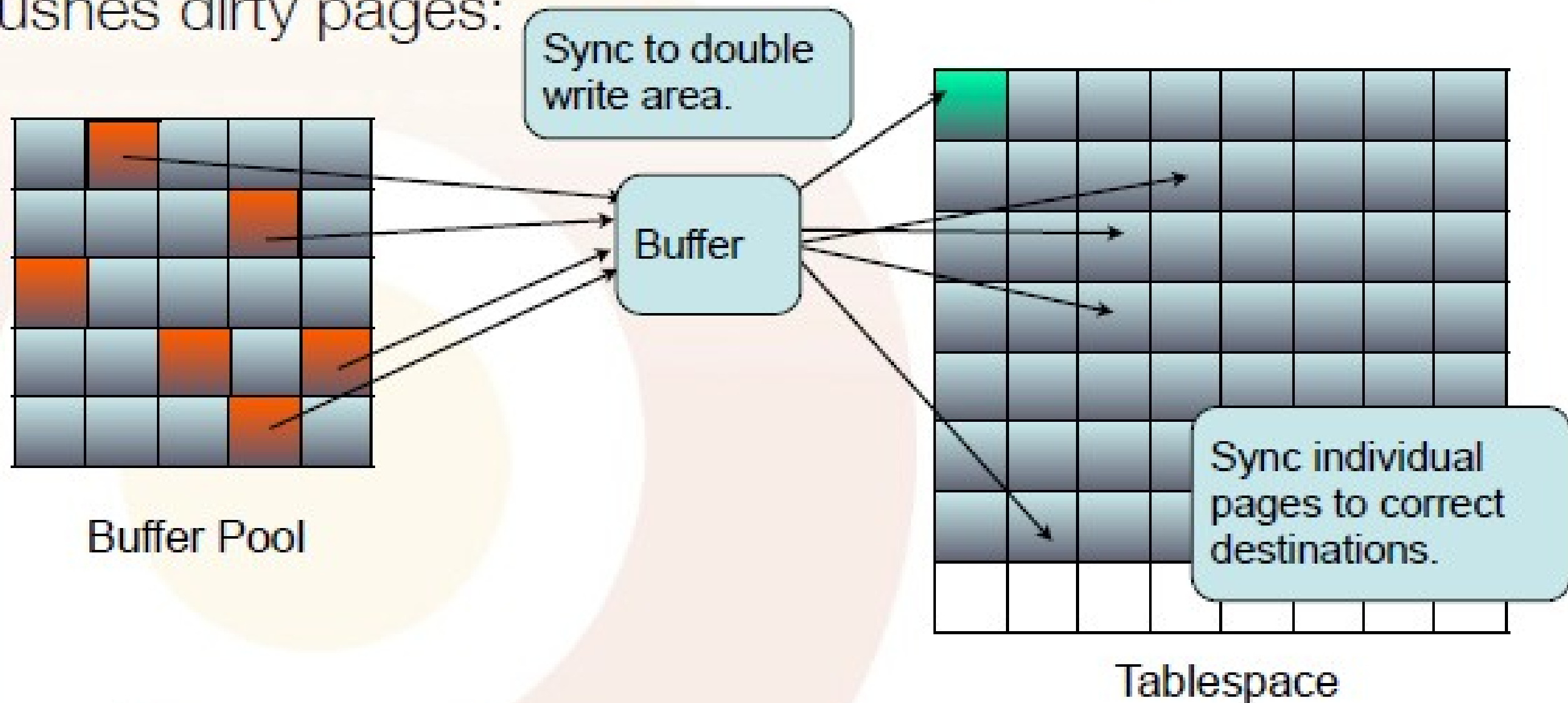
Double Write Buffer (cont.)

- ★ The extended description of how the background thread flushes dirty pages:



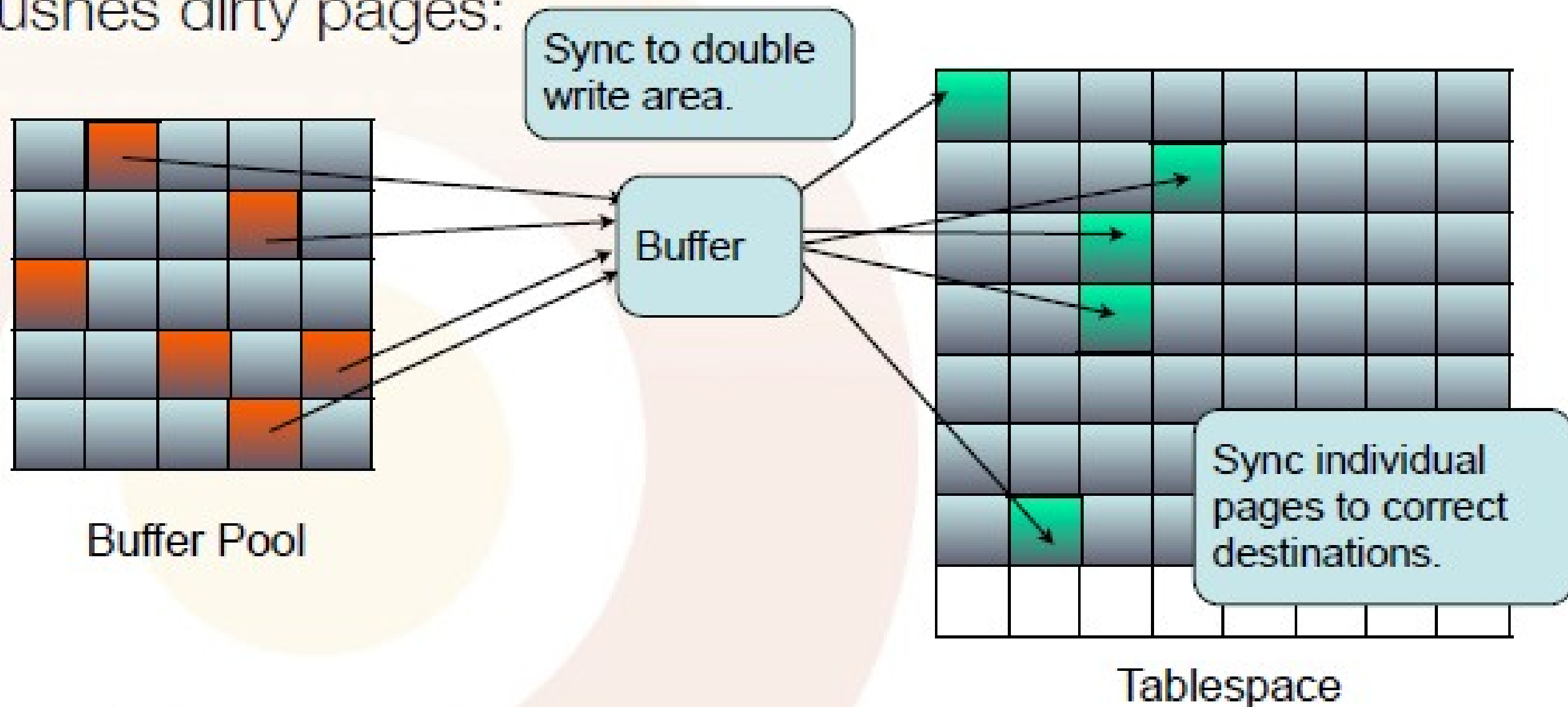
Double Write Buffer (cont.)

- ★ The extended description of how the background thread flushes dirty pages:



Double Write Buffer (cont.)

- ★ The extended description of how the background thread flushes dirty pages:



Double Write Buffer (cont.)

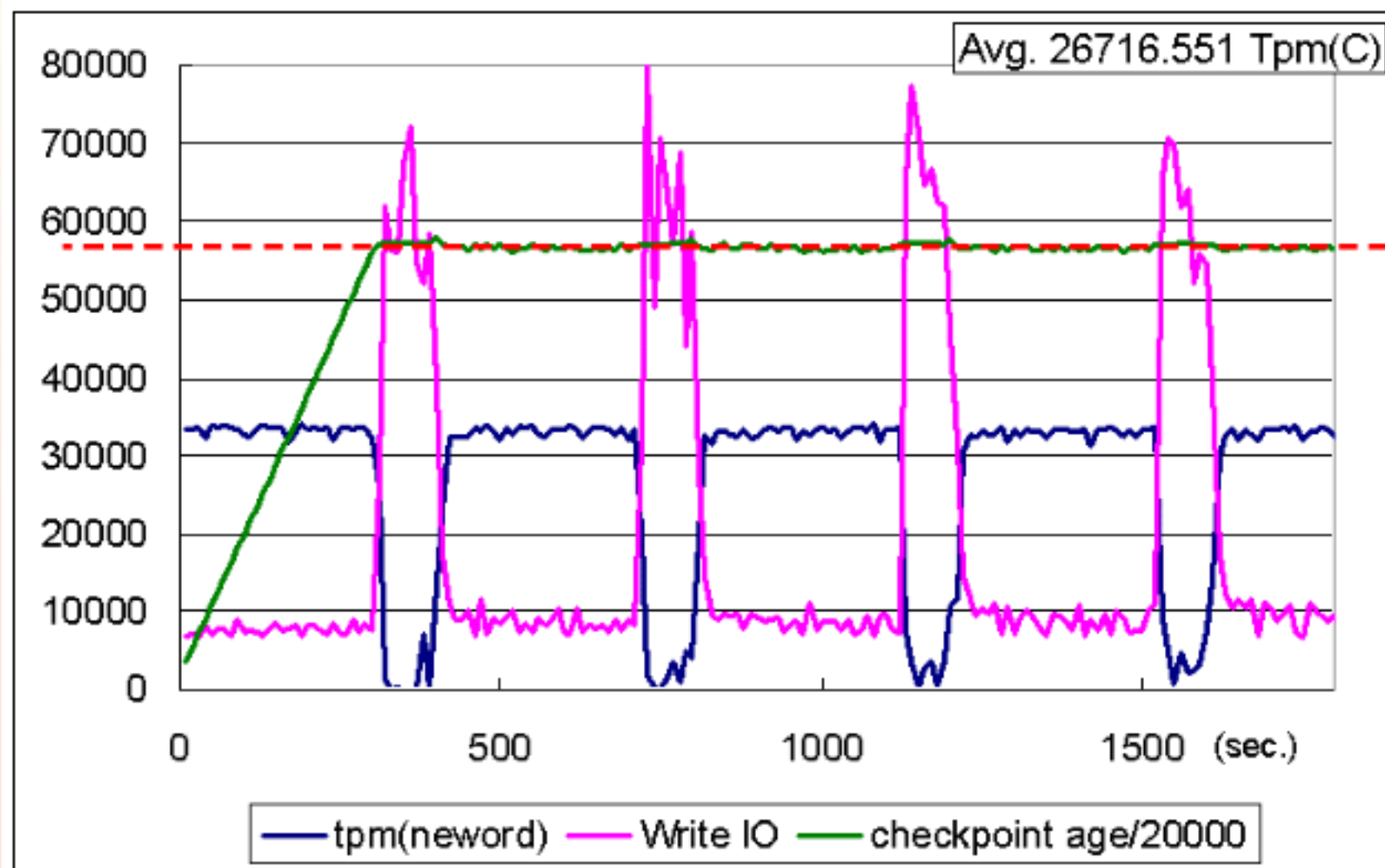
- ★ **Q:** When would you want to disable the double write buffer?
- ★ **A:** *Transactional* filesystems like ZFS make sure that writes are atomic. This is about the only reason you would want to disable it.

Double Write Buffer Concerns

- ★ Doublewrite buffer concentrates writes to one main area of the ibdata1 tablespace.
 - ◆ Performance becomes more critical on storage media with limited write capacity and/or media that has no random IO penalty (SSDs).
- ★ In XtraDB it's possible to move the doublewrite buffer out of the main tablespace.

Adaptive Flushing

★ **Pictures speak louder than words:**



Adaptive Flushing (cont.)

- ★ When the log “fills up” InnoDB needs to flush dirty pages which could not be recovered once the recovery data is overwritten.
- ♦ This means it's very important not to get too far behind in flushing dirty pages!
- ♦ By default background operations are limited to 200 IOPS.

Adaptive Flushing (cont.)

- ★ Adaptive flushing is a heuristic-based algorithm to smooth out all I/O as much as possible:
 - ★ Now enabled by default. Can be disabled with `innodb_adaptive_flushing = 0`.
- ★ This feature also exists in XtraDB as “adaptive checkpointing”.
- ★ You must increase `innodb_io_capacity` to make adaptive flushing effective

Buffer pool LRU improvements

- ★ Changed from a pure LRU to something that is more resistant to large reads (BUG #45015).
 - ♦ Historically a backup process or table scan could evict all of the hot contents from the buffer pool.
- ★ New behaviour is to logically divide the LRU into two segments:
 - ♦ The front 5/8 of the LRU for 'hot' data.
 - ♦ The remaining 3/8 desirable for eviction.

LRU improvements (cont.)

- ★ Further configuration possible with:
 - ♦ `innodb_old_blocks_pct` (defaults to 37)
 - ♦ `innodb_old_blocks_time` (default is 0).
- ★ Old blocks time prevents a page from making it to the most-recently used end of the LRU.
 - ♦ Most likely situation is a table scan touches a page a few times in quick succession and then never uses it again.

InnoDB Status Sections

- ★ Semaphores
- ★ Deadlocks
- ★ Foreign Keys
- ★ Transactions
- ★ File I/O
- ★ Insert Buffer and Adaptive Hash
- ★ Log
- ★ Buffer Pool and Memory
- ★ Row Operations

SEMAPHORES

OS WAIT ARRAY INFO: reservation count 996617, signal count 628914
--Thread 1397500240 has waited at ../../storage/innobase/include/log0log.ic line 322 for 0.0000 seconds the semaphore:
Mutex at 0x19a112f0 created file log/log0log.c line 746, lock var 0
waiters flag 0
--Thread 1403357520 has waited at ../../storage/innobase/include/log0log.ic line 322 for 0.0000 seconds the semaphore:
Mutex at 0x19a112f0 created file log/log0log.c line 746, lock var 0
waiters flag 0
--Thread 1399363920 has waited at ../../storage/innobase/include/log0log.ic line 322 for 0.0000 seconds the semaphore:
Mutex at 0x19a112f0 created file log/log0log.c line 746, lock var 0
waiters flag 0
--Thread 1396967760 has waited at ../../storage/innobase/include/log0log.ic line 322 for 0.0000 seconds the semaphore:
Mutex at 0x19a112f0 created file log/log0log.c line 746, lock var 0
waiters flag 0
--Thread 1400961360 has waited at ../../storage/innobase/include/log0log.ic line 322 for 0.0000 seconds the semaphore:
Mutex at 0x19a112f0 created file log/log0log.c line 746, lock var 0
waiters flag 0
--Thread 1401760080 has waited at btr/btr0cur.c line 442 for 0.0000 seconds the semaphore:
S-lock on RW-latch at 0x19d2fc60 created in file dict/dict0dict.c line 1635
number of readers 0, waiters flag 0, lock_word: 100000
Last time read locked in file btr/btr0cur.c line 442
Last time write locked in file btr/btr0cur.c line 435
Mutex spin waits 4126944, rounds 24444227, OS waits 265505
RW-shared spins 566249, OS waits 598523; RW-excl spins 91973, OS waits 44173
Spin rounds per wait: 5.92 mutex, 25.80 RW-shared, 39.81 RW-excl

Semaphores

- ★ Can show if default spin lock was used, or OS wait. See "**Mutex spin waits 5672442, rounds 3899888, OS waits 4719**"
 - ♦ Spin lock burns CPU
 - ♦ OS Wait requires context switching back in.
- ★ If an OS wait was required, it should show where in the source this occurred.
 - ♦ "btr0sea" might be adaptive hash waits.
 - ♦ "trx0rseg" is rollback segment.
 - ♦ "buf0buf" is the main buffer pool mutex.

..

LATEST DETECTED DEADLOCK

060717 4:16:48

*** (1) TRANSACTION:

TRANSACTION 0 42313619, ACTIVE 49 sec, process no 10099, OS thread id 3771312

starting index read

mysql tables in use 1, locked 1

LOCK WAIT 3 lock struct(s), heap size 320

MySQL thread id 30898, query id 100626 localhost root Updating

update iz set pad='a' where i=2

*** (1) WAITING FOR THIS LOCK TO BE GRANTED:

RECORD LOCKS space id 0 page no 16403 n bits 72 index `PRIMARY` of table `test/iz` trx

id 0 42313619 lock_mode X locks rec but not gap waiting

Record lock, heap no 5 PHYSICAL RECORD: n_fields 4; compact format; info bits 0

0: len 4; hex 80000002; asc ;; 1: len 6; hex 00000285a78f; asc ;; 2: len

7; hex 00000040150110; asc @ ;; 3: len 10; hex 61202020202020202020; asc a

;;

..

*** (2) TRANSACTION:

TRANSACTION 0 42313620, ACTIVE 24 sec, process no 10099, OS thread id 4078512

starting index read, thread declared inside InnoDB 500

mysql tables in use 1, locked 1

3 lock struct(s), heap size 320

MySQL thread id 30899, query id 100627 localhost root Updating

update iz set pad='a' where i=1

*** (2) HOLDS THE LOCK(S):

RECORD LOCKS space id 0 page no 16403 n bits 72 index `PRIMARY` of table `test/iz`

trx id 0 42313620 lock_mode X locks rec but not gap

Record lock, heap no 5 PHYSICAL RECORD: n_fields 4; compact format; info bits 0

0: len 4; hex 80000002; asc ;; 1: len 6; hex 00000285a78f; asc ;; 2: len
7; hex 00000040150110; asc @ ;; 3: len 10; hex 61202020202020202020; asc a
;;

*** (2) WAITING FOR THIS LOCK TO BE GRANTED:

RECORD LOCKS space id 0 page no 16403 n bits 72 index `PRIMARY` of table `test/iz`

trx id 0 42313620 lock_mode X locks rec but not gap waiting

Record lock, heap no 4 PHYSICAL RECORD: n_fields 4; compact format; info bits 0

0: len 4; hex 80000001; asc ;; 1: len 6; hex 00000285a78e; asc ;; 2: len
7; hex 000000003411d9; asc 4 ;; 3: len 10; hex 61202020202020202020; asc a
;;

*** WE ROLL BACK TRANSACTION (2)

LATEST DETECTED DEADLOCK

090221 15:54:57

*** (1) TRANSACTION:

TRANSACTION 0 1736253712, ACTIVE 5 sec, process no 8189, OS thread id

2011474240 setting auto-inc lock

mysql tables in use 1, locked 1

LOCK WAIT 1 lock struct(s), heap size 368

MySQL thread id 6304968, query id 1702990793 10.x.x.x webuser update

INSERT INTO `my_table` (`event_id`, `updated_at`,

`news_feed_only`, `actee_id`, `extra_id`, `actor_id`, `actee_type`,

`user_id`, `actor_type`, `extra_type`, `event_type`, `created_at`)

VALUES(251183223, '2009-02-21 23:54:52', 0, 114040217, 742361767,

110698807, 'user', 114040217, 'user', 'bonus', 'standard',

'2009-02-21 23:54:52')

*** (1) WAITING FOR THIS LOCK TO BE GRANTED:

TABLE LOCK table `my_database/my_table` trx id 0 1736253712

lock mode AUTO-INC waiting


```

*** (2) TRANSACTION:
TRANSACTION 0 1736253703, ACTIVE (PREPARED) 5 sec, process no 8189, OS
thread id 1844541760
mysql tables in use 1, locked 1
2 lock struct(s), heap size 368, undo log entries 1
MySQL thread id 6304653, query id 1702990765 10.x.x.x webuser update
INSERT INTO `my_table` (`event_id`, `updated_at`,
`news_feed_only`, `actee_id`, `extra_id`, `actor_id`, `actee_type`,
`user_id`, `actor_type`, `extra_type`, `event_type`, `created_at`)
VALUES(251183379, '2009-02-21 23:54:51', 0, 113419017, 742361765,
115279155, 'user', 115279155, 'user', 'NewsStoryData', 'stanard',
'2009-02-21 23:54:51')
*** (2) HOLDS THE LOCK(S):
TABLE LOCK table `my_database/my_table` trx id 0 1736253703
lock mode AUTO-INC
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
TABLE LOCK table `my_database/my_table` trx id 0 1736254081
lock mode AUTO-INC waiting
TOO DEEP OR LONG SEARCH IN THE LOCK TABLE WAITS-FOR GRAPH
*** WE ROLL BACK TRANSACTION (2)

```

Deadlocks

- ★ Locking is done at the index level - so if your queries are not indexed well, it's not quite *row level* locking.
- ★ Two common issues you should be aware of:
 - ♦ Auto_increment scalability is improved in 5.1:
<http://dev.mysql.com/doc/refman/5.1/en/innodb-auto-increment>
 - ♦ Updating the same *row* near simultaneously can cause a deadlock. Both connections acquire a shared lock before one can escalate to an exclusive lock.

Deadlocks (cont.)

- ★ The “not quite row level” is next-key locking for binary log safety. Sometimes it’s interesting to spot one of the transactions holds way too many row locks:

```
---TRANSACTION 1931, ACTIVE 39 sec, OS thread id 4327256064  
fetching rows  
mysql tables in use 1, locked 1  
23137 lock struct(s), heap size 2062320, 1249317 row lock(s),  
undo log entries 1  
MySQL thread id 1, query id 67 localhost root Updating  
UPDATE my_locking_innodb SET a = REPEAT('b', 255) WHERE id2 =  
323255
```

..

LATEST FOREIGN KEY ERROR

060717 4:29:00 Transaction:

TRANSACTION 0 336342767, ACTIVE 0 sec, process no 3946, OS thread id 1151088992

inserting, thread

declared inside InnoDB 500

mysql tables in use 1, locked 1

3 lock struct(s), heap size 368, undo log entries 1

MySQL thread id 9697561, query id 188161264 localhost root update

insert into child values(2,2)

Foreign key constraint fails for table `test/child`:

,

CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`) REFERENCES `parent` (`id`) ON
DELETE CASCADE

Trying to add in child table, in index `par_ind` tuple:

DATA TUPLE: 2 fields;

0: len 4; hex 80000002; asc ;; 1: len 6; hex 000000000401; asc ;;

But in parent table `test/parent`, in index `PRIMARY`,

the closest match we can find is record:

PHYSICAL RECORD: n_fields 3; 1-byte offs TRUE; info bits 0

0: len 4; hex 80000001; asc ;; 1: len 6; hex 0000140c2d8f; asc - ;; 2: len 7;
hex

80009c40050084; asc @ ;;

Foreign Keys

- ★ This section (like others) is added as required.
- ★ Usually fairly simple to debug without developer locking information, i.e.
 - ★ MySQL thread id 9697561, query id 188161264 localhost root update
insert into child values(2,2)
Foreign key constraint fails for table **`test/child`**:
**CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`)
REFERENCES `parent` (`id`) ON DELETE CASCADE**

```

..
-----
TRANSACTIONS
-----
Trx id counter 0 80157601
Purge done for trx's n:o <0 80154573 undo n:o <0 0
History list length 6
Total number of lock structs in row lock hash table 0
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0 0, not started, process no 3396, OS thread id 1152440672
MySQL thread id 8080, query id 728900 localhost root
show innodb status
---TRANSACTION 0 80157600, ACTIVE 4 sec, process no 3396, OS thread id 1148250464,
thread declared inside InnoDB 442
mysql tables in use 1, locked 0
MySQL thread id 8079, query id 728899 localhost root Sending data
select sql_calc_found_rows * from b limit 5
Trx read view will not see trx with id>= 0 80157601, sees <0 80157597
---TRANSACTION 0 80157599, ACTIVE 5 sec, process no 3396, OS thread id 1150142816
fetching rows, thread
declared inside InnoDB 166
mysql tables in use 1, locked 0
MySQL thread id 8078, query id 728898 localhost root Sending data
select sql_calc_found_rows * from b limit 5
Trx read view will not see trx with id>= 0 80157600, sees <0 80157596

```

```
..
---TRANSACTION 0 80157598, ACTIVE 7 sec, process no 3396, OS thread id 1147980128
fetching rows, thread declared inside InnoDB 114
mysql tables in use 1, locked 0
MySQL thread id 8077, query id 728897 localhost root Sending data
select sql_calc_found_rows * from b limit 5
Trx read view will not see trx with id>= 0 80157599, sees <0 80157595
---TRANSACTION 0 80157597, ACTIVE 7 sec, process no 3396, OS thread id 1152305504
fetching rows, thread declared inside InnoDB 400
mysql tables in use 1, locked 0
MySQL thread id 8076, query id 728896 localhost root Sending data
select sql_calc_found_rows * from b limit 5
Trx read view will not see trx with id>= 0 80157598, sees <0 80157594
```

-- XtraDB --

TRANSACTIONS

Trx id counter 11D771A

Purge done for trx's n:o < 11D1AC8 undo n:o < 0

History list length 11436

LIST OF TRANSACTIONS FOR EACH SESSION:

---TRANSACTION 0, not started, process no 25247, OS thread id 1091651920

MySQL thread id 34, query id 5287216 localhost pz

show engine innodb status

---TRANSACTION 0, not started, process no 25247, OS thread id 1091385680

MySQL thread id 33, query id 90089 localhost root

---TRANSACTION 11D7719, ACTIVE 0 sec, process no 25247, OS thread id 1401227600 updating
or deleting

mysql tables in use 1, locked 1

4 lock struct(s), heap size 1216, 2 row lock(s), undo log entries 2

MySQL thread id 21, query id 5287414 localhost root Updating

UPDATE orders SET o_carrier_id = ? WHERE o_id = ? AND o_d_id = ? AND o_w_id = ?

Trx read view will not see trx with id >= 11D771A, sees < 11D7665

TABLE LOCK table `tpcc`.`new_orders` trx id 11D7719 lock mode IX

RECORD LOCKS space id 49 page no 1867 n bits 664 index `PRIMARY` of table

`tpcc`.`new_orders` trx id 11D7719 lock_mode X locks rec but not gap

TABLE LOCK table `tpcc`.`orders` trx id 11D7719 lock mode IX

RECORD LOCKS space id 50 page no 22715 n bits 456 index `PRIMARY` of table

`tpcc`.`orders` trx id 11D7719 lock_mode X locks rec but not gap

Transactions

- ★ We mentioned the purge earlier:
 - ★ `Trx id counter 0 80157601`
`Purge done for trx's n:o < 80154573 undo n:o < 0`
- ★ History list length is number of unpurged transactions in undo space - emerging as a tuning problem in new versions.
- ★ The list of running transactions is now better read from the information schema tables if you are using InnoDB Plugin.

```

-----
FILE I/O
-----
I/O thread 0 state: waiting for i/o request (insert buffer thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (read thread)
I/O thread 4 state: waiting for i/o request (read thread)
I/O thread 5 state: waiting for i/o request (read thread)
I/O thread 6 state: waiting for i/o request (read thread)
I/O thread 7 state: waiting for i/o request (read thread)
I/O thread 8 state: waiting for i/o request (read thread)
I/O thread 9 state: waiting for i/o request (read thread)
I/O thread 10 state: waiting for i/o request (read thread)
I/O thread 11 state: waiting for i/o request (read thread)
I/O thread 12 state: waiting for i/o request (read thread)
I/O thread 13 state: waiting for i/o request (read thread)
I/O thread 14 state: waiting for i/o request (read thread)
I/O thread 15 state: waiting for i/o request (read thread)
I/O thread 16 state: waiting for i/o request (read thread)
I/O thread 17 state: waiting for i/o request (read thread)
I/O thread 18 state: waiting for i/o request (write thread)
I/O thread 19 state: waiting for i/o request (write thread)
I/O thread 20 state: waiting for i/o request (write thread)
I/O thread 21 state: waiting for i/o request (write thread)
I/O thread 22 state: waiting for i/o request (write thread)
I/O thread 23 state: waiting for i/o request (write thread)
I/O thread 24 state: waiting for i/o request (write thread)
I/O thread 25 state: waiting for i/o request (write thread)
I/O thread 26 state: waiting for i/o request (write thread)
I/O thread 27 state: waiting for i/o request (write thread)
I/O thread 28 state: waiting for i/o request (write thread)
I/O thread 29 state: waiting for i/o request (write thread)
I/O thread 30 state: waiting for i/o request (write thread)
I/O thread 31 state: waiting for i/o request (write thread)
I/O thread 32 state: waiting for i/o request (write thread)
I/O thread 33 state: waiting for i/o request (write thread)
Pending normal aio reads: 0, aio writes: 0,
  ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
295336 OS file reads, 159620 OS file writes, 154669 OS fsyncs
738.63 reads/s, 16397 avg bytes/read, 1027.19 writes/s, 1010.44 fsyncs/s

```

File IO

- ★ These are physical reads/writes. Later on there's a section that shows logical operations.
 - ♦ Any of the "pending" counts always being high values could mean a loaded IO system.
- ★ You can count reads+writes in terms of IOPS, as well as number of fsyncs/s
 - ♦ Poorer IO systems really suffer on fsyncs, but this should be a non issue on a battery backed write-back cache.

..

INSERT BUFFER AND ADAPTIVE HASH INDEX

Ibuf for space 0: size 1, free list len 887, seg size 889, is not empty
Ibuf for space 0: size 1, free list len 887, seg size 889,
2431891 inserts, 2672643 merged recs, 1059730 merges
Hash table size 8850487, used cells 2381348, node heap has 4091 buffer(s)
2208.17 hash searches/s, 175.05 non-hash searches/s

-- XtraDB Example --

INSERT BUFFER AND ADAPTIVE HASH INDEX

Ibuf: size 4289, free list len 6928, seg size 11218,
689814 inserts, 67723 merged recs, 17341 merges
Hash table size 31874747, node heap has 9526 buffer(s)
25448.49 hash searches/s, 54424.21 non-hash searches/s

Insert / Change Buffer

- ★ All measurements in pages:
 - ♦ size = current size
 - ♦ seg size = size allocated up to
 - ♦ free list length = free space
- ★ Merges is approximately insert buffer efficiency.
- ★ Doesn't appear to be a way to measure update/delete efficiency, all are tied together

Adaptive Hash

- ★ There's really not anything you can do with these numbers, so it's really just informational.

```
..
---
LOG
---
Log sequence number 84 3000620880
Log flushed up to    84 3000611265
Last checkpoint at   84 2939889199
0 pending log writes, 0 pending chkp writes
14073669 log i/o's done, 10.90 log i/o's/second
```

-- XtraDB Example --

```
---
LOG
---
Log sequence number 43921309413
Log flushed up to    43921308508
Last checkpoint at   43497448671
Max checkpoint age    1303883551
Modified age          423860742
Checkpoint age        423860742
0 pending log writes, 0 pending chkp writes
154286 log i/o's done, 1008.78 log i/o's/second
```

LOG

- ★ The first three numbers show a lot of information on how the background flushing is going.
 - ♦ Log sequence number is current number “handed out”
 - ♦ Log flushed up to shows where we’ve written up to in the transaction logs.
 - ♦ Checkpointed up to shows how far behind out dirty page writing may be.

..

BUFFER POOL AND MEMORY

Total memory allocated 4648979546; in additional pool allocated 16773888
Buffer pool size 262144
Free buffers 0
Database pages 258053
Modified db pages 37491
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 57973114, created 251137, written 10761167
9.79 reads/s, 0.31 creates/s, 6.00 writes/s
Buffer pool hit rate 999 / 1000

..

BUFFER POOL AND MEMORY

Total memory allocated 16515072000; in additional pool allocated 0

Internal hash tables (constant factor + variable factor)

Adaptive hash index 411076168 (254997976 + 156078192)

Page hash 15938152

Dictionary cache 63827784 (63751024 + 76760)

File system 87336 (82672 + 4664)

Lock system 39898760 (39844312 + 54448)

Recovery system 0 (0 + 0)

Threads 414936 (406936 + 8000)

Dictionary memory allocated 76760

Buffer pool size 983040

Buffer pool size, bytes 16106127360

Free buffers 669543

Database pages 303971

Modified db pages 246726

Pending reads 1

Pending writes: LRU 0, flush list 0, single page 0

Pages read 295410, created 8561, written 12606

739.63 reads/s, 62.56 creates/s, 57.94 writes/s

Buffer pool hit rate 998 / 1000

LRU len: 303971, unzip_LRU len: 0

I/O sum[40576]:cur[526], unzip sum[0]:cur[0]

Buffer Pool

- ★ Total memory allocated is always interesting compared to buffer pool size * 16KiB
 - ♦ It really shows where InnoDB has stolen extra memory for overhead.
- ★ Modified db pages is “dirty pages”.
- ★ The hit rate shows the efficiency of the cache, and how often a page had to be loaded from disk.

Defining a “Working Set”

- ★ You don't need as much memory as you do data.
- ★ Data naturally has hotspots that need to be accessed frequently, but older data can just reside on disk.
- ★ Some applications have a working set of just a few percent - others are close to 100%.

..

ROW OPERATIONS

0 queries inside InnoDB, 0 queries in queue

1 read views open inside InnoDB

Main thread process no. 10099, id 88021936, state: waiting for server activity

Number of rows inserted 143, updated 3000041, deleted 0, read 24865563

0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s

-- XtraDB example --

ROW OPERATIONS

0 queries inside InnoDB, 0 queries in queue

33 read views open inside InnoDB

Main thread process no. 25247, id 1395636560, state: sleeping

Number of rows inserted 813805, updated 1627101, deleted 62432, read 6888247

5632.92 inserts/s, 11247.52 updates/s, 432.17 deletes/s, 47422.71 reads/s

Row Operations

- ★ Always interesting to see the logical rows operations versus what file IO/buffer pool sections said.
- ★ Great to graph this as an indication of capacity of the server.
- ★ The “main thread” is the server background thread. The diagnostics on it here are terrible, but it's expanded in XtraDB.

Show mutex status

- ★ Mainly developer related material is exposed:

```
-----+-----+-----+| dict0dict.c | 3753 |          0 | | fil0fil.c  
dict0mem.c | 90 |          0 | ..
```

- ★ It shows mutex waits - but does not expose much detail.