OpenEdge RDBMS Storage Internals

Rob Fitzpatrick
Software Architect / DBA Consultant
White Star Software
rf@wss.com

About White Star Software

For over 30 years, we have been helping companies around the world simplify the job of managing and monitoring the world's best OpenEdge applications.

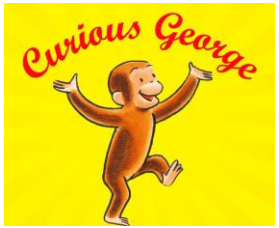
Our experts, combined with ProTop, the #1 OpenEdge monitoring and alerting tool, deliver unparalleled peace of mind for your OpenEdge environments.

Agenda

- Introduction
- Database storage hierarchy & definitions
- History
- Block types
- Block headers
- Block contents by type

Introduction

- Ask questions as we go
- Please correct me if I say something wrong
  - We'll all learn together!
- Thanks to George Potemkin and Mike Furgal for their valuable input!



Why this talk?

- When I started working with the OE RDBMS…
  - No formal training
  - Heard lots of lingo!
    - Areas, extents, blocks, clusters, RPB, BPC, objects, storage objects, recids, rowids, dbkeys, …
  - No clue what any of it meant!
- As technologists, we need a common language and *some* understanding of each other's domains, to collaborate on cross-disciplinary problems & solutions

Why this talk?

- Some of this material is “deep in the weeds”
- However, I strongly believe that most of it is essential knowledge for the working DBA
- Much of it is also useful for the non-DBA
  - Data-access developers
  - Application schema designers
  - Solution architects

Storage hierarchy: physical

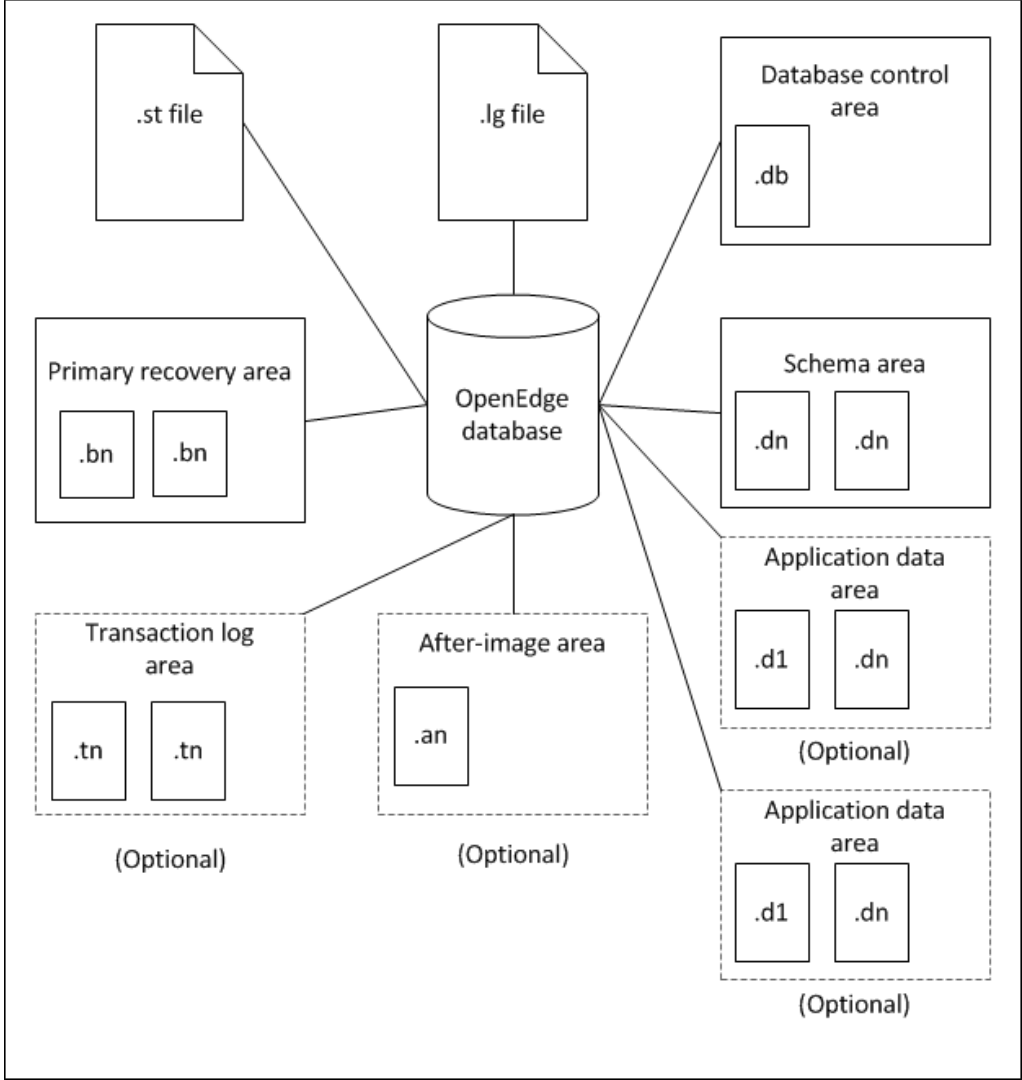
- Storage areas consist of files called *extents*
- Most area types can comprise one or more extents
  - Exceptions: control area (.db), after-image areas (.an)
- An extent is a collection of *blocks*
  - AI and BI areas each have their own *block size*, which can be changed offline
  - All other blocks have the *database block size* , which is fixed at DB creation
- Blocks are the fundamental unit of physical I/O

Storage hierarchy: physical

- Each block has a *block type* that determines the layout of the block's data, e.g.:
  - Index (IX) blocks store index entries from one index
  - Record (RM) blocks store *record fragments*
- A *record* (logical) consists of one or more record fragments
- A *record fragment* is a piece of a record that is stored in a record block
- Each fragment has a logical address (*rowid*)
- A record's rowid remains the same for its lifetime

Areas & Extents

- Each area has a record in _area
- Each extent has a record in _areaextent
- Parent-child relation
- These tables and their indexes are in .db (control area)
- Populated based on .st



Storage hierarchy: logical

- Databases consist of logical *storage areas*
- Storage areas have various attributes, including *area type*

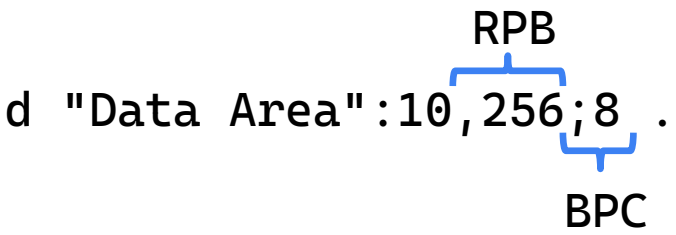
Where are Type 1 and Type 2?

_area._area-type			
3	Before image area	system	Undo/redo log
4	Transaction log	system	Used with 2PC
6	Data area	system / application	Storage objects
7	After image area	system	Recovery log

Focus of this talk

Storage hierarchy: logical

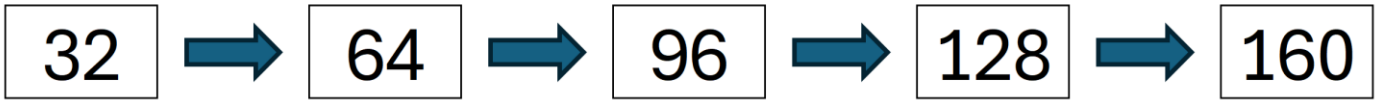
- Other important area attributes:
  - Records per block (RPB)
    - Maximum record fragments per record block
    - Stored in _area._area-recbits (2 ^ _area-recbits = RPB)
  - Blocks per cluster (BPC), aka cluster size
    - Type 2 architecture only
    - Stored in _area._area-clustersize: 8, 64, or 512
    - A cluster is a *logically*-contiguous collection of blocks
    - It is the allocation unit for *storage objects* in Type 2 areas



Storage hierarchy: logical

- Each block has a logical address called the *dbkey*
- Block dbkey = (previous block's dbkey) + RPB

E.g.:
RPB = 32



- Logical block # in area = dbkey / RPB
- Calculate dbkey from a rowid:
  - dbkey = rowid – (rowid modulo RPB)

Rowids and dbkeys

Example:

- 32-bit dbkeys and rowids
- RPB 8 (_area-recbits = 3)
- record block with dbkey 96

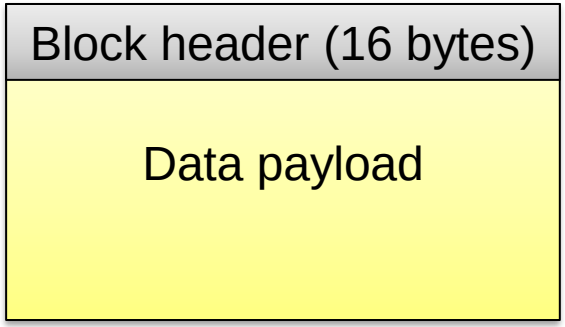
Type 1 versus Type 2

- Strictly speaking, Type 1 and Type 2 are not area types
- They are versions (or generations) of storage architecture for data areas (_area-type = 6)
- There is no reason to create new Type 1 areas!
  - Unless you are stuck on a pre-10.0A release. Sorry!
- Type 2 areas are superior in a variety of ways

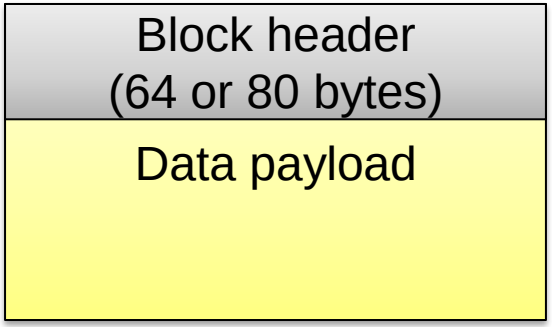
Block headers: Type 1 and Type 2

- At a high level, a block looks like this:

Type 1:



Type 2:



- The format of the payload varies by block type
- The format and length of the header varies
  - Type 1 vs. Type 2 vs. Type 2 extended
- Numeric content is encoded in machine order
  - E.g.: little-endian on x64, big-endian on POWER

Type 1 versus Type 2

- Type 1:
  - Old architecture
  - Small block headers (16 bytes)
  - 32-bit addresses (dbkeys/rowids)
    - Very limited maximum area size; it is a function of RPB
  - Limits use of modern features
  - An area is a *collection of blocks* (and some area meta-data)
  - Space management done at the area level
  - Record blocks may contain fragments from any table in the area

Type 1 versus Type 2

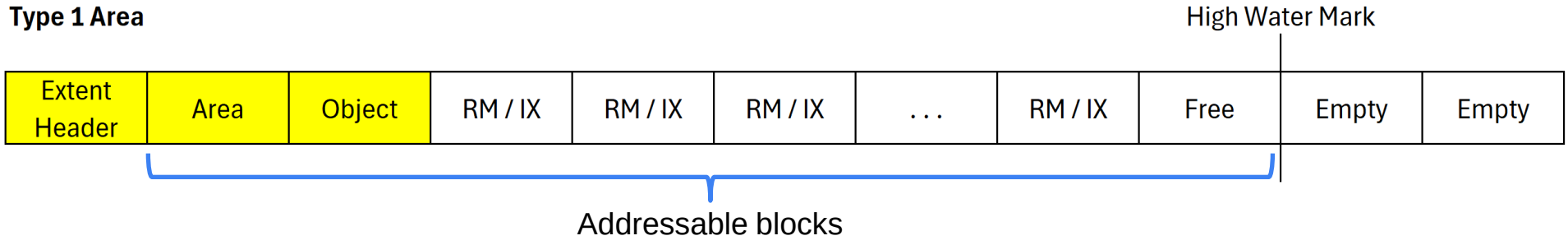
- Type 2:
  - New architecture (OE 10+)
  - Extended block headers (64+ bytes)
  - Blocks are grouped into units called clusters
  - Required or preferred for modern RDBMS features
    - CDC, Auditing, TDE, Table Partitioning, Multi-tenant
  - Superior for reliability, maintenance, performance, scalability
  - Addresses (dbkeys/rowids):
    - 10.0A – 10.1A: 32-bit
    - 10.1B+: 64-bit
      - Max area size limited by (max. extent size * max # of extents)

Type 1 versus Type 2

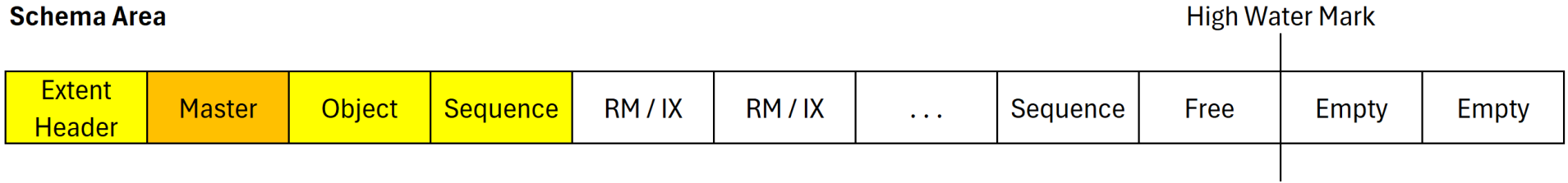
- Type 2 (continued):
  - An area is a *collection of storage objects* (and some area meta-data)
  - A storage object is a chain of clusters, which in turn are chains of blocks
  - A storage object is the physical storage for a schema object: table, index, or LOB column
    - Also: table instances (MT); partitions & local indexes (TP)
  - Space management is done at the storage object level
  - The head of the cluster chain is the object block
    - Stored in _storageobject._object-block
    - Also stored in area's Object List block

Type 1 area layout

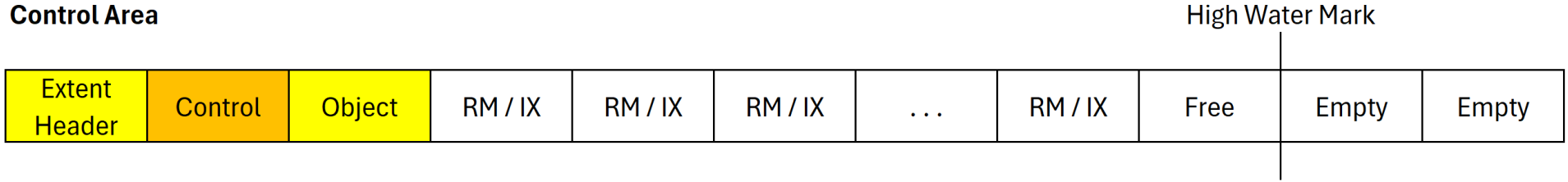
Type 1 Area



Schema Area



Control Area



Type 2 area layout

Type 2 Area (cluster size 8)

Area Meta-data (could grow, given enough storage objects)

Extent Header	Area	Object	Cluster List	Object Allocation	Cluster Allocation	Object List	Free / OL

Table 1 Storage Object

Object	Cluster List	Object Allocation	Cluster Allocation	RM	RM	RM	RM

Index 1 Storage Object

Object	Cluster List	Object Allocation	Cluster Allocation	IX	Free	Free	Free

A brief history

- 8.x and earlier:
  - All data is in one logical container *(dbname*.db) (Type 1 architecture)
  - Dbkeys/rowids are unique database-wide
- 9.x:
  - Multiple data areas in a database (Type 1)
  - Control area, Schema area, application data areas
  - Dbkeys/rowids are unique per area
- 10.0A: Type 2 data areas
  - New data structures: Type 2 blocks headers; clusters
  - Addresses (dbkeys/rowids) still 32-bit

A brief history

- 10.1B:
  - Type 2 areas now have 64-bit addresses
- 11.0:
  - Multi-tenancy
  - MT tables have multiple table instances
- 11.4:
  - Table partitioning
  - ROWID() extended to include 16-bit partitionId to ensure uniqueness within the table
  - Note: recids may not be unique within a partitioned table; avoid RECID() function for application data

Block types

bk_type	Description
1	Master block
2	Index (IX) block
3	Record (RM) block
4	Free block
6	Sequence block
7	Empty block
9	Area block

bk_type	Description
12	Object block
13	Control block
14	Object List block
15	Cluster Allocation block
16	Cluster List block
17	Object Allocation block
254	Extent Header block

Type 1 headers

Block dbkey (32-bit)	Block type	Chain type	DB bkup ctr at last blk update	Dbkey of next block in chain (32-bit)	Block update counter

16 bytes total:

#	Field name	Bytes	Notes
1	bk_dbkey	4	Dbkey (32-bit)
2	bk_type	1	Block type
3	bk_frchn	1	Chain type
4	bk_incr	2	DB backup counter at last block update
5	bk_nextf	4	Dbkey of the next block in the chain (if any)
6	bk_updctr	4	Block update counter (referenced in BI/AI notes)

Type 2 headers

- First six fields *are* the Type 1 header
- Type 2 extends it with more fields; 64 bytes total
- Note the duplicate dbkey and nextf fields
- First or last block in a cluster: header extends to 80 bytes total

Block dbkey (32-bit)		Block type	Chain type	DB bkup ctr at last blk update	Dbkey of next block in chain (32-bit)			Block update counter
Block checksum	Block header size	Object ID		Object type	Dbkey of object block (64-bit)			
Block dbkey (64-bit)					Dbkey of next block in chain (64-bit)			
Block last BI note					Partition ID (11.0+)	DB bckp ctr	reserved	reserved
Transaction ID (first block in cluster) Dbkey of first block in next cluster (last block in cluster)					Serial number (first block in cluster) Dbkey of first block in previous cluster (last block in cluster)			

Type 2 header

#	Field name	
7	bkChecksum	
8	bkHeaderSize	
9	objectId	
10	objectType	
11	bkObjDbkey	
12	bkDbkey	
13	bkNextf	
14	bkLastBiNote	
15	partitionId	
16	bk_incr_HIGH	

```
0000 bk_dbkey:      0x00000600          1536
     bk_type:       0x03                3 (Data Block)
     bk_frchn:      0x01                1 (RMCHN)
     bk_incr:       0x0001              1
     bk_nextf:      0x00000680          1664
     bk_updctr:     0x00000002          2

0010 bkCheckSum:    0x2214              8724
     bkHeaderSize:  0x0040              64
     objectId:      0x0001              1
     partitionId:   0x0000              not there
     objectType:    0x0001              1
     bkObjDbkey:    0x0000000000000400  1024
0020 bkDbkey:       0x0000000000000600  1536
     bkNextf:       0x0000000000000680  1664
0030 bkLastBiNote:  0x0000000000000000  0
     partitionId:   0x0000              0
     bk_incr_HIGH:  0x00                0
```

Type 2 header fields

#	Field name	Bytes	Notes
17	Reserved	1	
18	Reserved	4	
			First block in cluster:
19	transactionId	4	
20	serialNumber	8	
			Last block in cluster:
19	nextCluster	8	Dbkey of first block in next cluster
20	prevCluster	8	Dbkey of first block in previous cluster

Block contents by block type

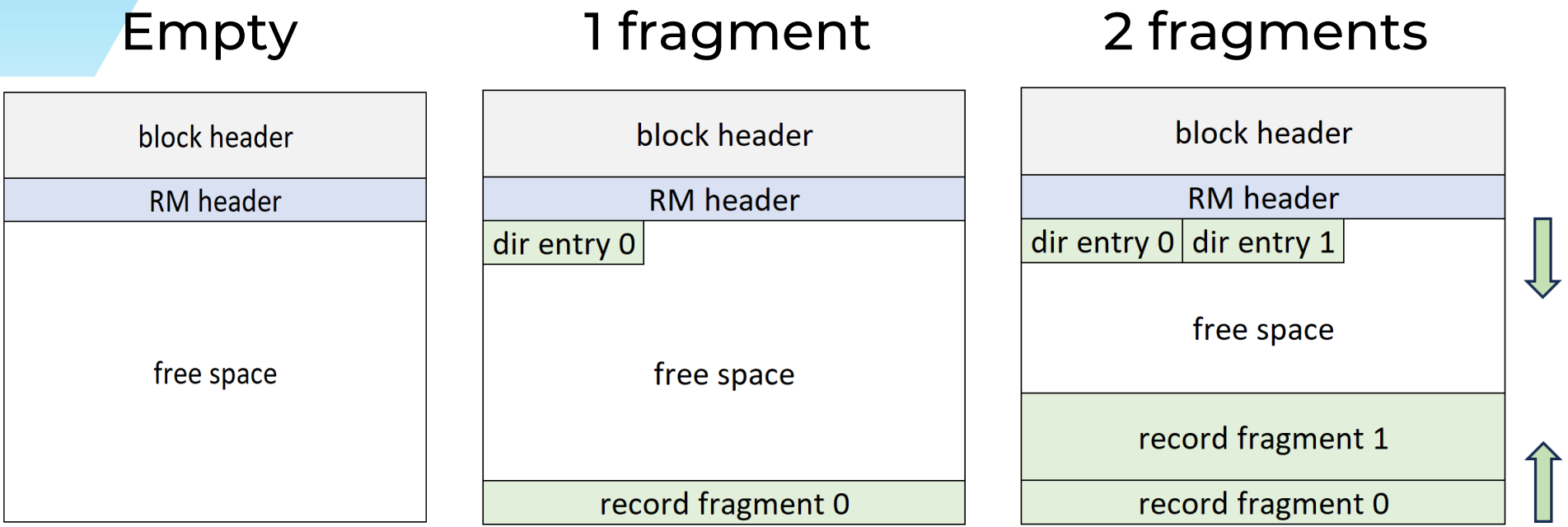
- You can look at block headers and contents with `proutil dbname -C dbrpr`
- **WARNING:**
  In the wrong hands, this utility can permanently damage or destroy a database
- It is unsupported & undocumented
- Use it on a sports db, not prod!

```
DATABASE REPAIR MENU
        ____________________


 1. Database Scan Menu
 2. Test One or More Indexes
 3. Remove Bad Record Fragment
 4. Dump Block
 5. Load Block
 6. Copy Bytes Between Files
 7. Load RM Dump File
 8. Reformat Block to a Free Block
 9. Change Current Working Area
10. Display the Free Chain
11. Display the RM Chain
12. Display the Index Delete Chain
13. Display Block Contents
14. Display Record Contents
15. Display Cluster Chain (Type II
Area)
16. Scan/Fix block checksum (Type
II Area)
 P. Print Info Menu

 Q. Quit
```

Record block (3)



- Row directory grows down into free space
- Record fragments grow up into free space

Record block (3) – RM header

- Begins after the block header
- Describes the row directory and the free space
- 3 fields:
  - **numdir**
    - Highest row directory entry used
    - 1 byte (0 – 255)
  - **freedir**
    - Number of available row directory entries
    - 1 byte (0 – 255)
  - **free:**
    - Bytes of contiguous free space in the block
    - 2 bytes

Record block (3) – row directory

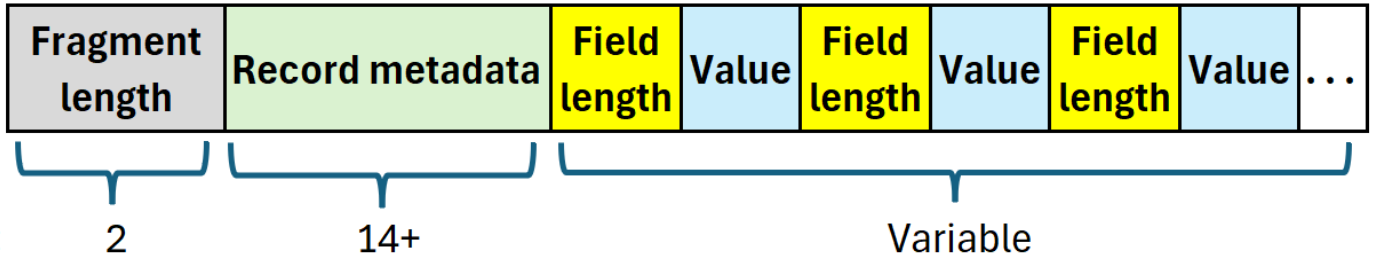
- Serves as a mapping layer of logical addresses (low-order bits of rowid) to physical (byte offset of fragment in block)
- Initially empty
- Grows into free space as needed
  - Up to RPB # of entries
- Each entry is 16 bits (2 bytes)
  - Consists of a bitmap and a byte offset

Record block (3) - row dir entries

- 16 bits:
  - High-order 3 bits: fragment meta-data bitmap
    - Bit 1:
      - 1 = “hold flag”: rowid placeholder for txn rollback
    - Bit 2:
      - 1 = “has a continuation fragment in another block”
      - i.e. it isn’t the last fragment in the record
    - Bit 3:
      - 1 = “is a continuation fragment”
      - i.e. it isn’t the first fragment in the record
  - Low-order 13 bits:
    - Byte offset from start of block to start of fragment

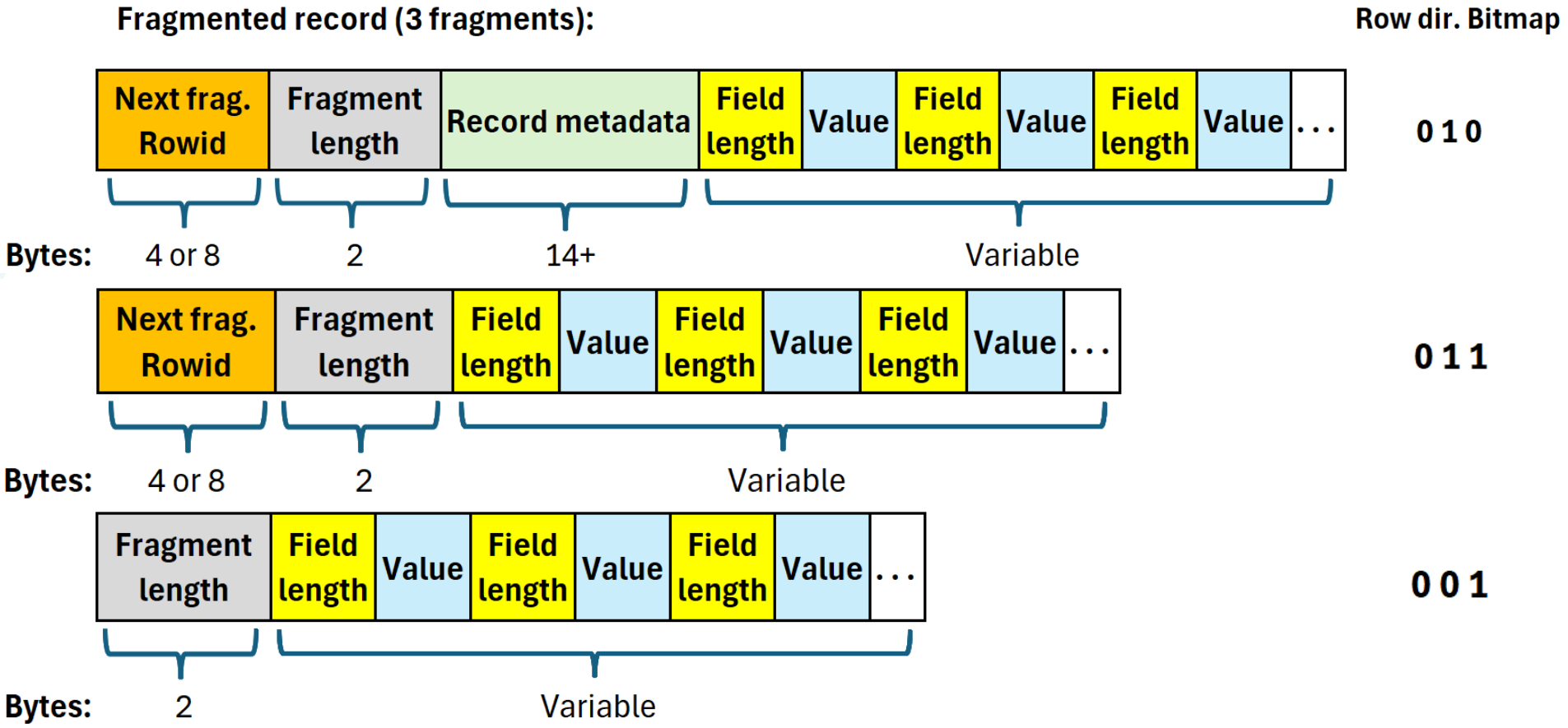
Record block (3) – record fragments

Unfragmented record:



- Record numeric data is always encoded big-endian
  - Makes binary dumps portable and fast (no translation required)
- Record metadata:
  - Field map
  - Skip table (conditional; only when 15 or more fields; 5+ bytes)
  - Table schema version #
  - Table number
  - Table field count

Record block (3) – record fragments



Index block (2)

- An index block stores a node of the index B-tree
  - Root, non-leaf, or leaf level
- A block stores *index entries*, sorted in key order
- An index entry consists of entry meta-data, a key value, and one or more rowids of records with that key value
  - There can be only one rowid if the index is unique
- Hard to visualize, as data compression is used in several ways

Index block (2)

- Mike Furgal's 2018 PUG Challenge talk: Index Internals: The Engine

https://pugchallenge.org/wp-content/uploads/2024/04/downloads2018/Furgal_IndexInternals.pptx

- pugchallenge.org
  - More | Archive
  - PUG Challenge Americas 2018

Object block (12)

- Each area contains an object block (3rd block in area)
- Type 2 storage objects begin with an object block
- Payload contains space-management fields
  - Chain info
    - RM / Free / Index Delete
  - Total blocks
  - HWM
  - Table Reorg info
  - Etc.

Object block (12)


```
OBJBLK:
004c totalBlocksOld:                0x00000000         0
0050 hiWaterBlockOld:               0x00000000         0
0054 chainFirst[FREECHN]:           0x0000000000000000 0
005c chainFirst[RMCHN]:             0x0000000000000c00 3072
0064 chainFirst[LOCKCHN]:           0x0000000000000000 0
006c numBlocksOnChainOld[FREECHN]: 0x00000000         0
0070 numBlocksOnChainOld[RMCHN]:    0x00000000         0
0074 numBlocksOnChainOld[LOCKCHN]: 0x00000000         0
0078 chainLast[FREECHN]:            0x0000000000000000 0
0080 chainLast[RMCHN]:              0x0000000000000f00 3840
0088 chainLast[LOCKCHN]:            0x0000000000000000 0
0090 objectId:                      0x0001             1
0092 objectType:                    0x0001             1
0094 serialNumber:                  0x0000000000000001 1
009c firstFreeCluster:              0x0000000000000000 0
00a4 lastFreeCluster:               0x0000000000000000 0
```

```
00ac totalBlocks:                   0x0000000000000008 8
00b4 hiWaterBlock:                  0x000000000000000f 15
00bc numBlocksOnChain[FREECHN]:     0x0000000000000000 0
00c4 numBlockOnsChain[RMCHN]:       0x0000000000000004 4
00cc numBlocksOnChain[LOCKCHN]:     0x0000000000000000 0
00d4 partitionId:                   0x0000             0
00d6 flags:                         0x00               0
00d7 objSpare1:                     0x00               0
00d8 objSpare2:                     0x00000000         0
00dc tableReorgAnchor:              0x0000000000000000 0
00e4 tableReorgRecs:                0x0000000000000000 0
00ec tableReorgRecid:               0x0000000000000000 0
00f4 tableReorgArea:                0x00000000         0
00f8 tableReorgIndex:               0x0000             0
00fa tableReorgFlag1:               0x00               0
00fb tableReorgFlag2:               0x00               0
00fc tblReorgReserved:              "" (16)
```

- Sample 12.8 Object block
- Layout varies by release

Master block (1)

- Contains a wide variety of fields related to database configuration and state
  - Varies by version
- It is the second block in the schema area
  - Dbkey 64 (8 KB blocks) or 32 (other block sizes)
- Takes the place of an area block in schema area
- Selected fields:
  - DB backup counter, “tainted” flag, DB time stamps, backup time stamps, AI/BI block size, BI cluster size, last TrID, log archiving, AI, Replication; many more!

Extent Header block (254)

- First block of every extent
- Dbkey is always zero (not addressable)
- Not loaded into buffer pool
- Always has a Type 1 block header
- Example data payload:

```
EXTHDR:
    dbVersion:          0x20b8 184 8192
    extentType:         0x0040 64  (DBMDB - multi-file data file)
    dateVerification: 0
    creationDate:       [0]=0x6043e4a4 (Mar 06 15:23:00 2021)
                        [1]=0x6043e4a4 (Mar 06 15:23:00 2021)
    lastOpenDate:       [0]=0x6043e7f2 (Mar 06 15:37:06 2021)
                        [1]=0x60464237 (Mar 08 10:26:47 2021)
```

Sequence block (6)

- Sequence blocks are in the schema area
- The first sequence block is always 4th block
  - Dbkey: 96 (4 KB blocks); 192 (8 KB blocks)
- Sequence blocks are “chained” together

```
Schema Area (6)  Extent 1  Block 4
-----------
0000 bk_dbkey:     0x00000060          96
     bk_type:      0x06                6 (Sequence Block)
     bk_frchn:     0x7f                127 (NOCHN)
     bk_incr:      0x0001              1
     bk_nextf:     0x00000920          2336
     bk_updctr:    0x000001f5          501
```

```
Schema Area (6)  Extent 1  Block 74
-----------
0000 bk_dbkey:     0x00000920          2336
     bk_type:      0x06                6 (Sequence Block)
     bk_frchn:     0x7f                127 (NOCHN)
     bk_incr:      0x0001              1
     bk_nextf:     0x00000000          0
     bk_updctr:    0x000001f9          505
```

- But there is no “sequence” chain type
- Sequence blocks are always in the buffer pool

Sequence block (6)

- Values are stored in _seq-num order
- *Fixed-width* integers, either 32-bit or 64-bit
  - Based on feature ID 11:

```
Database Features
   ID   Feature                                  Active  Details
 ____   _______________________________________  ______  _______
    5   Large Files                              Yes
    9   64 Bit DBKEYS                            Yes
   10   Large Keys                               Yes
   11   64 Bit Sequences                         Yes
```

- In 12.x, all sequence values are 64-bit
- Older releases limited to 2 sequence blocks
  - Limits are much higher now; 32,000 sequence defn's

Sequence block (6)

Hex dump sample of a sequence block with 64-bit sequences:

```
00003000: 6000 0000 067f 0100 e008 0000 f501 0000  `...............
00003010: 0100 0000 0000 0000 0100 0000 0000 0000  ................
00003020: 0100 0000 0000 0000 0100 0000 0000 0000  ................
00003030: 0100 0000 0000 0000 0100 0000 0000 0000  ................
00003040: 0100 0000 0000 0000 0100 0000 0000 0000  ................
00003050: 0100 0000 0000 0000 0100 0000 0000 0000  ................
etc.
```

And a sample from a 10.1A database with 32-bit sequences:

```
00000c00: 6000 0000 067f 0100 0000 0000 ee02 0000  `...............
00000c10: 0100 0000 0100 0000 0100 0000 0100 0000  ................
00000c20: 0100 0000 0100 0000 0100 0000 0100 0000  ................
00000c30: 0100 0000 0100 0000 0100 0000 0100 0000  ................
00000c40: 0100 0000 0100 0000 0100 0000 0100 0000  ................
00000c50: 0100 0000 0100 0000 0100 0000 0100 0000  ................
etc.
```

Object List block (14)

- 7th block in a Type 2 area
  - If it fills, more will be added and chained together
- Contains a list of the storage objects in the area and their attributes

```
OBJLISTBLK:
    nextBlock:     0x0000000000000000 0
    prevBlock:     0x0000000000000000 0
    numObjects:    0x0000001a         26
    objListEntry:
      Obj      Serial Num    Type       Obj DBkey   Obj ID   Partition
       0                0      8             512      0      0
       1                1      1            2048      1      0
       2                3      1            4096      2      0
       3                5      1            6144      3      0
       4                9      1            8192      4      0
       5               15      1           10240      5      0
```

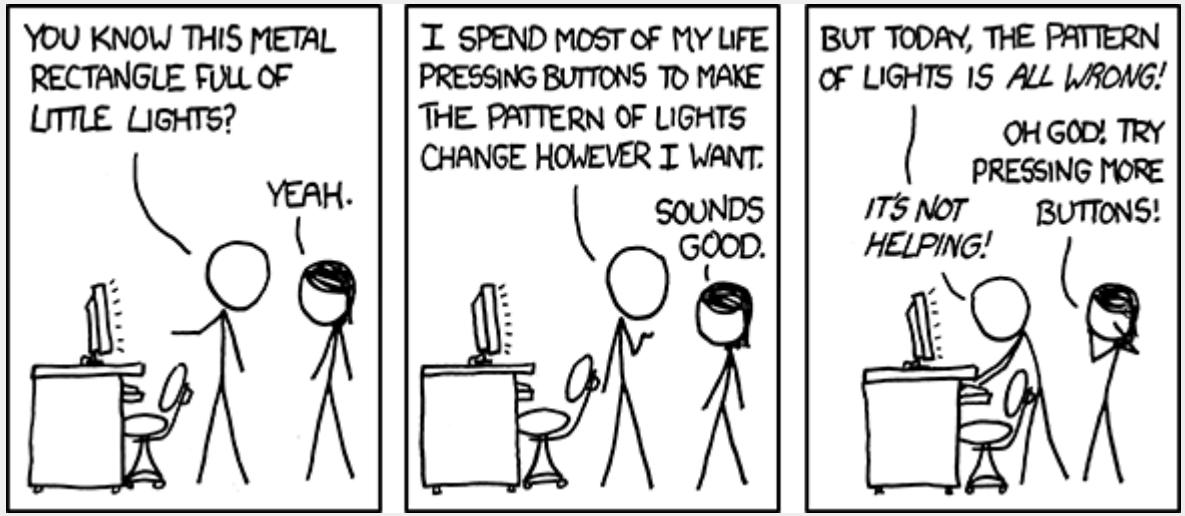
Free block (4)

- Unused blocks below the HWM
- Will be reformatted to other block types as needed
- Contains only a block header

Empty block (7)

- Unused blocks above the HWM
- Will be reformatted to other block types as needed, if the HWM is raised
- Contains no data (all zero bytes)

Questions?



https://xkcd.com/722

Thank you!