Transaction Internals

Julian Dyke

Independent Consultant

Web Version

Agenda

- Transactions
  - Redo
  - Undo
  - Rollback
  - Read Consistency
- Undo-based Features
  - ORA_ROWSCN
  - Flashback

Examples

- All examples in this presentation are based on cricket

- The following table has been used in all examples in this presentation

SCORE

TEAM	VARCHAR2(30)
RUNS	NUMBER
WICKETS	NUMBER

- The table has no indexes

Transactions

- A transaction is a set of DML statements executed sequentially by a session

- Starts with the first of the following statements executed by the session:

  - INSERT
  - UPDATE
  - DELETE
  - MERGE
  - SELECT FOR UPDATE
  - LOCK TABLE

- Ends with either a COMMIT or ROLLBACK

Transactions

- ACID properties
  - Atomicity - all changes made by the transaction are either committed or rolled back
  - Consistency - the database is transformed from one valid state to another
  - Isolation - results of the transaction are invisible to other transactions until the transaction is complete
  - Durability - once the transaction completes, the results of the transaction are permanent
- In Oracle transactions can also be:
  - recursive
  - audit
  - autonomous

Redo

- All database changes generate redo
  - Records changes made to
    - Data and index segments
    - Undo segments
    - Data dictionary
    - Control files (indirectly)

- Redo is used:
  - During recovery of database
    - Instance recovery
    - Media recovery

Undo

- Ensures ACID properties are maintained for each transaction

- Contains changes required to reverse redo including:
  - changes to data and index blocks
  - changes to transaction lists
  - changes to undo blocks

- All undo operations generate redo
  - Not all redo operations generate undo

- Implemented using undo segments
  - Manually-managed (rollback segments)
  - System-managed (Oracle 9.0.1 and above)

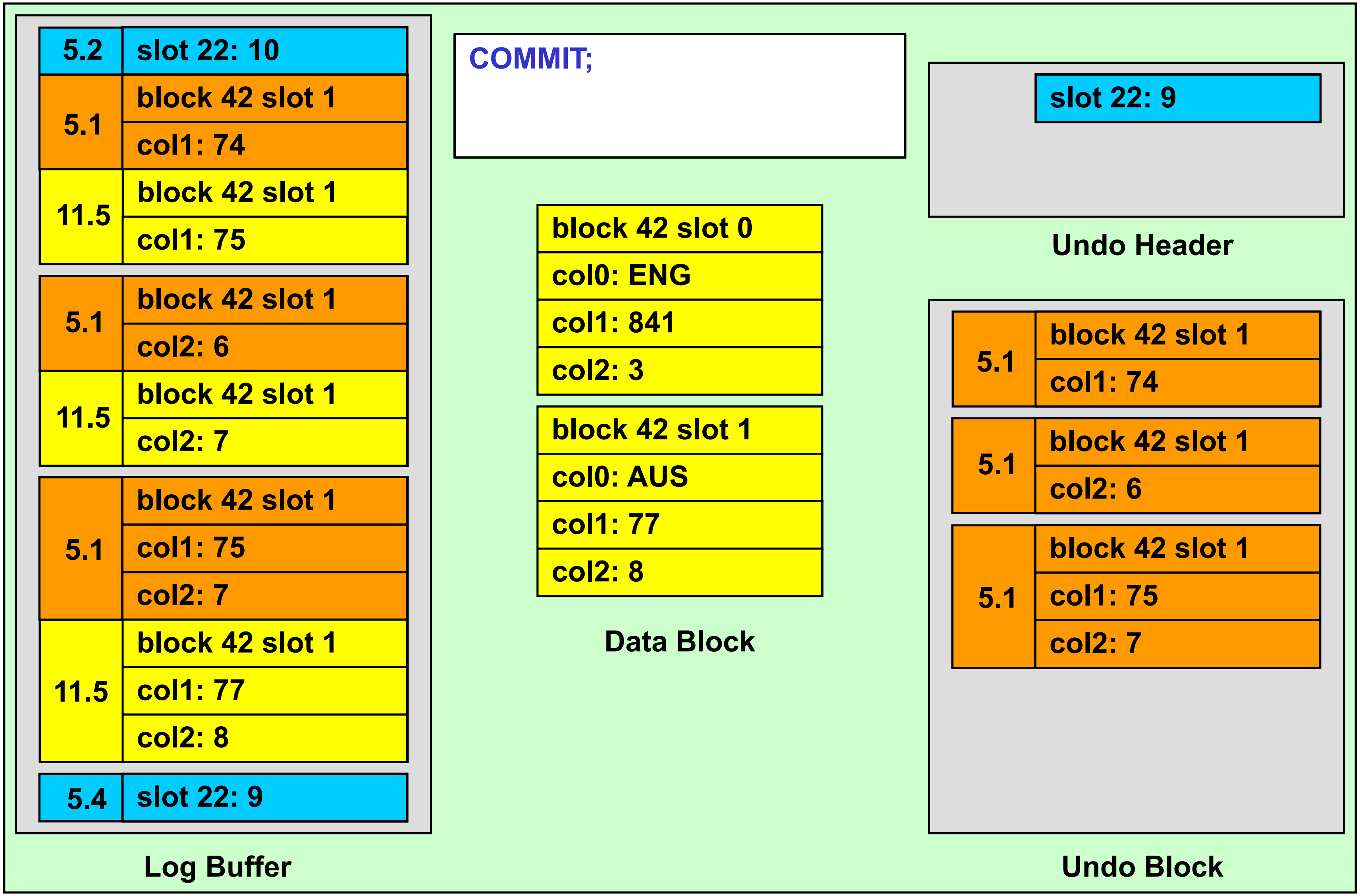
Undo

- Used to rollback uncommitted transactions
  - By session issuing ROLLBACK statement
  - By PMON on behalf of failed session
  - During instance recovery
  - During media recovery

- Used to implement read-consistency
  - Uncommitted changes cannot be seen by other sessions

- Used to implement flashback
  - Oracle 9.0.1 and above

Redo and Undo



5.2	slot 22: 10
5.1	block 42 slot 1
	col1: 74
11.5	block 42 slot 1
	col1: 75
5.1	block 42 slot 1
	col2: 6
11.5	block 42 slot 1
	col2: 7
5.1	block 42 slot 1
	col1: 75
	col2: 7
11.5	block 42 slot 1
	col1: 77
	col2: 8
5.4	slot 22: 9

Log Buffer

COMMIT;

block 42 slot 0
col0: ENG
col1: 841
col2: 3
block 42 slot 1
col0: AUS
col1: 77
col2: 8

Data Block

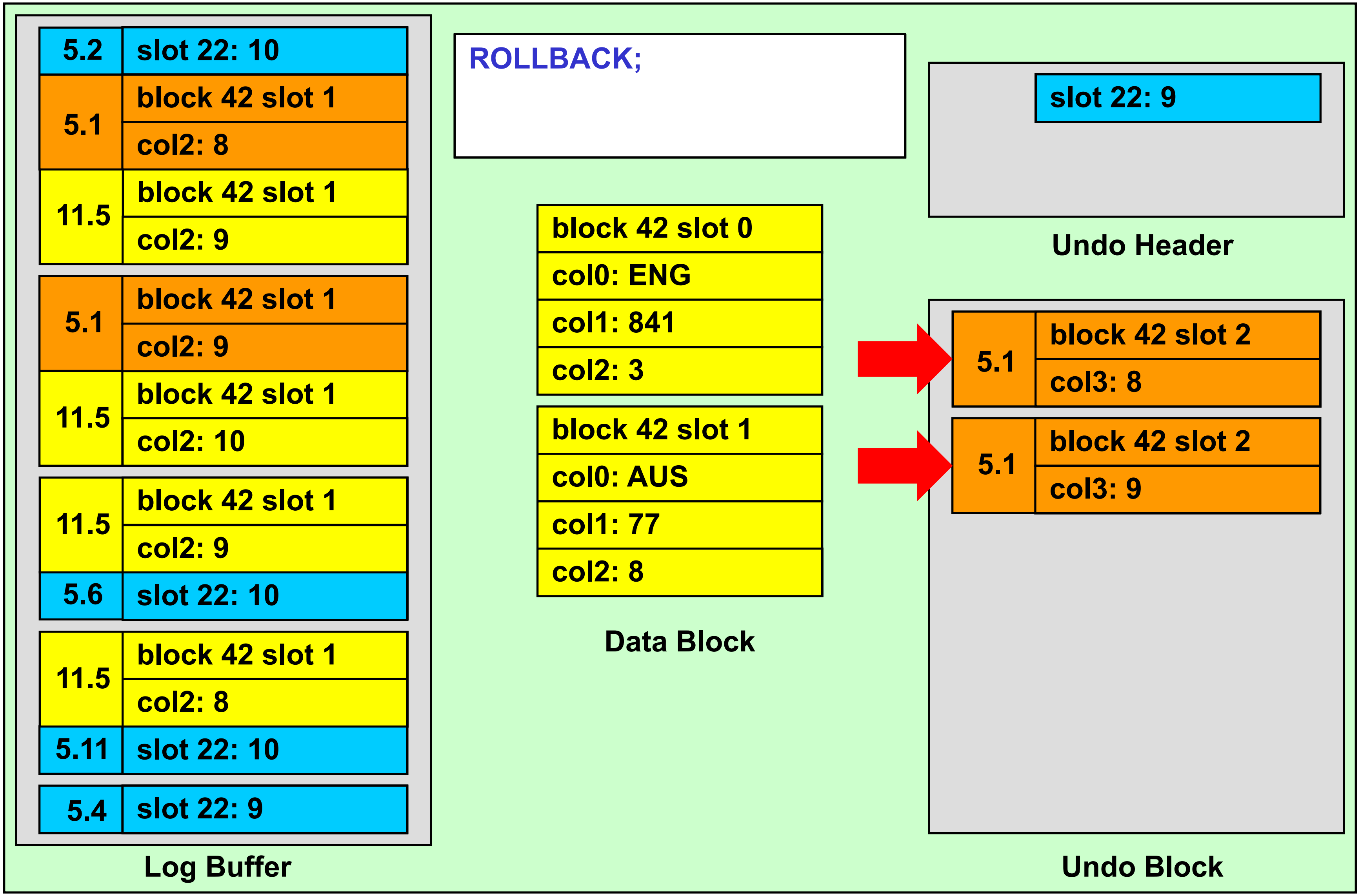
slot 22: 9

Undo Header

5.1	block 42 slot 1
	col1: 74
5.1	block 42 slot 1
	col2: 6
5.1	block 42 slot 1
	col1: 75
	col2: 7

Undo Block

Rollback



ROLLBACK;

Log Buffer

5.2	slot 22: 10
5.1	block 42 slot 1
	col2: 8
11.5	block 42 slot 1
	col2: 9
5.1	block 42 slot 1
	col2: 9
11.5	block 42 slot 1
	col2: 10
11.5	block 42 slot 1
	col2: 9
5.6	slot 22: 10
11.5	block 42 slot 1
	col2: 8
5.11	slot 22: 10
5.4	slot 22: 9

Data Block

block 42 slot 0
col0: ENG
col1: 841
col2: 3
block 42 slot 1
col0: AUS
col1: 77
col2: 8

Undo Header

slot 22: 9

Undo Block

5.1	block 42 slot 2
	col3: 8
5.1	block 42 slot 2
	col3: 9

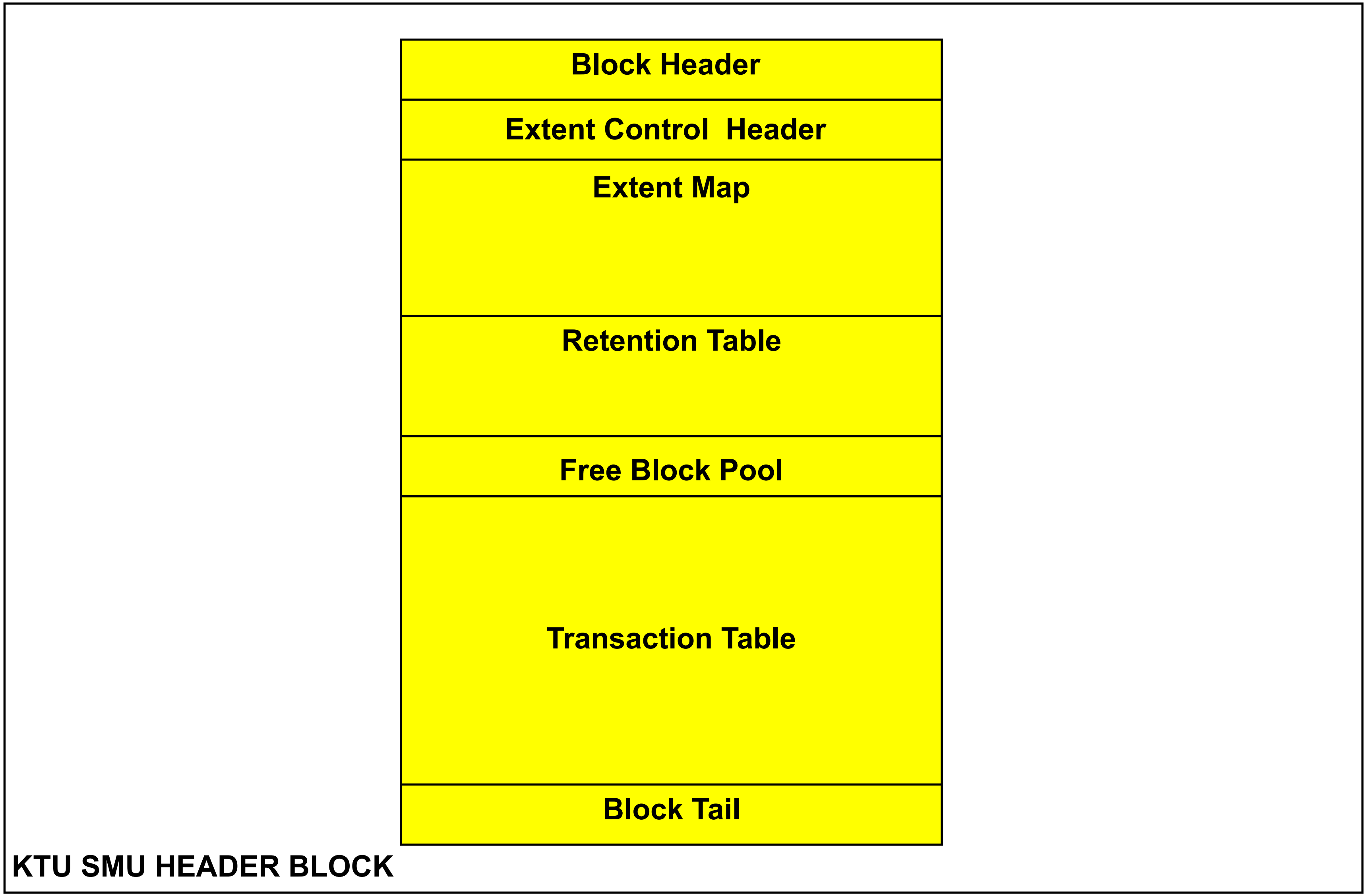
Undo Segment Header

- Undo segments are allocated at instance startup
  - Undo segments can be added dynamically

- Each undo segment header contains
  - Pool of free undo extents
  - Set of undo slots

- One undo slot is allocated to each transaction
  - Undo slot contains list of undo extents
  - Extents can migrate from one undo segment to another
  - Undo slots are used cyclically
    - remain in header as long as possible
    - reduces probability of ORA-01555: Snapshot too old

Undo Segment Header Structure



Block Header

Extent Control Header

Extent Map

Retention Table

Free Block Pool

Transaction Table

Block Tail

KTU SMU HEADER BLOCK

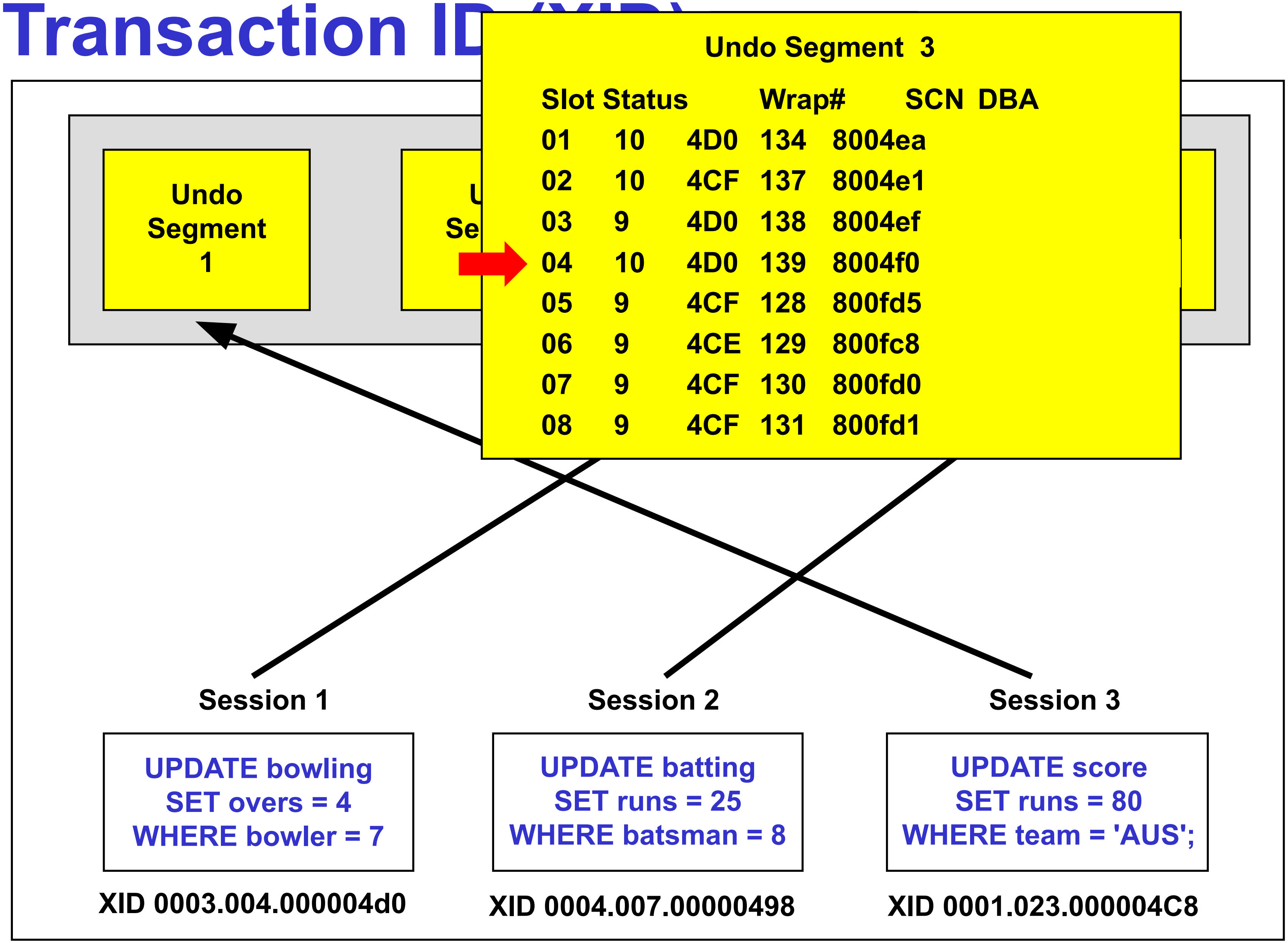
Transaction ID (XID)

- Every transaction has a unique ID based on
  - Undo segment number
  - Undo segment slot number
  - Undo segment sequence number (wrap)

- A transaction ID (XID) is allocated to each transaction during the first DML statement. For example:
  - 0002.028.000004DA

- Details about transaction can be found in V$TRANSACTION
  - XIDUSN Segment number
  - XIDSLOT Slot number
  - XIDSQN Sequence number

Transaction ID (XID)



Undo Segment 3

Slot	Status	Wrap#	SCN	DBA
01	10	4D0	134	8004ea
02	10	4CF	137	8004e1
03	9	4D0	138	8004ef
04	10	4D0	139	8004f0
05	9	4CF	128	800fd5
06	9	4CE	129	800fc8
07	9	4CF	130	800fd0
08	9	4CF	131	800fd1

Session 1

UPDATE bowling
SET overs = 4
WHERE bowler = 7

XID 0003.004.000004d0

Session 2

UPDATE batting
SET runs = 25
WHERE batsman = 8

XID 0004.007.00000498

Session 3

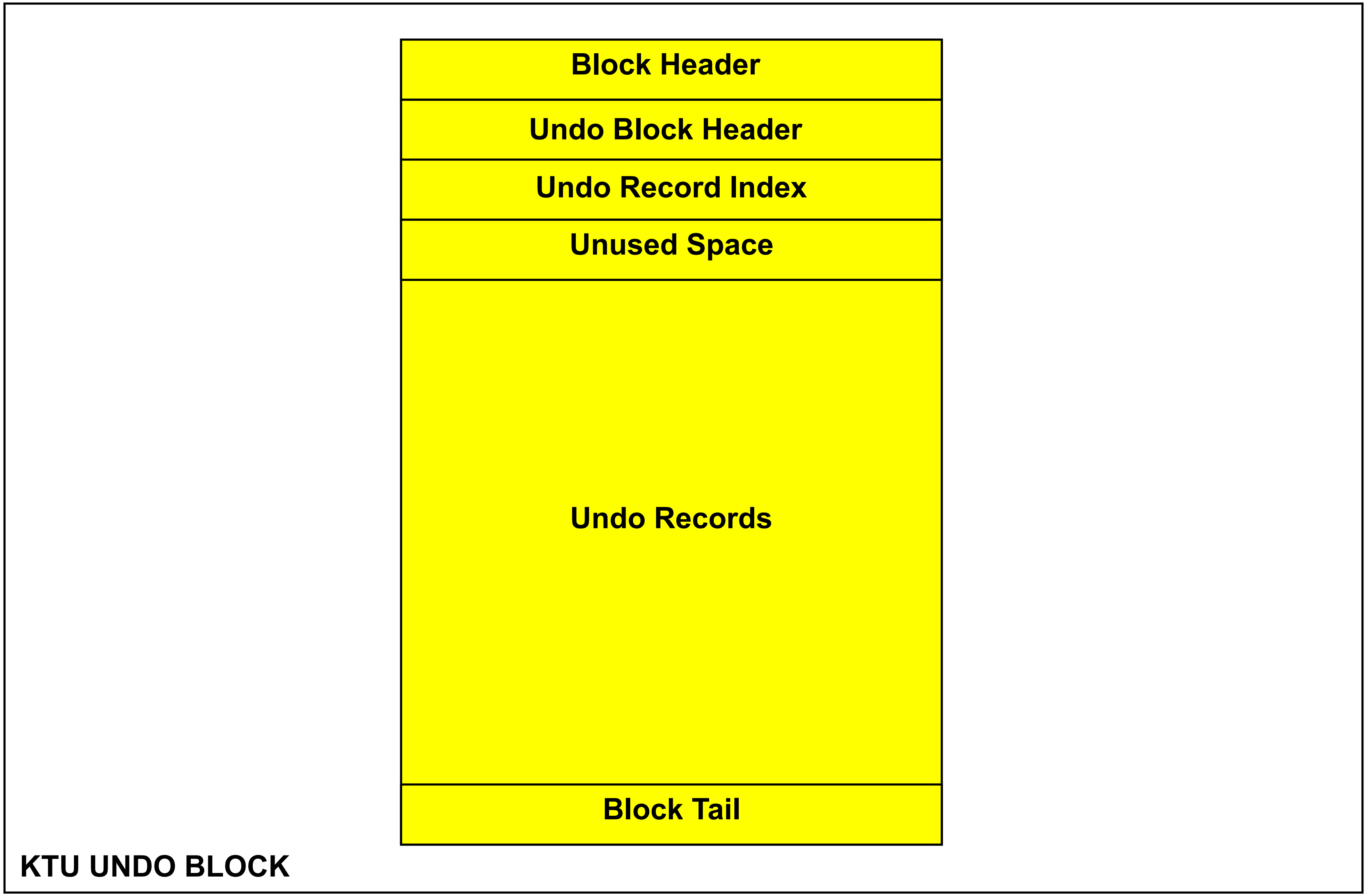
UPDATE score
SET runs = 80
WHERE team = 'AUS';

XID 0001.023.000004C8

Undo Extents

- Each undo extent contains contiguous set of undo blocks
- Each undo block can only be allocated to one transaction
- Undo blocks contain
  - Undo block header
  - Undo records

Undo Block Structure



Block Header

Undo Block Header

Undo Record Index

Unused Space

Undo Records

Block Tail

KTU UNDO BLOCK

Undo Block

- Undo Block Header contains
  - Transaction ID (XID) for current / last transaction to use block
  - Sequence number of undo block
  - Number of undo records in undo block
    - Not necessarily in current transaction

- Undo records are chained together
  - Allow transaction to be rolled back

- Undo records are also used cyclically
  - remain in block for as long as possible
  - reduces probability of ORA-01555: Snapshot too old

Undo Byte Address (UBA)

- Specifies address of undo record (not just the undo block)

- Contains
  - DBA of undo block
  - Sequence number of undo block
  - Record number in undo block

- For example: 0x008004f1.0527.1f

- Most recent UBA for transaction reported in V$TRANSACTION
  - UBAFIL, UBABLK - file and block number
  - UBASQN - sequence number
  - UBAREC - record number

Undo Change Vectors - Data Blocks

- For data blocks

	INSERT	UPDATE	DELETE

	INSERT INTO score
	(team, runs, wickets)
	VALUES
	('AUS',100,4);
		UPDATE score
		SET
		runs = 104,
		wickets = 5
		WHERE team = 'AUS';
			DELETE FROM score
			WHERE team = 'AUS';

Undo	OP 5.1 (11.1) Delete Row Piece - DRP Slot 4:	OP 5.1 (11.1) Update Row Piece - URP Slot 4: c1: 100 c2: 4	OP 5.1 (11.1) Insert Row Piece - IRP Slot 4: c0: 'AUS' c1: 104 c2: 5
Redo	OP 11.2 Insert Row Piece - IRP Slot 4: c0: 'AUS' c1: 100 c2: 4	OP 11.5 Update Row Piece - URP Slot 4: c1: 104 c2: 5	OP 11.3 Delete Row Piece - DRP Slot 4:

Undo Change Vectors - Index Blocks

- Assume unique index on SCORE (TEAM)

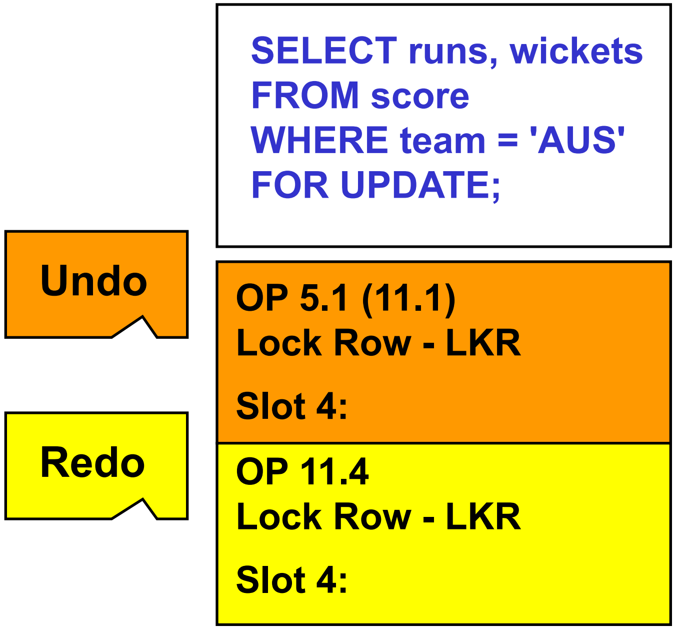
	INSERT	UPDATE	DELETE

	INSERT INTO score (team, runs, wickets) VALUES ('AUS',100,4);	UPDATE score
		SET team = 'ENG'
		WHERE team = 'AUS';
			DELETE FROM score
			WHERE team = 'ENG';

Undo	OP 5.1 (10.22) Purge Leaf Row key: 'AUS'	OP 5.1 (10.22) Restore leaf row key: 'AUS' data: 01 00 20 47 00 00	OP 5.1 (10.22) Restore leaf row key: 'ENG' data: 01 00 20 47 00 00
Redo	OP 10.2 Insert Leaf Row key: 'AUS' data: 01 00 20 47 00 00	OP 10.4 Delete leaf row slot: 0	OP 10.4 Delete leaf row slot: 0
Undo		OP 5.1 (10.22) Purge leaf row key: 'ENG'	
Redo		OP 10.2 Insert Leaf Row key: 'ENG' data: 01 00 20 47 00 00	

SELECT FOR UPDATE

- Redo and Undo Generation



```
SELECT runs, wickets
FROM score
WHERE team = 'AUS'
FOR UPDATE;
```

Undo

OP 5.1 (11.1)
Lock Row - LKR

Slot 4:

Redo

OP 11.4
Lock Row - LKR

Slot 4:

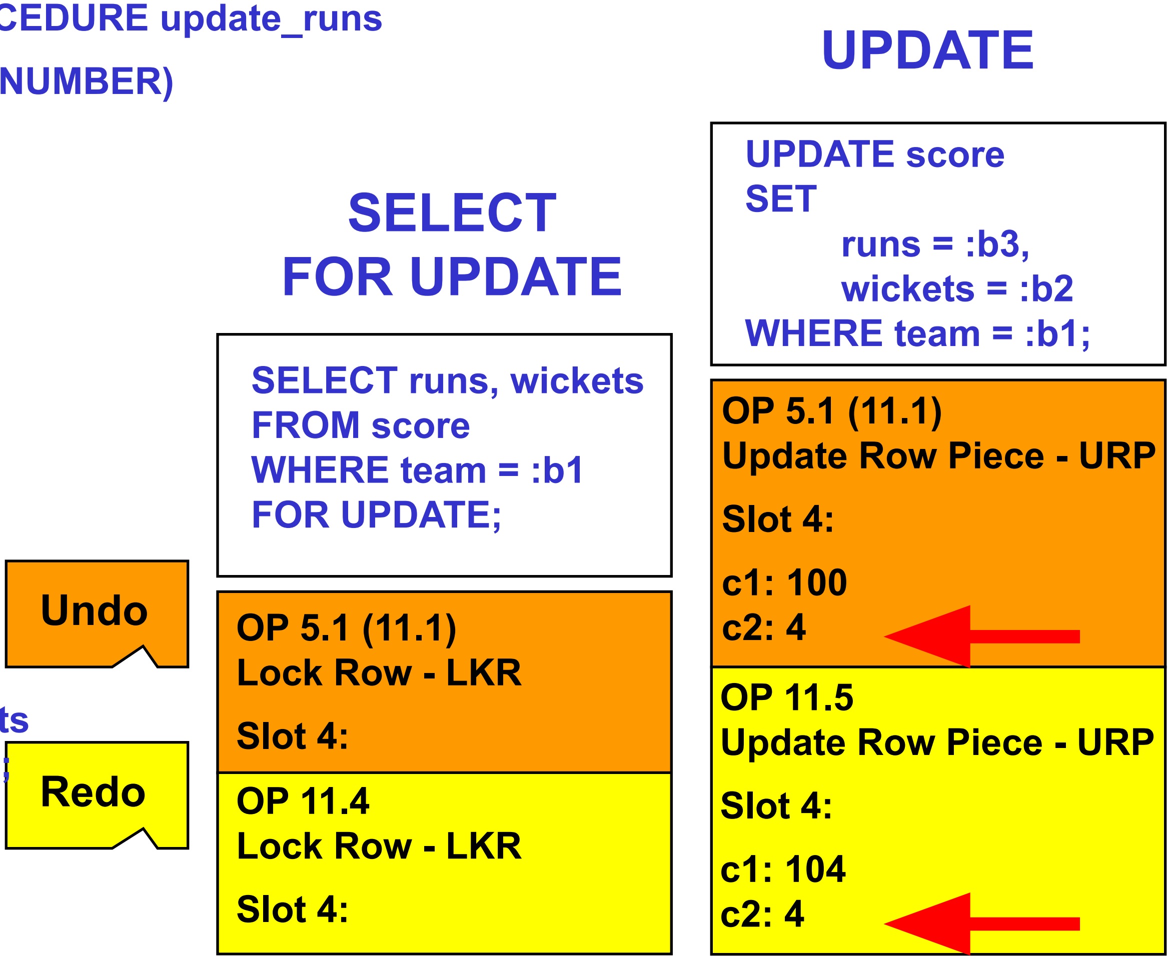
SELECT FOR UPDATE

- SELECT FOR UPDATE is bad for so many reasons.....
  - Rows are locked pessimistically:
    - More chance of contention
  - Rows could be locked optimistically by any subsequent UPDATE statement
    - Application logic may need modification
  - SELECT FOR UPDATE generates:
    - Undo - more space in buffer cache, ORA01555 etc
    - Redo - increased physical I/O
  - SELECT FOR UPDATE statements cannot be batched
    - Each requires a separate pair of change vectors

UPDATE Statements

- Redo and Undo Generation

```
CREATE OR REPLACE PROCEDURE update_runs
(p_team VARCHAR2,p_runs NUMBER)
IS
        l_runs NUMBER;
        l_wickets NUMBER;
BEGIN
        SELECT runs, wickets
        INTO l_runs, l_wickets
        FROM score
        WHERE team = p_team
        FOR UPDATE;

        UPDATE test3
        SET
                runs = l_runs,
                wickets = l_wickets
        WHERE team = p_team;
END;
/
```



Undo

Redo

SELECT
FOR UPDATE

```
SELECT runs, wickets
FROM score
WHERE team = :b1
FOR UPDATE;
```

OP 5.1 (11.1)
Lock Row - LKR

Slot 4:

OP 11.4
Lock Row - LKR

Slot 4:

UPDATE

```
UPDATE score
SET
        runs = :b3,
        wickets = :b2
WHERE team = :b1;
```

OP 5.1 (11.1)
Update Row Piece - URP

Slot 4:

c1: 100
c2: 4

OP 11.5
Update Row Piece - URP

Slot 4:

c1: 104
c2: 4

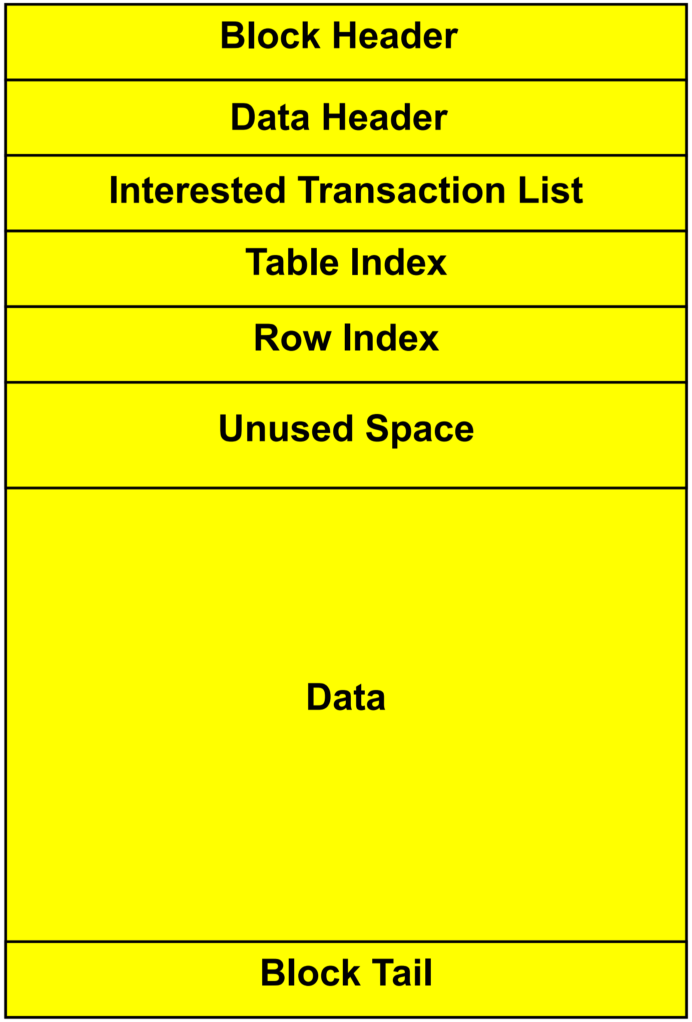
UPDATE Statements

- UPDATE statements that include unchanged columns

- Advantages
  - Reduce parse overhead
    - Good on single instance, even better on RAC
  - Reduce space required in library cache
    - Less chance cursors will be aged out

- Disadvantages
  - Increase physical I/O to online redo logs
  - Increase number of undo blocks in buffer cache
  - Increase probability of ORA-01555

Data Block Structure



Block Header

Data Header

Interested Transaction List

Table Index

Row Index

Unused Space

Data

Block Tail

Interested Transaction List

- Each data/index block has an Interested Transaction List
  - list of transactions currently active on block
  - stored within block header

- Each data/index row header contains a lock byte
  - Lock byte records current slot in ITL

- Each row can only be associated with one transaction
  - If a second transaction attempts to update a row it will experience a row lock waits until first transaction commits/ rolls back

- Initially two ITL entries are reserved in block header
  - ITL list can grow dynamically according to demand
  - ITL list cannot shrink again

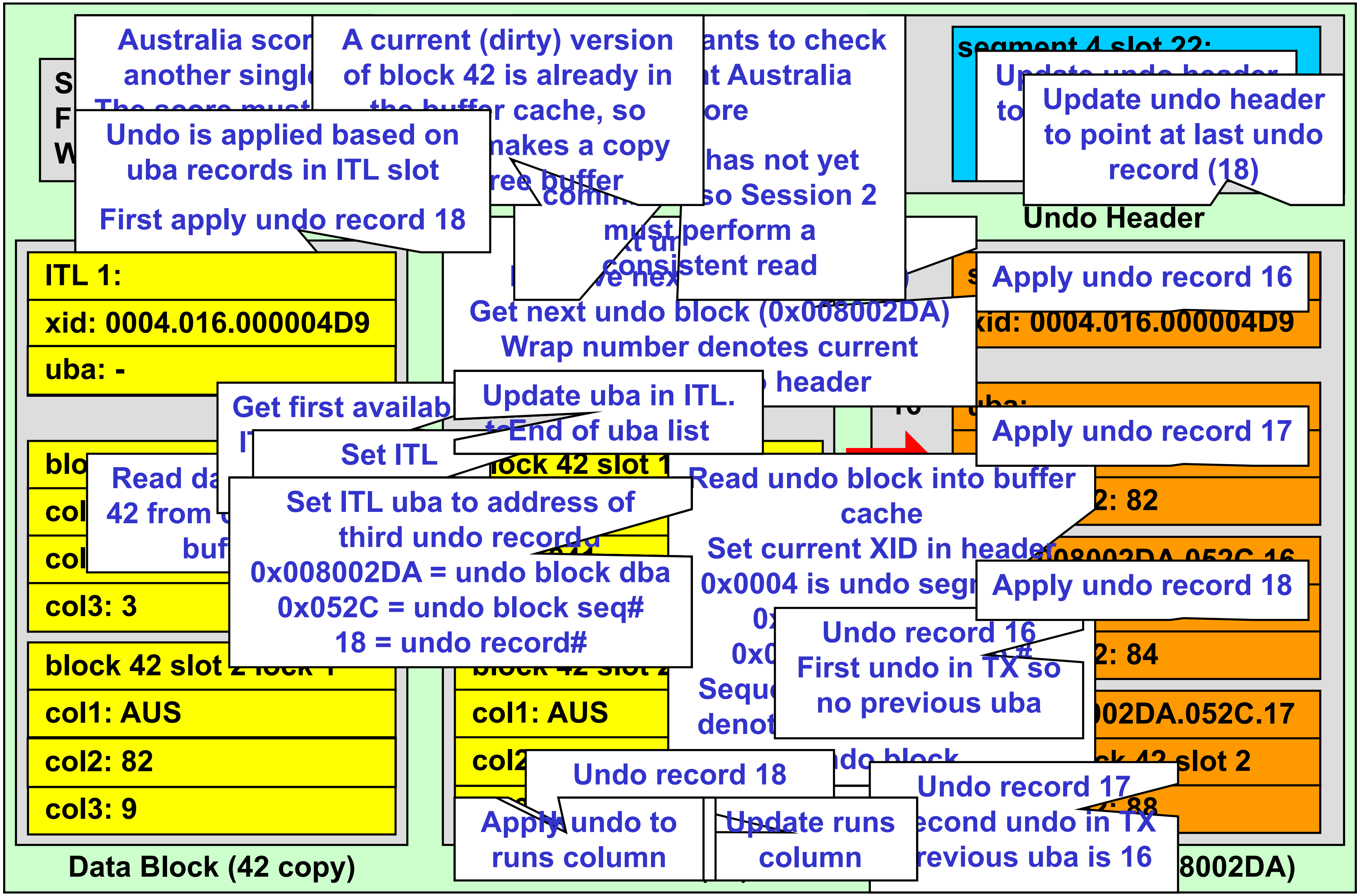
Interested Transaction List

- ITL entry includes
  - Transaction ID (XID)
  - Undo byte address (UBA)
  - System Change Number (SCN)
- ITL entry is overwritten by each change to the block by the current transaction
- Previous change is recorded in undo block
- During rollback, changes are restored to ITL from undo block

Read Consistency

- Required to maintain ACID properties of transaction
    - Transactions must always see consistent versions of blocks modified by other transactions
    - Can be applied at
        - Statement level (default)
        - Transaction level
    - Uncommitted block updates are rolled back when block is read
        - Consistent reads
    - More specifically undo is applied to return block to consistent state
    - Undo must still be available in undo segment
        - If undo has been overwritten, querying session will receive ORA-01555: Snapshot too old

Read Consistency



S
F
W

Australia scor
another singl
The score must

A current (dirty) version
of block 42 is already in
the buffer cache, so
makes a copy
ree buffer

ants to check
t Australia
ore
has not yet
so Session 2
must perform a
consistent read

Undo is applied based on
uba records in ITL slot

First apply undo record 18

segment 4 slot 22:

Up
to

Update undo header
to point at last undo
record (18)

Undo Header

ITL 1:

xid: 0004.016.000004D9

uba: -

Apply undo record 16

xid: 0004.016.000004D9

Get next undo block (0x008002DA)
Wrap number denotes current
header

Get first availab
IT

Update uba in ITL.
toEnd of uba list

Set ITL

Read da
42 from
buf

Set ITL uba to address of
third undo record
0x008002DA = undo block dba
0x052C = undo block seq#
18 = undo record#

blo
col
col
col3: 3

lock 42 slot 1

block 42 slot 2 lock 1
col1: AUS
col2: 82
col3: 9

block 42 slot 2
col1: AUS
col2

uba:

Apply undo record 17

2: 82

Read undo block into buffer
cache
Set current XID in header
0x0004 is undo segr
0x
0x0
Sequ
denot

008002DA.052C.16

Apply undo record 18

2: 84

Undo record 16
First undo in TX so
no previous uba

02DA.052C.17

ck 42 slot 2

2: 88

Undo record 18

ndo block

Apply undo to
runs column

Update runs
column

Undo record 17
econd undo in TX
revious uba is 16

Data Block (42 copy)

8002DA)

STOP

SET TRANSACTION

- Determines level at which read-consistency is applied
- Can be:
  - SET TRANSACTION READ WRITE
    - establishes statement-level read consistency
    - subsequent statements see any changes committed before that statement started
    - default behaviour
  - SET TRANSACTION READ ONLY
    - establishes transaction-level read consistency
    - all subsequent statements only see changes committed before transaction started
    - not supported for SYS user
- SET TRANSACTION statement must be first statement in transaction

SET TRANSACTION

- For example:

Session 1	Session 2	Session 3

SELECT runs
FROM score
WHERE team = 'ENG';
Runs
127		

	SET TRANSACTION
	READ WRITE;
		SET TRANSACTION
		READ ONLY;

UPDATE team
SET runs = 131
WHERE team = 'ENG';
COMMIT;		

	SELECT runs
	FROM score
	WHERE team = 'ENG';
	Runs
	131
		SELECT runs
		FROM score
		WHERE team = 'ENG';
		Runs
		127

ORA_ROWSCN Pseudocolumn

- Returns conservative upper-bound SCN for most recent change in row
- Uses SCN stored for transaction in ITL
- Shows last time a row in same block was updated
  - May show more accurate information for an individual row
- Not supported with flashback query
- To convert ORA_ROWSCN to an approximate timestamp use the SCN_TO_TIMESTAMP built-in function e.g.

```
SELECT ORA_ROWSCN,
SCN_TO_TIMESTAMP (ORA_ROWSCN)
FROM score;
```

ORA_ROWSCN Pseudocolumn

- For example - no row dependencies (default)

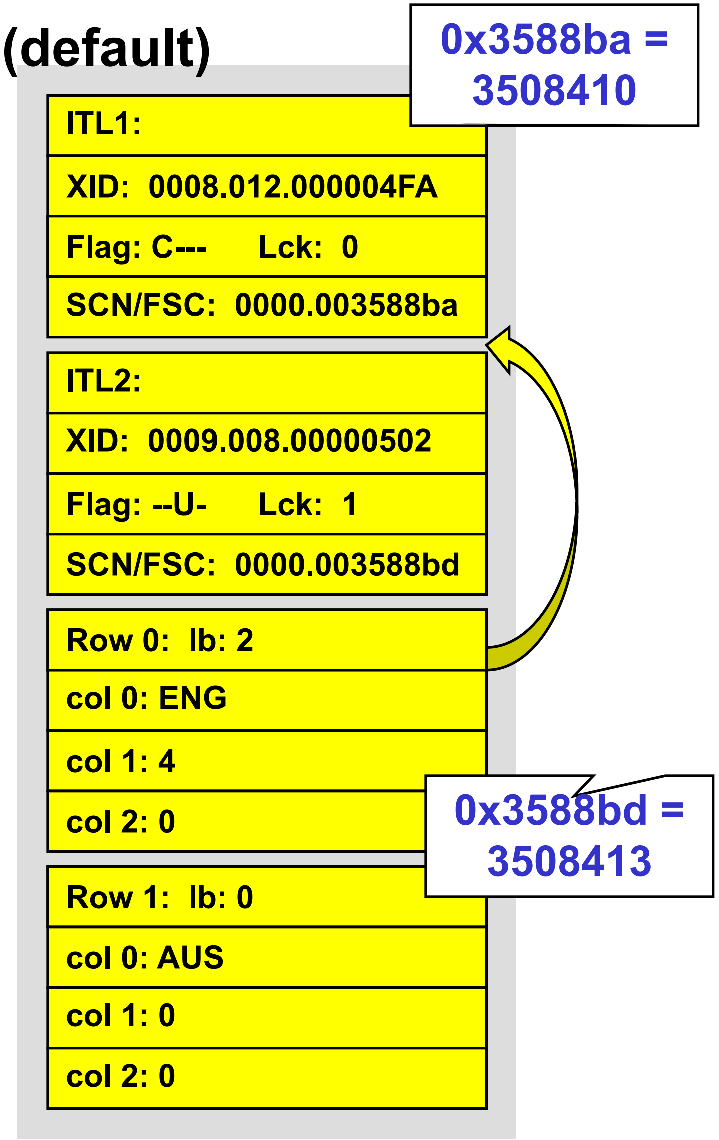
```
CREATE TABLE score
(team NUMBER, runs NUMBER, wickets NUMBER);
```

```
INSERT INTO score (team, runs, wickets) VALUES ('ENG',0,0);
INSERT INTO score (teams,runs,wickets) VALUES ('AUS',0,0);

COMMIT;
```

```
SELECT ORA_ROWSCN, teams, runs, wickets FROM score;

ORA_ROWSCN      Teams         Runs  Wickets
3508410    ENG   0     0
3508410    AUS   0     0
```

```
UPDATE score
SET runs = 4
WHERE team = 'ENG';

COMMIT;
```

```
SELECT ORA_ROWSCN, teams, runs, wickets FROM score;

ORA_ROWSCN      Teams         Runs  Wickets
3508413    ENG   4     0
3508413    AUS   0     0
```



0x3588ba = 3508410

ITL1:
XID: 0008.012.000004FA
Flag: C--- Lck: 0
SCN/FSC: 0000.003588ba

ITL2:
XID: 0009.008.00000502
Flag: --U- Lck: 1
SCN/FSC: 0000.003588bd

Row 0: lb: 2
col 0: ENG
col 1: 4
col 2: 0

0x3588bd = 3508413

Row 1: lb: 0
col 0: AUS
col 1: 0
col 2: 0

ORA_ROWSCN Pseudocolumn

- For example (row dependencies)

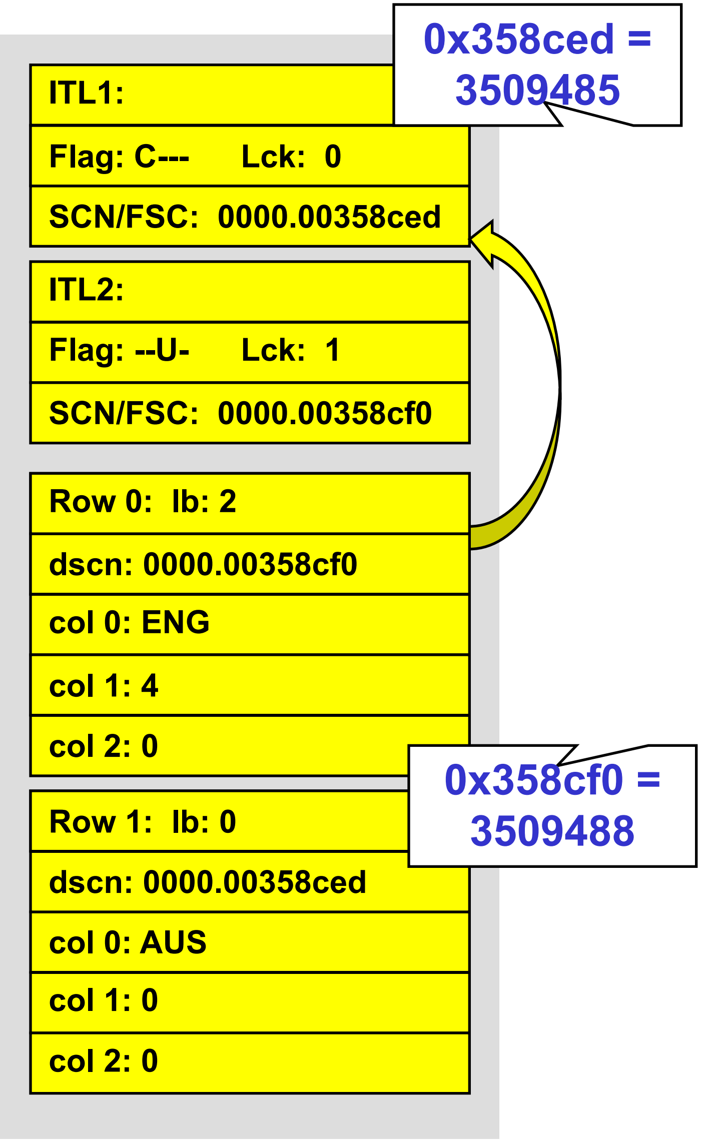
```
CREATE TABLE score
(team NUMBER, runs NUMBER, wickets NUMBER)
ROWDEPENDENCIES;
```

```
INSERT INTO score (team, runs, wickets) VALUES ('ENG',0,0);
INSERT INTO score (teams,runs,wickets) VALUES ('AUS',0,0);

COMMIT;
```

```
SELECT ora_rowscn, teams, runs, wickets FROM score;

ORA_ROWSCN    Teams      Runs  Wickets
3509485    ENG   0    0
3509485    AUS   0    0
```

```
UPDATE score
SET runs = 4
WHERE team = 'ENG';

COMMIT;
```

```
SELECT ora_rowscn, teams, runs, wickets FROM score;

ORA_ROWSCN    Teams      Runs  Wickets
3509488    ENG   4    0
3509485    AUS   0    0
```



0x358ced = 3509485

ITL1:

Flag: C--- Lck: 0

SCN/FSC: 0000.00358ced

ITL2:

Flag: --U- Lck: 1

SCN/FSC: 0000.00358cf0

Row 0: lb: 2

dscn: 0000.00358cf0

col 0: ENG

col 1: 4

col 2: 0

0x358cf0 = 3509488

Row 1: lb: 0

dscn: 0000.00358ced

col 0: AUS

col 1: 0

col 2: 0

Flashback Query

- Example

Session 1	Session 2

SELECT runs
FROM score
WHERE team = 'ENG';
Runs
137
	SELECT dbms_flashback.get_system_change_number FROM dual;
	SCN
	3494824

UPDATE team
SET runs = 141
WHERE team = 'ENG';
COMMIT;	

	SELECT dbms_flashback.get_system_change_number FROM dual; SCN 3494833

	SELECT team, runs, wickets FROM score
	WHERE team = 'ENG';
	Team Runs Wickets
	ENG 141 1

	SELECT team, runs, wickets FROM score AS OF SCN 3494824;
	WHERE team = 'ENG';
	Team Runs Wickets
	ENG 137 1

Flashback Query

- Can specify AS OF clause:
  - Returns single-row
  - Syntax is

```
AS OF [ SCN <scn> | TIMESTAMP <timestamp> ]
```

- For example:

```
SELECT team, runs, wickets
FROM score AS OF SCN 3506431
WHERE team = 'ENG';
```

Flashback Query

- Can also specify VERSIONS clause:
  - Returns multiple rows
  - Syntax is

```
VERSIONS BETWEEN SCN [ <scn> | MINVALUE ]
AND [ <scn> | MAXVALUE
```

```
VERSIONS BETWEEN TIMESTAMP [ <timestamp> | MINVALUE ]
AND [ <timestamp> | MAXVALUE
```

- For example:

```
SELECT team, runs, wickets
FROM score VERSIONS BETWEEN SCN 3503511 AND 3503524
WHERE team = 'ENG';
```

Version Query Pseudocolumns

- Valid only for Flashback Version Query. Values can be:
  - VERSIONS_STARTTIME
    - timestamp of first version of rows returned by query
  - VERSIONS_ENDTIME
    - timestamp of last version of rows returned by query
  - VERSIONS_STARTSCN
    - SCN of first version of rows returned by query
  - VERSIONS_ENDSCN
    - SCN of last version of rows returned by query
  - VERSIONS_XID
    - For each row returns transaction ID of transaction creating that row version
  - VERSIONS_OPERATION
    - For each row returns operation creating that row version. Can be I(nsert) U(pdate) or D(elete)

Version Query Pseudocolumns

- **Example:**

Session 1	Session 2

SELECT runs
FROM score
WHERE team = 'ENG';
 	
Runs
141
	SELECT dbms_flashback.get_system_change_number FROM dual;
	
	SCN
	3503136

UPDATE team
SET runs = 145
WHERE team = 'ENG';
 	
COMMIT;	

UPDATE team
SET runs = 151
WHERE team = 'ENG';
 	
COMMIT;	

UPDATE team
SET runs = 153
WHERE team = 'ENG';
 	
COMMIT;	

	SELECT dbms_flashback.get_system_change_number FROM dual; SCN 3503143

Version Query Pseudocolumns

- **Example (continued):**

Session 1

Session 2

```
SELECT
        VERSIONS_STARTSCN "Start",
        VERSIONS_ENDSCN "End",
        VERSIONS_XID "XID",
        VERSIONS_OPERATION "Op",
        score.team "Team",
        score.runs "Runs",
        score.wickets "Wickets"
FROM score VERSIONS BETWEEN SCN 3503136 AND 3503143
WHERE team = 'ENG';
```

```
Start  End          XID    Op     Team  Runs  Wickets
3503142                    08000A00FC040000      U       ENG   153    1
3503139      3503142       07001A00F6040000      U       ENG   151    1
3503136      3503139       06002C00EA040000      U       ENG   145    1
             3503136                  ENG   141    1
```

XID = 0066.02C.000004EA
(Architecture = X86)

Can be I(nsert), U(pdate)
or D(elete)

Thank you for your interest

For more information and to provide feedback please contact me

My e-mail address is:

info@juliandyke.com

My website address is:

www.juliandyke.com