

A Tour of PostgreSQL's On-Disk Storage

James Coleman
PGConf.NYC, 4-October 2023

About Me

- Architect for Data Engineering at Braintree Payments (PayPal)
- Contributor to Postgres versions 12+, including patches like:
 - Incremental sort (Postgres 13)
 - `ScalarArrayOpExpr ([NOT] IN, = ANY/ALL)` planner optimization and query execution improvements

Agenda

1. How to find data on disk
2. Heap table storage
3. BTree index storage
4. Data modeling implications

Relations: What and Where

Relations

What do they include?

- Tables
 - Note: We'll only discuss Heap tables.
 - Sequences
- Views
- Indexes

Relations

Where can we find them?

- As with all Postgres objects; start with catalog tables.
- `pg_catalog.pg_class` contains tables, indexes, sequences, views, etc.
 - `relfilenode` points us to the on-disk file (if non-zero); must also consider `reltablespace` if using tablespaces.

<https://www.postgresql.org/docs/current/catalogs.html>

<https://www.postgresql.org/docs/current/catalog-pg-class.html>

Relations

Where can we find them?

```
my_db=# select oid, relname, relfilenode from pg_class where relname = 'foo';
```

oid	relname	relfilenode
16393	foo	16393

Relations

Where can we find them?

```
my_db=# vacuum full foo;  
VACUUM
```

```
my_db=# select oid, relname, relfilenode from pg_class where relname = 'foo';  
  oid  | relname | relfilenode  
-----+-----+-----  
16393 | foo    |          16396
```


Relations

Where can we find them?

```
[dev]jcoleman-postgres:~/postgresql-test-data% ls
```

```
PG_VERSION      pg_hba.conf     pg_replslot/    pg_subtrans/
postgresql.auto.conf
base/           pg_ident.conf   pg_serial/      pg_tblspc/      postgresql.conf
global/        pg_logical/     pg_snapshots/   pg_twophase/    postmaster.opts
pg_commit_ts/  pg_multixact/   pg_stat/        pg_wal/         postmaster.pid
pg_dynshmem/   pg_notify/      pg_stat_tmp/    pg_xact/
```

Relations

Where can we find them?

```
[dev]jcoleman-postgres:~/postgresql-test-data% cd base  
[dev]jcoleman-postgres:~/postgresql-test-data/base% ls
```

```
1/ 16392/ 4/ 5/
```

- We previously found `relfilenode = 16396`; why isn't that here?

```
my_db=# select oid, datname from pg_database;
```

oid	datname
5	postgres
1	template1
4	template0
16392	my_db

Relations

Where can we find them?

```
[dev]jcoleman-postgres:~/postgresql-test-data/base/16392% ls
112          2336          2616_fsm    2685          3079_fsm    3576          4156
...
2187         2615          2681        2841          3541_vm     4152          PG_VERSION
2224         2615_fsm     2682        2995          3542        4153          pg_filenode.map
2228         2615_vm     2683        2996          3574        4154          pg_internal.init
2328         2616          2684        3079          3575        4155

[dev]jcoleman-postgres:~/postgresql-test-data/base/16392% ls -lah 16396*
-rw----- 1 admin admin 0 Sep 25 15:34 16396
```

Relations

Where can we find them?

```
my_db=# SHOW data_directory;  
      data_directory
```

```
-----  
/home/admin/postgresql-test-data
```

```
my_db=# SELECT pg_relation_filepath('foo');  
      pg_relation_filepath
```

```
-----  
base/16392/16396
```

Relations

What happens when we insert lots of data?

```
my_db=# create table bar(pk bigserial primary key, i int, some_fk bigint,  
  other_fk bigint, t text, created_at timestamp default 'now()',  
  updated_at timestamp default 'now()');  
CREATE TABLE
```

```
my_db=# insert into bar(i, some_fk, other_fk, t) select s.n, s.n % 10,  
  s.n % 100, md5(s.n::text) from generate_series(1, 13_000_000) s(n);  
INSERT 0 13000000
```

```
my_db=# select pg_size_pretty(pg_relation_size('bar'));  
pg_size_pretty
```

1451 MB

Relations

What happens when we insert lots of data?

```
my_db=# SELECT pg_relation_filepath('bar');
pg_relation_filepath
```

```
-----
base/16392/16400
```

```
[dev]jcoleman-postgres:~/postgresql-test-data/base/16392% ls -lah 16400*
-rw----- 1 admin admin 1.0G Sep 25 17:33 16400
-rw----- 1 admin admin 427M Sep 25 17:41 16400.1
-rw----- 1 admin admin 384K Sep 25 17:41 16400_fsm
-rw----- 1 admin admin  48K Sep 25 17:41 16400_vm
```

- Relation segment size is configurable at compile time with **RELSEG_SIZE**, but cluster must also be initialized to match the running binaries.

Heap Pages

Pages

Size

- Each relation segment file is divided into 8 KB pages.
 - Similarly to relation segment size, this can be configured at compilation time (**BLCKSZ**) and must match the cluster's storage.
 - Always the same on all relations.

Pages

Layout (Tables)

PageHeaderData	ItemIdData	Free	Items	Special Space
24 bytes	[4 bytes, ...]		[..., Item]	

- **ItemIdData** is an array that grows front-to-back.
 - Each id is a pointer (offset, size) to an item in the Items array.
- Items is an array that grows back-to-front.
 - An “item” may be a tuple (on a heap table relation) or an index entry.

Pages

PageHeaderData

pd_lsn	pd_checksum	pd_flags	pd_lower	pd_upper	pg_special	pd_pagesize_version	pd_prune_xid
8 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	4 bytes

- **pd_lsn**: first byte of WAL that was written after this page was last changed
- **pd_flags**: hints about free space; flag denoting if all items are globally visible
- **pd_lower/pd_upper**: offsets to free space on the page
- **pd_prune_xid**: max XID of any unpruned tuples on the page

Heap Tuples

Heap Tuple Layout

HeapTupleHeaderData	Null Bitmap	Padding	Optional OID	Columns
23 bytes	0..n bytes	0..8 bytes	0 or 4 bytes	...

- Column data must start on a multiple of MAXALIGN (usually 4 or 8 bytes)
 - Virtually all systems will have at least one byte for the null bitmap (first 8 nulls are free!)
 - Assume at least 24 bytes of overhead per tuple.
- Each column value may also have type-dependent alignment requirements.

Heap Tuples

HeapTupleHeaderData

t_xmin	t_xmax	t_cid/t_xvac	t_ctid	t_infomask2	t_infomask	t_hoff
4 bytes	4 bytes	4 bytes	6 bytes	2 bytes	2 bytes	1 byte

- **t_xmin** / **t_xmax**: Implement MVCC visibility.
- **t_ctid**: can link to other tuples within the same page to implement HOT update chains.
- **t_hoff**: offset to column data.

Heap Tuples

- Each tuple must fit within a single page (excluding TOAST data).
- Variable length fields (e.g., **text**, **bytea**, **jsonb**) stored with 1 or 4 bytes of overhead inline.
- TOAST (The Oversized-Attribute Storage Technique)
 - Max size of 1 GB per value
 - 18 byte pointer stored inline; storage is moved outside of the page.

<https://www.postgresql.org/docs/current/storage-page-layout.html>

<https://www.postgresql.org/docs/current/limits.html>

<https://www.postgresql.org/docs/current/storage-toast.html>

Tooling

pageinspect

- The pageinspect extension is included in the contrib package that ships (depending on your distribution) with PostgreSQL.
- Provides functions that allow low-level inspection of database pages.
 - Unfortunately the reads still go through standard buffer validation, and so you can't read pages with an invalid checksum.

pageinspect

Raw pages

```
my_db=# select get_raw_page('bar', 1);
```

get_raw_page

```
-----  
\x00000000a00fe40100000400300160010020042000000000909fe000209fe000b09ee  
000409ee000d09de000609de000f09ce000809ce000109ce000a09be000309be000c09a  
e000509ae000e099e0007099e0000099e0009098e0002098e000b097e0004097e000d09  
6e0006096e000f095e0008095e0001095e000a094e0003094e000c093e0005093e000e0  
92e0007092e0000092e0009091e0002091e000b090e0004090e000d08fe000608fe000f  
08ee000808ee000108ee000a08de000308de000c08ce000508ce000e08be000708be000  
008be000908ae000208ae000b089e0004089e000d088e0006088e000f087e0008087e00  
01...
```

pageinspect

Page headers

```
my_db=# select * from page_header(get_raw_page('bar', 1));
```

```
-[ RECORD 1 ]-----
```

```
lsn          | 0/1E40FA0
```

```
checksum     | 0
```

```
flags        | 4
```

```
lower        | 304
```

```
upper        | 352
```

```
special      | 8192
```

```
pagesize     | 8192
```

```
version      | 4
```

```
prune_xid    | 0
```


pageinspect

Page headers (after deleting a tuple)

```
my_db=# select * from page_header(get_raw_page('bar', 1));
```

```
-[ RECORD 1 ]-----
```

```
lsn          | 0/A861DD50
```

```
checksum     | 0
```

```
flags        | 0
```

```
lower        | 304
```

```
upper        | 352
```

```
special      | 8192
```

```
pagesize     | 8192
```

```
version      | 4
```

```
prune_xid    | 758
```

pageinspect

Page headers (after vacuuming)s

```
my_db=# select * from page_header(get_raw_page('bar', 1));
```

```
-[ RECORD 1 ]-----
```

lsn	0/A861FF20
checksum	0
flags	0
lower	304
upper	464
special	8192
pagesize	8192
version	4
prune_xid	0

1	112	749	0	0	(5, 1)	7	2306	24	\x5f010000000000005f010000000000000100000000000000330000000000000043656665393337373830653935353734323530646162653037313531626463323300000000000002c2cb6e2630a90
1	112	749	0	0	(5, 2)	7	2306	24	\x600100000000000060010000000000000200000000000000340000000000000043333731626365376463383338313762373839336263646555643133373939623500000000000002c2cb6e2630a90
1	112	749	0	0	(5, 3)	7	2306	24	\x61010000000000006101000000000000030000000000000035000000000000004331333862623036393635393562333386166626162233333633535353239326100000000000002c2cb6e2630a90
1	112	749	0	0	(5, 4)	7	2306	24	\x62010000000000006201000000000000040000000000000036000000000000004338646434864366132653263616432313331373961333939326330626535336300000000000002c2cb6e2630a90
1	112	749	0	0	(5, 5)	7	2306	24	\x630100000000000063010000000000000500000000000000370000000000000043382636563393664343238316237633936343755373436323334393600000000000002c2cb6e2630a90
1	112	749	0	0	(5, 6)	7	2306	24	\x64010000000000006401000000000000060000000000000038000000000000004336633532346639643564323730323734641373833633834323253062613731000000000000002c2cb6e2630a90
1	112	749	0	0	(5, 7)	7	2306	24	\x650100000000000065010000000000000700000000000000390000000000000043656237623956666135343632303834633566346537653835613039336536643700000000000002c2cb6e2630a90
1	112	749	0	0	(5, 8)	7	2306	24	\x6601000000000000660100000000000008000000000000003a0000000000000043616139343261623262666136656264613438343065373336306365366537656600000000000002c2cb6e2630a90
1	112	749	0	0	(5, 9)	7	2306	24	\x6701000000000000670100000000000009000000000000003b0000000000000043633035386635343463373337373832646561636566613533326439616464346300000000000002c2cb6e2630a90
1	112	749	0	0	(5, 10)	7	2306	24	\x68010000000000006801000000000000000000000000003c0000000000000043653762323462313132613434666454396565393362646639393863366361306500000000000002c2cb6e2630a90
1	112	749	0	0	(5, 11)	7	2306	24	\x6901000000000000690100000000000001000000000000003d00000000000000433532373230653030333534376337303536316266356530616139396600000000000002c2cb6e2630a90
1	112	749	0	0	(5, 12)	7	2306	24	\x6a010000000000006a0100000000000002000000000000003e0000000000000043363365383738653237653265326135376163666439613736666436636600000000000002c2cb6e2630a90
1	112	749	0	0	(5, 13)	7	2306	24	\x6b010000000000006b0100000000000003000000000000003f0000000000000043303034313134363066376339326432313234613637656130663463623566383500000000000002c2cb6e2630a90
1	112	749	0	0	(5, 14)	7	2306	24	\x6c010000000000006c01000000000000040000000000000040000000000000043626163393136326234376335366663386134643261353139383033643531623300000000000002c2cb6e2630a90
1	112	749	0	0	(5, 15)	7	2306	24	\x6d010000000000006d010000000000000500000000000000410000000000000043396265343053656535623065656531343632633832633639363430383766663900000000000002c2cb6e2630a90
1	112	749	0	0	(5, 16)	7	2306	24	\x6e010000000000006e010000000000000600000000000000420000000000000043356566363938636439666536353039323365613333316331356166336231363000000000000002c2cb6e2630a90
1	112	749	0	0	(5, 17)	7	2306	24	\x6f010000000000006f01000000000000070000000000000043000000000000004303353034396539306661346635033396138636164633661636262346232636300000000000002c2cb6e2630a90
1	112	749	0	0	(5, 18)	7	2306	24	\x7001000000000000700100000000000008000000000000004400000000000000436366303036366303036366613161343937326633365326631636133000000000000002c2cb6e2630a90
1	112	749	0	0	(5, 19)	7	2306	24	\x7101000000000000710100000000000009000000000000004500000000000000433063373462237663738343039613430323261326334633561356361336565313900000000000002c2cb6e2630a90
1	112	749	0	0	(5, 20)	7	2306	24	\x720100000000000072010000000000000000000000000046000000000000004600000000000004364373039663338656637353862353036366565333162313830333962386365350000000000002c2cb6e2630a90
1	112	749	0	0	(5, 21)	7	2306	24	\x73010000000000007301000000000000010000000000000047000000000000004334316631663139313736643338333438306166613635643332356330366564300000000000002c2cb6e2630a90
1	112	749	0	0	(5, 22)	7	2306	24	\x740100000000000074010000000000000200000000000000480000000000000043323462313666656465396136376339323531643365376337313631633833616300000000000002c2cb6e2630a90
1	112	749	0	0	(5, 23)	7	2306	24	\x750100000000000075010000000000000300000000000000490000000000000043666643532663363365313234333561372346313866663330666464616464396300000000000002c2cb6e2630a90
1	112	749	0	0	(5, 24)	7	2306	24	\x7601000000000000760100000000000004000000000000004a0000000000000043616439373266333653266316361330000000000000002c2cb6e2630a90
1	112	749	0	0	(5, 25)	7	2306	24	\x7701000000000000770100000000000005000000000000004b00000000000000436536316436393437343637636364336161356166323464623332303233356464000000000002c2cb6e2630a90
1	112	749	0	0	(5, 26)	7	2306	24	\x7801000000000000780100000000000006000000000000004c0000000000000043313432393439646635366561386165306265386235333036393731393030613400000000000002c2cb6e2630a90
1	112	749	0	0	(5, 27)	7	2306	24	\x7901000000000000790100000000000007000000000000004d0000000000000043643334616231363962373063396465364333565363238393630313063643966660000000000002c2cb6e2630a90
1	112	749	0	0	(5, 28)	7	2306	24	\x7a010000000000007a0100000000000008000000000000004e0000000000000043386266313231316664346237623934353238383939646530613433623966623300000000000002c2cb6e2630a90
1	112	749	0	0	(5, 29)	7	2306	24	\x7b010000000000007b0100000000000009000000000000004f000000000000004361303266666439313656336535653765666562343664623866313061373430353900000000000002c2cb6e2630a90
1	112	749	0	0	(5, 30)	7	2306	24	\x7c010000000000007c010000000000000000000000000050000000000000004362636138326534316565323038333335393962316663637137376337000000000000002c2cb6e2630a90
1	112	749	0	0	(5, 31)	7	2306	24	\x7d010000000000007d0100000000000001000000000000005100000000000000433065633533633436383264333666356334333539663461653762643762613100000000000002c2cb6e2630a90
1	112	749	0	0	(5, 32)	7	2306	24	\x7e010000000000007e0100000000000002000000000000005200000000000000433466366666553133613564373562326436613339323339323262339323265350000000000002c2cb6e2630a90
1	112	749	0	0	(5, 33)	7	2306	24	\x7f010000000000007f010000000000000300000000000000530000000000000043626565643133363032623962306536656362356235363866663530353866303700000000000002c2cb6e2630a90
1	112	749	0	0	(5, 34)	7	2306	24	\x800100000000000080010000000000000400000000000000540000000000000043303538346365353635633832346237623766353032383264396131393934356200000000000002c2cb6e2630a90
1	112	749	0	0	(5, 35)	7	2306	24	\x8101000000000000810100000000000005000000000000005500000000000000436463393132613235336431653962613430653263353937656432333736363430000000000002c2cb6e2630a90
1	112	749	0	0	(5, 36)	7	2306	24	\x8201000000000000820100000000000006000000000000005600000000000000433339365646466623338355631613139653965646466623330662316561343800000000000002c2cb6e2630a90
1	112	749	0	0	(5, 37)	7	2306	24	\x8301000000000000830100000000000007000000000000005700000000000000433865666231303061323232666634343637626238000000000000002c2cb6e2630a90
1	112	749	0	0	(5, 38)	7	2306	24	\x84010000000000008401000000000000080000000000000058000000000000004364396663356237336138643738666164336436646666653431393338346537300000000000002c2cb6e2630a90
1	112	749	0	0	(5, 39)	7	2306	24	\x85010000000000008501000000000000090000000000000059000000000000004363383661376565336438656630623535316564353865333534613833366632620000000000002c2cb6e2630a90
1	112	749	0	0	(5, 40)	7	2306	24	\x86010000000000008601000000000000000000000000005a0000000000000043613031613033383063613363363134323863323661323331663065343961303900000000000002c2cb6e2630a90
1	112	749	0	0	(5, 41)	7	2306	24	\x8701000000000000870100000000000001000000000000005b0000000000000043356134623235616165643235633265653162373464653732646330336331346500000000000002c2cb6e2630a90
1	112	749	0	0	(5, 42)	7	2306	24	\x8801000000000000880100000000000002000000000000005c0000000000000043663733623736636533966633239626632613533363666134323665386600000000000002c2cb6e2630a90
1	112	749	0	0	(5, 43)	7	2306	24	\x8901000000000000890100000000000003000000000000005d0000000000000043373063363330664663653306264653439653463646665653366200000000000002c2cb6e2630a90
1	112	749	0	0	(5, 44)	7	2306	24	\x8a010000000000008a0100000000000004000000000000005e0000000000000043323866306238363435393861313239313535376265643234386139393864346500000000000002c2cb6e2630a90
1	112	749	0	0	(5, 45)	7	2306	24	\x8b010000000000008b0100000000000005000000000000005f0000000000000043313534333834336134373233656432616223038653138303533616536646335620000000000002c2cb6e2630a90
1	112	749	0	0	(5, 46)	7	2306	24	\x8c010000000000008c0100000000000006000000000000006000000000000004366386331663233643661386438643739303466633065613865303636623362620000000000002c2cb6e2630a90
1	112	749	0	0	(5, 47)	7	2306	24	\x8d010000000000008d01000000000000070000000000000061000000000000004365343664653765316263616163655439613534663165396430643266383030640000000000002c2cb6e2630a90
1	112	749	0	0	(5, 48)	7	2306	24	\x8e010000000000008e01000000000000080000000000000062000000000000004362376231366563663863613533373233353933383343131363037313730306300000000000002c2cb6e2630a90
1	112	749	0	0	(5, 49)	7	2306	24	\x8f010000000000008f01000000000000090000000000000063000000000000004333353266653265646361326633636626463161636561000000000000002c2cb6e2630a90
1	112	749	0	0	(5, 50)	7	2306	24	\x900100000000000090010000000000000000000000000064000000000000004331386438303432333836623739653263323739666431363264663032303563380000000000002c2cb6e2630a90
1	112	749	0	0	(5, 51)	7	2306	24	\x910100000000000091010000000000000100000000000000650000000000000043383136623131326336313035623365626435333738323861333961663438313800000000000002c2cb6e2630a90
1	112	749	0	0	(5, 52)	7	2306	24	\x92010000000000009201000000000000020000000000000066000000000000004336396362336561333137613332633465363134336536363666462323062313400000000000002c2cb6e2630a90
1	112	749	0	0	(5, 53)	7	2306	24	\x93010000000000009301000000000000030000000000000067000000000000004362626639346233346562333232363861646135376133626535303632665376400000000000002c2cb6e2630a90
1	112	749	0	0	(5, 54)	7	2306	24	\x940100000000000094010000000000000400000000000000680000000000000043346634616463626633666336663665463666338613333832616332626631306100000000000002c

pageinspect

Heap items with attributes

```
my_db=# select * from heap_page_item_attrs(get_raw_page('bar', 1), 'bar'::regclass);
```

```
-[ RECORD 70 ]-----  
lp          | 70  
lp_off      | 352  
lp_flags    | 1  
lp_len      | 112  
t_xmin      | 749  
t_xmax      | 0  
t_field3    | 0  
t_ctid      | (1,70)  
t_infomask2 | 7  
t_infomask  | 2306  
t_hoff      | 24  
t_bits      |  
t_oid       |  
t_attrs     | {"\\x8c00000000000000", "\\x8c000000", "\\x0000000000000000", "\\x2800000000000000",  
  "\\x433133383539373465643539303461343338363136666637626462336637343339",  
  "\\x2ccb6e2630a90200", "\\x2ccb6e2630a90200"}
```

pageinspect

Heap page items with flags

```
my_db=# select * from heap_page_item_attrs(get_raw_page('bar', 1), 'bar'::regclass),
      lateral heap_tuple_infomask_flags(t_infomask, t_infomask2);
```

```
-[ RECORD 70 ]-----  
lp          | 70  
lp_off      | 352  
lp_flags    | 1  
lp_len      | 112  
t_xmin      | 749  
t_xmax      | 0  
t_field3    | 0  
t_ctid      | (1,70)  
t_infomask2  | 7  
t_infomask   | 2306  
t_hoff      | 24  
t_bits      |  
t_oid       |  
t_attrs     | {...}  
raw_flags   | {HEAP_HASVARWIDTH,HEAP_XMIN_COMMITTED,HEAP_XMAX_INVALID}  
combined_flags | {}
```

pageinspect

Heap page items with flags (after deleting a tuple)

```
my_db=# select * from heap_page_item_attrs(get_raw_page('bar', 1), 'bar'::regclass),
      lateral heap_tuple_infomask_flags(t_infomask, t_infomask2);
```

```
-[ RECORD 70 ]-----
lp          | 70
lp_off      | 352
lp_flags    | 1
lp_len      | 112
t_xmin      | 749
t_xmax      | 758
t_field3    | 0
t_ctid      | (1,70)
t_infomask2 | 8199
t_infomask  | 258
t_hoff      | 24
t_bits      |
t_oid       |
t_attrs     | {...}
raw_flags   | {HEAP_HASVARWIDTH,HEAP_XMIN_COMMITTED,HEAP_KEYS_UPDATED}
combined_flags | {}
```

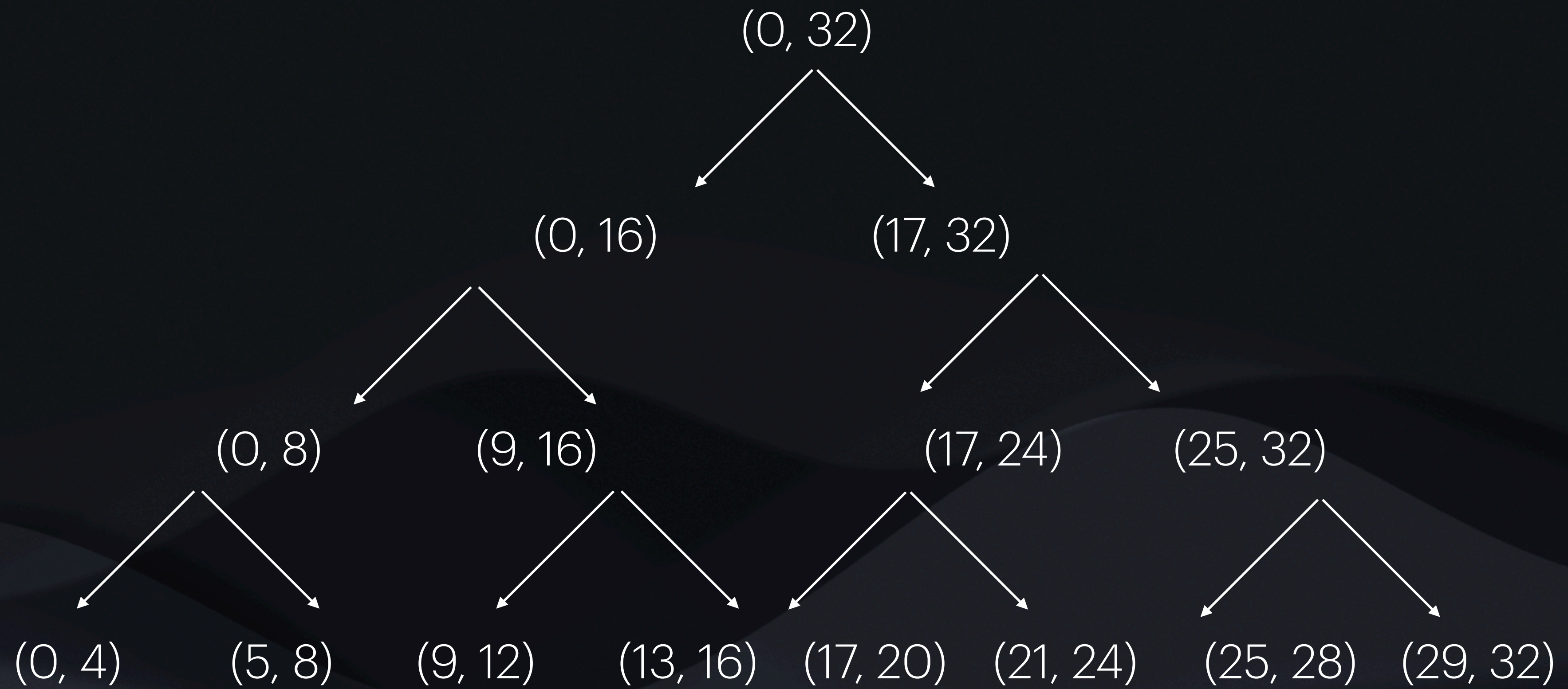
Heap Trade-offs

- MVCC implementation treats updates as inserts.
 - Simplifies MVCC as all visible tuple versions are physically present in the same heap.
 - Requires vacuuming dead tuples.
 - Updating a single attribute means copying the whole tuple.
- Bloat!
 - The per-page items array can end up with holes that are too small to hold a new tuple.

BTrees

BTrees

Not a binary tree, per se



and so on...

BTrees

Nodes and Pages

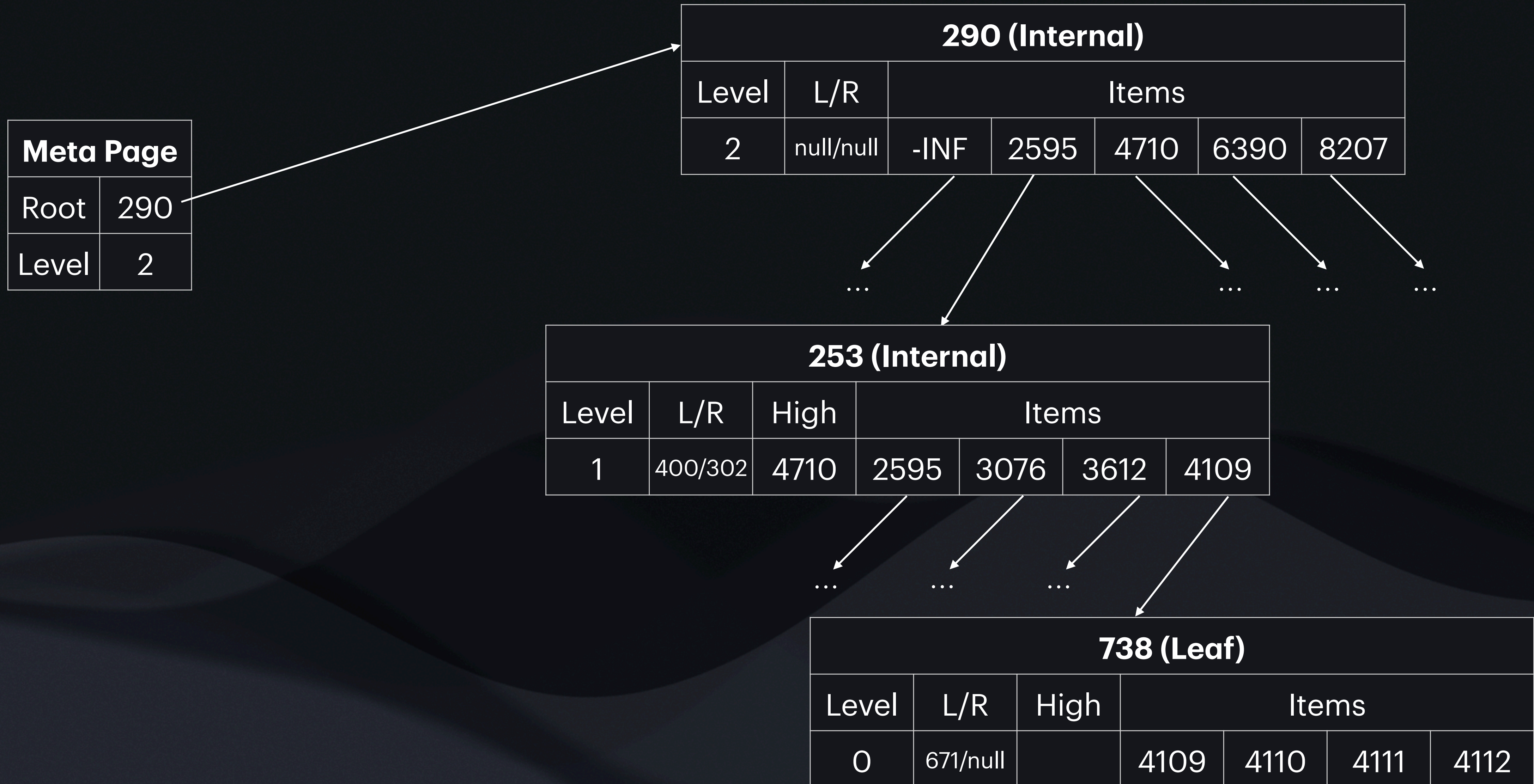
- Can store just the low key at each fork or the tree (rather than the full range).
- Each node in the tree is a whole page (i.e., 8 KB).
- Each level in the tree is a doubly-linked list (i.e., each page has left and right pointers).
- Two types of nodes:
 - Internal: contain tuples pointing to the next level of the tree.
 - Leaf: contain tuples pointing to rows in the table.

BTrees

Design choices

- PostgreSQL indexes are always indirect
 - Index entries contain a pointer to a physical row rather than a logical row.
 - Clustering on an index rarely makes sense.
- Multi-column indexes store the full index key at each tree level (as opposed to e.g. nested trees).

BTrees



BTrees

Tooling

```
my_db=# insert into bar2(i)
        select case
                when n % 5 < 3 then round(n * random())
                else n % 1_000_000
                end
        from generate_series(1, 10_000_000) s(n);
INSERT 0 10000000

my_db=# create index index_bar2_on_i on bar2(i);
CREATE INDEX
```

BTrees

Tooling

```
my_db=# select * from bt_metap('index_bar2_on_i');
```

```
-[ RECORD 1 ]-----+-----
```

magic	340322	#define BTREE_MAGIC 0x053162
version	4	
root	290	
level	2	
fastroot	290	
fastlevel	2	
last_cleanup_num_deletes	0	
last_cleanup_num_tuples	-1	
allequalimage	t	

BTrees

Tooling

```
my_db=# select * from bt_page_items('index_bar2_on_i', 290);
```

itemoffset	ctid	itemlen	nulls	vars	data	dead	htid	tids
1	(3,0)	8	f	f				
2	(289,1)	16	f	f	ee 72 00 00 00 00 00 00			
3	(575,1)	16	f	f	9e ef 00 00 00 00 00 00			
4	(860,1)	16	f	f	b0 70 01 00 00 00 00 00			
5	(1145,1)	16	f	f	d6 f4 01 00 00 00 00 00			
6	(1430,1)	16	f	f	d6 7b 02 00 00 00 00 00			
7	(1715,1)	16	f	f	0b 05 03 00 00 00 00 00			
8	(2000,1)	16	f	f	76 90 03 00 00 00 00 00			
9	(2285,1)	16	f	f	66 1d 04 00 00 00 00 00			
10	(2570,1)	16	f	f	40 ac 04 00 00 00 00 00			
11	(2855,1)	16	f	f	36 3c 05 00 00 00 00 00			

...

Note:

ee 72 00 00 = 29,422

36 3c 05 00 = 343,094

BTrees

Tooling

```
my_db=# select * from bt_page_items('index_bar2_on_i', 575);
```

itemoffset	ctid	itemlen	nulls	vars	data	dead	htid	tids
1	(857,1)	16	f	f	b0 70 01 00 00 00 00 00			
2	(572,0)	8	f	f				
3	(573,1)	16	f	f	10 f0 00 00 00 00 00 00			
4	(574,1)	16	f	f	82 f0 00 00 00 00 00 00			
5	(576,1)	16	f	f	f6 f0 00 00 00 00 00 00			
6	(577,1)	16	f	f	66 f1 00 00 00 00 00 00			
7	(578,1)	16	f	f	dd f1 00 00 00 00 00 00			
8	(579,1)	16	f	f	4d f2 00 00 00 00 00 00			
9	(580,1)	16	f	f	c4 f2 00 00 00 00 00 00			
10	(581,1)	16	f	f	39 f3 00 00 00 00 00 00			
11	(582,1)	16	f	f	ac f3 00 00 00 00 00 00			

...

Note:

b0 70 01 00 = 94,384

10 f0 00 00 = 61,456

ac f3 00 00 = 62,380

BTrees

Tooling

```
my_db=# select * from bt_page_items('index_bar2_on_i', 579);
```

itemoffset	ctid	itemlen	nulls	vars	data	dead	htid	tids
1	(16,1)	16	f	f	c4 f2 00 ...			
2	(16,8204)	88	f	f	4d f2 00 ...	f	(274,105)	{“(274,105)”,“(382,84)”,...}
3	(16,8196)	40	f	f	4e f2 00 ...	f	(358,157)	{“(358,157)”,“(7194,188)”,...}
4	(16,8197)	48	f	f	4f f2 00 ...	f	(731,189)	{“(731,189)”,“(2704,96)”,...}
5	(14571,114)	16	f	f	50 f2 00 ...	f	(14571,114)	
6	(16,8204)	88	f	f	51 f2 00 ...	f	(274,109)	{“(274,109)”,“(1506,109)”,...}
7	(16,8205)	96	f	f	52 f2 00 ...	f	(274,110)	{“(274,110)”,“(448,162)”,...}
8	(16,8194)	32	f	f	53 f2 00 ...	f	(811,105)	{“(811,105)”,“(1498,169)”}
9	(16,8197)	48	f	f	54 f2 00 ...	f	(798,3)	{“(798,3)”,“(1602,129)”,...}
10	(16,8194)	32	f	f	55 f2 00 ...	f	(365,182)	{“(365,182)”,“(1764,147)”}
11	(16,8202)	80	f	f	56 f2 00 ...	f	(274,114)	{“(274,114)”,“(4699,64)”,...}

...

Note:

c4 f2 00 00 = 62,148

4d f2 00 00 = 62,029

56 f2 00 00 = 62,038

BTrees

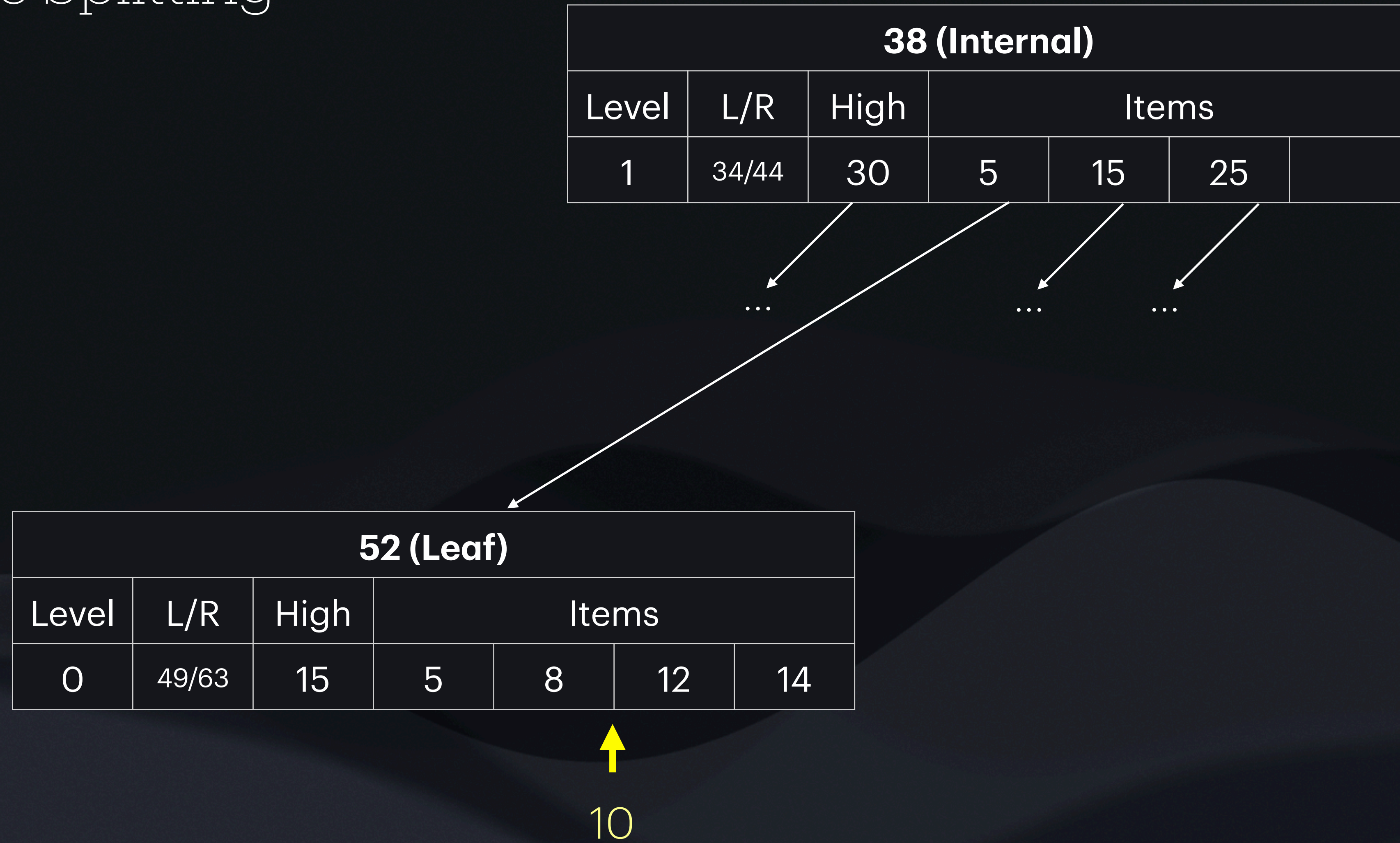
Tooling

```
my_db=# select * from bar2 where ctid in ('(274,114)', '(4699,64)');
```

pk	i
62038	62038
1062038	62038

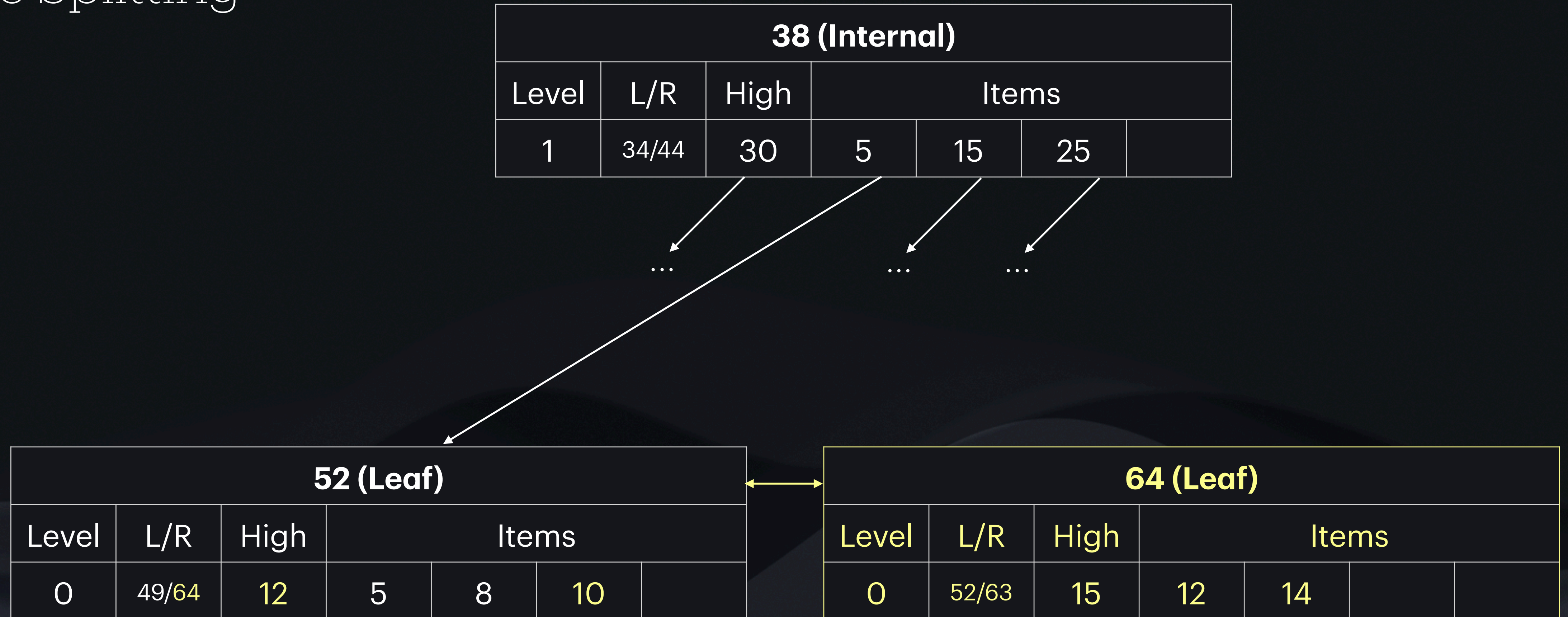
BTrees

Page Splitting



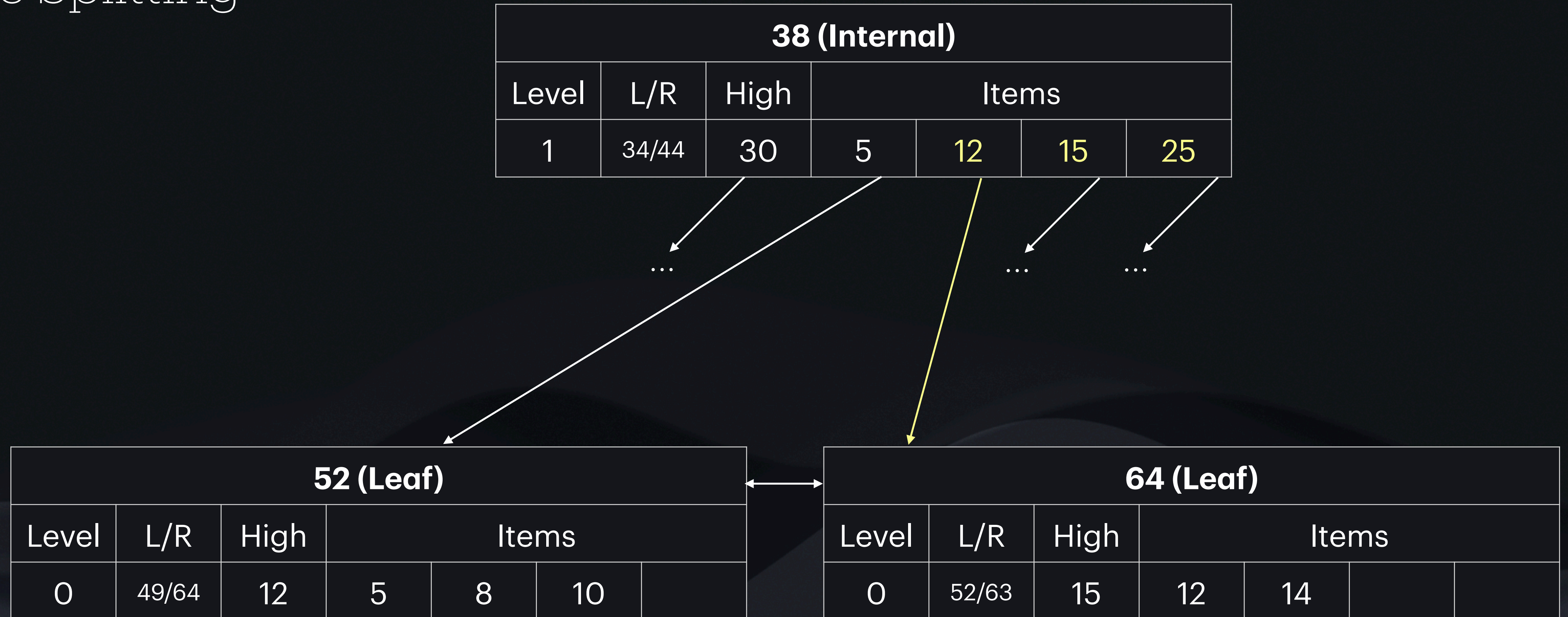
BTrees

Page Splitting



BTrees

Page Splitting



BTrees

MVCC

- New versions of the same logical tuple are inserted as separate physical entries in the index.
 - Old versions must be cleaned up later (either opportunistically or when vacuuming).
 - Bloat!
 - WAL writes *can be generated by index scans*.
 - This may also require splitting a page!
 - Can encode bloat structurally in the index structure (lots of dead space).

BTrees

MVCC

- HOT (Heap Only Tuple) Updates
 - When an update 1.) doesn't logically change the entry in an index and 2.) there's space for the new version on the same heap page, the index update can be skipped.
 - Another place where reads can generate WAL traffic (cleaning up dead tuples).

BTrees

MVCC

```
my_db=# select * from heap_page_item_attrs(get_raw_page('bar3', 1), 'bar3'::regclass),
      lateral heap_tuple_infomask_flags(t_infomask, t_infomask2)
      where lp = 127;
```

```
-[ RECORD 1 ]-----+-----
lp           | 127
lp_off       | 4128
lp_flags     | 1
lp_len       | 28
t_xmin       | 826
t_xmax       | 0
t_field3     | 0
t_ctid       | (1,127)
t_infomask2  | 1
t_infomask   | 2304
t_hoff       | 24
t_bits       |
t_oid        |
t_attrs      | {"\x23040000",NULL}
raw_flags    | {HEAP_XMIN_COMMITTED,HEAP_XMAX_INVALID}
combined_flags | {}
```

BTrees

MVCC

```
my_db=# select * from heap_page_item_attrs(get_raw_page('bar3', 1), 'bar3'::regclass),
      lateral heap_tuple_infomask_flags(t_infomask, t_infomask2)
      where lp = 127;
```

```
-[ RECORD 1 ]-----+-----
lp           | 127
lp_off       | 4128
lp_flags     | 1
lp_len       | 28
t_xmin       | 826
t_xmax       | 831
t_field3     | 0
t_ctid       | (1,184)
t_infomask2  | 16385
t_infomask   | 256
t_hoff       | 24
t_bits       |
t_oid        |
t_attrs      | {"\x23040000",NULL}
raw_flags    | {HEAP_XMIN_COMMITTED,HEAP_HOT_UPDATED}
combined_flags | {}
```

BTrees

MVCC

```
my_db=# select * from heap_page_item_attrs(get_raw_page('bar3', 1), 'bar3'::regclass),
      lateral heap_tuple_infomask_flags(t_infomask, t_infomask2)
      where lp = 184;
```

```
-[ RECORD 1 ]-----+-----
lp           | 184
lp_off       | 2304
lp_flags     | 1
lp_len       | 32
t_xmin       | 831
t_xmax       | 0
t_field3     | 0
t_ctid       | (1,184)
t_infomask2  | 32770
t_infomask   | 10240
t_hoff       | 24
t_bits       |
t_oid        |
t_attrs      | {"\\x23040000", "\\x05000000"}
raw_flags    | {HEAP_XMAX_INVALID,HEAP_UPDATED,HEAP_ONLY_TUPLE}
combined_flags | {}
```

BTrees

MVCC

```
my_db=# select * from heap_page_item_attrs(get_raw_page('bar3', 1), 'bar3'::regclass),
      lateral heap_tuple_infomask_flags(t_infomask, t_infomask2)
      where lp = 127;
```

```
-[ RECORD 1 ]---+-----
```

lp	127
lp_off	184
lp_flags	2
lp_len	0
t_xmin	
t_xmax	
t_field3	
t_ctid	
t_infomask2	
t_infomask	
t_hoff	
t_bits	
t_oid	
t_attrs	
raw_flags	
combined_flags	

```
#define LP_REDIRECT 2 /* HOT redirect (should have lp_len=0) */
```

Wrapping it all up

Data Modeling Implications

Storage size

- Lots of small tuples have (relatively) high storage cost.
- Sparse columns are (relatively) cheap.
- Ordering columns by alignment requirements can save measurable space.
- Append-only workloads can avoid bloat.
- Use **fillfactor** to tune HOT frequency.
- Indexes on columns you don't query by are more expensive than you realize.

Data Modeling Implications

Performance

- Bloat can significantly impact performance.
- TOAST can be slow (don't abuse large fields; use a file system for those images!)
- Inserts/updates of pseudo-random values in a BTree.
 - Cache locality
 - Full page writes to WAL

Additional Topics

- Buffer cache (as well as the Linux page cache)
- FSM (free space map): Optimizes finding a page with free space. See also the `pg_freespace` module for inspection.
- Visibility map: Separate relation fork for heap tables; tracks pages with all-visible and all-frozen rows. Optimizes vacuum and supports index-only scans.
- `amcheck`: Tool to verify table and index consistency
- `pageinspect` functions for other index types

<https://www.postgresql.org/docs/current/storage-fsm.html>

<https://www.postgresql.org/docs/current/storage-vm.html>

<https://www.postgresql.org/docs/current/amcheck.html>

Q/A