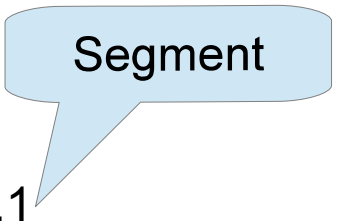
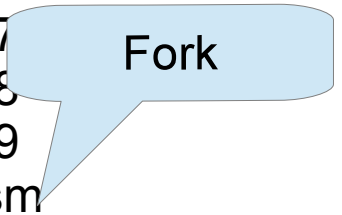
Compressed file system

Konstantin Knizhnik

Postgres file layer

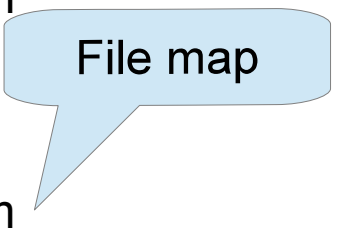
```
ls -l pgsql/base/12449/
total 15350632
-rw------- 1 knizhnik knizhnik 1073741824 Jul 20 09:43 16396
-rw------- 1 knizhnik knizhnik 1073741824 Jul 20 09:43 16396.1
-rw------- 1 knizhnik knizhnik 1073741824 Jul 20 09:43 16396.10
-rw------- 1 knizhnik knizhnik 1073741824 Jul 20 09:43 16396.11
-rw------- 1 knizhnik knizhnik  569958400 Jul 20 09:43 16396.12
-rw------- 1 knizhnik knizhnik 1073741824 Jul 20 09:43 16396.2
-rw------- 1 knizhnik knizhnik 1073741824 Jul 20 09:43 16396.3
-rw------- 1 knizhnik knizhnik 1073741824 Jul 20 09:43 16396.4
-rw------- 1 knizhnik knizhnik 1073741824 Jul 20 09:43 16396.5
-rw------- 1 knizhnik knizhnik 1073741824 Jul 20 09:43 16396.6
-rw------- 1 knizhnik knizhnik 1073741824 Jul 20 09:43 16396.7
-rw------- 1 knizhnik knizhnik 1073741824 Jul 20 09:43 16396.8
-rw------- 1 knizhnik knizhnik 1073741824 Jul 20 09:43 16396.9
-rw------- 1 knizhnik knizhnik    3325952 Jul 20 09:43 16396_fsm
-rw------- 1 knizhnik knizhnik     417792 Jul 20 09:43 16396_vm
```



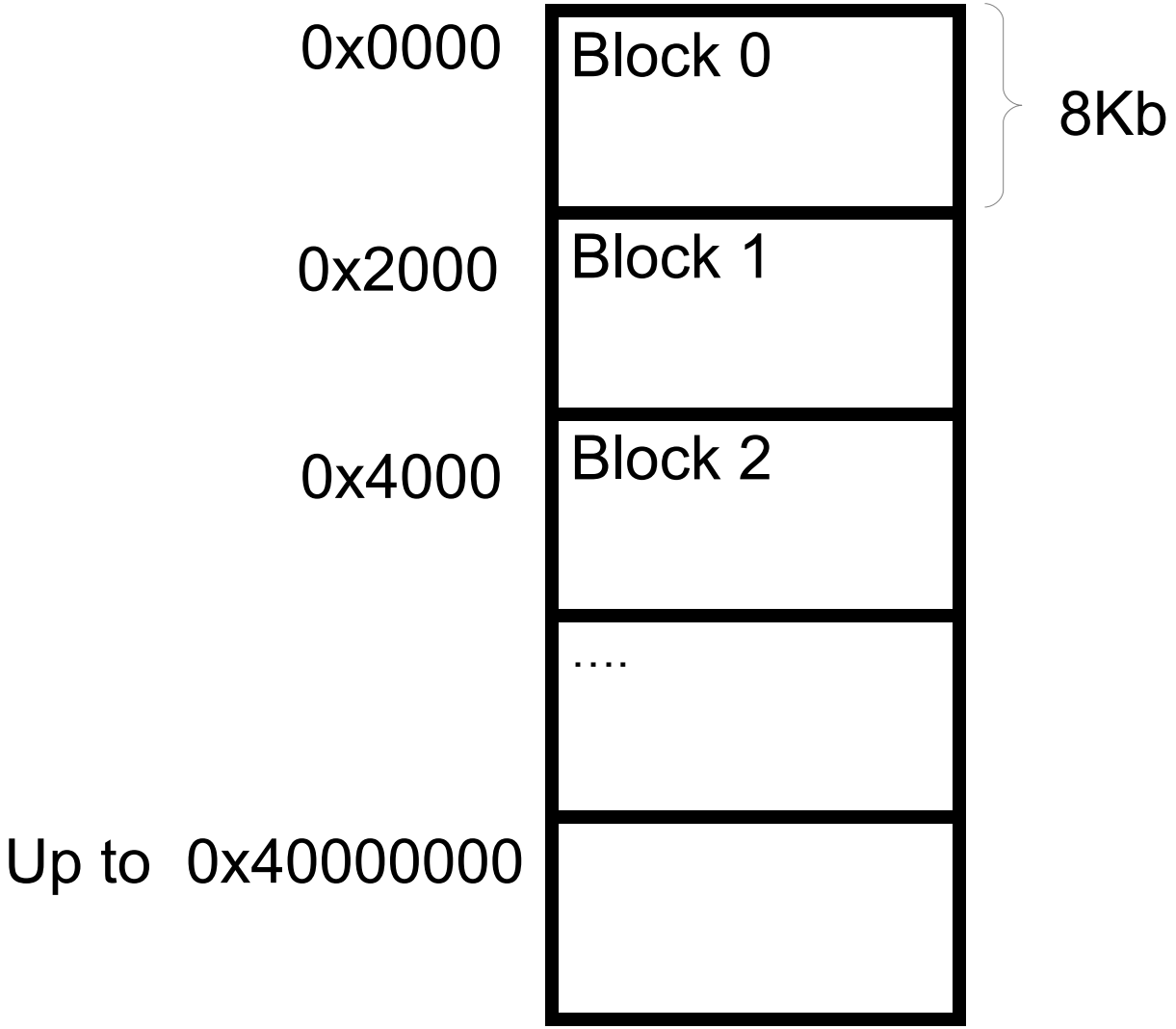


Compressed tablespace

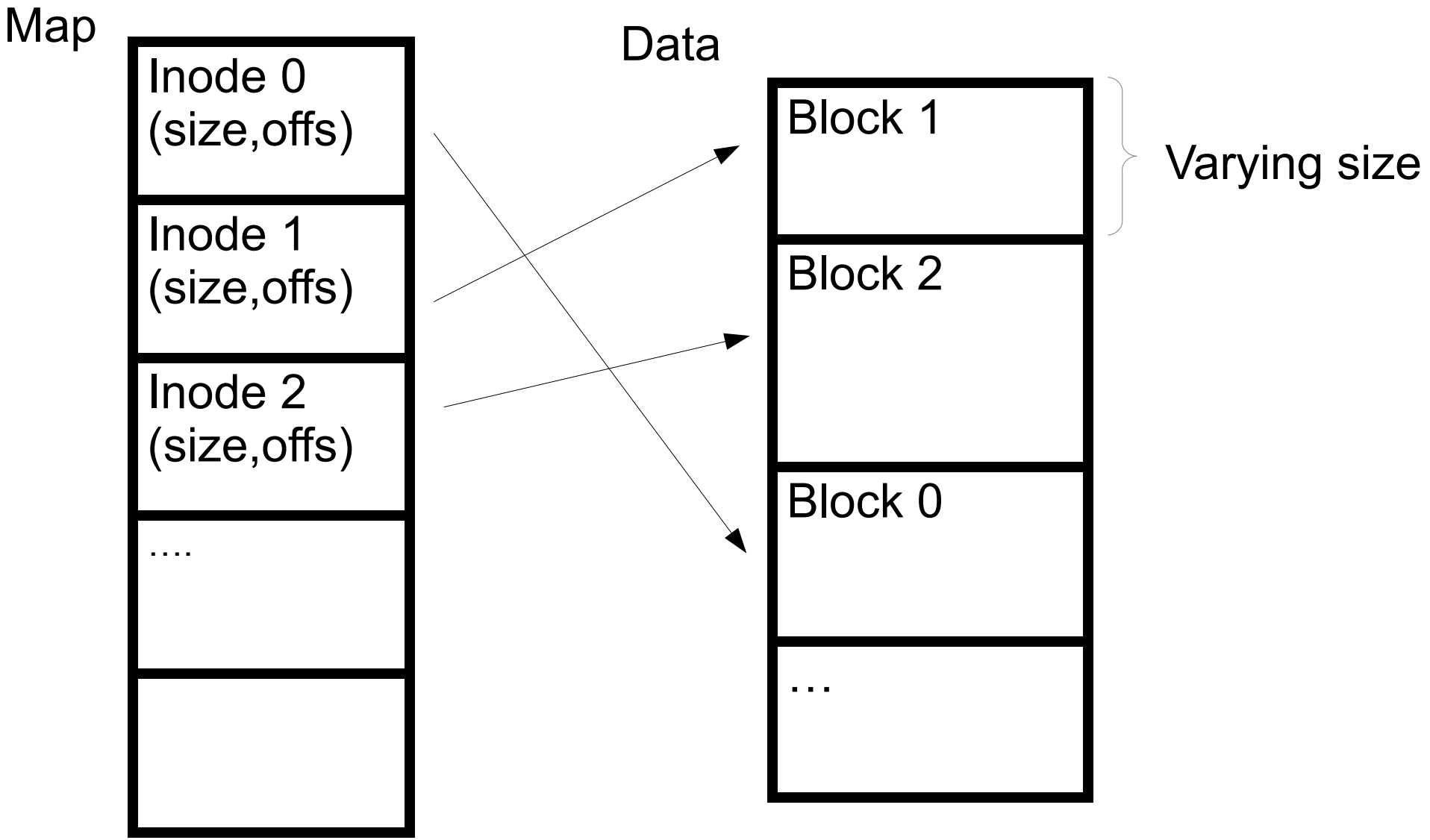
```
ls -l zfs/PG_9.6_201607071/12449/
total 872620
-rw------- 1 knizhnik knizhnik   562316 Jul 20 10:08 16388
-rw------- 1 knizhnik knizhnik  1048600 Jul 20 10:08 16388.cfm
-rw------- 1 knizhnik knizhnik    24576 Jul 20 10:08 16388_fsm
-rw------- 1 knizhnik knizhnik     8192 Jul 20 10:08 16388_vm
-rw------- 1 knizhnik knizhnik    88874 Jul 20 10:08 16394
-rw------- 1 knizhnik knizhnik  1048600 Jul 20 10:08 16394.cfm
-rw------- 1 knizhnik knizhnik    24576 Jul 20 10:08 16394_fsm
-rw------- 1 knizhnik knizhnik     8192 Jul 20 10:08 16394_vm
-rw------- 1 knizhnik knizhnik 67947970 Jul 20 10:08 16397
-rw------- 1 knizhnik knizhnik 72327904 Jul 20 10:08 16397.1
-rw------- 1 knizhnik knizhnik  1048600 Jul 20 10:08 16397.1.cfm
-rw------- 1 knizhnik knizhnik 73965221 Jul 20 10:08 16397.10
-rw------- 1 knizhnik knizhnik  1048600 Jul 20 10:08 16397.10.cfm
-rw------- 1 knizhnik knizhnik 68394290 Jul 20 10:08 16397.11
-rw------- 1 knizhnik knizhnik  1048600 Jul 20 10:08 16397.11.cfm
-rw------- 1 knizhnik knizhnik 37390503 Jul 20 10:08 16397.12
-rw------- 1 knizhnik knizhnik  1048600 Jul 20 10:08 16397.12.cfm
-rw------- 1 knizhnik knizhnik 68090079 Jul 20 10:08 16397.2
-rw------- 1 knizhnik knizhnik  1048600 Jul 20 10:08 16397.2.cfm
```



Postgres segment



CFS segment


Map
Inode 0
(size,offs)
Inode 1
(size,offs)
Inode 2
(size,offs)
....
Data
Block 1
Block 2
Block 0
...
Varying size

CFS map structure

```
#define CFS_INODE_SIZE(inode) ((uint32)((inode) >> 32))
#define CFS_INODE_OFFS(inode) ((uint32)(inode))
#define CFS_INODE(size,offs)  (((inode_t)(size) << 32) | (offs))

typedef uint64 inode_t;

typedef struct
{
    pg_atomic_uint32 physSize;
    pg_atomic_uint32 virtSize;
    pg_atomic_uint32 usedSize;
    pg_atomic_uint32 lock;
    uint64           generation;
    inode_t          inodes[RELSEG_SIZE];
} FileMap;
```

Postgres PROFESSIONAL

Lockless access using atomics

```
/*
 * Get position for storing updated page
 */
uint32 cfs_alloc_page(FileMap* map, uint32 oldSize, uint32 newSize)
{
    pg_atomic_fetch_add_u32(&map->usedSize, newSize - oldSize);
    return pg_atomic_fetch_add_u32(&map->physSize, newSize);
}

/*
 * Update logical file size
 */
void cfs_extend(FileMap* map, uint32 newSize)
{
    uint32 oldSize = pg_atomic_read_u32(&map->virtSize);
    while (newSize > oldSize
        && !pg_atomic_compare_exchange_u32(&map->virtSize,
                                           &oldSize, newSize));
}
```

CFS pros and contras

Advantages	Disadvantages
Better compression: The whole page is compressed including record headers	**Extra level of indirection:** Page is access through map
Minimal changes in Postgres core: CFS works at lowest level	**Fragmentation:** Needs garbage collector to defragment data file
Better locality: New pages are always written to new place sequentially	**Worse memory utilization:** Page pool keeps uncompressed pages
Flexibility: Easy adding of new compression algorithms	**Inefficient replication:** Replica has to perform compression and GC itself
Speed: No overhead for cached pages	**No domain specific compression:** Type info is not available

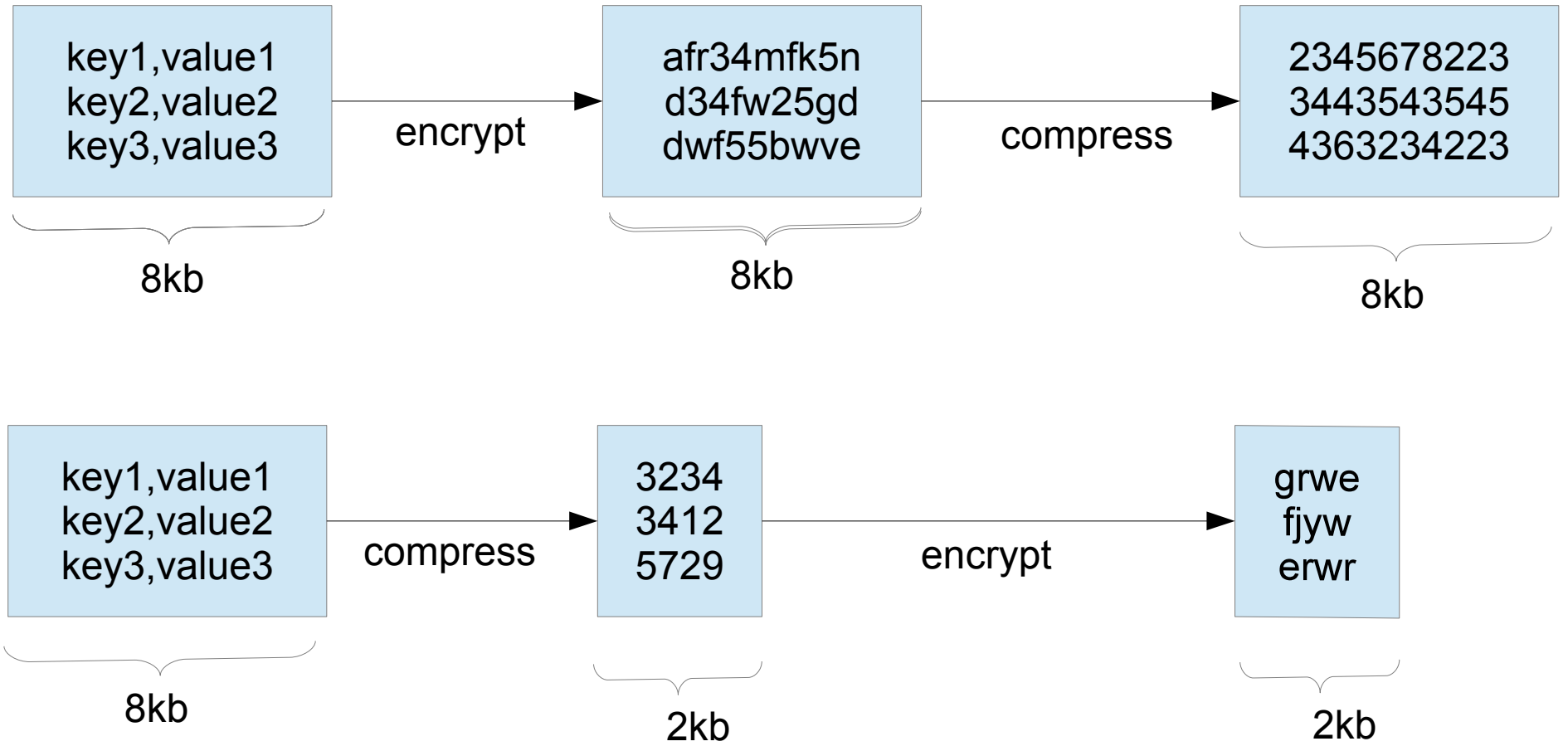
CFS usage

```
create tablespace cfs location
'/home/knizhnik/dtm-data/cfs'
with (compression=true);

set default_tablespace=cfs;

insert into foo values
(generate_series(1, 1000000
update foo set x=x+1;

select cfs_start_gc(4);
```

New tablespace property

Number of workers

Encryption and compression



CFS parameters

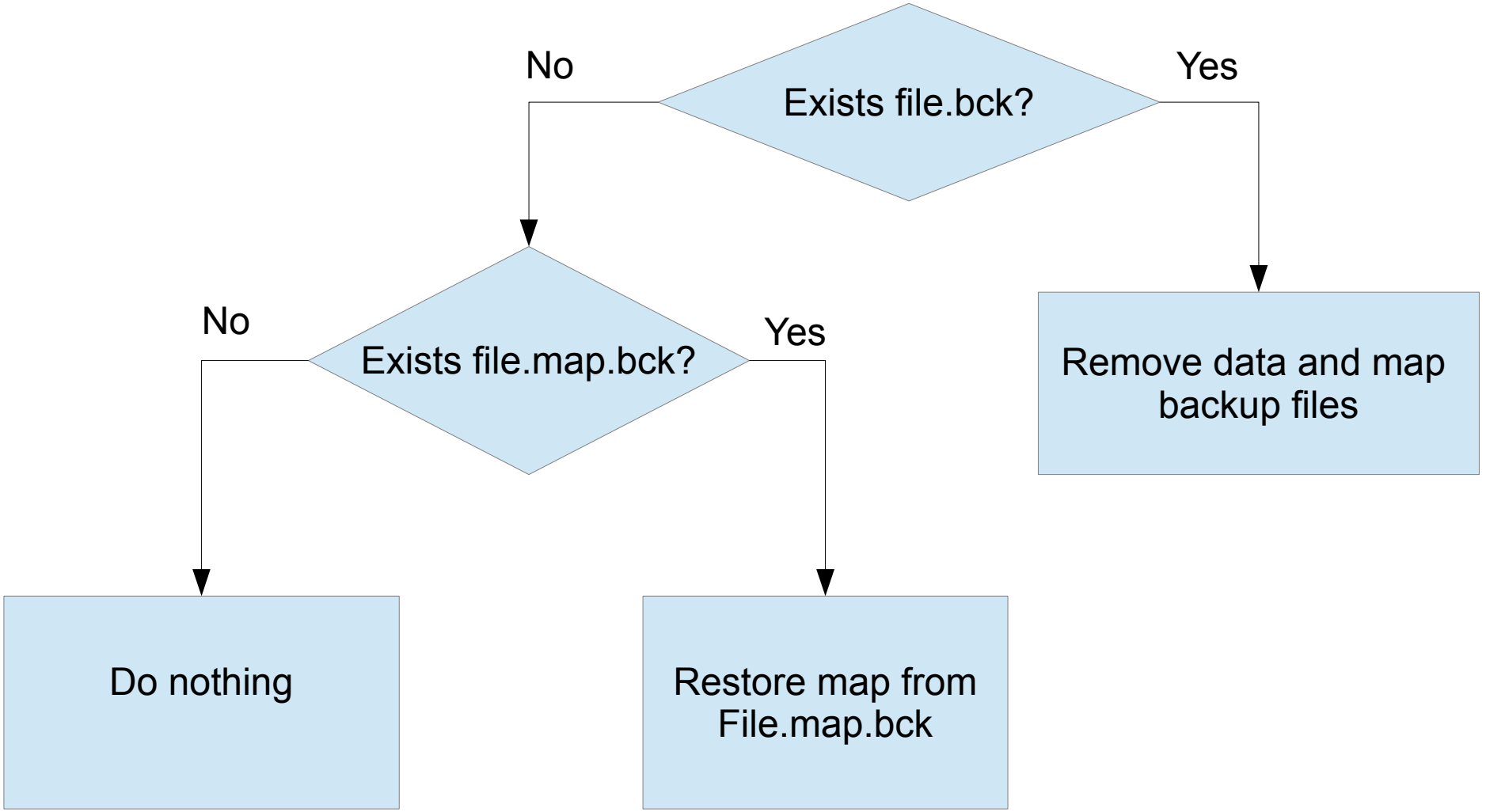
Parameter	Description	Default value
cfs_gc_workers	Number of background workers performing CFS garbage collection.	1
cfs_gc_threashold	Percent of garbage in the file after which defragmentation started	50%
cfs_encryption	Perform encryption of pages	off
cfs_gc_period	Interval between CFS garbage collection iterations	5 seconds
cfs_gc_delay	Delay between files defragmentation	0 milliseconds

Postgres PROFESSIONAL

Garbage collection

- Copy used pages to new data file
- Create copy of map file
- Flush copies of data and map files
- Atomic rename copy of data file
- Inplace update of map file
- Flush new map file
- Remove copy of map file

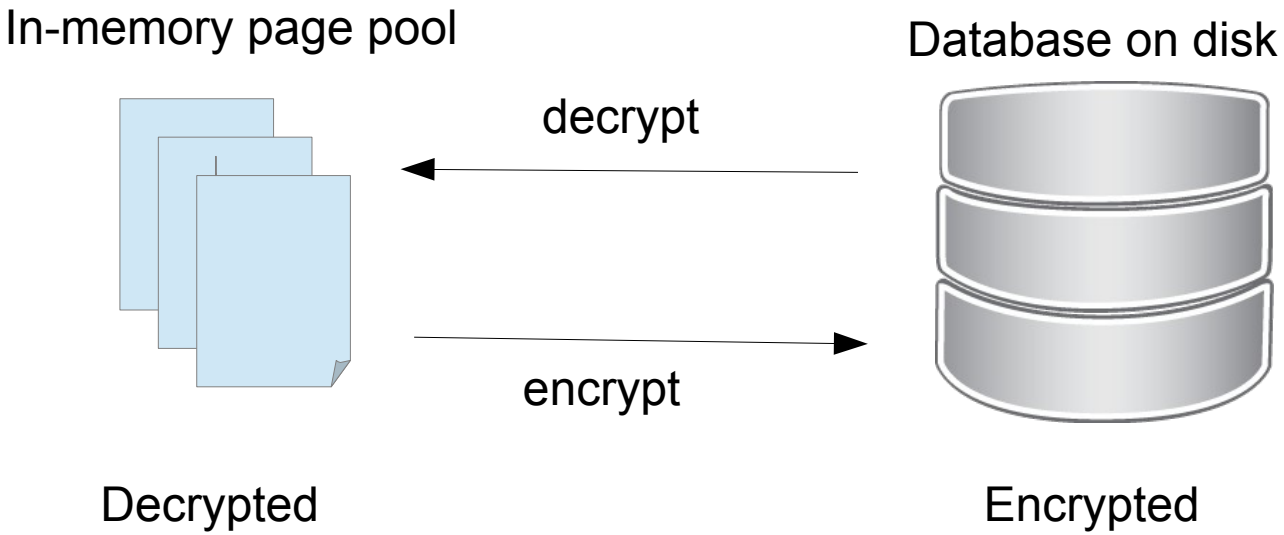
CFS recovery



Page encryption

Environment variable

```
$ export PG_CIPHER_KEY=”my top secrete”
```



Postgres PROFESSIONAL

Comparison of compression algorithms

pgbench -i -s 1000

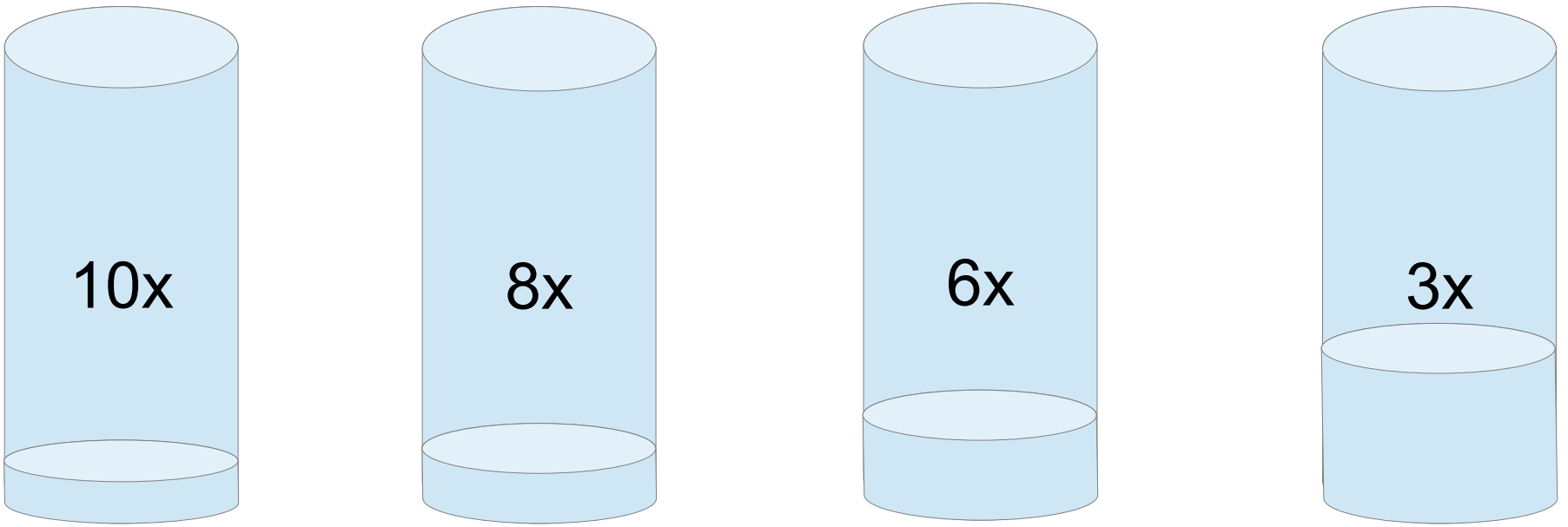
Conf guration	Size (Gb)	Time (sec)
vanilla postgres	15.31	92
zlib (default level)	2.37	284
zlib (best speed)	2.43	191
postgres internal lz	3.89	214
lz4	4.12	95
snappy (google)	5.18	99
lzfse (apple)	2.80	1099
zstd (facebook)	1.69	125

Performance

(with background garbage collector)

File Storage	Standard	CFS
Initailization (sec): pgbench -s 1000 -i	98	214
Database size (Mb):	15334	827
Benchmark (TPS): pgbench -c 10 -j 10 -t 10000	3904	4126

Compression ratio



Synthetic data

Financial data

Telecom data

Astronomic data

TODO

- Compression/decompression of existed tables (support of “alter tablespace”)
- Support multiple compression algorithms and make it possible for user to make choice
- Support compression in default tablespace?
- Optimization for append only tables? (no map is needed)