
RAI 1

Why `fsync()` on OpenZFS can’t fail (and what happens when it does)

Rob Norris, Klara Inc.

Hello!

#robnfacts

- He/him
- Australian
- 🇦🇺🏳️‍🌈🇦🇺
- One wife, two cats, three kids
- 1989 - 1999: Kid messing with computers
- 1999 - 2023: Linux sysadmin
- 2023 - ....: OpenZFS developer
- 2023 - ....: one FreeBSD server
- Hundreds of side-projects and dumb experiments

#notrobnfacts


from a
parallel
universe


SYNC


Seagate
Laptop SSHD
1000GB
DOCUMENTS
zip
PC100
SONY
CD-R
700MB

/usr/bin/boyband

```
int main(int argc, char **argv) {

    return (0);
}
```

/usr/bin/boyband

```c
int main(int argc, char **argv) {
    void *song_data;
    size_t song_len;
    generate_song(&song_data, &song_len, TYPE_DANCE|TYPE_LOVE);









    return (0);
}
```

/usr/bin/boyband

```
int main(int argc, char **argv) {
    void *song_data;
    size_t song_len;
    generate_song(&song_data, &song_len, TYPE_DANCE|TYPE_LOVE);



    int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
    write(fd, song_data, song_len);
    close(fd);





    return (0);
}
```

/usr/bin/boyband

```
int main(int argc, char **argv) {
    void *song_data;
    size_t song_len;
    generate_song(&song_data, &song_len, TYPE_DANCE|TYPE_LOVE);

    time_t start = time();

    int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
    write(fd, song_data, song_len);
    close(fd);

    time_t end = time();
    printf("wrote '%s' in %d seconds\n", argv[0], end-start);

    return (0);
}
```

/usr/bin/boyband

```
$ boyband debut_song.wav
wrote 'debut_song.wav' in 30 seconds
```

💾 Writing a file

💾	name	data

💾 Writing a file

```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
```

💾	name	data
🔵	debut_song.wav	

💾 Writing a file

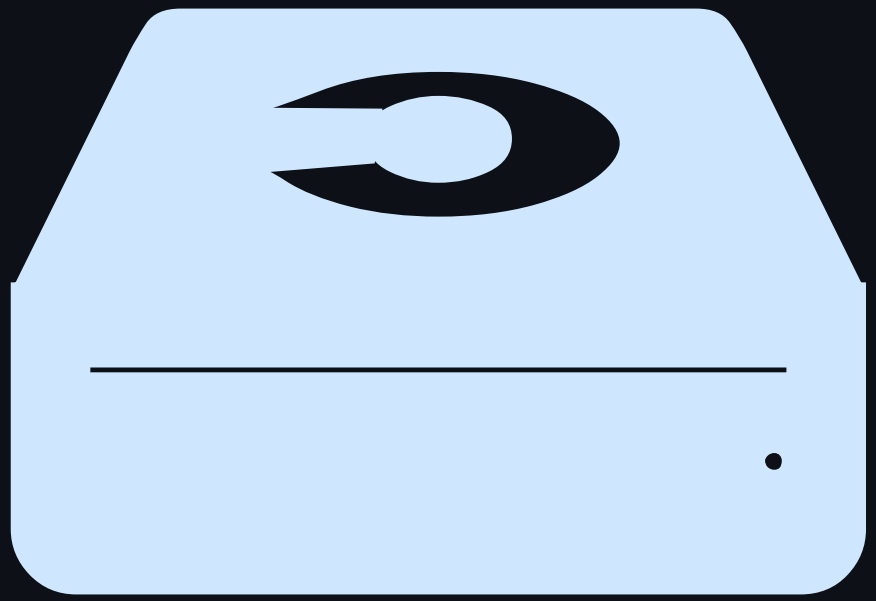
```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
write(fd, song_data, song_len);
```

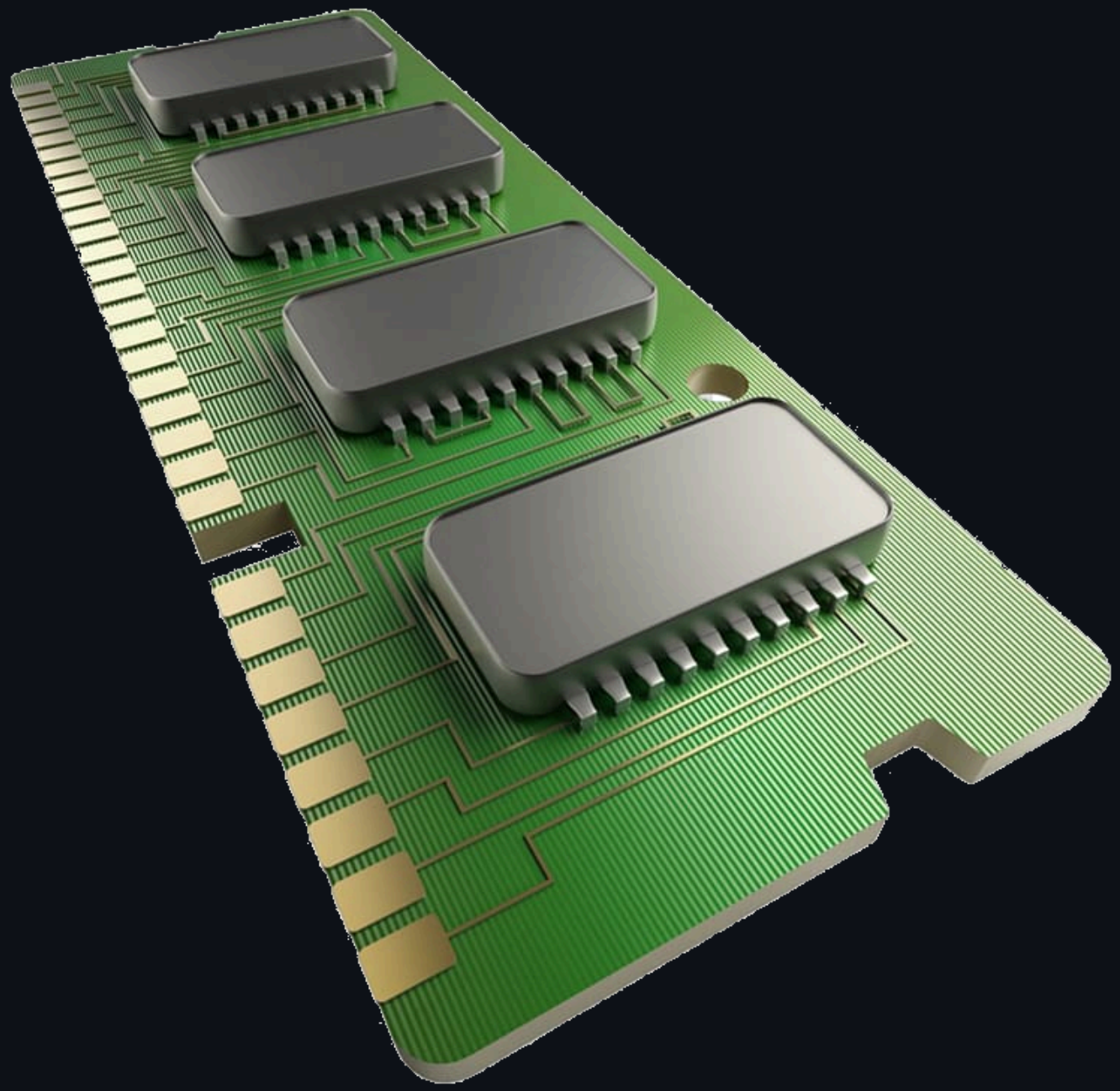
💾	name	data
🔵	debut_song.wav	🎶🎶🎶🎶🎶🎶🎶🎶🎶🎶🎶🎶

💾 Writing a file

```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
write(fd, song_data, song_len);
close(fd);
```

💾	name	data
	debut_song.wav	🎶🎶🎶🎶 🎶🎶🎶🎶 🎶🎶🎶🎶





/usr/bin/boyband

```
$ boyband hit_song.wav
wrote 'hit_song.wav' in 3 seconds
```

Writing a file

	name	data
	debut_song.wav	

Writing a file

```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
```

	name	data
	debut_song.wav	🎵🎵🎵🎵 🎵🎵🎵🎵 🎵🎵🎵🎵
🔵	hit_song.wav	

Writing a file

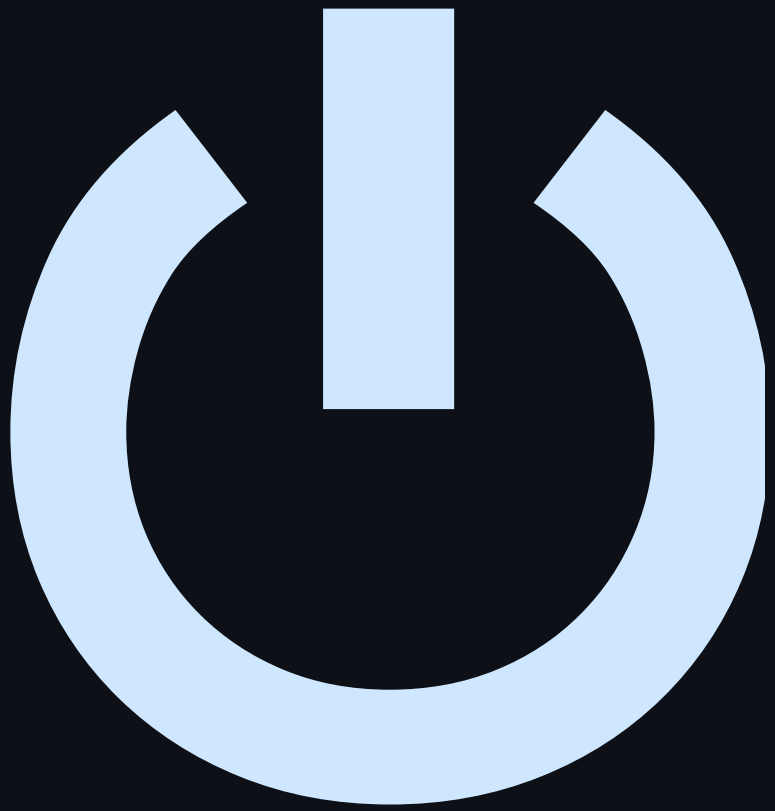
```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
write(fd, song_data, song_len);
```

	name	data
	debut_song.wav	🎵🎵🎵🎵 🎵🎵🎵🎵 🎵🎵🎵🎵
🔵	hit_song.wav	🎵🎵🎵🎵 🎵🎵🎵🎵 🎵🎵🎵🎵 🎵🎵🎵🎵

Writing a file

```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
write(fd, song_data, song_len);
close(fd);
```

	name	data
	debut_song.wav	🎵🎵🎵🎵 🎵🎵🎵🎵 🎵🎵🎵🎵
	hit_song.wav	🎵🎵🎵🎵 🎵🎵🎵🎵 🎵🎵🎵🎵 🎵🎵🎵🎵



	name	data
	debut_song.wav	

Writing a file (with cache)

	name	data

	name	data
	debut_song.wav	

Writing a file (with cache)

```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
```

	name	data
	hit_song.wav	

	name	data
	debut_song.wav	

Writing a file (with cache)

```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
write(fd, song_data, song_len);
```

	name	data
	hit_song.wav	

	name	data
	debut_song.wav	

Writing a file (with cache)

```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
write(fd, song_data, song_len);
close(fd);
```

	name	data
	hit_song.wav	

	name	data
	debut_song.wav	

⏻ Power off

⚡⚡⚡⚡⚡

	name	data
	debut_song.wav	🎶🎶🎶🎶🎶🎶🎶🎶🎶🎶🎶🎶

fsync()

Writing a file (with `fsync()`)

	name	data

	name	data
	debut_song.wav	

Writing a file (with `fsync()`)

```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
```

	name	data
	hit_song.wav	

	name	data
	debut_song.wav	

Writing a file (with `fsync()`)

```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
write(fd, song_data, song_len);
```

	name	data
	hit_song.wav	

	name	data
	debut_song.wav	

Writing a file (with `fsync()`)

```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
write(fd, song_data, song_len);
fsync(fd);
```

	name	data

	name	data
	debut_song.wav	
	hit_song.wav	

Writing a file (with `fsync()`)

```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
write(fd, song_data, song_len);
fsync(fd);
close(fd);
```

	name	data

	name	data
	debut_song.wav	
	hit_song.wav	

/usr/bin/boyband

```
$ boyband hit_song.wav
wrote 'hit_song.wav' in 10 seconds
```

/usr/bin/boyband

```
$ boyband best_song.wav
wrote 'best_song.wav' in 5 seconds
```


SPAIN


System Request
Disk structure corrupt
Use DISKDOCTOR to correct it
Retry
Cancel

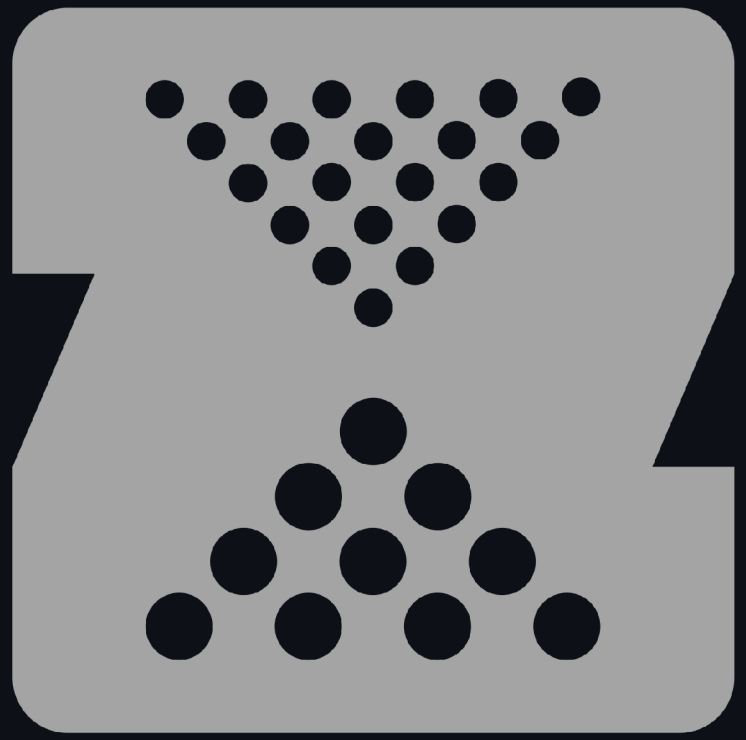
```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
write(fd, song_data, song_len);
fsync(fd);
close(fd);
```

```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);




        write(fd, &song_data[pos], song_len);




    fsync(fd);



close(fd);
```

```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
if (fd < 0) {
    perror("open");
    exit (EX_IOERR);
}



          write(fd, &song_data[pos], song_len);









    fsync(fd);




close(fd);
```

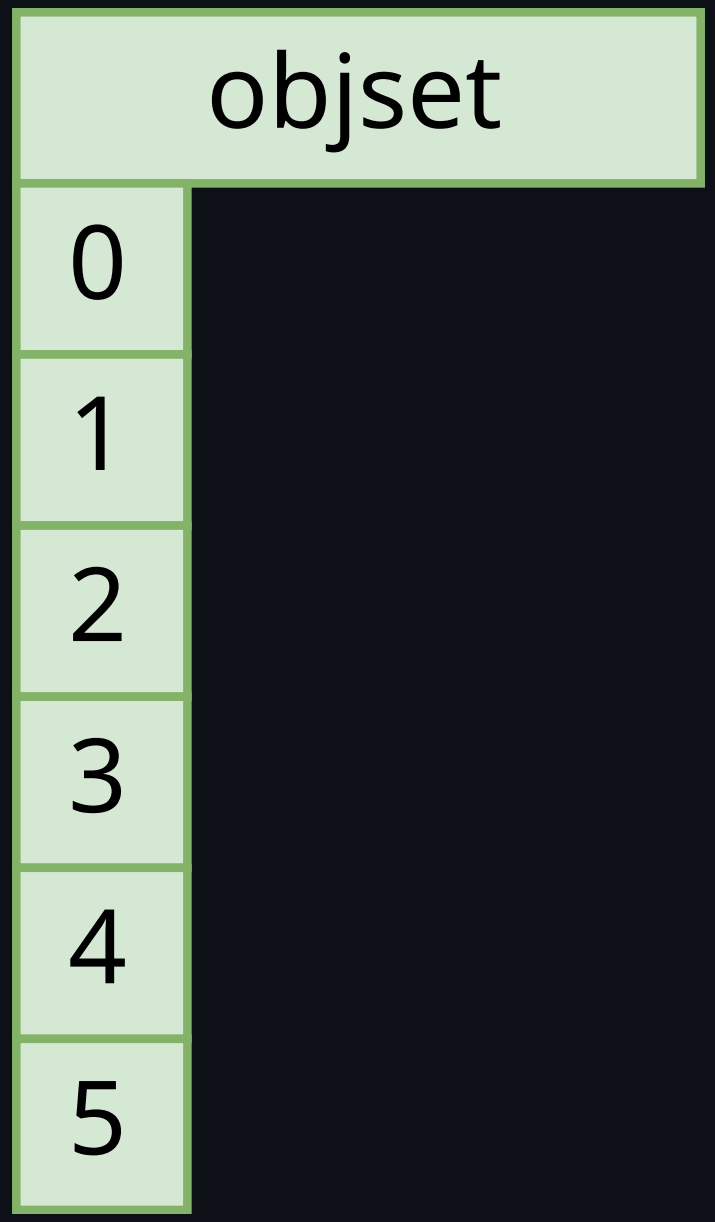
```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
if (fd < 0) {
    perror("open");
    exit (EX_IOERR);
}

ssize_t pos = 0;
while (pos < song_len) {
    pos = write(fd, &song_data[pos], song_len);
    if (pos < 0) {
        perror("write");
        exit (EX_IOERR);
    }
    song_len -= pos;
}

    fsync(fd);




close(fd);
```

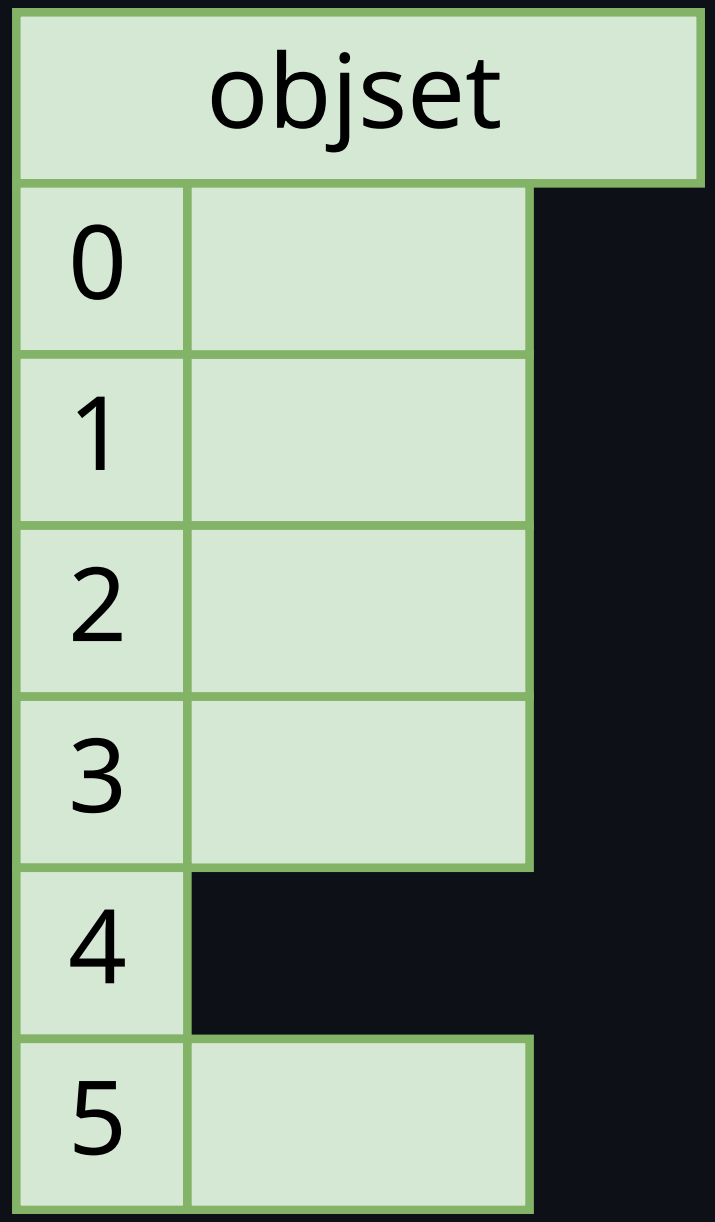
```
int fd = open(argv[0], O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
if (fd < 0) {
    perror("open");
    exit (EX_IOERR);
}

ssize_t pos = 0;
while (pos < song_len) {
    pos = write(fd, &song_data[pos], song_len);
    if (pos < 0) {
        perror("write");
        exit (EX_IOERR);
    }
    song_len -= pos;
}

if (fsync(fd) < 0) {
    perror("fsync");
    exit (EX_IOERR);
}

close(fd);
```

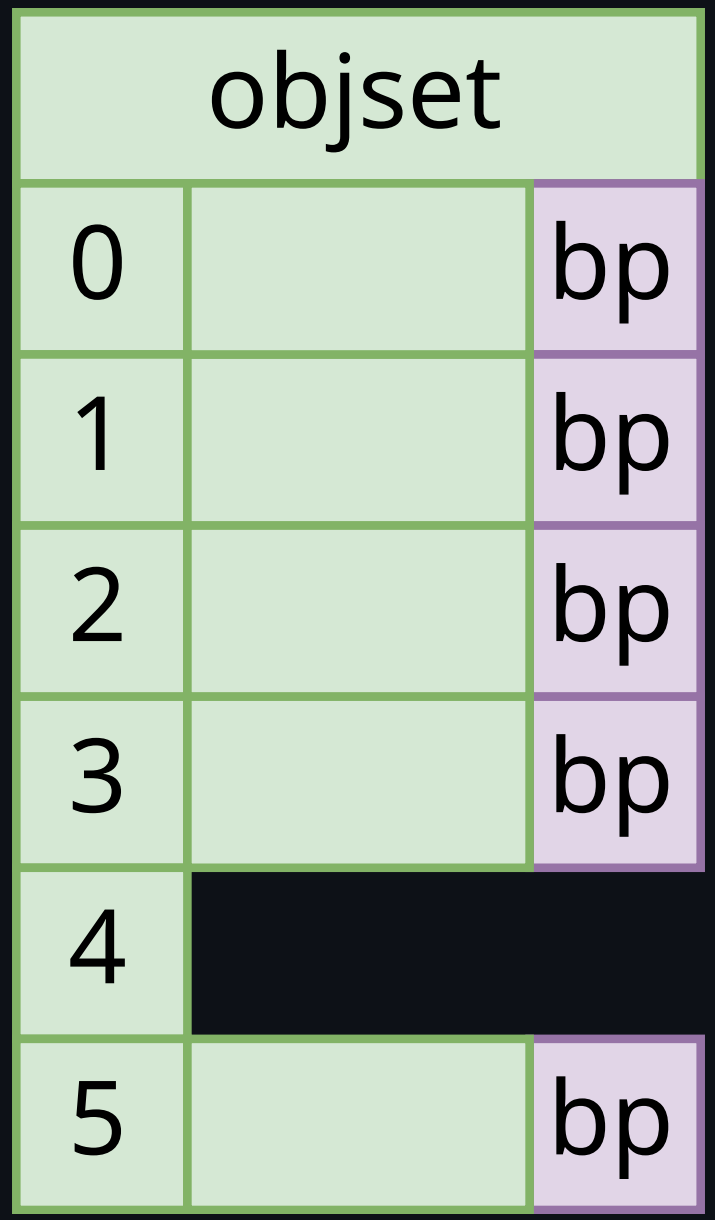


OpenZFS

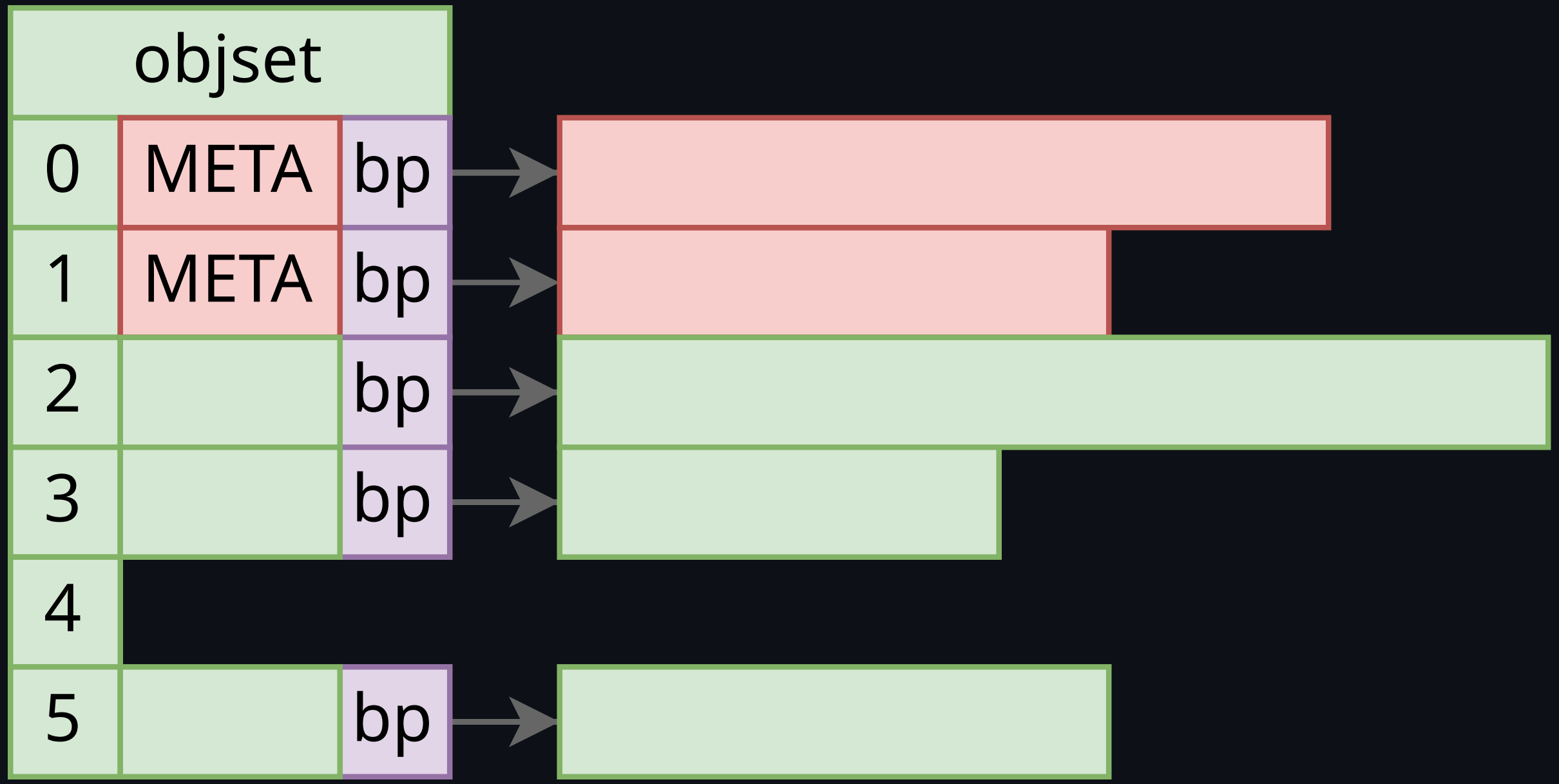
objset


objset
0
1
2
3
4
5

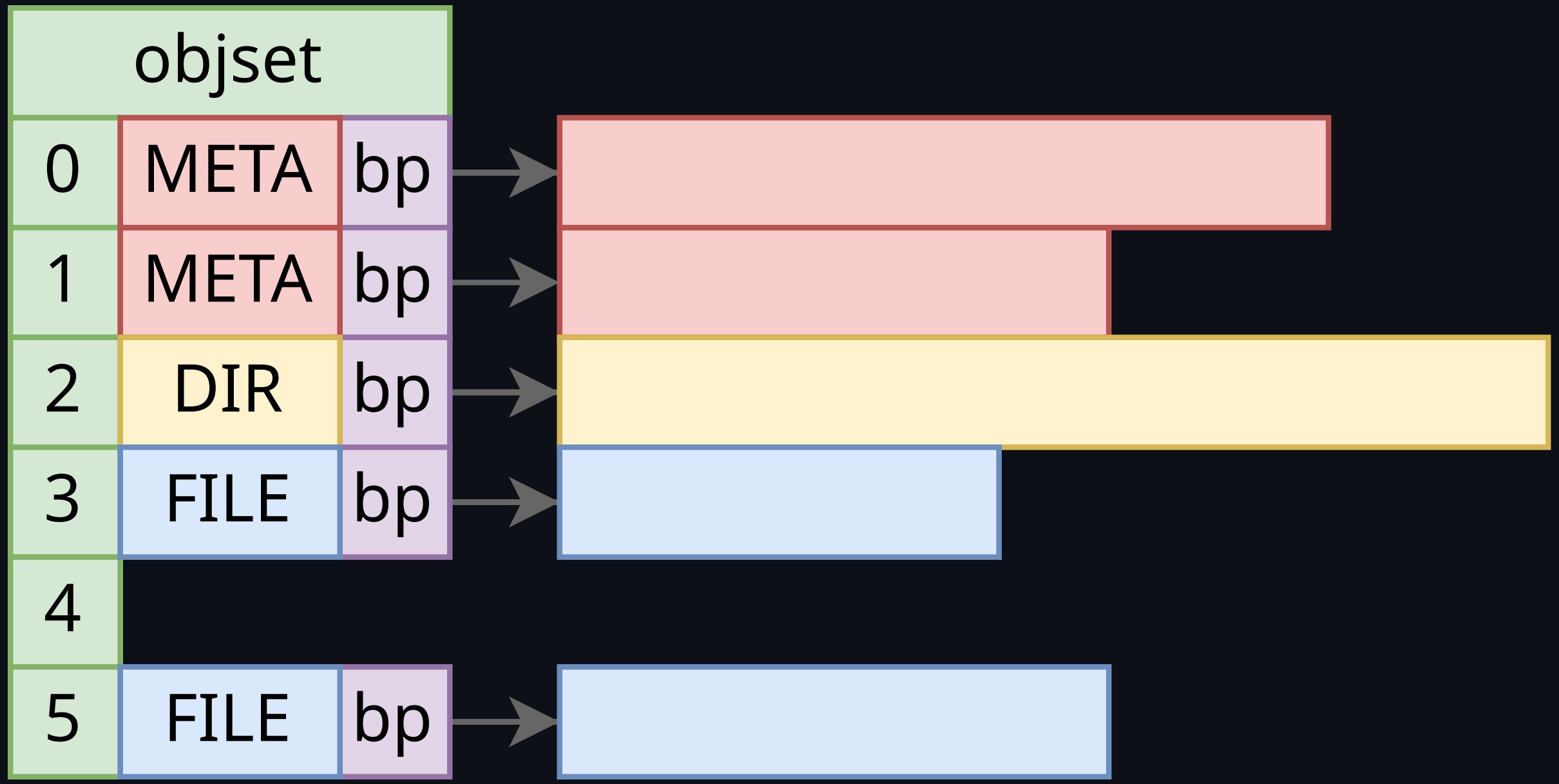

objset
0
1
2
3
4
5

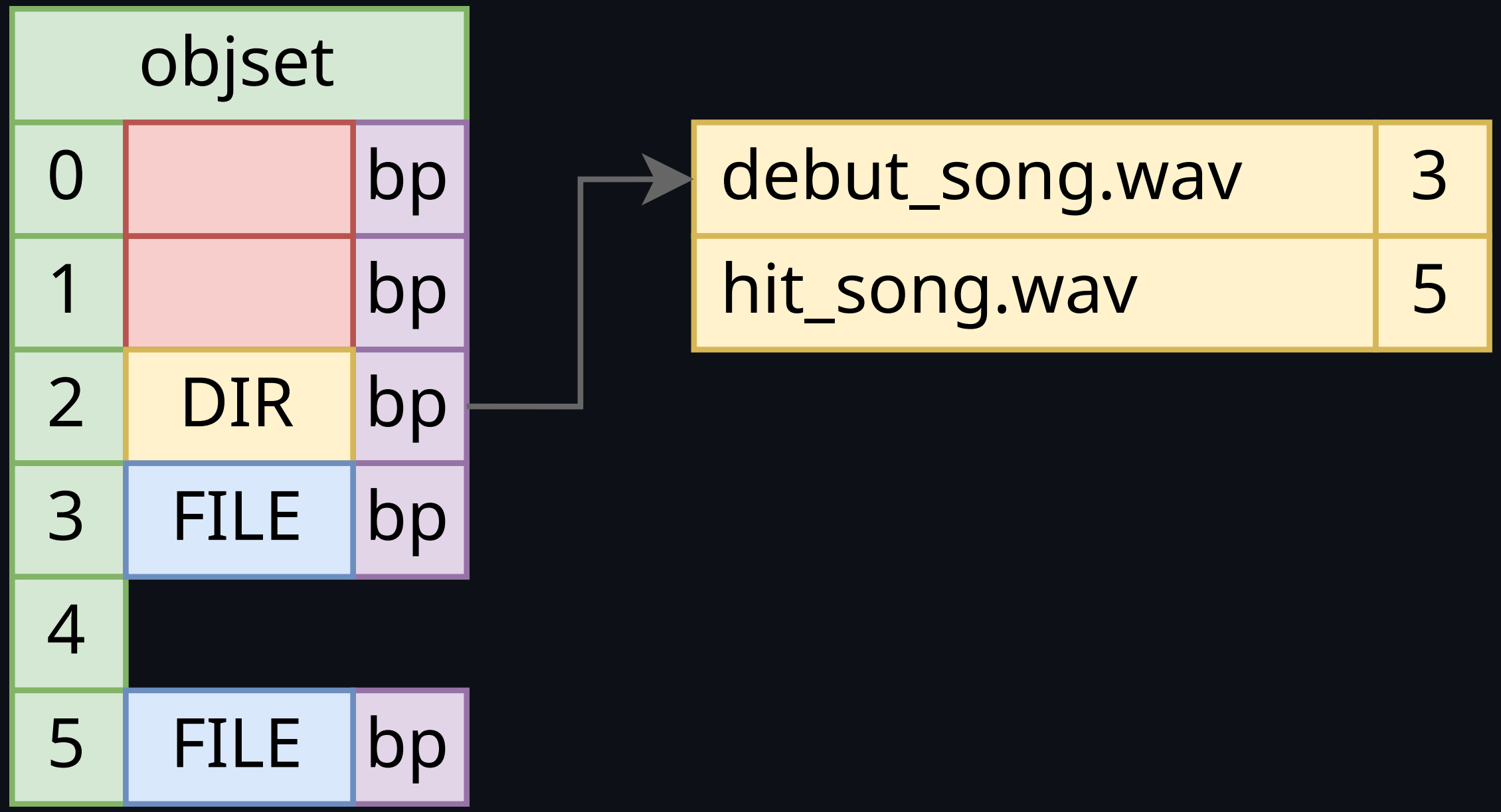

objset
0
bp
1
bp
2
bp
3
bp
4
5
bp


objset
0
bp
1
bp
2
bp
3
bp
4
5
bp


objset
0
META
bp
1
META
bp
2
bp
3
bp
4
5
bp

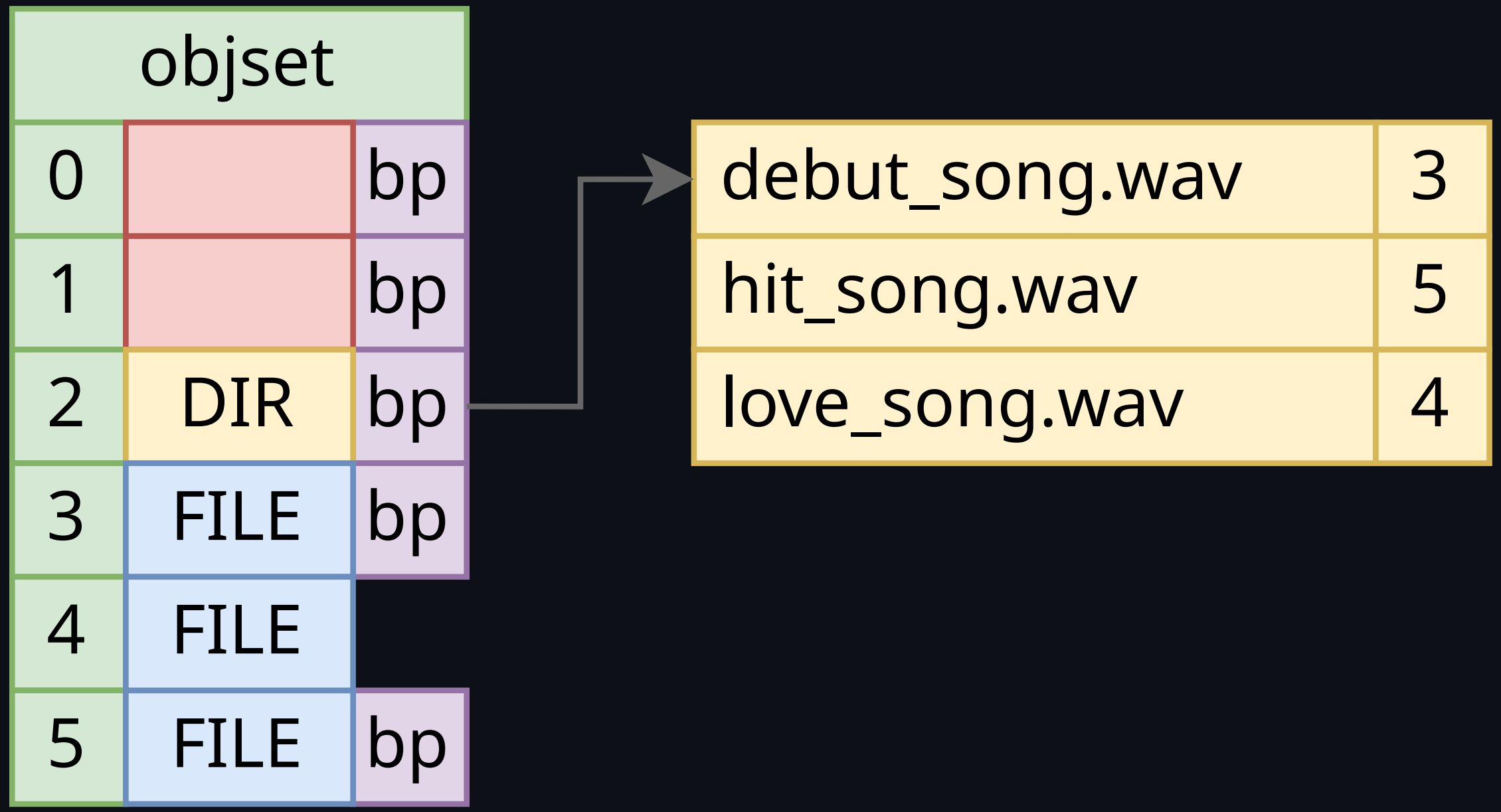

objset
0
META
bp
1
META
bp
2
bp
3
FILE
bp
4
5
FILE
bp

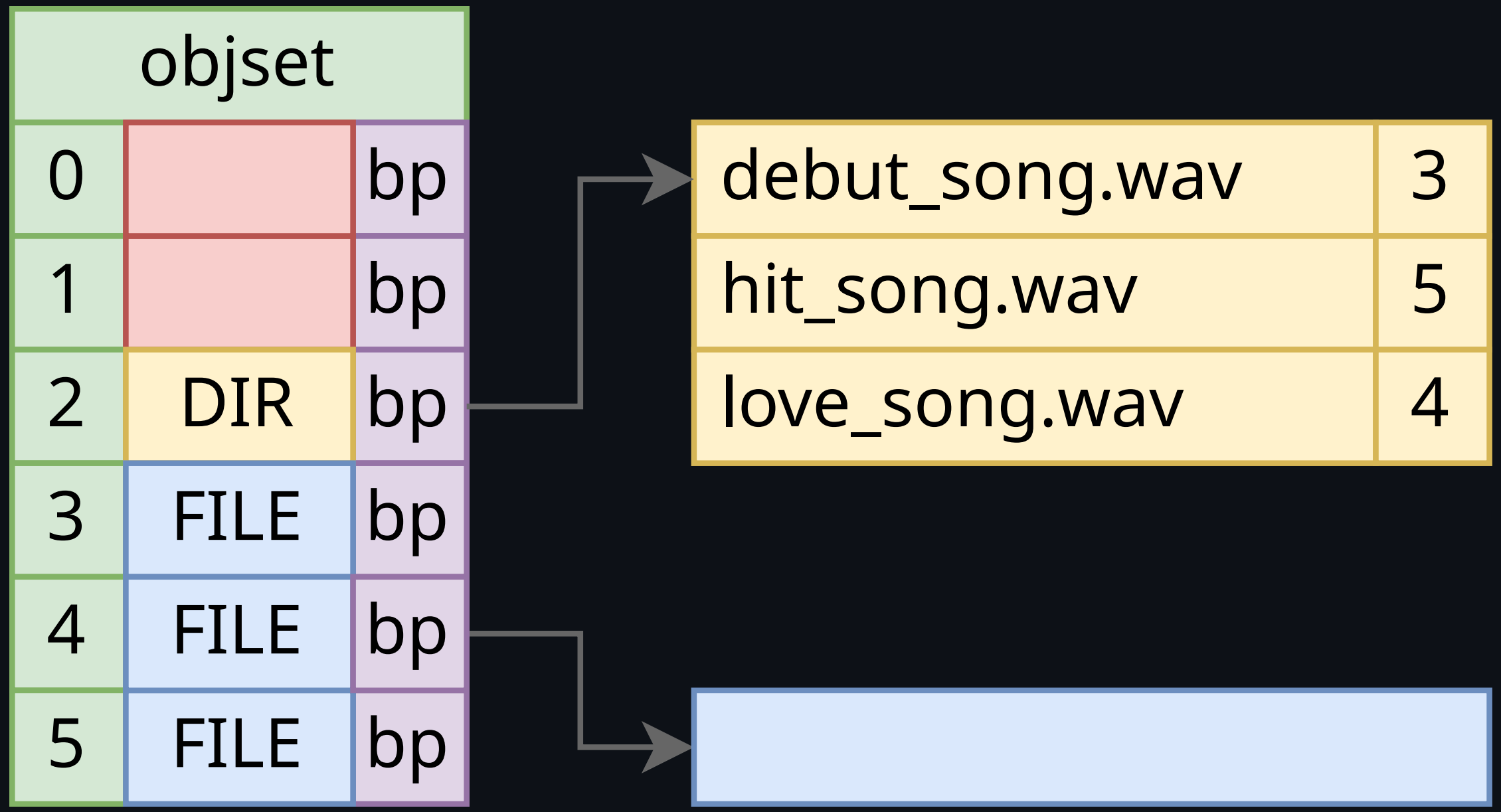

objset
0
META
bp
1
META
bp
2
DIR
bp
3
FILE
bp
4
5
FILE
bp

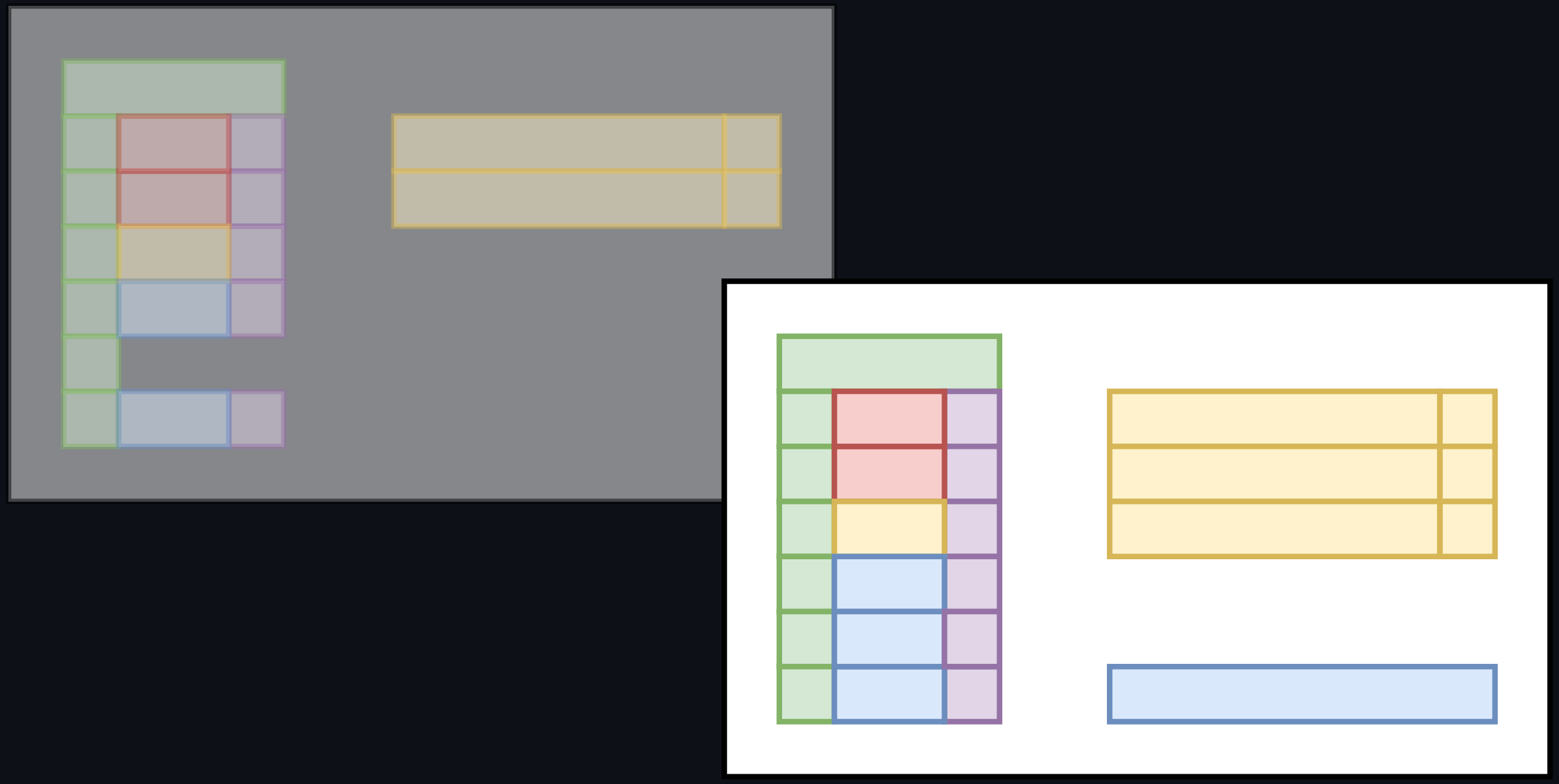

objset
0
bp
1
bp
2
DIR
bp
3
FILE
bp
4
5
FILE
bp
debut_song.wav
3
hit_song.wav
5

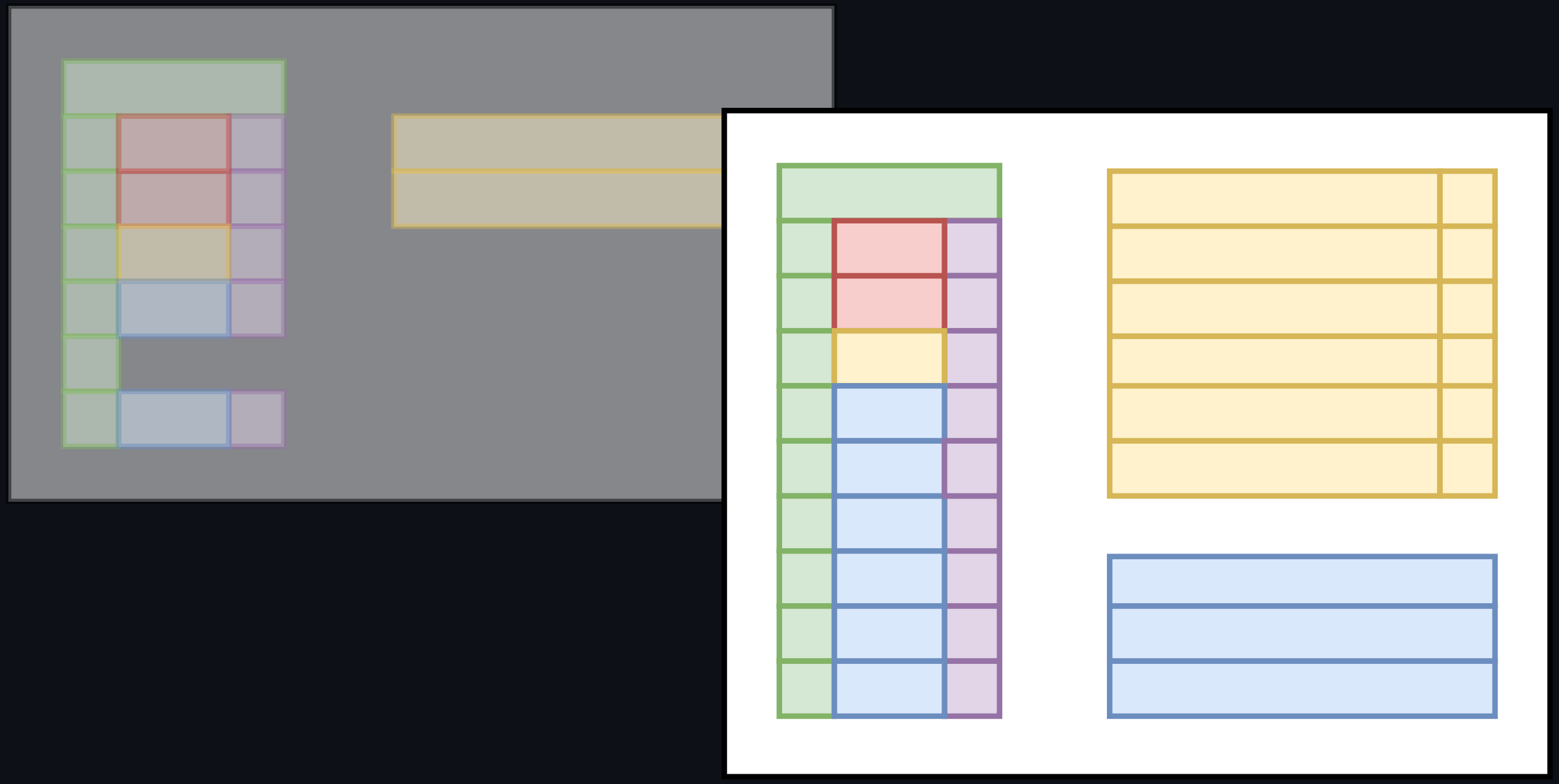
/usr/bin/boyband

```
$ boyband love_song.wav
wrote 'love_song.wav' in 10 seconds
```


objset
0
bp
1
bp
2
DIR
bp
3
FILE
bp
4
FILE
5
FILE
bp
debut_song.wav
3
hit_song.wav
5
love_song.wav
4


objset
0
bp
1
bp
2
DIR
bp
3
FILE
bp
4
FILE
bp
5
FILE
bp
debut_song.wav
3
hit_song.wav
5
love_song.wav
4





fsync()

📜 Royal decree

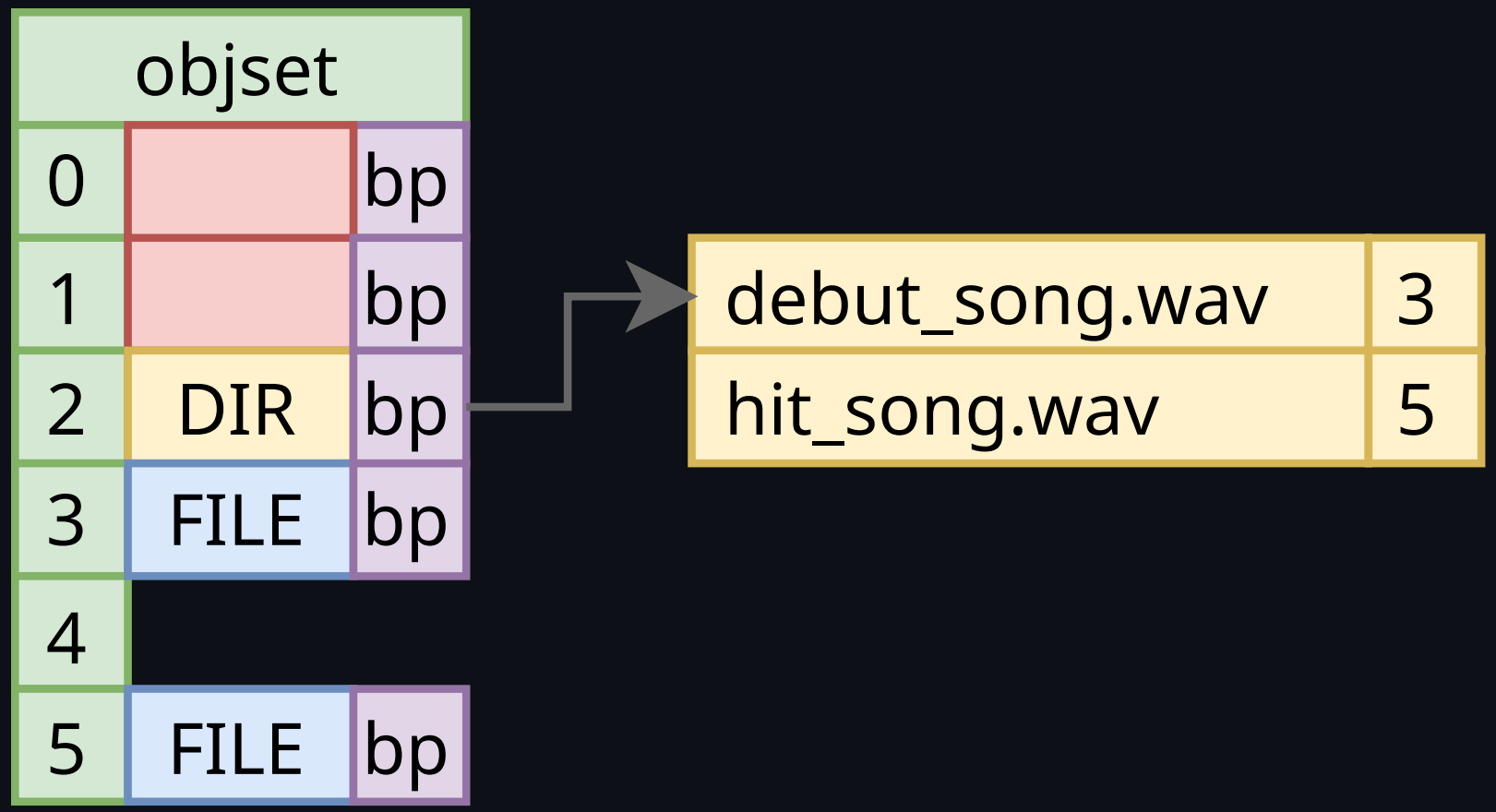
POSIX (IEEE Std 1003.1-2017):

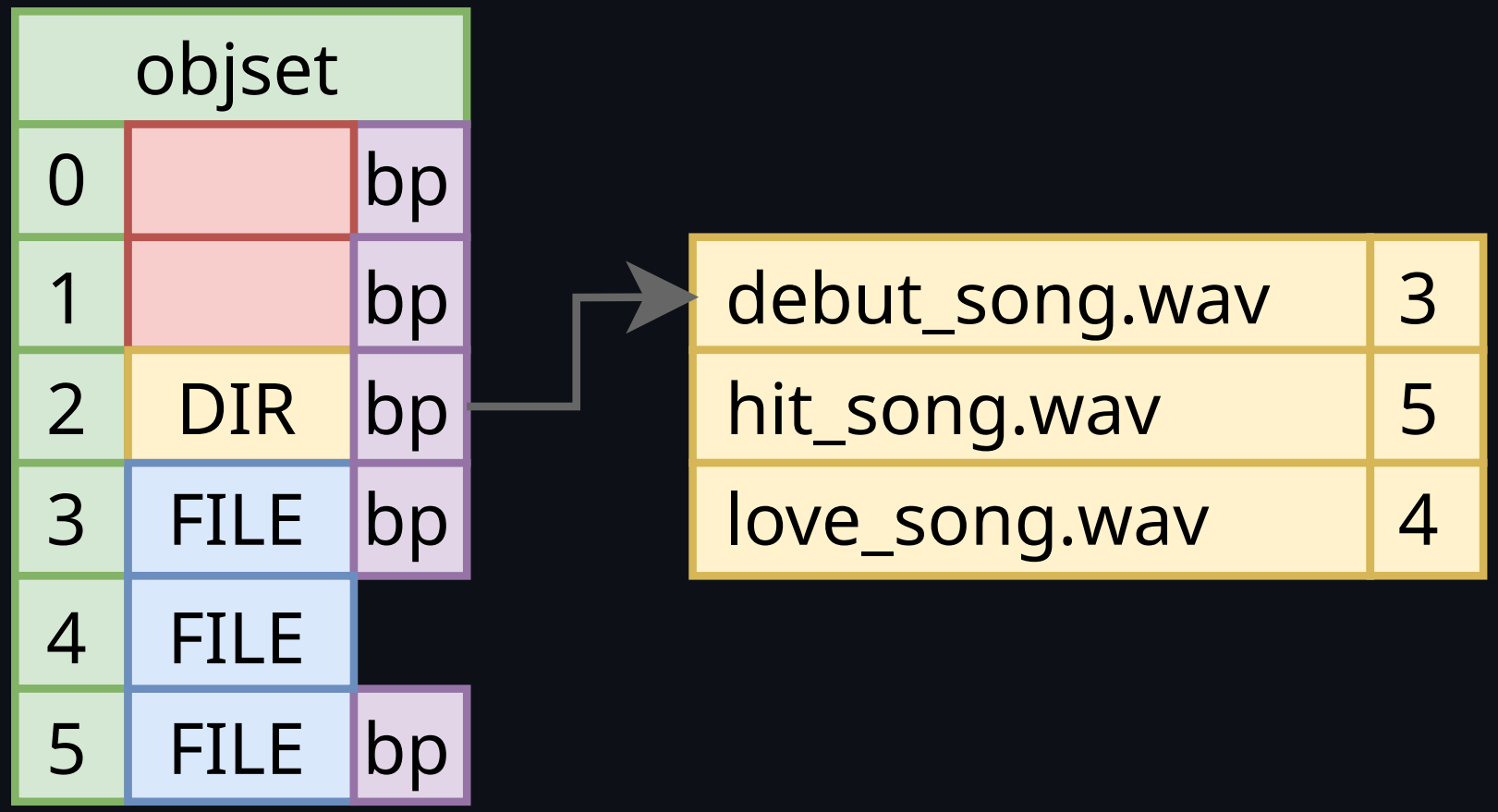
> The *fsync*() function shall request that all data for the open file descriptor named by *fildes* is to be transferred to the storage device associated with the file described by *fildes*. The nature of the transfer is implementation-defined. The *fsync*() function shall not return until the system has completed that action or until an error is detected.

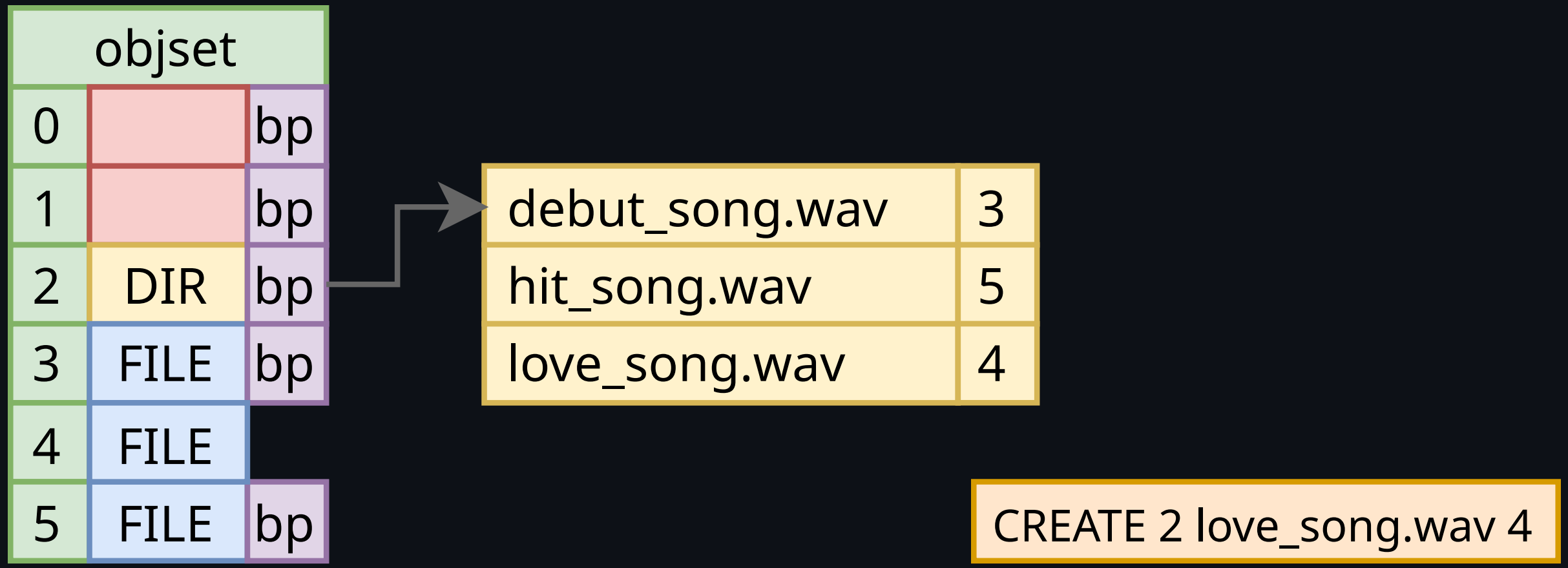
IMPLEMENTATION DEFINED

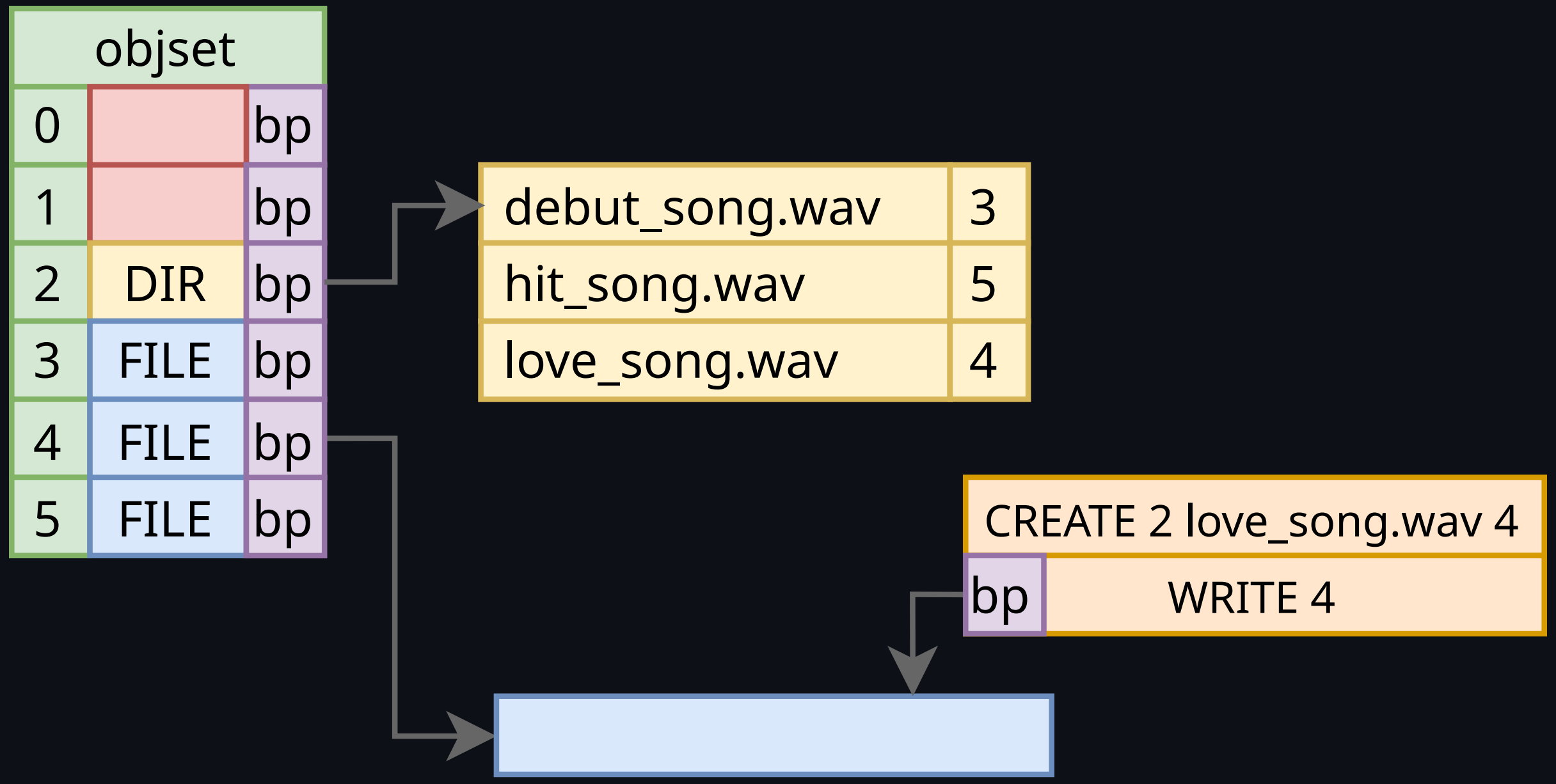

objset
0
bp
1
bp
2
bp
3
bp
4
5
bp

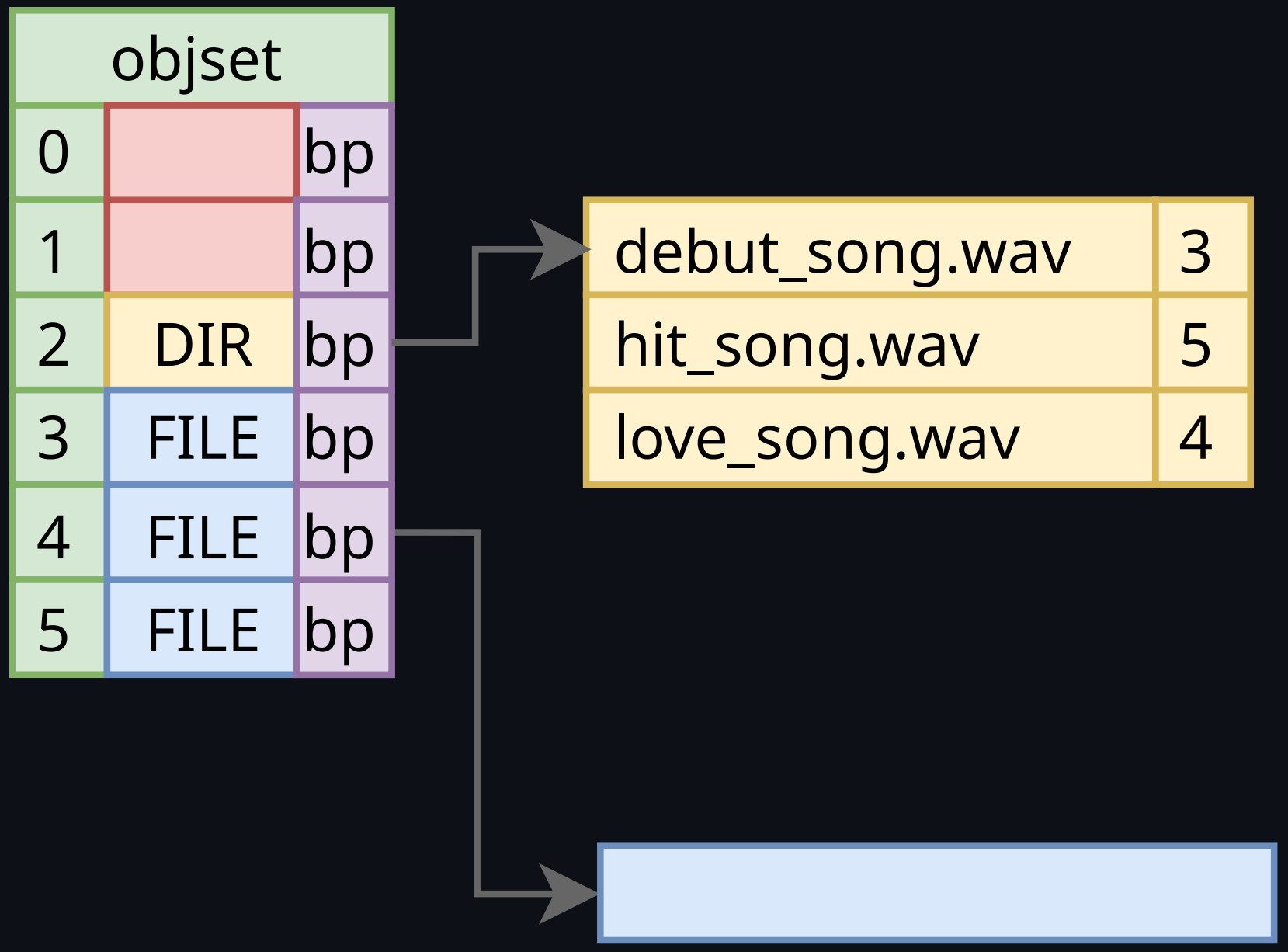

objset
bp
0
bp
1
bp
2
bp
3
bp
4
5
bp

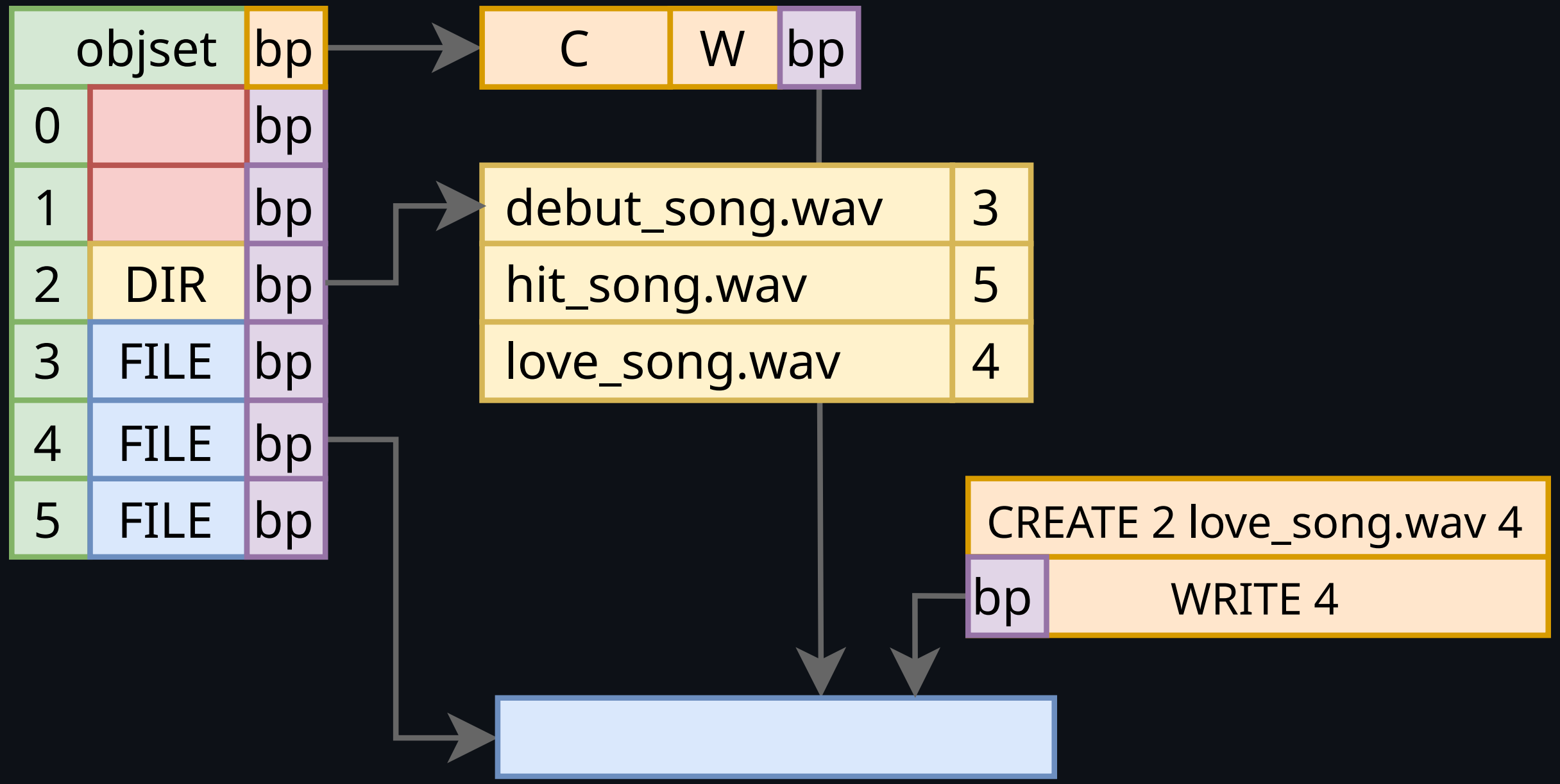

objset
0
bp
1
bp
2
DIR
bp
3
FILE
bp
4
5
FILE
bp
debut_song.wav
3
hit_song.wav
5

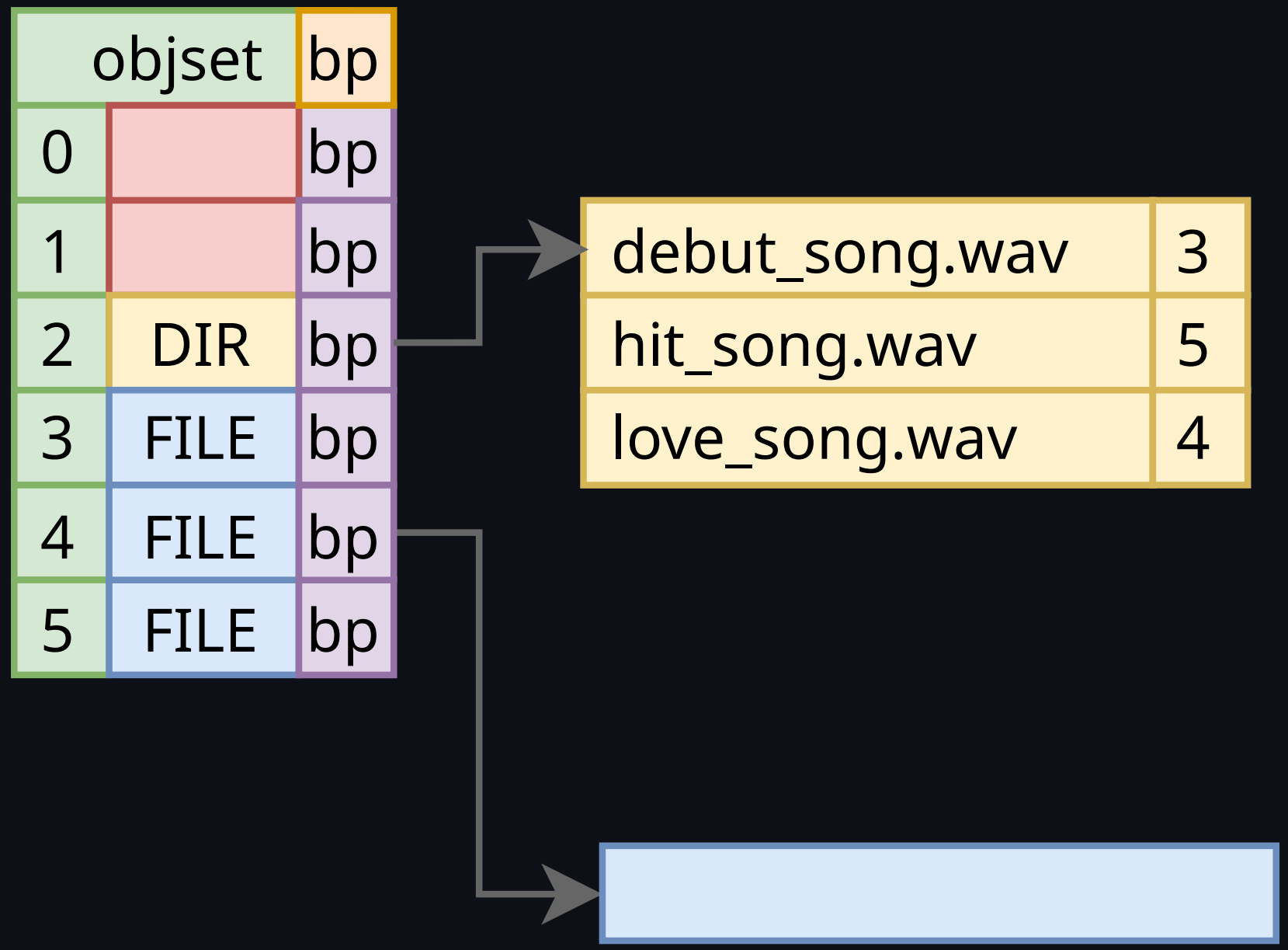

objset
0
bp
1
bp
2
DIR
bp
3
FILE
bp
4
FILE
5
FILE
bp
debut_song.wav
3
hit_song.wav
5
love_song.wav
4

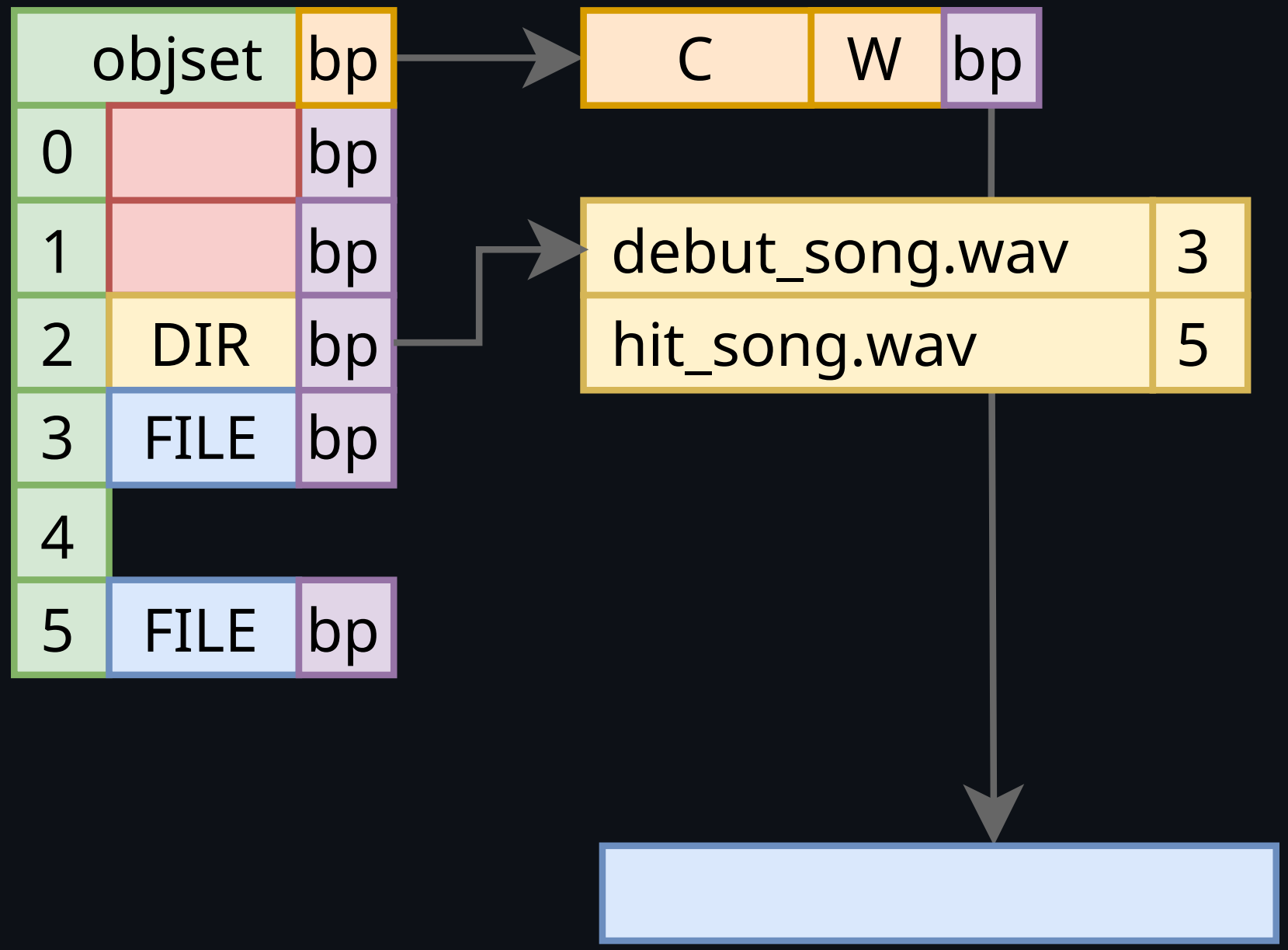

objset
0
bp
1
bp
2
DIR
bp
3
FILE
bp
4
FILE
5
FILE
bp
debut_song.wav
3
hit_song.wav
5
love_song.wav
4
CREATE 2 love_song.wav 4

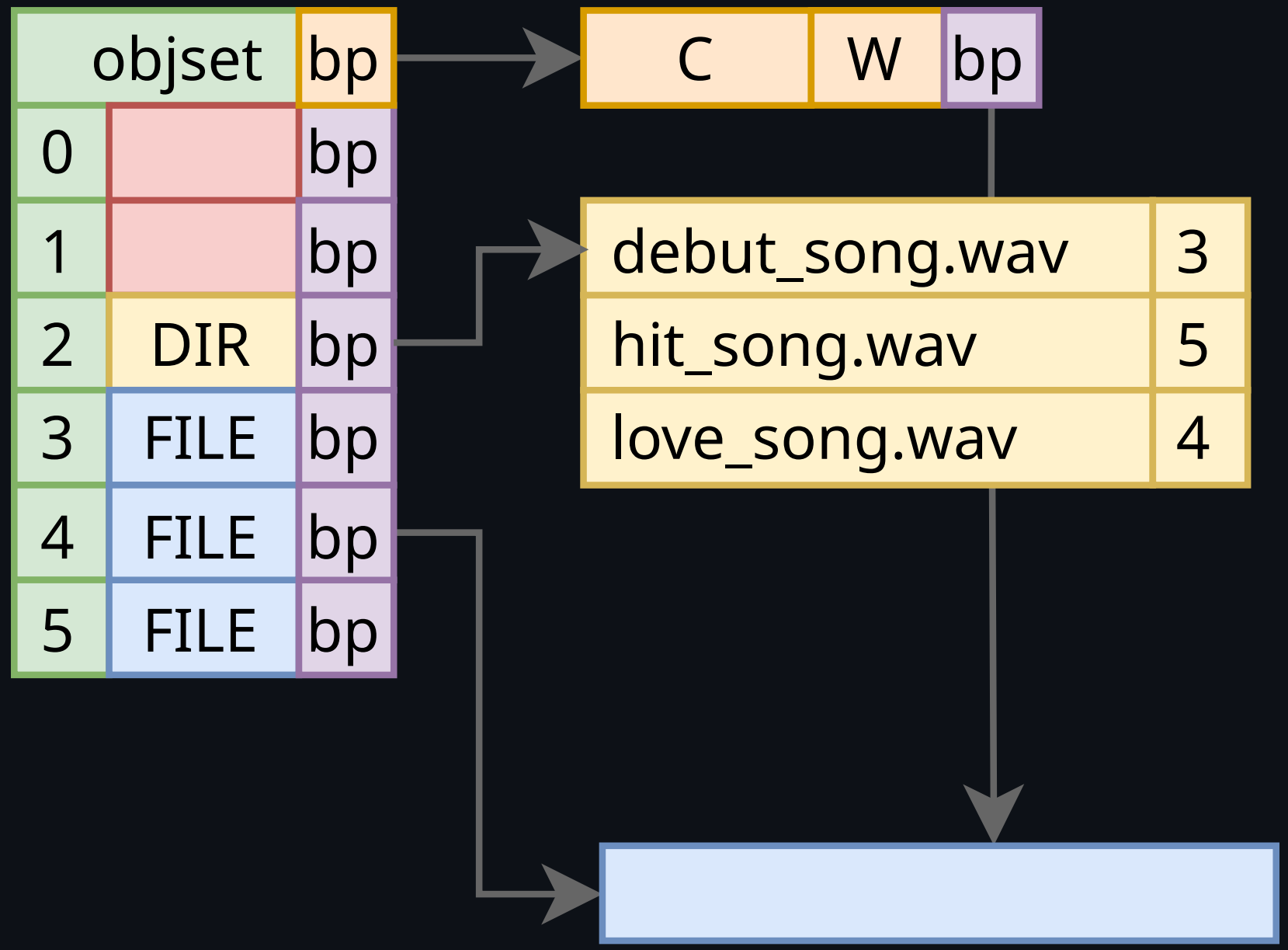

objset
0
bp
1
bp
2
DIR
bp
3
FILE
bp
4
FILE
bp
5
FILE
bp
debut_song.wav
3
hit_song.wav
5
love_song.wav
4
CREATE 2 love_song.wav 4
bp
WRITE 4

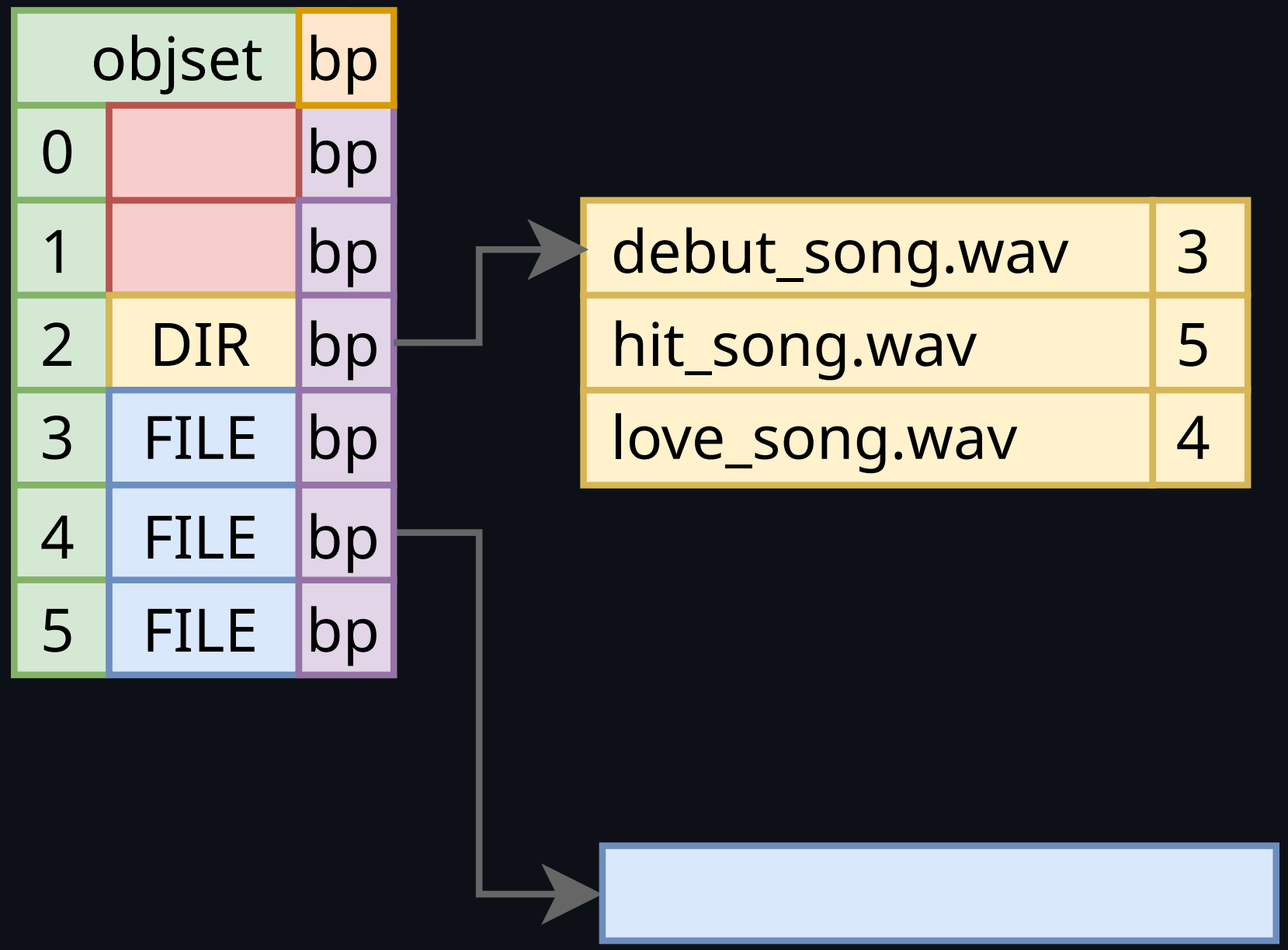

objset
0
bp
1
bp
2
DIR
bp
3
FILE
bp
4
FILE
bp
5
FILE
bp
debut_song.wav
3
hit_song.wav
5
love_song.wav
4


objset
bp
0
bp
1
bp
2
DIR
bp
3
FILE
bp
4
FILE
bp
5
FILE
bp
C
W
bp
debut_song.wav
3
hit_song.wav
5
love_song.wav
4
CREATE 2 love_song.wav 4
bp
WRITE 4

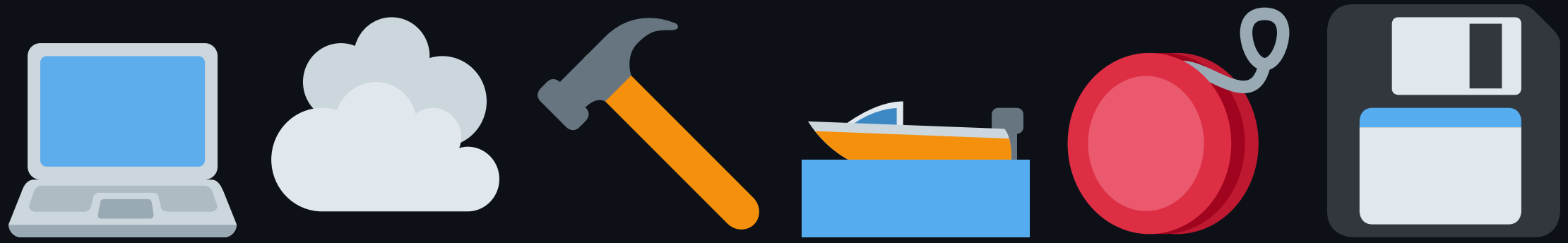

objset
bp
0
bp
1
bp
2
DIR
bp
3
FILE
bp
4
FILE
bp
5
FILE
bp
debut_song.wav
3
hit_song.wav
5
love_song.wav
4

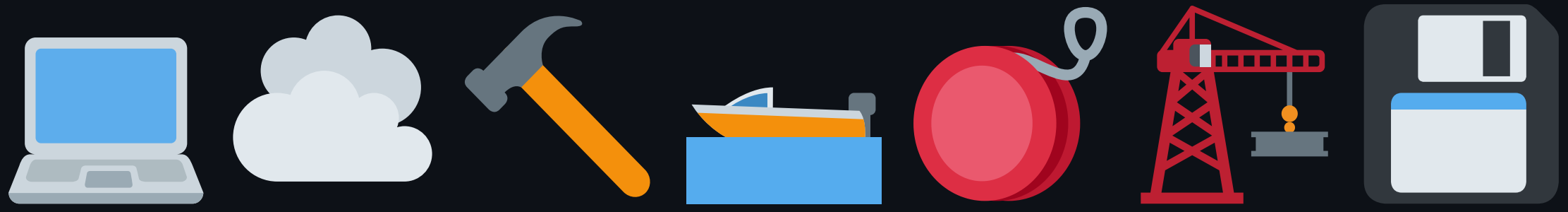

objset
bp
0
bp
1
bp
2
DIR
bp
3
FILE
bp
4
5
FILE
bp
C
W
bp
debut_song.wav
3
hit_song.wav
5

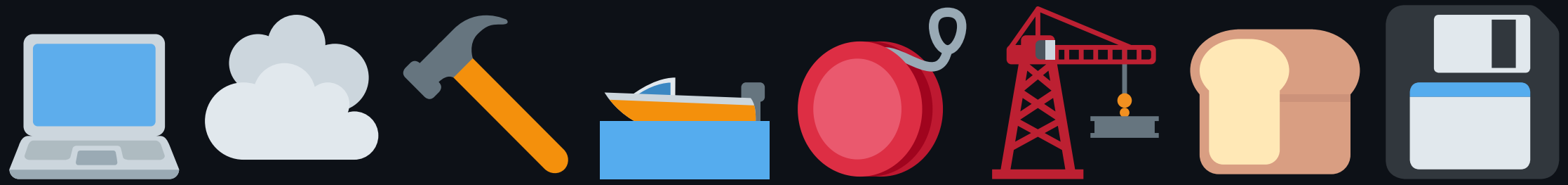

objset
bp
0
bp
1
bp
2
DIR
bp
3
FILE
bp
4
FILE
bp
5
FILE
bp
C
W
bp
debut_song.wav
3
hit_song.wav
5
love_song.wav
4


objset
bp
0
bp
1
bp
2
DIR
bp
3
FILE
bp
4
FILE
bp
5
FILE
bp
debut_song.wav
3
hit_song.wav
5
love_song.wav
4







🐢 Caches all the way down

🐢 Caches all the way down

POSIX

```
write(fd)
fsync(fd)
```

🐢 Caches all the way down

POSIX

```
write(fd)
fsync(fd)
```

SCSI

```
WRITE
SYNCHRONIZE CACHE
```

🐢 Caches all the way down

POSIX

```
write(fd)
fsync(fd)
```

SCSI

```
WRITE
SYNCHRONIZE CACHE
```

NVMe

```
Write
Flush
```

💾 OpenZFS transaction commit

- Wait for all the writes to complete
- Flush all devices

- Write the "even" labels
- Flush all devices

- Write the uberblocks
- Flush all devices

- Write the "odd" labels
- Flush all devices

⚠️ Error handling: writes

- OpenZFS subsystems submit write IO requests
- Grouped together, form the "transaction"
- One fails, all fail

- Writes issued to the disk

⚠️ Error handling: writes

If a write fails:

- Set the IO request aside
- Send a "probe" (label read+write cycle)
- If the probe fails, the write IO request is failed
  - (Invoking redundancy/self-healing behaviours)
- If the probe succeeds, the original IO is retried
- If the IO fails a second time, *the pool is suspended*
  - All outstanding IO held (blocked) until the pool unsuspends
  - Then retried

⚠️ Error handling: flush

⚠️ Error handling: flush

⚠️ Error handling: flush

⚠️ Error handling: flush


I BELIEVED THE FALSEHOODS YOU TOLD ME
BECAUSE TRUST!
NATHANWPYLE

🚽 Flushed away

`zio_flush()`

```
zio_ioctl(pio, vd->vdev_spa, vd, DKIOCFLUSHWRITECACHE, NULL, NULL,
    ZIO_FLAG_DONT_RETRY | ZIO_FLAG_CANFAIL | ZIO_FLAG_DONT_PROPAGATE));
```

- `DONT_RETRY`: if this operation fails, don't bother trying again
- `CANFAIL`: if this operation fails, don’t suspend the pool
- `DONT_PROPAGATE`: if this operation fails, don’t tell me about it

fsync()

💔 Commitment issues

💔 Commitment issues

```
int fsync(int fd) {
    zil_commit(fd);
    return (0);
}
```

💔 Commitment issues

```
int fsync(int fd) {
    zil_commit(fd);
    return (0);
}

void zil_commit(int fd) {









}
```

💔 Commitment issues

```
int fsync(int fd) {
    zil_commit(fd);
    return (0);
}

void zil_commit(int fd) {
    zio_t *write_zio = zil_make_zio_for_fd(fd, ZIO_FLAG_CANFAIL);









}
```

💔 Commitment issues

```
int fsync(int fd) {
    zil_commit(fd);
    return (0);
}

void zil_commit(int fd) {
    zio_t *write_zio = zil_make_zio_for_fd(fd, ZIO_FLAG_CANFAIL);
    int write_err = zio_wait(zio);









}
```

💔 Commitment issues

```
int fsync(int fd) {
    zil_commit(fd);
    return (0);
}

void zil_commit(int fd) {
    zio_t *write_zio = zil_make_zio_for_fd(fd, ZIO_FLAG_CANFAIL);
    int write_err = zio_wait(zio);

    zio_t *flush_zio = zil_root();
    zio_flush(flush_zio);
    int flush_err = zio_wait(flush_zio);




}
```

💔 Commitment issues

```
int fsync(int fd) {
    zil_commit(fd);
    return (0);
}

void zil_commit(int fd) {
    zio_t *write_zio = zil_make_zio_for_fd(fd, ZIO_FLAG_CANFAIL);
    int write_err = zio_wait(zio);

    zio_t *flush_zio = zil_root();
    zio_flush(flush_zio);
    int flush_err = zio_wait(flush_zio);

    if (write_err < 0 || flush_err < 0)
        txg_wait_synced(...);
}
```

- `/usr/bin/boyband`: writes data, calls `fsync()`
- `zil_commit()`: issues write IO, succeeds

- `/usr/bin/boyband`: writes data, calls `fsync()`
- `zil_commit()`: issues write IO, succeeds
- ⚡Array loses power
- `zil_commit()`: issues flush IO, "succeeds"
- 🚨 **`zil_commit()` returns, `fsync()` returns success** 🚨
- Transaction commit begins, writes issued
- 🛑 Write failed, IO held, pool suspended 🛑

- ⚡Power restored
- `zpool clear`
- IO reissued
- Transaction completes

❌❌💾💾💾💾💾💾

🛠️ ZIL flush error propagation

```
vdev_t *vd = vdev_lookup_top(spa, zv->zv_vdev);
if (vd != NULL) {
        /*
         * The "ZIO_FLAG_DONT_PROPAGATE" is currently
         * always used within "zio_flush". This means,
         * any errors when flushing the vdev(s), will
         * (unfortunately) not be handled correctly,
         * since these "zio_flush" errors will not be
         * propagated up to "zil_lwb_flush_vdevs_done".
         */
        zio_flush(lwb->lwb_root_zio, vd);
}
```

🛠️ ZIL flush error propagation

```diff
diff --git module/zfs/zio.c module/zfs/zio.c
index 213fe5c48..002f117df 100644
--- module/zfs/zio.c
+++ module/zfs/zio.c
@@ -1633,7 +1633,7 @@ zio_flush(zio_t *pio, vdev_t *vd)
        if (vd->vdev_children == 0) {
                zio_nowait(zio_ioctl(pio, vd->vdev_spa, vd,
                    DKIOCFLUSHWRITECACHE, NULL, NULL, ZIO_FLAG_CANFAIL |
-                   ZIO_FLAG_DONT_PROPAGATE | ZIO_FLAG_DONT_RETRY));
+                   ZIO_FLAG_DONT_RETRY));
        } else {
                for (uint64_t c = 0; c < vd->vdev_children; c++)
                        zio_flush(pio, vd->vdev_child[c]);
```


[Everyone liked that]

```
void
zio_flush(zio_t *pio, vdev_t *vd)
{
        if (vd->vdev_children == 0) {
                zio_nowait(zio_ioctl(pio, vd->vdev_spa, vd,
                    DKIOCFLUSHWRITECACHE, NULL, NULL, ZIO_FLAG_CANFAIL |
                    ZIO_FLAG_DONT_RETRY));
        } else {
                for (uint64_t c = 0; c < vd->vdev_children; c++)
                        zio_flush(pio, vd->vdev_child[c]);
        }
}
```

🚽 The right amount of flushing 👍

POSIX

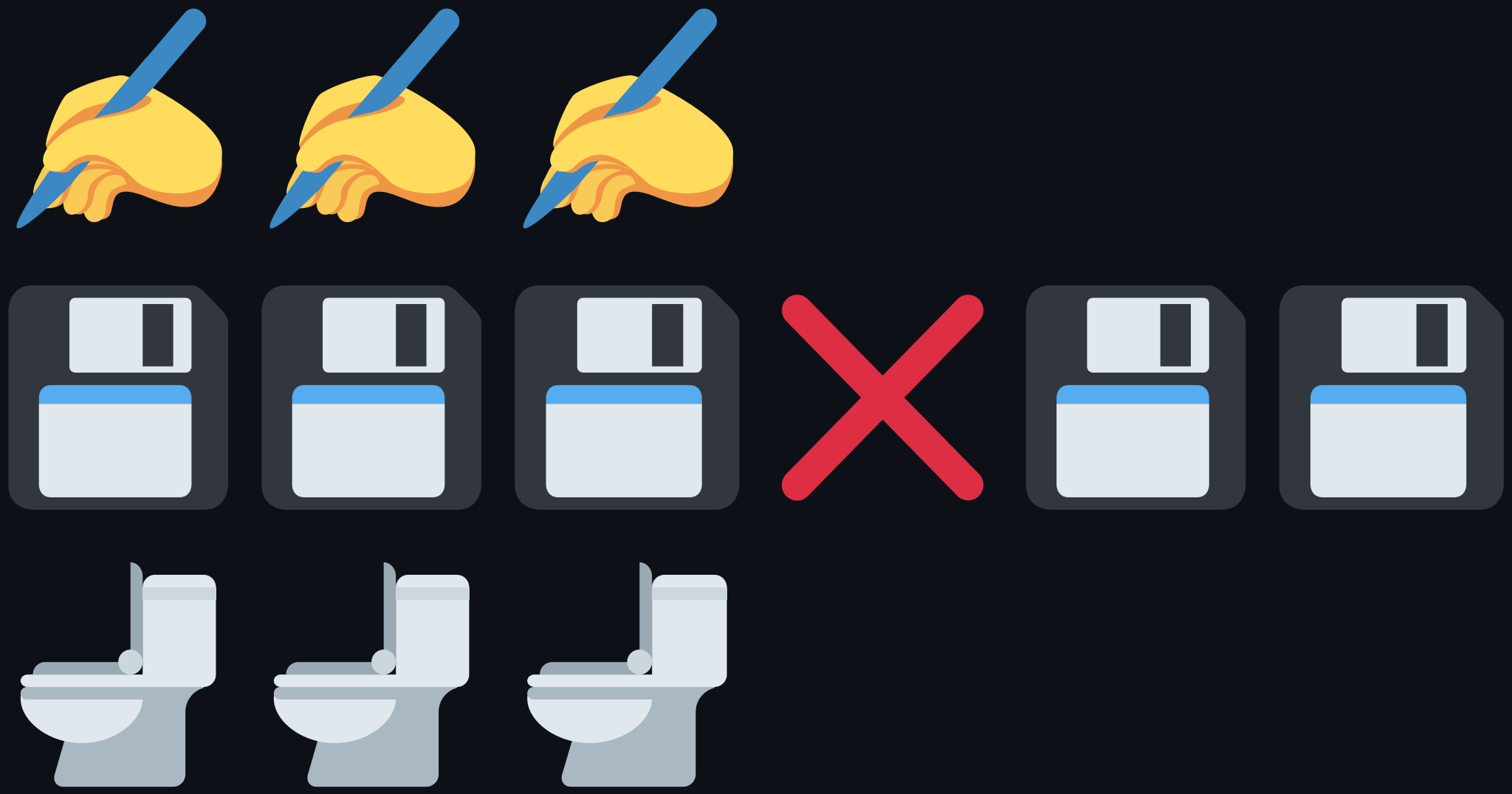
```
write(fd)
fsync(fd)
```

SCSI

```
WRITE
SYNCHRONIZE CACHE
```

NVMe

```
Write
Flush
```



🔍 vdev tracing

- `ZIO_FLAG_VDEV_TRACE`
- `zio->io_vdev_trace_tree`
- `zio_flush_traced(flush_zio, write_zio)`

Status report

- In production at a customer site
- Upstreaming has started:
  - Test cases added to demonstrate the fault
  - Additional test tools needed (GH#15953)
  - Patches ready to go once test suite can support it

🚀 Future work

- Extend to transaction commit flushes
- IO replay after suspend
- `fsync()` error return


SYNC