State Database Replicas

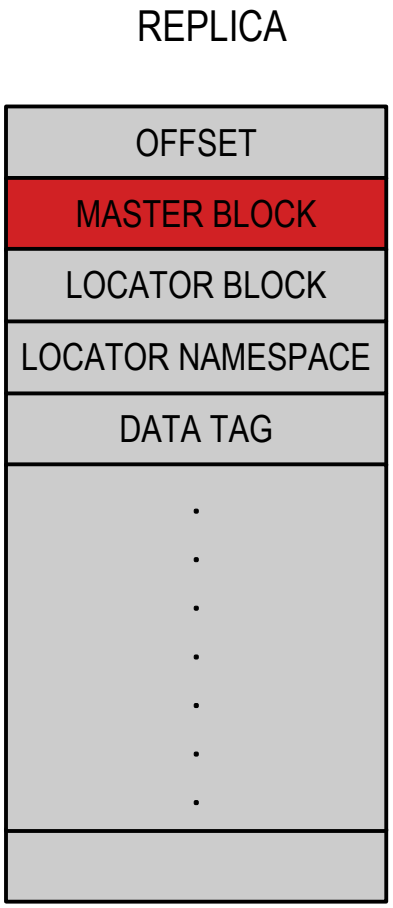
What is a replica?

- State Database Replica – A database that contains information on svm metadevices and configuration.
  - Aka: metadb, replica, mddb.
- Metadbs track information about the locations and states of all SVM metadevices.
- Replicas typically exist in two places: on disk and in-core(in the kernel).
- On-disk and in-core replicas
  - The in-core metadb is a composition of all on-disk replicas for local set or diskset

Replica Types

- Local Metadb
  - Minimum requirement to run SVM is one local metadb.
  - All metadevices that are not in a diskset are tracked in the local mddb, including root mirror.
- Diskset Metadb (traditional, shared)
  - Replicas that only contain information about metadevices in the diskset
- Multi-owner Diskset Metadb
  - Replicas that contain special fields for running on multiple nodes.

Master Block



- Devid
  - Allows SVM to import replicated disksets
- Mapping for mddb
  - Contains offset for locator block. Replica copies will have different content in their masterblocks even if rest of replica is identical
  - Can have multiple copies of an mddb on same slice. Provides logical to physical block translations in replica
- Master Block is 512 bytes(1 block)
  - On-disk the offset for the first master block on a slice is 16 blocks

Master Block Datastructures

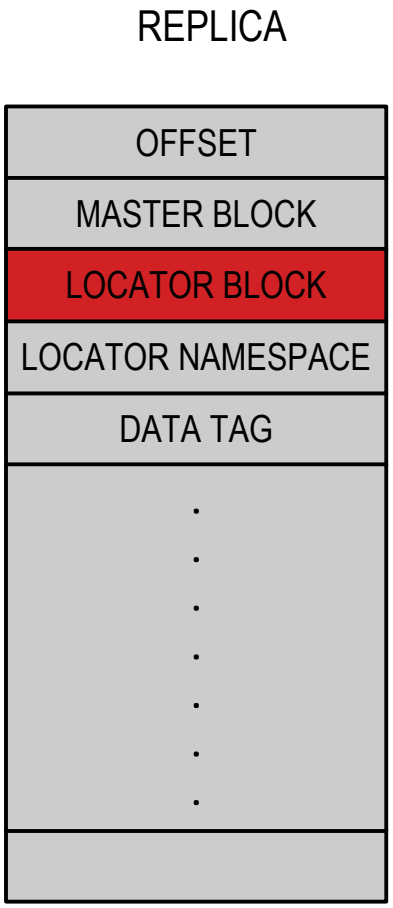
On-Disk Master Block

```
typedef struct mddb_mb {
        int                     mb_magic;       /* used for verification */
        uint_t          mb_revision;    /* used for verification */
        uint_t          mb_checksum;    /* used for verification */
#ifdef _LP64
        uint32_t        mb_next;        /* incore to next mb */
#else
        struct mddb_mb*mb_next;         /* incore to next mb */
#endif  /* _LP64 */
        daddr32_t       mb_nextblk;     /* block # for next mb */
        md_timeval32_t mb_timestamp;    /* timestamp */
        daddr32_t       mb_blkcnt;      /* size of blkmap */
        daddr32_t       mb_blkno;       /* physical loc. for this MB */
        set_t           mb_setno;       /* used for verification */
        struct timeval32 mb_setcreatetime; /* set creation timestamp */
        int                     spares[7];
        mddb_map_t      mb_blkmap;      /* logical->physical blk map */
        int                     mb_devid_magic; /* verify devid in mb */
        short           mb_devid_len;   /* len of following devid */
        char            mb_devid[1];    /* devid byte array */
} mddb_mb_t;
```

In-Core Master Block

```
typedef struct mddb_mb_ic {
        struct mddb_mb_ic       *mbi_next;
        struct mddb_mb          mbi_mddb_mb;
} mddb_mb_ic_t;
```

Locator Block



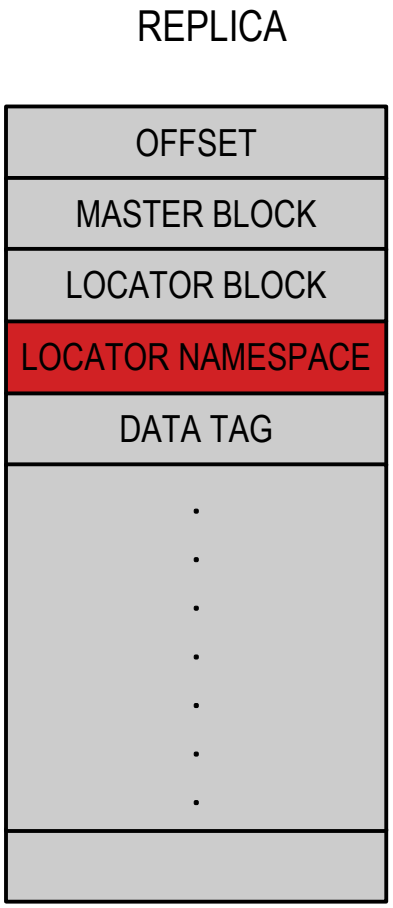
- Contains the creation time of the first replica in set
- Tracks location and state of all replicas in set
- Contains pointer to devids for all disks with replica copies in set
- Locator block has logical offset 0

Locator Block Datastructures

On-Disk and In-Core Locator Block - (local and diskset)

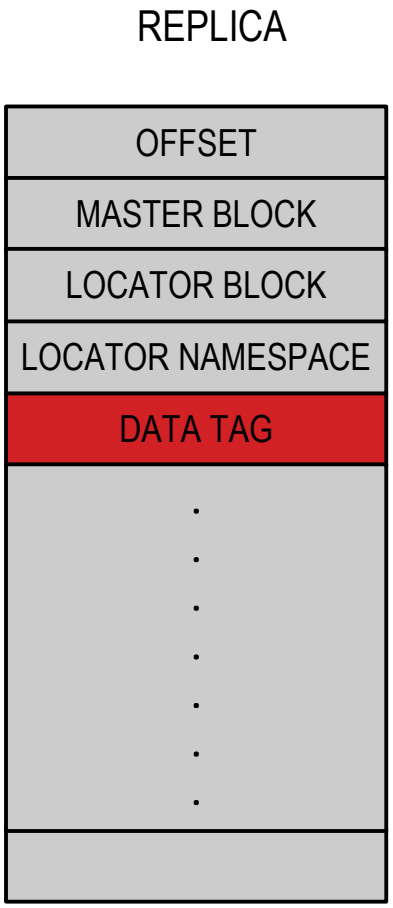
```
typedef struct mddb_lb {
        int                     lb_magic;       /* used for verification */
        uint_t                  lb_revision;    /* used for verification */
        int                     lb_checksum;    /* used for verification */
        uint_t                  lb_commitcnt;   /* IMPORTANT */
        struct timeval32        lb_timestamp;   /* informative only */
           .
        uint_t                  lb_flags;       /* flags describing LB */
        uint_t                  lb_spare[8];    /* Spare/Pad */
        mddb_block_t            lb_didfirstblk; /* Devid Array Start Block */
        mddb_block_t            lb_didblkcnt;   /* Devid Array Number Blocks */
           .
           .
        struct timeval32        lb_inittime;    /* creation of database */
        set_t                   lb_setno;       /* used for verification */
        mddb_block_t            lb_blkcnt;      /* used for verification */
           .
        mddb_drvnm_             lb_drvnm[MDDB_DRVNMCNT];
        mddb_locator_t          lb_locators[MDDB_NLB];
        mddb_sidelocator_t      lb_sidelocators[MD_MAXSIDES][MDDB_NLB];
} mddb_lb_t;
```

Locator Namespace



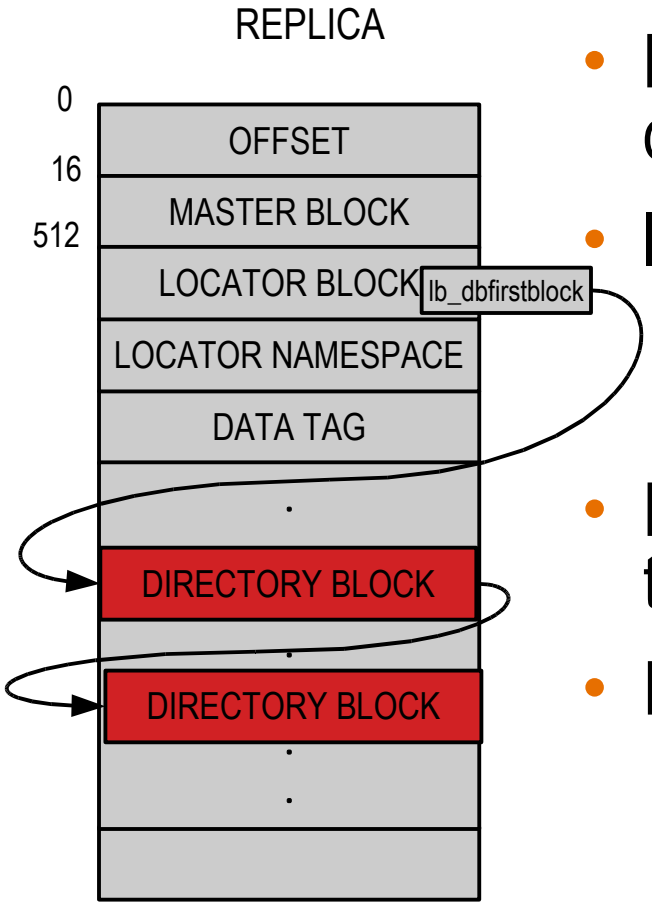
- Pathnames for dev_t in locator block array
- Used for error messaging for all locator block devices

Data Tag



- Set when diskset reaches 50% of replicas.
  - > Allows set to recover after reboot when only have 50% of replicas.
- When kernel chooses the best replica (golden replica) to use for the set, it will only read in the: master block, locator block (including devid array), locator namespace ,and data tag.
- Can also be set for a replica by user, so the kernel will bypass the normal mechanism for choosing best replica and choose one with the data tag set.

Directory Blocks



- Most of the data is located under directory blocks.
- lb_dbfirstblock field in Locator block
  - Pointer to location of first directory block.
- Each directory block contains pointer to next, null if last.
- First directory entry is at:
  - Address of db_firstentry + sizeof(db_firstentry)

Directory Block Datastructures

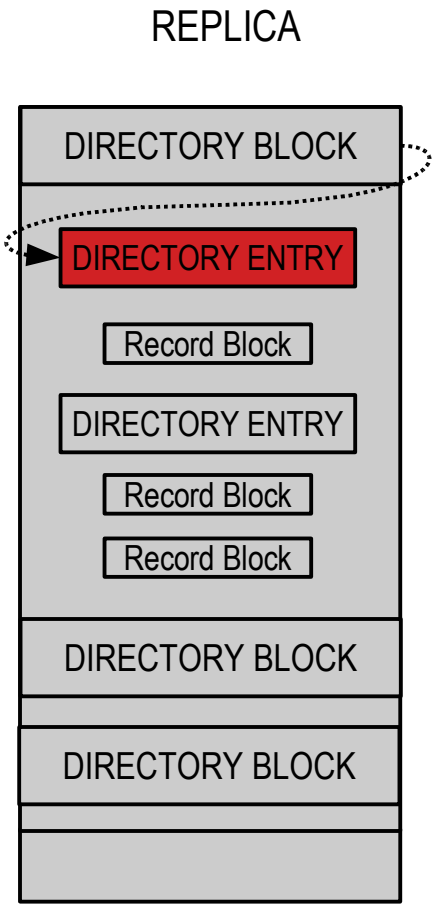
On-Disk Directory Block

```
typedef struct mddb_db32 {
        uint_t              db32_magic;
        uint_t              db32_revision;
        uint_t              db32_checksum;
        mddb_block_t            db32_blknum;
        uint32_t             db32_next;
        mddb_block_t            db32_nextblk;
        struct timeval32       db32_timestamp;
        uint_t              db32_recsum;
        uint32_t             db32_firstentry;
} mddb_db32_t;
```

In-Core Directory Block

```
typedef struct mddb_db {
        uint_t              db_magic;
        uint_t              db_revision;
        uint_t              db_checksum;
        mddb_block_t            db_blknum;
        struct mddb_db         *db_next;
        mddb_block_t            db_nextblk;
        struct timeval32       db_timestamp;
        uint_t              db_recsum;
#ifdef _KERNEL
        mddb_de_ic_t            *db_firstentry;
#else
        mddb_de_t              *db_firstentry;      <--userland
#endif
} mddb_db_t;
```

Directory Entry



- Directory blocks contain a linked list of pointers to directory entries
  - > In-core: Each directory entry points to the next directory entry under that directory block
  - > On-disk: need to calculate DE offsets
- Directory entries contain a pointer to associated record block(s)
- Directory entries have a flag field that can be used to determine what type of record data it contains
  - > Match to entry in mddb_type_t

Directory Entry Datastructures

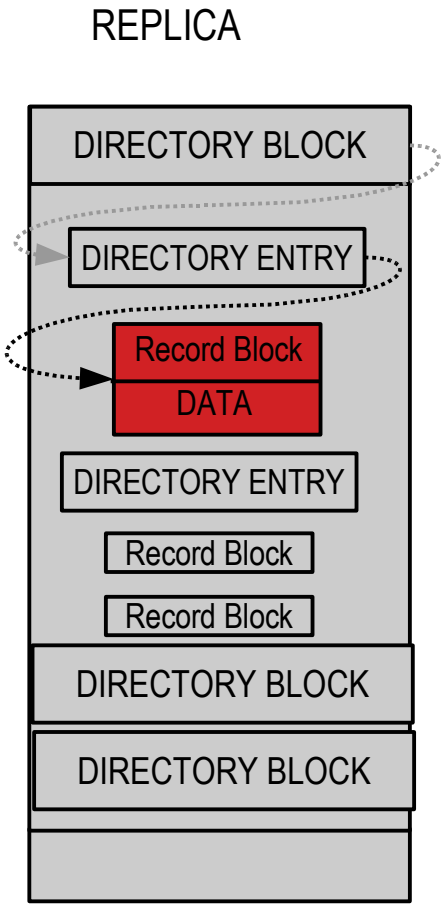
On-Disk Directory Entry

```
typedef struct mddb_de32 {
        uint32_t        de32_next;
        uint32_t        de32_rb;
        mddb_recid_t    de32_recid;
        mddb_type_t     de32_type1;
        uint_t          de32_type2;
        uint_t          de32_reqsize;
        uint_t          de32_recsize;
        mddb_block_t    de32_blkcount;
        uint_t          de32_flags;
        mddb_optinfo_t  de32_optinfo[2];
        mddb_block_t    de32_blks[1];
} mddb_de32_t;
```

In-Core Directory Entry

```
typedef struct mddb_de_ic {
        void                    *de_rb_userdata;
        void                    *de_rb_userdata_ic;
        uint_t                  de_owner_nodeid;
        struct mddb_de_ic       *de_next;
        mddb_rb32_t             *de_rb;
        mddb_recid_t            de_recid;
        mddb_type_t             de_type1;
        uint_t                  de_type2;
        size_t                  de_reqsize;
        size_t                  de_icreqsize;
        size_t                  de_recsize;
        uint_t                  de_blkcount;
        uint_t                  de_flags;
        mddb_optinfo_t          de_optinfo[2];
        mddb_block_t            de_blks[1];
} mddb_de_ic_t;
```

Record Blocks



- Pointer to record block is stored inside of its associated directory entry
- On-disk: the actual data for record starts from the last field(rb_data) of datastructure mddb_rb32_t.
- In-core: rb_userdata points to the actual data(rb_data).

Record Block Datastructures

On-Disk Record Block

```
typedef struct mddb_rb32 {
        uint_t              rb_magic;
        uint_t              rb_revision;
        uint_t              rb_checksum;
        uint_t              rb_checksum_fiddle;
        uint_t              rb_private;
        uint32_t             rb_userdata;
        uint_t              rb_commitcnt;
        uint_t              rb_spare[1];
        struct timeval32       rb_timestamp;
        int                 rb_data[1];
} mddb_rb32_t;
```

In-Core Record Block

```
typedef struct mddb_rb {
        uint_t              rb_magic;
        uint_t              rb_revision;
        uint_t              rb_checksum;
        uint_t              rb_checksum_fiddle;
        uint_t              rb_private;
        void                *rb_userdata;
        uint_t              rb_commitcnt;
        uint_t              rb_spare[1];
        struct timeval32       rb_timestamp;
        int                 rb_data[1];
} mddb_rb_t;
```

Record types

- Namespace
  - The directory entry flag is set to 0 for these records
- Metadevice
- Optimized Resync
- In Local Replica:
  - User records for: sets, drives, and nodes
- In Multi-owner Diskset replica only:
  - Transaction log for ongoing operations

Namespace Records

- NM_HDR (Namespace Header)
- SHR_NM (Shared Namespace)
  - Common entries for metadevice types and paths
- NM (Namespace)
  - Unique names of metadevices or physical devices
- DID_NM_HDR (Devid Namespace Header)
- DID_NM (Devid Namespace)
  - Unique slice numbers for physical devices in diskset
- DID_SHR_NM (Devid Shared Namespace)
  - Common devids for the disks in diskset

Metadevice Records

- MD_STRIPE
- MD_HOTSPARE, MD_HOTSPARE_POOL
- MD_MIRROR
- MD_SP
- MD_RAID
- MD_TRANSLOG, MASTER
- Can have 64bit or 32bit records on disk depending on whether or not the metadevices is greater than a Terabyte

Checksums

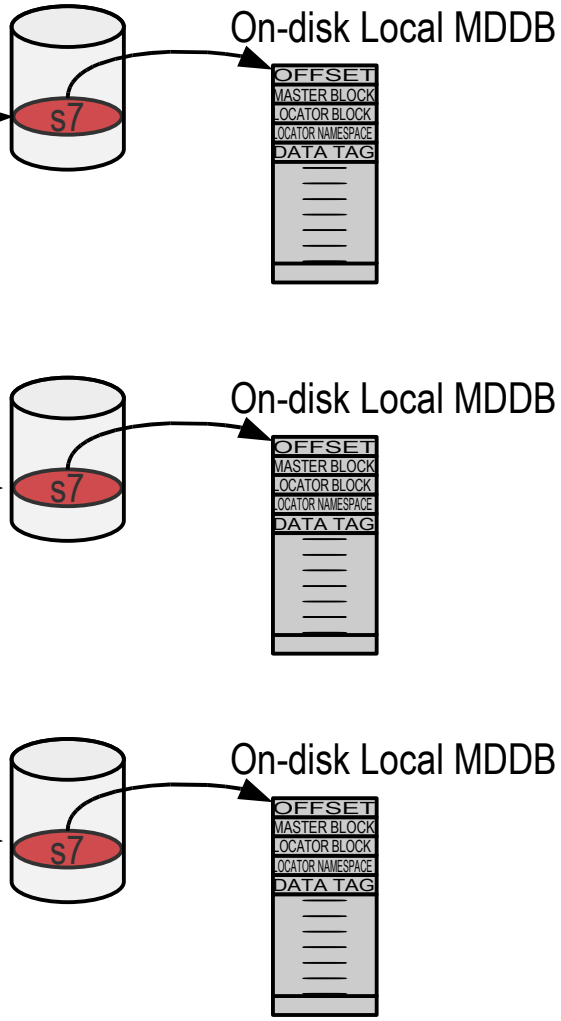
- Checksums used to check integrity of individual blocks and the entire replica.
- Checksum fiddles ensure consistency by means of a single-block write
- Checksum + checksum_fiddle for all blocks in replica should add together to be 0.

Local Replica Specifics

- Local Replica's user records track information about all disksets known to the system.
- In-core copy of local replica is in md_set[0] and is a list of pointers to different parts of replica

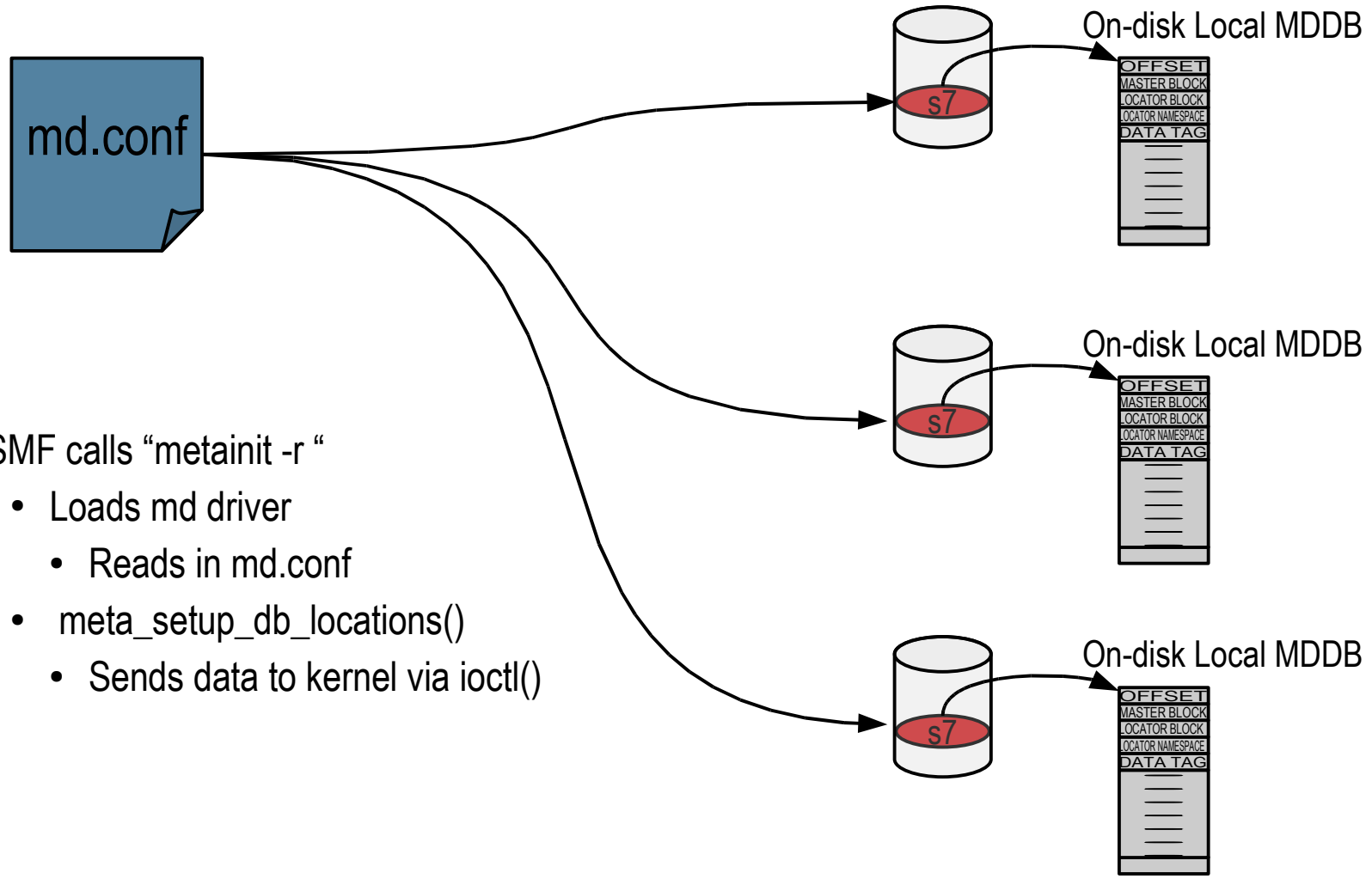
md.conf

/kernel/drv/md.conf



- Must be on system if SVM is running.
- Contains Locations of on-disk local replicas

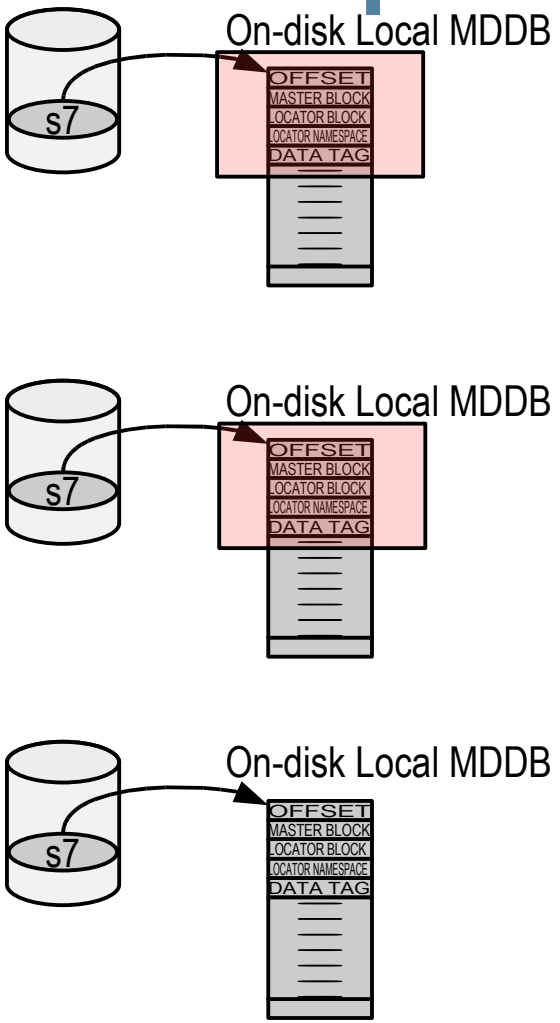
Boottime: Loading Local Replicas

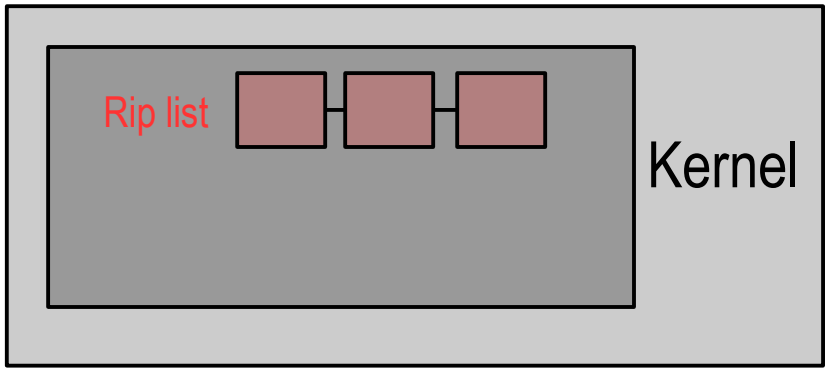


- SMF calls “metainit -r “
  - Loads md driver
    - Reads in md.conf
  - meta_setup_db_locations()
    - Sends data to kernel via ioctl()

Boottime: Loading Local Replicas

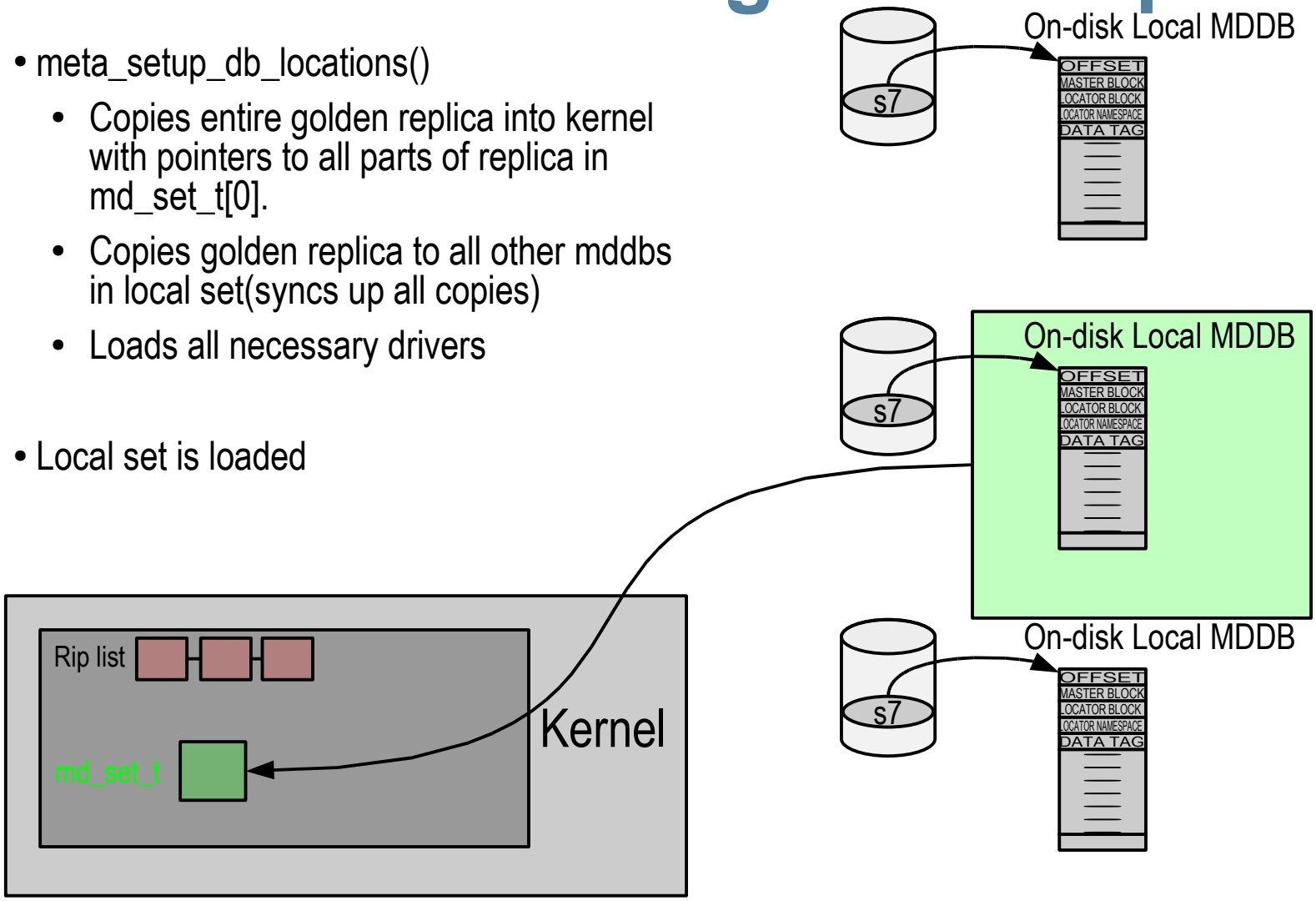
- meta_setup_db_locations()
  - Sends data to kernel
  - Reads in part of all local mddbs
  - Picks best replica(golden replica)





Boottime: Loading Local Replicas

- meta_setup_db_locations()
  - Copies entire golden replica into kernel with pointers to all parts of replica in md_set_t[0].
  - Copies golden replica to all other mddbs in local set(syncs up all copies)
  - Loads all necessary drivers
- Local set is loaded



Loading Diskset Replicas

- Local set contains records for the diskset
- The user must run take command for specific diskset: metaset -s foo -t
  - Grabs informations for set from local diskset
  - Process similar to local diskset take
- Creates entry in md_set[<set number>] with pointers to all relevant replica data
- Loads drivers if necessary

Loading Mirrored Root

- Root filesystem on top of the root mirror metadevice looks the same as the filesystem on the physical drive(no concatenations allowed)
- Newboot(x86 x64)
  - Root filesystem is initially mounted read only, read off of the physical device
  - At milestone: multiuser, remounts filesystem as read/write
- Sparc
  - ufs_remount_root()
    - loads local set and remounts metadevice as root

State Database Replicas