GEOM Tutorial

Poul-Henning Kamp

phk@FreeBSD.org

Outline

- Background and analysis.
- The local architectural scenery
- GEOM fundamentals.
- (tea break)
- Slicers (not a word about libdisk!)
- Tales of the unexpected.
- Q/A etc.

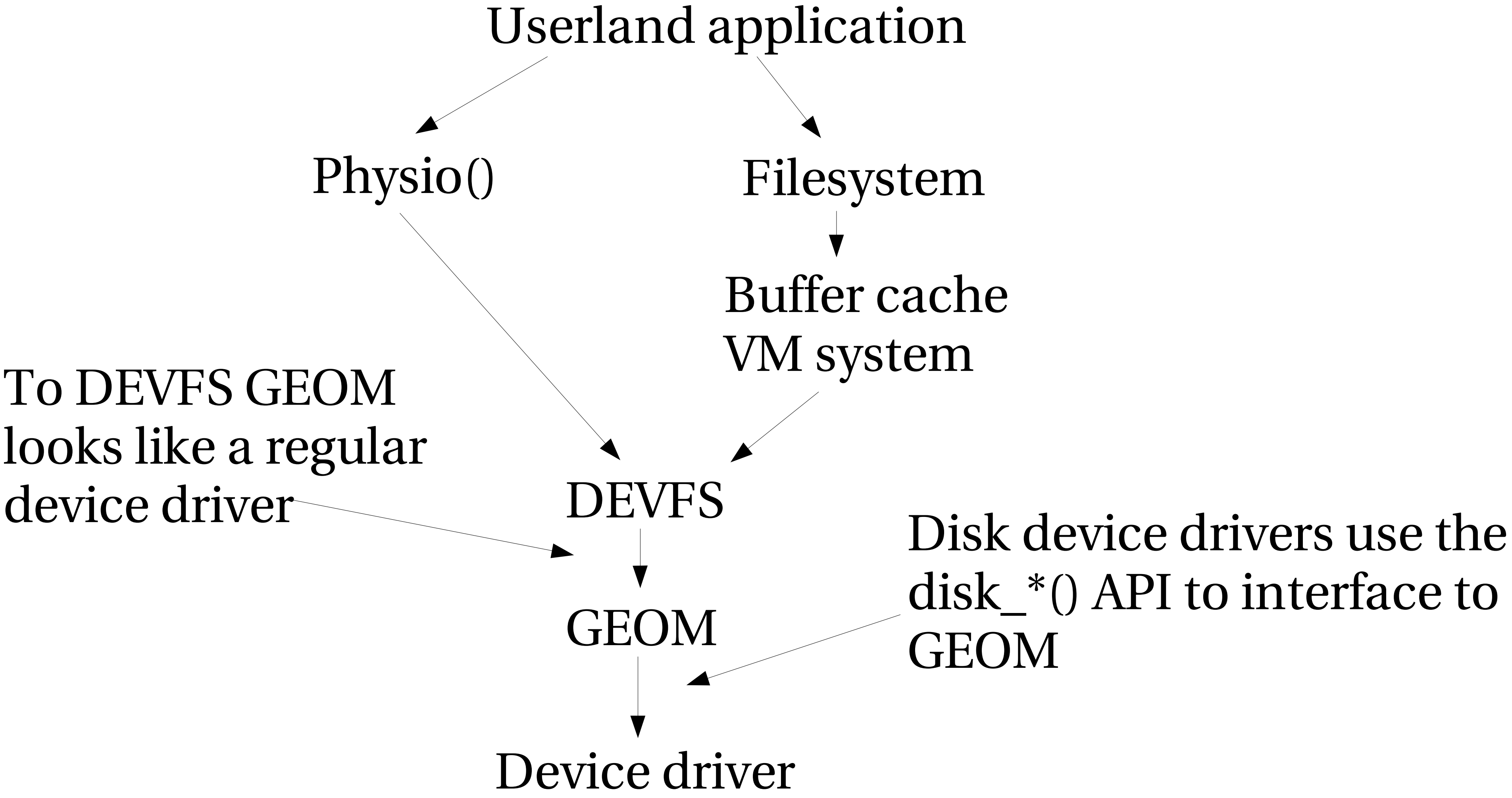
UNIX Disk I/O

- A disk is a one dimensional array of sectors.
  - 512 bytes/sector typical, but not required.
- Two I/O operations: read+write
  - Sectorrange: First sector + count.
  - RAM must be mapped into kernel.
- I/O request contained in struct buf/bio
- Schedule I/O by calling strategy()
- Completion signaled by biodone() callback.

GEOM does what ?

- Sits between DEVFS and device-drivers
- Provides framework for:
  - Arbitrary transformations of I/O requests.
  - Collection of statistics.
  - Disksort like optimizations.
  - Automatic configuration
  - Directed configuration.

“You are here”



Userland application

Physio()

Filesystem

Buffer cache
VM system

To DEVFS GEOM
looks like a regular
device driver

DEVFS

GEOM

Disk device drivers use the
disk_*() API to interface to
GEOM

Device driver

The GEOM design envelope.

- Modular.
- Freely stackable.
- Auto discovery.
- Directed Configuration.
- POLA
- DWIM
- No unwarranted politics.

“Modular”

- You cannot define a new transformation and insert it into Veritas volume manager, AIX LVM, Vinum or RaidFrame.
- They are all monolithic and closed.
  - “A quaint feature from the seventies”.

Freely stackable.

- Put your transformations in the order you like.
  - Mirror ad0 + ad1, partition the result.
  - Partition ad0 and ad1, mirror ad0a+ad1a, ad0b+ad1b, ad0c+ad1c, ad0d+ad1d ...
- Strictly defined interfaces between classes.

Auto discovery.

- Classes allowed to “automagically” respond to detectable clues.
  - Typically reacts to on-disk meta-data.
    - MBR, disklabel etc
  - Could also be other types of stimuli.

Directed configuration

- “root is always right”
  -- the kernel.
- Root should always be able to say “You may think it sounds stupid, but I want it!”
- ...as long as it does not compromise kernel integrity.

POLA

- Principle of Least Astonishment.
- Pola is not the same as
  “retain 1.0 compatibility at any cost!”
- Very hard to describe or codify, but intuitively obvious when violated.

DWIM

- Do What I Mean.
- Have sensible defaults.
- Make interfaces versatile but precise.
- Make sure interfaces have the right granularity.
- Be liberal to input, conservative in output.
- And be a total bastard to the programmers.

Say again ?

- I detest people who take short-cuts rather than do things right, because they leave shit for the rest of us to clean up.
- GEOM is fascist to prevent certain “obvious” hacks.
  - Try to sleep in the I/O path -> panic.
  - Lots of KASSERTS.
  - Etc.

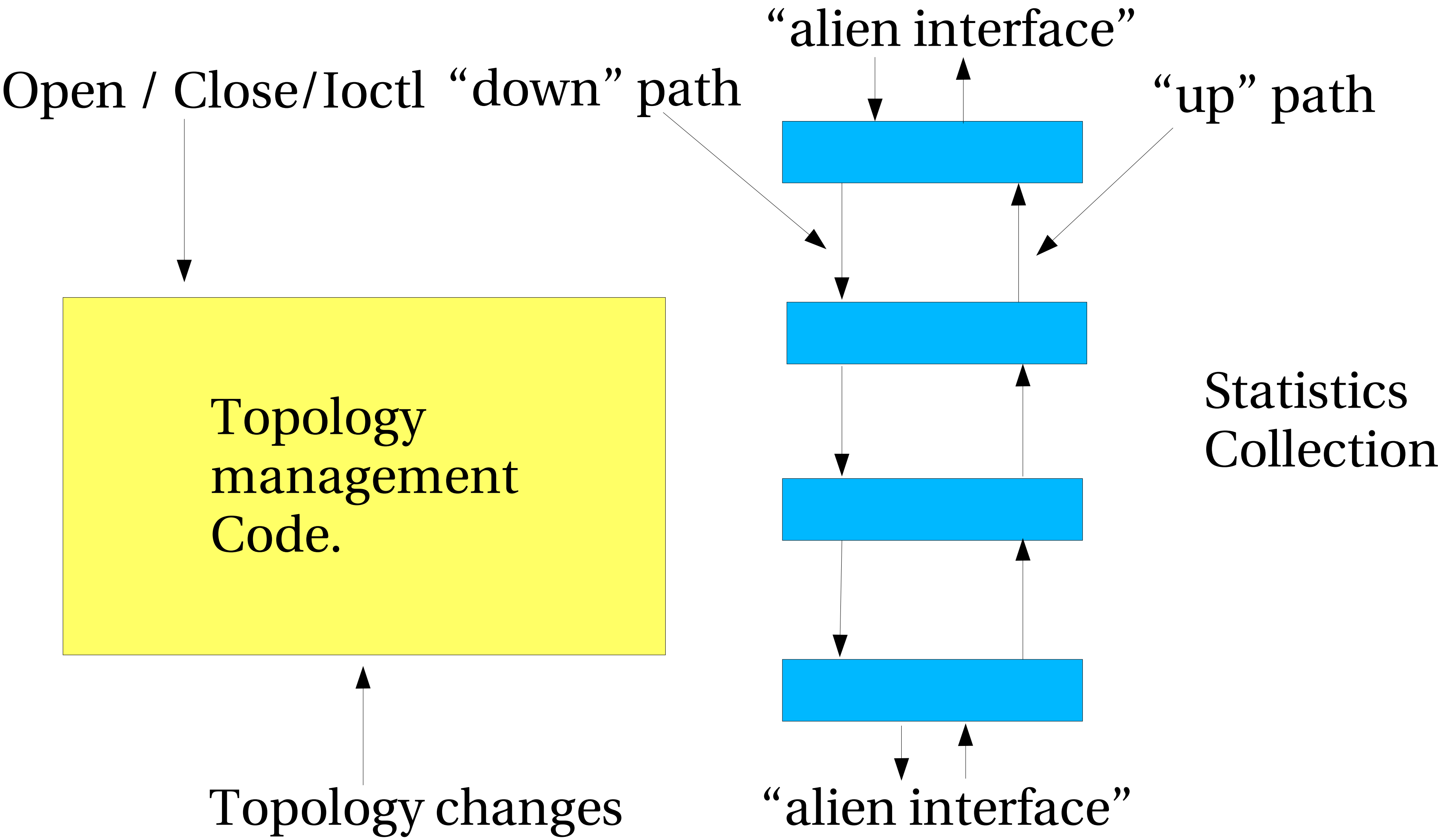
No unwarranted Policies.

- “FreeBSD: tools, not policies”.
- We are not in the business of telling people how they should do their work.
- We are in the business of giving them the best tools for their job.
- “UNIX is a tool-chest”

No unwarranted Policies.

- Leave maximal flexibility to the admin.
- Don't restrict use based on your:
  - High moral ground posturing
    - “Telnet is insecure, REMOVE IT!”
  - Unfounded theories
    - More or less anything Terry ever said.
  - Weak assumptions
    - “Heck nobody would ever do that!”

GEOM, the big view.



“alien interface”

Open / Close/Ioctl

“down” path

“up” path

Topology management Code.

Statistics Collection

Topology changes

“alien interface”

GEOM terminology.

- “A transformation”
  - The concept of a particular way to modify I/O requests.
    - Partitioning (BSD, MBR, GPT, PC98...).
    - Mirroring
    - Striping
    - RAID-5
    - Integrity checking
    - Redundant path selection.

GEOM terminology.

- “A class”
  - An implementation of a particular transformation.
    - MBR (partitioning)
    - BSD (ditto)
    - Mirroring
    - RAID-5
    - …

GEOM terminology.

- “A geom” (NB: lower case)
  - An instance of a class.
    - “the MBR which partitions the ad0 device”
    - “the BSD which partitions the ad0s1 device”
    - “the MIRROR which mirrors the ad2 and ad3 devices”
    - …

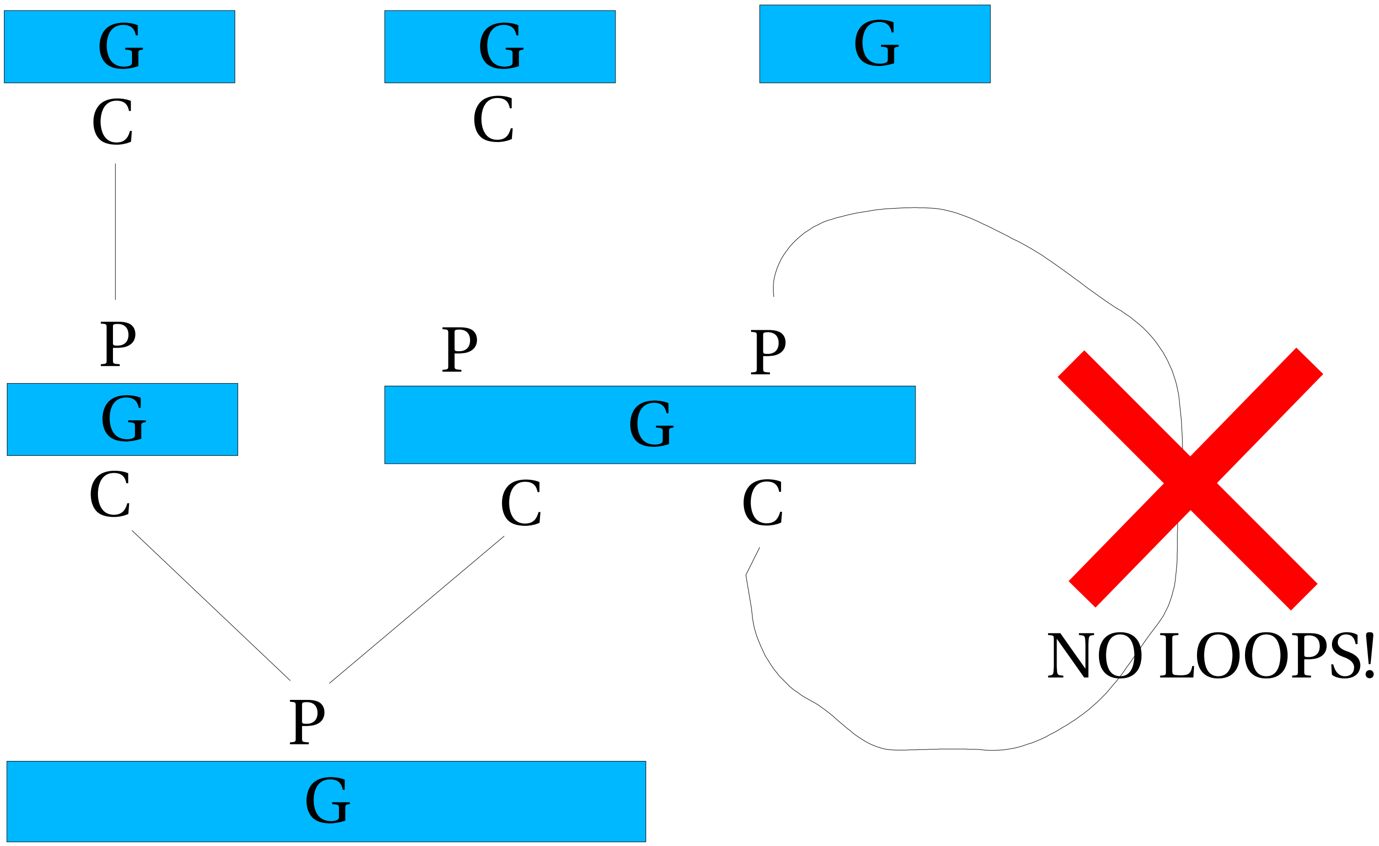
GEOM terminology.

- “A Provider”
  - A service point offered by a geom.
  - Corresponds loosely to “/dev entry”
    - ad0
    - ad0s1
    - ad0s1a
    - ad0.ad1.mirror

GEOM terminology.

- “A consumer”
  - The hook which a geom attach to a provider.
  - name-less, but not anonymous.

GEOM topology.



Topology limits:

- A geom can have 0..N consumers
- A geom can have 0..N providers.
- A consumer can be attached to a single provider.
- A provider can have many consumers attached.
- Topology must be a strictly directed graph.
  - No loops allowed.

I/O path.

- Requests are contained in “struct bio”.
- A request is **not** transitive.
  - Clone it
  - Modify the clone
  - … and pass the clone down.
- “start” entry point in geom used to schedule requests.
- bio->bio_done() used to signal completion.

I/O path

- Sleeping in I/O path is NOT allowed.
  - Queue the request and use a kthread or taskqueue.
  - ENOMEM handling is automatic
    - Returning a request with ENOMEM triggers retry with automatic backoff.
- Dedicated non-sleepable threads for pushing bios around.

I/O efficiency.

- Cannot sleep in up/down path
  - Enforced with hidden mutex.
- Don't do CPU heavy tasks in the up/down paths, use separate kthreads or task queue.
- Only one thread for each direction
  - Simplifies locking for classes.
  - Typically use .1% of cpu power.

I/O locking.

- Mutex on individual bio queues.
- Bio request scheduled on consumer.
  - Fails if not attached and open(ed enough).
- Bio records “from + to”.
- Bio reply follows recorded “to->from” path
  - Possible to answer after path has been removed.

Locking hierarchy

- To initiate I/O request:
  - Must have non-zero access count on consumer.
- To set access count on consumer:
  - Must hold "topology lock"
  - Consumer must be attached to provider.
  - Provider must accept.

Topology rules

- To attach consumer to provider:
  - Must not create a loop.
- To detach consumer
  - Must have zero access counts.
  - No outstanding I/O requests.

Topology rules

- To destroy consumer
  - Must not be attached.
- To destroy provider
  - Must not be attached.

Topology locking.

- The “topology lock”
  - Must be held to change the topology.
  - Must be held during open/close processing.
  - Not needed for I/O processing.
  - Doesn't stop I/O processing.
- Single “giantissimo” lock warranted by low frequency of use.

Class primitives.

- Create Class
  - Adds class to list of classes.
- Destroy Class
  - Fails if class in use.
- Normally handled by standard GEOM/KLD macros.

Geom primitives

- Create geom of specified class.
- Destroy geom
  - Fails if geom has consumers
  - Fails if geom has providers.

Provider primitives.

- Create provider on specified geom.
- Set provider error code.
  - Specify error code to start/stop all I/O.
- Orphan provider.
  - Tell consumers to bugger off.
- Destroy provider
  - Fails if attached.

Provider properties

- Name
- Mediasize
  - Total bytes on device
- Sectorsize
  - Size of addressable unit
- Stripesize and Stripoffset
  - Defines optimal request boundaries.

Other optional properties

- Can be queried with GET_ATTR() request.
  - Namespace is string
    - “class::attribute”
    - “GEOM::attribute”
- Examples:
  - GEOM::fwsectors
  - MBR::type
  - BSD::labelsum

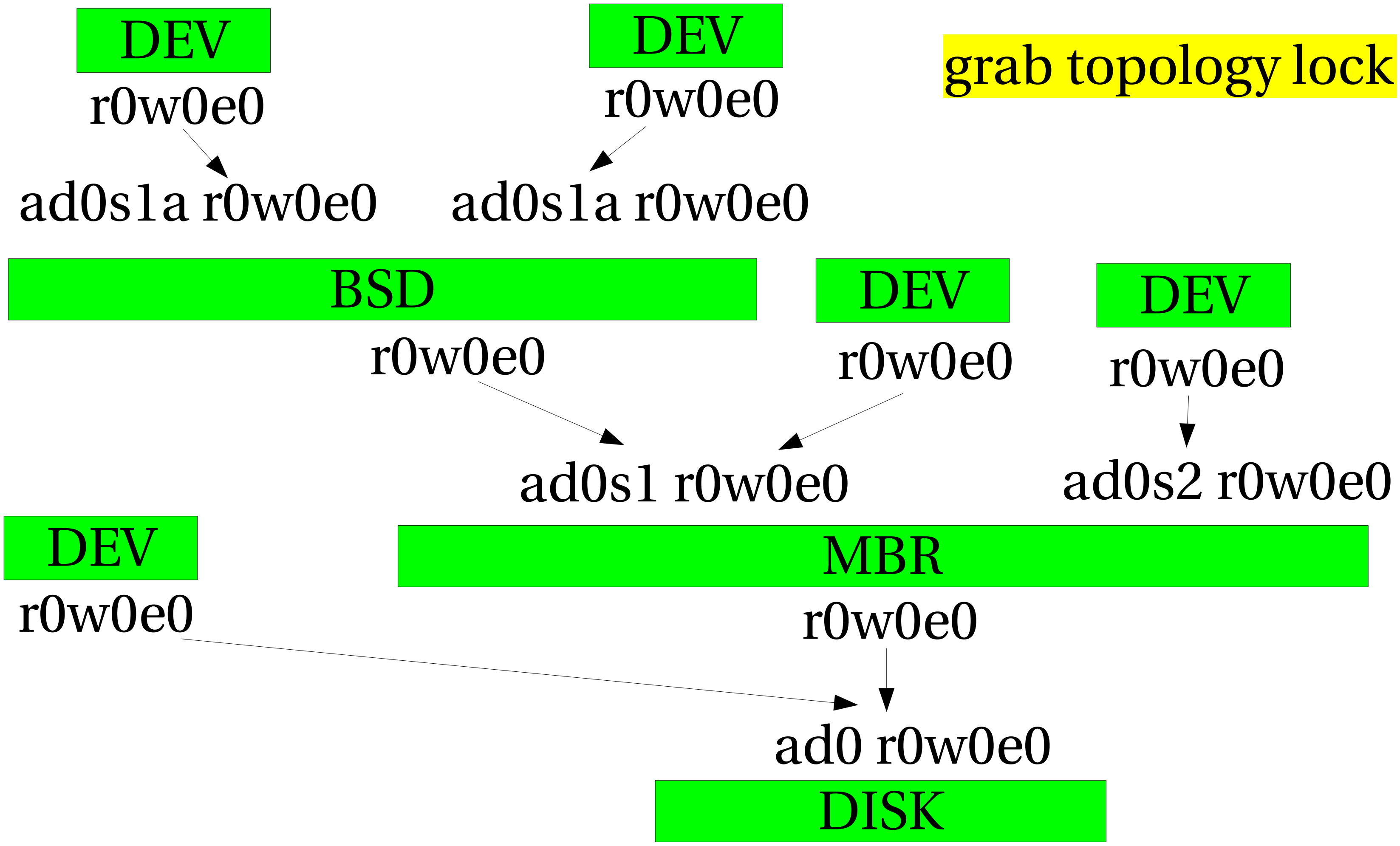
Consumer primitives.

- Create consumer on specified geom.
- Attach consumer to specified provider
- Change access counts of consumer.
  - Fails if not permitted or not attached.
- Detach
  - Fails if non-zero access or I/O counts.
- Destroy
  - Fails if attached

Access counts.

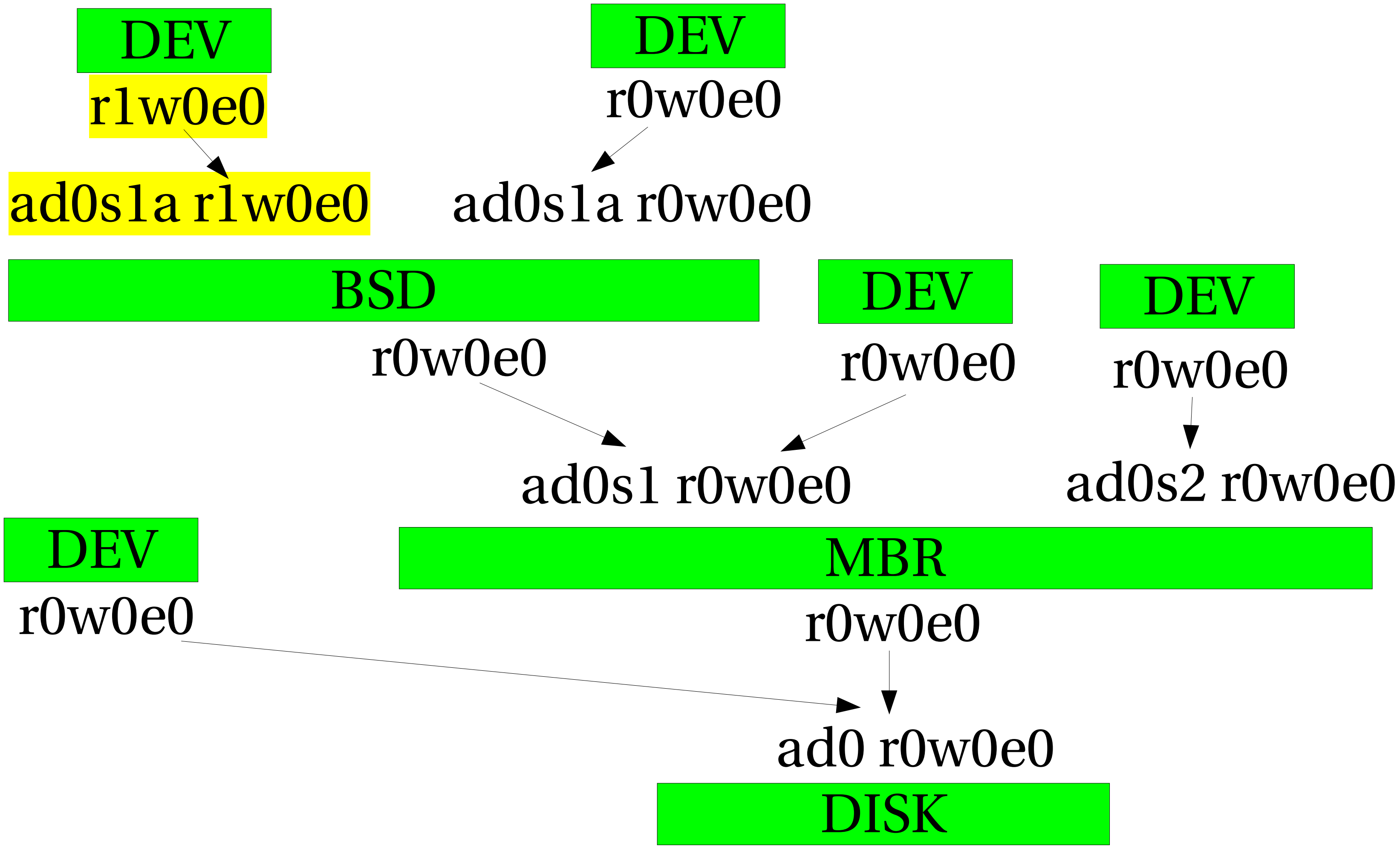
- Access is tracked as three reference counts:
  - Read gives read access.
  - Write gives write access.
  - Exclusive prevents others write access.
- Consumer and providers have associated counts.
- Providers count is the sum of all attached consumers counts.

How access counts work (1)

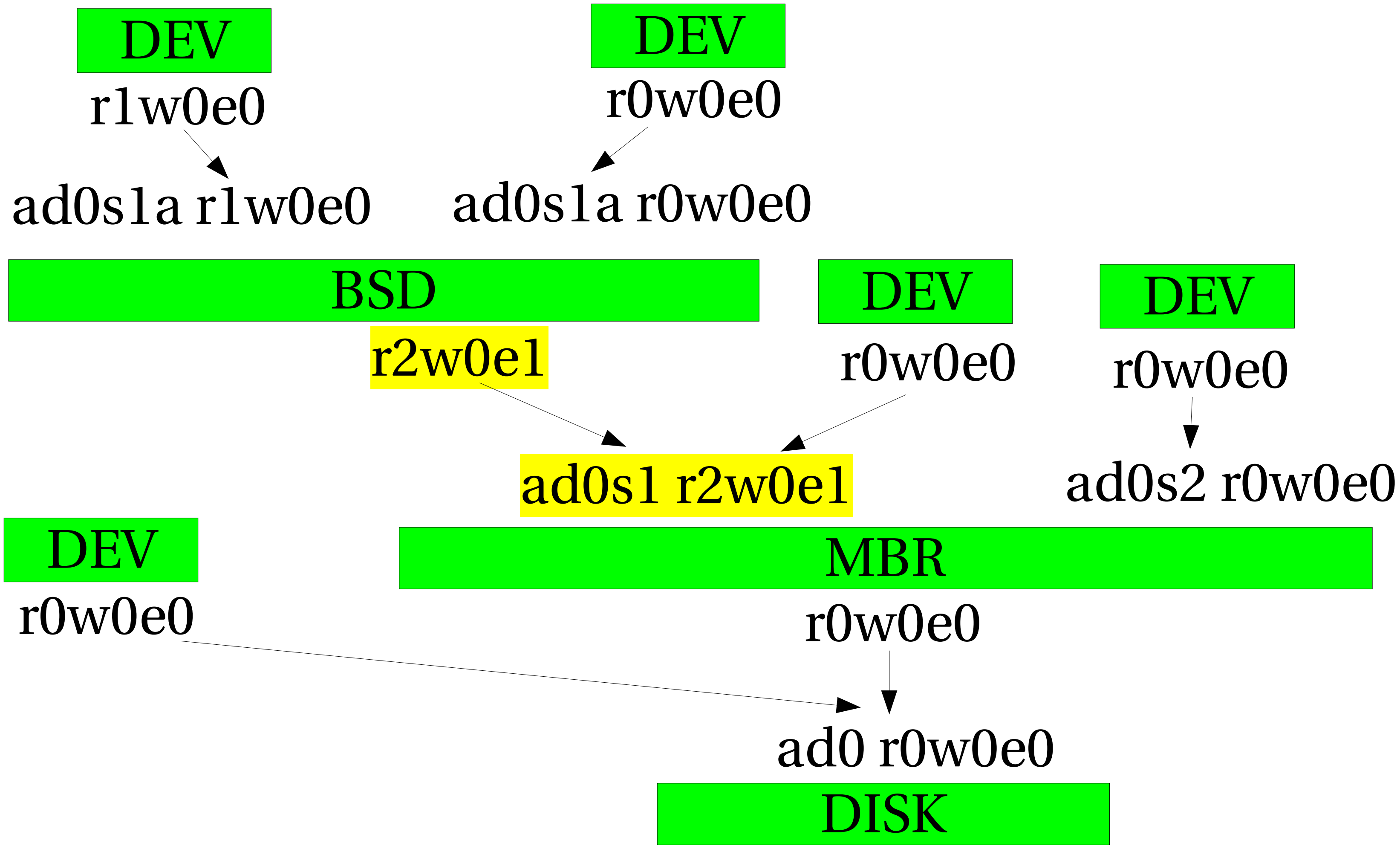


grab topology lock

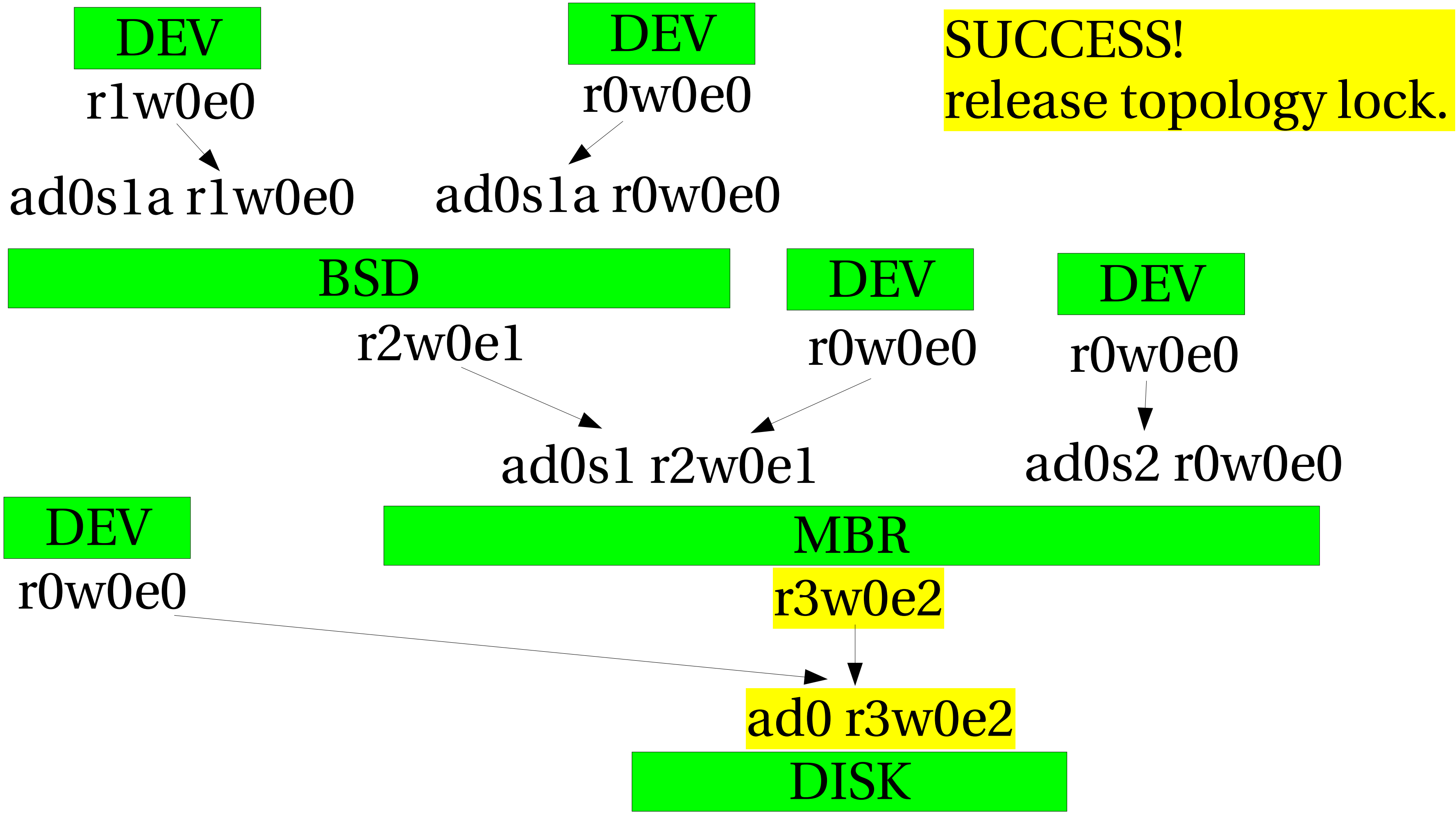
How access counts work (2)



How access counts work (3)

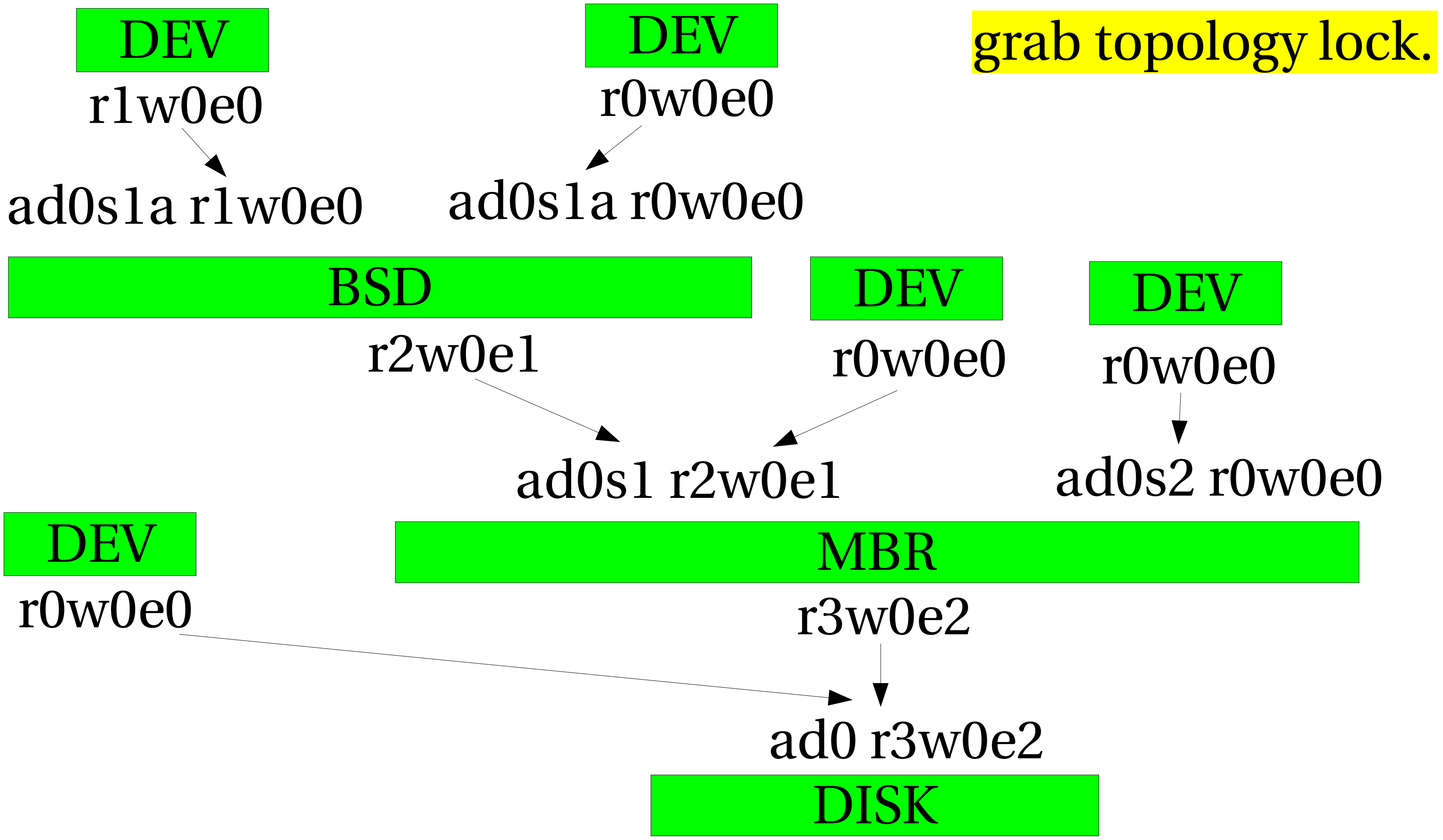


How access counts work (4)



SUCCESS!
release topology lock.

How access counts work (5)



DEV

r1w0e0

ad0s1a r1w0e0

DEV

r0w0e0

ad0s1a r0w0e0

grab topology lock.

BSD

r2w0e1

DEV

r0w0e0

ad0s1 r2w0e1

DEV

r0w0e0

ad0s2 r0w0e0

DEV

r0w0e0

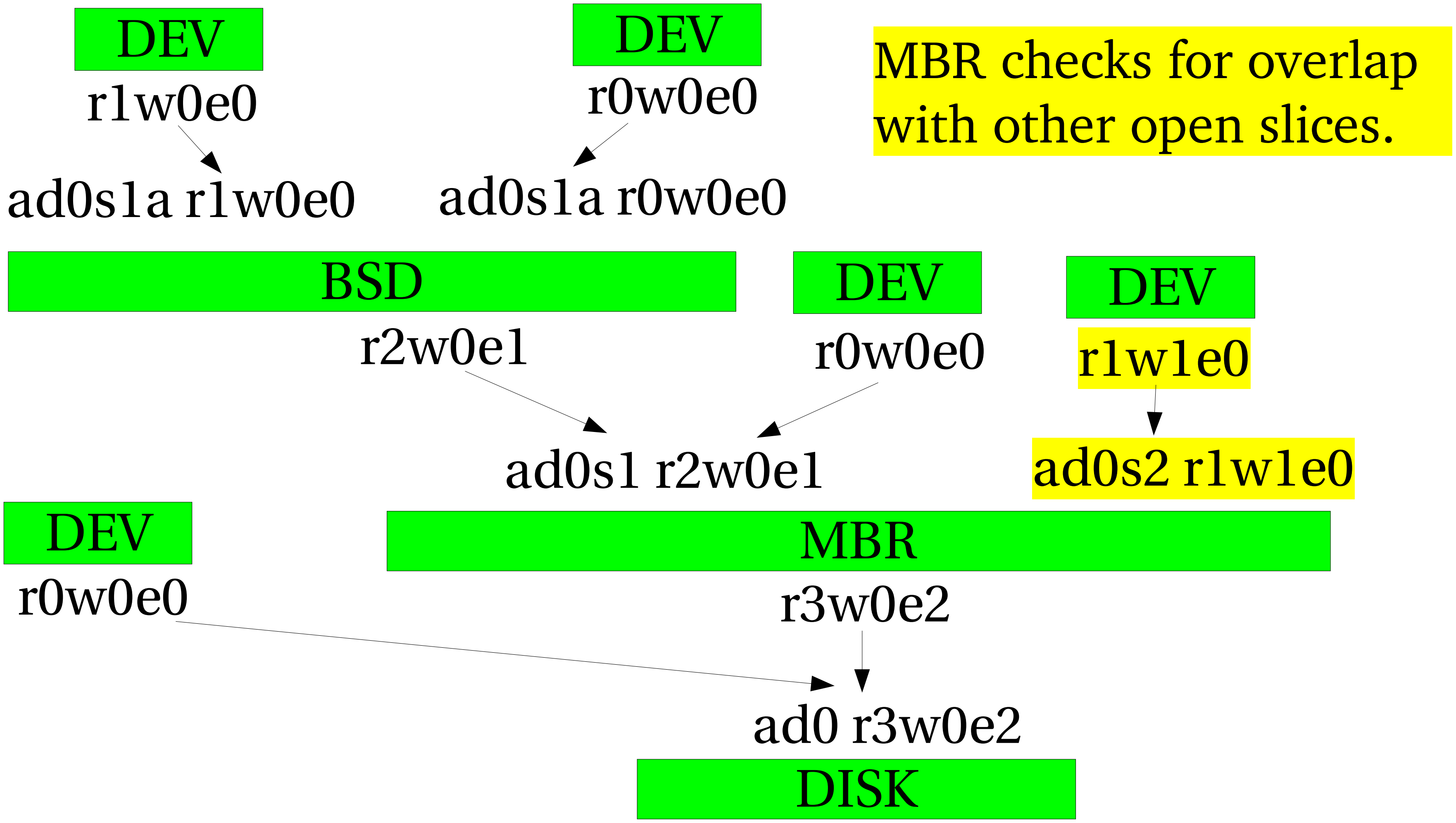
MBR

r3w0e2

ad0 r3w0e2

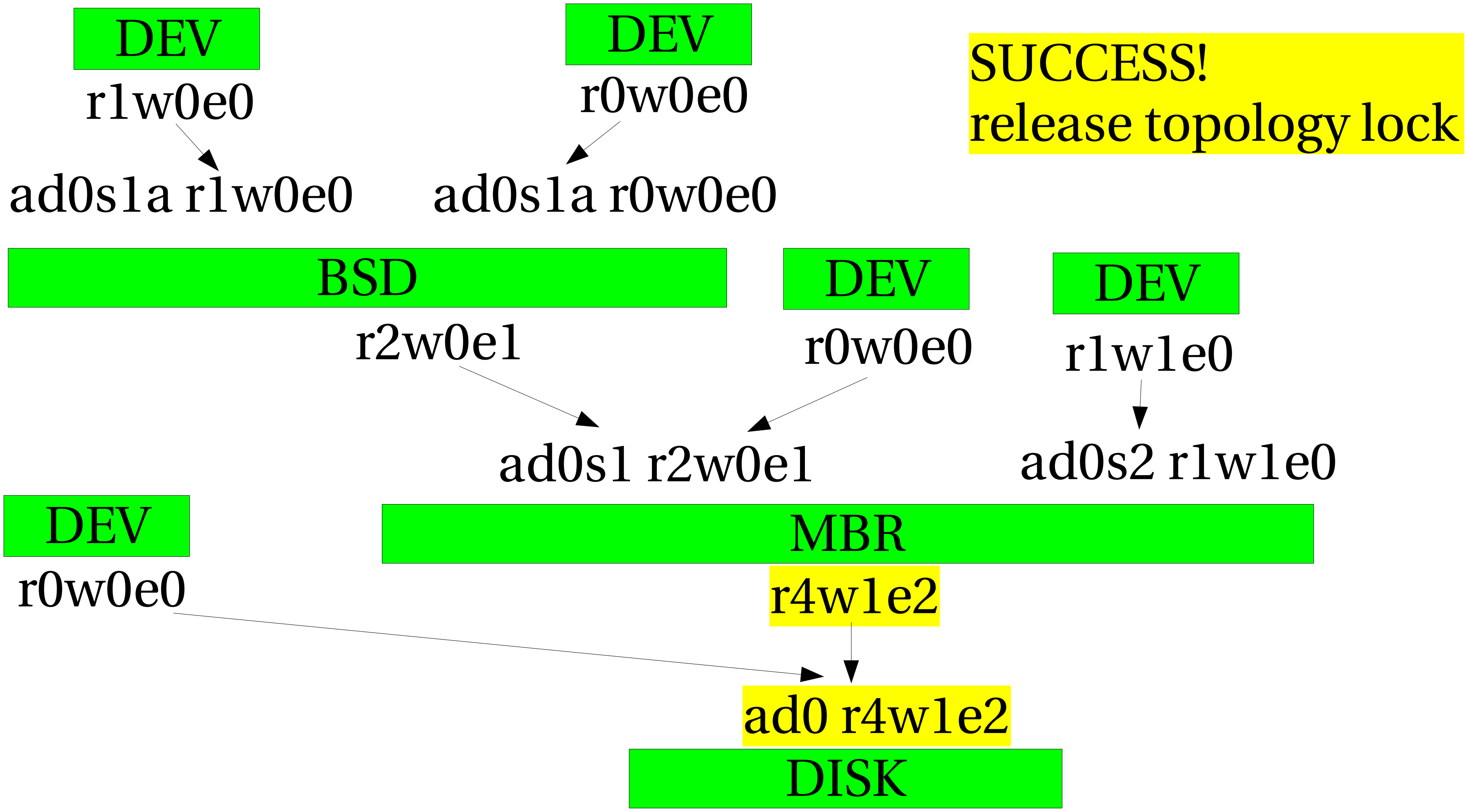
DISK

How access counts work (6)



MBR checks for overlap with other open slices.

How access counts work (7)



SUCCESS!
release topology lock

DEV
r1w0e0

DEV
r0w0e0

ad0s1a r1w0e0

ad0s1a r0w0e0

BSD
r2w0e1

DEV
r0w0e0

DEV
r1w1e0

ad0s1 r2w0e1

ad0s2 r1w1e0

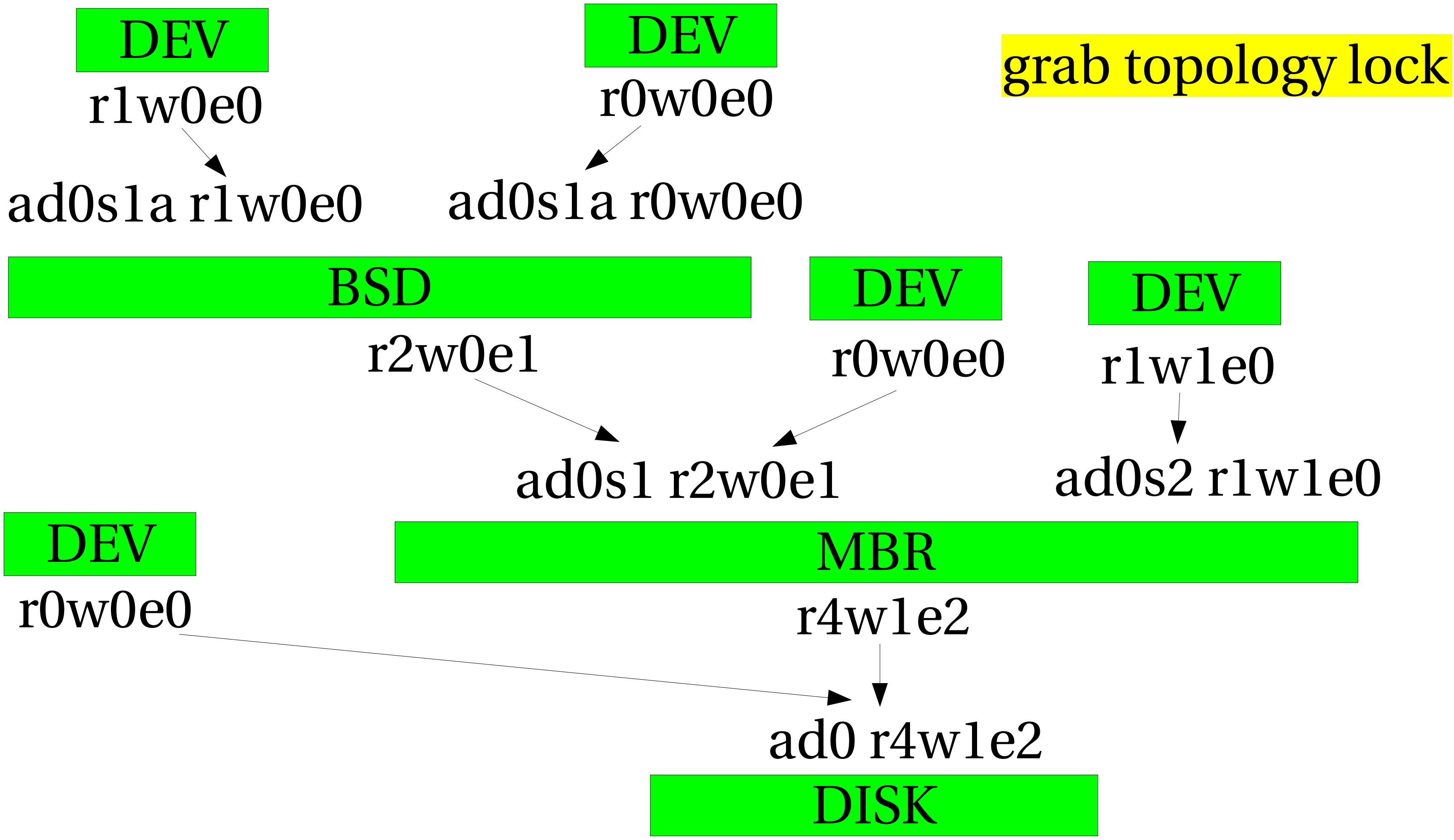
DEV
r0w0e0

MBR
r4w1e2

ad0 r4w1e2

DISK

How access counts work (8)



grab topology lock

DEV

r1w0e0

ad0s1a r1w0e0

DEV

r0w0e0

ad0s1a r0w0e0

BSD

r2w0e1

DEV

r0w0e0

ad0s1 r2w0e1

DEV

r1w1e0

ad0s2 r1w1e0

MBR

r4w1e2

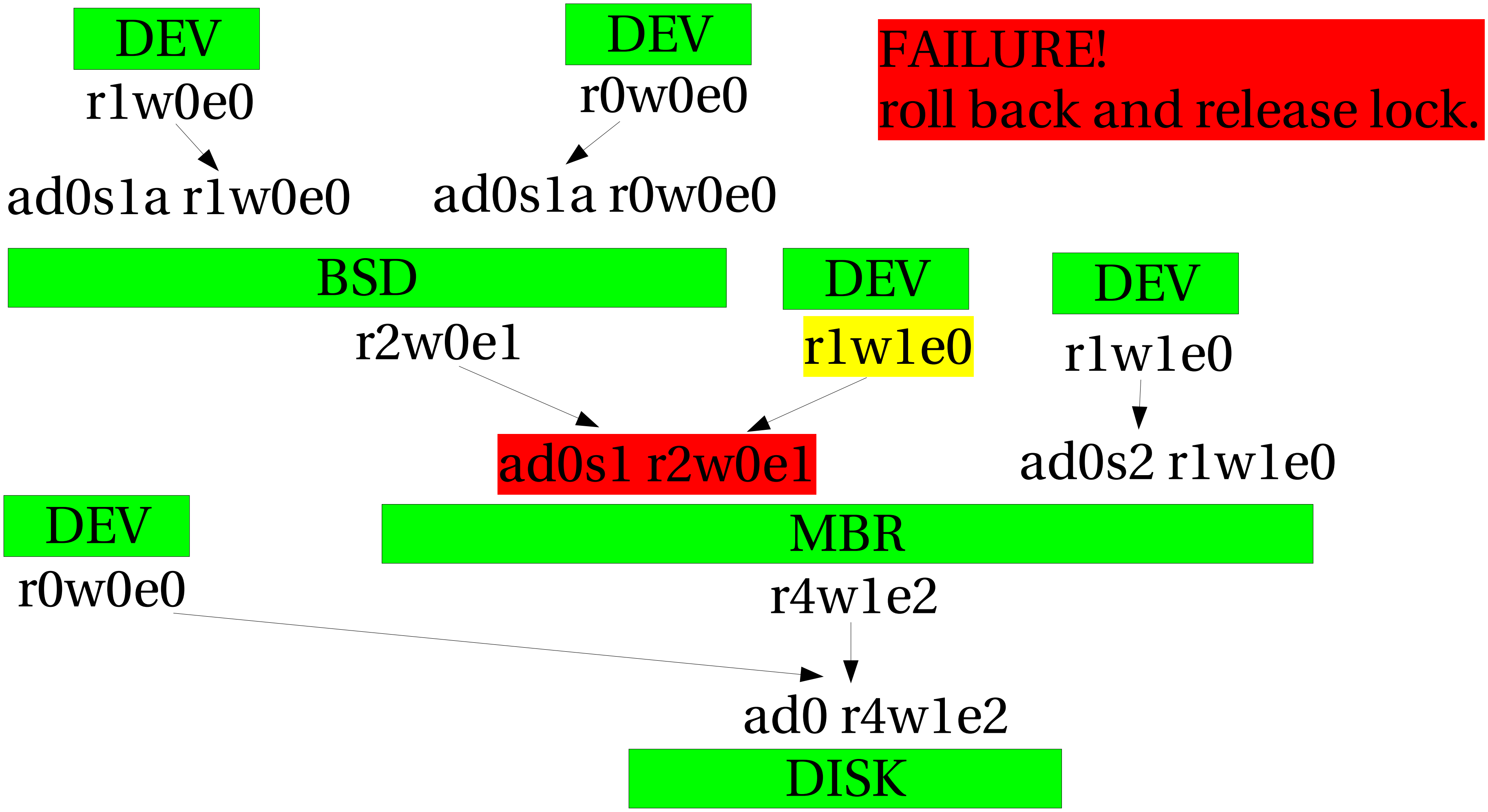
DEV

r0w0e0

ad0 r4w1e2

DISK

How access counts work (9)



FAILURE!
roll back and release lock.

GEOM ahead of the kernel.

- Kernel didn't used to provide strong access checks at the disk-IO level.
- Primitives insufficient to express R/W/E policy fully.
- File systems sloppy with handling even what is supported.
  - mount r/o => open r/o
  - remount r/w => no reopen to r/w mode.

Events and all that.

- GEOM has an internal job-queue for executing auto discovery and other housekeeping.
- Events posted on a queue.
  - Orphan events on dedicated queue.
  - Event queue protected by event mutex.
- Dedicated event thread grabs topology lock, executes event and releases lock.

Event queue

- Strictly FIFO processing.
  - Orphans before general events.
- Events tagged by identifiers
  - (void *)
- Events can be cancelled by identifier.
- Once Giant is removed, the event kqueue can become a normal taskqueue function.

User land and events.

- All user land operations which need topology lock must wait for empty event queue.
  - open/close/ioctl
- Explicit “process all events” calls may be needed in class code.
- Event queue useful to isolate Giant infected code from Giant free code.

“New Class” event.

- Posted when a class is added.
- Results in the class being offered a chance to “taste” all current providers in the system.

“New Provider” event.

- Posted when provider is created.
  - All classes gets the offer.
- Posted when a provider write access count goes to zero.
  - Meta data for a class may have been created.
  - Only classes not already attached are offered a chance to taste the provider.

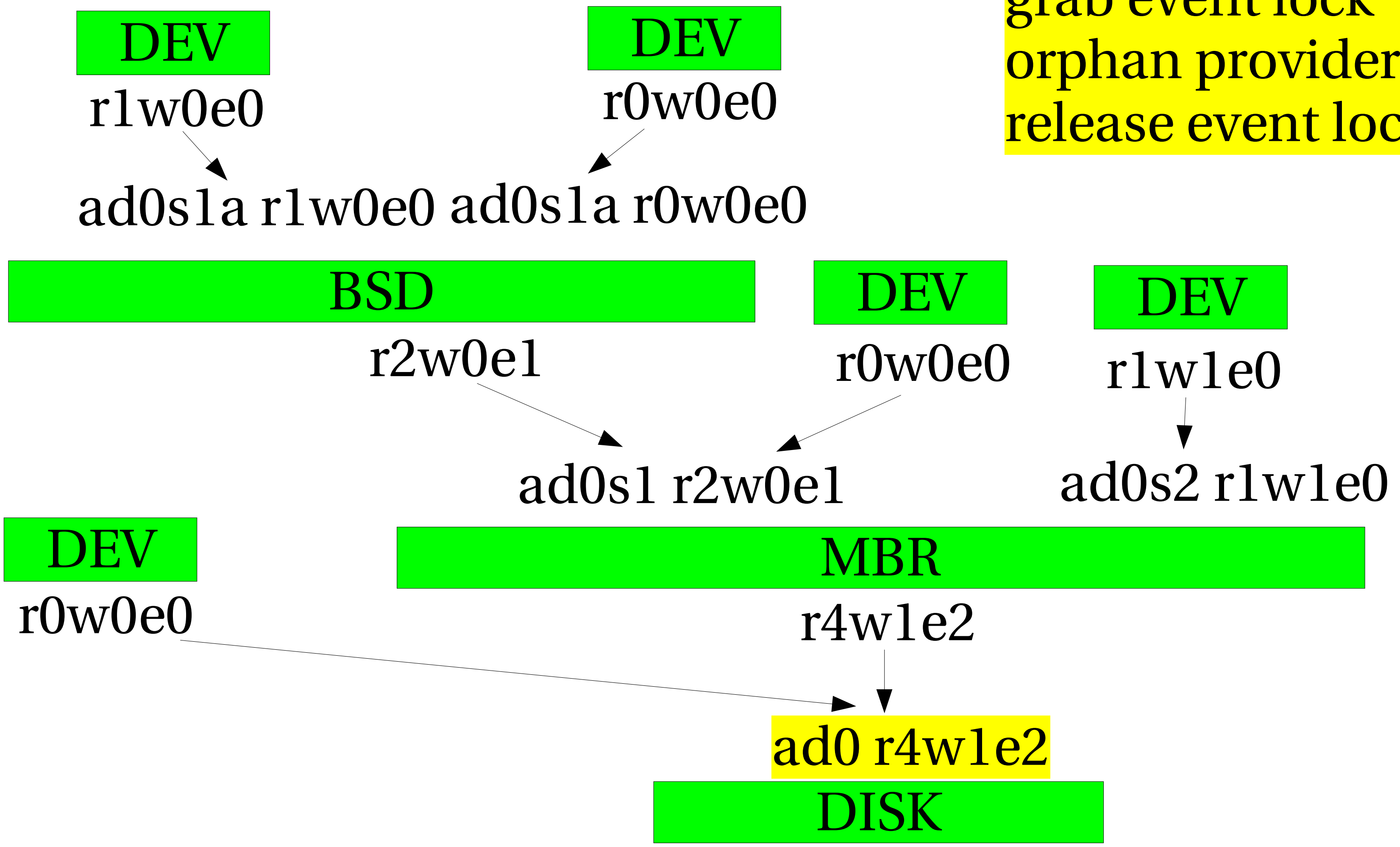
“Orphan” event..

- Devices disappear without notice.
- That's hardware for you...
- Not nice from a UNIX philosophy.
- But we have to cope...

“Orphan” event..

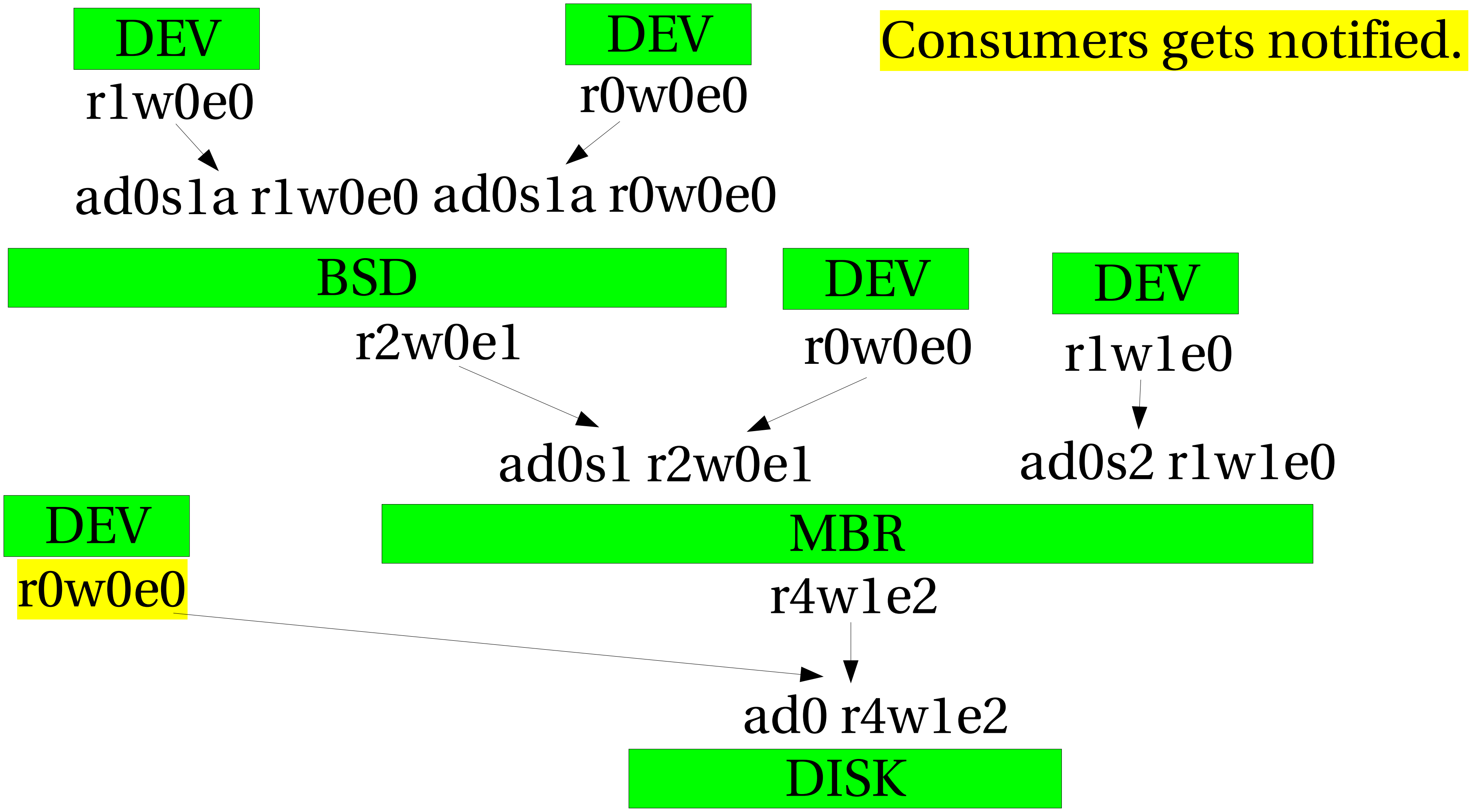
- A provider can be “orphaned” by its geom.
  - All future I/O requests fail.
  - All In-transit I/O requests can still complete
    - They shall complete!
  - Consumers get notified.
  - Consumers expected to zero access counts and detach.
  - Only then can the provider be destroyed.

How orphaning work (1)

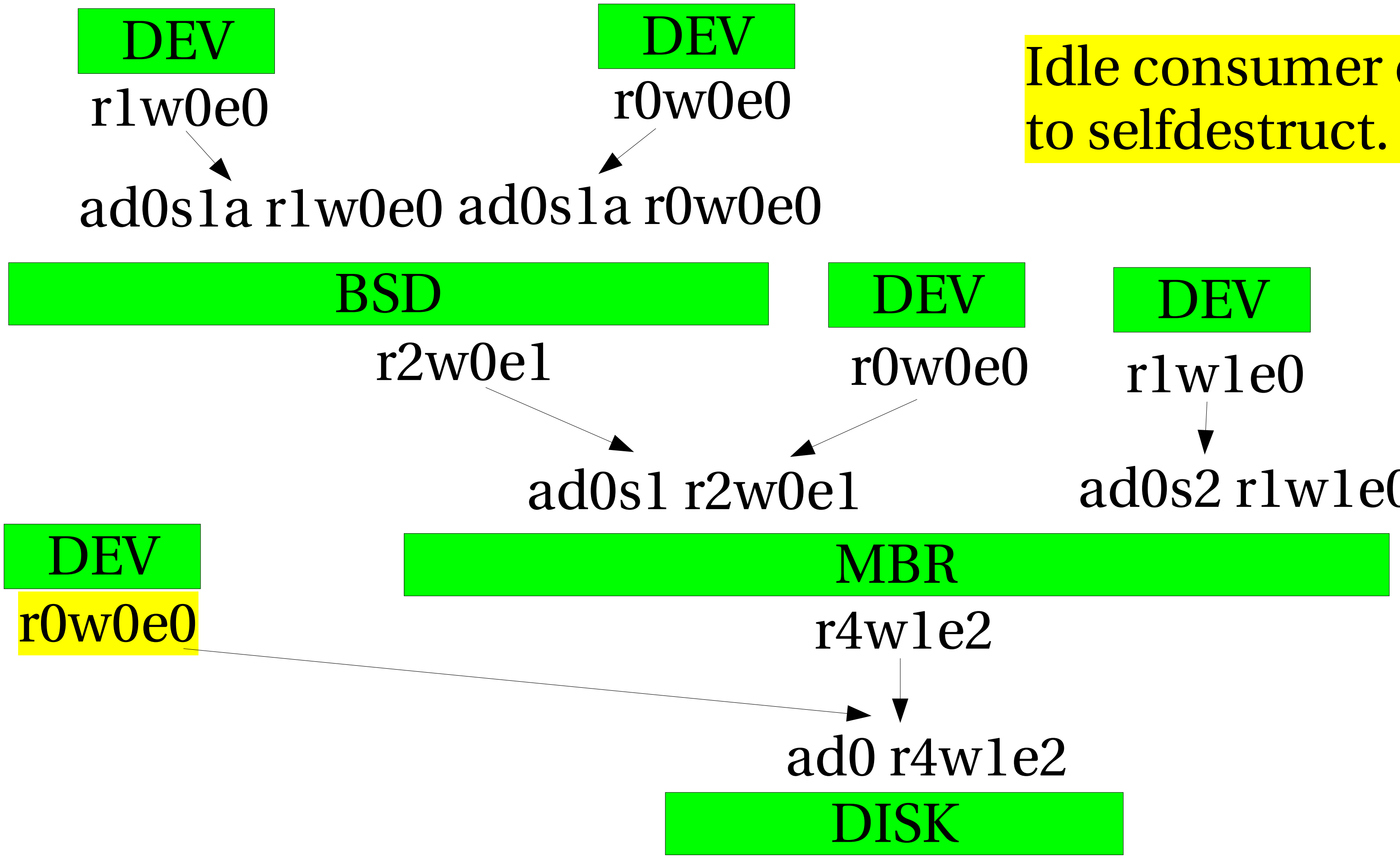


grab event lock
orphan provider.
release event lock.

How orphaning work (2)

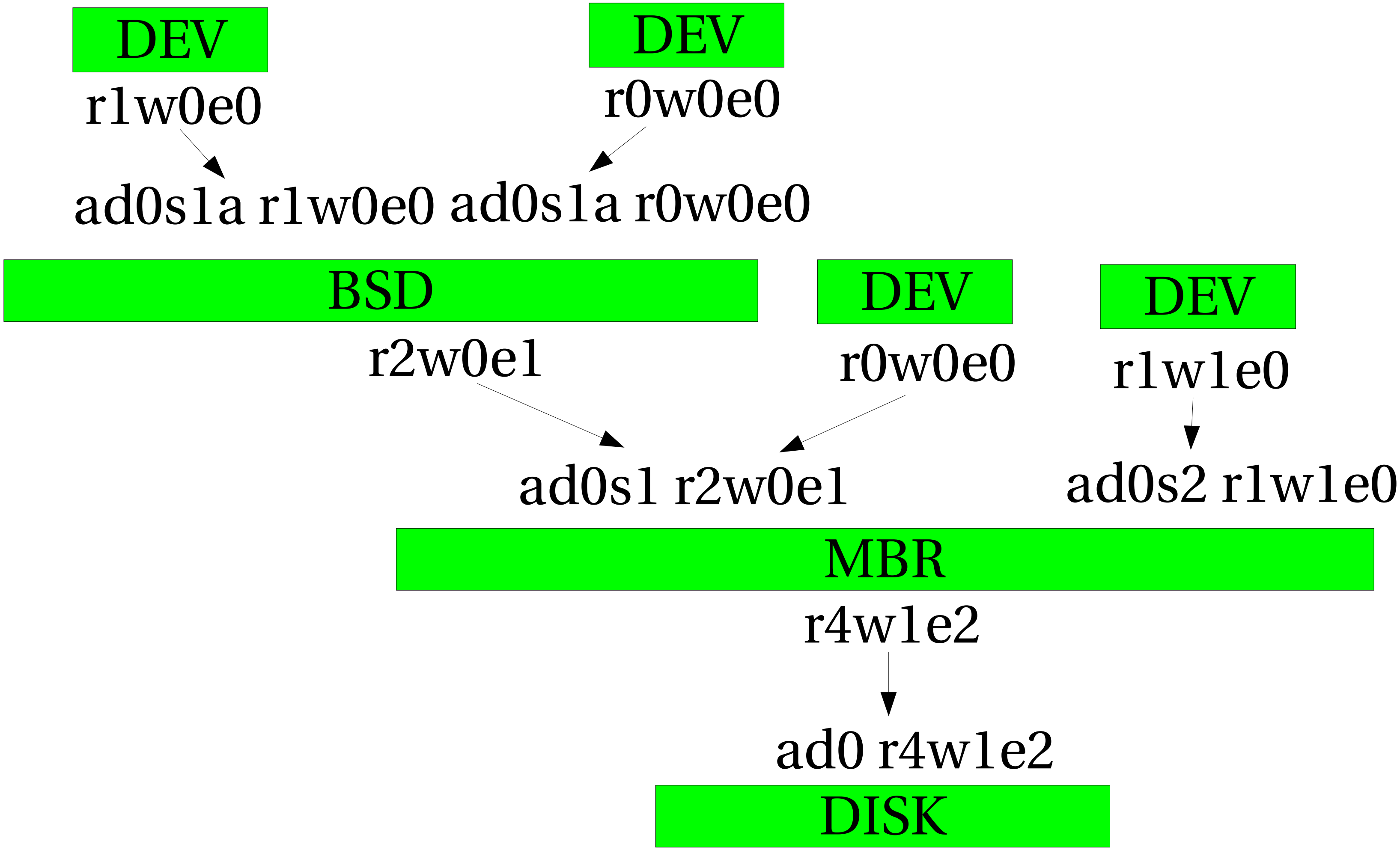


How orphaning work (3)

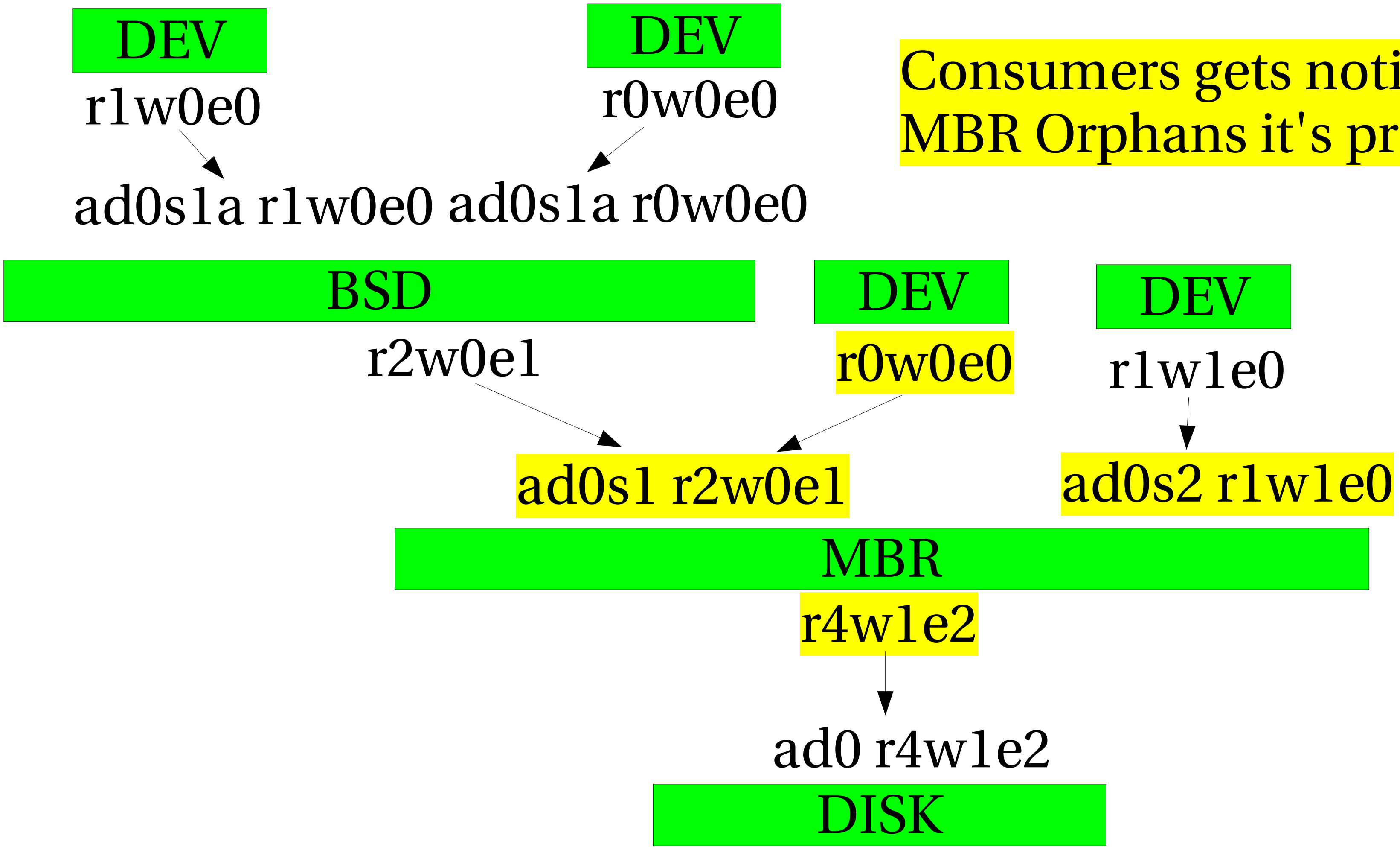


Idle consumer decides to selfdestruct.

How orphaning work (4)



How orphaning work (5)



DEV

r1w0e0

DEV

r0w0e0

ad0s1a r1w0e0 ad0s1a r0w0e0

BSD

DEV

DEV

r2w0e1

r0w0e0

r1w1e0

ad0s1 r2w0e1

ad0s2 r1w1e0

MBR

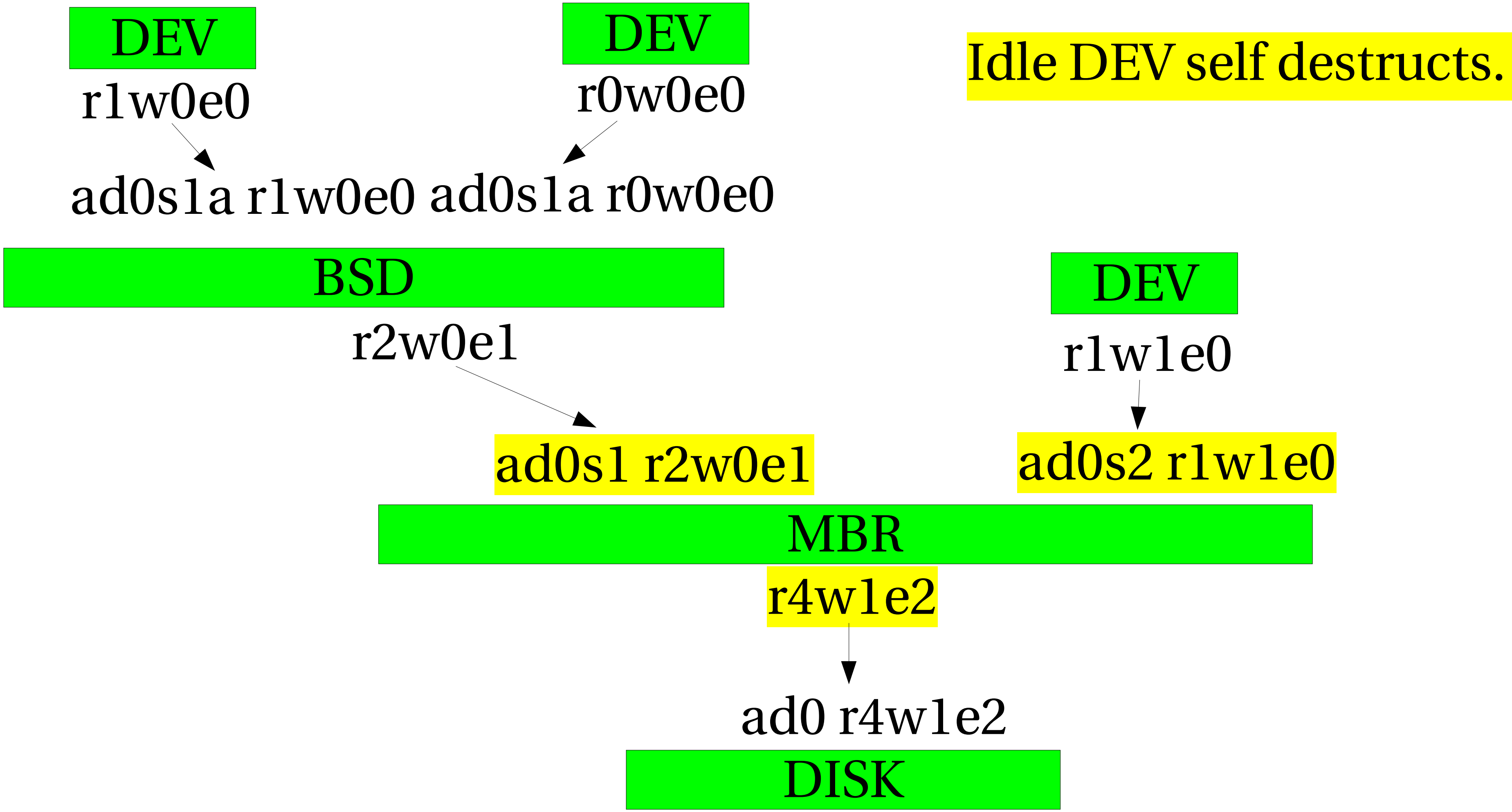
r4w1e2

ad0 r4w1e2

DISK

Consumers gets notified.
MBR Orphans it's providers.

How orphaning work (6)



DEV

r1w0e0

DEV

r0w0e0

Idle DEV self destructs.

ad0s1a r1w0e0 ad0s1a r0w0e0

BSD

r2w0e1

DEV

r1w1e0

ad0s1 r2w0e1

ad0s2 r1w1e0

MBR

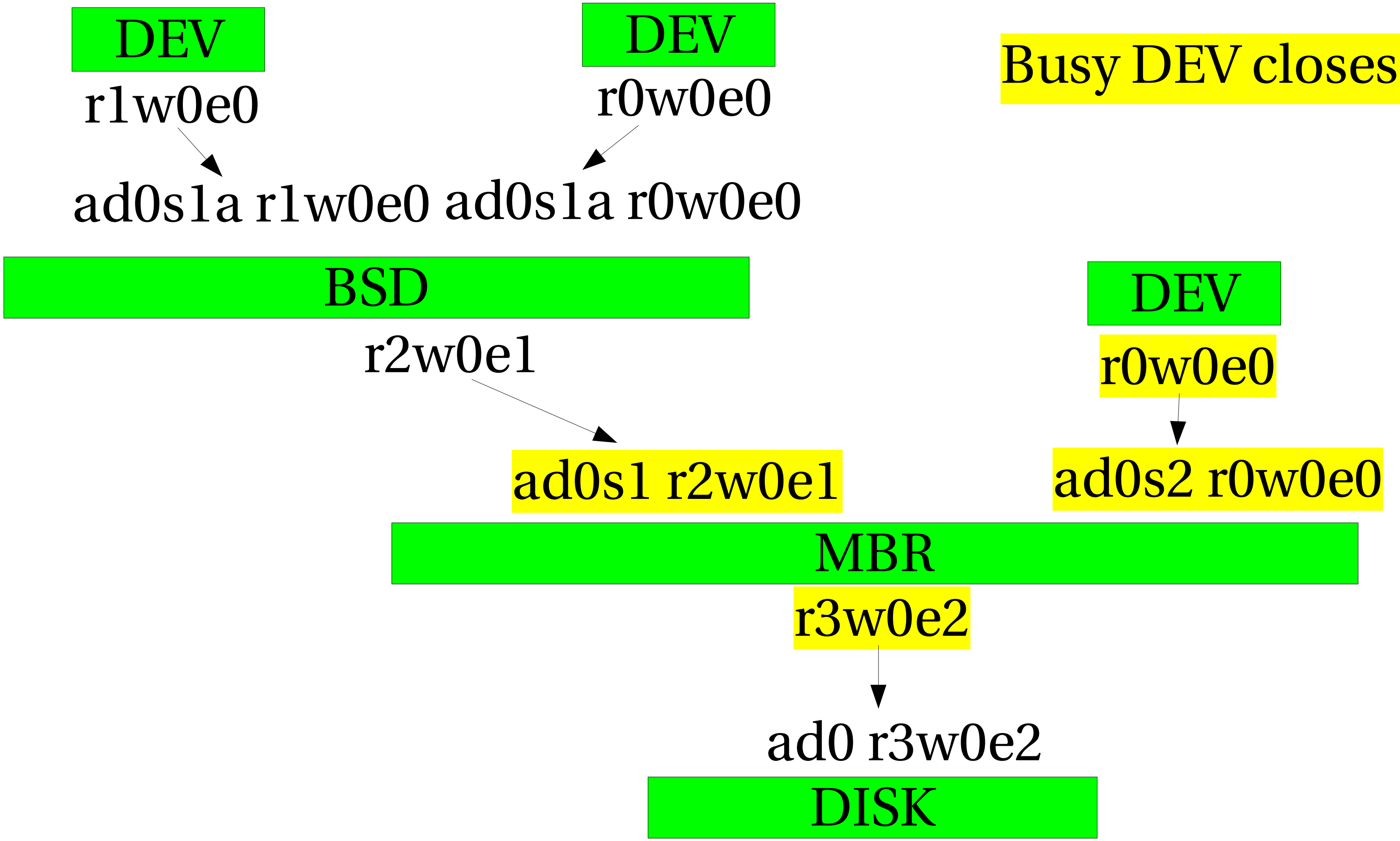
r4w1e2

ad0 r4w1e2

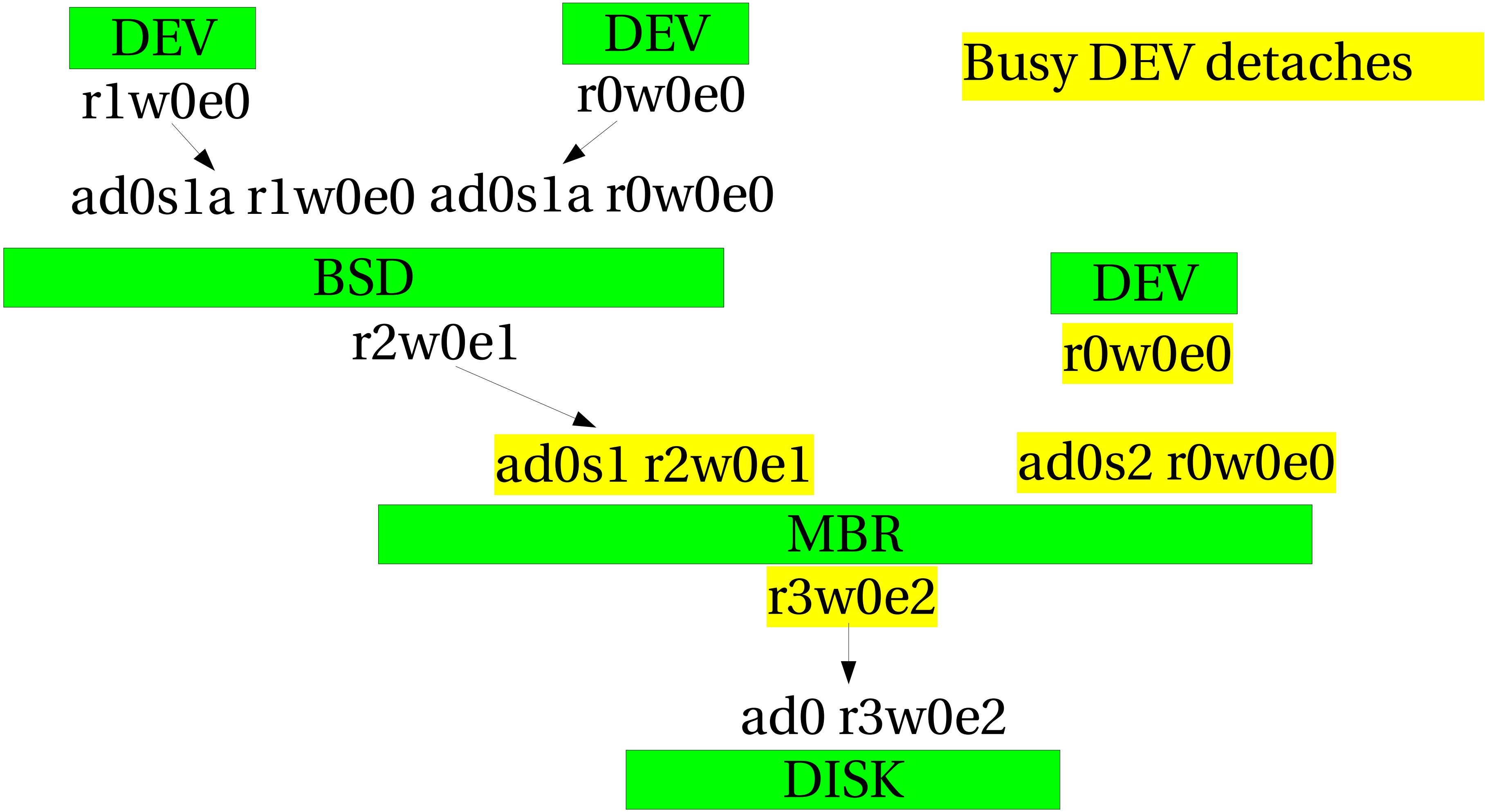
DISK

How orphaning work (7)

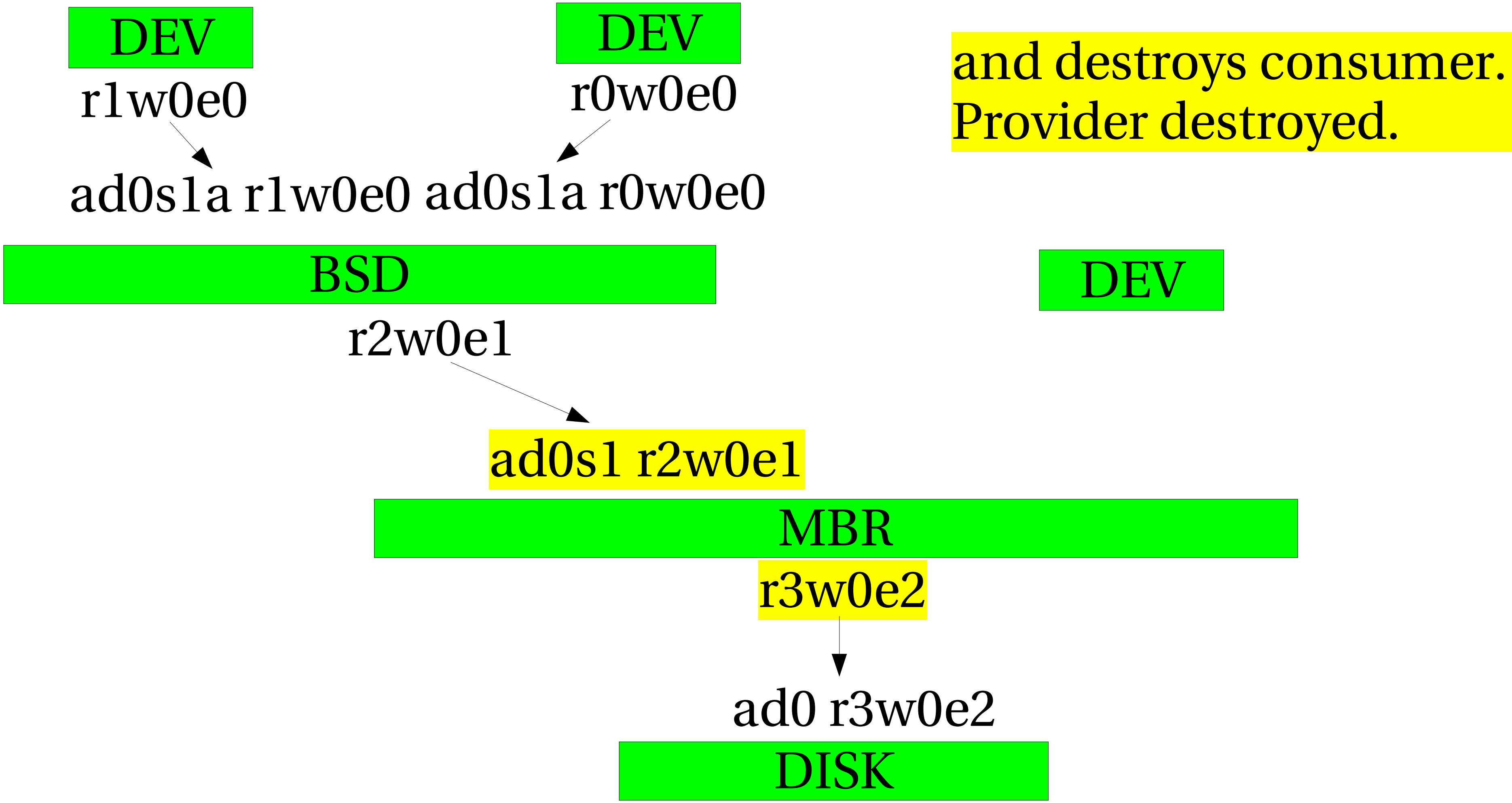
Busy DEV closes



How orphaning work (8)



How orphaning work (9)



and destroys consumer.
Provider destroyed.

DEV

r1w0e0

DEV

r0w0e0

ad0s1a r1w0e0 ad0s1a r0w0e0

BSD

r2w0e1

DEV

ad0s1 r2w0e1

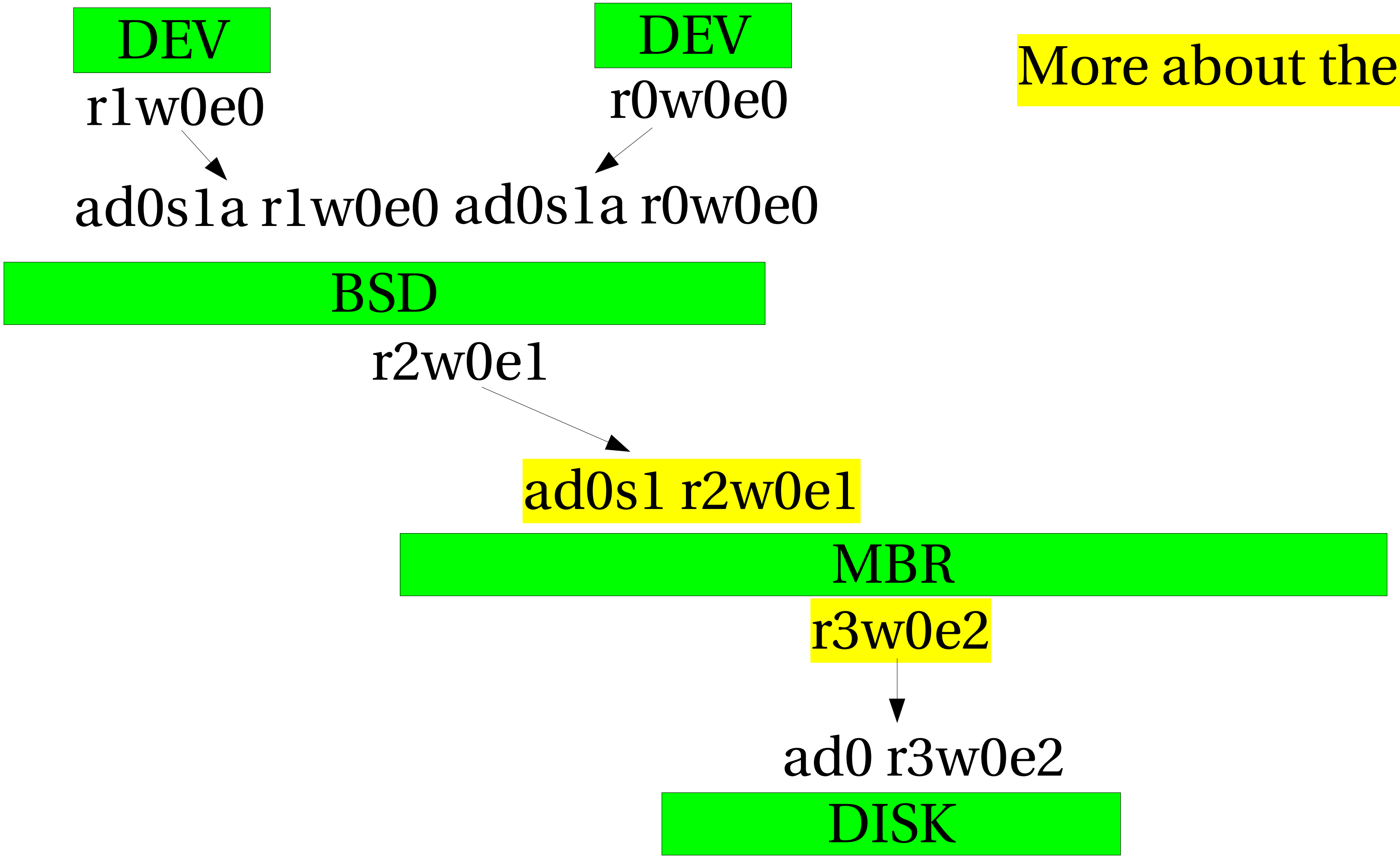
MBR

r3w0e2

ad0 r3w0e2

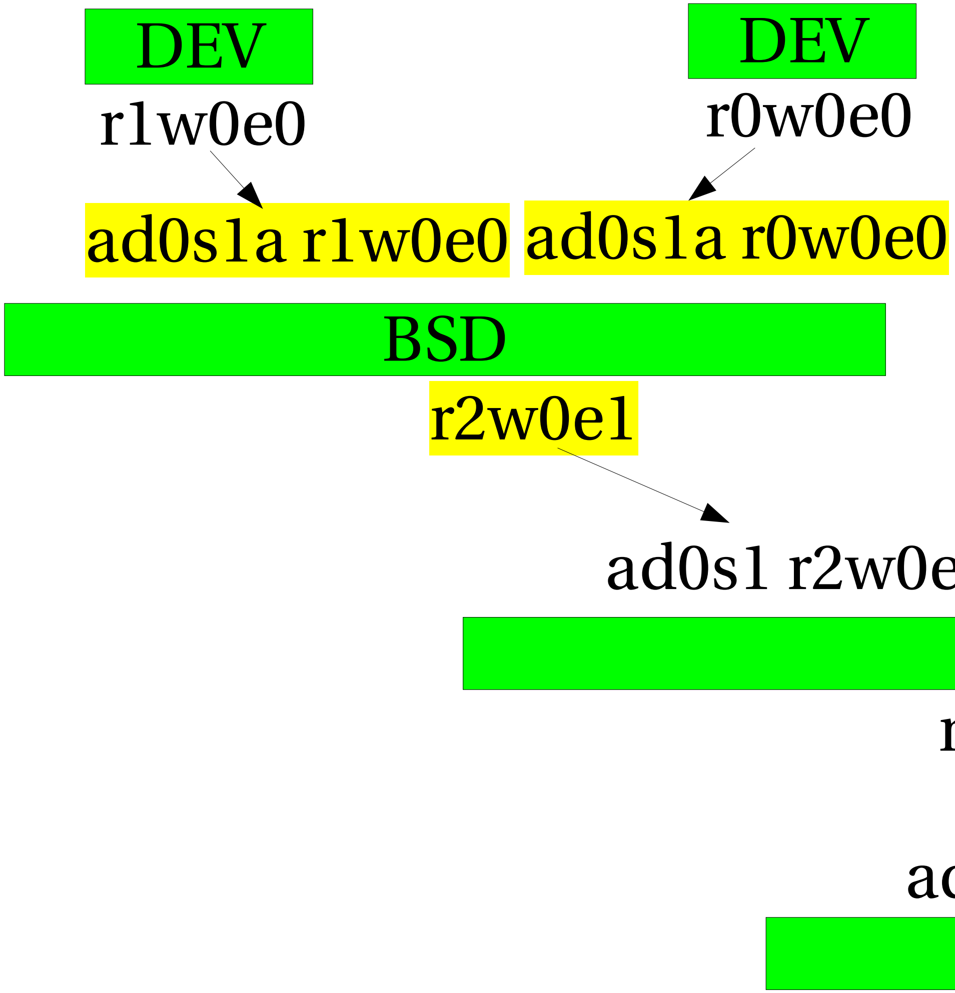
DISK

How orphaning work (10)



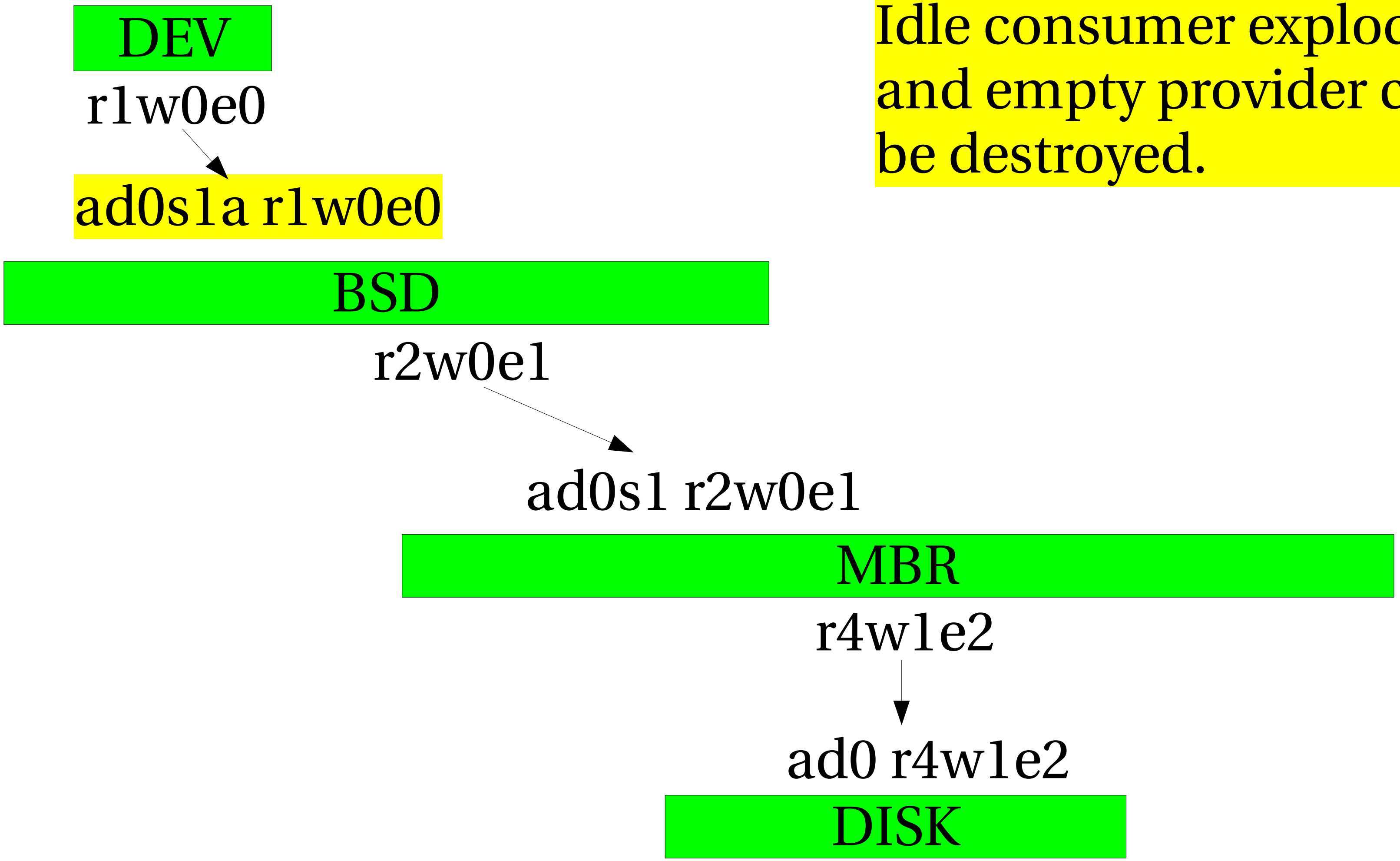
More about the DEV late

How orphaning work (11)



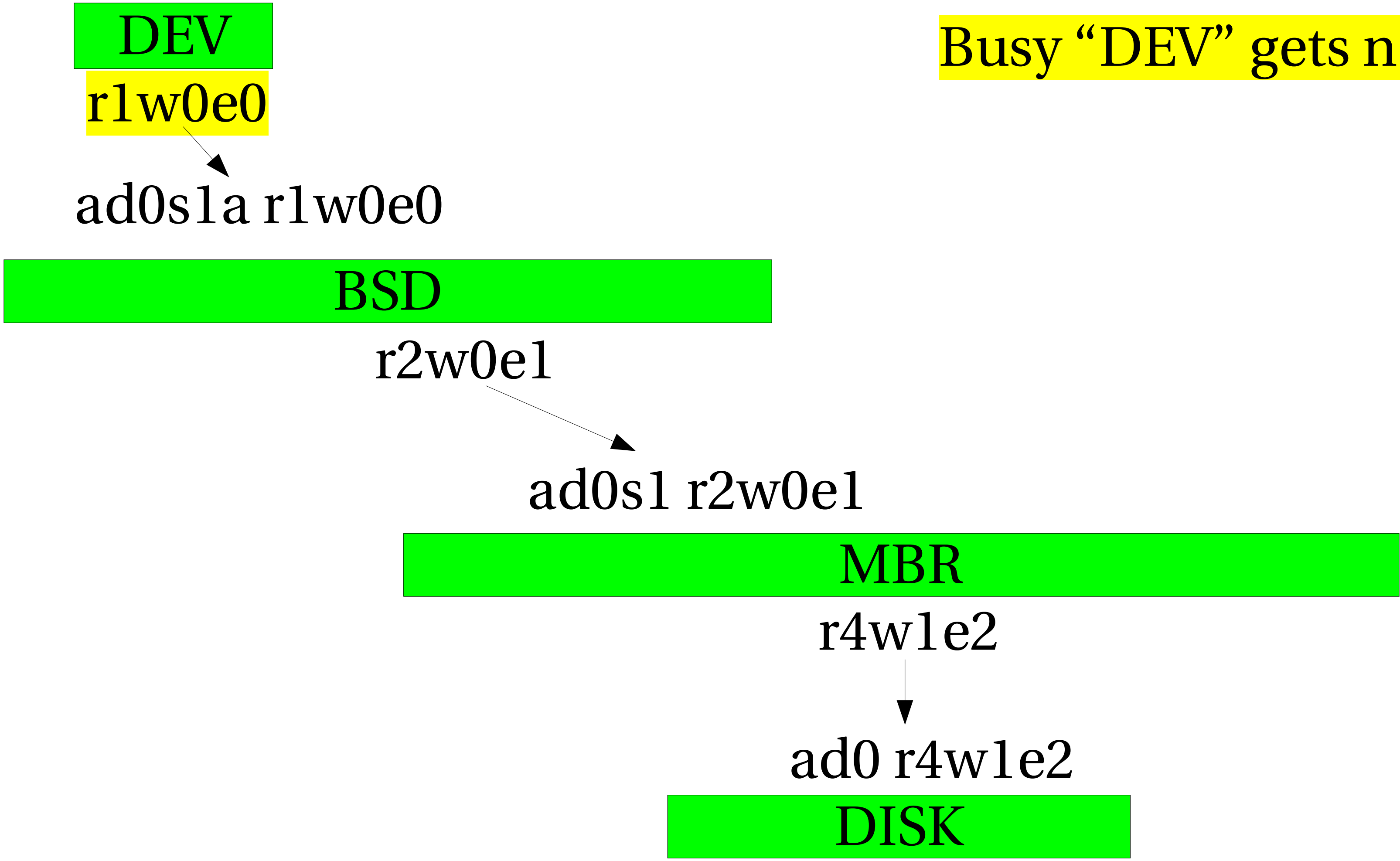
BSD geom decides to orphan its providers.

How orphaning work (12)



Idle consumer explodes and empty provider can be destroyed.

How orphaning work (13)



DEV

r1w0e0

ad0s1a r1w0e0

BSD

r2w0e1

ad0s1 r2w0e1

MBR

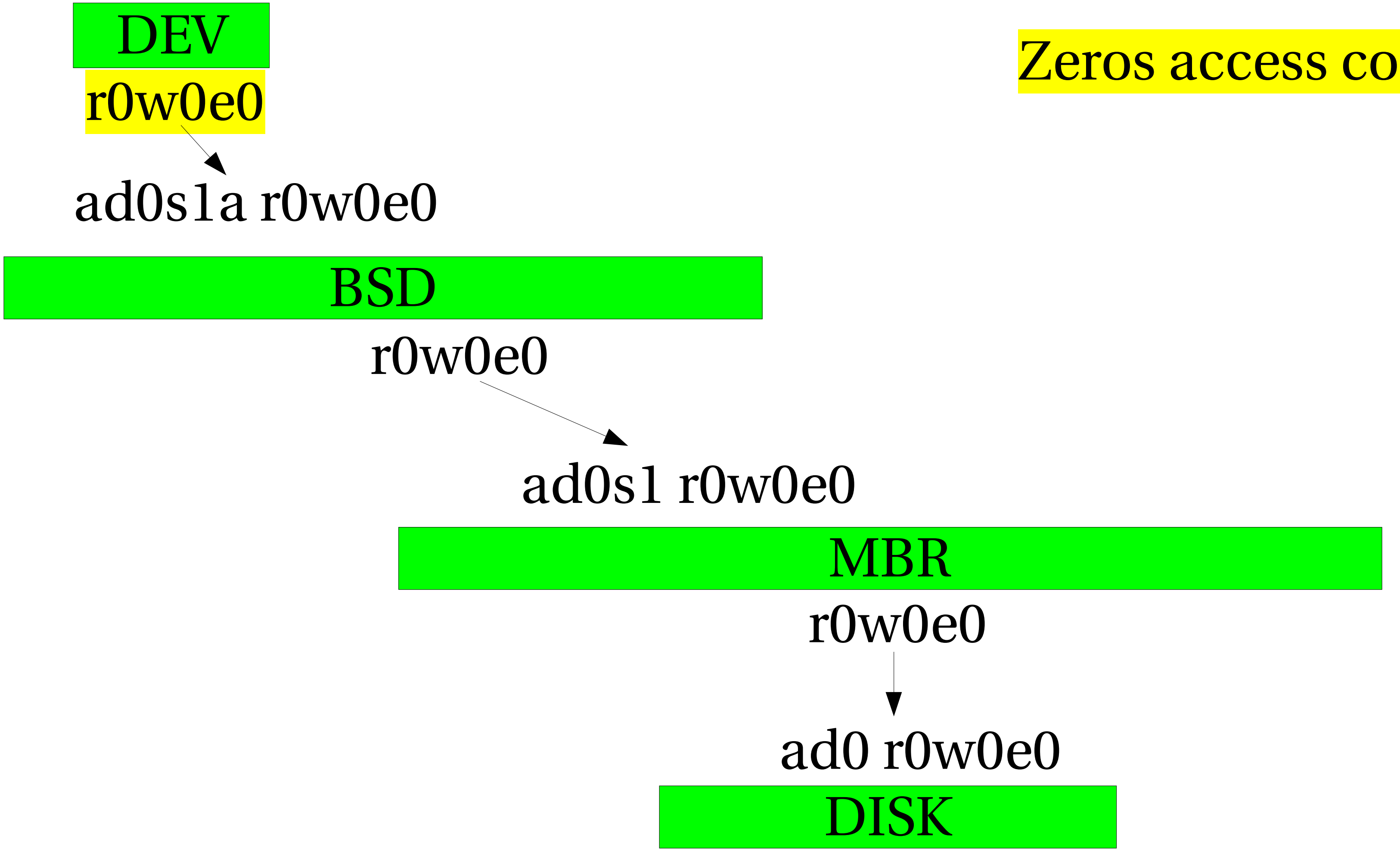
r4w1e2

ad0 r4w1e2

DISK

Busy “DEV” gets notified

How orphaning work (14)



DEV

r0w0e0

ad0s1a r0w0e0

BSD

r0w0e0

ad0s1 r0w0e0

MBR

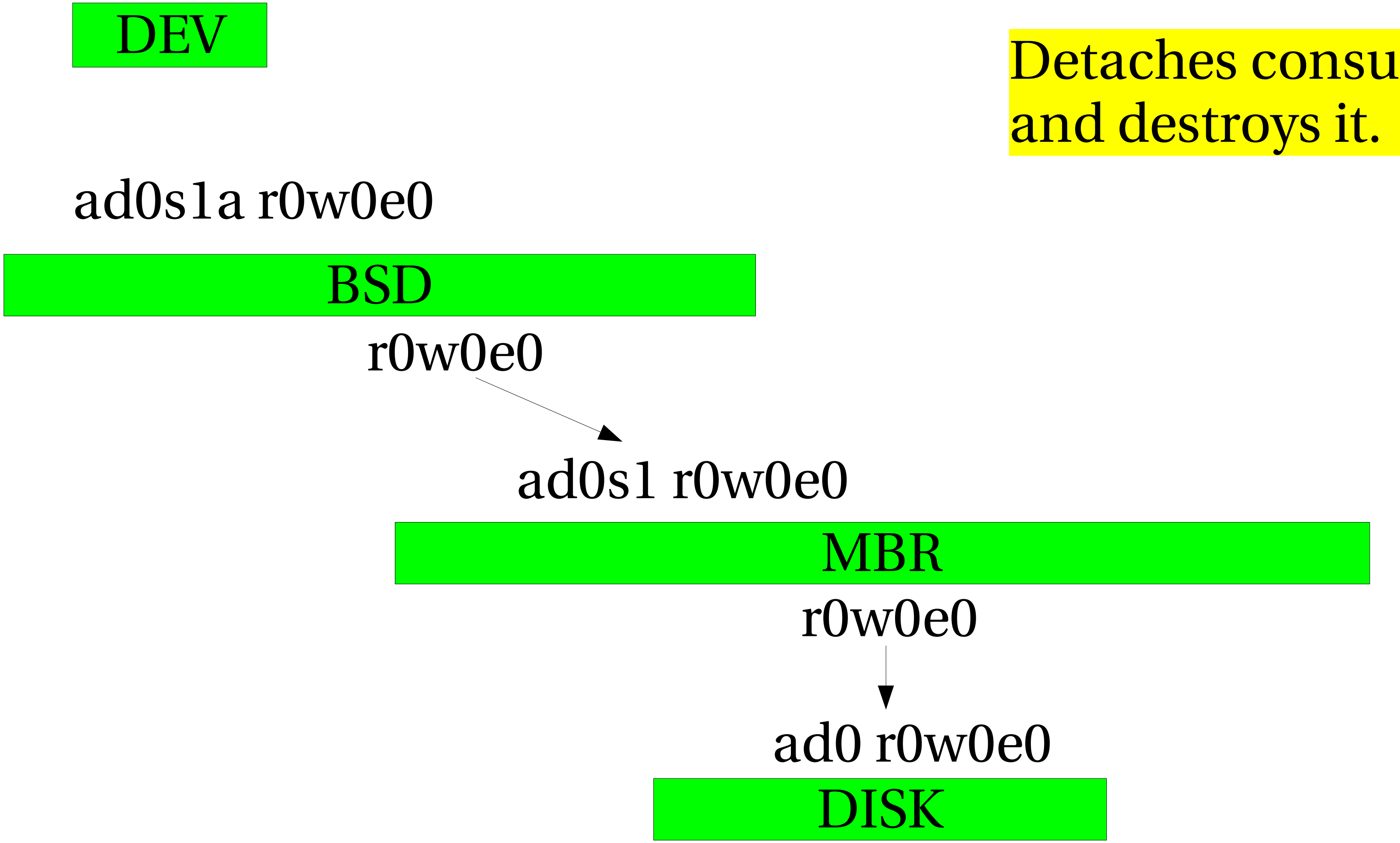
r0w0e0

ad0 r0w0e0

DISK

Zeros access count

How orphaning work (15)



DEV

ad0s1a r0w0e0

BSD

r0w0e0

ad0s1 r0w0e0

MBR

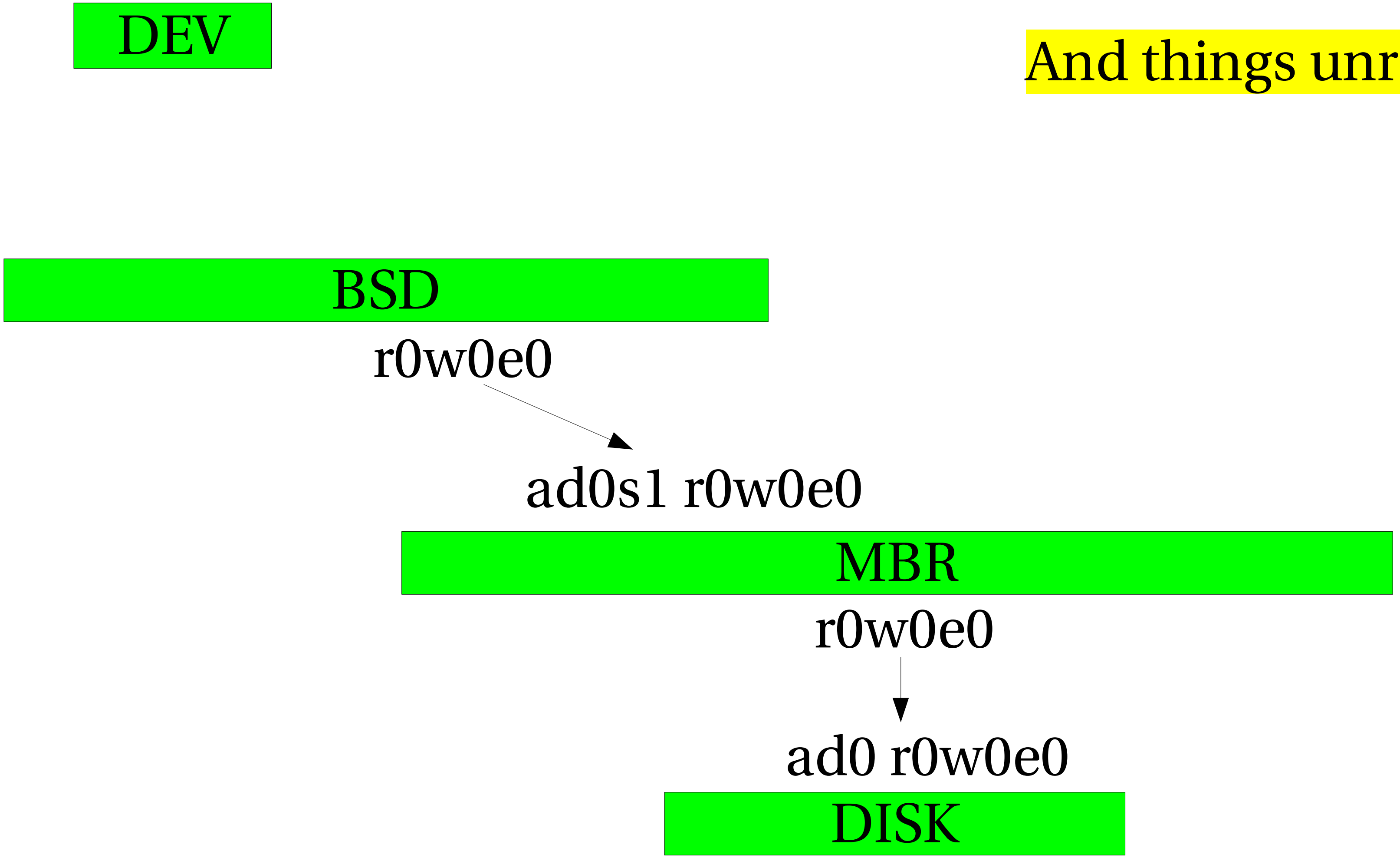
r0w0e0

ad0 r0w0e0

DISK

Detaches consumer and destroys it.

How orphaning work (16)



DEV

BSD

r0w0e0

ad0s1 r0w0e0

MBR

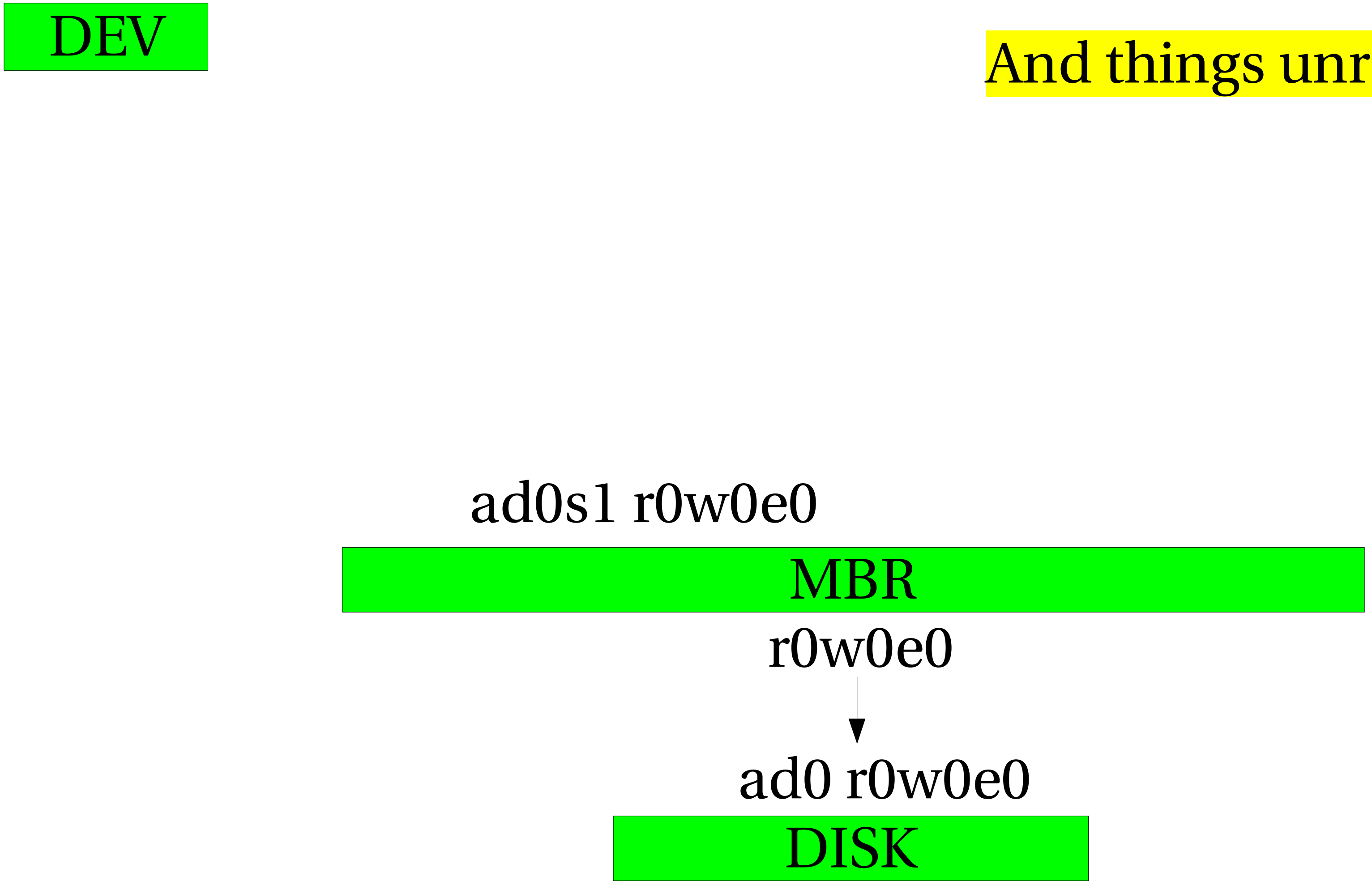
r0w0e0

ad0 r0w0e0

DISK

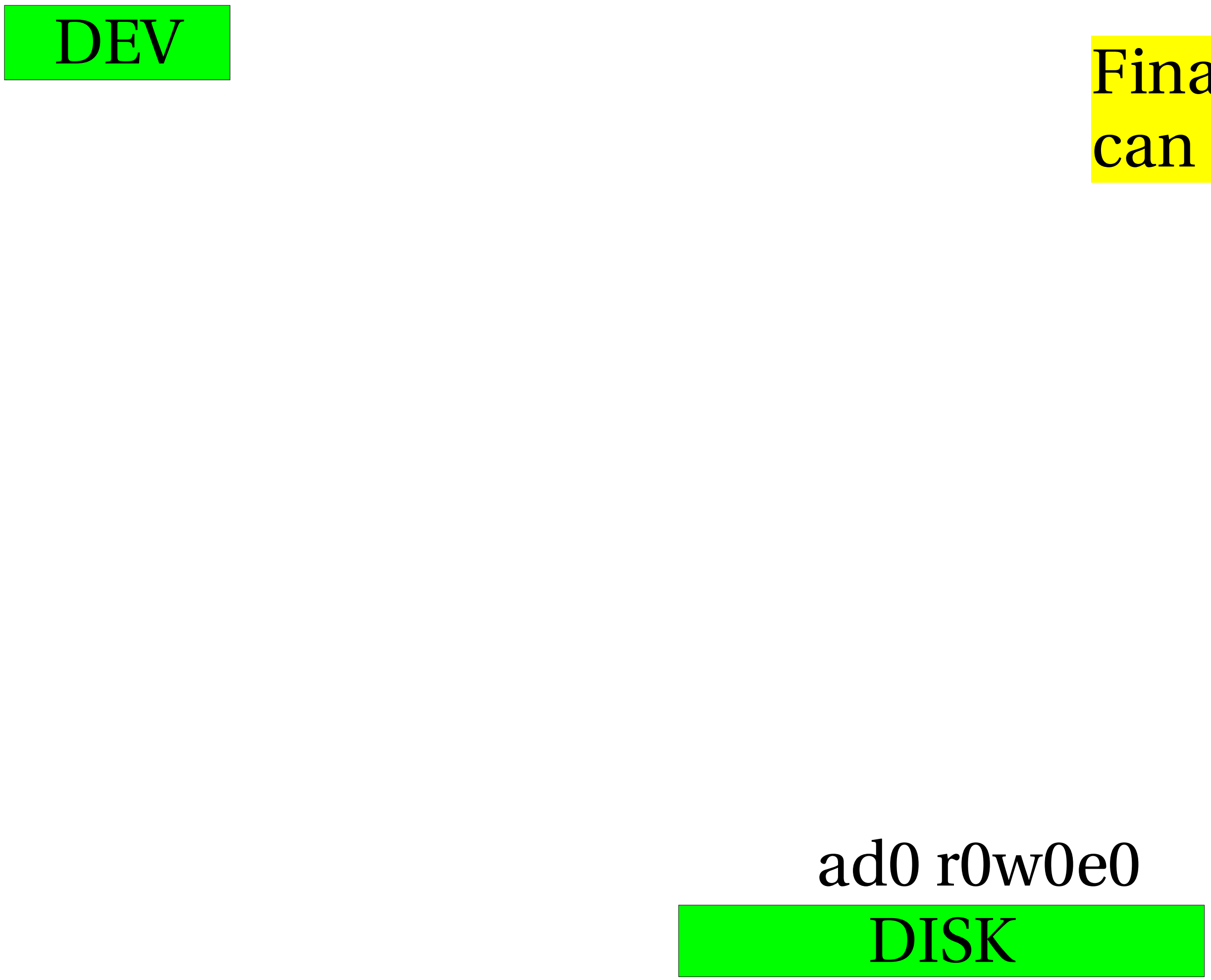
And things unravel.

How orphaning work (17)



And things unravel.

How orphaning work (18)



Finally, the provider can be destroyed.

How orphaning work (19)

DEV

The DEV class calls destroy_dev()
and properly selfdestructs.
Leaving the users to their own devices
(Sorry, couldn't resist pun)

Spoiling

- A new disk arrives: /dev/da0
- A NEW_PROVIDER event gets posted.
- All classes gets to taste the disk.
- BSD finds a disklabel and attaches.
- User does: dd if=/dev/zero of=/dev/da0
- The disklabel which configured the BSD is gone, and the BSD geom needs to know.

“Spoiled” event.

- Posted when a provider gets a non-zero write access count.
  - Can change or destroy a class' metadata.
- All attached consumers, except the guilty party, notified.

Spoiling (1)

- A class which relies on on-disk meta data will set exclusive bit if it is open in any way.
- This prevents opens which could overwrite the meta-data while it is being used.
- Does not solve the problem when the meta data is not actively being used
  - Ie: no partitions on BSD geom open.

Spoiling (2)

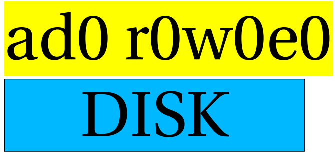
- When a provider is opened for writing first time (write access count goes non-zero):
  - Post spoil event on all attached consumers except the guilty party.
  - Consumers which rely on meta data, are obviously closed (otherwise you couldn't open for writing) and they typically self destruct.

Spoiling (3)

- When the provider is closed (ie: write access count goes to zero)
  - NEW_PROVIDER event posted on provider.
  - All classes gets chance to (re)taste and reattach.

Spoiling Cartoons

Disk device driver calls disk_create() and the DISK class creates a new geom.

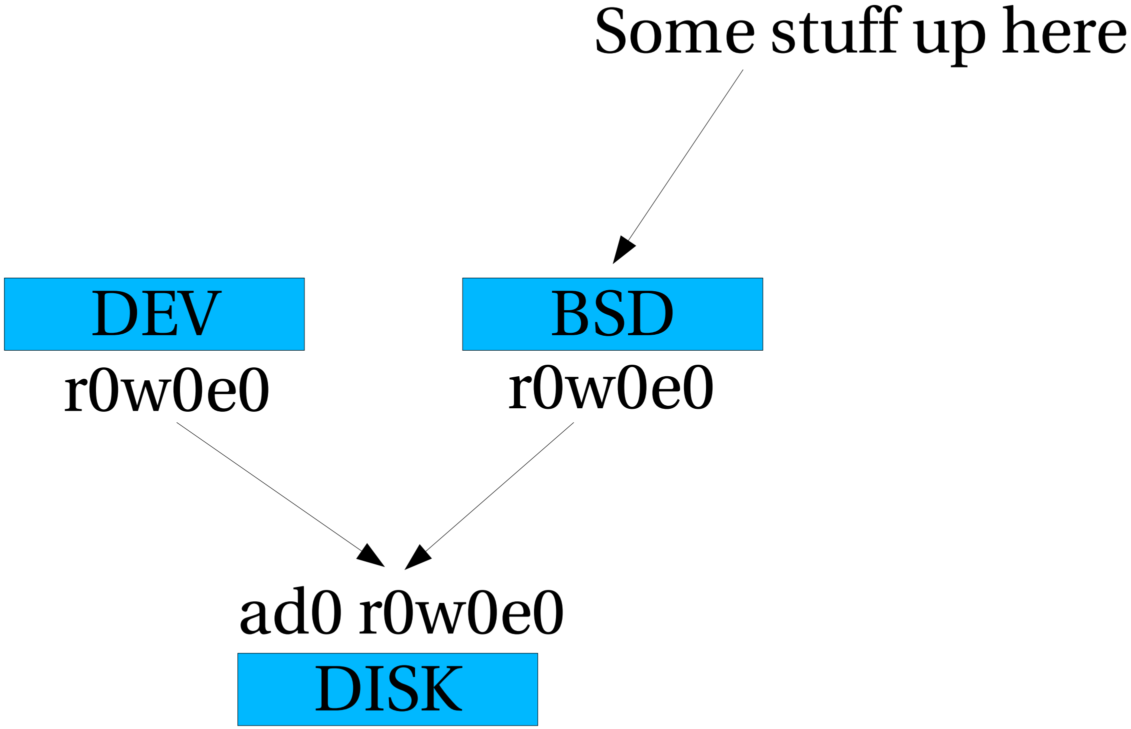


ad0 r0w0e0

DISK

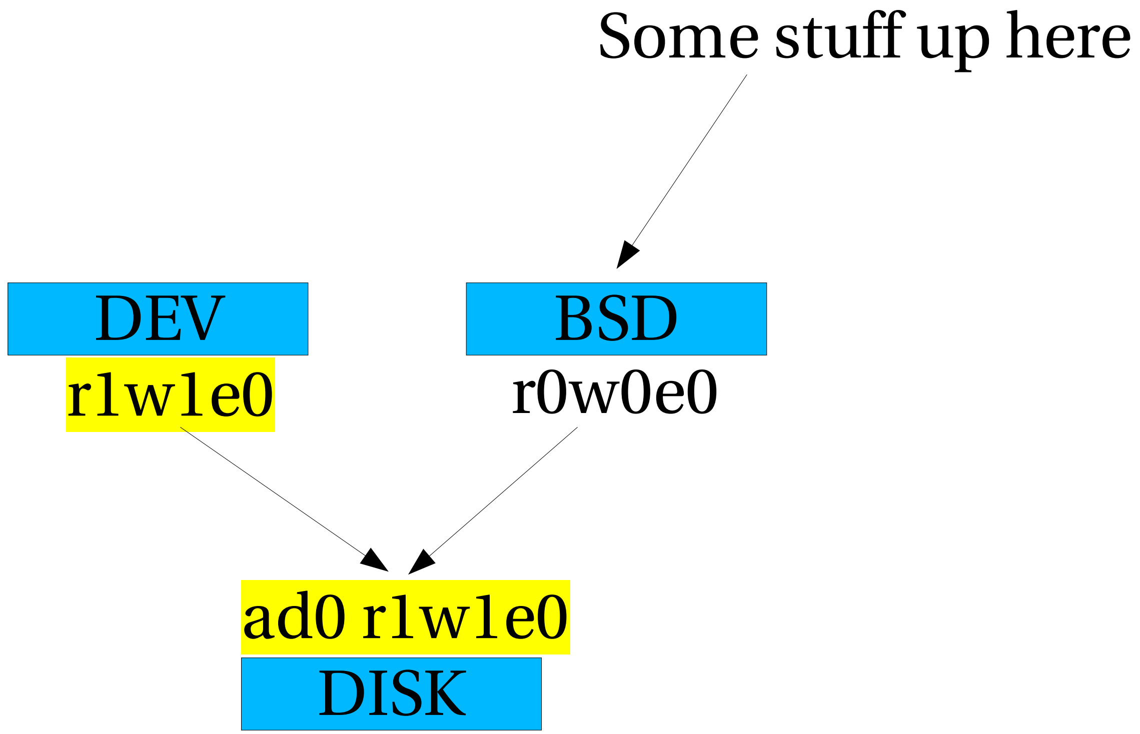
Spoiling Cartoons

NEW_PROVIDER event triggers
a round of tasting. DEV always grabs.
BSD discovers label on disk and grabs.



Spoiling Cartoons

We open /dev/ad0 for writing



Some stuff up here

DEV

r1w1e0

BSD

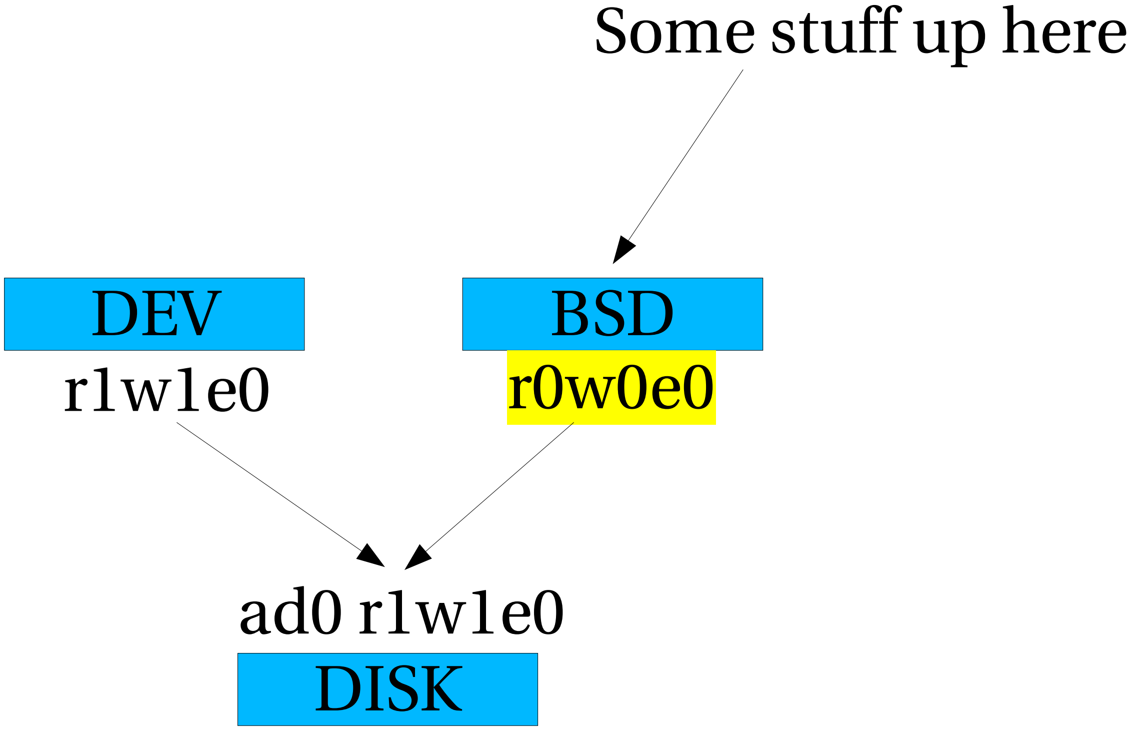
r0w0e0

ad0 r1w1e0

DISK

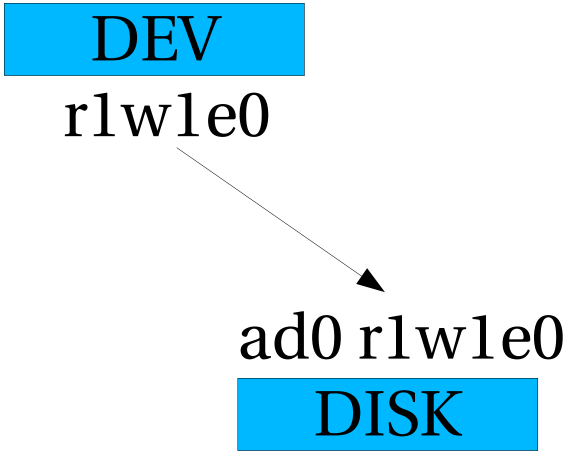
Spoiling Cartoons

write access count goes non-zero and we spoil the BSD geom.



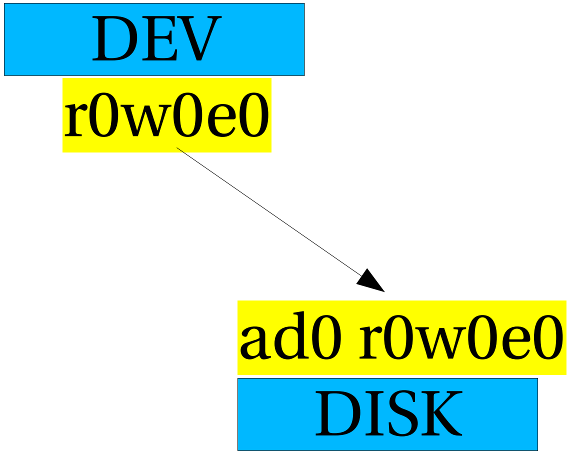
Spoiling Cartoons

BSD geom decides to self destruct.



Spoiling Cartoons

We write something to the device and the DEV is closed again.



DEV

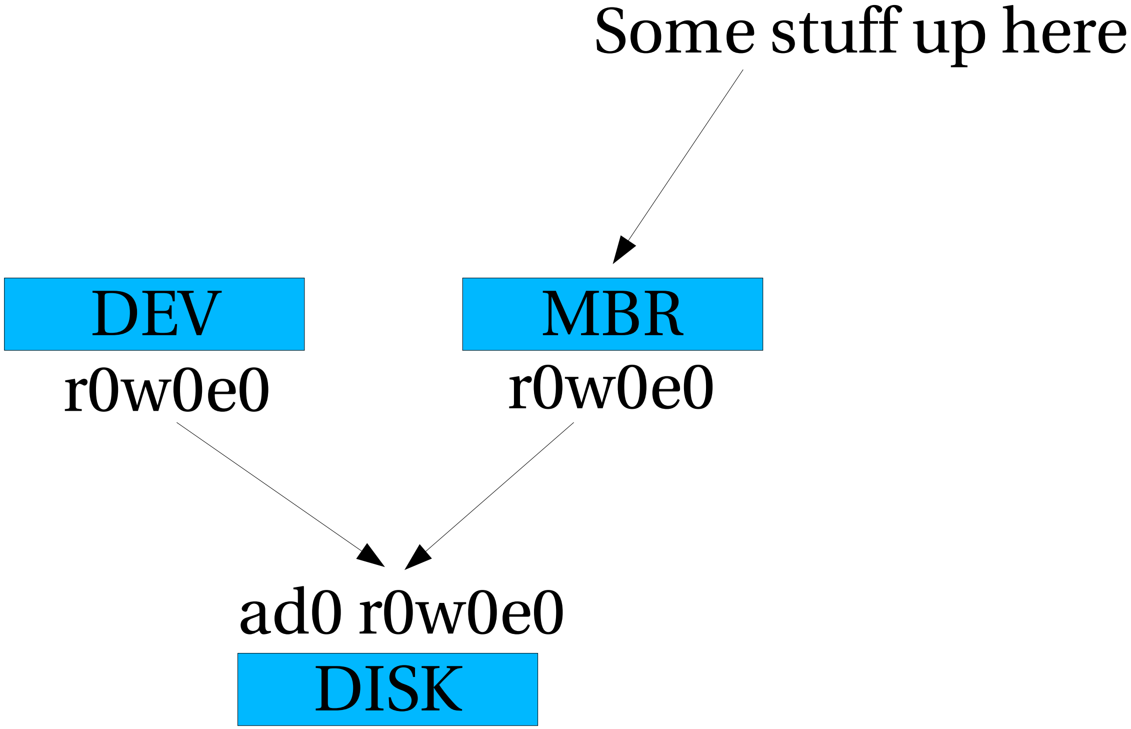
r0w0e0

ad0 r0w0e0

DISK

Spoiling Cartoons

A new round of tasting starts
And now MBR finds a label.



This is why…

- You cannot open /dev/ad0 for writing if any slices or labels are open.
- This is policy in the slicer classes, <u>not</u> in GEOM.
- Each geom/class must decide for itself how to react to spoiling.

Special GEOM classes.

- There are no special GEOM classes.

“different” GEOM classes.

- All GEOM classes are treated the same.
- … But not all GEOM classes have the same kind of job.
  - “DISK” class talks to disk device drivers.
    - disk_create(), disk_destroy() etc.
  - “DEV” class talks to dev_t/SPECFS/DEVFS.
    - make_dev(), destroy_dev() etc.

The DISK geom class.

- Upper side interface: GEOM
- Lower side interface: “disk minilayer”
  - disk_create().
    - Do magic necessary for disk device-driver.
    - Create a provider.
  - disk_destroy().
    - Orphan provider.
    - Do various magic for the disk device-driver.
    - Self-destruct when possible.

The DEV geom class.

- Lower side interface: geom consumer.
  - Attaches to anything taste presents to it.
- Upper side: disk device-driver.
  - Calls make_dev() with suitable args.
- When Orphaned:
  - Calls destroy_dev()
  - Selfdestructs.

Would it be possible…

- To write a GEOM class to sit on top of the network ?
- To give disk device drivers a native GEOM interface instead of using the DISK class ?
- To … ?
- YES, Geom classes are very very general.

“Slicers” as a concept

- “Slicers” are GEOM classes which partition a device into some number of sub devices.
- Commonality includes:
  - Transformation consists of offset + limit.
  - Refuse overlapping slices from opening.
  - On-the-fly change of slice configuration.

Trying to raise the bar…

- Use explicit byte-stream decode for on-disk meta data.
  - This gives the geom modules wordsize and endianess agility.
- Put i386 disk in sparc64 and access the partitions.
- Not really that useful until file systems are agile as well.

So what does a slicer take ?

- Three (or Four) “hard” routines:
  - “modify”
    - Take label image, validate, configure.
  - “taste”
    - Read label image from disk
  - “config”
    - Receive label image from userland.
  - “hotwrite”
    - Intercept label image overwrites.

Management interface(s).

- GEOM needs to be able to report config to userland.
- Since we don't know what the classes are and what they can do, we cannot know what they would like to report.
- => use extensible format.

XML in the KERNEL ???

- No, “XML out of the kernel”.
- There is no point in inventing my own hierarchal extensible modular format when there is one with a lot of tools and growing recognition already.
- Generating XML in the kernel is simple:
  - sbufs - string buffers with memory management.
  - sprintf.

Sample XML output

```
critter phk> sysctl -b kern.geom.confxml | head -20
<mesh>
  <class id="0xc03b1200">
    <name>MBREXT</name>
  </class>
  <class id="0xc03b11a0">
    <name>MBR</name>
    <geom id="0xc4042f40">
      <class ref="0xc03b11a0"/>
      <name>ad0</name>
      <rank>2</rank>
      <config>
      </config>
        <consumer id="0xc406b000">
          <geom ref="0xc4042f40"/>
          <provider ref="0xc4148980"/>
          <mode>r8w8e3</mode>
          <config>
          </config>
        </consumer>
        <provider id="0xc4148800">
```

Generating XML from a class

- Class implementes “dumpconf” method
- Appends text into provided sbuf.
- Gets called per instance of a class:
    - Once with geom argument only.
    - For every provider with geom & provider arg.
    - For every consumer with geom & consumer arg.

Sample dumpconf method

```
void
g_slice_dumpconf(struct sbuf *sb, const char *indent,
    struct g_geom *gp, struct g_consumer *cp, struct g_provider *pp)
{
        struct g_slicer *gsp;

        gsp = gp->softc;

        if (pp != NULL) {
                sbuf_printf(sb, "%s<index>%u</index>\n", indent, pp->index);
                sbuf_printf(sb, "%s<length>%ju</length>\n",
                    indent, (uintmax_t)gsp->slices[pp->index].length);
                sbuf_printf(sb, "%s<seclength>%ju</seclength>\n", indent,
                    (uintmax_t)gsp->slices[pp->index].length / 512);
                sbuf_printf(sb, "%s<offset>%ju</offset>\n", indent,
                    (uintmax_t)gsp->slices[pp->index].offset);
                sbuf_printf(sb, "%s<secoffset>%ju</secoffset>\n", indent,
                    (uintmax_t)gsp->slices[pp->index].offset / 512);
        }
}
```

Sample class output

```
<provider id="0xc4148800">
  <geom ref="0xc4042f40"/>
  <mode>r8w8e2</mode>
  <name>ad0s1</name>
  <mediasize>40007729664</mediasize>
  <sectorsize>512</sectorsize>
  <config>
    <index>0</index>
    <length>40007729664</length>
    <seclength>78140097</seclength>
    <offset>32256</offset>
    <secoffset>63</secoffset>
    <type>165</type>
  </config>
</provider>
```

Reading XML from userland

- /usr/src/lib/libexpat
  - Snapshot version of Expat XML library.
- /usr/src/lib/libgeom
  - Contains handy “xml2tree” function which builds c-struct representation.

User instruction channel.

- /dev/geom.ctl
  - Prefer device over sysctl because it offers access control mechanisms people can understand.
  - Unified command interface.

GEOMs OAM api

- “gctl” api in libgeom used to send requests to GEOM classes.
- A request holds any number of parameters, read/only or read/write.
- Error reporting in string form
  - Many error situations are too complex to express with numeric error codes, for some reason I just don't think we can live with ECPARTITIONOVERLAPSOPENPARTITION

OAM…

- Accumulative error handling
  - Only need to check error at the very end.
- Please use of text for information
  - Makes it possible to have portable, extensible admin tools learn about a new class.
- Not intended for high frequency use.

Gctl_*()

```
H = gctl_get_handle();
gctl_ro_param(H, “verb”, -1, “destroy geom”);
gctl_ro_param(H, “class”, -1, “CCD”);
sprintf(buf, “ccd%d”, ccd);
gctl_ro_param(H, “geom”, -1, buf);
errstr = gctl_issue(H);
if (errstr != NULL)
    err(1, “Could not destroy ccd:%s”, errstr);
```

Receivng gctl_ requests

```
static void
g_ccd_create(struct gctl_req *req, struct g_class *mp)
{
        int *unit, *ileave, *nprovider;
        struct provider *pp
        [...]

        g_topology_assert();
        unit = gctl_get_paraml(req, "unit", sizeof (*unit));
        ileave = gctl_get_paraml(req, "ileave", sizeof (*ileave));
        nprovider = gctl_get_paraml(req, "nprovider", sizeof (*nprovider));
        [...]
        /* Check all providers are valid */
        for (i = 0; i < *nprovider; i++) {
                sprintf(buf, "provider%d", i);
                pp = gctl_get_provider(req, buf);
                if (pp == NULL)
                        return;
        }
```

Exporting statistics

- Performance statistics are collected on all consumers and all providers.
- Uses updated libdevstat library
  - Export info with shared memory
    - Very fast, <1msec update rates possible.
  - Now also contains info on response time.
- The gstat(8) program presents statistics in curses window.

Gstat(8)

```
DT: 0.510  flag_I 500000us  sizeof 240  i -1
 L(q)  ops/s    r/s   kBps   ms/r    w/s   kBps   ms/w    %busy Name
    1     75     75    149    6.8      0      0    0.0   50.6| ad0
    1     75     75    149    6.8      0      0    0.0   51.0| ad0s1
    0      0      0      0    0.0      0      0    0.0    0.0| ad0s1a
    0      0      0      0    0.0      0      0    0.0    0.0| ad0s1b
    0      0      0      0    0.0      0      0    0.0    0.0| ad0s1c
    0      0      0      0    0.0      0      0    0.0    0.0| ad0s1d
    0      0      0      0    0.0      0      0    0.0    0.0| ad0s1e
    0      0      0      0    0.0      0      0    0.0    0.0| ad0s1f
    1     75     75    149    6.9      0      0    0.0   51.4| ad0s1g
    0      0      0      0    0.0      0      0    0.0    0.0| ad0s1h
    0      0      0      0    0.0      0      0    0.0    0.0| ad0s1f.bde
```

L(q) = length of queue
ops/s, r/s, w/s = operations, reads and writes per second
kBps = kiloBytes per second
ms/r, ms/w = milliseconds per read and write
%busy = % of time with at least one entry in queue

Using events

- Says “Please call me from the event queue”.
- Use this for doing things which would sleep in the up/down I/O path.
  - Typically if you need the topology lock.
- Or for Giant isolation.

Debugging GEOM

- Use the XML info
  - Contains everything you may need to know.
- Use the regression tests
  - /usr/src/tools/regression/geom
- Undocumented debugging tools:
  - sysctl -b kern.geom.confdot | dot -Tps > _.ps
  - gv _.ps

Debugging GEOM

- sysctl kern.geom.debugflags=N
  - N = 1
    - Traces topology related stuff
  - N=2
    - Traces individual I/O requests (very noisy!)
  - N=4
    - Traces access count related issues.
  - N=8
    - Enable sanity checks on topology tree.

What then is GEOM ?

- GEOM is an entirely new way to think about disk-like storage I/O requests.
- GEOM is very very very general compared to what we had before.
  - New possibilities.
  - New problems.
    - What if two providers both want to be “ad0s1” ?

The End.

- A big thanks to:
  - Robert Watson for finding, taming milking and keeping the paper tiger on its diet.
  - DARPA/SPAWAR for sponsoring this work under contract N66001-01-C-8035 ("CBOSS"), as part of the DARPA CHATS research program.
  - All the giants whose shoulders we stand on.
  - FreeBSD developers and users for putting up with me.