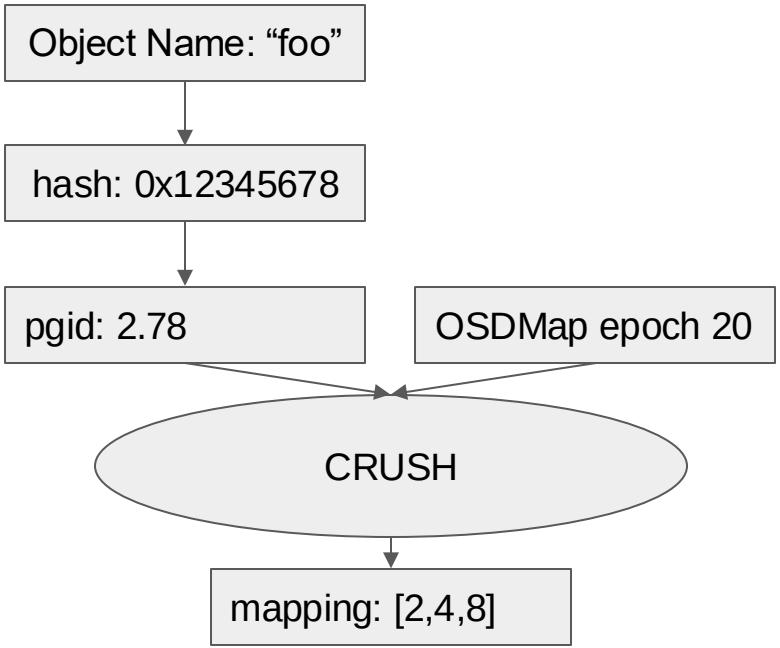
CRUSH MSR

CRUSH


Object Name: “foo”
hash: 0x12345678
pgid: 2.78
OSDMap epoch 20
CRUSH
mapping: [2,4,8]

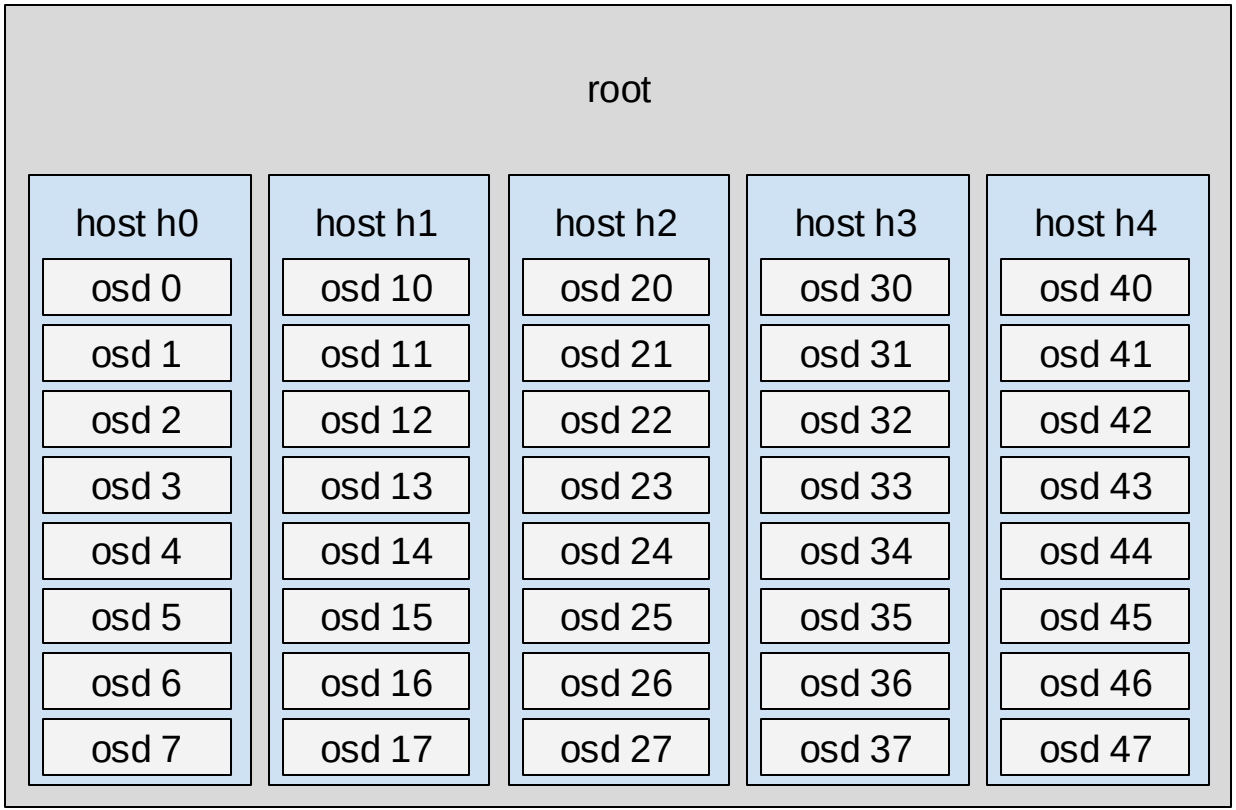
CRUSH

- Deterministic: given the same OSDMap and pgid, CRUSH will output the same set of OSDs
- Balance: Generated mappings will (with some limitations) reflect the OSD weightings
- Stability: Small changes in the map need to result in a proportionally small number of mapping changes.

Basics: Split 3 OSDs across Hosts

```
rule replicated_rule_1 {
    ...
    step take default class ssd
    step chooseleaf firstn 3 type host
    step emit
}
```

Basics: Split 3 OSDs across Hosts

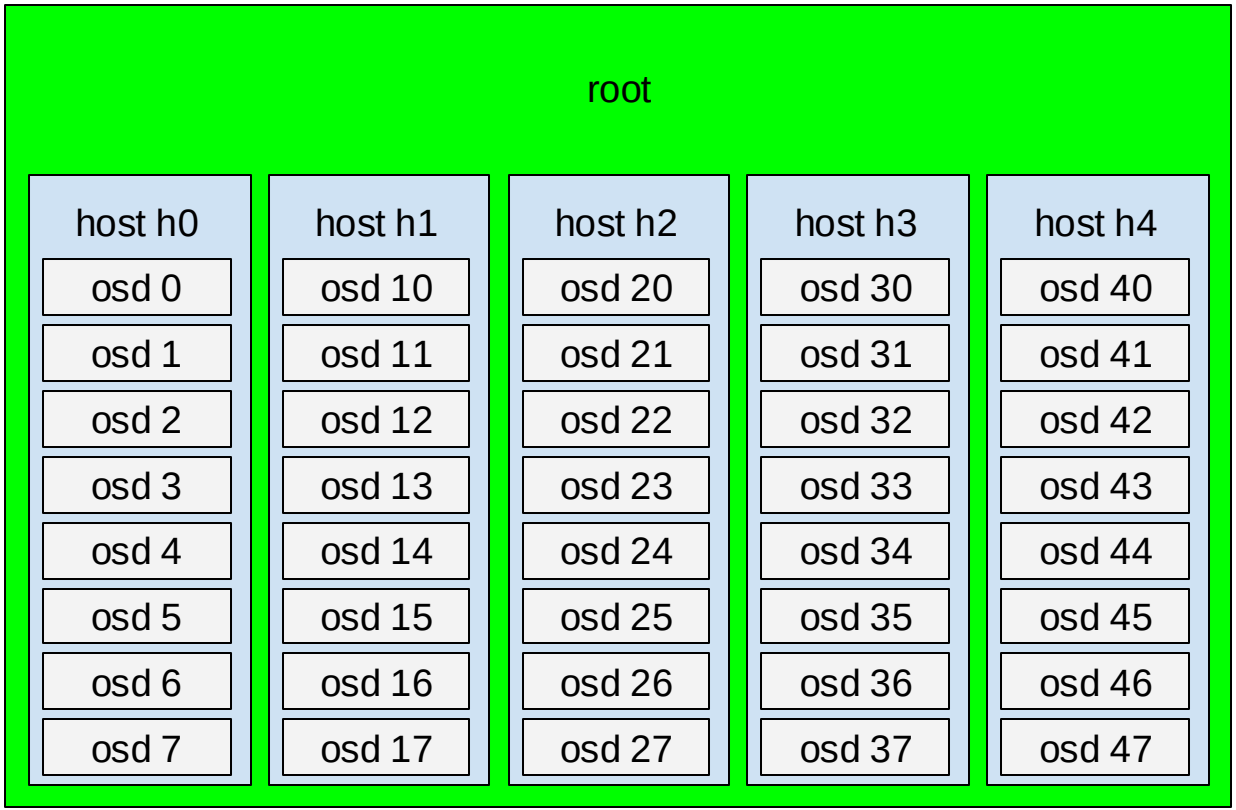


Basics: Split 3 OSDs across Hosts

in: []

step take default class ssd

out: [root]



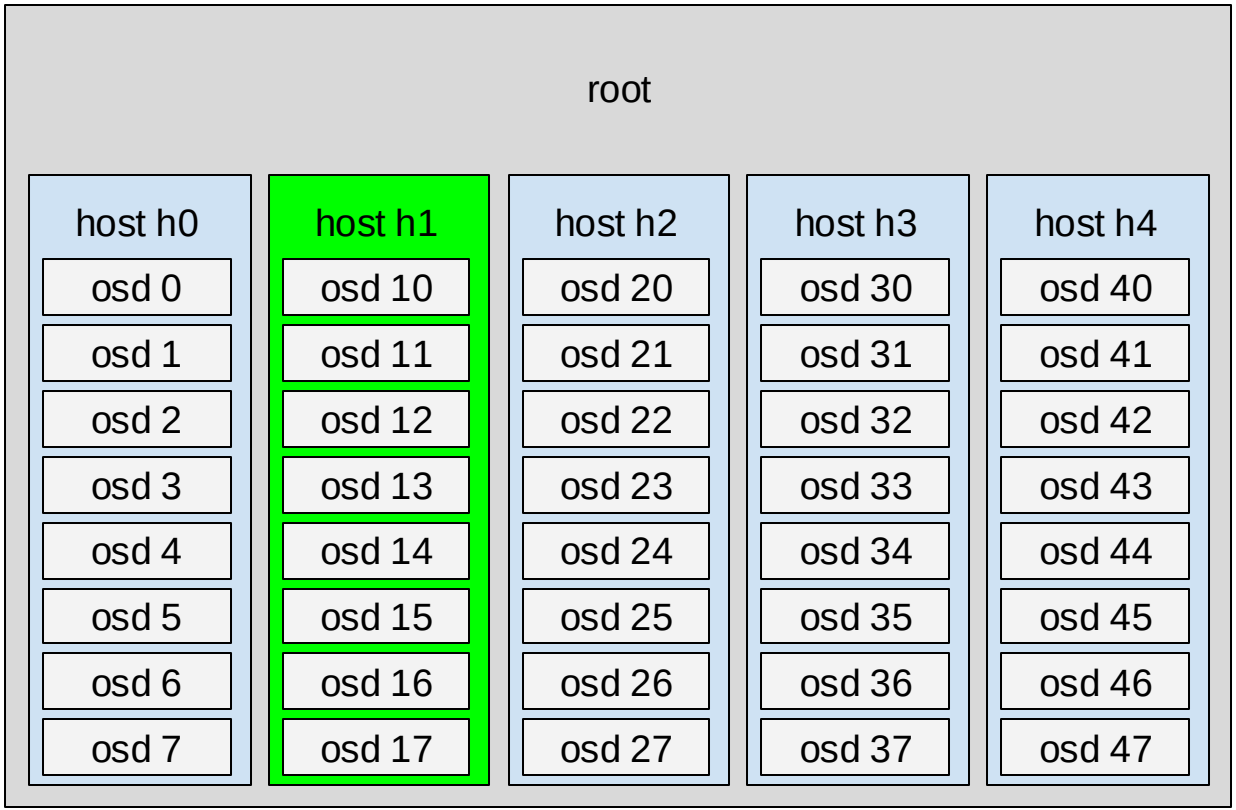
Basics: Split 3 OSDs across Hosts

in: [root]

step chooseleaf firstn 3 type host

out: [h1]

out2: []



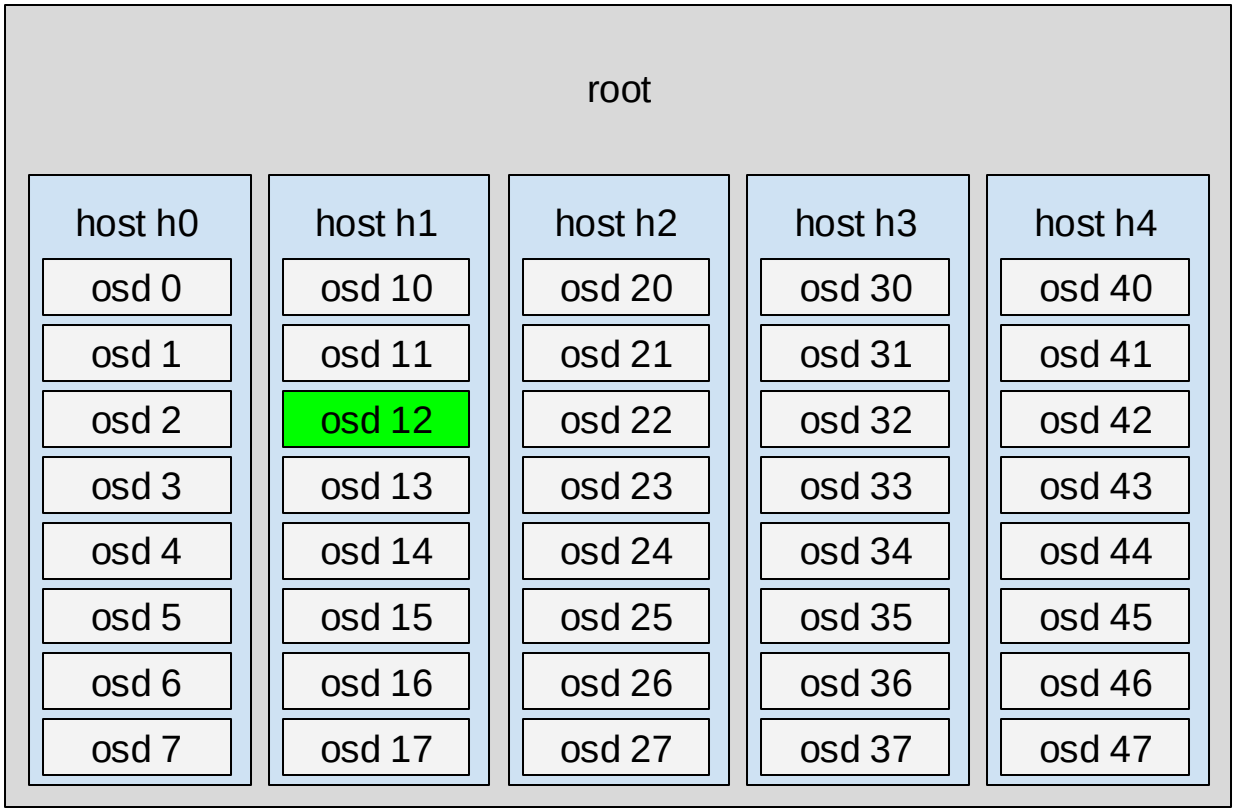
Basics: Split 3 OSDs across Hosts

in: [root]

step chooseleaf firstn 3 type host

out: [h1]

out2: [12]



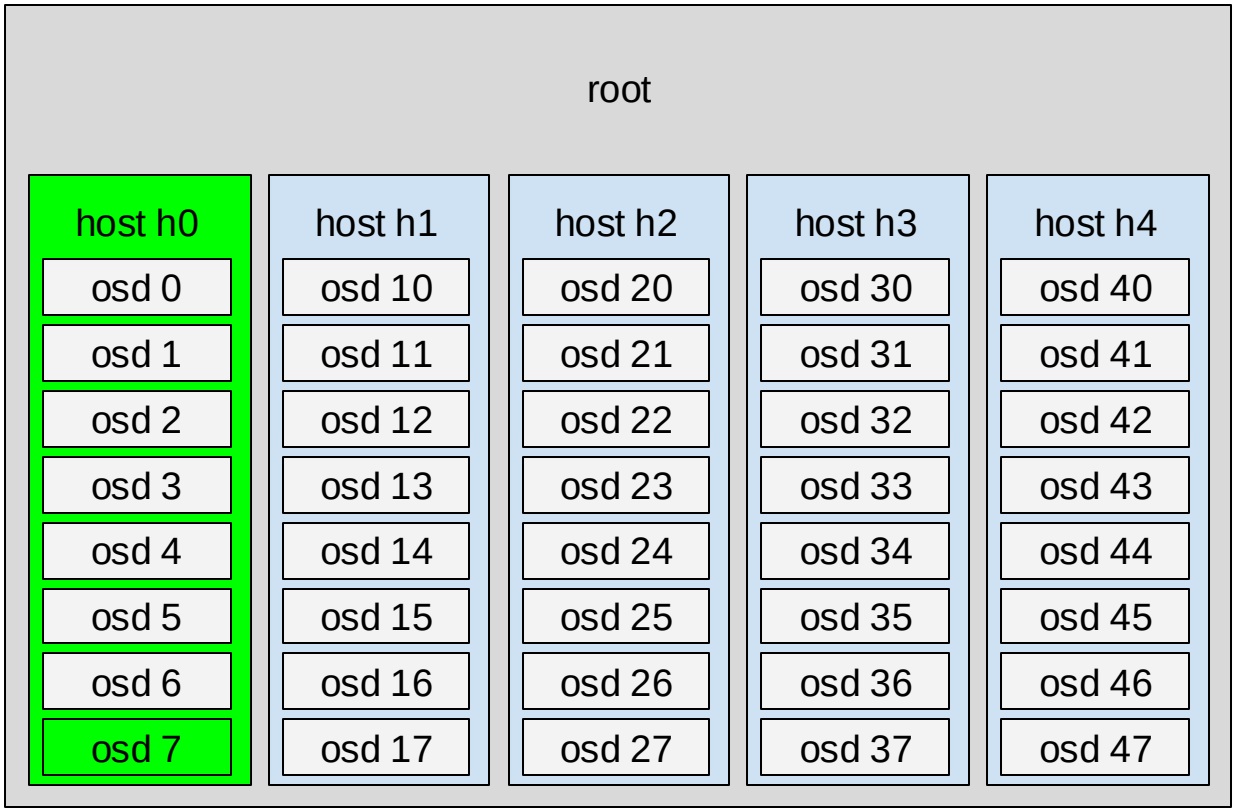
Basics: Split 3 OSDs across Hosts

in: [root]

step chooseleaf firstn 3 type host

out: [h1, h0]

out2: [12, 7]



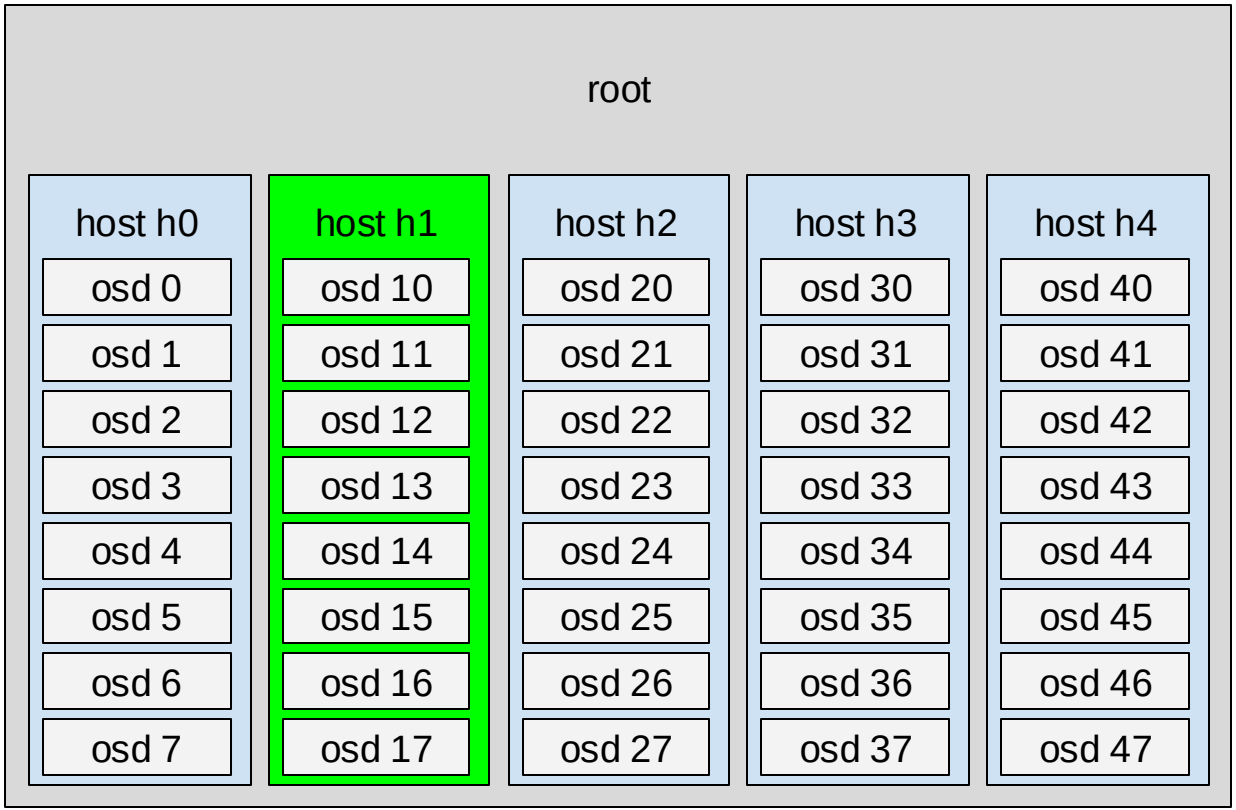
Basics: Split 3 OSDs across Hosts

in: [root]

step chooseleaf firstn 3 type host

out: [h1, h0]

out2: [12, 7]



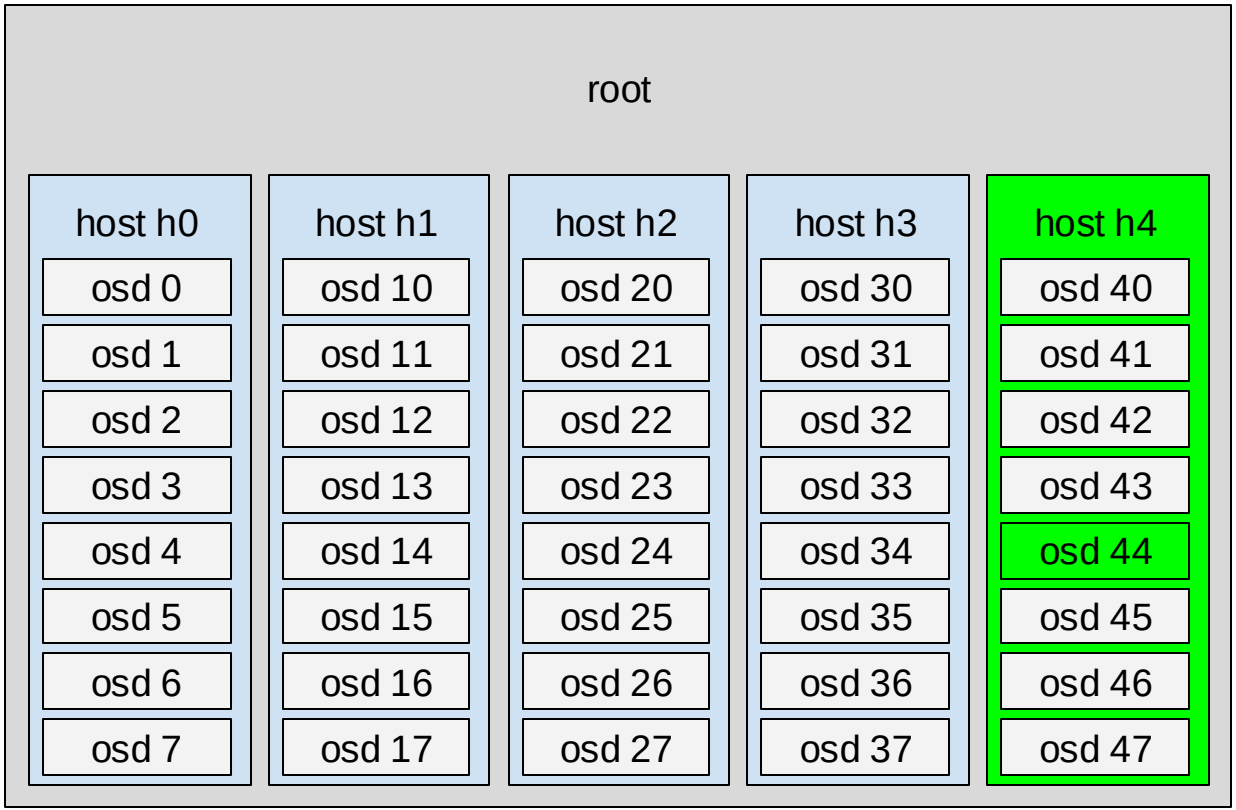
Basics: Split 3 OSDs across Hosts

in: [root]

step chooseleaf firstn 3 type host

out: [h1, h0, h4]

out2: [12, 7, 44]



Why chooseleaf?

```
rule replicated_rule_1 {
    ...
    step take default class ssd
    step chooseleaf firstn 3 type host
    step emit
}
```



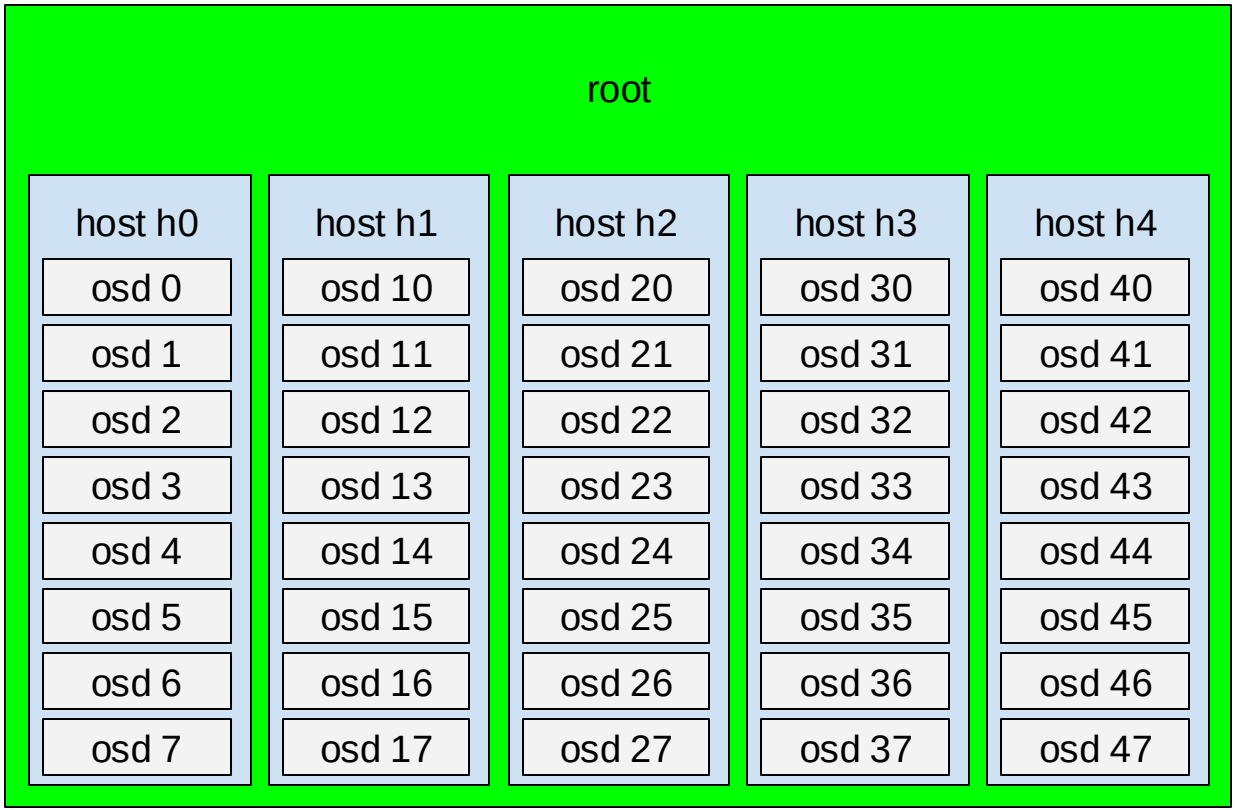
```
rule replicated_rule_1 {
    ...
    step take default class ssd
    step choose firstn 3 type host
    step choose firstn 1 type osd
    step emit
}
```

Why chooseleaf?

in: []

step take default class ssd

out: [root]

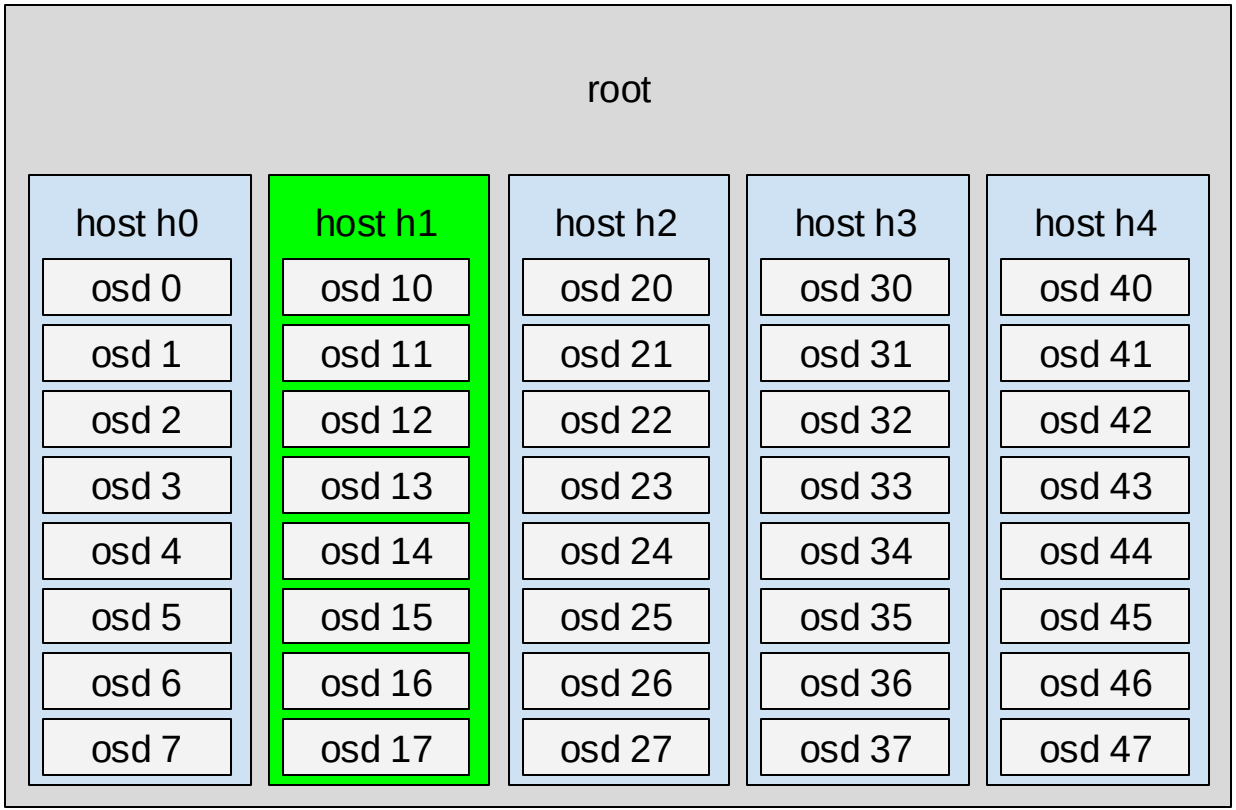


Why chooseleaf?

in: [root]

step choose firstn 3 type host

out: [h1]

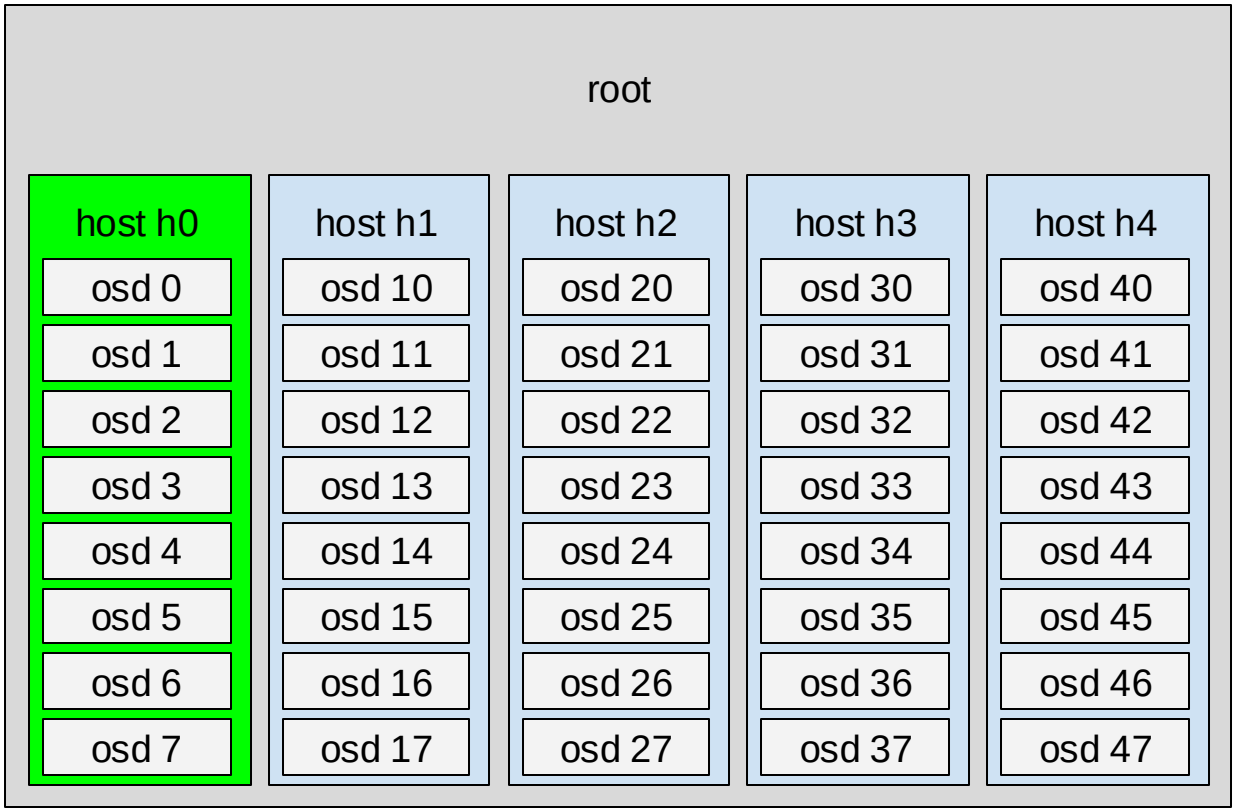


Why chooseleaf?

in: [root]

step choose firstn 3 type host

out: [h1, h0]

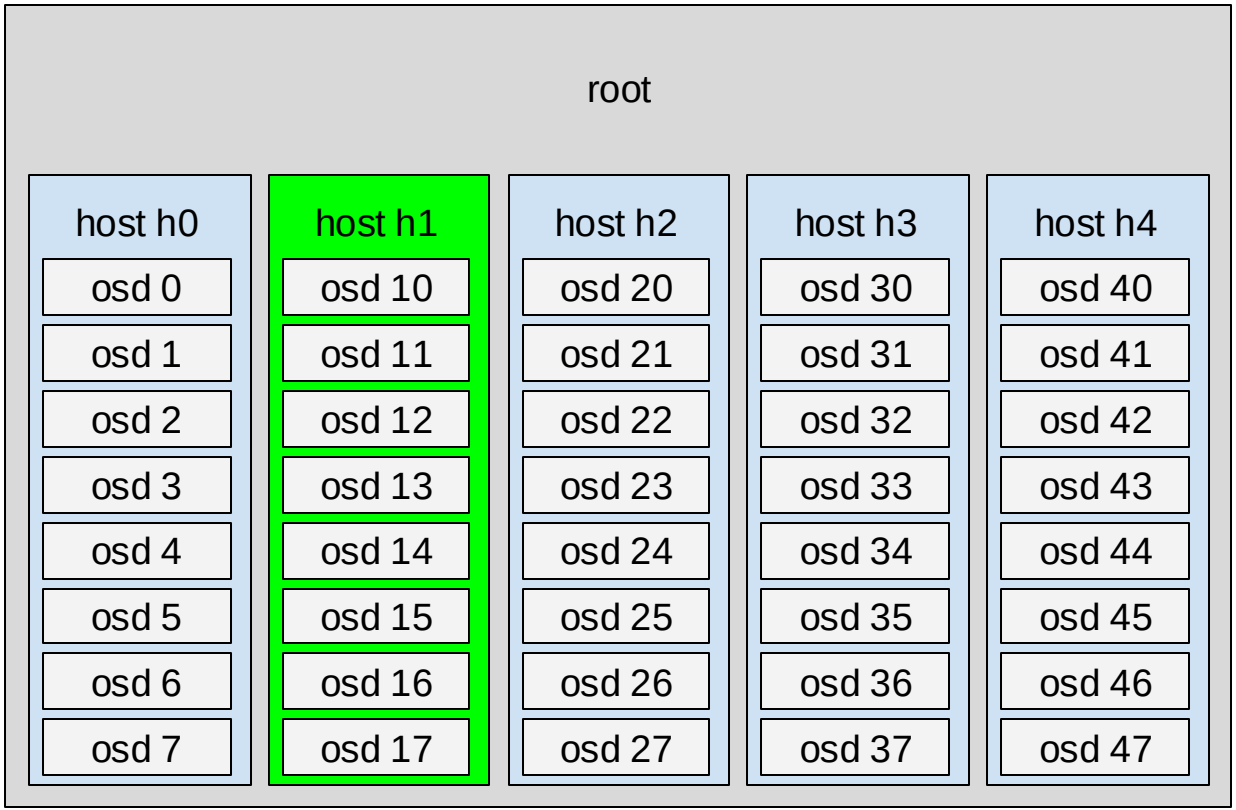


Why chooseleaf?

in: [root]

step choose firstn 3 type host

out: [h1, h0]

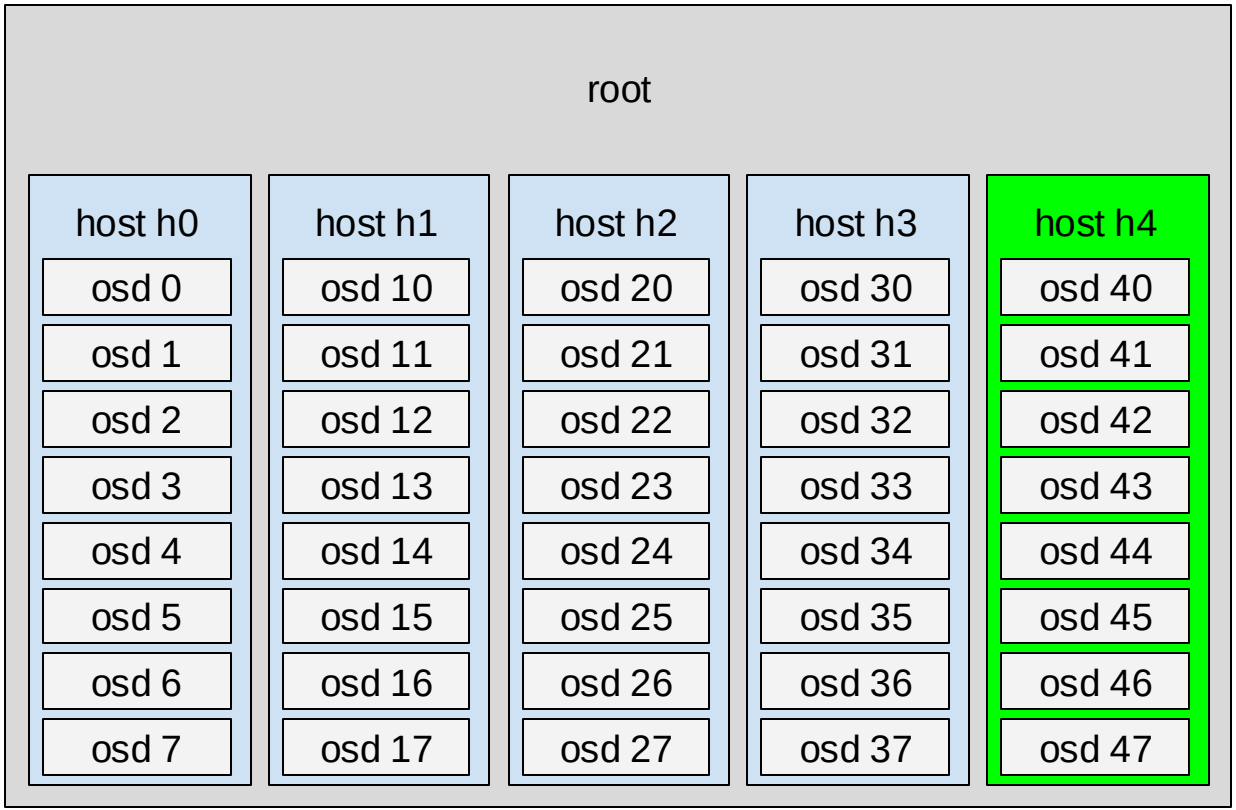


Why chooseleaf?

in: [root]

step choose firstn 3 type host

out: [h1, h0, h4]

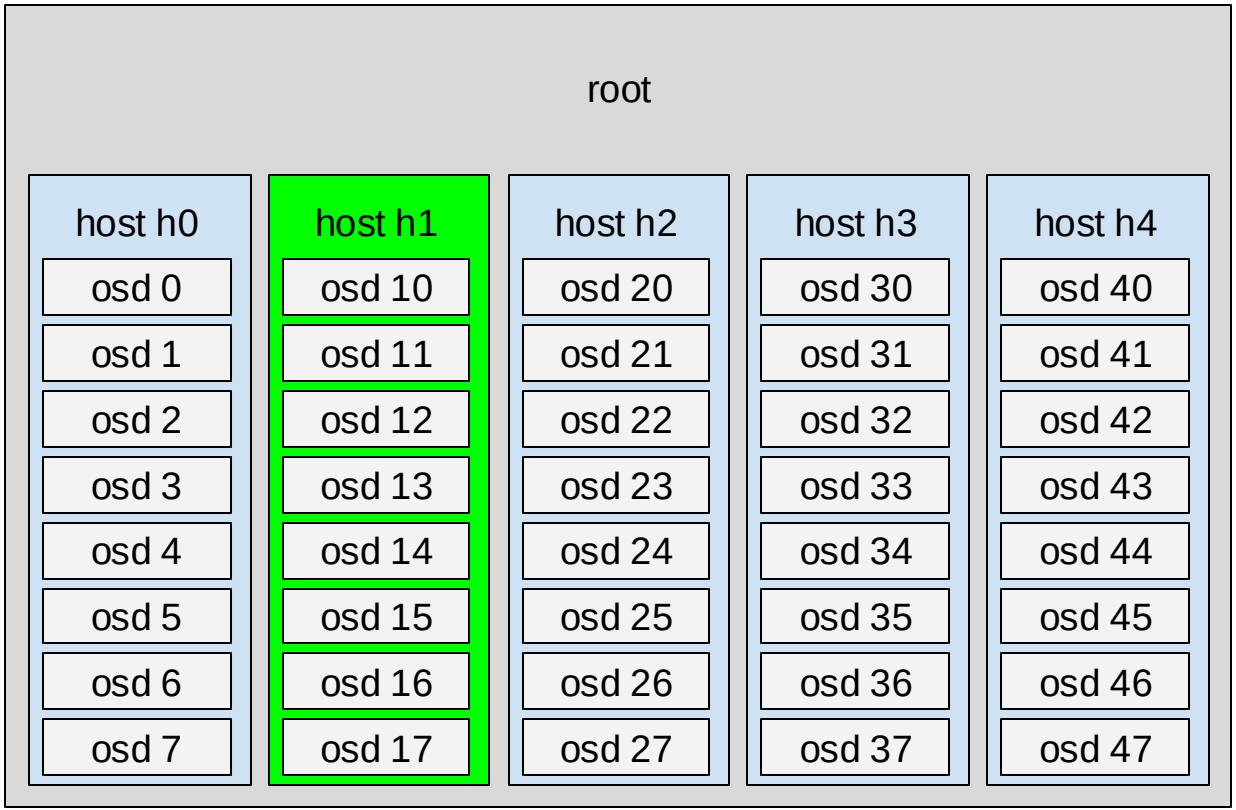


Why chooseleaf?

in: [h1, h0, h4]

step choose firstn 1 type osd

out: []

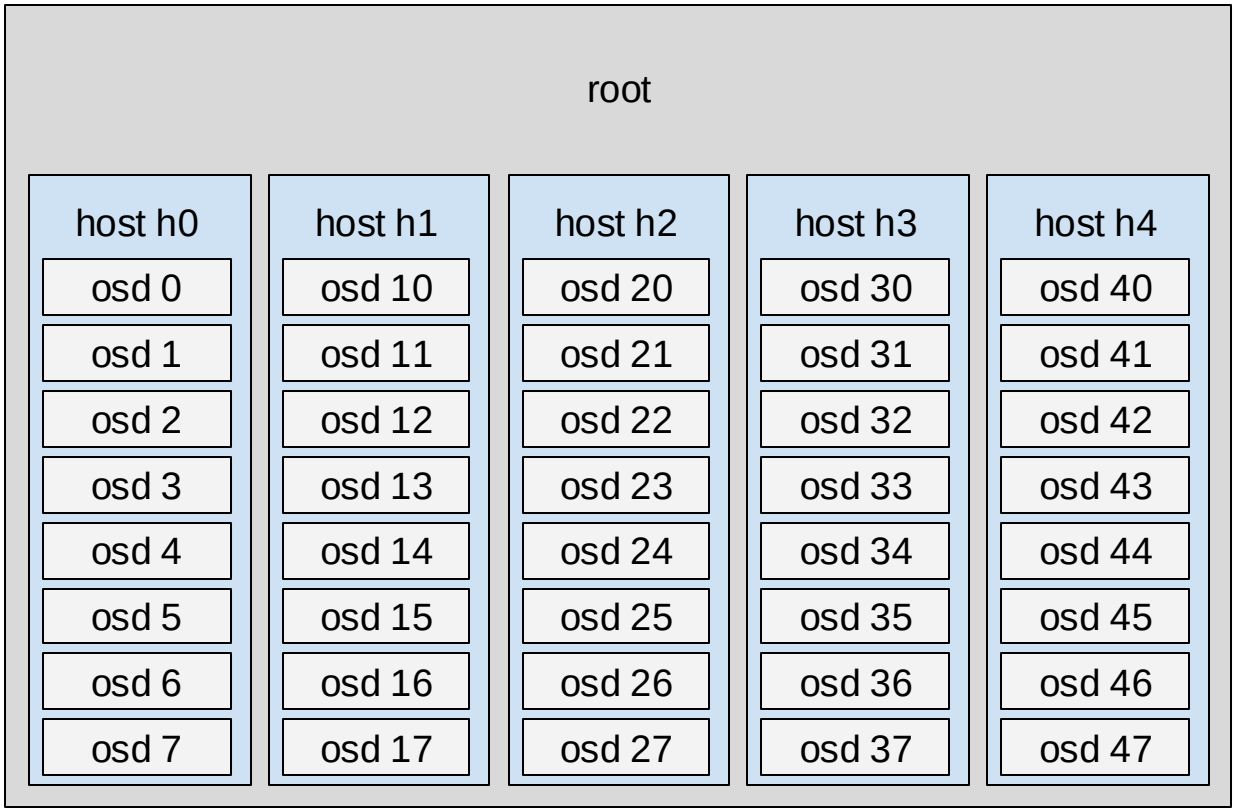


Why chooseleaf?

in: [h1, h0, h4]

step choose firstn 1 type osd

out: []

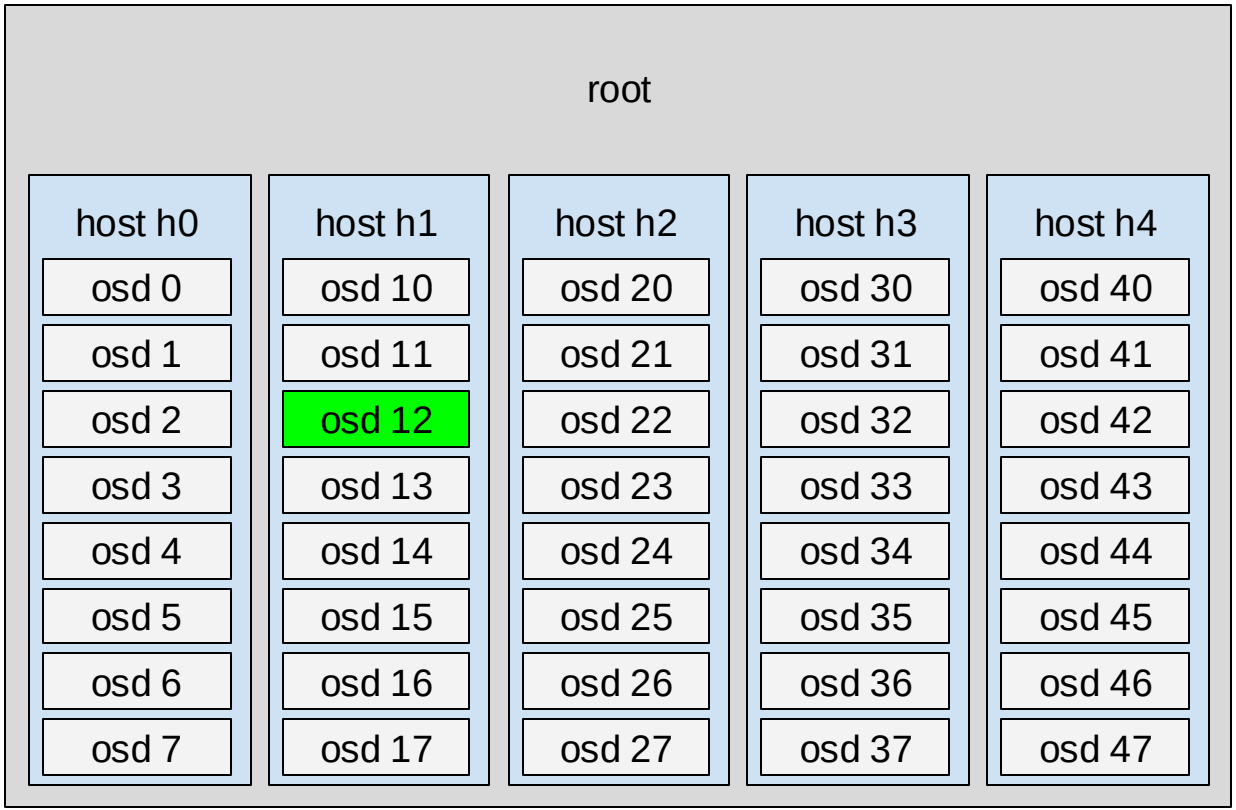


Why chooseleaf?

in: [h1, h0, h4]

step choose firstn 1 type osd

out: [12]

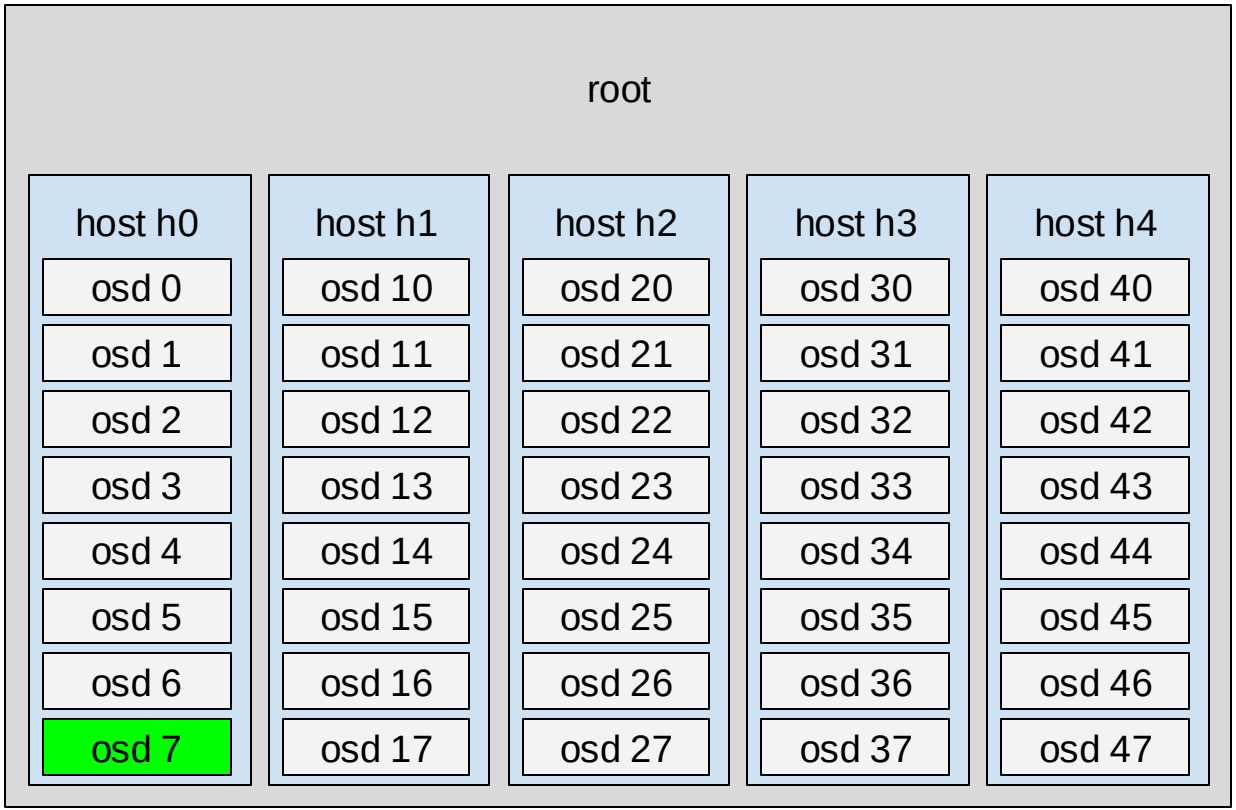


Why chooseleaf?

in: [h1, h0, h4]

step choose firstn 1 type osd

out: [12, 7]

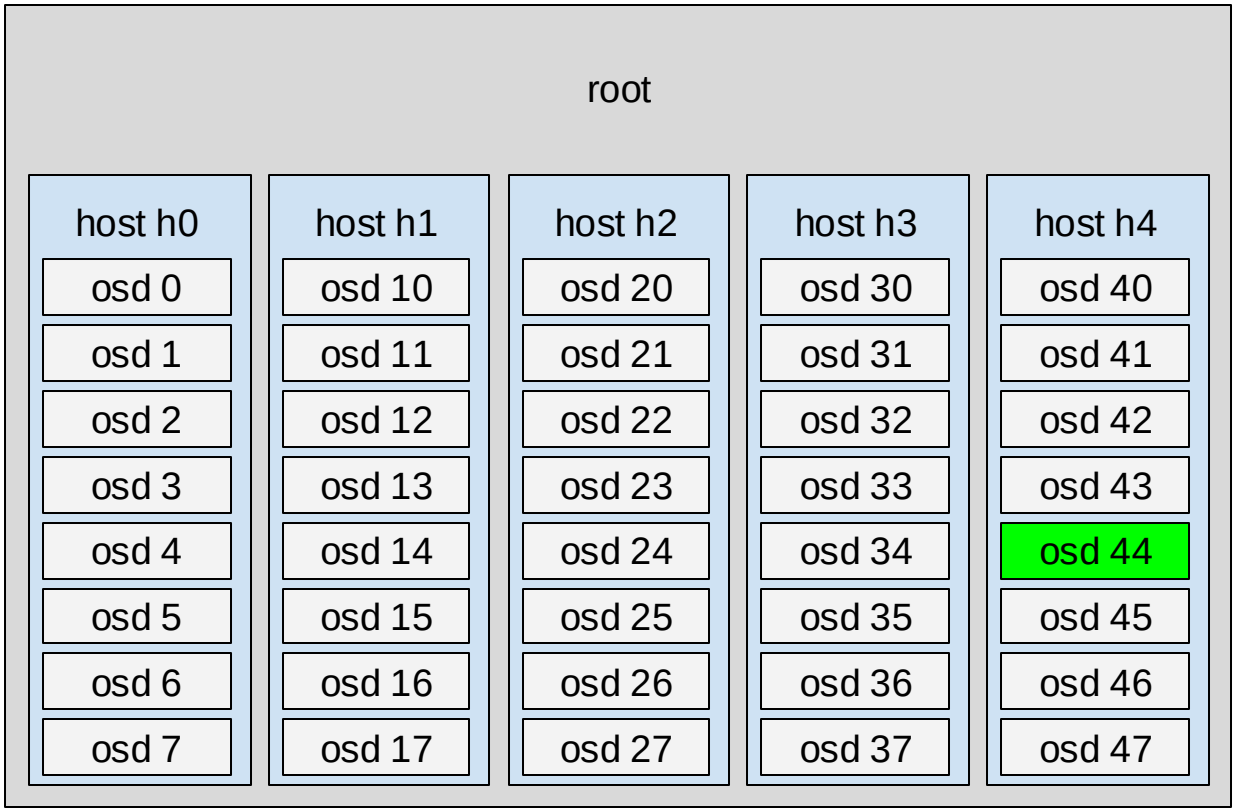


Why chooseleaf?

in: [h1, h0, h4]

step choose firstn 1 type osd

out: [12, 7, 44]



Why chooseleaf?

```
rule replicated_rule_1 {
    ...
    step take default class ssd
    step chooseleaf firstn 3 type host
    step emit
}
```

```
rule replicated_rule_1 {
    ...
    step take default class ssd
    step choose firstn 3 type host
    step choose firstn 1 type osd
    step emit
}
```

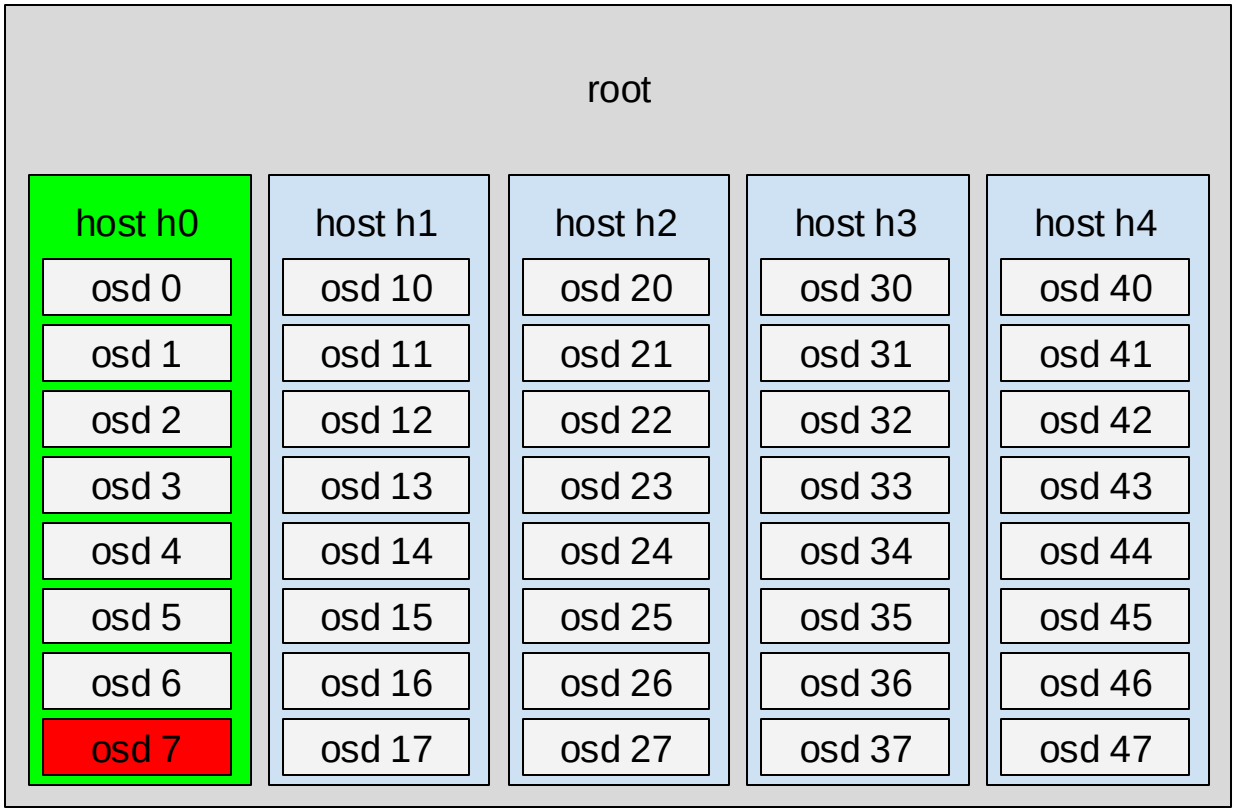
How do we handle an out osd?

in: [root]

step chooseleaf firstn 3 type host

out: [h1, h0]

out2: [12]



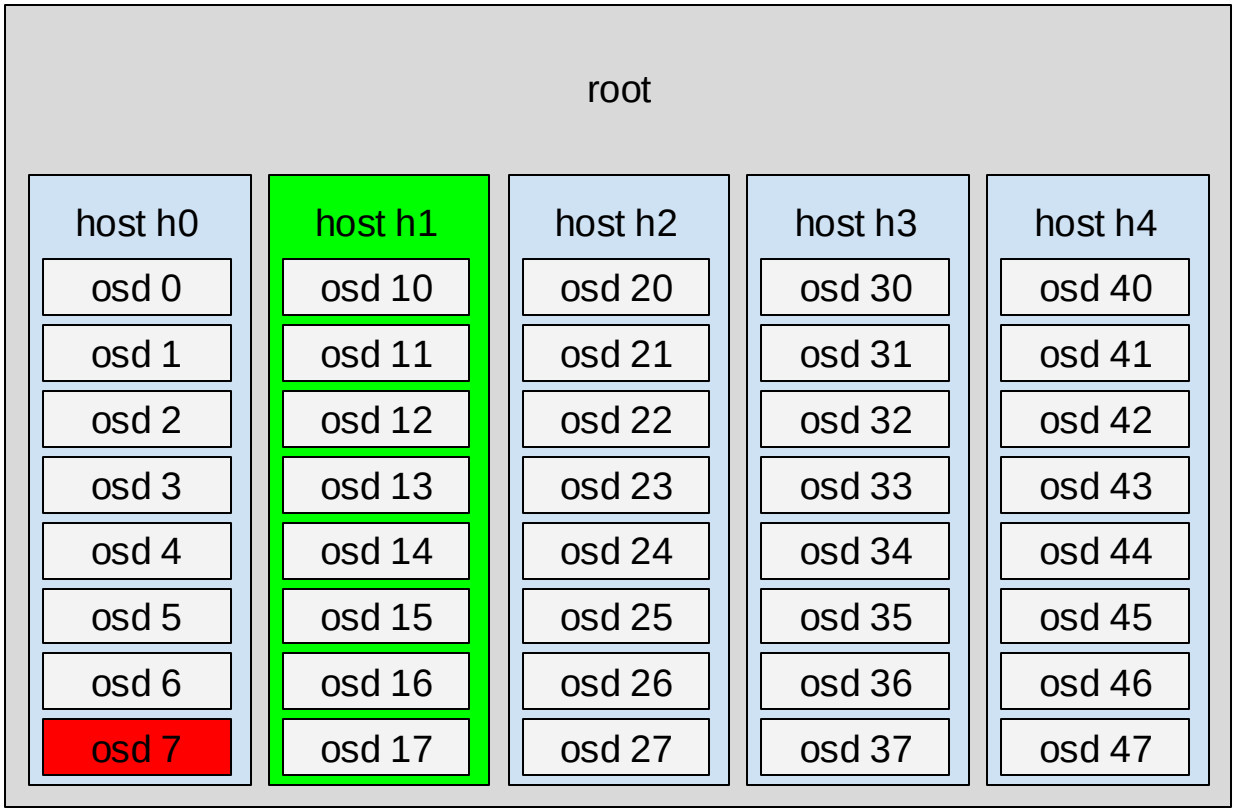
How do we handle an out osd?

in: [root]

step chooseleaf firstn 3 type host

out: [h1]

out2: [12]



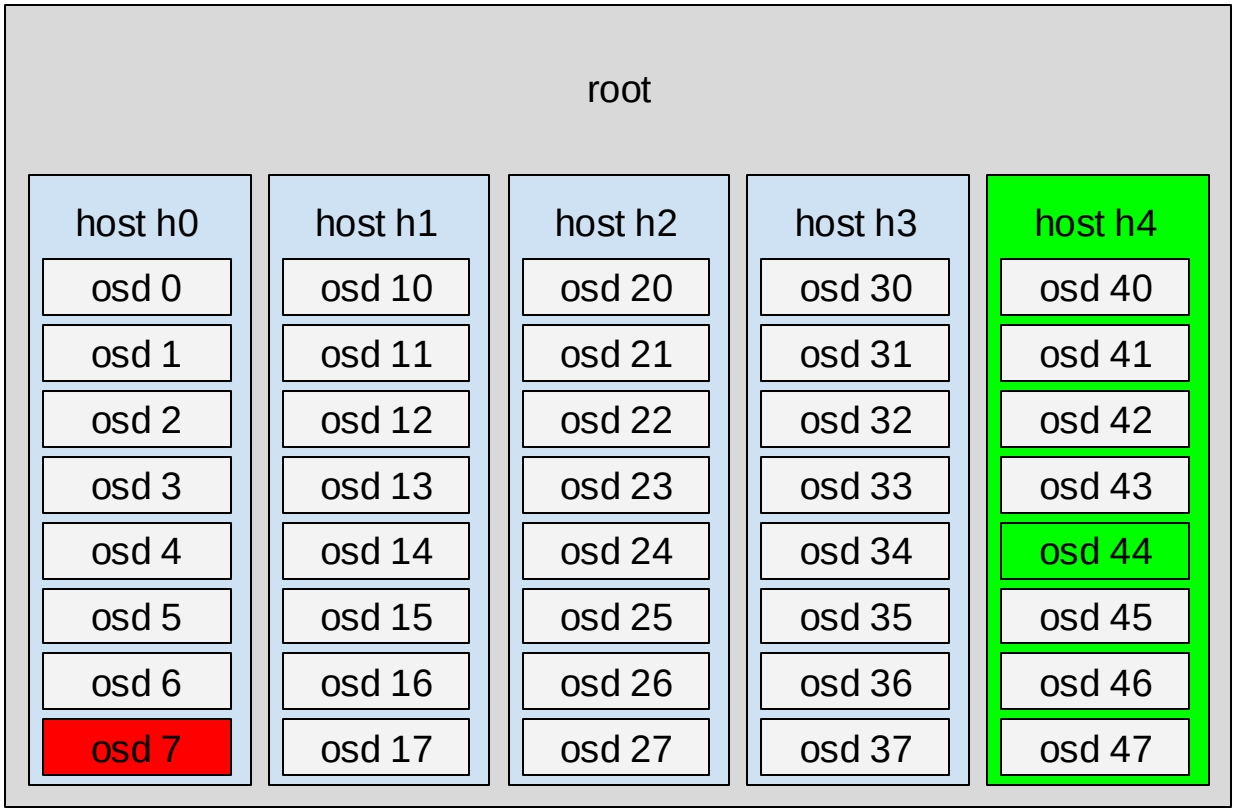
How do we handle an out osd?

in: [root]

step chooseleaf firstn 3 type host

out: [h1, h4]

out2: [12, 44]



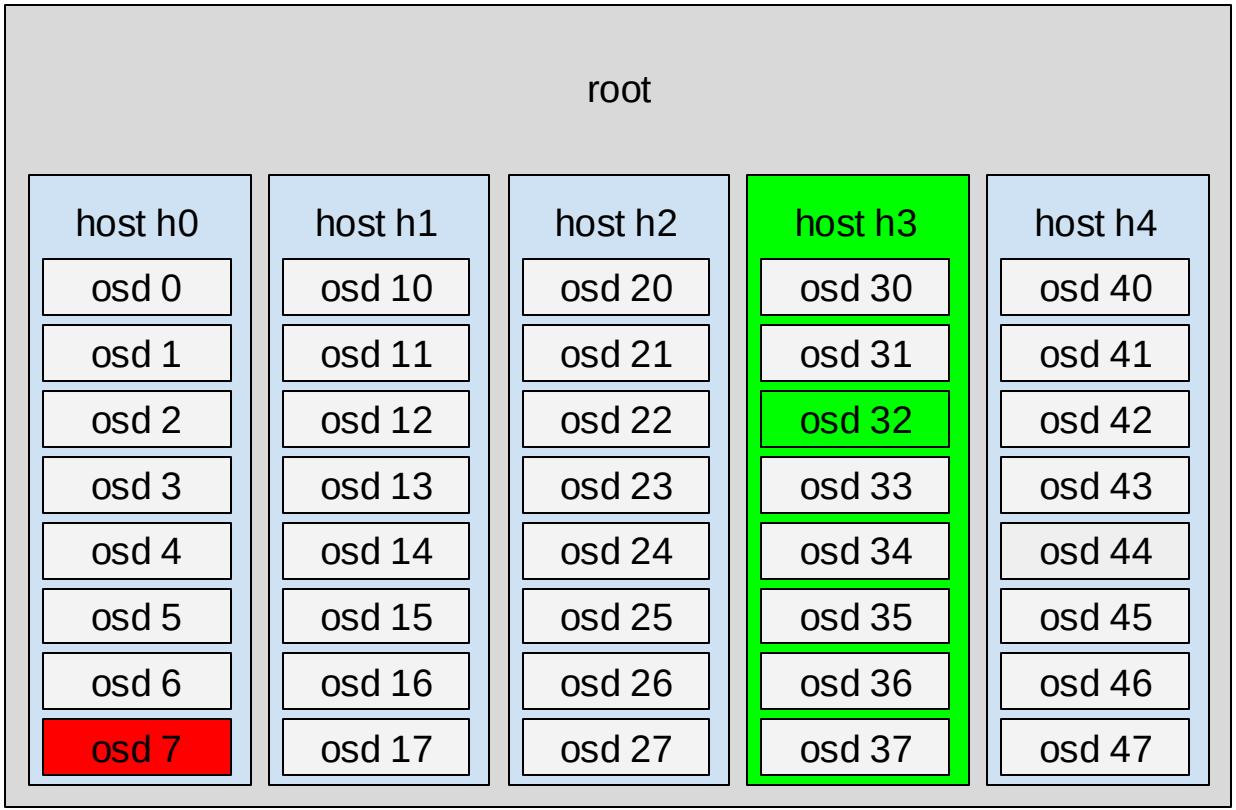
How do we handle an out osd?

in: [root]

step chooseleaf firstn 3 type host

out: [h1, h4, h3]

out2: [12, 44, 32]



Why chooseleaf?

```
rule replicated_rule_1 {
    ...
    step take default class ssd
    step chooseleaf firstn 3 type host
    step emit
}
```

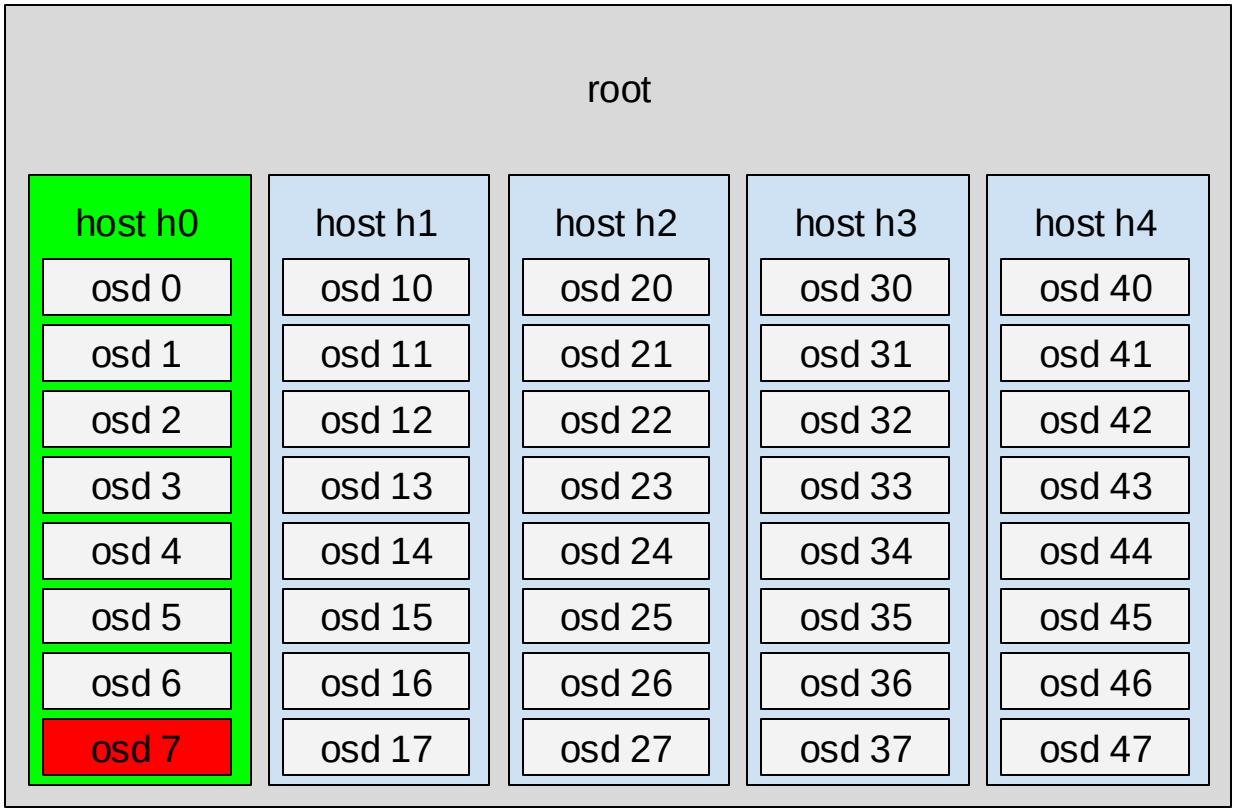
```
rule replicated_rule_1 {
    ...
    step take default class ssd
    step choose firstn 3 type host
    step choose firstn 1 type osd
    step emit
}
```

How do we handle an out osd?

in: [h1, h0, h4]

step choose firstn 1 type osd

out: [12]

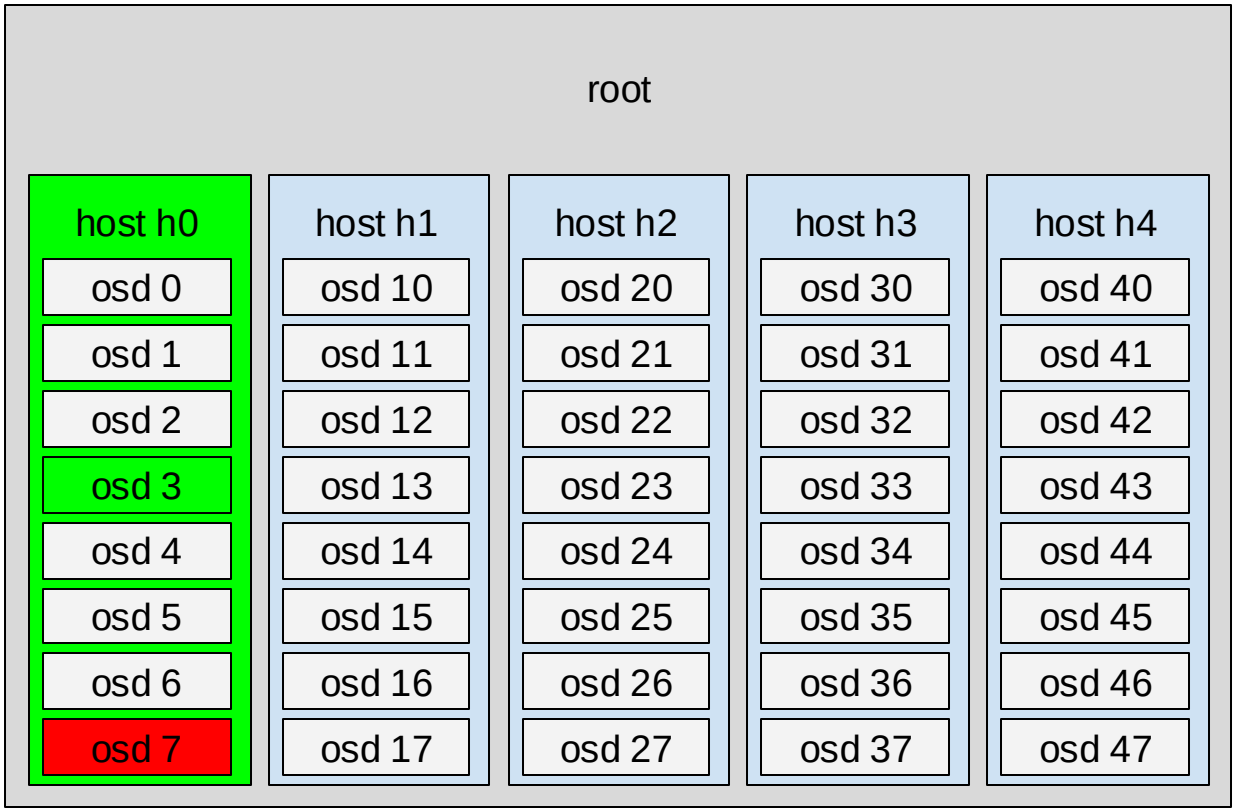


How do we handle an out osd?

in: [h1, h0, h4]

step choose firstn 1 type osd

out: [12, 3]

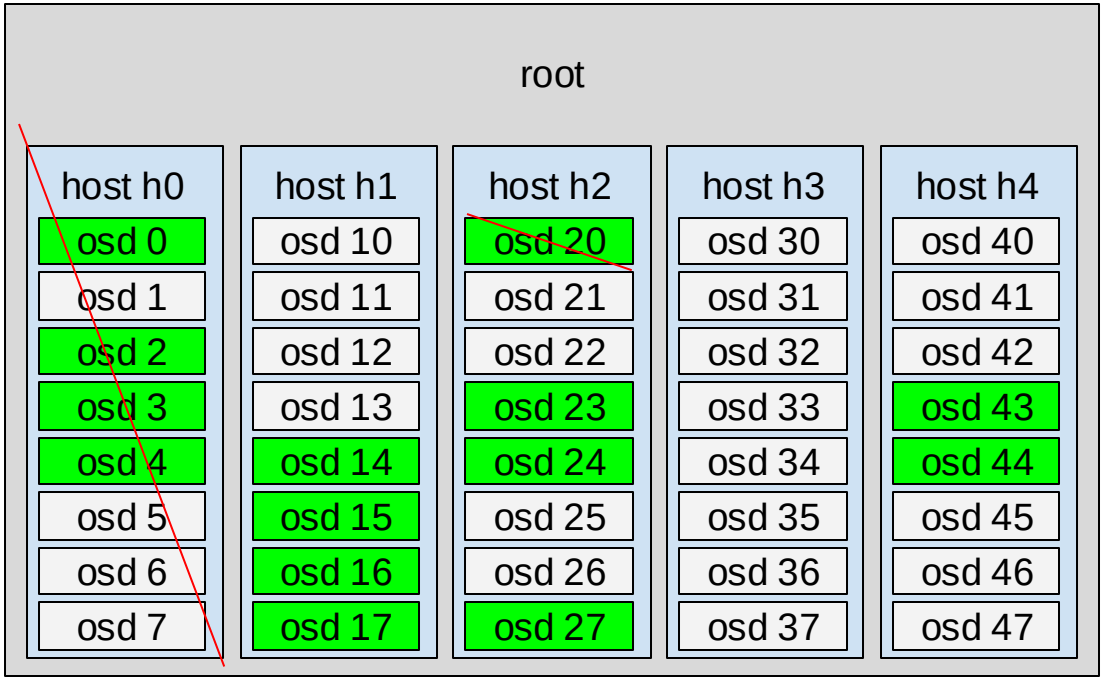


Why chooseleaf?

- Once normal CRUSH has completed a step, it cannot retry it.
- In the example above with two choose steps, marking osd.7 out will always remap its pgs to other osds on the same host resulting in that host being overloaded. If all of the osds on that host are marked out, some pgs will simply fail to map to 3 replicas.
- chooseleaf sidesteps the problem by combining a step with a descent to leaf. crush_choose_firstn has special support for retrying the bucket mapping when it hits an out osd.

Ok, why not just use chooseleaf?

- 8+6 EC split across 4 hosts (4/4/4/2) with min_size=9
- Storage overhead is 1.75
- Losing any single host + any single osd still leaves any pg with at least 9 copies and therefore writable.
- Doing the same thing with replication would require 3 replicas and therefore an overhead of 3.0.



Multiple steps…but with retry?

```
rule ec_rule_8_6 {
    ...
    step take default class ssd
    step choose indep 4 type host
    step choose indep 4 type osd
    step emit
}
```

Multi Step Retry -> MSR

```
rule ec_rule_8_6 {
    ...
    step take default class ssd
    step choose indep 4 type host
    step choose indep 4 type osd
    step emit
}
```

```
rule ec_rule_8_6 {
    type msr_indep
    ...
    step take default class ssd
    step choosemsr 4 type host
    step choosemsr 4 type osd
    step emit
}
```

MSR – Simple Case

```
rule msr_rule_2_2 {
   type msr_indep
    ...
    step take default class ssd
    step choosemsr 2 type host
    step choosemsr 2 type osd
    step emit
}
```

MSR – Simple Case

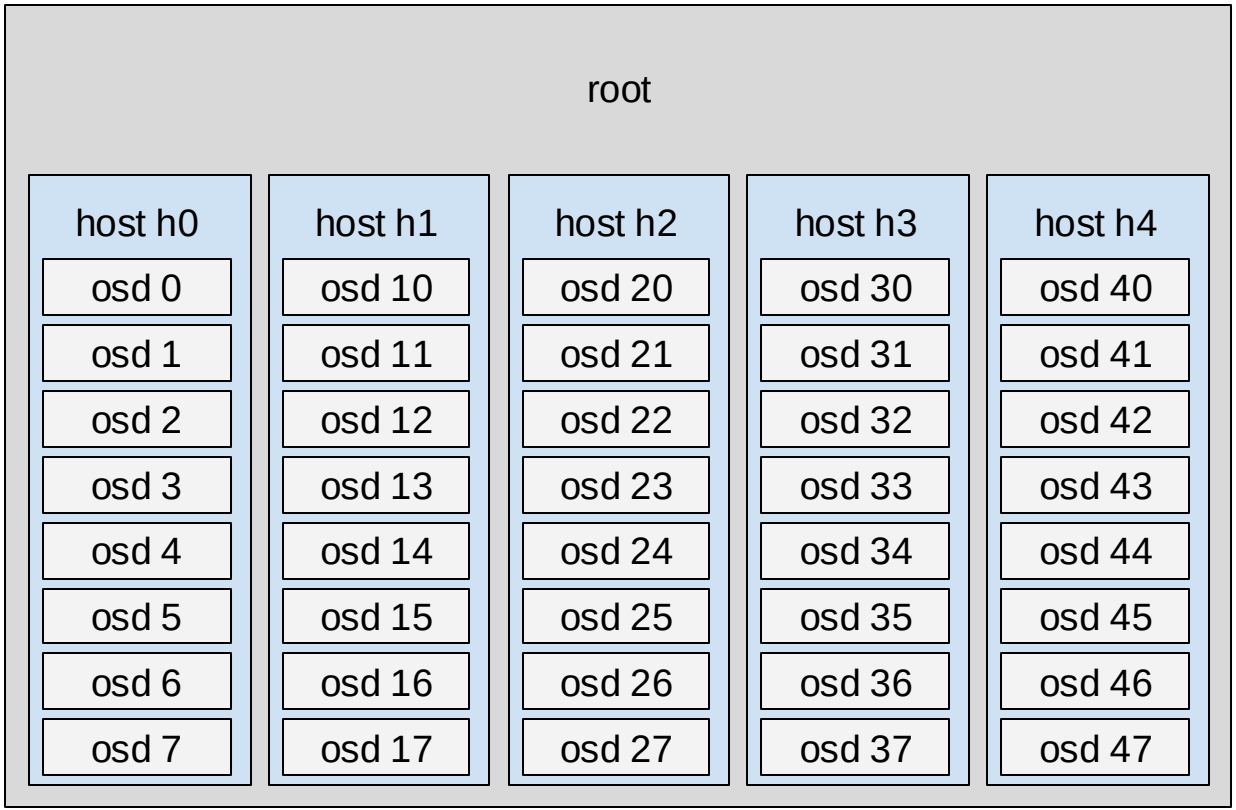
bucket: root

step choosemsr 2 type host

[X, X, X, X]

step choosemsr 2 type osd

[X, X, X, X]



MSR – Simple Case

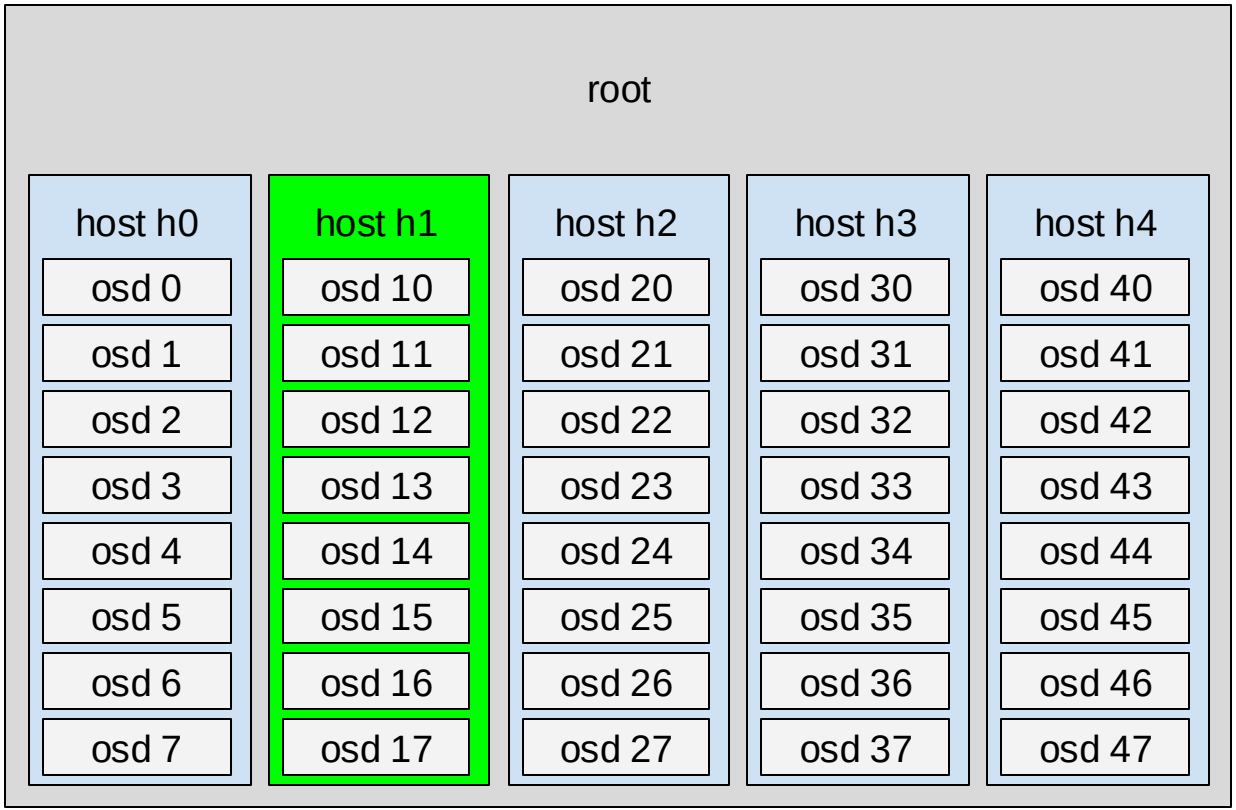
bucket: root

step choosemsr 2 type host

[h1, X, X, X]

step choosemsr 2 type osd

[X, X, X, X]



MSR – Simple Case

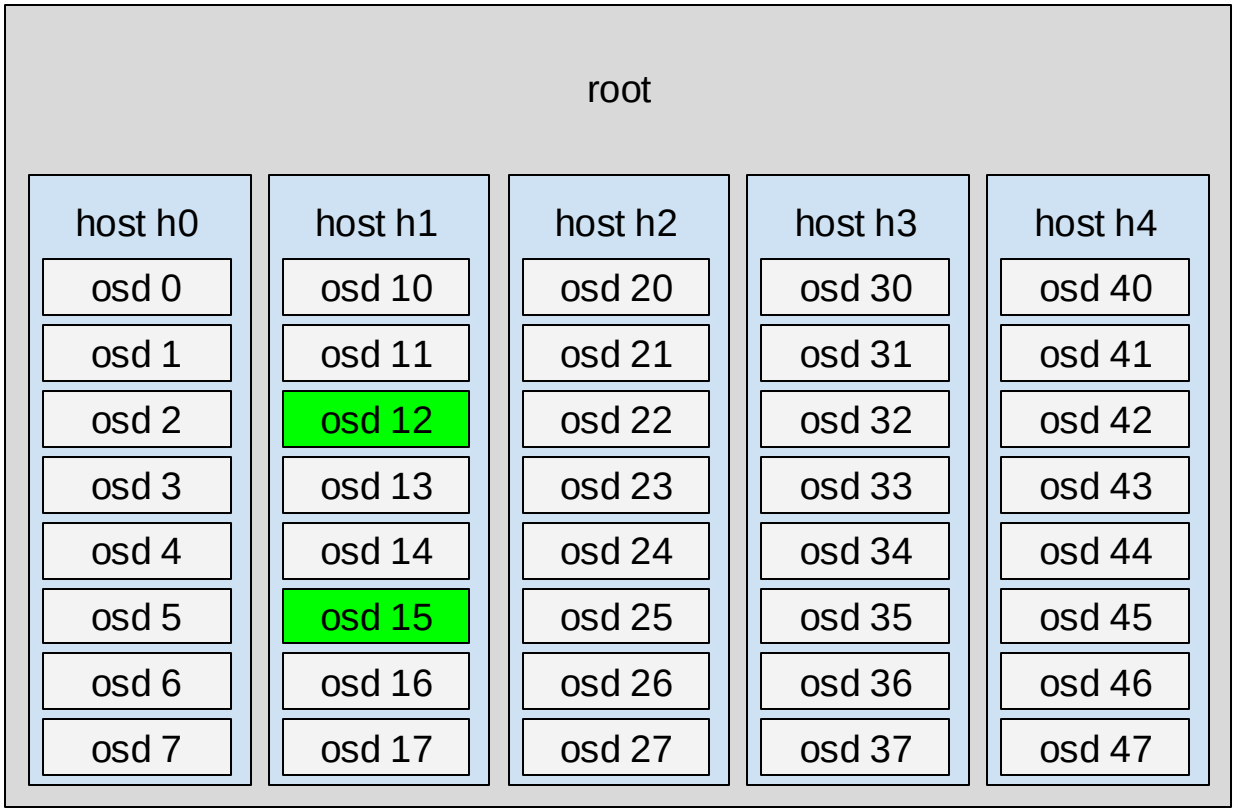
bucket: root

step choosemsr 2 type host

[h1, X, X, X]

step choosemsr 2 type osd

[12, 15, X, X]



MSR – Simple Case

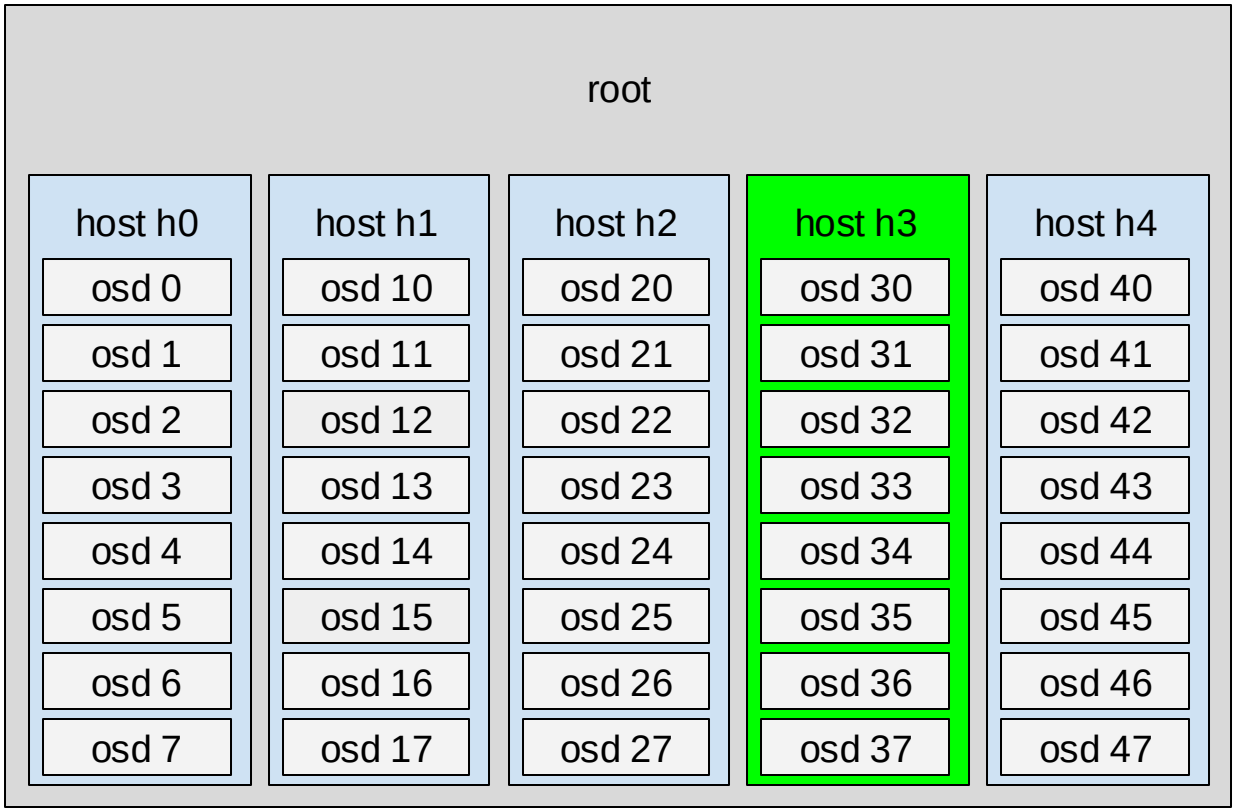
bucket: root

step choosemsr 2 type host

[h1, X, h3, X]

step choosemsr 2 type osd

[12, 15, X, X]



MSR – Simple Case

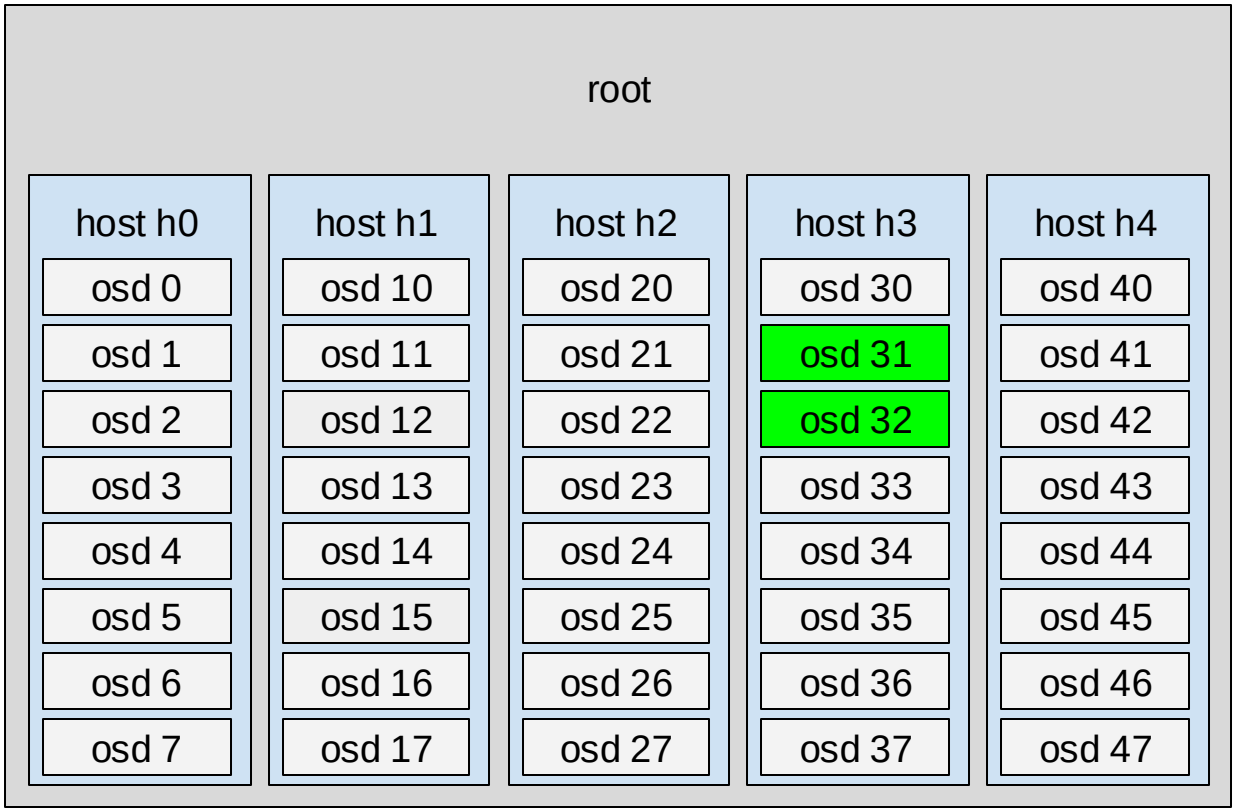
bucket: root

step choosemsr 2 type host

[h1, X, h3, X]

step choosemsr 2 type osd

[12, 15, 31, 32]



MSR – Out OSD Handling

```
rule msr_rule_2_2 {
   type msr_indep
    ...
    step take default class ssd
    step choosemsr 2 type host
    step choosemsr 2 type osd
    step emit
}
```

MSR – Out OSD Handling

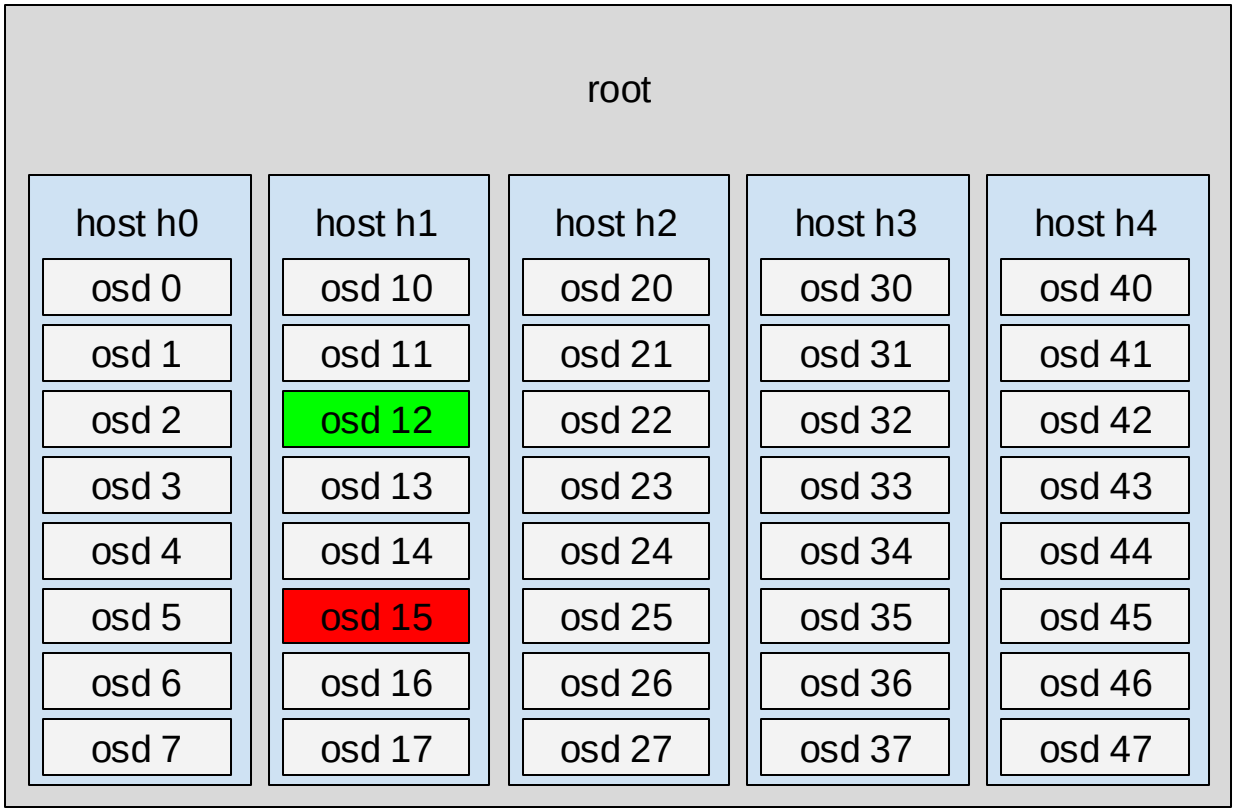
bucket: root

step choosemsr 2 type host

[h1, X, X, X]

step choosemsr 2 type osd

[12, X, X, X]



MSR – Out OSD Handling

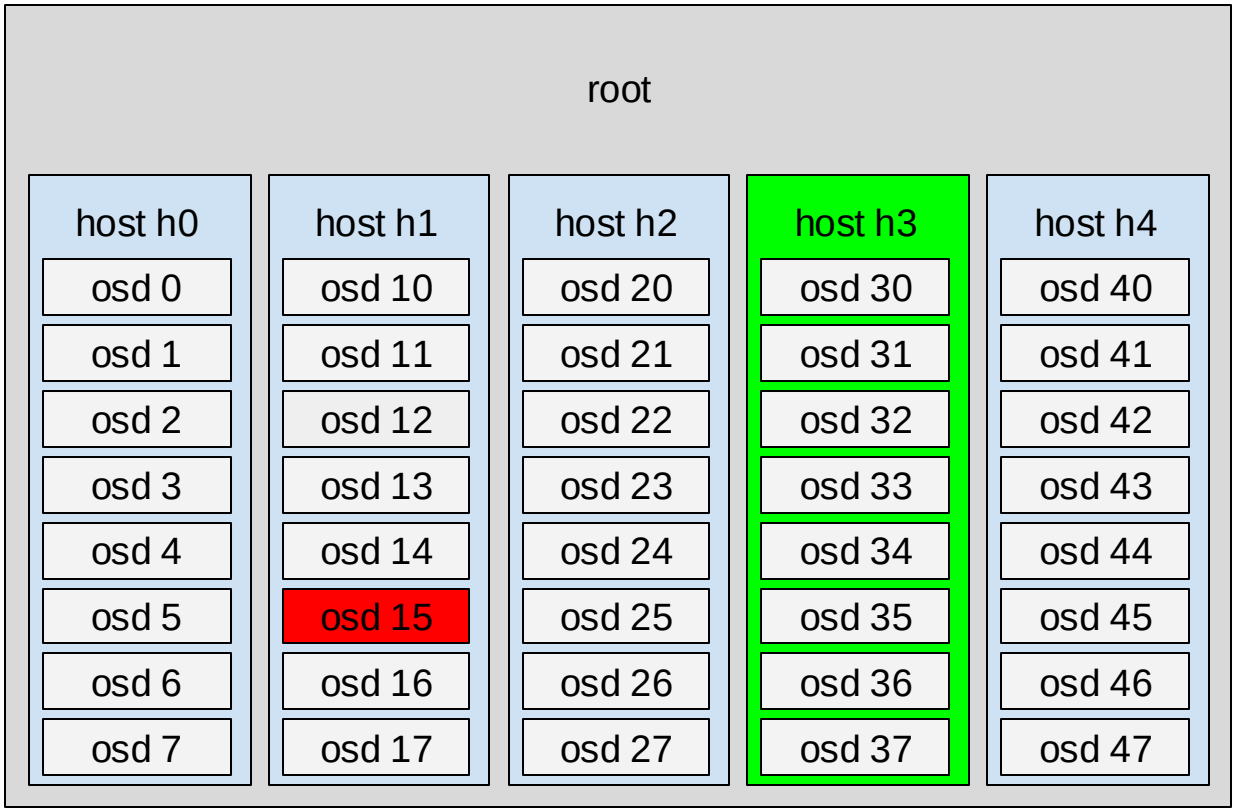
bucket: root

step choosemsr 2 type host

[h1, X, h3, X]

step choosemsr 2 type osd

[12, X, X, X]



MSR – Out OSD Handling

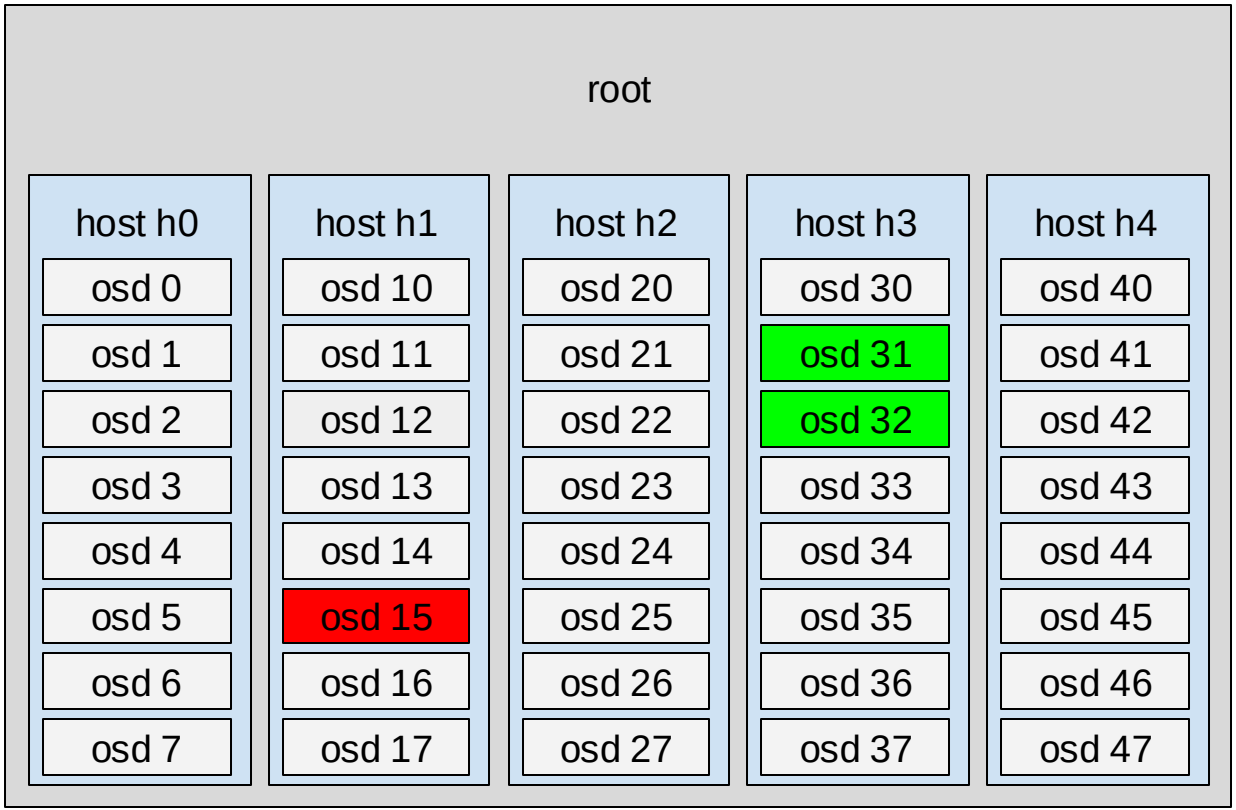
bucket: root

step choosemsr 2 type host

[h1, X, h3, X]

step choosemsr 2 type osd

[12, X, 31, 32]



MSR – Out OSD Handling

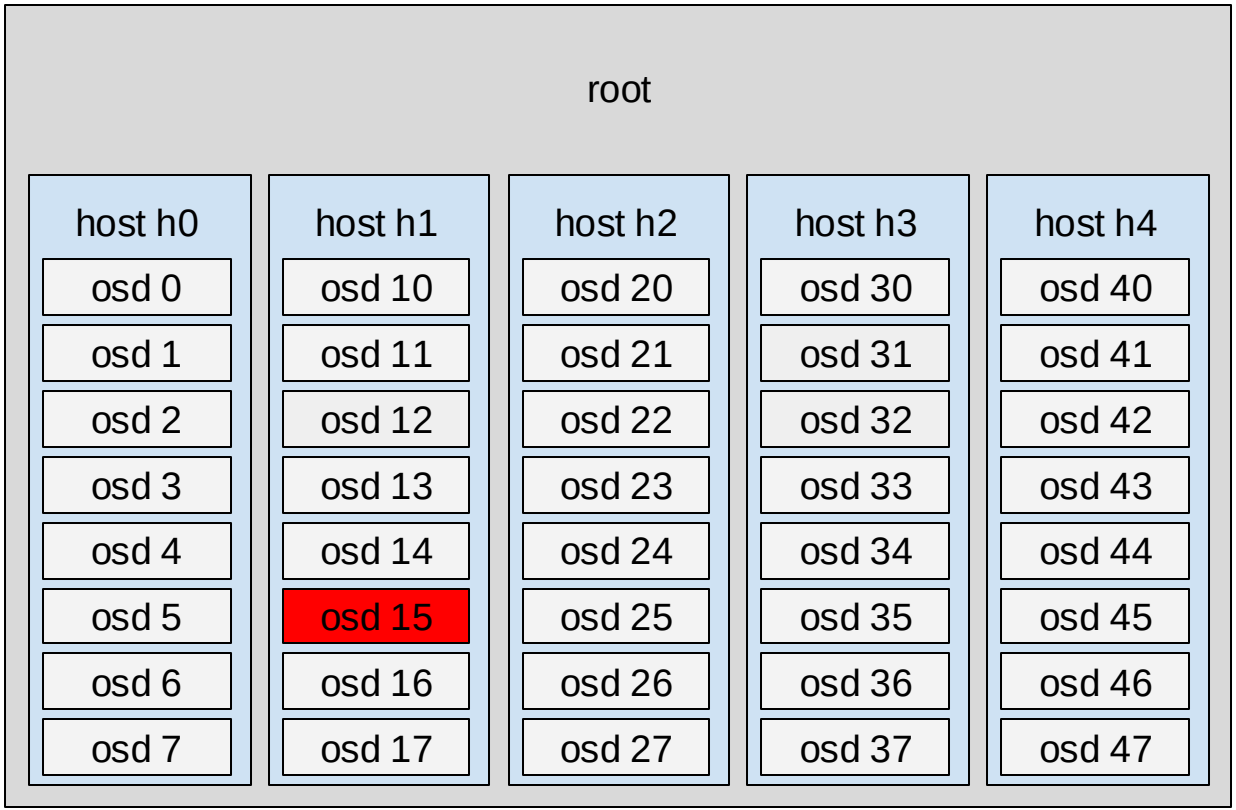
bucket: root

step choosemsr 2 type host

[h1, X, h3, X]

step choosemsr 2 type osd

[12, X, 31, 32]



MSR – Out OSD Handling

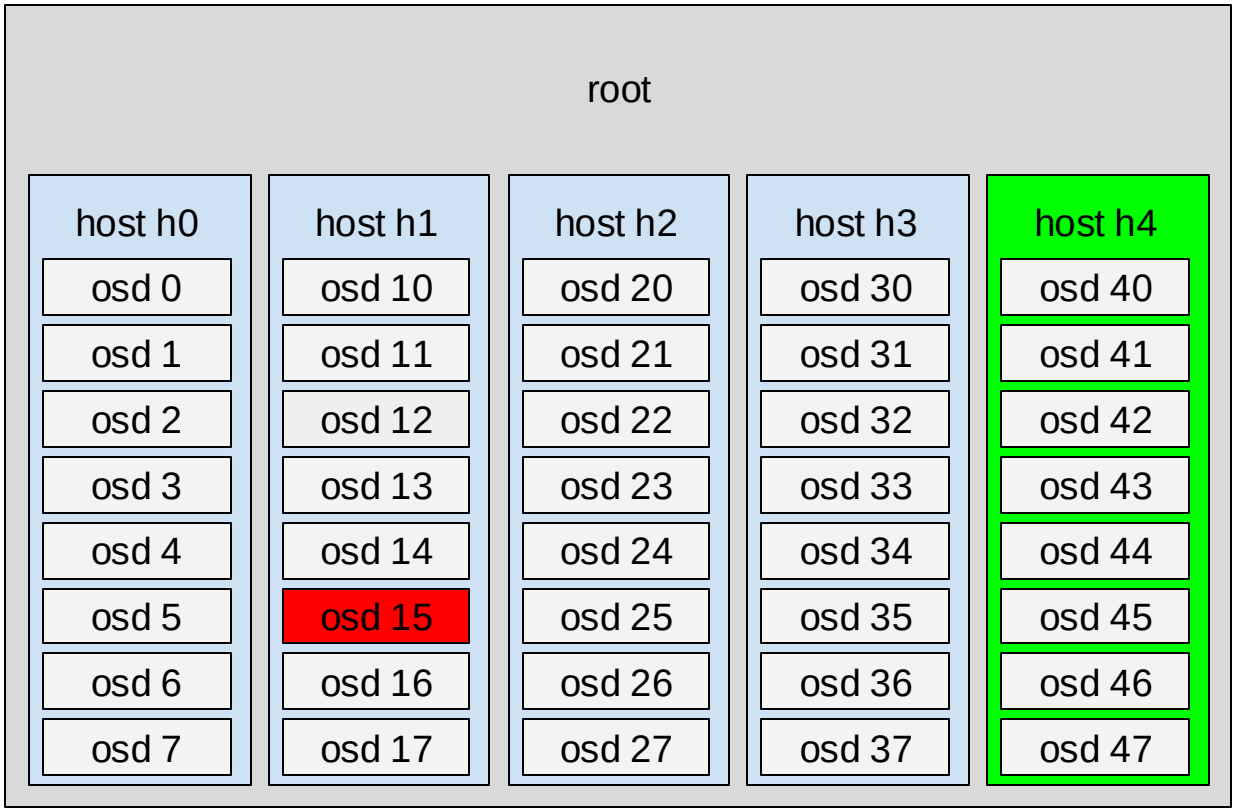
bucket: root

step choosemsr 2 type host

[h1, h4, h3, X]

step choosemsr 2 type osd

[12, X, 31, 32]



MSR – Out OSD Handling

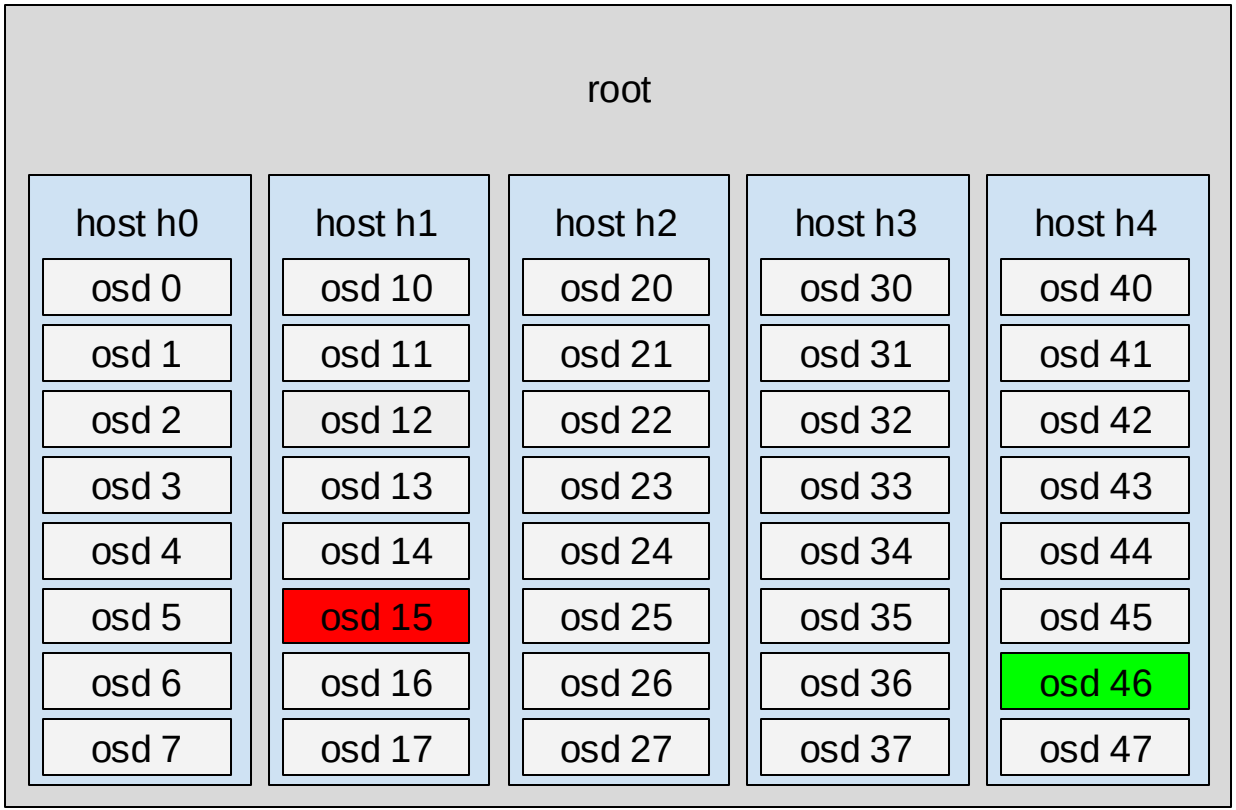
bucket: root

step choosemsr 2 type host

[h1, h4, h3, X]

step choosemsr 2 type osd

[12, 46, 31, 32]



MSR – We got 3 hosts! That's ok!

- chooseleaf only ever maps a single leaf to a single failure domain
- When chooseleaf hits an out OSD, it can simply retry the whole mapping.
- MSR isn't limited to a single OSD per failure domain – if it hits an out osd on the 3rd of 4 mappings, should it retry all of them? What if a host only has 3 live OSDs?
- Instead, MSR relaxes the restrictions slightly – instead of requiring 2 hosts in our simple example, it instead requires *at least* 2 hosts such that each has *at most* 2 osds.

MSR

- Depth First – For each output position, descends to leaf before proceeding to the next position.
- Retries from root.
- May map more failure domains than specified.
- Retries happen after complete pass.

MSR – More Information

- Generally used via the crush-osds-per-failure-domain erasure code profile option https://docs.ceph.com/en/latest/rados/operations/crush-map/#crush-msr-rules
- https://docs.ceph.com/en/latest/dev/crush-msr/
  - Higher level explanation
- src/crush/mapper.c
  - crush_msr_do_rule and crush_msr_choose are the main entry points
  - *Heavily* documented