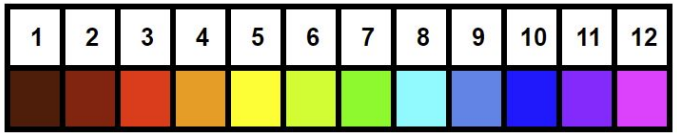
ZFS Vdev Reliability

Vdev Reliability Examples

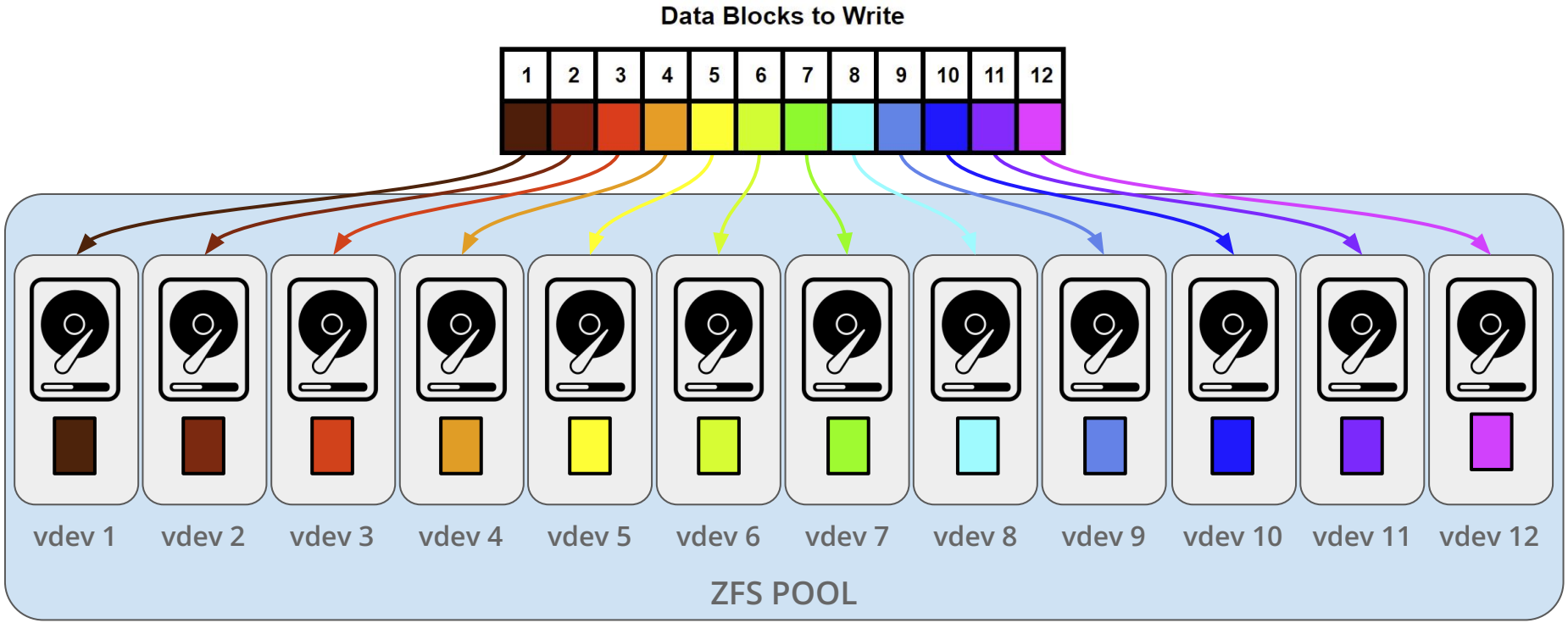
- In the following slides, we'll go through examples of how each vdev type spreads its data across its drives
- When data is written to a ZFS pool, it is broken up into smaller chunks called "blocks". The size of those blocks can actually vary depending on the file size, but we won't worry about that. Just know that ZFS breaks up data into "blocks".
- We'll write data to the pools represented by a series of rainbow blocks, with one color representing on block of a file we're writing:

Data Blocks to Write



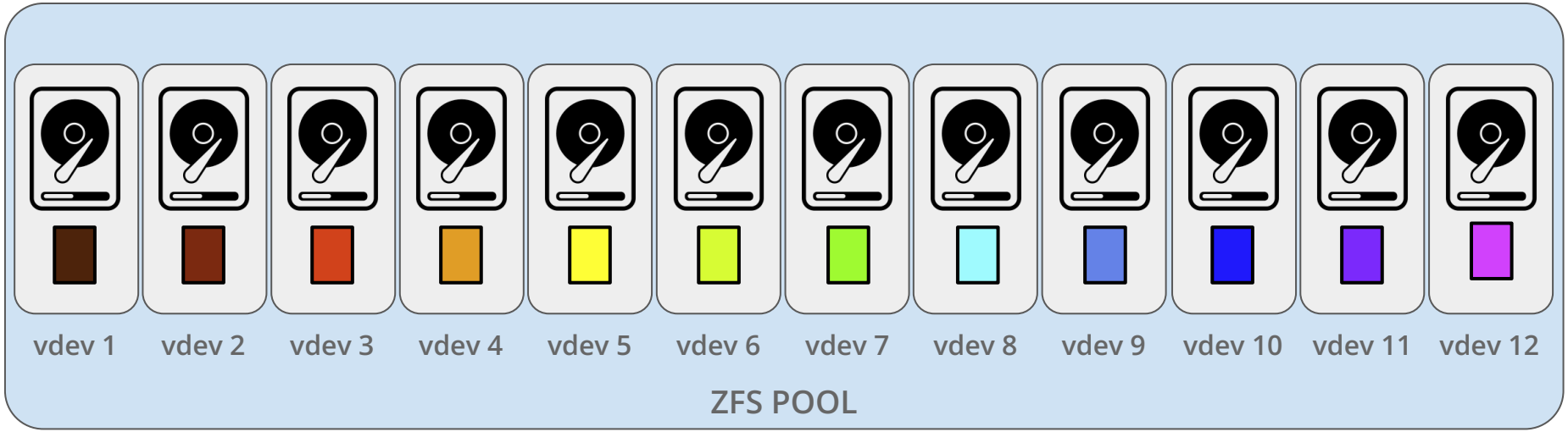
- We'll then see what happens to this data as disks fail. Our objective will always be to reconstruct the rainbow in spite of disk failures.
- The RAIDZ examples will do something a little different, but we'll discuss that when we get there...

Striped Vdev Example



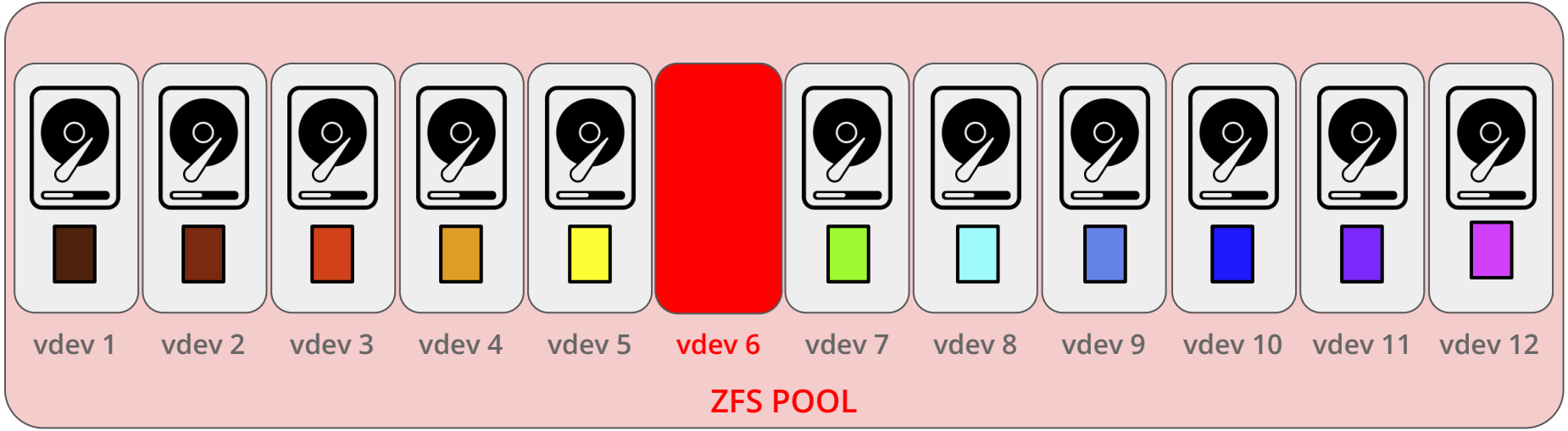
Striped Vdev Example

- Note that each single disk is a vdev in this example. Data is “striped” across all the vdevs in the same way we’ll see in later examples.
- **Striped pools have no redundancy!** If we lose a single disk, we’ll lose one color in our rainbow and our data will be incomplete…

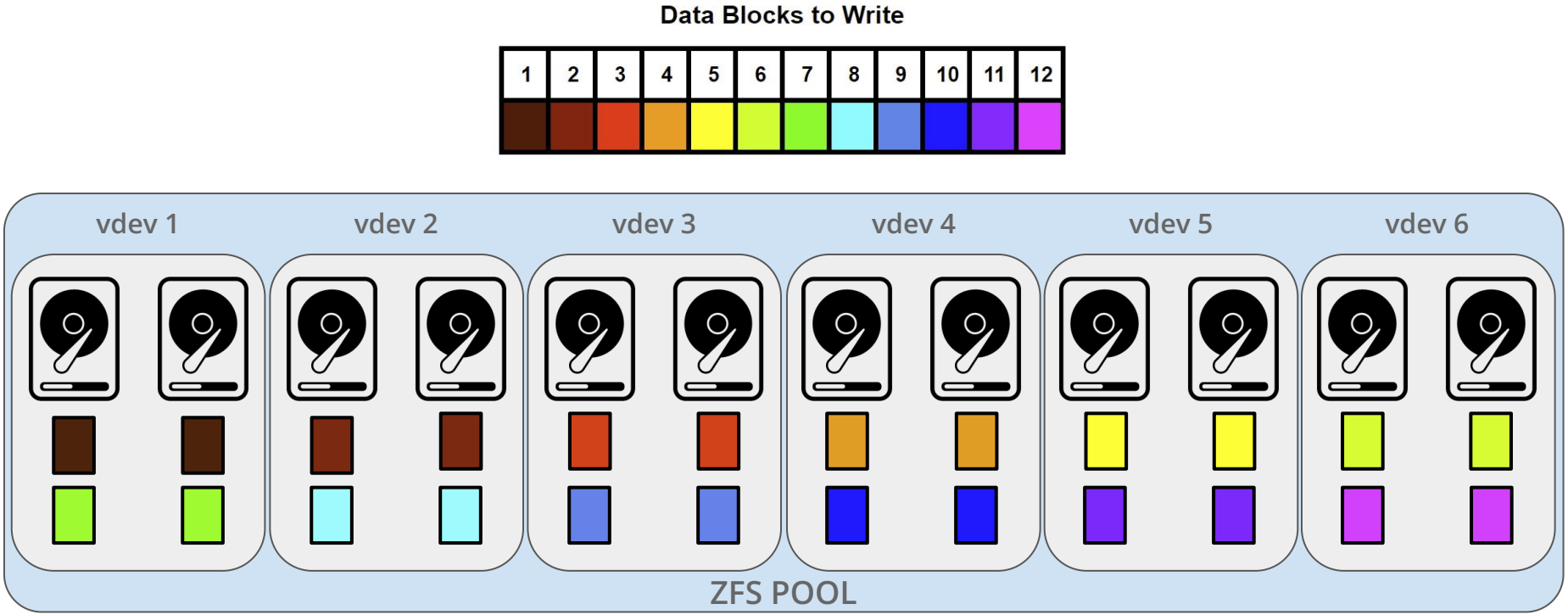


Striped Vdev Example

- Note that each single disk is a vdev in this example. Data is “striped” across all the vdevs in the same way we’ll see in later examples.
- **Striped pools have no redundancy!** If we lose a single disk, we’ll lose one color in our rainbow and our data will be incomplete…
- For this reason, striped pools are almost never used in production systems. In the next example, we’ll introduce some redundancy.

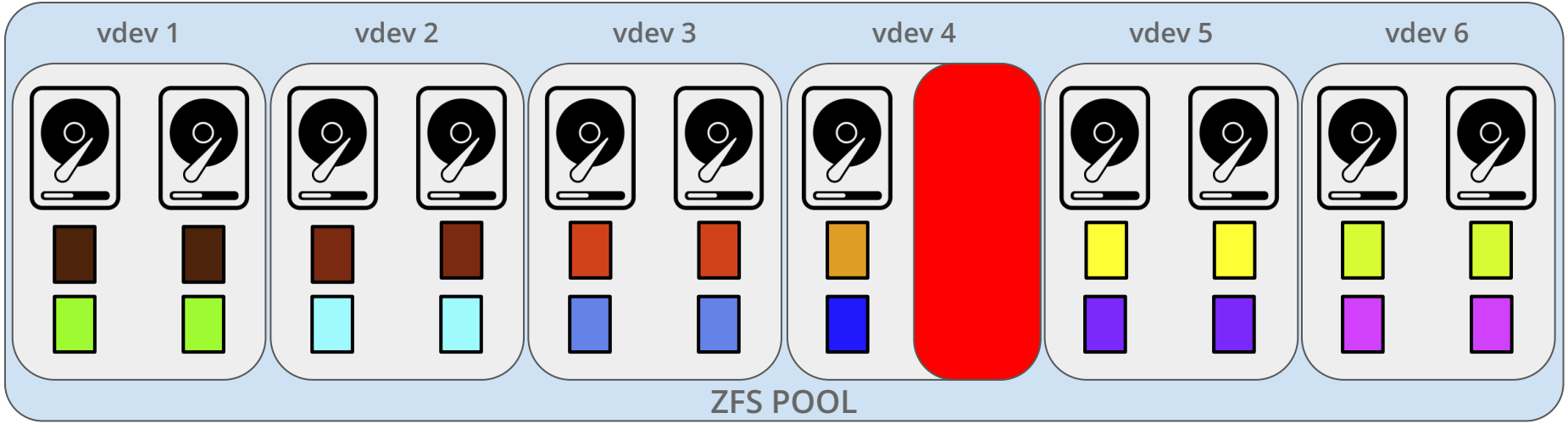


Mirrored Vdev Example



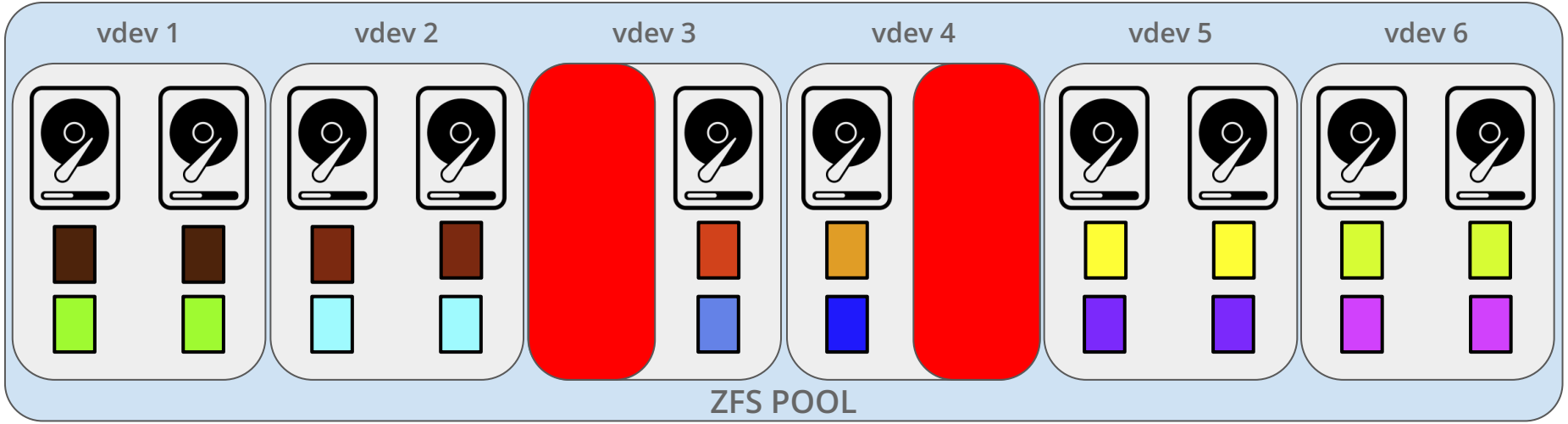
Mirrored Vdev Example

- Data is striped across all vdevs just like it was in the previous example, except each vdev now has two disks.
- In a mirrored vdev, each disk in the vdev gets one full copy of all the data written to the vdev, adding redundancy!
- Can lose all but one disk per vdev and not suffer data loss.



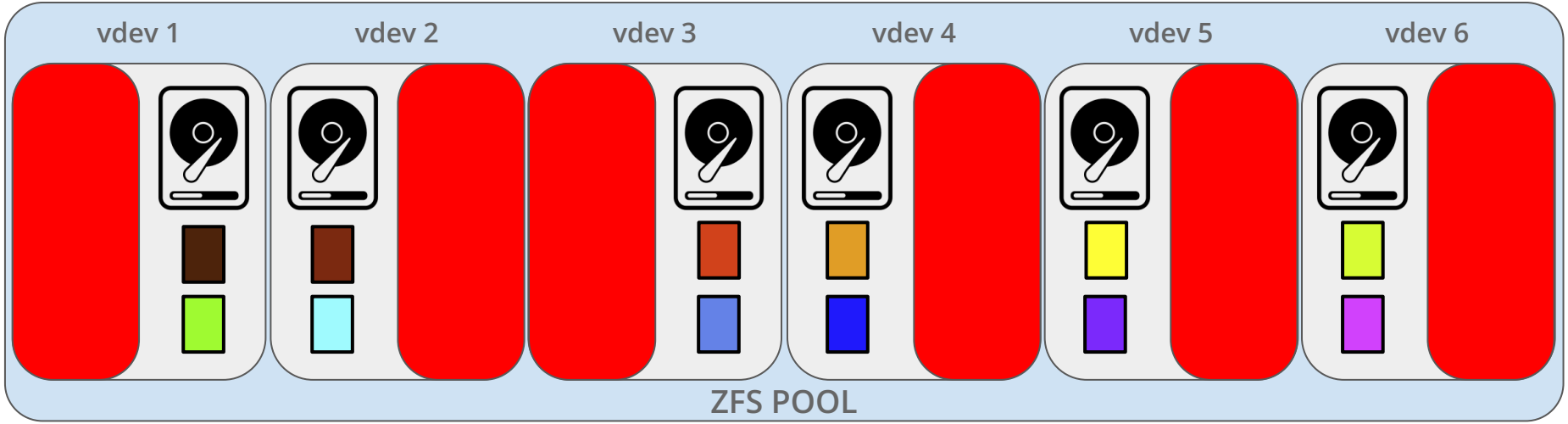
Mirrored Vdev Example

- Data is striped across all vdevs just like it was in the previous example, except each vdev now has two disks.
- In a mirrored vdev, each disk in the vdev gets one full copy of all the data written to the vdev, adding redundancy!
- Can lose all but one disk per vdev and not suffer data loss.
- Can lose multiple disks per pool, as long as no one vdev is completely lost (i.e., all drives in that vdev lost).



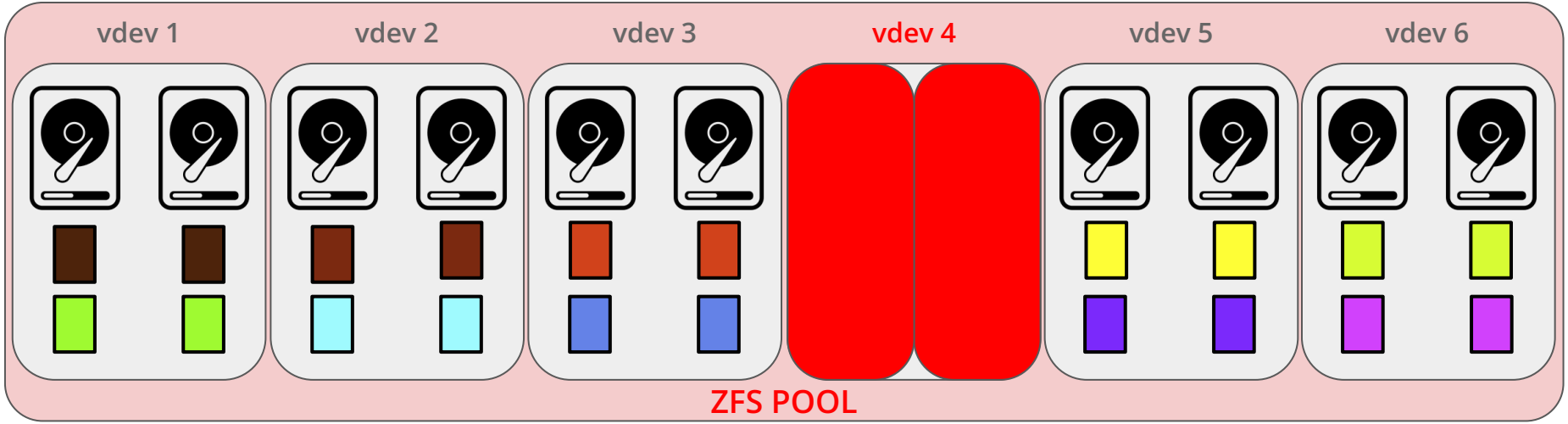
Mirrored Vdev Example

- Data is striped across all vdevs just like it was in the previous example, except each vdev now has two disks.
- In a mirrored vdev, each disk in the vdev gets one full copy of all the data written to the vdev, adding redundancy!
- Can lose all but one disk per vdev and not suffer data loss.
- Can lose multiple disks per pool, as long as no one vdev is completely lost (i.e., all drives in that vdev lost).



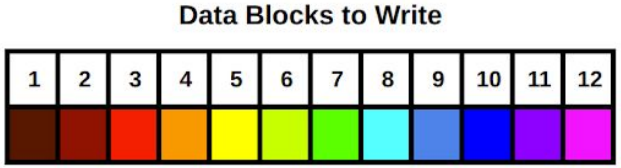
Mirrored Vdev Example

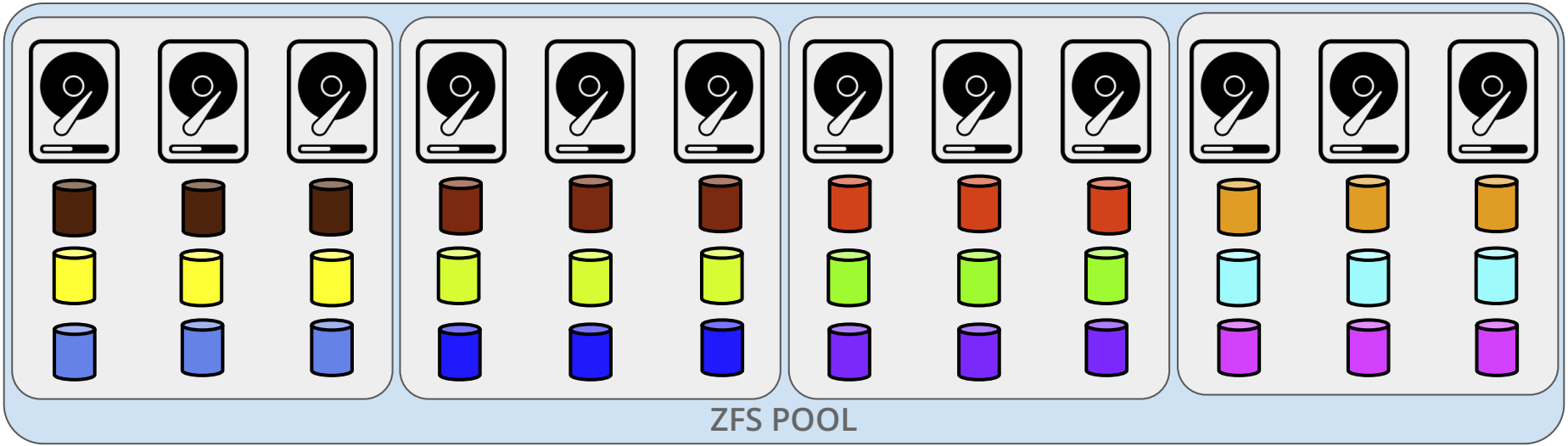
- Data is striped across all vdevs just like it was in the previous example, except each vdev now has two disks.
- In a mirrored vdev, each disk in the vdev gets one full copy of all the data written to the vdev, adding redundancy!
- Can lose all but one disk per vdev and not suffer data loss.
- Can lose multiple disks per pool, as long as no one vdev is completely lost (i.e., all drives in that vdev lost).
- All pool data is lost if any one mirror fails (i.e., all disks in the mirror fail).



Mirrored Vdev Example 2

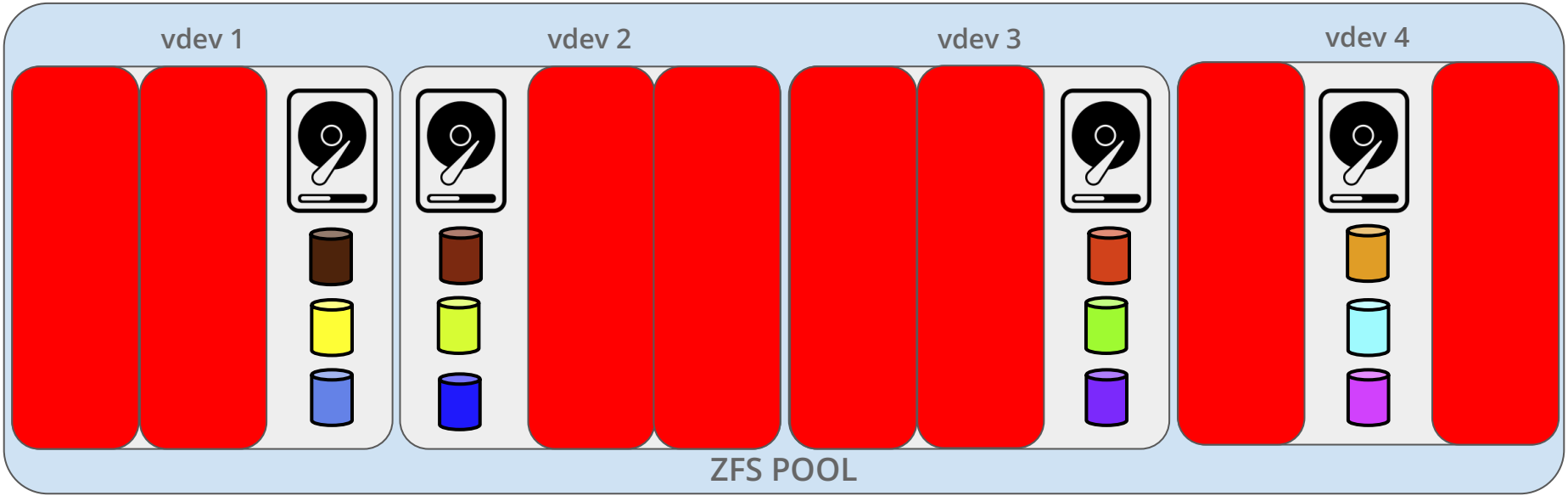
- Mirrored vdevs can have more than two disks!





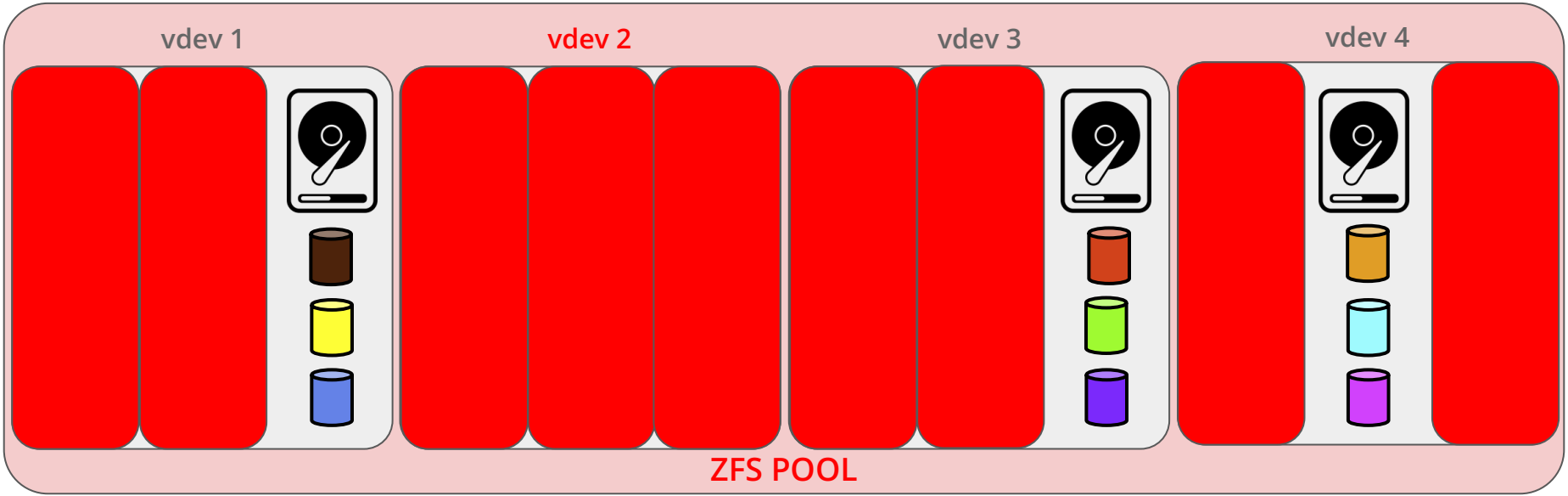
Mirrored Vdev Example 2

- Mirrored vdevs can have more than two disks!
- This adds more fault tolerance at the cost of usable disk space



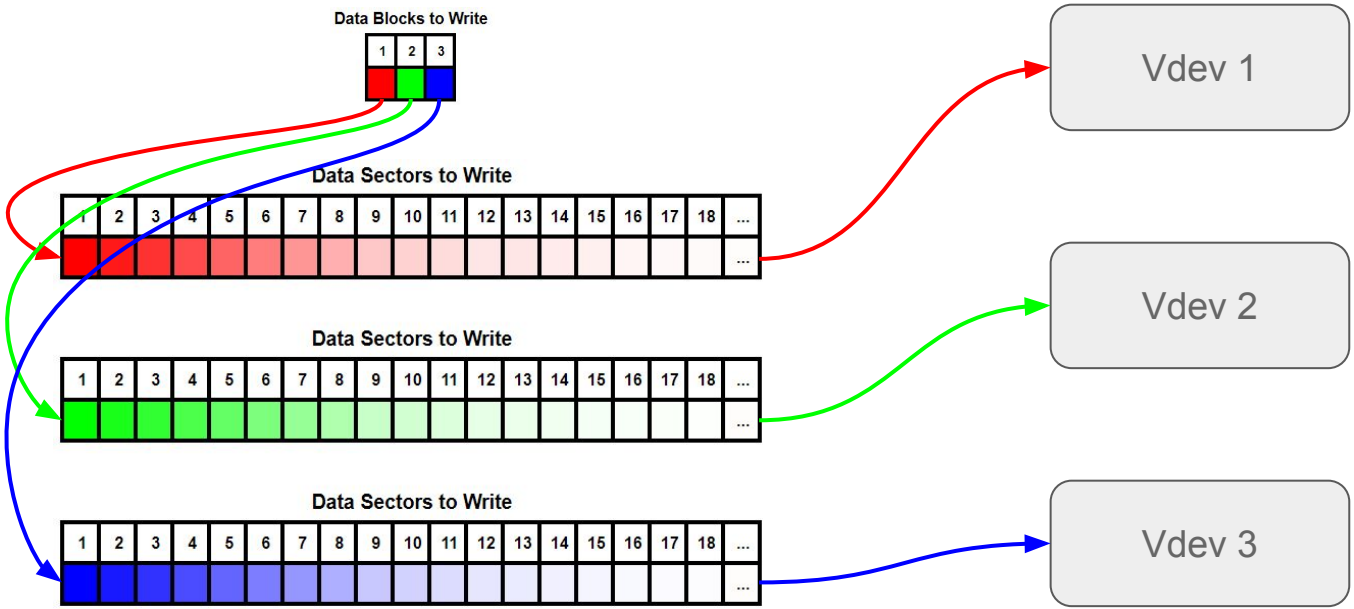
Mirrored Vdev Example 2

- Mirrored vdevs can have more than two disks!
- This adds more fault tolerance at the cost of usable disk space
- But as before, if we lose all disks in a single mirrored vdev, pool is lost...



RAIDZ Examples

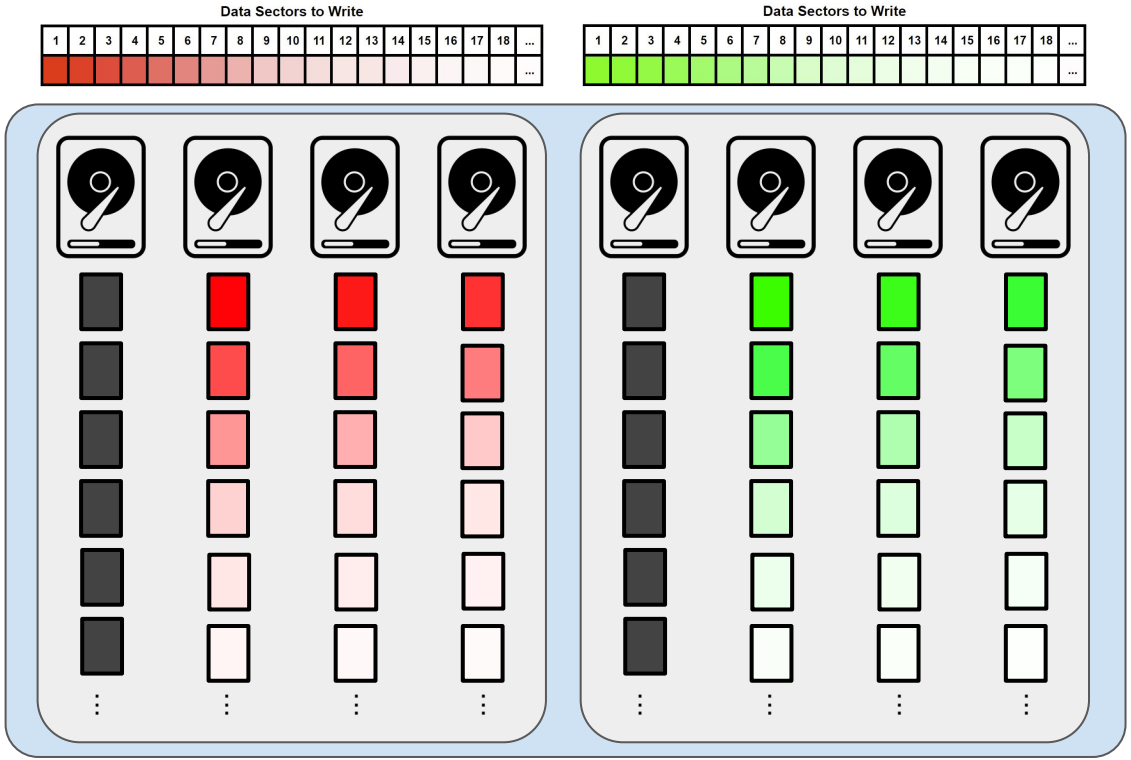
In the previous slides, each vdev got a single block from our rainbow. In the RAIDZ examples, this is still true, but each block will be broken up even further into smaller chunks called “sectors”, which we’ll represent as a gradient of that block’s color:



RAIDZ1 Vdev Example

= Parity Data

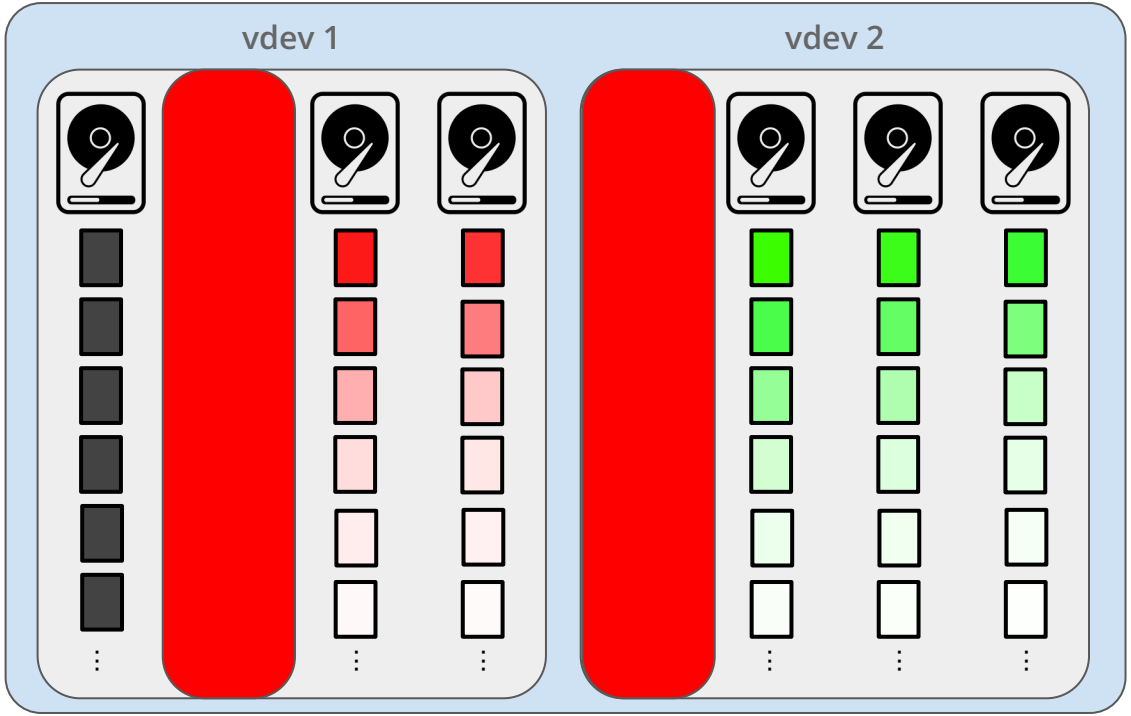
- Pool layout is similar to RAID 50 (RAID 5 + 0)
- Like RAID 5, we have one parity disks per vdev (and two vdevs in this pool).
- RAIDZ1 requires at least 3 disks per vdev, but can have as many total disks per vdev as you want!
- The number of disks in a vdev is referred to as its "**width**"; these are 4-wide Z1 vdevs.



RAIDZ1 Vdev Example

= Parity Data

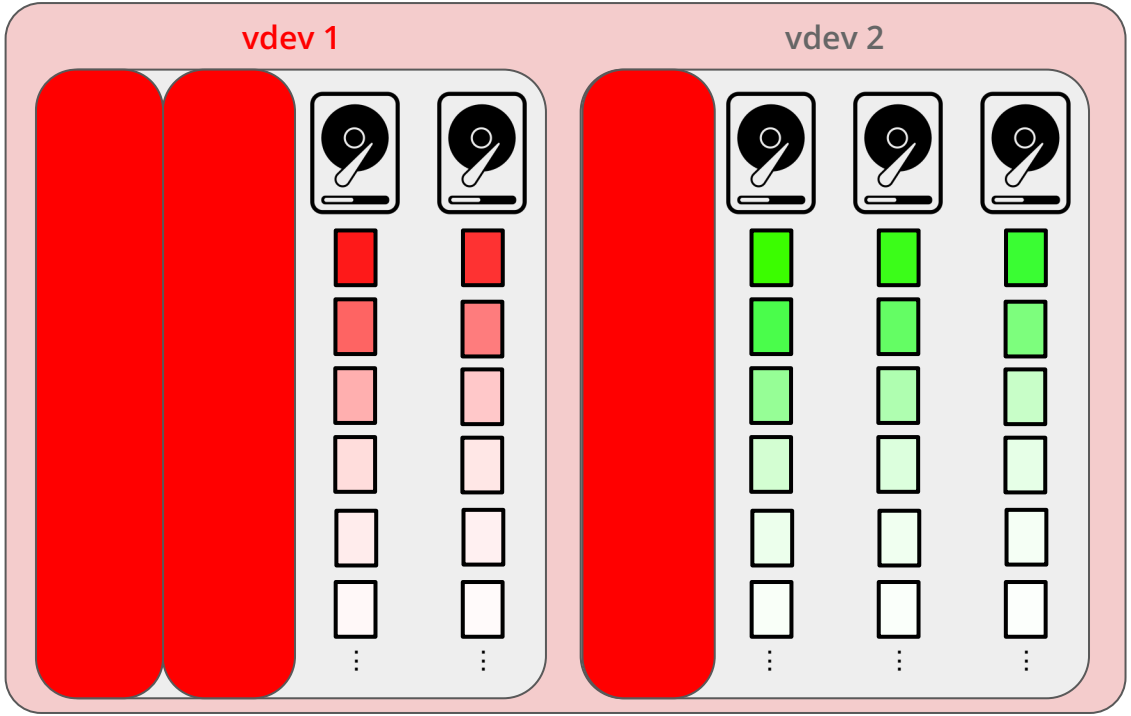
- Pool layout is similar to RAID 50 (RAID 5 + 0)
- Like RAID 5, we have one parity disks per vdev (and two vdevs in this pool).
- RAIDZ1 requires at least 3 disks per vdev, but can have as many total disks per vdev as you want!
- The number of disks in a vdev is referred to as its "**width**"; these are 4-wide Z1 vdevs.
- We can lose **up to 1 disk per Z1 vdev** (regardless of vdev width)



RAIDZ1 Vdev Example

= Parity Data

- Pool layout is similar to RAID 50 (RAID 5 + 0)
- Like RAID 5, we have one parity disks per vdev (and two vdevs in this pool).
- RAIDZ1 requires at least 3 disks per vdev, but can have as many total disks per vdev as you want!
- The number of disks in a vdev is referred to as its "**width**"; these are 4-wide Z1 vdevs.
- We can lose **up to 1 disk per Z1 vdev** (regardless of vdev width)
- If we lose more than 1 disk in a single vdev, the pool fails!

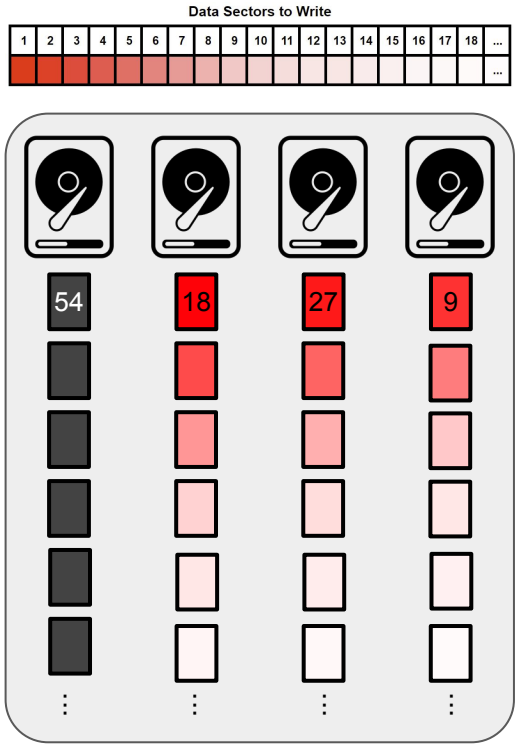


RAIDZ Aside: Parity

? = Parity Data

- What's this parity block? How do we use it to recover data after a disk fails?
- *"A function whose being even (or odd) **provides a check** on a set of binary values."*
- The parity data is calculated from the other data in vdev's the same row such that the calculation can be reversed easily.

- *Simple Example:*
  - Row 1 Col 2 data is **18**, Row 1 Col 3 data is **27**, Row 1 Col 4 data is **9**
  - **Parity Data** (Row 1 Col 1) could be ***54*** = *18 + 22 + 9*



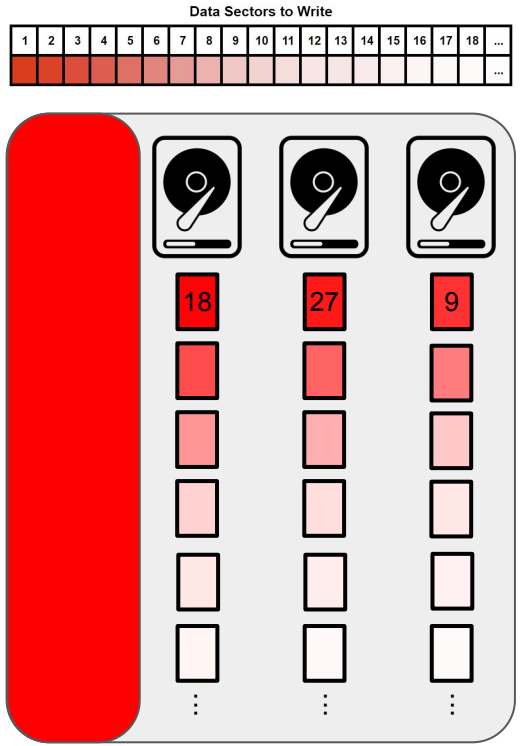
RAIDZ Aside: Parity

? = Parity Data

- What's this parity block? How do we use it to recover data after a disk fails?
- *"A function whose being even (or odd) **provides a check** on a set of binary values."*
- The parity data is calculated from the other data in vdev's the same row such that the calculation can be reversed easily.

- *Simple Example:*
  - Row 1 Col 2 data is **18**, Row 1 Col 3 data is **27**, Row 1 Col 4 data is **9**
  - **Parity Data** (Row 1 Col 1) could be ***54*** = *18 + 22 + 9*

- With this setup, here's how we could recover the data if a given disk fails:
  - **Lost disk 1**: Just recalculate all parity data



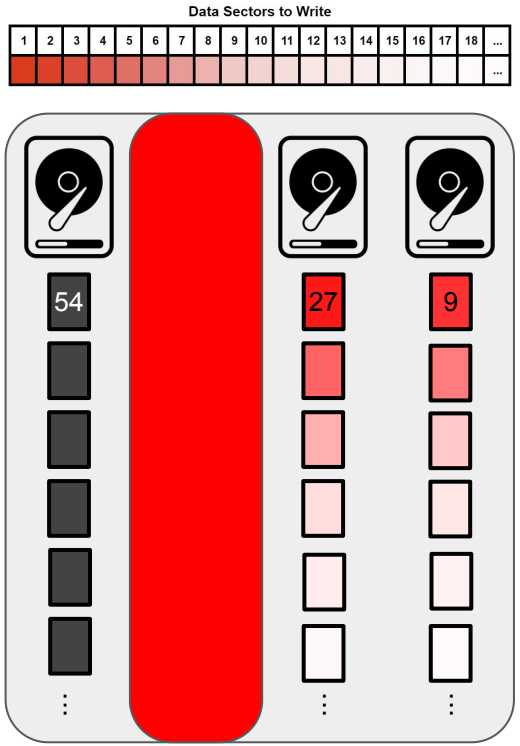
RAIDZ Aside: Parity

? = Parity Data

- What's this parity block? How do we use it to recover data after a disk fails?
- *"A function whose being even (or odd) **provides a check** on a set of binary values."*
- The parity data is calculated from the other data in vdev's the same row such that the calculation can be reversed easily.

- *Simple Example:*
  - Row 1 Col 2 data is **18**, Row 1 Col 3 data is **27**, Row 1 Col 4 data is **9**
  - **Parity Data** (Row 1 Col 1) could be ***54*** = *18 + 22 + 9*

- With this setup, here's how we could recover the data if a given disk fails:
  - **Lost disk 1**: Just recalculate all parity data
  - **Lost disk 2**: Solve *54 =* ***x*** *+ 22 + 9* to recover data in first row, repeat for all rows



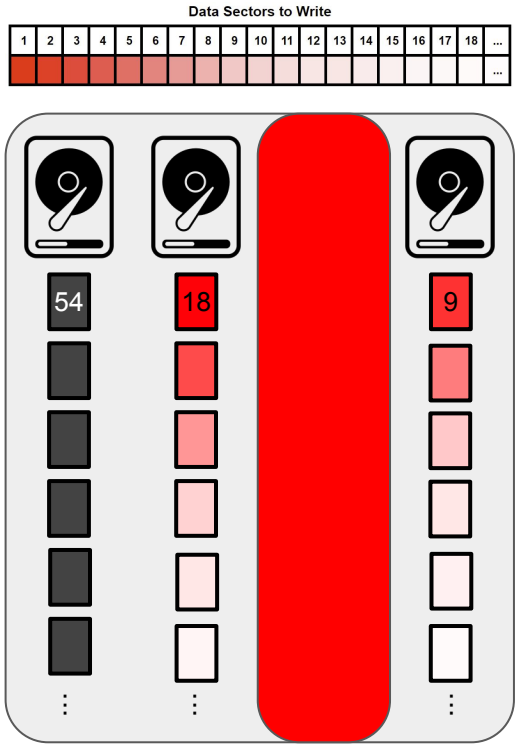
RAIDZ Aside: Parity

? = Parity Data

- What's this parity block? How do we use it to recover data after a disk fails?
- *"A function whose being even (or odd) **provides a check** on a set of binary values."*
- The parity data is calculated from the other data in vdev's the same row such that the calculation can be reversed easily.

- *Simple Example:*
  - Row 1 Col 2 data is **18**, Row 1 Col 3 data is **27**, Row 1 Col 4 data is **9**
  - **Parity Data** (Row 1 Col 1) could be ***54*** = *18 + 22 + 9*

- With this setup, here's how we could recover the data if a given disk fails:
  - **Lost disk 1**: Just recalculate all parity data
  - **Lost disk 2**: Solve *54 = **x** + 22 + 9* to recover data in first row, repeat for all rows
  - **Lost disk 3**: Solve *54 = 18 + **x** + 9* to recover data, repeat for all rows



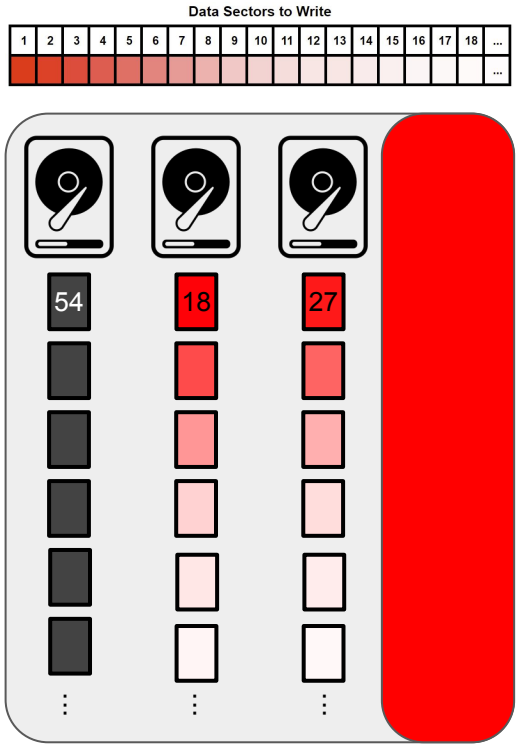
RAIDZ Aside: Parity

? = Parity Data

- What's this parity block? How do we use it to recover data after a disk fails?
- *"A function whose being even (or odd) **provides a check** on a set of binary values."*
- The parity data is calculated from the other data in vdev's the same row such that the calculation can be reversed easily.

- *Simple Example:*
  - Row 1 Col 2 data is **18**, Row 1 Col 3 data is **27**, Row 1 Col 4 data is **9**
  - **Parity Data** (Row 1 Col 1) could be ***54*** = *18 + 22 + 9*

- With this setup, here's how we could recover the data if a given disk fails:
  - **Lost disk 1**: Just recalculate all parity data
  - **Lost disk 2**: Solve *54 =* ***x*** *+ 22 + 9* to recover data in first row, repeat for all rows
  - **Lost disk 3**: Solve *54 = 18 +* ***x*** *+ 9* to recover data, repeat for all rows
  - **Lost disk 4**: Solve *54 = 18 + 22 +* ***x*** to recover data, repeat for all rows



RAIDZ Aside: Parity

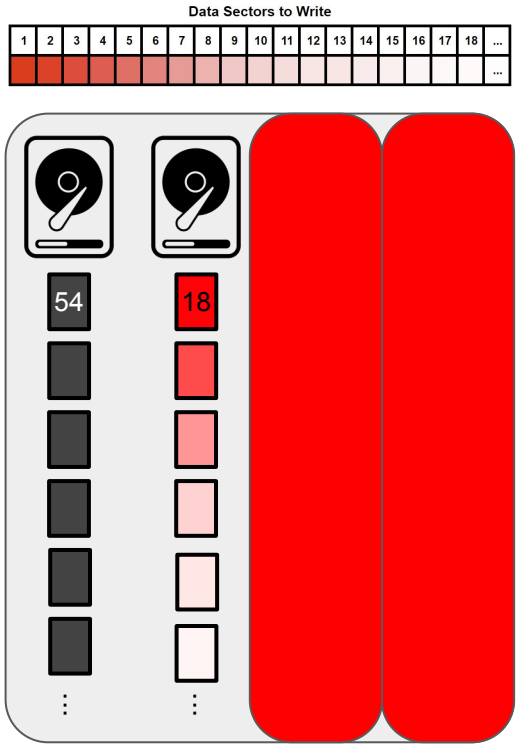
? = Parity Data

- What's this parity block? How do we use it to recover data after a disk fails?
- *"A function whose being even (or odd) **provides a check** on a set of binary values."*
- The parity data is calculated from the other data in vdev's the same row such that the calculation can be reversed easily.

- *Simple Example:*
  - Row 1 Col 2 data is **18**, Row 1 Col 3 data is **27**, Row 1 Col 4 data is **9**
  - **Parity Data** (Row 1 Col 1) could be ***54*** = *18 + 22 + 9*

- With this setup, here's how we could recover the data if a given disk fails:
  - **Lost disk 1**: Just recalculate all parity data
  - **Lost disk 2**: Solve *54 = **x** + 22 + 9* to recover data in first row, repeat for all rows
  - **Lost disk 3**: Solve *54 = 18 + **x** + 9* to recover data, repeat for all rows
  - **Lost disk 4**: Solve *54 = 18 + 22 + **x*** to recover data, repeat for all rows
  - **Lost 2+ disks:** Too many unknowns in the equation! We can't solve *54 = 18 + **x** + **y*** 😭

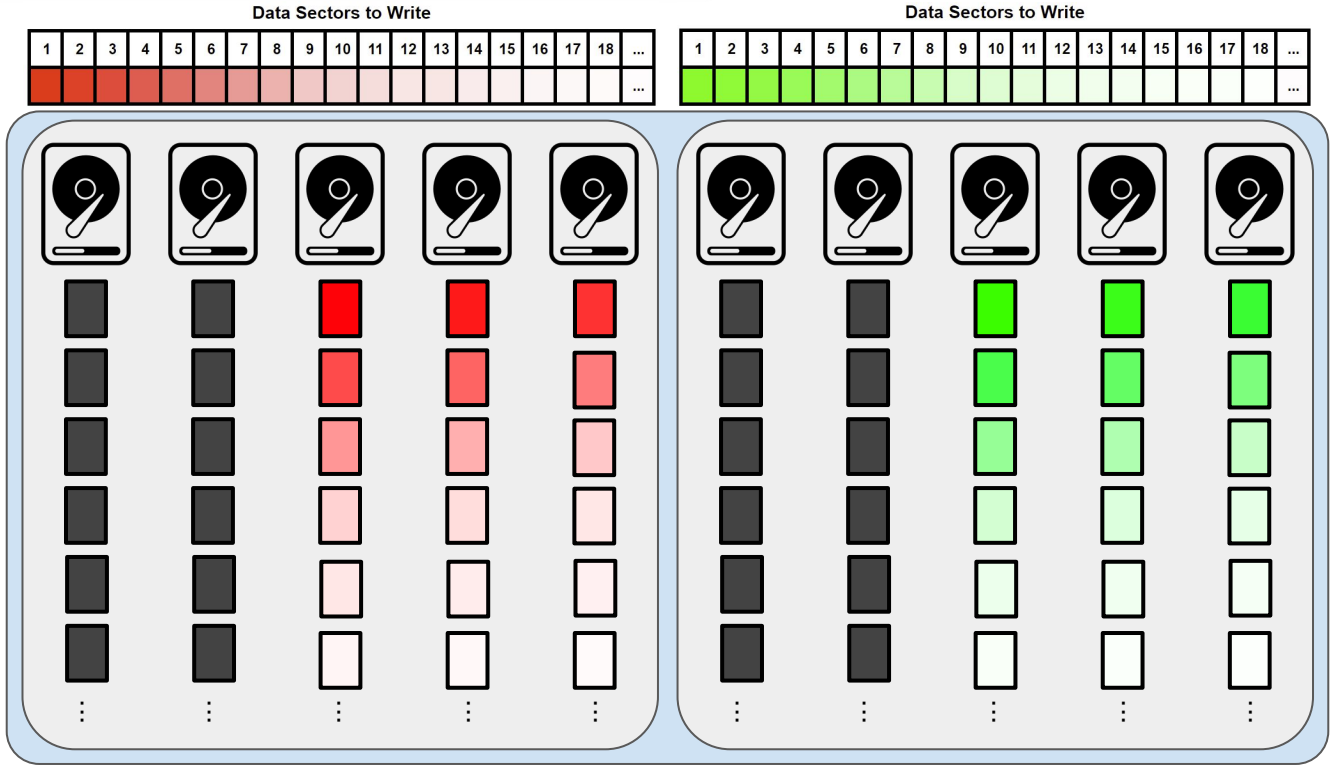
- *Note: Parity calculations **don't actually use addition** because the answer can take up more space when stored on the disk than its inputs. RAIDZ1 uses the "exclusive or" or **XOR function**, Z2 and Z3 use something more complicated called **Reed-Solomon coding***



RAIDZ2 Vdev Example

= Parity Data

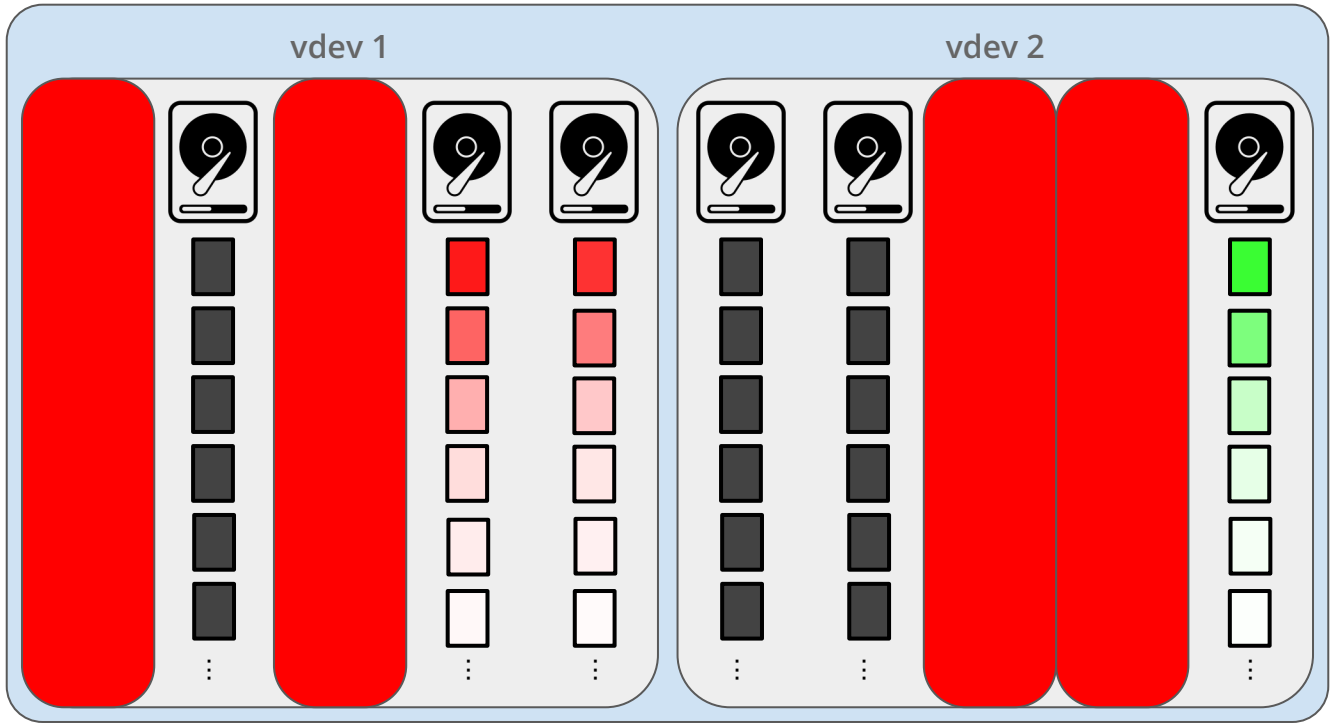
- Pool layout is similar to RAID 60 (RAID 6 + 0)
- Like RAID 6, we have **two parity disks** per vdev (and two vdevs in this pool).



RAIDZ2 Vdev Example

= Parity Data

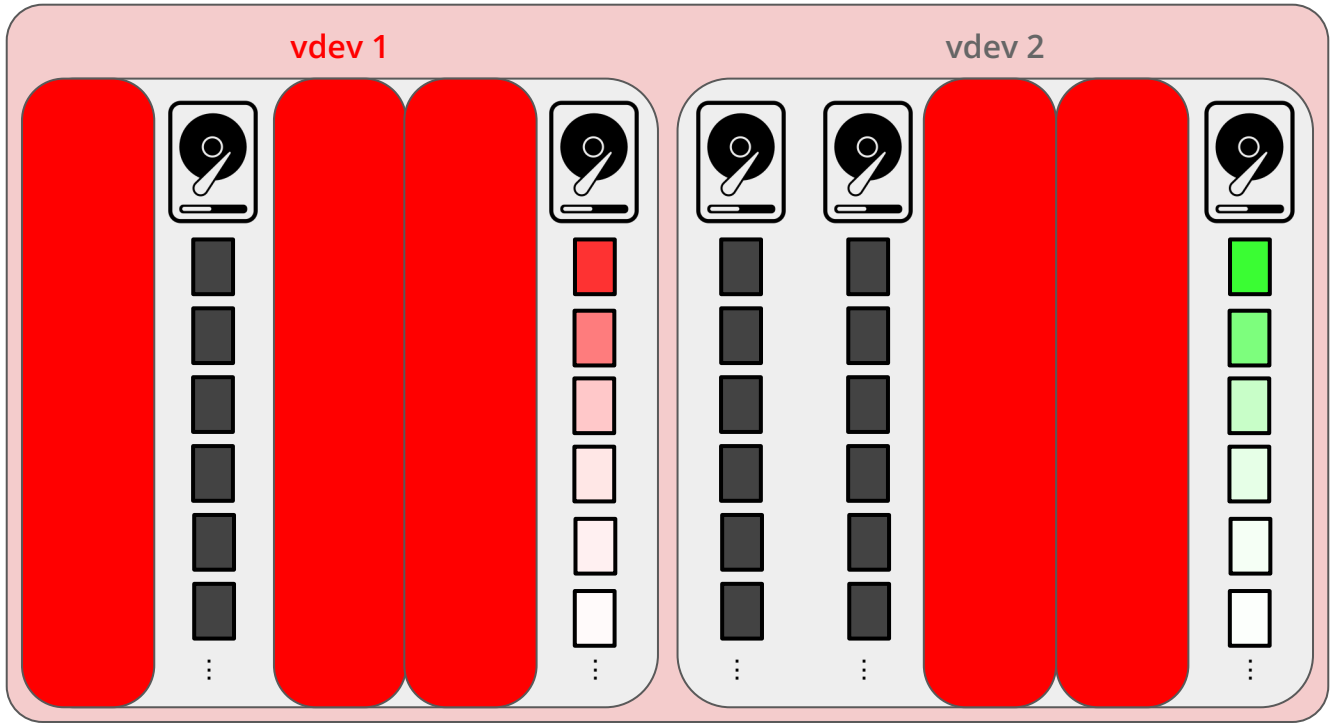
- Pool layout is similar to RAID 60 (RAID 6 + 0)
- Like RAID 6, we have **two parity disks** per vdev (and two vdevs in this pool).
- **Can lose up to two disks per vdev** and still maintain data integrity.



RAIDZ2 Vdev Example

= Parity Data

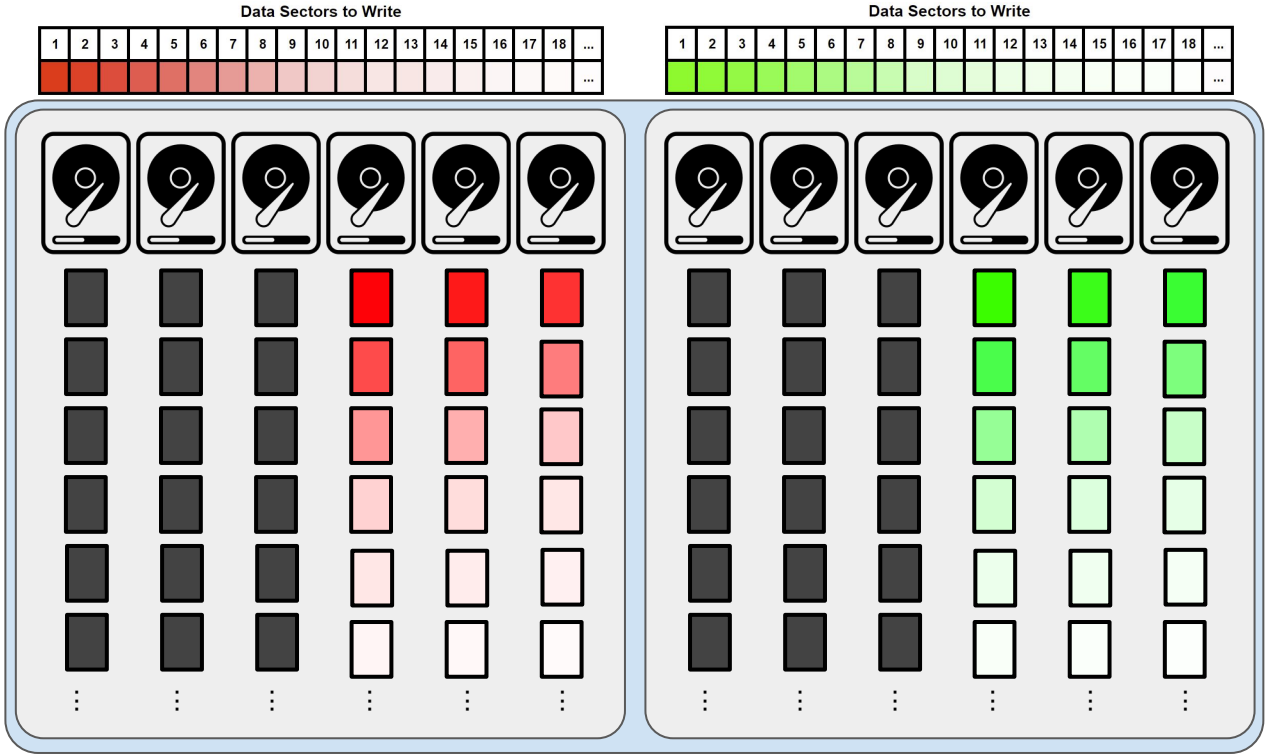
- Pool layout is similar to RAID 60 (RAID 6 + 0)
- Like RAID 6, we have **two parity disks** per vdev (and two vdevs in this pool).
- **Can lose up to two disks per vdev** and still maintain data integrity.
- If three (or more) disks are lost in a single vdev, all pool data is lost!



RAIDZ3 Vdev Example

= Parity Data

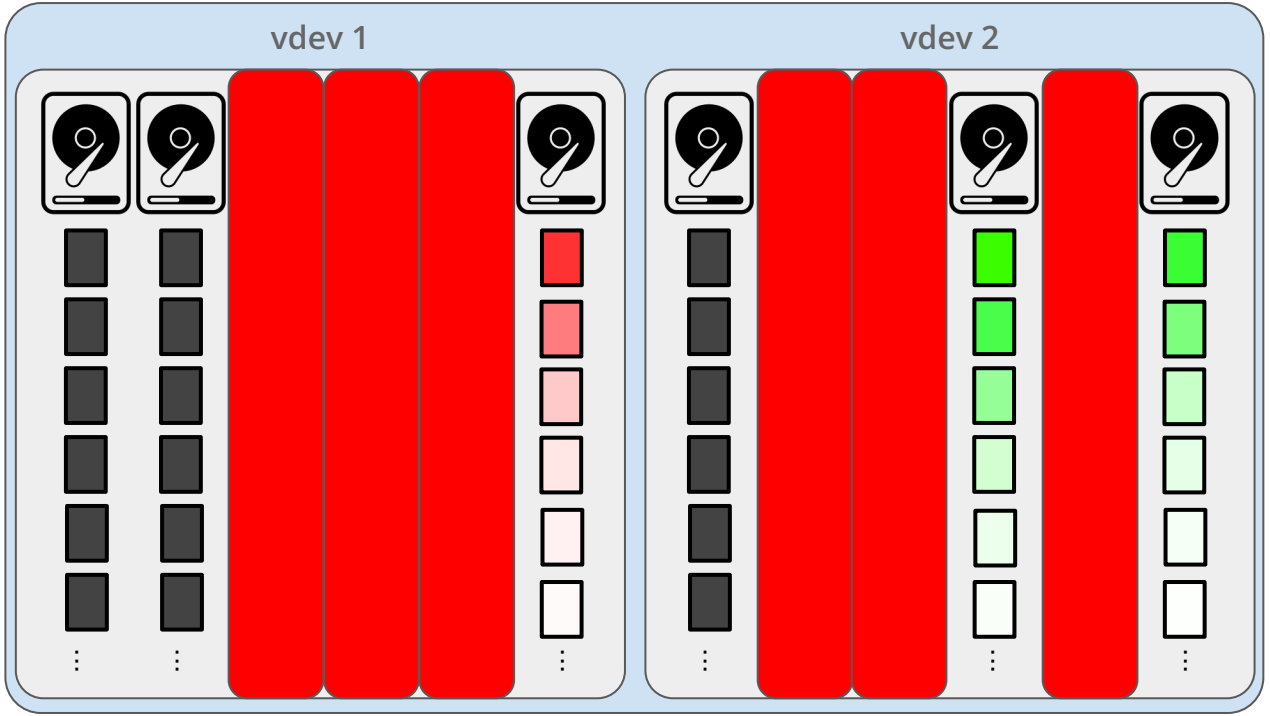
- We have **three parity disks** per vdev (and two vdevs in this pool).
- “Normal” doesn’t have an equivalent configuration



RAIDZ3 Vdev Example

= Parity Data

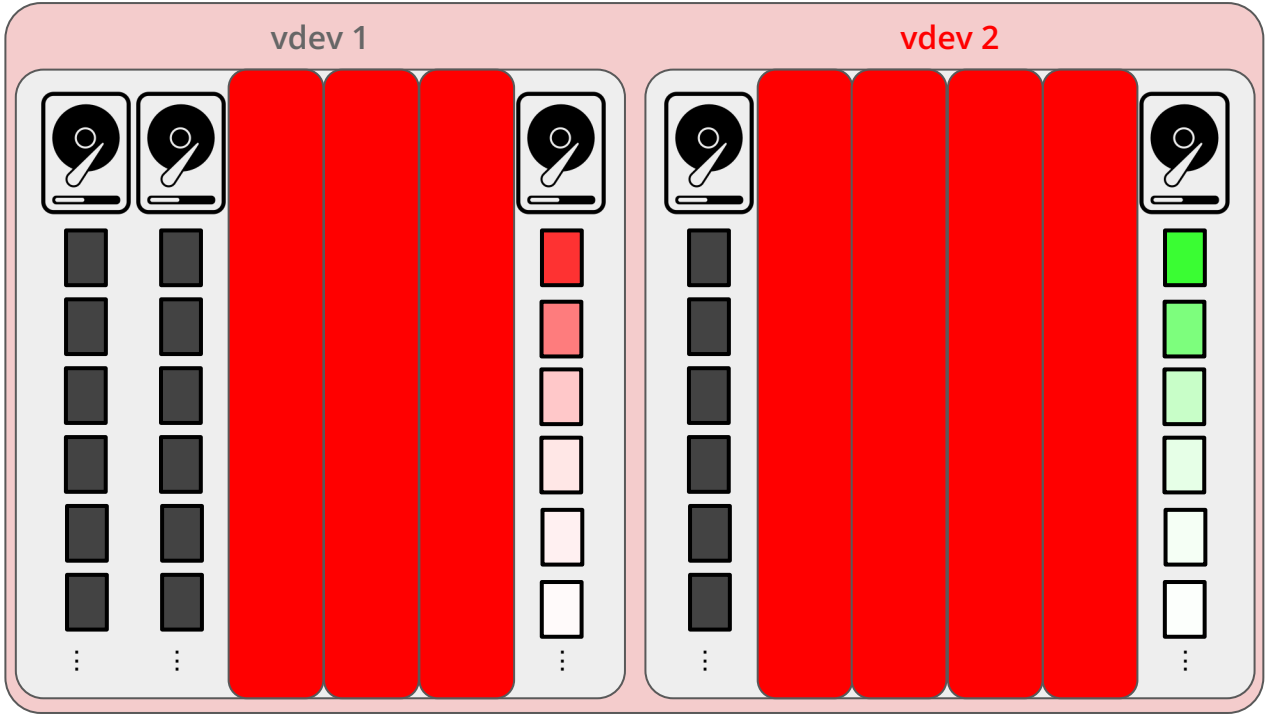
- We have **three parity disks** per vdev (and two vdevs in this pool).
- "Normal" doesn't have an equivalent configuration
- **Can lose up to three disks per vdev** and still maintain data integrity.



RAIDZ3 Vdev Example

= Parity Data

- We have **three parity disks** per vdev (and two vdevs in this pool).
- "Normal" doesn't have an equivalent configuration
- **Can lose up to three disks per vdev** and still maintain data integrity.
- If four (or more) disks are lost in a single vdev, all pool data is lost!



RAID Z***p***:

We can lose up to ***p*** disks per vdev and be okay!

RAID Z**1**: Can lose **1 disk** per vdev

RAID Z**2**: Can lose **2 disks** per vdev

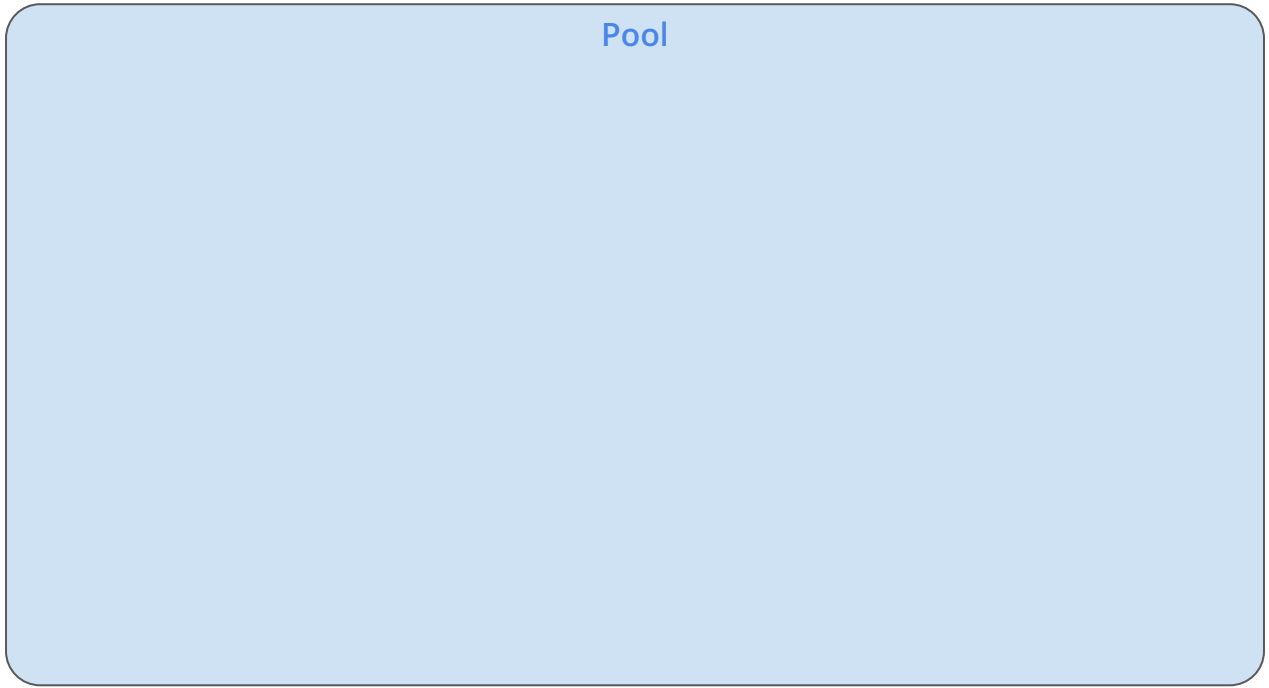
RAID Z**3**: Can lose **3 disks** per vdev

ZFS Datasets & ZVols

- Now that we have our pool, we can store data on it. We have a couple different types of “containers” in which we can store the data: **datasets** and **zvols**
- **Datasets** contain file systems with folders and files that you can browse through. This is what you would typically think of when you store data on a hard drive.
  - A dataset acts kind of like a folder that contains all your stuff, but you can set more detailed attributes on it, like capacity limits, compression type, dedupe, sync settings, etc.
  - You can nest datasets (i.e., a dataset inside another dataset) just like you can nest normal folders. The “child” dataset can have different settings than the parent (e.g., no compression).
  - File-sharing protocols like SMB, AFP, and NFS will work out of datasets.
- **Zvols** are simply a chunk of raw disk space. ZFS just keeps track of the 1s and 0s on it.
  - Zvols are created inside a dataset but don’t appear as a file or folder in that dataset; they just take up a chunk of disk space.
  - The ZFS system doesn’t know what files are on the zvol, it’s just tracking the 1s and 0s
  - Block protocols like iSCSI and FC use zvols; client will create the file system on the raw space.

ZFS Pools, Vdevs, Datasets, & Zvols

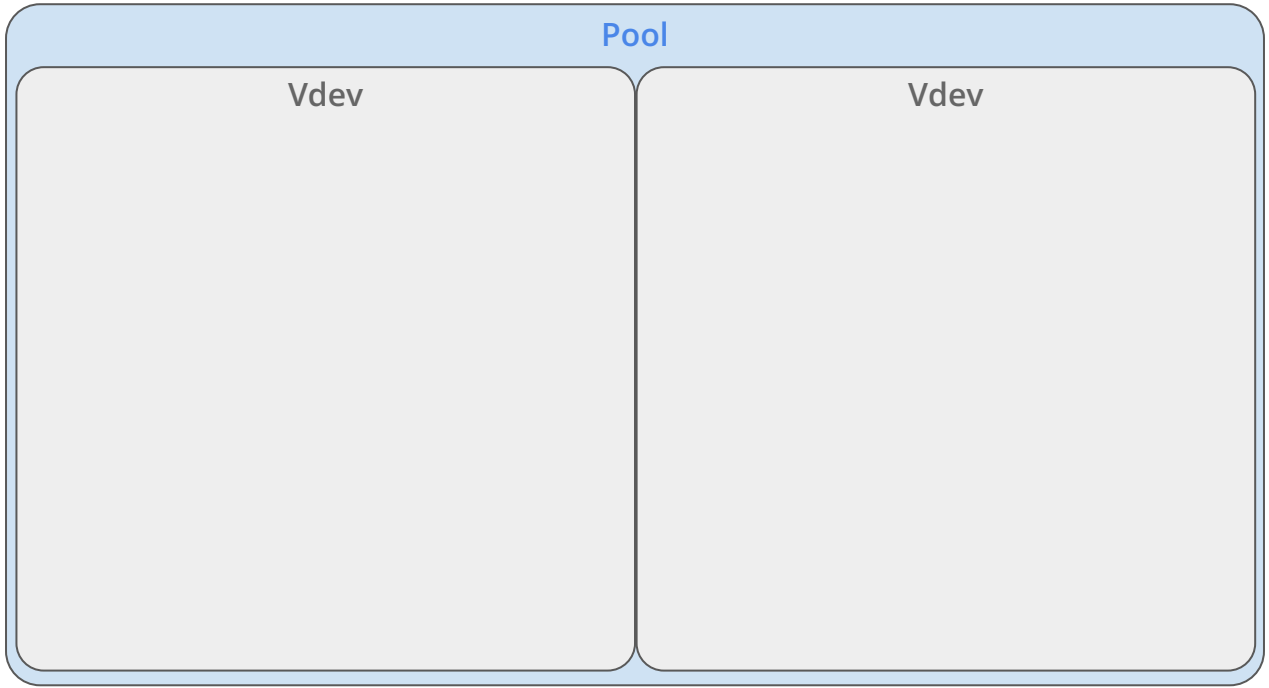
- Let's review all the building blocks we just covered and look at how they all fit together in a ZFS system:
- At the top of the hierarchy, we have our pool



ZFS Pools, Vdevs, Datasets, & Zvols

- Let's review all the building blocks we just covered and look at how they all fit together in a ZFS system:

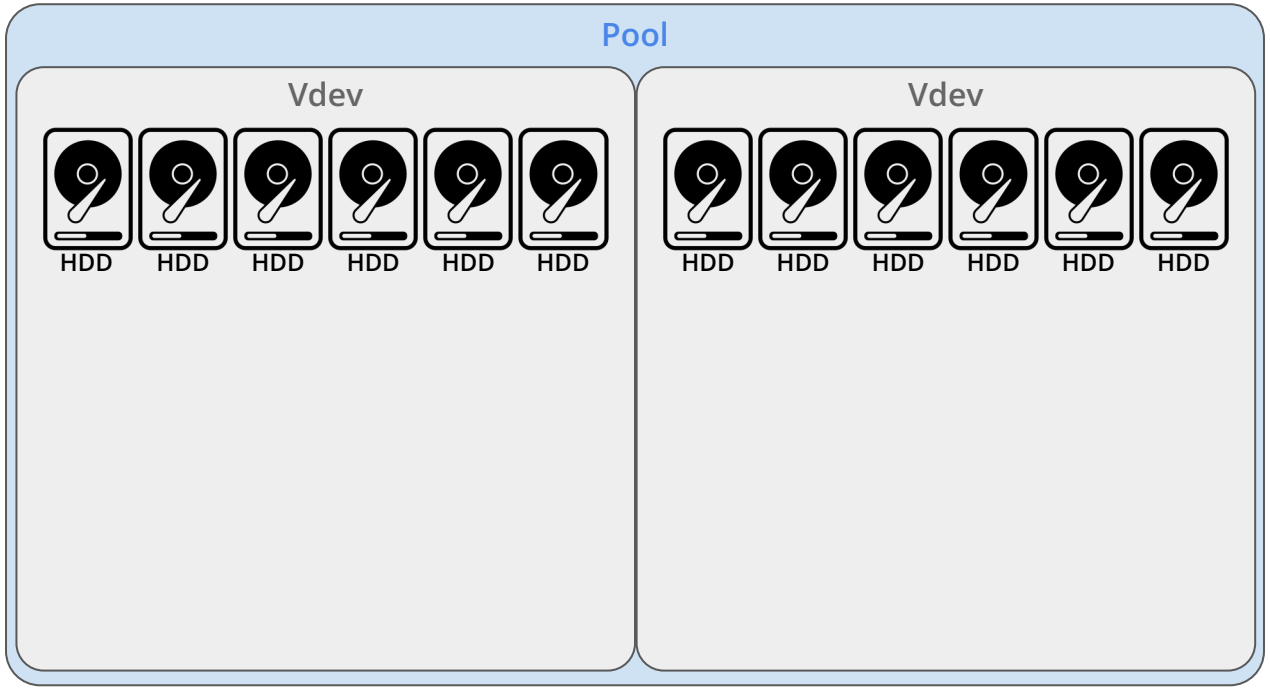
- At the top of the hierarchy, we have our pool
- The pool is made up of one or more vdevs



ZFS Pools, Vdevs, Datasets, & Zvols

- Let's review all the building blocks we just covered and look at how they all fit together in a ZFS system:

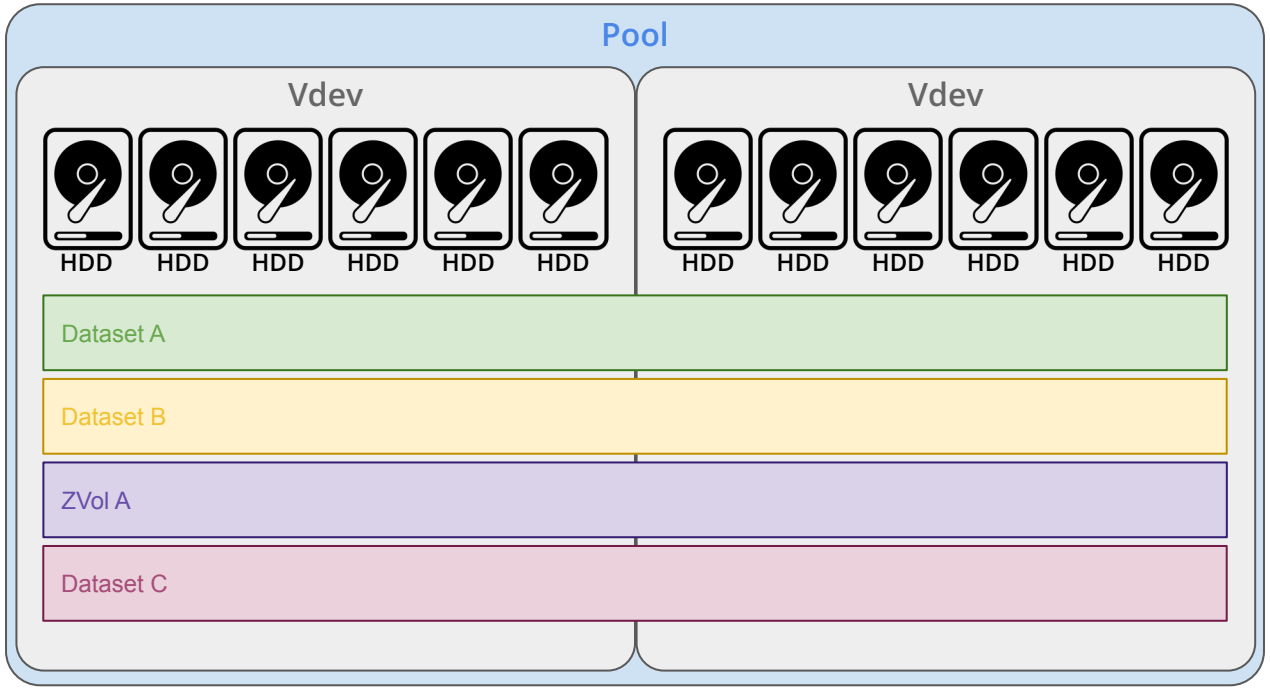
- At the top of the hierarchy, we have our pool
- The pool is made up of one or more vdevs
- Each vdev consists of one or more hard drives grouped together in some configuration



ZFS Pools, Vdevs, Datasets, & Zvols

- Let's review all the building blocks we just covered and look at how they all fit together in a ZFS system:

- At the top of the hierarchy, we have our pool
- The pool is made up of one or more vdevs
- Each vdev consists of one or more hard drives grouped together in some configuration
- Datasets & Zvols are striped across all the Vdevs



ZFS Pools, Vdevs, Datasets, & Zvols

- Let's review all the building blocks we just covered and look at how they all fit together in a ZFS system:

- At the top of the hierarchy, we have our pool
- The pool is made up of one or more vdevs
- Each vdev consists of one or more hard drives grouped together in some configuration
- Datasets & Zvols are striped across all the Vdevs
- Datasets can be repeatedly nested inside of other datasets
- Zvols can be nested in datasets, but nothing can be nested in a Zvol
- All of these children and grandchildren are also striped across all the drives