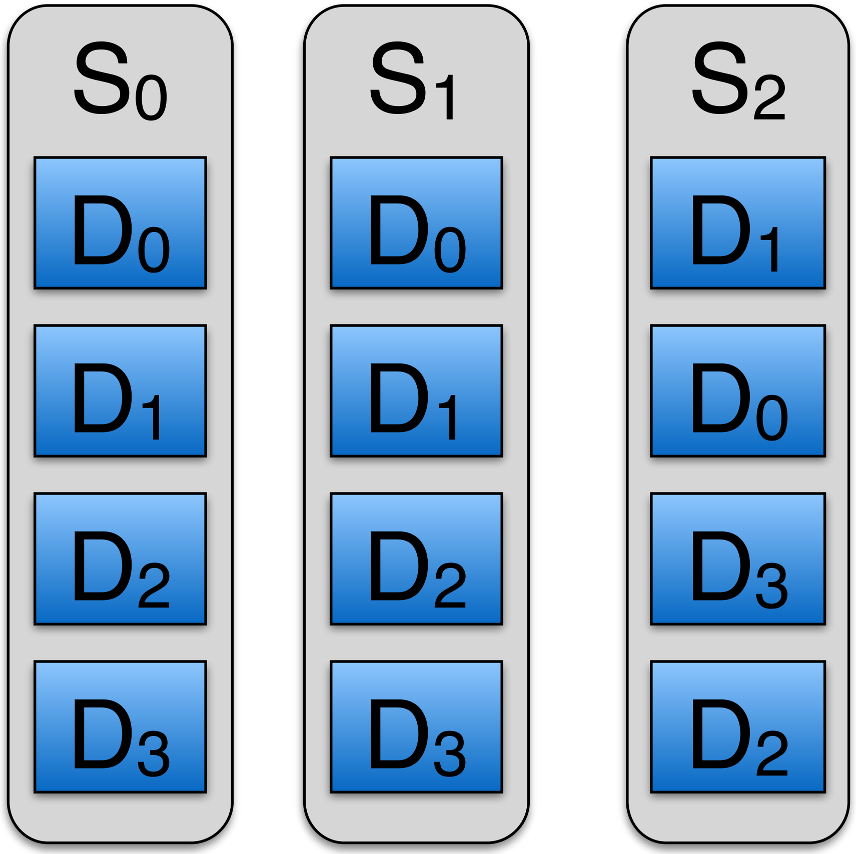
The Basics of Erasure Codes for Archival Storage

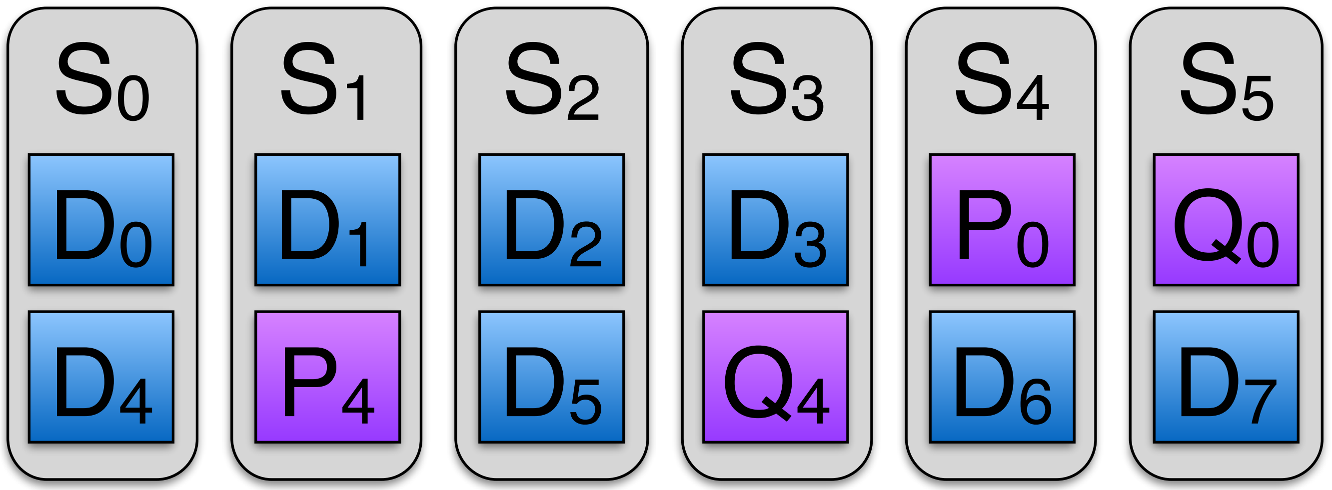
Ethan L. Miller
Symantec Presidential Chair in Storage & Security
Center for Research in Storage Systems
University of California, Santa Cruz

Protecting archival data

- Mirroring: keep multiple copies
  - All copies are identical
  - Read *any* copy
  - Write *all* copies

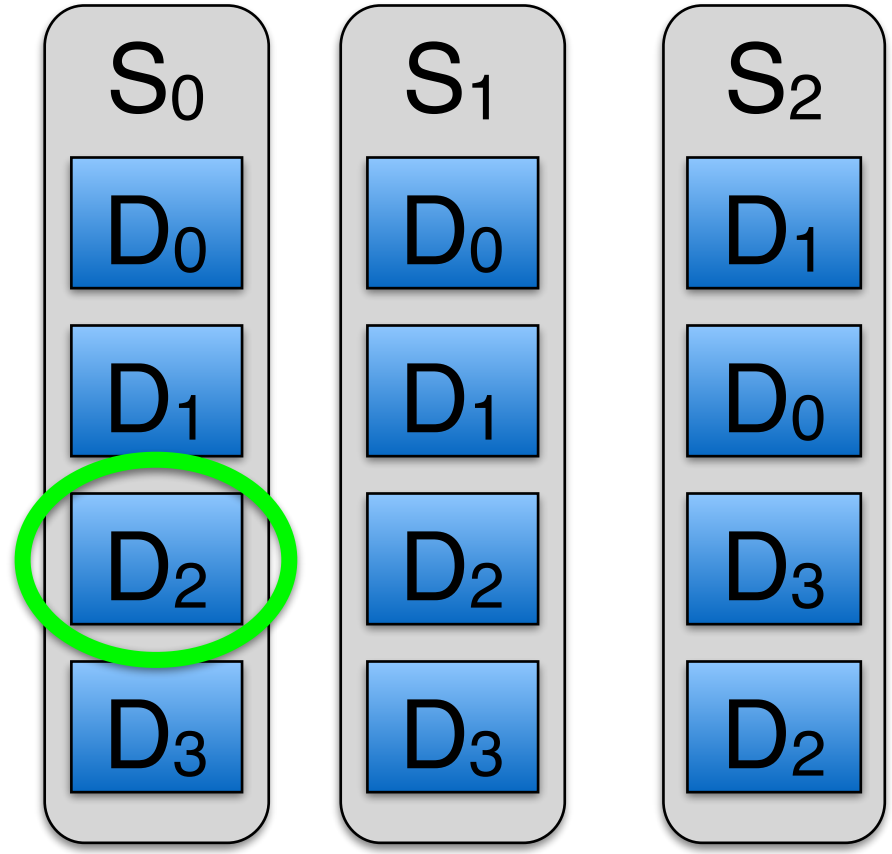


- Erasure codes: use multiple "chunks" for redundancy
  - Read from desired chunk(s)
  - Write is more complex
    - Write an entire stripe, or
    - Read-Modify-Write to update data and parity
  - Lower storage overhead!

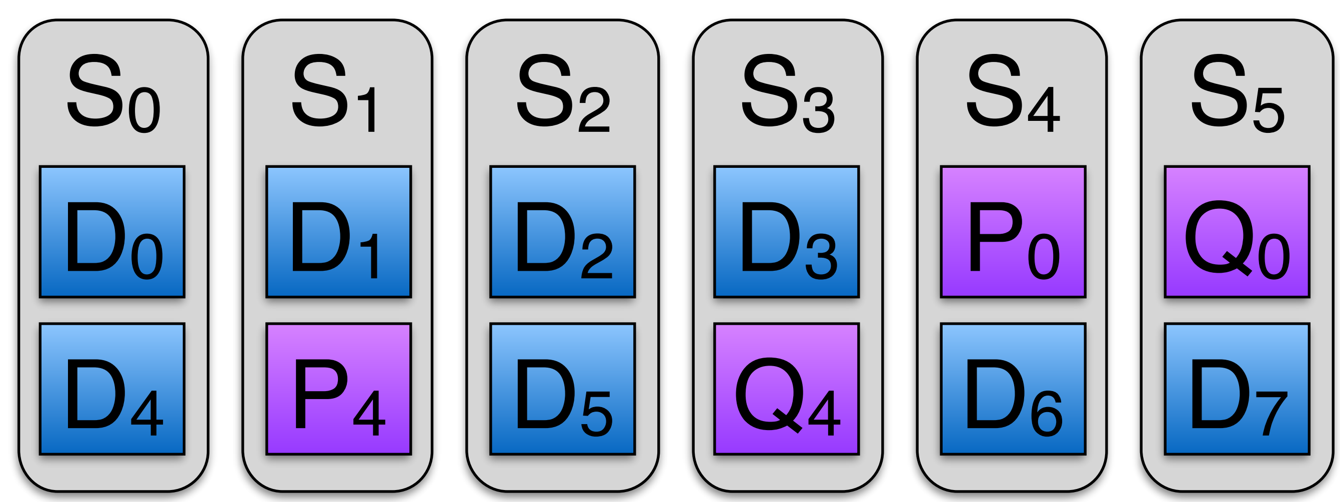


Protecting archival data

- Mirroring: keep multiple copies
  - All copies are identical
  - Read *any* copy
  - Write *all* copies

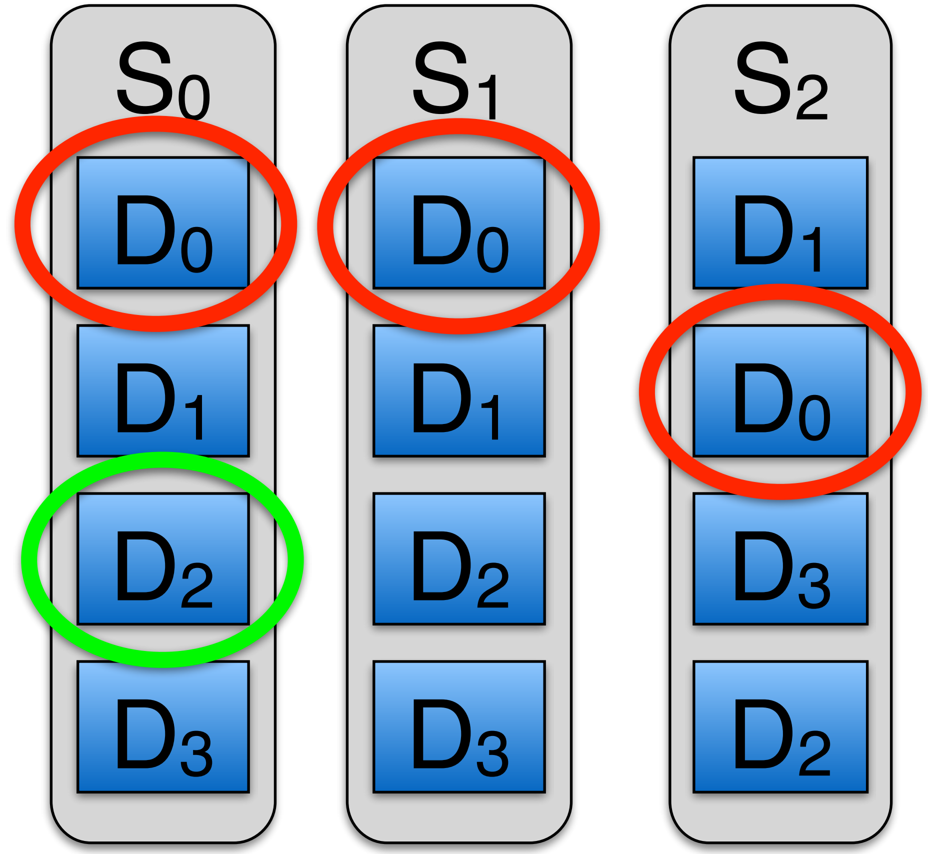


- Erasure codes: use multiple "chunks" for redundancy
  - Read from desired chunk(s)
  - Write is more complex
    - Write an entire stripe, or
    - Read-Modify-Write to update data and parity
  - Lower storage overhead!

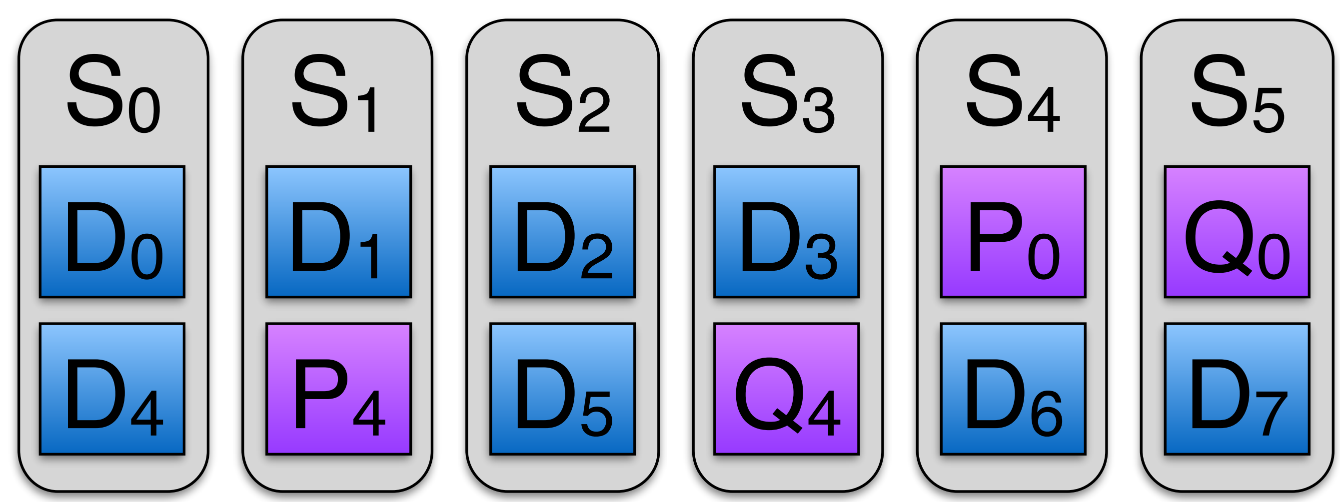


Protecting archival data

- Mirroring: keep multiple copies
  - All copies are identical
  - Read *any* copy
  - Write *all* copies

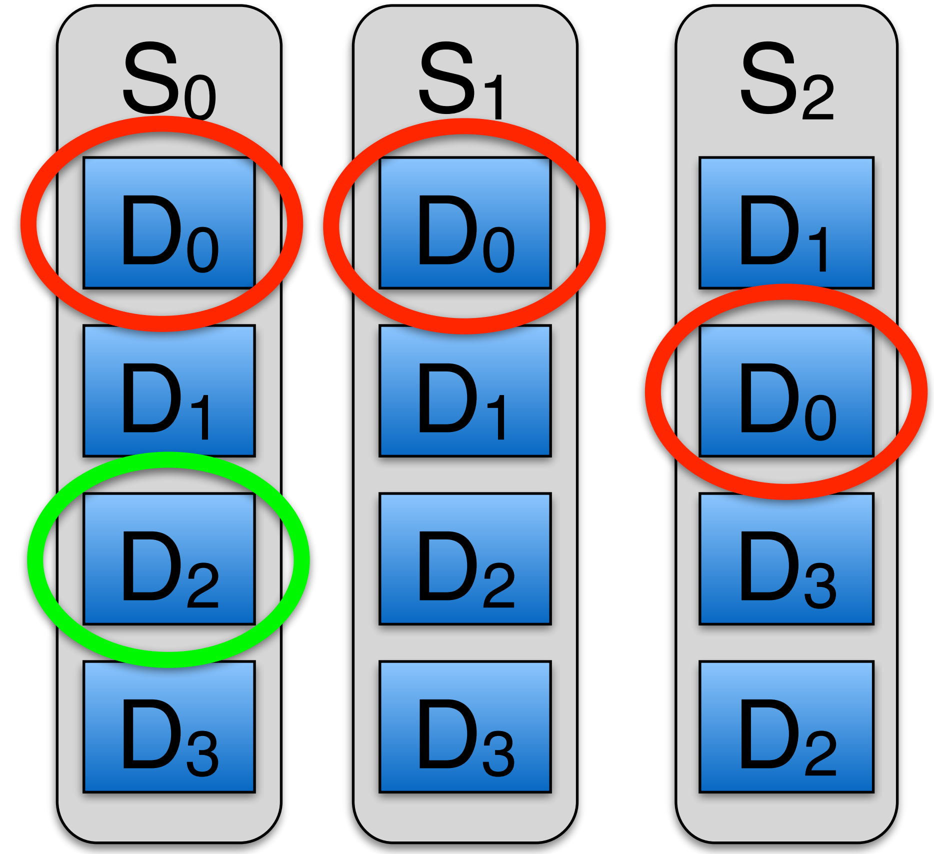


- Erasure codes: use multiple "chunks" for redundancy
  - Read from desired chunk(s)
  - Write is more complex
    - Write an entire stripe, or
    - Read-Modify-Write to update data and parity
  - Lower storage overhead!

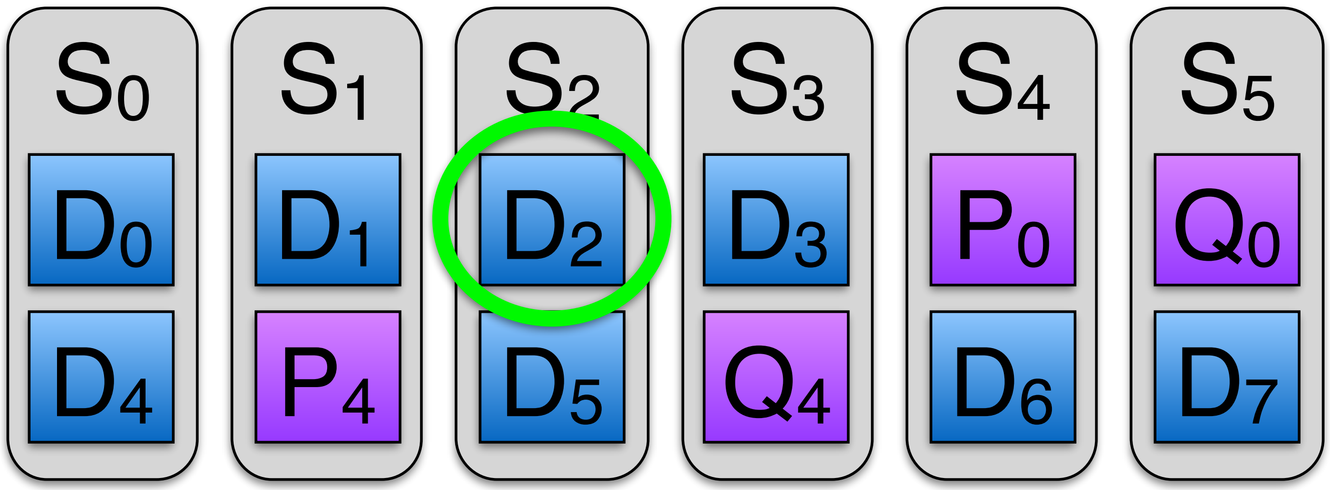


Protecting archival data

- Mirroring: keep multiple copies
  - All copies are identical
  - Read *any* copy
  - Write *all* copies

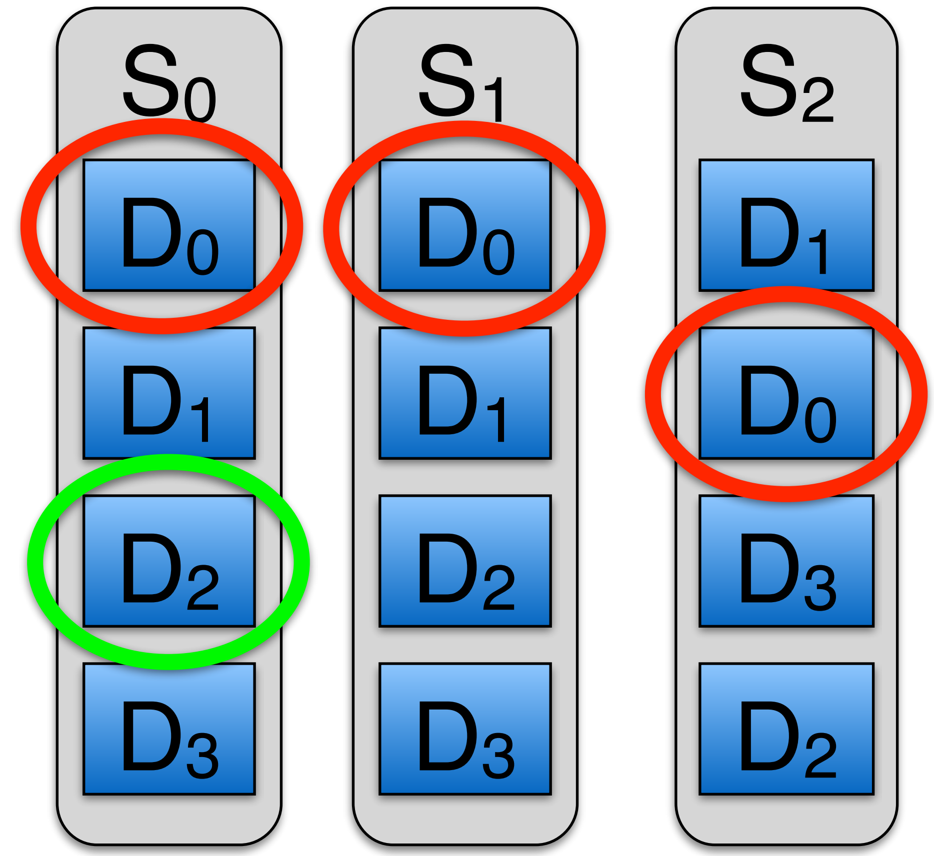


- Erasure codes: use multiple “chunks” for redundancy
  - Read from desired chunk(s)
  - Write is more complex
    - Write an entire stripe, or
    - Read-Modify-Write to update data and parity
  - Lower storage overhead!

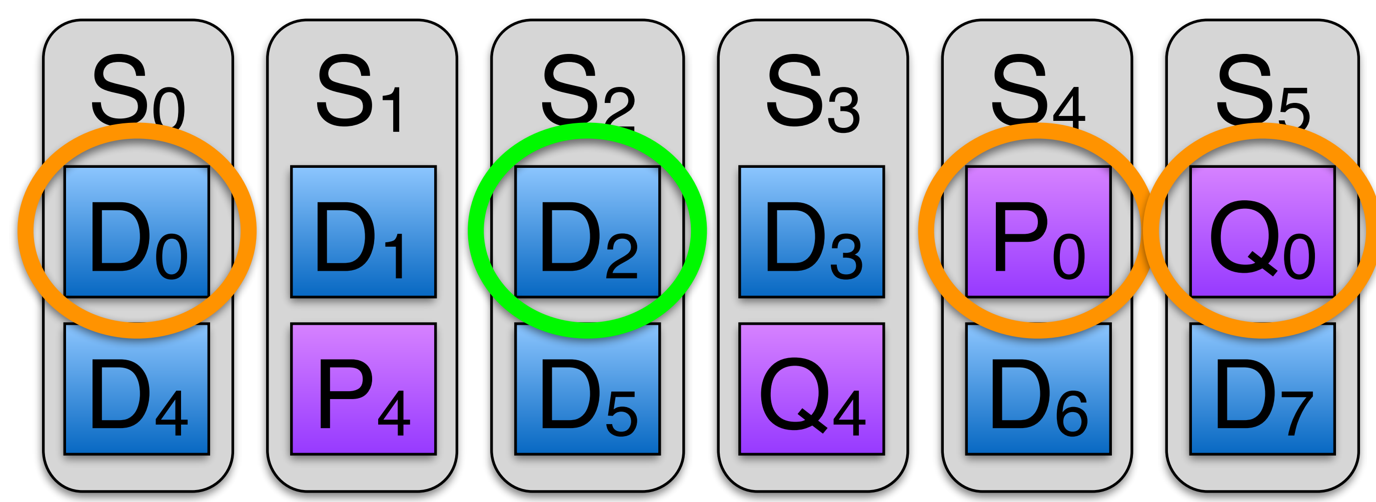


Protecting archival data

- Mirroring: keep multiple copies
  - All copies are identical
  - Read *any* copy
  - Write *all* copies

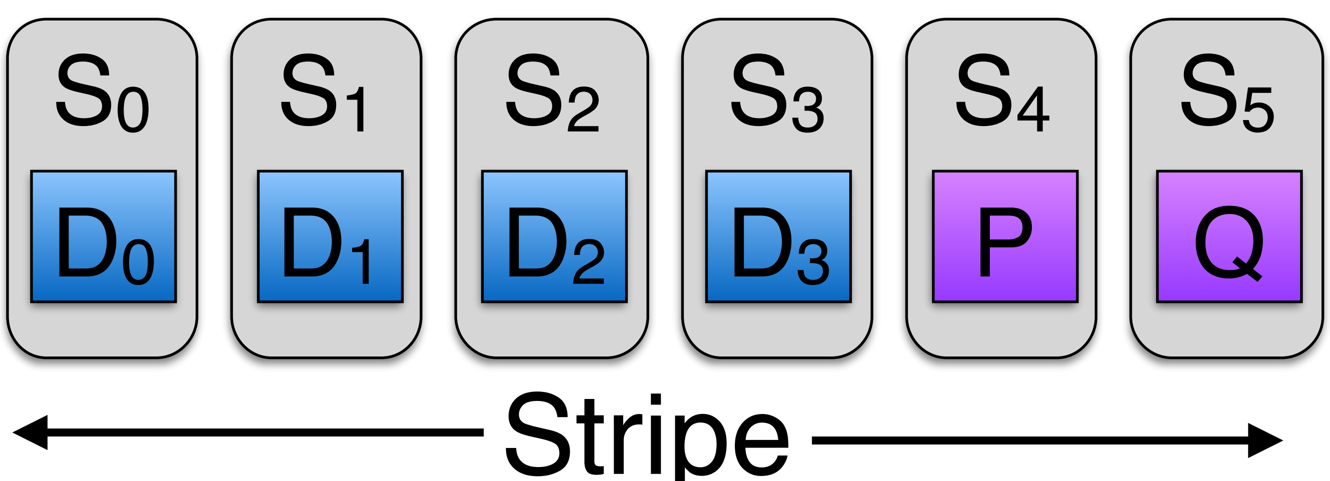


- Erasure codes: use multiple "chunks" for redundancy
  - Read from desired chunk(s)
  - Write is more complex
    - Write an entire stripe, or
    - Read-Modify-Write to update data and parity
  - Lower storage overhead!



How do erasure codes work?

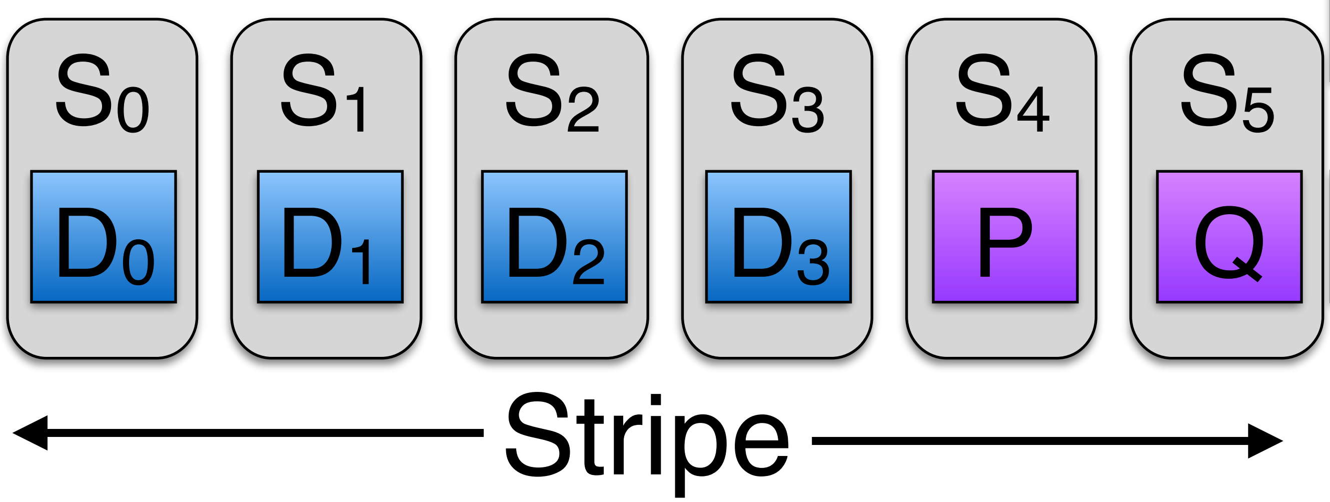
- Error correcting code: find the error *and* rebuild it
- Erasure correcting code: given the broken (erased location), rebuild it
  - Most archival storage systems are this type
  - Use hashing to figure out which chunks are broken
- Erasure correcting codes use linear equations to rebuild missing data
  - Each symbol in the "row" has its own equation
  - *n* data values & *m* erasure correcting symbols
  - *n* data values ➡ need *n* equations to rebuild



$$D_0 = D_0$$
$$D_1 = D_1$$
$$D_2 = D_2$$
$$D_3 = D_3$$
$$D_0 \oplus D_1 \oplus D_2 \oplus D_3 = P$$
$$D_0 \oplus 2 \otimes D_1 \oplus 4 \otimes D_2 \oplus 8 \otimes D_3 = Q$$

How do erasure codes work?

- Error correcting code: find the error *and* rebuild it
- Erasure correcting code: given the broken (erased location), rebuild it
  - Most archival storage systems are this type
  - Use hashing to figure out which chunks are broken
- Erasure correcting codes use linear equations to rebuild missing data
  - Each symbol in the “row” has its own equation
  - *n* data values & *m* erasure correcting symbols
  - *n* data values ➡ need *n* equations to rebuild



$D_0 = D_0$

$D_1 = D_1$

$D_2 = D_2$

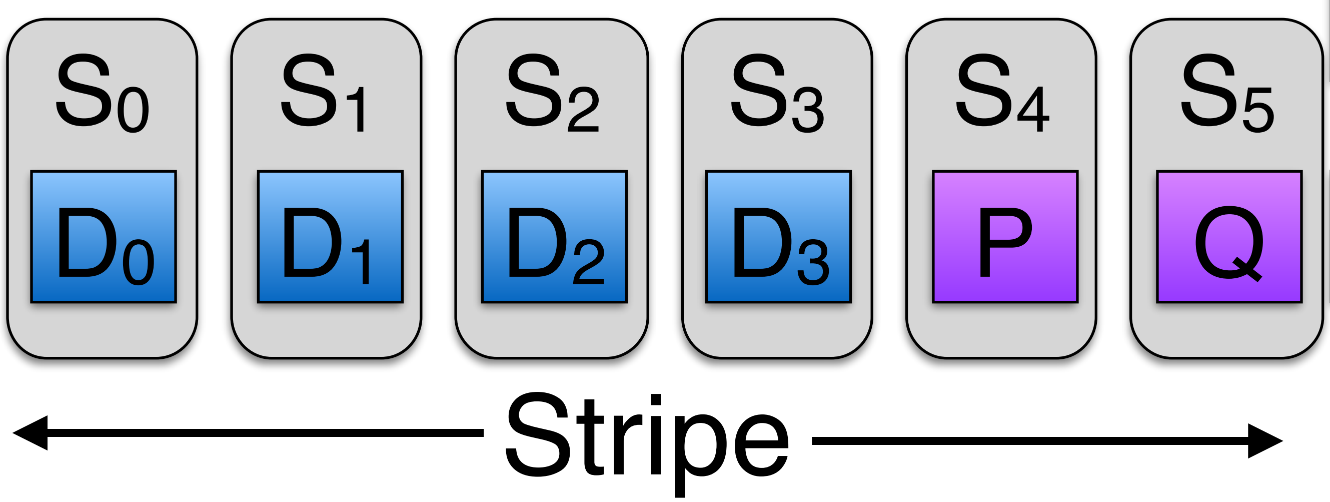
$D_3 = D_3$

$D_0 \oplus D_1 \oplus D_2 \oplus D_3 = P$

$D_0 \oplus 2 \otimes D_1 \oplus 4 \otimes D_2 \oplus 8 \otimes D_3 = Q$

How do erasure codes work?

- Error correcting code: find the error *and* rebuild it
- Erasure correcting code: given the broken (erased location), rebuild it
  - Most archival storage systems are this type
  - Use hashing to figure out which chunks are broken
- Erasure correcting codes use linear equations to rebuild missing data
  - Each symbol in the “row” has its own equation
  - *n* data values & *m* erasure correcting symbols
  - *n* data values ➡ need *n* equations to rebuild



$$D_0 = D_0$$

$$D_1 = D_1$$

$$D_2 = D_2$$

$$D_3 = D_3$$

$$D_0 \oplus D_1 \oplus D_2 \oplus D_3 = P$$

$$D_0 \oplus 2 \otimes D_1 \oplus 4 \otimes D_2 \oplus 8 \otimes D_3 = Q$$

Solve for D_1 and D_3

The math behind erasure codes

- Erasure codes rely on *finite fields* (also called *Galois fields*)
  - Add and multiply defined on fixed-width elements (usually 8 or 16 bits for erasure codes)
  - Normal “arithmetic” rules apply
- This math can be done very quickly
  - Addition is XOR
  - Multiplication is more complex, but runs at gigabytes per second on modern CPUs
- Number of multiplications is usually the limiting factor
  - There are usually shortcuts for creating the original symbols
  - Rebuilding missing symbols (data) usually needs more multiplications

Reliability and erasure codes

- Long-term survival of data depends on several factors
  - How many failures the erasure code can survive
  - How much data is impacted by a failure
  - How long it takes to restore protection after a failure
- Systems can vary any of these parameters to decrease the likelihood of data loss
  - Fast rebuild ➡ less likely that too many failures will accumulate
  - Tolerate more failures ➡ can rebuild more slowly and still be safe
  - Decluster (spread data around) ➡ multiple independent failures unlikely to place large amounts of data in jeopardy
- Estimates of data reliability based on
  - Analytical modeling
  - Simulations

Evaluating erasure codes

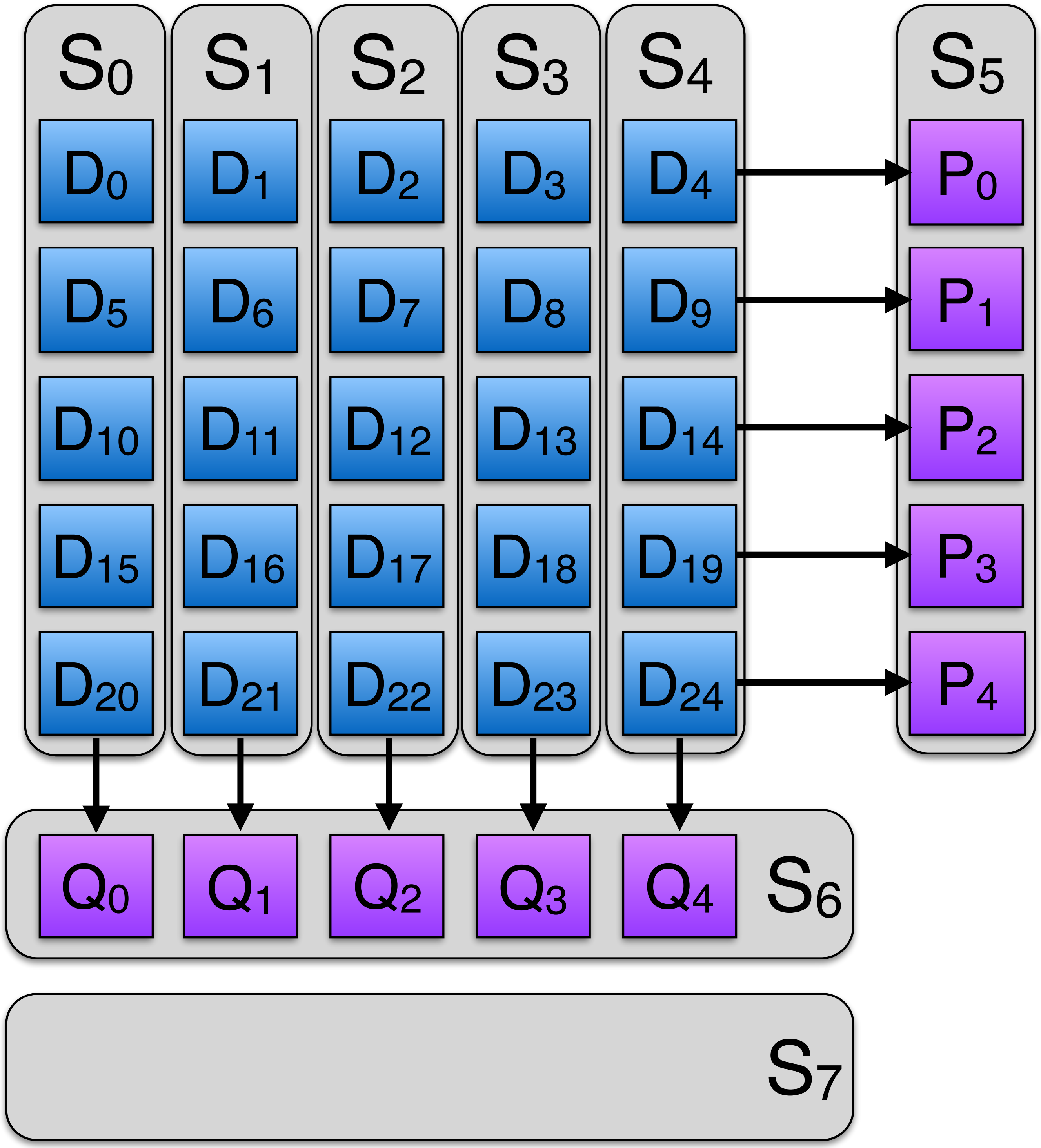
- Erasure codes can differ in many ways
  - Number of data storage devices in a "stripe"
    - Often, different stripes span different subsets of storage devices
  - Number and type of failures that can be tolerated
  - Amount of overhead
  - Number of devices that must be written (or read) for
    - Normal case
    - Single device failure
    - Larger-scale failures
  - Complexity of generating the parity symbols
  - Complexity of rebuilding missing data from surviving information
- Evaluating erasure codes is all about tradeoffs
  - If there were such a thing as a perfect erasure code, we'd all use it!
  - Choice of erasure code depends on what *you* want from it

Reed-Solomon codes

- Reed-Solomon is the most common erasure code
  - RAID-5 (*N*+1 parity) is a special case of Reed-Solomon
  - RAID-6 (P+Q parity) is also a type of Reed-Solomon
- Reed-Solomon can be generated *very* quickly for 1–2 parities
  - XOR is essentially
  - Multiplication by 2 is extremely fast, and is all that's needed for Q
- RS with more parity is a bit slower
  - Each successive parity symbol is a linear combination of all of the data symbols in the stripe
- Rebuilding missing data is more expensive
  - Invert an $N \times N$ matrix (once, so not so bad)
  - Rebuild missing symbol using dot product of *N* existing symbols and one row of the matrix

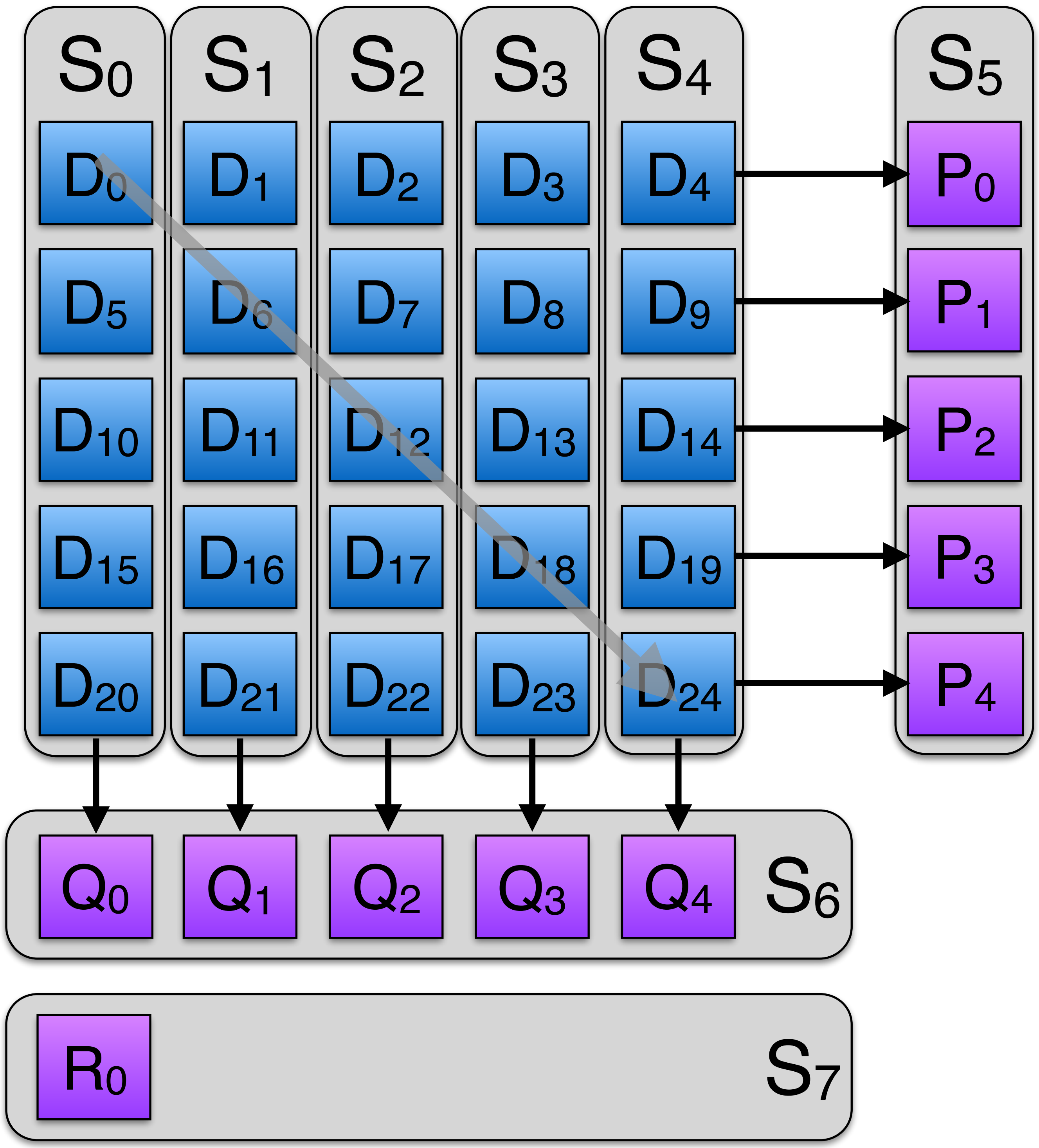
XOR-based codes

- XOR is cheap ➡ just use XOR!
- Typically use bigger basic chunks on each device
  - Example: run XORs across and diagonally
- There can be arbitrarily complex approaches to this
  - Paper on STAIR codes in FAST 2014
  - Survive combinations of failed devices and failed individual chunks
  - May combine XOR and multiplication



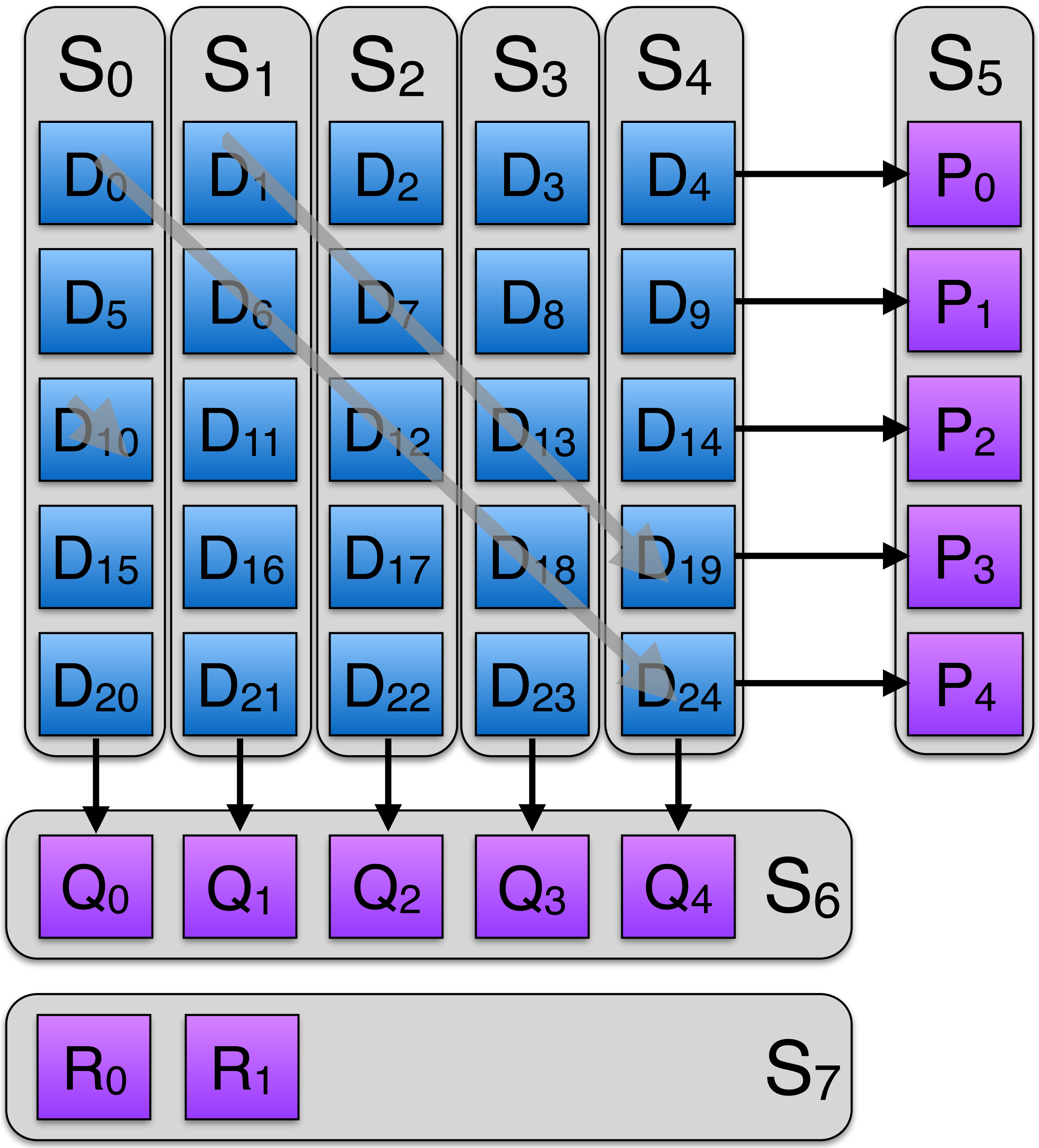
XOR-based codes

- XOR is cheap ➡ just use XOR!
- Typically use bigger basic chunks on each device
  - Example: run XORs across and diagonally
- There can be arbitrarily complex approaches to this
  - Paper on STAIR codes in FAST 2014
  - Survive combinations of failed devices and failed individual chunks
  - May combine XOR and multiplication



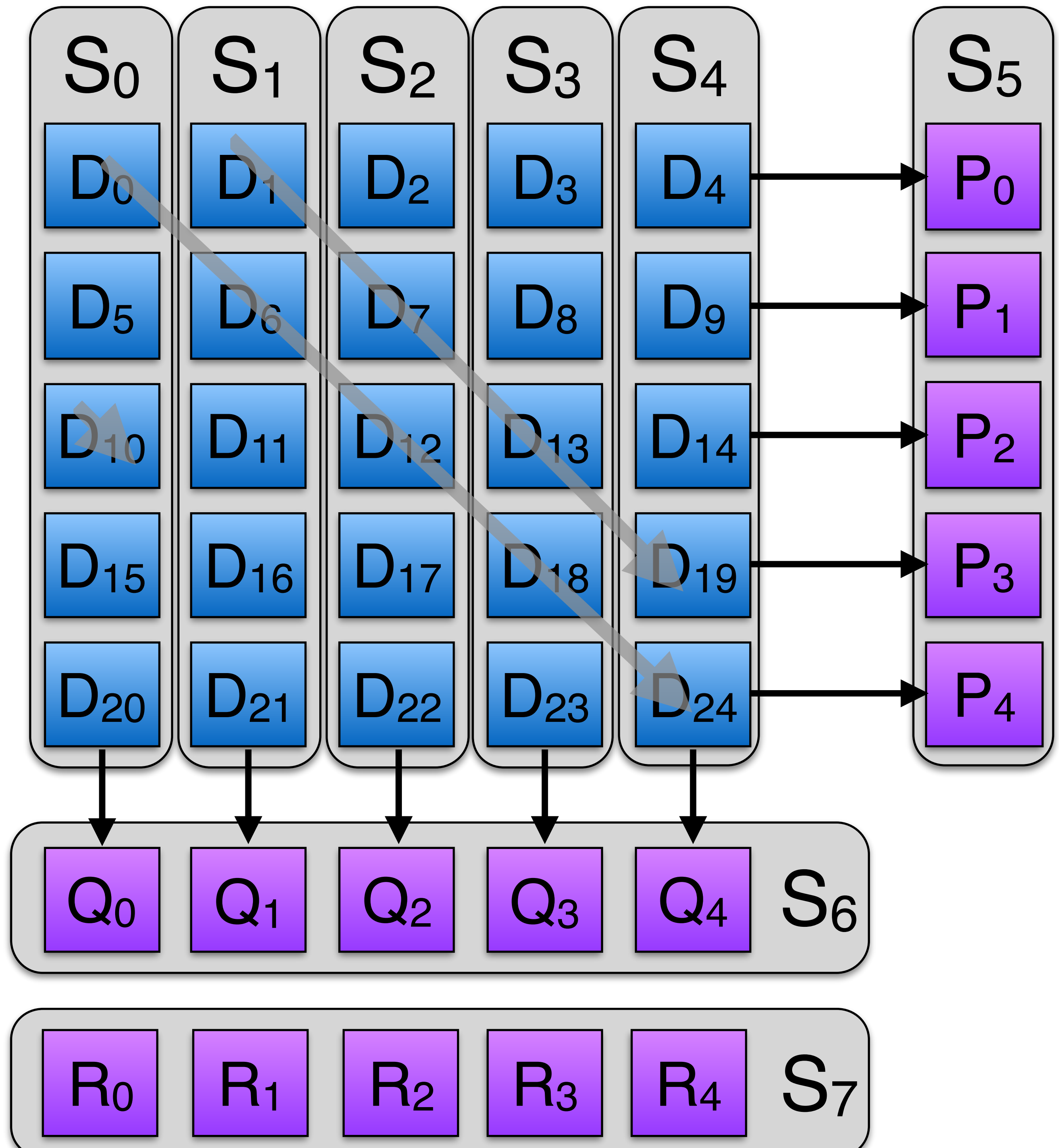
XOR-based codes

- XOR is cheap ➡ just use XOR!
- Typically use bigger basic chunks on each device
  - Example: run XORs across and diagonally
- There can be arbitrarily complex approaches to this
  - Paper on STAIR codes in FAST 2014
  - Survive combinations of failed devices and failed individual chunks
  - May combine XOR and multiplication



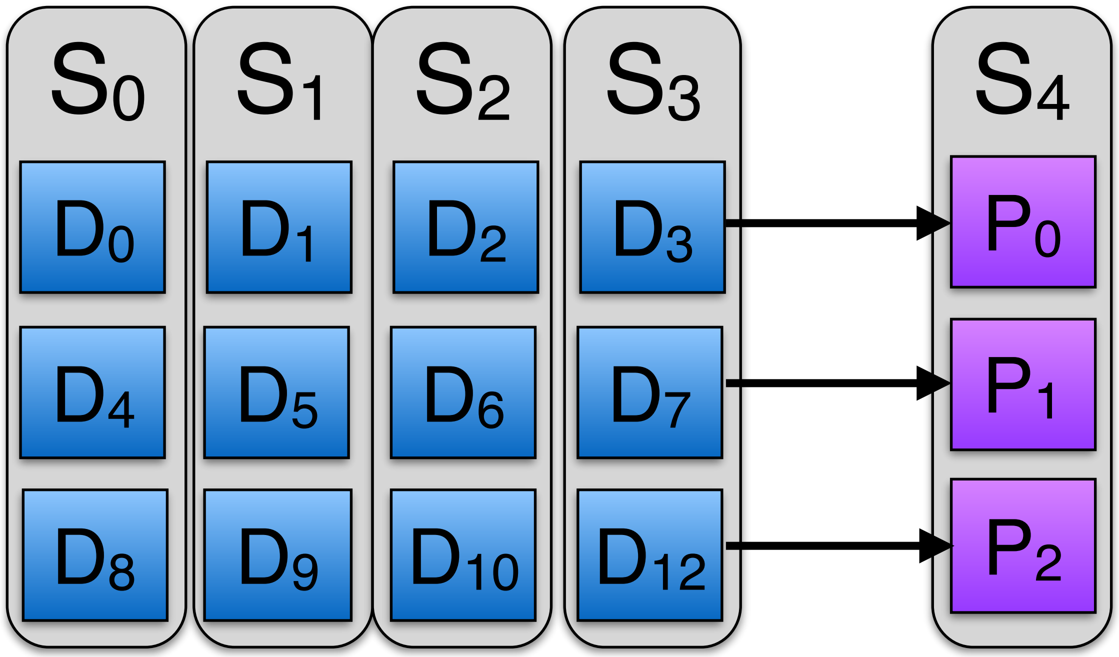
XOR-based codes

- XOR is cheap ➡ just use XOR!
- Typically use bigger basic chunks on each device
  - Example: run XORs across and diagonally
- There can be arbitrarily complex approaches to this
  - Paper on STAIR codes in FAST 2014
  - Survive combinations of failed devices and failed individual chunks
  - May combine XOR and multiplication



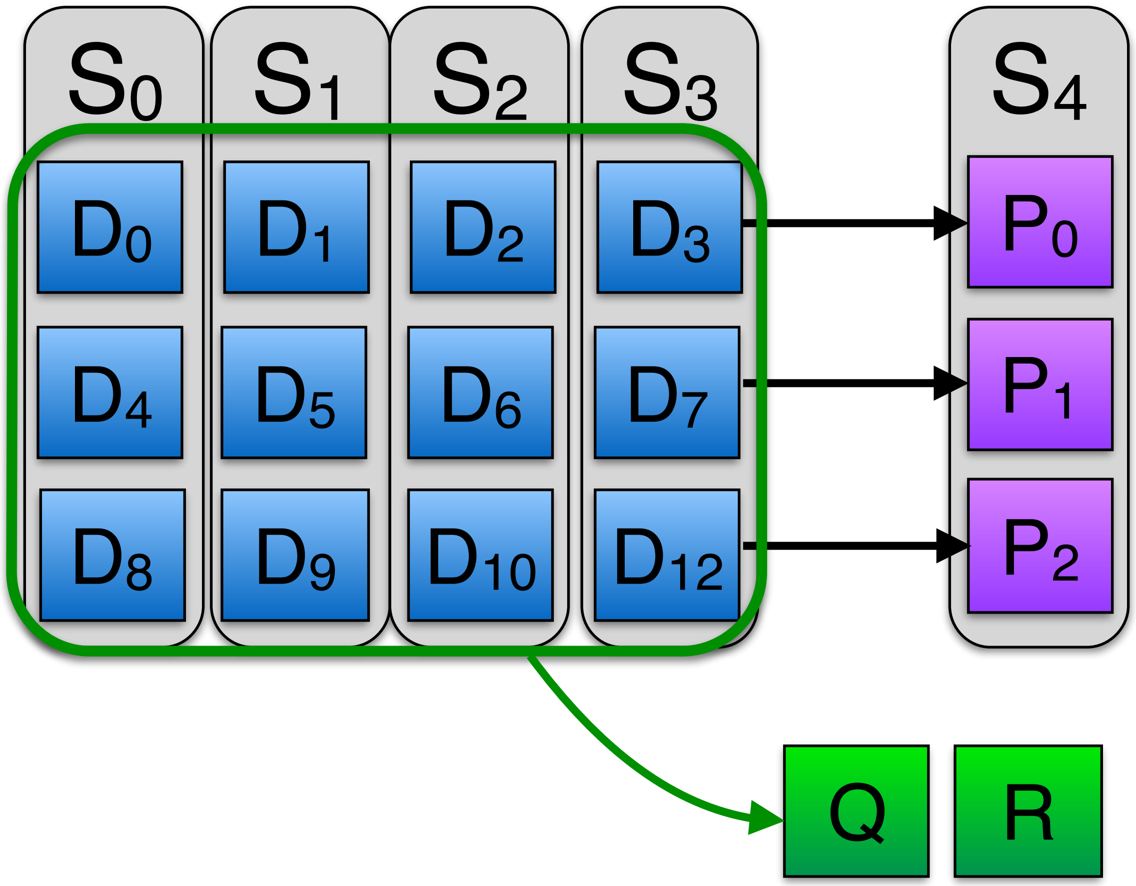
Hierarchical (pyramid, LRC) codes

- Differential RAID coverage
  - N+1 RAID (usually) for smallish groups
  - More parity covering multiple N+1 RAID groups
- Fast recovery from small failures
- Ability to recover (more slowly) from larger-scale failures
  - Additional parity uses multiplication
  - Additional parity is across a larger set of data elements
- Survives more failures with lower overhead



Hierarchical (pyramid, LRC) codes

- Differential RAID coverage
  - N+1 RAID (usually) for smallish groups
  - More parity covering multiple N+1 RAID groups
- Fast recovery from small failures
- Ability to recover (more slowly) from larger-scale failures
  - Additional parity uses multiplication
  - Additional parity is across a larger set of data elements
- Survives more failures with lower overhead



Distributed around the system

Regenerating codes

- Combine multiple data symbols on a single device ➡ rebuilding requires reading fewer devices
  - Less network traffic for rebuilding
  - Still survives same number of failures
  - May need to read more from a device just to return data
  - Need to read ***more*** from each device, in most cases!
- Can require more multiplications for most operations
  - Basic (non-failure) data reads
  - Reconstruction
- May be useful in deep archive: access fewer devices to rebuild
  - Performance of reads is often less important

So what should you ask?

- Common cases:
  - How long does a “regular” data read take?
  - How long does it take if a device has failed and the data is on the device?
  - How much does rebuilding impact performance even for non-affected data?
- How is data distributed across the devices?
  - Typically declustered: each stripe uses a different set of devices
- How many device failures can the erasure code withstand?
  - How long does it take to rebuild *all* of the data from a lost device, and where does it get rebuilt?
  - How likely is it that data will be lost, and how much data would be lost?
- Is the bottleneck due to computation, network or I/O?
  - Typically, computation is easiest to overcome
  - I/O is often the hardest, especially for archival systems
- There are a lot more variations on erasure codes than we could cover today!

Questions?

We're happy to answer questions about erasure codes!

elm@cs.ucsc.edu

http://www.ssrc.ucsc.edu/