StripeFinder

Erasure Coding of Small Objects over Key-Value Storage Devices

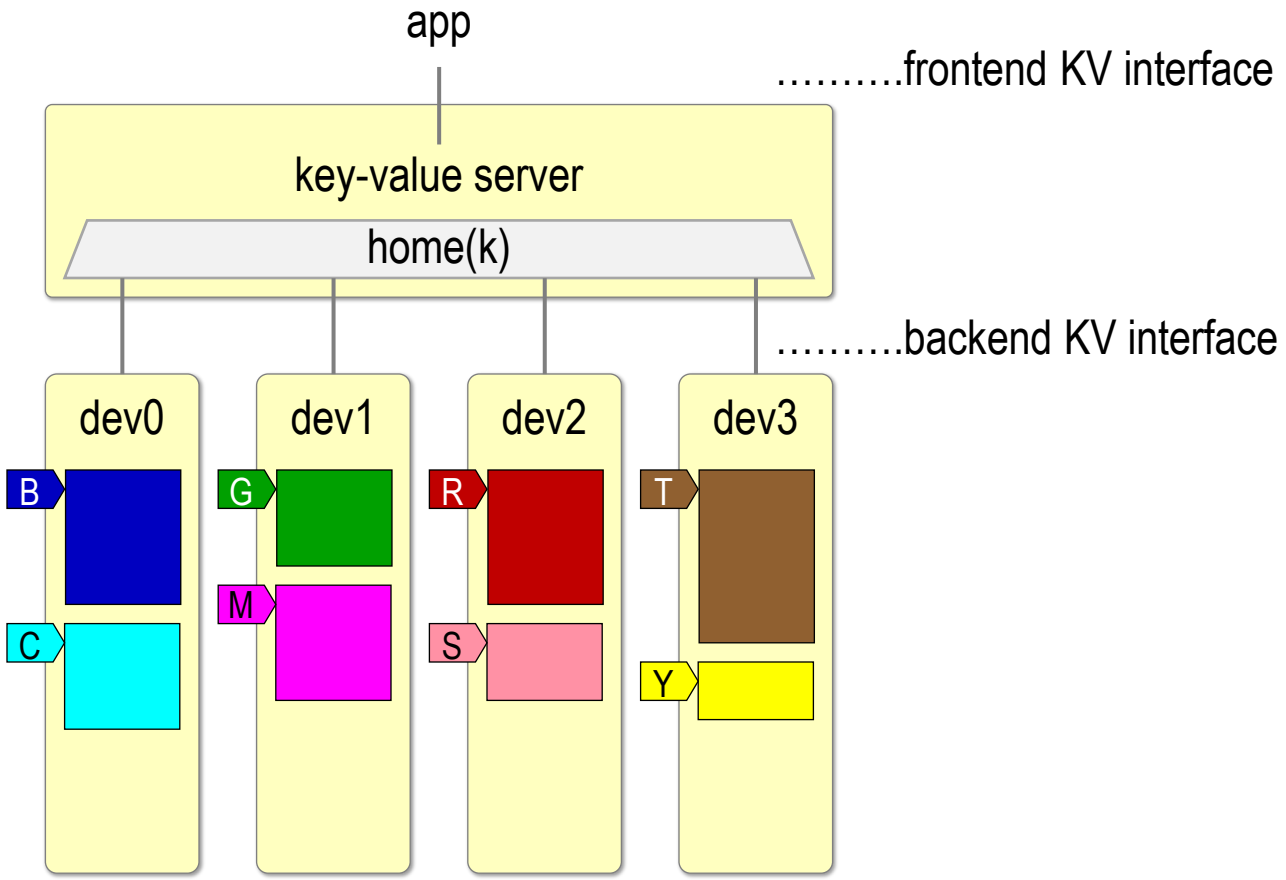
Umesh Maheshwari
Chiku Research

USENIX HotStorage 2020

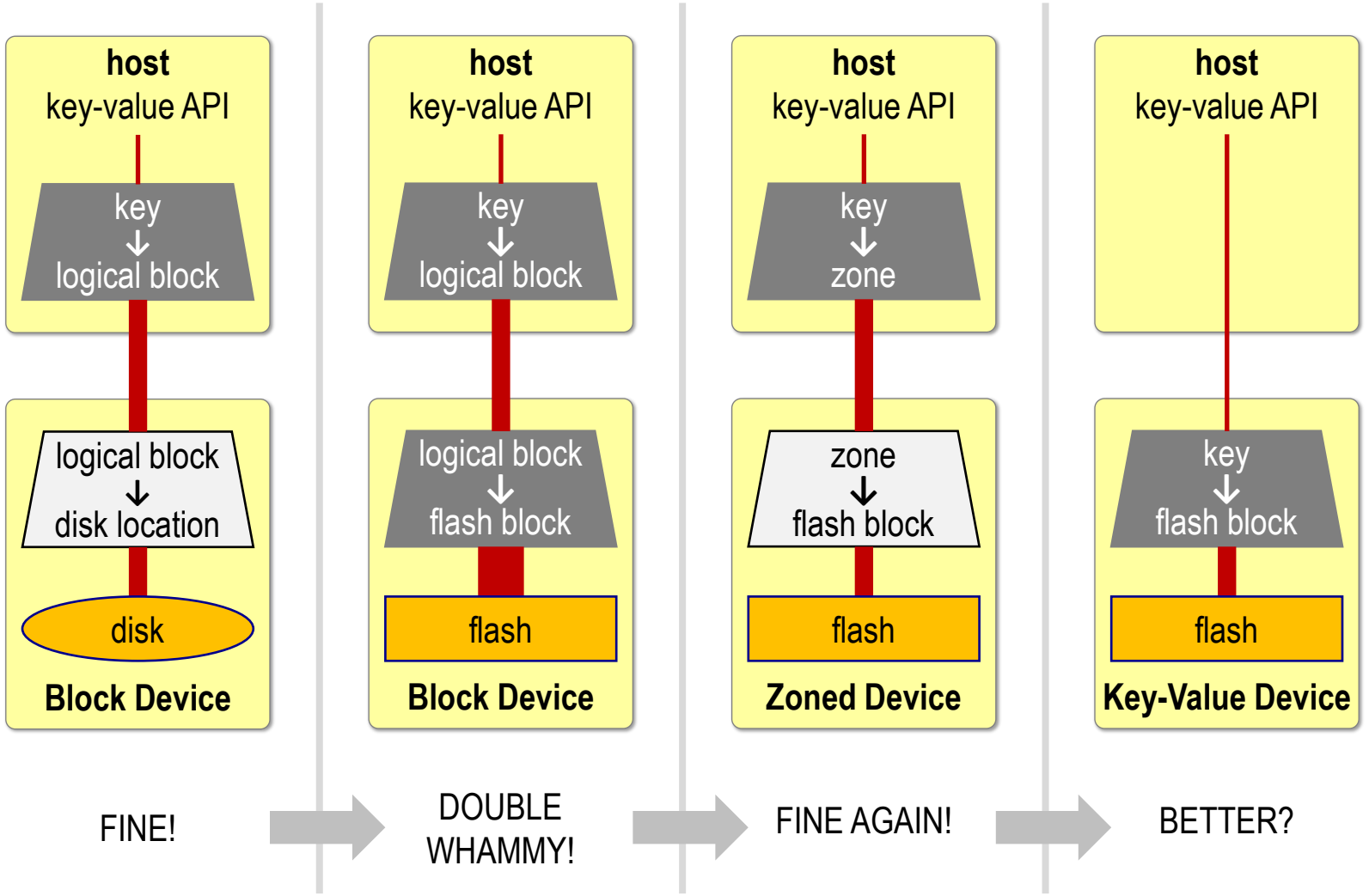
Context

- Host: key-value server
- Device media: flash

- Device namespace: key-value
- Lightweight server: no translation

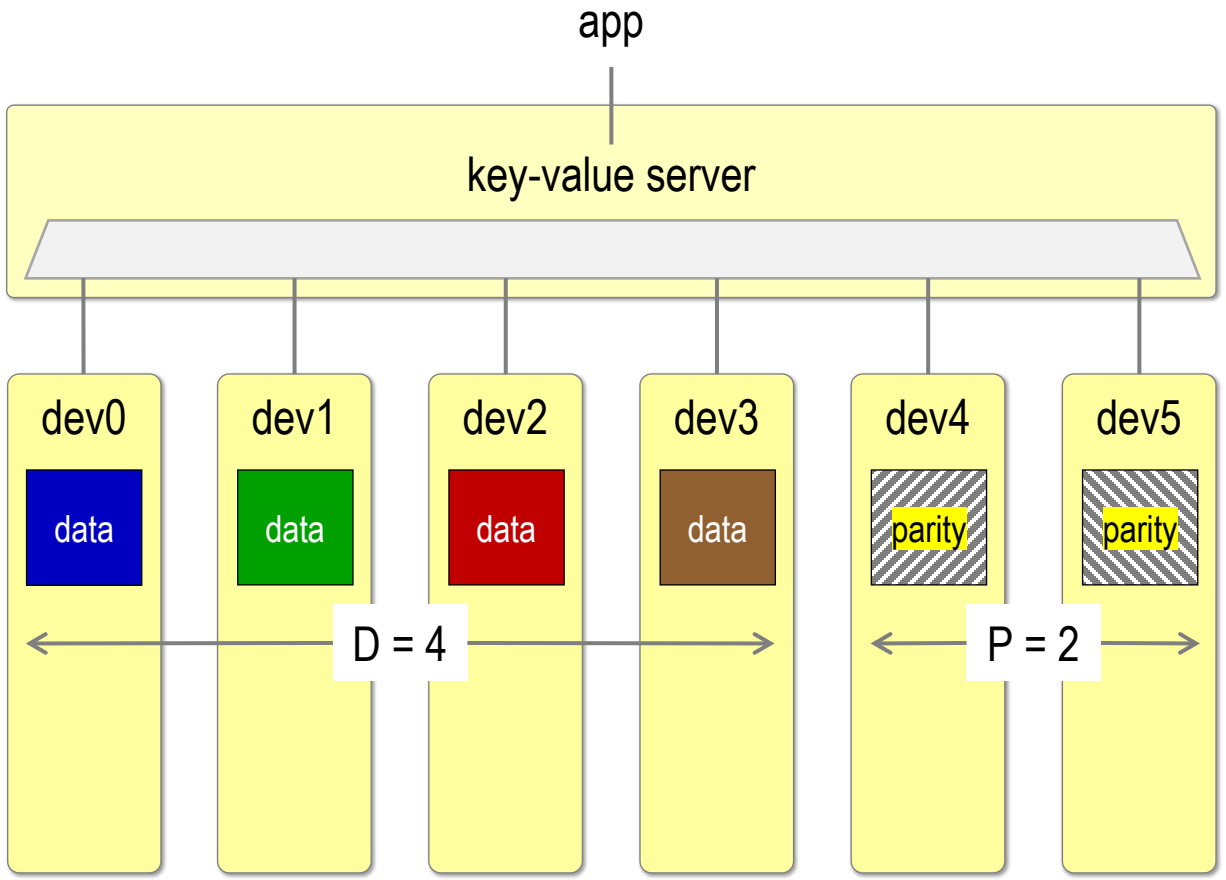


Why Key-Value Devices



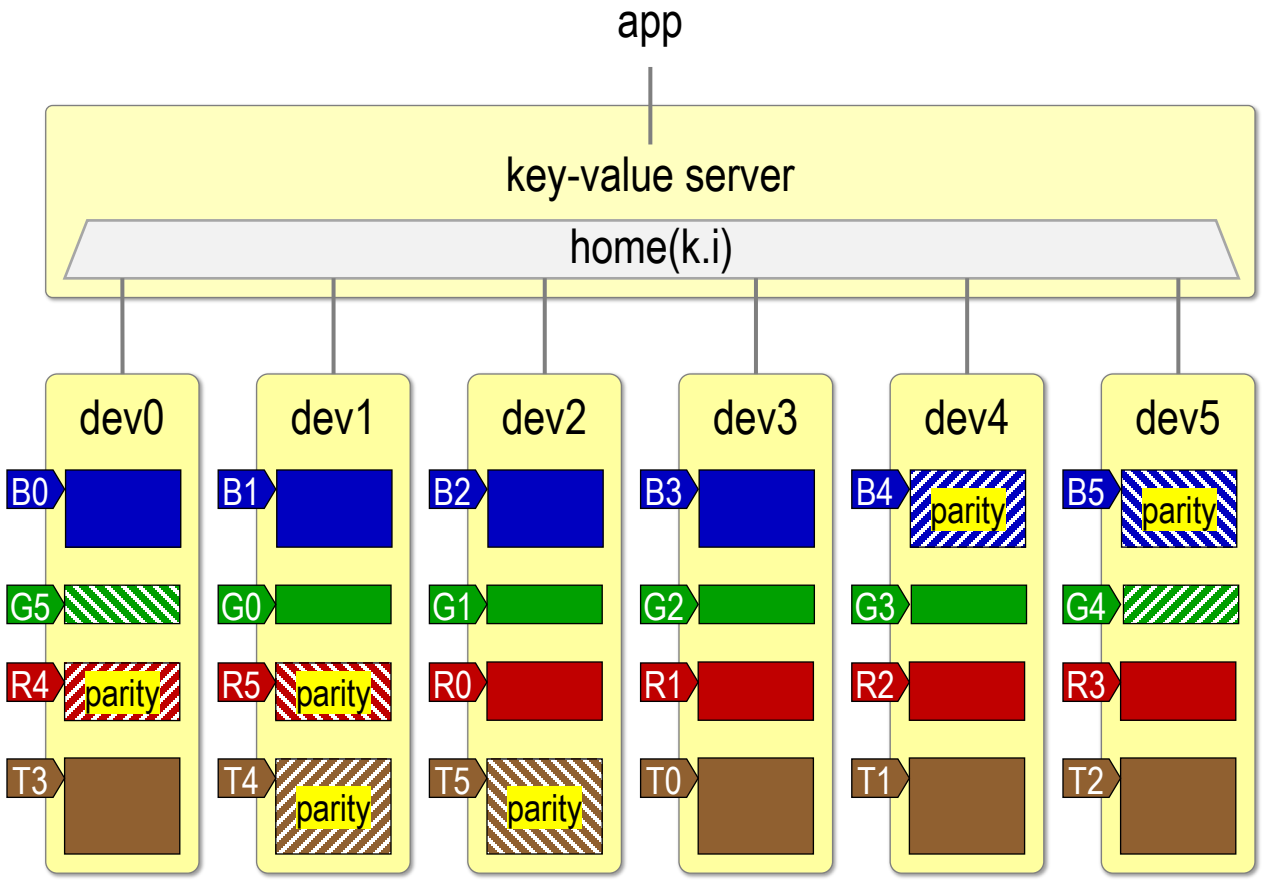
Erasure Coding Goals

- Tolerate loss of P devices (full/partial)
- Restore corrupted object without full scan
- Limit storage amplification
  - replication
    - replicas: 1 + P
    - storage amp = 1 + P
  - parity coding
    - stripe units: D data + P parity
    - storage amp = 1 + P/D



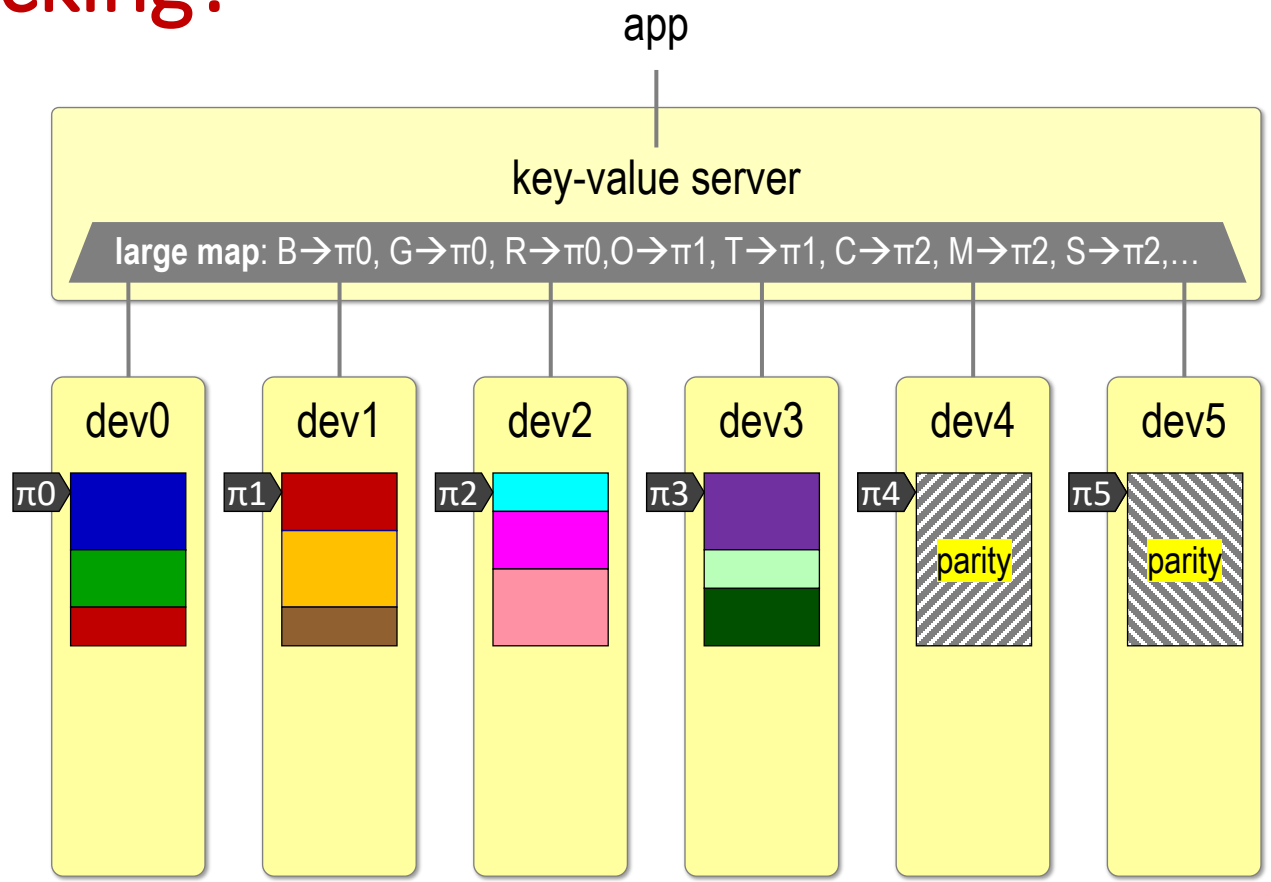
For Large Objects: Splitting

- For frontend object ≥ ~16 KB
  - 1 frontend object → D backend objects
- Lightweight translation
  - backend_key = frontend_key . seq_num
  - home(k.i) = home(k)+i mod N
- Inefficient for small objects
  - degrades read throughput
  - creates too many objects



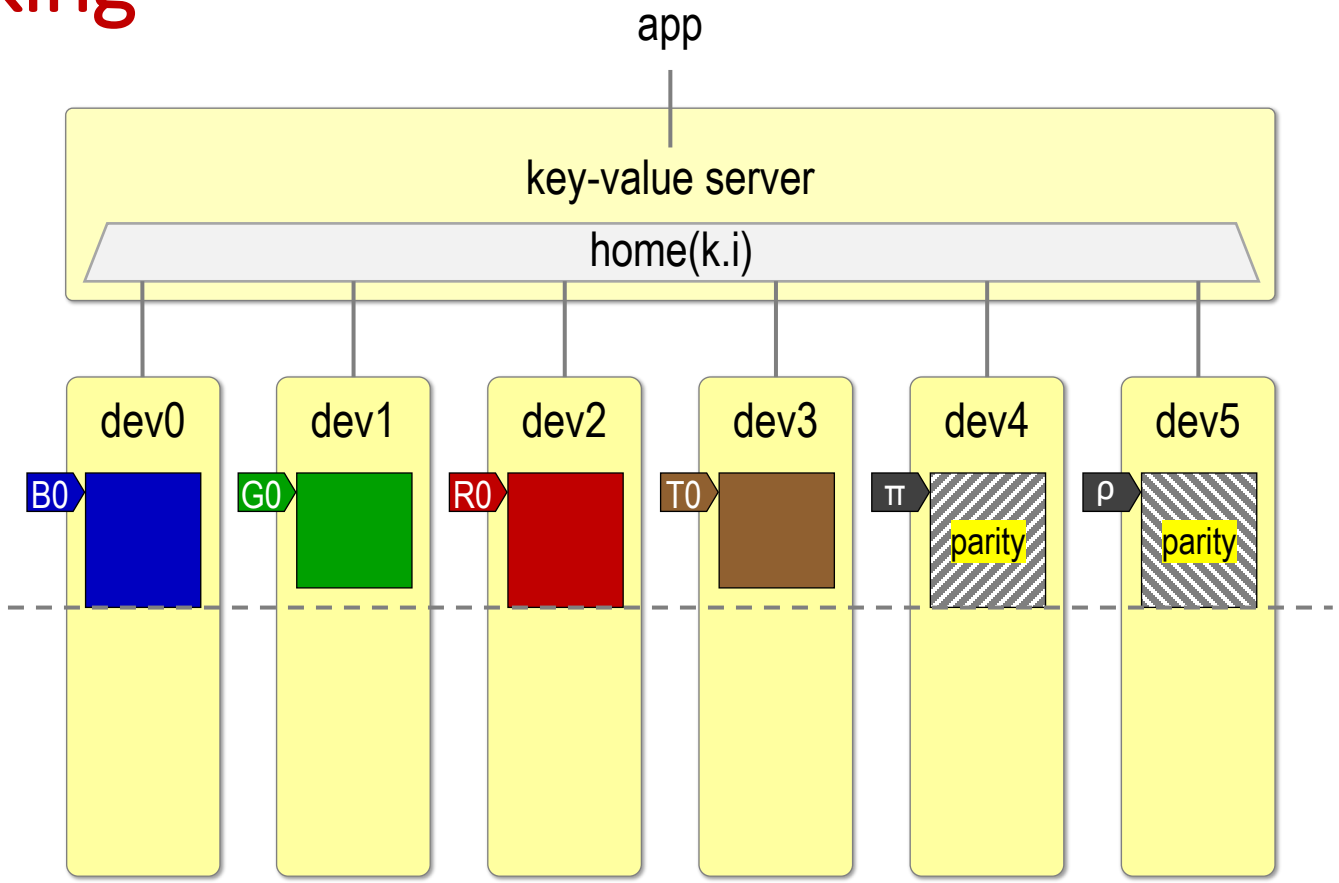
For Small Objects: Multi-Packing?

- A tempting approach
  - many frontend objects → 1 large object
  - 1 large object → D backend objects
- Heavyweight translation
  - large map: frontend key → backend key
- Defeats the purpose of key-value devices!



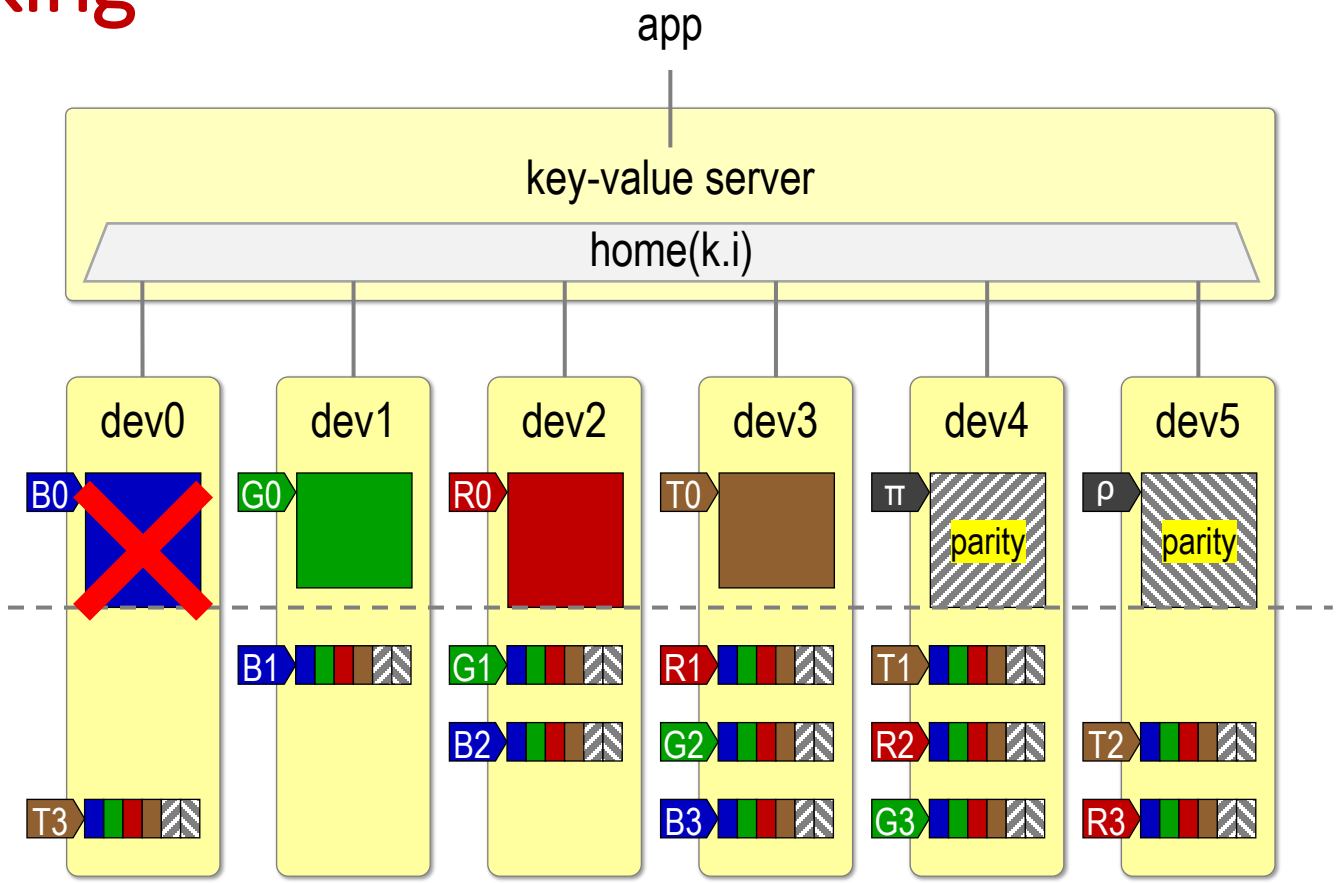
For Small Objects: Uni-Packing

- KVMD [1/FAST'20]: uni-packing
  - 1 frontend object → 1 backend object
- Challenge 1: variable size objects
  - wait for D objects of similar size
- Challenge 2: find an object's stripe mates!
  - store per-stripe metadata



For Small Objects: Uni-Packing

- KVMD [1/FAST'20]: uni-packing
  - 1 frontend object → 1 backend object
- Challenge 1: variable size objects
  - wait for D objects of similar size
- Challenge 2: find an object's stripe mates!
  - store per-stripe metadata
- Stripe metadata object
  - key = frontend_key . meta_num
  - value = sequence of keys in stripe
- Copies of metadata object
  - D siblings: one for each frontend object
  - P+1 clones: to tolerate P failures



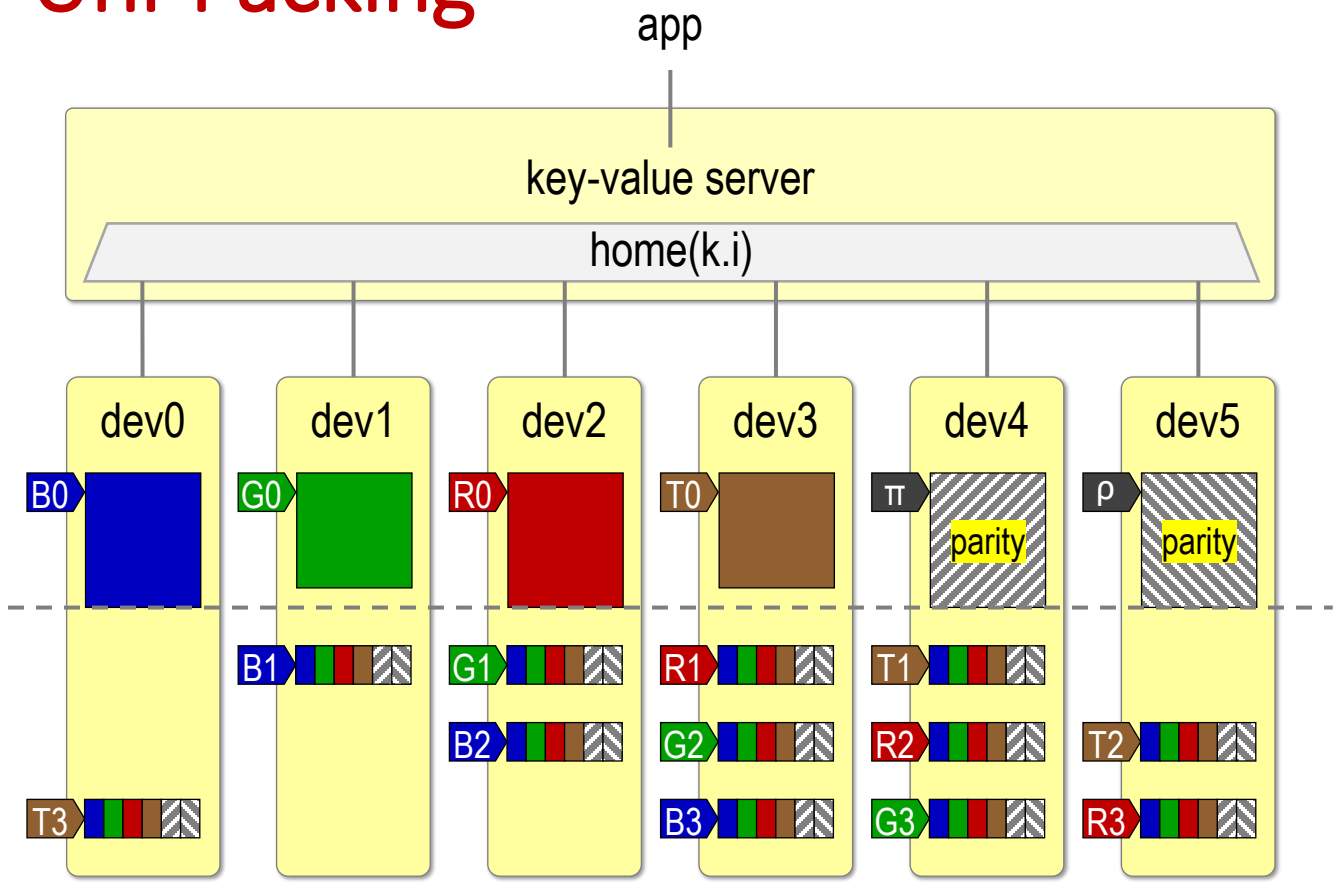
The Metadata Overhead of Uni-Packing

- KVMD: object ≥ 1 KB, “metadata is … small”
- Facebook [2/FAST’20]: object < 100 B
- Object count
  - data: D = 4; parity: P = 2
  - metadata: D(P+1) = 4x3 = 12

  Amp = (4+2+12)/4 = 18/4 = 4.5
- Byte size
  - data: 256; parity: 256
  - metadata: K(1+D+P) = 16x7 = 112

  Amp = (256x6+112x12)/(256x4) = 2.8

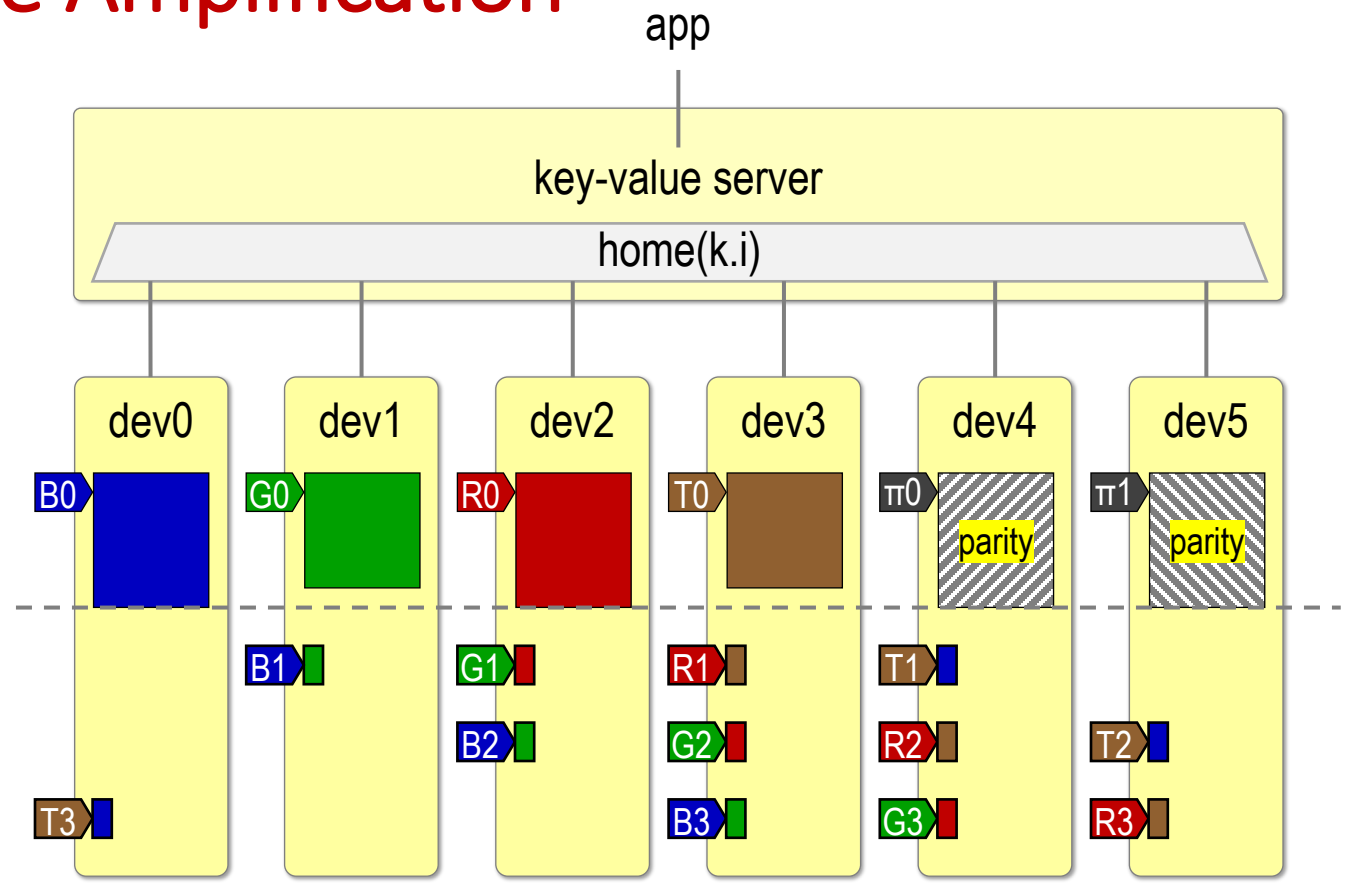
	parity	repl	unipack
obj amp	1.5	3.0	4.5
byte amp	1.5	3.0	2.8



StripeFinder: Reducing Byte Amplification

- Finder object
  - key = frontend_key . meta_num
  - value = next data key in stripe
- Finder ring
  - finds all data keys in stripe
  - tolerates P failures
- Parity key = strong hash of data keys
- Byte size
  - finder: K(1+1) = 16x2 = 32

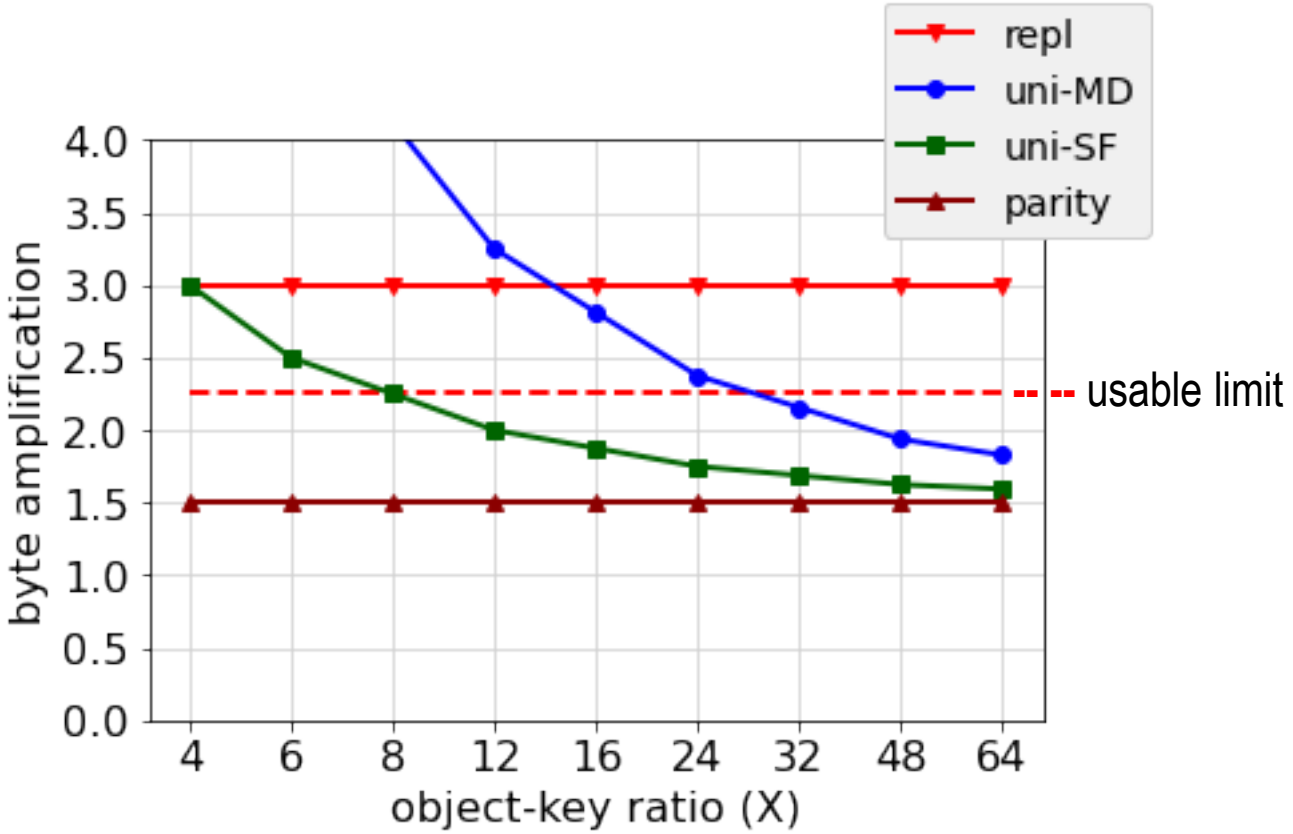
  Amp = (256x6+32x12)/(256x4) = 1.9



	parity	repl	uni-MD	uni-SF
obj amp	1.5	3.0	4.5	4.5
byte amp	1.5	3.0	2.8	1.9

Byte Amplification

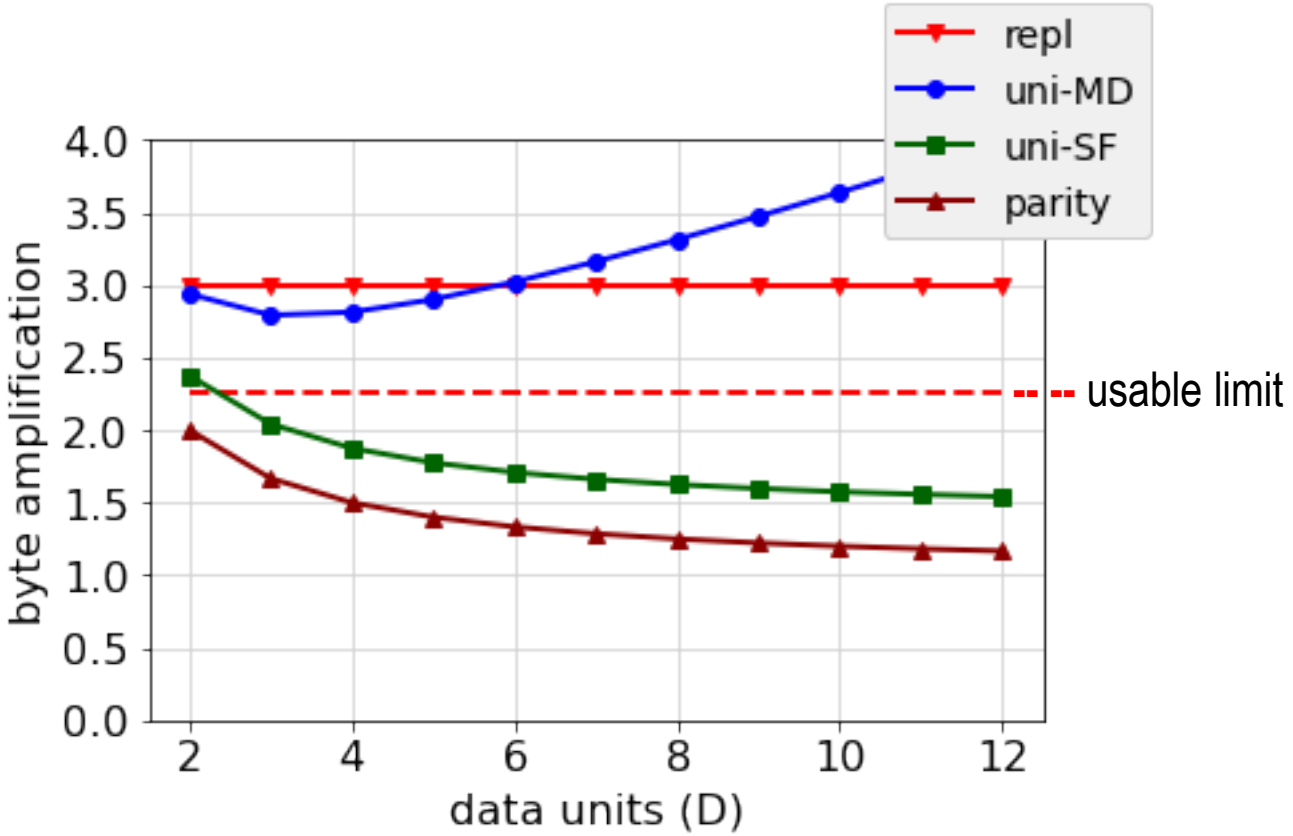
- Parameters
  - P: losses to tolerate
  - D: data units in a stripe
  - X: object-key ratio
- Byte amplification
  - replication: 1+P
  - parity: 1+P/D
  - unipack-MD: (1+P/D) + (1+P)(1+D+P)/X
  - unipack-SF: (1+P/D) + (1+P)(1+1)/X



fixed: P = 2, D = 4

Byte Amplification

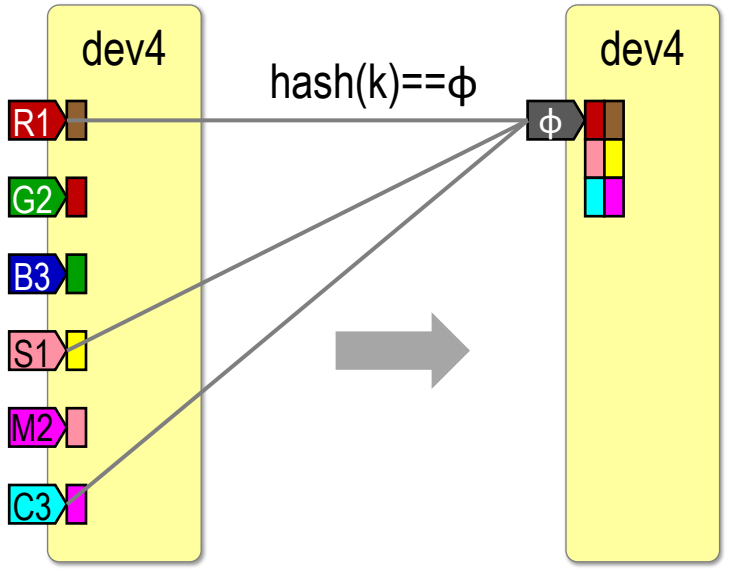
- Parameters
  - P: losses to tolerate
  - D: data units in a stripe
  - X: object-key ratio
- Byte amplification
  - replication: 1+P
  - parity: 1+P/D
  - unipack-MD: (1+P/D) + (1+P)(1+D+P)/X
  - unipack-SF: (1+P/D) + (1+P)(1+1)/X



fixed: X = 16

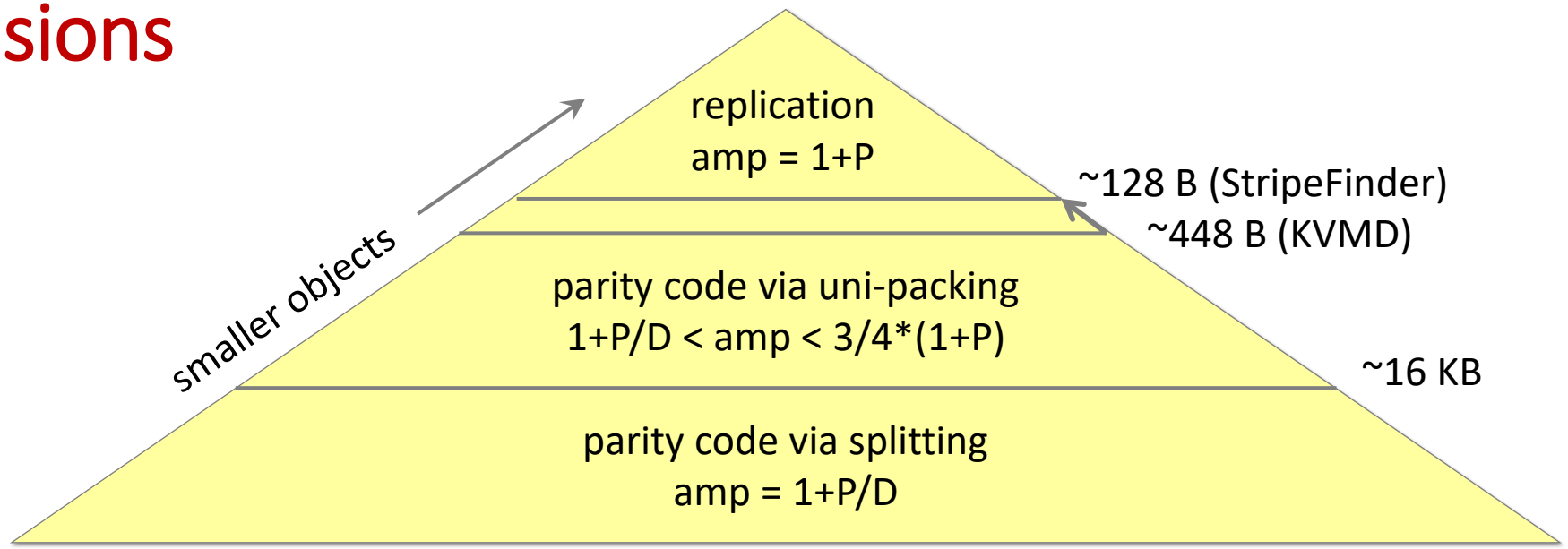
StripeFinder: Reducing Object Amplification

- Combine C finder objects
  - within same device, across stripes
  - whose keys hash to same value
- Combined finder object
  - key = hash(finder key)
  - can be accessed using any of C frontend keys
- Object amp
  - pre: (1+P/D) + (1+P)
  - post: (1+P/D) + (1+P)/C



	parity	repl	uni-MD	uni-SF	
				C=1	C=10
obj amp	1.5	3.0	4.5	4.5	1.8
byte amp	1.5	3.0	2.8	1.9	1.9

Conclusions



1. Multi-packing is a tempting approach, but not suitable for key-value devices.
2. Uni-packing is a plausible option, but imposes high metadata overhead.
   - StripeFinder reduces metadata overhead, expanding usability to smaller objects.
   - Even with StripeFinder, uni-packing is not practical for tiny objects.
   - A real implementation might expose further inefficiencies and complexities.
3. Parity coding of small objects over key-value devices is an uphill battle!
   - This is a significant weakness of key-value devices in comparison to zoned devices.

Future Directions

1. Extend key-value device interface to store stripe metadata efficiently.
2. Design a fresh technique for erasure coding of small key-value objects.
   - Or, prove a lower bound on the amount of stripe metadata that must be maintained!
3. Question the need for parity coding on key-value devices:
   - The amount of data stored on key-value devices might be a small fraction of overall data.
   - Simplicity might be more important than storage efficiency.

Information

- Author contact: umesh@alum.mit.edu

- Acknowledgments: This work was done while author was affiliated with Hewlett Packard Enterprise.

- References

1. R. Pitchumani and Y. Kee. Hybrid Data Reliability for Emerging Key-Value Storage Devices. In 18th USENIX Conference on File and Storage Technologies (FAST'20), 2020.
2. Z. Cao, S. Dong, S. Vemuri, and D. Du. Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook. In 18th USENIX Conference on File and Storage Technologies (FAST'20), 2020.