Compression

Making Flash (Even) Cheaper

Thomas Parnell, IBM Research - Zurich

Agenda

1. Why do we want compression?
2. What algorithms exist?
3. How should we choose an algorithm?
4. How does compression affect controller design?

Why Do We Want Compression?

- NAND Flash technology has seen unprecedented technological innovation over the last few years
  - 1y/z nm, TLC, 3D, QLC
- Despite this, Flash remains a relatively expensive storage medium (compared to consumer HDD)
- Data reduction techniques such as compression and deduplication offer a way to “close the gap”
- Compression can bring additional performance benefits

Compression Algorithms

- Lempel-Ziv
- Huffman Coding
- Arithmetic Coding
- Adaptive Statistical Compression
- Context Mixing

Lempel-Ziv Coding

There is a lot of talk about Flash Memory at the Flash Memory Summit

Replace long matches of characters with (Length, Distance) back-pointers

There is a lot of talk about Flash Memory at the (12,20) Summit

Length=12 chars

Distance=20 chars

[Ziv & Lempel, 1977]

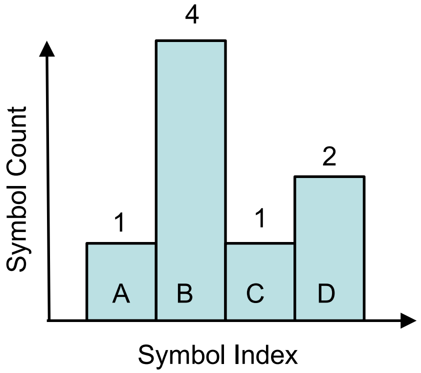
Lempel-Ziv coding underpins many compression algorithms widely in use today:

LZ4, LZO, GZIP

GZIP uses Lempel-Ziv together with a secondary compression technique…..

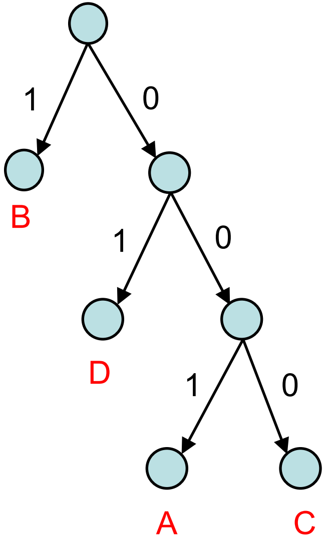
Huffman Coding

Data Model



Histogram that can be extracted from input data

Huffman Tree



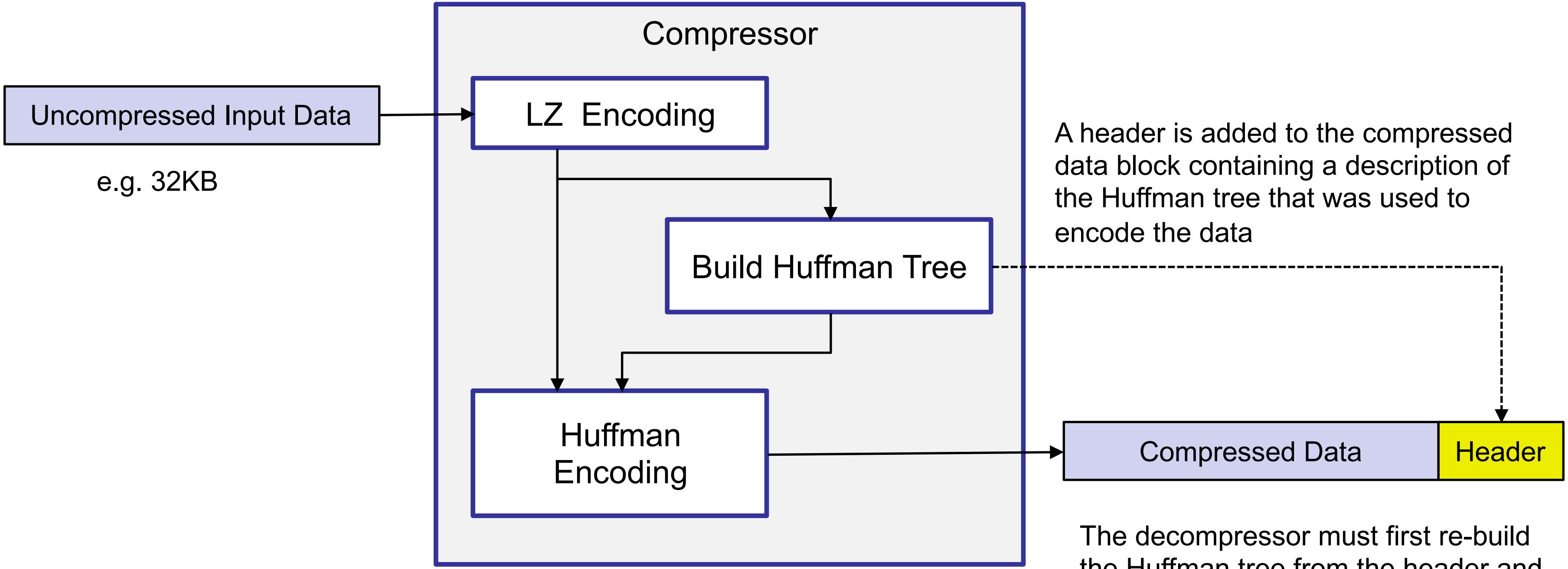
[Huffman, 1952]

Variable Length Encoding

Input Symbol	Output Code
A	001
B	1
C	000
D	01

Frequent symbols are assigned short codes, infrequent symbols are assigned long codes

Example: GZIP

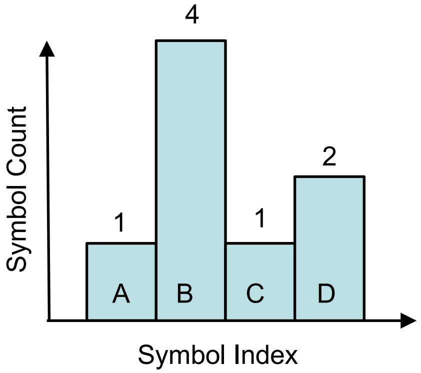


A header is added to the compressed data block containing a description of the Huffman tree that was used to encode the data

The decompressor must first re-build the Huffman tree from the header and then un-do the two encoding steps

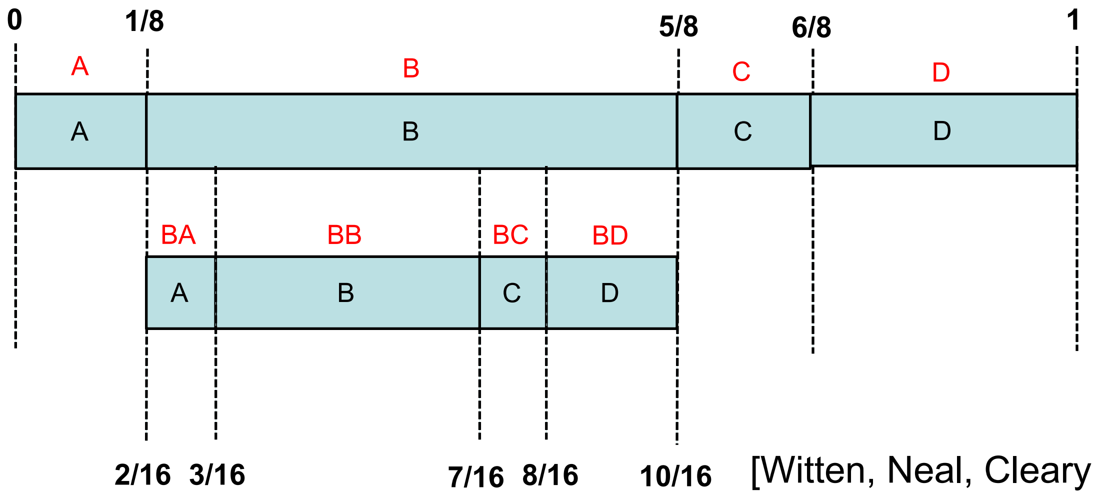
Arithmetic Coding

Data Model



Histogram that can be extracted from input data

Whole sequence of input symbols is encoded as an interval

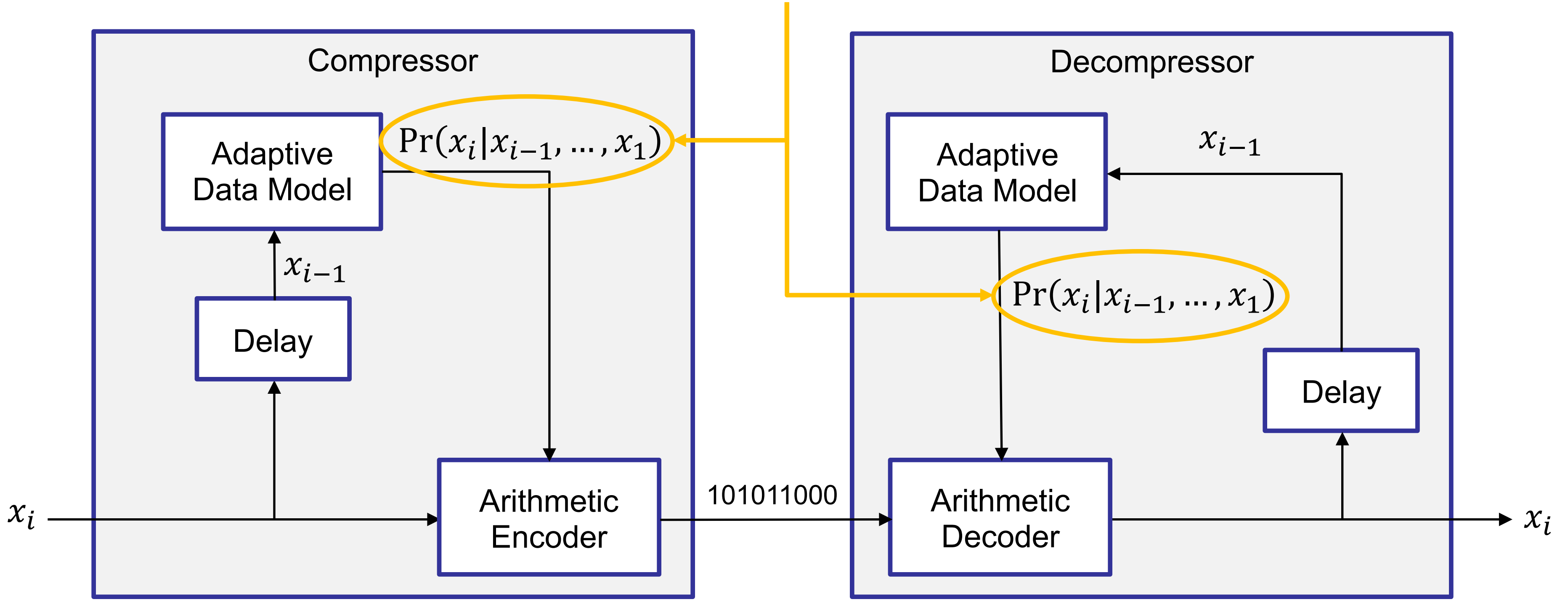


[Witten, Neal, Cleary 1987]

One key advantage of Arithmetic coding over Huffman coding:
the data model can change during encoding

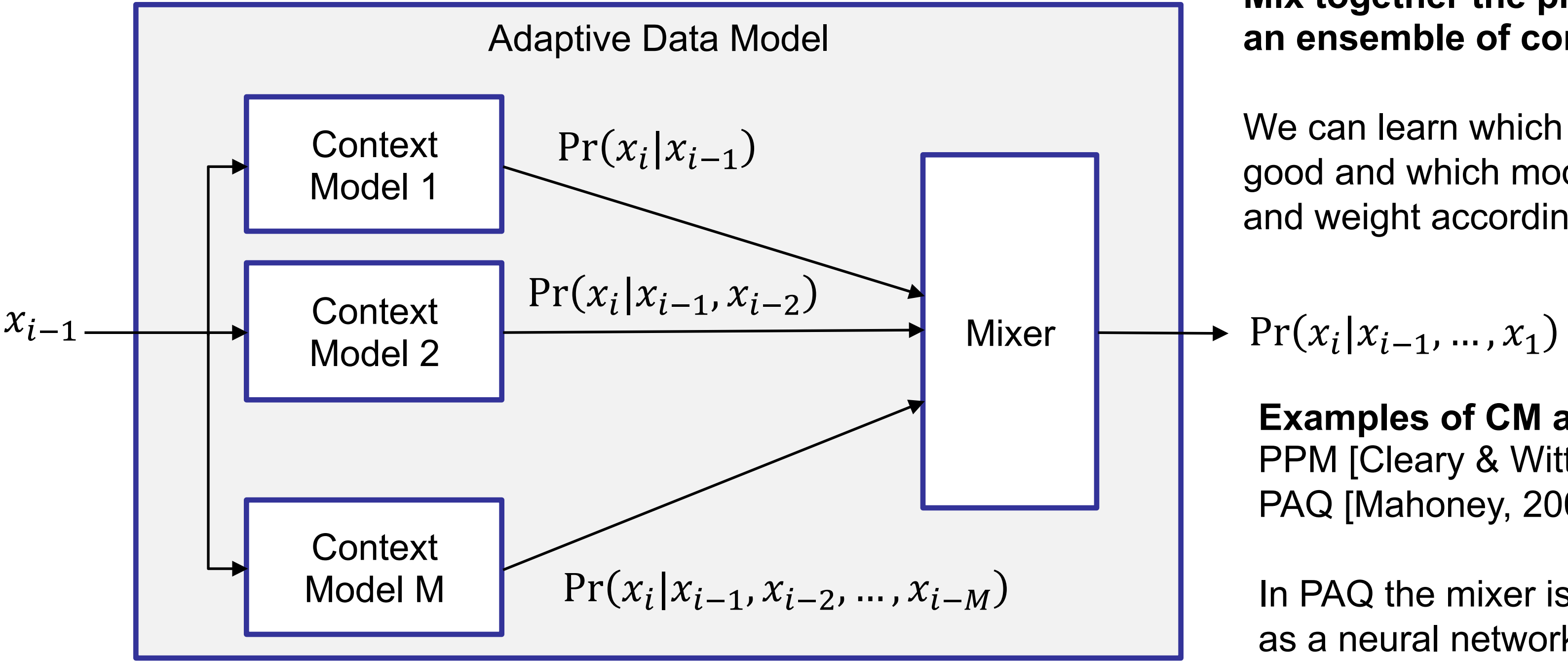
Adaptive Statistical Compression

Exact same model is generated in comp and decomp. There is no need to store model in header.



Unlike GZIP, we only need 1 pass through the data.

Context Mixing



Mix together the predictions of an ensemble of context models

We can learn which models are good and which models are bad and weight accordingly

Examples of CM algorithms:
PPM [Cleary & Witten, 1988]
PAQ [Mahoney, 2005]

In PAQ the mixer is implemented as a neural network. This algorithm won the **Hutter Prize.**

Which is the best algorithm?

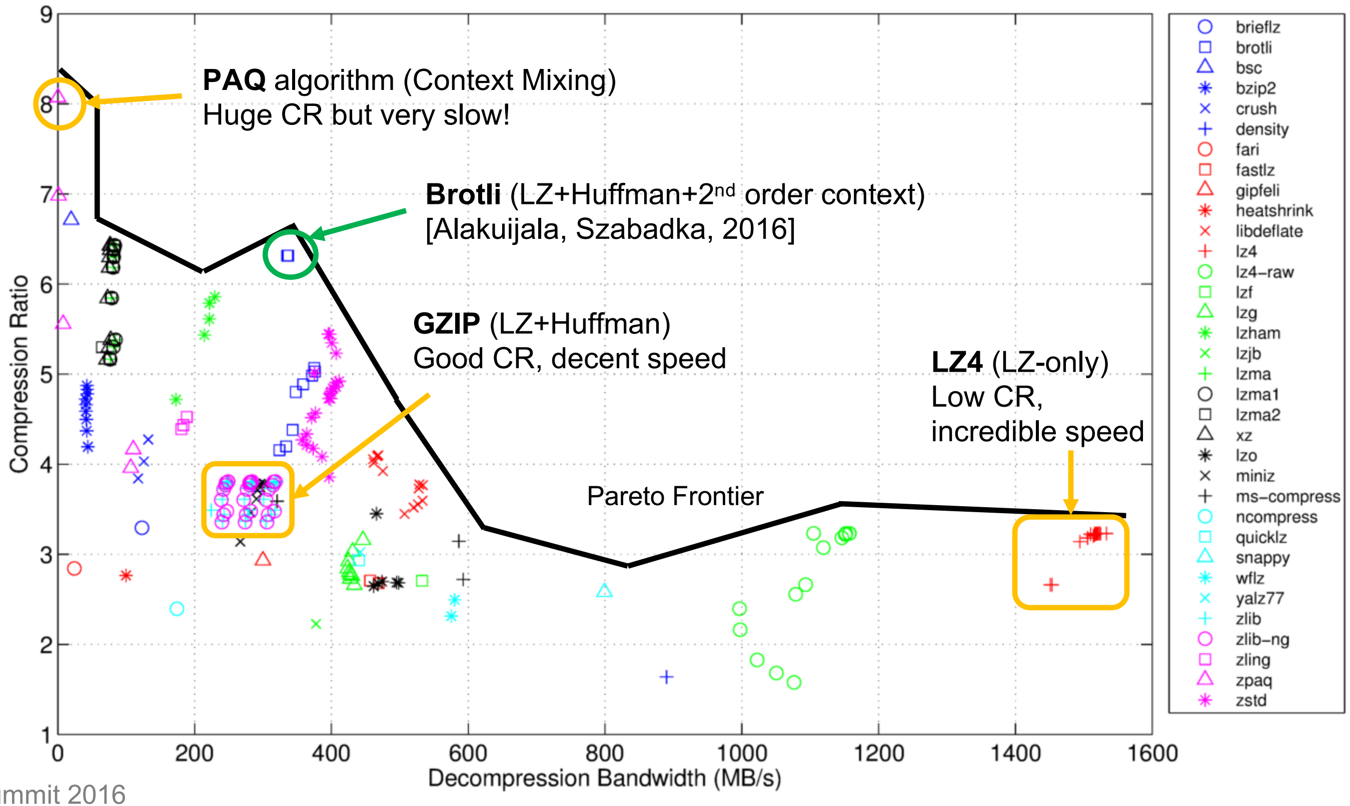
- The best choice of algorithm varies depending on application
- Let us define compression ratio as follows:

  CR = {Uncompressed size of data} / {Compressed size of data}
- CR=2 would results in a 2x increase in the capacity of the system that can be exposed to the user
- Key trade-offs involved in selecting an algorithm:

Compression Ratio Vs. Compression Bandwidth
Decompression Bandwidth
Memory Usage

Most critical issues for Flash-based storage??

Exploring the Algorithm Space



Squash compression benchmark

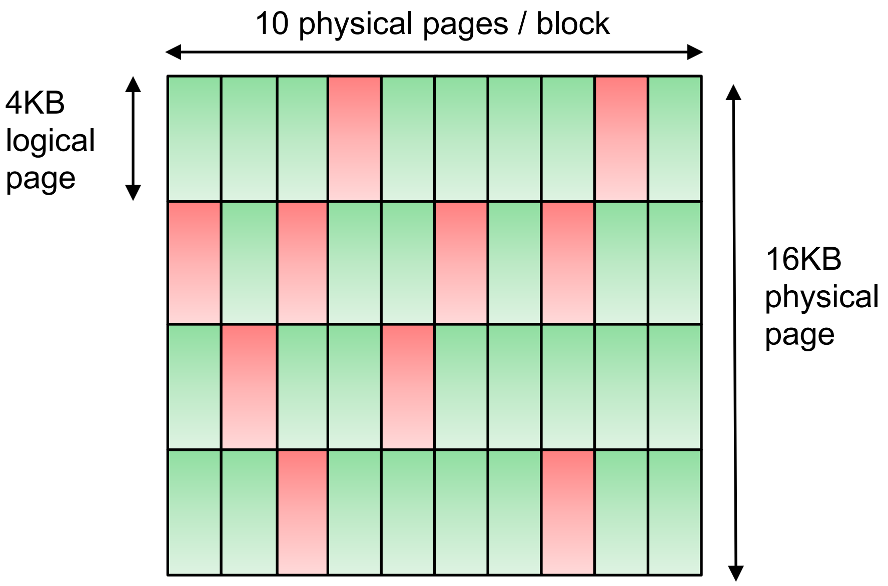
IBM data corpus

Flash-specific Considerations

- Integration of compression can have a profound impact on the design of a Flash controller
- Compression can also change performance
- We will consider two interesting examples:
  1. Write amplification
  2. Data placement

Write Amplification (CR=1)

WA = {Total Physical Data Written} / {Total Logical Data Written}

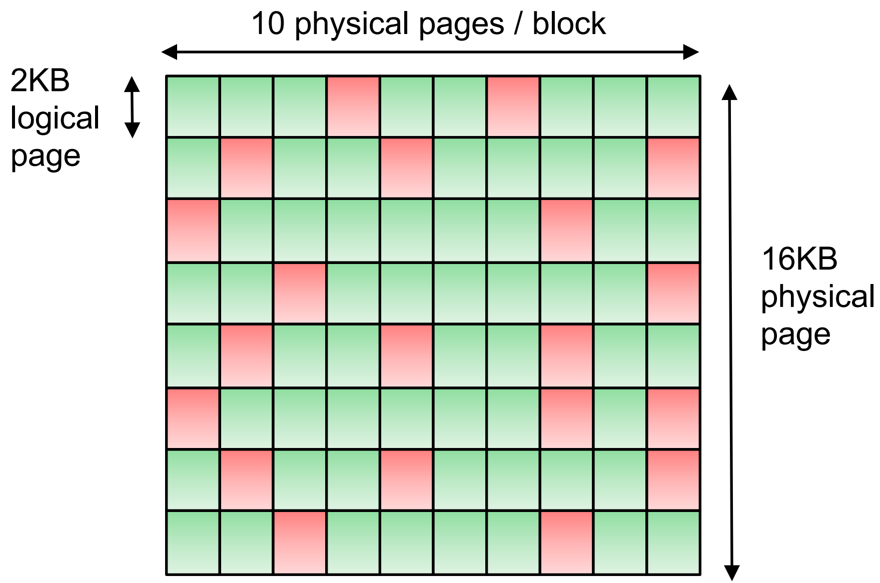


# of logical pages in block: 40
# of valid logical pages: 30
# of invalid logical pages: 10 (1/4 of block)
Logical host writes: 40 * 4KB = 160KB
Physical host writes: 40 * 4KB = 160KB
Physical relocation writes: 30 * 4KB = 120KB

WA = (160KB+120KB)/160KB
= **1.75**

Write Amplification (CR=2)

WA = {Total Physical Data Written} / {Total Logical Data Written}



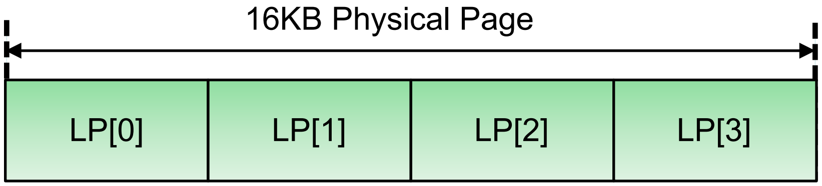
# of logical pages in block:	80
# of valid logical pages:	60
# of invalid logical pages:	20 (1/4 of block)
Logical host writes:	80 * 4KB = 320KB
Physical host writes:	80 * 2KB = 160KB
Physical relocation writes:	60 * 2KB = 120KB

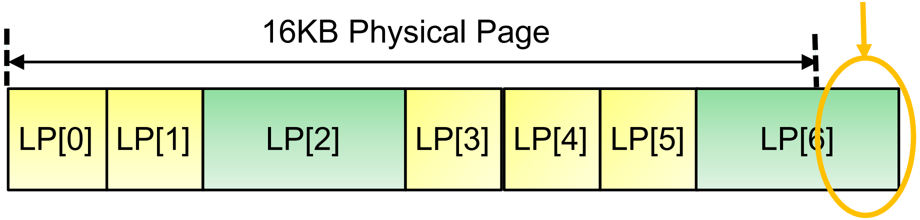
WA = (160KB+120KB)/320KB
= **0.875**

Compression leads to a reduction in WA

Data Placement

Logical pages can straddle physical pages!





Logical Page Index	Physical Page Index	Offset
0	p	0KB
1	p	4KB
2	p	8KB
3	p	12KB

Logical Page Index	Physical Page Index	Offset	Length
0	p	0KB	2KB
1	p	2KB	2KB
2	p	4KB	4KB
3	p	8KB	2KB
4	p	10KB	2KB
5	p	12KB	2KB
6	p,(p+1)	14KB	4KB

Conclusions

- Compression has significant benefits for Flash-based systems:
  - Increased logical capacity
  - Decreased write amplification
- However, it makes controller design more challenging:
  - Increased meta-data requirements
  - Increased complexity of Logical-to-Physical mapping
- Choose the right compression algorithm for your system:
  - What compression/decompression bandwidth is required?
  - How much memory is available?