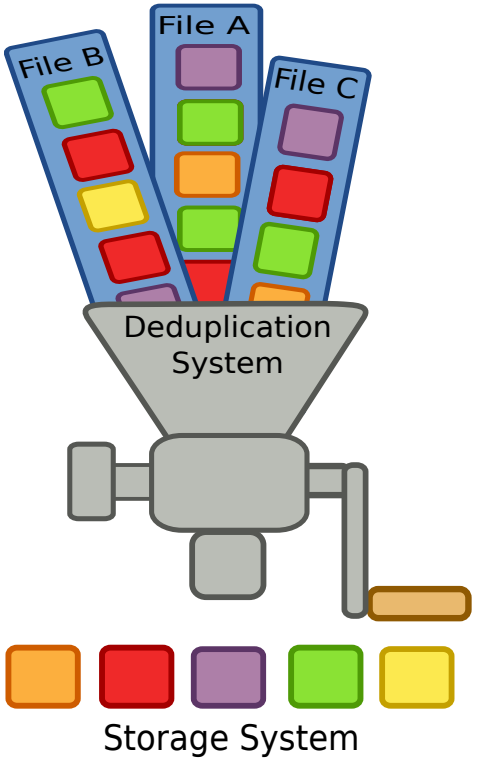
File Recipe Compression in Data Deduplication Systems

Dirk Meister
André Brinkmann
Tim Süß

JOHANNES GUTENBERG UNIVERSITÄT MAINZ
JG|U

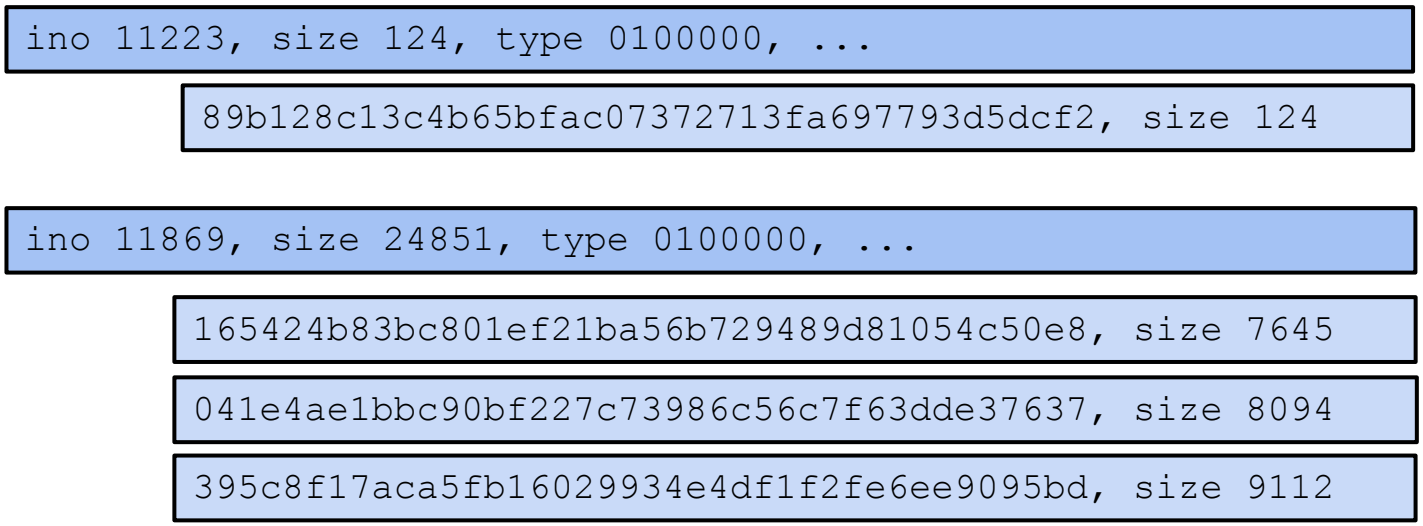
Outline

- File Recipes
- Zero-Chunk Suppression
- Chunk Index Page-Based
- Statistical Approaches
- Evaluation



File Recipes

- Most file-based deduplication systems use *file recipes*
  - List of chunk fingerprints of a file
  - Similar to data block pointer in “normal” file systems
  - Differences
    - Variable-sized chunks
    - Fingerprints at least 20 bytes



Motivation

- File recipes can occupy significant disk space
  - Chunk data grows with *post-deduplication* capacity
  - File recipes grows with *pre-deduplication* capacity

- Examples
  - Year of weekly full backups, 95% weekly redundancy

    6.6% of (estimated) total disk space
  - Year of daily full backups, 99% daily redundancy

    28.4% of (estimated) total disk space

The compression of file recipes promises significant savings of the overall storage usage

File Recipe Compression

- Use (shorter) code word instead of fingerprint in file recipe

- Requirement
  - Low overhead: memory, IO, storage
  - Limited impact on write and restore speed

- Assumptions
  - Fingerprinting-based data deduplication systems
  - Full chunk index available, e.g. on SSD storage
  - Backup workloads
  - Reverse lookup necessary

1. Zero-Chunk Suppression

- Chunk reference skew
  - Few chunks responsible for high number of references
- “Zero-chunk”
  - A chunk completely filled with zeros
  - Fingerprint of zero-chunk can be pre-calculated

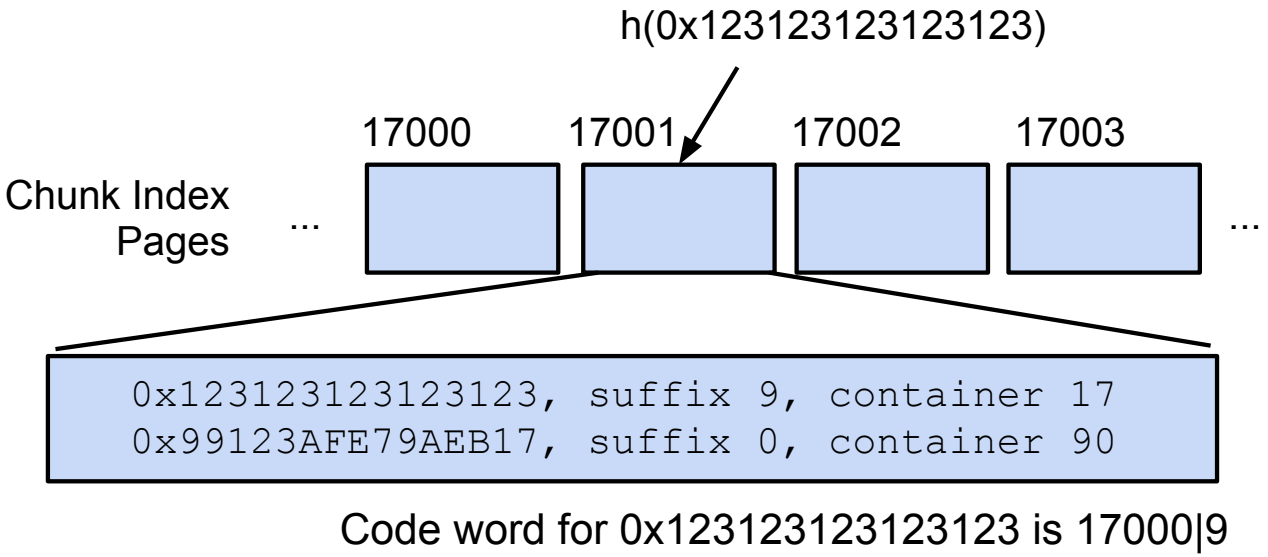
- Special handling in some deduplication systems
  - Never stored, never looked up on disk

- Replace zero-chunk with 1 byte code word
  - Likely already established, e.g., Wei et al., dedupv1

2. Chunk Index Page-Based

- Assign a code word to *each* fingerprint
- Determine code word based on chunk index

- Chunk index page, e.g. of persistent hash table
- Code word = page id | unique identifier in page



3. Statistical Dictionary Compression

- Generalization of zero-chunk suppression
- Determine entropy of each chunk
  - *entropy* = $-\log_2$(*reference count* / *total references*)
  - Easy with reference-counting garbage collection
  - Only possible after some backup runs

- Assign code words to chunks below a entropy threshold
  - E.g., around 0.3% of the chunks
  - Code words equally sized, e.g., 24 bit
  - Maintain (in-memory) reverse index

- Similar to Huffman coding
- But: relaxed, non-optimal, scales to billions of chunks

4. Statistical Prediction Compression

- Per Chunk
  - Determine most-likely following fingerprint
    - *(This is the order-1 statistic)*
  - Select as prediction fingerprint
  - Replace correct prediction by 1 byte code

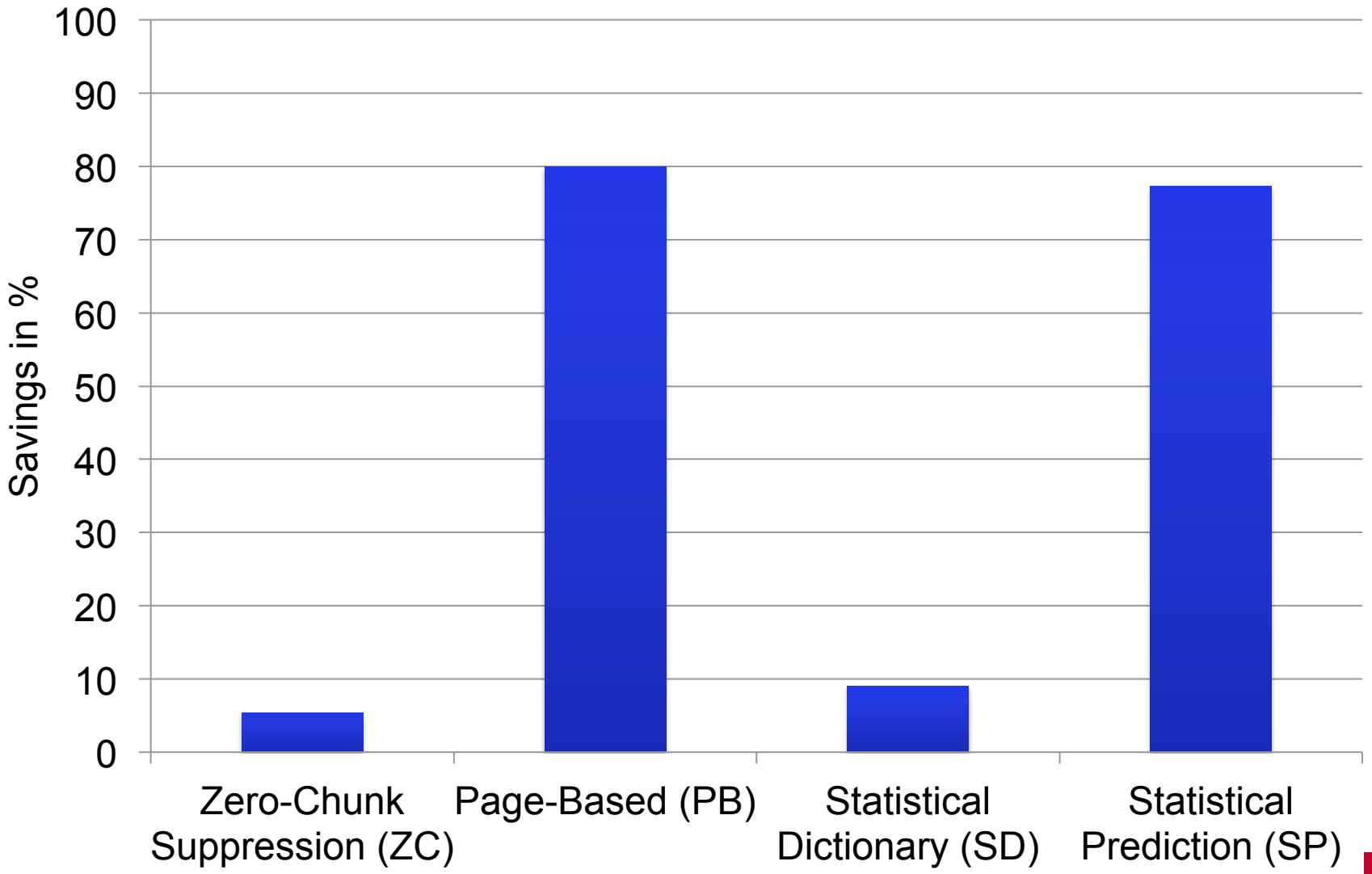
- Using locality property of backup workloads
- Data structure for order-1 too large at scale
- Using estimation of most-frequent following fingerprint
- Prediction not exact, but sufficient

Evaluation

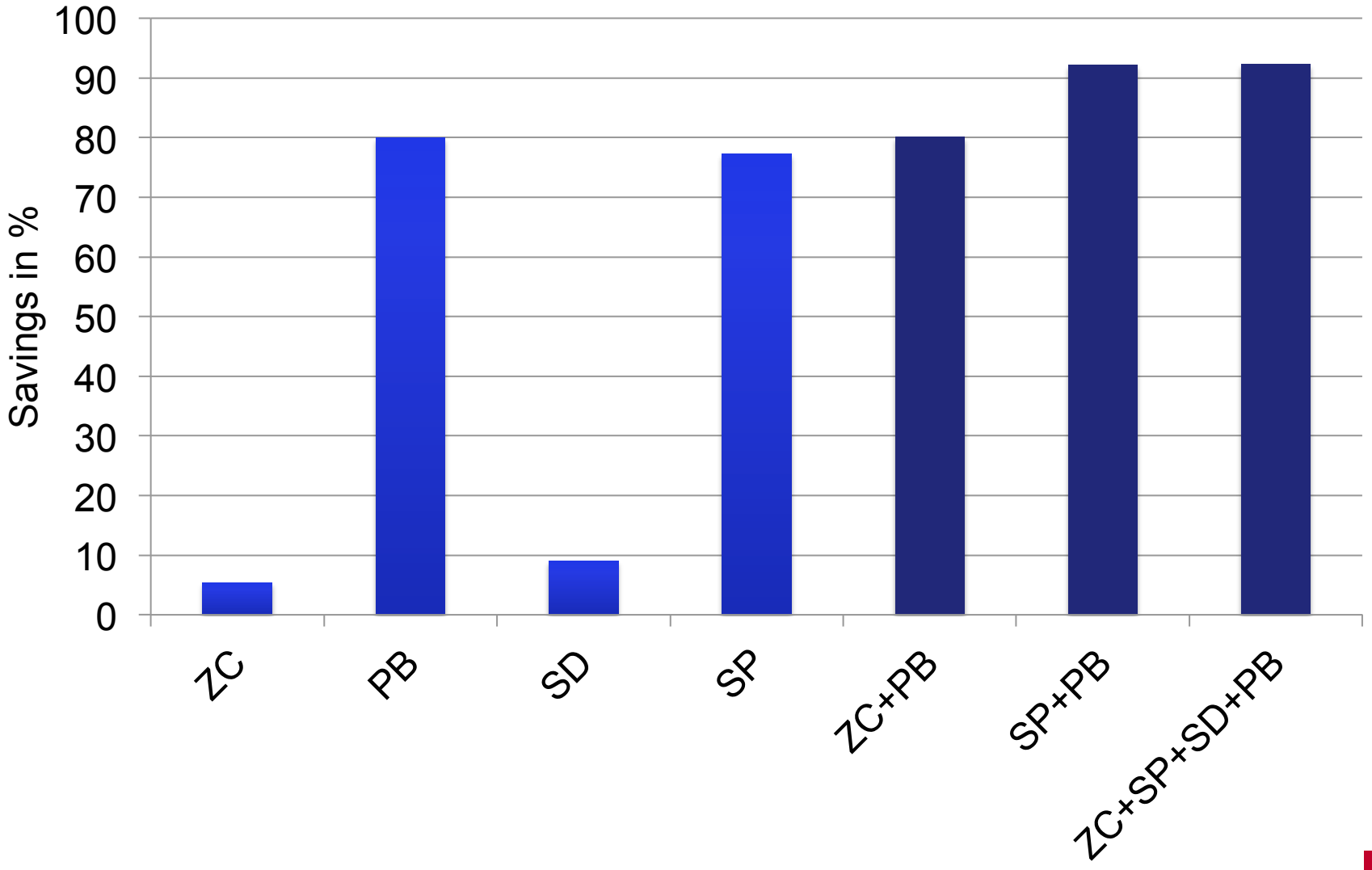
- Trace-based simulation of weekly full backup

- Three data sets in paper
- Here: HOME1 data set
  - Home directory file server of the University of Paderborn
  - Content-defined chunking, expected chunk size 8 KB.
  - 15 weekly full backup traces

Compression Ratio



Compression Ratio (Combined)



Conclusion

- File recipes *can* occupy a significant fraction of disk space
- New compression approaches for file recipes
- Combination shrinks file recipes by more than 90%.
- File recipe compression enables additional data reduction

- File recipes are largely unexplored territory in data deduplication research.

- Simulation tool FS-C and dedupv1 are open source.

Questions?

THANK YOU