Sorted Deduplication: How to Process Thousands of Backup Streams

Jürgen Kaiser
Zentrum für Datenverarbeitung (ZDV)
Johannes Gutenberg-Universität Mainz

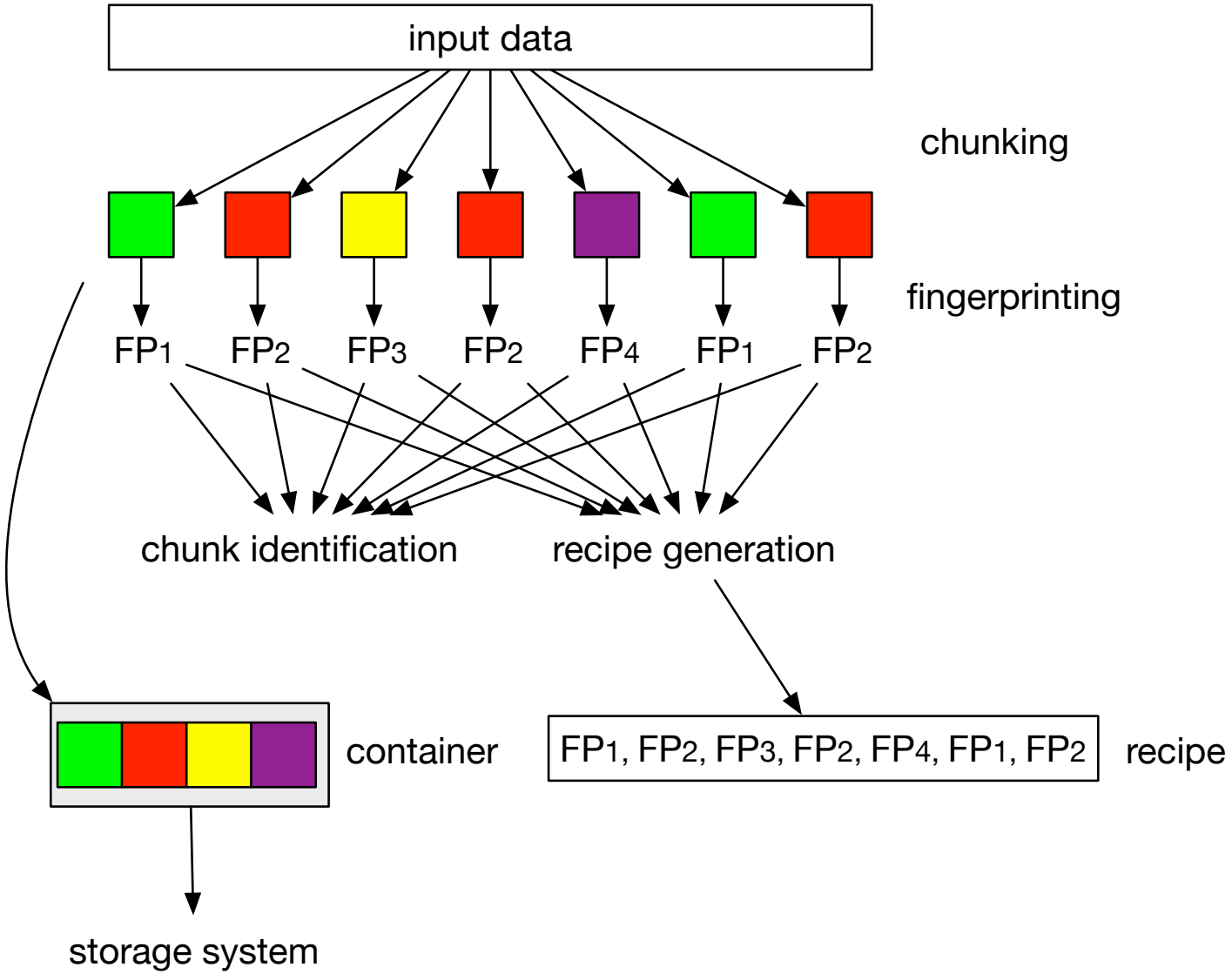
JOHANNES GUTENBERG UNIVERSITÄT MAINZ

JG|U

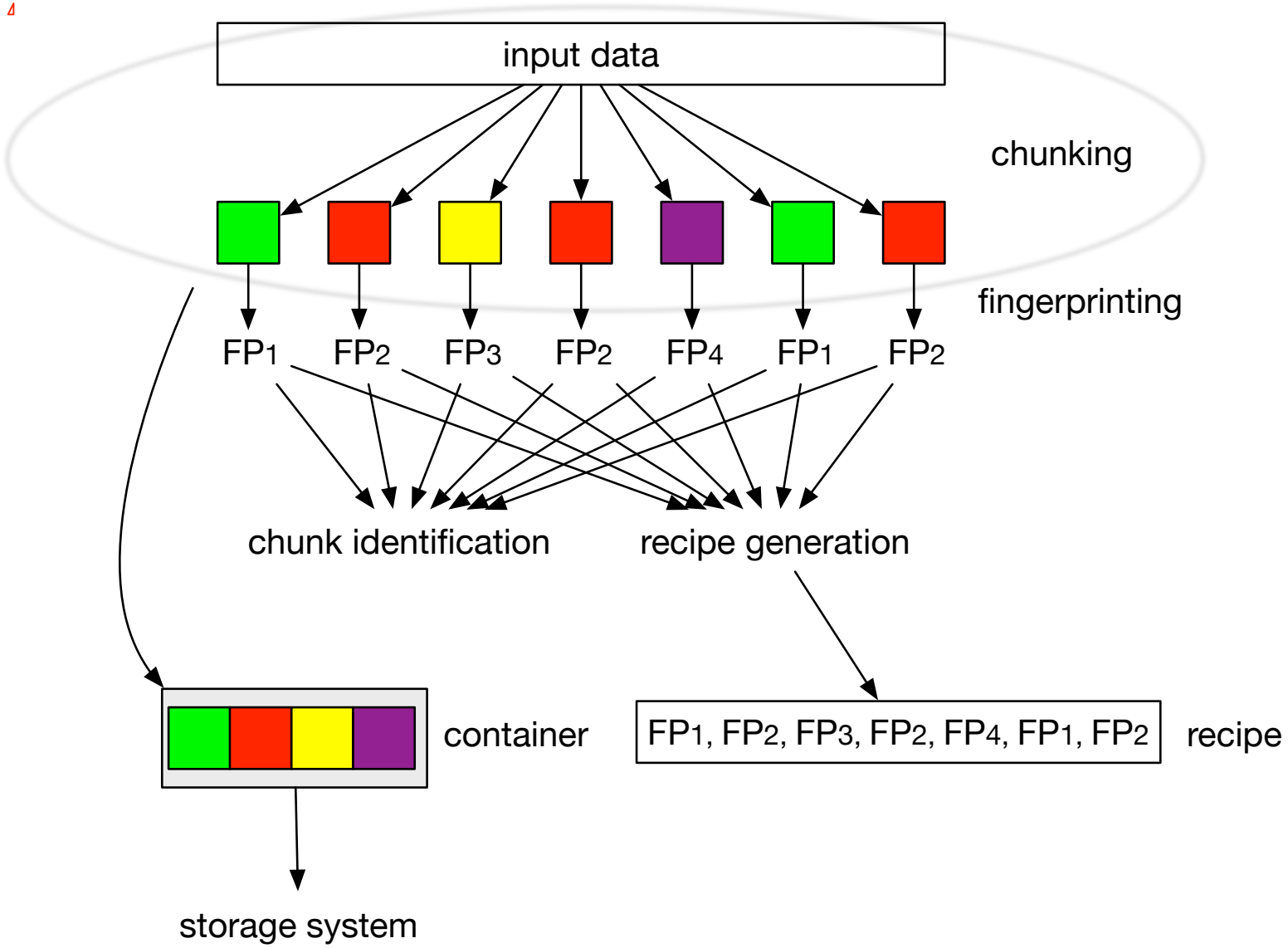
Motivation

- Deduplication: popular compression technique in (backup) storage systems
- First systems around 2000
- Today: changed requirements
  - Before: few (big) streams
  - Now: many streams (example: cloud backup)
- Traditional performance optimizations are less efficient for many streams
- Our Contribution: deduplication approach tailored for many streams

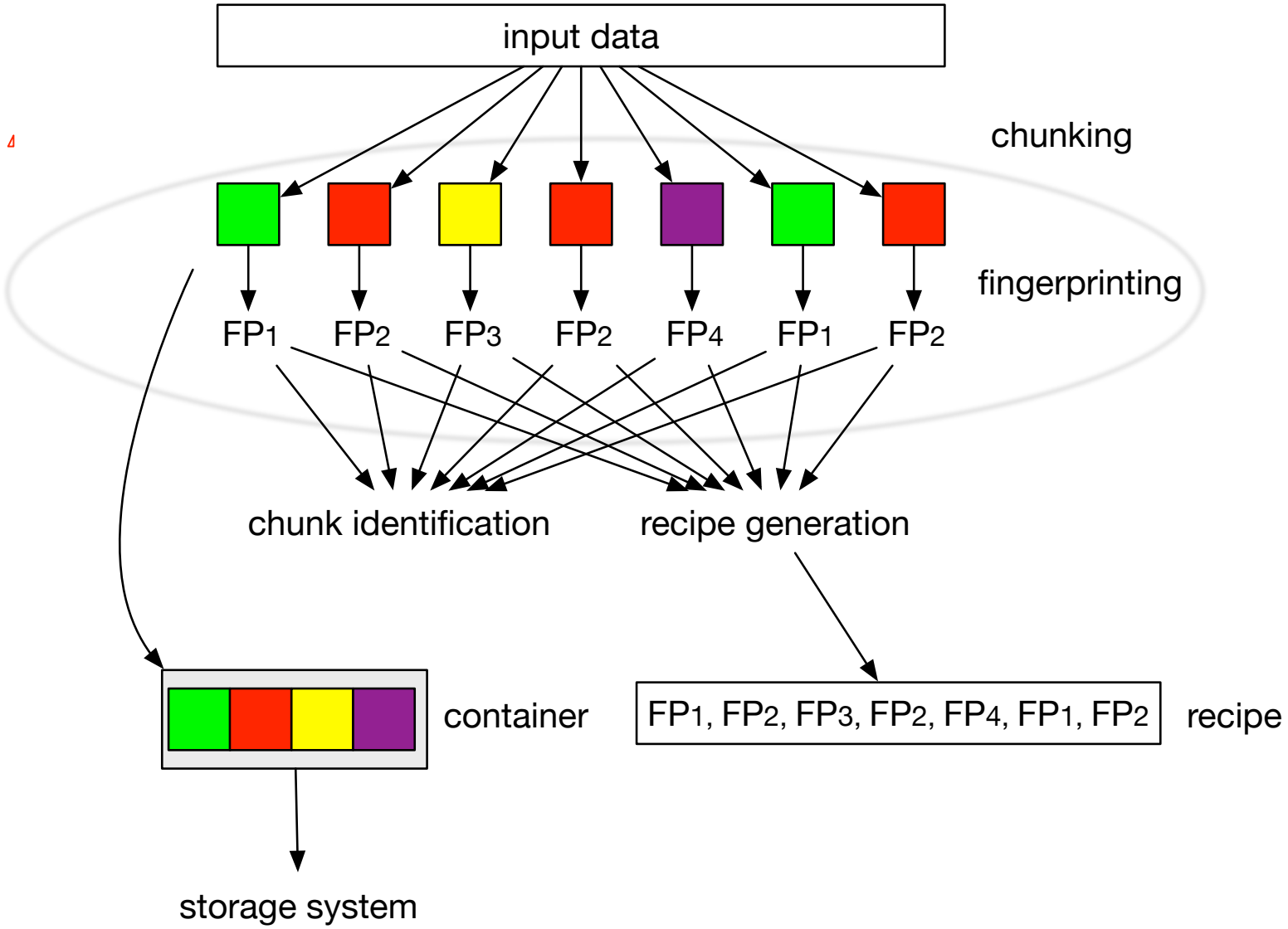
Reminder: Deduplication



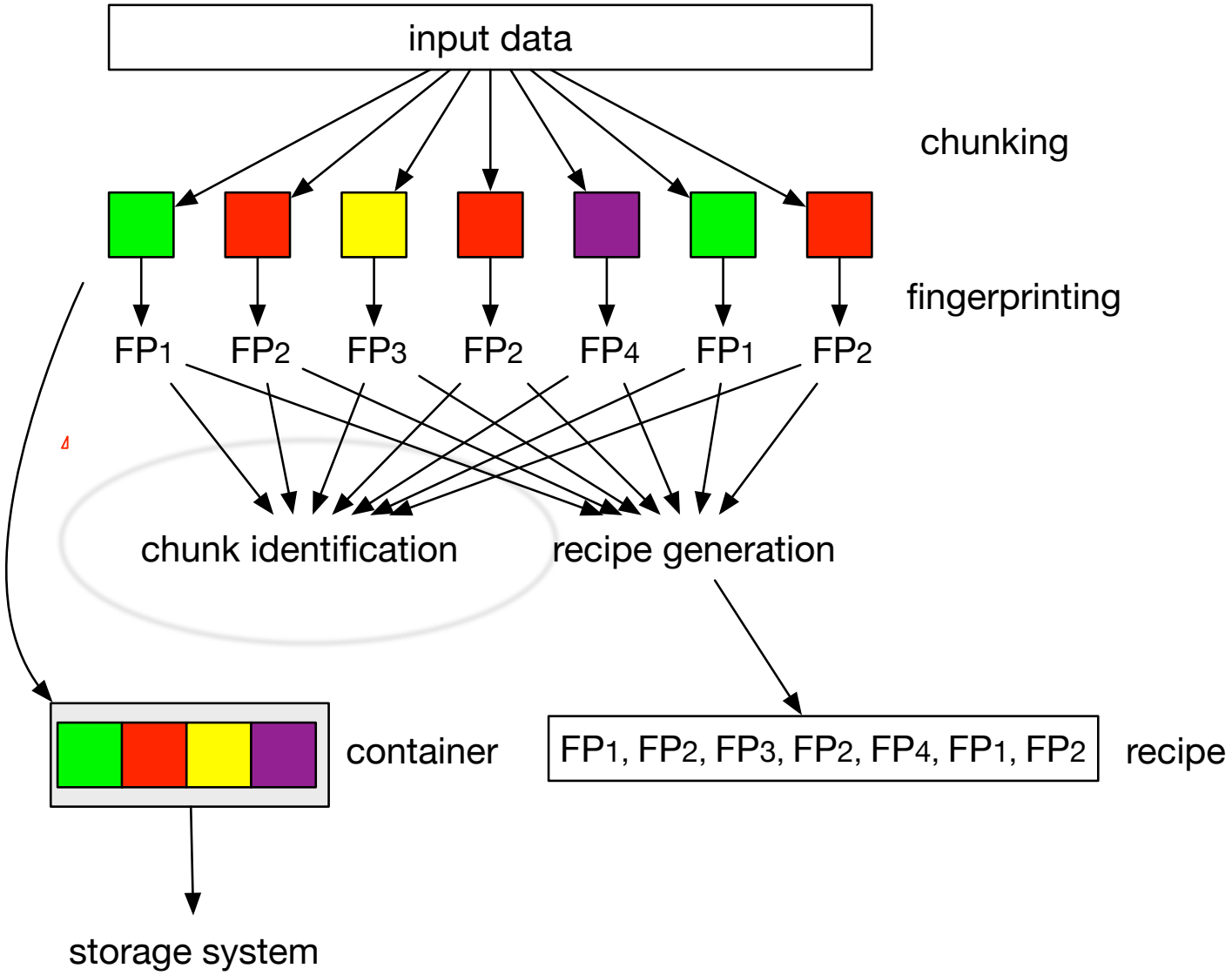
Reminder: Deduplication



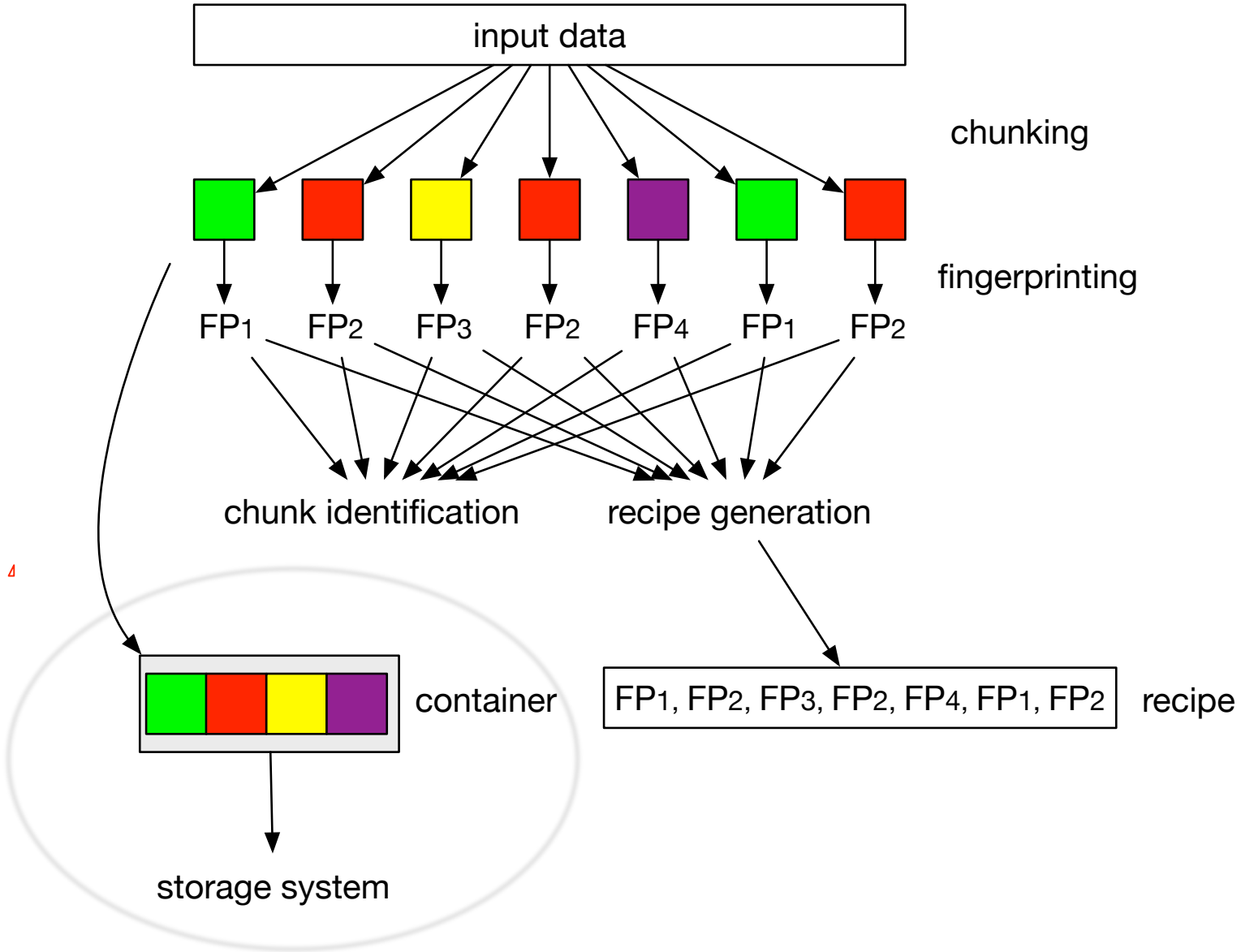
Reminder: Deduplication



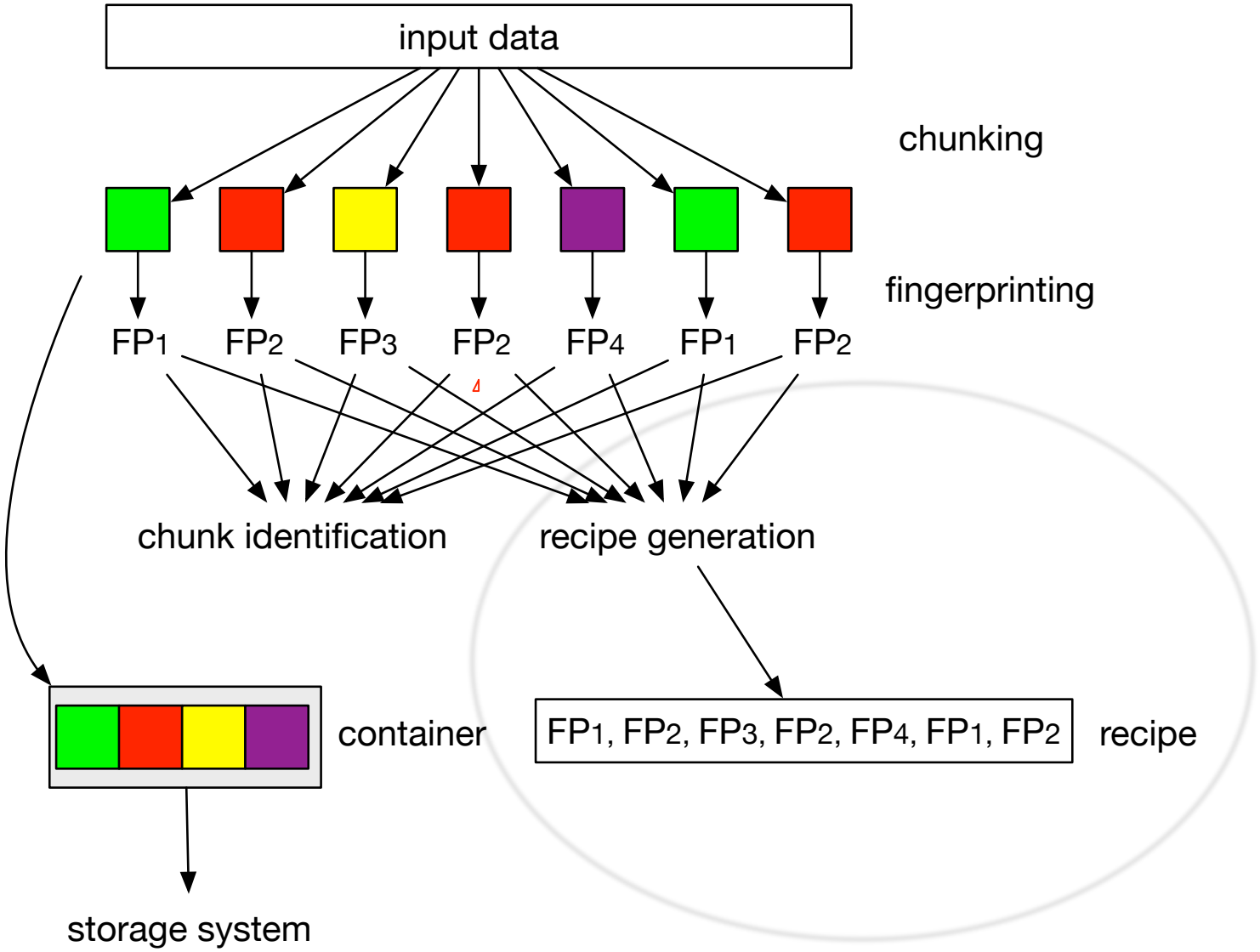
Reminder: Deduplication



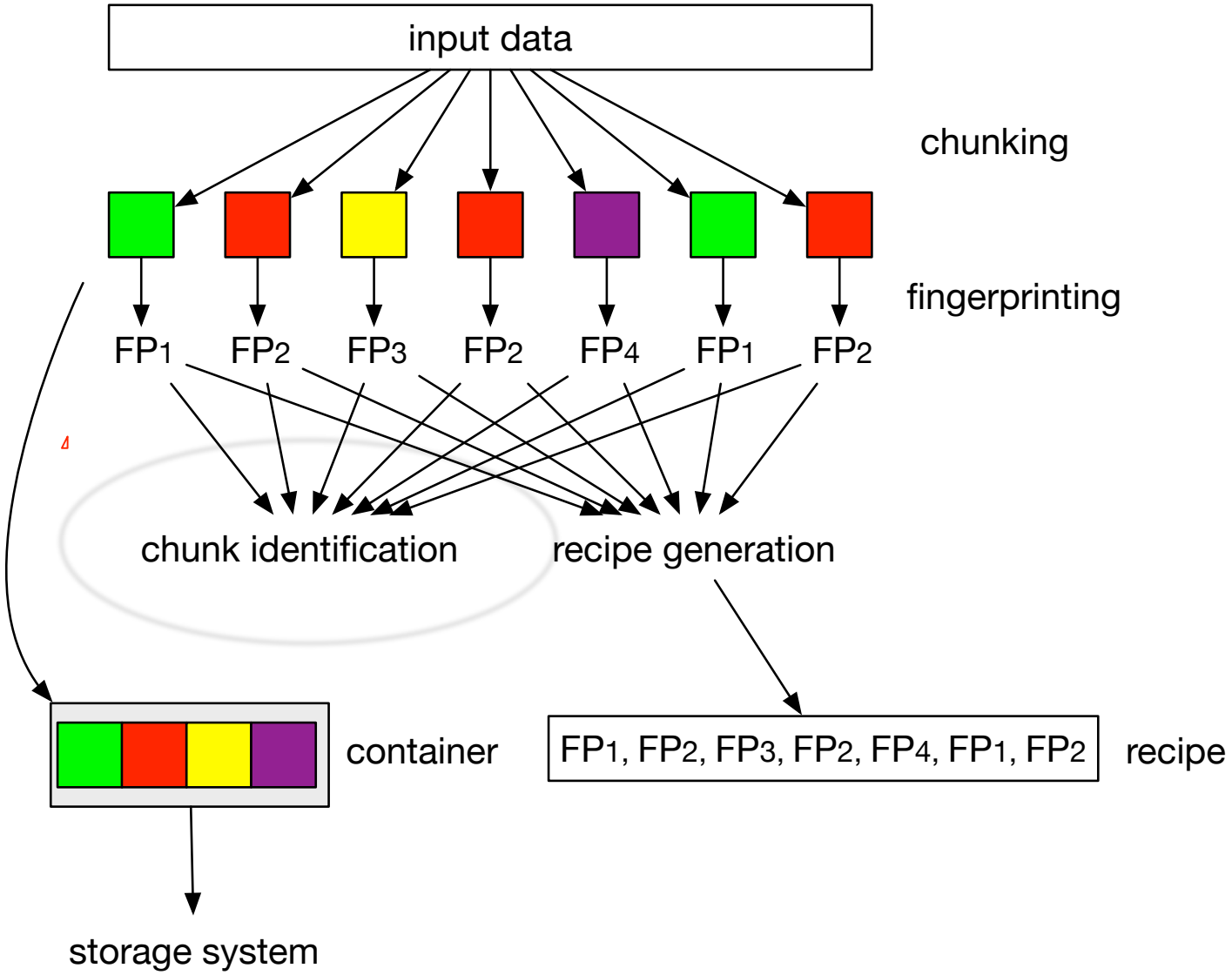
Reminder: Deduplication



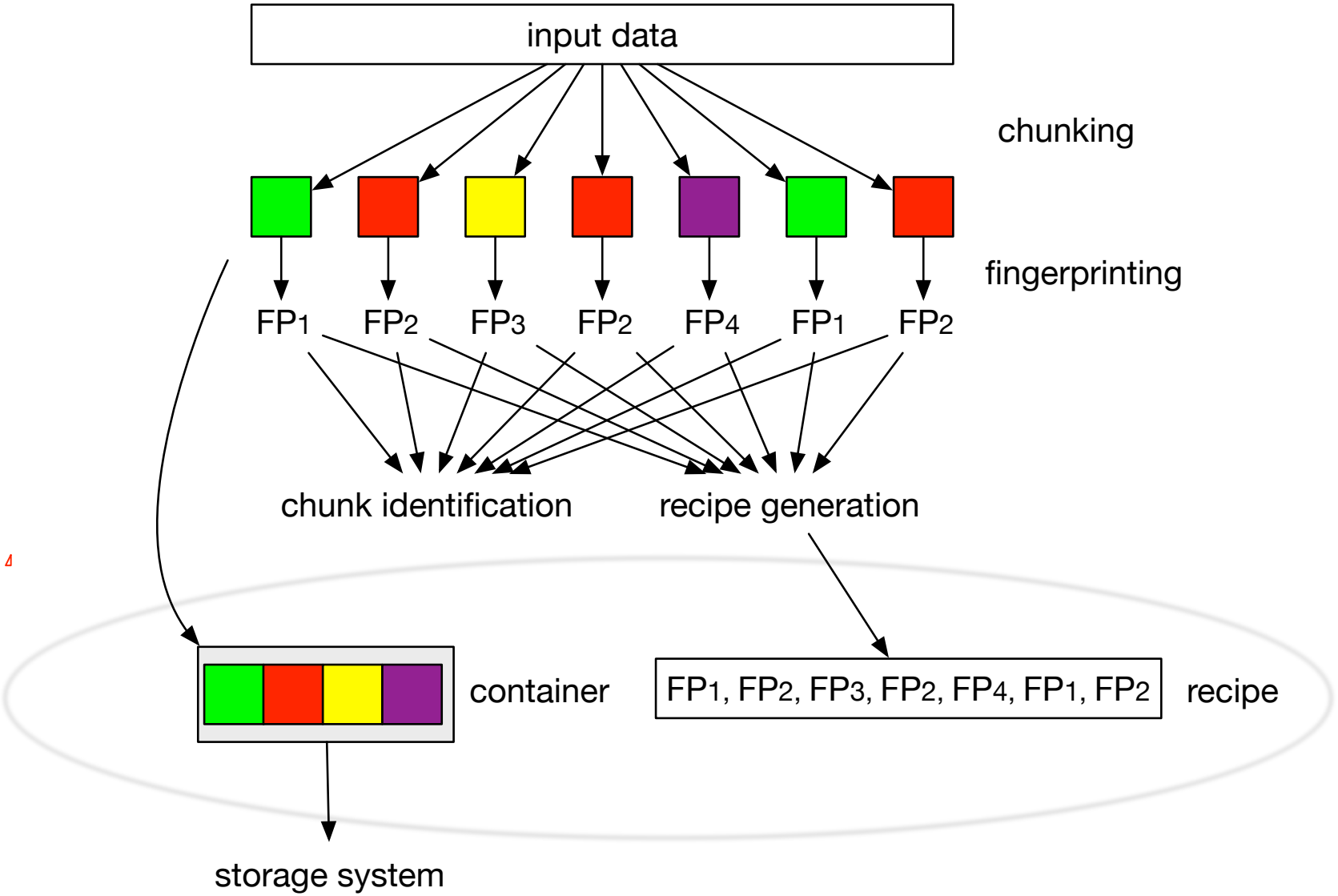
Reminder: Deduplication



Reminder: Deduplication



Reminder: Deduplication

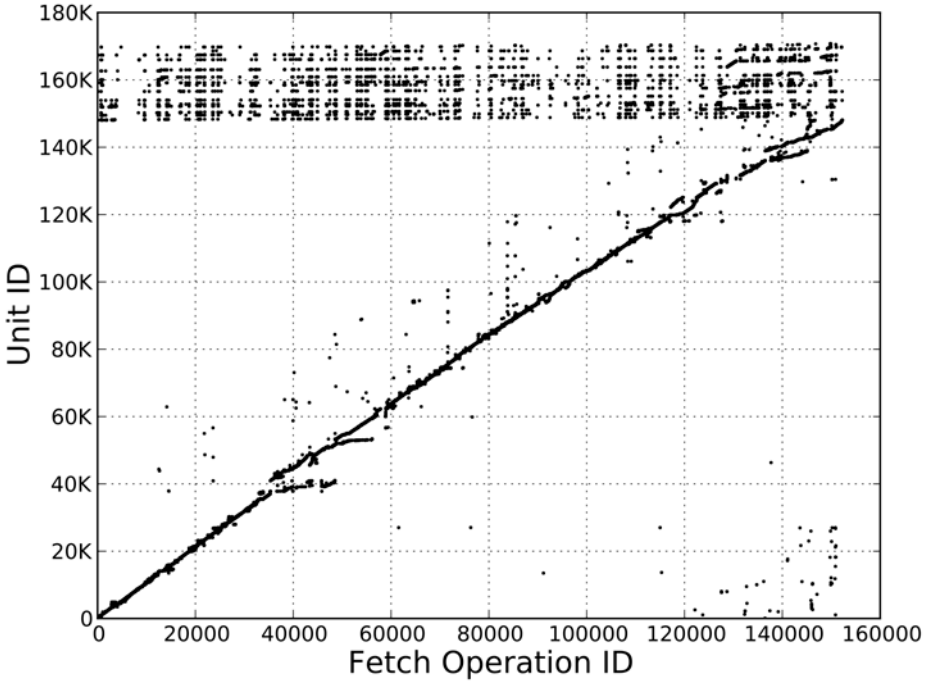


Scaling up to 1000+ streams...

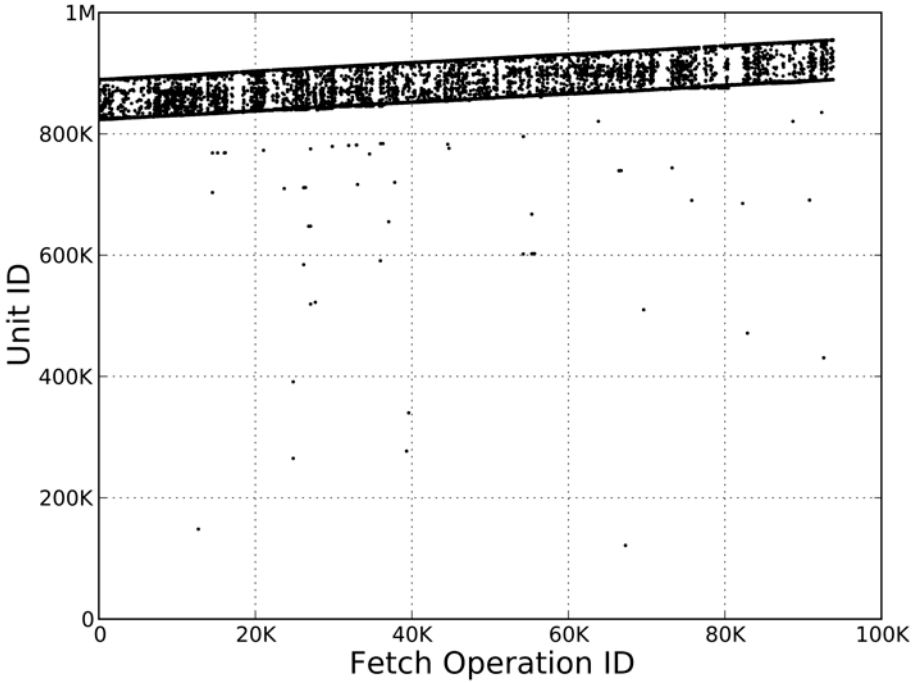
- Low memory per stream
  - Reduced cache efficiency → more I/O operations
- Each stream generates own I/O pattern
- At this scale: stream-locality utilization becomes less effective

Patterns for 1 stream

DDFS

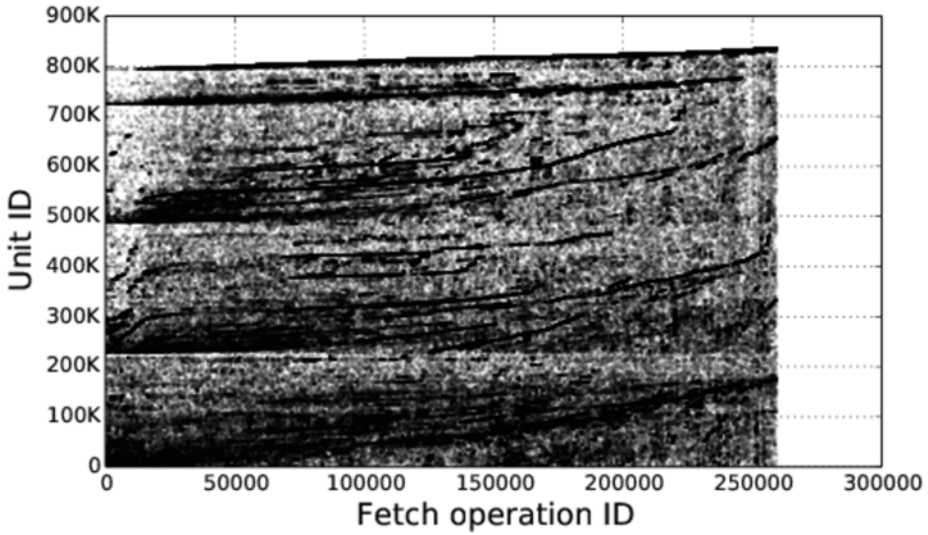


Sparse Indexing

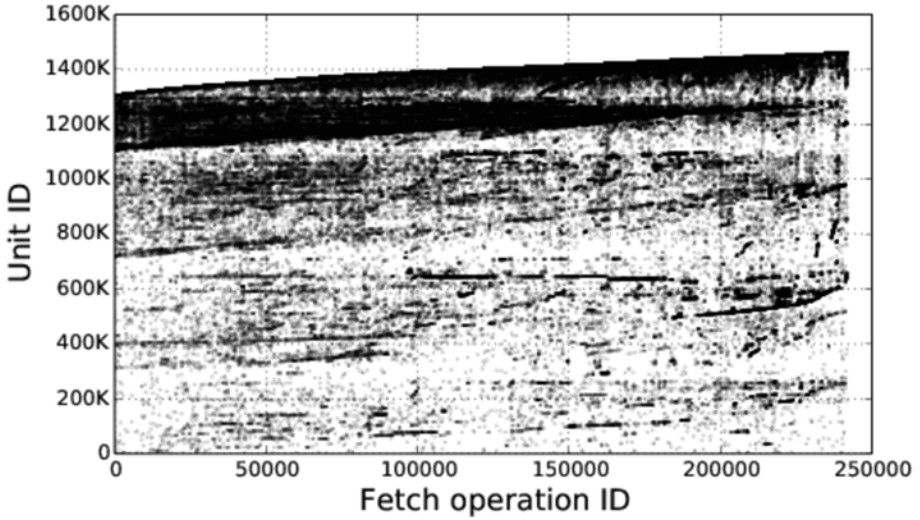


Patterns for 128 streams

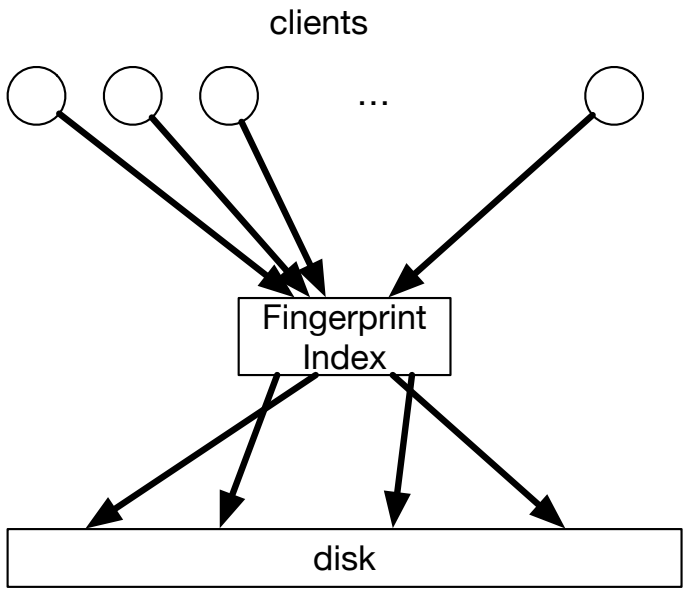
DDFS



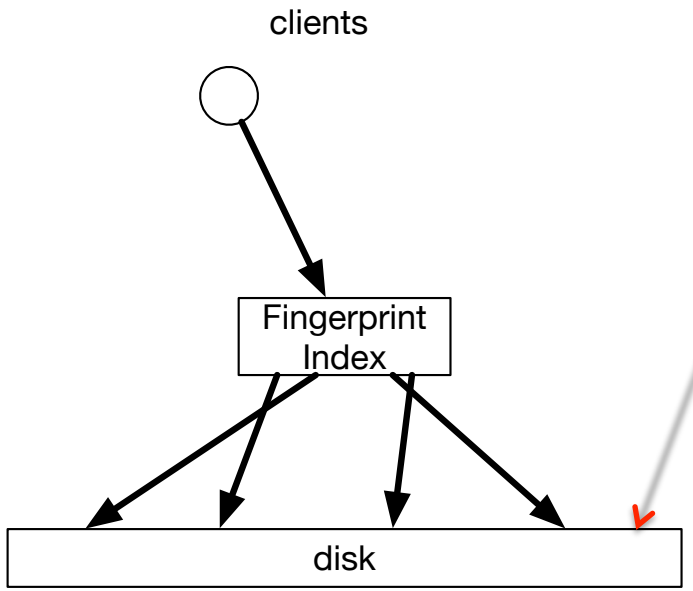
Sparse Indexing



Scaling up to 1000+ streams...



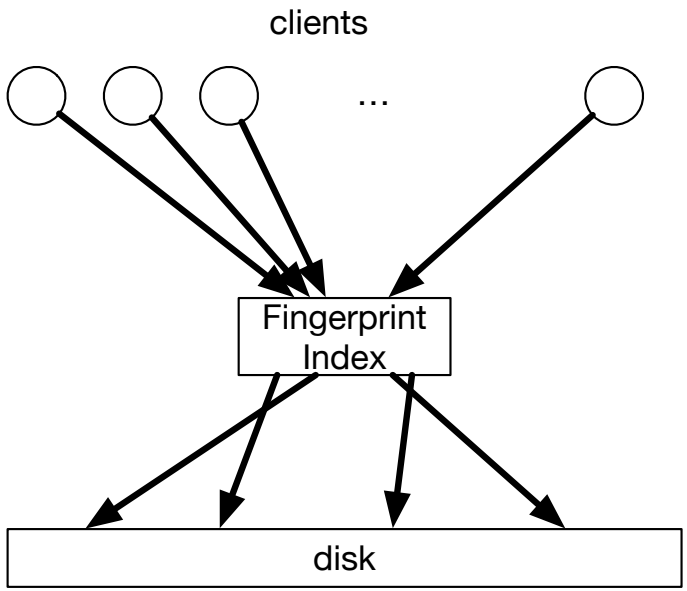
Scaling up to 1000+ streams...



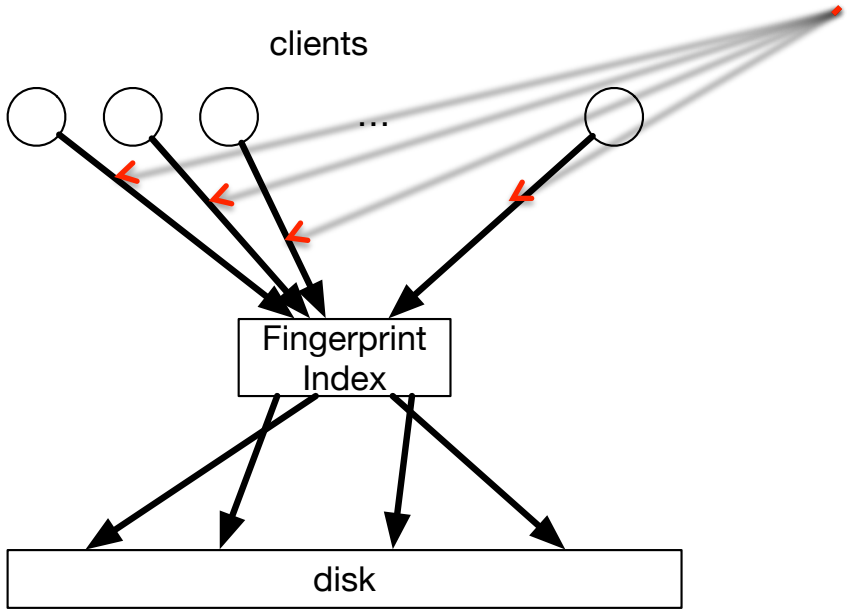
With a single client:

- Sort index entries according to stream
- Almost sequential access

Scaling up to 1000+ streams...



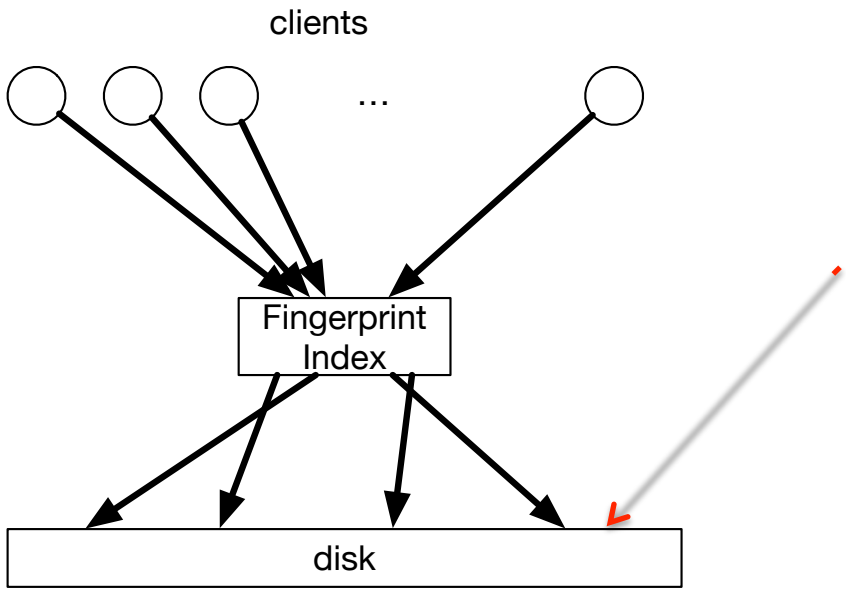
Scaling up to 1000+ streams...



Step 1: establish same fingerprint arrival order

- Sort the fingerprints in clients

Scaling up to 1000+ streams...



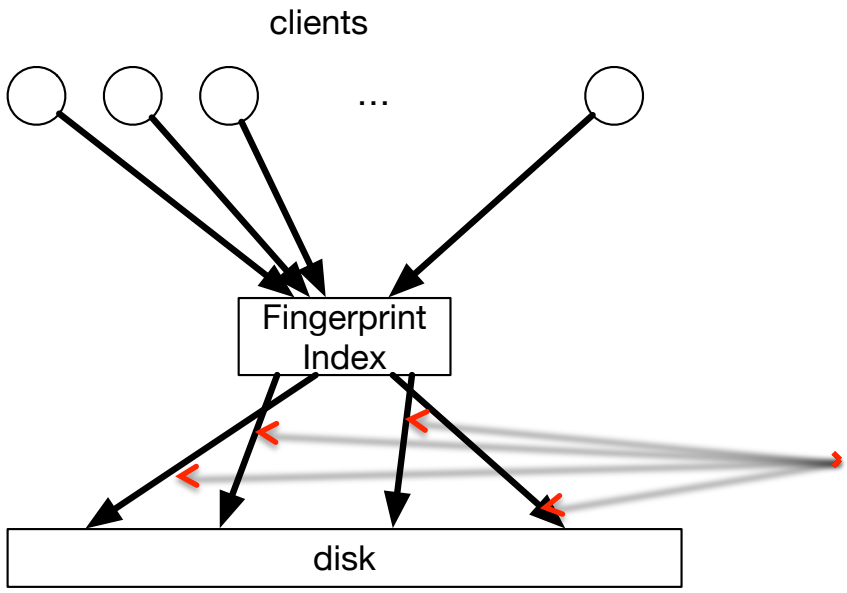
Step 1: establish same fingerprint arrival order

- Sort the fingerprints in clients

Step 2: hold index entries in the same ordering

- Given by index implementation (LSM-Tree)

Scaling up to 1000+ streams...



Step 1: establish same fingerprint arrival order

- Sort the fingerprints in clients

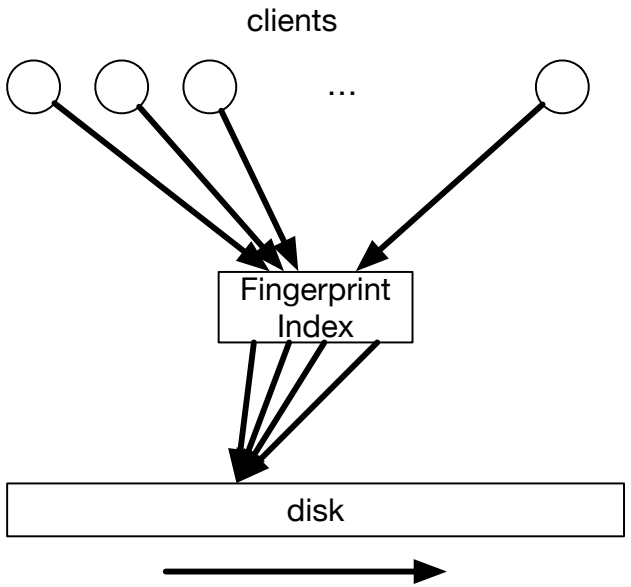
Step 2: hold index entries in the same ordering

- Given by index implementation (LSM-Tree)

Step 3: synchronize stream processing

- Simple merge

Scaling up to 1000+ streams...

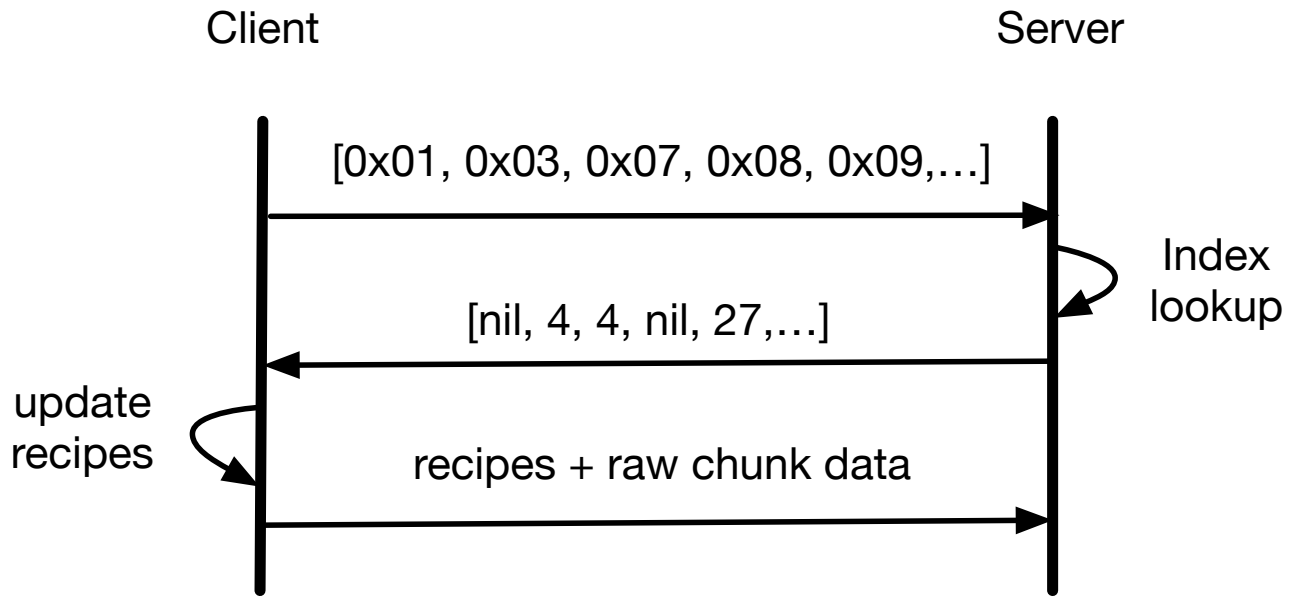


Result:

- Sequential I/O, independently from #clients
- High index locality: Max 1 I/O per index region (= index page)
- No interference among the streams

New Problem

- How to restore data?
  - Before: process and restore in same order
  - Now: process order != restore order
- Clients chunk & fingerprint data
- Send fingerprints first, send chunks in orig. order



More Problems

- What to do with weak clients?
- What to do with weak interconnects?
- How to scale to multiple servers?

Evaluation

1. Comparison with other systems
2. Scaling properties a prototype implementation (SCI)

Comparison with DDFS and SI

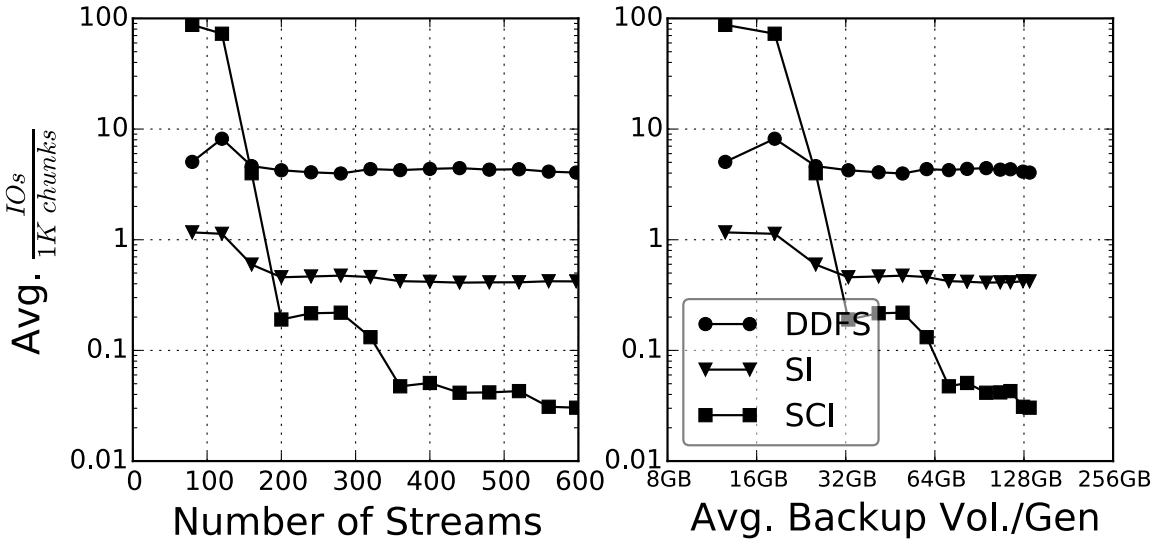
- DDFS: Data Domain Deduplication File System
  - Exact deduplication
- SI: Sparse Indexing (HP)
  - Approximate deduplication

- Data Sets:
  - HOME: 597 diff. streams, 7TB total, home directories
  - Microsoft: 140 diff. streams, 49TB total, workstations

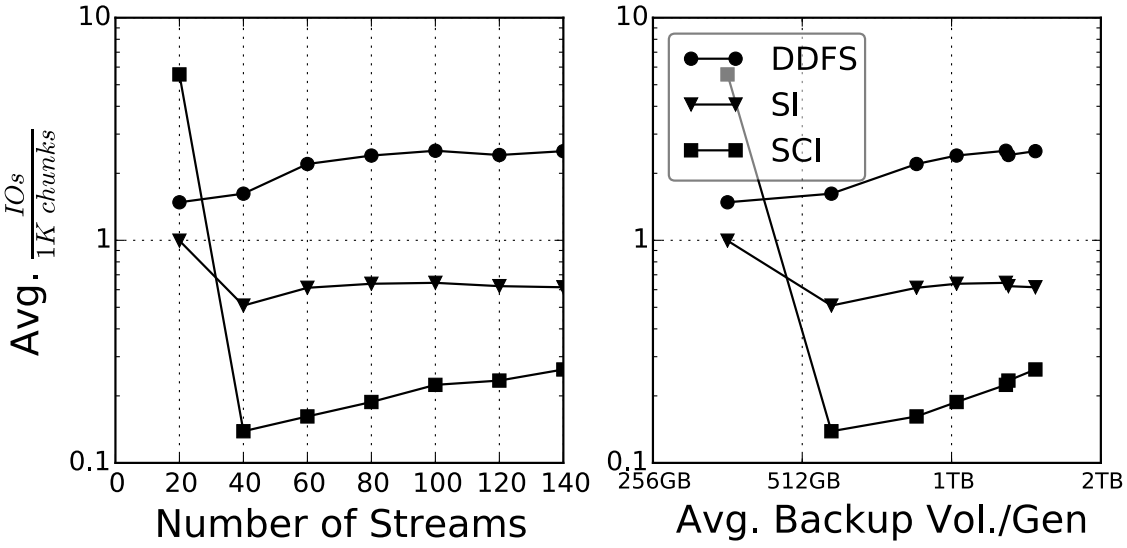
- Metric: Average number of generated I/Os per 1K chunks

Comparison with DDFS and SI

HOME

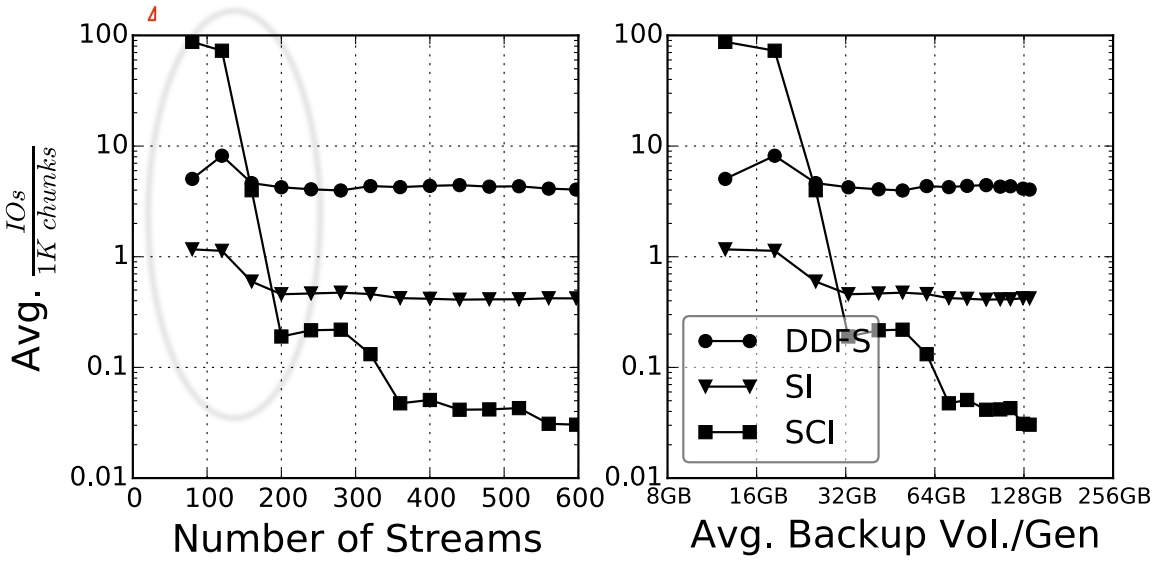


Microsoft

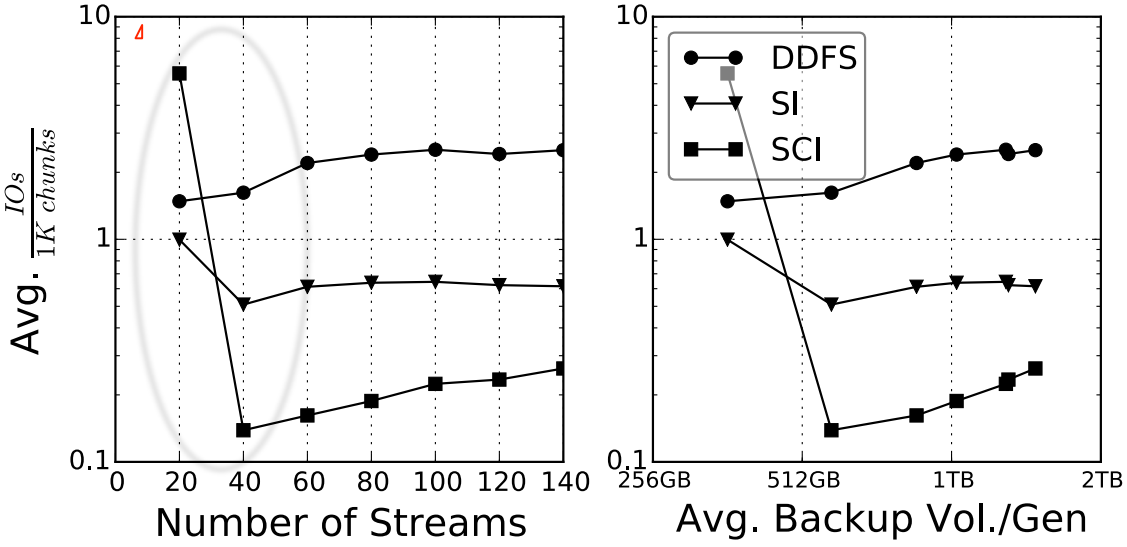


Comparison with DDFS and SI

HOME

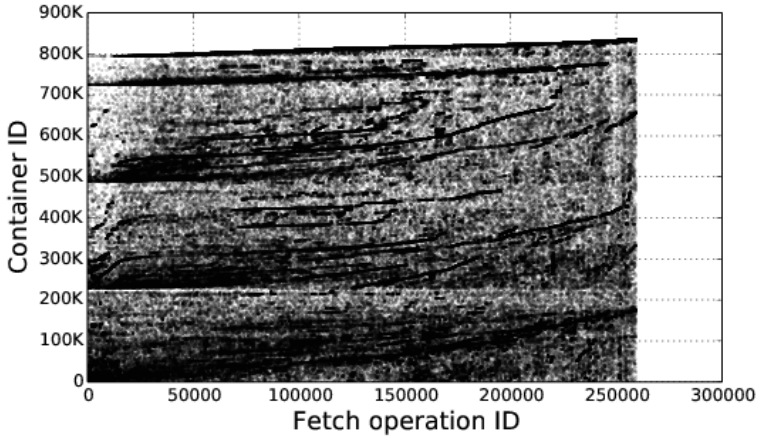


Microsoft

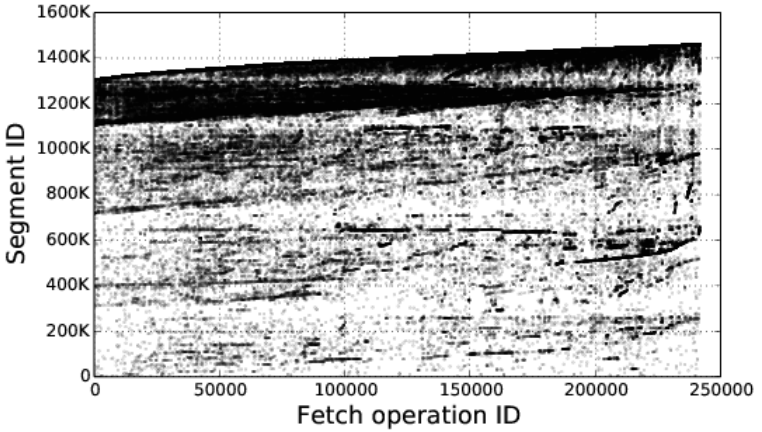


Patterns for Microsoft, 28th backup gen.

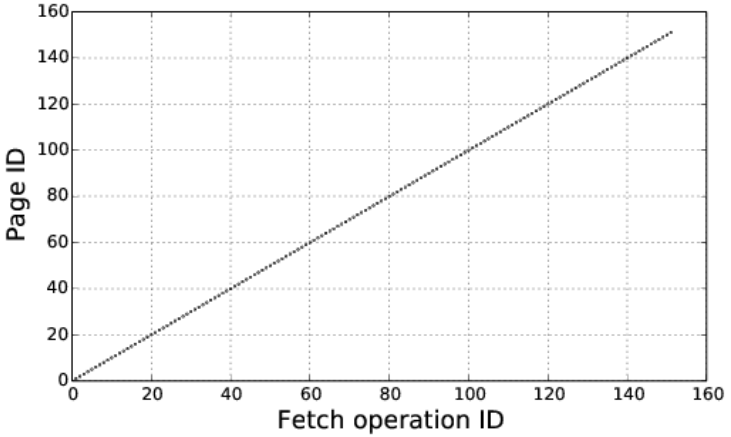
DDFS



Sparse Indexing

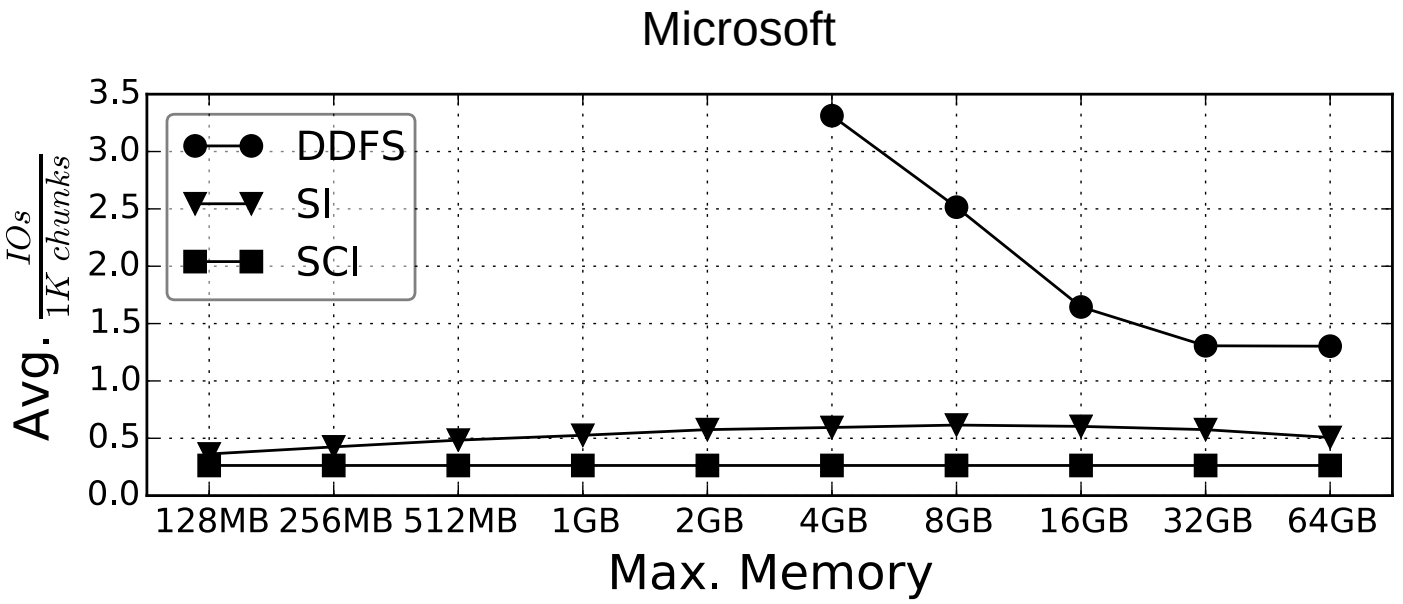


SCI



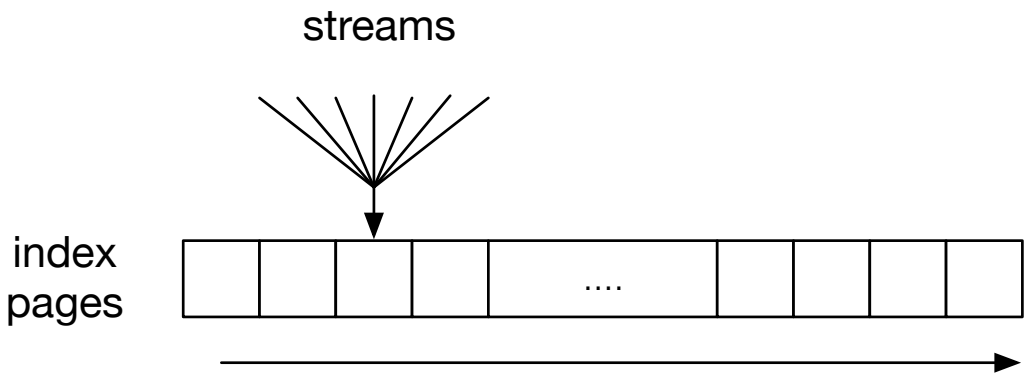
Memory Consumption

- In Simulations: used 8GB
- Today: much more main memory
- Question: How do the systems compare for more memory?



Prototype

- Questions:
  - How fast is the chunk identification?
  - What is the bottleneck?
- Backup volume (per client)
  - Bigger: more checks p. index page → higher CPU load
- Index size
  - Bigger: more index pages → higher HDD load



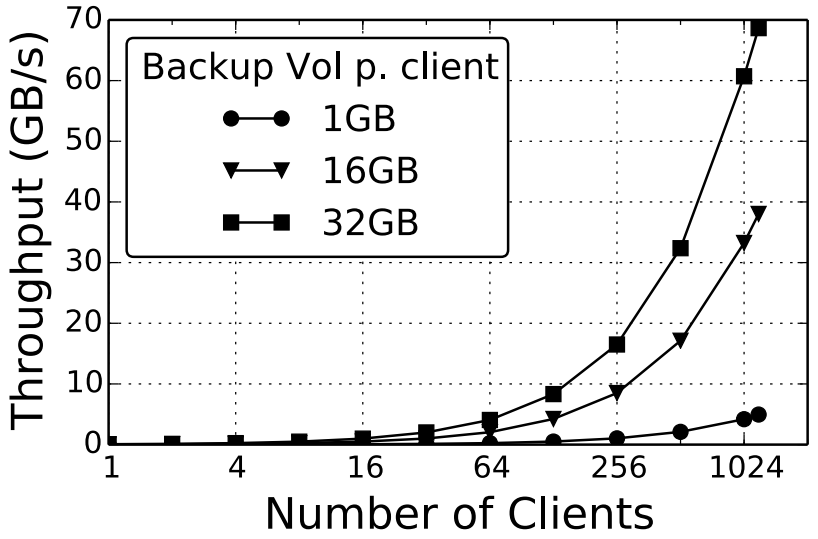
Prototype

- Hardware:
  - Intel Xeon, 4 cores @3.3GHz
  - 16GB RAM
  - 10Gbit ethernet
  - Single HDD

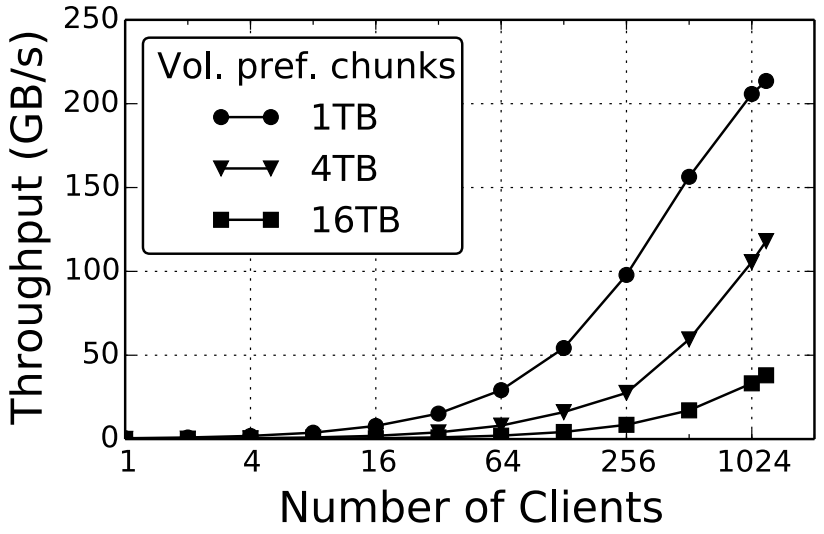
- Used artificial data
- Assumptions:
  - 8KB chunks
  - 90% deduplication ratio
  - 50% compressability of raw data

Scaling properties of the prototype

backup size: variable
fingerprint index size: fixed



backup size: fixed
fingerprint index size: variable



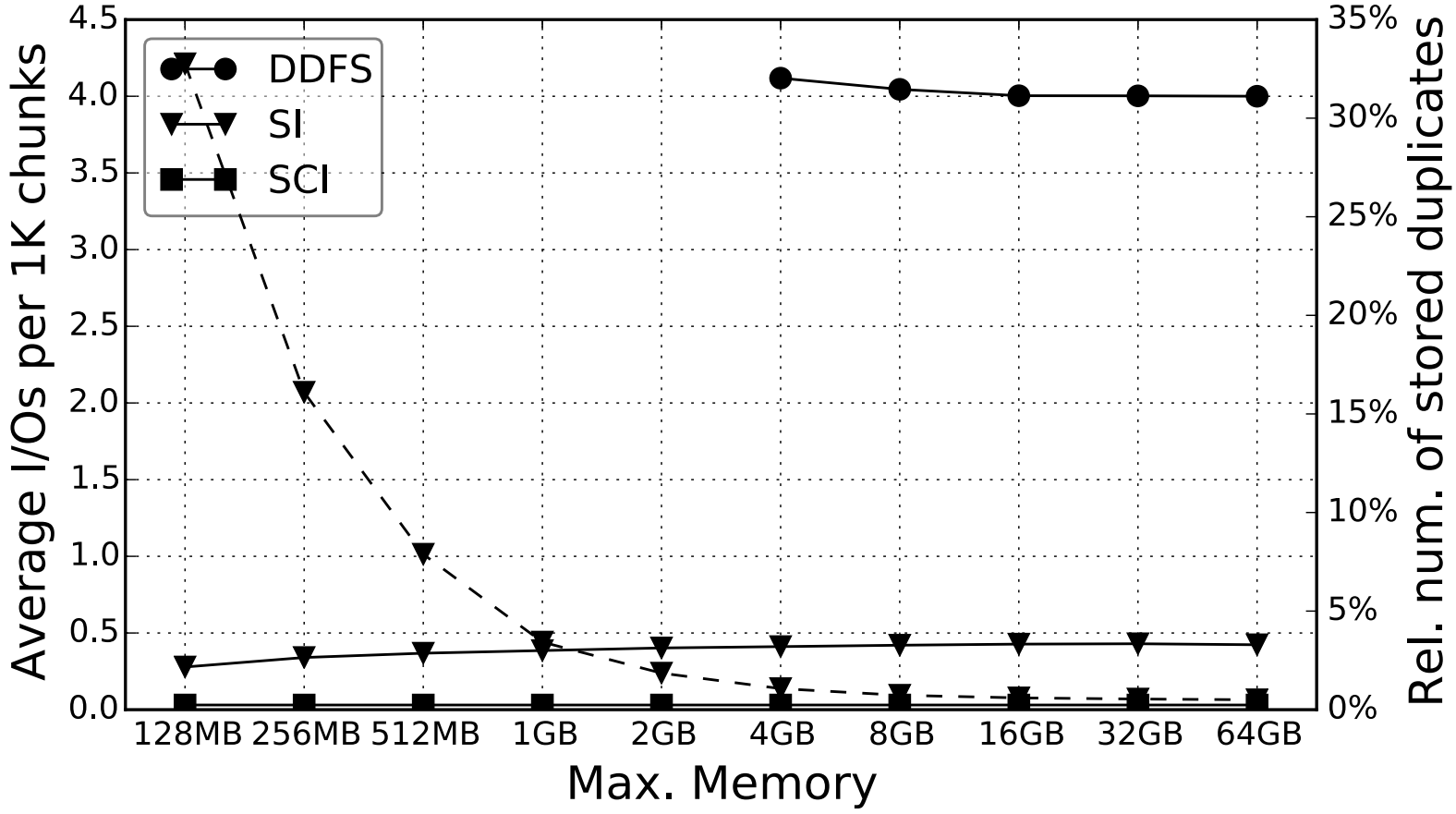
Conclusion

- SCI: Sorted chunk indexing
- Core idea: enforce stream processing to
  - access same index region
  - at the same time
  - in the same order
- Works best with many streams + big backups
  - Low I/Os per 1K chunks
  - Sequential disk access
  - Eliminates the chunk identification as bottleneck

Thank you!

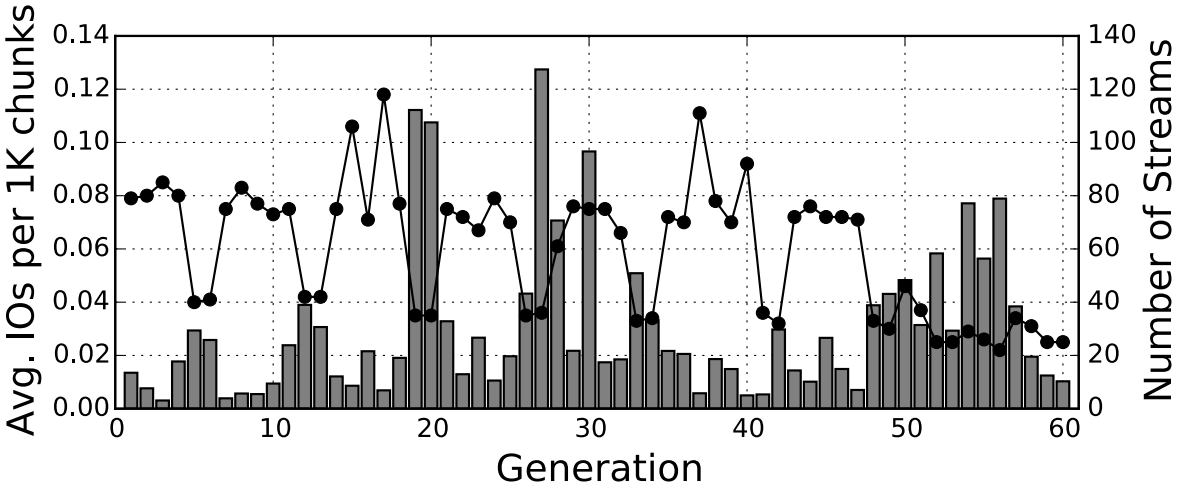
Questions?

Diff. Memory usage, HOME

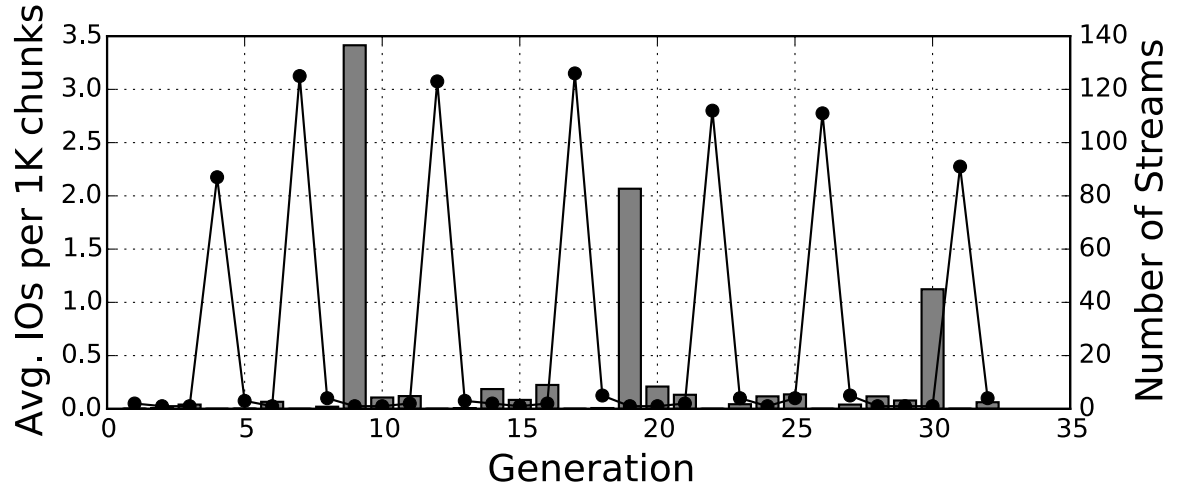


SCI per generation

HOME

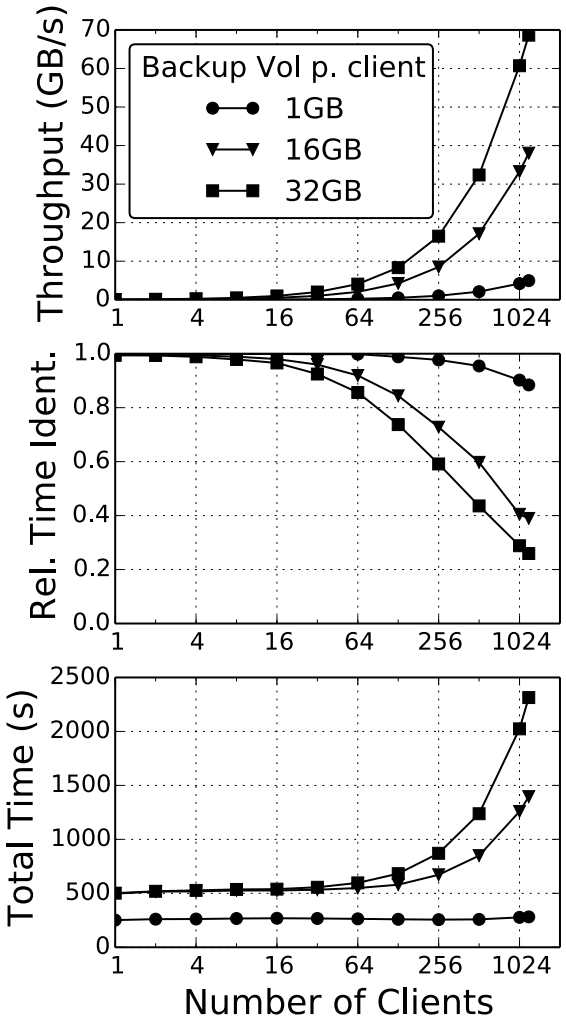


Microsoft

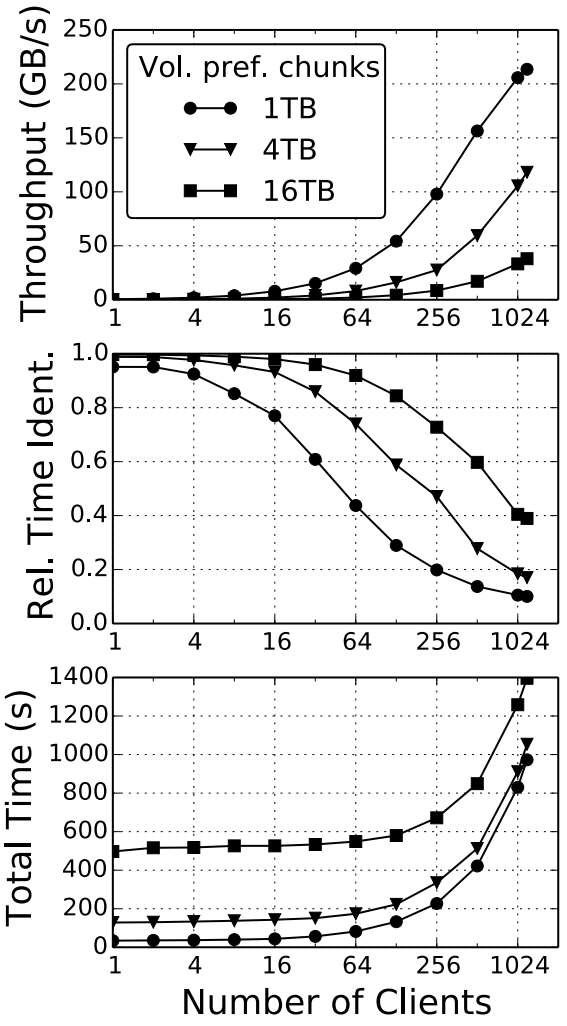


Scaling properties of the prototype

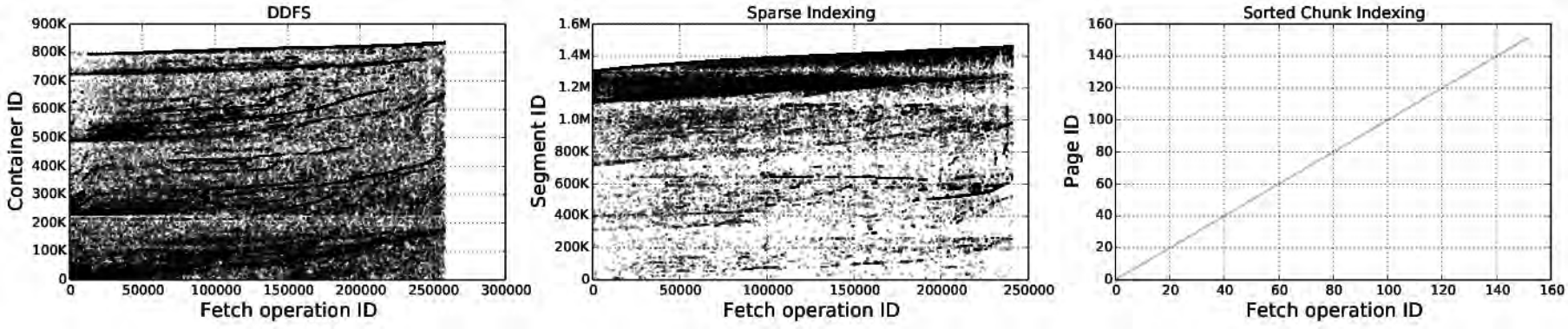
backup size: variable
fingerprint index size: fixed



backup size: fixed
fingerprint index size: variable



IO Patterns


DDFS
Container ID
900K
800K
700K
600K
500K
400K
300K
200K
100K
0
0
50000
100000
150000
200000
250000
300000
Fetch operation ID
Sparse Indexing
Segment ID
1.6M
1.4M
1.2M
1.0M
800K
600K
400K
200K
0
0
50000
100000
150000
200000
250000
Fetch operation ID
Sorted Chunk Indexing
Page ID
160
140
120
100
80
60
40
20
0
0
20
40
60
80
100
120
140
160
Fetch operation ID