Tiny-Tail Flash
Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs

Shiqin Yan, Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman*, Andrew Chien, and Haryadi S. Gunawi
Why SSDs don't perform

From their earliest days, people have reported that SSDs were not providing the performance they expected. As SSDs age, for instance, they get slower. Here's why.

Google: Taming The Long Latency Tail - When More Machines Equals Worse Results

Why it’s hard to meet SLAs with SSDs

Why SSDs don't perform

From their earliest days, people have reported that SSDs were not providing the performance they expected. As SSDs age, for instance, they get slower. Here's why.

**Google: Taming The Long Latency Tail - When More Machines Equals Worse Results**

“if your read is stuck behind an erase you may have wait 10s of milliseconds. That’s a 100x increase in latency variance”

Why it’s hard to meet SLAs with SSDs


The Tail at Scale [CACM’13]

Reads + Writes

Clean/Empty SSD
Reads + Writes

Clean/Empty SSD

Read Latency

Time
Reads + Writes

Clean/Empty SSD

Read Latency

0.3ms

Time

TTFlash @ FAST'17
Reads + Writes

Clean/Empty SSD

Read Latency

Time

Convert to CDF

Percentile

100%

NoGC

0.3ms

Read Latency

0.3ms

80%

TTFFlash @ FAST'17
Reads + Writes

Clean/Empty SSD

Convert to CDF

NoGC

Percentile

0.3ms

0.3ms

Read Latency

Time

TTFFlash @ FAST’17
Reads + Writes

Clean/Empty SSD

0.3ms

Read Latency

Time

Convert to CDF

Percentile

NoGC

0.3ms

Read Latency

TTFlash @ FAST’17
Reads + Writes

Clean/Empty SSD

Read Latency

Time

Percentile

NoGC

Convert to CDF

0.3ms

100%

80%

0.3ms

Read Latency
Aged/Full SSD
Reads + Writes

Aged/Full SSD
Reads + Writes

Aged/Full SSD

Time

Read Latency

0.3ms

80 ms!
Reads + Writes → Aged/Full SSD

Read Latency

Time

Percentile

0.3ms

80 ms!

NoGC

100%

80%

0.3ms

Read Latency

80s

with GC
Reads + Writes

Aged/Full SSD

Read Latency

Time

0.3ms

NoGC

100%

3%

≥5 ms

with GC

80 ms!
Reads + Writes

Aged/Full SSD

NoGC

≥5 ms

Long tail!
Reads + Writes → Aged/Full SSD

80 ms!

Objective: cut tail

Long tail!
How GC delays read I/Os?
How GC delays read I/Os?
How GC delays read I/Os?
How GC delays read I/Os?
How GC delays read I/Os?

A GC moves tens of valid pages!
How GC delays read I/Os?

A GC moves tens of valid pages!

which makes channel/chips busy for tens of ms!
How GC delays read I/Os?

A GC moves tens of valid pages!

which makes channel/chips busy for tens of ms!
How GC delays read I/Os?

A GC moves tens of valid pages!

which makes channel/chips busy for tens of ms!
How to cut tail latencies?
How to cut tail latencies?

Tail-tolerant techniques in distributed/storage systems:
Leverage redundancy to cut tail!
How to cut tail latencies?

Tail-tolerant techniques in distributed/storage systems:
*Leverage redundancy to cut tail!*

**Full Stripe Read**

RAID:

![Diagram showing RAID]
How to cut tail latencies?

Tail-tolerant techniques in distributed/storage systems:
*Leverage redundancy to cut tail!*

**RAID:**

- A
- B
- C

- Fast
- Tail!

- Slow / busy

**Full Stripe Read**
How to cut tail latencies?

Tail-tolerant techniques in distributed/storage systems:

Leverage redundancy to cut tail!

RAID:

Full Stripe Read

\[ C = \text{XOR}(A, B, P) \]

- Fast
- Tail!
- Slow / busy

Flash @ FAST’17
How to cut tails in SSD?
How to cut tails in SSD?

Error rate increases $\rightarrow$ RAIN (Redundant Array of Independent NAND)
How to cut tails in SSD?

Error rate increases $\Rightarrow$ **RAIN** (Redundant Array of Independent NAND)

Similarly, we leverage RAIN to cut “tails”!

**Full Stripe Read**

```
A B C
```

**SSD:**

```
A B C
```

GC
How to cut tails in SSD?

Error rate increases $\rightarrow$ RAIN (Redundant Array of Independent NAND)

Similarly, we leverage RAIN to cut “tails”!

Full Stripe Read

SSD:
How to cut tails in SSD?

Error rate increases \(\rightarrow\) **RAIN** (Redundant Array of Independent NAND)

Similarly, we leverage RAIN to cut “tails”!

**Full Stripe Read**

\[
\begin{align*}
C &= \text{XOR} (B, C, P) \\
\text{fast} \\
\text{slow!}
\end{align*}
\]

SSD:
Contribution

New techniques:

Current SSD technology: RAIN (Parity-based Redundancy)
Contribution

I. Plane-Blocking GC

New techniques:

Current SSD technology: RAIN (Parity-based Redundancy)
 Contribution

New techniques:

I. Plane-Blocking GC
II. GC-Tolerant Read

Current SSD technology: **RAIN** (Parity-based Redundancy)
Contribution

New techniques:

I. Plane-Blocking GC
II. GC-Tolerant Read
III. Rotating GC

Current SSD technology: RAIN (Parity-based Redundancy)
Contribution

New techniques:

I. Plane-Blocking GC
II. GC-Tolerant Read
III. Rotating GC
IV. GC-Tolerant Flush

Current SSD technology: RAIN
(Parity-based Redundancy)
Results

CDF (Percentile)

Latency

TTFlash @ FAST’17
Results

CDF (Percentile)

NoGC

Base

0.3ms  Latency  80ms

95%

100%
Results

CDF (Percentile)

- NoGC
- +Plane-Blocking
- Base

Latency

0.3ms  80ms
Results

CDF (Percentile)

NoGC

+GC-Tolerant Read

+Plane-Blocking

Base

TTFlash @ FAST'17
# Results

[Graph showing CDF (Percentile) for latency with various configurations: NoGC, +Rotating GC, +GC-Tolerant Read, and +Plane-Blocking.]

- **NoGC**: Fastest performance, reaching 100% at 0.3ms.
- **+Rotating GC**: Improved over NoGC, with a slight increase in latency.
- **+GC-Tolerant Read**: Further improvement, with minimal impact on latency.
- **+Plane-Blocking**: Significantly slower than NoGC, not reaching 100% until 80ms.

TTFlash @ FAST'17
Tiny tail!
ttFlash: 1-3x slower than NoGC
Base: 5-138x slower than NoGC

Overall results achieved:

Between 99 - 99.99\textsuperscript{th} percentiles:
Outline

- Introduction
- **Background**
- Tiny-Tail Flash Design
- Evaluation, limitations, conclusion

TTFlash @ FAST'17
SSD Internals
SSD Internals

$C_0$  $C_1$  $C_N$

\[\ldots\]
SSD Internals

$C_0$, $C_1$, $C_N$

Chip

Die [0]

Plane[0]

Plane[N]

Die [1]
SSD Internals

Plane

$C_0$ $C_1$ $C_N$

Chip

Plane

Valid Page
SSD Controller
1. **read** to controller
   *(check with ECC)*
SSD Controller

1. **read** to controller
   (check with ECC)
2. **write** to another block
for (1 ... # of valid pages):
1. **read** to controller
   (check with ECC)
2. **write** to another block
for (1 … # of valid pages):
   1. read to controller
      (check with ECC)
   2. write to another block
for (1 … # of valid pages):
   1. **read** to controller
      (check with ECC)
   2. **write** to another block
for (1 … # of valid pages):
1. read to controller (check with ECC)
2. write to another block

GCed pages block the channel

SSD Controller

1. read
2. write to another block

blocked!
3. **Erase the old block**
3. **Erase the old block**

*Erase operation block the plane*
blocked!

GCing plane
Channel blocking GC

“Base” approach

GCing plane

blocked!
Outline

- Introduction
- Background
- Tiny-Tail Flash Design
  - Plane-Blocking GC
  - GC-Tolerant Read
  - Rotating GC
  - GC-Tolerant Flush
- Evaluation, limitations, conclusion
Base: Channel Blocking

Plane Blocking

GCing plane

GCing plane

blocked!
Base: Channel Blocking

Plane Blocking

Unblock the channel

Leverage intra-plane copyback support

GCing plane

GCing plane
Base GC Logic:

for (every valid page)
1. flash read+write (over channel)
2. wait
Base GC Logic:
for (every valid page)
1. flash read+write (over channel)
2. wait

Plane Blocking GC Logic:
for (every valid page)
1. flash read+write (inside plane)

SSD Controller

“Intra-plane copyback”
Plane Blocking

SSD Controller

Base GC Logic:

for (every valid page)
1. flash read+write (over channel)
2. wait

Plane Blocking GC Logic:

for (every valid page)
1. flash read+write (inside plane)
2. serve other user I/Os

“Intra-plane copyback”
Base GC Logic:
for (every valid page)
1. flash read+write (over channel)
2. wait

Plane Blocking GC Logic:
for (every valid page)
1. flash read+write (inside plane)
2. serve other user I/Os

SSD Controller

Plane Blocking

“Intra-plane copyback”
**Base GC Logic:**
for (every valid page)
1. flash read+write (over channel)
2. wait

**Plane Blocking GC Logic:**
for (every valid page)
1. flash read+write (inside plane)
2. serve other user I/Os

**Overlap**
1. *intra-plane copyback with*
2. channel usage for other non-GCing planes

**SSD Controller**

---

“*Intra-plane copyback*”

Old block

Empty block

Read Page
3% of I/Os are blocked by GC
3% of I/Os are blocked by GC

NoGC

Only 1.5%

Blocking

Base (Channel-Blocking)

100%

95%

0.3ms

Latency

80ms
Issue 1: No ECC check for garbage-collected pages
  - (will discuss later)
- **Issue 1:** No ECC check for garbage-collected pages
  - *(will discuss later)*

- **Issue 2:**

  - **Read X**
  - **Read Y**
  - **Read Z**

  *GC-ing plane still blocks*
- **Issue 1:** No ECC check for garbage-collected pages
  - *(will discuss later)*

- **Issue 2:**

  ![Diagram showing read operations and delayed processing]

  - Read X
  - Read Y
  - Read Z

  GC-ing plane still blocks delayed!
Outline

- Introduction
- Background

**Tiny-Tail Flash Design**
- Plane-Blocking GC
- RAIN + GC-Tolerant Read
- Rotating GC
- GC-Tolerant Flush

- Evaluation, limitations, conclusion
RAIN

C₀  C₁  C₂  C₃
# RAIN

<table>
<thead>
<tr>
<th></th>
<th>( C_0 )</th>
<th>( C_1 )</th>
<th>( C_2 )</th>
<th>( C_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PG_0</strong></td>
<td>( \square )</td>
<td>( \square )</td>
<td>( \square )</td>
<td>( \square )</td>
</tr>
<tr>
<td><strong>PG_1</strong></td>
<td>( \square )</td>
<td>( \square )</td>
<td>( \square )</td>
<td>( \square )</td>
</tr>
<tr>
<td><strong>PG_2</strong></td>
<td>( \square )</td>
<td>( \square )</td>
<td>( \square )</td>
<td>( \square )</td>
</tr>
</tbody>
</table>
RAIN

LPN (Logical Page #)

<table>
<thead>
<tr>
<th>C₀</th>
<th>C₁</th>
<th>C₂</th>
<th>C₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>PG₀</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PG₁</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PG₂</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rain

LPN (Logical Page #)

Static mapping:
LPN0 → C[0]PG[0]
LPN1 → C[1]PG[0]
...

C₀  C₁  C₂  C₃
PG₀ 0  1  2
PG₁
PG₂
RAIN

LPN (Logical Page #)

Static mapping:
LPN0 $\rightarrow$ C[0]PG[0]
LPN1 $\rightarrow$ C[1]PG[0]

... 

Add parity:
LPN 0, 1, 2 $\rightarrow$ $P_{0,1,2}$
Für die Dauer des Experiments wird ein Atemzugsträger für jede Einzelreihe benötigt. Es gibt eine Vielzahl von Atemzugsträgern auf dem Markt, die unterschiedliche Merkmale aufweisen. Einige der Merkmale sind: 

- Farben: Die Atemzugsträger können in verschiedenen Farben erhältlich sein, was persönliche Präferenzen beeinflussen kann. 
- Material: Die meisten Atemzugsträger bestehen aus einem leichten, atmungsaktivem Material, das es ermöglicht, die Luft zu durchatmen. 
- Handhabung: Einige Atemzugsträger sind einfacher zu handhaben als andere, was besonders bei der langem Verwendung von Bedeutung ist. 
- Preis: Der Preis der Atemzugsträger kann von günstig bis sehr teuer variieren. 
- Versorgung: Es ist wichtig, sicherzustellen, dass die Atemzugsträger regelmäßig und ausreichend versorgt werden.
RAIN enables GC-Tolerant Read

Full Stripe Read

\[
\begin{array}{c|c|c}
0 & 1 & 2 \\
\end{array}
\]
RAIN enables GC-Tolerant Read

Full Stripe Read

fast

tail

0 1 2

0 1 2

P_{0,1,2}
\textbf{RAIN} enables \textbf{GC-Tolerant Read}

\textit{Full Stripe Read}

\[ 2 = \text{XOR} \quad (0, 1, \ P_{0,1,2}) \]
RAIN enables GC-Tolerant Read

Full Stripe Read

2 = XOR (0, 1, P_{0,1,2})

Read in parallel + XOR cost
\(~\text{0.01 ms}\)
RAIN enables GC-Tolerant Read

**Full Stripe Read**

\[ 2 = \text{XOR} \ (0, 1, P_{0,1,2}) \]

Read in parallel + XOR cost ~0.01 ms

Wait for GC 2 to 10s of ms
GC-Tolerant Read

Issue: *partial* stripe read
GC-Tolerant Read

Issue: *partial* stripe read

Partial stripe read: 2

slow!
GC-Tolerant Read

*Issue: partial stripe read*

Partial stripe read: \( \boxed{2} \) slow!

Must generate extra \( N-1 \) reads!
GC-Tolerant Read

**Issue:** partial stripe read

Partial stripe read: slow!

2 = XOR (0, 1, P_{0,1,2})

Must generate extra \( N-1 \) reads!
GC-Tolerant Read

Issue: *partial stripe read*

Partial stripe read: slow!

\[ 2 = \text{XOR} \ (0, 1, P_{0,1,2}) \]

Must generate extra \( N-1 \) reads!

Add *contention* to other \( N-1 \) channels and planes
GC-Tolerant Read

**Issue:** partial stripe read

Partial stripe read: slow!

2 = XOR (0, 1, P_{0,1,2})

Must generate extra \( N-1 \) reads!

Add contention to other \( N-1 \) channels and planes

Convert to full stripe if:

\( T_{\text{extra-reads}} < T_{GC} \)
Issue: more than 1 GCs in a plane group?

Full-stripe read

0 1 2
Issue: **more than 1 GCs in a plane group?**

One parity → cut one tail
Can’t cut two tails!

Full-stripe read

| 0 | 1 | 2 |

0 1 2

P₀,₁,₂
Issue: more than 1 GCs in a plane group?

One parity → cut one tail
Can’t cut two tails!

Full-stripe read

PG₀

0

1

2

P₀,₁,₂

GC

GC
Issue: more than 1 GCs in a plane group?

Full-stripe read

2 tails!

PG₀

GC

GC

P₀,₁,₂

One parity → cut one tail
Can’t cut two tails!
Issue: **more than 1 GCs**
in a plane group?

*One parity → cut one tail
Can’t cut two tails!*

Full-stripe read

2 tails!
**Issue:** more than 1 GCs in a plane group?

Full-stripe read

2 tails!

One parity → cut one tail
Can’t cut two tails!

DOES NOT HELP!
Outline

- Introduction
- Background
- Tiny-Tail Flash Design
  - Plane-Blocking GC
  - GC-Tolerant Read
  - Rotating GC
  - GC-Tolerant Flush
- Evaluation, limitations, conclusion
Rotating GC:
Anytime, at most 1 plane per plane group can perform GC
Rotating GC:

Anytime, at most 1 plane per plane group can perform GC
**Rotating GC:**

Anytime, at most 1 plane per plane group can perform GC

<table>
<thead>
<tr>
<th>PG₀</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>P₀,₁,₂</th>
</tr>
</thead>
</table>

Postpone!
**Rotating GC:**

Anytime, **at most 1 plane per plane group can perform GC**
Rotating GC:
Anytime, at most 1 plane per plane group can perform GC
Rotating GC:
Anytime, at most 1 plane per plane group can perform GC.
Rotating GC:

Anytime, at most 1 plane per plane group can perform GC

Concurrent GCs in different PGs are permitted.
CDF (Percentile)

Latency

0.3ms 80ms

100% 0.5%

95%
Tiny tail!

CDF (Percentile)

Latency

0.3ms

80ms

+Rotating GC

100%

95%
Why still tiny tails?

Small/partial-stripe read

→ Sometimes may be better to wait for GC than adding extra reads/contentions!
Outline

- Tiny-Tail Flash Design
  - Plane-Blocking GC
  - GC-Tolerant Read
  - Rotating GC
  - GC-Tolerant Flush (in paper)

- Evaluation
- Limitations
- Conclusion
Implementation

- **SSDsim** (~2500 LOC)
  - Device simulator

- **VSSIM** (~900 LOC)
  - QEMU/KVM-based
  - Run Linux and applications

- **OpenSSD**
  - Many limitations of the simple programming model

- Future: ttFlash on **OpenChannel SSD**
Evaluation

- Simulator: **SSDsim** (verified against hardware)
- Workload: 6 real-world traces from Microsoft Windows
- Settings and SSD parameters:
  - SSD size: 256GB, **plane group width = 8 planes** (1 parity, 7 data)

<table>
<thead>
<tr>
<th>Sizes</th>
<th>Latencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD Capacity: 256 GB</td>
<td>Page Read 40(\mu s) (flash-to-register)</td>
</tr>
<tr>
<td>#Channels 8</td>
<td>Page Write 800(\mu s) (register-to-flash)</td>
</tr>
<tr>
<td>#Planes/channel 8</td>
<td>Page data transfer 100(\mu s) (via channel)</td>
</tr>
<tr>
<td>Plane size 4 GB</td>
<td>Block erase 2 ms</td>
</tr>
<tr>
<td>#Planes/chip 1**</td>
<td></td>
</tr>
<tr>
<td>#Blocks/plane 4096</td>
<td></td>
</tr>
<tr>
<td>#Pages/block 256</td>
<td></td>
</tr>
<tr>
<td>Page size 4 KB</td>
<td></td>
</tr>
</tbody>
</table>
NoGC

CDF (Percentile)

95%

100%

0.3ms  Latency  80ms
Developer Tools Release Server Trace

NoGC

CDF (Percentile)

95%

100%

0.3ms Latency 80ms
Developer Tools Release Server Trace

CDF (Percentile)

NoGC

Base

Latency

0.3ms

80ms

95%

100%
Developer Tools Release Server Trace
Developer Tools Release Server Trace

- NoGC
- +GC-Tolerant Read
- +Plane-Blocking
- Base

CDF (Percentile)

- 100%
- 95%

Latency

- 0.3ms
- 80ms
Developer Tools Release Server Trace

NoGC + Rotating GC
+GC-Tolerant Read + Plane-Blocking

CDF (Percentile)

95% 100%

0.3ms Latency 80ms

Base

Flash @ FAST’17

Developer Tools Release Server Trace
Developer Tools Release Server Trace

- NoGC
- +Rotating GC
- +GC-Tolerant Read
- +Plane-Blocking
- Base
- ttFlash
Developer Tools Release Server Trace

NoGC
+Rotating GC
+GC-Tolerant Read
+Plane-Blocking
Base

CDT (Percentile)

99.99th percentile:

Result:

- ttFlash 3x slower than NoGC
- Base 138x slower than NoGC
Evaluated on 6 windows workload traces with various characteristics

Reduced blocked I/Os (total) from 2 – 7% to 0.003 – 0.05%

99 – 99.99%: 1.0 – 2.6x slower for ttFlash and 5.6 – 138.2x for Base
Other Evaluations
Other Evaluations

- Filebench on VSSIM+ttFlash
  - *ttFlash achieves better average latency than base case*
Other Evaluations

- **Filebench on VSSIM+ttFlash**
  - *ttFlash achieves better average latency than base case*

- **Vs. Preemptive GC**
  - *ttFlash is more stable than semi-preemptive GC*
    - (If no idle time, preemptive GC will create GC backlogs, creating latency spikes)
Tradeoffs/Limitations
Tradeoffs/Limitations

- ttFlash depends on RAIN
  - 1 parity for \( N \) parallel pages/channels
  - We set \( N = 8 \), so we lose one channel out of 8 channels.
  - Average latencies are \( 1.09 - 1.33 \)x slower than NoGC, No-RAIN case
Tradeoffs/Limitations

- ttFlash depends on RAIN
  - 1 parity for N parallel pages/channels
  - We set $N = 8$, so we lose one channel out of 8 channels.
  - Average latencies are $1.09 - 1.33x$ slower than NoGC, No-RAIN case

- RAID $\rightarrow$ more writes (P/E cycles)
  - ttFlash increases P/E cycles by 15 – 18% for most of workloads
  - Incur > 53% P/E cycles for TPCC, MSN (random write)
Tradeoffs/Limitations

- **ttFlash depends on RAIN**
  - 1 parity for $N$ parallel pages/channels
  - We set $N = 8$, so we lose one channel out of 8 channels.
  - Average latencies are $1.09 - 1.33x$ slower than NoGC, No-RAIN case

- **RAID $\rightarrow$ more writes (P/E cycles)**
  - *ttFlash* increases P/E cycles by 15 – 18% for most of workloads
  - Incur > 53% P/E cycles for TPCC, MSN (random write)

- **ECC is not checked during GC**
  - Suggest background scrubbing (read is fast & not as urgent as GC)
  - Important note: in ttFlash, foreground/user reads are still ECC checked
Under write burst and at high watermark, ttFlash must dynamically disable Rotating GC to ensure there is always enough number of free pages.
Conclusion
Conclusion

CDF (Percentile)

Latency

GC-induced long tail
Conclusion

**technology:** Powerful Controller
- **RAIN** (parity-based redundancy)
- Capacitor-backed RAM

![Graph showing CDF (Percentile) vs Latency with a long tail](image)

GC-induced long tail
Conclusion

New techniques:

I. Plane-Blocking GC
II. GC-Tolerant Read
III. Rotating GC
IV. GC-Tolerant Flush

technology: Powerful Controller
RAIN (parity-based redundancy)
Capacitor-backed RAM
Conclusion

New techniques:

I. Plane-Blocking GC
II. GC-Tolerant Read
III. Rotating GC
IV. GC-Tolerant Flush

technology: Powerful Controller
RAIN (parity-based redundancy)
Capacitor-backed RAM

Overall results achieved:

Between 99 - 99.99<sup>th</sup> percentiles:

**ttFlash** 1-3x slower than NoGC
**Base** 5-138x slower than NoGC
Thank you!

Questions?

http://ucare.cs.uchicago.edu

https://ceres.uchicago.edu