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

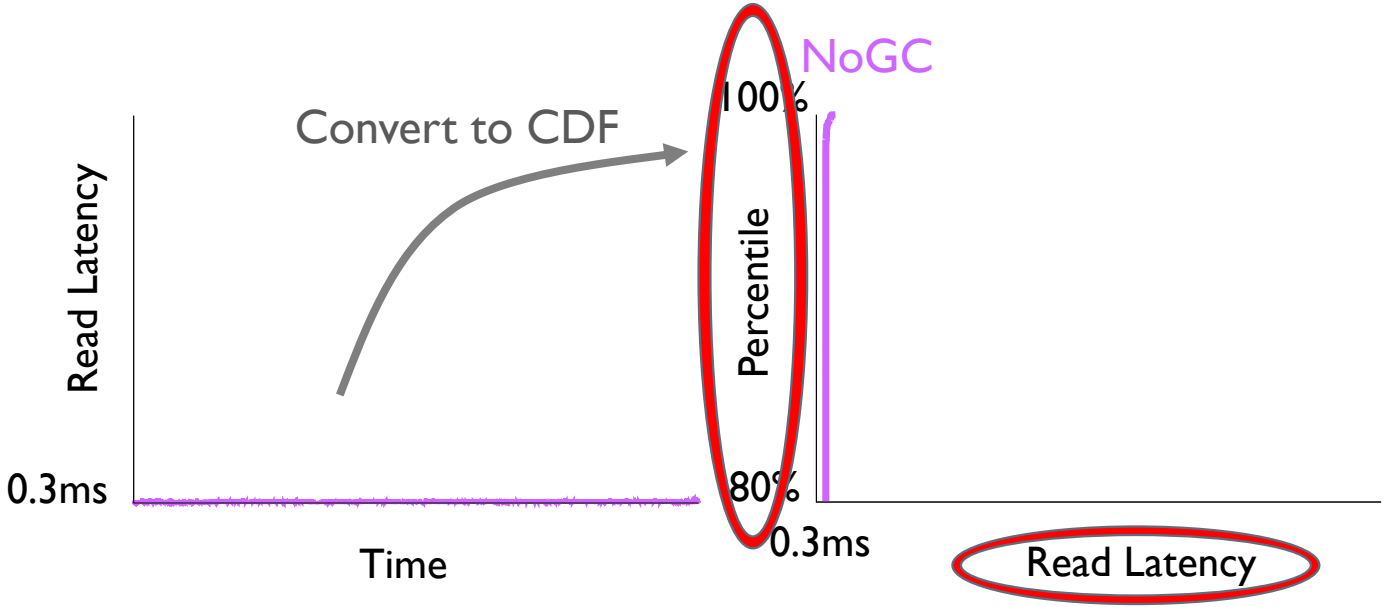
*"if your read is stuck behind an erase you may have wait **10s of milliseconds**. That's a **100x** increase in latency variance"*

Why it's hard to meet SLAs with SSDs

http://www.zdnet.com/article/why-ssds-dont-perform/
*The Tail at Scale **[CACM'13]***
https://storagemojo.com/2015/06/03/why-its-hard-to-meet-slas-with-ssds/

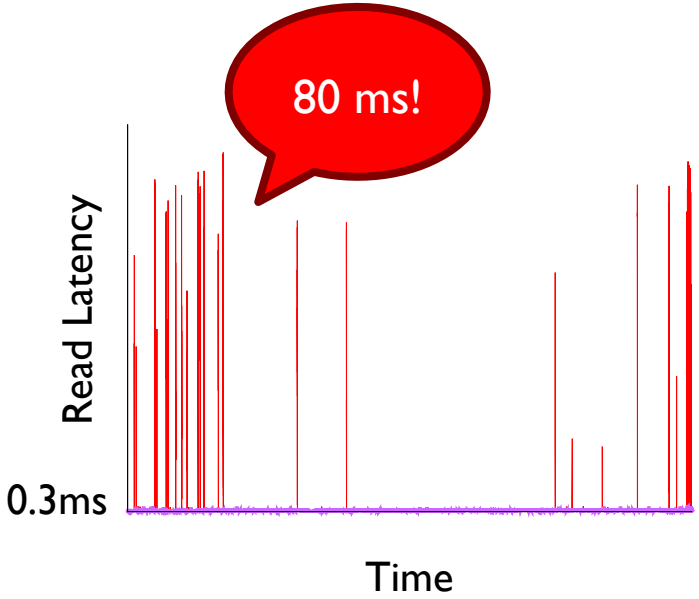
Reads + Writes

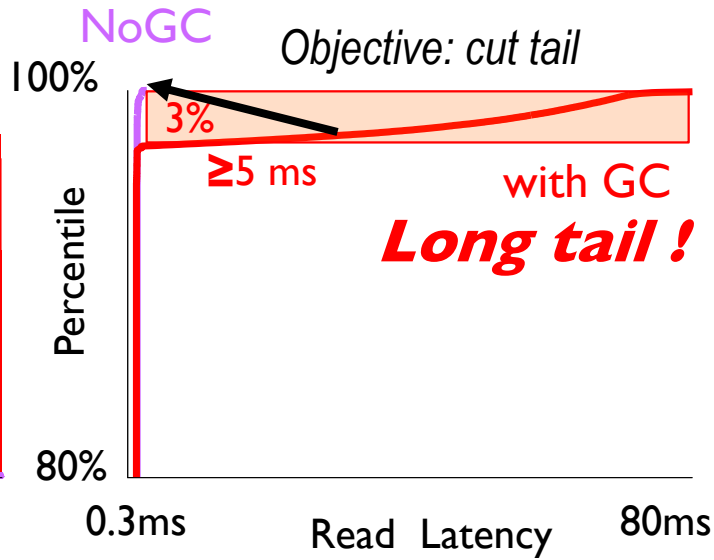
Clean/Empty SSD



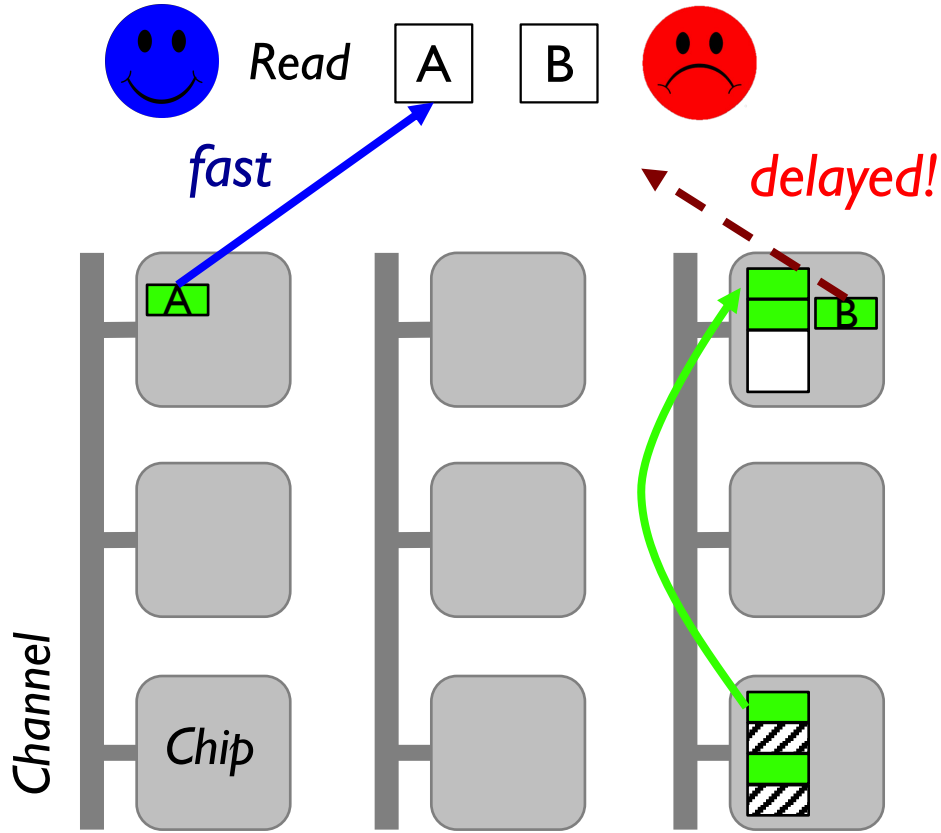
Reads + Writes

Aged/Full SSD





How GC delays read I/Os?



A GC moves tens of valid pages!

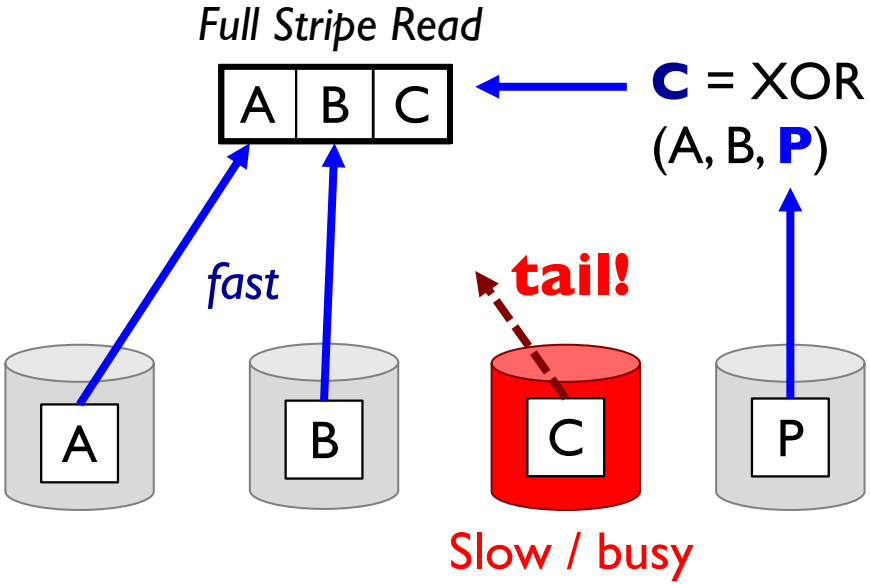
which makes channel/chips busy for **tens of ms !**

How to cut tail latencies?

Tail-tolerant techniques in distributed/storage systems:

Leverage redundancy to cut tail!

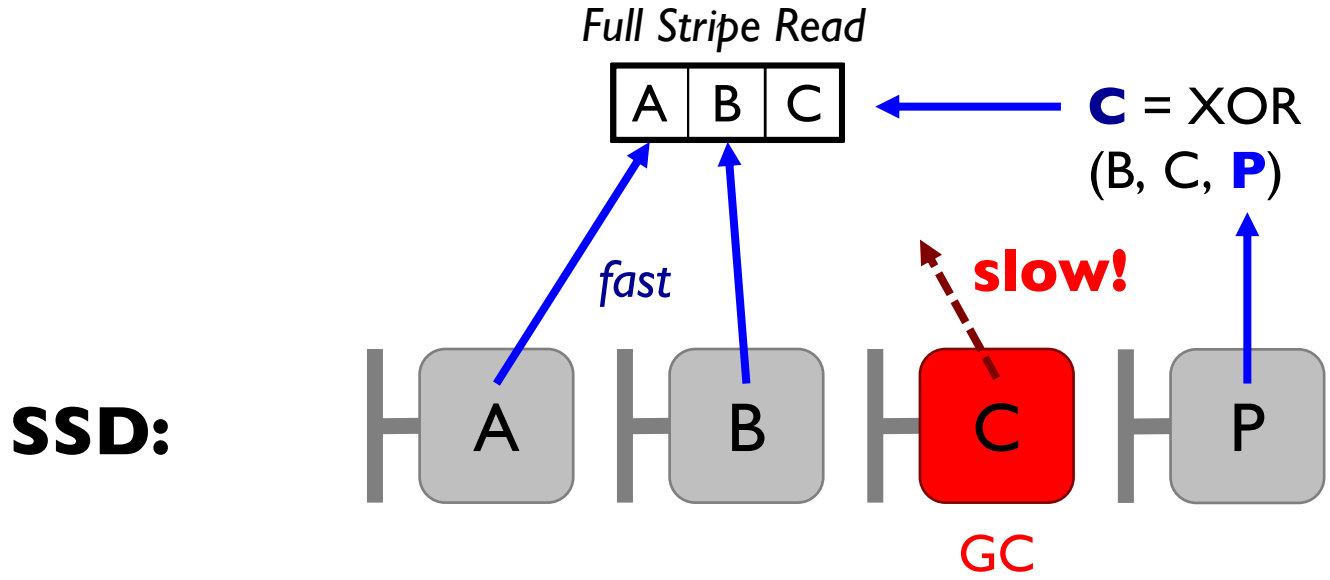
RAID:



How to cut tails in SSD?

Error rate increases → **RAIN** (**R**edundant **A**rray of **I**ndependent **N**AND)

Similarly, we leverage RAIN to cut "tails"!



Contribution

New techniques:

I. Plane-Blocking GC

II. GC-Tolerant Read

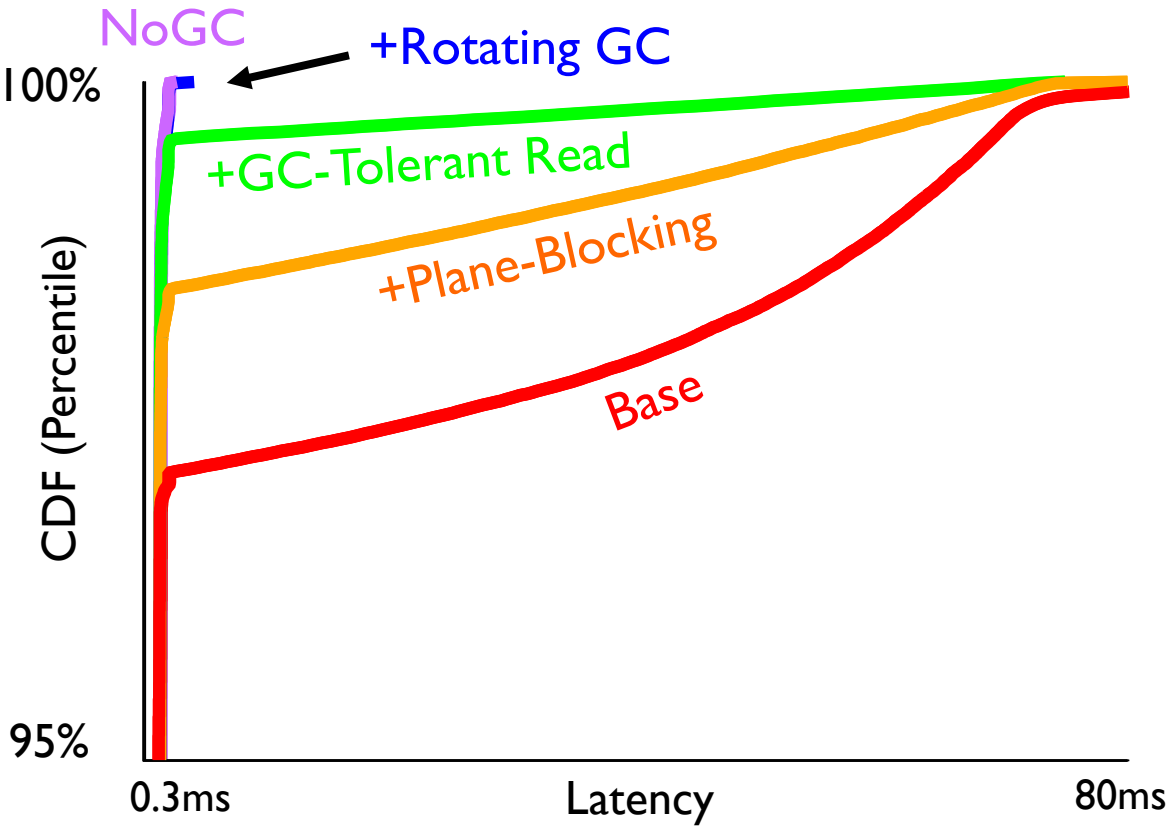
III. Rotating GC

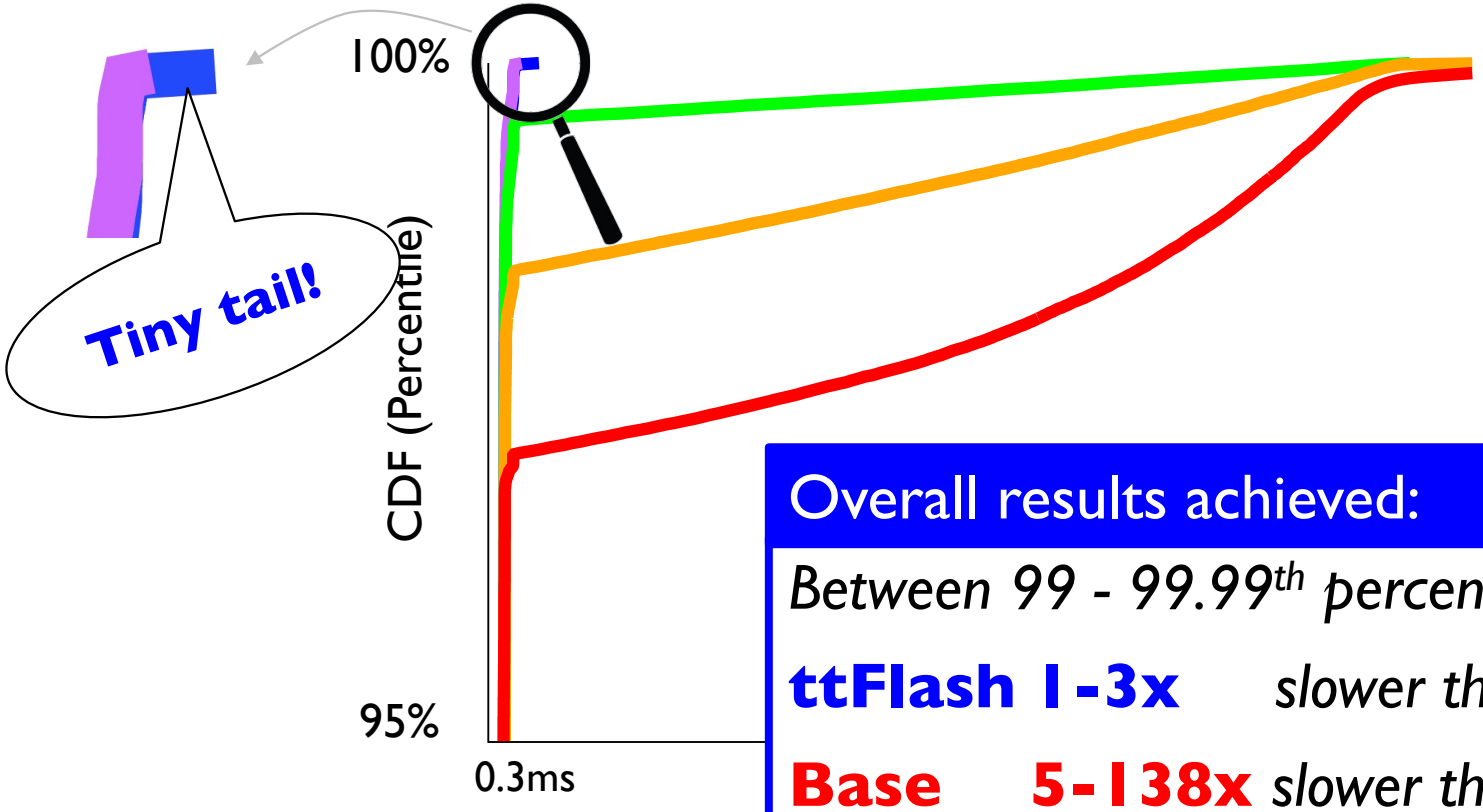
IV. GC-Tolerant Flush

Current SSD technology:

RAIN
(Parity-based Redundancy)

Results


NoGC
+Rotating GC
100%
+GC-Tolerant Read
+Plane-Blocking
Base
CDF (Percentile)
95%
0.3ms
Latency
80ms



Overall results achieved:

Between 99 - 99.99th percentiles:

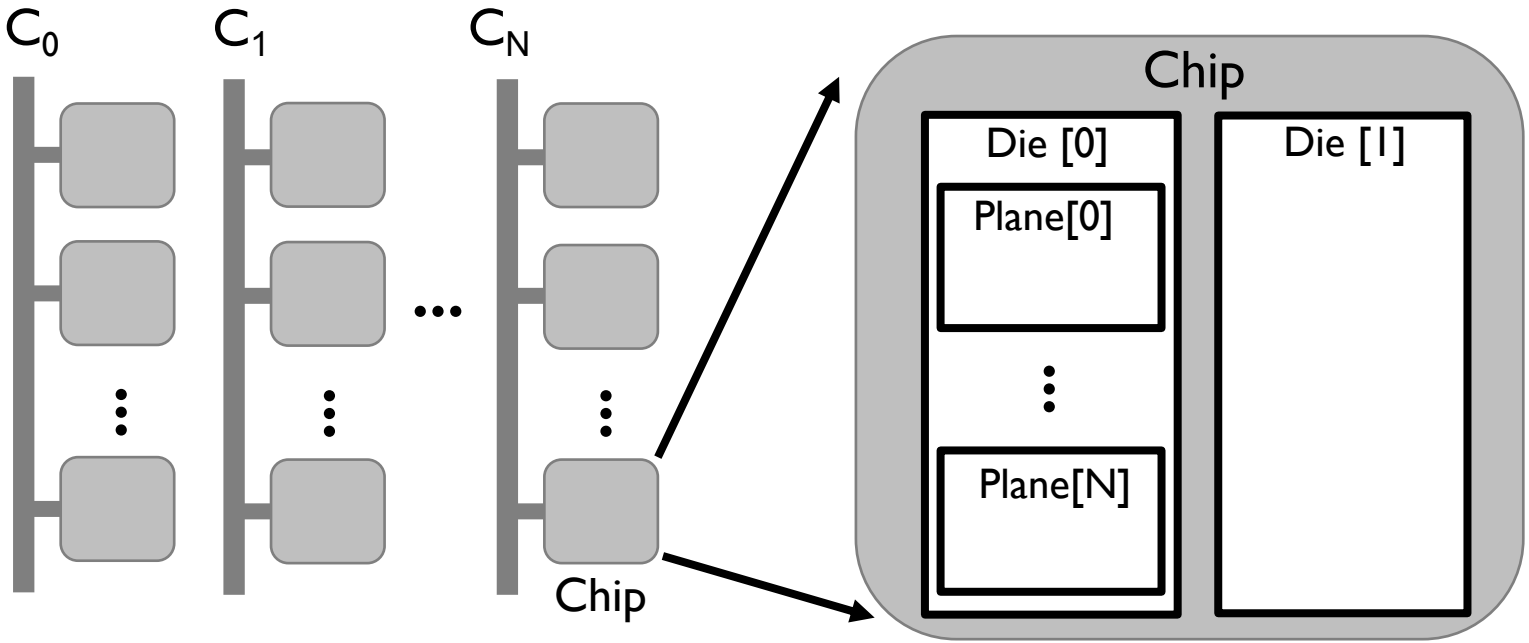
ttFlash 1-3x *slower than NoGC*

Base 5-138x *slower than NoGC*

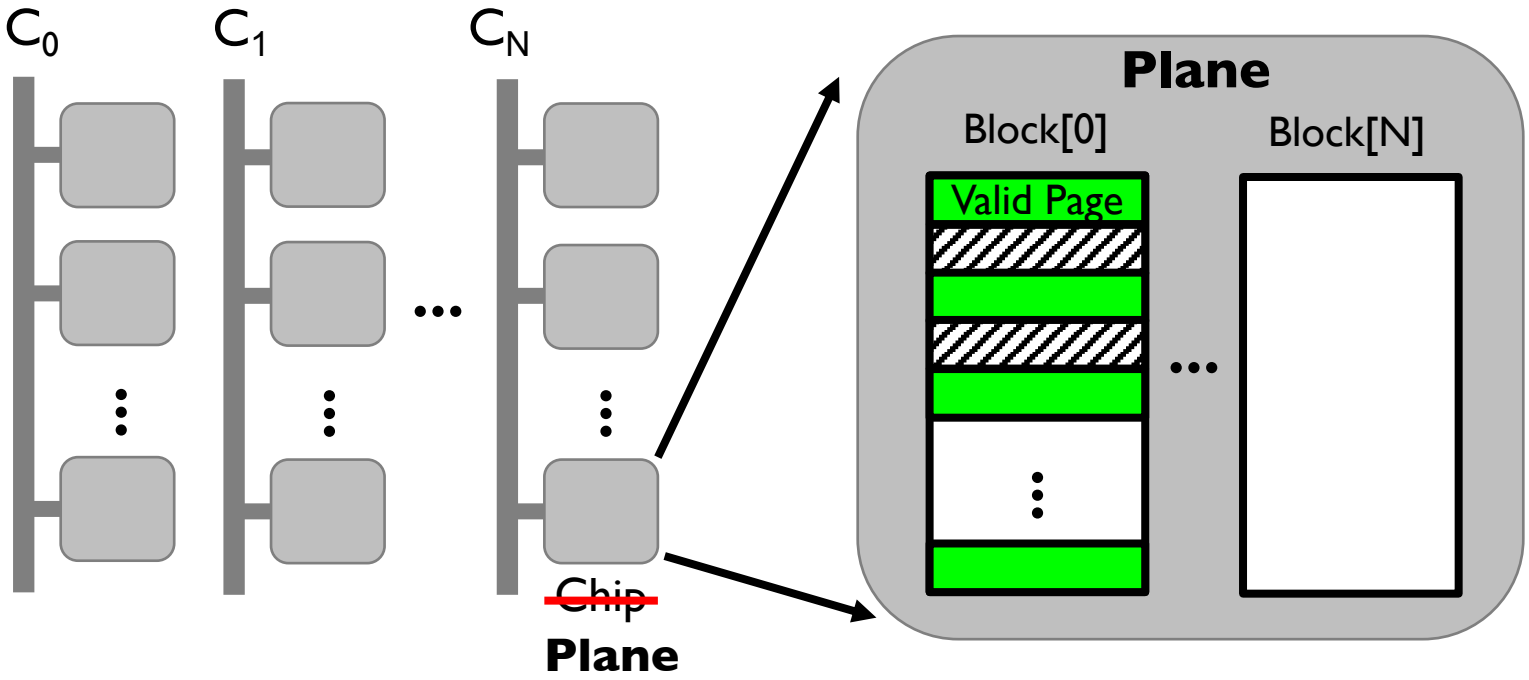
Outline

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

SSD Internals

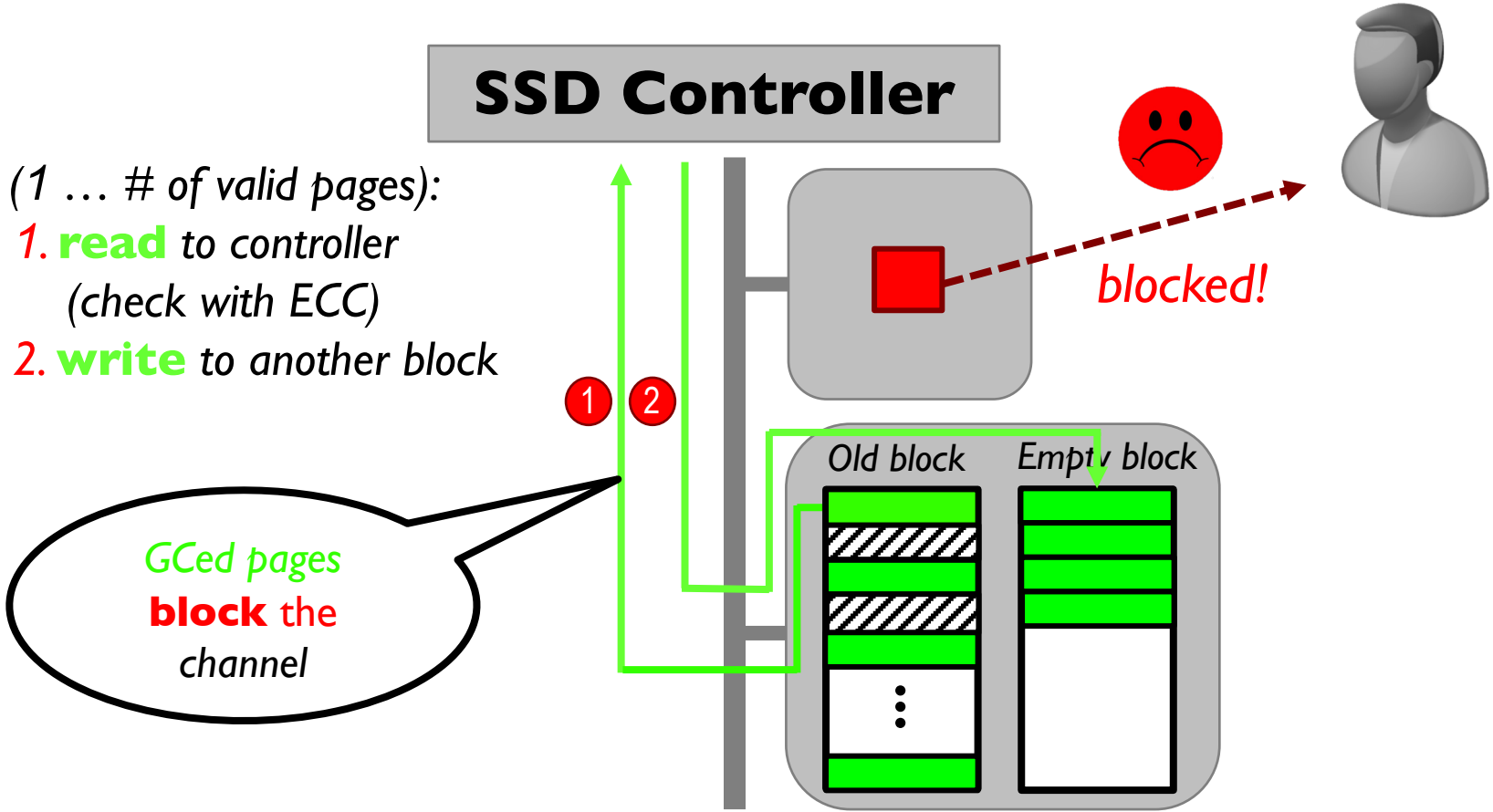


SSD Internals

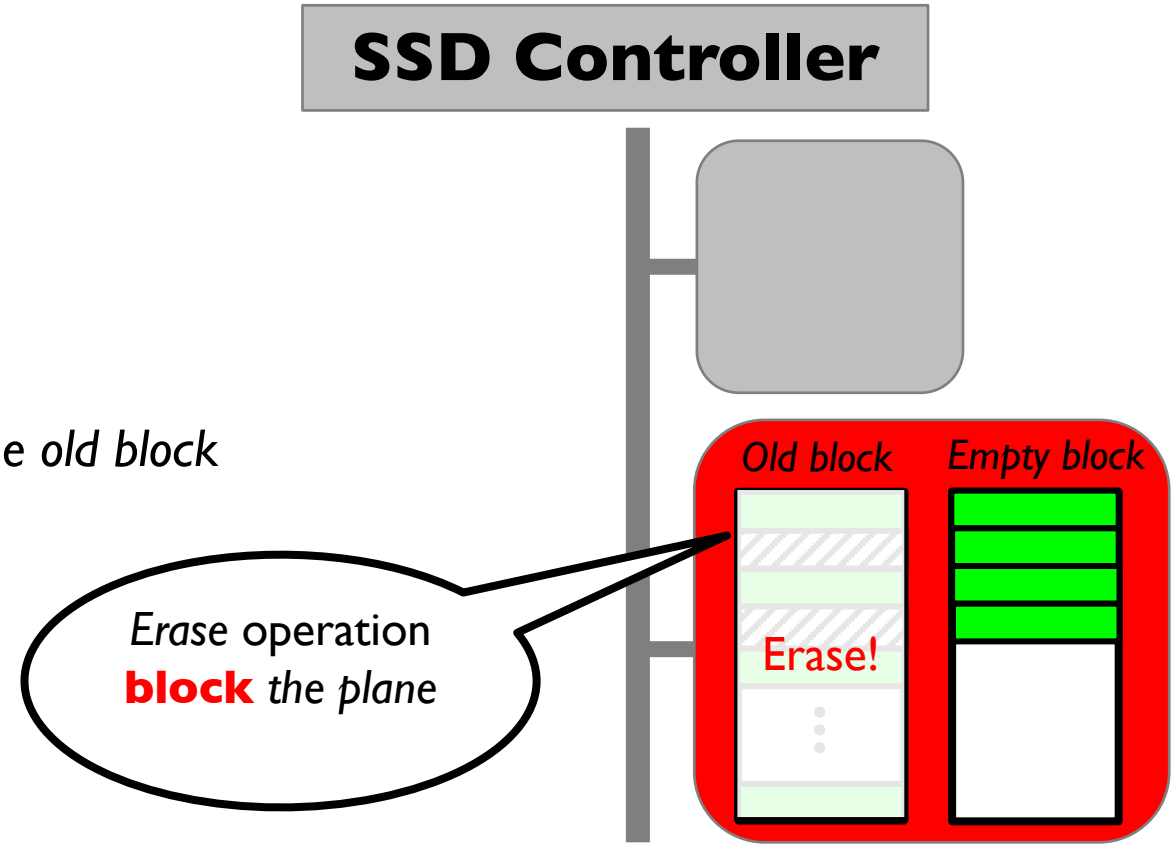


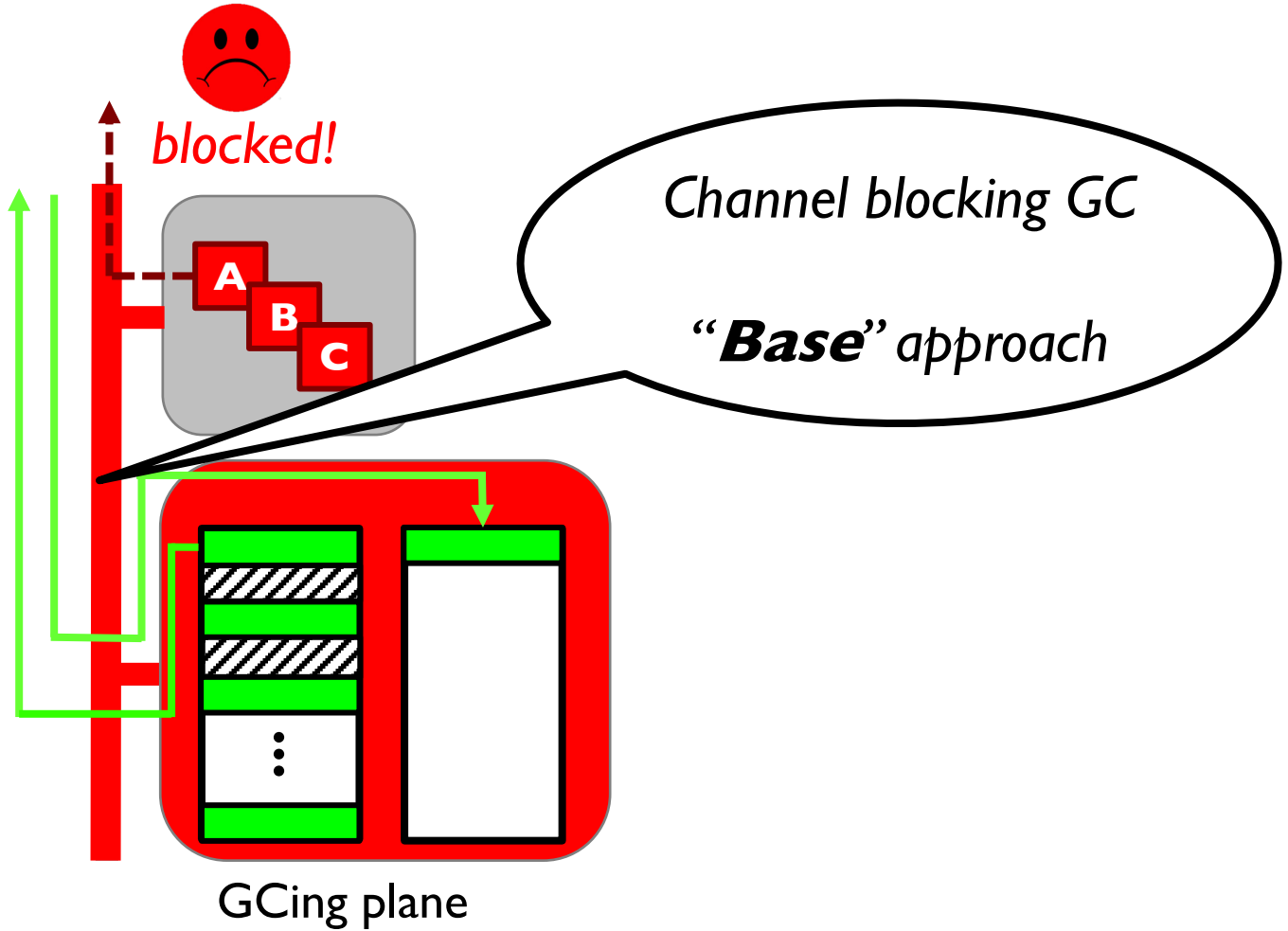
for (1 … # of valid pages):

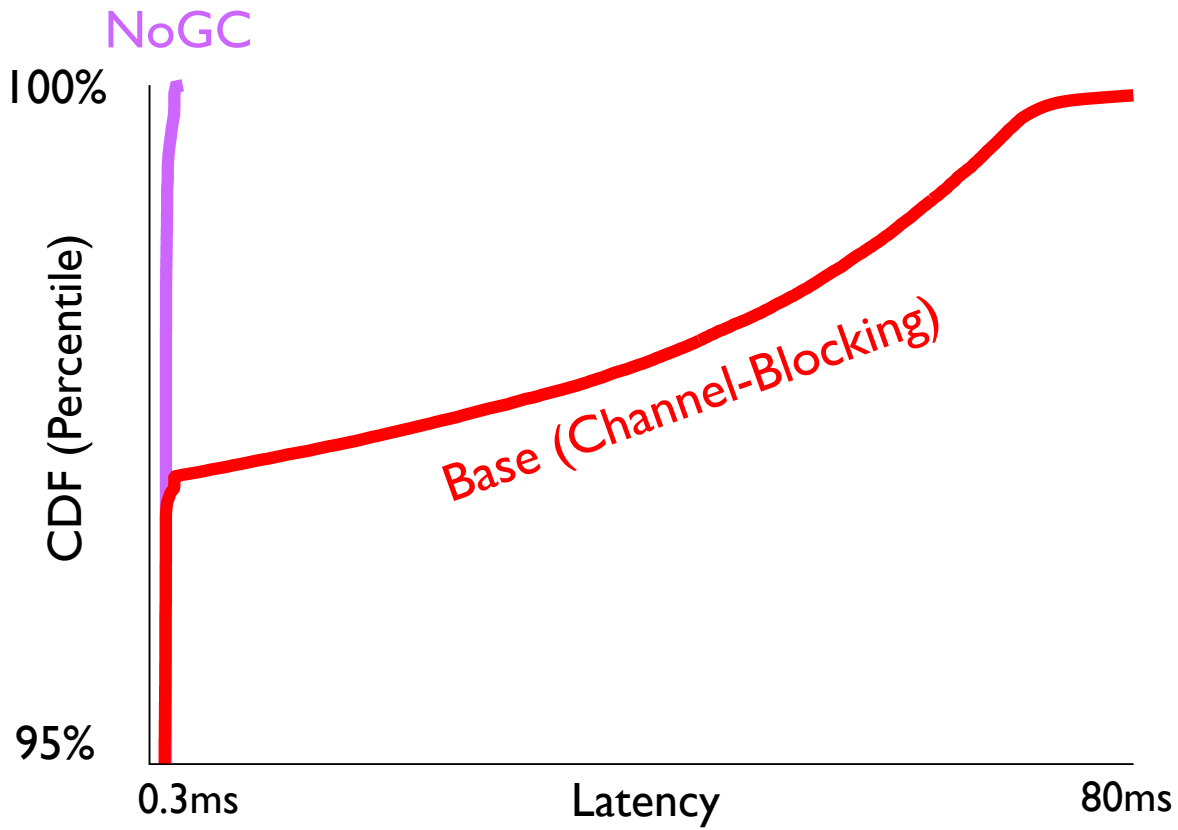
1. **read** *to controller (check with ECC)*
2. **write** *to another block*



3. *Erase* the old block

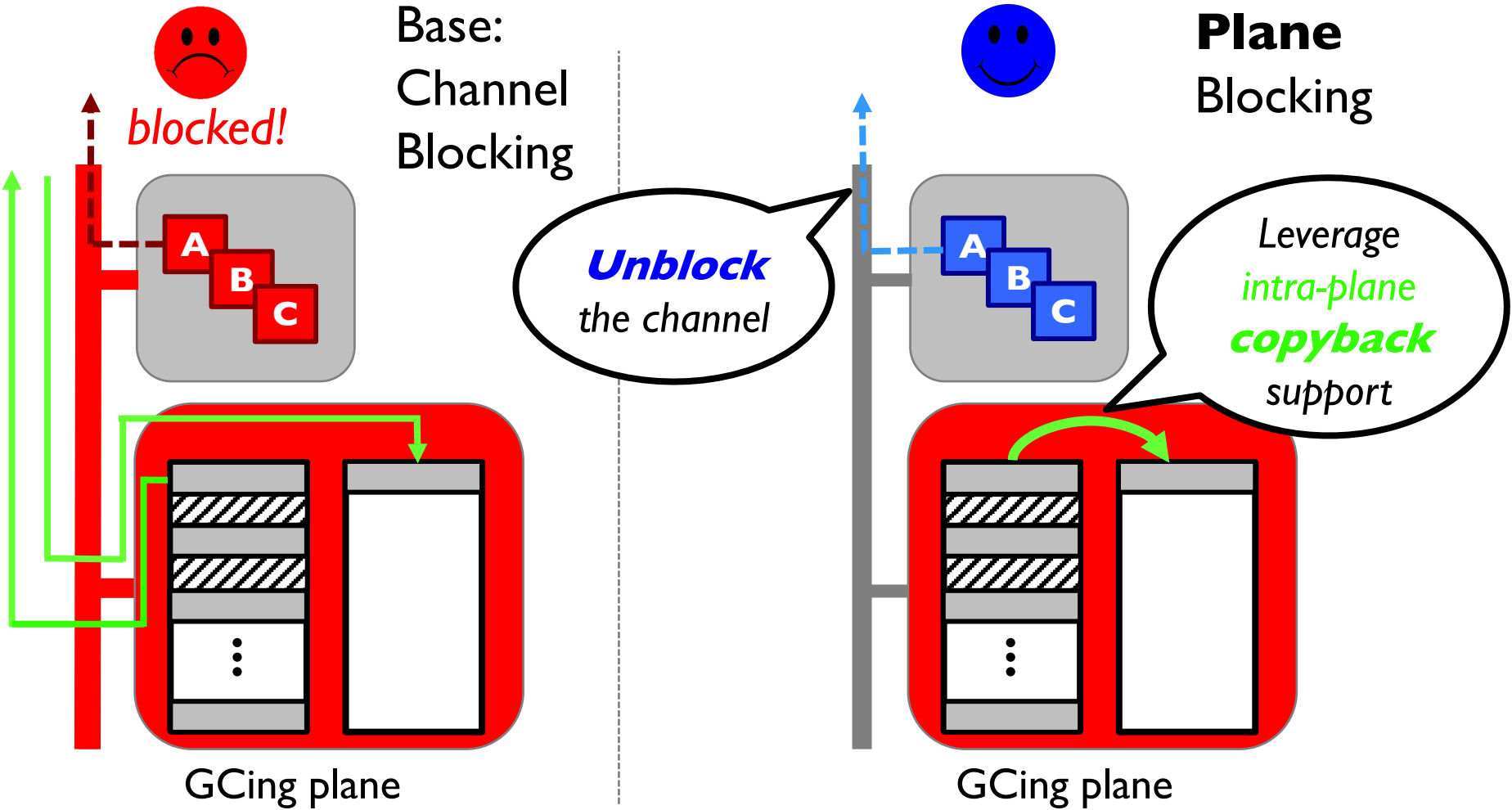



blocked!
A
B
C
Channel blocking GC
"Base" approach
GCing plane


NoGC
100%
95%
CDF (Percentile)
Base (Channel-Blocking)
0.3ms
Latency
80ms

Outline

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

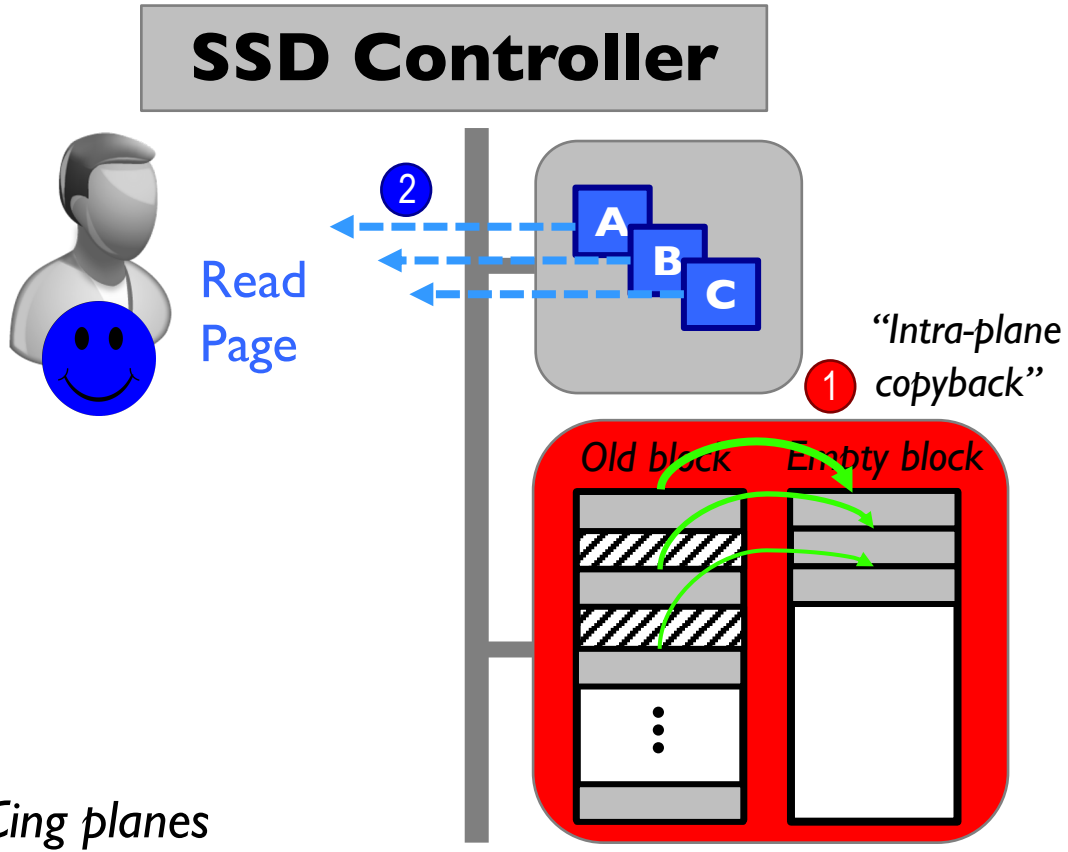

Base:
Channel
Blocking
blocked!
A
B
C
GCing plane
Unblock
the channel
Plane
Blocking
A
B
C
Leverage
intra-plane
copyback
support
GCing plane

Plane Blocking

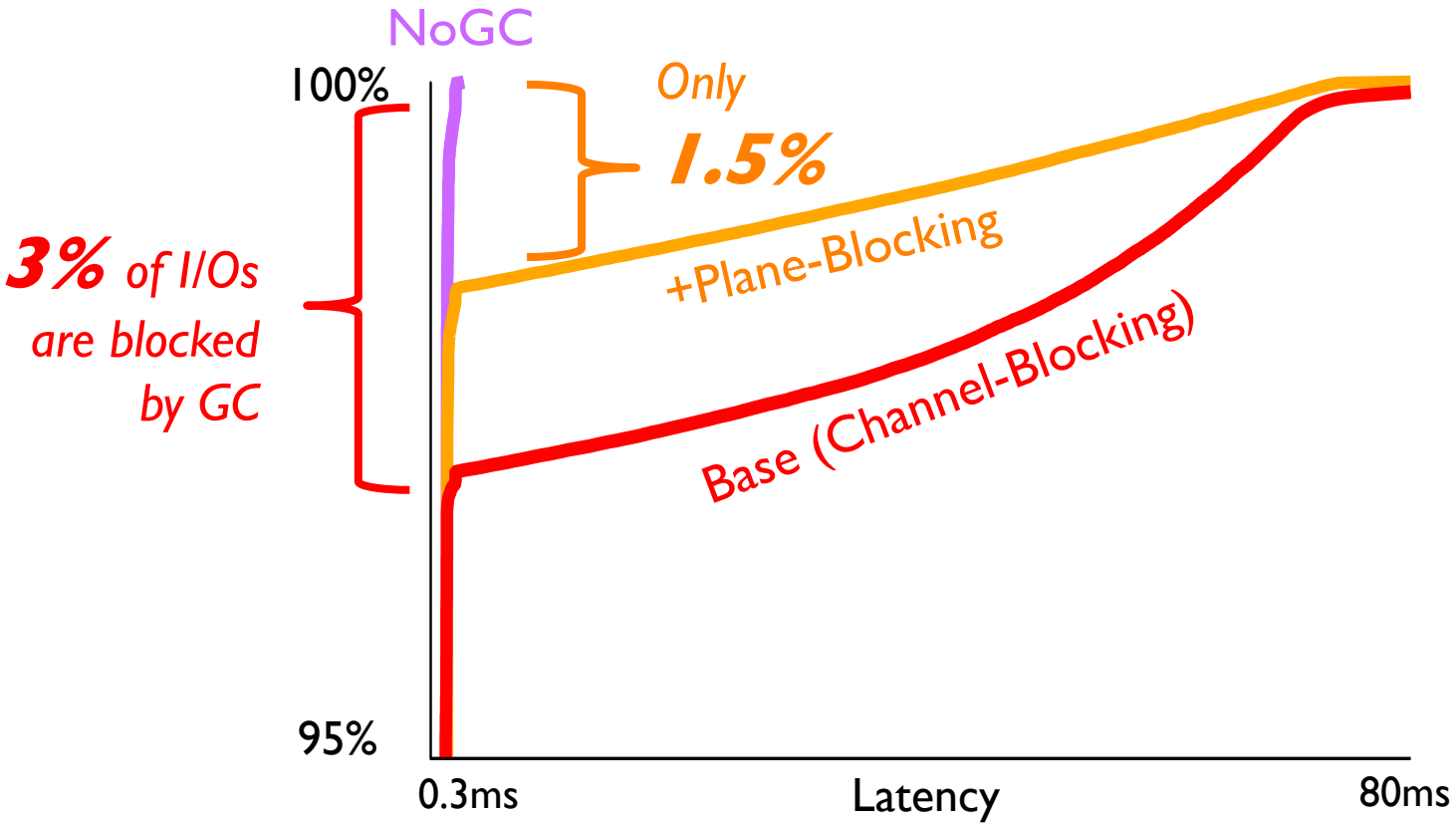
Base GC Logic:

for (every valid page)
1. flash r
(over c
2. wait

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

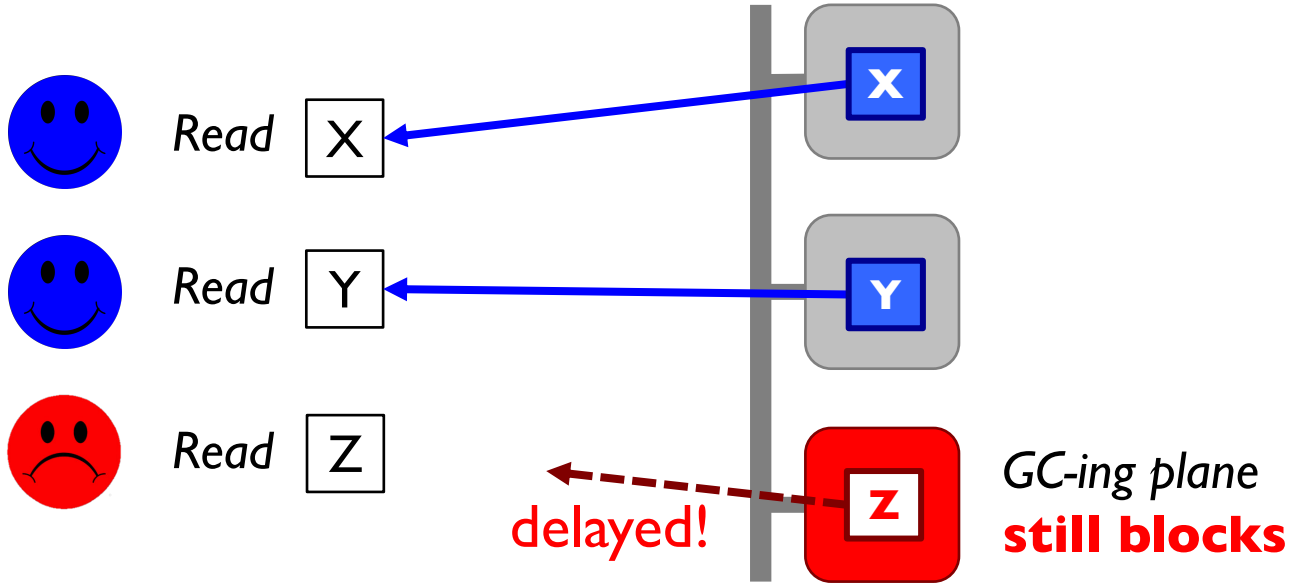


Overlap
1. *intra-plane copyback* **with**
2. *channel usage for other non-GCing planes*


NoGC
100%
Only
1.5%
+Plane-Blocking
3% of I/Os
are blocked
by GC
Base (Channel-Blocking)
95%
0.3ms
Latency
80ms

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

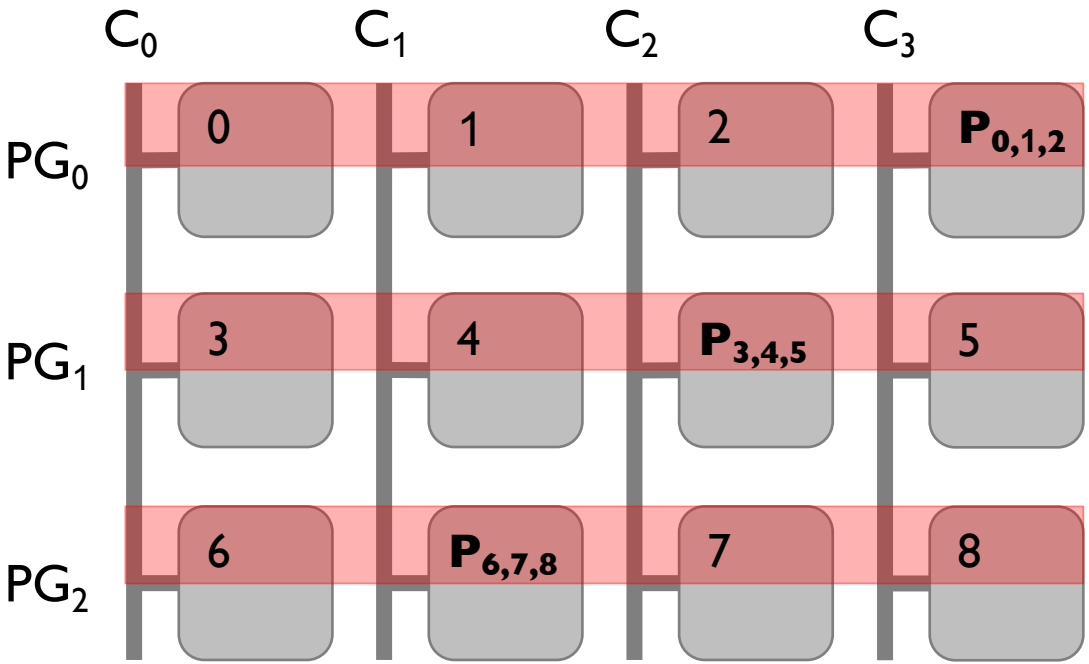
- **Issue 2:**



Outline

- Introduction
- Background
- **Tiny-Tail Flash Design**
  - Plane-Blocking GC
  - **RAIN + GC-Tolerant Read**
  - Rotating GC
  - GC-Tolerant Flush
- Evaluation, limitations, conclusion

RAIN



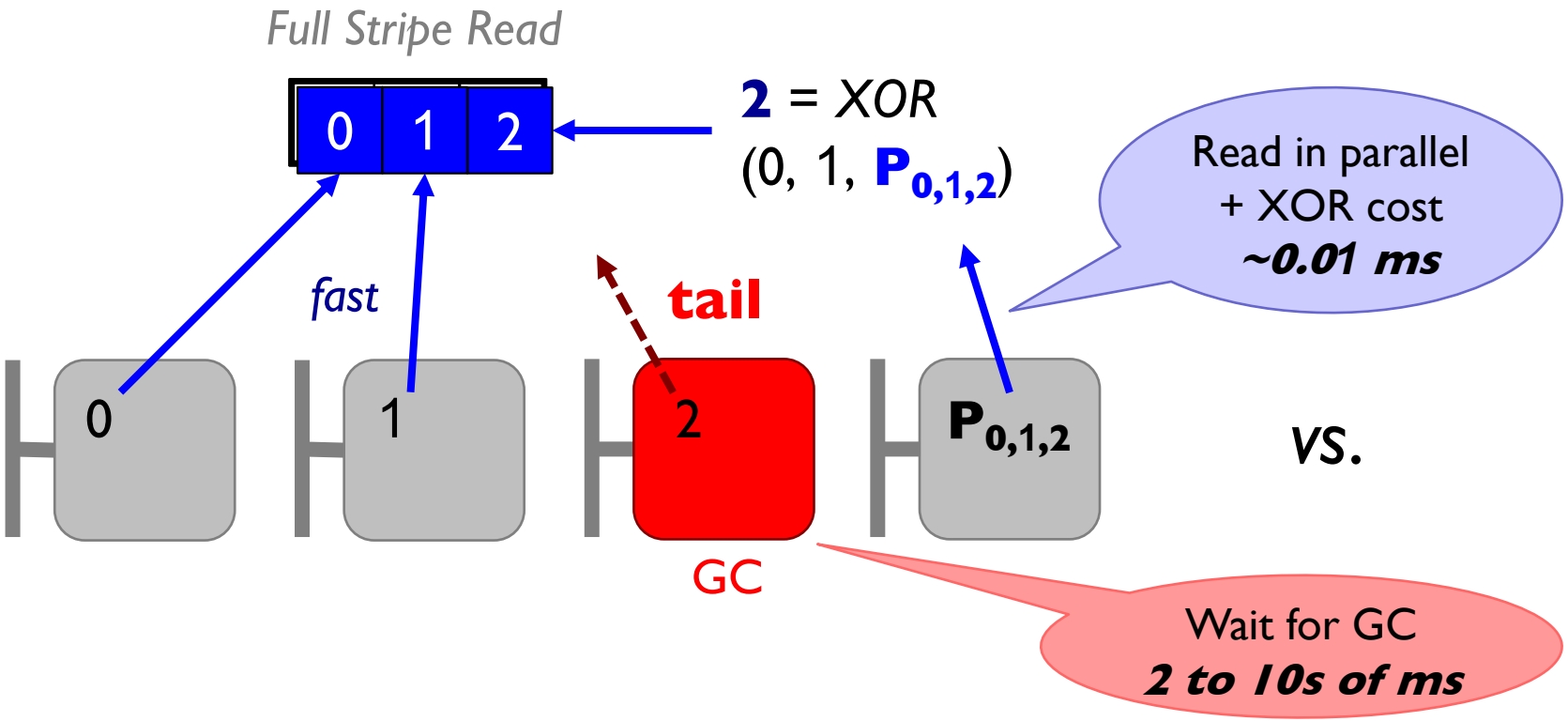
LPN (Logical Page #)

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

Add parity:
LPN 0, 1, 2 → **$P_{0,1,2}$**

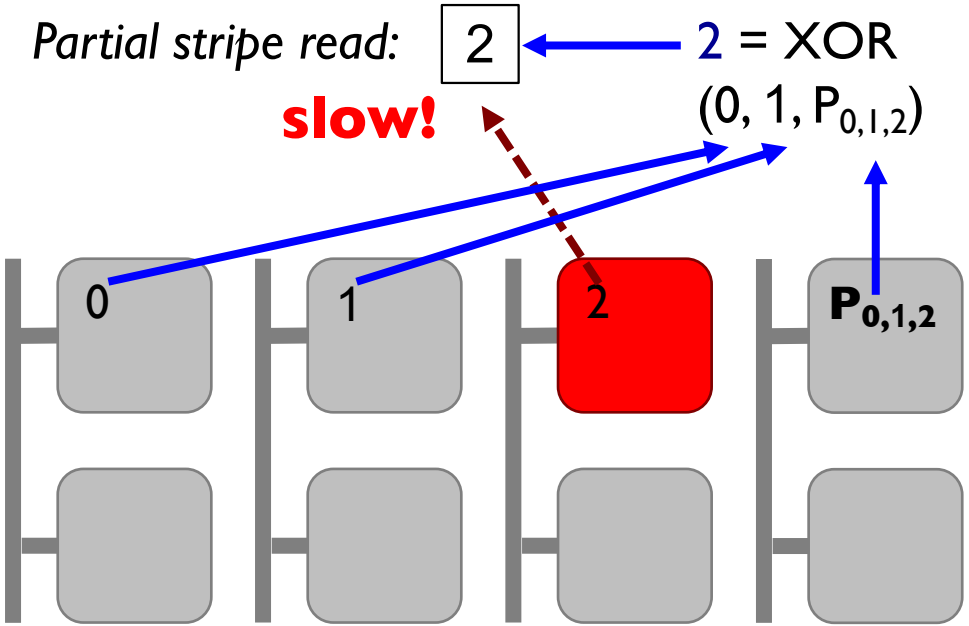
Rotating parity as RAID 5

RAIN enables **GC-Tolerant Read**



GC-Tolerant Read

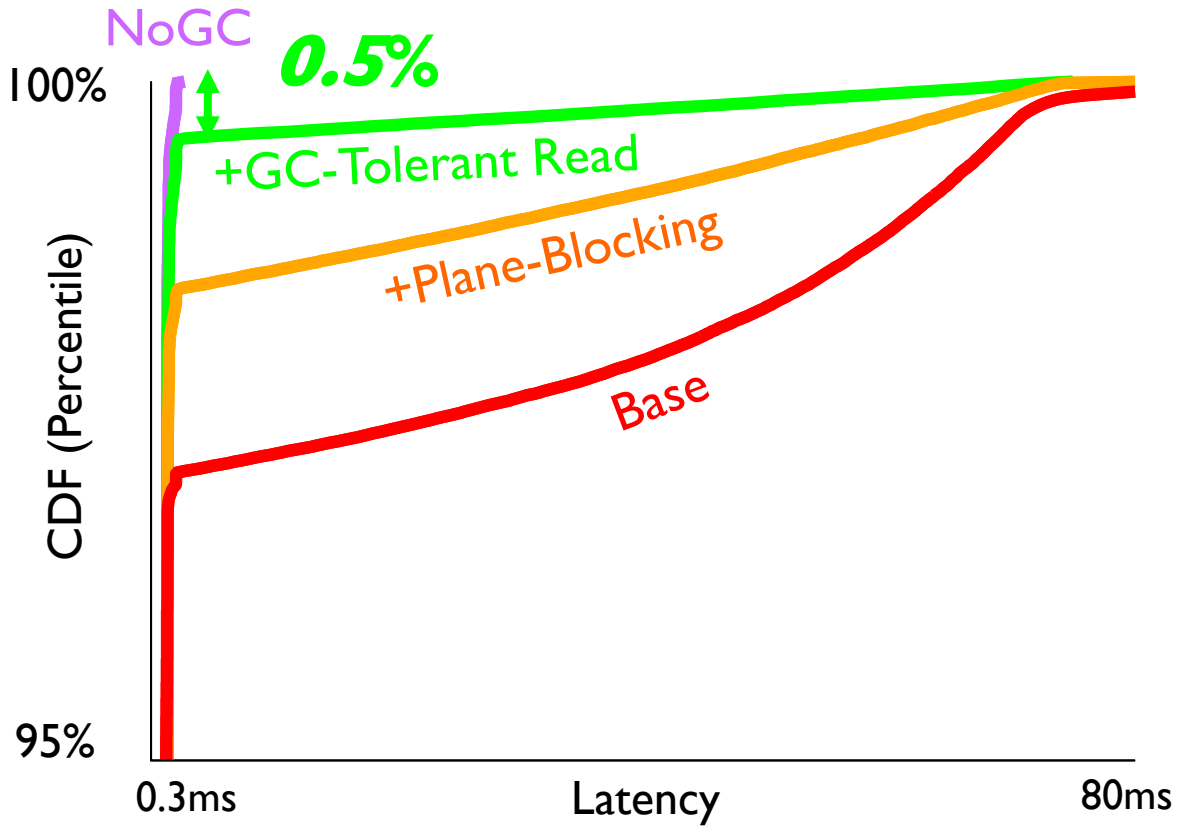
*Issue: **partial** stripe read*



Must generate extra
N-1 reads!

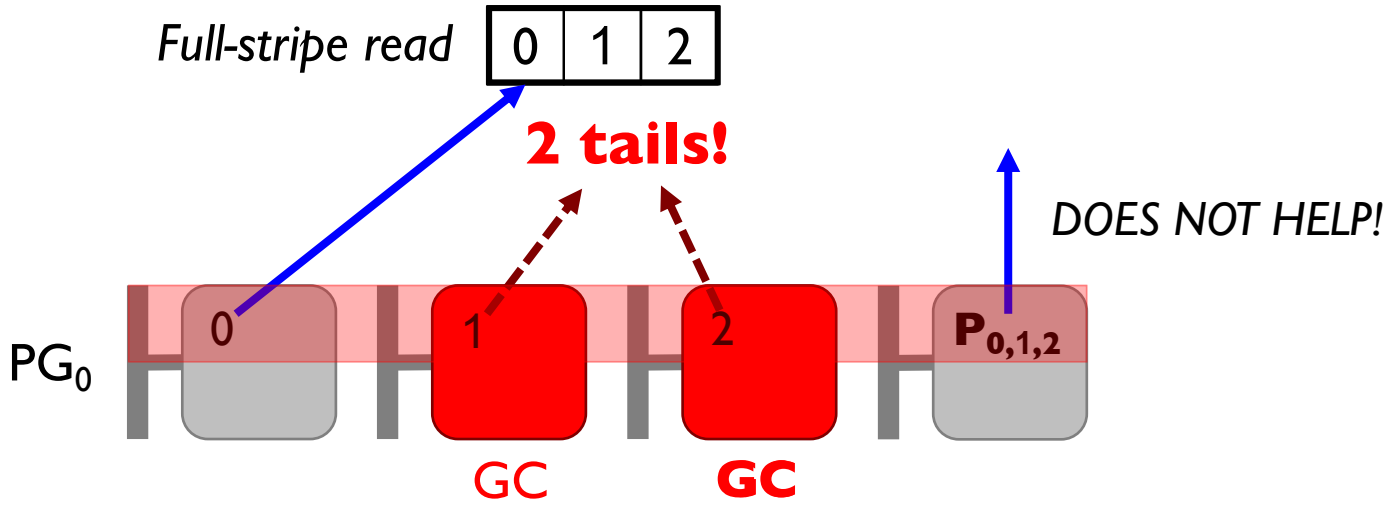
*Add **contention** to other*
N -1 channels and planes

Convert to full stripe if:
$T_{extra\text{-}reads} < T_{GC}$


NoGC
0.5%
100%
+GC-Tolerant Read
+Plane-Blocking
Base
CDF (Percentile)
95%
0.3ms
Latency
80ms

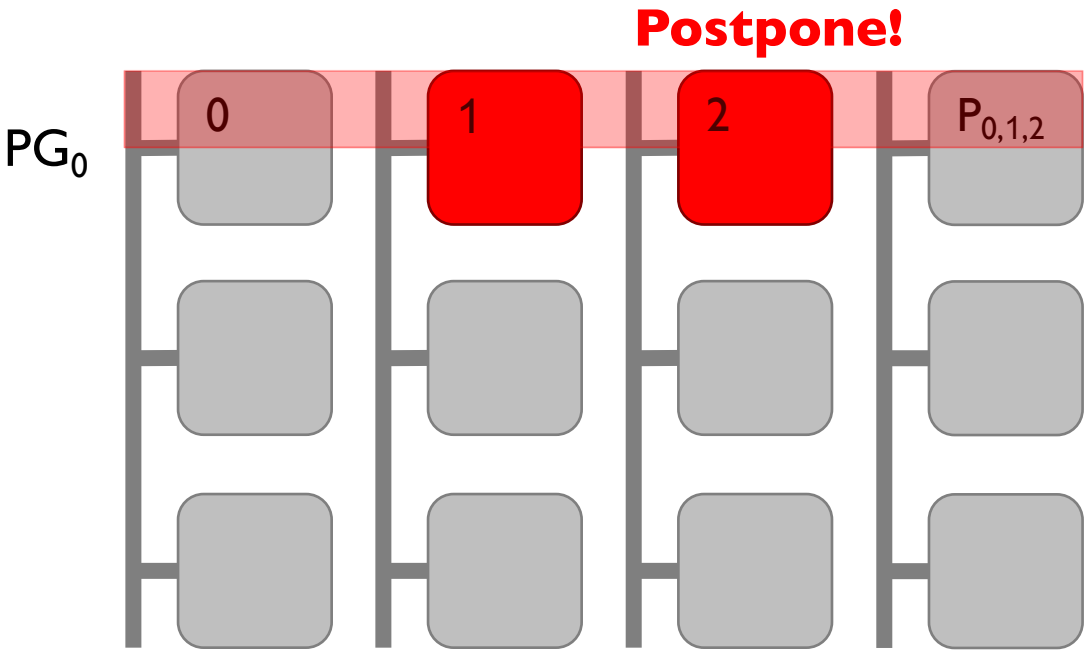
Issue: **more than 1 GCs** in a plane group?

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



Outline

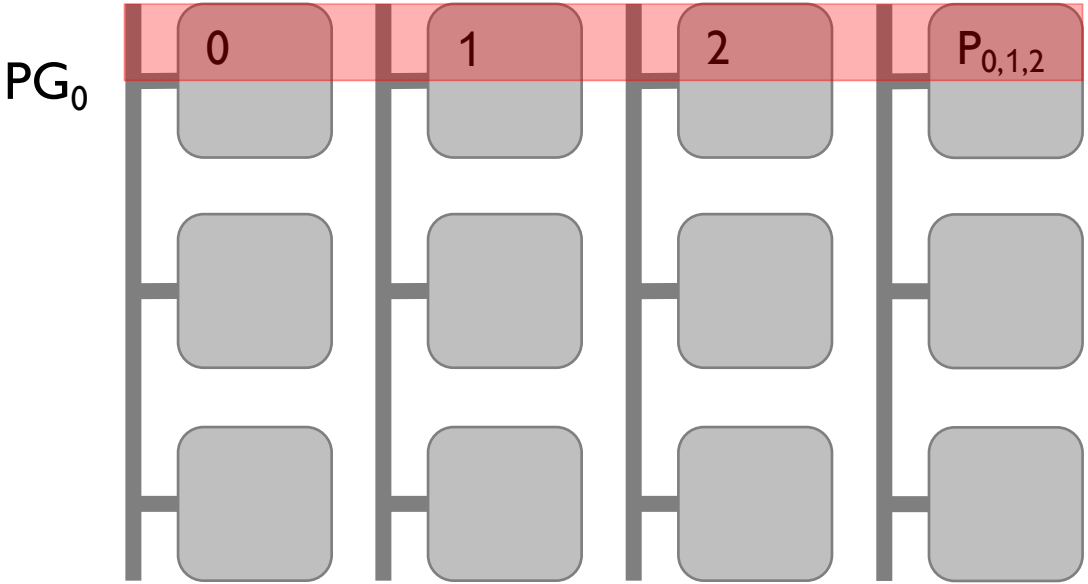
- Introduction
- Background
- **Tiny-Tail Flash Design**
  - Plane-Blocking GC
  - GC-Tolerant Read
  - **Rotating GC**
  - GC-Tolerant Flush
- Evaluation, limitations, conclusion



Rotating GC:

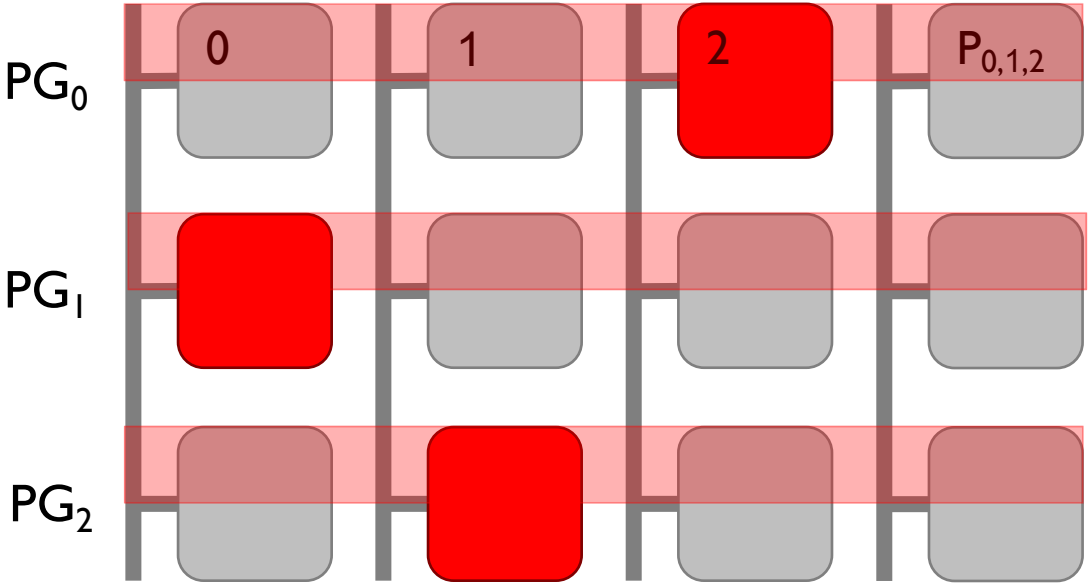
Anytime, **at most 1** *plane per plane group can perform GC*

Rotating!



Rotating GC:

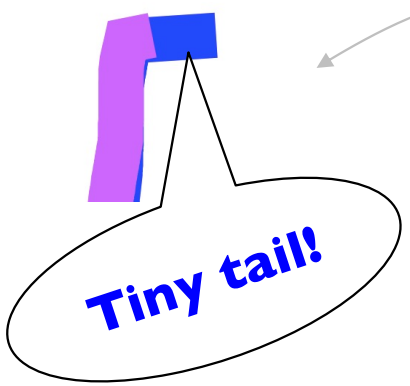
Anytime, **at most 1** *plane per plane group can perform GC*



Rotating GC:

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

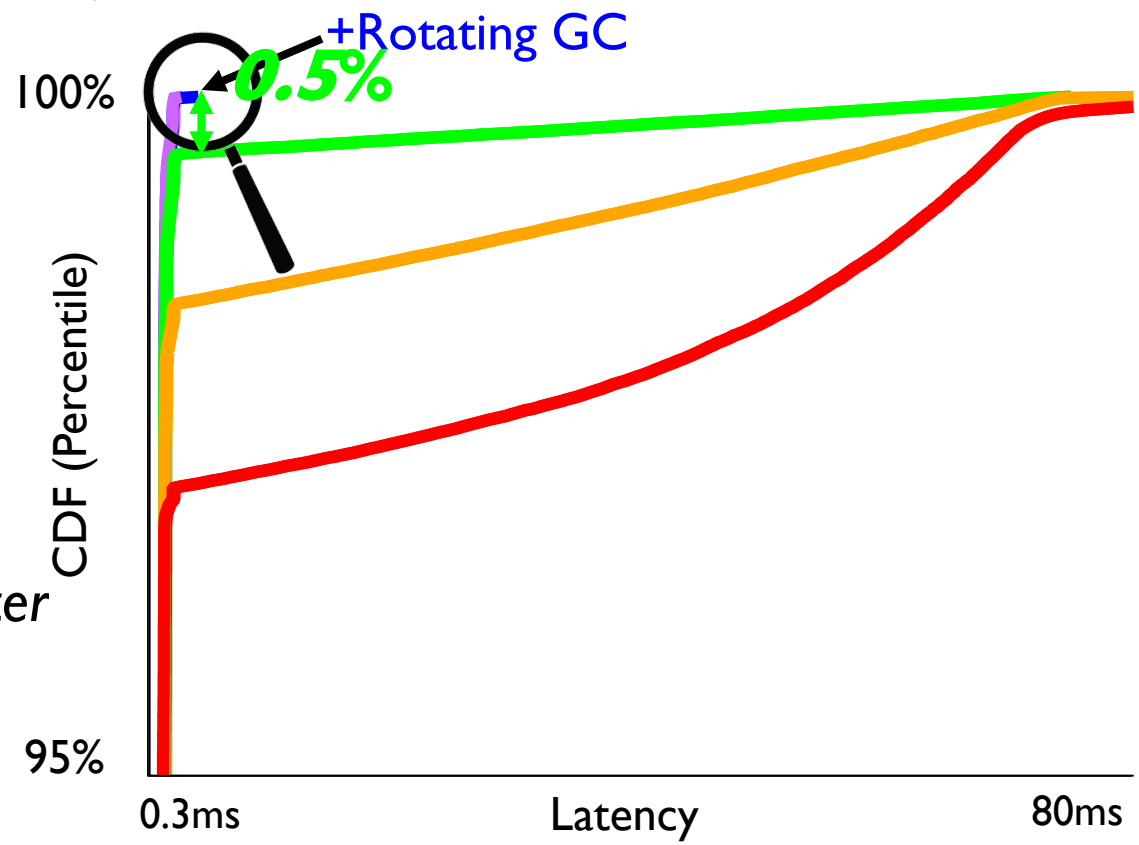
Concurrent GCs in **different** *PGs are permitted.*



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

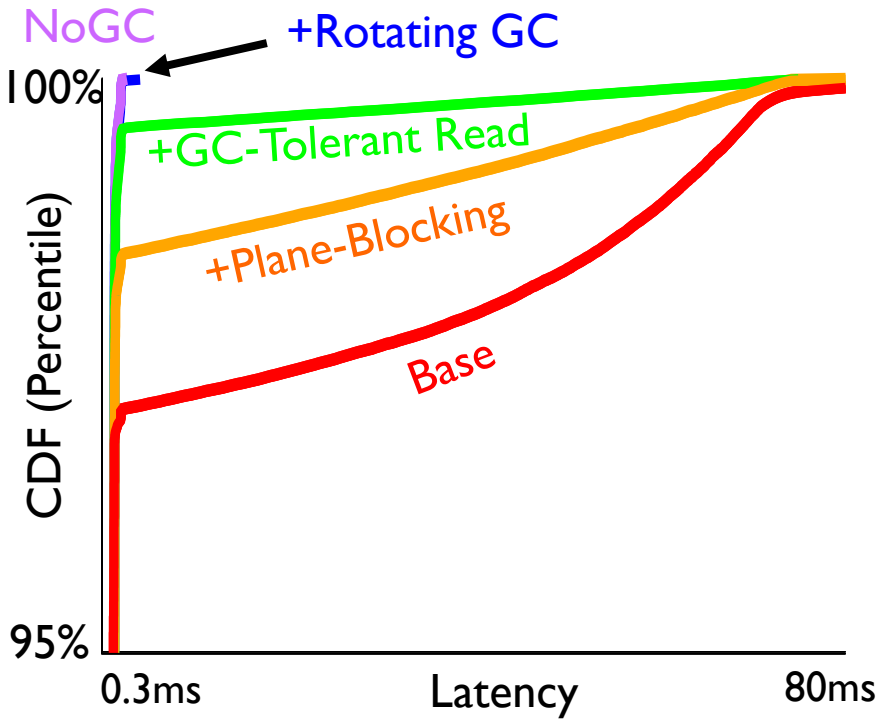
- **SSDsim** (*~2500 LOC*)
  - *Device simulator*
- **VSSIM** (*~900 LOC*)
  - *QEMU/KVM-based*
  - *Run Linux and applications*
- **OpenSSD**
  - *Many limitations of the simple programming model*
- Future: ttFlash on **OpenChannel SSD**

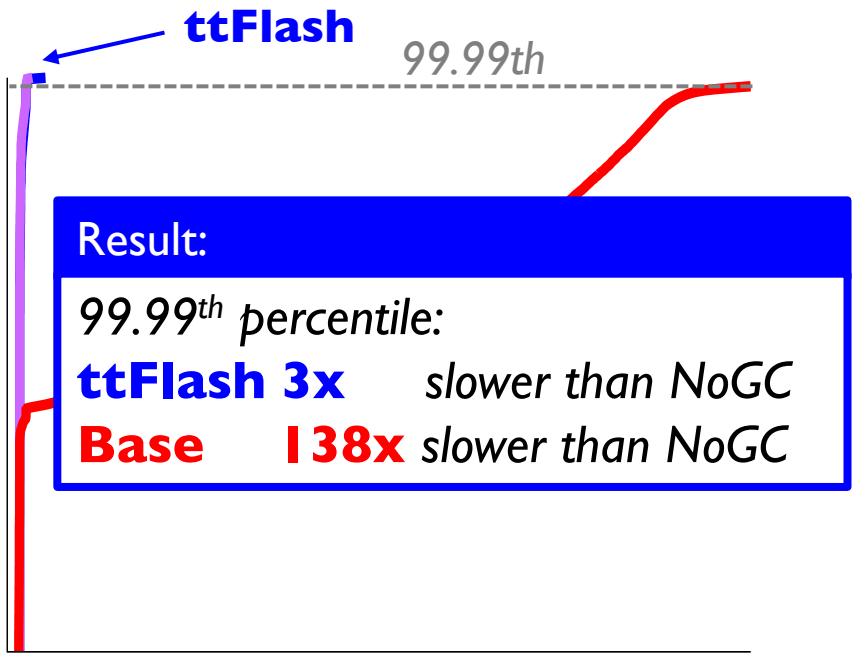
Evaluation

- 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)

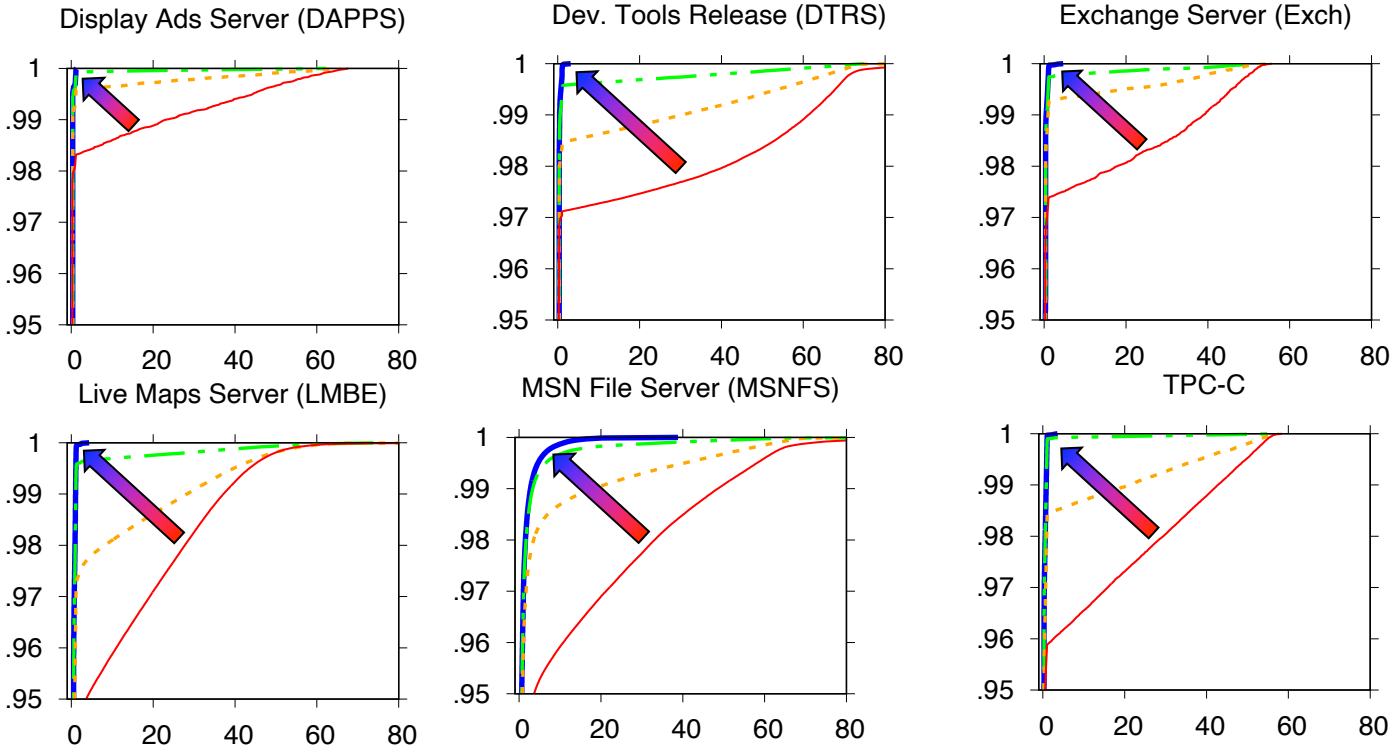
Sizes		Latencies	
SSD Capacity	256 GB	Page Read	40μs
#Channels	8	(flash-to-register)	
#Planes/channel	8	Page Write	800μs
Plane size	4 GB	(register-to-flash)	
#Planes/chip	** 1	Page data transfer	100μs
#Blocks/plane	4096	(via channel)	
#Pages/block	256	Block erase	2 ms
Page size	4 KB		

Developer Tools Release Server Trace





Evaluated on 6 windows workload traces with various characteristics

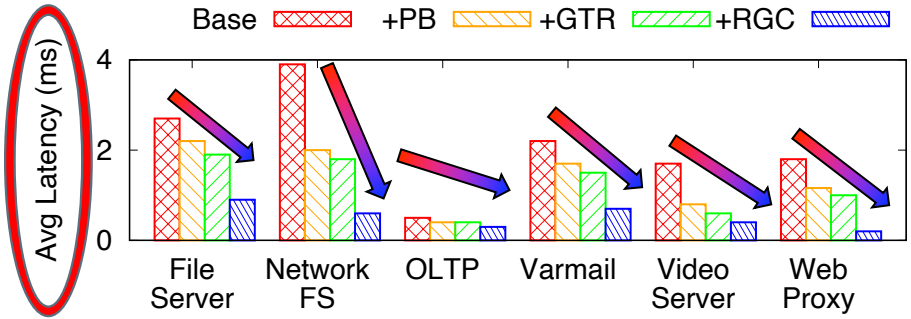


Reduced blocked I/Os (total) from 2 – 7% *to* 0.003 – 0.05%

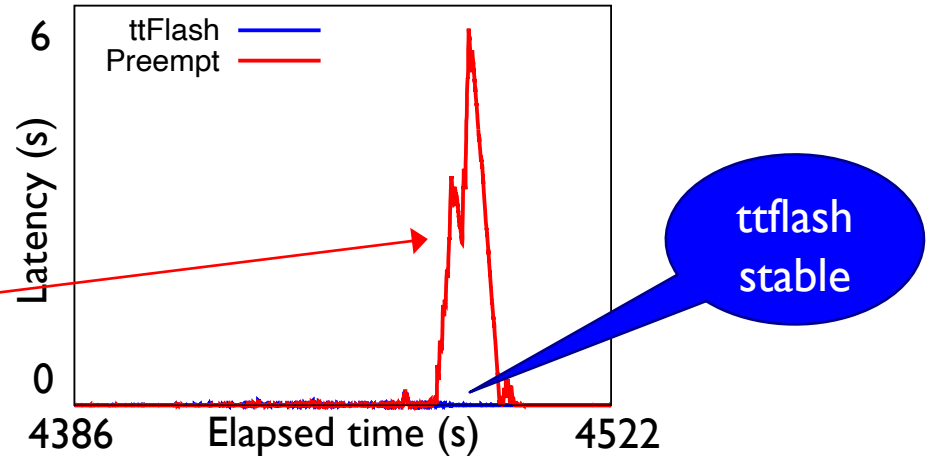
99 – 99.99%: 1.0 – 2.6x slower *for* ttFlash *and* 5.6 – 138.2x *for* Base

Other Evaluations

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



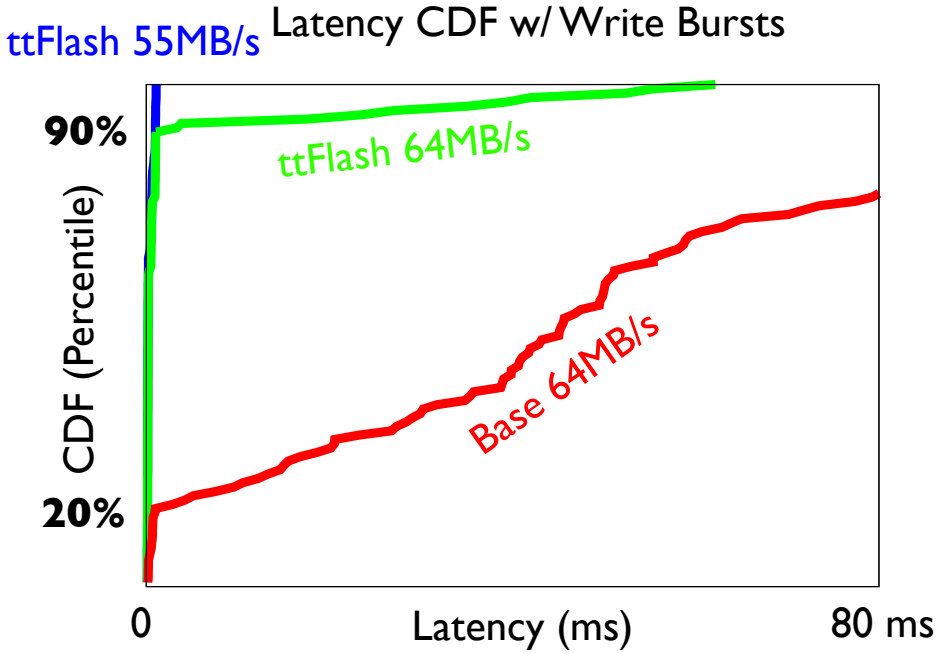
- Vs. Preemptive GC
  - *ttFlash is more* stable *than semi-preemptive GC*
    - *(If no idle time, preemptive GC will create GC backlogs, creating latency spikes)*



Tradeoffs/Limitations

- 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 → 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*

Tails under Write Bursts



Under **write burst** *and* **at high watermark**, *ttFlash must dynamitcally* **disable** *Rotating GC to ensure there is always enough number of free pages.*

Conclusion

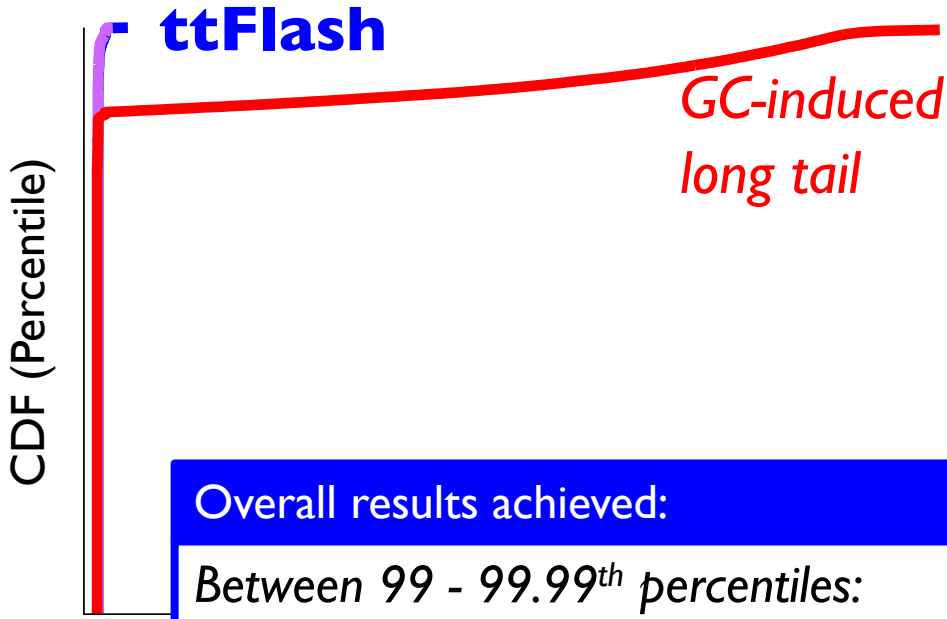
New techniques:

I. Plane-Blocking GC

II. GC-Tolerant Read

III. Rotating GC

IV. GC-Tolerant Flush



Overall results achieved:

Between 99 - 99.99th percentiles:

ttFlash 1-3x *slower than NoGC*

Base 5-138x *slower than NoGC*

technology: *Powerful Controller*

RAIN (parity-based redundancy)

Capacitor-backed RAM

Thank you!
Questions?

http://ucare.cs.uchicago.edu

https://ceres.uchicago.edu