2021 USENIX Annual Technical Conference (USENIX ATC ’21)

ASAP: Fast Mobile Application Switch via Adaptive Prepaging

Sam Son[1] Seung Yul Lee[1] Yunho Jin[1] Jonghyun Bae[1] Jinkyu Jeong[2]

Tae Jun Ham[1] Jae W. Lee[1] Hongil Yoon[3]

[1] Seoul National University [2] Sungkyunkwan University [3] Google

Memory Pressure in Today's Smartphone Usage

- Memory capacity is becoming a scarce resource on mobile devices
  - The application size and memory footprint have been growing
  - Users run more than 5 applications concurrently[1]
- However, the cost/power/area budget often limits its size

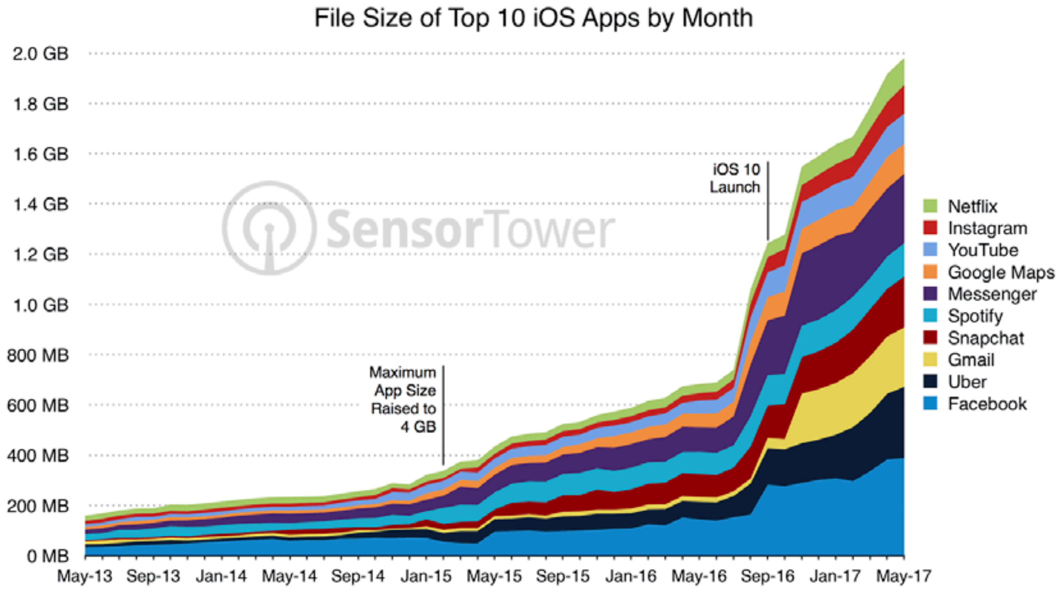


Image from https://sensortower.com/blog/ios-app-size-growth

[1] Yu Liang et al., "Acclaim: Adaptive Memory Reclaim to Improve User Experience in Android Systems" in *ATC'20*

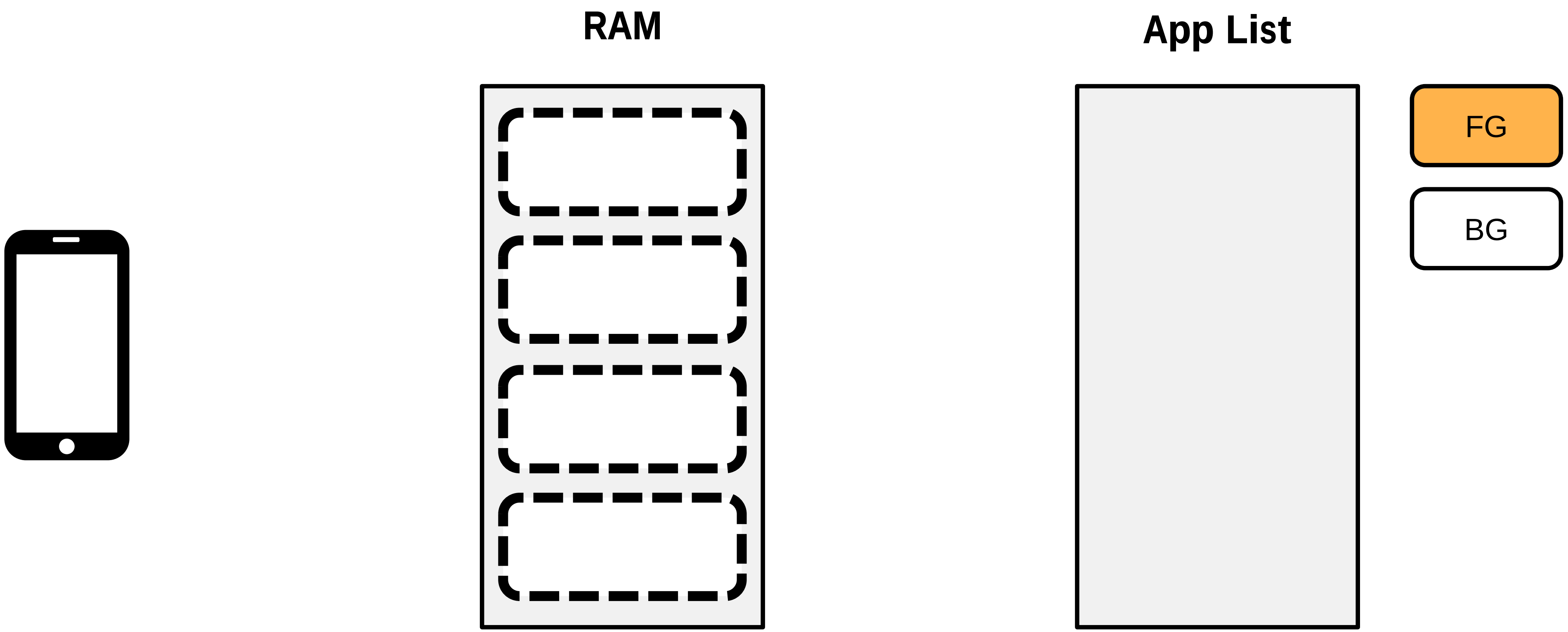
Memory Pressure Degrades UX

- Causes latency when users switch applications

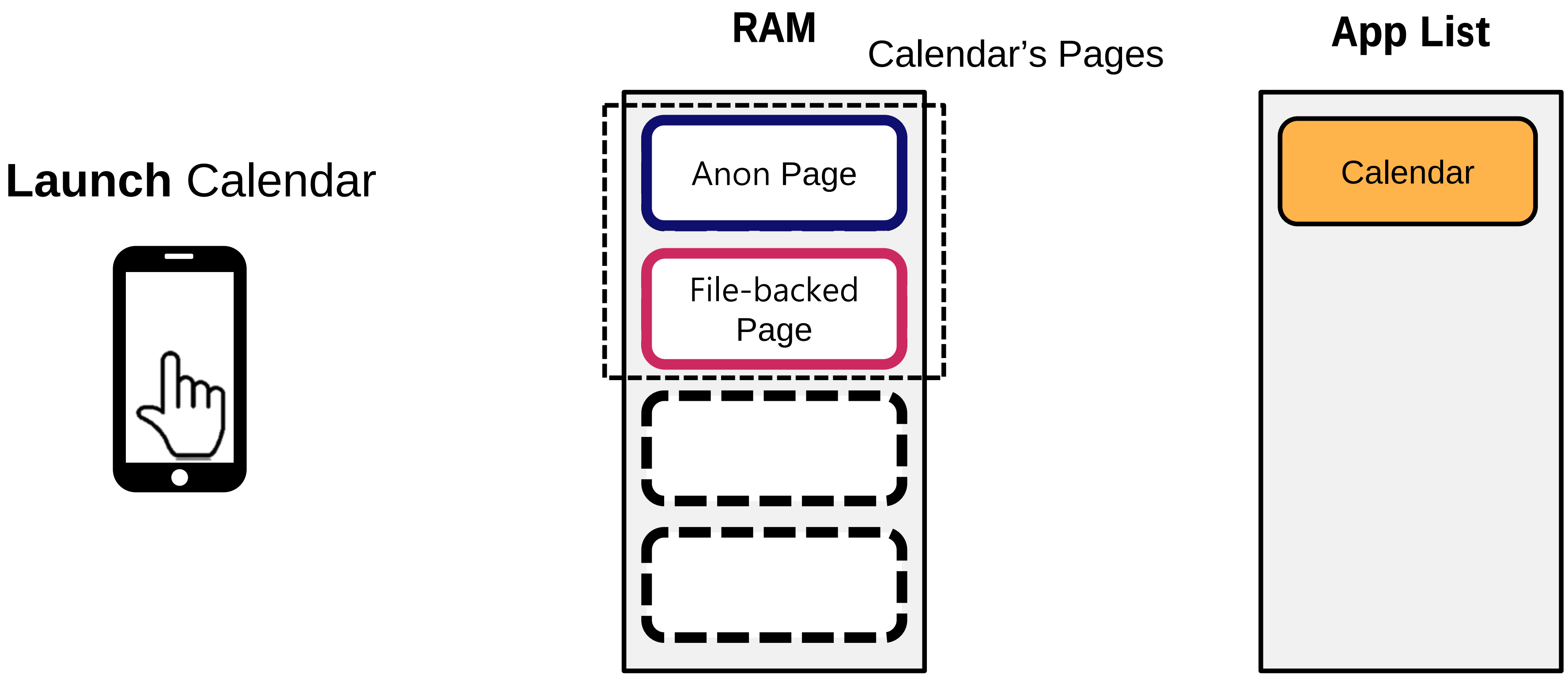
- Maintaining low latency is crucial
  - Users switch applications more than 100 times a day[2]

[2] Tao Deng et al., “Measuring smartphone usage and taskswitching with log tracking and self-reports” in Mobile Media & Communications 2018

Android Memory Management

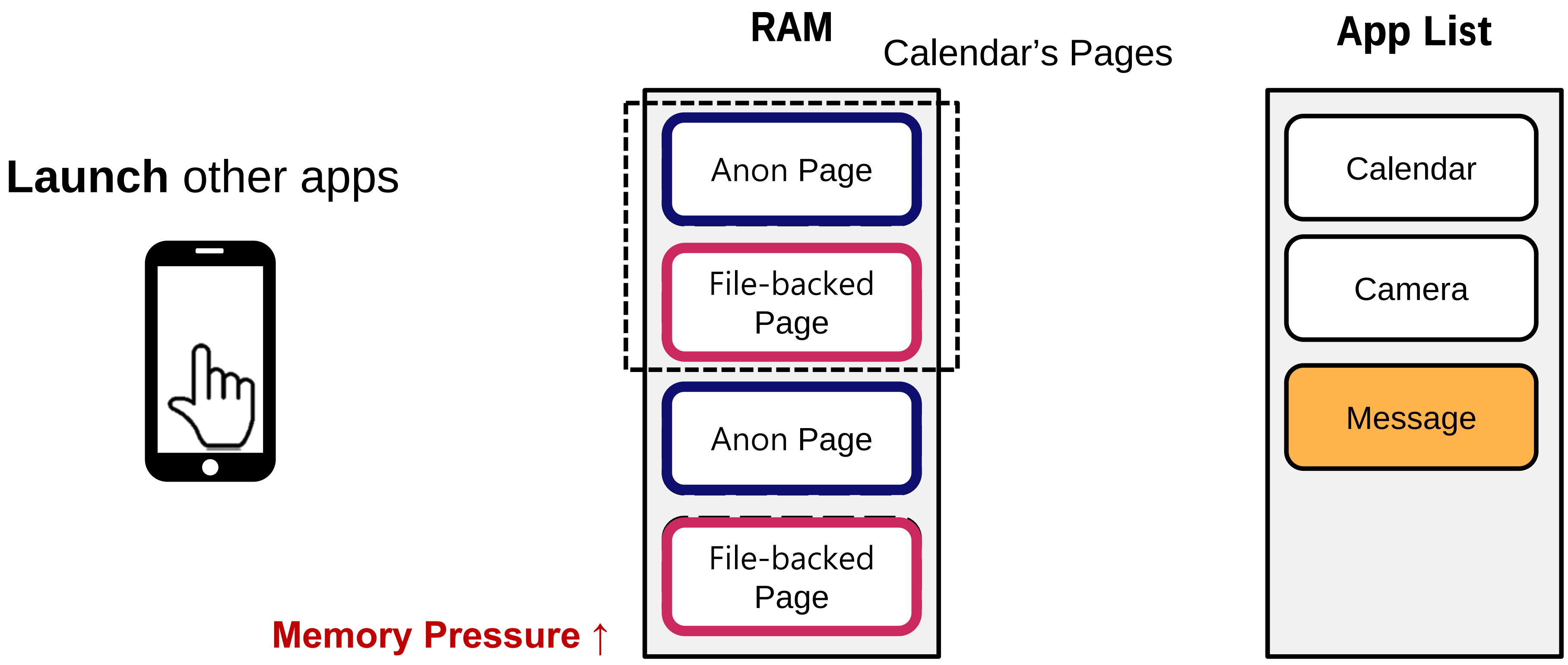


Android Memory Management



Application Launch creates an application process from scratch → takes long time

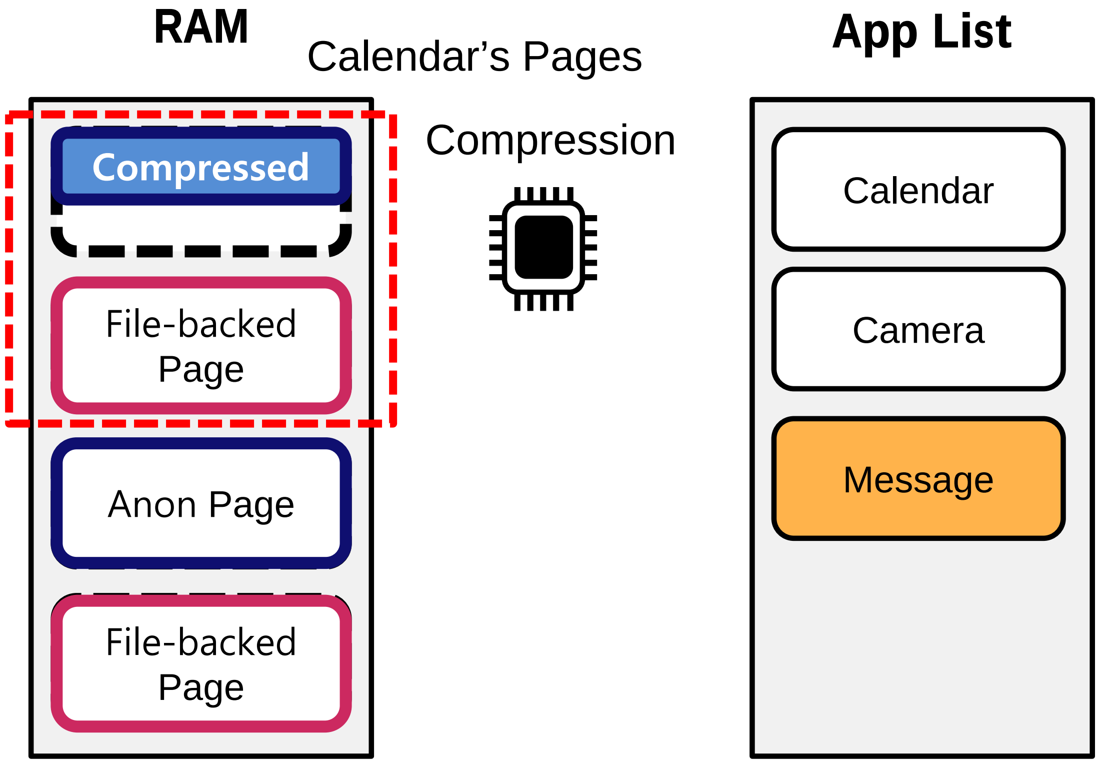
Android Memory Management



Launching more apps uses up all the memory

Android Memory Management

Launch other apps

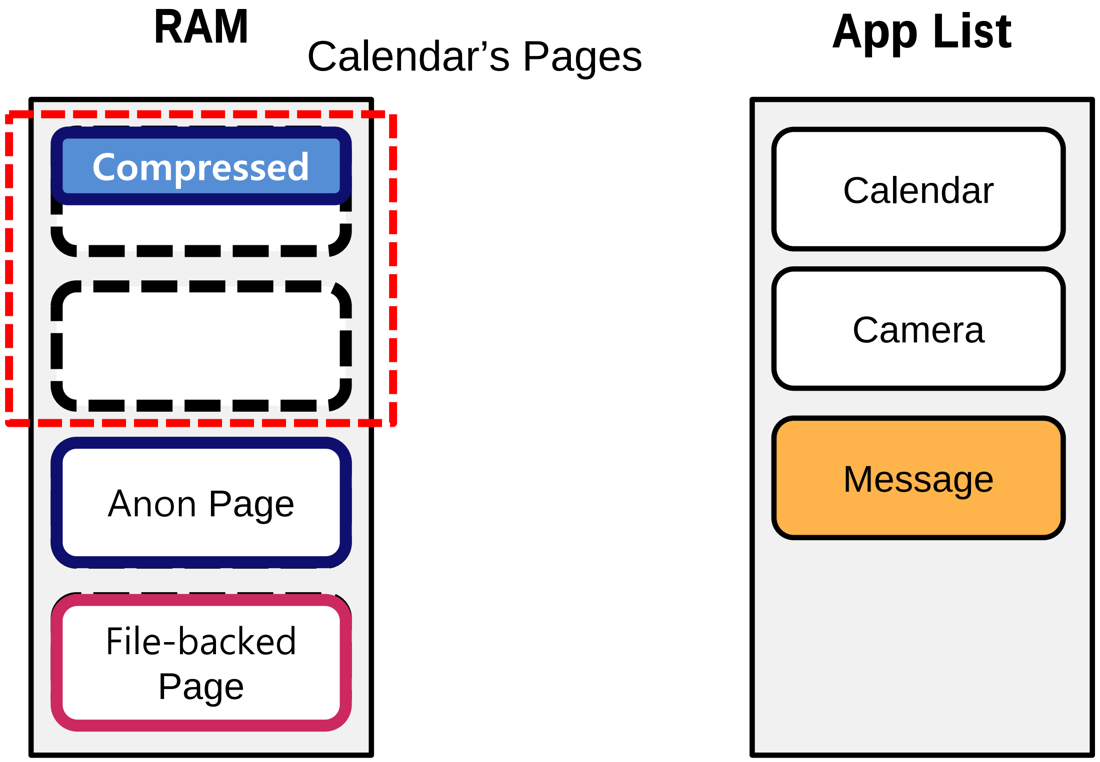


(1) Page Eviction

To secure free memory, OS **compresses** anonymous pages (compression-based swap)

Android Memory Management

Launch other apps

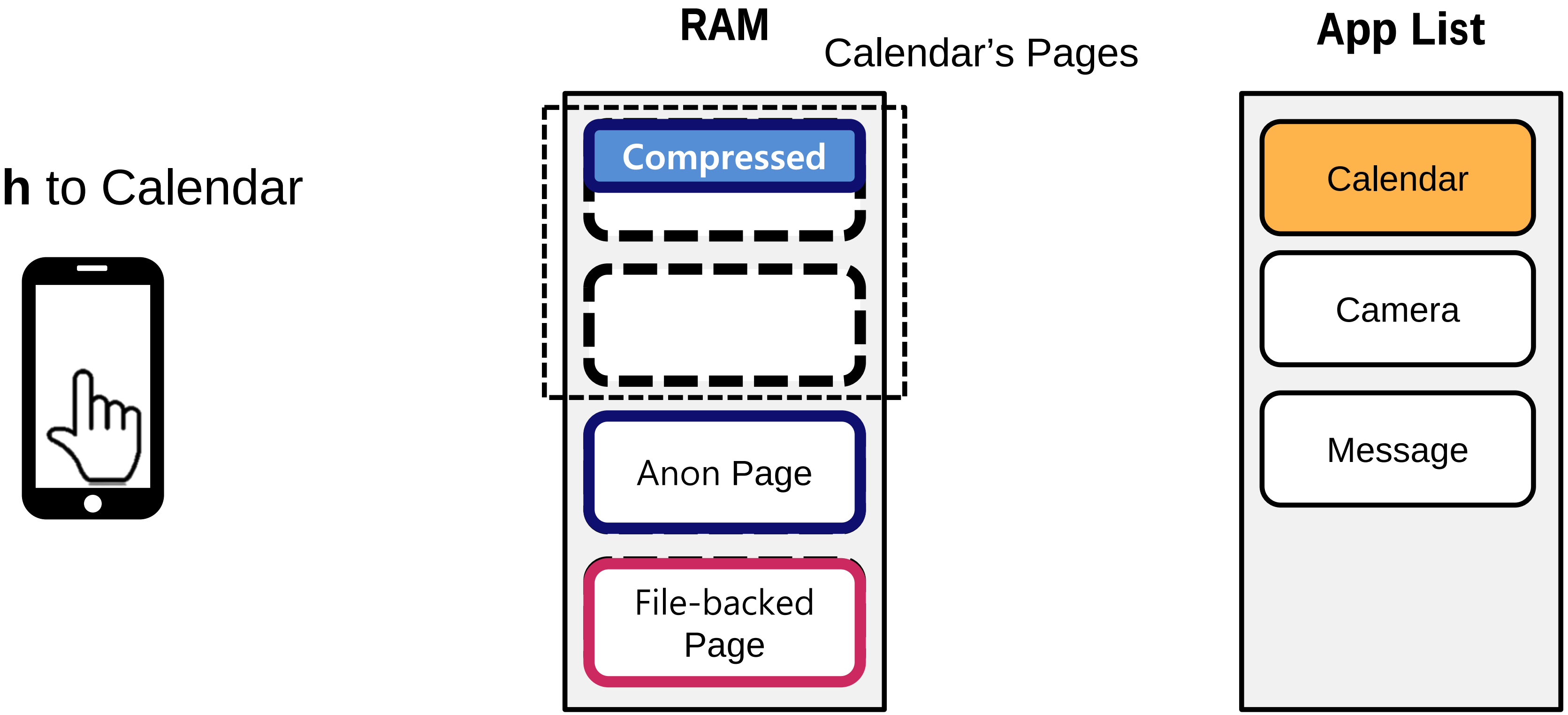


(1) Page Eviction

To secure free memory, OS **discards** file-backed pages

Android Memory Management

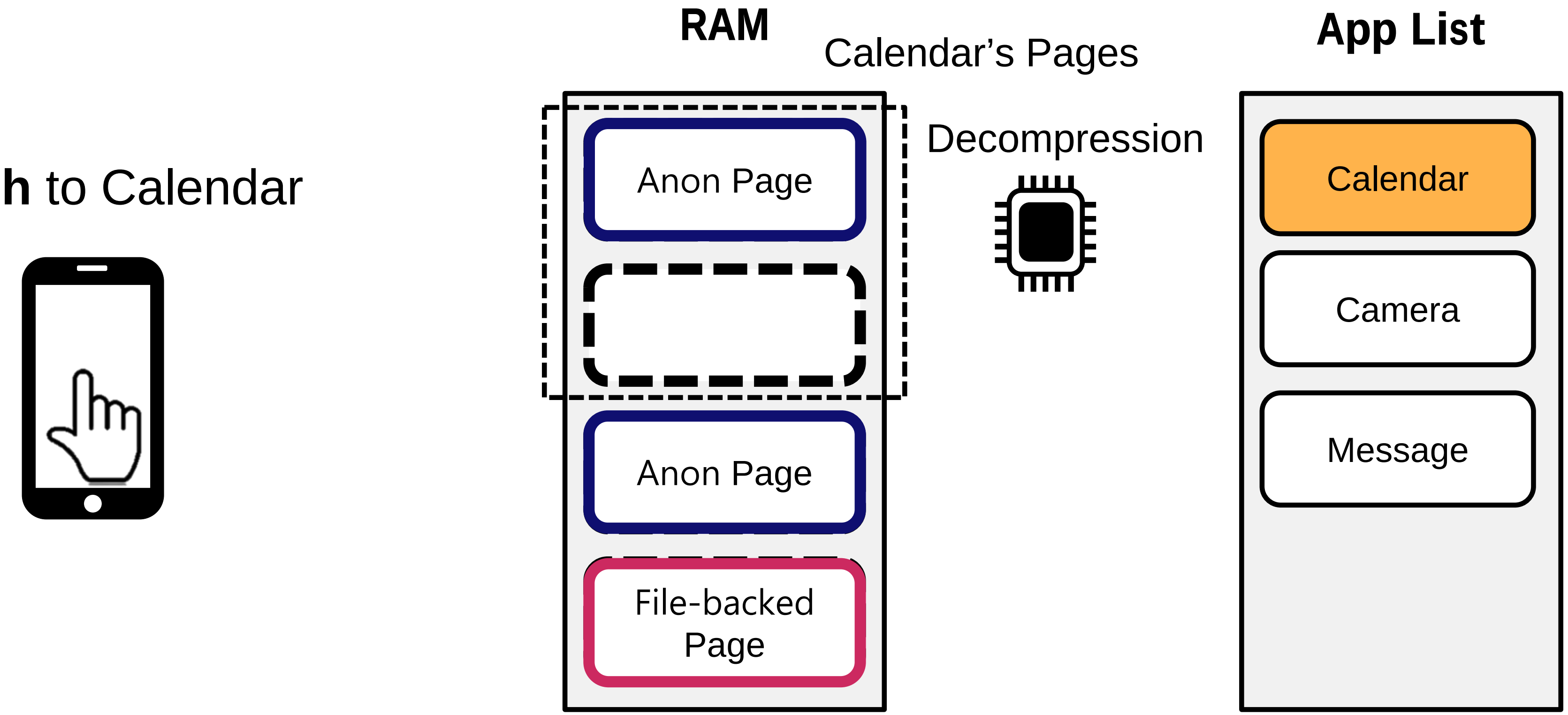
Switch to Calendar



Switching to Calendar is delayed due to on-demand page fetching

Android Memory Management

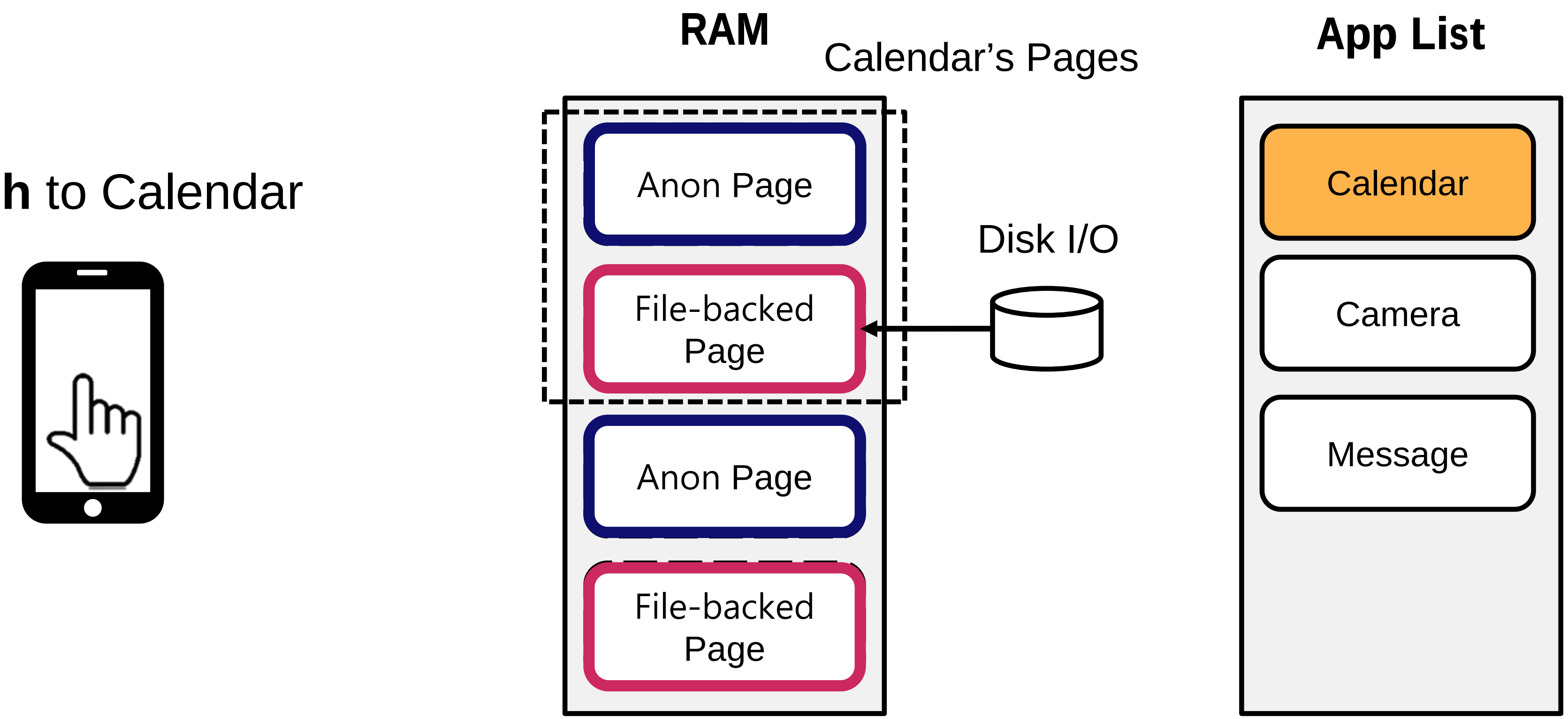
Switch to Calendar



Switching to Calendar is delayed due to decompressing anonymous pages

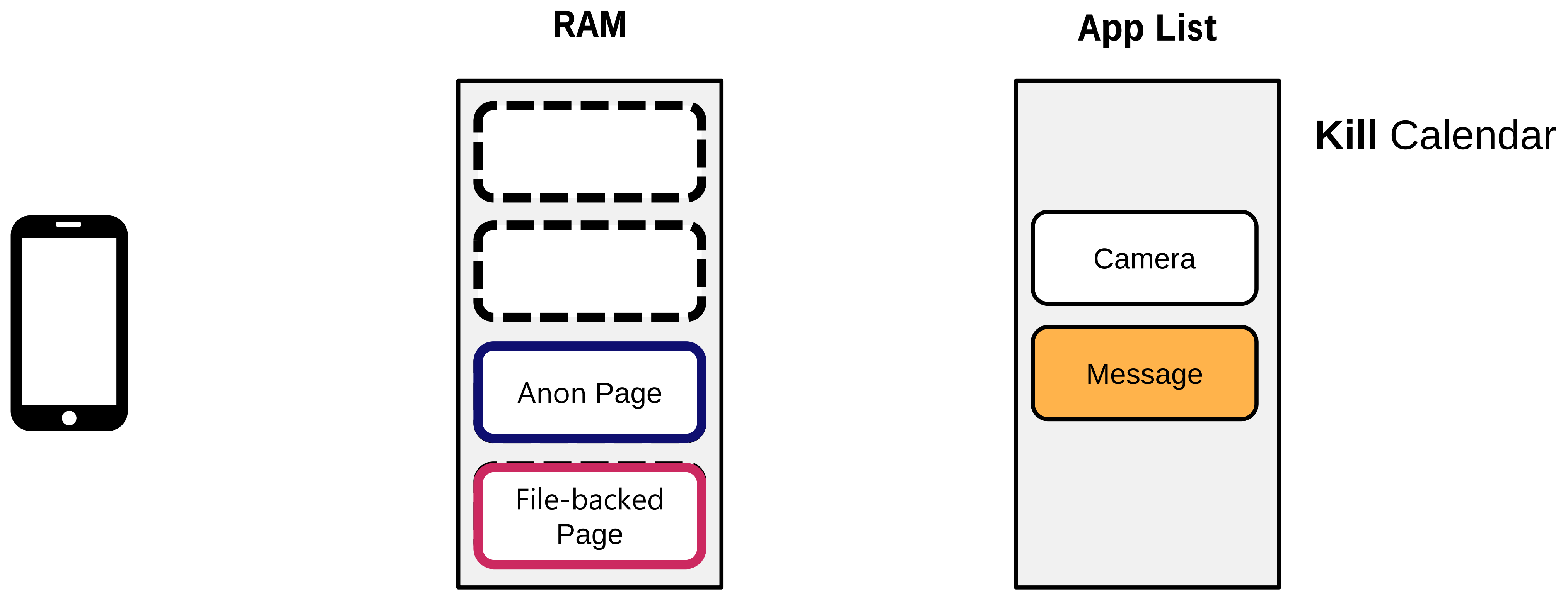
Android Memory Management

Switch to Calendar



Switching to Calendar is delayed due to reading file-backed pages from disk

Android Memory Management

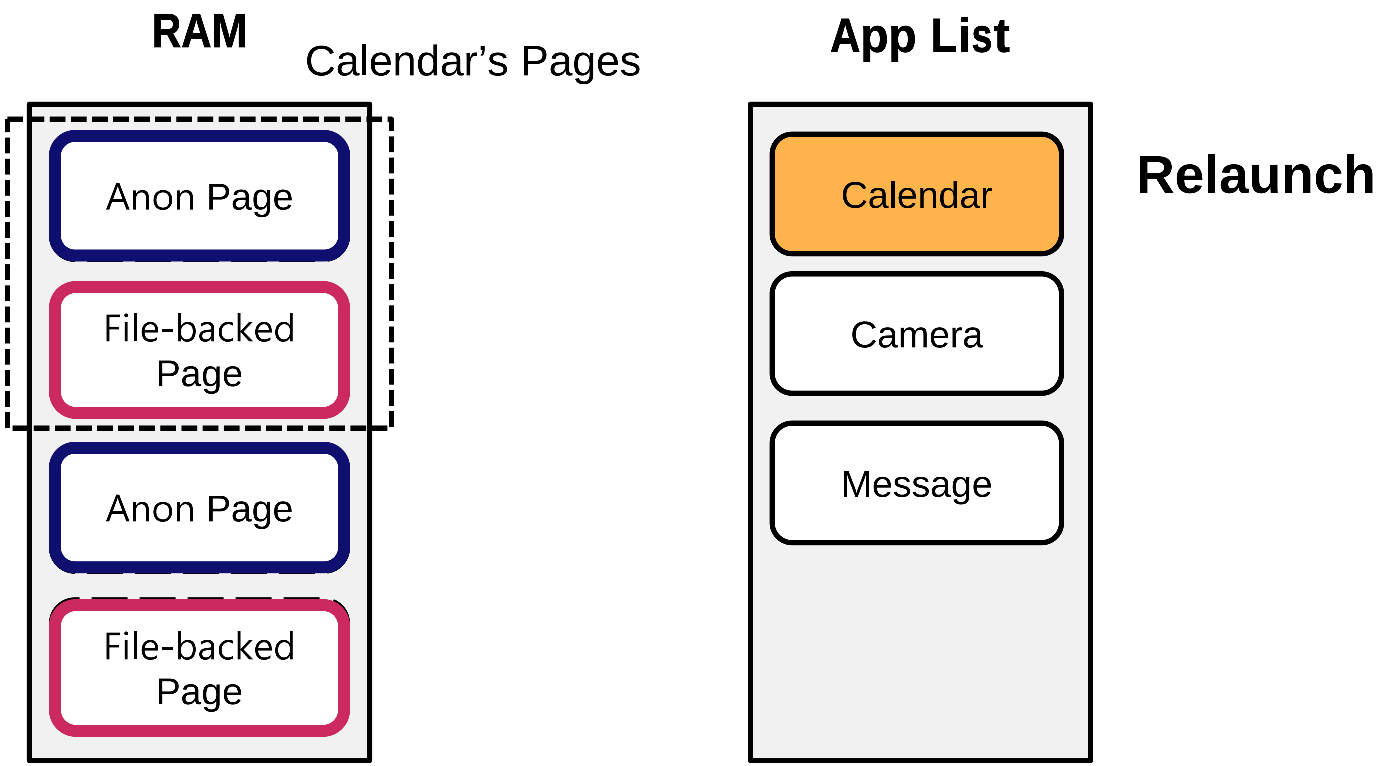


(2) Low Memory Killer (LMK)

Killing background application frees up pages

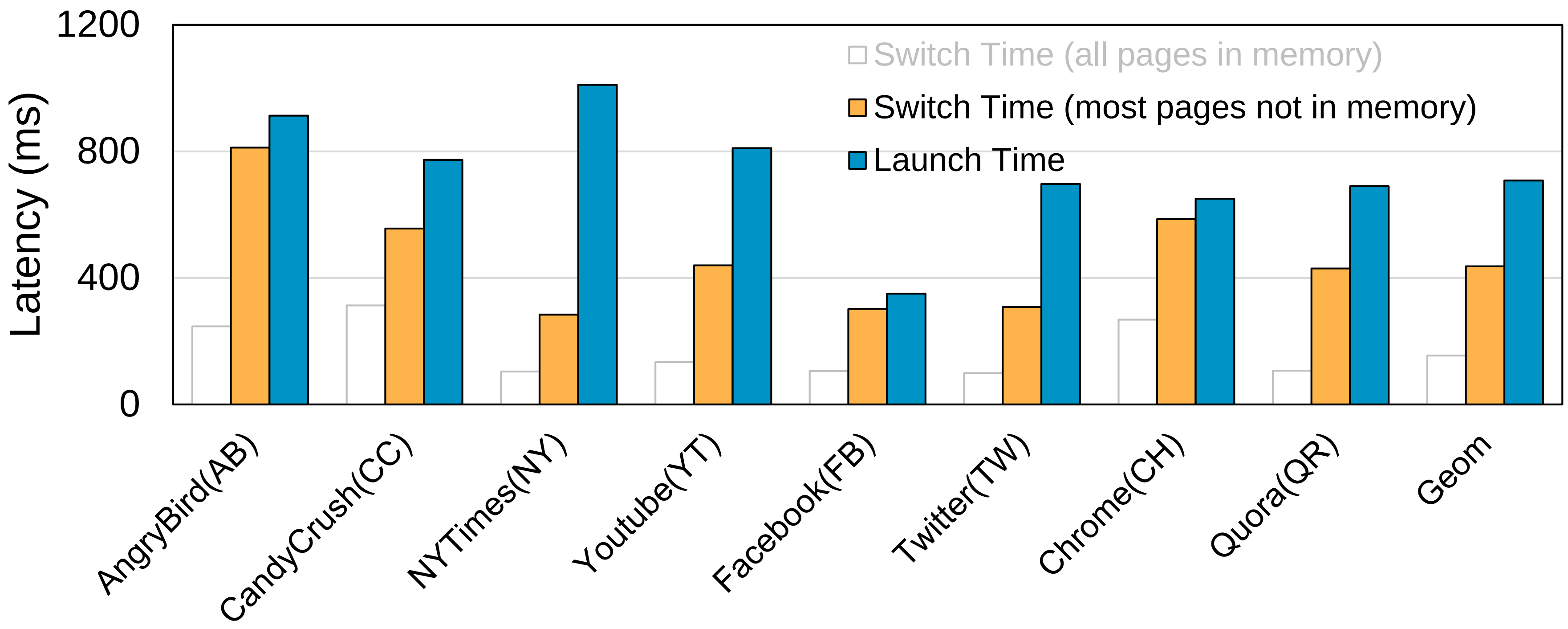
Android Memory Management

Switch to Calendar



This time, switching to Calendar causes slow **re-launching** of Calendar

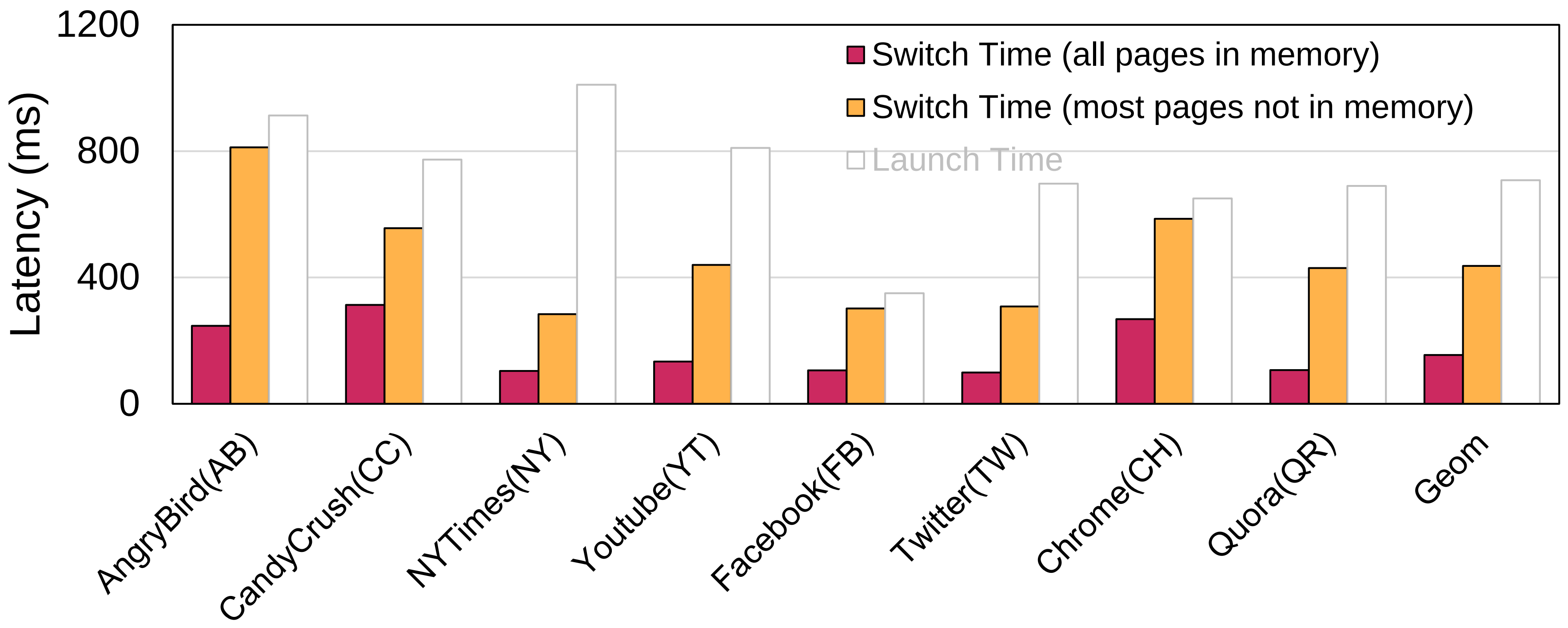
Application Switching Latency under Memory Pressure



Observation 1: Launch time is longer than switch time even when most pages not in memory

Implication: It is better to avoid relaunching by disabling LMK

Application Switching Latency under Memory Pressure

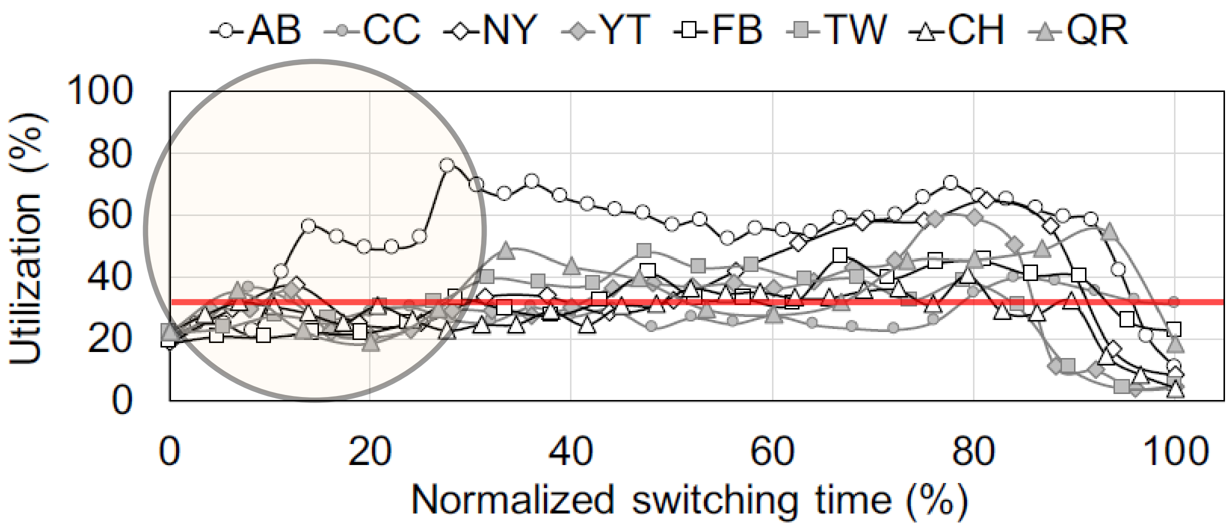


Observation 2: Switch time can increase by 4x on average under memory pressure

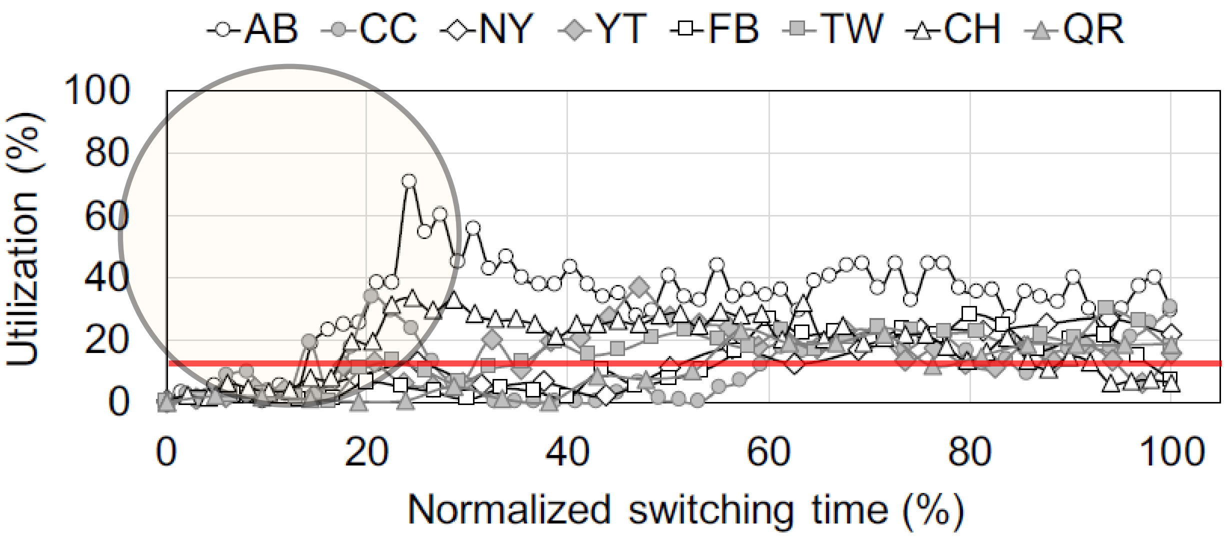
Implication: Retrieving relevant pages on-demand increases switch time a lot

Limitation of Demand-Paging

- Both CPU and disk BW are under-utilized during switch time
  - Page decompression is delayed until anonymous page fault occurs → low CPU utilization
  - Disk I/O is delayed until file-backed page fault occurs → low disk BW utilization
- On average, only **34%** of CPU and **15%** of disk BW are utilized during the switch time



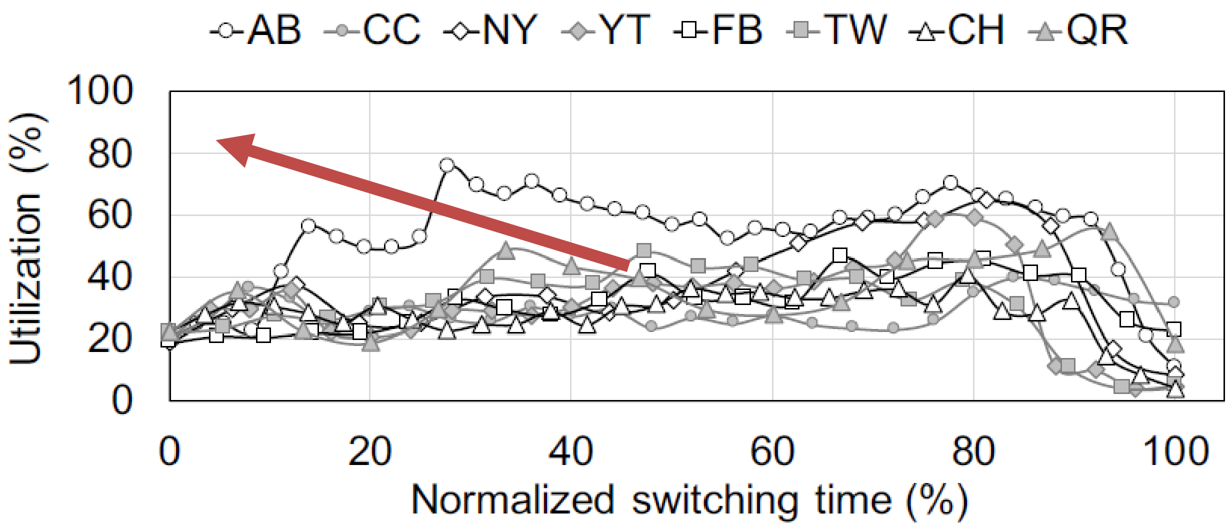
(a) CPU utilization



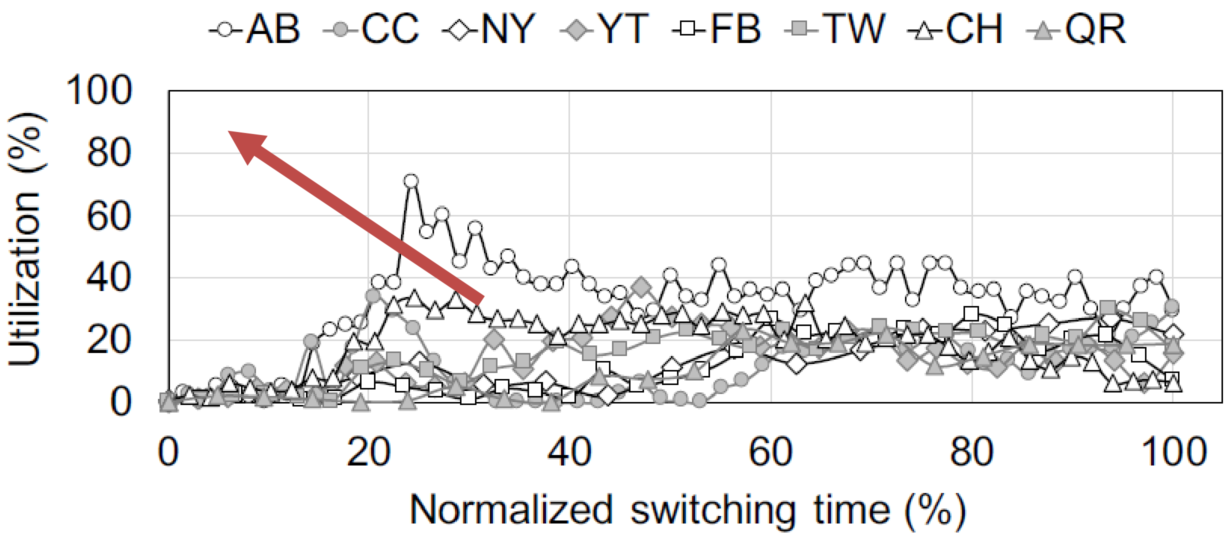
(b) Disk BW utilization

Opportunity of Prepaging

- Switch time can be improved by leveraging prepaging at the beginning of switch
- By doing so, available system resources (i.e., CPU cycles and disk bandwidth) can be fully utilized



(a) CPU utilization



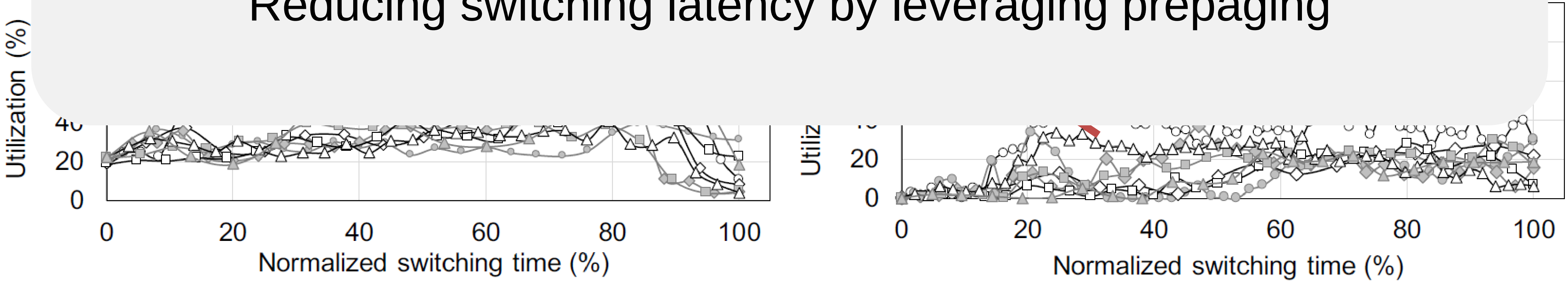
(b) Disk BW utilization

Opportunity of Prepaging

- Switch time can be improved by leveraging prepaging at the beginning of switch
- By doing so, available system resources (i.e., CPU cycles and disk bandwidth) can be fully

Our Goal

Reducing switching latency by leveraging prepaging



(a) CPU utilization

(b) Disk BW utilization

Challenges of Prepaging

What to Prepage?

- Applications' contexts keep changing
- Achieving both high coverage and low misprediction ratio

How to Prepage?

- Maximizing the efficiency by achieving high system resource utilization
- Minimizing contention with application threads

Application Switch via Adaptive Prepaging (ASAP)

- ASAP maintains low switching latency without LMK
- ASAP is **application-agnostic**, and requires **no changes** to applications codes

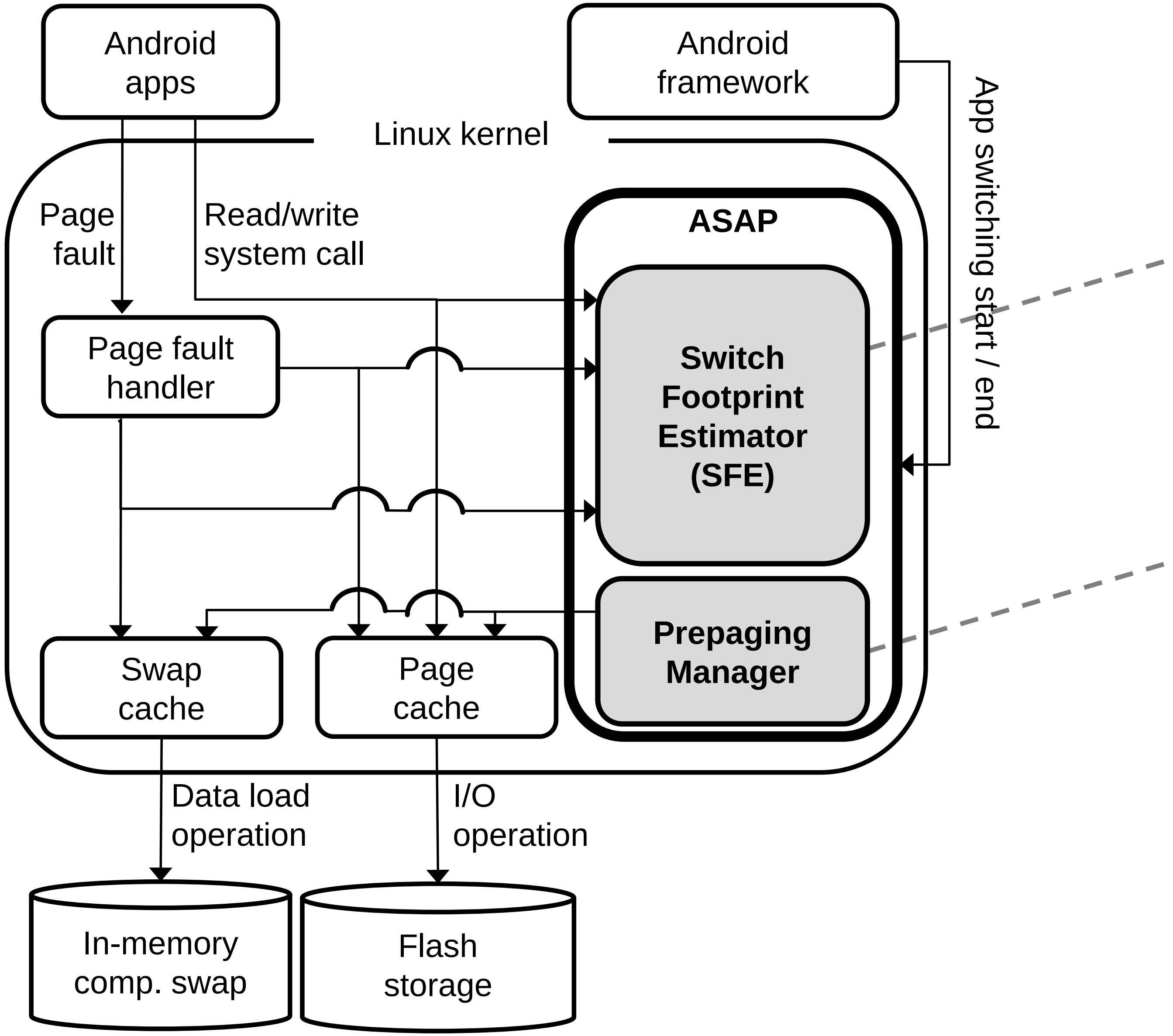
What to Prepage?

- Logging both page faults and I/O syscalls → **High coverage**
- Adaptively update based on feedback → **Low misprediction**

How to Prepage?

- Multiple prepaging threads → **High utilization**
- Opportunistically prepaging to minimize contention → **Low contention**

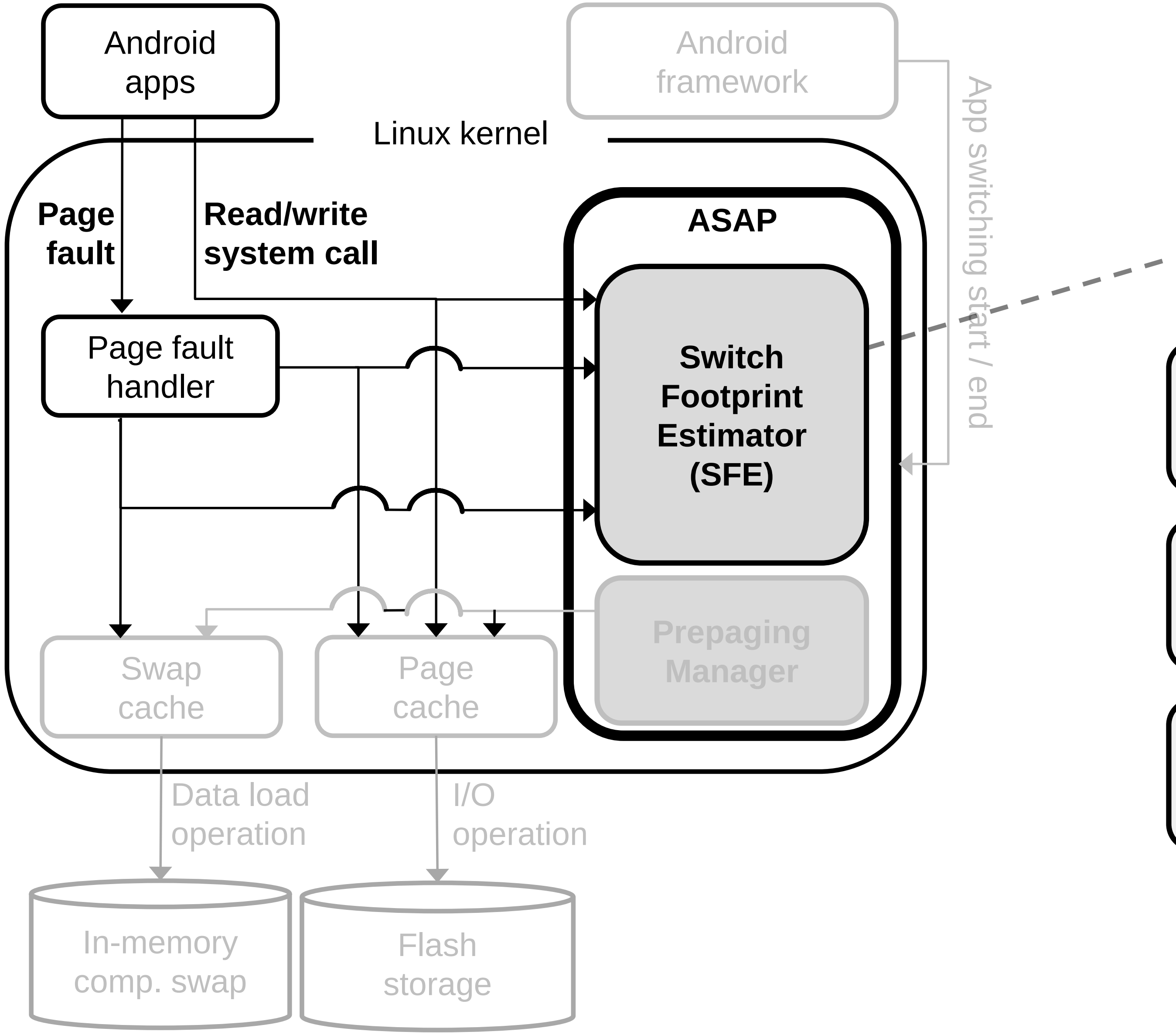
ASAP: Design Overview



What to prepage?

How to prepage?

Switch Footprint Estimator (SFE)



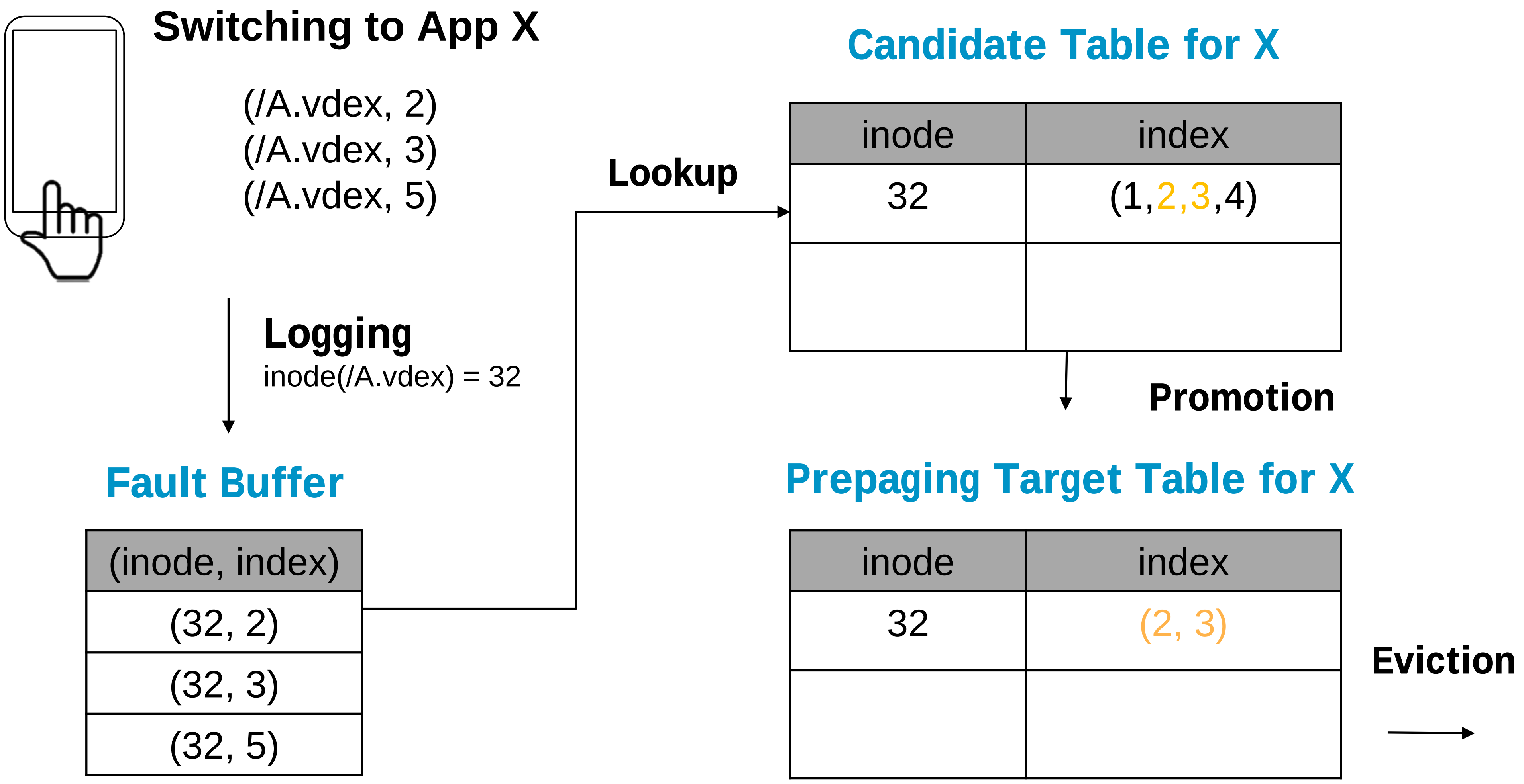
What to prepage?

Logging page access information

Maintaining Candidate Table

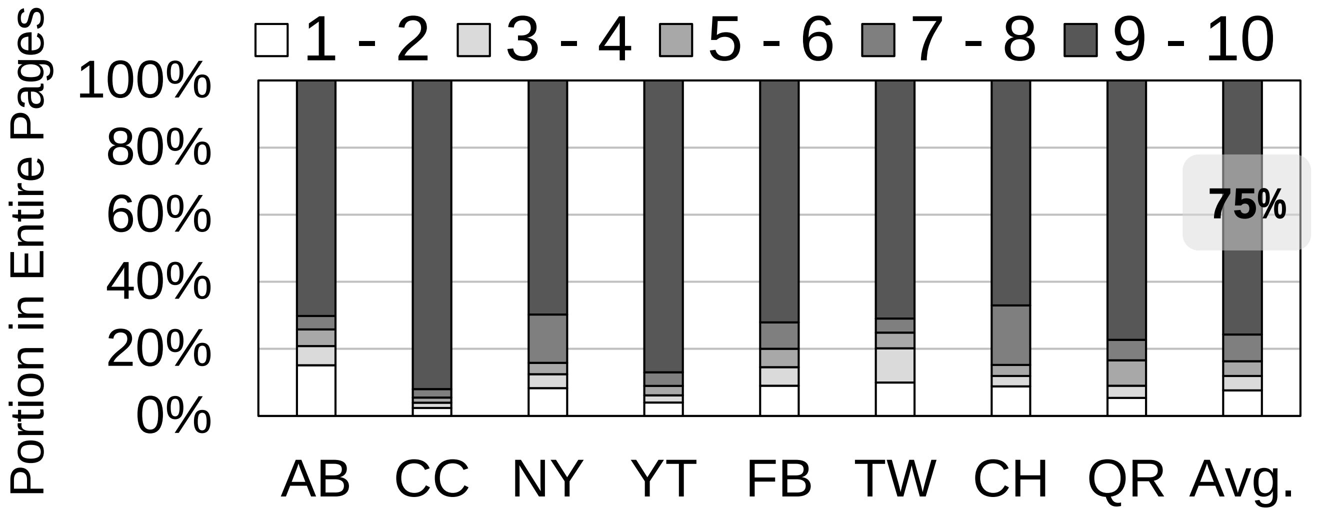
Maintaining Prepaging Target Table

Switch Footprint Estimator: Mechanism

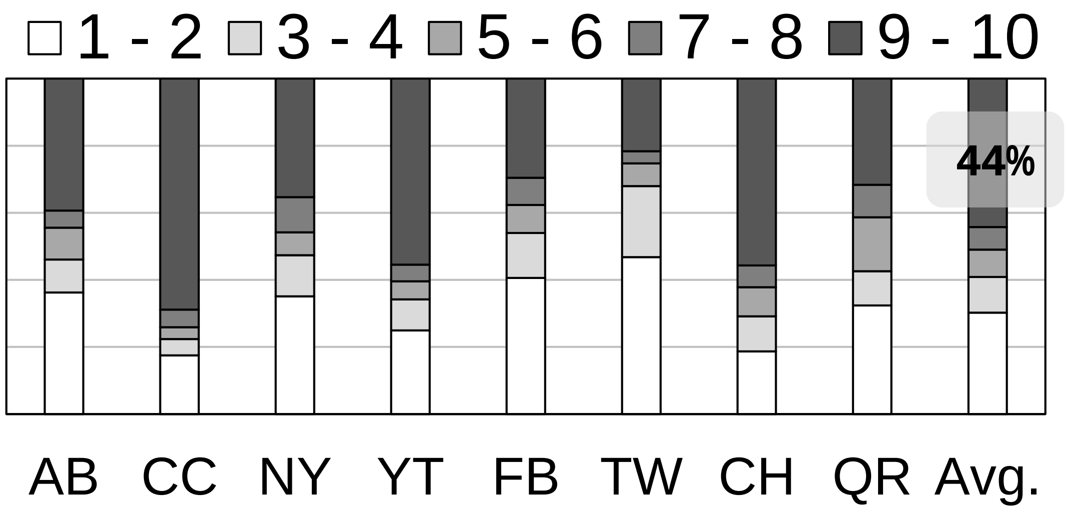


Optimized SFE for Each Type of Pages

- Anonymous pages and file-backed pages have different access patterns
- About 75% of all accessed file-backed pages are invariant across switches, while only 44% of anonymous pages are invariant

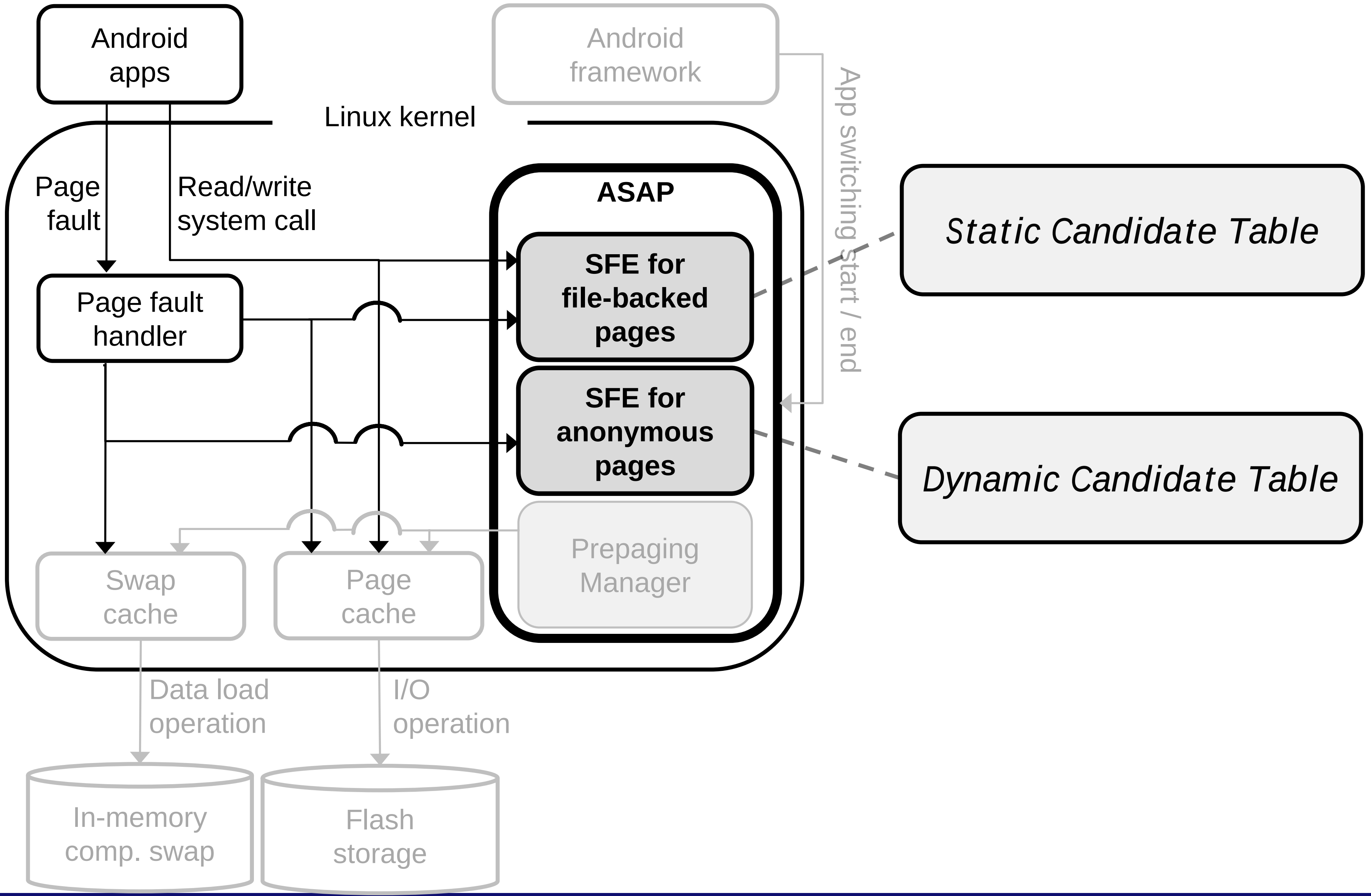


Access Frequency of **File-backed Pages** (over 10 switches)

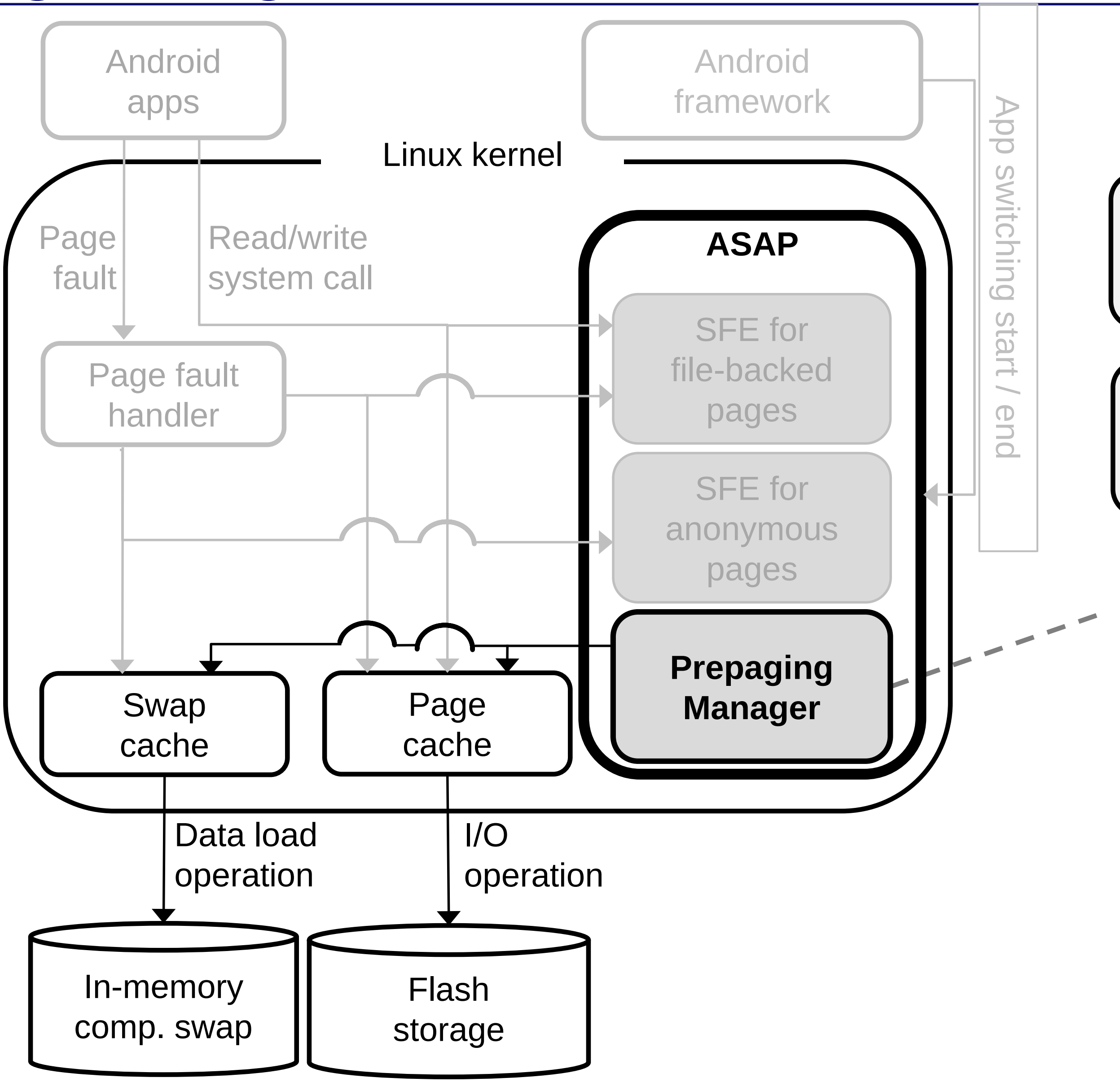


Access Frequency of **Anonymous Pages** (over 10 switches)

Optimized SFE for Each Type of Pages



Prepaging Manager



Fetching prepaging target

Optimzing prepaging threads

How to prepage?

Optimzing Prepaging Threads

- Batch processing minimizes lock contention between prepaging threads
  - 16 pages for anonymous pages
  - All target pages of one file for file-backed pages

- Giving low schedule priority to avoid CPU contention with app threads
  - SCHED_IDLE(lowest) for prepaging threads
  - Opportunistically prepaging

Evaluation Methodology

- Integrated ASAP into Android OS
- Evaluated ASAP on high-end and mid-end devices (Google Pixel 4 and Pixel 3a)
- 8 popular mobile applications with diverse automated usage patterns

Device Specification

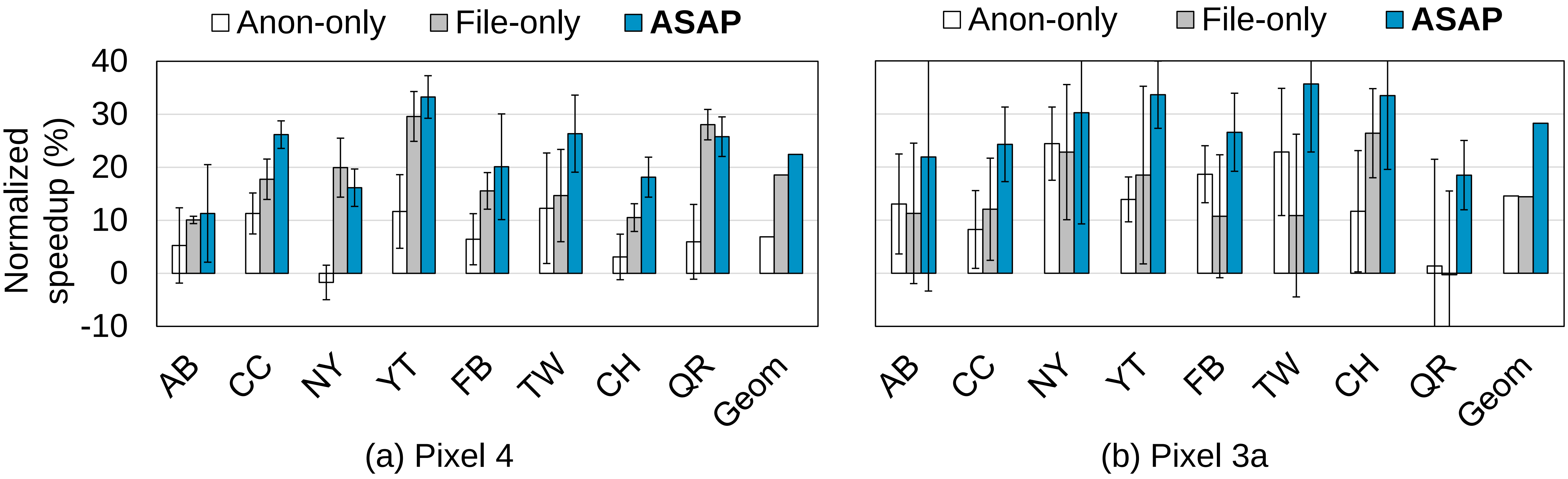
Device	Google Pixel 4	Google Pixel 3a
CPU	Snapdragon 855	Snapdragon 670
DRAM	6GB (effective 4GB)	4GB
Storage	UFS 2.1	eMMC 5.1
OS	Android 10.0.0(r41) with Linux kernel 4.14	Android 10.0.0(r41) with Linux kernel 4.9

Benchmark Applications and Usage Pattern

Application	**Usage Pattern**
Angry Bird (AB)	Play a stage
Candy Crush (CC)	Play a stage
New York Times (NY)	Browse and read articles
Youtube (YT)	Watch Videos
Facebook (FB)	Browse and read posts
Twitter (TW)	Browse and read posts
Chrome (CH)	Browse keywords
Quora (QR)	Browse questions and answers

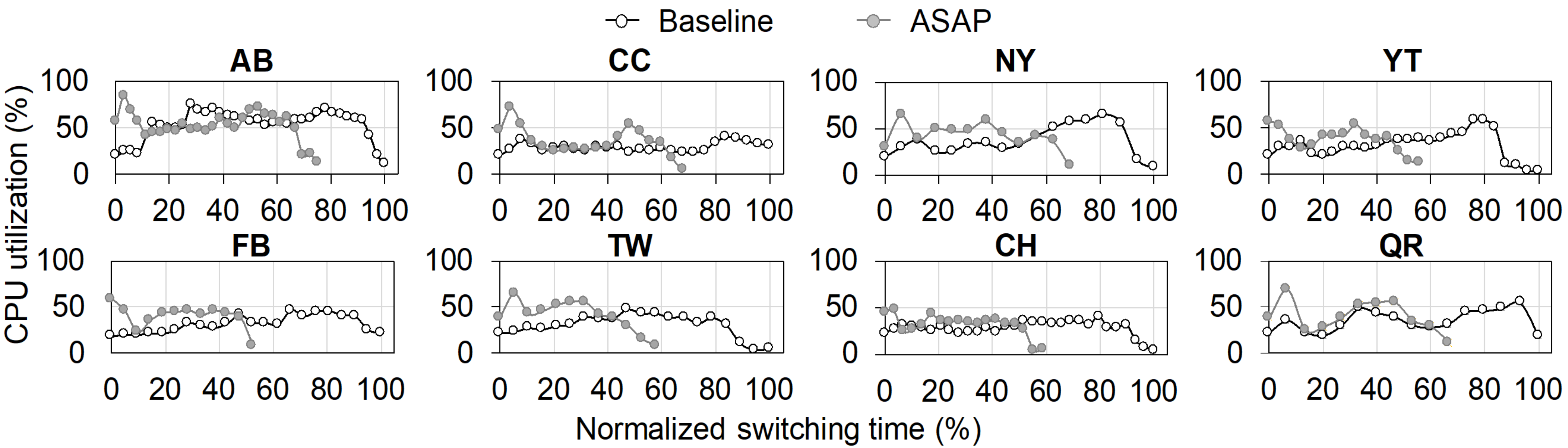
Switching Latency Reduction

- Baseline: switching latency when 8 applications run concurrently (high memory pressure)
- Up to **33.3%** (**22.2%** on average) latency reduction on Google Pixel 4
- Up to **35.7%** (**28.3%** on average) latency reduction on Google Pixel 3a



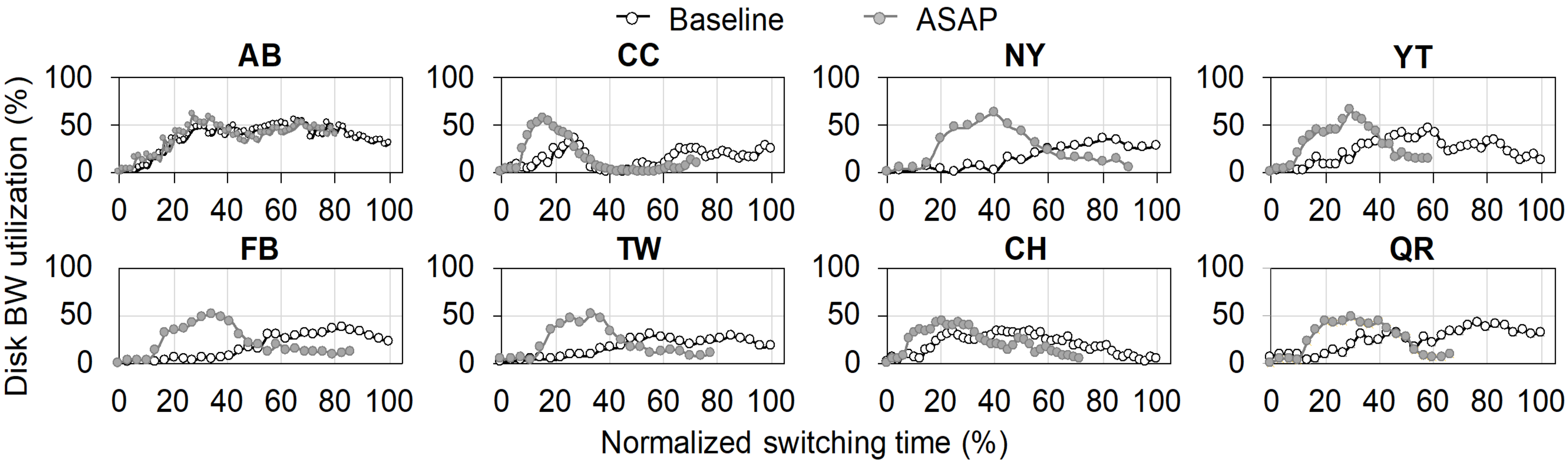
Improved CPU Utilization

- Noticeable increase in the CPU cycles at the early phase of switching
- Higher CPU utilization (Up to **35%**, average **18%**)



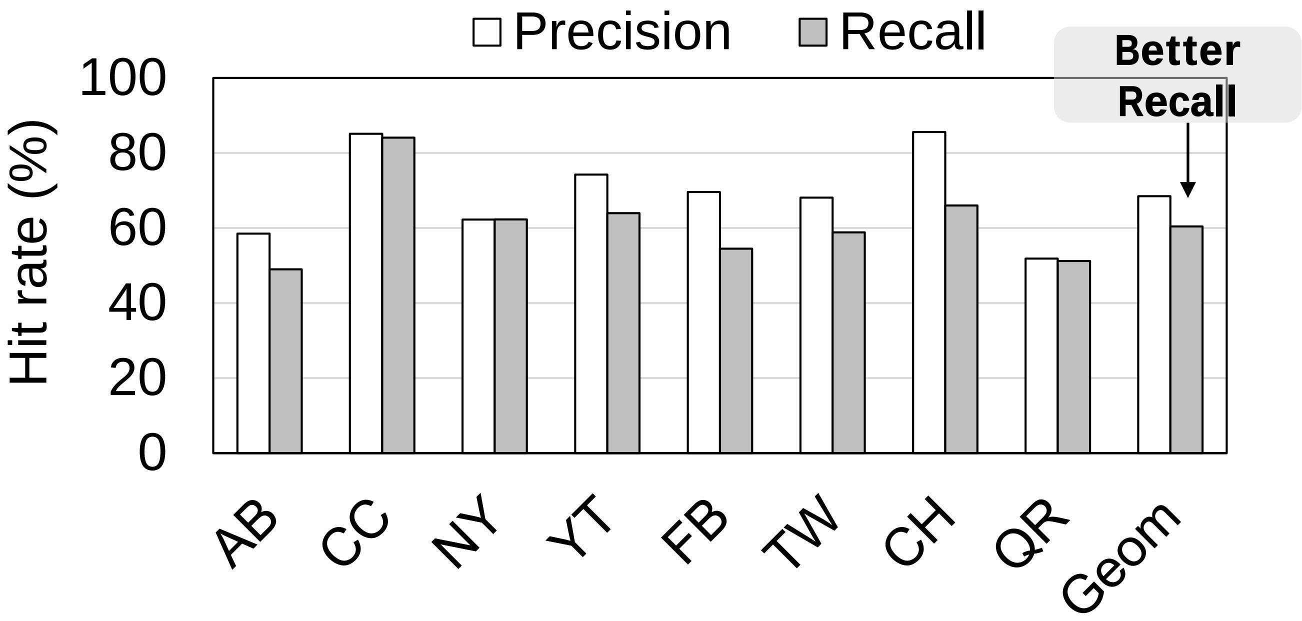
Improved Disk Bandwidth Utilization

- Noticeable increase in the I/O bandwidth at the early phase of switching
- Higher disk BW utilization (Up to **35%**, average **25%**)

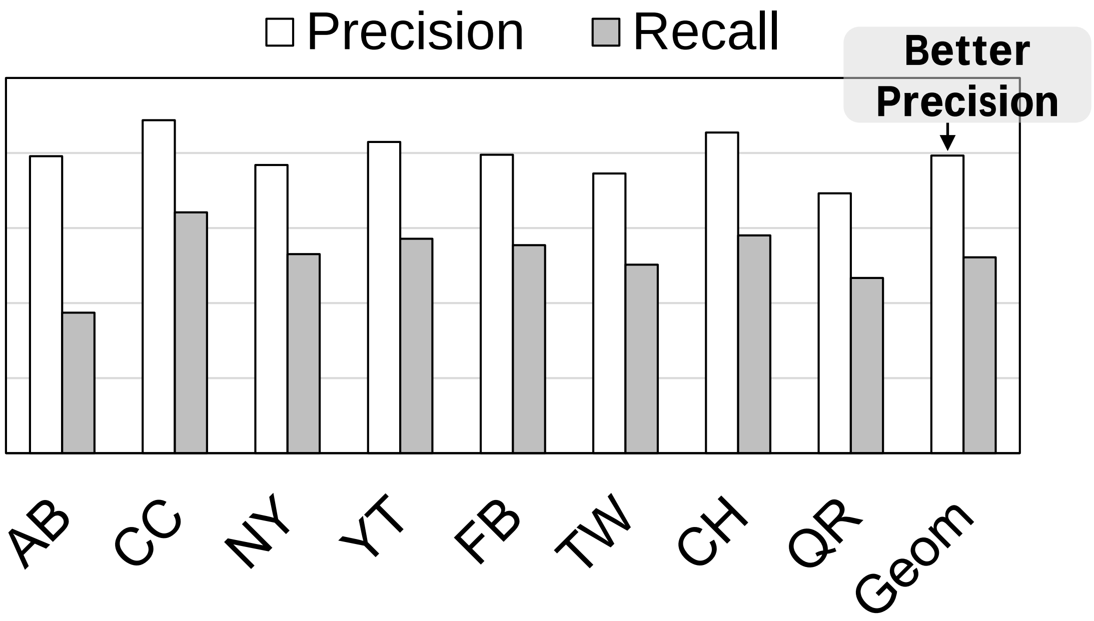


Switch Footprint Estimator Efficiency

- Higher Precision → Lower misprediction
- Higher Recall → Higher coverage
- SFE for file-backed pages shows **better precision** due to static access pattern
- SFE for anonymous pages shows **better recall** due to dynamic candidate table



(a) SFE for anonymous pages



(b) SFE for file-backed pages

Conclusion

ASAP provides better UX to mobile users by reducing latency of application switch

Contributions:

- Identified performance bottlenecks of application switching time
- Identified the root cause of low resource utilization during application switch
- Designed an application-agnostic prepaging technique
- Achieved up to 35.7% latency reduction on Google Pixel devices

Thank You!

ASAP's Android kernel code is available at

https://github.com/SNU-ARC/atc21-asap-kernel

Sam Son, sosson97@snu.ac.kr