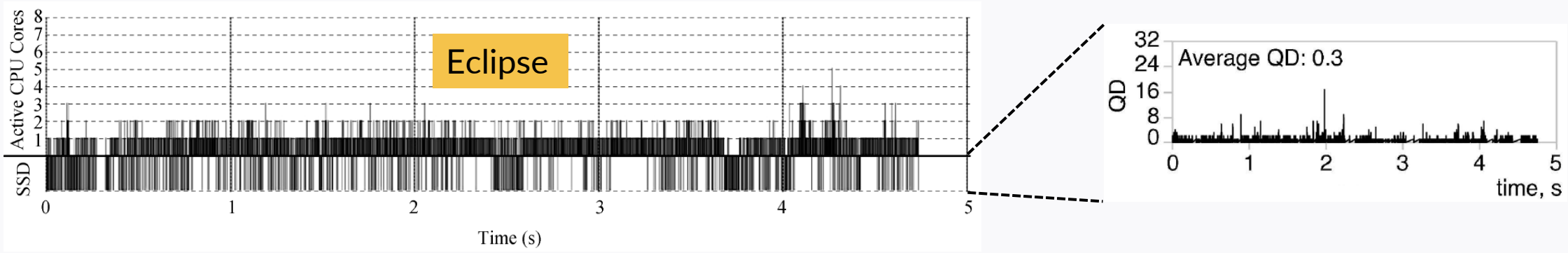
Fast Application Launch on Personal Computing/Communication Devices

Junhee Ryu, Dongeun Lee, Kang G. Shin, Kyungtae Kang

The first author works at SK Hynix, but this work was done as a personal interest.

Motivation

- Launch times remain an important problem
  - The performance of flash storage does not always come up to expectations
    - The use of QLC SSDs extends the launch time significantly compared to TLC SSDs
      - ***Blade and Soul***: ***91s → 114s***, ***Horizon Zero Dawn: 15.7s → 21.4s***
      - ***Windows 10 boot: 27.3s → 35.1s***, ***Visual Studio: 7.0s → 8.3s***
    - Cost-effective architecture such as DRAM-less SSD exhibits slow access performance
  - Application launches are rarely able to exploit hardware-level parallelism



Related Works to Expedite Application Launch

Hybrid SSD

Intel H20 (Optane + QLC), Enmotus MiDrive (SLC + QLC), Solidigm P41+ (pSLC + QLC)

Flash Cache

Intel Turbo Memory, Intel Smart Response Technology, Enmotus FuzeDrive, DM-Cache, Bcache

Predictive Disk Prefetcher

Preload, Windows Superfetch, Falcon [Mobisys'12]

I/O Scheduler

FastTrack [ATC'18], BFQ I/O Scheduler [Systor'12]

Memory Management

Marvin [ATC'20], Acclaim [ATC'20]

Application Prefetcher

- ***Windows Prefetcher*** (since Windows XP)
  - Customized for HDDs
  - File-level sorting
  - Defragment blocks (every 3 days)
- ***GSoC Prefetch*** (Linux)
  - Similar to Windows Prefetch
  - Selected for the Google Summer of Code 2007
- ***FAST*** (Linux) [FAST '11]
  - Customized for SSDs
  - I/O-level monitoring (using **blktrace**)
  - Creates a prefetch program that loads monitored I/Os into disk caches
  - Threaded prefetching

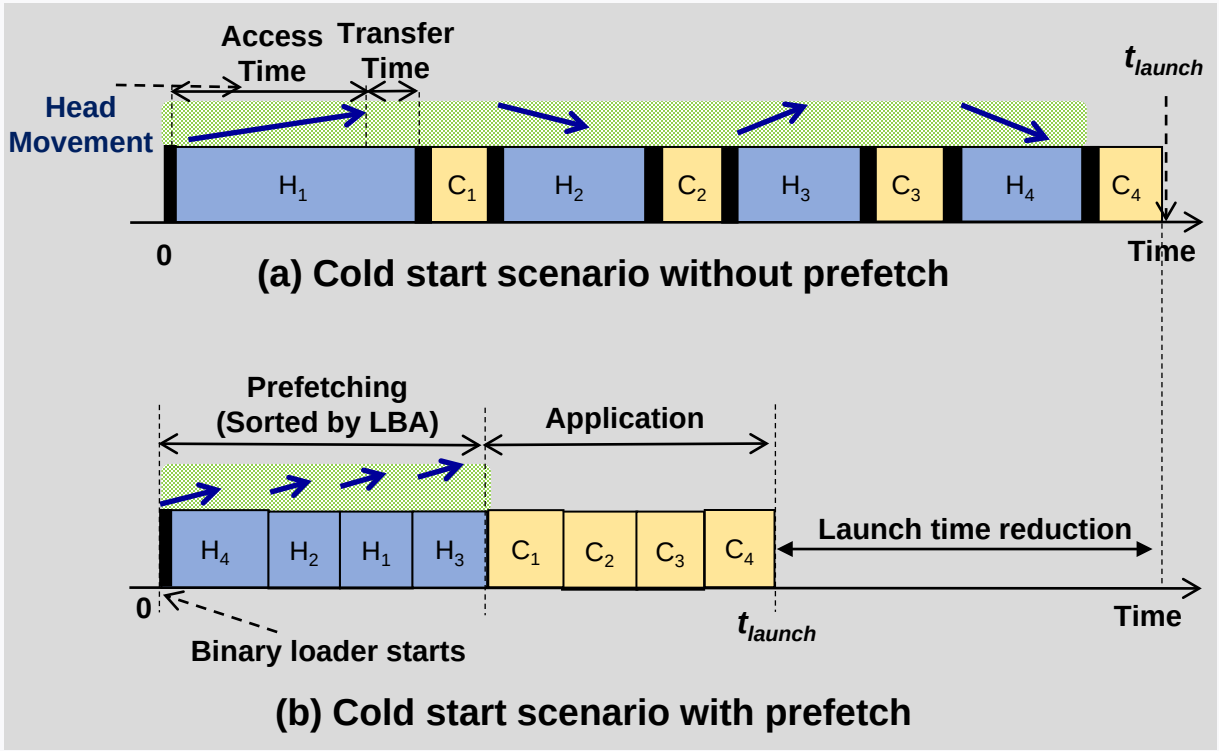
Launch Scenarios: Cold Start vs. Warm Start

- Launch times depend on the previous state of the system, especially the disk cache
- A ***cold start*** occurs when the disk cache contains none of the data required by the application
  - A ***system cold start*** occurs when there is no user-launched app
- A ***warm start*** occurs when the disk cache contains all the requested data during the launch of an app

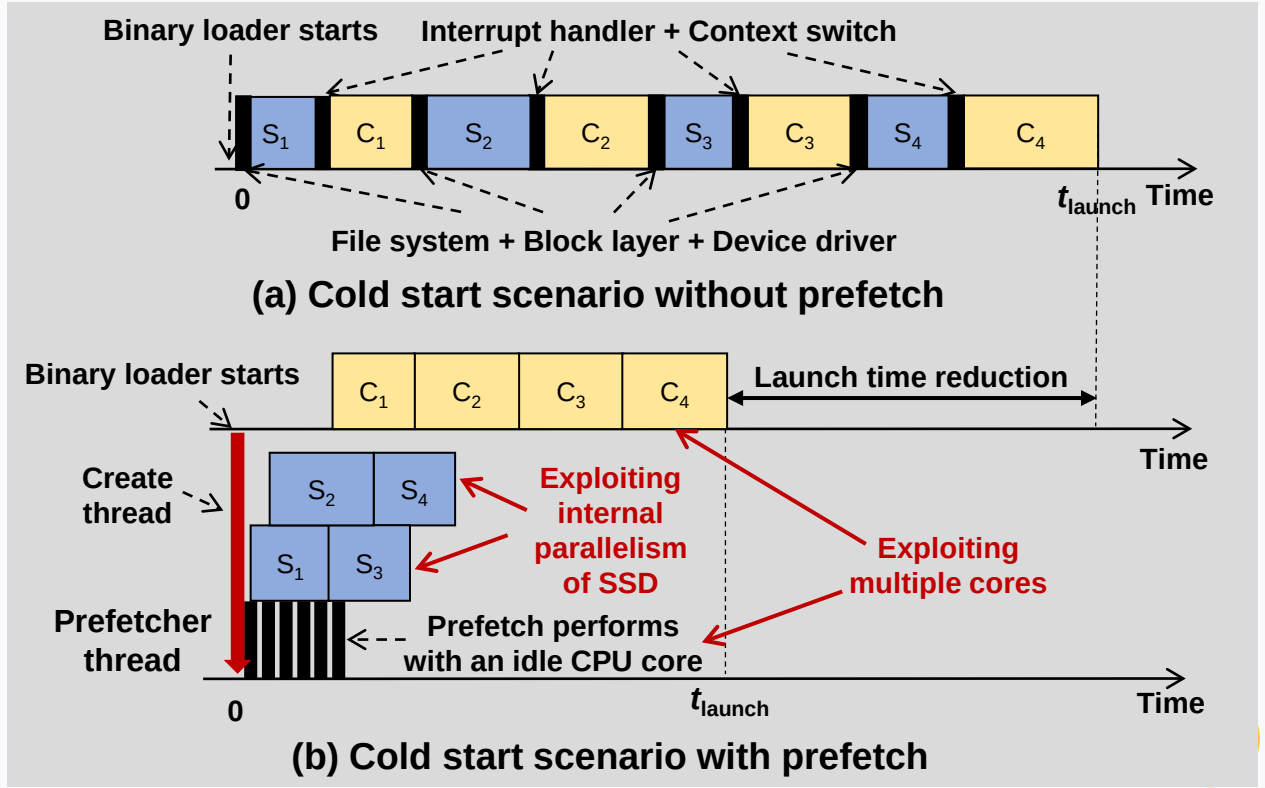
Paralfetch: Application Prefetcher

- ***Learning phase***: monitor disk reads and page faults during the first launch of an application
- ***Prefetch phase***: stored information is used to accelerate loading during a subsequent launch of the application

Sorted Prefetch for HDDs



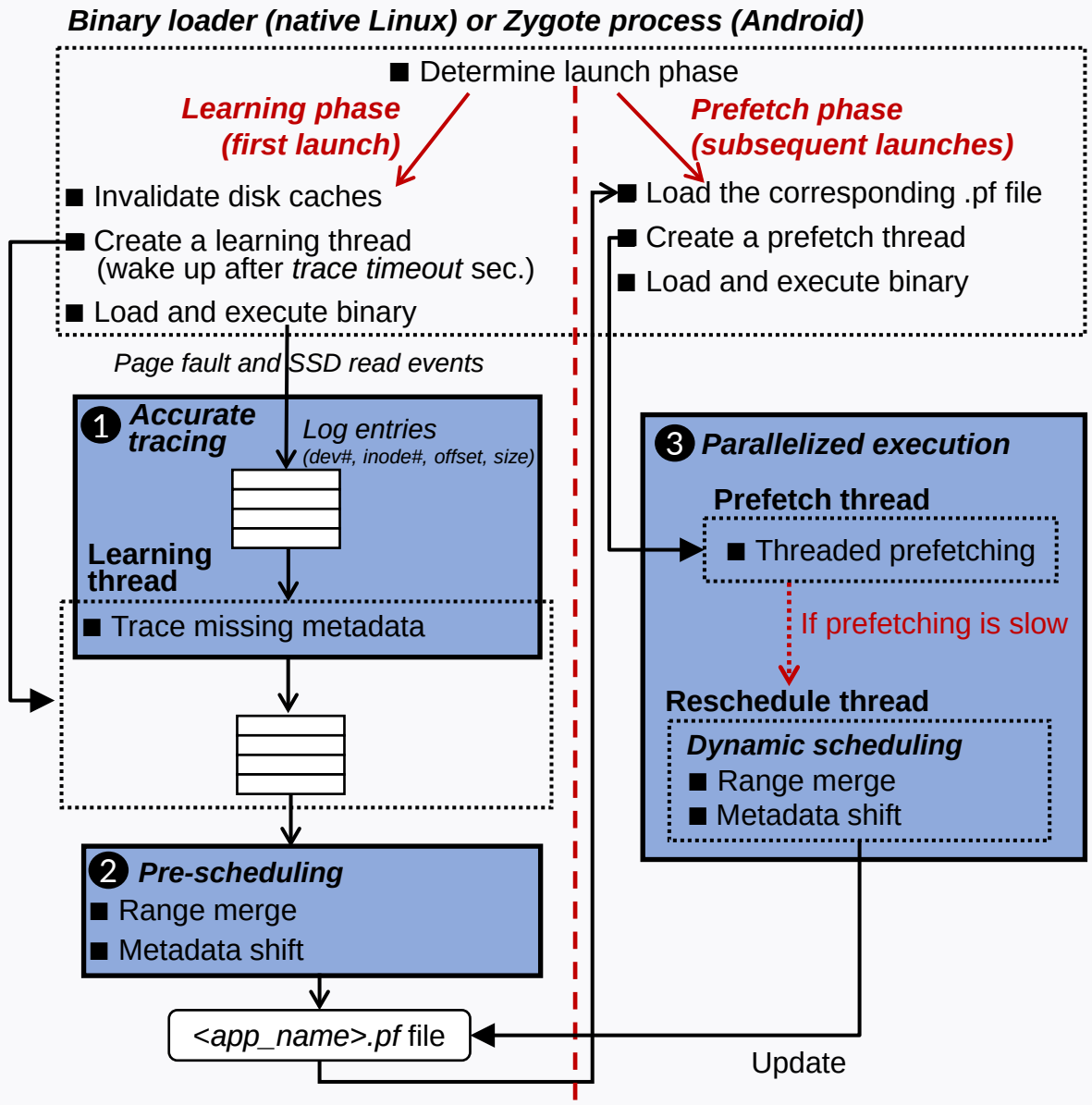
Threaded Prefetch for SSDs



Fundamental Challenges

- ***Accurate collection*** of launch-related disk blocks
- ***Pre-scheduling*** of these blocks to enhance prefetch throughput
- ***Parallelized execution:*** *overlapping* application execution with disk prefetching

Overall Architecture of Paralfetch

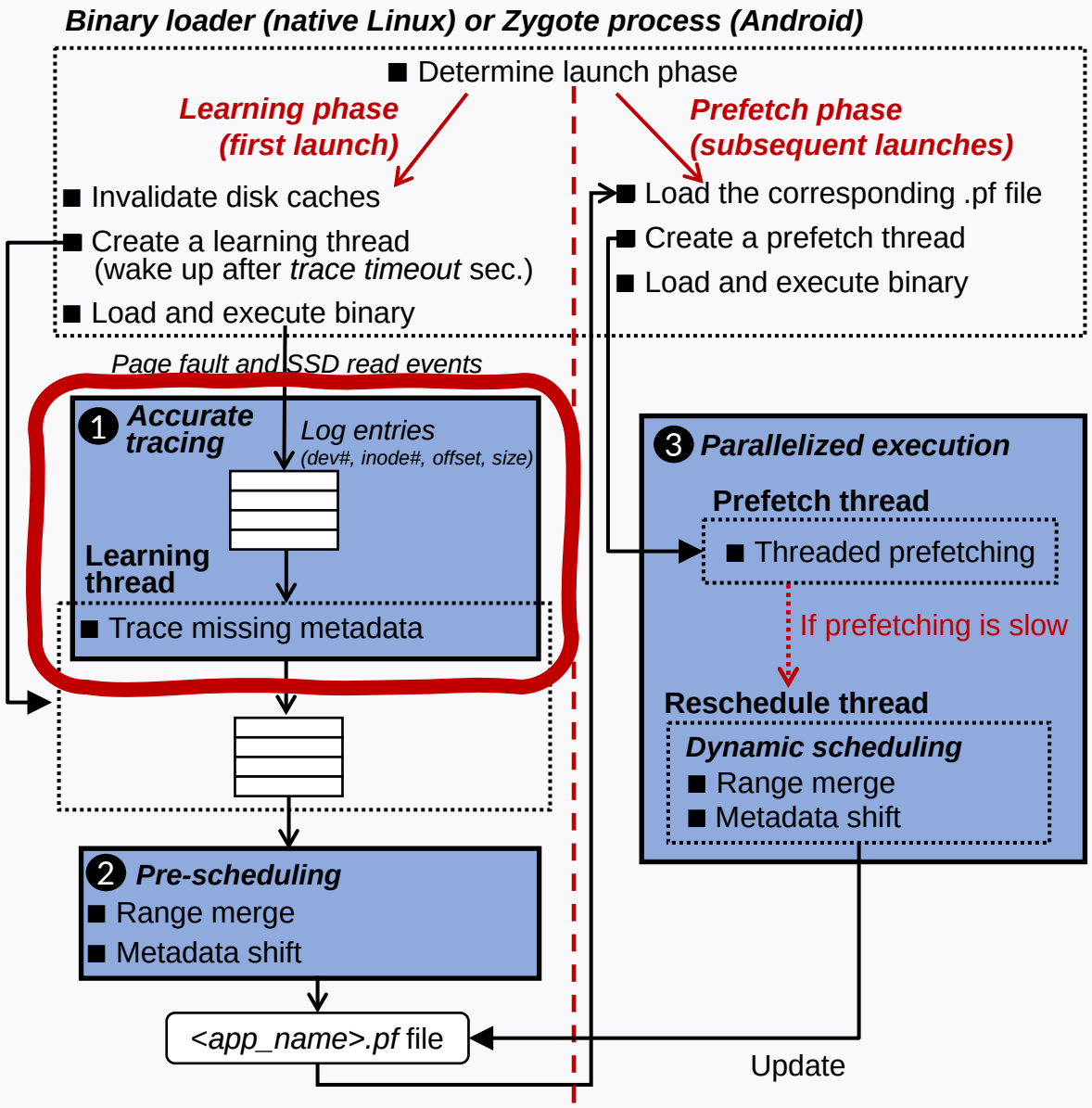


gimp-2.10.pf

dev	inode	start	length
0x800003	0	20972080	4096
0x800003	0	1175	4096
0x800003	0	20981216	4096
0x800003	0	20972896	81920
0x800003	0	20972917	49152
0x800003	0	20972916	4096
0x800003	0	20971567	4096
0x800003	0	103292960	4096
0x800003	5251334	0	50
0x800003	0	103293270	4096
0x800003	0	71827492	4096
0x800003	0	71860378	4096
0x800003	0	103284828	20480
0x800003	0	103284827	4096
0x800003	5251334	2402	39
0x800003	0	20981212	4096
0x800003	5243125	43	17
0x800003	0	20972032	53248
0x800003	0	20972046	77824
0x800003	5251334	50	68
0x800003	0	20972045	4096
0x800003	0	20981208	4096
0x800003	17956944	2954	559
0x800003	0	20981226	4096
0x800003	0	20971872	45056
0x800003	25822139	0	51
0x800003	0	20971883	4096

Buffer-cached prefetch entry: {dev #, inode # (0), start block number, length (byte)}
Page-cached prefetch entry: {dev #, inode #, in-file offset (page), length (page)}

Overall Architecture of Paralfetch



gimp-2.10.pf

dev	inode	start	length
0x800003	0	20972080	4096
0x800003	0	1175	4096
0x800003	0	20981216	4096
0x800003	0	20972896	81920
0x800003	0	20972917	49152
0x800003	0	20972916	4096
0x800003	0	20971567	4096
0x800003	0	103292960	4096
0x800003	5251334	0	50
0x800003	0	103293270	4096
0x800003	0	71827492	4096
0x800003	0	71860378	4096
0x800003	0	103284828	20480
0x800003	0	103284827	4096
0x800003	5251334	2402	39
0x800003	0	20981212	4096
0x800003	5243125	43	17
0x800003	0	20972032	53248
0x800003	0	20972046	77824
0x800003	5251334	50	68
0x800003	0	20972045	4096
0x800003	0	20981208	4096
0x800003	17956944	2954	559
0x800003	0	20981226	4096
0x800003	0	20971872	45056
0x800003	25822139	0	51
0x800003	0	20971883	4096

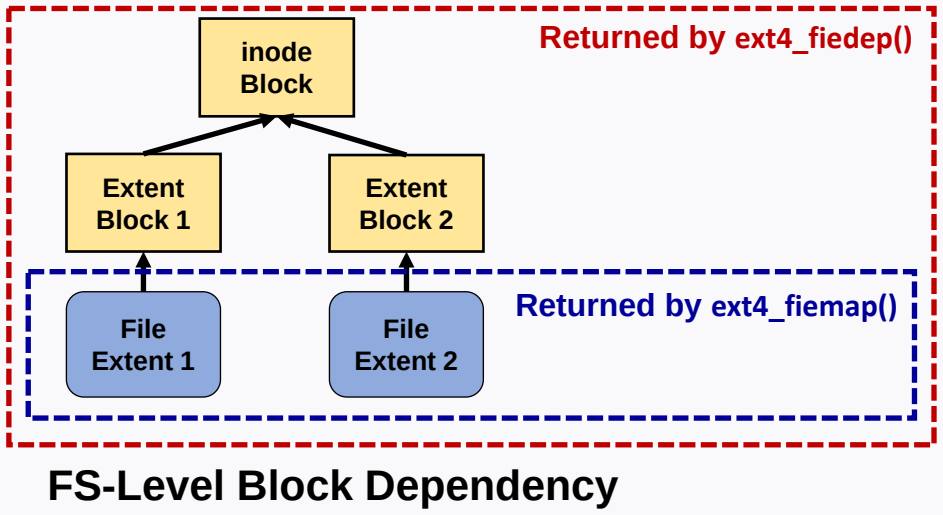
Buffer-cached prefetch entry: {dev #, inode # (0), start block number, length (byte)}
Page-cached prefetch entry: {dev #, inode #, in-file offset (page), length (page)}

Accurate Tracing

- Accurate tracing is essential for better launch performance
- Unfortunately, disk cache invalidation is not perfect!
  - Page cache pages in “dirty, under-writeback, mapped into page tables” states are not invalidated
  - Slab objects (for caching metadata) with “reference count > 0” are not invalidated
- Solutions
  - ***Syncing before I/O tracing*** to flush dirty, under-writeback blocks
  - ***Page fault monitoring*** to detect missing page cache blocks (for regular files)
  - ***Finding missing metadata blocks***: a file system-level dependency check to identify launch-related metadata blocks (*i.e.*, *inode* and *extent* blocks) that have not been traced but have dependency on traced data blocks

Accurate Tracing

- ext4_fiemap function is used to get file extent mappings (lba extents for file block range)
- ext4_fiemap naturally accesses associated metadata blocks
- Unlike ext4_fiemap, ext4_fiedep returns not only file extent mappings but also their associated metadata blocks



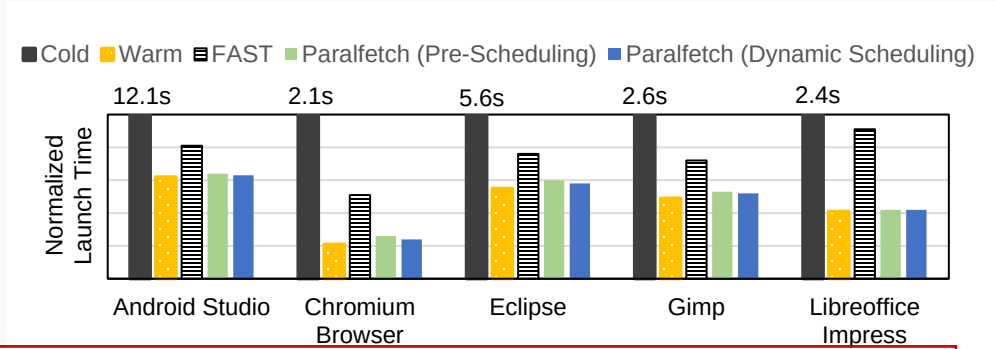
FS-Level Block Dependency

ext4_fiedep is called for each log entry for regular files

Accurate Tracing

- Comparison of Paralfetch and FAST ’11 launch times on a laptop
  - EXT3 file system is used
  - Tracing of each application is performed when LibreOffice Writer is running

Intel Core i5-8265 CPU
16 GB of RAM
Samsung 860 QVO 1TB QLC SSD



	Android Studio	Chromium Browser	Eclipse	Gimp	Libreoffice Impress
Missed amount of blocks when Libreoffice Writer is running in the background	*10.0 MB*	*24.1 MB*	*28.9 MB*	*13.4 MB*	*56.1 MB*
(Traced amount of blocks in a system-cold scenario)	*(206.3 MB)*	*(134.7 MB)*	*(72.0 MB)*	*(42.8 MB)*	*(86.7 MB)*

Results using *FAST*

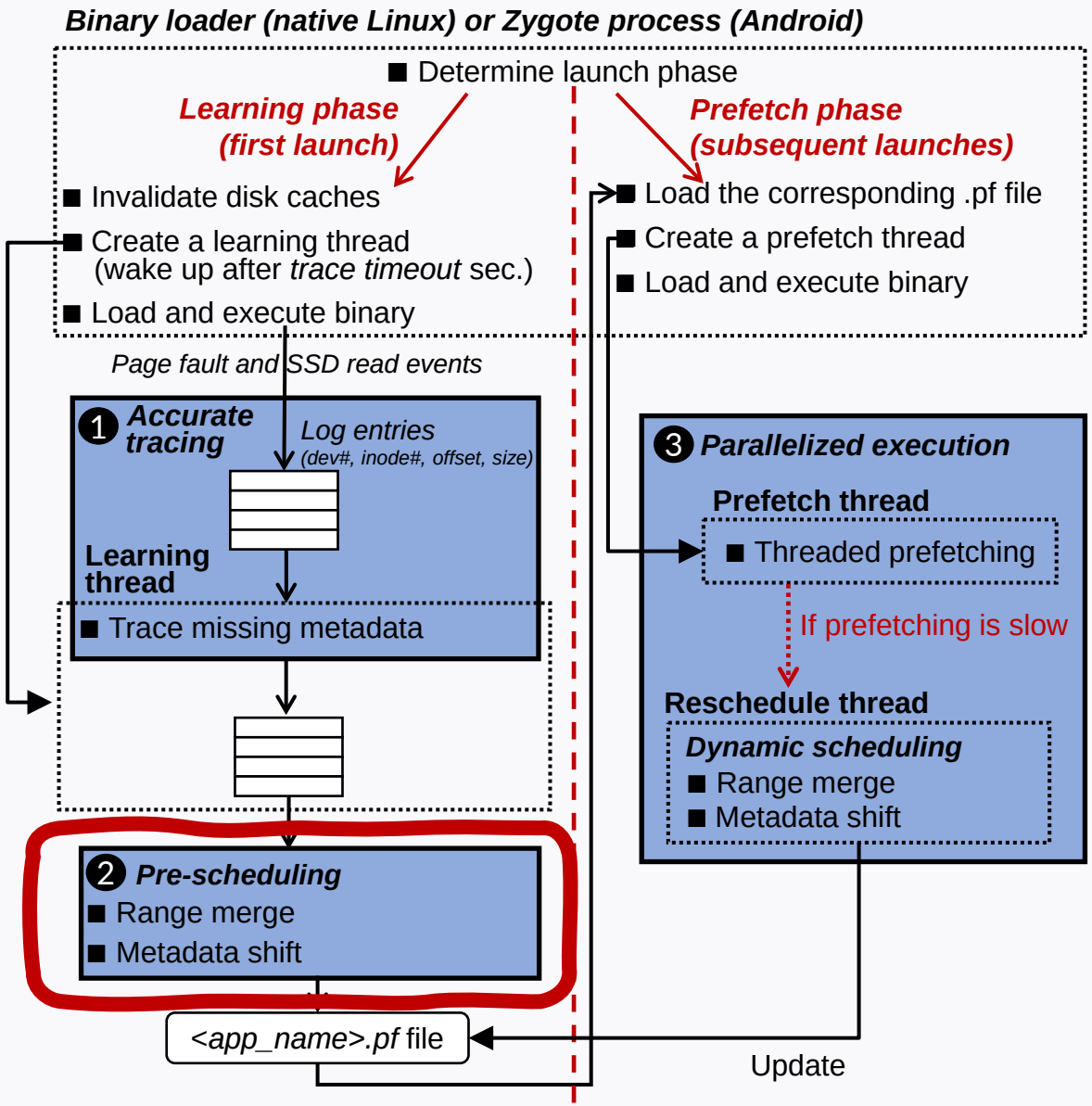
Accurate Tracing

Missing metadata detection

Still misses a few

	Application	Read requests traced by Paralfetch: Metadata accesses (total size in KB)	Read requests traced by Paralfetch: File data accesses (total size in KB)	Number of missing metadata blocks detected	Number of accessed files: regular files	Number of accessed files: mmaped files	Number of missing I/Os
Ubuntu Linux (Laptop PC)	Android Studio	1,330 (6,844)	3,845 (197,932)	58	954	10	38
	Chromium Browser	612 (3,048)	1,135 (130,728)	37	629	108	34
	Eclipse	565 (3,348)	1,669 (67,256)	28	744	328	49
	GIMP	489 (2,620)	1,026 (38,512)	20	975	474	28
	LibreOffice Impress	590 (2,900)	706 (83,004)	37	438	232	32
	LibreOffice Writer	552 (2,800)	729 (83,824)	25	476	227	33
	Okular	1,093 (5,720)	426 (23,640)	41	349	238	36
	Scribus	840 (5,984)	1,560 (141,056)	35	1,230	682	21
	VLC Player	682 (5,420)	444 (20,192)	41	375	104	32
	Xilinx ISE	573 (3,024)	1,028 (176,504)	42	657	273	33
Raspbian OS (Raspberry Pi 3)	Chromium Browser	496 (1,984)	2,017 (138,600)	40	473	68	41
	Frozen Bubble	605 (2,420)	3,769 (32,992)	25	3,425	26	12
	GIMP	618 (2,472)	1,863 (46,664)	38	991	296	47
	LibreOffice Writer	596 (2,384)	911 (35,164)	33	395	154	36
	Scratch 2	332 (1,328)	839 (48,580)	40	294	73	19
	Xpdf	127 (508)	169 (7,236)	15	75	21	11
	0 A.D.	206 (509)	669 (86,272)	19	162	139	21
Android 8.0 (Google Pixel XL)	Asphalt 8	131 (988)	838 (217,240)	49	179	N/A	11
	Dragon Quest 8	95 (852)	4,339 (333,812)	46	335	N/A	12
	FIFA 16 UT	76 (772)	805 (166,120)	39	265	N/A	47
	GTA SA	104 (560)	377 (82,928)	41	95	N/A	36
	Truck Pro	96 (792)	1,792 (115,732)	41	175	N/A	19
	Devil May Cry	237 (1,728)	1,904 (316,004)	45	407	N/A	19
	The War of Mine	127 (696)	517 (128,300)	43	101	N/A	11

Overall Architecture of Paralfetch



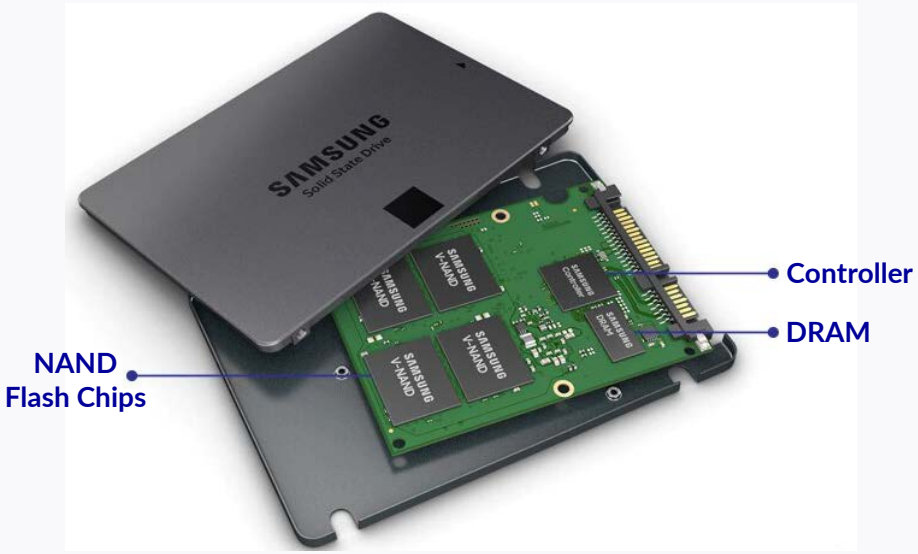
gimp-2.10.pf

dev	inode	start	length
0x800003	0	20972080	4096
0x800003	0	1175	4096
0x800003	0	20981216	4096
0x800003	0	20972896	81920
0x800003	0	20972917	49152
0x800003	0	20972916	4096
0x800003	0	20971567	4096
0x800003	0	103292960	4096
0x800003	5251334	0	50
0x800003	0	103293270	4096
0x800003	0	71827492	4096
0x800003	0	71860378	4096
0x800003	0	103284828	20480
0x800003	0	103284827	4096
0x800003	5251334	2402	39
0x800003	0	20981212	4096
0x800003	5243125	43	17
0x800003	0	20972032	53248
0x800003	0	20972046	77824
0x800003	5251334	50	68
0x800003	0	20972045	4096
0x800003	0	20981208	4096
0x800003	17956944	2954	559
0x800003	0	20981226	4096
0x800003	0	20971872	45056
0x800003	25822139	0	51
0x800003	0	20971883	4096

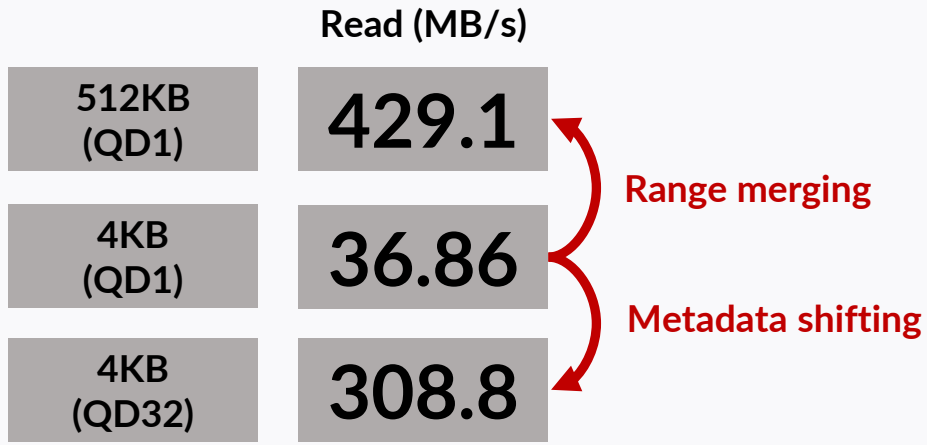
Buffer-cached prefetch entry: {dev #, inode # (0), start block number, length (byte)}
Page-cached prefetch entry: {dev #, inode #, in-file offset (page), length (page)}

Pre-scheduling

- Slow prefetching is often the bottleneck in a launch
- Solutions: exploiting *internal parallelism* of SSDs
  - ***Metadata shifting*** to increase the number of concurrently issued I/Os
  - ***Range merging*** to merge LBA-consecutive I/Os nearby into a single large one

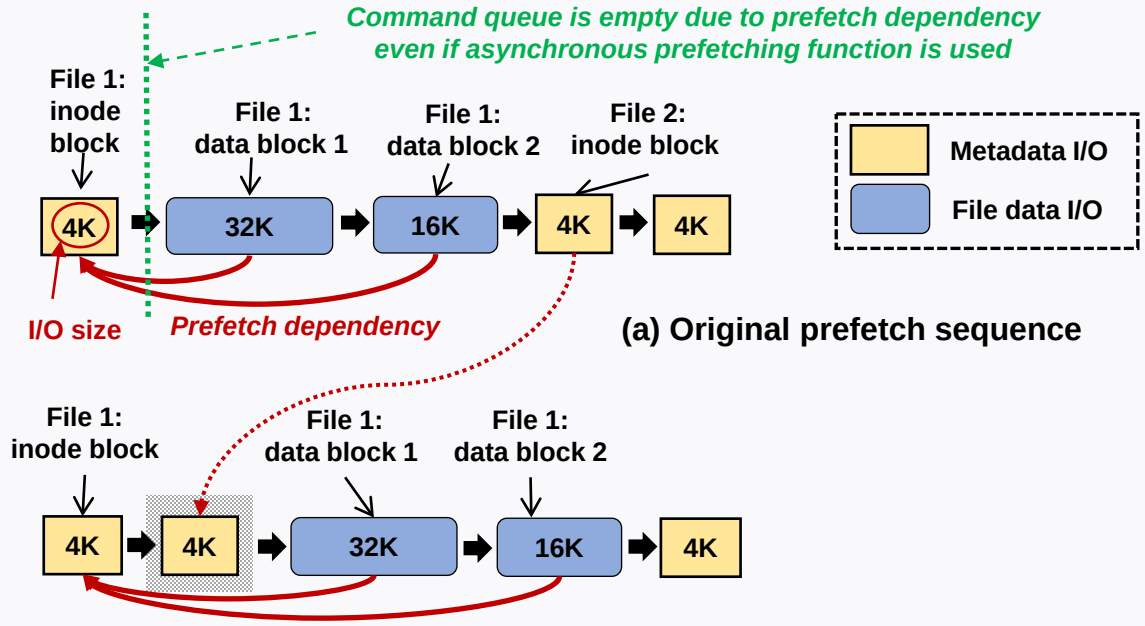


SSD Internals



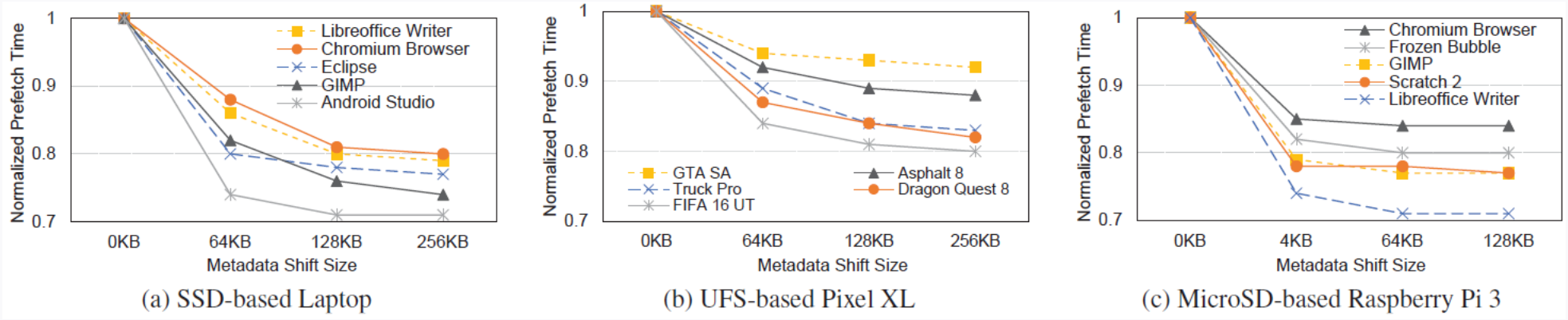
CrystalDiskMark
(Samsung 860 QVO 1TB)

Pre-scheduling: Metadata Shifting

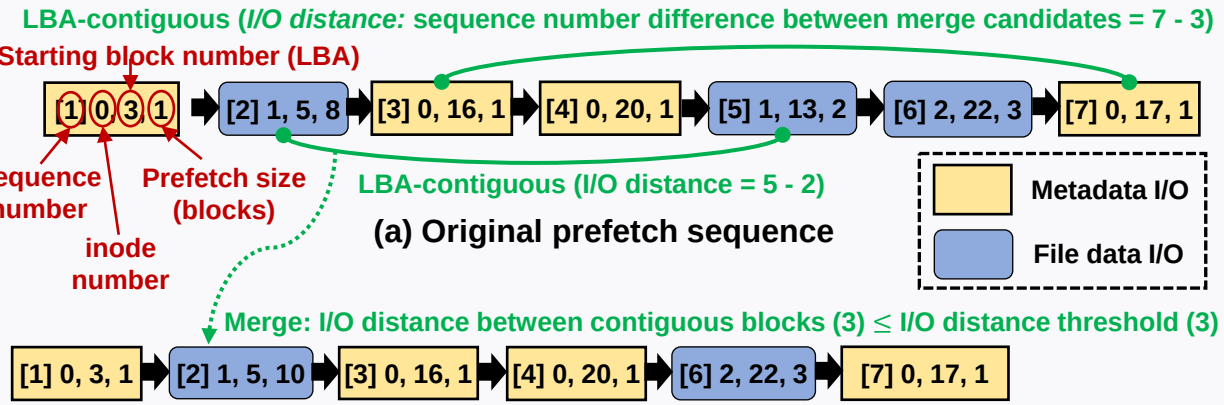


- Metadata shifting to boost the number of I/O requests in the command queue
- *Shift size* controls the size of metadata blocks to be left-shifted.

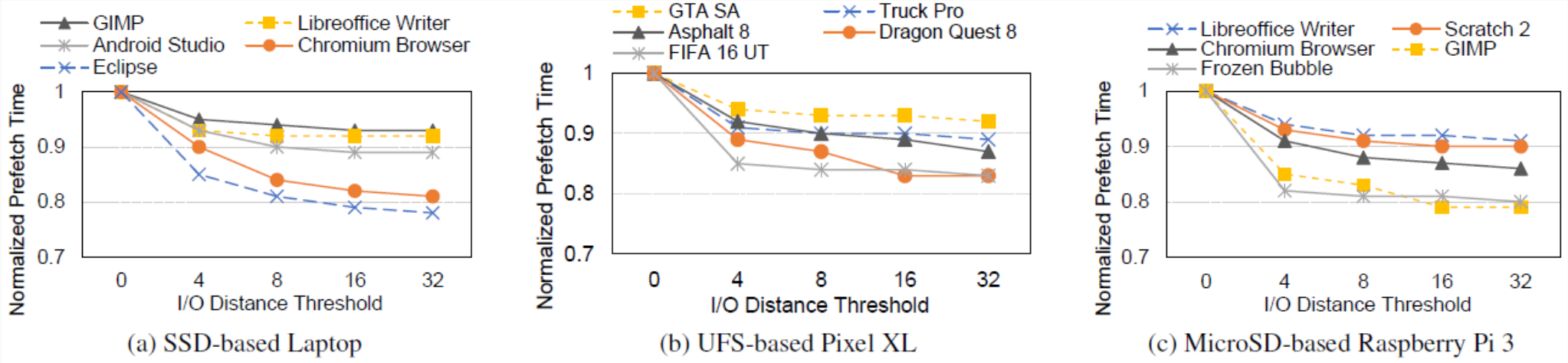
There are no prefetching-level dependencies among buffer-cached (metadata) blocks



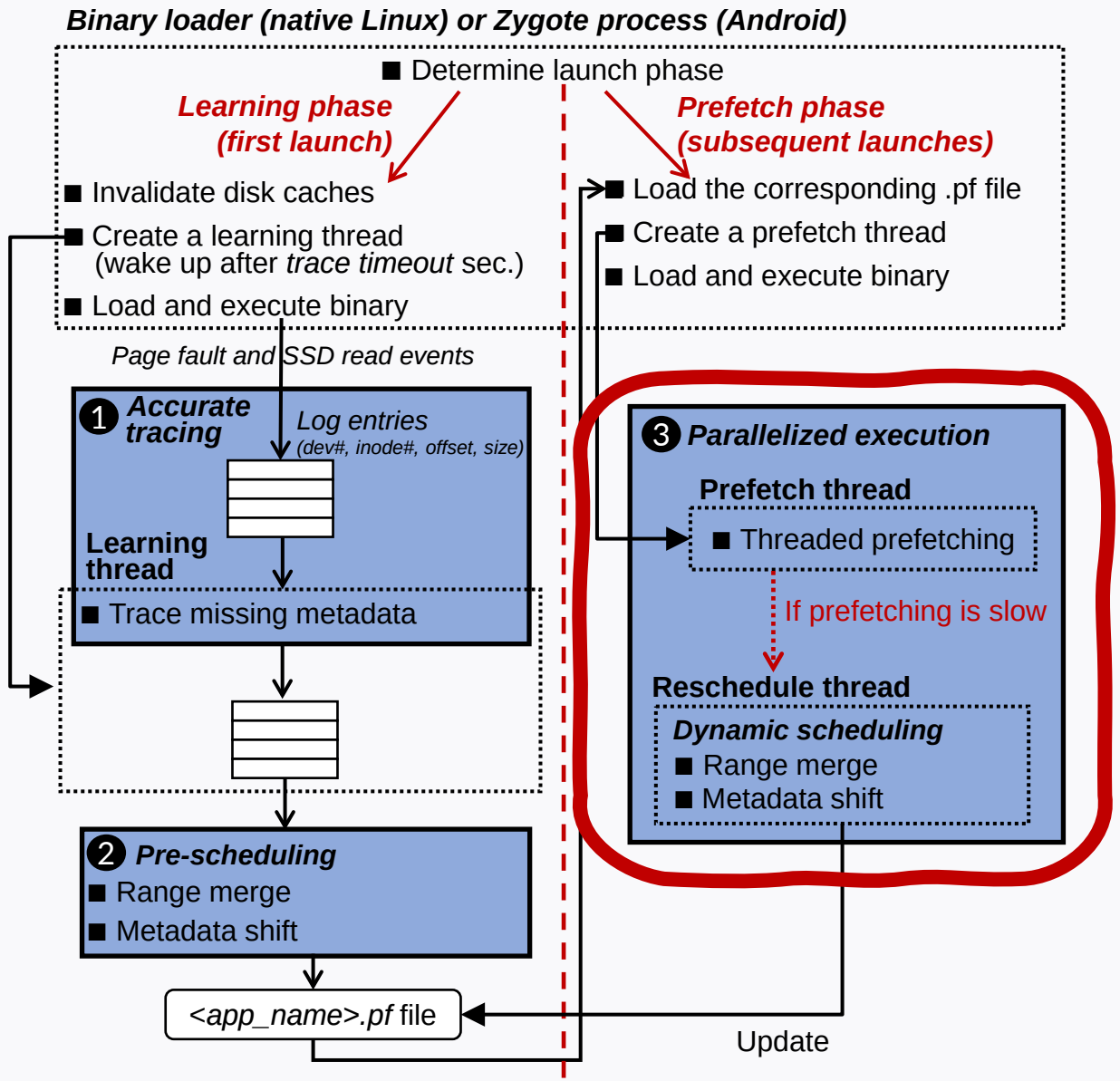
Pre-scheduling: Range Merging



- Range merging to merge I/O requests nearby into a large one, improving I/O throughput



Overall Architecture of Paralfetch



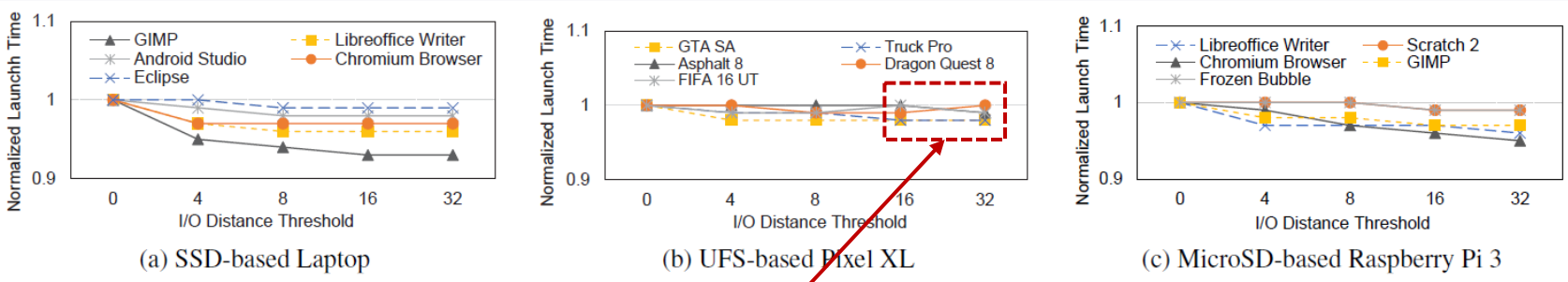
gimp-2.10.pf

dev	inode	start	length
0x800003	0	20972080	4096
0x800003	0	1175	4096
0x800003	0	20981216	4096
0x800003	0	20972896	81920
0x800003	0	20972917	49152
0x800003	0	20972916	4096
0x800003	0	20971567	4096
0x800003	0	103292960	4096
0x800003	5251334	0	50
0x800003	0	103293270	4096
0x800003	0	71827492	4096
0x800003	0	71860378	4096
0x800003	0	103284828	20480
0x800003	0	103284827	4096
0x800003	5251334	2402	39
0x800003	0	20981212	4096
0x800003	5243125	43	17
0x800003	0	20972032	53248
0x800003	0	20972046	77824
0x800003	5251334	50	68
0x800003	0	20972045	4096
0x800003	0	20981208	4096
0x800003	17956944	2954	559
0x800003	0	20981226	4096
0x800003	0	20971872	45056
0x800003	25822139	0	51
0x800003	0	20971883	4096

Buffer-cached prefetch entry: {dev #, inode # (0), start block number, length (byte)}
Page-cached prefetch entry: {dev #, inode #, in-file offset (page), length (page)}

Overlapping Application Execution with Prefetching

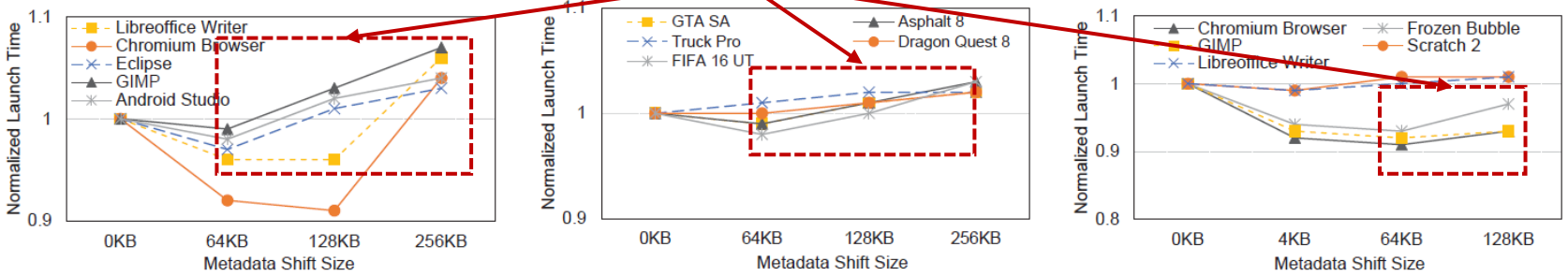
- The parallel utilization of CPU and SSD is key to the reduction of application launch time
- However, aggressive prefetching can negatively affect a launch performance
  - SSD-level (i.e., controller) I/O reordering
  - I/O contentions between the prefetch thread and the launching app



(a) SSD-based Laptop (b) UFS-based Pixel XL (c) MicroSD-based Raspberry Pi 3

Figure 7: Normalized launch times with varying I/O distance thresholds.

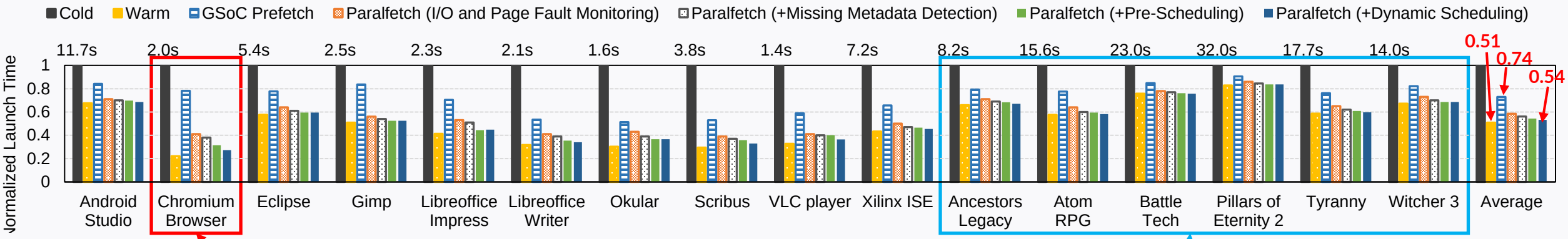
Aggressive prefetching slows down an application launch



(a) SSD-based Laptop (b) UFS-based Pixel XL (c) MicroSD-based Raspberry Pi 3

Figure 8: Normalized launch times for different metadata shift sizes.

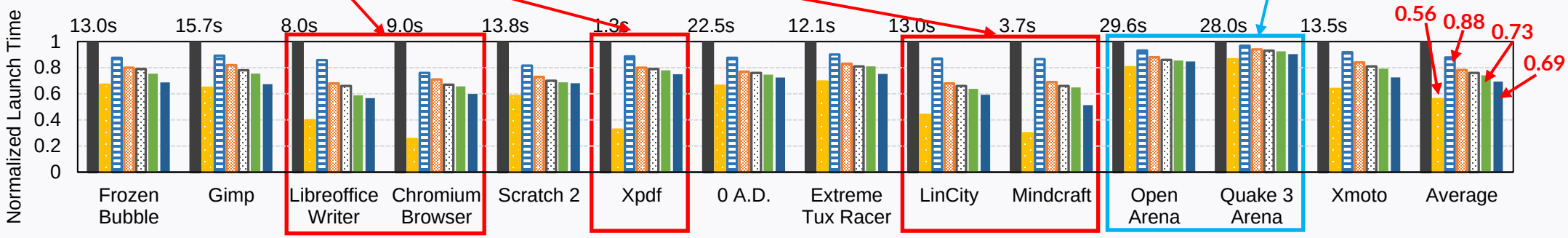
Evaluation



Laptop PC (Intel i5-8265, Samsung 860 QVO 1TB QLC SSD)

Even with aggressive dynamic scheduling, it is challenging to achieve prefetch time close to warm start time

For CPU-intensive apps, Paralfetch achieves performance close to warm start



Raspberry Pi 3 (Samsung Class-10 16 GB MicroSD)

Google Pixel XL Smartphone results are shown in the paper.

Summary

- Three fundamental challenges in threaded prefetching methods
  - Accurate tracing
    - Missing metadata detection and page fault monitoring
  - Pre-scheduling
    - Range merging and metadata shifting
  - Overlapping application execution with disk prefetching
    - Optimization strength control
- Implementation and evaluation of Paralfetch
  - Negligible overhead in terms of CPU, memory, and storage
  - Launch performance close to the warm start scenario in most cases

Appendix

[A-1] Accurate Tracing

- Paralfetch still misses a few I/O reads.
  - Missing metadata detection should fail to detect metadata-only I/O reads
    - lstat() system call
  - Office/IDE tools usually read a user's home directory which changes often.
  - Program or library updates

[A-2] Accurate Tracing

- An application and libraries update often.
- To handle this issue, Paralfetch handles incorrect prefetch entries.
  - A daemon periodically removes obsolete prefetch entries in <*app_name*>.*pf* files.
  - If the regular file of the prefetch entry was not accessed after the <*app_name*>.*pf* file.

[A-3] Overlapping Application Execution with Prefetching

- Solutions
  - To find the best thresholds for the metadata shifting and range merging, Paralfetch gradually increases the level of optimization if prefetching is not effective.

	CQ-disabled SSD	CQ-enabled SSD
Metadata shift (KB)	4	64 proactively and increasing dynamically
Range merge (I/O distance)	8 proactively and increasing dynamically	8

Table 2: **Default configuration for prefetch optimization.**

  - Paralfetch detects a prefetch-bottleneck situation based on the number of context switches made by the launching application during the prefetching period.
    - *i.e.*, the quantity of context switches is above a user-defined threshold (by default, 5% of the number of prefetch entries)

[A-4] Overhead (On a Laptop PC)

- Tracing overhead
  - I/O logging overhead of 136ms for Android Studio
- Pre-scheduling (background job)
  - Between 42ms for VLC player and 153ms for Android Studio (c.f., FAST took 23 seconds to generate the prefetch program for Eclipse)
- Space overhead
  - 672KB to store *<app_name>.pf* files for 16 applications (c.f., 8.2MB with FAST)

[A-5] Paralfetch on HDDs (On a Laptop PC)

- Sorted prefetch + threaded prefetch
  - Pre-scheduling: LBA-sorted prefetching + infill merge
  - Dynamic scheduling: threaded prefetch for late-deadline blocks