SafeFetch:

Practical Double-Fetch Protection with Kernel-Fetch Caching

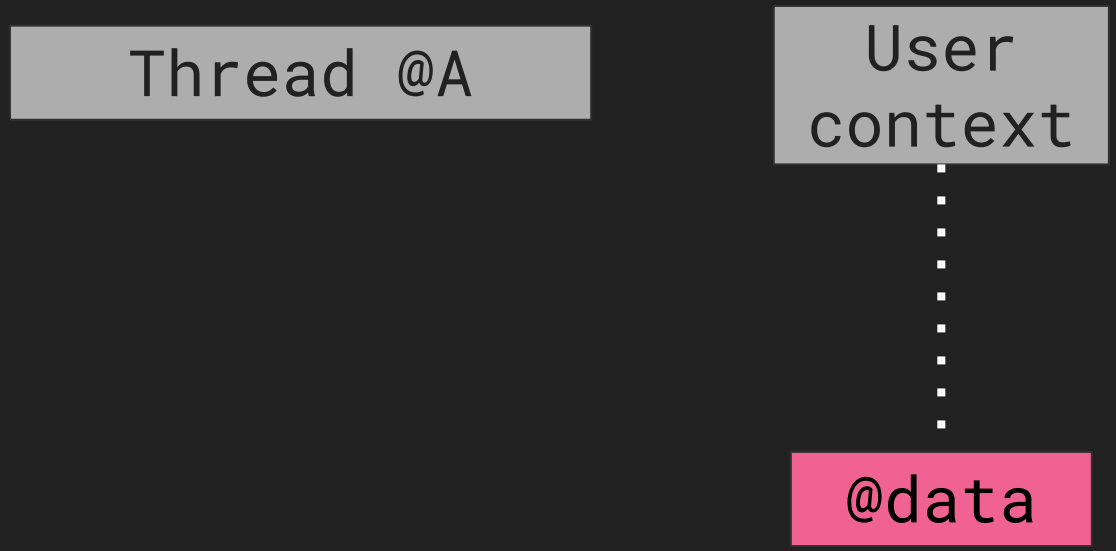
Victor Duta

Mitchel Alonserij Cristiano Giuffrida

Teaser

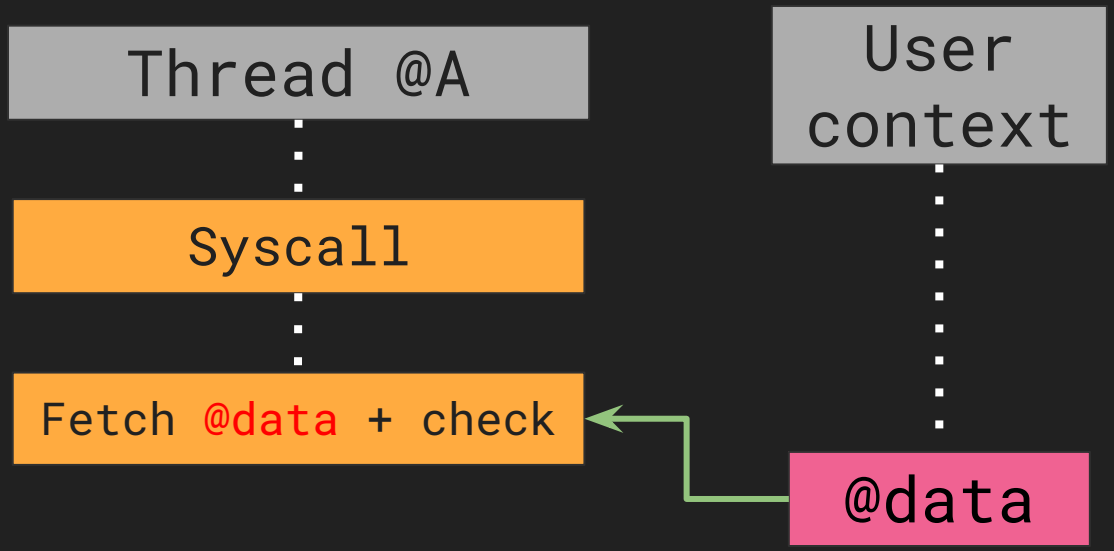
- Kernel double-fetch bugs
- State-of-the-art relies on COW semantics
- SafeFetch ideea => per-syscall caching of user data
- Low performance impact:
    - Less than 5% geomean overhead

Double-Fetch Anatomy

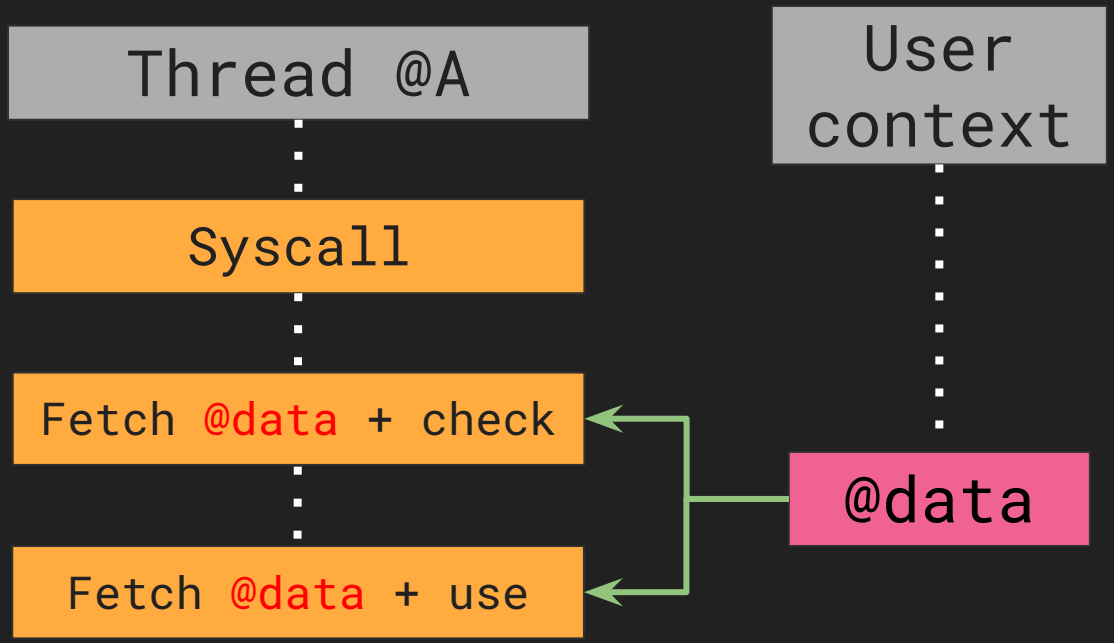


Double-Fetch Anatomy

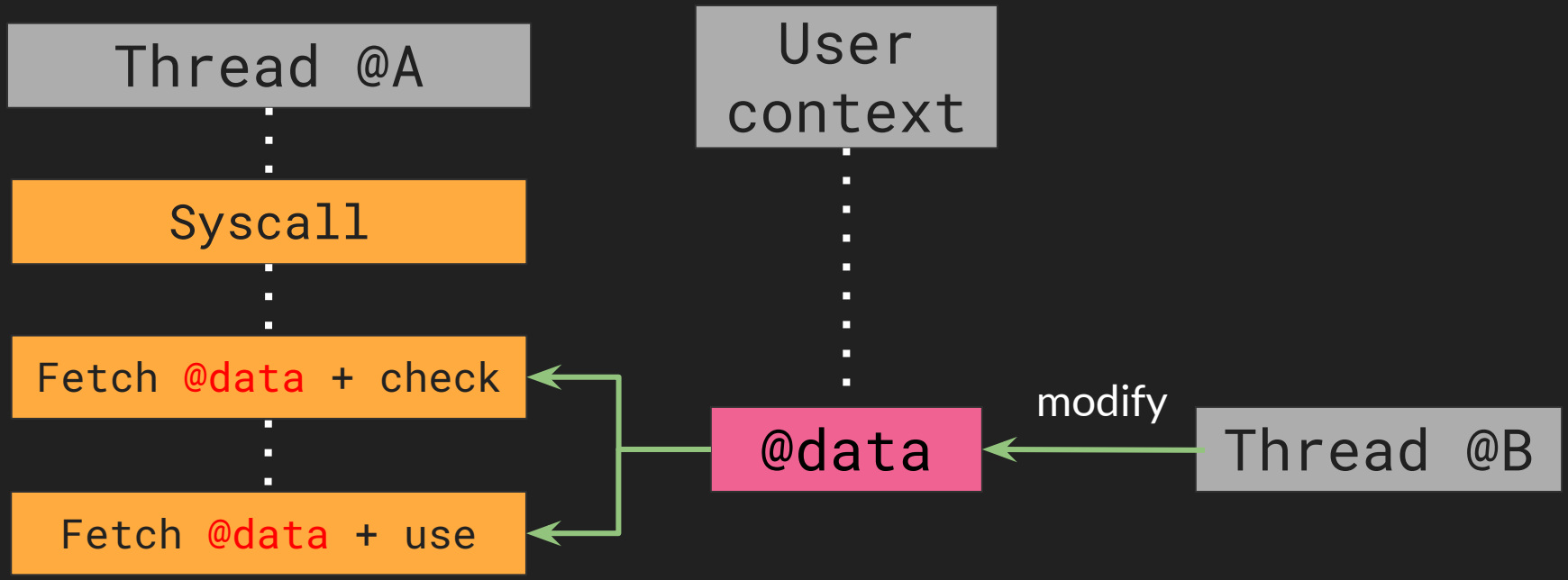
Double-Fetch Anatomy



Double-Fetch Anatomy



Double-Fetch Anatomy



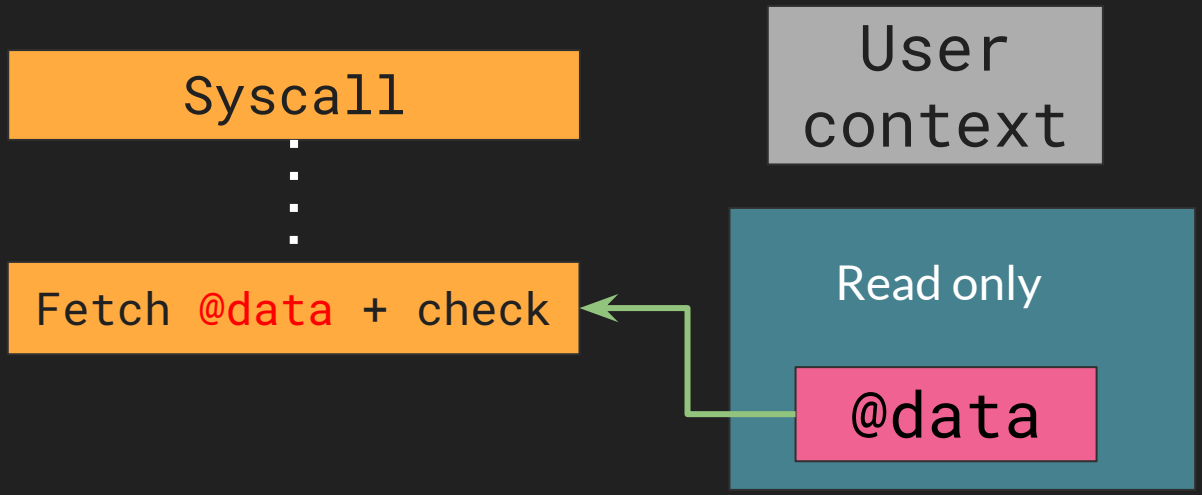
Midas

Syscall

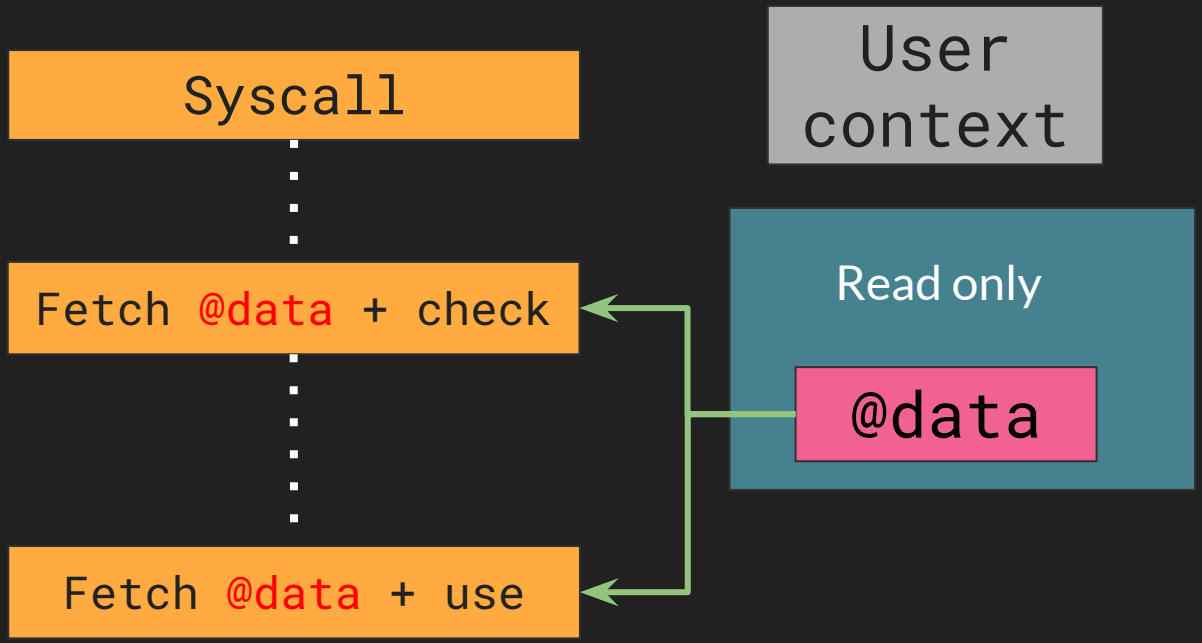
User context

@data

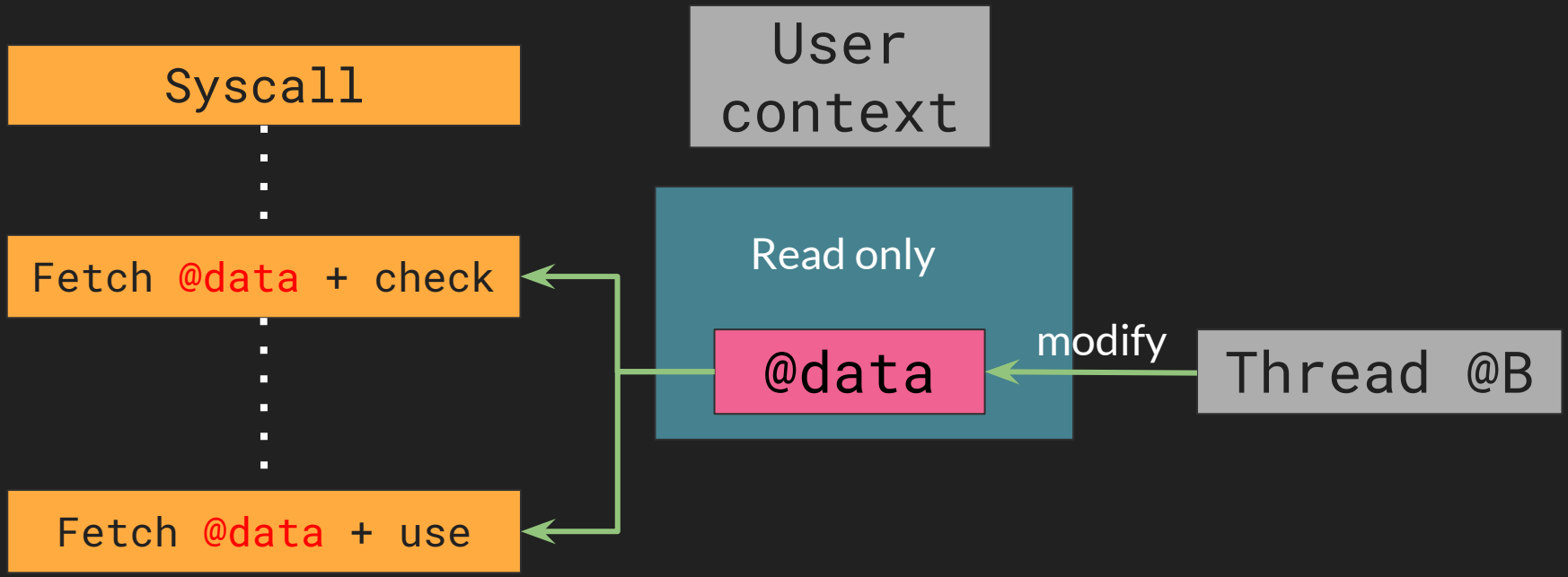
Midas



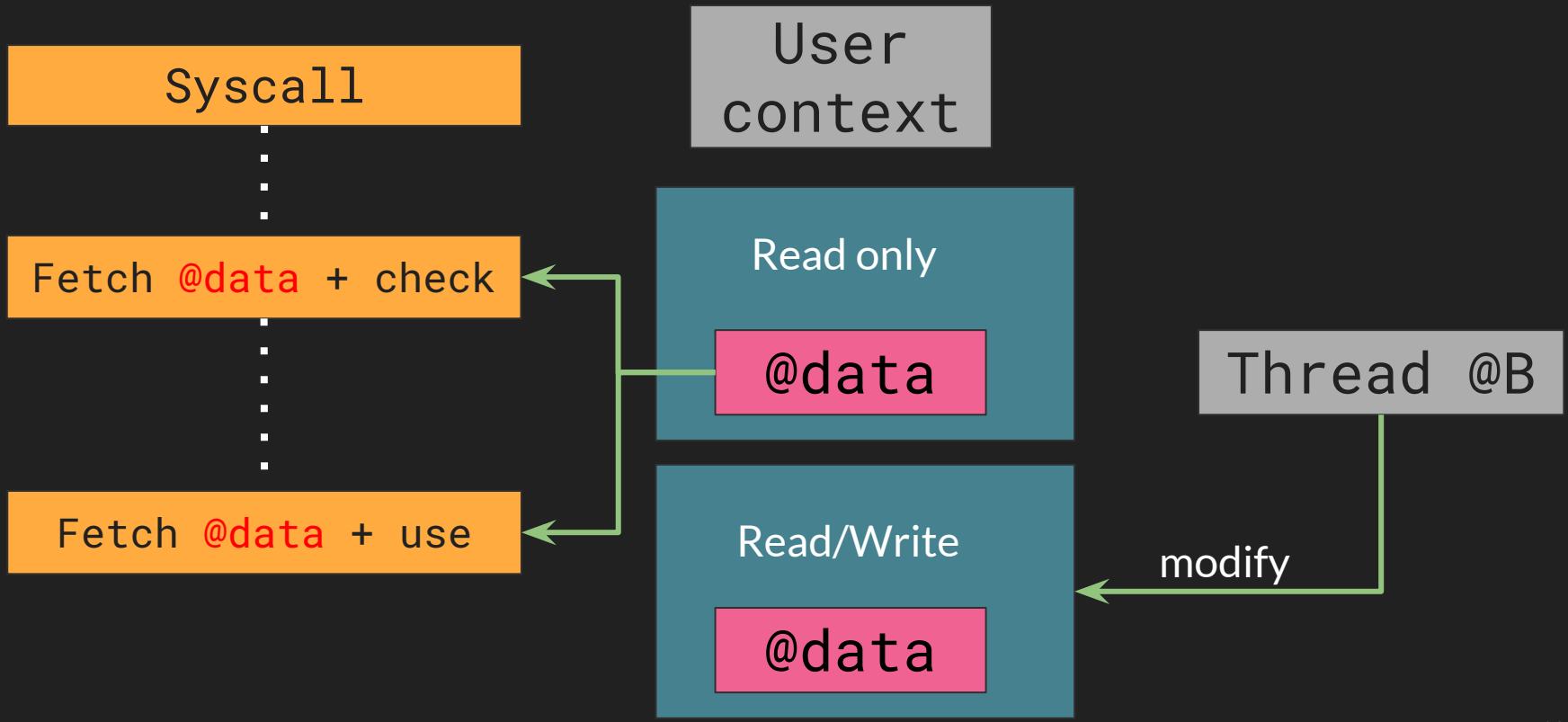
Midas


Syscall
Fetch @data + check
Fetch @data + use
User context
Read only
@data

Midas


Syscall
Fetch @data + check
Fetch @data + use
User context
Read only
@data
modify
Thread @B

Midas


Syscall
Fetch @data + check
Fetch @data + use
User context
Read only
@data
Read/Write
@data
Thread @B
modify

Key Insights

Key Insights

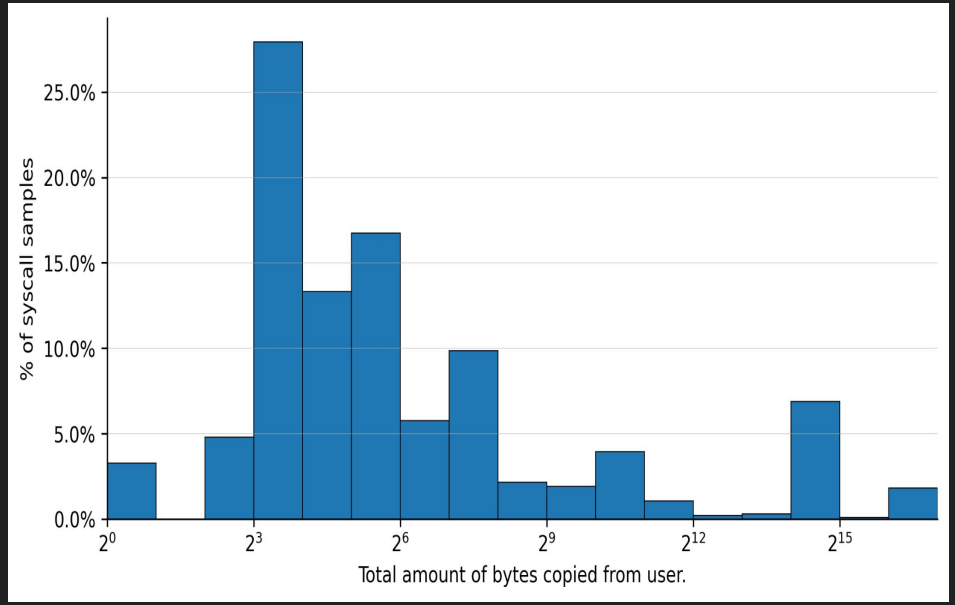
- 60% of syscall executions do not fetch user data

Key Insights

- 60% of syscall executions do not fetch user data
- On average syscalls fetch far less than a page of data

Key Insights

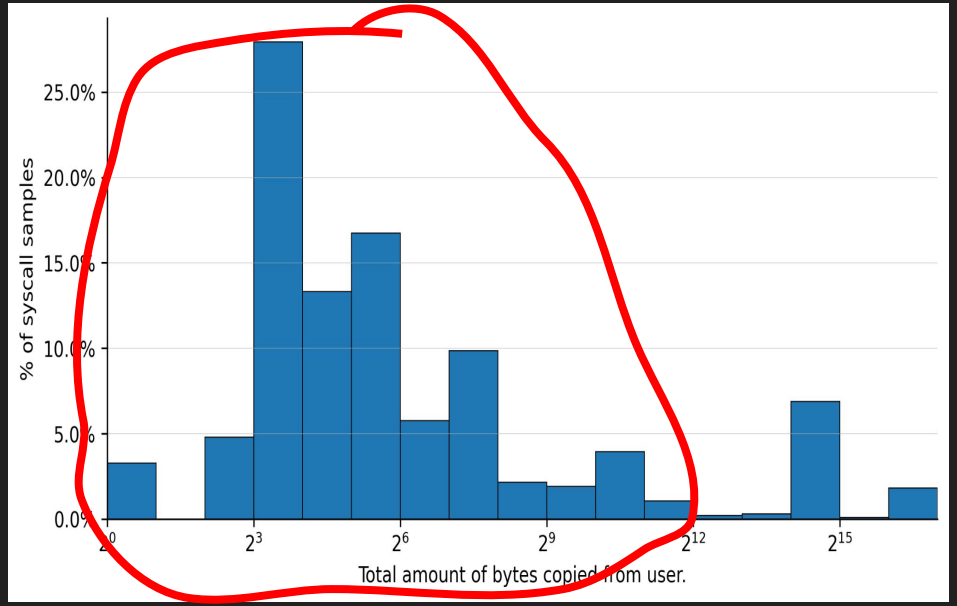
- 60% of syscall executions do not fetch user data
- On average syscalls fetch far less than a page of data



Key Insights

- 60% of syscall executions do not fetch user data
- On average syscalls fetch far less than a page of data

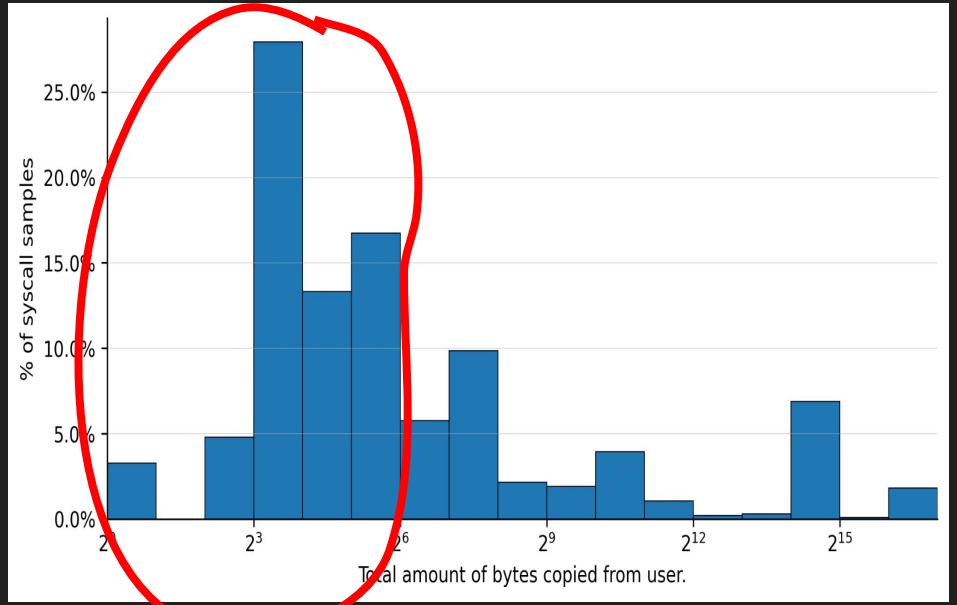
90% of syscalls fetch <= 1 Page of user data



Key Insights

- 60% of syscall executions do not fetch user data
- On average syscalls fetch far less than a page of data

90% of syscalls fetch <= 1 Page of user data

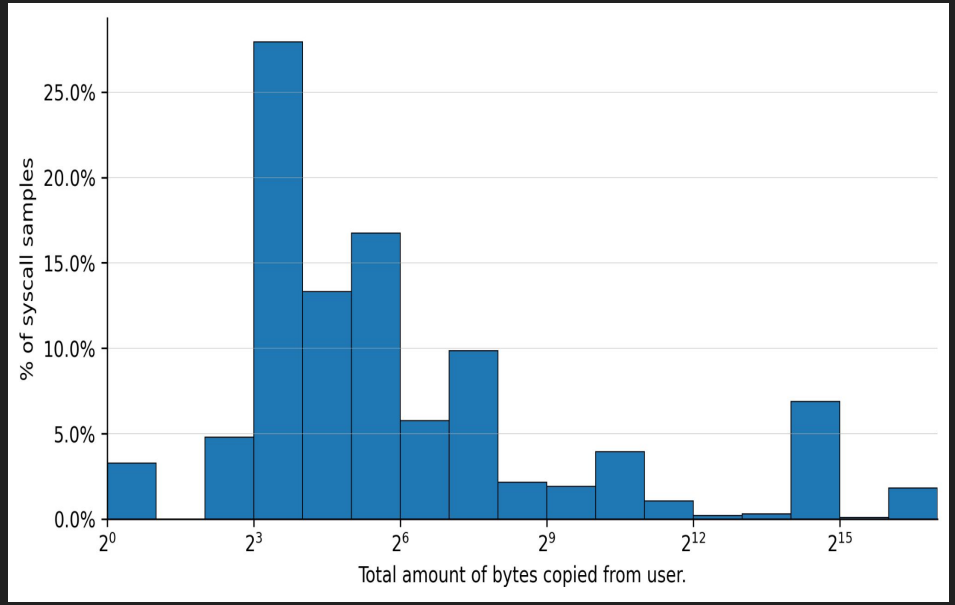


66% of syscalls fetch < 64 bytes of data

Key Insights

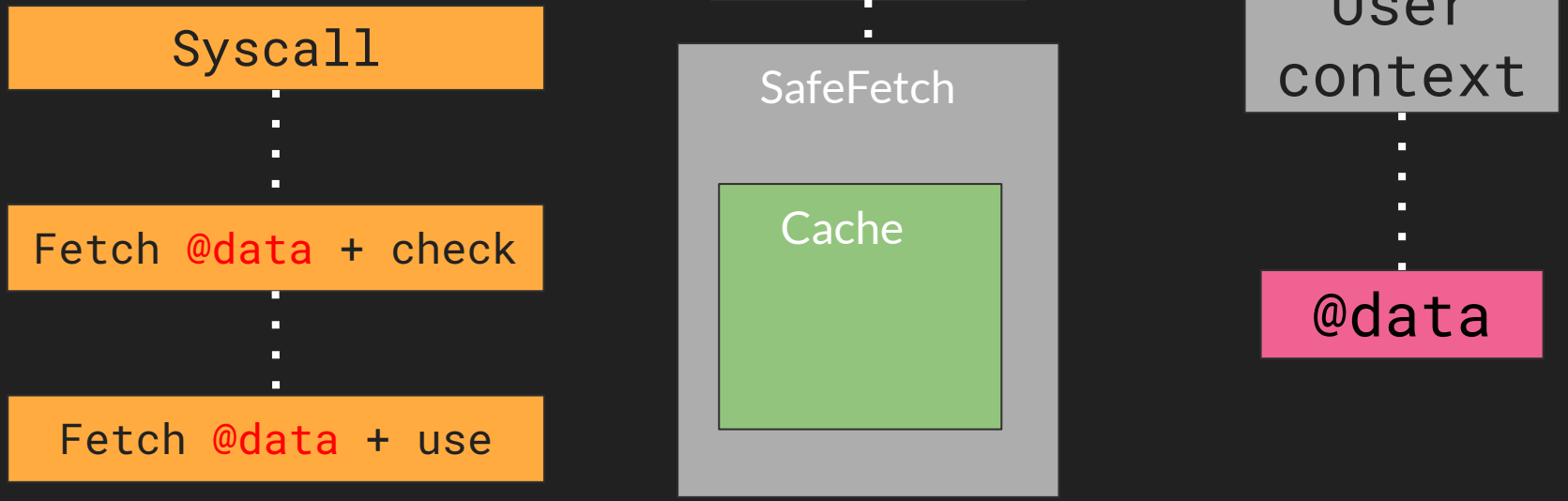
- 60% of syscall executions do not fetch user data
- On average syscalls fetch far less than a page of data
- Caching is more efficient

90% of syscalls fetch <= 1 Page of user data

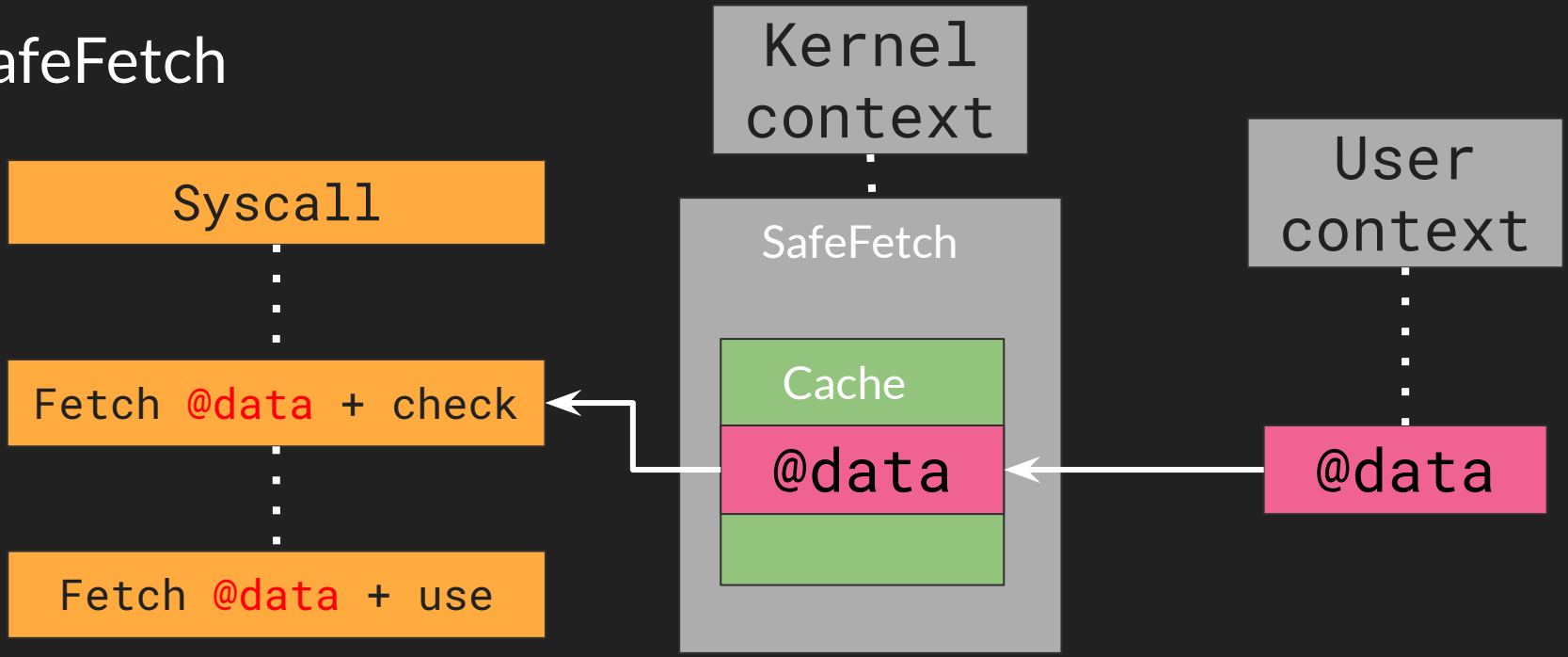


66% of syscalls fetch < 64 bytes of data

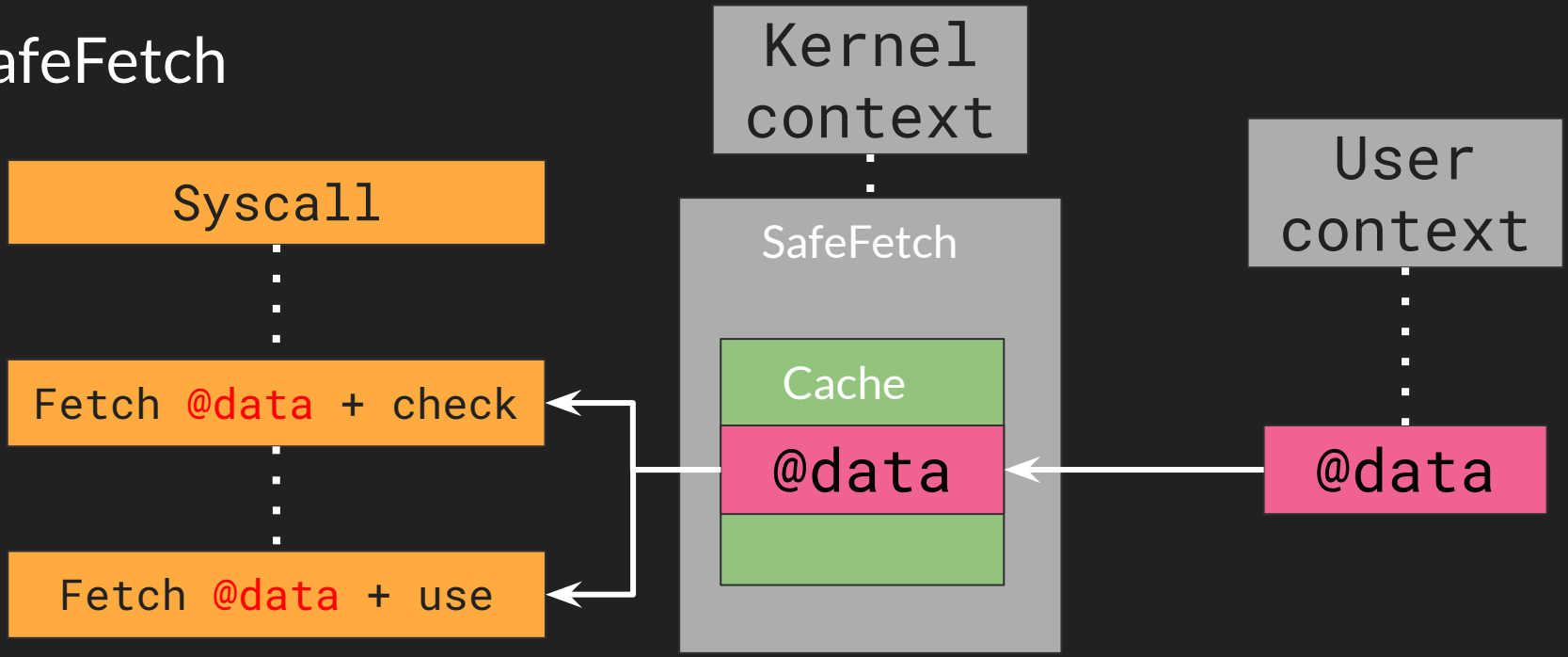
SafeFetch


Syscall
Fetch @data + check
Fetch @data + use
Kernel context
SafeFetch
Cache
User context
@data

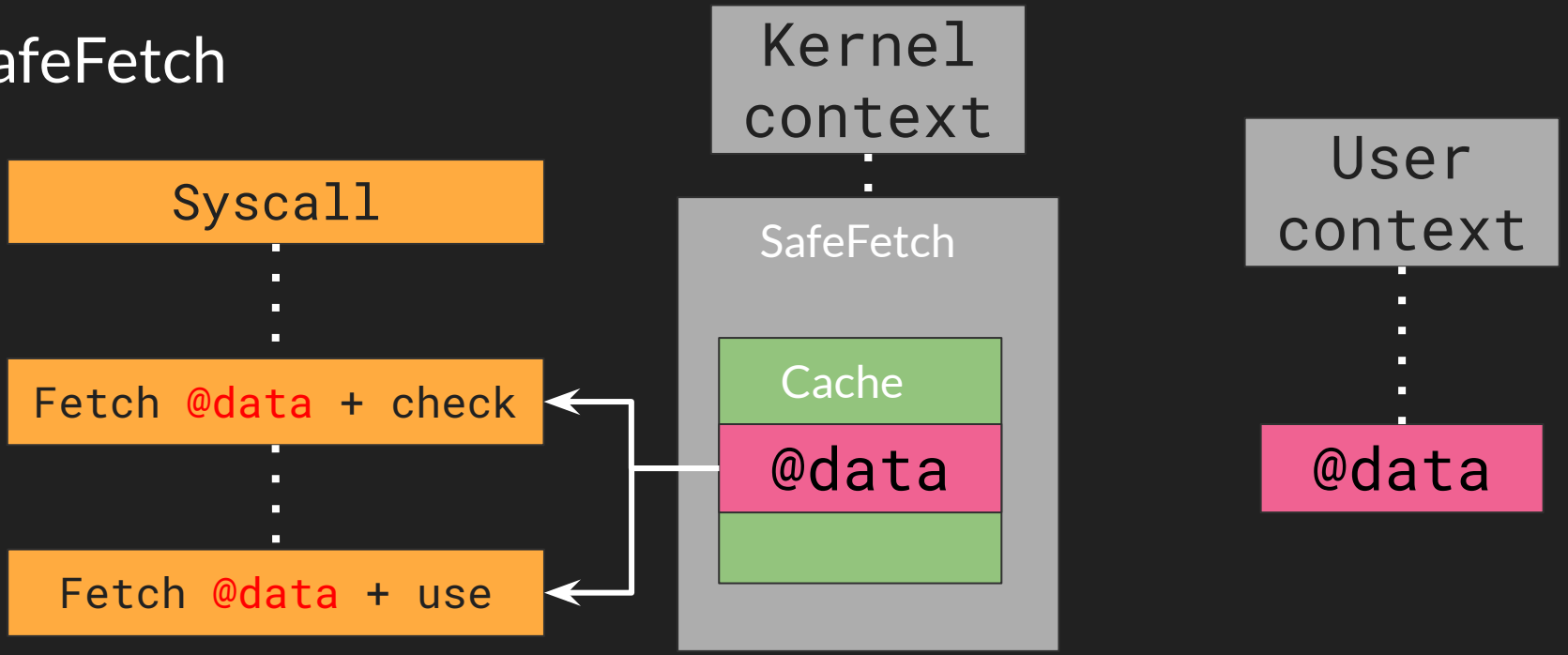
SafeFetch


Kernel context
User context
Syscall
SafeFetch
Fetch @data + check
Cache
@data
@data
Fetch @data + use

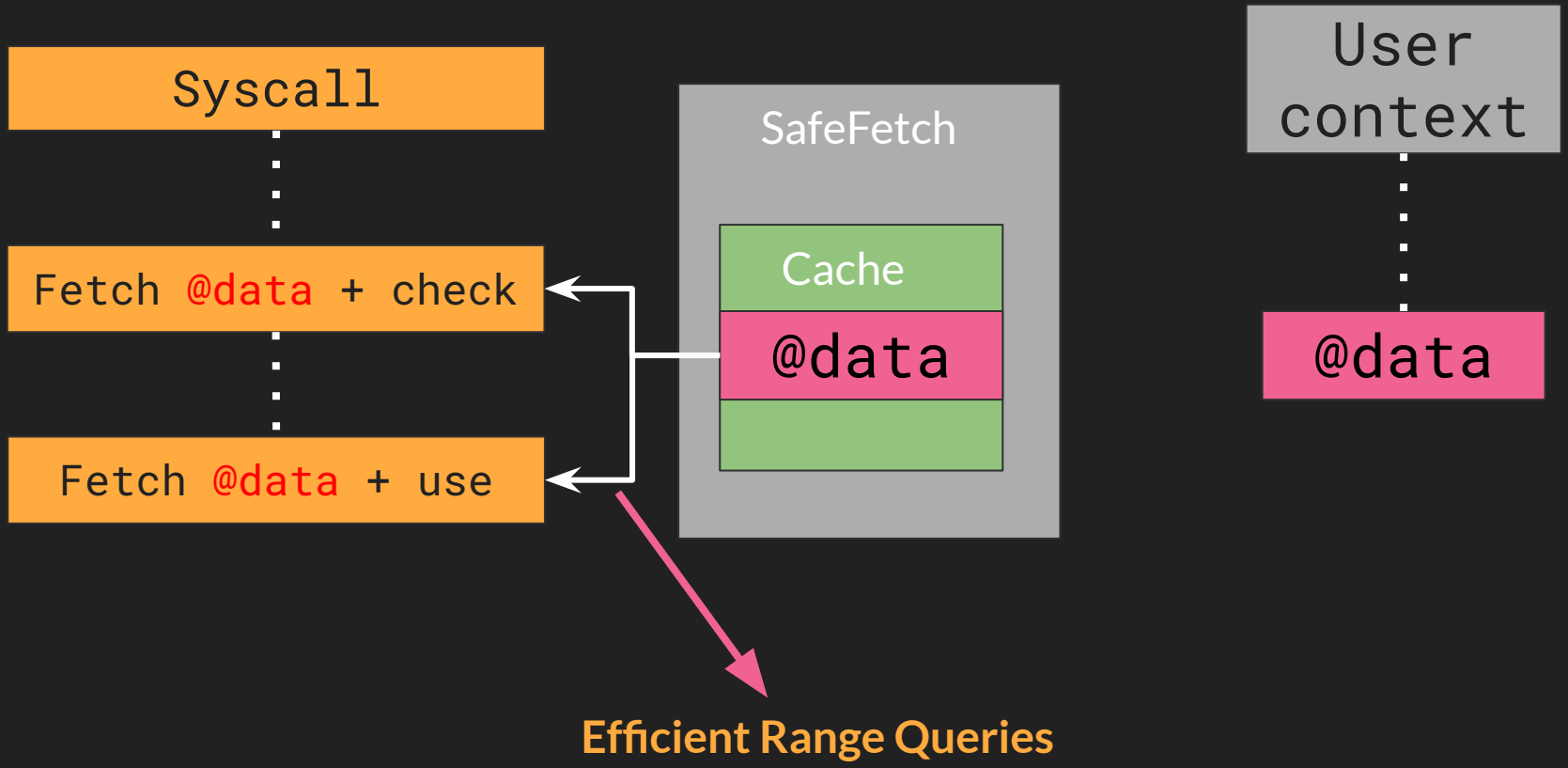
SafeFetch


Kernel context
User context
Syscall
SafeFetch
Fetch @data + check
Cache
@data
@data
Fetch @data + use

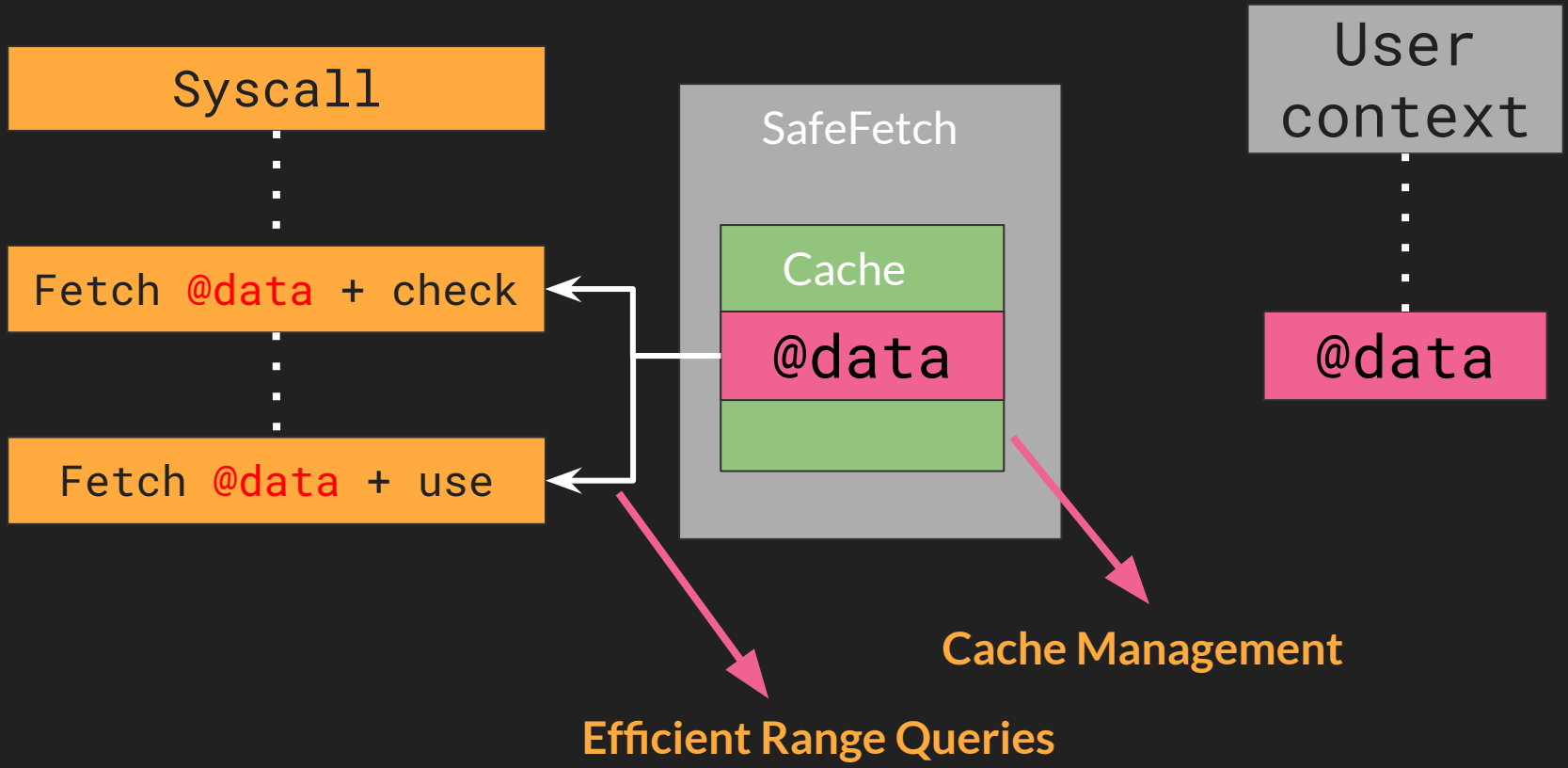
SafeFetch


Kernel context
User context
Syscall
SafeFetch
Fetch @data + check
Cache
@data
@data
Fetch @data + use

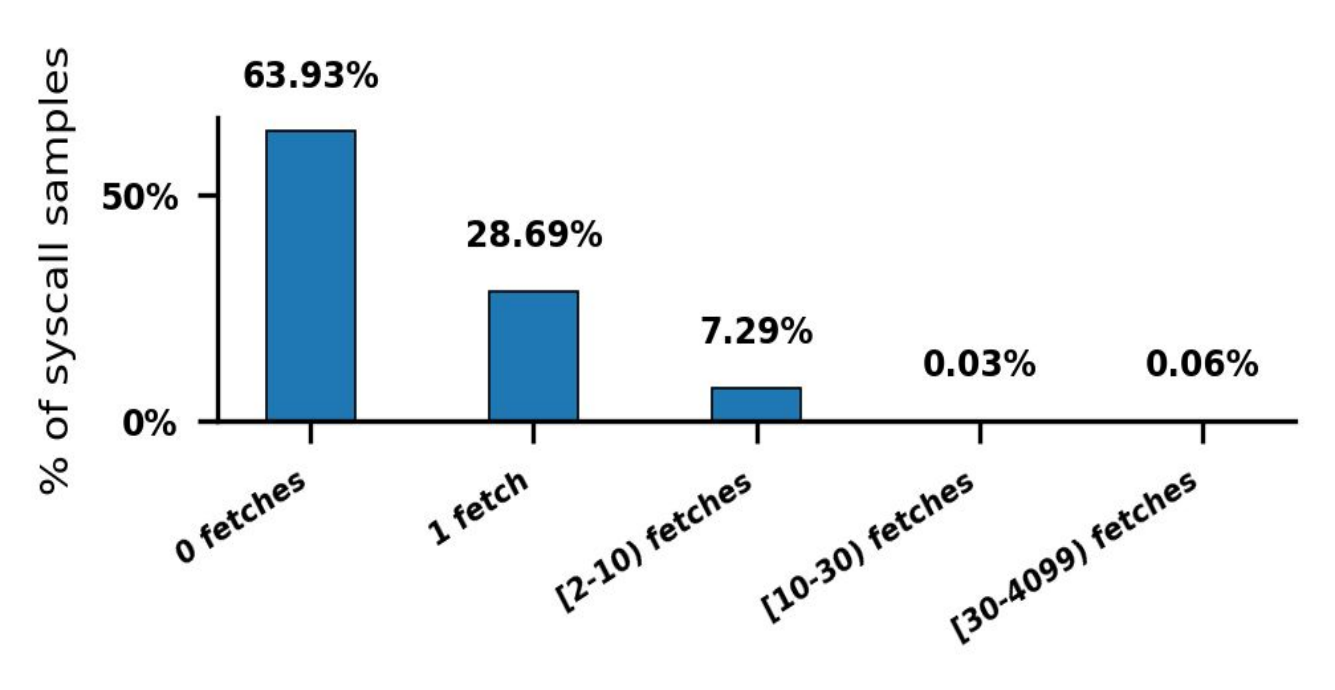
SafeFetch


Syscall
Fetch @data + check
Fetch @data + use
SafeFetch
Cache
@data
User context
@data
Efficient Range Queries

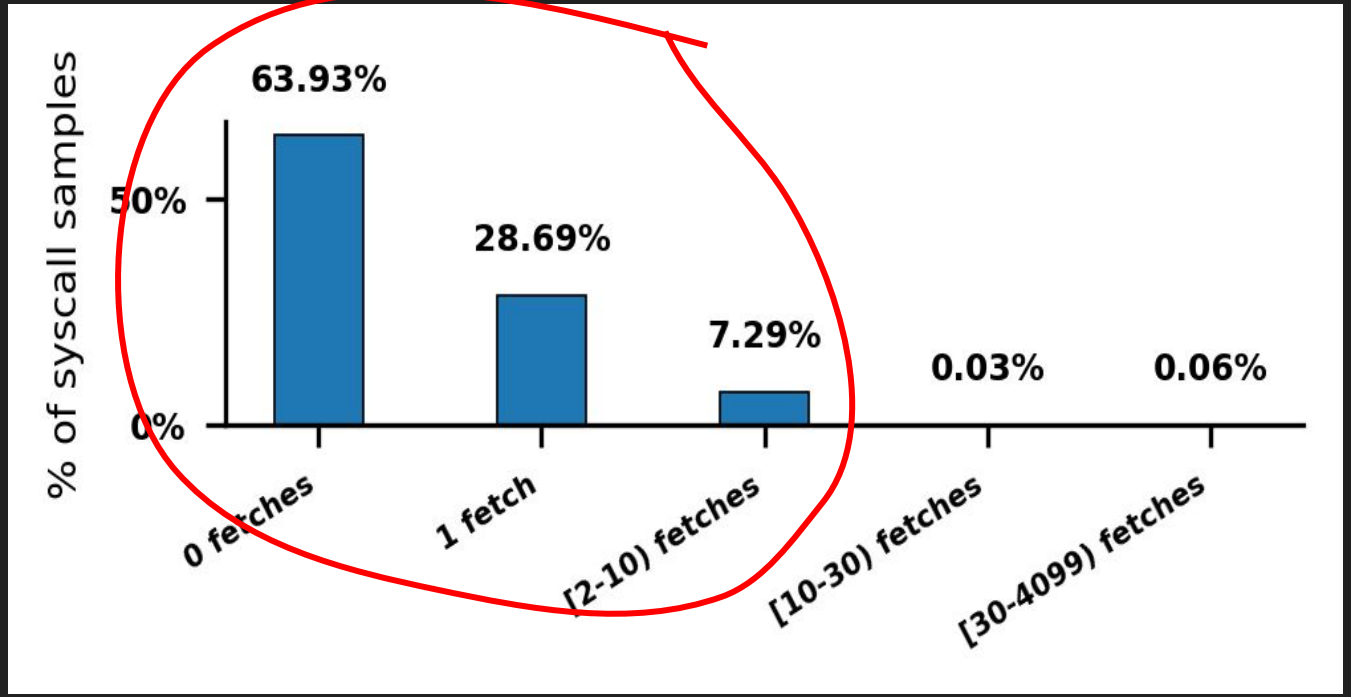
SafeFetch


Syscall
Fetch @data + check
Fetch @data + use
SafeFetch
Cache
@data
User context
@data
Cache Management
Efficient Range Queries

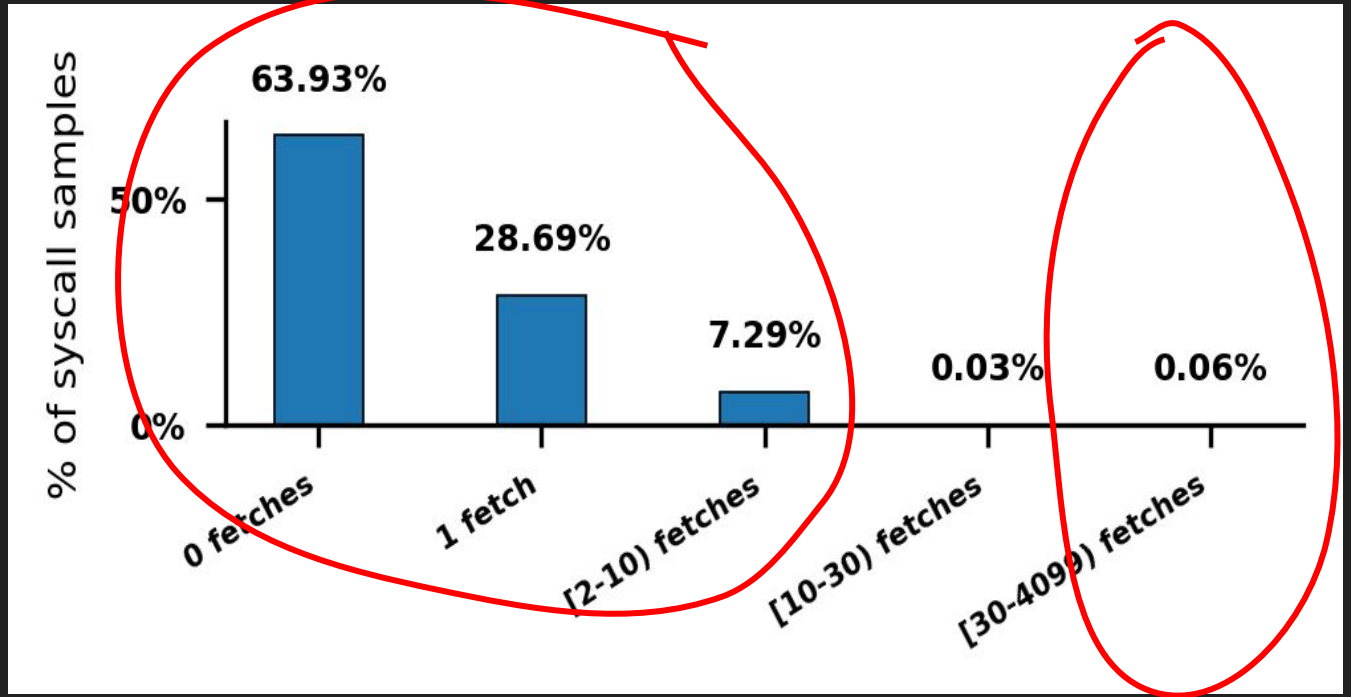
Efficient queries



Efficient queries

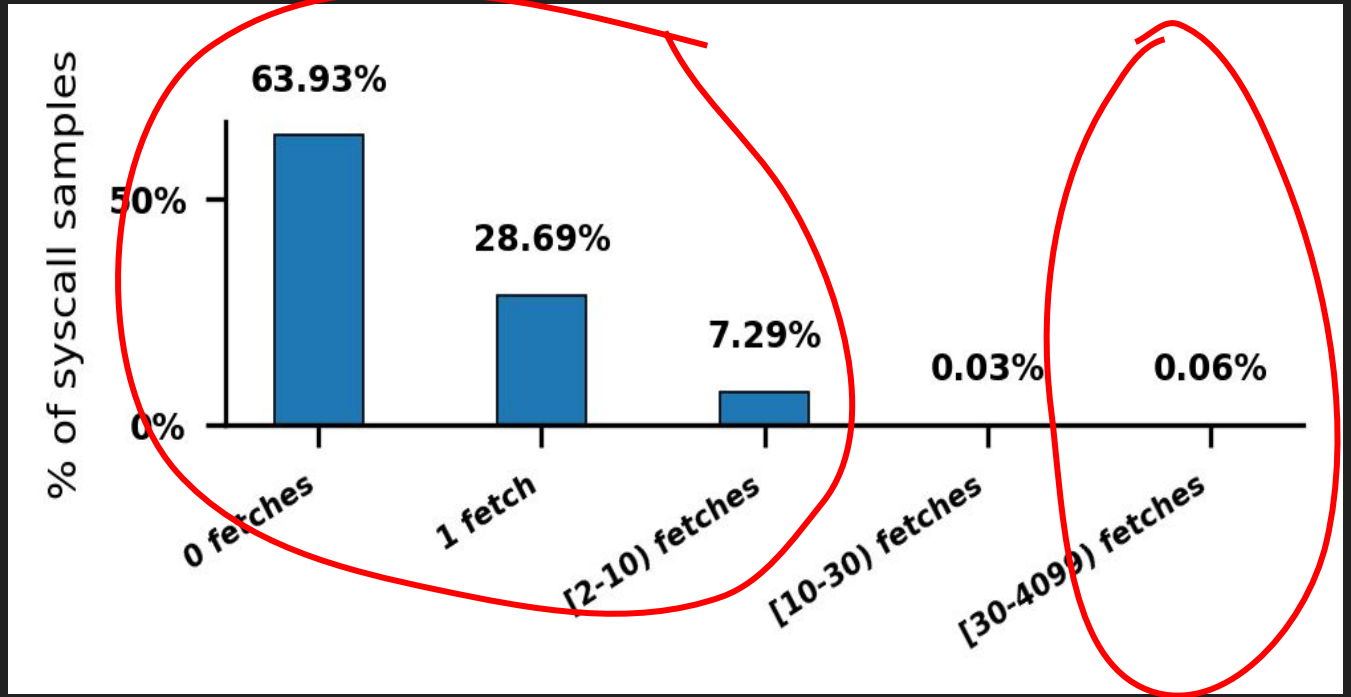


Efficient queries



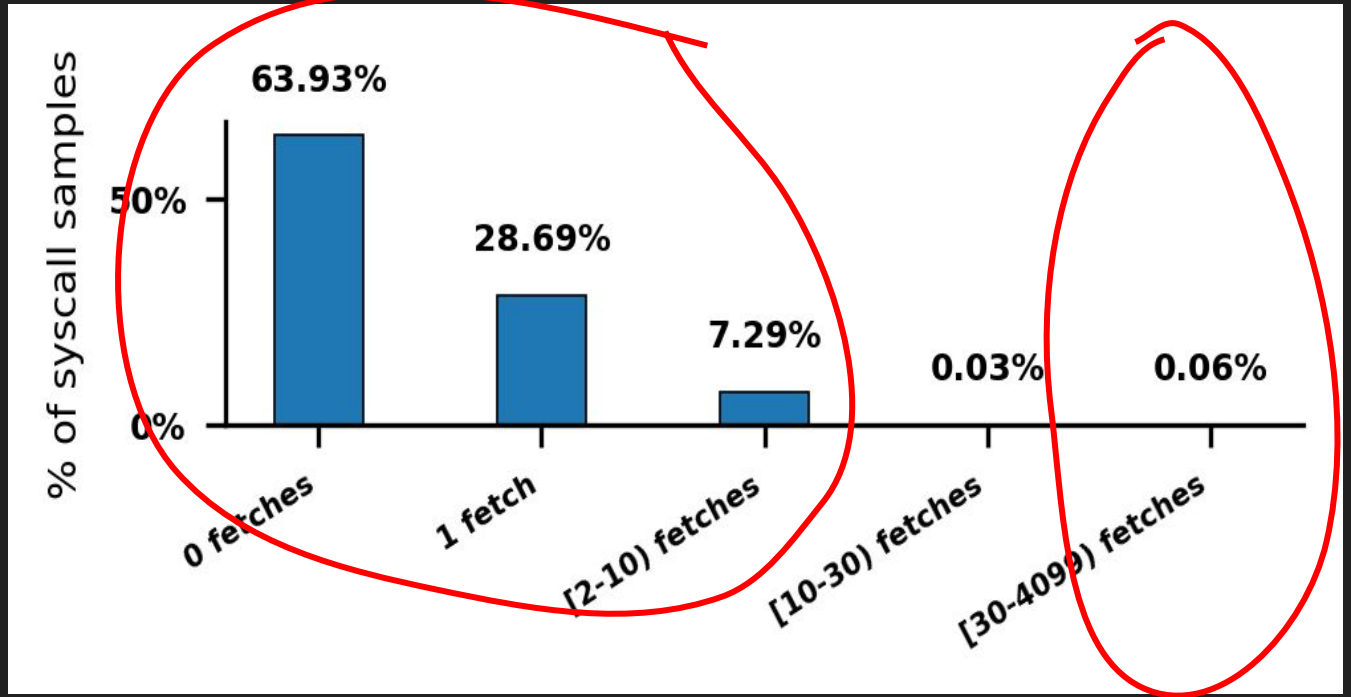
Efficient queries

linked list



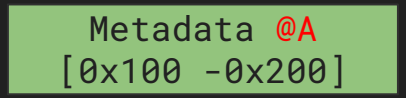
Efficient queries

linked list → rb-tree



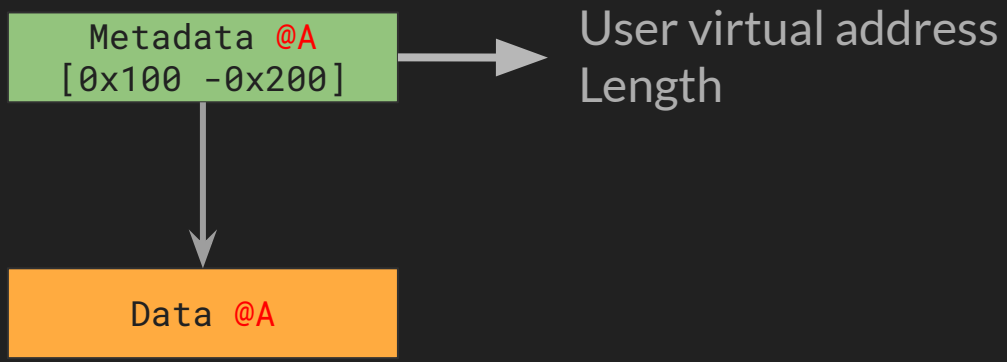
Cache Management

Cache Management

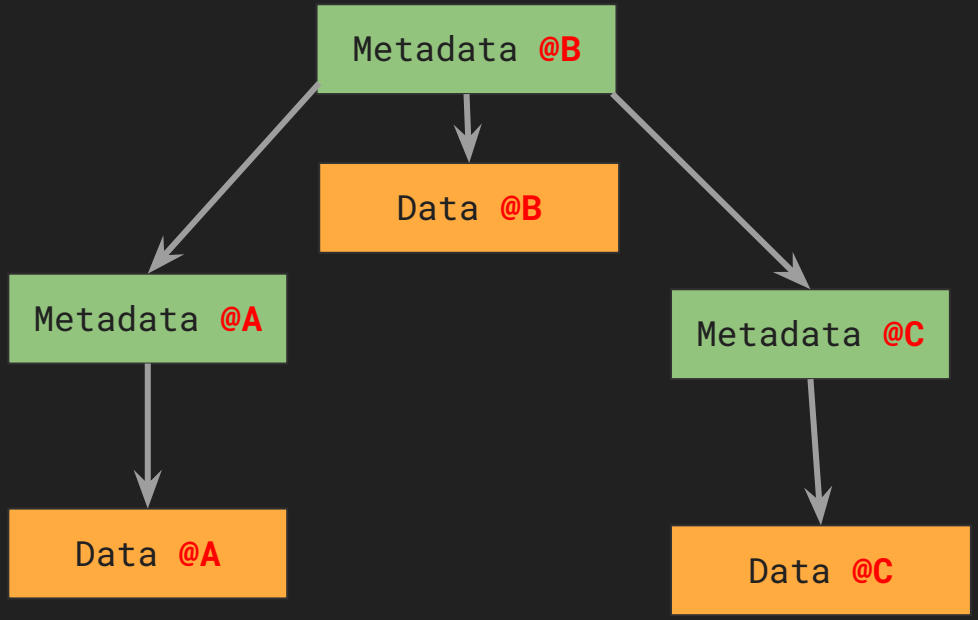


Cache Management

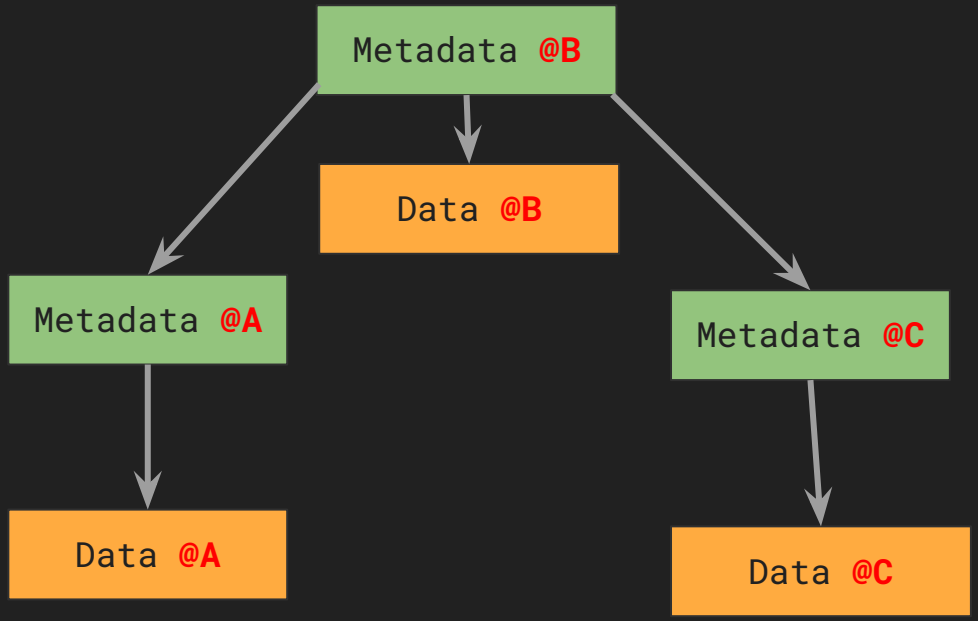
Cache Management



Cache management

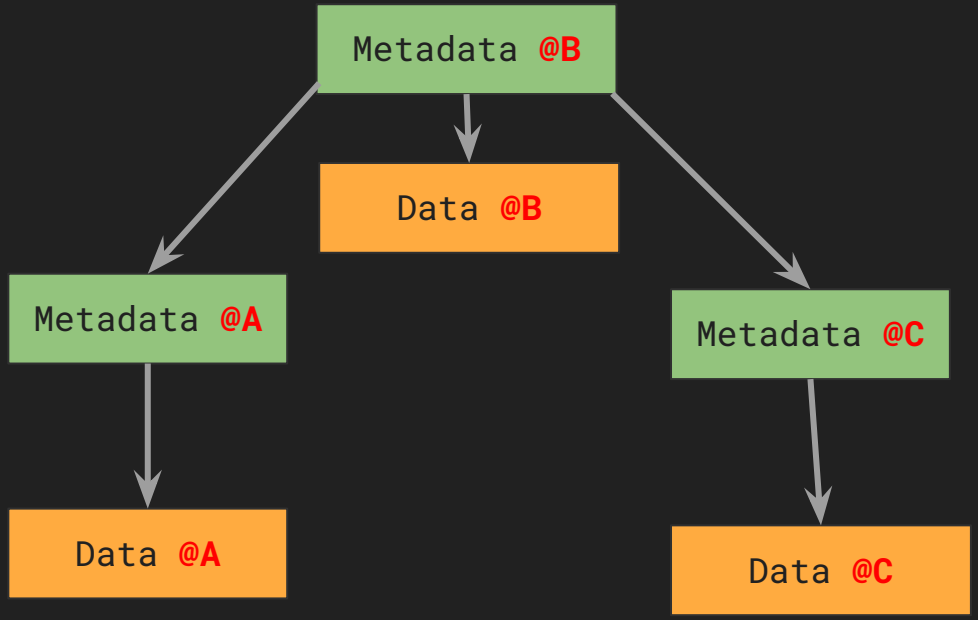


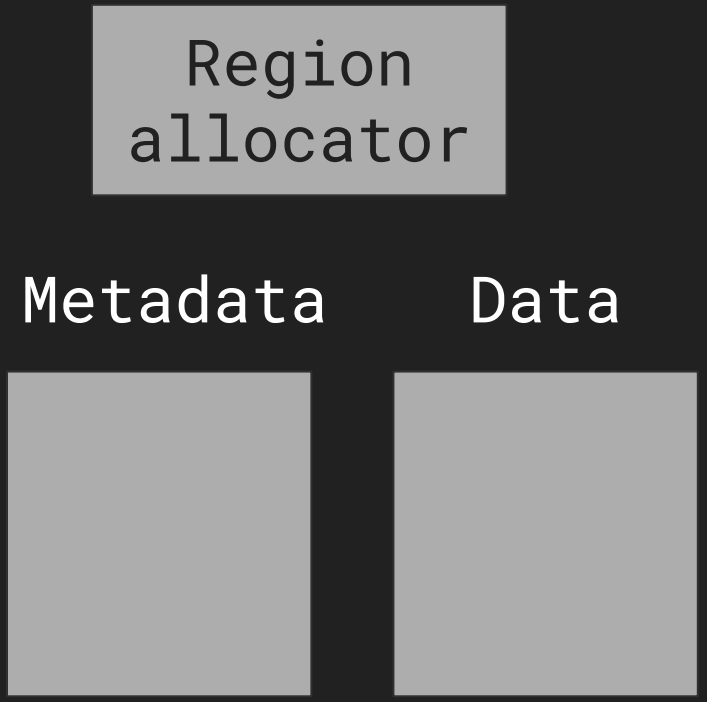
Cache management



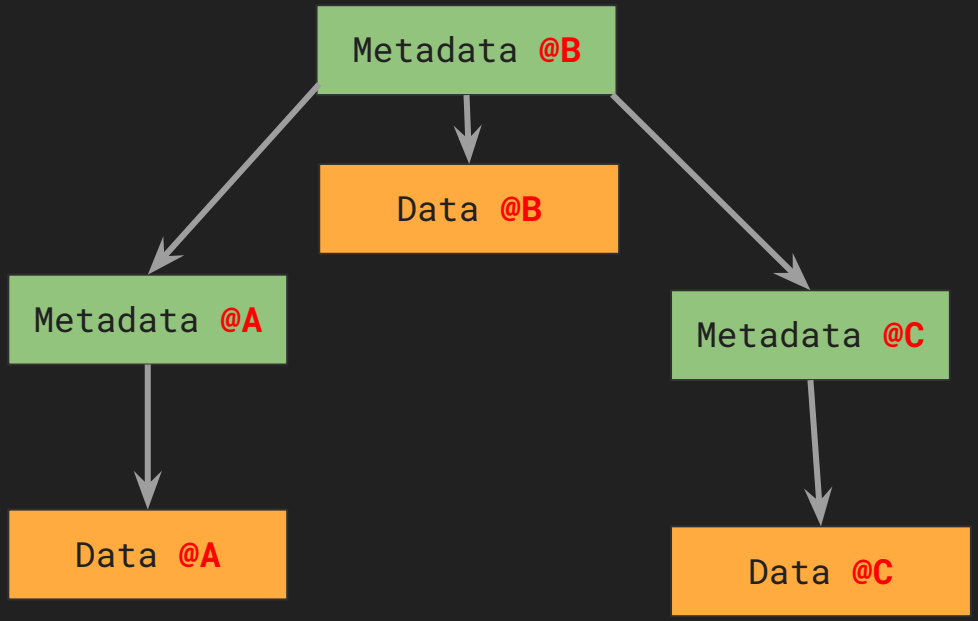
Region allocator

Cache management





Cache management



Region allocator

Metadata

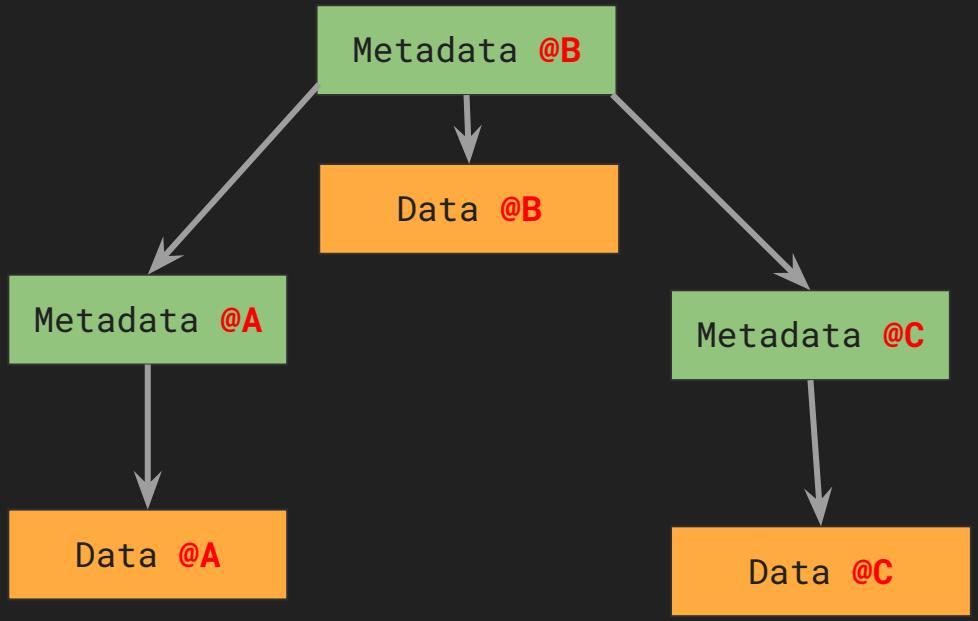
Metadata @A

Metadata @B

Metadata @C

Data

Cache management



Region allocator

Metadata

Metadata @A

Metadata @B

Metadata @C

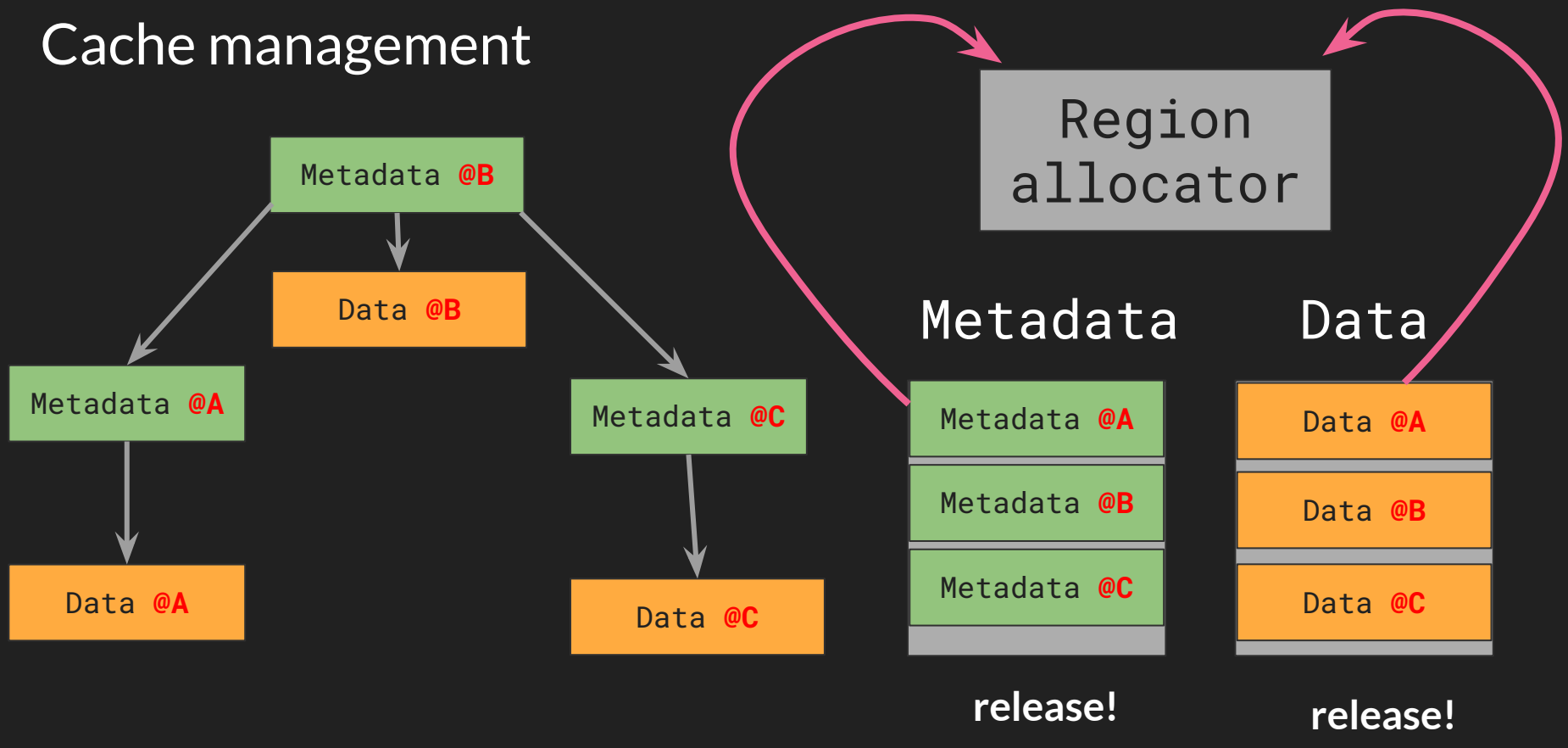
Data

Data @A

Data @B

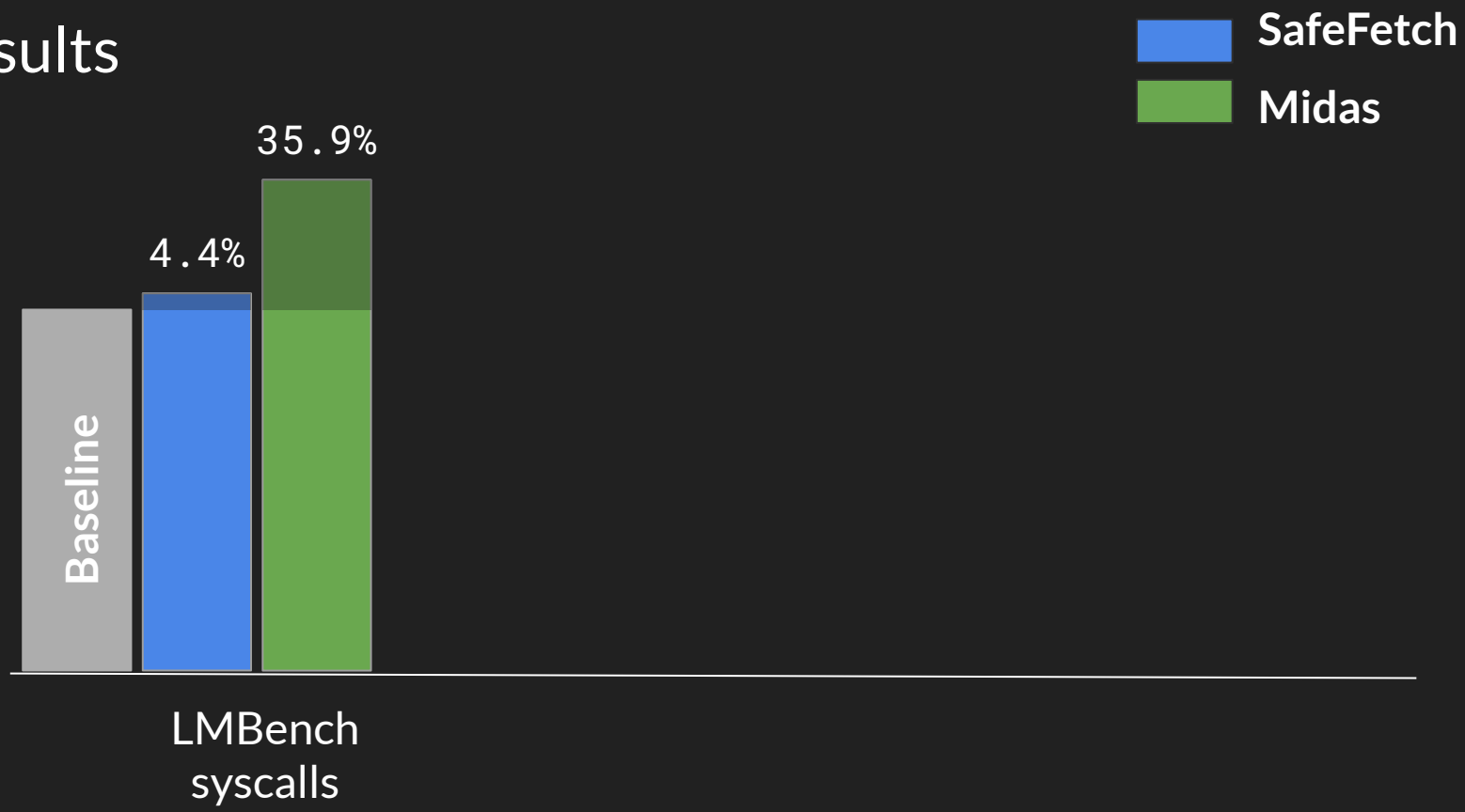
Data @C

Cache management

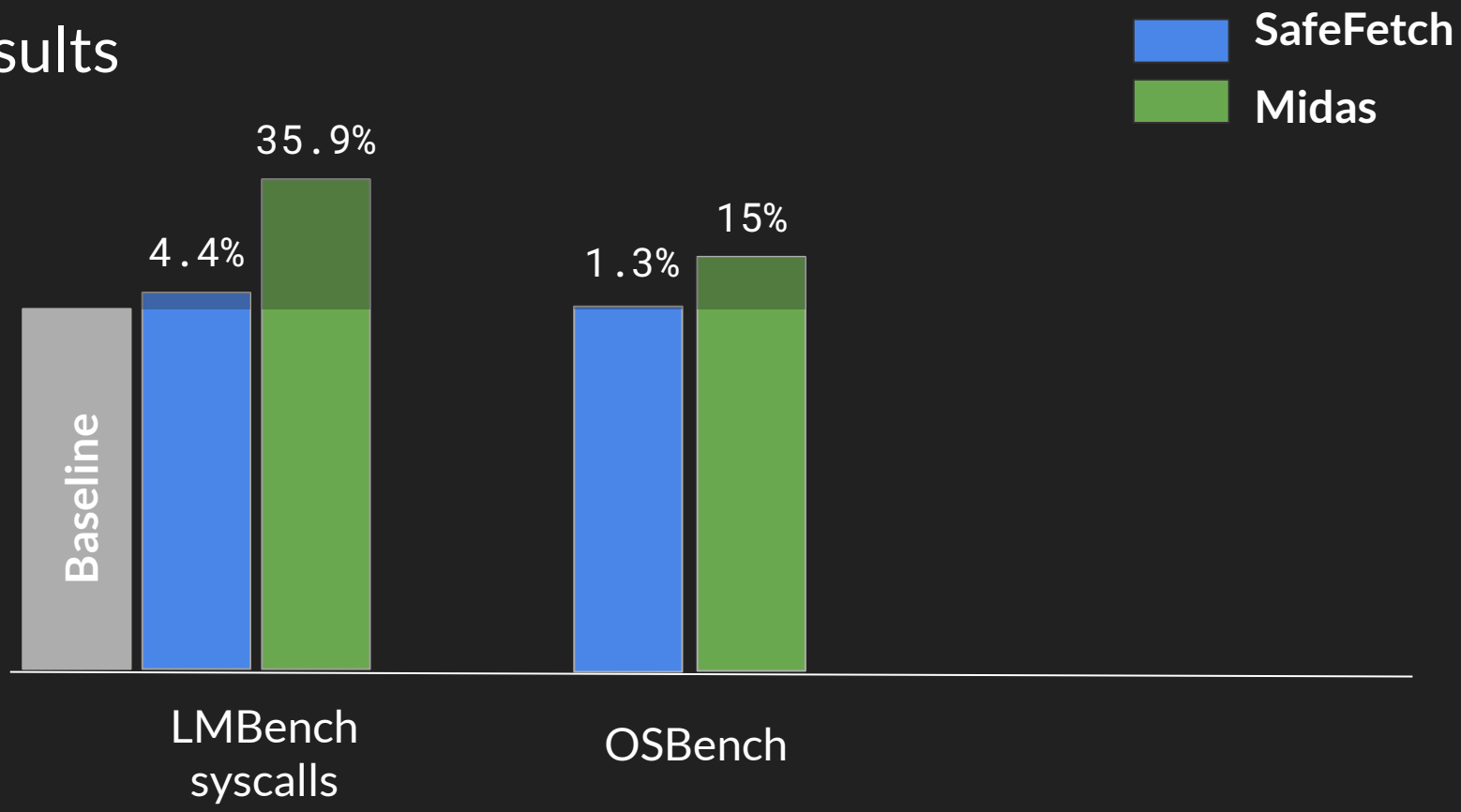


Results

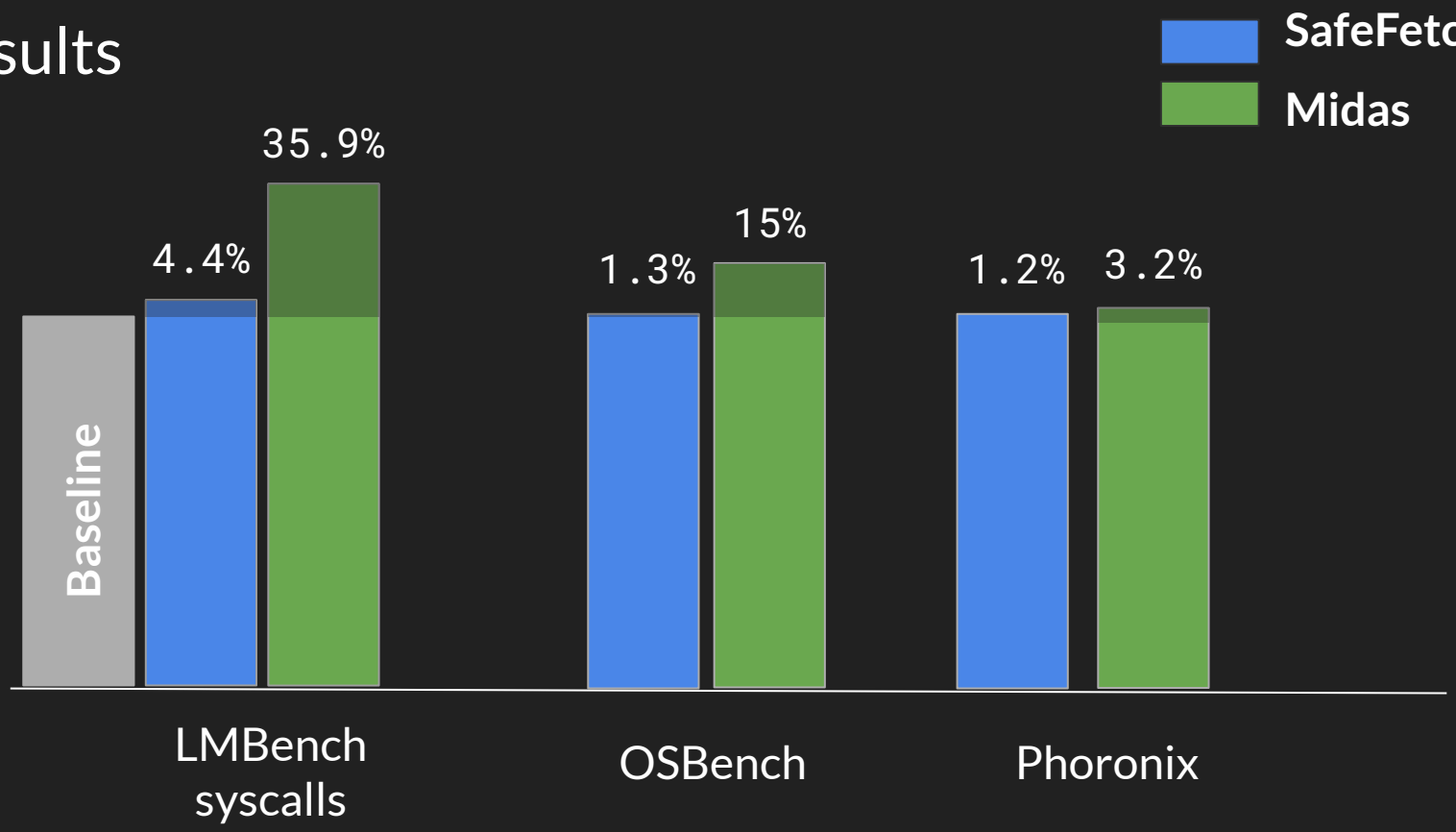
Results



Results


SafeFetch
Midas
35.9%
4.4%
15%
1.3%
Baseline
LMBench
syscalls
OSBench

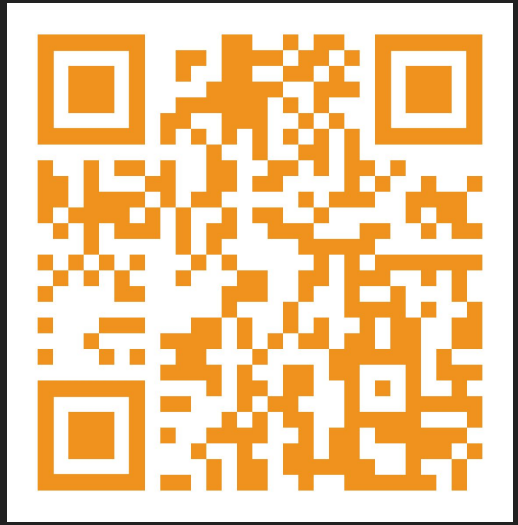
Results


SafeFetch
Midas
Baseline
4.4%
35.9%
1.3%
15%
1.2%
3.2%
LMBench syscalls
OSBench
Phoronix

Conclusion

- Comprehensive protection against double-fetch bugs
- Less than 5% geometric overhead across a variety of benchmarks
- CVE-2016-6516

Questions?



Github Repository

Memory utilization

Table 3: `SafeFetch`-induced memory utilization.

Benchmark	SafeFetch raw memory utilization					
	cache provisioning			zero-copy pins		
	avg.	99^{th}	peak	avg.	99^{th}	peak
LMBench	**8.02 KB**	**8 KB**	**236 KB**	**0.65 KB**	**0 KB**	**9.6 MB**
OSBench	**8.04 KB**	**12 KB**	**12 KB**	**0 KB**	**0 KB**	**0 KB**
Phoronix	**8.04 KB**	**8 KB**	**44 KB**	**5.82 KB**	**16 KB**	**128 KB**

(Double-)Fetch Rate

Table 5: Statistics for (double) fetch rates.

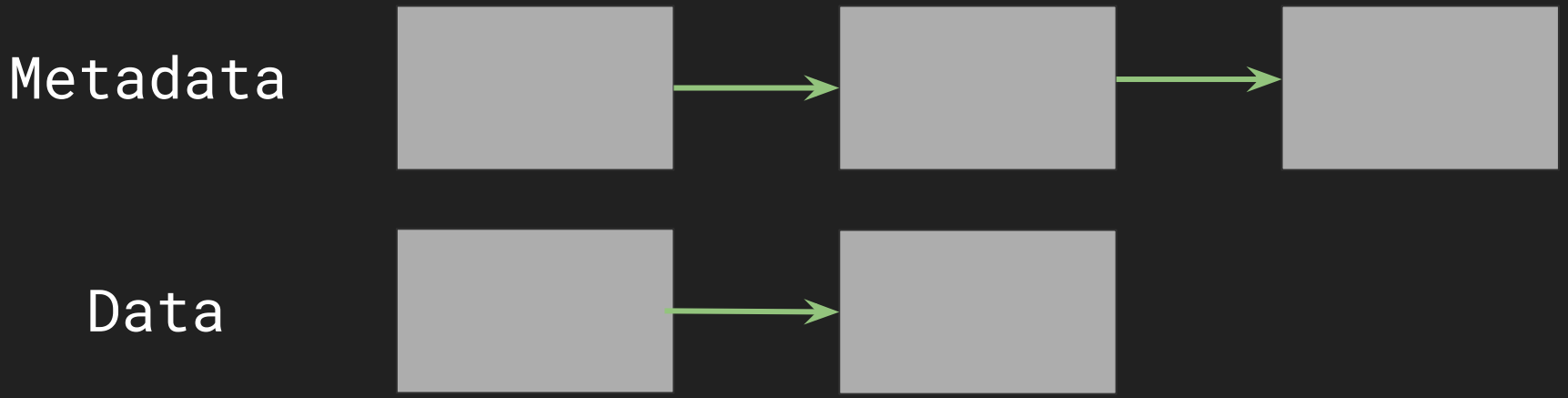
Benchmark	Statistic	Fetches	Double Fetches
LMBench	Rate	1/2	1/273457
	Syscalls	38	7
	Min/Avg/Max	1/1/459	1/50/67
OSBench	Rate	1/3	1/80
	Syscalls	17	5
	Min/Avg/Max	1/6/134	1/42/43
Phoronix	Rate	1/4	1/5993
	Syscalls	47	21
	Min/Avg/Max	1/1/661	1/1/218
Background	Rate	1/2	1/644
	Syscalls	93	20
	Min/Avg/Max	1/2/4099	1/130/467

Optimization benefits

Table 4: `SafeFetch`-induced throughput degradation.

Benchmark	SafeFetch /wo zero-copy (%)	SafeFetch /w zero-copy (%)
AF_UNIX	39.1%	8.4%
Pipe	16.1%	5.1%
Nginx	5.9%	1.3%
Apache	9.3%	1.8%

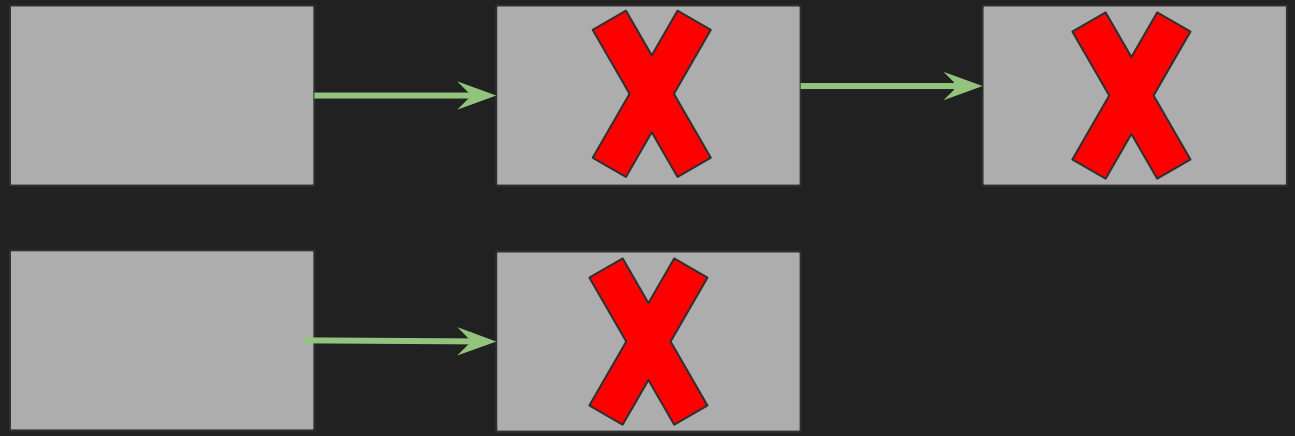
Cache management



Cache management

Metadata

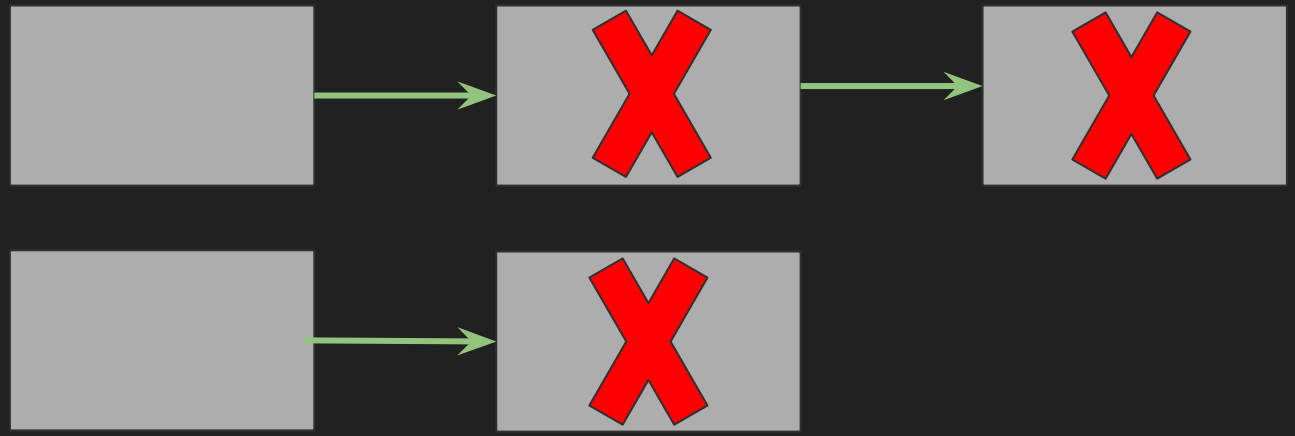
Data



Cache management

Metadata

Data



2 page allocations + 2 page releases + extra writes

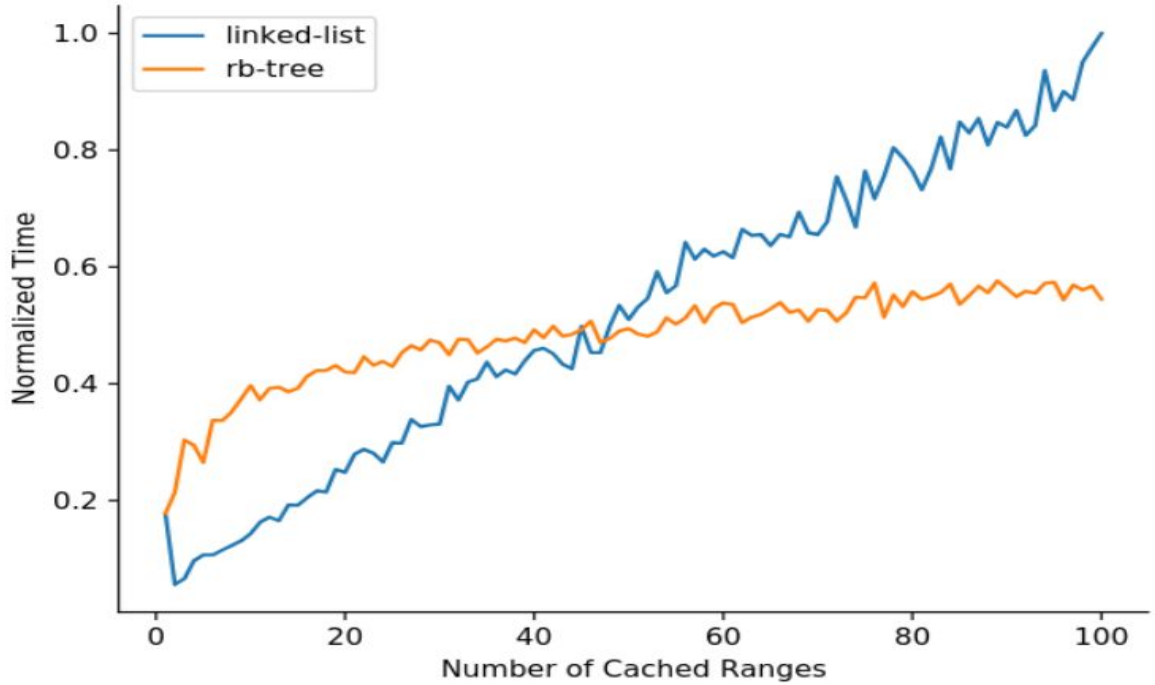
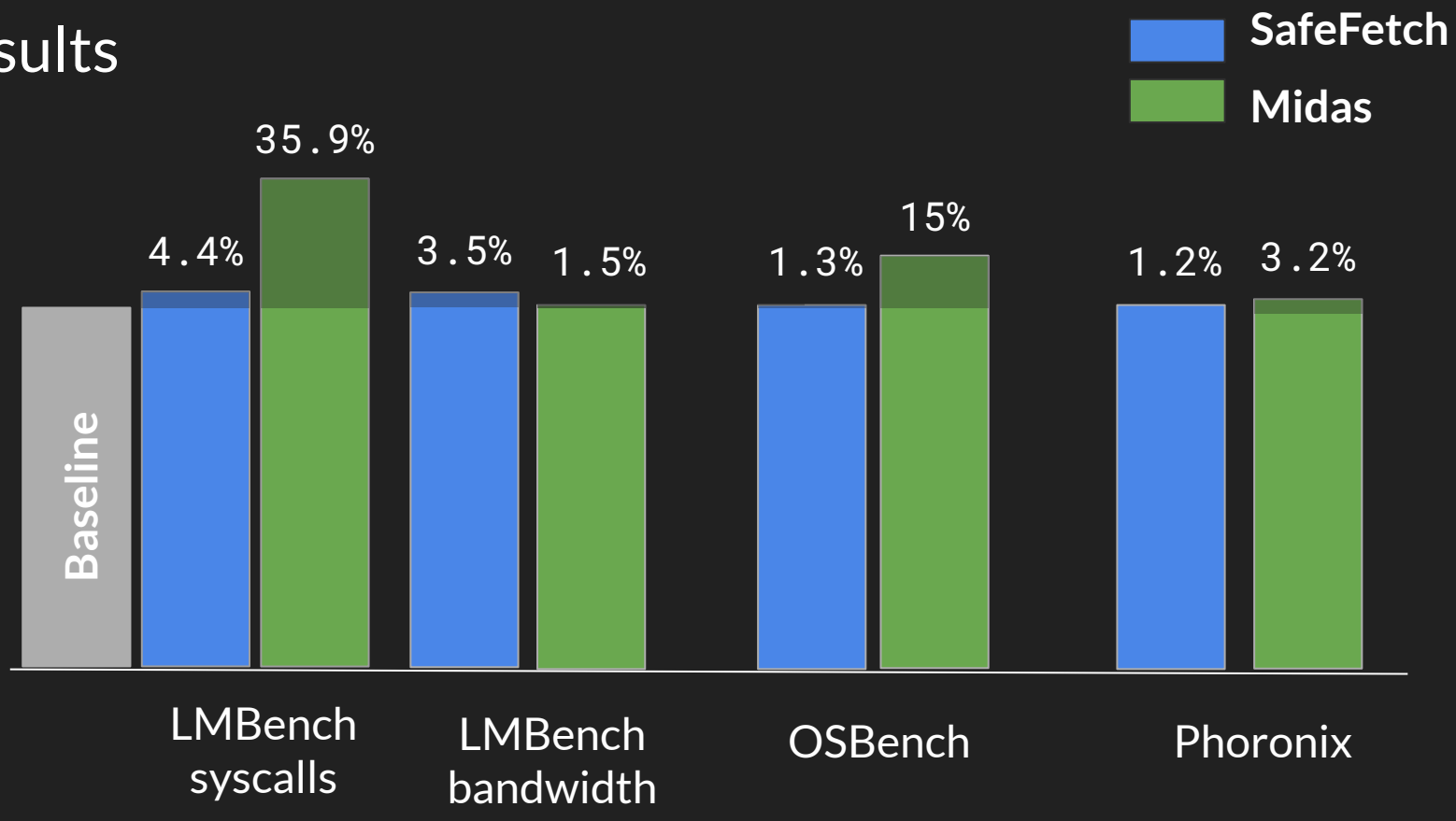
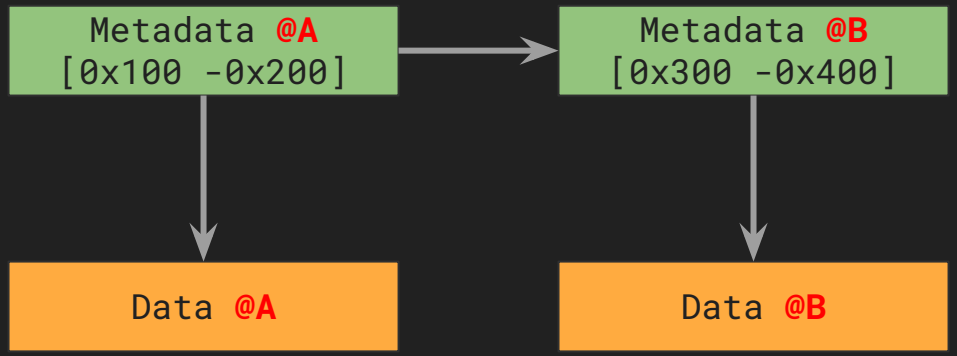


Figure 13: Average normalized time for each fetch vs. number of cached ranges.

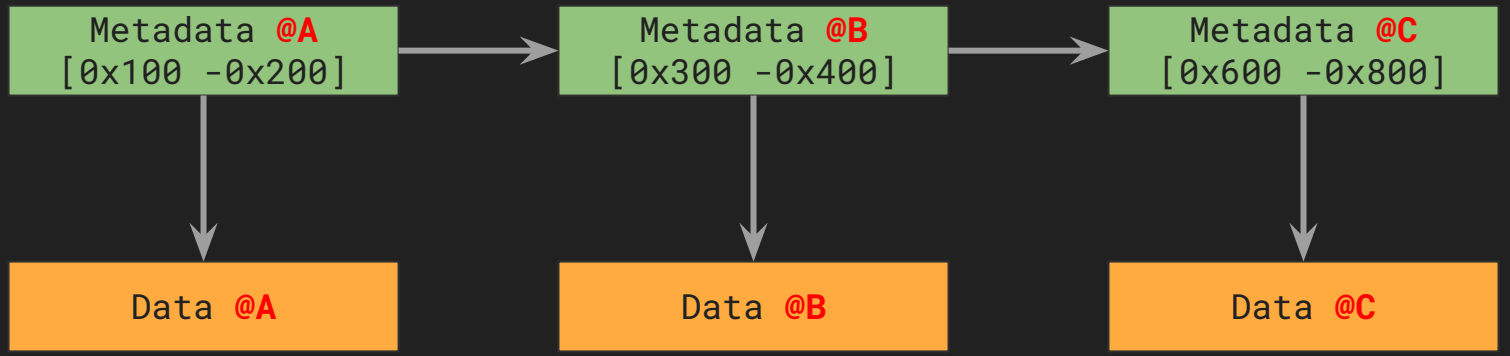
Results


SafeFetch
Midas
Baseline
4.4%
35.9%
3.5%
1.5%
1.3%
15%
1.2%
3.2%
LMBench syscalls
LMBench bandwidth
OSBench
Phoronix

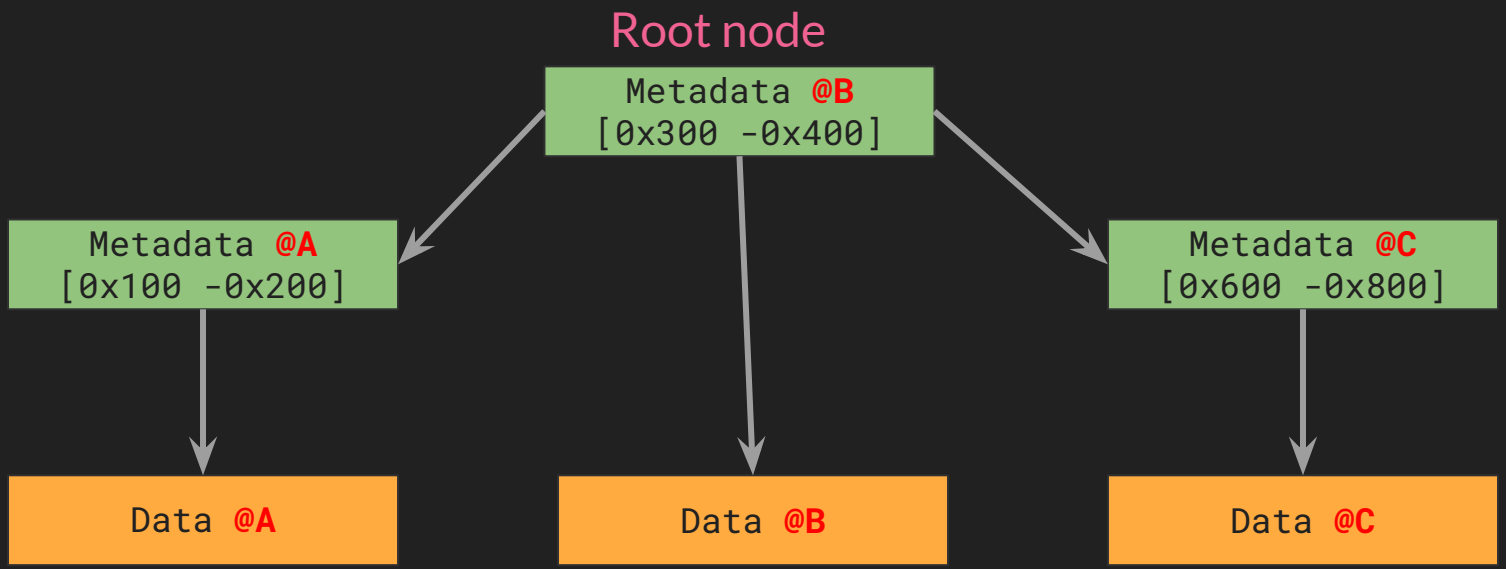
Efficient queries



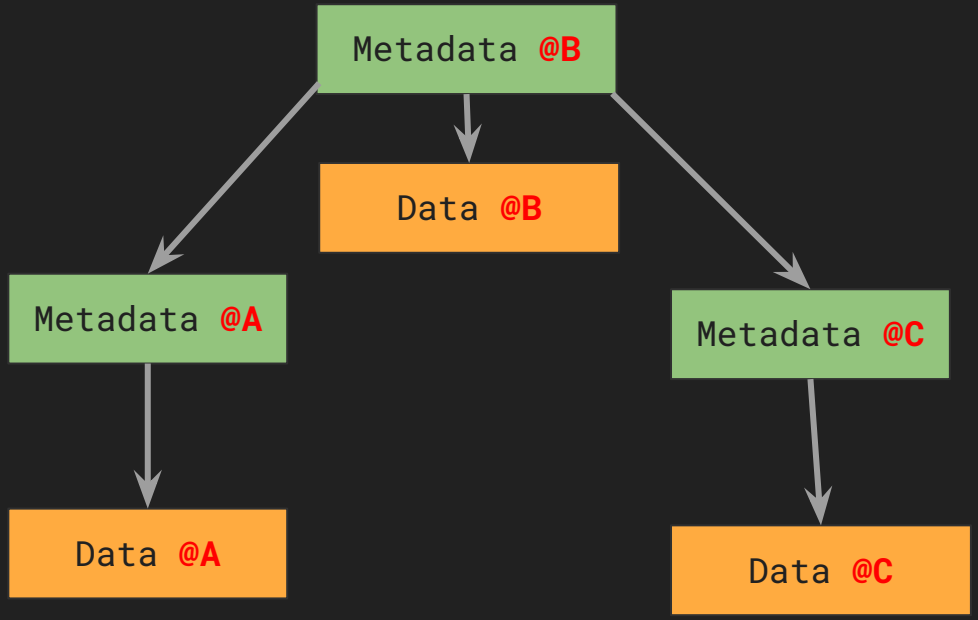
Efficient queries



Efficient queries

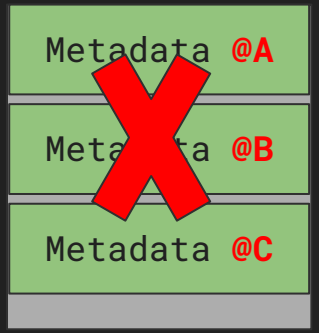


Cache management



Region allocator

Metadata



Data