Making Peace Between Mortal Enemies: Running a Database Management System in the Linux Kernel

Matt Butrovich

Carnegie Mellon University

September 16, 2024

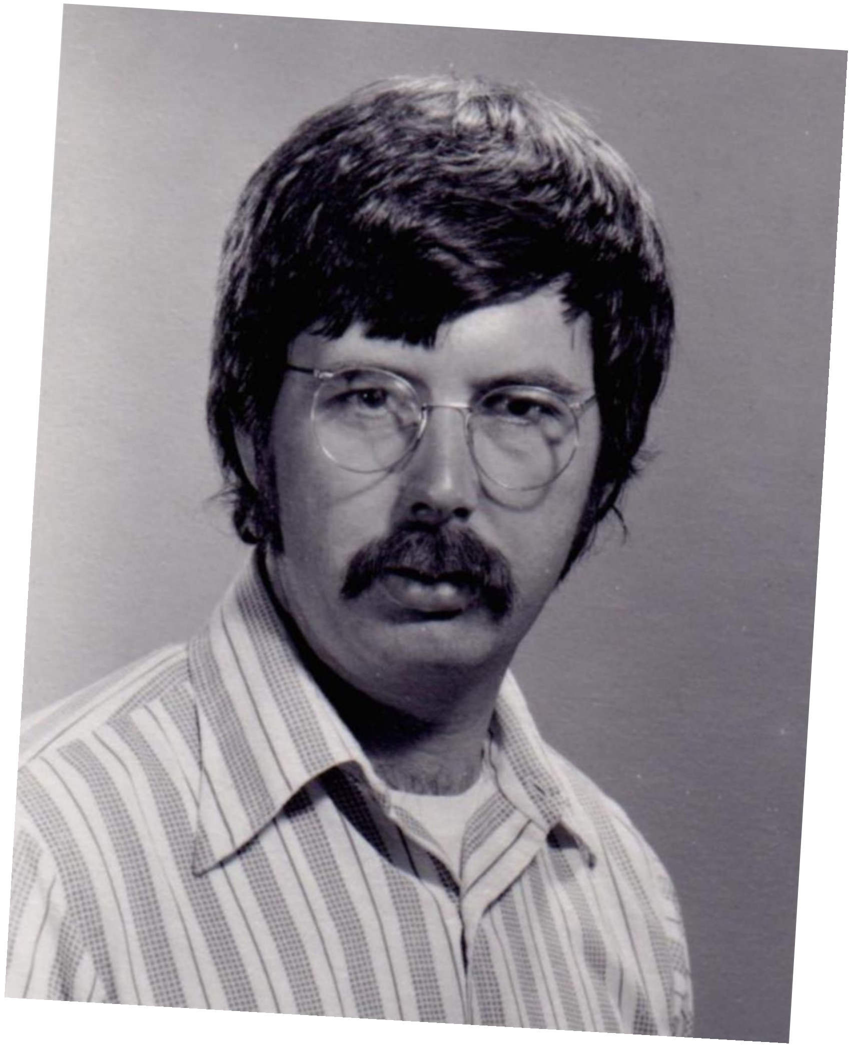
CARNEGIE MELLON
DATABASE GROUP

The OS Is Not Our Friend

The OS Is Not Our Friend

“The bottom line is that operating system services in many existing systems are either too slow or inappropriate.”

Michael Stonebraker. Operating System Support for Database Management. *Commun. ACM*. 1981.



My friend Mike

The Feud Goes On...

The Feud Goes On…

[Async I/O] is a horrible ad-hoc design, with the main excuse being "other, less gifted people, made that design, and we are implementing it for compatibility because database people - **who seldom have any shred of taste** - actually use it".

Linus Torvalds. Re: [PATCH 09/13] aio: add support for async openat(). *LKML*. 2016.

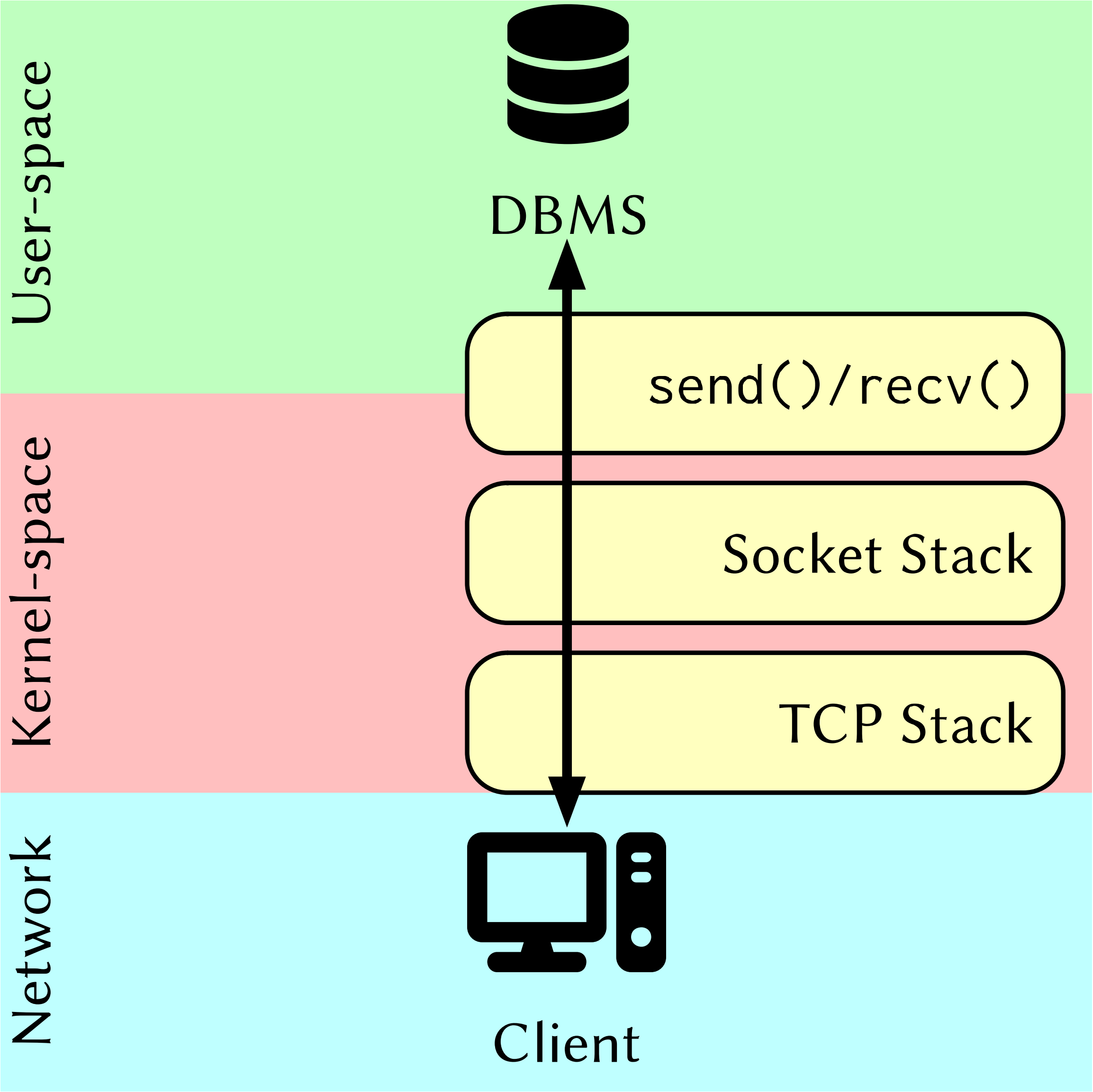
And On…

And On…

Linux tends to kill the postmaster in out-of-memory situations, because it blames the postmaster for the sum of child process sizes *including shared memory*. (**This is unbelievably stupid, but the kernel hackers seem uninterested in improving it.**)

postgres/src/backed/postmaster/fork_process.c:74

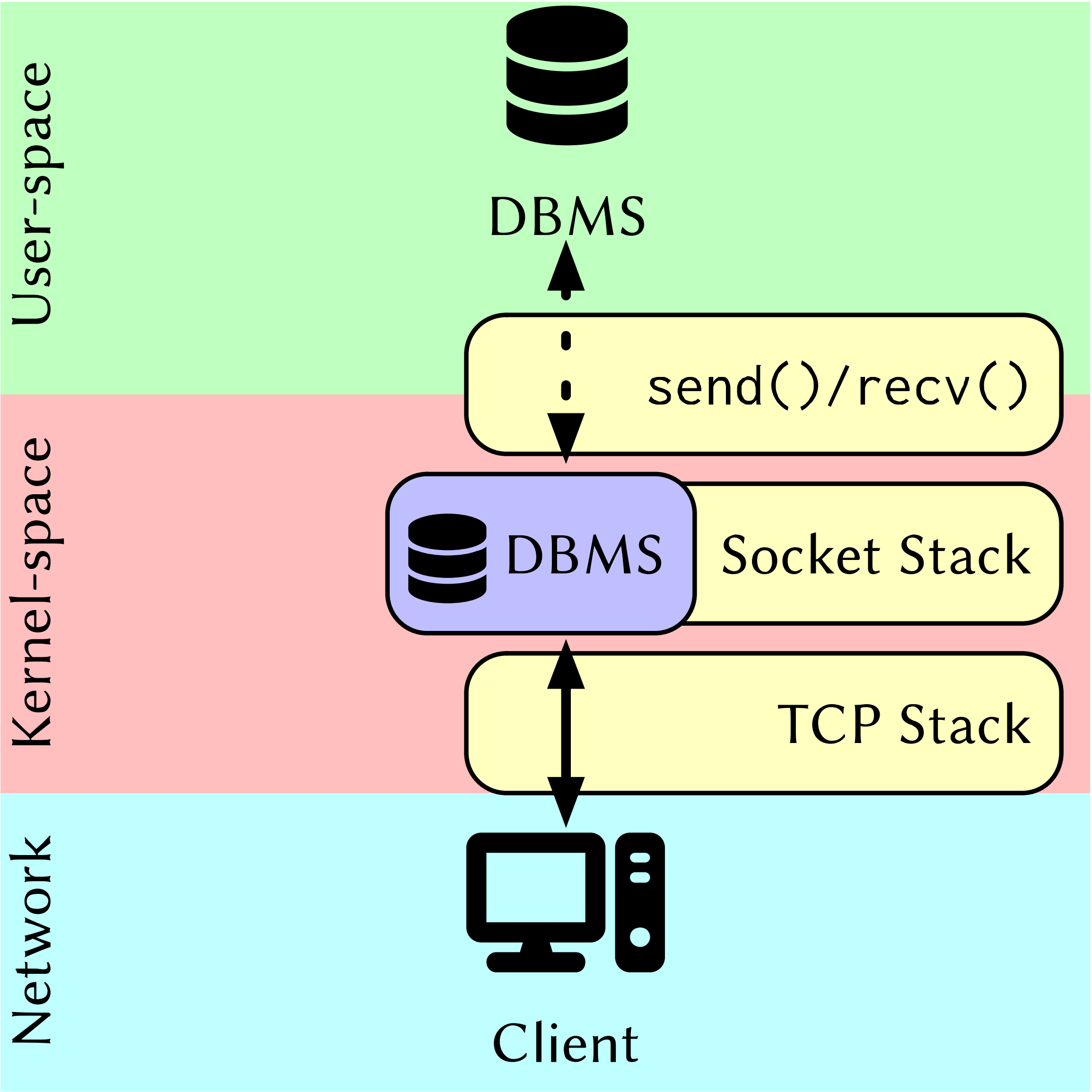
User-Bypass



User-space DBMS

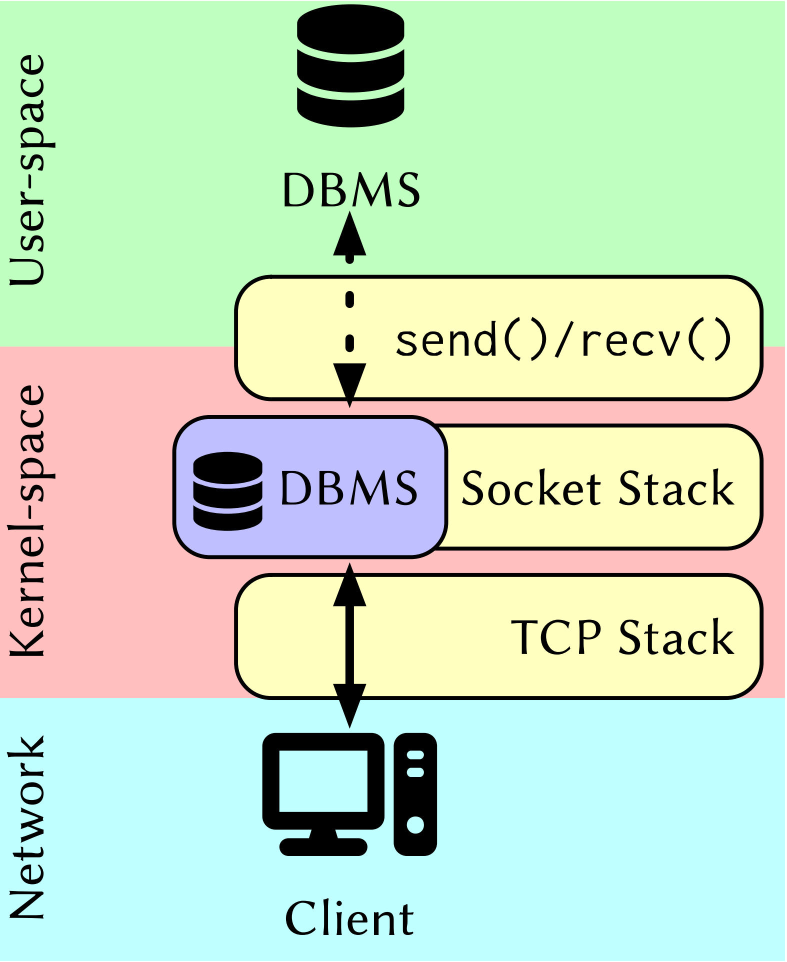
User-Bypass

CMU·DB



User-bypass DBMS

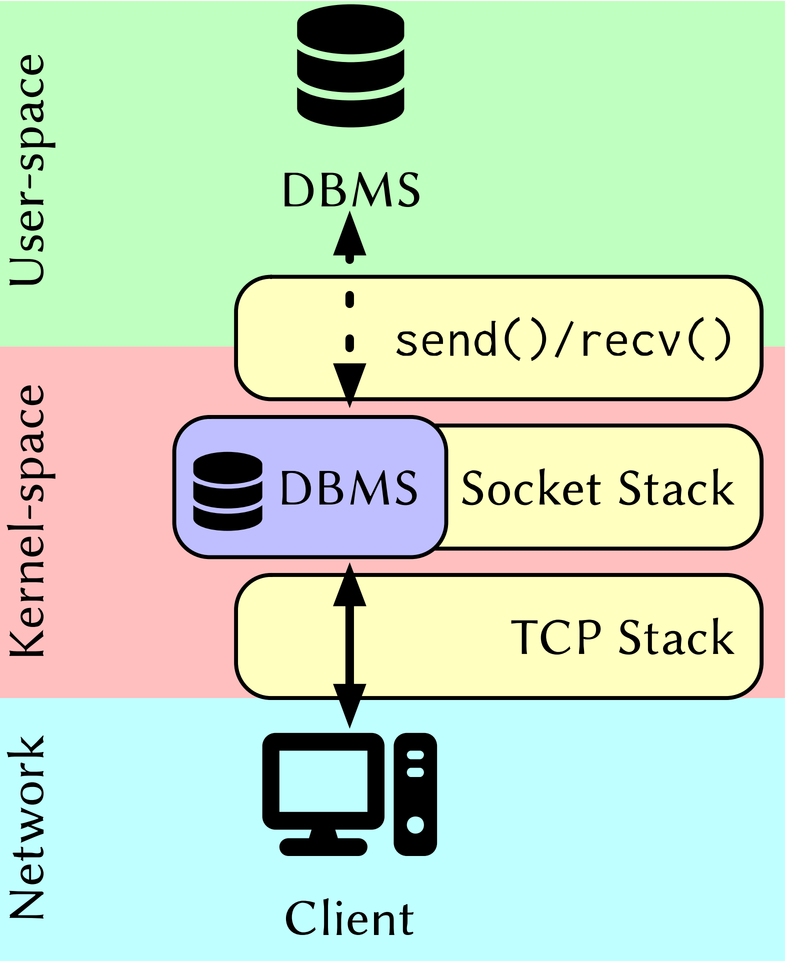
User-Bypass



User-bypass DBMS

User-Bypass

- Don’t pull DBMS data to user-space, push DBMS logic to kernel-space

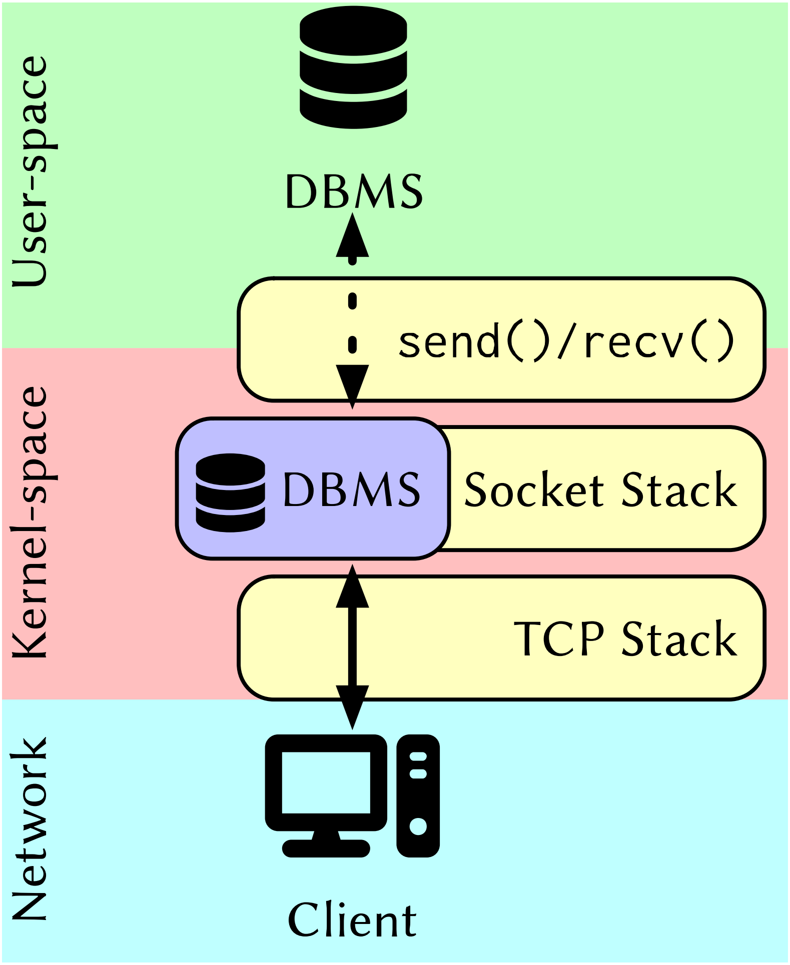


User-bypass DBMS

User-Bypass

- Don’t pull DBMS data to user-space, push DBMS logic to kernel-space

- Avoid copying buffers, scheduling user threads, and system call overhead



User-bypass DBMS

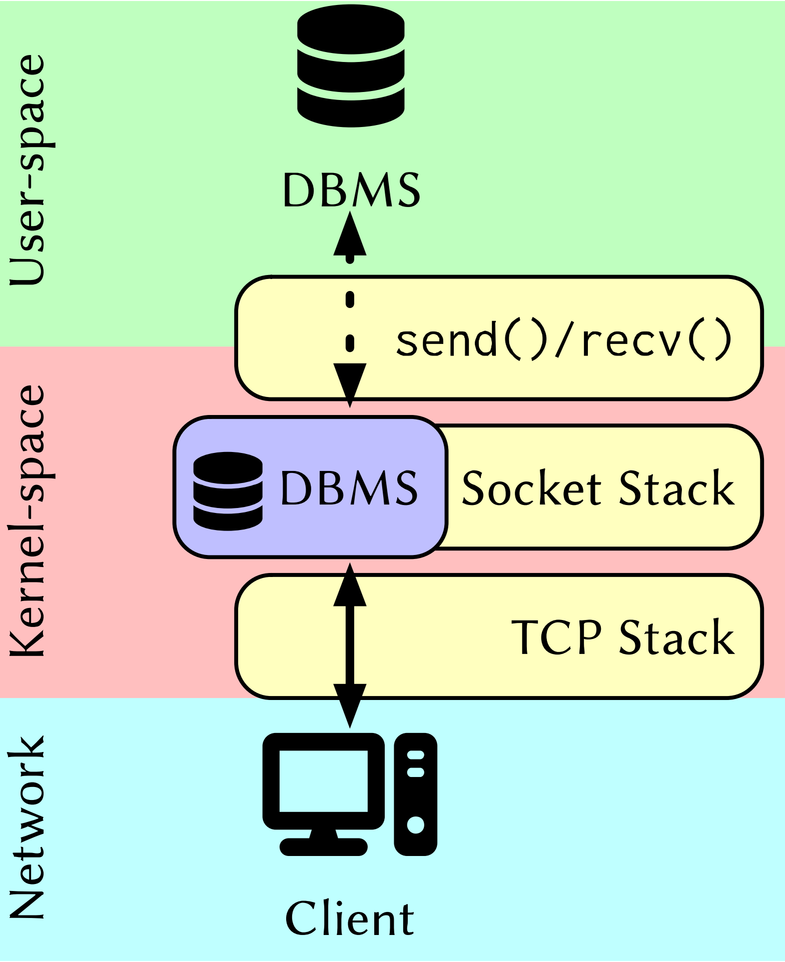
User-Bypass

- Don't pull DBMS data to user-space, push DBMS logic to kernel-space
- Avoid copying buffers, scheduling user threads, and system call overhead

Brian N. Bershad et al. Extensibility, Safety and Performance in the SPIN Operating System. *SOSP*. 1995.

Greg Ganger et al. Fast and flexible application-level networking on exokernel systems. *ACM Trans. Comput. Syst*. 2002.

Margo I. Seltzer et al. Dealing with Disaster: Surviving Misbehaved Kernel Extensions. *OSDI*. 1996.



User-bypass DBMS

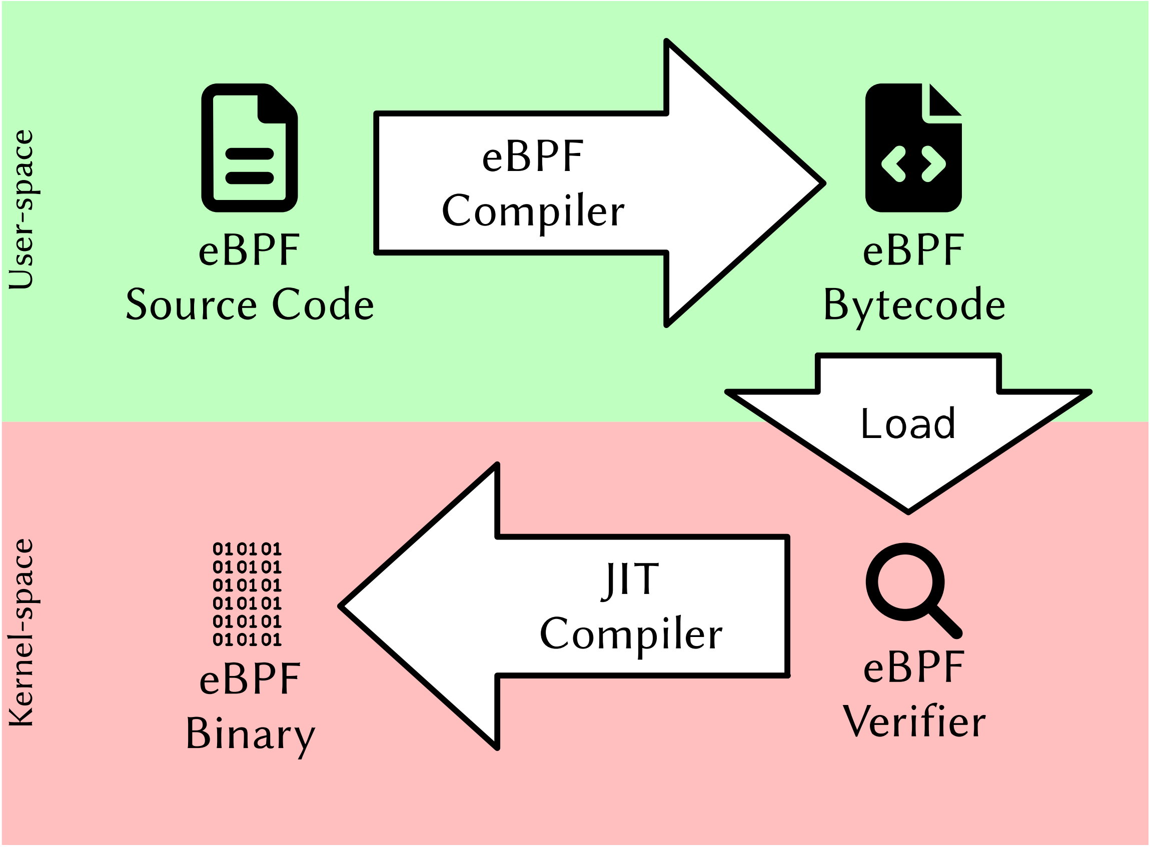
extended Berkeley Packet Filter

extended Berkeley Packet Filter

- Safe, event-driven programs in kernel-space

extended Berkeley Packet Filter

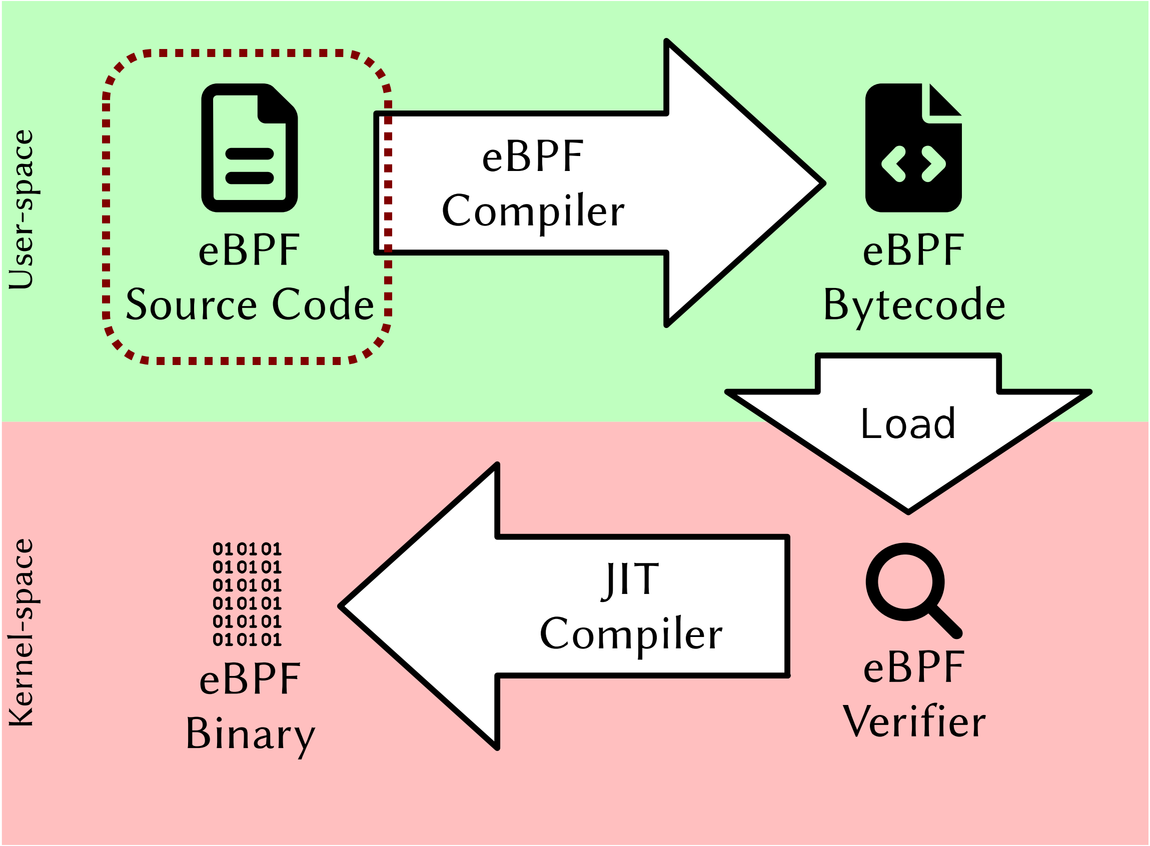
- Safe, event-driven programs in kernel-space



extended Berkeley Packet Filter

- Safe, event-driven programs in kernel-space

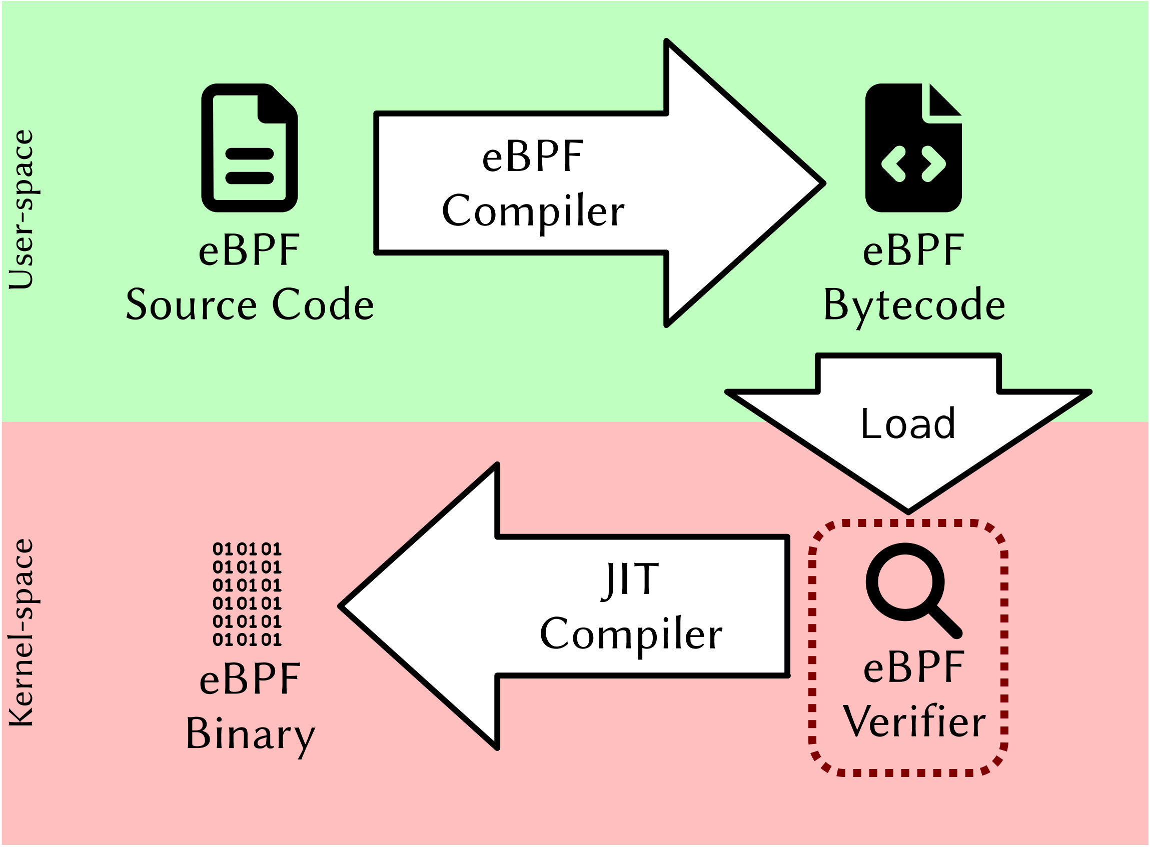
- Write in C and compile to eBPF



extended Berkeley Packet Filter

- Safe, event-driven programs in kernel-space

- Write in C and compile to eBPF

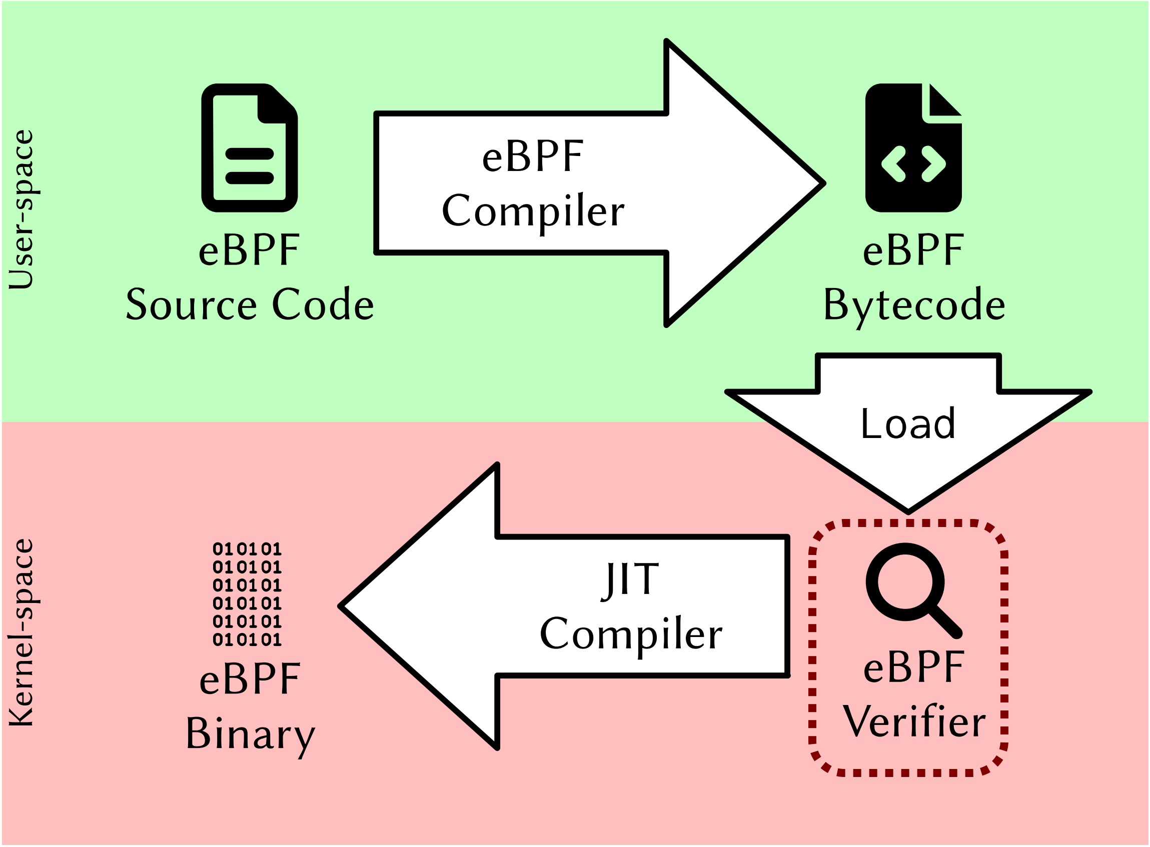


extended Berkeley Packet Filter

- Safe, event-driven programs in kernel-space

- Write in C and compile to eBPF

- Verifier constraints:
    - # instructions, boundedness, memory safety, limited API



eBPF Environment

eBPF Environment

- Attach to user-space or kernel-space hooks
  - User-space ⇒ “new system call”
  - Kernel-space ⇒ observe/modify OS logic

eBPF Environment

- Attach to user-space or kernel-space hooks
  - User-space ⇒ “new system call”
  - Kernel-space ⇒ observe/modify OS logic

- Ephemeral program execution
  - No heap allocations

eBPF Environment

- Attach to user-space or kernel-space hooks
  - User-space ⇒ “new system call”
  - Kernel-space ⇒ observe/modify OS logic

- Ephemeral program execution
  - No heap allocations

- eBPF maps: kernel-resident data structures
  - Key-value interface
  - Hash tables, stacks/queues, arrays, etc.

eBPF in the Wild

eBPF in the Wild

- eBPF merged into Linux kernel in 2014

eBPF in the Wild

- eBPF merged into Linux kernel in 2014

- New features with each kernel release

eBPF in the Wild

- eBPF merged into Linux kernel in 2014

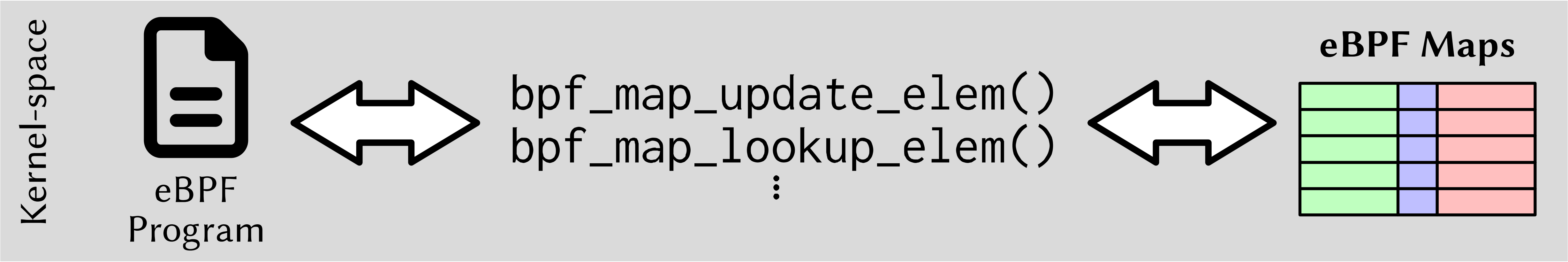
- New features with each kernel release

- Widely deployed in hyperscalers

eBPF in the Wild

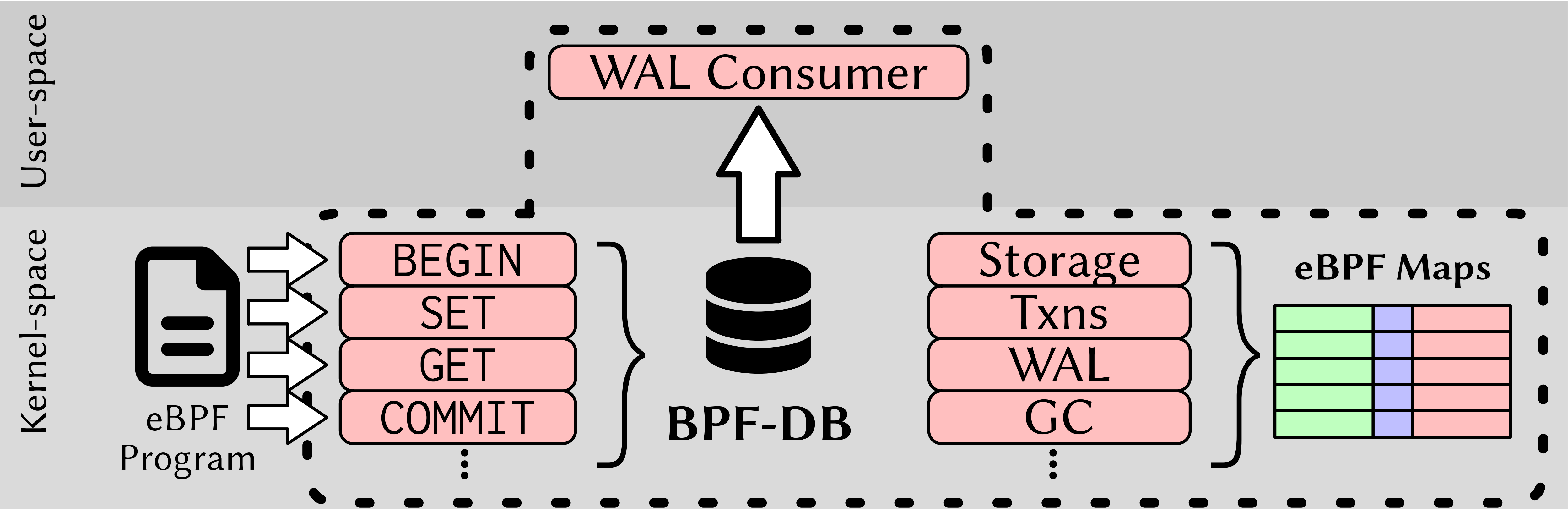
- eBPF merged into Linux kernel in 2014
- New features with each kernel release
- Widely deployed in hyperscalers

eBPF DBMS



```
bpf_map_update_elem()
bpf_map_lookup_elem()
⋮
```

eBPF DBMS


User-space
WAL Consumer
Kernel-space
eBPF Program
BEGIN
SET
GET
COMMIT
BPF-DB
Storage
Txns
WAL
GC
eBPF Maps

The Verifier Strikes Back

The Verifier Strikes Back

- User-bypass programs limited to 1M eBPF instructions

The Verifier Strikes Back

- ~~User-bypass programs limited to 1M eBPF instructions~~

- User-bypass programs limited to 1M *verified* eBPF instructions

The Verifier Strikes Back

- ~~User-bypass programs limited to 1M eBPF instructions~~

- User-bypass programs limited to 1M *verified* eBPF instructions
  - Branches and loops all need to be explored

The Verifier Strikes Back

- ~~User-bypass programs limited to 1M eBPF instructions~~

- User-bypass programs limited to 1M *verified* eBPF instructions
  - Branches and loops all need to be explored
  - Recursion is almost impossible

The Verifier Strikes Back

- ~~User-bypass programs limited to 1M eBPF instructions~~

- User-bypass programs limited to 1M *verified* eBPF instructions
  - Branches and loops all need to be explored
  - Recursion is almost impossible

- Tail-calling between eBPF programs (up to 32) helps

BPF-DB Goals

BPF-DB Goals

- Design and implement BPF-DB with traditional DBMS components and features (e.g., ACID transactions)

BPF-DB Goals

- Design and implement BPF-DB with traditional DBMS components and features (e.g., ACID transactions)

- Decompose DBMS components using continuation passing style to satisfy eBPF verifier

BPF-DB Goals

- Design and implement BPF-DB with traditional DBMS components and features (e.g., ACID transactions)

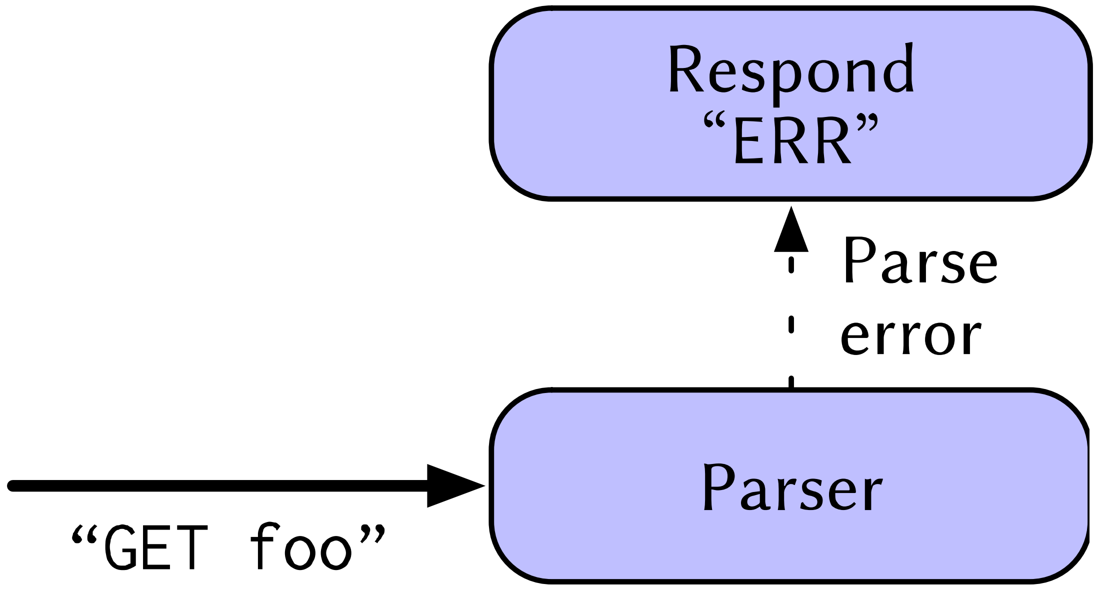
- Decompose DBMS components using continuation passing style to satisfy eBPF verifier

- Developers build rich applications using BPF-DB as their backing store (e.g., RocksDB for eBPF)

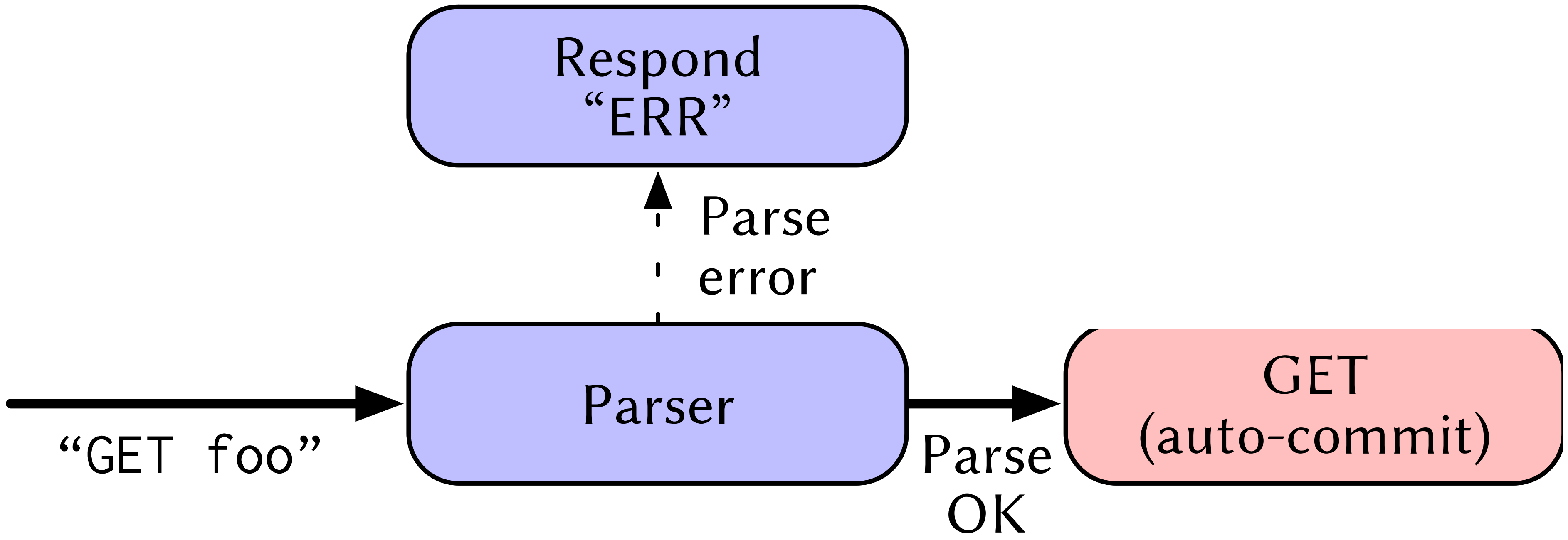
GET Tail-Calls

“GET foo”

GET Tail-Calls



GET Tail-Calls



Respond “ERR”

Parse error

“GET foo”

Parser

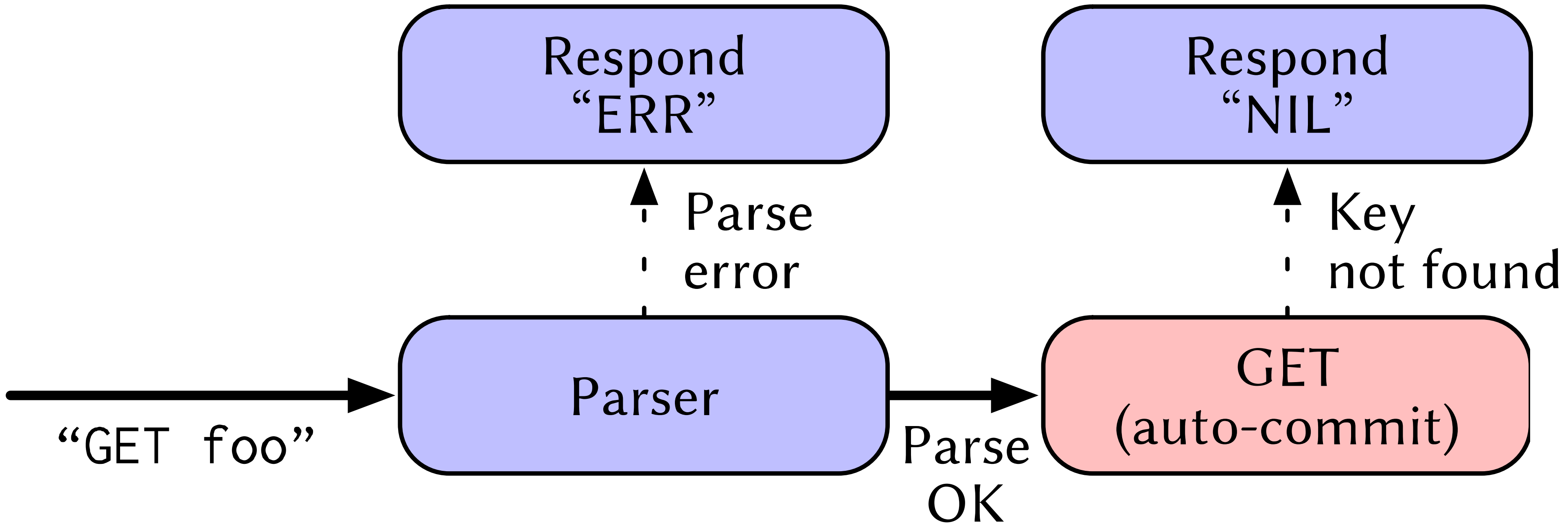
Parse OK

GET (auto-commit)

Context

Key:	“foo”
Value:	

GET Tail-Calls



Respond “ERR”

Respond “NIL”

Parse error

Key not found

“GET foo”

Parser

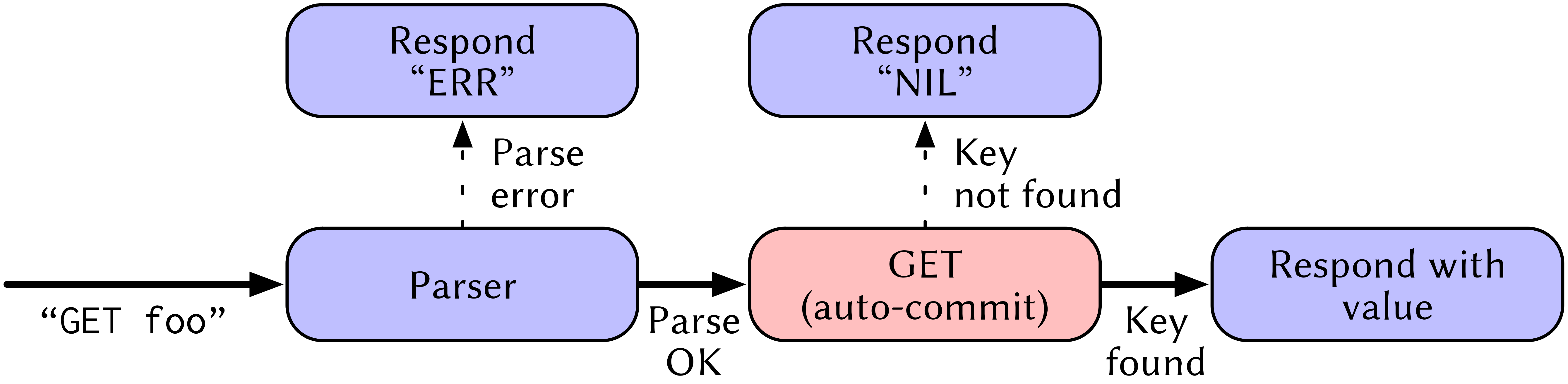
Parse OK

GET (auto-commit)

Context

Key:	“foo”
Value:	

GET Tail-Calls



Respond “ERR”

Respond “NIL”

Parse error

Key not found

“GET foo”

Parser

Parse OK

GET (auto-commit)

Key found

Respond with value

Context

Key:	“foo”
Value:	

Context

Key:	“foo”
Value:	“bar”

Storage Management

Storage Management

- Goal: Store database contents in kernel-resident thread-safe data structures

Storage Management

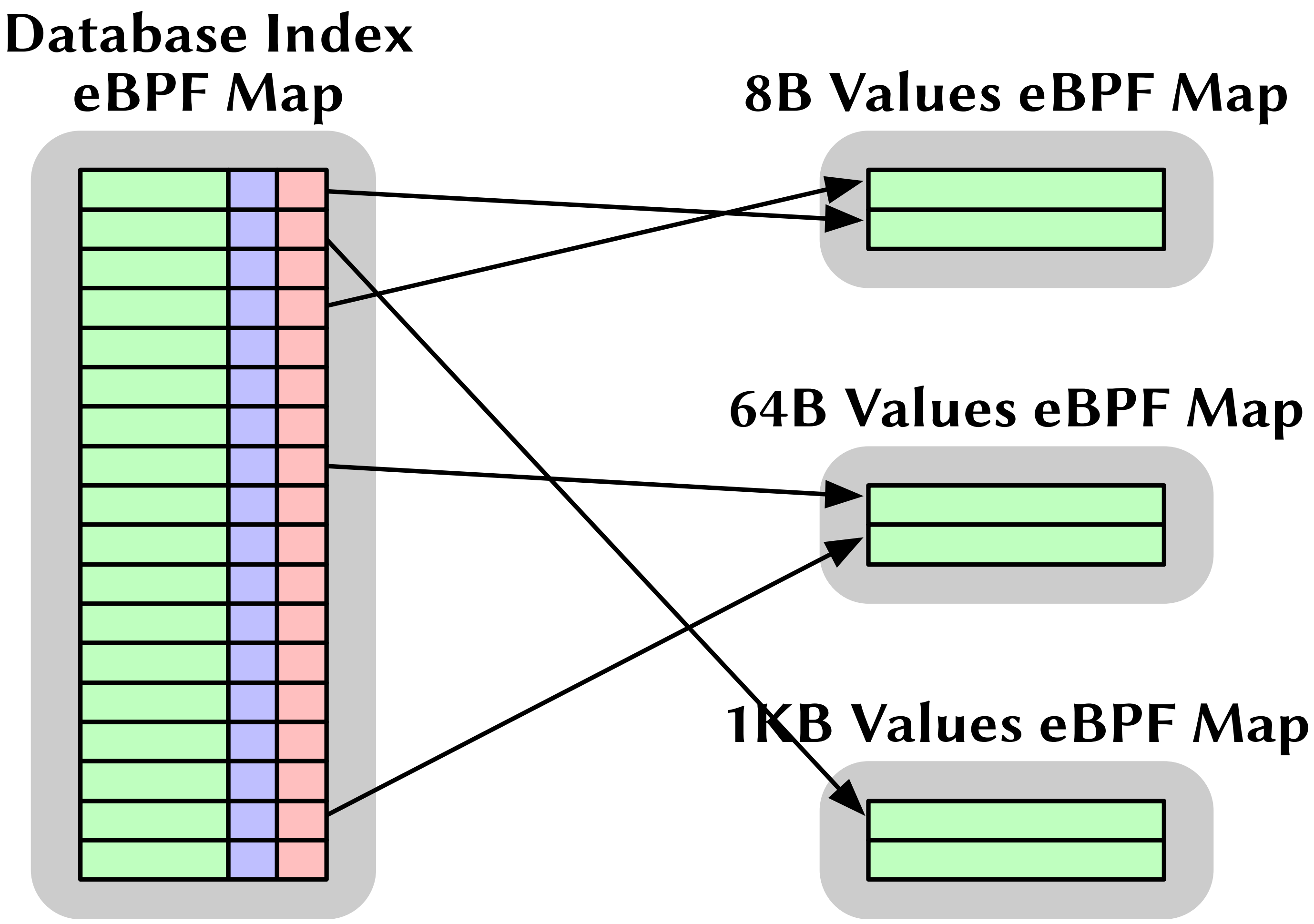
- Goal: Store database contents in kernel-resident thread-safe data structures

- Challenges:
  - No heap ⇒ no `malloc`
  - eBPF maps use fixed size keys and values
  - Verifier limits versions

Storage Management

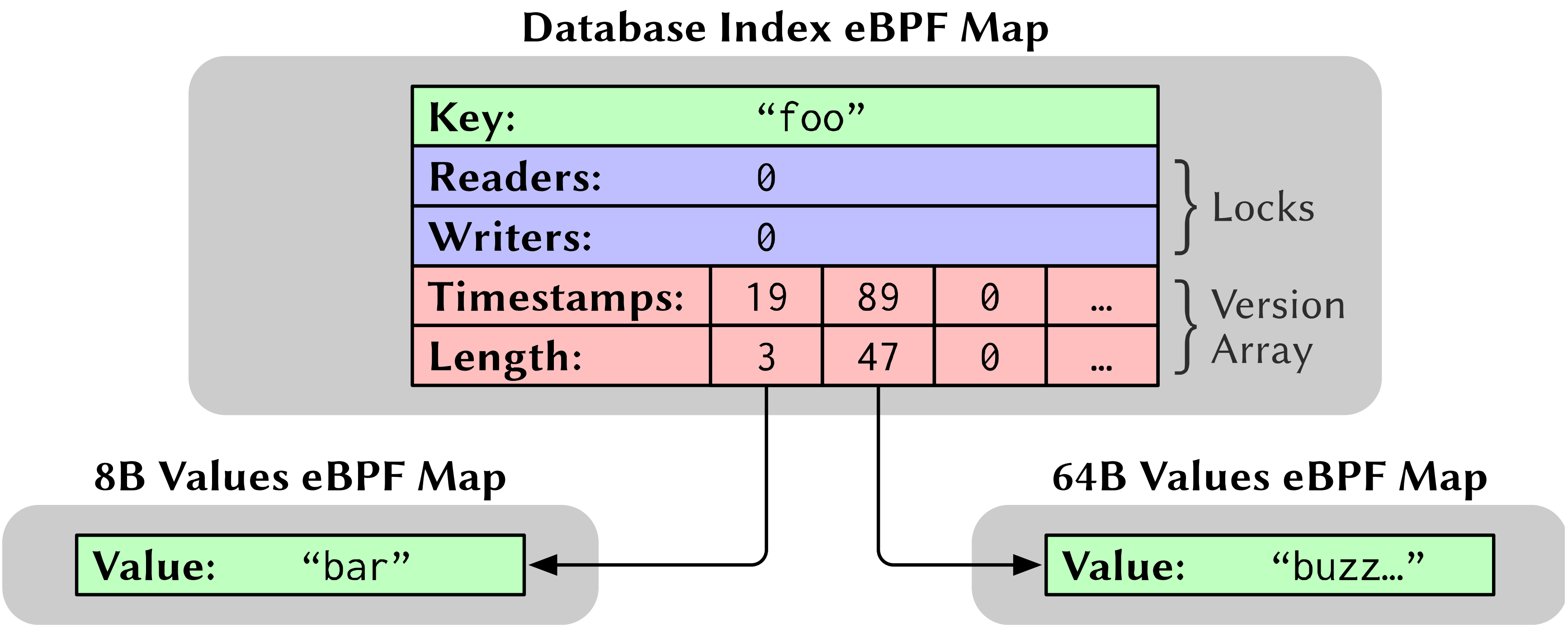
- Goal: Store database contents in kernel-resident thread-safe data structures

- Challenges:
  - No heap ⇒ no malloc
  - eBPF maps use fixed size keys and values
  - Verifier limits versions



Storage Management

Storage Management



Database Index eBPF Map

Key:	"foo"			
Readers:	0			
Writers:	0			
Timestamps:	19	89	0	…
Length:	3	47	0	…

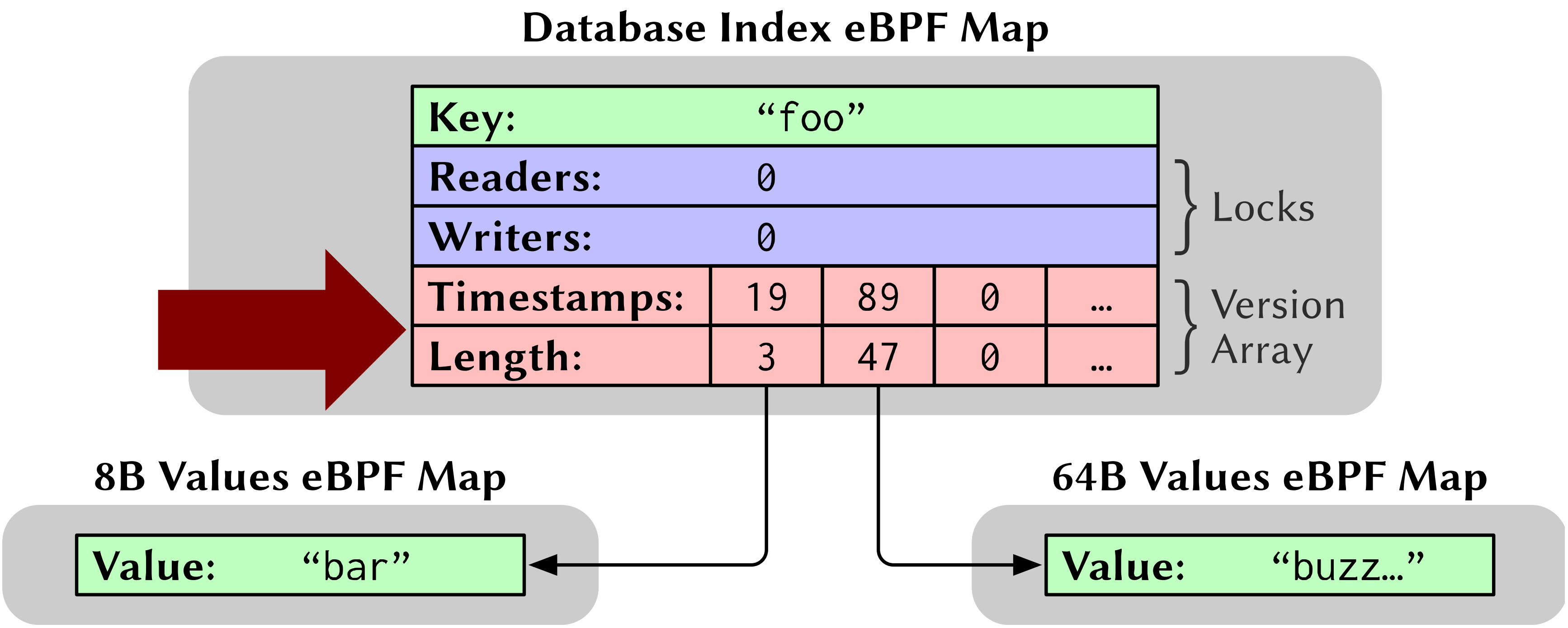
8B Values eBPF Map

Value: "bar"

64B Values eBPF Map

Value: "buzz…"

Storage Management



Database Index eBPF Map

Key:	"foo"			
Readers:	0			
Writers:	0			
Timestamps:	19	89	0	…
Length:	3	47	0	…

8B Values eBPF Map

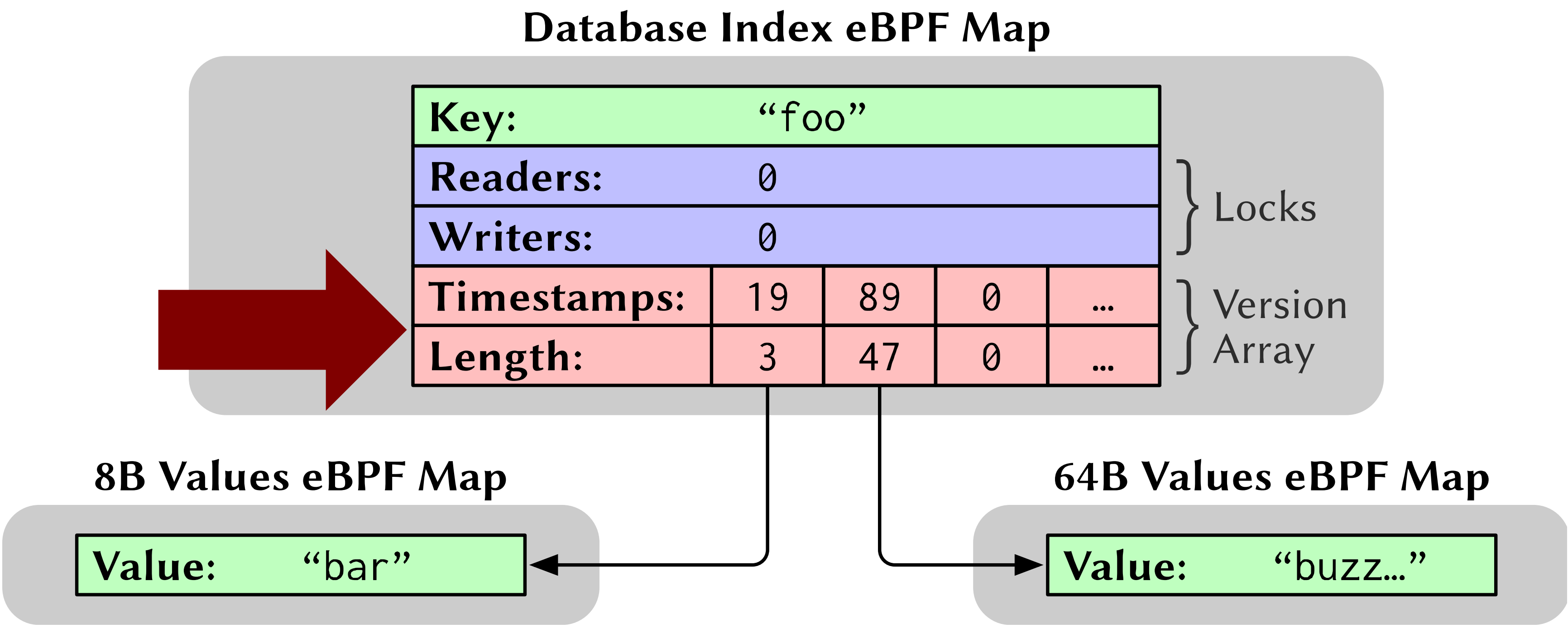
Value: "bar"

64B Values eBPF Map

Value: "buzz…"

Storage Management

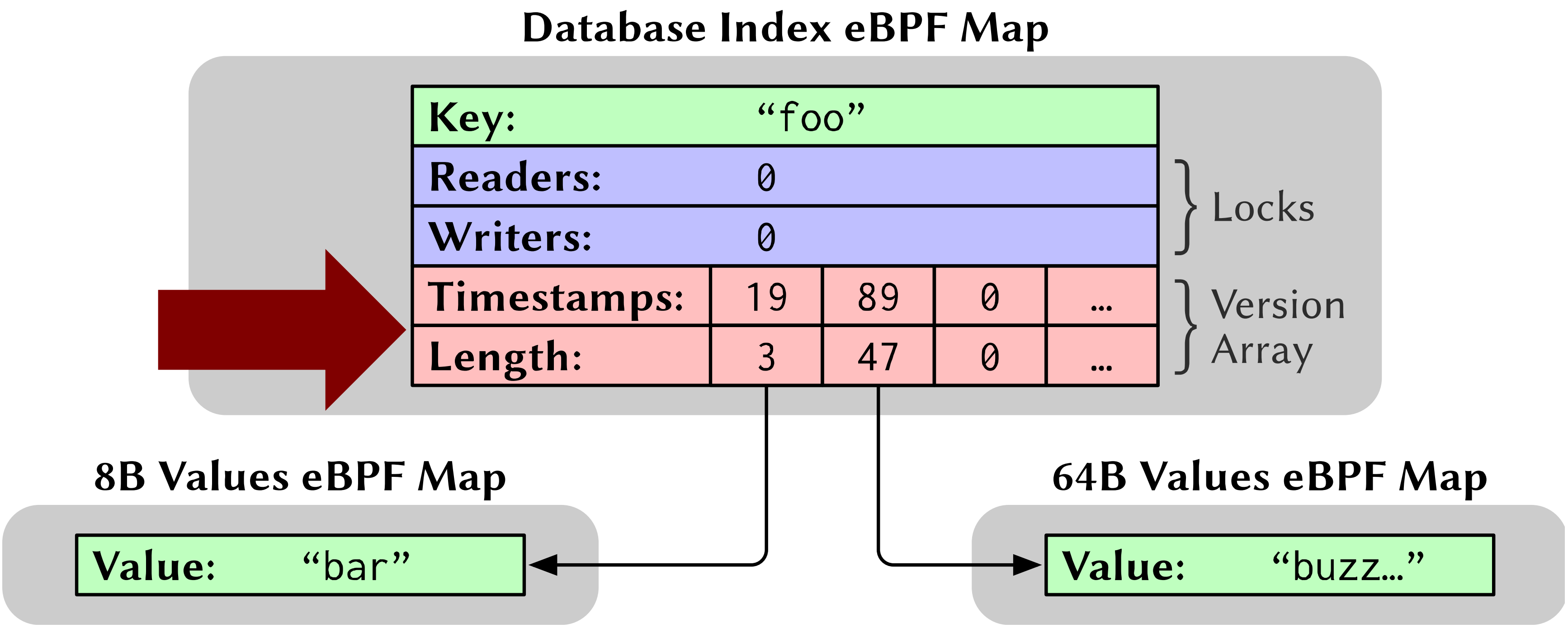
- Bounded, unordered version arrays



Storage Management

- Bounded, unordered version arrays

- Cooperative GC on SET



Database Index eBPF Map

Key: "foo"

Readers: 0

Writers: 0

Locks

Timestamps: 19 | 89 | 0 | …

Length: 3 | 47 | 0 | …

Version Array

8B Values eBPF Map

Value: "bar"

64B Values eBPF Map

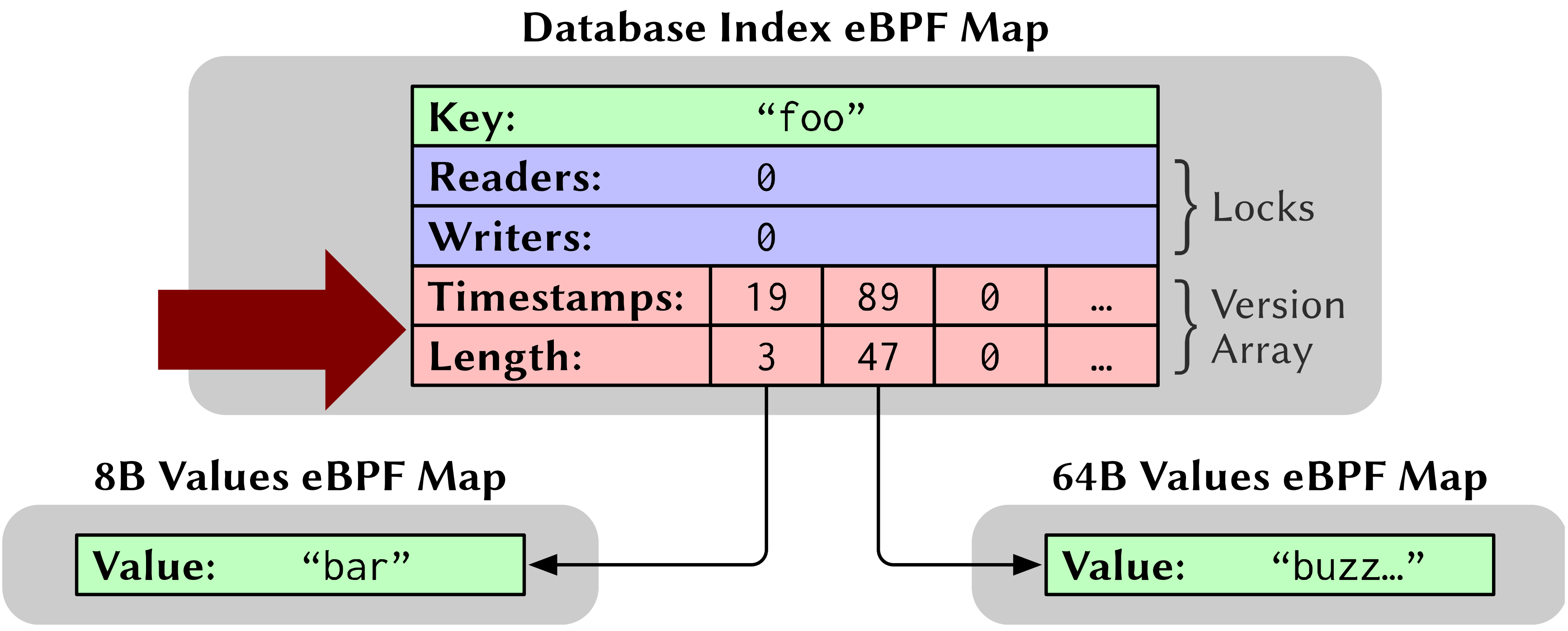
Value: "buzz…"

Storage Management

- Bounded, unordered version arrays

- Cooperative GC on SET

- Database contents separate from DBMS logic



Database Index eBPF Map

Key:	"foo"			
Readers:	0			
Writers:	0			
Timestamps:	19	89	0	...
Length:	3	47	0	...

Aakash Goel et al. Fast Database Restarts at Facebook. *SIGMOD*. 2014.

Transaction Management

Transaction Management

- Goal: Implement concurrency control protocol to allow multi-statement transactions that ensure ACID properties

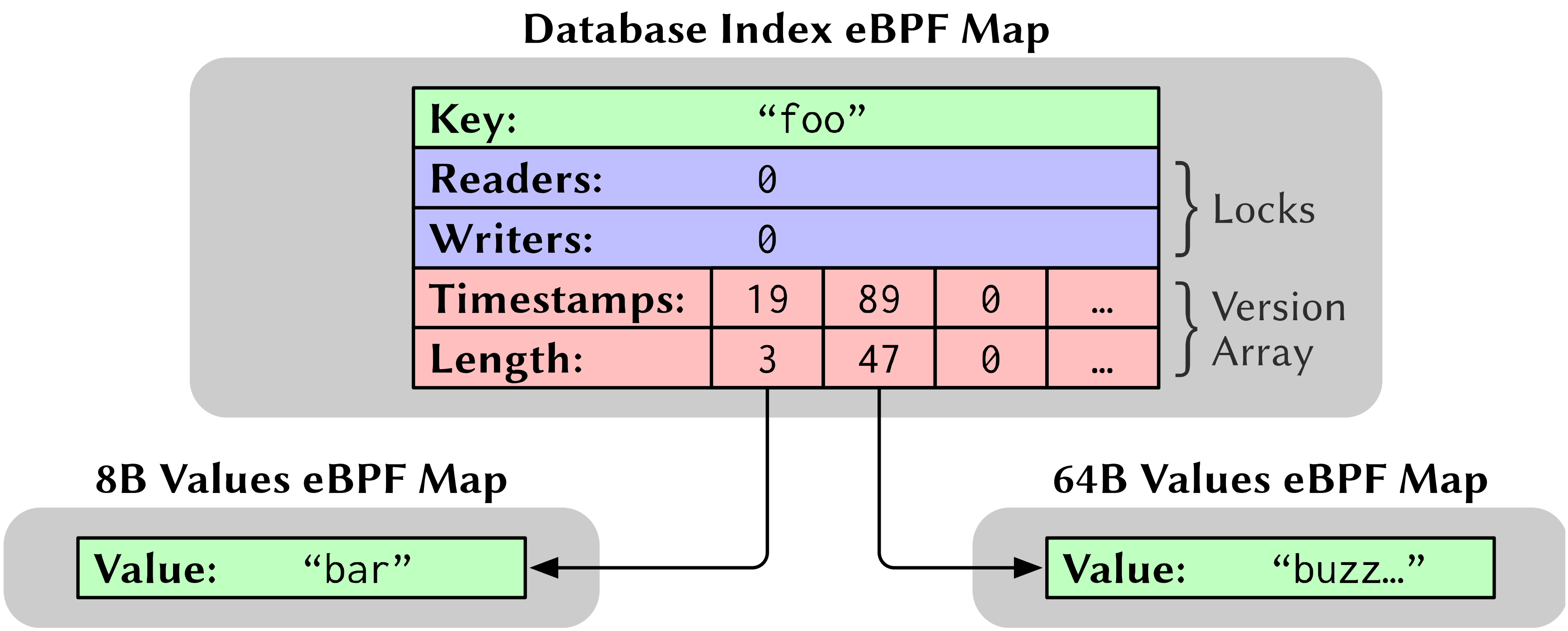
Transaction Management

- Goal: Implement concurrency control protocol to allow multi-statement transactions that ensure ACID properties

- Challenges:
  - Restrictive atomic primitives
  - Boundedness limits spinning
  - eBPF program execution cannot yield

Transaction Management

Transaction Management



Database Index eBPF Map

Key:	“foo”			
Readers:	0			
Writers:	0			
Timestamps:	19	89	0	…
Length:	3	47	0	…

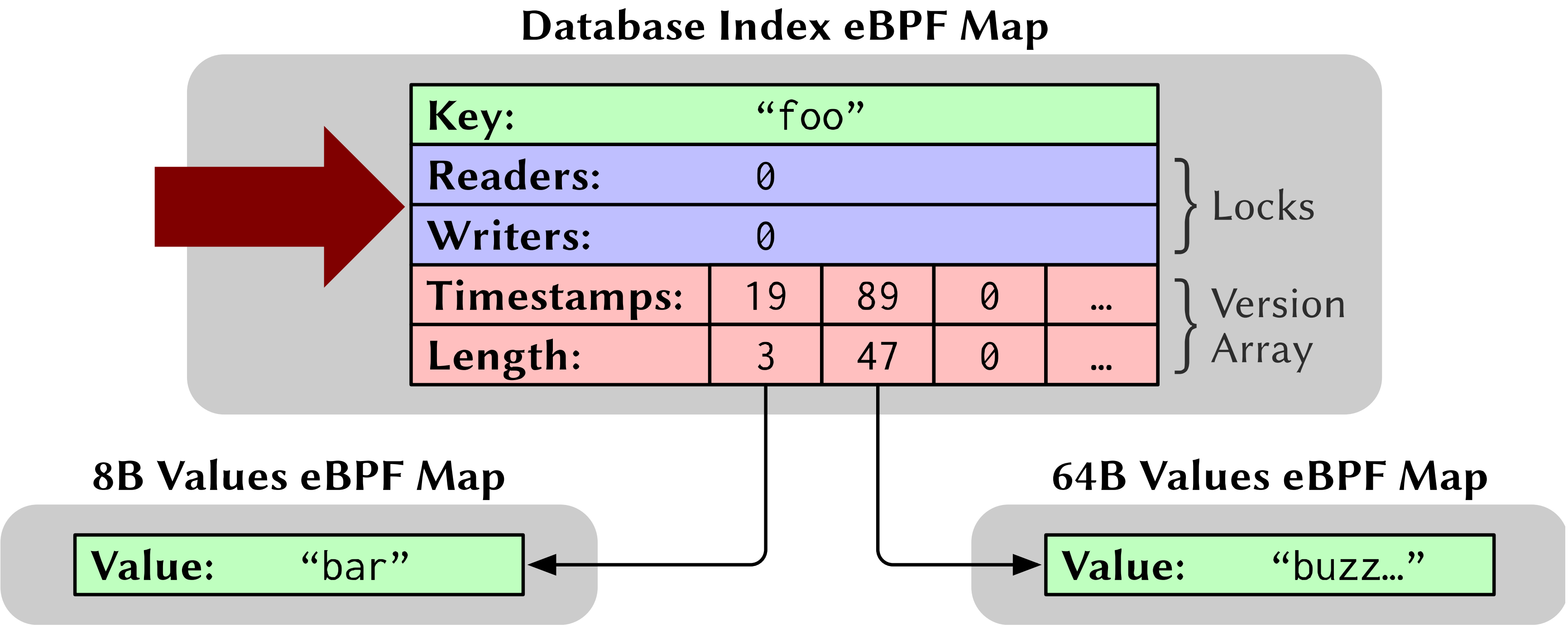
8B Values eBPF Map

Value: “bar”

64B Values eBPF Map

Value: “buzz…”

Transaction Management



Database Index eBPF Map

Key:	"foo"			
Readers:	0			
Writers:	0			
Timestamps:	19	89	0	…
Length:	3	47	0	…

8B Values eBPF Map

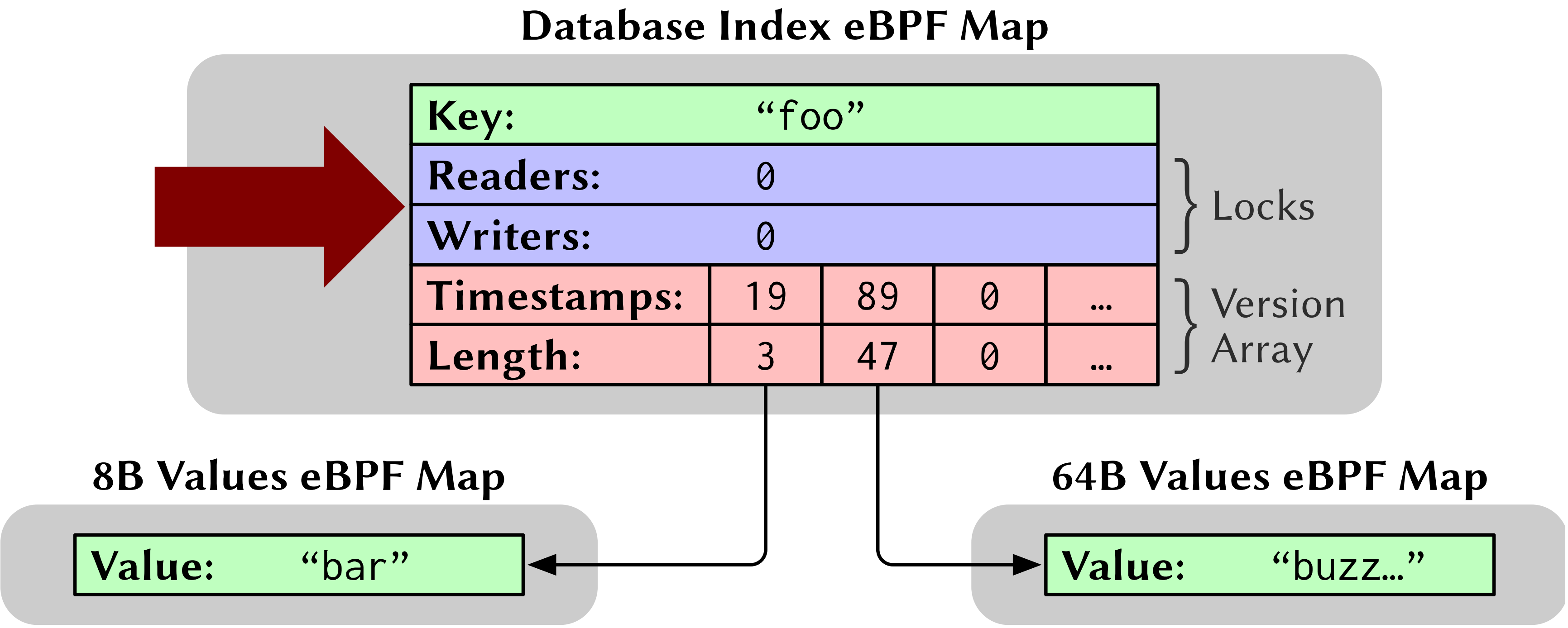
Value: "bar"

64B Values eBPF Map

Value: "buzz…"

Transaction Management

- Strict MV2PL



Database Index eBPF Map

Key:	“foo”			
Readers:	0			
Writers:	0			
Timestamps:	19	89	0	…
Length:	3	47	0	…

Locks

Version Array

8B Values eBPF Map

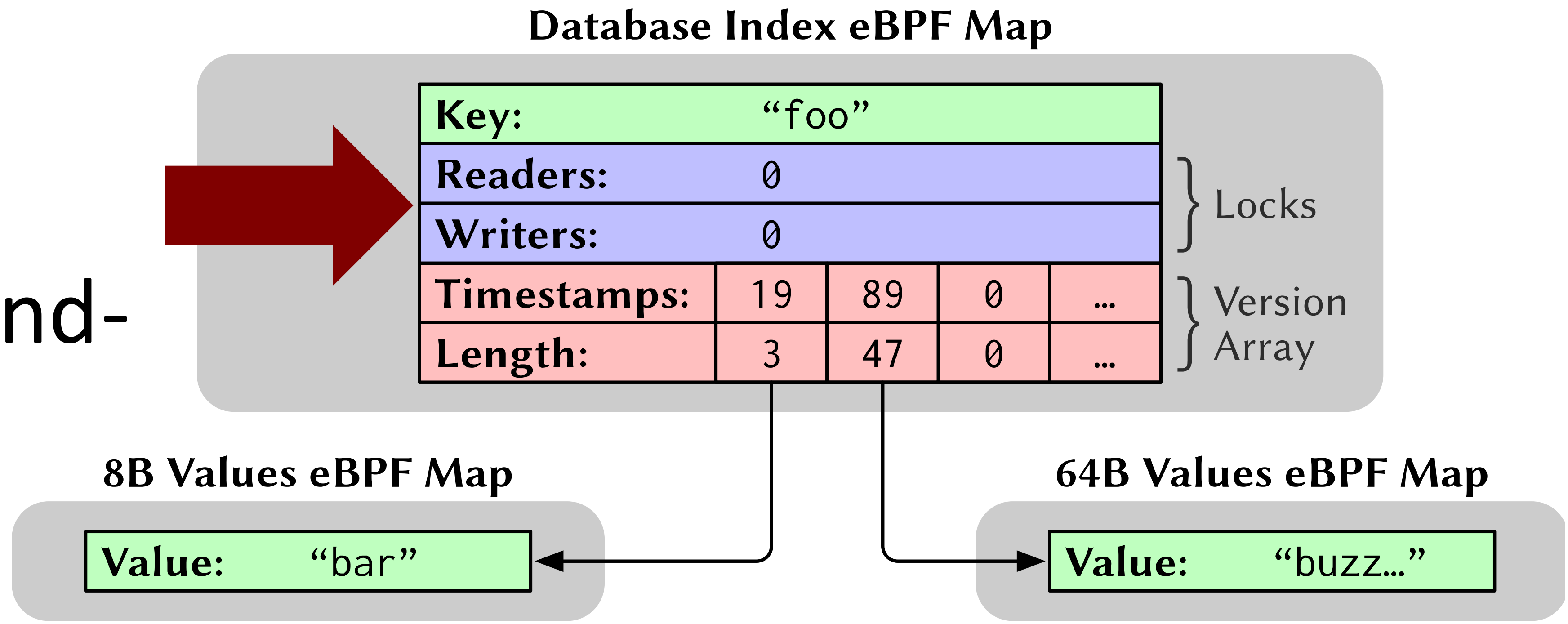
Value: “bar”

64B Values eBPF Map

Value: “buzz…”

Transaction Management

- Strict MV2PL
- No-wait instead of wound-wait or wait-die



Database Index eBPF Map

Key:	“foo”			
Readers:	0			
Writers:	0			
Timestamps:	19	89	0	…
Length:	3	47	0	…

8B Values eBPF Map

Value: “bar”

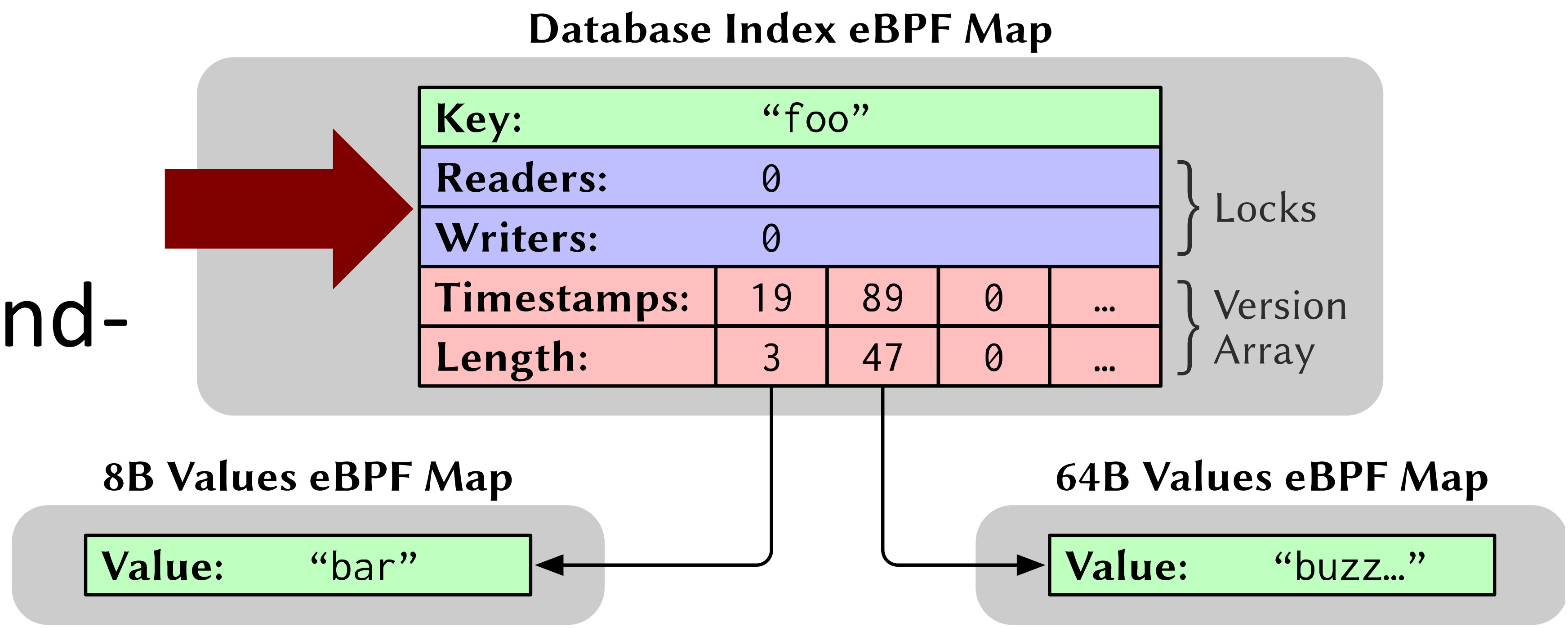
64B Values eBPF Map

Value: “buzz…”

Yingjun Wu et al. An Empirical Evaluation of In-Memory Multi-Version Concurrency Control. *VLDB*. 2017.

Transaction Management

- Strict MV2PL
- No-wait instead of wound-wait or wait-die
- Read-only optimizations



Database Index eBPF Map

Key:	“foo”			
Readers:	0			
Writers:	0			
Timestamps:	19	89	0	…
Length:	3	47	0	…

8B Values eBPF Map

Value: “bar”

64B Values eBPF Map

Value: “buzz…”

Philip A. Bernstein et al. *Concurrency Control and Recovery in Database Systems*. 1987.

Yingjun Wu et al. An Empirical Evaluation of In-Memory Multi-Version Concurrency Control. *VLDB*. 2017.

Write-Ahead Logging

Write-Ahead Logging

- Goal: Persist database contents to disk both through write-ahead logging

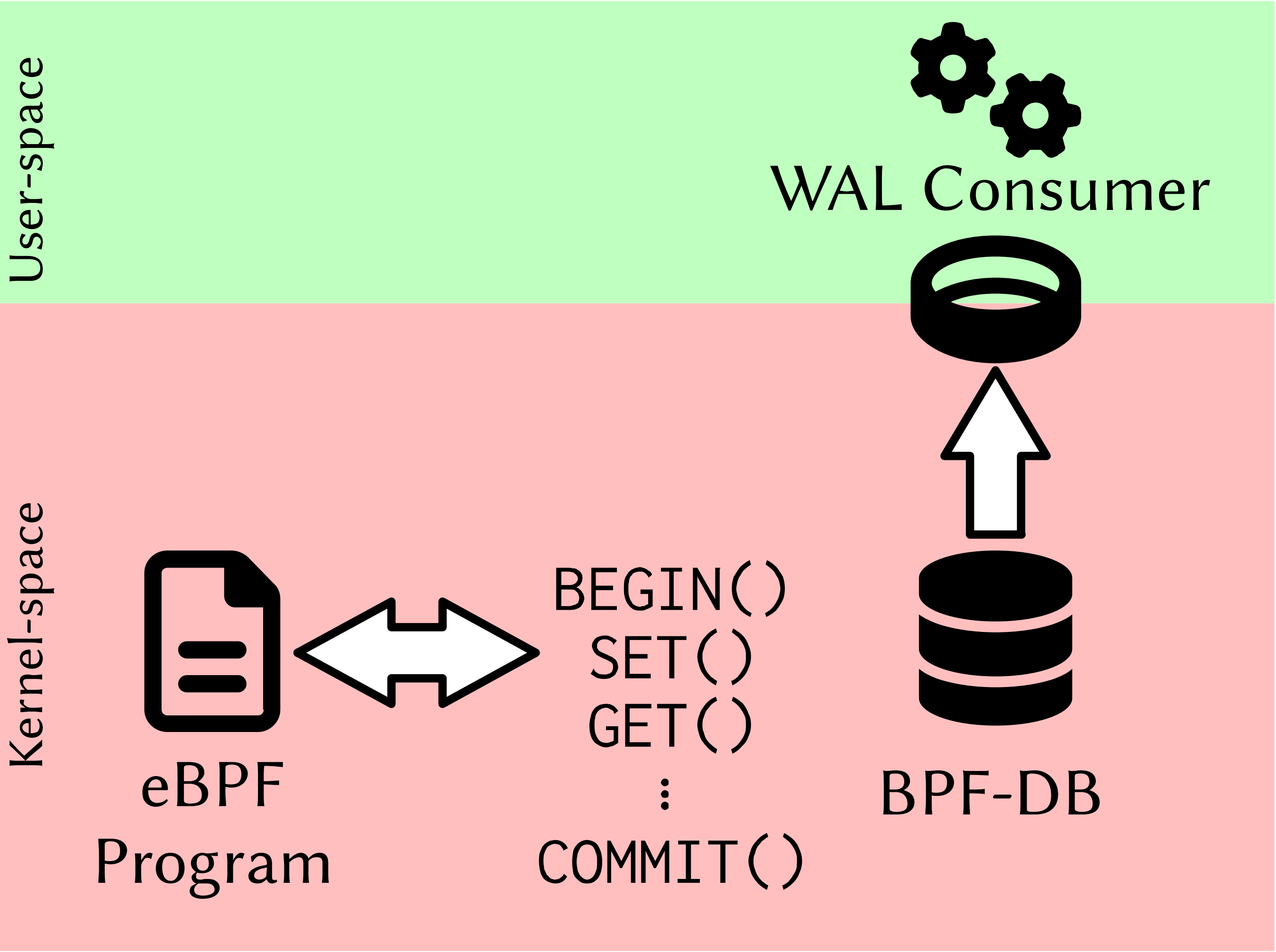
Write-Ahead Logging

- Goal: Persist database contents to disk both through write-ahead logging

- Challenges:
  - eBPF programs cannot initiate disk access
  - Database contents are stored in kernel-resident data

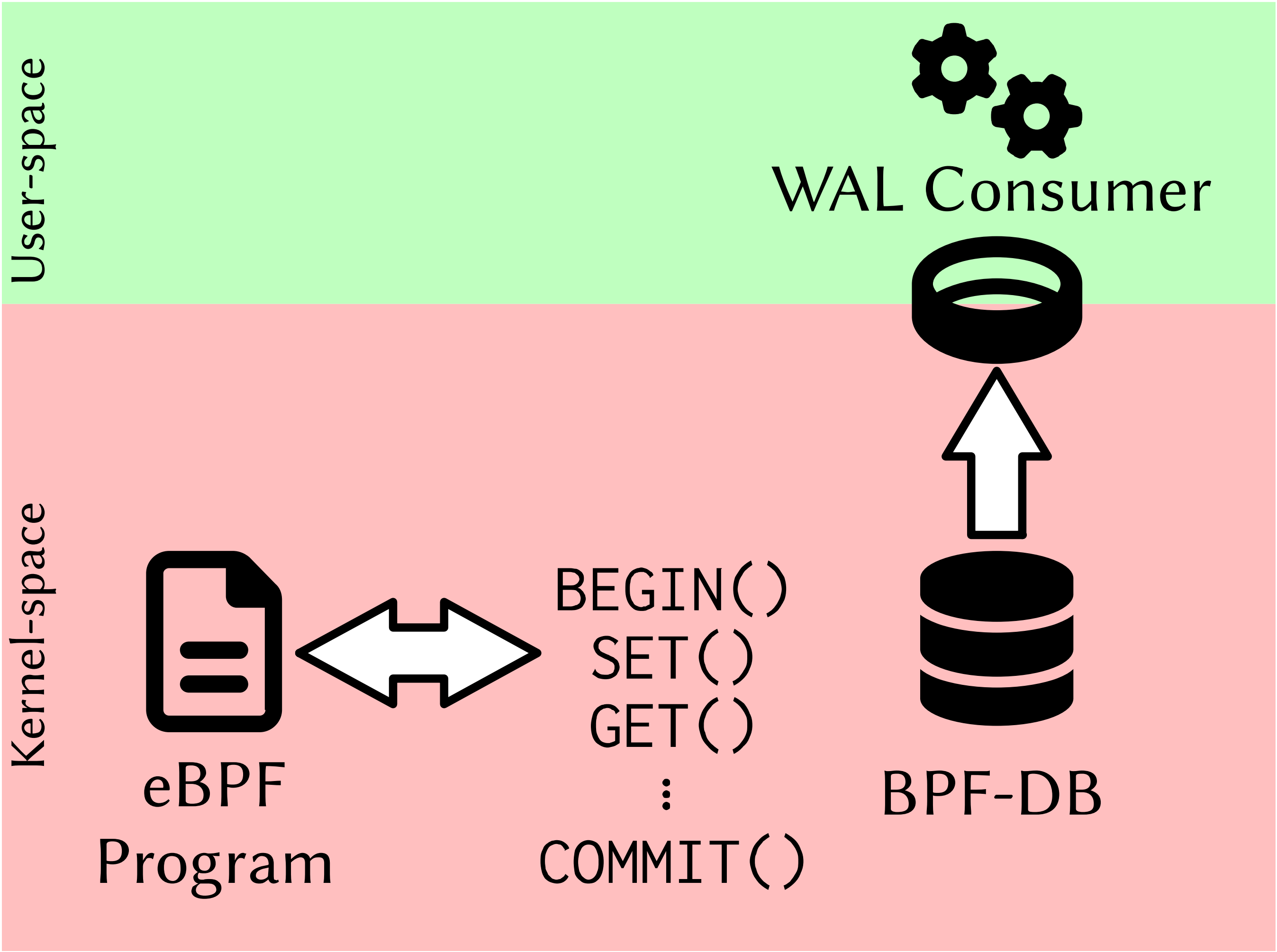
Write-Ahead Logging

Write-Ahead Logging



Write-Ahead Logging

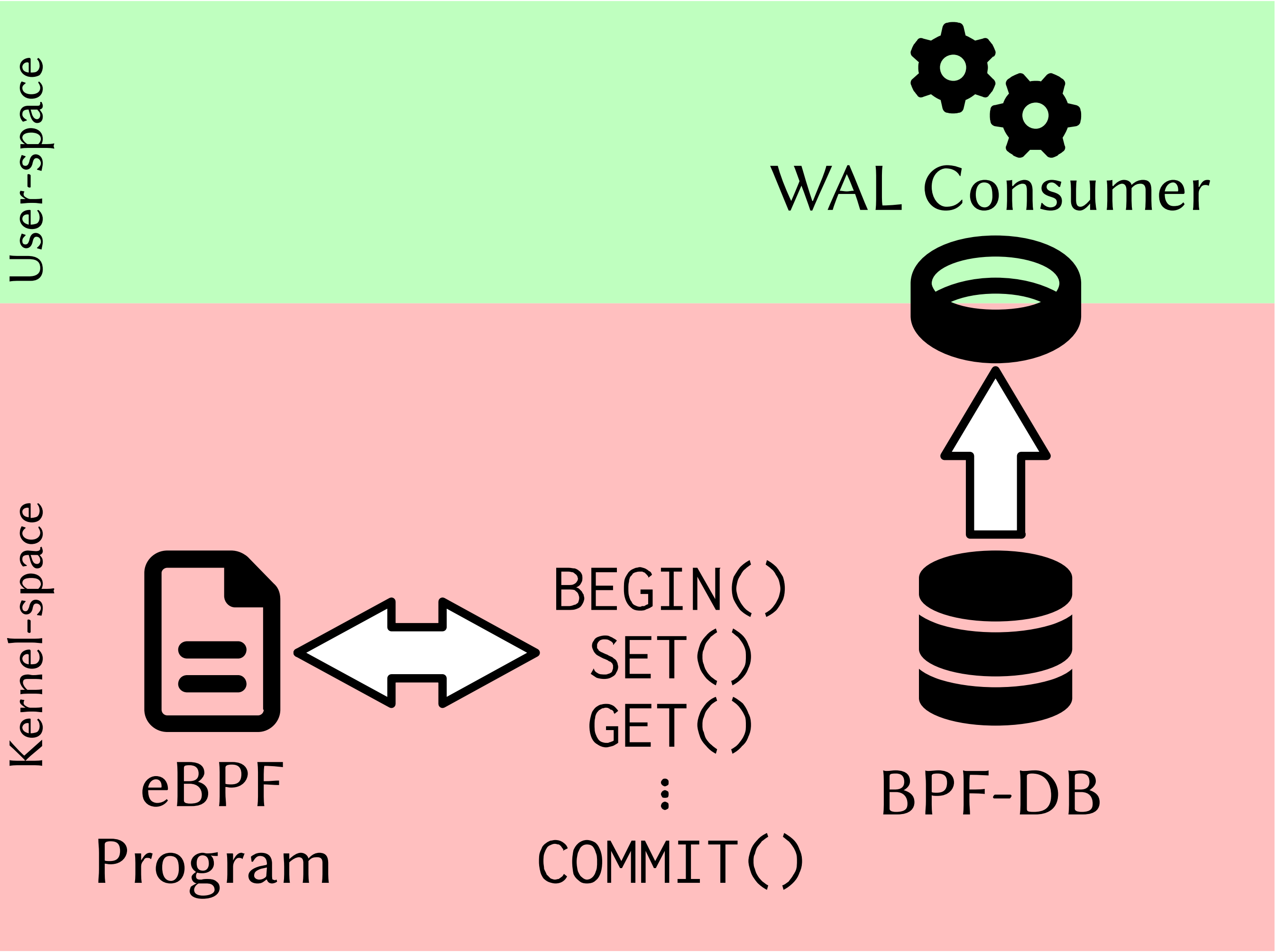
- Logical logging via ring buffer to user-space



Write-Ahead Logging

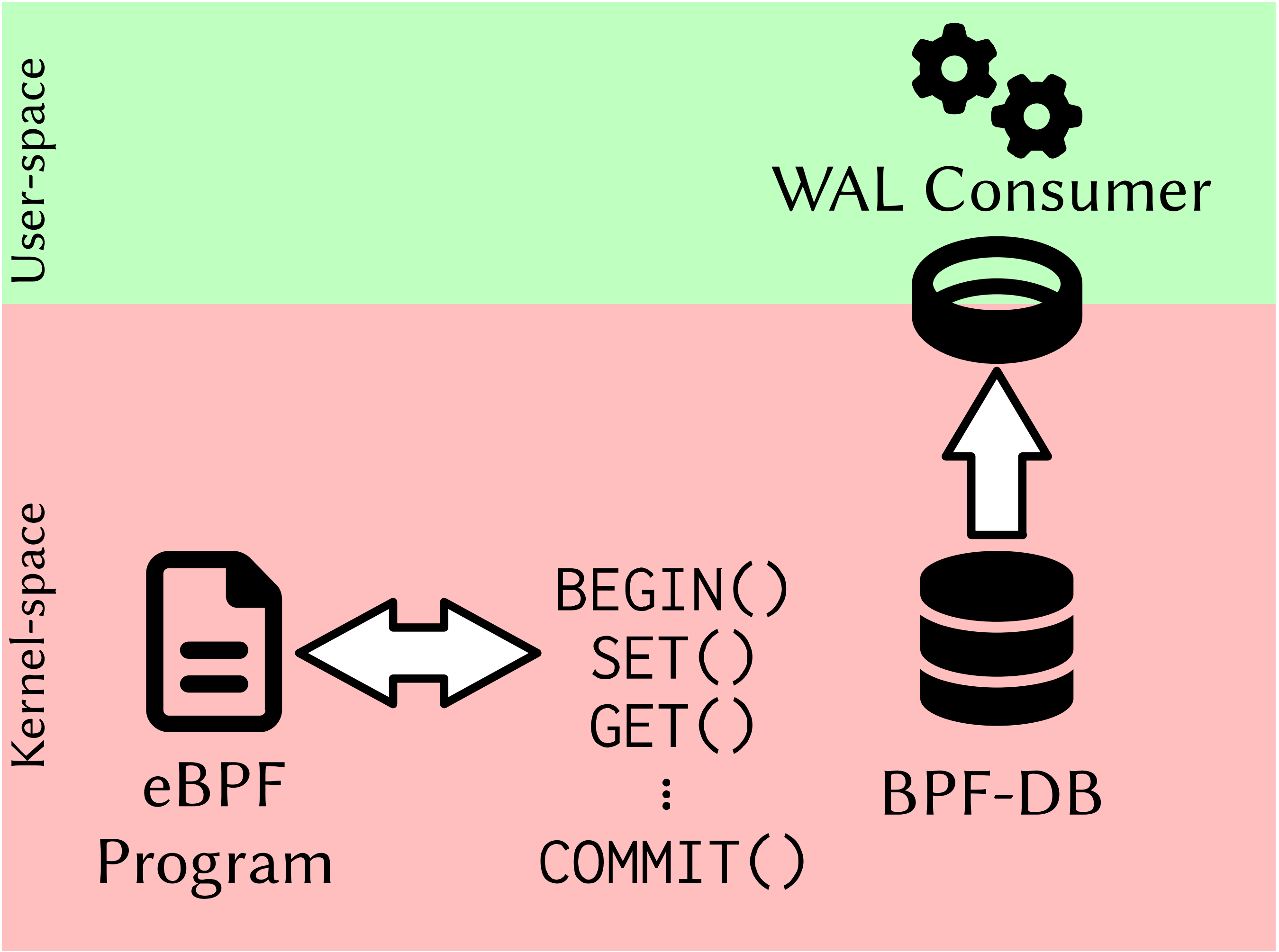
- Logical logging via ring buffer to user-space

- User-space can persist on disk or over network



Write-Ahead Logging

- Logical logging via ring buffer to user-space
- User-space can persist on disk or over network
- Checkpointing requires to quiesce the DBMS



Conclusion

Conclusion

- eBPF’s verifier introduces new design constraints beyond traditional software engineering and runtime performance limits

Conclusion

- eBPF’s verifier introduces new design constraints beyond traditional software engineering and runtime performance limits

- Our eBPF DBMS benefits from storing database contents in kernel-resident data structures and enables new classes of eBPF applications

Conclusion

- eBPF's verifier introduces new design constraints beyond traditional software engineering and runtime performance limits

- Our eBPF DBMS benefits from storing database contents in kernel-resident data structures and enables new classes of eBPF applications

- Adaptation generates a customized DBMS for the client application