PAPER REVIEW: LOG-STRUCTURED PROTOCOLS IN DELOS

Guided by: Prof. Abhilash Jindal

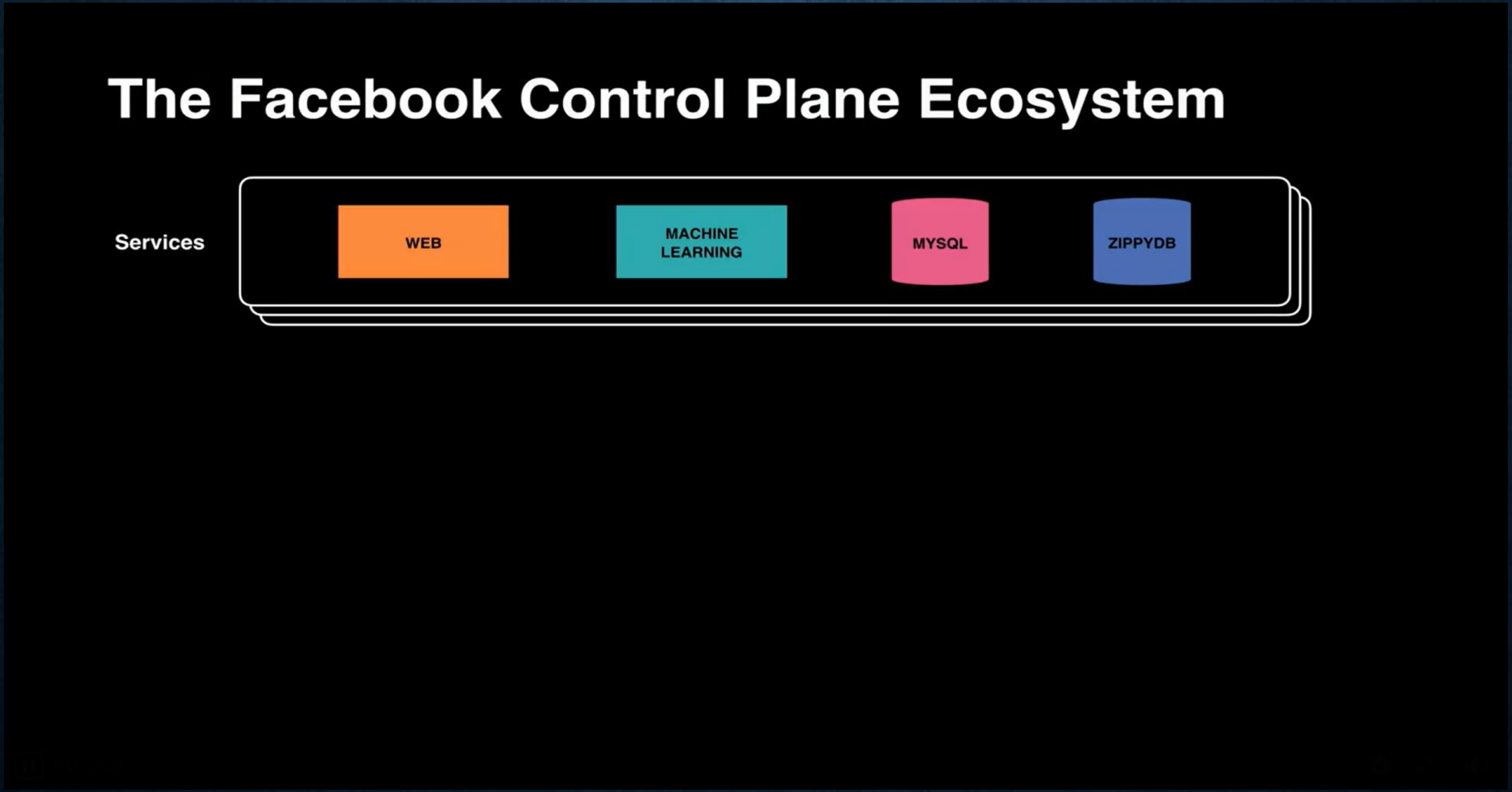
Presented by: Ramita Sardana

AGENDA

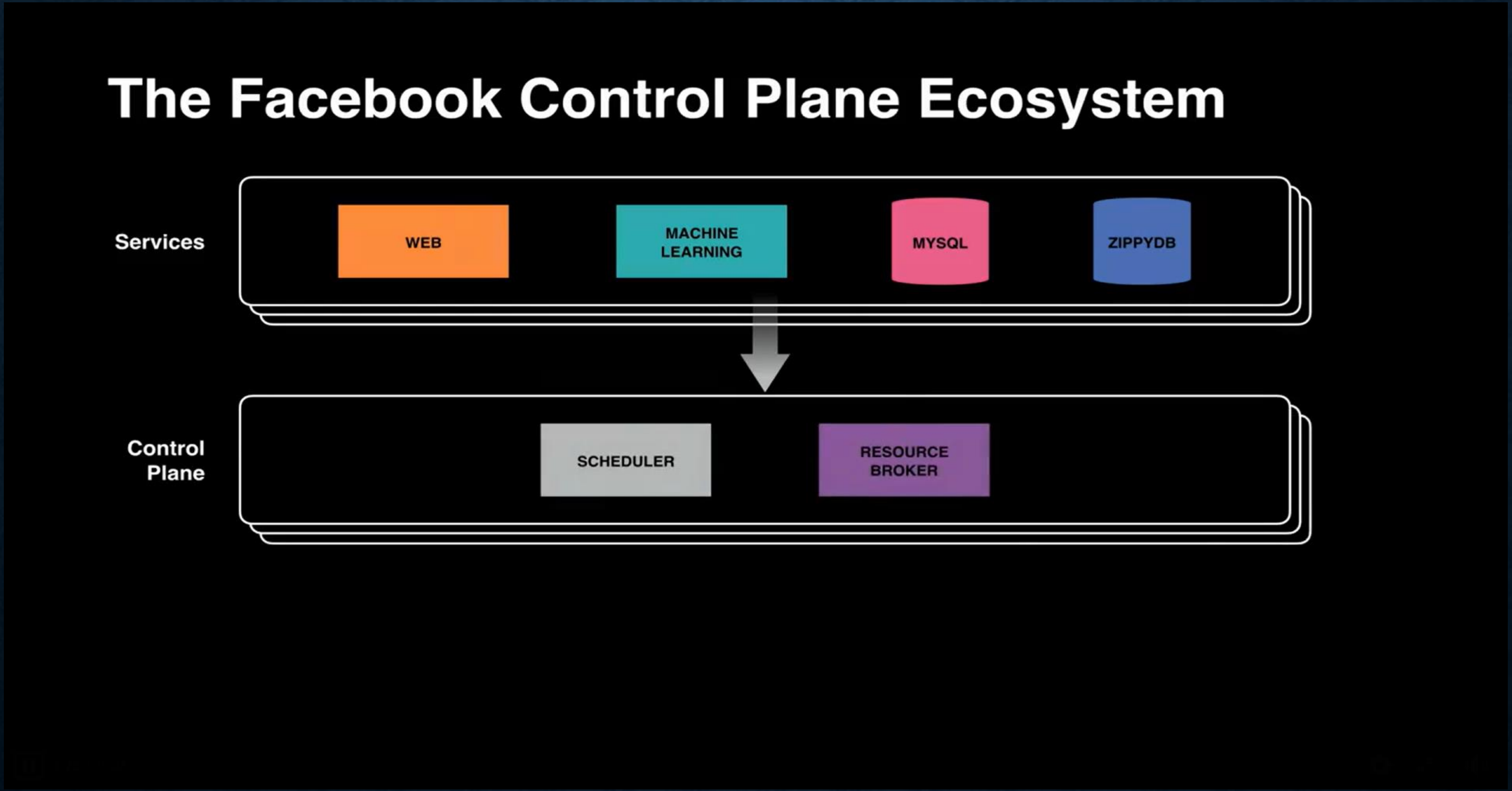
- Background and Motivation
- Contributions
  - Log-structured protocols design
  - Implementation of Nine Log-structured protocols
  - Two production databases using the abstraction
- Benefits
- Evaluation

BACKGROUND AND MOTIVATION

BACKGROUND

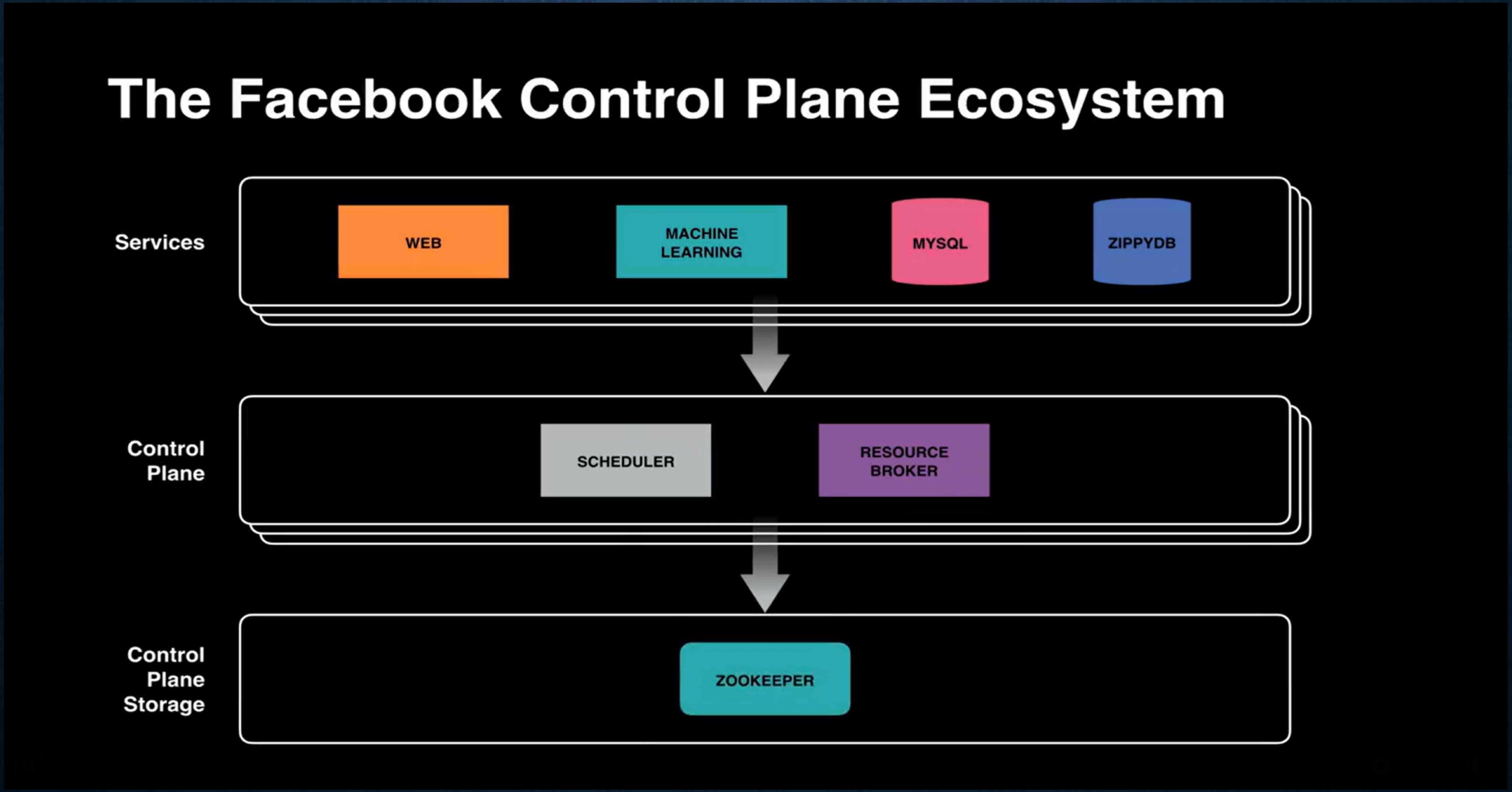

The Facebook Control Plane Ecosystem
Services
WEB
MACHINE LEARNING
MYSQL
ZIPPYDB

BACKGROUND

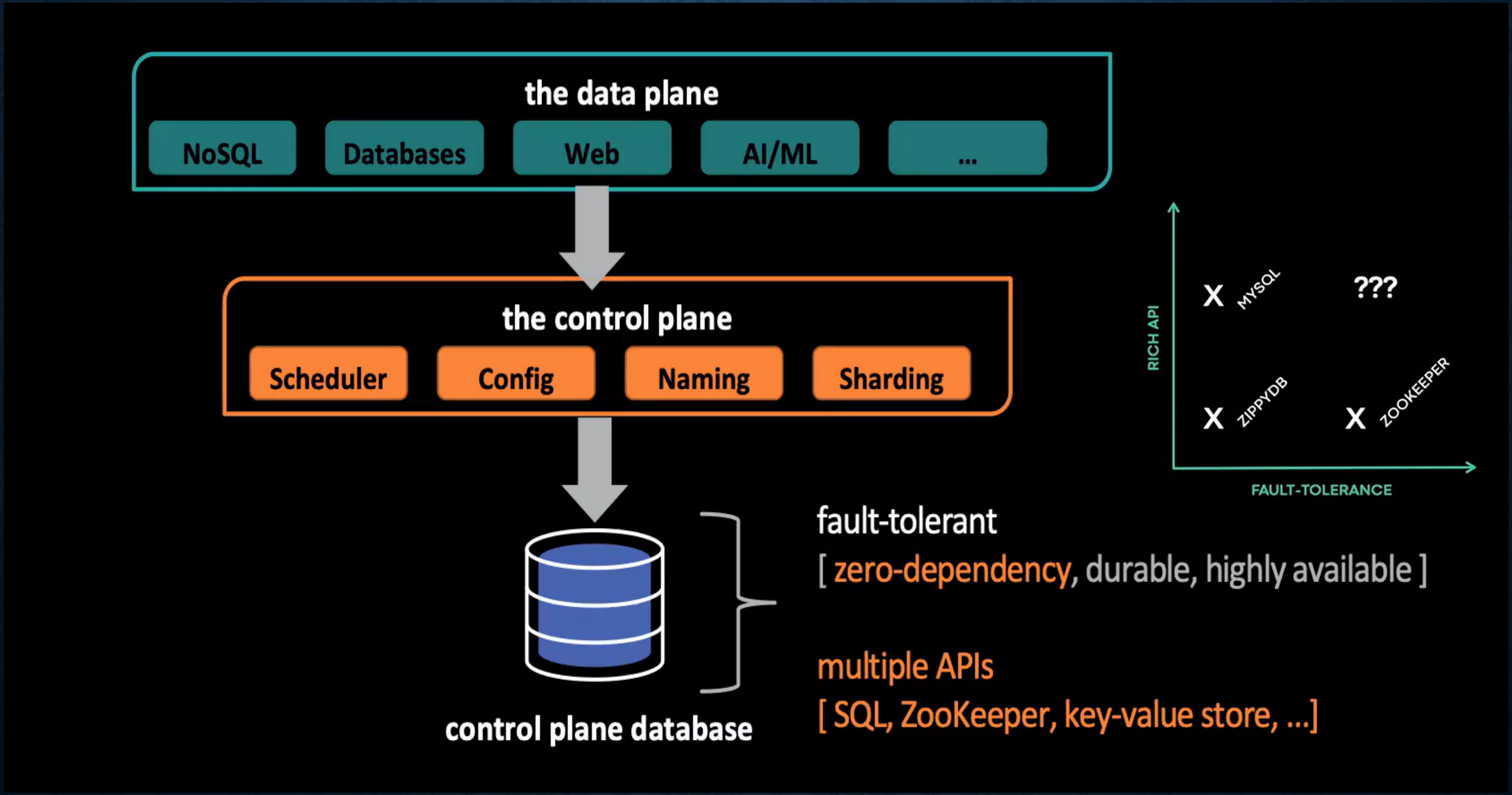


The Facebook Control Plane Ecosystem
Services
WEB
MACHINE LEARNING
MYSQL
ZIPPYDB
Control Plane
SCHEDULER
RESOURCE BROKER

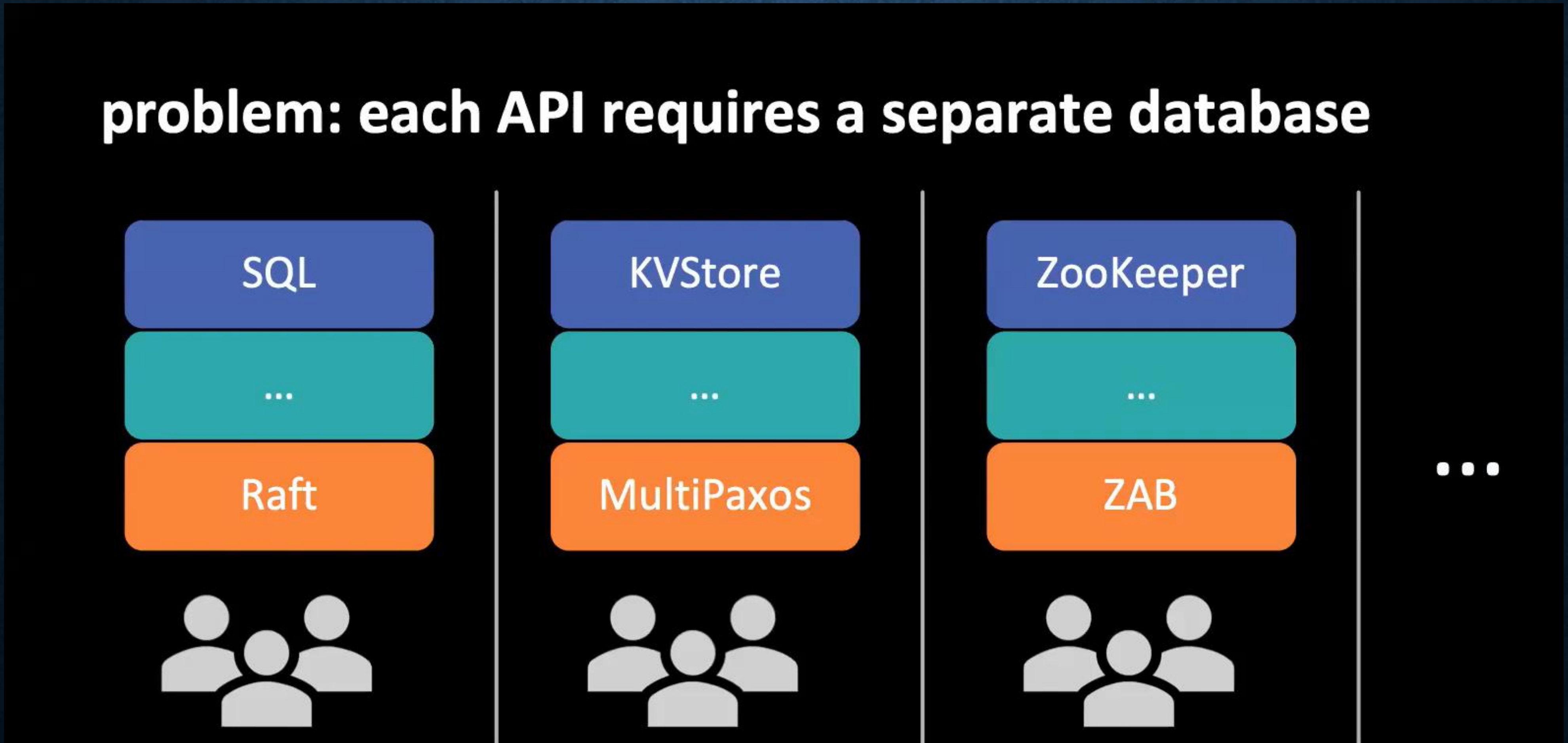
BACKGROUND


The Facebook Control Plane Ecosystem
Services
WEB
MACHINE LEARNING
MYSQL
ZIPPYDB
Control Plane
SCHEDULER
RESOURCE BROKER
Control Plane Storage
ZOOKEEPER

BACKGROUND: CONTROL PLANE STORAGE REQUIREMENTS

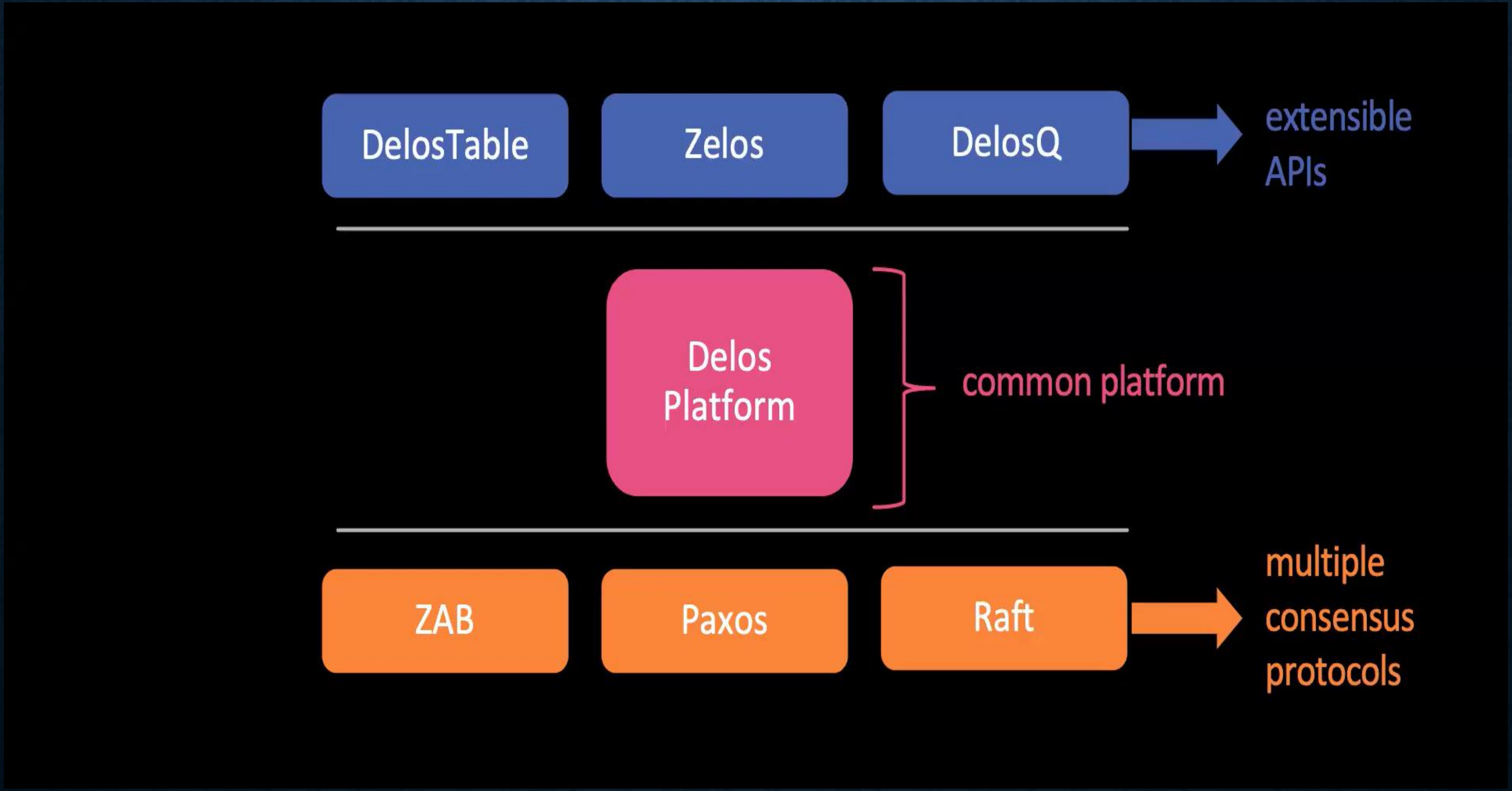


BACKGROUND: CONTROL PLANE STORAGE REQUIREMENTS

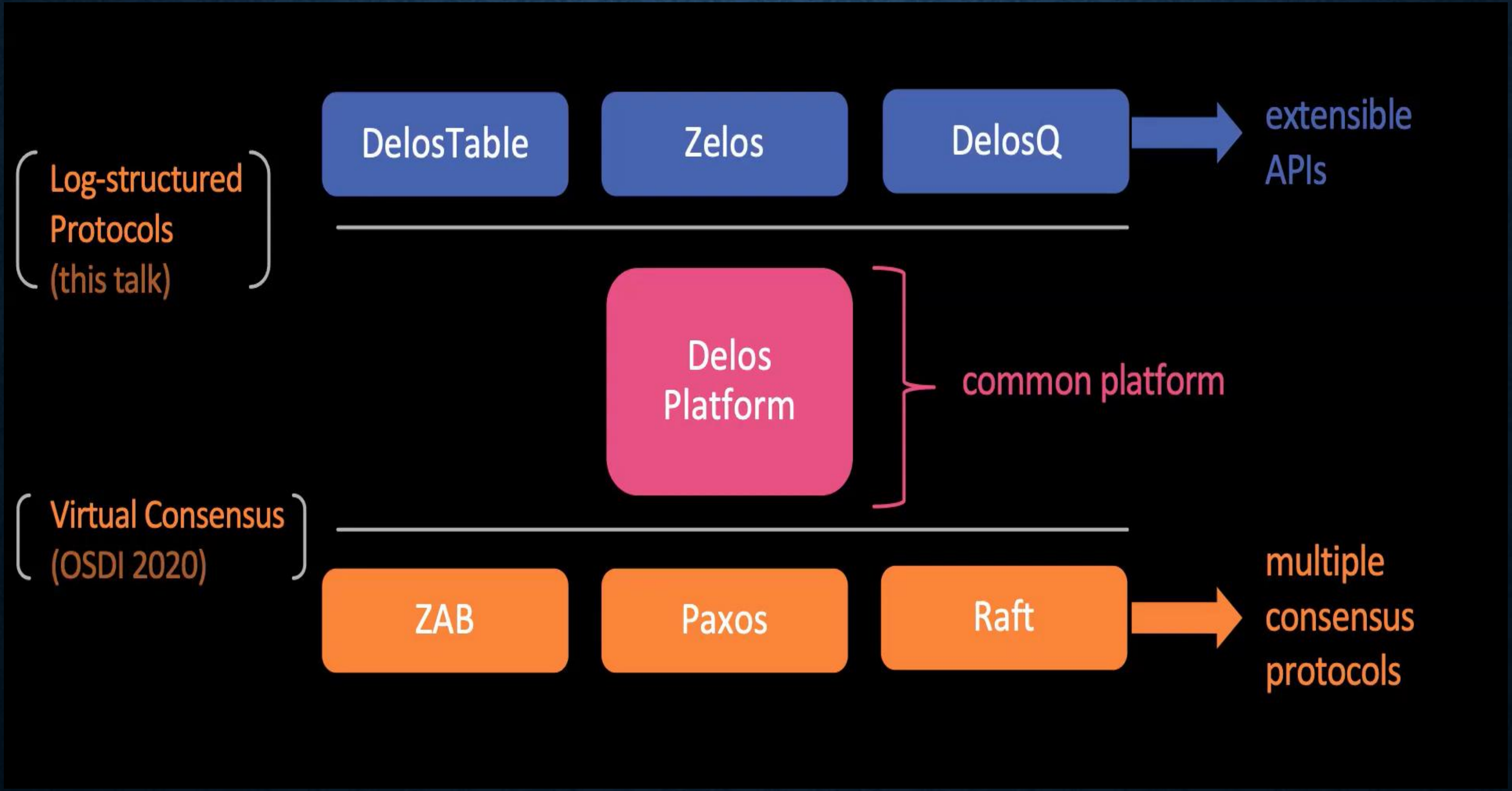


... but most databases look similar!

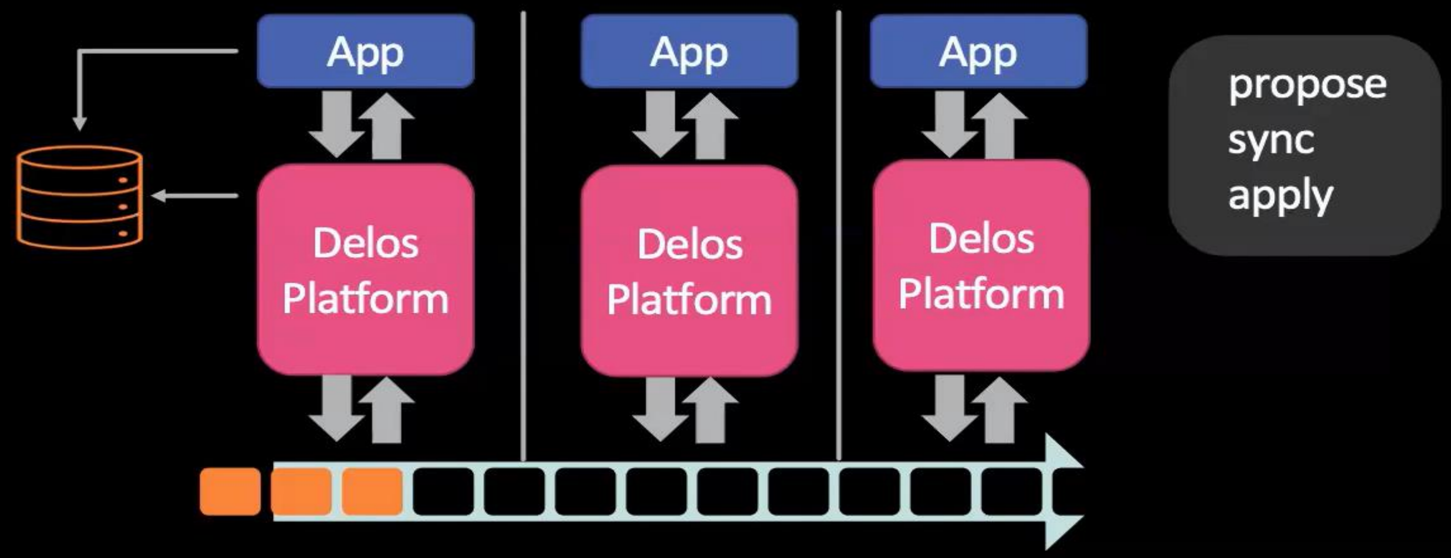
DELOS PLATFORM: HOURGLASS ARCHITECTURE



DELOS PLATFORM: HOURGLASS ARCHITECTURE

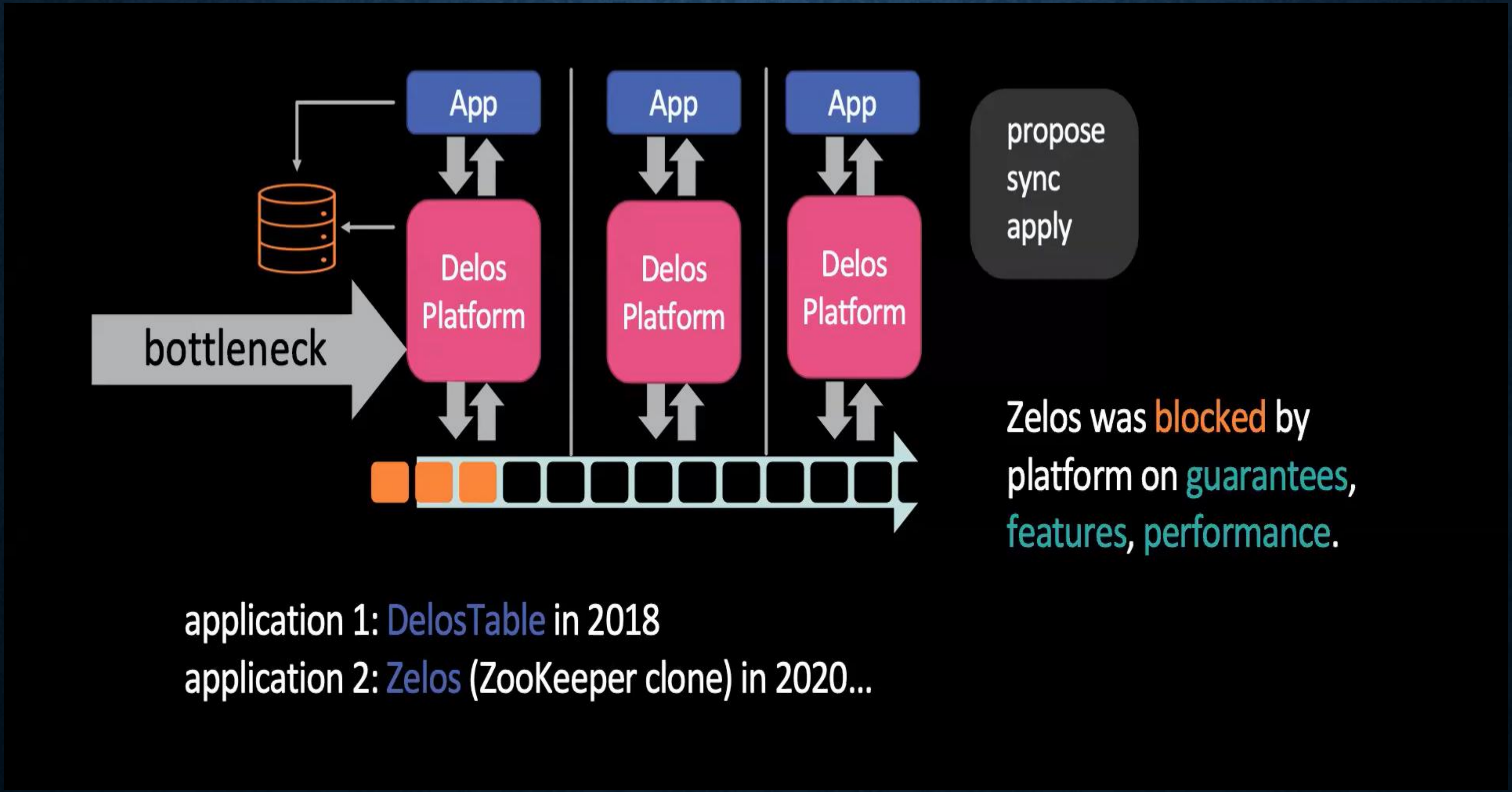


USING DELOS

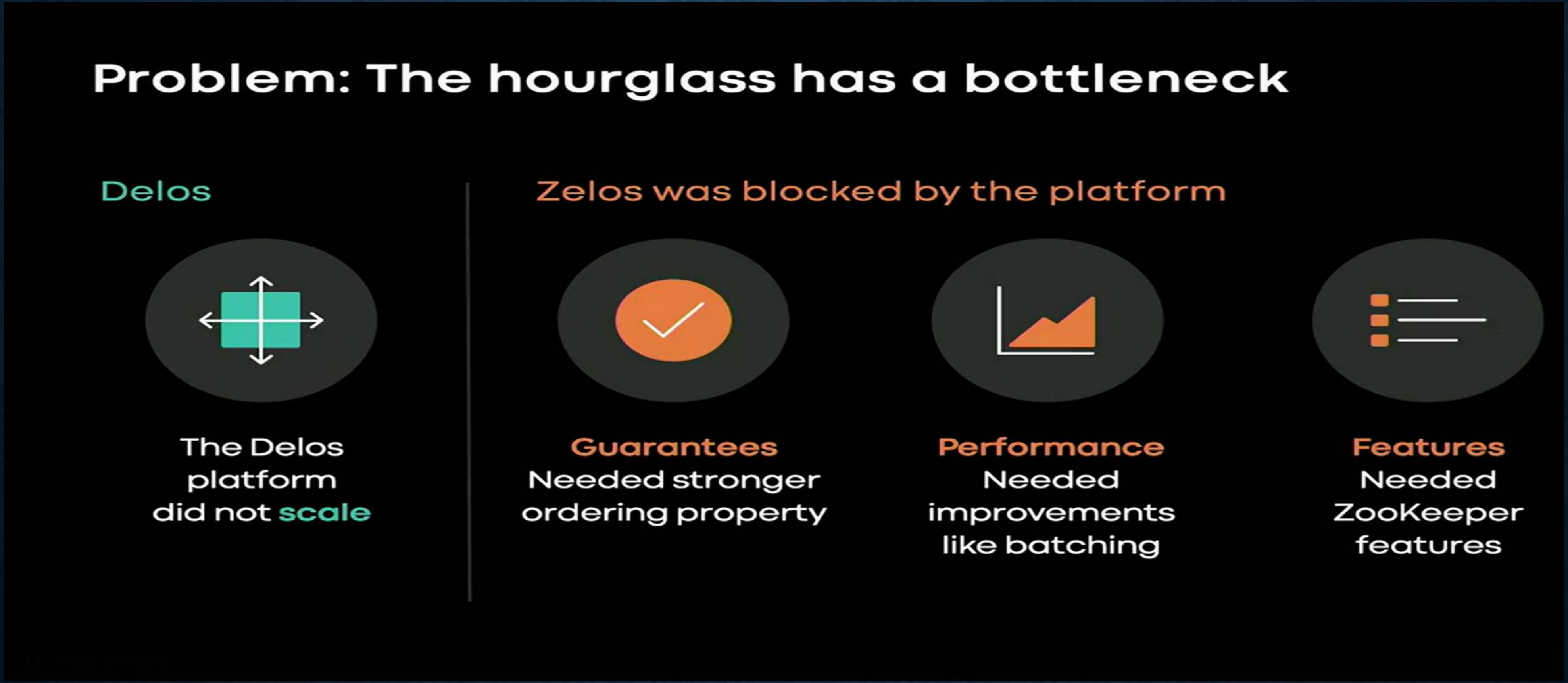


application 1: DelosTable in 2018
application 2: Zelos (ZooKeeper clone) in 2020...

USING DELOS


App
App
App
propose
sync
apply
Delos Platform
Delos Platform
Delos Platform
bottleneck
Zelos was blocked by platform on guarantees, features, performance.
application 1: DelosTable in 2018
application 2: Zelos (ZooKeeper clone) in 2020...

USING DELOS



The platform became a bottleneck not in terms of scaling or throughput but in terms of developer productivity.

“

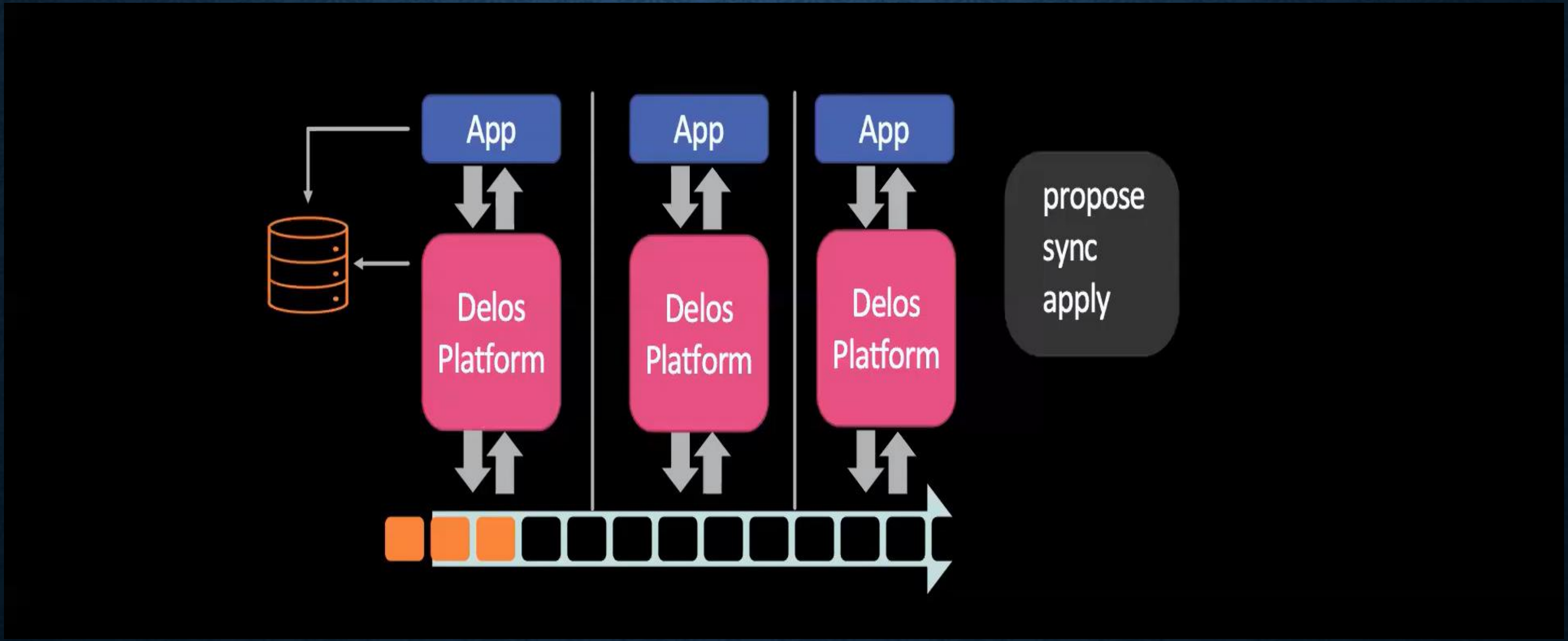
USE A GOOD IDEA AGAIN.

”

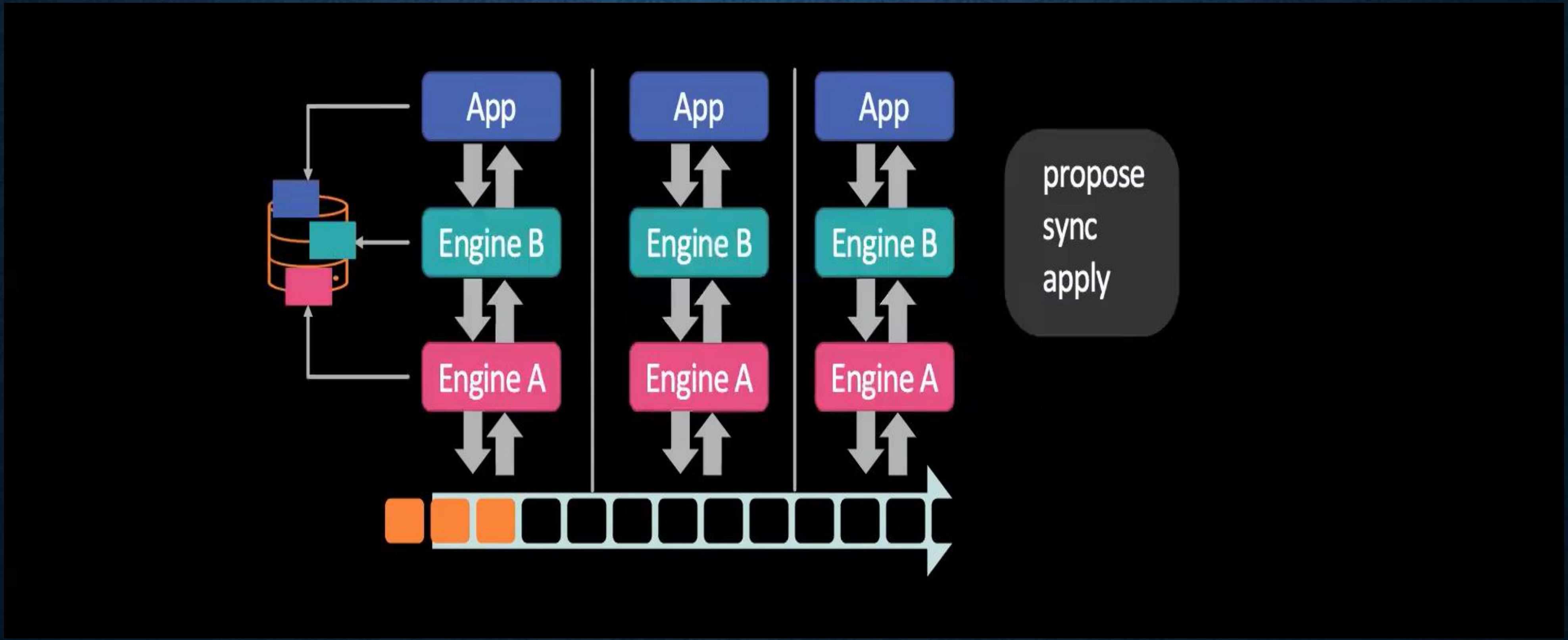
Butler Lampson, *Hints for Computer System Design*.

Butler W. Lampson. 1983. Hints for computer system design. SIGOPS Oper. Syst. Rev. 17, 5 (October 1983), 33-48. https://doi.org/10.1145/773379.806614

LOG STRUCTURED PROTOCOLS: LAYERING THE STATE MACHINE



LOG STRUCTURED PROTOCOLS: LAYERING THE STATE MACHINE



Breaking the platform state machine into lots of fine-grained state machines, and layering these in a protocol state, much like network packet layering.

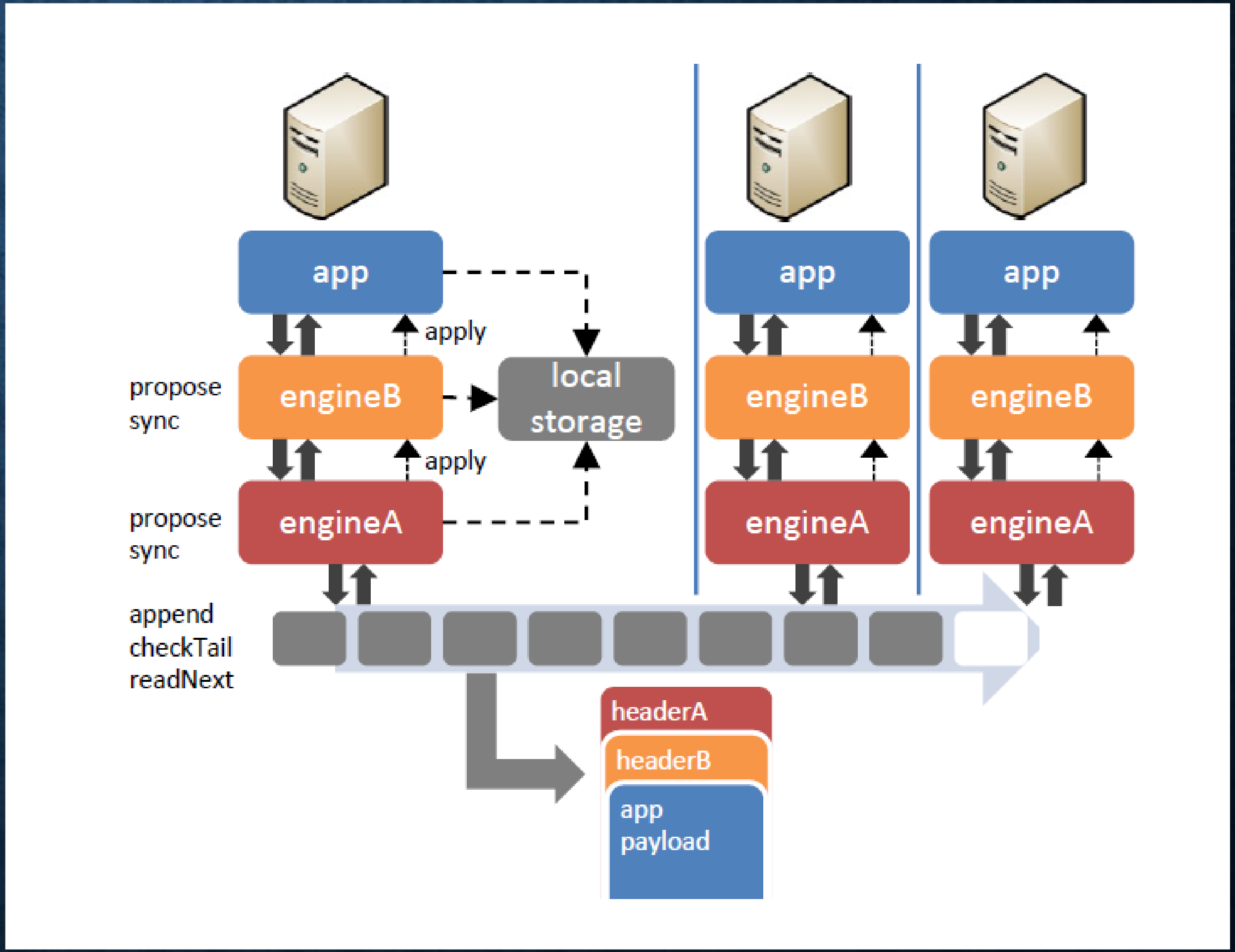
LOG-STRUCTURED PROTOCOLS DESIGN

LOG STRUCTURED PROTOCOL

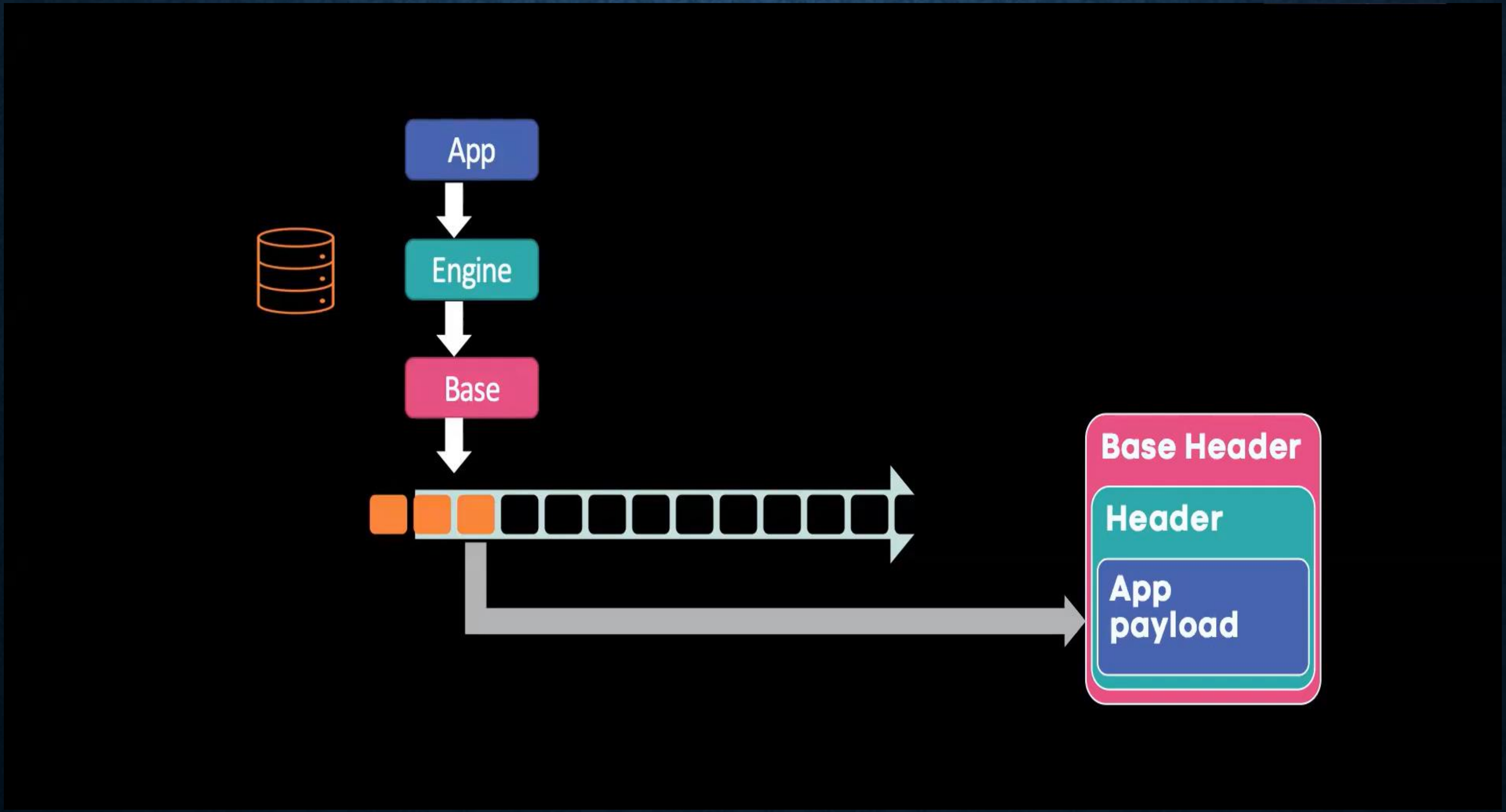
A fine-grained replicated state machine executing above a shared log that can be layered into reusable protocol stacks under different databases.

Components

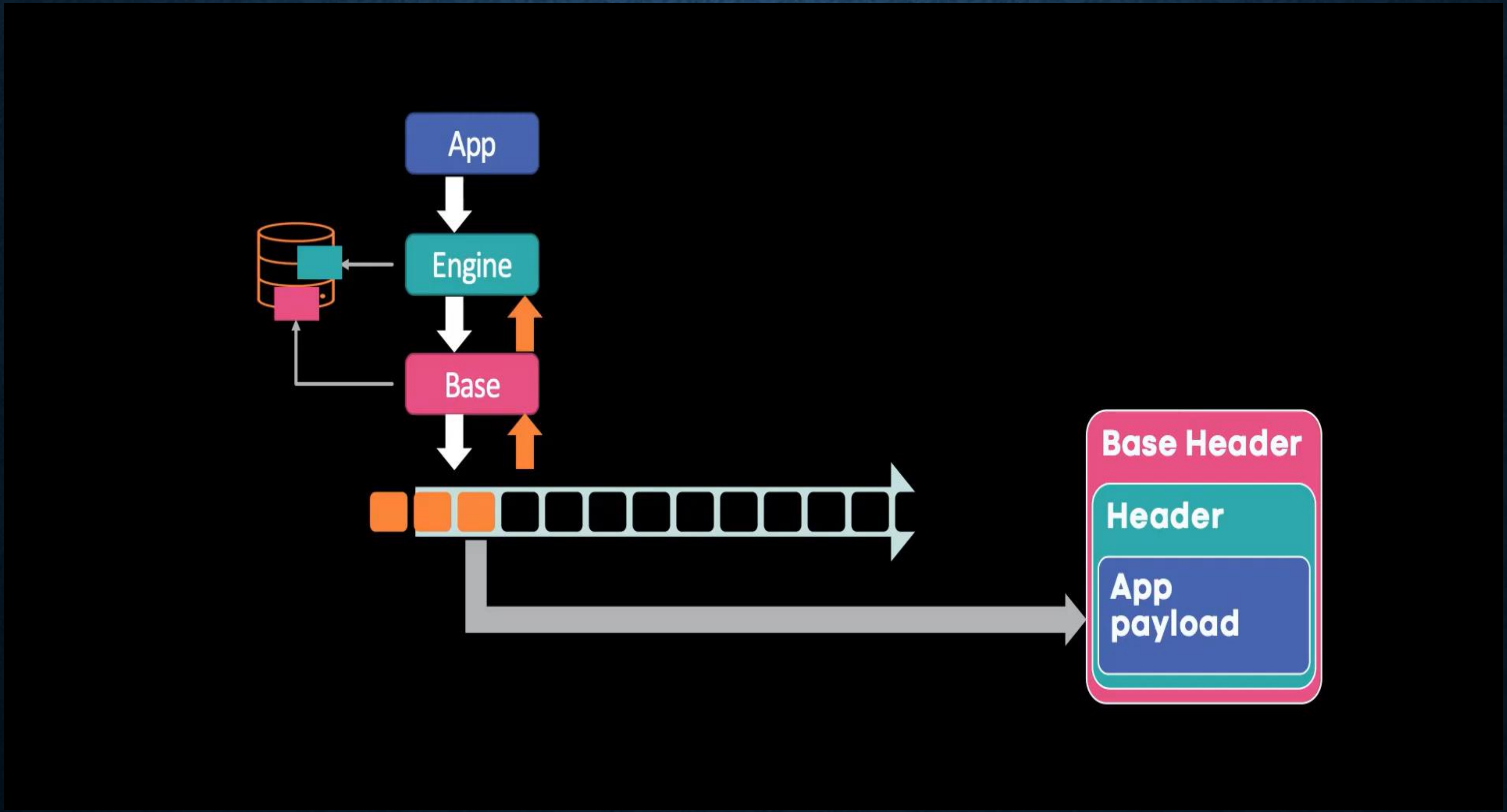
- Application Logic
- Engine
- Local Store
- Shared Log



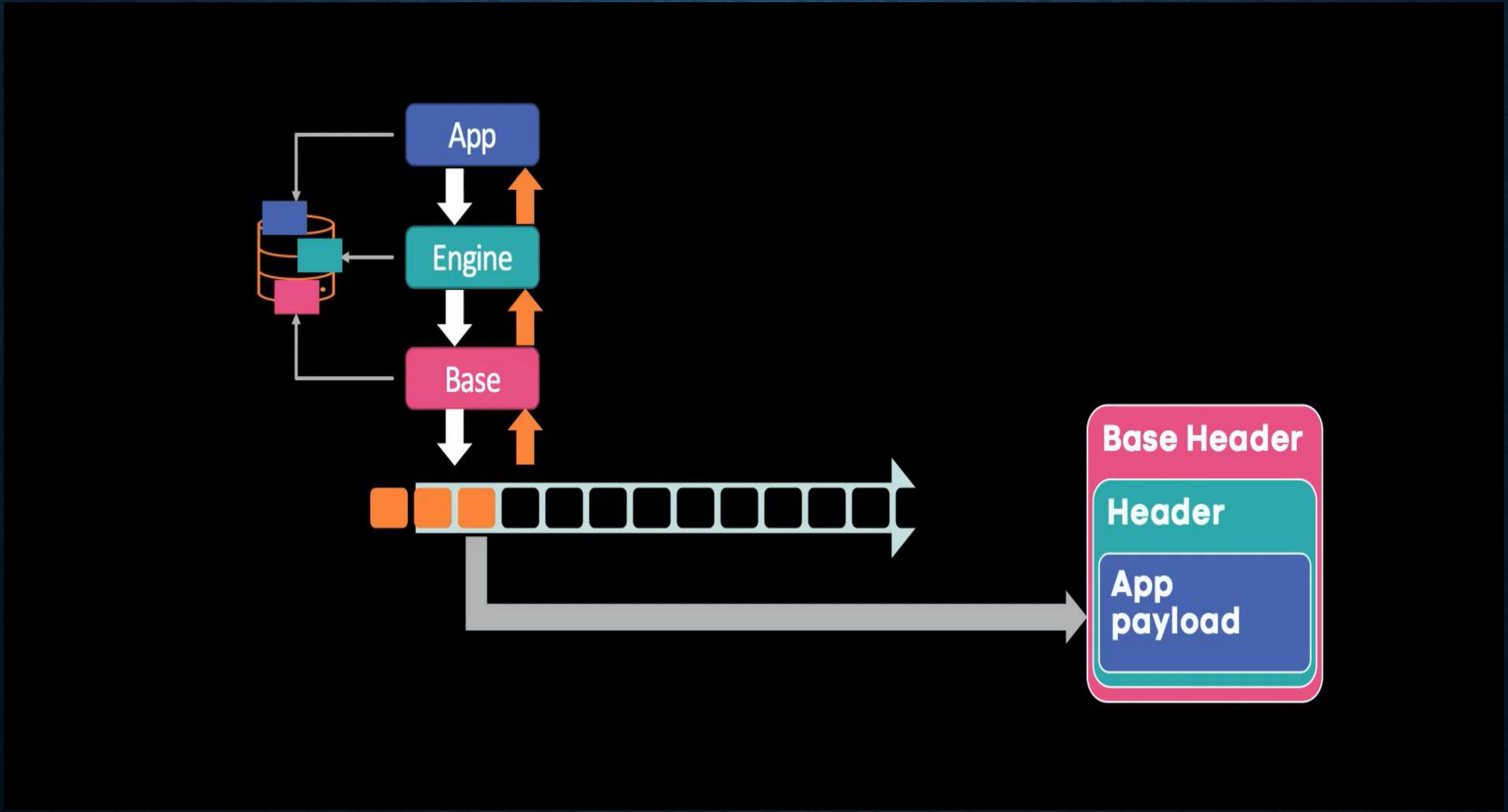
LOG-STRUCTURED PROTOCOLS: CRITICAL PATH



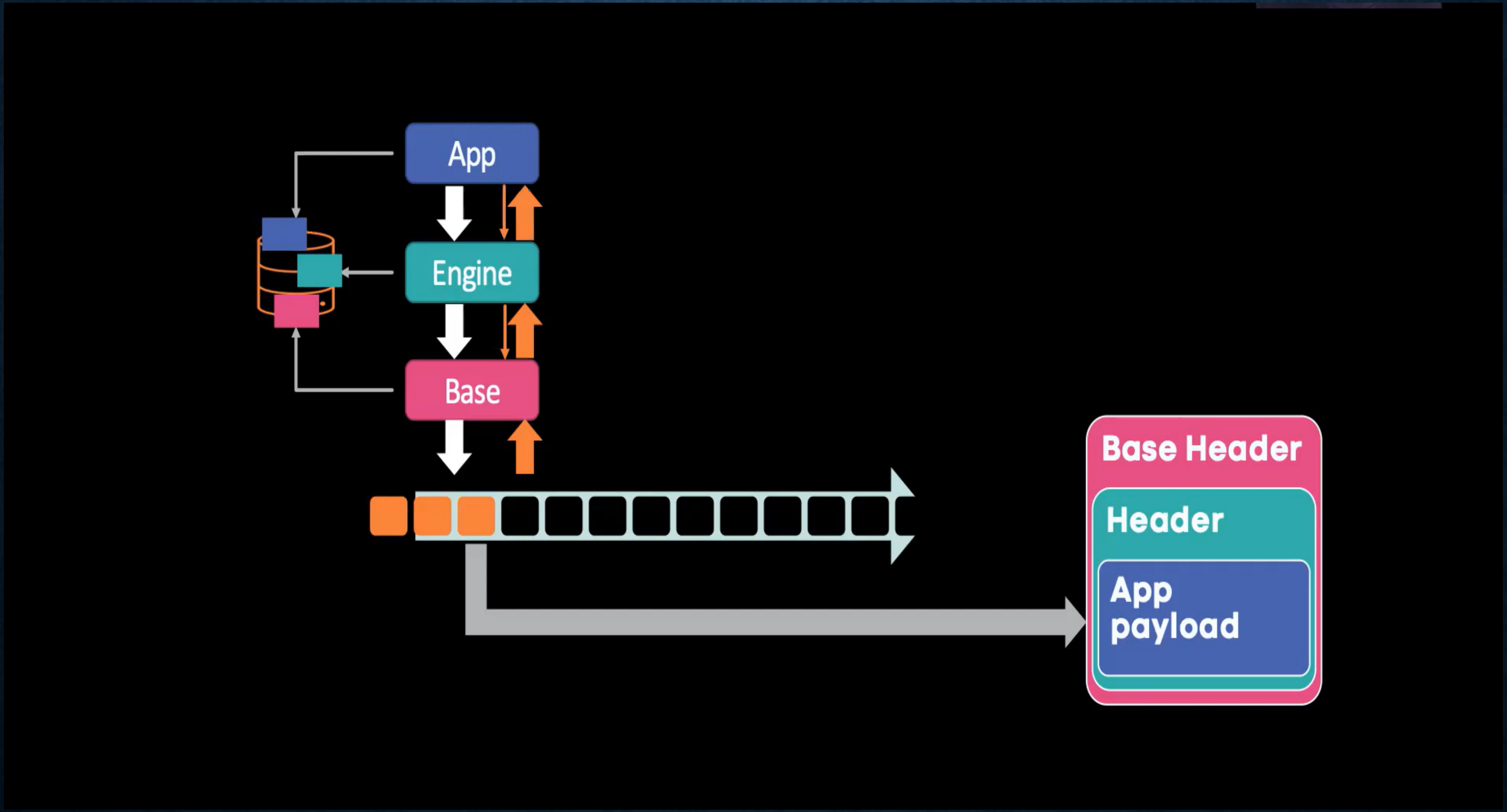
LOG-STRUCTURED PROTOCOLS: CRITICAL PATH



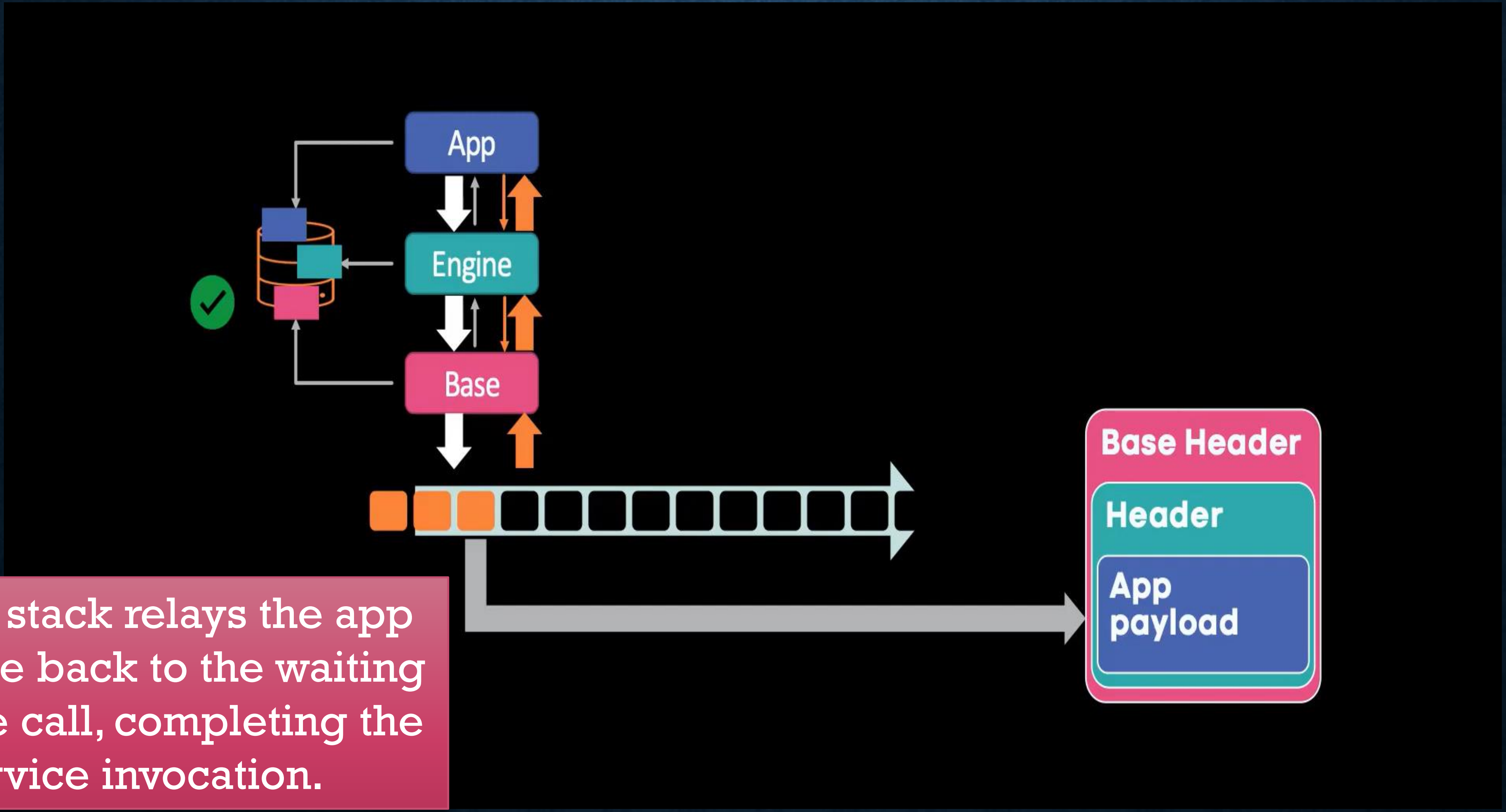
LOG-STRUCTURED PROTOCOLS: CRITICAL PATH



LOG-STRUCTURED PROTOCOLS: CRITICAL PATH



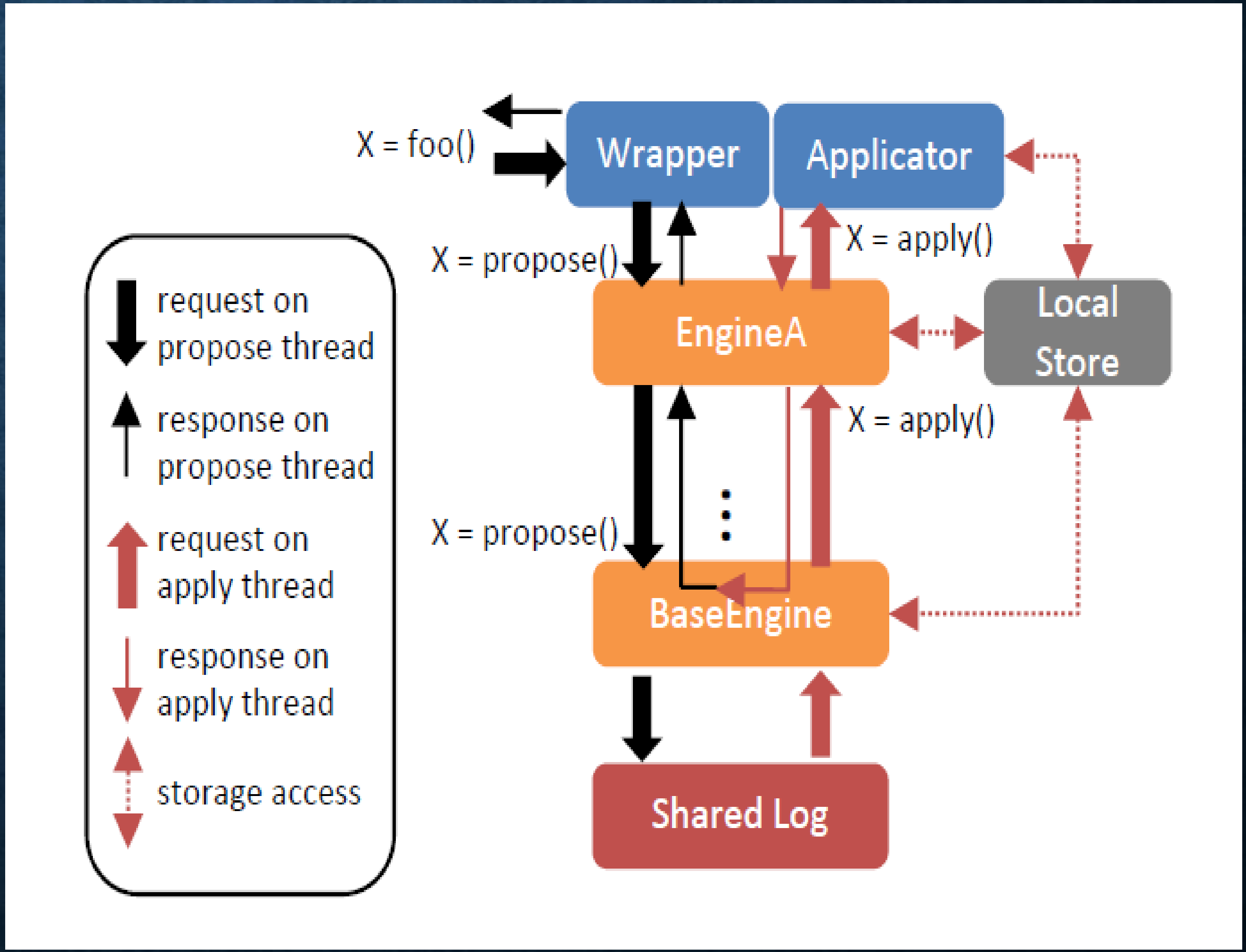
LOG-STRUCTURED PROTOCOLS: CRITICAL PATH



Engine stack relays the app response back to the waiting propose call, completing the service invocation.

THE LIFECYCLE OF A PROPOSAL

- Application split it into two parts: a Wrapper (exposing foo) and an Applicator
- Wrapper serializes an incoming request (without executing it) and calls propose on the top-most engine.
- Applicator receives this request from the top engine via the apply upcall, executes foo, and returns the response to the top engine.



LOG-STRUCTURED PROTOCOL ENGINE APIS

```
template <class ReturnType, class EntryType>
class IEngine {
  Future<ReturnType> propose(EntryType e);
  Future<ROTx> sync();
  void registerUpcall(IApplicator<ReturnType,
      EntryType> app);
  void setTrimPrefix(logpos_t pos);
}

template <class ReturnType, class EntryType>
class IApplicator {
  ReturnType apply(RWTx txn, EntryType e,
      logpos_t pos);
  void postApply(EntryType e, logpos_t pos);
}
```

A TALE OF TWO DATABASES AND NINE ENGINES

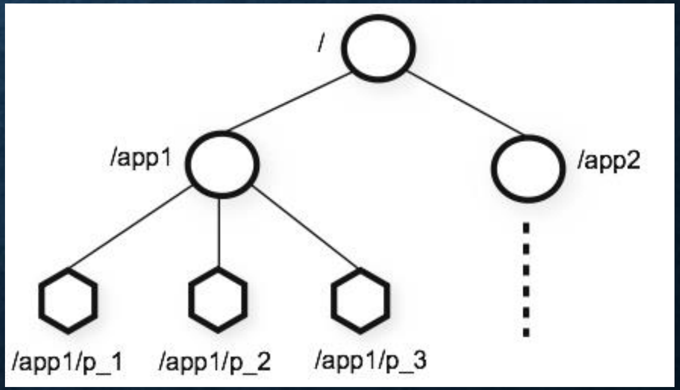
TWO DATABASES

DelosTable

- Rich API
- Support for transactions, secondary indexes, and range queries
- Strong guarantees on consistency, durability, and availability

Zelos

- Zookeeper-like interface
- Supports CRUD operations on a hierarchical structure of nodes



NINE LOG-STRUCTURED PROTOCOL ENGINES

Year	Engine	Prod	State/Prot	Use Case	LOC
2018	Base	Both	Yes/No	*State Machine Replication over the log.*	1081
2018	ViewTracking	Both	Yes/No	*Track durable copies of DB for trimming the log.*	844
2018	Observer	Both	No/Yes	*Monitor underlying stack.*	208
2019	BrainDoctor	Both	Yes/No	*Edit LocalStore directly, bypassing DB.*	274
2019	LogBackup	Both	Yes/No	*Coordinate learners to back up the log.*	688
2020	SessionOrder	Zelos	Yes/Yes	*Enforce session-ordering guarantee.*	521
2020	Batching	Zelos	No/Yes	*Improve throughput via batching + group commit.*	512
2021	Time	None	Yes/No	*Implement distributed time-outs.*	904
2021	Lease	None	Yes/Yes	*Enable 0-RTT strongly consistent reads.*	371

BASE ENGINE (1/2)

- BaseEngine resides at the bottom of the stack and implements the IEngine API over a shared log.
- Primary role: To play the log forward and apply each entry to the application above it.
- On a Propose,
  - Append the entry to the shared log
  - Play the log forward until the newly appended entry
  - Pass each entry up to the apply upcall of the application
  - Relay the application response to the waiting propose call

BASE ENGINE (2/2)

- A form of replicated RPC
- Durable: Returns only once the entry is stored durably on the shared log
- Failure-atomic: Executed on each server within a LocalStore transaction;
- Linearizable: Ordered via the shared log before executing on the local server
- Also responsible for the mechanism of GC: Periodically trims the shared log

VIEW TRACKING ENGINE

- ViewTrackingEngine coordinates the trimming of the log.
- It tracks the playback position of each server
- When all servers have played the log past some point X, the log can be trimmed until X.

OBSERVER ENGINE

- The ObserverEngine is placed between different layers of Delos stacks.
- A lightweight layer that measures and externally logs end-to-end latencies on each propose/sync call
- Provides reusable monitoring functionality by tracking the time spent in a given engine

BRAIN DOCTOR ENGINE

- BrainDoctorEngine acts as a simple pass-through engine, with one addition: an external call that accepts a list of raw LocalStore writes and proposes it into the log.
- Used in emergencies to perform “brain surgery” on the key-value store (to fix a bug in the state of database)
- Directly changing the state of a running Delos database without going through application logic

LOG BACKUP ENGINE

- Customer request for Point-in-Time restore
- Need to copy the shared log to a backup store before trimming it
- Reconstruct any intermediate state of the database by starting from a prior snapshot backup and playing the log backup forward

SESSION ORDER ENGINE

- The SessionOrderEngine implements the idea of Zookeeper sessions
  - ZooKeeper provides a session-ordering guarantee: within a session, if a client first issues a write and then a concurrent read (without waiting for the write to complete), the read must reflect the write.
  - This property is stronger than linearizability, which allows concurrent writes and reads to be ordered arbitrarily; and encompasses exactly-once semantics
- Delos implements these semantics in the SessionOrderEngine by assigning sequence numbers (essentially autoincrementing IDs) to outgoing writes.
- When other nodes read from the log, they check that the writes are ordered based on the sequence number, reordering them into the correct sequence as necessary.

TIME ENGINE

- TimeEngine implemented to support time-based trimming in a way that is robust to clock skew and drift
- Allows the creation of a timer object which fires once a fixed amount of time has elapsed on a constant number of servers within the cluster

BATCHING ENGINE

- The BatchingEngine groups entries into a single transaction write to the LocalStore.
- Group commit optimization approach enables higher performance and provides a common implementation that both DelosTable and Zelos use (related to Delos' design goal of code re-use).

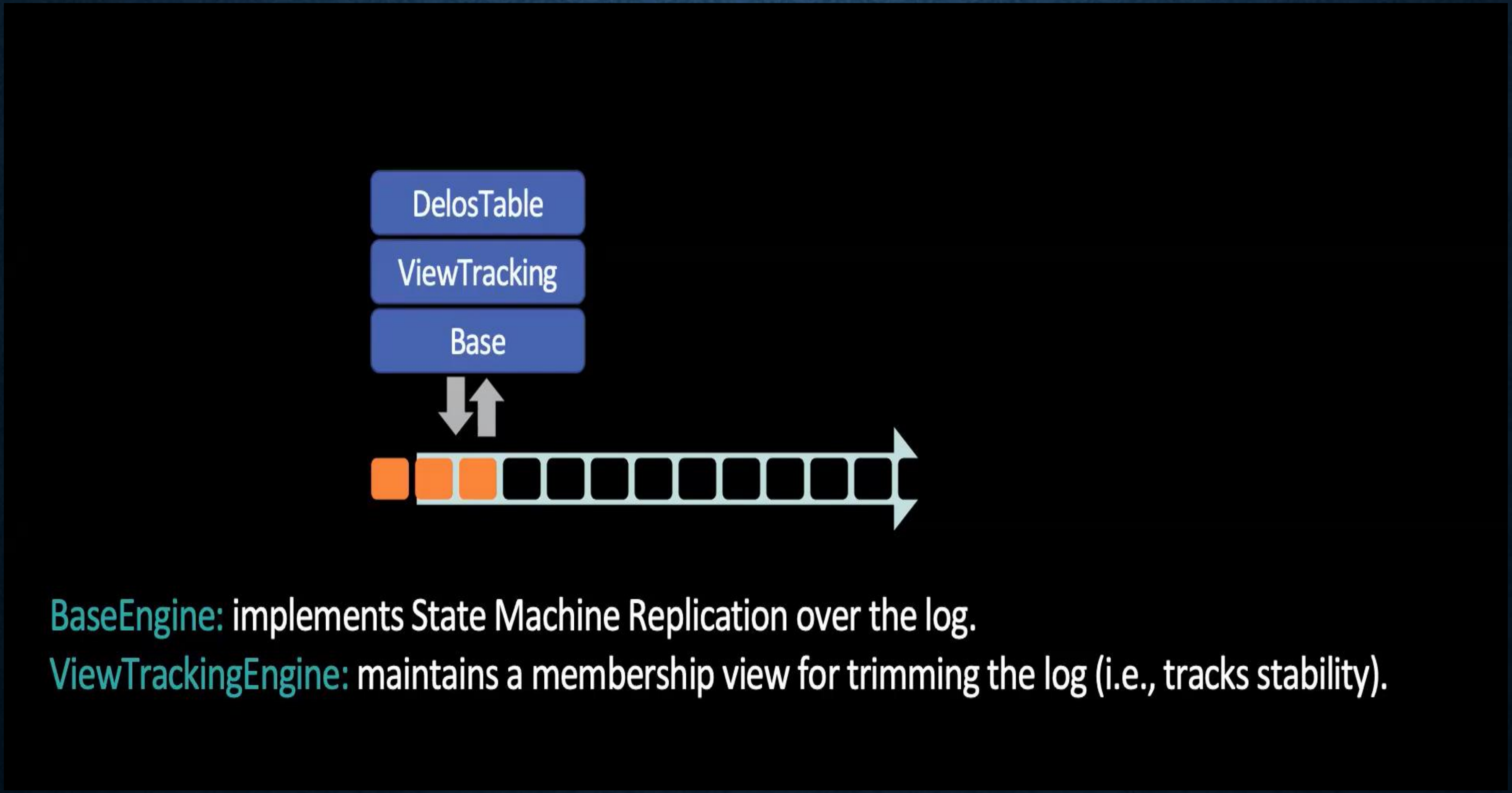
LEASE ENGINE

- The BaseEngine has a leaderless design above the shared log
  - Any server can propose a command, while each server can sync with the shared log to ensure strong consistency.
  - Advantage: The loss of a single server does not disrupt availability.
  - Disadvantage: The sync before a strongly consistent read incurs a round-trip to the shared log.
- Designs with a strong leader can provide 0-RTT strongly consistent reads at the leader.
- LeaseEngine elects a server as a designated proposer above the shared log.
- Reads at this server can be satisfied with strong consistency without accessing the shared log.

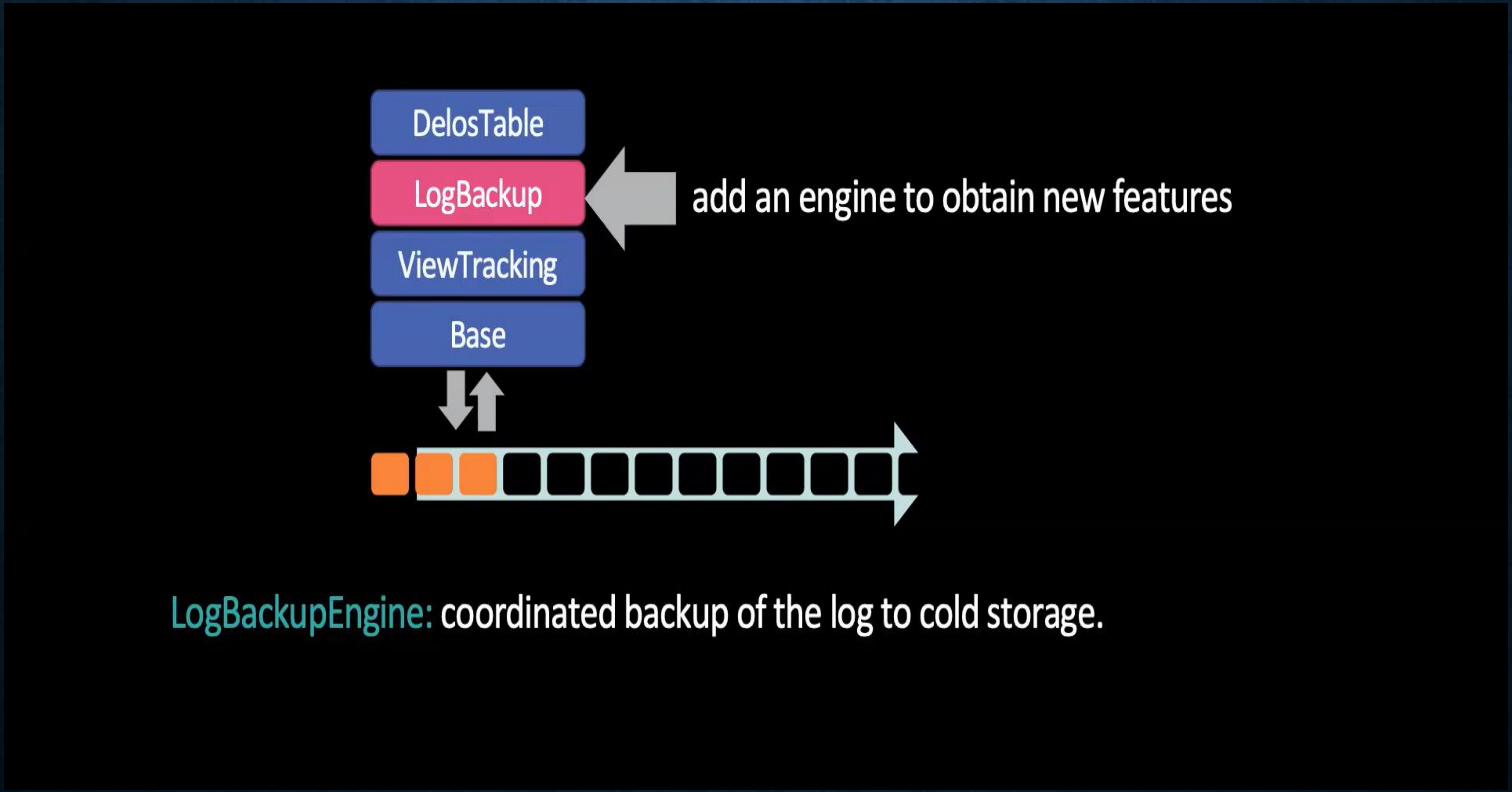
PRODUCTION DELOS STACKS

BENEFITS

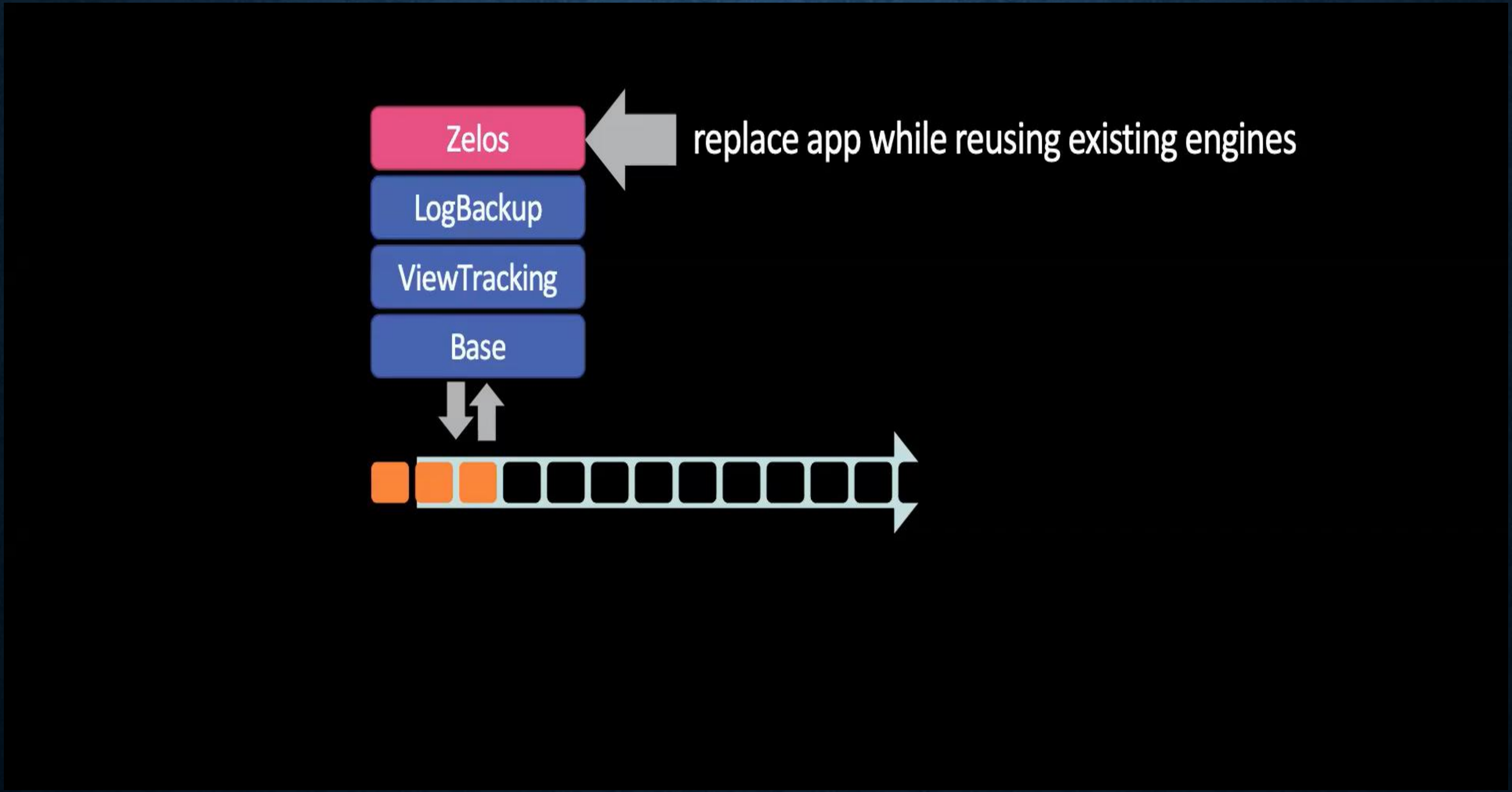
BENEFIT#0: RAPID DEPLOYMENT



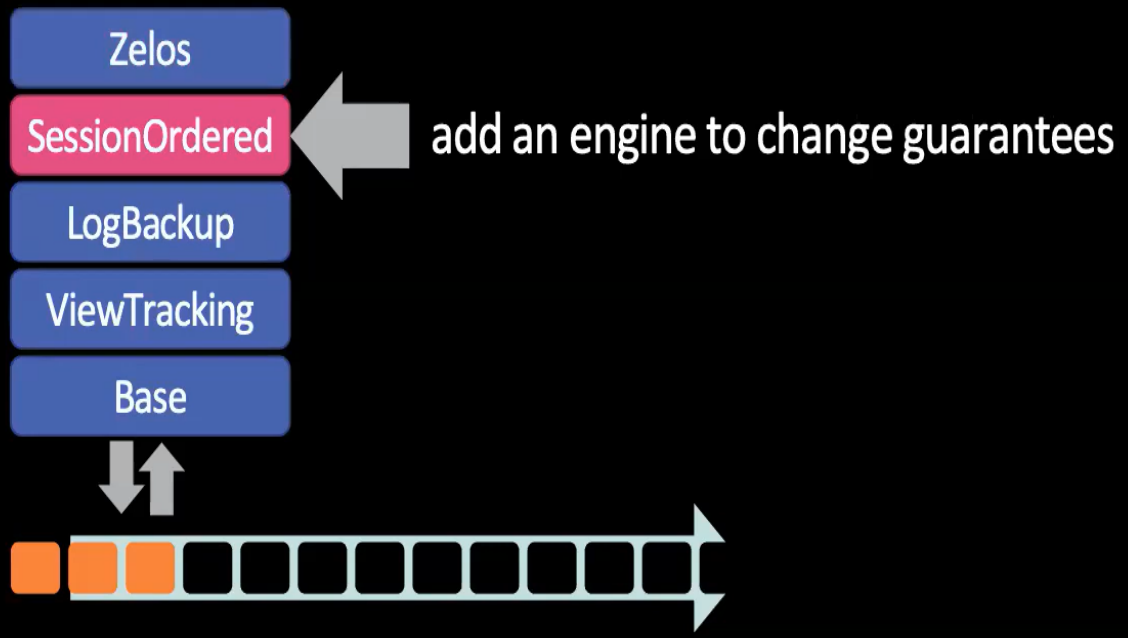
BENEFIT#1: INCREMENTAL UPGRADES



BENEFIT#2: CODE REUSE

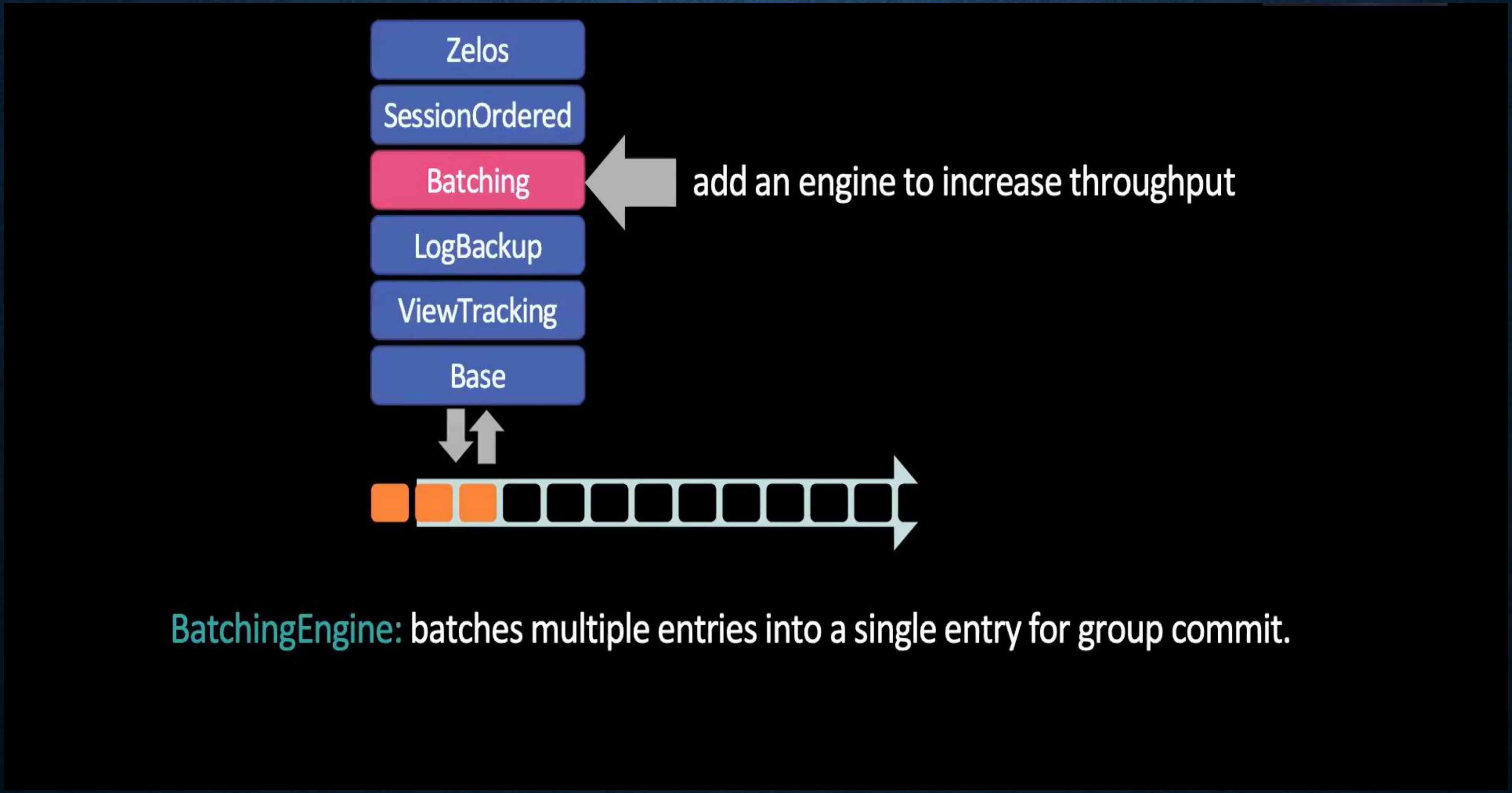


BENEFIT #3: CUSTOMIZING BEHAVIOR

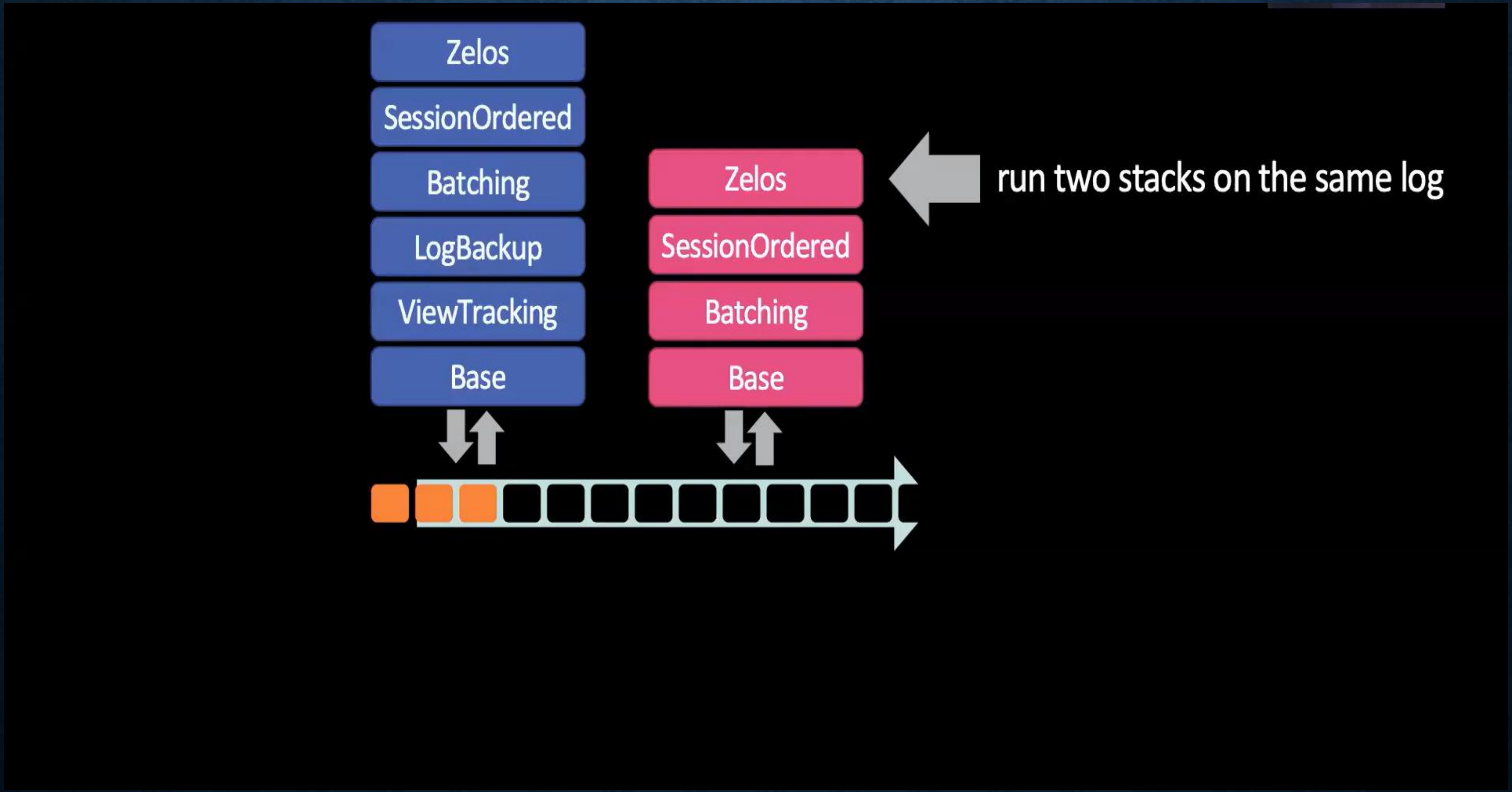


SessionOrderedEngine: filters and re-orders to enforce Session-Ordering.

BENEFIT #4: IMPROVING PERFORMANCE



BENEFIT #5: DIFFERENT ROLES



EVALUATION

EVALUATION

The Overhead of Layering

Log-structured protocols are lightweight.

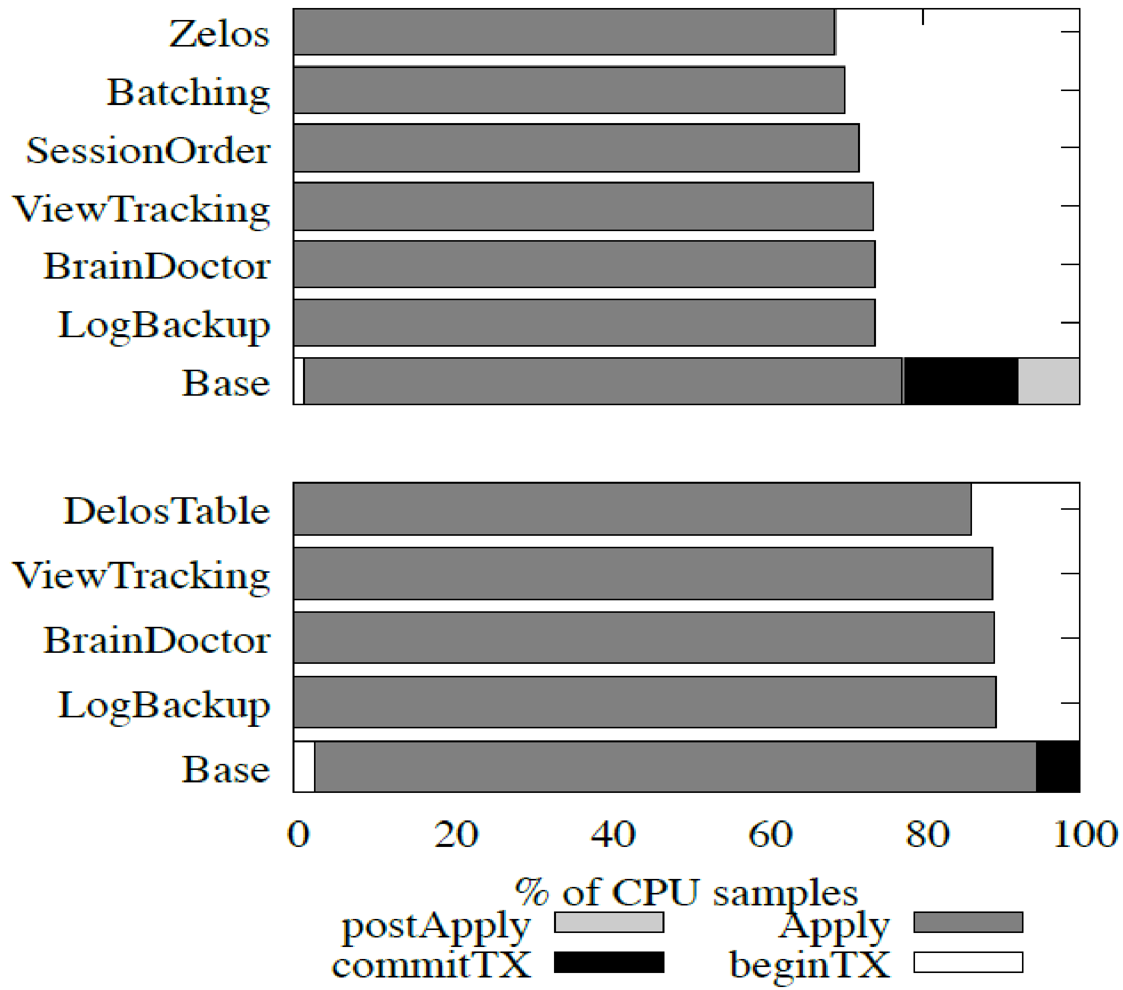


Figure 7: *Fleet-wide sampling of the apply thread in production clusters shows layering adds low overhead.*

EVALUATION

The Overhead of Layering

The apply thread is not the bottleneck.

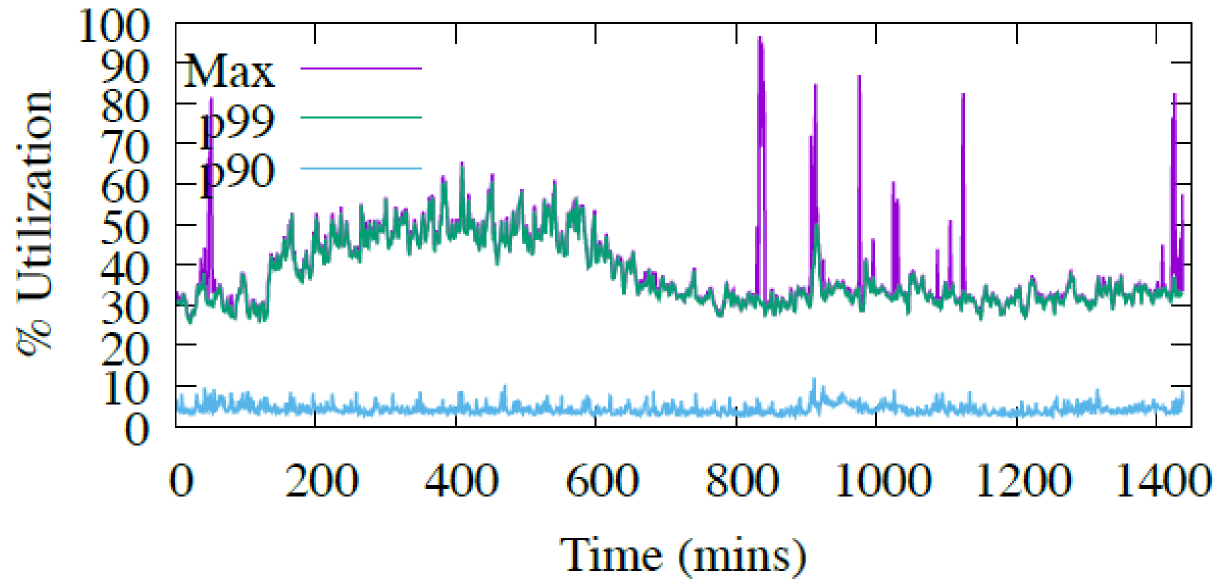


Figure 8: *Apply thread utilization across the fleet for a single day, measured over 1-minute periods: for each minute, we show the three clusters with the max / p99 / p90 utilizations. For any given minute, 90% of the clusters are below 10% apply utilization.*

The p99 latency is the highest latency value (slowest response) of the fastest 99 percent of requests i.e. worst latency observed by 99% of all requests if you ignore the top 1%.

EVALUATION

Benefits of Layering

Log-structured protocols can optimize performance significantly.

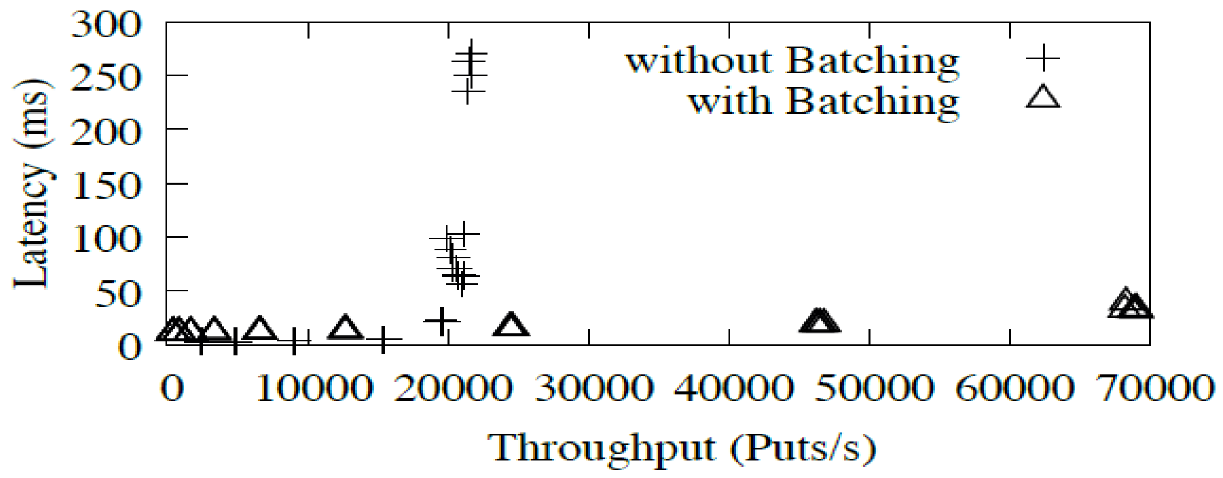


Figure 9: *The BatchingEngine provides a 2X increase in maximum throughput under 20ms p99 latency.*

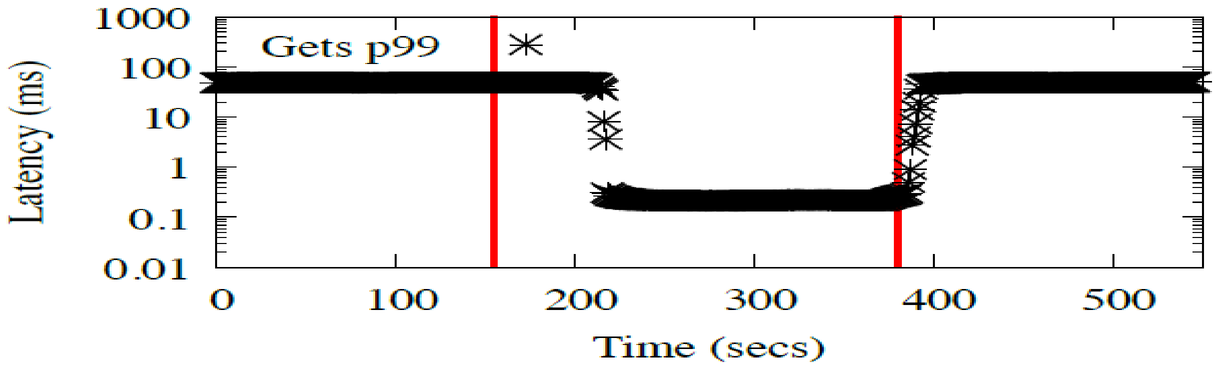


Figure 10: *When enabled, the LeaseEngine allows zero-coordination strongly consistent reads at the server holding a lease, lowering read latency by 100X for a deployment distributed across the continental USA.*

EVALUATION

Benefits of Layering

Log-structured protocols enhance observability.

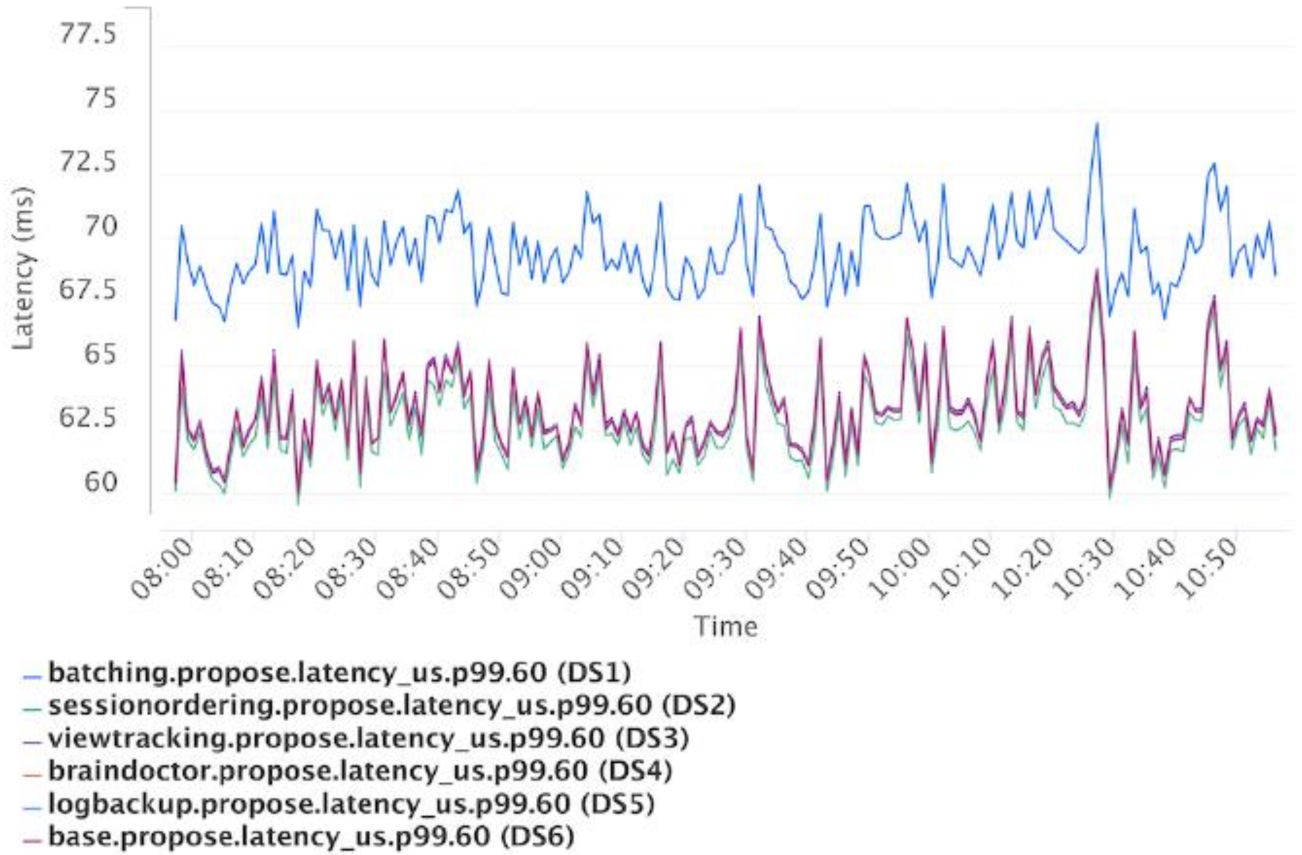


Figure 11: *Screenshot of ObserverEngine-enabled production dashboard for monitoring engines on a cluster.*

CONCLUSION

- Delos is a control plane database at the bottom of the Facebook Stack
- Log-Structured Protocols enabled multiple databases on a single platform:
  - DelosTable
  - Zelos
  - DelosQ
  - …

REFERENCES

- Mahesh Balakrishnan, Chen Shen, Ahmed Jafri, Suyog Mapara, David Geraghty, Jason Flinn, Vidhya Venkat, Ivailo Nedelchev, Santosh Ghosh, Mihir Dharamshi, Jingming Liu, Filip Gruszczynski, Jun Li, Rounak Tibrewal, Ali Zaveri, Rajeev Nagar, Ahmed Yossef, Francois Richard, and Yee Jiun Song. 2021. Log-structured Protocols in Delos. In Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21). Association for Computing Machinery, New York, NY, USA, 538–552. https://doi.org/10.1145/3477132.3483544
- https://www.facebook.com/atscaleevents/videos/194783778140362/

REFERENCES

- https://engineering.fb.com/2019/06/06/data-center-engineering/delos/#:~:text=Delos%20is%20designed%20around%20the,new%20update%20to%20the%20VirtualLog.
- https://www.youtube.com/watch?v=4EGArLVavbg
- https://www.youtube.com/watch?v=H-7OCFnTeMY
- http://muratbuffalo.blogspot.com/2021/10/log-structured-protocols-in-delos-sosp21.html
  https://www.facebook.com/atscaleevents/videos/389699398783537/?t=1790
- https://atscaleconference.com/2021/03/15/virtualizing-consensus/
- https://www.micahlerner.com/2021/11/23/log-structured-protocols-in-delos.html

THANK YOU