Lineage Stash: Fault Tolerance Off the Critical Path

Stephanie Wang, John Liagouris, Robert Nishihara, Philipp Moritz, Ujval Misra, Alexey Tumanov, Ion Stoica

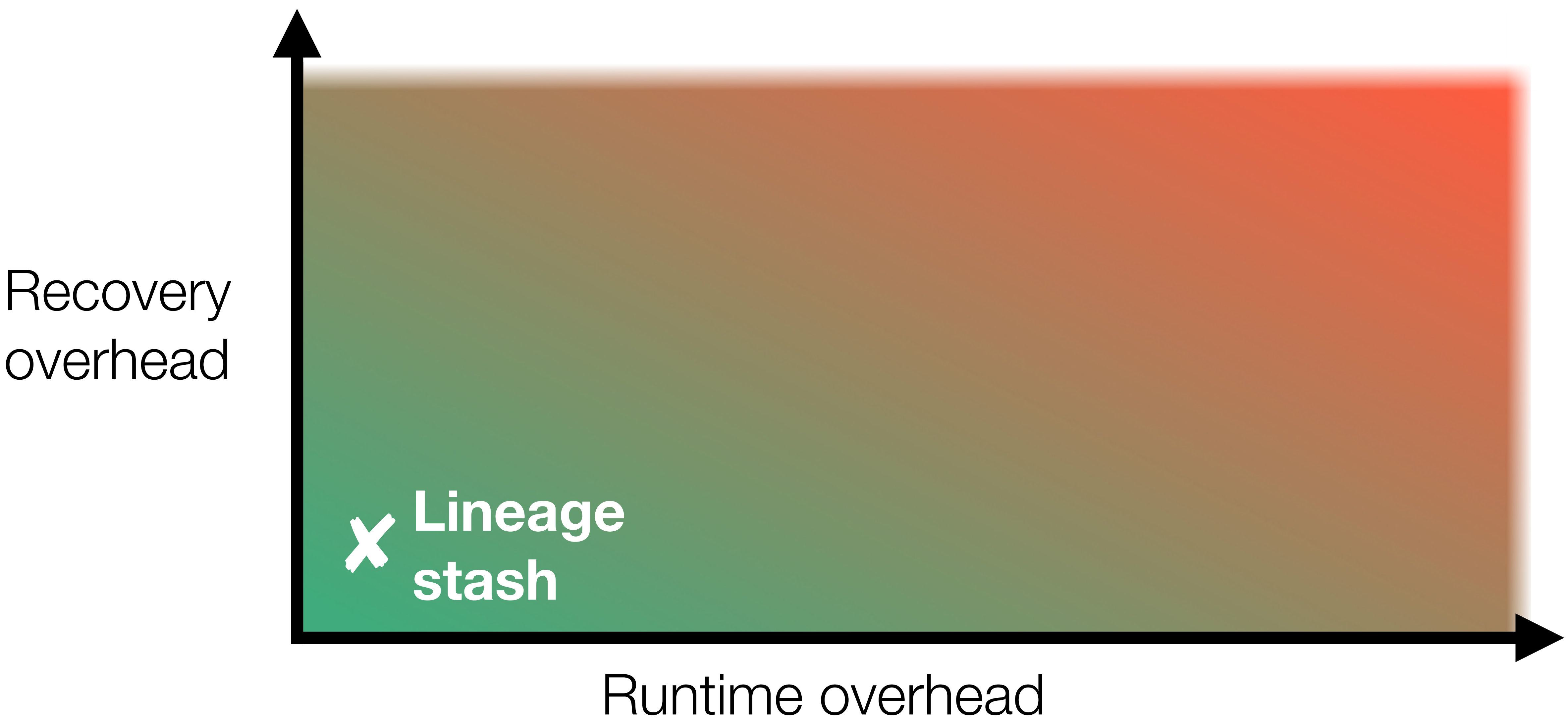
Low latency is increasingly important in data processing systems

Data processing is used today in **online** systems.

- Stream processing
- Graph processing
- Control systems

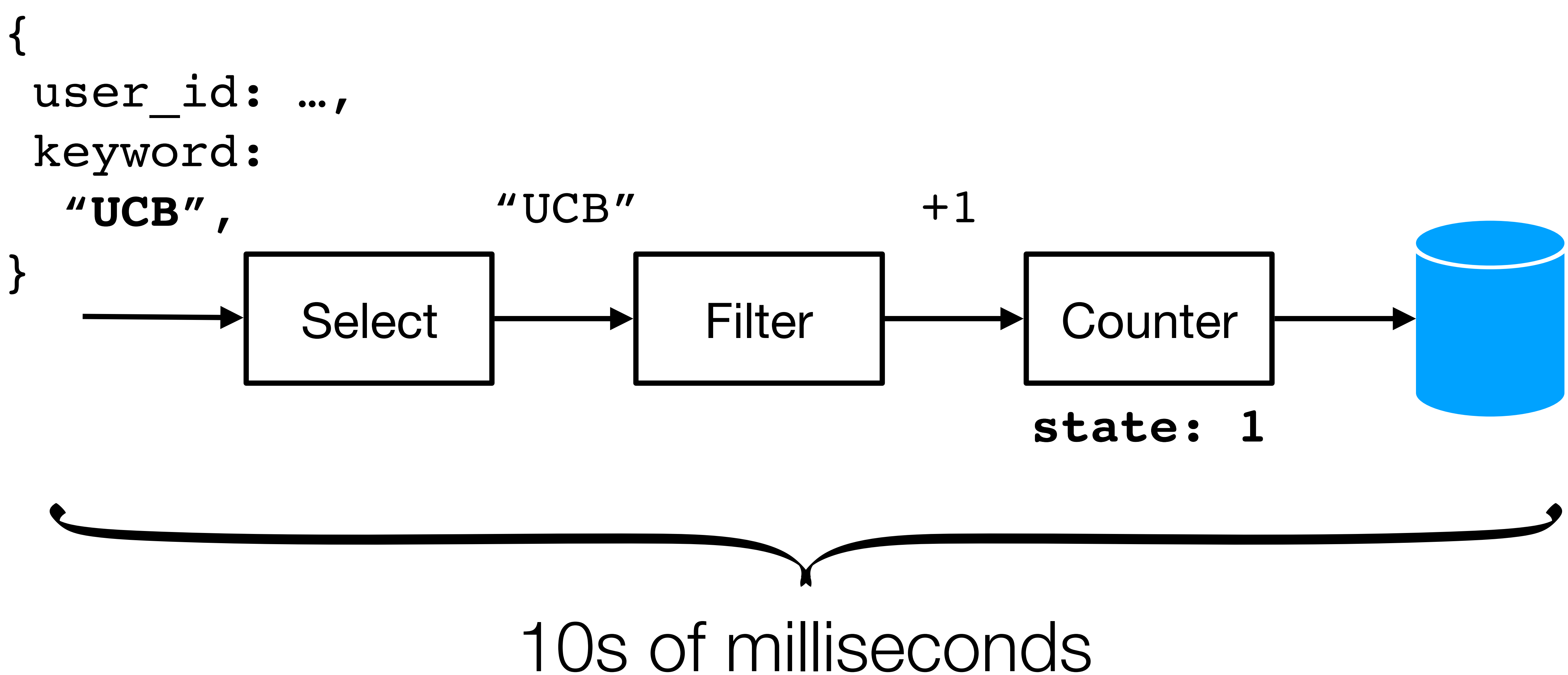
Data processing at **large scale** also requires the ability to **recover** results after a failure.

Tradeoff between low latency and recovery time

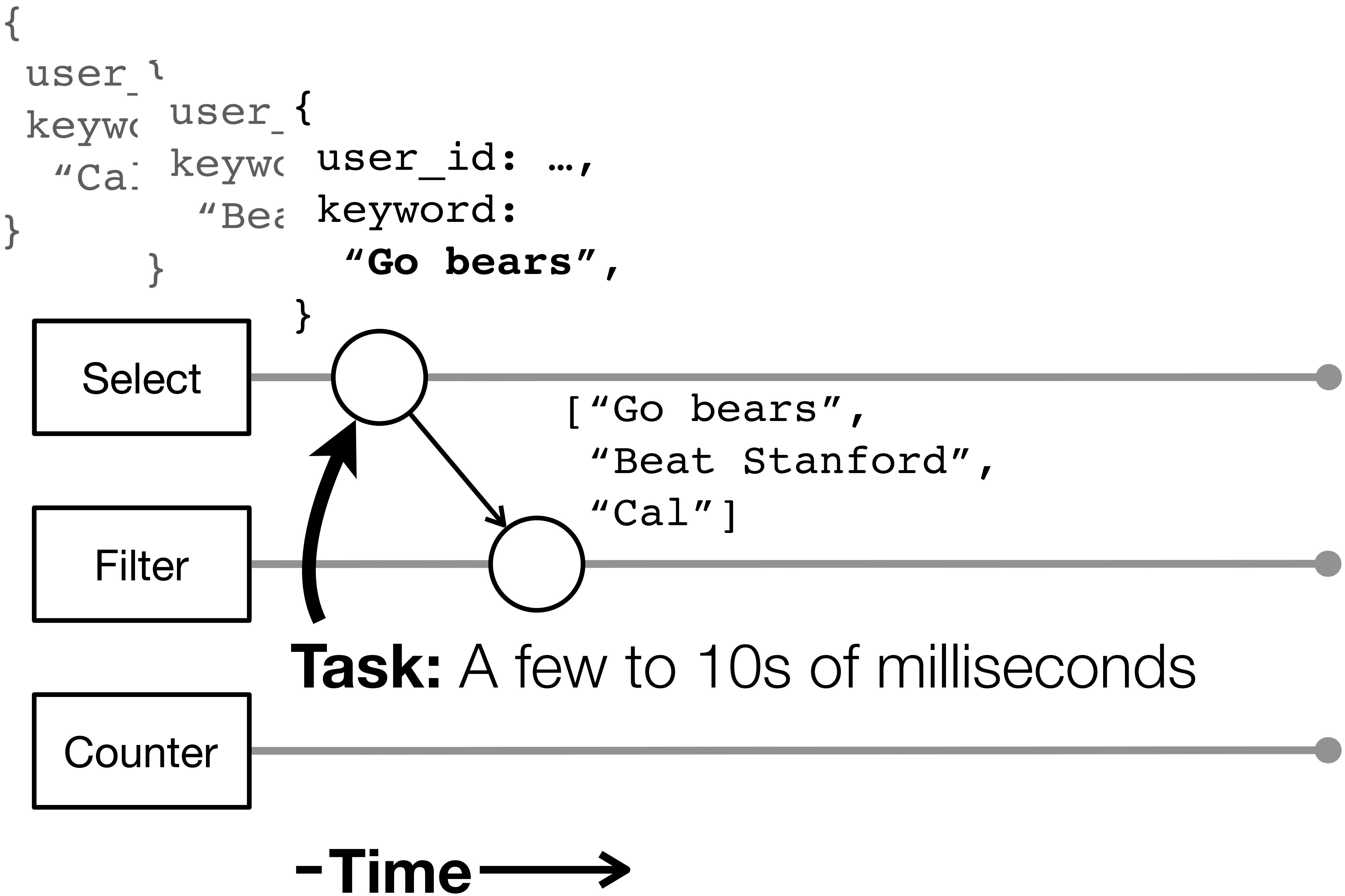


Stream processing example

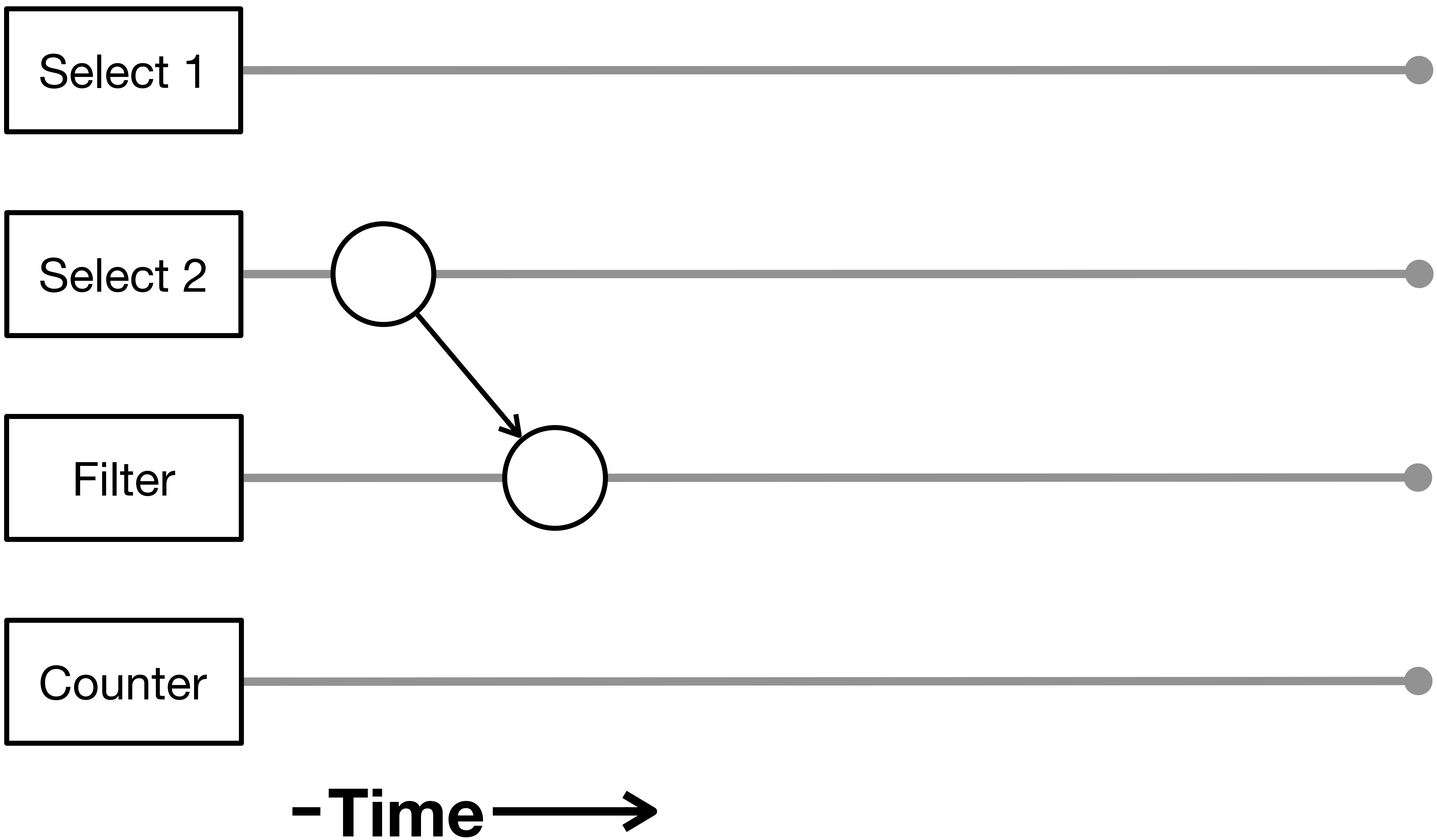
Display the number of times “UCB” has been queried.



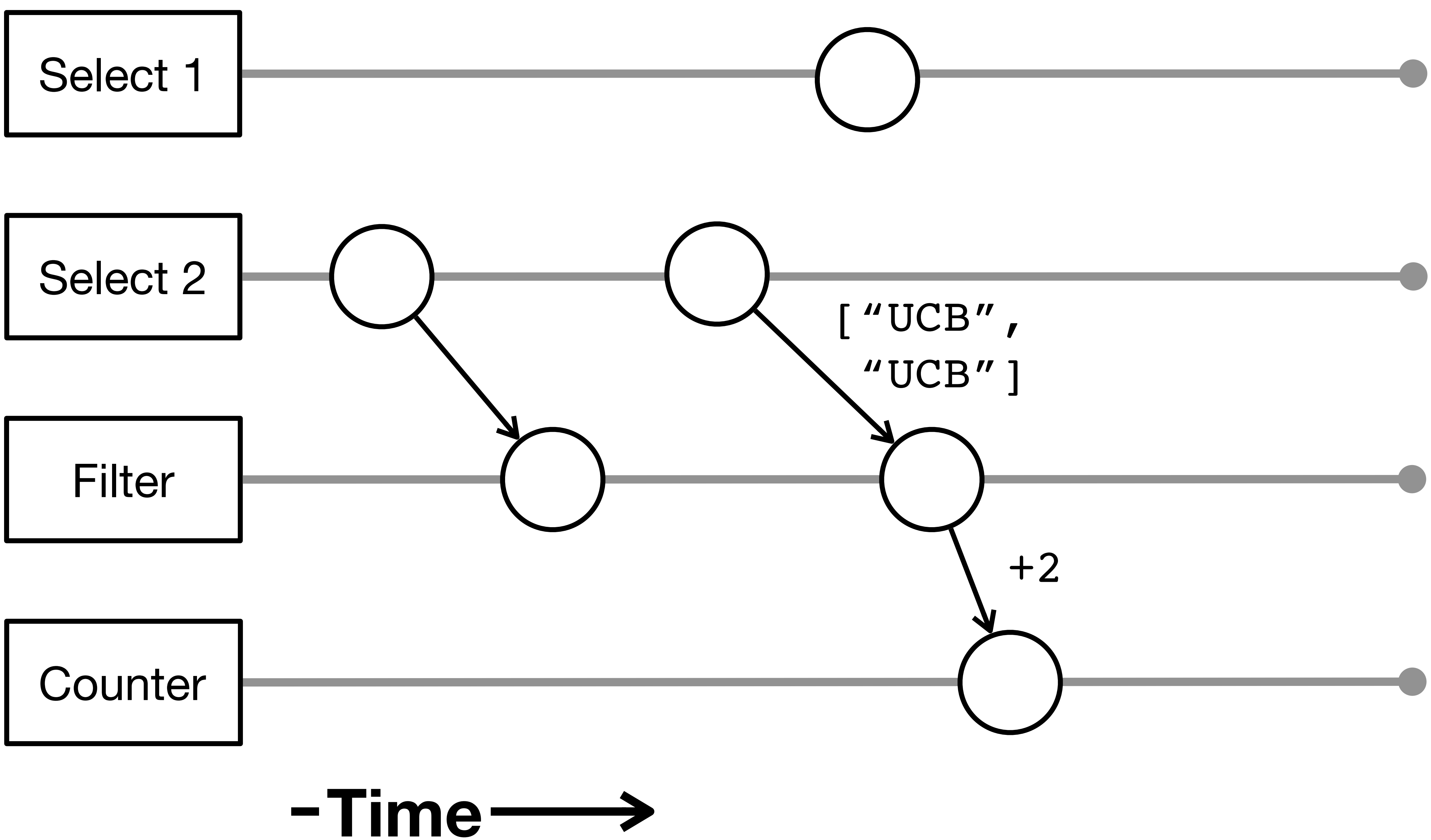
Stream processing example



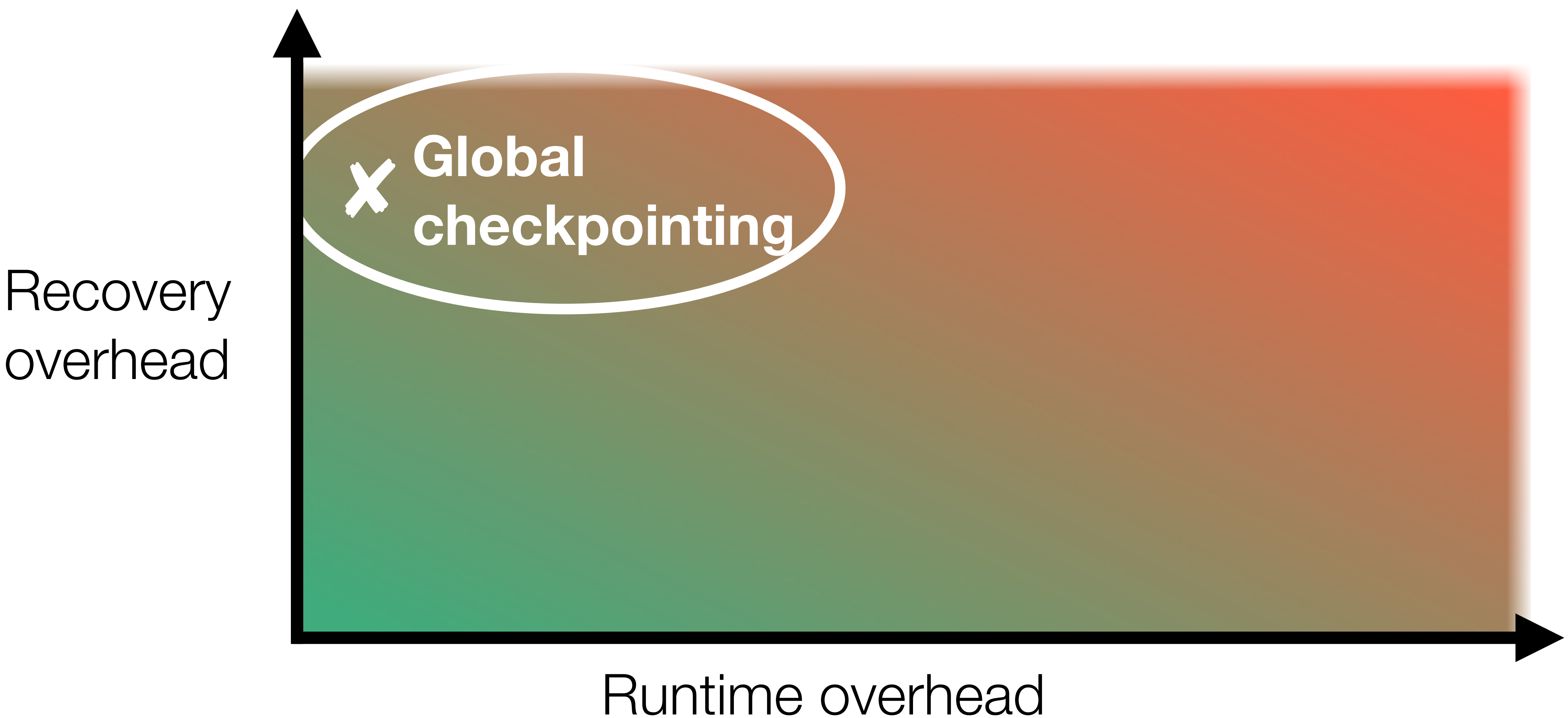
Stream processing example



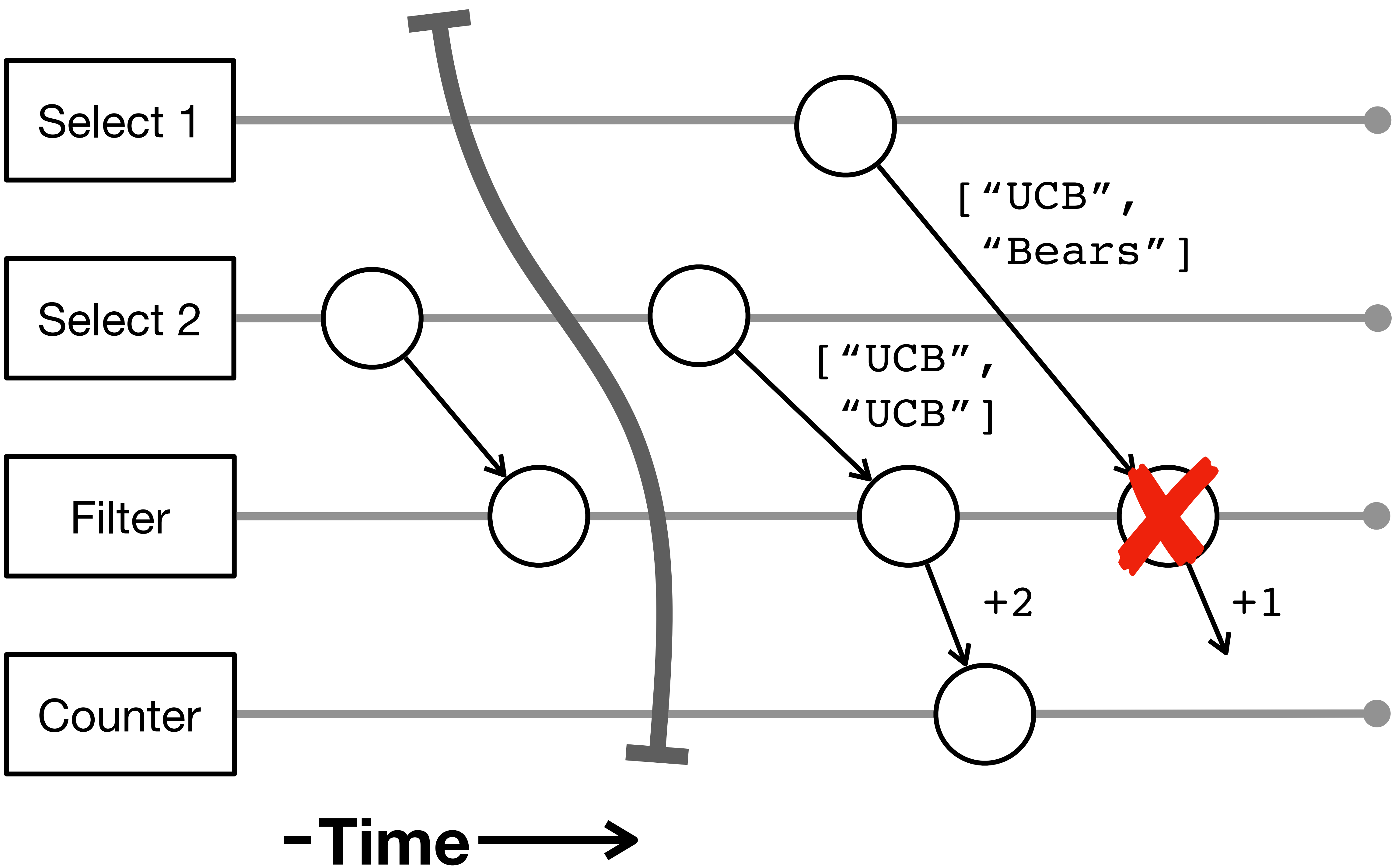
Stream processing example



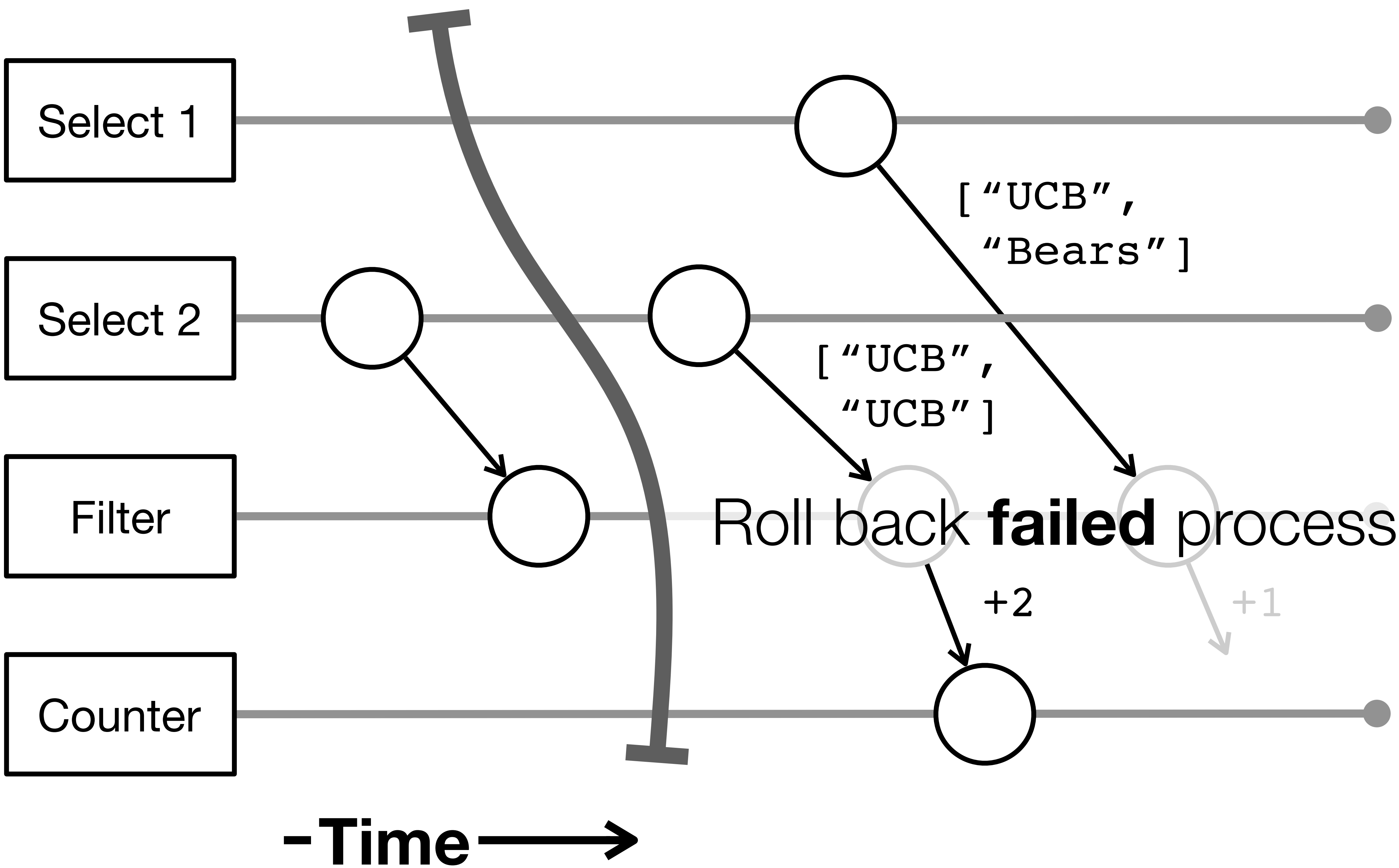
Tradeoff between low latency and recovery time



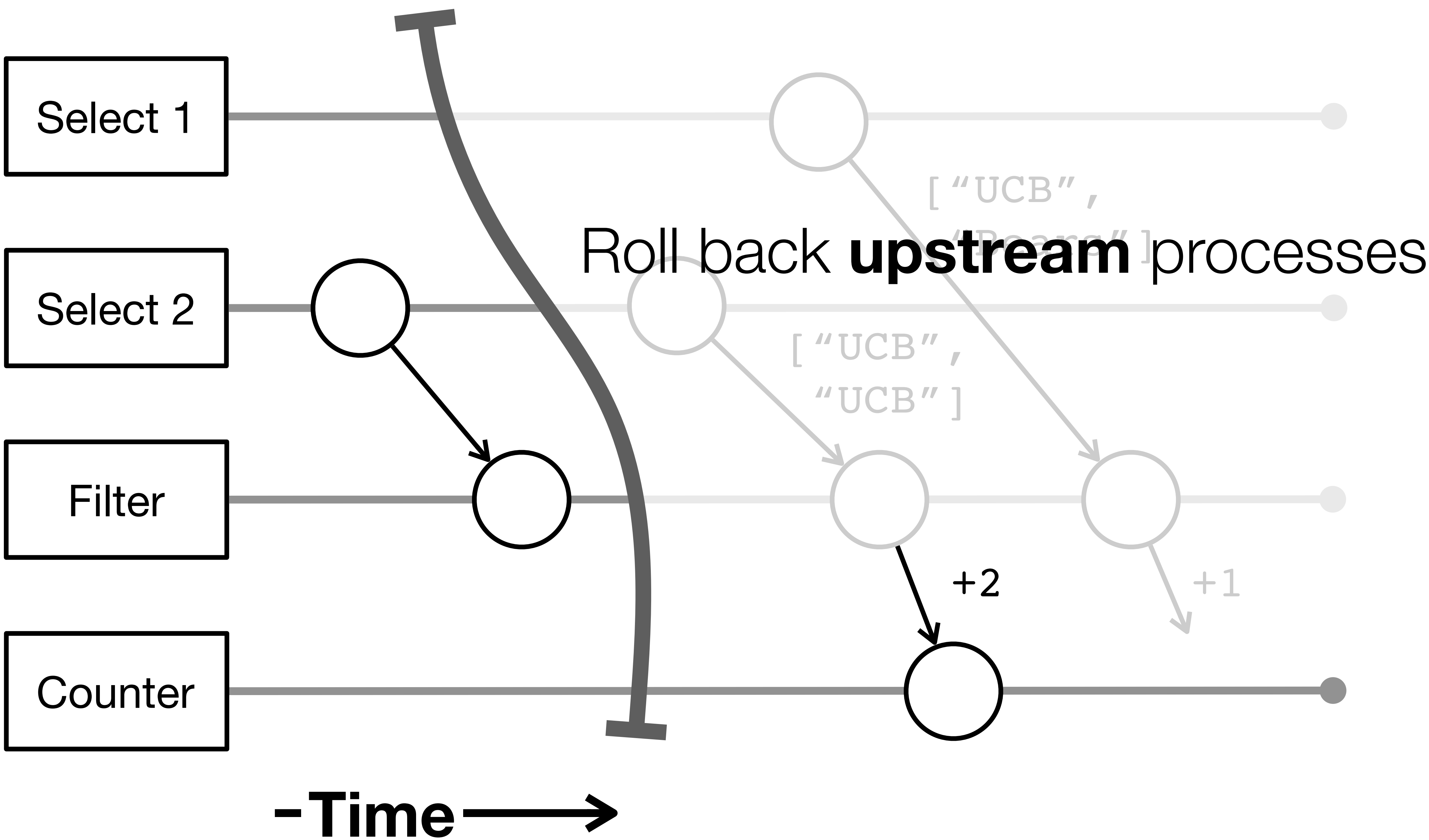
Global checkpointing



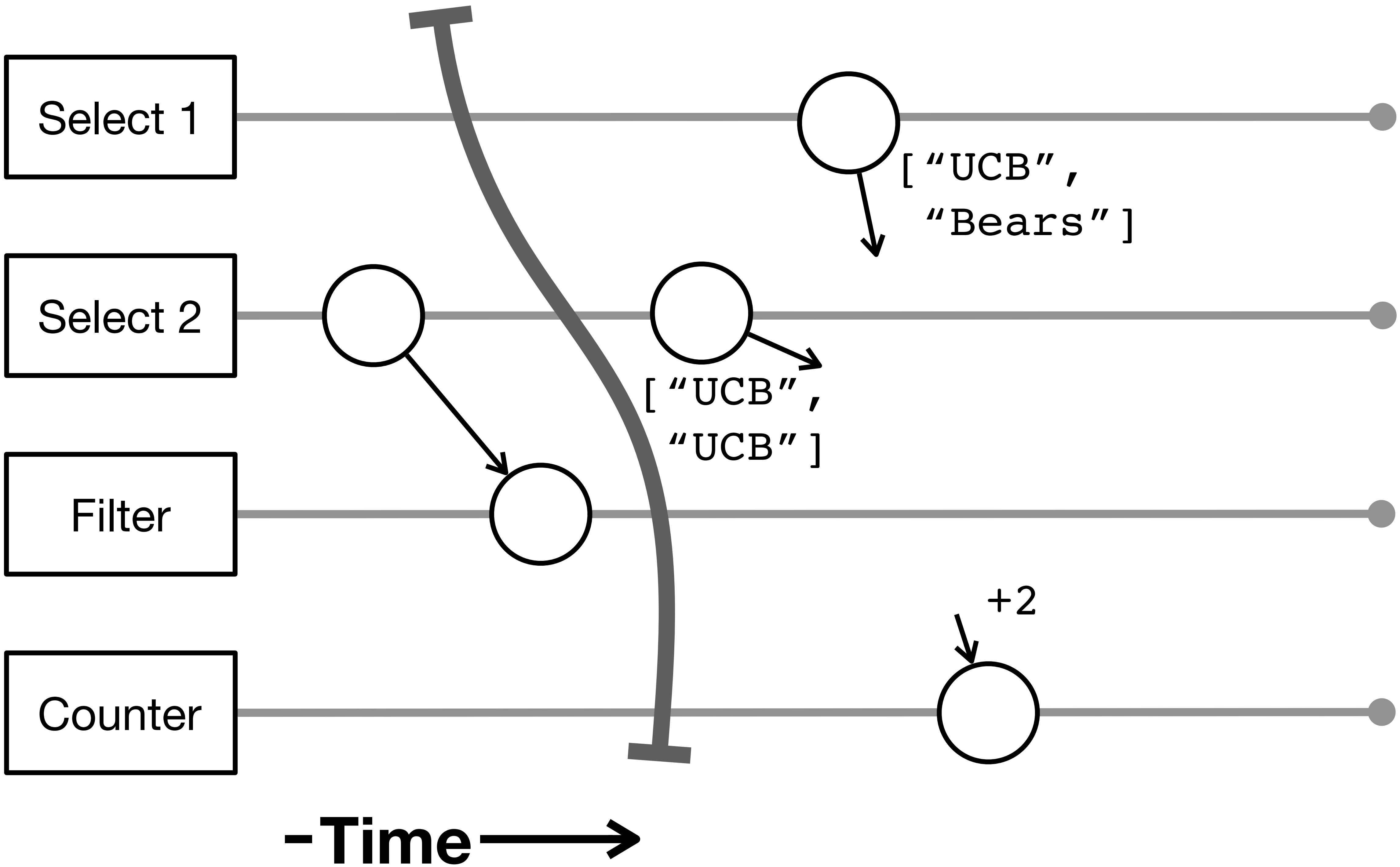
Global checkpointing



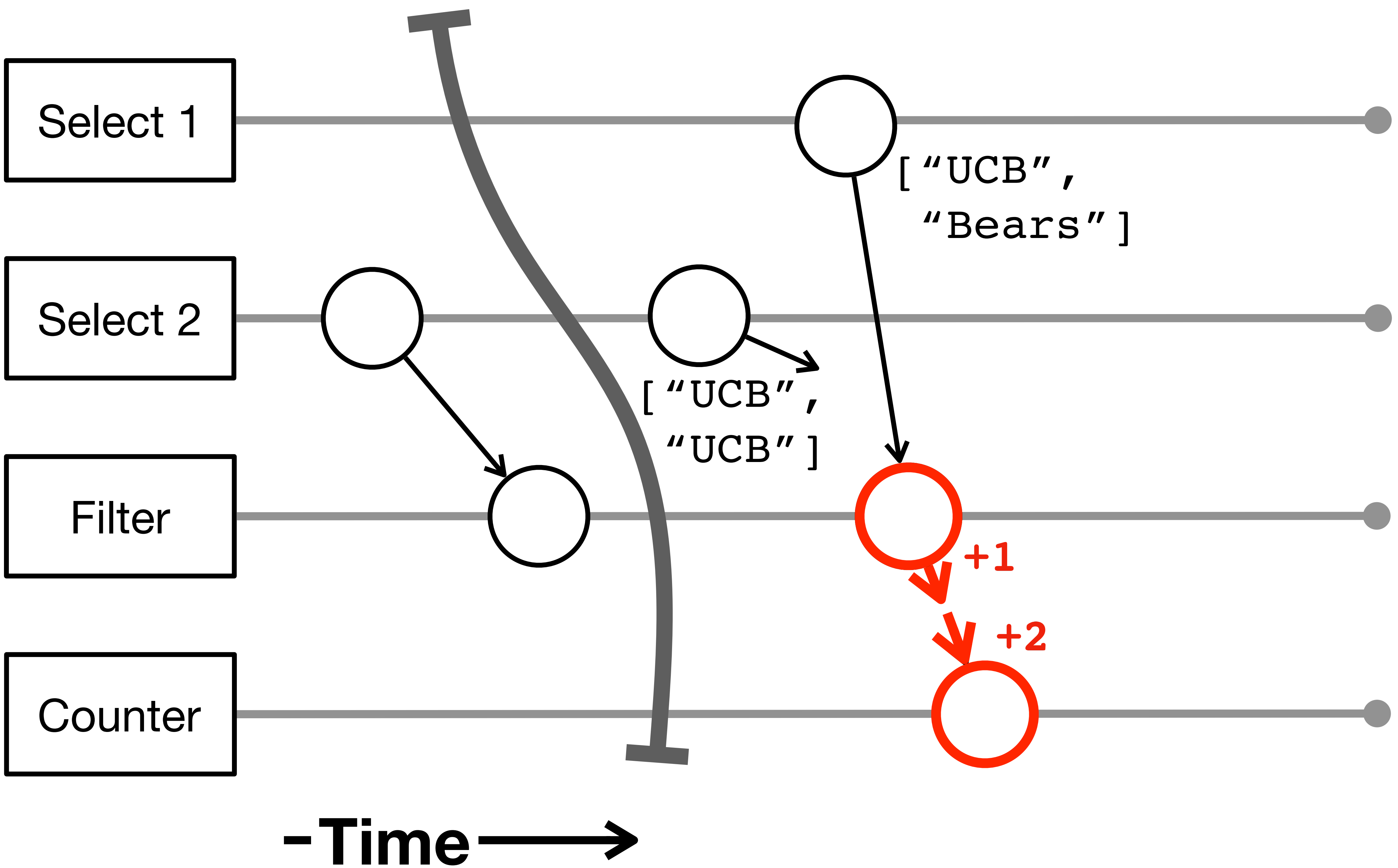
Global checkpointing



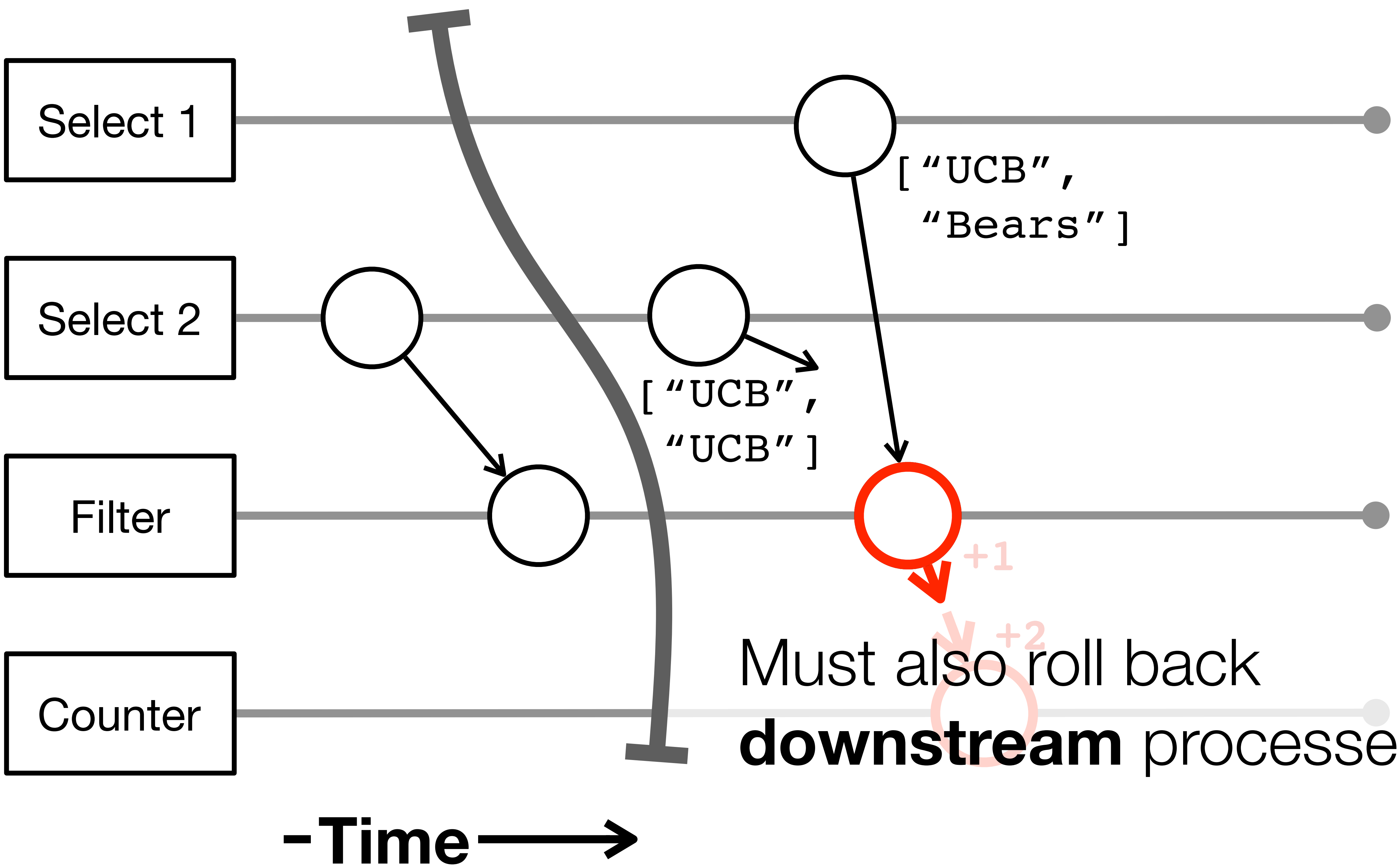
Global checkpointing



Global checkpointing

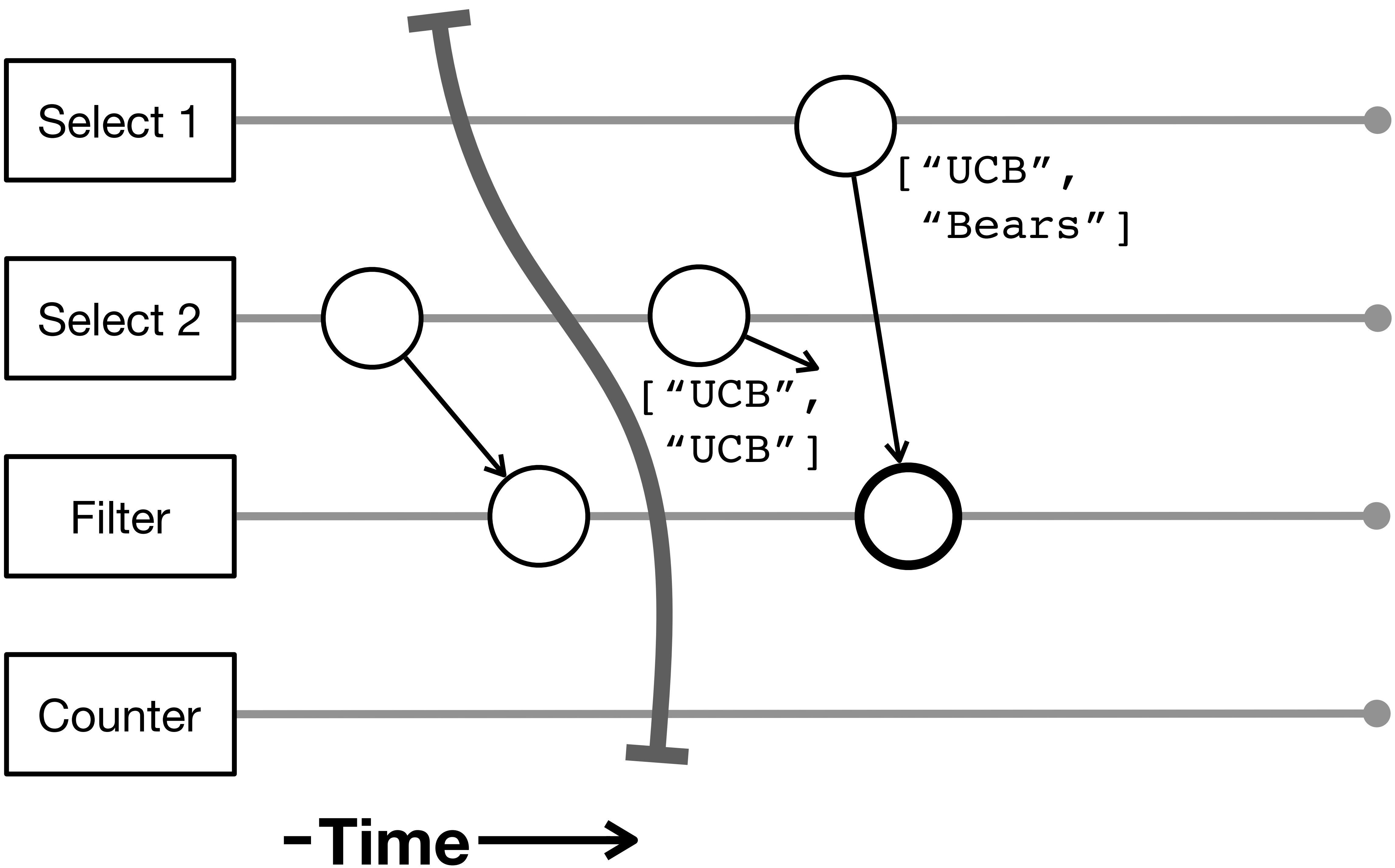


Global checkpointing

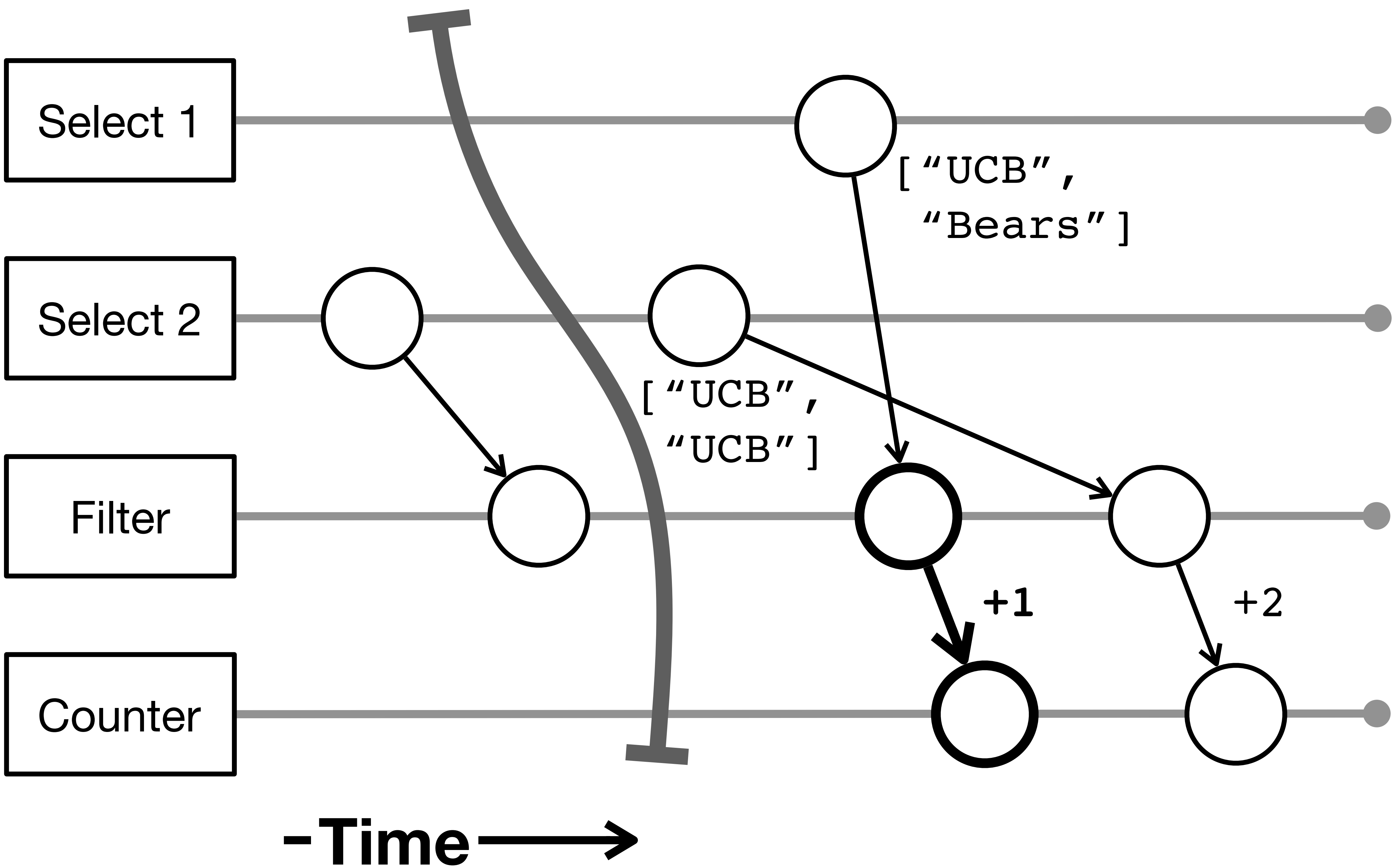


Must also roll back **downstream** processes

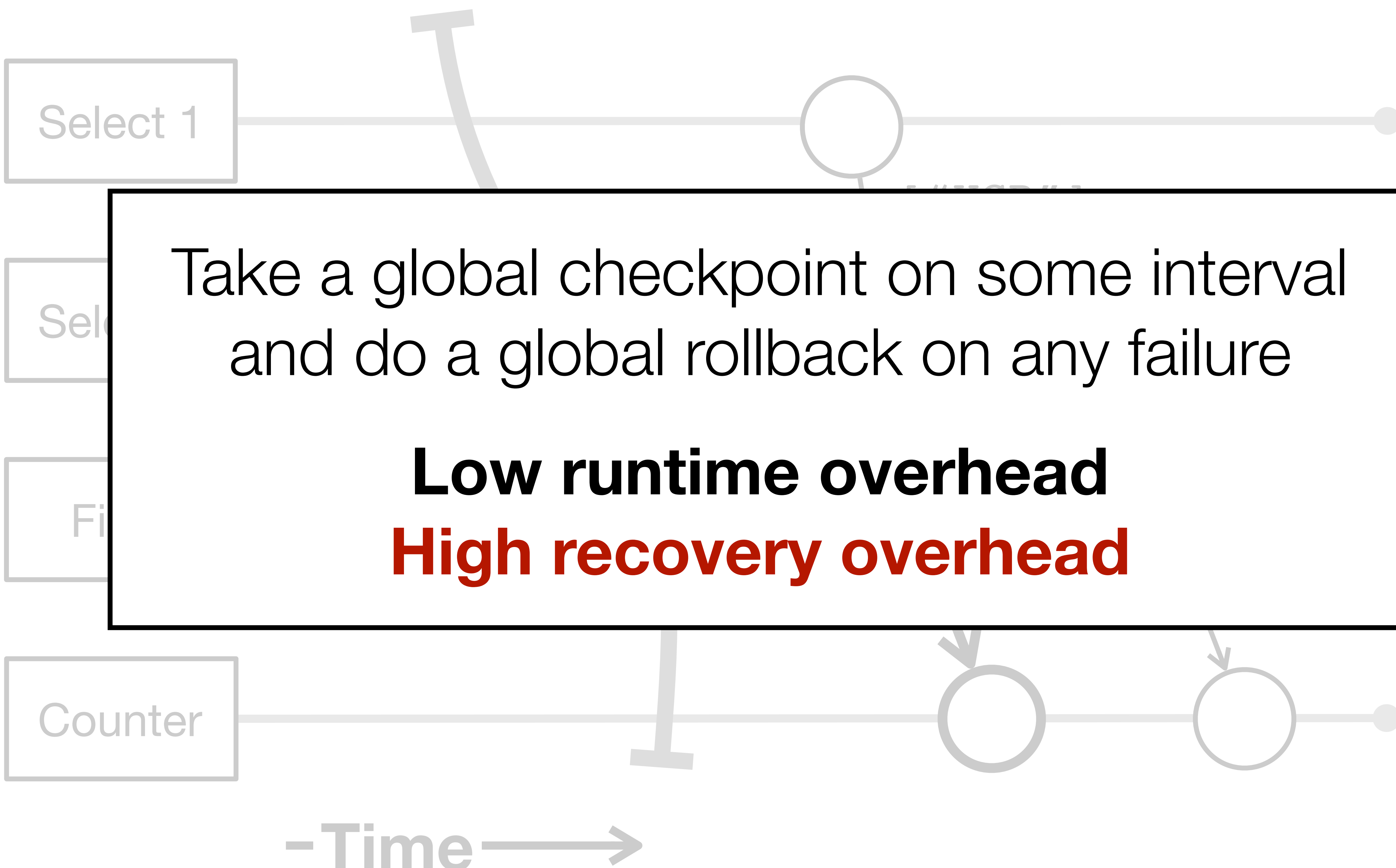
Global checkpointing



Global checkpointing



Global checkpointing

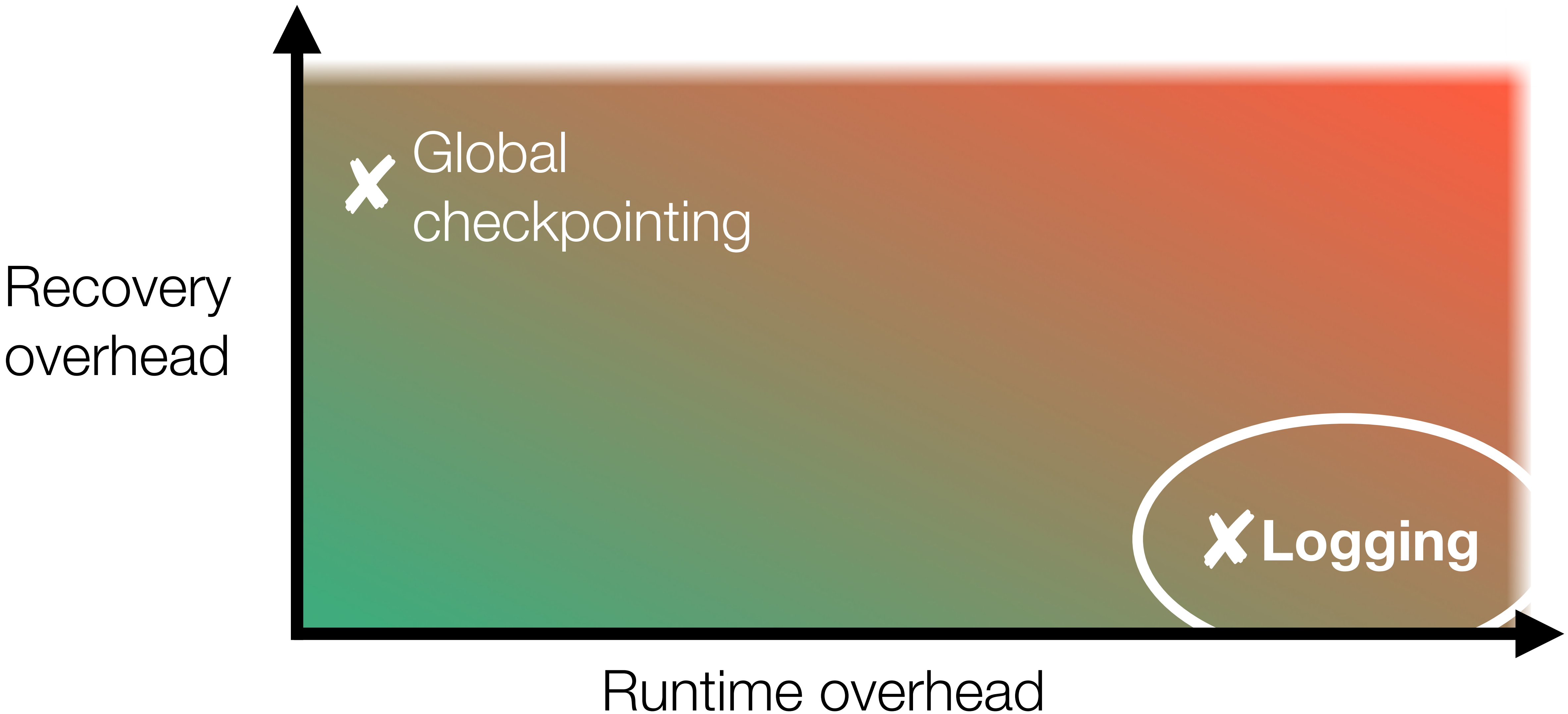


Take a global checkpoint on some interval and do a global rollback on any failure

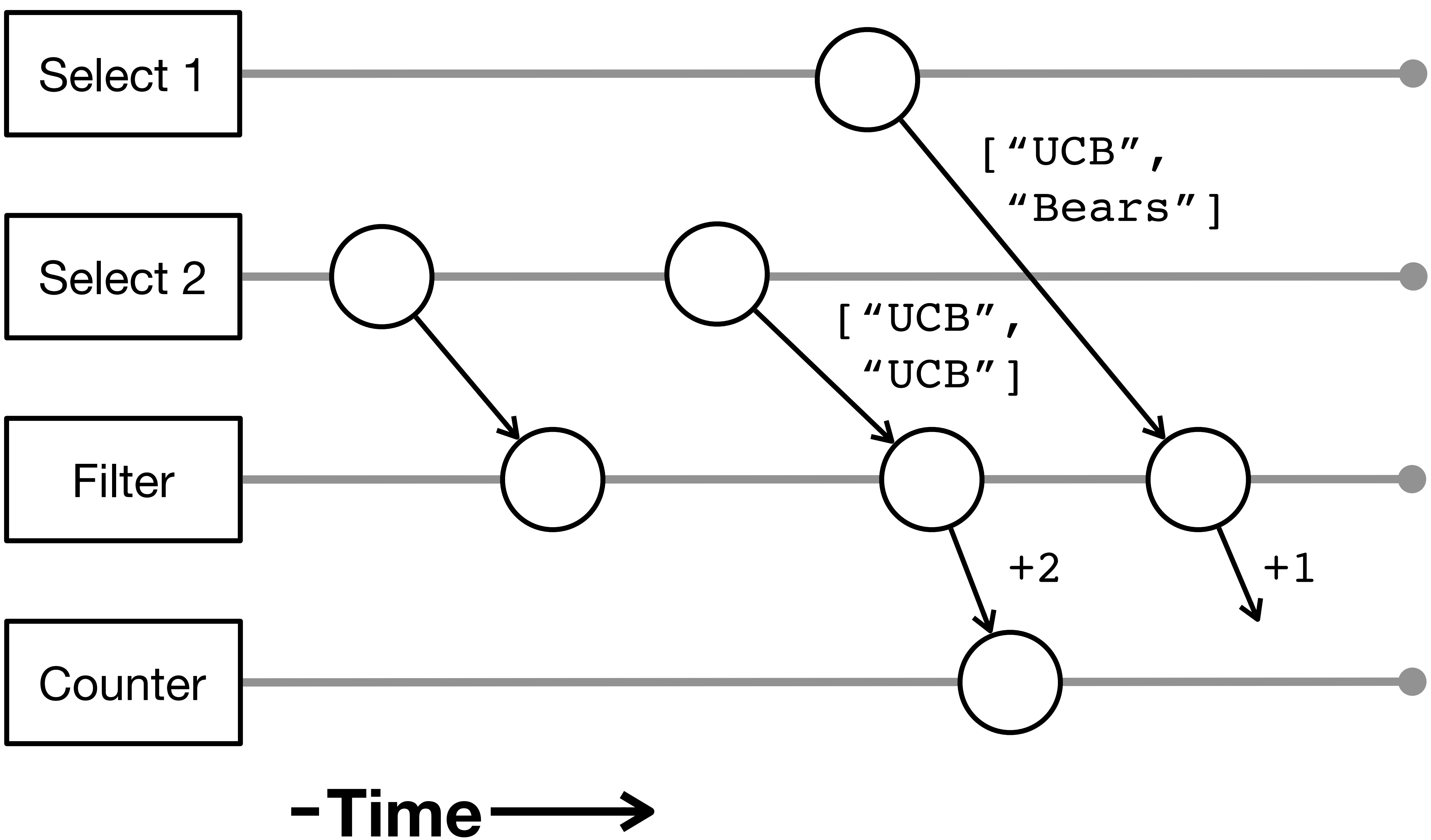
Low runtime overhead

High recovery overhead

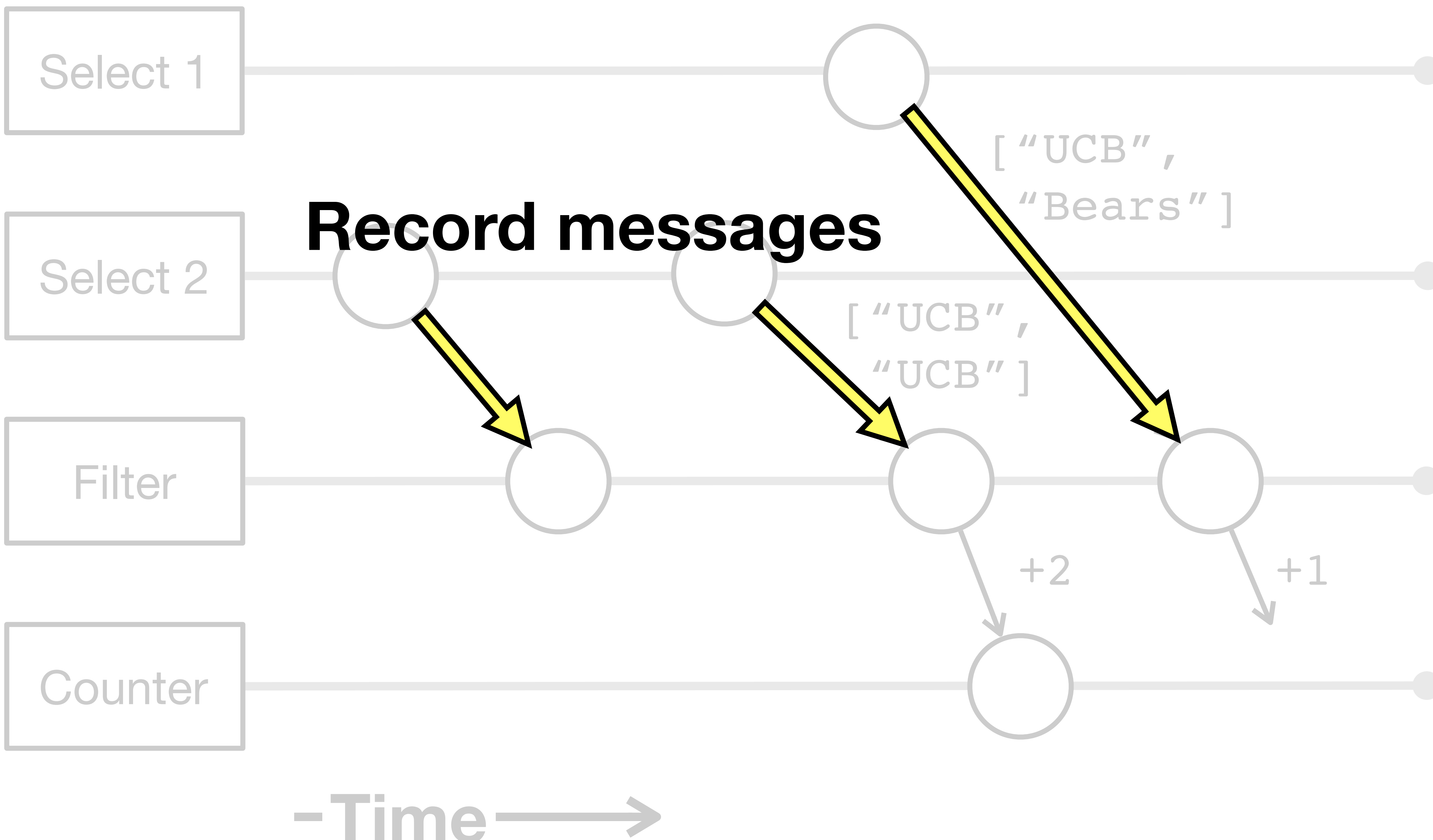
Tradeoff between low latency and recovery time



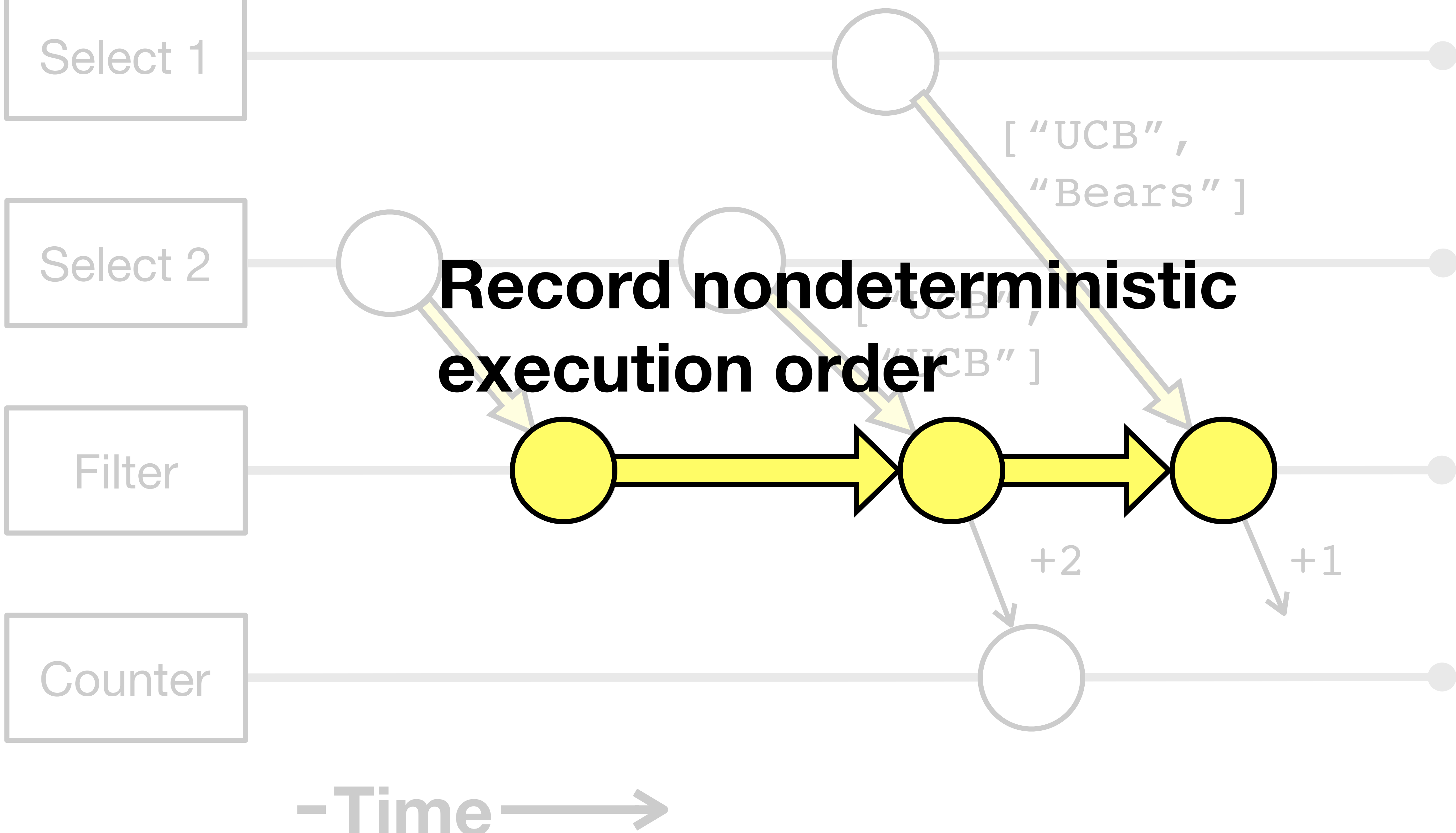
Logging



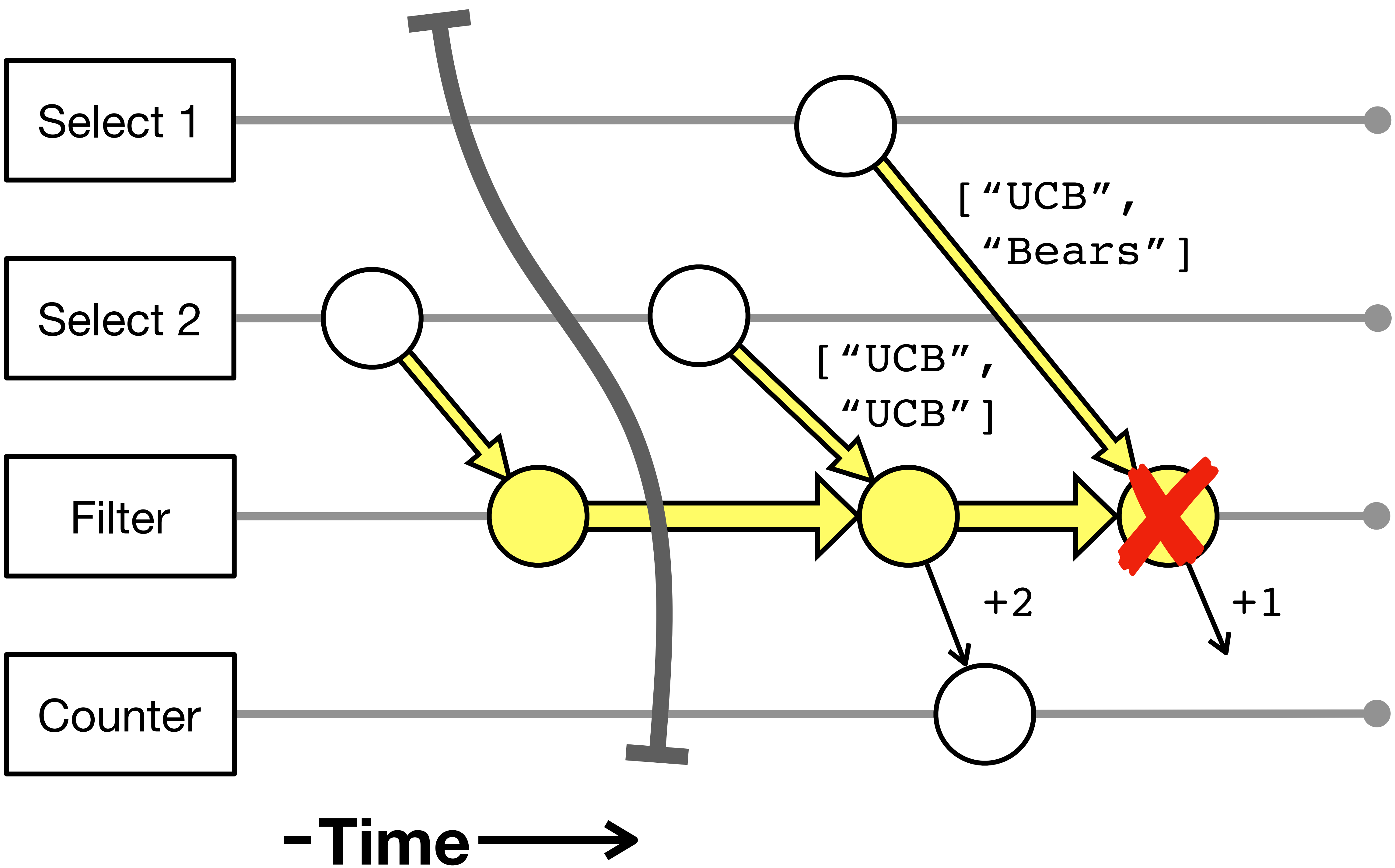
Logging


Select 1
Select 2
Filter
Counter
Record messages
["UCB",
"Bears"]
["UCB",
"UCB"]
+2
+1
Time

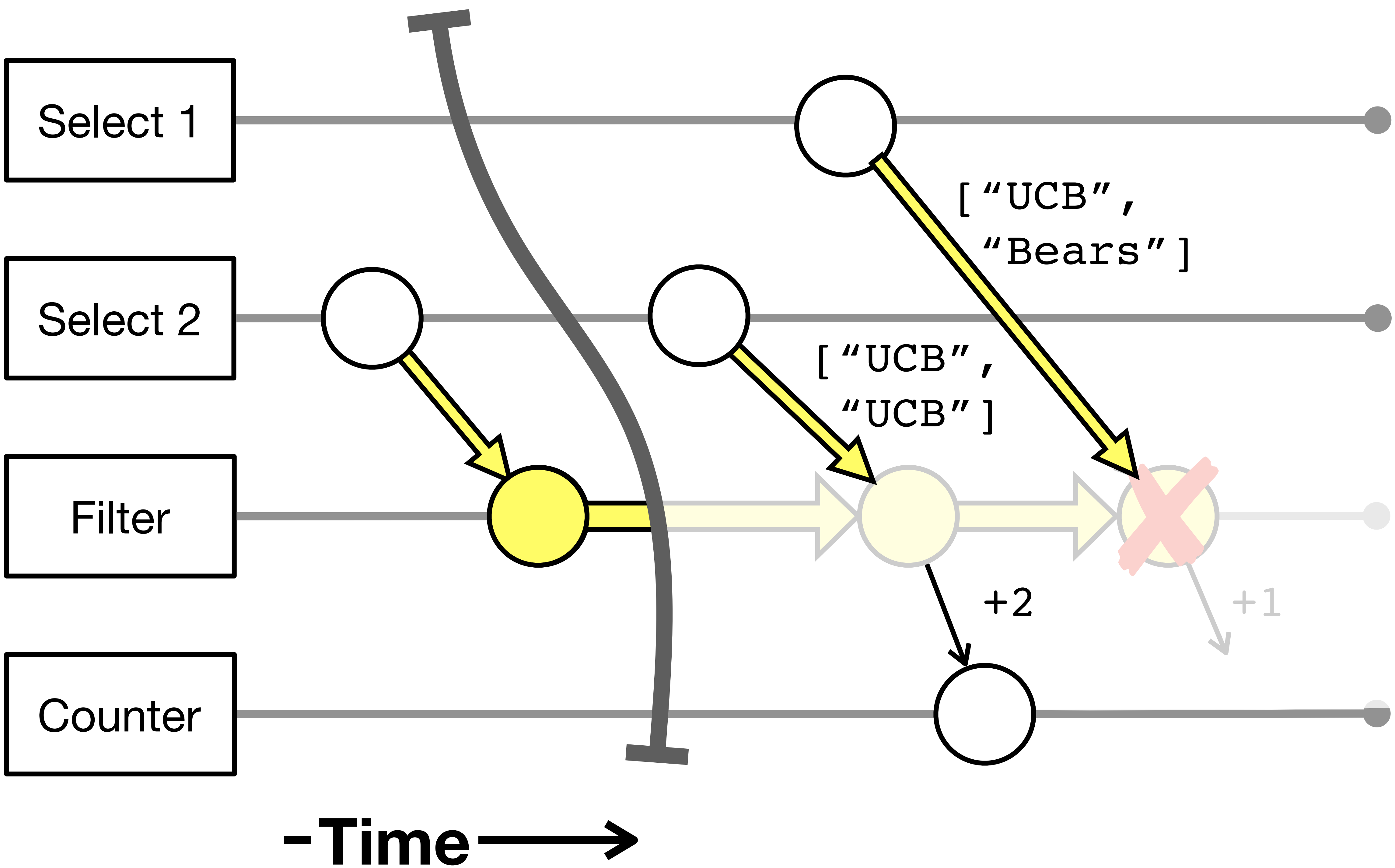
Logging


Select 1
Select 2
Filter
Counter
[“UCB”,
“Bears”]
Record nondeterministic
execution order
+2
+1
Time

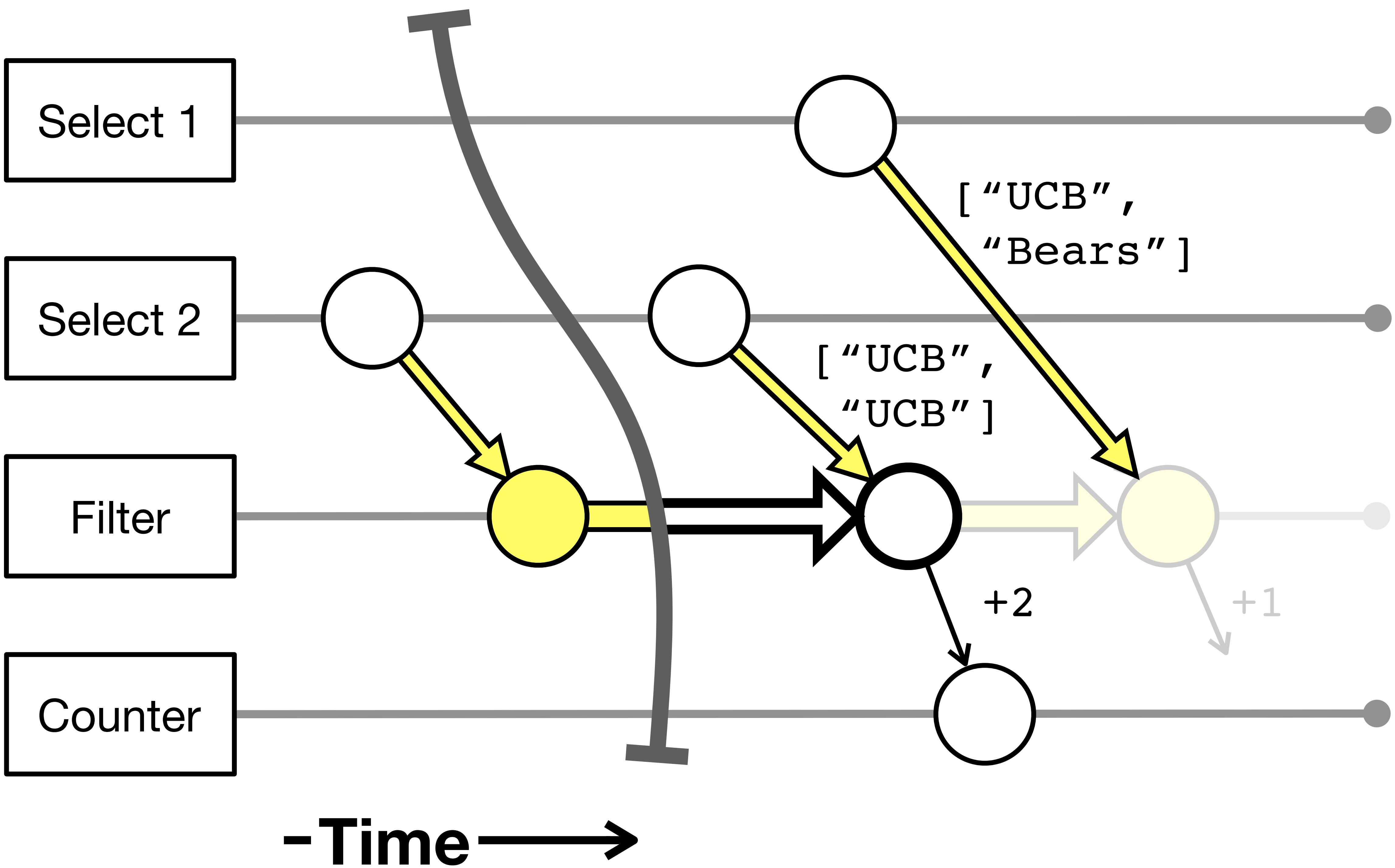
Logging



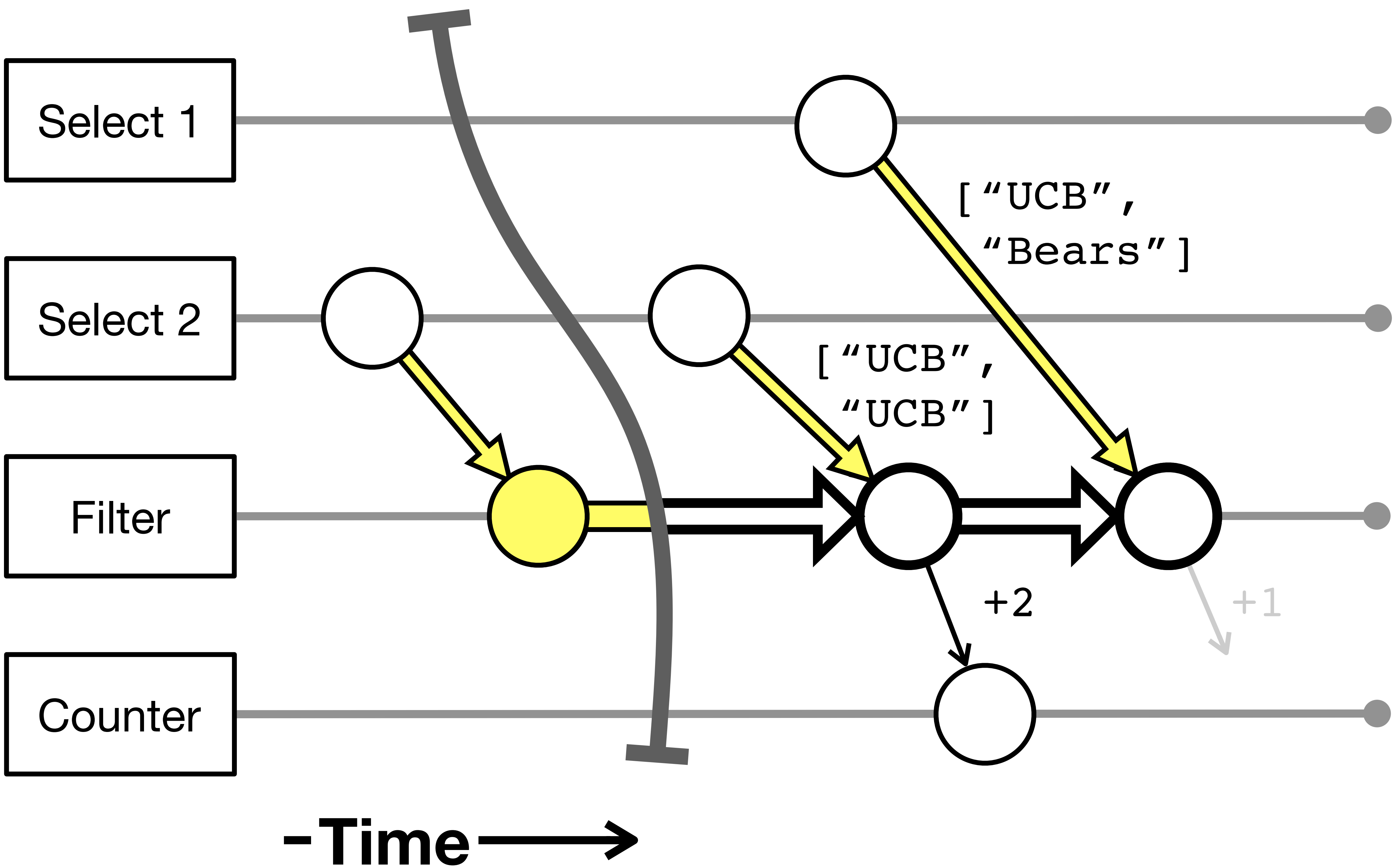
Logging



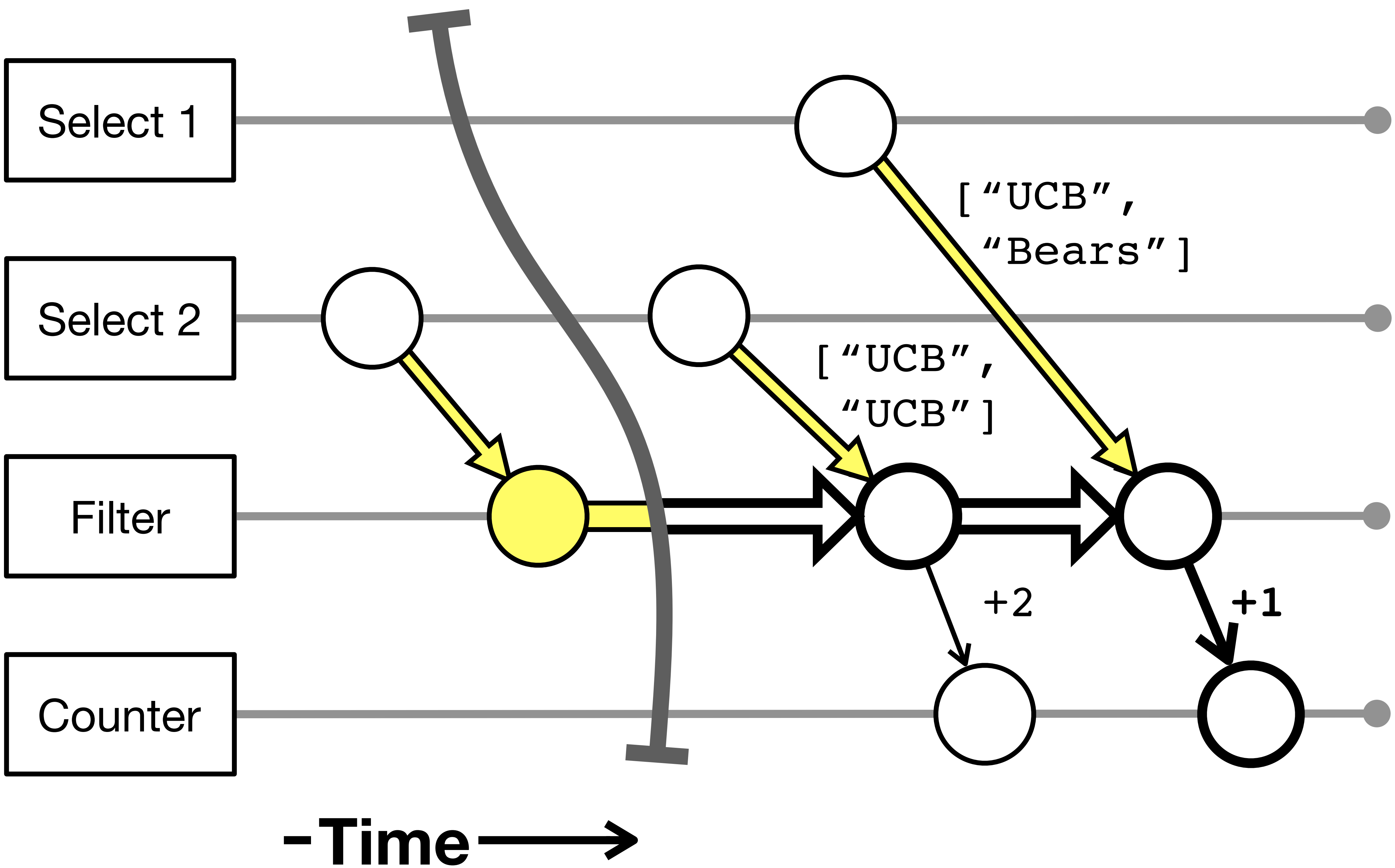
Logging



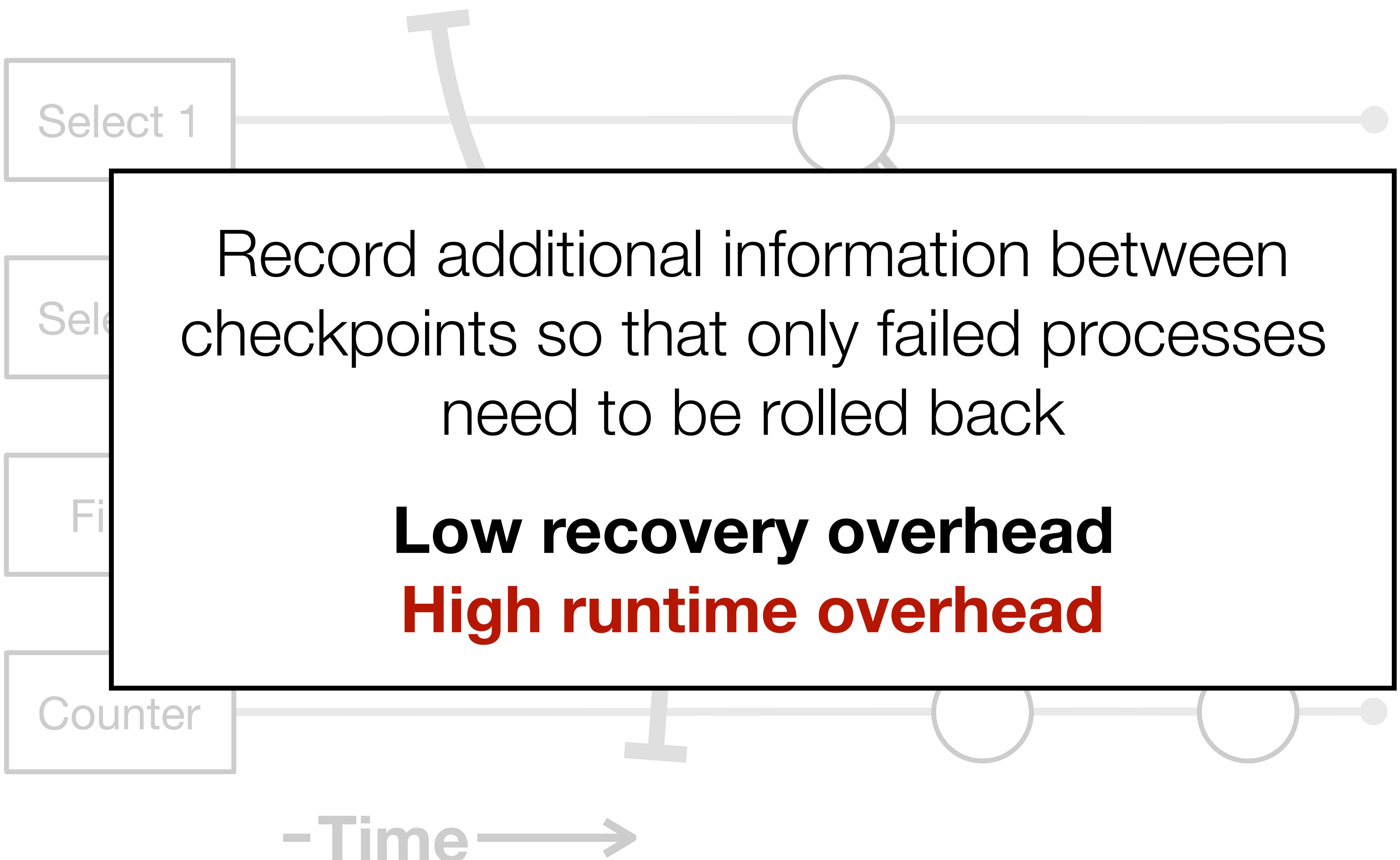
Logging



Logging



Logging

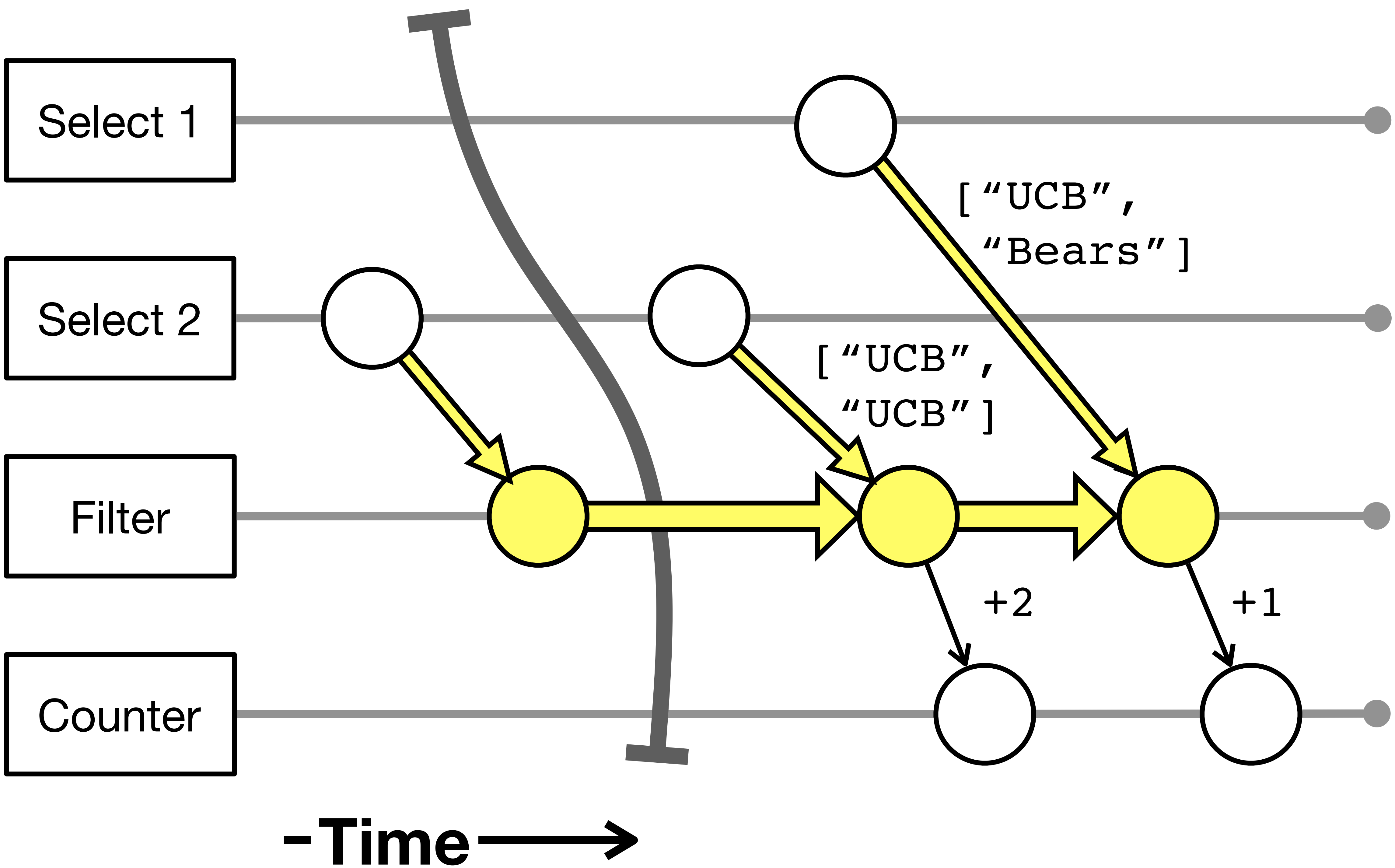


Record additional information between checkpoints so that only failed processes need to be rolled back

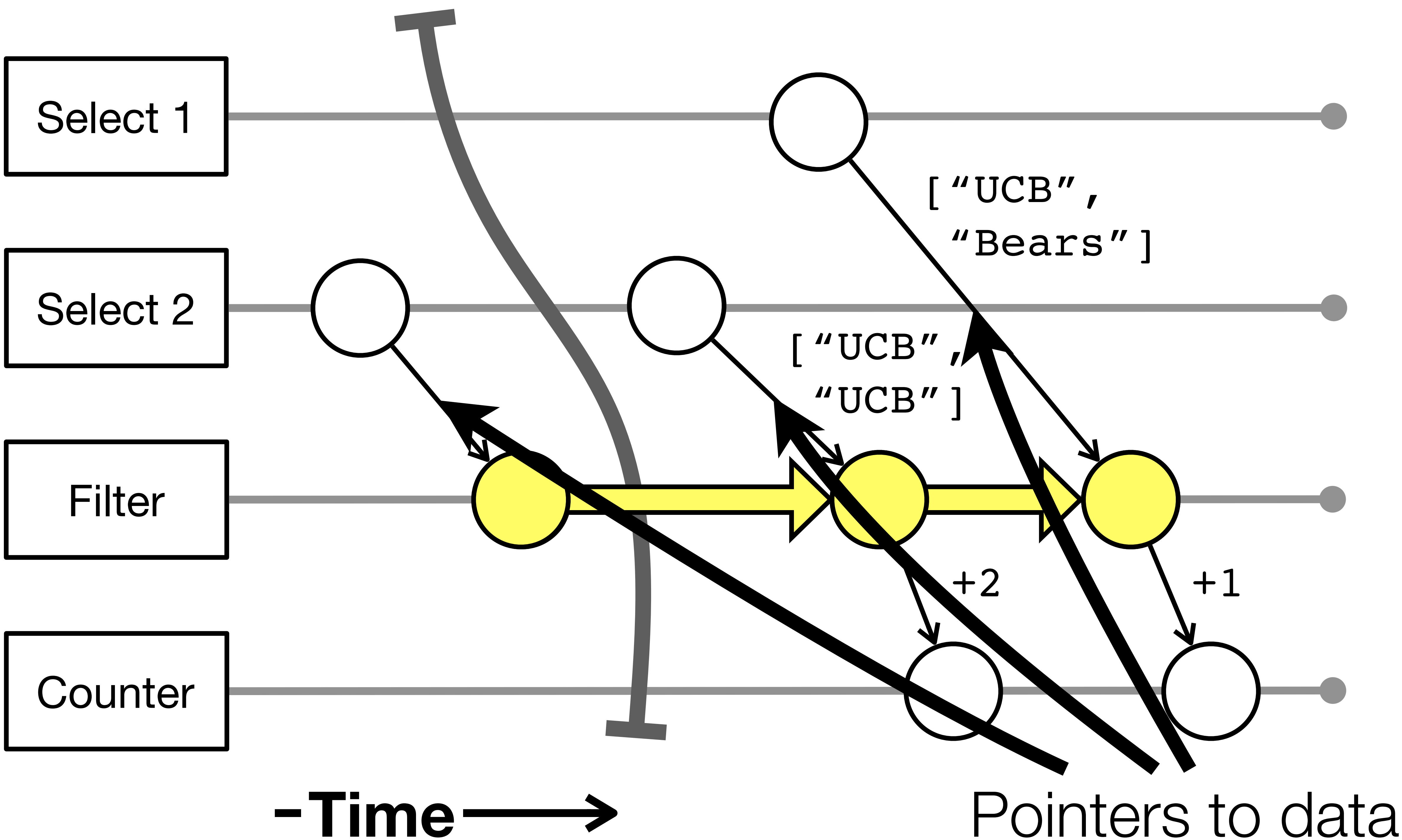
Low recovery overhead

High runtime overhead

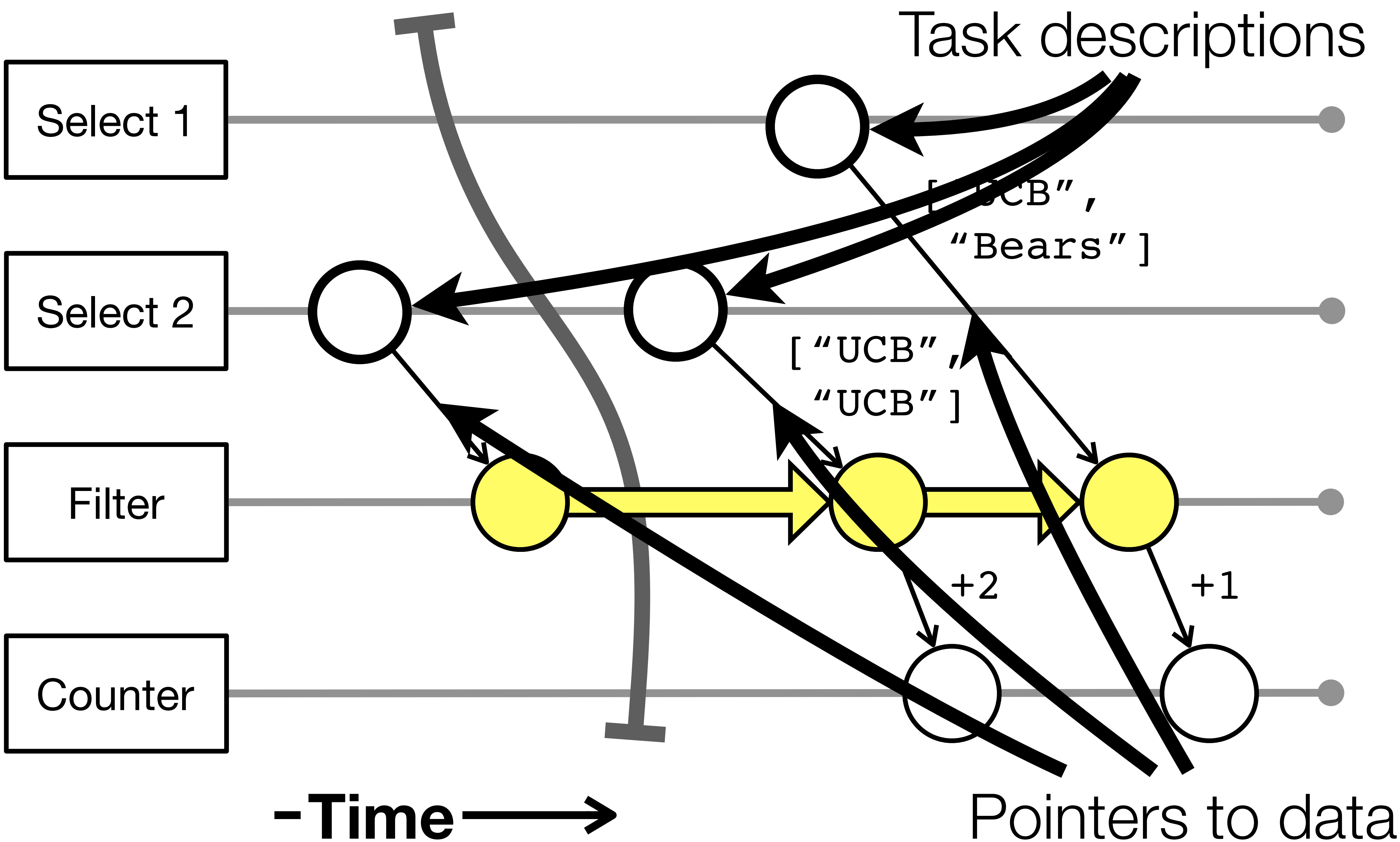
Log the **lineage** to reduce the **amount** logged



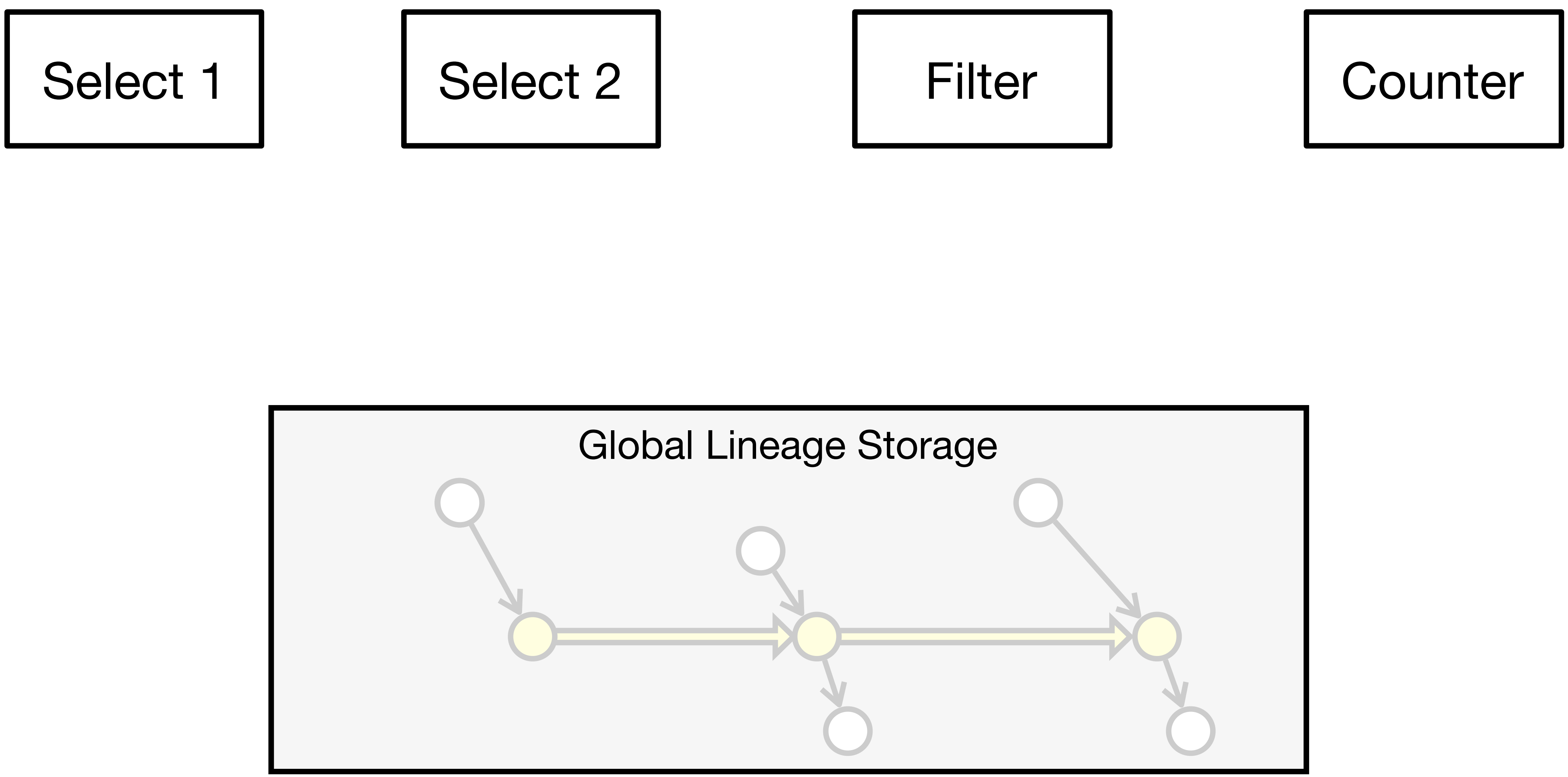
Log the **lineage** to reduce the **amount** logged



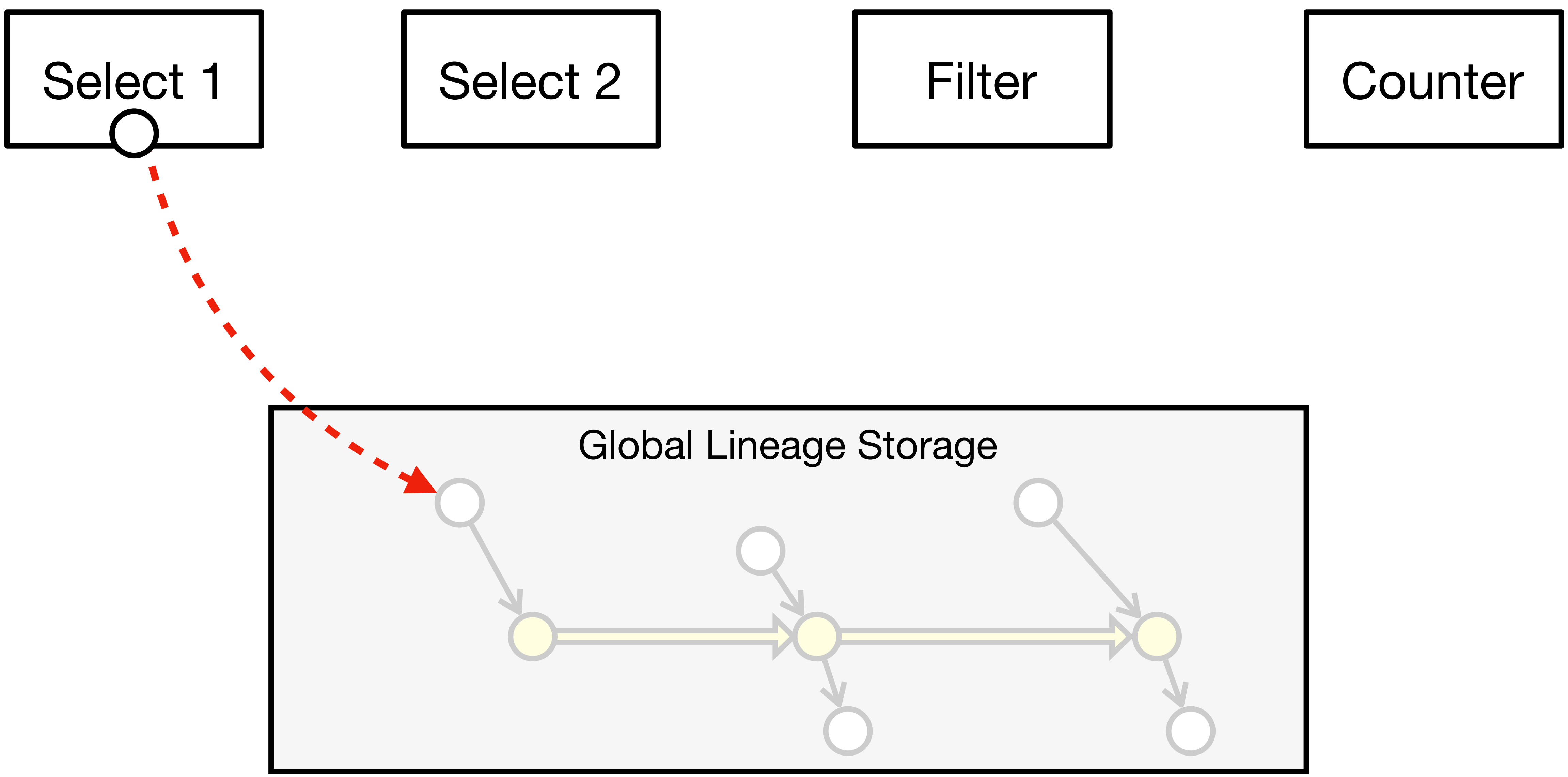
Log the lineage to reduce the amount logged



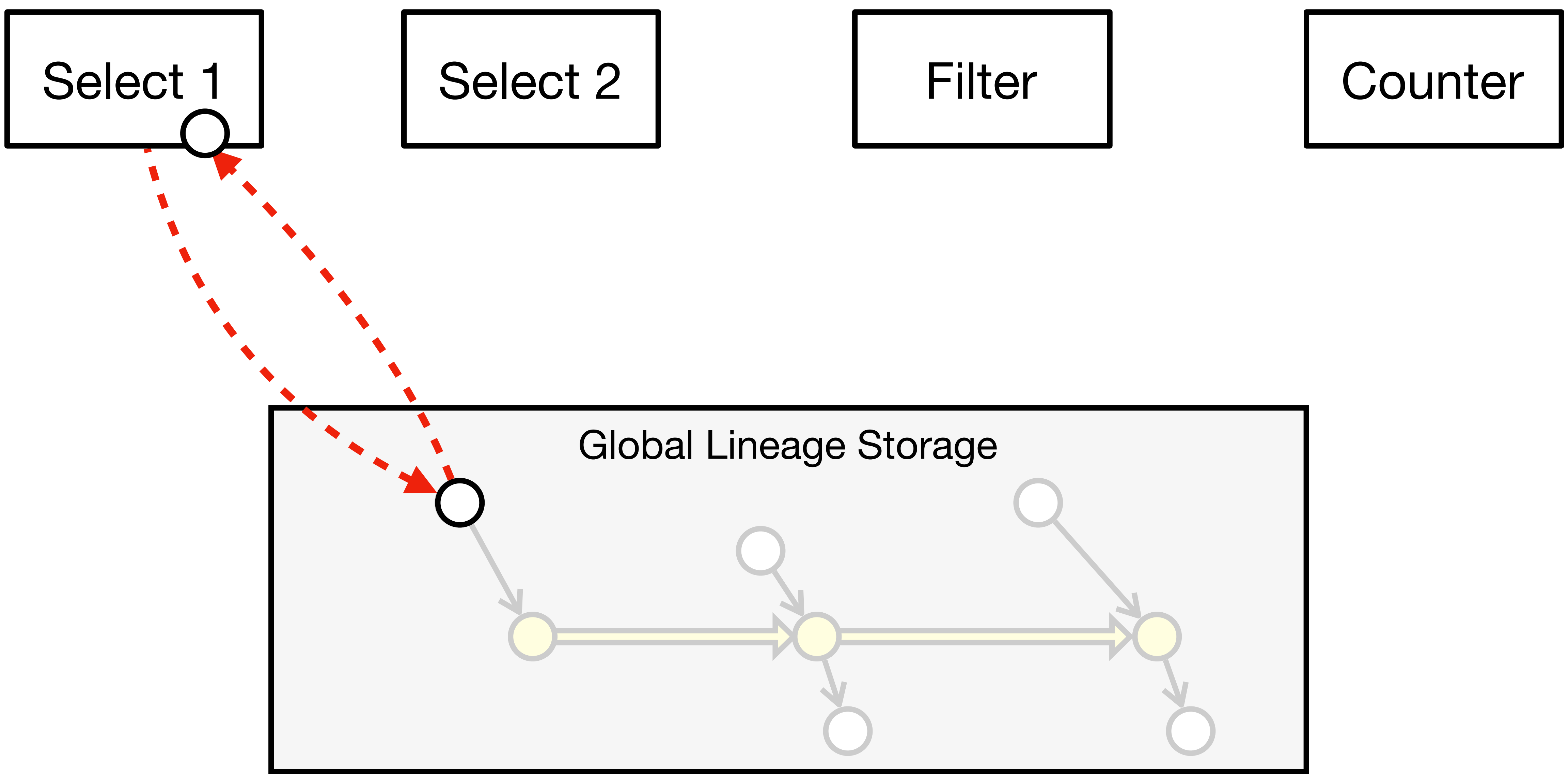
But logging still requires a **synchronous** round-trip to remote storage



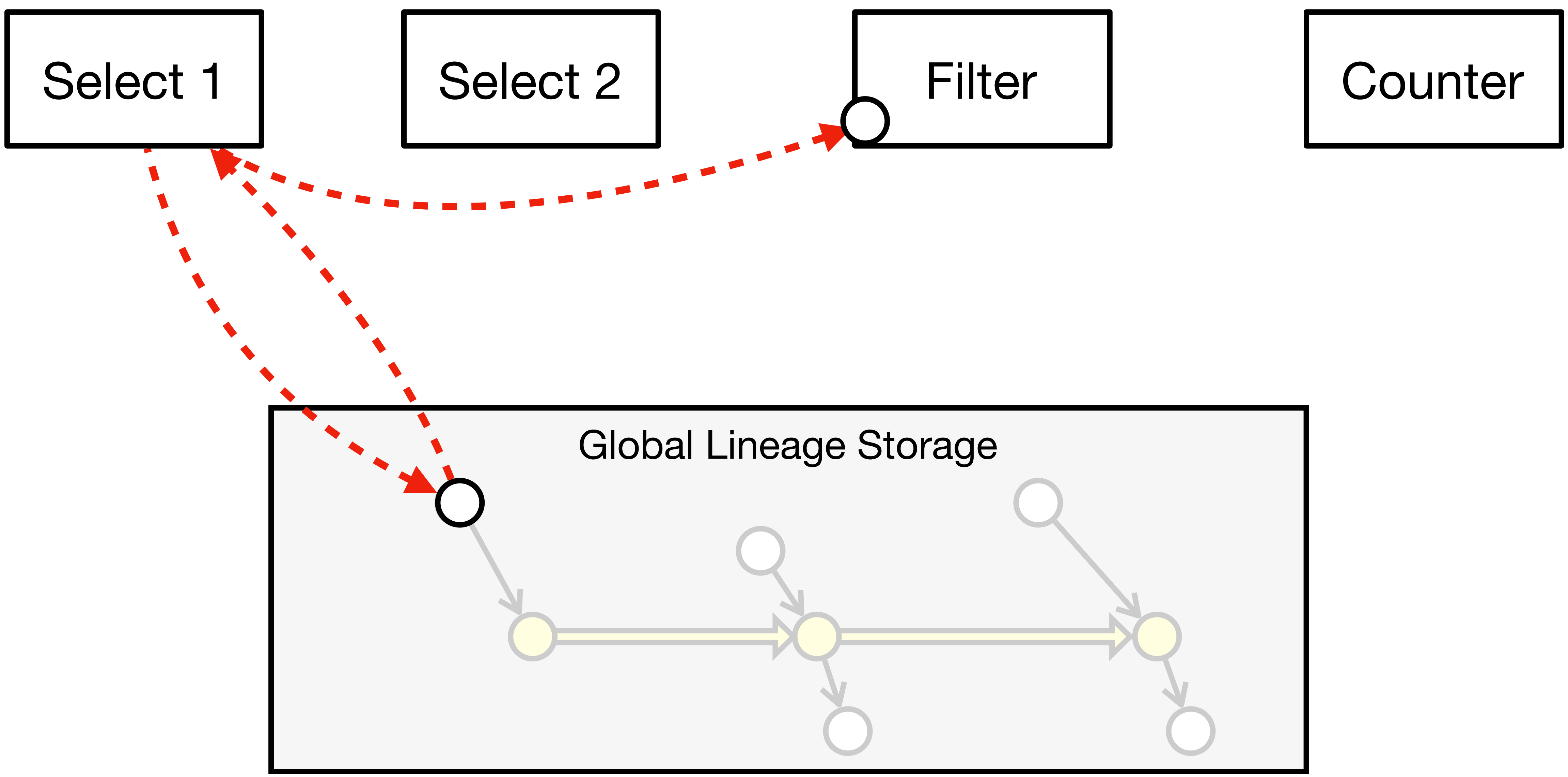
But logging still requires a **synchronous** round-trip to remote storage



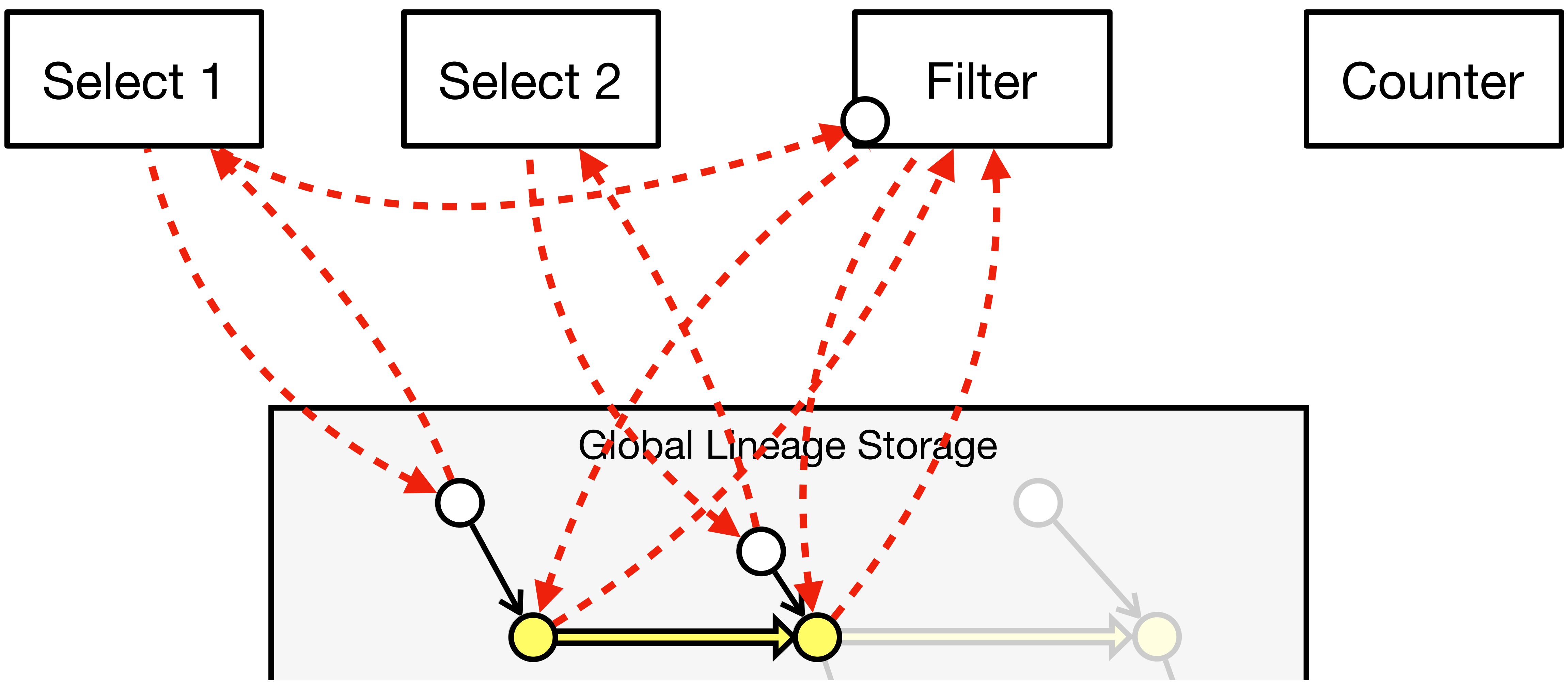
But logging still requires a **synchronous** round-trip to remote storage



But logging still requires a **synchronous** round-trip to remote storage



But logging still requires a **synchronous** round-trip to remote storage



Scheduling delay / task >= **1RTT** + 1RPC

Task latency depends on global storage latency

Lineage stash contribution

How do we achieve both **low runtime *and* low recovery overhead** for **fine-grained** data processing applications?

Solution: Asynchronously log the lineage off the critical path of execution.

Lineage reconstruction to reduce amount logged

Causal logging to log nondeterminism

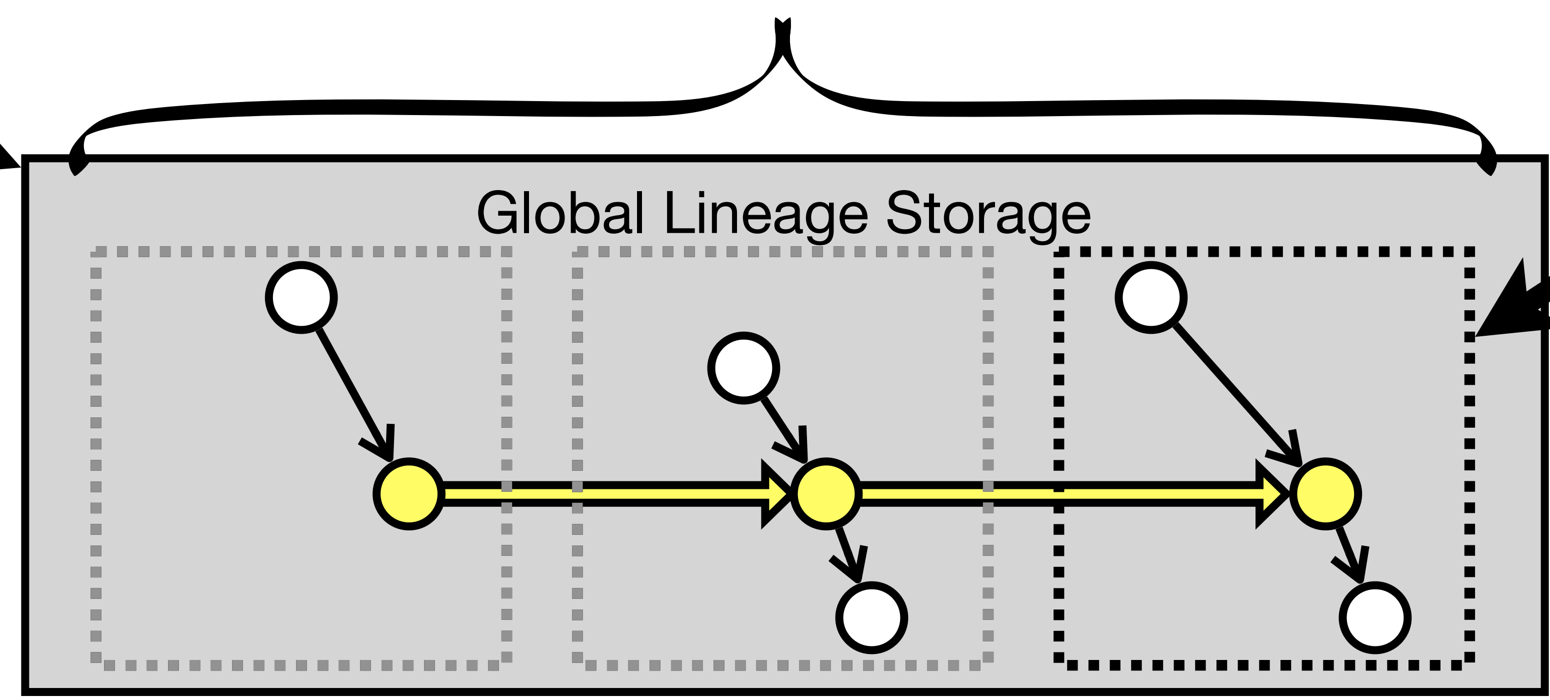
Lineage stash architecture

Replicated for **durability**.

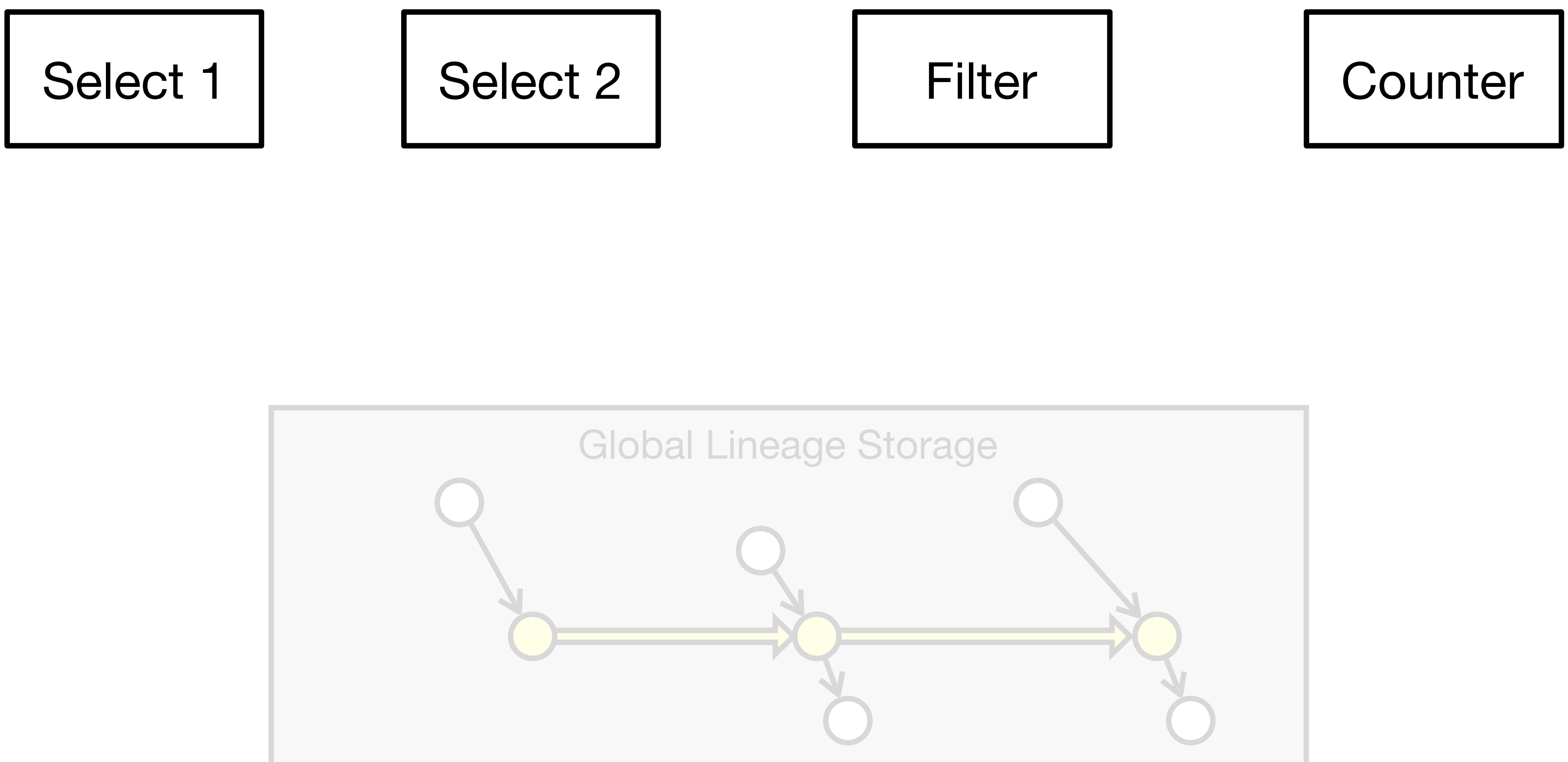
Select 1 Select 2 Filter Counter

Persistent key-value store.

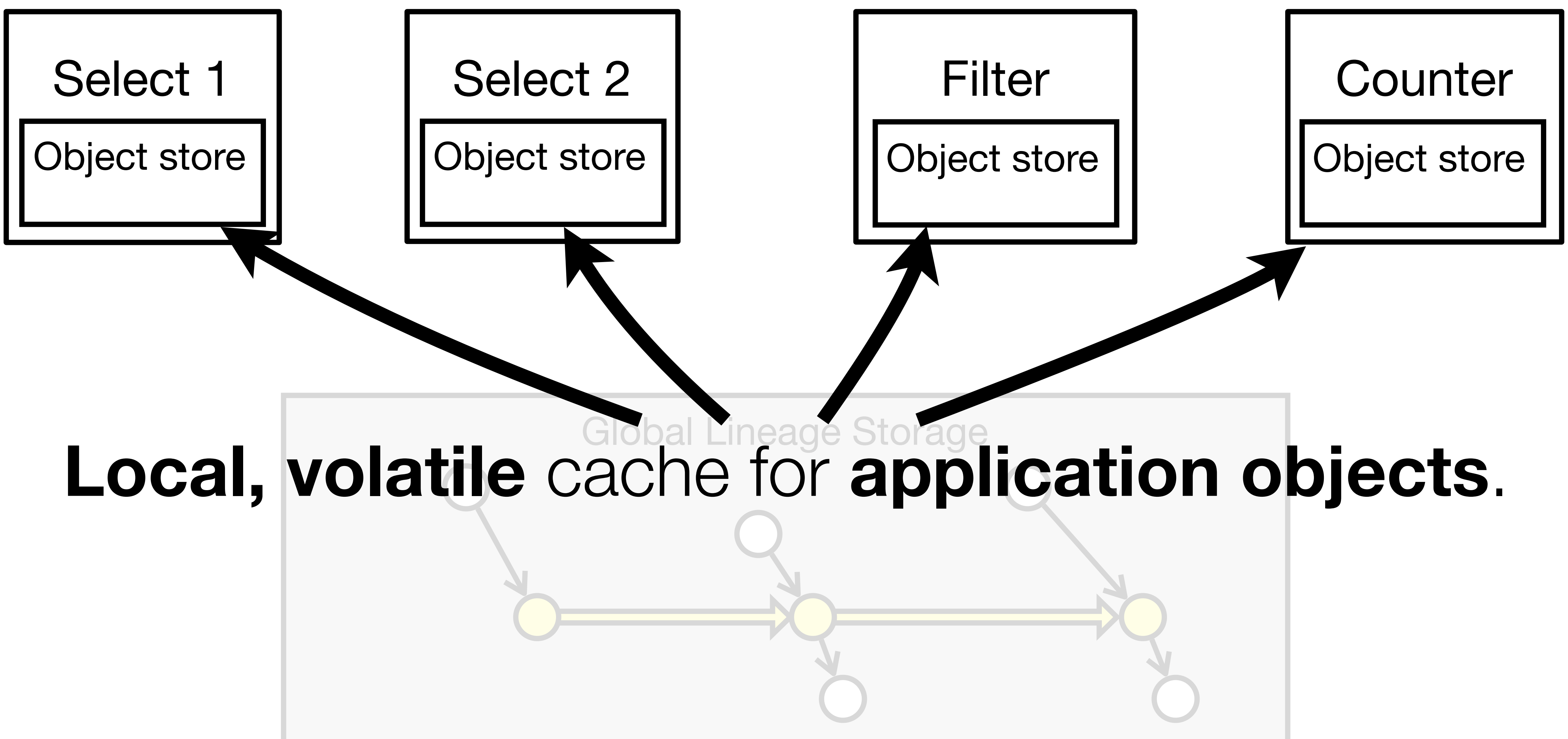
Sharded for horizontal **scalability**



Lineage stash architecture

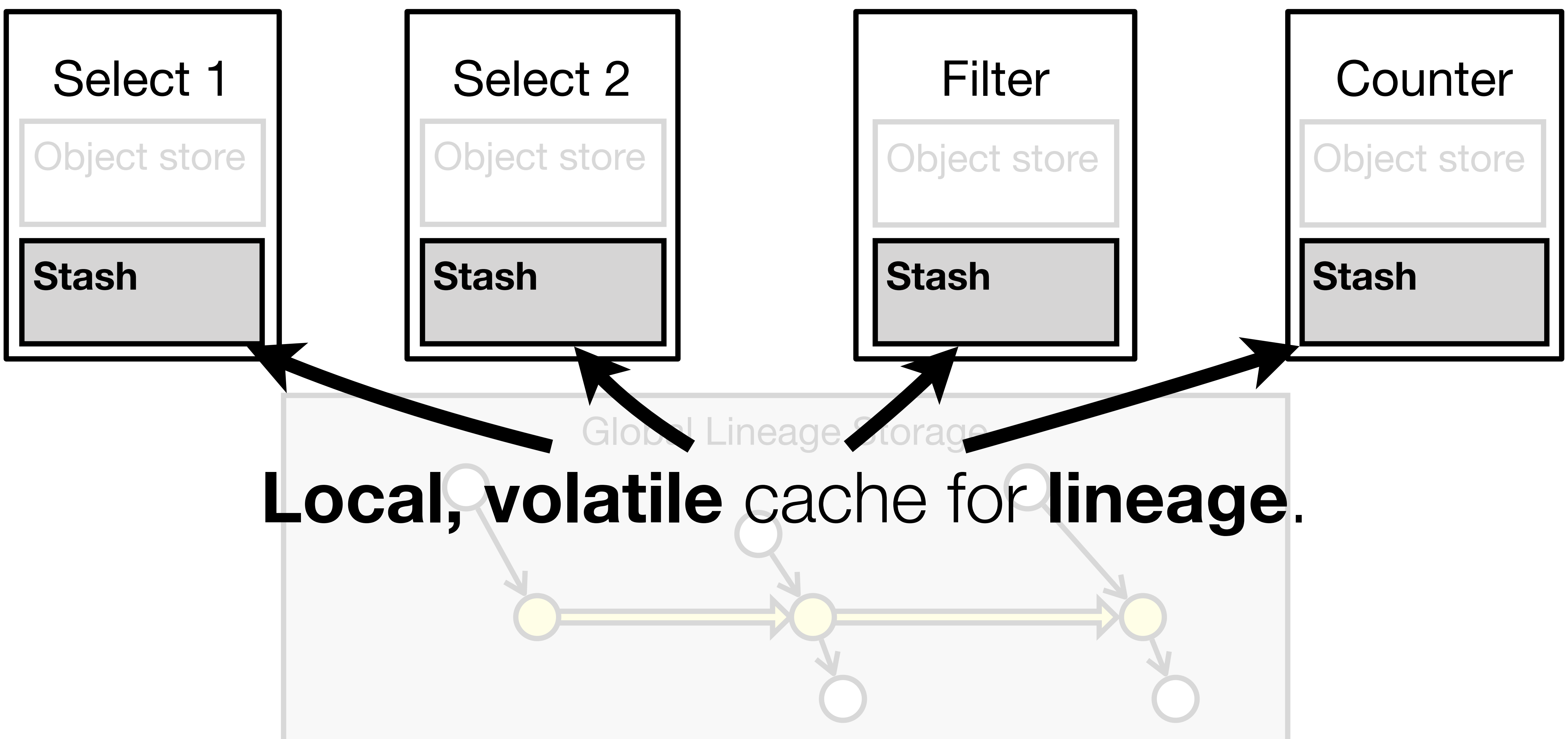


Lineage stash architecture



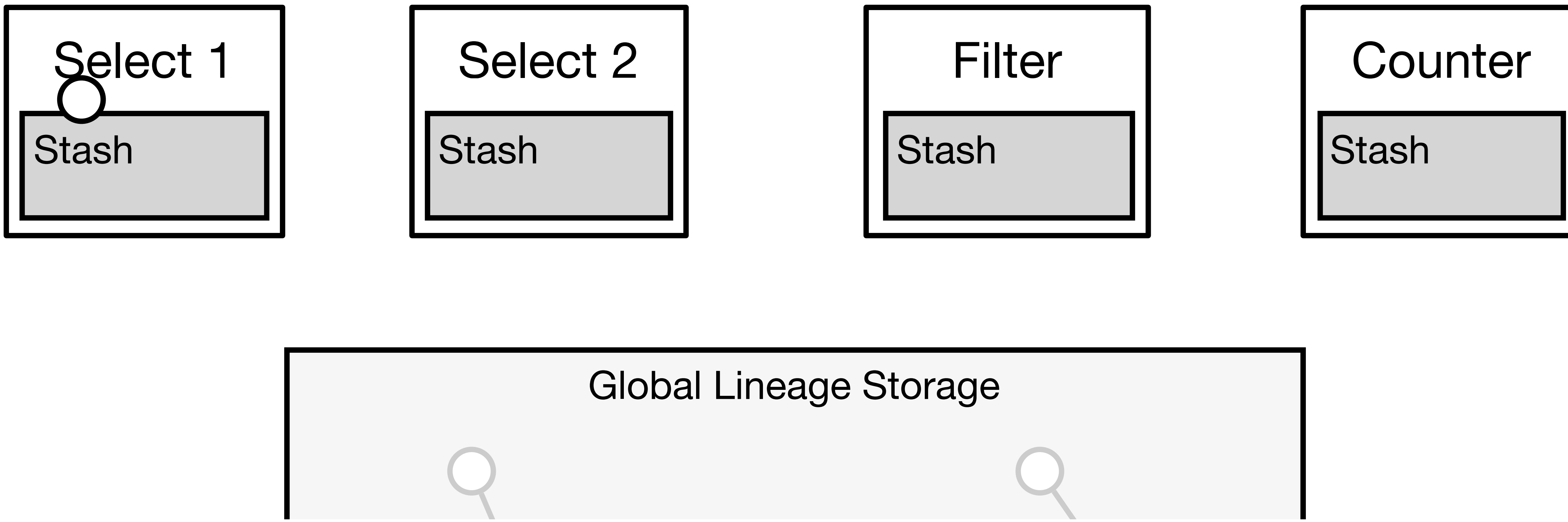
Local, volatile cache for **application objects**.

Lineage stash architecture



Local, volatile cache for **lineage**.

Lineage stash: Logging the **lineage, asynchronously**



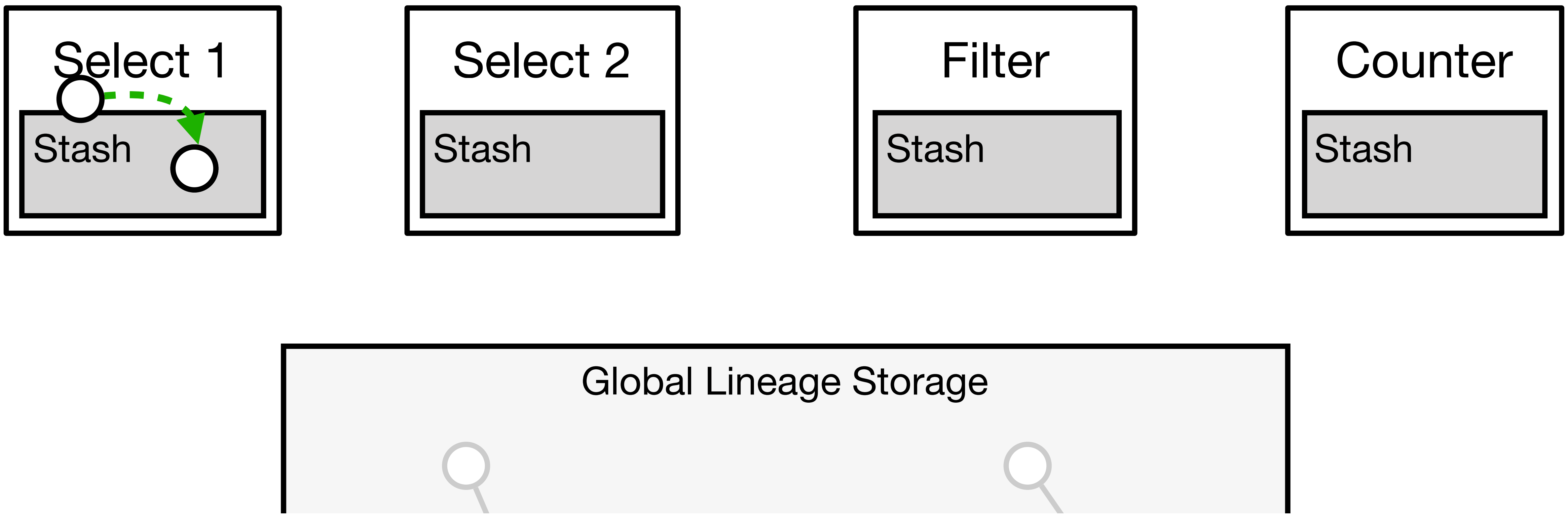
(1) Write lineage to **local, volatile** lineage stash.

(2) **Asynchronously** flush to **remote** storage.

Scheduling delay / task = ~~1RTT~~ + **1RPC**

Task latency **independent of** global storage latency

Lineage stash: Logging the **lineage, asynchronously**



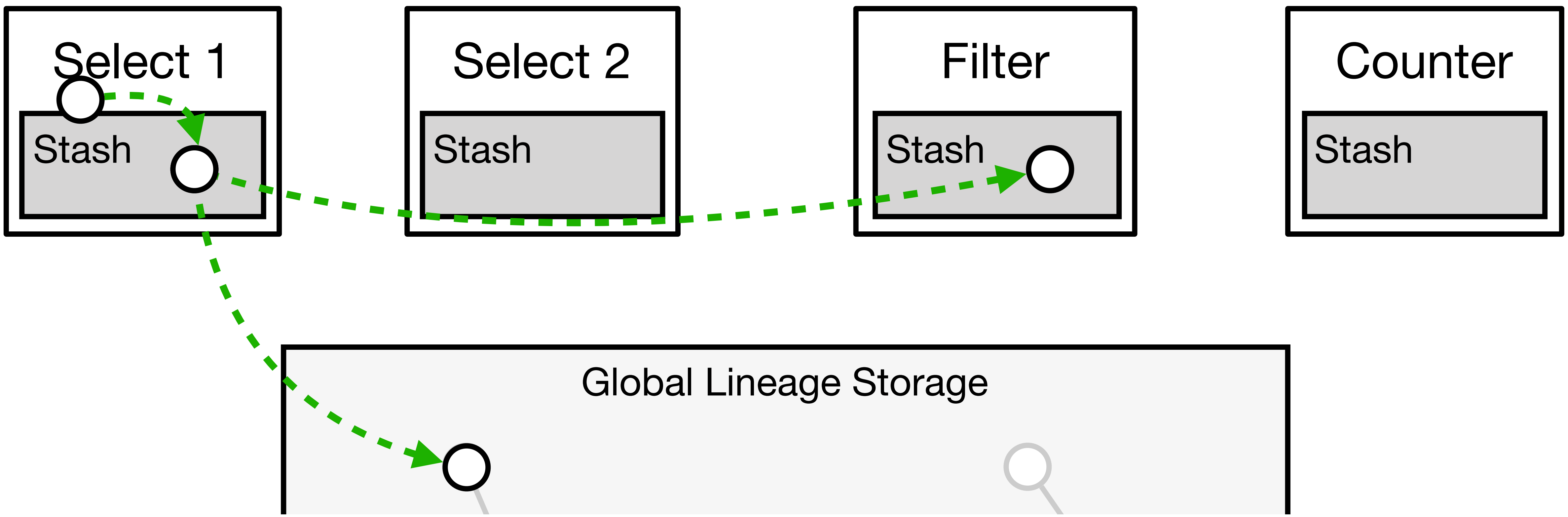
(1) Write lineage to **local, volatile** lineage stash.

(2) **Asynchronously** flush to **remote** storage.

Scheduling delay / task = ~~1RTT~~ + **1RPC**

Task latency **independent of** global storage latency

Lineage stash: Logging the **lineage, asynchronously**



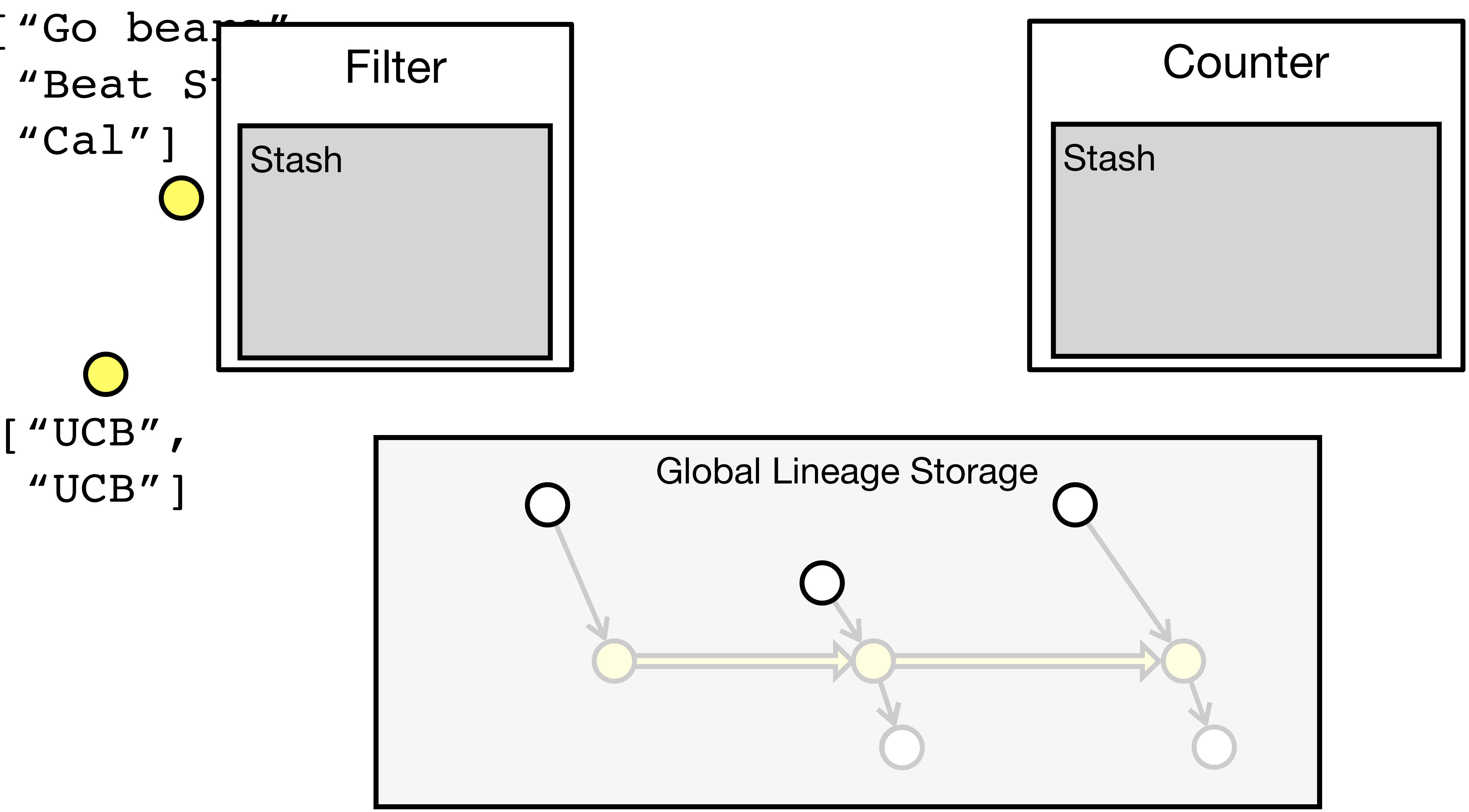
(1) Write lineage to **local, volatile** lineage stash.

(2) **Asynchronously** flush to **remote** storage.

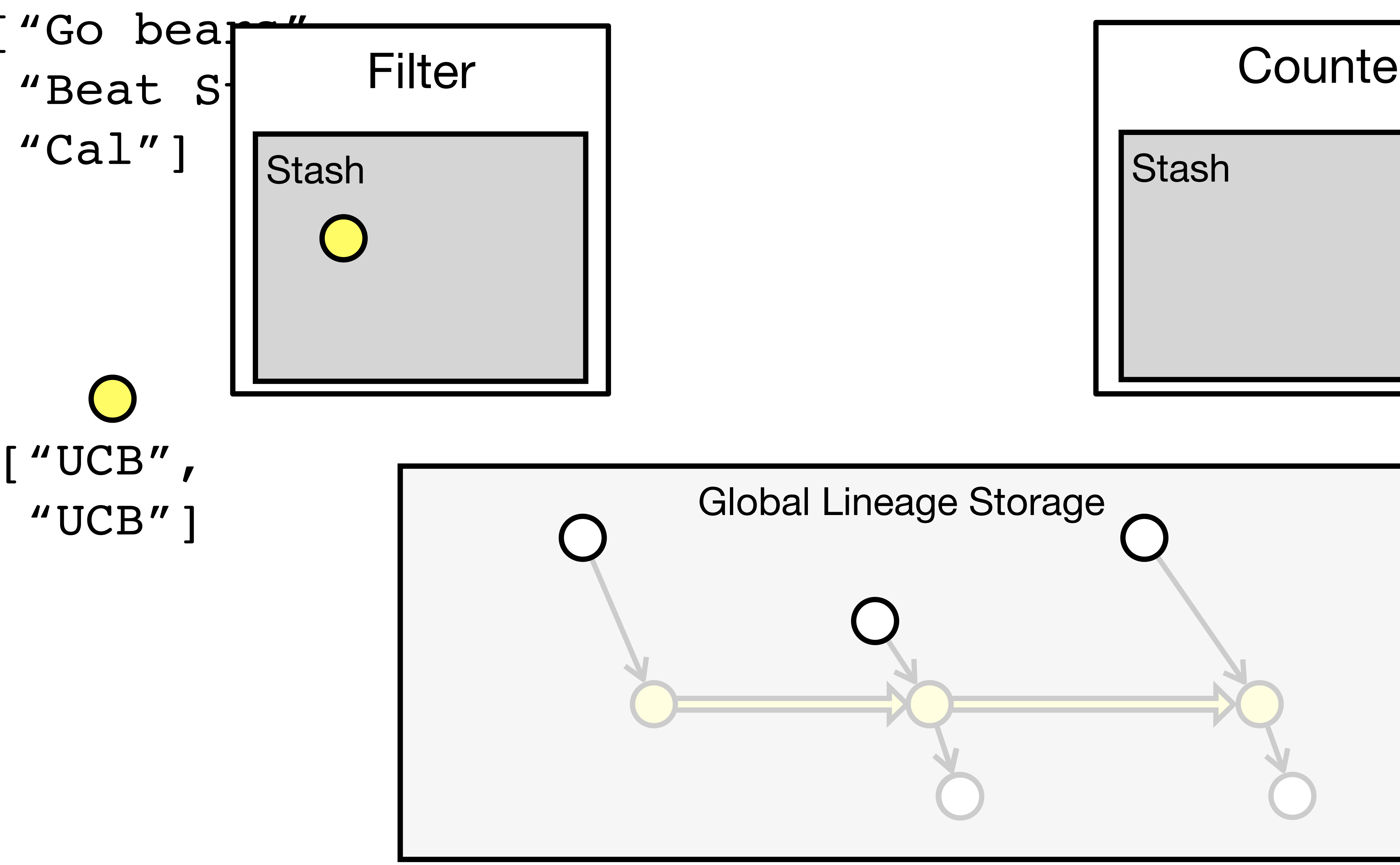
Scheduling delay / task = ~~1RTT~~ + **1RPC**

Task latency **independent of** global storage latency

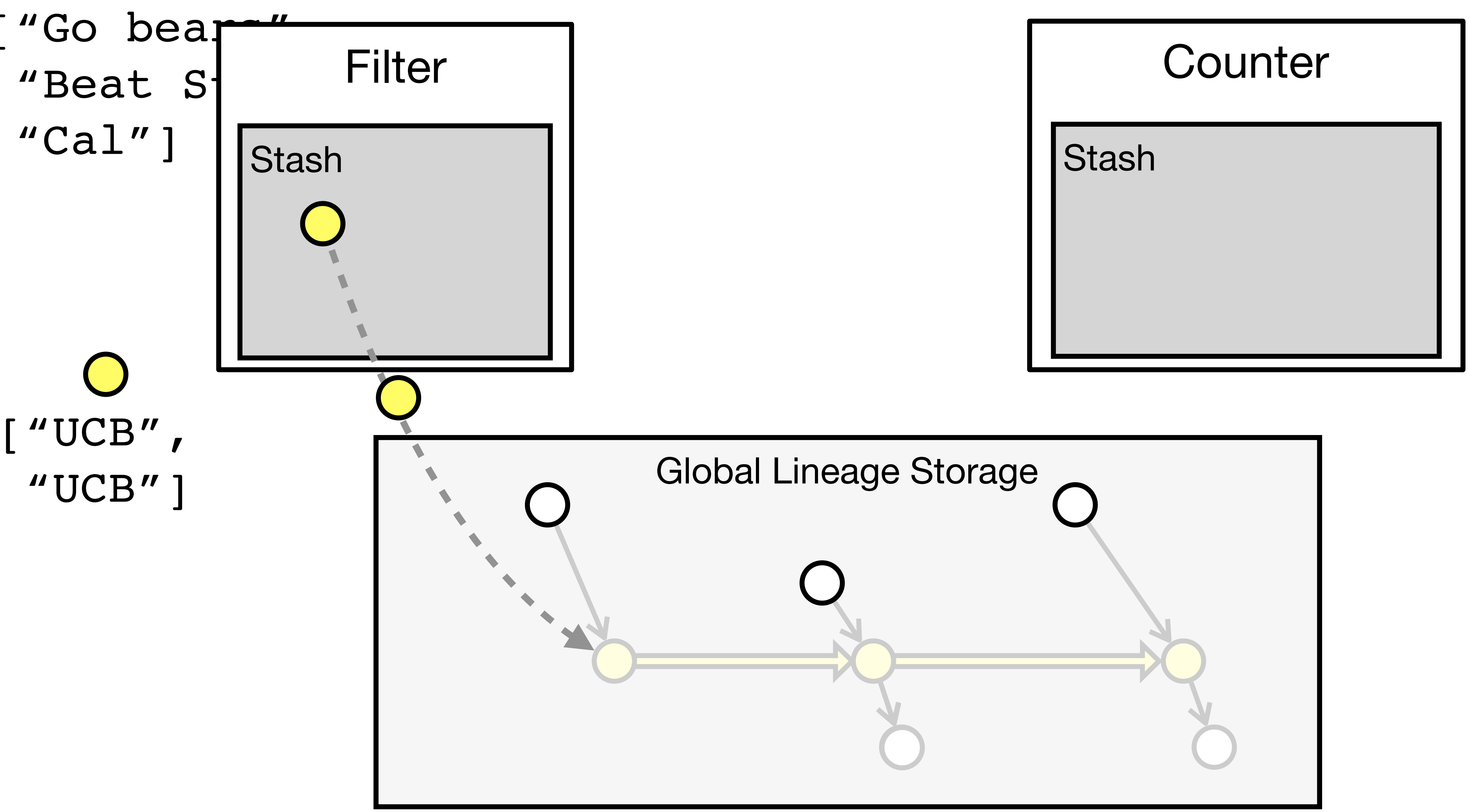
Lineage stash: Logging the **lineage, asynchronously**



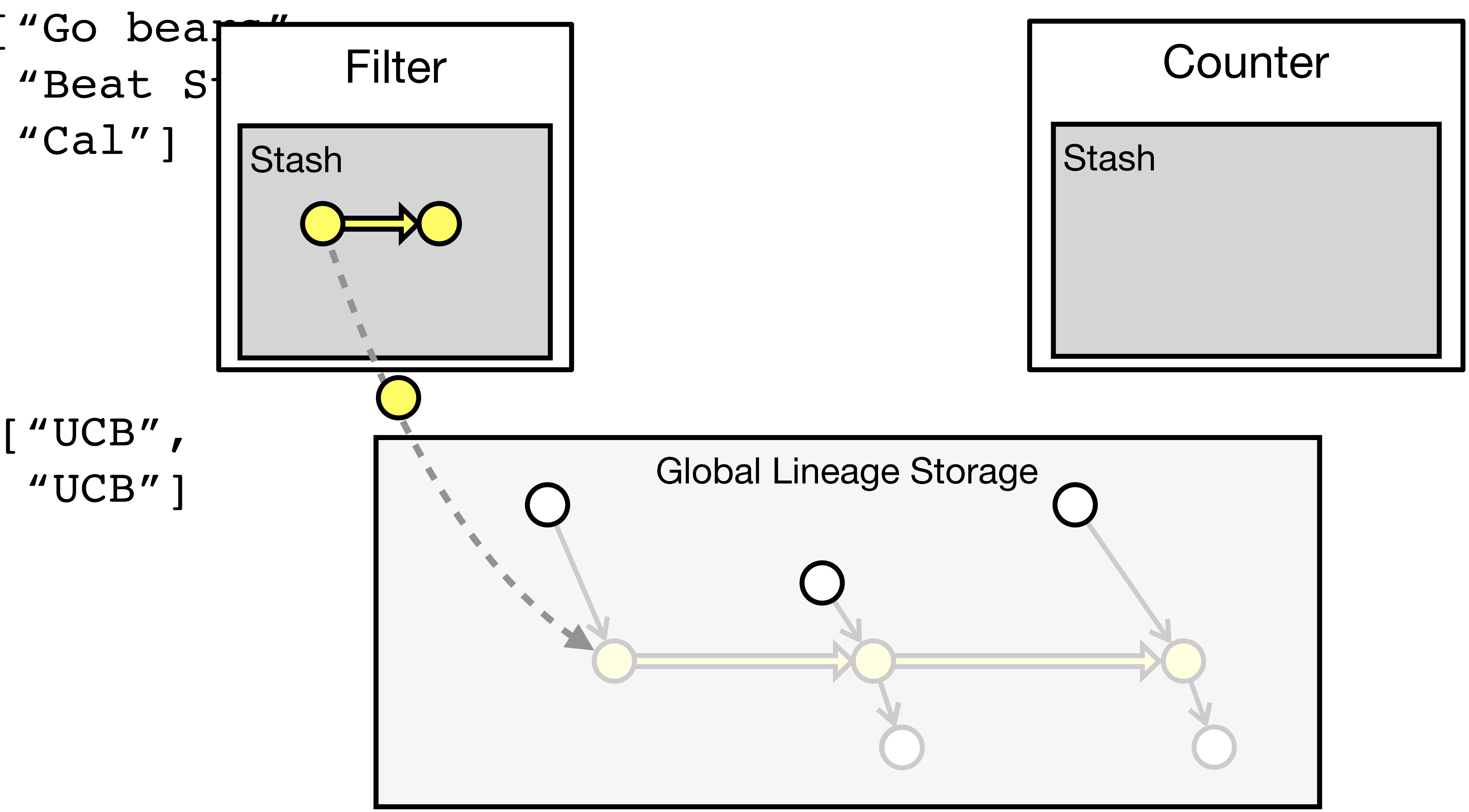
Lineage stash: Logging the **lineage, asynchronously**



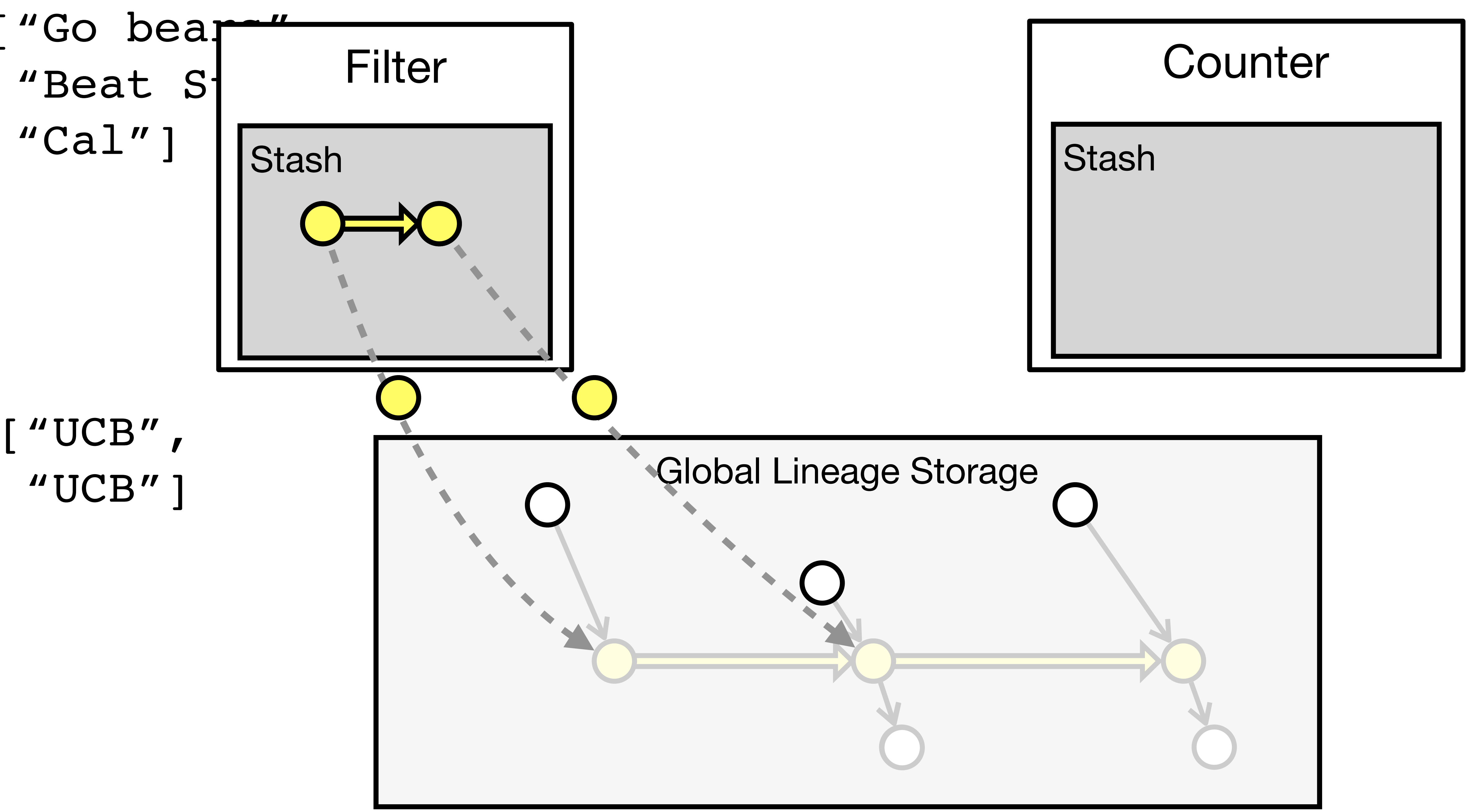
Lineage stash: Logging the **lineage, asynchronously**



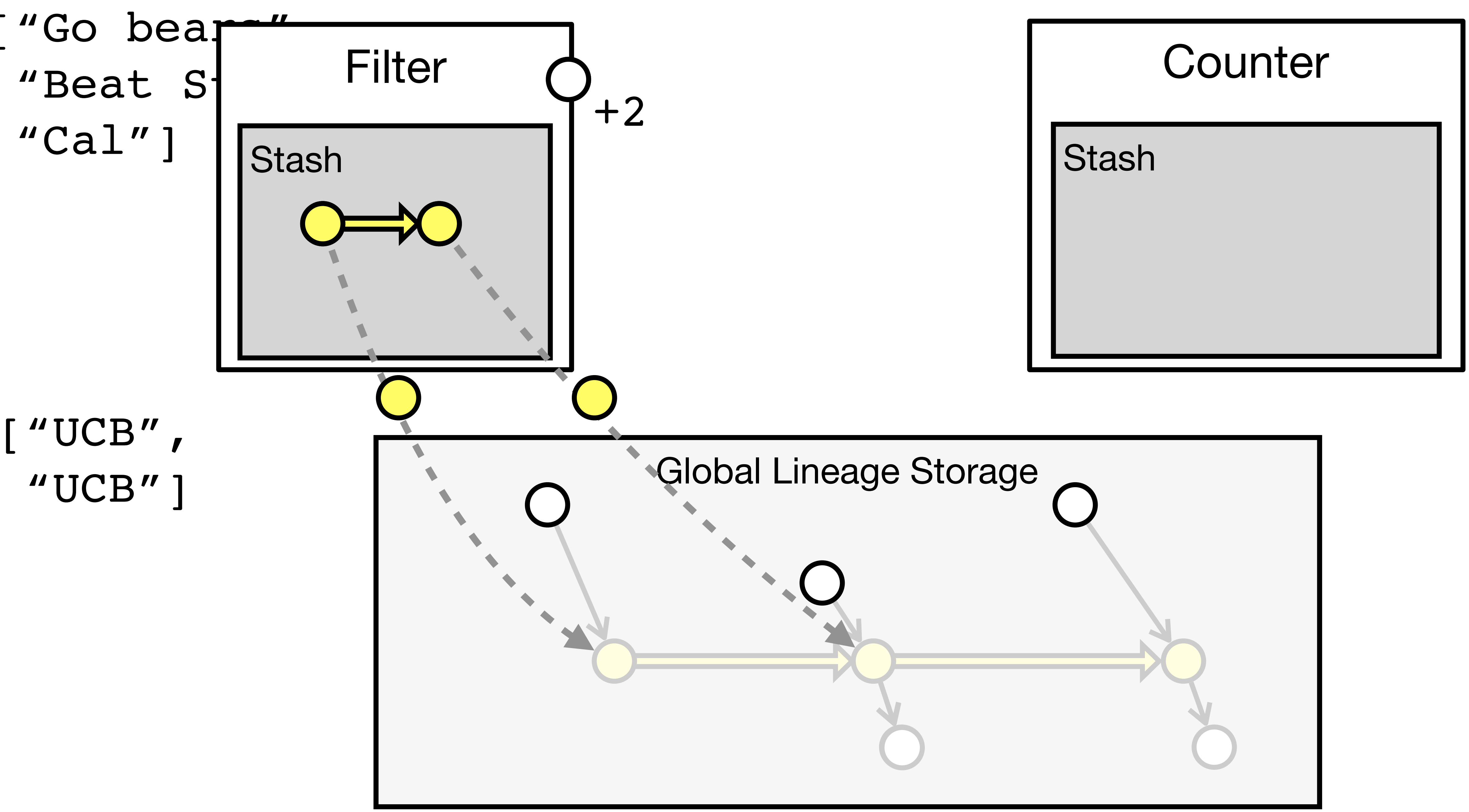
Lineage stash: Logging the **lineage, asynchronously**



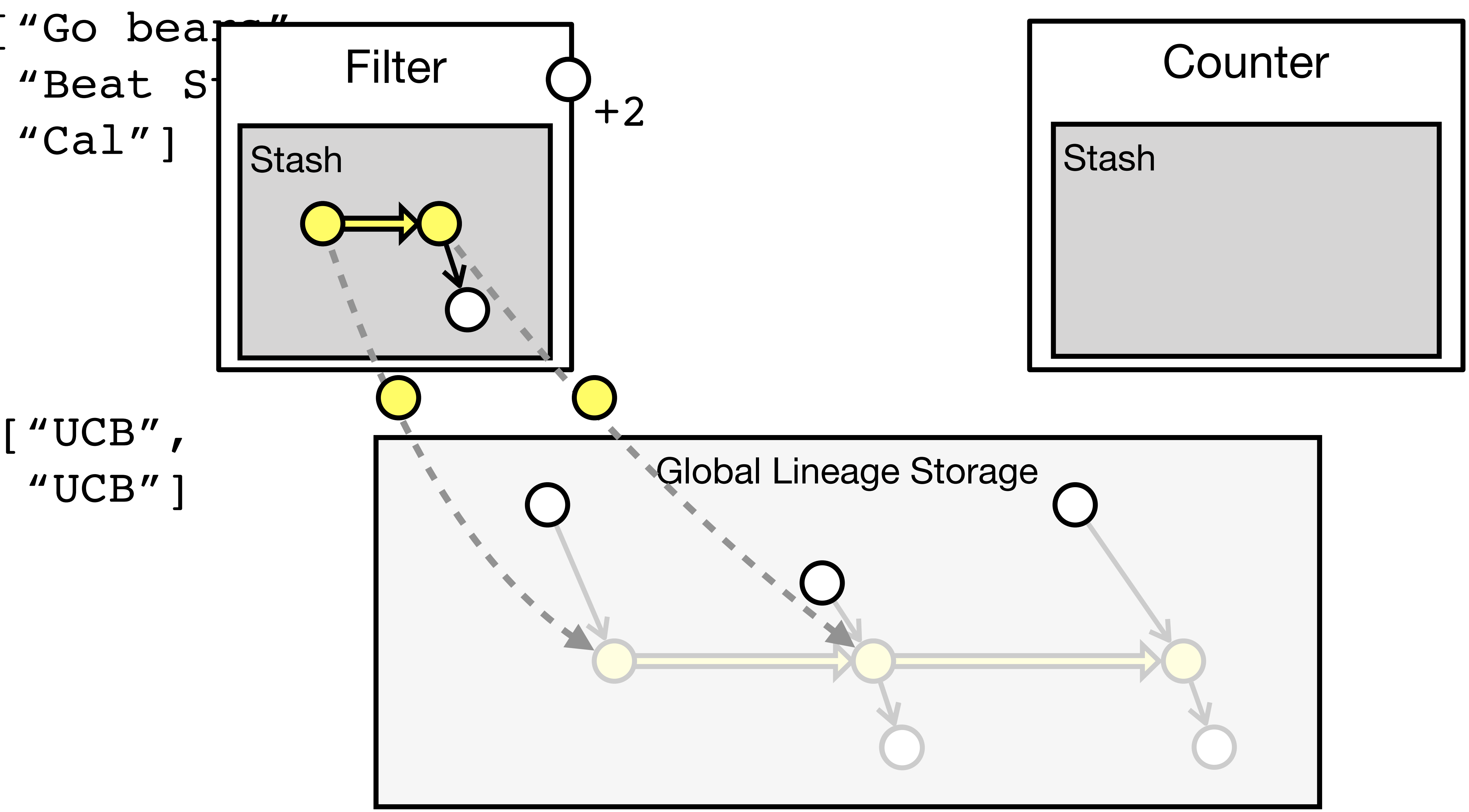
Lineage stash: Logging the **lineage, asynchronously**



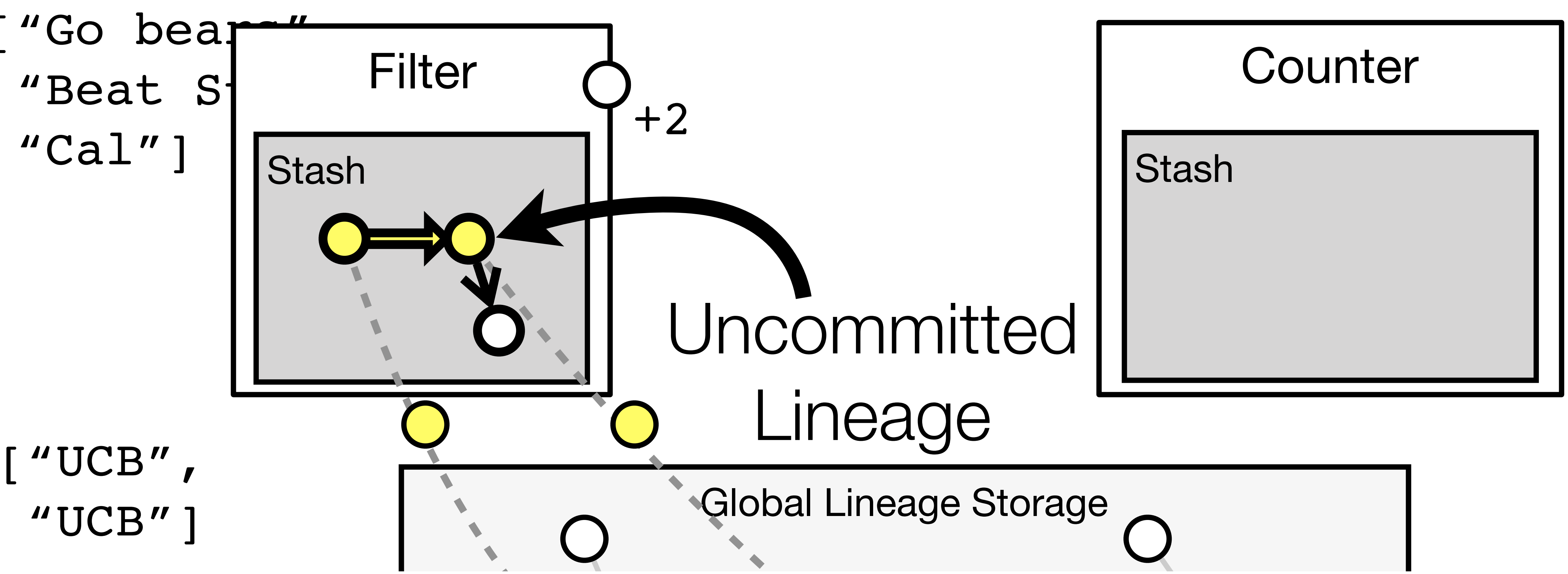
Lineage stash: Logging the **lineage, asynchronously**



Lineage stash: Logging the **lineage, asynchronously**

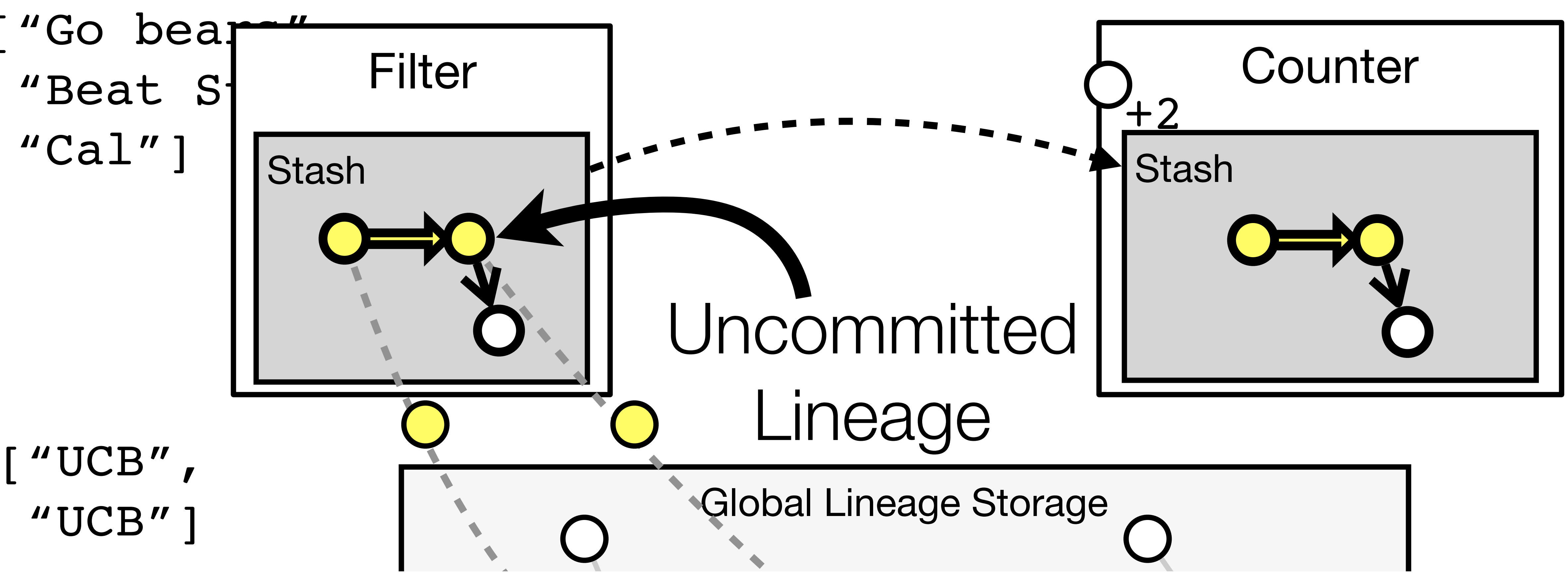


Lineage stash: Logging the **lineage, asynchronously**



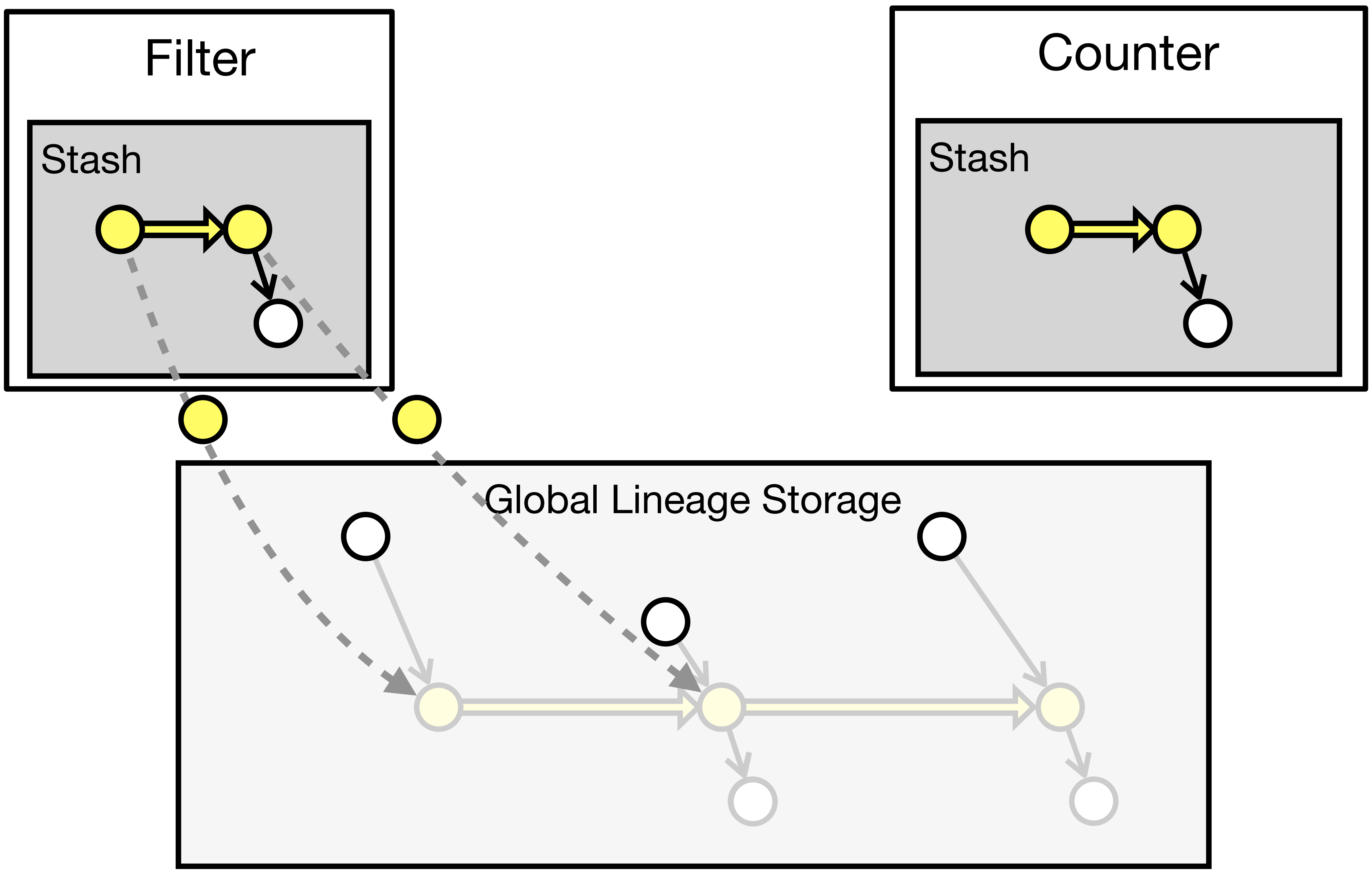
(3) Forward uncommitted lineage.

Lineage stash: Logging the **lineage, asynchronously**

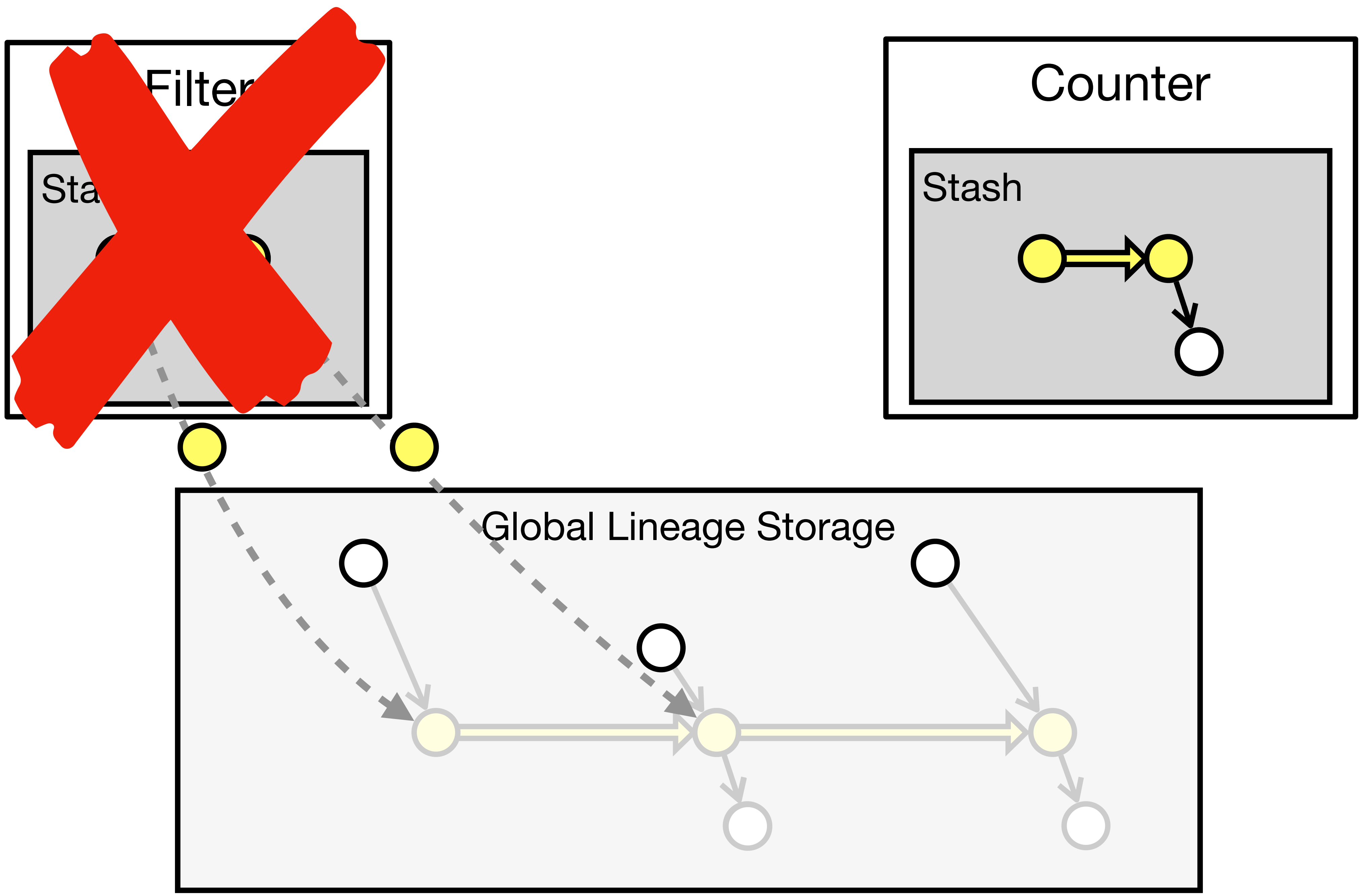


(3) Forward uncommitted lineage.

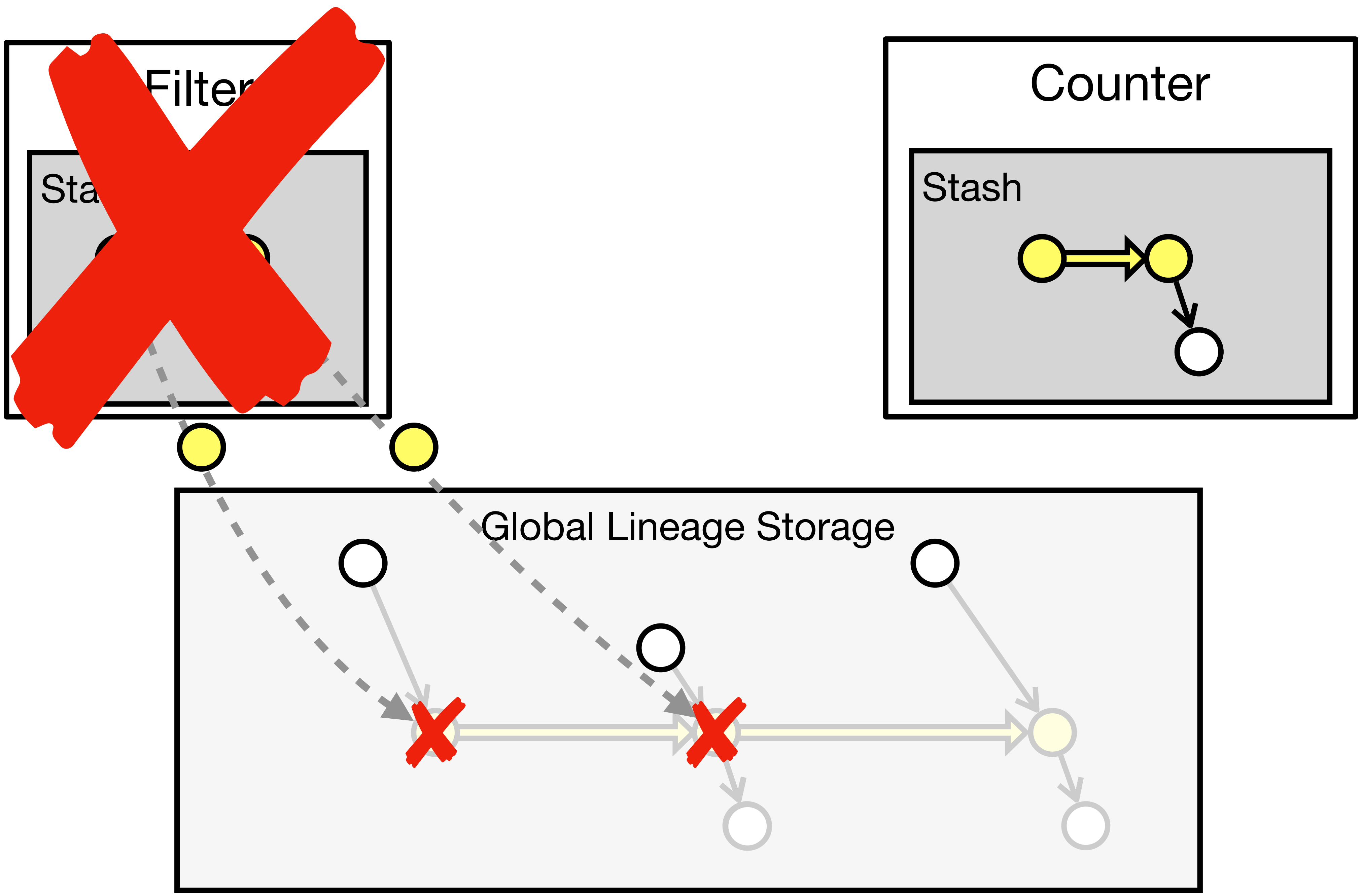
Lineage stash recovery



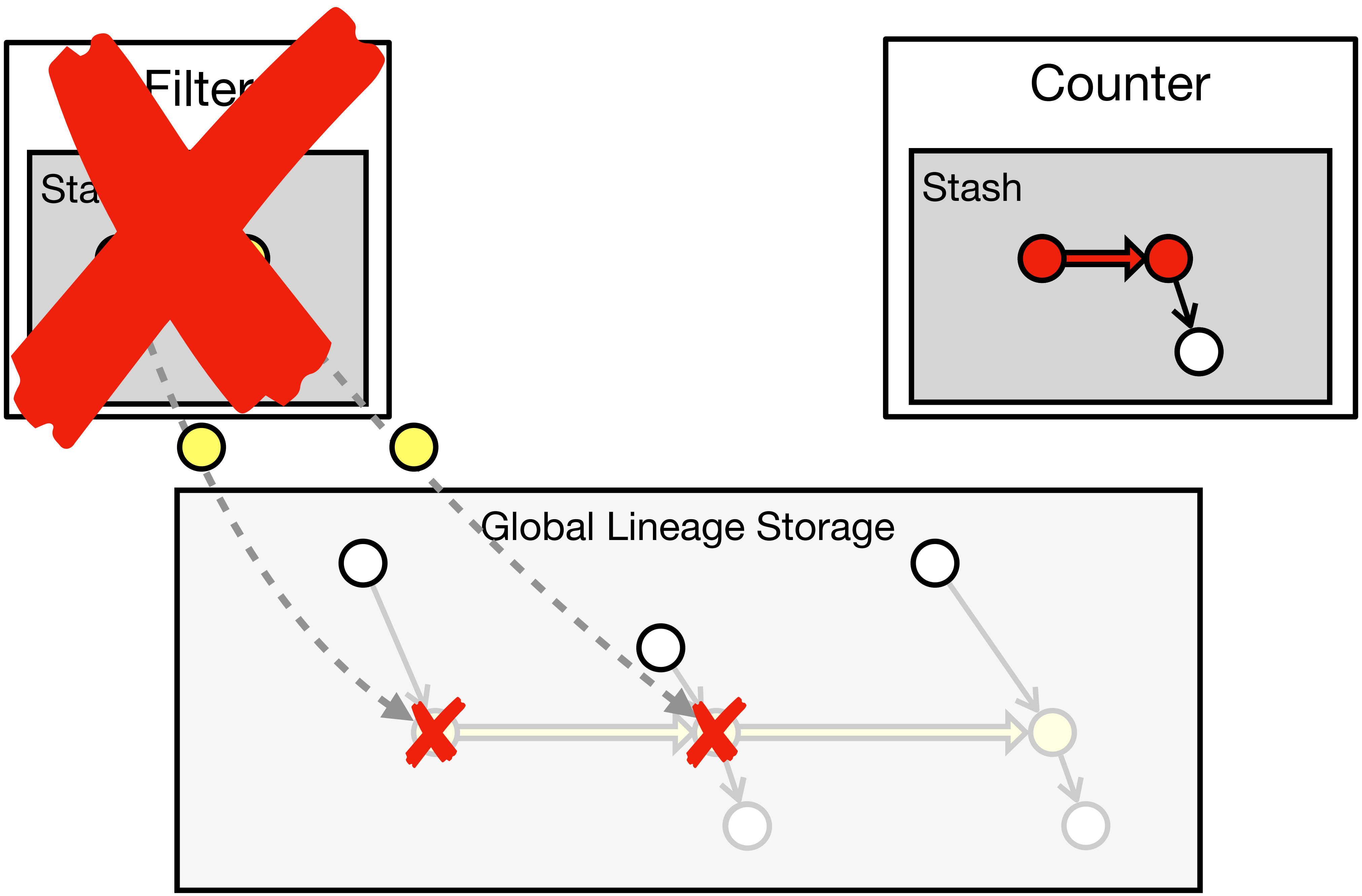
Lineage stash recovery



Lineage stash recovery

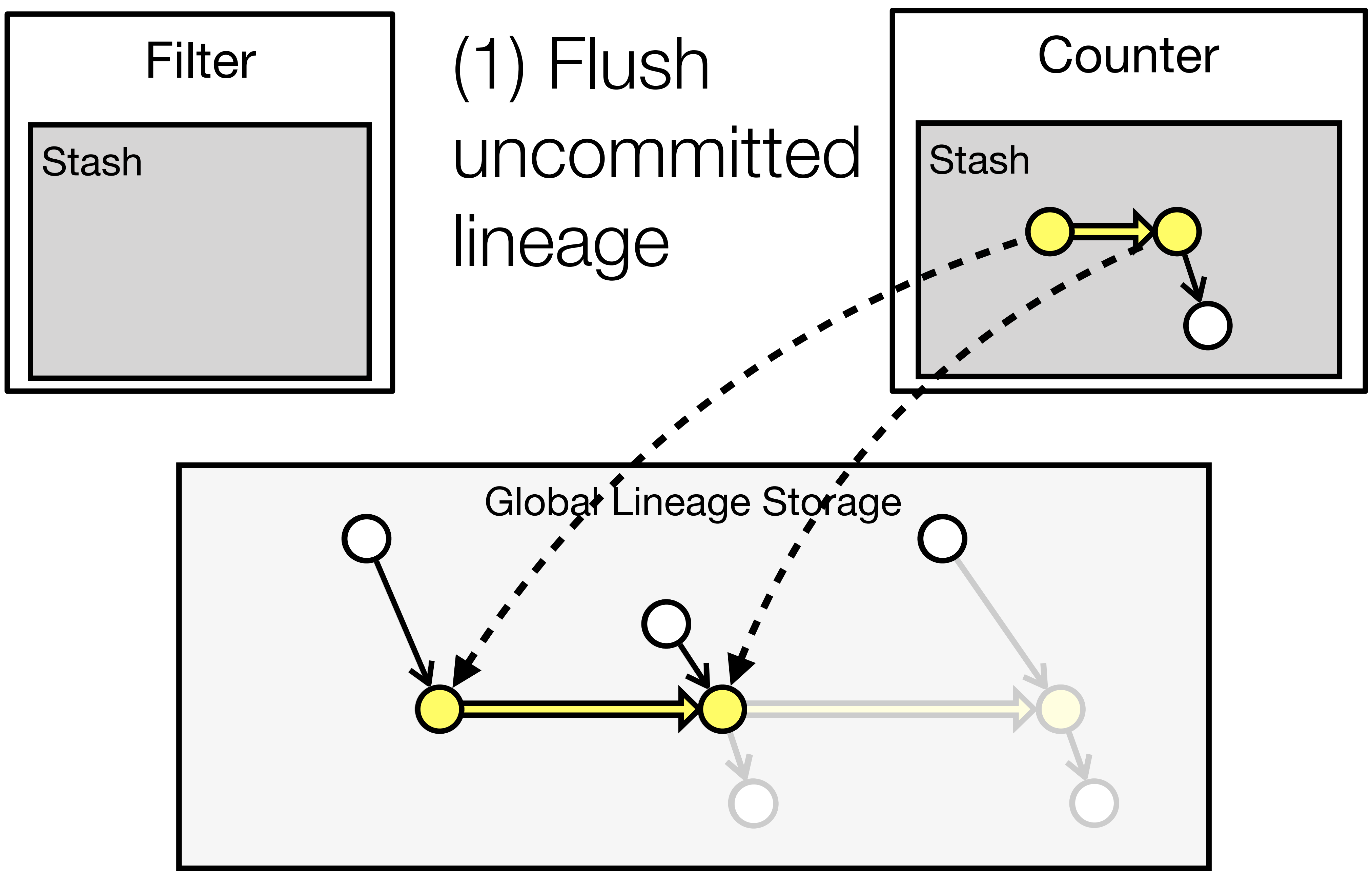


Lineage stash recovery

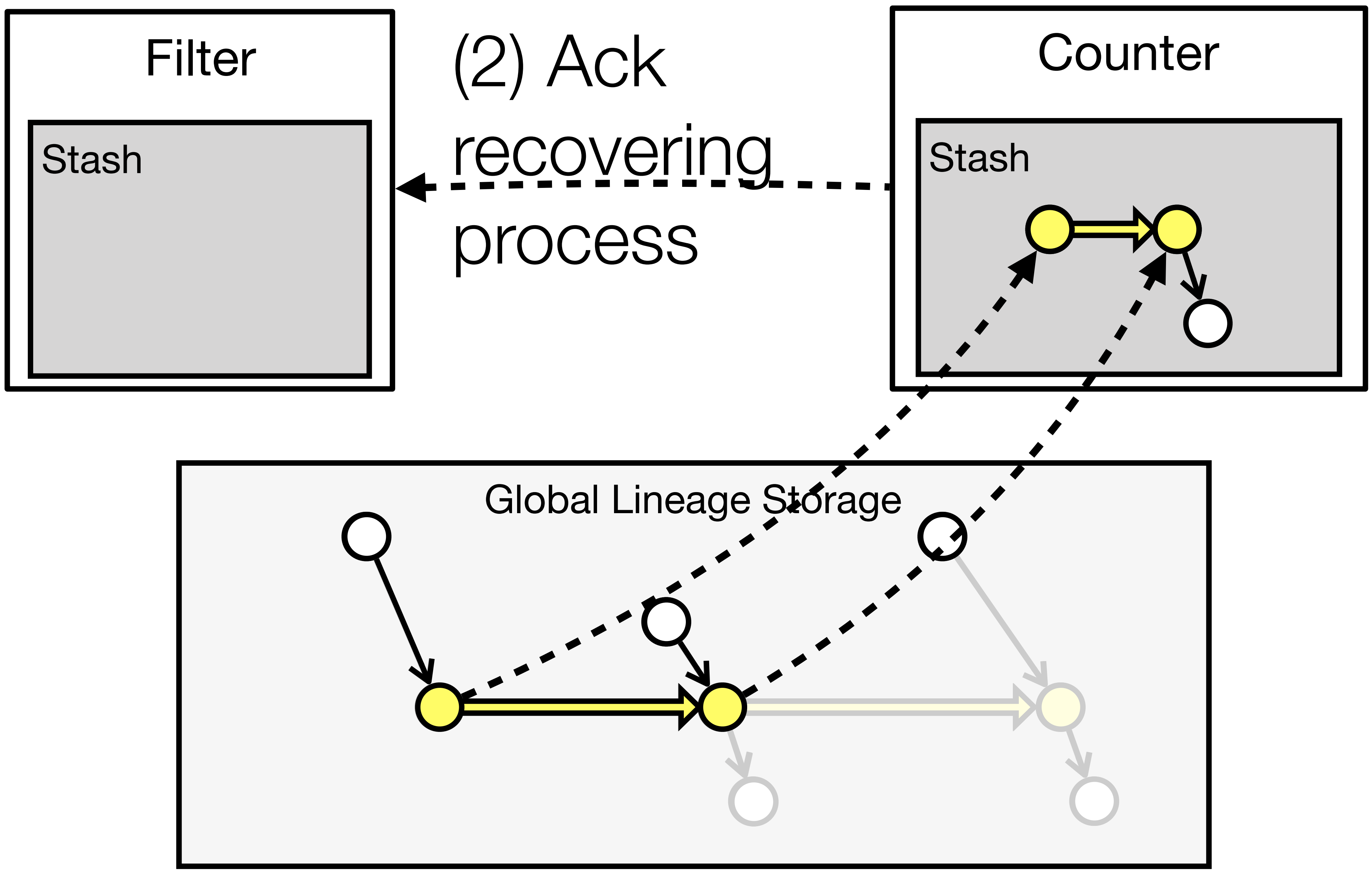


Lineage stash recovery

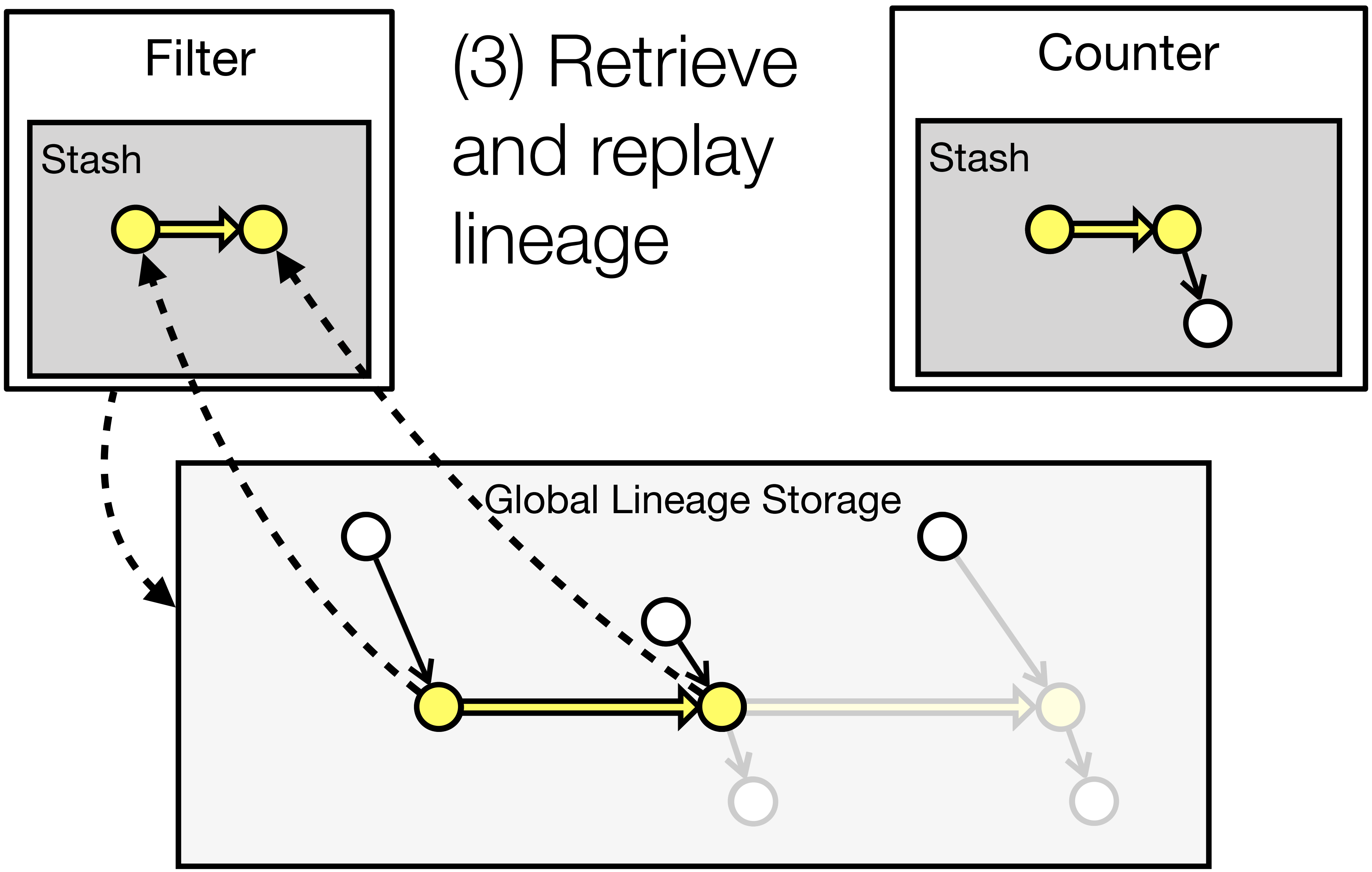
(1) Flush uncommitted lineage



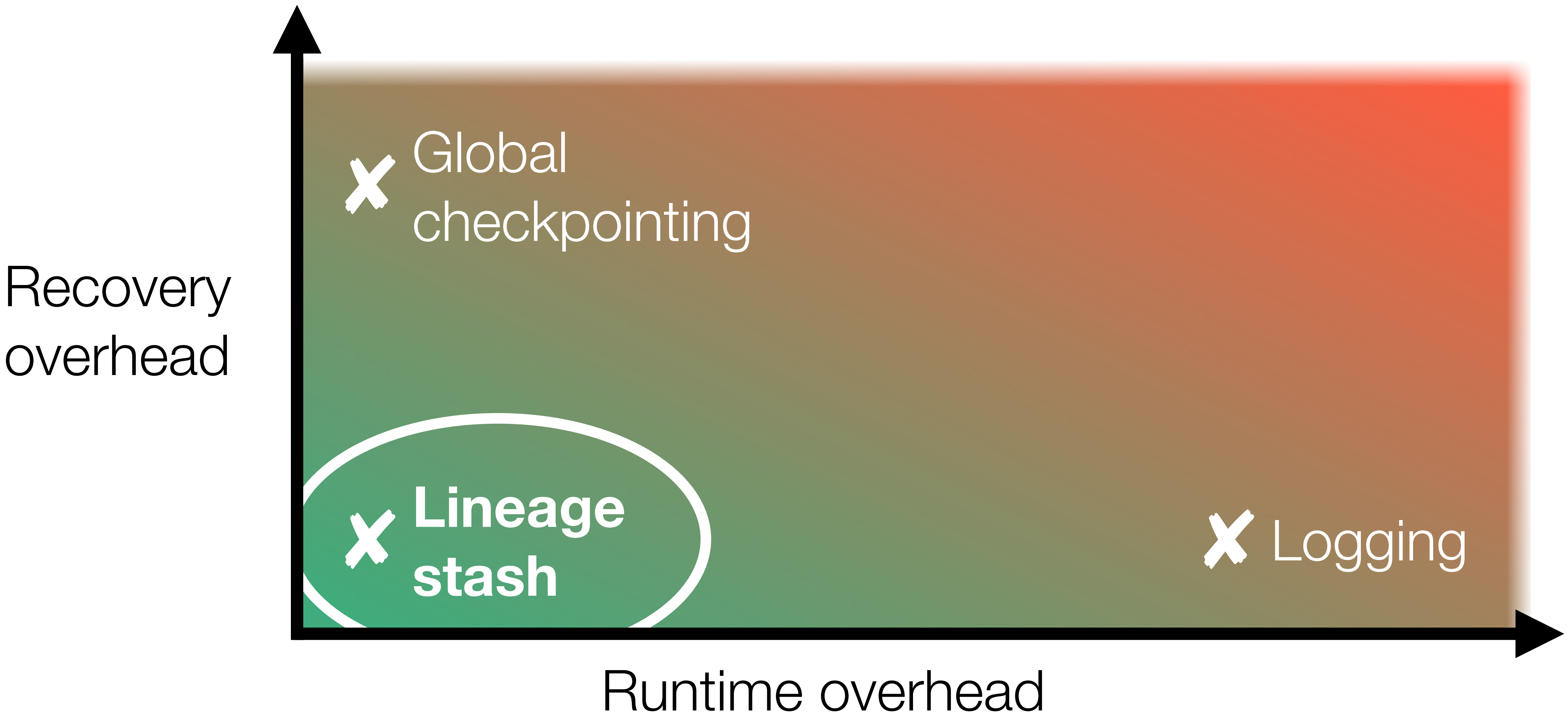
Lineage stash recovery



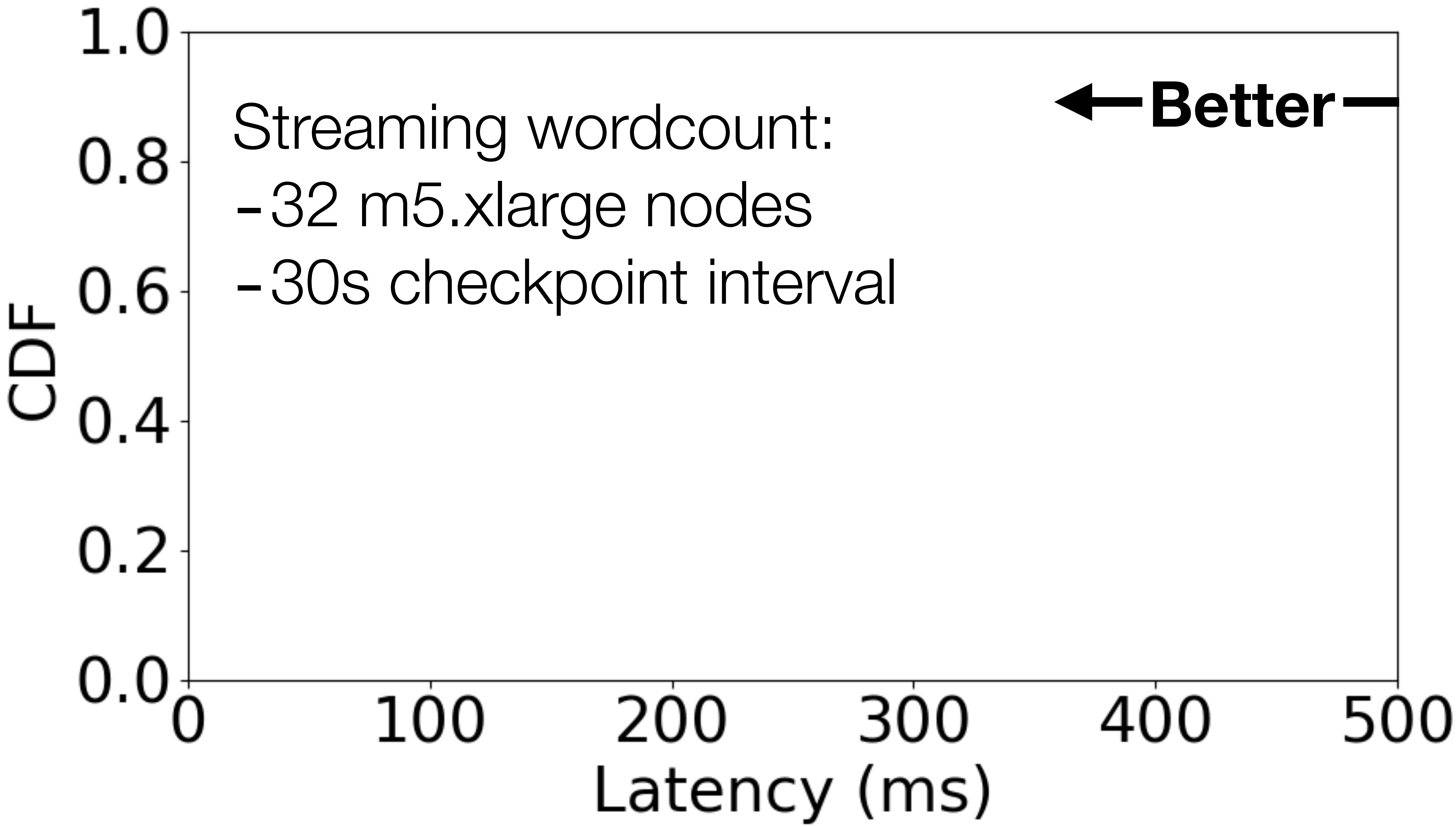
Lineage stash recovery



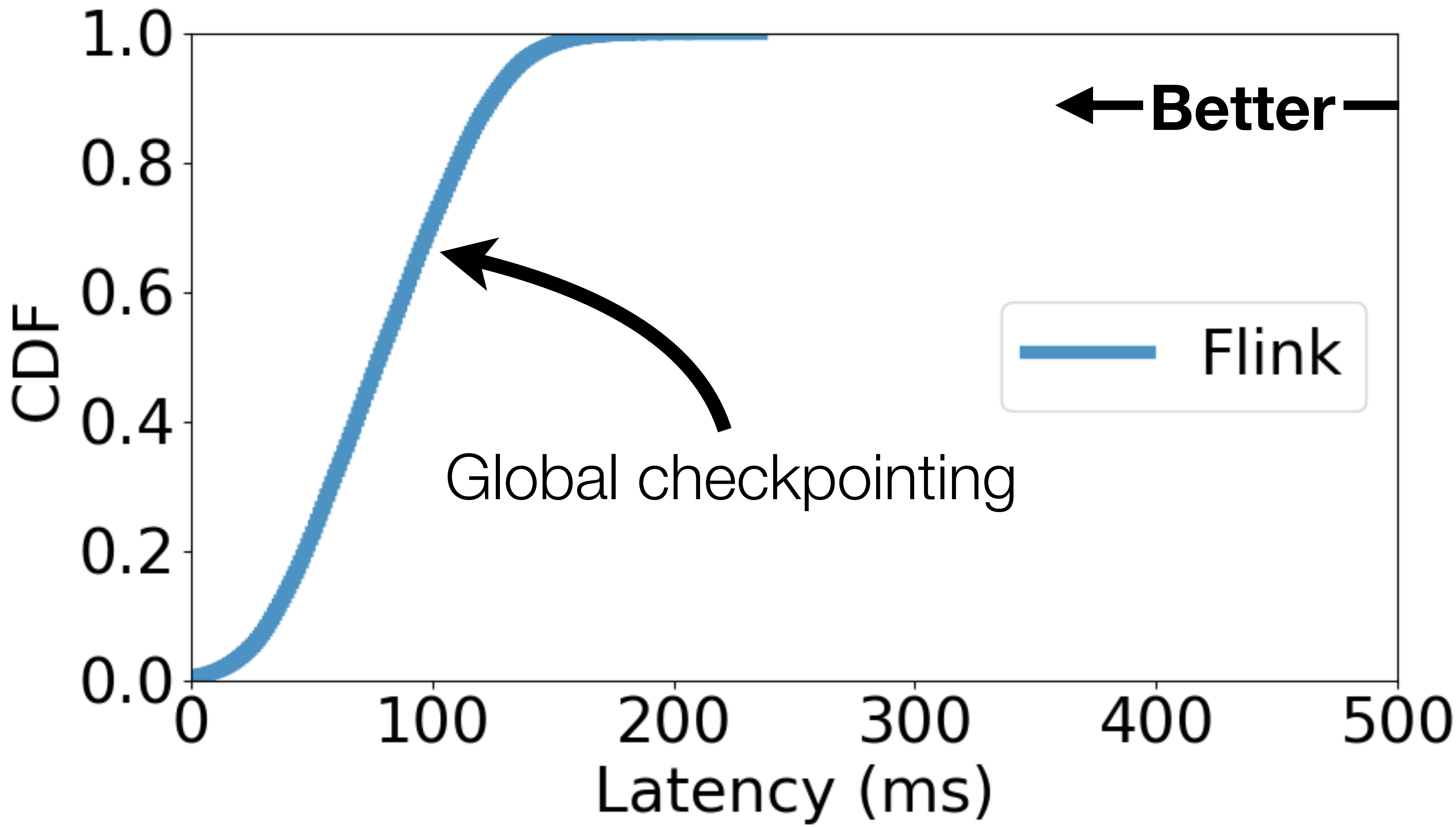
Tradeoff between low latency and recovery time



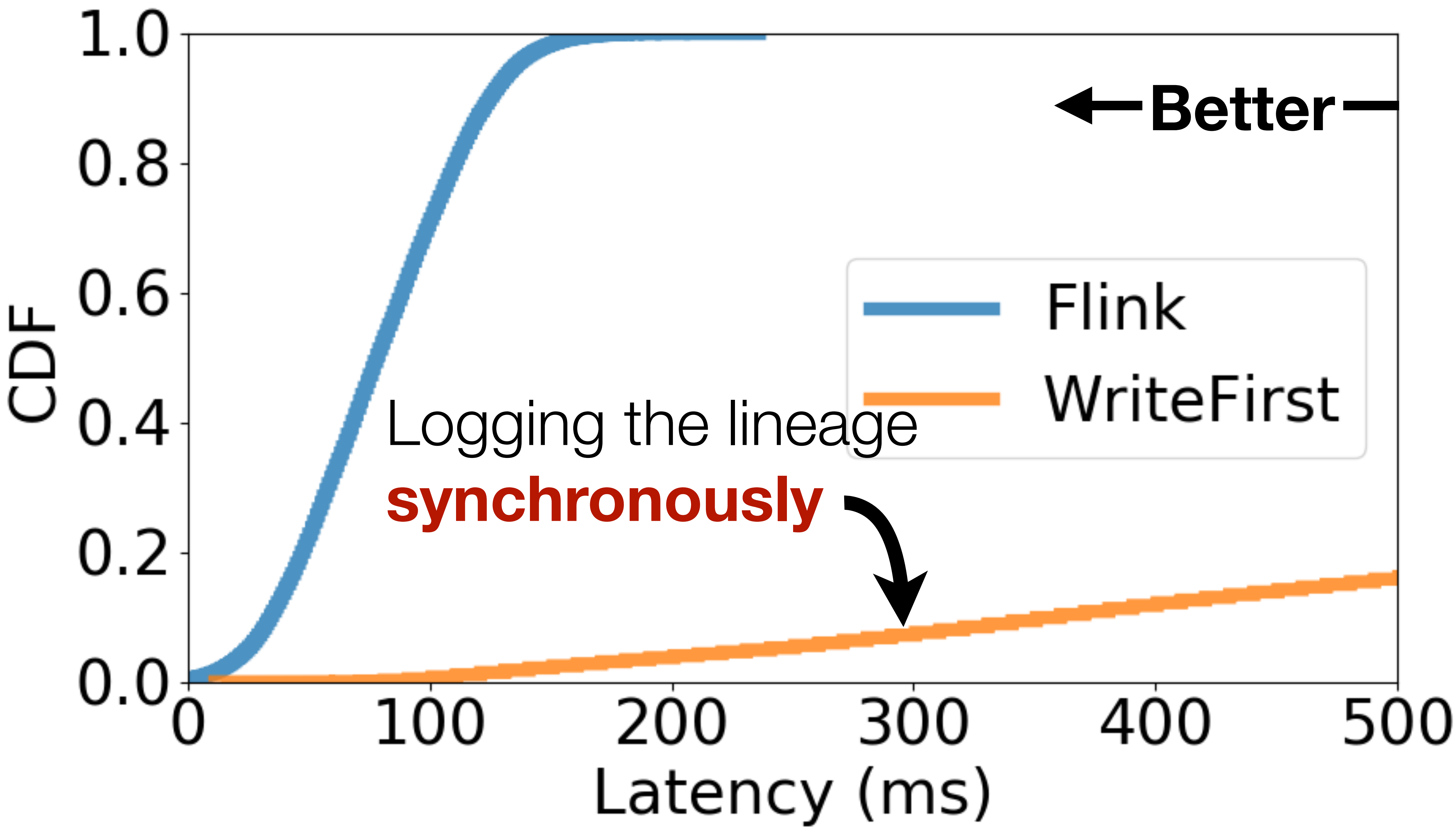
Latency without failures



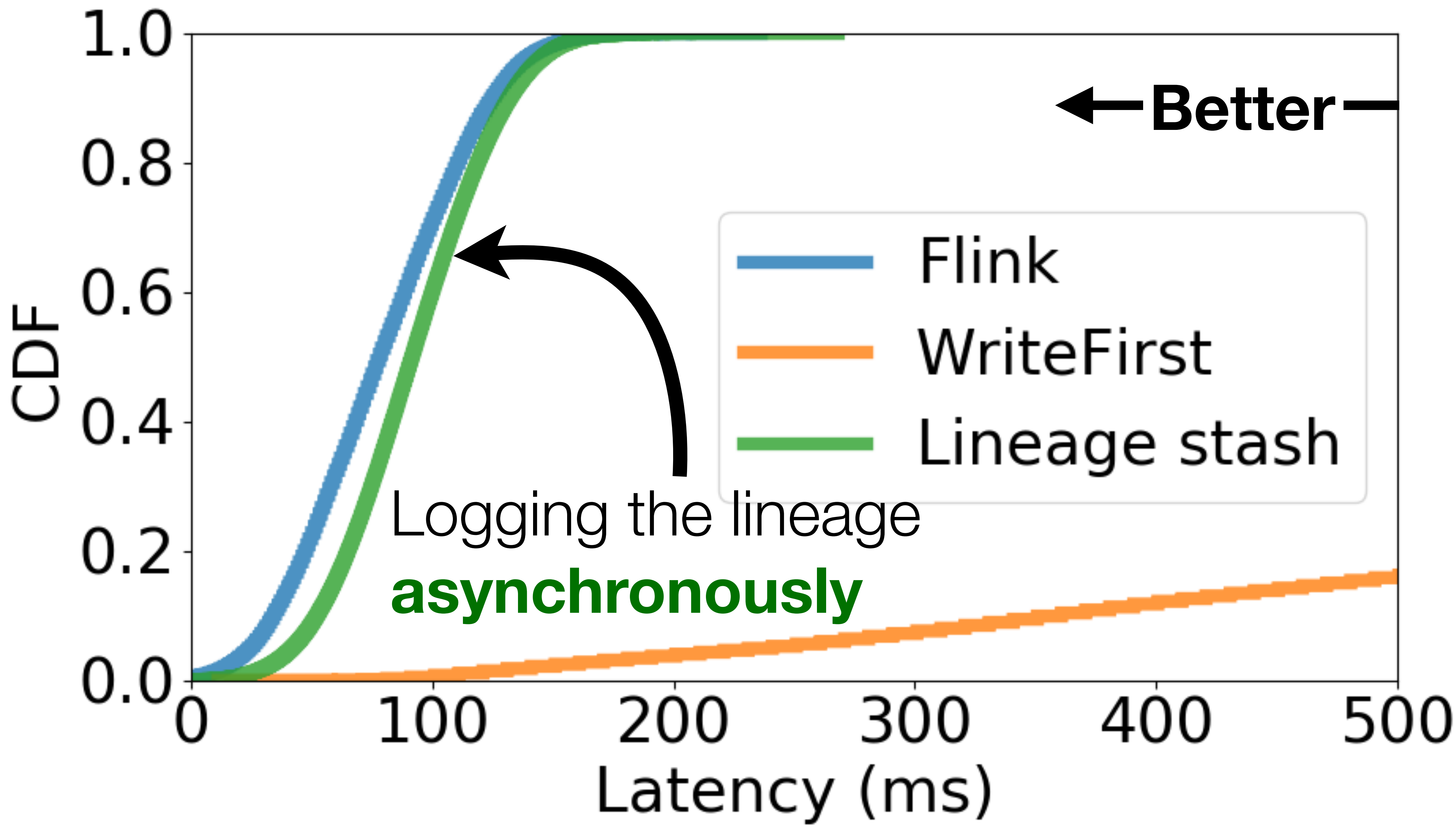
Latency without failures



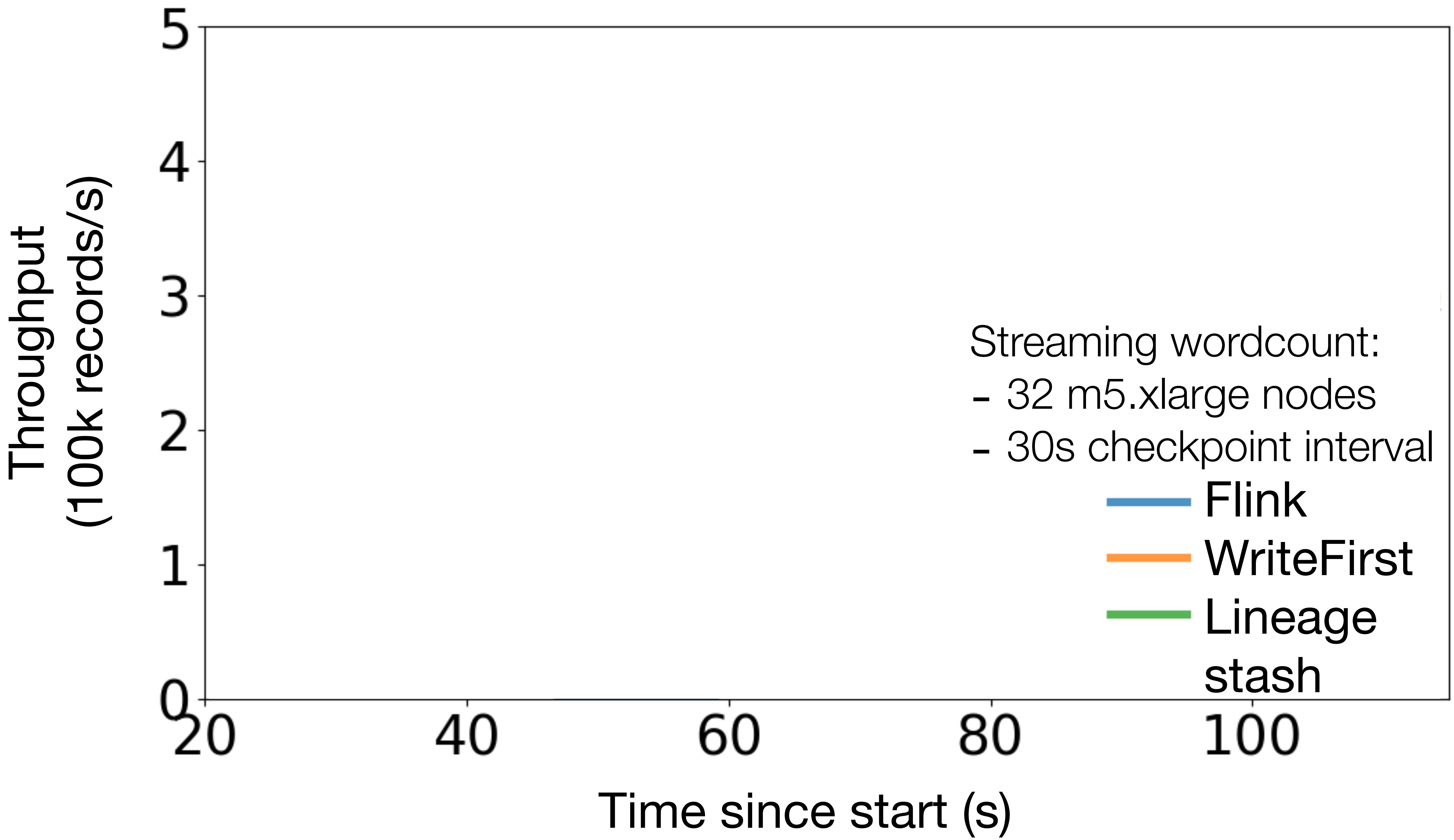
Latency without failures



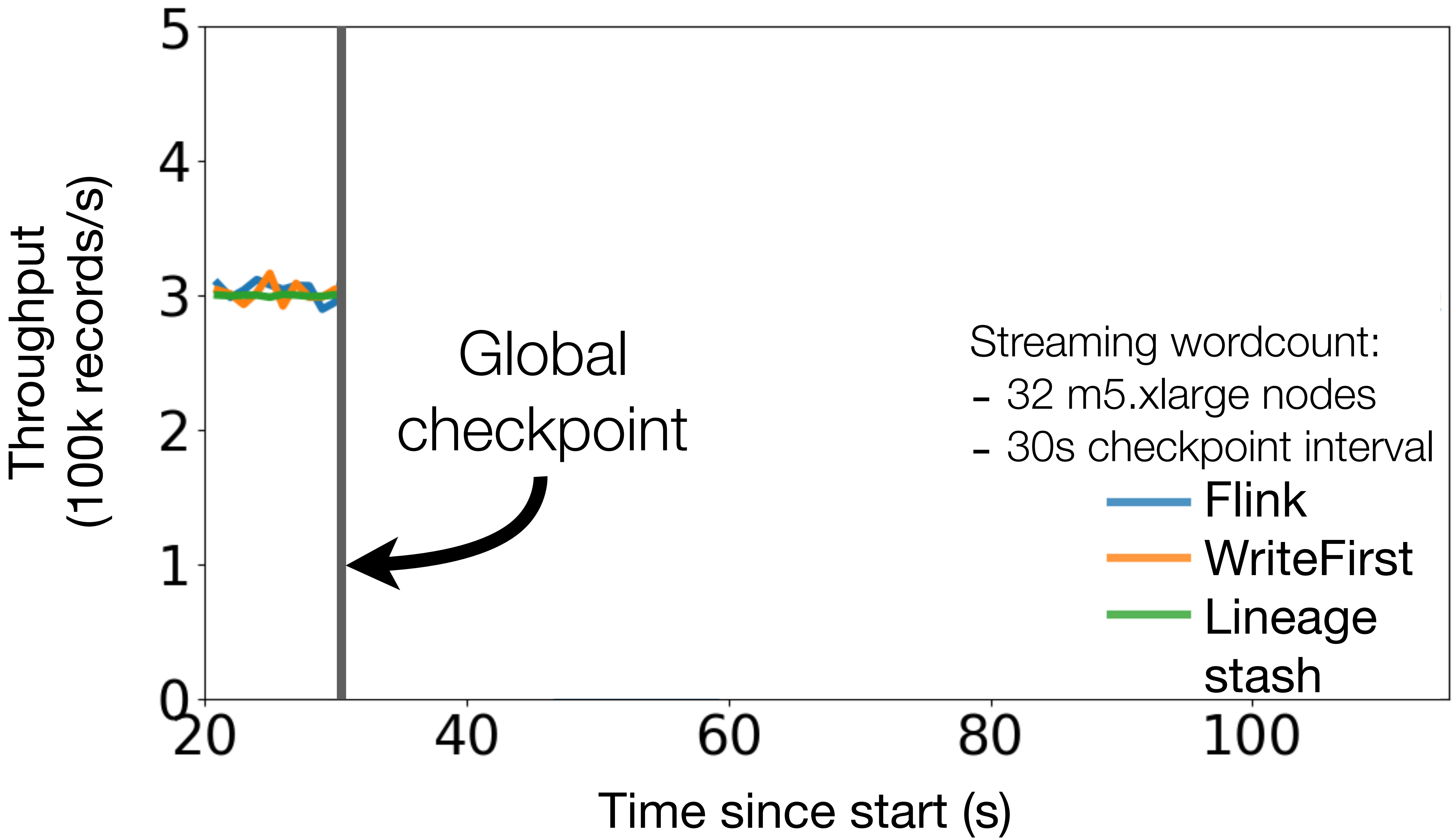
Latency without failures



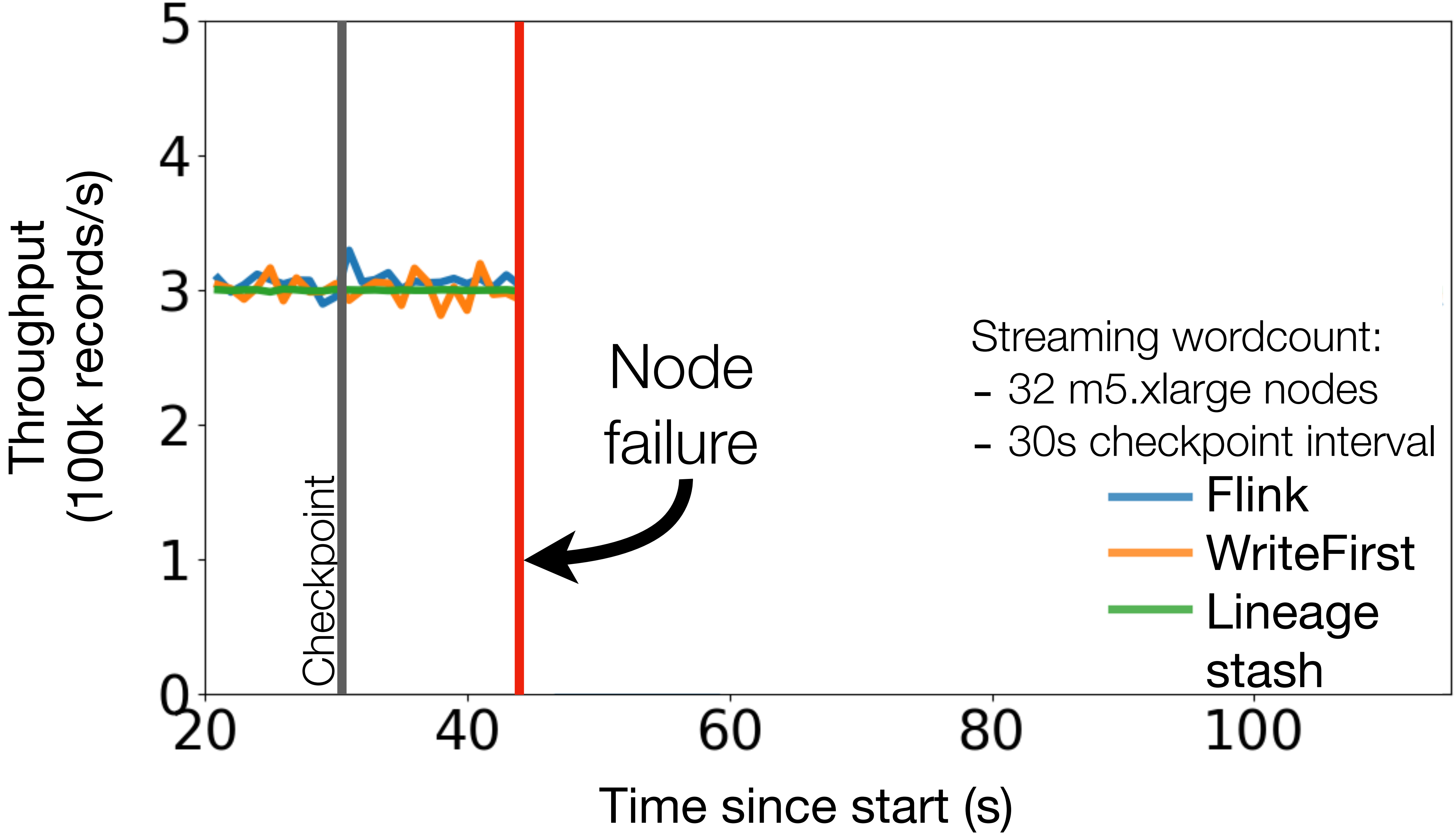
Throughput during recovery



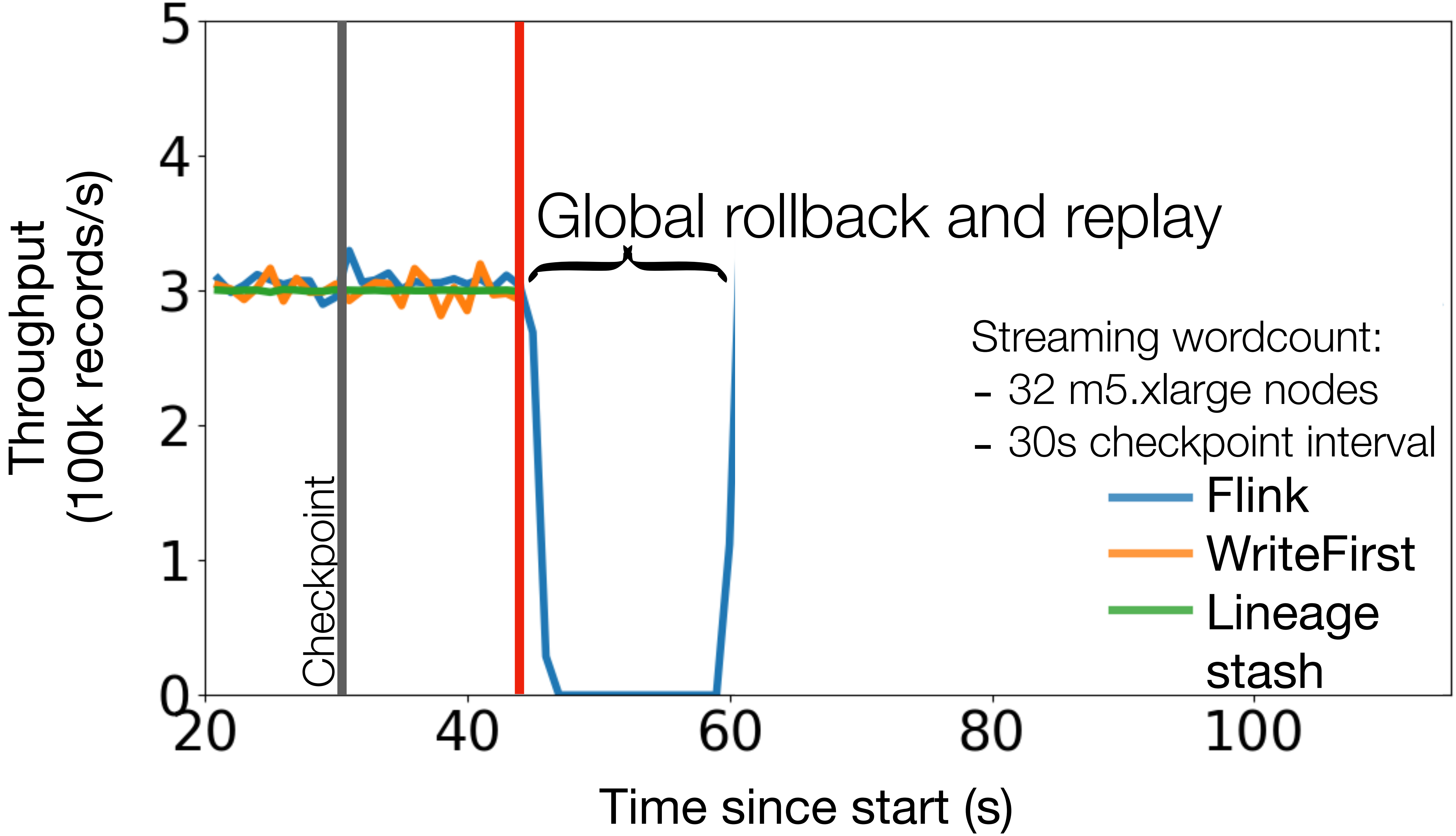
Throughput during recovery



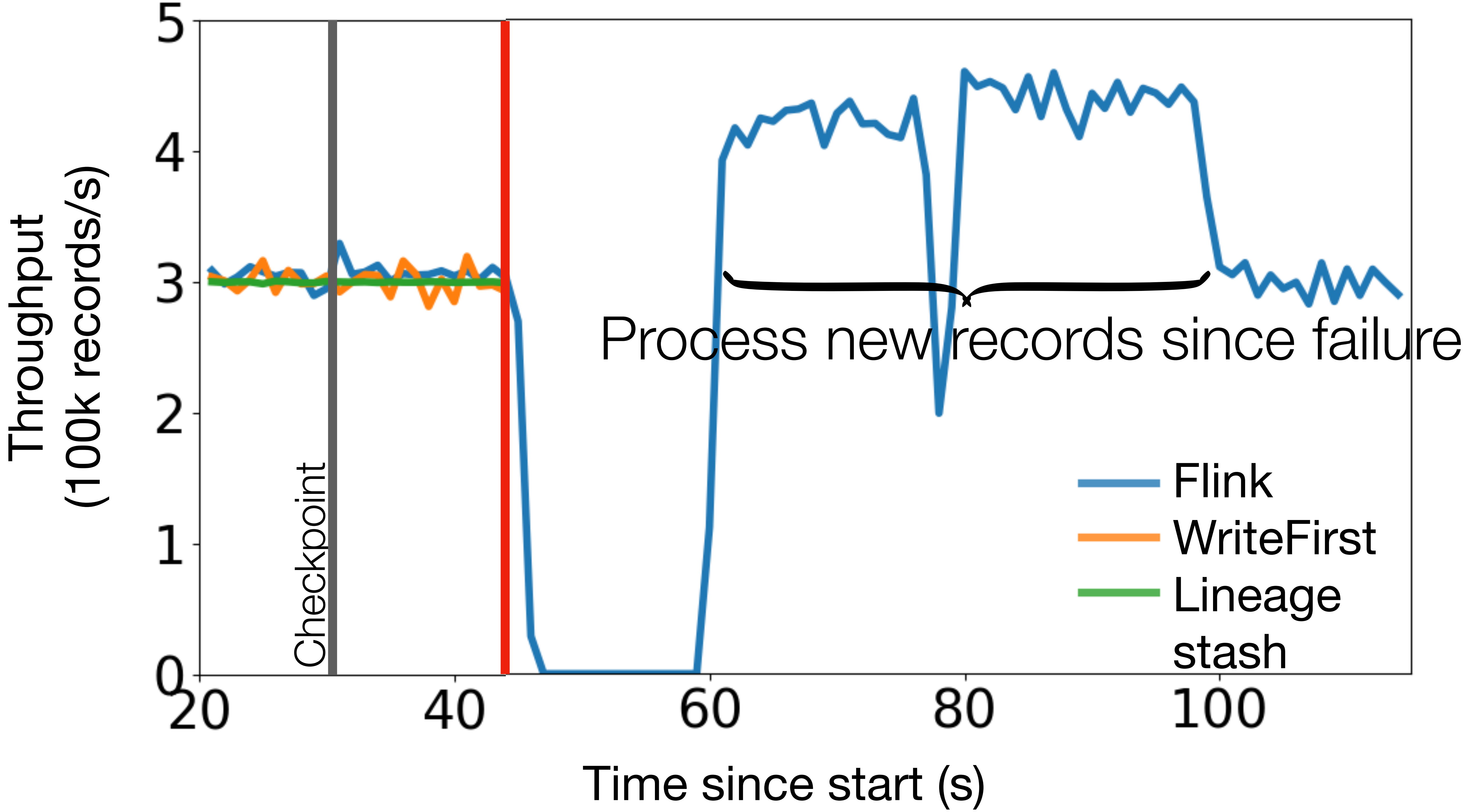
Throughput during recovery



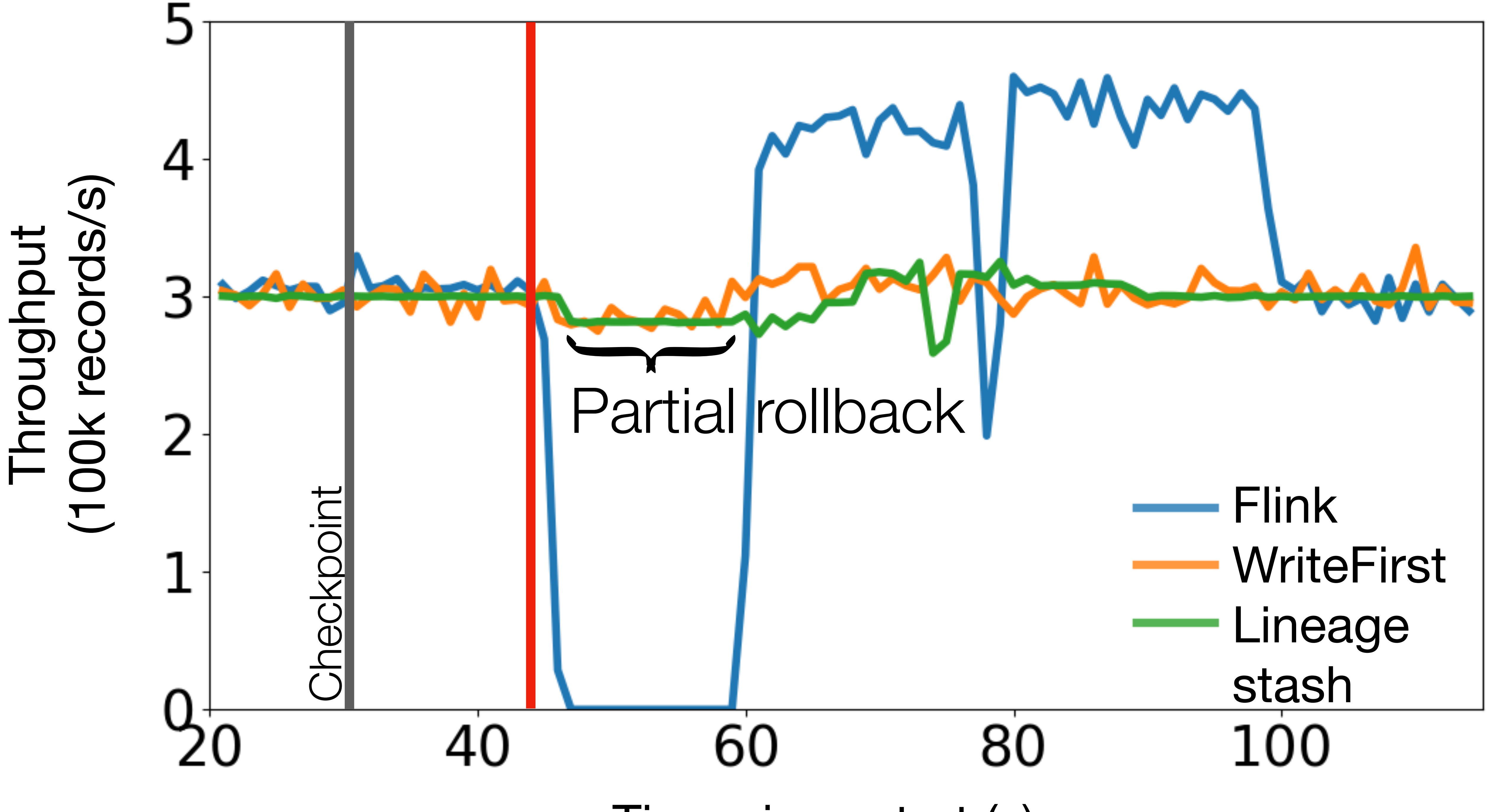
Throughput during recovery



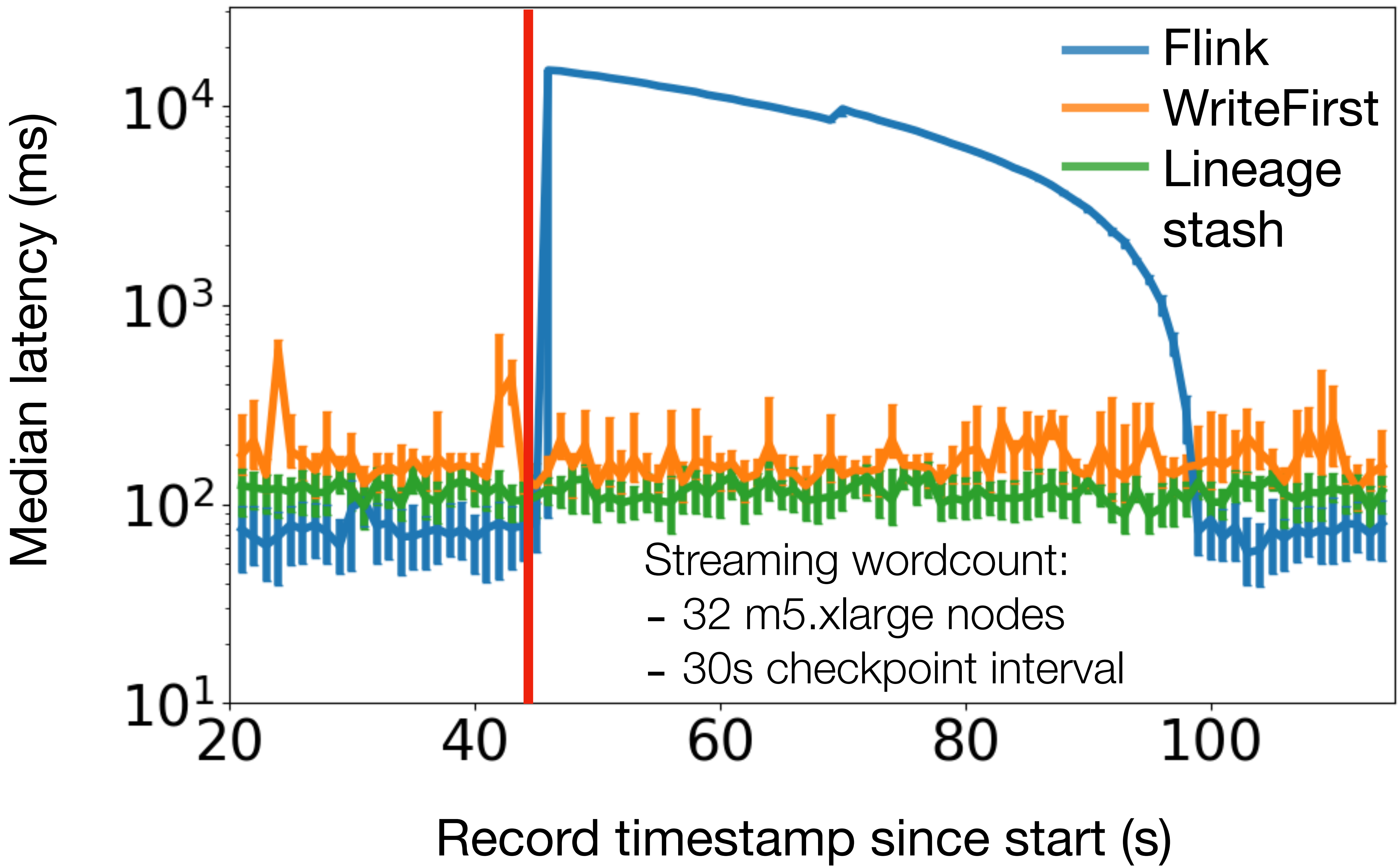
Throughput during recovery



Throughput during recovery



Latency during recovery



Lineage Stash

See the paper (or email me: swang@berkeley.edu) for:

- Discussion and evaluation of other applications
- Nondeterminism in data processing applications
- Protocols for flushing and recovering the stash

Key idea: **Asynchronously** log the **lineage** and forward **uncommitted** lineage to guarantee recovery correctness.

Low latency during execution and low downtime after a failure for large-scale decentralized data processing applications.