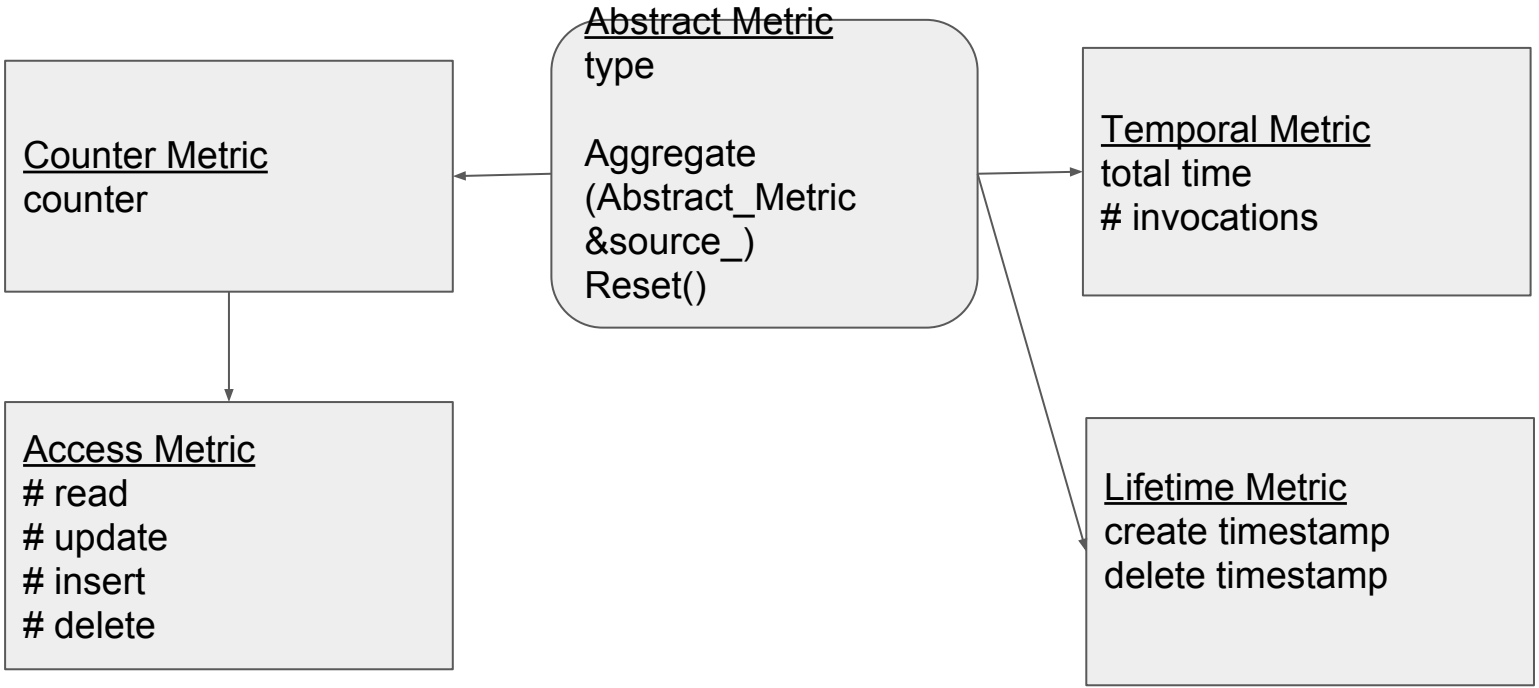
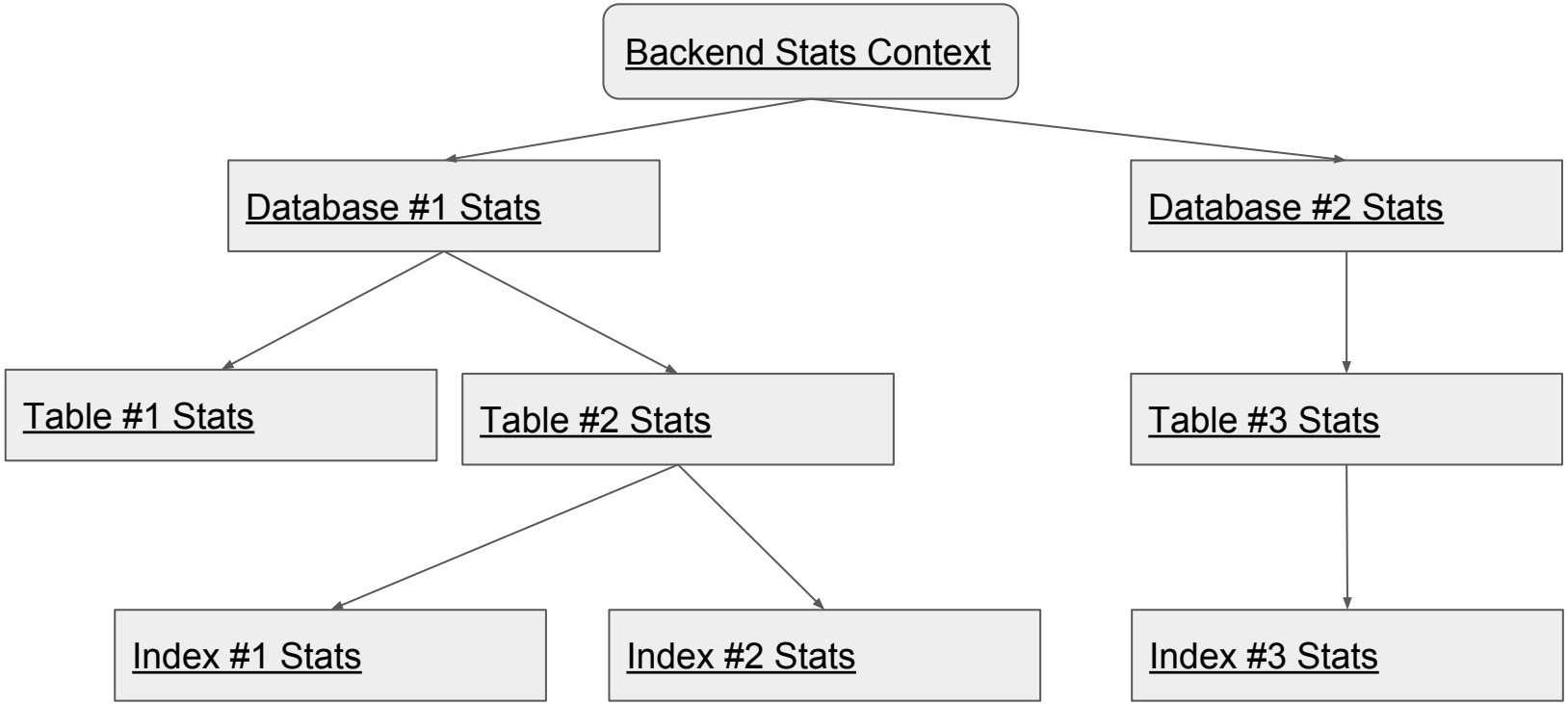
Project #3 Final: Statistics Collection

Aaron Harlap, Lin Ma, Dana Van Aken

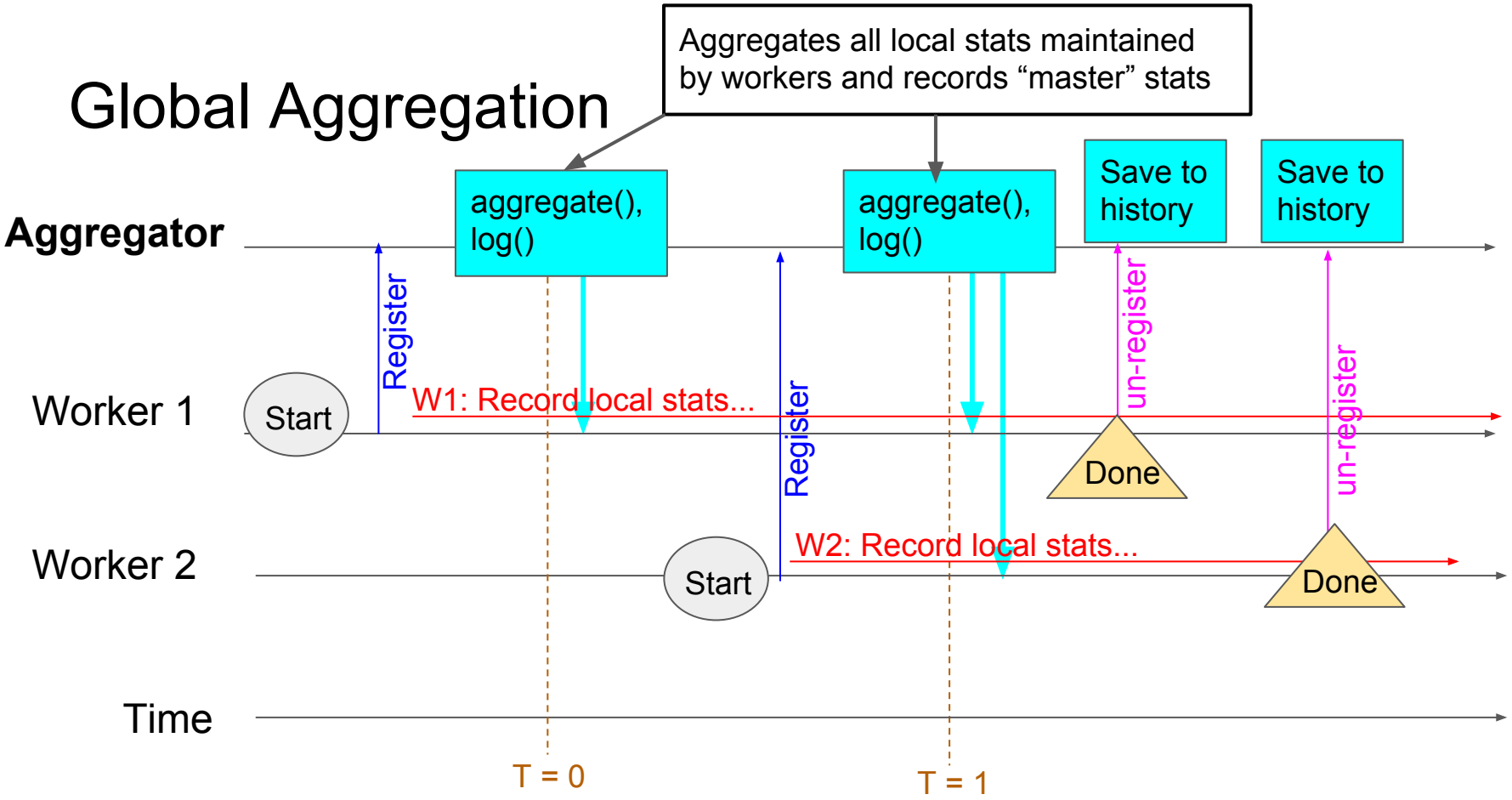
Metrics


Abstract Metric
type
Aggregate
(Abstract_Metric
&source_)
Reset()
Counter Metric
counter
Access Metric
read
update
insert
delete
Temporal Metric
total time
invocations
Lifetime Metric
create timestamp
delete timestamp

Backend Stats Context (per thread)



Global Aggregation



Demo

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
At interval: 147
//===--------------------------------------------------------------------===//
// DATABASE_ID 0
//===--------------------------------------------------------------------===//
# transactions committed: 1
# transactions aborted:   0


//===--------------------------------------------------------------------===//
// DATABASE_ID 12145
//===--------------------------------------------------------------------===//
# transactions committed: 9
# transactions aborted:   0
----------------------------
  TABLE FOO
----------------------------
[ reads=7, updates=7, inserts=7, deletes=0 ]


Average throughput:     0.068027 txn/s
Moving avg. throughput: 0.000000 txn/s
Current thoughput:      0.000000 txn/s
```

Developed an extensible stats collection module

- Identified “primitive” (counters, temporal, lifetime) & “composite” stat types (access metrics, profiling)
- Hierarchical stat types (global, database, table, index)
- Threads maintain their own local stats
  - Background thread to aggregate per-thread stats
  - Easy to support per-session stats
- Access methods:
  - Stats are currently logged to a file
  - Will be added to catalog once PG code is removed
- Configurable options:
  - Reset, enable/disable stats
  - Control aggregation/logging frequency

Testing

- Thorough unit-testing
- Start stats aggregator
- Execute actions
- Compare and Contrast!