Lecture #NN

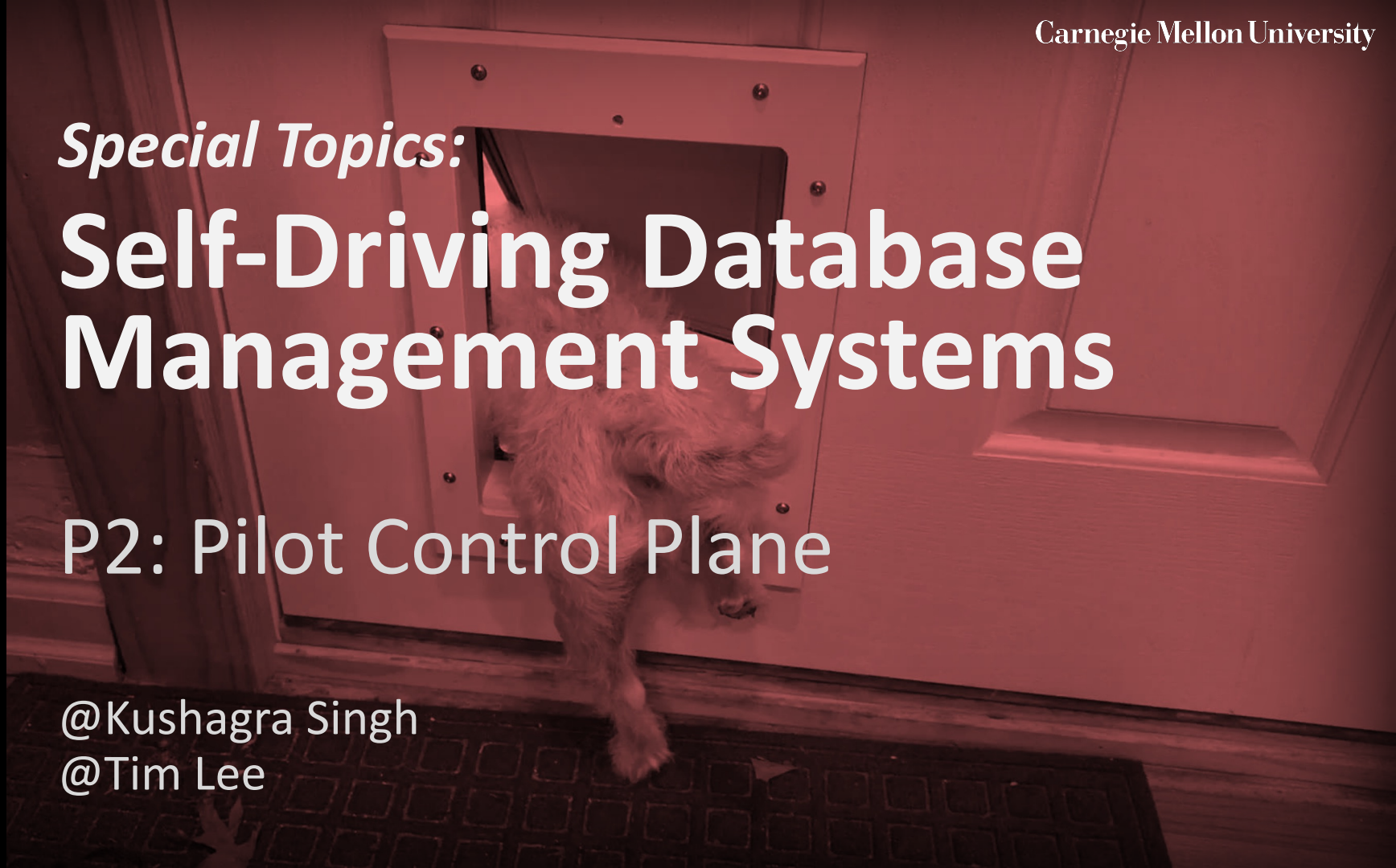
Carnegie Mellon University

Special Topics:

Self-Driving Database Management Systems

P2: Pilot Control Plane

@Kushagra Singh
@Tim Lee

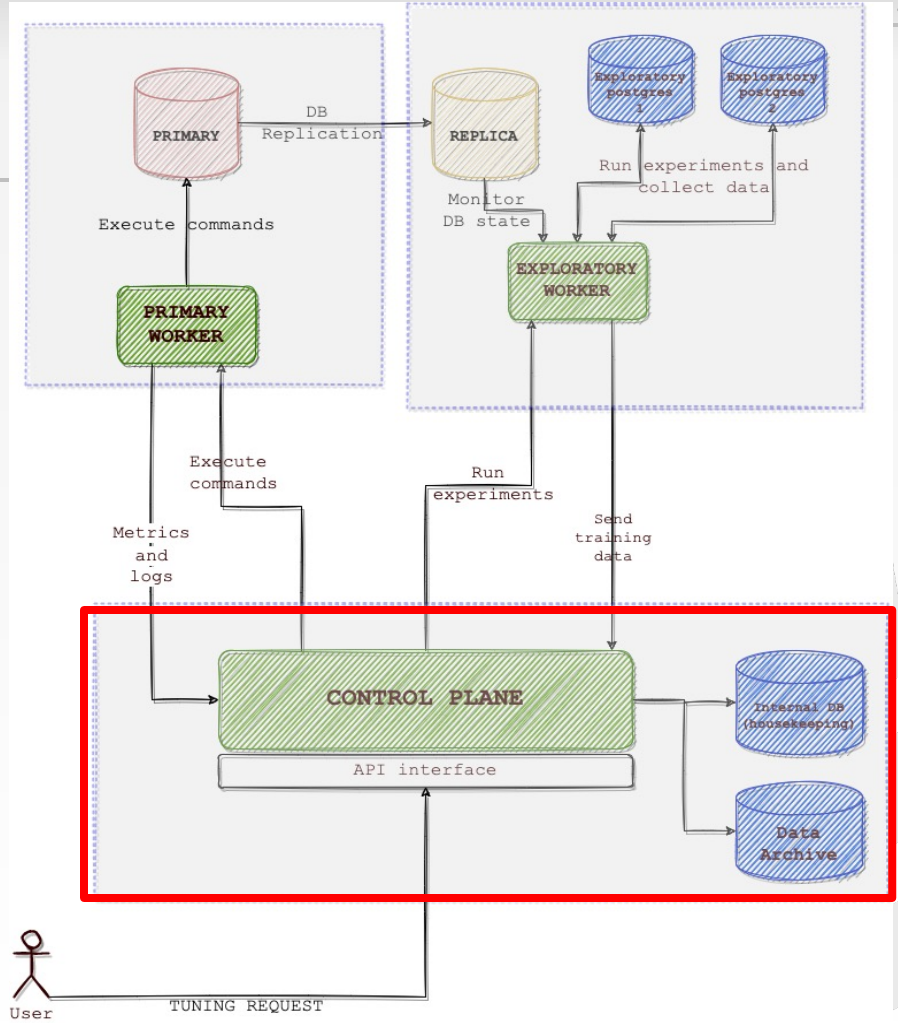


Pilot Control Plane

- Orchestrates the tuning life cycle. Build communication channels between Pilot and other services (DBMS instances, ML components).
- Some examples:
  - Capture and archiving data from PG instances
  - Pipelining ML models training & inference

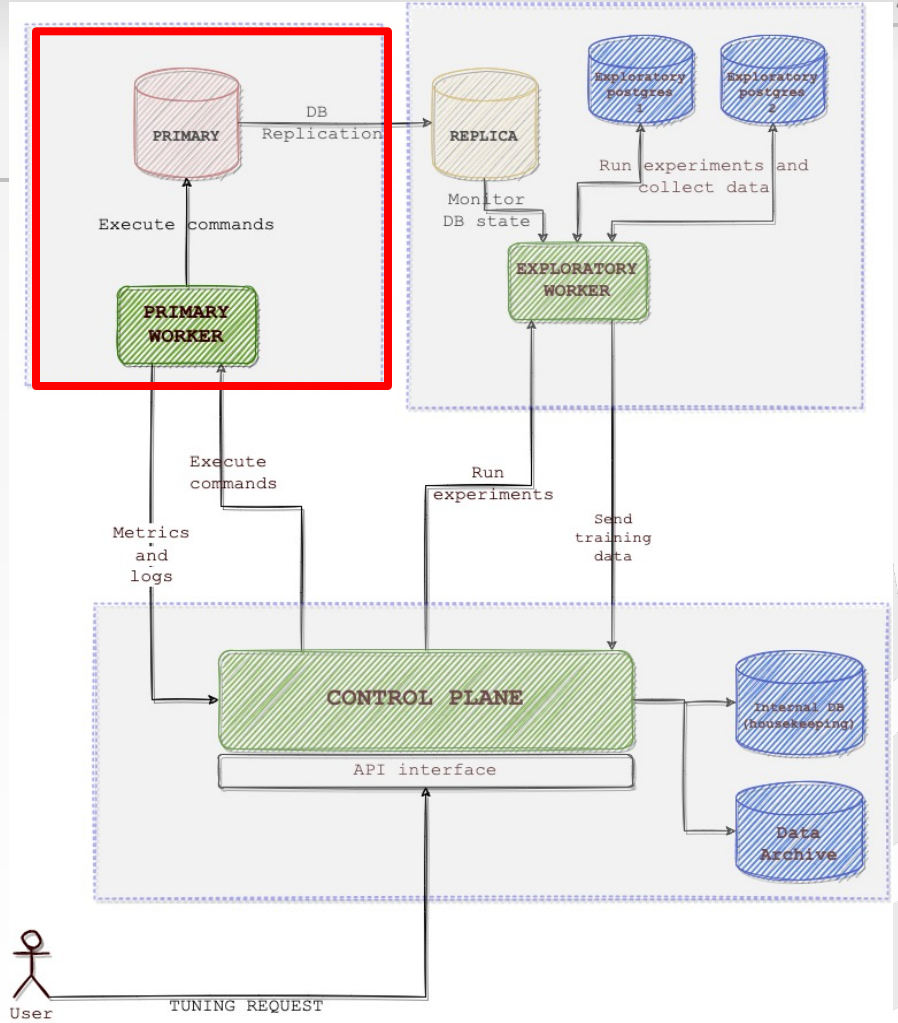
Architecture

- Control plane
  - Spawn primary worker and exploratory worker
  - Issue commands to workers
  - Collect and archive data
  - Handle tuning life cycle; maintain states



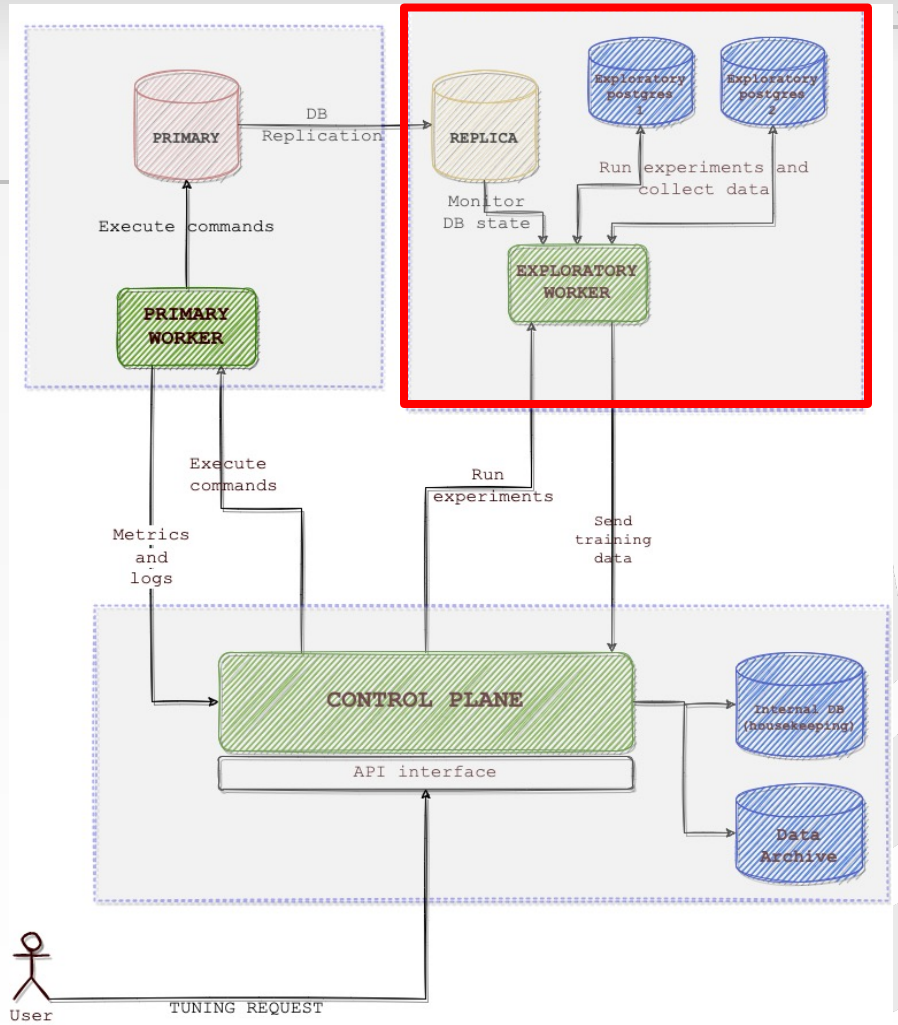
Architecture

- Primary Worker
  - Capture workload, transfer back to Control Plane
  - Apply recommended actions

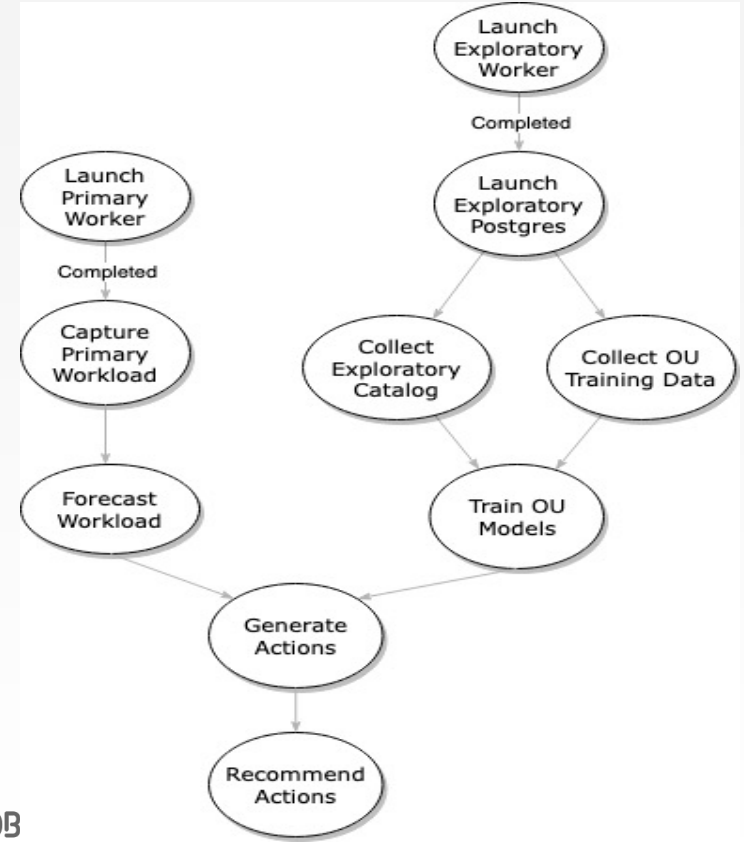


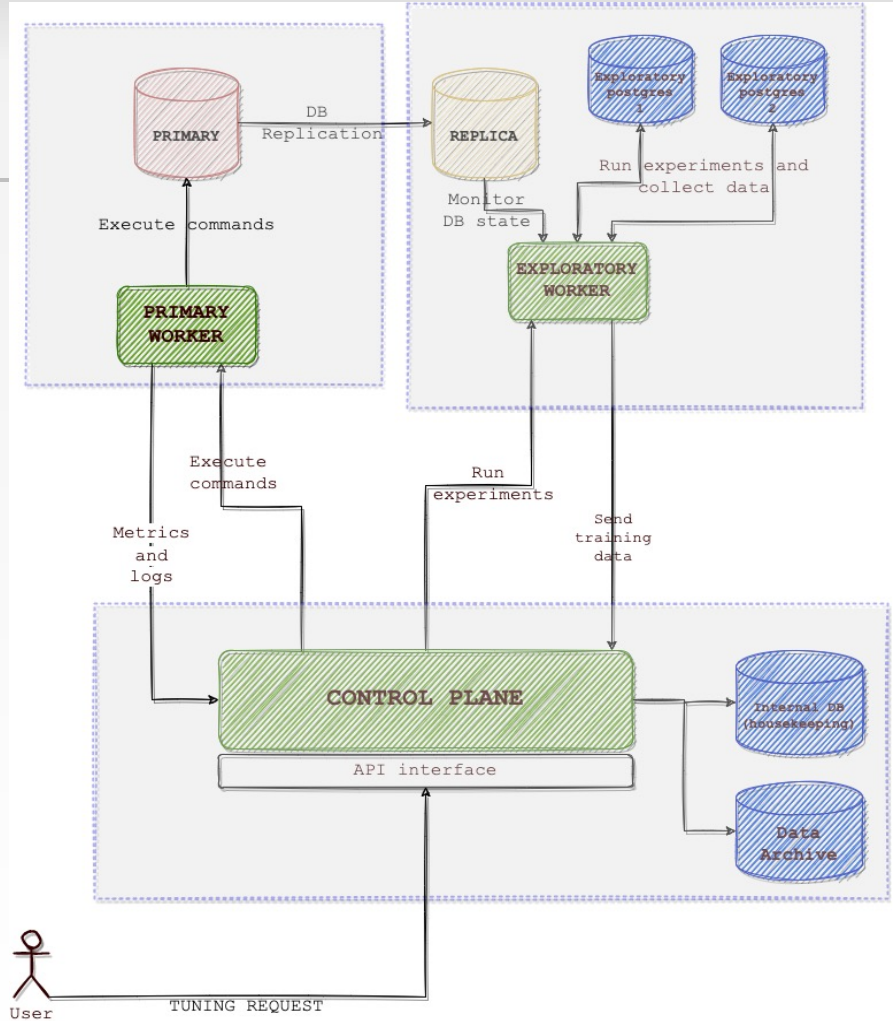
Architecture

- Exploratory Worker
  - Start / stop exploratory PG instances
  - Take snapshot from replica (if specify)
  - Collect data (execution cost, catalog, etc.)



Sample Workflow





DAG Execution Engine

- An execution engine that can run DAGs consiting of customized commands
- More extensible than fixed workflow per endpoint

```
}, {
    "command_type": "COLLECT_DATA_FROM_EXPLORATORY",
    "command_name": "COLLECT_DATA_FROM_EXPLORATORY_abc",
    "parent_command_names" : [
        "LAUNCH_EXPLORATORY_POSTGRES_1"
    ],
    "config": {
        "target": "replica",
        "data_collector_type": "CATALOG",
        "data_collector_config": {
            "dummy": true
        }
    }
}, {
```

```
[{
    "command_type": "LAUNCH_PRIMARY_WORKER",
    "command_name": "LAUNCH_PRIMARY_WORKER_abc",
    "parent_command_names" : []
}, {
    "command_type": "LAUNCH_EXPLORATORY_WORKER",
    "command_name": "LAUNCH_EXPLORATORY_WORKER_abc",
    "parent_command_names" : []
}, {
    "command_type": "LAUNCH_EXPLORATORY_POSTGRES",
    "command_name": "LAUNCH_EXPLORATORY_POSTGRES_1",
    "config": {
        "snapshot": false
    },
    "parent_command_names" : [
        "LAUNCH_EXPLORATORY_WORKER_abc"
    ]
}, {
    "command_type": "CAPTURE_PRIMARY_WORKLOAD",
    "command_name": "CAPTURE_PRIMARY_WORKLOAD_abc",
    "config": {
        "time_period": 10
    },
    "parent_command_names" : [
        "LAUNCH_PRIMARY_WORKER_abc"
    ]
}, {
    "command_type": "COLLECT_DATA_FROM_EXPLORATORY",
    "command_name": "COLLECT_DATA_FROM_EXPLORATORY_abc",
    "parent_command_names" : [
        "LAUNCH_EXPLORATORY_POSTGRES_1"
    ],
    "config": {
        "target": "replica",
        "data_collector_type": "CATALOG",
        "data_collector_config": {
            "dummy": true
        }
    }
}, {
    "command_type": "STOP_EXPLORAOTRY_POSTGRES",
    "command_name": "STOP_EXPLORAOTRY_POSTGRES_2",
    "config": {
        "launch_command": "LAUNCH_EXPLORATORY_POSTGRES_1"
    },
    "parent_command_names" : [
        "COLLECT_DATA_FROM_EXPLORATORY_abc"
    ]
}]
```

Exploratory Data Collector

- A generic interface to collect data on exploratory PG instance
- E.g. training data collection team can implement this base class to collect data with benchbase / sqlsmith

```python
class BaseDataCollector(ABC):
    def __init__(self, postgres_port, data_dir, config):
        """
        postgres_port: port on which the exploratory postgres is running
        data_dir: directory where collected data will be stored
        config: any config to be passed to the DataCollector
        """

        self.postgres_port = postgres_port
        self.data_dir = data_dir
        self.config = config

    @abstractmethod
    def setup(self):
        pass

    @abstractmethod
    def collect(self):
        pass

    @abstractmethod
    def cleanup(self):
        pass
```

What we completed

- Messaging infrasturcutre between 3 workers
- DAG Execution Engine
- Commands supported:
  - Launch Primary Worker
  - Launch Exploratory Worker
  - Capture + archive workload
  - Launch exploratory PG instance (can take snapshot from replica)
  - Stop exploratory PG instance
  - An extendible interface to collect training data

Demo

Original Timeline

Mar 17

- Server setup
- Control Plane messaging setup

Mar 24

- Worker (primary + exploratory) spawn setup

Mar 31

- Exploratory PG orchestration

Apr 7

- Workload capture (75%)
- Workload archival

Apr 14

- Training data execution and archival (100%)

Future

- train OU models (125%)
- Forecasting, action recommendation

Future Work

- OU model training
- Workload forecasting model training + inference
- Action generation + picking
- Run 3 workers on different node (Ansible)
- Support concurrent tuning sessions
- Web UI?
- TODO list: https://github.com/cmu-db/noisepage-control/issues/7