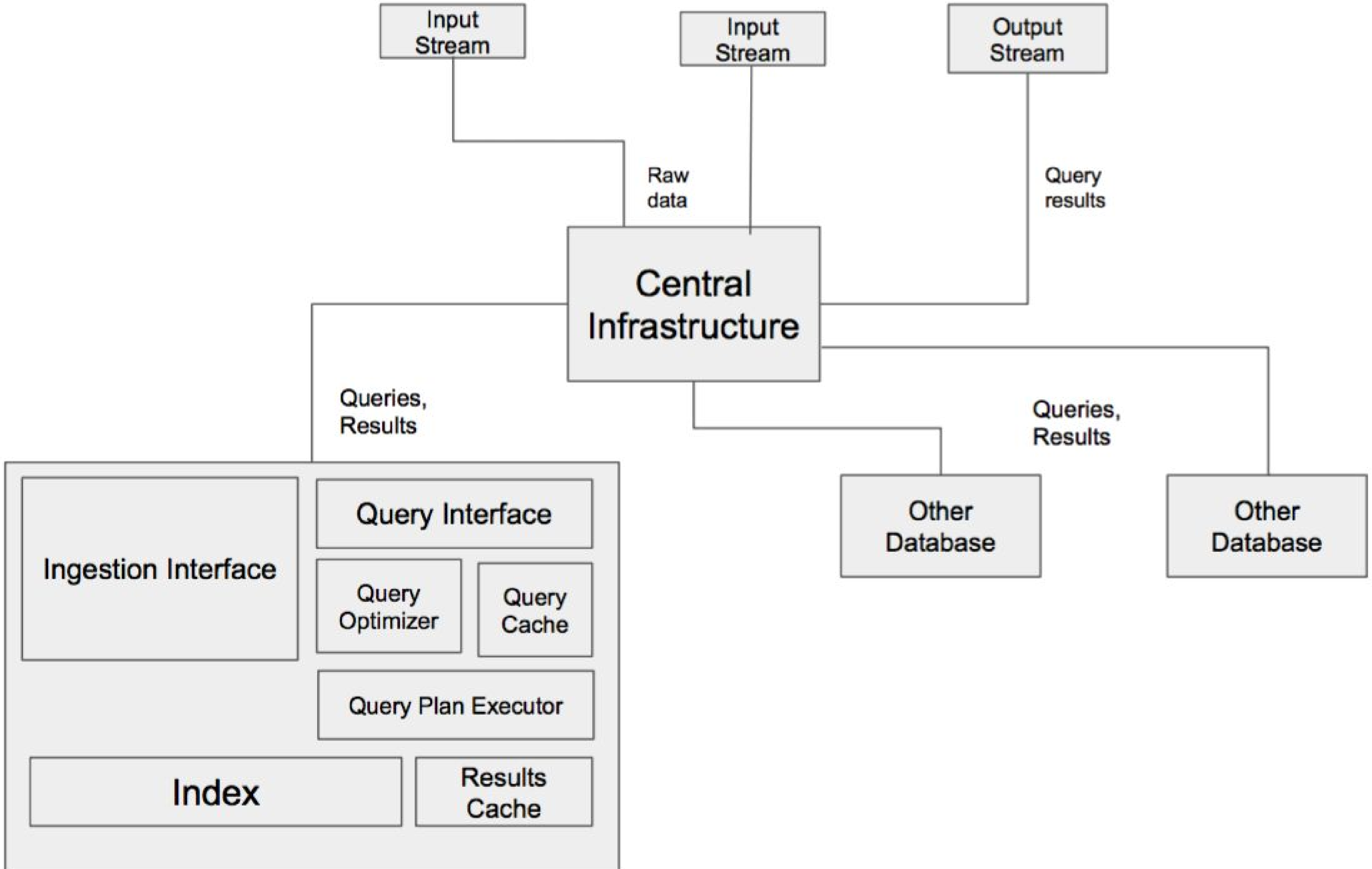
Lightweight Runtimes (Galileo IoT)

Team Sparkle

Dhinesh
Shiva
Keno
Guru


Input Stream
Input Stream
Output Stream
Raw data
Query results
Central Infrastructure
Queries, Results
Queries, Results
Other Database
Other Database
Ingestion Interface
Query Interface
Query Optimizer
Query Cache
Query Plan Executor
Index
Results Cache

Central Infrastructure

- Web Interface
  - Queries in form of `HTTP GET/POST`
  - Most queries decided a priori
  - Insert:
    - `http://128.205.39.183/insert?timestamp=1446175861&temp=17.4&room=2&occupants=4`
    - Python - tornado + requests
- 2 Sensors posting at 1Hz
- Occupants: `random.randint(0, 4)`
  - Only updates every 30seconds

Central Infrastructure

- Currently stored in SQLite DB
- Eventually push to Java via sockets

The Obligatory Code Slide(s)

- We’ve come a long way

The Obligatory Code Slide(s)

```
#TODO: Someone write this
```

The Obligatory Code Slide(s)

```
class SensorHandler(tornado.web.RequestHandler):
    def get(self):
        session = database_models.get_session()
        response = ''
        data = self.query_string_as_dict()
        room_id = int(data['room'])
        timestamp = int(data['timestamp'])
        temperature = float(data['temperature'])
        occupants = int(data['occupants'])

        room = None
        try:
            room = database_models.query_first(Room, {'id' : room_id}, session)
        except:
            logger.error("Did not find room for id '%s'" % (room_id))
            room = Room(id=room_id)
            session.add(room)
        finally:
            database_models.update_room(room, session)
            sensor = Sensor(room=room_id, timestamp=timestamp, temperature=temperature, occupants=occupants)
            database_models.update_sensor(sensor, session)
            self.write(response)
```

The Obligatory Code Slide(s)

- Using `sqlalchemy` as ORM

```
class Room(Base):
    __tablename__ = 'room'
    id = Column(Integer, primary_key=True)

class Sensor(Base):
    __tablename__ = 'sensor'
    id = Column(Integer, primary_key=True)
    room = Column(Integer, ForeignKey("room.id"))
    timestamp = Column(Integer)
    temperature = Column(Float)
    occupants = Column(Integer)
```

The Obligatory Code Slide(s)

- HTTP implemented using `requests`

```
def upload(host, port, payload):
    timestamp = time_since_epoch()
    payload['timestamp'] = timestamp
    try:
        r = requests.get('http://%s:%d/insert' % (host, port), params=payload)
        if not r.ok:
            raise Exception('%d: %s' % (r.status_code, r.text))
    except Exception, e:
        logger.error("Failed to upload to server: %s" % (str(e)))
```

The Obligatory Code Slide(s)

- HTTP implemented using `requests`

```
def upload(host, port, payload):
    timestamp = time_since_epoch()
    payload['timestamp'] = timestamp
    try:
        r = requests.get('http://%s:%d/insert' % (host, port), params=payload)
        if not r.ok:
            raise Exception('%d: %s' % (r.status_code, r.text))
    except Exception, e:
        logger.error("Failed to upload to server: %s" % (str(e)))
```

- So technically...we lied!
    - Only GET, no POST

Processing Query Requests

```
private static boolean initiateServer(int portNumber ){
	ServerSocket serverSocket = new ServerSocket(portNumber);
	try{
		while(true){
			Socket socket = serverSocket.accept();
			ObjectInputStream ois = new
ObjectInputStream(socket.getInputStream());
			Object request = ois.readObject();
			String response = processRequest(request);
			ObjectOutputStream oos = new
ObjectOutputStream(socket.getOutputStream());
			oos.writeObject(response);
		   ois.close();
		   oos.close();
		}

	}catch(Exception e){

	}finally{
		serverSocket.close();
	}
}
```

```
private String processRequest(Object request){
	if(request instanceof Insert){
		//insert into mentioned table
	}else if(request instaceof Select){
		return processQuery(select);
	}
}
```

Windowed Inserts

```
public class WindowedTable {
    LinkedList<String> windowedTable = new
LinkedList<String>();
   String tableName = "";
    int window_records= 10 * 60 * 5;  //default size
    int currentIndexRead=0;

public WindowedTable(String tableName){
        this.tableName = tableName;
    }

public String insert(String rowString){
    if(windowedTable.size() >= window_records)
    {
        windowedTable.poll();
    }
    boolean isDone = windowedTable.offer(rowString);
    if(isDone) return "Success";
    else return "failure";
}
```

```
 public String readOneTuple(){
        if(currentIndexRead == windowedTable.size())
        {
              currentIndexRead = 0;
              return null;
        }
        currentIndexRead++;
        return windowedTable.get(currentIndexRead-
1);
    }

    public void setWindowSize(int
frequency_InsertsPerMinute,int multiplier)
    {
        int temp =
frequency_InsertsPerMinute*multiplier*60;

        //poll the extra elements in current window
Records
        for(int i=temp;i < window_records;i++ )
              windowedTable.poll();

        window_records = temp;
    }
}
```

Windowed Inserts

- Initialising Windowed Table Object for each Table while parsing create table statements and storing it in a map
- Scan Operators for each table reads rows from underlying windowed tables instead of files
- Window of records streaming into the DB is being tracked and only those records being inserted for the past (x) minutes will be stored in table / window_record variable will be updated by the ingestion interface

```
public static HashMap<String, WindowedTable> table_windowedObject_map =  new HashMap<>();
    if(statement instanceof CreateTable){
        CreateTable createTableObj = (CreateTable) statement;
        prepareTableSchema(createTableObj);
        testInsert();//tested inserts //inserts will be done by socket listeners
    }
    Inside PrepareSchema() -> table_windowedObject_map.put(tableName, new WindowedTable(tableName));
}
```

Query Intuition tested with inserts

```
CREATE TABLE ROOMPROPERTIES (
           id                  INT,
          room_id             INT,
          temperature         INT,
          person_count        INT,
          timestamp           INT
);
```

INSERT “1|23|70|0|1100 ” → “Success” returned from database

READ “person_count | 23room_id” → 0 returned from database runs the following query and returns SUM(person_count)

```
SELECT SUM(person_count) FROM ROOMPROPERTIES WHERE room_id = 23;
```