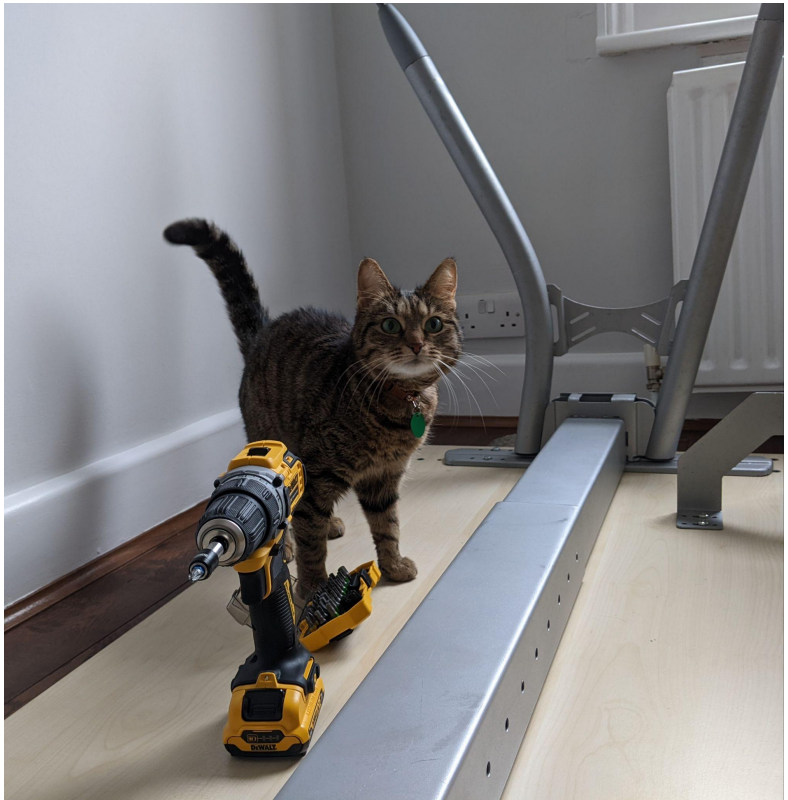
You decided to send a table

ACCU on Sea 2026

Evgenii Seliverstov @ Bloomberg

Send a table

- Pack, wrap, send
- Multi-region communications
- High latency (weeks)
- Assembly required
- Hardware issues
- Cat assistance included



Send a table in a Unix way

- **Rely on Unix's philosophy with pre-built tools**

```
mutt -a table.csv -s "Table" -- bob@acme.org
```

...the file flies over SMTP...

```
mutt -f /var/mail/bob -e "exec last-entry;
exec pipe-message" | ripmime -i - -d table.csv
```

A table at arm's length

- **Table is in memory**
- **Cross-language support**
- **Send it as a file**

```
std::vector<std::vector<std::string>> table;
std::ofstream os("table.csv");
os << csv_adapter(table)
```

...the file flies over the network...

```
import pandas as pd
table = pd.read_csv("table.csv")
```

Databases contain tables

- **Common shared state**
- **Use RDBMS to store your tables**
- **Sender:**
  - **Inserts tabular data**
  - **Sends a key**
- **Receiver:**
  - **Receives a key (or assumes latest)**
  - **Reads tabular data**

Table in a lake

- **Drop your table into a lake**
- **Data Lake: Store now, access later**
- **Object store over S3 or filesystem**
- **Modifying tables? → Create a new one**
- **How do we track changes? → Catalogs**
- **Access data? → Query engines**
- **Efficient at scale**

Table foundation

- What lies beneath a table?
- Parquet
  - Better than CSV
  - On-disk format
  - Common, interoperable format
  - Columnar encoding
- Still a file

Parquet - to store

Parquet is good for storing

- Rich data structures
- Compressed
- Chunked into row groups

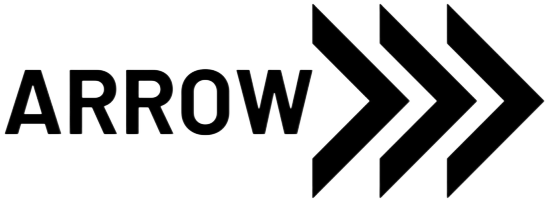
Not the best for compute

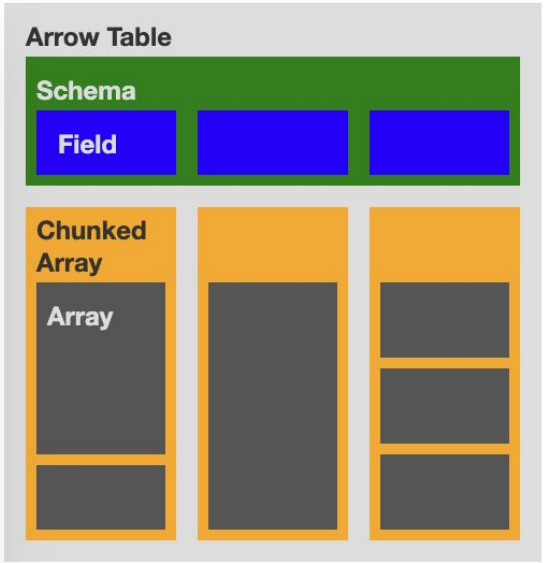
- Files are decoupled from sender
- Serialisation is expensive
- Memory usage is doubled

Why do we even need to serialise?

Apache Arrow - to compute

- Welcome to Apache Arrow
- In-memory format for data processing
- Language agnostic
- Columnar format
- Interoperability with Parquet
- Literally a table





Arrow IPC

- **IPC is a layout and schema for Arrow Arrays**
- **File layout is exactly the same as memory**
- **No CPU overhead for serialisation**
- **Write a table:**

```
use arrow_ipc::writer::FileWriter;
let batch = record_batch!(("a", Int32, [1, 2, 3]))?;
let mut writer = FileWriter::try_new(&mut file, &batch.schema())?;
writer.write(&batch)?;
writer.finish()?
```

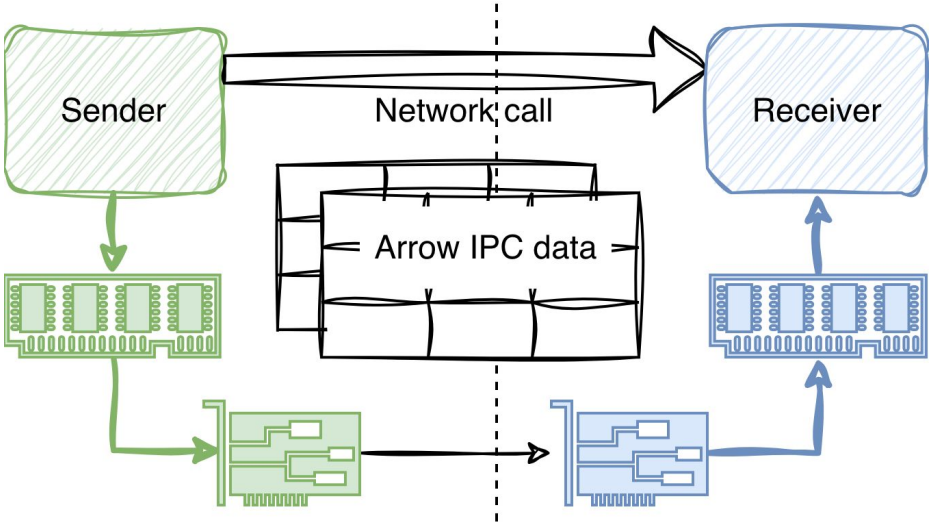
Arrow IPC zero copy reads

- **Do we need to read and write to files?**
- **Memory-mapped files with Arrow IPC**
- **Zero copies**
- **No CPU or memory overhead**
- **Read a table:**

```
import pyarrow as pa
source = pa.memory_map(“table.arrow”, “rb”)
table = pa.ipc.RecordBatchFileReader(source).read_all()
```

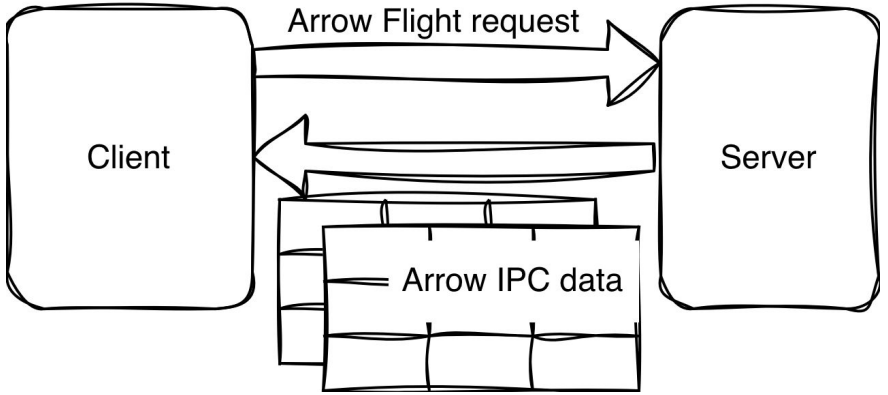
Arrow IPC zero copy writes

- **Send data directly to network**
- **Shared memory pages with NIC**
- **No extra memory copies**
- **No serialisation overhead**
- **Mix-and-match of languages**



Flying tables

- **Not only data format**
- **Arrow Flight**
  - **Ergonomic RPC protocol**
  - **gRPC + Arrow**
- **Arrow Flight SQL**
  - **Interface with databases**
  - **Save tables into tables**
- **Inherently tabular format**



Takeaways

- **Data design influences communication protocols**
- **Use standard in-memory and on-disk formats**
- **Avoid serialisation costs**
- **Language agnosticism is commonplace**

This talk is right there ➡

Discuss? @theirix