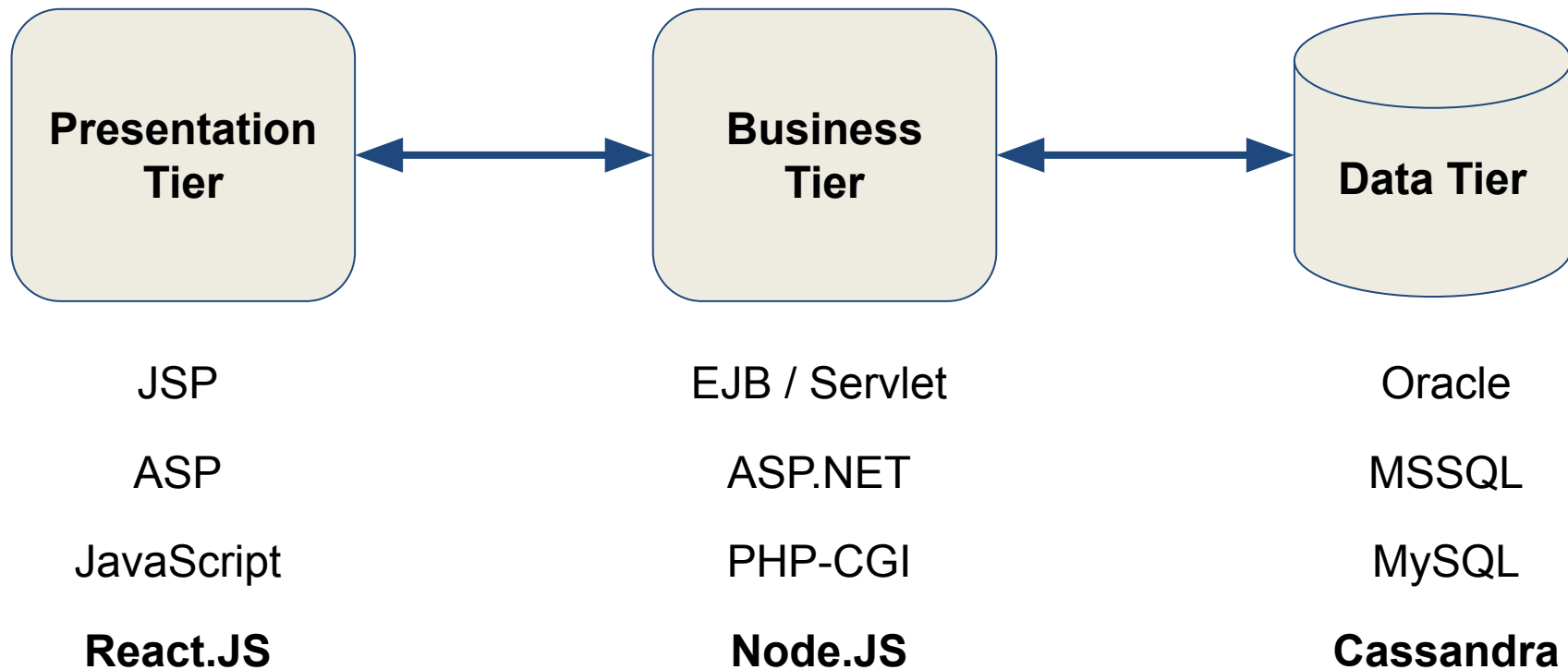


Chapter 3. Cloud Application Architectures

Bilkent University | CS443 | 2020, Spring | Dr. Orçun Dayıbaşı

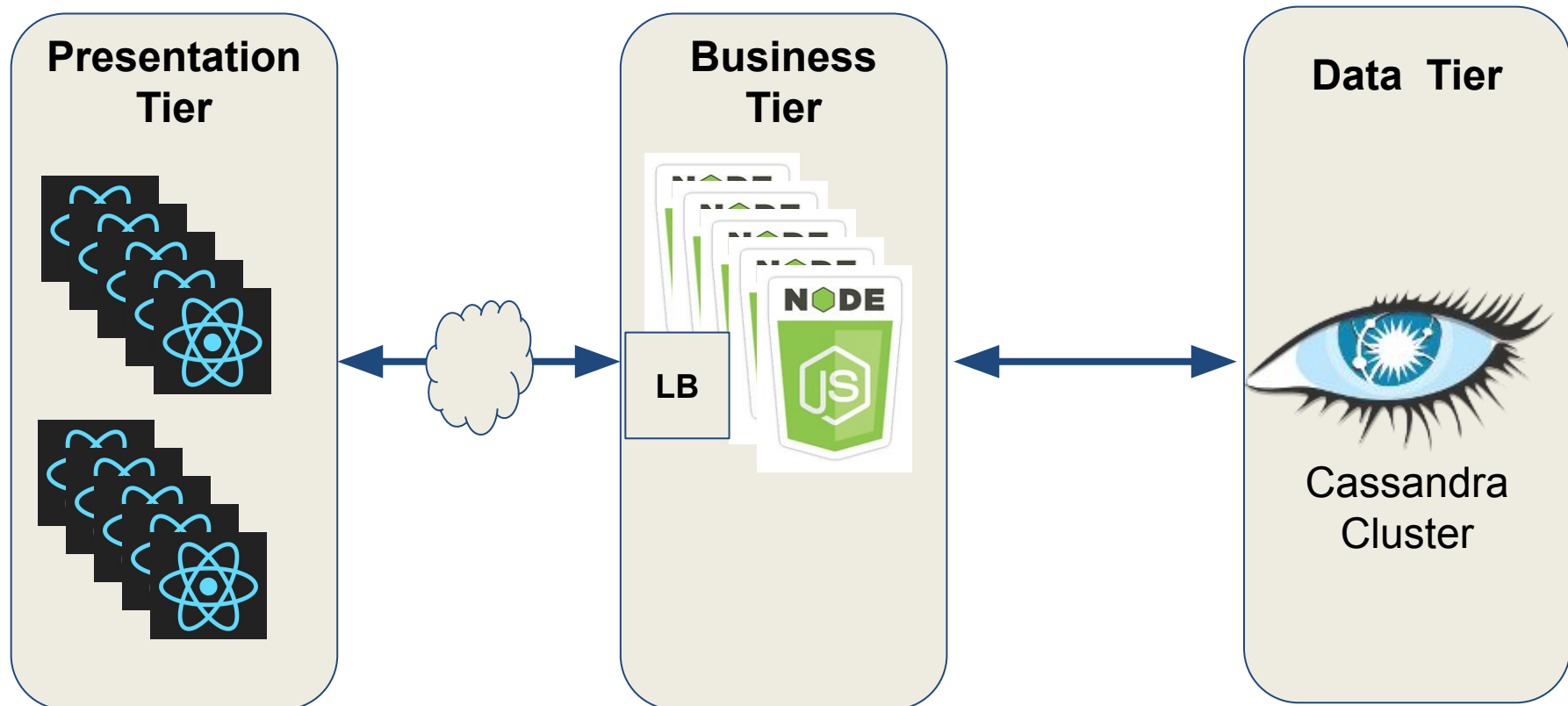
Reference Architectures

- **Ex.1: Modern 3-tier**



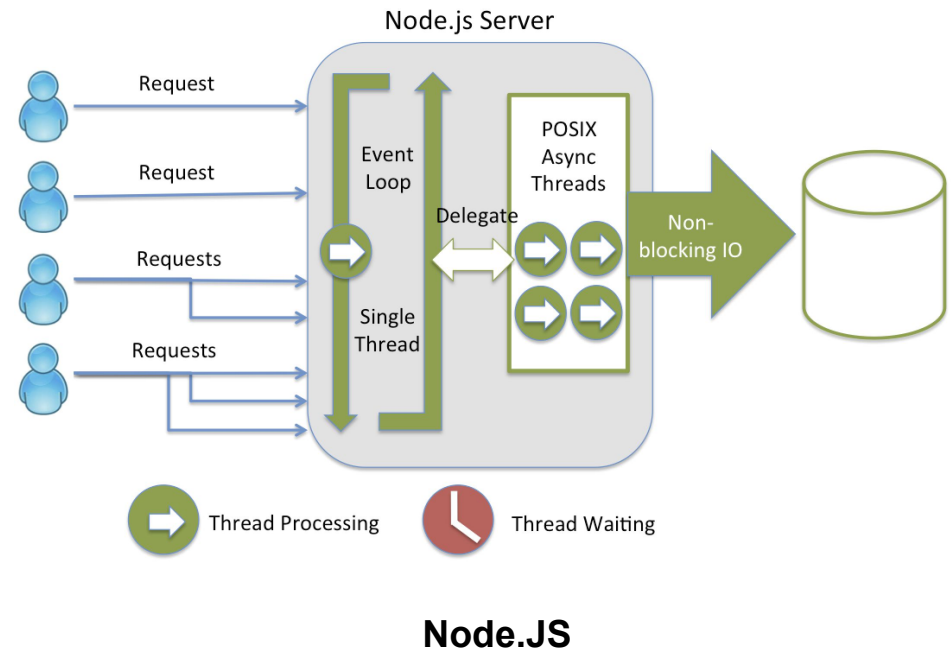
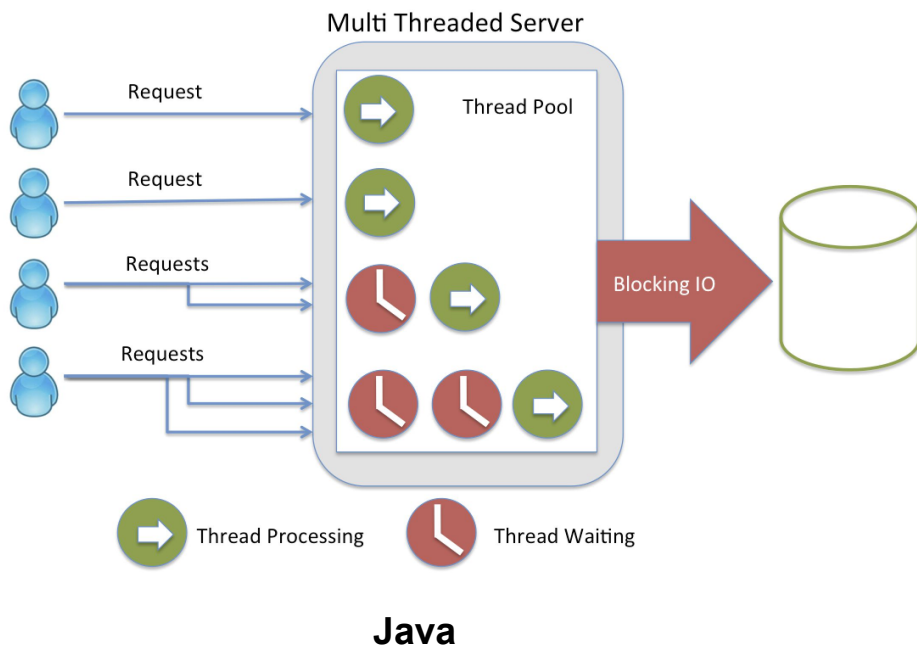
Reference Architectures

- **Ex.1: Modern 3-tier**
 - How this system scales?



Reference Architectures

- **Ex.1: Modern 3-tier**
 - **A little detour:** How to compare/select dev. platforms?



Reference Architectures

- **Ex.1: Modern 3-tier**

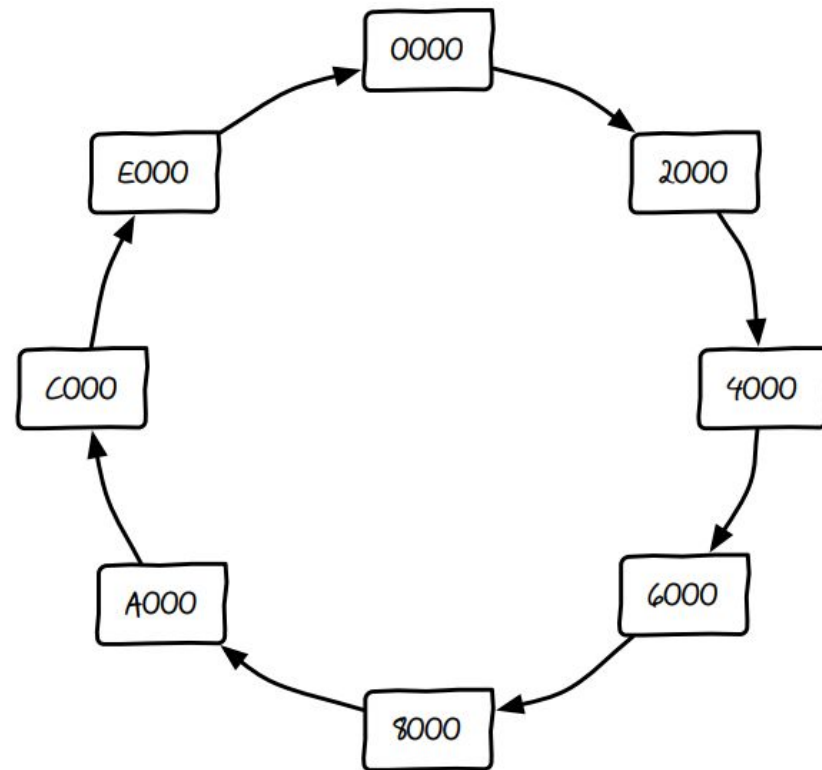
- **A little detour:** How to compare/select dev. platforms? [Speaking intelligently:](#)
 - What kinds of languages are my team comfortable with?
 - What kind of applications do I want to build?
 - In terms of performance:
 - How many requests per second do expect to have?
 - What is the maximum acceptable "average request latency"?
 - What are my environmental constraints?
 - How easy is it to recruit for this platform?

Reference Architectures

- **Ex.1: Modern 3-tier**

- **Cassandra Cluster**

- Each node has a unique token
- Ring structure: the cluster work in a Ring fashion.
- **Write:** use `hash(key)` to find the corresponding node and send to it.
- **Read:** same idea.



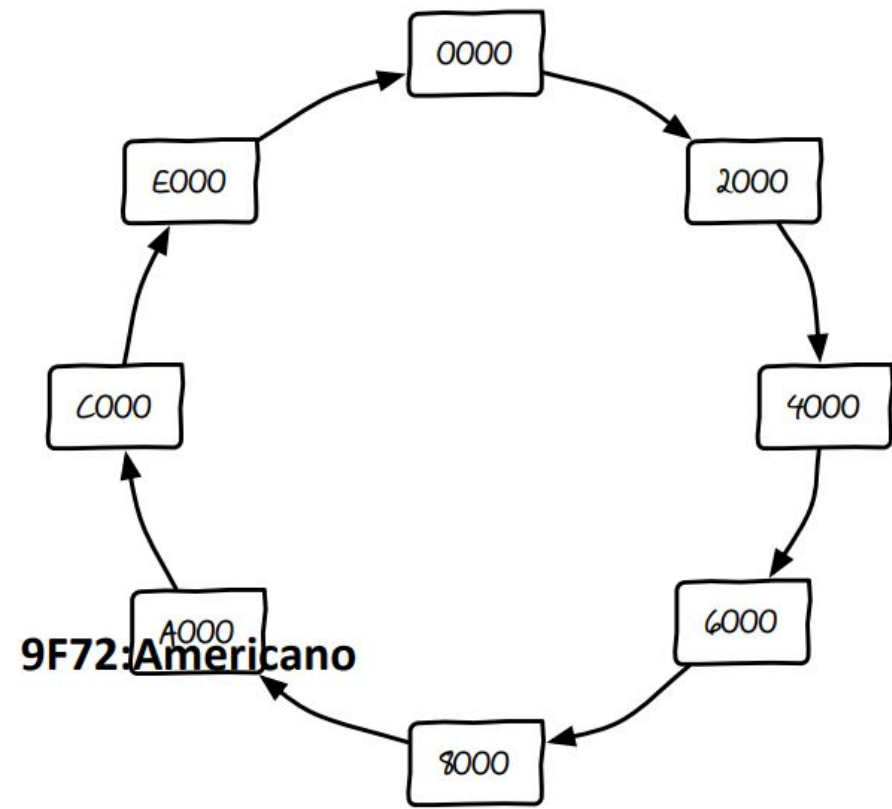
Reference Architectures

- **Ex.1: Modern 3-tier**

- Tim:Americano as KV
- 9F72:Americano
- Replicate data to next N node

- Is fav. coffee type an immutable?

- No → Problem:
Consistency (more on this later)



Reference Architectures

- **Ex.1: Modern 3-tier**

- Pros

- Lots of options for front-end (you can push many functionality to there).
 - Scaling middle tier is not that hard (FWs, etc.)
 - Data tier is highly scalable.

- Cons

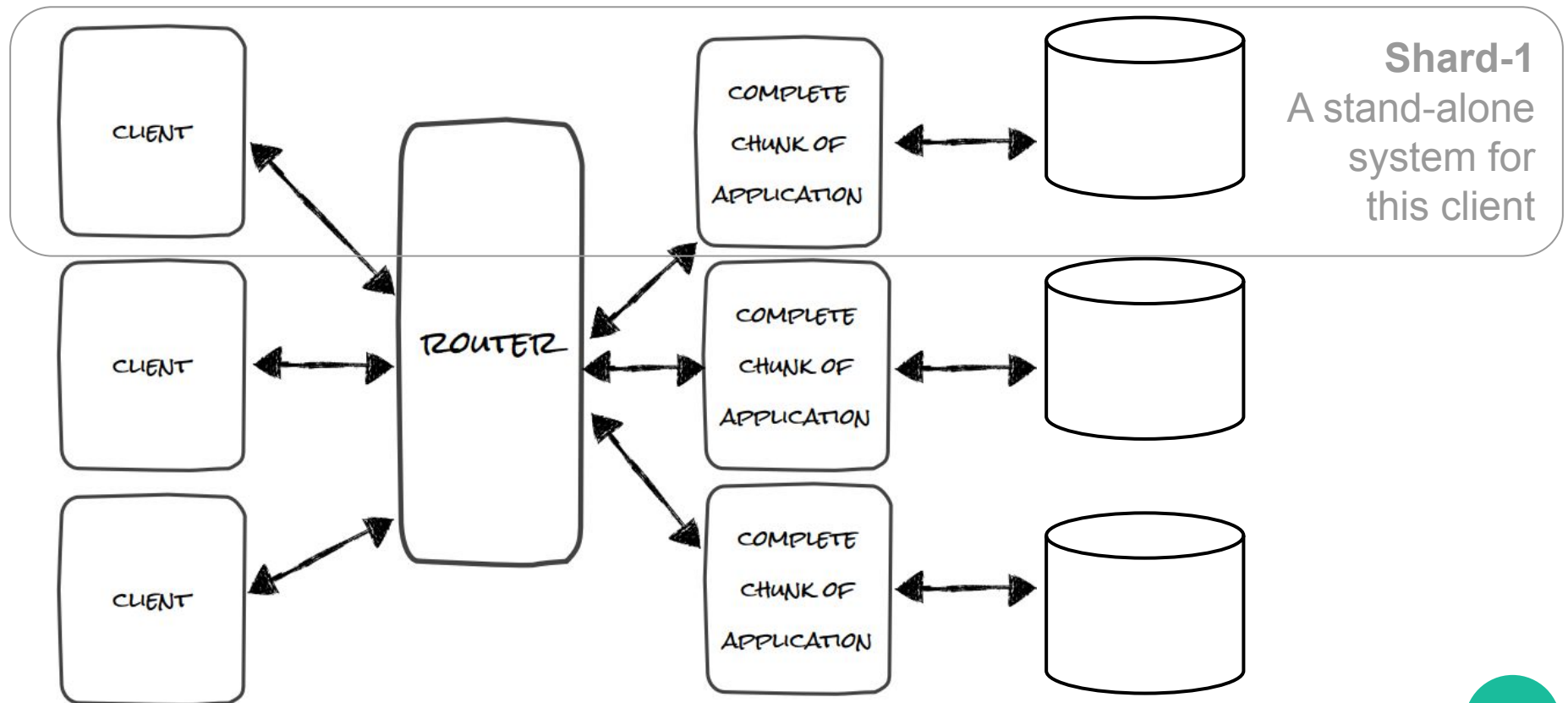
- State in the middle tier (with low latency req.)
 - All you have HTTP req. + Data tier

Reference Architectures



- **Ex.2: Sharded**

- Motto: “Do not build distributed systems if you don’t have to”



Reference Architectures

- **Ex.2: Sharded**

- Pros

- Client isolation easy (e.g. GDPR/KVKK)
 - Simple technologies

- Cons

- Complexity (esp. general monitoring/logging, implementing routing)
 - No comprehensive view of data (Data model, ETL, etc.)
 - Oversized shards
 - HW limits may create an inner dist.sys.

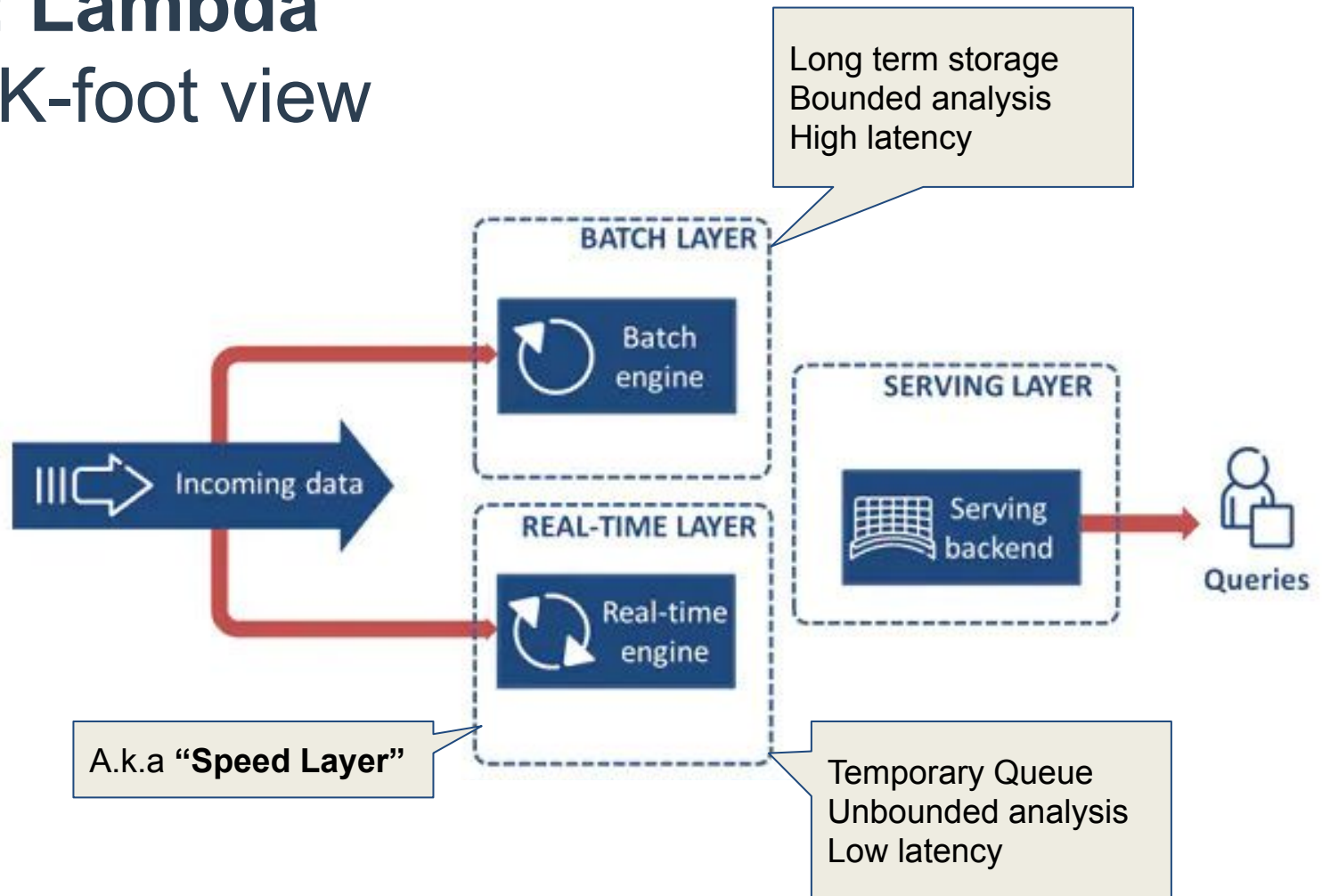
Reference Architectures

- **Ex.3: Lambda**

- Not a general purpose architecture but ...
- **Focus:** dealing with data analysis
- Streaming vs. Batch process
 - Batch: Data in rest (Bounded)
 - Stream: Data in motion (Unbounded)
- Lambda assumes unbounded, immutable data
 - Events are immutable

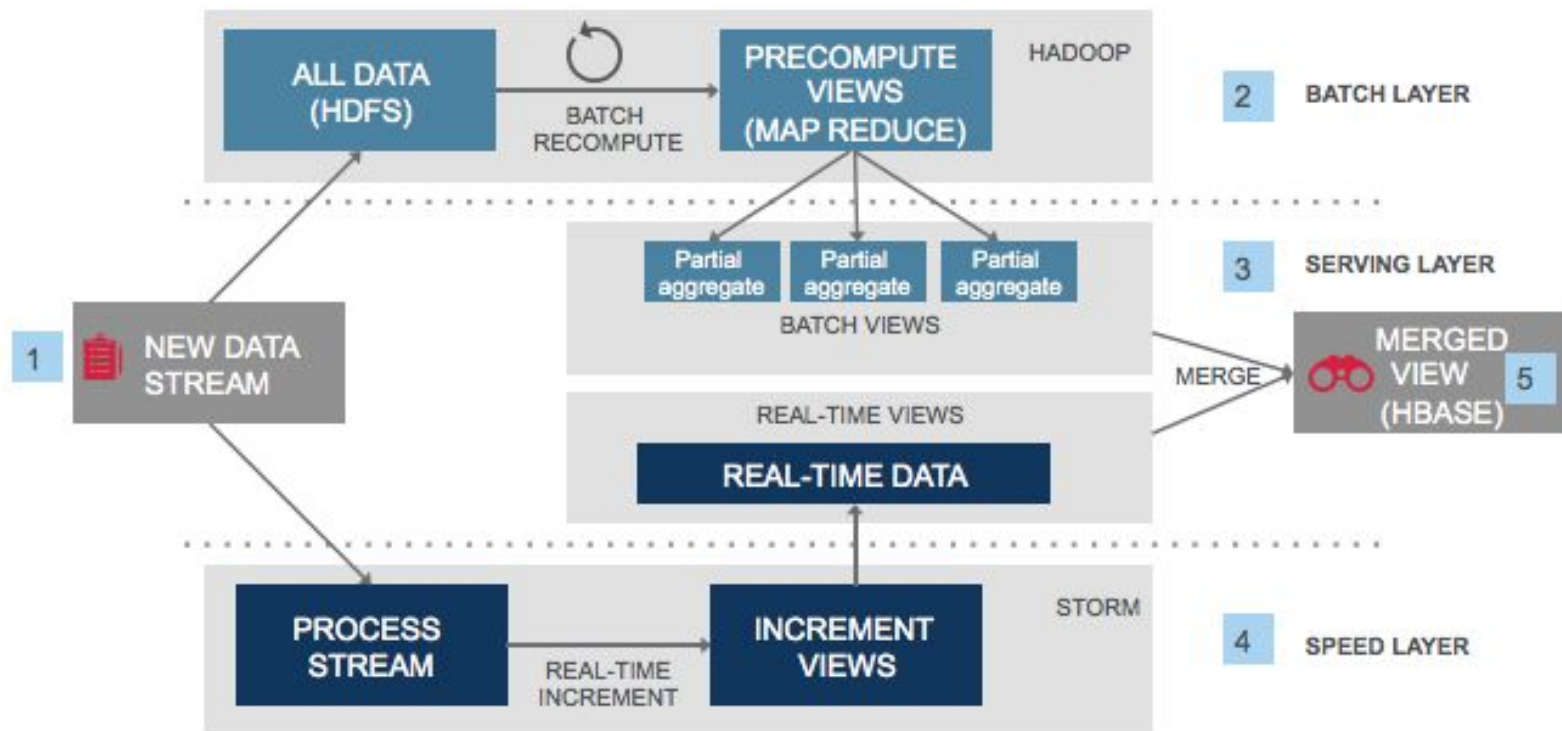
Reference Architectures

- **Ex.3: Lambda**
 - 30K-foot view



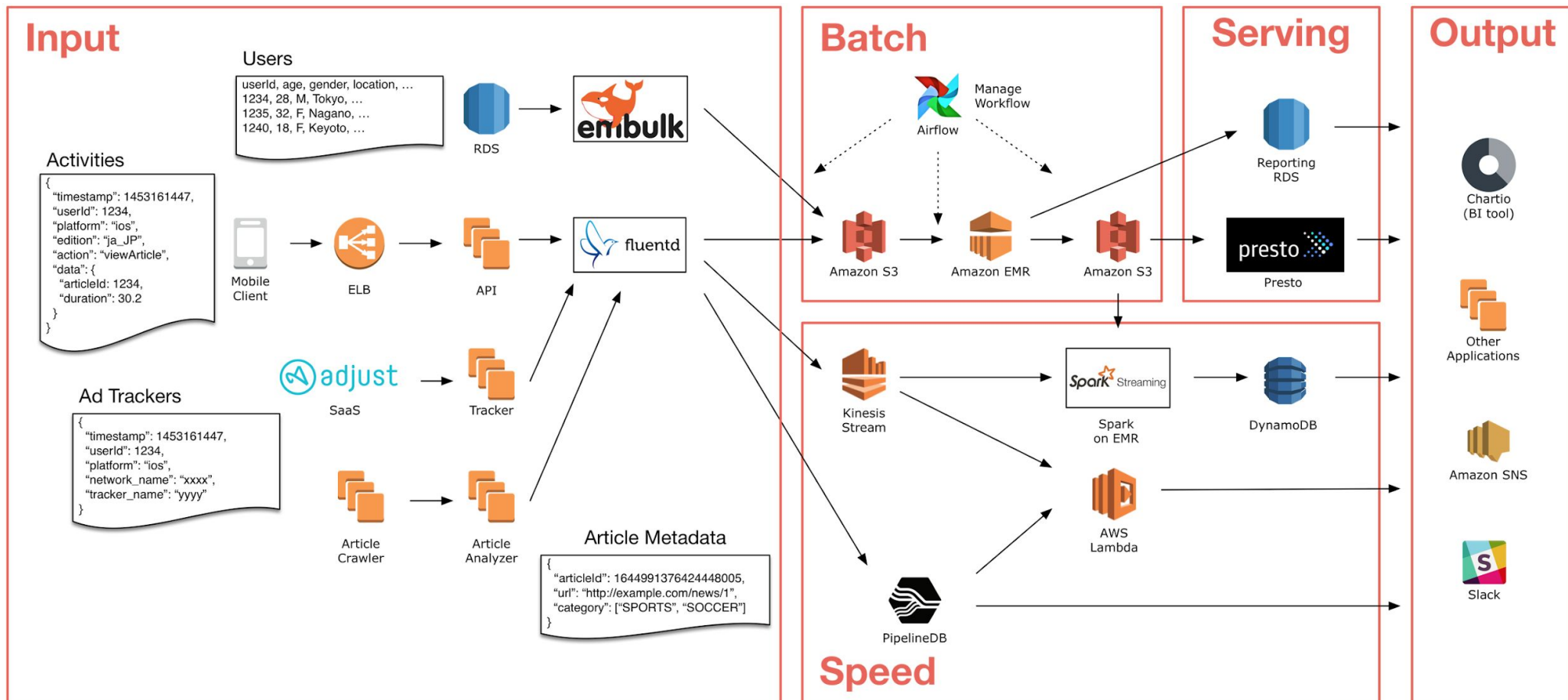
Reference Architectures

- **Ex.3: Lambda**
 - Real life



Reference Architectures

● Ex.3: Lambda



Reference Architectures

- **Ex.3: Lambda**

- Pros

- Optimizes subsystems based on operational requirements
 - Good at unbounded data (previous two are not)

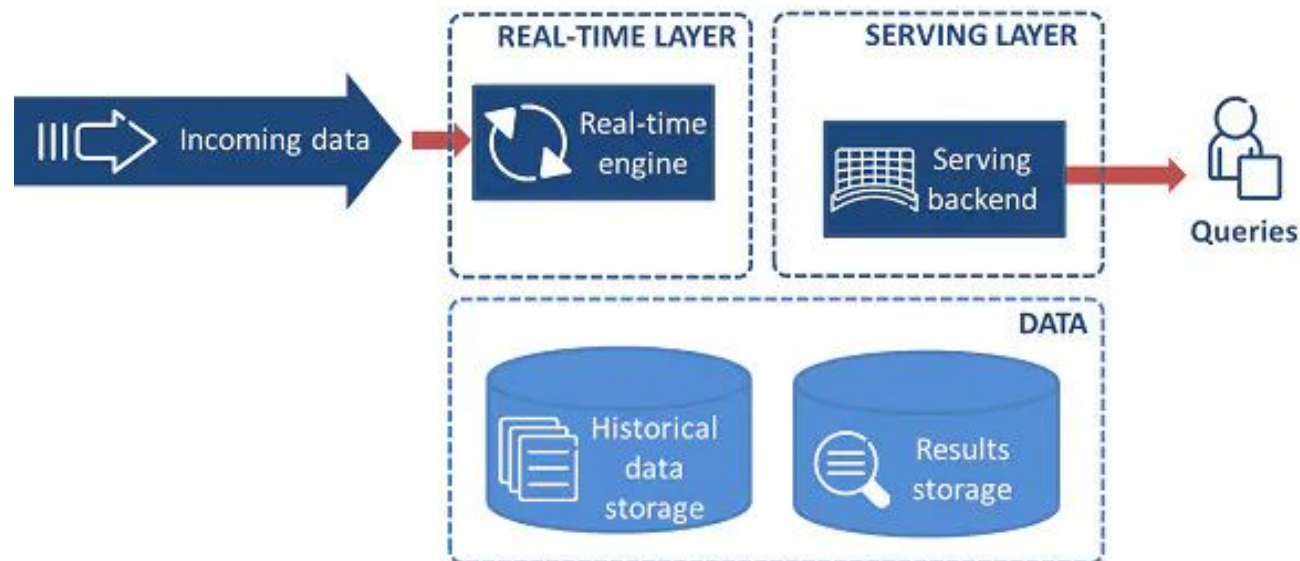
- Cons

- Complexity (operational)
 - Write all the code twice

Reference Architectures

- **Ex.4: Kappa**

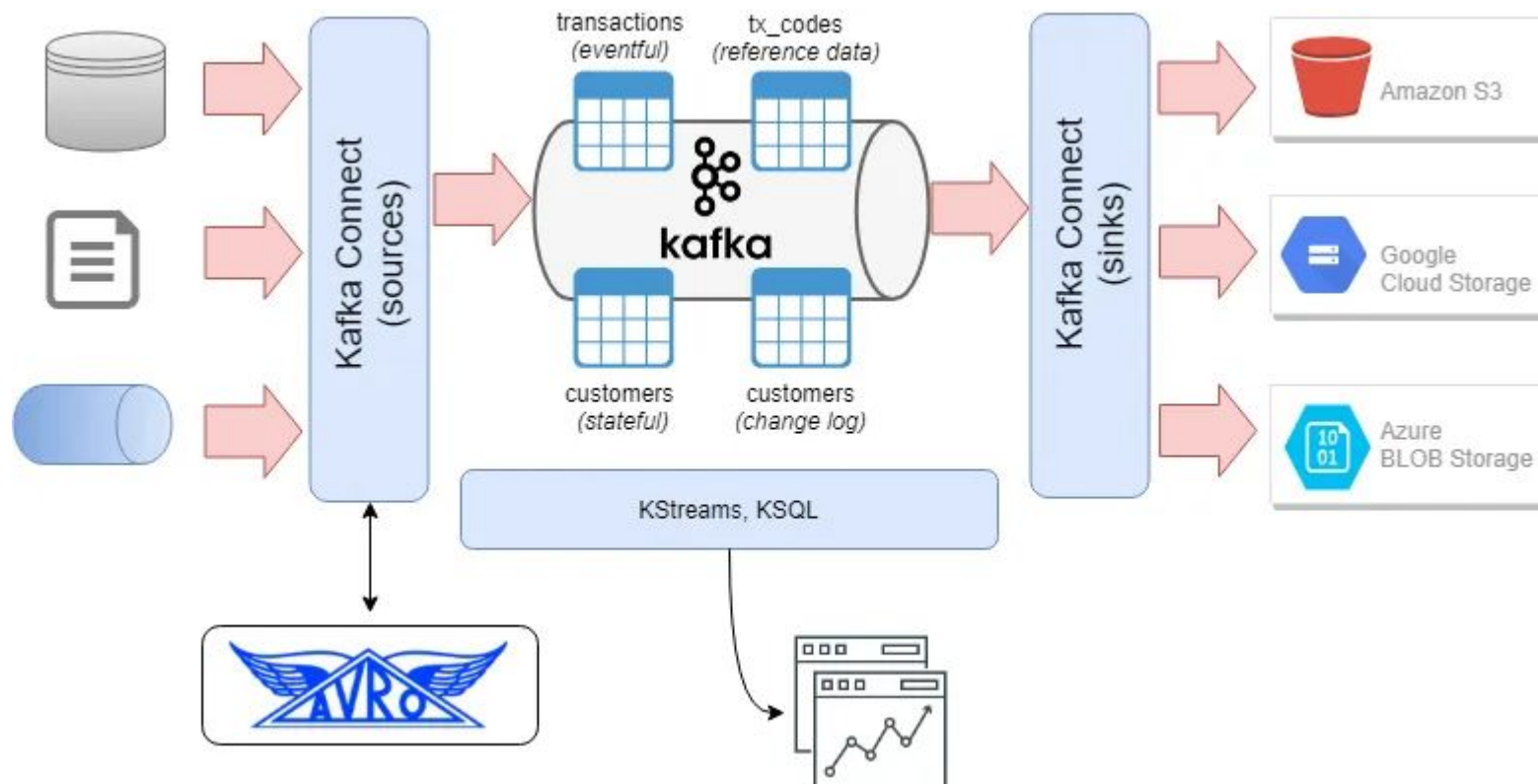
- “Stream” as a first class citizen
- Life is dynamic but DBs are static
 - (tables = streams)



Reference Architectures

- **Ex.4: Kappa**

- Ex: Modern Kafka (not just a MQ)



Reference Architectures

- **Ex.4: Kappa**

- Pros

- Less moving parts (easier to manage)
 - Single processing FW
 - If batch and streaming analysis are identical, then using Kappa is likely the best solution.

- Cons

- Harder to implement (Less moving parts → Complexity is pushed to here)
 - Many real-time use cases requires optimize them (batch/stream) separately.

Reference Architectures

- **Recap**

- Modern cloud applications are complex by nature
- Architectural patterns helps to handle complexity
- Understanding reference architectures is important aspect to take decisions
 - There is no silver bullet
 - Build/buy decisions (CAPEX/OPEX balance)



Q/A