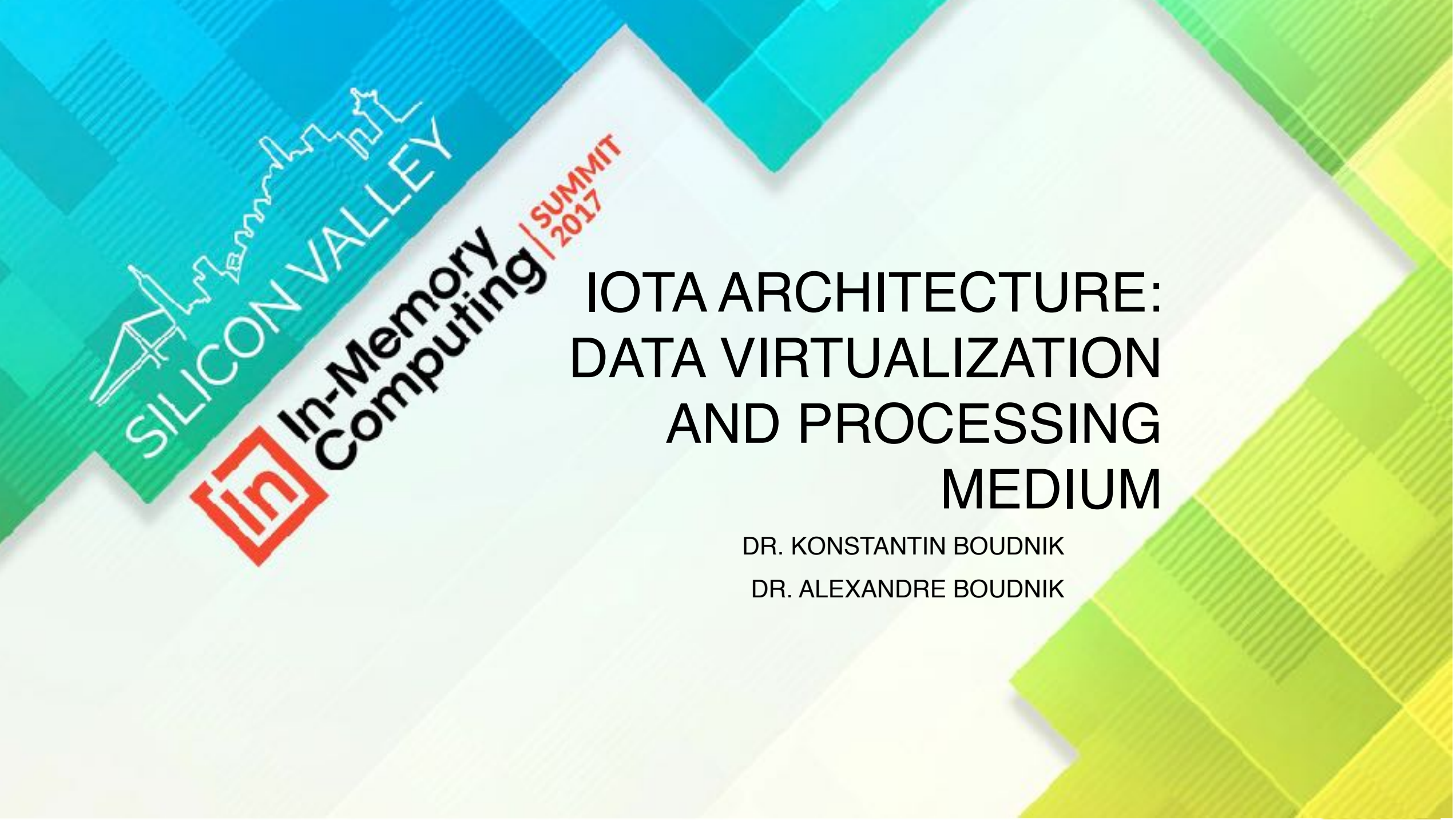


IOTA ARCHITECTURE: DATA VIRTUALIZATION AND PROCESSING MEDIUM

DR. KONSTANTIN BOUDNIK

DR. ALEXANDRE BOUDNIK

DR. KONSTANTIN BOUDNIK

DR.KONSTANTIN BOUDNIK

EPAM SYSTEMS
CHIEF TECHNOLOGIST BIGDATA,
OPEN SOURCE FELLOW

- Over 20+ years of expertise in distributed systems, big- and fast-data platforms
- Apache Ignite Incubator Champion
- Author of 17 US patents in distributed computing
- A veteran Apache Hadoop developer
- Co-author of Apache Bigtop, used by Amazon EMR, Google Cloud Dataproc, and other major Hadoop vendors
- Co-author of the book "Professional Hadoop”

DR. ALEXANDRE BOUDNIK

DR.ALEXANDRE BOUDNIK

EPAM SYSTEMS
LEAD SOLUTION ARCHITECT
BIG& FAST DATA

- Over 25 years of expertise in compilers, query engine for MPP development, computer security, distributed systems, Big Data and Fast Data
- Architect and Visionary at EPAM's BigData CC
- Focusing is on scalable, fault tolerant distributed share-nothing clusters
- Led projects for financial and banking industries with intensive distributed in-memory calculations

AGENDA

- Modern data-processing architectures
- In-memory Data Fabric
- Iota in action: virtual data platform
- Use cases

EVERYTHING IS IN ONE SLIDE

THE REST IS MERE DETAILS

- Don’t separate batch and stream data processing
- Compute should be co-located with data
- Data mutations have to be tracked
- Data concurrency is annoying

- That’s it: you can go now

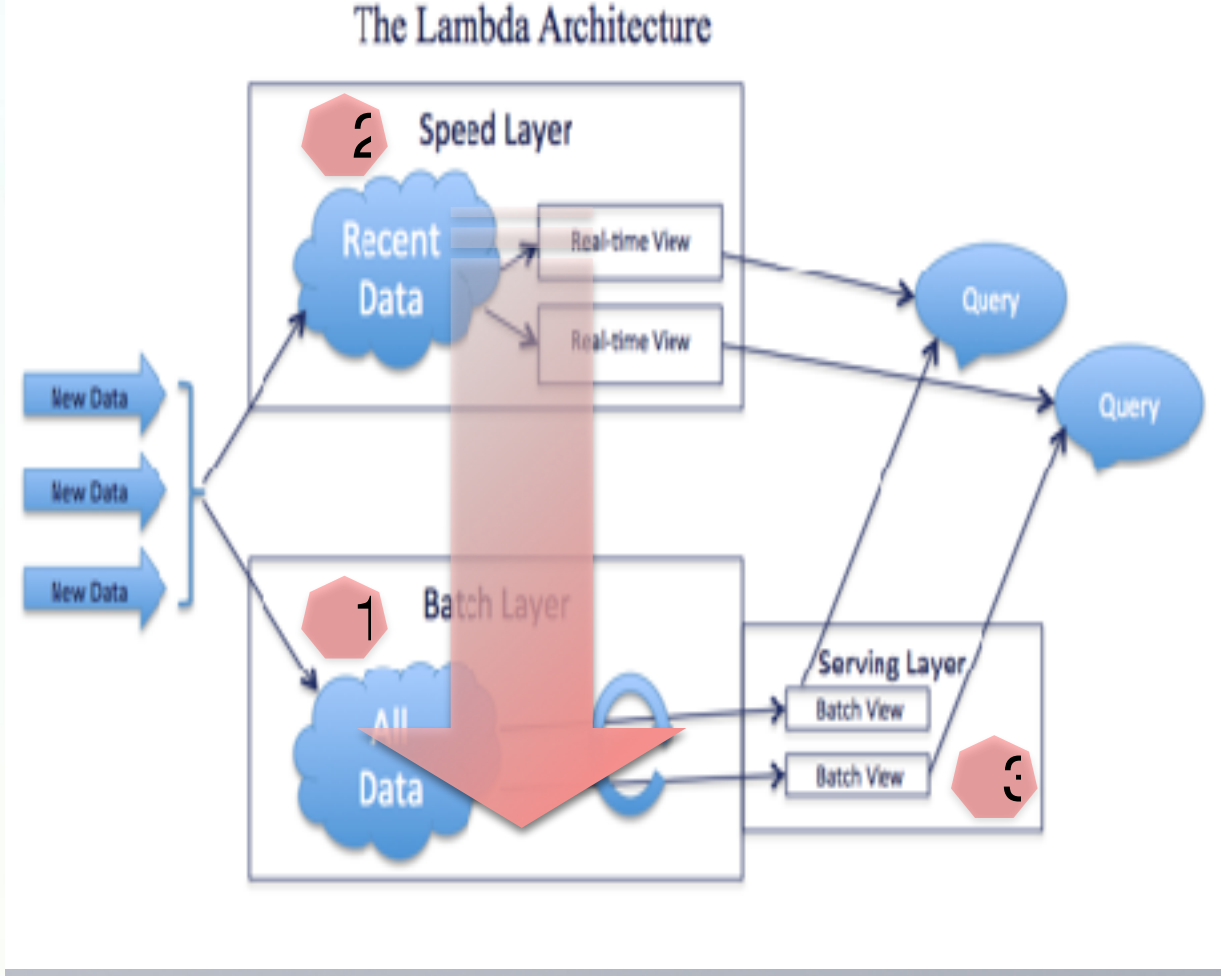
NOT ALL LAMBDAs ARE EQUAL

Greek alphabet needs more letters

- Lambda (λ): an anonymous function (closure)
  - def greeting = { it -> "Hello, $it!" }
    assert greeting('SEC 2017') == 'Hello, SEC 2017!'
- PaaS server-less architecture (AWS Lambda and alike)
  - exports.handler = function (event, context) {
    context.succeed('Hello, SEC 2017!');
    };

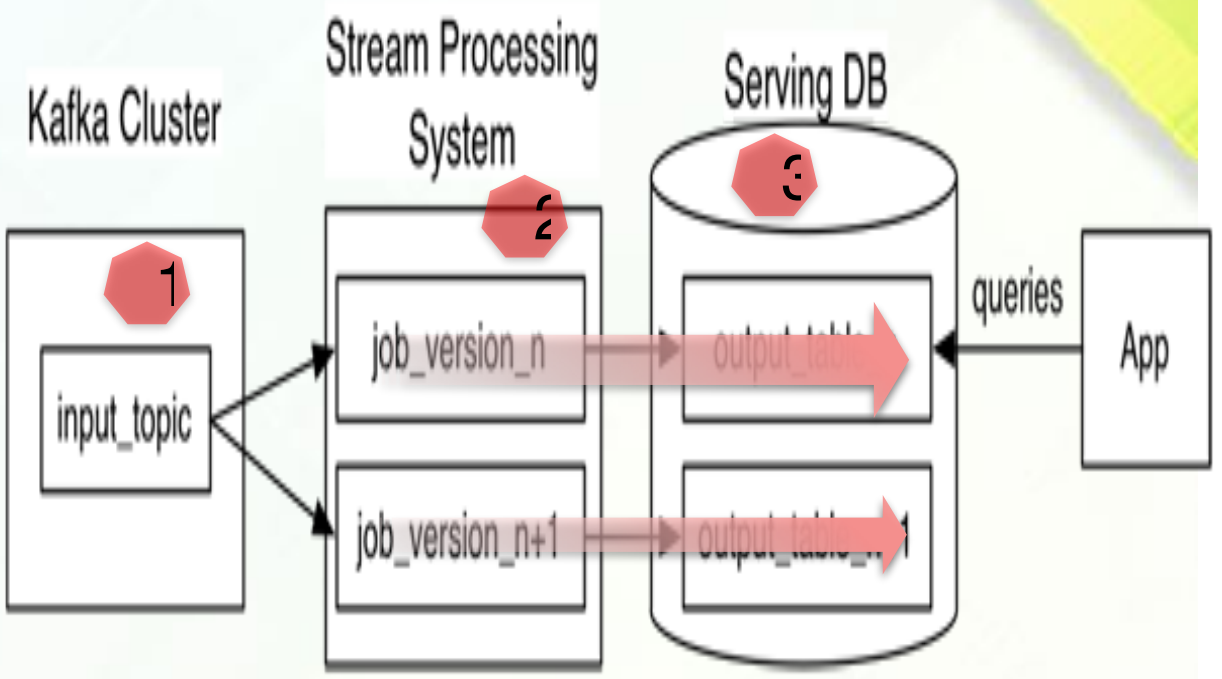
LAMBDA: QUICK OVERVIEW

- Consists of three main layers
  1. High-latency layer for historical
  2. Speed layer for recent/stream data
  3. Smart reconciliation layer
- Properties
  - Immutable, one-way data ingest
- Drawbacks
  - Data accuracy is an issue
  - High operational complexity

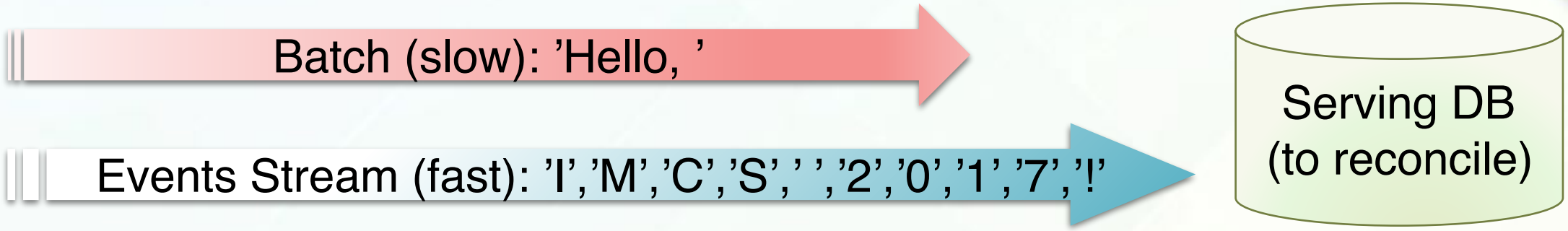


SOME LAMBDAs ARE KAPPAs

- Simplified to
  1. Streaming source
  2. Streaming processing
  3. Stream-only serving DB
- Properties
  - Historical processing is a stream
  - Reprocessing is just a stream job
- Drawbacks
  - (Re)streaming of the historical data on replay
  - Moderate operational complexity

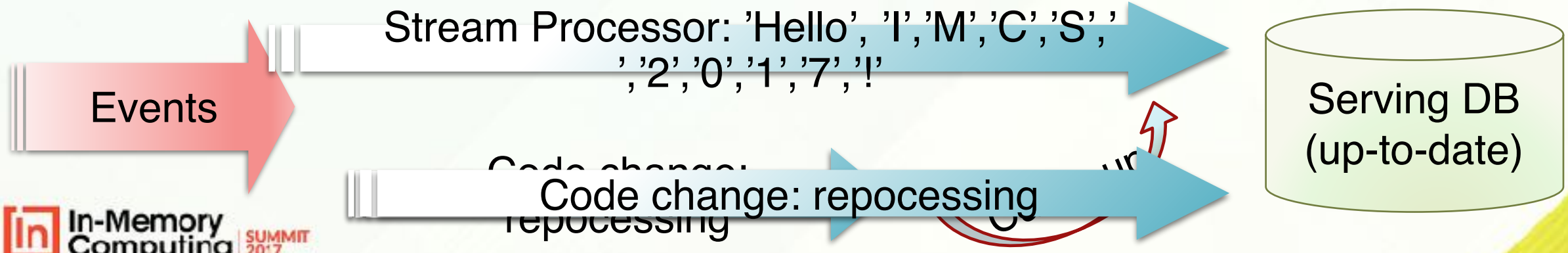


NEXT TO EACH OTHER



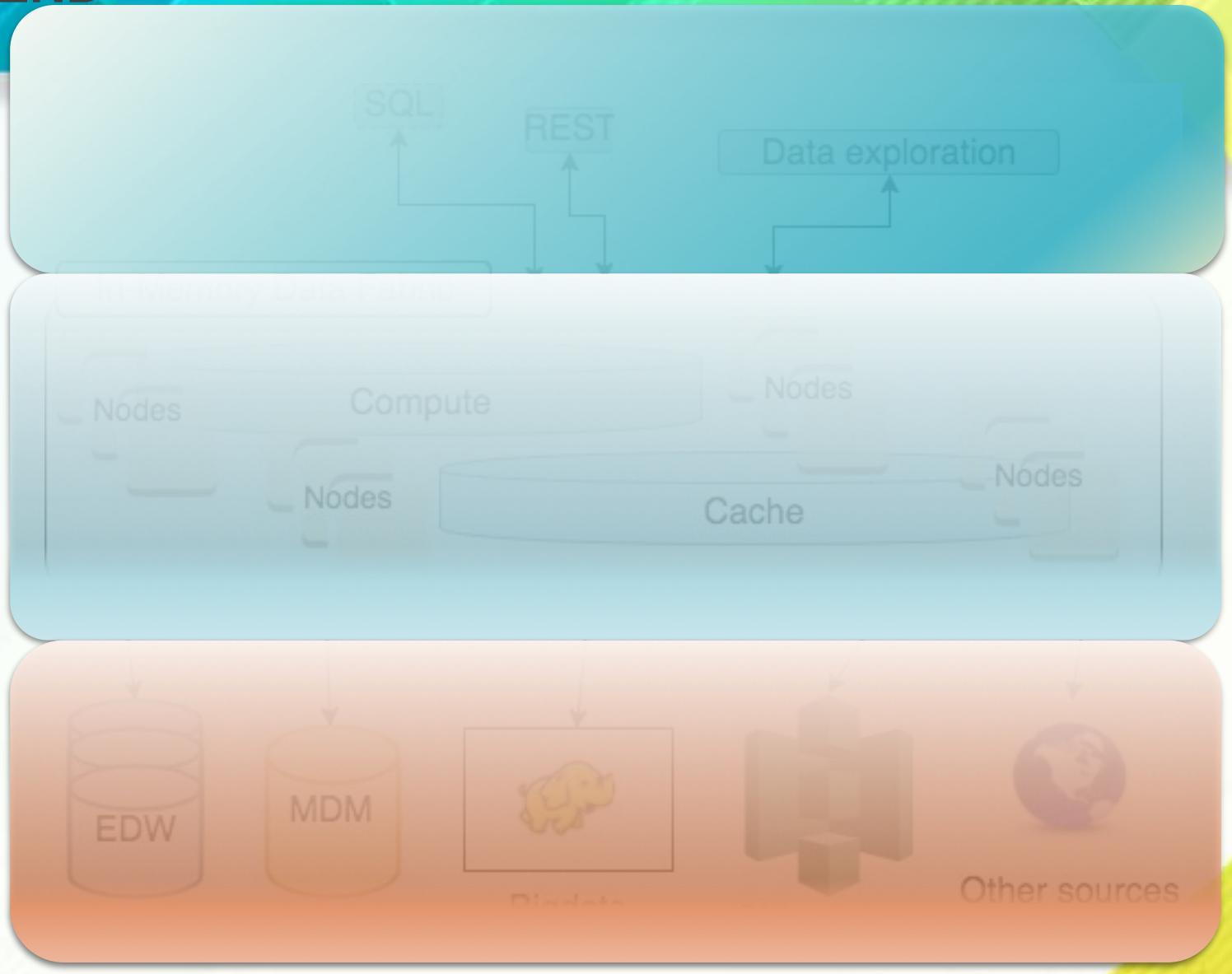
- Processing (Lambda) architecture for slow and fast data

- Some Lambdas are really Kappas



IN-MEMORY DATA FABRIC
PICTURE OR IT NEVER HAPPEND

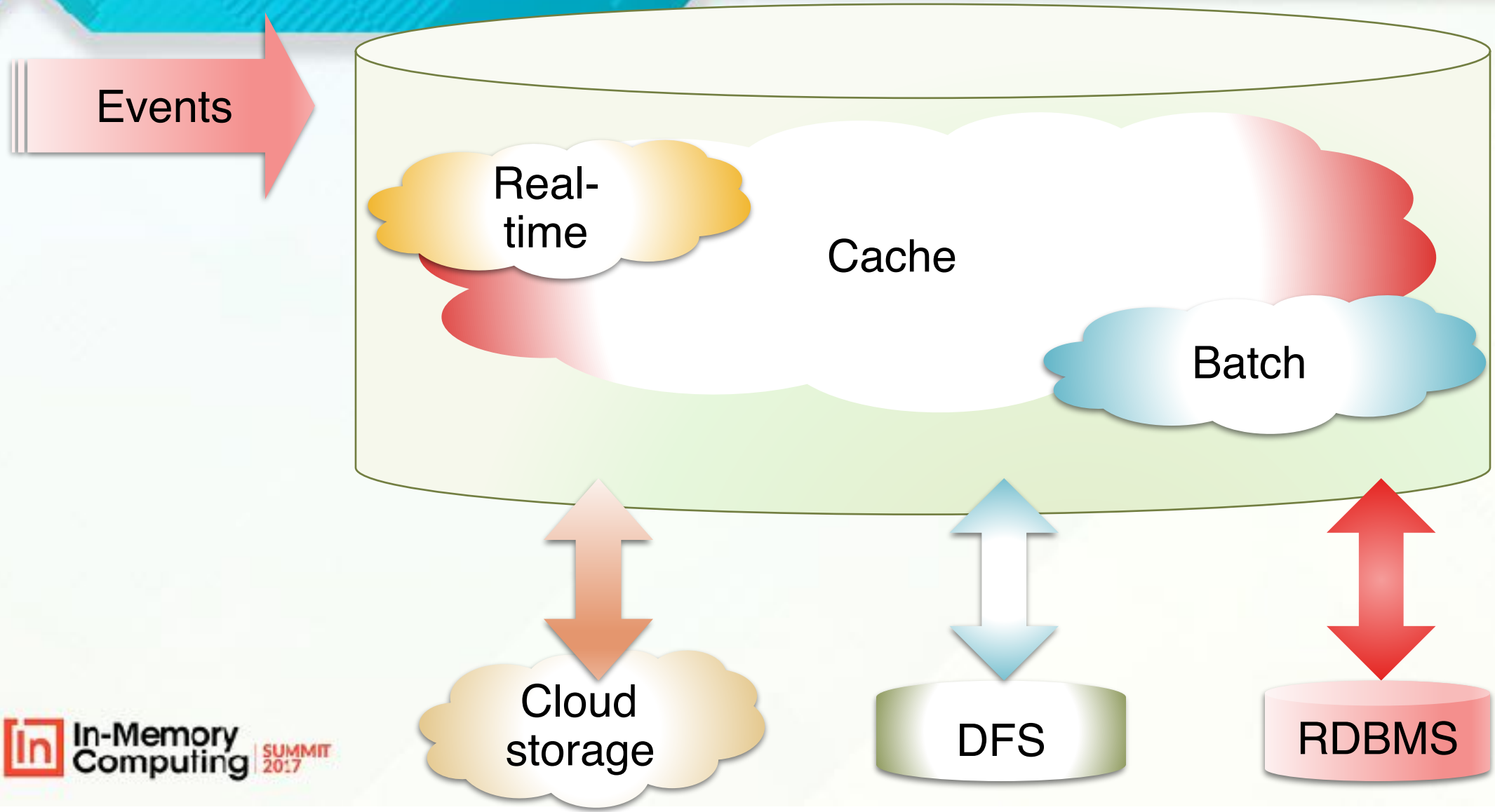
- Separation of concerns
  - Sources
  - Consumers
  - Abstraction and processing



IN-MEMORY DATA FABRIC
IN A NUTSHELL

- Data Fabric is a unified view of data in multiple systems
- A layer for data access
  - Low redundancy; few data movements
  - Write-through caching (might violate legacy app data integrity)
- Affinity sensitive compute medium
- Highly-available and fault tolerant
- Variety of APIs and integration with BigData

NEXT STEP: IOTA
BIGMEMORY



A STEP TOWARDS THE DATA

- Don’t separate batch and stream data processing
- Compute should be co-located with data
- Data mutations have to be tracked (watched and versioned)
- Data concurrency is annoying

ISSUES OF DATA STORING & PROCESSING

- Data state, persistency and immutability
- Misperception of data primacy – what is the main copy?
- Versioning of data, data structures, code and metadata
- Uniform data access, Multi-structured data
- Granular data access rights and security
- ETL/ELT & Data Marts, Data lifecycle

TWO BREEDS OF DATAWAREHOUSES

Update-Driven

- Provides higher performance
- Integrates Data from heterogeneous sources
- Simplifies analyses: Data are ready for direct querying
- Extra storage for copied data
- Complex CDC for each data source

Heterogeneous Query-Driven

- Builds wrappers/mediators on top of heterogeneous databases
- Translates query to data-source specific
- Single-Source-of-Truth practice
- Complex information filtering
- Massive data pull from data sources

BIGDATA & QUERY-DRIVEN WAREHOUSE

- Query-Driven Warehouse borrowed from BigData:
  - On demand extraction from schema-on-read data
  - Avoids complex ETLs
- BigData addresses high query costs of Query-Driven Warehouse:
  - Read less data: partitioning
  - Lesser shuffle: share nothing, collocation, local filtering (pushdown)
- Requires sophisticated extendable metadata

TWO BREEDS OF DATA
PRIMARY & DERIVED

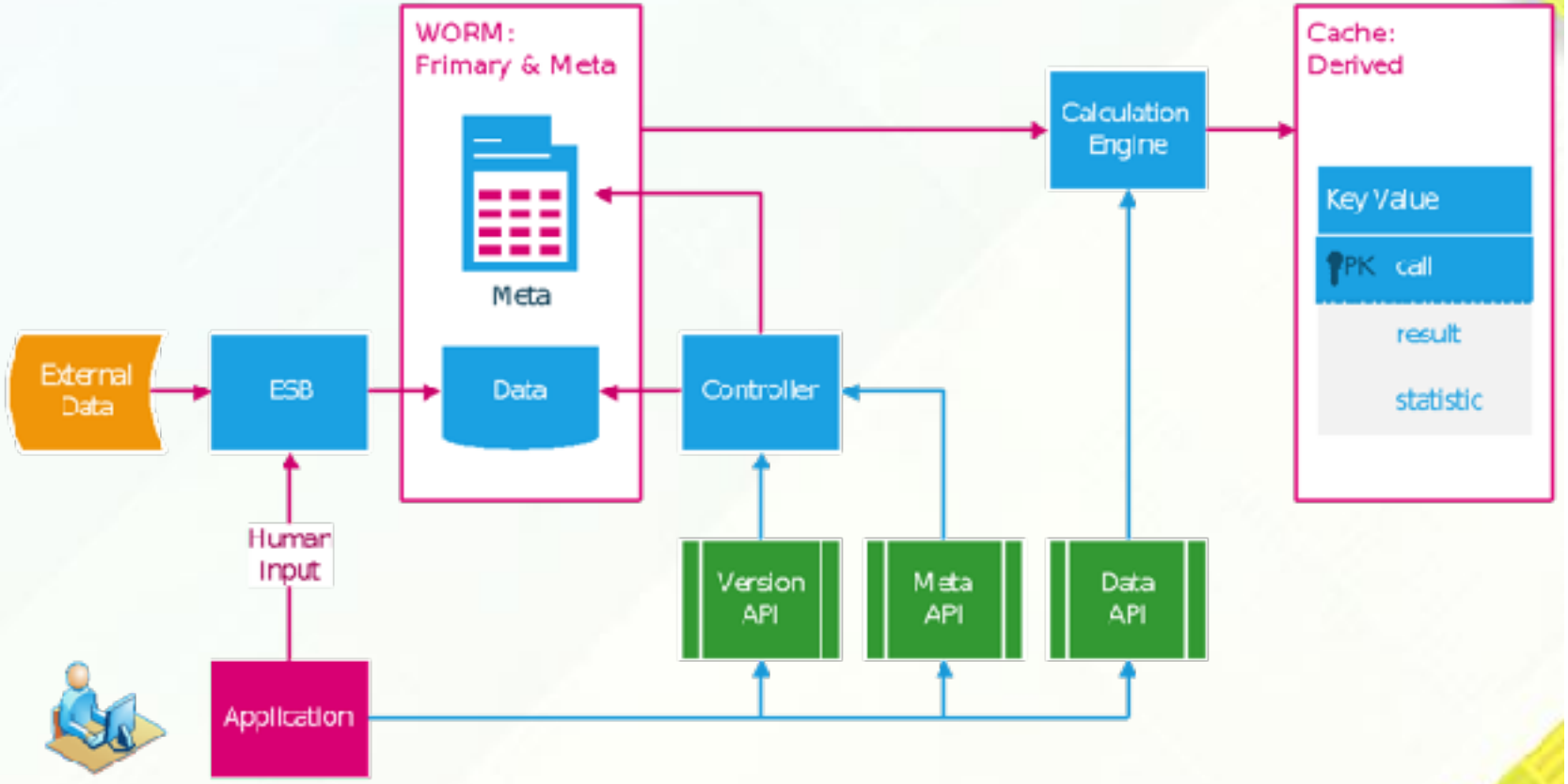
- Primary Data are nondeterministic, non-reproducible and UNIQUE
  - persistent and immutable
- Derived Data are deterministic and reproducible EXACTLY
  - ephemeral and immutable
- Versioned metadata are Primary by its nature
  - persistent and immutable
- Versioned Code is Primary by its nature
  - persistent and immutable
- All abovementioned are immutable and therefor, STATELESS!

BENEFITS OF STATELESSNESS

- No data concurrency issues
  - Majority of transactions are RAMP
- Leveraging functional programming paradigm (lambda again!)
  - Read-through & memoization
  - Higher re-use of the code
- Avoiding complex ETLs
- On-demand extraction from schema-on-read data

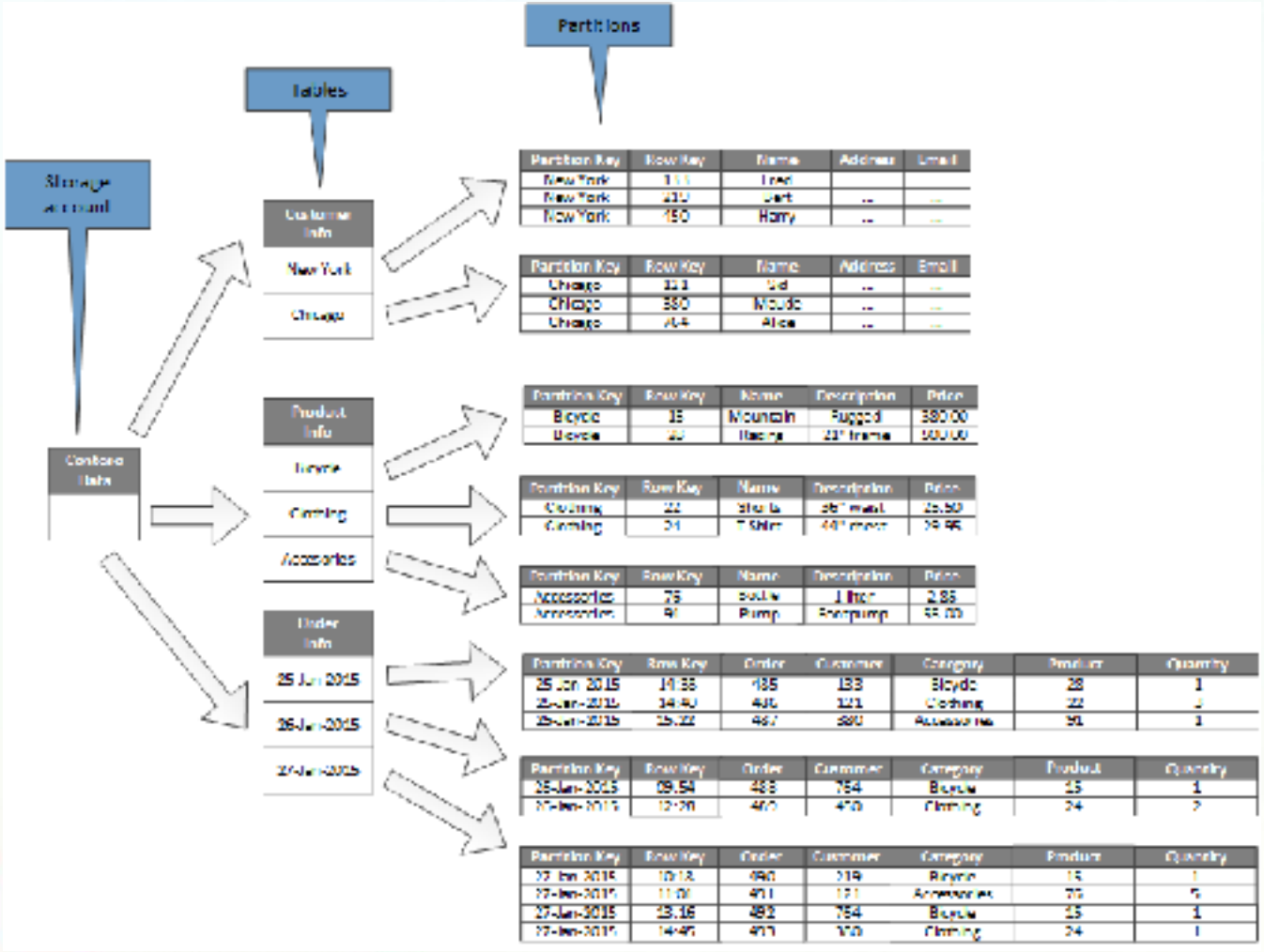
MOVING PARTS

- Persistent WORM stores (Write Once Read Many)
  - Primary data
  - Metadata & Code
- Transient Cache stores
  - Derived data
- Compute Engine
  - Reads WORM & Cache
  - Produces results
  - Puts results to Cache



PARTITIONING VS PATCHWORK

HOW TO READ LESS



- Partitions: statically defined in DDL
- Patchworks: arbitrary structure of dynamically built patches

PATCHWORK
DATA BLOCKS & DATA CATALOG

- **Data Blocks:**
  - Describe a quantum of data
  - A set of semantically similar objects, limited by some dimensions
  - A URI: ftp, web, files, a parametrized SQL SELECT
- **Data Catalog:**
  - A part of versioned metadata
  - Organizes Data Blocks into a Patchwork
  - Is a functional equivalent of RDBMS catalog

CACHE

- Cache is transparent and transient by its nature:
  - Holds function results, instead of actual calls
  - Might hold Data Blocks
- Cache Entry includes Key, Value, and Statistics:
  - last time value was accessed and how often (frequency)
  - dependency depth
  - resources spent, like CPU and IOs
- Retention & Eviction:
  - Is based on Cache Entry statistics
  - The dependency graph' Data Blocks are evicted with root entry

MISCELLANEOUS ASPECTS

- Dependency graph is built from data access' history:
  - Could be replaced by a reference to Data Block (compacted)
- Invalidation & Lineage is driven by dependency graph
- Functions: follow memoization pattern
- Scalability – just put more boxes there, if:
  - WORM uses distributed Key-Value storage
  - Cache & Calculation engine use In-Memory Data Fabric

USE CASES

- **Better data lakes**: bi-directional data movements
  - Minimal networking, Memory-centric, Integration with legacy
- **Real-time personalization**
  - Better shopping with mobile devices, Location-based marketing
  - Near real-time promotions, Advanced analytics
  - Simplified ML-driven CEP
- **Fraud detection**
  - Discovery of complex fraud patterns, based on historical data
  - Real-time detection of abnormal behavior
  - Simplified ML-driven CEP

IOTA BENEFITS

- Avoiding multiple copies of the data, instant consistency
- In-memory caching with read-ahead/write-behind support
- Batch, streaming, CEP, and (near) real-time processing
- Speeding up a traditionally slow, batch oriented frameworks
- Variety of data processing: read-only, read-write, transactional
- Lower inter-component impedance

Q & A