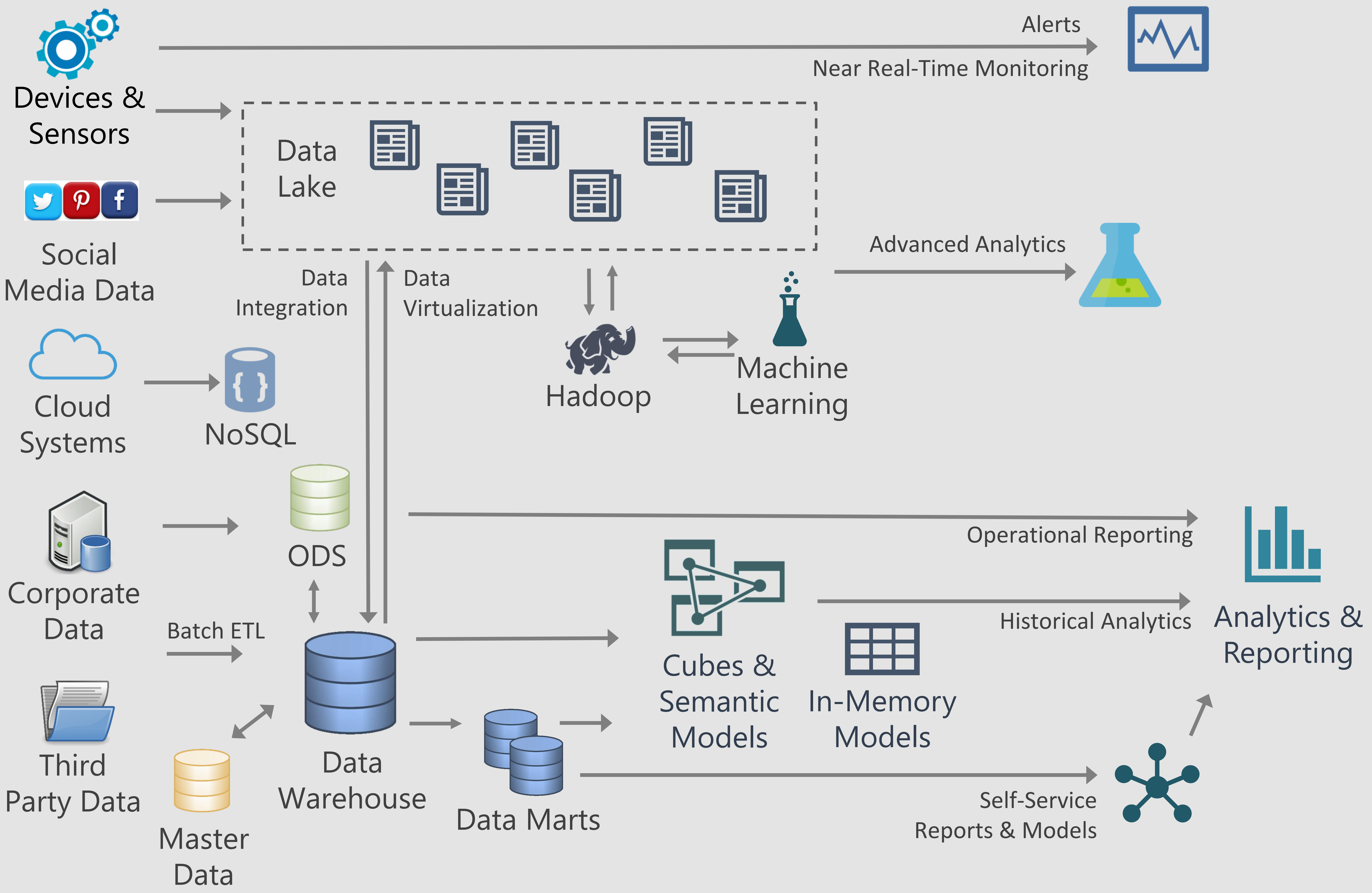
Architecting a Data Lake

Chad Gronbach
Chief Technology Architect
Microsoft Technology Center - Boston

Content Credit: JamesSerra.com

Modern
Multi-Platform Architectures

Modern Data Warehousing & Analytics



Multi-platform architecture

- ✓ Handle a variety of data types & sources
- ✓ Larger data volumes at lower latency
- ✓ Bimodal: self-service + corporate BI to support all types of users
- ✓ Newer cloud services
- ✓ Advanced analytics scenarios
- ✓ Balance data integration & data virtualization

Definitions

Data Warehouse	Repository of data from multiple sources, cleansed & enriched for reporting; generally 'schema on write'
Data Lake	Repository of data for multi-structured data; generally 'schema on read'
Hadoop	(1) Data storage via HDFS (Hadoop Distributed File System), and (2) Set of Apache projects for data processing and analytics
Lambda Architecture	Data processing & storage with batch, speed, and serving layers
ETL	Extract > Transform > Load: traditional paradigm associated with data warehousing and 'schema on write'
ELT	Extract > Load > Transform: newer paradigm associated with data lakes & 'schema on read'
Semantic Model	User-friendly interface for users on top of a data warehouse and/or data lake

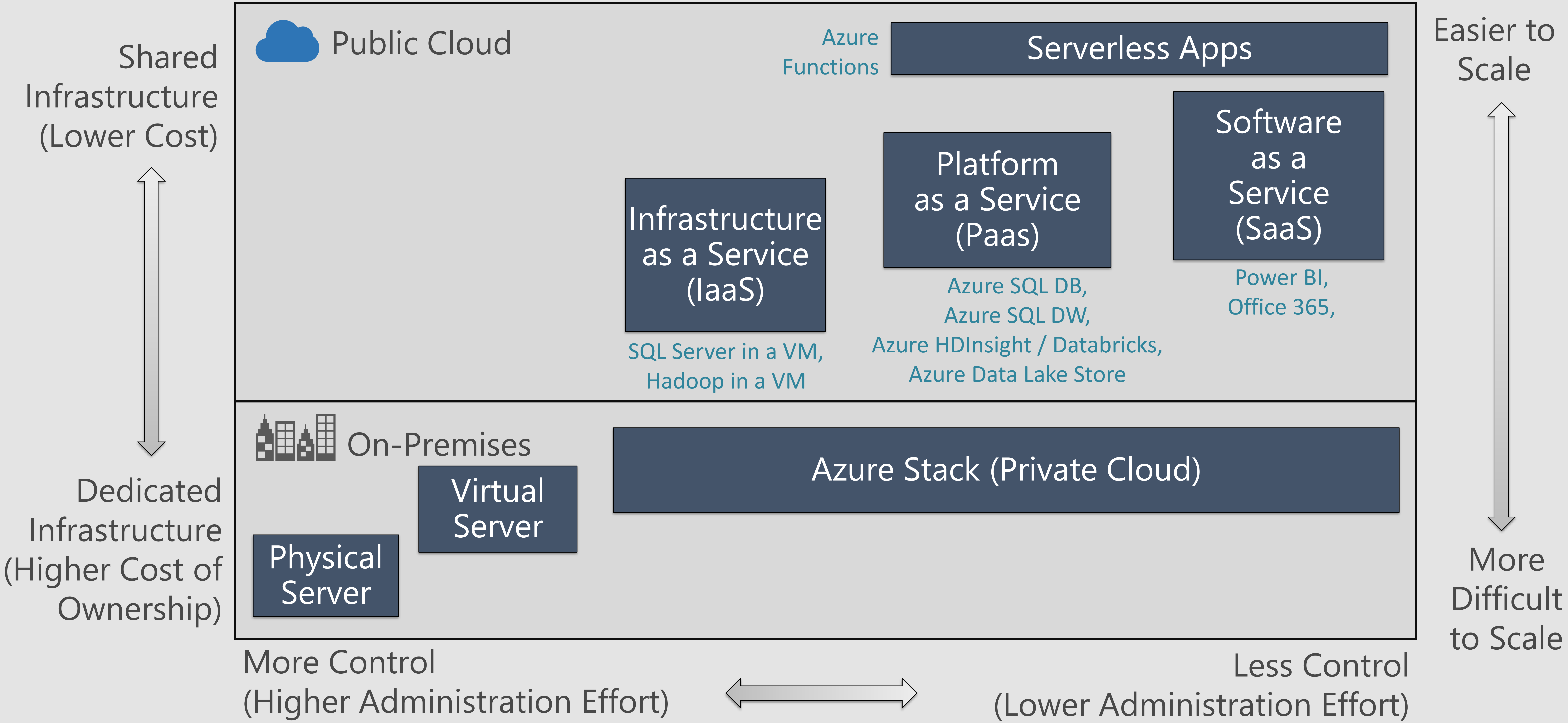
Definitions

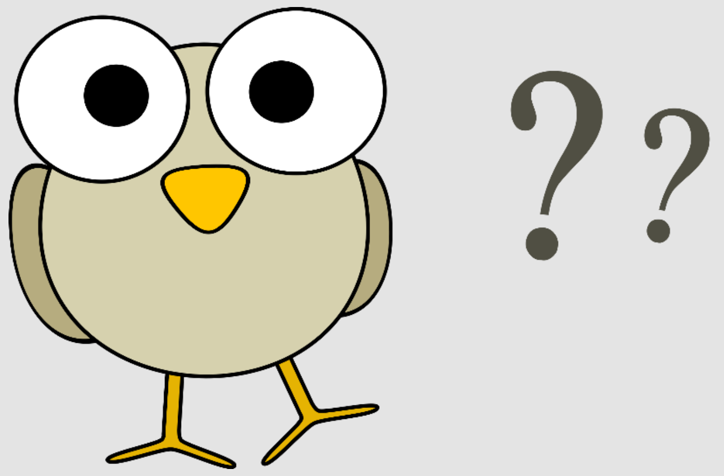
Data Integration	Physically moving data to integrate multiple sources together
Data Virtualization	Access to one or more distributed data sources without requiring the data to be physically materialized in another data structure
Federated Query	A type of data virtualization: access & consolidate data from multiple distributed data sources
Polyglot Persistence	A multi-platform strategy which values using the most effective technology based on the data itself ("best fit engineering")
Schema on Write	Data structure is applied at design time, requiring additional up-front effort to formulate a data model (relational DBs)
Schema on Read	Data structure is applied at query time rather than when the data is initially stored (data lakes, NoSQL)

More in-depth definitions: https://www.sqlchick.com/entries/2017/1/9/defining-the-components-of-a-modern-data-warehouse-a-glossary

Definitions

(3/3)





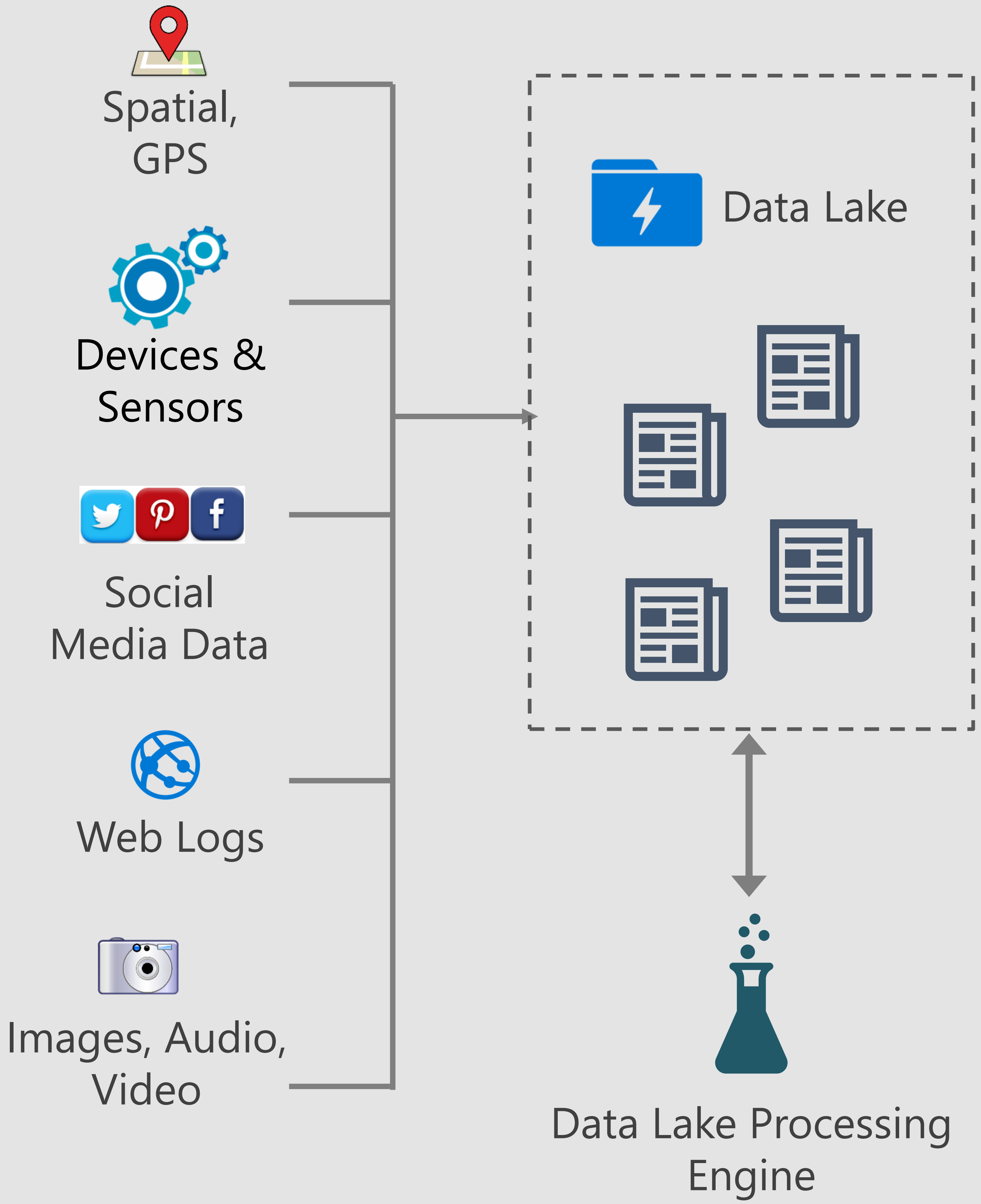
What are some common challenges of analytical environments?

Challenges of Analytical Environments

Agility	Complexity	Balance	Never-Ending
✓ Reducing time to value	✓ Hybrid scenarios	✓ Self-service solutions challenge corporate DW solutions	✓ Data quality
✓ Minimizing chaos with self-service	✓ Multi-platform architecture	✓ Operationalizing valuable user-created solutions (including data science)	✓ User trust
✓ Evolving & maturing technology	✓ Ever-increasing data volumes	✓ Handling ownership changes of a productionized solution	✓ Master data
✓ Balancing schema-on-read with schema-on-write	✓ Diversity of file types & formats		✓ Security
✓ How strict to be with dimensional design?	✓ Effort & cost of data integration		✓ Governance
	✓ Many skillsets needed		✓ Performance

Data Lake Overview
&
Use Cases

Data Lake



A A repository for storing large quantities of disparate sources of data in its native format

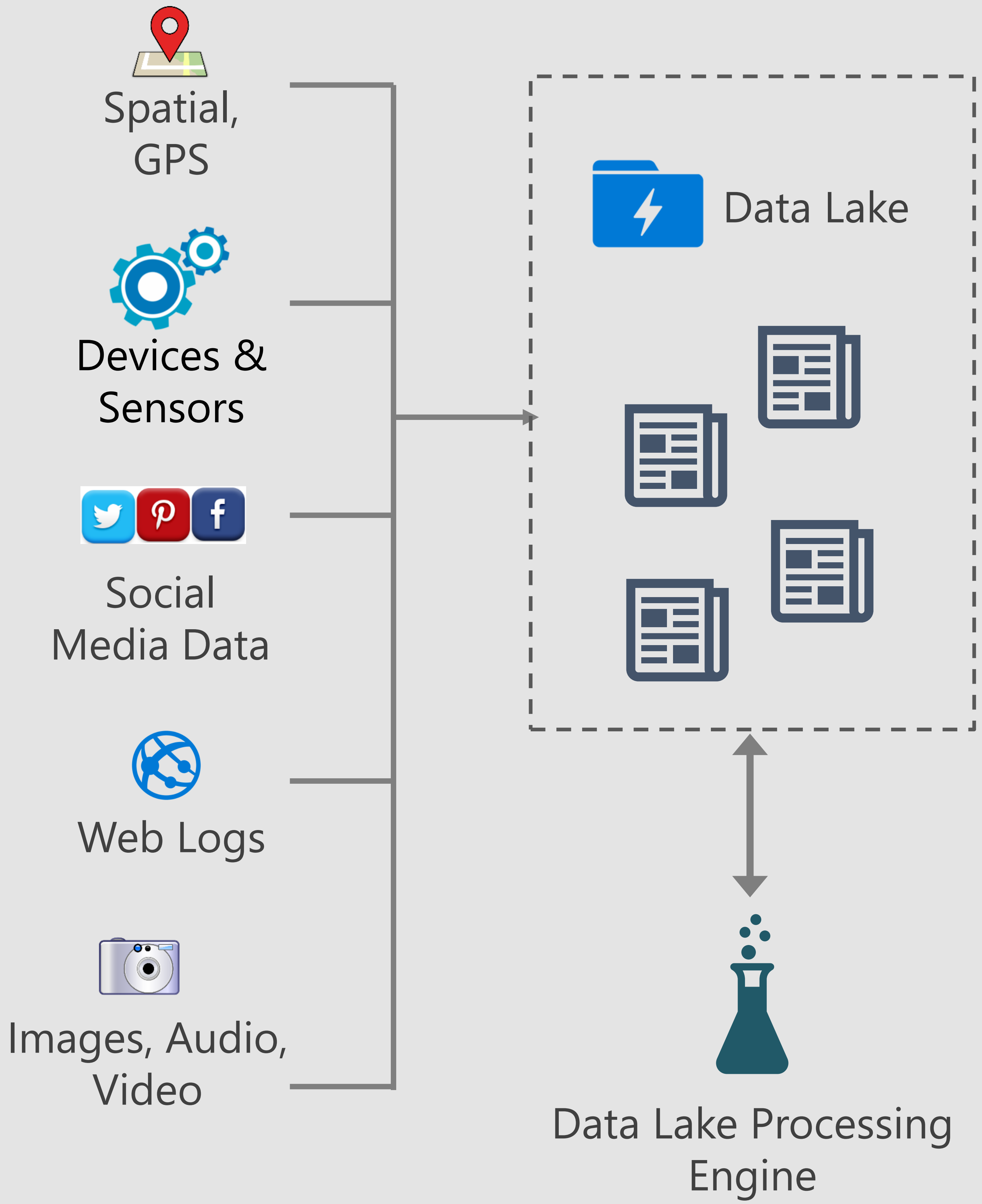
One architectural platform to house all types of data:

- ✓ Machine-generated data (ex: IoT, logs)
- ✓ Human-generated data (ex: tweets, e-mail)
- ✓ Traditional operational data (ex: sales, inventory)

B A processing engine for analyzing data

Data Lake Objectives

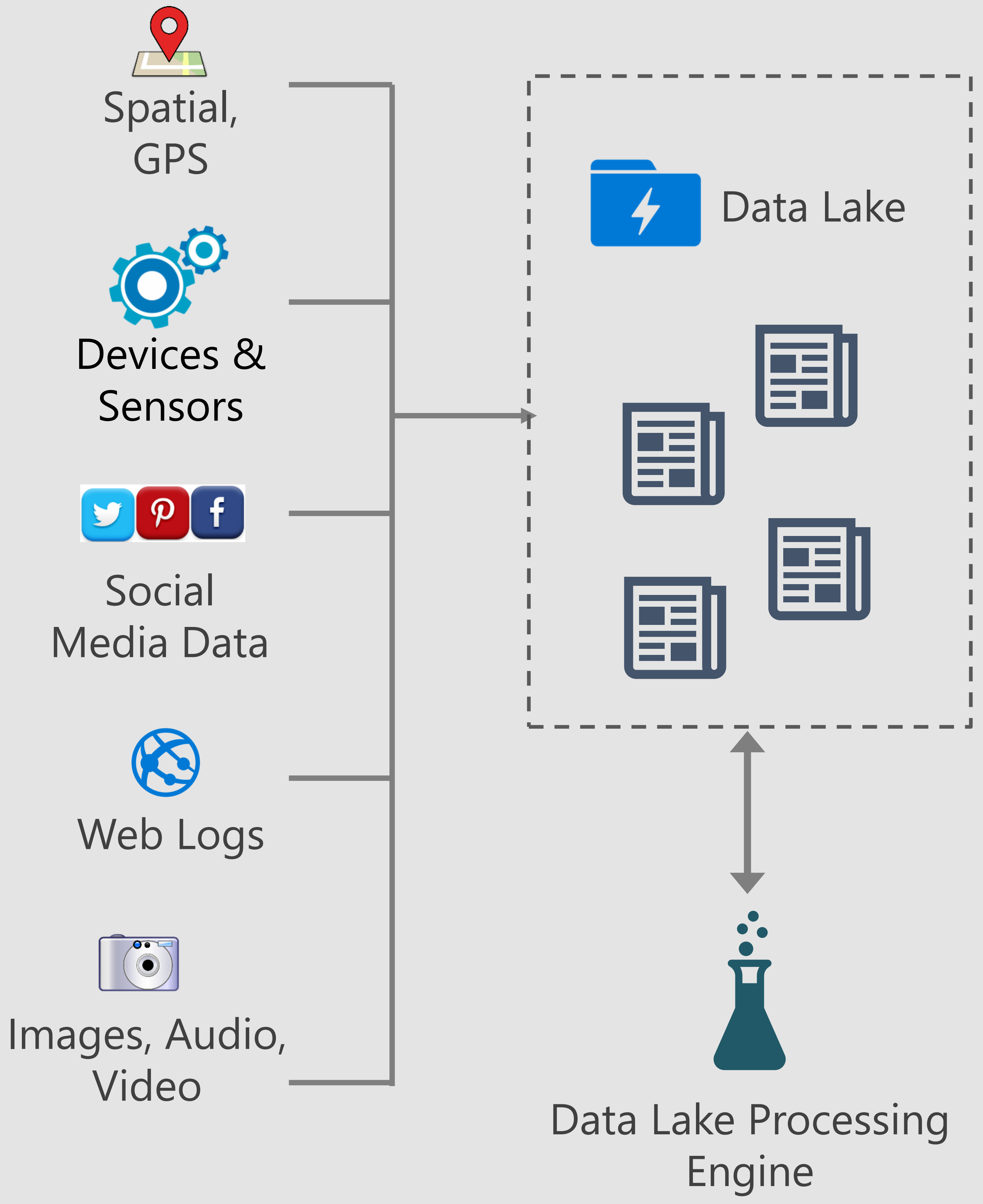
- ✓ Reduce up-front effort by ingesting data in any format, any size, without requiring a schema initially
- ✓ Make acquiring new data easy, so it can be available for data science & analysis quickly
- ✓ Store large volume of multi-structured data in its native format
- ✓ Storage for additional types of data which were historically difficult to obtain or store
- ✓ Reduce the long-term ownership cost of data management & storage



Data Lake Objectives

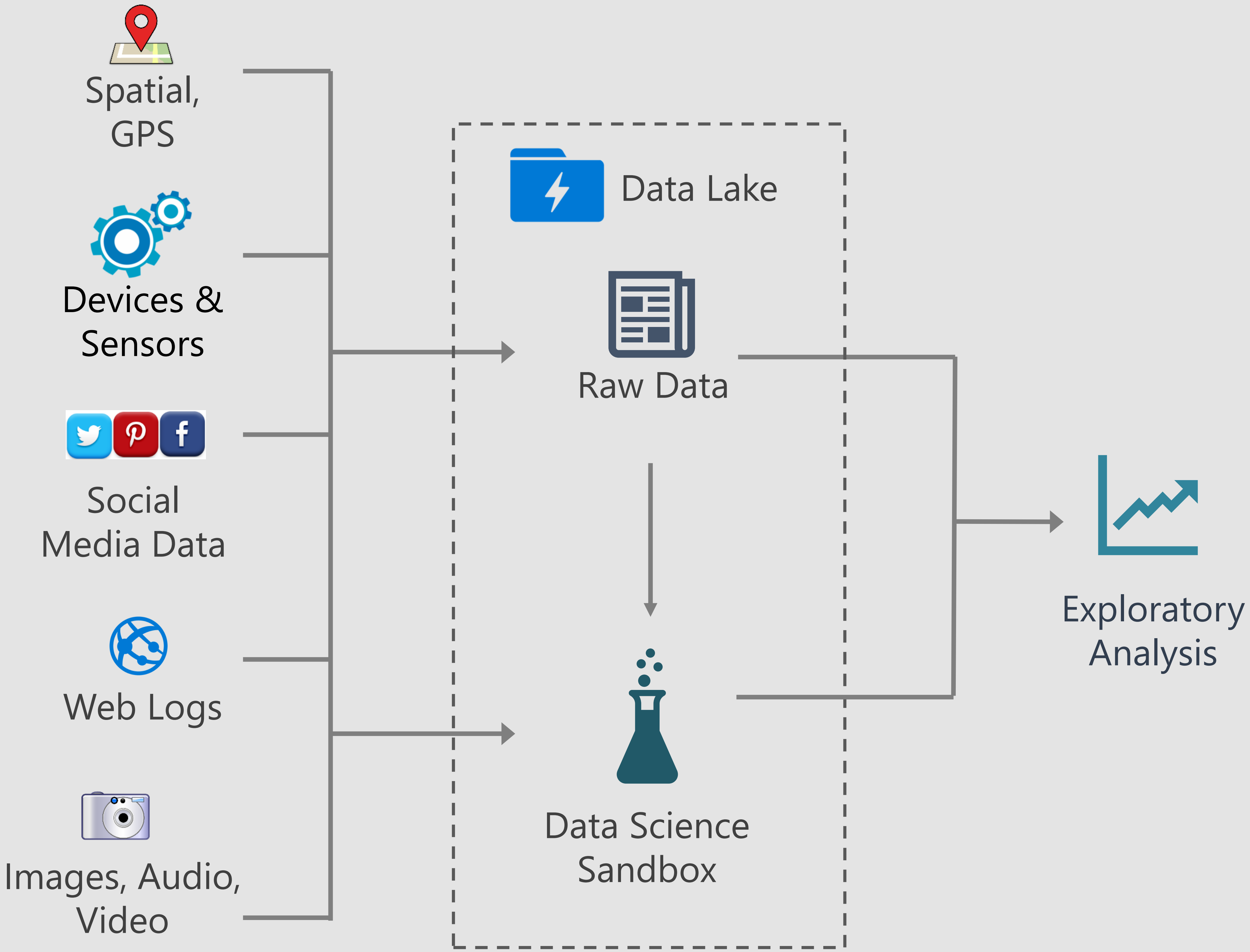
(2/2)

- ✓ Schema-on-read: Defer work to ‘schematize’ after value & requirements are known
- ✓ Achieve agility faster than a traditional data warehouse can to speed up decision-making ability
- ✓ Access to low-latency data
- ✓ Different / new value proposition vs. traditional data warehousing
- ✓ Facilitate advanced analytics scenarios



Data Lake Use Cases

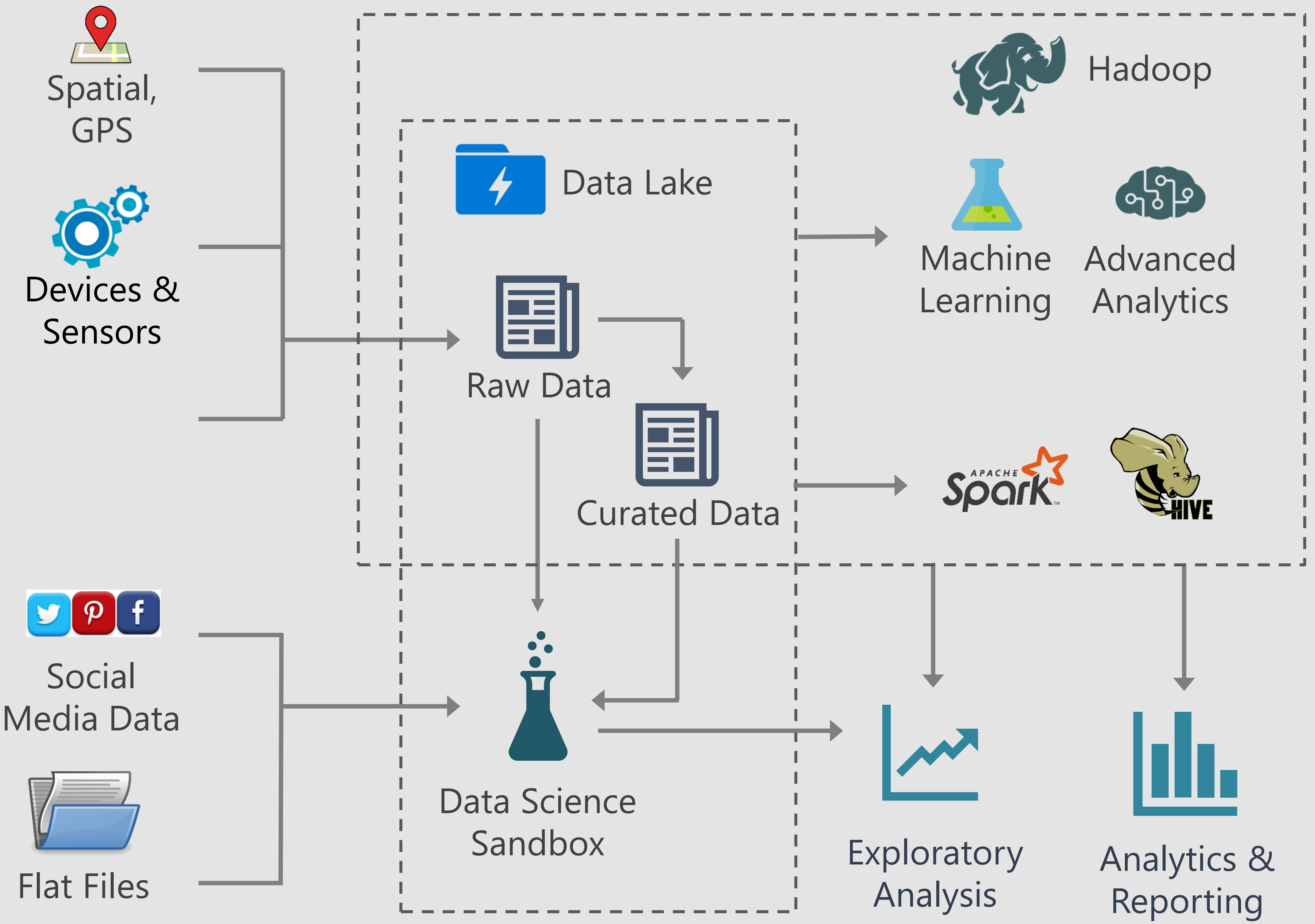
Ingestion of New File Types



- ✓ Preparatory file storage for multi-structured data
- ✓ Exploratory analysis + POCs to determine value of new data types & sources
- ✓ Affords additional time for longer-term planning while accumulating data or handling an influx of data

Data Lake Use Cases

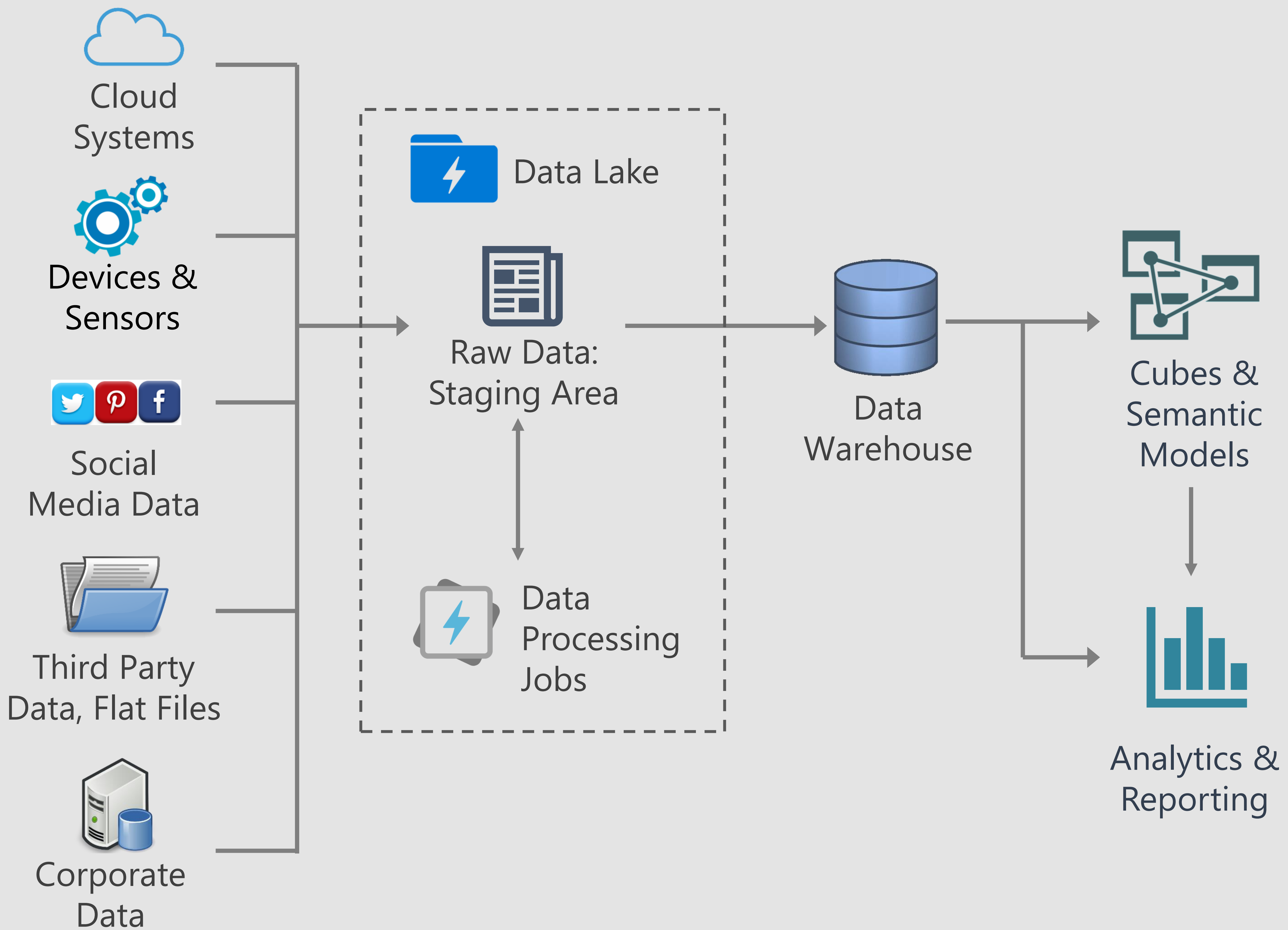
Data Science Experimentation | Hadoop Integration



- ✓ Sandbox solutions for initial data prep, experimentation, and analysis
- ✓ Migrate from proof of concept to operationalized solution
- ✓ Integrate with open source projects such as Hive, Pig, Spark, Storm, etc.
- ✓ Big data clusters
- ✓ SQL-on-Hadoop solutions

Data Lake Use Cases

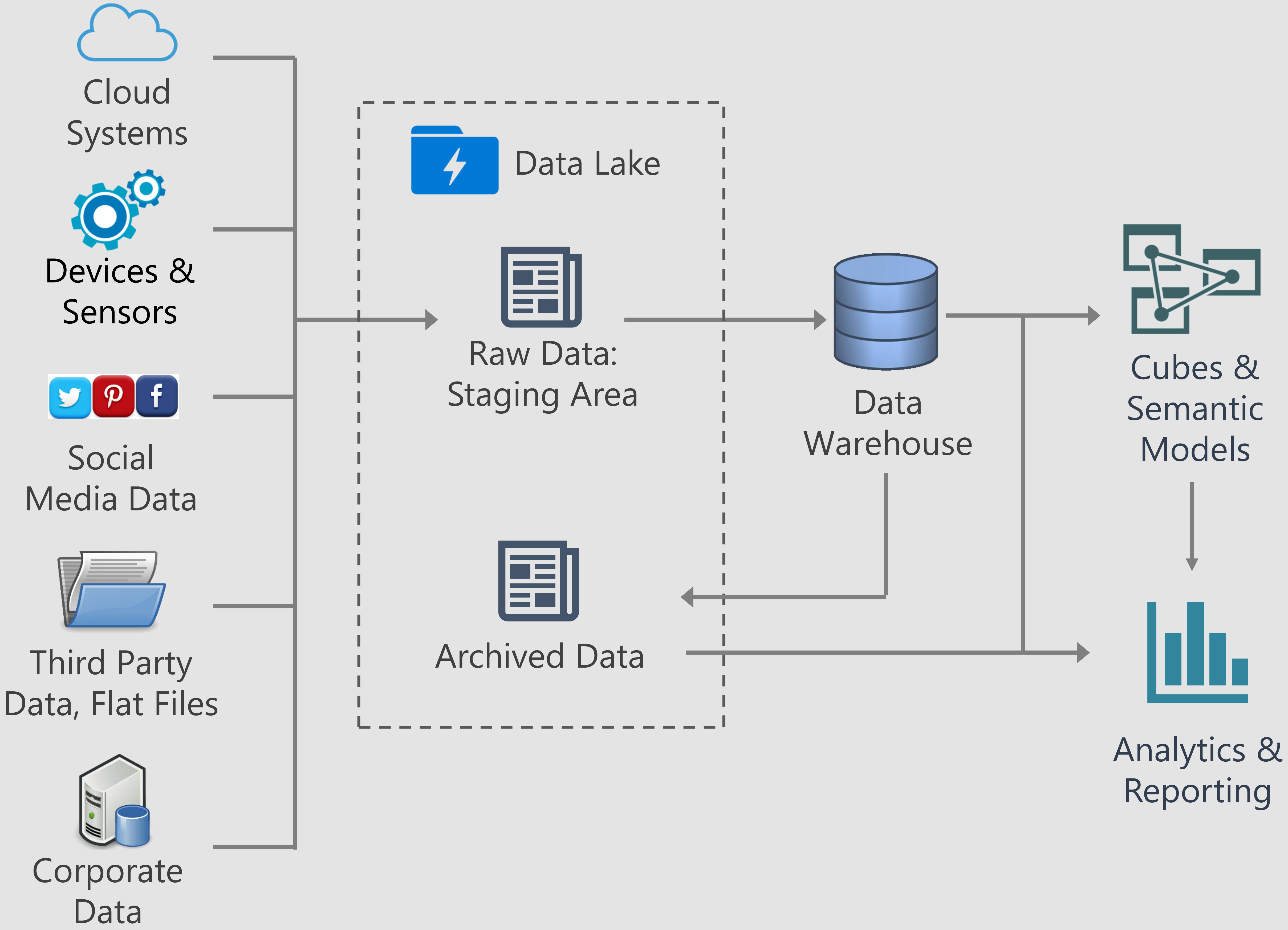
Data Warehouse Staging Area



- ✓ ELT strategy
- ✓ Reduce storage needs in relational platform by using the data lake as landing area
- ✓ Practical use for data stored in the data lake
- ✓ Potentially also handle transformations in the data lake

Data Lake Use Cases

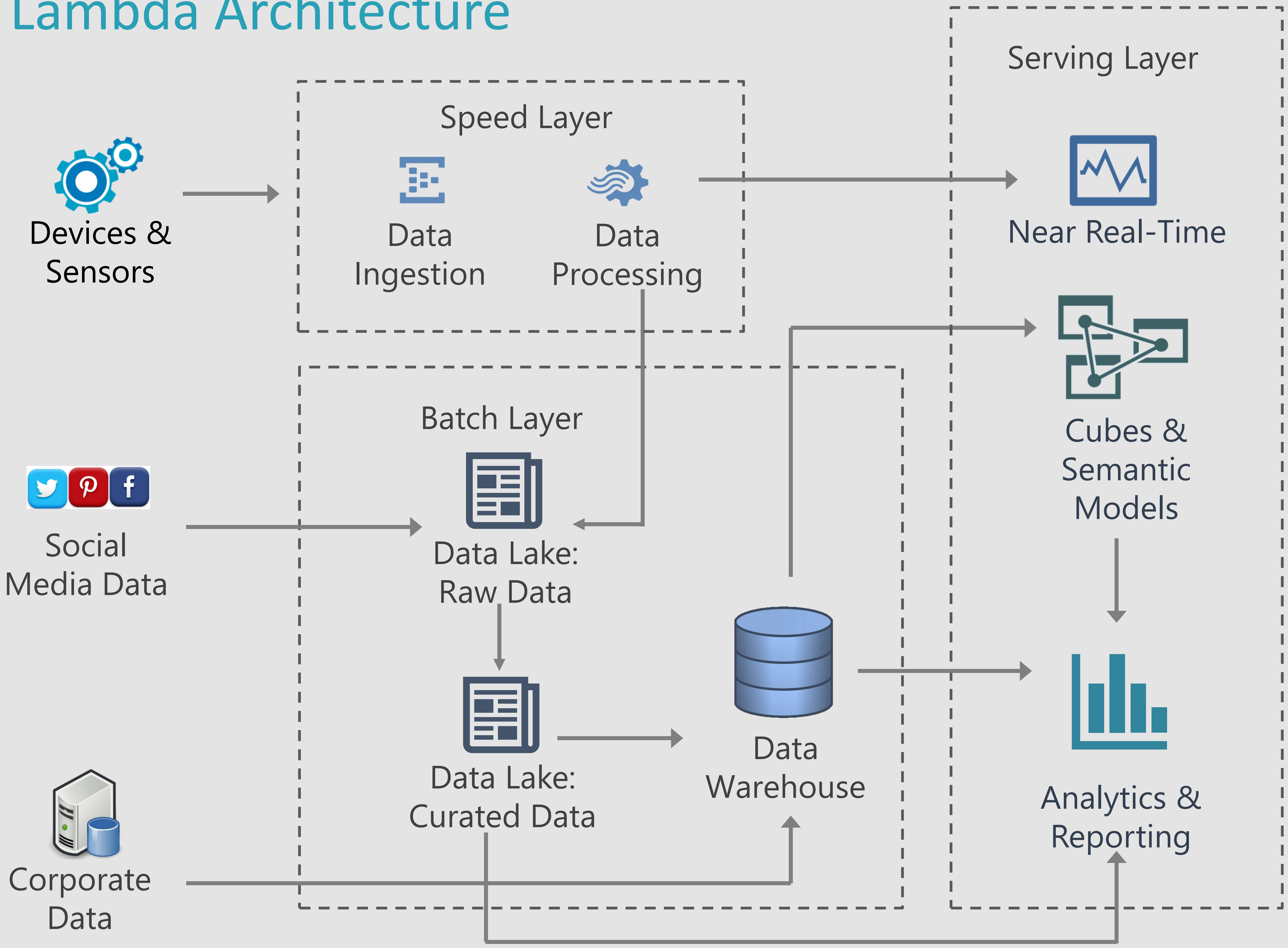
Integration with DW | Data Archival | Centralization



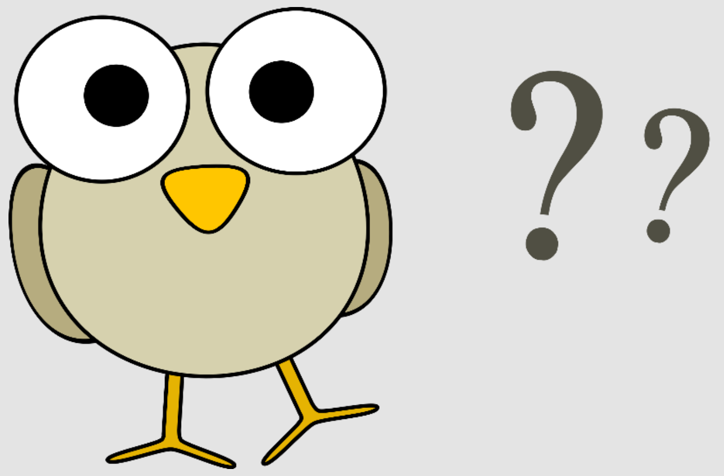
- ✓ Grow around existing DW
- ✓ Aged data available for querying when needed
- ✓ Complement to the DW via data virtualization
- ✓ Federated queries to access current data (relational DB) + archive (data lake)

Data Lake Use Cases

Lambda Architecture



- ✓ Support for low-latency, high-velocity data in near real time
- ✓ Support for batch-oriented operations



What are some initial considerations for deciding if a data lake is right for you?

Is a Data Lake Right For You?

Initial Considerations:

Do you have *non-relational* data?

Do you have *IoT* type of data?

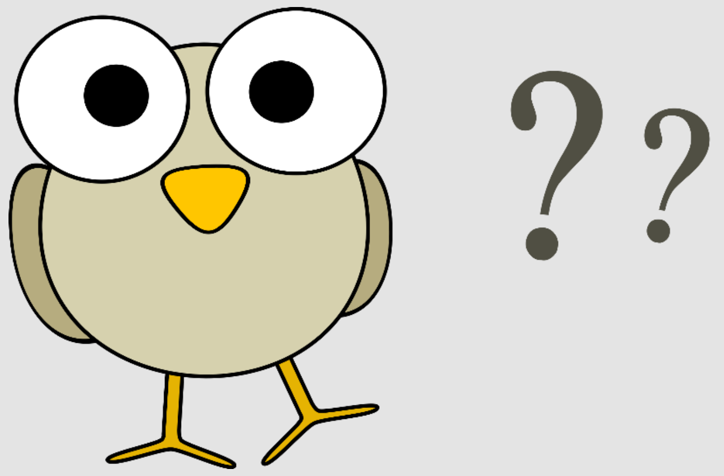
Do you have *advanced analytics scenarios* on unusual datasets?

Do you need to *offload ETL processing (ELT)* and/or *archival data* from a data warehouse?

Readiness:

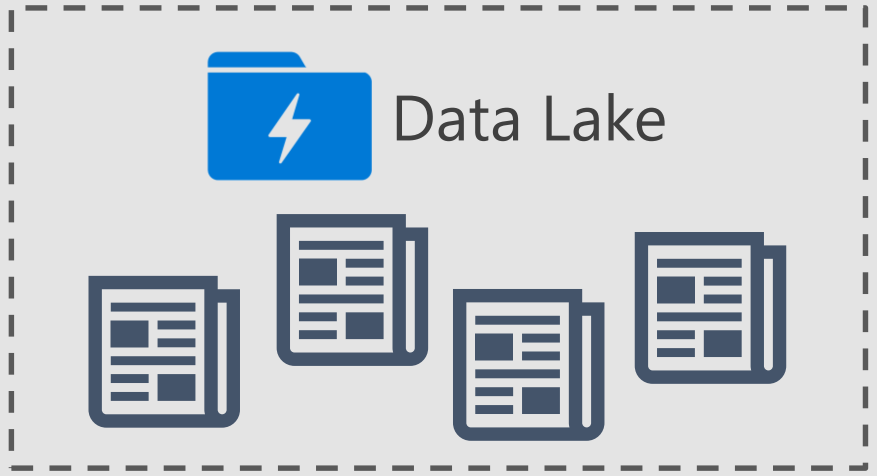
Are you ready willing to learn *different development patterns* and/or *new technologies*?

Are you ready to handle the *trade-offs of 'schema on read'* vs 'schema on write'?



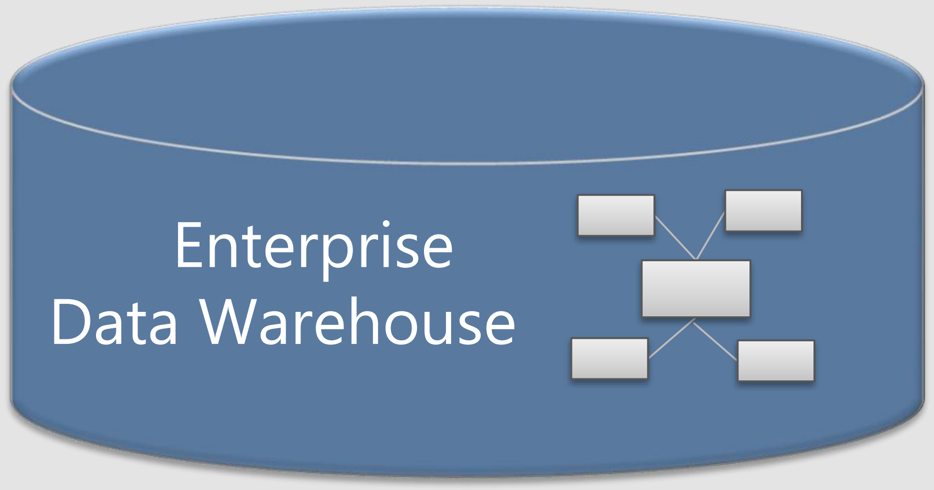
What are some key differences between a data warehouse & a data lake?

Data Lake + Data Warehouse: Inverse Relationship



Data Lake focuses on:

- ✓ Agility
- ✓ Flexibility
- ✓ Easy data acquisition
- ✓ Early exploration activities



Data warehouse focuses on:

- ✓ Cleansed, user-friendly data
- ✓ Reliability
- ✓ Standardization
- ✓ Process-oriented operationalization

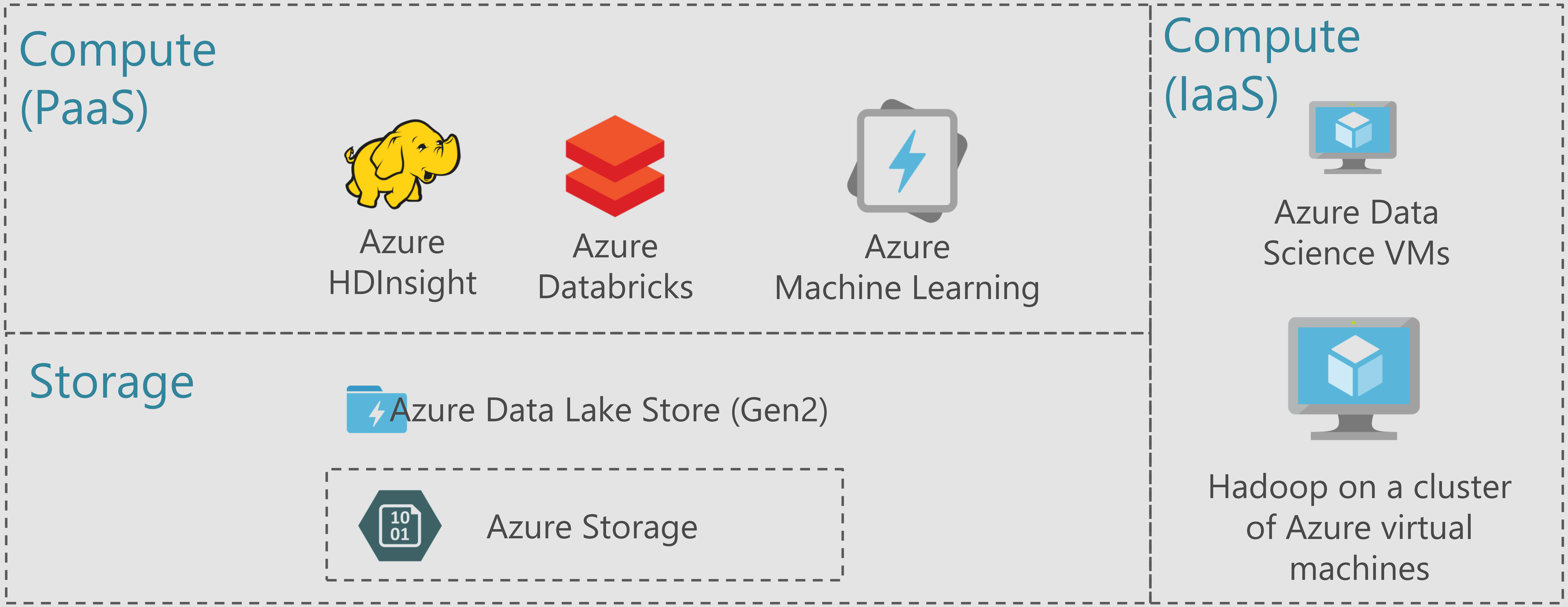
Schema on Read		**Schema on Write**
⇩ Less effort	**Data acquisition**	⇧ More effort
⇧ More effort	**Data retrieval**	⇩ Less effort

Data Lake Challenges

Technology	Process	People
✓ Addt'l component(s) in a multi-layered architecture	✓ Right balance of deferred work vs. up-front work	✓ Expectations & trust
✓ Unknown storage & scalability	✓ Ignoring established best practices for data management	✓ Data stewardship
✓ Data retrieval	✓ Data quality	✓ Redundant effort
✓ Working with un-curated data	✓ Governance	✓ Skills required to effectively use the data
✓ Performance	✓ Security	
✓ Change management	✓ Disaster recovery for large solutions	

Big Data in Azure

Big Data in Azure



Big Data in Azure: Compute

Higher level of complexity, control, & customization

Greater ease of use

Less administrative effort

Azure Machine Learning

Azure Databricks (PaaS)

Azure HDInsight (PaaS)

Hadoop / Data Science Azure virtual machines (IaaS)

Greater administrative effort

Greater integration with Apache projects

Less integration with Apache projects

Deciding Between Compute Services

	Hadoop VM	HDInsight	Databricks	Azure ML
Type:	IaaS	PaaS	PaaS	SaaS
Purpose:	Running your own cluster of Hadoop virtual machines	Running a managed cluster	Running optimized Spark framework	Running packaged AI, R or Python Script
Suitable for:	Full control over everything; investment in distributions such as Hortonworks, Cloudera, MapR	Integration with open source Apache projects (ex: Hive, Storm, Kafka, Spark, etc)	Collaborative notebooks, easier deployments	An ideal initial entry point for sandbox experimentation

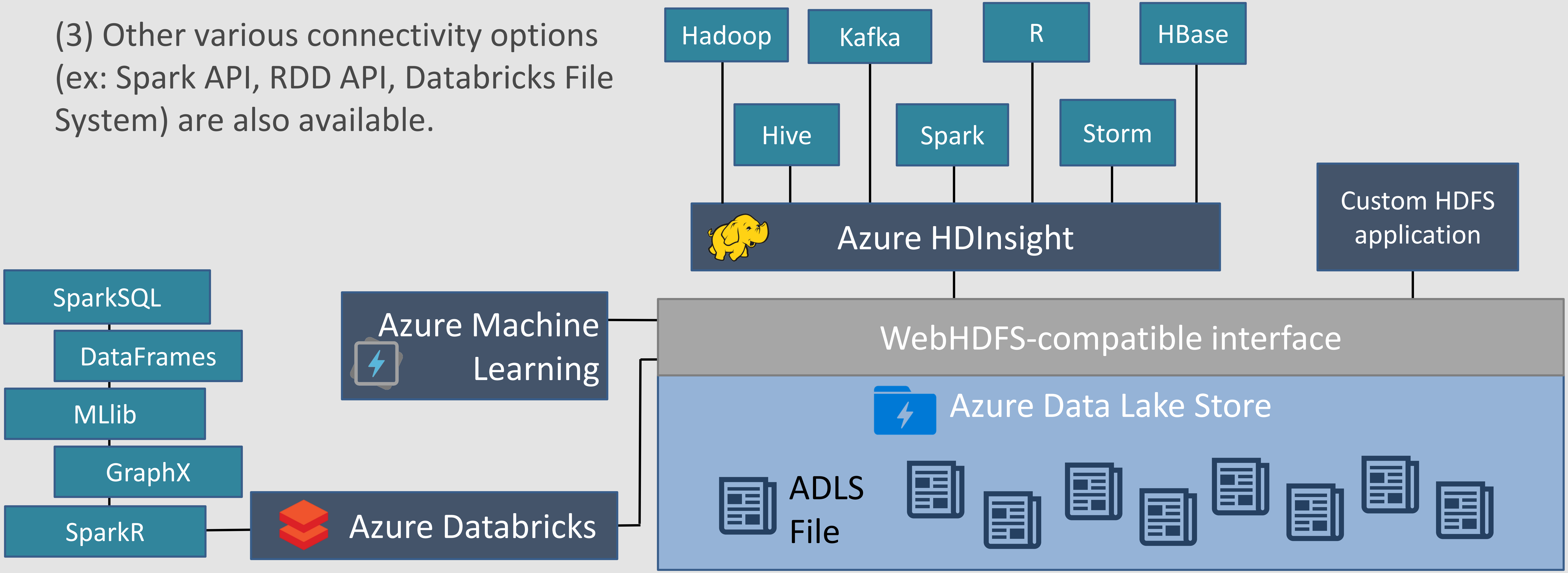
Intro to
Azure Data Lake

Azure Data Lake Store - Compatibility

(1) WebHDFS endpoint (https://) allows integration with open source projects.

(2) “AzureDataLakeFilesystem” (adl://) provides additional performance enhancements not available in WebHDFS.

(3) Other various connectivity options (ex: Spark API, RDD API, Databricks File System) are also available.



Azure Data Lake Store – Distributed File System

Files of any size can be stored because ADLS is a distributed system which file contents are divided up across backend storage nodes.

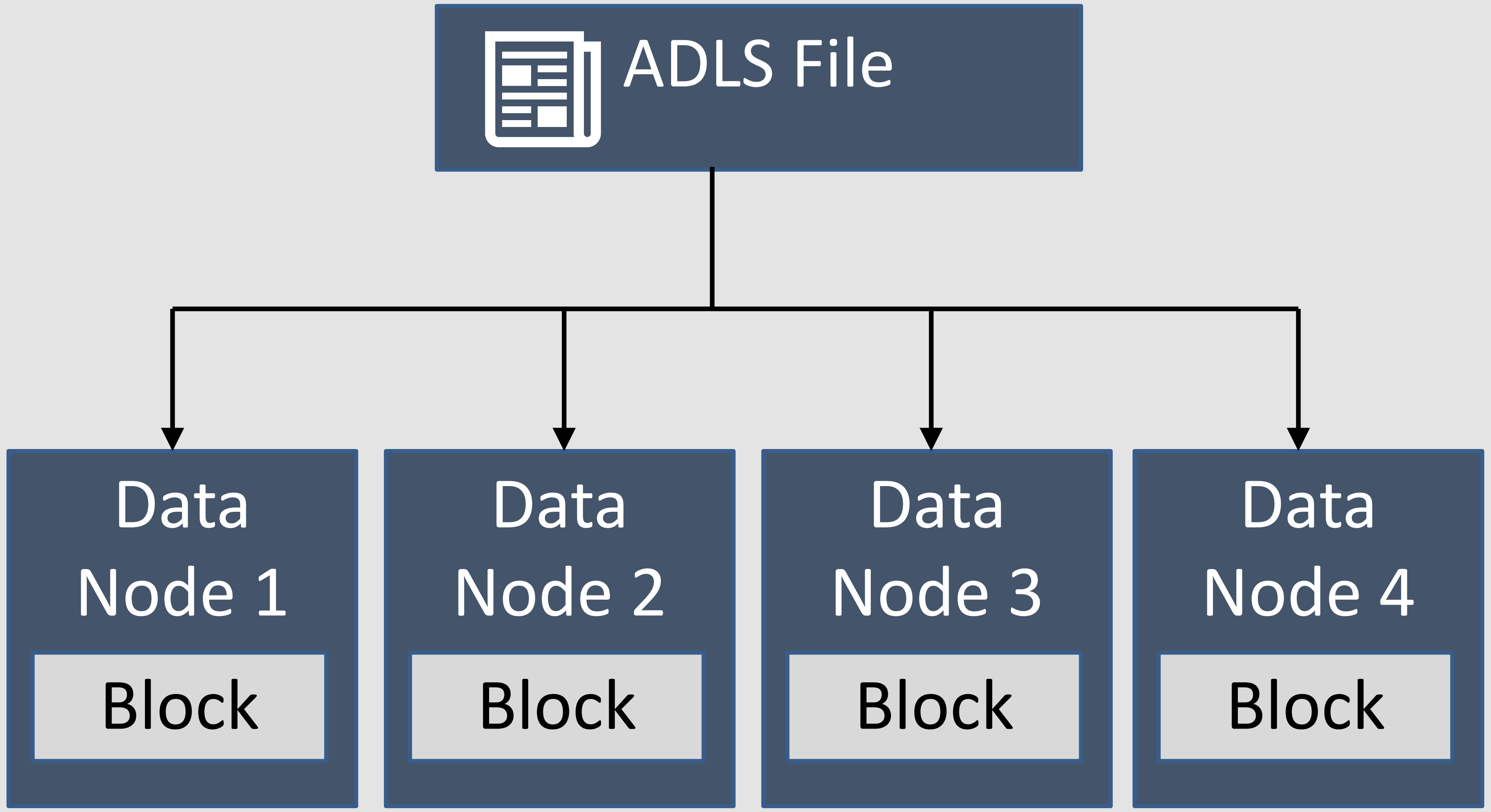
A read operation on the file is also parallelized across the nodes.

Blocks are also replicated for fault tolerance.

The ideal file size in ADLS is 256MB – 2GB in size.

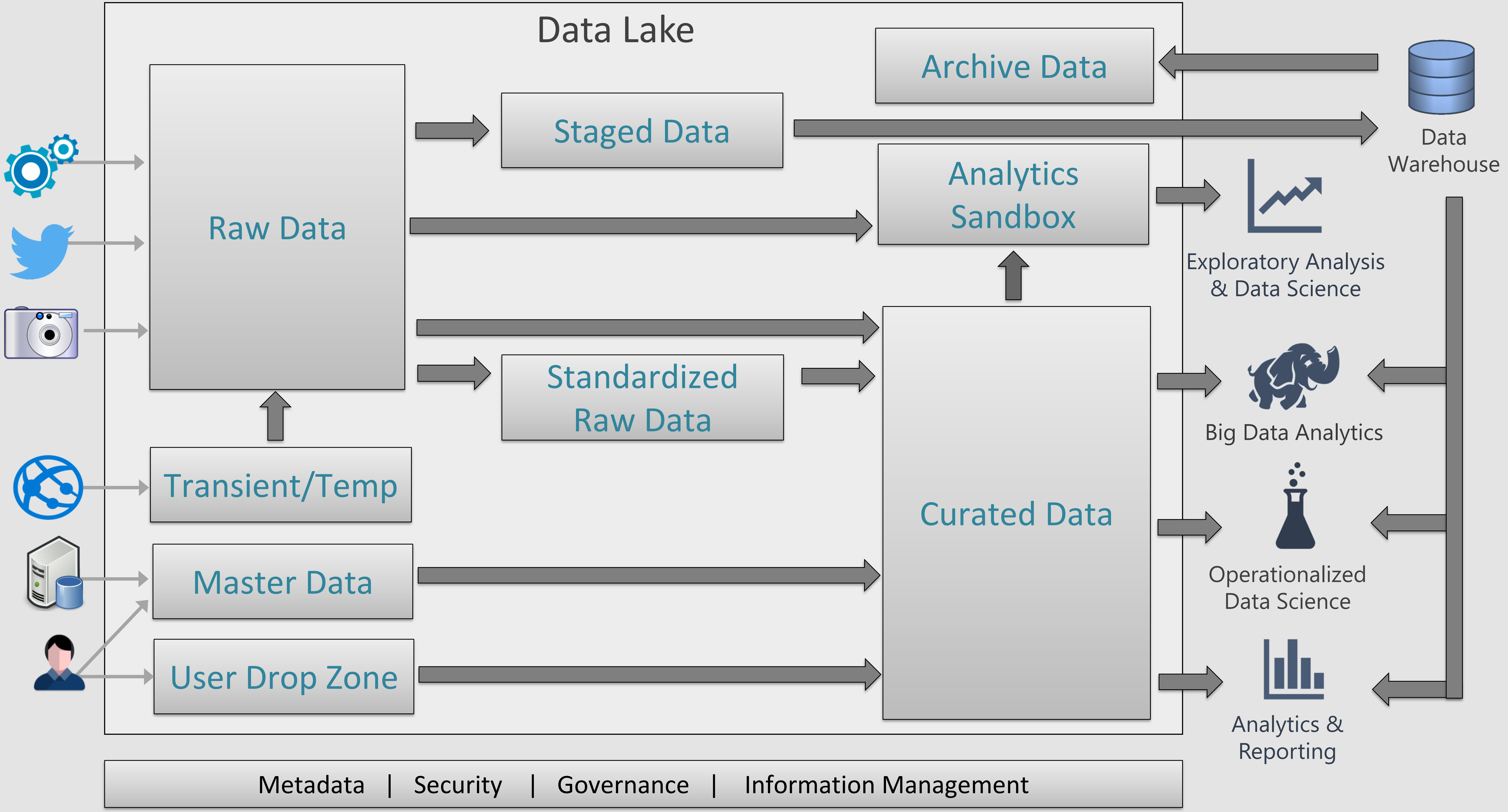
Many very tiny files introduces significant overhead which reduces performance. This is a well-known issue with storing data in HDFS. Techniques:

- Append-only data streams
- Consolidation of data into larger files

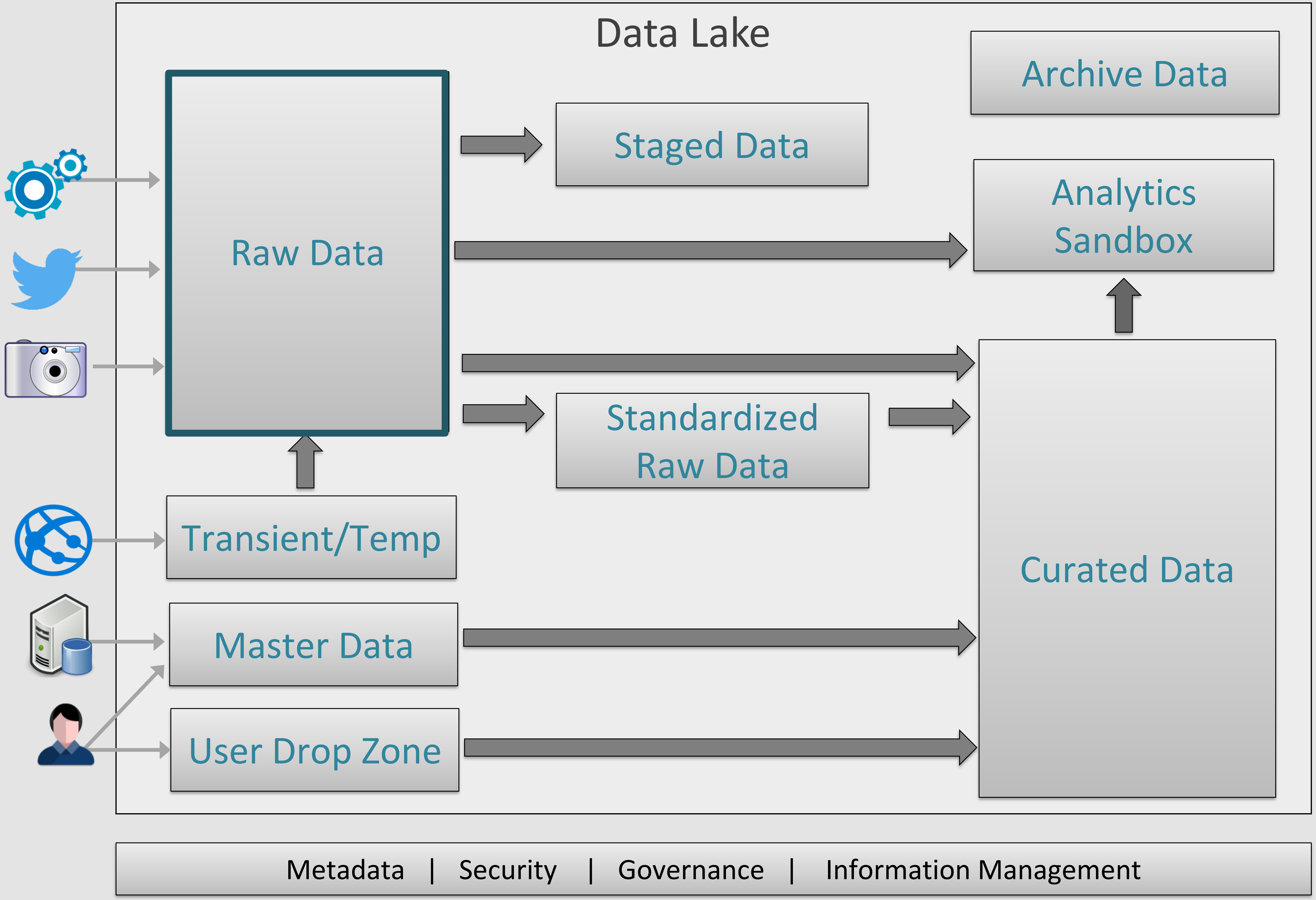


Designing the Structure of a Data Lake

Designing the Zones of a Data Lake

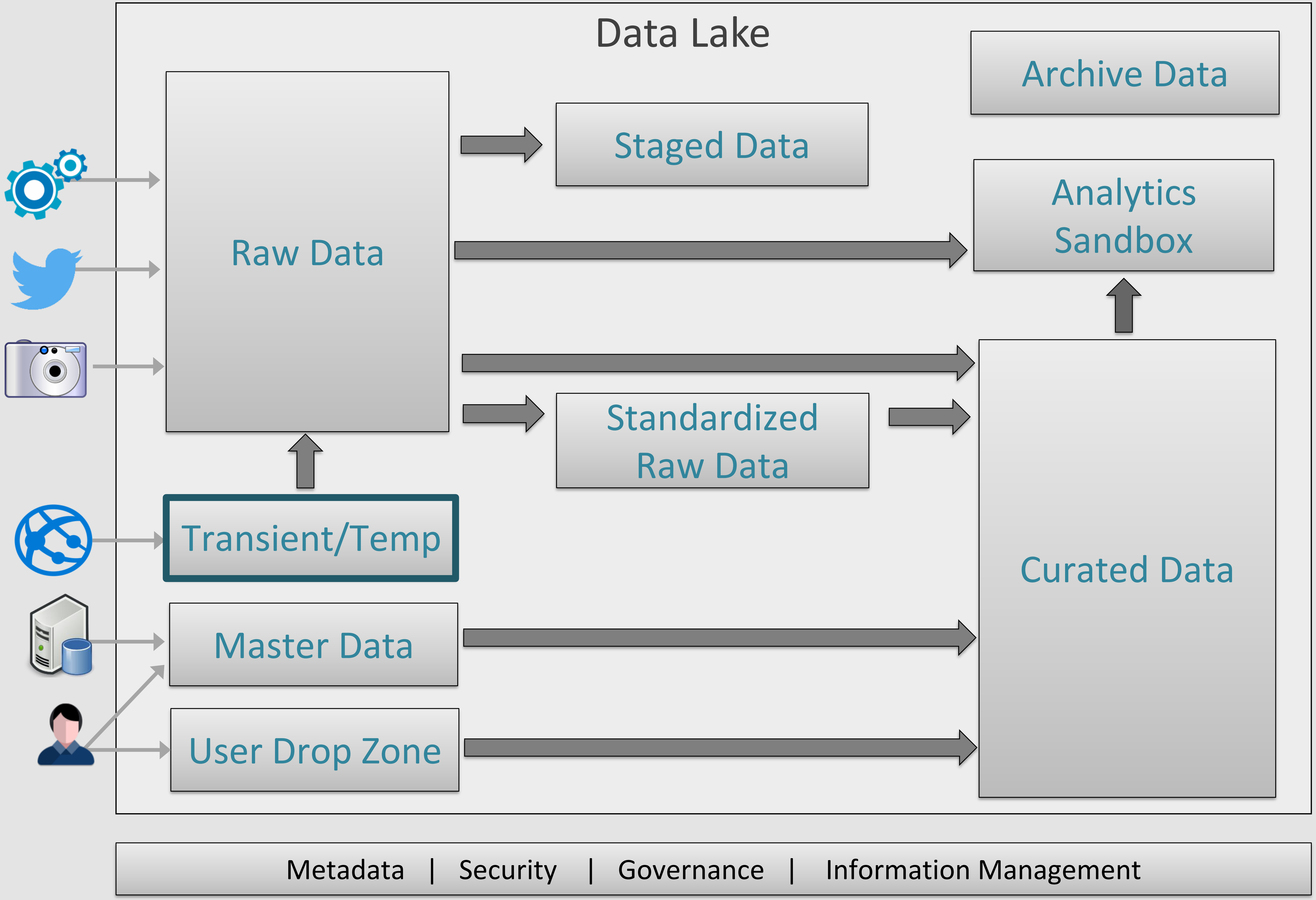


Raw Data Zone



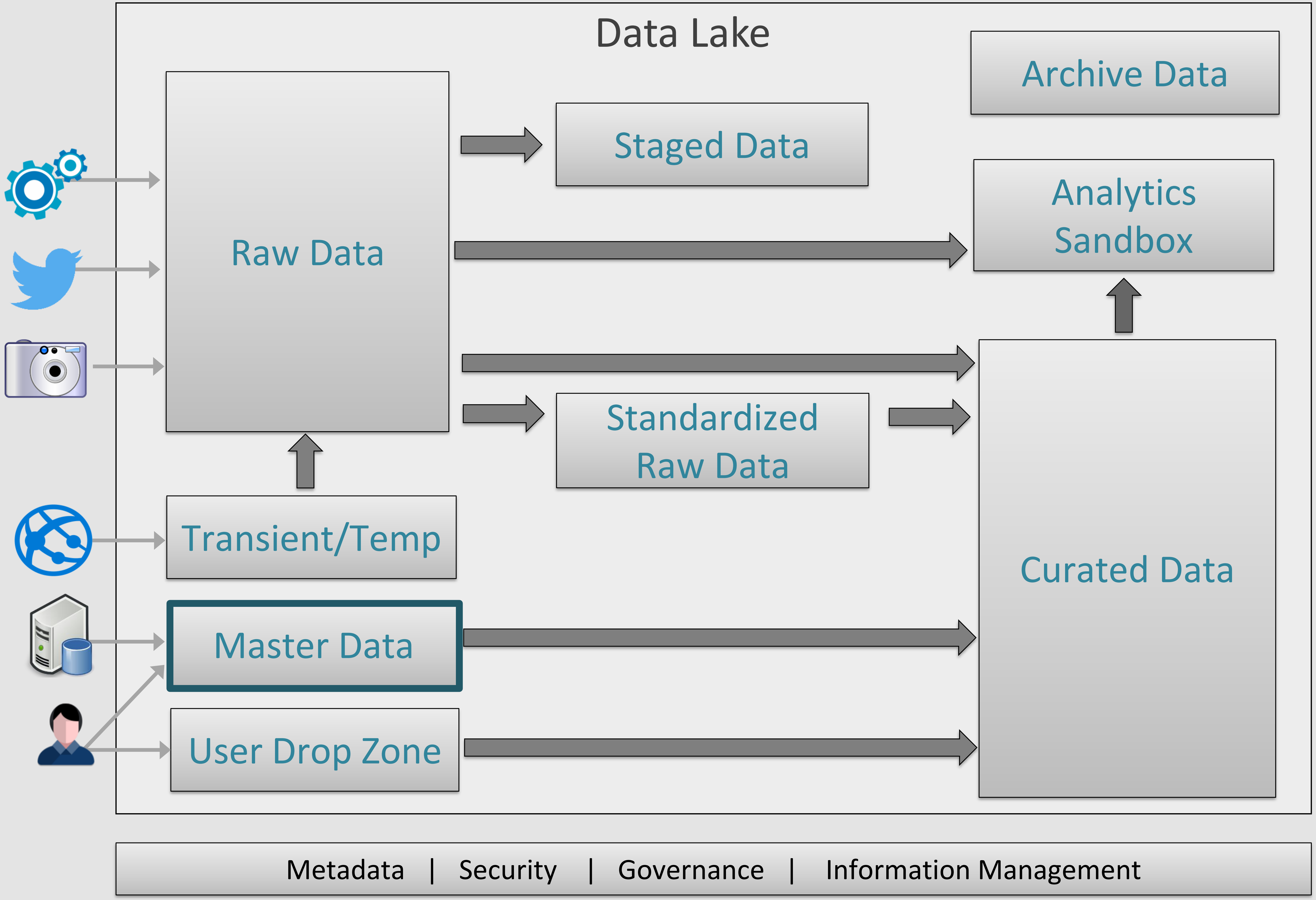
- ✓ Storage in native format for any type of data
- ✓ Exact copy from the source
- ✓ Immutable to change
- ✓ Typically append-only
- ✓ History retained indefinitely
- ✓ Extremely limited access to the Raw Data Zone – no operationalized usage
- ✓ Everything downstream from here can be regenerated from raw data

Transient/Temp Zone



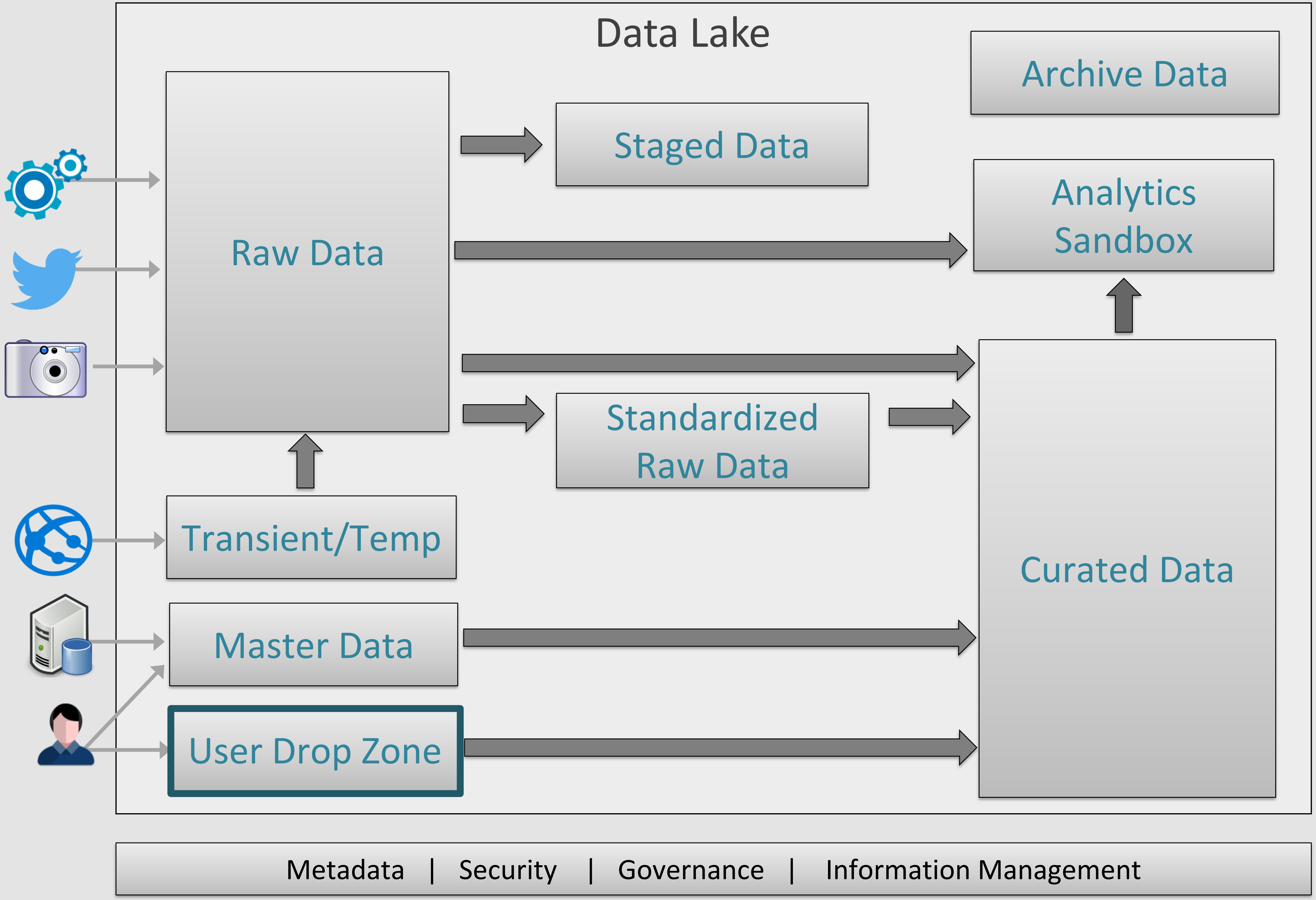
- ✓ Selectively utilized
- ✓ Useful when data quality checks or validation is required before the data is routed to the Raw Data Zone for retention
- ✓ Useful when you need a "New Data" zone separate from Raw Data Zone
  (ex: to ensure that jobs pulling data from Raw receive consistent data)
- ✓ Could contain transient, low-latency data
  (aka 'speed layer')

Master Data Zone



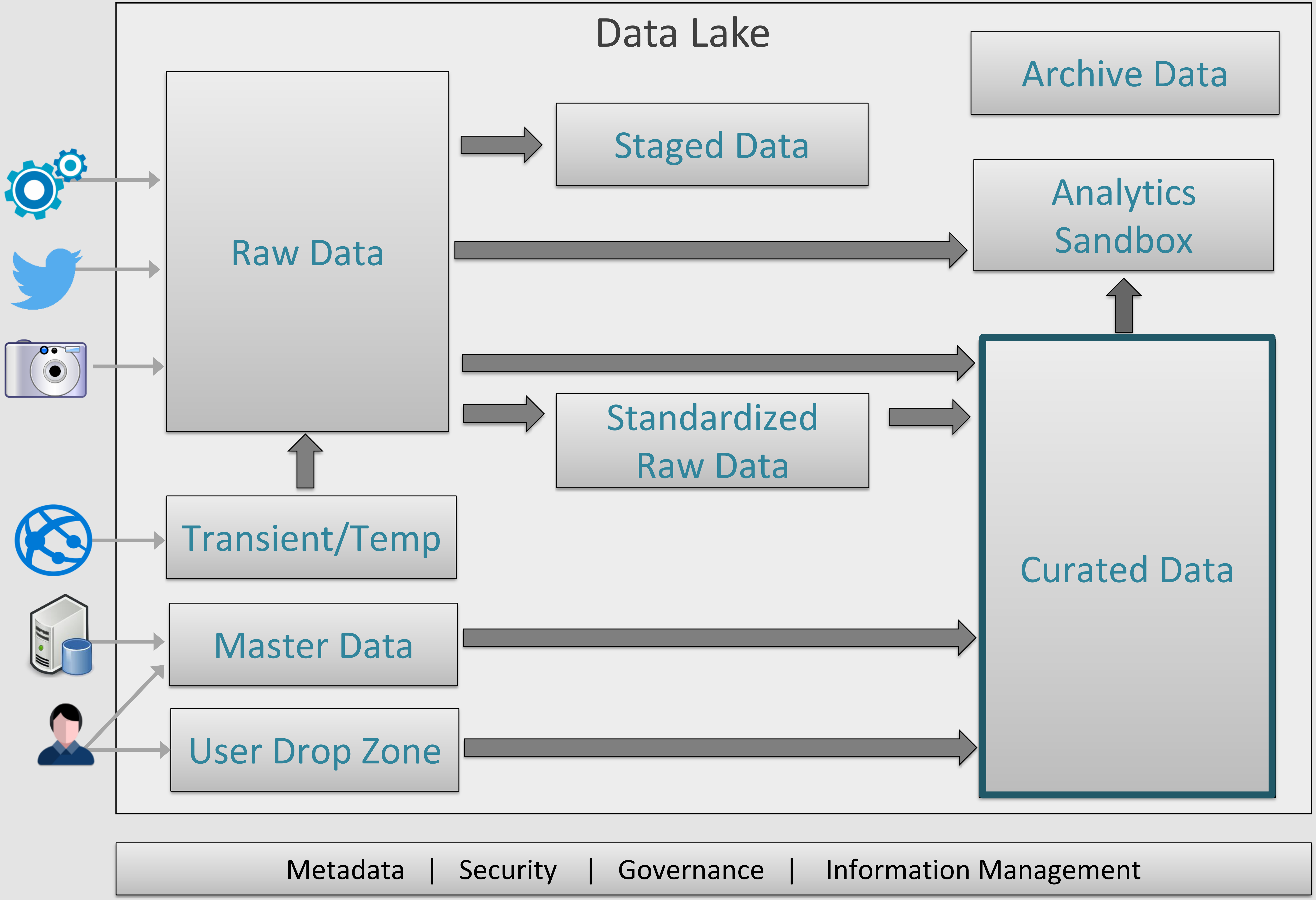
- ✓ Reference data to augment analysis

User Drop Zone



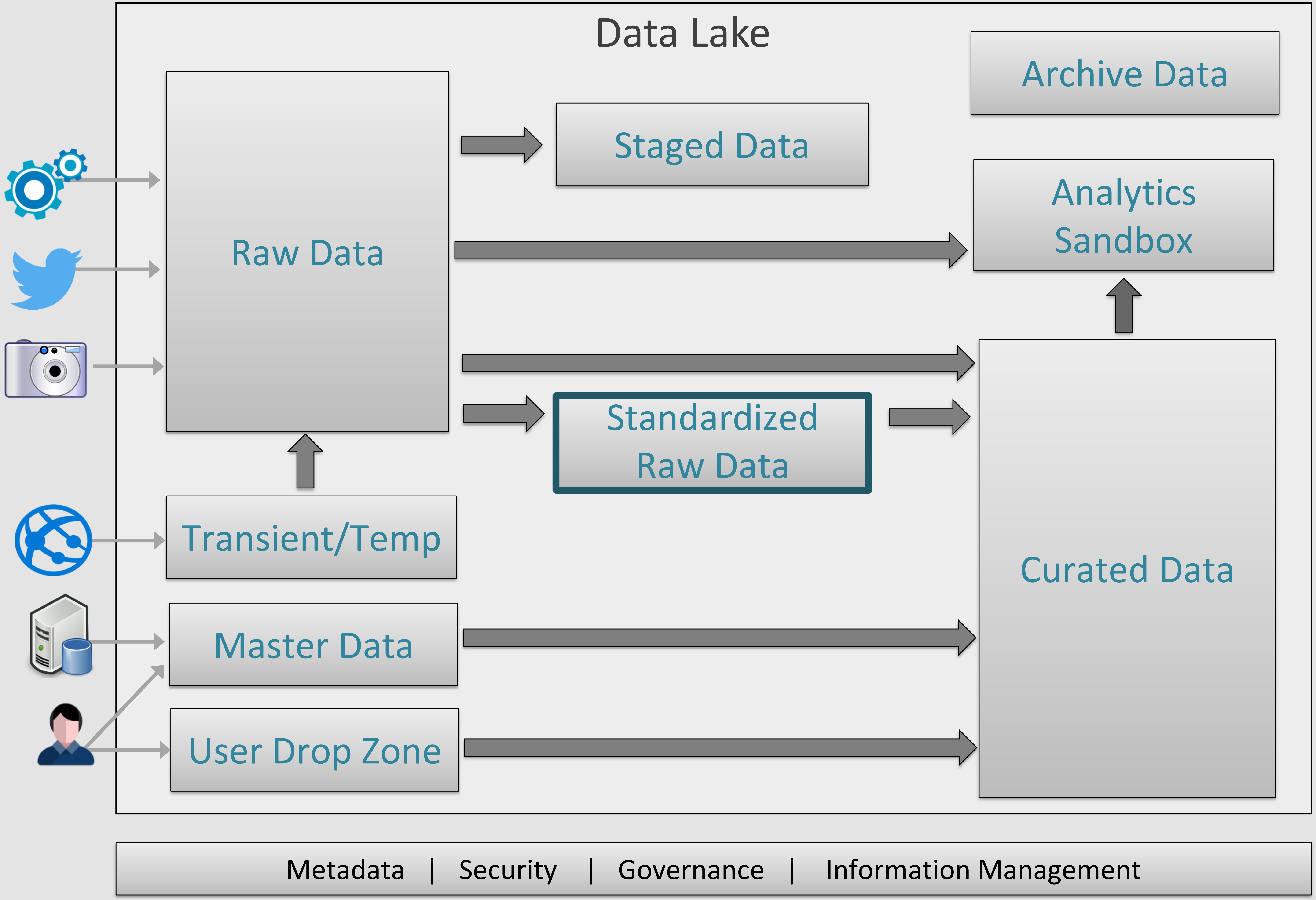
- ✓ Manually-generated data to augment analysis

Curated Data Zone



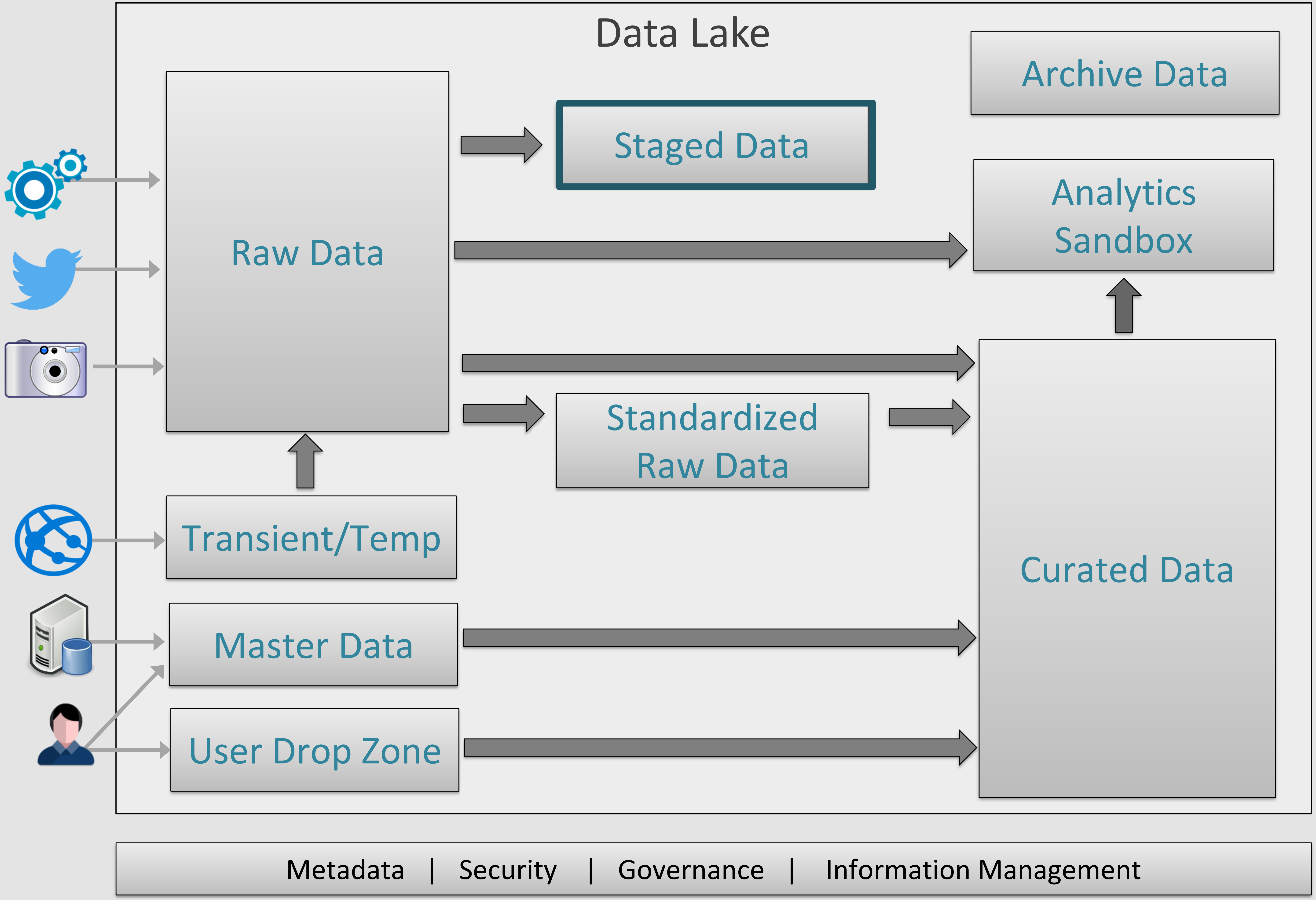
- ✓ Cleansed and transformed
- ✓ Organized for optimal data delivery (aka 'serving layer')
- ✓ Nearly all self-service data access comes from the Curated Data Zone
- ✓ Standard governance and security policies
- ✓ Standard change management principles

Standardized Data Zone



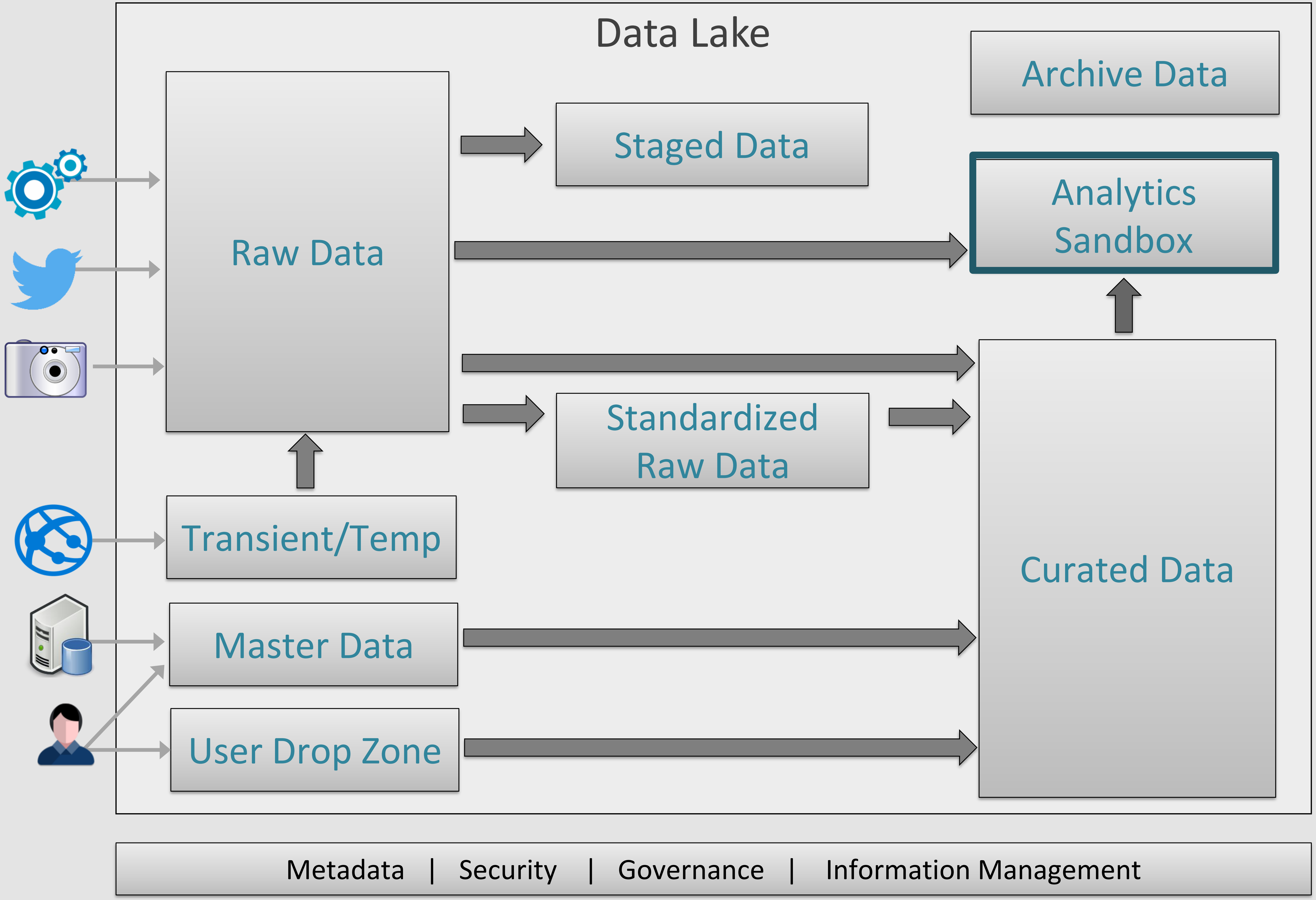
- ✓ A standardized version of the Raw Data Zone applicable to data structures which vary in format – ex: JSON which is standardized into consistent columns & rows (aka 'semantic normalization')
- ✓ No real cleansing or transformations applied
- ✓ Intermediary to assist creation of curated data
- ✓ File consolidations (ex: solve 'small files' performance issues)

Staged Data Zone



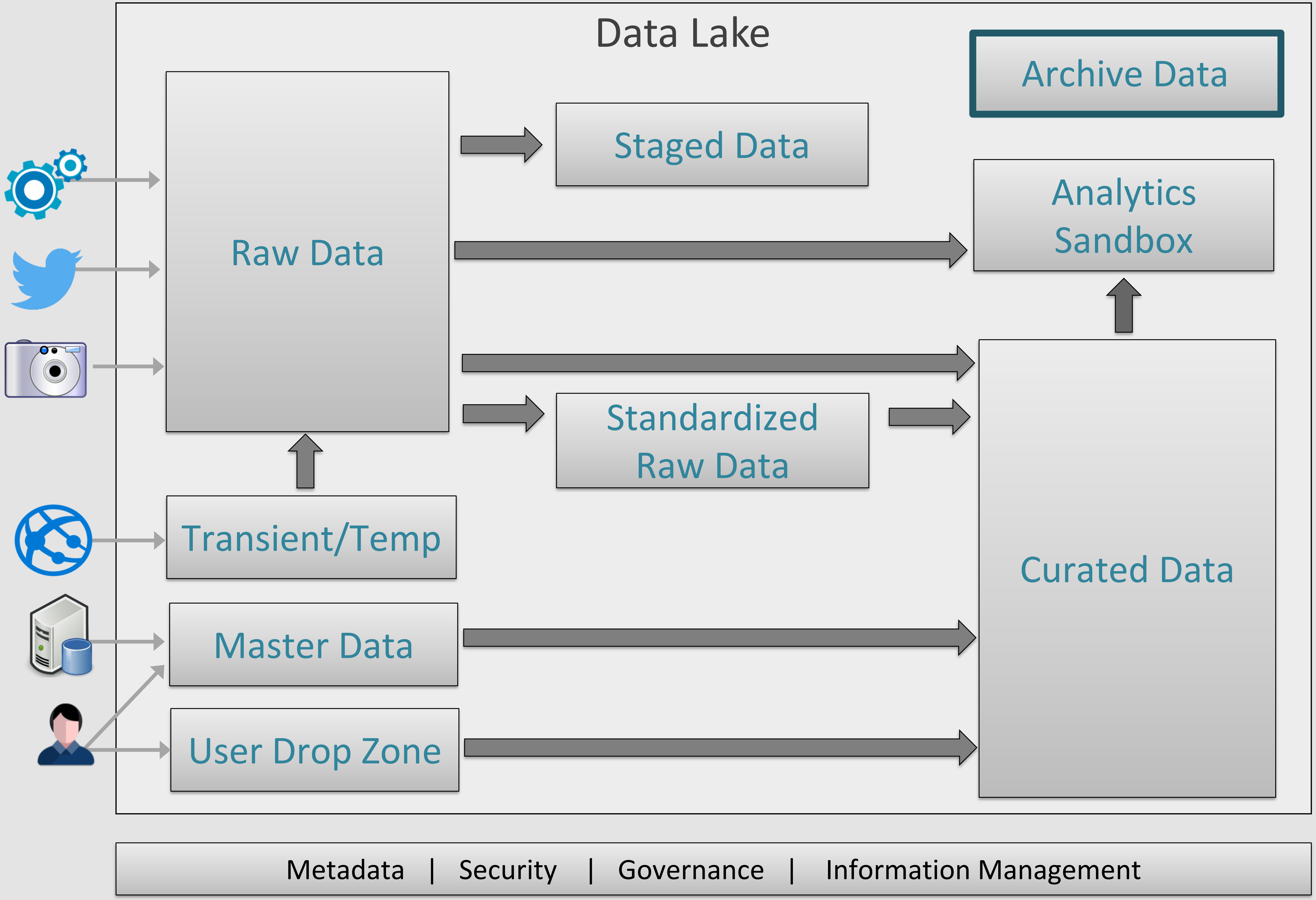
- ✓ Data which is staged for a particular purpose or application (thus has certain columns, certain formats, with or without headers, etc.)

Analytics Sandbox Zone

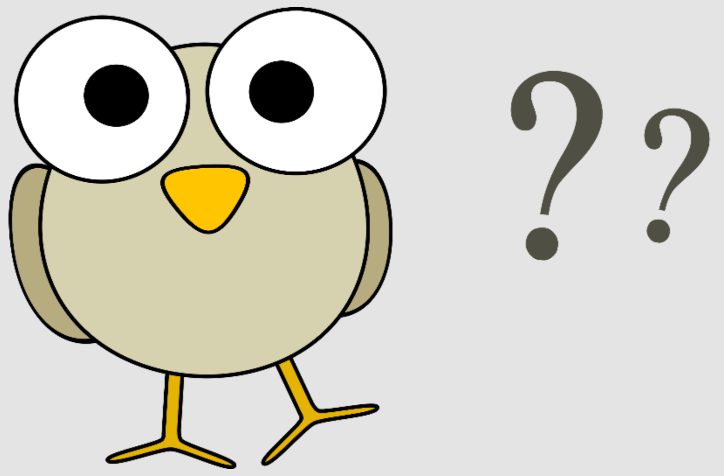


- ✓ Workspace for data science and exploratory activities
- ✓ Minimal, if any, governance and standards (purposely undisciplined)
- ✓ Valuable efforts are "productionized" and "operationalized" to the Curated Data Zone
- ✓ Not used for self-service, operationalized, purposes

Archive Data Zone



- ✓ An active archive
- ✓ Contains aged data offloaded from a data warehouse or other application
- ✓ Available for querying when needed (typically only occasionally)



What are some ways we could potentially organize data in a data lake?

Organizing a Data Lake

Objectives

- ✓ Plan the structure based on optimal data retrieval
- ✓ Avoid a chaotic, unorganized data swamp

Common ways to organize the data:

Time Partitioning
Year/Month/Day/Hour/Minute

Subject Area

Security Boundaries
Department
Business unit
etc…

Downstream App/Purpose

Data Retention Policy
Temporary data
Permanent data
Applicable period (ex: project lifetime)
etc…

Business Impact / Criticality
High (HBI)
Medium (MBI)
Low (LBI)
etc…

Owner / Steward / SME

Probability of Data Access
Recent/current data
Historical data
etc…

Confidential Classification
Public information
Internal use only
Supplier/partner confidential
Personally identifiable information (PII)
Sensitive – financial
Sensitive – intellectual property
etc…

Organizing a Data Lake (2/7)

Raw Data Zone

Subject Area
- Data Source
  - Object
    - Date Loaded
      - File(s)

Sales
- Salesforce
  - CustomerContacts
    - 2016
      - 12
        - 01
          - CustContact_2016_12_01.txt

Example 1

Pros: Subject area at top level, organization-wide
Partitioned by time

Cons: No obvious security or organizational boundaries

Curated Data Zone

Purpose
- Type
  - Snapshot Date
    - File(s)

Sales Trending Analysis
- Summarized
  - 2016_12_01
    - SalesTrend_2016_12_01.txt

Organizing a Data Lake

Raw Data Zone

- Organization Unit
  - Subject Area
    - Data Source
      - Object
        - Date Loaded
          - File(s)

- East Division
  - Sales
    - Salesforce
      - CustomerContacts
        - 2016
          - 12
            - 01
              - CustContact_2016_12_01.txt

Example 2

Pros: Security at the organizational level
Partitioned by time

Cons: Potentially siloed data, duplicated data

Curated Data Zone

- Organizational Unit
  - Purpose
    - Type
      - Snapshot Date
        - File(s)

- East Division
  - Sales Trending Analysis
    - Summarized
      - 2016_12_01
        - SalesTrend_2016_12_01.txt

Organizing a Data Lake

Example 3

Pros: Segregates records coming in, going out, as well as error records
Time partitioning can go down to the hour, or even minute level, depending on volume (ex: IoT data)

Cons: Not obvious by the names what the purpose of ‘out’ is (which could be ok if numerous downstream applications utilize the same ‘out’ data)

Raw Data Zone

Organization Unit	Organization Unit	Organization Unit
Subject Area	Subject Area	Subject Area
In	Out	Error
YYYY	YYYY	YYYY
MM	MM	MM
DD	DD	DD
HH	HH	HH
File(s)	File(s)	File(s)

Organizing a Data Lake

Subject Area 1
- RawData
  - YYYY
    - MM
- CuratedData
- MasterData
- StagedData

Subject Area 2
- RawData
  - YYYY
    - MM
- CuratedData
- MasterData
- StagedData

Example 4

Zones are a logical need, but they don’t necessarily have to be at the top of the structure

Pros: Security by subject area

Cons: All raw data is not centralized

Organizing a Data Lake

Do:

- ✓ Hyper-focus on ease of data discovery & retrieval – will one type of structure make more sense?
- ✓ Focus on security implications early – what data redundancy is allowed in exchange for security
- ✓ Include data lineage & relevant metadata with the data file itself whenever possible (ex: columns indicating source system where the data originated, source date, processed date, etc)
- ✓ Include the time element in ***both*** the folder structure & the file name
- ✓ Be liberal yet disciplined with folder structure (lots of nests are ok)
- ✓ Clearly separate out the zones so governance & policies can be applied separately
- ✓ Register the curated data with a catalog (ex: Azure Data Catalog) to document the metadata–a data catalog is even more important with a data lake
- ✓ Implement change management for migrating from a sandbox zone (discourage production use from the sandbox)
- ✓ Assign a data owner & data archival policies as part of the structure, or part of the metadata

Organizing a Data Lake (7/7)

Don't:

- × Do not combine mixed formats in a single folder structure
  - ✓ If it's looping through all files in a folder schema-on-read will fail if it finds a different format
  - ✓ Files in one folder should all be able to be traversed with the same script

- × Do not put your date partitions at the beginning of the file path -- it's much easier to organize & secure by subject area/department/etc if dates are the lowest folder level

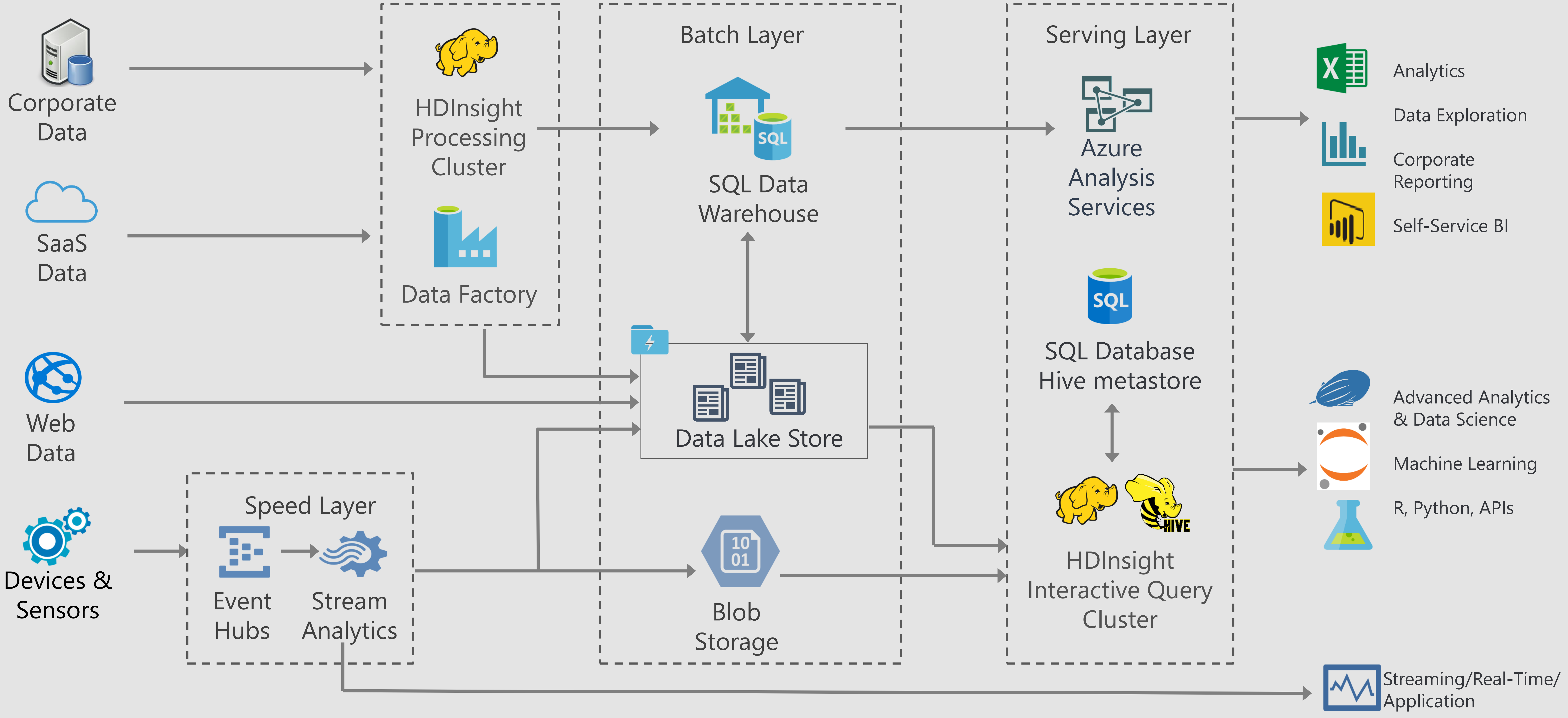
  Optimal for top level security:
  \SubjectArea\YYYY\MM\DD\FileData_YYYY_MM_DD.txt

  Tedious for enforcing security:
  \YYYY\MM\DD\SubjectArea\FileData_YYYY_MM_DD.txt

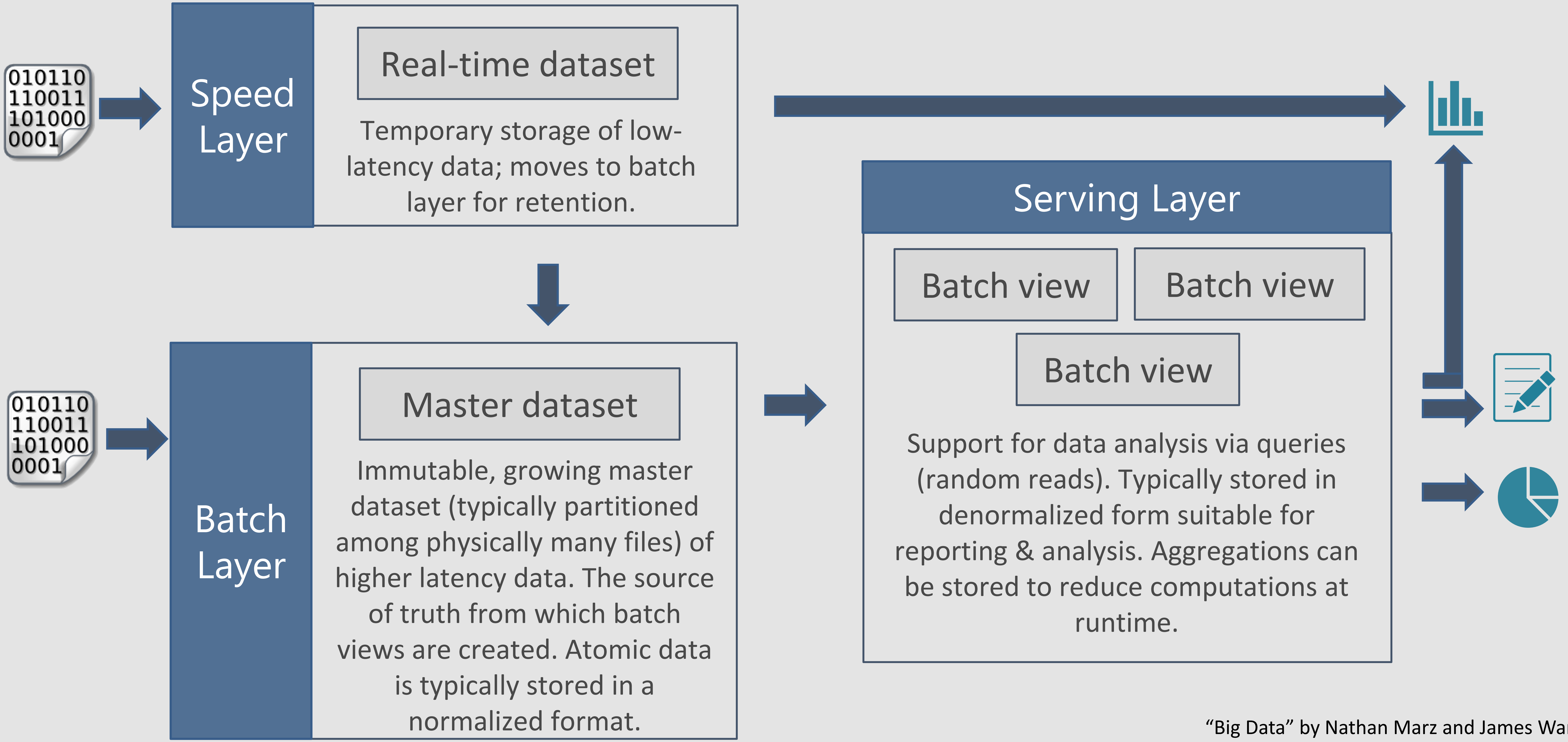
- × Do not neglect naming conventions. You might use camel case, or you might just go with all lower case – either is ok, as long as you're consistent because some languages are case-sensitive

Following
Big Data Principles
When Designing
A Data Lake

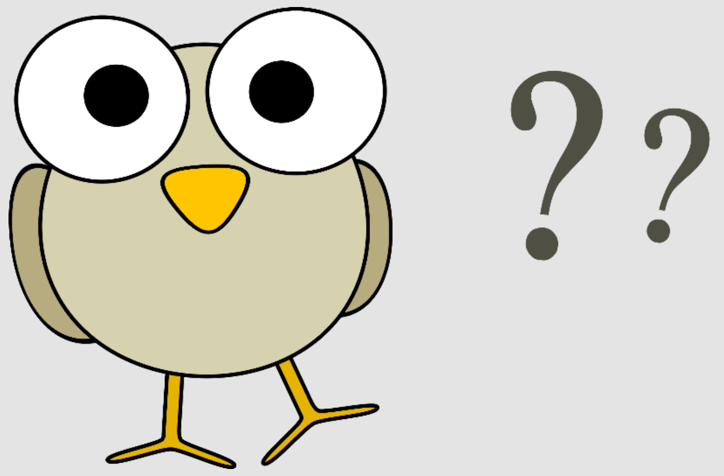
Lambda Architecture



Lambda Architecture



“Big Data” by Nathan Marz and James Warren



What principles might you expect to follow in a big data project?

Big Data Principles to Follow in a Data Lake Project

Immutable Raw Data

- Raw data is append-only & unchanging
- Continually growing
- No summarizations or deletions
- Bad data can be deleted, but it's rare
- Immutable data is resilient to human error

Recreatable

- Everything downstream from the raw data can be regenerated (error tolerant)
- Schema changes can be handled
- Unstructured data can always be re-structured ("semantic normalization")
- Speed layer may use approximations, corrected in the batch layer (eventual consistency)

Identifiable Data

- Timestamped
- Unique (tolerant of duplicates from retries)

Rawness of Data

- Obtain the rawest, most atomic, data available

Separate Layers

- Redundant data in both the batch & serving layers allows normalized & denormalized data

"Big Data" by Nathan Marz and James Warren

Schema Changes Over Time (1/2)

Schema changes include:

Addition of new columns
Removal of columns
Renaming of columns

Two options:

(1) Schema enforcement upon the ingestion of data
(2) Schema flexibility for the developers; deal with “standardizing” the data after ingestion

Schema Changes Over Time (1/2)

Raw Data:


```
{
"ID":"a4791906-a9a7-4d47-956b-9df12b5f72
"SID":"a93c45cb-9420-4efc-919e-ca5338660
"TID":null,
"CID":"98FD9198EA4A47C41AB8CDC31DD53077"
"LID":"87F4BE0AEB0CD4CF56C280F7660F0281"
"PID":"S1",
"PV":"11.2.0.0",
"TS":"2017-09-18T00:01:17.7781017Z",
"EC":"47623b79-f290-45fe-a23a-2e840ff5f6
"ED":
    {
    "Name":"UsageStopped",
    "Path":"WindowsPerformanceConsole",
    "D":"00:02:01.0644843"
    },
"EventProcessedUtcTime":"2017-09-18T00:0
"PartitionId":1,
"EventEnqueuedUtcTime":"2017-09-18T00:01:46.2120000Z"
}
```

```
{
"ID":"28439047-b4a7-4aa6-9a4b-ebcd9cc1e028",
"SID":"69f947db-52de-4f19-8d4a-3b16ed11c27d",
"TID":null,
"CID":"BD5145B8D4CDFBFEC72A23793F309
"LID":"577C0A72744D3A13F9A40663C3D34
"PID":"S1",
"PV":"11.2.0.0",
"TS":"2017-09-18T14:14:38.4585269Z",
"EC":"c0a0d257-752a-4899-a0eb-5680a3
"ED":
    {
    "Val":74.0,
    "PrevVal":0.0,
    "Name":"PropertyValueChanged",
    "Path":"EH\\ScoreChange"
    },
"EventProcessedUtcTime":"2017-09-18T
"PartitionId":1,
"EventEnqueuedUtcTime":"2017-09-18T1
}
```

```
{
"ID":"fa27bf3f-b101-4c92-87e3-dc63f20614b2",
"SID":"a73215bb-b831-43f2-a3cd-fed140d2c1eb",
"TID":null,
"CID":"71021EFAAF18DF534D0F1E986E687B04",
"LID":"DAA3C04DF132B74027475ADE1D1788D8",
"PID":"S1",
"PV":"11.2.0.0",
"TS":"2017-09-18T03:29:49.7600422Z",
"EC":"e95861bb-cb49-4579-804c-6d0fafbed33c",
"ED":
    {
    "Name":"Navigation",
    "Path":"EM"
    },
"EventProcessedUtcTime":"2017-09-18T03:31:19.3962219Z",
"PartitionId":1,
"EventEnqueuedUtcTime":"2017-09-18T03:31:19.3510000Z"
}
```


Standardized Raw Data:

(semantic normalization)

InstanceID	SessionID	TransactionID	ClientID	LocationID	ProductID	ProductVersion
a4791906-a9a7-4d47-956b-9df12b5f7296	a93c45cb-9420-4efc-919e-ca5338660bbc	0	98FD9198EA4A47C41AB8CDC31DD53077	87F4BE0AEB0CD4CF56C280F7660F0281	S1	11.2.0.0
fa27bf3f-b101-4c92-87e3-dc63f20614b2	a73215bb-b831-43f2-a3cd-fed140d2c1eb	0	71021EFAAF18DF534D0F1E986E687B04	DAA3C04DF132B74027475ADE1D1788D8	S1	11.2.0.0
28439047-b4a7-4aa6-9a4b-ebcd9cc1e028	69f947db-52de-4f19-8d4a-3b16ed11c27d	0	BD5145B8D4CDFBFEC72A23793F309ECF	577C0A72744D3A13F9A40663C3D34CC0	S1	11.2.0.0
daf30e65-c7ab-4bed-b651-4e122e63cebf	d5fa951e-5ea9-4230-b2ef-9b48ef0e622f	0	BD5145B8D4CDFBFEC72A23793F309ECF	577C0A72744D3A13F9A40663C3D34CC0	S1	11.2.0.0

Row continues

TimestampUtc	EventClassID	EventClassName	EventPath	EventDuration	EventValue	EventPreviousValue
2017-09-18T00:01:17.7781017Z	47623b79-f290-45fe-a23a-2e840ff5f63f	UsageStopped	WindowsPerformanceConsole	02:01.1	0	0
2017-09-18T03:29:49.7600422Z	e95861bb-cb49-4579-804c-6d0fafbed33c	Navigation	EM	00:00.0	0	0
2017-09-18T14:14:38.4585269Z	c0a0d257-752a-4899-a0eb-5680a323bd19	PropertyValueChanged	EH\ScoreChange	00:00.0	74	0
2017-09-18T14:16:29.8271102Z	4989e98a-fedb-4156-9eba-ee7f6961a48a	PropertyValue	MonitoringService\Count	00:00.0	1	0

Data Formats & Data Compression

CSV

Commonly used. Human-readable. Not compressed. Typically not the best choice for large datasets.

JSON

Commonly used. Human-readable. Self-describing schema.

Parquet

Columnar format; highly compressed.

Avro

Row-based format. Supports compression. Schema encoded on the file.

ORC (optimized row columnar)

Columnar format with collections of rows. Light indexing and statistics.

Deciding on a format

- Supported formats by key systems
- Integration with other systems
- File sizes
- Schema changes over time
- If a self-describing schema is desired
- Data type support
- Data format compatibility
- Performance of workload (read vs. write)
- Convenience & ease of use

Techniques to Recompute the Serving Layer

Full recomputation

The entire master dataset is used to recompute the batch views in the serving layer.

Pros: Simplicity

Better human fault-tolerance

Ability to continually reap benefits of improved algorithms or calculations

Easier to keep wide datasets which contain redundant data synchronized/consistent

Cons: Performance; speed of updates

CPU and I/O heavy

Not practical for extremely large datasets

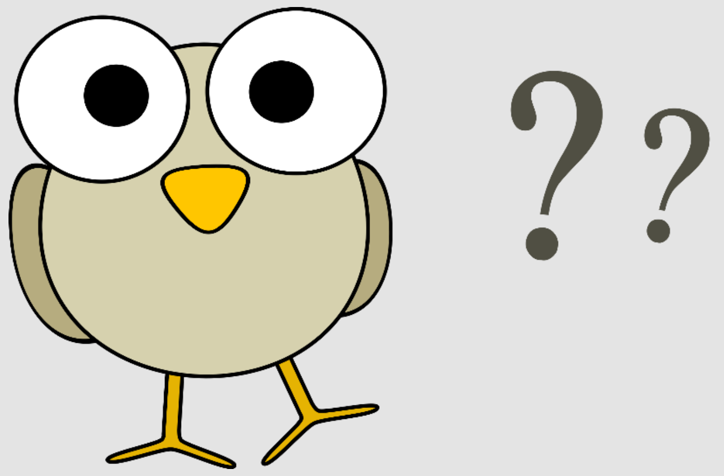
Incremental recomputation

Only new data from the master dataset is involved in recomputations.

Pros: Better performance

Cons: Significantly more complex

Still need a way to do a full recomputation in the event of errors or significant changes



What is the state of data modeling for files stored in a data lake?

Data Modeling for Files in a Data Lake

Wide datasets, with all data needed in one file, are commonly used

Pros: Easy to do analysis.
Data can be co-located on the nodes as the data gets distributed (depending on the tool).
The desired format frequently for data scientists & the tools they use.
Usually well-suited to in-memory, columnar, data formats.

Cons: Data is repeated (particularly dimensional data) across lots of files.
Keeping data updated across many files can take time.
Data of different granularities can get tricky.
Immutable, append-only data means everything acts like a slowly changing dimension.

Recap, Suggestions, Q&A