Data Lake:
centralize in on-prem vs. decentralize on cloud

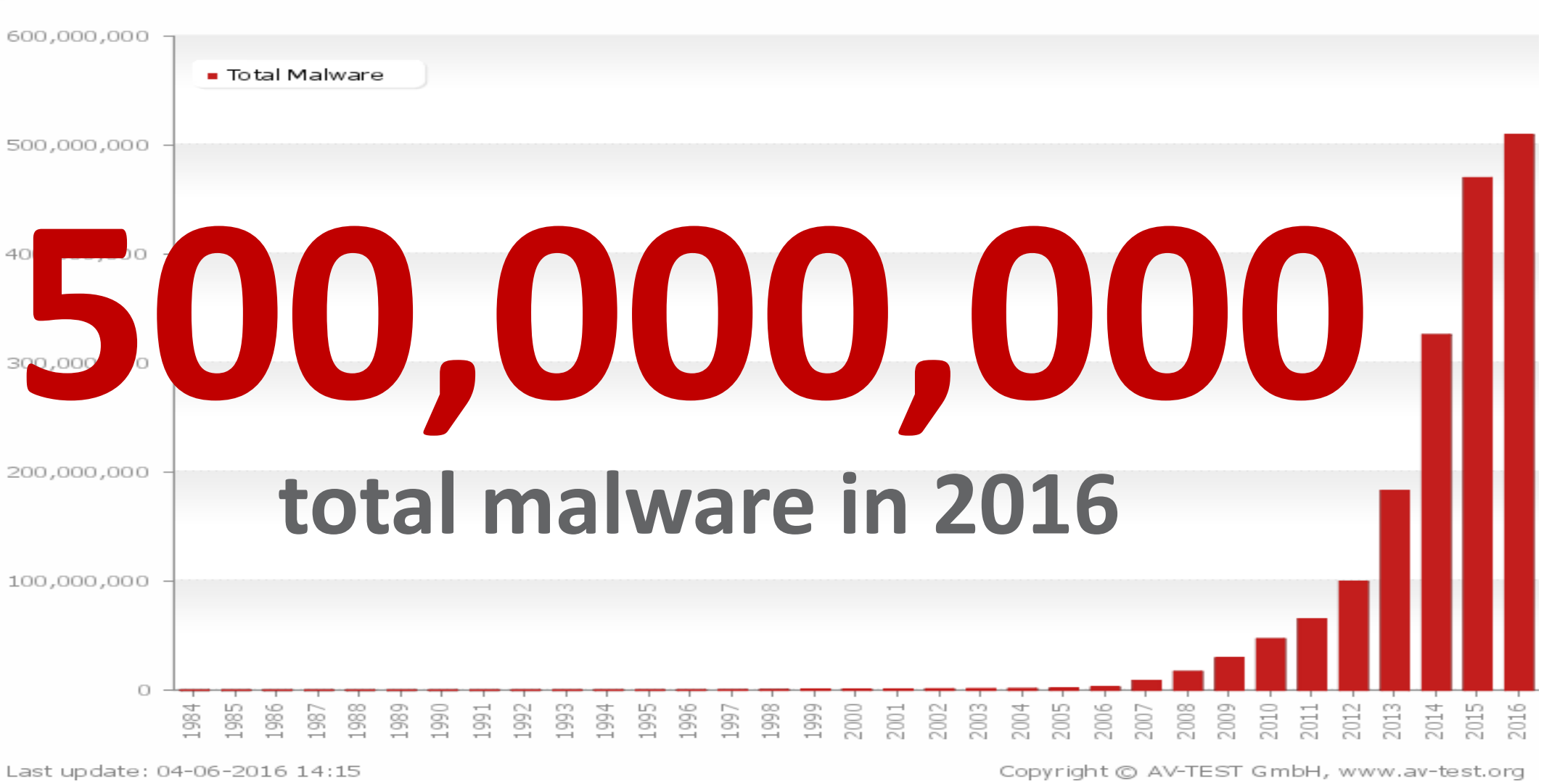
Jeff Hung / Trend Micro

Sep 30, 2017

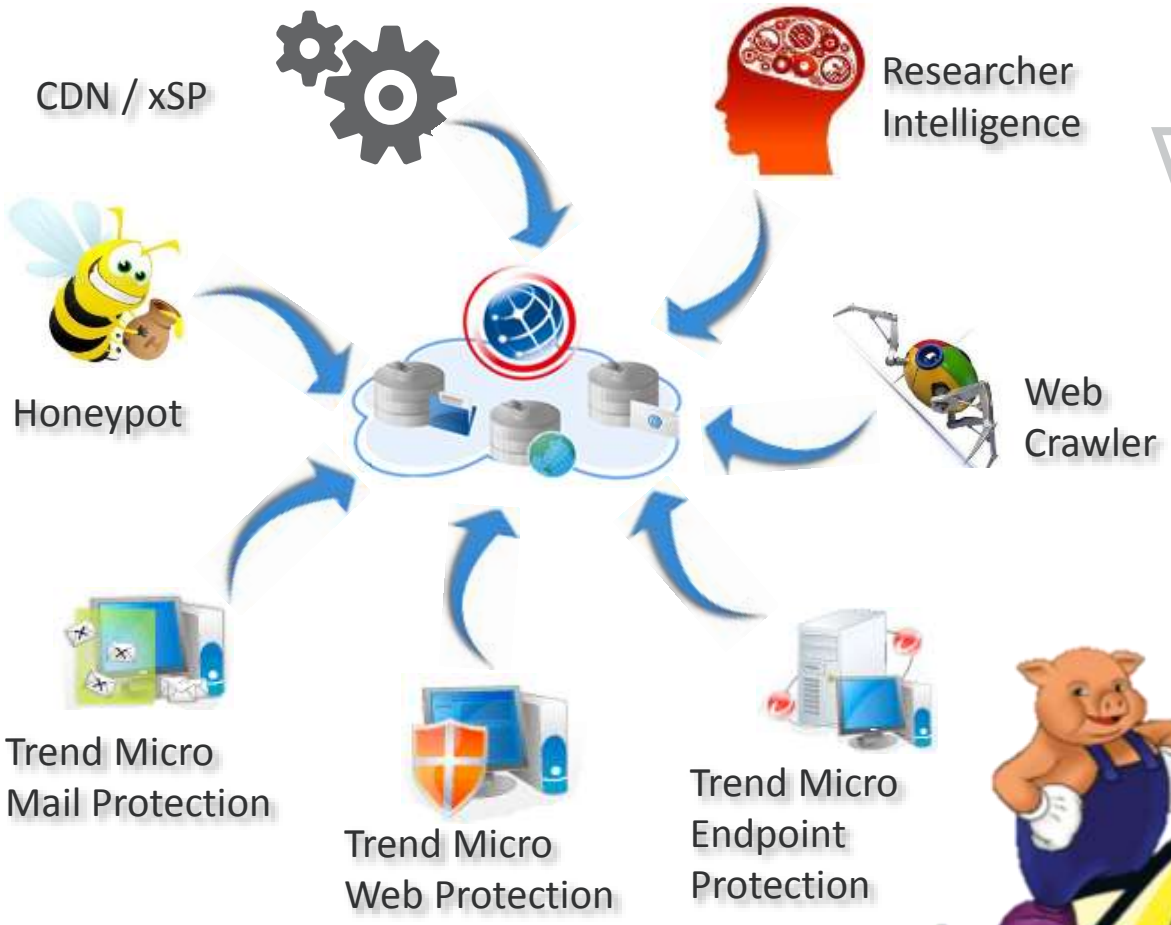
@jeffhung

- Smart Protection Network (SPN)
- Big-data Platform & Solution
- Hadoop PROD since 2009
- Work on AWS/EMR since 2014

Malware Explosion



From data to solution



Threat Protections

Regional distribution of ransomware threats from January 2016 to March 2017

2009

~40 Hadoop nodes

~15 Service/user accounts

3 Teams

<50 TB storage

<100 Jobs per day

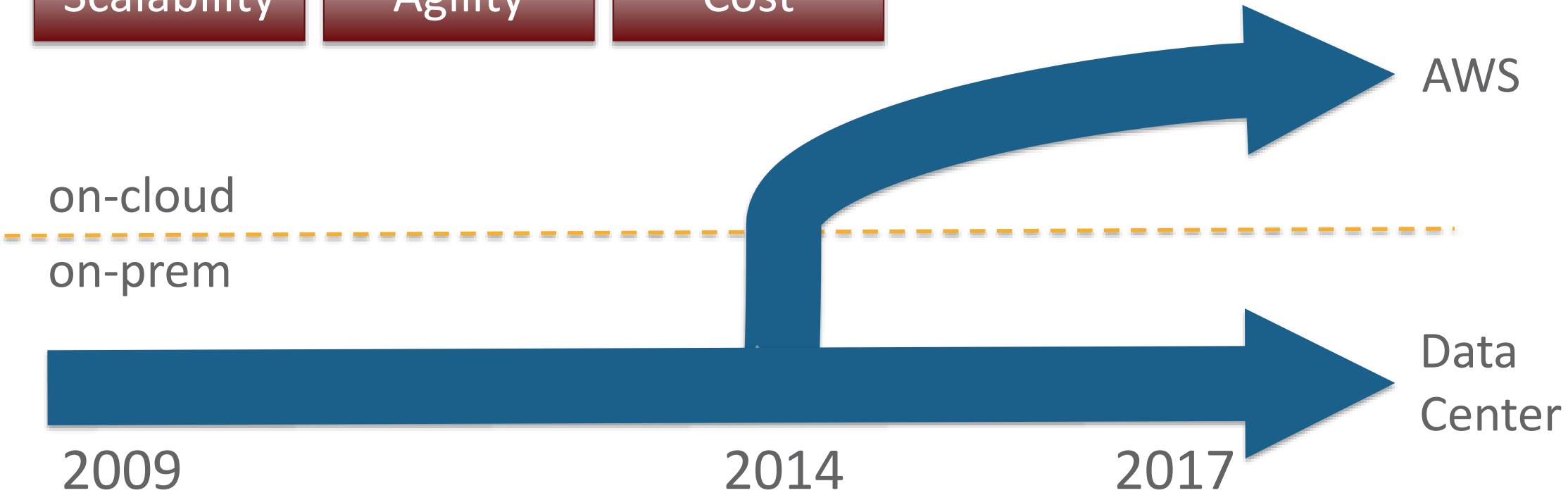
2014

~250	Hadoop nodes
~140	Service/user accounts
13	Teams
~1,500	TB storage
>16,000	Jobs per day

The SPN Journey

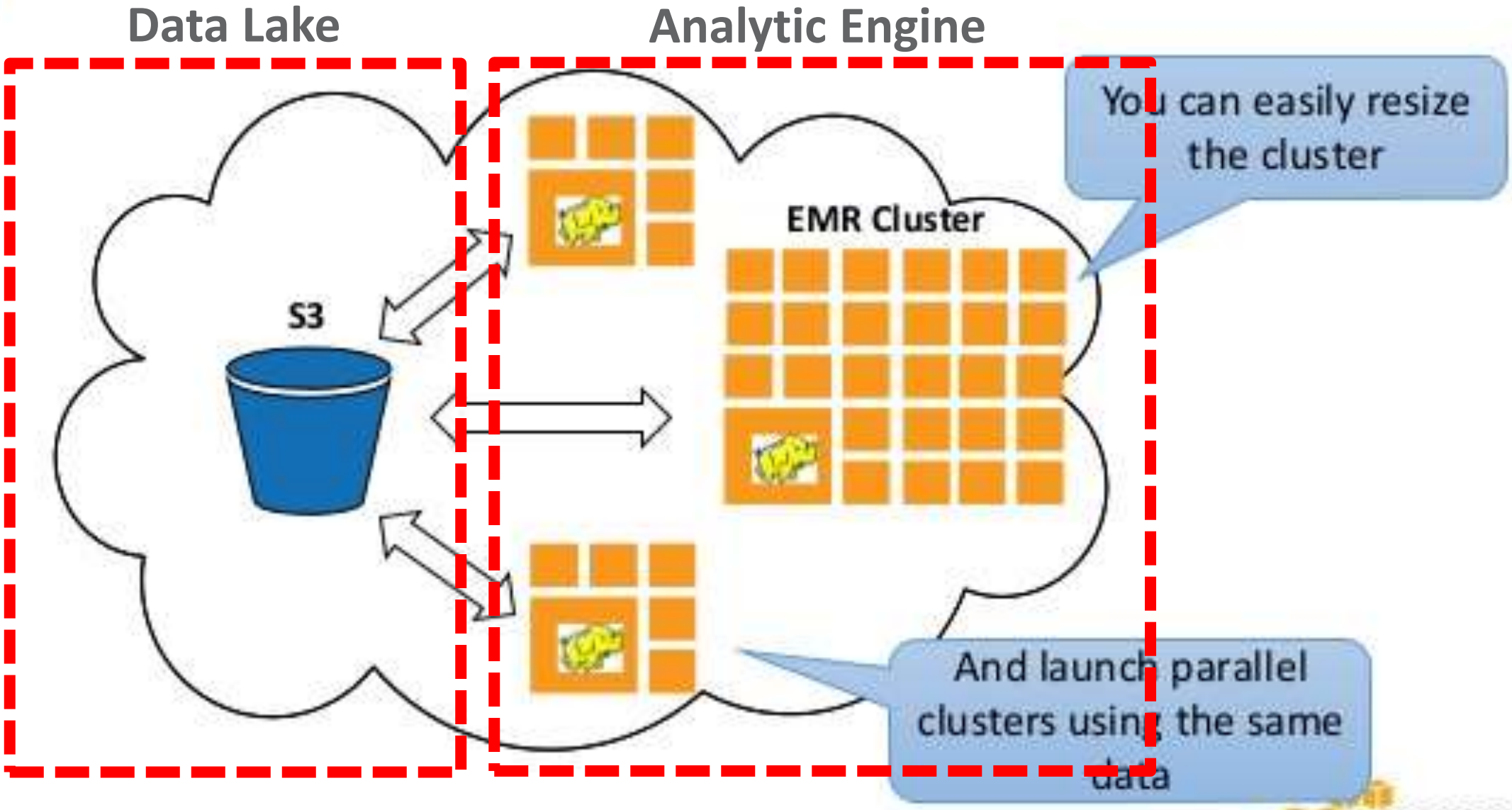
Why cloud?

Scalability | Agility | Cost



Big-data on the Cloud

EMR



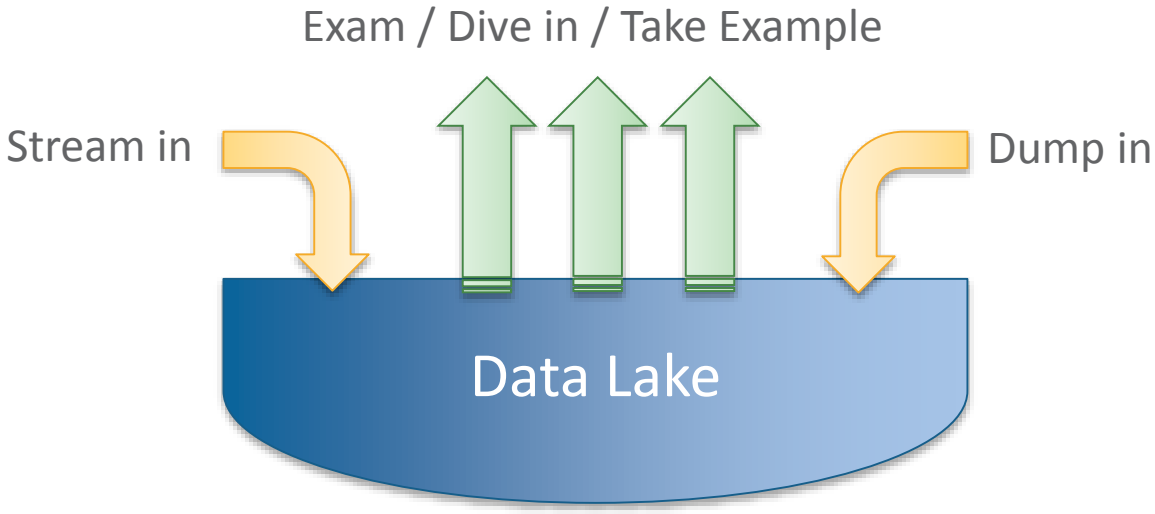
amazon web services

Data Lake


Jia-Ming Lake, Taitung, Taiwan
| PHOTO BY WISELEE | TAIPEI, TW | 2010-2013 |

“If you think of a datamart as a store of bottled water – cleansed and packaged and structured for easy consumption – the **data lake** is a large body of water in a more natural state. The contents of the data lake stream in from a source to fill the lake, and various users of the lake can come to examine, dive in, or take samples.”

– James Dixon, CTO of Pentaho



Data Lake fixes 2 problems

- Unknown Questions
- Information Silos

Unknown Questions

- You don’t know what you don’t know
- Premature optimization is the root of all evil

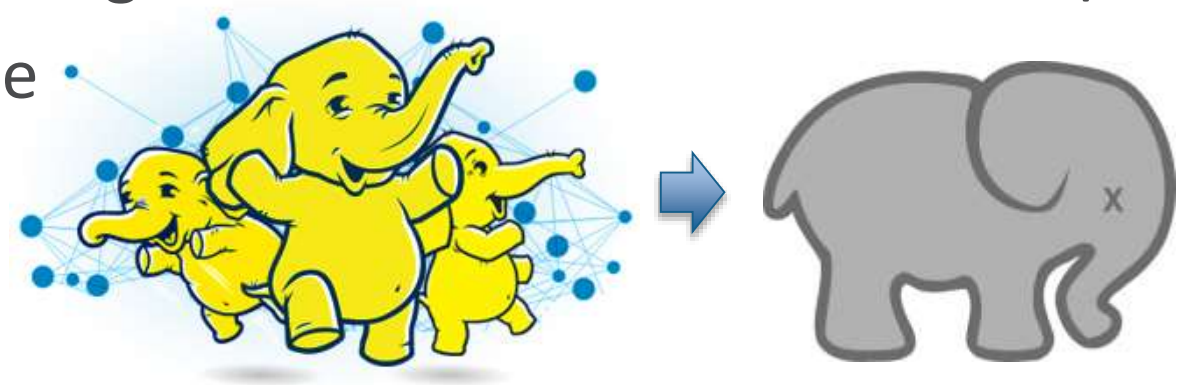
Data Warehouse/Mart	vs.	Data Lake
structured and preprocessed data	DATA	structured / semi-structured / unstructured / raw data
schema-on-write	PROCESSING	schema-on-read
expensive for large data volume	STORAGE	designed for low-cost storage

Information Silos

- Incompatible data system built by different teams
  - Conway's Law & NIH Syndrome
  - Technology Limitation
- Hadoop come to rescue
  - Free technology
  - Free computing
  - Free storage

But...

- Cost justification
- Development agility



Data Lake fixes 2 problems – How ?

- Unknown Questions ➔ Preserve low-level visibility
- Information Silos ➔ Make accessing data easy

We've done these right in on-premises datacenter.
Now we need to make them work on Cloud, too.

On-Prem vs. On Cloud

Centralized Infrastructure ←		→ Decentralized on Cloud Services
One Big Hadoop Cluster ←		→ Many EMR Clusters / S3 Buckets
Fine-grain Resource Sharing ←		→ Go Dutch & Pay-as-you-Go
Single Big-data Stack ←		→ Diverse Technology
UNIX Style Permission Control ←		→ Cloud Compatible ACL

Things apply to both scenarios: **File Format & Schema Design**

Centralized Infrastructure

- Have only one datacenter
- PROD/STG in same place
- Jobs & data in same cluster
- Do everything on our own

Access data in one place

Decentralized on Cloud Services

- Could use multiple AWS regions
- PROD/STG in different regions
- Jobs & data in different places
- Could leverage cloud services

Access data from anywhere

One Big Hadoop Cluster

- Runs all applications in same big cluster
- Bigger cluster = better flexibility
- Resource managed by YARN

Many EMR Clusters / S3 Buckets

- Each applications runs in its own EMR cluster
- Specific cluster = better flexibility
- Resource managed by AWS

Work in shared place

Work in my place

Fine-grain Resource Sharing

- Finer granularity by CPU/Mem
- Centralized budget account
- Do usage statistics on our own

Go Dutch & Pay-as-you-Go

- Granularity in EMR level
- Budget in each application team
- AWS provides billing reports

Finer usage management

Manage my own works

Single Big-data Stack

- Tend to select best practices that are well tested
- Other technology brings trouble
- Easy to optimize from infra-level
- Access through HDFS interface

Use verified best practice

Diverse Technology

- Tend to allow using different AWS services
- Technologies covered by AWS
- Infra optimization rely on AWS
- Different access mechanism

Use tools I like

UNIX Style Permission Control

- UNIX style “file” permissions
- Tend to make data lake read-only to everybody
- No widely adopted encryptions

Simple that just works

Cloud Compatible ACL

- IAM-based ACL policies
- Allow granting access rights in dataset level
- S3 provides native encryptions

Complex but rich

Make Accessing Data Easy

On-prem in datacenter

- Raw data is read-only to everybody
- Canonical software stack with best practices
- Plan ahead the infra by strong team

On-cloud in AWS

- Simplify permission granting process
- Encourage leveraging AWS services
- Pay-as-you-Go by every involved teams

File Format & Schema Design Considerations

- **Size reduction** to lower storage consumption
- **Read/write performance** to speed up computing
- **Data characteristics** to hold complex data structure
- **Schema evolution** which changes from time to time
- **Tool interoperability** for different kinds of access

Data Characteristics

- The **characteristics** of the data **schema**
  - What will not work?
  - The mitigations

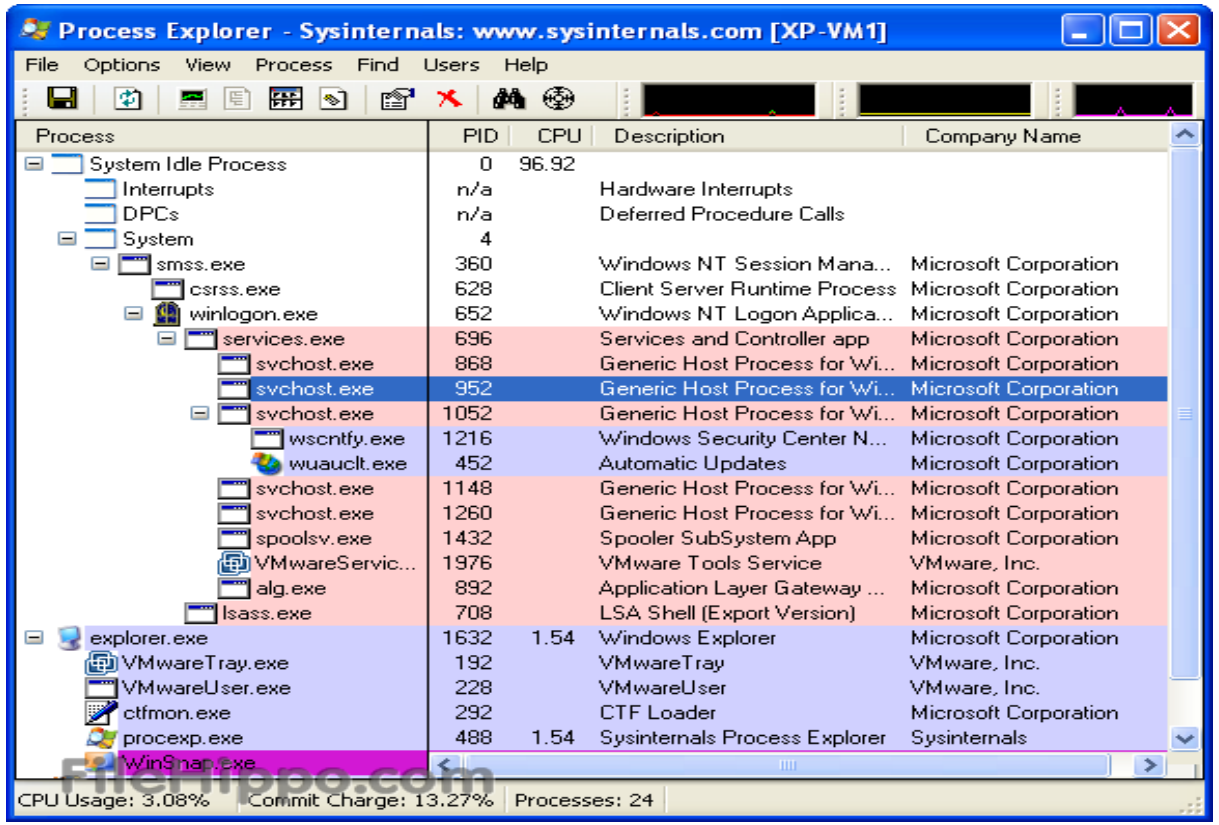
Fundamental characteristics and principles for “logs”:

- Records shall be independent – don’t reference other record
- Records shall be self-contained – repeat info in path
- Record exist means something, not-exist may mean another

These can only be documented → need **schema portal**

Data Characteristics – ~~Recursion~~

- Avoid recursive schema
- Recursive schema?



Bad Design

```
ProcessInfo {
  process_id: long,
  image_path: chararray,
  image_sha1: bytearray,
  is_detected: int,
  action: long,
  action_result: {
    terminate: chararray
  },
  children_processes: [
    ProcessInfo
  ]
}

root_process: ProcessInfo;
```


25

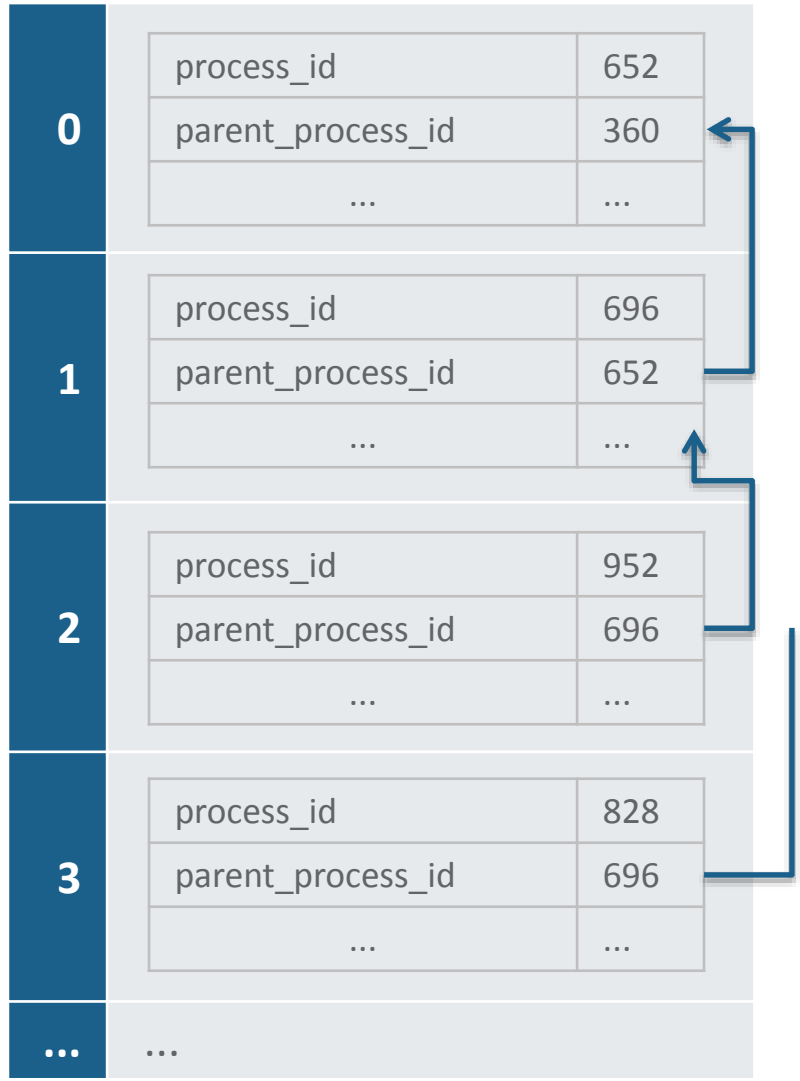
Data Characteristics – ~~Recursion~~

- Mitigation: array and index/id

Good Design

```
ProcessInfo {
  process_id: long,
  image_path: chararray,
  image_sha1: bytearray,
  is_detected: int,
  action: long,
  action_result: {
    terminate: chararray
  },
  parent_process_id: long
}

process_infos: [ ProcessInfo ];
```



26

Data Characteristics – Tabular or Nested?

- Depends on data nature
  - Raw data like feedback logs are often nested
  - Intermediate data are often tabular
- File Format Capability

Protobuf	Parquet	ORC	key/value pairs
Good at **nested** data		Good at **tabular** data	

- Strategy
  - Parquet for nested data
  - ORC for tabular data

Data Characteristics – ~~Custom Key Name~~

- Avoid custom key names for KV pairs

```
{ "key1": "value1", "key2": "value2", ... }
```

  - Runtime-determined key names are hard to process
  - Hard to parallelize key-enumeration (tool constraint)

- Mitigation

```
[
  { "name": "key1", "value": "value1" },
  { "name": "key2", "value": "value2" },
  ...
]
```


28

Data Characteristics – Not All JSON Works

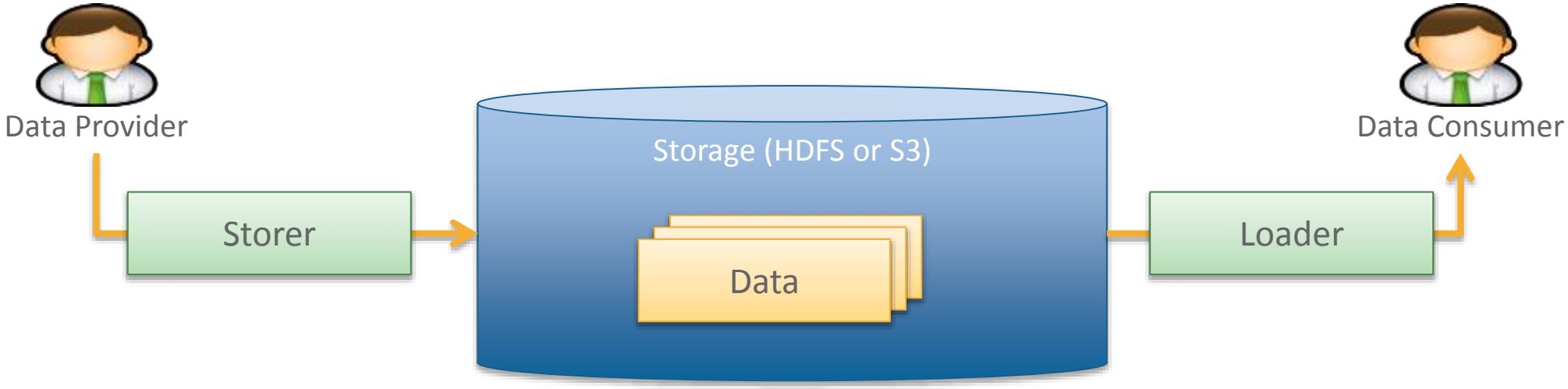
- In summary, not all data presentable by JSON are easy to process using big-data tools & other formats

Characteristic	Parquet	ORC	JSON
Recursion	No	Yes	Yes
Tabular vs. Nested	Nested	Tabular	Yes
Custom Key Name	No	No	Yes

- Be careful to design schema if you choose JSON

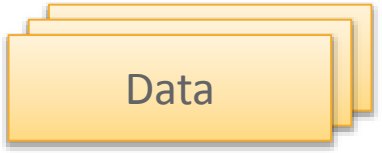
Schema Evolution

- Schema evolves from time to time
  - Backward compatible is not that simple as imaged
  - In reality each role will have multiple versions
- Roles:



Data Provider

Storer



Loader

Data Consumer

Product/v1	Validator/v1	Data/v1	Loader/v1	Service1/v1

Year 1 – Only one version in the universe

Product/v1	Validator/v2	Data/v1	Loader/v2	Service1/v1
Product/v2	Validator/v2	Data/v2	Loader/v2	Service2/v2

Year 2 – Second version appeared while v1 still exist

Product/v1	Validator/v3	Data/v1	Loader/v2	Service1/v1
Product/v2	Validator/v3	Data/v2	Loader/v2	Service2/v2
Product/v3	Validator/v3	Data/v3	Loader/v2	

Year 3 – Collect v3 data in advance due to schedule or to accumulate data

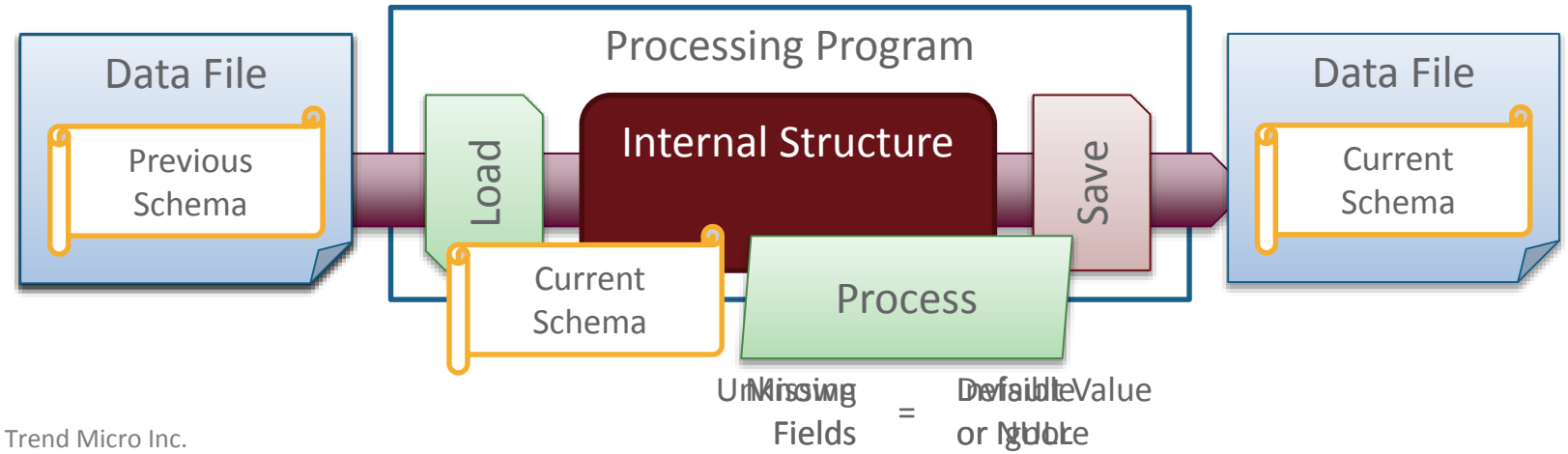
Product/v1	Validator/v3	Data/v1	Loader/v3	Service1/v1
Product/v2	Validator/v3	Data/v2	Loader/v3	
Product/v3	Validator/v3	Data/v3	Loader/v3	Service2/v3

Year 4 – Data consumer catch up and start using v3 data

31

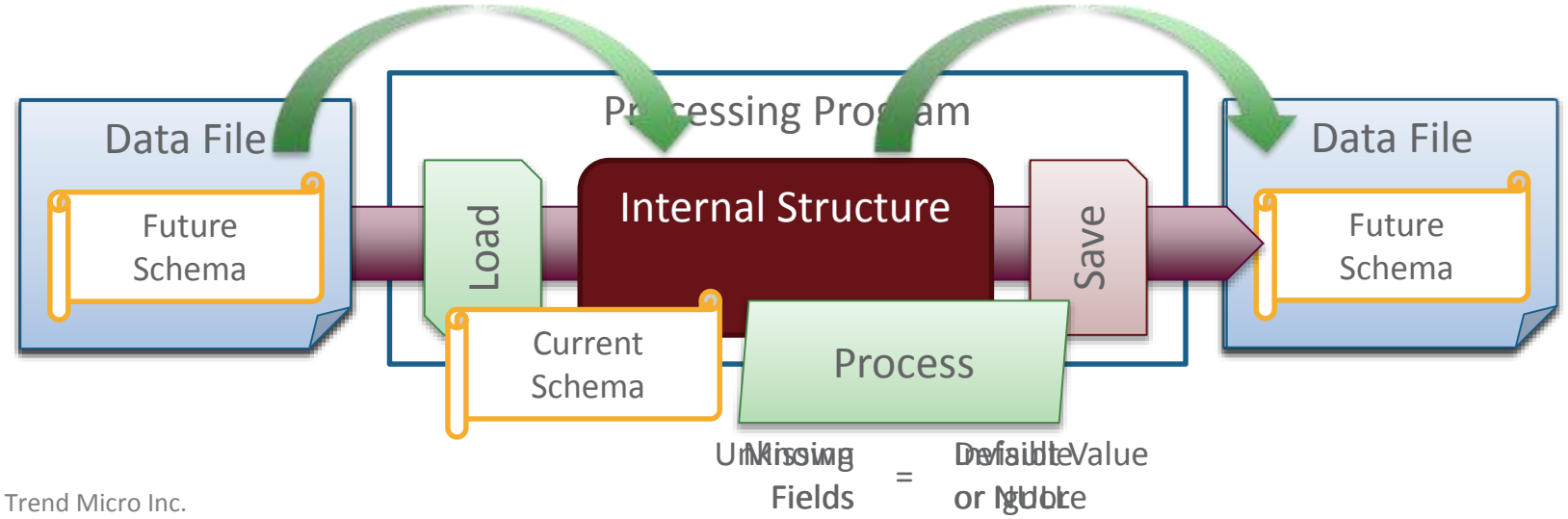
Schema Evolution – Requirements

- File format requirement to enable schema evolution
  - Mostly “Loader” requirement, for Pig and Spark
- Requirements
  - Can load current version
  - Can load previous version
  - Can load future version



Schema Evolution – Requirements

- File format requirement to enable schema evolution
  - Mostly “Loader” requirement, for Pig and Spark
- Requirements
  - Can load current version
  - Can load previous version
  - Can load future version
  - **Store as original version** when collecting data



33

Schema Evolution – Requirements

- File format requirement to enable schema evolution
  - Mostly “Loader” requirement, for Pig and Spark
- Requirements

Requirement	SF+PB	Parquet	ORC	SF+PB	Parquet	ORC	Parquet	ORC
Load Previous	v	v	v	v	v	v	v	v
Load Current	v	v	v	v	v	v	v	v
Load Future	v	v	v	v	v	v	v	v
Save Original	v	v	v	x	x	x	x	x
Language	Java			Pig			Spark	

  - The internal structure of loaded data in Pig and Spark cannot preserve entire original structure
  - Write data ingestion tool in Java whenever possible

34

Preserve low-level visibility

- Choose file format based on usage scenario
- Deview schema to avoid bad design

Wrap ups

- Preserve low-level visibility
  - Choose file format wisely
  - Design schema carefully
- Make accessing data easy
  - On-prem & on-cloud are different
  - Do something to lower barrier


Any Questions?