The Evolution of Data Infrastructure at Splunk

Flink Forward SF/Virtual 2020

Eric Sammer - VP, Distinguished Engineer

esammer@splunk.com

splunk> turn data into doing™

What is Splunk?

A platform for the collection, storage, query, and analysis of event and time series data.

Logs, but other kinds of events too

Tons of query-time processing features

Core platform experience, increasingly domain-specific applications

Classes of Workload

Ad hoc query - Human and apps

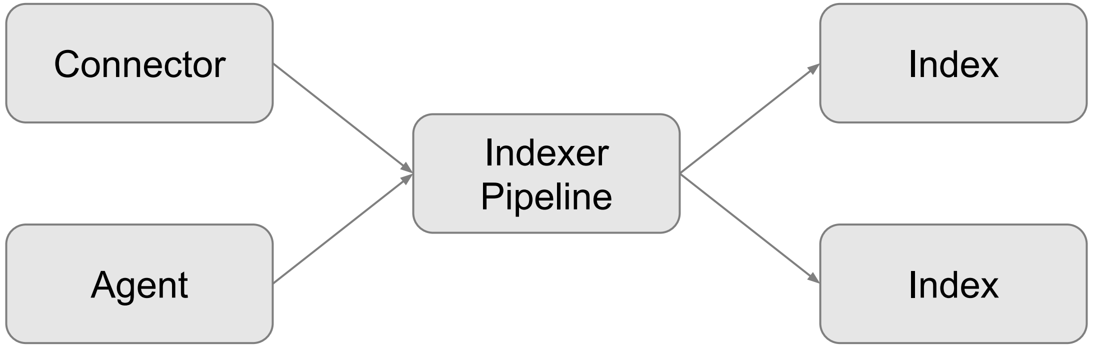
Scheduled query - Materialized view maintenance, app processes, alerting

Realtime query - Ad hoc (“Live tail”), app processes

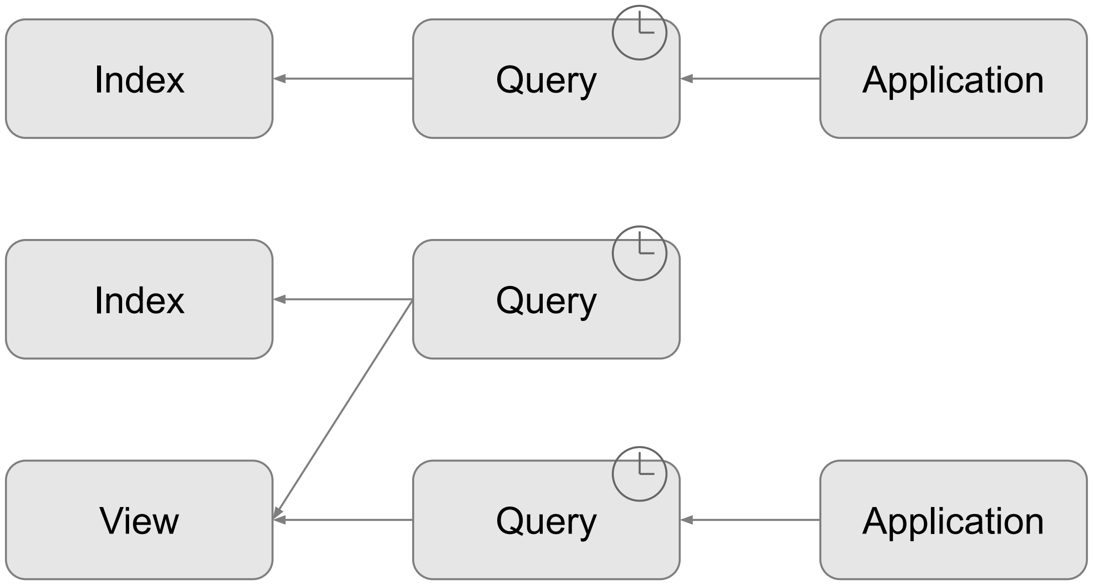
> +

two and a half years ago...

All data lands in an index



Applications and platform services use scheduled queries to build views or process data



Challenges

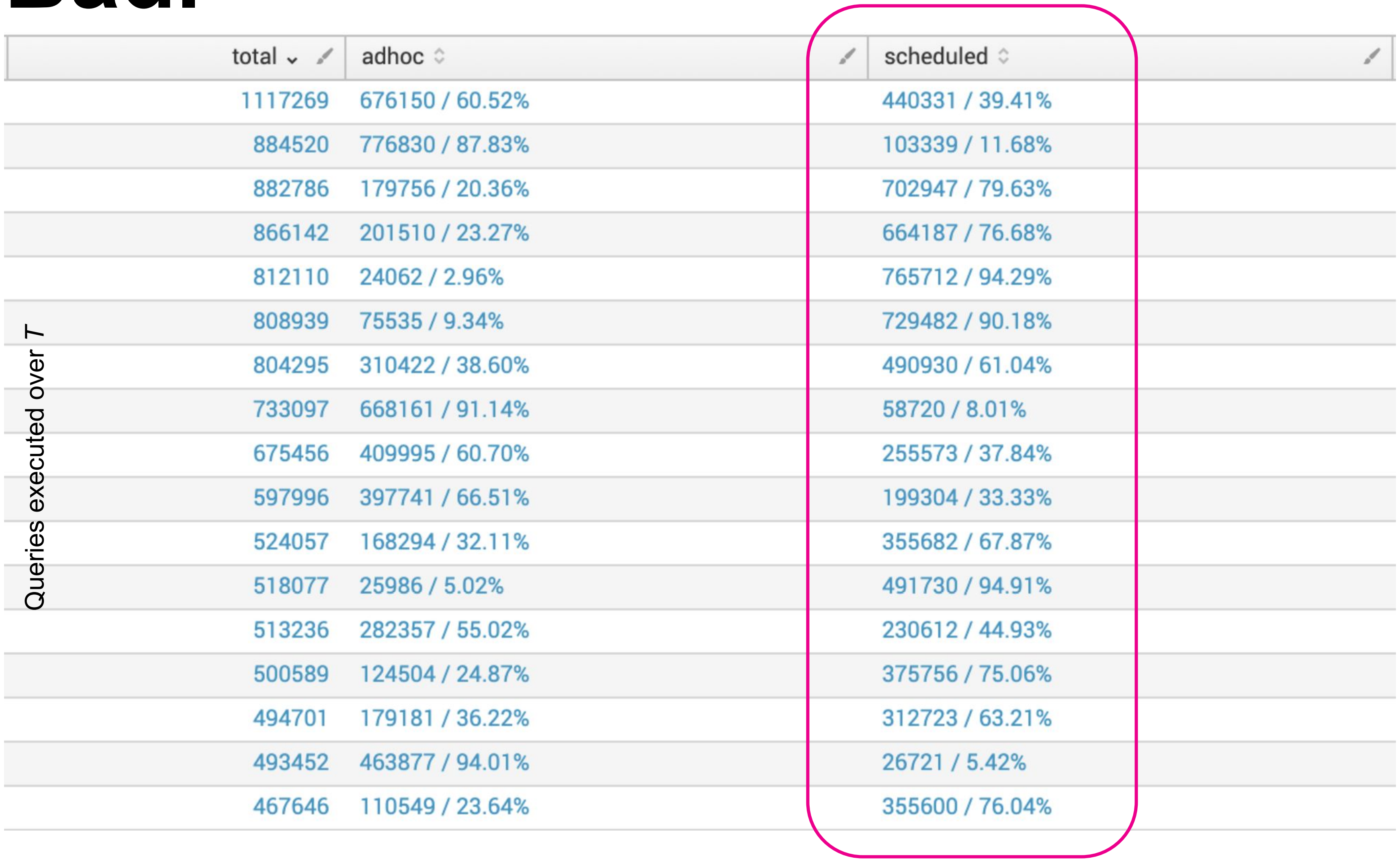
Latency

Polling query workload overhead - View maintenance, alerts / triggers, application processing

Insufficient programming model - State management, failure and recovery semantics, parallelization, optimization, orchestration left to applications in many cases

Polling query overhead: how bad is it, really?

Bad.



Queries executed over *T*

total	adhoc	scheduled
1117269	676150 / 60.52%	440331 / 39.41%
884520	776830 / 87.83%	103339 / 11.68%
882786	179756 / 20.36%	702947 / 79.63%
866142	201510 / 23.27%	664187 / 76.68%
812110	24062 / 2.96%	765712 / 94.29%
808939	75535 / 9.34%	729482 / 90.18%
804295	310422 / 38.60%	490930 / 61.04%
733097	668161 / 91.14%	58720 / 8.01%
675456	409995 / 60.70%	255573 / 37.84%
597996	397741 / 66.51%	199304 / 33.33%
524057	168294 / 32.11%	355682 / 67.87%
518077	25986 / 5.02%	491730 / 94.91%
513236	282357 / 55.02%	230612 / 44.93%
500589	124504 / 24.87%	375756 / 75.06%
494701	179181 / 36.22%	312723 / 63.21%
493452	463877 / 94.01%	26721 / 5.42%
467646	110549 / 23.64%	355600 / 76.04%

This cohort:

p99: 94.91%
p90: 94.29%
p75: 79.63%
p50: 63.21%

Anonymized real deployments.

If we could ... we'd get ...

Move view building, alerting, and app processing to the ingest stream

- Up to 94% query workload reduction / more time for ad hoc queries
- Reduced latency to data visibility / faster time to act
- Safer, more consistent and efficient programming model for developers

If we could ... we’d get ...

Let developers / end users tap into the ingest stream

- Deeper integration opportunities with other data infra
- New ingest-time extension points / support for more use cases

If we could ... we’d get ...

Support additional ingest-time functionality

- Better data quality downstream
- Better data governance and control opportunities
- Better DR/BCP/scaling primitives and function

What would Splunk look like if we added stream processing as a platform primitive?

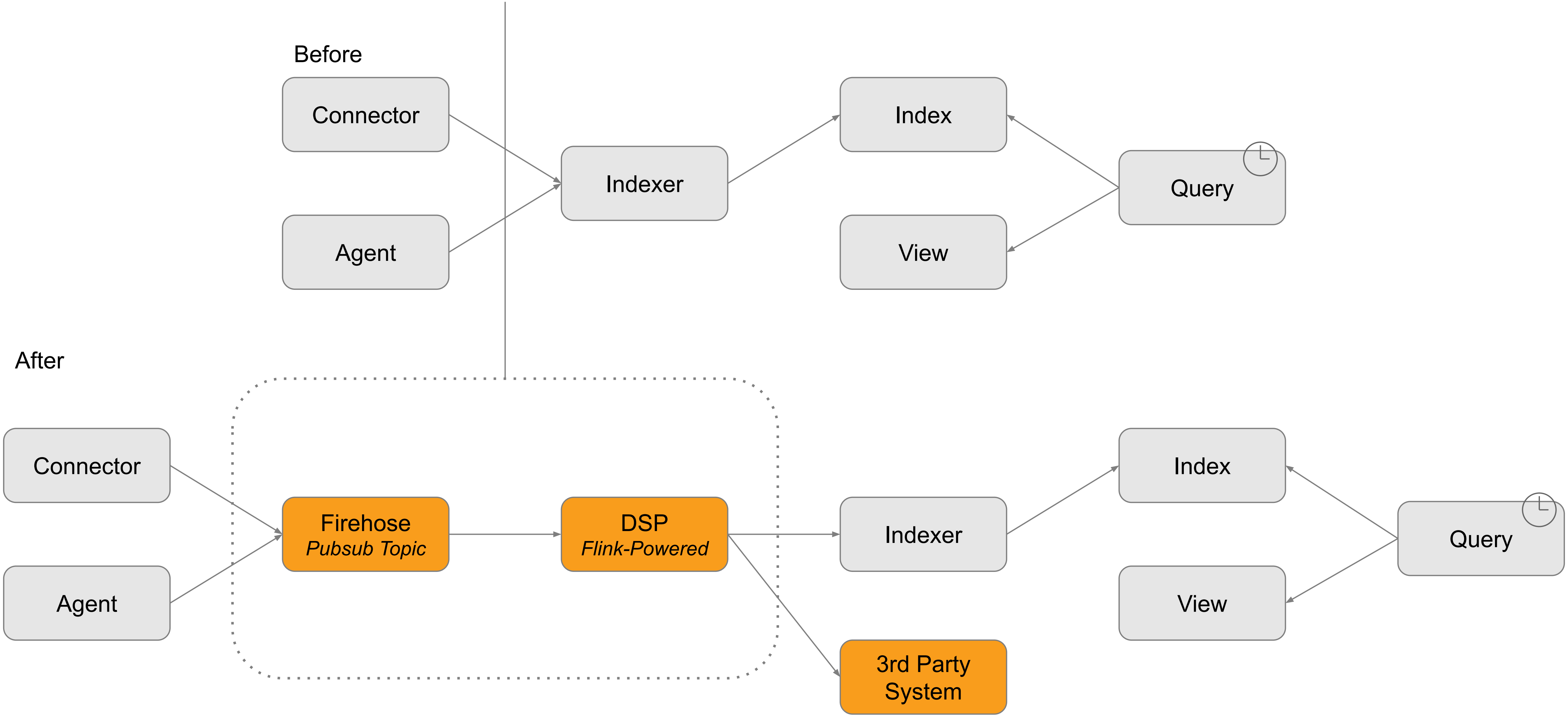
phase 1: ingest

Focus on primitives and ingestion process

Optional - support incremental adoption by on prem customers

Minimal disruption to current semantics (only improvements)

Easy wins



Before
Connector
Agent
Indexer
Index
View
Query
After
Connector
Agent
Firehose
Pubsub Topic
DSP
Flink-Powered
Indexer
Index
View
Query
3rd Party
System

Streaming Infrastructure

Data Stream Processor (DSP) is Splunk's Flink-powered stream processing engine

All connectors produce to the *firehose* pubsub topic in a common envelope

DSP has both system- and user-configured *pipelines* that process data

- Pipeline = Source(s) → Function(s) → Sink(s)
- Parse, route, enrich, filter, aggregate, join, union, branch, etc.
- Functions can be written in DSL or "native" SDK

The Firehose

Common envelope for all data

“Body” can be any data type, “sourcetype” provides semantic information

Firehose represented as a source in DSP

- More specific sources are firehose + filter

Pipeline Patterns

“Upgrade” of semantic information

“Routing” implemented as branches into filter + sink pairs

“External functions” use queues to produce to / consume from external systems

StateFun 2.0 is more interesting though - adds state mgmt, removes overhead

“Group functions” are fragments of a DAG that can be inlined

Function-level RBAC + group functions = efficiently limit data access

Phase 1 gives us...

Deeper integration opportunities with other data infra

New ingest-time extension points / support for more use cases

Better data quality downstream

Better data governance and control opportunities

Better DR/BCP/scaling primitives and function

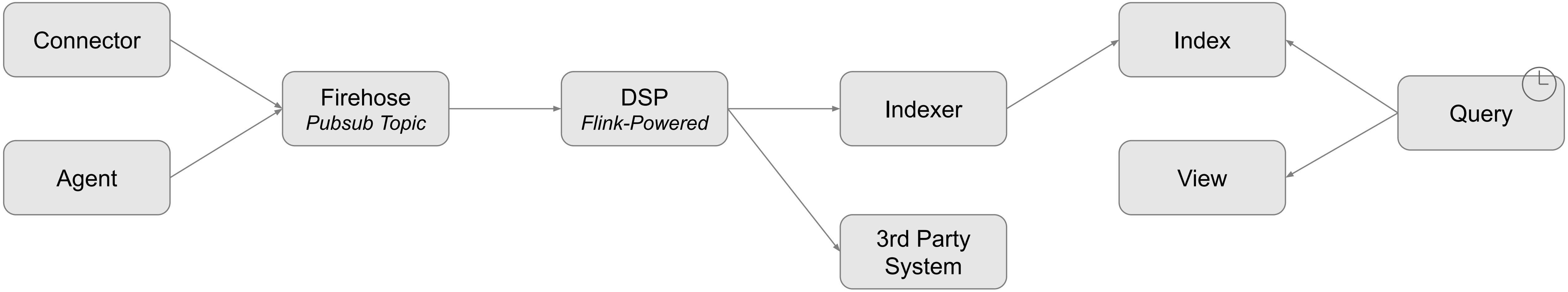
phase 2: move processing upstream

Push polling operations upstream (“to the left”)

Allow developers to migrate batch processes to the stream, in part or whole

Make a number of platform functions (e.g. alerting) streaming by default

Before



Connector

Agent

Firehose
Pubsub Topic

DSP
Flink-Powered

Indexer

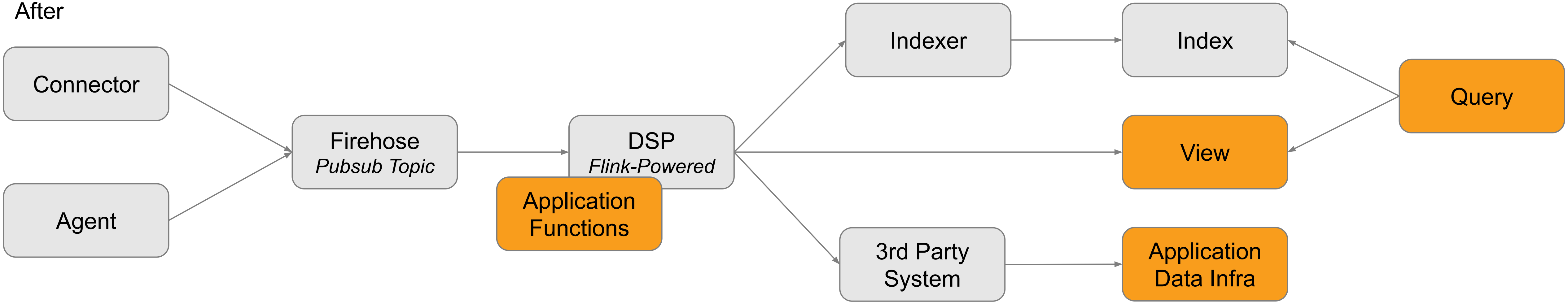
3rd Party
System

Index

View

Query

After



Connector

Agent

Firehose
Pubsub Topic

DSP
Flink-Powered

Application
Functions

Indexer

3rd Party
System

Index

View

Application
Data Infra

Query

Critial enablers

DSP uses a dynamic function registry to load plugins

Pre-built content system for applications to add pipeline templates, functions

Pubsub firehose lets applications see whatever they need

Auto-scale Flink clusters based on slot availability

Phase 2 gives us...

Up to 94% query workload reduction / more time for ad hoc queries

Reduced latency to data visibility / faster time to act

Safer, more consistent and efficient programming model for developers

Even more

- Deeper integration opportunities with other data infra
- New ingest-time extension points / support for more use cases

phase 3: realtime query

In phase 2 we moved application workloads; this is about ad hoc realtime queries

Splunk has realtime query today, but there are some gotchas

- Computationally expensive (but still better than many systems)
- Complex data guarantees

Can we make it cheap to run thousands of concurrent realtime queries?

- With rapid start/stop?

I’ll let you know when I do. Sorry.

adding it up

When working with data, streams are the right primitive

Concrete semantics, patterns, and architecture are key

The details are important (e.g. state management)

As patterns evolve, so must we

Thank You

esammer@splunk.com | @esammer

splunk> turn data into doing™