PURESTORAGE®

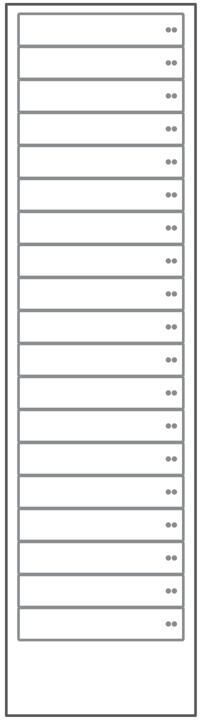
Advanced Analytics & AI

Semih Eskin
Principal Technologist

Data Lakehouse

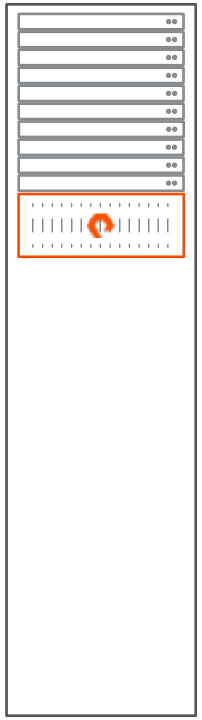
Storage Disaggregation

DIRECT ATTACHED STORAGE



INDEPENDENTLY Scale Compute & Storage
REDUCE Infrastructure Expense
INCREASED Compute Performance
DECREASE Software Spend
FASTER Recovery Time
CONSISTENT Application Performance

DISAGGREGATED STORAGE AND COMPUTE



Three Key Components in a Data Lakehouse

A simplified view of how to implement an open data lakehouse

01

Data Lake

Leverage current data lake and open data format.

02

Metadata Layer

Add metadata layer for advanced data management.

03

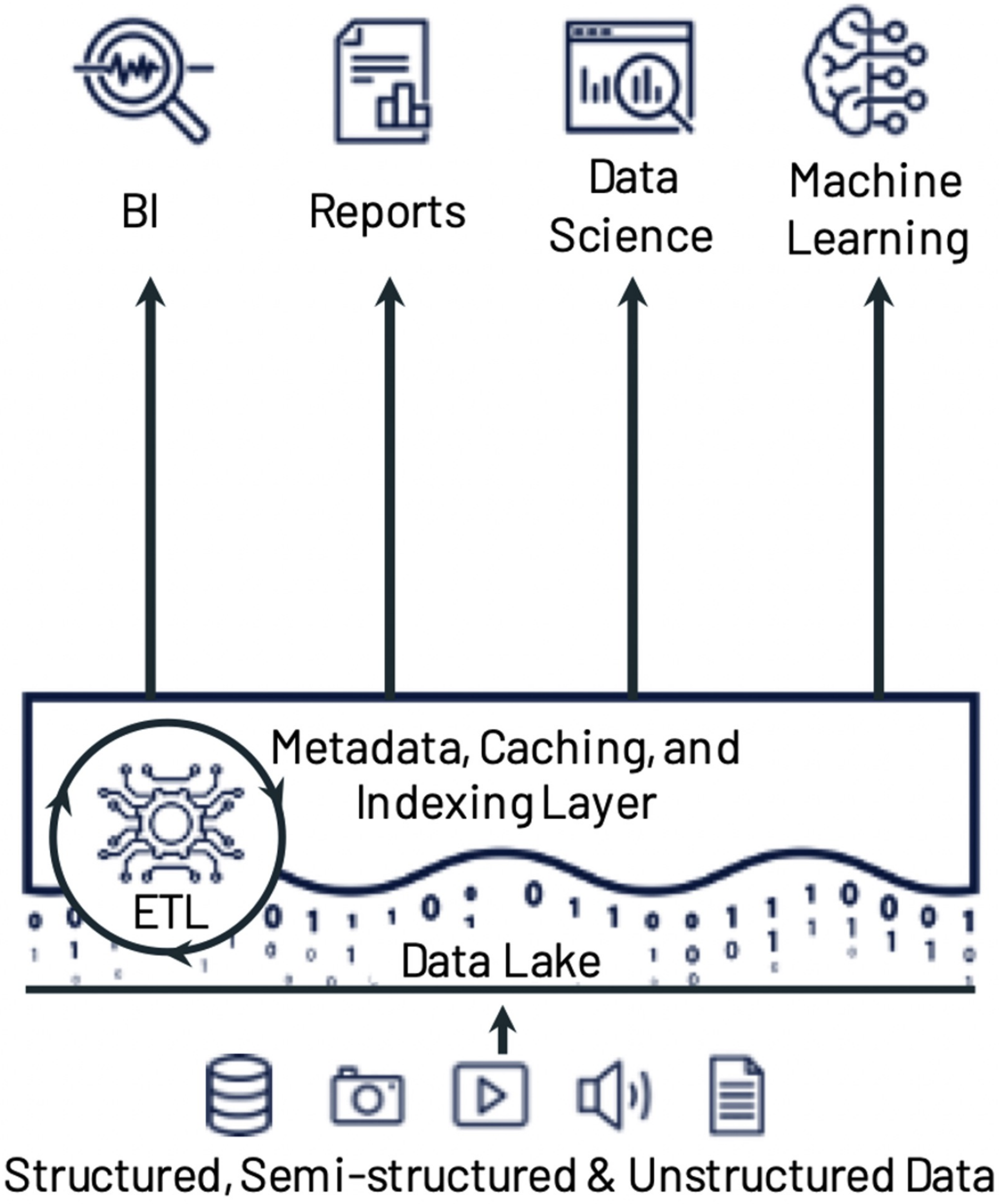
Processing Engine

Use processing engine that understands the lakehouse spec.

The Data Lakehouse Architecture

Unites the strength of data lakes and data warehouse

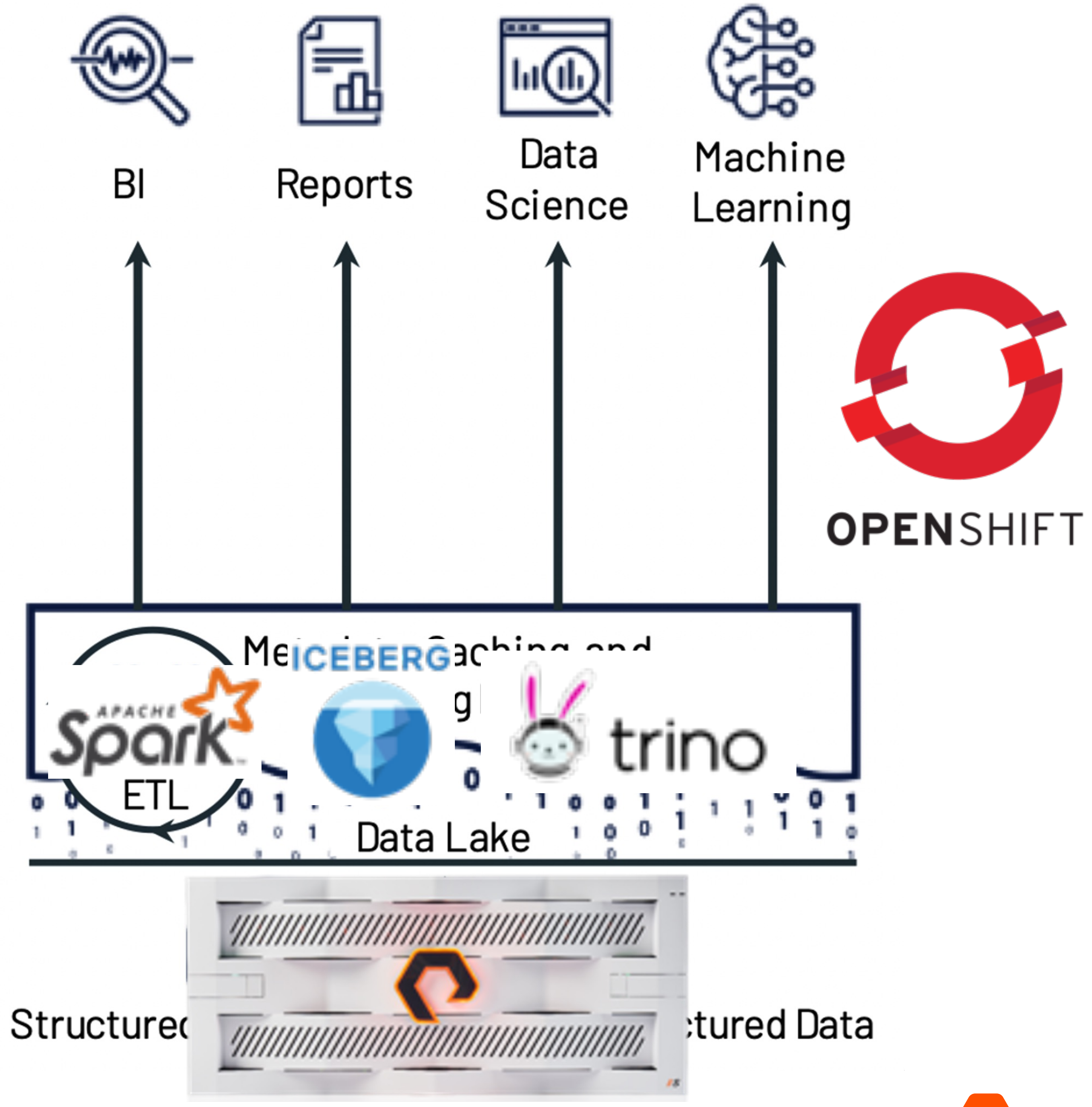
- Supports ETL, SQL, and ML workloads
- In a single system



The Data Lakehouse Architecture

Unites the strength of data lakes and data warehouse

- Supports ETL, SQL, and ML workloads
- In a single system



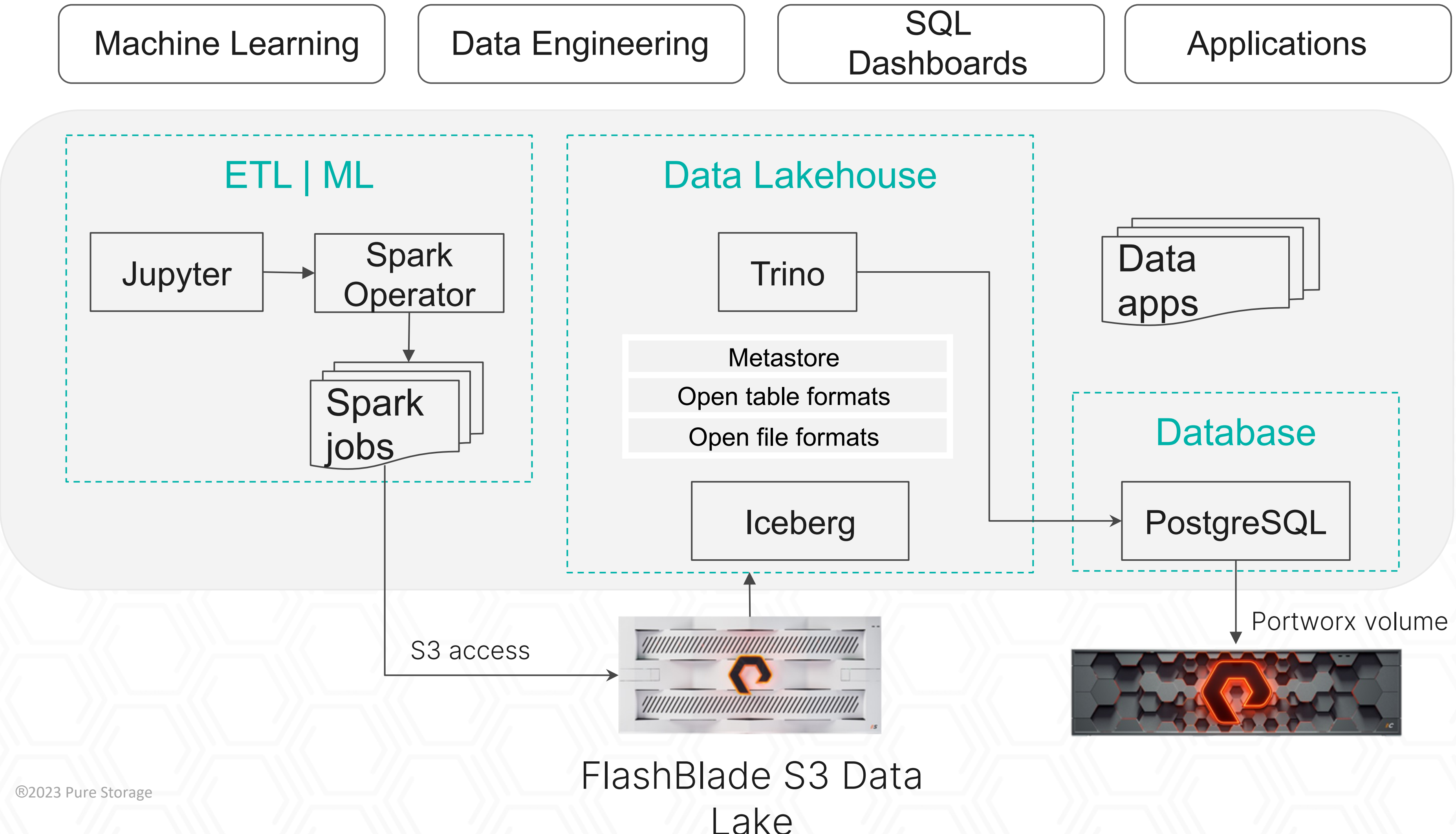
Recap - Three Key Components in a Data Lakehouse

Build an open data lakehouse with high performance object storage.

01	02	03
Data lake Leverage current data lake and open data format	**Metadata layer** Add metadata layer for advanced data management	**Processing engine** Use processing engine that understands the lakehouse spec
High performance object storage	**Delta Lake, Apache Iceberg**	**Apache Spark, Trino, Dremio, Vertica, Snowflake**

Open Data Lakehouse in Action

An example architecture for simple, fast and open data lakehouse with **Pure Stora**



- No lock-in
- Inexpensive
- Fast data exploration
- Simple
- Cloud ready

AI

…AI is more than just the Model

"Hidden Technical Debt in Machine Learning Systems", Google NIPS 2015

Hidden Technical Debt in Machine Learning Systems

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips
{dsculley,gholt,dgg,edavydov,toddphillips}@google.com
Google, Inc.

Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, Dan Dennison
{ebner,vchaudhary,mwyoung,jfcrespo,dennison}@google.com
Google, Inc.

Abstract

Machine learning offers a fantastically powerful toolkit for building useful complex prediction systems quickly. This paper argues it is dangerous to think of these quick wins as coming for free. Using the software engineering framework of *technical debt*, we find it is common to incur massive ongoing maintenance costs in real-world ML systems. We explore several ML-specific risk factors to account for in system design. These include boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, configuration issues, changes in the external world, and a variety of system-level anti-patterns.

1 Introduction

As the machine learning (ML) community continues to accumulate years of experience with live systems, a wide-spread and uncomfortable trend has emerged: developing and deploying ML systems is relatively fast and cheap, but maintaining them over time is difficult and expensive.

This dichotomy can be understood through the lens of *technical debt*, a metaphor introduced by Ward Cunningham in 1992 to help reason about the long term costs incurred by moving quickly in software engineering. As with fiscal debt, there are often sound strategic reasons to take on technical debt. Not all debt is bad, but all debt needs to be serviced. Technical debt may be paid down by refactoring code, improving unit tests, deleting dead code, reducing dependencies, tightening APIs, and improving documentation [8]. The goal is *not* to add new functionality, but to enable future improvements, reduce errors, and improve maintainability. Deferring such payments results in compounding costs. Hidden debt is dangerous because it compounds silently.

In this paper, we argue that ML systems have a special capacity for incurring technical debt, because they have all of the maintenance problems of traditional code plus an additional set of ML-specific issues. This debt may be difficult to detect because it exists at the *system* level rather than the code level. Traditional abstractions and boundaries may be subtly corrupted or invalidated by the fact that data influences ML system behavior. Typical methods for paying down code level technical debt are not sufficient to address ML-specific technical debt at the system level.

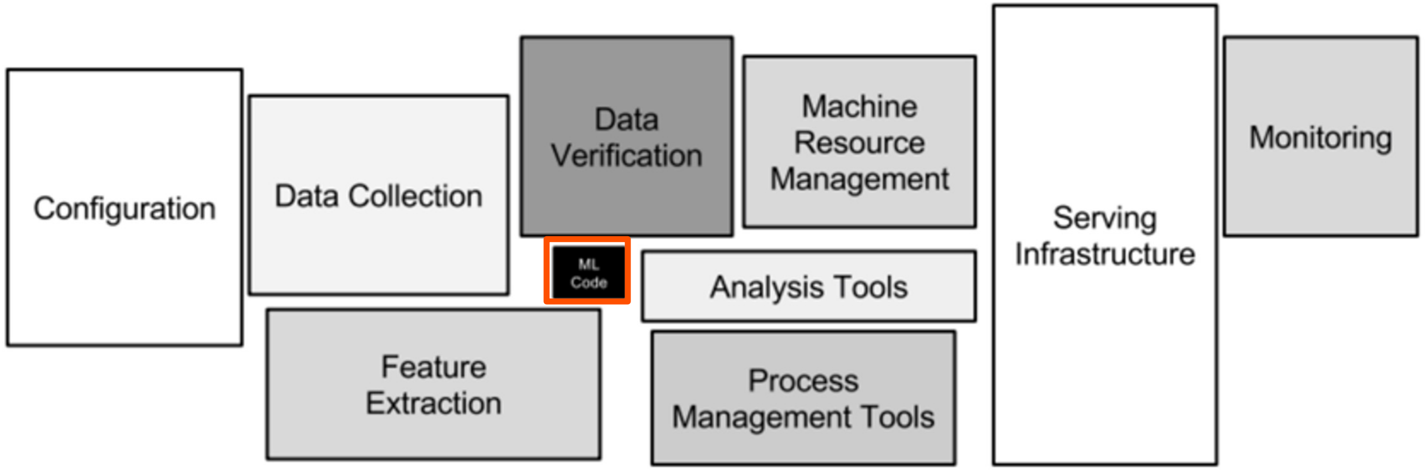
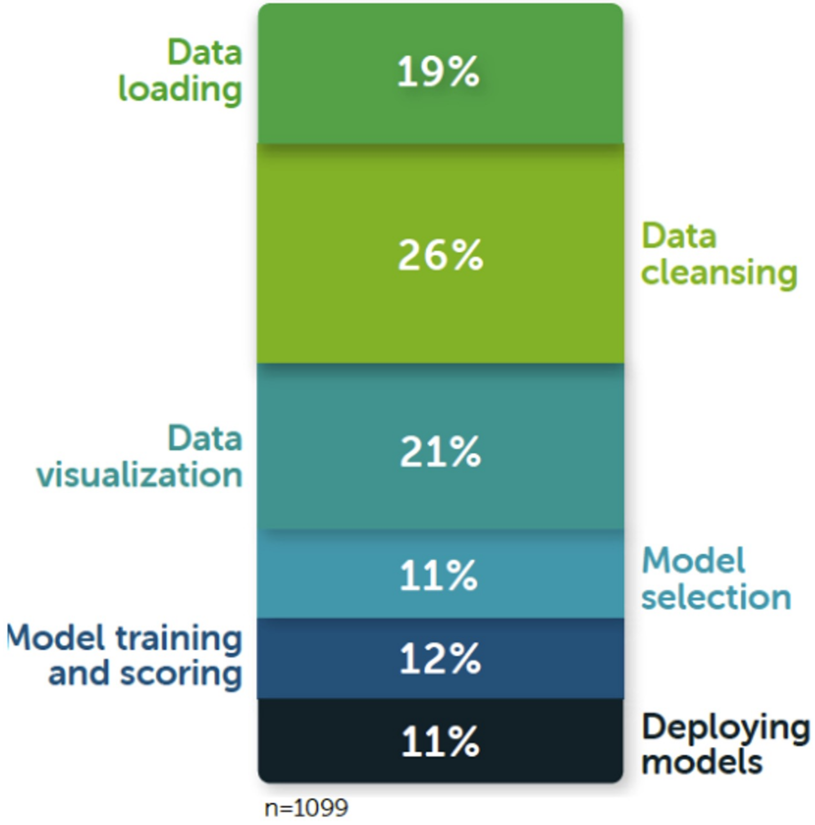


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

So why should AI care about Pure Storage

Time to science



How data scientists spend their time (Image courtesy Anaconda "2020 State of Data Science: Moving From Hype Toward Maturity.")