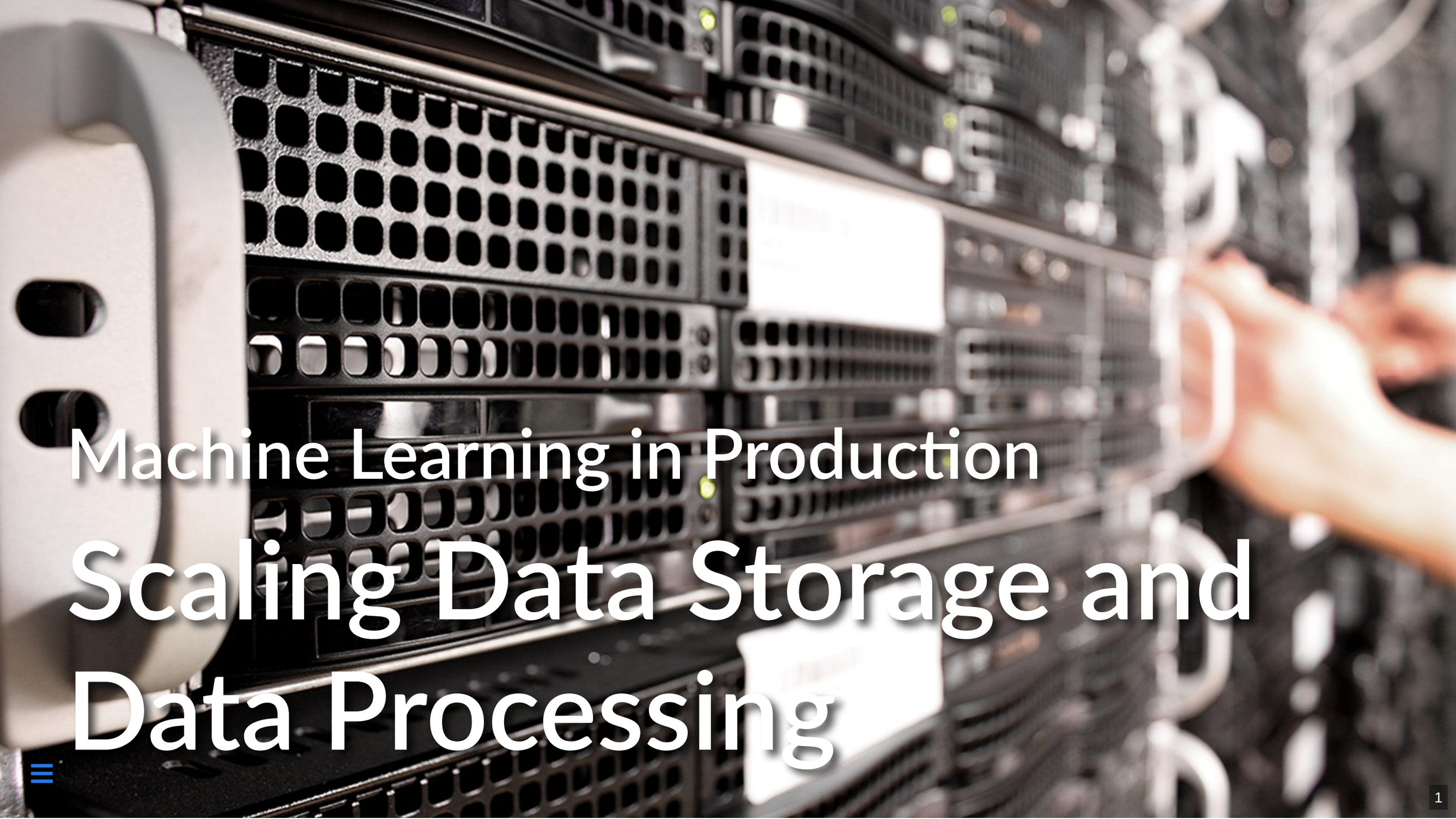


Machine Learning in Production

Scaling Data Storage and Data Processing

Design and operations

Fundamentals of Engineering AI-Enabled Systems

Holistic system view: AI and non-AI components, pipelines, stakeholders, environment interactions, feedback loops

Requirements:
System and model goals
User requirements
Environment assumptions
Quality beyond accuracy
Measurement
Risk analysis
Planning for mistakes

Architecture + design:
Modeling tradeoffs
Deployment architecture
Data science pipelines
Telemetry, monitoring
Anticipating evolution
Big data processing
Human-AI design

Quality assurance:
Model testing
Data quality
QA automation
Testing in production
Infrastructure quality
Debugging

Operations:
Continuous deployment
Contin. experimentation
Configuration mgmt.
Monitoring
Versioning
Big data
DevOps, MLOps

Teams and process: Data science vs software eng. workflows, interdisciplinary teams, collaboration points, technical debt

Responsible AI Engineering

Provenance, versioning, reproducibility

Safety

Security and privacy

Fairness

Interpretability and explainability

Transparency and trust

Ethics, governance, regulation, compliance, organizational culture

Readings

Required reading: Nathan Marz. Big Data: Principles and best practices of scalable realtime data systems. Simon and Schuster, 2015. Chapter 1: A new paradigm for Big Data

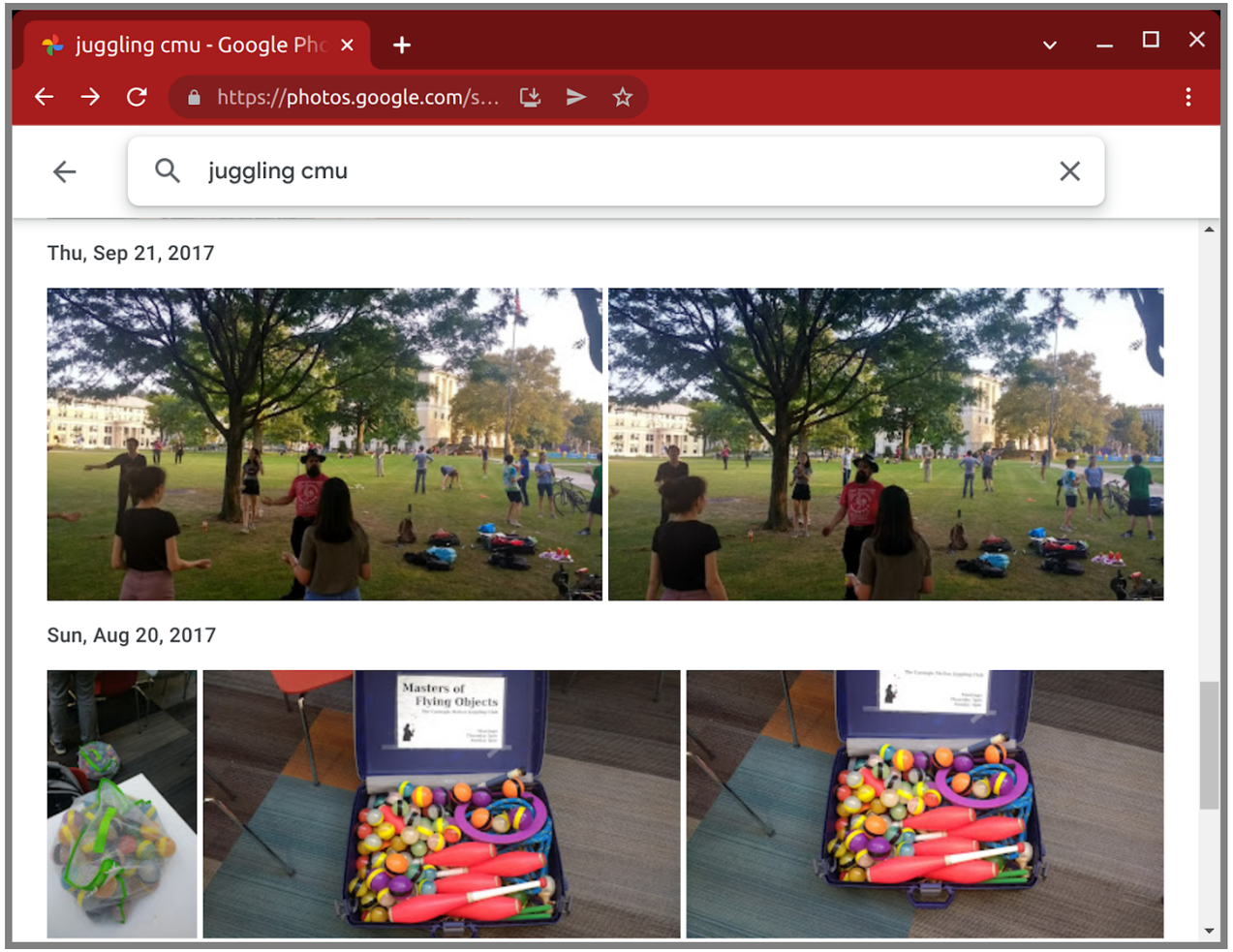
Suggested watching: Molham Aref. Business Systems with Machine Learning. Guest lecture, 2020.

Suggested reading: Martin Kleppmann. Designing Data-Intensive Applications. OReilly. 2017.

Learning Goals

- Organize different data management solutions and their tradeoffs
- Understand the scalability challenges involved in large-scale machine learning and specifically deep learning
- Explain the tradeoffs between batch processing and stream processing and the lambda architecture
- Recommend and justify a design and corresponding technologies for a given system

Case Study


juggling cmu - Google Pho
https://photos.google.com/s...
juggling cmu
Thu, Sep 21, 2017
Sun, Aug 20, 2017
Masters of
Flying Objects

Speaker notes

- Discuss possible architecture and when to predict (and update)
- in may 2017: 500M users, uploading 1.2billion photos per day (14k/sec)
- in Jun 2019 1 billion users

Adding capacity

Stories of catastrophic success?

Data Management and Processing in ML-Enabled Systems

Kinds of Data

- Training data
- Input data
- Telemetry data
- (Models)

all potentially with huge total volumes and high throughput

need strategies for storage and processing

Data Management and Processing in ML-Enabled Systems

Store, clean, and update training data

Learning process reads training data, writes model

Prediction task (inference) on demand or precomputed

Individual requests (low/high volume) or large datasets?

Often both learning and inference data heavy, high volume tasks

Scaling Computations

Efficent Algorithms

Faster Machines

More Machines

Distributed Everything

Distributed data cleaning

Distributed feature extraction

Distributed learning

Distributed large prediction tasks

Incremental predictions

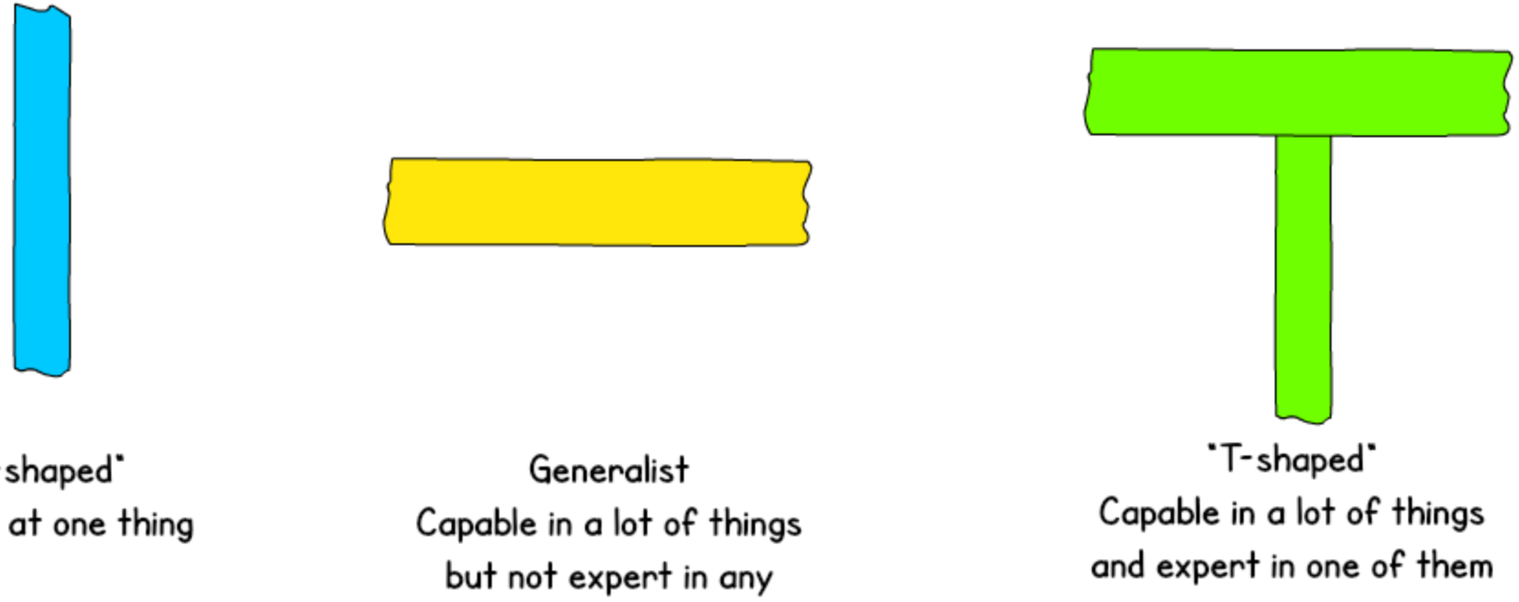
Distributed logging and telemetry

Reliability and Scalability Challenges in AI-Enabled Systems?

Distributed Systems and AI-Enabled Systems

- Learning tasks can take substantial resources
- Datasets too large to fit on single machine
- Nontrivial inference time, many many users
- Large amounts of telemetry
- Experimentation at scale
- Models in safety critical parts
- Mobile computing, edge computing, cyber-physical systems

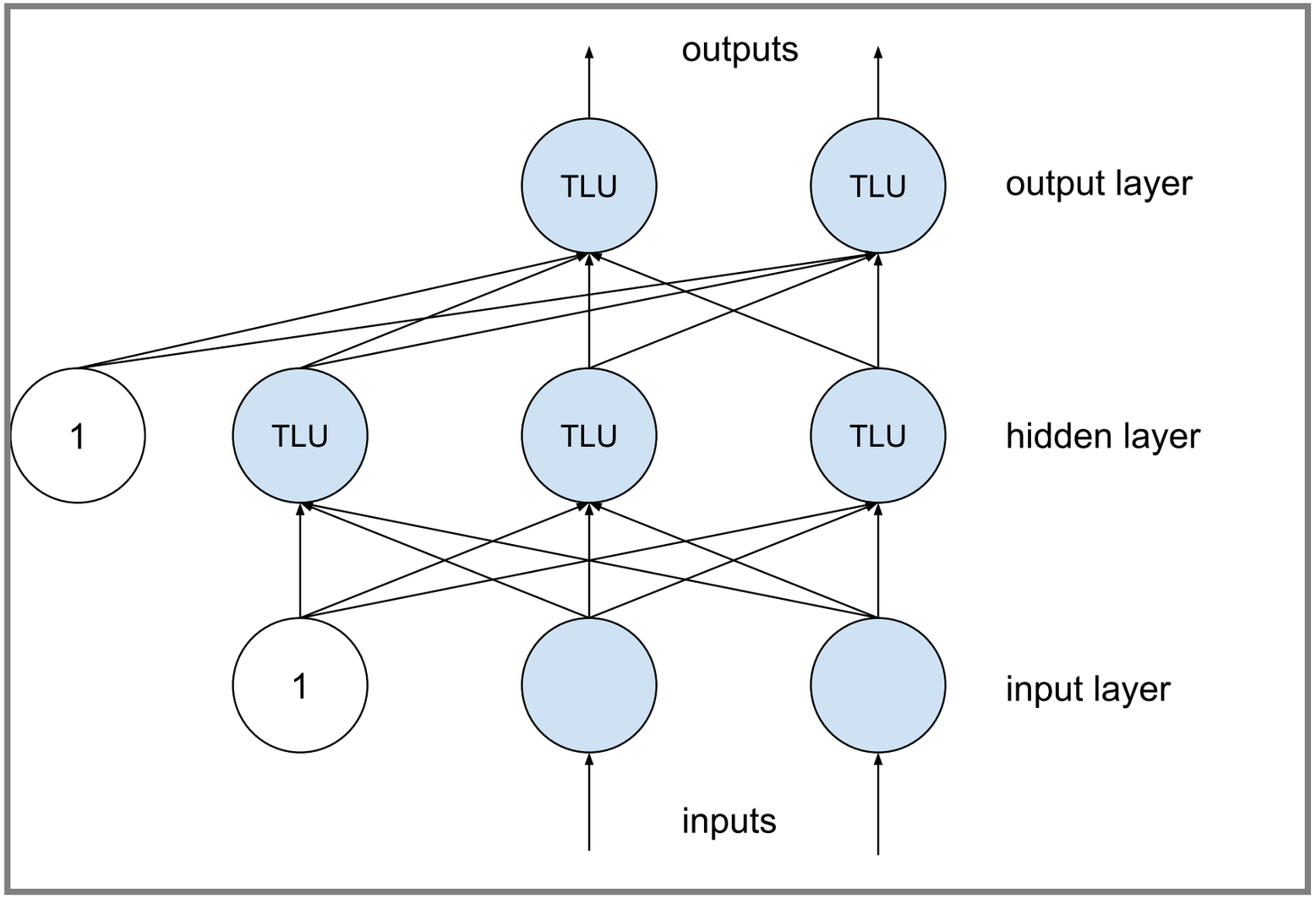
Reminder: T-Shaped People



Go deeper with: Martin Kleppmann. Designing Data-Intensive Applications. OReilly. 2017.

Excursion: Distributed Deep Learning with the Parameter Server Architecture

Recall: Backpropagation



Training at Scale is Challenging

Already 2012 at Google: 1TB-1PB of training data, $10^9 - 10^{12}$ parameters

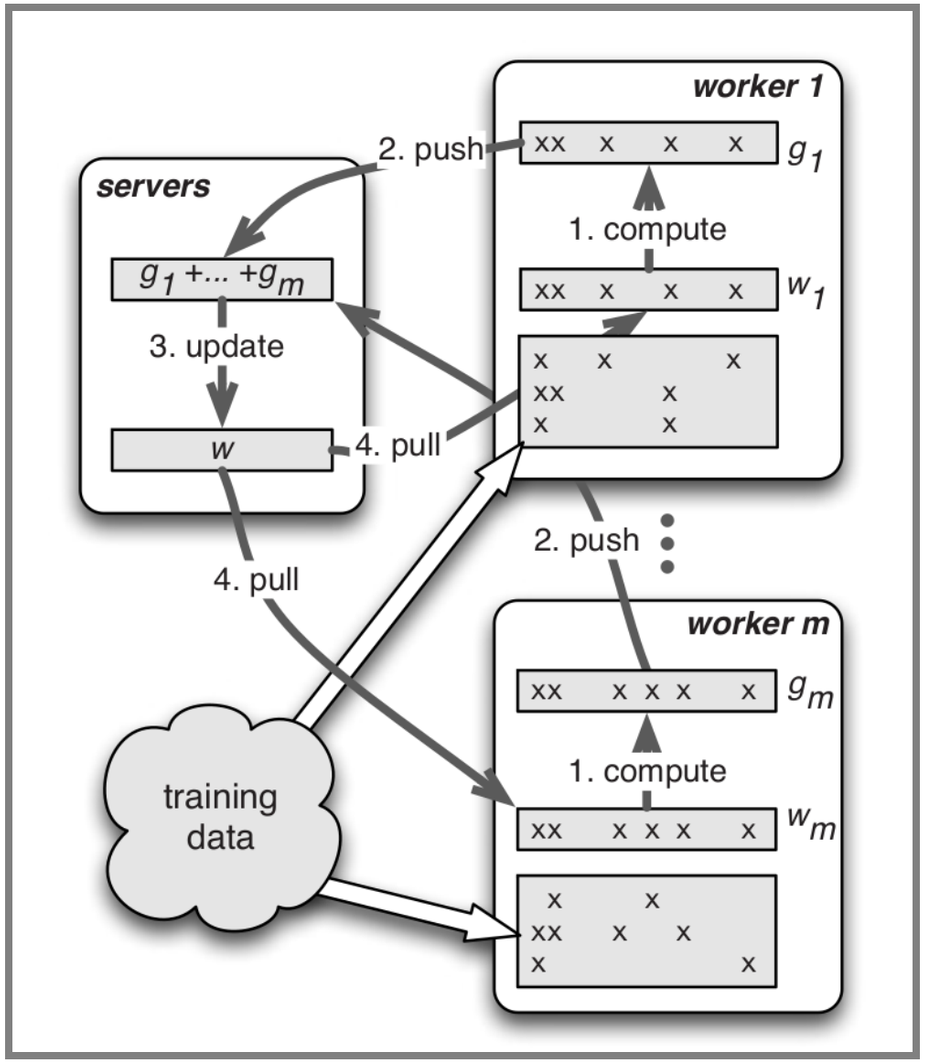
Need distributed training; learning is often a sequential problem

Just exchanging model parameters requires substantial network bandwidth

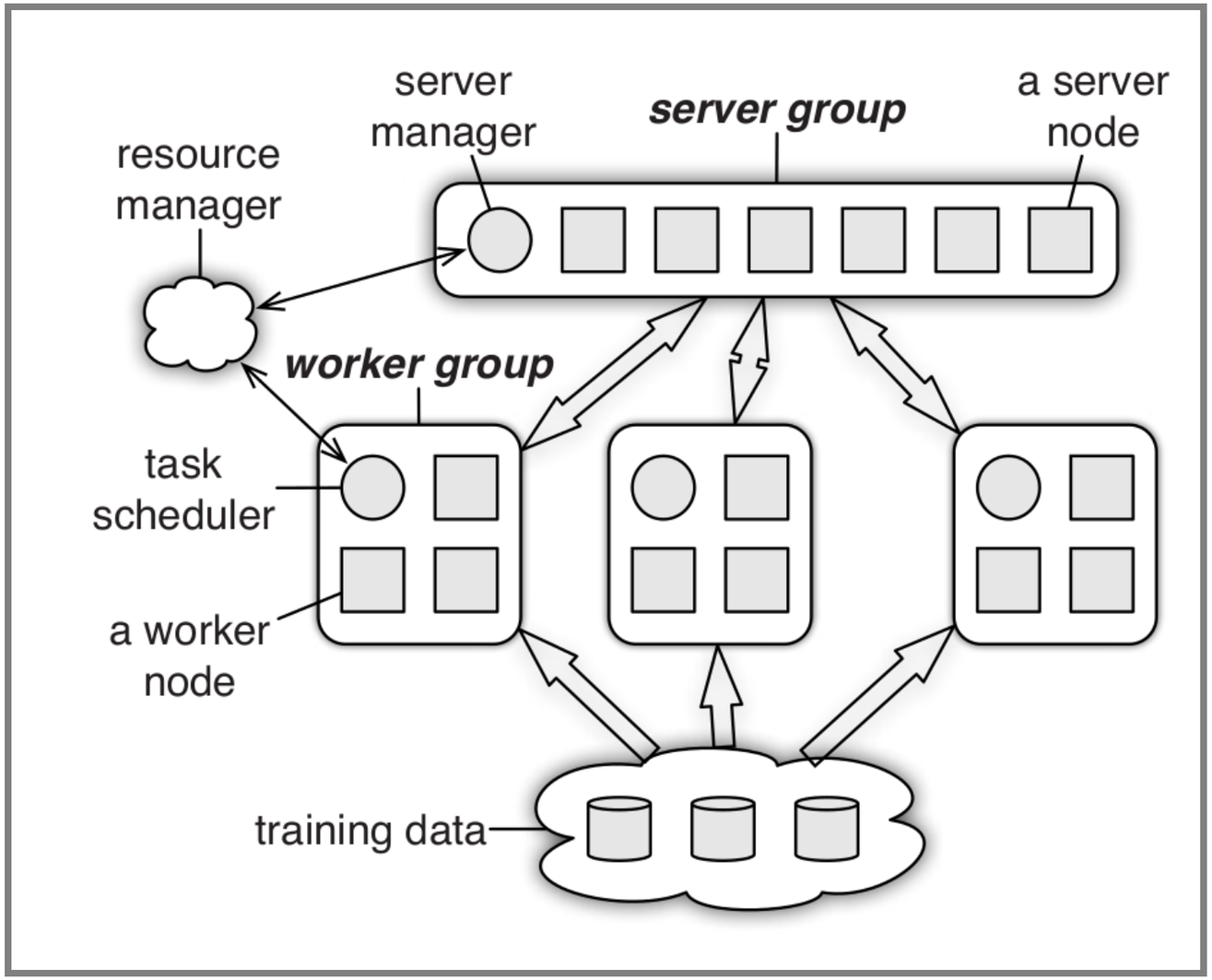
Fault tolerance essential (like batch processing), add/remove nodes

Tradeoff between convergence rate and system efficiency

Distributed Gradient Descent



Parameter Server Architecture



Speaker notes

Multiple parameter servers that each only contain a subset of the parameters, and multiple workers that each require only a subset of each

Ship only relevant subsets of mathematical vectors and matrices, batch communication

Resolve conflicts when multiple updates need to be integrated (sequential, eventually, bounded delay)

Run more than one learning algorithm simulaneously

SysML Conference

Increasing interest in the systems aspects of machine learning

e.g., building large scale and robust learning infrastructure

https://mlsys.org/

Data Storage Basics

Relational vs document storage

1:n and n:m relations

Storage and retrieval, indexes

Query languages and optimization

Relational Data Models

Photos:

photo_id	user_id	path	upload_date	size	camera_id	camera_setting
133422131	54351	/st/u211/1U6uFl47Fy.jpg	2021-12-03T09:18:32.124Z	5.7	663	ƒ/1.8; 1/120; 4.44mm; ISO271
133422132	13221	/st/u11b/MFxlL1FY8V.jpg	2021-12-03T09:18:32.129Z	3.1	1844	ƒ/2, 1/15, 3.64mm, ISO1250
133422133	54351	/st/x81/ITzhcSmv9s.jpg	2021-12-03T09:18:32.131Z	4.8	663	ƒ/1.8; 1/120; 4.44mm; ISO48

Users:

user_id	account_name	photos_total	last_login
54351	ckaestne	5124	2021-12-08T12:27:48.497Z
13221	eva.burk	3	2021-12-21T01:51:54.713Z

Cameras:

camera_id	manufacturer	print_name
663	Google	Google Pixel 5
1844	Motorola	Motorola MotoG3

```
select p.photo_id, p.path, u.photos_total
from photos p, users u
where u.user_id=p.user_id and u.account_name = "ckaestne"
```

Document Data Models

```
{
    "_id": 133422131,
    "path": "/st/u211/1U6uFl47Fy.jpg",
    "upload_date": "2021-12-03T09:18:32.124Z",
    "user": {
        "account_name": "ckaestne",
        "account_id": "a/54351"
    },
    "size": "5.7",
    "camera": {
        "manufacturer": "Google",
```

```
db.getCollection('photos').find( { "user.account_name": "ckaes
```

Log files, unstructured data

```
02:49:12 127.0.0.1 GET /img13.jpg 200
02:49:35 127.0.0.1 GET /img27.jpg 200
03:52:36 127.0.0.1 GET /main.css 200
04:17:03 127.0.0.1 GET /img13.jpg 200
05:04:54 127.0.0.1 GET /img34.jpg 200
05:38:07 127.0.0.1 GET /img27.jpg 200
05:44:24 127.0.0.1 GET /img13.jpg 200
06:08:19 127.0.0.1 GET /img13.jpg 200
```

Tradeoffs

Data Encoding

Plain text (csv, logs)

Semi-structured, schema-free (JSON, XML)

Schema-based encoding (relational, Avro, ...)

Compact encodings (protobuffer, ...)

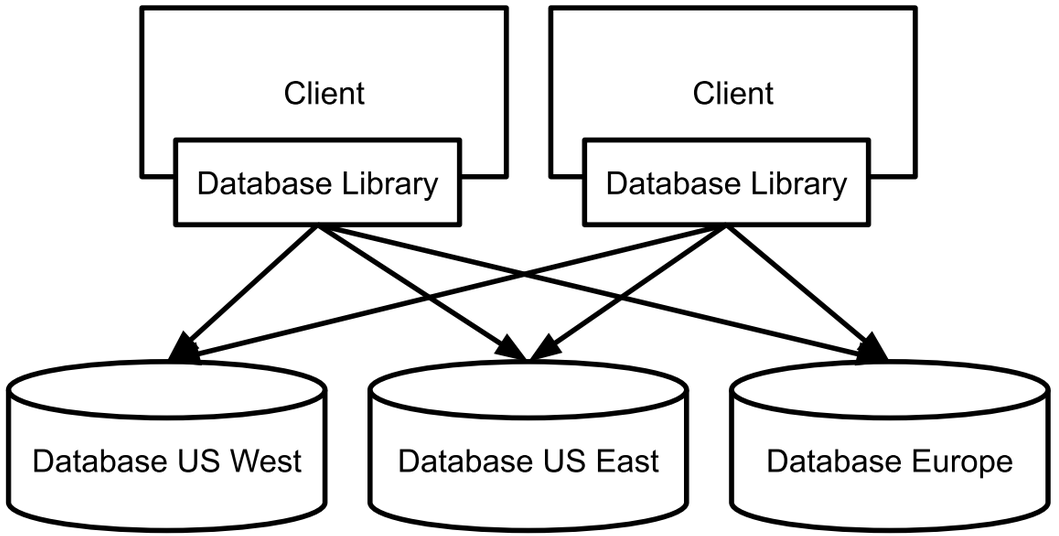
Distributed Data Storage

Replication vs Partitioning

Partitioning

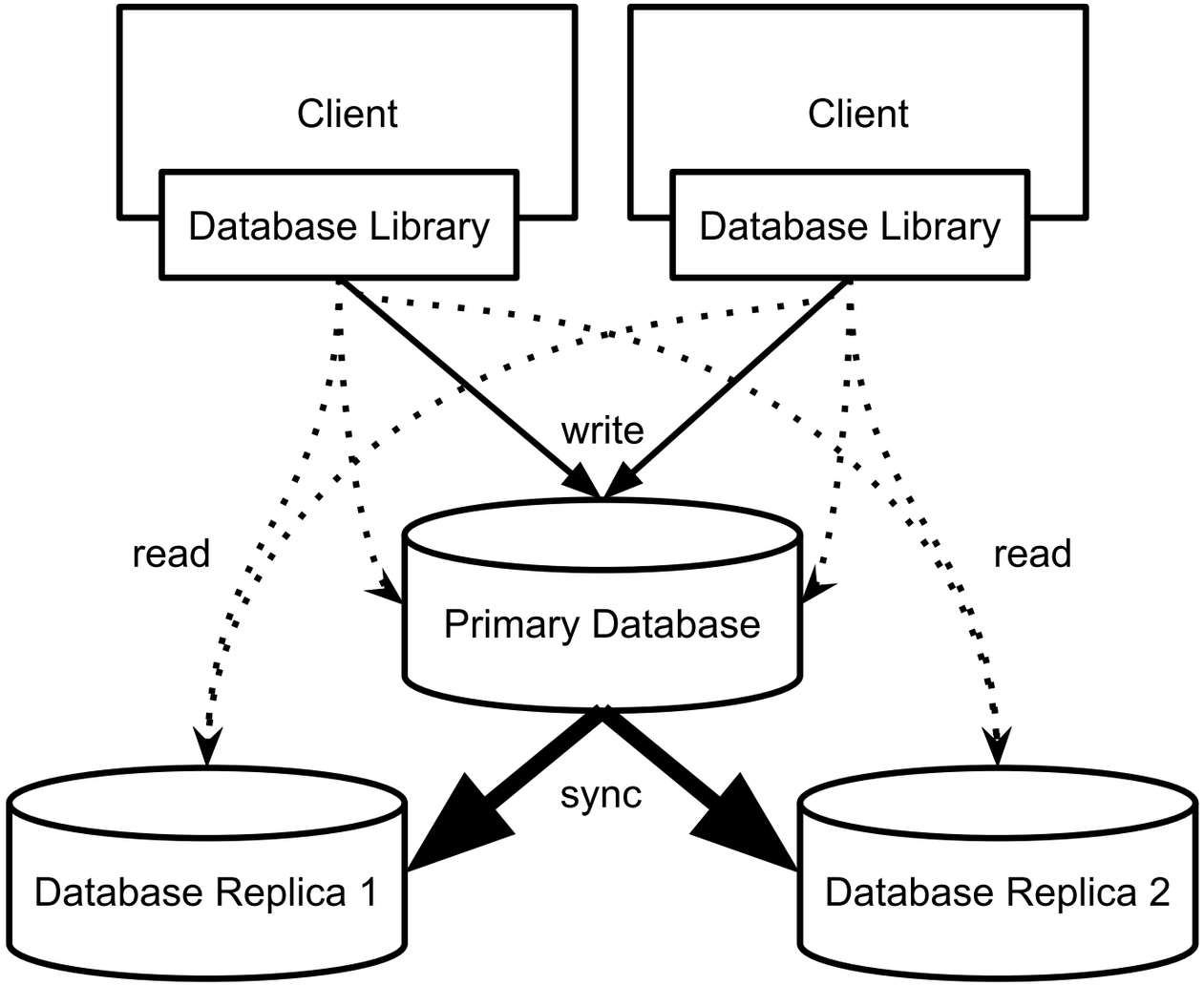
Divide data:

- *Horizontal partitioning*: Different rows in different tables; e.g., movies by decade, hashing often used
- *Vertical partitioning*: Different columns in different tables; e.g., movie title vs. all actors



Tradeoffs?

Replication with Leaders and Followers



Replication Strategies: Leaders and Followers

Write to leader, propagated synchronously or async.

Read from any follower

Elect new leader on leader outage; catchup on follower outage

Built in model of many databases (MySQL, MongoDB, ...)

Benefits and Drawbacks?

Recall: Google File System

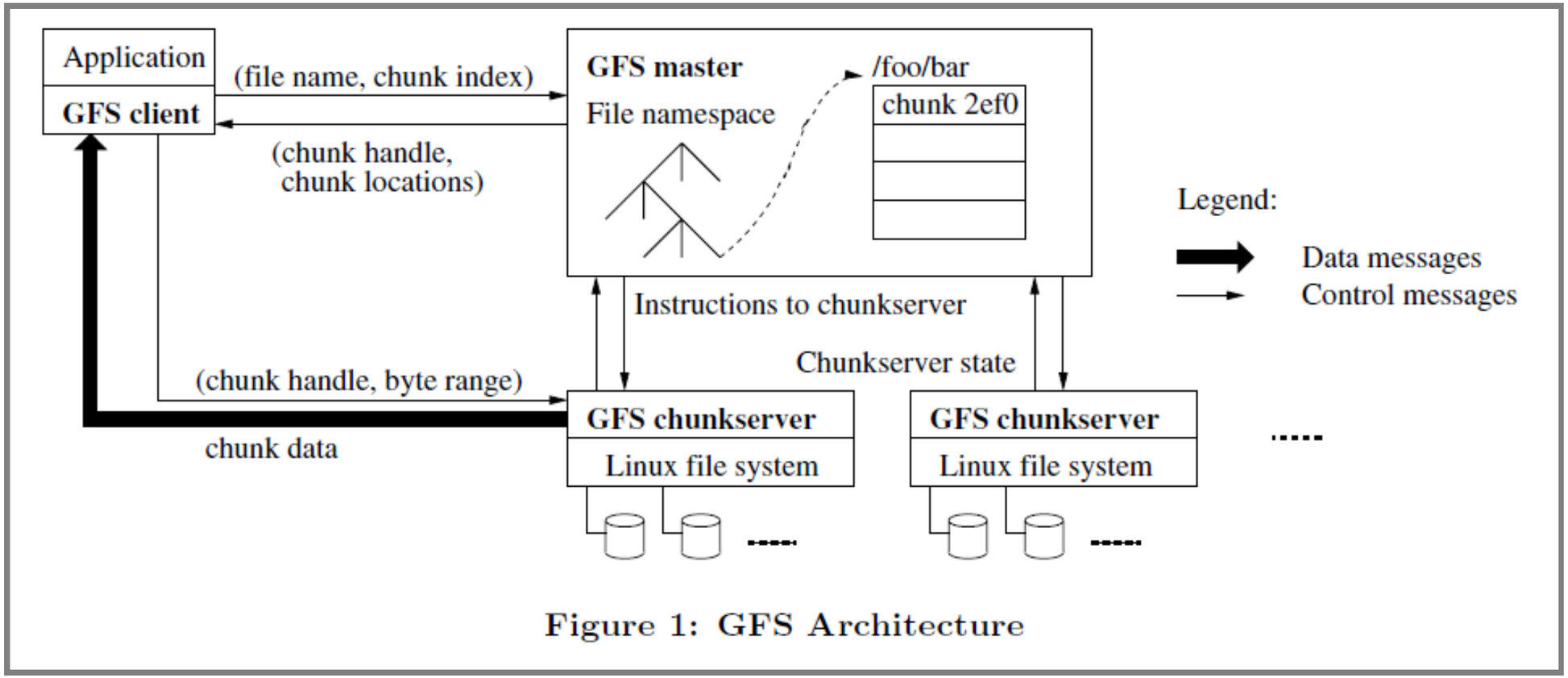


Figure 1: GFS Architecture

Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." ACM SIGOPS operating systems review. Vol. 37. No. 5. ACM, 2003.

Multi-Leader Replication

Scale write access, add redundancy

Requires coordination among leaders

- Resolution of write conflicts

Offline leaders (e.g. apps), collaborative editing

Leaderless Replication

Client writes to multiple replica, propagate from there

Read from multiple replica (quorum required)

- Repair on reads, background repair process

Versioning of entries (clock problem)

e.g. Amazon Dynamo, Cassandra, Voldemort

Transactions

Multiple operations conducted as one, all or nothing

Avoids problems such as

- dirty reads
- dirty writes

Various strategies, including locking and optimistic+rollback

Overhead in distributed setting

Data Processing (Overview)

- Services (online)
  - Responding to client requests as they come in
  - Evaluate: Response time
- Batch processing (offline)
  - Computations run on large amounts of data
  - Takes minutes to days; typically scheduled periodically
  - Evaluate: Throughput
- Stream processing (near real time)
  - Processes input events, not responding to requests
  - Shortly after events are issued

Microservices

Microservices

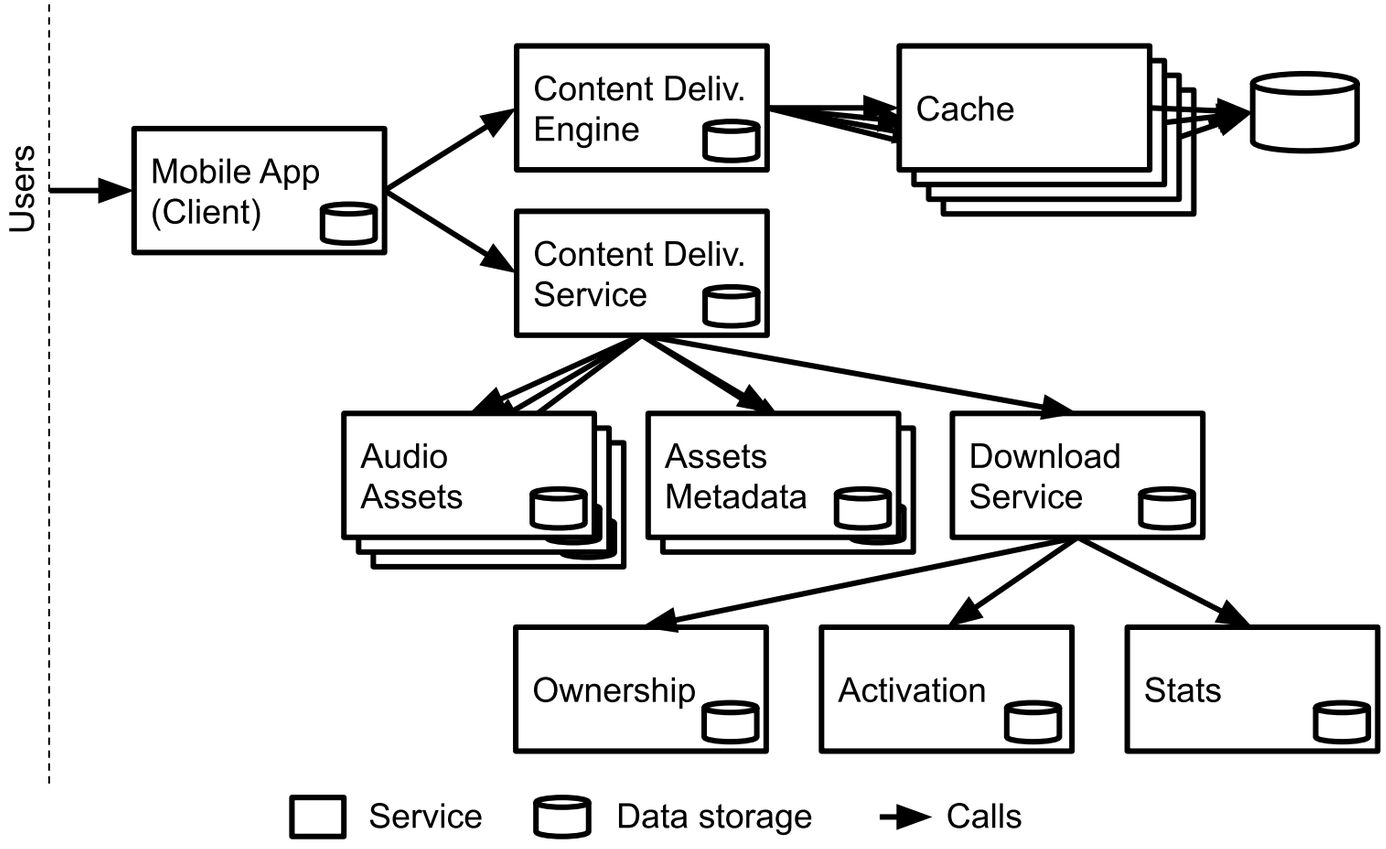


Figure based on Christopher Meiklejohn. Dynamic Reduction: Optimizing Service-level Fault Injection Testing With Service Encapsulation. Blog Post 2021

Microservices

Independent, cohesive services

- Each specialized for one task
- Each with own data storage
- Each independently scalable through multiple instances + load balancer

Remote procedure calls

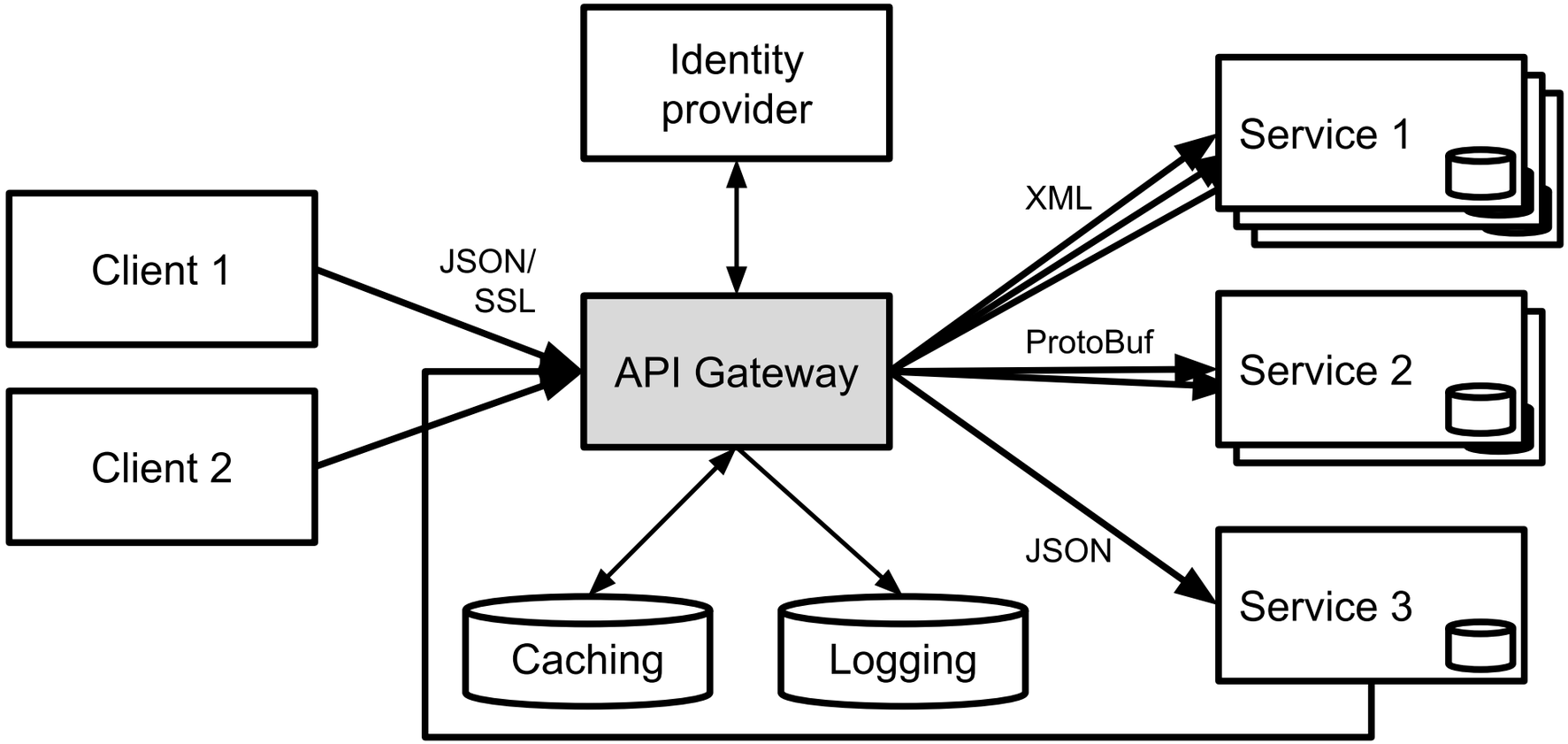
Different teams can work on different services independently (even in different languages)

But: Substantial complexity from distributed system nature: various network failures, latency from remote calls, ...

Avoid microservice complexity unless really needed for scalability

API Gateway Pattern

Central entry point, authentication, routing, updates, ...

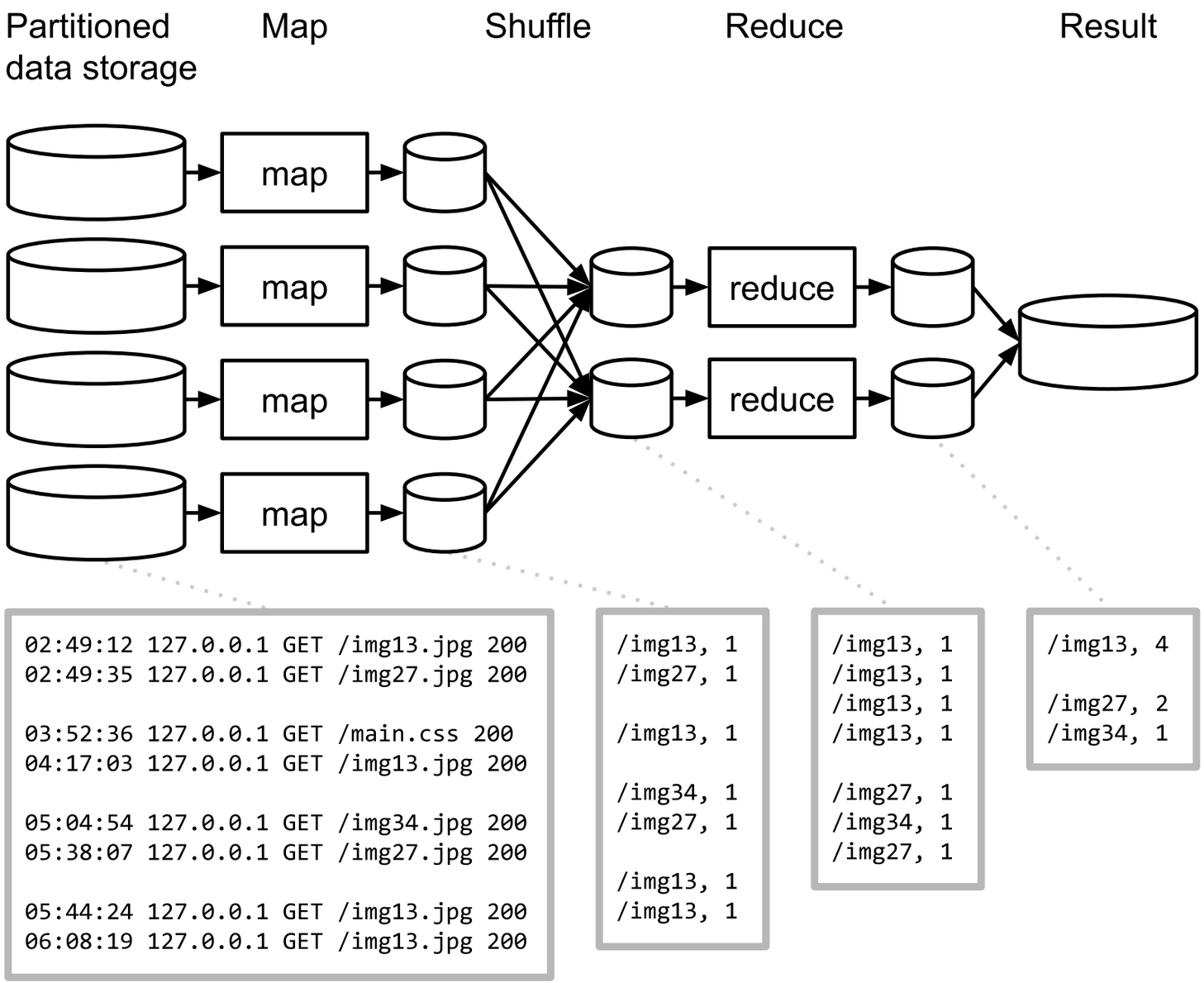


Batch Processing

Large Jobs

- Analyzing TB of data, typically distributed storage
- Filtering, sorting, aggregating
- Producing reports, models, ...

```
cat /var/log/nginx/access.log |
    awk '{print $7}' |
    sort |
    uniq -c |
    sort -r -n |
    head -n 5
```


Partitioned data storage
Map
Shuffle
Reduce
Result
map
map
map
map
reduce
reduce
02:49:12 127.0.0.1 GET /img13.jpg 200
02:49:35 127.0.0.1 GET /img27.jpg 200
03:52:36 127.0.0.1 GET /main.css 200
04:17:03 127.0.0.1 GET /img13.jpg 200
05:04:54 127.0.0.1 GET /img34.jpg 200
05:38:07 127.0.0.1 GET /img27.jpg 200
05:44:24 127.0.0.1 GET /img13.jpg 200
06:08:19 127.0.0.1 GET /img13.jpg 200
/img13, 1
/img27, 1
/img13, 1
/img34, 1
/img27, 1
/img13, 1
/img13, 1
/img13, 1
/img13, 1
/img13, 1
/img13, 1
/img27, 1
/img34, 1
/img27, 1
/img13, 4
/img27, 2
/img34, 1

Distributed Batch Processing

Process data locally at storage

Aggregate results as needed

Separate pluming from job logic

MapReduce as common framework

MapReduce -- Functional Programming Style

Similar to shell commands: Immutable inputs, new outputs, avoid side effects

Jobs can be repeated (e.g., on crashes)

Easy rollback

Multiple jobs in parallel (e.g., experimentation)

Machine Learning and MapReduce

Speaker notes

Useful for big learning jobs, but also for feature extraction

Dataflow Engines (Spark, Tez, Flink, ...)

Single job, rather than subjobs

More flexible than just map and reduce

Multiple stages with explicit dataflow between them

Often in-memory data

Pluming and distribution logic separated

Key Design Principle: Data Locality

> *Moving Computation is Cheaper than Moving Data -- Hadoop Documentation*

Data often large and distributed, code small

Avoid transfering large amounts of data

Perform computation where data is stored (distributed)

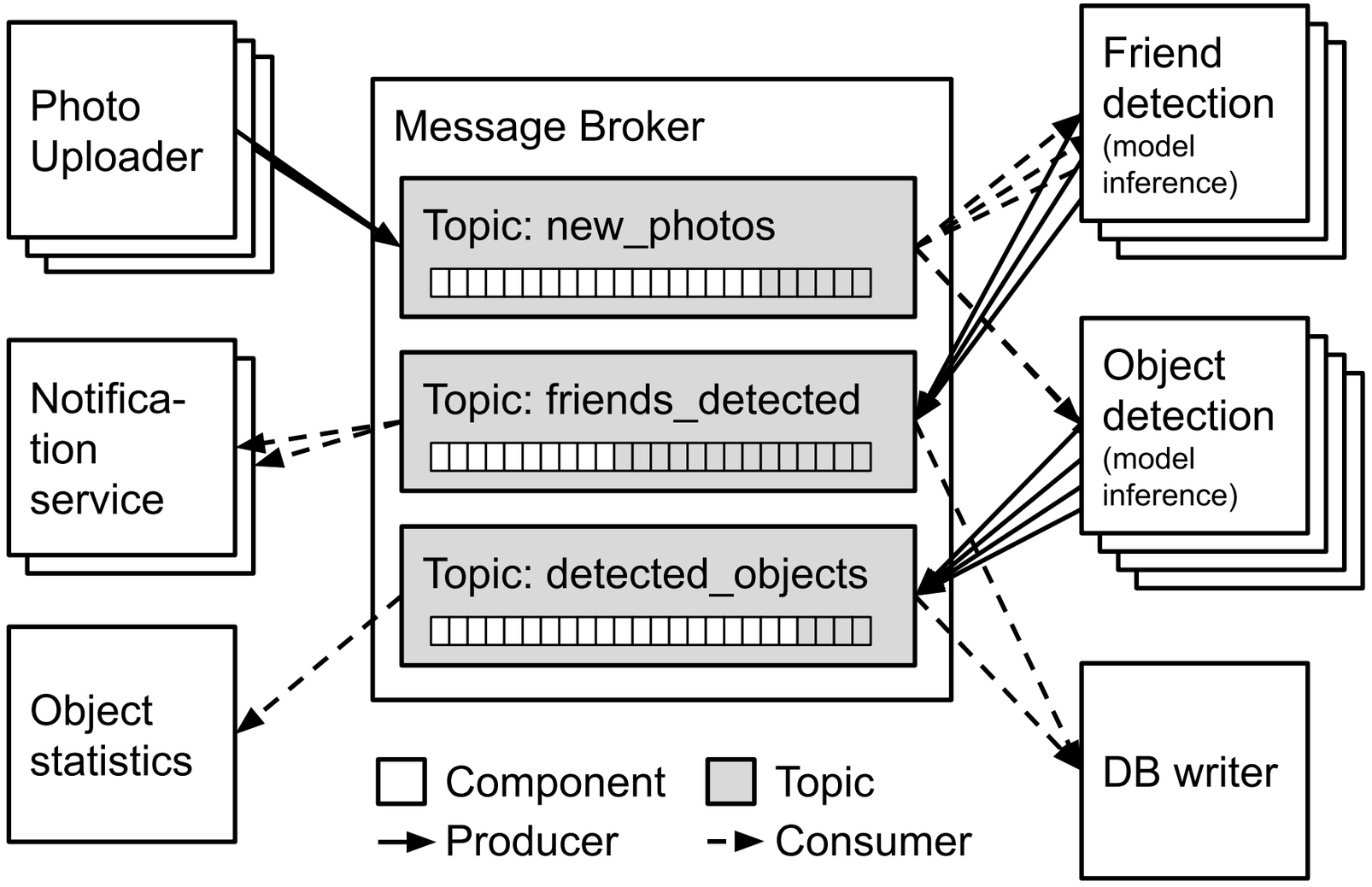
Transfer only results as needed

"The map reduce way"

Stream Processing

Event-based systems, message passing style, publish subscribe

Stream Processing (e.g., Kafka)



Messaging Systems

Multiple producers send messages to topic

Multiple consumers can read messages

-> Decoupling of producers and consumers

Message buffering if producers faster than consumers

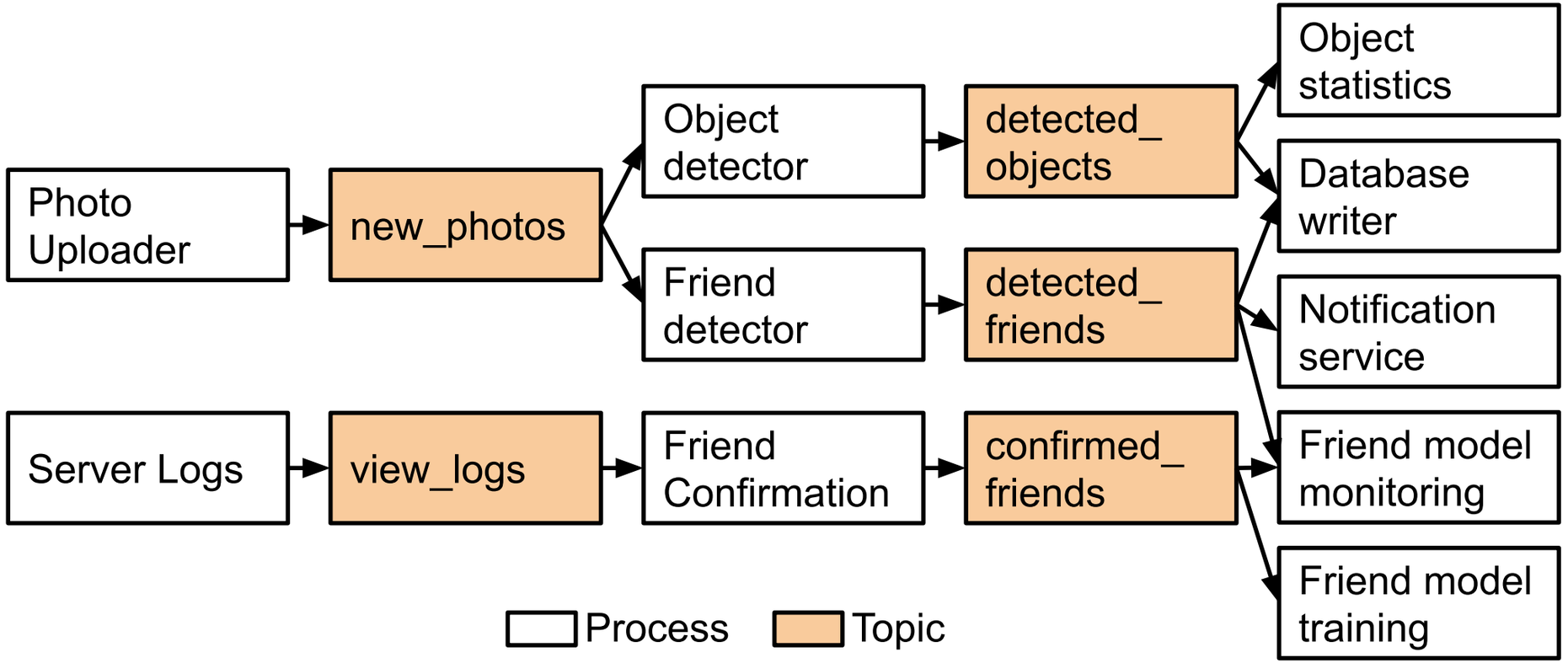
Typically some persistency to recover from failures

Messages removed after consumption or after timeout

Various error handling strategies (acknowledgements, redelivery, ...)

Common Designs

Like shell programs: Read from stream, produce output in other stream. -> loose coupling



Stream Queries

Processing one event at a time independently

vs incremental analysis over all messages up to that point

vs floating window analysis across recent messages

Works well with probabilistic analyses

Consumers

Multiple consumers share topic for scaling and load balancing

Multiple consumers read same message for different work

Partitioning possible

Design Questions

Message loss important? (at-least-once processing)

Can messages be processed repeatedly (at-most-once processing)

Is the message order important?

Are messages still needed after they are consumed?

Stream Processing and AI-enabled Systems?

Speaker notes

Process data as it arrives, prepare data for learning tasks, use models to annotate data, analytics

Event Sourcing

- Append only databases
- Record edit events, never mutate data
- Compute current state from all past events, can reconstruct old state
- For efficiency, take state snapshots
- *Similar to traditional database logs, but persistent*

```
addPhoto(id=133422131, user=54351, path="/st/u211/1U6uFl47Fy.j
updatePhotoData(id=133422131, user=54351, title="Sunset")
replacePhoto(id=133422131, user=54351, path="/st/x594/vipxBMFl
deletePhoto(id=133422131, user=54351)
```

Benefits of Immutability (Event Sourcing)

- All history is stored, recoverable
- Versioning easy by storing id of latest record
- Can compute multiple views
- Compare *git*

> *On a shopping website, a customer may add an item to their cart and then remove it again. Although the second event cancels out the first event [...], it may be useful to know for analytics purposes that the customer was considering a particular item but then decided against it. Perhaps they will choose to buy it in the future, or perhaps they found a substitute. This information is recorded in an event log, but would be lost in a database [...].*

Drawbacks of Immutable Data

Speaker notes

- Storage overhead, extra complexity of deriving state
- Frequent changes may create massive data overhead
- Some sensitive data may need to be deleted (e.g., privacy, security)

The Lambda Architecture

3 Layer Storage Architecture

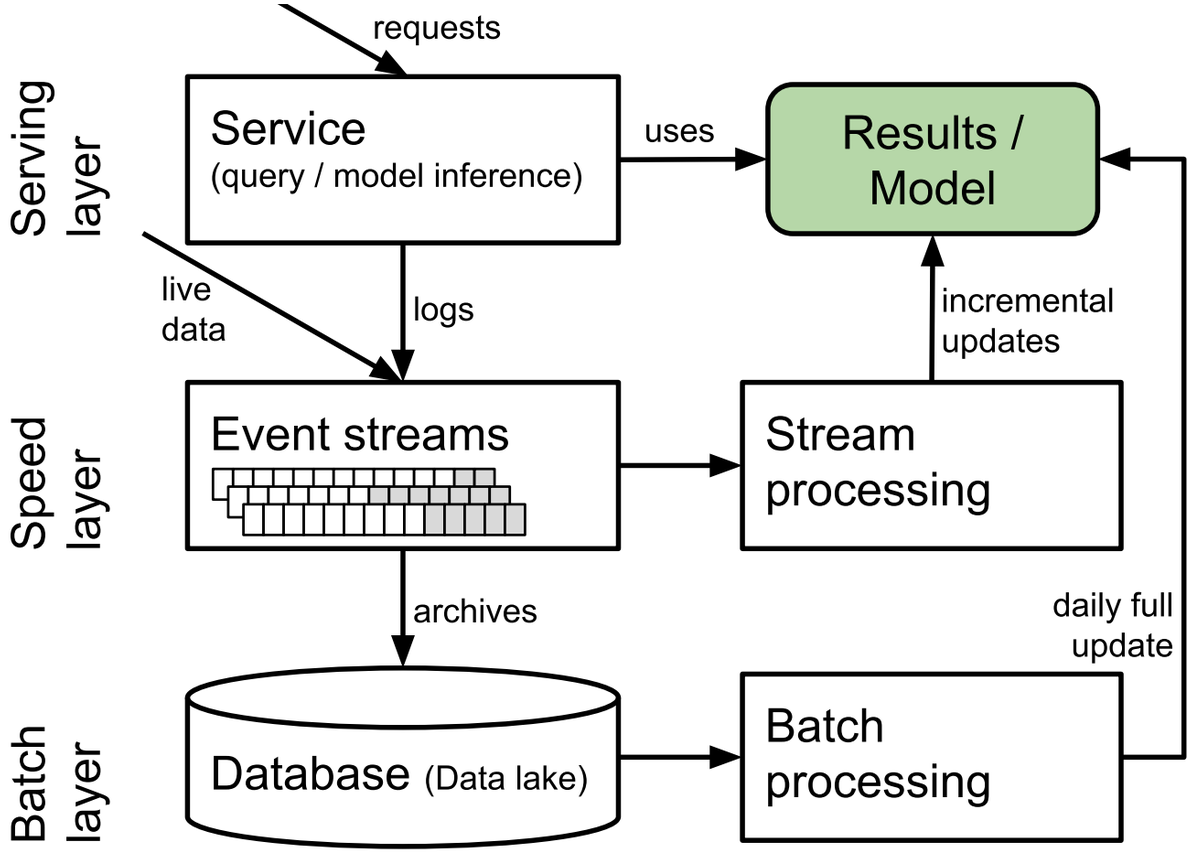
- Batch layer: best accuracy, all data, recompute periodically
- Speed layer: stream processing, incremental updates, possibly approximated
- Serving layer: provide results of batch and speed layers to clients

Assumes append-only data

Supports tasks with widely varying latency

Balance latency, throughput and fault tolerance

Lambda Architecture and Machine Learning



Data Lake

Trend to store all events in raw form (no consistent schema)

May be useful later

Data storage is comparably cheap

Data Lake

Trend to store all events in raw form (no consistent schema)

May be useful later

Data storage is comparably cheap

Bet: *Yet unknown future value of data is greater than storage costs*

Reasoning about Dataflows

Many data sources, many outputs, many copies

Which data is derived from what other data and how?

Is it reproducible? Are old versions archived?

How do you get the right data to the right place in the right format?

Plan and document data flows

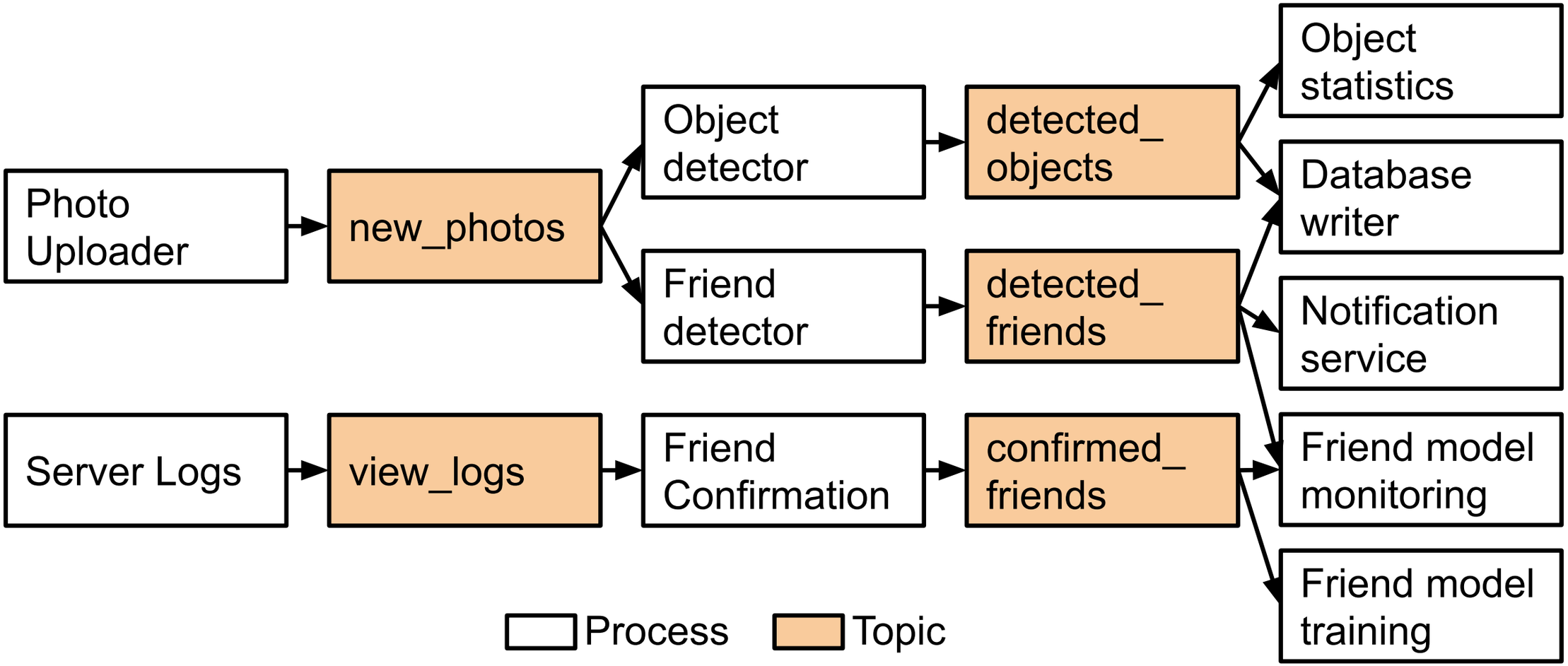
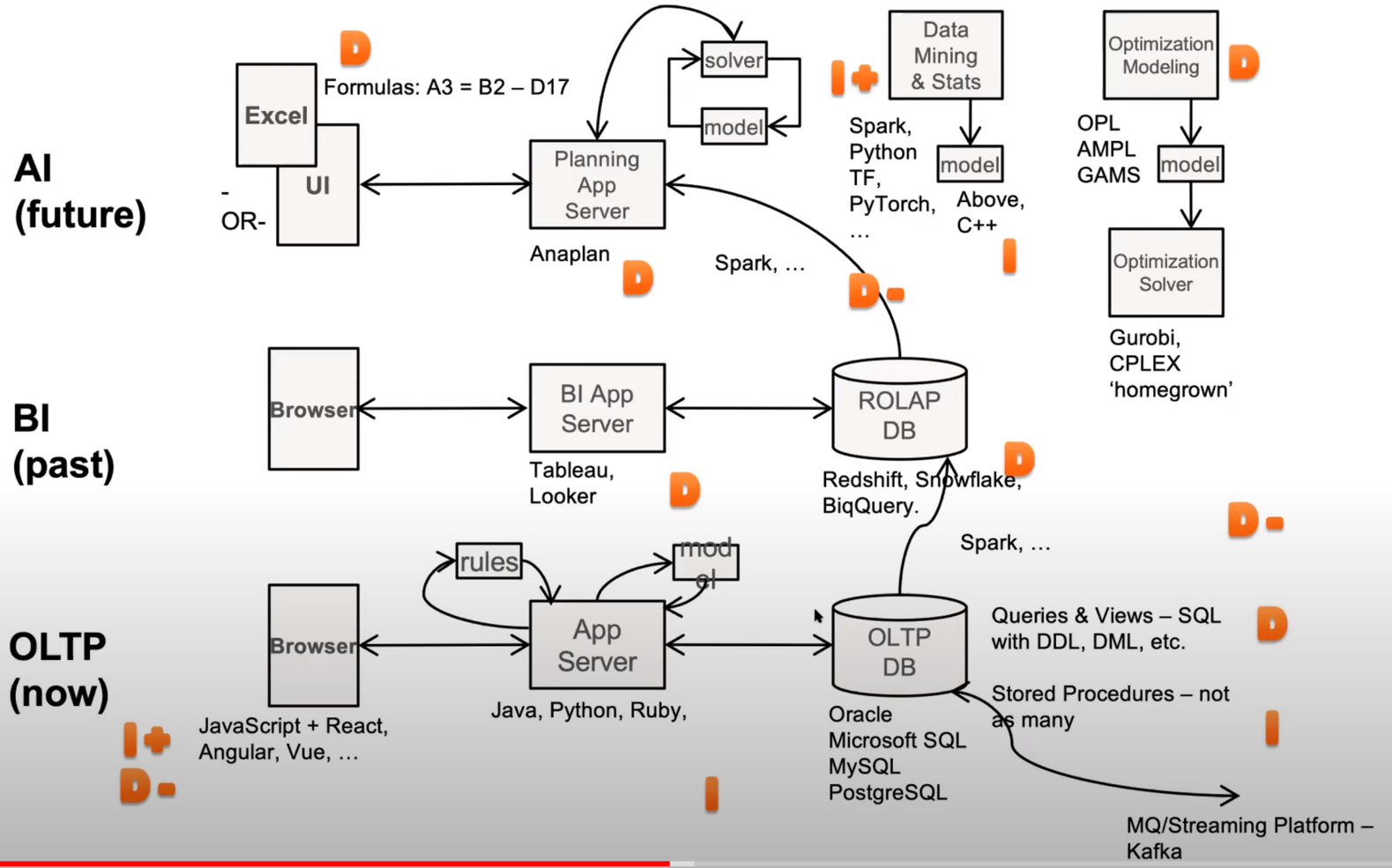


Photo Uploader
new_photos
Object detector
Friend detector
detected_ objects
detected_ friends
Object statistics
Database writer
Notification service
Server Logs
view_logs
Friend Confirmation
confirmed_ friends
Friend model monitoring
Friend model training
Process
Topic

SE4AI: Invited Talk Molham Aref "Business Systems with Machine Learning"

Enterprise Tech Stack – Now isn't much different



24:12 / 47:52

Scroll for details

Breakout: Vimeo Videos

As a group, discuss and post in `#lecture`, tagging group members:

- How to distribute storage:
- How to design scalable copy-right protection solution:
- How to design scalable analytics (views, ratings, ...):

Excursion: ETL Tools

Extract, tranform, load

The data engineer's toolbox

Data Warehousing (OLAP)

Large denormalized databases with materialized views for large scale reporting queries

- e.g. sales database, queries for sales trends by region

Read-only except for batch updates: Data from OLTP systems loaded periodically, e.g. over night

Speaker notes

Image source: https://commons.wikimedia.org/wiki/File:Data_Warehouse_Feeding_Data_Mart.jpg

ETL: Extract, Transform, Load

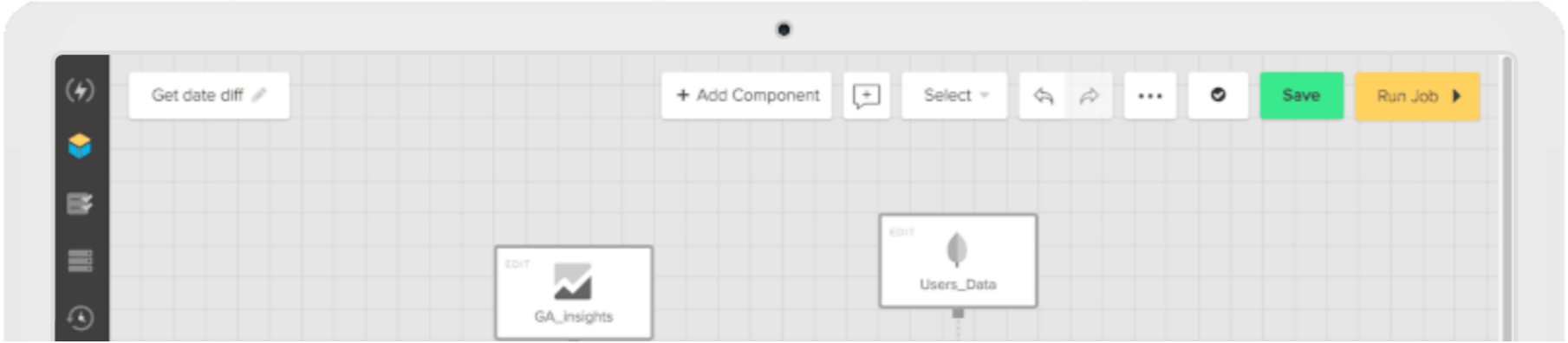
- Transfer data between data sources, often OLTP -> OLAP system
- Many tools and pipelines
  - Extract data from multiple sources (logs, JSON, databases), snapshotting
  - Transform: cleaning, (de)normalization, transcoding, sorting, joining
  - Loading in batches into database, staging
- Automation, parallelization, reporting, data quality checking, monitoring, profiling, recovery
- Many commercial tools

Xplenty

The leading data integration platform to bring all your data sources together.

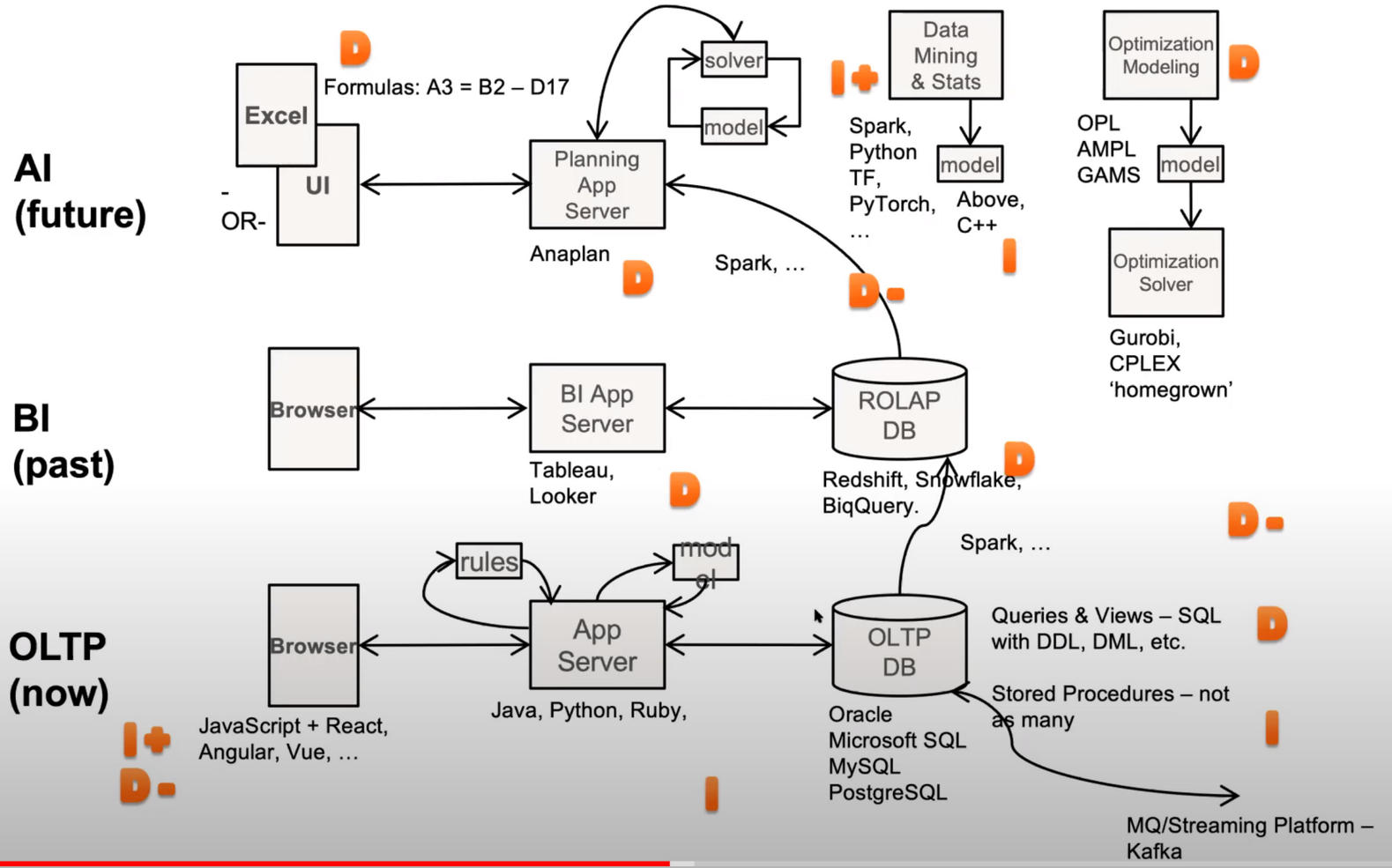
Create simple, visualized data pipelines to your data warehouse or data lake.

GET STARTED



SE4AI: Invited Talk Molham Aref "Business Systems with Machine Learning"

Enterprise Tech Stack – Now isn't much different



24:12 / 47:52

Scroll for details

Complexity of Distributed Systems


A problem has been detected and windows has been shut down to prevent damage
to your computer.
DRIVER_IRQL_NOT_LESS_OR_EQUAL
If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:
Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any windows updates you might need.
If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.
Technical information:
*** STOP: 0x000000D1 (0x0000000C,0x00000002,0x00000000,0xF86B5A89)
*** gv3.sys - Address F86B5A89 base at F86B5000, DateStamp 3dd991eb
Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further
assistance.

Common Distributed System Issues

- Systems may crash
- Messages take time
- Messages may get lost
- Messages may arrive out of order
- Messages may arrive multiple times
- Messages may get manipulated along the way
- Bandwidth limits
- Coordination overhead
- Network partition
- ...

Types of failure behaviors

- Fail-stop
- Other halting failures
- Communication failures
  - Send/receive omissions
  - Network partitions
  - Message corruption
- Data corruption
- Performance failures
  - High packet loss rate
  - Low throughput, High latency
- Byzantine failures

Common Assumptions about Failures

- Behavior of others is fail-stop
- Network is reliable
- Network is semi-reliable but asynchronous
- Network is lossy but messages are not corrupt
- Network failures are transitive
- Failures are independent
- Local data is not corrupt
- Failures are reliably detectable
- Failures are unreliably detectable

Strategies to Handle Failures

- Timeouts, retry, backup services
- Detect crashed machines (ping/echo, heartbeat)
- Redundant + first/voting
- Transactions

- Do lost messages matter?
- Effect of resending message?

Test Error Handling

- Recall: Testing with stubs
- Recall: Chaos experiments

Performance Planning and Analysis

Performance Planning and Analysis

Ideally architectural planning upfront

- Identify key components and their interactions
- Estimate performance parameters
- Simulate system behavior (e.g., queuing theory)

Existing system: Analyze performance bottlenecks

- Profiling of individual components
- Performance testing (stress testing, load testing, etc)
- Performance monitoring of distributed systems

Performance Analysis

What is the average waiting?

How many customers are waiting on average?

How long is the average service time?

What are the chances of one or more servers being idle?

What is the average utilization of the servers?

-> Early analysis of different designs for bottlenecks

-> Capacity planning

Queuing Theory

Queuing theory deals with the analysis of lines where customers wait to receive a service

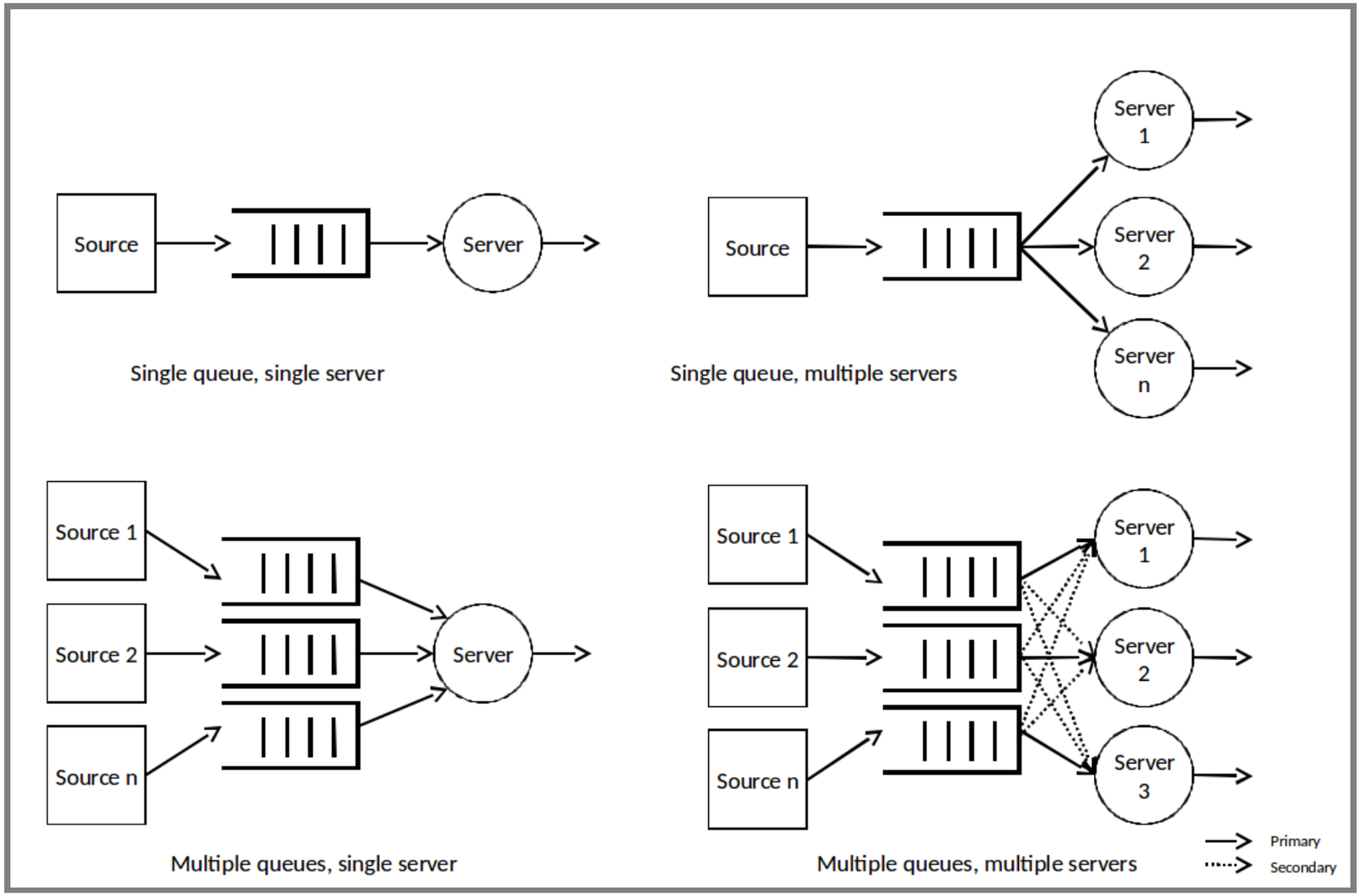
- Waiting at Quiznos
- Waiting to check-in at an airport
- Kept on hold at a call center
- Streaming video over the net
- Requesting a web service

A queue is formed when request for services outpace the ability of the server(s) to service them immediately

- Requests arrive faster than they can be processed (unstable queue)
- Requests do not arrive faster than they can be processed but their processing is delayed by some time (stable queue)

Queues exist because infinite capacity is infinitely expensive and excessive capacity is excessively expensive

Queuing Theory



Analysis Steps (roughly)

Identify system abstraction to analyze (typically architectural level, e.g. services, but also protocols, datastructures and components, parallel processes, networks)

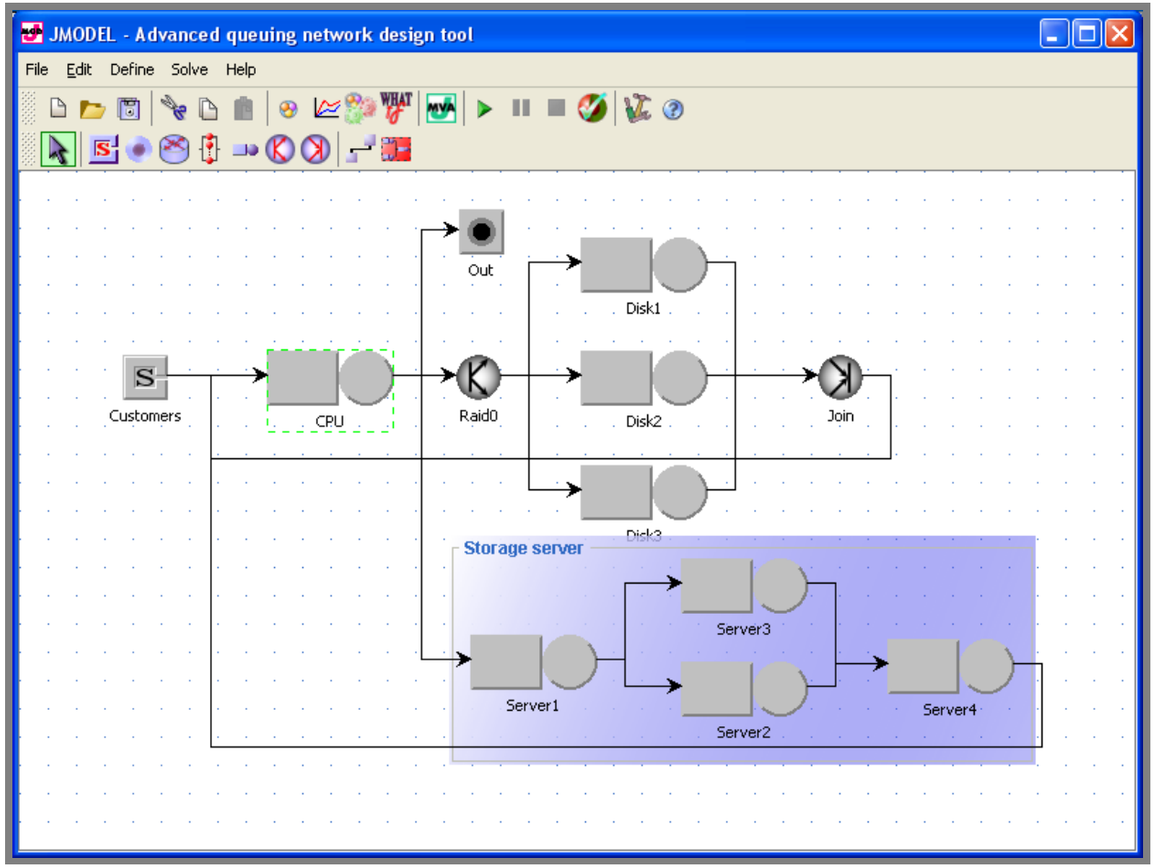
Model connections and dependencies

Estimate latency and capacity per component (measurement and testing, prior systems, estimates, …)

Run simulation/analysis to gather performance curves

Evaluate sensitivity of simulation/analysis to various parameters (‘what-if questions’)

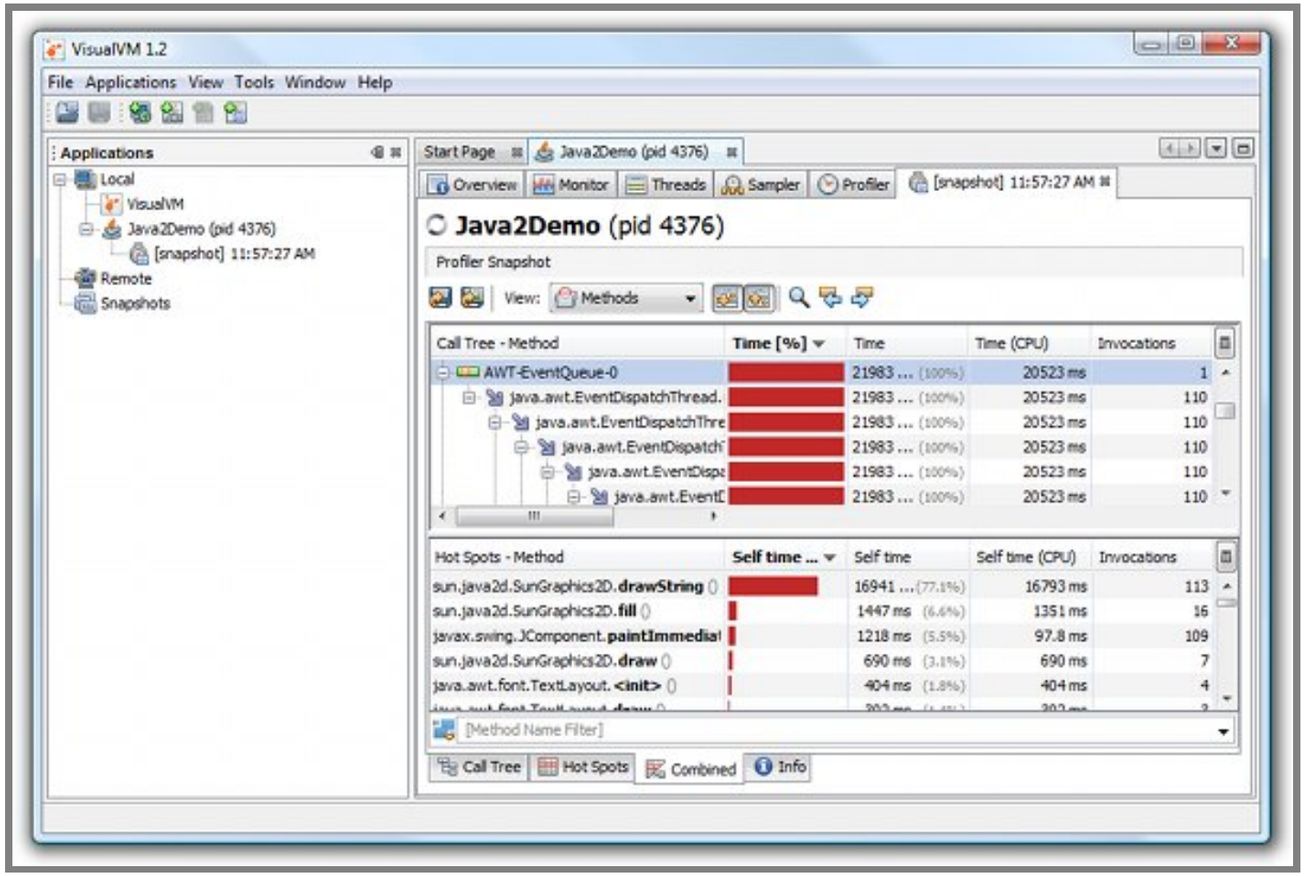
Simulation (e.g., JMT)



G.Serazzi Ed. Performance Evaluation Modelling with JMT: learning by examples. Politecnico di Milano - DEI, TR 2008.09, 366 pp., June 2008

Profiling

Mostly used during development phase in single components

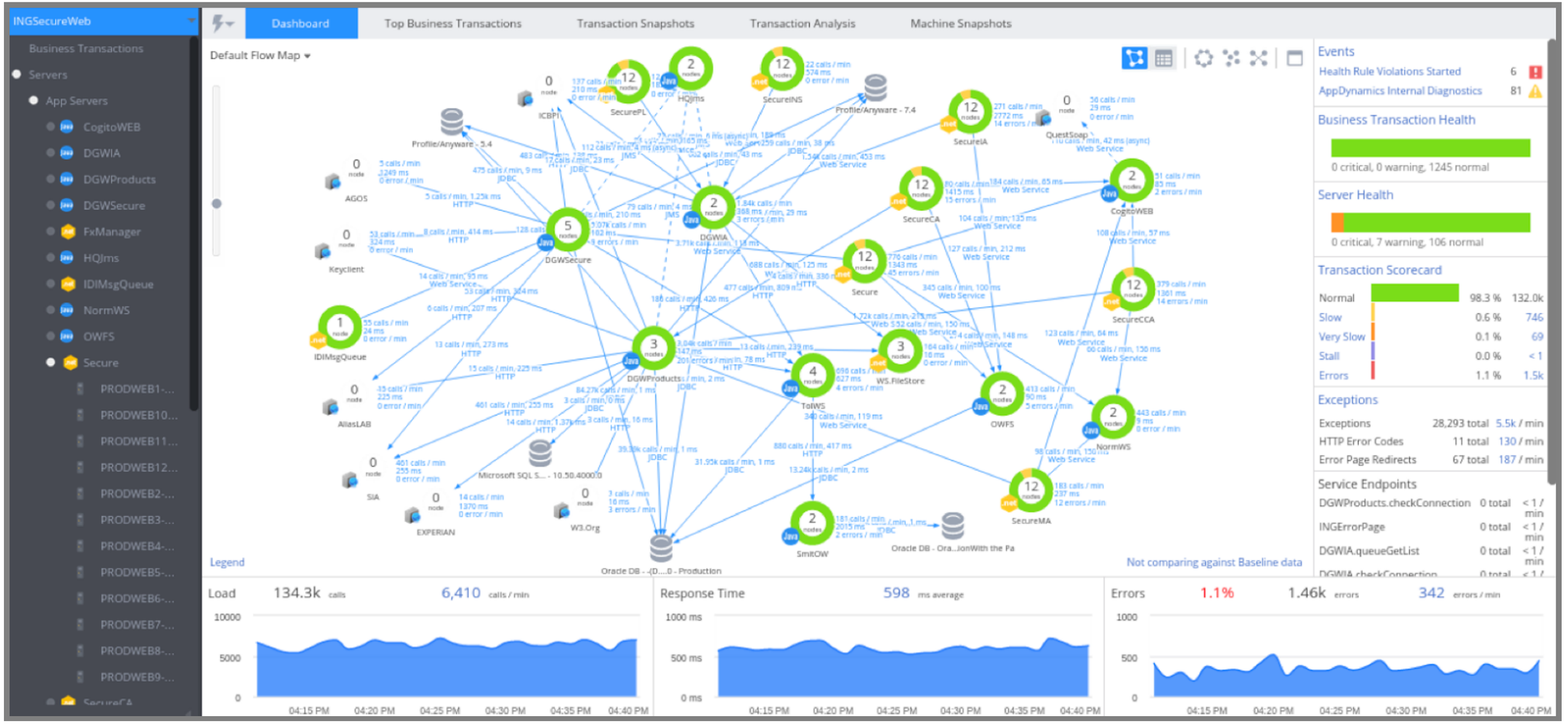


Performance Testing

- Load testing: Assure handling of maximum expected load
- Scalability testing: Test with increasing load
- Soak/spike testing: Overload application for some time, observe stability
- Stress testing: Overwhelm system resources, test graceful failure + recovery

- Observe (1) latency, (2) throughput, (3) resource use
- All automateable; tools like JMeter

Performance Monitoring of Distr. Systems



Source: https://blog.appdynamics.com/tag/fiserv/

Performance Monitoring of Distributed Systems

- Instrumentation of (Service) APIs
- Load of various servers
- Typically measures: latency, traffic, errors, saturation

- Monitoring long-term trends
- Alerting
- Automated releases/rollbacks
- Canary testing and A/B testing

Summary

- Large amounts of data (training, inference, telemetry, models)
- Distributed storage and computation for scalability
- Common design patterns (e.g., batch processing, stream processing, lambda architecture)
- Design considerations: mutable vs immutable data
- Distributed computing also in machine learning
- Lots of tooling for data extraction, transformation, processing
- Many challenges through distribution: failures, debugging, performance, ...

Further Readings

- Molham Aref "Business Systems with Machine Learning" Invited Talk 2020
- Sawadogo, Pegdwendé, and Jérôme Darmont. "On data lake architectures and metadata management." Journal of Intelligent Information Systems 56, no. 1 (2021): 97-120.
- Warren, James, and Nathan Marz. Big Data: Principles and best practices of scalable realtime data systems. Manning, 2015.
- Smith, Jeffrey. Machine Learning Systems: Designs that Scale. Manning, 2018.
- Polyzotis, Neoklis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2017. “Data Management Challenges in Production Machine Learning.” In Proceedings of the 2017 ACM International Conference on Management of Data, 1723–26. ACM.