Distributed Systems CS6421

IoT, Big Data, and ML

Prof. Tim Wood and Prof. Roozbeh Haghnazar

Includes material adapted from Van Steen and Tanenbaum's Distributed Systems book

Final Project

Questions?

- Implementation phase!
  - Get coding!
  - Keep your code in GitHub

- See website for details on:
  - Report
  - ~15 minutes Video

- Timeline
  - Milestone 0: Form a Team - 10/12
  - Milestone 1: Select a Topic - 10/19
  - Milestone 2: Literature Survey - 10/29
  - Milestone 3: Design Document - 11/8
  - **Milestone 4: Final Video/Report – 12/22**
    - **Bonus for submitting early!**

https://gwdistsys20.github.io/project/

Last Time…

- Cloud Application Design
    - Monolith vs Microservices
    - Serverless
    - PaaS, IaaS, etc

This Time…

- IoT
- Big data
- Machine learning
- Security/Privacy
- Wrap up

DistSys Challenges

- **Heterogeneity**
- **Openness**
- **Security**
- **Failure Handling**
- **Concurrency**
- **Quality of Service**
- **Scalability**
- **Transparency**

Covered Challenges

Distributed systems and Internet of things

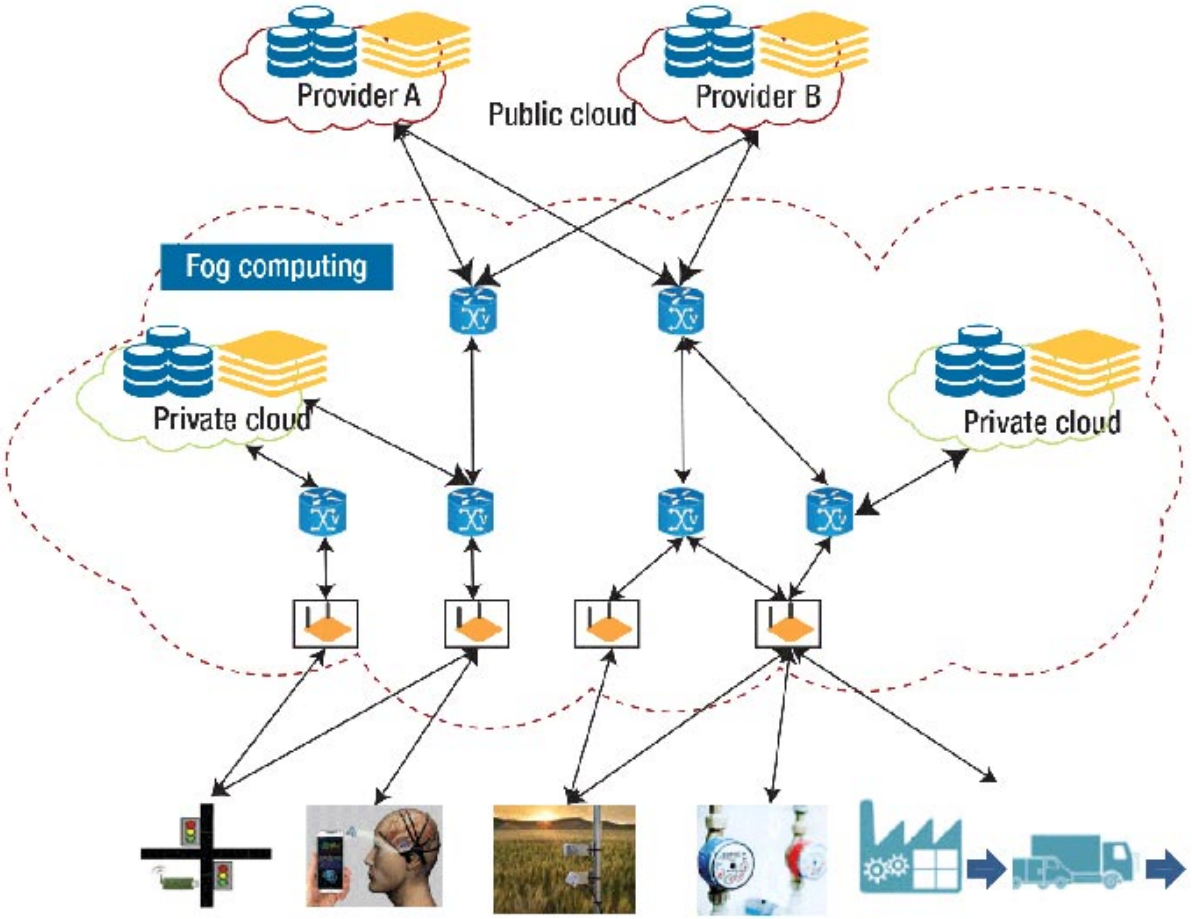
Prof. Tim Wood & Prof. Roozbeh Haghnazar

What is IoT?

IoT is a collection of connected devices and services that work together to do useful stuff.

- Sensors
- Communication
- Computation
- Service

In IoT there is billions of smart objects, such as Sensors, Actuators, Smart phones and Smart Vehicles, etc. are connected with each other to sensing physical signals and giving better service in real time human need.



CHALLENGES

- Ultra-Big data
- Infeasible and inefficient to handle with cloud services because of computing and communication recourses.
- Using centralized approached where all the data analysis works are executed on cloud servers.
- Having access from different types of users and devices
- Addressing the proper device, sensor or any components for purposes

Heterogeneity
Openness
Security
Failure Handling
Concurrency
Quality of Service
Scalability
Transparency

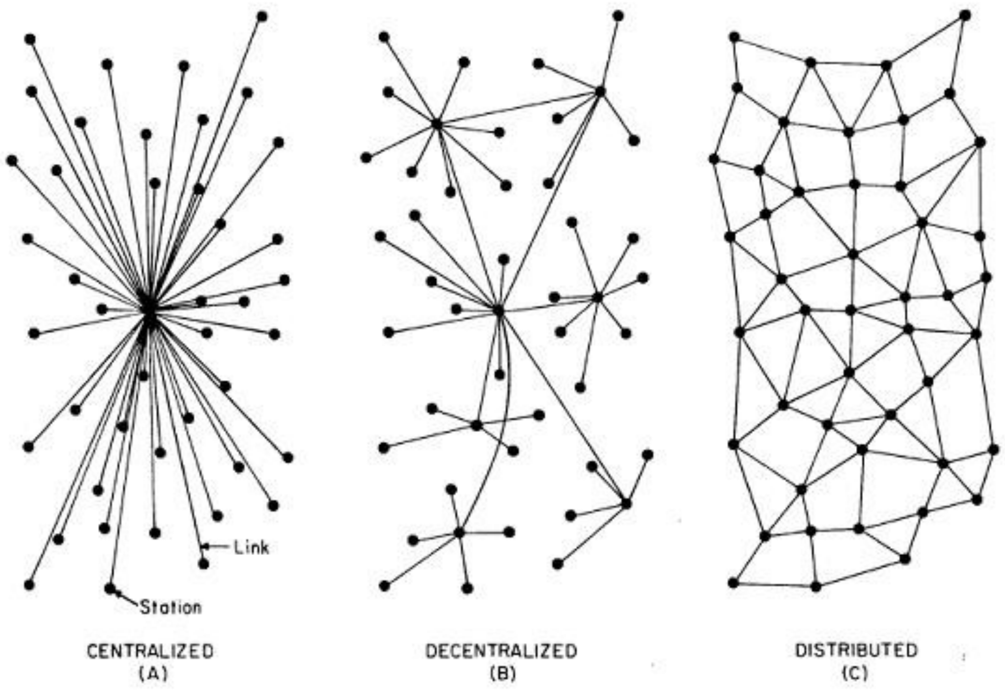
IoT Characteristics

- Intelligence
- Connectivity
- Dynamic Nature
- Enormous scale
- Sensing
- Heterogeneity
- Security
- Zero-Configuration

Solution

IoT Main Models

- There are mainly 3 types of system architectures utilized by modern software applications which in turn determine the architecture of an application:
  - Centralized
  - Decentralized
  - Distributed



DECENTRALIZED

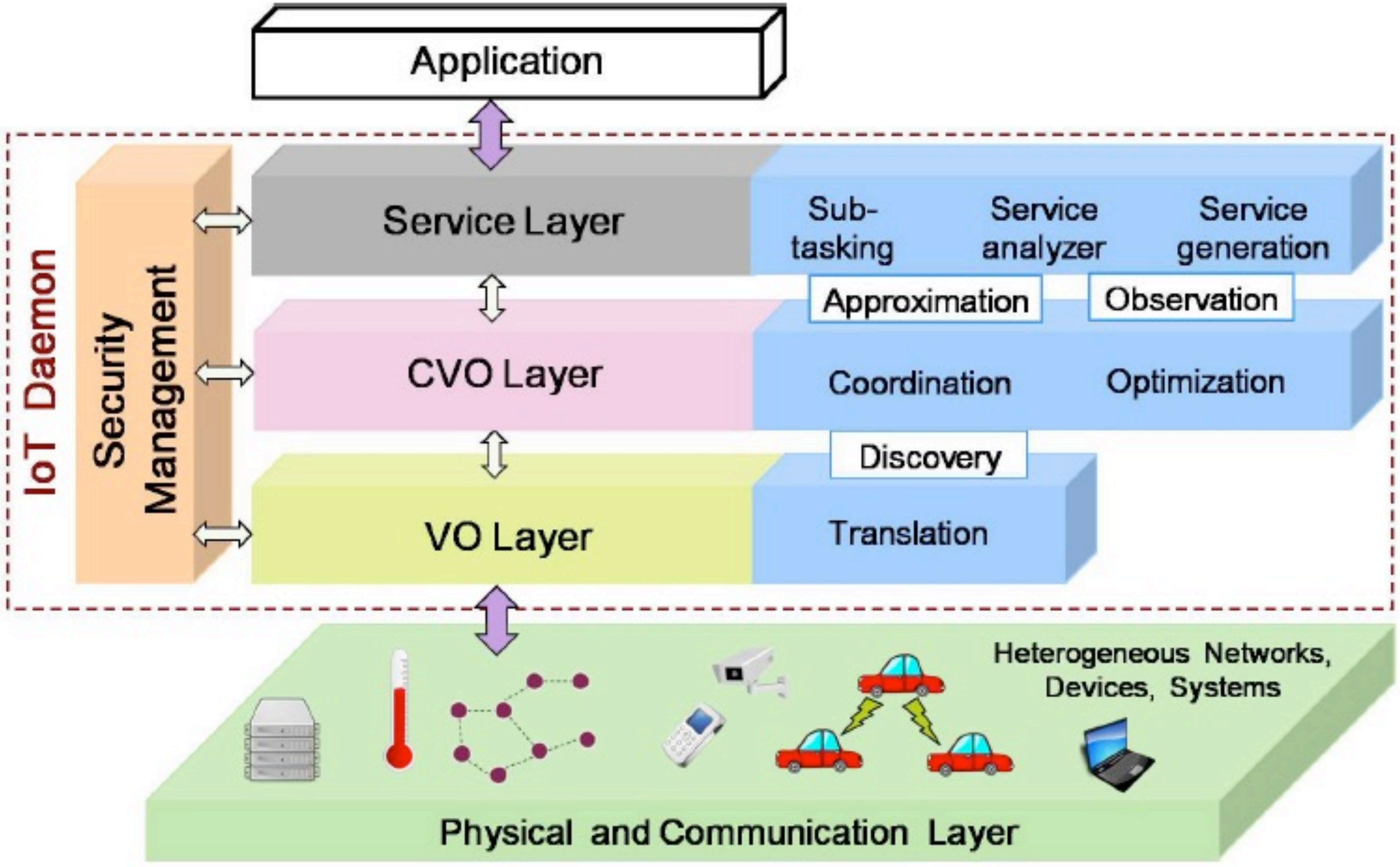
- In a decentralized, non-distributed (or co-located) system, all the parts of the system are in the same physical location.
- This eliminates the biggest problem of single point of failure with centralized network.
- Decentralized systems form the basis of parallel and cluster computing where different processors on the network can execute the same instruction set on different parts of the same dataset (shards) or different dataset.
- Features:
  - **Individual goals:** the nodes are working to achieve individual goals which may need to be coordinated to get the end result (eg map reduce)
  - **Loosely coupled:** the nodes on the network should be loosely coupled
  - **Direct synchronization with other nodes on the network:** the nodes on the network should be synchronized via a common clock
  - **Shared memory:** the nodes on the network should use a shared memory space to store intermediate state and data
  - **Geographical distribution:** the nodes on the network should be collocated, with minimum geographical distribution for data redundancy if required
  - **Homogeneity**: the nodes on the network should ideally be autonomous and heterogeneous in nature

Distributed

- In a distributed network the nodes are not collocated but distributed geographically also It avoids the centralization completely.
- A node can either be a communication endpoint or a point of communication redistribution, linking to other nodes.
- Features:
  - **Common goal:** the nodes are working to achieve a common goal that cannot be achieved using a single processor
  - **Loosely coupled:** the nodes on the network should be loosely coupled and should be able to communicate via a messaging system using a standard protocol
  - **No direct synchronization with other nodes on the network:** the nodes on the network should not be synchronized via a common clock
  - **Distributed memory:** the nodes on the network should not have a shared memory space if possible, but shared distributed memory space for the entire system can be provided as an abstraction
  - **Geographical distribution:** the nodes on the network should be geographically distributed and should avoid any centralization
  - **Autonomy and heterogeneity**: the nodes on the network should be autonomous and heterogeneous in nature

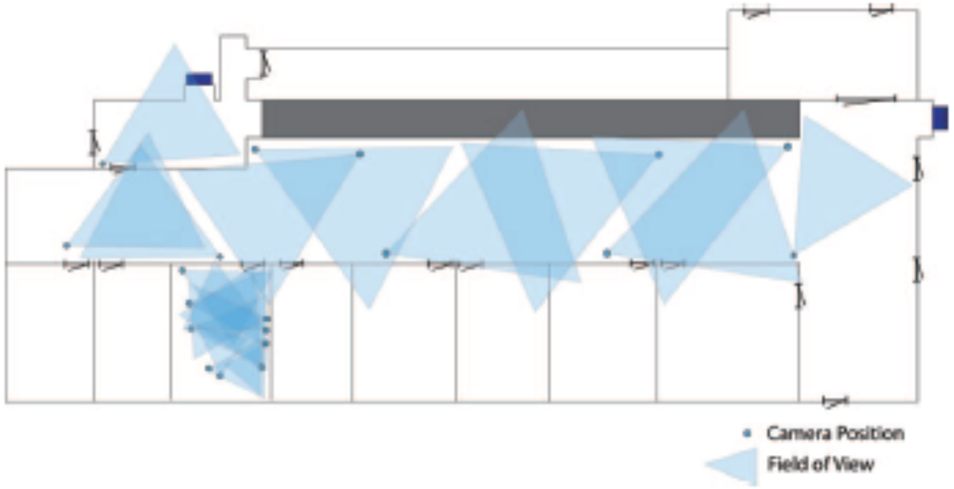
DIAT: A Scalable Distributed Architecture for IoT

- Virtual Object Layer (VOL), for object virtualization
- Composite Virtual Object Layer (CVOL), for service composition and execution
- Service Layer (SL), for service creation and management



EXAMPLE: DISTRIBUTED COMPUTING IN VIDEO SENSOR NETWORK

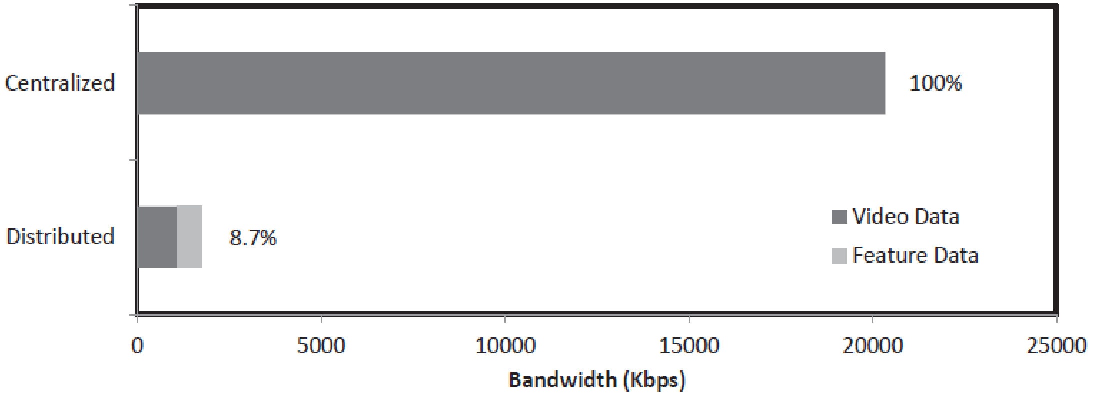
- Video Surveillance:
- In this case target is to store video sequence when critical event occurs. And the cameras have motion detection feature
- Approaches:
  - Centralized approach: All data are streamed back to the server.
  - Distributed approach: Intra processing stage and Inter Processing stage



(a)

Example: Distributed Computing In Video Sensor Network

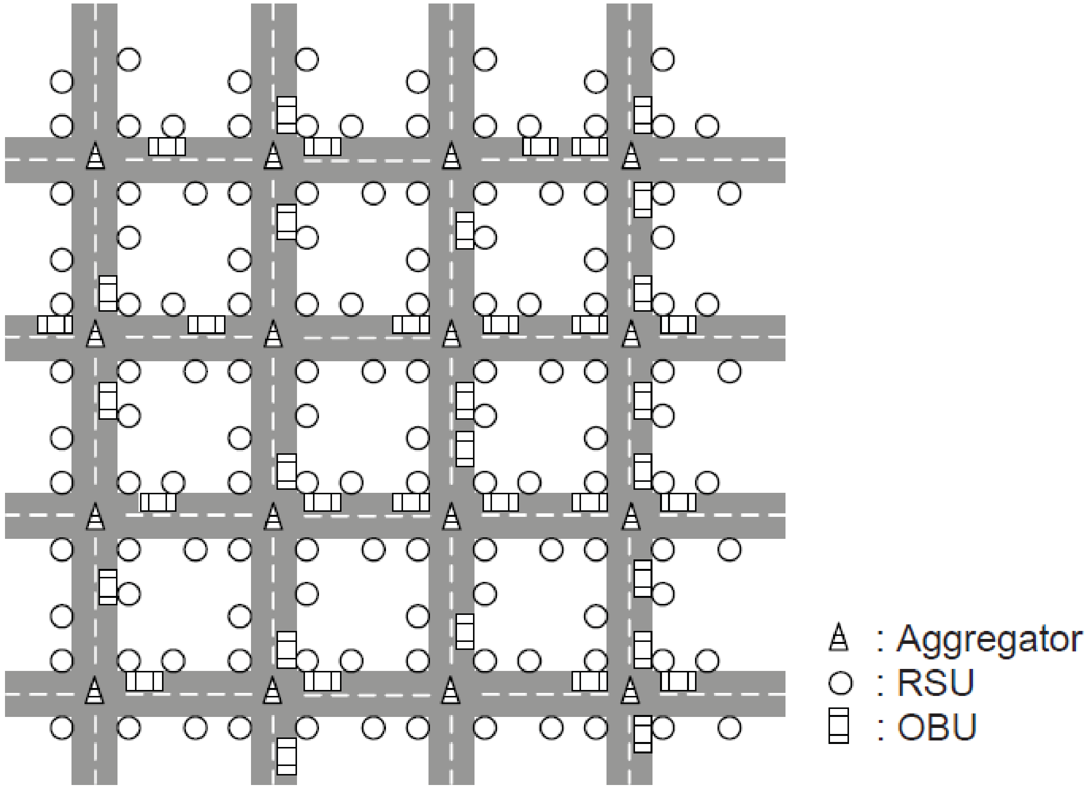
- As showing in figure, with Distributed Approach, 91.3% of the transmission bandwidth can be saved.



(c)

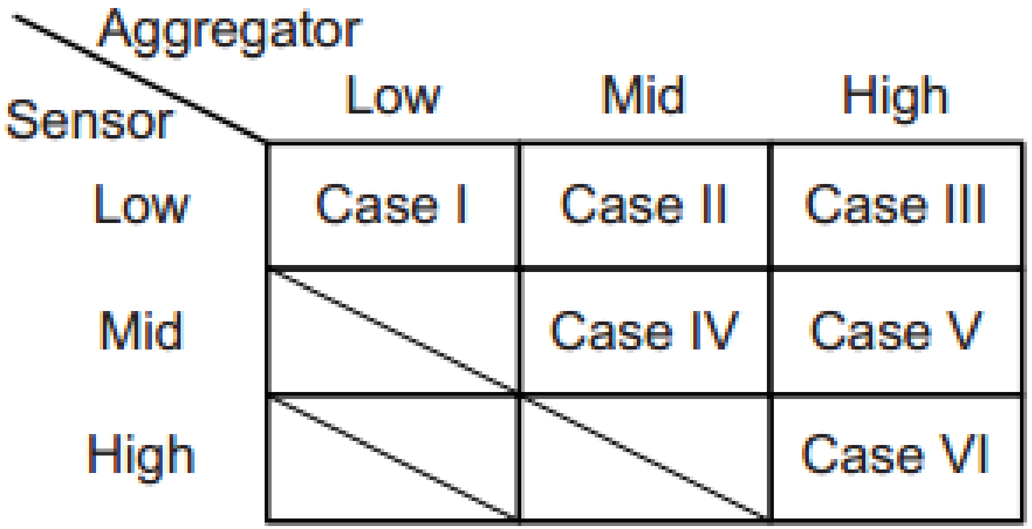
Example: Distributed Computing In Video Sensor Network

- Vehicle Localization
- Vehicle neighboring map generation system with video cameras in an intelligent transportation system.
  - Aggregator:
  - RSU (Road site unit)
  - OBU (On-Board unit)



Example: Distributed Computing In Video Sensor Network

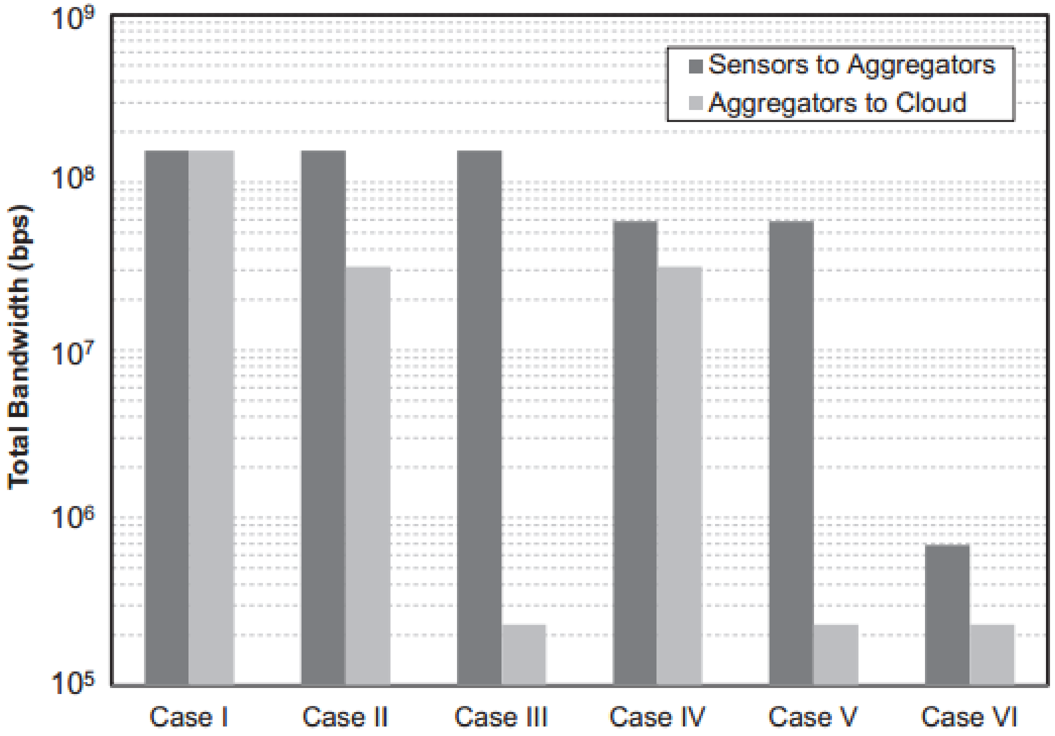
- Here six different cases taken to test case:
- **Low-end sensor** is the video sensor with simple video capturing, coding and transmission ability.
- **Middle-level sensor** is the video sensor with moving object detection ability. It can transmit video data to aggregator only when it detects moving object.
- **High-end sensor** is the video sensor with a vehicle detection subsystem. It can transmit vehicle information instead of sending video data to aggregator.



Sensor \ Aggregator	Low	Mid	High
Low	Case I	Case II	Case III
Mid		Case IV	Case V
High			Case VI

Example: Distributed Computing In Video Sensor Network

- Sensors → Corresponding Aggregators
- All aggregators → Cloud severs



BigData

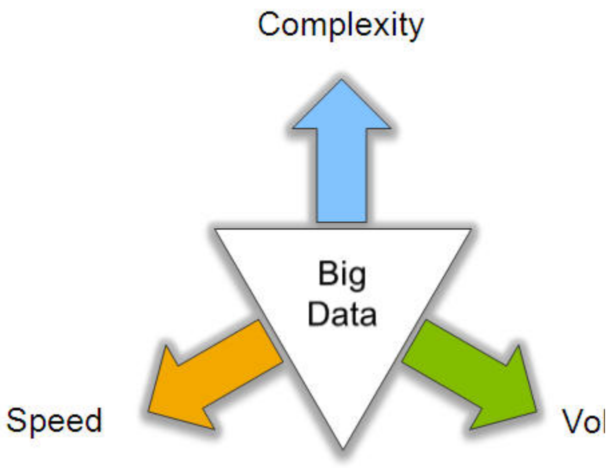
Prof. Tim Wood & Prof. Roozbeh Haghnazar

WHAT'S BIG DATA?

No single definition; here is from Wikipedia:

- **Big data** is the term for a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications.
- The challenges include capture, curation, storage, search, sharing, transfer, analysis, and visualization.
- The trend to larger data sets is due to the additional information derivable from analysis of a single large set of related data, as compared to separate smaller sets with the same total amount of data, allowing correlations to be found to "spot business trends, determine quality of research, prevent diseases, link legal citations, combat crime, and determine real-time roadway traffic conditions."

Big Data: 3V's



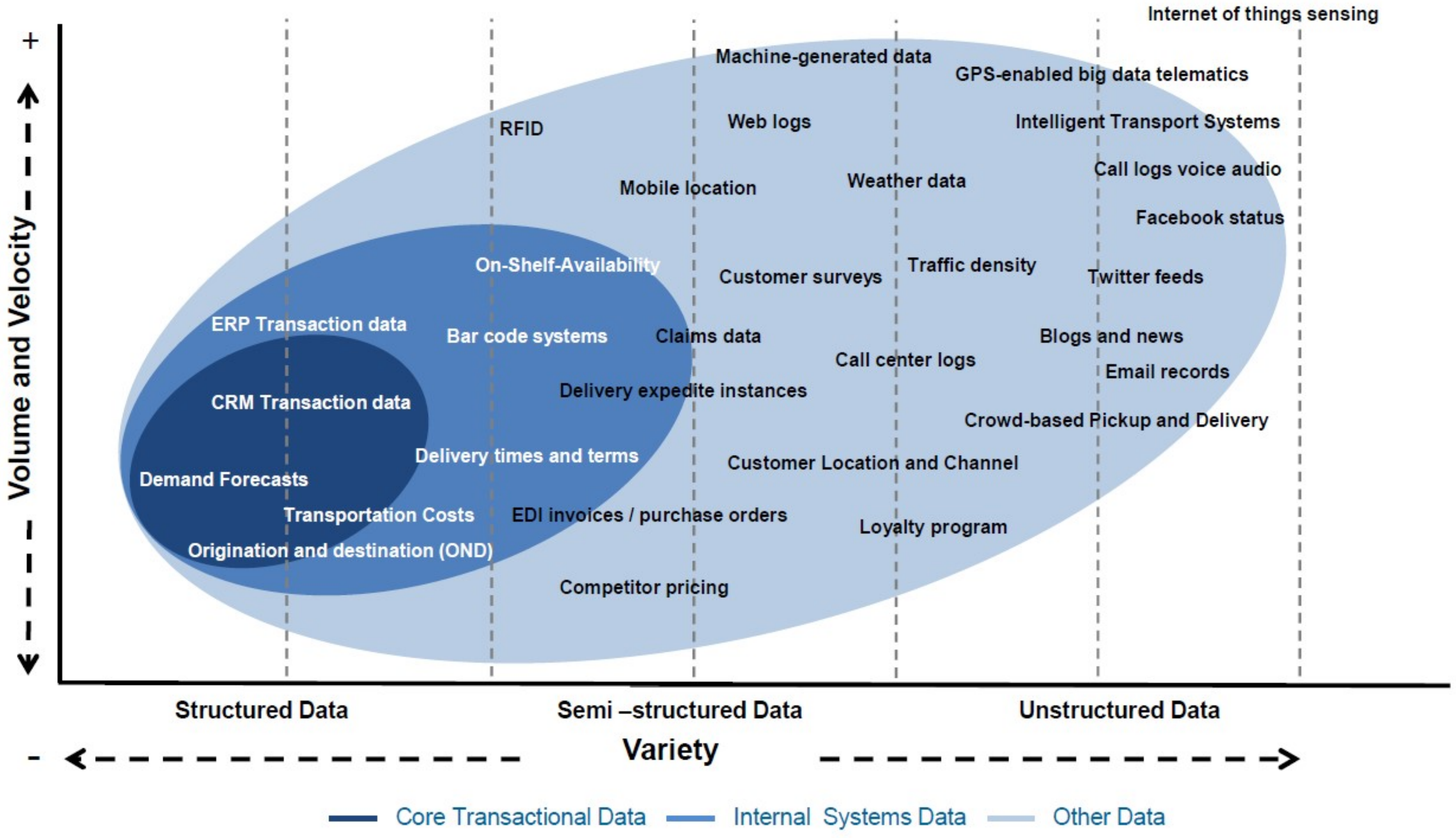
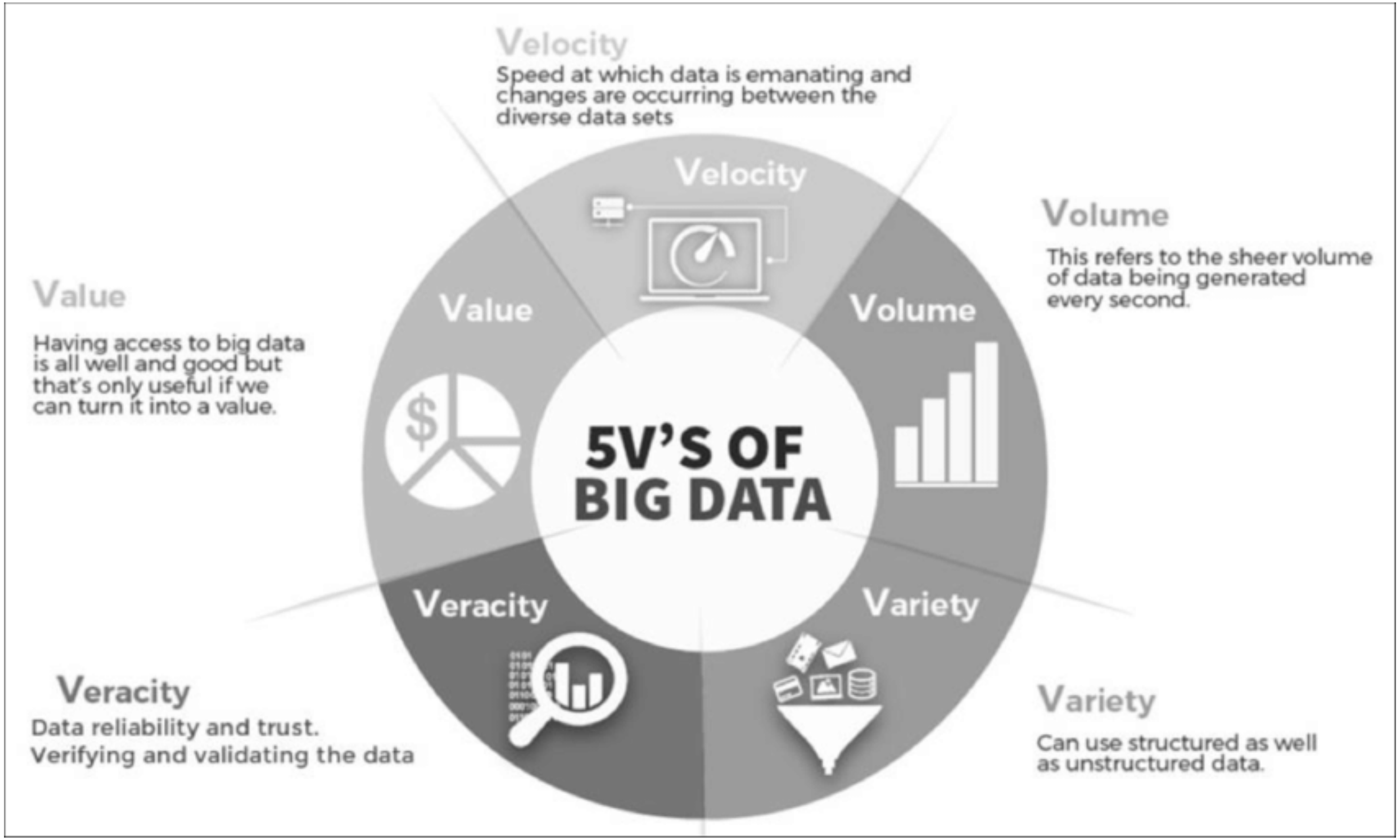


Figure 1. SCM Data Volume and Velocity vs. Variety

Big Data 5Vs



Velocity

Speed at which data is emanating and changes are occurring between the diverse data sets

Volume

This refers to the sheer volume of data being generated every second.

Value

Having access to big data is all well and good but that's only useful if we can turn it into a value.

Veracity

Data reliability and trust.

Verifying and validating the data

Variety

Can use structured as well as unstructured data.

The Scale

12+ TBs **of tweet data every day**

? TBs of data every day

25+ TBs **of log data every day**

30 billion RFID tags today (1.3B in 2005)

4.6 billion camera phones world wide

100s of millions of GPS enabled devices sold annually

76 million smart meters in 2009… 200M by 2014

2+ billion people on the Web by end 2011

Cloud Computing and Big Data

- One of the vital issues that organizations face with the **storage and management** of Big Data is the **huge amount of investment** to get the required hardware setup and software packages.
- Some of these resources may be **overutilized or underutilised with varying requirements overtime**. We can overcome these challenges by providing a set of computing resources that can be shared through cloud computing.
- Cloud platforms provide a **shared infrastructure** and **pay-as-you-go model** that lowers cost. This has helped enable the “big data revolution”

In-Memory Technology for Big Data

- Disks are really really slow! This might prevent us from analysing data in **real-time**
- **In-memory** big data computing tools overcome this issue by keeping the important data in memory.
  - But this isn't always possible! **Why not?**

What types of applications or analytics are a good fit for in-memory processing?

Big Data Techniques

- To analyze the datasets, there are many techniques available, some of which are as follows:
  - Massive Parallelism
  - Data Distribution
  - High-Performance Computing
  - Task and Thread Management
  - Data Mining and Analytics
  - Data Retrieval
  - Machine Learning
  - Data Visualisation

Fundamental Concepts of Distributed Computing Used in Big Data Analytics

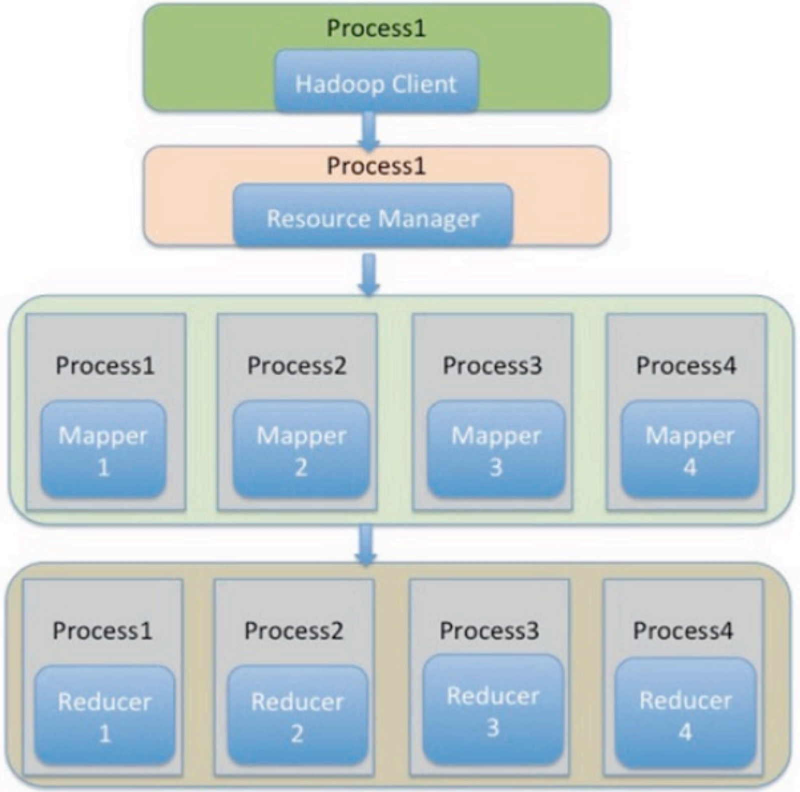
Prof. Tim Wood & Prof. Roozbeh Haghnazar

Big Data and…

- **Heterogeneity**
- **Openness**
- **Security**
- **Failure Handling**
- **Concurrency**
- **Quality of Service**
- **Scalability**
- **Transparency**

Map Reduce / Hadoop

- The first popular "big data" platform for the cloud
- Can install **Apache Hadoop** on your own **IaaS** VMs/containers
- Or use a **PaaS** version:
  - AWS Elastic Map Reduce, Google Dataproc, Azure HDInsight
- MapReduce uses distributed systems concepts to achieve high scalability
  - Partitioning, failure detection, data replication for performance and reliability…
- MapReduce relies on a cloud infrastructure to provide its computational and storage resources



Multiprocessing model in the Hadoop runtime environment

MAP REDUCE **QUALITY OF SERVICE**

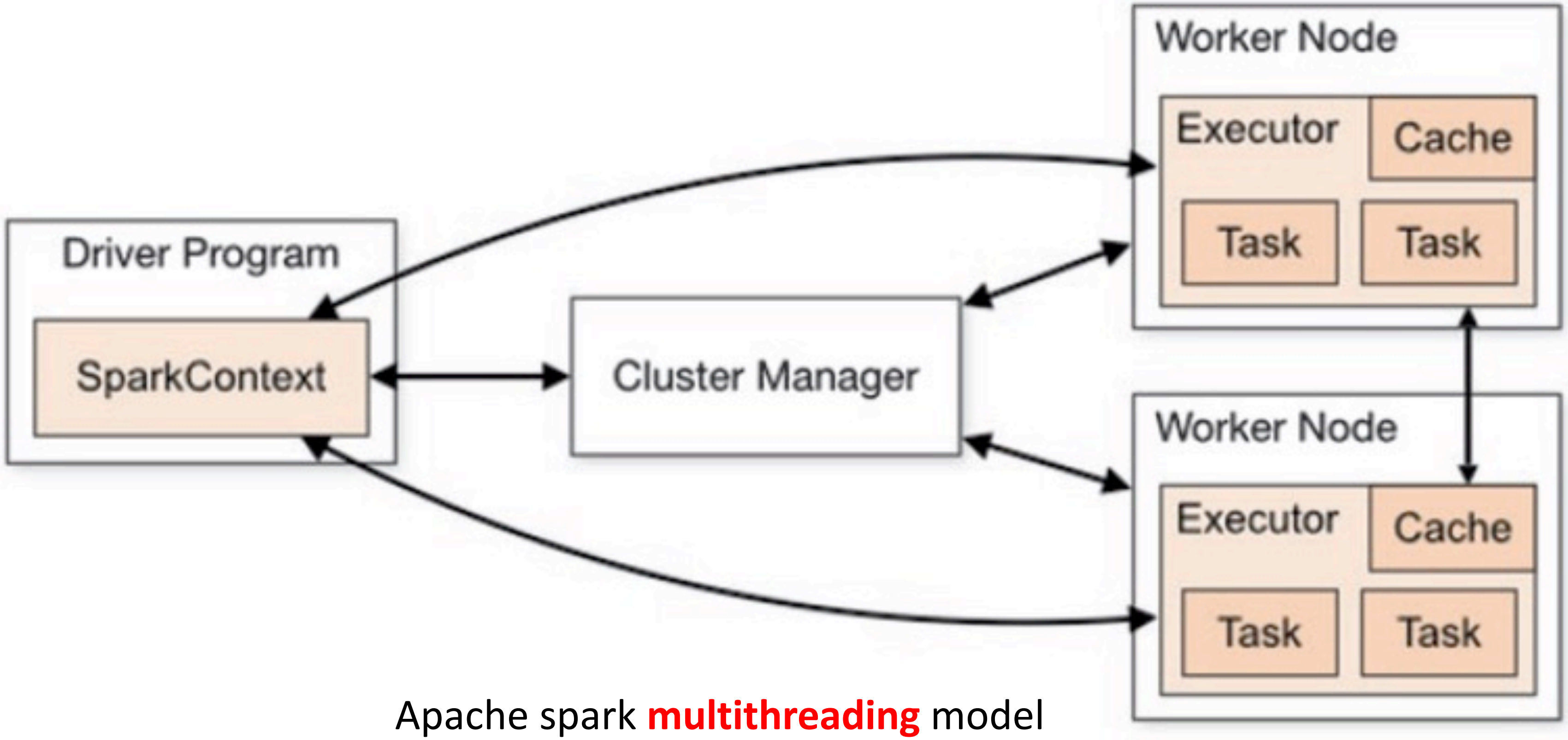
- We already discussed how Haddop/MR can have different scheduling algorithms to decide which tasks to process when there are several jobs

- Straggler issue: It can be difficult to tell the difference between a really slow and a failed replica
  - MapReduce solution:

Stream Processing

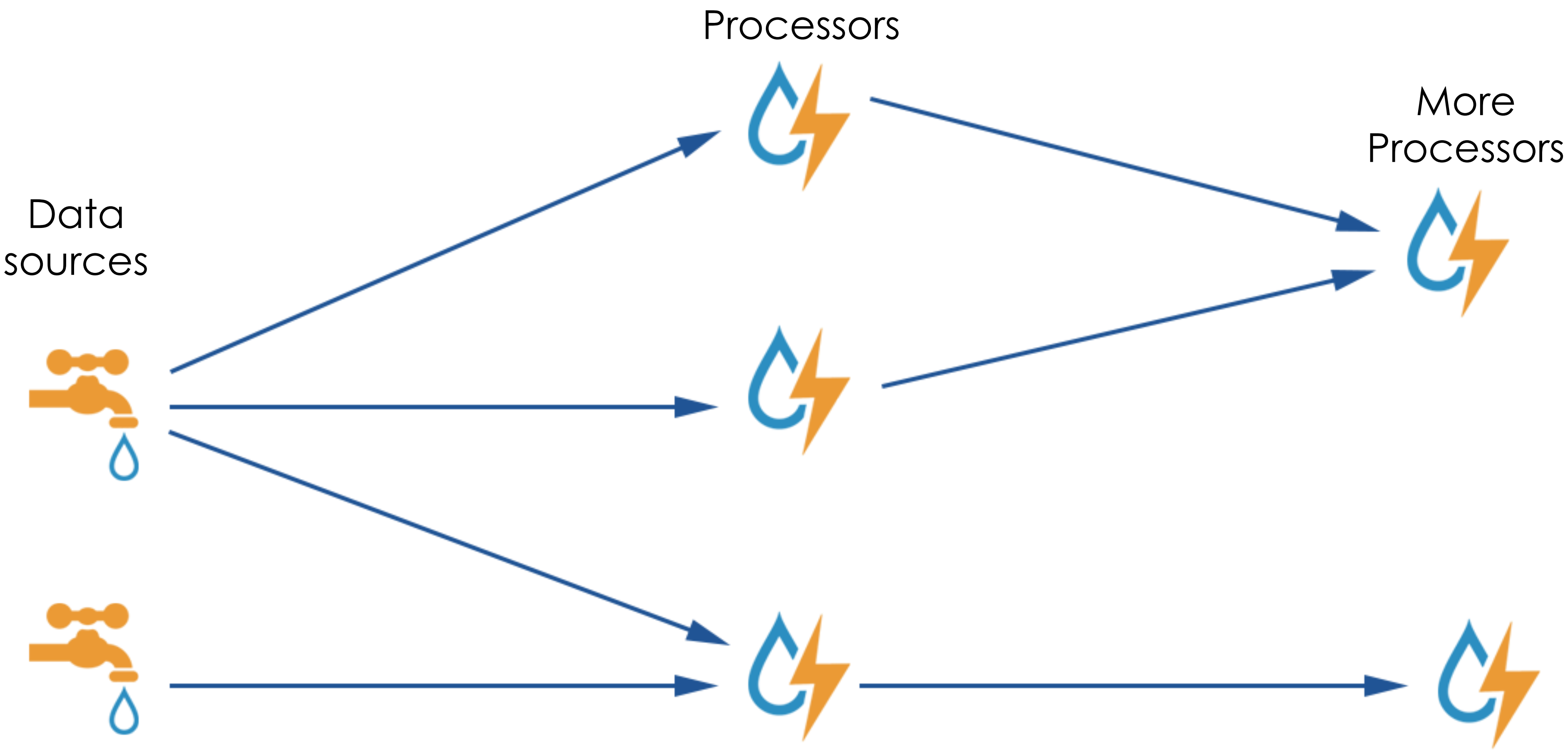
- MR/Hadoop are for **batch** processing
  - Long running jobs (minutes, hours total)
- Sometimes you want **stream** processing
  - Continuously arriving data with millisecond scale response
- **Storm** and **Spark** are basically Hadoop for streams
  - Define a graph of processing nodes
  - Stream data through the graph
  - Manage the workers (each executing a part of the graph)
  - Detect failure, carefully buffer data in queues, etc

SPARK REQUEST HANDLING MODLE



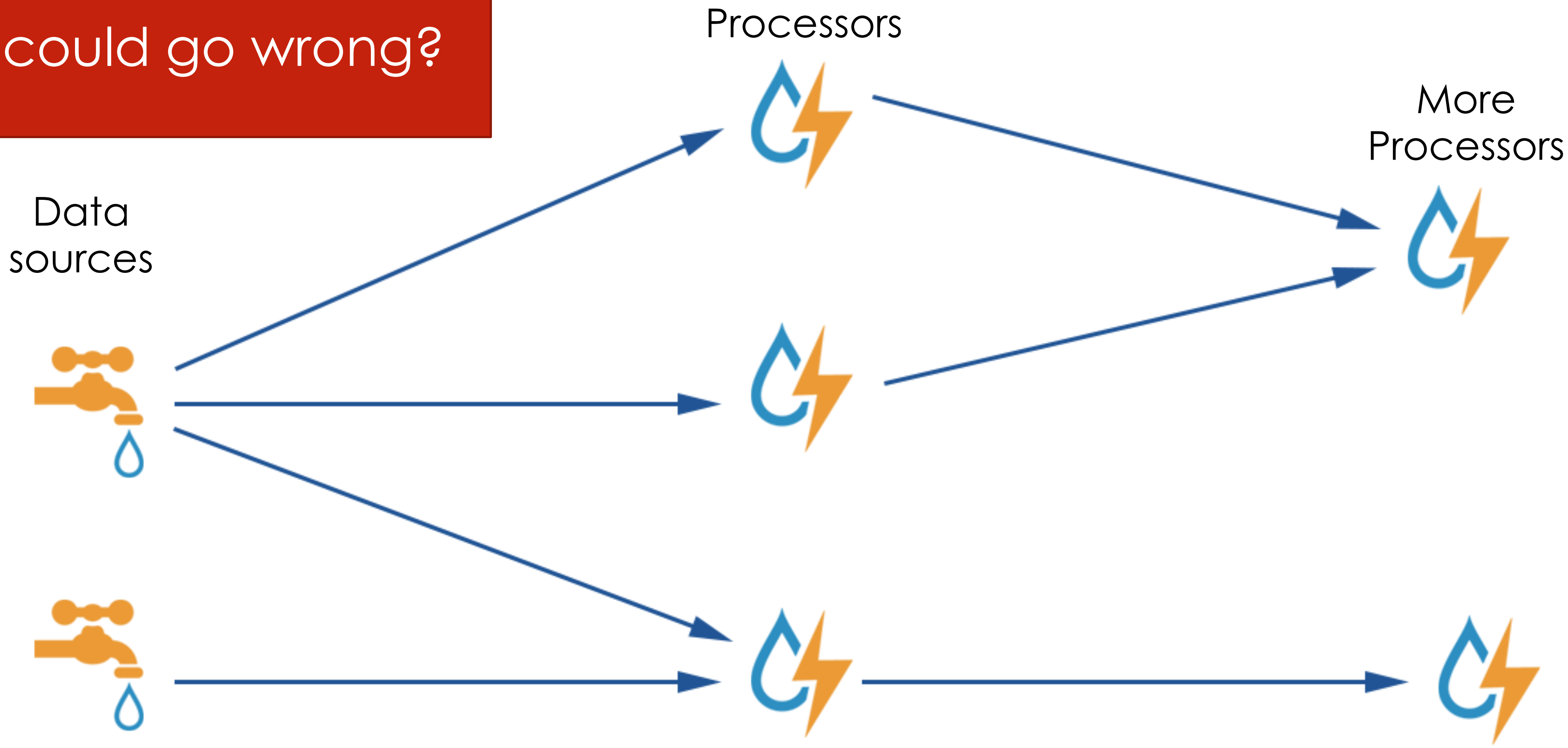
Apache spark **multithreading** model

Stream Processing



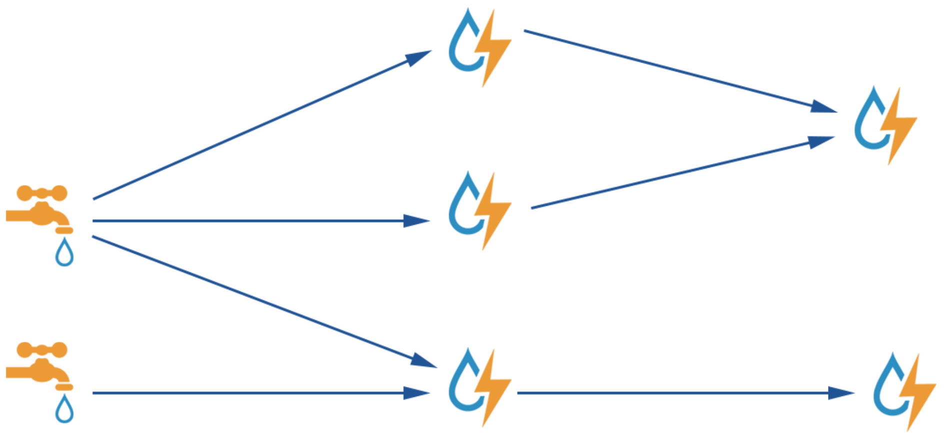
Stream Processing

What could go wrong?



STORM FAULT TOLERANCE

- Data is constantly arriving -> handling faults is more difficult
- We want to understand the **fault tolerance semantics** provided by the stream processing system



- **Best effort**: no guarantees
- **At least once**: a data element will be fully processed, but it might be processed several times
- **Exactly once**: a data element will be fully processed, precisely one time by each step

Stream Processing **Scalability**

- Need to manage scalability of stream processing workers

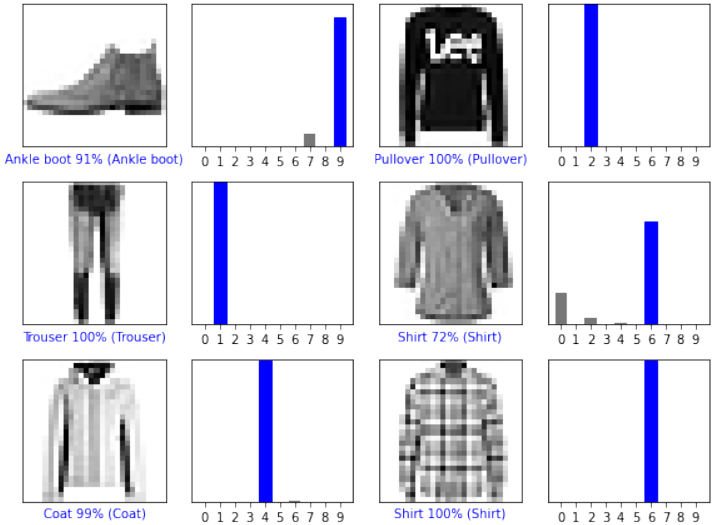
- How to make data available to workers?
  - Hadoop Distributed File System would be too slow

- **Kafka** Message Queue
  - Distributed queue of requests
  - Queue can be scaled up and down as needed
  - Can support large numbers of workers accessing data concurrently
  - Currently uses **Zookeeper** for consistency, next version will use **Raft**

Kafka

Tensor Flow

- Library from Google primarily for training and running Deep Learning models
  - Designed in part by Jeff Dean (watch his talk!)
- Training machine learning models takes a lot of data
- Doing training on a single machine can be too slow
  - Or a single machine may not have enough memory
- TensorFlow handles all the ML math and helps with distributing the training and inference tasks across multiple devices



TENSOR FLOW **HETEROGENEITY**

- Many ML tasks are highly parallelizable ☺
  - But they require a lot of data -> high communication costs ☹
- GPUs are very good for this
- A network of distributed GPUs is even better…
- But custom hardware is best!

- TPU = Tensor Processing Unit
  - Customized processor specifically for tensor flow

- TensorFlow software library needs to support all of these, possibly spread across multiple servers

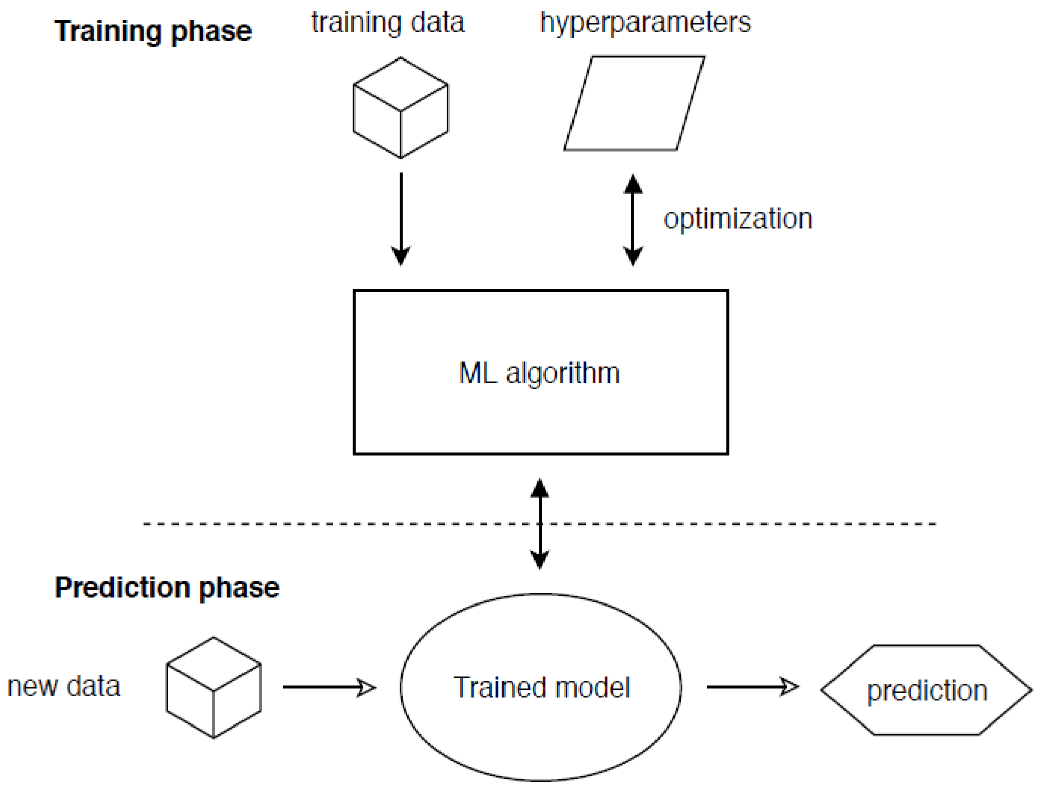
Machine Learning and Distributed Systems

Prof. Tim Wood & Prof. Roozbeh Haghnazar

General ML problem

- General Overview of Machine Learning. During the training phase a ML model is optimized using training data and by tuning hyper parameters. Then the trained model is deployed to provide predictions for new data fed into the system.

- Aside from choosing a suitable algorithm for a given problem, we also need to find an optimal set of hyperparameters for the chosen algorithm

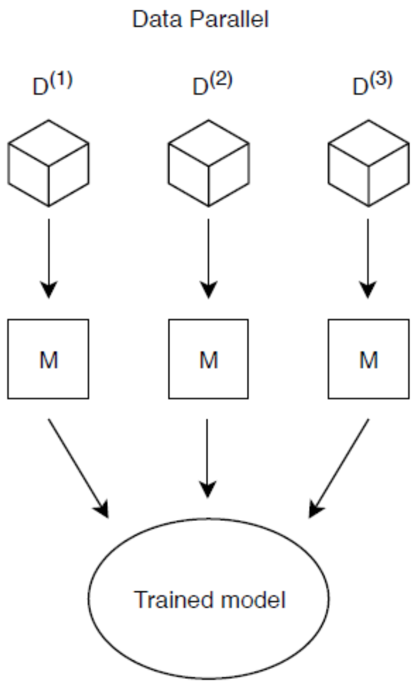


Distribution of Training Phase

- **Data-Parallel**
- **Model-Parallel**

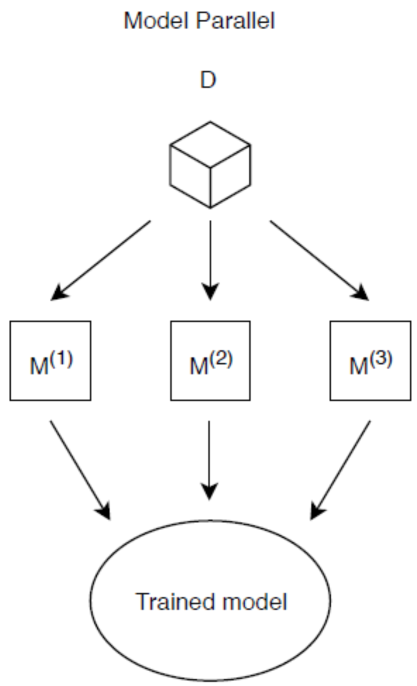
Distribution of Training Phase

- In the Data-Parallel approach, the data is partitioned as many times as there are worker nodes in the system and all worker nodes subsequently apply the same algorithm to different data sets.
- The same model is available to all worker nodes (either through centralization, or through replication) so that a single coherent output emerges naturally.
- The technique can be used with every ML algorithm with an independent and identically distribution (i.i.d.) assumption over the data samples



Distribution of Training Phase

- In the Model-Parallel approach, exact copies of the entire data sets are processed by the worker nodes which operate on different parts of the model.
- The model is therefore the aggregate of all model parts.
- The model-parallel approach cannot automatically be applied to every machine learning algorithms because the model parameters generally cannot be split up.



Machine Learning Algorithms

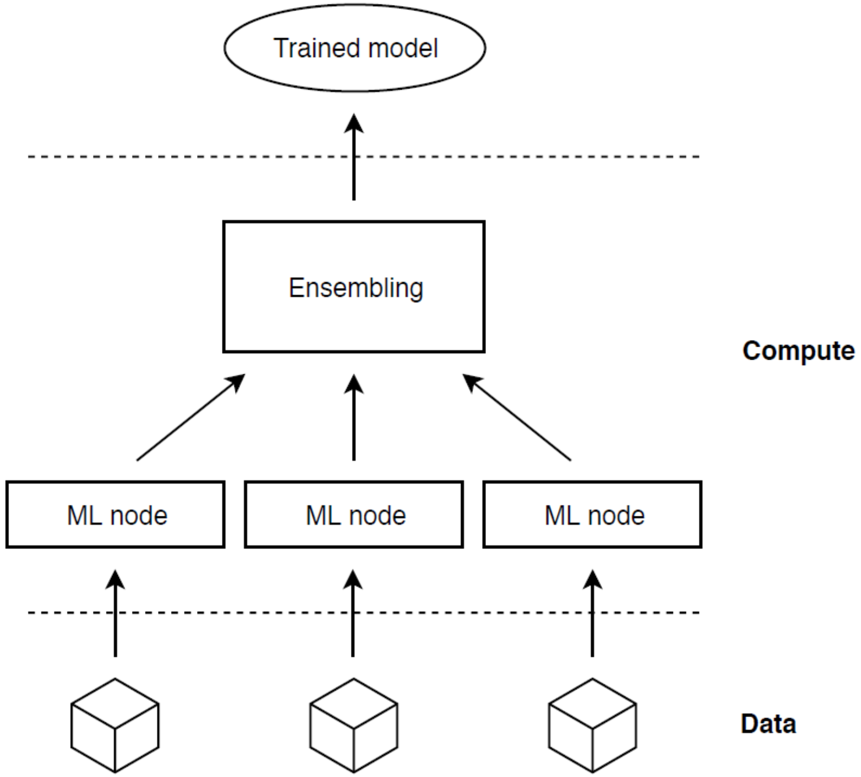
- We categorize current ML algorithms based on the following three characteristics:
  - **Feedback**, the type of feedback that is given to the algorithm while learning
    - ***Supervised learning***
    - ***Unsupervised learning***
    - ***Semi-supervised learning***
    - ***Reinforcement learning***
  - **Purpose**, the desired end result of the algorithm
    - ***Anomaly detection***
    - ***Classification***
    - ***Clustering***
    - ***Dimensionality reduction***
    - ***Representation learning***
    - ***Regression***
  - **Method**, the nature of model evolution that occurs when given feedback
    - ***Evolutionary algorithms (EAs)***
    - ***Stochastic gradient descent (SGD) based algorithms***
      - Support vector machines (SVMs)
      - Artificial neural networks (ANNs)
      - Perceptrons
    - ***Rule-based machine learning (RBML) algorithms***
    - ***Topic Models (TM)***
    - ***Matrix Factorization***

TOPOLOGIES

- Centralized (Ensembling)
- Decentralized (Tree)
- Decentralized (Parameter Server)
- Fully Distributed (Peer to Peer)

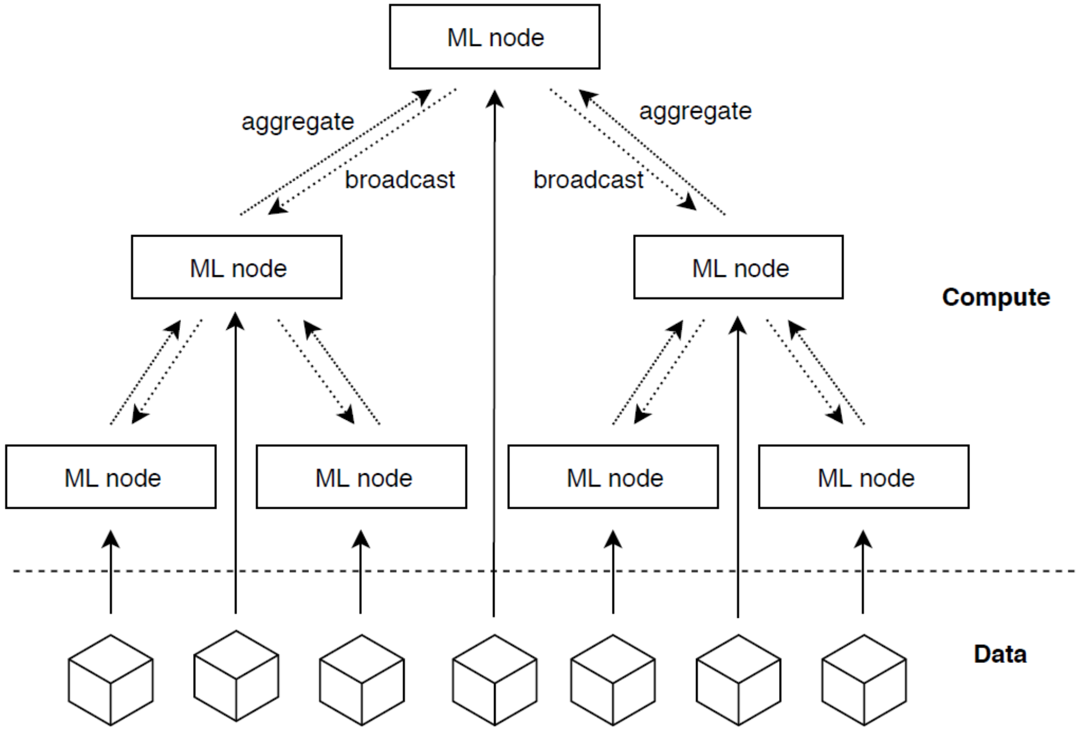
Centralized (Ensembling)

- employ a strictly hierarchical approach to aggregation, which happens in a single central location.



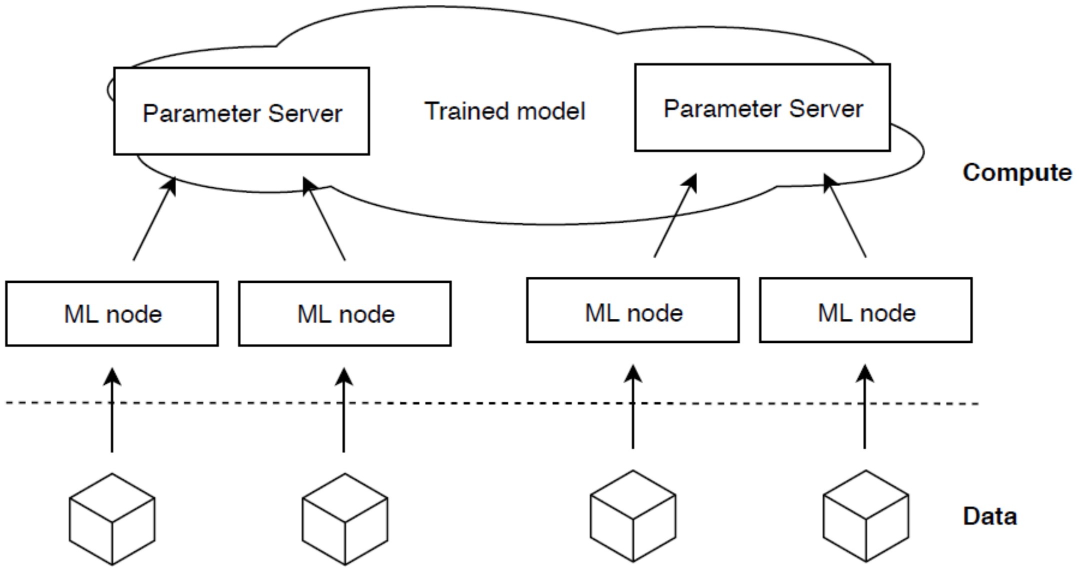
Decentralized (Tree)

- allow for intermediate aggregation, with a replicated model that is consistently updated when the aggregate is broadcast to all nodes such as in tree topologies



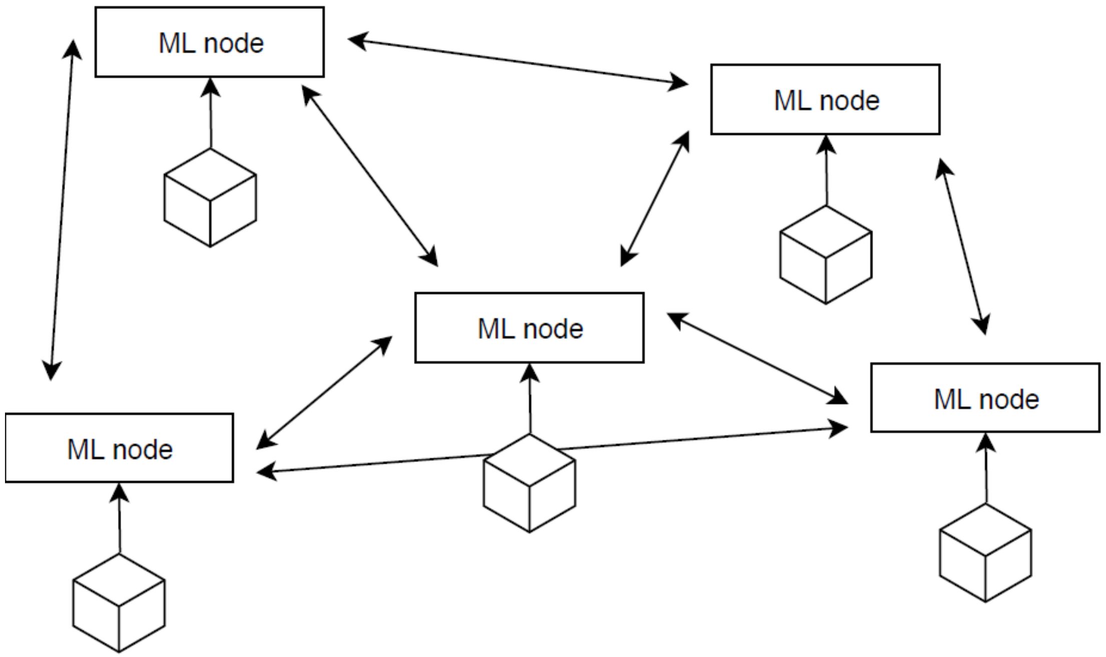
DECENTRALIZED (PARAMETER SERVER)

- allow for intermediate aggregation, with a partitioned model that is sharded over multiple parameter servers

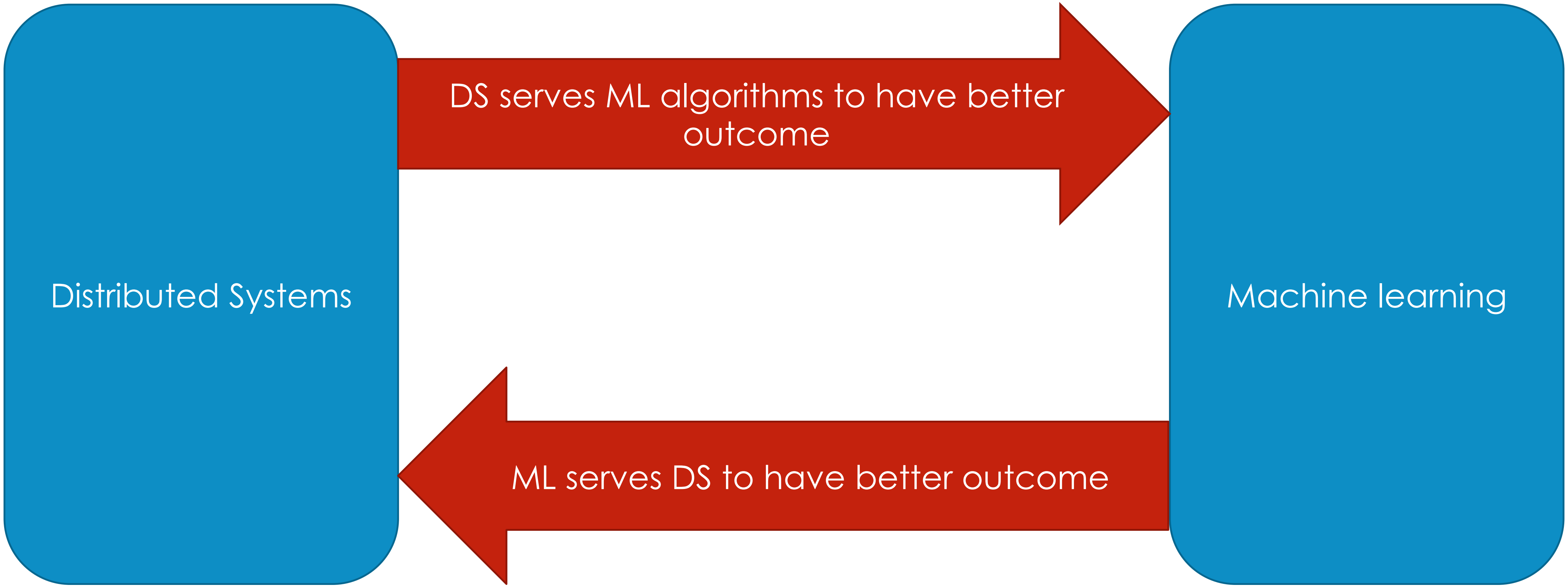


Fully Distributed (Peer to Peer)

- consists of a network of independent nodes that ensemble the solution together and where no speci!c roles are assigned to certain nodes.



Coexistence of Distributed system and Machine Learning



Distributed Systems

DS serves ML algorithms to have better outcome

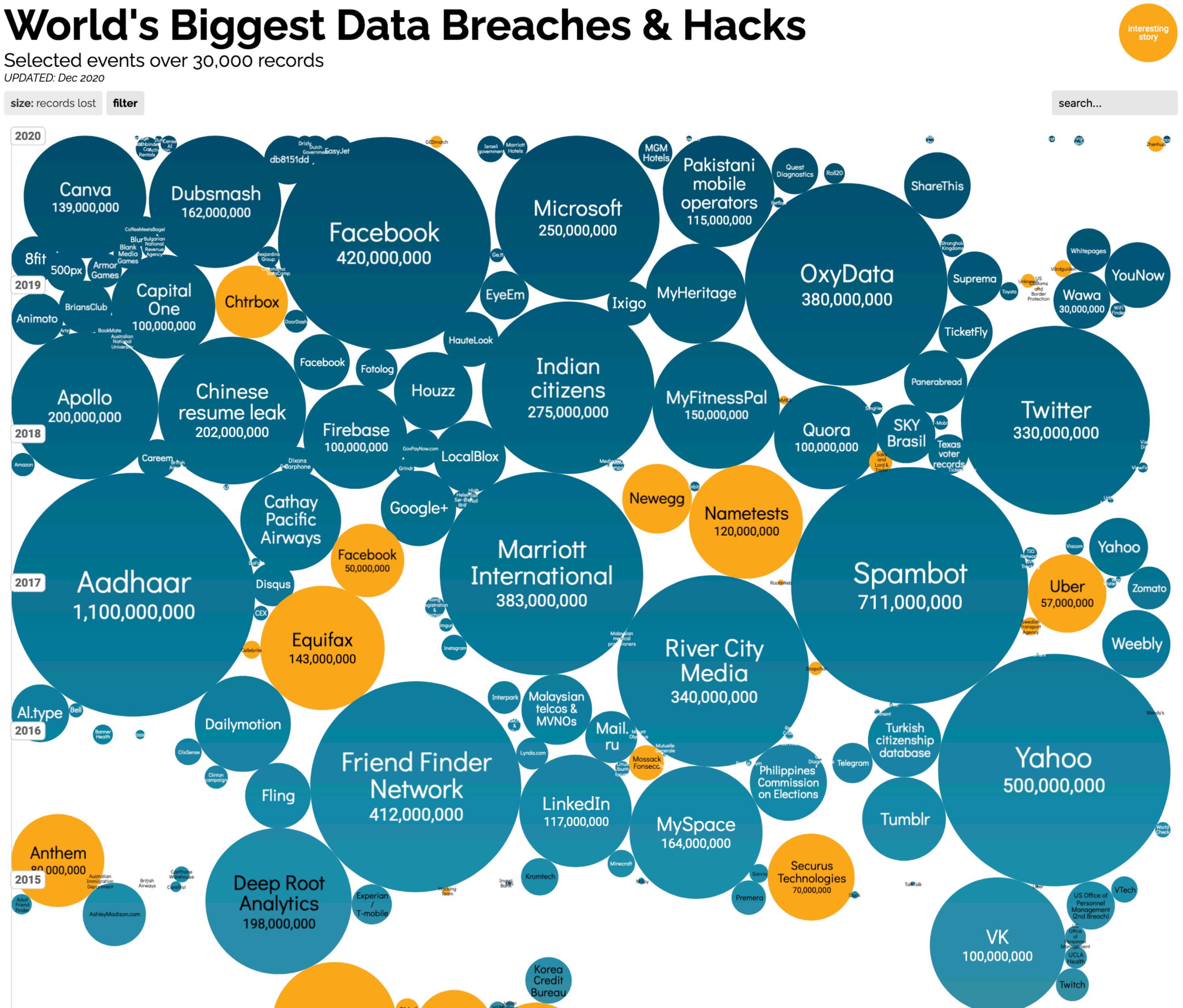
ML serves DS to have better outcome

Machine learning

Big Data and Cloud Security

Prof. Tim Wood & Prof. Roozbeh Haghnazar

- Big data is often sensitive!
- Data breaches let attacker steal vast quantities of personal information
- We need a way to keep data **secure** and **private**



World's Biggest Data Breaches & Hacks

Selected events over 30,000 records

UPDATED: Dec 2020

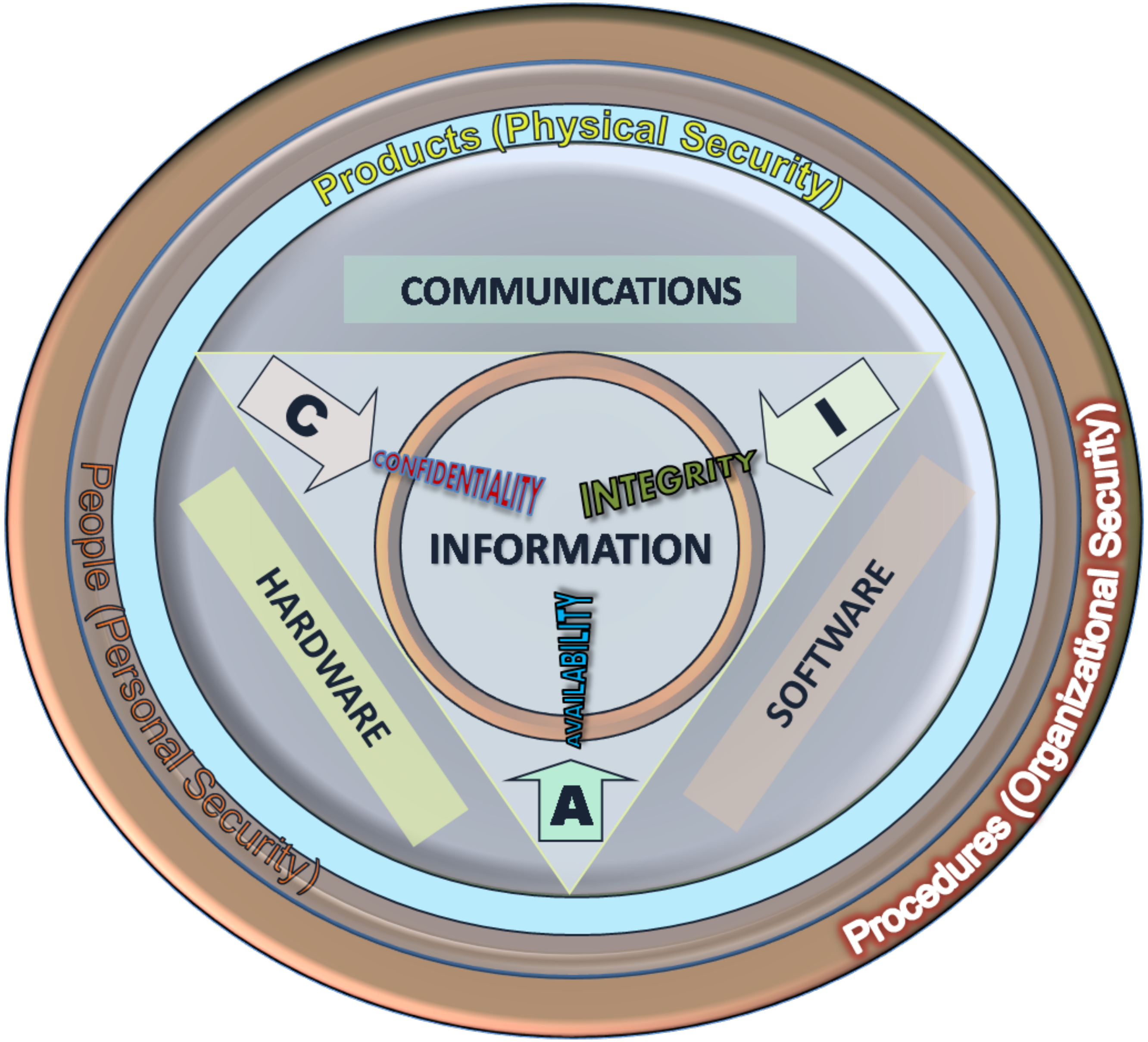
size: records lost **filter**

search...

https://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/

CIA Triad

- *A system should provide…*
- **Confidentiality**: only authorized users are able to access data
- **Integrity:** data cannot be manipulated
- **Availability:** data can be accessed when it is needed


Products (Physical Security)
COMMUNICATIONS
C
I
CONFIDENTIALITY
INTEGRITY
INFORMATION
AVAILABILITY
HARDWARE
SOFTWARE
A
People (Personal Security)
Procedures (Organizational Security)

What is an attack or technique that hurts these goals?

Cloud Security

How is security affected by a **distributed system?**

How is security affected by using **the cloud?**

Cloud Security

- How is security affected by a **distributed system**?
  - Almost all software is distributed these days!
  - Distributed systems rely on communication -> open ports are potential weaknesses
  - Many distributed algorithms (like leader election, Raft, etc) can't handle malicious nodes (Byzantine faults)

- How is security affected by using **the cloud**?
  - Many cloud applications are "public facing" -> what if your users are malicious?
  - Clouds share resources for multiple tenants -> what if collocated VMs are malicious?
  - What if the cloud is malicious???

Prof. Tim Wood & Prof. Roozbeh Haghnazar

Distributed Systems Wrap Up

Distributed Systems are Hard

- Difficult to design, implement, debug, and test
- Key recurring ideas:
  - Replication, partitioning, communication, locking, consensus, ordering
- Lots of building blocks to go on top of

Distributed Systems are Everywhere

- Every web application, almost all mobile applications, etc

- Let us know what you do with them in the future!
- Connect on LinkedIn, email us about your job experience!

- Full out the course evaluation form!