

Eötvös Loránd University (ELTE)
Faculty of Informatics (IK)
Pázmány Péter sétány 1/c
1117 Budapest, Hungary



APPLIED CYBERSECURITY

*Open Source Technologies for Real-Time Data
Analytics*

Imre Lendák, PhD, Associate Professor

Cybersecurity intro outline

- Cybersecurity incidents
- Key definitions
- Threat types & sources
- Challenges
- Vulnerabilities
- Security monitoring data





Positions in Cybersecurity

- Chief Information Security Officer (CISO)
- Cybersecurity Manager
- **Cybersecurity Architect**
- Security Auditor
- **Security Analyst – often data-intensive!**
- **Penetration tester**
- Incident response manager
- **Digital forensic expert**

- Positions marked with bold font require thorough understanding of the topics and tools discussed!

CYBER INCIDENTS



High profile incidents...

Stuxnet (2010)



Ukraine (2015)

BBC | Sign in

News Sport Weather Shop Earth Travel More

NEWS

Home | Video | World | UK | Business | Tech | Science | Magazine | Entertainment & Arts

Technology

Hackers caused power cut in western Ukraine - US

12 January 2016 | Technology



Ukraine has been forced to turn to back-up power sources in recent months following a spate of power cuts



Stuxnet sources

- Alex Gibney, “Zero Days”, documentary, 2016
- Kim Zetter, “Countdown to Zero Day: Stuxnet and the Launch of the World's First Digital Weapon”, Broadway Books, September 2015
- David Kushner, “The Real Story of Stuxnet”, IEEE Spectrum, February 2013,
<http://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet>



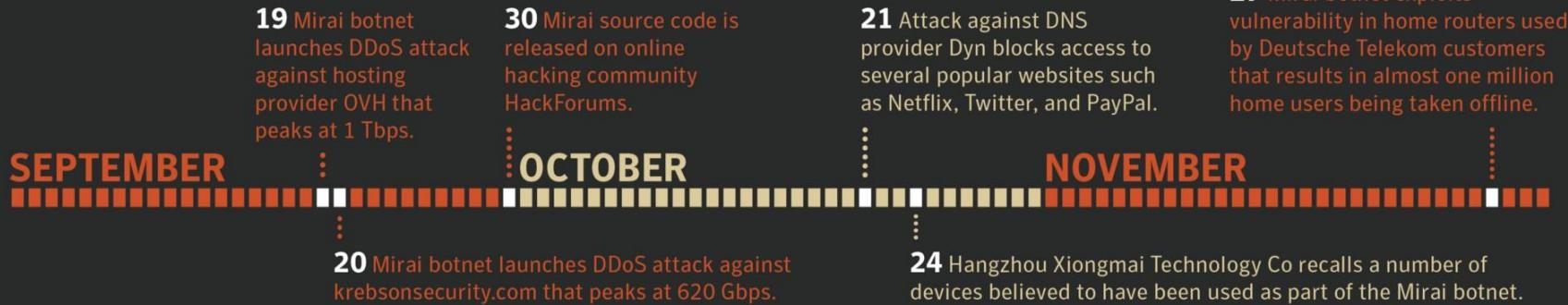
Ukraine 2015 & 2016 literature

- SANS, “Analysis of the Cyber Attack on the Ukrainian Power Grid”, Report, March 2016
- ICS-CERT, “Cyber-Attack Against Ukrainian Critical Infrastructure”,
- National Cybersecurity and Communications Integration Center (NCCIC), “Seven Strategies to Defend ICSs”, December 2015
- Wikipedia, “December 2015 Ukraine power grid cyber attack”,
- Kim Zetter (2016). "Everything We Know About Ukraine's Power Plant Hack". Wired. January 20th, 2016.
- Kim Zetter (2016). "Inside the Cunning, Unprecedented Hack of Ukraine's Power Grid". Wired. March 3rd, 2016.
- Hackernews (2016), “Hackers Suspected of Causing Second Power Outage in Ukraine”, <http://thehackernews.com/2016/12/power-outage-ukraine.html>

...more high profile incidents...



▼ Mirai's trail of disruption in 2016





Mirai botnet (2016) sources

- Lily Hay Newman, "What We Know About Friday's Massive East Coast Internet Outage". Wired.
- Nate Lanxon, Jeremy Kahn & Joshua Brustein, "The Possible Vendetta Behind the East Coast Web Slowdown", Bloomberg.com
- Darrell Etherington & Kate Conger, "Many sites including Twitter, Shopify and Spotify suffering outage". TechCrunch.
- Wikipedia, "2016 Dyn cyberattack"
- Wikipedia, "Mirai (malware)"

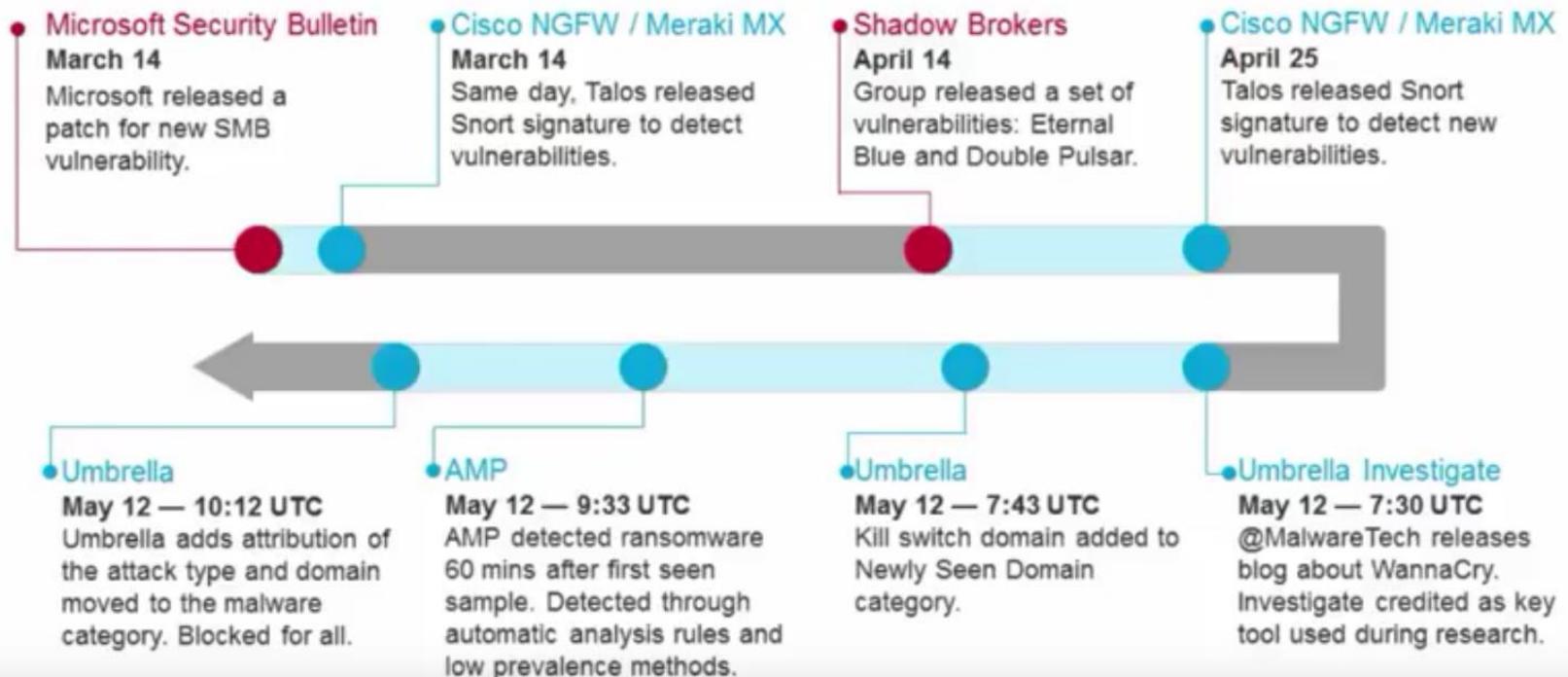


...more high profile incidents...

Anatomy of the attacks: WannaCry ransomware & Google OAuth phishing



Timeline of WannaCry Ransomware



Cisco Umbrella

▶ 🔍 43:26 / 52:46

CC 🔍 YouTube 🔍



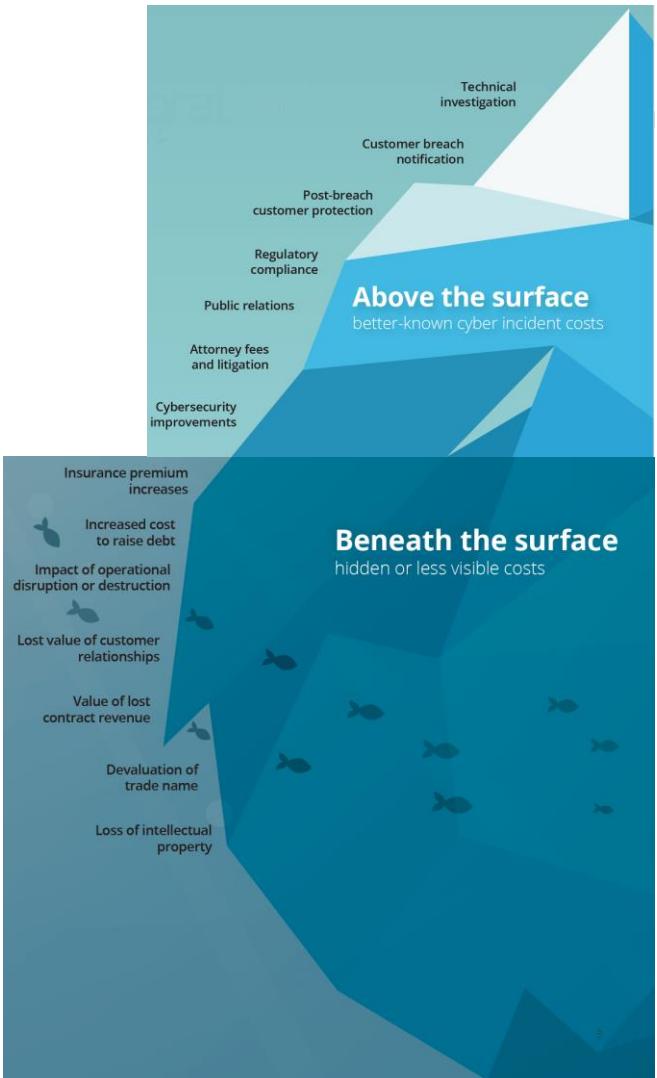
WannaCry (2017) sources

- Thomas P. Bossert, "It's Official: North Korea Is Behind WannaCry", The Wall Street Journal.
- Thomas Fox-Brewster, "An NSA Cyber Weapon Might Be Behind A Massive Global Ransomware Outbreak". Forbes.
- Victoria Woollaston, "Wanna Decryptor: what is the 'atom bomb of ransomware' behind the NHS attack?". Wired UK.
- Wikipedia, "WannaCry ransomware attack"



Cyberattack impact

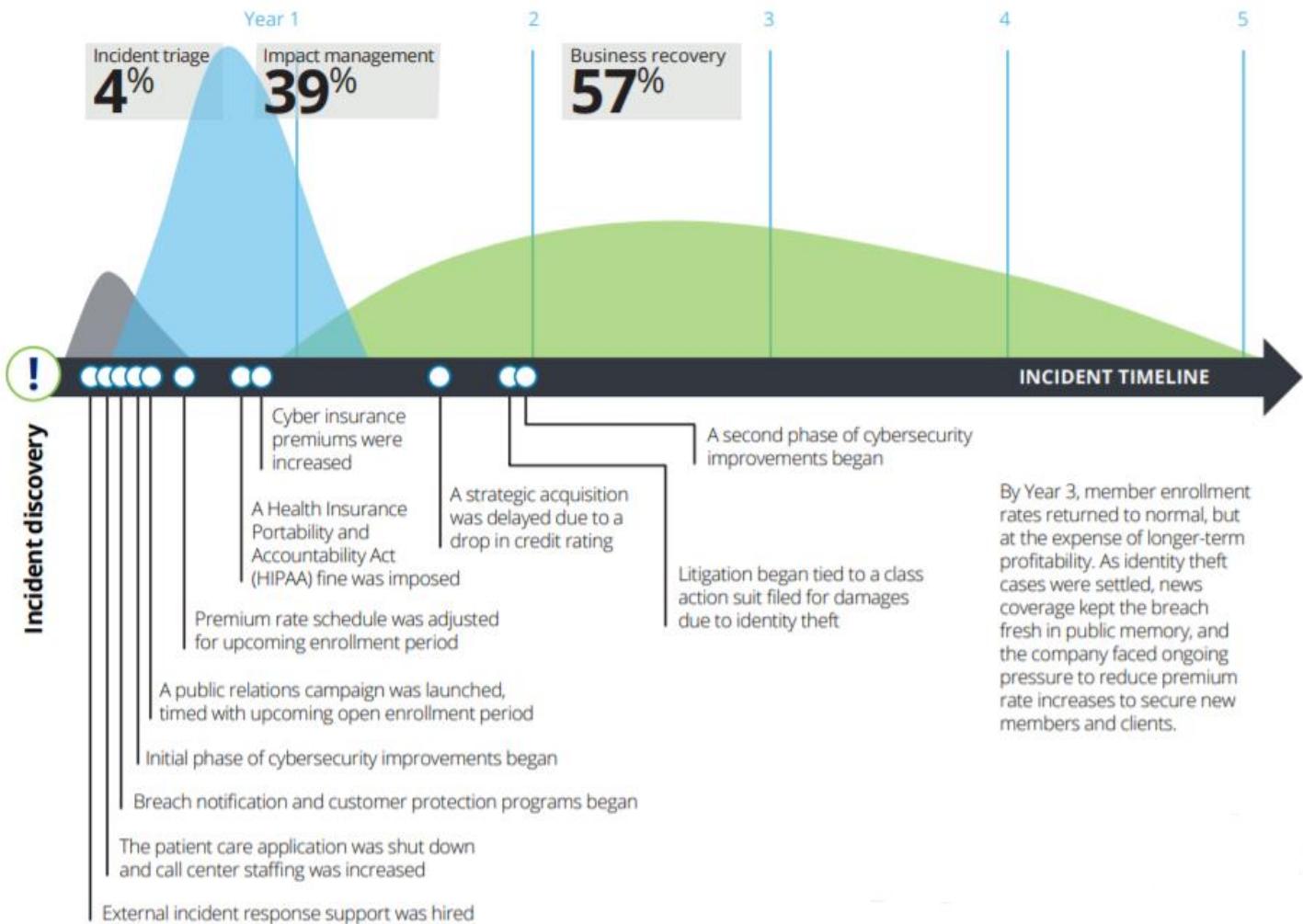
- Above the surface 7/14
 - Technical investigation
 - Customer breach notification
 - Post-breach customer protection
 - Regulatory compliance
 - Public relations
 - Attorney fees and litigation
 - Cyber Security improvements
- Underneath the surface 7/14
 - Service price increase
 - Increased cost to raise debt, i.e. borrow money
 - Impact of operational disruption or destruction
 - Lost value of customer relationships
 - Value of lost contract revenue
 - Devaluation of trade name
 - Loss of intellectual property



<https://www2.deloitte.com/content/dam/Deloitte/us/Documents/risk/us-risk-beneath-the-surface-of-a-cyber-attack.pdf>



Cyber incident timeline



<https://www2.deloitte.com/content/dam/Deloitte/us/Documents/risk/us-risk-beneath-the-surface-of-a-cyber-attack.pdf>

Open source tools & technologies

CYBERSECURITY DEFINED



Common information security goals

- Confidentiality
- Integrity
- Availability
- Authenticity
- Non-repudiation
- Privacy



<https://www.comtact.co.uk/blog/what-is-the-cia-triad>



Cybersecurity elements

- **Threat** – anything that has the potential to cause serious harm to an information system
- **Vulnerability** – a flaw (often unintentional) in a system that can leave it open to attack
- **Attack** – attempt to expose, alter, disable, destroy, steal or gain unauthorized access to or make unauthorized use of an asset – usually by exploiting a vulnerability
- **Risk** – a combination of the likelihood and impact of a security incident, e.g. attack or vulnerability-related outage
- **Security controls** – safeguards or countermeasures to avoid, detect, counteract, or minimize risks to physical property, information, computer systems, or other assets
- **Trust** – degree of trust end users have in/about a system



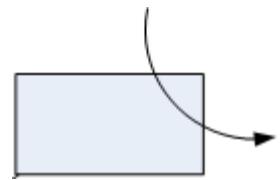
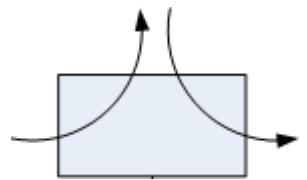
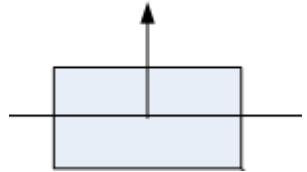
Vulnerabilities

- Information systems have physical and electronic **access points** (PAP & EAP)
 - Physical: gates, doors, windows, human resources, mobile devices (e.g. laptops, mobiles, USB drives), switches, routers
 - Electronic: website, remote access, software running on physical access points
- Potential **locations** of vulnerabilities: all access points
- Most **frequent locations** of vulnerabilities (2019):
 - Human resources, e.g. employees, subcontractors, visitors
 - Publicly available electronic access points, e.g. websites
 - Mobile devices, e.g. USB drives, laptops

THREAT TYPES & SOURCES

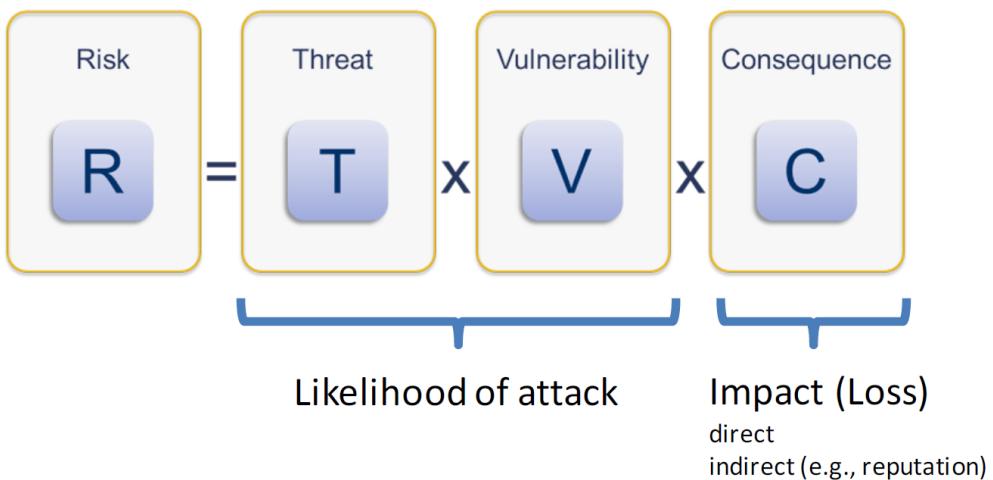
Threat types

- **Interception** – an unauthorized entity gains access to services and/or data, e.g. spyware
- **Interruption** – service or data becomes unavailable due to corruption or cyber attack, e.g. Distributed Denial of Service (DDoS)
- **Modification** – unauthorized change in data and/or services → deviation from specification, e.g. unwanted change of middleware
- **Fabrication** – generate unwanted data, services or requests which would not exist during normal operation, e.g. add entries to a shadow (password) file



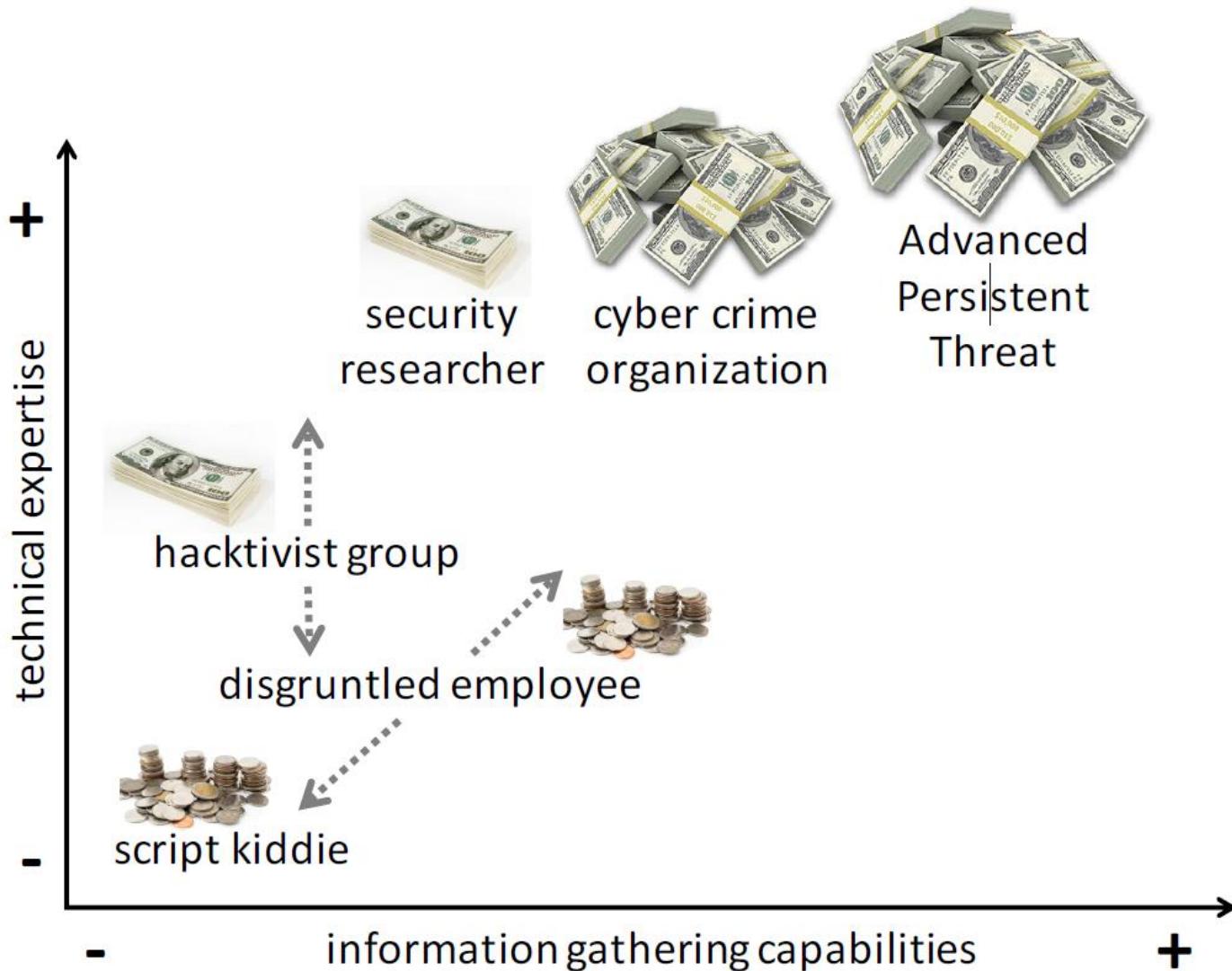
Risk in cyberspace

- Risk in cyberspace is a composition of:
 - **Threat** - a product of both intention and capability
 - **Vulnerability** - depends on the characteristics of the target and the probability that an attempted attack will be successful
 - **Consequences** - political, social, economic or environmental damages or costs caused by a successful attack



* Computer Security course content, CrySyS Lab, BME, Budapest, Hungary

Threat sources



* Computer Security course content, CrySyS Lab, BME, Budapest, Hungary



Cyber Kill Chain model (steps)

- **Reconnaissance** – analyze all access points and find vulnerability of the target, e.g. company, person
- **Weaponization** – create an attack tool/process to exploit the vulnerability
- **Delivery** – deliver the ‘weapon’
- **Exploitation** – initiate the attack
- **Installation** – install tools necessary to reach objectives
- **Command-and-Control** – maintain persistence in the breached system
- **Actions on Objectives** – perform the planned actions, e.g. exfiltrate data, interrupt normal business activities

CYBERSECURITY THREAT SOURCES

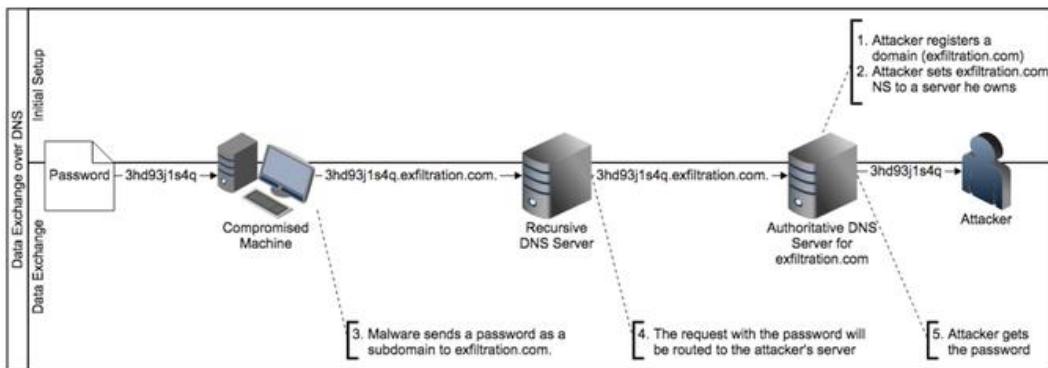


External threats

- A **criminal organization** or an **adversarial nation state** executes a zero day attack against a financial institution's public web server
 - www.examplebank.hu is accessed by the attackers
- The attackers pivot (i.e. move laterally inside the bank's information system) into the internal network
- The attackers gain access to the database server without getting noticed
- The attackers want to exfiltrate data about the bank's customers through DNS by installing appropriate custom scripts

Internal threats

- Mick is a malicious insider within a hedge fund investing large amounts for investors, i.e. bank employee with bad intentions
- Mick wants to exfiltrate the fund's investment strategies to a competitor and get paid for that
- Mick receives malware on a USB drive, inserts it in his workstations and the malware infects the system
- The malware exfiltrates trade secrets and receives commands via DNS queries/responses



<https://blogs.akamai.com/2017/09/introduction-to-dns-data-exfiltration.html>

VULNERABILITIES

Cybersecurity vulnerabilities

- Social engineering
- Software bugs
- Malware
- Misused DNS
- Misused BGP
- ...
- The above is not a definite list!





Social engineering

- **DEF:** Social engineering is psychological manipulation of human resources into performing actions or revealing confidential information
 - **Interception:** either at the physical or electronic access points, e.g. voice or video recording devices, installed keyloggers or other spyware
 - **Shoulder surfing:** often seen in films, involves the attacker coming near the victim and visually capturing sensitive information
 - **Impersonation:** pose as a colleague, system administrator or top manager and convince the victim to perform the desired actions.
 - **Phishing:** a subtype of impersonation, involves the attacker impersonating a trusted service and asking for secrets



Social engineering – CEO fraud

From eventos@grupocoimbra.org.br ★
Subject **Newsletter #1 of the XI General Assembly and X International Seminar of the Coimbra Group of Brazilian Universities**
To Me <vmat@vmat.rs> ★
Date 18 Jul 2018 09:44:39 -0300
Message ID <20180718124435.B34AB17FF8A@saasauth0006.smtpdlv.com.br>
Return-Path <bounce-d57865a1ee86992704754c7ed577ba99@smtplw-09.com>
Delivered-To: vmat@vmat.rs

To protect your privacy, Thunderbird has blocked remote content in this message.

GCUB Logo

Brasilia, July 18, 2018

Your Magnificence
Imre Lendák
President
Hungarian Academic Council of Vojvodina

Dear Sir/Madam,

Please receive the best compliments from the Coimbra Group of Brazilian Universities (CGBU) and the Hungarian Rectors' Conference (HRC).

In this email, you will find attached the **Newsletter #1 of the XI General Assembly and X International Seminar of the Coimbra Group of Brazilian Universities (CGBU)**.

Please note that the deadline for hotel reservations without penalty for cancellation or change of date is **July 23, 2018**. Check below some suggestions of hotels with special rates:

Mercure Budapest Korona (<https://bit.ly/2KURJhl>)****



Software bugs & backdoors

- **DEF:** Software bugs are unintentional mistakes in source code which cause services to produce incorrect or unexpected results, or to behave in unintended ways
 - Might cause system crashes, i.e. interruption
 - Might allow attackers remote code execution (RCE), command injection (CI) or other unauthorized actions
- **DEF:** Backdoors are physical or electronic access points allowing system administrators or attackers to gain access to a system
 - E.g. hard-coded sysadmin password for accessing systems during remote maintenance



Malware

Simple

- Self-replication
 - Virus
 - Worm – zero human interaction!
- (Usually) no self-replication
 - Trojan
 - Backdoor
 - Spyware
 - Logical bomb

Complex

- Botnet
- Ransomware
- Cryptomining malware



Malware – Ransomware

- Ransomware is a form of malware that encrypts a victim's files.
- Ransomware steps:
 1. Phishing emails with malicious attachments or links pointing to malicious web pages.
 2. Admin-level access is gained on the victim's computer.
 3. The ransomware encrypts the user's files, e.g. PDF, DOC, PNG, etc.
 4. The (decryption) key is uploaded to a remote command & control server (C&C) controlled by the attacker.
 5. The attacker then demands a ransom from the victim to restore access to the data upon payment.

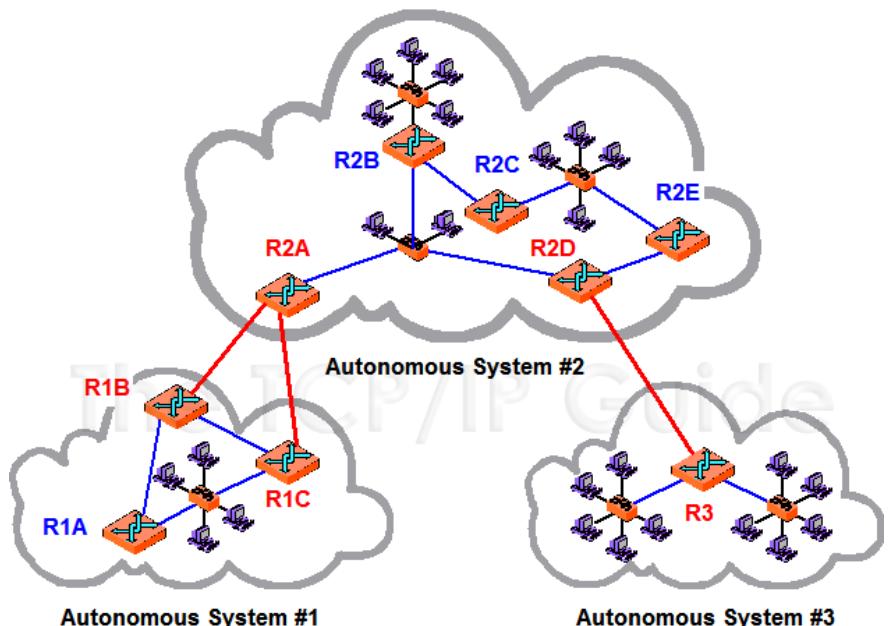


Misused DNS

- DNS is one of the basic protocols underpinning the Internet
 - It resolves domain names by finding Internet Protocol (IP) addresses based on symbolic domain names, e.g. finds the IP address corresponding to www.inf.elte.hu
 - Not strictly monitored by most organizations (!)
- DNS for **data exfiltration**
 - Resolve 2317540194857019487.attackerdomain.com – split & encode data in the least significant part of the domain name
- DNS **tunnelling** in botnets
 - Use of DNS response messages to send commands to bots (who first need to send DNS queries to the attacker's server(s))
- DNS **cache poisoning**
 - Redirect users to malicious websites
- DNS **amplification and reflection** attack
 - DNS request with spoofed IP address sent to (multiple) open DNS resolvers
→ Distributed Denial of Service (DDoS)

BGP hijacking

- **DEF:** An autonomous system (AS) is a collection of resources under the control of a single administrative entity with common, clearly defined routing policy to the internet.
 - Large Internet Service Providers
- **DEF:** Border Gateway Protocol (BGP) is an exterior gateway protocol designed to exchange routing and reachability information among autonomous systems (AS).
 - Type: path-vector routing protocol
→ routers exchange path vectors between source & destination AS.
- **DEF:** BGP hijacking constitutes the takeover of IP address ranges by corrupting BGP routing tables, usually by advertising unauthorized routes.



http://www.tcpipguide.com/free/t_OverviewOfKeyRoutingProtocolConceptsArchitecturesP-2.htm

MONITORING DATA



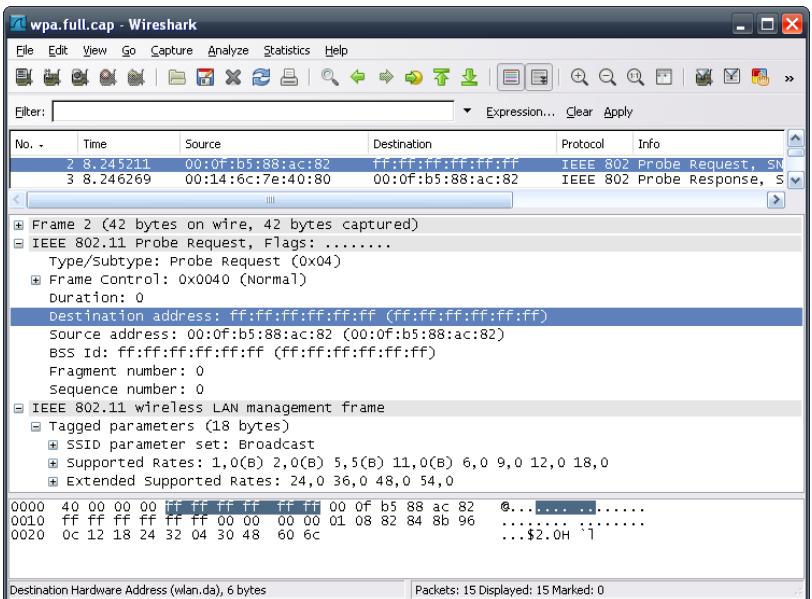
Data types

- **Security data feeds** – published by cybersecurity organizations (e.g. national Computer Emergency Response Team – CERT), or companies, might be industry-specific
- **Transaction data** – any element of an information system records an event which can have security relevance
- **Session data** – capture aggregated data about the communication sessions occurring among entities in an information system
- **(Full) Packet Capture** – capture every piece of data exchanged between entities in an information system



Full packet capture data

- **DEF:** Packet capture is a technique in which data packets are captured and recorded at a specific point in a network.
 - Possible capture locations: routers, switches, servers, workstations
 - Captured packets can be retained and analyzed.
 - In **full packet capture** every data packet is captured and recorded
- Packet capture **challenges:**
 - Storage capacity
 - Realtime analytics capabilities
 - Encryption, i.e. what if the captured data is encrypted?
 - Filtering, i.e. how to filter data?





Session data

- In session data the exchanged data is aggregated for a time period or the entire duration of a communication session between two entities
 - Example: Cisco Netflow falls into this category

Src IP	Dst IP	Appln	Src Port	Dst Port	Protocol	DSGP	TCP FLAGS	Flow Rate	Traffic	Packets	NextHop	FNF NbarApp
192.168.10.192.168.13.1	compressnet	2	169	TCP	AF12	UAP SF	0 Kbps	1.0 KB	2	15.80.39.28	-	
192.168.10.192.168.13.1	compressnet	2	564	TCP	AF12	UAP SF	0 Kbps	1.0 KB	2	190.23.69.213	-	
192.168.10.192.168.13.1	compressnet	2	741	TCP	AF12	UAP SF	0 Kbps	1.0 KB	2	95.55.61.159	-	
192.168.10.192.168.13.1	compressnet	281	2	TCP	AF12	UAP SF	1.0 Kbps	1.0 KB	2	223.191.78.79	-	
192.168.10.192.168.13.1	compressnet	165	3	TCP	AF12	UAP SF	0 Kbps	1.0 KB	2	64.15.70.93	-	
192.168.10.192.168.13.1	compressnet	424	3	TCP	AF12	UAP SF	0 Kbps	1.0 KB	2	149.191.44.148	-	
192.168.10.192.168.13.1	compressnet	822	3	TCP	AF12	UAP SF	0 Kbps	1.0 KB	2	73.7.175.22	-	
192.168.10.192.168.13.1	rje	800	5	TCP	AF12	UAP SF	0 Kbps	1.0 KB	2	74.157.168.220	-	
192.168.10.192.168.13.1	discard	9	714	TCP	AF12	UAP SF	0 Kbps	1.0 KB	2	1.209.56.6	-	
192.168.10.192.168.13.1	daytime	13	252	TCP	AF12	UAP SF	0 Kbps	1.0 KB	2	232.237.195.100	-	
192.168.10.192.168.13.1	daytime	960	13	TCP	AF12	UAP SF	0 Kbps	1.0 KB	2	22.88.95.248	-	
192.168.10.192.168.13.1	msp	18	116	TCP	AF12	UAP SF	1.0 Kbps	1.0 KB	2	81.203.252.131	-	
192.168.10.192.168.13.1	msp	18	735	TCP	AF12	UAP SF	0 Kbps	1.0 KB	2	18.50.17.126	-	
192.168.10.192.168.13.1	ftp-data	20	513	TCP	AF12	UAP SF	0 Kbps	1.0 KB	2	240.7.195.4	-	

Transaction data

- Physical access point (PAP) transaction logs
 - Security gate/door access
 - Physical access to a network router
 - Example: security card swiped to cross into a higher security zone
- Electronic access point (EAP) transaction logs
 - Operating system logs, e.g. system file changes
 - Application logs, e.g. data warehouse transactions, web server/proxy, DNS server, email server, authentication-authorization-accounting (AAA) server
 - Example: a syslog entry generated by an authentication service recording a user logging in via mobile banking app on Android 7 from at 02:17 CET from Malaysia





Security alerts

- **DEF:** Security alert data is generated by the security controls when anomalous, unwanted actions are detected in an information system
- Sources of security alerts at the **physical access points**:
 - Human guards or (regular) employees
 - Physical security controls, e.g. security doors, motion sensors
- Sources of security alerts at the **electronic access points**:
 - Endpoint security solutions: firewall, IDS/IPS, anti-malware
 - Network/system-level security solutions: edge firewall, edge IDS/IPS, security analytics solutions (e.g. packet capture analysis)
- Security data feeds (next slide!) might raise alerts as well



Security data feeds

- Advisories about the latest threats
- Provide valuable advance notification about the latest threats in near real-time
- Can be exchanged on the following levels:
 - Between nation states
 - Between cybersecurity companies
 - Between actors in a specific sector, e.g. electric power system operators
- Examples:
 - AlienVault OTX → provides an API
 - Secjuice, Feed Your SIEM With Free Threat Intelligence Feeds, <https://www.secjuice.com/threat-intelligence-siem-free/>

Summary

- Major cybersecurity incidents
- Definitions:
 - Threat types & sources
 - Challenges
 - Vulnerabilities
- Data types in security analytics
- What is next? → Security analytics, i.e. tools & technologies in cybersecurity



Thank you for your attention!





FINANCIAL MARKET DATA ANALYSIS

*Open Source Technologies for Real-Time Data
Analytics*

Imre Lendák, PhD, Associate Professor

Introduction

- Financial market types
- Technical analysis and indicators
- Market deep(er) dive
 - #1: FOREX
 - #2: Digital currencies





Financial markets

- **DEF:** (Financial) **securities** (instruments/assets) is a generic term to describe stocks, bonds and other instruments representing the right to receive future benefits under a set of stated conditions
- **DEF:** **Financial markets** are the places where (financial) securities are exchanged at efficient market prices
 - Financial markets exist to bring buyers and sellers together
 - Financial markets are domestic (national) or international
 - **Efficient market price** = unbiased price based on the collective speculation of all investors about (future) value, i.e. a price which reflects supply and demand
 - Securities can be exchanged directly between seller and buyer (**over the counter** – OTC), or via different intermediaries (**exchange traded** – ET)
- Real-time market data source examples: Thomson Reuters, Bloomberg, Binance



Types of actors

Lenders-savers

- Companies
- Financial institutions
- Government
- Households / individuals
- Foreigners

Borrower-spenders

- Companies
- Financial institutions
- Government
- Households / individuals
- Foreigners



Types of markets

Traditional markets

- Money market
- Capital market
- Foreign exchange markets, i.e. FOREX
- Commodity market
- Derivatives market
- Insurance markets

Others

- Labor market
- Digital currencies, e.g. Bitcoin, Ethereum



Money market

- **Function:** center for short term borrowing with maturity of 1 year or less
- **Core activity:** interbank lending
- **Instruments:** certificate of deposit, treasury bill, short-term government securities transactions, commercial papers
- **Amounts:** large
- **Players:** financial institutions, e.g. central-, commercial, and savings banks
- **Motivation:** players with surplus funds borrow to others in need of cash, allows governments to raise funds
- **Impact:** determine short-term interest rates
- **Data:** Money Markets Statistical Reporting (MMSR)



Capital markets

- **Function:** market for long- and short-term funds
 - Primary market = first issue, sold to initial buyers
 - Secondary markets = re-sale of instruments issued earlier
- **Core activity:** financing via bond/share issue & trade
- **Instruments:** bonds, stocks
- **Amounts:** all
- **Players:** individuals and institutions in demand and supply of long-term capital, e.g. stock markets, commercial banks, insurance companies, investment funds, companies
- **Motivation:** raise money via issuing bonds/shares, speculate on future prices, dividends
- **Impact:** set market prices of shares (companies) and bonds (governments, companies), secondary markets provide liquidity
- **Data:** Center for Research in Security Prices (CRSP), Thomson Reuters, Bloomberg, S&P



Foreign exchange markets

- **Function:** a **global** decentralized or over-the-counter (OTC) market for the trading of **currencies** (Wikipedia)
 - Note: no central exchange → buyer & seller negotiate directly
- **Core activity:** 24/7 trading in currency pairs in Tokyo, Hong Kong, London, New York
- **Instruments:** currency pairs, e.g. USD/EUR
- **Amounts:** all
- **Players:** large international banks, central banks, currency traders/speculators, governments (e.g. large international contracts)
- **Motivation:** provide liquidity in and between currency pairs, low margins on numerous trades → use of leverage (“multiplier”)
- **Impact:** impacts international trade via setting exchange rates, most liquid financial market of all
- **Data:** Thomson Reuters, Bloomberg, MetaTrader 4 & 5, myFXbook



Commodity market

- **Function:** commodity markets trade in primary resources, i.e. not manufactured products
- **Core activity:** global exchange of primary resources = commodities, e.g. iron/steel, precious stones/metals, minerals, oil
- **Instruments:** physical/derivatives trading, forwards, futures, options, swaps and Exchange-traded commodities (ETC)
- **Amounts:** usually large
- **Players:** producers of primary resources (e.g. large mining corporations)
- **Motivation:** set the efficient market prices for commodities
- **Impact:** sets the prices of raw resources → impacts prices of manufactured goods
- **Data:** International Monetary Fund (IMF) Primary Commodity Prices, Aspect Decision Support Center (DSC)



Derivatives market

- **Function:** market for exchanging derivatives, which are financial instruments derived from other (usually financial) assets/instruments
- **Core activity:** package other financial instruments into derivatives and trade
- **Instruments:** exchange-traded and over-the-counter derivatives, futures contracts, options
- **Amounts:** large
- **Players:** producers (e.g. primary resources), investment funds, insurance companies, individuals (2020 note!)
- **Motivation:** earn by speculating on the future prices or by buying primary resources in advance (e.g. 1-year ahead) and at a discount
- **Impact:** provides liquidity to producers well in advance of their produce becoming available
- **Data:** Optionmetrics, Thomson Reuters, Bloomberg



Insurance markets

- **Function:** measurable risk of loss transferred from one entity to another for a fee, e.g. Casco insurance for a car
- **Core activity:** buy and sell insurance
- **Instruments:** insurance, social insurance (e.g. healthcare)
- **Amounts:** all
- **Players:** insurance companies, private companies, banks (e.g. against credit card fraud), governments (in health)
- **Motivation:** transfer risk to another entity and stay in business even when suffering large losses
- **Impact:** operational (periodic) cost factored into service and product prices, e.g. car insurance is paid as part of a monthly instalment
- **Data:** ?



Digital currency market

- **Function:** avoid expensive intermediaries (i.e. banks) in a truly democratic electronic market
- **Core activity:** anonymous online payments (at least mostly)
- **Instruments:** different cryptocurrencies, e.g. Bitcoin (BTC), Ethereum (ETH), Litecoin (LTC), Peercoin
- **Amounts:** all
- **Players:** all
- **Motivation:** create a democratic, de-centralized, low-cost (?!), short-term digital financial market
- **Impact:** transparent fees, banks lower fees, use in criminal activities (anonymous), more open
- **Data:** Binance, Coin Metrics, Morningstar



Financial intermediaries

Type of intermediary	Sources of funds (Primary liabilities)	Uses of funds (Primary assets)
Depository institutions, i.e. Banks		
Commercial banks	Deposits	Loans, mortgages, government securities, bonds
Credit unions	Deposits	Consumer loans
Contractual savings institutions		
Life insurance companies	Premium from (insurance) policies	Corporate bonds and mortgages
Fire and casualty insurance	Premium from (insurance) policies	Bonds, stock, government securities
Pension funds	Employer and employee transfers	Corporate bonds and stock
Investment intermediaries		
Finance companies	Stock, bonds	Consumer and business loans
Mutual funds	Shares	Stocks, bonds
Money market mutual funds	Shares	Money market instruments

TECHNICAL ANALYSIS



Additional definitions

- **DEF:** **Technical analysis** is methodology for forecasting the direction of financial instrument prices based on the study of historical market data, e.g. price, (trading) volume.
 - Traces of technical analysis in Amsterdam-based merchant's account of the Dutch financial markets in the ~17th century.
 - Candlestick techniques appeared in the ~18th century Asia.
- **DEF:** An **automated trading system (ATS)** is a software-based solution which creates and automatically submits orders to an exchange (or market center).
- **DEF:** A **trading strategy** is a predefined plan (e.g. set of rules) whose goal is to achieve a profitable return by going long or short in financial markets.

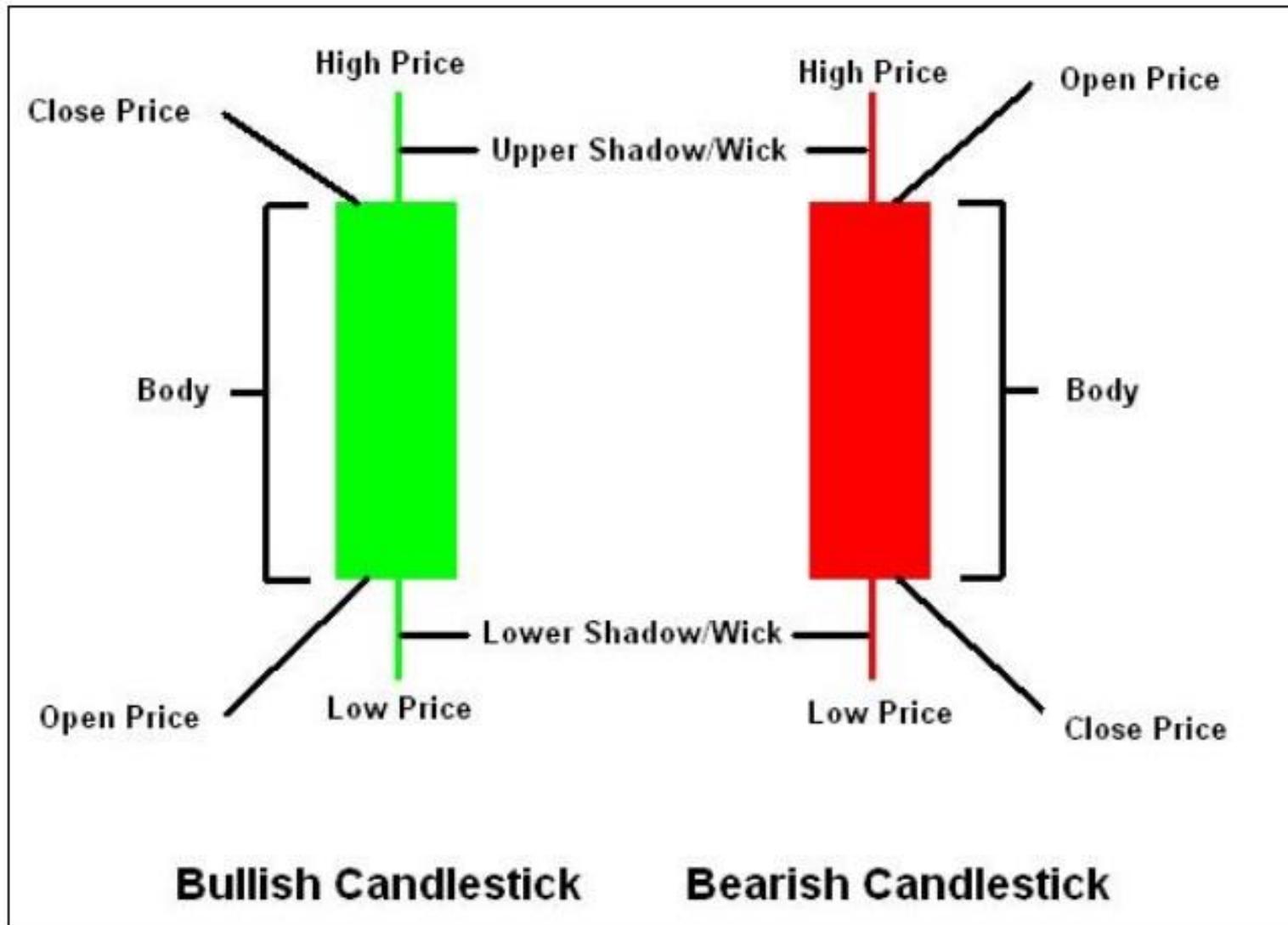
Candlestick charts – 1



<http://www.learntotradethemarket.com/forex-japanese-candlestick-charts>



Candlestick charts – 2



Bullish Candlestick

Bearish Candlestick

<http://www.forexposed.com/2017/09/how-to-read-candlestick-chart-in-forex.html>



Candlestick patterns

(Bearish Trend Pattern)				(Bullish Trend Pattern)			
Bearish III [Continues]	Bearish Harami	Bearish Harami Cross	Dark Cloud Cover [Bar Confirm]	Bullish III [Continues]	Bullish Harami	Bullish Harami Cross	Piercing Line [Bar Confirm]
Engulfing Bearish Line	Evening Doji Star	Evening Star	Gravestone Doji				

<http://forexpops.com/candlestick-patterns/>

Indicators

- Due to the large number and volume of transactions, derived indicators are used alongside raw data
 - Indicators are a secondary measure to actual price movements
- Indicator types by timeline:
 - **Lagging indicators** follow price movements and act as a confirmation tool.
 - **Leading indicators** precede price movements and try to predict the future.
- Indicator type by intent:
 - Trend
 - Volume
 - Momentum
 - Volatility





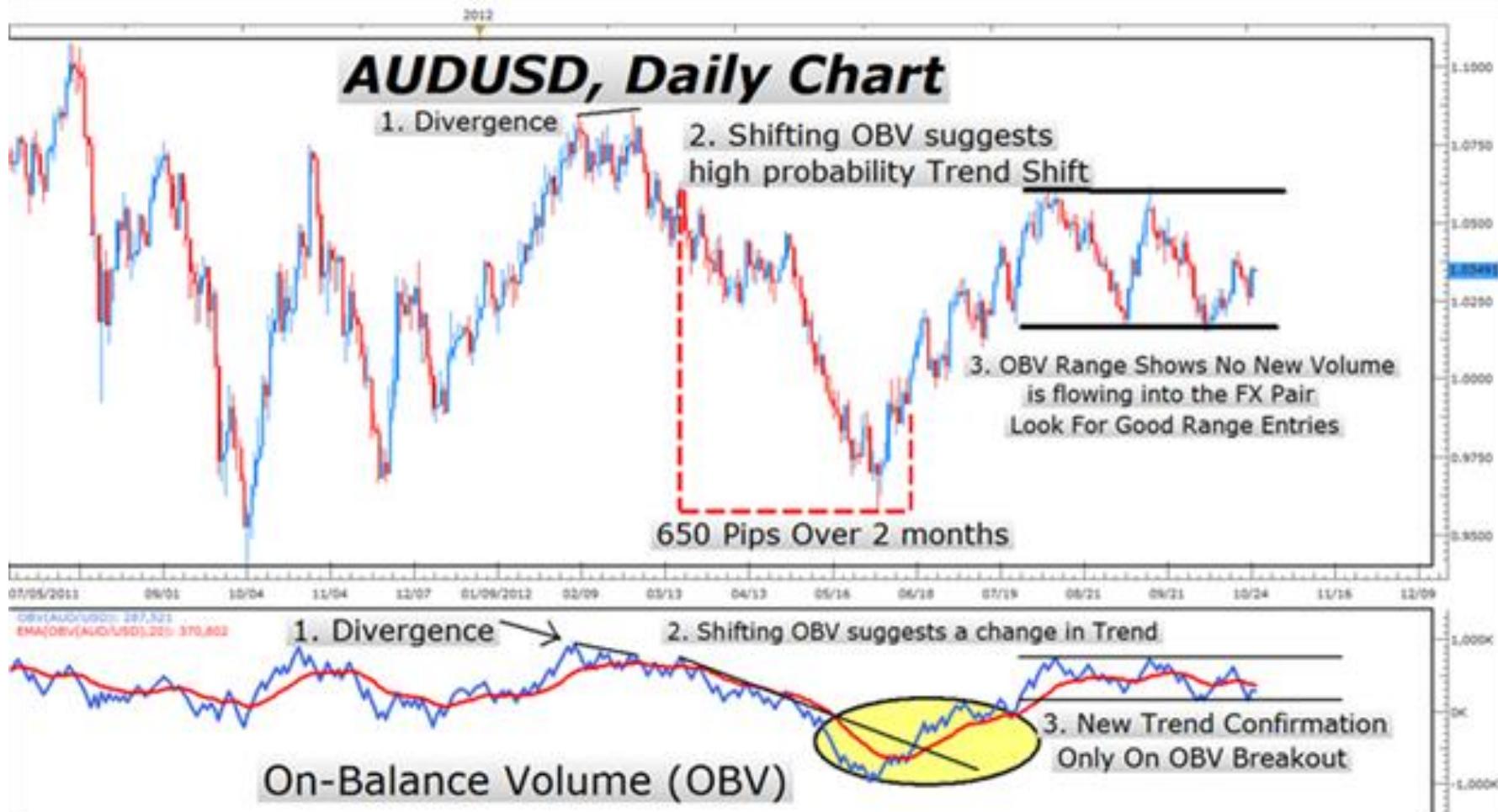
Trend: Simple Moving Average (SMA)



<https://www.babypips.com/learn/forex/using-moving-averages>



Volume: On Balance Volume (OBV)



<https://www.babypips.com/learn/forex/using-moving-averages>

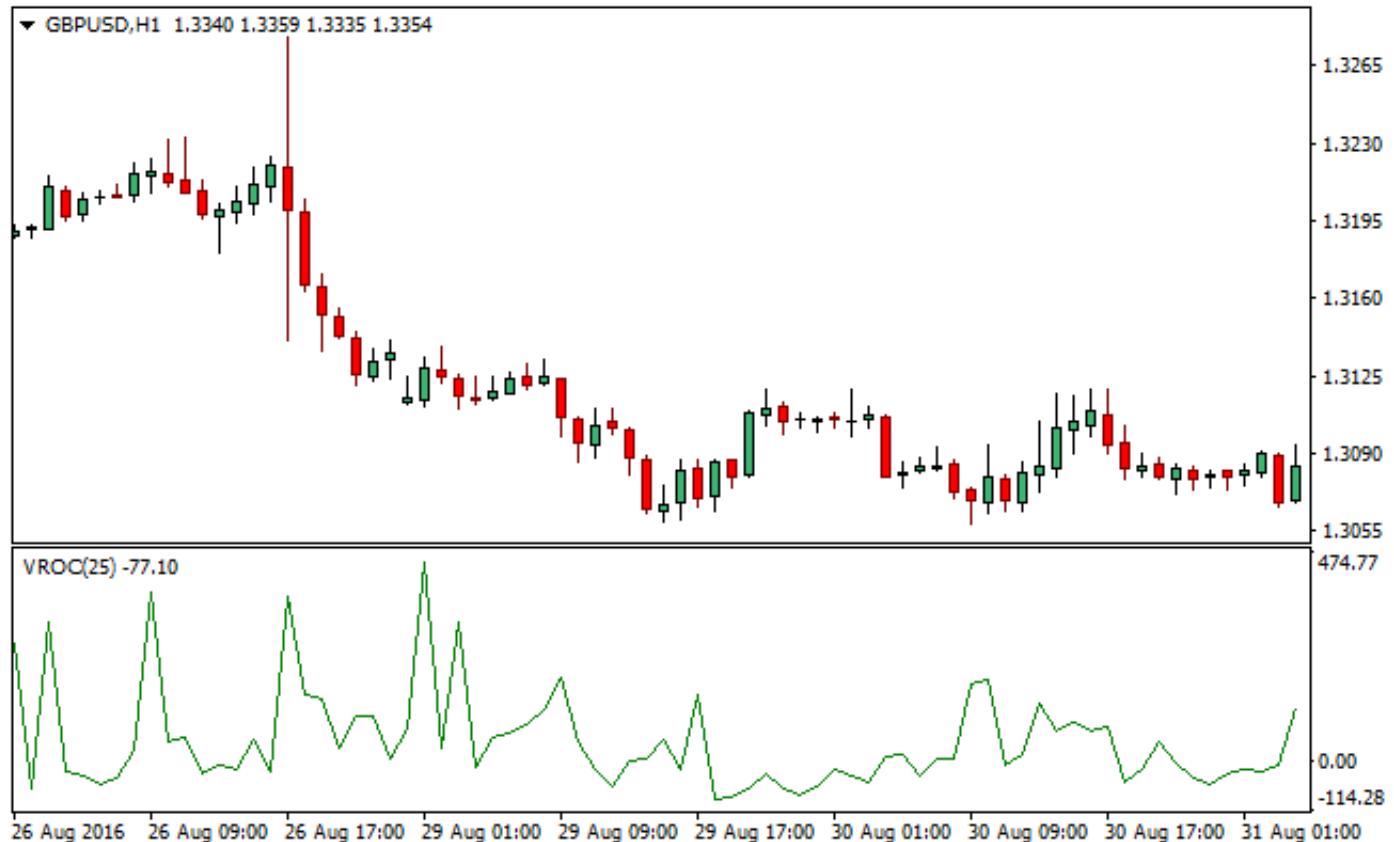
Volume: Accumulation/Distribution (Acc/Dist)



$$CLV = \frac{(C - L) - (H - C)}{H - L}$$

$$ACCDIST_i = ACCDIST_{i-1} + CLV \times Volume$$

Momentum: Rate of Change (ROC)



$$ROC = \frac{Closing_i - Closing_{i-N}}{Closing_{i-N}} \times 100$$

<https://www.dolphintrader.com/volume-rate-change-vroc-forex-indicator-mt4/>

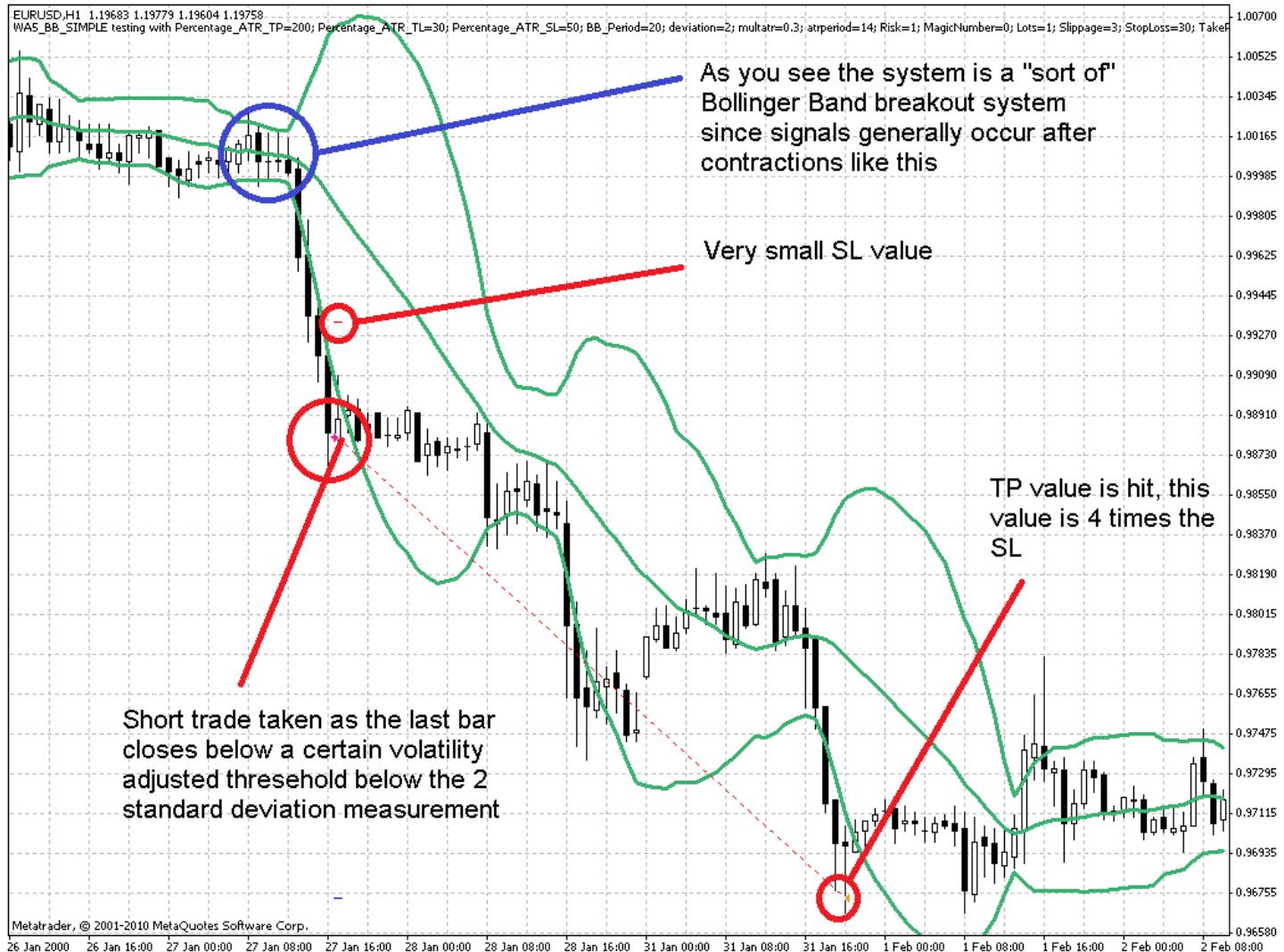
Momentum: Relative Strength Index (RSI)



$$RSI = 100 - \frac{100}{1 + RS}, \text{ where } RS_{\text{period}} = \frac{\text{AvgGain}}{\text{AvgLoss}}$$

<https://www.investoo.com/indicators-relative-strength-index-rsi/>

Volatility: Bollinger band



<https://mechanicalforex.com/2010/06/playing-with-bollinger-bands-likely.html>

FOREX



FOREX intro

- The foreign exchange market (Forex, FX, or currency market) is a **global** decentralized or over-the-counter (OTC) market for the trading of **currencies** (Wikipedia)
 - Currencies are traded in pairs, e.g. EURUSD (or EUR/USD)
 - The **largest** trading market in **trading volume**
 - The largest participants in the FX markets are big banks, e.g. Citi, JP Morgan, UBS, Deutsche Bank, etc.
- Top 5 trading locations by popularity (data from 2017)
 - London, United Kingdom (41%),
 - New York, United States (19%),
 - Singapore (6%),
 - Tokyo, Japan (6%) and
 - Hong Kong, China (4%).



Round-the-clock trading

EDT	GMT	Trading Sessions, Summer time		EST	GMT	Trading sessions, Winter time	
8:00	12:00		LONDON		12:00		
9:00	13:00				13:00	LONDON	
10:00	14:00				14:00		
11:00	15:00				15:00		
12:00	16:00				16:00		
13:00	17:00				17:00		
14:00	18:00				18:00		
15:00	19:00				19:00		
16:00	20:00				20:00		
17:00	21:00				21:00		
18:00	22:00				22:00		
19:00	23:00				23:00		
20:00	0:00				0:00		
21:00	1:00				1:00		
22:00	2:00				2:00		
23:00	3:00				3:00		
0:00	4:00				4:00		
1:00	5:00				5:00		
2:00	6:00				6:00		
3:00	7:00	LONDON			7:00		
4:00	8:00				8:00		
5:00	9:00				9:00		
6:00	10:00				10:00		
7:00	11:00				11:00		

<http://www.learnfxtrade.com/trading-sessions/>



Raw data example

Sign In

Mon, Jul 23, 12:03:39

You Are Here : Home / Forex Market / Currencies / EURUSD / Historical Data

Share

EURUSD Historical Data

EURUSD - Euro vs US Dollar 1.16994 -0.22% -25.4 pips

Timeframe: 1 Day Start: 03/28/2018 21:00 End: 07/22/2018 21:00 Filter

Please note data goes back approximately 1000 data points per each timeframe.

Date	Open	High	Low	Close	Change (Pips)	Change (%)
Jul 22, 2018 21:00	1.17248	1.17509	1.16865	1.16997	-25.1	-0.21%
Jul 21, 2018 21:00	1.1725	1.17456	1.17225	1.17404	+15.4	+0.13%
Jul 19, 2018 21:00	1.16406	1.17389	1.16263	1.17209	+80.3	+0.69%
Jul 18, 2018 21:00	1.1638	1.16785	1.1575	1.16422	+4.2	+0.04%
Jul 17, 2018 21:00	1.16594	1.1665	1.1602	1.16397	-19.7	-0.17%
Jul 16, 2018 21:00	1.17101	1.17447	1.16493	1.16611	-49.0	-0.42%
Jul 15, 2018 21:00	1.16897	1.17253	1.16756	1.1711	+21.3	+0.18%
Jul 14, 2018 21:00	1.16898	1.16912	1.16767	1.16786	-11.2	-0.10%
Jul 12, 2018 21:00	1.167	1.16868	1.16131	1.16861	+16.1	+0.14%
Jul 11, 2018 21:00	1.16716	1.16957	1.16497	1.16702	-1.4	-0.01%
Jul 10, 2018 21:00	1.17284	1.17572	1.16651	1.16748	-53.6	-0.46%
Jul 09, 2018 21:00	1.17494	1.17629	1.16902	1.1728	-21.4	-0.18%
Jul 08, 2018 21:00	1.17388	1.17908	1.17327	1.175	+11.2	+0.10%
Jul 07, 2018 21:00	1.17384	1.17544	1.1736	1.17465	+8.1	+0.07%
Jul 05, 2018 21:00	1.16895	1.17679	1.16801	1.17417	+52.2	+0.44%
Jul 04, 2018 21:00	1.16554	1.17202	1.165	1.1692	+36.6	+0.31%
Jul 03, 2018 21:00	1.16547	1.1682	1.16308	1.16566	+1.9	+0.02%
Jul 02, 2018 21:00	1.16386	1.16731	1.16204	1.16586	+20.0	+0.17%
Jul 01, 2018 21:00	1.16657	1.16882	1.15911	1.16386	-27.1	-0.23%
Jun 30, 2018 21:00	1.16657	1.16884	1.16616	1.16778	+12.1	+0.10%
Jun 28, 2018 21:00	1.15669	1.16903	1.1558	1.16826	+115.7	+0.99%
Jun 27, 2018 21:00	1.15517	1.16008	1.15272	1.15686	+16.9	+0.15%
Jun 26, 2018 21:00	1.1546	1.16724	1.15407	1.15545	-91.5	-0.70%

M3 Money Supply (YoY) (1h 56min) US: Existing home sales likely to surge ... (1 min ago) EURUSD 1.16998 GBPUSD 1.31243 USDJPY 111.108 USDCAD 1.31487 XM



Input data

- Select lookback (time) period (e.g. 1 year's data for building a model) and currency pair(s) (e.g. EURUSD)
- Raw data:
 - Historical price data for the chosen currency pair(s), e.g. <https://www.myfxbook.com/en/forex-market/currencies/EURUSD-historical-data>
- Calculated data
 - One or more trend indicators, e.g. 20 SMA
 - One or more volume indicators, e.g. ACC/DIST
 - One or more momentum indicators, e.g. Relative Strength Index (RSI)
 - One or more volatility indicators, e.g. Bollinger bands
- Turn it into a series of historical data for training purposes:
 - (currency pair, data & time, 20_sma, roc, rsi, bb_lo, bb_hi)



Problems in ‘fortune-telling’

- Forecasts are noisy and with relatively low confidence
- Confidence is always a challenge
 - How to calculate confidence level for a 1-minute ahead forecast on a 1-5 scale?
- Holding period
 - How long to keep open a long or short position to obtain the optimal yield (i.e. highest gain with lowest risk)?
- Volume to invest
 - Which % of investment funds to invest into a certain position?
 - Note: volume is very important because of the leverage mechanism (!)
- Hedging strategy?



References

- Yves Hilpisch, “Python for Finance”, O'Reilly Media, 2014.
- Philip J. Romero, Tucker Balch, “What hedge funds really do – An introduction to portfolio management”, Business Expert Press, 2014.
- “Machine learning for Trading”, (free) Udacity course
<https://www.udacity.com/course/machine-learning-for-trading--ud501>.
- Investopedia, <https://www.investopedia.com>
- MyFXBook, <https://www.myfxbook.com>
 - They organize contests (!)
- Mechanical FOREX, <https://mechanicalforex.com>



DIGITAL CURRENCIES



Digital currency

- **DEF:** A digital currency is available (only) in digital form
 - Also known as: digital money, electronic money/currency
 - Stored in an electronic wallet associated with a physical person or legal entity
- **Examples:** cryptocurrencies (e.g. Bitcoin), in-game currencies in online gaming
- Advantages:
 - Instantaneous transactions
 - Borderless transfer of ownership
 - Anonymity
 - Potential for lower transaction fees
- Disadvantages:
 - Hard to regulate, i.e. governments might struggle to monitor and/or control digital currencies
 - Might be used by criminals



Bitcoin definitions

- **DEF:** Bitcoin is cryptographic software-based online payment system, i.e. cryptocurrency
 - Described by Satoshi Nakamoto in 2008.
 - Introduced as open-source software in 2009.
 - Abbreviation: BTC.
- Payments are recorded in a public ledger using its own unit of account (Bitcoin).
- It is a form of digital currency (physical form is absent), created and held electronically.
- It can be used to buy things electronically.
- Bitcoin can be divided into smaller unit called Satoshi.
 - 1 Satoshi = one hundred millionth of 1 BTC.



Bitcoin history

- 2009: Bitcoin announced by Satoshi Nakamoto
 - Pseudonym for person or group of person
- 2009-2011: slow start...
- 2011-2013: Silk Road and Dread Pirate Roberts
- End 2013: Bitcoin price skyrockets
- and the world notices!

Bitcoin characteristics

- It is decentralized
- It is easy to set up and it is fast
- It is anonymous
- It is completely transparent
- Transaction fees are minuscule, i.e. rather small
- Transactions are irreversible





Decentralized

- Base for the Bitcoin protocol is a peer-to-peer system which means that there is no need for a third party.
- Therefore, in theory, bitcoin network is not controlled by central authority (fully decentralized monetary system).
- Bitcoins are being created by a community of people that anyone can join.
- In theory, there is no authority (financial institution) which can tinker with monetary policy and in that sense devalue or revalue Bitcoin currency.



Anonymous & Transparent

- Bitcoins are stored in wallet with digital credentials for your bitcoin holdings and allows you to access them.
- Wallet uses public-key cryptography, in which two keys, one public and one private are generated. Public key can be thought of as an account number or name and the private key, ownership credentials.
- Bitcoin is transferred to the next owner when the next owner gives a public key and previous owner uses his private key to publish a record into system announcing that the ownership has changed to the new public key.
- Bitcoin protocol stores details of every single transaction that occurred in the network in huge version of general ledger (Block chain).



Small fees & Irreversible

- Bitcoin doesn't charge fees for either national or international transfers.
- Bitcoin is not the first private money, not the first digital currency, and not the first currency based on cryptography, but it has been the first to rely on a peer to peer network decentralization to avoid double spending.
- Bitcoin protects against double spending by verifying each transaction added to the blockchain to ensure that the inputs for the transaction had not previously already been spent.



Mining = BTC creation

- Miners use special software to solve math problems (Bitcoin algorithm), and upon completing the task they receive certain amount of coins.
- They are created each time a user discovers a new block i.e. finds hash value with required characteristics.
- Software is creating new units until it reaches amount of 21 million units → Bitcoin is a currency with finite supply.
- The rate of block creation is approximately constant over time (6 per hour) with 50 % reduction every four years.
- Halving (in theory) continues until 2110-2140 when 21 million BTC will have been issued.





Earn BTCs

- Mining
- Accept BTC as payment
- Earn BTC via trading
- Earn Bitcoins as regular income
- Earn Bitcoin from interest payments
- Other: donations, gambling, getting tipped, completing tasks on websites...



<https://blog.internshala.com/2018/10/how-to-earn-money-online-top-10-ways-for-making-money-online-in-india/>

Summary

- Financial market types
- Technical analysis and indicators in real-time financial data analysis
- Market focus #1: FOREX
- Market focus #2: Digital currencies



Thank you for your attention!





DATA STORAGE #1: MONGODB, LOGSTASH & CASSANDRA

*Open Source Technologies for Real-Time Data
Analytics*

Imre Lendák, PhD, Associate Professor



Key storage requirements

- Ability to handle extremely large amounts of data → distributed storage
- Ability to handle both structured and unstructured data
- Open source access → vibrant community driven by business needs developing additional tools (a'la Hadoop, as opposed to MapReduce which was internal to Google)
 - Commercial use
 - Distribution
 - Modification
 - Patent use
 - The open source 'consumer' might not want to open source the new code



Data storage timeline

Database world

- PostgreSQL (**from – to**)
- MySQL (**from – to**)
- SQLite (**from – to**)

'Big data' world

- MapReduce (**from – to**)
- Hadoop (**from – to**)
- MongoDB (**from – to**)
- Cassandra (**from – to**)
- CouchDB (**from – to**)
- Neo4j (**from – to**)
- Logstash (**from – to**)



Data storage timeline

Database world

- PostgreSQL (1996 – now)
- MySQL (1995 – now)
- SQLite (2000 – now)

'Big data' world

- MapReduce (2003-2014)
- Hadoop (2004 – now)
- MongoDB (2009 – now)
- Cassandra (2008 – now)
- CouchDB (2005 – now)
- Neo4j (2007 – now)
- Logstash (2010 – now)

RDBMS timeline

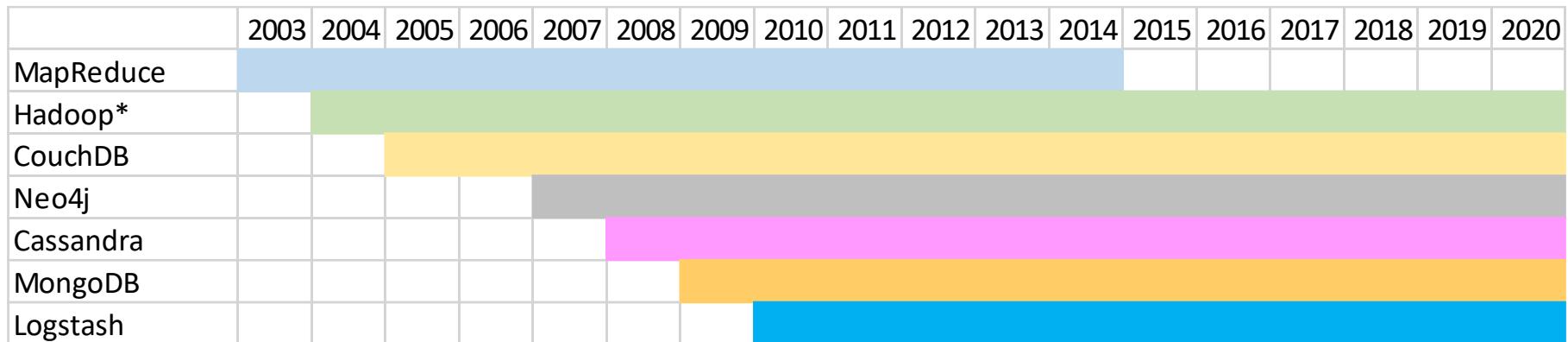


A horizontal timeline chart from 1995 to 2020. The x-axis is labeled with years from 1995 to 2020. The y-axis lists database systems: MySQL, PostgreSQL, and SQLite. MySQL is shown in yellow, PostgreSQL in green, and SQLite in blue. MySQL starts in 1995. PostgreSQL starts in 1996. SQLite starts in 1998.

	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	...	2016	2017	2018	2019	2020
MySQL	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	...	2016	2017	2018	2019	2020
PostgreSQL		1996										2016	2017	2018	2019	2020
SQLite						1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008



'Big data' storage timeline



* Hadoop lives as a distributed data storage platform, not as data processor

Chosen technologies

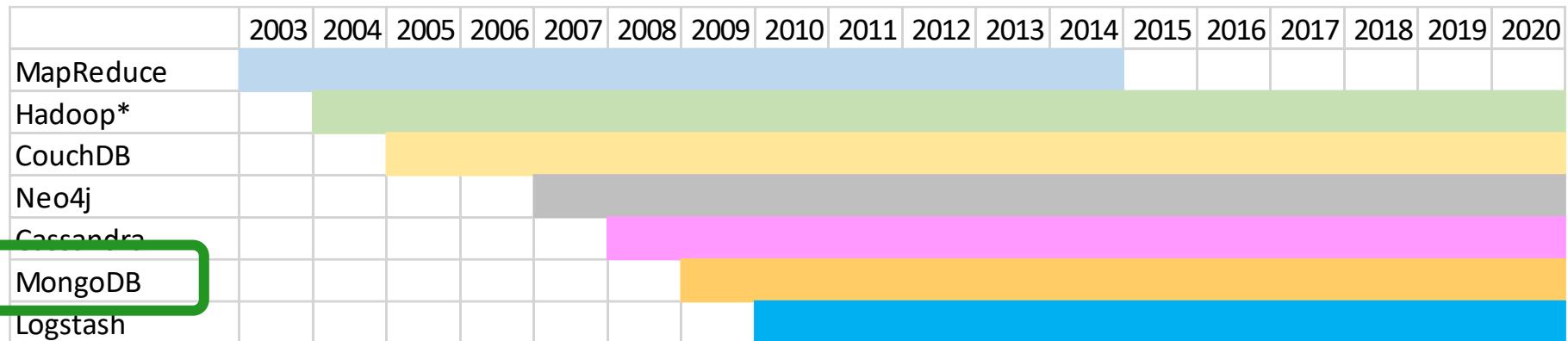
- **MongoDB** is a general purpose, document-based, distributed database
- **Logstash** is the log management element of the ELK stack
- **Cassandra** is a decentralized structured storage system



MONGO DB



MongoDB in the timeline



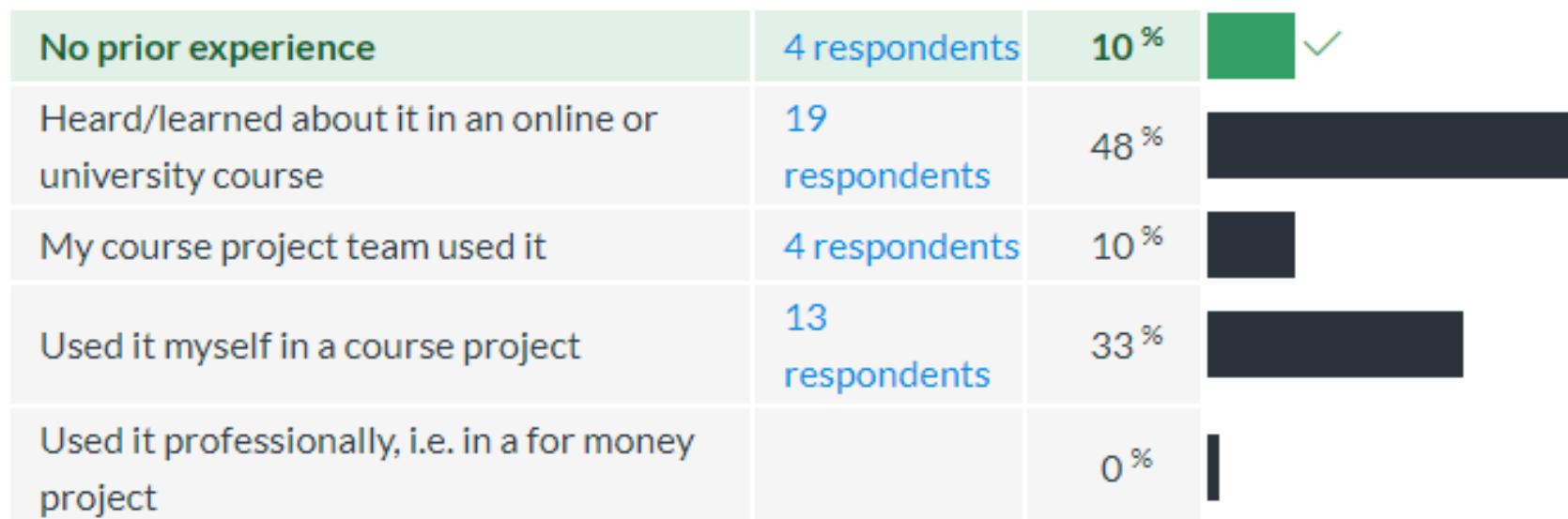
* Hadoop lives as a distributed data storage platform, not as data processor



Why MongoDB @ OST?

Attempts: 40 out of 40

Please rate your past experience in using the MongoDB in data storage:





History

- **DEF:** MongoDB is a general purpose, document-based, distributed database
- **mongoDB = “Humongous DB”**
 - Open-source
 - Document-based
 - “High performance, high availability”
 - Automatic scaling
 - C-P on CAP

[-blog.mongodb.org/post/475279604/on-distributed-consistency-part-1](http://blog.mongodb.org/post/475279604/on-distributed-consistency-part-1)
[-mongodb.org/manual](http://mongodb.org/manual)



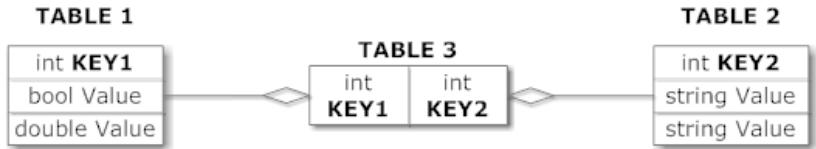
Other NoSQL Types

Key/value (Dynamo)

Columnar/tabular
(HBase)

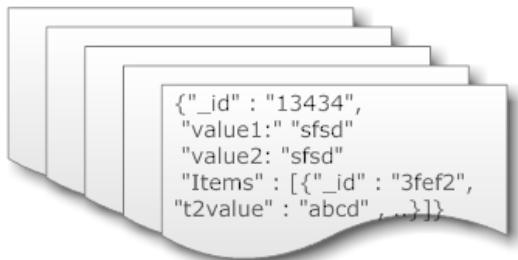
Document (mongoDB)

Relational Model



Document Model

Collection ("Things")





Motivations

Problems with SQL

- Rigid schema
- Not easily scalable (designed for 90's technology or worse)
- Requires unintuitive joins

Perks of mongoDB

- Easy interface with common languages (Java, Javascript, PHP, etc.)
- DB tech should run anywhere (VM's, cloud, etc.)
- Keeps essential features of RDBMS's while learning from key-value noSQL systems



Data Model

- Document-Based (max 16 MB)
- Documents are in BSON format, consisting of field-value pairs
- Each document stored in a collection
- Collections
 - Have index set in common
 - Like tables of relational DB's.
 - Documents do not have to have uniform structure



What is BSON?

JSON

- “JavaScript Object Notation”
- Easy for humans to write/read, easy for computers to parse/generate
- Objects can be nested
- Built on
 - name/value pairs
 - Ordered list of values
- E.g. AlienVault threat intelligence feed
- <http://json.org/>

BSON

- “Binary JSON”
- Binary-encoded serialization of JSON-like docs
- Also allows “referencing”
- Embedded structure reduces need for joins
- Goals
 - Lightweight
 - Traversable
 - Efficient (decoding and encoding)
- <http://bsonspec.org/>



Deserialized BSON example

```
{  
  "_id" : "XXXXXX"  
  "city" : "Budapest",  
  "pop" : 1660,  
  "state" : "HUN",  
  "professor" : {  
    name: "Péter Péterffy"  
    address: "Pázmány Péter sétány 1/a"  
  }  
}
```



BSON Types

Type	Number
Double	1
String	2
Object	3
Array	4
Binary data	5
Object id	7
Boolean	8
Date	9
Null	10
Regular Expression	11
JavaScript	13
Symbol	14
JavaScript (with scope)	15
32-bit integer	16
Timestamp	17
64-bit integer	18
Min key	255
Max key	127

The number can
be used with the
\$type operator to
query by type!



The `_id` Field

- By default, each document contains an `_id` field
- `_id` serves as primary key for collection.
- Its value is unique, immutable, and may be any non-array type.
- The default data type is `ObjectId`, which is “small, likely unique, fast to generate, and ordered.” Sorting on an `ObjectId` value is roughly equivalent to sorting on creation time.

<http://docs.mongodb.org/manual/reference/bson-types/>



mongoDB vs. SQL

	mongoDB	SQL
Data ‘atom’	Document	Tuple
Data group	Collection	Table/View
Identifier	PK: <code>_id</code> Field	PK: Any attribute(s)
Schema	Uniformity not Required	Uniform Relation Schema
Efficiency	Index	Index
Linking	Embedded Structure	Joins
Distribution	Shard	Partition



Who uses Mongo DB?

Used by millions of developers to power the world's most innovative products and services

facebook

invision

ebay

Adobe

Google

 **SQUARESPACE**

coinbase

SEGA®

intuit

 **eharmony**

EA

verizon

shutterfly

 **GOV.UK**

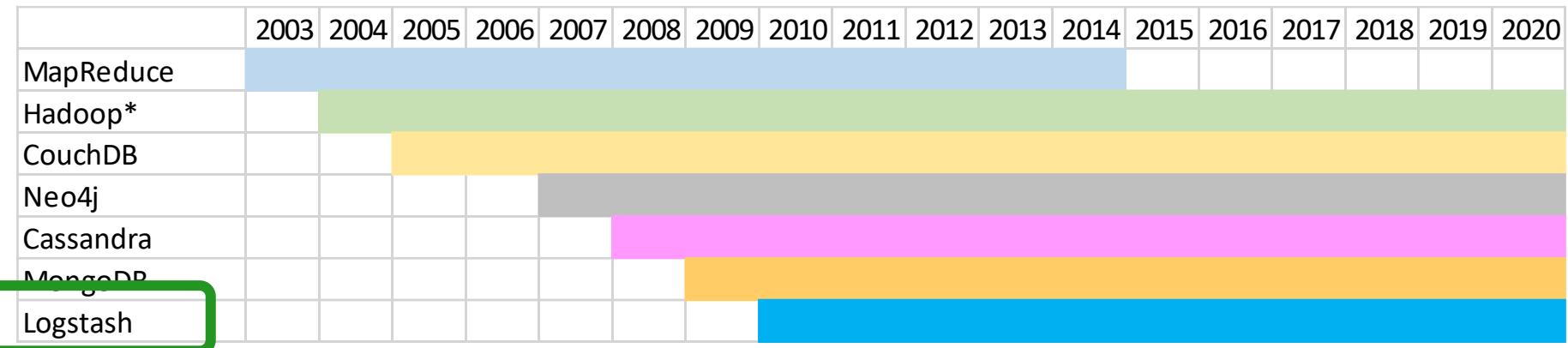
SAP®

[View customer stories](#)

LOGSTASH



Logstash in the timeline



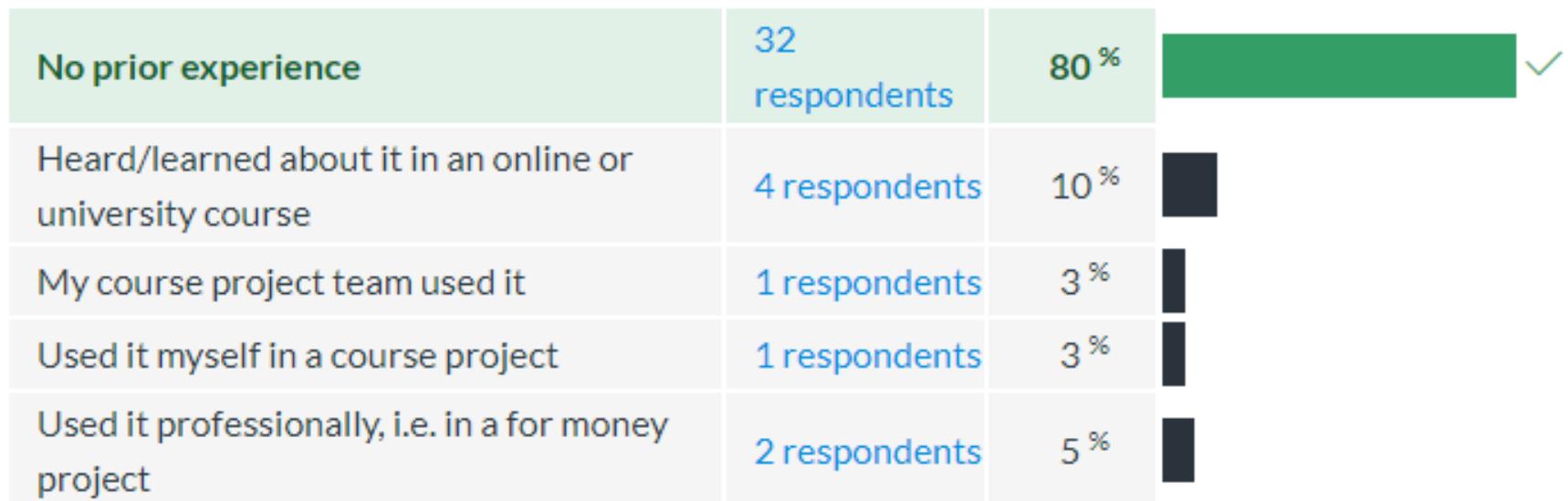
* Hadoop lives as a distributed data storage platform, not as data processor



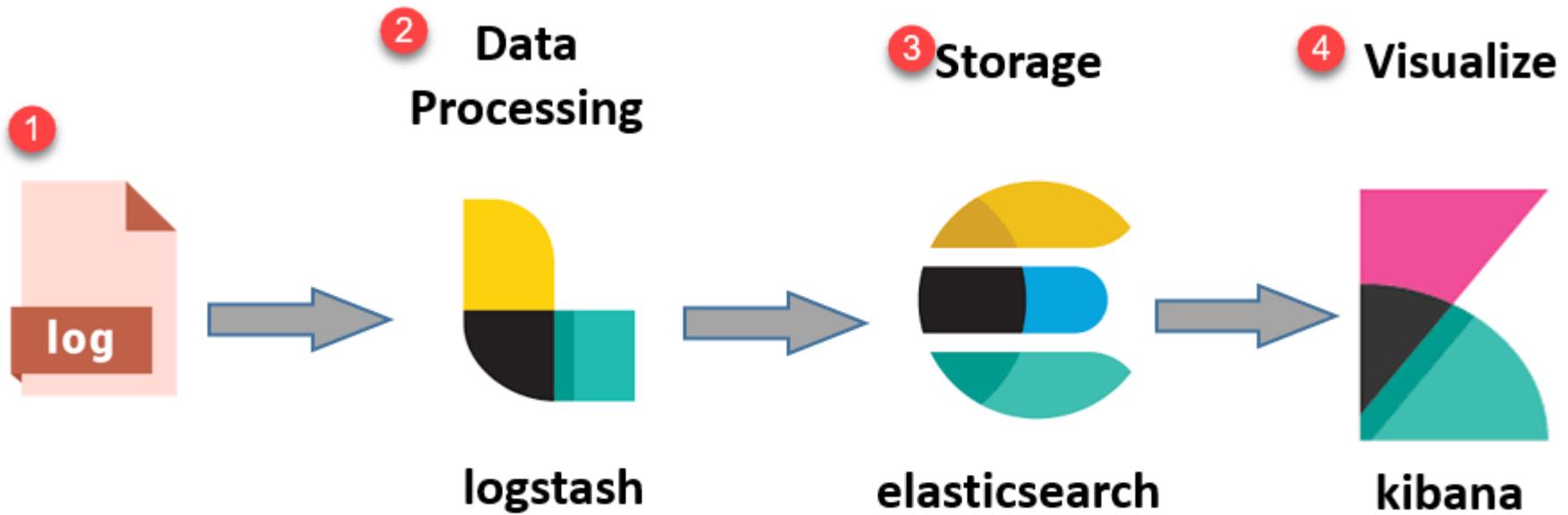
Why Logstash @ OST?

Attempts: 40 out of 40

Please rate your past experience in using Elasticsearch:



ELK stack



© guru99.com

<https://www.guru99.com/elk-stack-tutorial.html>



Logstash

Components & Features

- Is a **data collection** pipeline
- Parse and **normalize** different kinds of logs into machine-based analysis ready format
- Advanced input filtering
- One instance can handle multiple pipelines of related events

- Instance = single LS process
- Event = the primary data unit in Logstash
- Pipeline = separate data flows
- Queue = input data received

Strong points

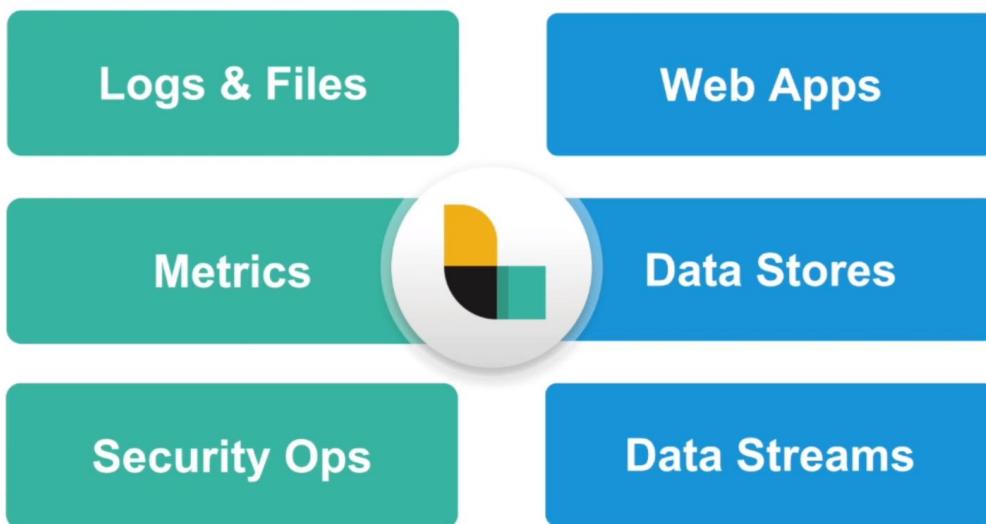
- Central data processing, like syslog servers earlier
- Accepts a wide variety of structured data
- Accepts unstructured data
- Plugins for various input sources and platforms
- Written in Java → cross platform
- In-memory and on-disk (durable) queues



Logstash use cases

Logstash Data Sources

Ingest All the Things

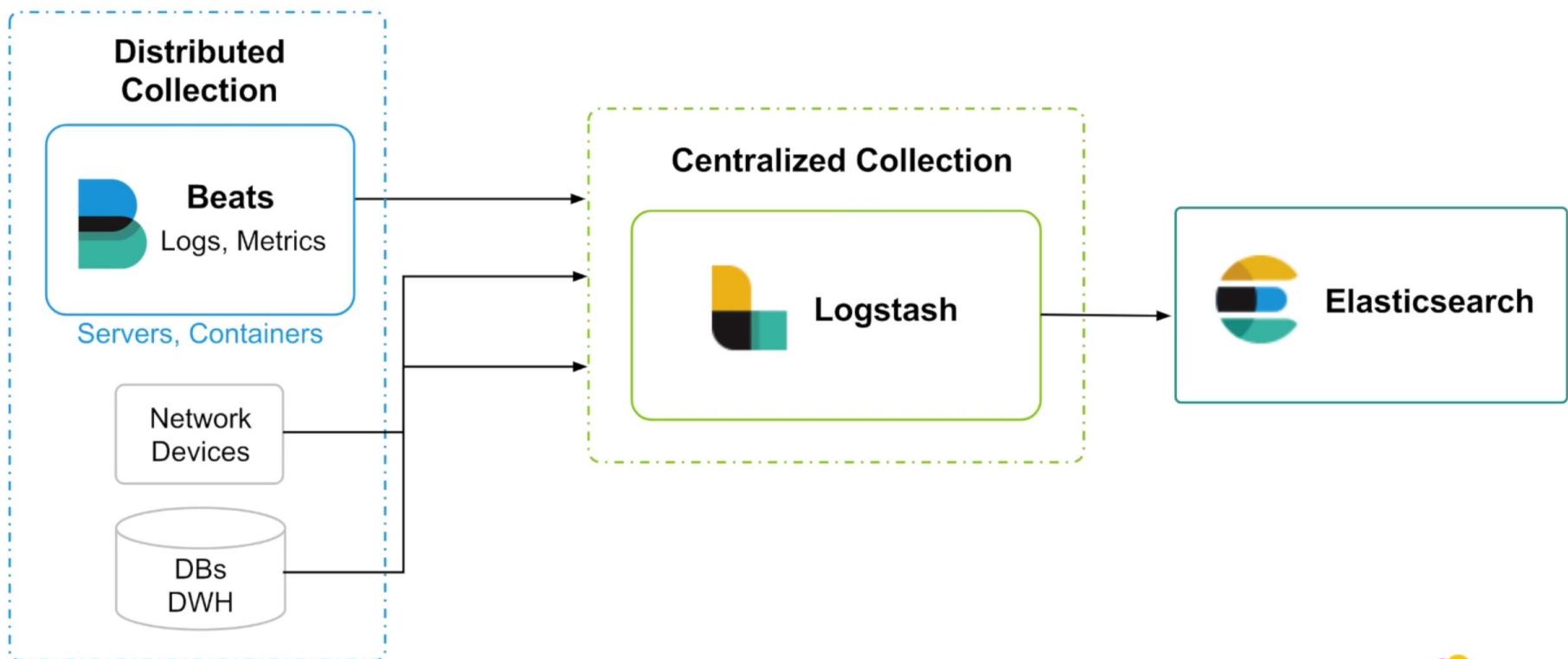


Logstash data sources



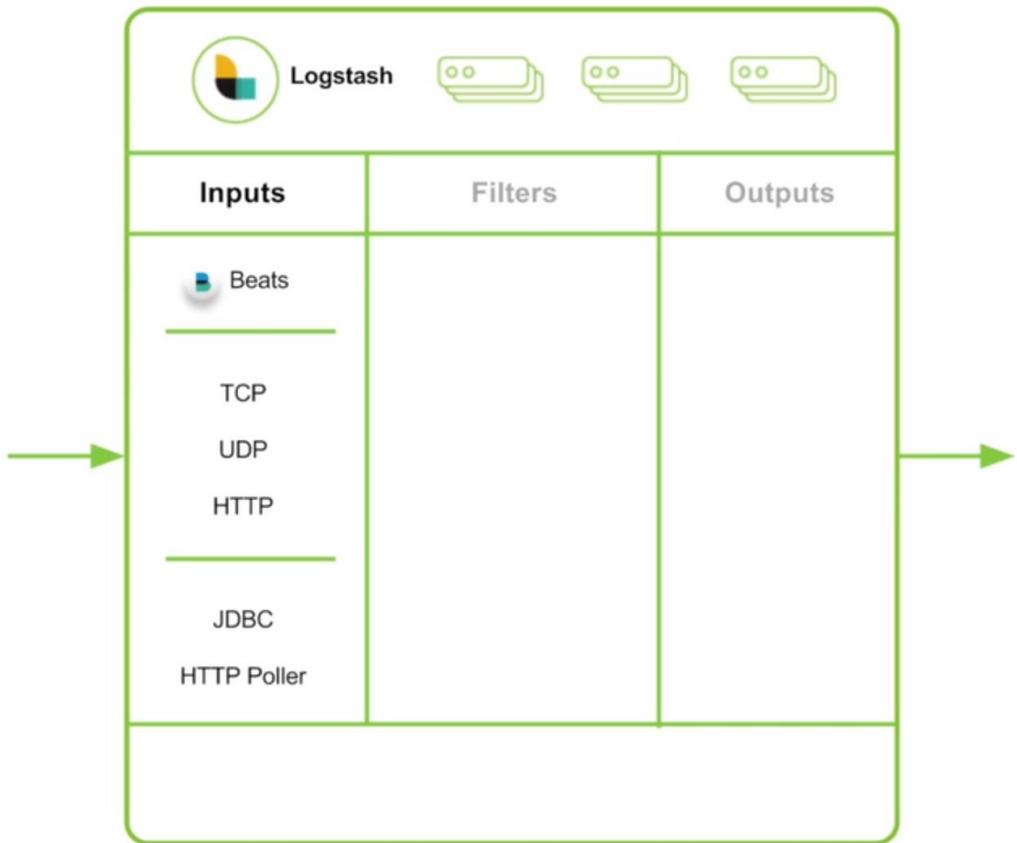
Logstash Data Sources

Ingest All the Things



Logstash Data Collection

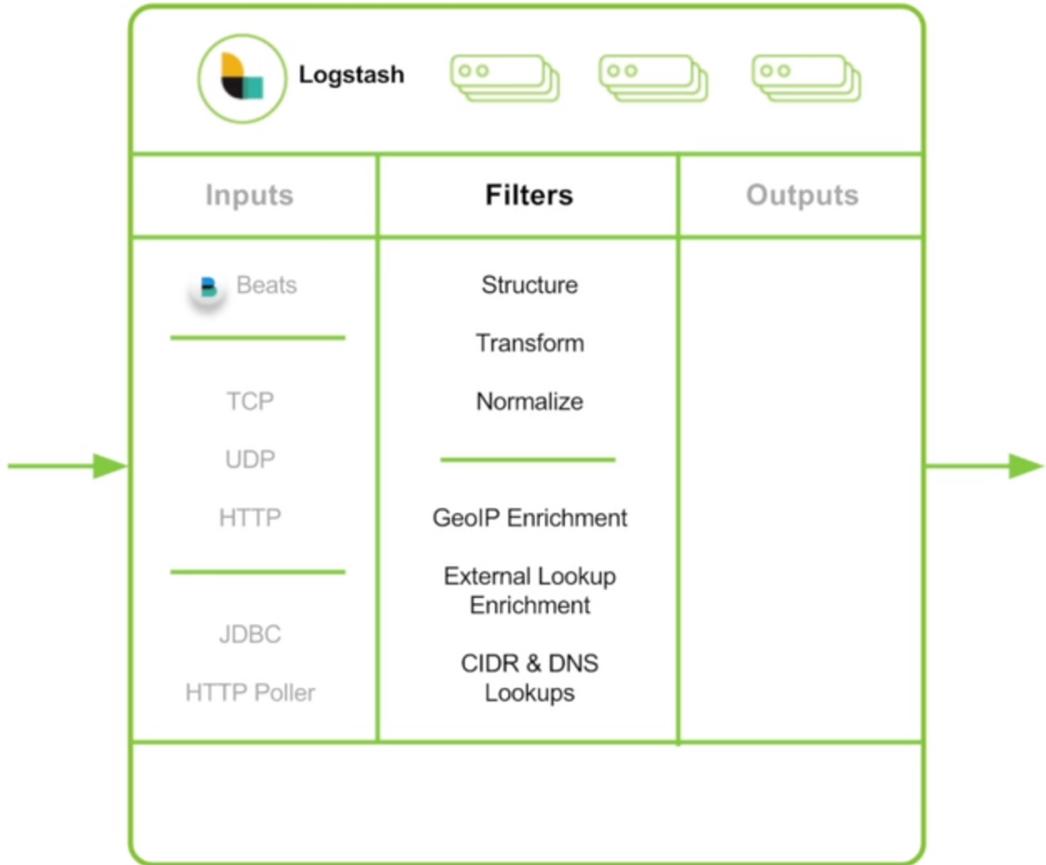
- Collect and deserialize data with input plugins
- Codecs may be used for deserialization



<https://www.elastic.co/webinars/getting-started-logstash>

Logstash Data Processing

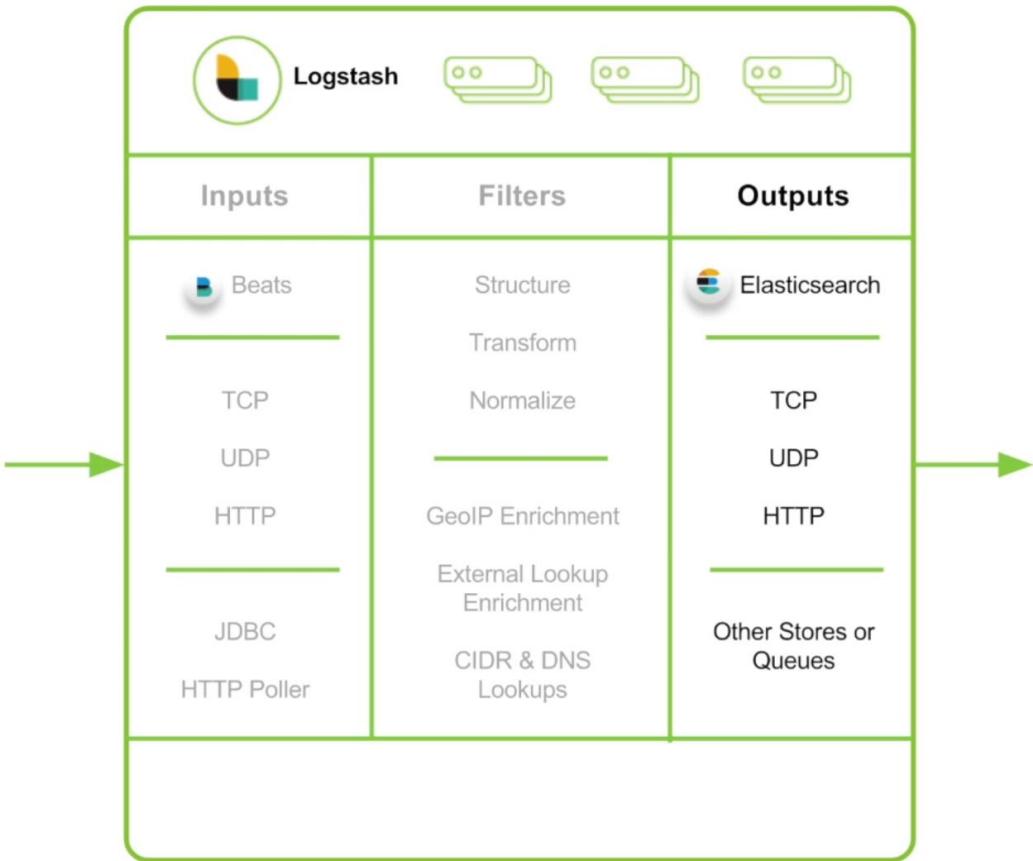
- Filter plugins
- Structure,
- Transform, and
- Enrich data



<https://www.elastic.co/webinars/getting-started-logstash>

Logstash Data Processing

- Emit (i.e. output) data to ElasticSearch or other destinations
- Use of output plugins



<https://www.elastic.co/webinars/getting-started-logstash>



Logstash filtering

- Grok filter – for parsing fields
- Date filter
- Dissect filter – like Grok, but minimal
- KV filter – key-value pair



Filter #1: Grok filter

- Field parsing filter

```
filter {  
    grok {  
        match => {"message" => "%{TIMESTAMP_8601:ts}%{SPACE}%{GREEDYDATA:message}"}  
    }  
}
```

<https://www.elastic.co/webinars/getting-started-logstash>



Filter #2: Date filter

- User strings for setting @timestamp based on

```
filter {  
  
    date {  
  
        match => ["timestamp_string", "ISO8601"]  
  
    }  
  
}
```

<https://www.elastic.co/webinars/getting-started-logstash>



Enriching data in Logstash

- GeoIP filter: Enrich IP address information with geographical context
- DNS filter: Enrich hostname with DNS info
- User Agent filter: Enrich user agent (i.e. browser info)
- Translate filter: Translate numerical codes and/or foreign languages



Enrich #1: GeolP filter

- Enrich IP address information

```
filter {
    geoip {
        fields => "my_geoip_field"
    }
}
```



Enrich #2: DNS filter

- Enrich hostname information with DNS info

```
filter {
    dns {
        fields => "my_dns_field"
    }
}
```



Enrich #3: User Agent filter

- Enrich browser user agent information

```
filter {
    useragent {
        source => "useragent"
    }
}
```



Enrich #4: Translate filter

- Use local data to map / enrich event descriptions

```
filter {  
    translate {  
        dictionary => [ "100", "Continue",  
                        "101", "Switching Protocols",  
                        "merci", "thank you",  
                        "old version", "new version" ]  
    }  
}
```



Logstash data manipulation

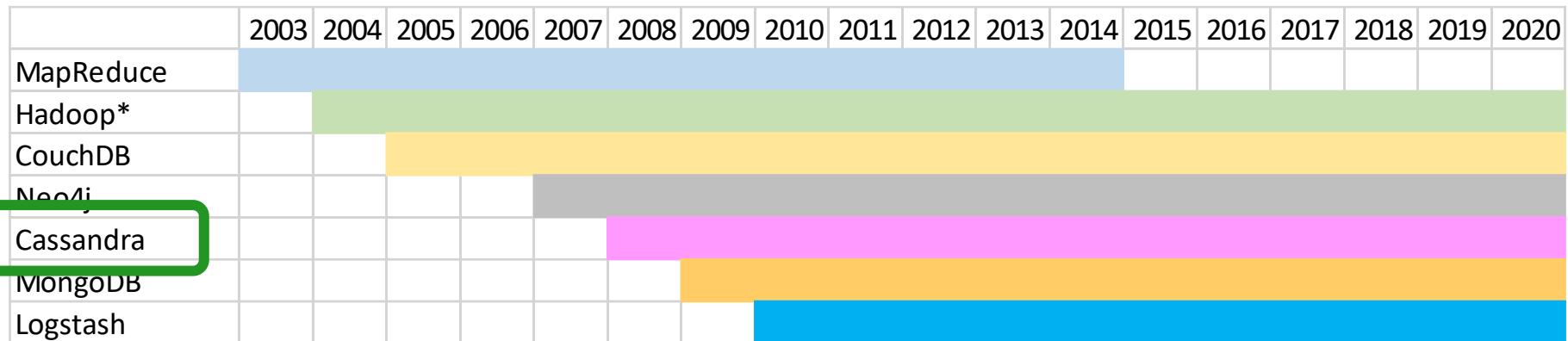
```
filter {  
    mutate { lowercase => "account" }  
  
    if [type] == "patch" {  
        split { field => actions target => action }  
    }  
  
    if { "action" =~ /special/ } {  
        drop {}  
    }  
}
```

- Convert field types
e.g. string → int
- Add, rename, replace,
remove or copy fields
- (Upper/lower) Case
transformations
- Join arrays
- Split fields → arrays
- Strip whitespace

CASSANDRA



Cassandra in the timeline



* Hadoop lives as a distributed data storage platform, not as data processor



Why Cassandra @ OST?

Attempts: 40 out of 40

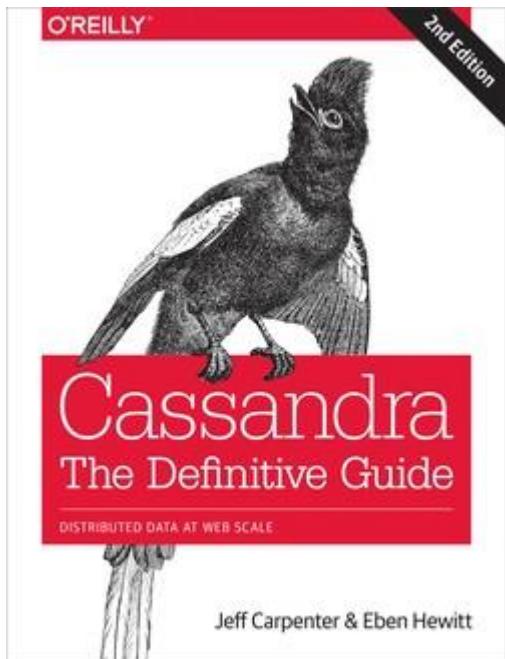
Please rate your past experience in using Cassandra for data storage:



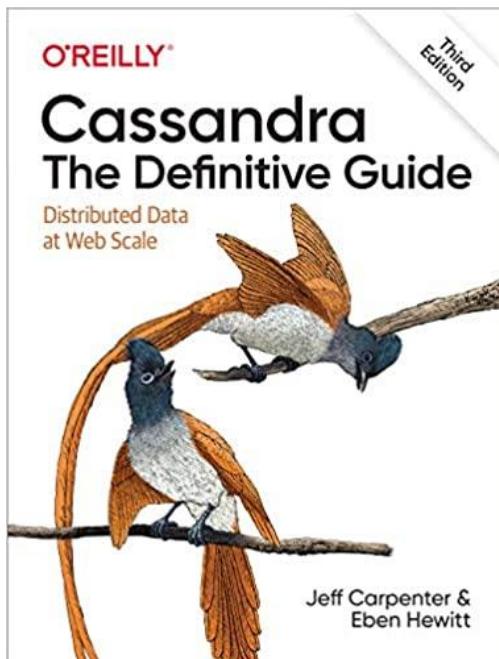


Key source(s)

O'Reilly: Cassandra Ed 2



O'Reilly: Cassandra Ed 3





Introduction

- “Apache Cassandra is an open-source, distributed, decentralized, elastically scalable, highly available, fault-tolerant, tunably consistent, column-oriented database that bases its distribution design on Amazon’s dynamo and its data model on Google’s Big Table.”
 - Clearly, it is buzz-word compliant!!
- DEF: A Decentralized Structured Storage System
- Key features
 - Partitioning
 - Replication
 - Cluster management



Key RDBMS challenges

1. **Scalability issues** → what if the database (content) becomes so large that it can not be stored on the owner's own hardware & what if there are so many users that an RDBMS can not answer queries with reasonable cost in time & resources
2. **Joins are slow** → each RDBMS, even the smaller ones rely on joins which are slow → users start queueing up if the load is heavy
3. **Transactions are synchronous** → this in essence means that transactions modify multiple atomic pieces of data which are blocked during the transaction → not available to (other) users.



Typical solutions

- ‘Throw **more hardware** at the problem’ by adding more memory, faster CPUs or disks
 - More CPUs and disks cause scalability challenges → replication & consistency issues (might) arise
- **Custom optimizations**, e.g. SSD drives, faster communication channels, turn off unnecessary RDBMS features (journaling?)
- **Restructure the data model** → modify the model to better support high throughput at the cost of lower clarity and higher complexity
 - This often results in de-normalization, i.e. creating a (data) sub-optimal model with duplicates
- **Application optimizations** → attention put on the apps we build, improve the indices (i.e. plural of index), modify the app source code
- **Introduce a caching layer** → in-memory data with yet more added complexity, e.g. Redis
- **NOTE:** All the above might not be sufficient in intensive operations

Data Model



keyspace

column family

settings

settings

column

name

value

timestamp

* Figure from Eben Hewitt's (author of Oreilly's Cassandra book) slides.



Partitioning

- Nodes are logically structured in Ring Topology.
- Hashed value of key associated with data partition is used to assign it to a node in the ring.
- Hashing rounds off after certain value to support ring structure.

- Lightly loaded nodes moves position to alleviate highly loaded nodes.



Replication

- Each data item is replicated at N (replication factor) nodes.
- Different Replication Policies
 - Rack Unaware – replicate data at N-1 successive nodes after its coordinator
 - Rack Aware – uses ‘Zookeeper’ to choose a leader which tells nodes the range they are replicas for
 - Datacenter Aware – similar to Rack Aware but leader is chosen at Datacenter level instead of Rack level.



Cluster Management (CM)

Gossip protocol

- DEF: Gossip protocols are network communication protocols inspired by real life rumor spreading
- Periodic, pairwise, inter-node communication.
- Low frequency communication ensures low cost.
- Random selection of peers.

Cassandra CM

- Uses Scuttleback (a Gossip protocol) to manage nodes.
- Uses gossip for node membership and to transmit system control state.

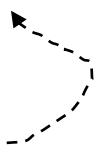


Basic Idea: Key-Value Store

Table T:

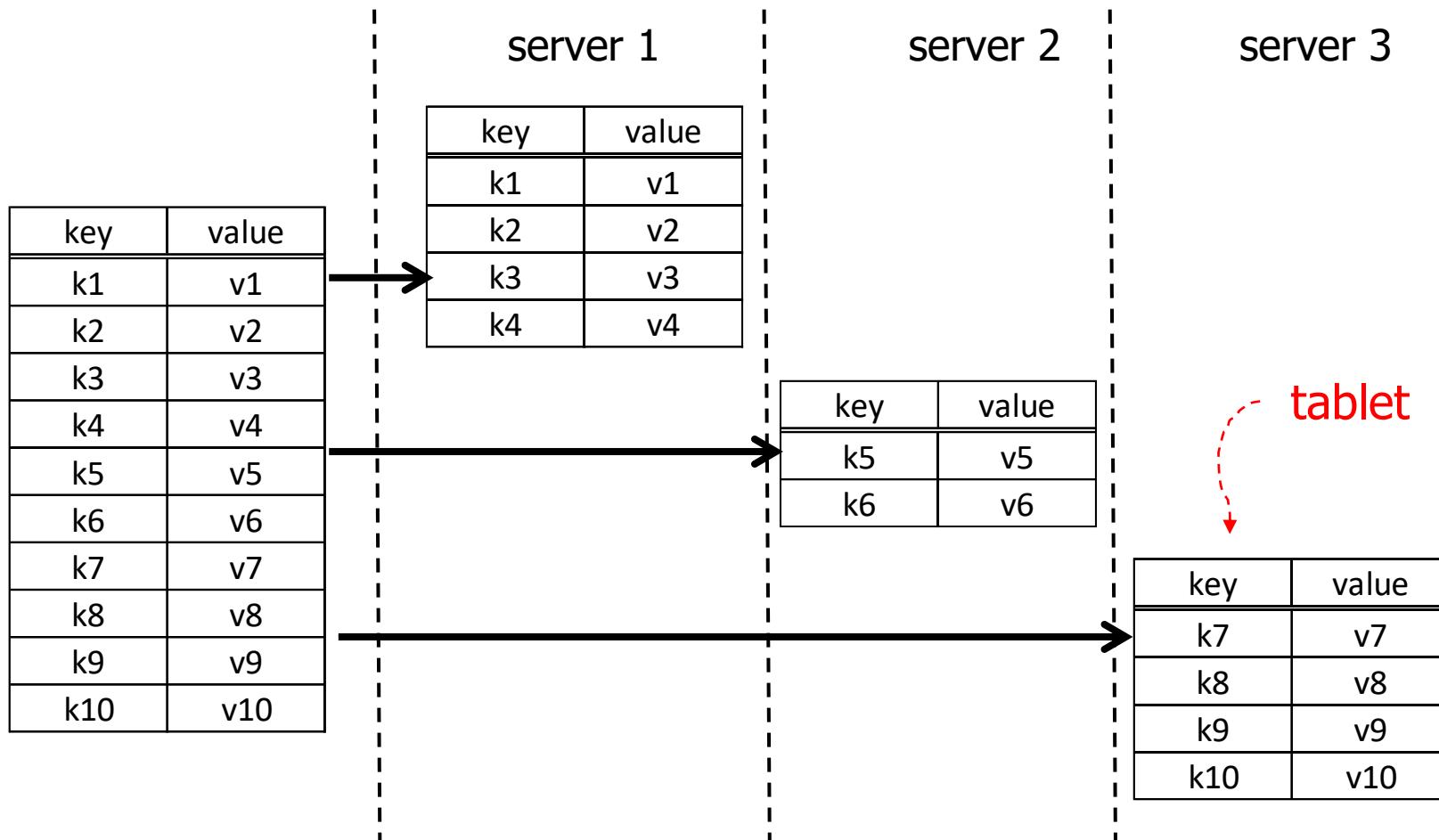
key	value
k1	v1
k2	v2
k3	v3
k4	v4

keys are sorted



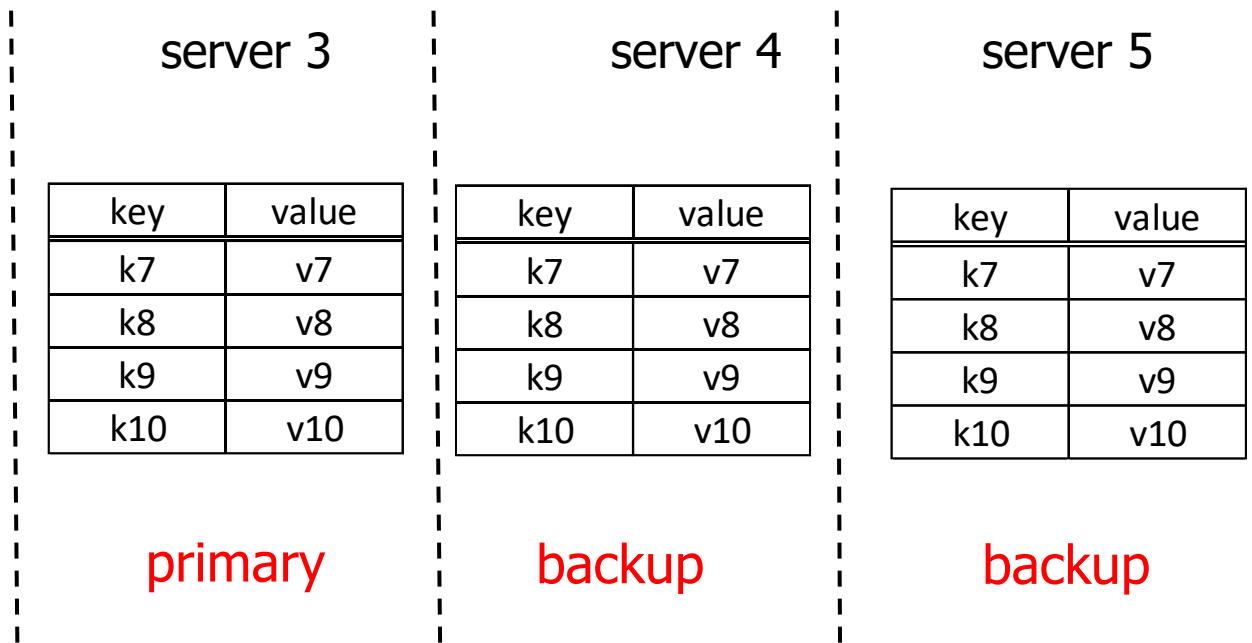
- API:
 - `lookup(key) → value`
 - `lookup(key range) → values`
 - `getNext → value`
 - `insert(key, value)`
 - `delete(key)`
- Each row is timestamp-ed
- Single row actions atomic
(but not persistent in some systems?)
- No multi-key transactions
- No query language!

Fragmentation (Sharding)



- use a partition vector
- “auto-sharding”: vector selected automatically

Tablet Replication



- Cassandra:
 - Replication Factor (# copies)
 - R/W Rule: One, Quorum, All
 - Policy (e.g., Rack Unaware, Rack Aware, ...)
 - Read all copies (return fastest reply, do repairs if necessary)



Tablet Internals

key	value
k3	v3
k8	v8
k9	delete
k15	v15

memory

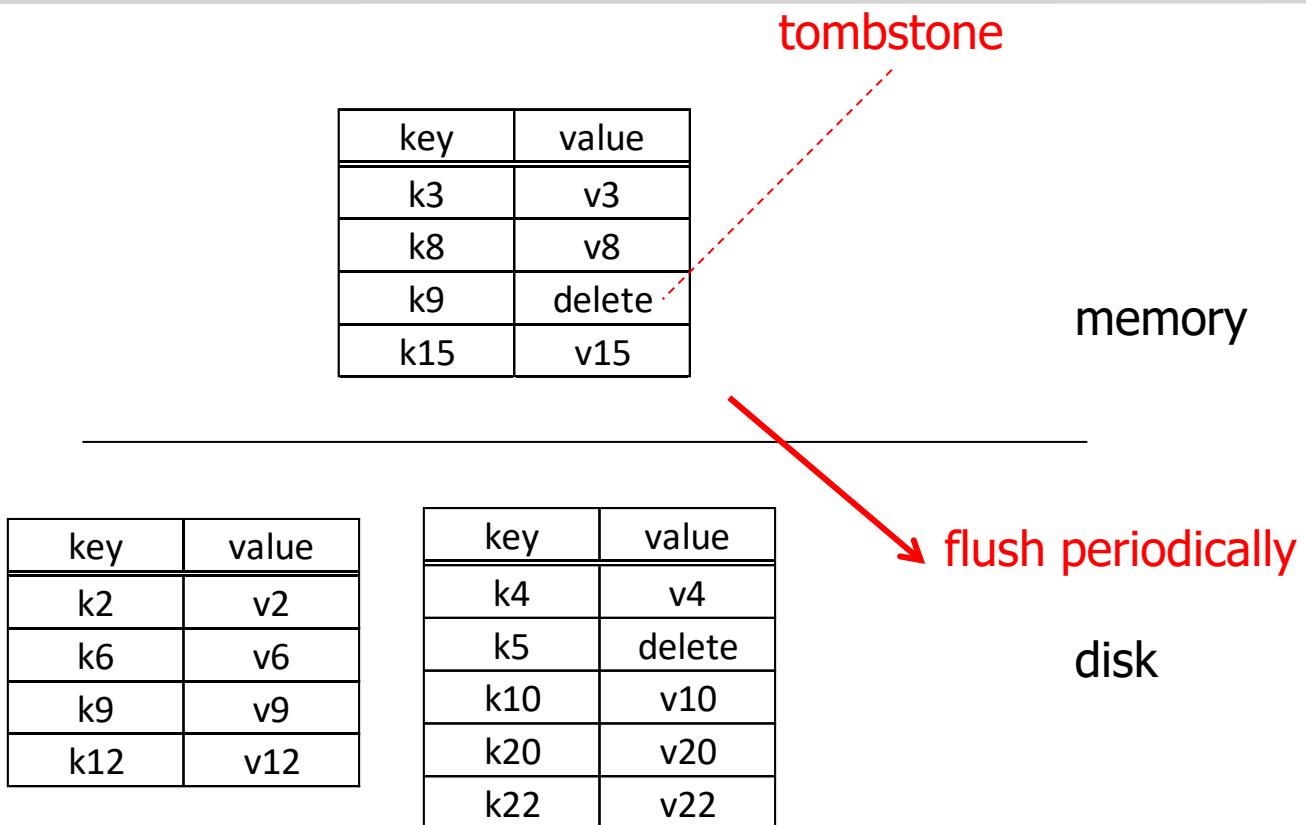
key	value
k2	v2
k6	v6
k9	v9
k12	v12

key	value
k4	v4
k5	delete
k10	v10
k20	v20
k22	v22

disk

Design Philosophy (?): Primary scenario is where all data is in memory.
Disk storage added as an afterthought

Tablet Internals



- tablet is merge of all segments (files)
- disk segments immutable
- writes efficient; reads only efficient when all data in memory
- periodically reorganize into single segment



Column Family

K	A	B	C	D	E
k1	a1	b1	c1	d1	e1
k2	a2	null	c2	d2	e2
k3	null	null	null	d3	e3
k4	a4	b4	c4	e4	e4
k5	a5	b5	null	null	null



Column Family

K	A	B	C	D	E
k1	a1	b1	c1	d1	e1
k2	a2	null	c2	d2	e2
k3	null	null	null	d3	e3
k4	a4	b4	c4	e4	e4
k5	a5	b5	null	null	null

- for storage, treat each row as a single “super value”
- API provides access to sub-values
(use family:qualifier to refer to sub-values
e.g., price:euros, price:dollars)
- Cassandra allows “super-column”:
two level nesting of columns
(e.g., Column A can have sub-columns X & Y)



Vertical Partitions

K	A	B	C	D	E
k1	a1	b1	c1	d1	e1
k2	a2	null	c2	d2	e2
k3	null	null	null	d3	e3
k4	a4	b4	c4	e4	e4
k5	a5	b5	null	null	null



can be manually implemented as

K	A
k1	a1
k2	a2
k4	a4
k5	a5

K	B
k1	b1
k4	b4
k5	b5

K	C
k1	c1
k2	c2
k4	c4

K	D	E
k1	d1	e1
k2	d2	e2
k3	d3	e3
k4	e4	e4



Vertical Partitions

K	A	B	C	D	E
k1	a1	b1	c1	d1	e1
k2	a2	null	c2	d2	e2
k3	null	null	null	d3	e3
k4	a4	b4	c4	e4	e4
k5	a5	b5	null	null	null



column family

K	A
k1	a1
k2	a2
k4	a4
k5	a5

K	B
k1	b1
k4	b4
k5	b5

K	C
k1	c1
k2	c2
k4	c4

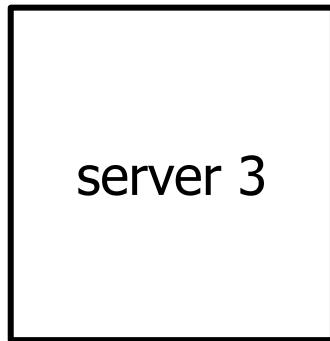
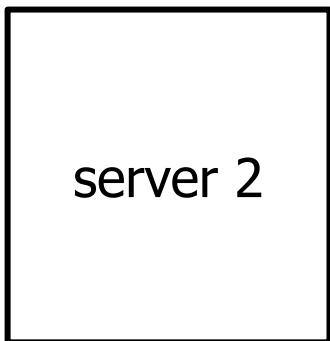
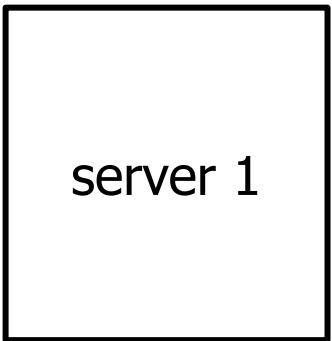
K	D	E
k1	d1	e1
k2	d2	e2
k3	d3	e3
k4	e4	e4

- good for sparse data;
- good for column scans
- not so good for tuple reads
- are atomic updates to row still supported?
- API supports actions on full table; mapped to actions on column tables
- API supports column “project”
- To decide on vertical partition, need to know access patterns



Failure recovery (Cassandra)

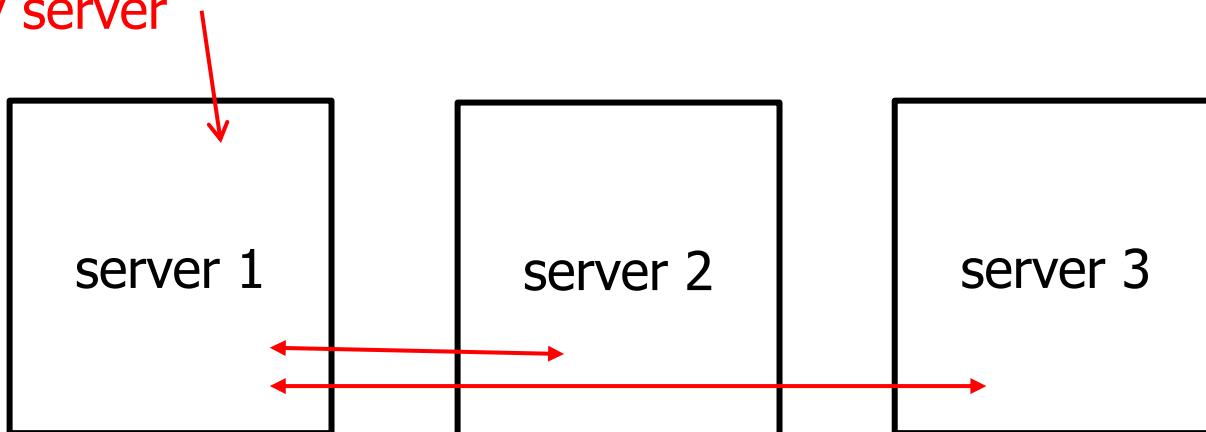
- No master node, all nodes in “cluster” equal



Failure recovery (Cassandra)

- No master node, all nodes in “cluster” equal

access any table in cluster
at any server



that server sends requests
to other servers



Cassandra vs MySQL

MySQL

- > 50 GB Data
- Writes Average : ~300 ms
- Reads Average : ~350 ms

Cassandra

- > 50 GB Data
 - Writes Average : 0.12 ms
 - Reads Average : 15 ms
-
- Stats calculated on Facebook data



Who uses Cassandra?

- Originally designed at Facebook
- Open-sourced
- Some of its myriad users:



Summary

- **MongoDB** is a general purpose, document-based, distributed database
- **Cassandra** is a decentralized structured storage system
- **Logstash** is the log management system element of the ELK stack
- Coming up next:
 - Hadoop Distributed Filesystem (**HDFS**)
 - **HBase** non-relational database, (usually) runs on top of HDFS



Thank you for your attention!





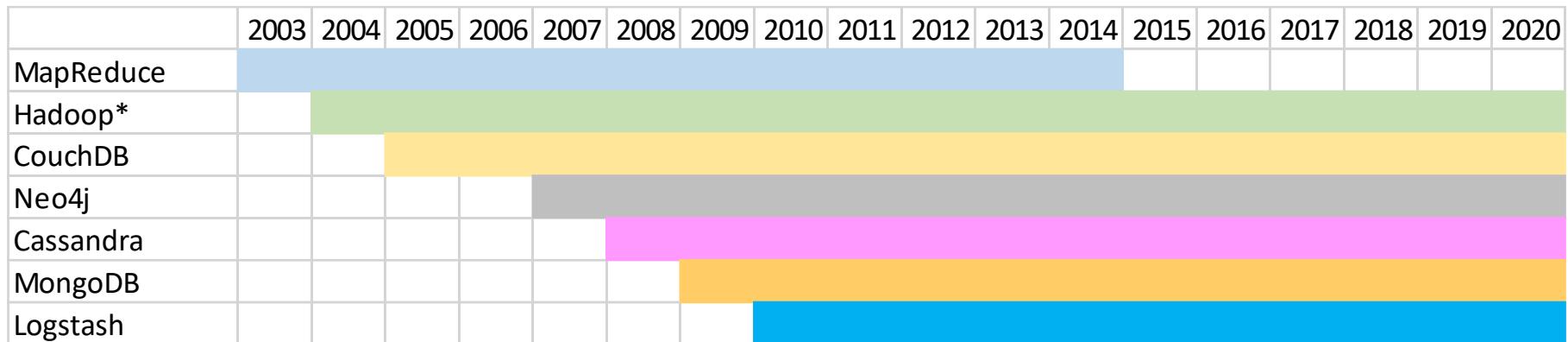
DATA STORAGE #2: THE HADOOP STORAGE ECOSYSTEM

*Open Source Technologies for Real-Time Data
Analytics*

Imre Lendák, PhD, Associate Professor



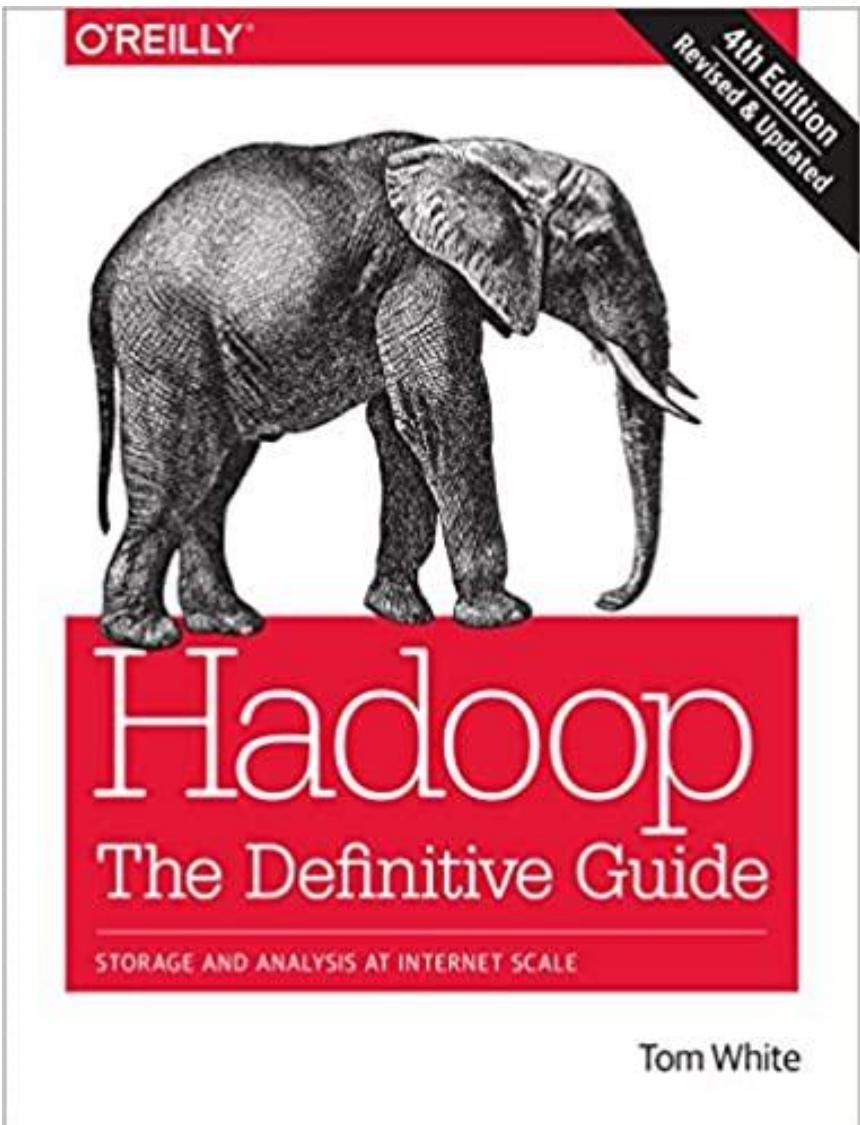
'Big data' storage timeline



* Hadoop lives as a distributed data storage platform, not as data processor

Chosen technologies

- Discussed earlier:
 - **MongoDB** is a general purpose, document-based, distributed database
 - **Logstash** is the log management element of the ELK stack
 - **Cassandra** is a decentralized structured storage system
- **Hadoop** is an open source software framework designed for data storage and processing
 - Hadoop Distributed Filesystem (**HDFS**)
 - **HBase** non-relational database, (usually) runs on top of HDFS
 - **Zookeeper** coordination service
- Tom White, Hadoop – The definitive guide, O'Reilly, 4th Edition, 2015

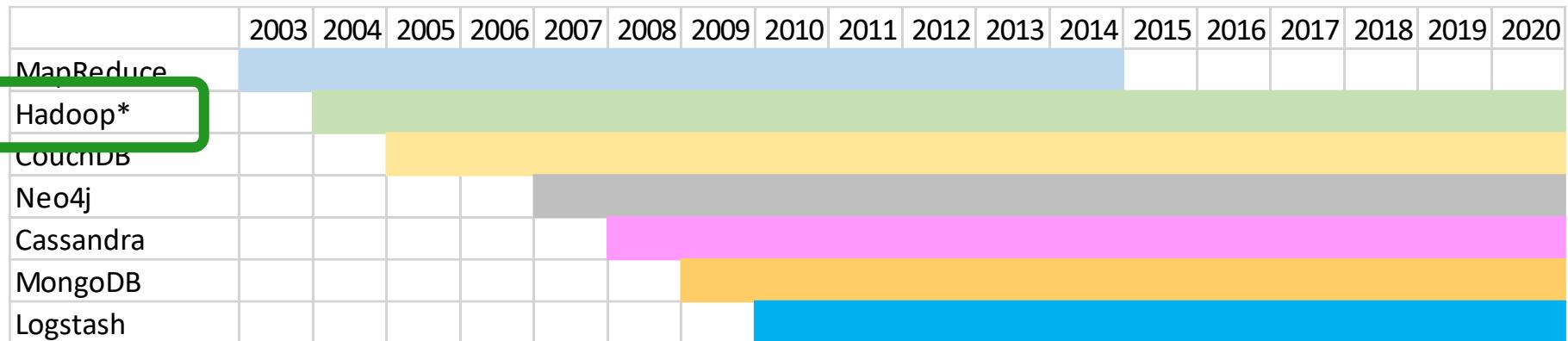


Tom White

HADOOP DISTRIBUTED FILESYSTEM



Hadoop in the timeline



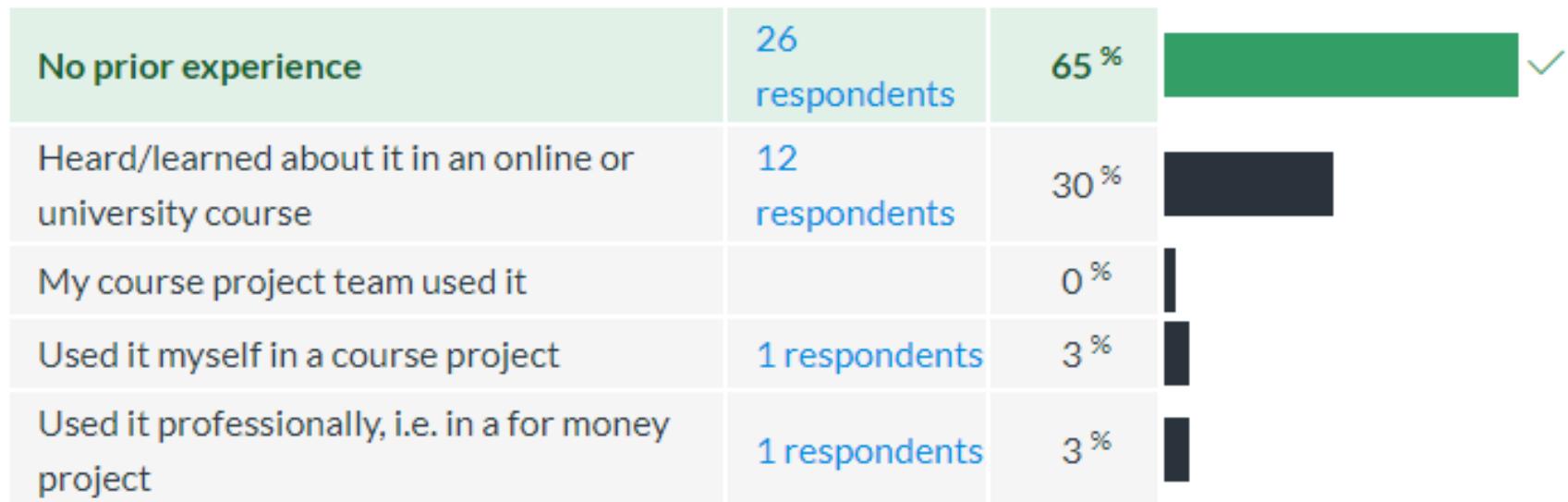
* Hadoop lives as a distributed data storage platform, not as data processor



Why Hadoop @ OST?

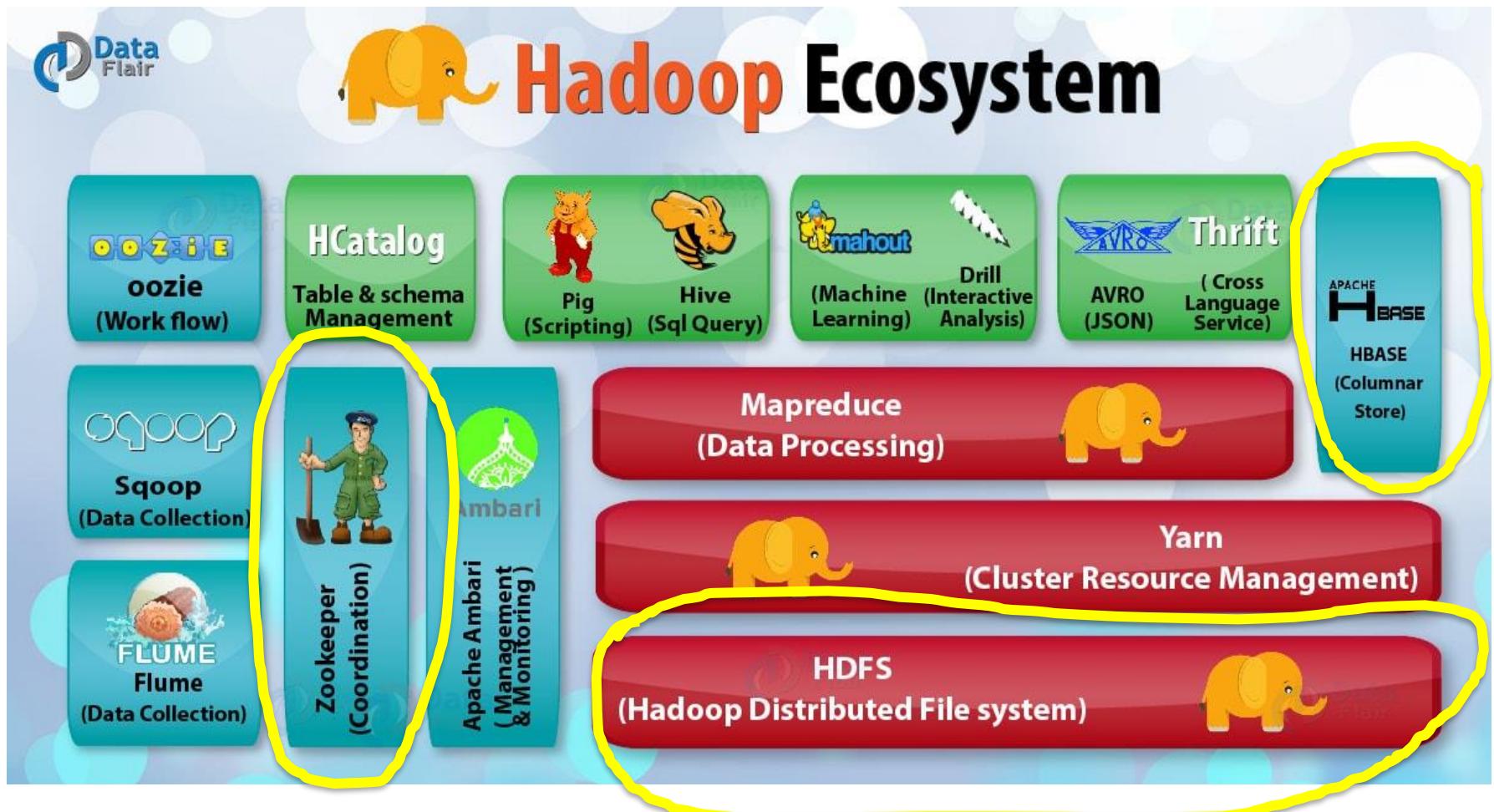
Attempts: 40 out of 40

Please rate your past experience in using the Hadoop for data storage:





The Hadoop Ecosystem



Hadoop introduction



Definitions

- **DEF:** Hadoop is an open source software framework designed for storage and processing of large-scale data on clusters of commodity hardware
- Created by Doug Cutting and Mike Carafella in 2005.
- **Trivia:** Cutting named the program after his son's toy elephant.

Use cases

- Data-intensive text processing
- Assembly of large genomes
- Graph mining
- Machine learning and data mining
- Large scale social network analysis



Hadoop motivation

Early large-scale computing

- Historically computation was processor-bound
 - Data volumes were relatively limited in size
 - Complicated computations were performed on that data
- Advances in computer technology was historically centered on improving the power of a single machine
- Today single machines (aka hosts/nodes) cannot handle the storage and processing needs of many use cases

Modern, distributed systems challenges

- “You know you have a distributed system when the crash of a computer you’ve never heard of stops you from getting any work done.” – Leslie Lamport
- Synchronizing data exchanges
- Managing a finite bandwidth
- Controlling computation timing is complicated
- Partial system failures
- More data in modern systems
- Data Node → Compute Node data copies are slow



HDFS defined

- **DEF:** Distributed filesystem = a filesystem which is capable to manage the storage of files (or generally data) across a distributed system consisting of networked computers (either cluster or grid)
 - Storage distribution is necessary when the datasets outgrows the storage capacity of the underlying hardware
- **DEF:** The Hadoop Distributed Filesystem (HDFS) is a filesystem designed for storing **very large**, bulk datasets on **commodity hardware** with **streaming data access** patterns
 - **Very large datasets** → up to terabytes in size
 - **Streaming data access** → write-once, read many times → read the whole or large segments of the data sequentially (streaming)
 - **Commodity hardware** → commonly available, inexpensive (server) HW available from multiple vendors, e.g. HP, Lenovo, Dell, etc.



File storage

Files are split into blocks

- The usual block size is 128 MB, but can be configured to be larger
- **Note:** A 1 MB piece of data does not consume 128 MB of space even if the block size is 128 MB

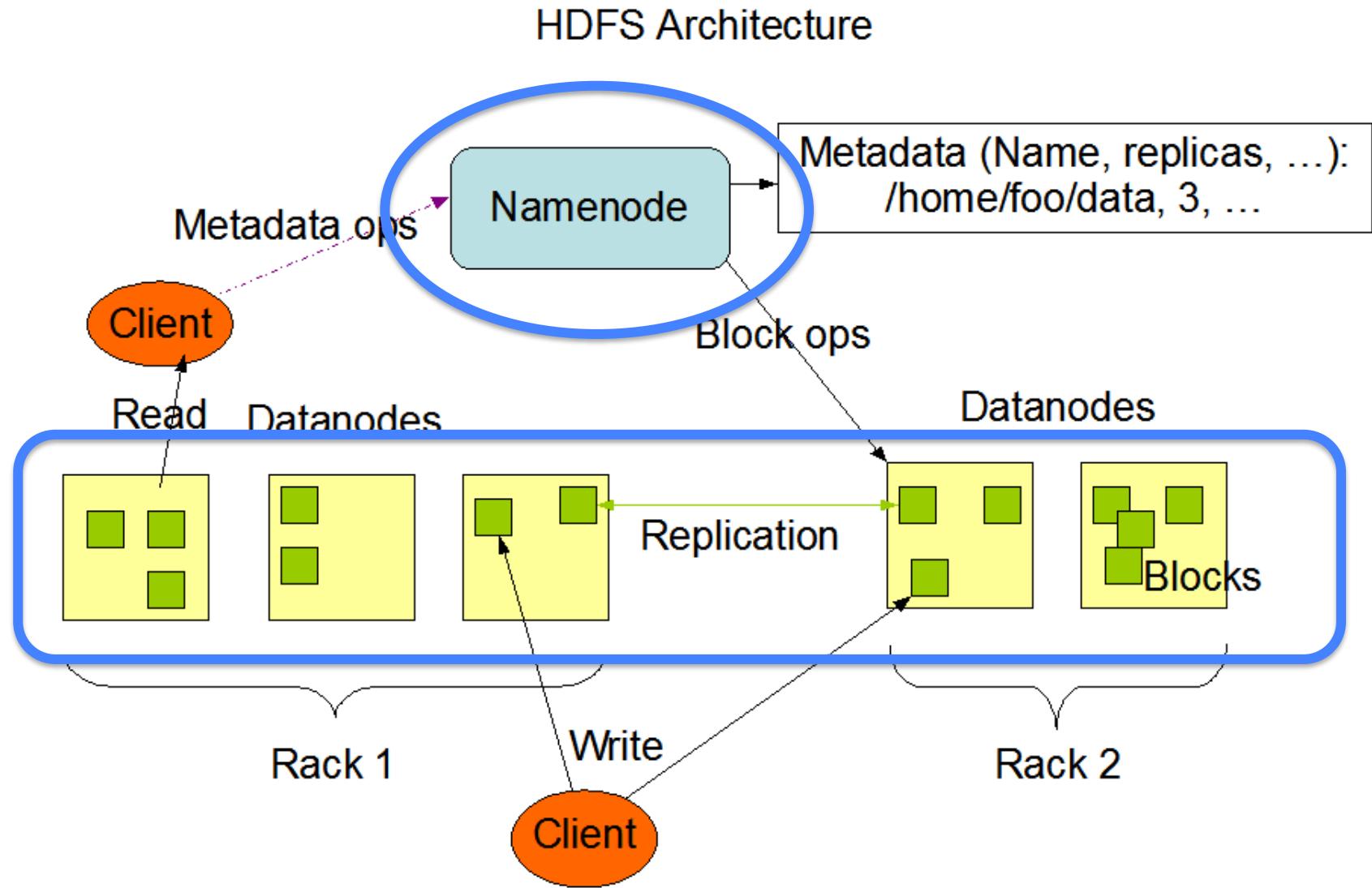
Blocks are split across many machines at load time

- Different blocks from the same file will be stored on different machines

Suboptimal use cases

- Low-latency (random) data access
- Lots of small files

HDFS Architecture





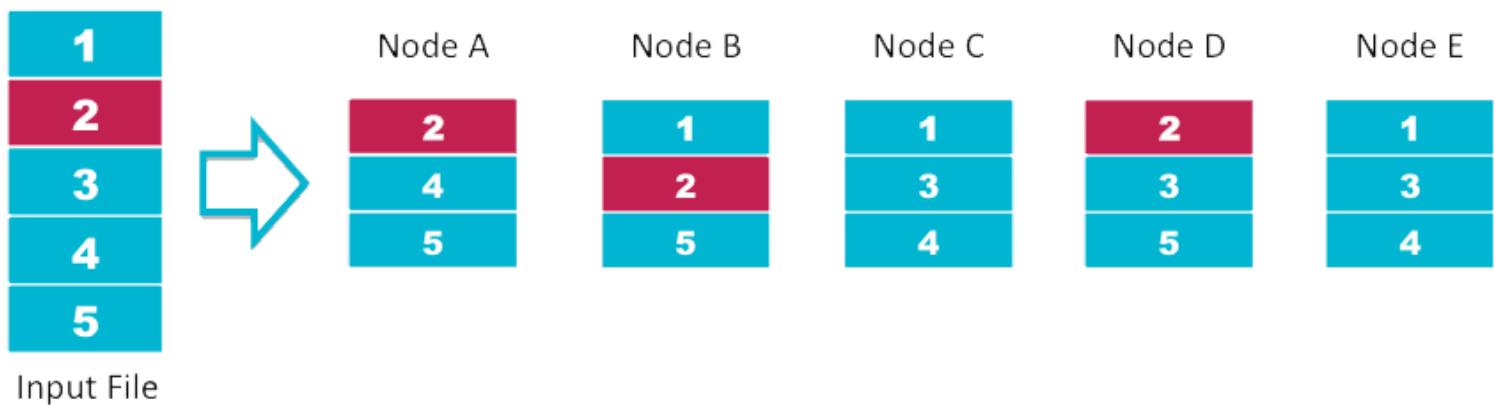
Namenodes

- HDFS namenodes manage the filesystem namespace
- The namenode **tracks the datanodes** on which the blocks for a given file are located
 - Block location information is not persisted → it is re-created when the system starts (simple!)
- They maintain an **in-memory** copy of
 - the filesystem tree, and
 - metadata for all files and directories in the tree.
- The **namenode state** is persisted on the local disk and consists of the following elements
 - Namespace image
 - Edit log

Datanodes

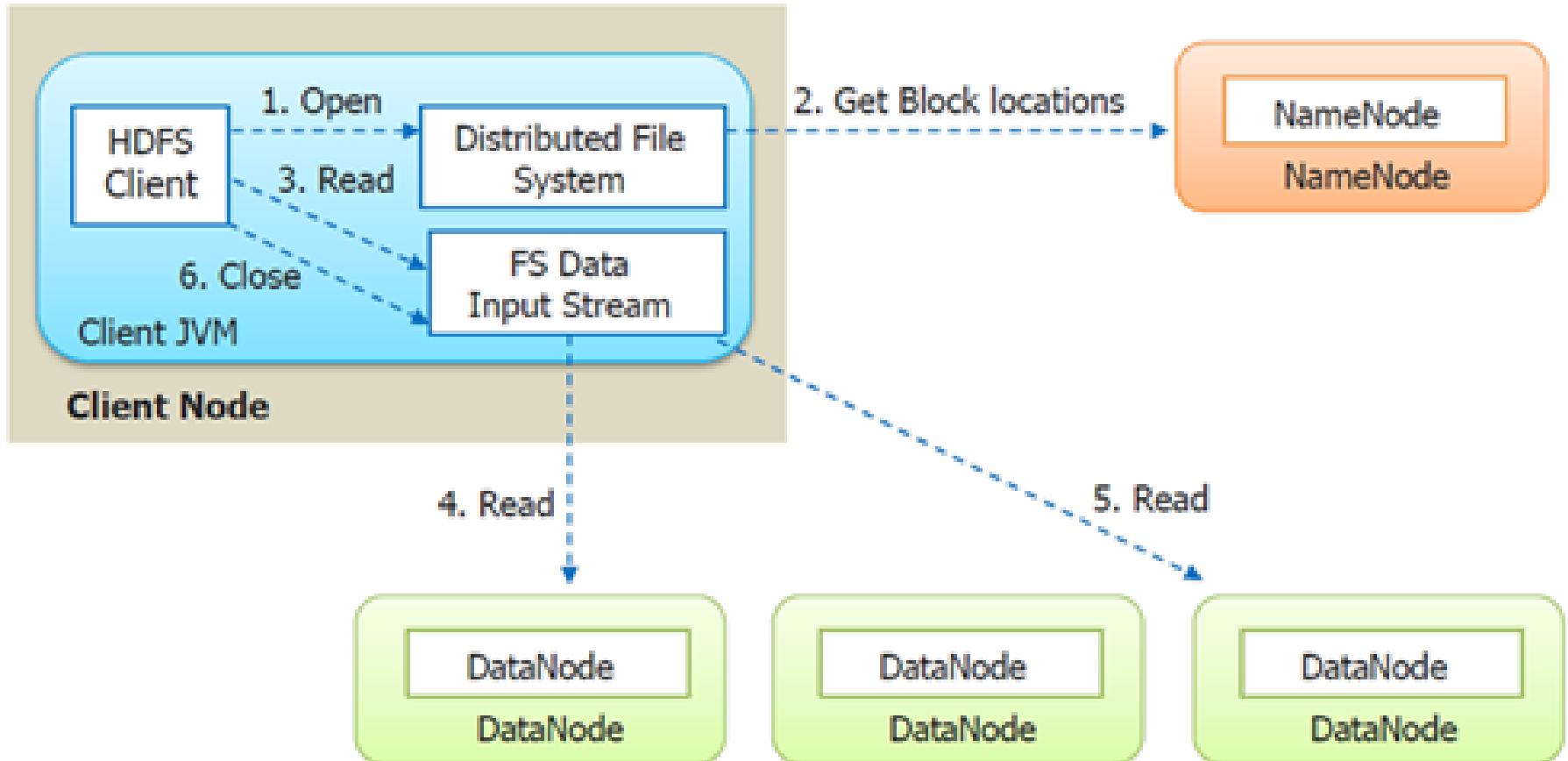
- Datanodes store and retrieve blocks
- Datanodes periodically report the list of blocks managed to the Namenode(s)

HDFS Data Distribution

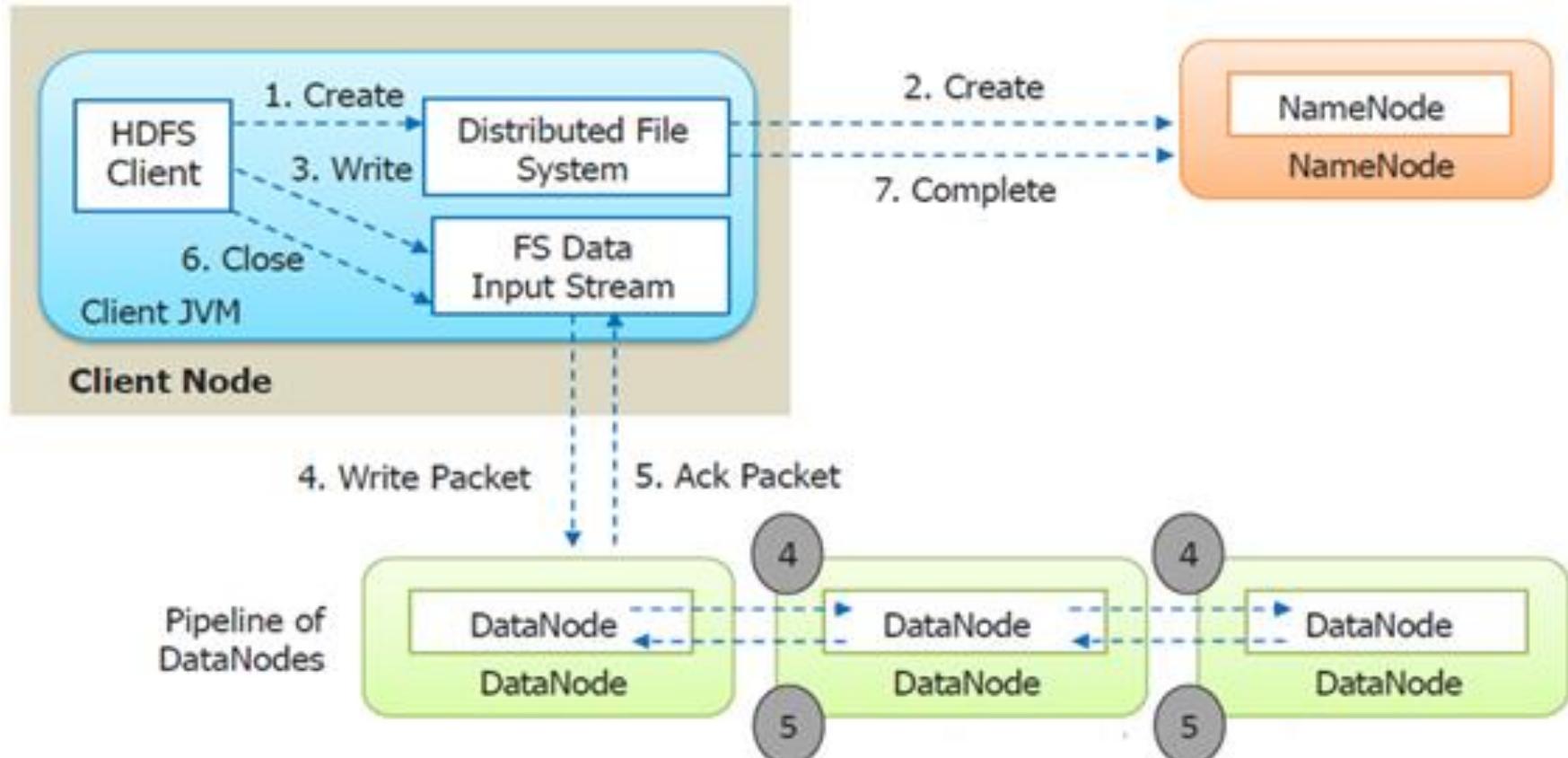


- Note: a completely failed Namenode would destroy the contents of the filesystem as it could not be reconstructed without the metadata stored on the Namenode

File read



File write

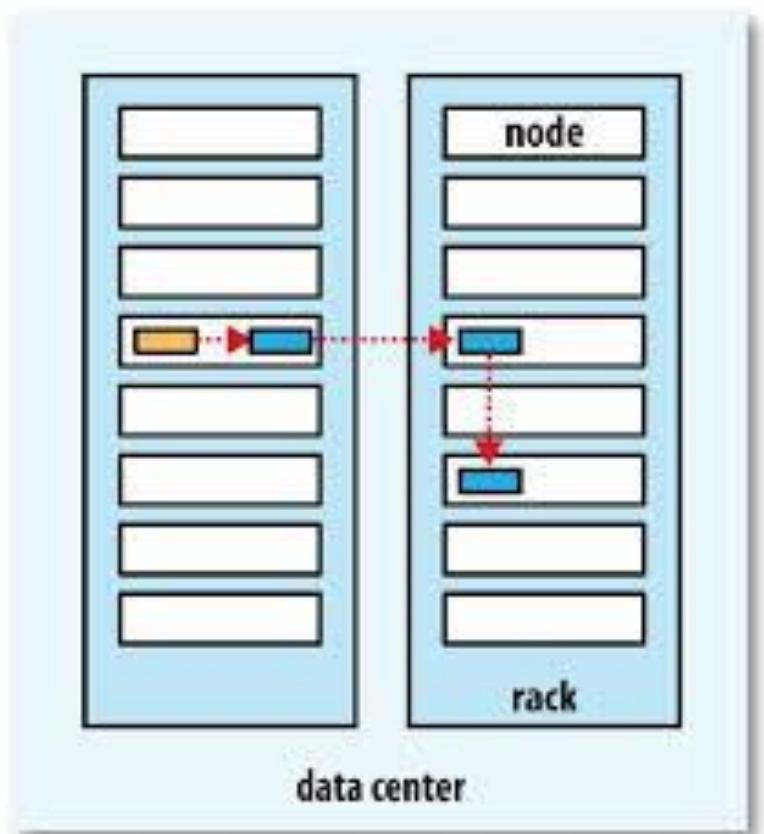


Namenode fault tolerance

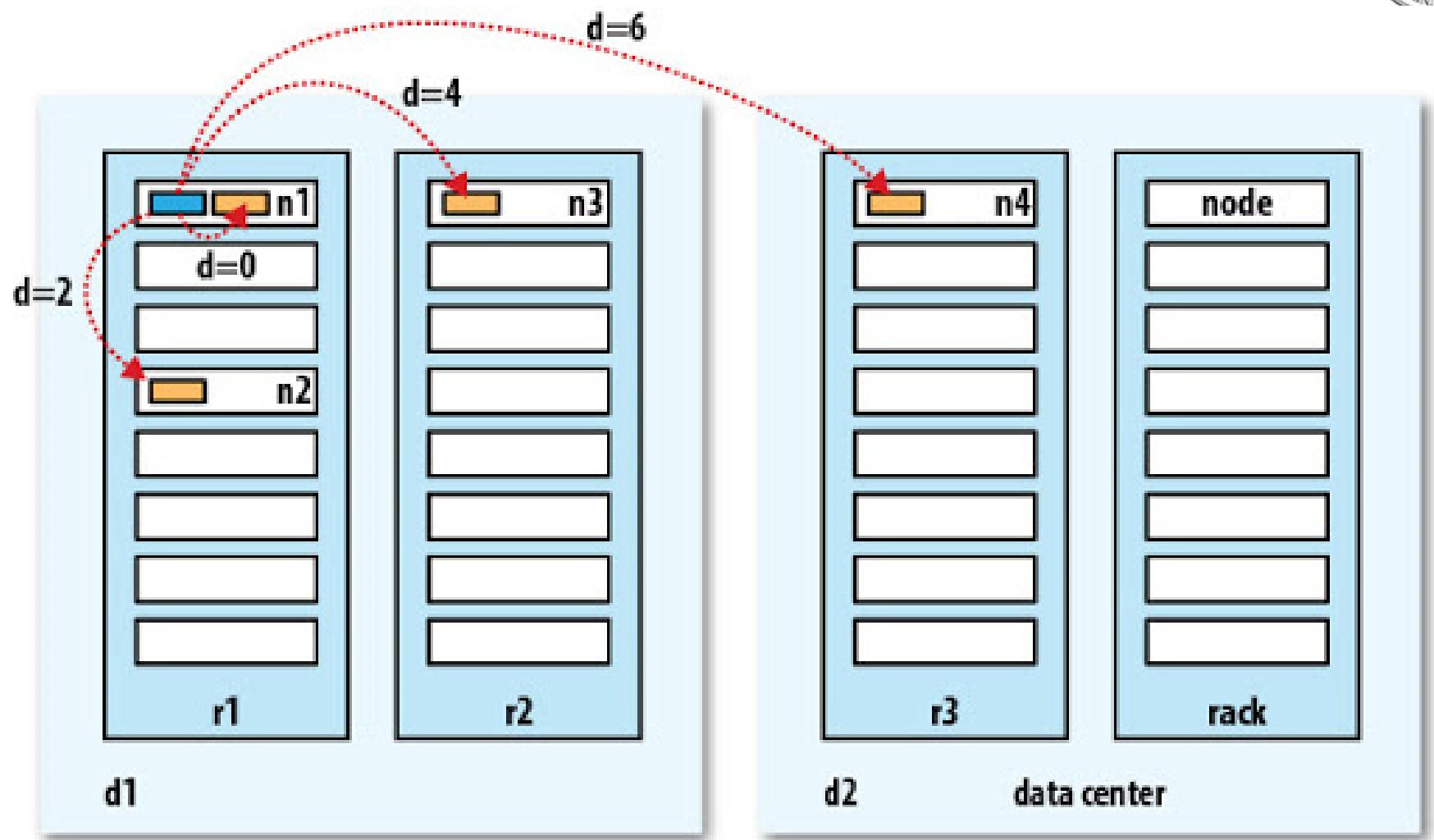
- Namenode instances are potential single points of failure (SPoF) → backup mechanisms are introduced
 - **Persist metadata to multiple filesystems**, usually local disk and a remote NFS mount
 - Introduce a **secondary namenode** which runs on a separate (physical) host which maintains a separate namespace image into which it merges the edit log → strong HW necessary
 - Clients need to be configured to handle namenode failover
- Namenode recovery steps in the case of primary failure:
 1. An administrator initializes a new primary namenode
 2. The namespace image is loaded into memory
 3. The (unmerged part of the) edit log is replayed
 4. Sufficient amount of block reports are received from the datanodes
 5. The (new) primary namenode leaves safe mode

Data replication

- How does the namenode decide where to store data replicas?
- Hadoop's default replication strategy is 3-fold (and implemented in the namenodes):
 - Place the 1st replica on the same node
 - Place the 2nd replica off-rack, i.e. in a randomly chosen different rack
 - Place the 3rd replica in the rack of the 2nd replica but on a different node
- **Note:** HDFS clusters are usually limited to single data center



HDFS replication & distances



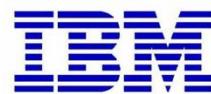
Who uses Hadoop/HDFS?



eHarmony®



facebook



amazon.com®



The New York Times

JPMorganChase



YAHOO!®

HBASE



Introduction

- DEF: **HBase** is a distributed column-oriented data store built on top of HDFS
 - HBase is the Hadoop solution for real-time, read/write random access to very large datasets
 - Development started in 2006 by Chad Walters & Jim Kellerman @ Powerset
 - Initial release: 2008
 - Preview release: Aug 2020
- The HBase data model is not relational and it natively does not support SQL
- Canonical use case: webtables of (billions of) crawled web pages



HDFS and HBase compared



Plain old HDFS

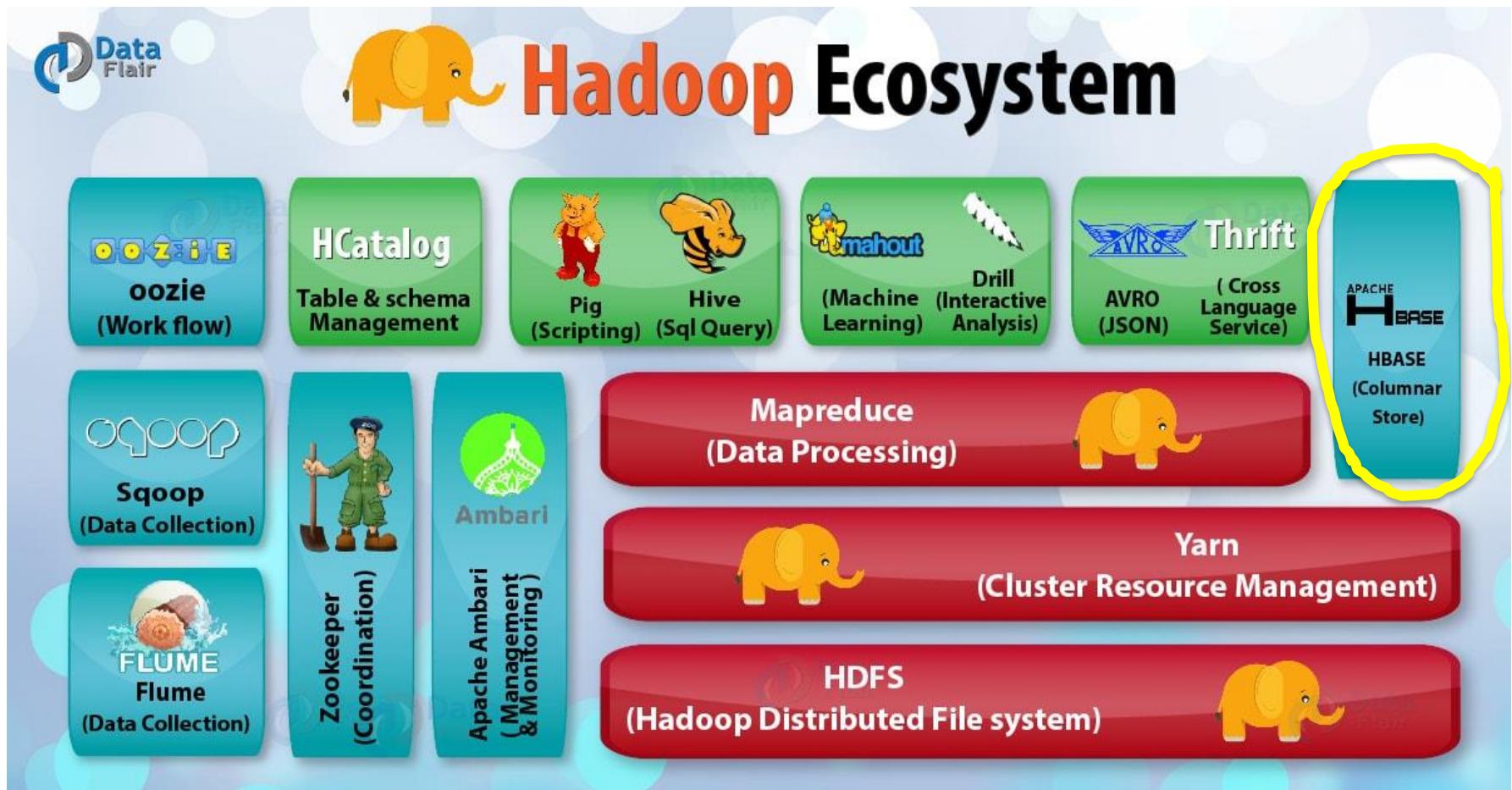
- Designed for batch processing relying on scans over (very) big files
- Not good for record lookup → slow seek
- Not good for incremental addition of small batches → designed for bulk load
- Not good for updates → append or reload (!)

HBase on HDFS

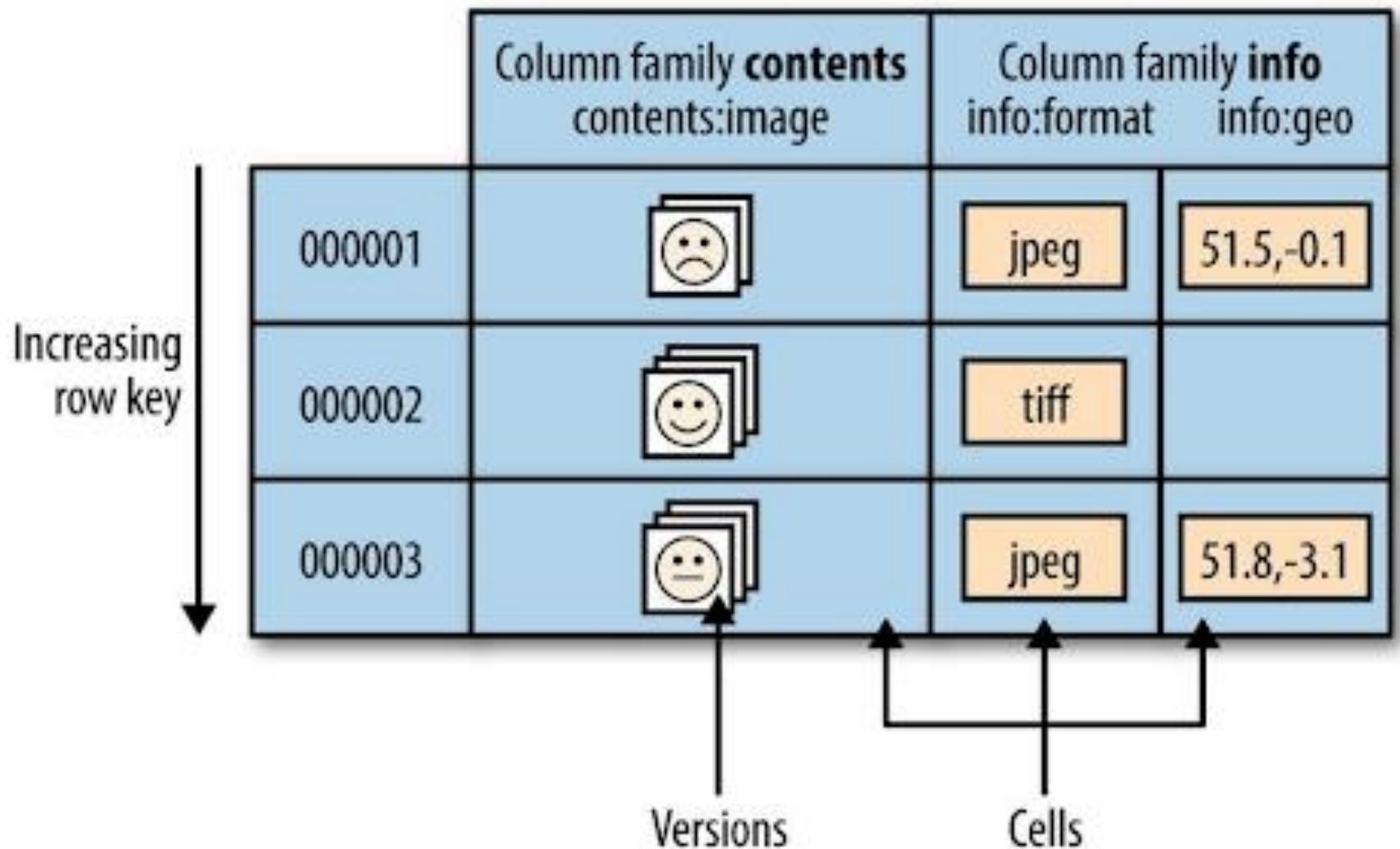
- Fast record lookup
- Support for record-level insertion → cell versioning
- Support for updates with versioning → row-level atomic updates



The Hadoop Ecosystem



Data model



Tom White, Hadoop – The definitive guide, O'Reilly, 4th Edition, 2015

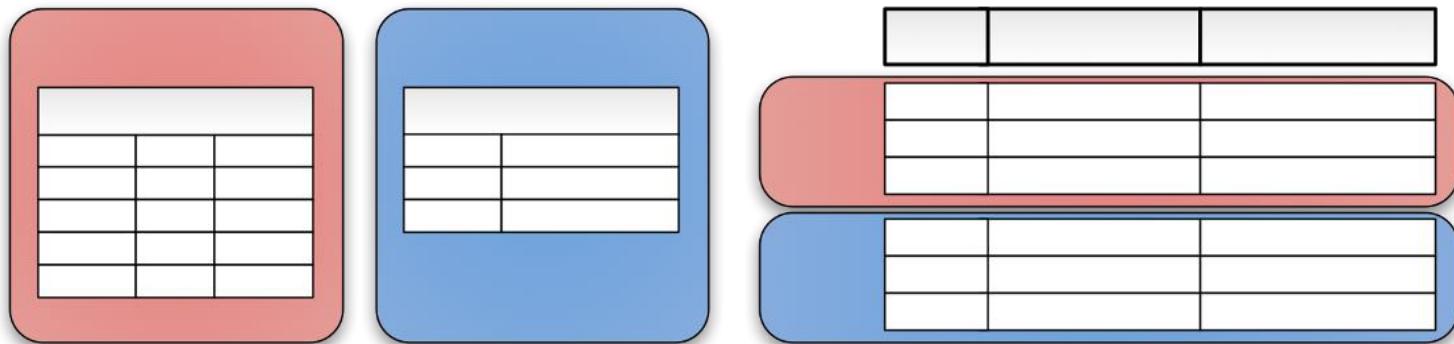


Data model described

- Tables are made of rows and columns similarly to RDBMS
- All table accesses are via **row keys**, aka primary keys
 - Row keys are byte arrays → keys can be of any data format
 - Table rows are sorted by row keys → the sort is byte-ordered
 - Row updates are atomic regardless of the column count → simple!
- Columns are grouped in **column families**
 - Share a common prefix, e.g. info:format, info:geo are members of the 'info' column family
 - Column family names consist of printable, human-readable characters
 - Column families are specified during schema definition
 - Columns can be added dynamically to column families
- **Table cells**
 - contain arrays of bytes
 - are versioned, usually timestamp assigned by HBase on entry

Data storage

- Column families are stored together → column family-oriented store (!)
- HBase performs automatic horizontal partitioning of tables into **regions** as the table (out)grows (a single host)
 - A region is defined by the table name, first row (inclusive) and last row (exclusive)

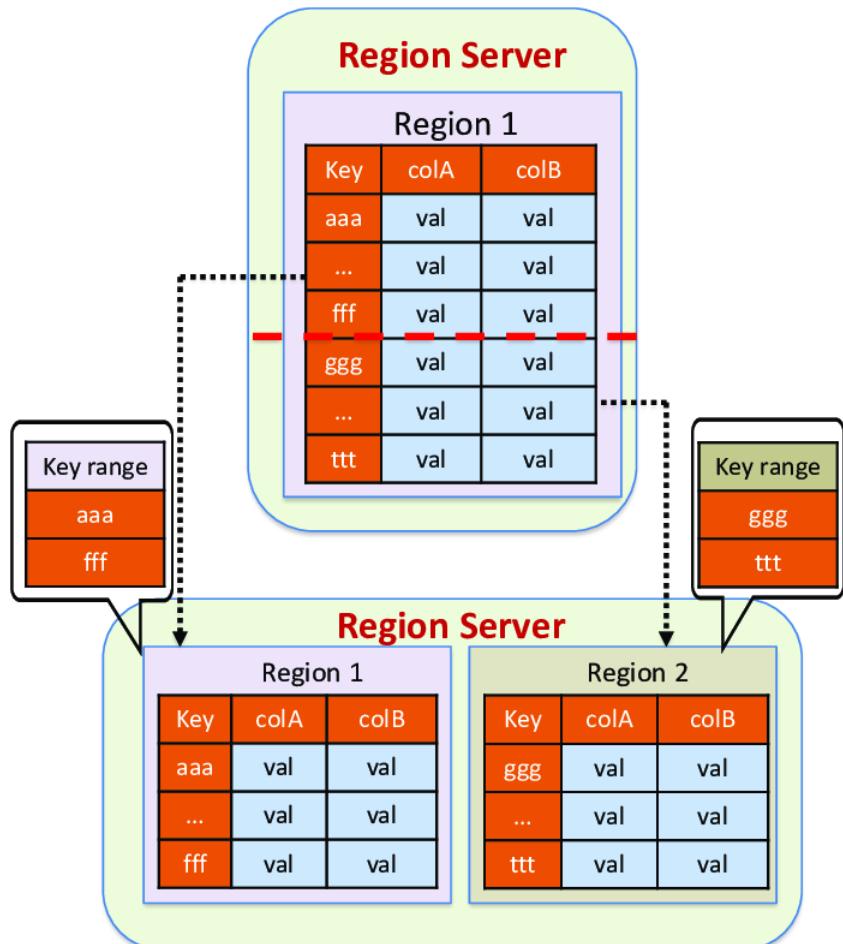


Vertical

Horizontal

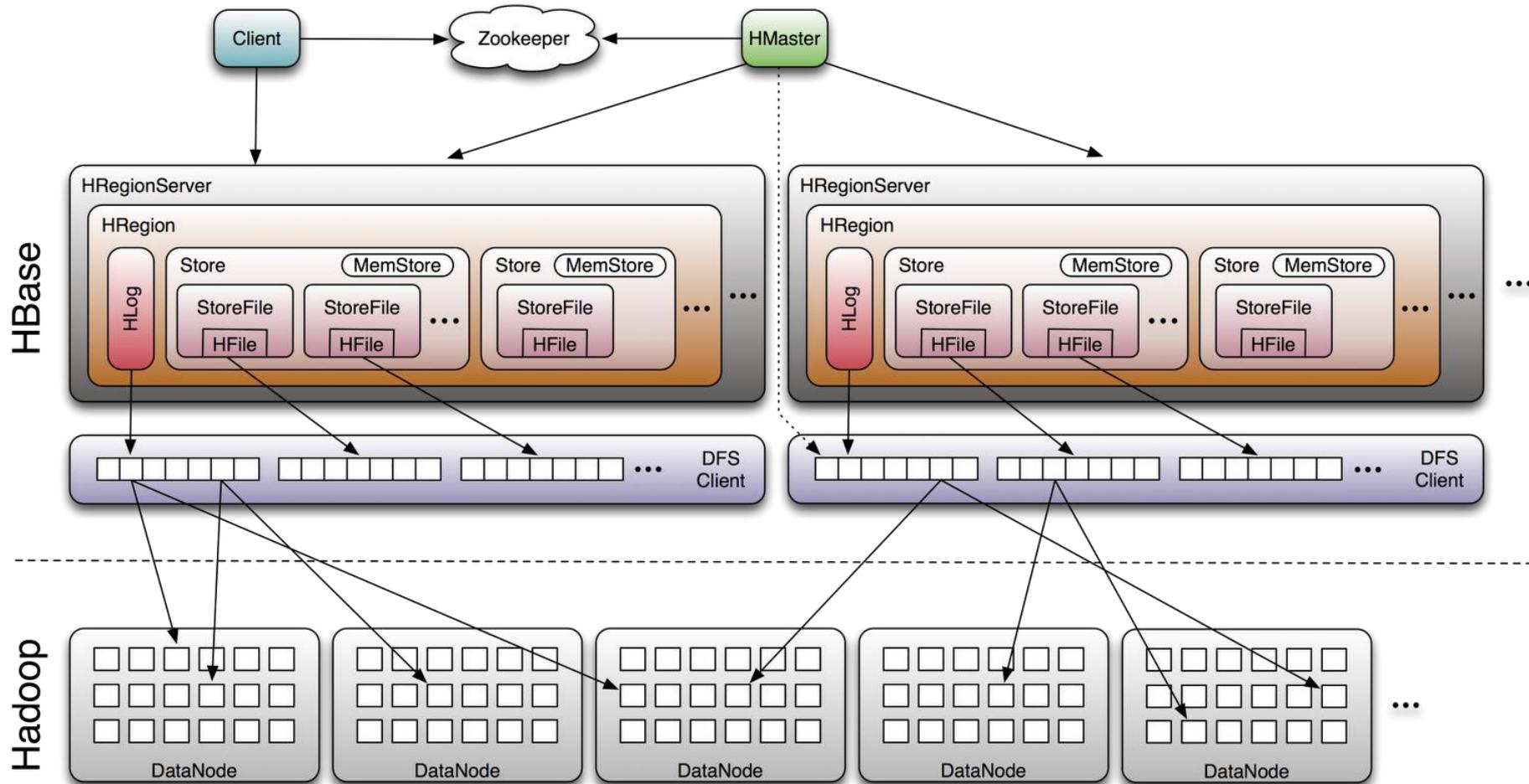
HBase table splitting

- Tables are initially hosted on a single node
- Tables are split horizontally when a table outgrows the confines of a single node, e.g. if there is insufficient storage capacity
 - The horizontal partitions are 'regions'
- Region servers are responsible for the storage of regions





Satellite view





Old(ie-goldie) meets new

RDBMS

1. Initial public launch of web server with a traditional RDBMS data store
2. Service popular → too many RDBMS reads → **memcache** common queries
3. Service even more popular → too many RDBMS writes → buy custom, **expensive HW**
4. New website features (requested by the millions of customers) increase (SQL) query complexity → **de-normalize data**
5. RDBMS server(s) overloaded, the website is too slow → **avoid joins, pre-materialize data** and keep in-memory
6. Reads work, writes are still slow → **no way forward** with an RDBMS

HBase(d) solution

- **No indexing** → performance independent of table size
- **Automatic partitioning** → as the data grows, it is split into regions
- **Linear scalability** → regions automatically utilize new (server) nodes added to the HW cluster
- **Commodity hardware** → runs on clusters of servers worth up to 5000 USD
- **Fault tolerance** → numerous cheap nodes lessen the worry caused by node downtime

ZOOKEEPER



Introduction

Definition

- Zookeeper is a highly-available process coordination services in distributed systems (DS).
- Designed and written to allow computer scientists and programmers to **not worry about coordination** in DS

Motivation

- No single computer can manage the vast amounts of data on the web scale → necessary to use distributed systems
- Solve (some of) the ages old challenges in distributed systems
 - Network reliability
 - Latency
 - Topology changes
 - Heterogenous systems
 - Limited bandwidth
 - Faults

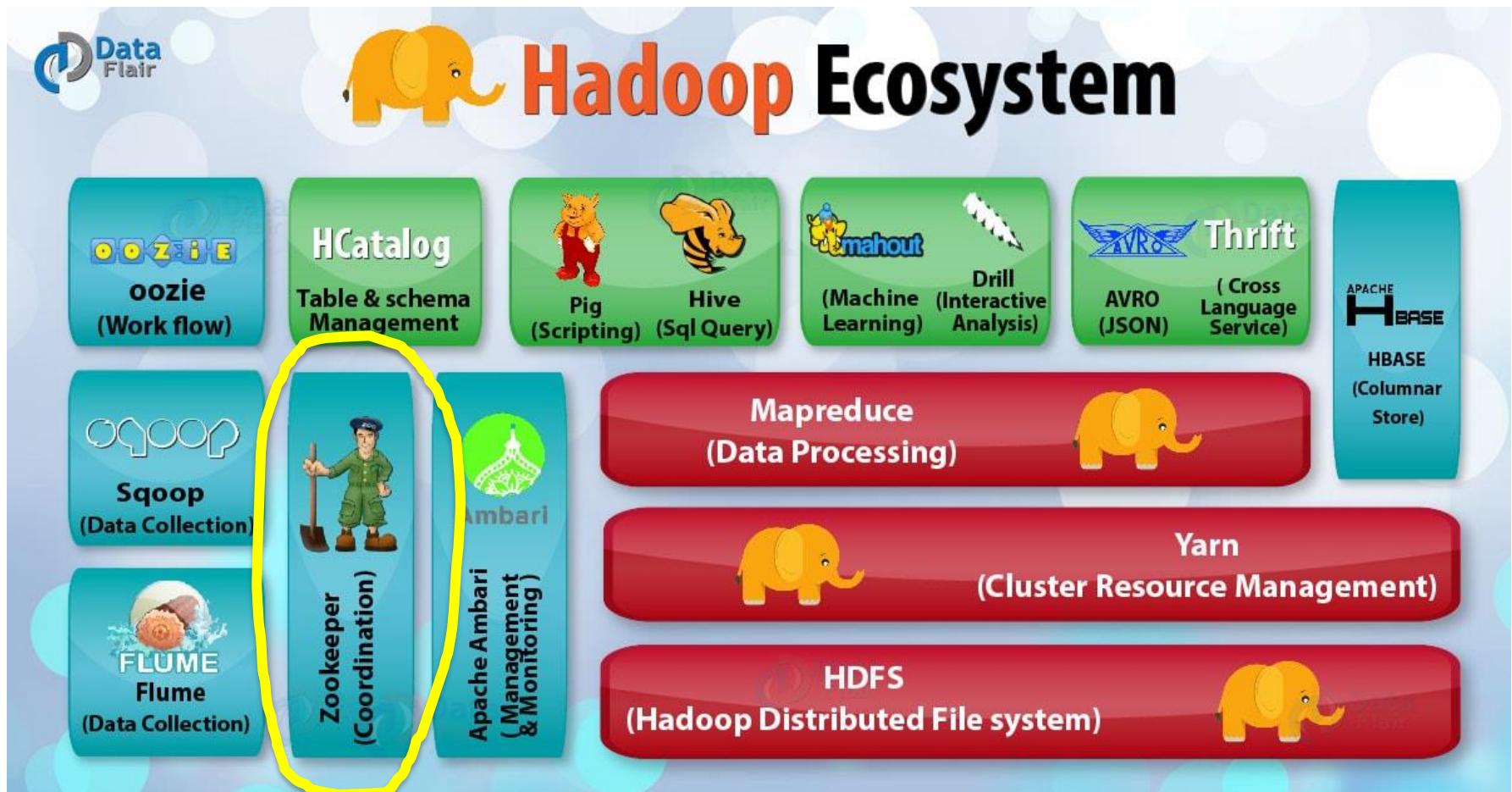


Challenges to be solved

- **Coordinator selection** → how to select a process from a group of equal processes
- **Locking** → how to coordinate the use of limited resources, e.g. off-site storage available via a slow link
- **Configuration management** → how to ensure that each node in a 150-node computing cluster receives the same configuration update

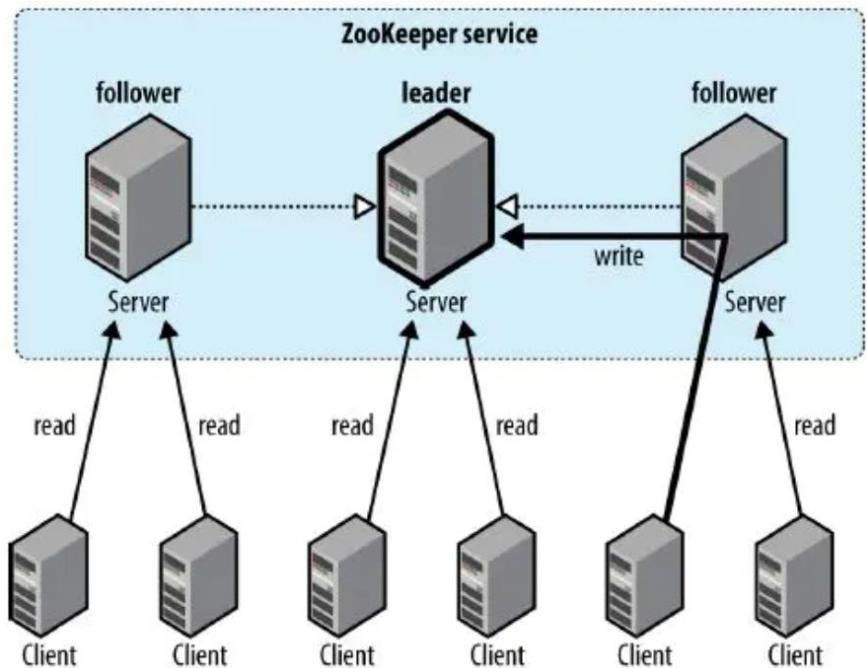


The Hadoop Ecosystem





Zookeeper system architecture

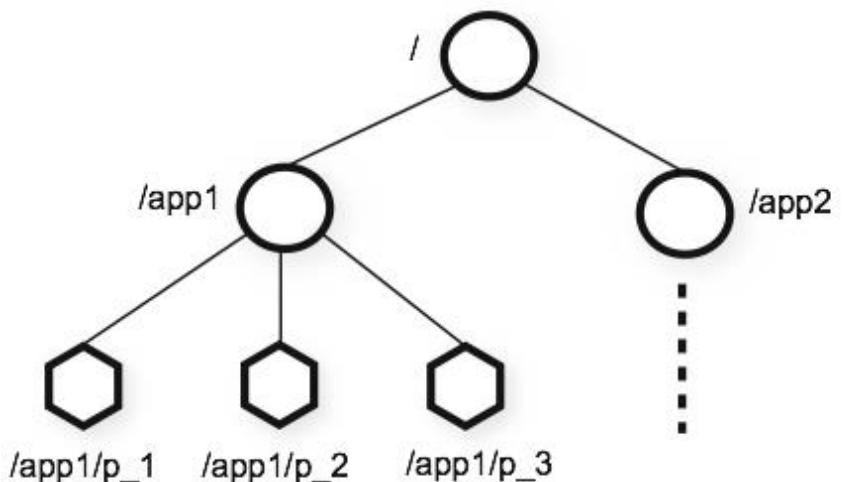


- ZooKeeper service → clients observe it as a single component
- Ensemble → cluster of zookeeper servers
- Server → provides access to Zookeeper
 - Leader → write point
 - Follower → replicas
- Client → ZK service client, usually a node in a(ny) cluster

(Figure 21-2) from Tom White, Hadoop – The definitive guide, O'Reilly

Data model

- ZK maintains a tree of nodes
 - Nodes are named ‘znodes’
- znodes are referenced by paths
 - znodes path namespaces are similar to filenames on Linux
- znode types
 - Ephemeral → tied to a client session & deleted when the session ends
 - Persistent → not tied to a client session, explicit delete
- Zookeeper is designed for coordination, not data storage
 - Data size is limited to 1 MB





Operations

- create = create a znode – the parent must exist
- delete = remove a znode without children nodes
- exists = check node existence
- getChildren = get list of children nodes
- getData, setData = access/modify the data associated with a node
- getACL, setACL = access/modify the access control list (ACL) of a node
- sync = synchronize a client's view with a znode

- multi = batch together multiple primitive operations into a transaction



Keeping clients updated

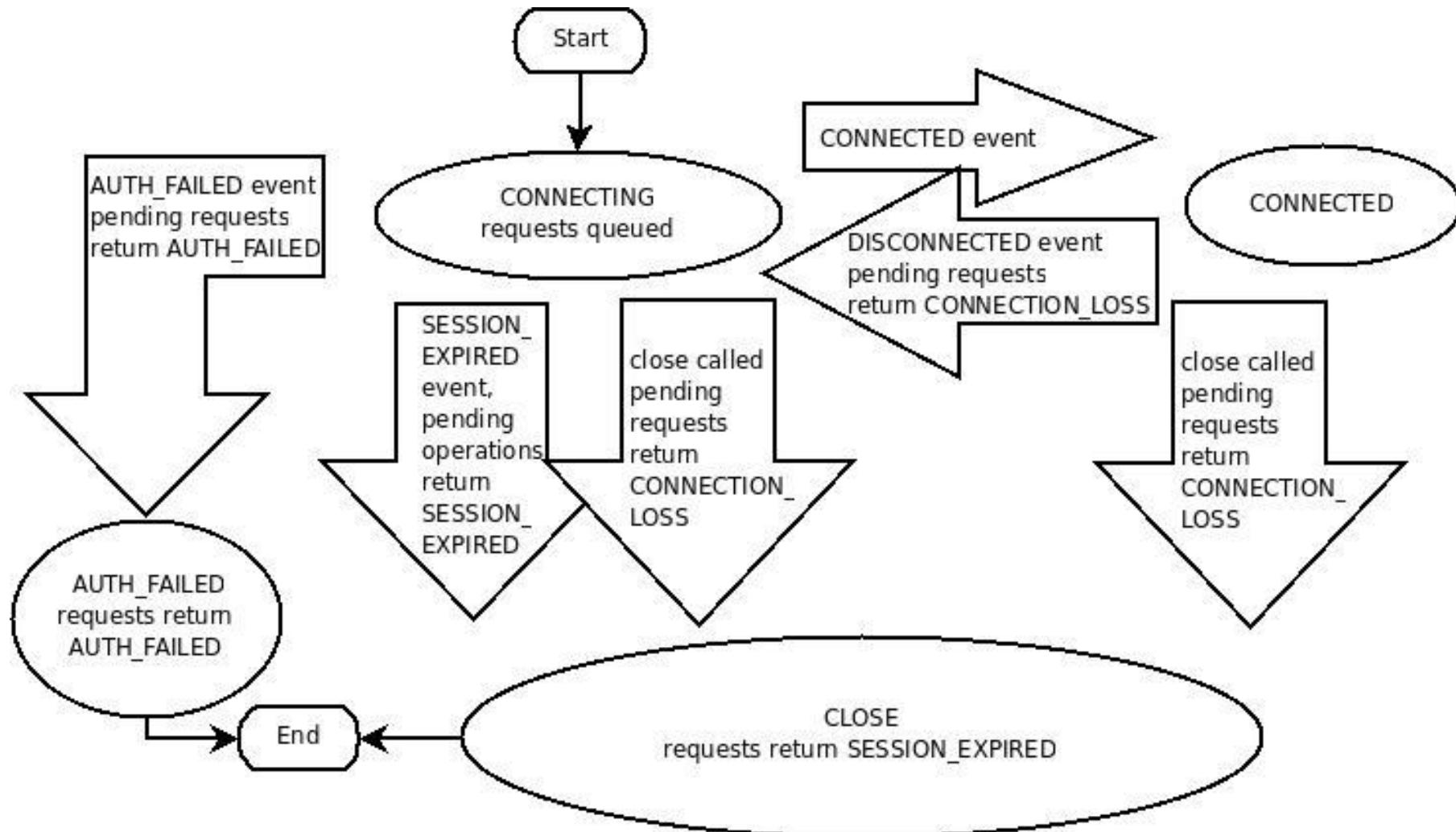
- Watch triggers are used to notify clients
- Watches can be set on:
 - ‘exists’ → triggered when the watched znode is created, deleted or its data is updated
 - ‘getData’ → triggered when the data watched is deleted or updated
 - ‘getChildren’ → triggered when any child of the watched znode is modified



Sessions

- ZK clients are configured with a (sub)set of servers in the ensemble
- Clients try to connect to the configured servers iteratively
- Once a successful connection is made, a session is established
 - Sessions are long-lasting, but can time out
 - Clients need to maintain sessions by sending heartbeat signals to the server
 - If the timeout period expires, the server tears down the session and all ephemeral znodes are deleted
- Failover is handled by the ZK client
 - Sessions and ephemeral nodes are persisted in the ensemble and survive partial failures

Sessions depicted



<https://zookeeper.apache.org/doc/r3.3.3/zookeeperProgrammers.html>



Synchronization and consistency

- ZK ensures that every modification is replicated to the majority of the servers in the ensemble
 - The ZooKeeper Atomic Broadcast (ZAB) protocol (2008) is used
 - Gossip or Paxos are not used, although they are similar
 - Zab operates in two phases:
 1. Leader election → the servers in the ensemble select a 'leader' – this phase is over when the majority of servers is synced with the leader
 2. Atomic broadcast → all write requests are forwarded to the 'leader', which broadcasts them to the followers
- Zookeeper ensembles consist of odd numbers of servers
 - The ensemble is operational if more than half of its servers are up → in a 5-node ensemble two nodes might fail w/o operational disruption



Data consistency guarantees

- **Sequential consistency** → updates from a particular client are applied sequentially, i.e. in the order they are observed by the ZK ensemble
- **Atomicity** → updates either succeed or fail (on the ensemble level)
- **Single system image** → clients see the same view of the system regardless of the ZK server chosen and connected
- **Durability** → successful updates survive server failures
- **Timeliness** → delays in synchronization are guaranteed to be short and measured in multiples of 10 seconds
 - Servers with stale data shut down → they avoid to serve clients with ‘old’ data



Security

Authentication

- Authentication is optional
- Authentication variants
 - ‘digest’ → authentication with username & password
 - ‘sasl’ → Kerberos
 - ‘ip’ → IP address-based auth

Authorization

- Authorization via Access Control Lists (ACL)
- Access levels:
 - CREATE – node creation
 - READ – read children/data
 - WRITE – allowed to setData
 - DELETE – delete child node
 - ADMIN – ability to setACL



Who uses ZooKeeper?

PMC Members

ZooKeeper's active PMC members are

Username	Name	Organization	Time Zone
tdunning	Ted Dunning	MapR Technologies	-8
camille	Camille Fournier	RentTheRunway	-5
phunt	Patrick Hunt	Cloudera Inc.	-8
fpj	Flavio Junqueira	Confluent	+0
ivank	Ivan Kelly	Midokura	+2
mahadev	Mahadev Konar	Hortonworks Inc.	-8
michim	Michi Mutsuzaki	Nicira	-8
cnauroth	Chris Nau Roth	Hortonworks Inc.	-8
breed	Benjamin Reed	Facebook	-8
henry	Henry Robinson	Cloudera Inc.	-8
rgs	Raul Gutierrez Segales	Pinterest	-8
rakeshr	Rakesh Radhakrishnan	Intel	+5:30
hanm	Michael Han	Twitter	-8
andor	Andor Molnar	Cloudera Inc.	+1

Summary

- Hadoop Distributed Filesystem (**HDFS**)
- **HBase** non-relational database, (usually) runs on top of HDFS
- **ZooKeeper** (cluster) coordination service



Thank you for your attention!





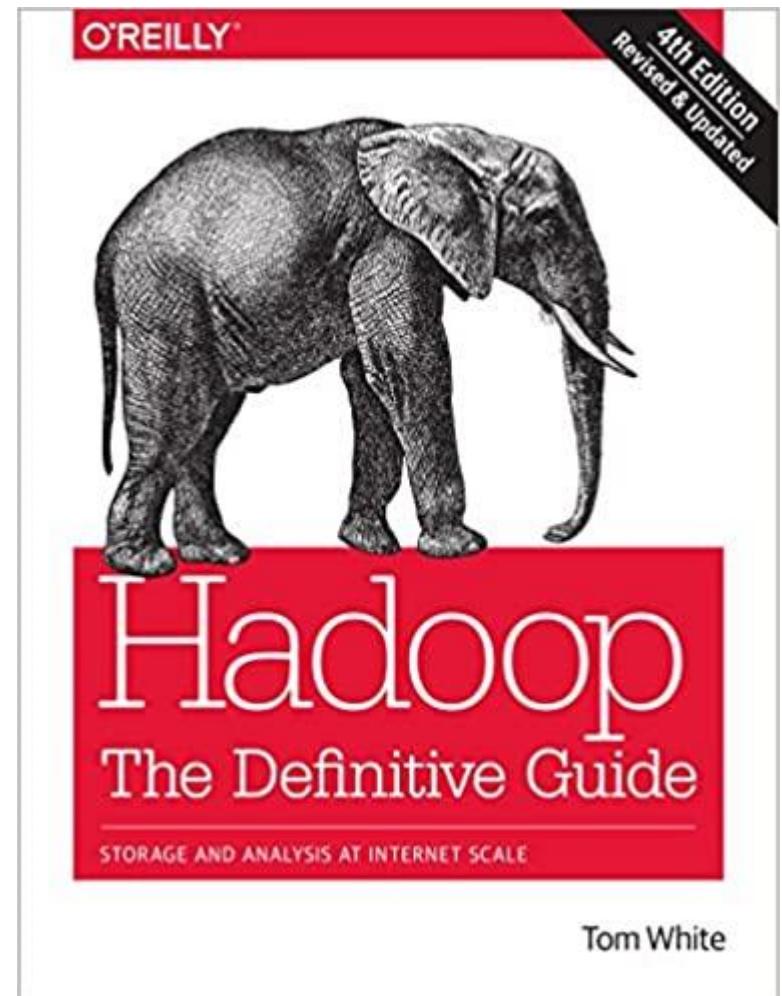
DATA ANALYSIS SOLUTIONS: MAPREDUCE

*Open-source Technologies for Real-Time Data
Analytics*

Imre Lendák, PhD, Associate Professor

Chosen data analytics topics

- **MapReduce** implements the map & reduce paradigm known from functional programming
 - Discussed in this lecture!
 - Mostly based on the Hadoop book by T.White
- **Spark** is an in-memory batch processing pipeline
 - Discussed later!
- **ElasticSearch** search & analytics engine

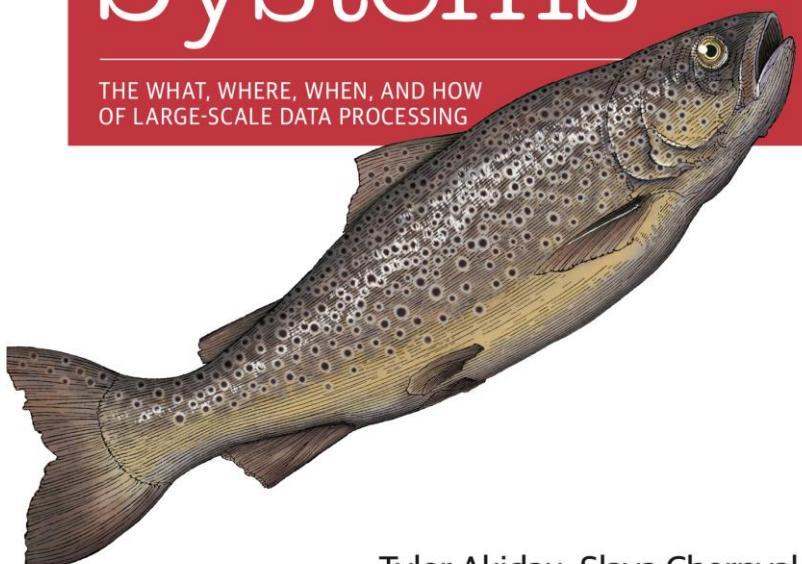
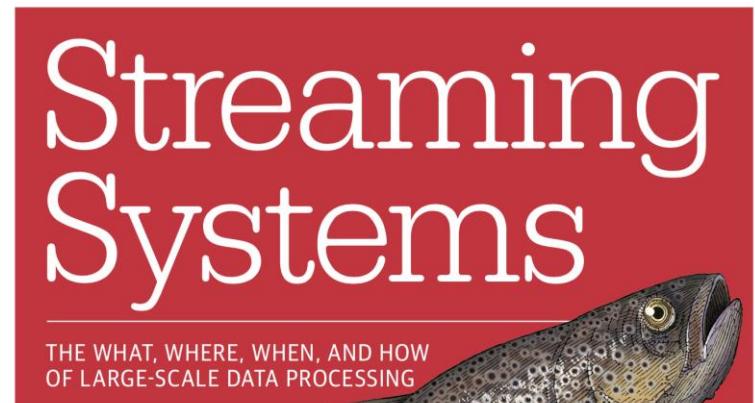


Tom White, Hadoop – The definitive guide, O'Reilly, 4th Edition, 2015



Additional resource

O'REILLY®

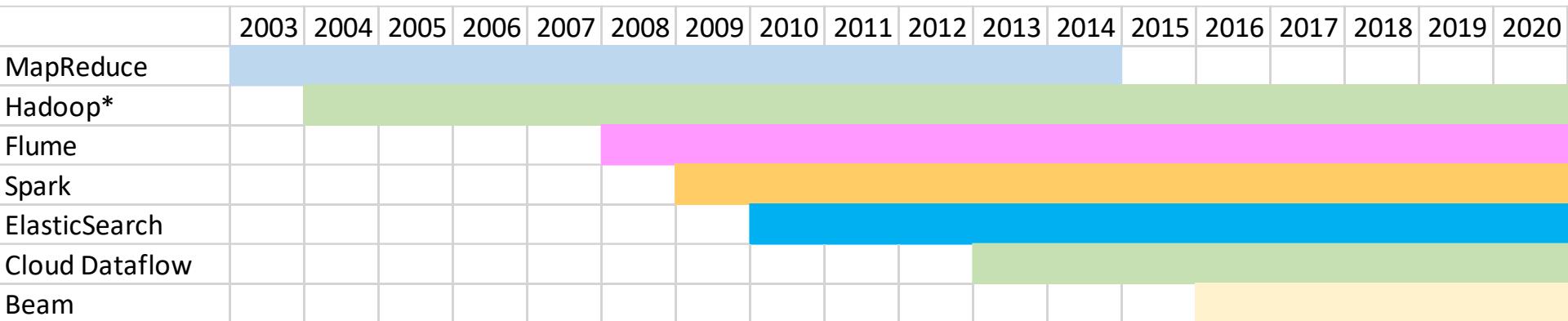


Tyler Akidau, Slava Chernyak
& Reuven Lax

- **Q:** Why the Streaming systems book if there is absolutely no streaming in this lecture?
- **A:** Akidau et al work at Google and have local knowledge about
- **Note:** MapReduce was initially a proprietary Google tech



Data analysis timeline

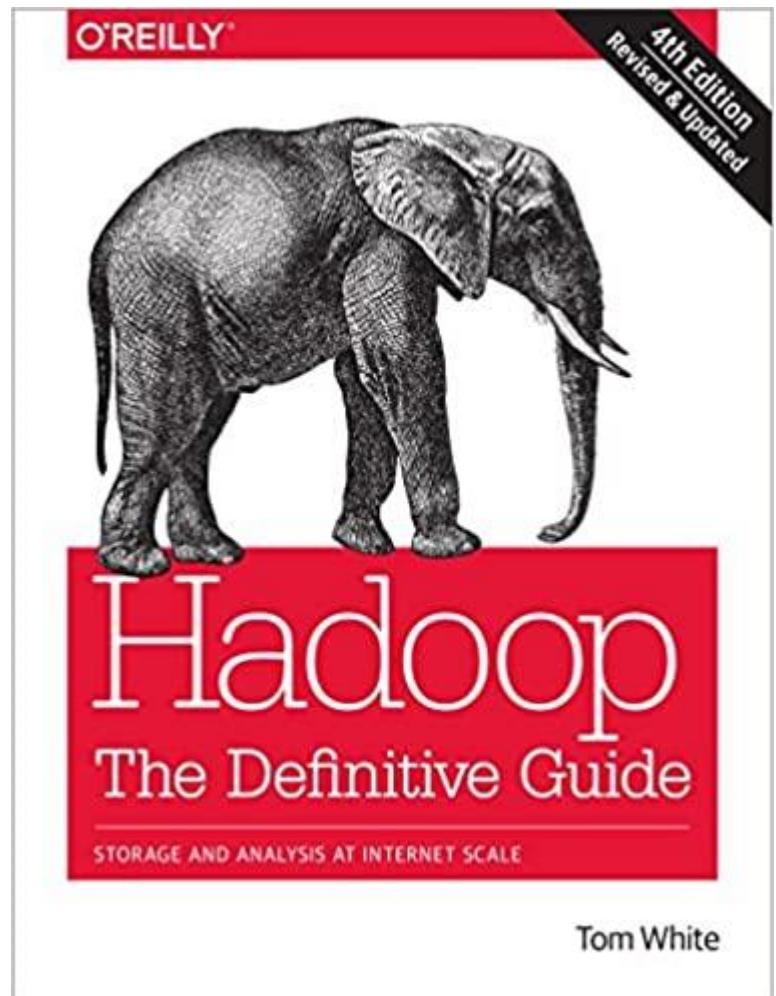


* Analysis elements of the Hadoop ecosystem



MapReduce lecture structure

- A bit of history
- Architecture
- Processes → map & reduce + job execution
- Scheduling & sync
- Monitoring & fault tolerance



Tom White, Hadoop – The definitive guide, O'Reilly, 4th Edition, 2015

HISTORY



The beginnings...

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* op-

- Built on the long history of distributed systems R&D
- Regarded as the first truly ‘big data’ platform capable to tackle data analyses on the web-scale
- Developed at Google
 - Originally proprietary Google technology
 - Open-sourced later
- Operational in Google as early as 2003
- MapReduce paper published in 2004

Dean, J., & Ghemawat, S. (2004).
MapReduce: Simplified data processing on
large clusters.



Motivation

- **Data processing is hard** → it is always challenging to extract useful information from data
- **Scalability is harder** → it is especially difficult to extract useful information from large-scale (e.g. web scale) data, e.g. crawled content from the web for Google searches
- **Fault-tolerance is even harder** → it is even more challenging to extract useful information from large-scale data in a fault-tolerant & correct way on commodity hardware, i.e. regular, low-cost server hardware
- The developers of MapReduce were motivated to solve the scalability & fault-tolerance challenges → ease large-scale data analysis for generations of data engineers & scientists
 - Developers can focus on data processing instead of solving above distributed system challenges

Additional reading

The screenshot shows a Microsoft Edge browser window with the following details:

- Title Bar:** History of massive-scale sorting experiments at Google
- Address Bar:** cloud.google.com/blog/products/gcp/history-of-massive-scale-sorting-experiments-at-google
- Page Content:**
 - Header:** Google Cloud Blog (Blog, Latest Stories, Product News, Topics)
 - Section:** DATA ANALYTICS
 - Post Title:** History of massive-scale sorting experiments at Google
 - Author:** Marián Dvorský, Software Engineer, Google Cloud Platform
 - Date:** February 18, 2016
 - Text:** We've tested MapReduce by sorting large amounts of random data ever since we created the tool. We like sorting, because it's easy to generate an arbitrary amount of data, and it's easy to validate that the output is correct.
 - Text:** Even the [original MapReduce paper](#) reports a TeraSort result. Engineers run 1TB or 10TB sorts as regression tests on a regular basis, because obscure bugs tend to be more visible on a large scale. However, the real fun begins when we increase the scale even further. In this post I'll talk about our experience with some petabyte-scale sorting experiments we did a few years ago, including what we believe to be the largest MapReduce job ever: a 50PB sort.
 - Text:** These days, GraySort is the large scale sorting benchmark of choice. In GraySort, you must sort at least 100TB of data (as 100-byte records with the first 10 bytes being the key), lexicographically, as fast as possible. The site [sortbenchmark.org](#) tracks official winners for this benchmark. We never entered the official competition.
- Right Side:** Social sharing icons for Facebook, Twitter, LinkedIn, and Email.
- Bottom:** Windows taskbar with search bar, pinned icons for File Explorer, Task View, File, Mail, Photos, OneDrive, Excel, and Paint, and system status indicators.

<https://cloud.google.com/blog/products/gcp/history-of-massive-scale-sorting-experiments-at-google>

ARCHITECTURE

Introduction



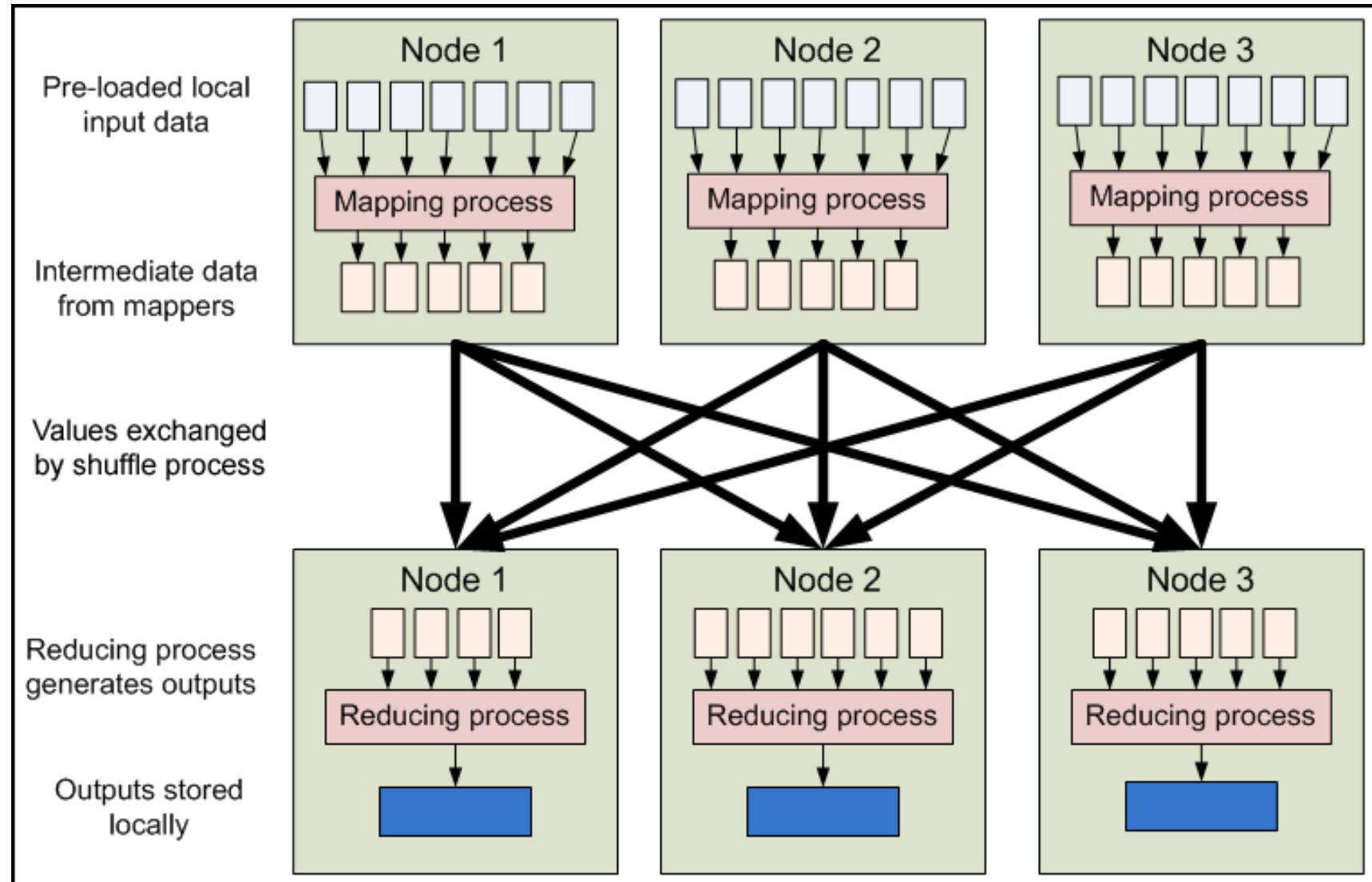
Definitions

- **DEF:** MapReduce is a programming model for efficient distributed computing
 - Presented in 2004 in a paper authored by Google employees
- Each node processes the data that is stored at that node
- Consists of two main phases: (1) Map & (2) Reduce

2-stage data processing

- Map
 - Read (large-scale) key-value data from a distributed filesystem
 - Transform and output different key-value pairs
- Reduce
 - Called once for each unique key
 - The reducer outputs zero or more final key/value pairs

MapReduce dataflow



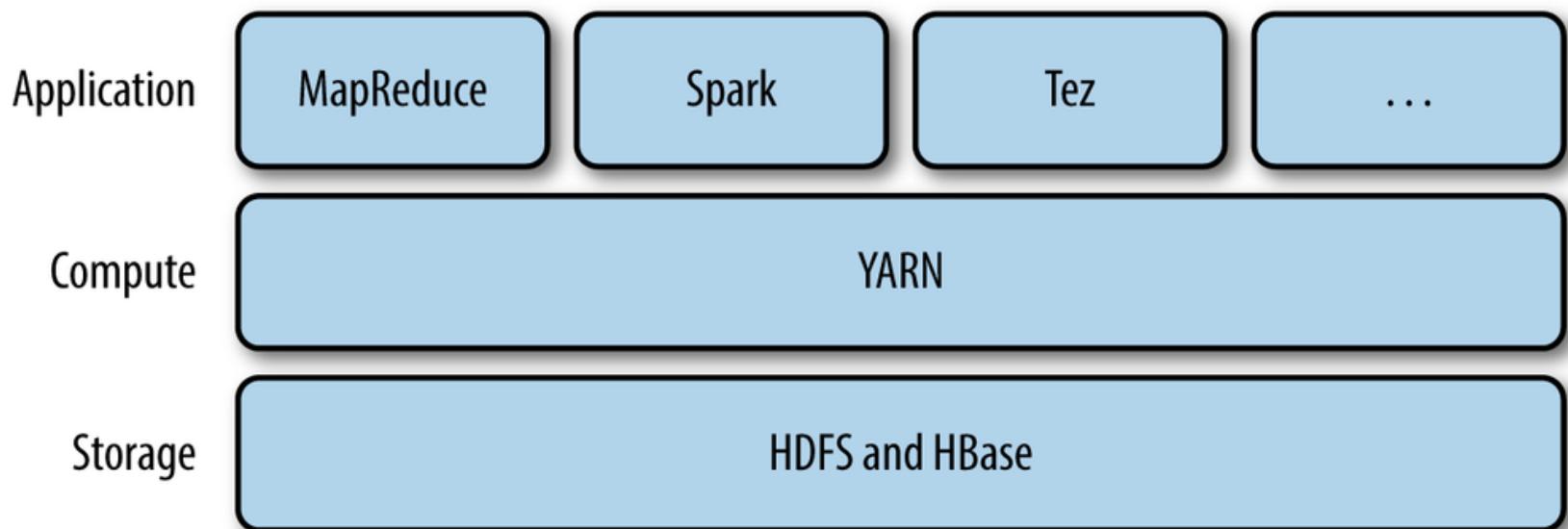
Input & output data types

- MapReduce inputs are large-scale data consisting of (K, V) key-value pairs
- A split is a chunk of input data processed by a single map process
 - Entries inside an input split are ‘records’
 - MapReduce is optimized for large-sized inputs
- Common file input/output formats
 - **Text:** lines, key-value pairs, XML
 - **Binary:** sequences of binary key-value pairs
 - **Database input:** read data from an RDBMS using JDBC
 - **Multiple inputs:** common solution for multiple proprietary versions of the same data source

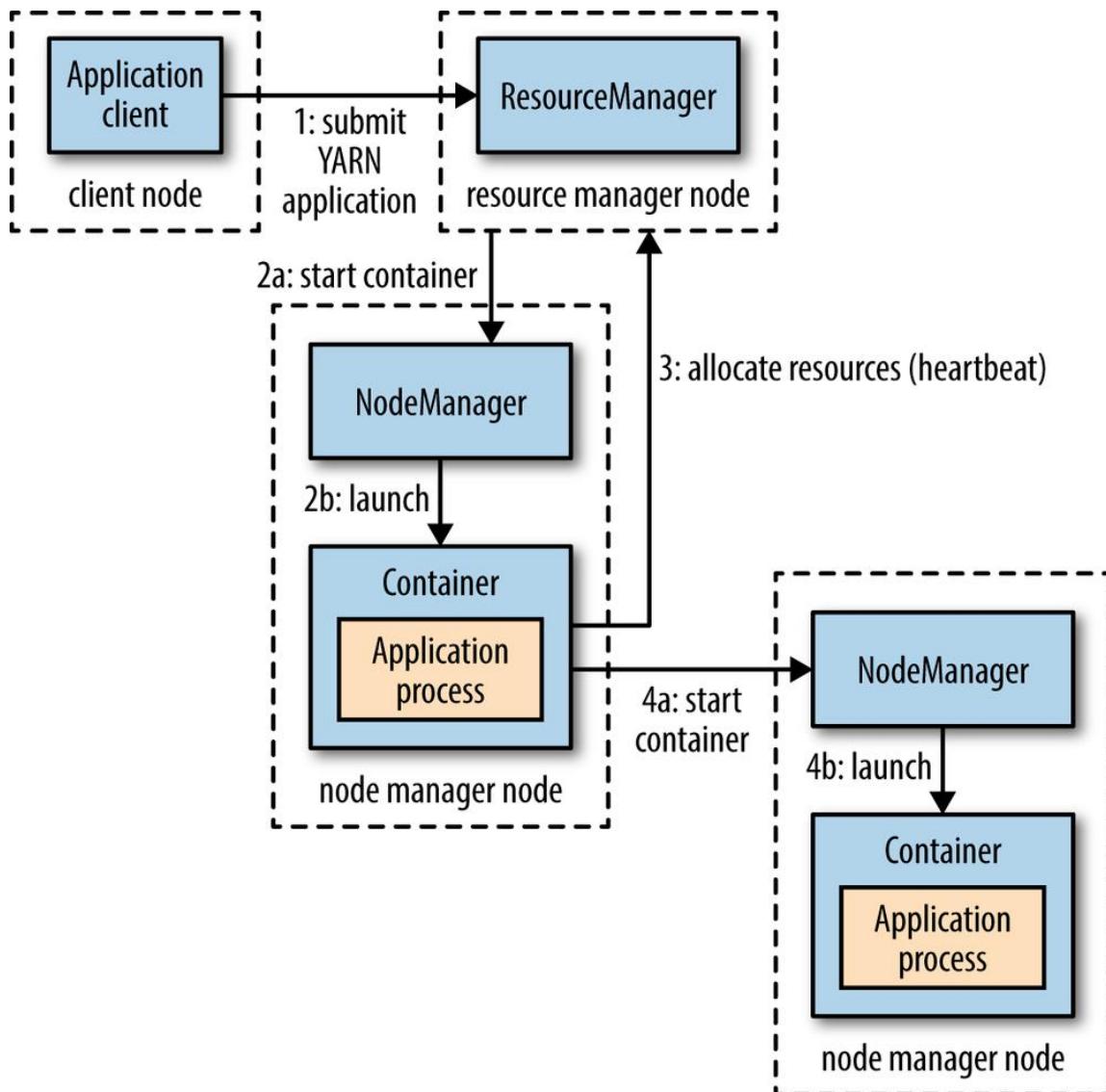


YARN

- **DEF:** Yet Another Resource Negotiator (YARN) is Hadoop's cluster resource management system
- Introduced in Hadoop 2
- Supports other distributed computing paradigms



YARN application execution



PROCESSES #1: MAP & REDUCE



Applications, jobs and tasks

Job-related definitions

- **Applications** submit **jobs**
- The job's input dataset is organized into (input) **splits**
- Each split is assigned a map **task**
 - Tasks are executed 'near' the (input data) split
 - Tasks execute in parallel

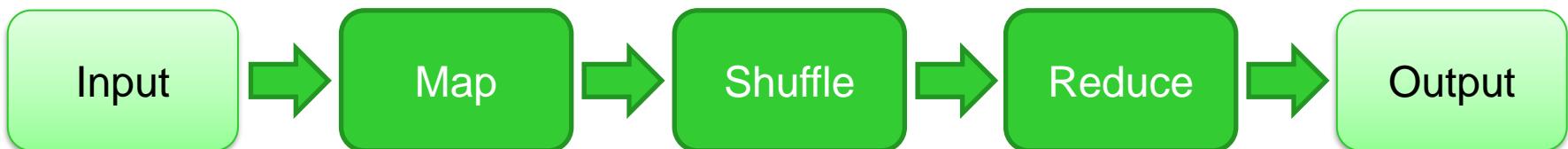
Job lifecycle

1. Job submission
 - Compute input splits
 - Copy resources needed to run
2. Job initialization
3. Task assignment
4. Task execution
5. Progress monitoring
6. Job completion



Data processing phases

- I/O: Key-value (K_1, V_1) input is loaded and split (usually HDFS)
- Process: The **map** process transforms (K_1, V_1) and generates intermediate results on the local disk
- Process: The **shuffle** process sorts, copies and merges the intermediate outputs on the reduce compute nodes
- Process: The **reduce** process transforms the outputs of the shuffle phase
- I/O: The reduce phase writes the outputs to storage (usually HDFS)





Map

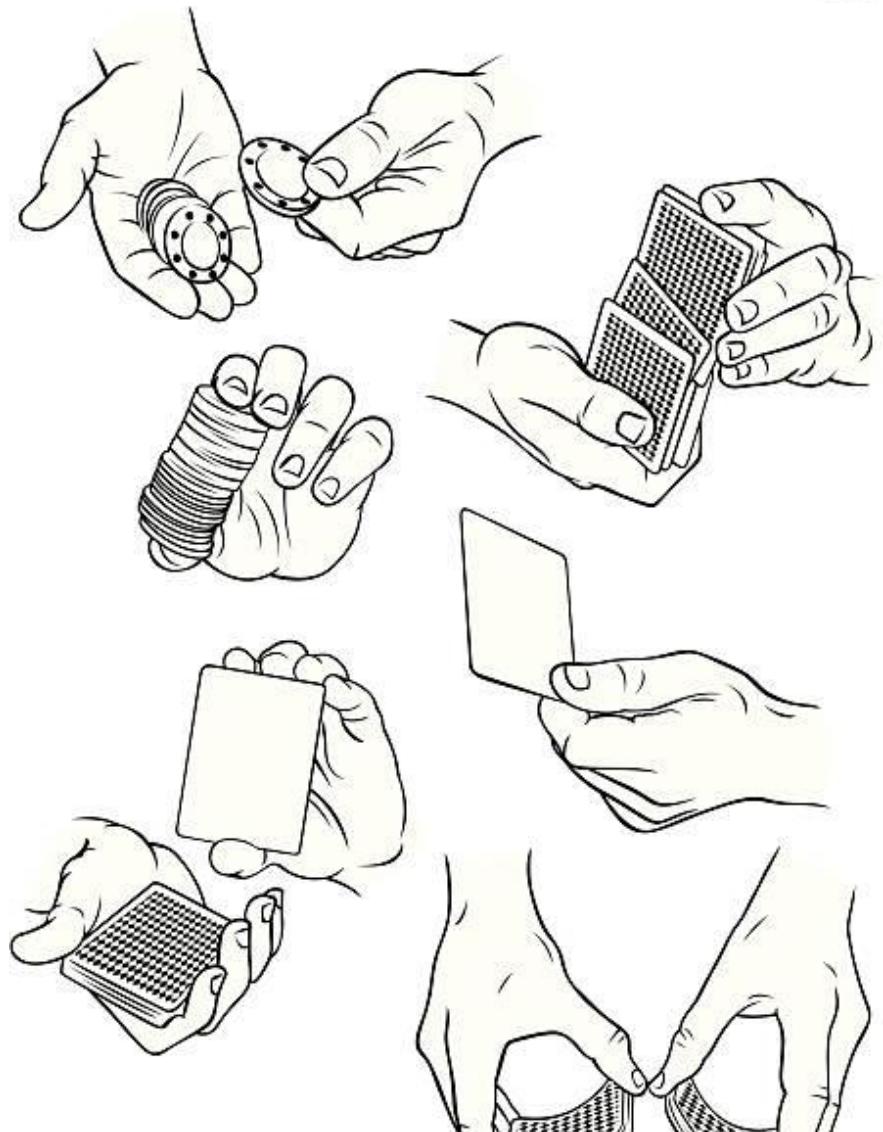
- Map transformation:

$$map(k, v) \rightarrow list(k_1, v_1)$$

- **Implemented by:** the user submitting the job(s)
- **Input:** Map is performed on a(n input) split of the dataset
 - Large datasets are usually stored in a distributed filesystem, e.g. HDFS
- **Execution:** run on multiple compute nodes with data locality optimization
- **Output:** Intermediate values are persisted only in local storage only → does not consume valuable cluster (bandwidth) resources
 - Spill to disk → when local (output) buffer is full, its contents are persisted to (local) disk
 - Map outputs can be compressed (ZIP) for optimized spill
 - Map outputs are reported via the heartbeat to the Application Master

Shuffle

- **Goal:** collect intermediate map outputs, sort by key and prepare for the reduce tasks
- **Implemented by:** the computing cluster itself, i.e. automatically executed and the client submitting the job is not burdened with the implementation of this phase
- **Input:** intermediate outputs generated by map tasks and stored locally
- **Output:** sorted key-value pairs collected on the compute nodes running the reduce tasks





Reduce

- Reduce transformation:

$$\text{reduce}(k_1, \text{list}(v_1)) \rightarrow (k_2, v_2)$$

- **Implemented by:** the user submitting the job(s)
- **Input:** collected by the shuffle phase, sorted and prepared for the reduce phase
 - Reduce executes multiple threads which fetch map outputs → reduce queries the Application Master for map process status
 - Inputs copied to memory if they fit, otherwise to disk
- **Execution:** one or more reduce tasks are executed to produce the output key-value pairs
 - Usually one reduce process per key k_1 in large-scale data
- **Output:** Reduce outputs key-value pairs (k_2, v_2)
 - Reduce outputs are persisted in persistent storage (HDFS)
 - Input & output keys (i.e. k_1 and k_2) might not be the same

Map & reduce in action

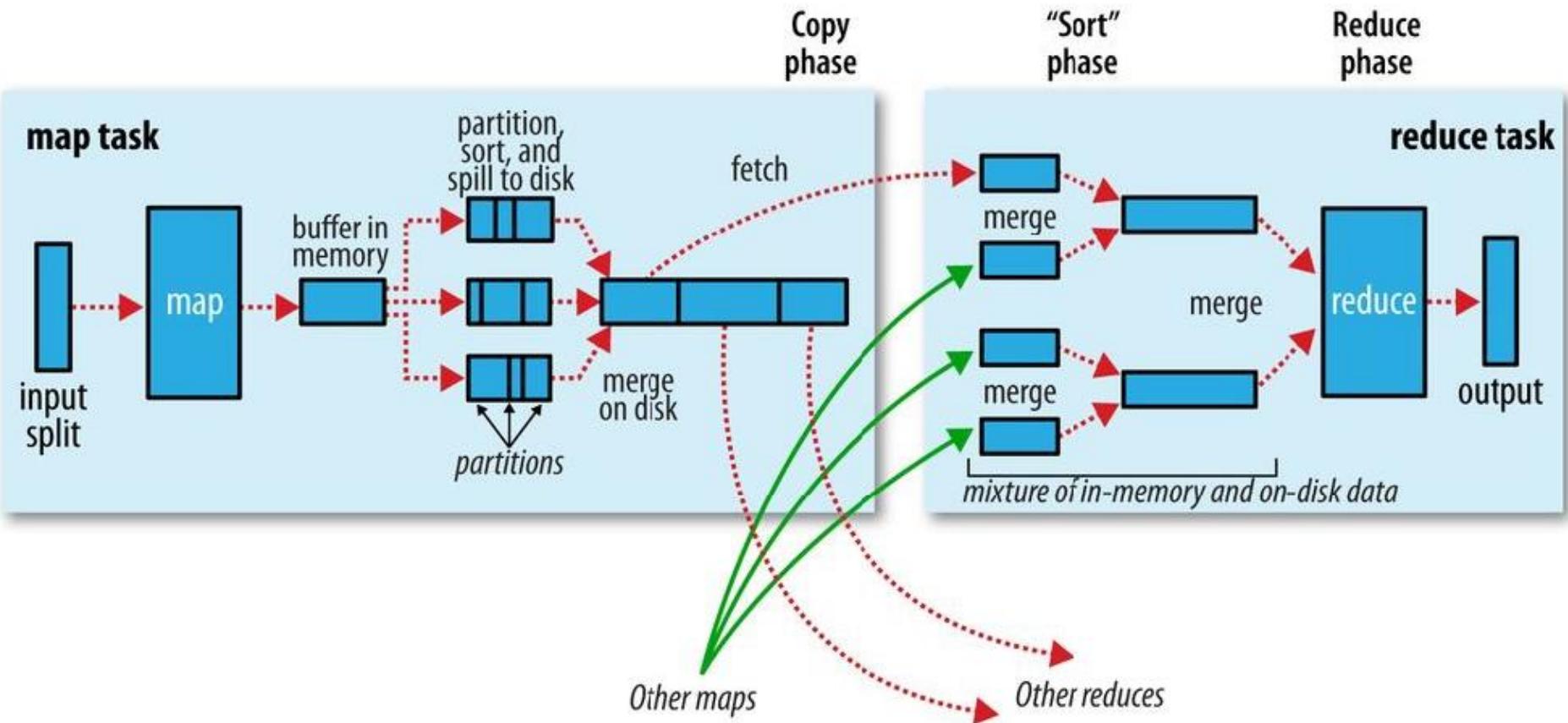


Figure 7-4. Shuffle and sort in MapReduce

Data-centered view

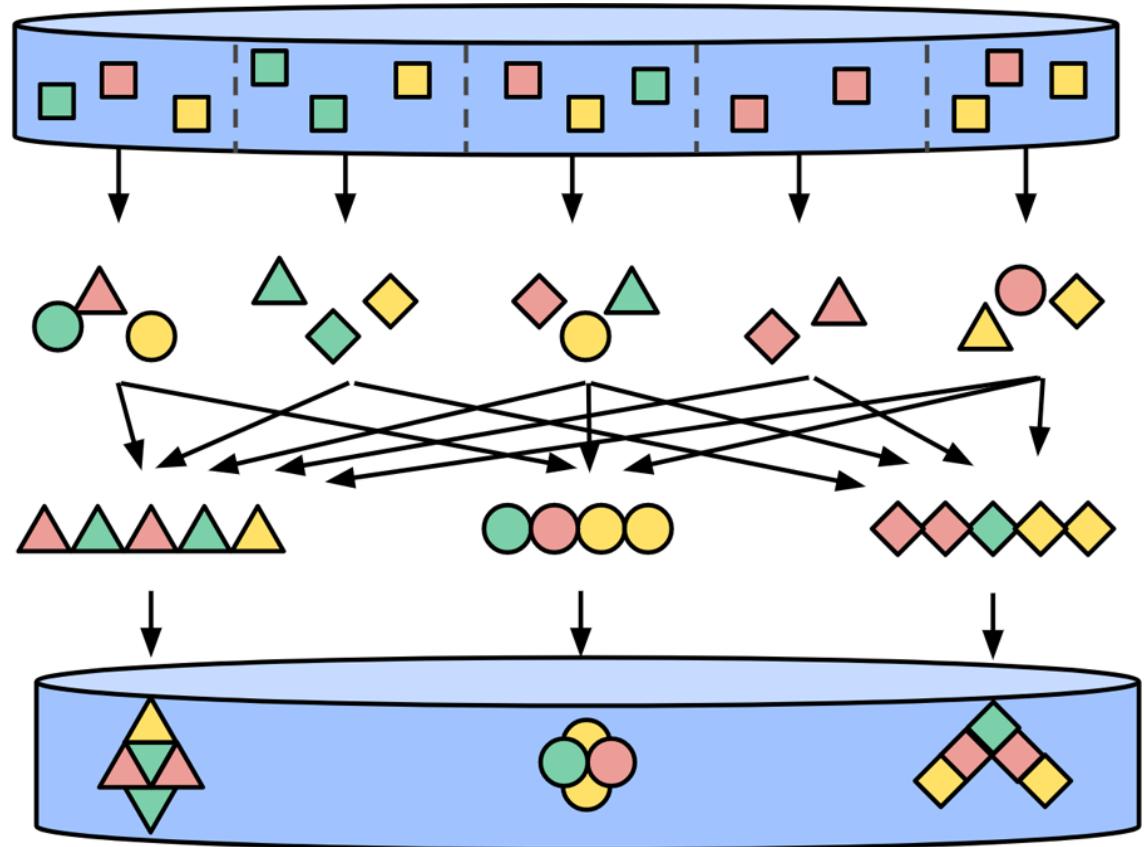
(Prepare)

Map

(Shuffle)

Reduce

(Produce)



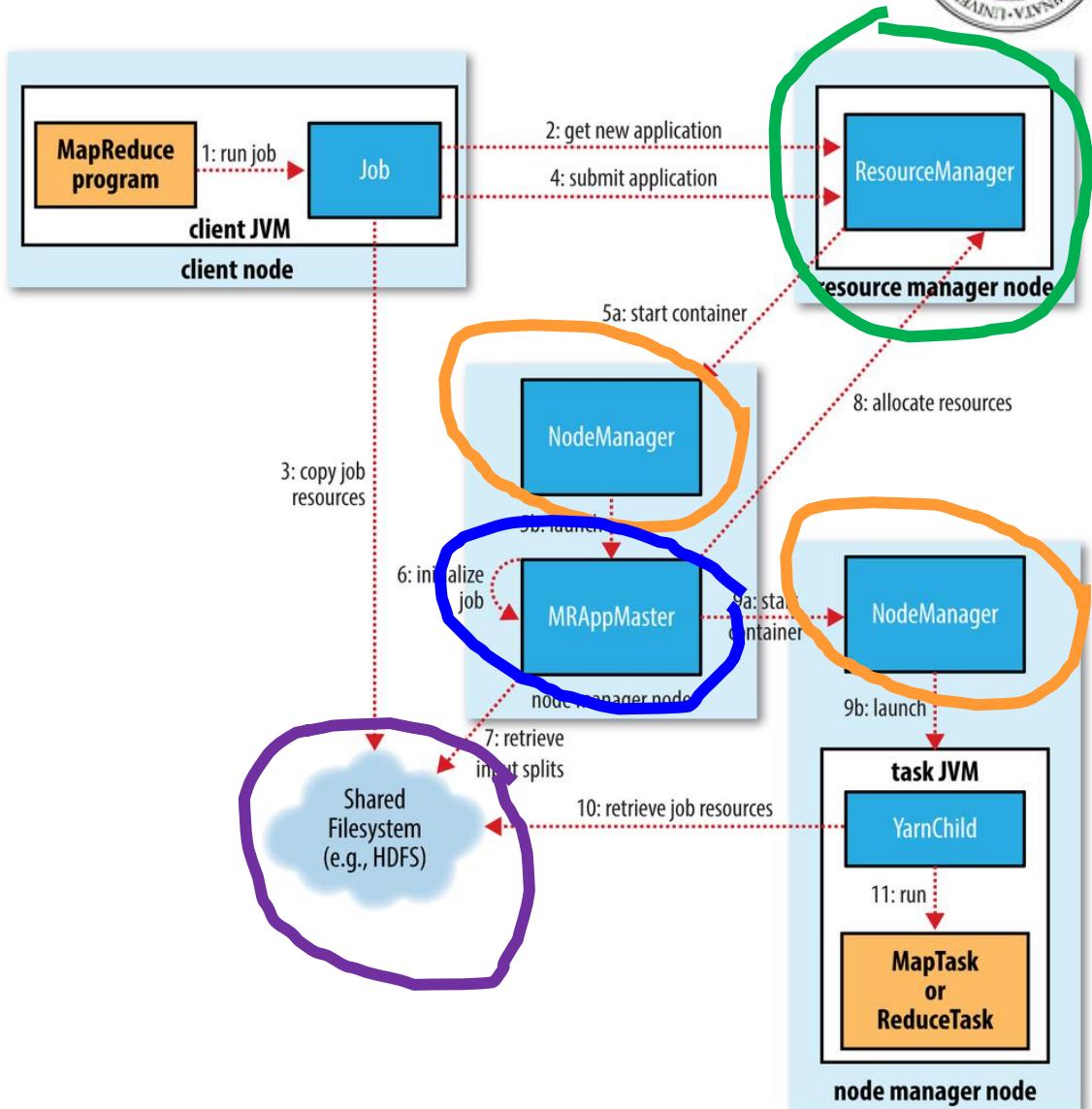
Akidau T., Chernyak S., Lax R. Streaming Systems: The What, Where, When, and how of Large-scale Data Processing, O'Reilly Media, 2018.

PROCESSES #2: JOB EXECUTION

Processes in job execution



1. Client submits a job
2. Resource manager
3. Node manager
4. Application master
5. Distributed filesystem, i.e. HDFS discussed earlier





Resource Manager

- The (YARN) **Resource Manager (RM)** coordinates the allocation of compute resources in the cluster
 - The RM is a long-running process → its lifecycle can be as long as complete cluster uptime
 - The RM has two main components: Scheduler and Application Manager
- The **Scheduler** allocates resources to running jobs
 - Does not monitor or track the progress of jobs
 - Does not offer guarantees in case of failures
- The **Application Manager** accepts job submissions
 - Assigns the 1st container to run the app-specific Application Master
 - Restarts the Application Master on failure

Node Manager



- Node Manager (NM) processes are executed on compute nodes, e.g. physical server computers in the computing cluster
- NMs report node status to the Resource Manager and/or Scheduler
 - Node resource usage (cpu, memory, disk, network)
 - Active containers/tasks

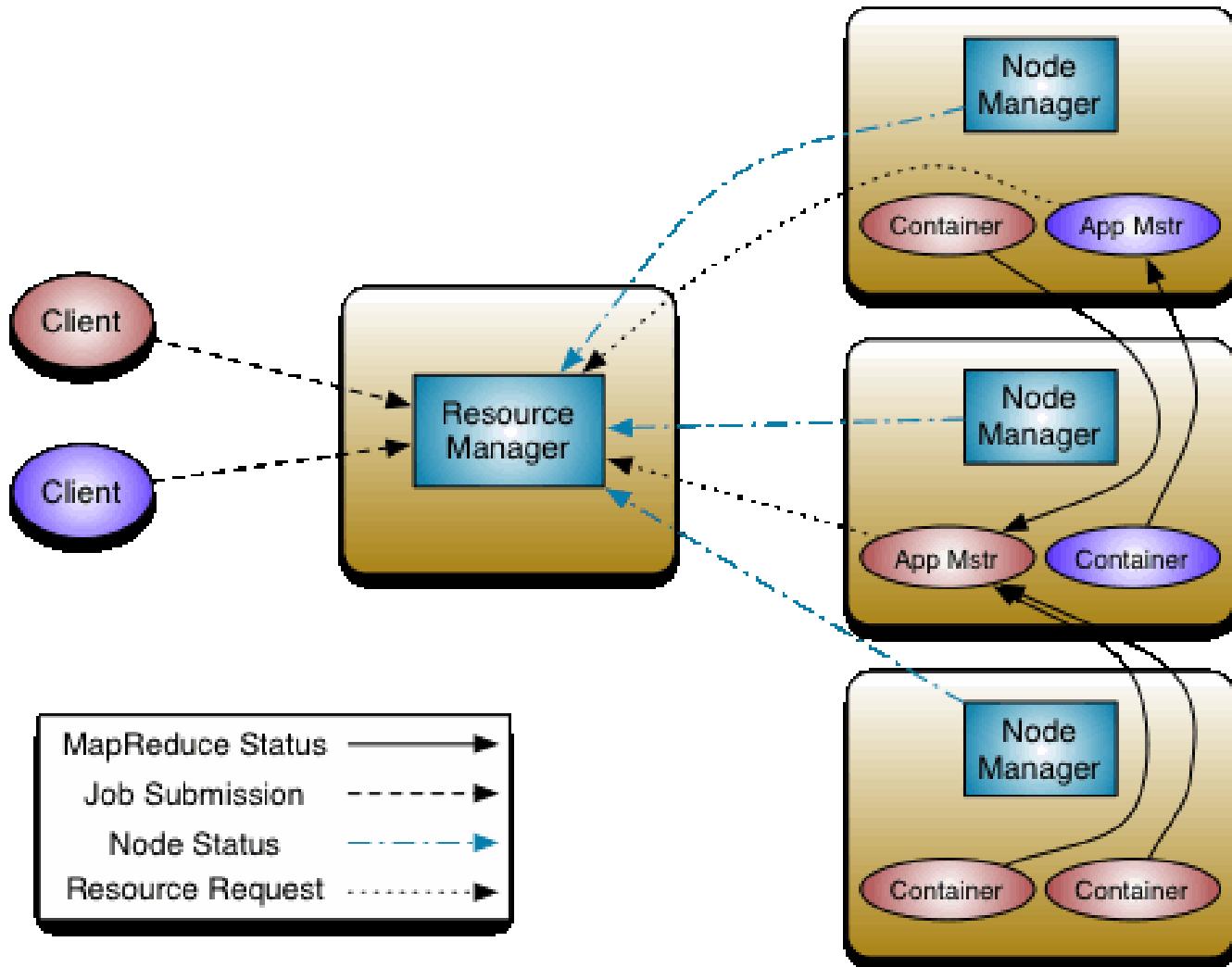
Application master

- A single Application Master (AM) is run **per application**
- Requests containers from the Scheduler
- Tracks the status of containers received and running the tasks of the current job, i.e. monitors container (task) progress



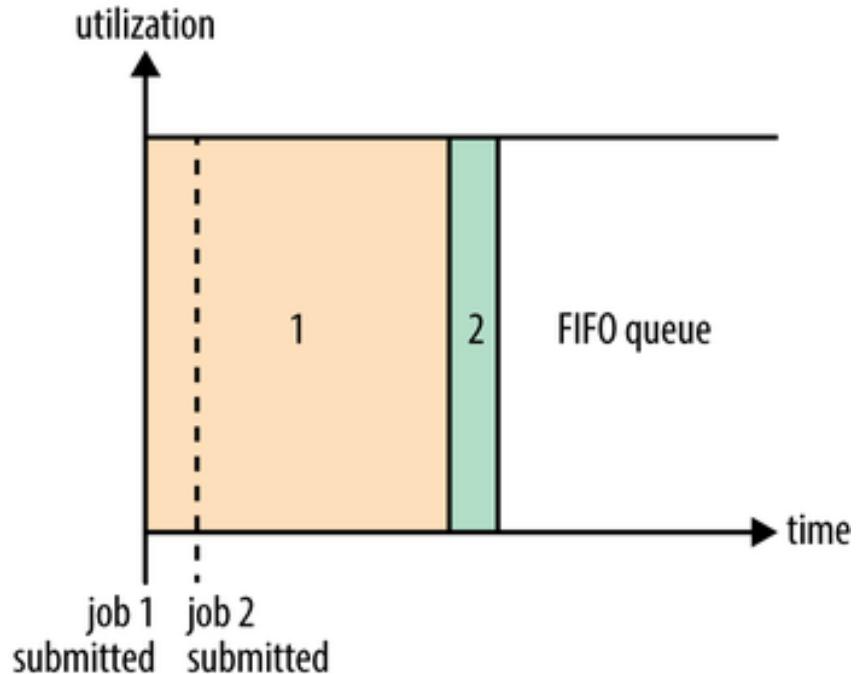
Note: Master Yoda is not the best fit, as he is the Grand Master in Star Wars, while the AM is only short-lived process

Processes in context



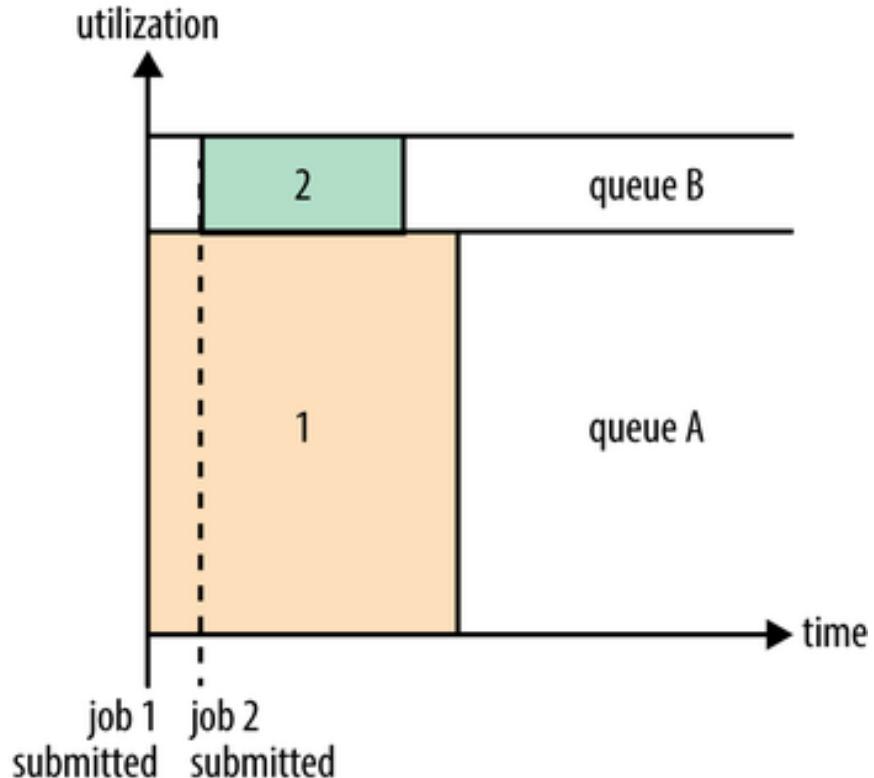
SYNCHRONIZATION & SCHEDULING

YARN scheduler: FIFO



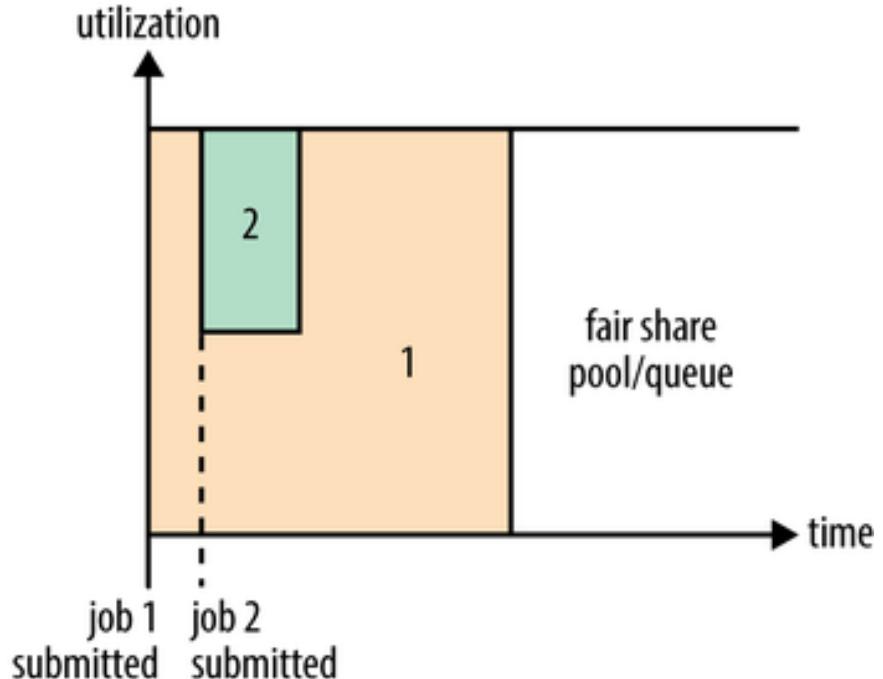
- First in, first out (FIFO)
- Simple (to understand and implement)
- Long-lasting jobs use 100% resources
- Other tasks wait until large, long-lasting jobs finish
- Not suitable for shared clusters with large numbers of jobs

YARN scheduler: Capacity



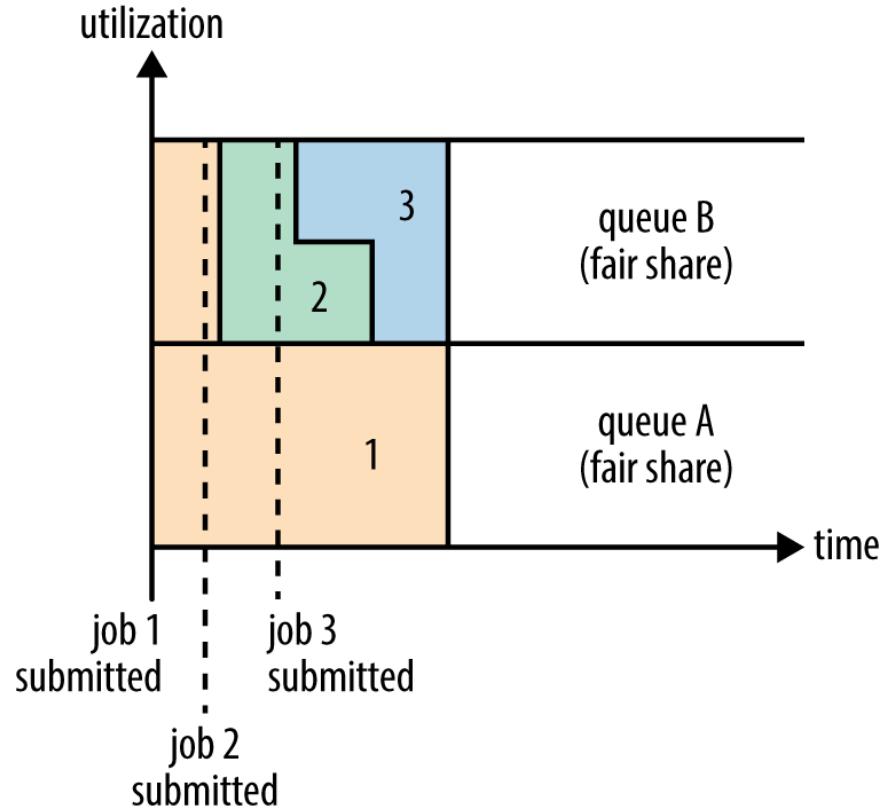
- The Capacity Scheduler allows cluster admin to configure additional queues
- The large job (1) finishes later compared to the FIFO Scheduler
- When job 2 finishes, the resources allocated to queue B are not freed → the cluster is not 100% utilized

YARN scheduler: Fair



- The Fair Scheduler does not rely on resource reservations
- Job 2 starts as soon as containers are freed by job 1
- Job 2 is allocated 50% of the cluster resources
- When job 2 finishes, the long-lasting job resumes to use 100% of cluster resources

YARN Fair Scheduler: 2 users



- **Users:** A, B
- **Queues:** A & B, each with up to 50% of cluster resources
- **Phase #1** – A single job by user A: At the start the large job 1 of user A uses 100% of cluster resources
- **Phase #2** – Additional jobs by user B: When the first job of user B is submitted, queue B is allocated 50% of cluster resources

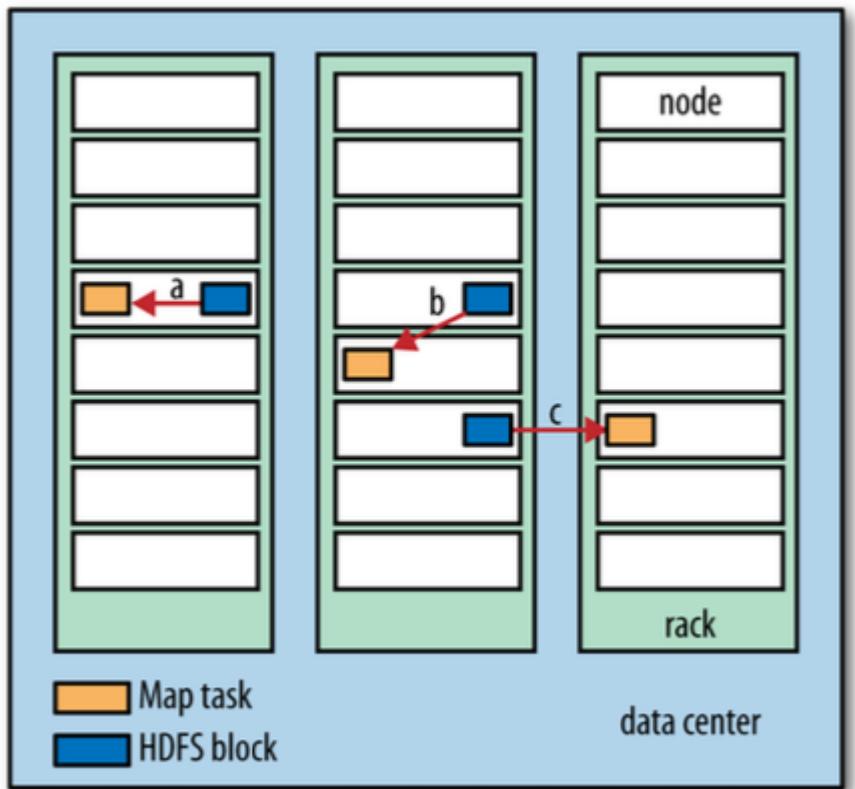


Preemption

1. Cluster utilization: 100%
 2. A job J1 is submitted to an empty queue
 3. J1 will wait for job-to-start time T1, i.e. until some containers complete their tasks
-
- **DEF:** Preemption is optional and allows the scheduler to kill tasks for queues which are running with more than their fair shares of cluster resources
 - **Pro:** preemption allows job-to-start times to be more predictable
 - **Contra:** preemption negatively impacts overall cluster efficiency, as the killed tasks are executed again

Data locality optimization

- **DEF: Data locality**
optimization aims to run (map) tasks on the same compute node where the input data (split) is located
- Levels of data locality in map task execution:
 - Data-local
 - Rack-local
 - Off-rack
- Data locality results in lower cluster bandwidth used





Delay scheduling

- Delay scheduling is done if the compute node requested and used to store the (input data) split is 100% busy with running other tasks
- Delay scheduling allows the resource manager to wait a short period of time before deciding to schedule the task on a different (compute) node in the same, or another rack
- **Pro:** Delay scheduling was shown to be superior to the resource manager immediately deciding to move the task to a different node and/or rack
 - Less cluster bandwidth is used for copying the split
- **Contra:** there is a slight delay in task execution



Dominant resource fairness

- Different MapReduce jobs/tasks can have different CPU and memory requirements
 - Job A might have high CPU and low memory load, while job B might have both high CPU and memory
- Dominant Resource Fairness (DRF) allows the resource scheduler to allocate resources based on dominant resource use (measured as a % of cluster capacity)
- DRF example:
 - Cluster capacity: 100 CPUs, 10 TB memory
 - Task A: 4 CPUs, 500 GB memory
 - Task B: 8 CPUs, 300 GB memory
 - Dominant resource: ? $4+8=12 \rightarrow 12\%$, $500+300=800 \rightarrow 8\%$
 - Amount of CPUs/memory allocated to A and B: ?
Task A gets 2x more CPUs than B

MONITORING & CONTROL



Progress & status updates

- MapReduce jobs can take from tens of seconds to hours to complete
- Each job and its tasks have **status** (running, completed, failed)
- Jobs also have **counters** of two varieties:
 - Built in counters, e.g. number of records written
 - User-defined counters
- User code can set the **status message or description**
- Map and reduce tasks measure **task progress** as
 - Map task progress is measured as the amount (%) of input processed
 - Reduce task progress is more challenging to measure → estimated proportion (%) of reduce input processed
- Tasks report task progress to the Application Master

Progress in MapReduce

- **Read** an input record
(map/reduce)
- **Write** an output record
(map/reduce)
- **Set status description** (from
inside the user's
map/reduce code)
- **Increment a counter**



https://www.vhv.rs/viewpic/hJoToxb_funny-work-in-progress-clipart-png-download-work/



Built-in job counters (selection)

Counter	Description
Launched map tasks	Number of map tasks launched, inclusive of speculative execution tasks
Launched reduce tasks	Number of reduce tasks launched
Failed map tasks	Number of map tasks failed
Failed reduce tasks	Number of killed reduce tasks
Killed map tasks	Number of killed map tasks
Killed reduce tasks	Number of killed reduce tasks
Data-local map tasks	Number of tasks ran on the same node with split*
Rack-local map tasks	Number of tasks ran in the same rack with split*
Other-local map tasks	Number of tasks ran in other racks*
Total time in map tasks	The total time taken to run map tasks*

* Note: the same counters exist for reduce tasks as well !



Built-in task counters (selection)

Counter	Description
Map input records	Number of records consumed by map tasks*
Map output records	Number of output records produced*
Split raw bytes	Number of input split bytes read by map tasks
Spilled records	Number of records spilled to disk
Reduce input groups	Number of distinct key groups consumed by reduce tasks
CPU milliseconds	Cumulative CPU time for a task

* Note: the same counters exist for reduce tasks as well !

Speculative execution



- Speculative execution (SE) → create a duplicate task
- SE is done when the RM detects slow-running tasks
- SE re-runs the same task on a different node → duplicate
 - The first to finish task's outputs are consumed
 - The other (late-comer) task is killed

FAULT TOLERANCE

Fault intro



- Errors and faults can occur at multiple locations
- We discuss the following fault locations
 - Tasks
 - Node Manager
 - Application Master
 - Resource Manager
- **Note:** we are not discussing HDFS failures

Task failure

- Both map and reduce tasks might fail, usually due to a software bug
- **Caught exceptions** are reported to the Application Master by the JVM → AM marks task failed and frees container resources
- **Hanging tasks** are noticed by the AM based on delayed heartbeats → AM kills hanging tasks after a default timeout of 10 min
- Failed tasks are **re-scheduled** on different nodes up to 4 times → # failures > 4 → job fails
- **Partial job success** is allowed if only a configured amount of tasks fails





Application Master failure



- The Application Master (AM) sends **periodic heartbeats** to the Resource Manager (RM)
- The **client polls** the AM for status updates
- When RM detects AM failure, it creates a **new AM process** in a different container
 - The running AM persists job history & task progress
 - New AM re-constructs job history → not all tasks re-run
- YARN **retries** failed Application Masters
 - Default retries: 2

https://www.123rf.com/photo_49076207_the-3d-guy-and-a-sign-of-error.html

Node manager failure

- Node Manager (NM) instances send **heartbeats** to the Resource Manager (RM) → detect failure/slowdown
- Tasks belonging to incomplete jobs on the failed node are re-run on different nodes
 - Intermediate outputs might not be accessible
- An Application Manager (AM) might **blacklist nodes** with high failure counts



<https://www.dreamstime.com/stock-illustration-error-d-people-upset-metaphor-image43976249>

Resource Manager failure



- The active Resource Manager (RM) stores running application information in stable storage
 - Note: Node Manager state is not stored → it is re-constructed based on NM heartbeats
- When the RM fails, no jobs and tasks can run
- A pair of **active-standby** RMs are run
- When a new RM is run, it reads **application state** from stable storage and restarts the Application Manager processes

https://www.123rf.com/photo_76996200_3d-man-presses-the-button-with-error-symbol-.html

MapReduce Summary

- A bit of history
- Architecture
- Processes
 - Map & reduce
 - Job execution
- Synchronization & scheduling
- Replication
- Monitoring
- Fault-tolerance



Thank you for your attention!





DATA ANALYSIS SOLUTIONS: SPARK

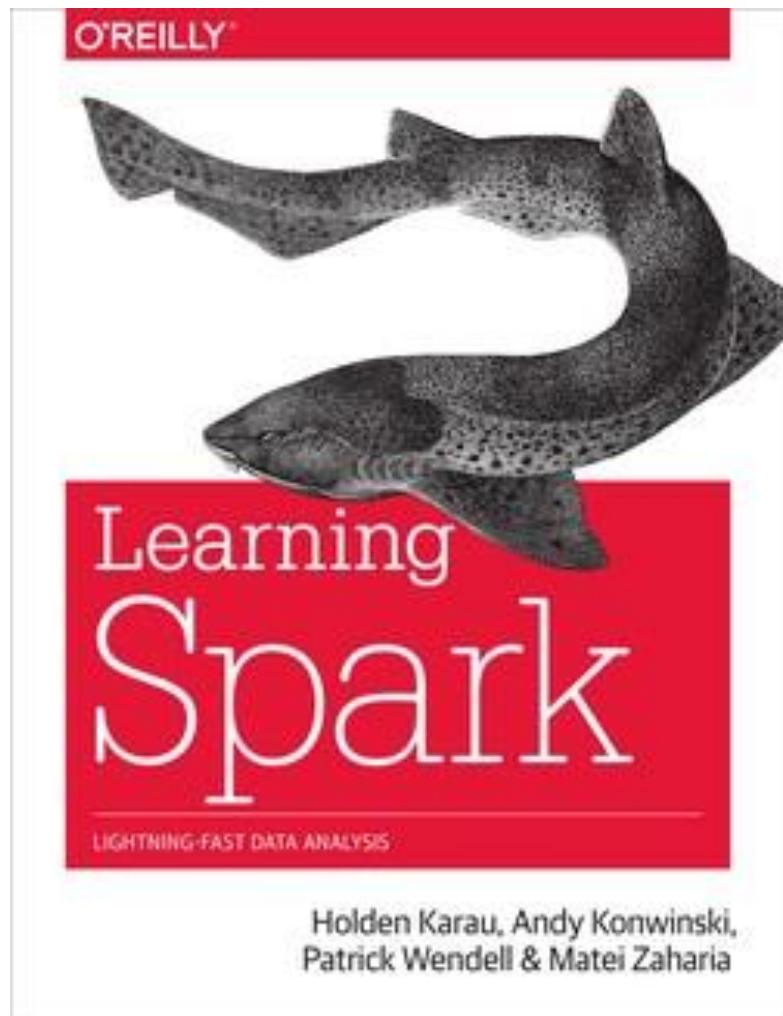
*Open-source Technologies for Real-Time Data
Analytics*

Imre Lendák, PhD, Associate Professor



Chosen data analytics topics

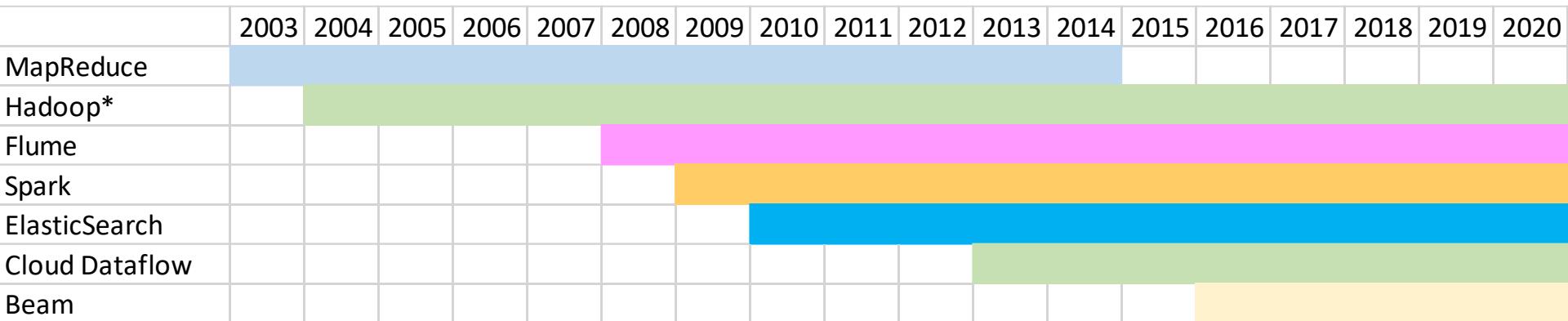
- **MapReduce** implements the map & reduce paradigm known from functional programming
 - Discussed in last time
- **Apache Spark** is an open-source, distributed, general-purpose cluster-computing framework
 - Discussed in this lecture!
- **ElasticSearch** search & analytics engine
 - Discussed later



Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia, "Learning Spark: Lightning-Fast Big Data Analysis", O'Reilly, 2015.



Data analysis timeline



* Analysis elements of the Hadoop ecosystem



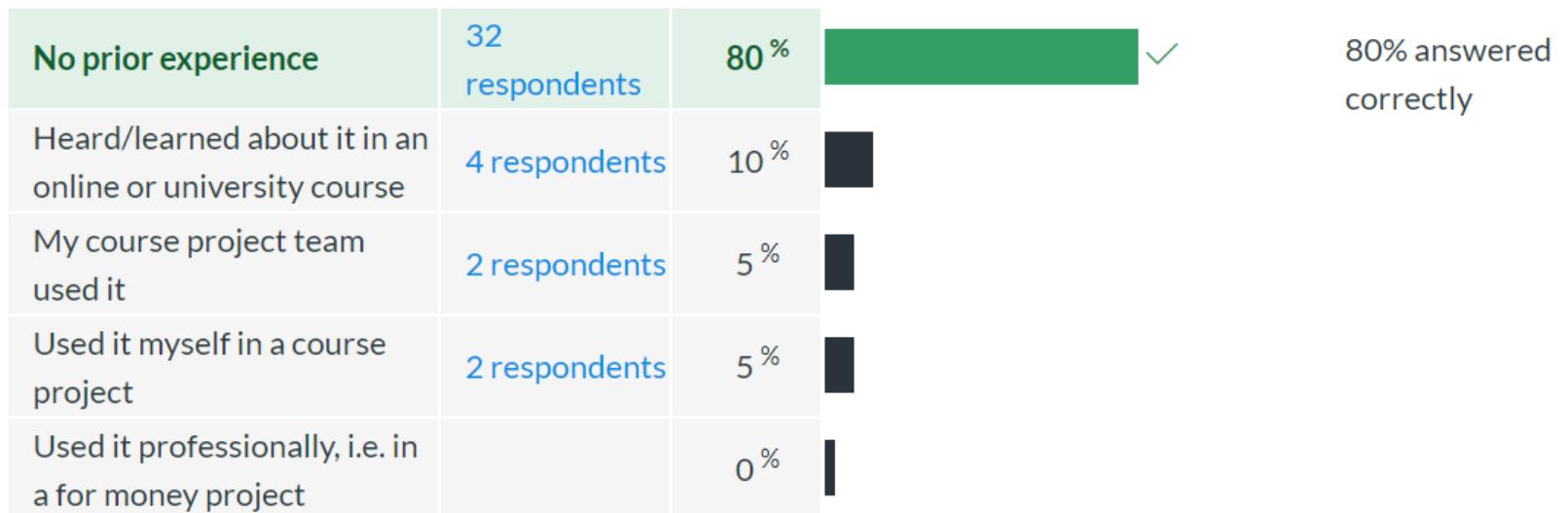
Survey: Spark (Streaming) XP

Attempts: 40 out of 40

+0.59

Discrimination
Index ⓘ

Please rate your past experience in using Spark Streaming:



SPARK INTRO



Introduction & history

Definitions

- **DEF:** Apache Spark is an open-source, distributed, general-purpose **cluster-computing framework**
- The authors aimed to perform **in-memory** calculations in computing clusters without ‘touching the disk’ before reaching the final data processing stage (i.e. output)
- Written in Scala
- Additionally optimized for interactive queries and iterative computing jobs

History

- Originally developed by the AMPLab at UC Berkeley around 2009
- As soon as 2009 it was outperforming MapReduce 10-20x in certain types of problems
- Open sourced in 2010 (BSD license)
- Donated to the Apache Software Foundation in 2013
- Top-level Apache project since 2014



Open-source permissions and limitations

- **Linking** - linking of the licensed code with code licensed under a different licence (e.g. when the code is provided as a library)
- **Distribution** - distribution of the code to third parties
- **Modification** - modification of the code by a licensee
- **Patent grant** - protection of licensees from patent claims made by code contributors regarding their contribution, and protection of contributors from patent claims made by licensees
- **Private use** - whether modification to the code must be shared with the community or may be used privately (e.g. internal use by a corporation)
- **Sublicensing** - whether modified code may be licensed under a different licence (for example a copyright) or must retain the same licence under which it was provided
- **TM grant** - use of trademarks associated with the licensed code or its contributors by a licensee



Open-source licenses compared

Licence	Author	Latest version	Publication date	Linking	Distribution	Modification	Patent grant	Private use	Sublicensing	TM grant
Academic Free License ^[11]	Lawrence E. Rosen	3.0	2002	Permissive	Permissive	Permissive	Yes	Yes	Permissive	No
Affero General Public License	Affero Inc	2.0	2007	Copylefted ^[12]	Copyleft except for the GNU AGPL ^[12]	Copyleft ^[12]	?	Yes ^[12]	?	?
Apache License	Apache Software Foundation	2.0	2004	Permissive ^[13]	Permissive ^[13]	Permissive ^[13]	Yes ^[13]	Yes ^[13]	Permissive ^[13]	No ^[13]
Apple Public Source License	Apple Computer	2.0	August 6, 2003	Permissive	?	Limited	?	?	?	?
Artistic License	Larry Wall	2.0	2000	With restrictions	With restrictions	With restrictions	No	Permissive	With restrictions	No
Beerware	Poul-Henning Kamp	42	1987	Permissive	Permissive	Permissive	No	Permissive	Permissive	No
BSD License	Regents of the University of California	3.0	?	Permissive ^[14]	Permissive ^[14]	Permissive ^[14]	Manually ^[14]	Yes ^[14]	Permissive ^[14]	Manually ^[14]



Whois AMPLab?

AMPLab

- AMP = Algorithms, Machines and People Lab
- Doing research and publishing scientific publications since 2008
- AMPLab officially launched in 2011
- Worked on different ‘big data’ projects under the Berkeley Data Analytics Stack (BDAS)
- UC Berkeley launched RISELab as the successor to AMPLab in 2017

Best known projects

- **Apache Spark** – distributed, general-purpose computing platform
- **Apache Mesos** – cluster management platform
- **Alluxio** – virtual distributed file system (VFDS) – Alluxio ‘sits’ between computation & storage in large-scale data processing environments. Used by Cray, IBM, Lenovo, Intel, etc.



RISELab in November 2020

 [HOME](#) [PEOPLE](#) [PROJECTS](#) [PUBLICATIONS](#) [SPONSORS](#) [DARE](#) [ACADEMICS](#) [NEWS](#) [EVENTS](#) [RISE CAMP](#) [BLOGS](#) [JENKINS](#) [LOGIN](#) [CONTACT US](#) [Q](#)

 An NSF Expedition Project

REAL-TIME INTELLIGENT SECURE EXPLAINABLE SYSTEMS

IN THE RISELAB, WE DEVELOP TECHNOLOGIES THAT ENABLE APPLICATIONS TO MAKE LOW-LATENCY DECISIONS ON LIVE DATA WITH STRONG SECURITY.



Current Founding Sponsors



ERICSSON

facebook



Google



Microsoft

NVIDIA



splunk>

vmware

<https://rise.cs.berkeley.edu>



Spark authors

- PhD students at UC Berkeley
 - Matei Zaharia – Spark founder, Databricks CTO and professor at Stanford
 - Benjamin Hindman (Mesos)
 - Andy Konwinski (Mesos, Spark)
 - Haoyuan Li (Alluxio)
- Uni-based research supervised by professor Ion Stoica
- Developers: Holden Karau, Patrick Wendell, Andy Konwinski, ...

Authors / Apache Spark

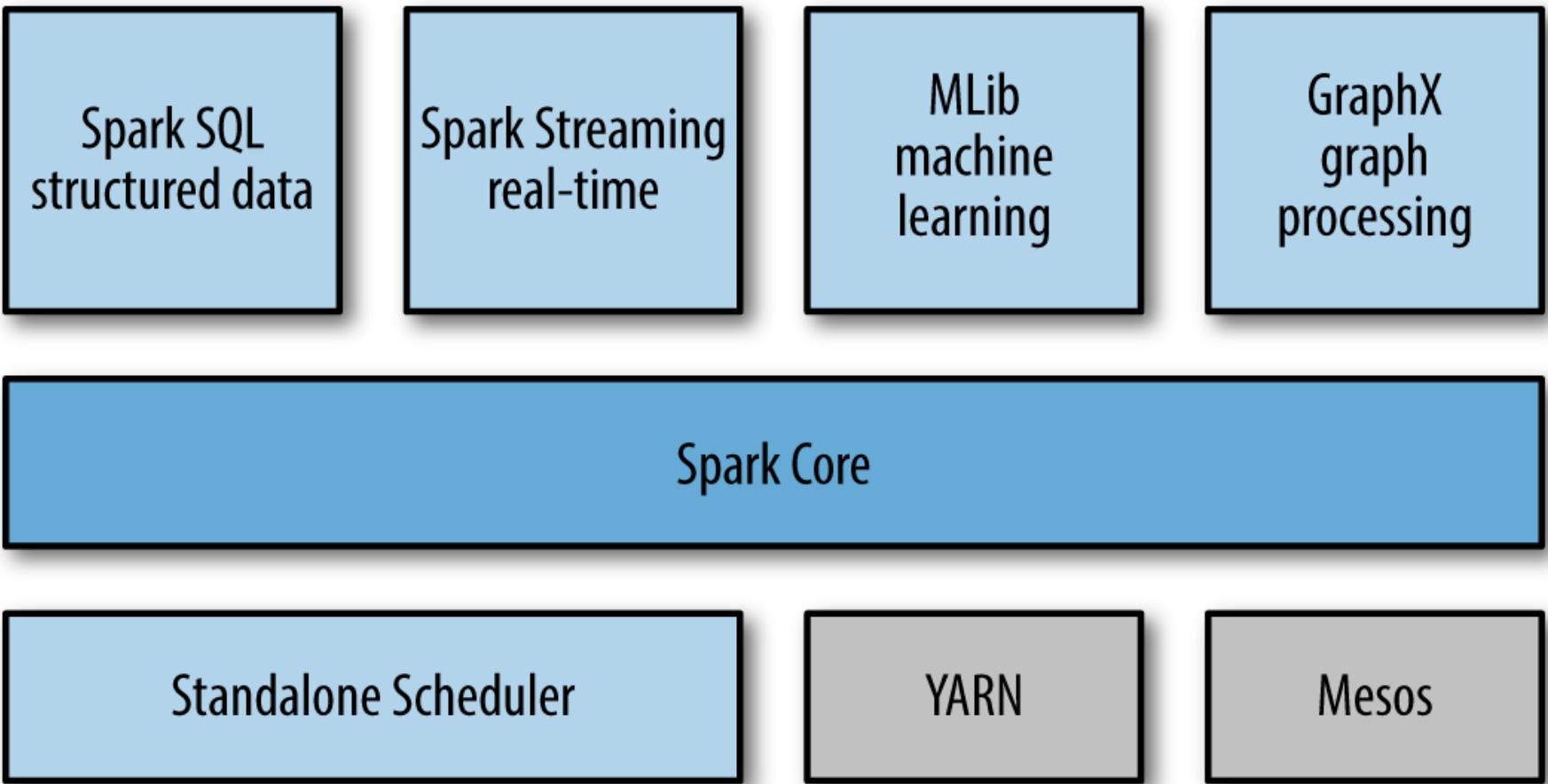


* Image source: www.google.com search

SPARCHITECTURE



The Spark stack



Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia, "Learning Spark: Lightning-Fast Big Data Analysis", O'Reilly, 2015.



Components

- **Spark Core** tasks:
 - Memory management
 - Fault recovery
 - Implements the RDD (v1.0) and Dataset (v2+) Application Programming Interfaces (RDD API vs Dataset API)
- **Spark SQL** is Spark's package for working with structured data.
- **Spark Streaming** is a consistent micro-batch processing environment for live streams of data.
- **Mlib** provides multiple types of machine learning algorithms, e.g. classification, regression, clustering
- **GraphX** is a library for manipulating graphs
- Cluster management via YARN, Mesos or Spark's own Standalon Scheduler

SPARCHITECTURE: RESILIENT DISTRIBUTED DATASET (RDD)



RDD intro

- **DEF:** Resilient Distributed Datasets (RDDs) are data items distributed over a cluster of machines and maintained in a fault-tolerant way
 - RDDs are essentially a restricted form of distributed shared memory
 - RDDs can contain different data types, not just (key, value) pairs as in MapReduce
 - Java objects are kept in memory deserialized
 - Python objects are ‘pickled’

Resilient Distributed Datasets (RDD)



Abstraction

- Partitioned collection of records
 - Data is spread across the cluster
 - RDDs are read-only, i.e. no in place updates
- Caching dataset in memory (if able)
 - different storage levels available
 - fallback to disk possible

Operations

- *Transformations* create new RDDs from existing RDDS
 - *map, filter, join*
 - Lazy operation
- *Actions* return a value to the Spark application or export data
 - Actions include *count, collect, etc.*
 - Triggers execution



Spark inputs & outputs

- Input/output **file formats**: text, JSON, CSV, sequence files & object files
 - File compression, e.g. gzip
 - Filesystems: local, HDFS, Amazon S3
- **Protocol buffers** are a fast, space-efficient multilanguage format
 - Originally developed at Google, open source, structured data, fields and types well-defined
- **RDBMS** accessed via JDBC
- **Distributed data stores**: Cassandra, HBase, Elasticsearch
- Note: **Broadcast variables** allow Spark applications to send shared, mid-size, static data to all worker nodes
 - The data for the broadcast variable is loaded from storage by the driver and sent out to all workers (e.g. lookup data)



Partitions

- In some Spark applications an RDD is scanned multiple times
 - In those cases it is useful to control the dataset's partitioning across the nodes
 - Partitioning is usually controlled with RDDs of key-value pairs
→ a set of keys are stored together on a node
- Operations benefiting from partitioning: various joins (e.g. with smaller lookup table), group by, reduce by, etc.
- Not all transformations set a partitioner for RDDs → e.g. general map calls might modify the keys → reset partitioner
- **Note:** after an explicit re-partitioning of an RDD it should be persisted as otherwise the RDD would be re-evaluated and repartitioned on each action

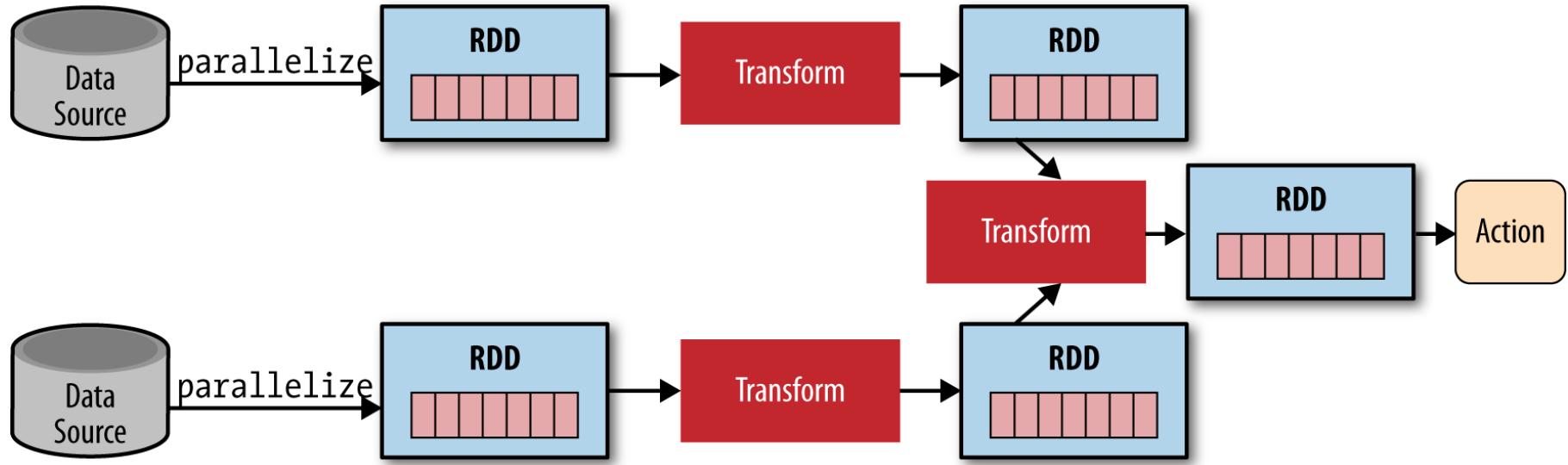
Storage levels and caching

- By default, Spark recomputes the RDD and its dependencies each time an action is called
- Spark can be told to persist an RDD, i.e. nodes which computed an RDD are asked to persist their partitions
- Off-heap caching can be implemented in the Alluxio data orchestrator, <https://www.alluxio.io>

Level	Space used	CPU time	In memory	On disk	Comments
MEMORY_ONLY	Hi	Lo	Yes	No	
MEMORY_ONLY_SER	Lo	Hi	Yes	No	
MEMORY_AND_DISK	Hi	Med	Some	Some	Spills to disk if insufficient memory
MEMORY_AND_DISK_SER	Lo	Hi	Some	Some	Serialized objects in memory, spills to disk
DISK_ONLY	Lo	Hi	No	Yes	

PROCESSES: DATA ANALYTICS

Data processing steps



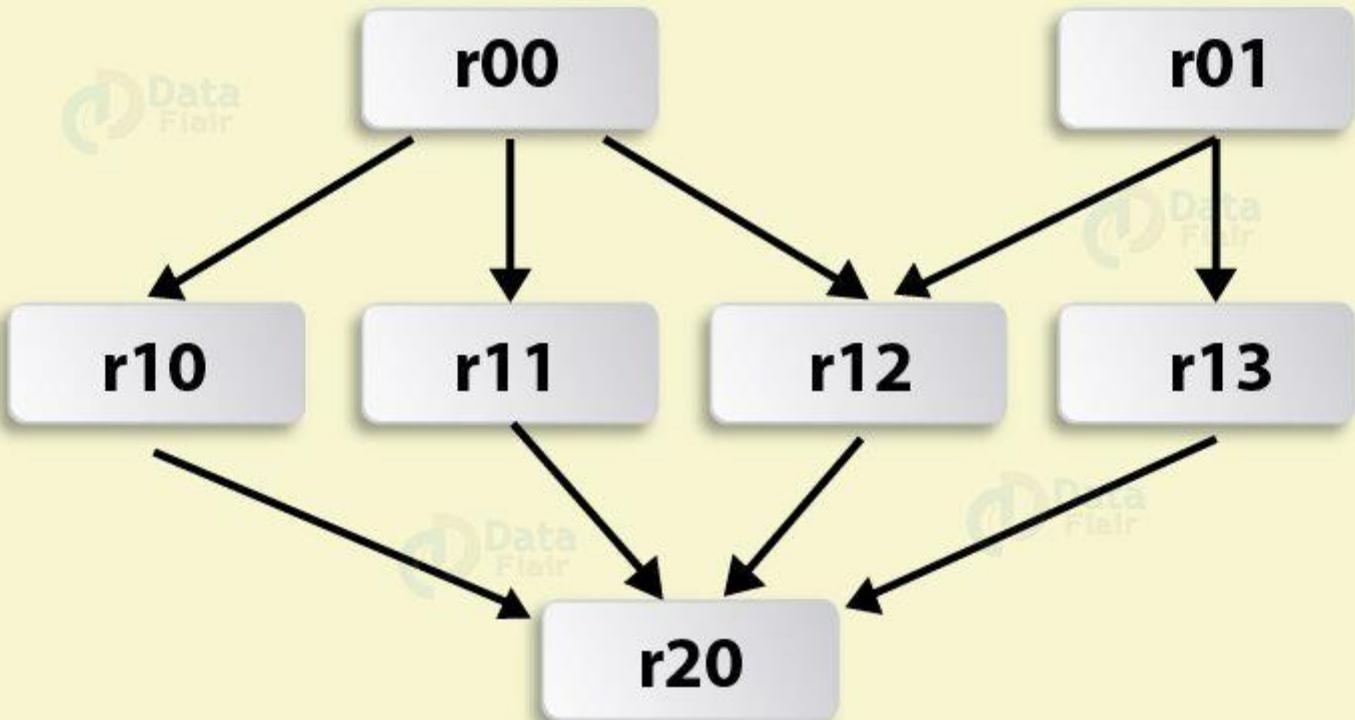
- Operation types on RDDs:
 - transformations and
 - actions



Transformations

- **DEF:** **Spark transformations** are operations which read RDDs as inputs and produce RDDs as outputs
 - Transformations do not mutate input RDDs → they just produce new output RDDs and return a pointer to it
 - Many, but not all transformations are element-wise, i.e. they operate on the elements of the input RDDs in sequence
 - Transformations can operate on one (e.g. filter()), two (e.g. union()) or more input RDDs
- **DEF:** The **Spark lineage graph** is the set of dependencies between RDDs
 - Lineage graphs are maintained for each Spark application separately
 - The lineage graph is used to re-compute RDDs on demand and to recover lost data if parts of a persisted RDD are lost
 - **Note:** be careful and do not confuse the lineage graph with the directed acyclic graph (DAG) of task execution

Example lineage graph





Element-wise transformations

- Most common element-wise transformations for an RDD containing [1,2,3,3]

Function name	Purpose	Example	Result
map()	Apply function to each element	rdd.map(x => x+1)	[2,3,4,4]
flatMap()	Apply function and return flat data	rdd.flatMap(x=>x.to(3))	[1,2,3,2,3,3,3]
filter()	Get RDD with elements filtered	rdd.filter(x=>x!=1)	[2,3,3]
distinct()	Remove duplicates	rdd.distinct()	[1,2,3]
sample(withReplacement, fraction, seed)	Select sample from an RDD w or w/o replacement	rdd.sample(false, 0.5)	? (non deterministic)



Pseudo-set transformations

- Most common element-wise transformations for RDDs containing {1, 2, 3} and {3, 4, 5}

Function name	Purpose	Example	Result
union()	Elements from both input RDDs	rdd.union(other)	{1, 2, 3, 3, 4, 5}
intersection()	Elements found in both RDDs	rdd.intersection(other)	{3}
subtract()	Remove contents of one RDD	rdd.subtract(other)	{1, 2}
cartesian()	Cartesian product	rdd.cartesian(other)	{(1, 3), (1, 4), ... (3,5)}

- The Cartesian product for sets A and B is denoted with $A \times B$. It is the set of all ordered pairs (a,b) where a is in A and b is B



Actions

- **DEF:** **Spark actions** are operations which make some calculation and return the result to the driver or persist it in external storage
- Actions force the evaluation of all (upstream) transformations in the lineage graph of the RDD they are called on
- Each different action forces the evaluation of upstream transformations unless the intermediate RDDs are persisted (which is not default behavior)
- The simplest actions are count(), take() and collect()
- **Note:** be careful when calling collect() as its return value should be limited in size and able to fit in the driver's memory → usually called during testing and/or when the transformations result in RDDs of limited size(s)



Basic Spark actions

- Basic actions on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
collect()	Retrieve all elements	rdd.collect()	{1, 2, 3, 3}
count()	Number of elements	rdd.count()	4
countByValue()	Number of each unique element	rdd.countByValue()	{(1, 1), (2, 1), (3, 2)}
take(num)	Return 'num' elements from the RDD	rdd.take(2)	{1, 2}
top(num)	Return 'num' top elements from RDD	rdd.top(2)	{3, 3}
reduce(func)	Combine RDD elements with function 'func'	rdd.reduce((x, y) => x + y)	9
fold(zero)(func)	Same as reduce, but with zero value	rdd.fold(0)((x, y) => x + y)	9



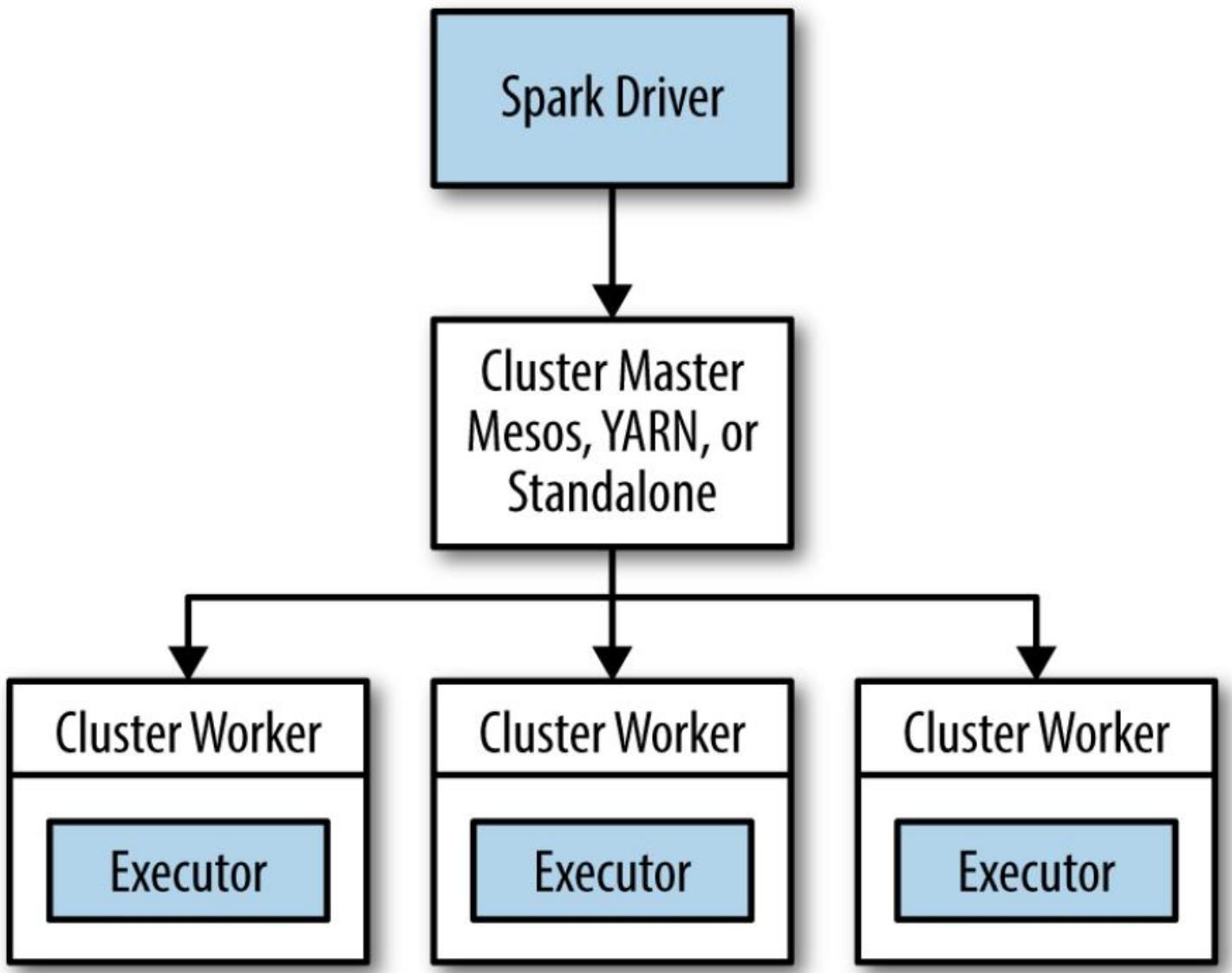
Lazy evaluation

- Transformations on RDDs are lazily evaluated → Spark will not begin to execute transformations until it sees an action
 - Many transformations can be chained together and none will execute until an action generating an output is ‘seen’
- Instead of immediate execution, Spark does the following:
 - Internally **record metadata** about transformation requests → this in essence means that in-memory RDDs can be regarded as instructions for computing data instead of data itself (which is not materialized immediately)
 - **Lazy data load**, i.e. actual data read and parallelize will be executed when needed to perform an action downstream
- Lazy evaluation allows Spark to optimize data processing pipelines inline, transparently to the user, thereby reducing the number of passes over the data

PROCESSES: SPARK APPLICATIONS



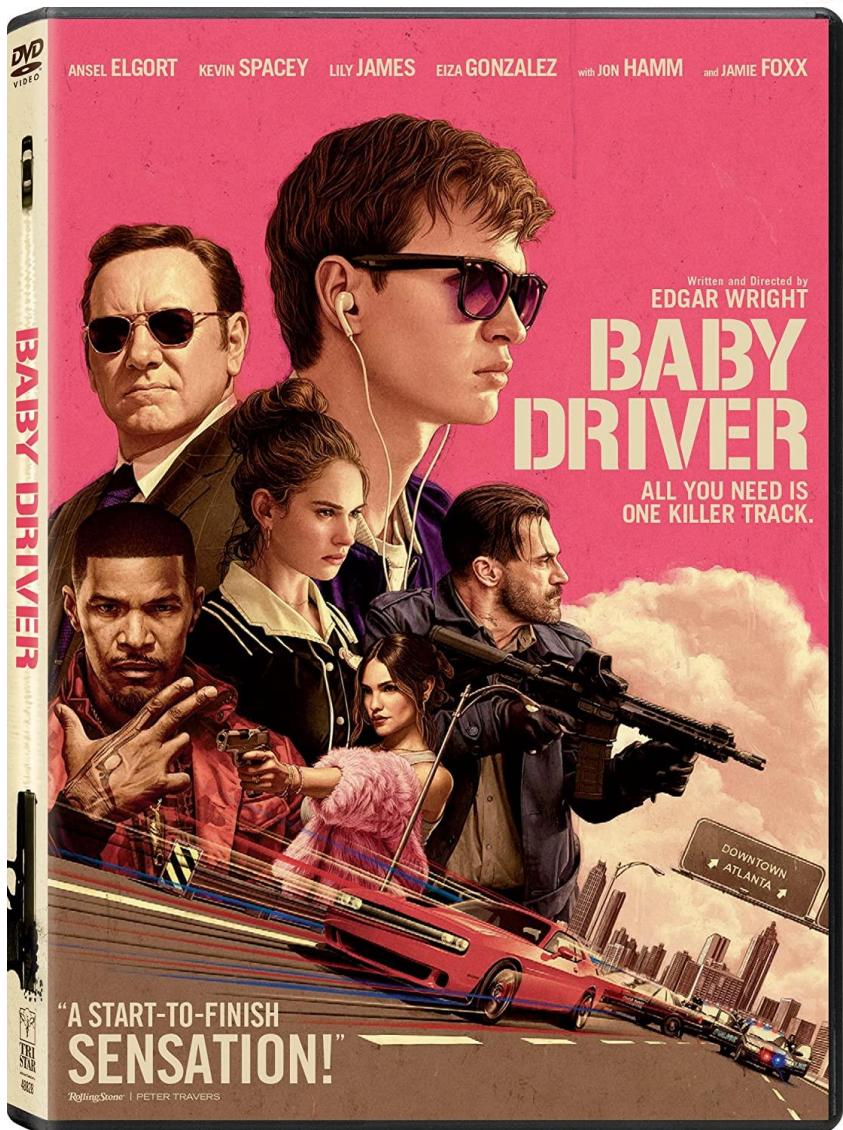
Anatomy of a Spark application





The Driver

- The driver is the center of a Spark application
- The main() method is in the driver
- It runs the user code
- Creates the Spark context
- Loads the input RDDs
- Performs transformations and actions





Driver duty #1: Create tasks

- **DEF:** Spark tasks are the smallest units of physical execution (inside the computing cluster)
- The driver converts the user's program into tasks
- A Spark program implicitly creates a logical directed acyclic graph (DAG) of operations
- Drivers convert the DAG into physical execution plans, pipeline transformations and merge them where able
- The DAG is converted into a set of 'stages'
- Each stage consist of multiple tasks
- Tasks are sent to the cluster for execution



Driver duty #2: Task scheduling

- Tasks are scheduled on individual ‘executors’
- Executors register with the driver when started
- Drivers analyze their current sets of executors and schedule tasks based on data placement
 - This is known in MapReduce as data locality
- When tasks execute, they produce intermediate data which can be cached, e.g. persisted RDDS
 - The driver tracks the location of cached data and schedules additional tasks which use the cached data
- The driver exposes information about the Spark application’s status via a web interface (usually HTTP on port 4040)

Spark executors



- Executors are worker processes running tasks
- Key executor roles:
 - Run tasks and return intermediate results to the driver
 - Provide in-memory storage for RDDs (this is done by the Block Manager process)
- Executors are launched when a Spark application is started
- Their lifetime is usually equal to the Spark app's



Launching a Spark application

- Spark applications are launched via the **spark-submit** script
 - It connects to the various supported cluster managers and controls resource usage
 - It launches the driver and invokes main()
- The **driver** contacts the cluster manager to acquire resources (CPU, memory) and run tasks
- The **cluster manager** launches executors on behalf of the driver
- Tasks are run in **executor** processes to compute and save results
- The **Spark app ends** when main() ends or when the Spark context is explicitly stopped from (user) code

SYNCHRONIZATION AND SCHEDULING

Scheduling and cluster mgmt

- Scheduling and cluster management is performed with
 - Spark's built-in Standalone Cluster Manager
 - Apache YARN
 - Apache Mesos



© Wiley Ink, inc./Distributed by Universal Uclick via Cartoonstock

<https://www.cartoonstock.com/directory/o/oars.asp>

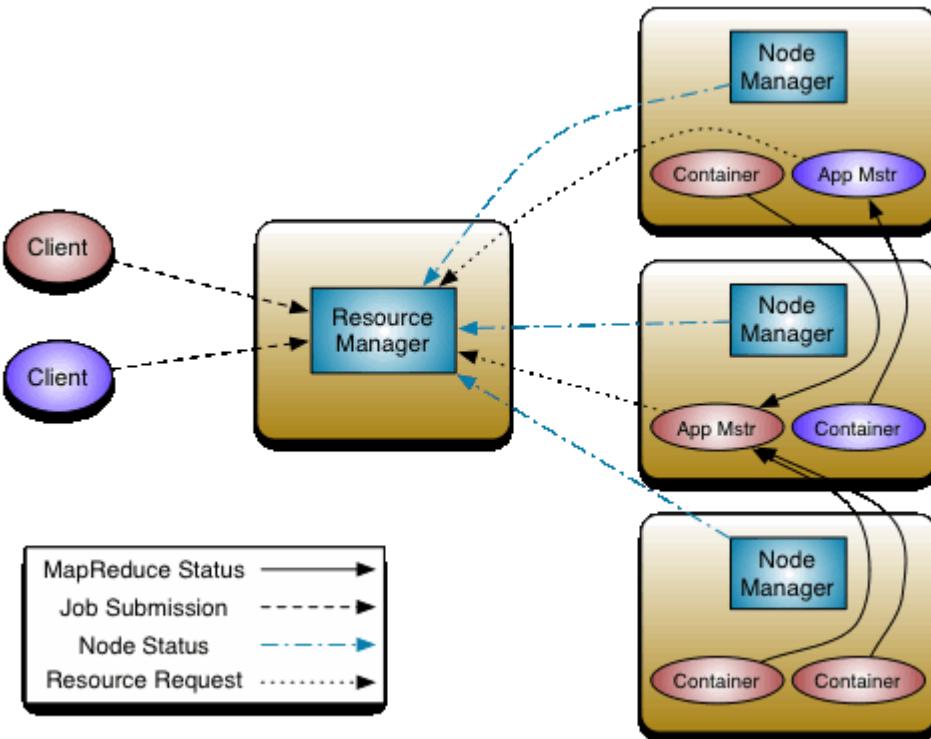


Standalone Cluster Manager

- The Standalone Cluster Manager (SCM) consists of a master and multiple workers
- Workers are assigned configured amounts of memory and CPU cores
- The SCM is by default available on the following URI:
`spark://masternode:7077`
- The SCM's web UI is usually accessible via
<http://masternode:8080>
- The SCM supports 2 deploy modes:
 - Client: the driver runs on the machine where spark-submit is run
 - Cluster mode: the driver is launched on one of the worker nodes
- **Note:** The SCM is a good fit when the computing cluster is not shared with other users and or computing platforms

Apache YARN

- The YARN cluster manager was introduced with Hadoop v2
- It runs on the HDFS nodes → YARN is good scheduling choice if the data consumed by the Spark application is stored in HDFS



Apache Mesos

- **DEF:** Apache Mesos is a general-purpose cluster manager
- Mesos can run both analytics workloads and long-running services
- Mesos clusters can also use ZooKeeper to elect a master when running in multi-master node
- Mesos modes: fine-grained & coarse-grained



Apache
MESOSTM



Cluster manager comparison

- The **Standalone Cluster Manager** is easiest to set up and is a good choice with dedicated clusters, i.e. when running only Spark on a set of compute nodes
- **YARN** is a good choice when Spark is run on a (shared) cluster where we already have Hadoop installed, e.g. when the data is stored in HDFS
- **Mesos** is attractive (compared to the SCM & YARN) when running multiple interactive user sessions, as it can scale up & down resource use (CPU & memory) between commands issued in a user session
- **Note:** In all cases, it is a good idea to design the Spark cluster with **data locality** in mind, i.e. to deploy the executors as close to the data as possible

FAULT TOLERANCE



Faults in the scheduler

- **Executor node failure:** If a node (e.g. a single server computer) fails, its tasks are re-run on a different node and the affected RDD partitions are re-computed based on the lineage graph
- **Driver node failure:** If the node running the driver, or the driver code fails, the Spark context is lost → re-launch the Spark application, re-start the driver and all executors
 - With file-based inputs this does not result in data loss → everything is re-computed
 - With input streams, buffered data would be lost in the executors → Spark 1.2+ have write-ahead logs
- **Cluster manager failure:** The Standalone Cluster Manager, YARN or Mesos can be run in hot-standby mode via the Apache Zookeeper distributed coordination system → this ensures that the Spark cluster will not fail when a single cluster manager fails

<https://stackoverflow.com/questions/24762672/how-does-apache-spark-handles-system-failure-when-deployed-in-yarn>
<https://databricks.com/blog/2015/01/15/improved-driver-fault-tolerance-and-zero-data-loss-in-spark-streaming.html>

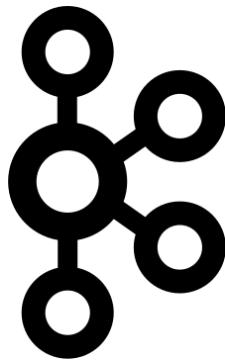
Spark summary

- Spark intro
- Spark architecture
- Resilient Distributed Dataset (RDD)
- Processes
 - Data analytics
 - The Spark cluster
- Synchronization and scheduling
- Fault tolerance



Thank you for your attention!





STREAMING SYSTEMS: KAFKA

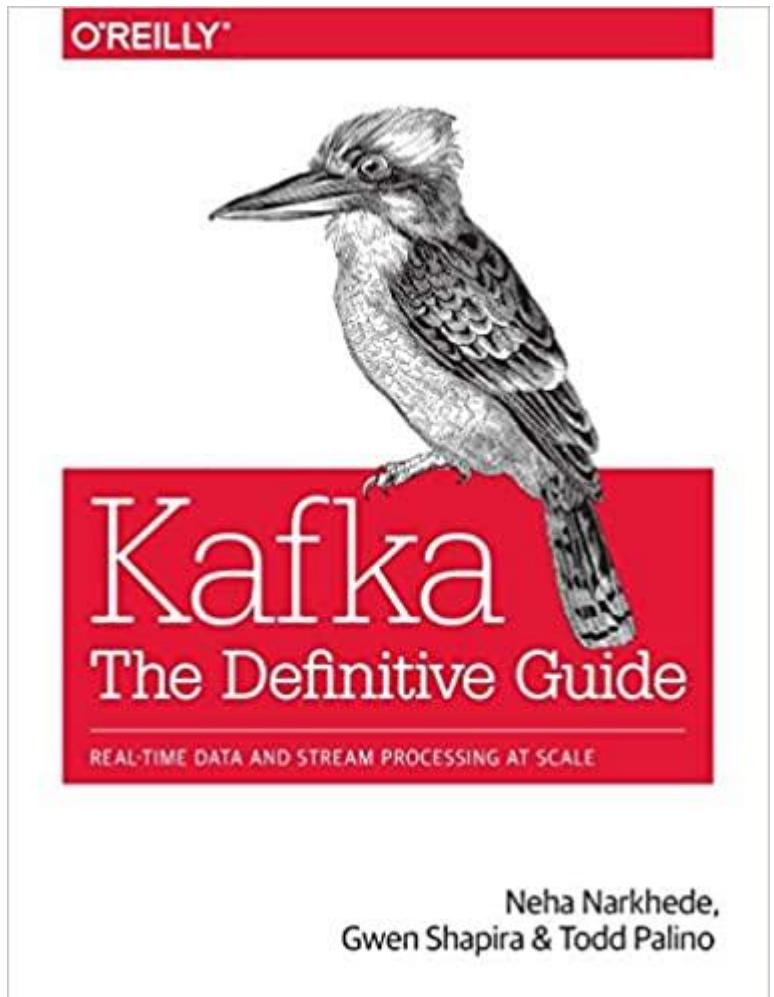
Open-source Technologies for Real-Time Data Analytics

Imre Lendák, PhD, Associate Professor



Overview

- Introduction
- Motivation
- Architecture
- Processes
 - Brokers
 - Producers
 - Consumers
- Replication & consistency
- Monitoring & control





Definition & origins



- **DEF:** Apache **Kafka** is an open-source data stream processing platform
- **Originally developed by:** LinkedIn
- **Initial release:** Jan 2011
- **Current release:** 2.6.0 in August 2020
- **Written in:** Scala & Java
- **License:** Apache License 2.0
- **Author(s):** 9 core committers, plus ~ 20 contributors
- **Website:** <http://kafka.apache.org/>



Key features

Guarantees

- Data integrity checks
- At least once delivery
- In order delivery, per partition

Characteristics

- Very high performance
- Elastically scalable
- Low operational overhead
- Durable, highly available

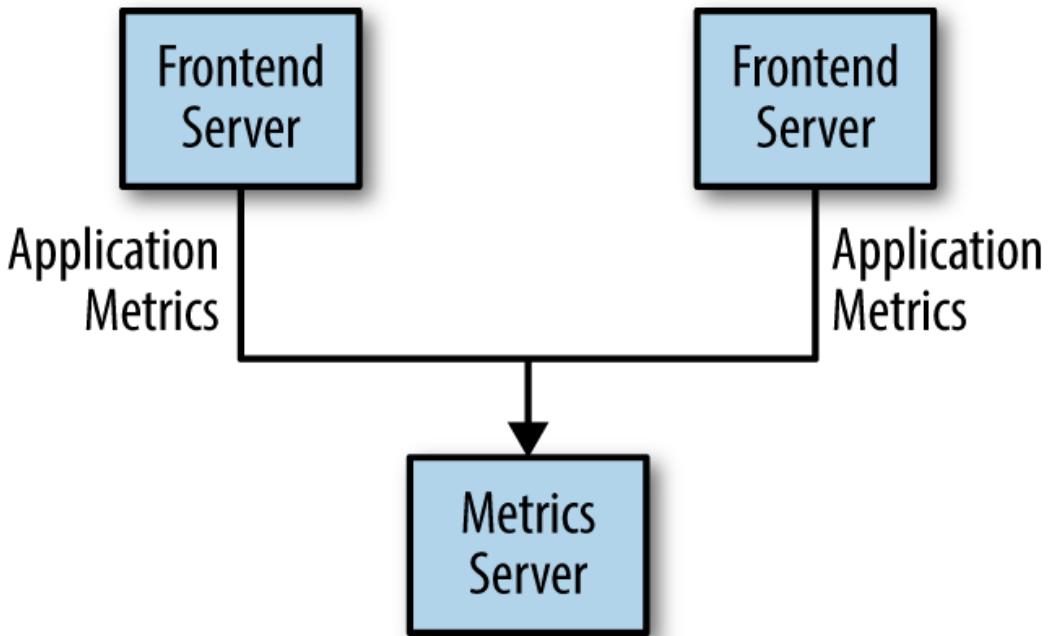
MOTIVATION

Motivation

- LinkedIn's motivation for Kafka was:
 - “A unified platform for handling all the real-time data feeds a large company might have.”

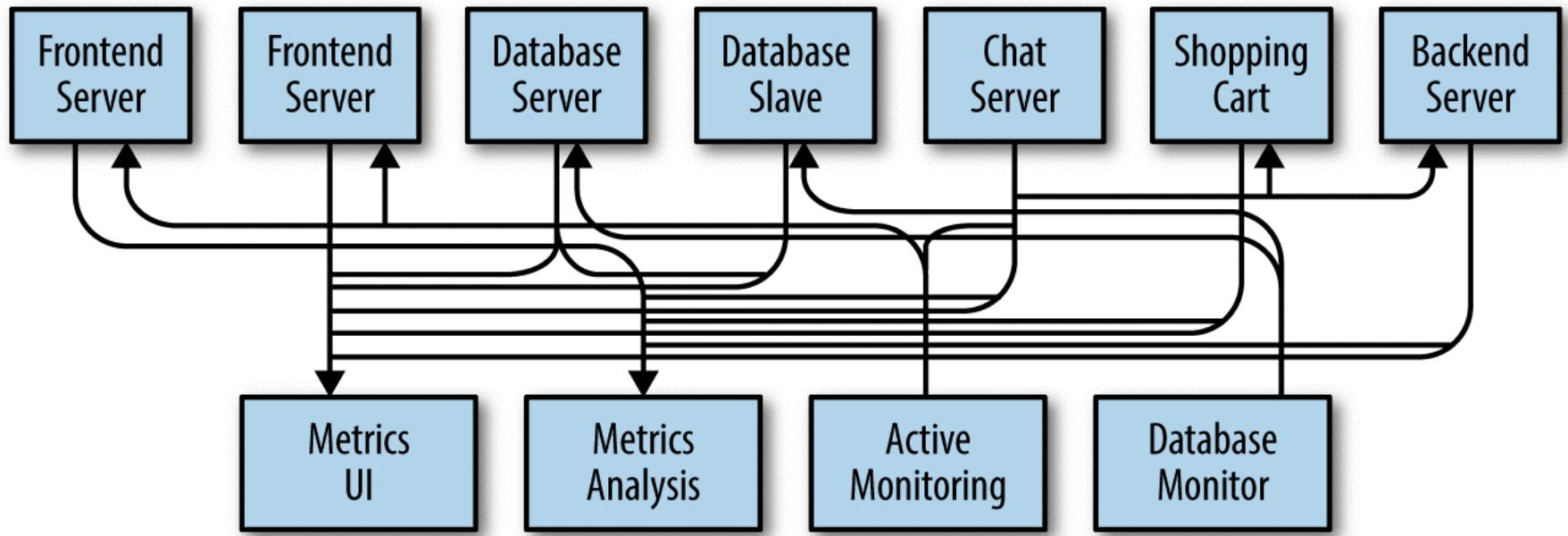
- Features
 - High throughput to support **high volume event feeds**.
 - Support real-time processing of these feeds to create **new, derived feeds**.
 - Support large data backlogs to handle periodic ingestion from **offline systems**.
 - Guarantee **fault-tolerance** in the presence of machine failures.

Scalability challenges – 1



- **Challenge:** What if the number of producers (i.e. server) and consumers (not shown here) increases?

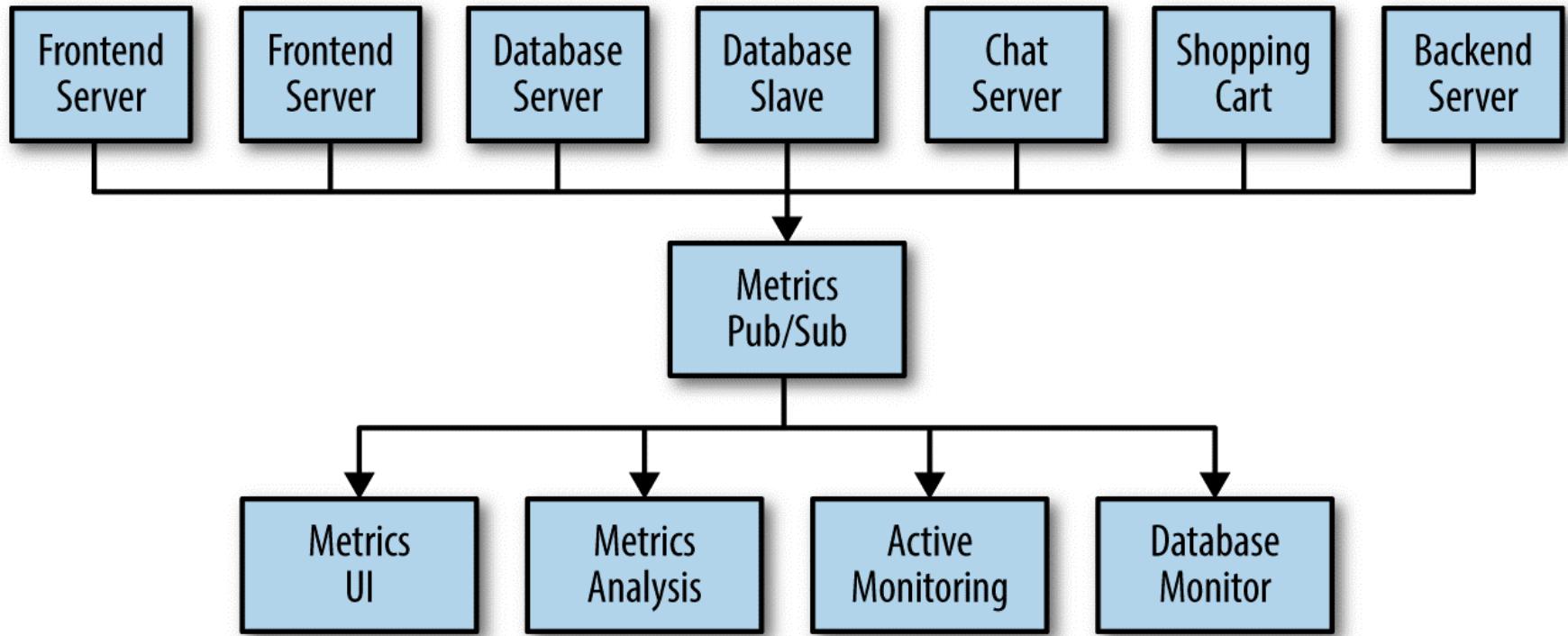
Scalability challenges – 2



- **Challenge:** How to handle the large number of interconnections between the different sources & sinks?

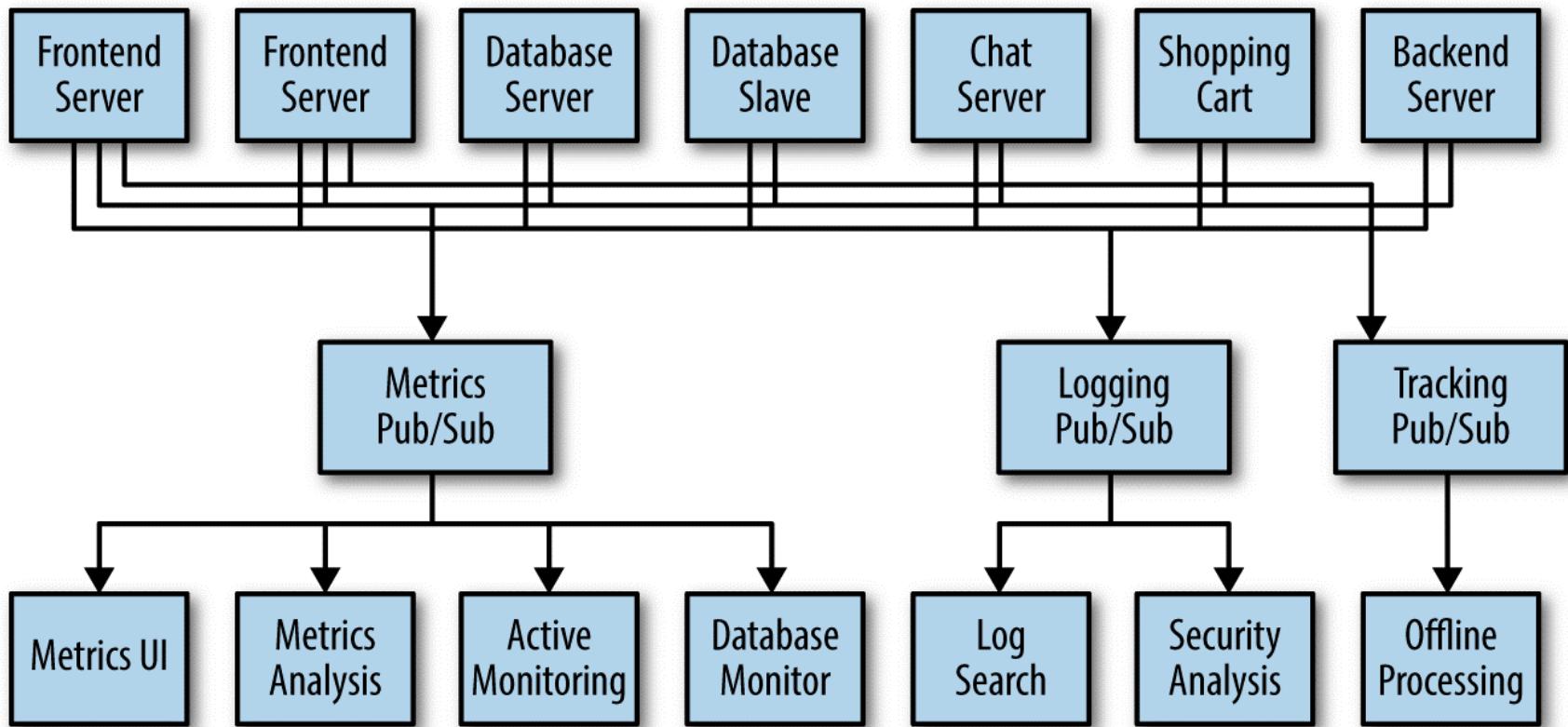


Scalability challenges – 3



- **Challenge:** Metrics data management solved! But what if there are other data types in a large enterprise?

Scalability challenges – 4

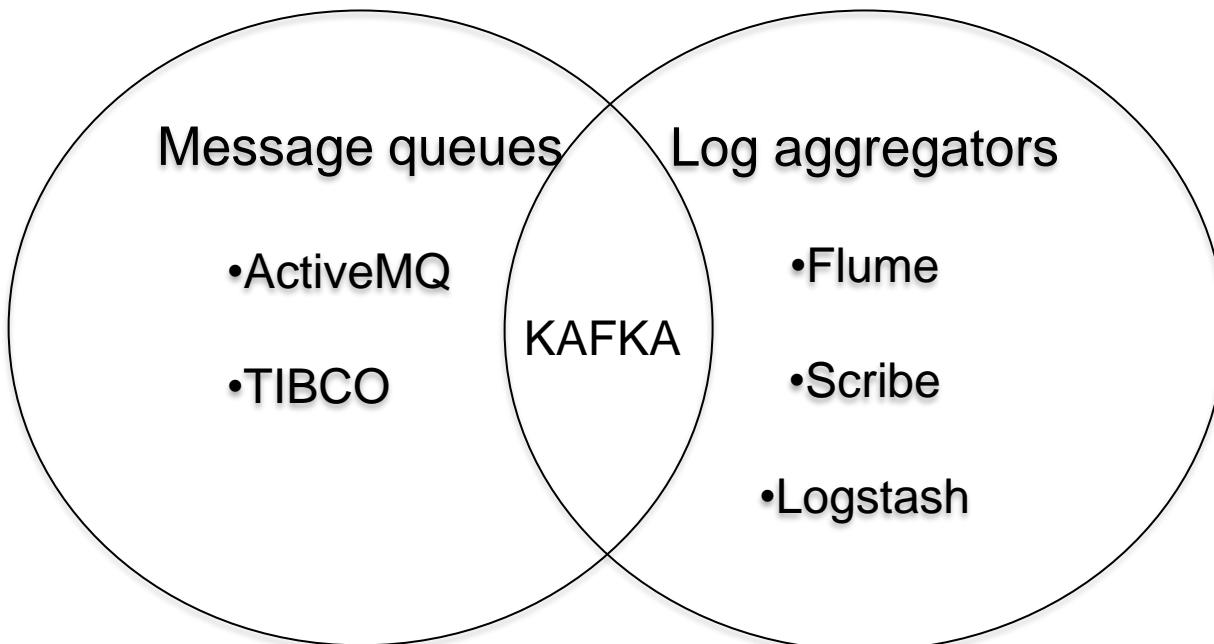


- **Challenge:** How to handle the diverse data types (in the nodes in the middle?)

ARCHITECTURE

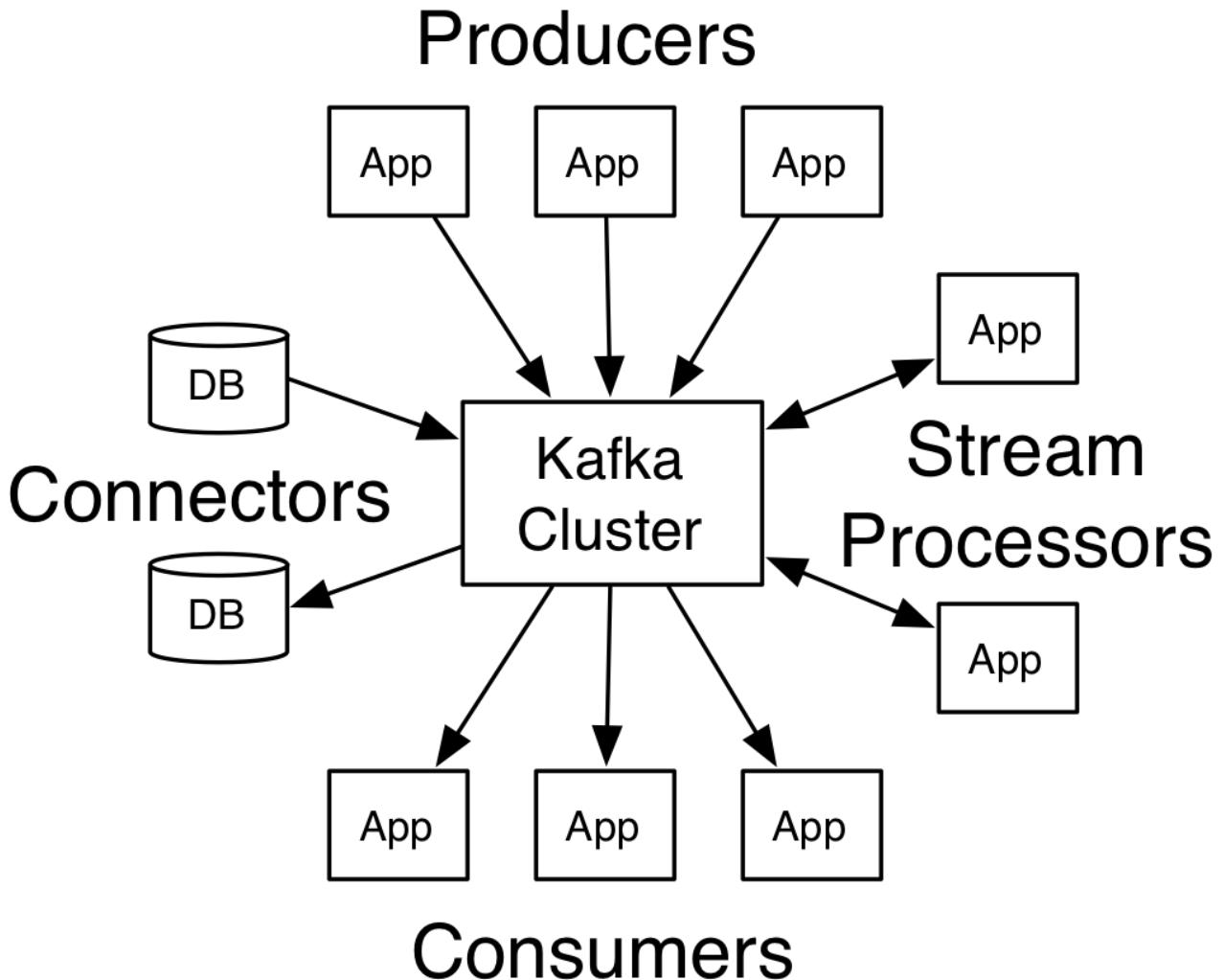


Queues and aggregators



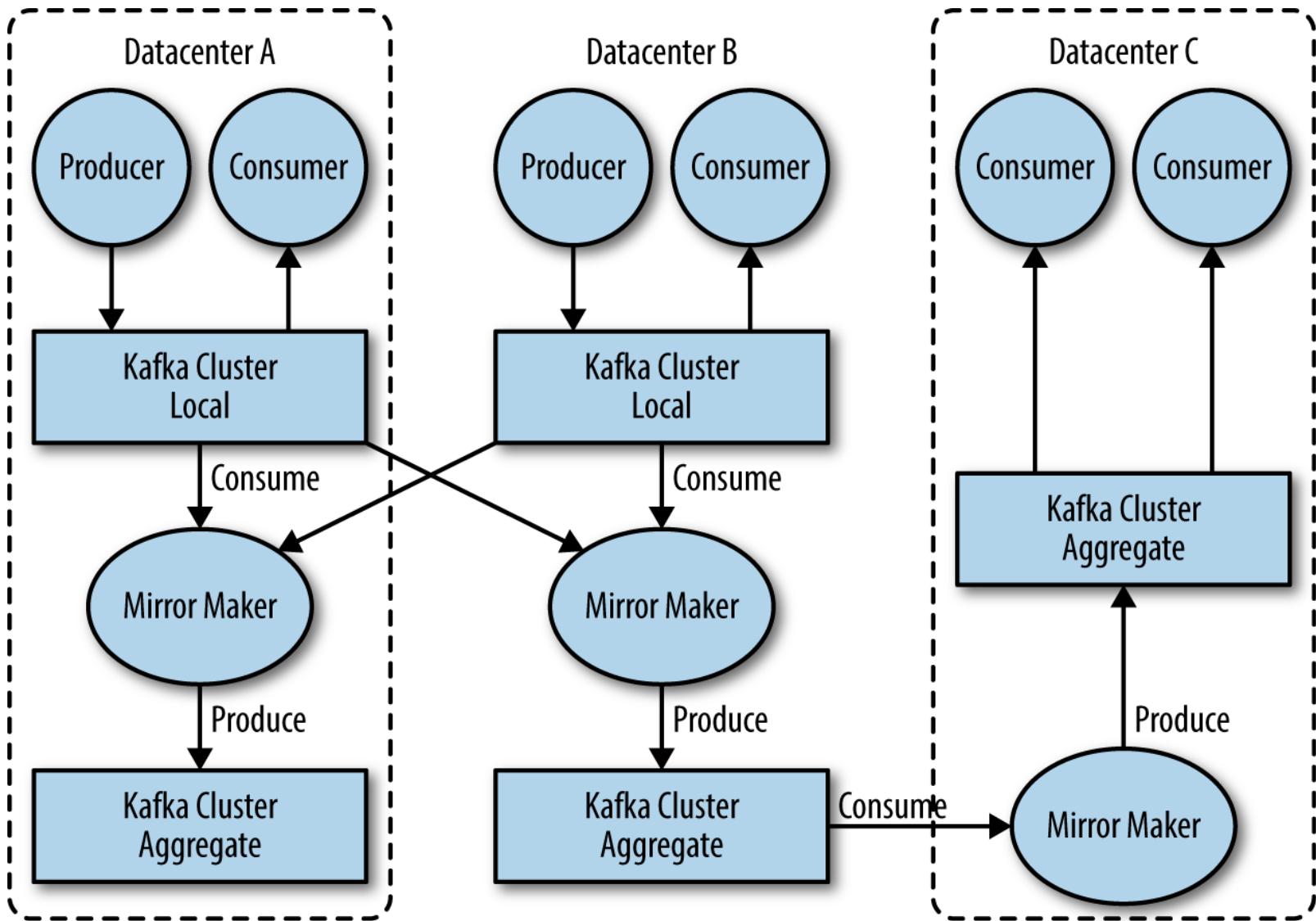
- Kafka has a publish-subscribe (often abbreviated as pubsub) architecture → a distributed mix of message queues and log aggregators

Architecture overview



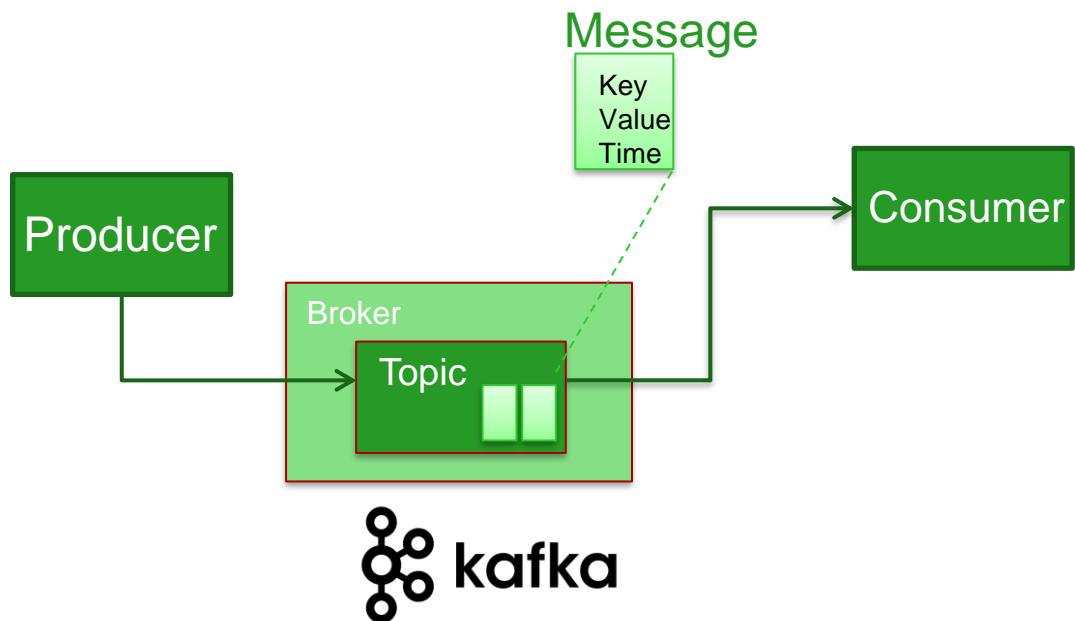


Multi-datacenter architecture

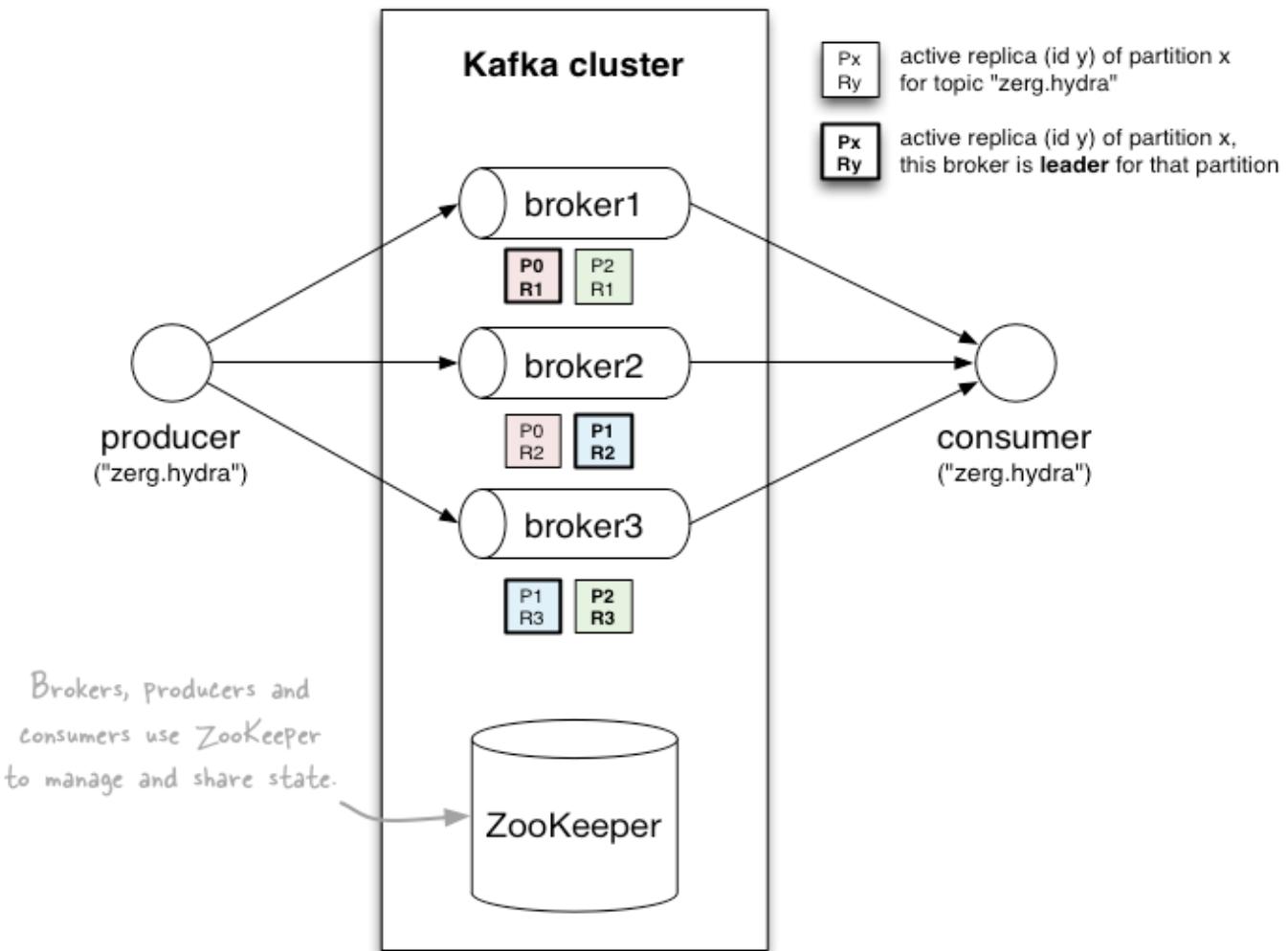


Who is who? (Terminology)

- Topic
 - partition
- Message
 - == ByteArray
- Broker
 - replicated
- Producer
- Consumer
 - Working together in Consumer Groups

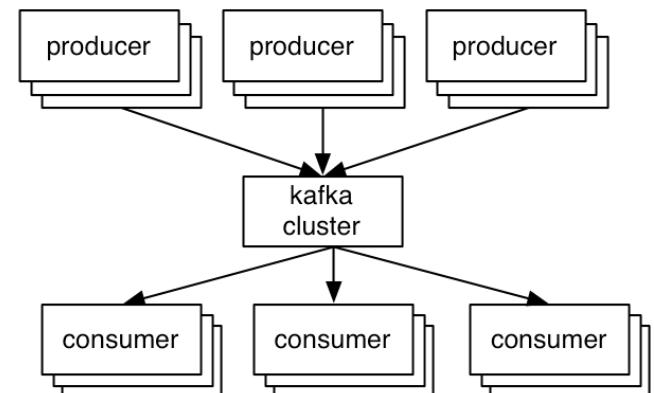


Architecture



Key processes

- The who is who
 - **Producers** write data to **brokers**.
 - **Consumers** read data from **brokers**.
- The data
 - Data is stored in **topics**.
 - **Topics** are split into **partitions**, which are **replicated**.



PROCESSES: BROKERS



Brokers

- **DEF:** Kafka brokers are the processes tasked to receive messages from producers, consistently store them and respond to requests from Kafka consumers
- A Kafka cluster consists of one or more brokers
- Brokers are usually executed different servers
- One broker can maintain multiple partitions of different Kafka topics
- The brokers maintain special, non-producer-defined topics for administrative purposes, e.g. topics for memorizing message offsets for consumers

Topics

Consumer group C1

Consumer group C2

Consumers use an “offset pointer” to track/control their read progress (and decide the pace of consumption)

...
Older msgs

Newer msgs

ne
w

Producer A1
Producer A2
...
Producer An

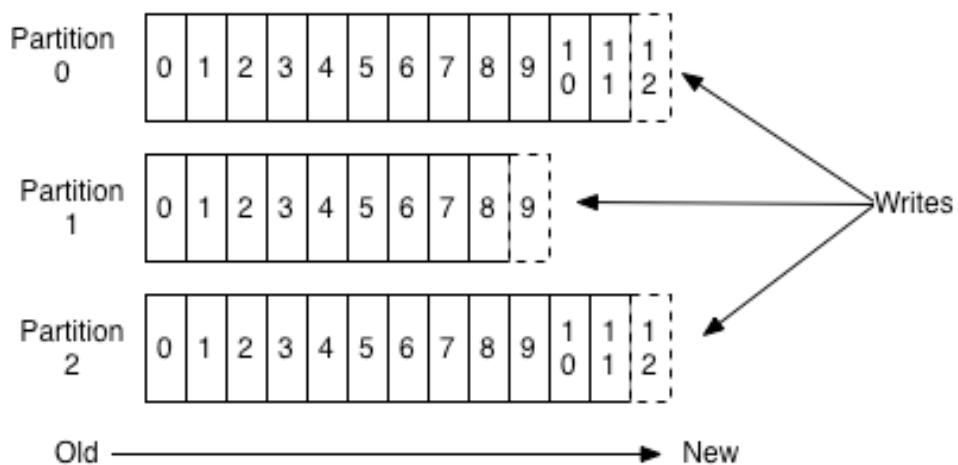
Producers always append to “tail” (think: append to a file)

Broker(s)

Partitions

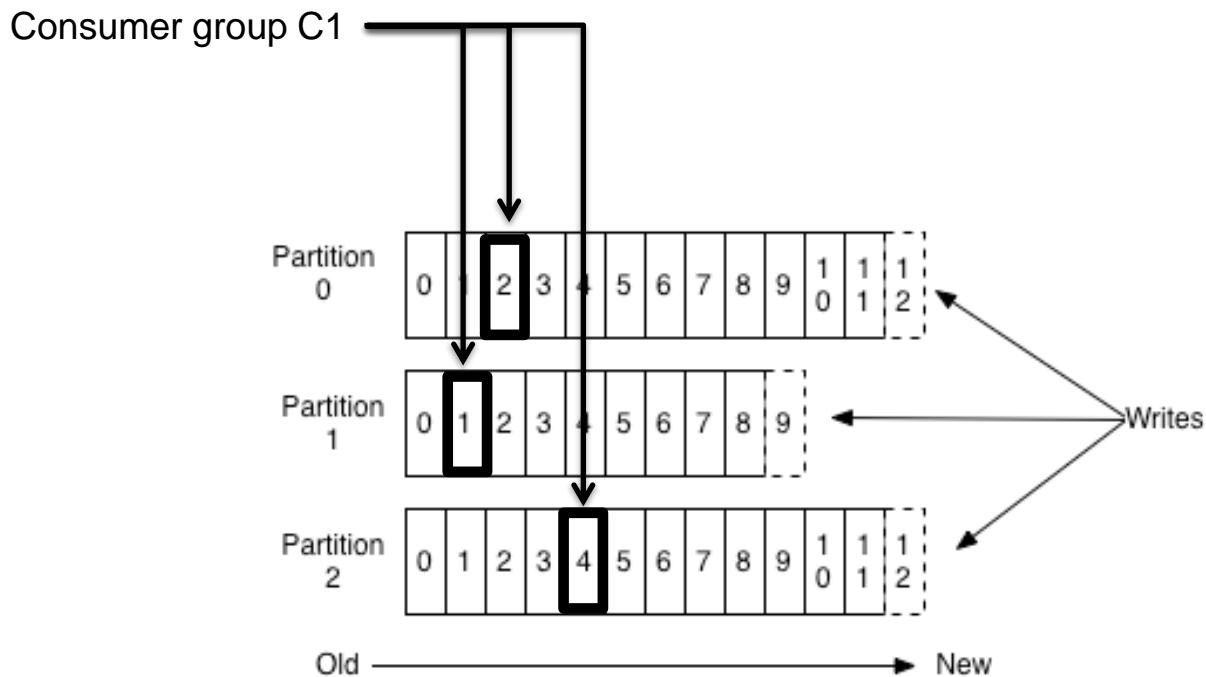
- A topic consists of **partitions**.
- Partition: **ordered + immutable** sequence of messages that is continually appended to

Anatomy of a Topic



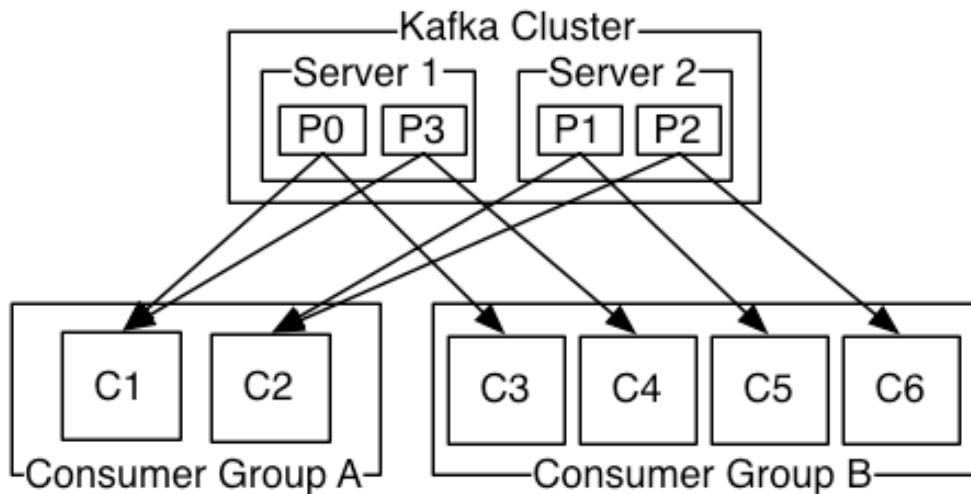
Partition offsets

- **Offset:** messages in the partitions are each assigned a unique (per partition) and sequential id called the *offset*
 - Consumers track their pointers via *(offset, partition, topic)* tuples



Partitions

- #Partitions of a topic is configurable
- #Partitions determines **max consumer (group) parallelism**



- Consumer group A, with 2 consumers, reads from a 4-partition topic
- Consumer group B, with 4 consumers, reads from the same topic

Broker performance



Efficiency

- Each topic has an ever-growing log
 - A log == a list of files
- A message is addressed by a log offset
- Batch send and receive
- No message caching in JVM
- Rely on file system buffering
- 1 file system operation per request

Implementation

- **Fast writes:**
 - While Kafka persists all data to disk, essentially all writes go to the **page cache** of OS, i.e. RAM.
- **Fast reads:**
 - Very efficient to transfer data from page cache to a network **socket**
 - Linux: **sendfile()** system call
- Combination of the above two features → highly efficient Kafka

PROCESSES: PRODUCERS



Producer intro

- Producer **use cases**:
 - Record user activities for auditing and analysis
 - Record infrastructure metrics, e.g. CPU load, RAM use, bandwidth utilization
 - Store logs
 - Record information from smart devices, e.g. in an electric power system setting
 - Buffer information before writing to a database
- Producers create **producer record** objects which consist of (topic, partition, key, value) tuples
 - The partition and key values are optional
 - When defined, the partition defines the destination partition → if it undefined, the partitioner assigns the record
- Producers rely on different serializers to convert records, e.g. Apache Avro, Java serialization, custom serialization

Producer workflow

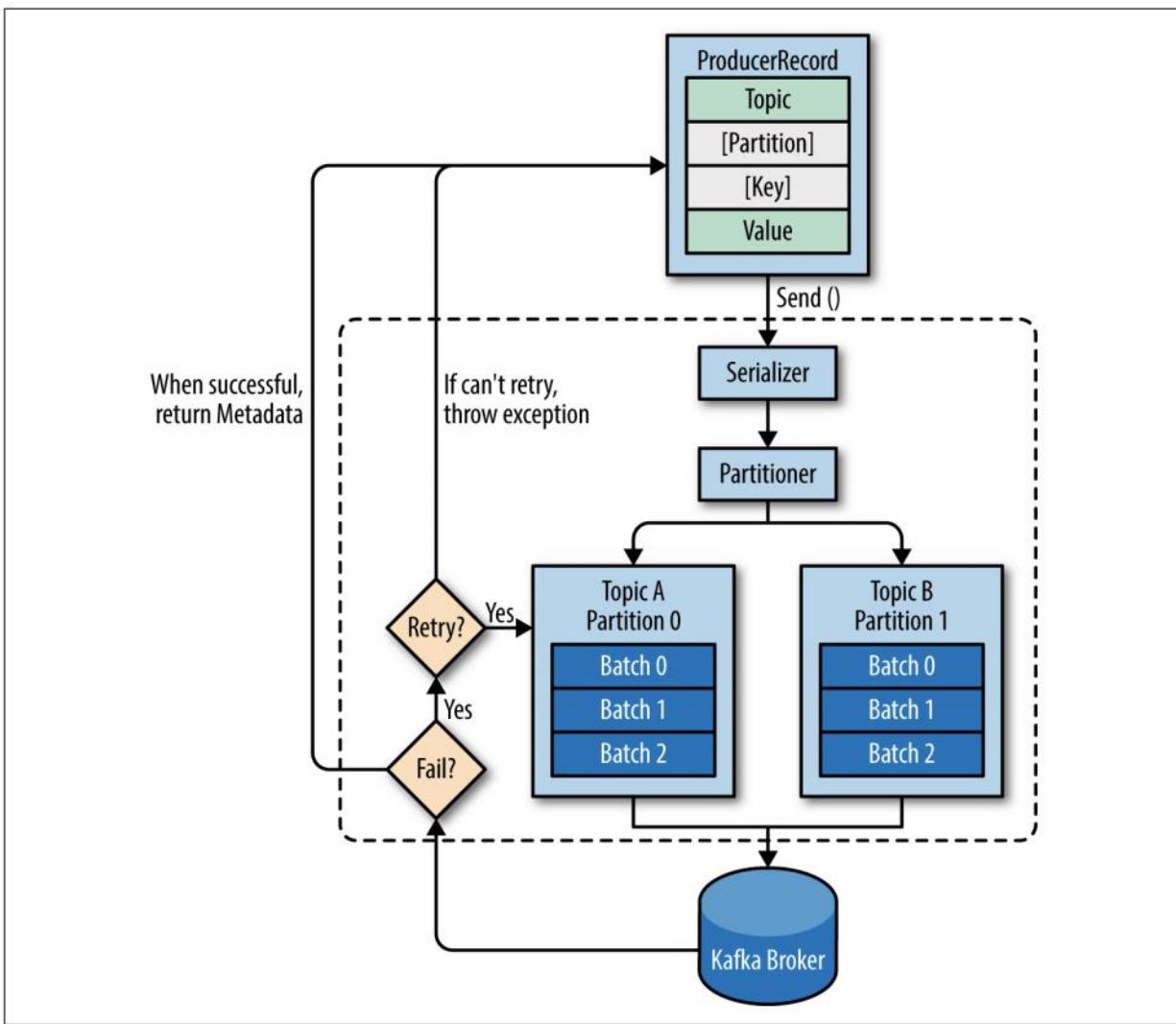


Figure 3-1. High-level overview of Kafka producer components

PROCESSES: CONSUMERS

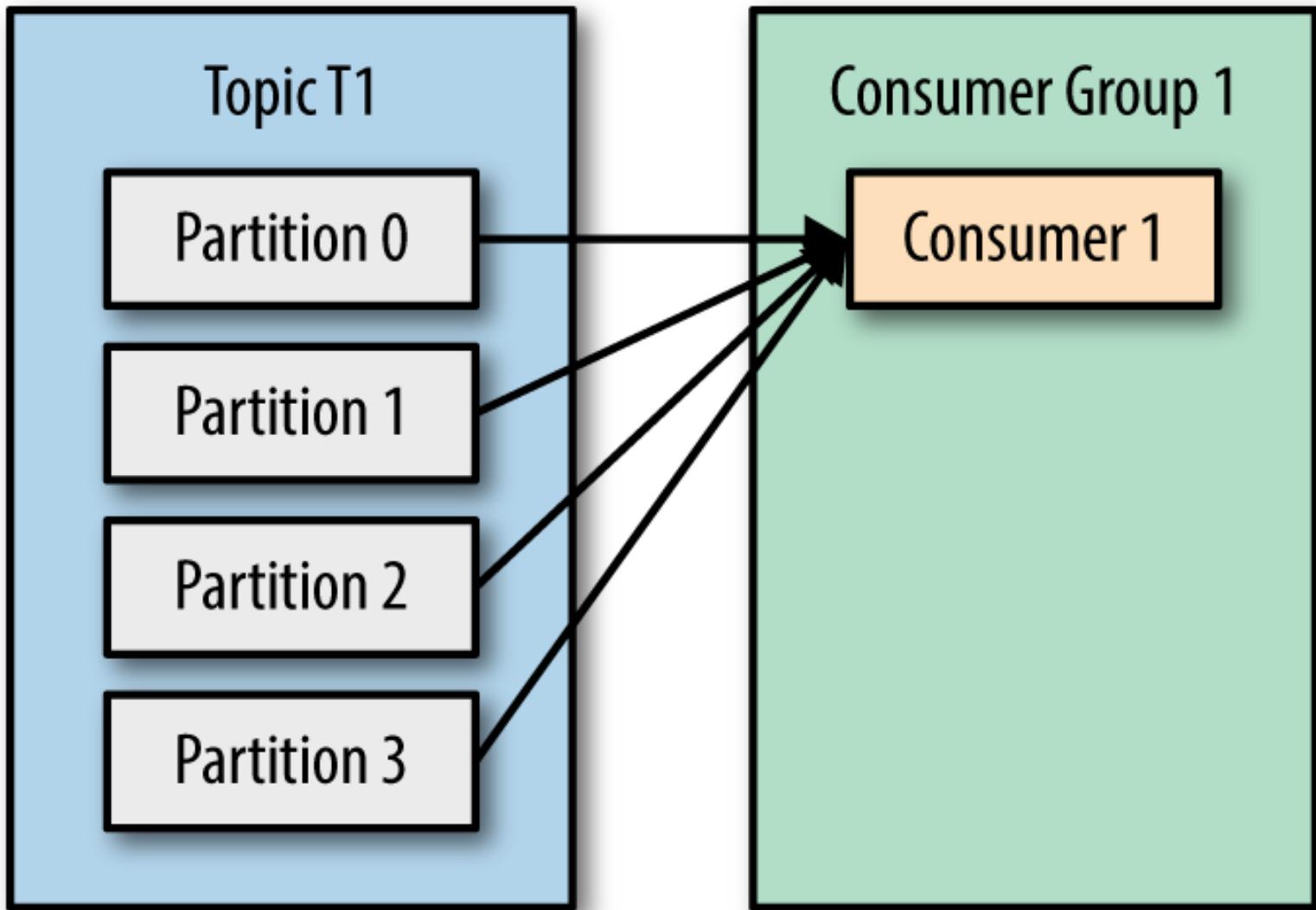


Consumer intro

- **DEF:** **Kafka consumers** are processes which subscribe to and receive messages from Kafka topics
- **Simple consumer** scenario: a single consumer subscribe to a single or multiple topics and processes the data
 - A single consumer can become a bottleneck if it performs costly data analyses or writes to a database → multi-consumer usage scenario
- **Multi-consumer** scenario: a single consumer process cannot process the tide of incoming, unbounded data flows → consumer groups with multiple consumers
 - The different consumers receive messages from a different subset of topic partitions
- When consumers consume messages, they commit the current partition offsets to a special Kafka topic
 - Earlier versions (prior to 0.10.x) committed offsets to Zookeeper

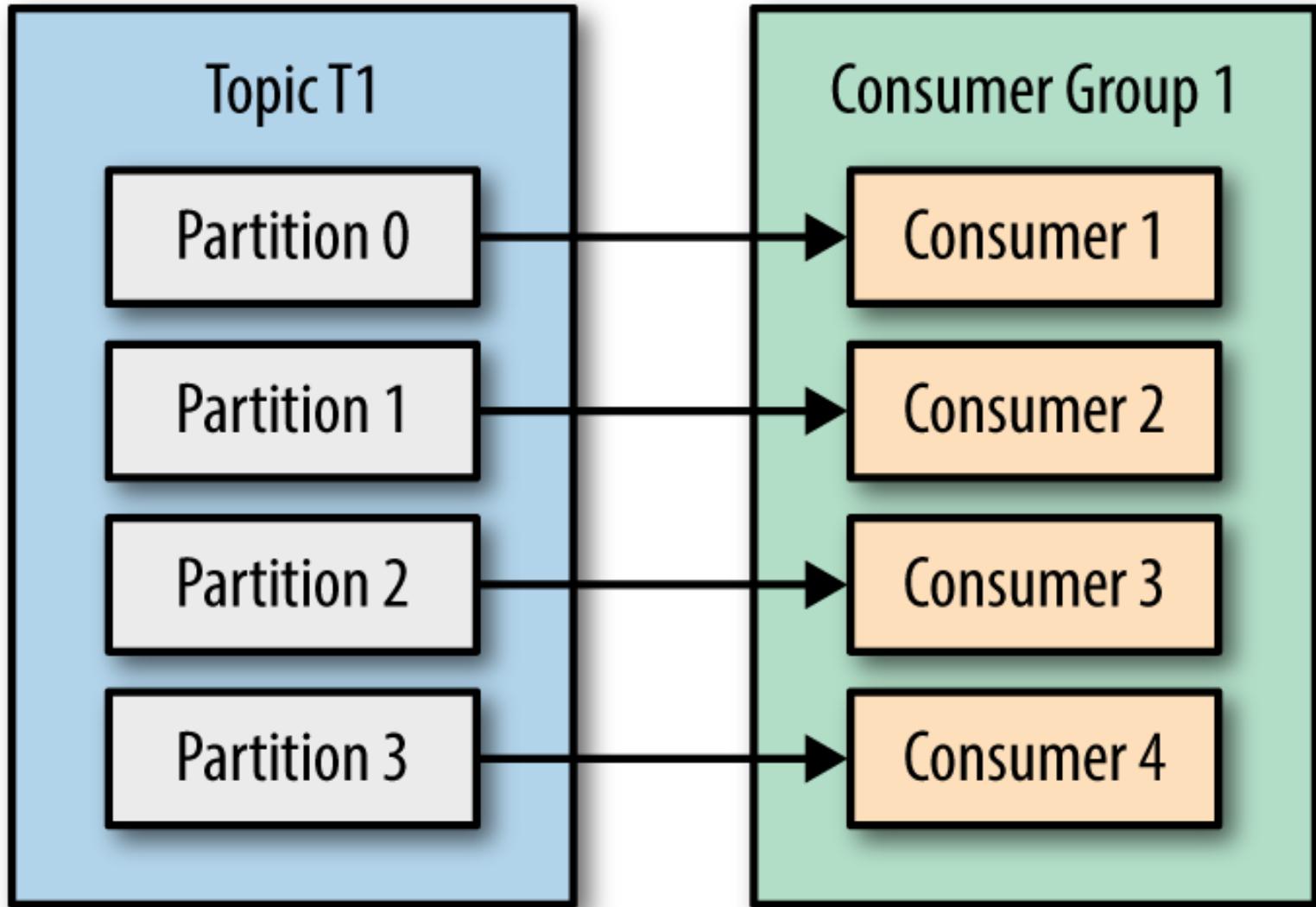


Single group, single consumer

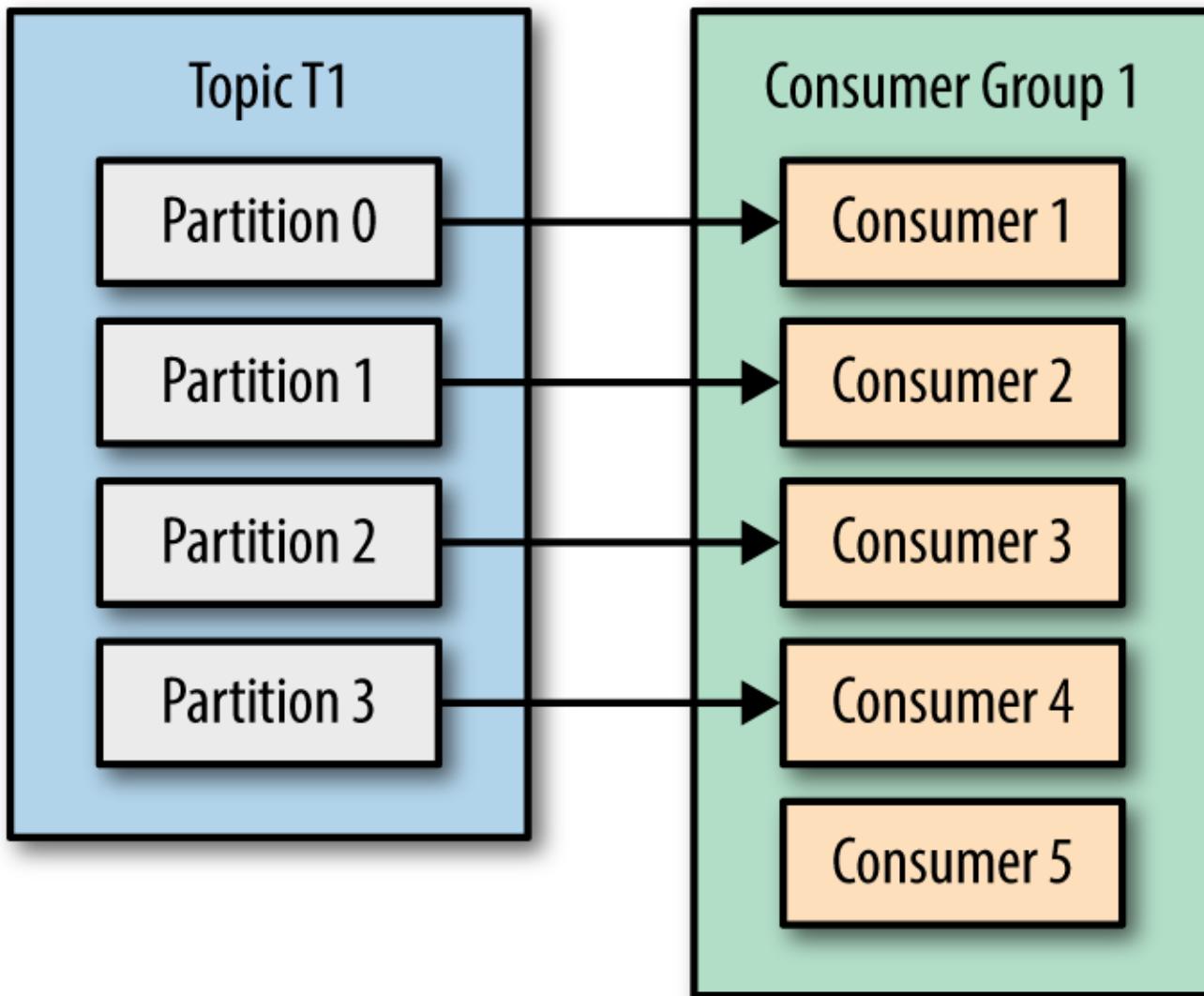




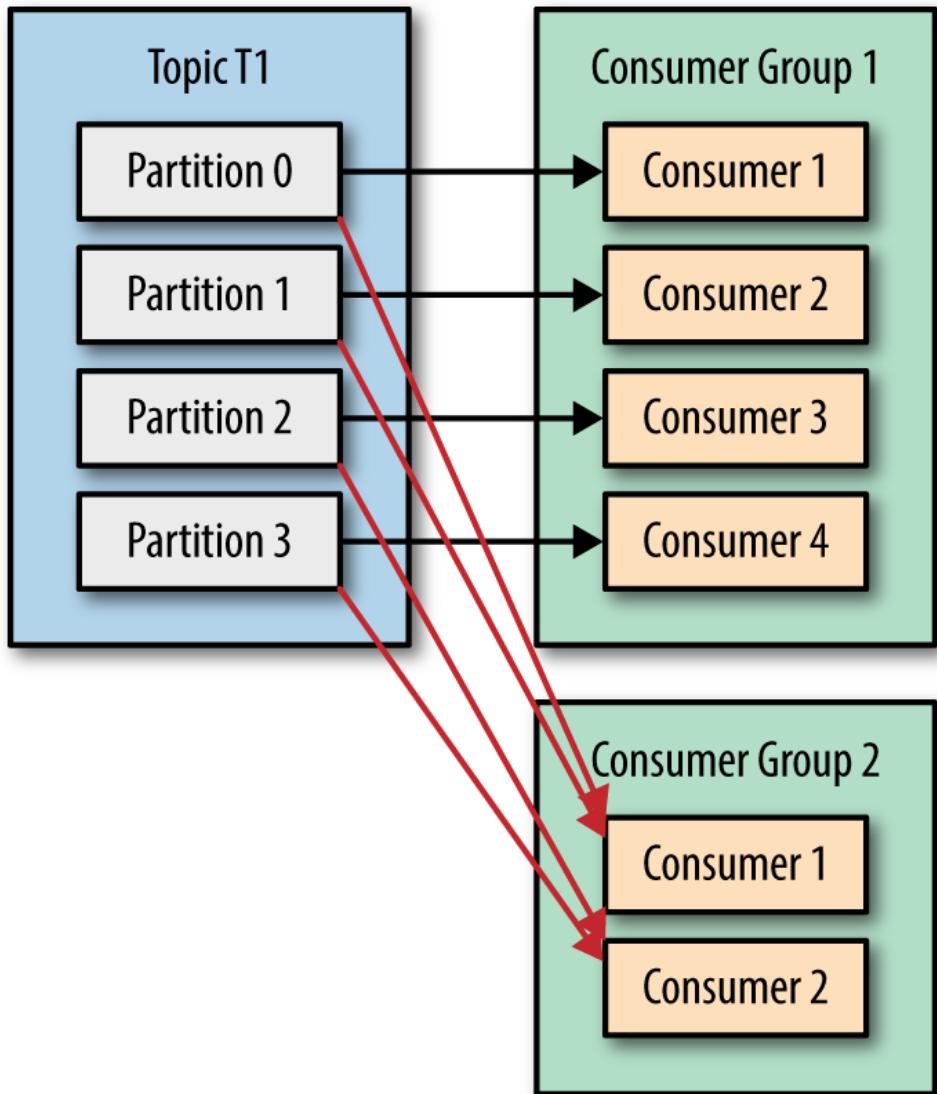
Optimum number of consumers



Too many consumers



Multiple consumer groups



- The different consumer groups can perform different types of data transformations
- Note: adding additional consumers on-the-fly is possible → partition rebalance

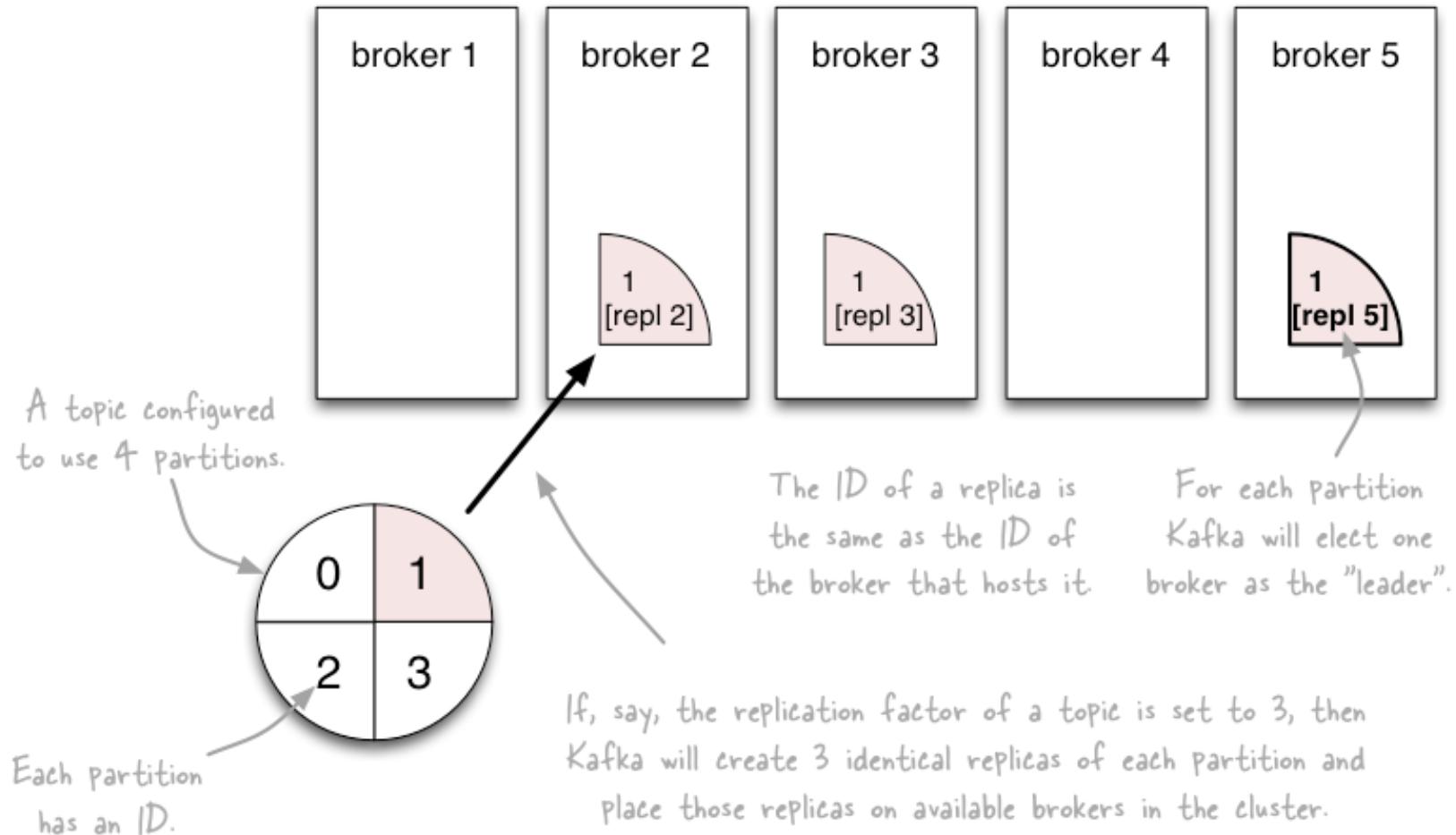
REPLICATION AND CONSISTENCY



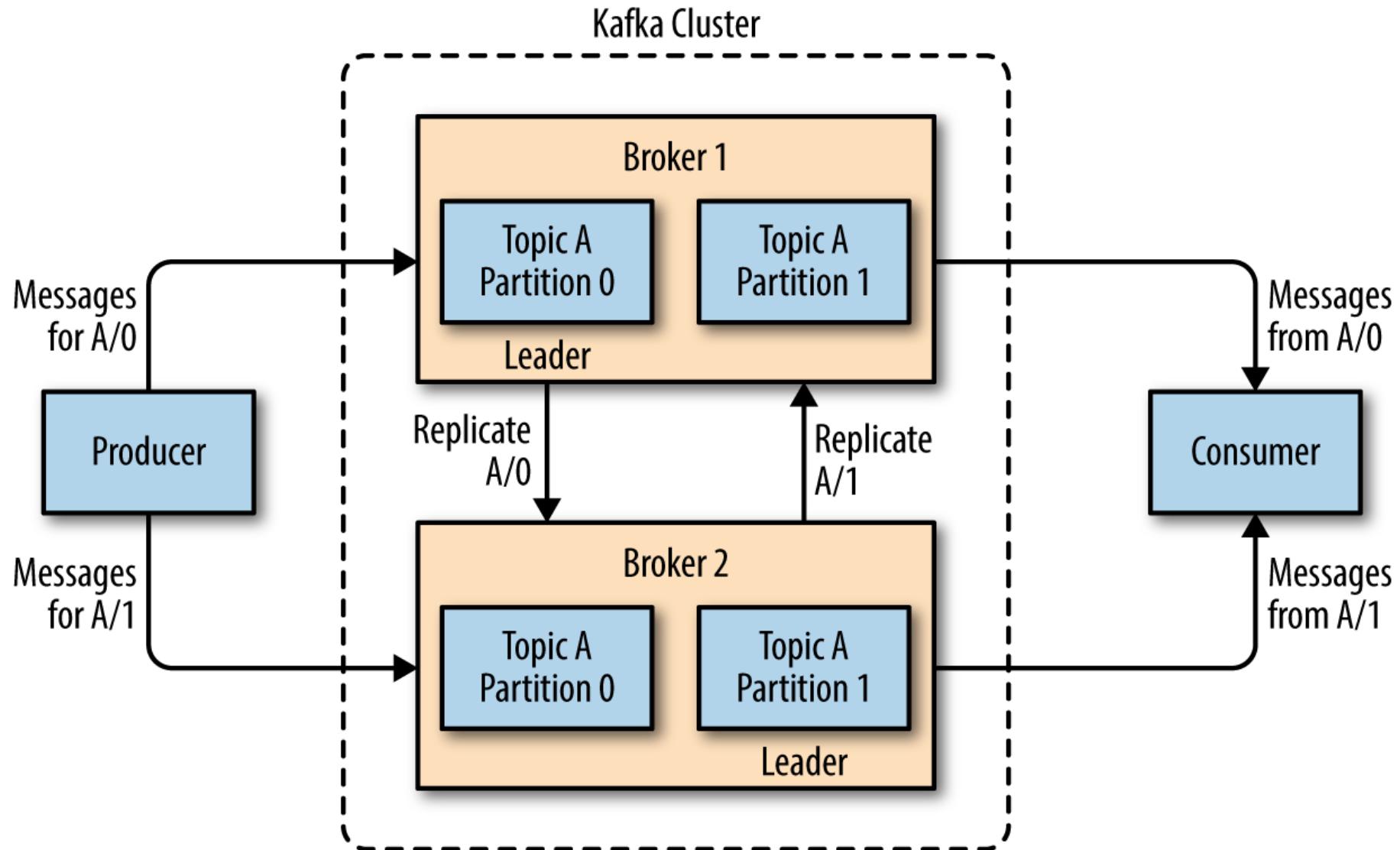
Replicas of a partition

- Kafka **replicas** are partition “backups”
 - Kafka tolerates $(numReplicas - 1)$ dead brokers before losing data
- Replica types:
 - **Leader replica:** Each partition has a single replica designated as the leader. All produce/consume requests go to the leader.
 - **Follower replica:** All non-leader replicas for each partition are called followers. Followers do not serve produce/consumer requests. They exist to avoid data loss.
- It is the task of the leader to know which follower replicas are up-to-date → in-sync replicas
- Only in-sync replicas can become leaders when the current leader exits the Kafka cluster, e.g. it fails

Topics vs. Partitions vs. Replicas



Partition replication



Replication and visibility

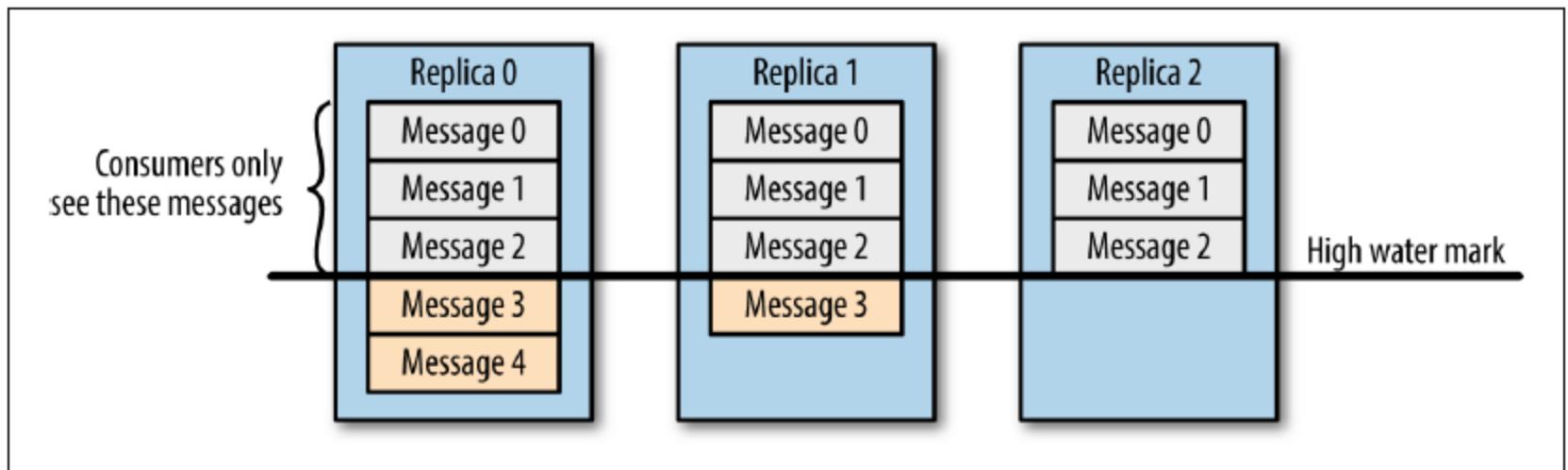


Figure 5-4. Consumers only see messages that were replicated to in-sync replicas

MONITORING AND CONTROL

Monitoring Kafka

- Nothing fancy built into but see:
 - <https://cwiki.apache.org/confluence/display/KAFKA/System+Tools>
 - <https://cwiki.apache.org/confluence/display/KAFKA/Ecosystem>

The screenshot shows the Kafka Web Console interface. On the left, there are three main sections: 'Zookeepers' (with a zk icon), 'Brokers' (with a brokers icon), and 'Topics' (with a mail icon). The 'Topics' section is currently active. It displays a table with columns: Zookeeper, Topic, and Partitions. The data is as follows:

Zookeeper	Topic	Partitions
Test SNAPSHOT	dead-letter-queue	1
Latest	exceptions	1
Latest	metrics	10
Latest	alerts	2
Latest	logs	5
Latest	notifications	10
Live	exceptions	1
Live	metrics	10
Live	alerts	2
Live	logs	5
Live	notifications	10

Kafka Web Console



Kafka Offset Monitor

SUMMARY

Who uses Kafka?

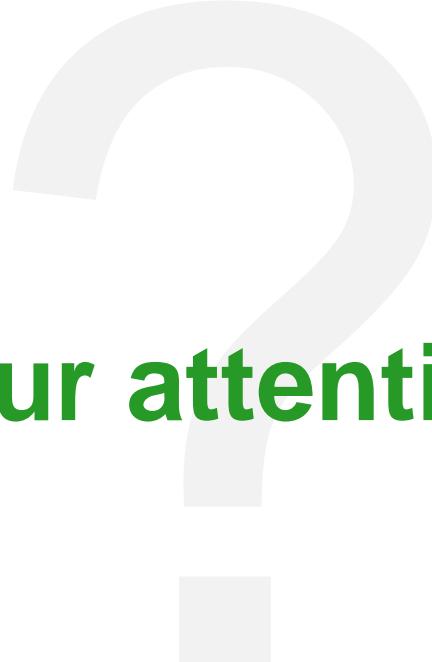
- **LinkedIn**: (user) activity streams, operational metrics, data bus
 - 400 nodes, 18k topics, 220B msg/day (peak 3.2M msg/s), May 2014
- **Netflix**: real-time monitoring and event processing
- **Twitter**: as part of their Storm real-time data pipelines
- **Spotify**: log delivery (from 4h down to 10s), Hadoop
- **Loggly**: log collection and processing
- **Mozilla**: telemetry data
- **Others**: Airbnb, Cisco, Gnip, InfoChimps, Ooyala, Square, Uber, etc.

Summary

- Introduction
- Motivation
- Architecture
- Processes
 - Brokers
 - Producers
 - Consumers
- Replication & consistency
- Monitoring & control



Thank you for your attention!





STREAMING SYSTEMS: SPARK STREAMING, STORM & FLINK

*Open-source Technologies for Real-Time Data
Analytics*

Imre Lendák, PhD, Associate Professor

Introduction

- **Spark Streaming** → a microbatch processing platform with strong consistency guarantees
- Apache **Storm** is a real-time, distributed stream processing platform
- Apache **Flink** a unified stream and batch processing framework





SPARK



Introduction & history

Definitions

- **DEF:** Apache Spark is an open-source, distributed, general-purpose **cluster-computing framework**
- The authors aimed to perform **in-memory** calculations in computing clusters without ‘touching the disk’ before reaching the final data processing stage (i.e. output)
- Written in Scala
- Additionally optimized for interactive queries and iterative computing jobs

History

- Originally developed by the AMPLab at UC Berkeley around 2009
- As soon as 2009 it was outperforming MapReduce 10-20x in certain types of problems
- Open sourced in 2010 (BSD license)
- Donated to the Apache Software Foundation in 2013
- Top-level Apache project since 2014



Whois AMPLab?

AMPLab

- AMP = Algorithms, Machines and People Lab
- Doing research and publishing scientific publications since 2008
- AMPLab officially launched in 2011
- Worked on different ‘big data’ projects under the Berkeley Data Analytics Stack (BDAS)
- UC Berkeley launched RISELab as the successor to AMPLab in 2017

Best known projects

- **Apache Spark** – distributed, general-purpose computing platform
- **Apache Mesos** – cluster management platform
- **Alluxio** – virtual distributed file system (VFDS) – Alluxio ‘sits’ between computation & storage in large-scale data processing environments. Used by Cray, IBM, Lenovo, Intel, etc.



Sparkitecture

Spark SQL
structured data

Spark Streaming
real-time

MLib
machine
learning

GraphX
graph
processing

Spark Core

Standalone Scheduler

YARN

Mesos

Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia, "Learning Spark: Lightning-Fast Big Data Analysis", O'Reilly, 2015.



Spark Streaming

- **Spark Core** tasks:
 - Memory management
 - Fault recovery
 - Implements the RDD (v1.0) and Dataset (v2+) Application Programming Interfaces (RDD API vs Dataset API)
 - One of the first platforms to guarantee consistent data analysis
- **Spark Streaming** is a consistent micro-batch processing environment for live streams of data
 - Note: it is important to notice that it is micro-batch oriented → its application is limited to use cases in which processing time windowing is acceptable
 - It was the first stream processing platform to provide strong consistency guarantees



Use cases

- The extension of Spark with Spark Streaming marked a turning point → the Lambda architecture became outdated (for many use cases) as soon as a single platform could handle both batch and stream processing consistently
 - Note: when Spark Streaming appeared, it additionally heated up the micro-batch vs streaming analysis debate(s)
- Common Spark Streaming use cases
 - In-order data
 - Event-time agnostic computations



STORM

Storm Intro



Definitions

- Apache **Storm** is a real-time, distributed stream processing platform
 - Designed as a directed acyclic graph (DAG) data transformation pipeline
 - Note: real-time is a key characteristic
 - Note: Storm was the 1st real streaming system (!)
- Comparable solutions:
Flink

Origins & History

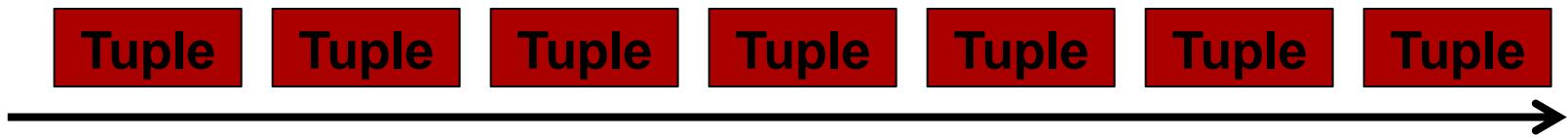
- Original author(s): Nathan Marz (@ BackType)
- Twitter acquired BackType
→ Storm in Twitter
- Written in: Clojure & Java
- Apache project: 2013
- License: Apache 2.0
- Initial release: September 2011
- Stable release: June 2020 (v2.2.0)



Motivation

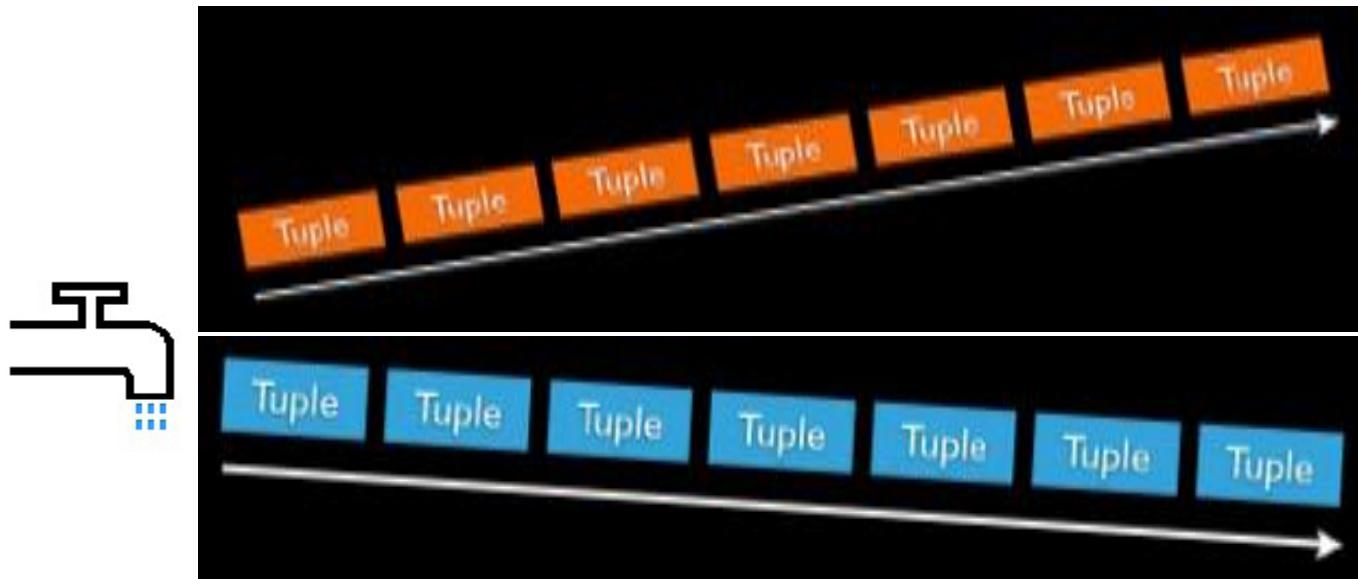
- How to handle the data flow in Twitter?
- How to obtain sufficiently low latency?
- Storm **design decisions**:
 - Combine at-most once and at-least once semantics with per-record processing
 - No internal notion of persistent state (i.e. non-consistent)
- These choices allowed Storm to provide lower latency than any other product on the marketplace in 2011
 - Eventual consistency (in the Twitter use case) was obtained by running a strongly consistent Hadoop cluster in parallel → Lambda architecture → years of dual-pipeline ‘darkness’
 - Note: Marz was the author of the Lambda architecture (!)

Streams



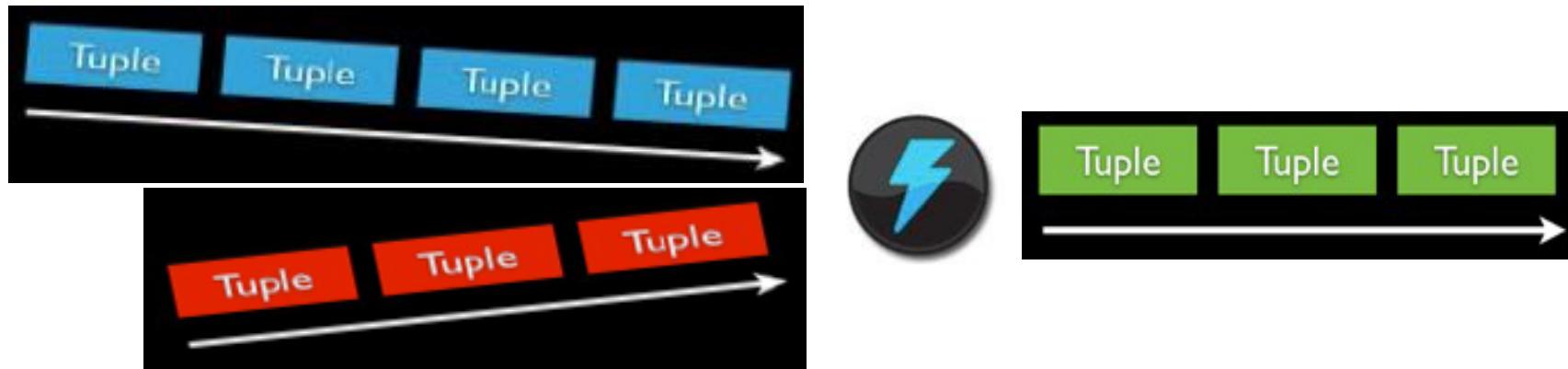
Unbounded sequence of tuples

Spouts



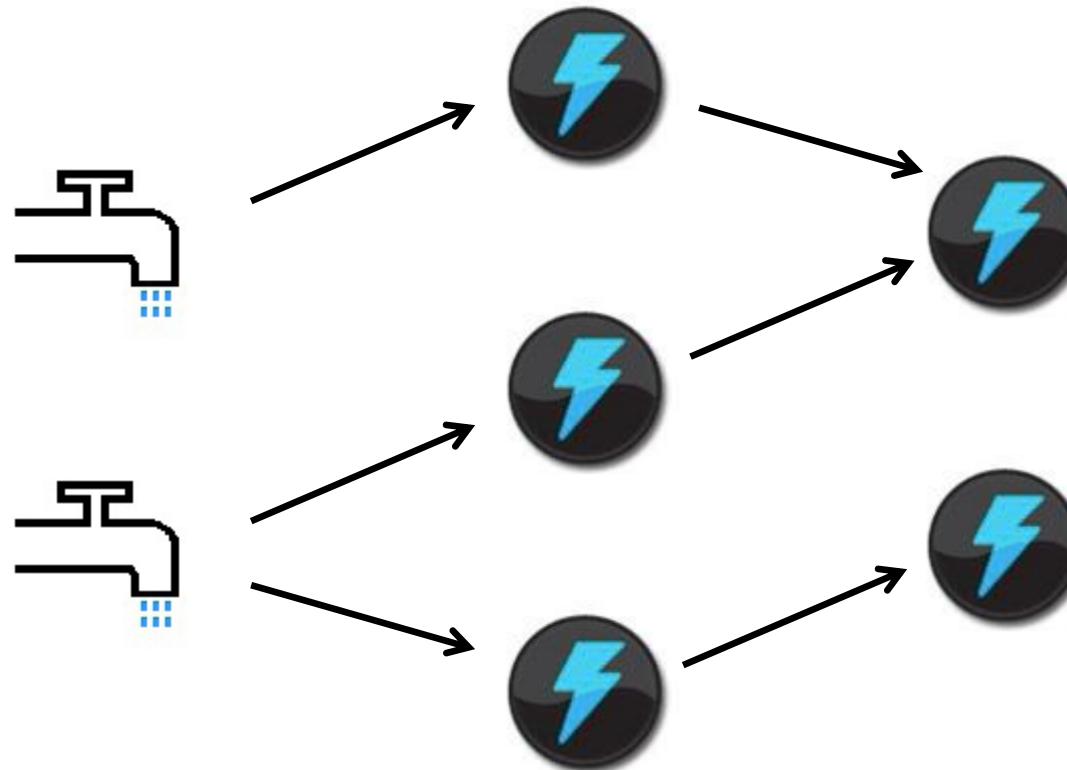
Source of streams

Bolts



Processes input streams and produces new streams:
Can implement functions such as filters, aggregation, join, etc

Topology



Network of spouts and bolts



Who uses the Storm?

Apache Storm vs Kafka Streams

Which companies use these tools?



Apache Storm



Spotify



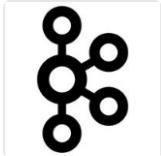
Twitter



Yelp



Keen IO



Kafka Streams



TransferWise



Red Bull Med...



Doodle



Bottega52



Life of Storm

- Twitter announced in 2015 that it was abandoning Storm as its in-house streaming system
- Twitter switched to Heron
- Heron maintained API-level compatibility with Storm
- Heron is an Apache incubator project
<https://github.com/apache/incubator-heron>
- Heron stable release: 0.20.1 (August 2019)
- Storm stable release: 2.2.0 (June 2020)





Additional references

- Twitter Storm Vs Heron
<https://medium.com/@vagrantdev/twitter-storm-vs-heron-12774056f45b>
- Apache Flink vs Twitter Heron?
<https://stackoverflow.com/questions/37635736/apache-flink-vs-twitter-heron>
- Leaving the Nest: Heron donated to Apache Software Foundation
https://blog.twitter.com/engineering/en_us/topics/open-source/2018/heron-donated-to-apache-software-foundation.html

FLINK



Flink quick facts

- **DEF:** Apache **Flink** is a unified stream and batch processing framework
- Flink programs consist of streams and transformations
- The Flink runtime supports iterative algorithms → ML
- Fault tolerance via distributed checkpoints
- Two APIs:
 - DataStream API for bounded or unbounded streams of data
 - DataSet API for bounded data sets
- Data source & sink connections: Kafka, HDFS, Cassandra, Elasticsearch





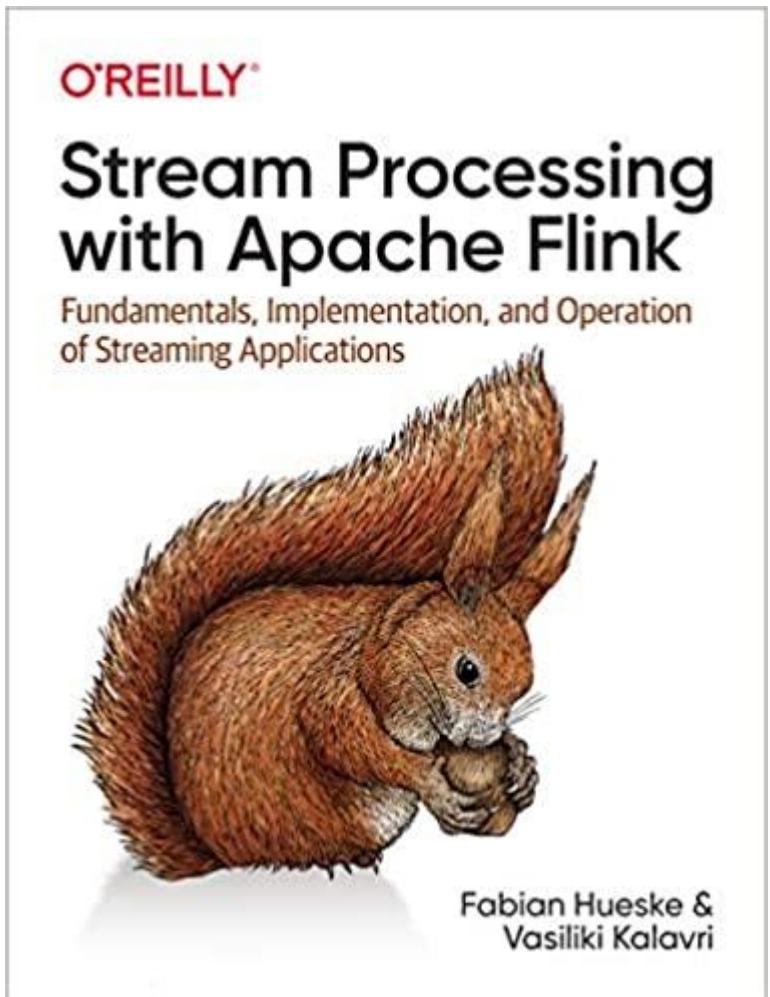
Flink background

- Original code based on the distributed **Stratosphere** project's distributed runtime
 - Stratosphere was financed by the German Research Foundation (DFG)
 - Stratosphere participants: Technical University Berlin, Humboldt-Universität zu Berlin, and Hasso-Plattner-Institut Potsdam
- **Initial release:** May 2011
- **Stable release:** 1.11.0 (July 2020)
- **Written in:** Java & Scala
- **License:** Apache License 2.0
 - Top level since Dec 2014
- Ververica (formerly Data Artisans) employs most of the key contributors



Flink references

- Hueske, F., & Kalavri, V. (2019). Stream Processing with Apache Flink: Fundamentals, Implementation, and Operation of Streaming Applications. O'Reilly Media.





Additional Flink references

- Savepoints: Turning Back Time
<https://www.ververica.com/blog/turning-back-time-savepoints>
- Apache Flink® — Stateful Computations over Data Streams
<https://flink.apache.org>
- Carbone, P., Ewen, S., Fóra, G., Haridi, S., Richter, S., & Tzoumas, K. (2017). State management in Apache Flink®: consistent stateful distributed stream processing. *Proceedings of the VLDB Endowment*, 10(12), 1718-1729.

Summary

- Spark **Streaming** → a microbatch processing platform with strong consistency guarantees
- Apache **Storm** is a real-time, distributed stream processing platform
- Apache **Flink** a unified stream and batch processing framework



Thank you for your attention!



Eötvös Loránd University (ELTE)
Faculty of Informatics (IK)
Pázmány Péter sétány 1/c
1117 Budapest, Hungary



VISUALIZATION SOLUTIONS

*Open-source Technologies for Real-Time Data
Analytics*

Imre Lendák, PhD, Associate Professor

Introduction

- Visualization types
 - Scientific viz
 - Information viz
 - Visual analytics
- Graph types
 - Line, bar, stacked bar, pie
 - Choropleth, scatter, heat
- Interactive visualization
- Viz tools: Tableau, Plotly, Datawrapper, Kibana, etc.



Data Visualization Process



Goals and data

- What is the goal of the visualization?
- What data do you have available?
- What level of detail does it go down to?
- How can you use other data to supplement your data?

Audience

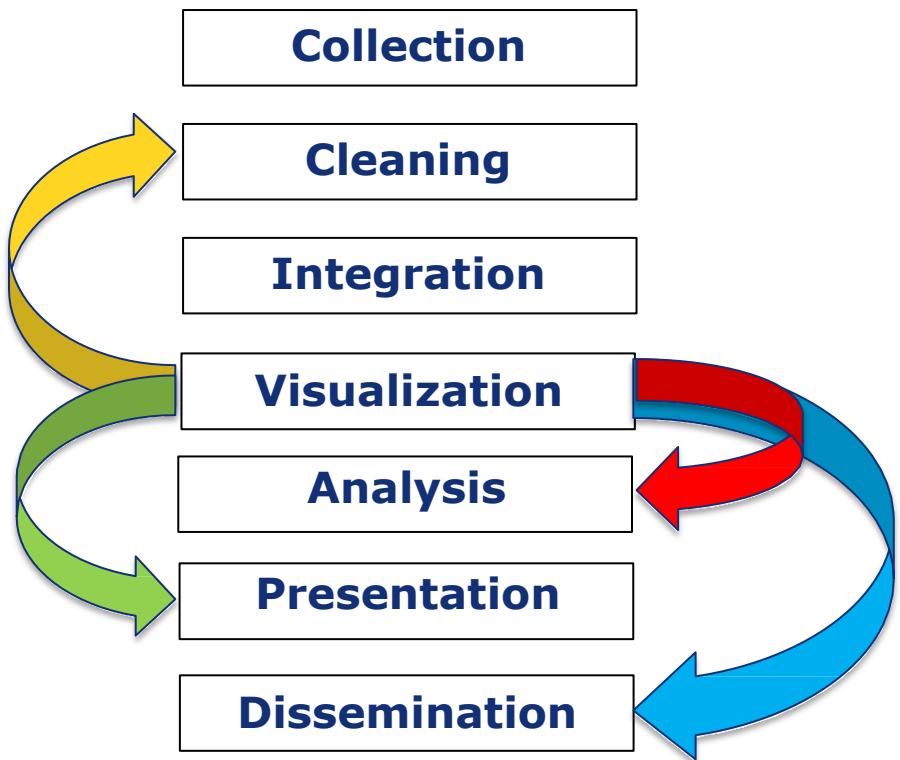
- How detailed do they want to see the data?
- Do they have a technical background?
- How will the visualization(s) be viewed? (desktop, mobile, print)

Data visualization goals (selection!)

- Reporting automation
- Executive reporting and presentation
- Customer reporting
- Self-service BI and visual data analysis
- Visual status monitoring
- Geo data visualization
- Visual data preparation
- Data journalism
- Math visualization
- Visual social media

Viz building blocks

- Collection → data acquisition
- Cleaning → data pre-processing
- Integration → merge data from different sources
- Visualization → create visual representations
- Analysis
- Presentation → create reports
- Dissemination → communicate





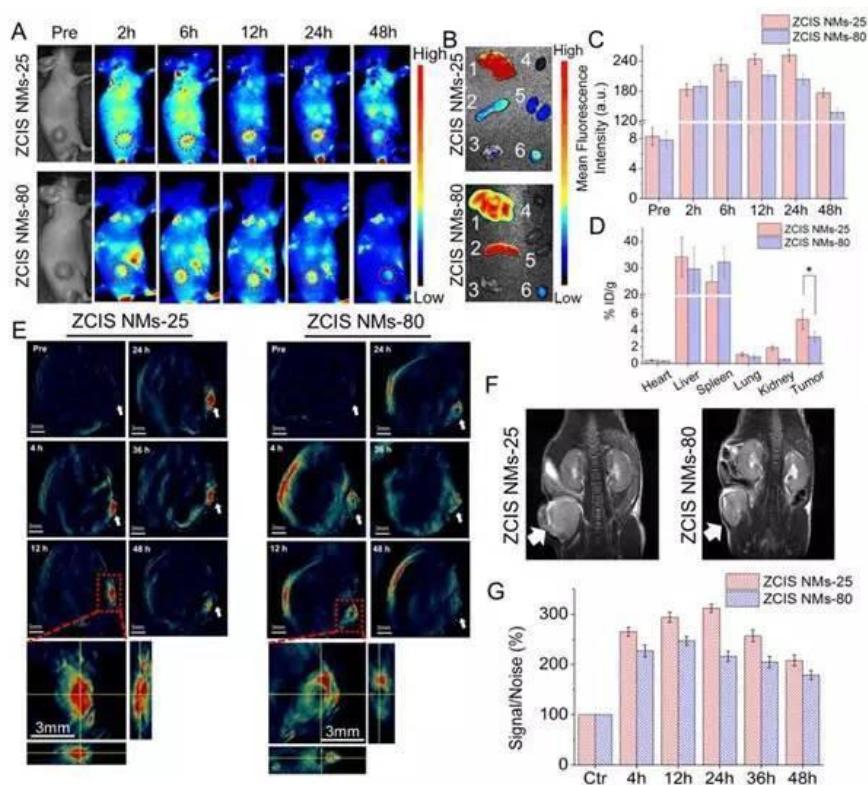
Pre-history of viz

- Selected figures
 - William Playfair ([1821](#)) – line, bar charts, etc.
 - Charles Joseph Minard ([1869](#)) – Napoleon's march, etc.
 - Jacques Bertin (1967) – “semiology of graphics”
 - John Tukey (1977) – “exploratory data analysis”
 - Edward Tufte (1983) – statistical graphics standards/practices
- 1985 NSF Workshop on Scientific Visualization
- 1990: S.K.Card, et al. Readings in Information Visualization: Using Vision to Think

VISUALIZATION TYPES

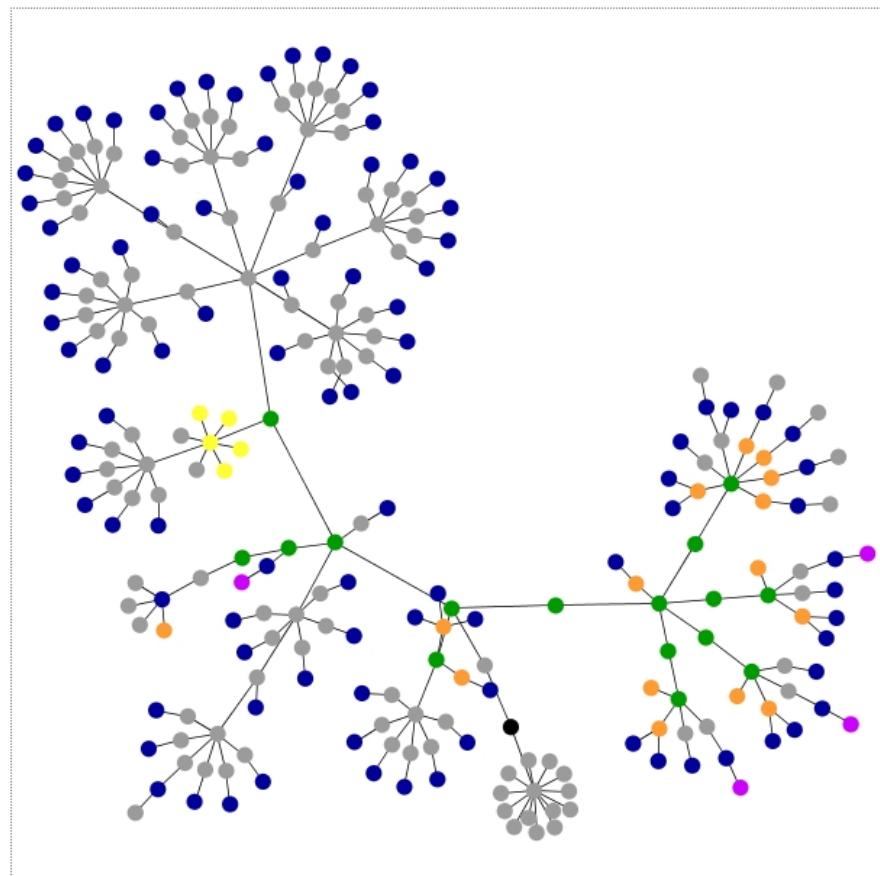
Type #1: Scientific visualization

- **DEF:** Scientific visualization focuses on the 2D or 3D visualization of scientific data.
- Used in:
 - architecture
 - meteorology
 - medicine
 - biological systems
 - ...



Type #2: Information Visualization

- **DEF:** Information visualization is the study of interactive visual representations of abstract data to enhance human cognition.
- Transforms abstract concepts into visually consumable information.
- Includes: histograms, trend graphs, flow charts, and tree diagrams



Type #3: Visual Analytics

- DEF: Visual analytics solutions allow analytical reasoning (usually about data) through an interactive visual interface (aka dashboard)



Dashboards in viz analytics



Introduction

- **DEF:** A data dashboard is an information management tool that visually **tracks, analyzes and displays** key performance indicators (**KPI**), metrics and key data points to monitor the health of a business, department or specific process
- A key goal of dashboards in general is to control performance, especially in a business environment, i.e. company
- Features:
 - Customizable to meet the specific needs of a department and company

Functionalities

- **Strategic planning**, e.g. impacts of new business line opened
- **Monitor efficiency**, e.g. lines of source code produced
- **Identify bottlenecks**, e.g. 3rd party supplier always late in delivery
- **Identify negative trends**, e.g. lower sales volume
- **Monitor efficiency of changes made**, e.g. new management installed



3 dashboard types

Strategic

- Develop, view and align company or institutional **strategy**
- Used by business developers and top managers

Tactical

- Measure the progress of **important** projects
- Used by project managers and mid- to top management

Operational

- Detailed monitoring of activities in (near) real-time
- Used by data analysts and up to mid-management, e.g. security analyst
- E.g. SIEM in a SOC



Dashboard in Windows OS

Control Panel

Control Panel > Control Panel >

Adjust your computer's settings

View by: Category

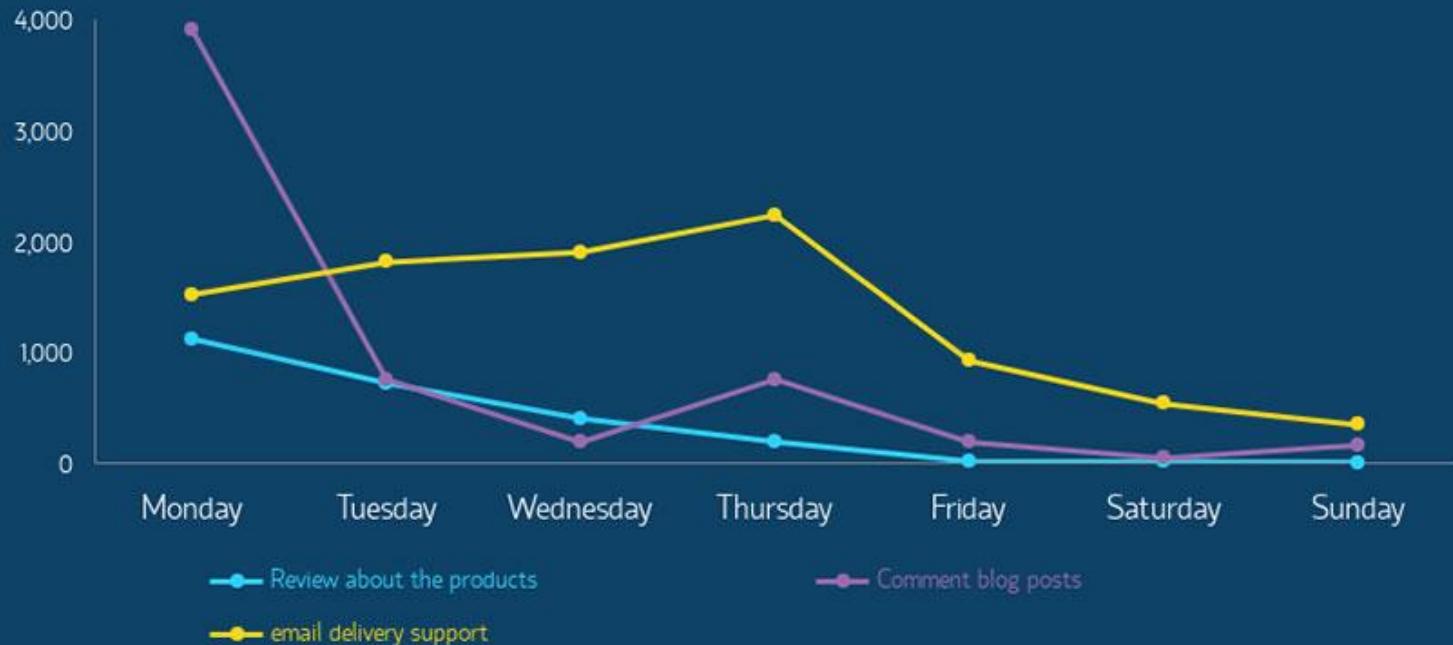
 System and Security Review your computer's status Save backup copies of your files with File History Backup and Restore (Windows 7)	 User Accounts Change account type
 Network and Internet View network status and tasks	 Appearance and Personalization
 Hardware and Sound View devices and printers Add a device Adjust commonly used mobility settings	 Clock and Region Change date, time, or number formats
 Programs Uninstall a program	 Ease of Access Let Windows suggest settings Optimize visual display

VISUALIZATION GRAPH TYPES



Line Graph

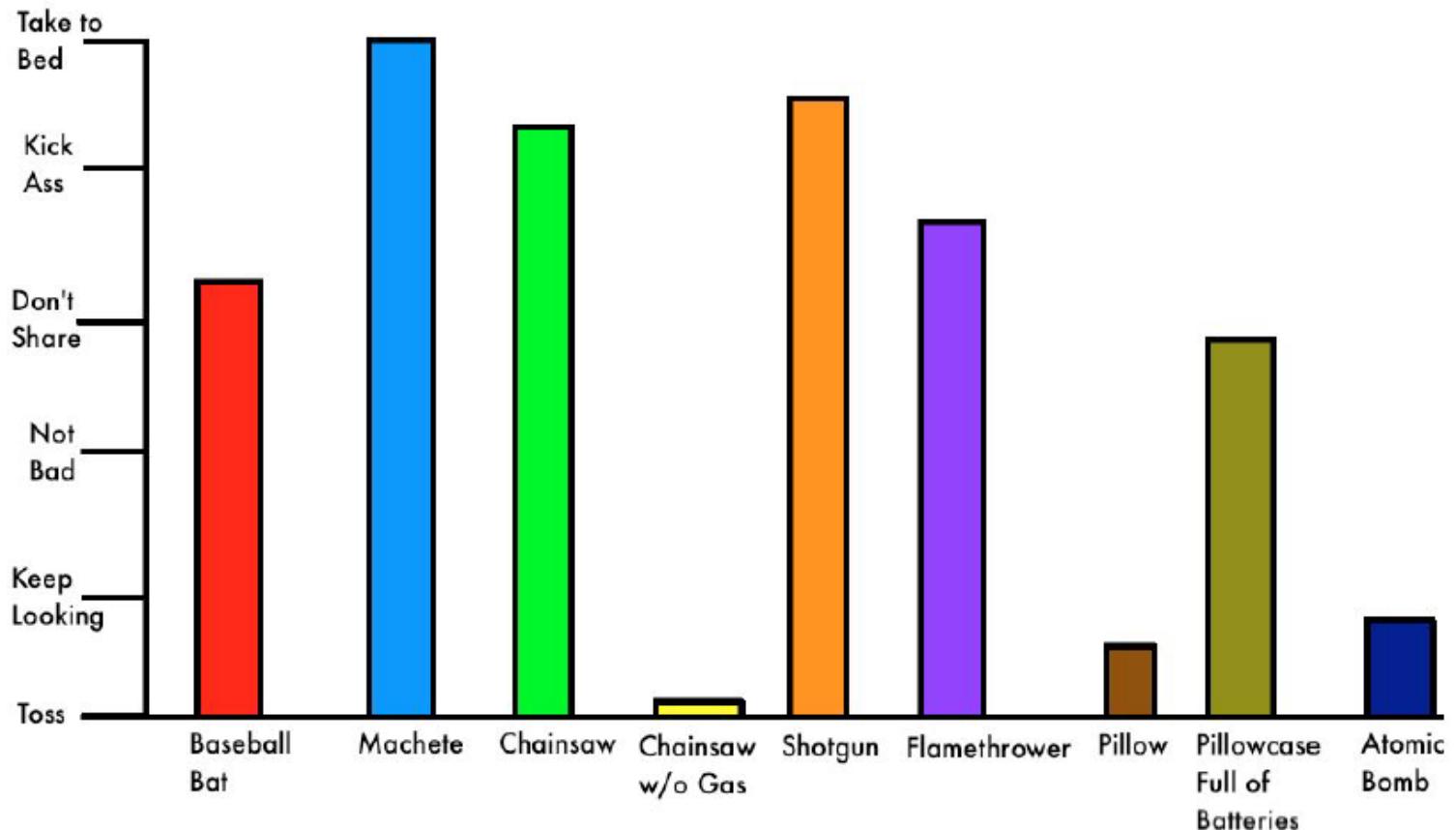
Customers activity During the week





Bar Graph

Usefulness of Weapons to Fight Zombies



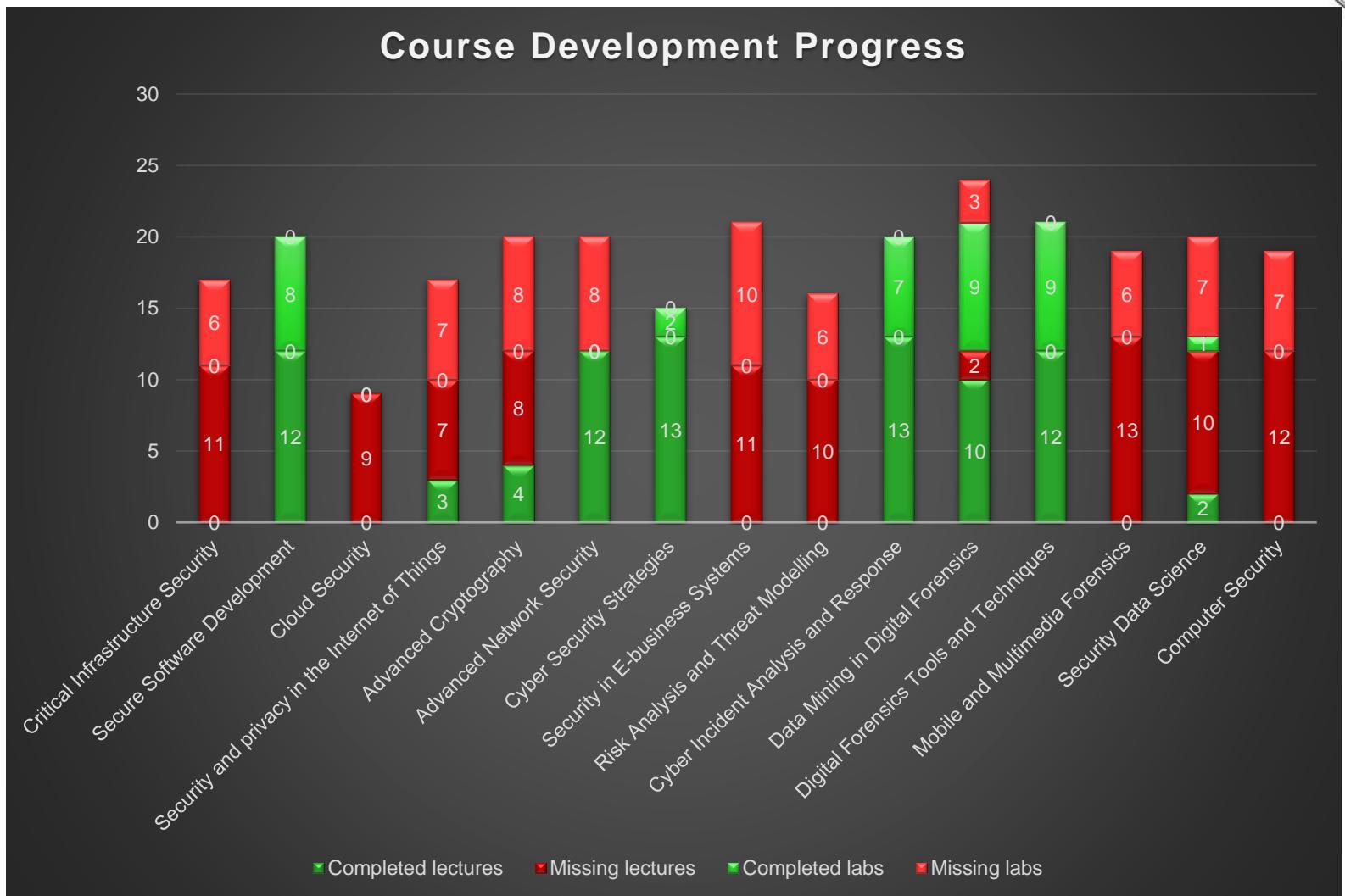


Pie Chart



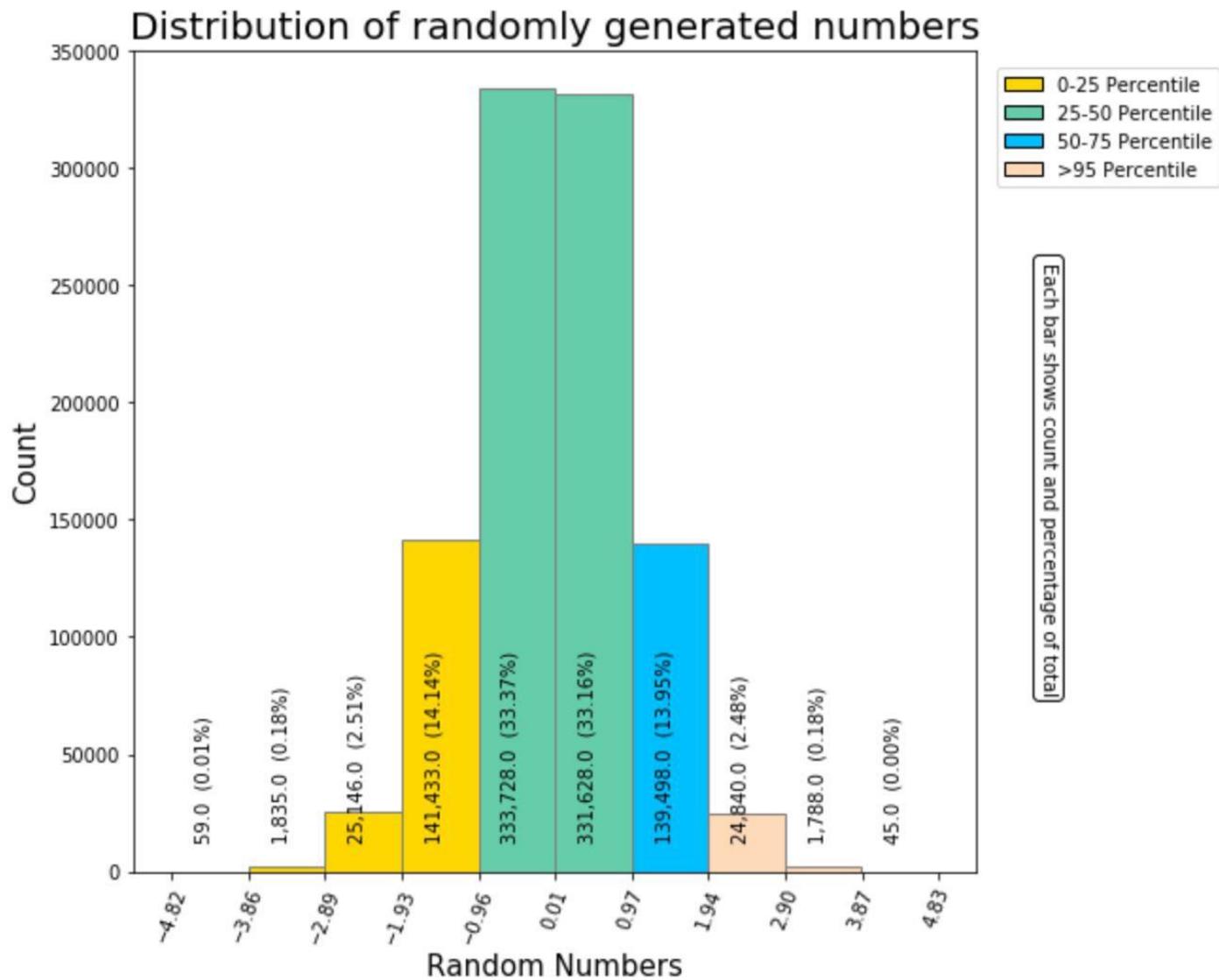


Stacked Bar Graph



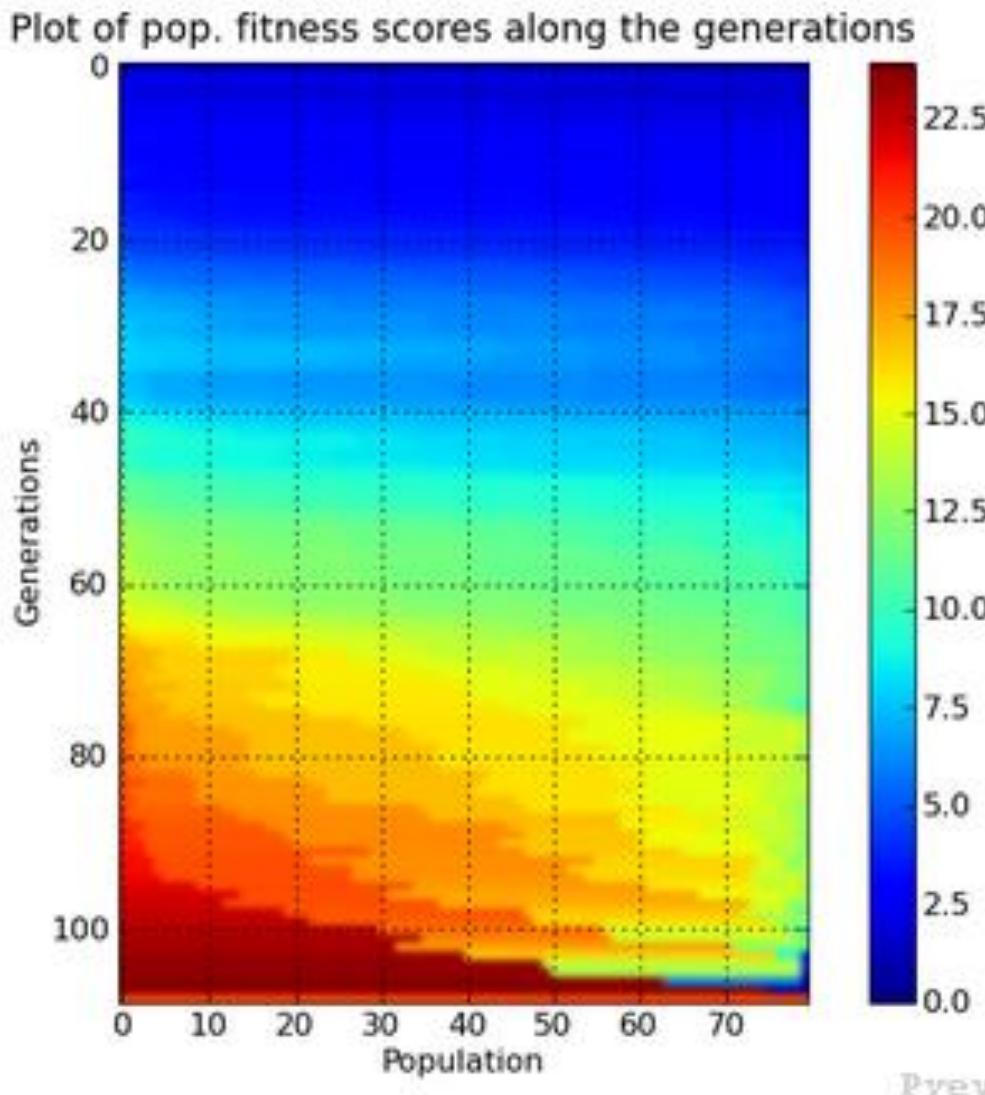


Histogram



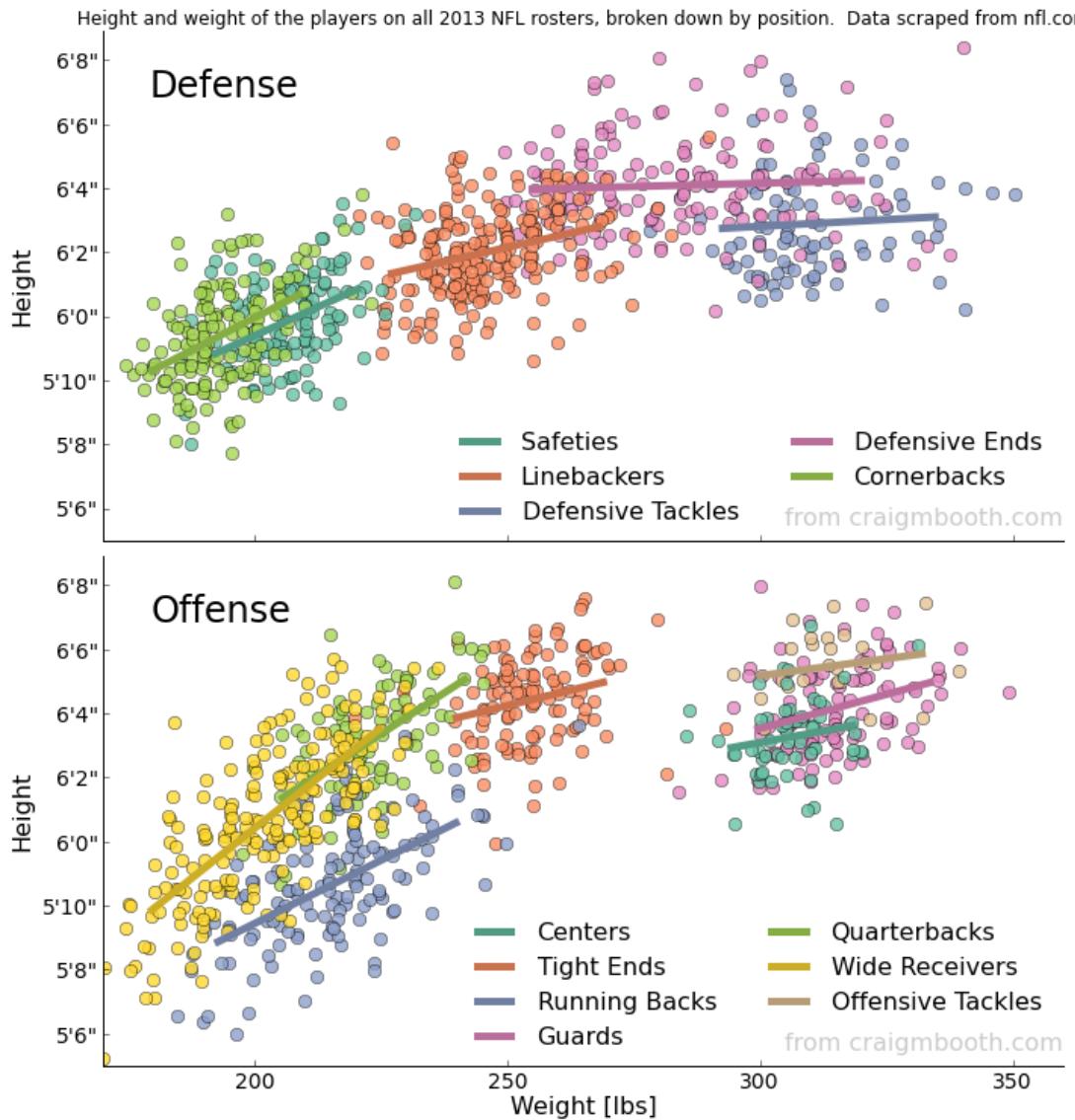


Heat Map





Scatterplot





Bubble chart

FILTERS

ABC City: Anchorage, Albany, A... (15) ▾

DRAG ABC OR HERE

MEASURE (X-AXIS)

Cart Additions M1
Sum of Cart Additions

MEASURE (Y-AXIS)

Checkouts M2
Sum of Checkouts

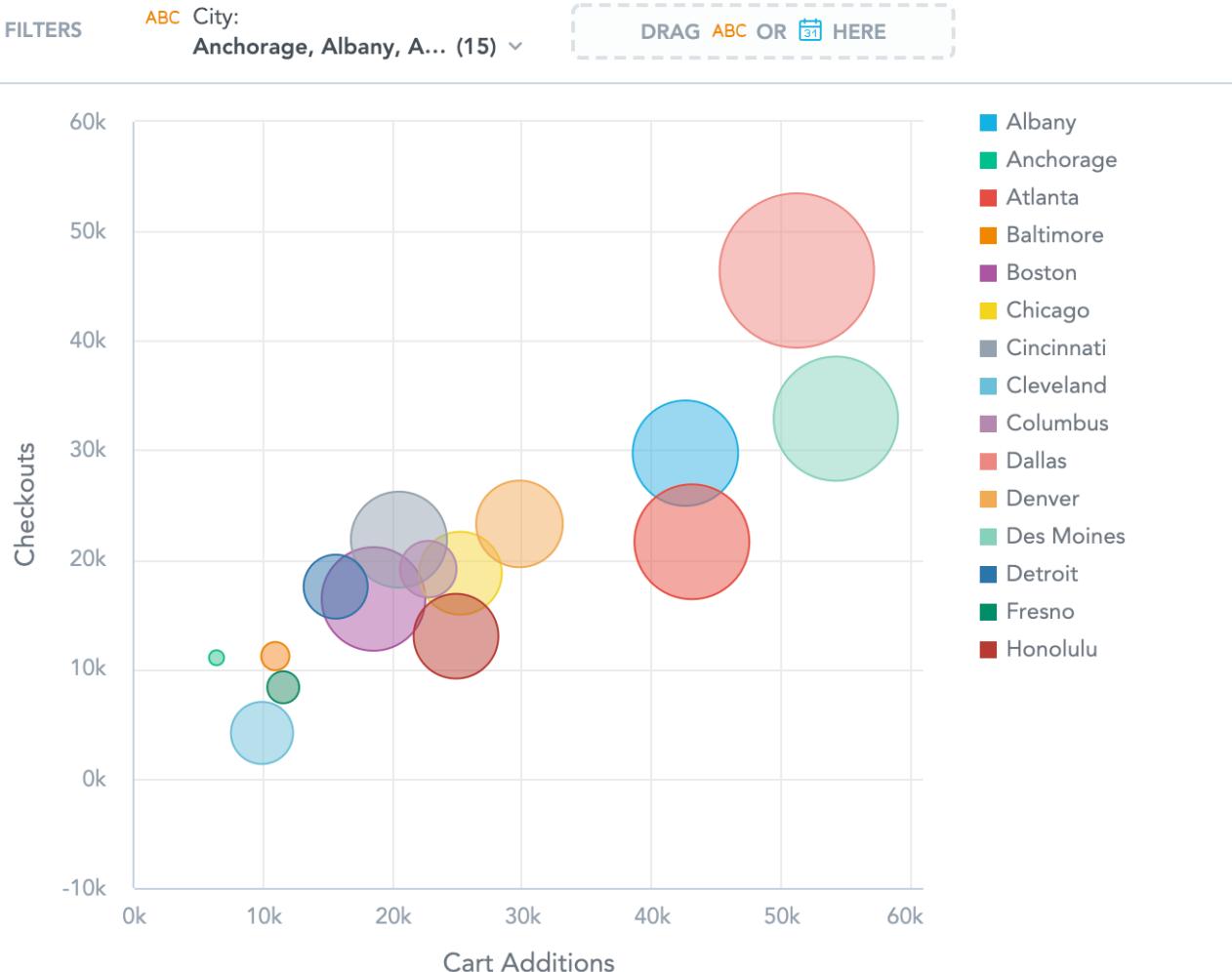
MEASURE (SIZE)

Spend M3

VIEW BY

ABC City

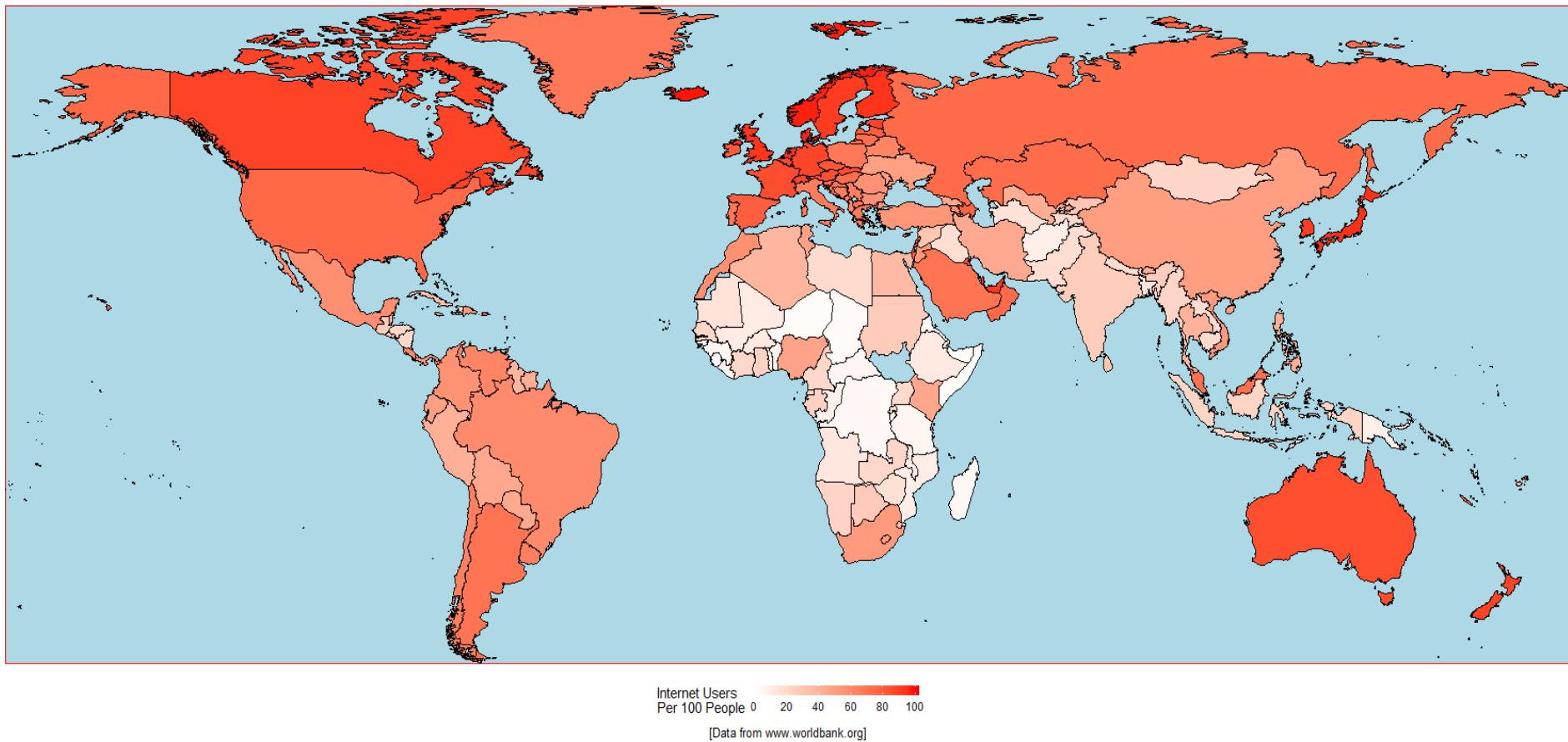
CONFIGURATION





Choropleth Map

Number of Internet Users per 100 People



https://www.reddit.com/r/dataisbeautiful/comments/6q811t/choropleth_world_map_of_internet_users100/



Graph functions, i.e. use cases

- Line → view trends (over time)
- Bar → compare categorical or time series points
- Pie → compare parts to a whole (up to 4-5 classes!)
- Stacked bar → pie chart alternative, supports more classes
- Histogram → view frequency/distribution
- Heat Map → color-coded frequency
- Scatterplot → relation of (at least) two variables
- Bubble → compare or rank
- Choropleth Map → shade/color on a geo map

CUSTOM VISUALIZATIONS

Olin Fellowship Alumnae in Law

Cecily Stewart-
Hawsworth

2019
Budapest, Hungary

Eötvös Loránd University (ELTE)
Faculty of Informatics (IK)
Pázmány Péter sétány 1/c
1117 Budapest, Hungary

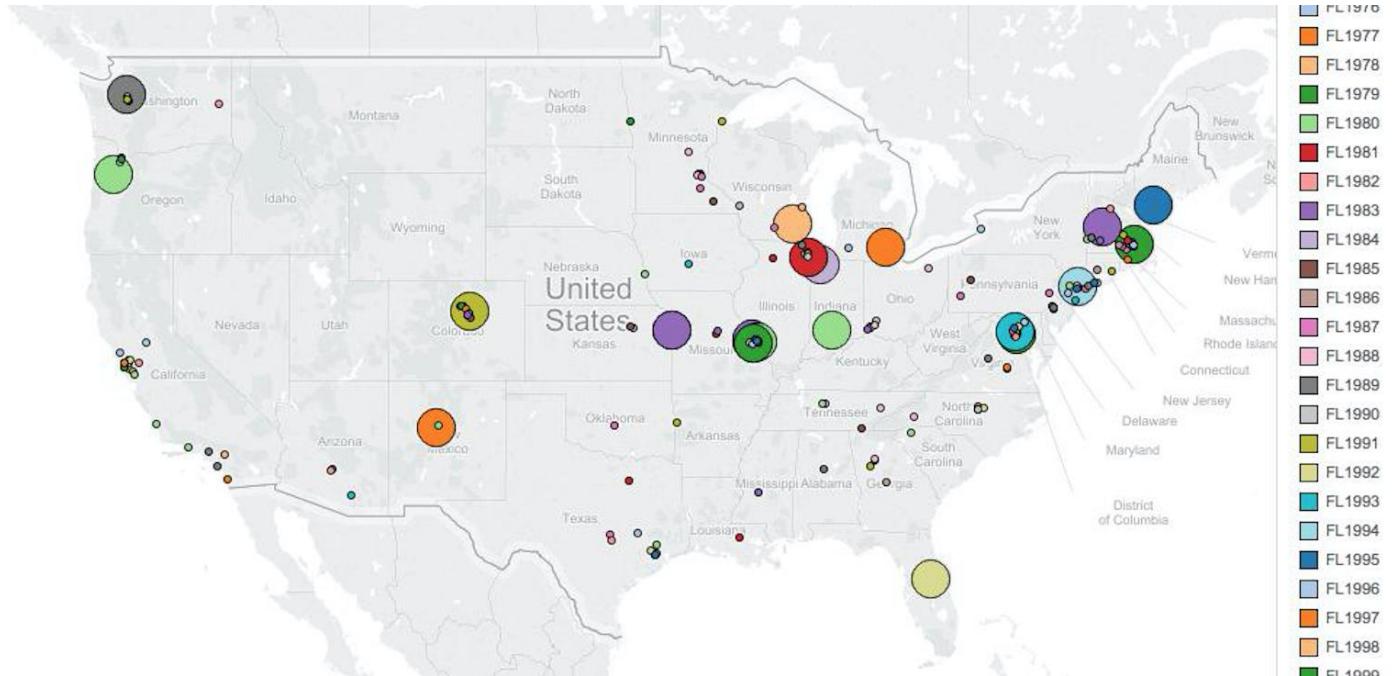


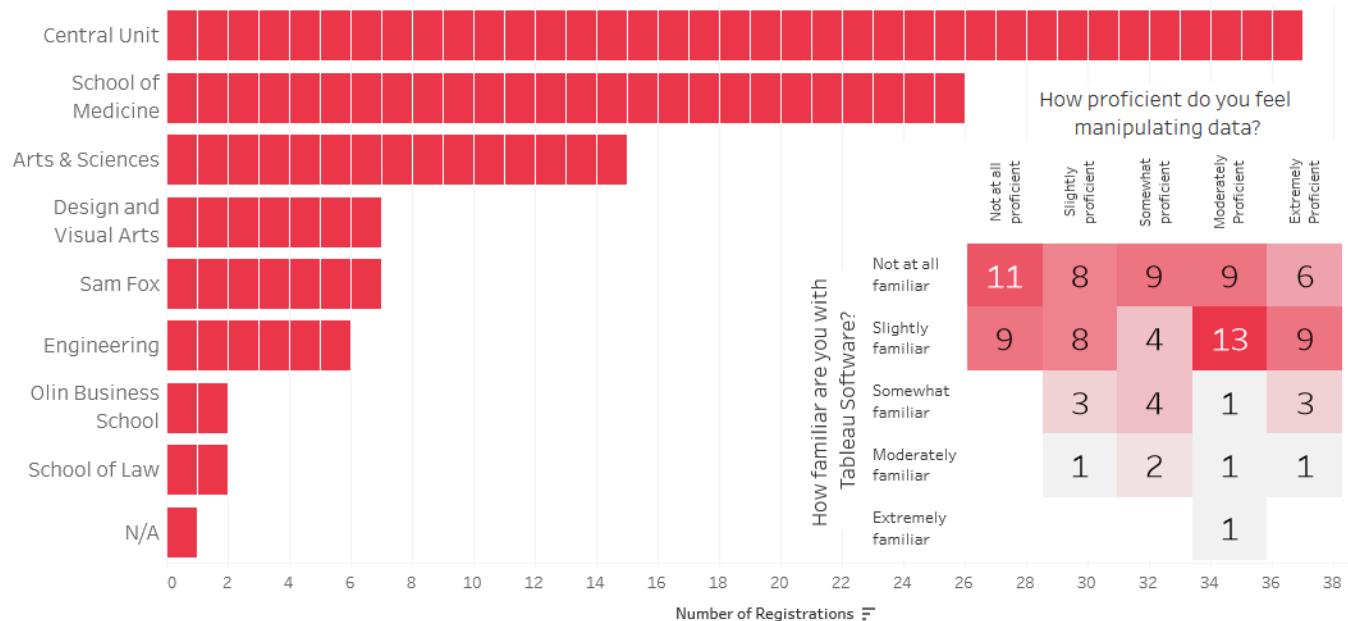


Tableau Bootcamp 2017: bringing together all seven schools and all four campuses

Survey and Attendance Data

Erin Daugherty

2019
Budapest, Hungary



INTERACTIVE VISUALIZATION



Interactive Visual Analytics

**Data
preprocessing
through visual
approaches**



- Data mining
- Machine learning
- Statistical methods



▪ Bring out meaningful:

- patterns
- outliers
- clusters
- gaps

**Interactive
visualization**



- Browse
- search
- monitor



- Discover the most interesting relationships among data
- Investigate what-if scenarios
- Verify the presence of biases
- Simulate changes impact

**Dissemination
tools**



- Show the data



- Enlighten the sense of data
- *Tell stories* about them



Interactive visualization

- Select (mark something as interesting)
- Explore (show me something else)
- Reconfigure (show me a different arrangement)
- Encode (show me a different representation)
- Abstract/elaborate (show me more or less detail)
- Filter (show me something conditionally)
- Connect (show me related items)



Interactive visualization

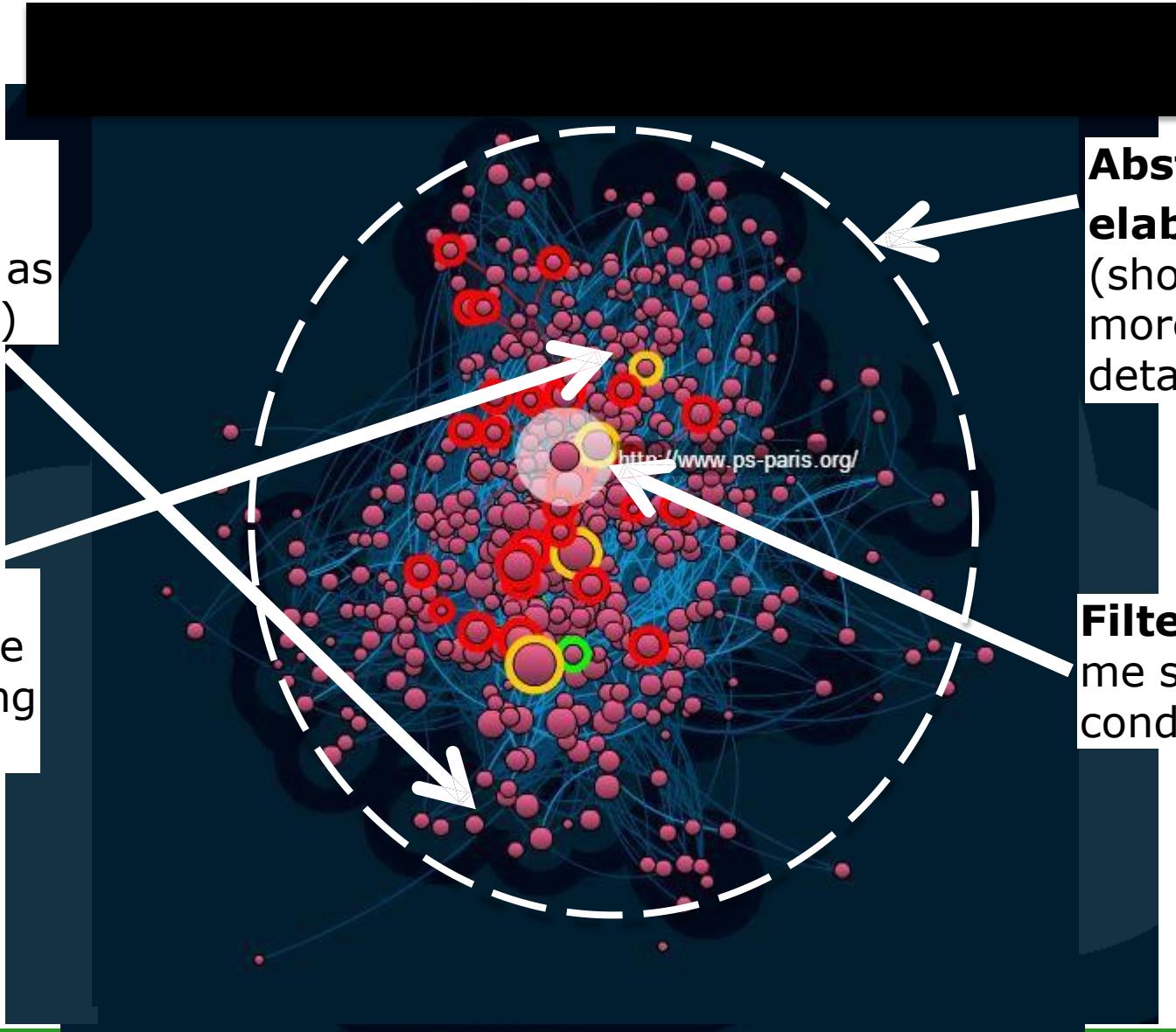
Select	Ability to mark data items of interest to highlight them	Outlier values
Explore	Enabling users to examine the different subsets in which the data can be divided	Panning across the data
Reconfigure	Provide users with different data perspectives	<ul style="list-style-type: none">• Revelation of hidden patterns• visual rearrangements of a series
Encode	Capability of a visualization system to handle and transform the basic elements of human vision	Pre-attentive processing, colours, shapes, dimensions
Abstract/elaborate	Capability of reduce or increase the details of the visualization	
Filter	Highlight some visual elements that are compliant with specific conditions defined by users	
Connect	Enables users to better emphasize relationships and associations already known or discover the hidden patterns of the data	

Select
(mark something as interesting)

Explore
(show me something else)

Abstract/elaborate
(show me more or less detail)

Filter (show me something conditionally)



VIZ TOOLS



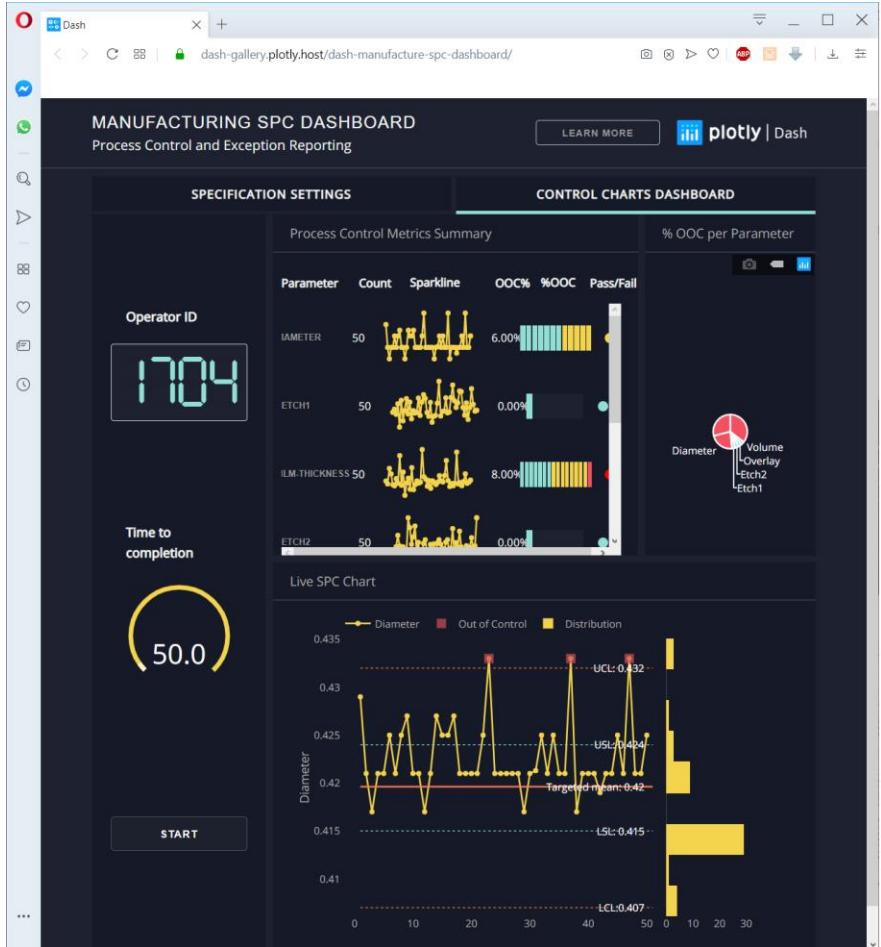
Introduction

- **DEF:** Data visualization software that allows developers to build interactive dashboards that are easily updated with new data and can be shared with a wider audience
 - Developer: Tableau Software Inc, California, USA
 - License: commercial, available for academic use
 - Link: <https://www.tableau.com>
 - Good: usable for data analysts with minimum programming experience
 - Bad: some licenses paid

Licensing

- Instructors and Researchers
 - Free Desktop license for a year (renewable)
 - Some caveats apply
 - <https://www.tableau.com/academic/teaching/course-llicenses>
- Students
 - Free Desktop license for a year (renewable)
 - <https://www.tableau.com/academic/students>

Plotly



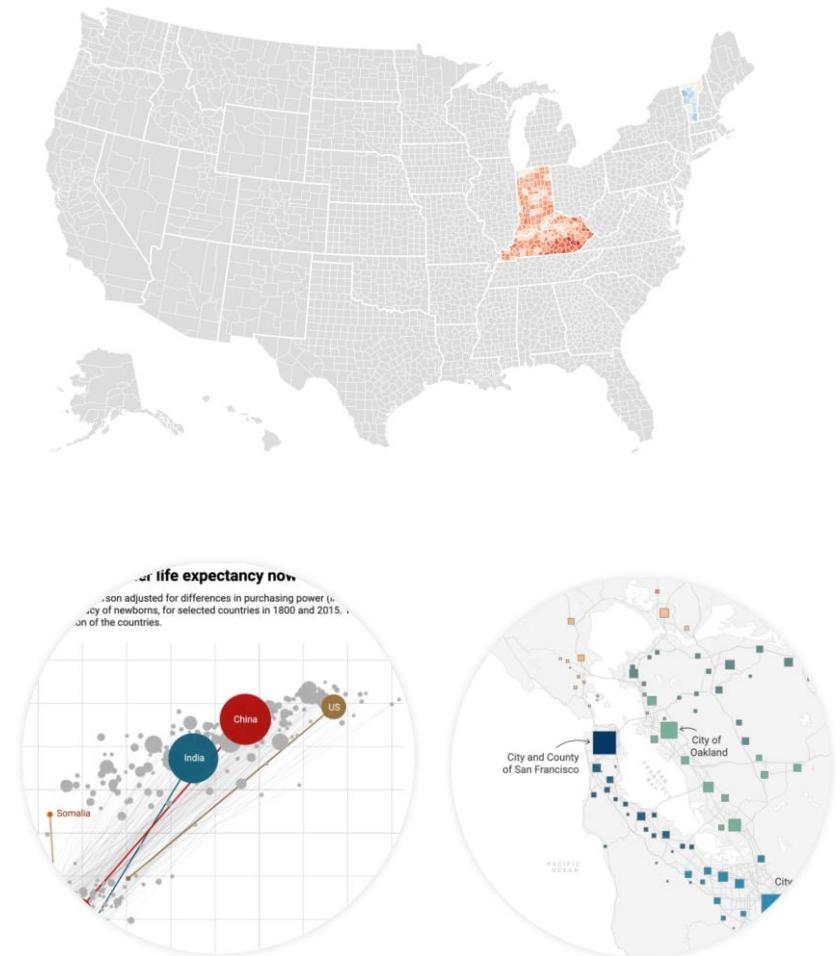
<https://dash-gallery.plotly.host/Portal/>

- **DEF:** Web-based platform for operationalizing Python & R models
- **Product:** Plotly Dash
- **Features:**
 - 2D and 3D charts
 - Designer input, i.e. visual customizability
 - Analytics language integrations: Python, R and Matlab
 - Built-in APIs
- **License:** open source, MIT license
- **Used by:** Amazon, Shell, Cisco, Pirelli
- **Link:** <https://plot.ly>

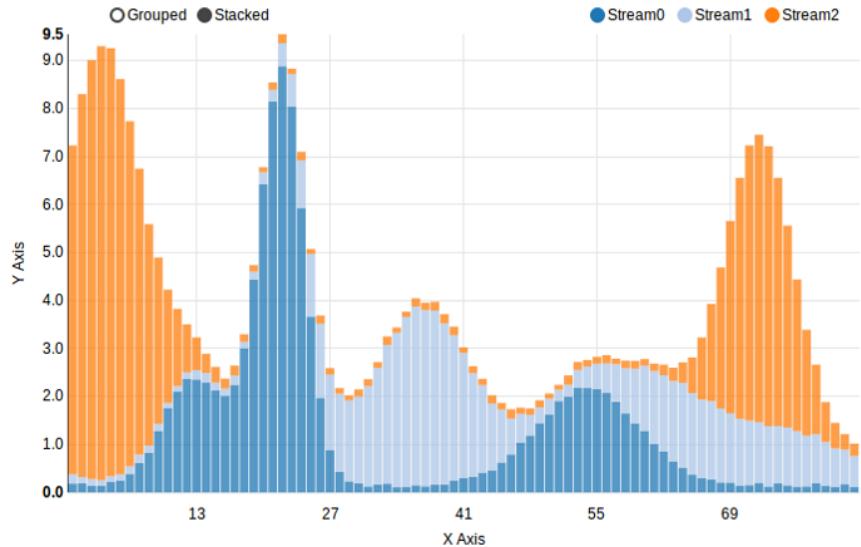
Datawrapper

- **DEF:** Datawrapper is an open-source web tool for basic interactive charts.
- **Features:**
 - Free and no sign-up needed
 - 19 chart types, 3 maps + tables (all interactive)
 - Minimum coding skills required
 - Minimum design skills required
 - Interactive charts
- **License:** MIT license
- **Used by:** Fortune, The New York Times, Wired, Süddeutsche Zeitung
- **Link:**

<https://www.datawrapper.de>



D3.js



- **DEF:** D3.js is a JavaScript library for web-based visualizations
 - Data-Driven Documents
- **Features:**
 - Web-based
 - Interactive viz
 - Downloadable from Github
- **License:** BSD
- **Formats:** Scalable Vector Graphics (SVG), HTML5, and Cascading Style Sheets (CSS)
- **Used by:** Coursera, Akamai
- **Link:** <https://d3js.org>





Google chart

- **DEF:** Google Charts is Google's big data visualization platform
- **Features:**
 - Completely free
 - Web-based
 - Supported by Google
 - Simple viz types
 - Multi-dimensional viz types
 - Interactive viz
- **License:** Apache 2.0
- **Used by:** BBC, Esquire
- **Link:**
[https://developers.google.com/c
hart](https://developers.google.com/chart)



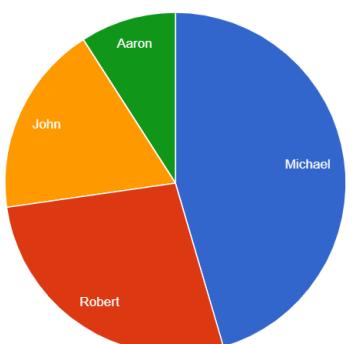
Donuts eaten per person

Age Filter:

3.0 54.0

Gender Selection:

Male



Name	Gender	Age	Donuts eaten
Michael	Male	12	5
Robert	Male	7	3
John	Male	54	2
Aaron	Male	3	1



Kibana (part of Elasticsearch)

Kibana features

- Open source data visualization tool
- Visualize ES documents
- Real-time dashboard
- Supports advanced data analytics
- Historical data visualization

Additional features

- Alerting solutions
 - Yelp's ElastAlert
 - Elastic's Watcher
- Shield - authentication and authorization for Kibana

<https://logz.io/blog/5-features-weve-added-to-kibana/>



More viz tools (A to Z)

- Chartio, <https://chartio.com>
- Domo, <https://www.domo.com>
 - Used by: TripAdvisor, Cisco, etc.
- Geckoboard, <https://www.geckoboard.com>
- Klipfolio, <https://www.klipfolio.com>
- Sisense, <https://www.sisense.com>
 - Used by: NASA, NASDAQ, Samsung
 - Merged with: Periscope Data

Summary

- Visualization types
 - Scientific viz
 - Information viz
 - Visual analytics
- Graph types
 - Line, bar, stacked bar, pie
 - Choropleth, scatter, heat
- Interactive visualization
- Viz tools: Tableau, Plotly, Datawrapper, etc.
 - + other viz tools in 2020 listed in the OST intro



Thank you for your attention!



Eötvös Loránd University (ELTE)
Faculty of Informatics (IK)
Pázmány Péter sétány 1/c
1117 Budapest, Hungary



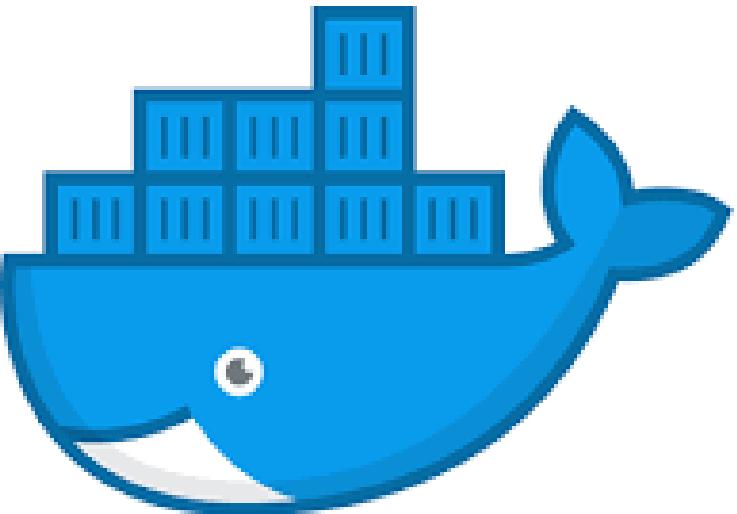
DOCKER TUTORIAL

*Open Source Technologies for Real-Time Data
Analytics*

Imre Lendák, PhD, Associate Professor

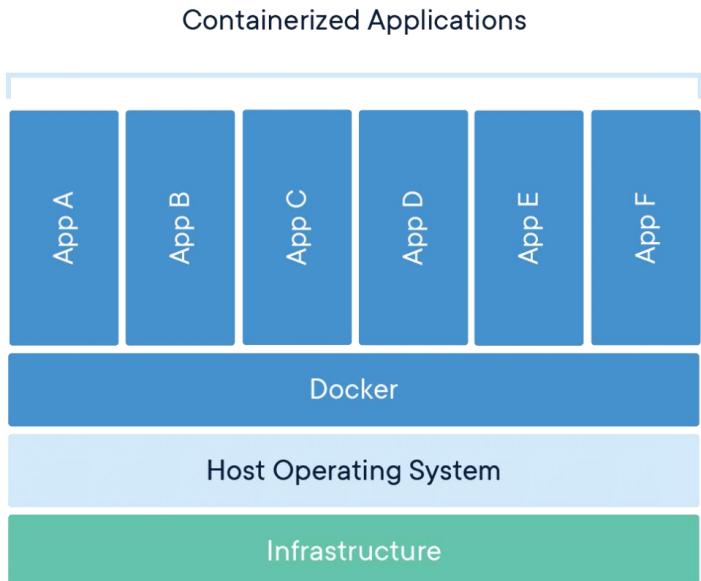
Introduction

- Introduction
- Working with Docker
 - Basic commands
 - Images
 - Networking
- Docker ecosystem
 - Docker Hub
 - Docker Compose
 - Kubernetes
- Security





Container intro

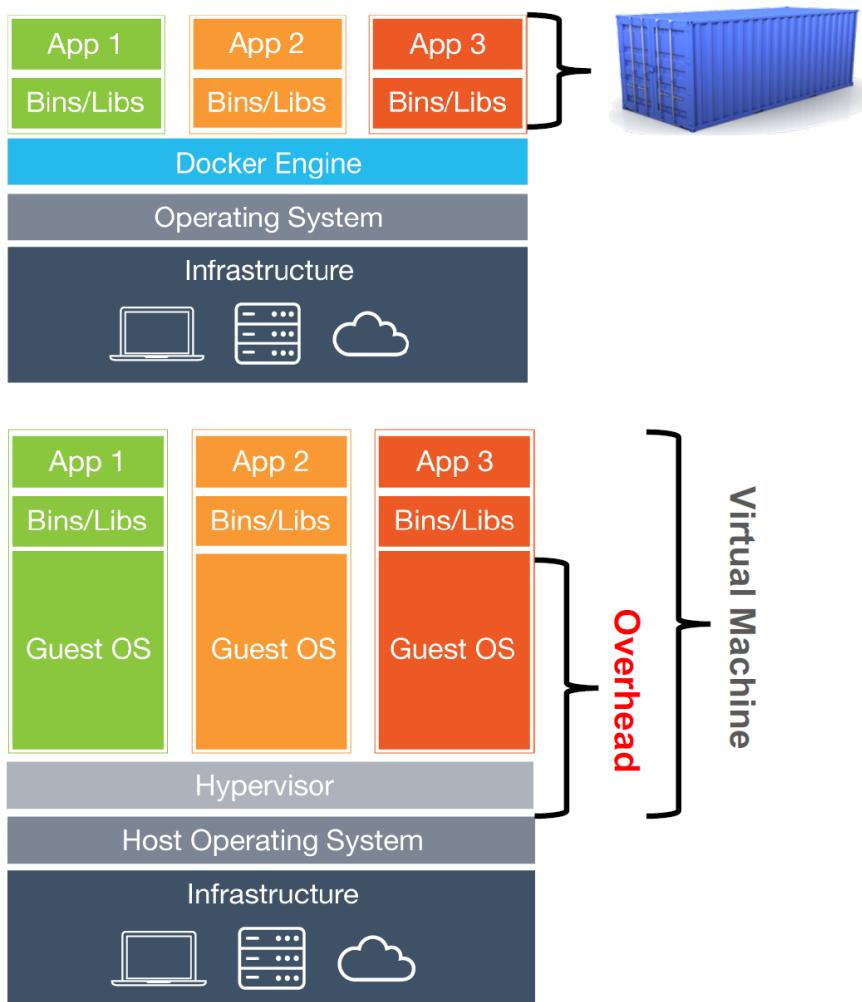


- **DEF:** A **container** is a lightweight, standalone, executable package of software that includes everything needed to run an application:
 - code,
 - runtime,
 - system tools,
 - system libraries and
 - settings.



Docker intro

- **DEF:** Docker is an open-source project that automates the deployment of software applications inside containers
 - Virtualization of application (i.e. process) instead of hardware
 - Runs on top of the core OS (Linux or Windows)
 - No dedicated CPU, memory, network requirements as in VMs
- Trivia:
 - **Initial release:** March 20th, 2013
 - **Latest stable release:** 19.03.13 (Sept 16, 2020)
 - **Original author(s):** Solomon Hykes
 - **License:** Apache License 2.0



Motivation



	Bare-metal	Virtual machine	Container
Underlying platform	N/A	Hypervisor on bare-metal	OS on VM or bare-metal
Performance	Best	Average	Average
Provisioning time	At least hours	Minutes	Seconds
Tenant isolation	Lowest (wo host OS)	Average (?)	Maximum (?)
Configuration flexibility	None	Average	Best
App portability	Backup & restore, ISO images	VM image, VM tools	Best
Granularity	Large	Average	Small

https://www.snia.org/sites/default/files/CSI/SNIA_Intro_to_Containers_Container_Storage_and_Docker_Final.pdf



Terminology

- **Images** are application blueprints which form the basis of containers
- **Containers** are created from Docker images and run the actual application. We create a container using 'docker run'
- The **Docker Daemon** is the background service running on the host that manages building, running and distributing Docker containers. The daemon is the (only) process that is run by the core OS
 - The daemon is part of the Docker Engine
- **Docker Client** is the command line tool that allows the user to interact with the daemon. There can be other forms of clients apart from Docker Client
- **Docker Hub** is a registry of Docker images



Docker installation

- The Docker platform consists (at least) of the following components
 - Docker Engine – the service capable to run processes in sandboxes inside a core OS
 - Docker Client – the client app used to manage images
 - Docker was originally built to run on Linux
- The Docker Desktop package is today (December 2020) available for Windows and Mac
 - URL: <https://www.docker.com/products/docker-desktop>
 - Docker Desktop contains both the Engine and Client
 - Installation steps: download, install & run the Engine

DOCKER HUB



Docker Hub

- DEF: **Docker Hub** is a registry of Docker images
 - Official Docker Hub URL:
<https://hub.docker.com/search?q=&type=image>
- If required anybody can set up his/her Docker registry to store & pull images
 - Software producers often publish their own images

The screenshot shows the Docker Hub search interface with the following details:

- Search Bar:** Search for great content (e.g., mysql)
- Navigation:** Explore, Pricing, Sign In, Sign Up
- Filters:** Docker, Containers, Plugins
- Results:** 1 - 25 of 4,340,041 available images.
- Items Listed:**
 - Oracle Database Enterprise Edition** (Verified Publisher)
By Oracle • Updated 3 years ago
Oracle Database 12c Enterprise Edition
Container, Linux, x86-64, Databases
 - Oracle Java 8 SE (Server JRE)** (Verified Publisher)
By Oracle • Updated 10 months ago
Oracle Java 8 SE (Server JRE)
Container, Linux, x86-64, Programming Languages
 - couchbase** (Official Image)
Updated 11 hours ago
Couchbase Server is a NoSQL document database with a distributed architecture.
Container, Linux, x86-64, Storage, Application Frameworks
 - postgres** (Official Image)
Updated 11 hours ago
The PostgreSQL object-relational database system provides reliability and data integrity.
Container, Linux, 386, ARM, IBM Z, x86-64, mips64le, PowerPC 64 LE, ARM 64, Databases
 - traefik** (Official Image)
Updated 11 hours ago
Traefik, The Cloud Native Edge Router
Container, Windows, Linux, ARM 64, x86-64, ARM, Application Infrastructure

BASIC COMMANDS



Basic commands

- `$ docker version` – check Docker version
 - `$ docker search` – search the Docker Hub (DH) for images
 - `$ docker pull` – pull image(s) from a repository (e.g. DH)
 - `$ docker images` – command for working with images
 - `$ docker container` – command family for working with containers
-
- `$ docker run` – run a container from an image
 - `$ docker ps` – list active containers
 - `$ docker attach` – attach to a container
 - `$ docker exec` – execute a command in a container
 - `$ docker kill` – kill one or more docker containers

IMAGES



Working with images – 1

- `$ docker pull elasticsearch:7.9.0`
 - Pull the elasticsearch image from Docker Hub
- `$ docker pull docker.elastic.co/elasticsearch/elasticsearch:6.3.2`
 - Pull from Elastic's image registry
- `$ docker images`
 - List locally available images
- `$ docker build`
 - Build docker image
- `$ docker run`
 - Run an image in a local sandbox → results in a container (!)



Working with images – 2

- \$ docker ps
 - Show all containers that are currently running on a core OS
- \$ docker ps -a
 - Show all containers that we ran
- \$ docker container ls
 - List running containers
- \$ docker container prune
 - Remove containers



Hands-on #1: Elasticsearch

- \$ docker pull elasticsearch:7.9.0
 - Note: pull ES does not work without specifying the required version
- \$ docker run --rm -d --name es -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node" elasticsearch:7.9.0
 - Note: --name allows us to name (i.e. tag) the container
 - Note: -p allows us to create port mappings between the container and the core OS
 - Note: --rm allows us to remove the named container after use → easier to re-run it afterwards, i.e. no need to remove it before re-running
- \$ docker container logs es
 - Check the logs of the container
- \$ docker ps
- \$ curl -X GET "localhost:9200"
 - Check whether it works → should respond with elasticsearch cluster info (in JSON)
 - Note: curl is an open-source, command-line tool for transferring data with URLs
 - Note: curl should be installed prior to issuing the above command
- \$ docker container stop es
- \$ docker ps

<https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html>



Hands-on 2 & 3 directory structure

Directory structure

python_client

 Dockerfile

 python_client.py

 requirements.txt

python_server

 Dockerfile

 python_server.py

 requirements.txt

Comments

- Requirements installed via dockerfiles:
 - Linux packages
 - Python libraries
- File types usually copied to containers:
 - Configuration
 - Binary
 - Source code, e.g. Python code in our case
- Usually a single command is run in the container with CMD



Hands-on #2: Python service

Python server

```
import datetime
import socketserver
import time

class MyTCPHandler(socketserver.BaseRequestHandler):
    def handle(self):
        # self.request is the TCP socket connected to the client
        has_error = False
        try:
            self.data = self.request.recv(1024).strip()
            self.data = bytes(self.data)
            print(datetime.datetime.now(), " - {} wrote:".format(self.client_address[0]))
            print(' data: ', str(self.data))

            server_hello = "Python Server is up and running."
            send_data = server_hello.to_bytes()
            self.request.sendall(send_data)
        except:
            print('Exception occurred in handle.')

if __name__ == "__main__":
    HOST, PORT = "localhost", 9999
    with socketserver.TCPServer((HOST, PORT), MyTCPHandler) as server:
        server.serve_forever()
```

Dockerfile

```
FROM python:3

RUN apt-get update && apt-get upgrade -y
RUN apt-get install -y sudo

WORKDIR /usr/src/app

COPY requirements.txt ./requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Secure against running as root, but allow sudo (for tcpdump)
RUN adduser --disabled-password --gecos "python_server"
RUN adduser python_server sudo
RUN echo '%sudo ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers
WORKDIR /home/python_server

COPY python_server.py /python_server.py

EXPOSE 9999

CMD [ "python", "/python_server.py" ]
```



Hands-on #3: Python client

Python client

```
import socket
import os

if __name__ == "__main__":
    HOST, PORT = "localhost", 9999
    HOST, PORT = "192.168.10.10", 9999

    binary_message = bytes([0, 0, 0])

    # Create a socket (SOCK_STREAM means a TCP socket)
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        # Connect to server and send data
        sock.connect((HOST, PORT))
        sock.sendall(binary_message)
        # Receive data from the server and shut down
        received = sock.recv(1024)
        print(received)
```

d

Dockerfile

```
FROM python:3

RUN apt-get update && apt-get upgrade -y
RUN apt-get install -y sudo
RUN apt-get install -y openssh-server nmap iputils-ping

COPY requirements.txt ./requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Secure against running as root
RUN useradd -rm -d /home/python_client -s /bin/bash -g root -G
sudo -u 1000 python_client
RUN echo '%sudo ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers
RUN echo 'python_client:python_client' | chpasswd
RUN /usr/bin/ssh-keygen -A
RUN service ssh start

EXPOSE 22

WORKDIR /home/python_client
COPY python_client.py ./python_client.py

#CMD [ "python", "python_client.py" ]
CMD ["/usr/sbin/sshd", "-D"]
```



Build & run both

Server

- `$ docker build -t python_server .`
 - Builds the container and tags it as 'python_server'
- `$ docker run -d --name python_server --rm -p 9999:9999 python_server`
 - Run in detached mod
 - Set container name
 - Remove after use
 - Map ports to host ports

Client

- `$ docker build -t python_server .`
 - Note: the dot is for the local folder, where build will look for the dockerfile
- `$ docker run -d --name python_client --rm python_client`
- `$ docker attach -it python_client sh`
 - Run a single command in the container

NETWORKING



Docker networks

- Q: Can the Python client and server communicate out-of-the-box with their listed implementations?
- A: Not really, it is necessary to properly set up a network for them.
 - **Note:** the containers will be assigned to different networks by default
- Common Docker network types:
 - **bridge:** the network in which containers are run by default – note: limited to a single host running the Docker engine
 - **overlay:** the network type used for multi-host network communication
 - **macvlan:** the network type used to connect Docker containers directly to the host network interface(s)



Setting up a network

- `$ docker network ls`
 - List docker networks
- `$ docker network inspect`
 - Inspect networks
- `$ docker network create python_net`
 - Create a new Docker network with default type: bridge
- `$ docker run -d --name python_server --net python_net`
- `$ docker run -d --name python_client --net python_net`
 - Both will be accessible via symbolic names, i.e. `python_server` and `python_client` on the `python_net` network

DOCKER COMPOSE



Docker Compose

- **DEF:** Docker Compose is a tool for defining and running multi-container applications with Docker.
 - Comes pre-installed with Docker Desktop on Windows and Mac
 - Allows app developers to define multi-container applications in a single YAML configuration file
- Trivia:
 - Docker Compose was originally named Fig
 - Fig lead developer: Aanand Prasad
 - Fig was acquired by Docker in 2014 and renamed to Docker Compose



Compose configuration

```
version: "3.8"
services:
  python_server:
    build: ./python_server/
    networks:
      python_net:
        ipv4_address: 192.168.10.1
    ports:
      - "9999:9999"
  python_client:
    build: ./python_client/
    tty: true
    depends_on:
      - python_server
    networks:
      python_net:
        ipv4_address: 192.168.10.100
    ports:
      - 22:22
networks:
  python_net:
    ipam:
      driver: default
      config:
        - subnet: 192.168.10.0/24
```

- The YAML file is positioned outside the folders of the Docker containers
- Config file contents:
 - Version number (Docker Compose)
 - Services, i.e. containers
 - Networks → can be more than one



Compose commands

- Docker Compose command are (usually) issued via the command line (at least during the testing phase)
- `$ docker-compose --version`
- `$ docker-compose up`
 - Run a ‘composed’ app, i.e. cluster of containers & networks
 - Should be issued in the folder of the YAML file
- `$ docker-compose ps`
 - List containers
- `$ docker network ls`
 - List Docker networks → notice the network created by Compose
- `$ docker-compose down`
 - Destroy the cluster, i.e. containers & networks



Directories and files

python_app

 docker-compose.yml

python_client

 Dockerfile

 python_client.py

 requirements.txt

python_server

 Dockerfile

 python_server.py

 requirements.txt

DOCKER SECURITY



Docker security

1. Only use images from trusted vendors to avoid malware
2. Use thin, short-lived containers to reduce your attack surface
3. Avoid mixing containers to protect critical data
4. Use Docker Bench for Security to analyze your settings
5. Configure your containers properly to protect the host
6. Harden the host to safeguard all other containers in the environment
7. Employ secure computing mode (seccomp) to filter your system calls, e.g. filter out all audio-related api calls
8. Enable troubleshooting without logging in to limit SSH access

Summary

- Introduction
- Docker Hub
- Basic commands
- Images
- Networking
- Docker Compose
- Security



Thank you for your attention!

