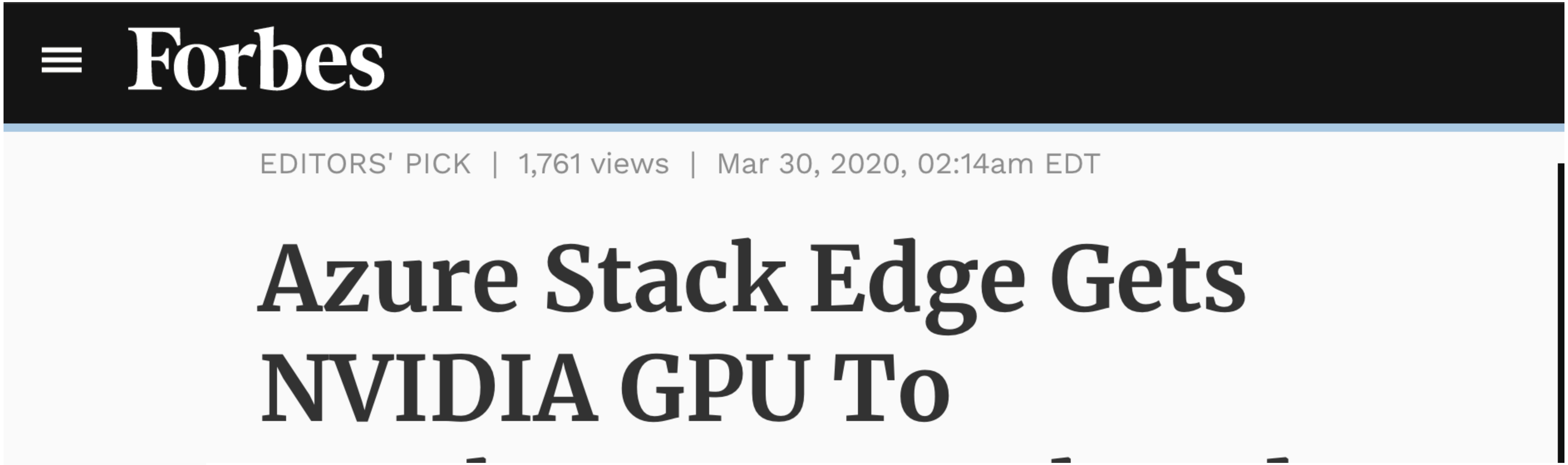
Life @ the Edge: Challenges Accelerating and Specializing

Theophilus A. Benson

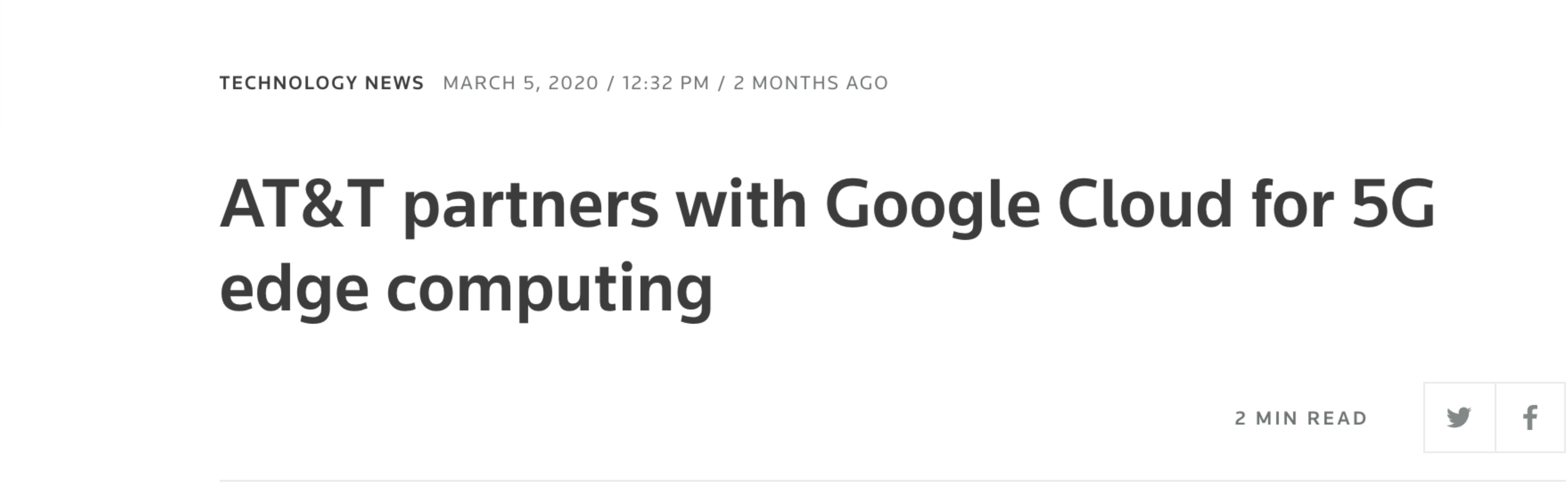
Brown University



Forbes

EDITORS' PICK | 1,761 views | Mar 30, 2020, 02:14am EDT

Azure Stack Edge Gets NVIDIA GPU To

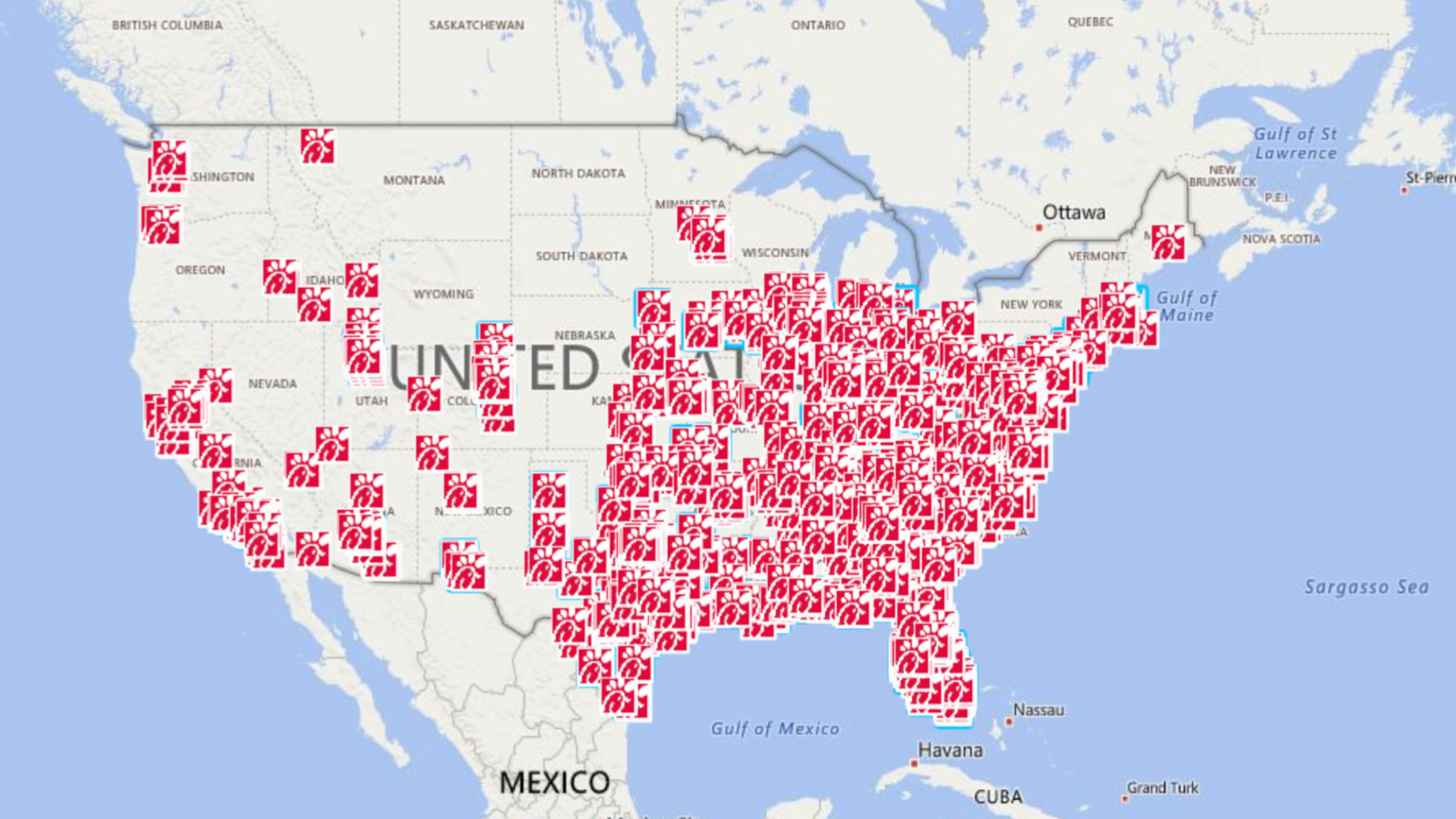


TECHNOLOGY NEWS MARCH 5, 2020 / 12:32 PM / 2 MONTHS AGO

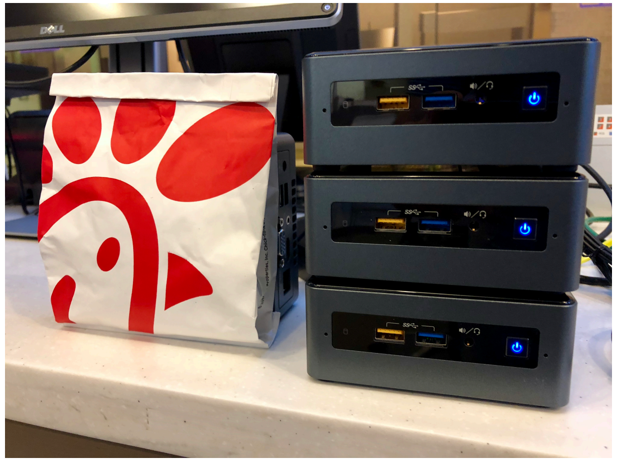
AT&T partners with Google Cloud for 5G edge computing

2 MIN READ

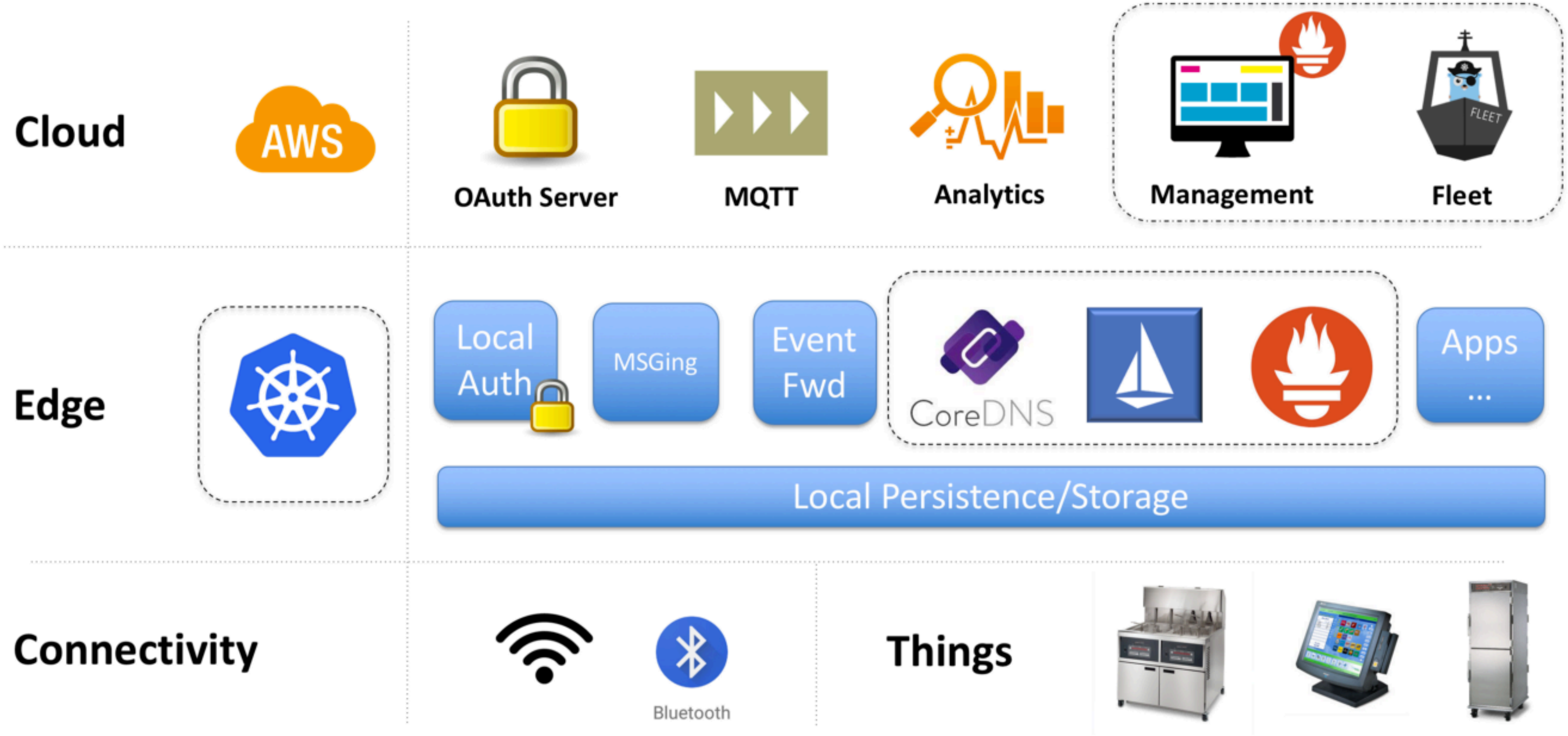
2000 edge deployments in the US?


BRITISH COLUMBIA
SASKATCHEWAN
ONTARIO
QUEBEC
Gulf of St Lawrence
NEW BRUNSWICK
P.E.I.
St-Pierre
NOVA SCOTIA
Ottawa
VERMONT
NEW YORK
Gulf of Maine
WASHINGTON
MONTANA
NORTH DAKOTA
MINNESOTA
SOUTH DAKOTA
WISCONSIN
OREGON
IDAHO
WYOMING
NEBRASKA
UNITED STATES
NEVADA
UTAH
COLORADO
KANSAS
CALIFORNIA
NEW MEXICO
Sargasso Sea
Gulf of Mexico
Nassau
Havana
MEXICO
CUBA
Grand Turk

Restaurant “Data Centers”



Intel: Quadcore processor, 8 GB RAM, SSD



Cloud

AWS

OAuth Server

MQTT

Analytics

Management

Fleet

Edge

Local Auth

MSGing

Event Fwd

CoreDNS

Apps ...

Local Persistence/Storage

Connectivity

Bluetooth

Things

Holistic Management of
Edge Accelerator for
maximum efficiency

Dynamic Specialization of
Edge Network Stack for Varying Objectives

Applications of Edge Computing

AI

5G

IoT

Platforms for Edge Computing

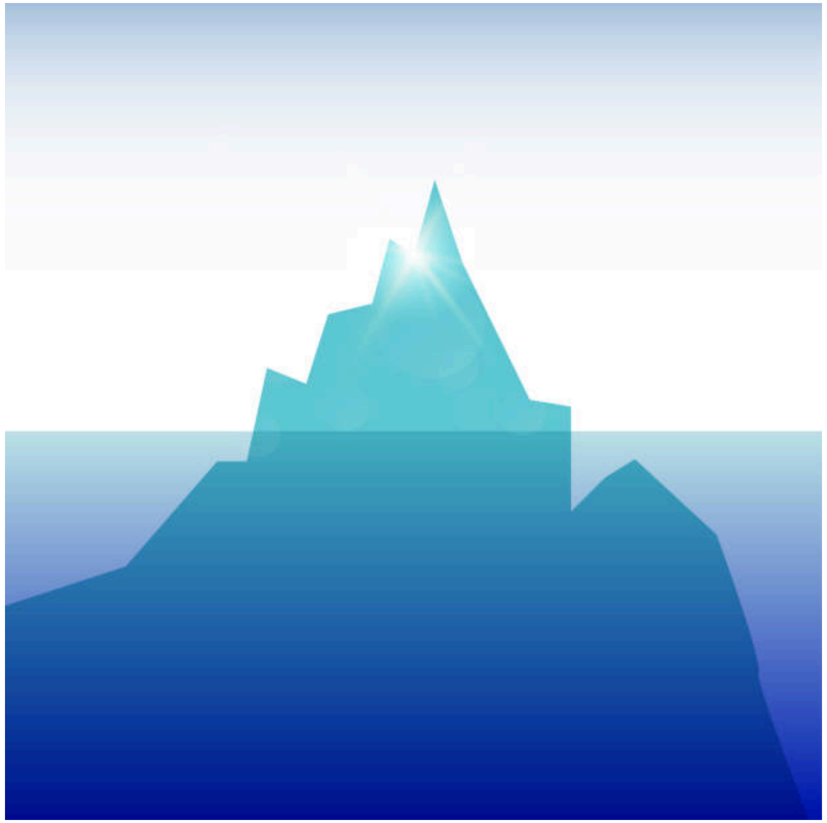
Networking Stack

Operating System

CPU

GPU

FPGA

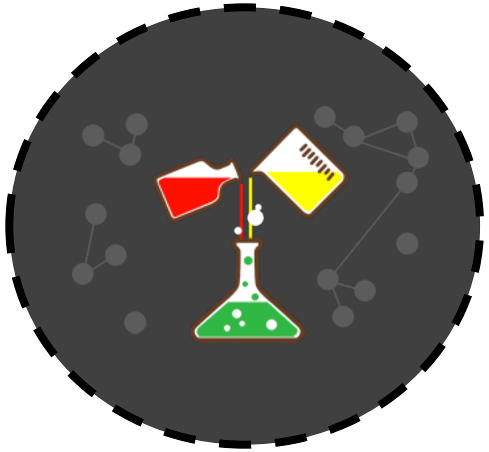


https://www.istockphoto.com/illustrations/tip-of-the

RoadMap

Motivations

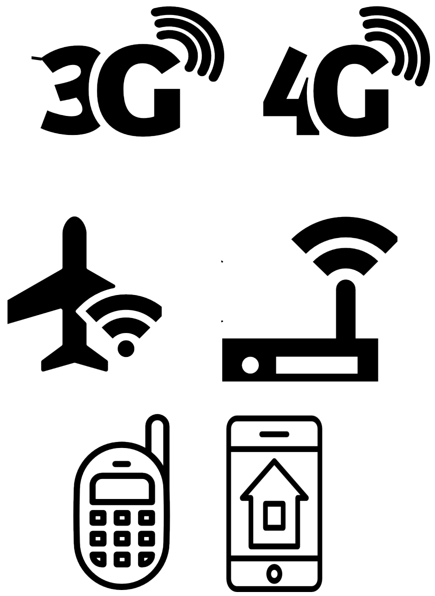
Data-Driven Specialization



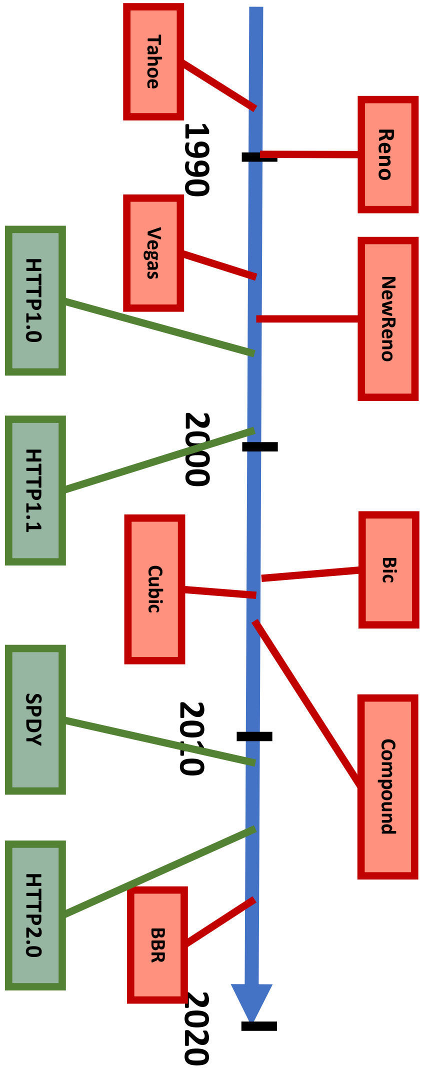
Accelerator Management

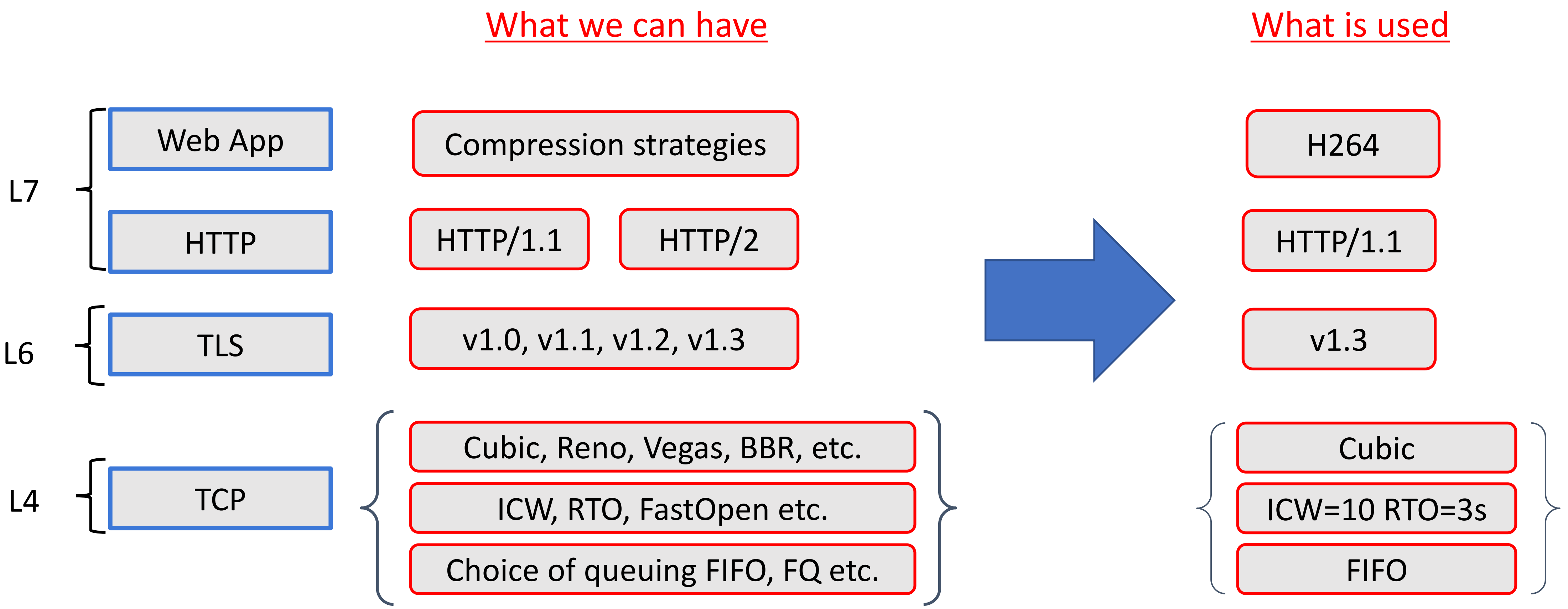
- Networks evolve organically
  - Adaptation of new technology
- Protocols are redesigned
  - Encapsulate domain-specific insights
- Protocols are statically deployed
  - Protocol use is agnostic of conditions

Infrastructure Heterogeneity



Protocol/Config Heterogeneity

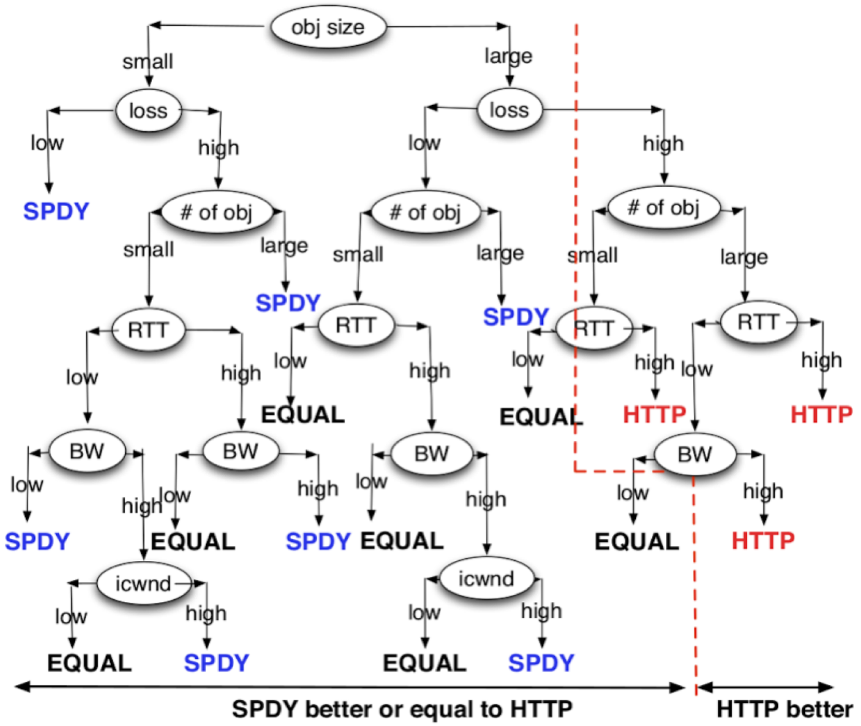




		What we can have		What is used
L7	Web App	Compression strategies		H264
	HTTP	HTTP/1.1	HTTP/2	HTTP/1.1
L6	TLS	v1.0, v1.1, v1.2, v1.3		v1.3
L4	TCP	Cubic, Reno, Vegas, BBR, etc.		Cubic
		ICW, RTO, FastOpen etc.		ICW=10 RTO=3s
		Choice of queuing FIFO, FQ etc.		FIFO

one size fits all to address diverse networks and end user devices!!!

Web Performance

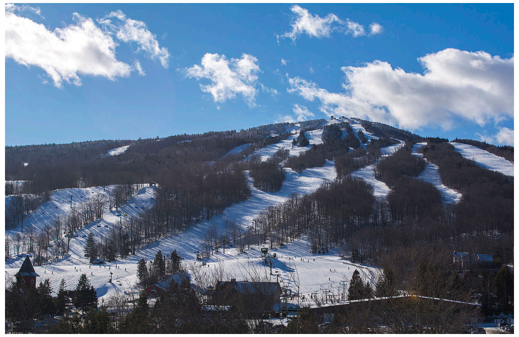


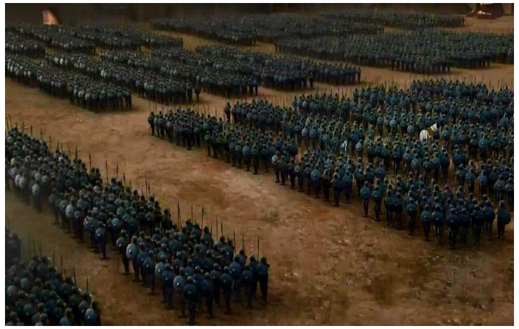
How speedy is SPDY? [NSDI ’14]

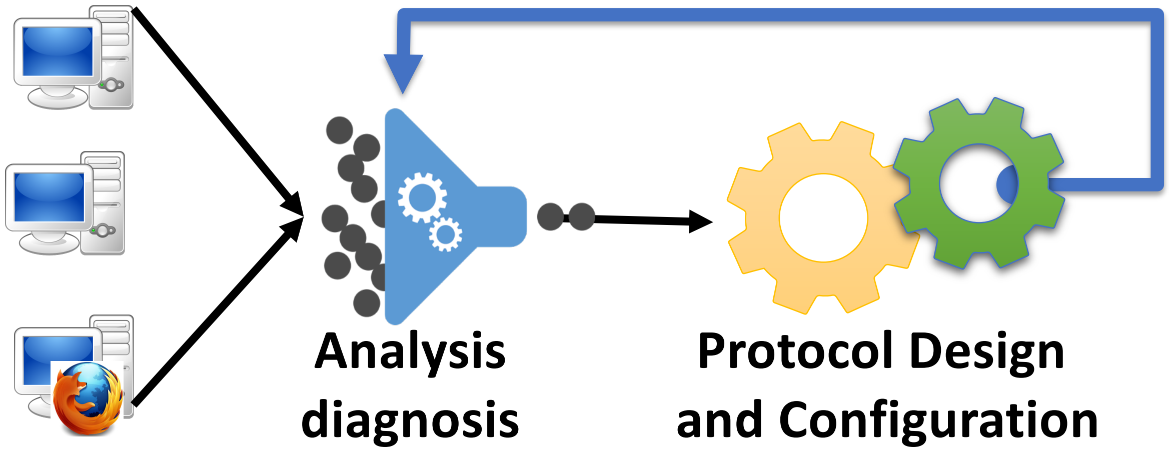
- How Speedy is SDPY? [NSDI ’15]
  -HTTP versions
- Verus [NSDI ’15]/Pantheon [USENIX ATC’18]
  -TCP versions
- How quick is QUIC? [IEEE ICC ’16]
  -QUIC vs HTTP2 vs HTTP1.1
- Overclocking the Yahoo! [IMC ’11]
  -HTTP pipelining, TCP ICW, ABC

How do you pick which car?

Garage of Cars







Infrastructure consolidation:

- Providers own data center, edge, client

Well defined web metrics

- Web pages: SpeedIndex
- Video: Netflix metric

Open Challenges:

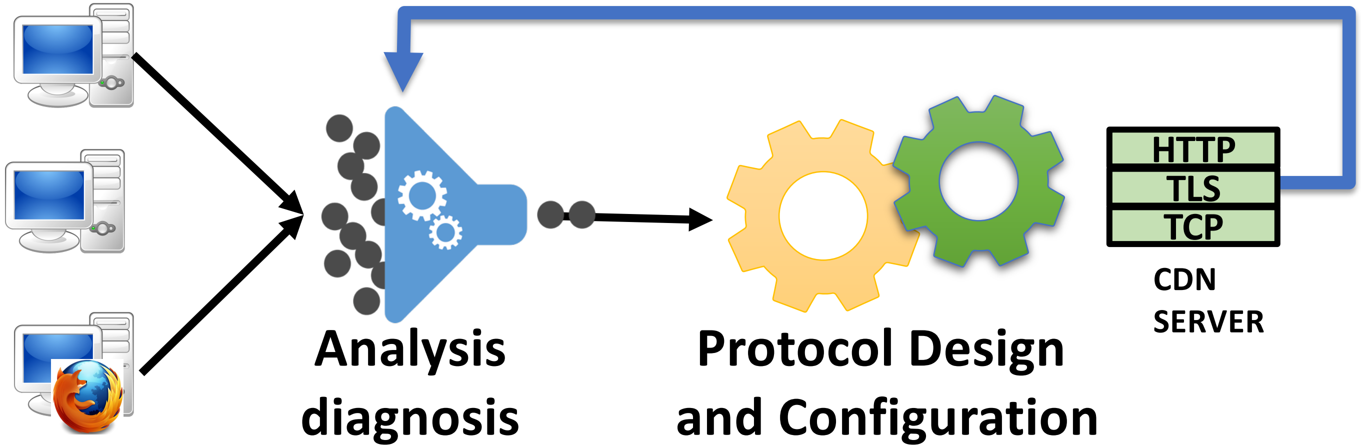
- Domain specific learning algorithm
- Fine-grained, low-overhead knobs

CDN
Application

HTTP
TLS
TCP

CDN
SERVER

Barriers to Fine-grained and Dynamic Tuning



- Algorithm learning challenges
  - Need effective and accurate predictive algorithms

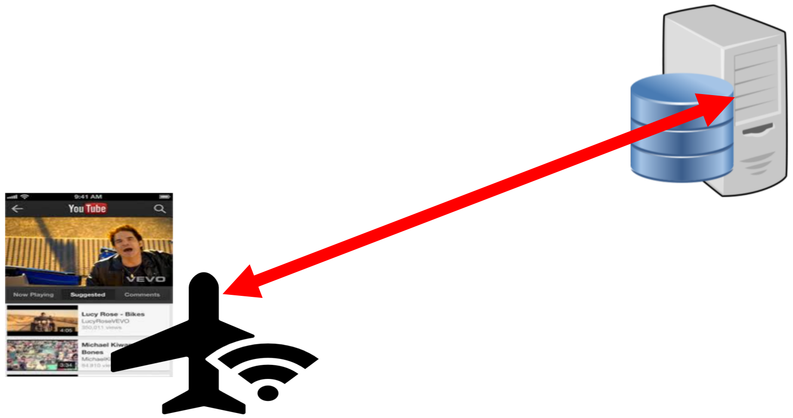
- System design challenges
  - Server must support low overhead fine grained tuning

Strawman Approach

- For each client
  - Greedily test each configuration
  - Pick optimal configuration

Challenges

- **Network is dynamic**
- **Data collection is costly**
- **Data can be noisy (non-gaussian noise)**



« Handle 700 Percent More Requests Using Squid and APC Cache | Main | Paper: Parallelizing the Web Browser »

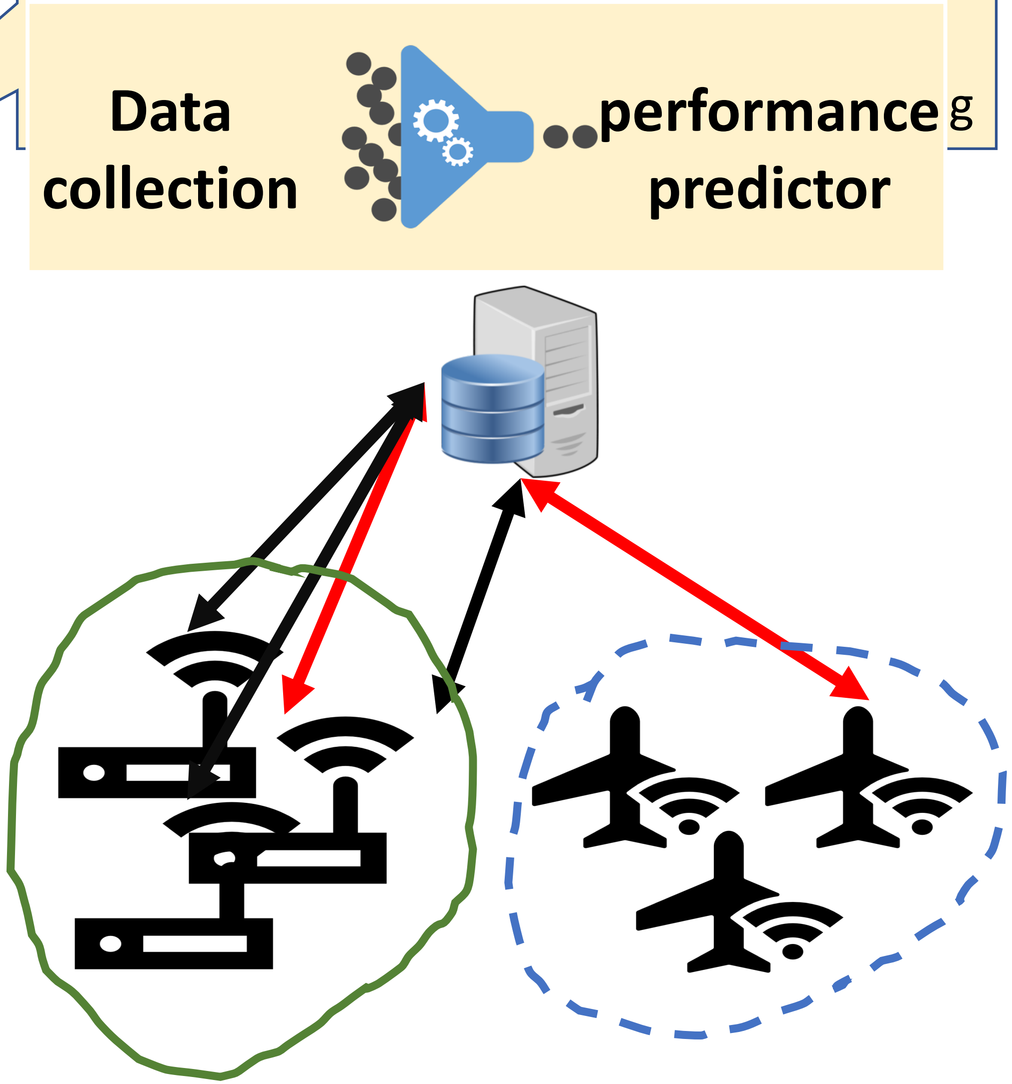
Latency Is Everywhere And It Costs You Sales - How To Crush It

Challenges

- Costly data collection
  - Distribute cost over similar users
  - Generalize observations
- Network dynamics
  - Online exploration of network
- Noisy data (non-gaussian noise)
  - Resample to get clean data

<u>Predictors</u>

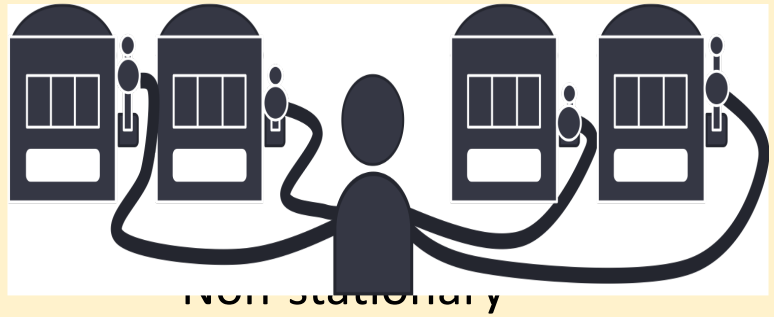
- Machine learning
  - Deep learning (CNN, RNN)

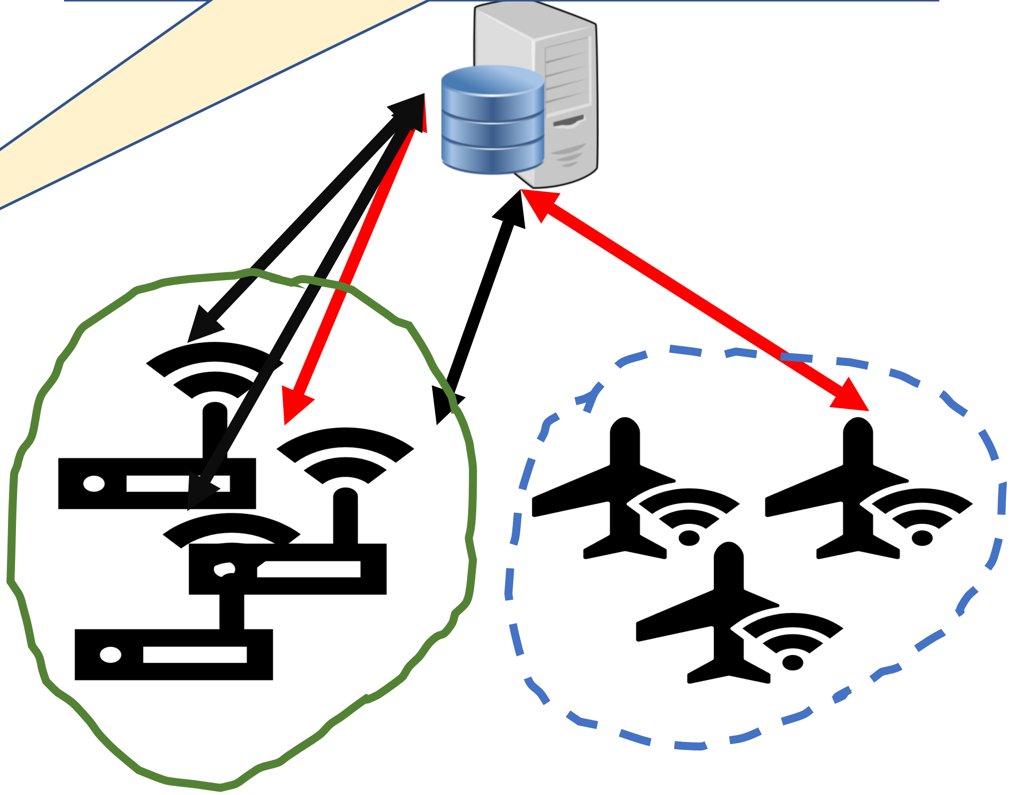


Challenges

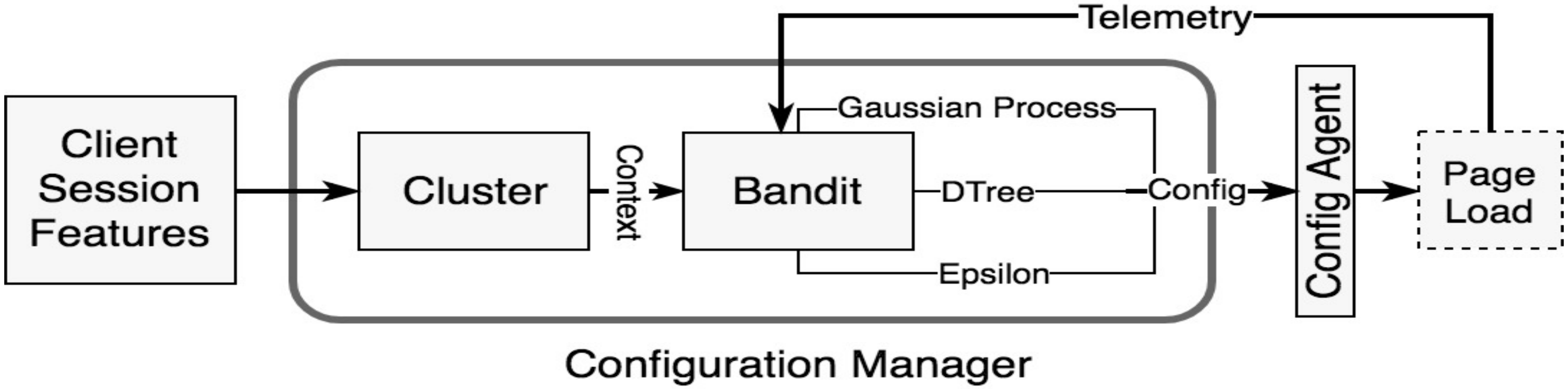
- Costly data collection
  - Distribute cost over similar users
  - Generalize observations
- Network dynamics
  - Online exploration of network
- Noisy data (non-gaussian noise)
  - Resample to get clean data

Exploit-Explore (MultiArmed Bandit)





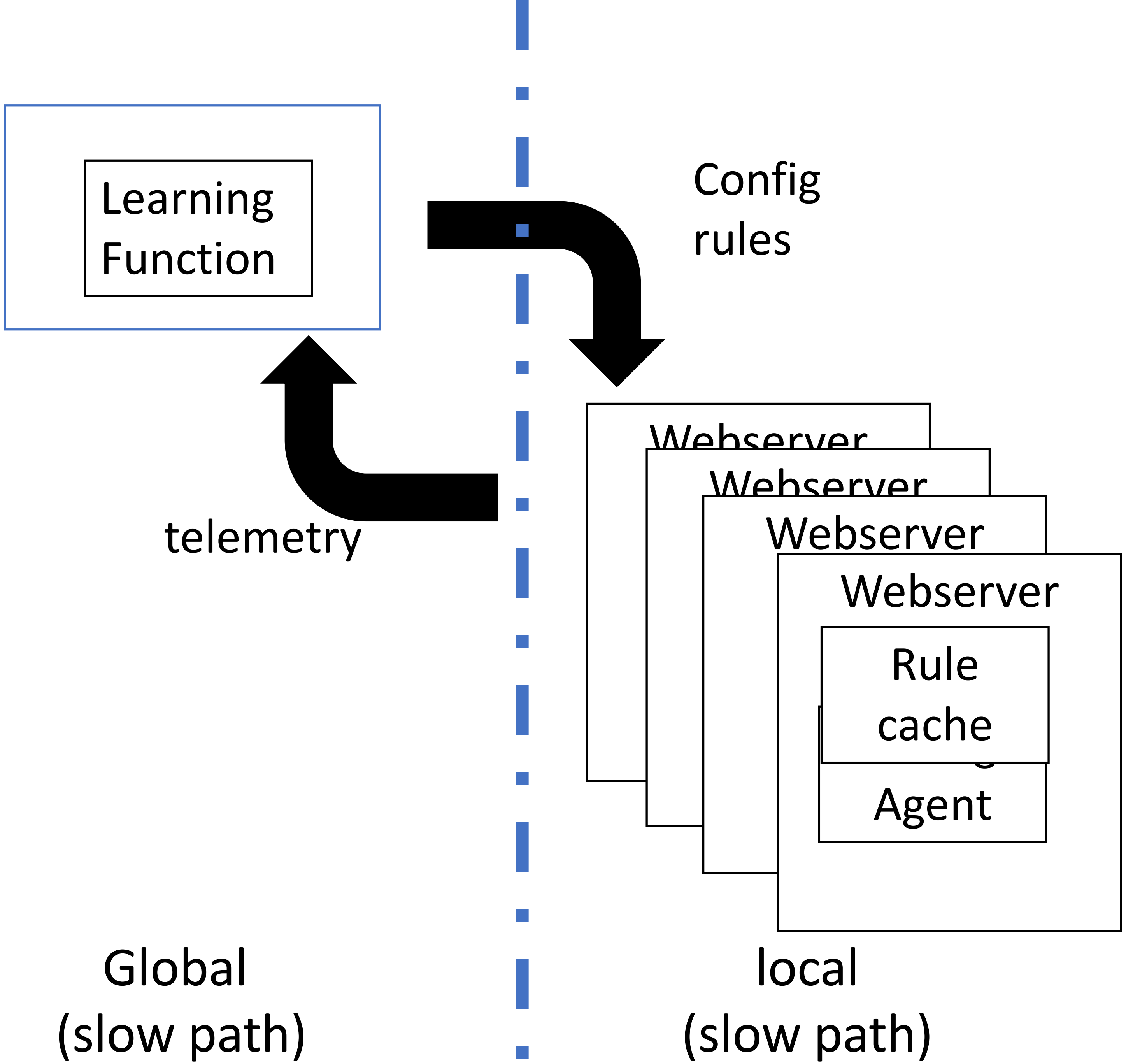
Domain-specific learning ensemble



- Contextual Multi-armed bandit: explore-exploit!
    - Contextual-exploration: cluster provides context
        - Non-Guassian: random exploration
        - Efficient learning: Gaussian exploration
    - Exploitation Arm:
        - D-Tree: Generalize observation

Systems Support for Reconfiguration

	Flexibility	Overheads
One VM per Configuration	👍	👎
One Container per Configuration	👎	👍
User Space Tuning + Kernel Module	👍	👎
Kernel Space Tuner+ kernel Module	👍	👍



Learning Function

Config rules

telemetry

Webserver

Webserver

Webserver

Webserver

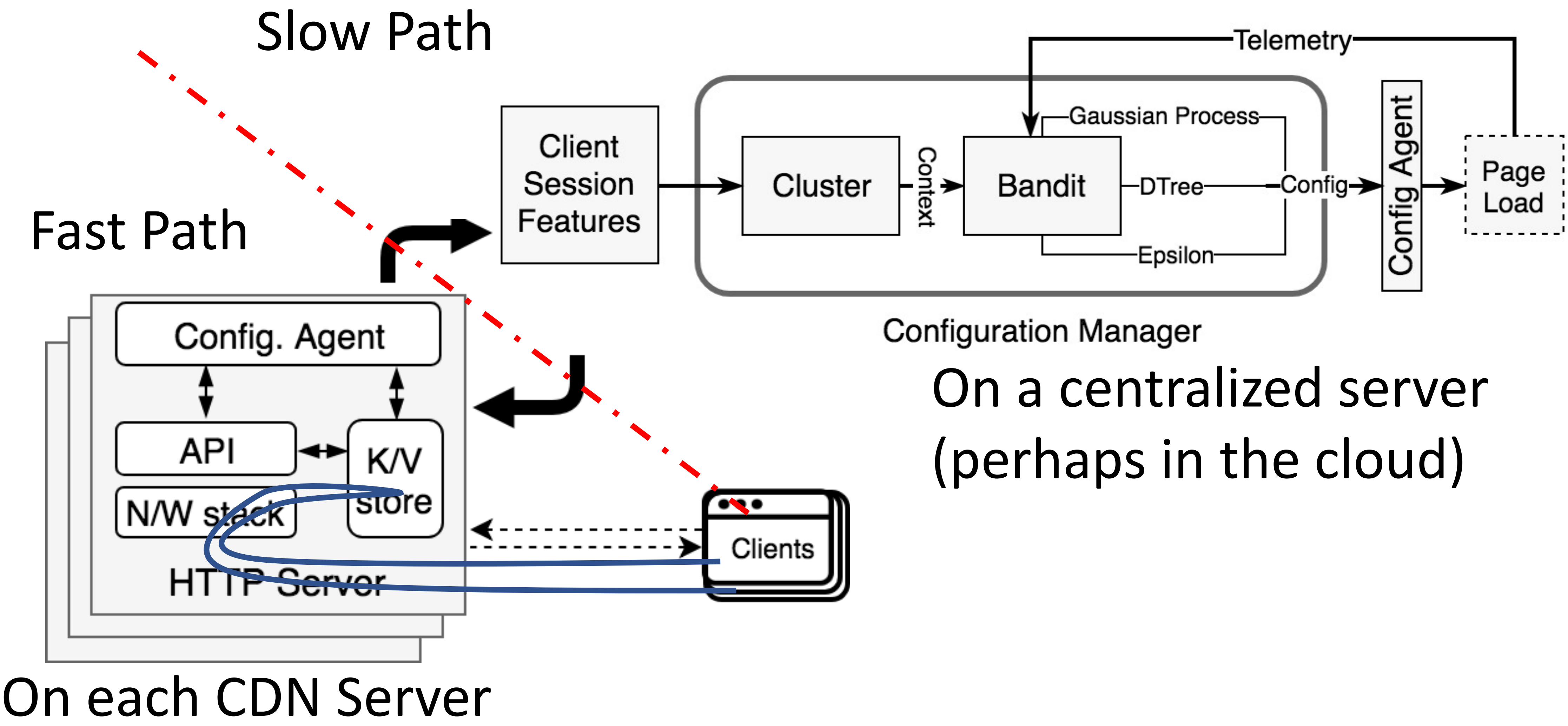
Rule cache

Agent

Global (slow path)

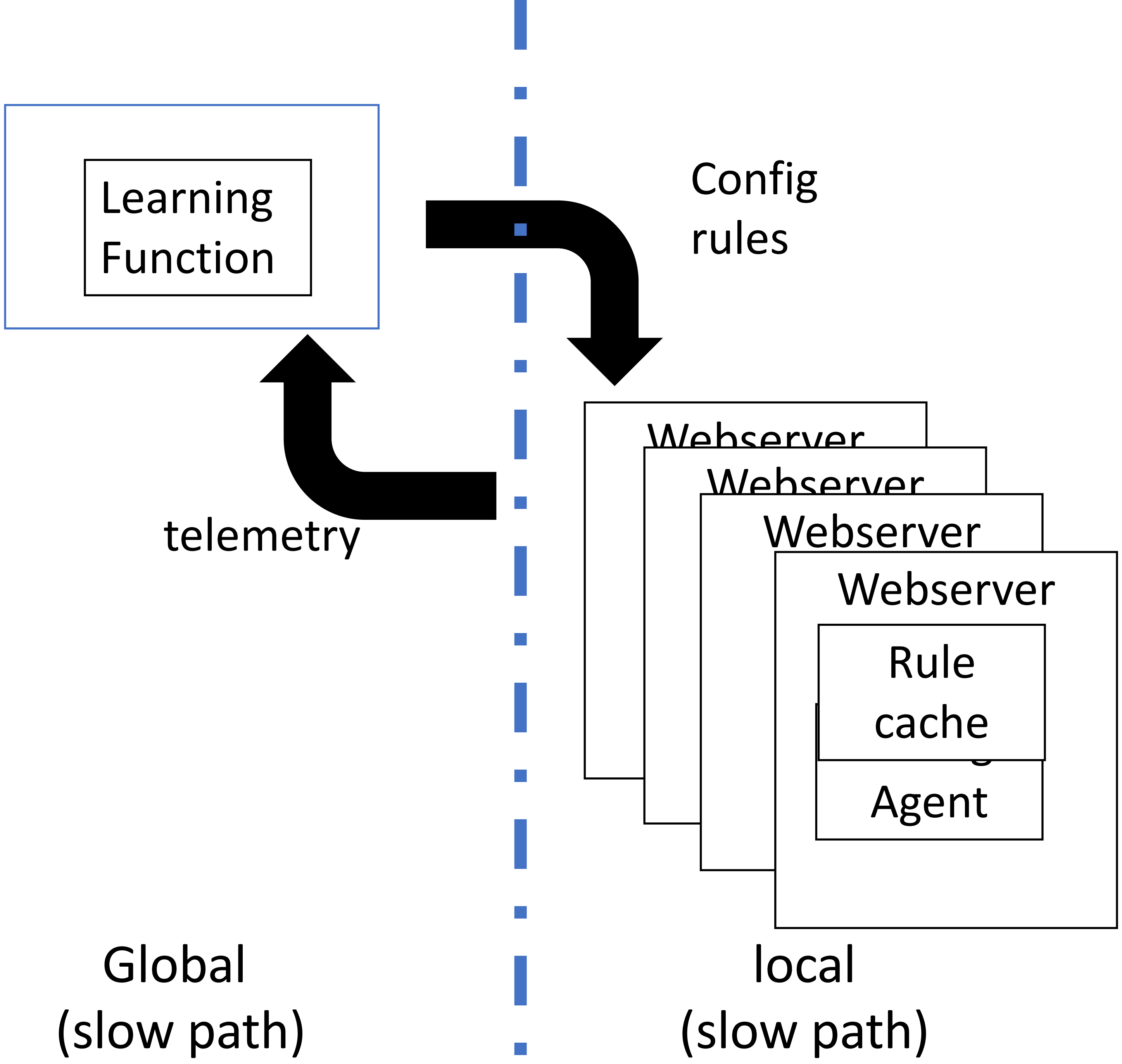
local (slow path)

ConfigTron



On a centralized server (perhaps in the cloud)

On each CDN Server



Learning Function

Config rules

telemetry

Webserver

Webserver

Webserver

Webserver

Rule cache

Agent

Global (slow path)

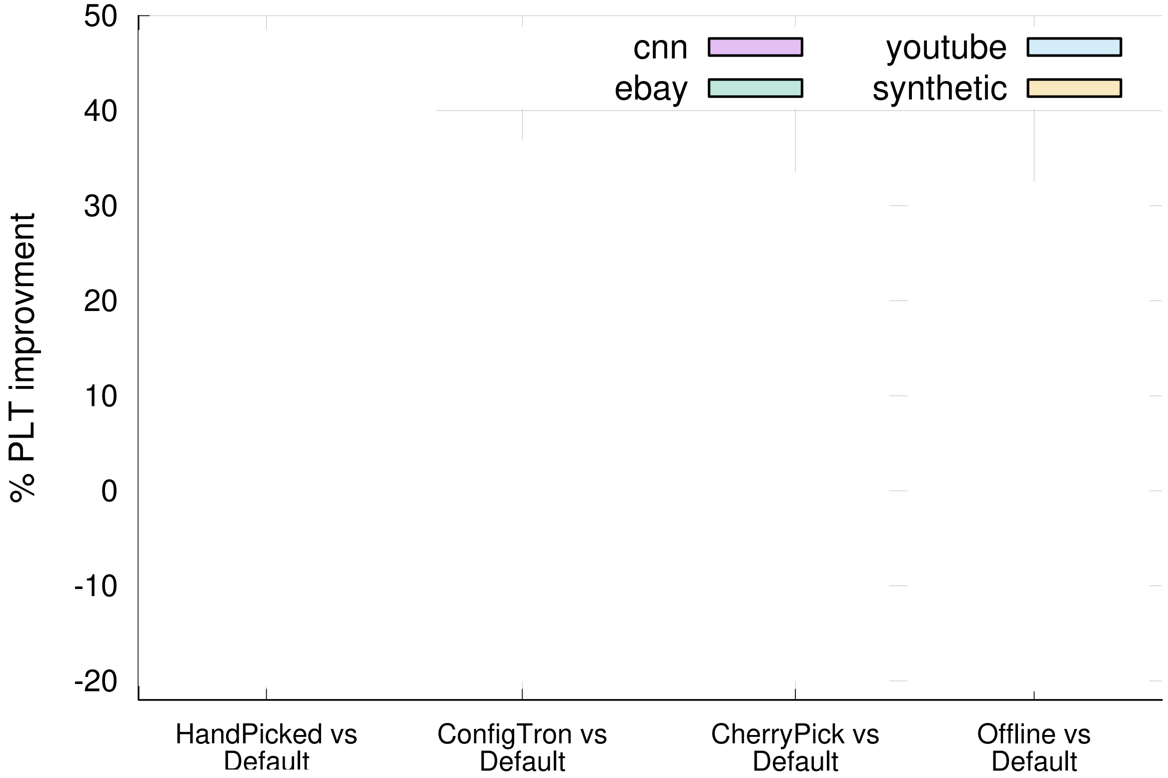
local (slow path)

Systems Approach to Configuration Tuning

- Manual protocol tuning
  - Coarse-grained and slow
    - Labor intensive
    - Slow to adapt to changes

- Bayesian-based tuning
  - DB: iTuned [VLDB'9], StarFish [CIDR'11]
  - Apps: WISP [SoCC'17], Ernest [NSDI'16]
  - Cloud: CherryPick [NSDI'17]
  - Fine-grained but static
    - Quickly finds "good" configuration
    - Doesn't adapt to network dynamics
    - Doesn't account for non-gaussian noise

- ConfigTron is fine-grained and dynamic



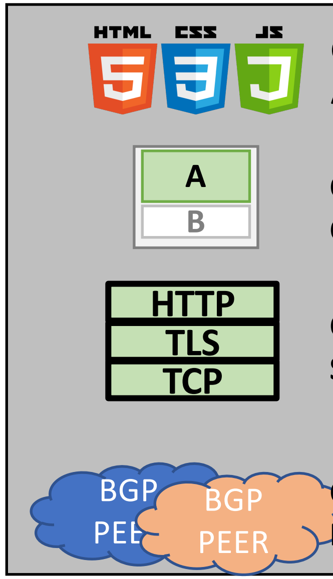
Objective Functions

ClientDevice Specific

Provider Specific

Application Specific

Configuration Knobs

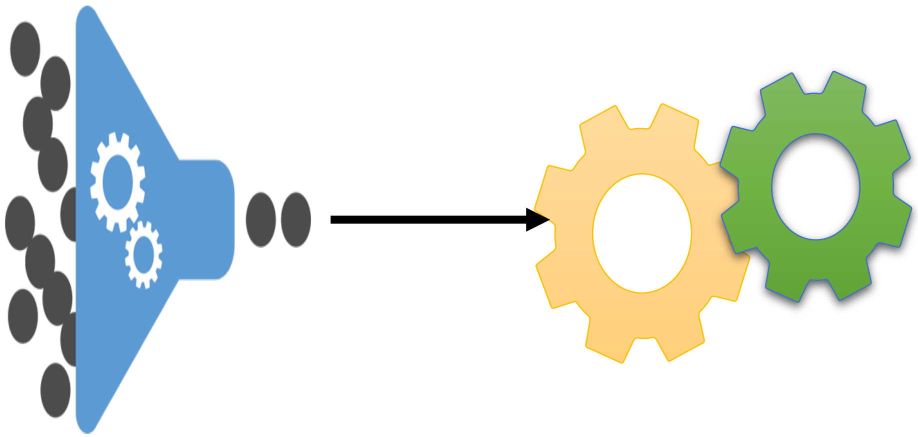


CDN Application

CDN Caching

CDN SERVER

CDN Network



Stability

Fairness

Security

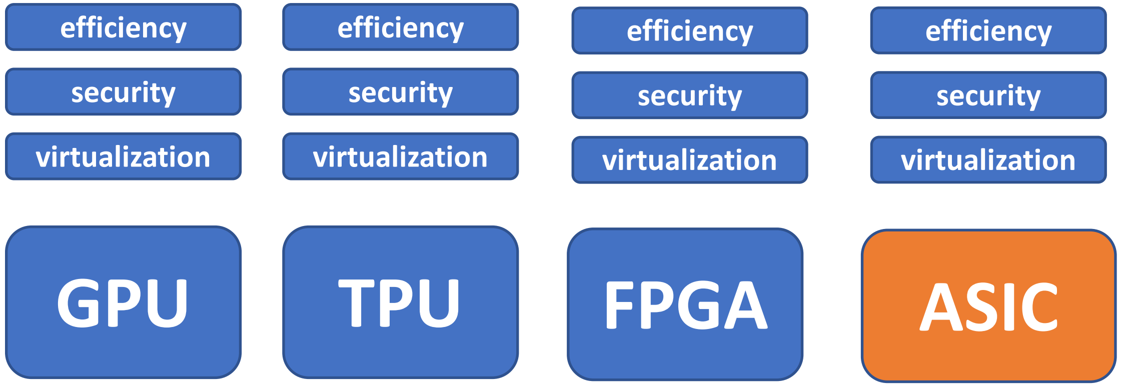
http://tradingeveryday.com/wp-content/uploads/2016/08/fastdotcom.png

RoadMap

Motivations

Data-Driven Specialization

Accelerator Management



efficiency

security

virtualization

GPU

efficiency

security

virtualization

TPU

efficiency

security

virtualization

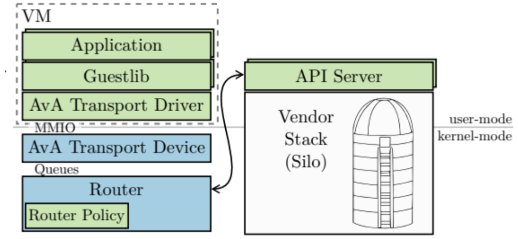
FPGA

efficiency

security

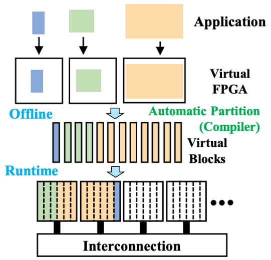
virtualization

ASIC



GPU Virtualization

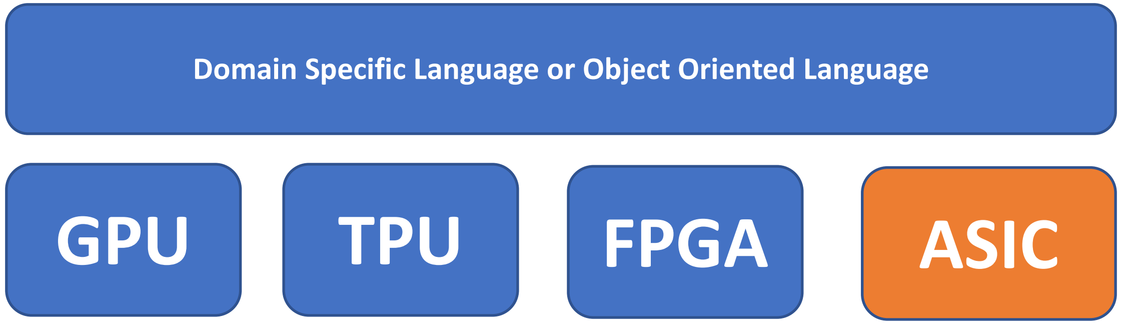
- CAVA[ASPLOS'20]
- Time-division multiplexing



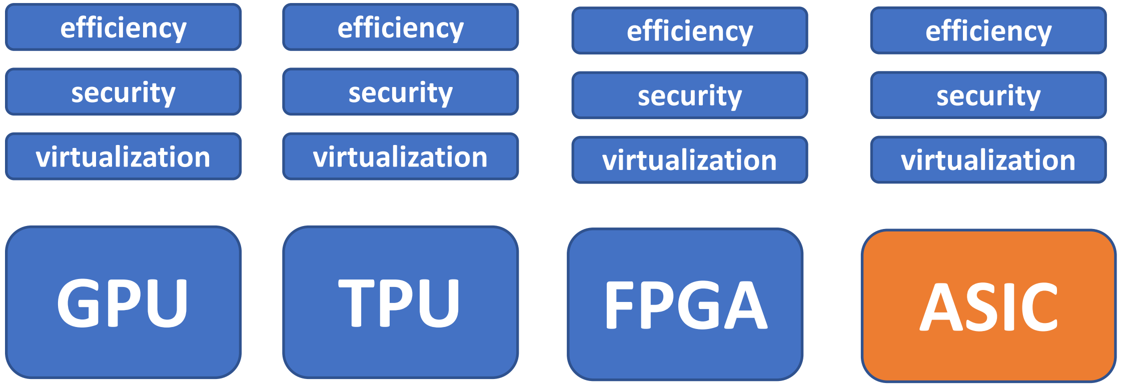
FPGA Virtualization

- Virtal [ASPLOS'20]
- Space-division multiplexing

- Rich body of work on hardware accelerators
  - Class one: Siloed approach → limits global orchestration and management



- Rich body of work on hardware accelerators
  - Class one: Siloed approach → limits global orchestration and management
  - Class two: Unified approach → disconnected from lower layer optimizations/advances

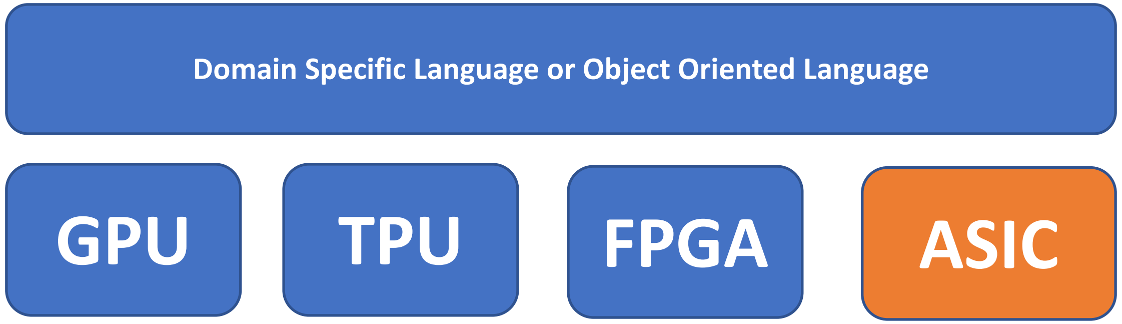


efficiency | efficiency | efficiency | efficiency

security | security | security | security

virtualization | virtualization | virtualization | virtualization

GPU | TPU | FPGA | ASIC



Domain Specific Language or Object Oriented Language

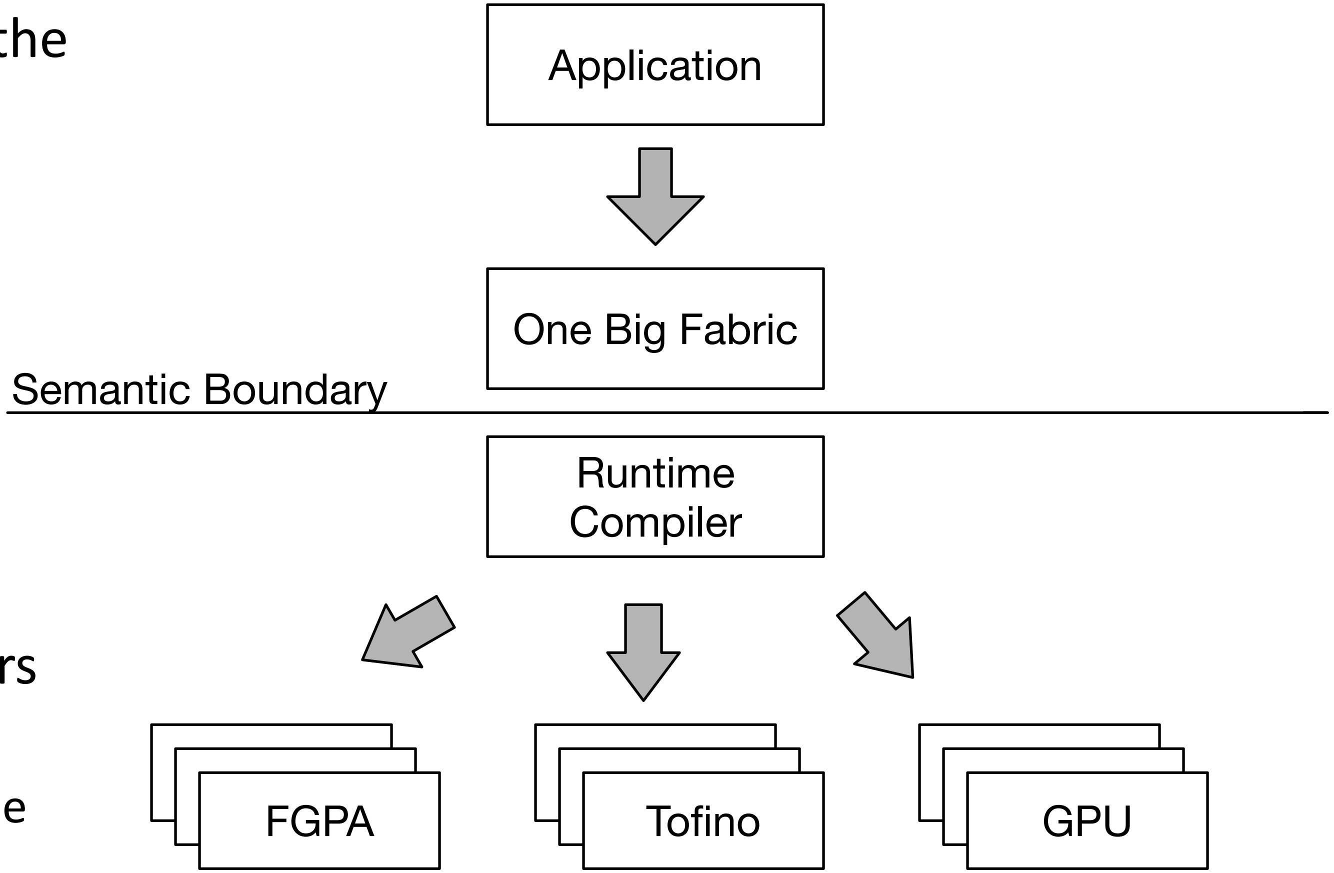
GPU | TPU | FPGA | ASIC

- Rich body of work on hardware accelerators
  - Class one: Siloed approach → limits global orchestration and management
  - Class two: Unified approach → disconnected from lower layer optimizations/advances

- Neither allows efficient management of a fabric of accelerators

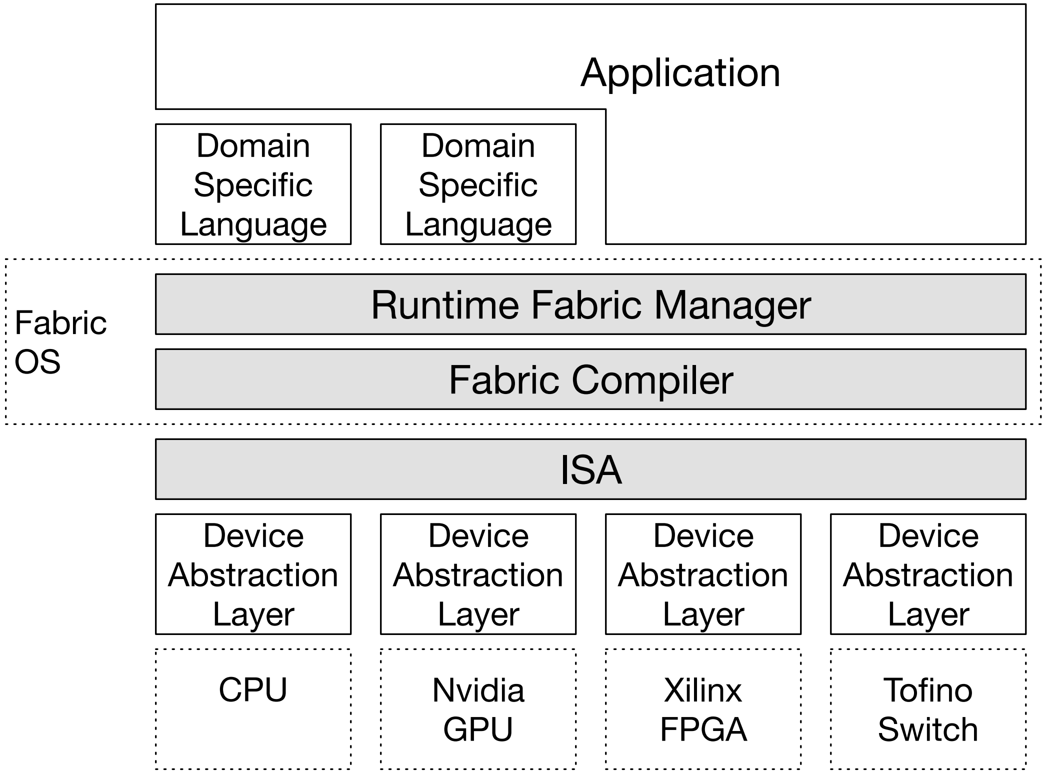
Ideal Cloud Accelerators Fabric

- ``Accelerators’’ and not developers are the first-class citizens
  - Simplify life-cycle management
- Provide ``big fabric’’ abstraction
  - Abstract device level details
- Ensure efficient utilization of accelerators
  - Capture data and execution pipelining
  - Tackle various level of parallelism available

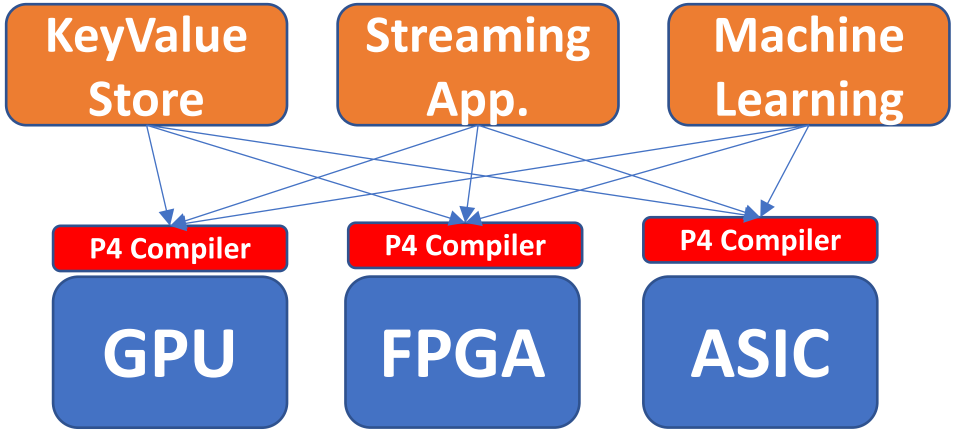


Unifying Instruction Set Architecture

- Unify at the instruction set level
  - Decouple fabric properties. From unification
- Enable innovation above and below ISA
  - Above (OS): global properties, e.g., security and efficiency
  - Below (Device drivers): local properties, e.g., security and efficiency



Preliminary Evidence of ISA for Accelerators



- Rich body of work on P4
  - Porting application to use P4
  - Developing P4 compilers for Accelerators

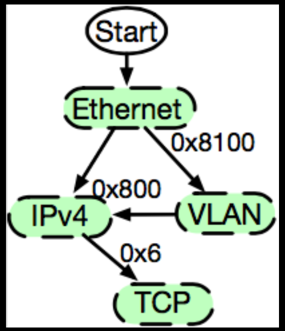
Primer on Programmable Data Planes

P4
Code

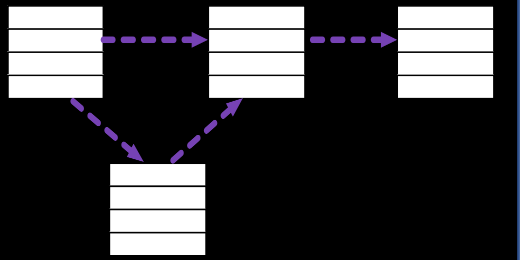
- Composed of multiple components:
  - Parser (FSM)
  - Control flow graph (weighted DAG)
  - Match/Action Tables

Language Limitations

- **No loops, pointers**



```
table ipv4_lpm {
   reads {
      ipv4.dstAddr : lpm;
   }
  actions {
      set_next_hop; drop;
  }
}
```



Primer on Programmable Data Planes

Language Limitations

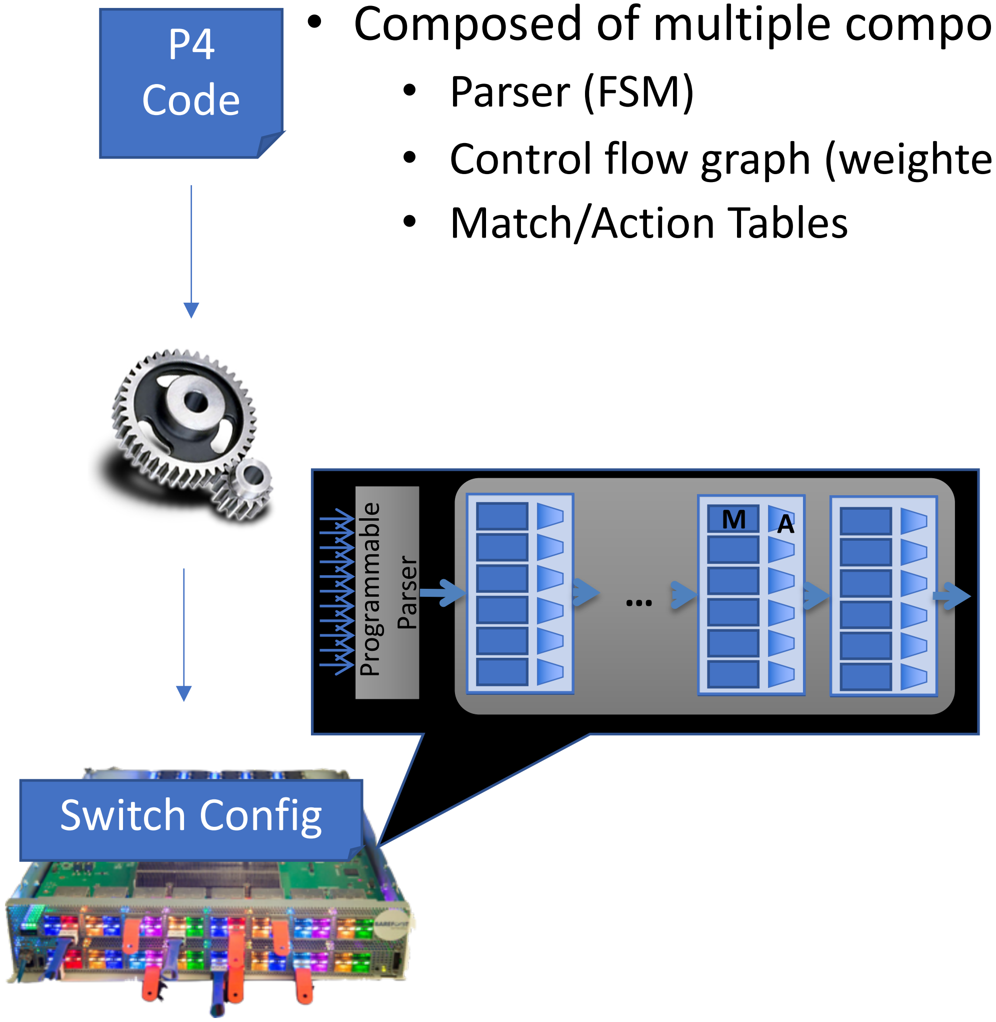
- **No loops, pointers**

Easy to Analyze

Hardware Limitations

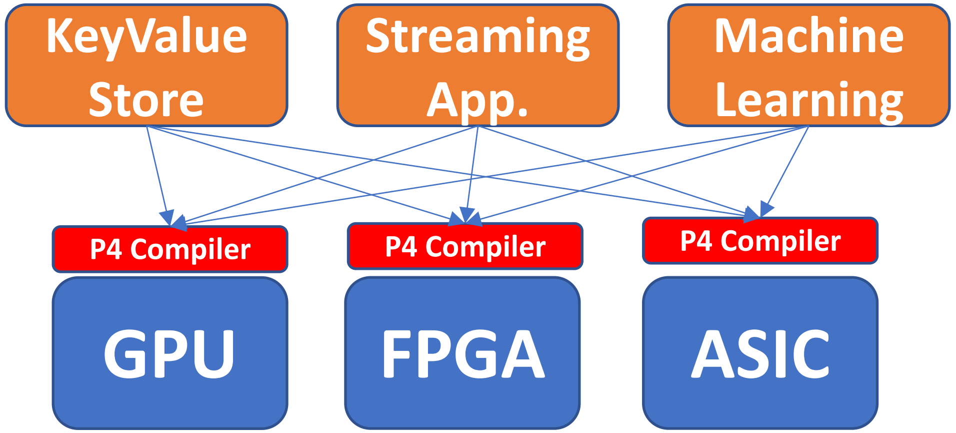
- **Limited resources**
- **Limited processing**

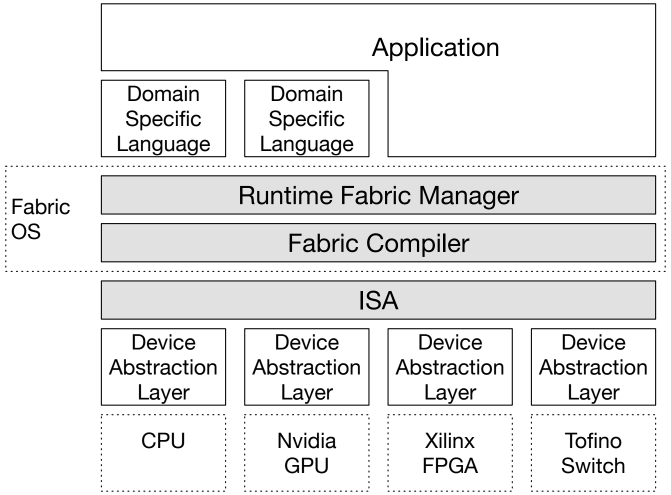
Predictable Performance



- Composed of multiple components:
  - Parser (FSM)
  - Control flow graph (weighted DAG)
  - Match/Action Tables

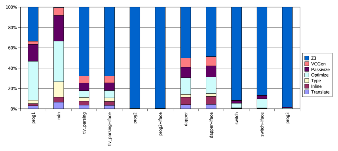
Preliminary Evidence of ISA for Accelerators

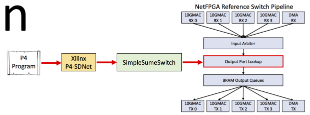




- Rich body of work on P4
  - Porting application to use P4
  - Developing P4 compilers for Accelerators

- Initial benefits
  - Practical verification → limited language
  - Demonstrated acceleration → manual partitioning
  - Simply development → More intuitive than Verilog

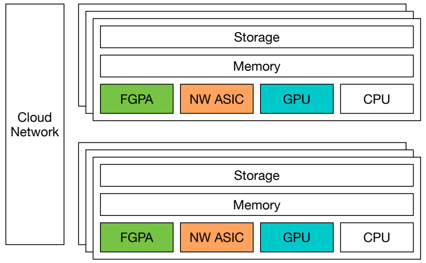




- Many open challenges
  - Security: device level and fabric level
  - Global management
  - Virtualization
  - Efficient utilization of hardware

The Road towards AcceleratorFabric

- Classic but Open Management Challenges
  - Security: device level and fabric level
  - Global resource management
  - Device virtualization
  - Efficient utilization of hardware
  - Program diagnosis and debugging

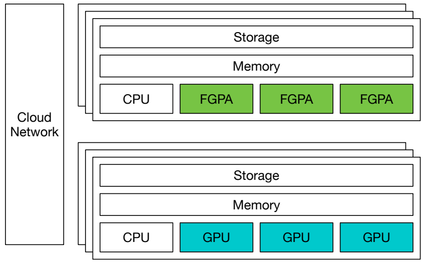


Cloud Network

Storage
Memory
FGPA | NW ASIC | GPU | CPU

Storage
Memory
FGPA | NW ASIC | GPU | CPU

Consolidated Accelerator Deployments

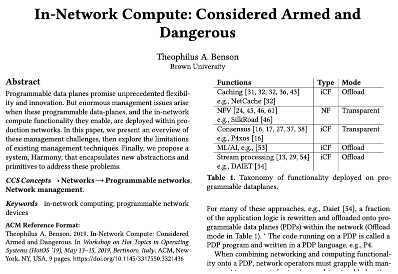


Cloud Network

Storage
Memory
CPU | FGPA | FGPA | FGPA

Storage
Memory
CPU | GPU | GPU | GPU

Disaggregated Accelerator Deployments



In-Network Compute: Considered Armed and Dangerous

Theophilus A. Benson
Brown University

Abstract

Programmable data planes promise unprecedented flexibility and innovation. But enormous management issues arise when these programmable data-planes, and the in-network compute functionality they enable, are deployed within production networks. In this paper, we present an overview of these management challenges, then explore the limitations of existing management techniques. Finally, we propose a system, Harmony, that encapsulates new abstractions and primitives to address these problems.

CCS Concepts • **Networks → Programmable networks**; **Network management**.

Keywords in-network computing; programmable network devices

ACM Reference Format:
Theophilus A. Benson. 2019. In-Network Compute: Considered Armed and Dangerous. In *Workshop on Hot Topics in Operating Systems (HotOS '19), May 13–15, 2019, Bertinoro, Italy*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3317550.3321436

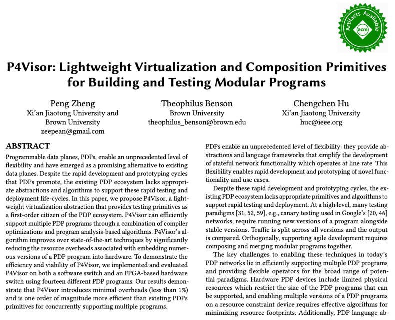
Functions	Type	Mode
Caching [31, 32, 32, 36, 43] e.g., NetCache [32]	iCF	Offload
NFV [24, 45, 46, 61] e.g., SilkRoad [46]	NF	Transparent
Consensus [16, 17, 27, 37, 38] e.g., P4xos [16]	iCF	Transparent
ML/AI, e.g., [53]	iCF	Offload
Stream processing [13, 29, 54] e.g., DAIET [54]	iCF	Offload

Table 1. Taxonomy of functionality deployed on programmable dataplanes.

For many of these approaches, e.g., Daiet [54], a fraction of the application logic is rewritten and offloaded onto programmable data planes (PDPs) within the network (Offload mode in Table 1). * The code running on a PDP is called a PDP program and written in a PDP language, e.g., P4.

When combining networking and computing functionality onto a PDP, network operators must grapple with man-

Harmony [HotOS'19]



P4Visor: Lightweight Virtualization and Composition Primitives for Building and Testing Modular Programs

Peng Zheng
Xi'an Jiaotong University and
Brown University
zeepean@gmail.com

Theophilus Benson
Brown University
theophilus_benson@brown.edu

Chengchen Hu
Xi'an Jiaotong University
huc@ieee.org

ABSTRACT

Programmable data planes, PDPs, enable an unprecedented level of flexibility and have emerged as a promising alternative to existing data planes. Despite the rapid development and prototyping cycles that PDPs promote, the existing PDP ecosystem lacks appropriate abstractions and algorithms to support these rapid testing and deployment life-cycles. In this paper, we propose P4Visor, a lightweight virtualization abstraction that provides testing primitives as a first-order citizen of the PDP ecosystem. P4Visor can efficiently support multiple PDP programs through a combination of compiler optimizations and program analysis-based algorithms. P4Visor's algorithm improves over state-of-the-art techniques by significantly reducing the resource overheads associated with embedding numerous versions of a PDP program into hardware. To demonstrate the efficiency and viability of P4Visor, we implemented and evaluated P4Visor on both a software switch and an FPGA-based hardware switch using fourteen different PDP programs. Our results demonstrate that P4Visor introduces minimal overheads (less than 1%) and is one order of magnitude more efficient than existing PDPs primitives for concurrently supporting multiple programs.

PDPs enable an unprecedented level of flexibility: they provide abstractions and language frameworks that simplify the development of stateful network functionality which operates at line rate. This flexibility enables rapid development and prototyping of novel functionality and use cases.

Despite these rapid development and prototyping cycles, the existing PDP ecosystem lacks appropriate primitives and algorithms to support rapid testing and deployment. At a high level, many testing paradigms [31, 52, 59], e.g., canary testing used in Google's [20, 46] networks, require running new versions of a program alongside stable versions. Traffic is split across all versions and the output is compared. Orthogonally, supporting agile development requires composing and merging modular programs together.

The key challenges to enabling these techniques in today's PDP networks lie in efficiently supporting multiple PDP programs and providing flexible operators for the broad range of potential paradigms. Hardware PDP devices include limited physical resources which restrict the size of the PDP programs that can be supported, and enabling multiple versions of a PDP programs on a resource constraint device requires effective algorithms for minimizing resource footprints. Additionally, PDP language ab-

P4Visor [CoNEXT'18]

Life @ the Edge

- The edge is real!! (and it gets you burgers)
- Heterogeneity dominates the edge
  - Let’s lean into this heterogeneity
  - New systems techniques and principles
- Think holistically about management
  - Performance, efficiency AND correctness, security

Questions?