From Programmability to Fungibility

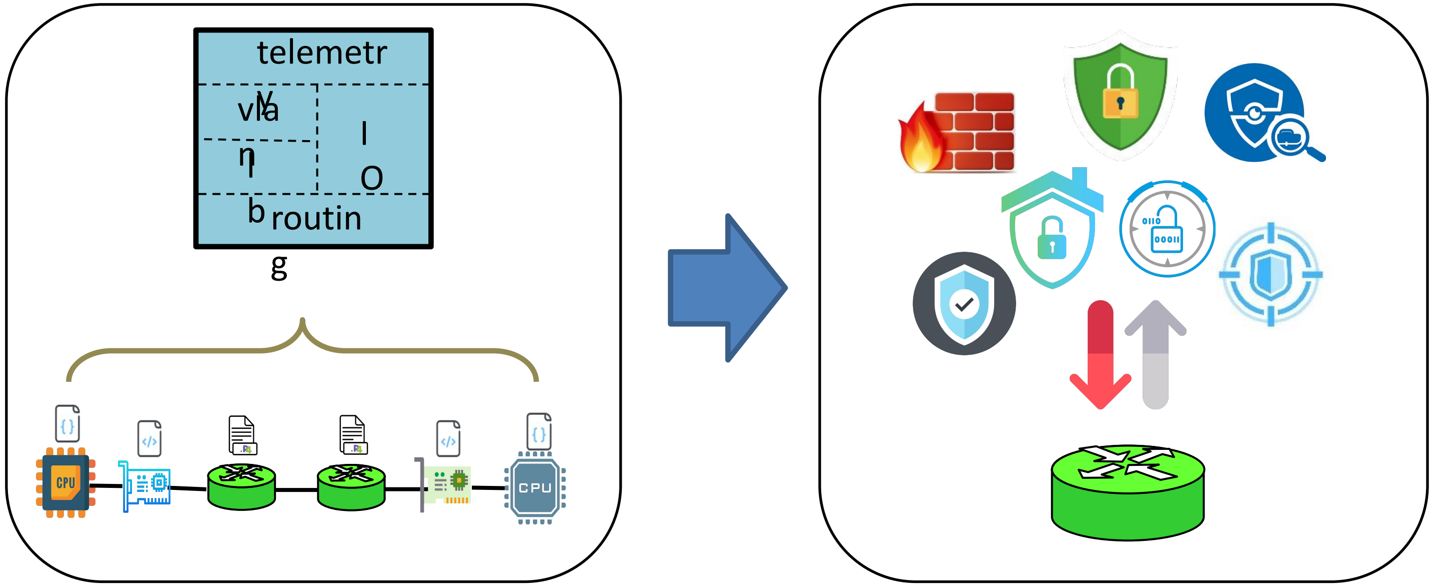
Ang Chen

Department of Computer Science
Rice University

SW functions increasingly sunk into HW

Open, programmable network devices

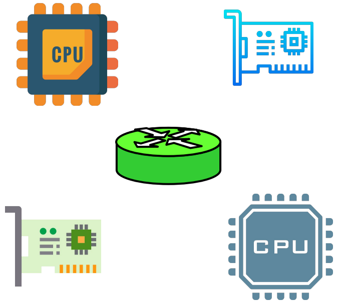
Resource-fungible, runtime changes



- Hardware must have software-like flexibility
  - Programmability: Wide range of tasks
  - Fungibility: Context switches across tasks

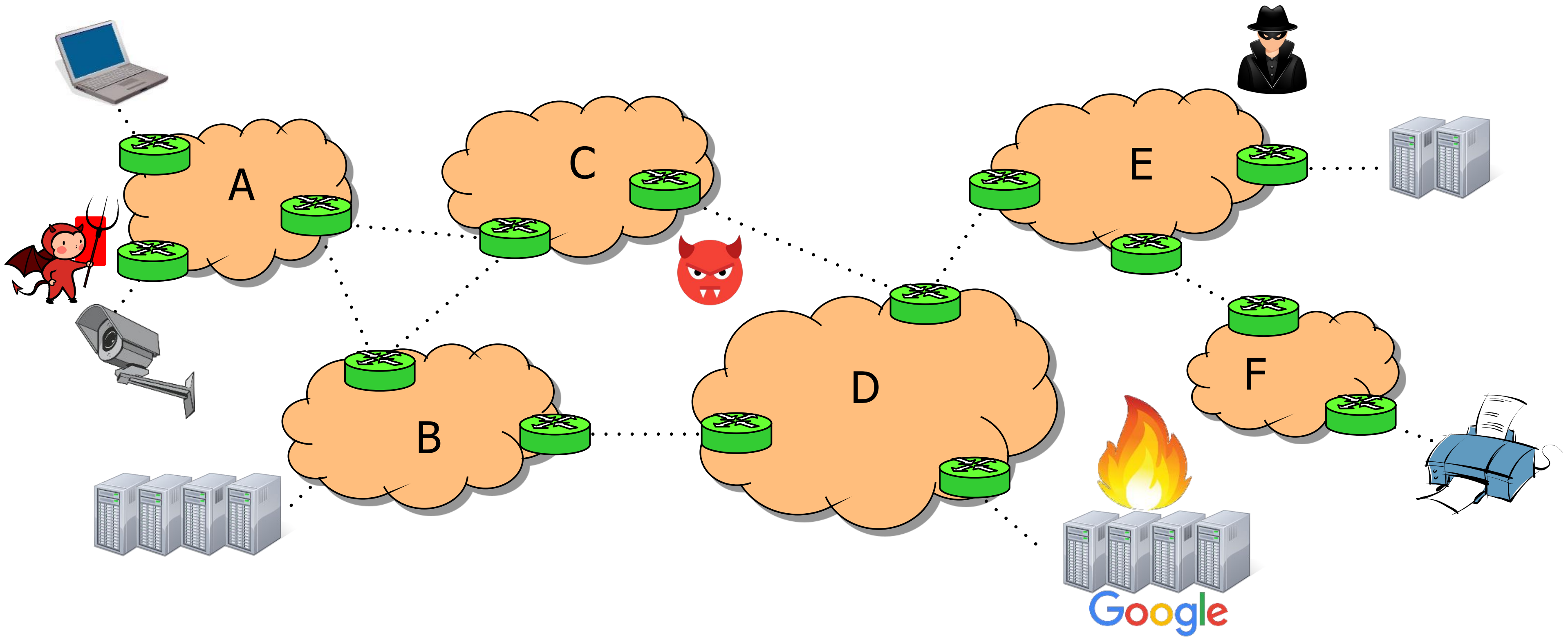
Programmability is already here for networks

Domain-spec. lang. | Many targets | Diverse vendors | Exciting "apps"



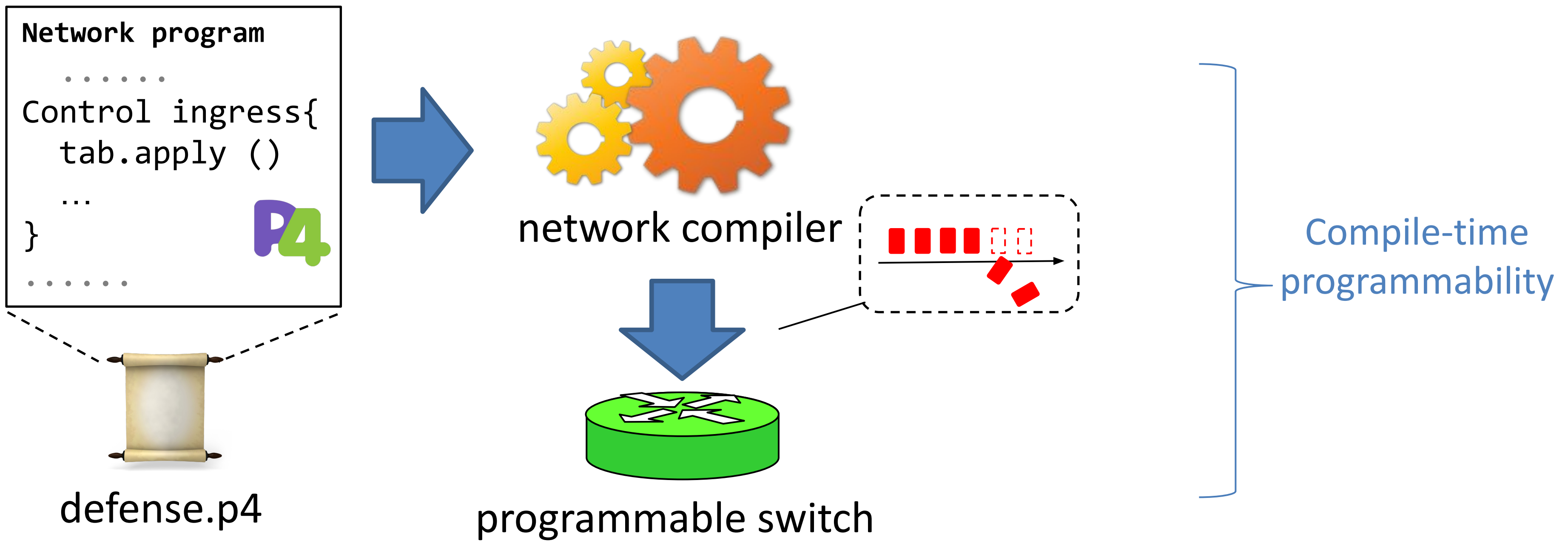
- Programmable network devices are prevalent
  - E.g., SmartNICs, DPUs, IPUs, DSCs, Switches
  - Capable of many tasks, easy feature development
  - Use cases: Security, telemetry, monitoring, ..
  - Some of our work: Ripple, NetWarden, Poise, Bedrock

Ex: Programmable network defenses



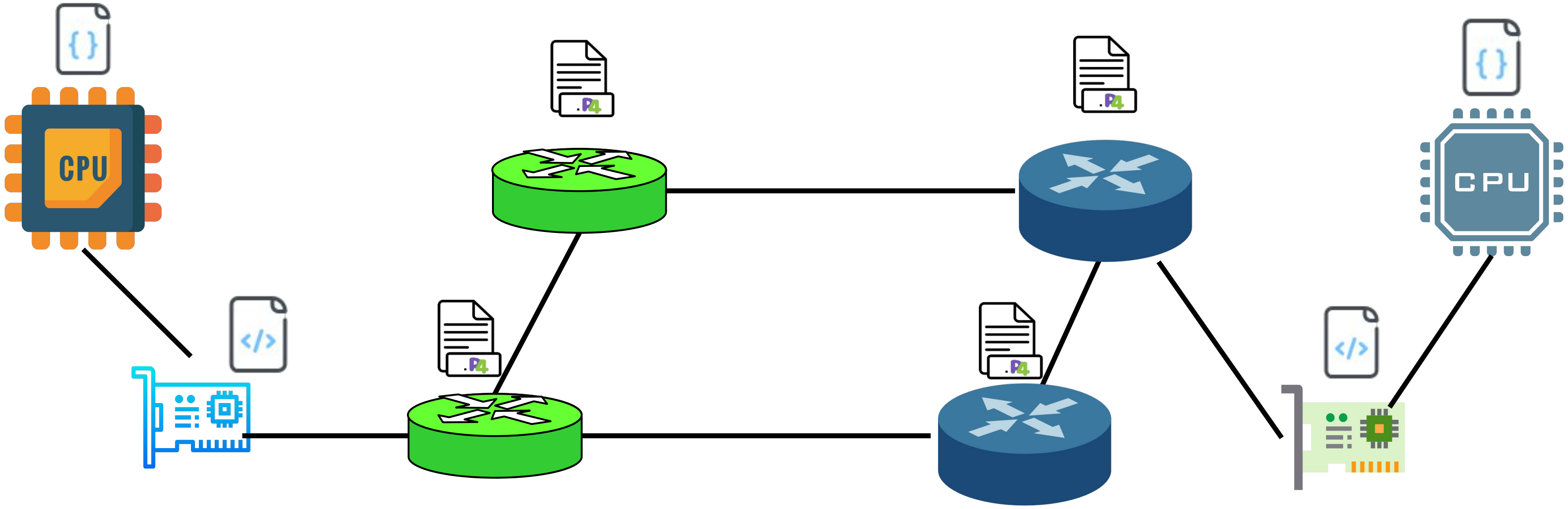
- Network attacks are dynamic and shapeshift quickly:
  - Changing attack strategies and locations
  - Programmable network defenses are a great match
- But defenses must be reconfigurable at runtime!

Today's programmable devices lack fungibility



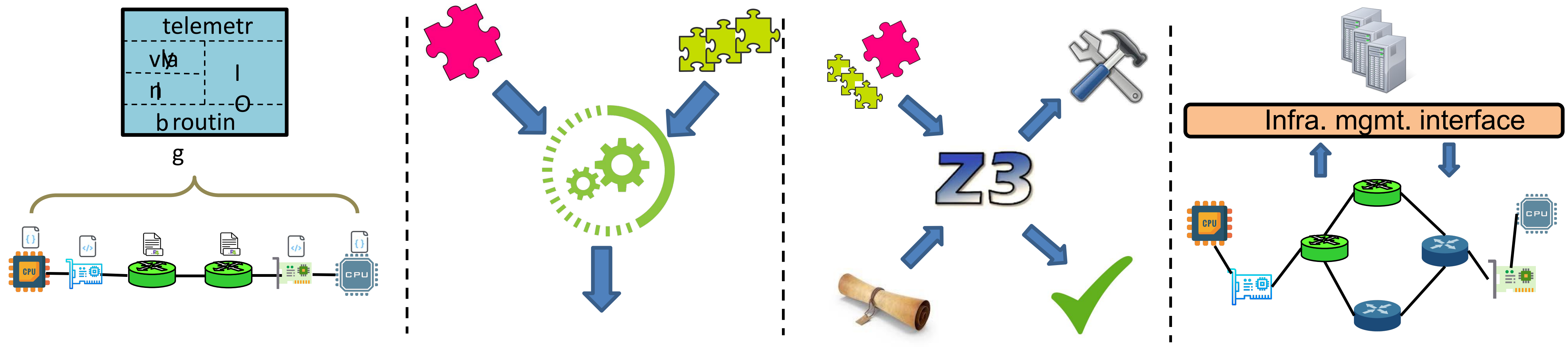
- Today, network programming is a compile-time activity
  - Incurs intrusive downtime, requires maintenance before reprogramming
- Also, can't pre-reinstall all programs we'll ever needed
  - Can't anticipate attacks, limited switch resources (e.g., 10Mb SRAM)

The FLEX vision



- Runtime network (re)programming end-to-end
- No downtime, zero packet loss, consistency guarantees
- From programmability to resource fungibility

Exciting challenges, require community work

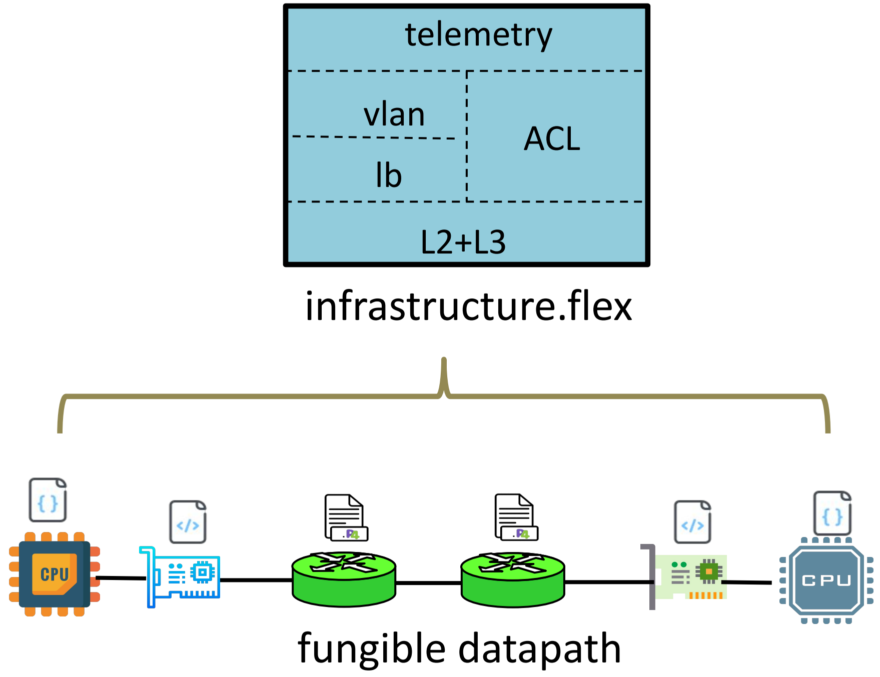


(a) Fungible datapath programming (b) Runtime compilation (c) Runtime infra. verification (d) Real-time infra. control

- Runtime changes to infrastructure stacks is challenging
  - SW/HW "touchpoints" create coupling, changes impact upper layers
  - Including network switches, but also NICs and OS
- Vision: Network/Infra stacks with resource fungibility
  - Programming, compilation, verification, and management

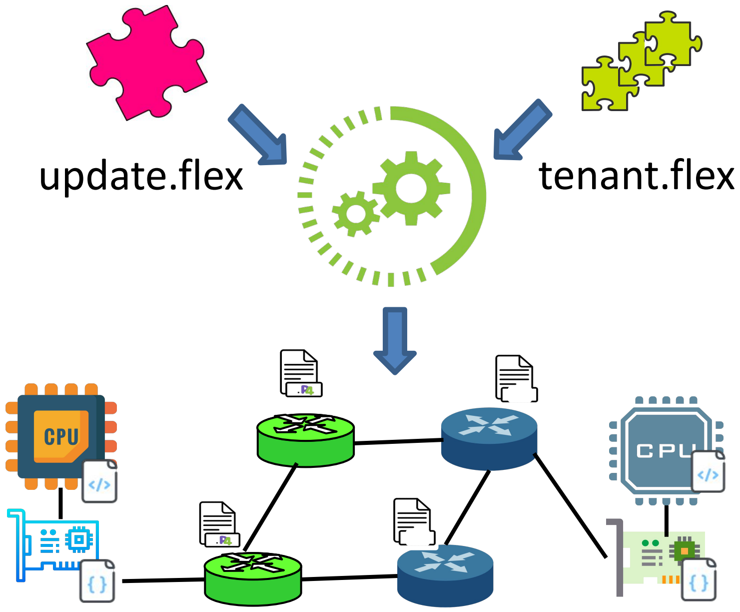
Exciting Questions

Programming a fungible datapath (flex program)



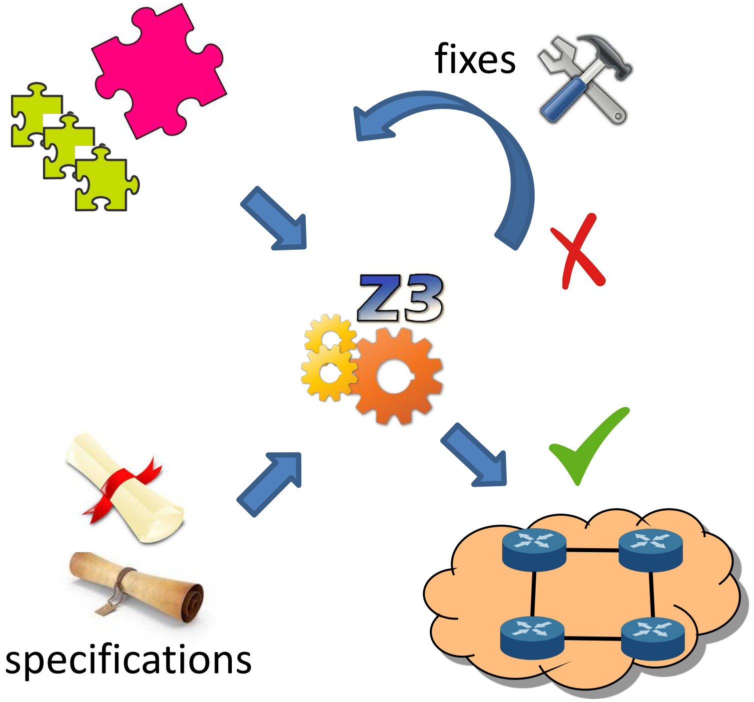
- How to enable a resource-fungible datapath across the stack?
  - Runtime resource allocation + reclamation, without downtime
  - SOTA: P4, NPL, PoF languages specify single-device behaviors

Real-time network extensions (flex extensions)



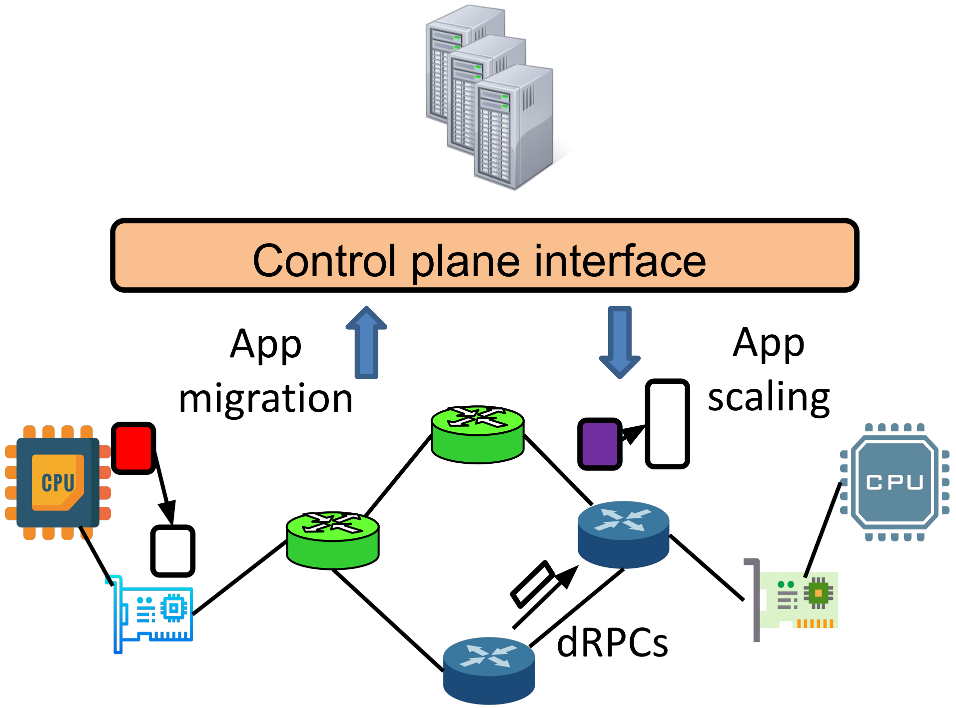
- How to program network extensions into a “base” program?
  - Infrastructure program: Basic utilities, e.g., ACL, telemetry
  - User-specific upgrades, e.g., DDoS, refined telemetry
  - SOTA: BPF extensions to OS kernels, at well-defined hooks

Verifying real-time changes (flex verification)



- How to provide high assurance for runtime changes?
  - Infrastructure changes are risky, especially at runtime
  - Runtime verification to eliminate bugs, constrain blast radius
  - SOTA: P4 verification and validation “before deployment”

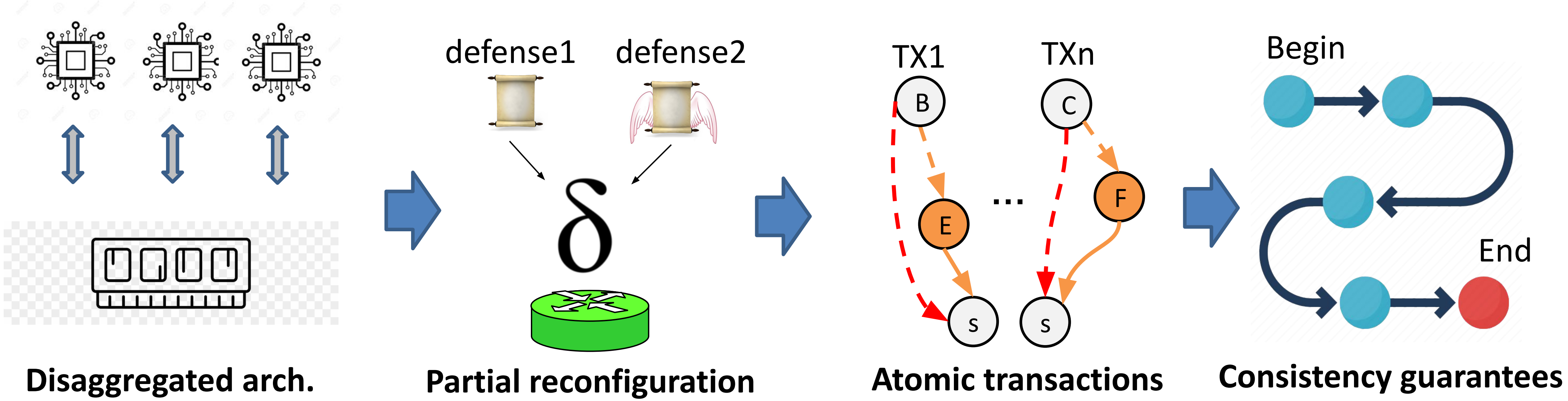
Runtime infrastructure management



- How to manage dynamic network programs as they roam?
  - Network "apps" migrate, expand, shrink at runtime
  - E.g., adding resources dynamically to attack locations
- SOTA: P4Runtime for micro-behaviors not macro-behaviors

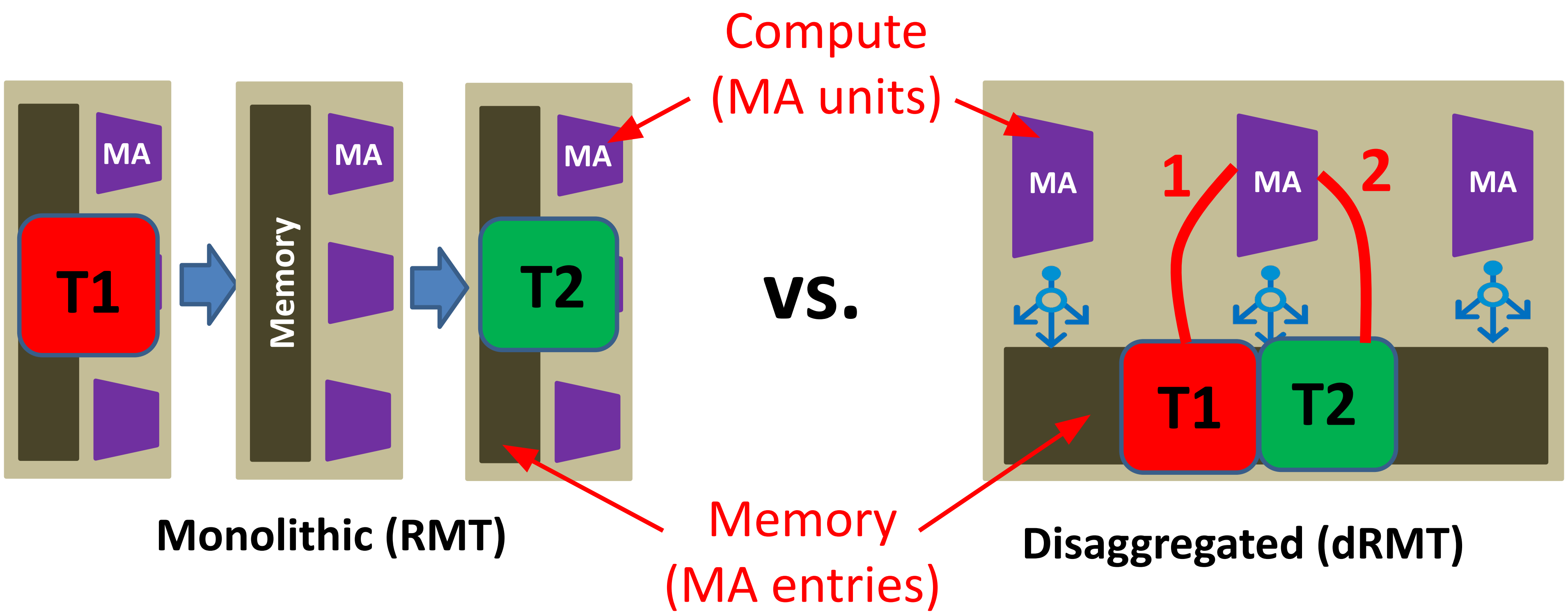
Preliminary Work

Runtime programmable switches



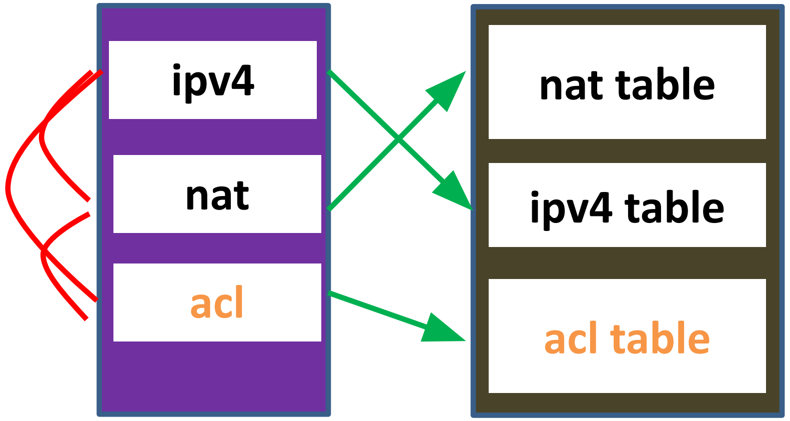
- Goal: Live network reprogramming w/ consistency guarantees
  - Use cases: Real-time attack mitigation, workload-driven optimizations, ..

Disaggregation offers runtime flexibility



- Monolithic: Tight coupling of memory/compute in stages
- Disaggregated: Decoupling for resource fungibility

Ex: Runtime table addition



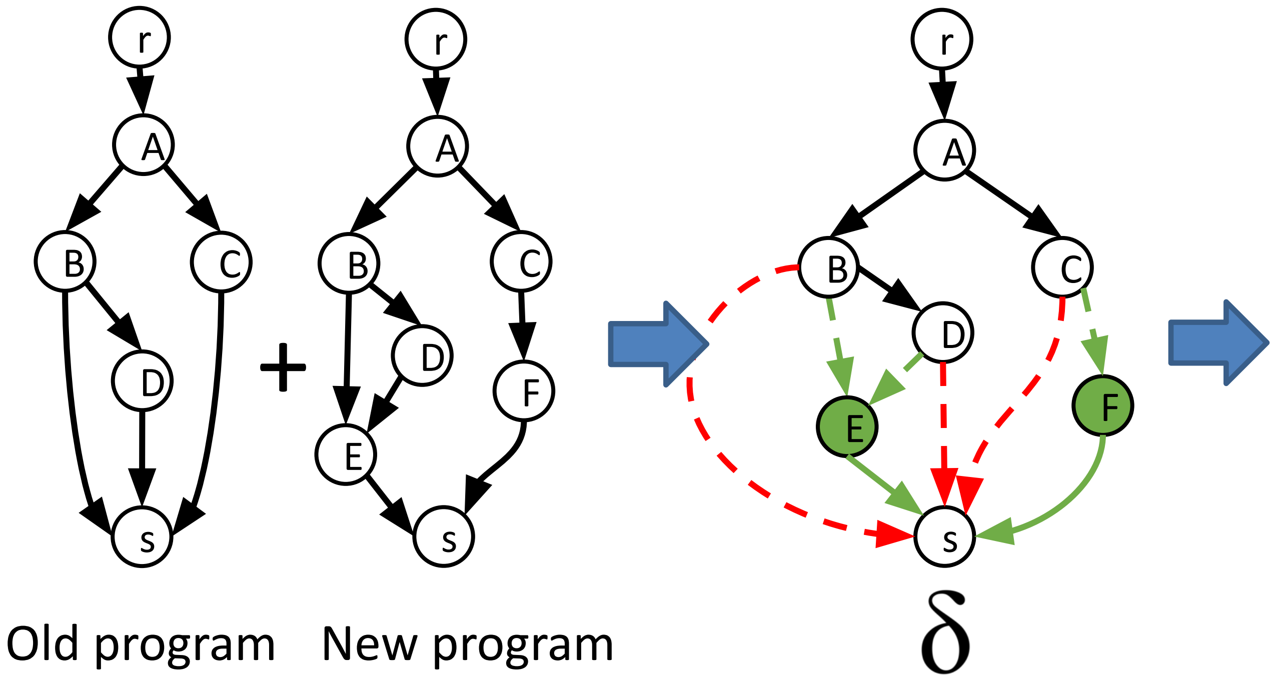
Compute   Memory

```
control ingress {
   ipv4.apply(); // 3Mb
+  acl.apply(); // 5Mb
   nat.apply();  // 5Mb
}
```

P4 program

- Ex: Insert Access Control List (ACL) into a live program
  - Install new elements in scratchpad, pointer swaps to place them in
- Finally, activate changes atomically for next pkt

Live, partial hardware reconfiguration



Primitive	# HW Ops
AllocTbl	112
GroupTbl	112
AllocCond	43
SetCondPtr	2
AllocEx	3
AllocState	22
AllocTrans	5
SetPtr	1

MA Tables

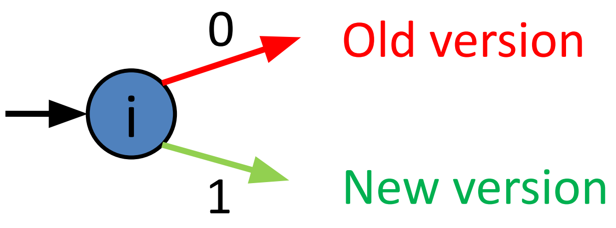
Control flow

Parser state and transition

Atomic

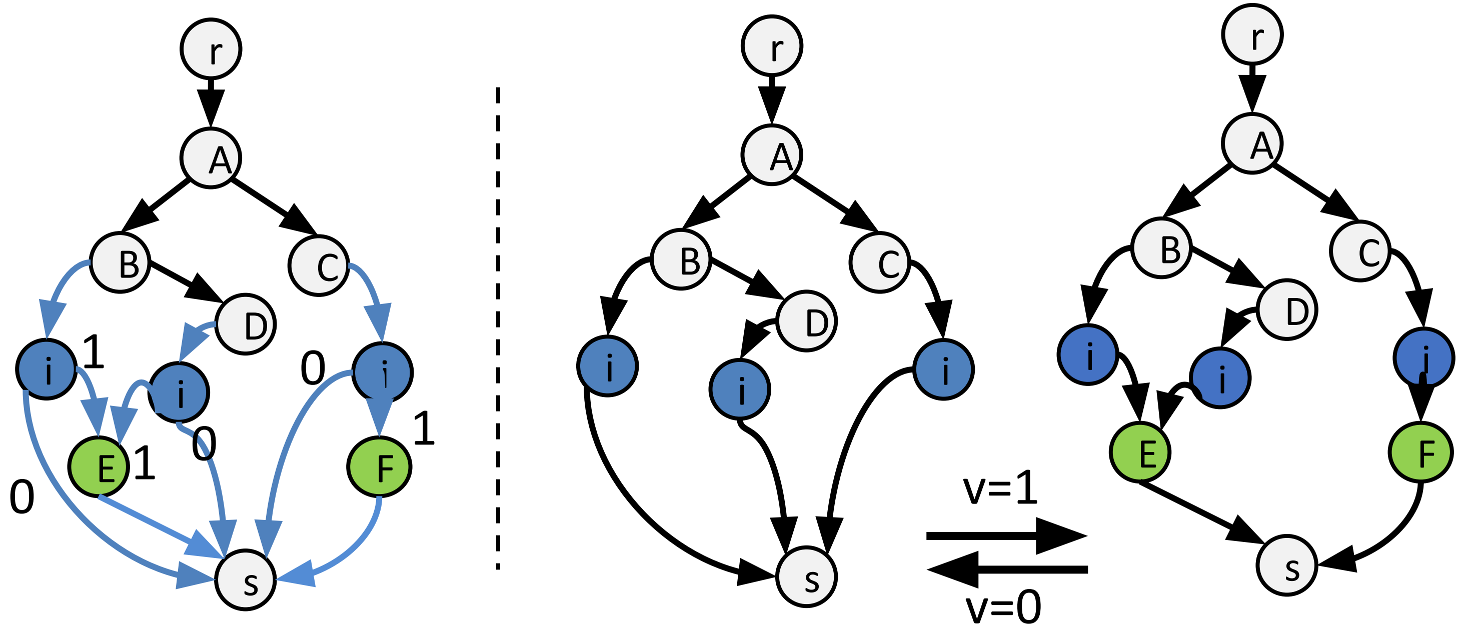
- Larger change: Use "delta" between old and new
  - Approach 1: minimum change graph (NSDI'22)
  - Approach 2: @add, @del, @mod annotations (NSDI'23)
- Transform "delta" into a set of PR primitives

Providing atomic transactions




```
if (metadata.v == 0) {
    next = old_table;
} else {
    next = new_table;
}
```

Atomic hardware operations



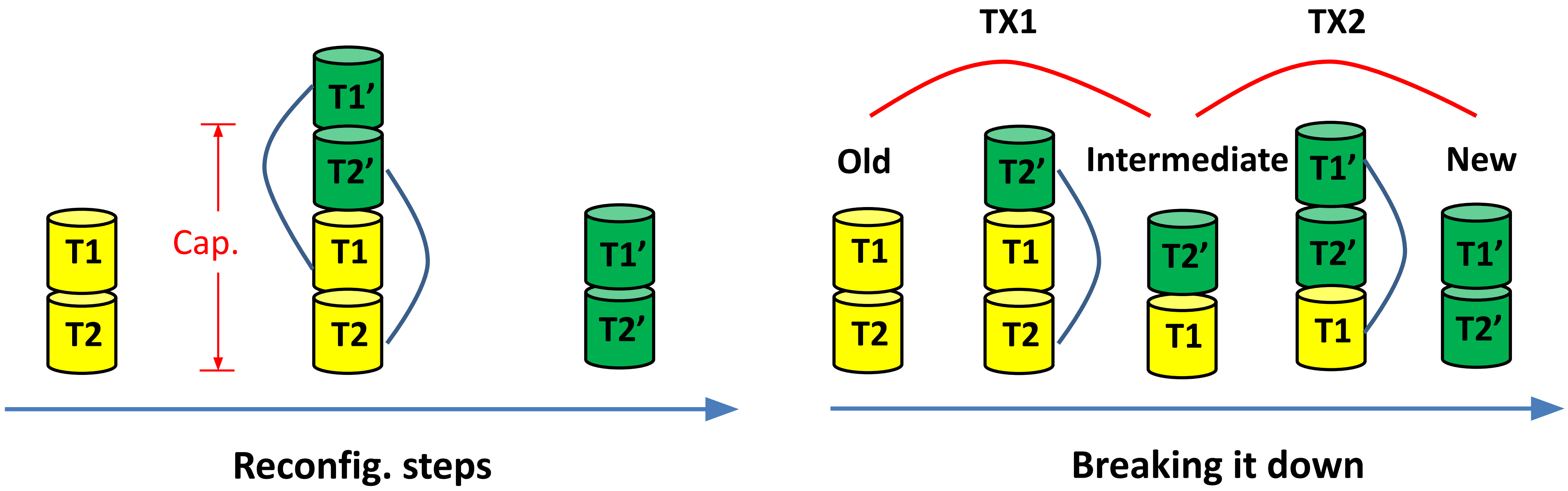
Delta w/ version control

v=0, old

v=1, new

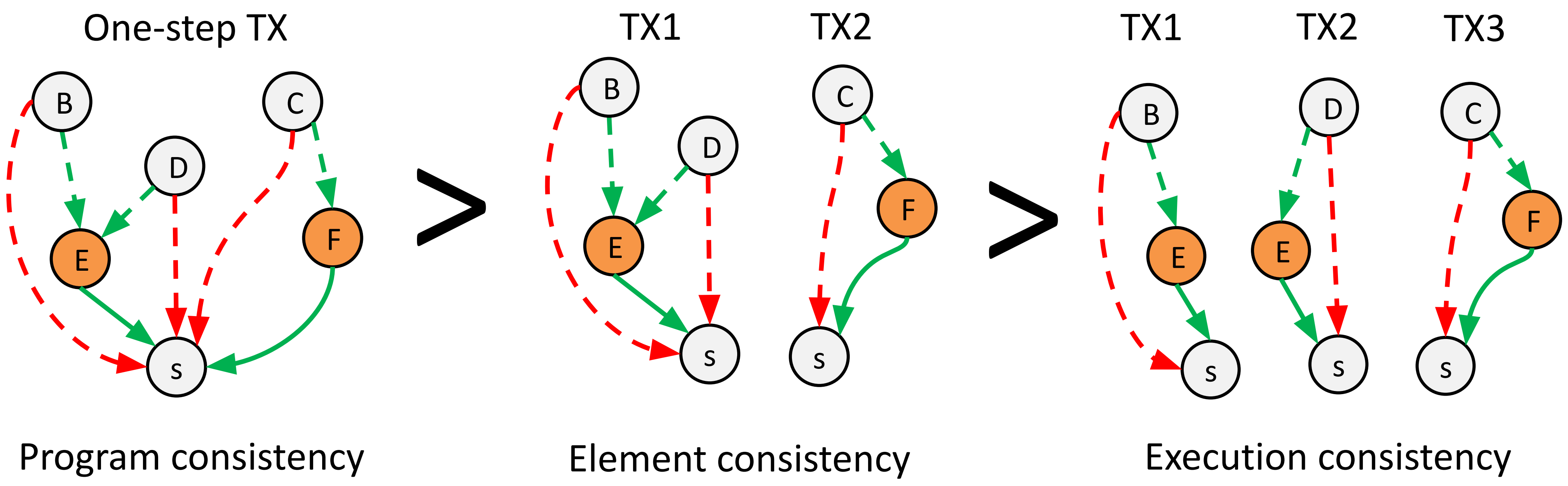
- Idea: Bootstrap from atomic operations to transactions
  - Prepare delta in scratch area, guarded by version control
  - Atomic version modification commits transaction
- But, need to prepare the entire delta before activating it

Resource headroom constrains TX sizes



- One-step TX not always possible b/c resource constraints
  - Ex: {@add T1', @del T1, @add T2', @del T2} ☐ must add before del
  - One-step TX: large peak utilization. Two-step TXs: Feasible
- In between TXs, we'll expose intermediate states!

Consistency guarantees



- Idea: Weaker consistency guarantees w/ granular TXs
  - E.g., pkts never mix old and new tables
  - E.g., User-defined specifications

Runtime update plan synthesis

Additions → Deletions →

```
control ingress {
  @add ecmp.apply();
  if (ipv4.isValid) {
    nat.apply();
  }
  @del {
    if (ipv6.isValid) {
      flowlet.apply();
    }
}}
```

Program w/ annotated changes

```
specification {
  ghost sawNew, sawOld;
  @old => {sawOld = true;}
  @new => {sawNew = true;}

  exec_consistency = {
    !($cur.eg.sawOld &&
     $cur.eg.sawNew);
  }
  assert exec_consistency;
```

Ghost variables

Constrain intermediate states w/ GVs

Assertions

Consistency specification

- User provides change annotation and consistency spec.
- Goal: Identify a sequence of safe and feasible TXs
  - Safe: Intermediate states between TXs satisfy spec
  - Feasible: Each TX fits within the resource headroom

An experiment with Nvidia ASIC

Nvidia 12.8Tbps ASIC

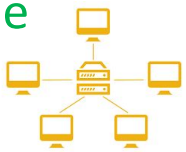
bmv2 emulator

Implementation

- Number of HW operations
- Consistency algorithms
- Transaction sizes, headroom
- Effective for diff. programs
- Intermediate program states

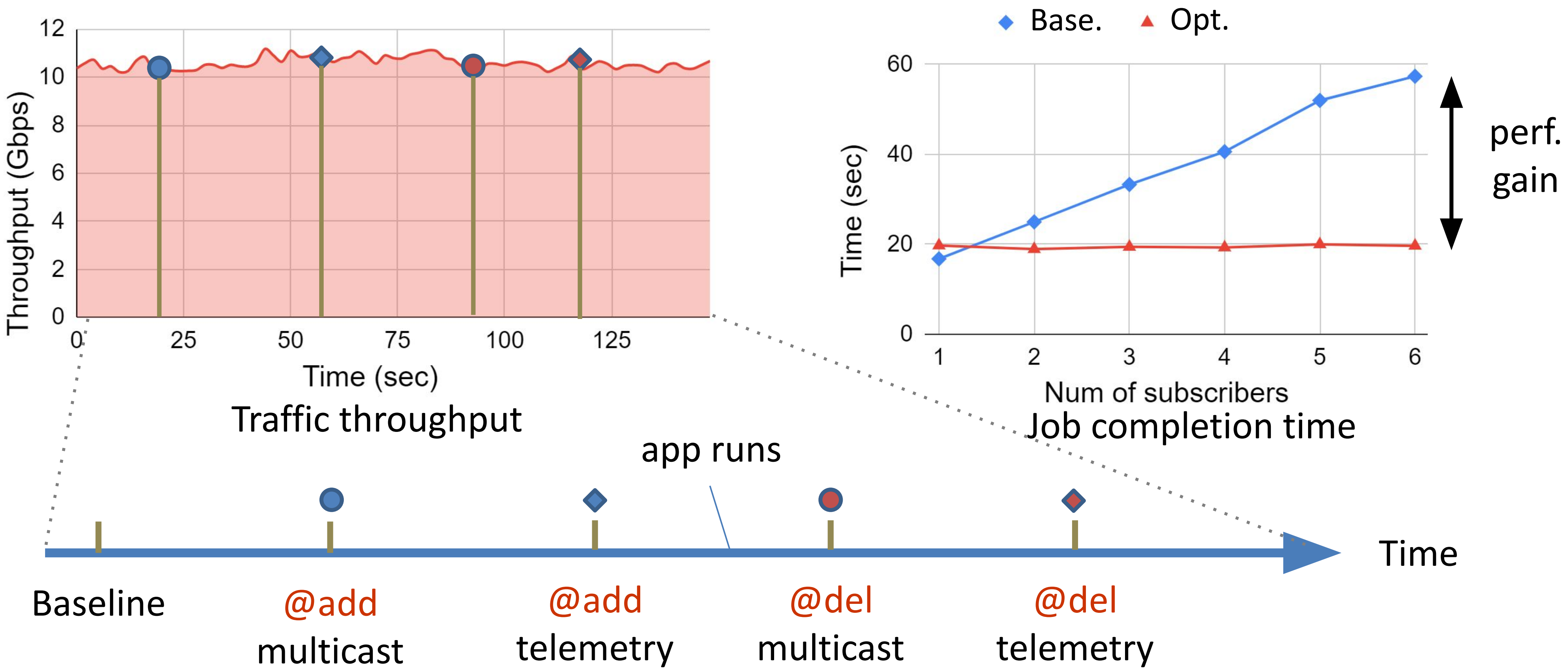
Evaluation metrics

- Real-time network defense
  - Covert channels
  - Access control
- Just-in-time optimization
  - Accelerated multicast
  - Scenario: ZeroMQ multicast w/ 1-6 receivers
  - FlexCore: Just-in-time injection of switch multicast program



Case studies

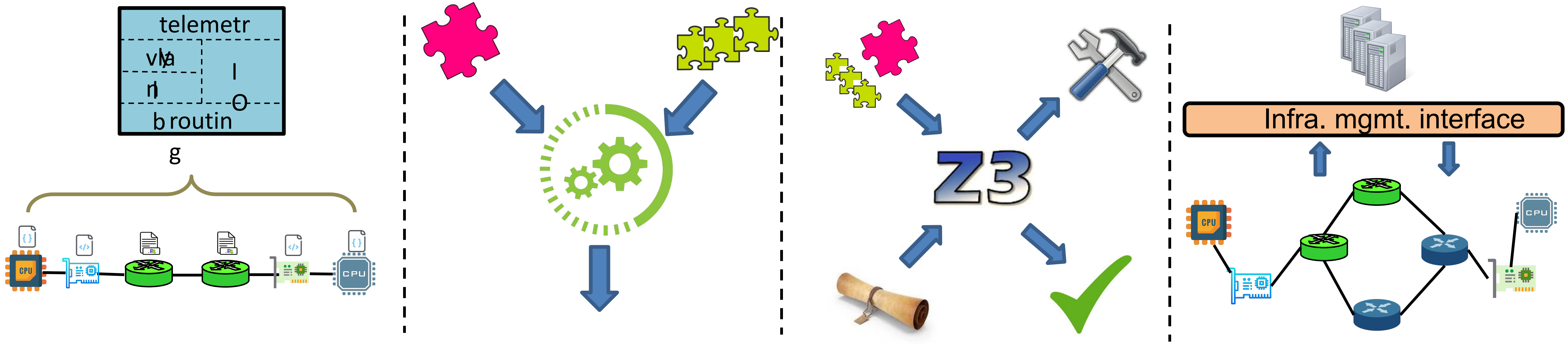
Case study: Accelerated multicast



- Pub/sub workload: Repeated unicast vs. switch multicast
  - Multicast program injected to switch at runtime
- Zero packet loss; dramatic performance gains

But More
Is Needed

Open questions abound, across the stack



(a) Fungible datapath programming **(b) Runtime compilation** **(c) Runtime infra. verification** **(d) Real-time infra. control**

- Fungibility as a first-order design goal
  - Device architectures?
  - Languages and abstractions?
  - Compilation and verification?
  - Network management stacks?

An academia/industry coalition

https://flexnet-project.org FLEX

Google amazon AMD ONF Open Networking Foundation CLOUDFLARE intel vmware NVIDIA TII CISCO Microsoft MANDIANT ASTRONOMER


National Science Foundation
WHERE DISCOVERIES BEGIN

RESEARCH AREAS FUNDING AWARDS DOCUMENT LIBRARY NEWS ABOUT NSF

Awards

- Search Awards
- Recent Awards
- Presidential and Honorary Awards
- About Awards
- How to Manage Your Award
- Grant General Conditions
- Cooperative Agreement Conditions
- Special Conditions
- Federal Demonstration Partnership

Award Abstract # 2214272
Collaborative Research: CNS Core: Large: Runtime Programmable Networks

NSF Org:	CNS Division Of Computer and Network Systems
Awardee:	WILLIAM MARSH RICE UNIVERSITY
Initial Amendment Date:	July 14, 2022
Latest Amendment Date:	July 14, 2022
Award Number:	2214272
Award Instrument:	Continuing Grant
Program Manager:	Darleen Fisher dlfisher@nsf.gov (703)292-8950 CNS Division Of Computer and Network Systems CSE Direct For Computer & Info Scie & Enginr
Start Date:	July 15, 2022
End Date:	June 30, 2026 (Estimated)

- Anchored by an NSF project, with industry engagement
- Looking for more collaborators and brainstorming partners!

From programmability to fungibility

None (before 2013)	Programmability (2013 – now)	Fungibility (going forward)
Vendors control network logic	Operators control network logic	Users control network logic
Hard to develop and deploy new features	Easy to develop, hard to deploy	Easy to develop and deploy new features
		

We need community work!

Acknowledgments

Aditya Akella
Tom Anderson
Ang Chen
Sushovan Das
Matty Kadosh
Patrick Kon
Arvind Krishnamurthy
Jiaxin Lin
Hongyi Liu
Alan Lo
Jeongyoon Moon
T. S. Eugene Ng
Yonatan Piasetzky
Hari Sezhiyan
Omer Shabtai
Yiming Qiu
Weitao Wang
Zhuang Wang
Crystal Wu
Jiarong Xing
https://jxing.me/