Multitenancy for Fast and Programmable Networks in the Cloud

Tao Wang[*], Hang Zhu[*], Fabian Ruffy, Xin Jin, Anirudh Sivaraman,
Dan Ports, and Aurojit Panda
([*]Equal contribution)

NEW YORK UNIVERSITY

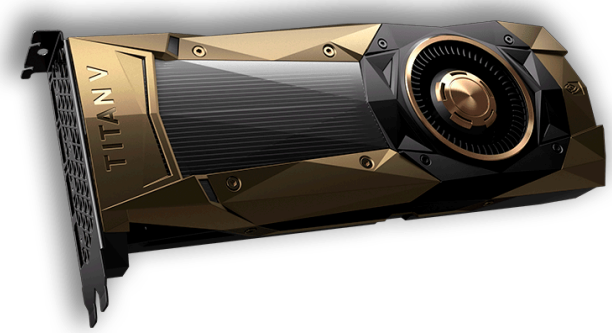
JOHNS HOPKINS UNIVERSITY

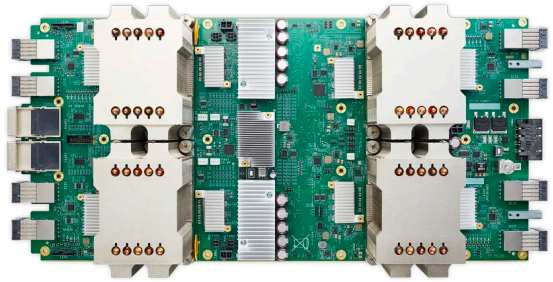
Microsoft Research

What does today's cloud offer as a service?

- Generic compute and storage resources

- Specialized accelerators





Emergence of programmable network devices

- Pipeline-based programmable devices
  - In-network switches
  - At-host SmartNICs

- Enable wide-range innovations for classical networked systems
  - Consensus: NOPaxos, NetPaxos
  - Concurrency control: Eris
  - Caching: NetCache, IncBricks
  - Storage: NetChain, SwitchKV
  - Applications: SwitchML, NetAccel
  - …

Why not offer such system as a cloud service?

- Need of multitenancy support
- Provider's aspect
  - Improve resource utilization
    - One application can hardly consume all the hardware resources
    - Heterogenous resource requirement
- Tenant's aspect
  - Enable innovations
    - New programs can be easily tested w/o impacting basic network functionality

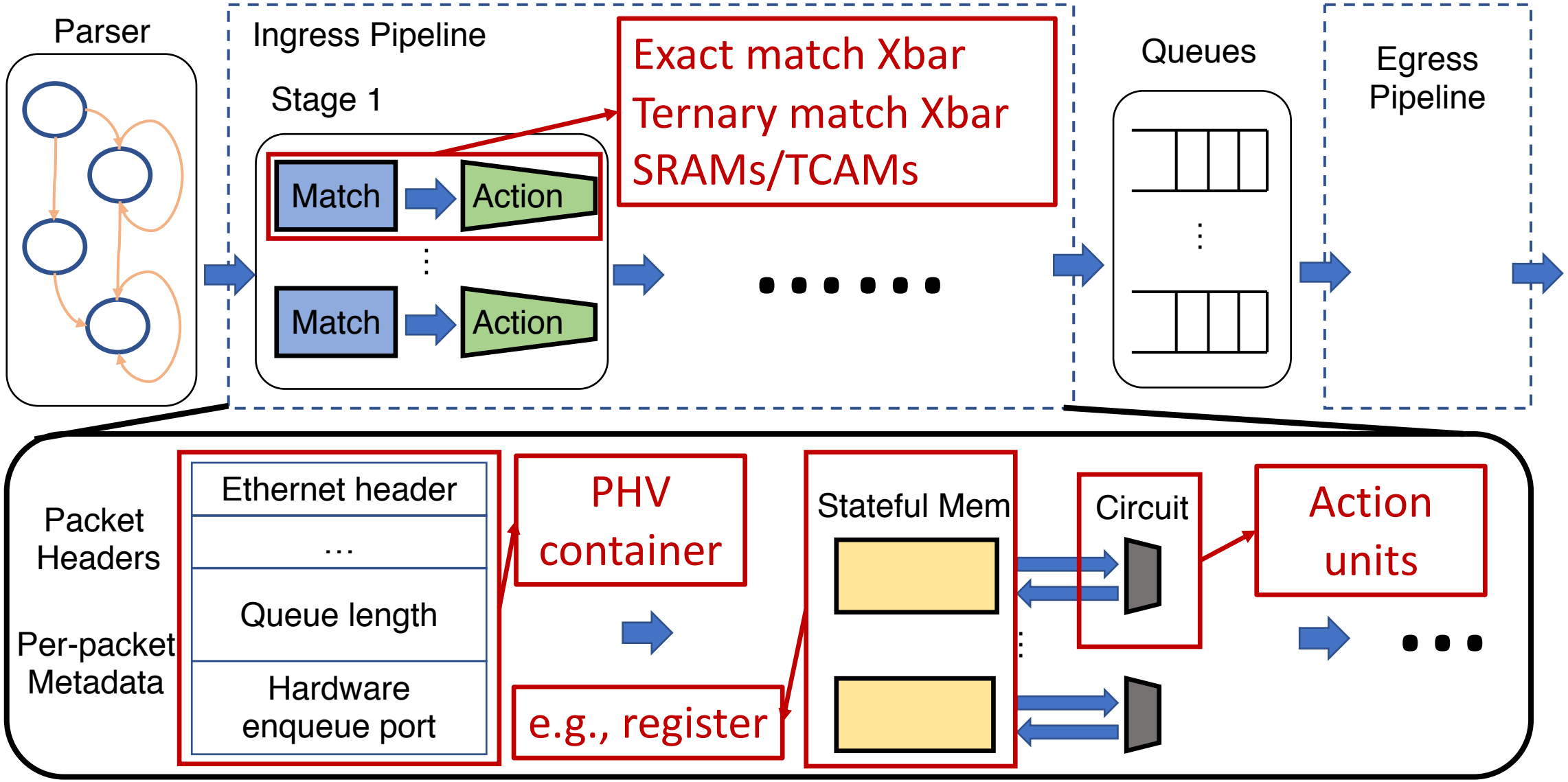
How to enable multitenancy for programmable devices?

Requirements:

- Resource efficiency
  - Little overhead
- Isolation
  - Performance
  - Allocated resource

Our vision: a hybrid compile-time and run-time solution

Background on programmable network devices



Programmable devices' characteristics

- Various types of hardware resources
  - Most of them are decided during compile time
- Limited run-time support
  - Hardware wirings are decided during compile time
    - Line-rate performance achieved after successful compilation
  - No temporal scheduling (e.g., CPU or NPU scheduling)
  - No spatial reconfiguration (e.g., FPGA [AmorphOS, OSDI'18])

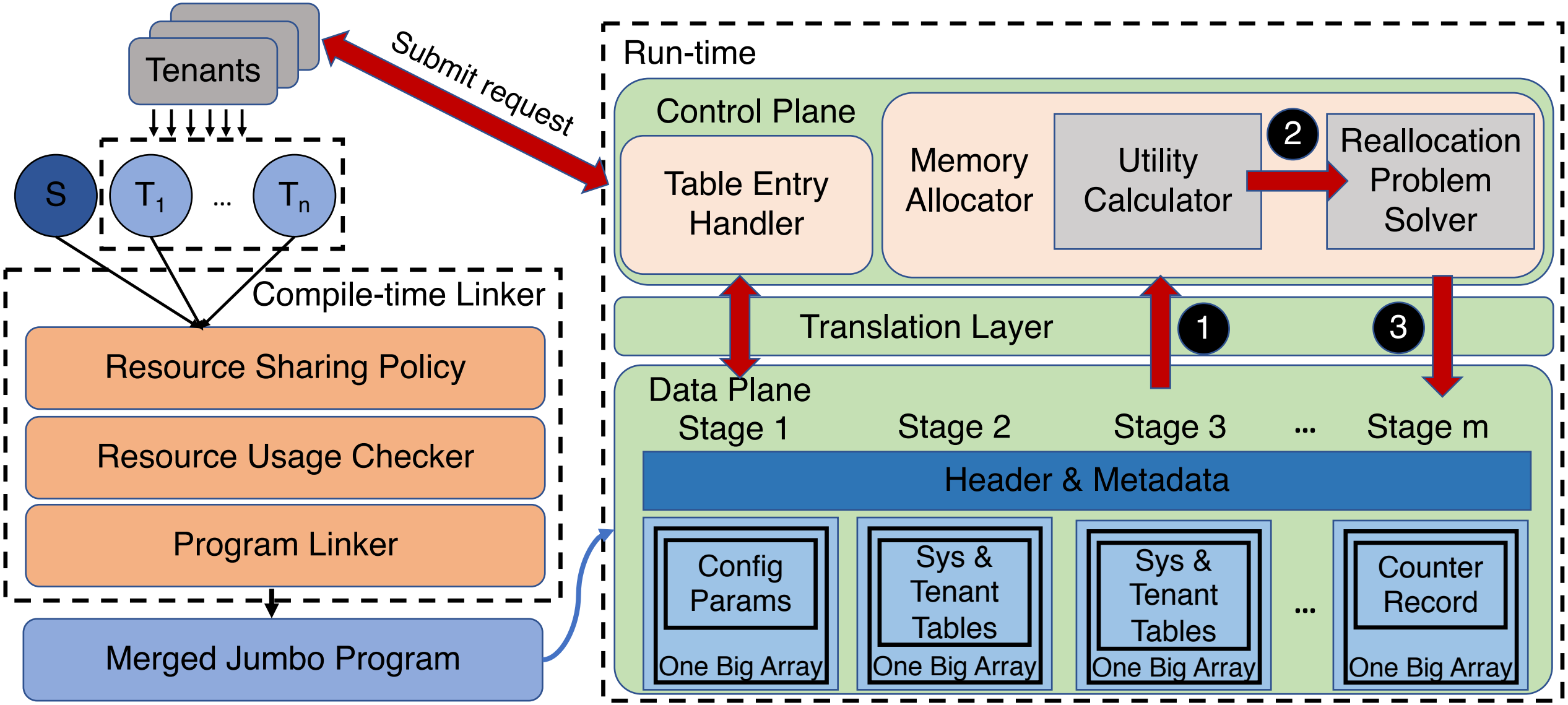
- Resource efficiency
  - Little overhead
- Isolation
  - Performance
  - Allocated resource

A hybrid compile-time and run-time solution

- Compile-time program linker
  - Target generic resources (e.g., SRAMs/TCAMs, action units, etc.)
  - But static

- Run-time memory allocator
  - Target stateful memory
  - But limited

System overview



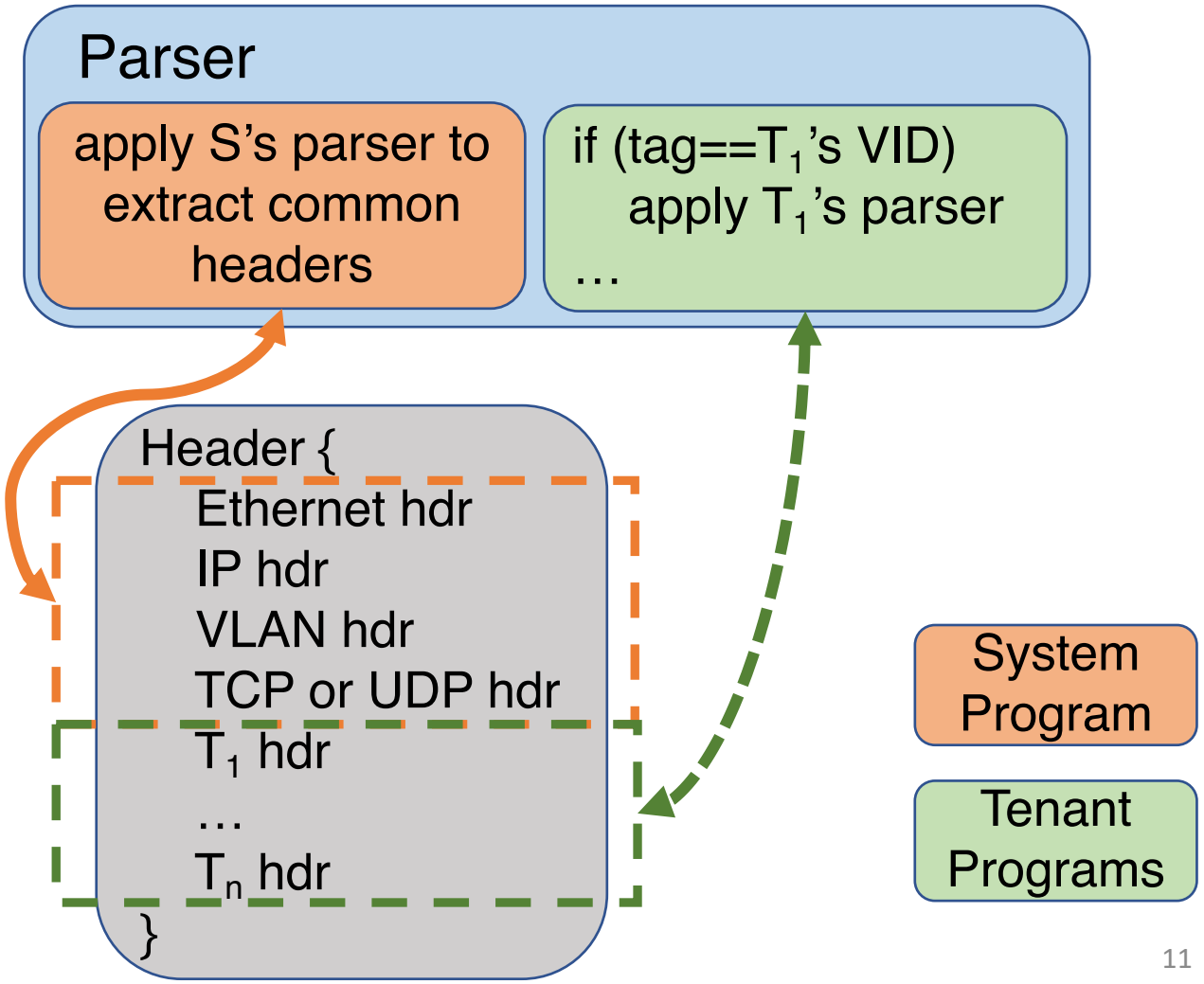
Goals of compile-time linker

- Restrict resource usage

- Provide isolation
  - Ensure tenant program does not inference with others’
  - Ensure no infinite packet resubmitting
  - Ensure no loop forwarding configuration
  - …

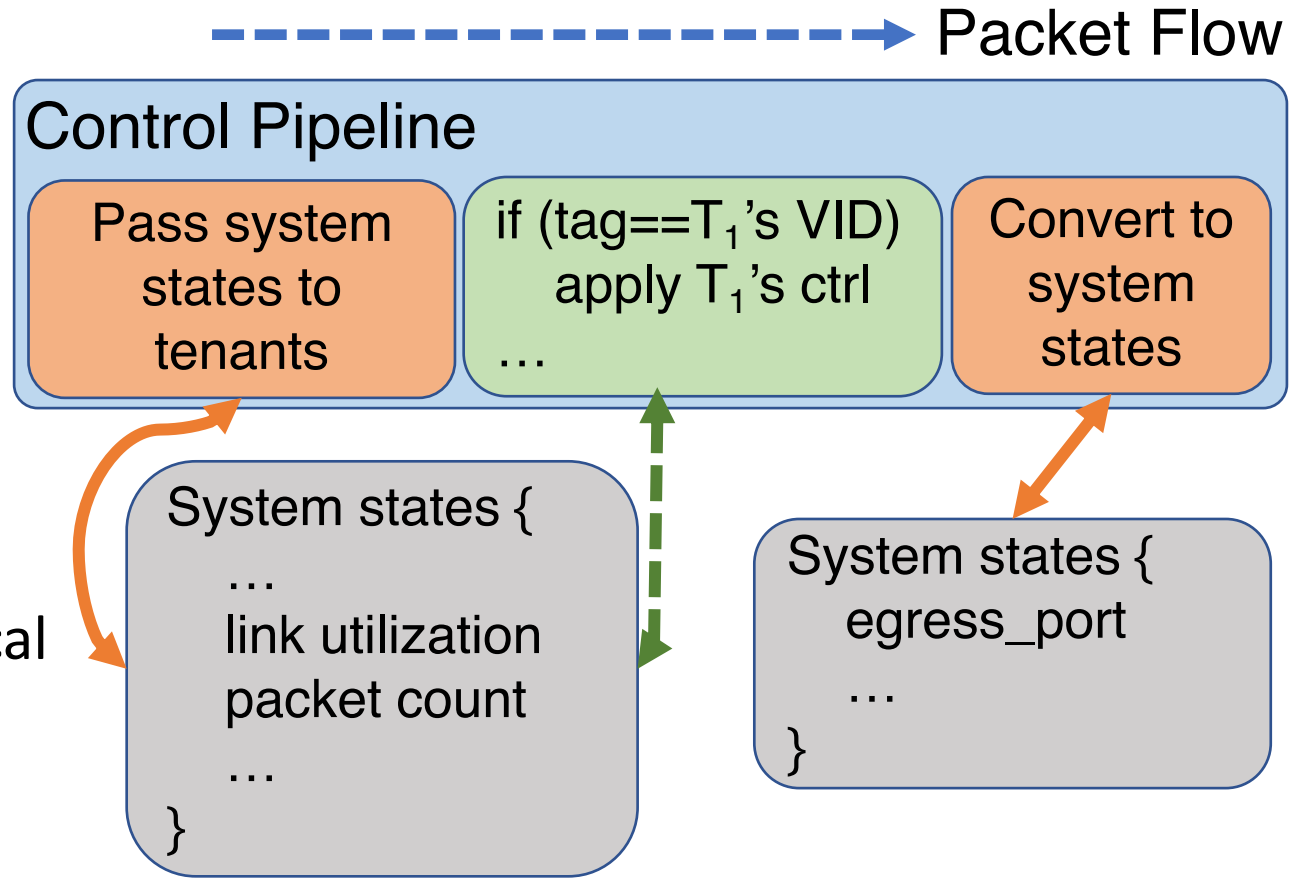
Parser

- Fixed packet format
  - Eth, VLAN, IP, TCP or UDP header followed by custom headers
- System program
  - Extract common headers
- Tenant Programs
  - Extract tenant-defined headers



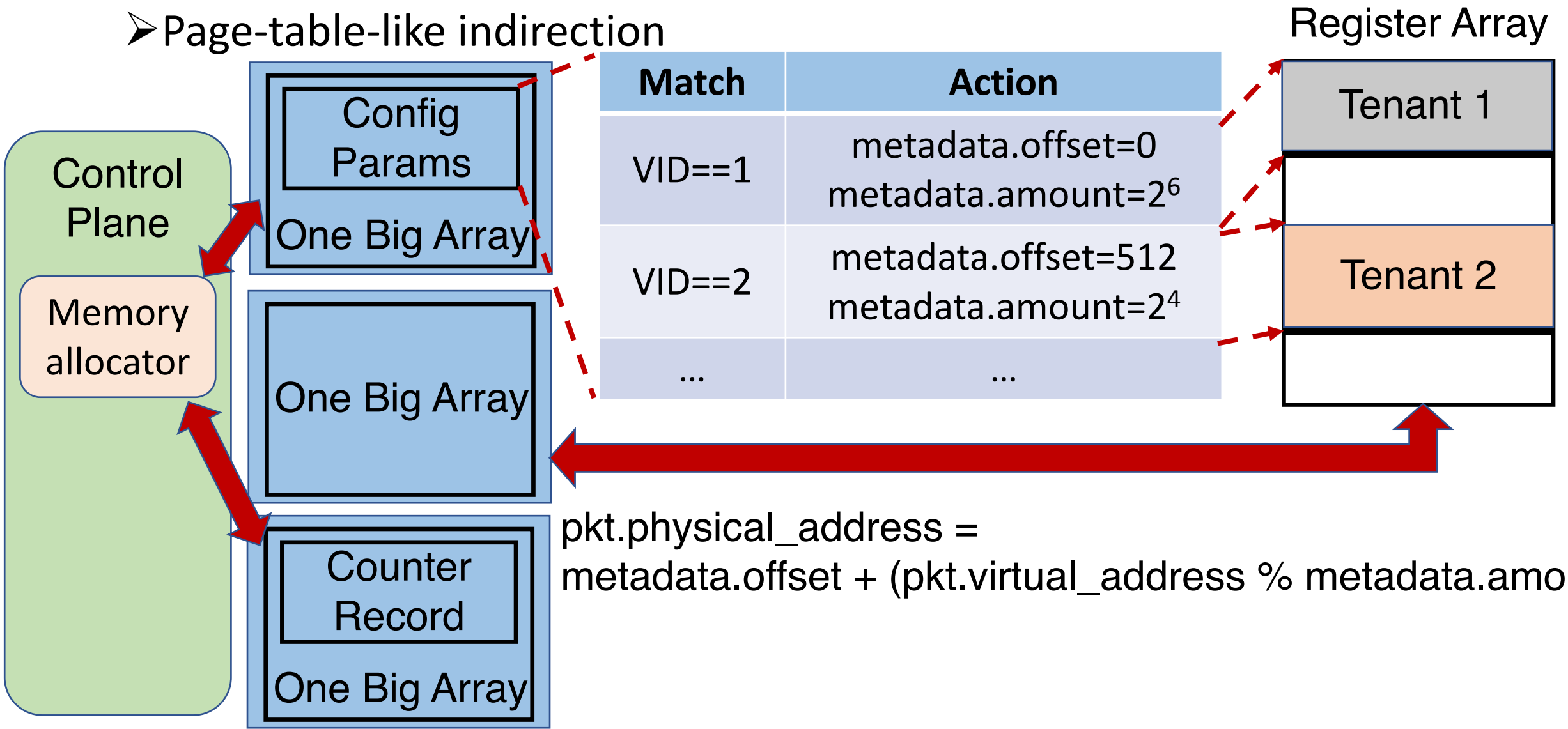
Control (ingress and egress) pipeline

- Feed-forward packet flow
  - "Sandwich" architecture
    - write-then-read half
    - read-then-write half
- System program
  - Interact with tenant programs
  - E.g., pass system states
  - Convert virtual addresses to physical ones



Run-time memory allocator

- Page-table-like indirection



pkt.physical_address =
metadata.offset + (pkt.virtual_address % metadata.amount)

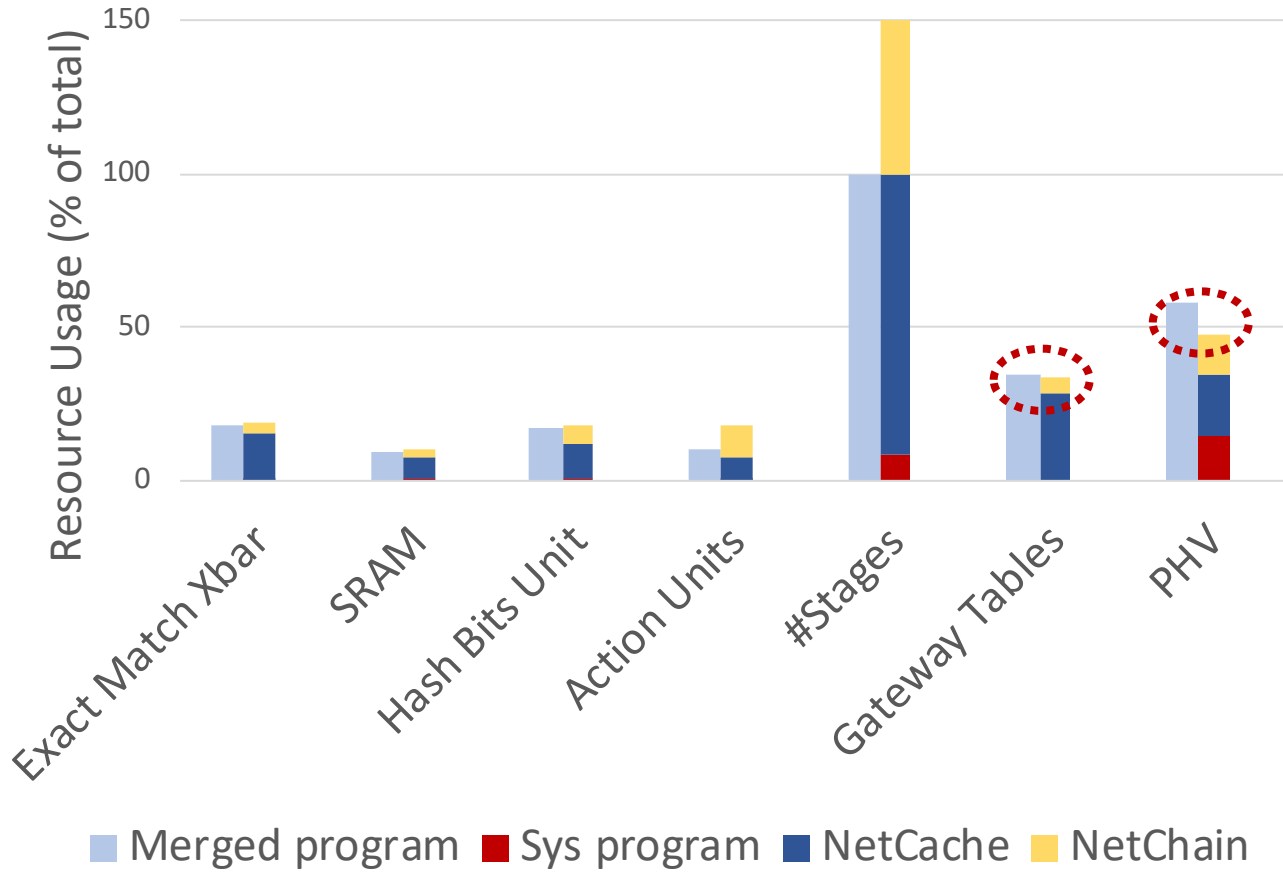
Implementation

- Prototype on Barefoot Tofino switch
- Compile-time linker
  - Extend open-source P4 compiler[1]
- Run-time memory allocator
  - Base on auto-generated APIs to pull records and modify table entries

[1] https://github.com/p4lang/p4c

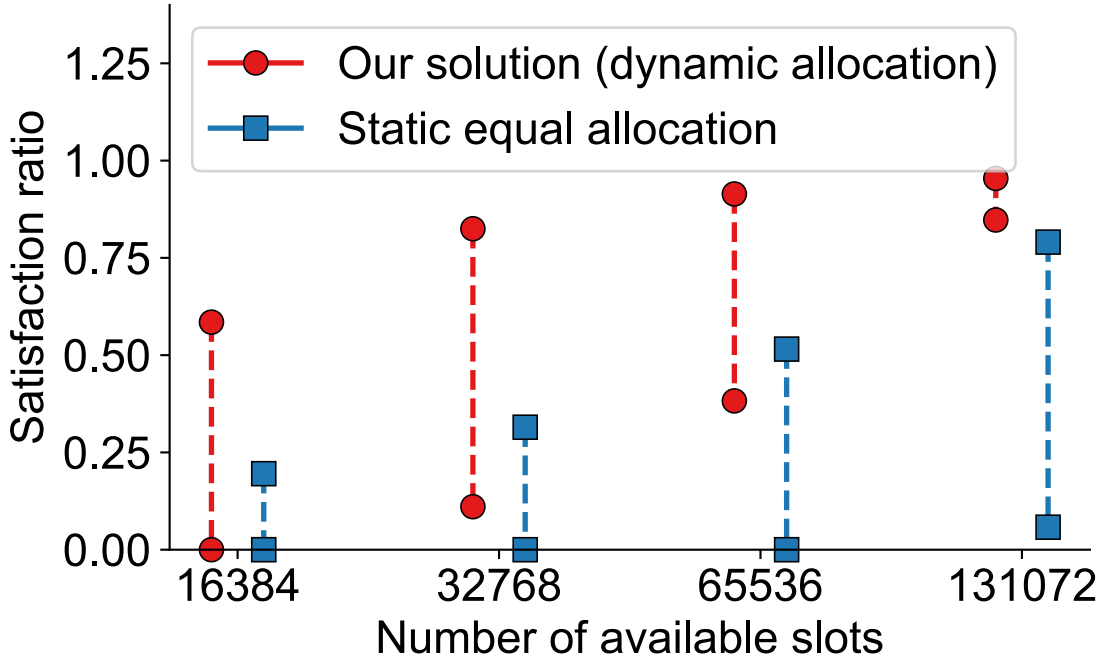
Compile-time program linker correctness

- Resource usage on Tofino
- Packet-level validation on PTF
  - Sys program
    - Basic parsing and forwarding logics
  - [SOSP'17] NetCache
  - [NSDI'18] NetChain
- Overhead
  - Additional gateway tables to check which program to be executed
  - Additional tag-along PHV containers



Run-time memory allocator efficiency

- Experimental Setting
  - 64 tenants submit 1-min heavy hitter detection task against source IP address within its /6 subnets
  - 10-min CAIDA trace replay
- Evaluation metric
  - Utility: memory hit ratio
  - Satisfaction: time fraction w/ utility > 0.9
  - We show the mean and 5th percentile



Conclusion

- Takeaways
  - A hybrid solution for multi-tenancy support
  - Compile-time linker: general but static
  - Run-time memory allocator: dynamic but limited

- Future work
  - Seek new hardware design
    - Both general and dynamic

Thanks!

Happy to take questions

tw1921@nyu.edu