How P4 as a language is evolving

Andy Fingerhut April, 2020

The P4 Language

- Past
- Present
- Future

Past

- P4_14
  - 2014 - first language specification
  - Several new versions developed, one of which was named P4_16
- P4_16
  - May 2017 - Version 1.0
  - Nov 2018 - Version 1.1
  - Oct 2019 - Version 1.2
  - May 2020 - Version 1.2.1

P4_16 Version 1.1

- Catching up to things P4_14 already had:
  - Saturating arithmetic
  - Parser value sets
- New things for programmer convenience
  - functions
  - Optional and named parameters
  - **enum** types that have an underlying **bit**<**W**> type, e.g. **bit**<**8**>

P4_16 Version 1.1

- New and primarily for control plane API
  - **type** keyword
    - Helps control plane API mark values with types such as switch port ids, Ethernet/IPv4/IPv6 addresses, etc.
  - Structured annotations
    - Restricted to lists of simple values, or dictionary of key/value pairs
    - Covers most use cases desired by control plane software developers to add custom annotations to tables, actions, parameters, etc.
    - Restricted structure makes them easier to parse/consume

Version 1.2

- sizeof(header) - spelled my_header.minSizeInBytes()
  - No longer do you need to #define a constant for header sizes
- Control plane API
  - String type, primarily for inclusion in annotations, not for data plane programming
- A dozen other things ...

Present

- LDWG has a goal of trying to keep a regular cadence of updates, which will mean some updates are smaller.
- P4_16 language specification 1.2.1 to be released soon (May 2020?)
  - Structure values expressions
    - Convenient literal syntax for values of a struct or header
      - my_struct = { a = 1, b = 17 };
  - @pure and @noSideEffects annotations to aid compiler in performing more kinds of optimizations, earlier.

“It’s tough to make predictions, especially about the future.”

Yogi Berra

Possible P4 futures

- More day-to-day programmer conveniences
- Making code a bit more reusable and modular
  - Module system
  - Generic types
  - These take just the right combination of knowledge, interest, and free time to develop

Possible futures

- More interest in P4 from companies developing programmable NICs
  - May drive new language features to take advantage of their capabilities.
  - Many packet processing features are inherently more stateful here.
  - Loops?
  - Arrays of structs for something besides header stacks to iterate over?
  - Tables with the option to add entries to them, in the data plane? Use case: flow tables
- externs have been in P4_16 since the beginning, and cover many use cases without extending the P4 language itself
  - e.g. video codec, TCP offload, TCP termination
  - Developing new P4_16 architectures will be at least as interesting here as adding to the language

Possible futures

- Java object system with garbage collection, class implementation inheritance, and interfaces

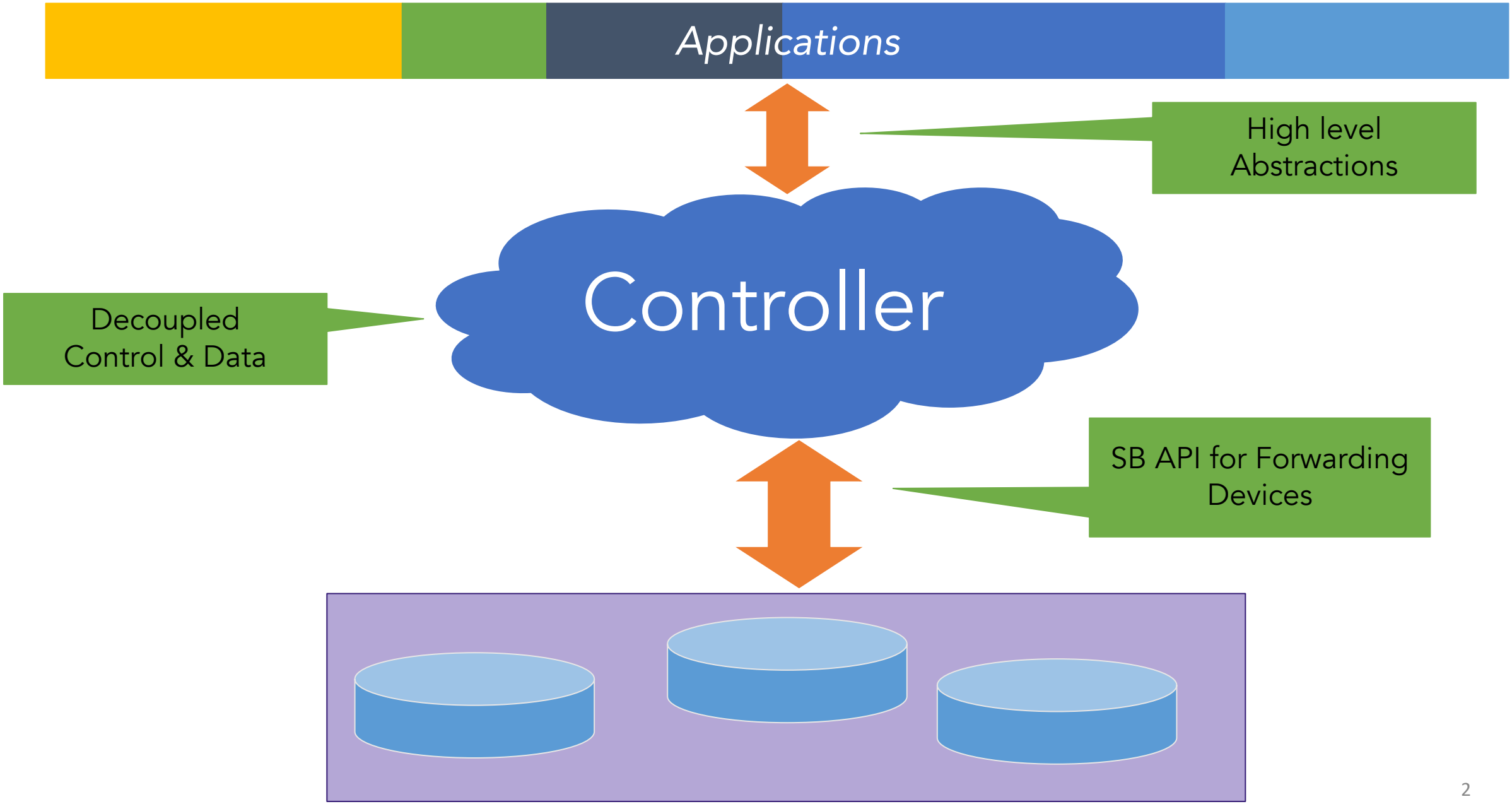
- :-)

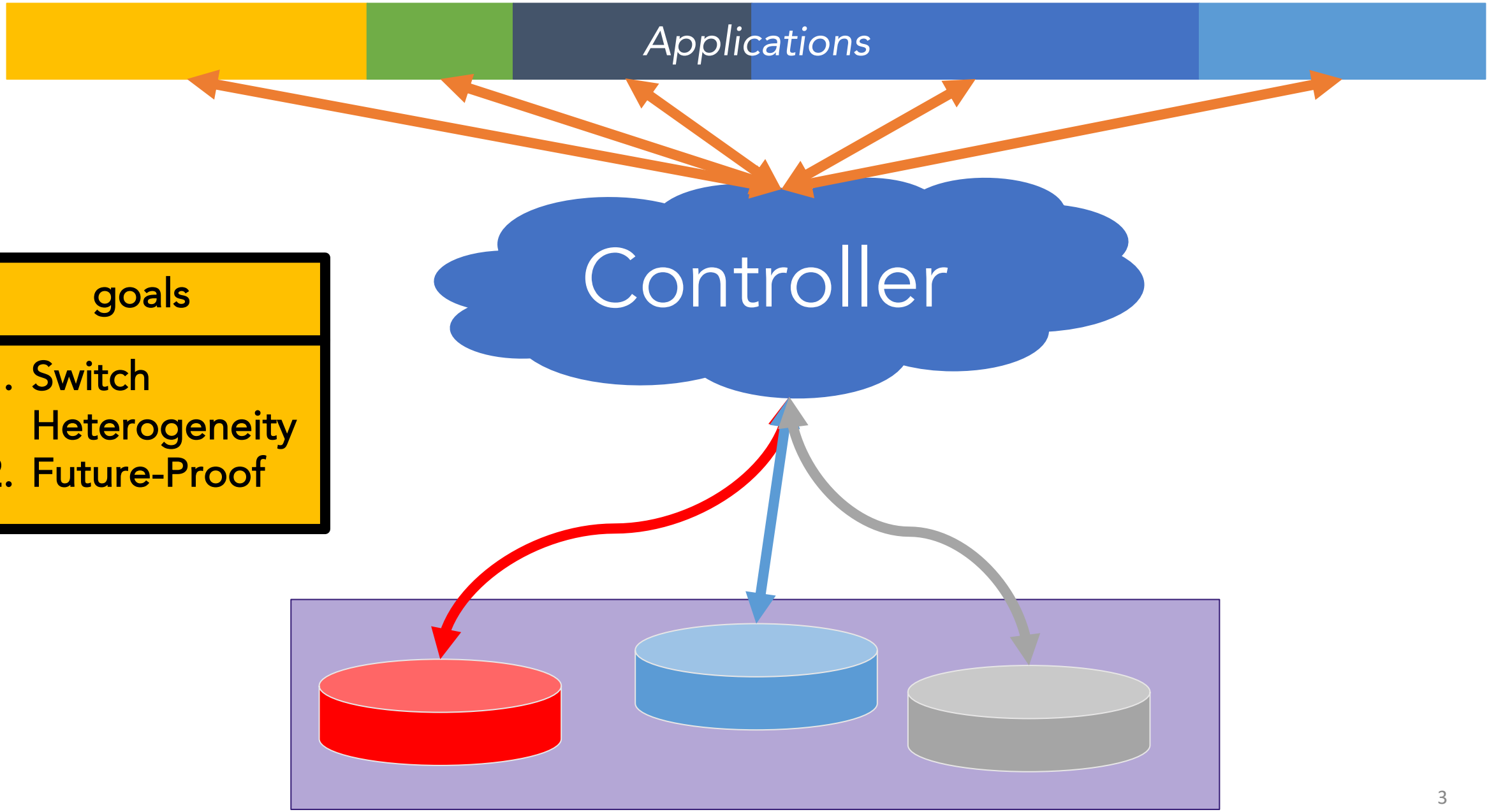
Join in!

- If you have an itch for P4 language enhancement, the Language Design Work Group welcomes participation.

Avenir: Future-Proofing the Control Plane via Data Plane Synthesis

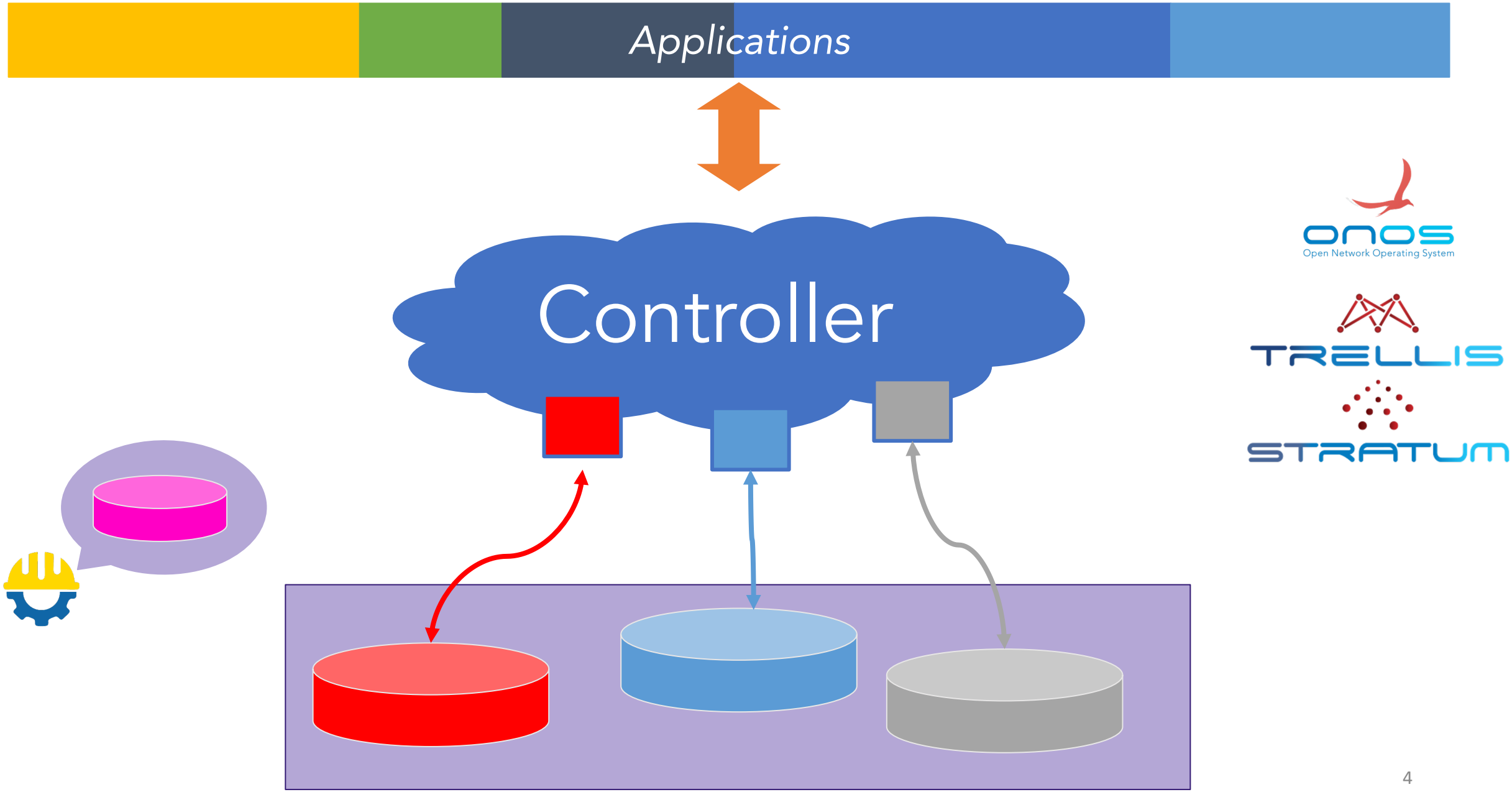
Eric Campbell
Cornell University

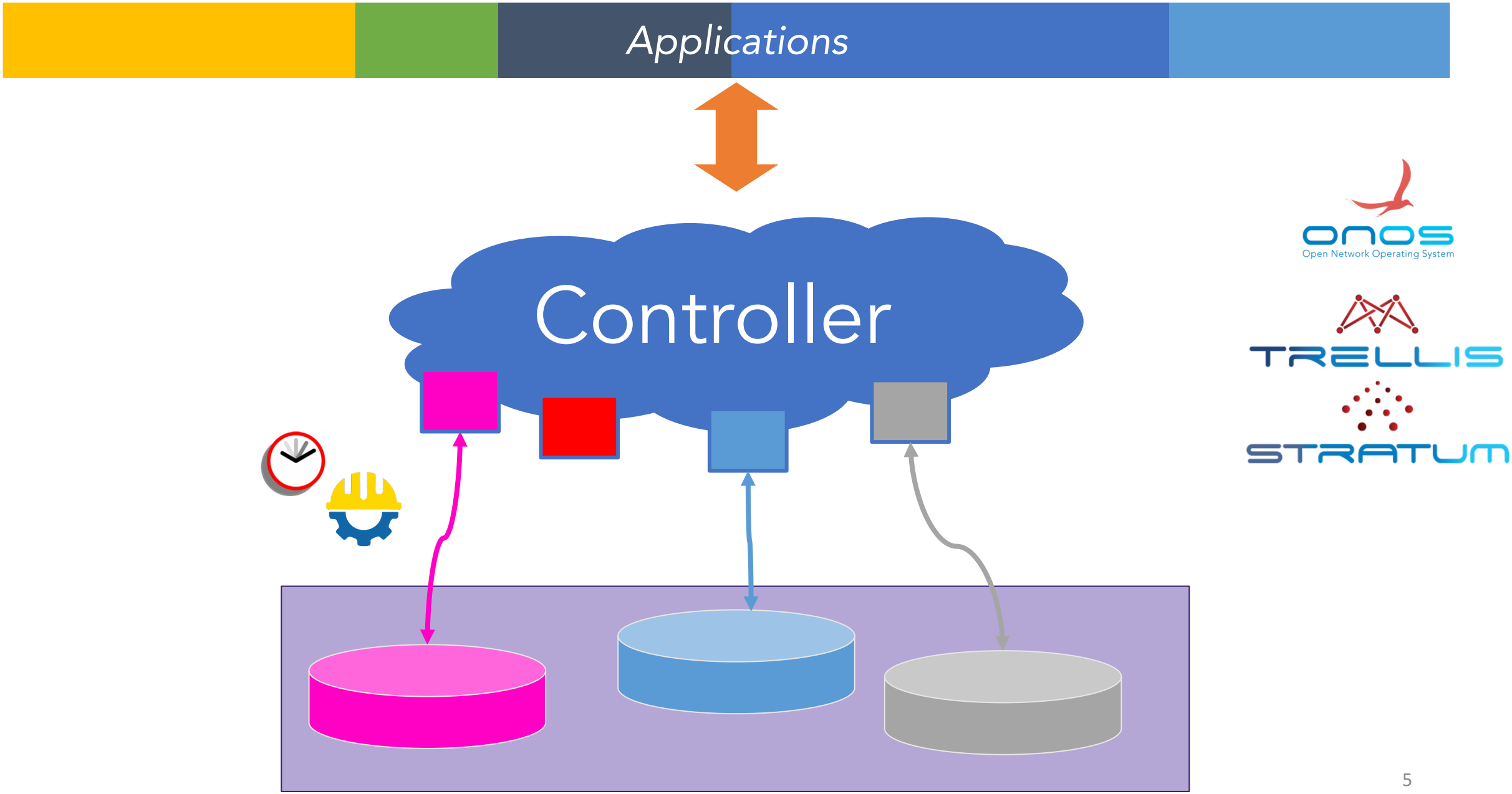

Applications
High level
Abstractions
Controller
Decoupled
Control & Data
SB API for Forwarding
Devices

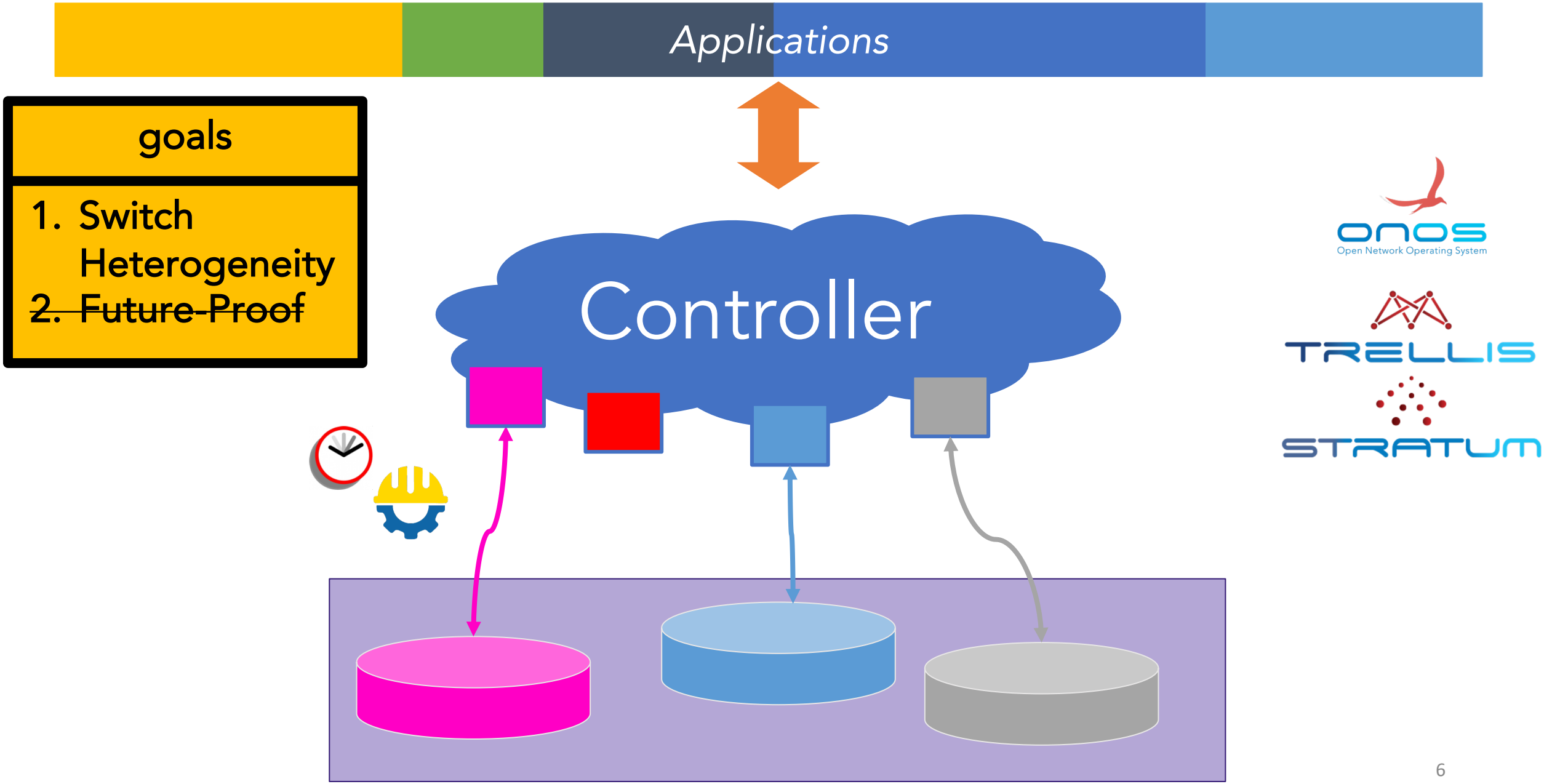


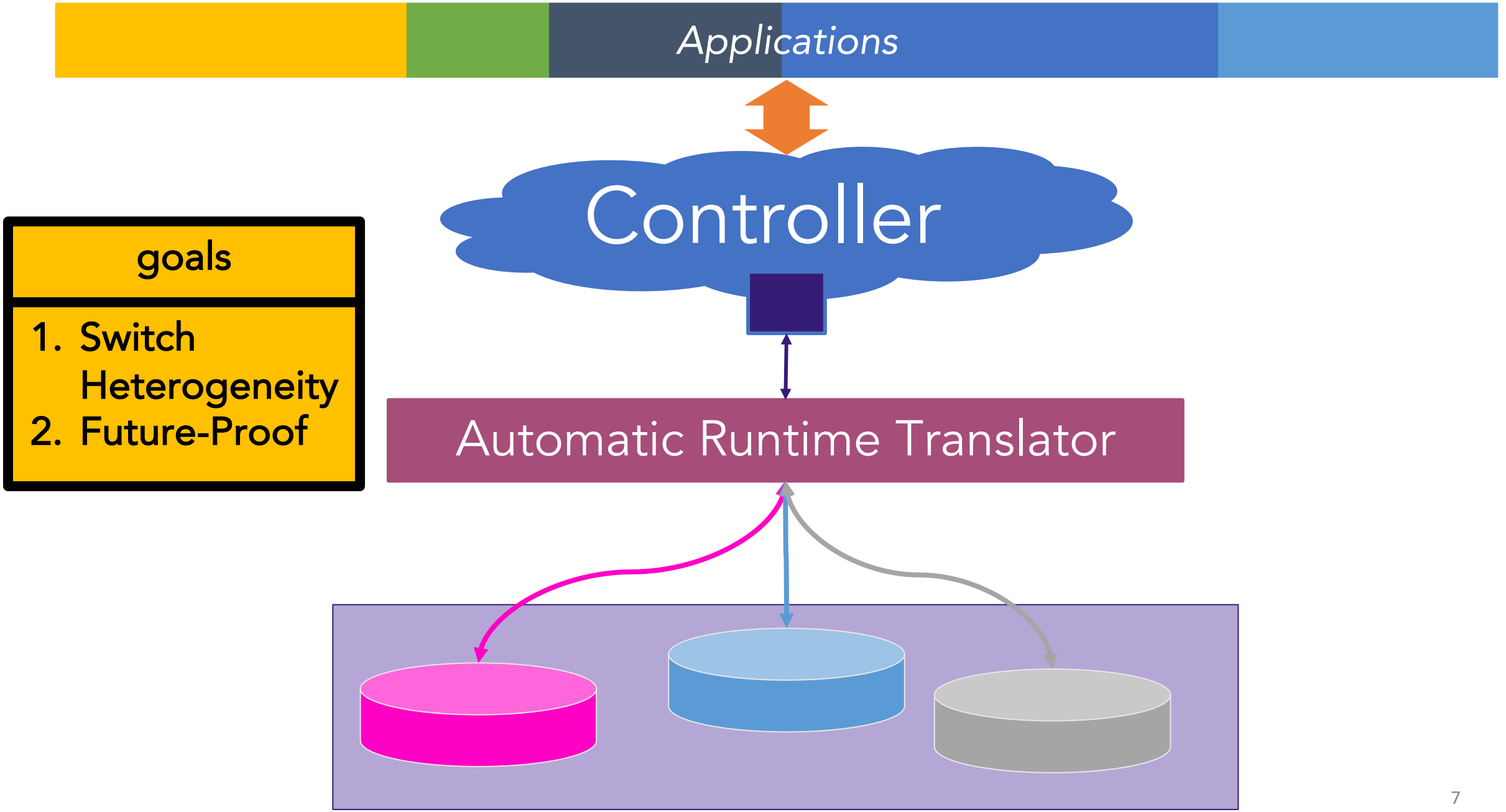
goals

1. Switch Heterogeneity
2. Future-Proof


Applications
Controller


Applications
Controller

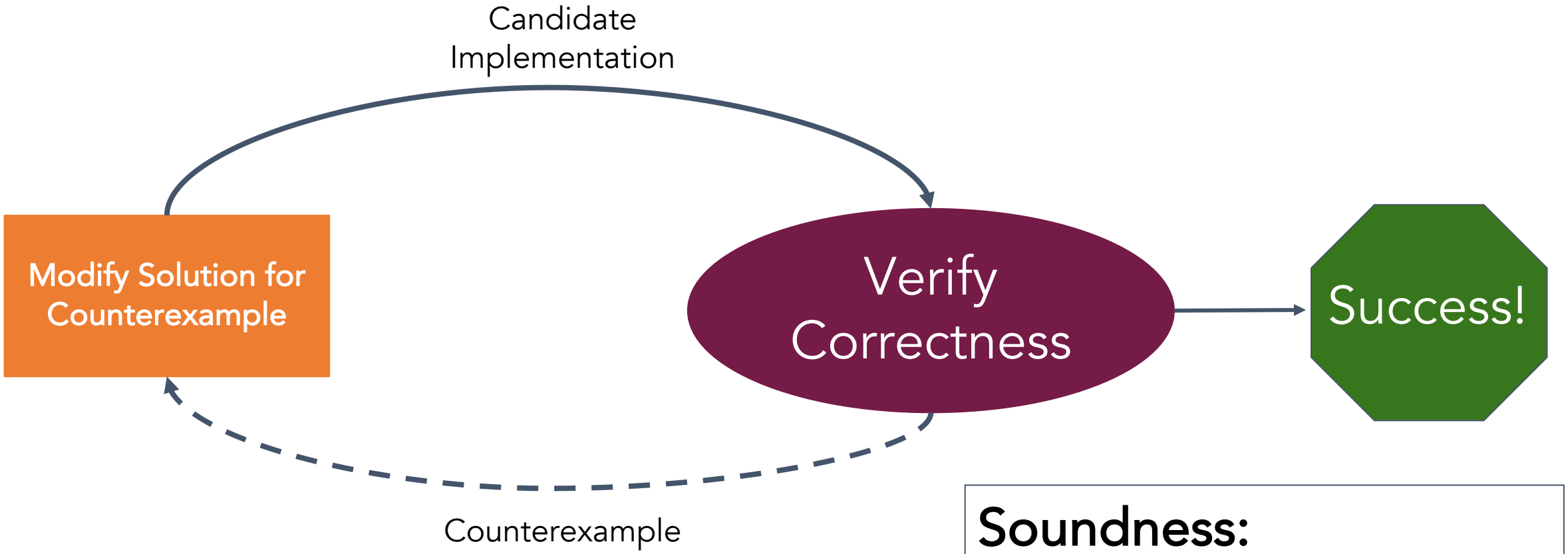

Applications
goals
1. Switch Heterogeneity
2. Future-Proof
Controller
onos
Open Network Operating System
TRELLIS
STRATUM


Applications
Controller
goals
1. Switch Heterogeneity
2. Future-Proof
Automatic Runtime Translator

The Secret Sauce?

Program Synthesis

How does Synthesis* Work?



Soundness:

Every solution is correct

Completeness:

A solution is found when one exists

*counter-example guided inductive synthesis

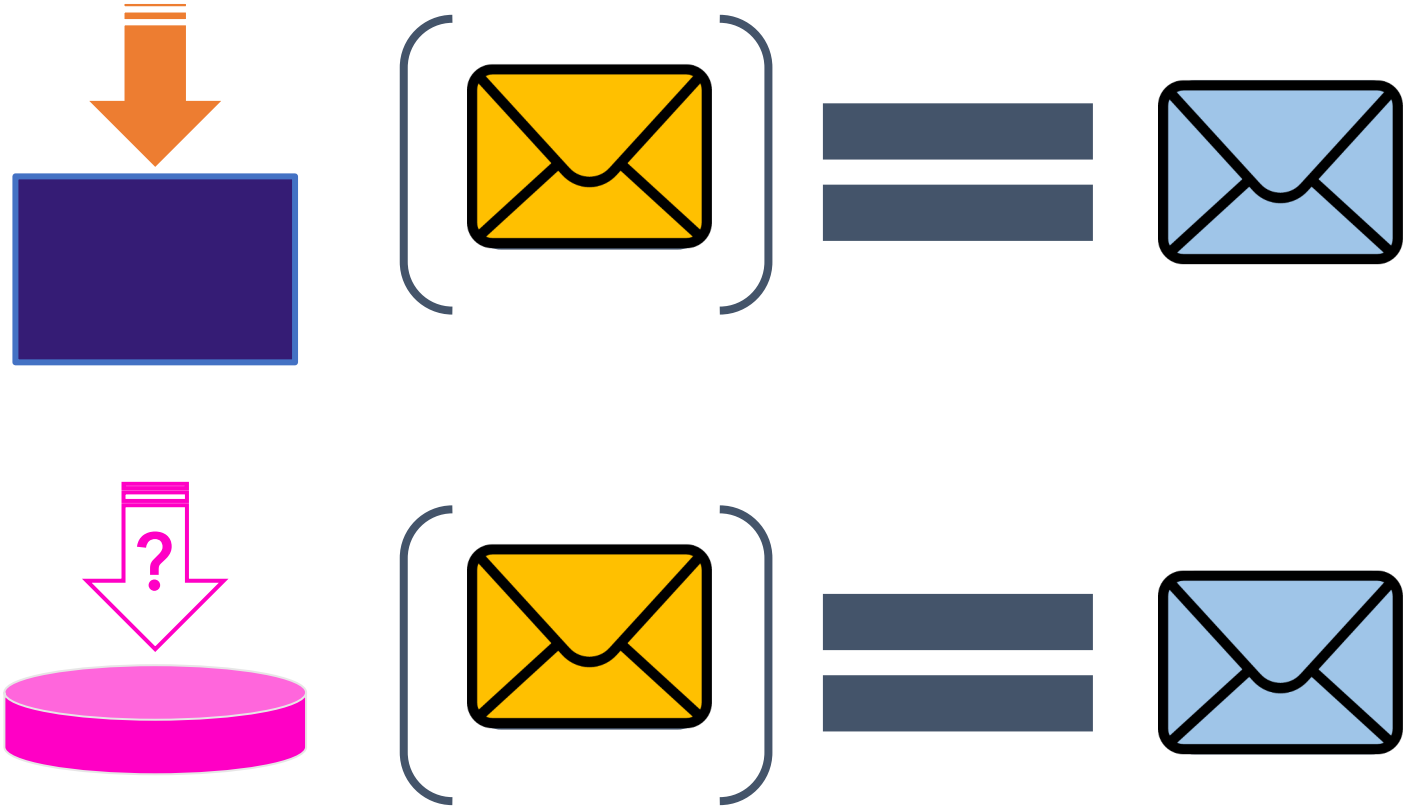
Correctness: Program Equivalence

Checked using a Theorem Prover (e.g. z3)

Extending With Counterexample

Abstract Runtime API Call

Abstract Switch



Physical Runtime API Call

Physical Switch

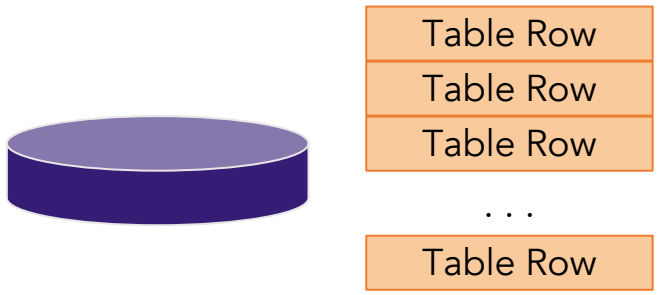
Heuristic-Guided Search for 

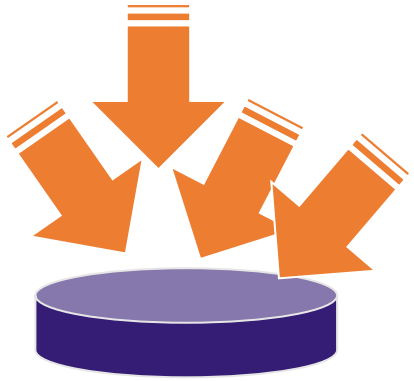
Preliminary Experiments

- ONOS Switch Reboot
  - Abstract: fabric.p4
  - Physical: broadcom switch
  - ~40k insertions
  - ONOS takes ~15mins,
    Avenir takes ~32mins
- Good performance is due to Incrementality!

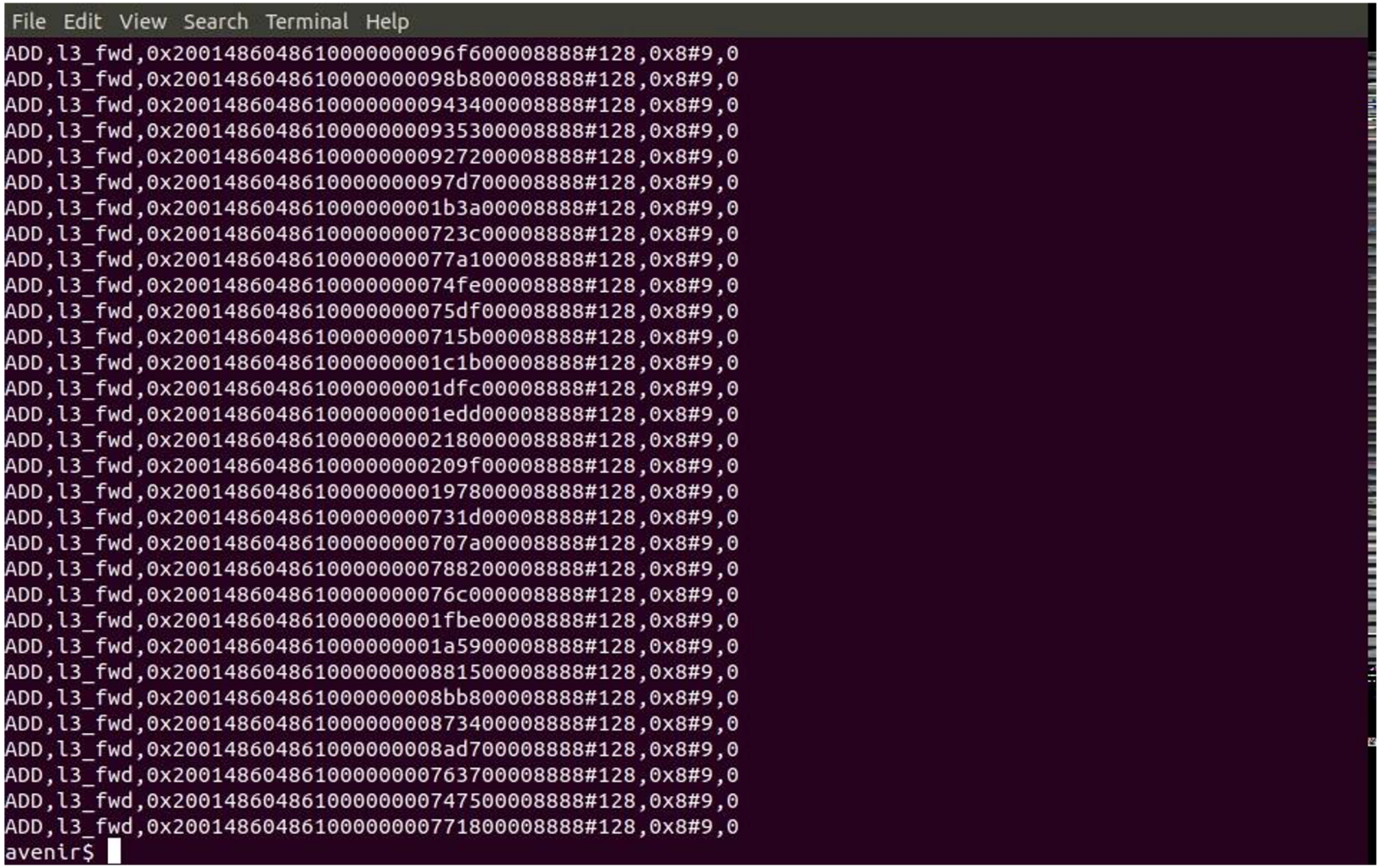
Caveats

- Parsers assumed equivalent
- Externs manually modelled
- Hashes are tricky



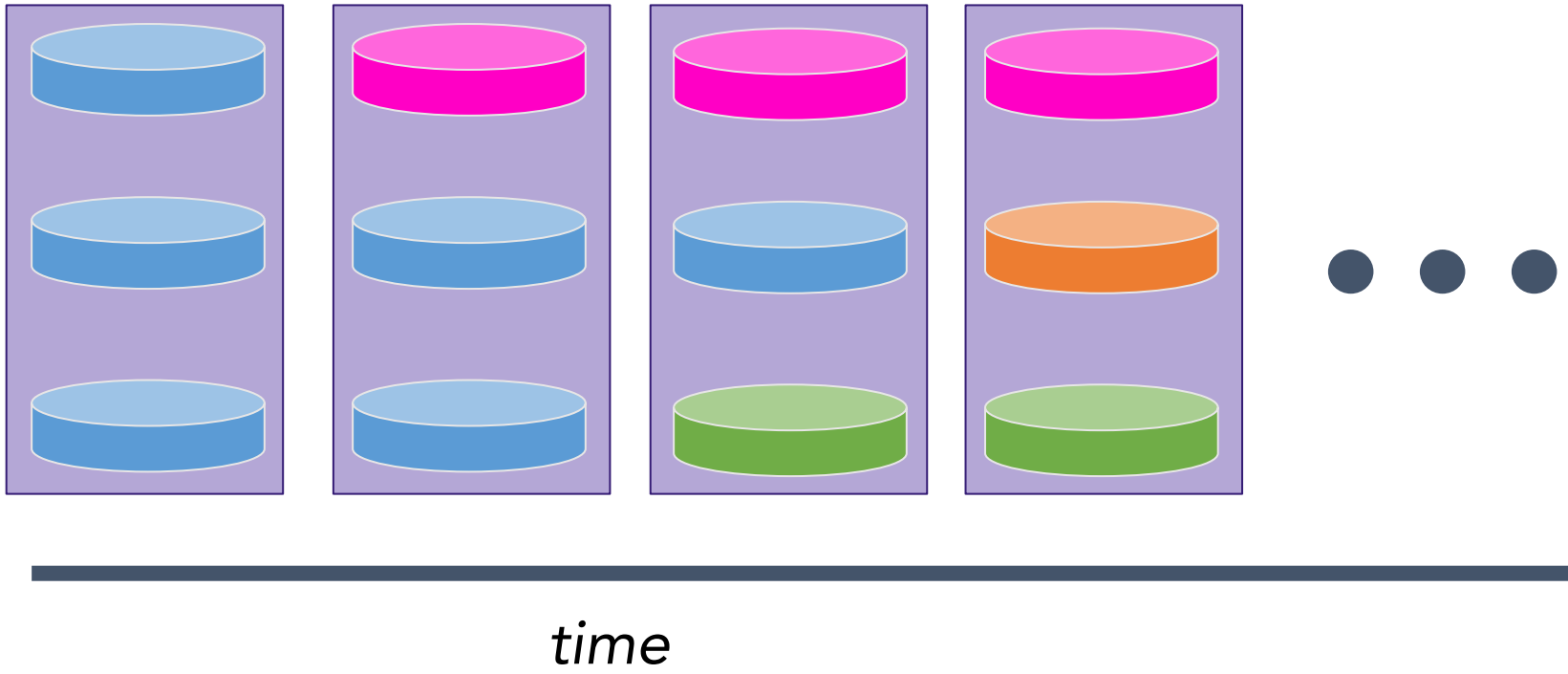


Switch Reboot



Automatic Runtime Translator

1. Easy Switch Heterogeneity
2. Future-Proof



Thank You

email: ehc86@cornell.edu

Special thanks to:
Priya Srikumar, Nate Foster, Hossein Hojjat,
Bill Hallahan, Ruzica Piskac, Robert Soulé,
Brian O'Connor, Carmelo Cascone,
Jed Liu, Andy Fingerhut, Vignesh Ramamurthy

Work supported by Infosys & the NSF

Sponsored By

Google intel STORDIS The Open Networking Expert

P4 to userspace BPF: Extending the userspace packet processing pipeline at runtime

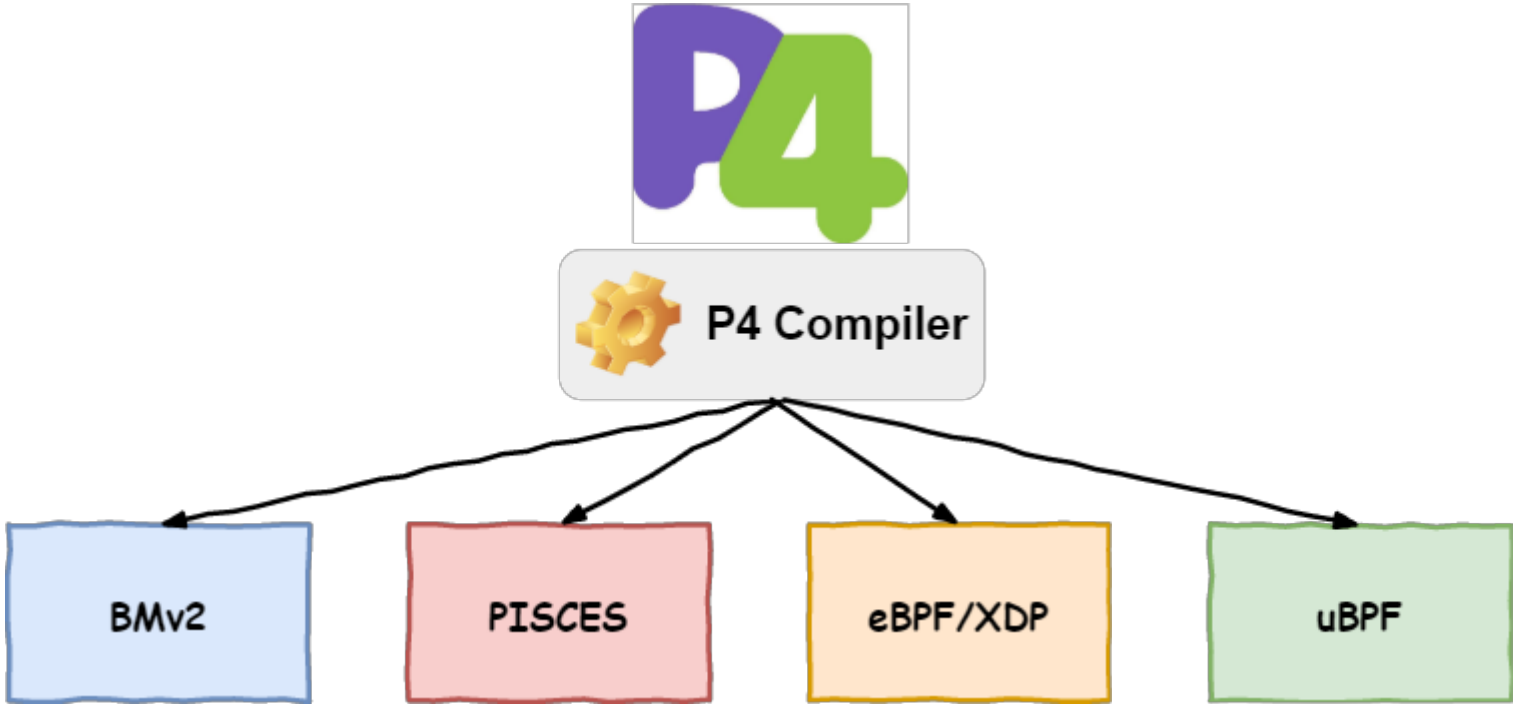
Tomasz Osiński

Team members:
Mateusz Kossakowski
Halina Tarasiuk

Orange Labs
Warsaw University of Technology

Context

- More and more P4 targets are emerging…
- Software targets have their role..

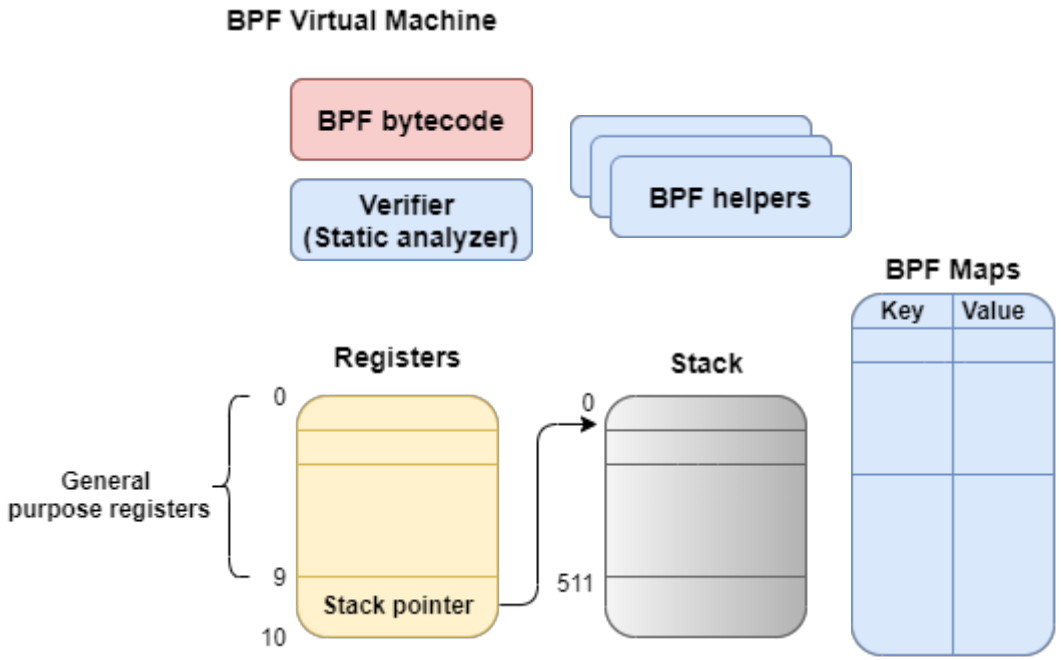


Motivation

- NFV deployments requires high-performance
- OVS-DPDK, VPP, OVS-AF_XDP…
  - all these targets runs in userspace
  - fixed packet processing pipeline
  - not extensible enough
- We need some safe runtime extensibility mechanism…
- **… uBPF!**

Introduction to Userspace BPF VM

- Userspace BPF Virtual Machine*
  - JIT compilation to x86 architecture
  - Static analyzer (verifier)
  - BPF maps
  - Provides set of helpers (e.g. *ubpf_update_map()*)
- eBPF vs. uBPF:
  - Kernel vs. Userspace
  - GPL vs. Apache 2.0 license
  - uBPF implements a „thin” VM (e.g. no tail calls)

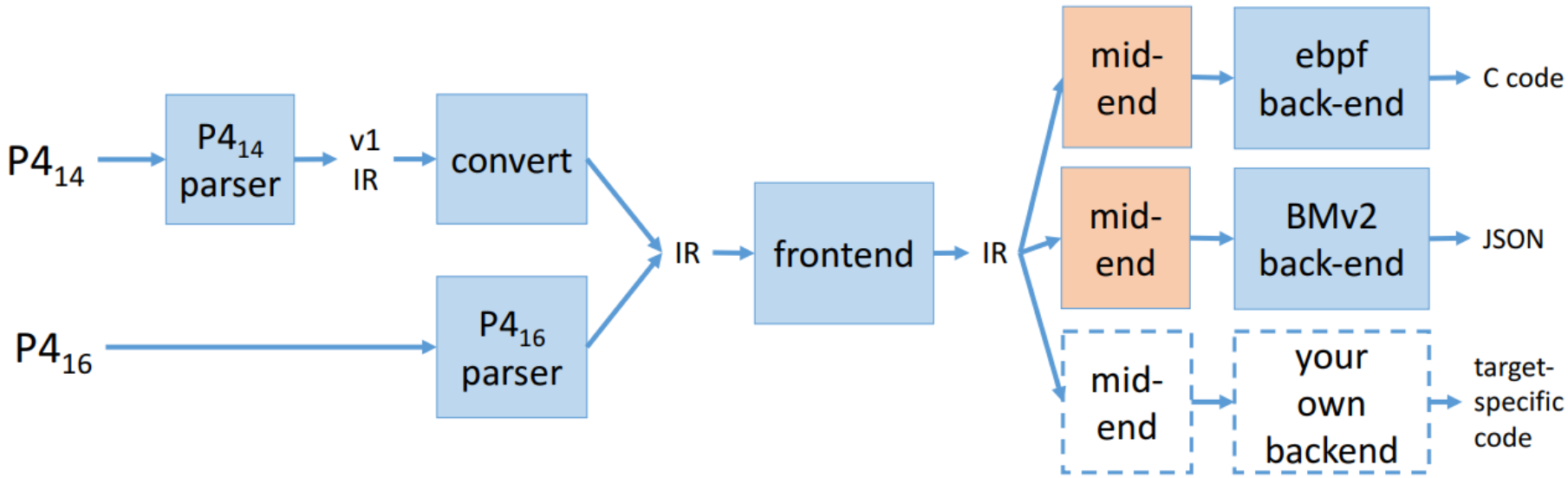


* https://github.com/iovisor/ubpf

P4 to userspace BPF

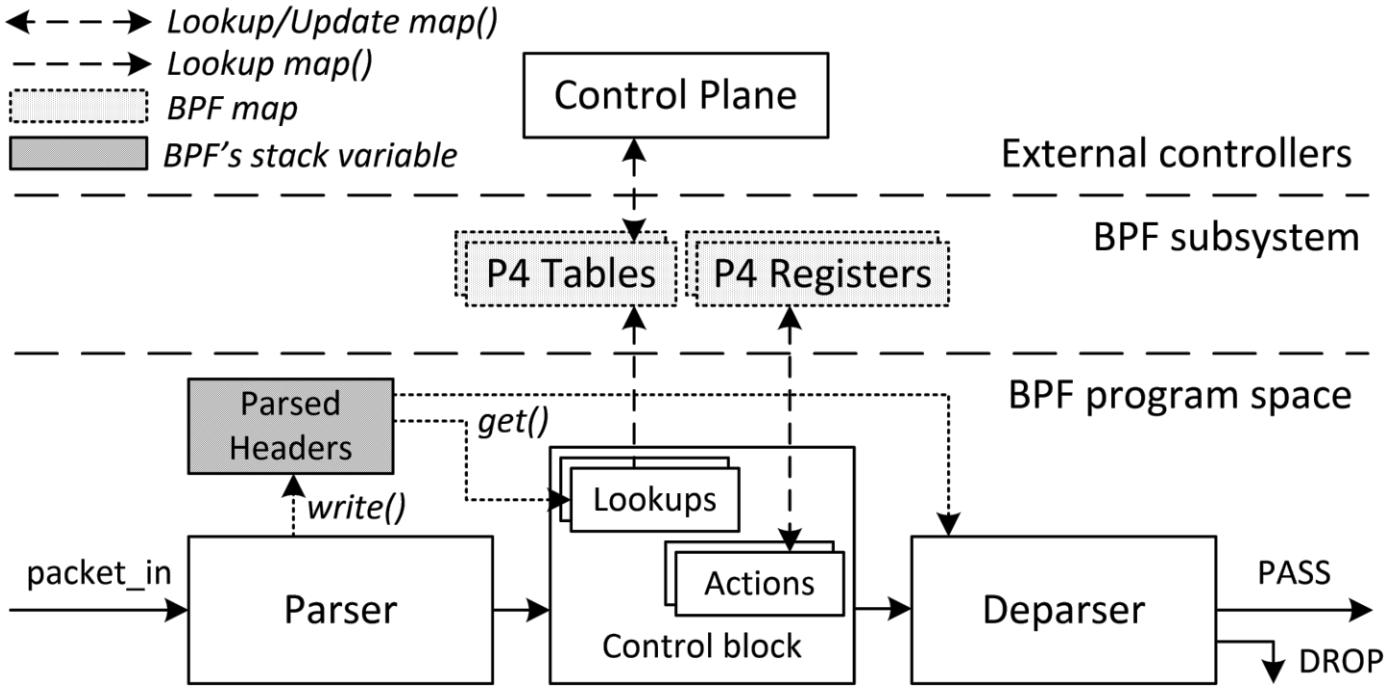
P4c-uBPF – design principles & workflow

- **The userspace BPF backend for the P4 compiler**
  - The design is strongly based on eBPF/XDP backend
  - BPF maps are used to implement P4 tables, registers, etc.
- **Generates BPF bytecode: $P4_{16}$ -> C -> uBPF**



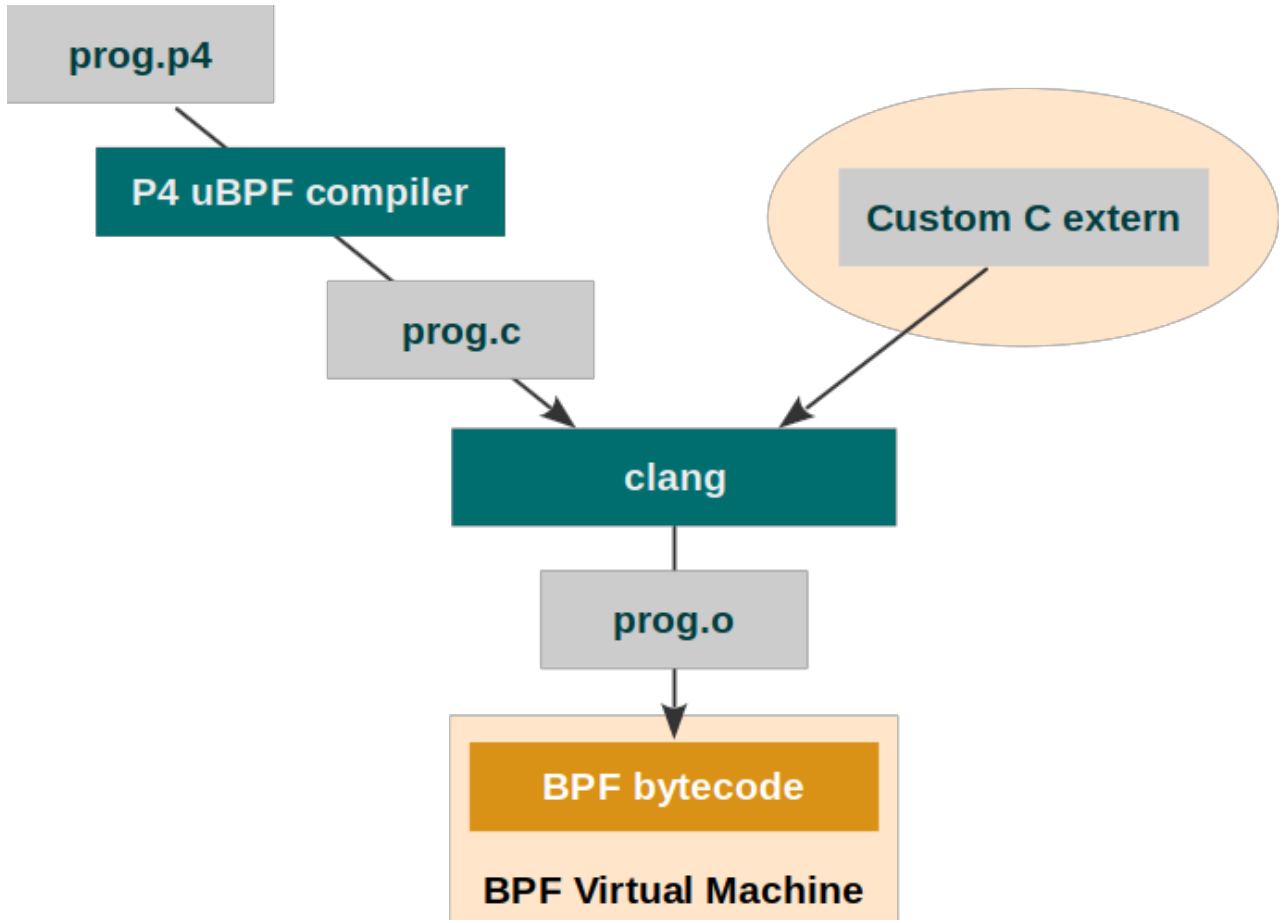
The ubpf_model.p4 architecture model

- **Main blocks:**
  - Parser
  - Control block
  - Deparser
- **Supported features:**
  - Registers
  - Hash functions
  - Checksum computation



Custom C extern functions support

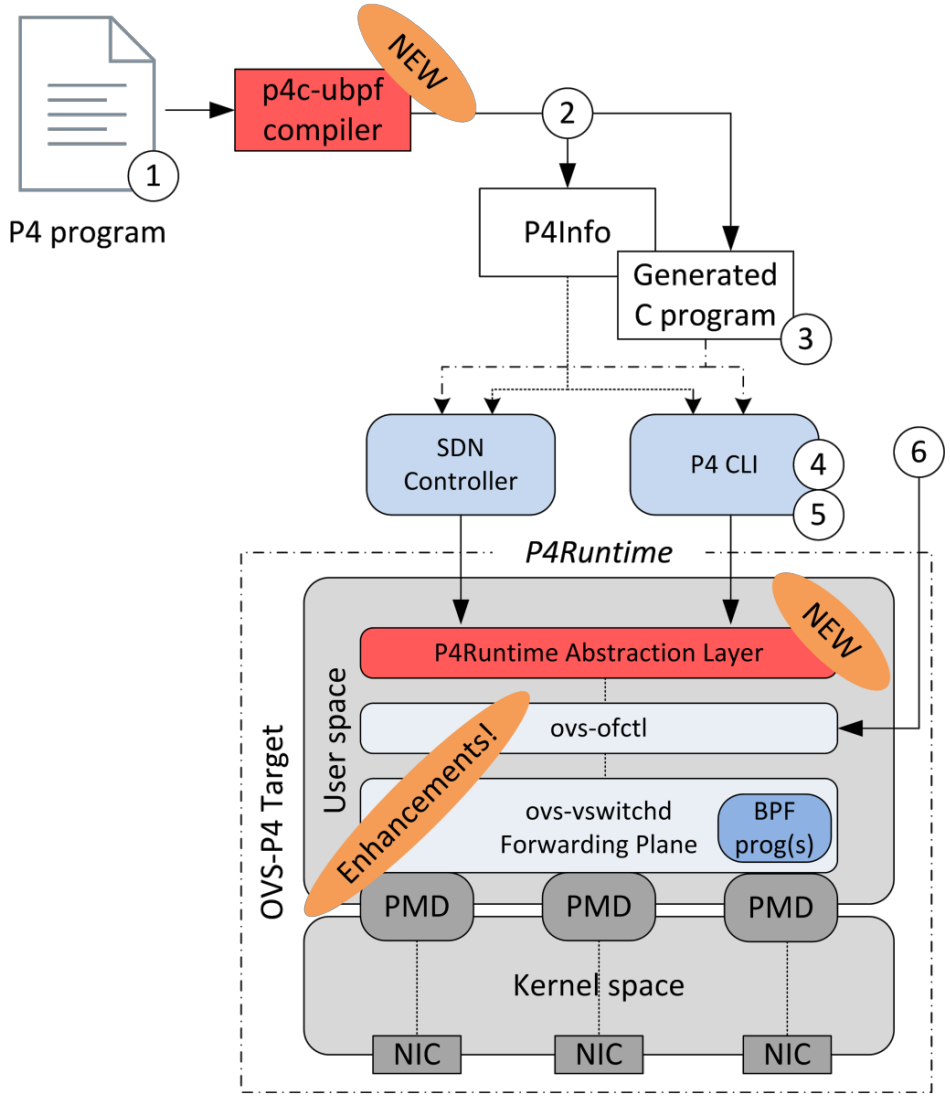
- **p4c-ubpf enables custom C externs:**
  - User-defined C functions invoked from the P4 program!
  - Must be compatible with BPF VM
- **New exciting use cases!**
  - Stateful externs using BPF maps



P4 to userspace BPF in action

P4rt-OVS: Extending the OVS pipeline at runtime

- **Programming workflow:**
  - Write „**prog.p4**” (use **ubpf_model.p4**)
  - **p4c-ubpf** –o prog.c **prog.p4**
  - **clang** -O2 -target **bpf** -c prog.c -o **prog.o**
  - **ovs-ofctl load-bpf-prog** br0 *<prog-id>* **prog.o**
  - **ovs-ofctl** add-flow br0 actions=**prog**:*<prog-id>*
- **Use cases:**
  - GPRS Tunnelling Protocol (GTP)
  - Point-to-Point Protocol over Ethernet (PPPoE)
  - In-Band Network Telemetry (INT)*
  - Stateful firewall



** Ongoing work*

Comparison with other BPF-related backends

- uBPF backend provides support for stateful packet processing!
- No counters!

Feature	p4c-ebpf	p4c-xdp	p4c-ubpf
Packet filtering	YES	YES	YES
Packet modifications	NO	YES	YES
Tunneling	NO	YES	YES
Packet forwarding	NO	YES	YES
Registers	NO	NO	**YES**
Counters	YES	YES	**NO**
Checksum computation	NO	YES	YES
Custom C externs	**YES**	Not tested	**YES**

Summary

- **p4c-ubpf** – the new back-end for the P4 compiler!
- **Open problem:**
  - uBPF VM is not standardized, different flavors
- **Future work:**
  - Enhancements to P4 compiler (e.g. counters)
  - Performance optimizations
  - Design and implement a new P4Runtime switch based on OVS AF_XDP/DPDK
- **Open questions:**
  - How far are we from industry-grade and production-ready implementation of P4 software switch?

Sponsored By

Google intel STORDIS The Open Networking Expert

Thank You

osinstom@gmail.com / tomasz.osinski2@orange.com

https://github.com/p4lang/p4c/tree/master/backends/ubpf
https://github.com/Orange-OpenSource/p4rt-ovs
https://www.openvswitch.org/support/ovscon2019/#4.3F