cisco

Exposing Data Plane Programmability on Turn-Key Network Devices

Opportunities, challenges, and options

Mario Baldi

Programmable Switch Deployment Flavors

Whitebox

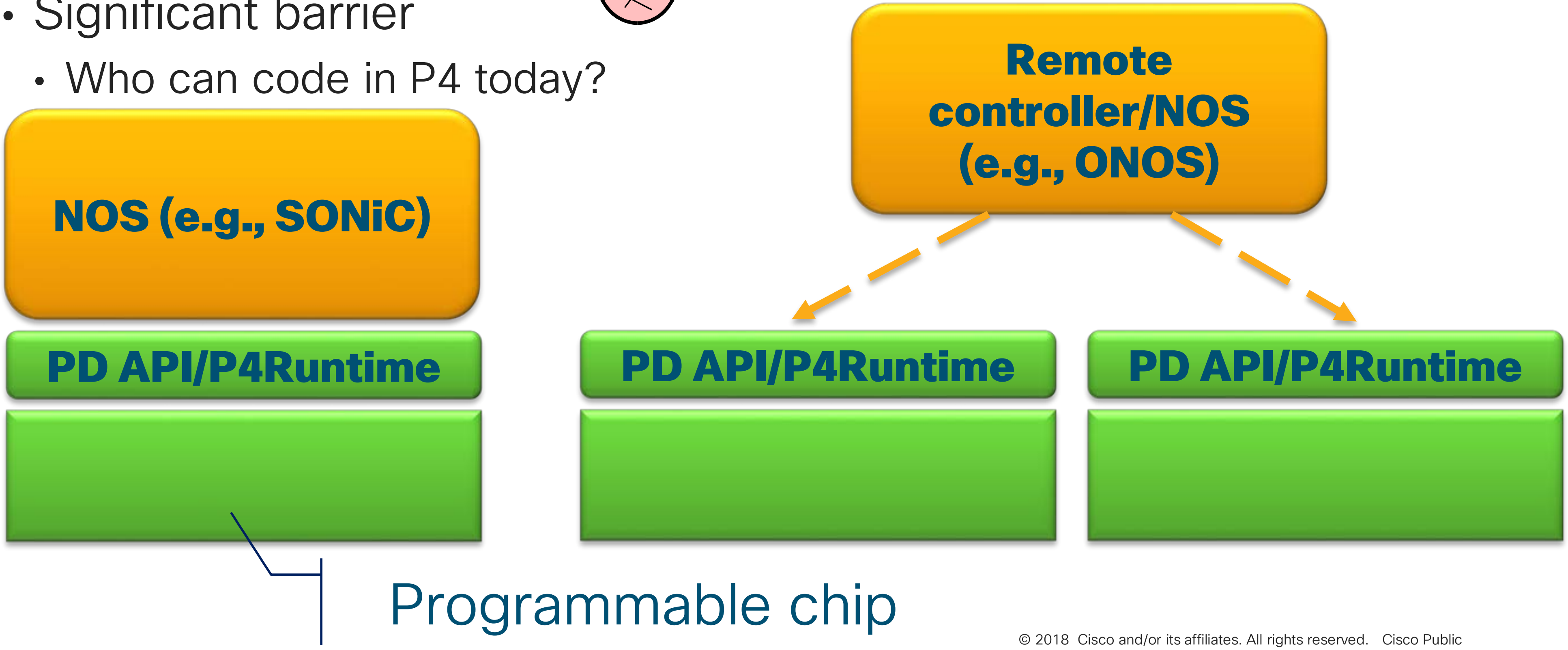
Turn-key

Hybrid

Whitebox Deployment

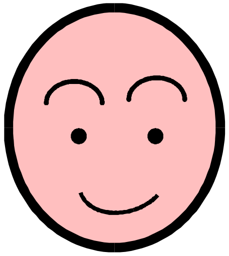
- Maximum flexibility
- Maximum disruption/risk
- Significant barrier
  - Who can code in P4 today?

🟦 Platform vendor (Cisco)
🟩 Chip vendor (Barefoot)
🟧 Customer/open source



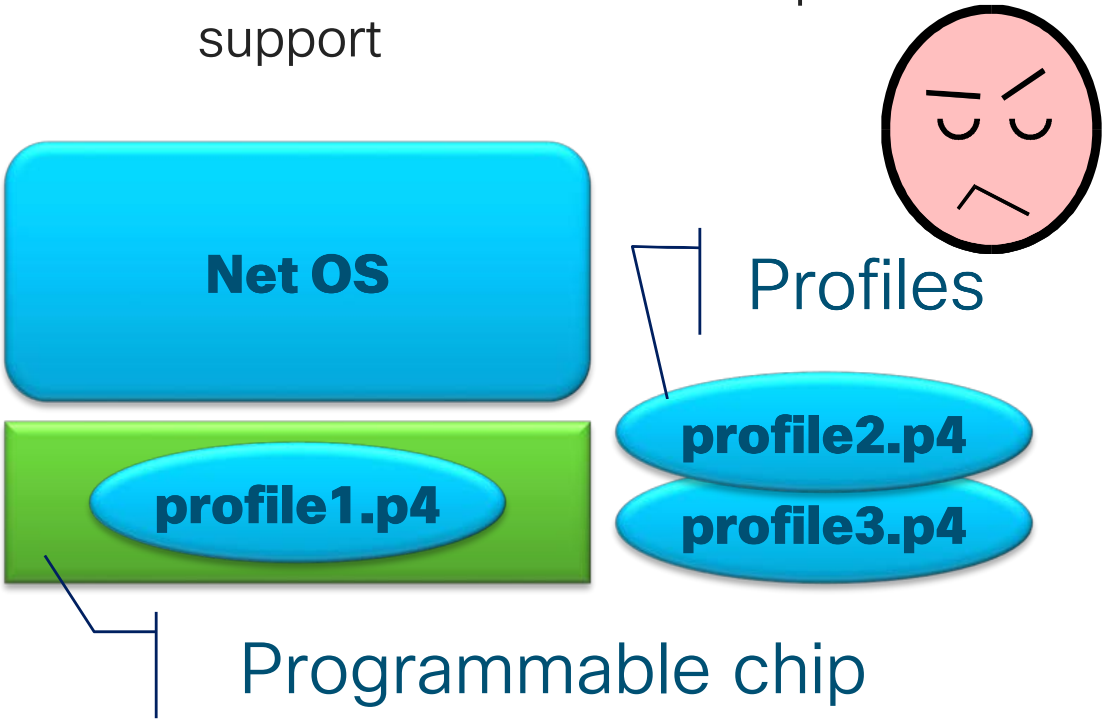
Turn-key Deployment

- Deployment as usual
  - Familiar features and interfaces
- Resource optimization
- Future proof
- Feature agility
- Streaming telemetry



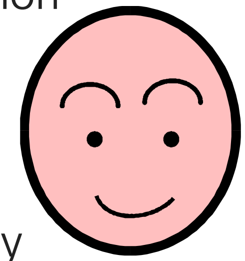
🟦 Platform vendor (Cisco)
🟩 Chip vendor (Barefoot)
🟧 Customer/open source

- No flexibility
  - No custom features and protocol support

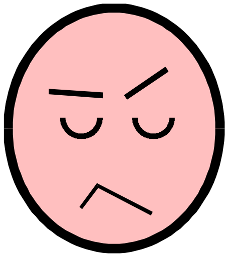


Open Platform

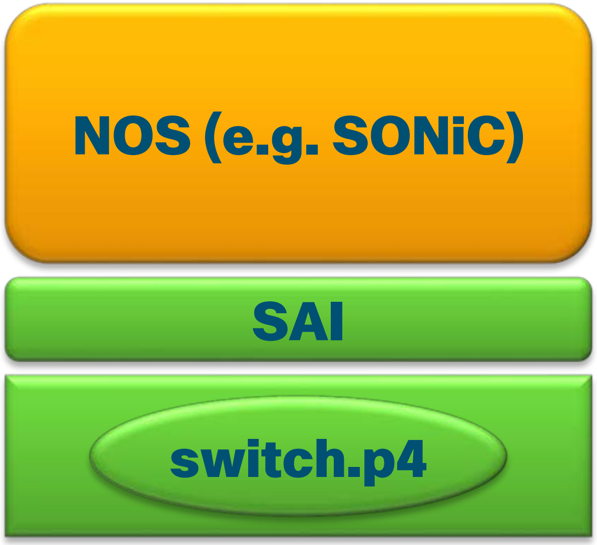
- Deployment as usual
  - Familiar features and interfaces
- Resource optimization
- Future proof
- Feature agility
- Streaming telemetry



- No flexibility
  - No custom features and protocol support



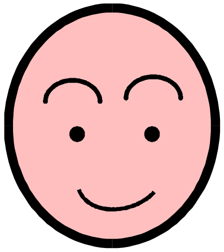
Same as Turn-key



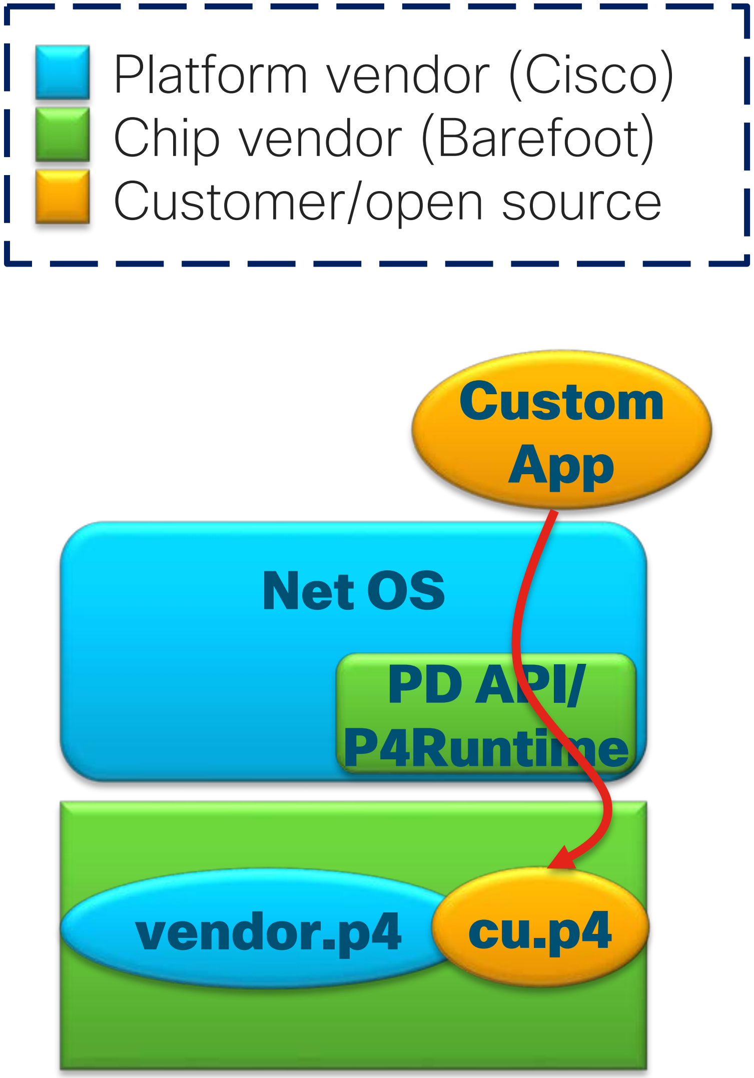
🟩 Chip vendor (Barefoot)
🟧 Customer/open source

Hybrid Deployment

- Best of breed
- Deployment as usual
  - Familiar features and interfaces
- And also flexibility



Without the added risk!



Hybrid Deployment Challenges

Do not break what works

- Vendor data plane code is well tested
- … and we don’t want to need regression testing

Don’t want to show, don’t want to see

- Vendor code and custom code may be confidential
- Not practical to familiarize with a lot of vendor code to just write a few lines

Resource availability

- Still “limited” on current chips

Data/control plane dependence

- Net OS should keep working
- Net OS should not be aware of custom data plane functions

In a nutshell

P4 and its ecosystem were not designed for *incremental programming*

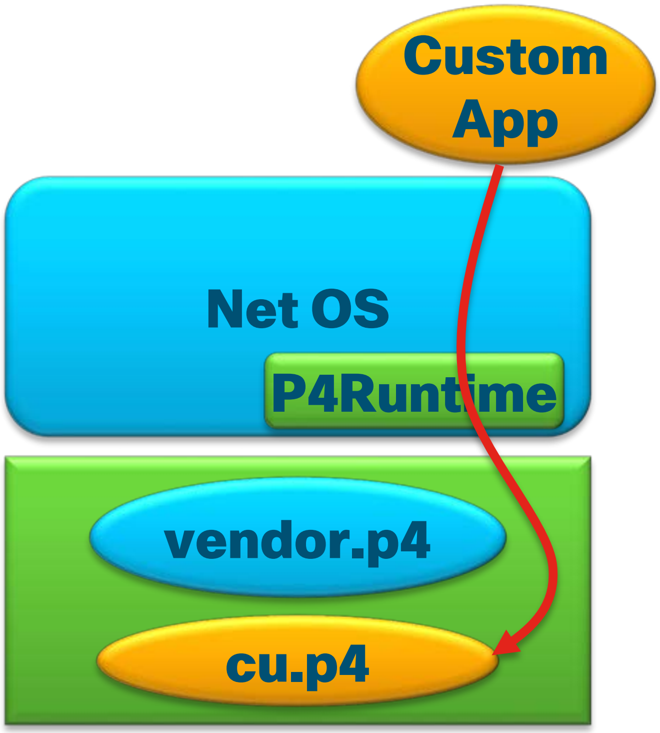
Single programmer

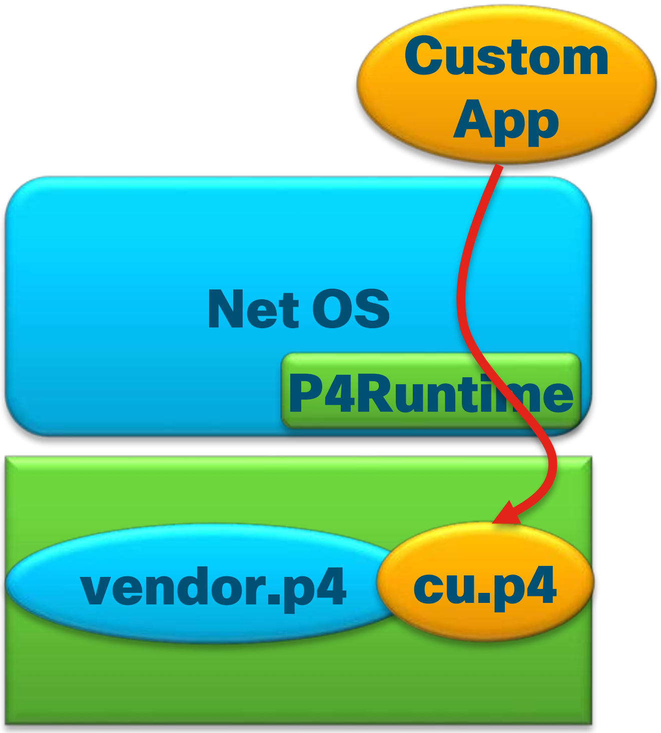
Single source code

Single control plane

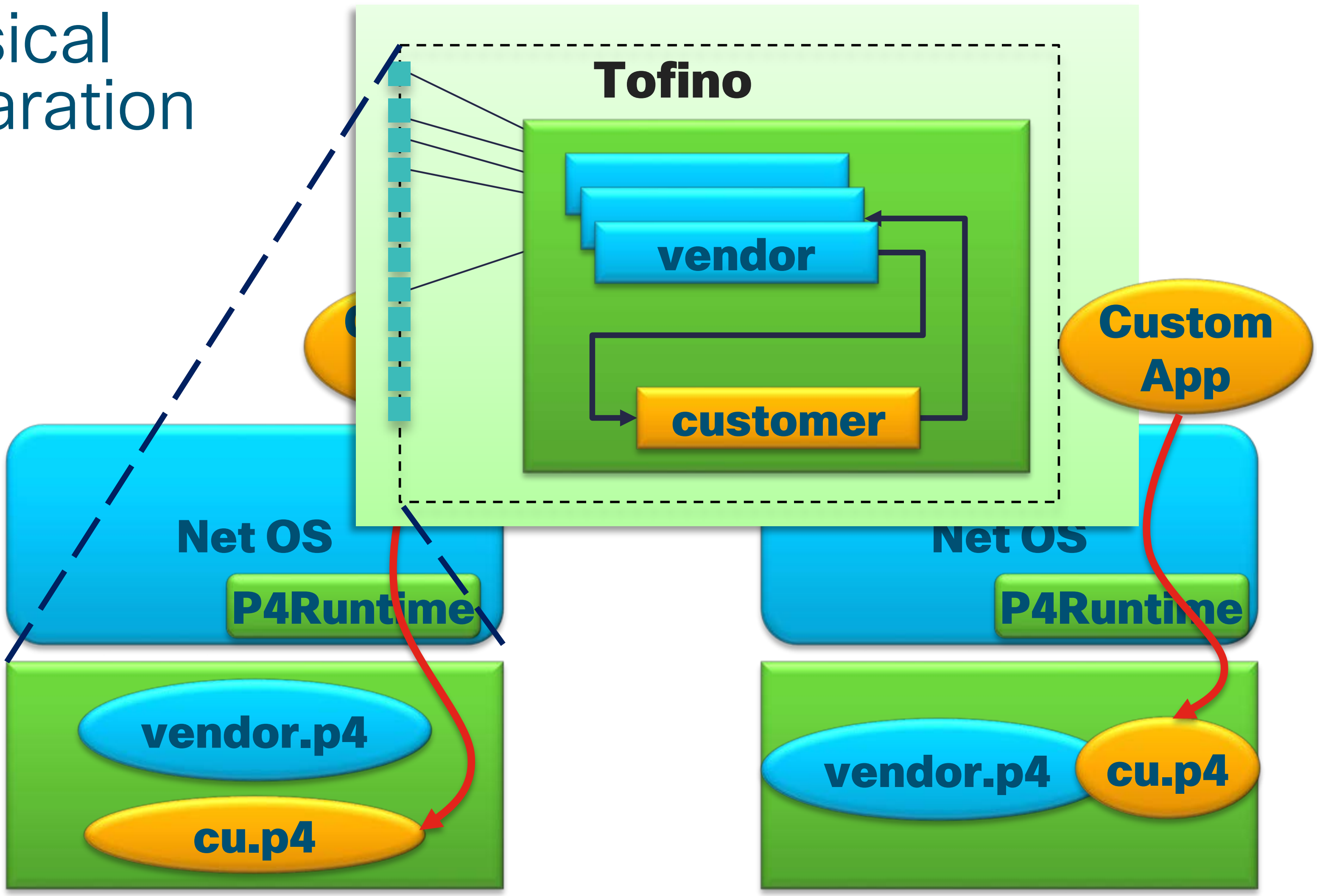
Possible Options

🟦 Platform vendor (Cisco)
🟩 Chip vendor (Barefoot)
🟧 Customer/open source

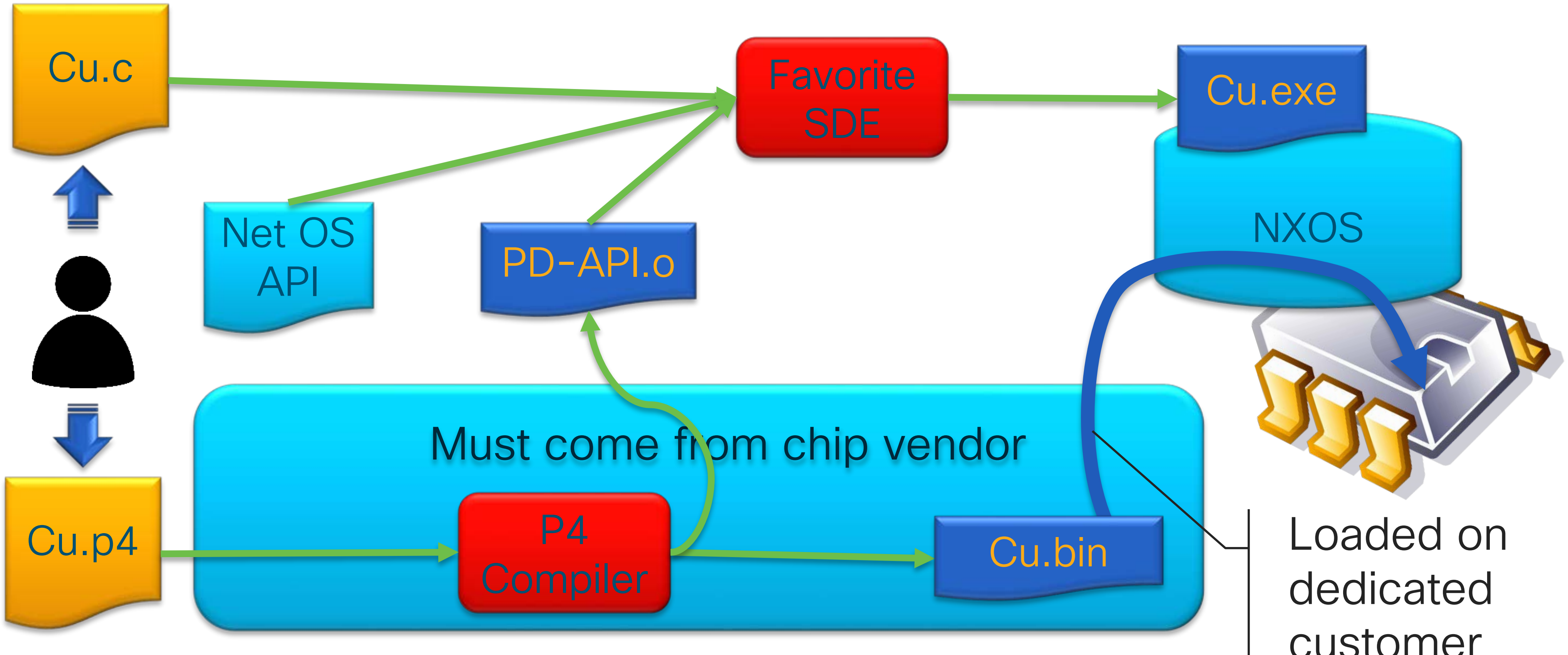




Physical Separation



Incremental Programming Workflow

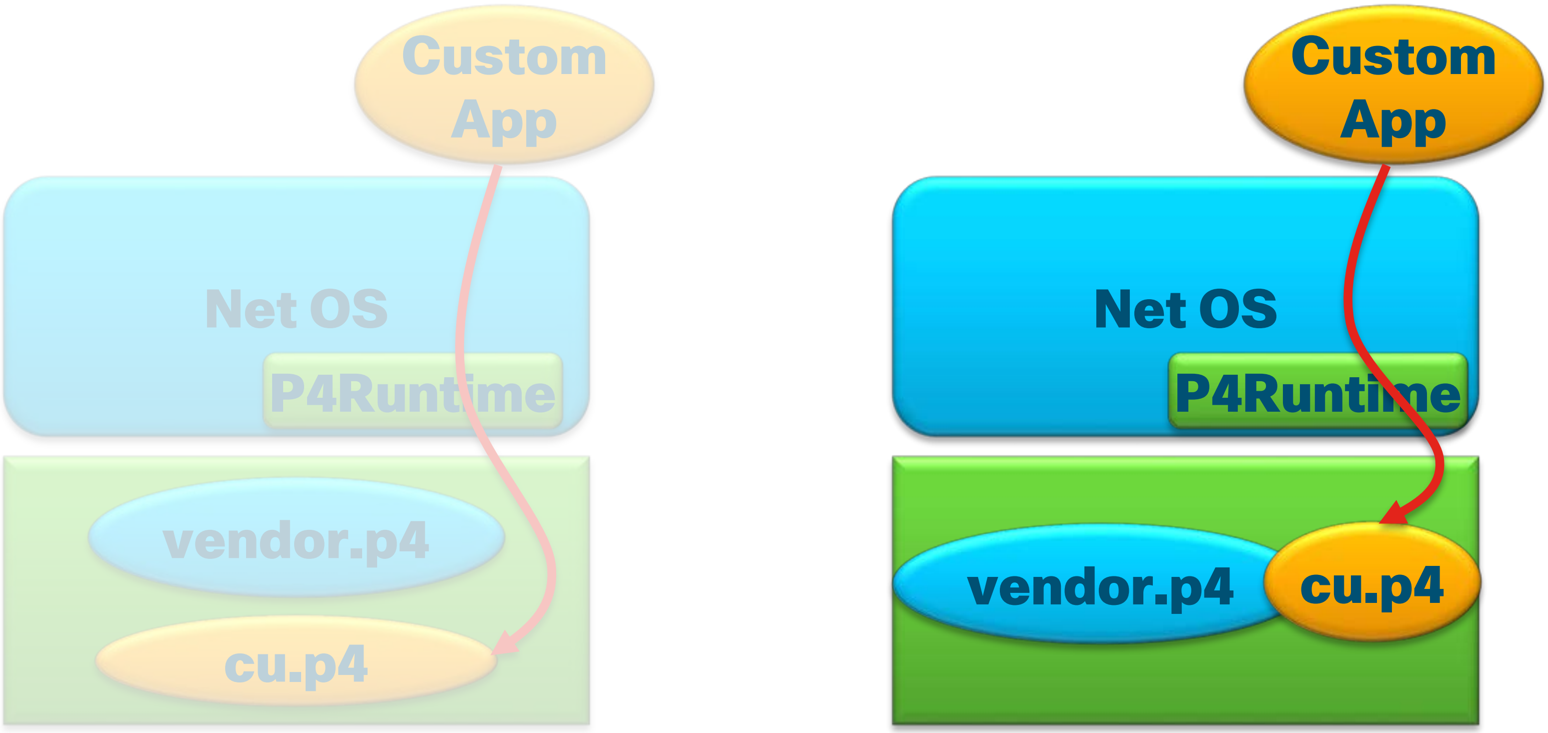


Loaded on dedicated customer pipeline

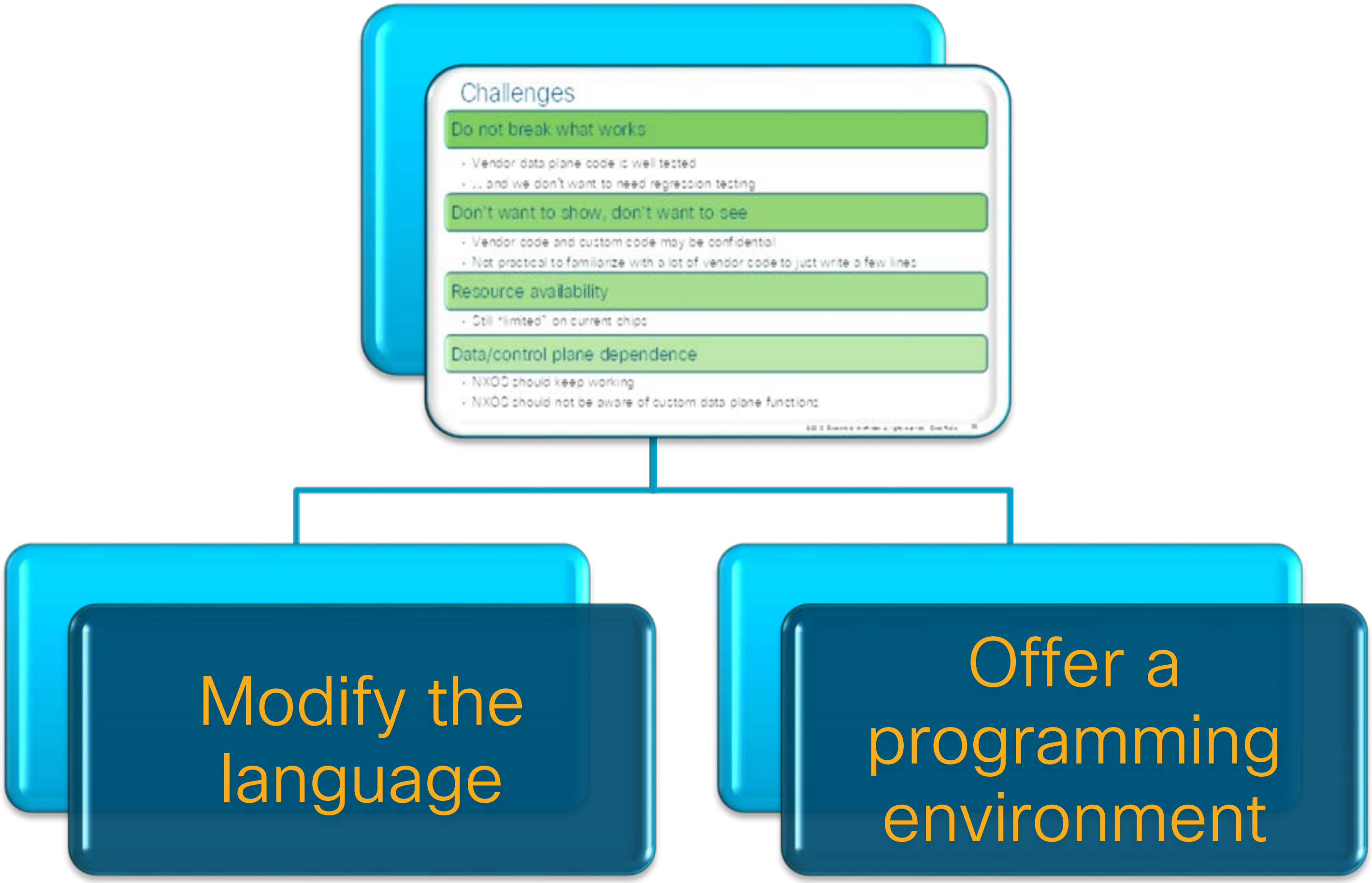
Other pipelines pre-loaded with vendor.bin

Software Solution

🟦 Platform vendor (Cisco)
🟩 Chip vendor (Barefoot)
🟧 Customer/open source



What about the challenges we mentioned earlier?



Challenges

Do not break what works

- Vendor data plane code is well tested
- … and we don't want to need regression testing

Don't want to show, don't want to see

- Vendor code and custom code may be confidential
- Not practical to familiarize with a lot of vendor code to just write a few lines

Resource availability

- Still "limited" on current chips

Data/control plane dependence

- NXOS should keep working
- NXOS should not be aware of custom data plane functions

Language Design Working Group

Modularity can help with incremental programming

Sub-working group to introduce modularity in P4

- March 2018

Started focusing on polymorphism

- Generic data type
- Generic function type

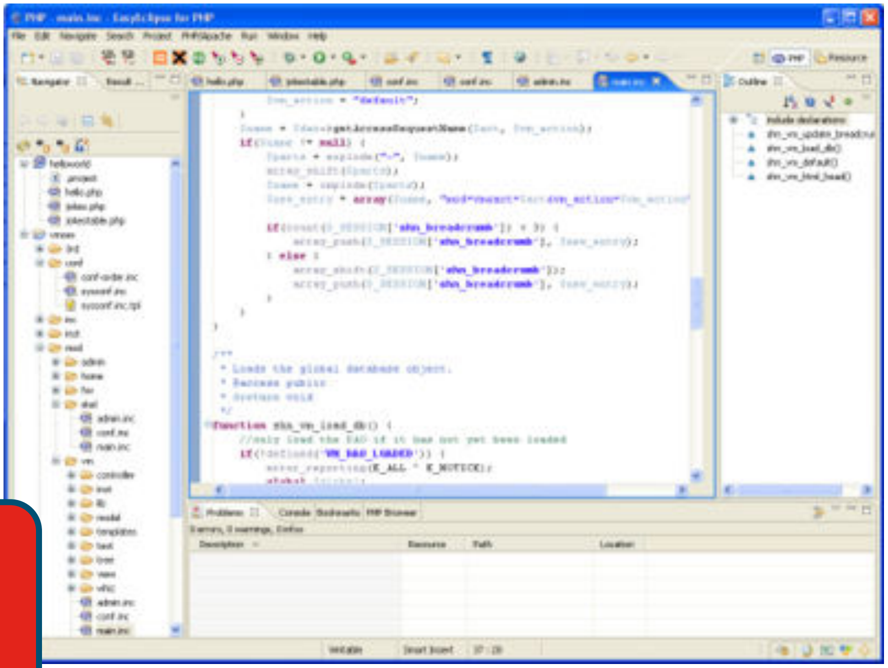
Intent to focus on modularity for incremental programming

daPIPE

Data Plane Incremental Programming Environment

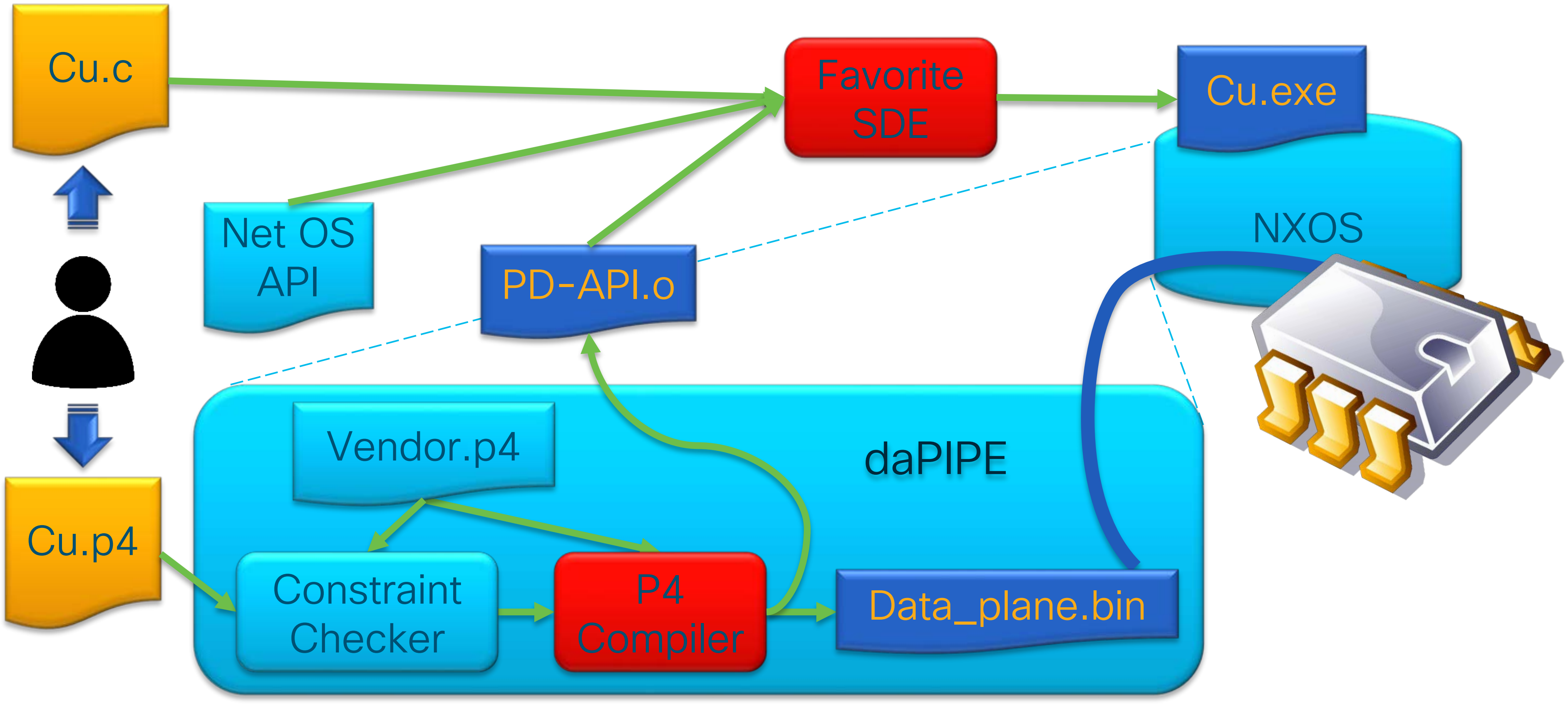
Identify constraints on new code

Impose those constraints on the program



Support developers and streamline their task (while enforcing needed constraints)

Customer Programming Workflow



Thank you

Any questions?