Towards a Network Operating System

Victor Lopez

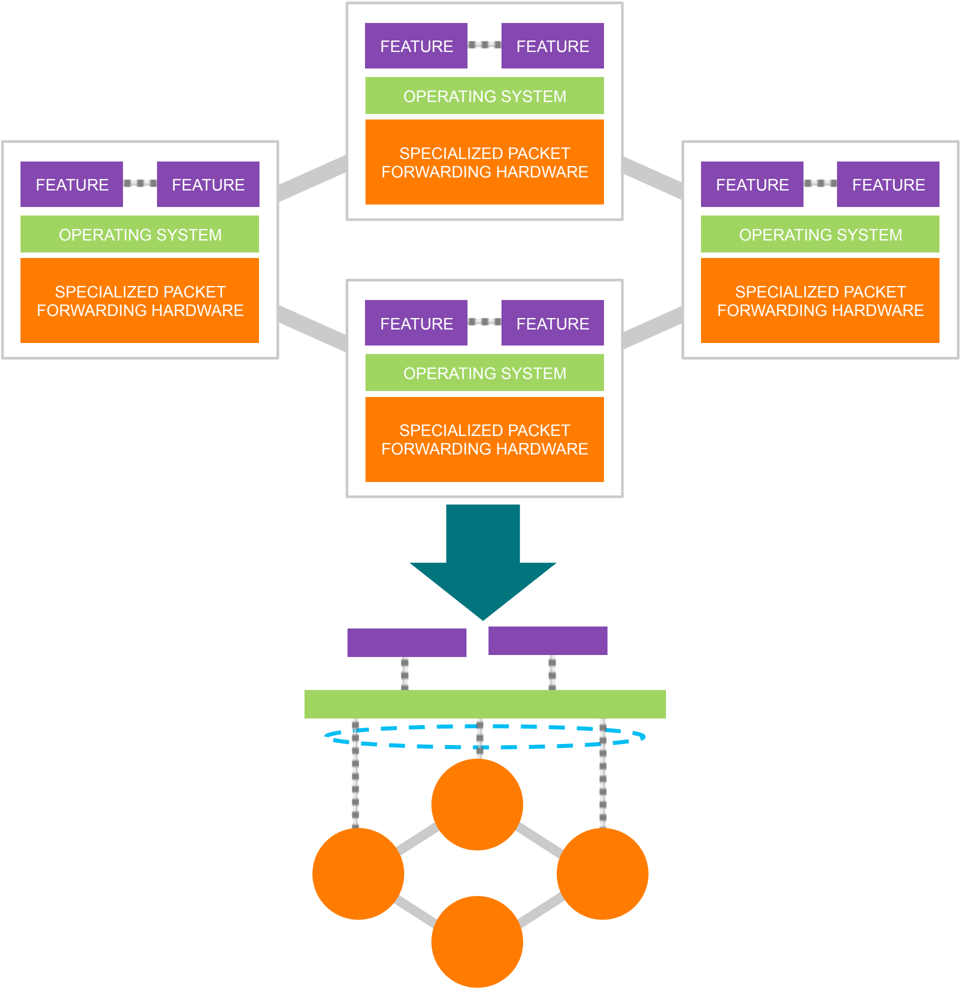
Telefonica

Shifting Paradigms

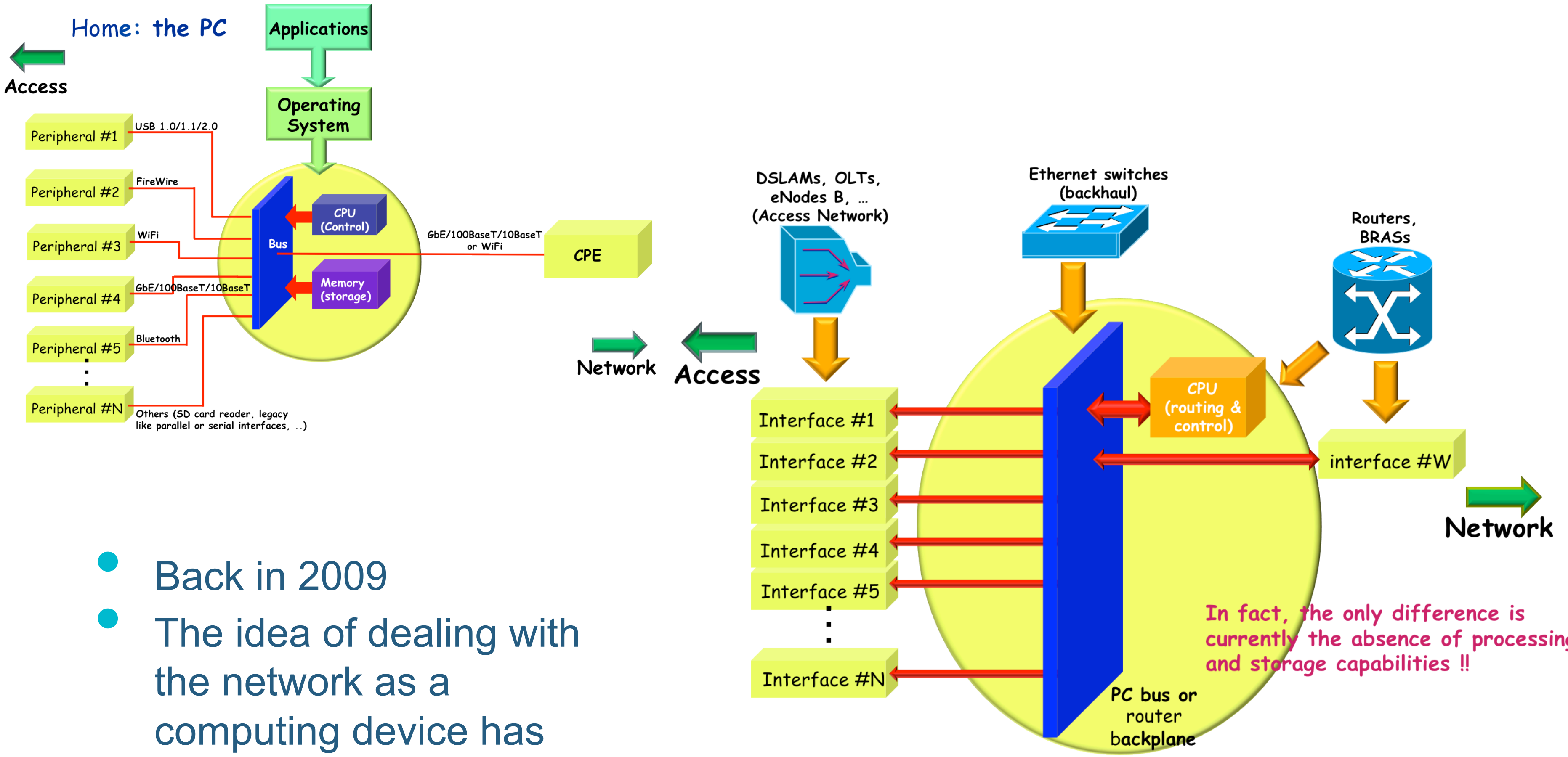
- SDN is a dramatic shift in the mechanisms to design and operate networks
  - Make network behaviour programmable beyond individual boxes
- Changes the vision from configuration to programming
  - Compiling, scripting, rapid prototyping, debugging, profiling, IDEs…
- Convergence of application and network APIs
  - Clearer, more comprehensive interfaces
- Provides a powerful toolset to deepen network virtualization

Out of the Boxes

- The network does not need to be seen any longer as a composition of individual elements
- User applications interact with the network controller(s)
- The network becomes a single entity
  - Suitable to be programmed
  - Aligned with current IT practices
- We can apply different levels of abstraction
  - Network processor and storage
  - Network Operating System
  - Network Abstract APIs
- And think of a network design flow
  - And even an IDE

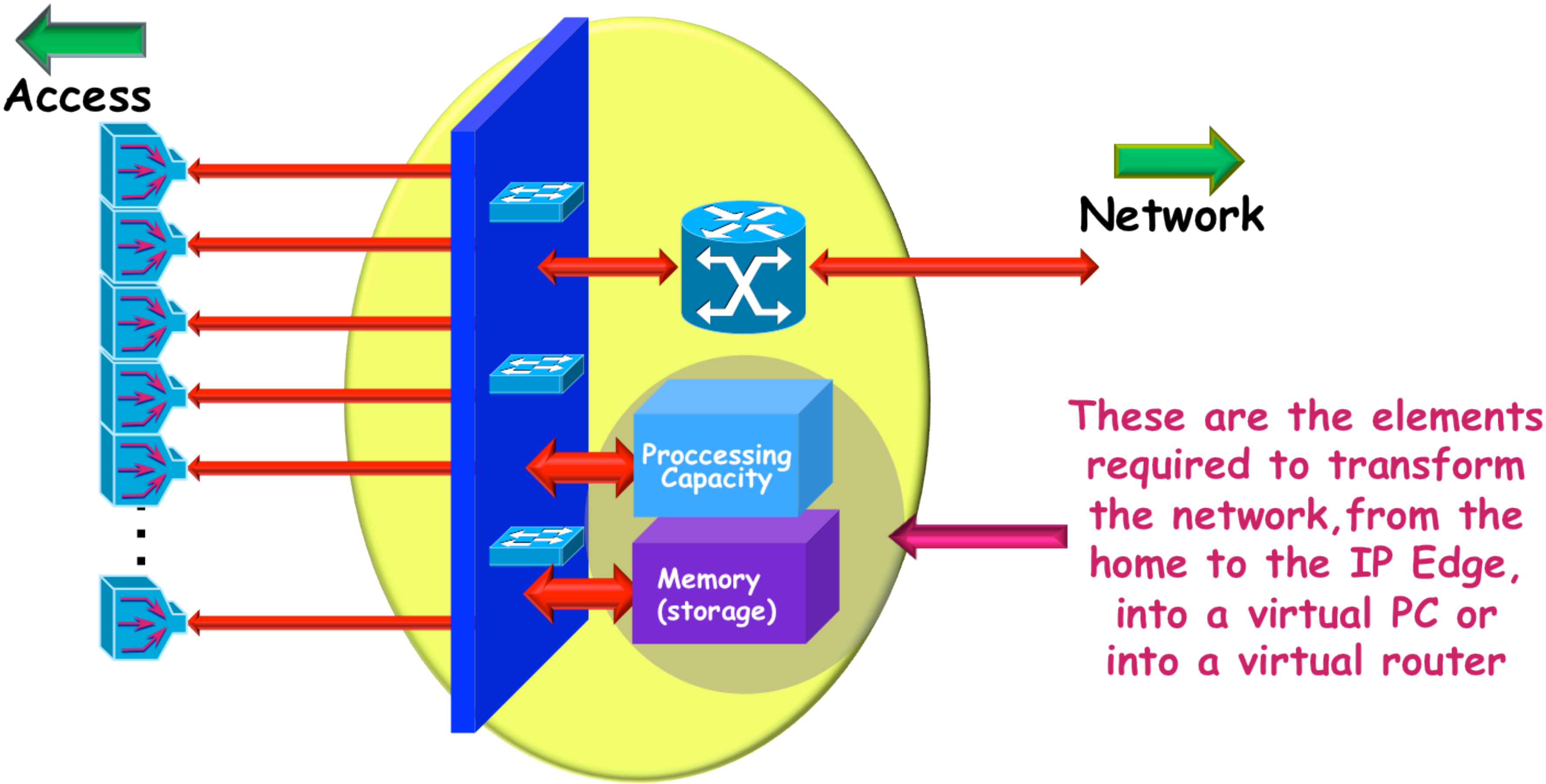


The Network and the Computer



- Back in 2009
- The idea of dealing with the network as a computing device has been around for quite some time

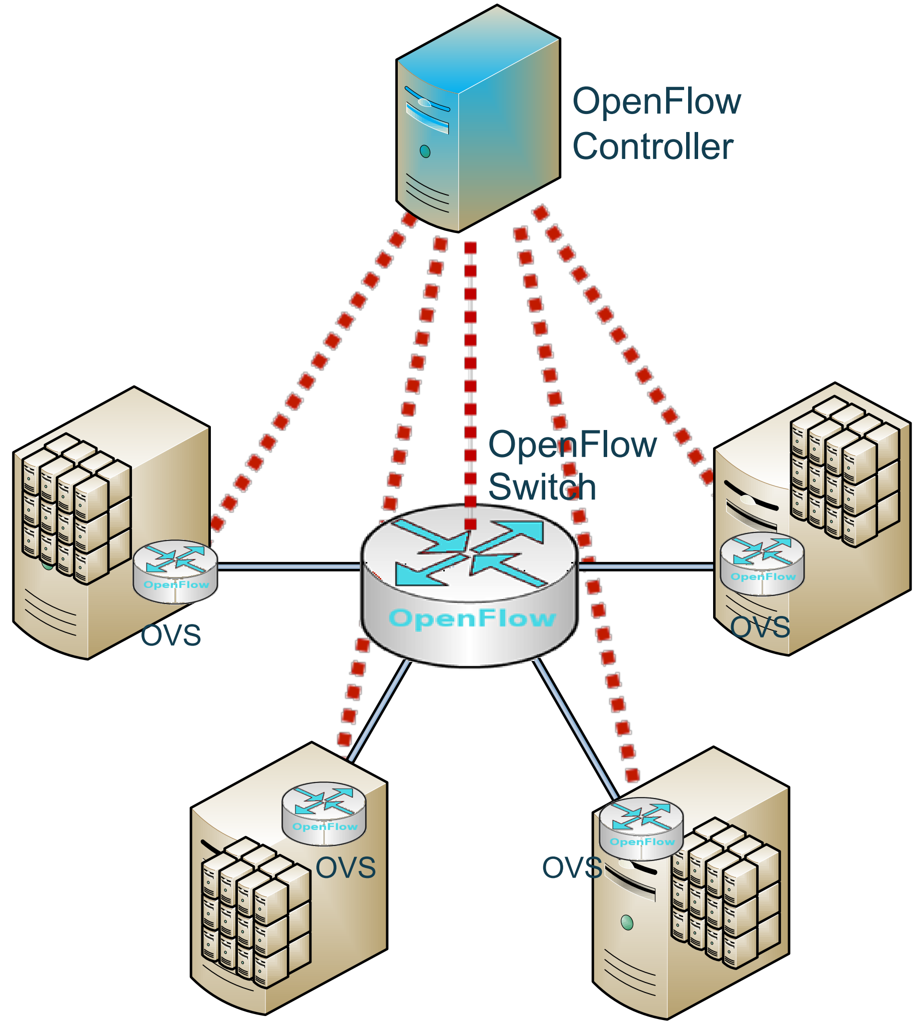
A Stored Program Model for the Network



- The SDN concepts bring into play the processing capabilities
- And the stored program

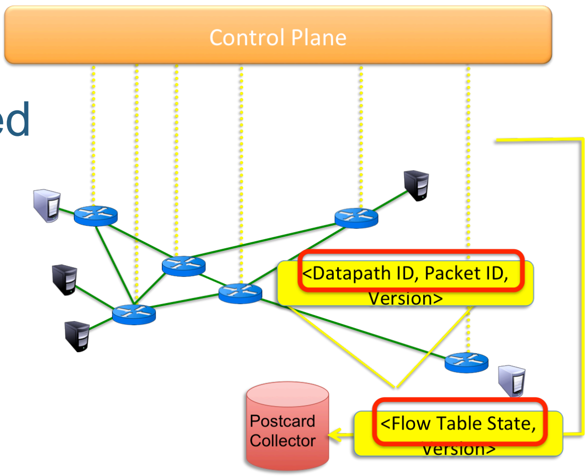
The Network Is *A* Computer

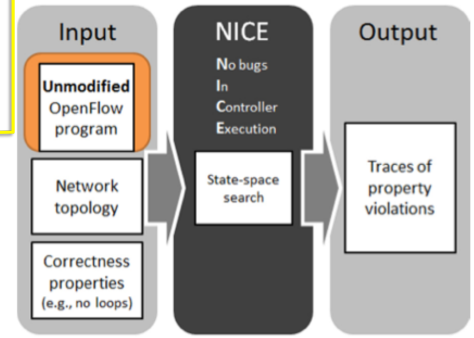
- So we can apply software development techniques and tools
- Software development and operation being multifaceted
  - Different tools for different tasks
- Static and dynamic verification
- Translation: assemblers, compilers, interpreters, linkers
- Testing and debugging
- Version and configuration control
- Dynamic composition and linking
- Development flows
- And abstraction capabilities

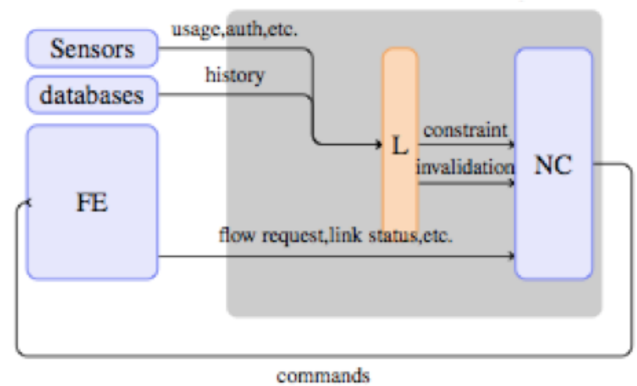


Tools on Their Way

- Considering those beyond extended controllers and simulation
- Mostly at prototype stage
- Debugging: ndb
  - Network breakpoints
  - Packet backtrace
- Verification: NICE
  - Model checking plus symbolic execution
  - Check against correctness properties
- Languages
  - Policy: FML, Procera
  - Functional: Frenetic
- Configuration control: Kinetic
  - Update mechanisms that preserve global network behaviors

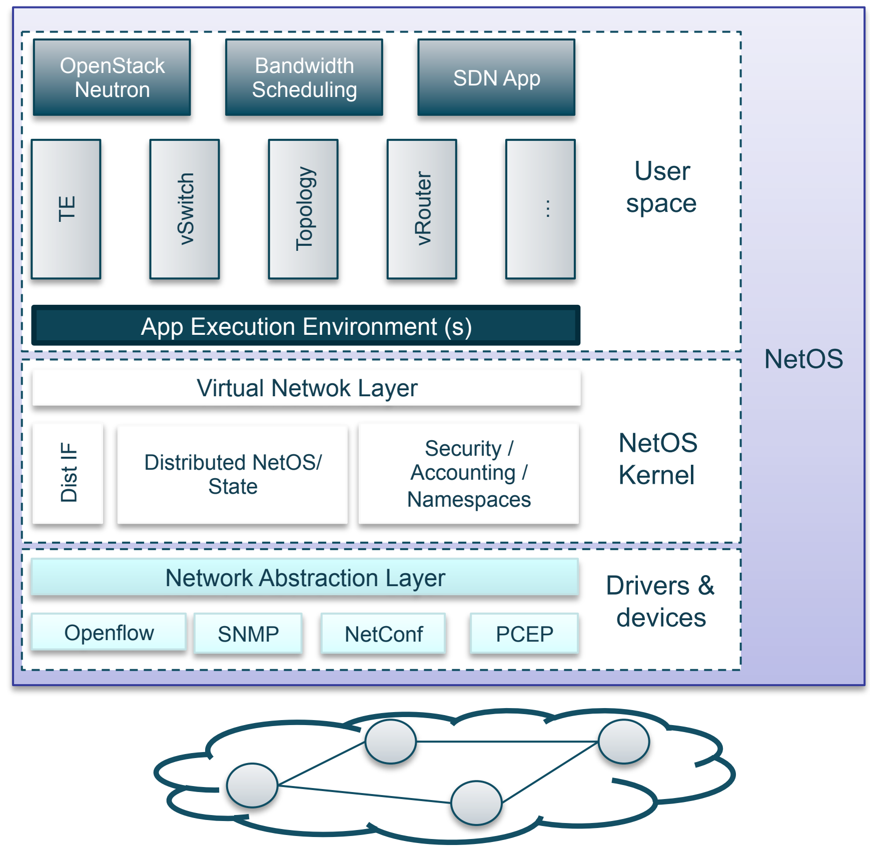






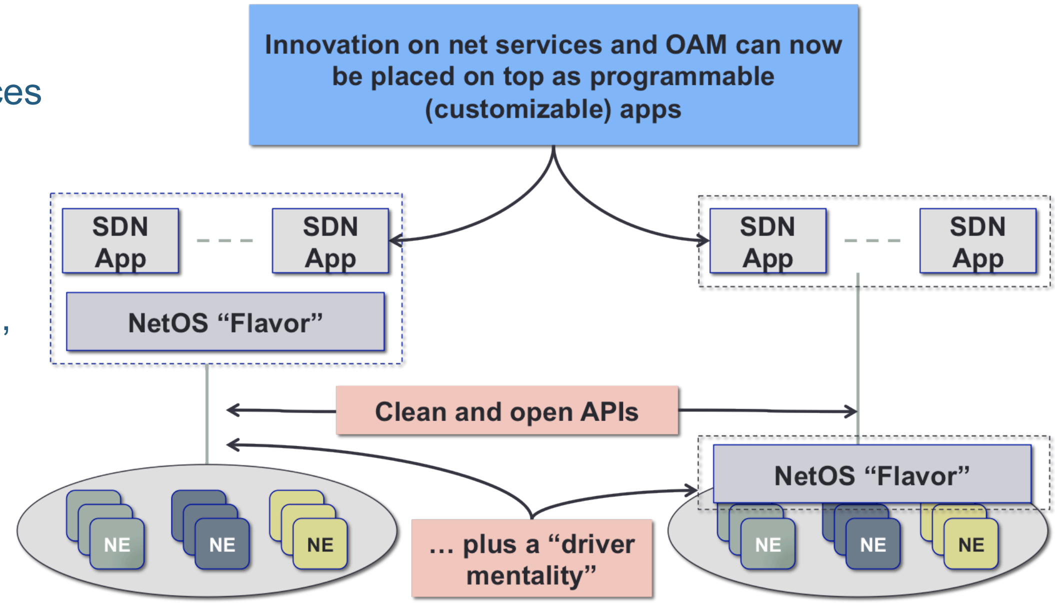
Network OS. SDN in the Widest Sense

- Providing a consistent interface to control, data and management plane
  - A layered model
  - The first take could follow an analogy with existing OS
- The kernel is realized by control plane mechanisms
- Data plane is associated with the file system
- The management plane is mapped to the system tools
  - Remember the shell
- Specific services to enforce policy and security
- And the APIs

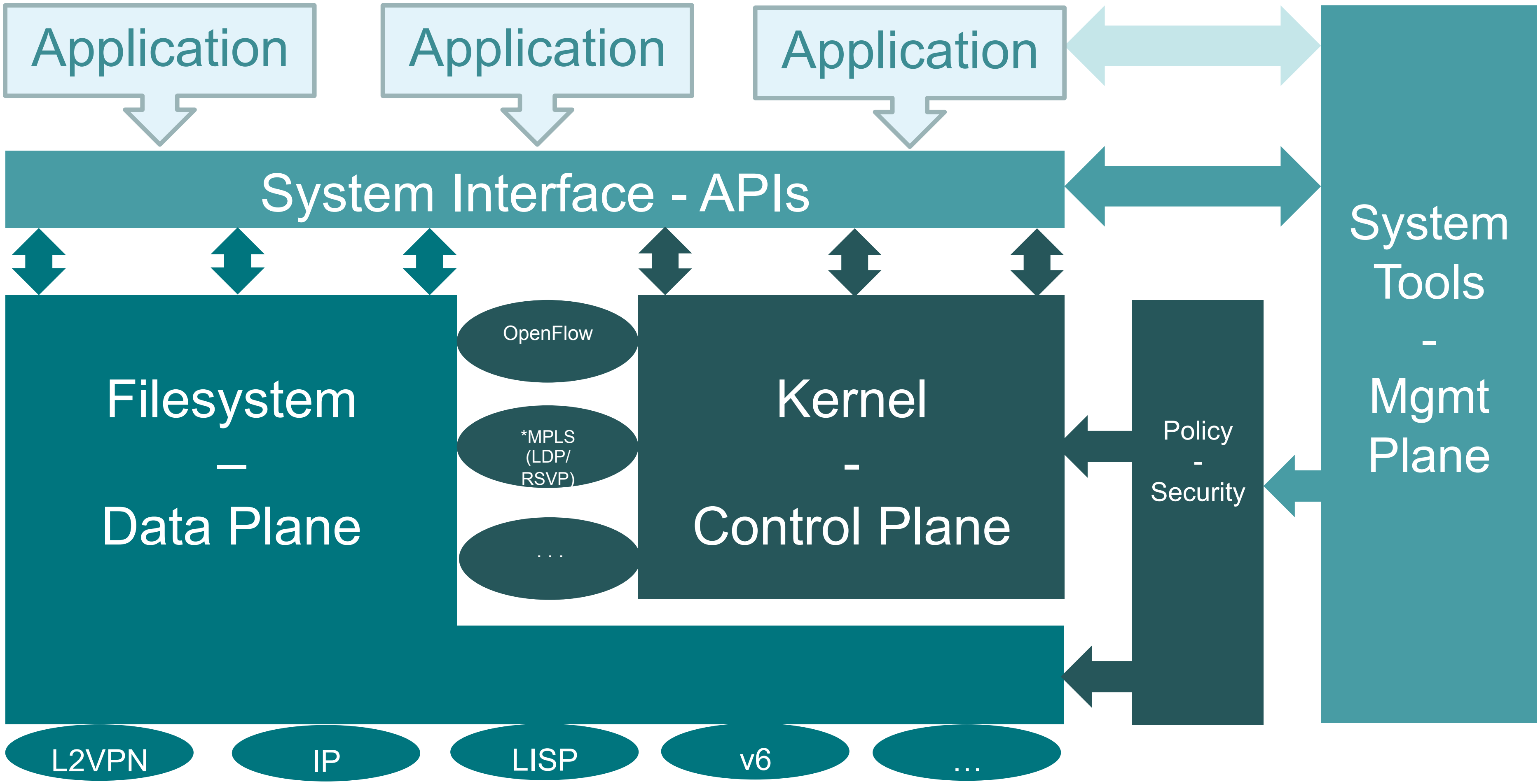


The Network OS Ecosystem

- The users
  - Network operators
    - Manage the network, create services and locate problems in a more efficient manner
  - Application providers
    - Reduced time to market for new applications, value added services, abstracted view of the network
- The networks
  - Need to address a wide variety of devices and protocols
- The goal
  - To simplify use and management of heterogeneous E2E networks
  - Access, core, datacenter….
- The POSIX reference model

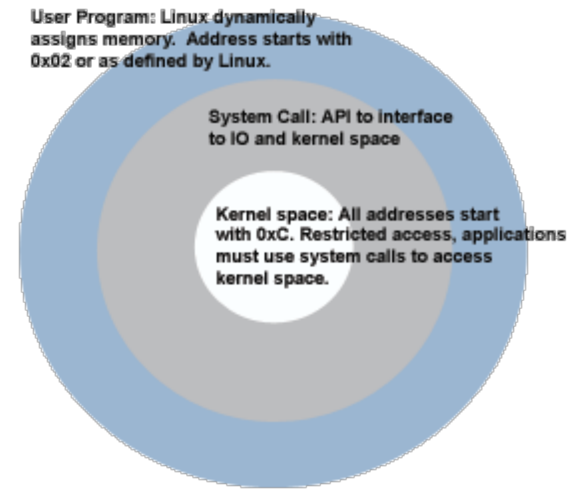


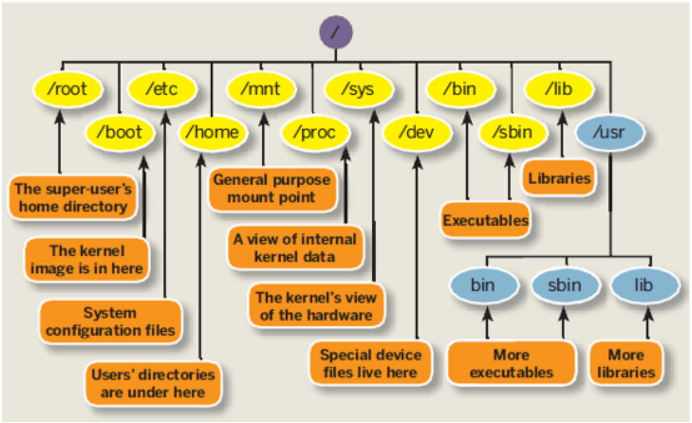
Net-wide, POSIX Style



Kernel and Filesystem

- OpenFlow as the default mechanism
  - And kernel drivers for other control plane technologies
- Strict control on kernel-mode access
  - Restricted API
- A filesystem for the data plane
  - A naming schema equivalent to directories plus filenames
  - Overlay transparent integration
  - Interaction with other Network OS instances
  - Consistent security model
- A neutral data model for internal representations
  - YANG is a clear candidate



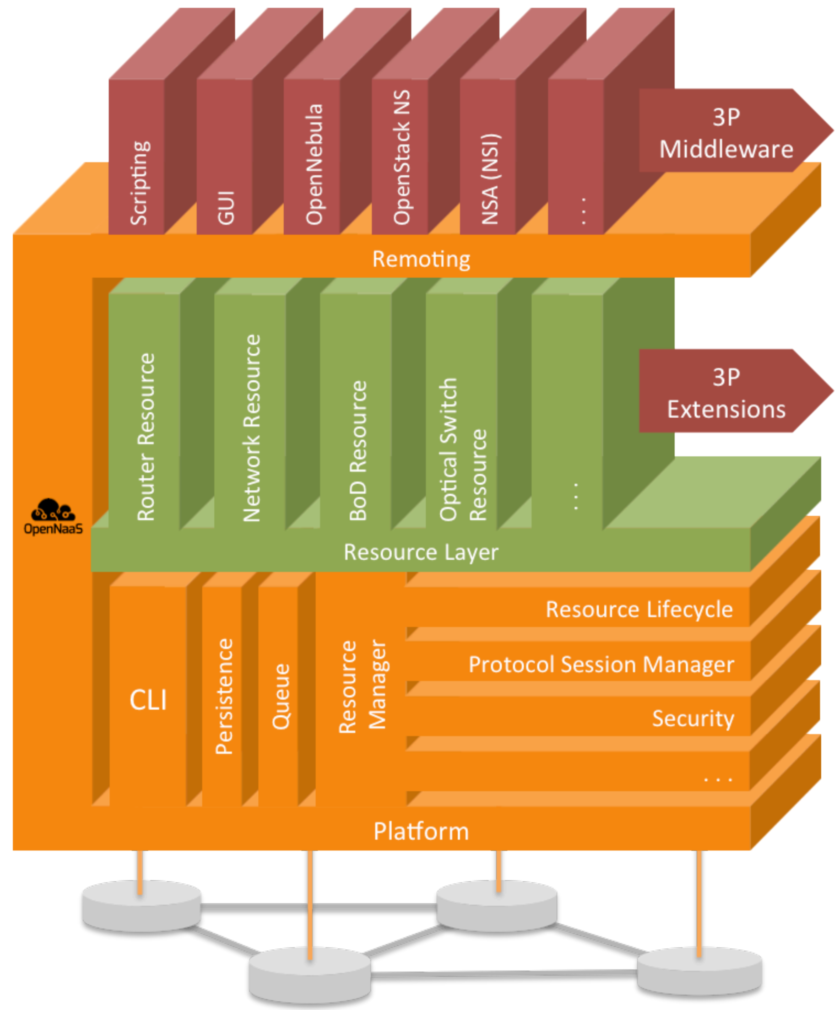


Policy and Management

- Management plane is mapped to the system process idea
  - Shell
  - Monitoring
  - Accounting
  - Policy definition
- A dedicated subset of services for policy enforcement and security
  - Converged authorization
  - Mapping from outer identities and roles
- Accountability becomes key
  - Security
  - Metering and auditing
  - Monetization

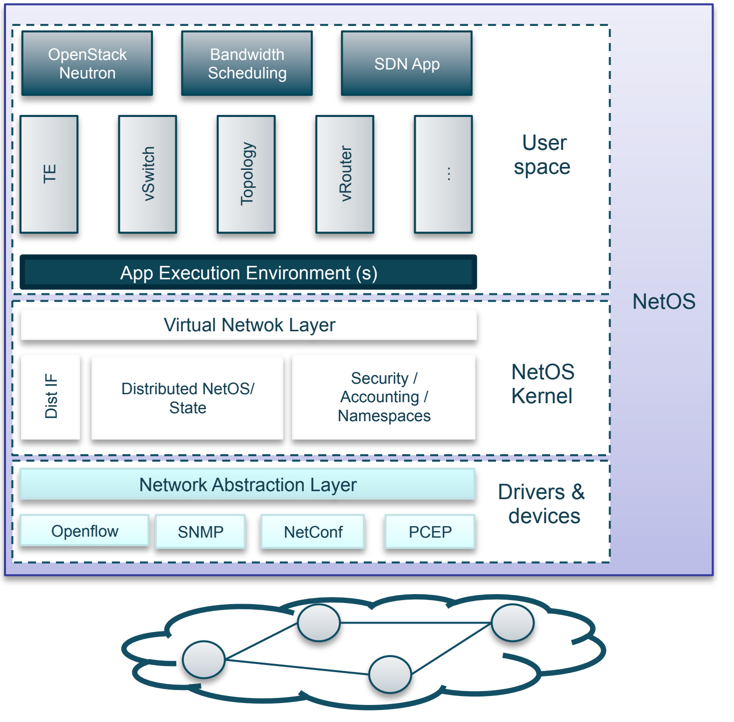
Upper Layers of Abstraction

- NaaS beyond itself
  - Current models are still very much box-oriented
  - Virtual view of current elements
- And beyond OpenFlow
  - An excellent practical base
  - As much as processor instruction sets
- A first step: consider the fabric
  - Extend OpenFlow to deal with overlay control
- And start thinking of the equivalents to
  - SQL
  - OO
  - Garbage collectors
  - `<YourPreferredITConstruct />`

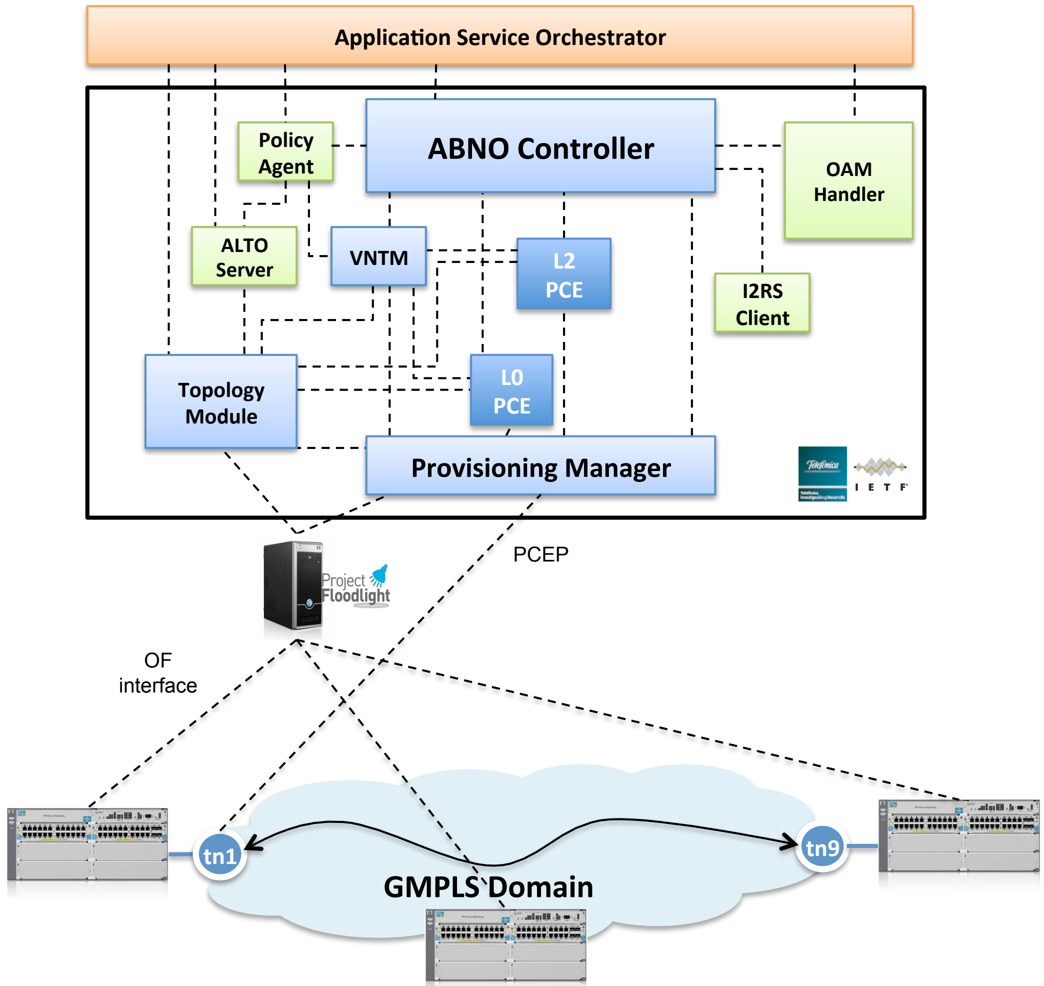


Southbound interfaces for Optical Networks

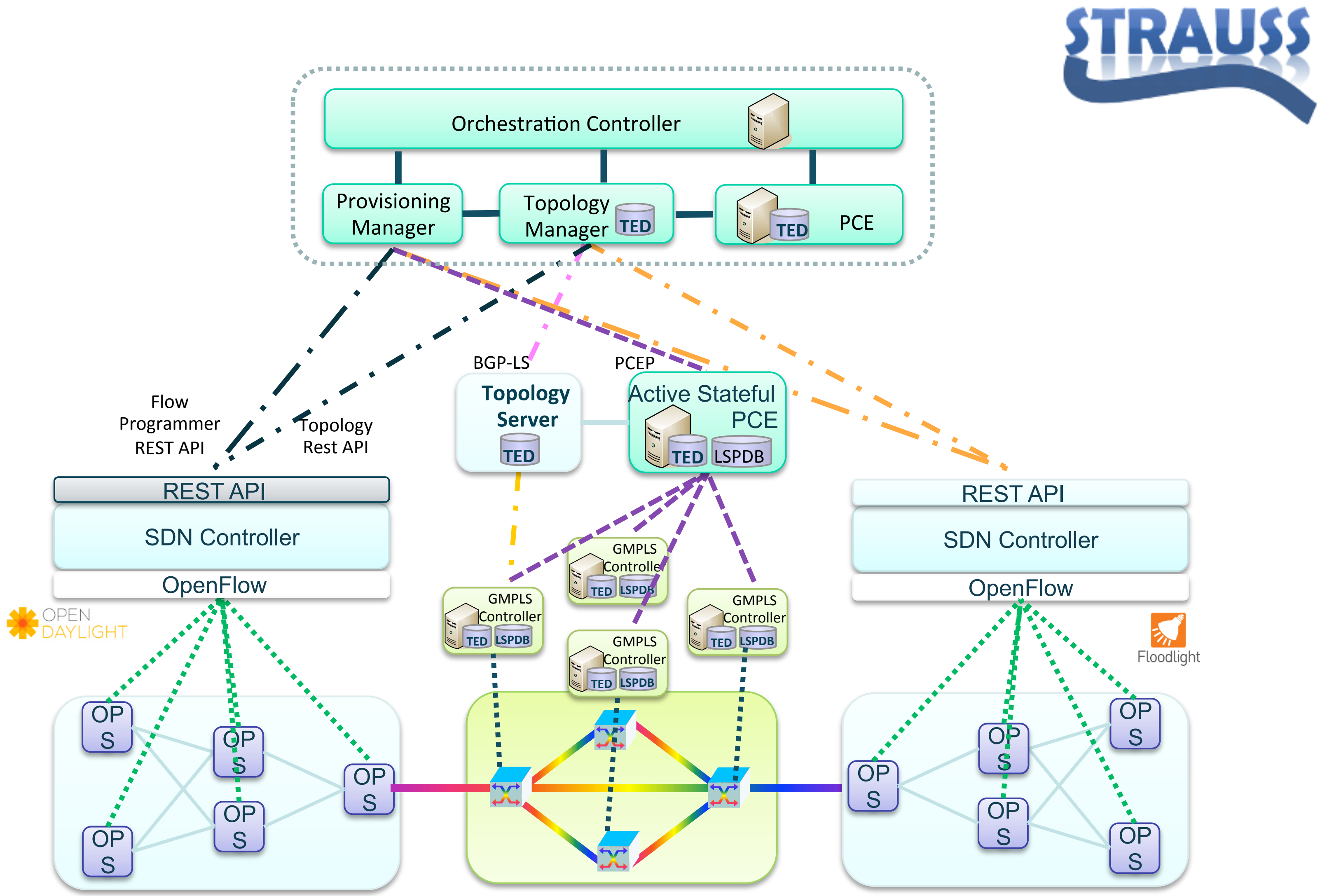
- SNMP problems with proprietary MIBs that keeps this technology as monovendor.
- PCEP extended to support provisioning and trigger the control plane.
- NETCONF is a standard to configure equipment.
  - Protocol is standard (RFC 6241), but data models are not defined (drafts).
  - Once these information models are standardized this can make easier the integration with proprietary tools.
- OpenFlow requires extensions to work with optical networks (on-going work).
  - Resilience mechanisms are required for realistic implementations.



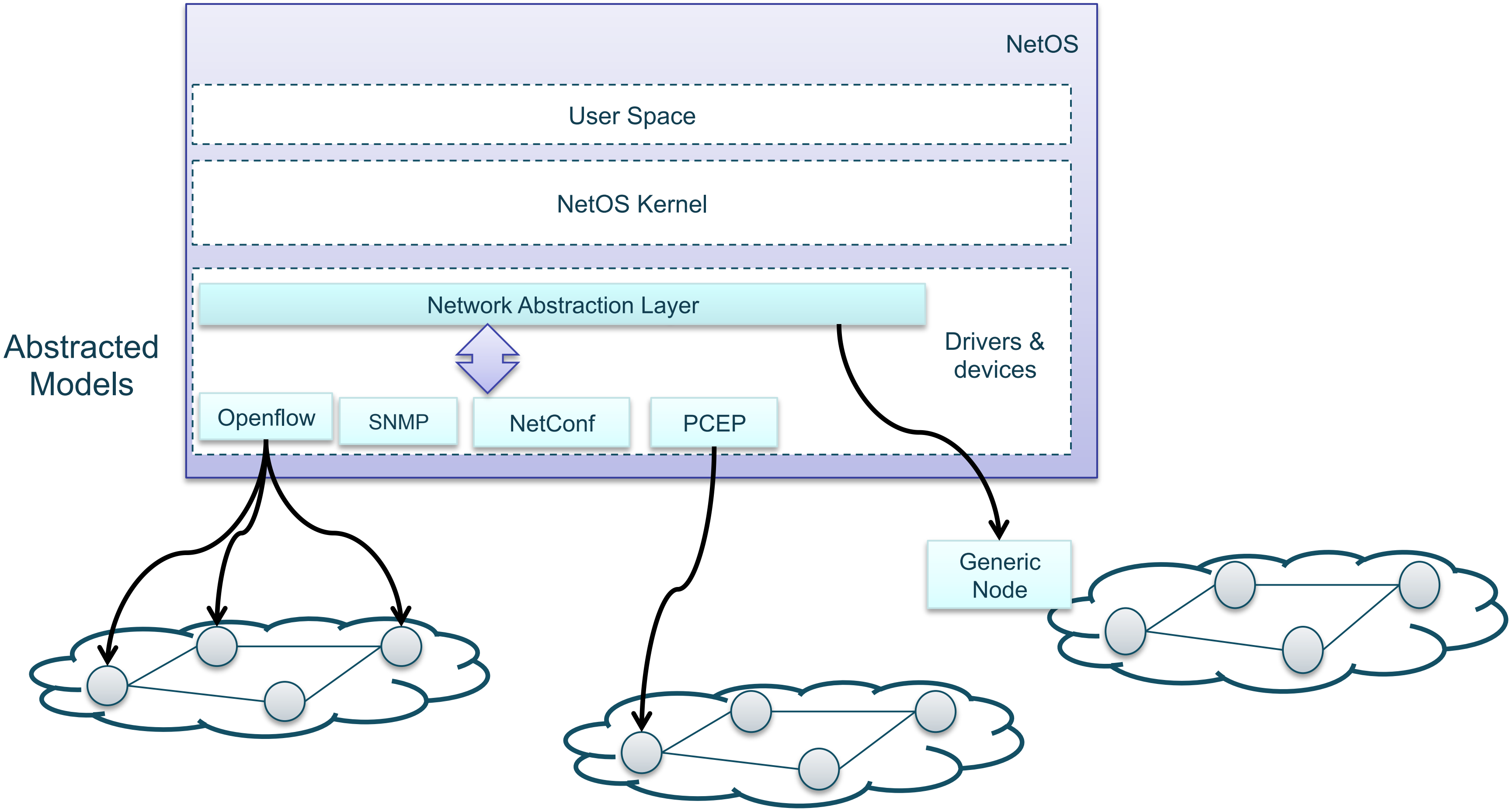
Southbound interfaces for Optical Networks



Soutbound interfaces for Optical Networks



Abstraction models for Optical Networks



Conclusions

- The network does not need to be seen any longer as a composition of individual elements.
- The network can be seen as a computer.
- We can apply software development techniques and tools.

- A environment is required to work on this direction → NetOS
  - Different abstraction models can be used.
  - Applications can run on top of the Operating System
  - Kernel of the system can grow as far as functions are required.

- South bound interfaces to optical networks are required.
  - Protocols should be extended to support remote instantiation
  - Abstracted models can help to have a common driver where we can plug any network element.