Architectural Musings on SDN
(“and now for something completely different…”)

David Meyer
CTO and Chief Scientist, Brocade
Director, Advanced Technology Center, University of Oregon
RIPE 66
May 2013
Dublin, Ireland
dmm@{brocade.com,uoregon.edu,1-4-5.net,…}
http://www.1-4-5.net/~dmm/talks/ripe66.pdf

Agenda

- Introduction
- Architectural Features for Scalability and Evolvability
  - and why we might care
- A Quick Tour Through the SDN Design Space
- A Few Conclusions
- Q&A

Danger Will Robinson!!!

This talk is intended to be controversial/provocative (and a bit "sciencey")

Introduction

- "Lots" of hype around OpenFlow, SDN, SDS, …
  - duh
- In trying to understand all of this, I went back architectural principles
  - An attempt to take an objective look at all of this
  - Ideas from control theory, systems biology, quantitative risk engineering, …
- Obviously we need programmatic automation of
  - Configuration, management, monitoring, optimization(s), …
  - Some components already available
    - Puppet, Chef, rancid, ...
  - Note everything open (interfaces, APIs, protocols, source) – along with s/w a macro-trend
- Perhaps obvious:
  - Scalability and Evolvability key to building/operating the Internet
  - But what are Scalability/Evolvability, and what architectures enable them?
- Through this lens: What *is* going on with OpenFlow, SDN, …?

Bottom Line

I hope to convince you that uncertainty and volatility are the “coin of the realm” of the future, why this is the case, how SDN (and the rise of software in general) is accelerating this effect, and finally, what we might do to take advantage of it.[0]

0 s/take advantage of/*survive*/ -- @smd

What are Scalability and Evolvability?

- First, why do we care?
  - Goes without saying?
  - That said…
- **Scalability** is robustness to changes to the size and complexity of a system as a whole
- **Evolvability** is robustness of lineages to changes on long time scales
- Other system features cast as robustness
  - **Reliability** is robustness to component failures
  - **Efficiency** is robustness to resource scarcity
  - **Modularity** is robustness to component rearrangements
- In our case: holds for protocols, systems, and operations

OK, Fine. But What is Robustness?

- ***Definition***: A *[property*] of a *[system]* is **robust** if it is *[invariant]* with respect to a *[set of perturbations], up to some limit*

- **Fragility** is the opposite of robustness
  - If you're fragile you depend on 2nd order effects (acceleration) and the curve is concave
  - Catch me later if you'd like to chat further about this…

- A system can have a *property* that is *robust* to one set of perturbations and yet *fragile* for a *different property* and/or perturbation → the system is **Robust Yet Fragile (RYF-complex)**
  - Or the system may collapse if it experiences perturbations above a certain threshold (K-fragile)

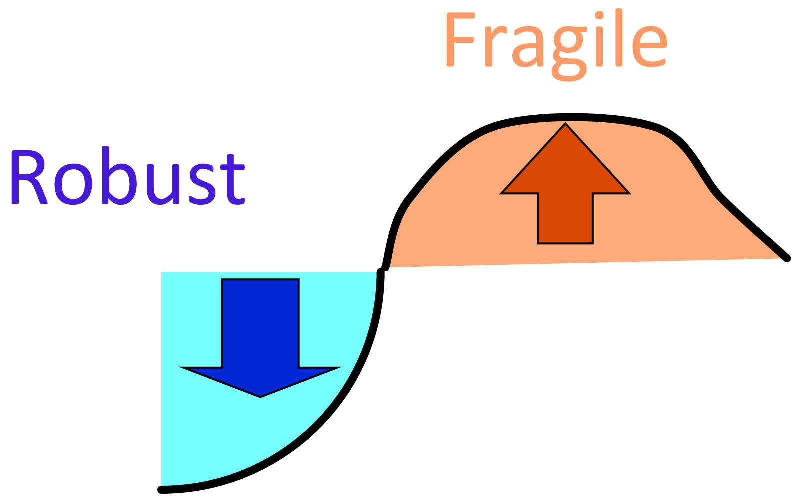
- Example: A possible ***RYF tradeoff*** is that a system with high efficiency (i.e., using minimal system resources) might be unreliable (i.e., fragile to component failure) or hard to evolve

See Alderson, D. and J. Doyle, "Contrasting Views of Complexity and Their Implications For Network-Centric Infrastructures", IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART A: SYSTEMS AND HUMANS, VOL. 40, NO. 4, JULY 2010

Robust Yet Fragile (RYF)

[a system] can have
[a property] ***robust*** for
[a set of perturbations]

Yet be ***fragile*** for
[a different property]
Or [a different perturbation]



Conjecture: The RYF tradeoff is a *hard limit* that cannot be overcome.

Slide courtesy John Doyle

RYF Examples

Robust	Yet Fragile
☺ Efficient, flexible metabolism	☹ Obesity and diabetes
☺ Complex development	☹ Rich microbe ecosystem
☺ Immune systems	☹ Inflammation, Auto-Im.
☺ Regeneration & renewal	☹ Cancer
🗐 Complex societies	☠ Epidemics, war, …
🪟 Advanced technologies	💣 Catastrophic failures

- "Evolved" mechanisms for robustness *allow for,* even *facilitate,* novel, severe fragilities elsewhere
- Often involving hijacking/exploiting the same mechanism
  - We've certainly seen this in the Internet space
- There are hard constraints (i.e., theorems with proofs)

Brief Aside: Fragility and Scaling (geeking out for a sec…)

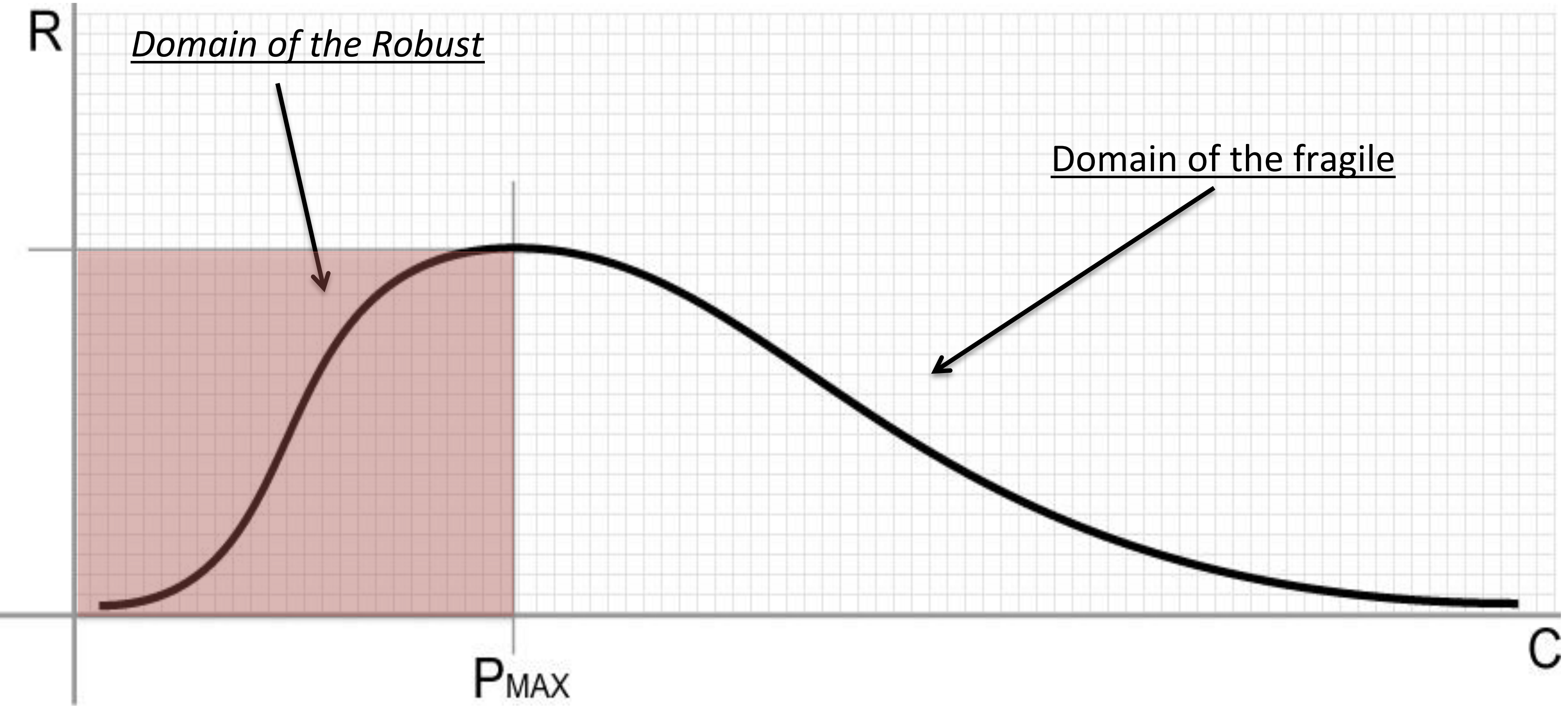
- A bit of a formal description of fragility
  - Let *z* be some stress level, *p* some property, and
  - Let *H(p,z)* be the (negative valued) harm function
  - Then for the fragile the following must hold
    - ***$H(p,nz) < nH(p,z)$ for $0 < nz < K$***
    - Basically, the "harm function" is non-linear
    - This inequality is importantly non-mean preserving (Jensen's Inequality)
    - Non-mean preserving: ***$H(p,(z_1 + z_2)/2) \mathrel{!=} (H(p,z_1) + H(p,z_2))/2$***
      - $\rightarrow$ model error and hence additional uncertainty

- For example, a coffee cup on a table suffers non-linearly more from large deviations (*H(p, nz)*) than from the cumulative effect of smaller events (*nH(p,z)*)
  - So the cup is damaged far more from (i.e., destroyed by) *tail events* than those within a few σ of the mean
  - Too theoretical? Perhaps, but consider: ARP storms, micro-loops, congestion collapse, AS 7007, …
  - BTW, nature requires this property
    - Consider: jump off something 1 foot high 30 times v/s jumping off something 30 feet high once

- When we say something scales like $O(n^2)$, what we mean is the damage to the network has constant acceleration (2) for *weird* enough n (e.g., outside say, 10 σ)
  - Again, ARP storms, congestion collapse, AS 7007, DDOS, … $\rightarrow$ non-linear damage

- Something we don't have time for: Antifragility
  - Is this related to our work? See http://www.renesys.com/blog/2013/05/syrian-internet-fragility.shtml

Robustness vs. Complexity
Systems View



What this curve is telling us is that a system needs complexity to achieve robustness (wrt some feature to some perturbation), but like everything else, too much of of a good thing….

Ok, but what is *Complexity*?

"In our view, however, complexity is most succinctly discussed in terms of functionality and its robustness. Specifically, ***we argue that complexity in highly organized systems arises primarily from design strategies intended to create robustness to uncertainty in their environments and component parts.***"

See Alderson, D. and J. Doyle, "Contrasting Views of Complexity and Their Implications For Network-Centric Infrastructures", IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART A: SYSTEMS AND HUMANS, VOL. 40, NO. 4, JULY 2010

BTW, This Might Also Be Obvious But…

- Networks are incredibly general and expressive structures
  - *G = (V,E)*

- Networks are extremely common in nature
  - Immune systems, energy metabolism, transportation systems, *Internet*, macro economies, forest ecology, the main sequence (stellar evolution), galactic structures, ….
  - "Almost everything you see can be explained as either a network and/or a queue"

- So it comes as no surprise that we study, for example, biological systems in our attempts to get a deeper understanding of complexity and the architectures that provide for scalability, evolvability, and the like

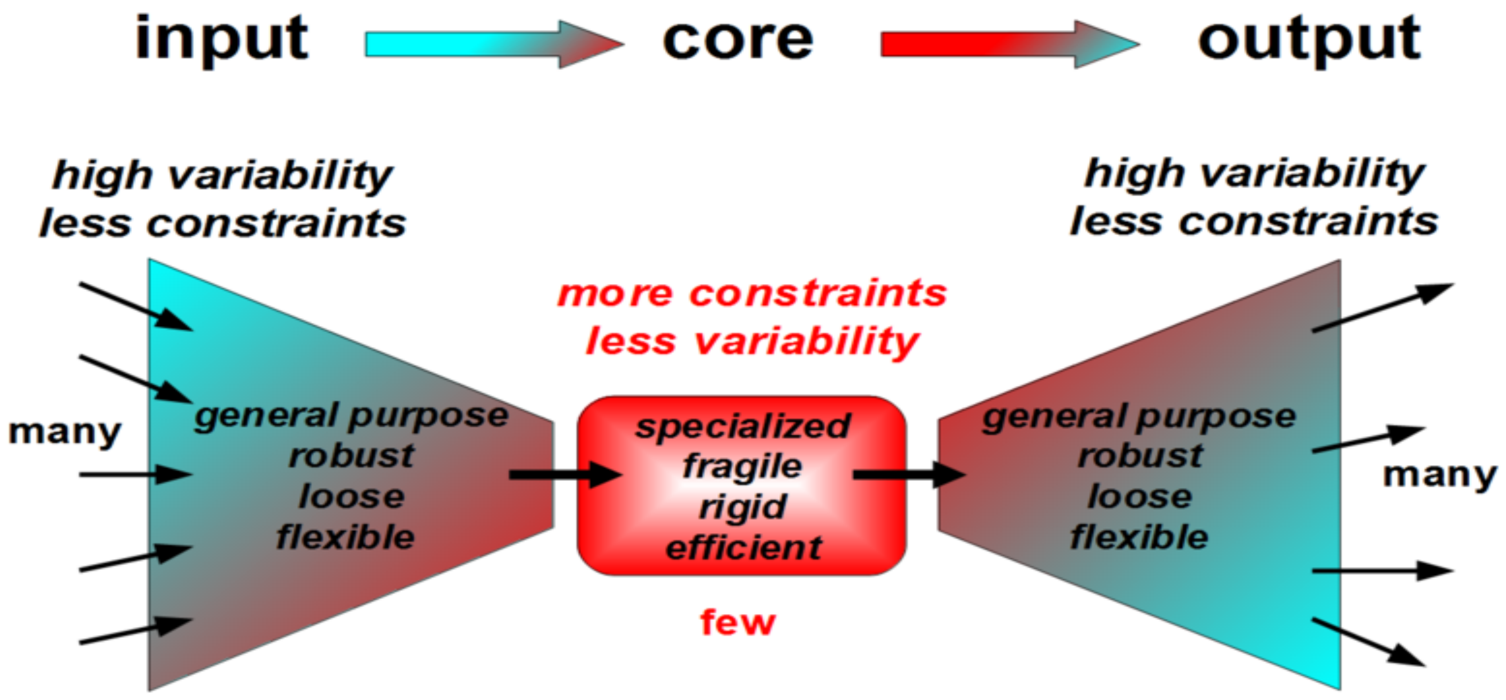
- Ok, this is cool, but what are the key architectural takeaways from this work for us ?
  - where *us \in {ops, engineering, architects …}*
  - And how might this effect the way we build and operate networks?

Key Architectural Takeaways

- What we have learned is that there are ***fundamental architectural building blocks*** found in systems that scale and are evolvable. These include
  - RYF complexity
  - **Bowtie architectures**
  - Massively distributed with *robust* control loops
    - Contrast optimal control loops and hop-by-hop control
  - Highly layered
    - But with layer violations
  - Protocol Based Architectures (PBAs)
  - Degeneracy

Bowties 101

Constraints that Deconstrain

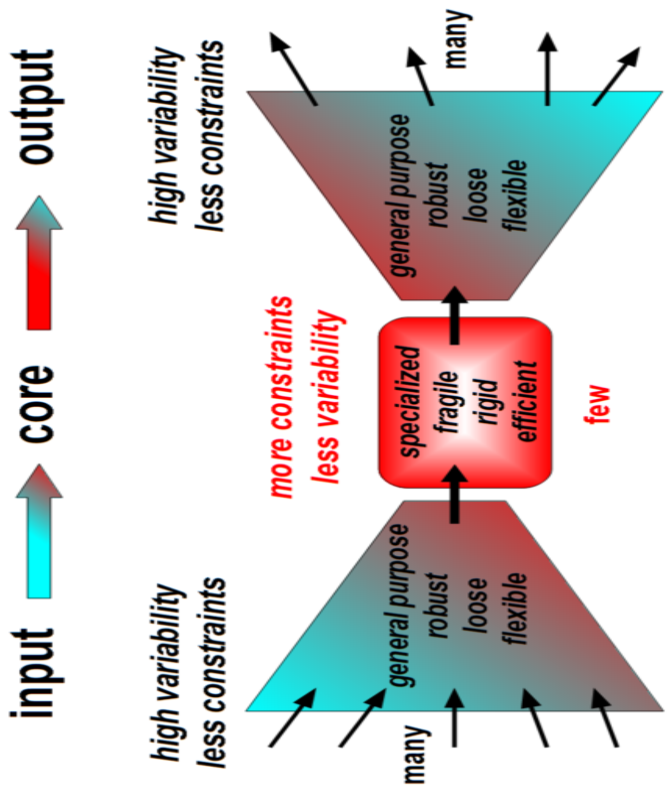


For example, the reactions and metabolites of core metabolism, e.g., *ATP metabolism*, Krebs/Citric Acid cycle signaling networks, …

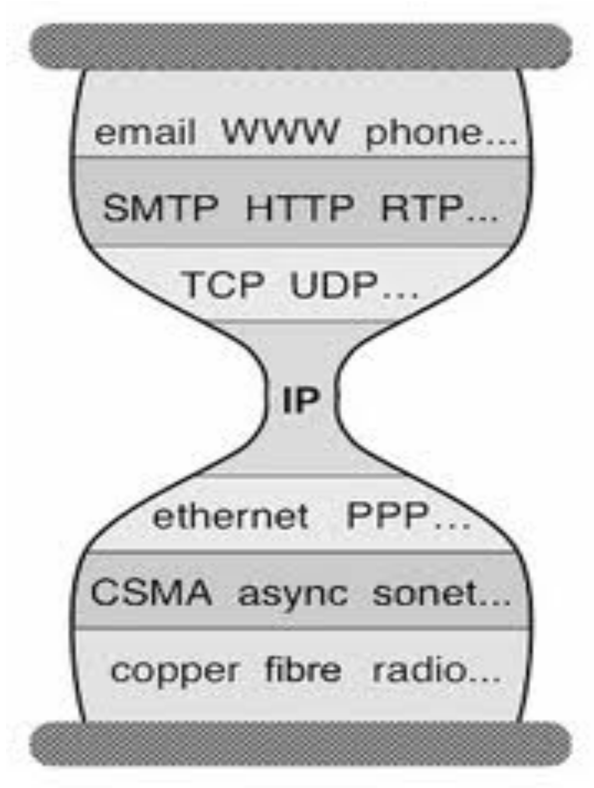
See Kirschner M., and Gerhart J., "Evolvability", Proc Natl Acad Sci USA , 95:8420–8427, 1998.

But Wait a Second

Anything Look Familiar?



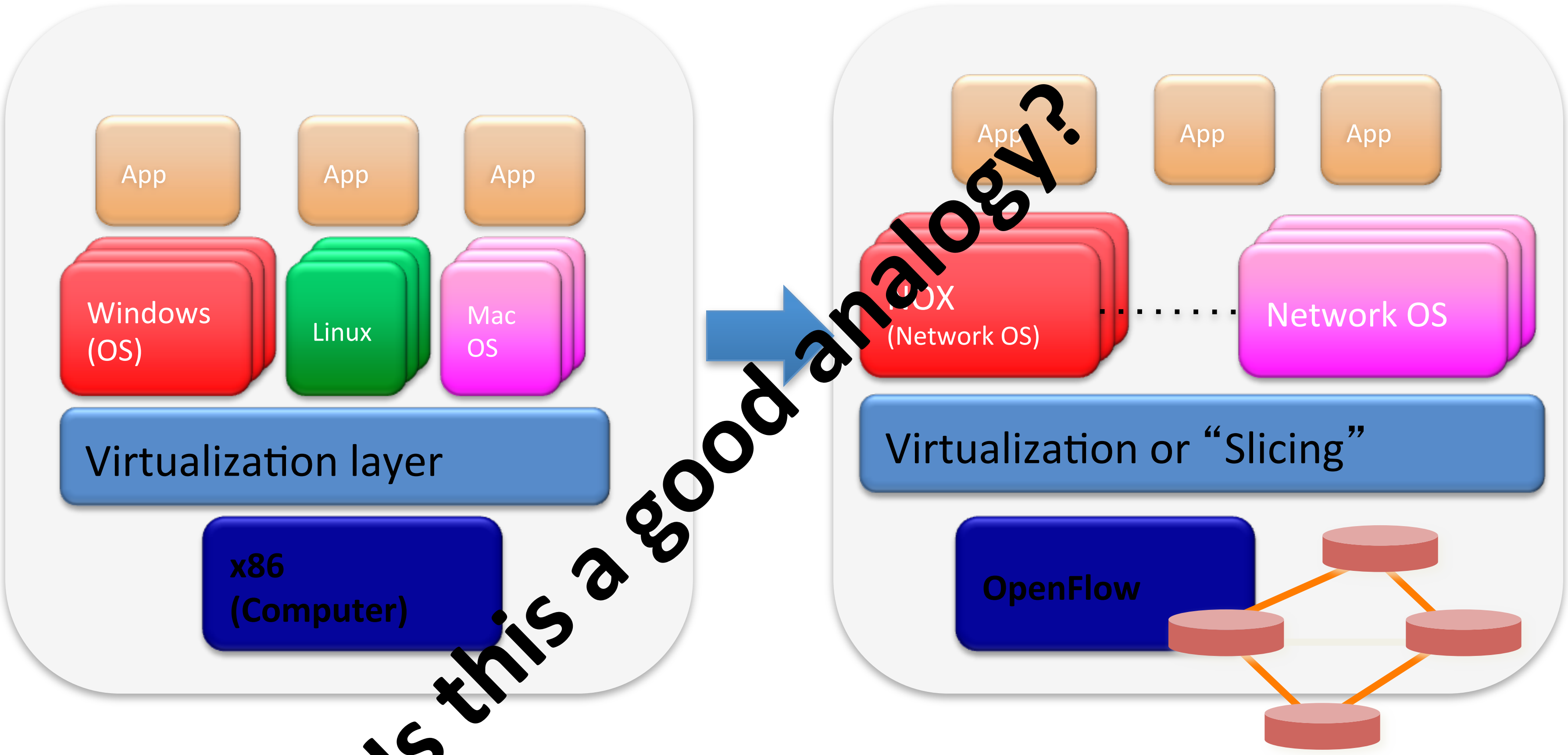
Bowtie Architecture



Hourglass Architecture

The *Protocol Hourglass* idea appears to have originated with Steve Deering. See Deering, S., "Watching the Waist of the Protocol Hourglass", IETF 51, 2001, http://www.iab.org/wp-content/IAB-uploads/2011/03/hourglass-london-ietf.pdf. See also Akhshabi, S. and C. Dovrolis, "The Evolution of Layered Protocol Stacks Leads to an Hourglass-Shaped Architecture", http://conferences.sigcomm.org/sigcomm/2011/papers/sigcomm/p206.pdf.

So Let's Have a Look at OF/SDN Here's the Thesis



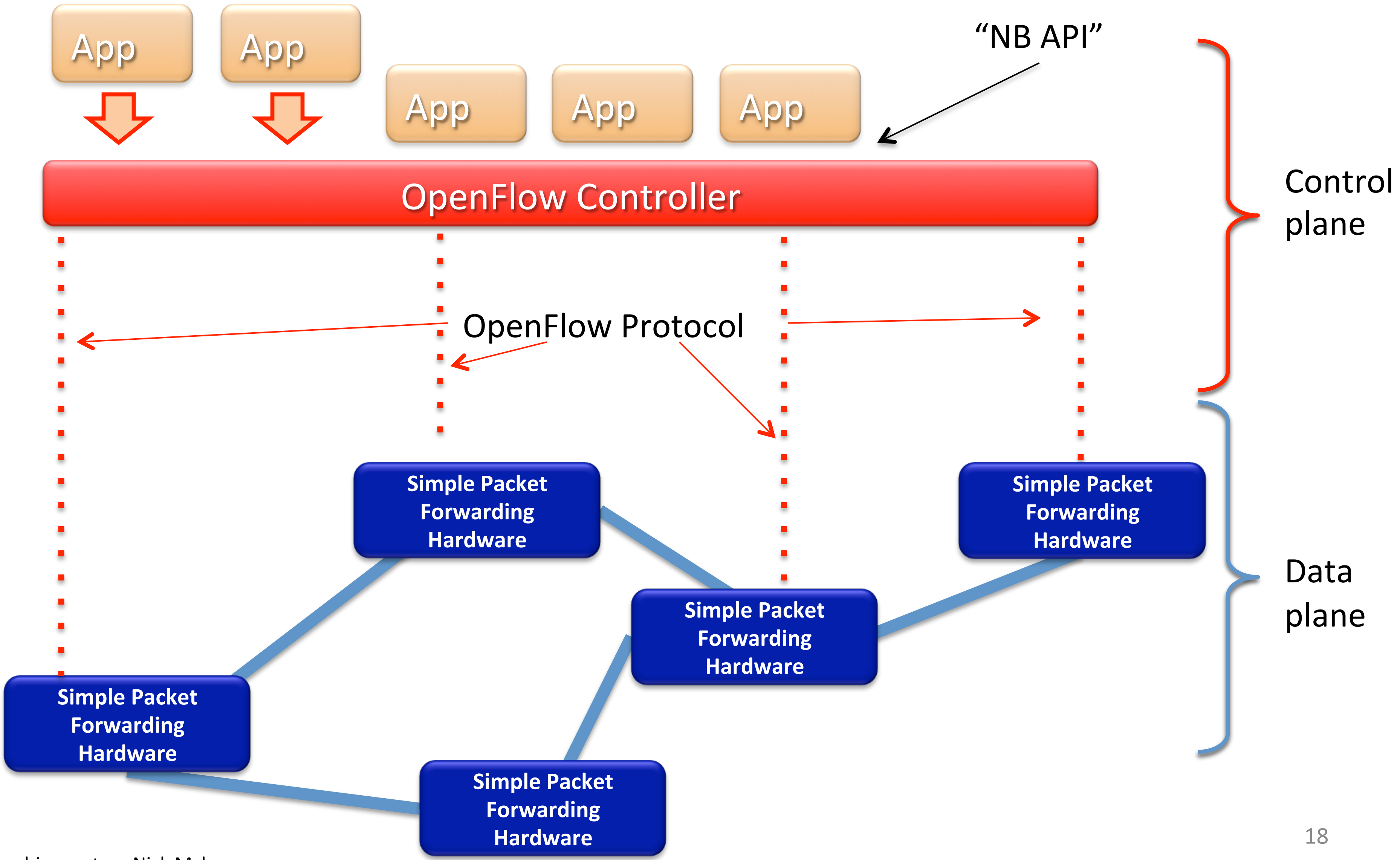
Computer Industry

Network Industry

- Separation of Control and Data Planes
- Open Interface to Data Plane
- Centralized Control (logically?)

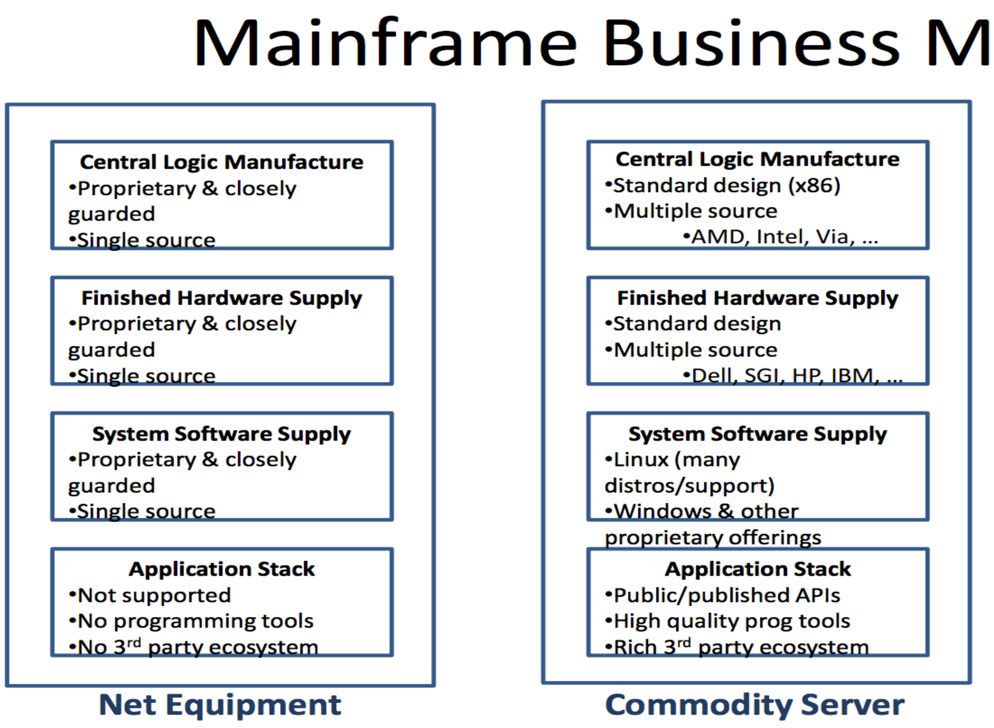
Graphic Courtesy Rob Sherwood

A Closer Look


App
App
App
App
App
"NB API"
OpenFlow Controller
Control plane
OpenFlow Protocol
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Data plane

Graphic courtesy Nick Mckeown

So Does the OF/SDN-Compute Analogy Hold?



Mainframe Business Model

Net Equipment	Commodity Server
Central Logic Manufacture •Proprietary & closely guarded •Single source	**Central Logic Manufacture** •Standard design (x86) •Multiple source •AMD, Intel, Via, …
Finished Hardware Supply •Proprietary & closely guarded •Single source	**Finished Hardware Supply** •Standard design •Multiple source •Dell, SGI, HP, IBM, …
System Software Supply •Proprietary & closely guarded •Single source	**System Software Supply** •Linux (many distros/support) •Windows & other proprietary offerings
Application Stack •Not supported •No programming tools •No 3rd party ecosystem	**Application Stack** •Public/published APIs •High quality prog tools •Rich 3rd party ecosystem

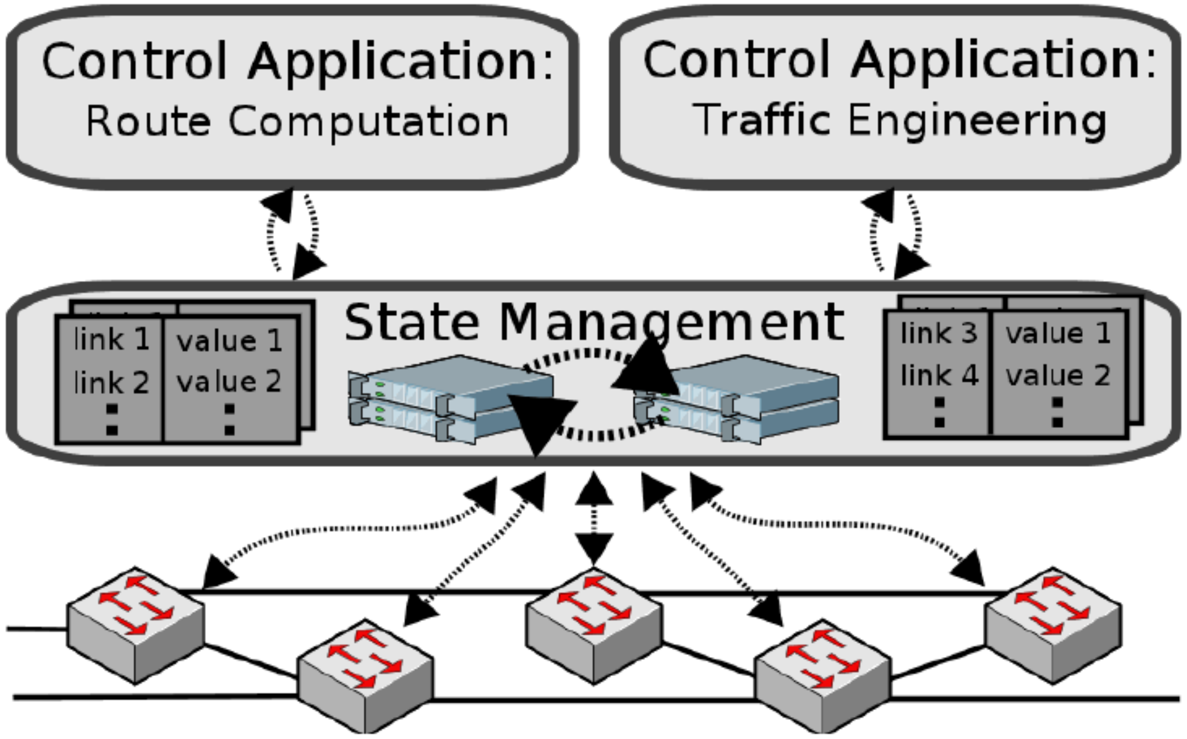
- **Example**:
  - Juniper EX 8216 (used in core or aggregation layers)
  - Fully configured list: $716k w/o optics and $908k with optics
- **Solution**: Merchant silicon, H/W independence, open source protocol/mgmt stack

Really Doesn't Look Like It

A better analogy would be an open source network stack/OS on white-box hardware

Graphic courtesy James Hamilton, http://mvdirona.com/jrh/TalksAndPapers/JamesHamilton_POA20101026_External.pdf.

BTW, Logically Centralized?



Key Observation: Logically centralized → distributed system → tradeoffs between control plane convergence and state consistency model. See the *CAP Theorem.*

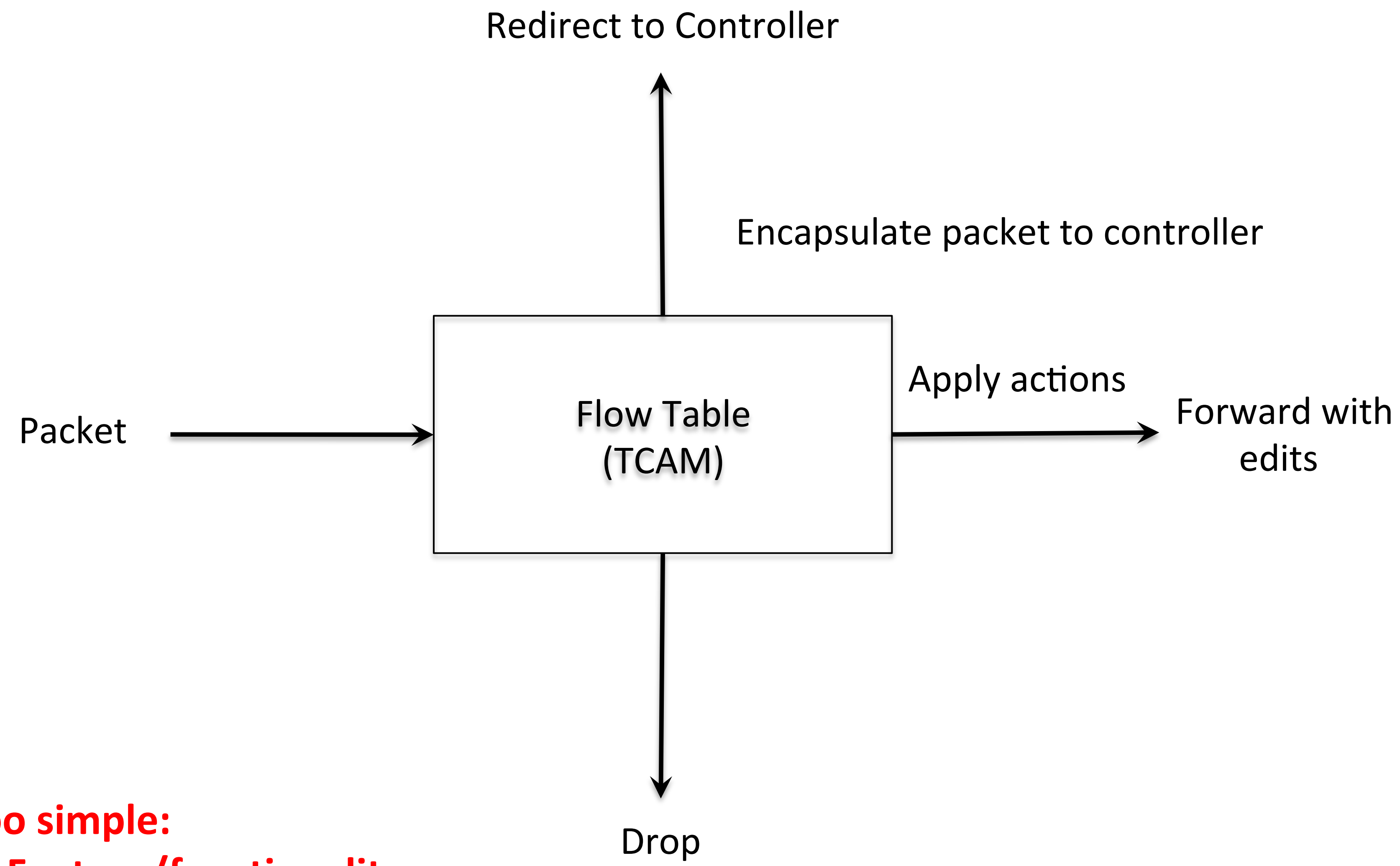
Architectural Implication: If you break CP/DP fate sharing, you have to deal the following physics: $\Omega(\text{convergence}) = \Sigma\, RTT(\text{controller}, \text{switch}_i) + PPT(\text{controller}) + PPT(\text{switch}_i)$

Graphic courtesy Dan Levin <dlevin@net.t-labs.tu-berlin.de>

BTW, Nothing New Under The Sun…

- ***Separation of control and data planes*** is not a new idea. Nor is flow-based forwarding. Examples include:
  - SS7
  - Ipsilon Flow Switching
    - Centralized flow based control, ATM link layer
    - GSMP (RFC 3292)
  - AT&T SDN
    - Centralized control and provisioning of SDH/TDM networks
  - A similar thing happened in TDM voice to VOIP transition
    - Softswitch → Controller
    - Media gateway → Switch
    - H.248 → Device interface
    - Note 2nd order effect: This was really about circuit → packet
  - ForCES
    - Separation of control and data planes
    - RFC 3746 (and many others)
  - …

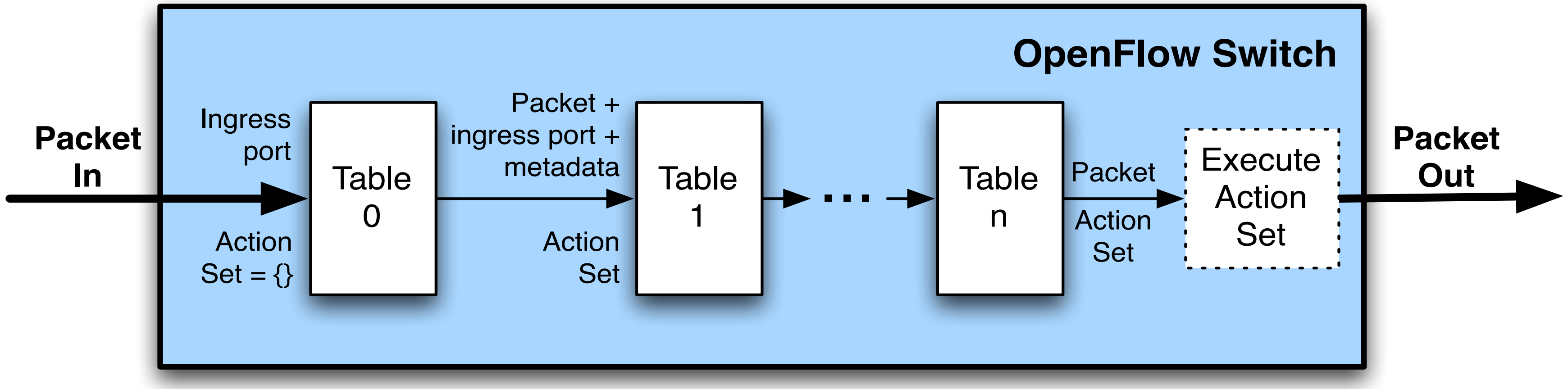
Drilling Down a Bit
OpenFlow Switch Model Version 1.0



Too simple:

- **Feature/functionality**
- **Expressiveness – consider shared table learning/forwarding bridge**

OK, Fast Forward to Today: OF 1.1+

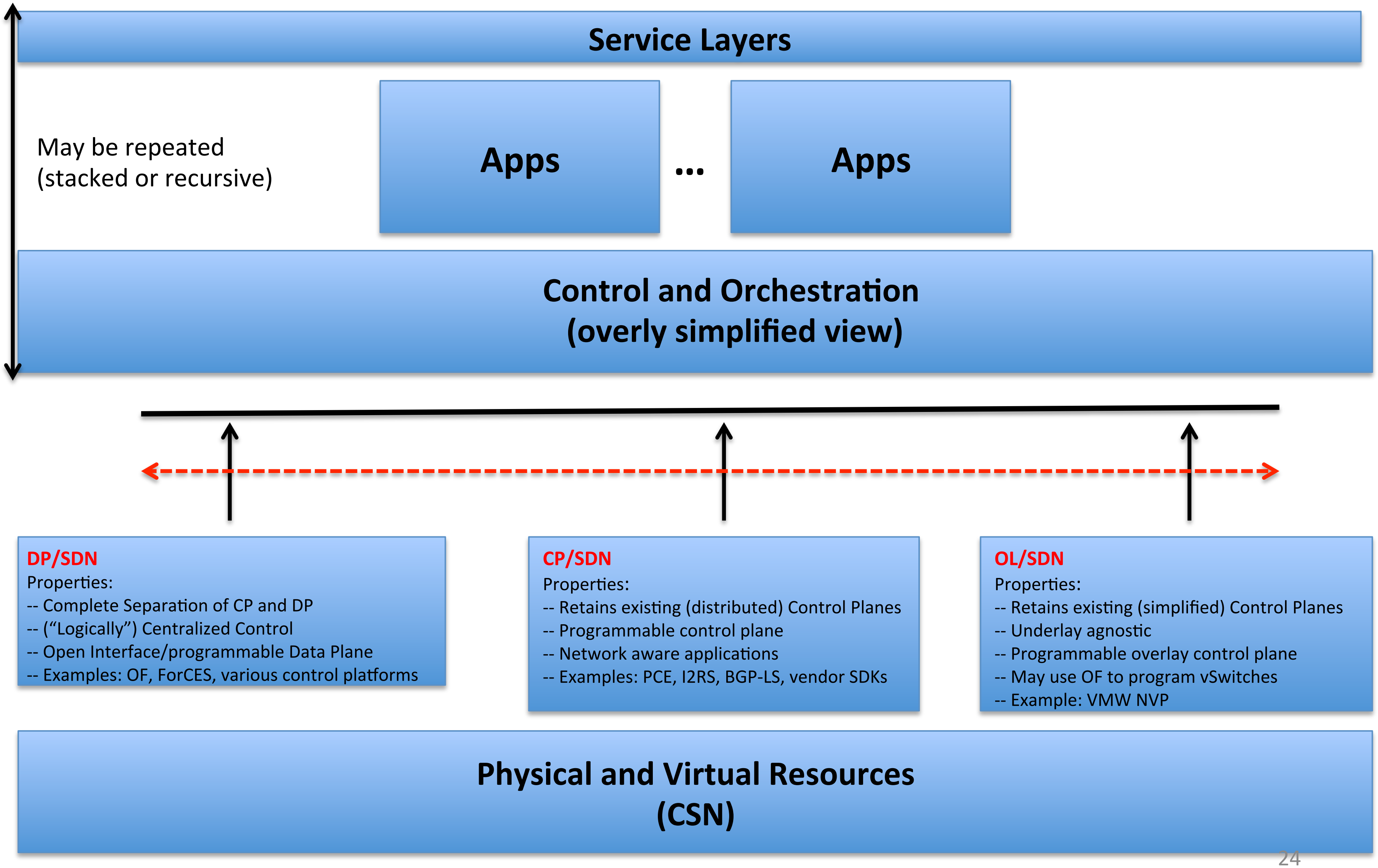


(a) Packets are matched against multiple tables in the pipeline

- Why this design?
  - Combinatoric explosion(s) s/a routes*policies in single table
- However, intractable complexity: *O(*n!) paths through tables of a *single switch*
  - $c \approx a^{(2^l)} + \alpha$
  - where a = number of actions in a given table, l = width of match field, and
  - α all the factors I didn't consider (e.g., table size, function, group tables, meter tables, …)
- Too complex/brittle
  - Algorithmic complexity
  - What is a flow?
  - Not naturally implementable on ASIC h/w
  - Breaks new reasoning systems (e.g., frenetic)
  - No fixes for lossy abstractions
  - Architectural questions

So question: Is the flow-based abstraction "right" for general network programmability?

The SDN Design Space


Service Layers
May be repeated (stacked or recursive)
Apps
…
Apps
Control and Orchestration (overly simplified view)
Physical and Virtual Resources (CSN)

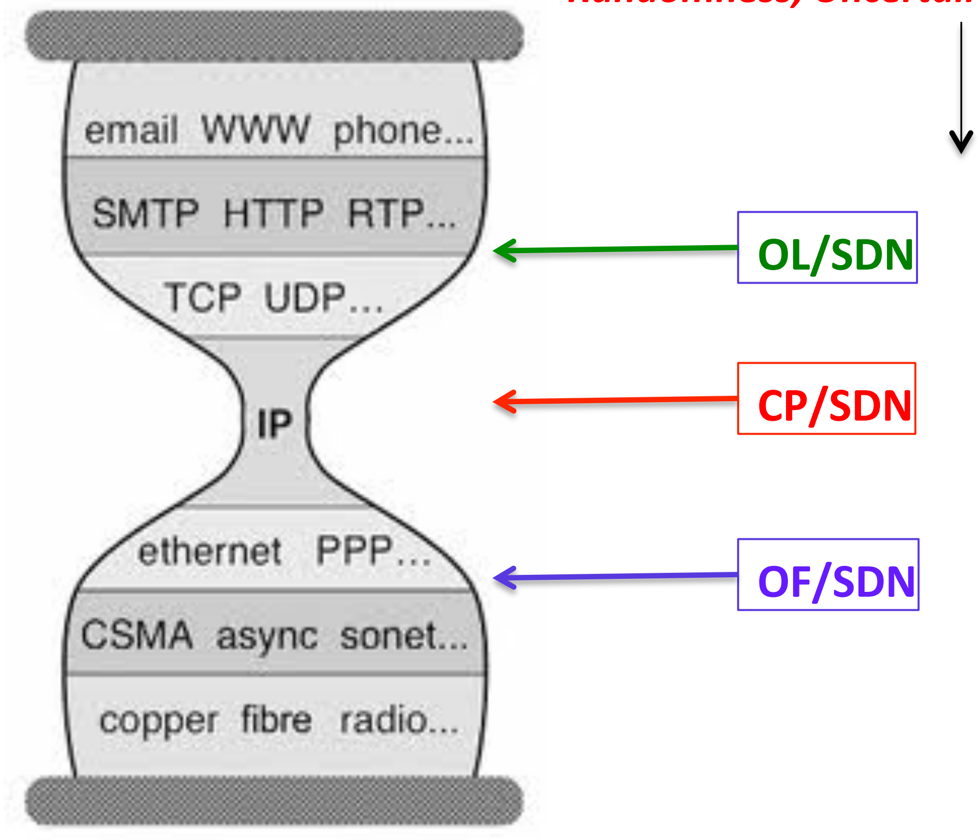
DP/SDN
Properties:
-- Complete Separation of CP and DP
-- ("Logically") Centralized Control
-- Open Interface/programmable Data Plane
-- Examples: OF, ForCES, various control platforms

CP/SDN
Properties:
-- Retains existing (distributed) Control Planes
-- Programmable control plane
-- Network aware applications
-- Examples: PCE, I2RS, BGP-LS, vendor SDKs

OL/SDN
Properties:
-- Retains existing (simplified) Control Planes
-- Underlay agnostic
-- Programmable overlay control plane
-- May use OF to program vSwitches
-- Example: VMW NVP

Putting it all Together

Open Loop Control + s/w + Moore's Law →
Randomness, Uncertainty, and Volatility



- OF/SDN proposes a new architectural waist (not exactly sure where)
- CP/SDN makes existing control planes programmable
- OL/SDN is an application *from the perspective of the Internet's waist*

Summary/Where to from Here?

- First, note that SDN doesn't do anything fundamentally different
  - Moves architectural features (and maybe complexity) around in the design space

- Be conservative with the narrow waist -- *constraints that deconstrain*
  - We're pretty good at this
  - Reuse parts where possible (we're also pretty good at this; traceroute a canonical example)

- Expect uncertainty and volatility from above
  - Inherent in software, and importantly, in acceleration
    - We know the network is RYF-complex so we know that for H(p,x), the "harm" function, $d^2H(p,x)/dx^2 \neq 0$
    - When you architect for robustness, understand what fragilities have been created
  - → Software (SDN or http://spotcloud.com or …) is inherently non-linear, volatile, and uncertain
    - We need to learn to live with/benefit from the non-linear, random, uncertain

- DevOps
  - We already have some components (Puppet, Chef, rancid, …)

- Develop our understanding bottom up (by "tinkering")
  - Actually an "Internet principle". We learn incrementally…
  - Avoid the top-down (in epistemology, science, engineering,…)
  - Bottom-up v. top-down innovation cycles – cf Curtis Carlson

- Design future software ecosystems to benefit from variability and uncertainty rather than trying to engineer it out (as shielding these systems from the random may actually cause harm)
  - For example, design in ***degeneracy*** -- i.e., "ability of structurally different elements of a system to perform the same function". In other words, design in partial functional overlap of elements capable of non-rigid, flexible and versatile functionality. This allows for evolution *plus* redundancy. Contrast m:n *redundancy* (i.e., we do just the opposite).

Q&A

Thanks!