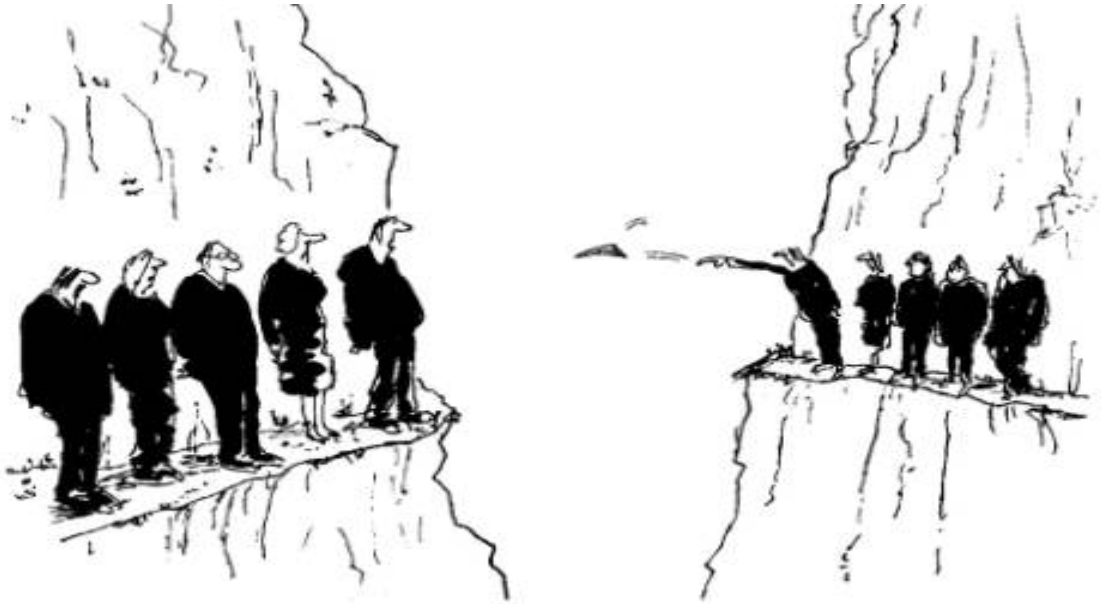
Complexity, Tradeoffs, Architecture, and the Evolution of DevOps



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

Agenda

- Goals for this Talk
- What is Complexity and Why is it Hidden
  - Robustness, Fragility, and Complexity
- Tradeoffs?
- The Architecture of Complex Systems
  - Universal Architectural Principles
- The Evolution DevOps
- A Few Conclusions and Q&A if we have time

Danger Will Robinson!!!

Standard Disclaimer

This talk might be controversial/provocative (and perhaps a bit "sciencey")

Goals for this Talk

- Characterize the essential features of *complexity*
  - in both technological and biological systems
- Understand the fundamental *tradeoffs*
  - that are made in these systems
- Understand the *universal architectural features*
  - found in both technological and biological networks
- *Evolving DevOps*
  - means bridging the DevOps and theory communities

Goals for this Talk

Said another way:

To open up our thinking about what the essential architectural features of our network are, how these features combine to provide robustness (and its dual, fragility), and how the universal architectural features that we find in both technological and biological networks effect Internet robustness, scalability and evolvability.

Agenda

- ~~Goals for this Talk~~
- What is Complexity and Why is it Hidden
  - Robustness, Fragility, and Complexity
- Tradeoffs?
- The Architecture of Complex Systems
  - Universal Architectural Principles
- The Evolution of DevOps
- A Few Conclusions and Q&A if we have time

What is Complexity (and why is it Hidden)?

- So what is Complexity?
  - Complexity is hidden *structure* that arises in systems
  - Its purpose is to create *robustness* to environmental and component uncertainty

- Why hidden?
  - Anti-lock/anti-skid brakes, packet loss, OS kernels, power grids, SDN controllers, lunar landing systems, …
  - You don't notice they are there…until they fail
    - often catastrophically
  - The hidden nature of complexity is a fundamental property of systems
    - and derives from universal architectural principles of complex systems
    - we will see examples of this in just a minute

- Ok, then what is Robustness?

Robustness is a Generalized System Feature

- **Scalability** is *robustness* to changes to the size and complexity of a system as a whole
- **Evolvability** is *robustness* of lineages to changes on various (usually 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
- Of course, these are the features we're seeking from the network

A Bit More Formally

- ***Robustness*** *is the preservation of a certain property in the presence of uncertainty in components or the environment*
  - Obviously a core Internet design principle
  - Systems Biology: Biological systems are designed such that their important functions are insensitive to the naturally occurring variations in their parameters.
    - Limits the number of designs that can actually work in the real environment
    - Exact adaptation in bacteria chemotaxis

- ***Fragility*** is the opposite of robustness
  - Another way to think about fragility
    - Technical: You are fragile if you depend on 2nd order effects (acceleration) and the "harm" curve is concave
  - A little more on this in the next few slides…

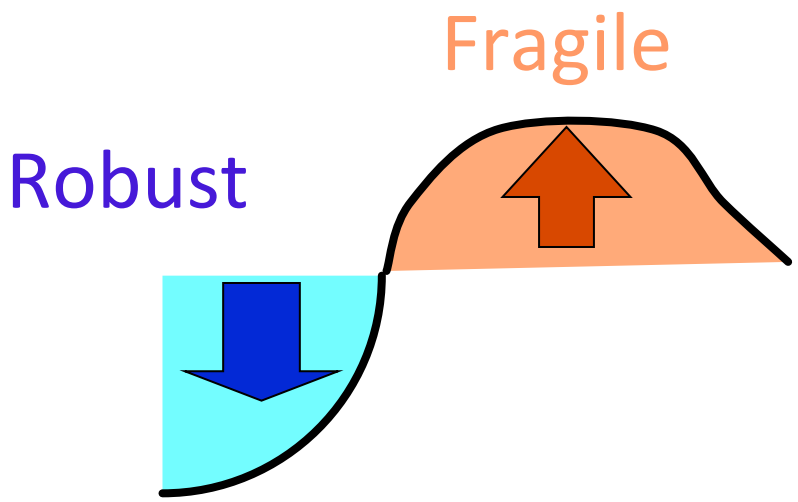
- A system can have a *property* that is *robust* to one set of perturbations and yet *fragile* for a *different property* and/or perturbation → the system is ***Robust Yet Fragile***
  - Or the system may collapse if it experiences perturbations above a certain threshold (K-fragile)

- For 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
  - VRRP, ISSU, HA, TE, …
  - Complexity/Robustness Spirals
  - Implications for Carrier Grade components?

Robust Yet Fragile?

[a system] can have
[a property] that is ***robust*** to
[a set of perturbations]

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



Recent results suggest that the RYF tradeoff is a hard tradeoff that cannot be overcome[1]

This is profound: If you create robustness somewhere you will create fragility somewhere else…

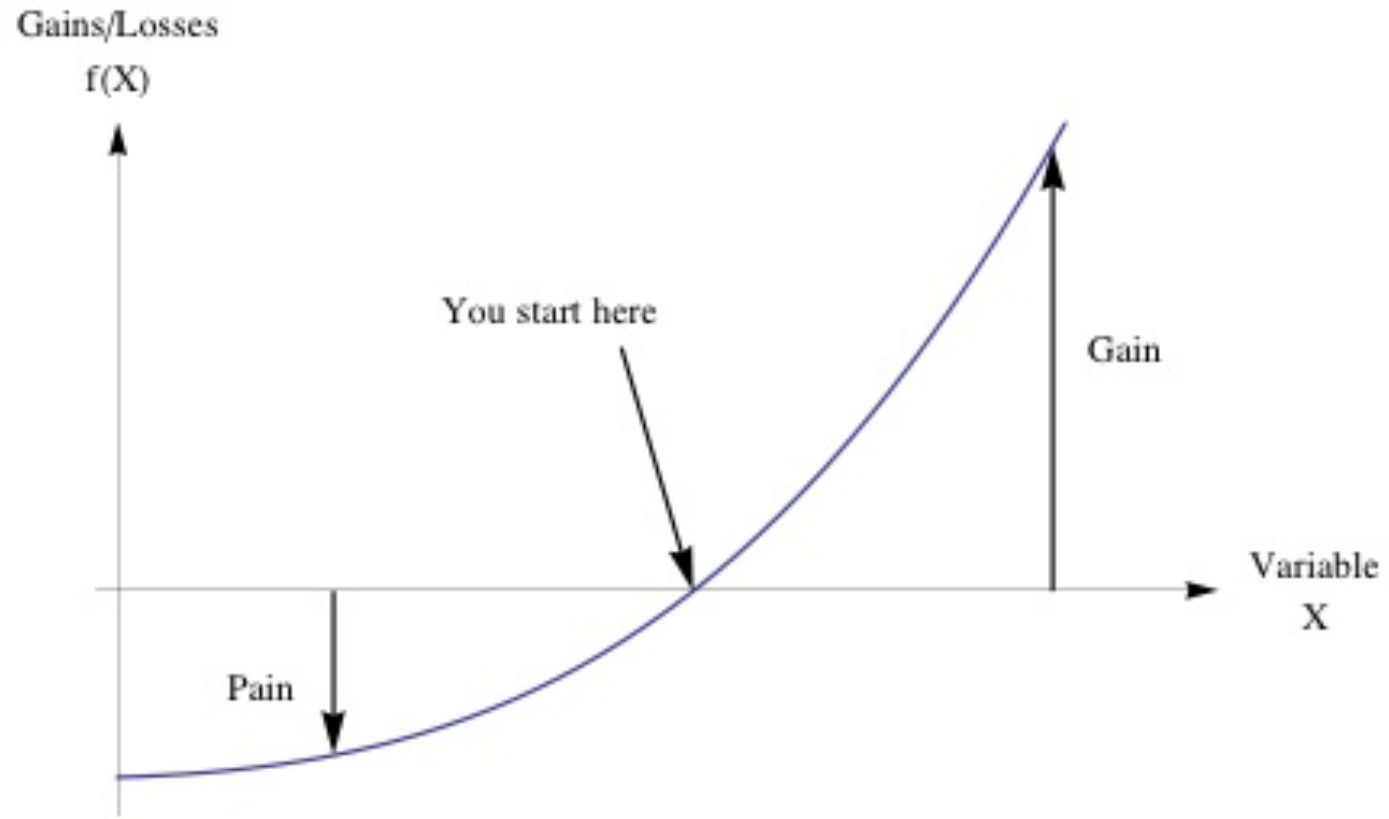
DevOps (along with most engineering disciplines) does not currently account for this effect

Harm Function: Concave → Fragile, Convex → Robust

[1] See Marie E. Csete and John C. Doyle, "Reverse Engineering of Biological Complexity", http://www.cds.caltech.edu/~doyle/wiki/images/0/05/ScienceOnlinePDF.pdf

Another way to think about Robustness

(*Convex Optionality* aka No Pain, No Gain)



If you are hurt much less by error/variation than you stand to gain…

Graphic courtesy Nassim Taleb

Interestingly, Fragility and Scaling are Related

- 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***

- → *A big event hurts non-linearly more than the sum of small events*

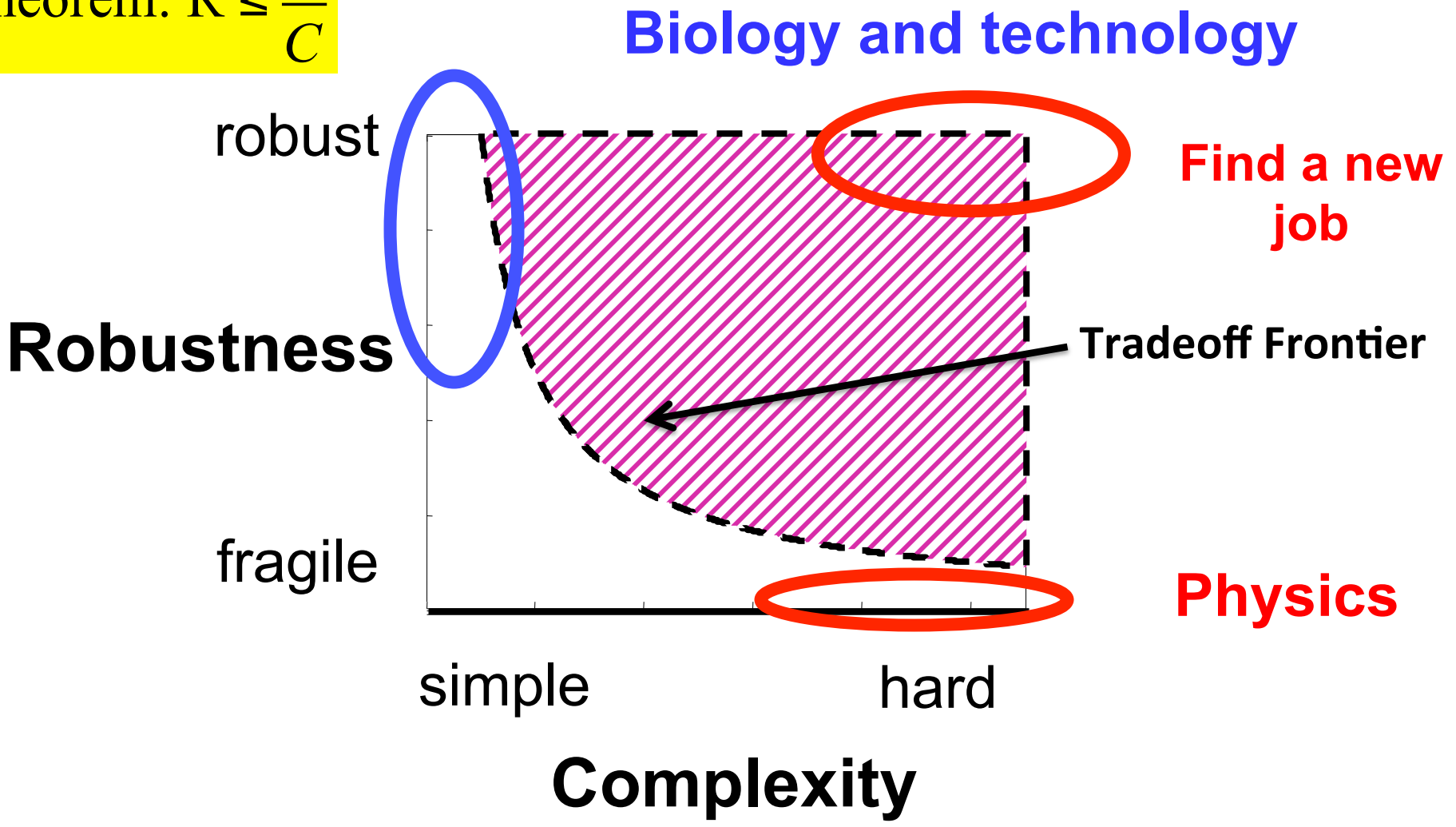
- For example, a coffee cup on a table suffers non-linearly more from large deviations (*H(p, nz)*) than from the cumulative effect of smaller events (*nH(p,z)*)
  - So the cup is damaged far more by *tail events* than those within a few σ's of the mean
  - Sensitivity to tail events → RYF
  - Too theoretical? 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

- So 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, … → non-linear damage

Coffee cup example courtesy Nassim Taleb. See http://www.fooledbyrandomness.com

Its All About (RYF) Tradeoffs

Theorem: $R \leq \frac{1}{C}$



BTW, RYF Behavior is Everywhere

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. That is, they are RYF-Complex
- Often involving hijacking/exploiting the same mechanism
  - We've certainly seen this in the Internet space (consider DDOS of various varieties)
- These are hard constraints (i.e., RYF behavior is conserved)

Robust

☺ Metabolism
☺ Regenerati
☺ Healing wo

Fragile

☹ Obesity, diabetes
☹ Fat accumulation
☹ Insulin resistance
☹ Proliferation
☹ Inflammation

Same Mechanisms

- Fragility ← Hijacking, si… tended…
  - DDOS, reflectio…
- Of mechanis… robustness
- Comple… robust/fragile tradeoffs
- M… e constraints (“conservation laws”)

Current DevOps approaches characterized by trial and error and/or ad-hoc techniques

Both

Accident or necessity?

Summary: Understanding RYF is ***The*** Challenge

- It turns out that managing/understanding RYF behavior is ***the most essential challenge*** in technology, society, politics, ecosystems, medicine, etc. This means…
  - Understanding *Universal Architectural Principles*
    - Look ahead: Layering, Bowties/Hourglasses, Constraints that Deconstrain
  - Managing spiraling complexity/fragility
  - Not predicting what is likely or typical
    - But rather understanding what is catastrophic
    - or in Taleb's terminology, that which is fat tailed

- BTW, it is much easier to create the robust features than it is to prevent the fragilities
  - And as I mentioned, there are poorly understood "conservation laws" at work[1]

- Bottom Line
  - ***Understanding RYF behavior means understanding architecture and the hidden nature of complexity***

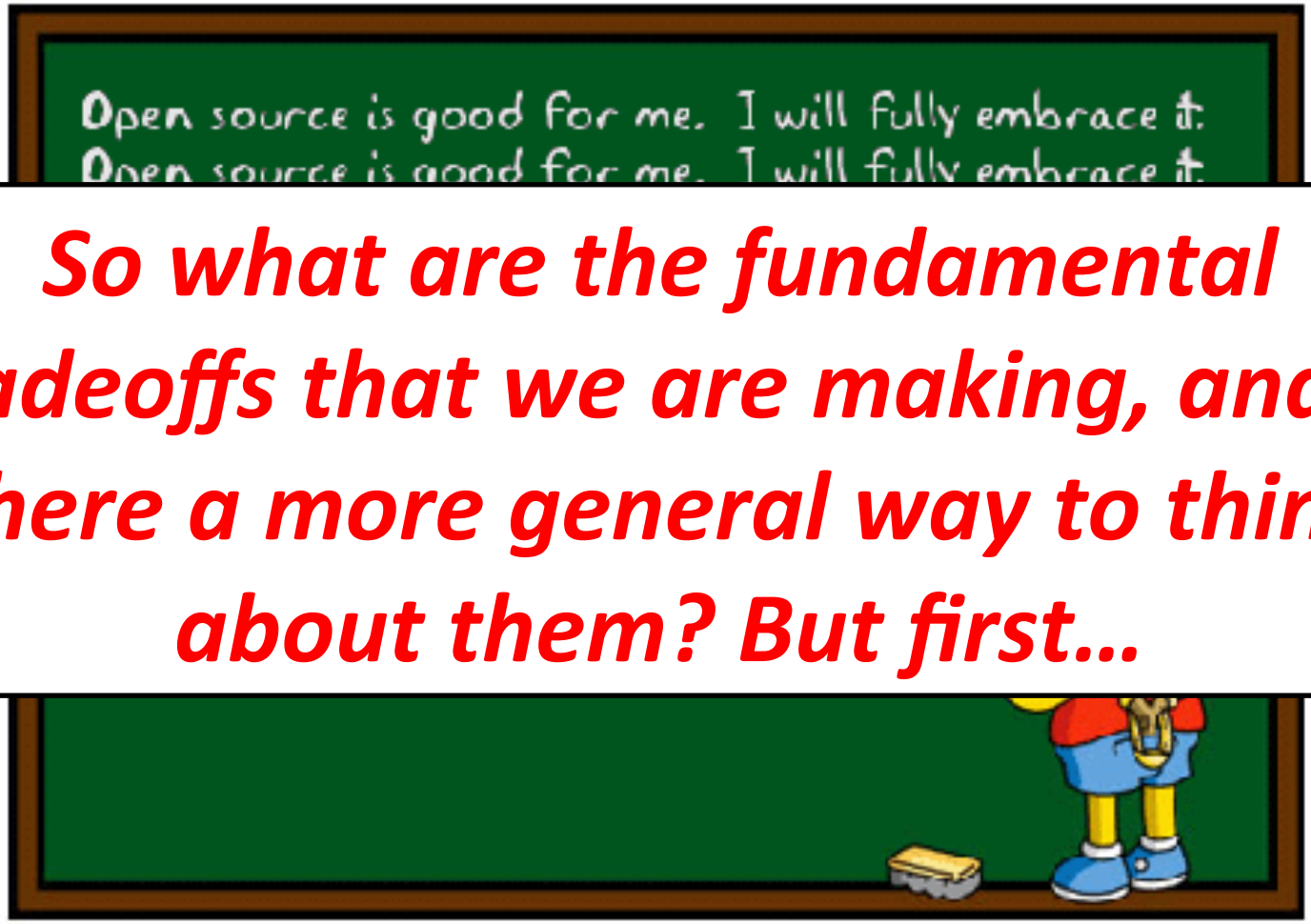
[1] See Marie E. Csete and John C. Doyle, "Reverse Engineering of Biological Complexity", http://www.cds.caltech.edu/~doyle/wiki/images/0/05/ScienceOnlinePDF.pdf

Agenda

- ~~Goals for this Talk~~
- ~~What is Complexity and Why is it Hidden~~
  - ~~Robustness, Fragility, and Complexity~~
- Tradeoffs?
- The Architecture of Complex Systems
  - Universal Architectural Principles
- The Evolution of DevOps
- A Few Conclusions and Q&A if we have time

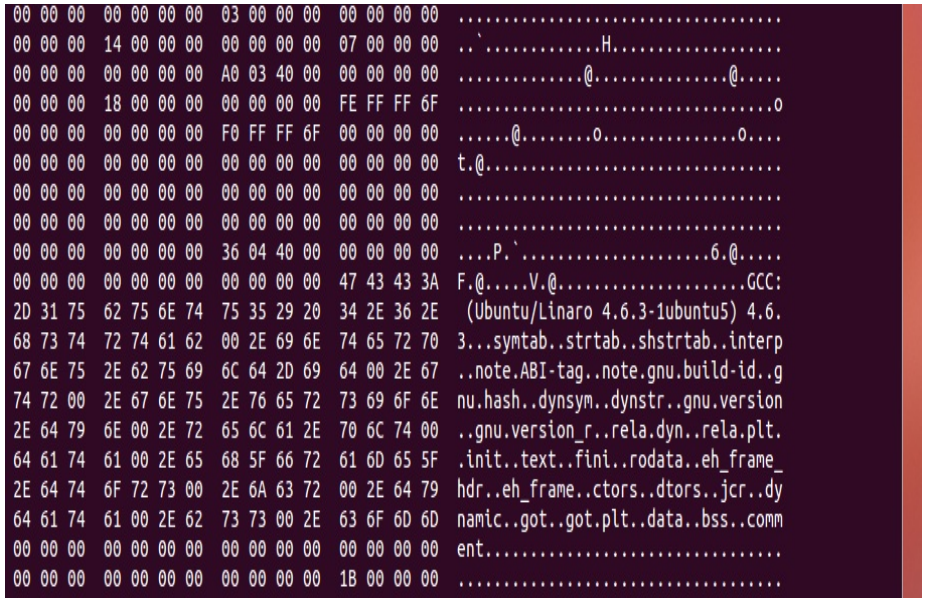
What is our Architectu
(and what tradeoffs are

ier Grade,



So what are the fundamental tradeoffs that we are making, and is there a more general way to think about them? But first…

First, what tradeoffs do we see everyday?



```
00 00 00  00 00 00 00  03 00 00 00  00 00 00 00  ................................
00 00 00  14 00 00 00  00 00 00 00  07 00 00 00  ..`.............H...............
00 00 00  00 00 00 00  A0 03 40 00  00 00 00 00  ..............@...............@.....
00 00 00  18 00 00 00  00 00 00 00  FE FF FF 6F  ................................o
00 00 00  00 00 00 00  F0 FF FF 6F  00 00 00 00  ......@........o..............o....
00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  t.@.............................
00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................................
00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................................
00 00 00  00 00 00 00  36 04 40 00  00 00 00 00  ....P.`.....................6.@.....
00 00 00  00 00 00 00  00 00 00 00  47 43 43 3A  F.@.....V.@.....................GCC:
2D 31 75  62 75 6E 74  75 35 29 20  34 2E 36 2E   (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.
68 73 74  72 74 61 62  00 2E 69 6E  74 65 72 70  3...symtab..strtab..shstrtab..interp
67 6E 75  2E 62 75 69  6C 64 2D 69  64 00 2E 67  ..note.ABI-tag..note.gnu.build-id..g
74 72 00  2E 67 6E 75  2E 76 65 72  73 69 6F 6E  nu.hash..dynsym..dynstr..gnu.version
2E 64 79  6E 00 2E 72  65 6C 61 2E  70 6C 74 00  ..gnu.version_r..rela.dyn..rela.plt.
64 61 74  61 00 2E 65  68 5F 66 72  61 6D 65 5F  .init..text..fini..rodata..eh_frame_
2E 64 74  6F 72 73 00  2E 6A 63 72  00 2E 64 79  hdr..eh_frame..ctors..dtors..jcr..dy
64 61 74  61 00 2E 62  73 73 00 2E  63 6F 6D 6D  namic..got..got.plt..data..bss..comm
00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ent.............................
00 00 00  00 00 00 00  00 00 00 00  1B 00 00 00  ................................
```

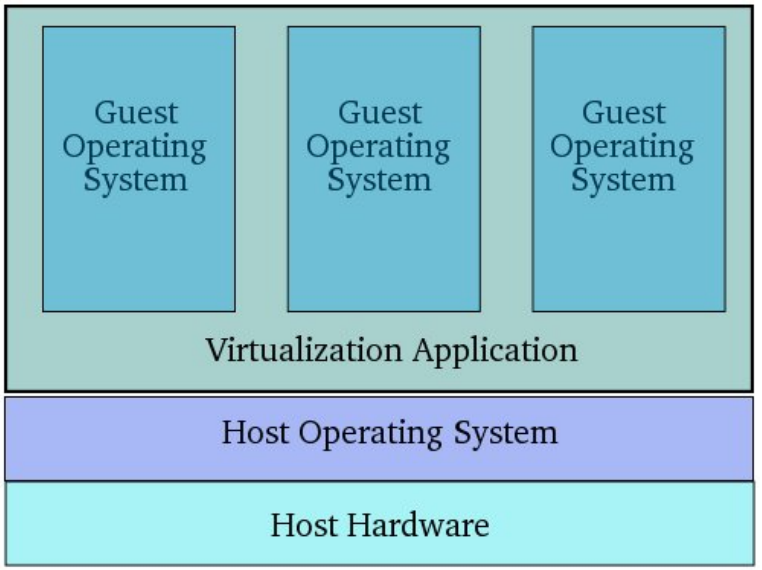
```
def add5(x):
   return x+5

def dotwrite(ast):
   nodename = getNodename()
   label=symbol.sym_name.get(int(ast[0]),ast[0])
   print ' %s [label="%s' % (nodename, label),
   if isinstance(ast[1], str):
      if ast[1].strip():
         print '= %s"];' % ast[1]
      else:
         print '"]'
   else:
      print '"];'
      children = []
      for n, child in enumerate(ast[1:]):
         children.append(dotwrite(child))
      print ' %s -> {' % nodename,
      for name in children:
         print '%s' % name,
```

What tradeoffs are being made here?

Speed vs. flexibility?

Everyday Tradeoffs



How about here?

Speed vs. flexibility (bare metal vs. VM vs. container)?

Everyday Tradeoffs

What about here?

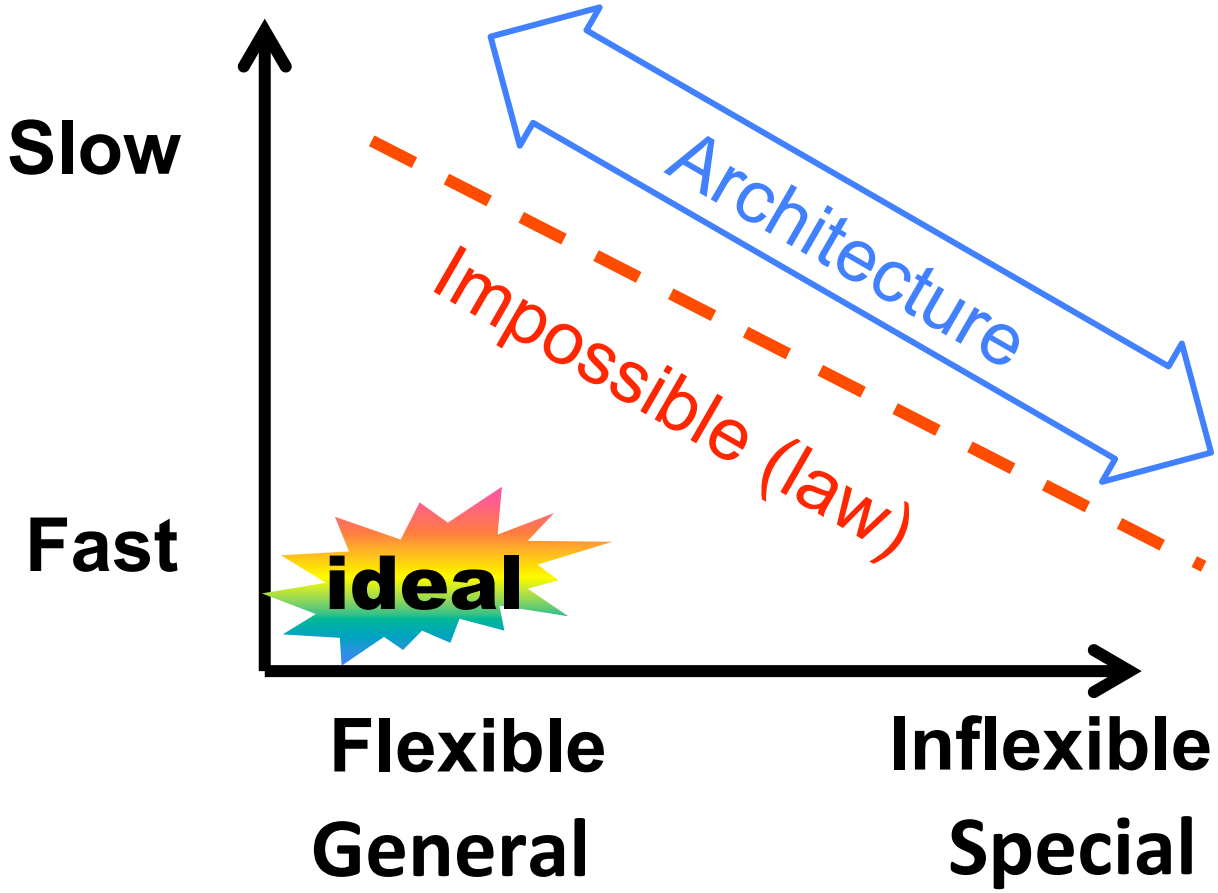
Summary: Common Tradeoffs

- Binary machine code vs. higher-level language
- VM vs. bare metal (vs. container)
- Fast path vs. slow path
- Hardware vs. software
- Formula-1 vs. minivan
- …
- But what are the essential features of these tradeoffs
  - What is fundamental in the tradeoff space
    - And are there “laws” governing these tradeoffs?
  - And how do they relate to RYF-complexity?

RYF tradeoffs are fundamental

Example: Computational Complexity

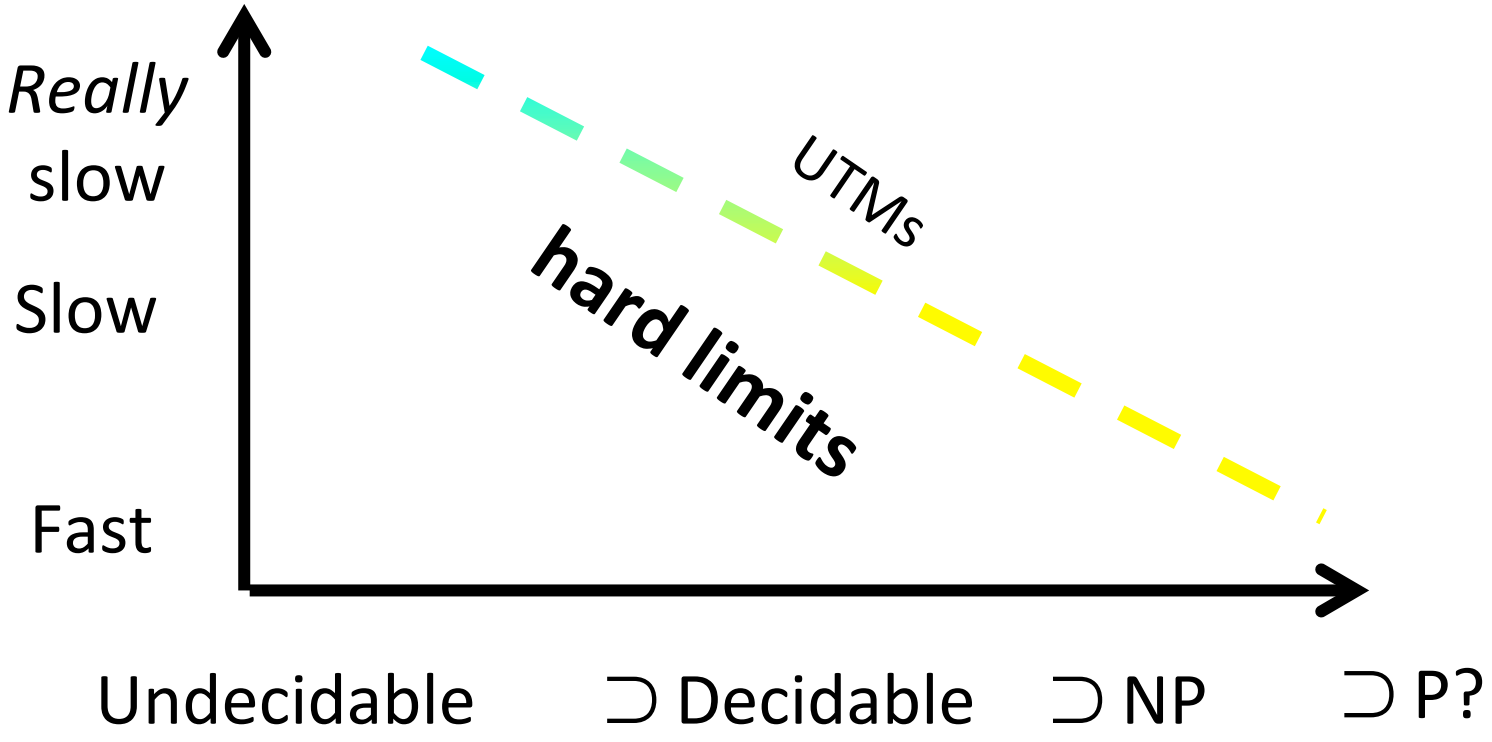
Layering, Formal Systems, Hard Tradeoffs



Architecture (*constraints that deconstrain*)

Original slide courtesy John Doyle

Drilling down a bit on the Computational Complexity Tradeoff Space



Flexible/
General

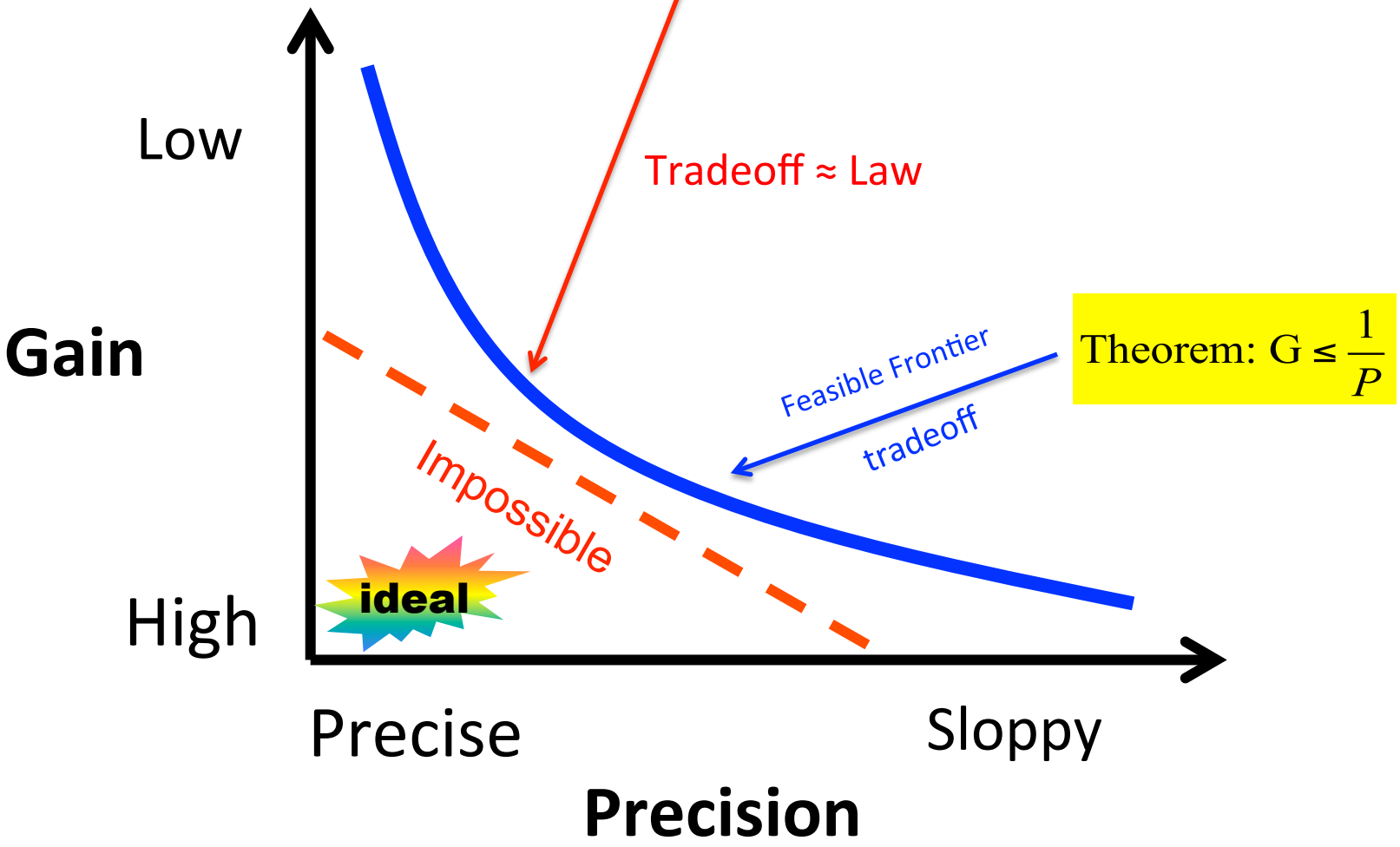
Inflexible/
Specific

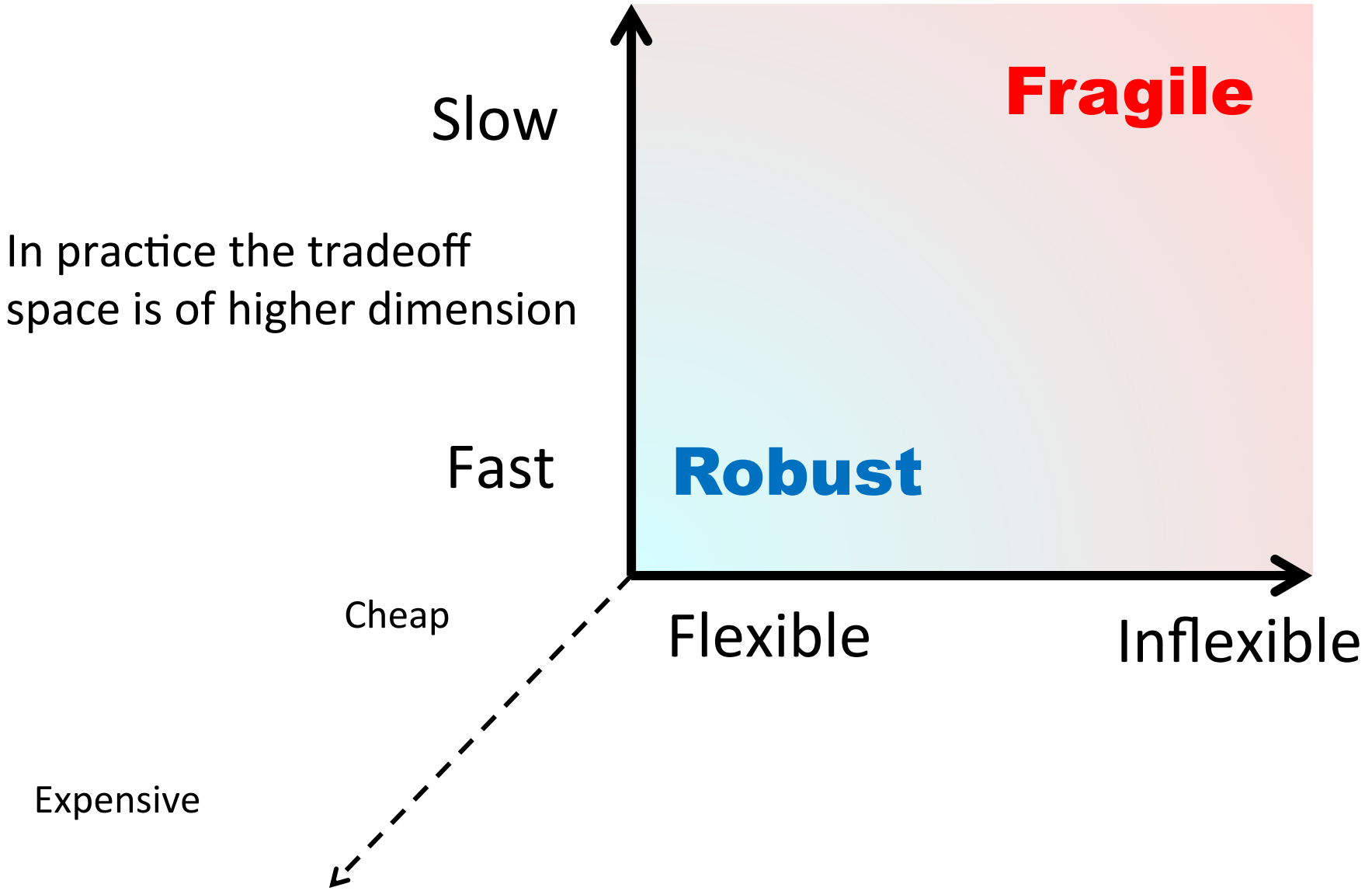
Another Example: Feedback Control Theory

(Gain/Sensitivity Tradeoff In Feedback Control)

$$\int_0^\infty \ln|S(i\omega)|d\omega = \int_0^\infty \ln\left|\frac{1}{1+L(i\omega)}\right| d\omega = \pi \sum Re(p_k) - \frac{\pi}{2}\lim_{s\to\infty} sL(s)$$

Bode Sensitivity Integral





Efficient vs **Robust**

Agenda

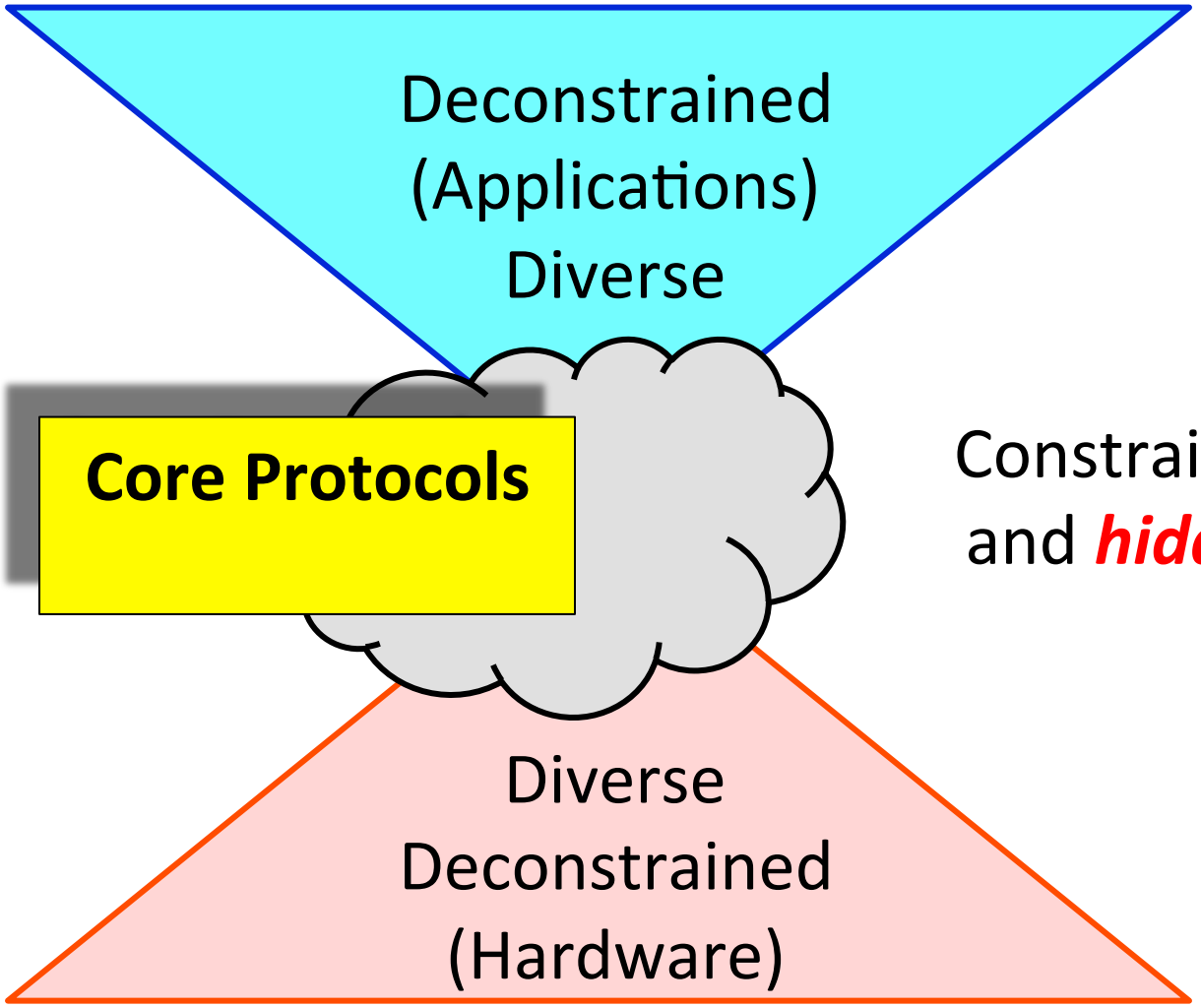
- ~~Goals for this Talk~~
- ~~What is Complexity and Why is it Hidden~~
  - ~~Robustness, Fragility, and Complexity~~
- ~~Tradeoffs?~~
- The Architecture of Complex Systems
  - Universal Architectural Principles
- The Evolution of DevOps
- A Few Conclusions and Q&A if we have time

The Architecture of Complex Systems

- What we have learned is that there are ***universal architectural building blocks*** found in systems that scale and are evolvable. These include
- **Architecture/Layering**
- **Laws, constraints, tradeoffs**
- Protocol Based Architectures
- Massively distributed with *robust* control loops
- Consequences
  - Hidden RYF Complexity
  - Hijacking, parasitism, predation

So What is the Basic Layered Architecture?

(hint: layering is *the* fundamental architectural feature)



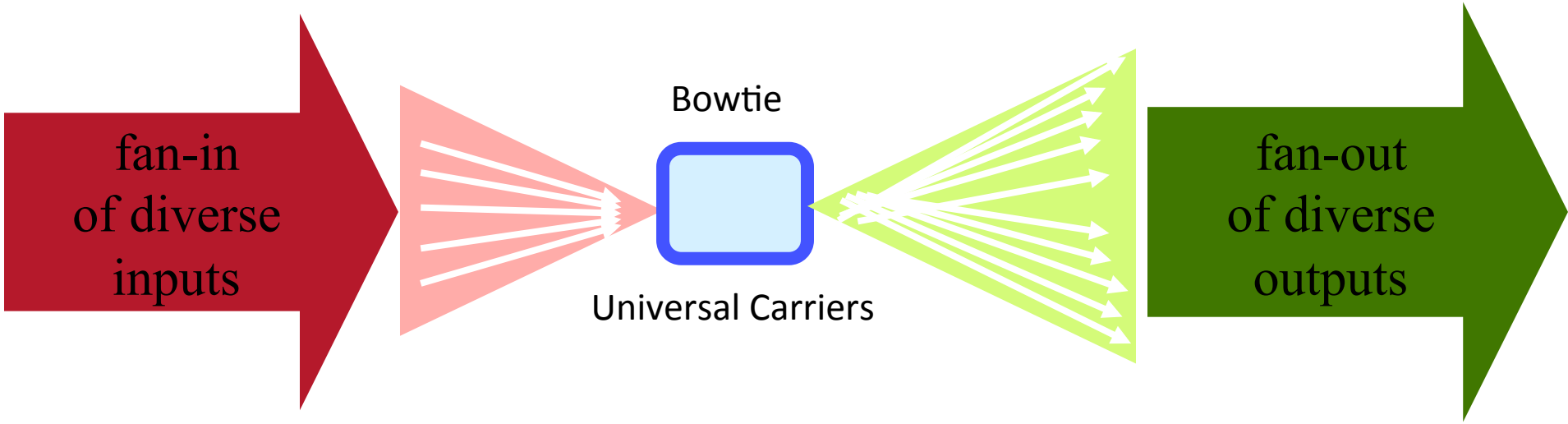
Apps

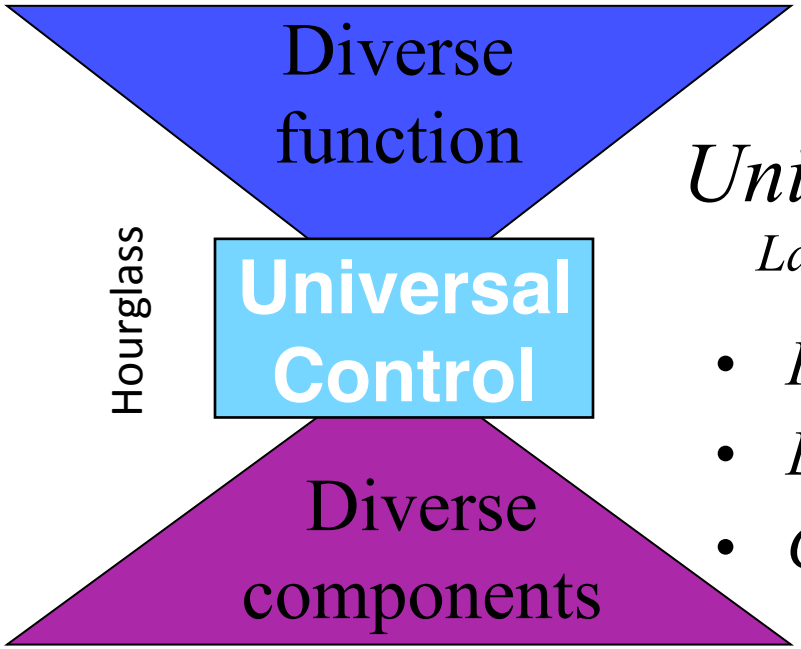
OS

HW

Slide courtesy John Doyle

Bowties, Hourglasses and Layering





Universal Architectural Principles

Layering As Optimization Decomposition View[1]

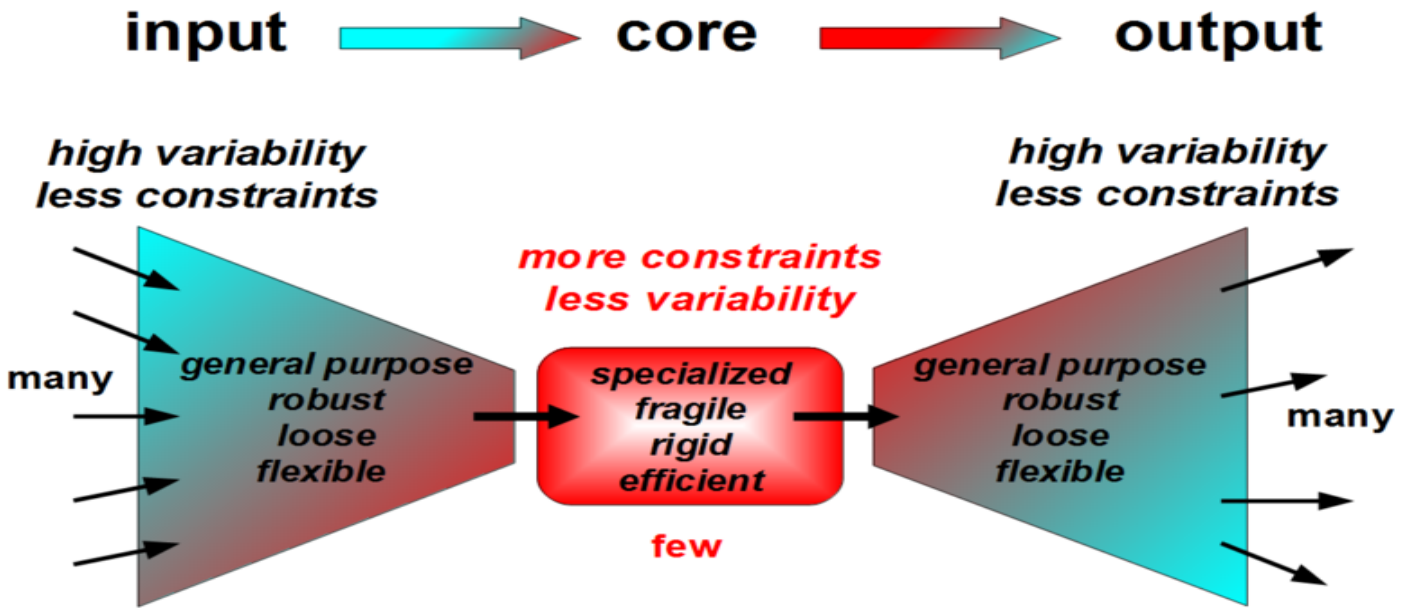
- *Hourglasses* for *layering of control*
- *Bowties* for *flows within layers*
- *Constraints that Deconstrain*

1 http://www.princeton.edu/~chiangm/layering.pdf

Bowties 101

Constraints that Deconstrain

Schematic of a "Layer"



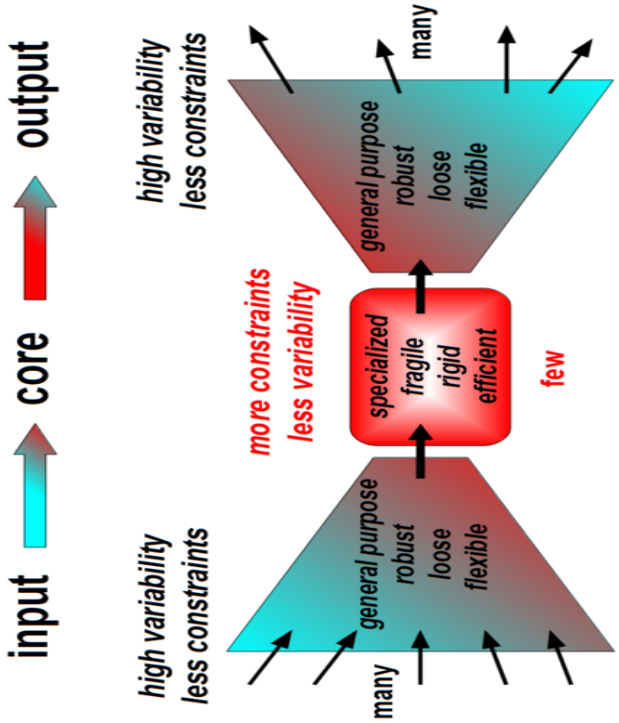
For example, the reactions and metabolites of core metabolism, e.g., Adenosine Triphosphate (*ATP) metabolism*, Krebs/Citric Acid Cycle, … form a "metabolic knot". That is, ATP is a ***Universal Carrier*** for cellular energy.

1. Processes L-1 information and/or raw material flows into a "standardized" format (the L+1 abstraction)
2. Provides plug-and-play modularity for the layer above
3. Provides robustness but at the same time fragile to attacks against/using the standardized interface

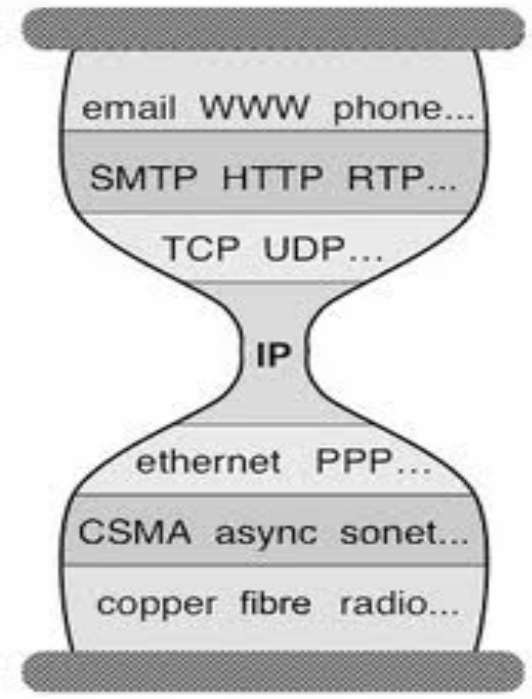
See Kirschner M., and Gerhart J., "Evolvability", Proc Natl Acad Sci USA , 95:8420–8427, 1998.

But Wait a Second

Anything Look Familiar?



Bowtie Architecture

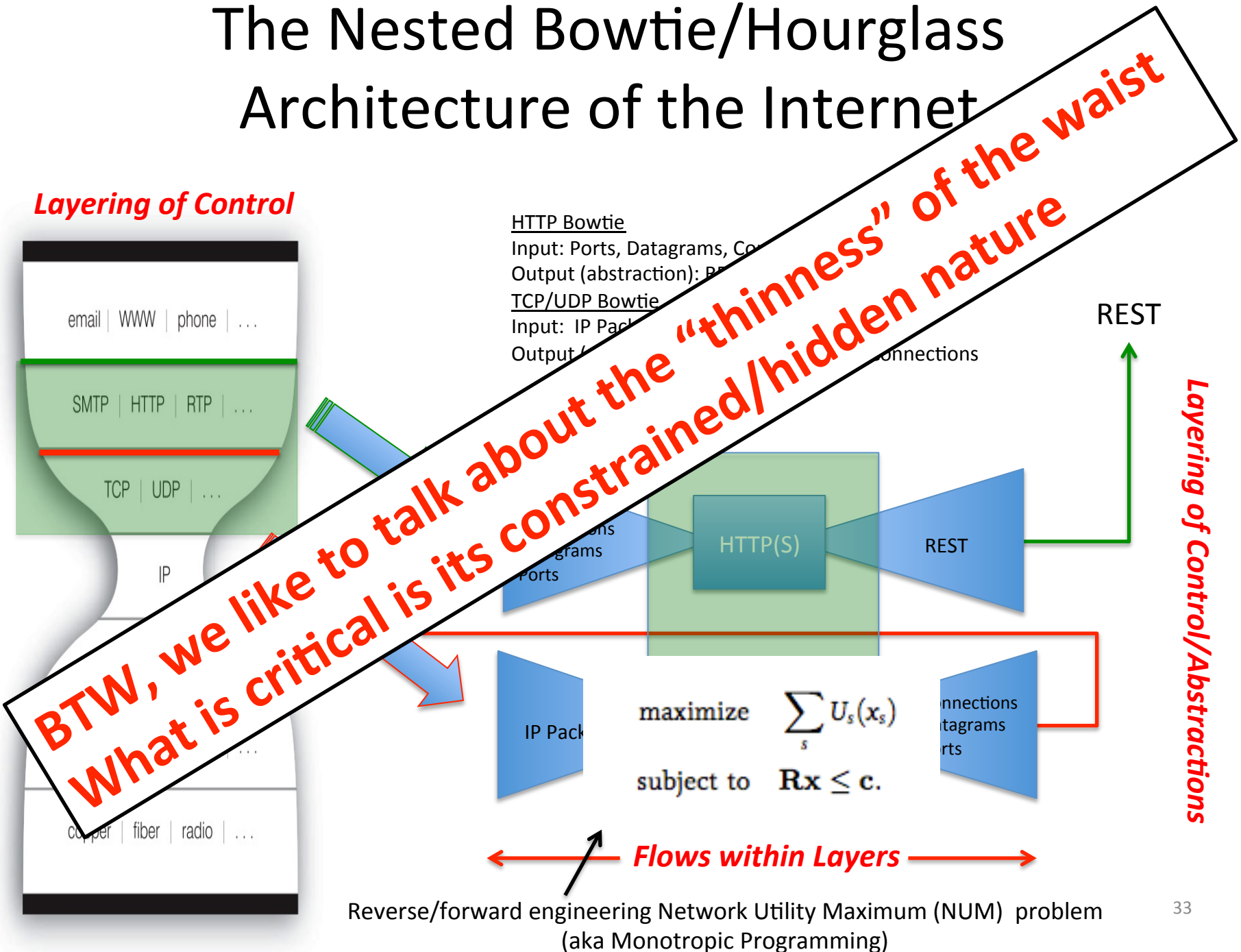


Hourglass Architecture

Comes down to whether you see layering as horizontal or vertical

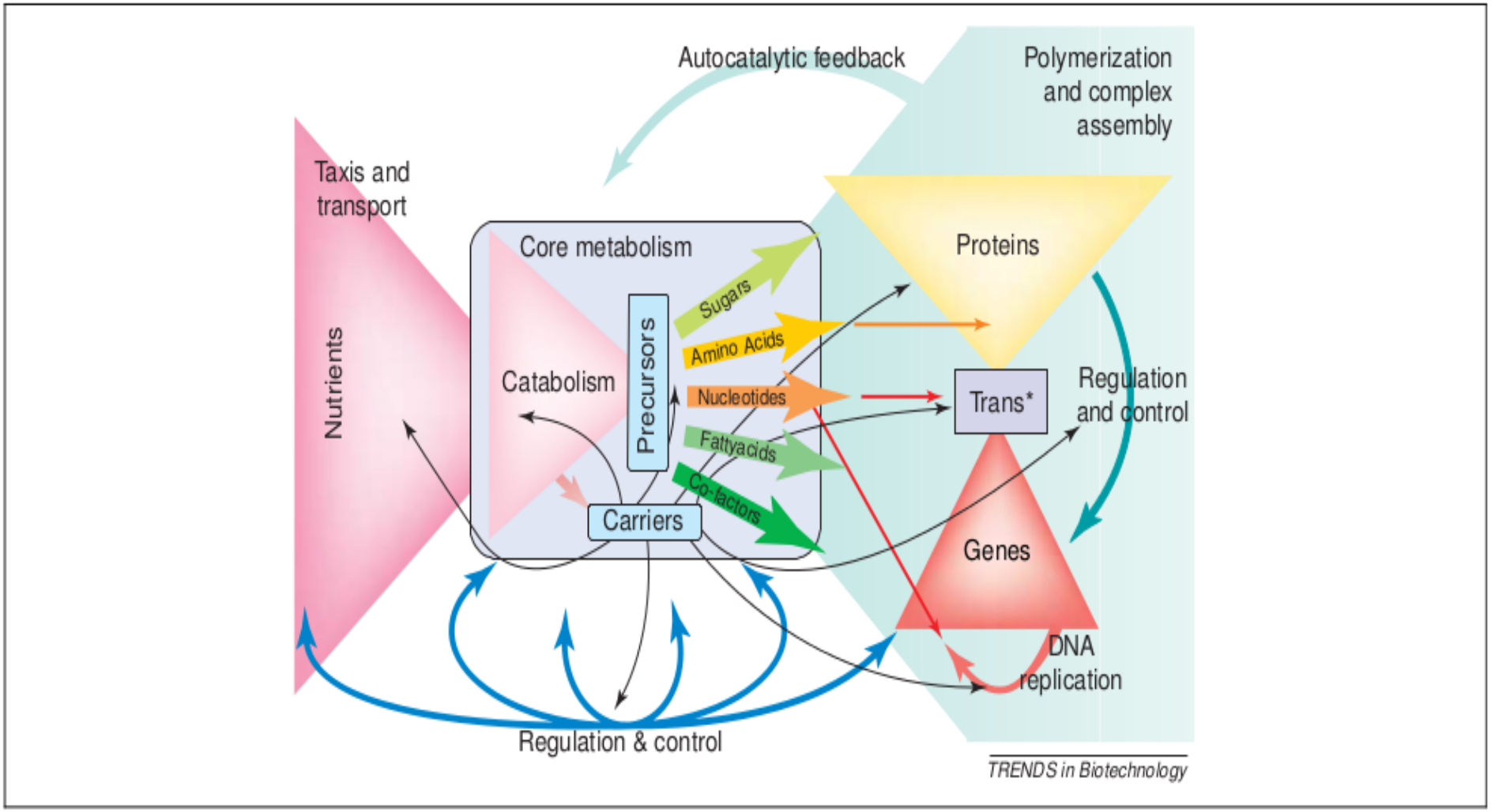
The *Protocol Hourglass* idea appears to have originated with Steve Deering. See Deering, S., "Watching the Waist of the Protocol Hourglass", IETF 51, 2001, http://www.iab.org/wp-content/IAB-uploads/2011/03/hourglass-london-ietf.pdf.

The Nested Bowtie/Hourglass Architecture of the Internet



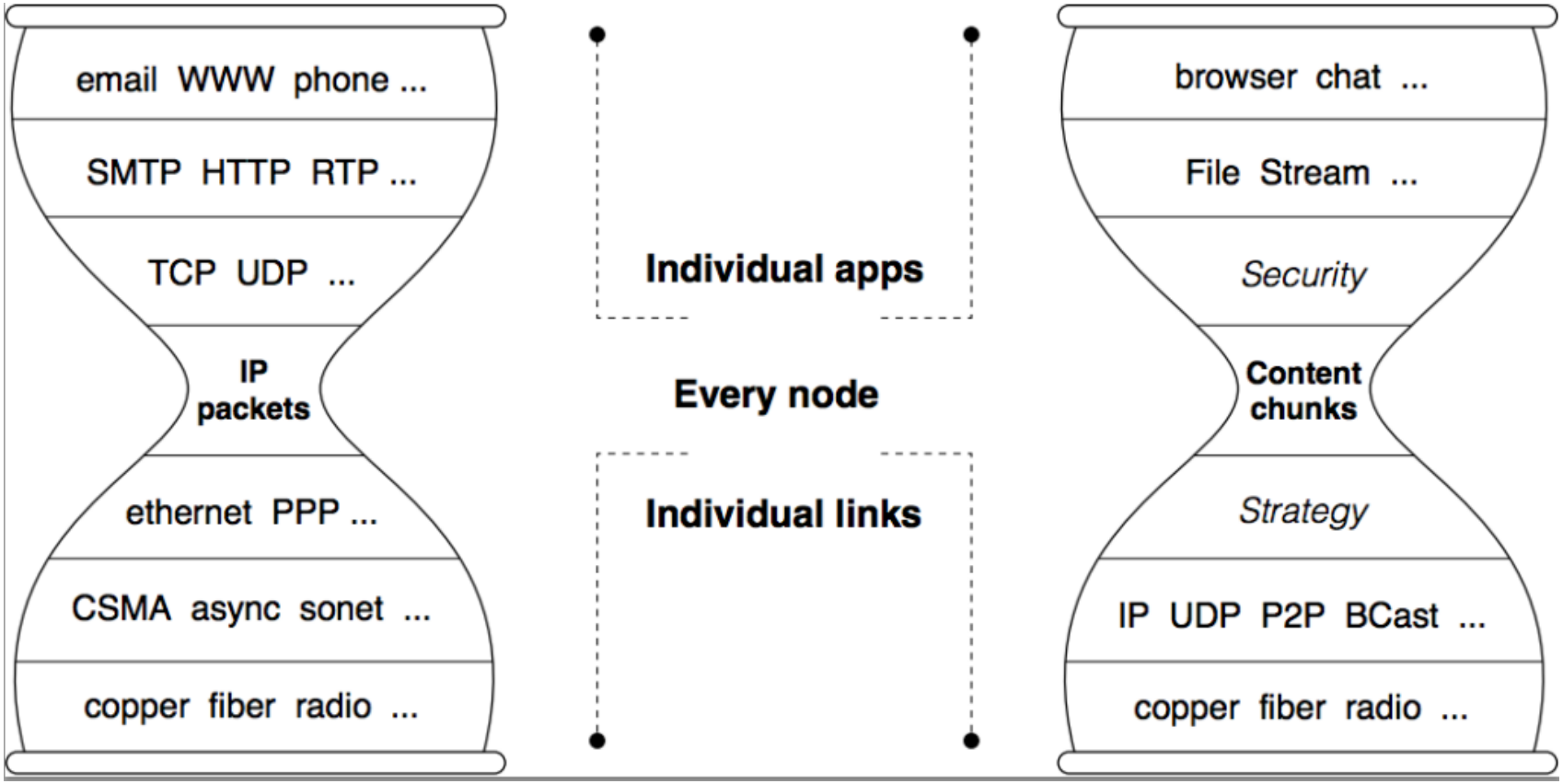
In Practice Things are More Complicated

The Nested Bowtie/Hourglass Architecture of Metabolism



See Csete, M. and J. Doyle, "Bowties, metabolism and disease", TRENDS in Biotechnology, Vol. 22 No. 9, Sept 2004

Named Data Networking Hourglass



See Named Data Networking, http://named-data.net/

Key Architectural Concept: Horizontal Trans

Deconstrained
(**Applications**)
Diverse

Horizontal App Transfer

OS

rained
ardware)

Horizontal HW Transfer

Apps

OS

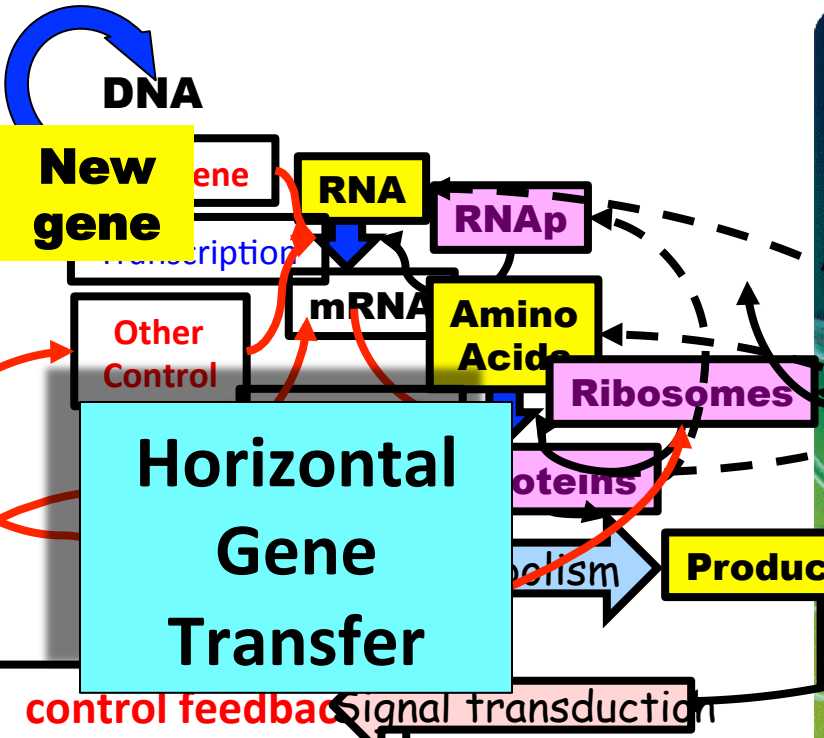
HW

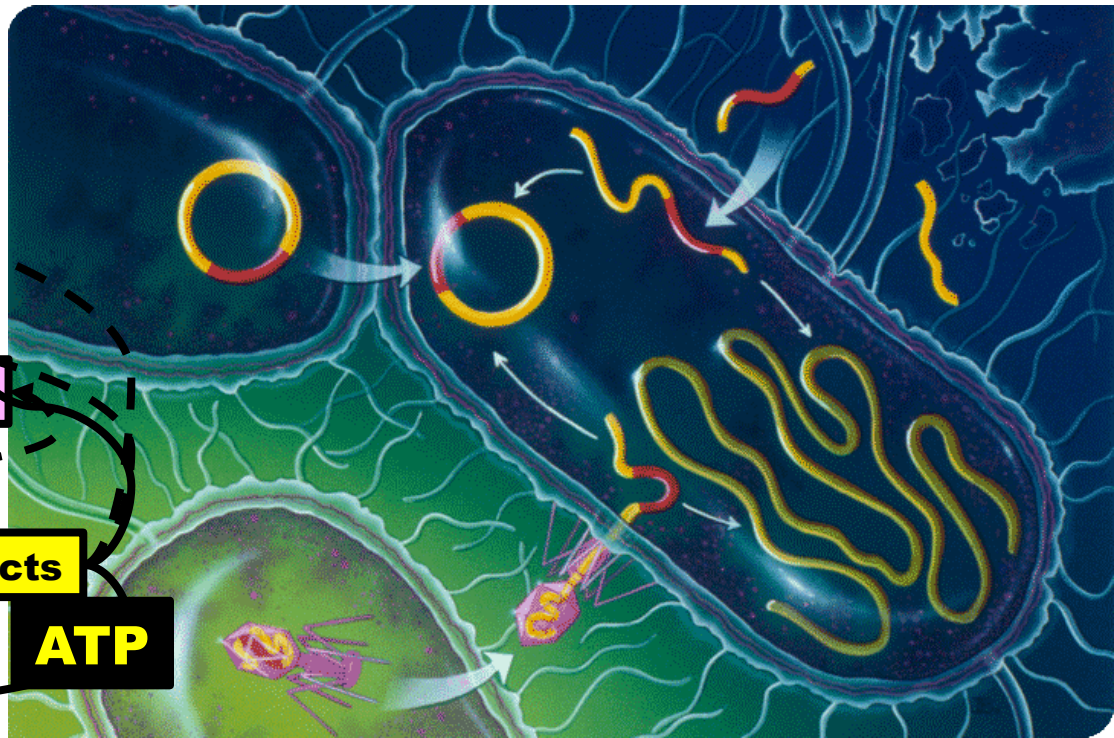
Layered Architectures make Robustness and Evolvability Compatible

Drilling Down: HGT and antibiotic resistance

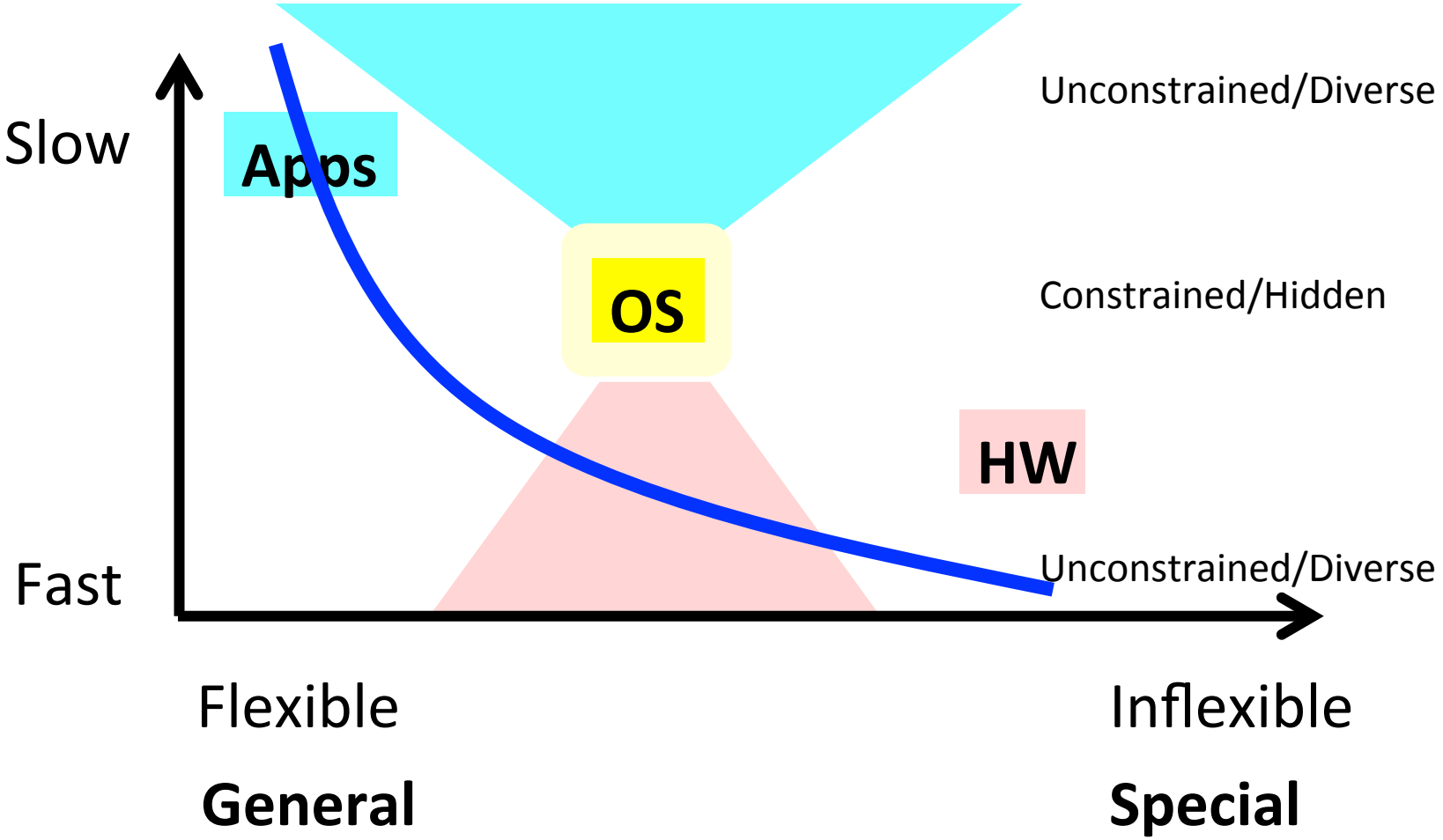
Sequence ~100 E Coli (*not* chosen randomly)

- ~ 4K genes per cell
- ~20K *different* genes in total (pangenome)
- ~ 1K universally shared genes
- ~ 300 essential (minimal) genes

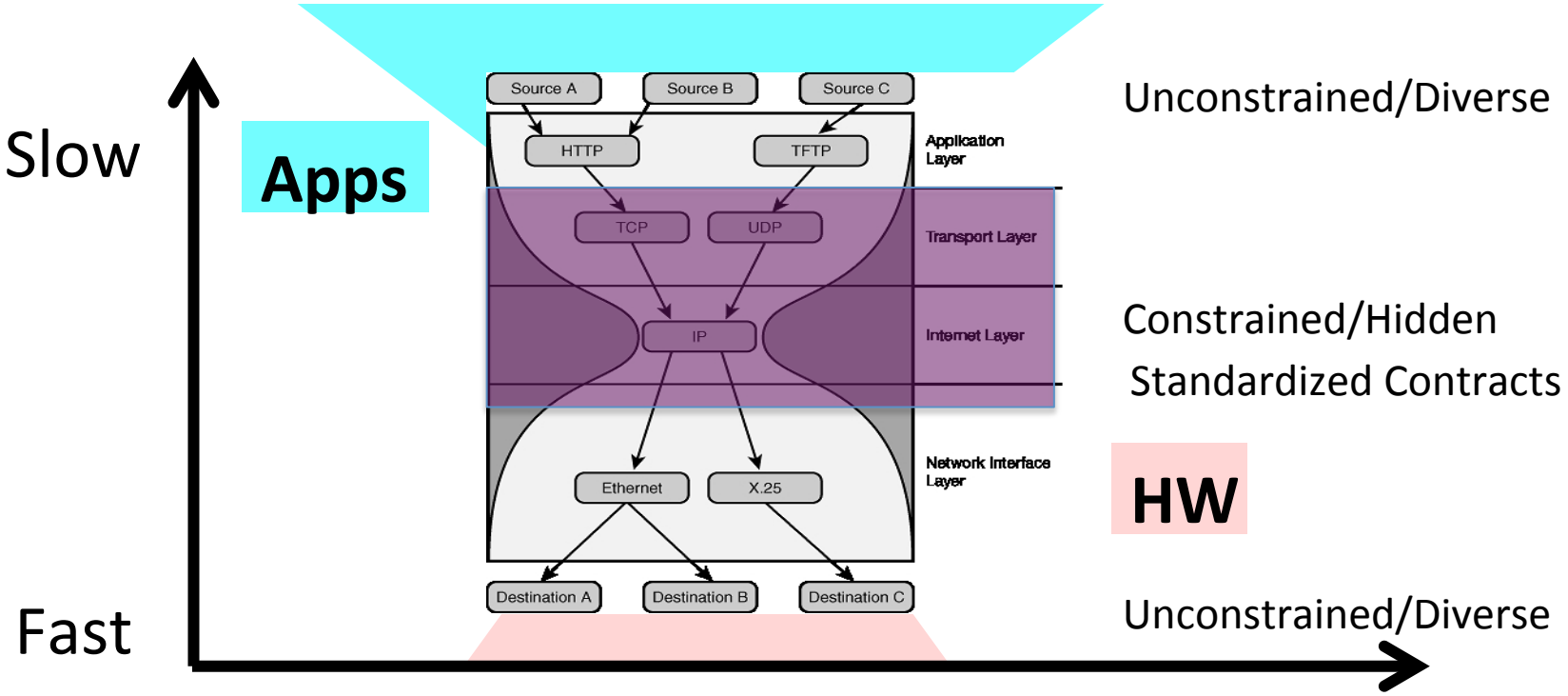




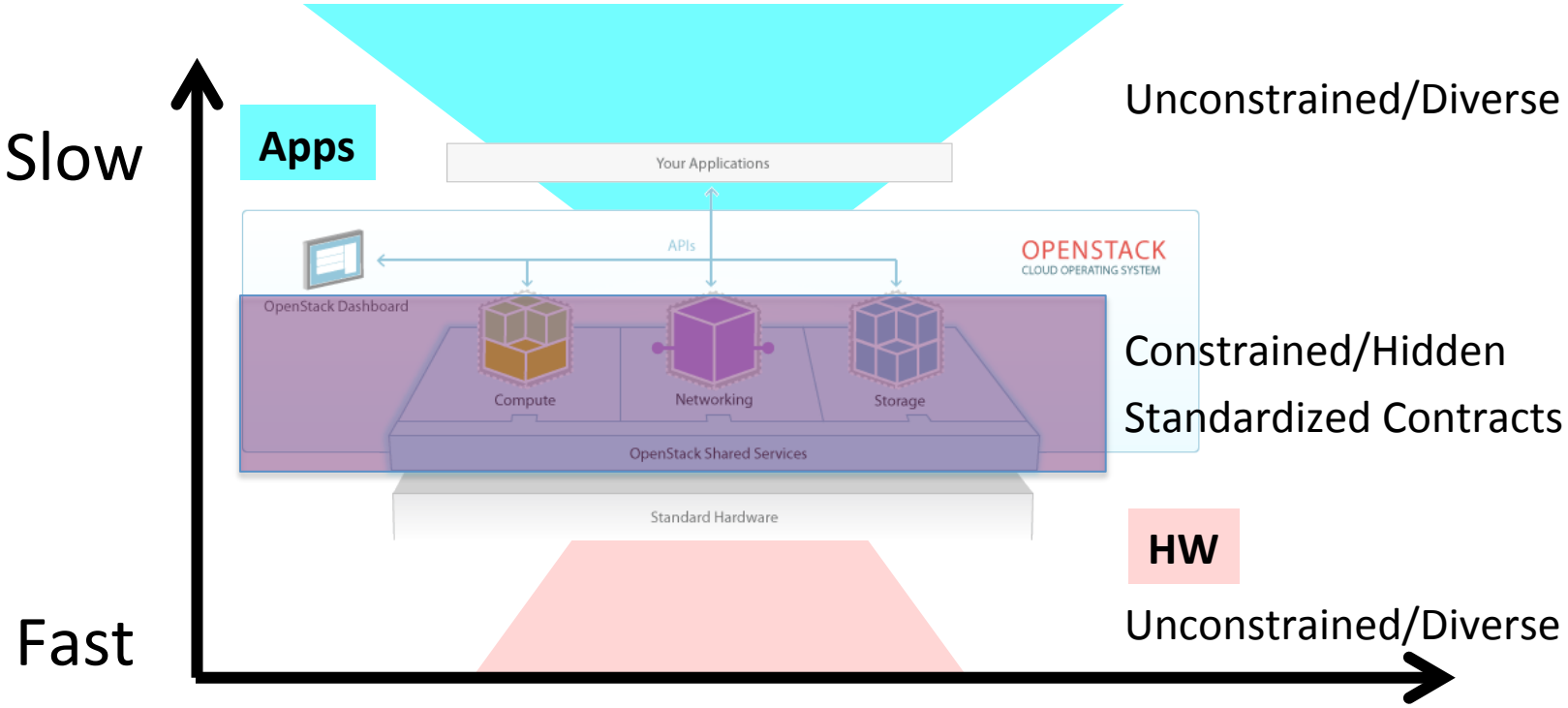
Putting it all Together
Architecture, Layering, and Tradeoffs



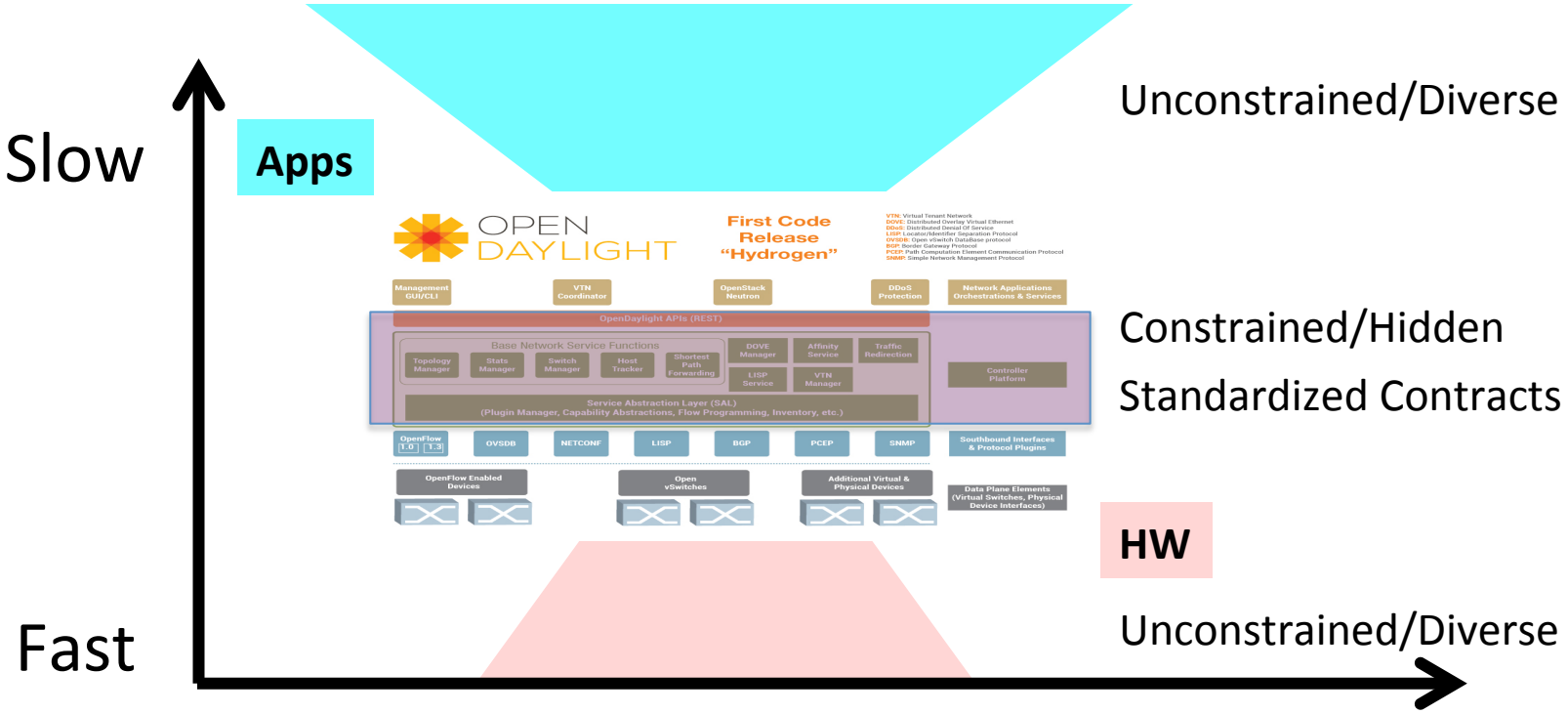
Example: Layered Network Architectures (Internet)



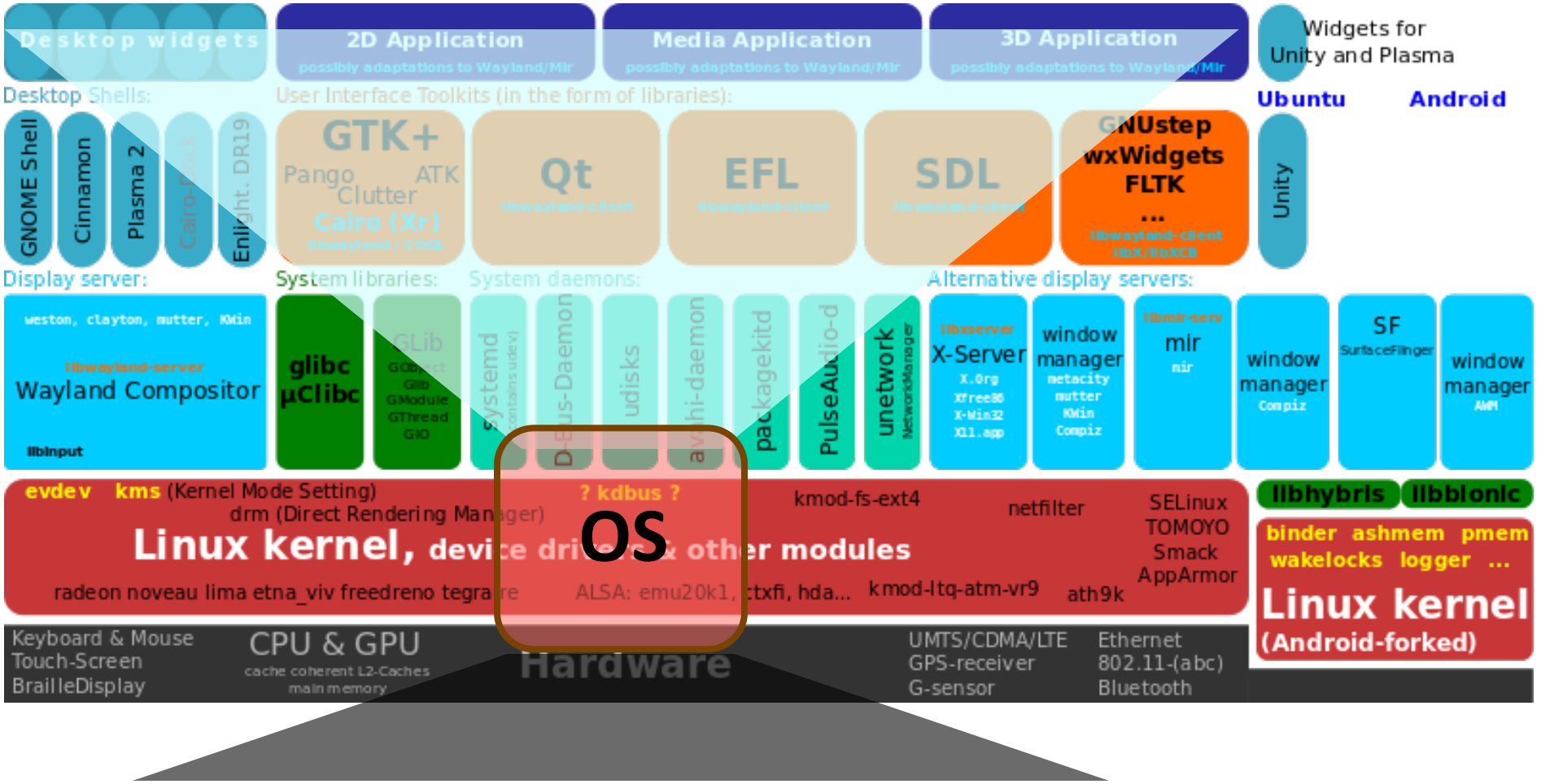
Example: OpenStack



Example: SDN


Slow
Fast
Apps
Unconstrained/Diverse
OPEN DAYLIGHT
First Code Release "Hydrogen"
Management GUI/CLI
VTN Coordinator
OpenStack Neutron
DDoS Protection
Network Applications Orchestrations & Services
OpenDaylight APIs (REST)
Base Network Service Functions
Topology Manager
Stats Manager
Switch Manager
Host Tracker
Shortest Path Forwarding
DOVE Manager
Affinity Service
Traffic Redirection
LISP Service
VTN Manager
Service Abstraction Layer (SAL)
(Plugin Manager, Capability Abstractions, Flow Programming, Inventory, etc.)
Controller Platform
OpenFlow 1.0 1.3
OVSDB
NETCONF
LISP
BGP
PCEP
SNMP
Southbound Interfaces & Protocol Plugins
OpenFlow Enabled Devices
Open vSwitches
Additional Virtual & Physical Devices
Data Plane Elements (Virtual Switches, Physical Device Interfaces)
Constrained/Hidden
Standardized Contracts
HW
Unconstrained/Diverse
Flexible
General
Inflexible
Special

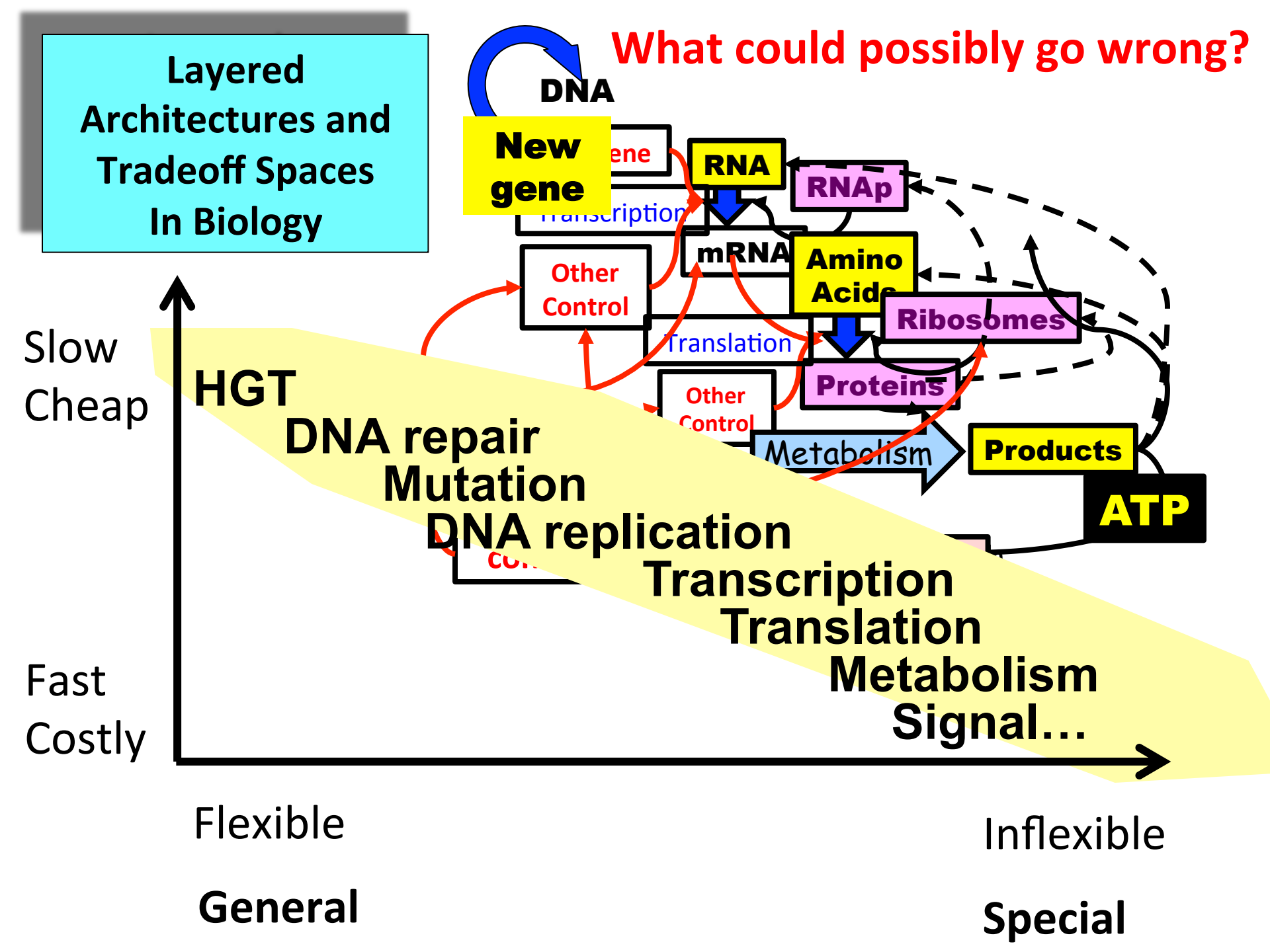
Linux Kernel?



So I've been talking a bit about Biology

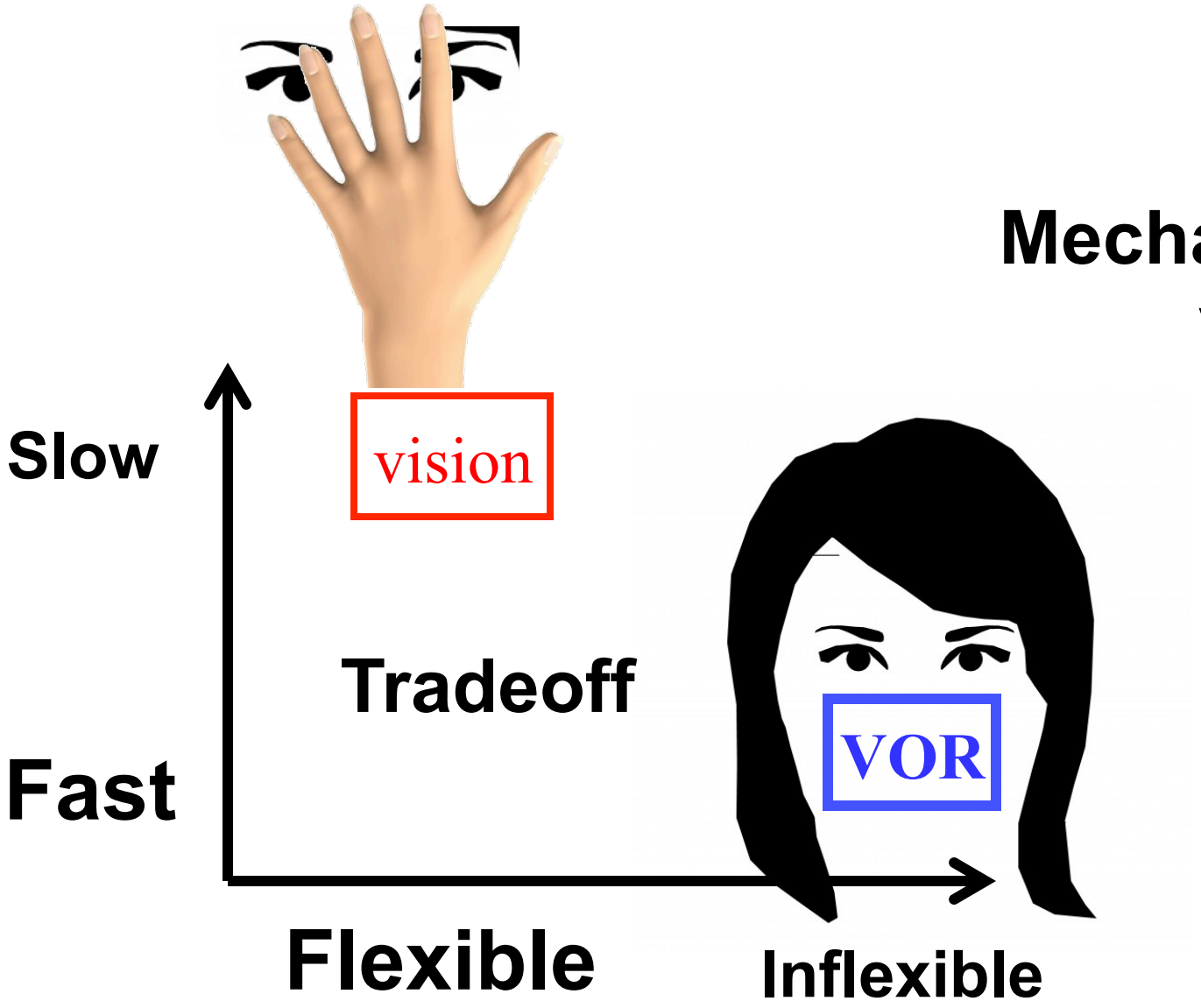
- Are biological systems architecturally layered in the same way?
  - Basically diverse apps and h/w, hidden kernel
  - Constraints that deconstrain
- If so, what kinds of systems can we analyze this way?

Layered Architectures and Tradeoff Spaces In Biology



Neuroscience Example: The Vestibular Ocular Reflex (VOR)

Reflex eye movement that stabilizes images on the retina during head movement

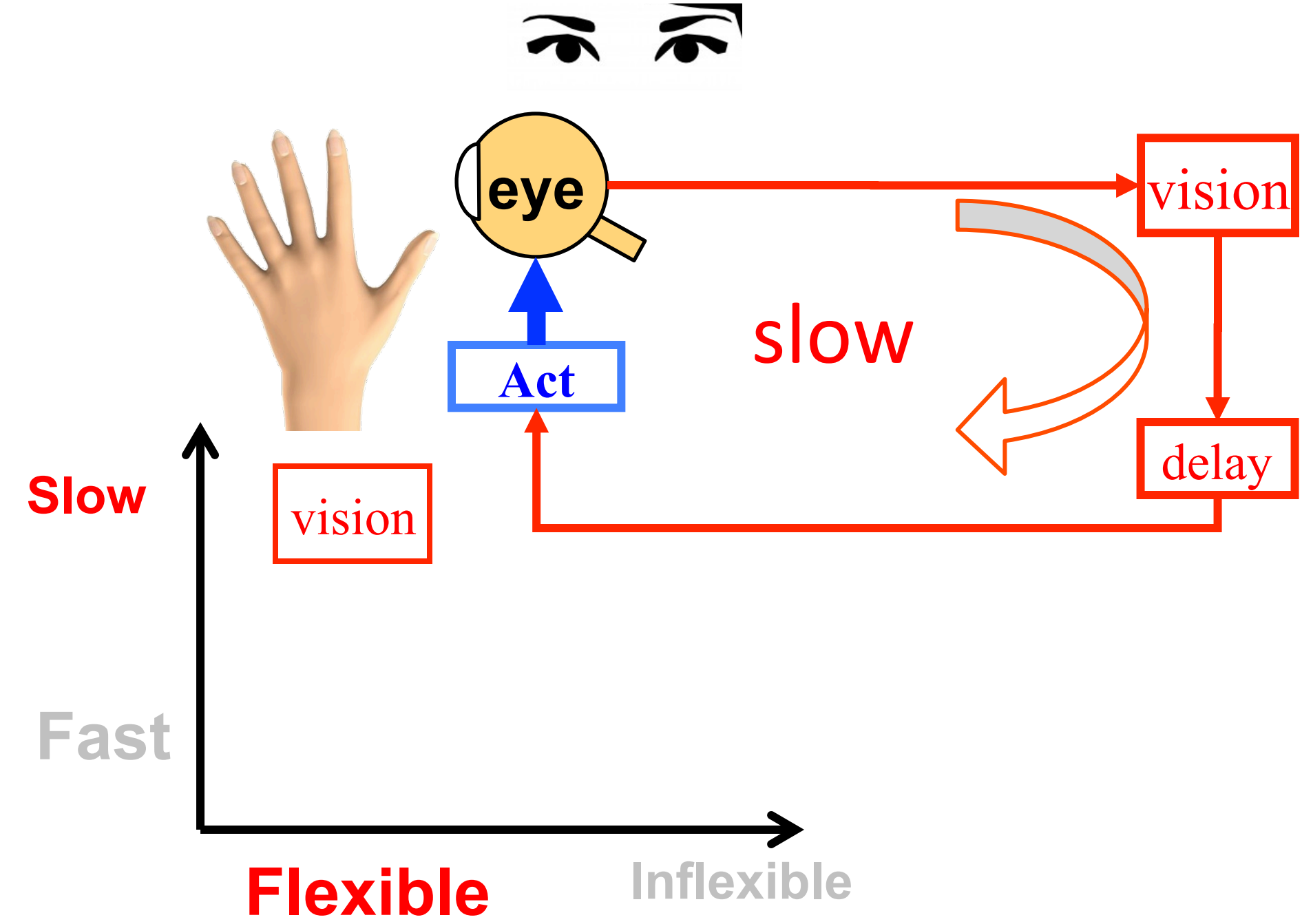


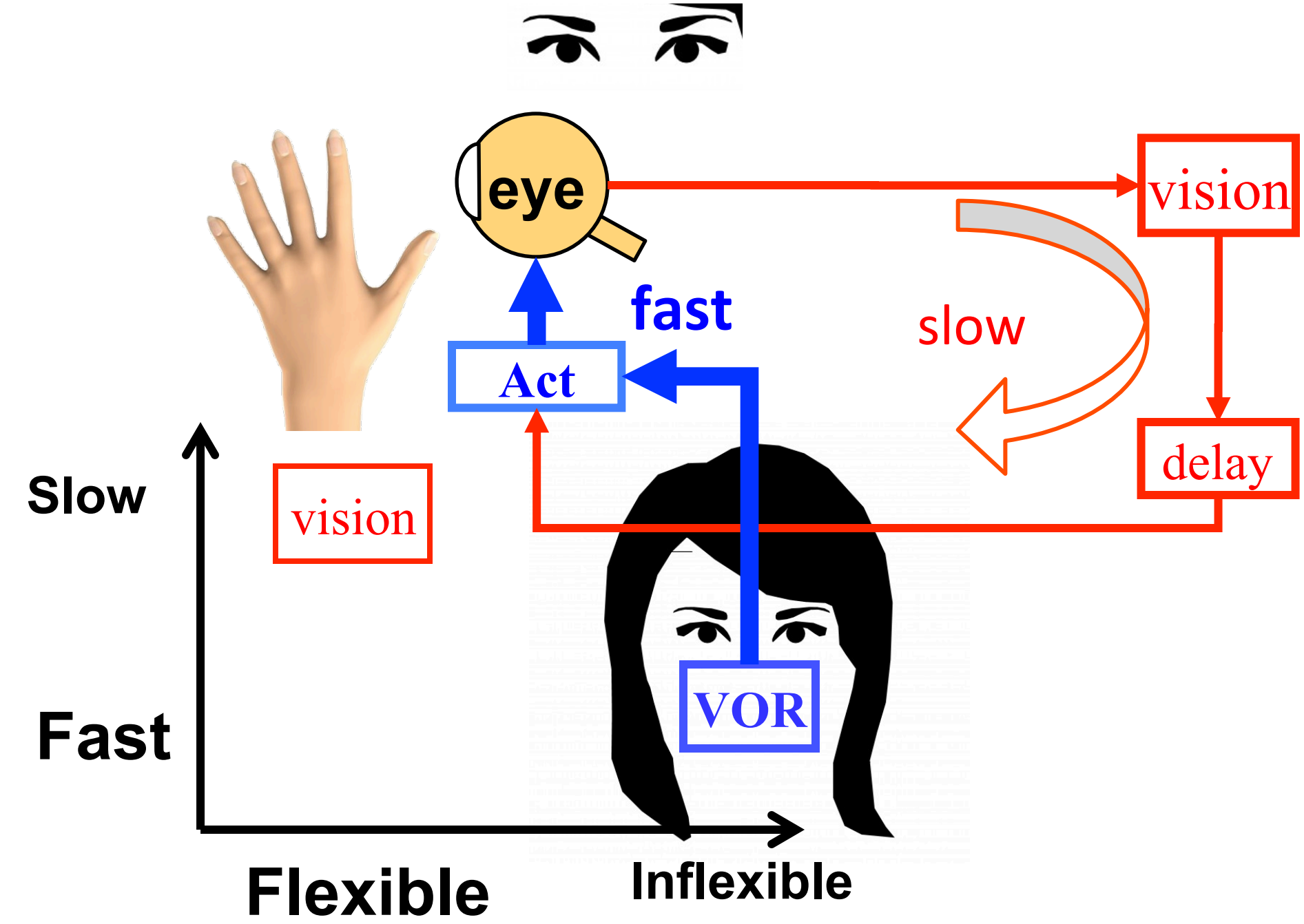
Mechanism

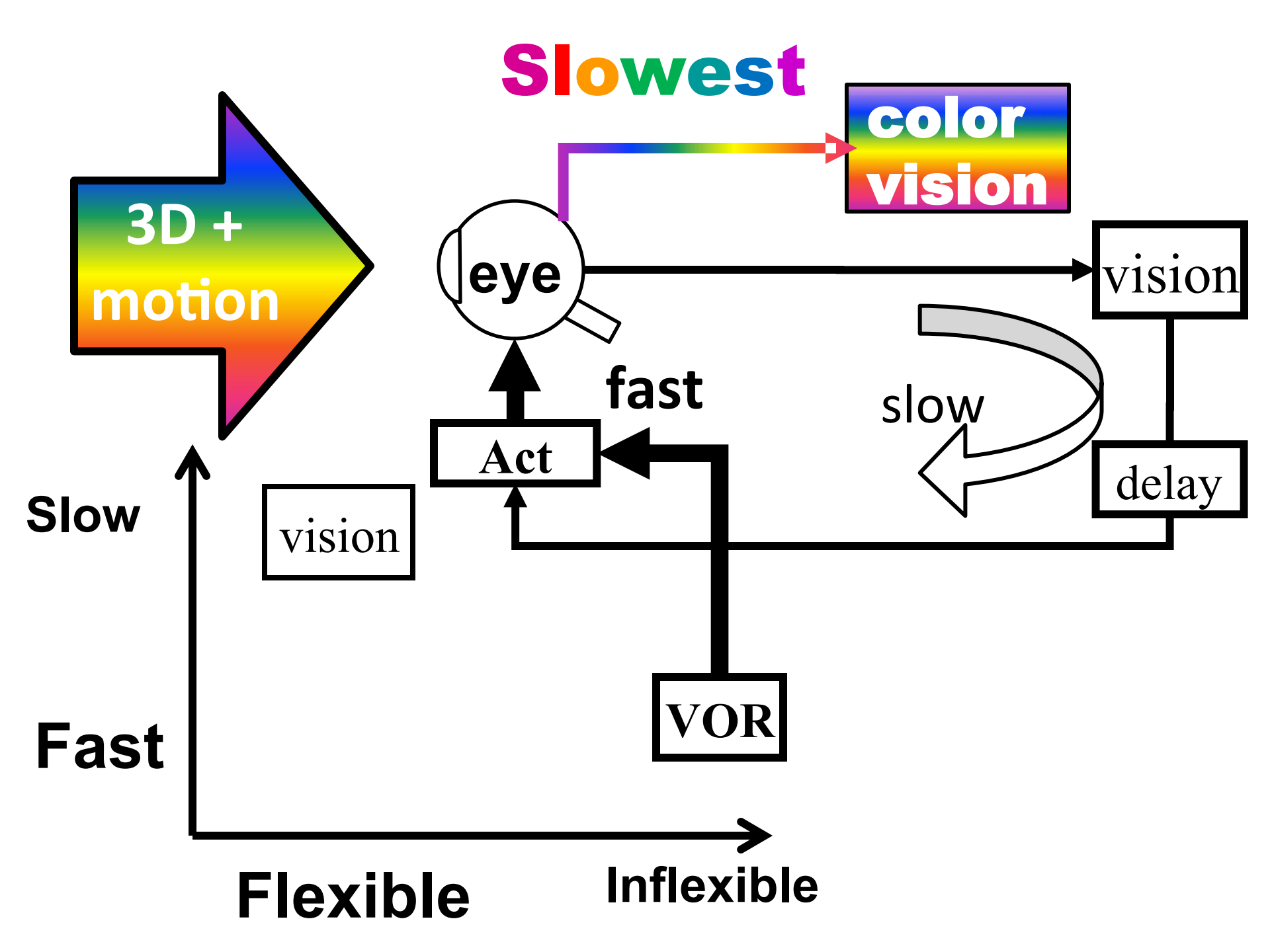
Vestibular

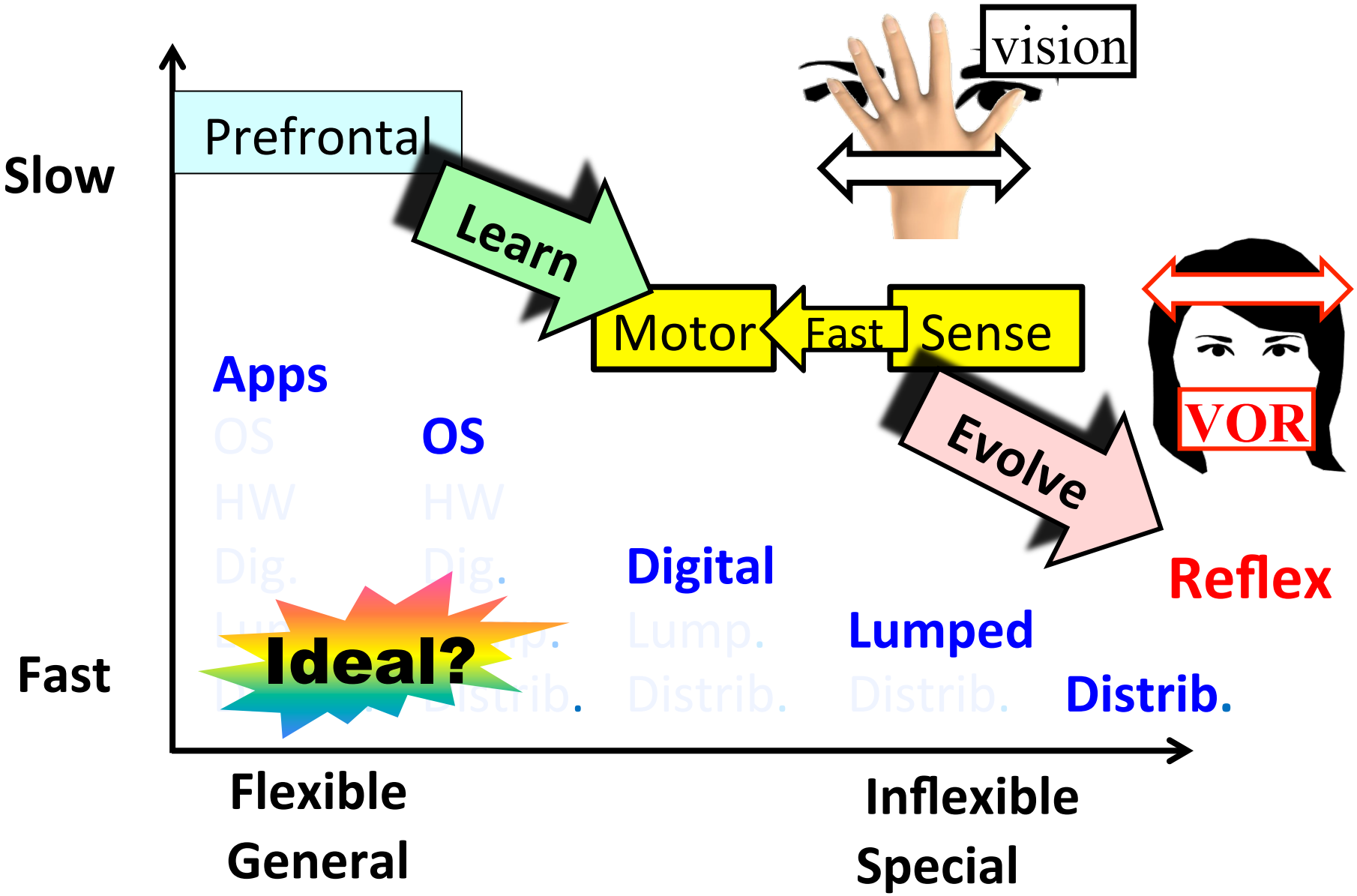
Ocular

Reflex

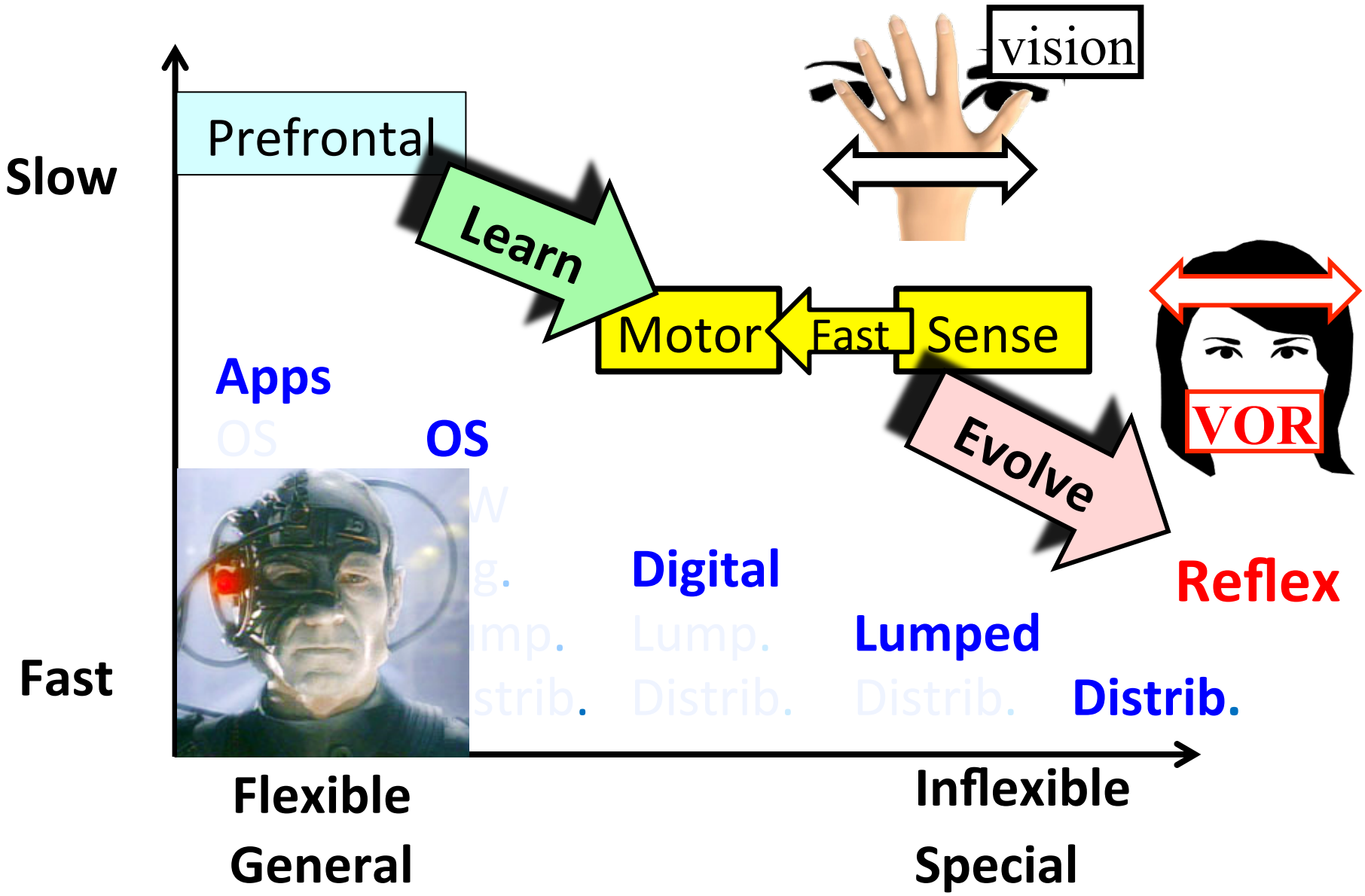

eye
vision
slow
Act
delay
vision
Slow
Fast
Flexible
Inflexible


eye
vision
fast
slow
Act
delay
vision
Slow
Fast
VOR
Flexible
Inflexible


Slowest
color vision
3D + motion
eye
vision
fast
slow
Act
delay
vision
Slow
VOR
Fast
Flexible
Inflexible

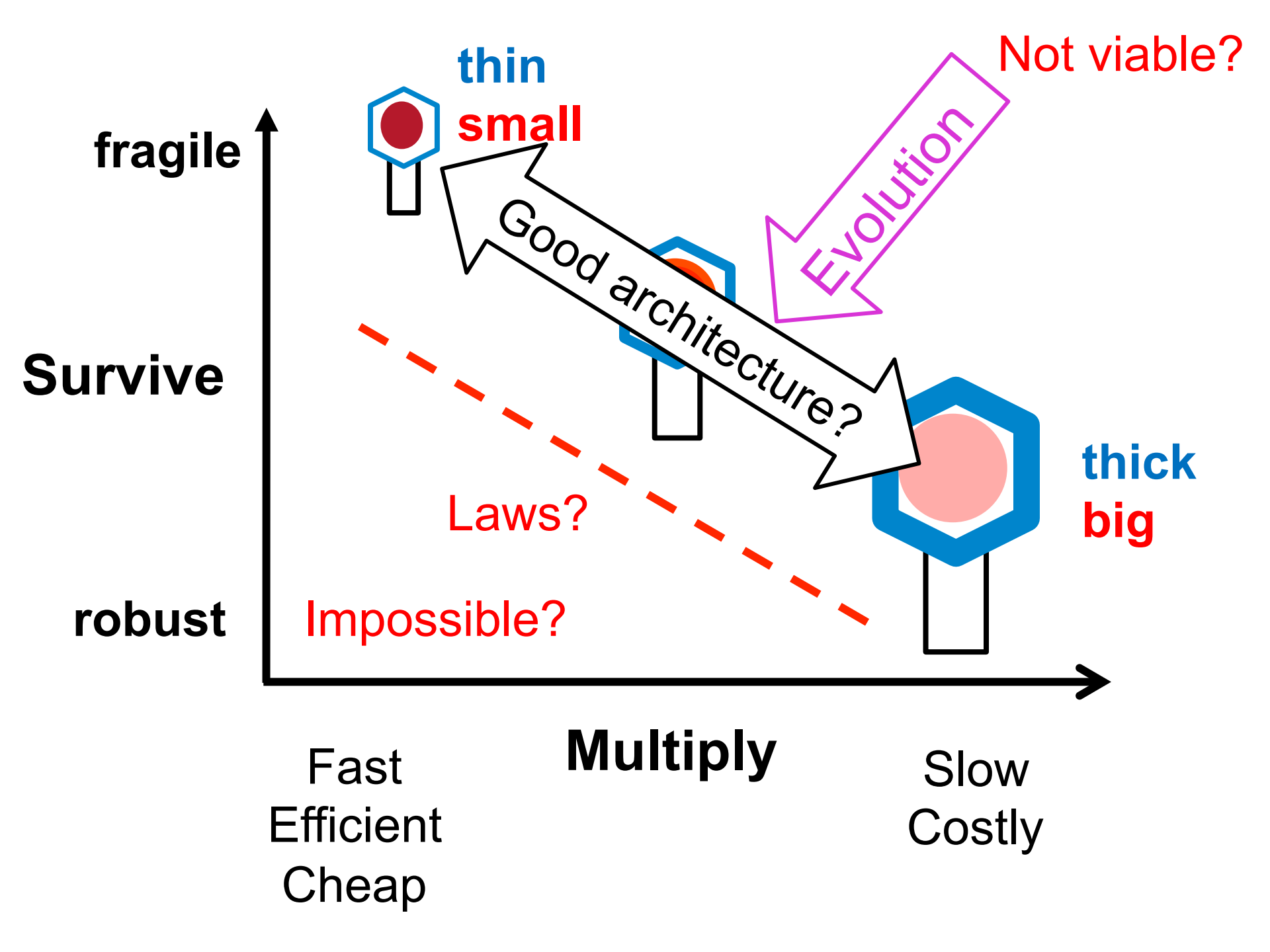


So can a new architecture beat the tradeoff?



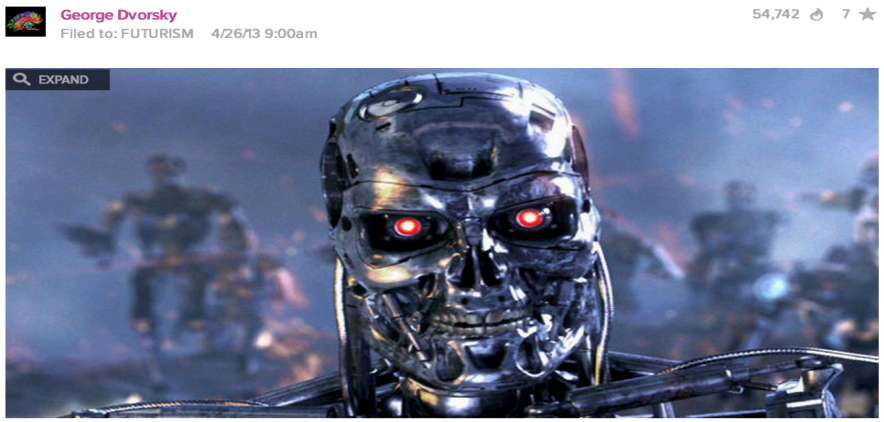
Be careful what you wish for…

http://packetpushers.net/artificial-intelligence-brains-networks-bugs-complexity/


thin
small
Not viable?
Evolution
fragile
Good architecture?
Survive
Laws?
thick
big
robust
Impossible?
Multiply
Fast
Efficient
Cheap
Slow
Costly

Speaking of Being Careful…

How Skynet Might Emerge From Simple Physics



A provocative new paper is proposing that complex intelligent behavior may emerge from a fundamentally simple physical process. The theory offers novel prescriptions for how to build an AI — but it also explains how a world-dominating superintelligence might come about. We spoke to the lead author to learn more.

In the paper, which now appears in *Physical Review Letters,* Harvard physicist and computer scientist Dr. Alex Wissner-Gross posits a **Maximum Causal Entropy Production Principle** — a conjecture that intelligent behavior in general spontaneously emerges from an agent's effort to ensure its freedom of action in the future. According to this theory, intelligent systems move towards those configurations which maximize their ability to respond and adapt to future changes.

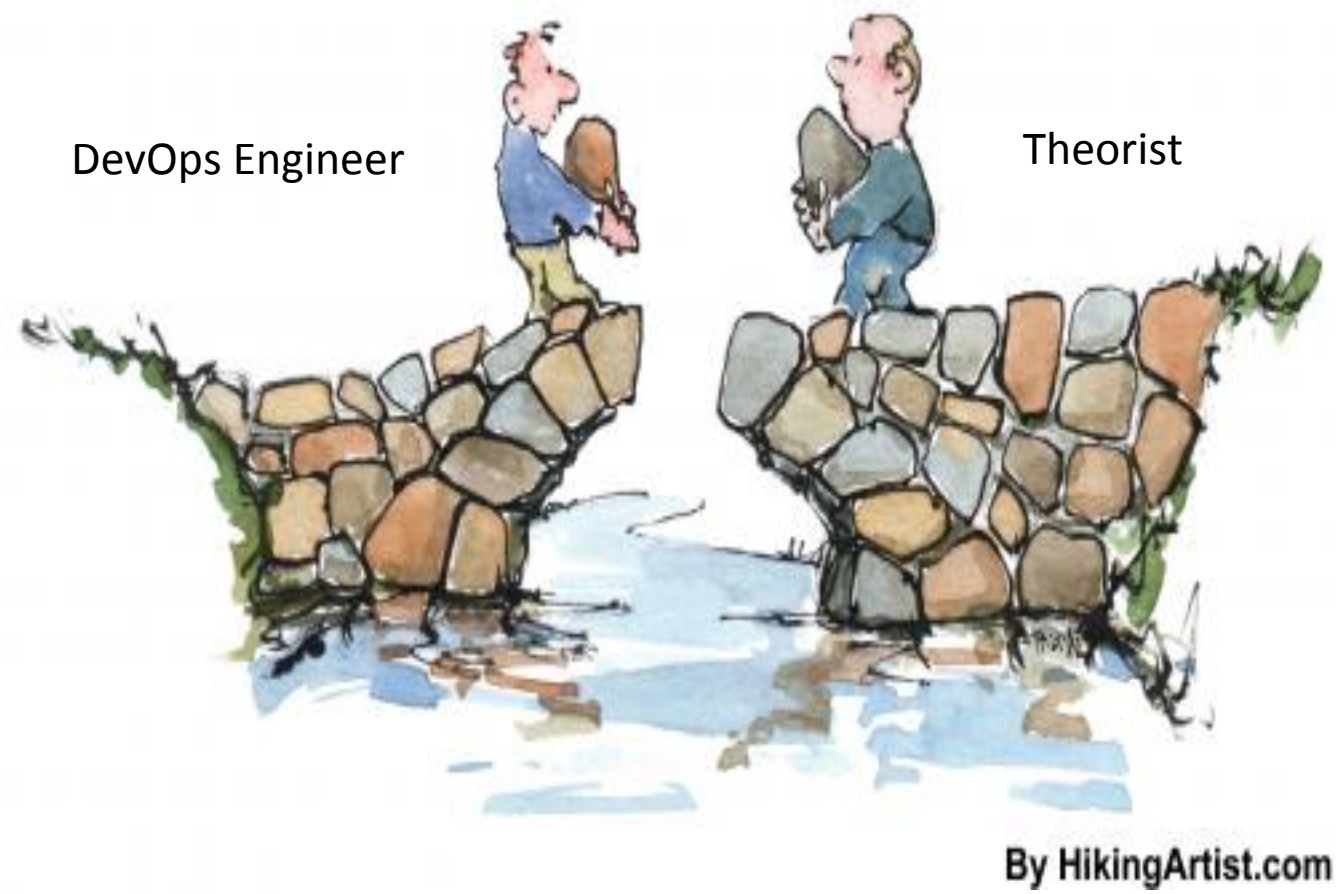
- Dr. Alex Wissner-Gross -- Maximum Causal Entropy Production Principle
- This is a conjecture that intelligent behavior in general spontaneously emerges from
  - an agent's effort to ensure its freedom of action in the future
- → Intelligent systems move towards the configurations which maximize
  - their ability to respond and adapt to future changes

http://io9.com/how-skynet-might-emerge-from-simple-physics-482402911

Agenda

- ~~Goals for this Talk~~
- ~~What is Complexity and Why is it Hidden~~
  - ~~Robustness, Fragility, and Complexity~~
- ~~Tradeoffs?~~
- ~~The Architecture of Complex Systems~~
  - ~~Universal Architectural Principles~~
- The Evolution of DevOps
- A Few Conclusions and Q&A if we have time

Evolving DevOps



Evolving DevOps

- Developers + NetOps = DevOps
  - Bridging development and operations
- However, networking is a young discipline
  - Most of network design/operation/…
    - is done via engineering heuristics
  - We need deeper/better understanding of these networks to get to the next level of scale
- Building bridges?
  - ***Theorists*** *need engineers to understand what is real*
  - ***Engineers*** *need the tools theorists can provide*
  - *We need to think about how to* ***bridge the two***
- Evolving DevOps → Bridging theory and practice
  - Theory + DevOps = <need a good name>
- One attempt
  - http://conferences.sigcomm.org/sigcomm/2014/tutorial-theory+eng.php

Hopefully I've Convinced You…

- That there are *Universal Architectural Features* that are common to biology and technology
- Laws, constraints, tradeoffs
  - Robust/fragile
  - Efficient/wasteful
  - Fast/slow
  - Flexible/inflexible
- Architecture/Layering
- Hidden RYF Complexity
- Hijacking, parasitism, predation

- **Ok, but why is this useful?**

Why is all of this Useful?

- Robust systems are intrinsically hard to understand
  - RYF is an inherent property of both advanced technology and biology
    - Understanding general principles informs what we build
  - Software (e.g., SDN, NFV, Cloud, …) exacerbates the situation
  - And the Internet has reached an unprecedented level of complexity →
    - Need new/analytic ways of designing, deploying, and operating networks

- Nonetheless, many of our goals for the Internet architecture revolve around how to achieve robustness…
  - which requires a deep understanding of the *necessary interplay between complexity and robustness, modularity, feedback, and fragility*[1]
    - which is neither accidental nor superficial
  - Rather, architecture arises from "designs" to cope with uncertainty in environment and components
  - The same "designs" make some protocols hard to evolve
    - Can anyone say, um, IPv6 (or even DNSSEC)?

[1] See Marie E. Csete and John C. Doyle, "Reverse Engineering of Biological Complexity", http://www.cds.caltech.edu/~doyle/wiki/images/0/05/ScienceOnlinePDF.pdf

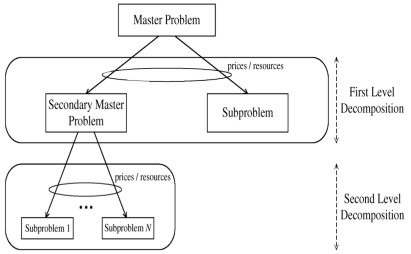
Why is all of this Useful, cont?

- This much seems obvious
  - Understanding these universal architectural features and tradeoffs will help us achieve the scalability and evolvability (operability, deployability, understandability) that we are seeking from the Internet architecture today and going forward

- Perhaps less obvious: This requires a mathematical theory of network architecture
  - Want to be able to analyze/compare/simulate/optimize all aspects of network design/operation
  - Mathematics the natural language for this
  - BTW, as engineers we solve problems ("engineers always know first"), but we can benefit from the tools that theory can provide to help us design/deploy/operate/optimize our networks

- First Cut: "Layering as Optimization Decomposition: A Mathematical Theory of Network Architectures"[1]
  - Network Utility Maximization (NUM)/Layering As Optimization (LAO)/Decomposition Theory

$$\begin{aligned} \text{maximize} \quad & \sum_s U_s(x_s, P_{e,s}) + \sum_j V_j(w_j) \\ \text{subject to} \quad & \mathbf{Rx} \le \mathbf{c}(\mathbf{w}, \mathbf{P}_e), \\ & \mathbf{x} \in \mathcal{C}_1(\mathbf{P}_e), \quad \mathbf{x} \in \mathcal{C}_2(\mathbf{F}) \text{ or } \in \mathbf{\Pi}(\mathbf{w}), \\ & \mathbf{R} \in \mathcal{R}, \quad \mathbf{F} \in \mathcal{F}, \quad \mathbf{w} \in \mathcal{W}. \end{aligned}$$



  - TCP and Stable Path Problem (BGP) reverse-engineered as *Generalized* NUM problems
  - Need something like LAO/G-NUM + "Constraints that deconstrain" view

- Evolving DevOps means bridging the DevOps and theory communities
  - http://conferences.sigcomm.org/sigcomm/2014/tutorial-theory+eng.php

1 http://www.princeton.edu/~chiangm/layering.pdf

Q&A

Thanks!