Innovation as the Driver of Cyber Insecurity

Herb Lin
Stanford University
herblin@Stanford.edu

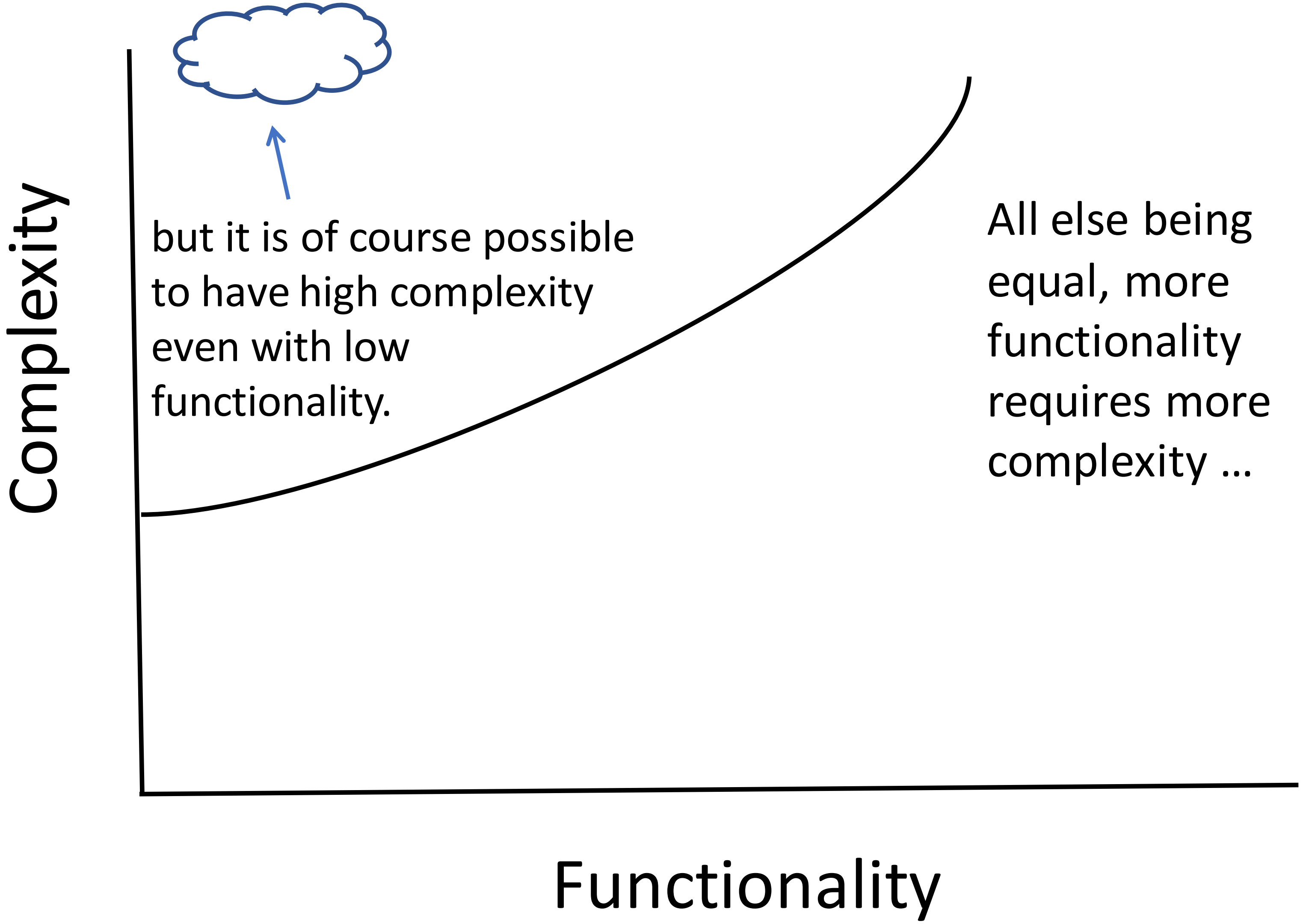
The one-slide version of this talk

- Our appetite for increased functionality afforded by information technology is unlimited.
- Increased functionality of information technology necessarily entails increased complexity of design and implementation.
- Complexity is the enemy of security.
- Systems will be increasingly insecure if we do not find a way to moderate/curb/manage our appetite for functionality.
- We need a structured, disciplined way to say “no” to some proposals for increased functionality because of the inevitable reduction of security. . . and we don’t have it.
- We do have messy, undisciplined ways to say no.

On the appetite for functionality

- Ultimate constraint on physical systems is the laws of physics.
- Ultimate constraint on software is the human imagination—which is unlimited.
- The entire Silicon Valley ecosystem is built on the premise that users of information technology will always want to do more with technology and hence will pay for continuing innovation.
- Consumers of information technology are more naturally optimizers than satisficers.
  - Good enough is never enough: we always want more: faster, easier, more features
- Costs of increased functionality (software-driven) have been hidden by increased hardware capability.

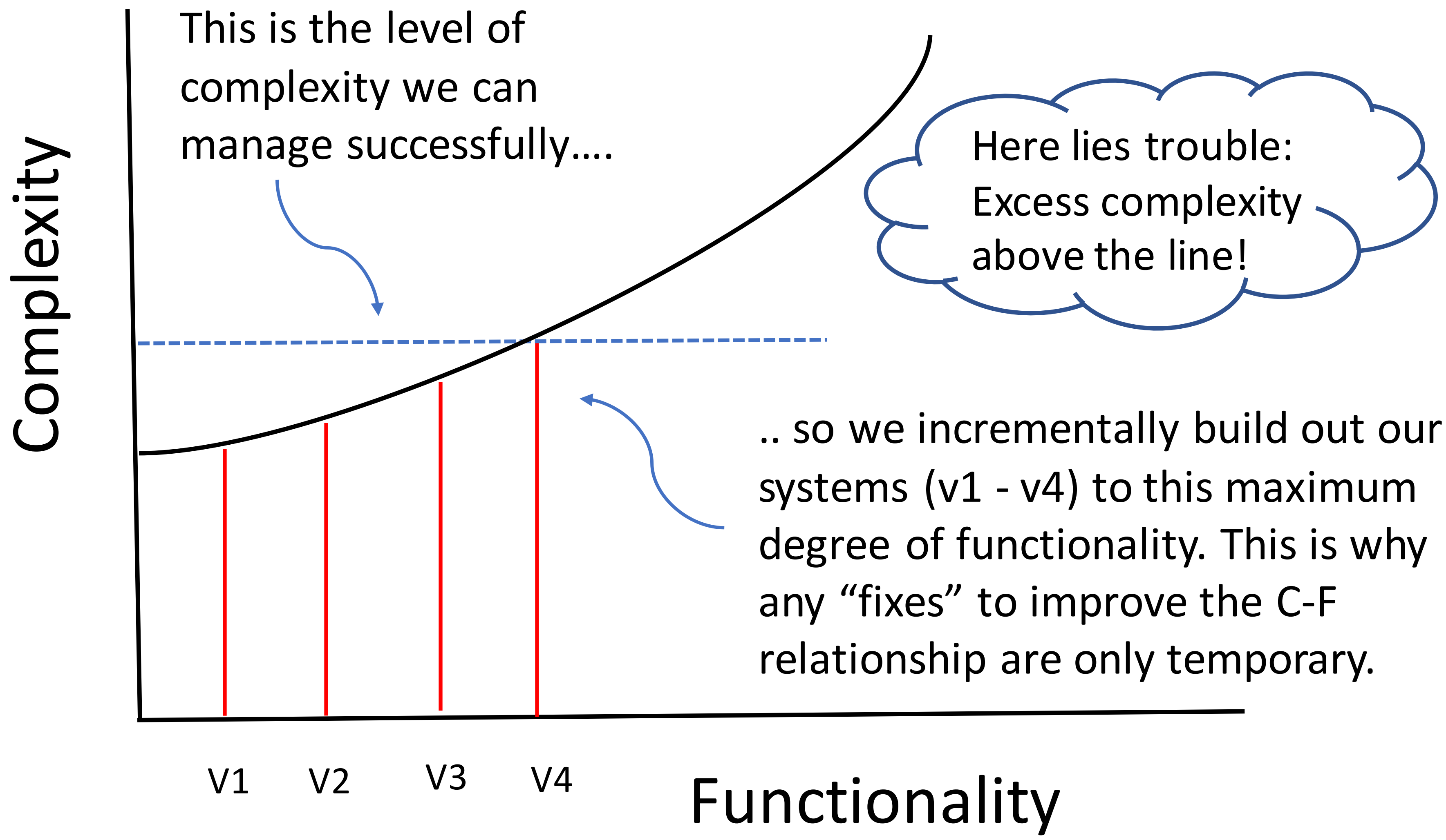
On the complexity-functionality linkage



but it is of course possible to have high complexity even with low functionality.

All else being equal, more functionality requires more complexity …

Incremental innovation

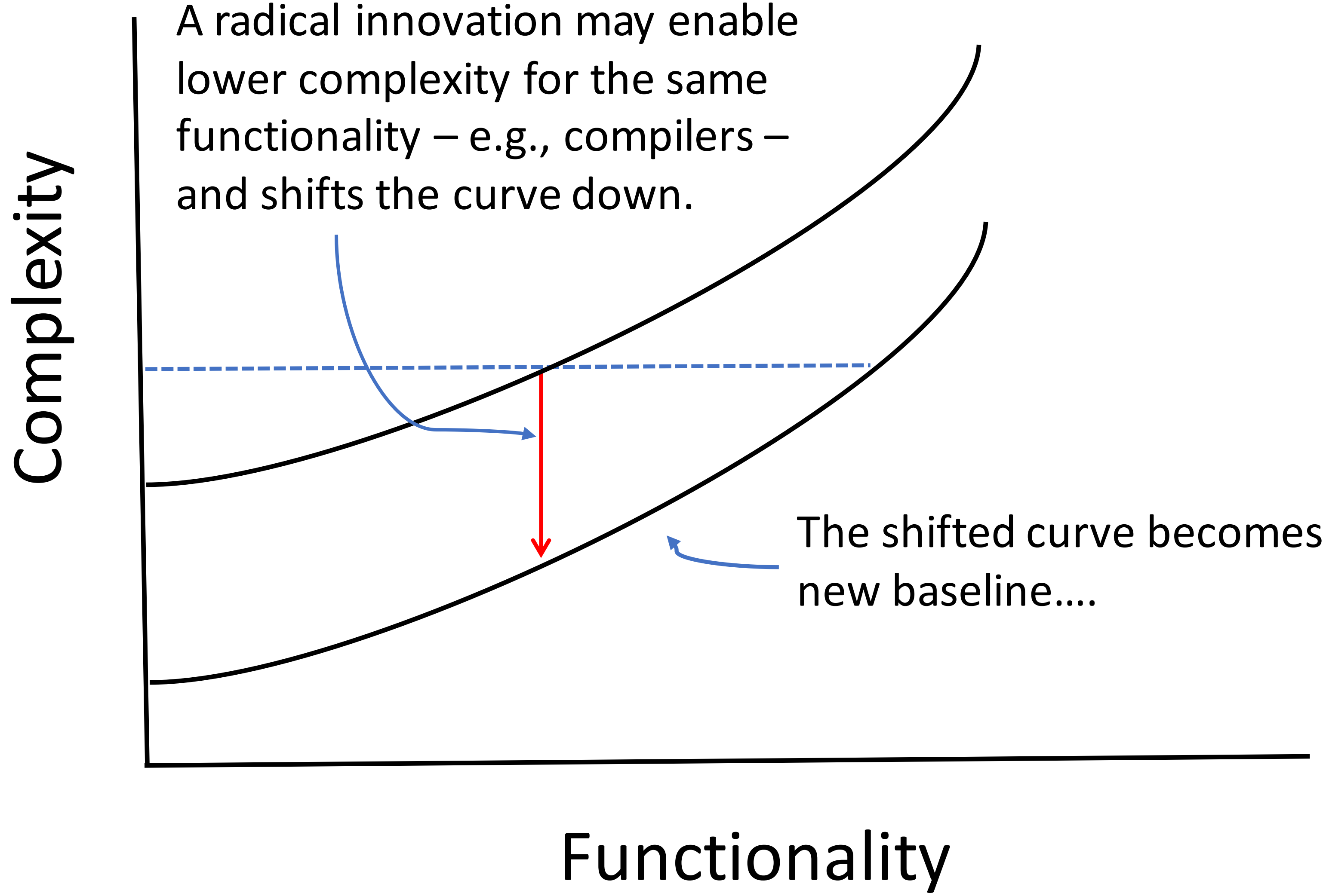


This is the level of complexity we can manage successfully….

Here lies trouble: Excess complexity above the line!

.. so we incrementally build out our systems (v1 - v4) to this maximum degree of functionality. This is why any “fixes” to improve the C-F relationship are only temporary.

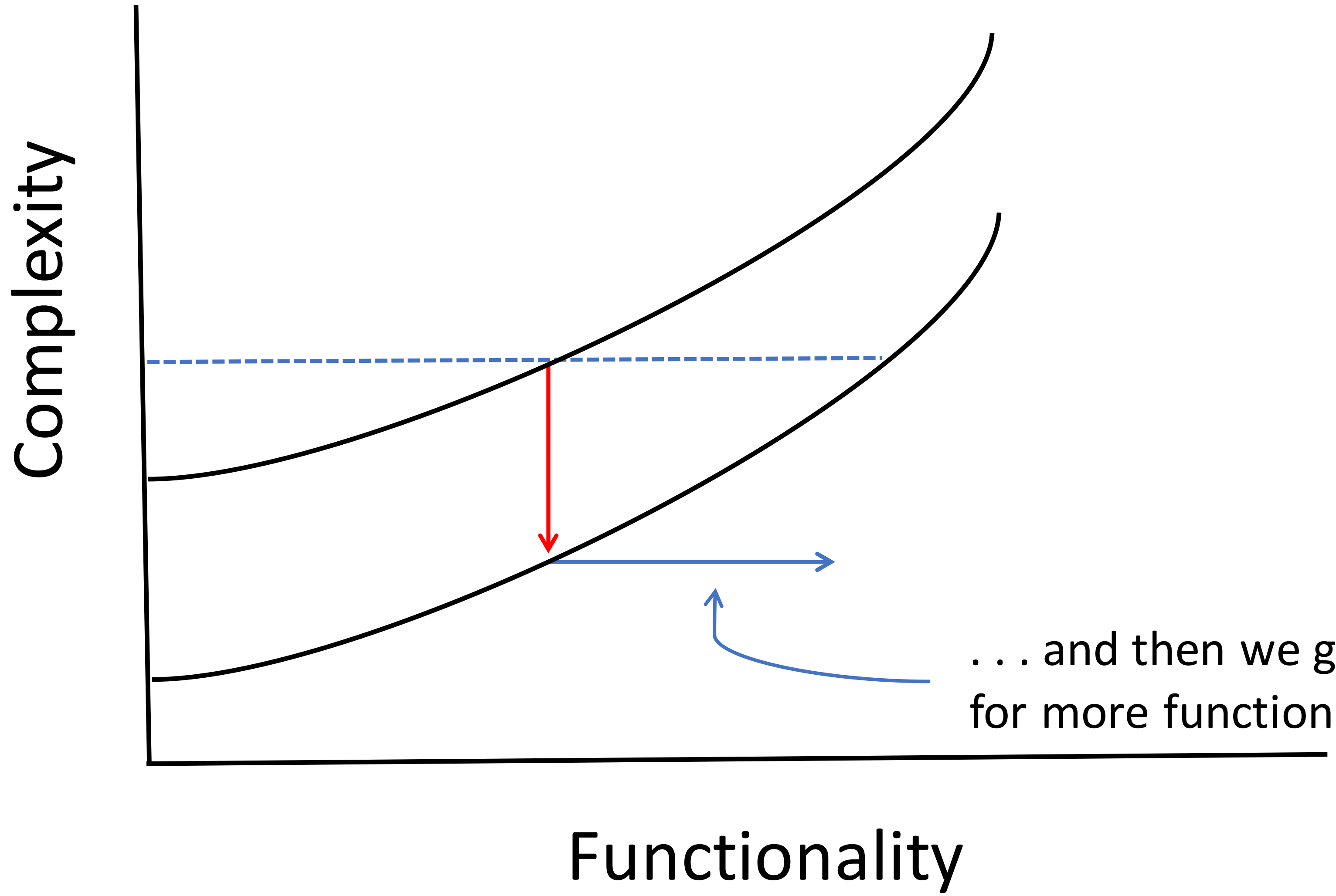
Radical innovation



A radical innovation may enable lower complexity for the same functionality – e.g., compilers – and shifts the curve down.

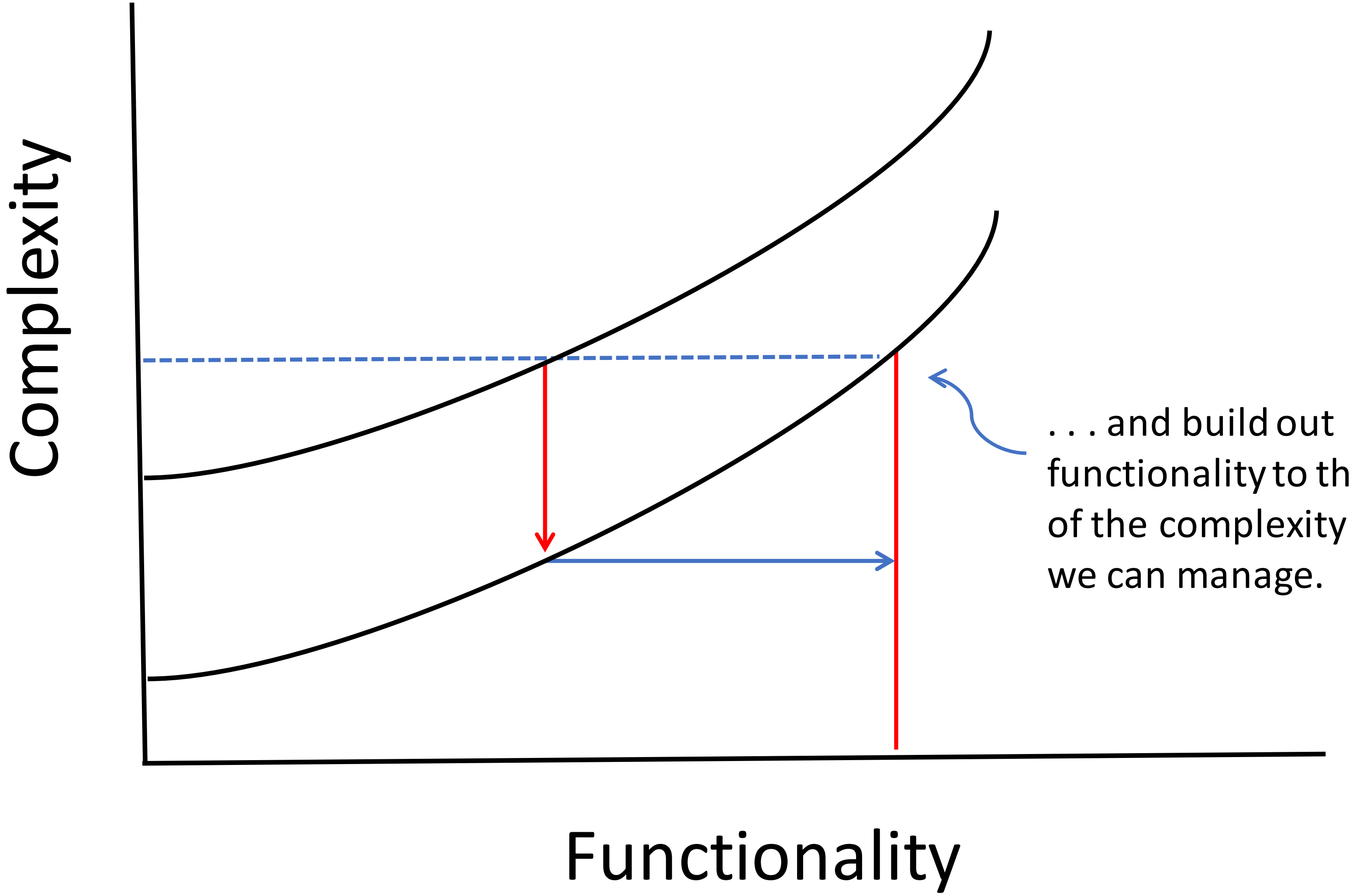
The shifted curve becomes the new baseline….

Radical innovation



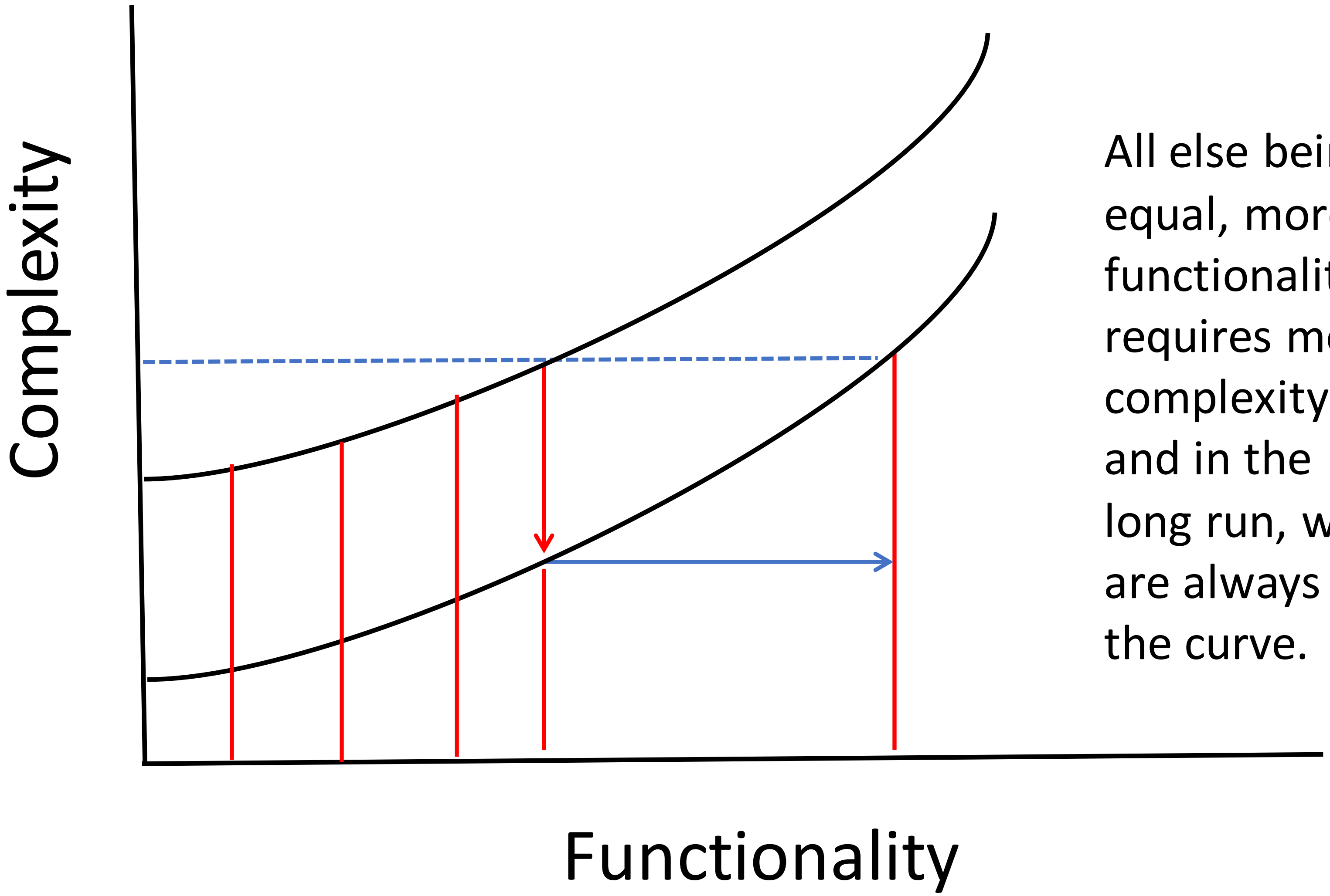
. . . and then we get greedy
for more functionality ...

Radical innovation



. . . and build out
functionality to the limit
of the complexity
we can manage.

Summary slide



All else being equal, more functionality requires more complexity – and in the long run, we are always on the curve.

Some observations

- Advances in reducing complexity become the new baseline.
- No known approach that can continually reduce the strength of the linkage (don’t know how to reduce the slope of the curve or to regularly shift the curve downwards).
- No equivalent to Moore’s law that can even predict reduction of linkage (and even Moore’s law isn’t an approach – it’s an empirical result driven by economics).
- If demands for functionality are unbounded, any given advance in managing complexity only pushes out the timeline.
- Determinants of the complexity that humans can manage are based on fundamental human limitations (speculation)
  - Short-term memory 7 ± 2 items
  - Limits on communication capability (when working in teams)

Complexity and reliability through the eyes of normal accident theory

- It is morning and you have an important meeting downtown.
- Your spouse has already left. Unfortunately, he/she left the glass coffee pot on a lit burner and it cracked.
- You desperately need your coffee so you rummage around for an old drip coffee pot.
- You pace back and forth waiting for the water to boil while watching the clock. After a quick cup you dash out the door.
- You get in your car only to realize that you left your car and apartment keys inside the house.
- That’s okay. You keep a spare house key hidden outside for just such emergencies. But then you remember that you gave your spare key to a friend. (failed redundant pathway)
- You try to borrow your neighbor’s car. He says his generator went out a week earlier. (failed backup system)
- Well, there is always the bus. But the neighbor informs you that the bus drivers are on strike. (unavailable work around)
- You call a cab but none can be had because of the bus strike. (tightly coupled events)
- You give up and call in saying you can’t make the meeting.

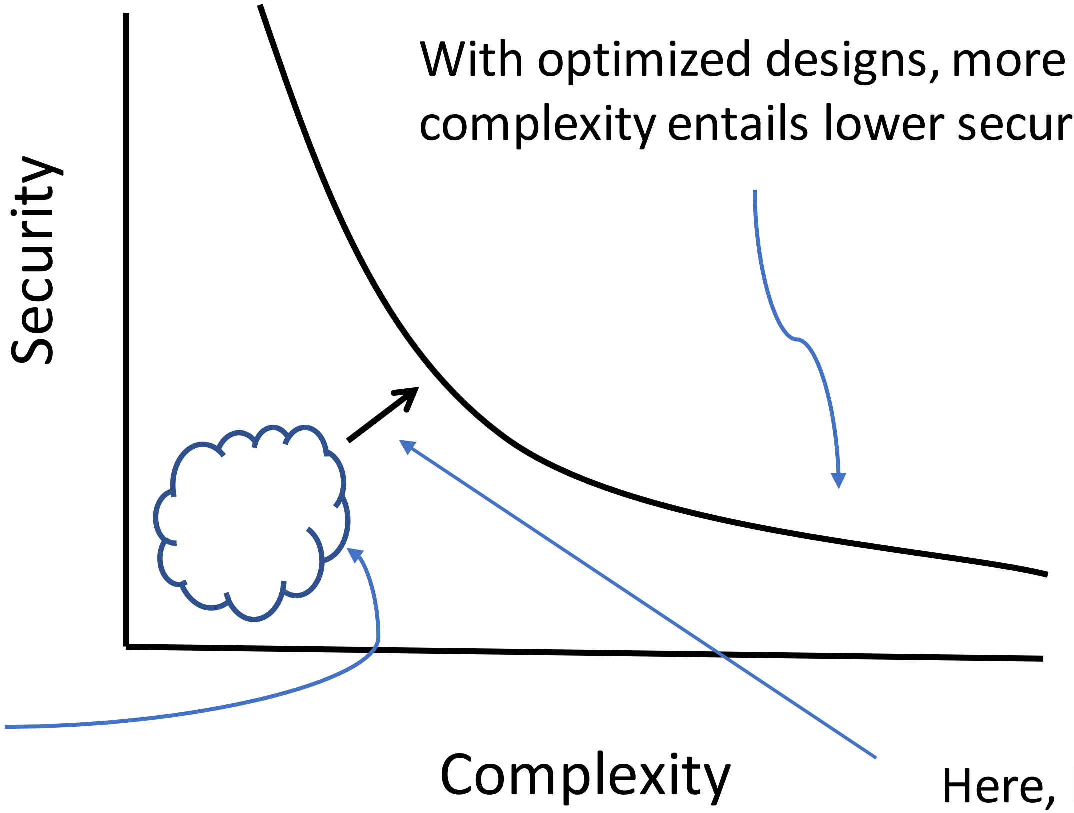
- The mission is simple—attend a meeting.
  - The process is apparently straightforward--have some coffee, get in the car, drive to the meeting, provide input.
  - Deviations from the “straightforward process” cause problems
    - Keys/car link expected (unsurprising).
    - Cracked coffeepot/use of car; taxi/bus contract dispute not expected (surprising)
    - These interactions were not taken into account in the design of the process.
- Cascading failures can accelerate out of control, confounding human operators and denying them a chance for recovery.

Complex systems exhibit:

- High interactivity:
  - Hard to address one problem without causing unintended side-effects in other areas and creating more problems (“the main cause of problems is solutions”)
- Non-linearity:
  - Small changes to inputs can cause large changes in output. Hence, intuition fails.
- Distributed connectivity:
  - peer-to-peer rather than centrally managed connections mean fewer control opportunities; many opportunities for failures to jump across subsystem boundaries
- Path dependency:
  - changes to system state cannot be undone only by reversing operations that produced the change; path back is not the same as the path forward.

- Cascading failures may result.
  - Hidden connections exposed
  - Protective redundancies neutralized
  - Separation mechanisms bypassed
  - Chance circumstances converge, rendering advance planning impossible.
- Resulting impact on security
  - More components ➔ more relationships to debug.
  - Problems harder to find, harder to fix ➔ more vulnerabilities to be exploited
- At a sufficiently high level of complexity, a system’s behavior is not entirely predictable (or even understood)—obviously a security risk, since poorly understood interactions are primary points of vulnerability.

Depicting the complexity-security tradeoff



With optimized designs, more complexity entails lower security. . .

but it is of course possible to have lousy security even with simple systems.

Here, better security is possible even with greater complexity, but eventually you hit the tradeoff boundary.

Three approaches to security and resilience

- After development (bolt-on security):
  - Establish high level requirements for system functionality.
  - Design system based on functionality requirements.
  - Implement design.
  - Add security after implementation is done.
- At the start of development (baked-in security or security by design):
  - Establish high level requirements for system functionality.
  - Design system securely starting from established functionality requirements (this is what most understand to mean “bake in security from the start”)
  - Implement design (which now has security baked in)
- Before development (security through requirements tradeoff)
  - Establish high-level performance requirements along with security requirements.
  - Trade off performance requirements against security when necessary.

A big problem with the argument so far—the lack of good metrics

- For complexity (examples)
  - Lines of code
  - Control flow (e.g., number of linearly independent paths through a program's code)
  - Interface complexity (data flows in and out of a module)
  - Decisional complexity (number of conditionals)
  - Data complexity (number of variables)
- For security (examples)
  - Process security metrics (# of policy violations, % of weak passwords)
  - Network security metrics (# instances of malware blocked, # port probes)
  - People security metrics (% workforce undergoing security awareness training)
  - Software security metrics (attack surface in code). Some problems:
    - Mostly input metrics, not output or outcome metrics
    - Attack surface doesn't account for implementation flaws, only for resources that can be used to attack
- Given two existing systems that perform the same tasks, which is more secure? Do today's metrics help make the comparison?

- Answer: not much
  - Measures are relative: no way of mapping to real-world needs (what does an attack surface of 93 vs 107 mean in real world terms?)
  - Measures are related to existing code base, not requirements (thus, predictions of complexity and security not possible)
  - No way to predict from general (and incomplete) requirements statement
  - In practice, neither security requirements nor system complexity are static

Nevertheless, despite the inutility of most formal metrics, we have an intuitive sense that some things are more or less complex or secure than others.

If there no natural constraints on functionality, what might be “unnatural” constraints?

- In the physical world, we do not make artifacts whose performance is constrained only by the laws of physics.
  - Economics: must build to be affordable on large scale
  - Law: must build to be safe for society at large (can drive 300 mph on your own property but not on city streets)
- What might work in the cyber world? Here’s some things that help (but not by themselves):
  - Proofs of correctness (only possible for relatively small systems)
    - doing the right things, not doing wrong things—both are important
  - Testing for problems
    - either before or after deliver to customer!
    - Failures in testing (or bugs as revealed in use) lead to system fixes (but need for fixes hard to predict in detail)

- Here are two things that might help more
  - Restructuring of corporate governance
    - C-suite must be more knowledgeable of security basics, AND..
    - CISO/CIO must be more knowledgeable of business basics.
    - CEO role is to make tradeoff between functionality and security; CISO role to ensure that the tradeoff is an informed one.
    - Security must be regarded with the same importance as cost to be truly regarded as a critical attribute of products, a business must sometimes forgo an aspect of product functionality to gain security benefits. Not all the time, but some of the time.
  - Liability for security flaws
    - Flaws can be reflected in requirements, design, implementation, operation.
    - Liability will induce caution in both developmental and operational processes.
    - However, liability will slow down innovation and allow others to fill the innovation gap.
    - Under some circumstances, perhaps the tradeoff is worth it.

- On regulation for security:
  - FDA model for drugs is at one extreme of regulation – do not sell unless safety and efficacy are proven
  - Software and other information technology is at other extreme—can sell anything: caveat emptor.
  - But FDA is slowly embracing greater acceptance of risk in certain circumstances.
  - Perhaps law will embrace greater constraints on information technology in the coming tech winter.
    - Tech winter is confluence of tech-driven unemployment; large wealth disparities; tech-induced depression and other dysfunctionality in kids; assault on privacy/civil liberties; election interference; fake news
    - US National Cyber Director has expressed sympathy for direct regulation of cybersecurity in certain areas.

A final word on security and reliability

- System failure: malfunction due to error or weakness in the design, implementation, or operation of a system.
  - Security failure if system failure is caused by malicious unauthorized access
  - Reliability failure otherwise

Security	Reliability
inability of the environment to have an undesired effect on the system.	inability of the system to have an undesired effect on its environment;
Doing the right thing; Insecurity when system does what it should not;	Doing the thing right Unreliability when system doesn't do what it should
External focus	Internal focus

For more information…

Herb Lin
Center for International Security and Cooperation
Hoover Institution
Stanford University
650-497-8600
herblin@stanford.edu