Security, Moore’s law, and the anomaly of cheap complexity

Thomas Dullien / “Halvar Flake”
@halvarflake

Google Project Zero

CYCON

Escalating complexity

- Almost 20 years since I wrote my first exploit
- I work hard to keep up with changes to the computing stack
- Every year, I feel like I understand **a smaller portion** of the computing stack

I thought this was because I was getting old, but in reality:

A modern CPU contains factor 1024+ more transistors than an 80486.

This does not account for things like GPUs, NICs, Basebands, TPUs etc.
Things are **objectively** getting more complicated, at superlinear rate.

Escalating security problems

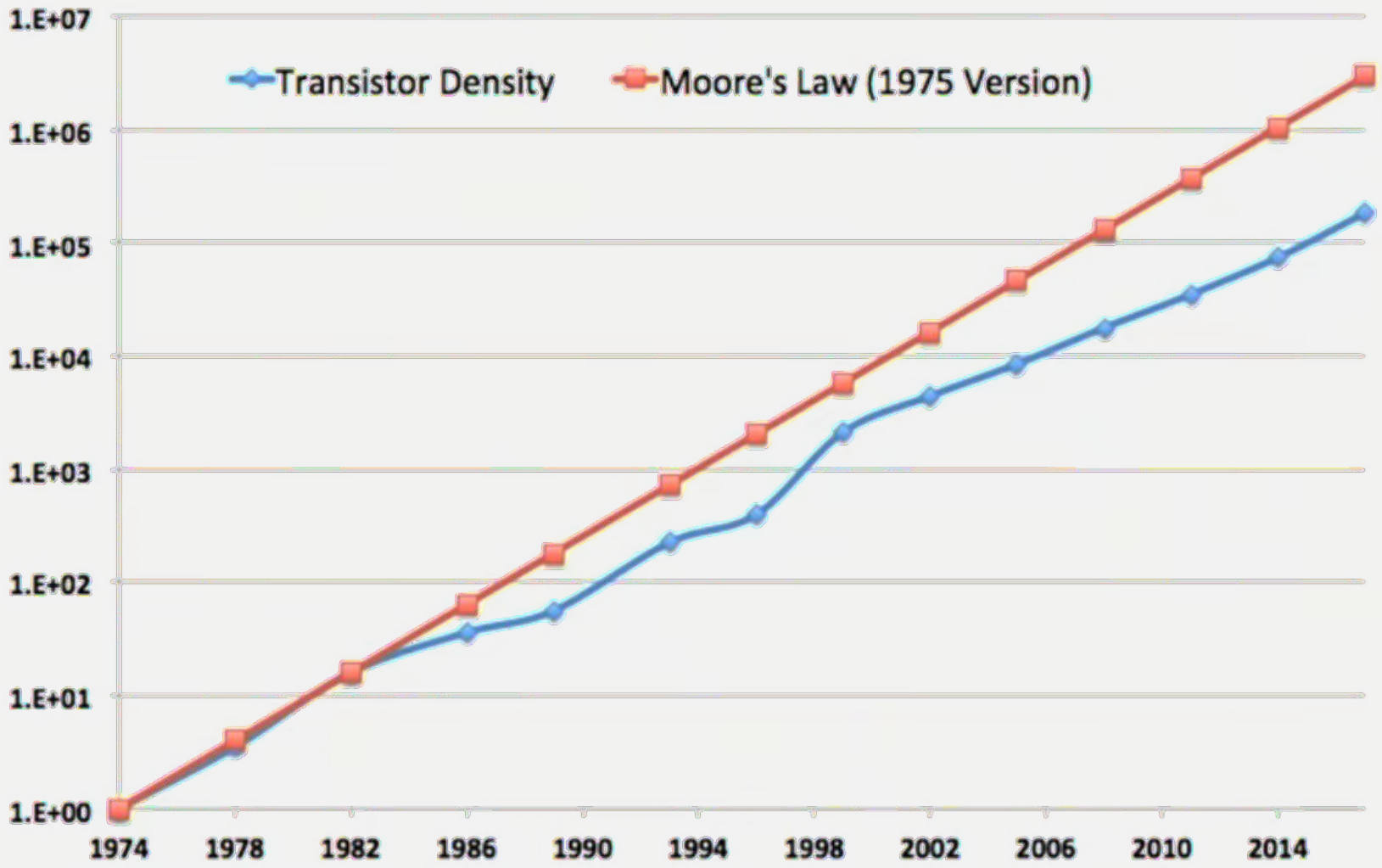
- For the first 10 years of those 20 years I was convinced security will be solved soon
- … but I am here now.

The only thing that ever yielded real security gains was controlling complexity.

This talk examines the relationship between **complexity** and **failure of security**, and discusses the underlying forces that drive both.

CYCON

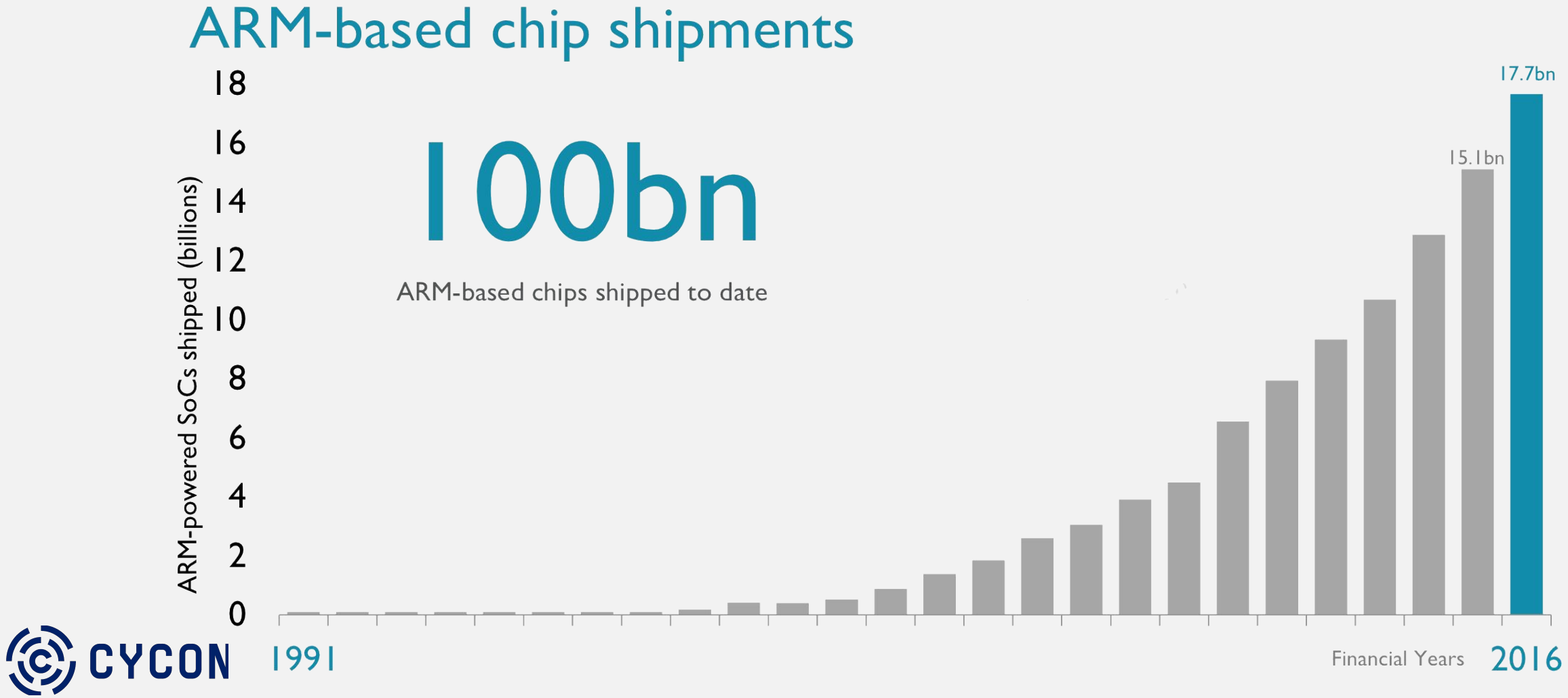
Transistor density is still moving up



Source: Henessy's Google I/O presentation

CYCON

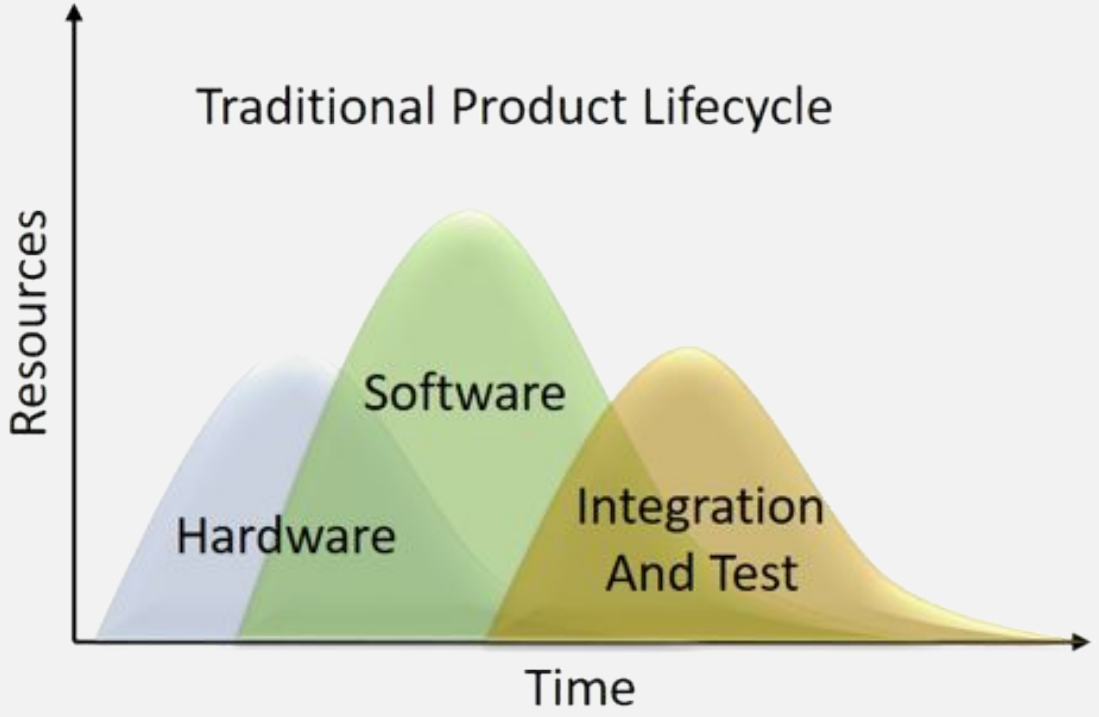
~3 new CPUs per human per year

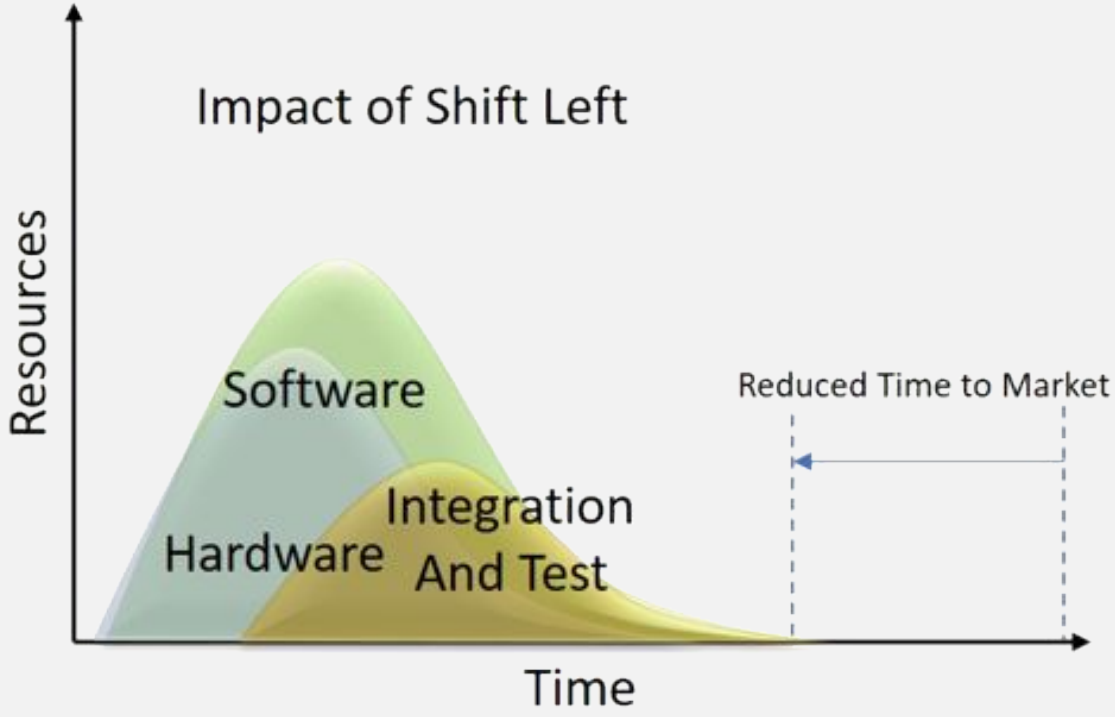


CYCON

Source: ARM Marketing material

Device manufacturers are “shifting left”

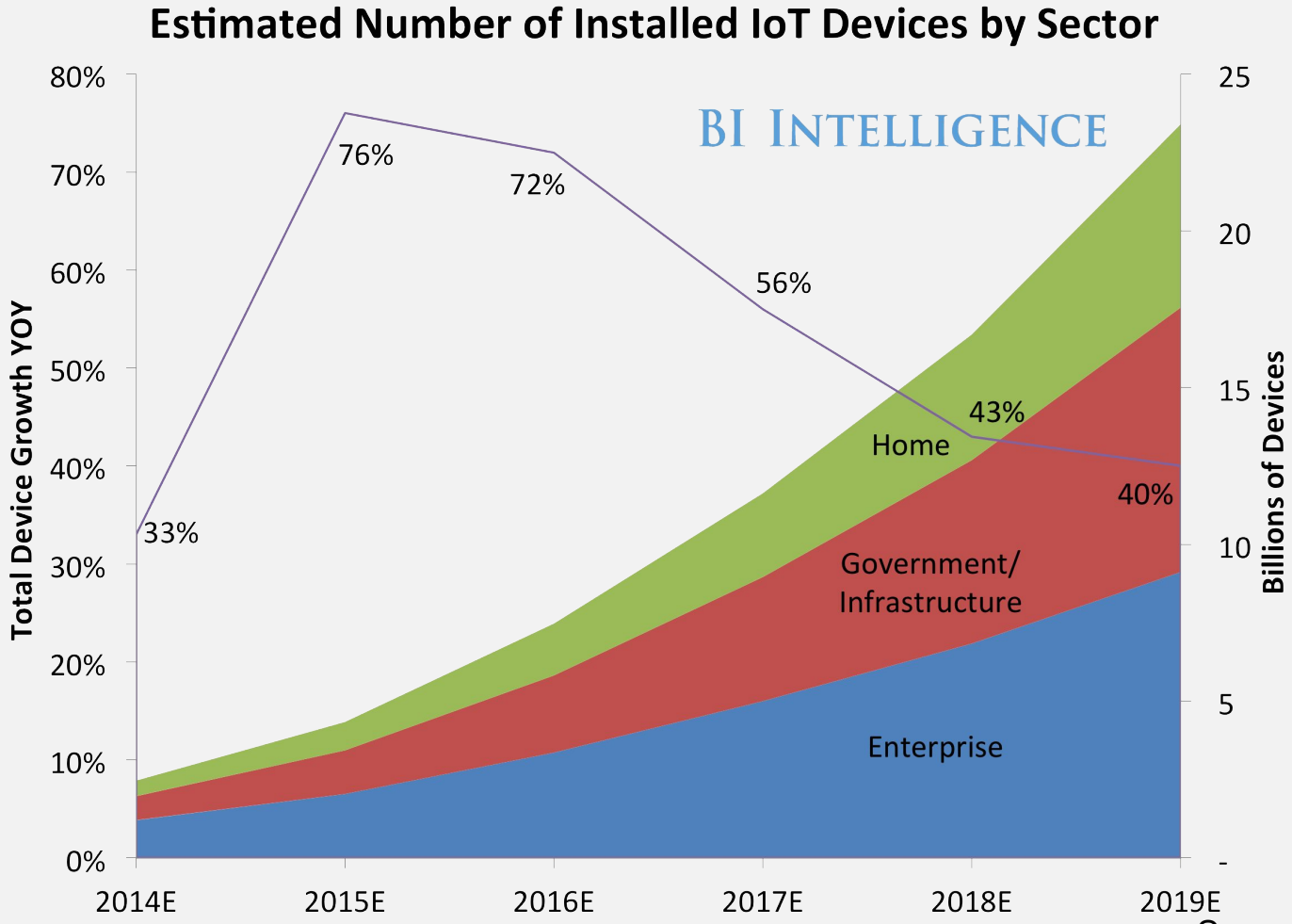




CYCON

Source: EE times

Security is improving, but insecure computing is growing faster



Source: BI intelligence

What is driving this complexity ?

The “anomaly of cheap complexity”.

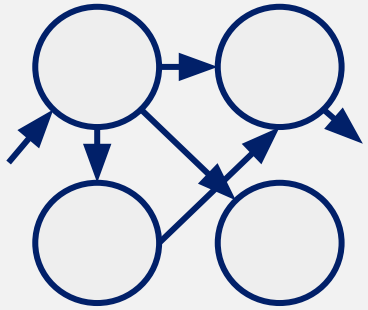
For most of human history, a more complex device was more expensive to build than a simpler device.

This is not the case in modern computing. **It is often more cost-effective to take a very complicated device, and make it simulate simplicity**, than to make a simpler device.

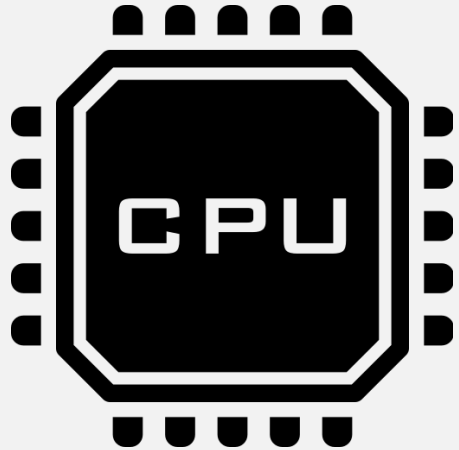
Simulate simplicity?

- You need a machine that does **something**.
- You could design a component that does that.
- Building that component and manufacturing it is expensive and complicated.
- Complex general-purpose CPUs are cheap. (Economies of scale and Moore's law)
- Software specializes a CPU that **could do anything** to become a device that **does something**.
  We **simulate** the simpler component.

CYCON



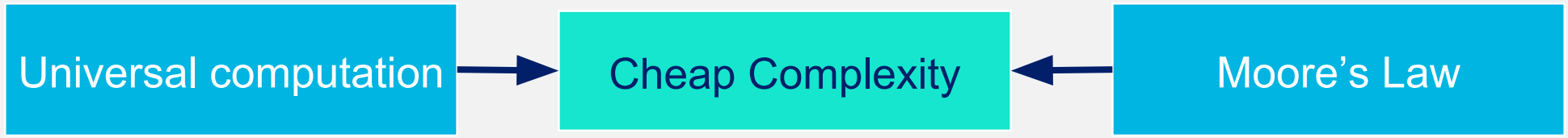
What you want: Restricted number of well-defined states.



What you have: Powerful general computer with theoretically finite but practically infinite states

What causes the anomaly ?

Two forces combine to create the “anomaly of cheap complexity”:



Universal computation means I can simulate one machine with another

In most practical scenarios I want a particular finite-state machine to perform a task.

Universal computation means I can take a general-purpose CPU, and simulate that finite-state machine.

I do not need to physically build the machine I want.
I can simply simulate it.

CYCON

Moore's Law generates massive economies of scale for general-purpose CPUs

ARM Cortex-M0 CPUs cost pennies.

Small development boards with CPUs that can boot a tiny Linux cost less than USD 2 wholesale.

CYCON

How does this affect security?

Some of the worst security challenges are:

1. Software security issues
2. Software supply chain issues
3. Hardware security / reliability & supply chain issues
4. Lack of device inspectability

How does this affect security?

Some of the worst security challenges are:

1. Software security issues
2. Software supply chain issues
3. Hardware security / reliability & supply chain issues
4. Lack of device inspectability

Why can an attacker hijack my device and make it do things it should not be able to do?

CYCON

How does this affect security?

Some of the worst security challenges are:

1. Software security issues
2. Software supply chain issues
3. Hardware security / reliability & supply chain issues
4. Lack of device inspectability

How did this research code someone wrote in two weeks 20 years ago end up in a billion devices?

CYCON

How does this affect security?

Some of the worst security challenges are:

1. Software security issues
2. Software supply chain issues
3. Hardware security / reliability & supply chain issues
4. Lack of device inspectability

Do I have to trust my CPU vendor?

CYCON

How does this affect security?

Some of the worst security challenges are:

1. Software security issues
2. Software supply chain issues
3. Hardware security / reliability & supply chain issues
4. Lack of device inspectability

In the real world, “possession” usually implies “control”.

In IT, “possession” and “control” are decoupled. Can I establish with certainty who is in control of a given device?

CYCON

Software security means limiting latent complexity

A general-purpose CPU can do pretty much anything.

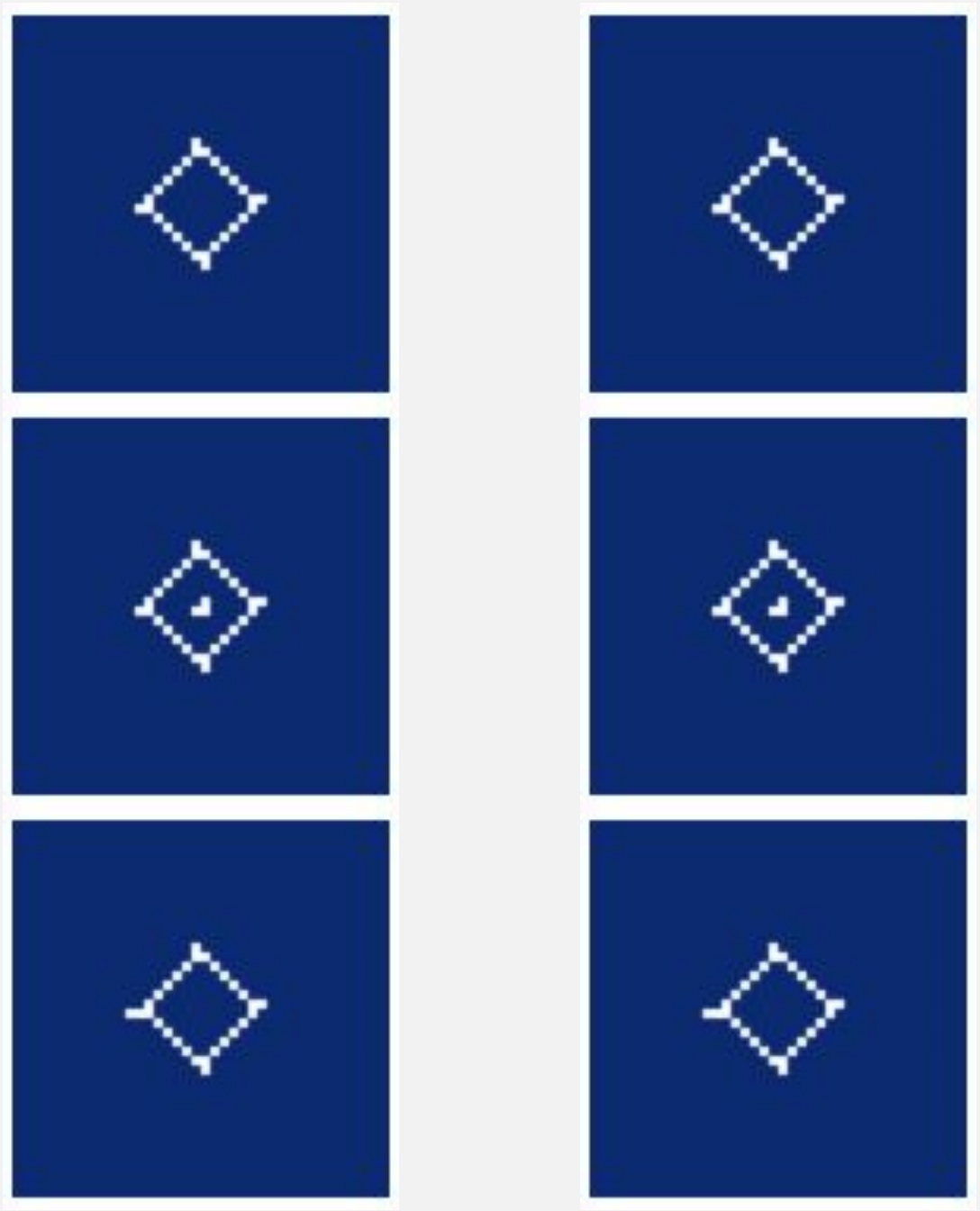
Building “security” into a system involves **limiting** what can happen in the system.

Software specializes a general-purpose CPU to perform a specific task, with a **strictly limited number of possible states and state transitions**.

CYCON

Dynamical systems

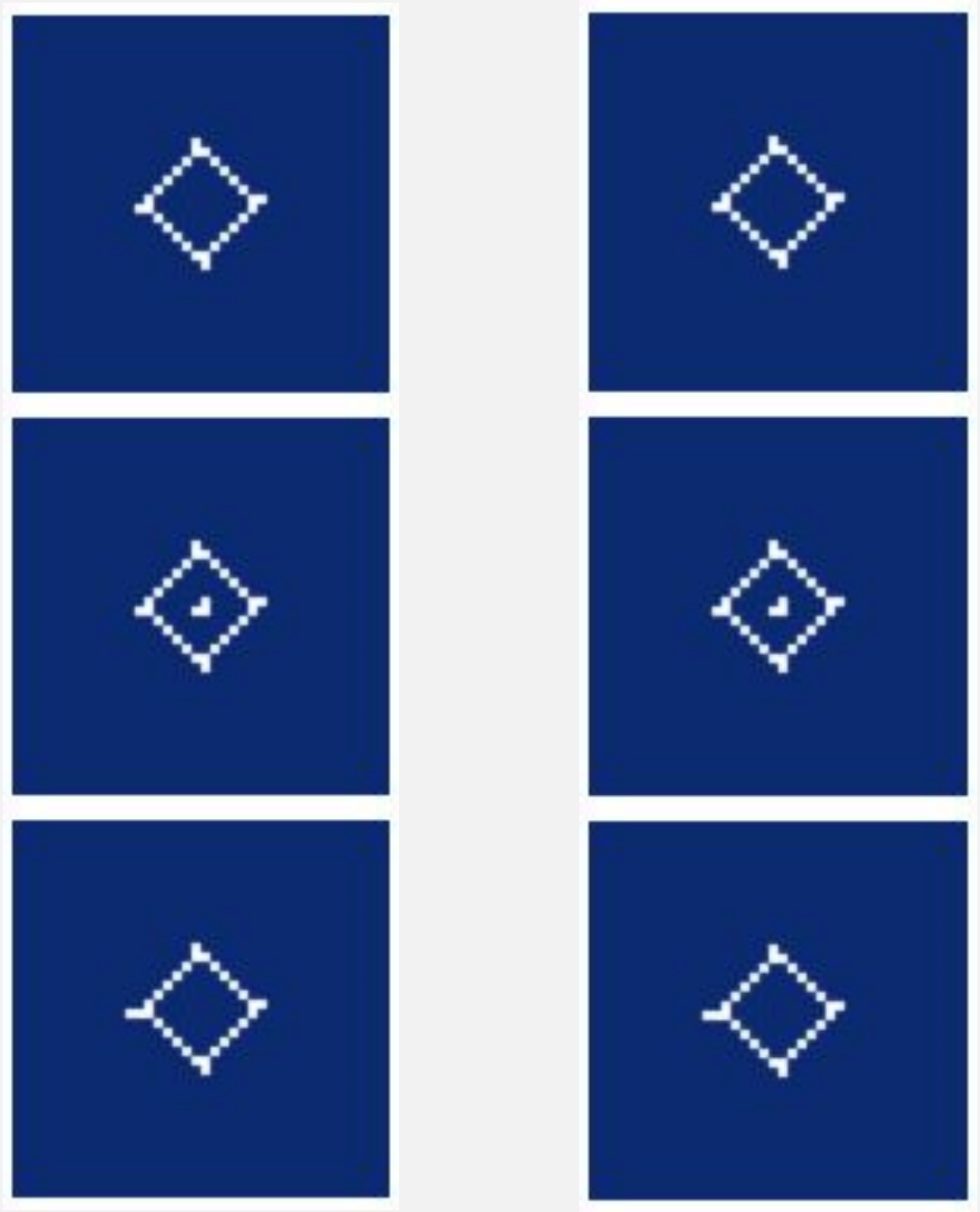
- Computers apply simple rules to a state to produce a new state.
- Simple rules repeatedly applied **can** lead to complex unforeseen behavior.
- Examples on the right: Apply the same rule repeatedly to three initial states with tiny input differences.
- Vastly different and unpredictable behavior **can** result, depending on the difference.



CYCON

Dynamical systems

- Computers apply simple rules to a state to produce a new state.
- Simple rules repeatedly applied **can** lead to complex unforeseen behavior.
- Examples on the right: Apply the same rule repeatedly to three initial states with tiny input differences.
- Vastly different and unpredictable behavior **can** result, depending on the difference.



CYCON

Complex behavior lurks just below the surface

When something goes wrong, the limits we imposed can go out of the window.

Operating on a broken input state, it is hard to predict what will happen.

Carefully chosen broken input states can propagate to all sorts of chaos.

Our experiences in security confirm this: Tiny issues / differences in initial state (single bitflips) can make a machine spin out of control, and the attacker can carefully control the escalating error to his advantage.

How does this affect security?

Some of the worst security challenges are:

1. Software security issues
2. Software supply chain issues
3. Hardware security / reliability & supply chain issues
4. Lack of device inspectability

Why can an attacker hijack my device and make it do things it should not be able to do?

It is **much** more cost-effective to build a general-purpose CPU into your device, and try to control the complexity, than to build something non-programmable / non-generic in the first place.

CYCON

Software is cheap

Once you have a general-purpose CPU, you can compile most of the software in the world for this platform.

Adding more features, more complexity, and more code is almost (if not quite) free.

Engineering

“How does one design an electric motor? Would you attach a bathtub to it, simply because one was available? Would a bouquet of flowers help? A heap of rocks? No, you would use just those elements necessary to its purpose and make it no larger than needed -- and you would incorporate safety factors. Function controls design.”

-- Prof. Bernardo de la Paz in *The Moon Is A Harsh Mistress (Robert A. Heinlein)*

Software makes adding bathtubs, bouquets of flowers, and rocks, almost free.

So that’s what we get.

How does this affect security?

Some of the worst security challenges are:

1. Software security issues
2. Software supply chain issues
3. Hardware security / reliability & supply chain issues
4. Lack of device inspectability

How did this research code someone wrote in two weeks 20 years ago end up in a billion devices?

It provided some marginal benefit, it was easy to re-use, it seemed to do the job most of the time, and it was less effort than to write the same functionality from scratch.

It was also, most likely, free.

CYCON

Tiny mistakes have big consequences

A modern high-end CPU has hundreds of millions or billions of transistors.

About 100.8 million per square millimeter.

A tiny number of malfunctioning transistors can have huge security consequences.

They cannot individually be inspected in any cost-effective manner.

The economics of chipmaking drive chipmakers to the edge

In order to continue Moore's Law, CPUs have to be manufactured on the boundary of what is "reliable".

Until recently, the link between "reliability" and "security" was poorly understood in a lot of the electrical engineering (and computer science) community.

The hardware maker has financial incentive to ship the 'most unreliable CPU that can not be detected as unreliable'.

CYCON

You need to trust your chip manufacturer

We do not know how to write real-world programs that can operate on untrusted hardware.

Any tiny, innocent-looking, intermittent and probabilistic misbehavior can, under current architectures, have disastrous security implications.

How does this affect security?

Some of the worst security challenges are:

1. Software security issues
2. Software supply chain issues
3. Hardware security / reliability & supply chain issues
4. Lack of device inspectability

Do I have to trust my CPU vendor?

You will have to trust whoever fabricates your CPUs, at least given today's software and systems.
Also: The vendor makes more money by shipping borderline cases.

CYCON

Scaling, firmware engines, and inspectability

Scaling has yielded the performance gains of the last decades, but scaling made physical inspection impossible.

Universal computation has replaced many formerly-simpler components in your computer with full CPUs + firmware - usually without mechanism for inspection.

Current approach for firmware security is based on “ensuring nobody can get in” (code signing), but transient faults can be locally induced to bypass, and signing keys get stolen with regularity.

Scaling, firmware engines, and inspectability

Nobody has a good way to assure that a given device is reset into a “known-good” state, **especially** if the hardware was under physical attacker control.

We can’t check the transistors.

We can’t check the firmware origin.

Establishing “who is in control” is near-impossible against strong adversaries.

How does this affect security?

Some of the worst security challenges are:

1. Software security issues
2. Software supply chain issues
3. Hardware security / reliability & supply chain issues
4. Lack of device inspectability

In the real world, “possession” usually implies “control”.

In IT, “possession” and “control” are decoupled. Can I establish with certainty who is in control of a given device?

Not with current technology and processes.
Too many uninspectable components with impact that can spin out of control.

CYCON

What does this mean?

The pervasive insecurity of modern IT infrastructure is driven by the same forces that make IT a success in the first place:

Universal computation & Moore's law provides exponential cost reduction.

Genericity + economies of scale make complexity cheaper than simplicity.

Near-zero short-term marginal cost to add software complexity.

So what’s next?

There is a lot of research and engineering ahead.

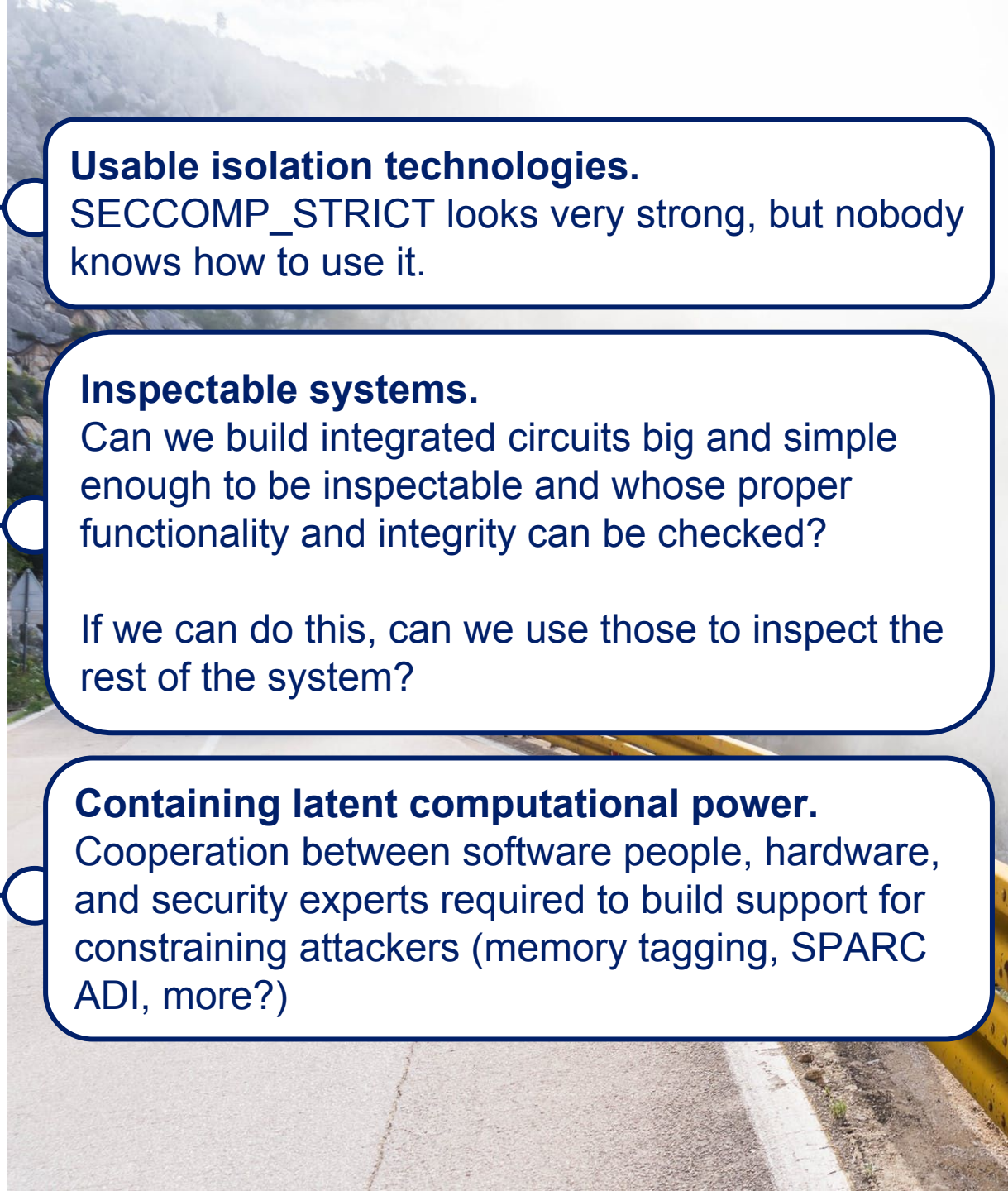
- How do we contain untrusted software components but still make use of them?
- How do we build systems that we can inspect?
- How do we build systems that do not easily spin out of control, or that can detect early when they do?
- How do we do this without sacrificing either the progress of Moore’s Law, nor the economies of scale and other advantages of generic programmability?

CYCON

So what's next?

There is a lot of research and engineering ahead.

- How do we contain untrusted software components but still make use of them?
- How do we build systems that we can inspect?
- How do we build systems that do not easily spin out of control, or that can detect early when they do?
- How do we do this without sacrificing either the progress of Moore's Law, nor the economies of scale and other advantages of generic programmability?



Usable isolation technologies.
SECCOMP_STRICT looks very strong, but nobody knows how to use it.

Inspectable systems.
Can we build integrated circuits big and simple enough to be inspectable and whose proper functionality and integrity can be checked?

If we can do this, can we use those to inspect the rest of the system?

Containing latent computational power.
Cooperation between software people, hardware, and security experts required to build support for constraining attackers (memory tagging, SPARC ADI, more?)

CYCON

How to pay for it?

- Securable systems require co-design of hardware & software.
- Re-architecting legacy infrastructure for security is not economical.
- How would we get there?

CYCON

End-of-Moore as opportunity

- Single-core scaling is dead.
- CPU-architecture and programming models are in flux for the first time since the 1980’s
- Computing is getting re-architected as we speak, for good economic reasons. Both hardware and software.
- Historic opportunity to influence design of computing for next N years.

CYCON


Questions?