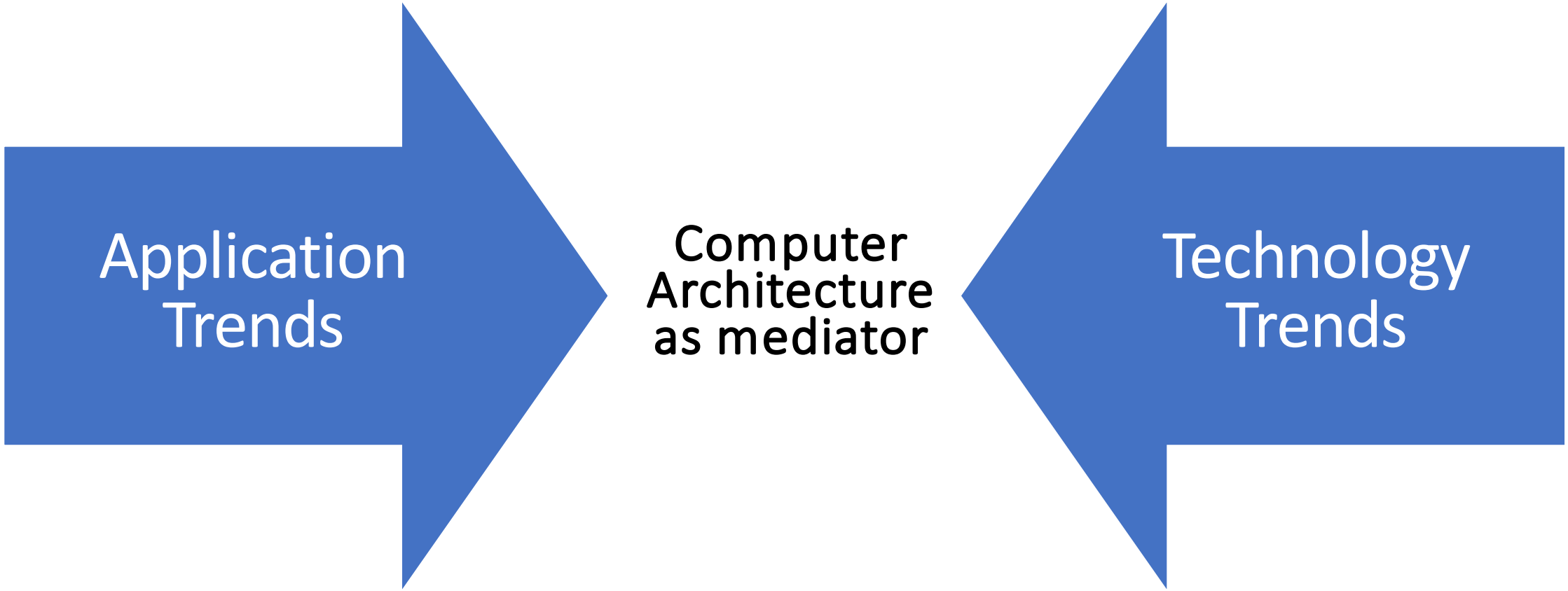
New Metrics & Models for a Post-ISA Era

Margaret Martonosi
H. T. Adams ‘35 Professor
Dept. of Computer Science
Princeton University

Computer Architecture…



This Talk's Fundamental Questions

- What are important design goals for today's computer systems?
- How can formal interface specs and models help to achieve them?
- What metrics do we use to assess progress in reaching them?

Moore’s Law

1965

Cramming more components onto integrated circuits

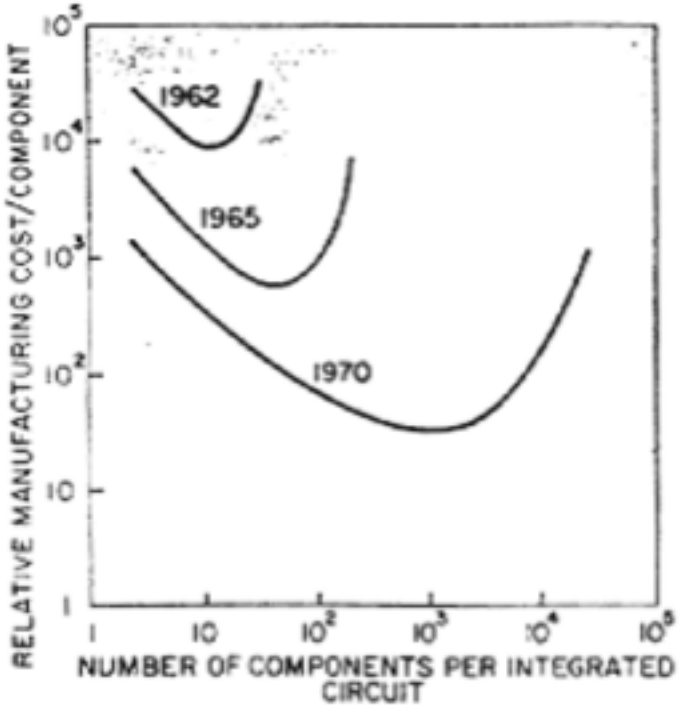
With unit cost falling as the number of components per circuit rises, by 1975 economics may dictate squeezing as many as 65,000 components on a single silicon chip

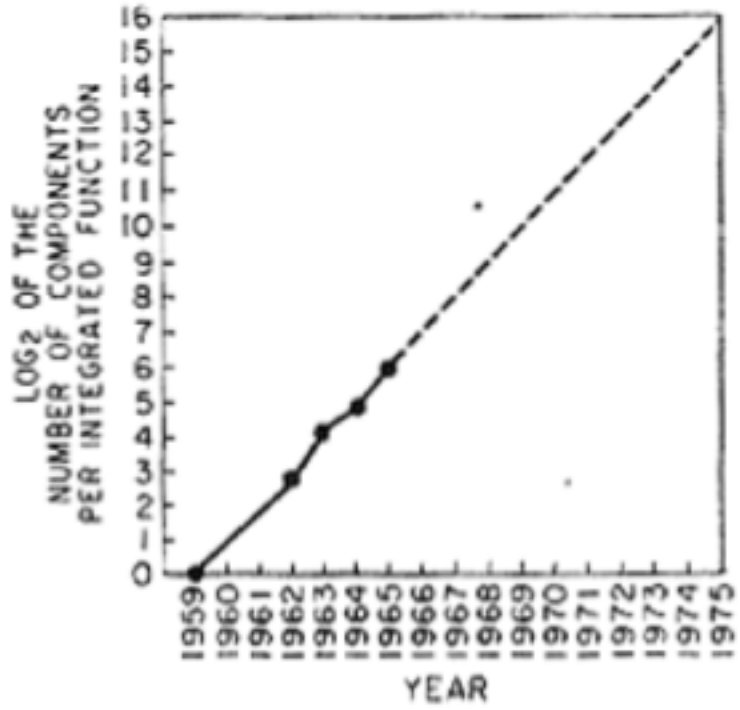
By Gordon E. Moore

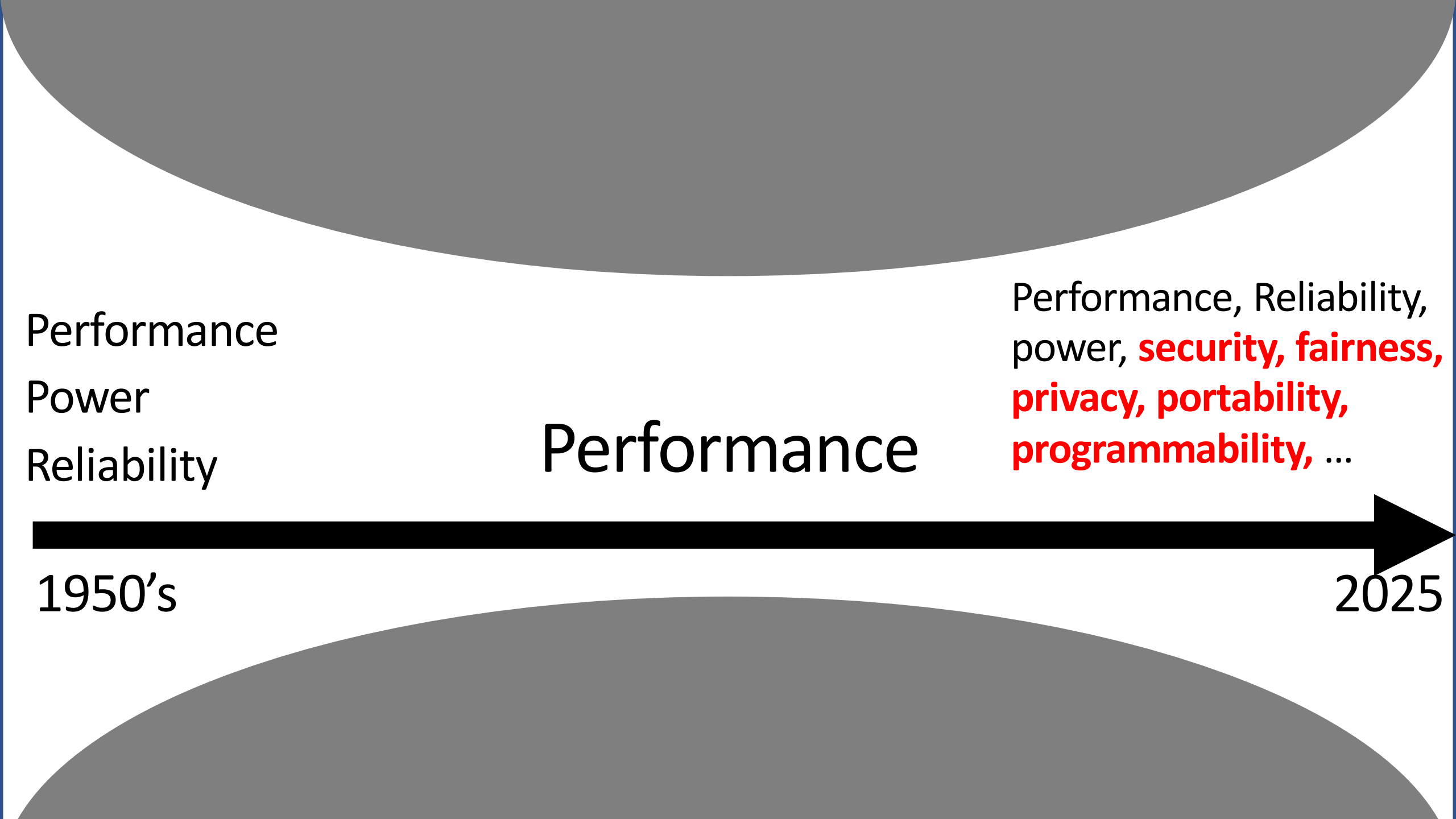
Director, Research and Development Laboratories, Fairchild Semiconductor division of Fairchild Camera and Instrument Corp.

Moore's Law

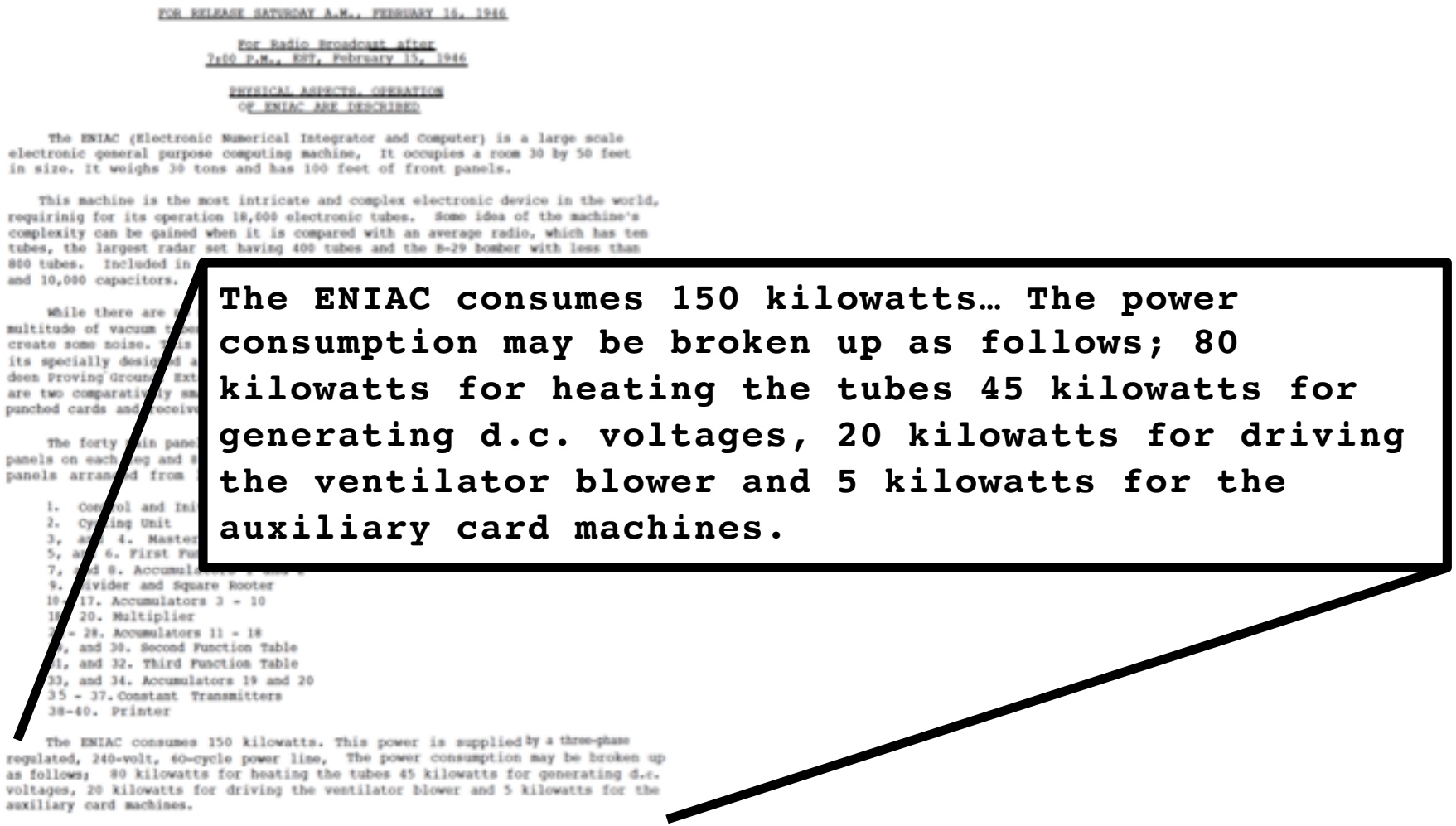
1965






Performance
Power
Reliability
Performance
Performance, Reliability, power, security, fairness, privacy, portability, programmability, …
1950’s
2025

Early Challenges 1/3: Power



FOR RELEASE SATURDAY A.M., FEBRUARY 16, 1946

For Radio Broadcast after
7:00 P.M., EST, February 15, 1946

PHYSICAL ASPECTS, OPERATION
OF ENIAC ARE DESCRIBED

The ENIAC (Electronic Numerical Integrator and Computer) is a large scale electronic general purpose computing machine. It occupies a room 30 by 50 feet in size, it weighs 30 tons and has 100 feet of front panels.

This machine is the most intricate and complex electronic device in the world, requiring for its operation 18,000 electronic tubes. Some idea of the machine's complexity can be gained when it is compared with an average radio, which has ten tubes, the largest radar set having 400 tubes and the B-29 bomber with less than 800 tubes. Included in
and 10,000 capacitors.

While there are
multitude of vacuum
create some noise.
its specially design
deen Proving Grou
are two comparativ
punched cards and

The forty
panels on each
panels arran

1. Co ol and In
2. Cy ing Unit
3, a 4. Maste
5, a 6. First Fu
7, 8. Accumul
9. ivider and Square Rooter
10 17. Accumulators 3 - 10
1 20. Multiplier
2 28. Accumulators 11 - 18
, and 30. Second Function Table
1, and 32. Third Function Table
3, and 34. Accumulators 19 and 20
35 - 37. Constant Transmitters
38-40. Printer

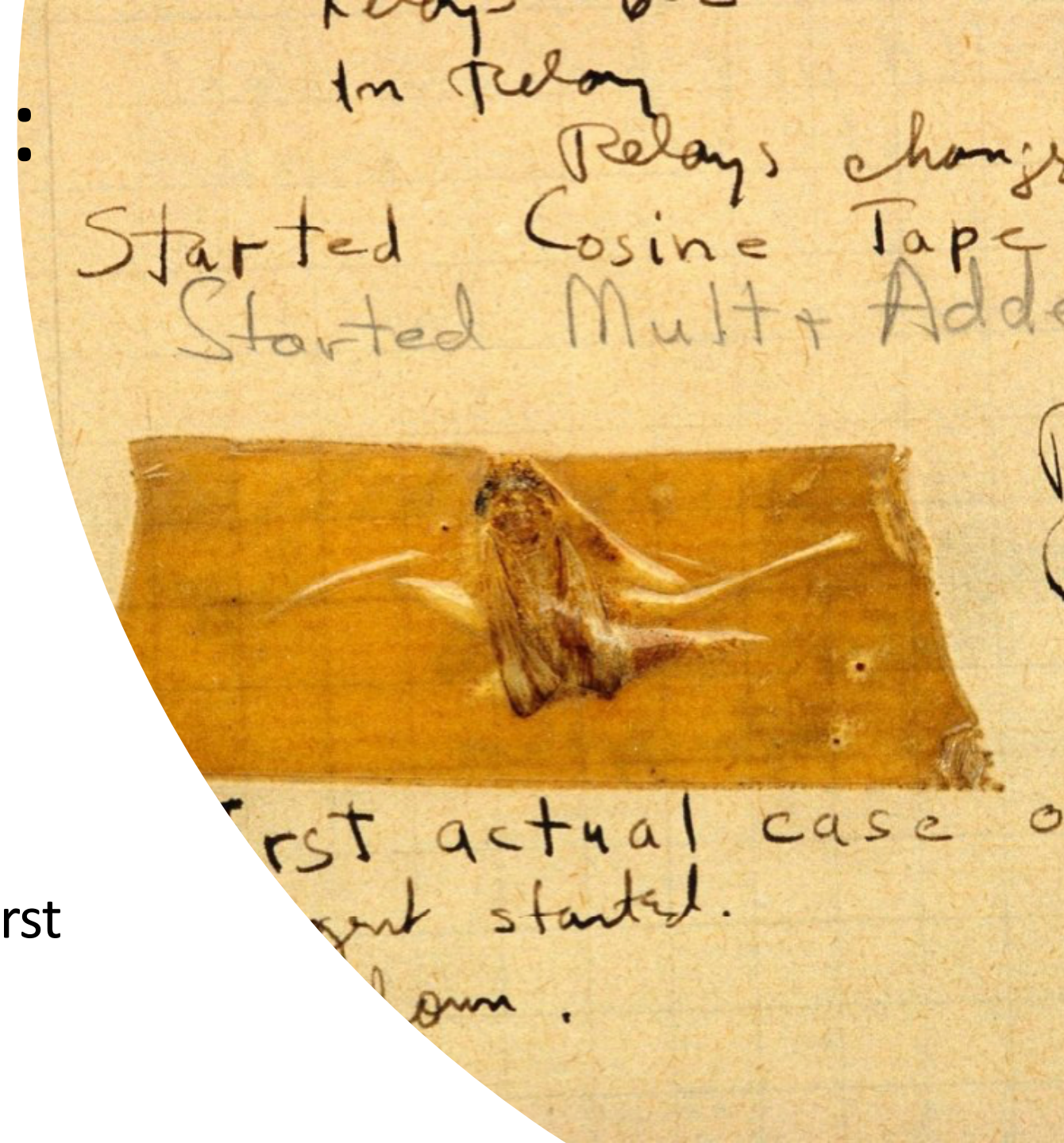
The ENIAC consumes 150 kilowatts. This power is supplied by a three-phase regulated, 240-volt, 60-cycle power line. The power consumption may be broken up as follows; 80 kilowatts for heating the tubes 45 kilowatts for generating d.c. voltages, 20 kilowatts for driving the ventilator blower and 5 kilowatts for the auxiliary card machines.

The ENIAC consumes 150 kilowatts… The power consumption may be broken up as follows; 80 kilowatts for heating the tubes 45 kilowatts for generating d.c. voltages, 20 kilowatts for driving the ventilator blower and 5 kilowatts for the auxiliary card machines.

Source: Original ENIAC press release. Feb, 1946

Early Challenges 2/3: Hardware Reliability

Grace Murray Hopper and the first computer bug



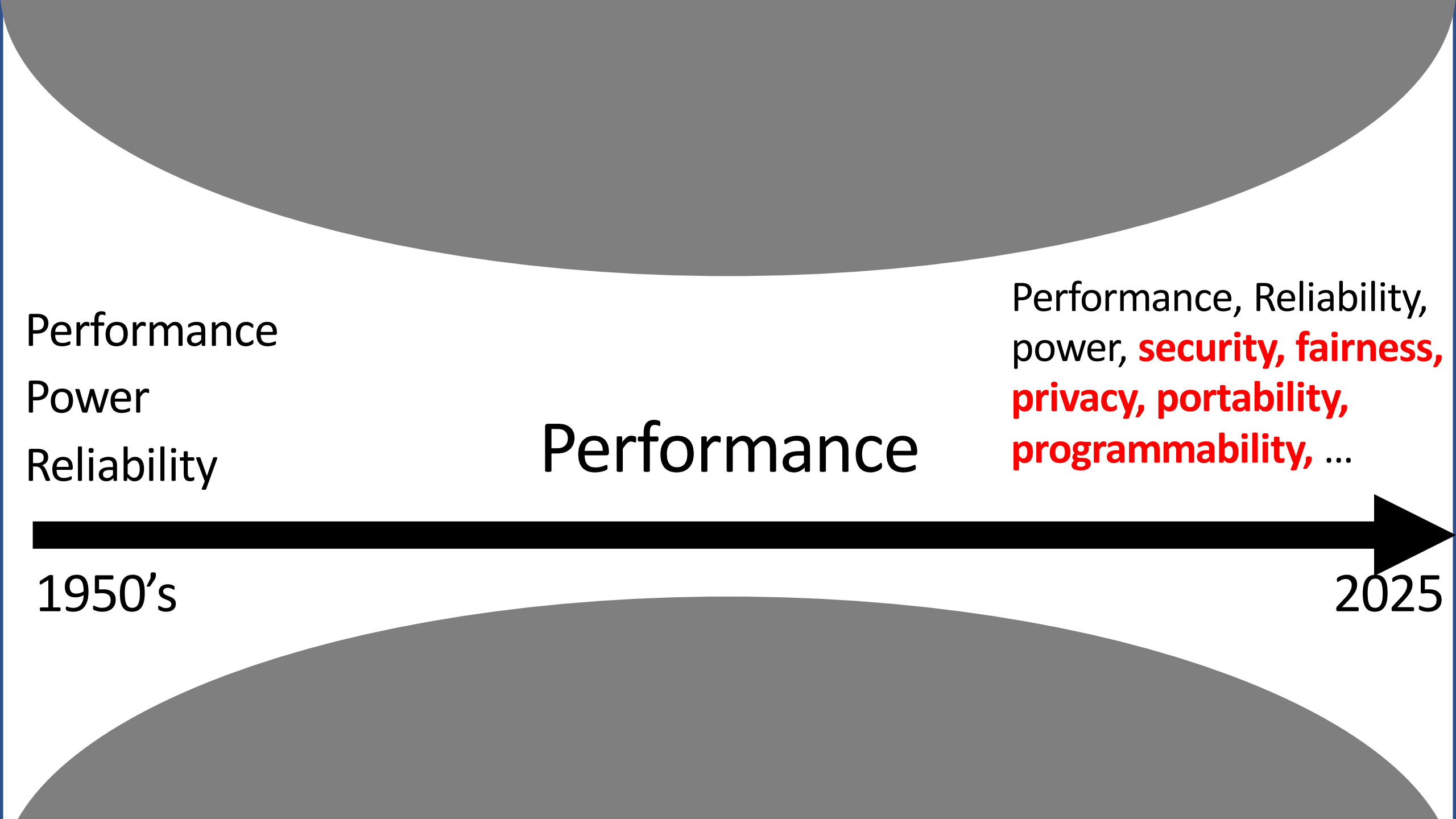
Early Challenges 3/3: Software Debugging…

"As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs."

Sir Maurice Wilkes

Early computer architect; 1967 Turing Award Winner

[Written in 1985, reflecting back on programming for the 1949 EDSAC]

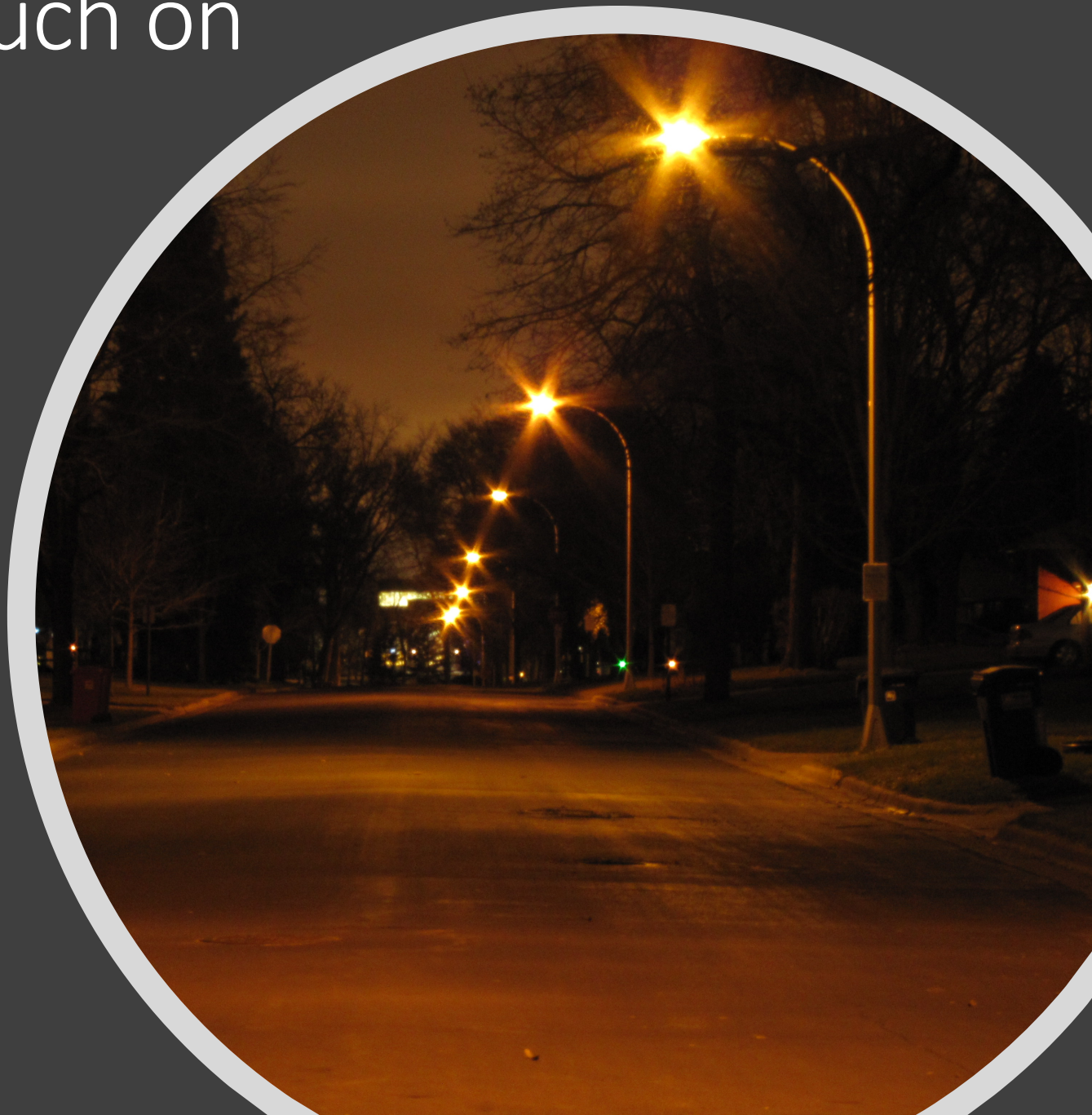

Performance
Power
Reliability
Performance
Performance, Reliability, power, security, fairness, privacy, portability, programmability, …
1950’s
2025

Why don’t we focus as much on these broader goals?

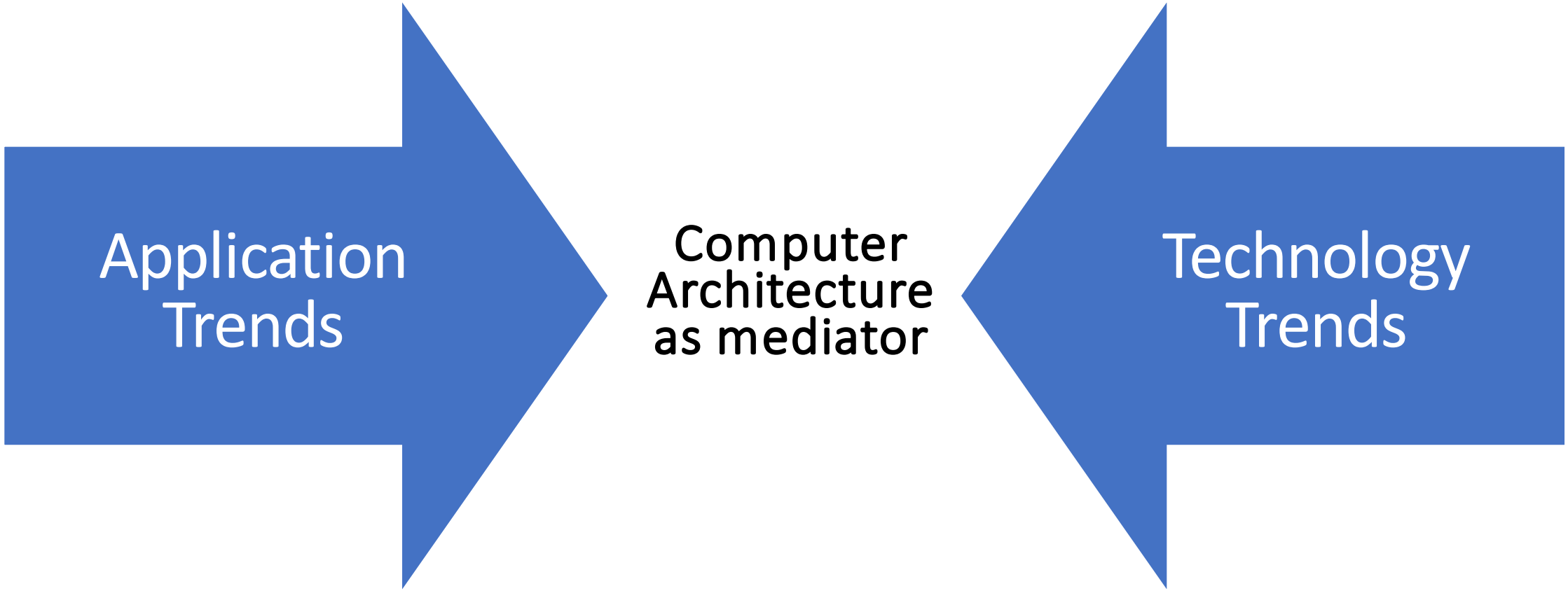
“Streetlight Effect”

Often our design goals are being set by what we can measure or model (easily)

Rather than by what we actually want to achieve…



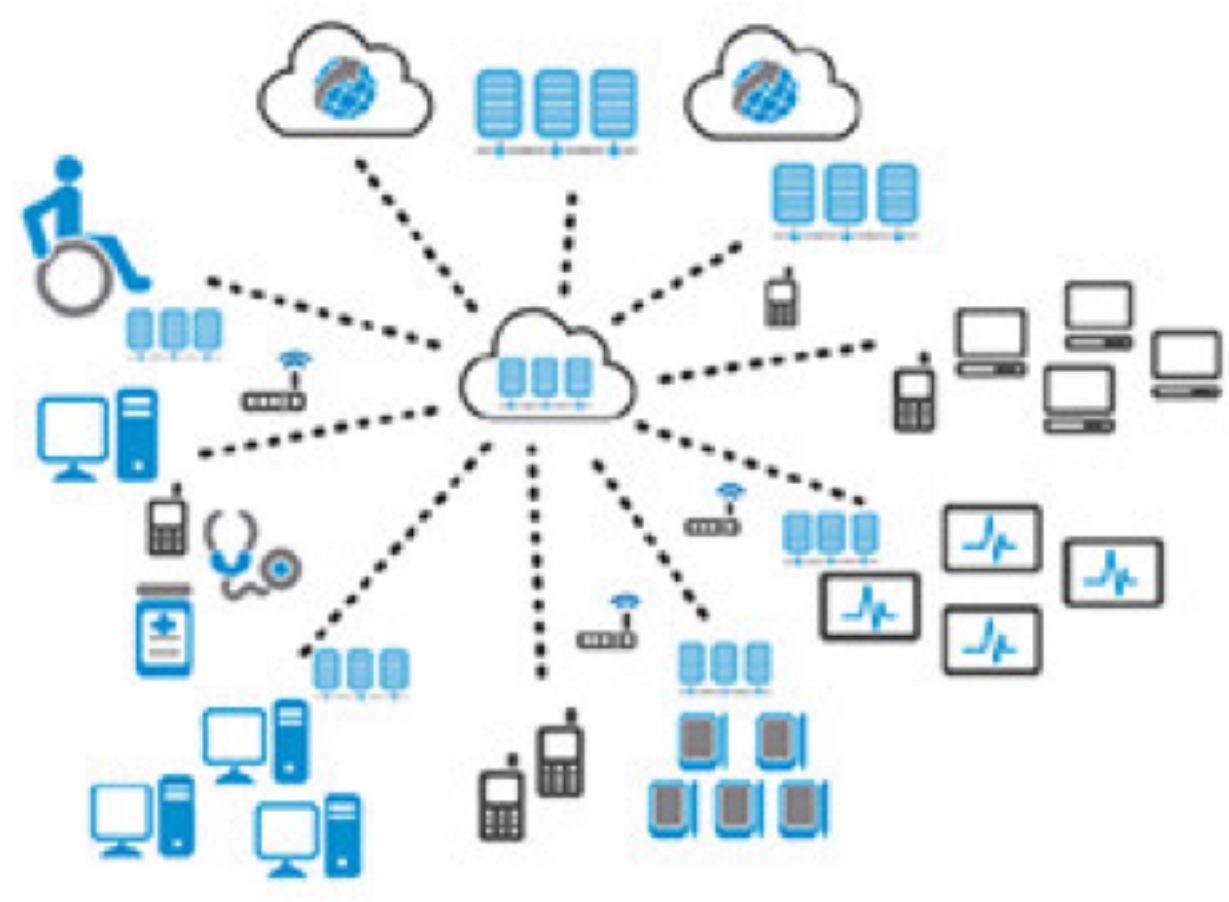
Computer Architecture…



Application Trends:
Sensory Swarm to Cloud Infrastructure

- Lots of data
- Highly-distributed
- Communication-intensive

- Diversity OF devices
- Diversity WITHIN devices



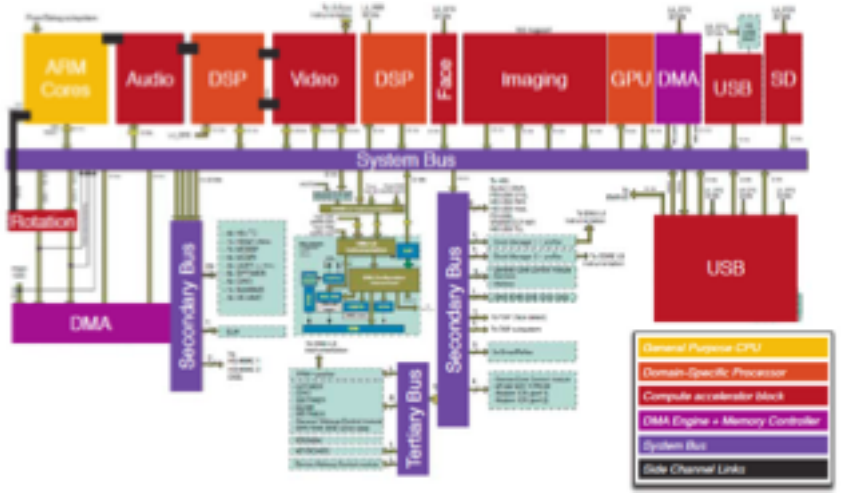
Which brings us here…

Heterogeneity “In the Large”:

- 4000+ distinct Android devices
- IoTs even more diverse
- Mobile/cloud adaptation and migration

Heterogeneity “In the Small”

- 6-10 ISAs on chip + Accelerators
- Memory, Data Heterogeneity
- Memory Consistency Models



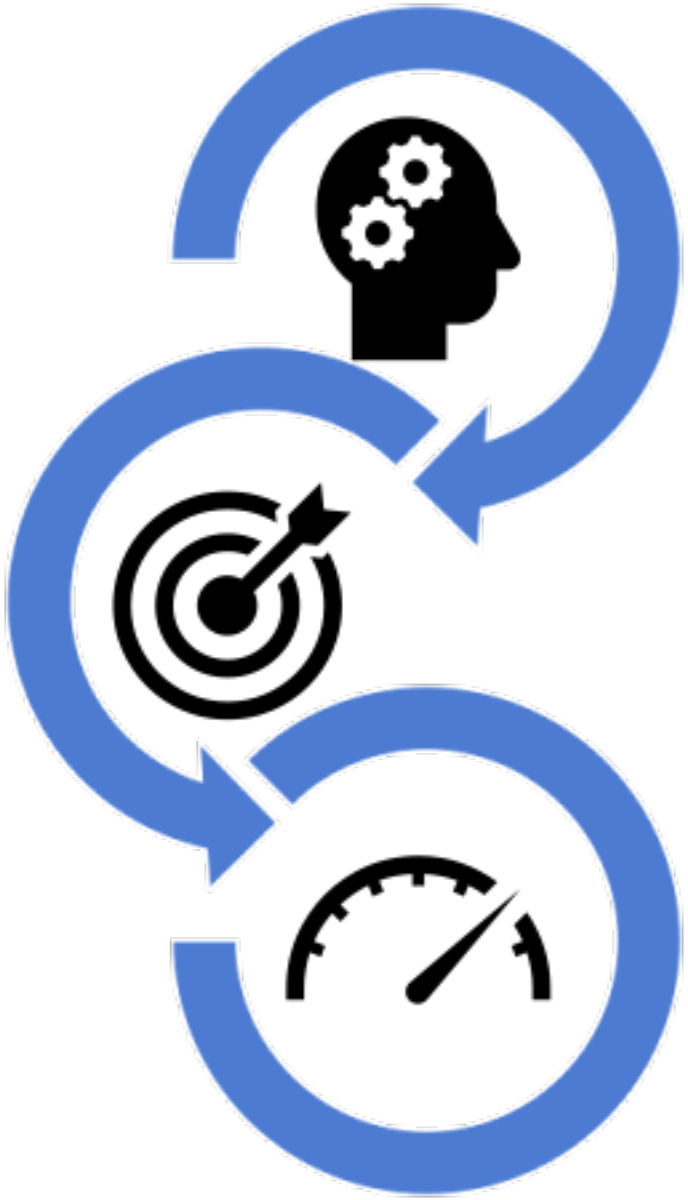
Entering A Post-ISA World

- **ISAs still useful, but little/no relevance as abstraction layer.**
  - Apple A8 and beyond: >50% of chip area is accelerators that have no ISA.
  - NVIDIA PTX vs. SASS: ISA hidden under other layers.
- **Challenge**: Lack of durable hardware-software interface.
- **Opportunity**: Now have the chance to develop design practices around new interface layers

Design Goals

Models & Interfaces

Metrics



Examples

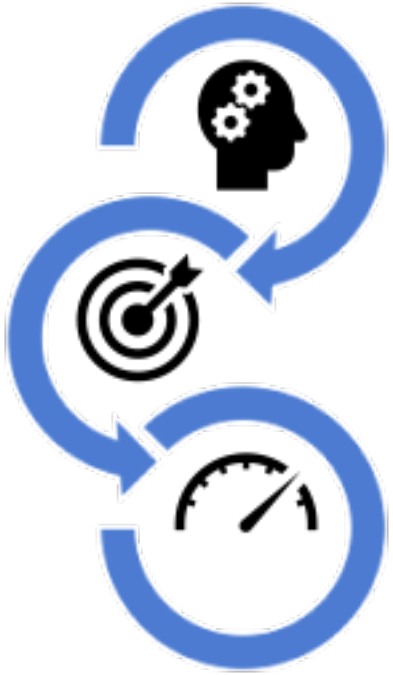
- **Power**: Power-efficient Architecture & Models
- **Correctness**: Concurrency -> Memory Models
- **Security**: Formal Methods for Secure Design

Examples

- **Power**: Power-efficient Architecture & Models
- **Correctness**: Concurrency -> Memory Models
- **Security**: Formal Methods for Secure Design

Example 1: Power-Efficient Computing, Circa 1999

- Computer Architects didn't measure, model, or design for power.
- No existing early-stage (architecture-level) power models.
- What power models existed were all late in the design process (Design automation, VLSI Circuits)



- **Idea: Power is important, so start by using other measurements as a "proxy" for power.**

From idea to “new” metric

The Idea:

- Narrow bitwidth values are common -> Perform narrower ops where possible

The Result:

- >50% reduction in integer ALU power
- Patent + Industry use

Bigger Picture:

- Spurred development of better architecture-level power models. (Wattch, ISCA 2000)
- **Now: Power is key metric in arch and systems design.**

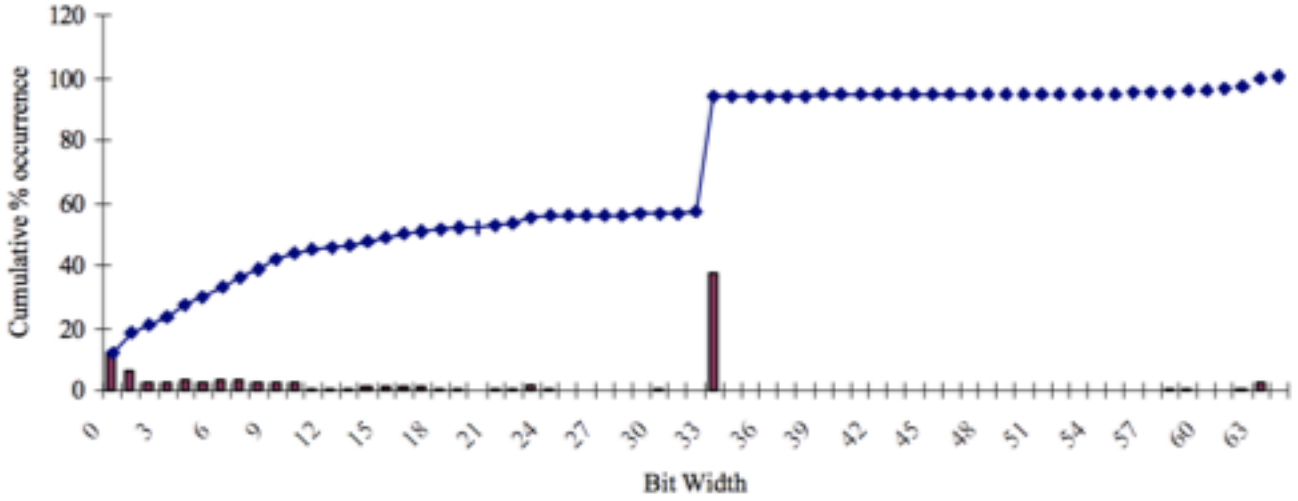


Figure 1 – Bitwidths for SPECint95 on 64-bit Alpha.

[Brooks & Martonosi. HPCA 1999. 2018 Test-of-Time Award]

[Brooks, Tiwari & Martonosi. ISCA 2000. 2015 Test-of-Time Award]

Examples

- **Power**: Power-efficient Architecture & Models
- **Correctness**: Concurrency -> Memory Models
- **Security**: Formal Methods for Secure Design

Example 2: Correctness
Memory Model Verification

- Sequential Consistency [Lamport 1979]: execution is the same as if:
  (R1) Memory ops of each processor appear in program order
  (R2) Memory ops of all processors were executed in some global sequential order

Program

Thread 0	Thread 1
x=1	y=1
r1=y	r2=x

Legal Executions

x=1	x=1	x=1	y=1	y=1	y=1
r1=y	y=1	y=1	r2=x	x=1	x=1
y=1	r1=y	r2=x	x=1	r2=x	r1=y
r2=x	r2=x	r1=y	r1=y	r1=y	r2=x

TSO Memory Consistency Model

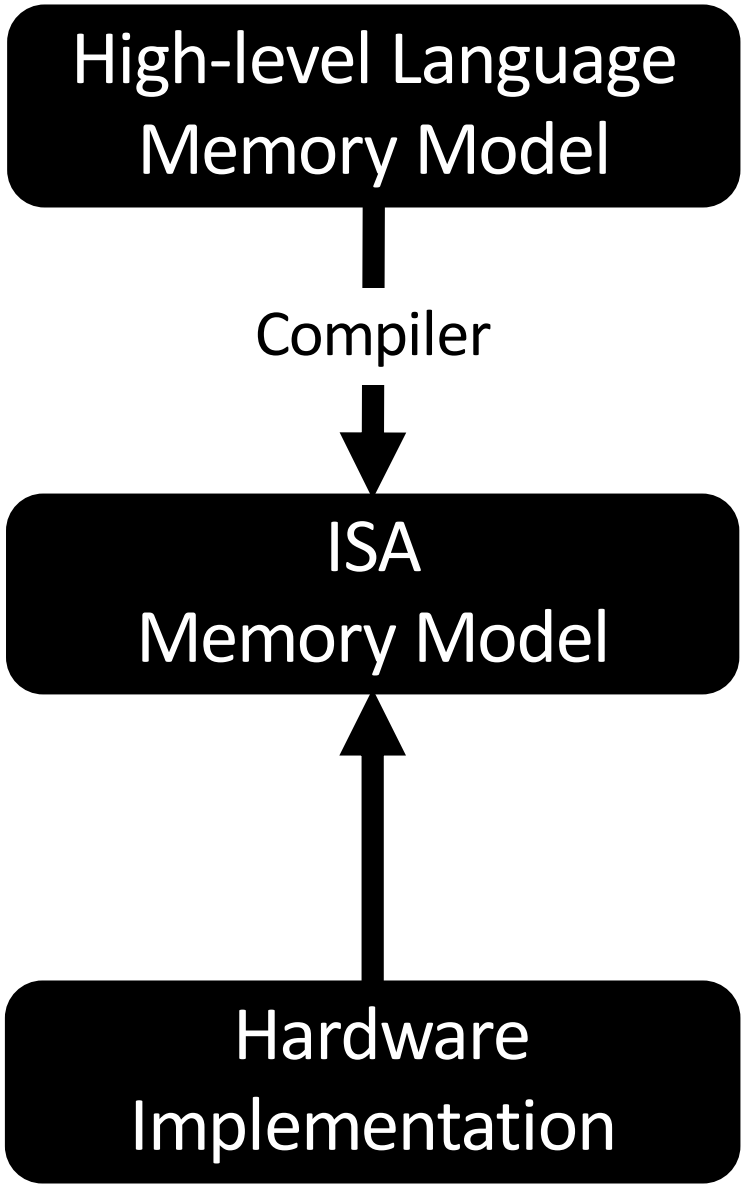
In a nutshell: MCMs are the spec of what value will be returned when your program does a load.
=> Important to specify and verify

8.2.2 Memory Ordering in

The Intel Core 2 Duo, Int
memory-ordering
can be cha

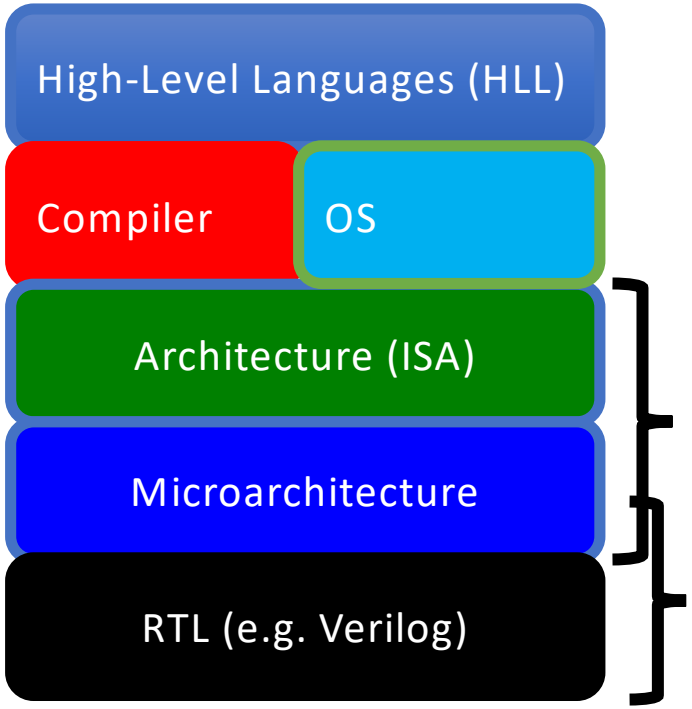
- W
  - —
  - — s with the non-temporal move instructions (MOVNTI, MOVNTQ, M OVNTPD); and
  - — str (see Section 8.2.4.1).
- Reads n e reordered with older writes to different locations but not with older writes to the same location.

Why is Memory Model Verification Important?



- **What can go wrong?**
  - Incorrect software answers
  - Unreliable operation
  - Security exploits
- **What can go wrong?**
  - Ill-specified High-level language memory model
  - Inadequate ISA specification
  - Incorrect HLL→ISA compilation
  - Incorrect hardware implementation

The Check Suite: An Ecosystem of Verification Tools



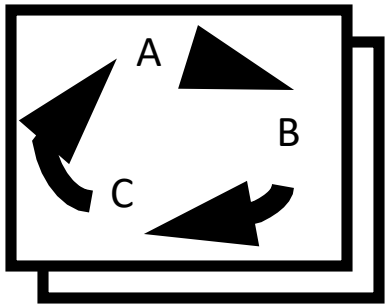
TriCheck [ASPLOS '17] [IEEE MICRO Top Picks]

COATCheck [ASPLOS '16] [IEEE MICRO Top Picks]

PipeCheck [Micro-47] [IEEE MICRO Top Picks]

CCICheck [Micro-48] [Nominated for Best Paper Award]

RTLCheck [Micro-50] [IEEE MICRO Top Picks Honorable Mention]



Our Approach

- Axiomatic specifications -> Happens-before graphs
- Check Happens-Before Graphs via Efficient SMT solvers
  - Cyclic => A->B->C->A… Can't happen
  - Acyclic => Scenario is observable

Motivation

ARM Read-Read Hazard: A Tale of Many Interfaces

ARM Read-Read Hazard

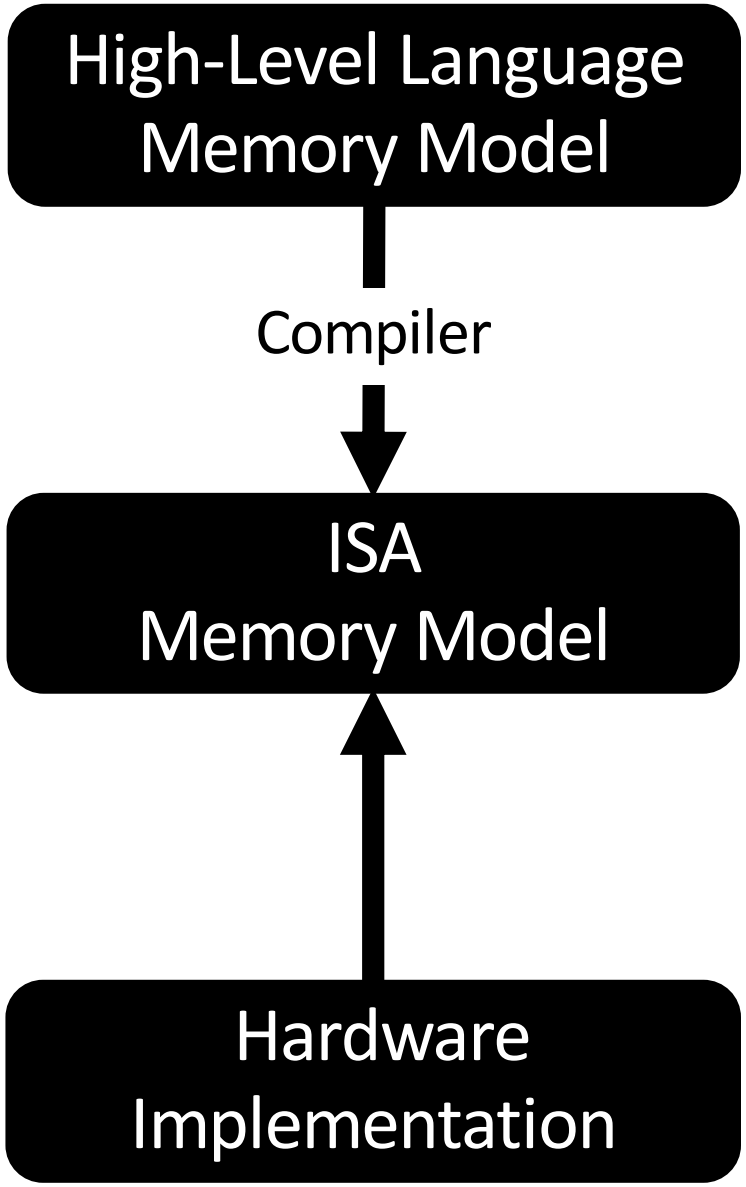
- ARM ISA spec ambiguous regarding same-address Ld→Ld ordering:
  - Compiler's job? Hardware job?
- C/C++ variables with atomic type require same-addr. Ld→Ld ordering
- ARM issued errata1:
  - Rewrite compilers to insert fences (ordering instructions) with performance penalties
- ARM ISA had the right ordering instructions – just needed to use them.

```
std::atomic<int> z = {0};
std::atomic<int> *y = {&z};

void thread0()
{
    z.store(1, std::memory_order_relaxed);
    int r0 = y->load(std::memory_order_relaxed);
    int r1 = z.load(std::memory_order_relaxed);
    if(r0 != r1)
        z.store(3, std::memory_order_relaxed);
}

void thread1()
{
    z.store(2, std::memory_order_relaxed);
}
```

Original: Alglave 2001

Google Nexus 6: http://check.cs.princeton.edu/tutorial_extras/SnapVideo.mov

ARM Read-Read Hazard




```
std::atomic<int> z = {0};
std::atomic<int> *y = {&z};

void thread0()
{
    z.store(1, std::memory_order_relaxed);
    int r0 = y->load(std::memory_order_relaxed);
    int r1 = z.load(std::memory_order_relaxed);
    if(r0 != r1)
        z.store(3, std::memory_order_relaxed);
}

void thread1()
{
    z.store(2, std::memory_order_relaxed);
}
```

C11 Source Code

C11/C++11	ARMv7
st(rlx)	STR
ld(rlx)	LDR
ld(acq)	LDR; DMB
…	…

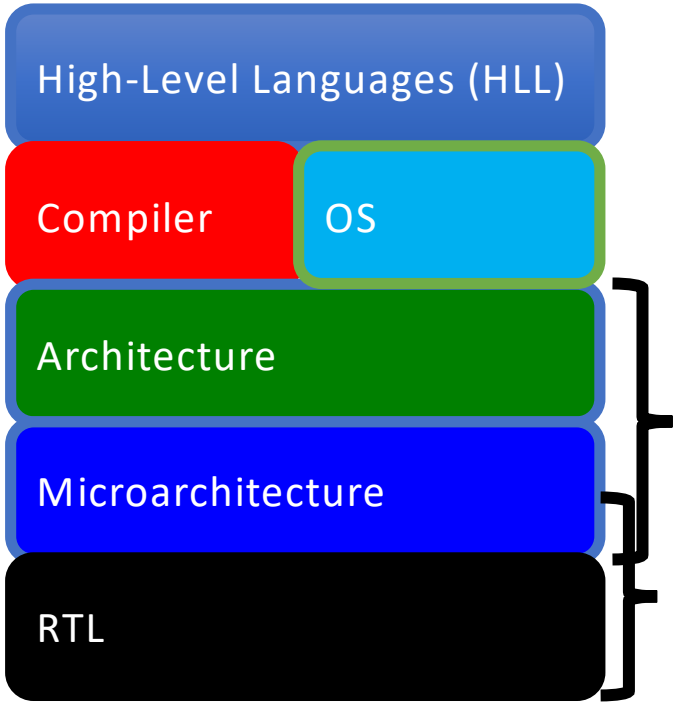
"Compiler Mappings"

C0	C1
ST [data]←1	ST [data]←2
LD [ptr]→r0	
LD [r0]→r1	
LD [data]→r2	

Assembly Code

Run on specific CPU: ARM Cortex A9

TriCheck: Linking HLLs->ISA->Microarchitecture



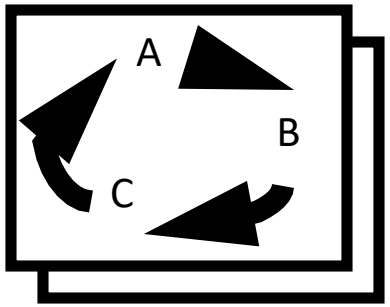
TriCheck [ASPLOS '17] [IEEE MICRO Top Picks]

COATCheck [ASPLOS '16] [IEEE MICRO Top Picks]

PipeCheck [Micro-47] [IEEE MICRO Top Picks]

CCICheck [Micro-48] [Nominated for Best Paper Award]

RTLCheck [Micro-50] [IEEE MICRO Top Picks Honorable Mention]



Our Approach

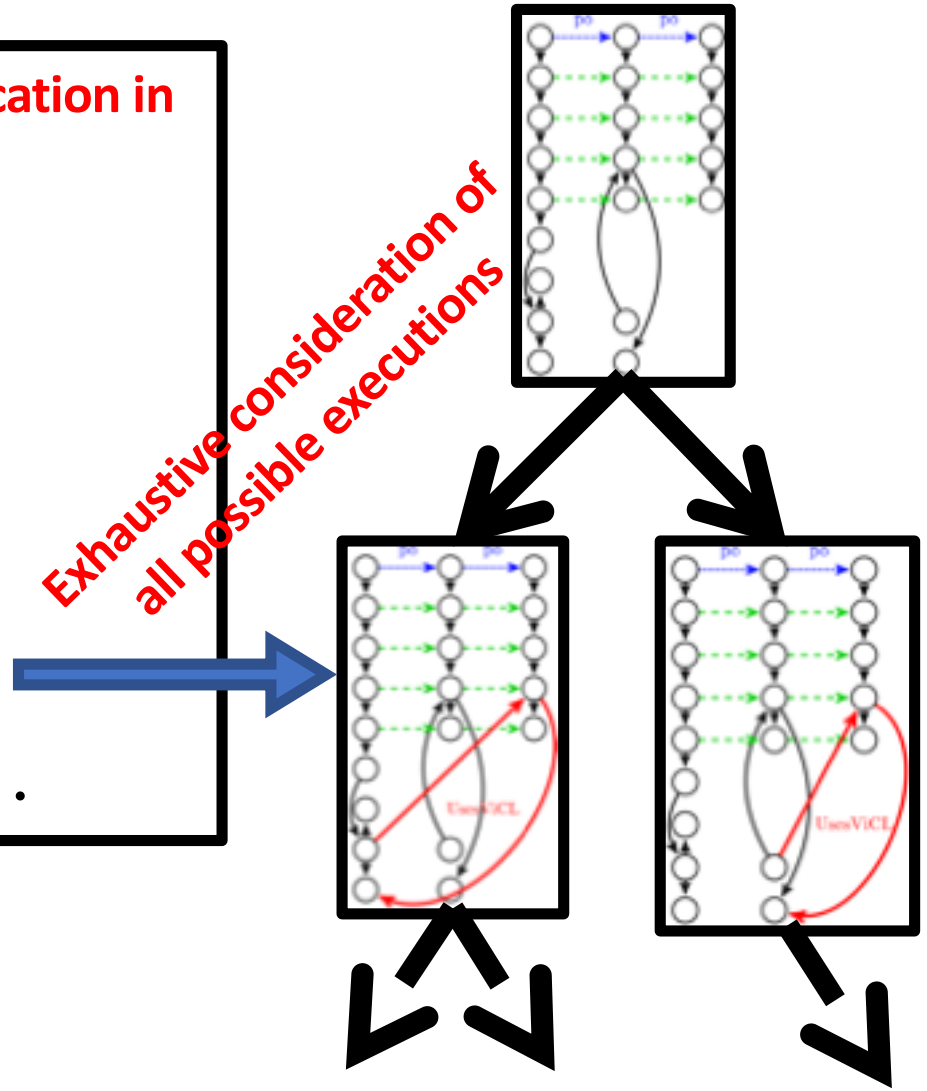
- Formal specifications -> Happens-before graphs
- Check Happens-Before Graphs via Efficient SMT solvers
  - Cyclic => A->B->C->A… Can't happen
  - Acyclic => Scenario is observable

Check: Formal, Axiomatic Models and Interfaces

```
Axiom "PO_Fetch":
forall microops "i1",
forall microops "i2",
SameCore i1 i2 /\ ProgramOrder i1 i2 =>
  AddEdge ((i1, Fetch), (i2, Fetch), "PO").

Axiom "Execute_stage_is_in_order":
forall microops "i1",
forall microops "i2",
SameCore i1 i2 /\
 EdgeExists ((i1, Fetch), (i2, Fetch)) =>
   AddEdge ((i1, Execute), (i2, Execute), "PPO").
```

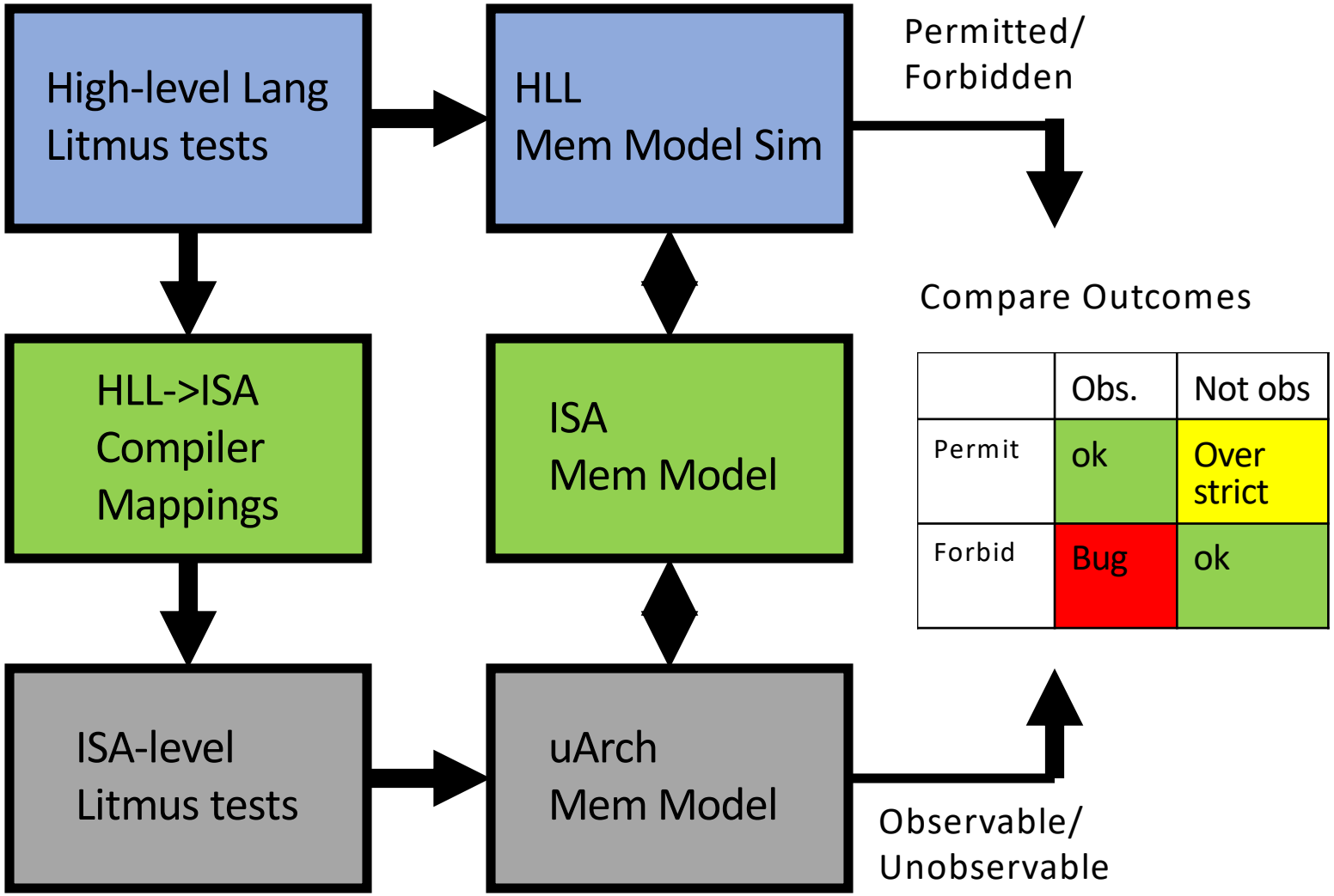
Microarchitecture Specification in *μSpec* DSL

Exhaustive consideration of all possible executions



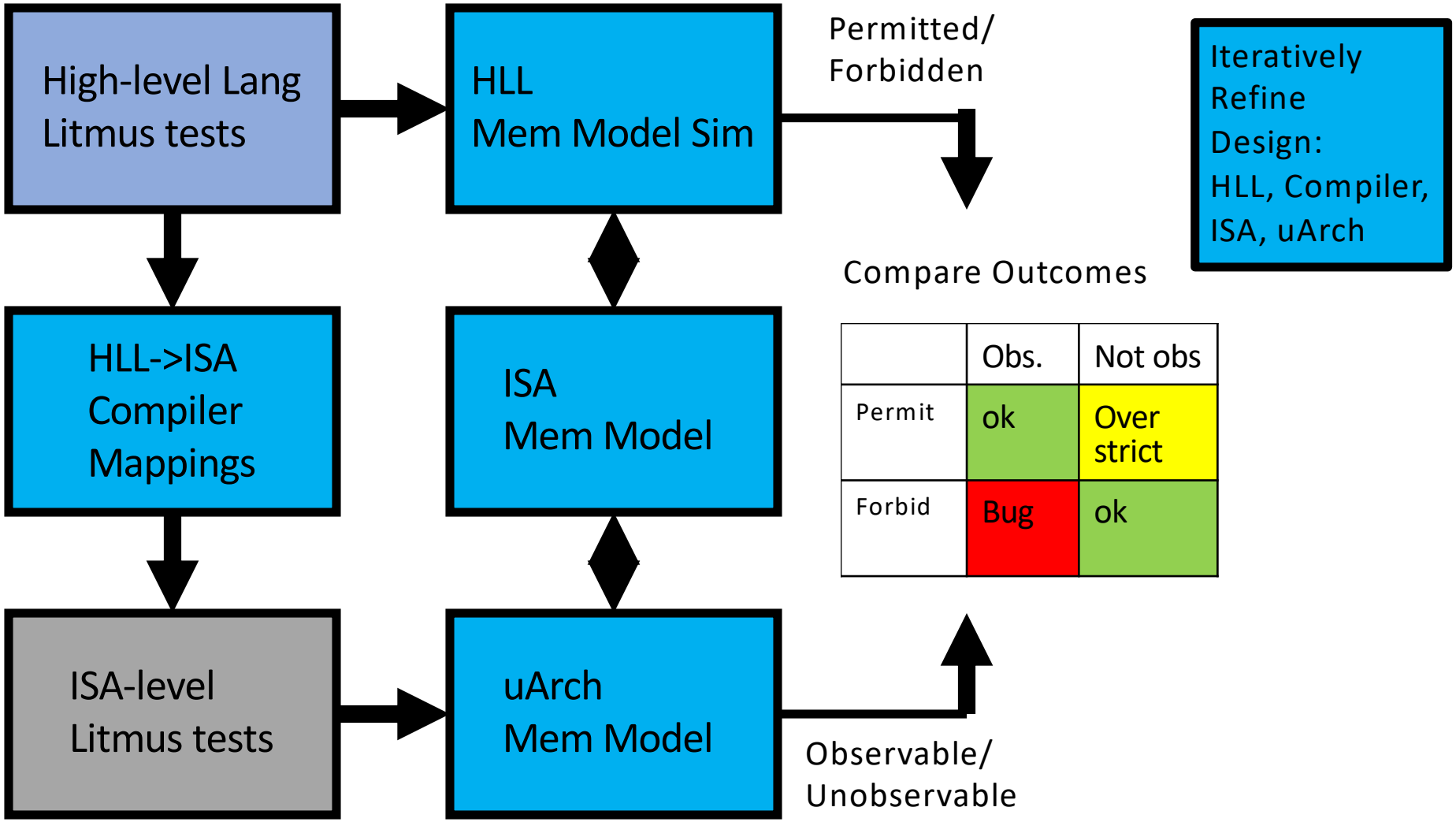
Microarchitectural happens-before (μhb) graphs

HLL <-> ISA <-> uArch: TriCheck Framework



	Obs.	Not obs
Permit	ok	Over strict
Forbid	Bug	ok

HLL <-> ISA <-> uArch: TriCheck Framework



RISC-V Case Study

- Apply TriCheck to 7 legal RISC-V implementations:
  - All abide by current RISC-V spec
  - Vary in preserved program order and store atomicity
- Results:
  - Impossible to compile C11 for RISC-V as specified.
  - Insufficiently strong fence instructions, load-load reordering, …
  - Out of 1,701 tested C11 programs:
  - RISC-V-Base-compliant design allows 144 buggy outcomes
  - RISC-V-Base+A-compliant design allows 221 buggy outcomes

Takeaway: Draft RISC-V spec could not serve as a legal C11 compiler target

Next Steps: RISC-V Memory Model Working Group formed to address these issues. Will soon ratify a new and formally specified RISC-V memory model that supports C11, Linux, etc.

The Check Suite: An Ecosystem of Tools

So far, tools have found bugs in:

- Widely-used Research simulator
- Cache coherence paper
- In-design commercial processors
- RISC-V ISA specification
- Compiler mapping proofs
- IBM XL C++ compiler (fixed in v13.1.5)
- C++ 11 mem model

TriCheck [ASPLOS '17] [IEEE MICRO Top Picks]

COATCheck [ASPLOS '16] [IEEE MICRO Top Picks]

PipeCheck [Micro-47] [IEEE MICRO Top Picks]

CCICheck [Micro-48] [Nominated for Best Paper Award]

RTLCheck [Micro-50] [IEEE MICRO Top Picks Honorable Mention]

Key Takeaways

Need formal, well-specified interfaces

From well-specified interfaces-> Interaction models and analysis

From well-specified interfaces -> Reliability and performance metrics

Examples

- **Power**: Power-efficient Architecture & Models
- **Corrrectness**: Concurrency -> Memory Models
- **Security**: Formal Methods for Secure Design

Example 3: Security

Well-known cache side-channel attack { Flush+Reload

+

Widely-used hardware feature { Speculation

New exploit =

2 new attacks {

January 2018: Spectre & Meltdown

c|net REVIEWS NEWS VIDEO HOW TO SMART HOME CARS DEALS DOWNLOAD

SECURITY / Leer en español

Spectre and Meltdown: Details you need on those big chip flaws

Design flaws in processors from leading chipmakers could let attackers access sensitive information. How did this happen, and what's the fix?

BY LAURA HAUTALA / JANUARY 8, 2018 11:51 AM PST

Project Zero

News and updates from the Project Zero team at Google

Wednesday, January 3, 2018

Reading privileged memory with a side-channel

Posted by Jann Horn, Project Zero

WIRED Triple Meltdown: How So Many Researchers Found a 20-Year-Old Ch

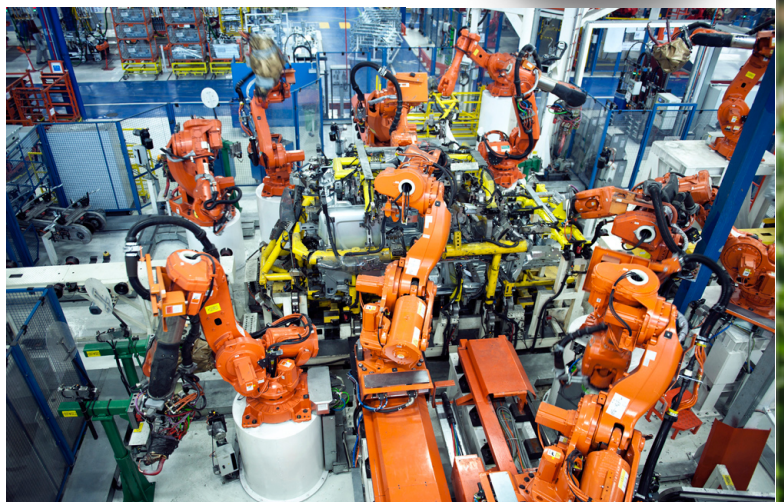
SHARE

TRIPLE MELTDOWN: HOW SO MANY RESEARCHERS FOUND A 20-YEAR-OLD CHIP FLAW AT THE SAME TIME

Security and Reliability in IoT Systems

- IoT/mobile applications are heterogeneous, distributed systems
  - Edge->Cloud
  - Run on diverse hardware
- Applications programmed by huge variety of people…
  - from many companies…
  - and not all well-versed in concurrency issues
- Blackbox Implementations
  - Consumers see very little: How to know what to trust?
  - Even designers of other submodules may not see enough to model, measure, verify.

Q: How to analyze the reliability, security, … of these systems?



[EETimes, MedTechBlog, KOMO News, Amazon, HAPILabs, Princeton, App State]

Security Today

- Too much emphasis on one-off exploit discoveries and fixes
- Too little emphasis on principled discovery mechanisms and corresponding defenses

Attack Discovery & Synthesis: What We Would Like

1. Specify system to study	Formal interface and specification of given system implementation
2. Specify attack pattern	E.g. Subtle event sequences during program's execution
3. Synthesis	Either output synthesized attacks. Or determine that none are possible

Attack Discovery & Synthesis:
Approach

1. Specify system to study
2. Specify attack pattern
3. Synthesis

- **What we did**: Developed a tool to do this, based on the uHB graphs from previous sections.
- **Results**: Automatically synthesized Spectre and Meltdown, as well as two new exploits.
- **Ongoing**: Applying similar techniques to IoT systems

[Trippel, Lustig, Martonosi. https://arxiv.org/abs/1802.03802]
[Trippel, Lustig, Martonosi. MICRO-51. October, 2018]

Example 3: Summary

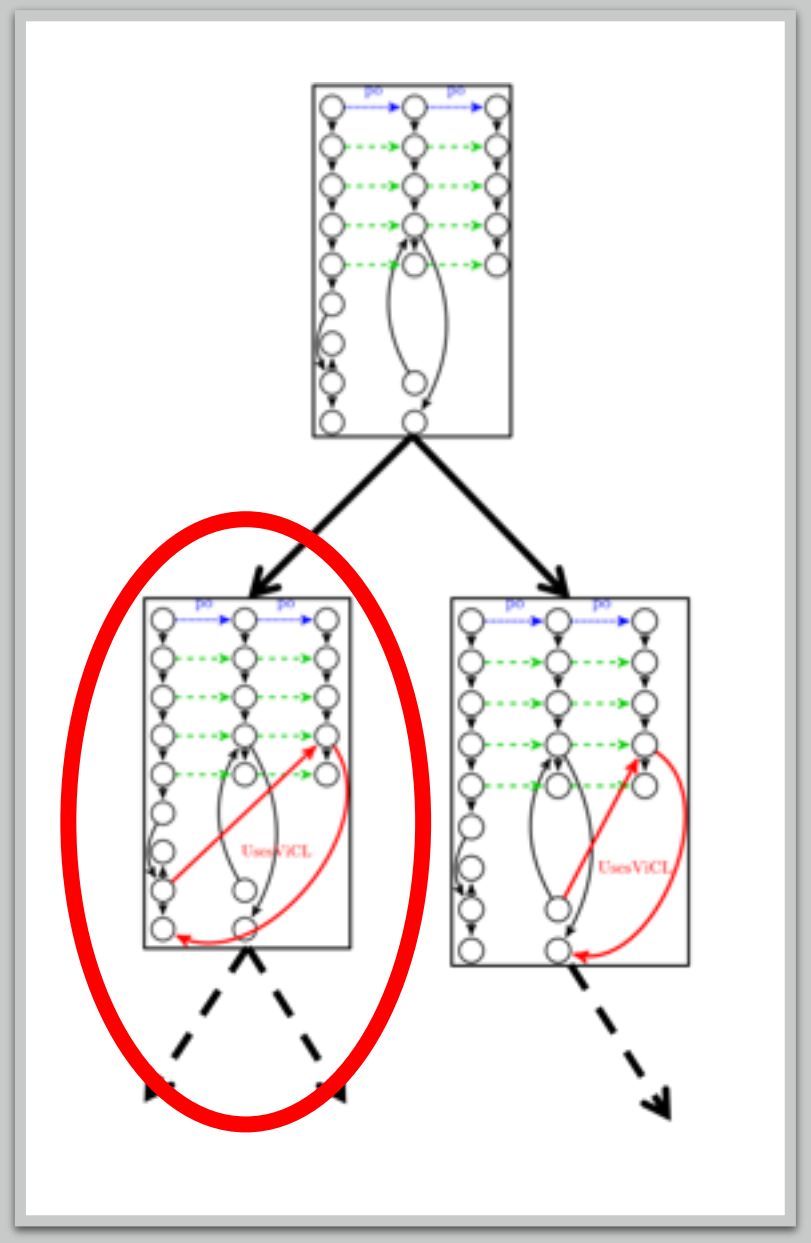
- Formal specification interfaces and analysis can help improve security
  - Move from anecdotal approaches to more comprehensive verification
- Full stack approach to security specification and verification
  - Software specs name hardware security assumptions, to be checked automatically

Next Steps:

- Metrics: How to *quantify* results of a security analysis like ours?
- From one axiomatic interface specification -> Models of Security, Reliability and Performance?

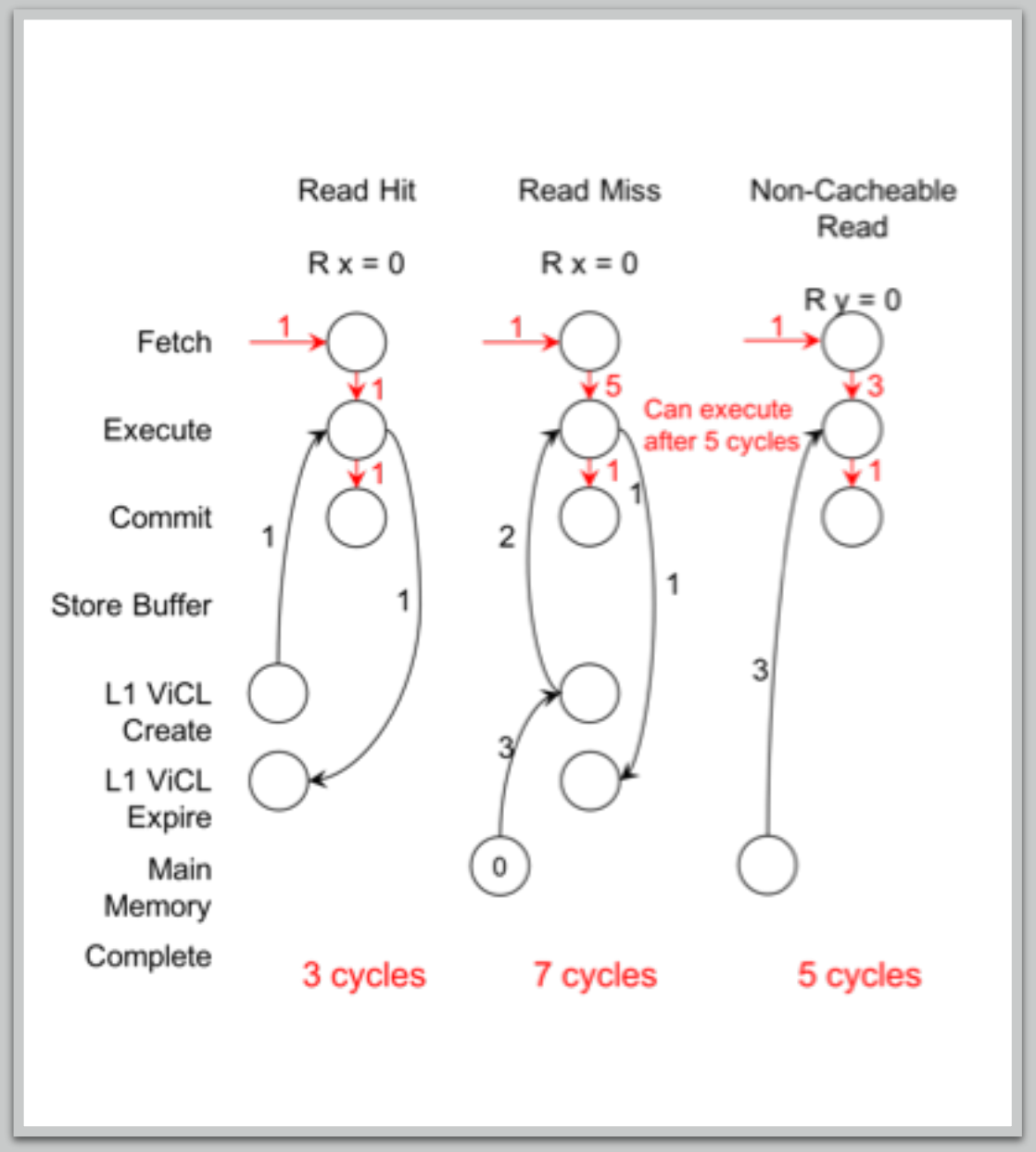
From Interfaces to Models to Metrics

- Axiomatically-specified interfaces can support formal security/reliability verification well.
- **How to extend these to metrics and statistical models?**
- **Across cases**: Statistical likelihood analysis
  - Example: Rate the likelihood of a particular instance being observed, not just verifying whether it is observable or not



From Interfaces to Models to Metrics

- Axiomatically-specified interfaces can support formal security/reliability verification well.
- **How to extend these to metrics and statistical models?**
- **Across cases**: Statistical likelihood analysis
  - Example: Rate the likelihood of a particular instance being observed, not just verifying whether it is observable or not
- **Within one case**: Weighted edges for latency, power, reliability
  - Example: "Observer model" to analyze security side channels based on power or thermal fluctuations instead of timing variations.



Summary

Move past the “Streetlight Effect”

With novel approaches for interface specification, modeling, and metrics.

To meet the performance, reliability, security, fairness, power, … needs of modern computer systems.

Thanks to…

Caroline Trippel, Yatin Manerkar, Tae Jun Ham, Themis Melissaris, Dan Lustig, Kelly Shaw.

Funding:

NSF, DARPA, JUMP Center on Applications Driving Architectures

For more info:

Me: http://www.princeton.edu/~mrm
Group Papers: http://mrmgroup.cs.princeton.edu
Verification Tools: http://check.cs.princeton.edu