# Scalable and Secure Systems

**Zixuan Wang**
**UC San Diego**

## "Integration Cost"

### Let's make a single thing fast!

Storage
Mem / Mem
Network

### Until it stops scaling…



50 Years of Microprocessor Trend Data

Transistors (thousands)
Single-Thread Performance (SpecINT x 10^3)
Frequency (MHz)
Typical Power (Watts)
Number of Logical Cores

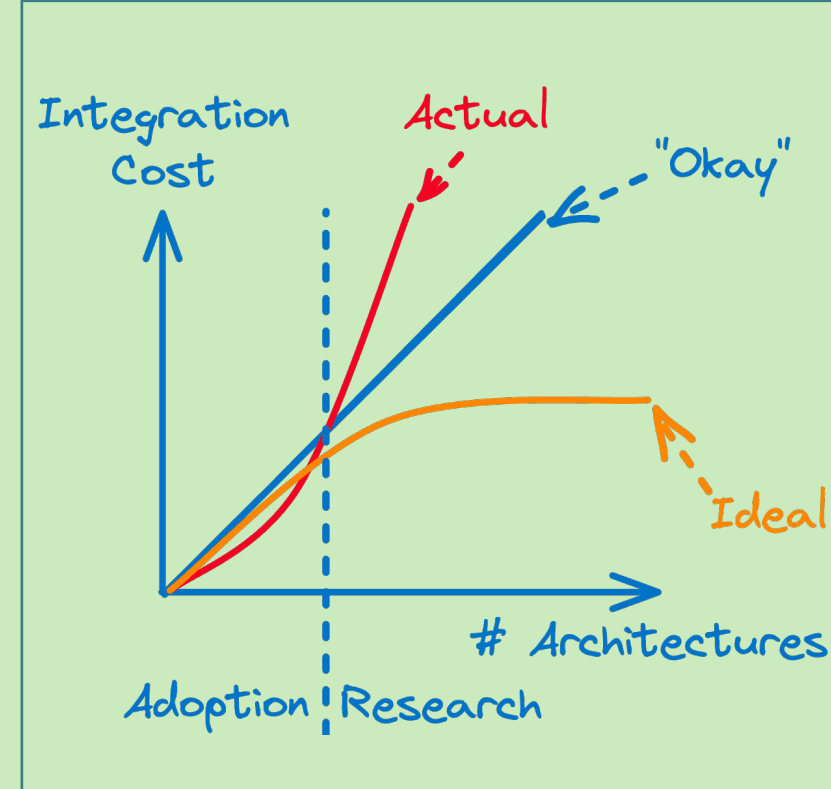Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten. New plot and data collected for 2010-2021 by K. Rupp

*https://github.com/karlrupp/microprocessor-trend-data*

### Let's integrate many fast things!

DPU / Storage
Mem / Mem
Network / GPU
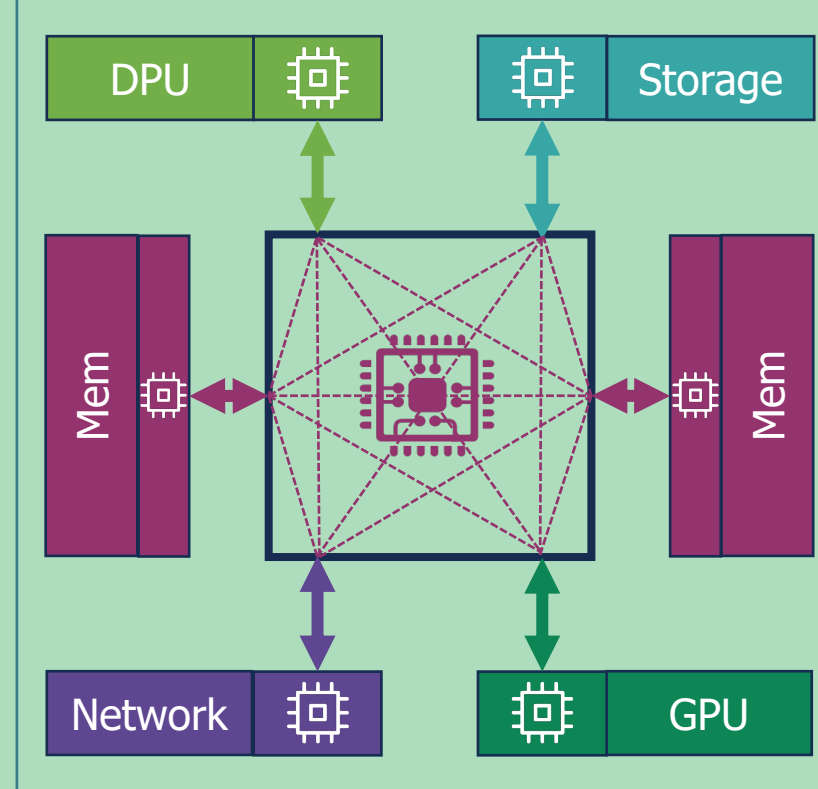
### Until things get exponentially complex…

Integration Cost
Actual
"Okay"
Ideal
# Architectures
Adoption | Research

### Now I'm making it easy to integrate things

DPU / Storage
Mem / Mem
Network / GPU
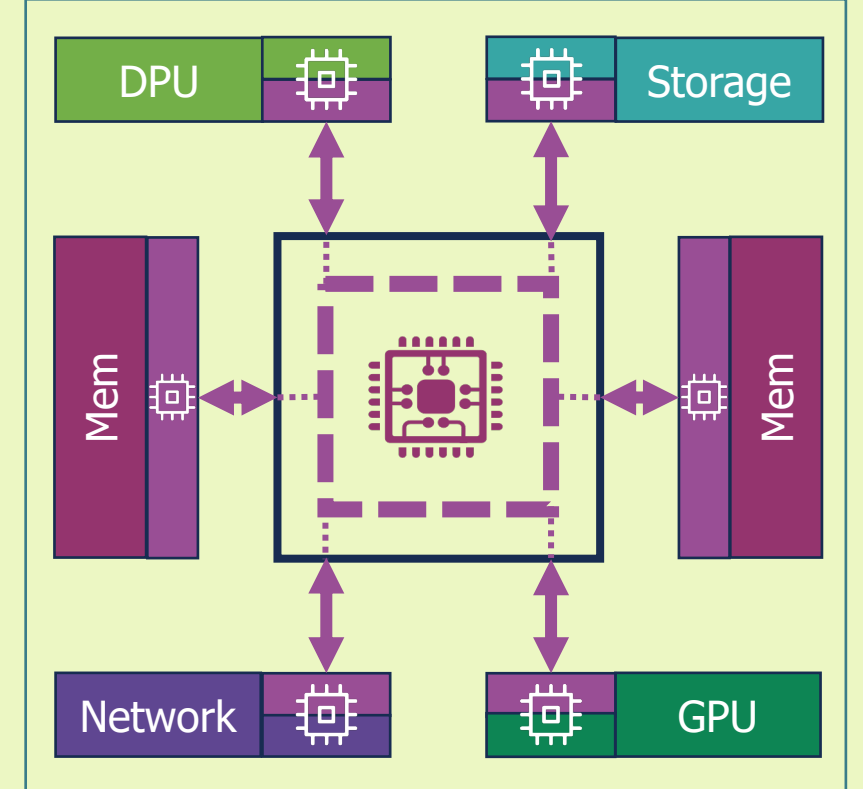
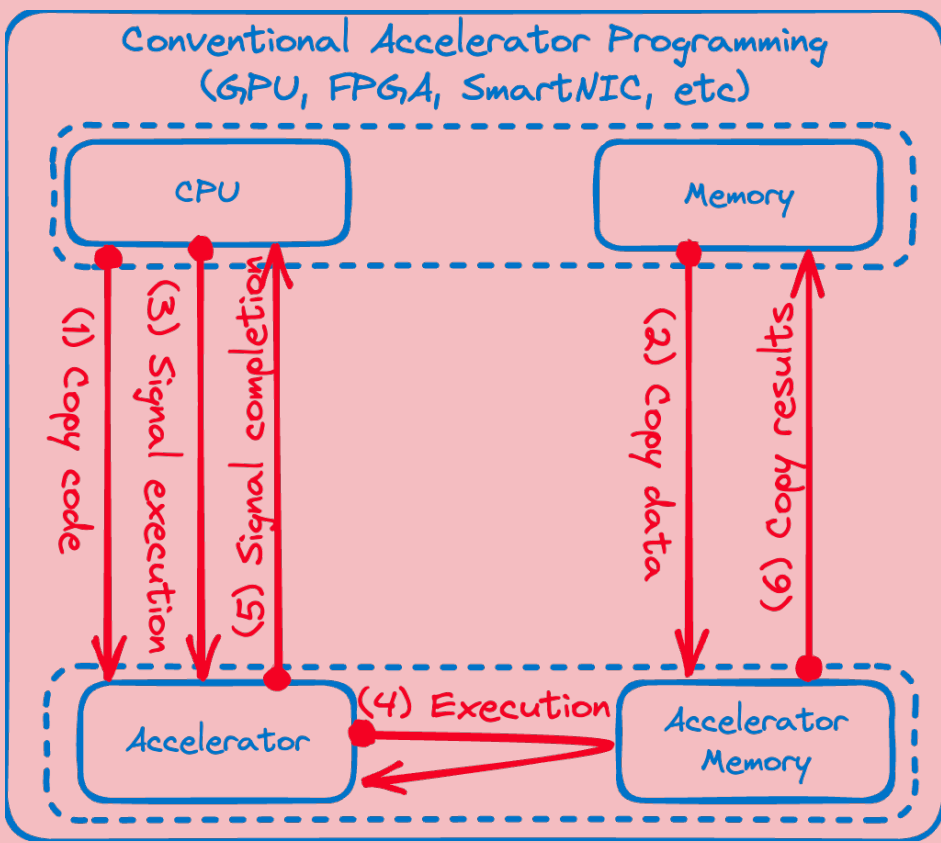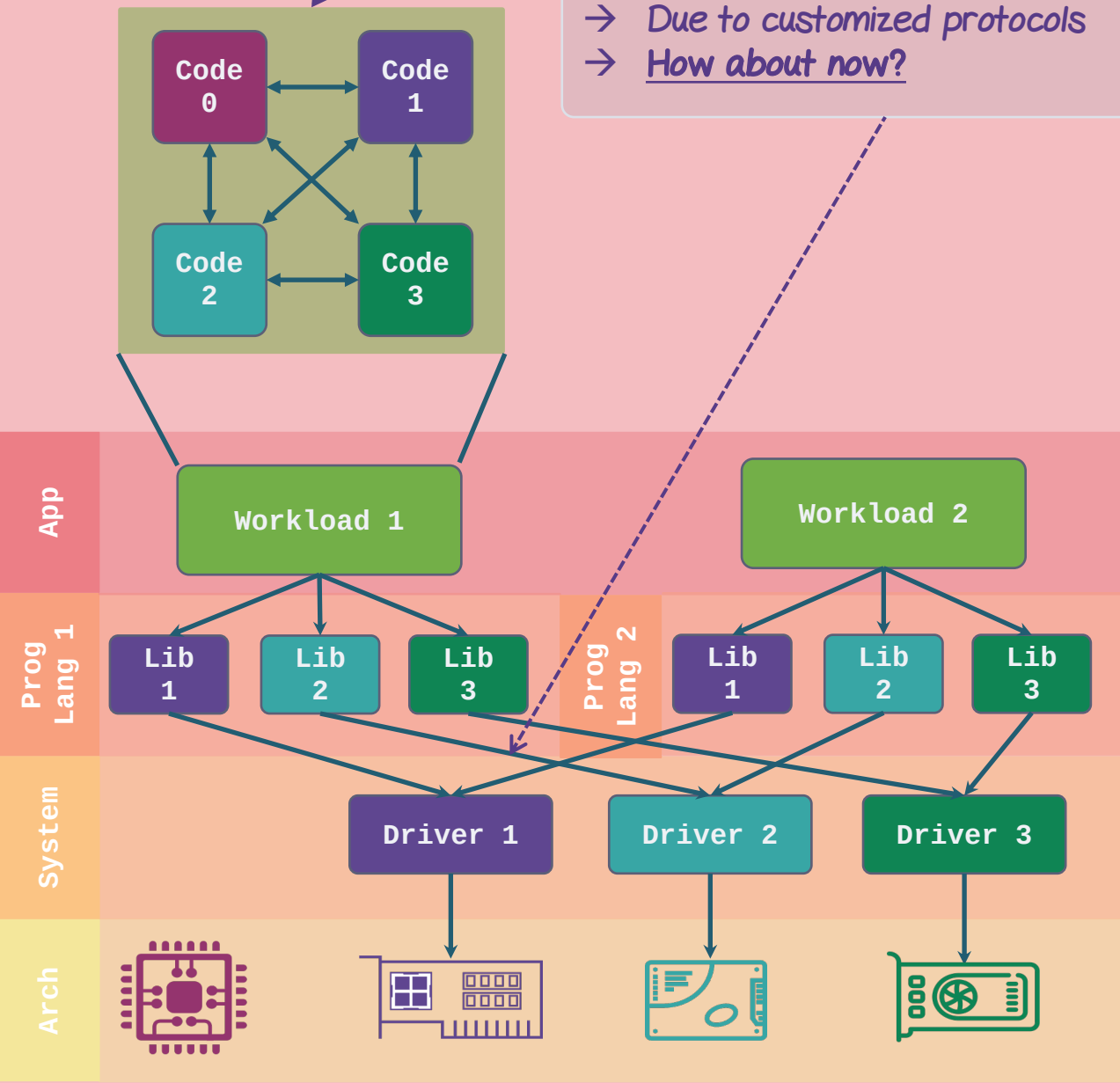## System Scalability and Security
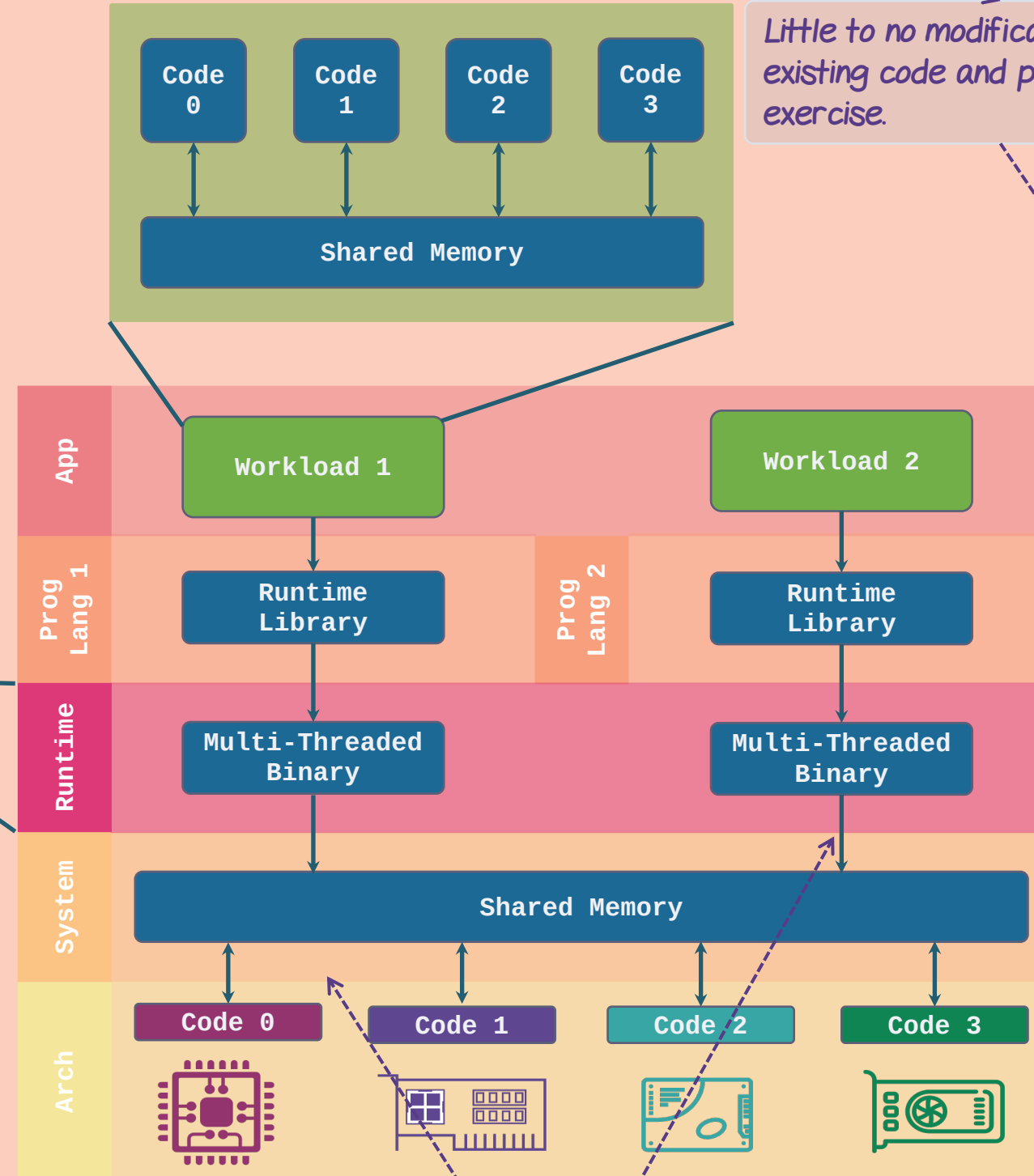
| Past | My Research | Future |
|---|---|---|

### Past

Programming is exponentially complex
→ Limiting the # of arch integrated
→ Bad scalability
→ More vulnerabilities

We must do it this way, in the past…
→ Due to customized protocols
→ How about now?

Code 0 / Code 1 / Code 2 / Code 3

App
Prog Lang 1 / Prog Lang 2
System
Arch

Workload 1 / Workload 2
Lib 1 / Lib 2 / Lib 3 / Lib 1 / Lib 2 / Lib 3
Driver 1 / Driver 2 / Driver 3

Conventional Accelerator Programming
(GPU, FPGA, SmartNIC, etc)

CPU — Memory
(1) Copy code
(3) Signal completion
(5) Signal execution
(2) Copy data
(6) Copy results
(4) Execution
Accelerator — Accelerator Memory

### My Research

Scalability: Make accelerators talk in the same language
→ A unified intermediated layer
→ Code compiled to one representation
→ Accelerators integrate to one layer
→ Efforts: COARSE, WPerf, CodeFlow

Code 0 / Code 1 / Code 2 / Code 3
Shared Memory

Little to no modification to existing code and programming exercise.

App
Prog Lang 1 / Prog Lang 2
Runtime
System
Arch

Workload 1 / Workload 2
Runtime Library / Runtime Library
Multi-Threaded Binary / Multi-Threaded Binary
Shared Memory
Code 0 / Code 1 / Code 2 / Code 3

Compile once, run everywhere

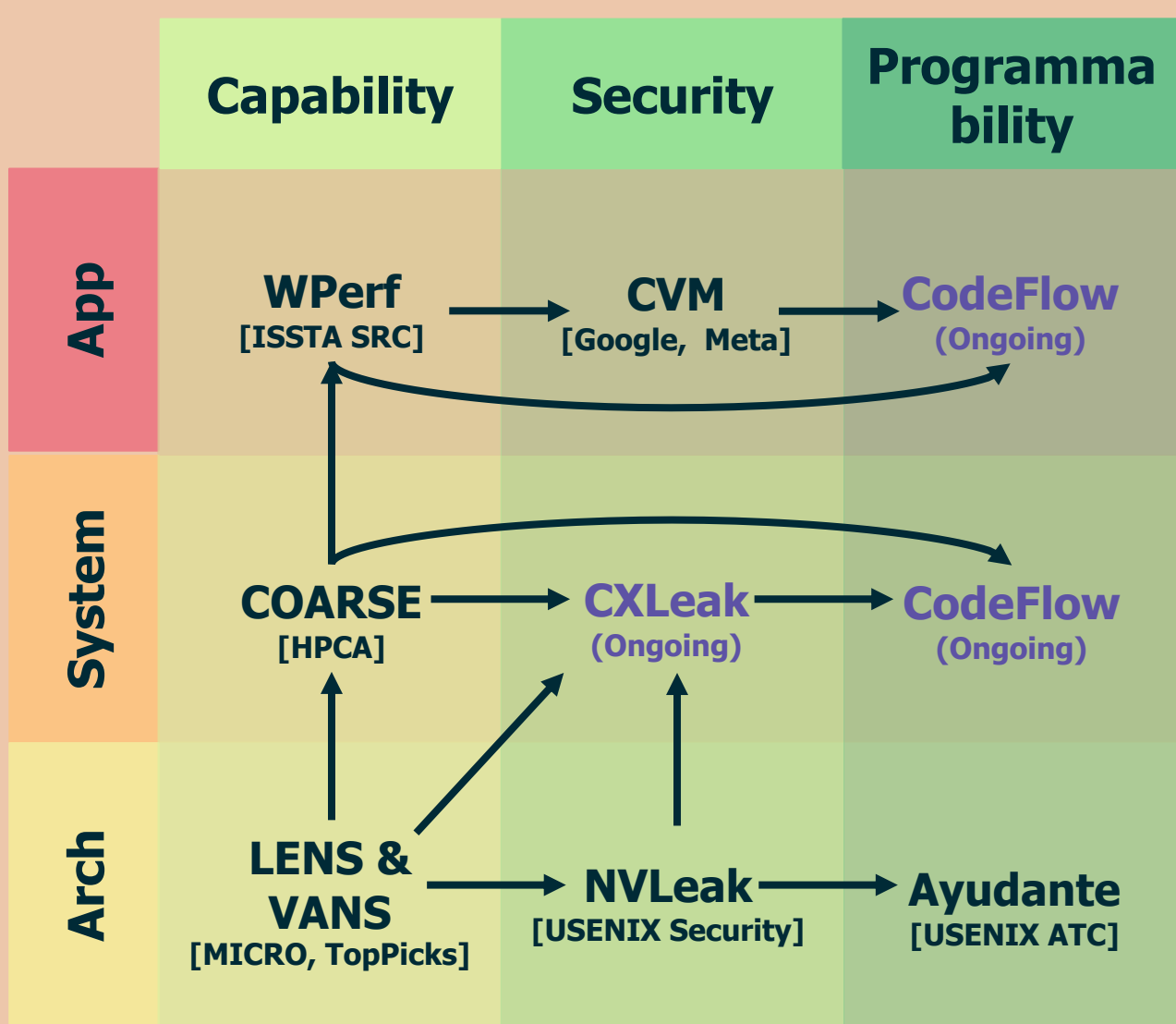Low overhead, good abstraction

Accelerators exchange code and data in a unified approach

Security: A systematic analysis framework
→ Unbox the arch design black-box
→ Attack and mitigate
→ Efforts: LENS, NVLeak, CXLeak

### Future

**Application**
LLM Training & Inference
Vector Database
KV-Store
Traditional workloads

**Programming**
Compiler & library support
More language supports
Compiler optimizations

**Runtime**
Just-in-time compilation
Abstraction of common operations
Abstraction of system resources
Extension for new accelerators
Runtime security

**System**
Code scheduling
Shared memory support
Resource isolation
Trusted execution

**Architecture**
Unified integration to the runtime
Arch-level optimization
Mitigating arch attacks
Trusted execution

**Hardware**
Modeling new hardware perf.
Modeling integration cost
H/W interface for the runtime

## Research & Industry

| | Capability | Security | Programmability |
|---|---|---|---|
| **App** | WPerf [ISSTA SRC] | CVM [Google, Meta] | CodeFlow (Ongoing) |
| **System** | COARSE [HPCA] | CXLeak (Ongoing) | CodeFlow (Ongoing) |
| **Arch** | LENS & VANS [MICRO, TopPicks] | NVLeak [USENIX Security] | Ayudante [USENIX ATC] |

## Zixuan Wang

**www.TheNetAdmin.net**
**PhD Candidate**
**UC San Diego**
**Working with Jishen Zhao**

Research:
- Arch security
- System for ML
- System for emerging arch

Industry – CVM:
- Google'21:  AMD SEV & SEV-ES
- Meta'22:    AMD SEV-SNP
- Google'23:  AMD SEV-SNP SVSM
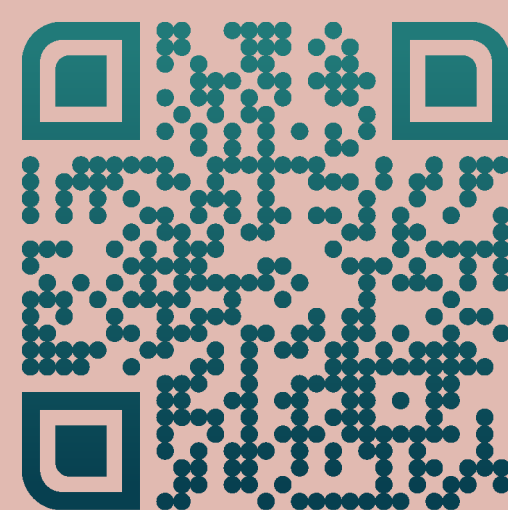
**Scan to access my website**

## Fun Facts