Practical Provable End-to-End Guarantees at the Edge

Amit Vasudevan
SEI/Carnegie Mellon University

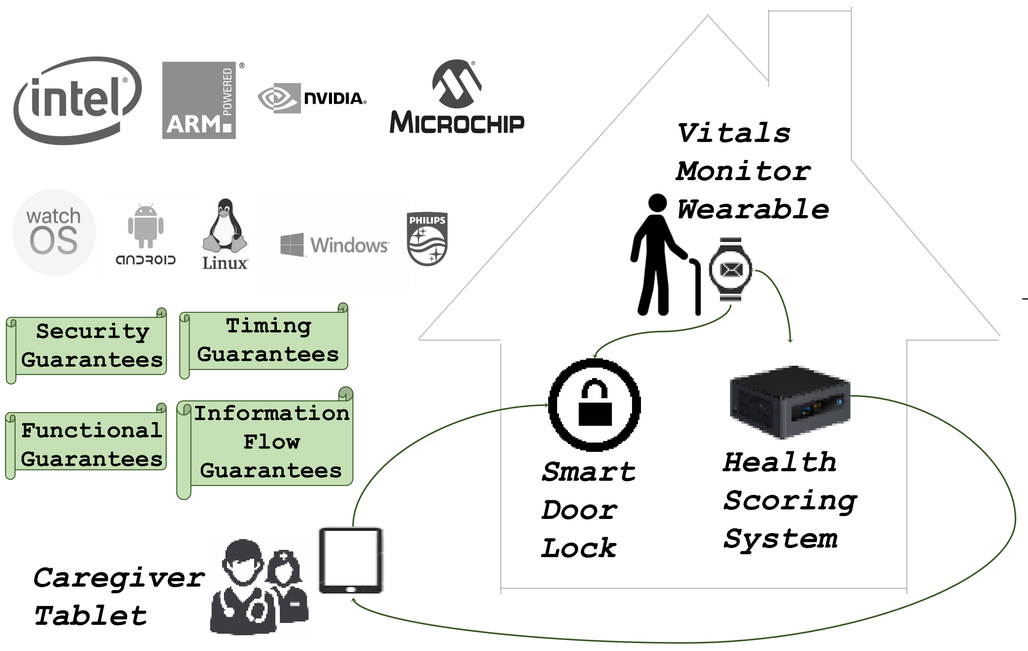
Petros Maniatis
Google Research

Ruben Martins
CSD/Carnegie Mellon University

Sagar Chaki
Mentor/Seimens

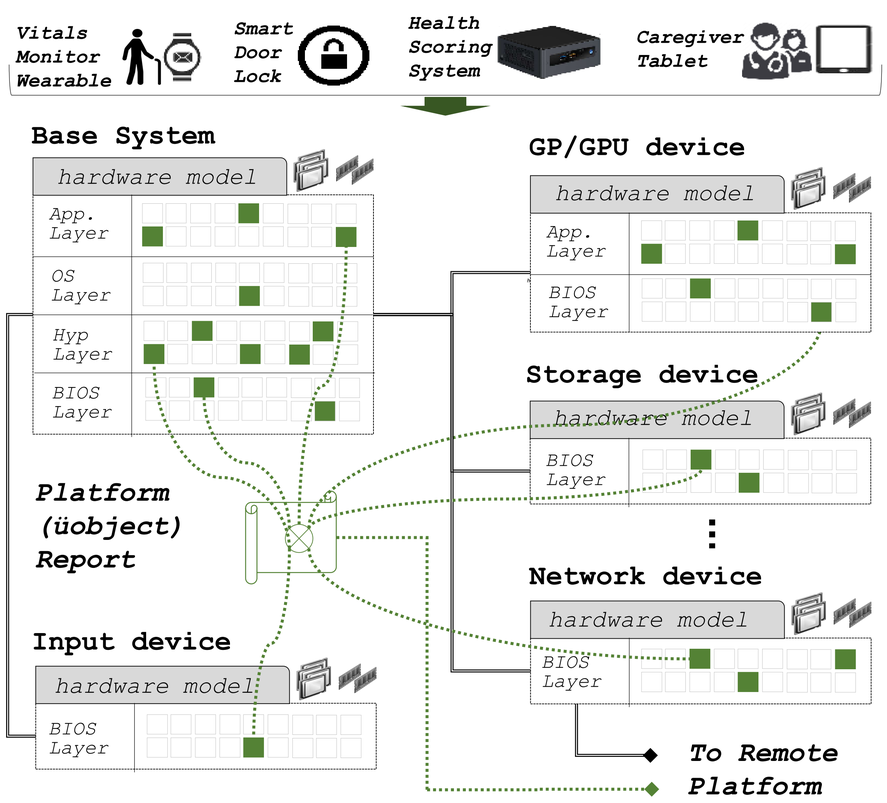
https://uberspark.org

ElderSafe, an Exemplar Edge Service



- *ElderSafe,* an exemplar edge service for elders who live in an assisted-care facility, comprises:
  - Heterogeneous hardware
  - Heterogeneous software
  - Heterogeneous properties
- **Challenge:** How do we achieve practical and provable end-to-end guarantees in this heterogeneous ecosystem?

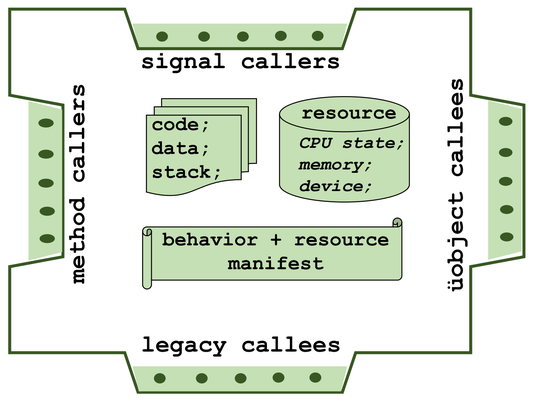
überSpark - Universal Object Abstractions



- **Implementation level** verifiable and trusted computing building blocks (*überobjects* and *überobject collections*)
- Applicable to all software layers and realizable on disparate hardware platforms
- Retrofit incrementally with legacy code at a fine-granularity
- Preserve existing functionality
- Compositional reasoning
- Development compatible
- Support multiple verification techniques

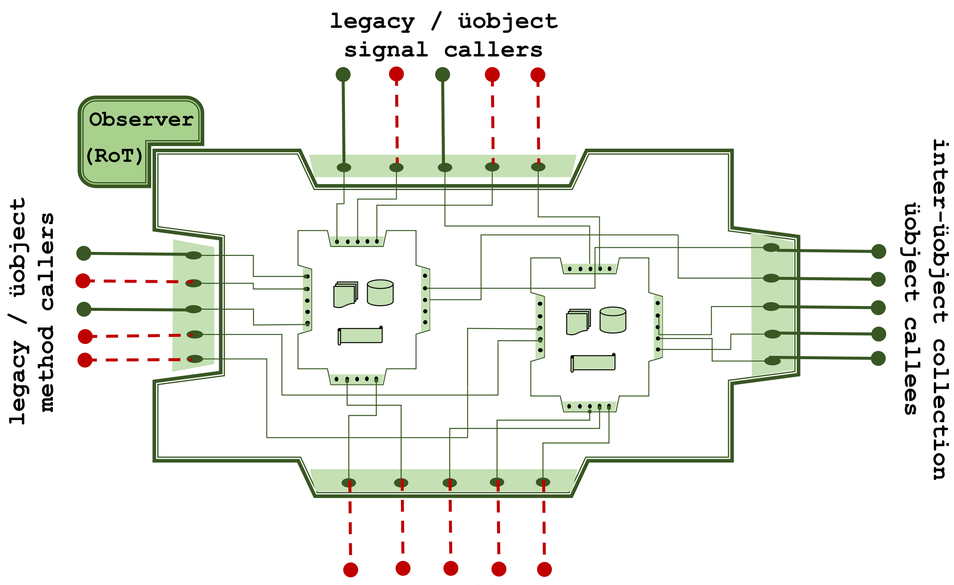
überSpark - Architectural Components

Design-time Abstraction: **überobject**



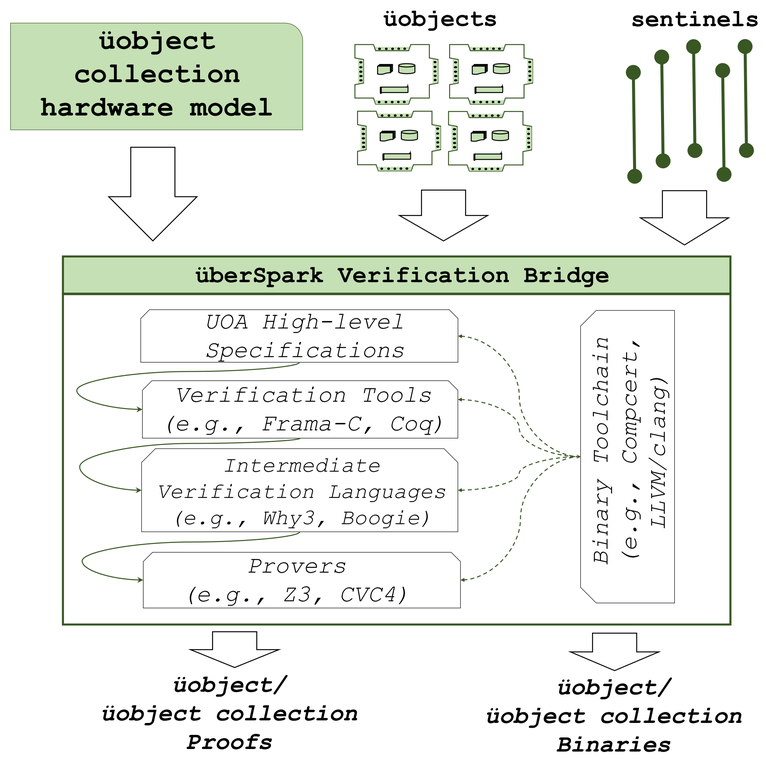
- Exclusive guard for indivisible system resource
- Principled entry, interruption, legacy code, and überobject invocations:
  - Execution traces respecting program control-flow enables use of state-of-the-art program verification tools
  - Facilitates AG reasoning and composition
- *call-return* interfacing retrofits with legacy code and common programming idioms at fine-granularity, while remaining development compatible
- Resource interface confinement provides resource protection and access control while supporting shared memory concurrency; enables multi-threaded implementation and reasoning

Run-time Abstraction: **überobject collection**



- Set of überobjects that share a memory address space
- **Observers:** Boot-strapping and root-of-trust entities
- **Sentinels:** Enforce call routings, caller-callee mediation, logical privilege separation, and provide flexible implementations

überSpark - Verification Bridge



- überobject independent base invariants
- überobject specific invariants and properties
- Verify überobject and generate binary while allowing the use of multiple verification tools and techniques
- Hardware model to bridge heterogenous hardware and verify and compose properties over hardware states

Research Questions

- Can we automate the creation of überobjects and automatically infer invariants?
- What are the classes of properties that can be formally verified and bridged with **überSpark**?
- What programming languages/model can **überSpark** accommodate?
- Can **überSpark** guarantee verified properties with practical performance?
- Can **überSpark** be applied to hardware itself (e.g., CPU micro-code, FPGA)?
- Can **überSpark** be applied hierarchically? (e.g., JVM written in C überobjects running Java überobjects as bytecode)

Open Challenges

- Expressivity of the verification bridge, hardware model, specification language, and protocols while preserving soundness
- Adoptability by platform owners and developers
- Physical platform tampering
- Spurious platform failures