RUN-TIME VALIDATION OF INCREMENTAL CODE CHANGES

Gennady Pekhimenko, Todd Mowry, Onur Mutlu (Carnegie Mellon), Phillip Gibbons, Michael Kozuch (Intel)

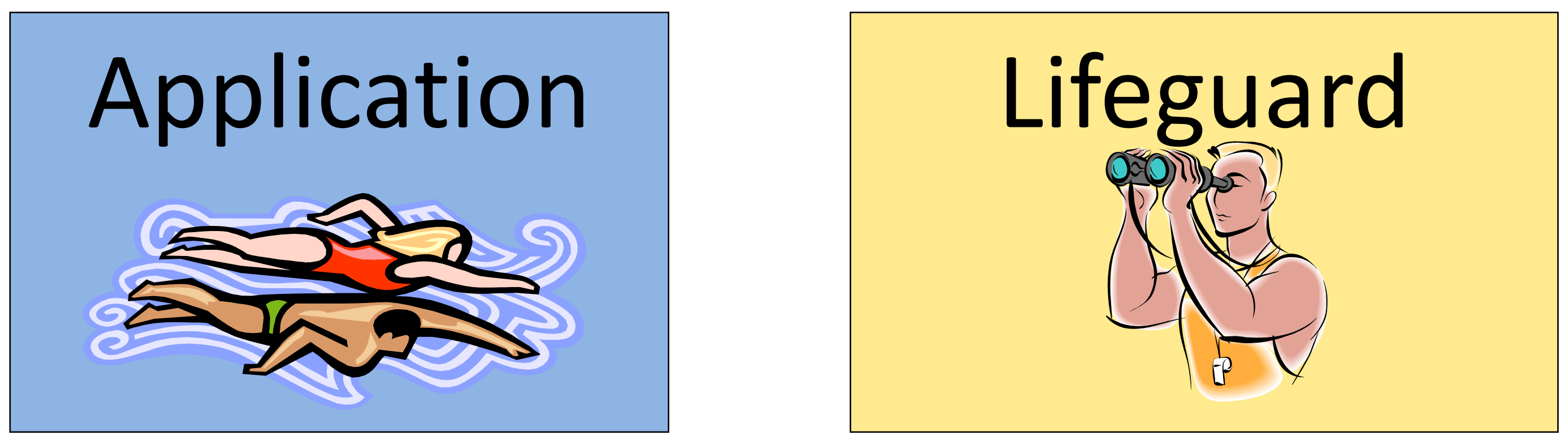
MOTIVATION AND GOALS

- Software security/correctness verification
  - Static verification is desirable, but complicated
    - requires formal specification
    - makes conservative decisions that lead to false positives

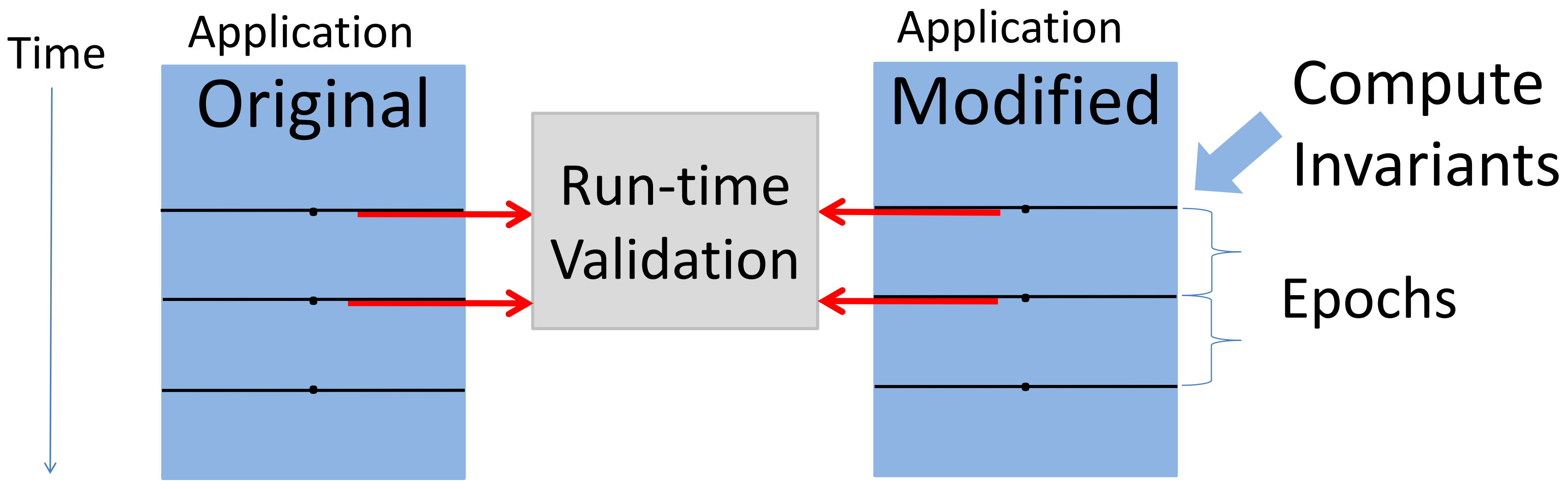
- Is it possible to make validation at *run-time*?
  - General case is complicated, can we make it more tractable?
  - *Yes*, if
    - we exploit cloud software specific characteristics
    - avoid complex full comparison

KEY INSIGHTS AND OBSERVATIONS

- Incremental code changes
  - Cloud software develops through “small” code changes
  - Similar to
    - production software patches
    - software/compiler optimizations
- Sophisticated fine-grain software analysis is possible
  - *Log-Based Architectures Projects* (LBA)



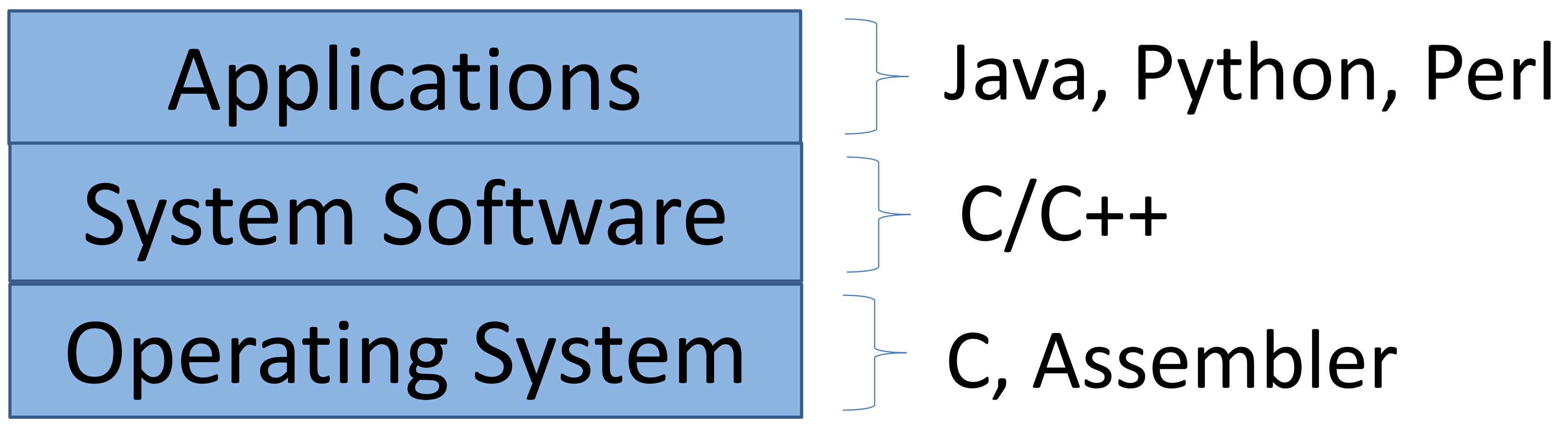
RESEARCH CHALLENGES



- Invariants description and detection
  - How to define acceptable differences in application?
  - How to detect useful invariants for future checking?
- Run-time validation tool
  - Software-only approach is inefficient
  - *Hardware-assisted* LBA threads will help
  - Domain *specific* optimizations + *static code analysis*
    - i.e. merge based on confidence status

RESEARCH CHALLENGES

- Software stack:



- Different levels have different granularity of validation checks
  - i.e. output comparisons for scripting languages

- Multiple options in how to define invariants and confidence

INVARIANTS DETECTION

- *Permanent* or static
  - E.g. :
    - pointer *p* is not NULL
    - value *v* is in the range (0,1000)
    - value *v* equals to 2*x + 3
    - for all treenode objects n, n.left.value < n.right.value
  - It is possible to collect such invariants automatically
    - E.g. Daikon invariant detector
- *Transient* or dynamic
  - Invariants at the point of comparison
    - loop iteration count
    - boolean flags
- *Global*
  - Control flow graph (CFG)

SOFTWARE TRANSFORMATIONS

- Software optimizations -- good starting point
  - Effects are more predictable
  - Simple invariants can be sufficient

- Speculative transformations
  - can potentially break sequential semantics
    - local memory pooling
    - auto-parallelization
    - semantic optimizations
    - data types with different precision

- Software patches

Intel Science & Technology Center for Cloud Computing

Carnegie Mellon University Georgia Tech intel PRINCETON UNIVERSITY UC Berkeley