**Proposed OOPSLA 2012 Paper**

The main contributions of this paper are:

* it examines the shortcomings of analyzing program executions via logged raw machine state such as heap dumps
  + **Unfamiliar:** The abstractions provided by application code and the programming language itself are not available, forcing the programmer to understand and work with low-level machine concepts
  + **Complex**: Analysis code must be written at the meta-programming level, which is bulky and time-consuming to write
  + **Brittle:** Analysis easy breaks, or worse produces incorrect results, if applied to a heap dump produced by an execution of newer code
  + **Type-unsafe:** a type mismatch is not detected until internal state mismatches, if at all
  + **Insecure:** Analysis code and tools can easily inspect private data, even if accidentally, since state encapsulation is lost
* it presents semantic requirements for *object holograms*, virtual objects based on object mirrors that exhibit a well-defined subset of behaviours consistent with their corresponding objects in remote or past executions
  + member and static field access is consistent with the original memory model, provided that the mirror implementations obey consistency requirements
  + a subset of special forms whose behaviour does not depend on the original context are also available
    - In Java, these include operators like “==” and “instanceof” and native methods like System.arraycopy and Thread.currentThread
* it demonstrates that the use of object holograms improves the accuracy, readability, simplicity and maintainability of remote or post-hoc execution debugging and analysis
  + **Familiar:** Analysis code can use the same APIs and data structure as the original program code
  + **Simple:** Analysis code becomes an order of magnitude smaller
  + **Robust:** The same analysis code works correctly against multiple versions of the source code iff the relevant APIs do not change
  + **Type-safe**: Analysis is compiled against the original data structure definitions and hence mismatches are caught at compile time
  + **Secure:** Code-level access ensures proper encapsulation; the same mechanisms for overriding access control through reflection are available, but require explicit use and are hence much easier to control
* it demonstrates that mirror-based virtual objects, generally studied in the context of dynamic languages, can be efficiently implemented in the statically-typed, pre-compiled Java language on commodity JVMs
  + An implementation based on bytecode rewriting is presented
  + Pre-built binaries without source code, such as the JRE, are fully supported
  + The overhead of object holograms is comparable to enabling debugging tools
  + The use of a mirror-based architecture is shown to offer benefits for bytecode rewriting frameworks in terms of rigor and simplicity
* and it reports on our experience applying object holograms to heap dump analysis and omniscient debugging
  + A collection of Eclipse MAT plugins for inspecting common object state is compared with their equivalent implementations using object holograms w.r.t. code complexity and execution speed
  + ???

Evidence/proofs:

* Illustrate points on problems with heap dump analysis using Eclipse MAT code as running examples
* Describe Java implementation:
  + Focus especially on transforming class hierarchy, dealing with array subtyping and special cases such as Throwable and String
    - Prove (\*\*\* how formally? \*\*\*) transformed bytecode is valid for any compliant JVM
  + Reasonable performance by limiting bytecode rewriting to a separate, sandboxed class loader
* Implement Eclipse MAT plugins and for each compare:
  + Code size: holograms implementation far smaller and simpler, generally a few lines of code compared to 100 or more
    - Include running example before and after side by side as figure
  + Execution time: holograms will be slower, but not by much