



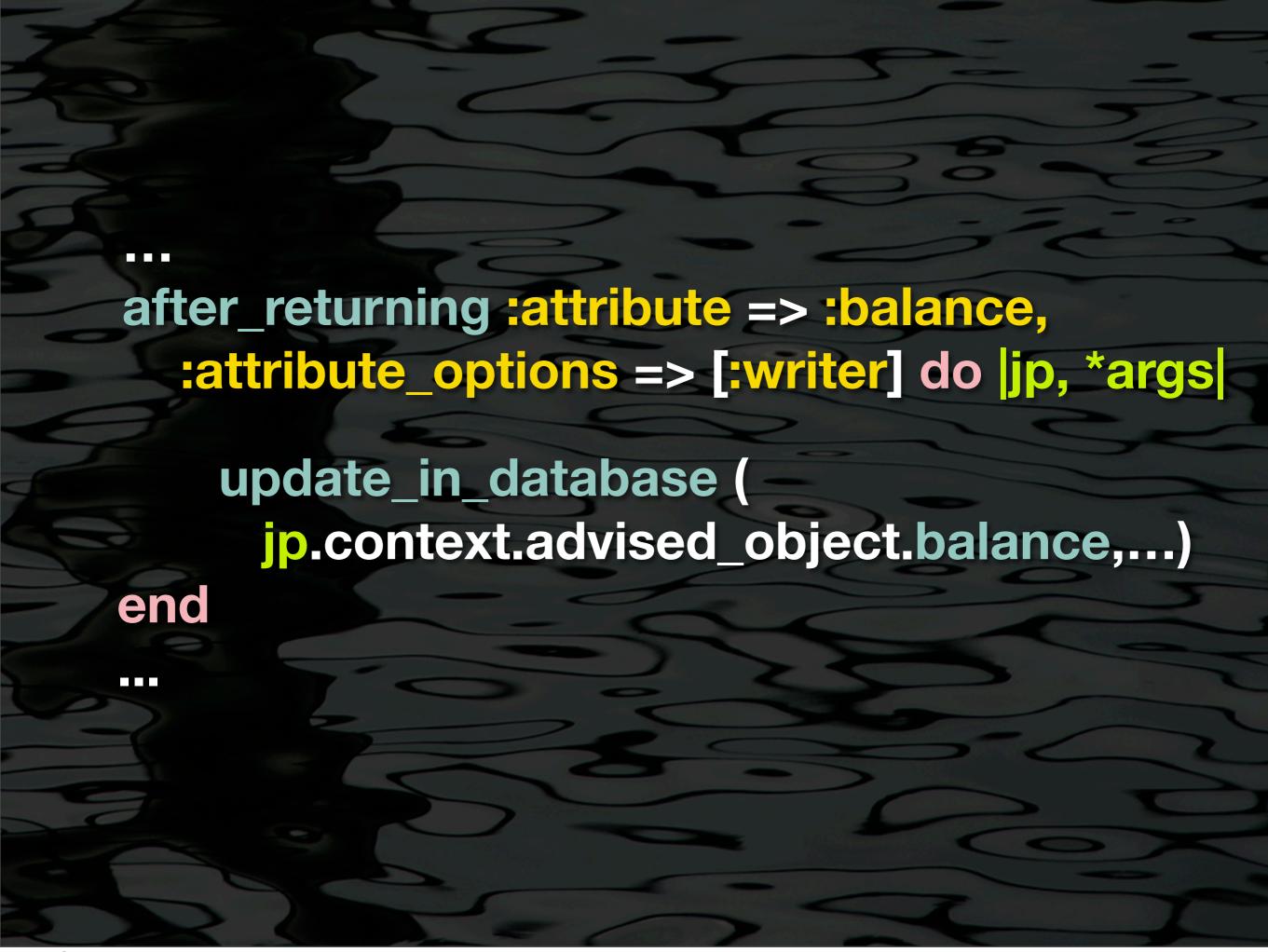


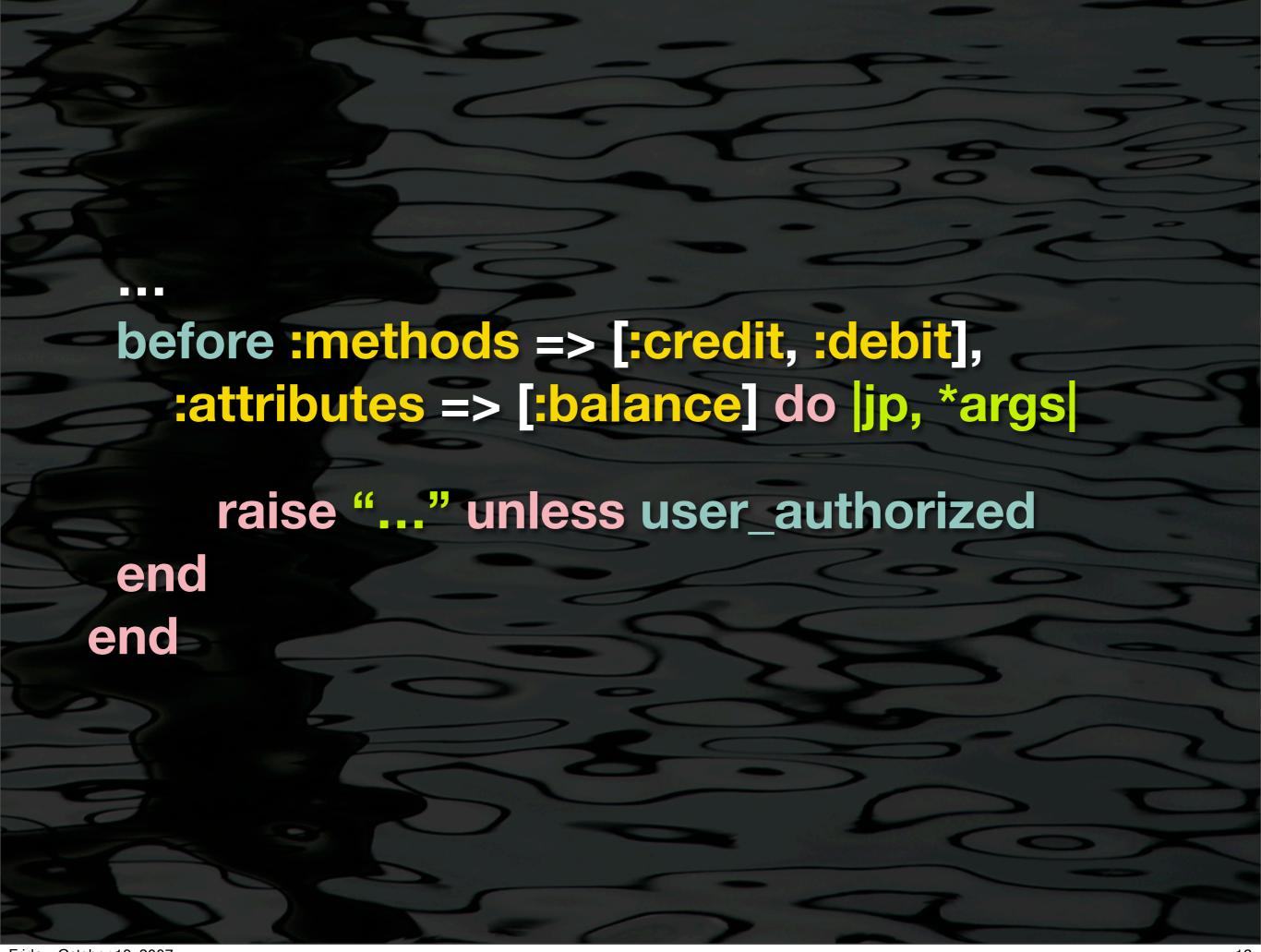
Before returning the Account balance, read the current balance from the persistence store.

After the Account balance changes, update the new balance in the persistence store.

Before changing the Account balance, authenticate and authorize the user.

```
require 'aquarium'
class Account
                      # reopen Account
 include Aquarium::Aspects::DSL::AspectDSL
 before :attribute => :balance,
  :attribute_options => [:reader] do |jp, *args|
  p.context.advised_object.balance =
   read_from_database(...)
 end
                        aquarium.rubyforge.org
                        jp: Join Point
```

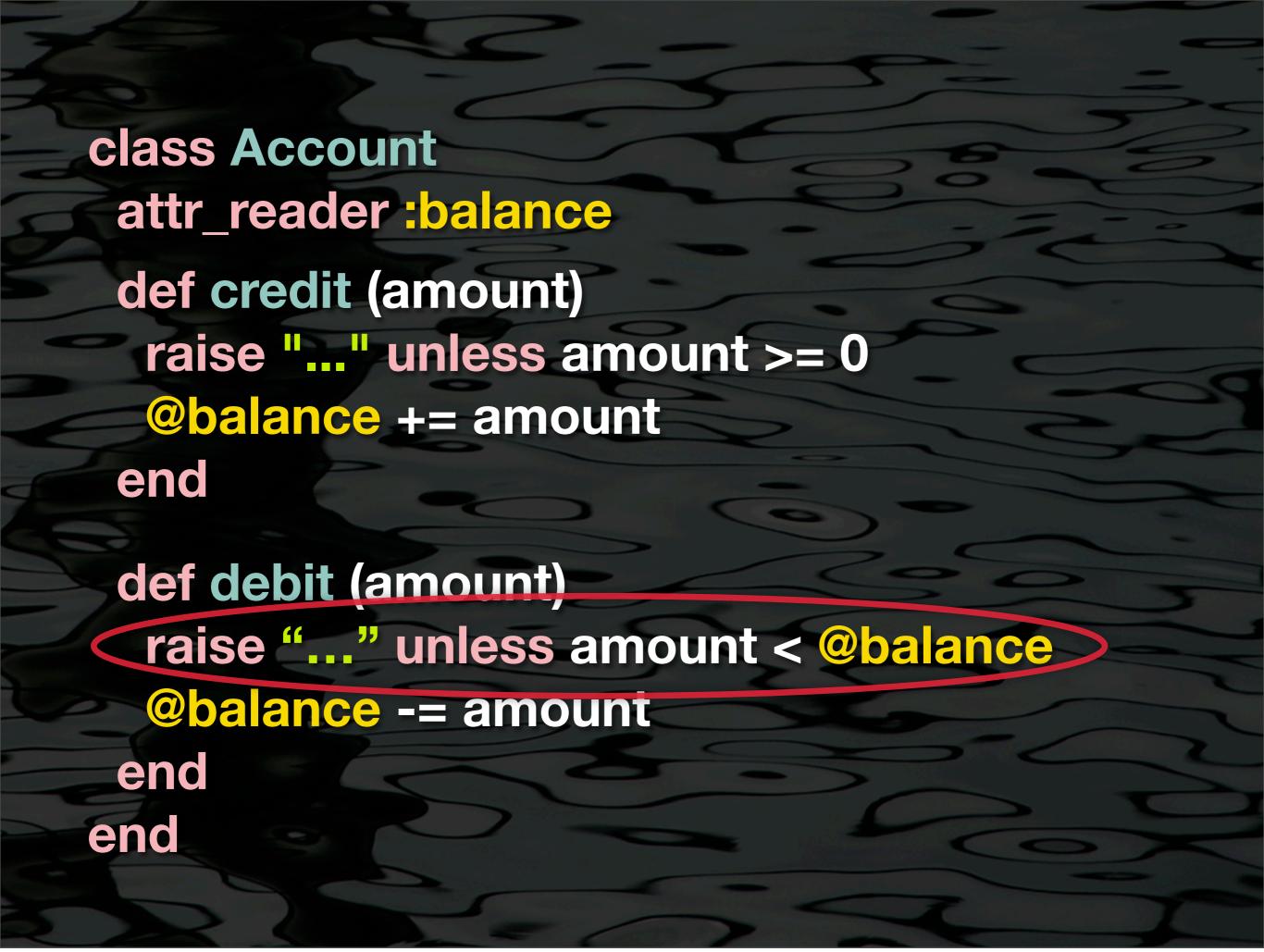


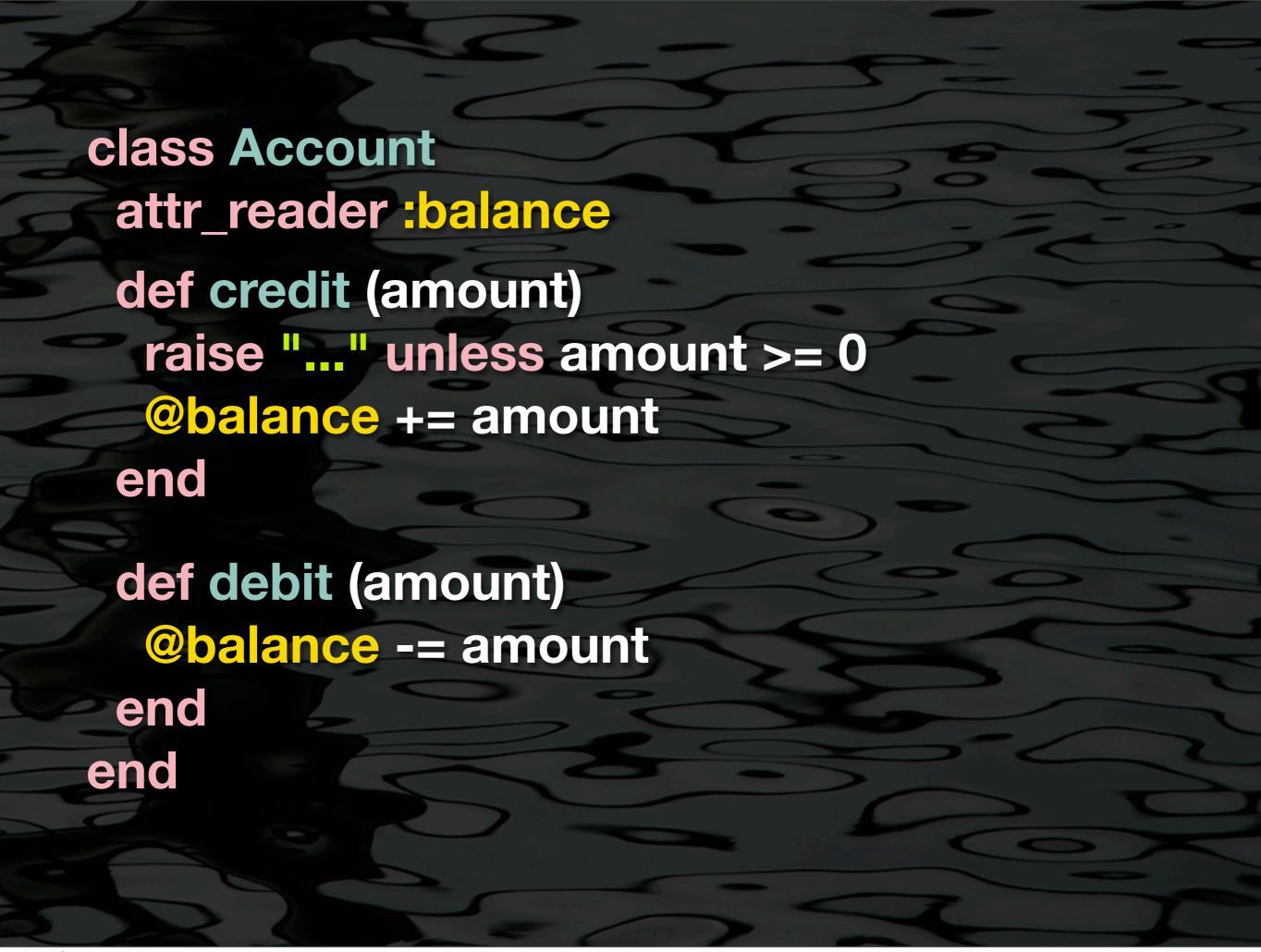


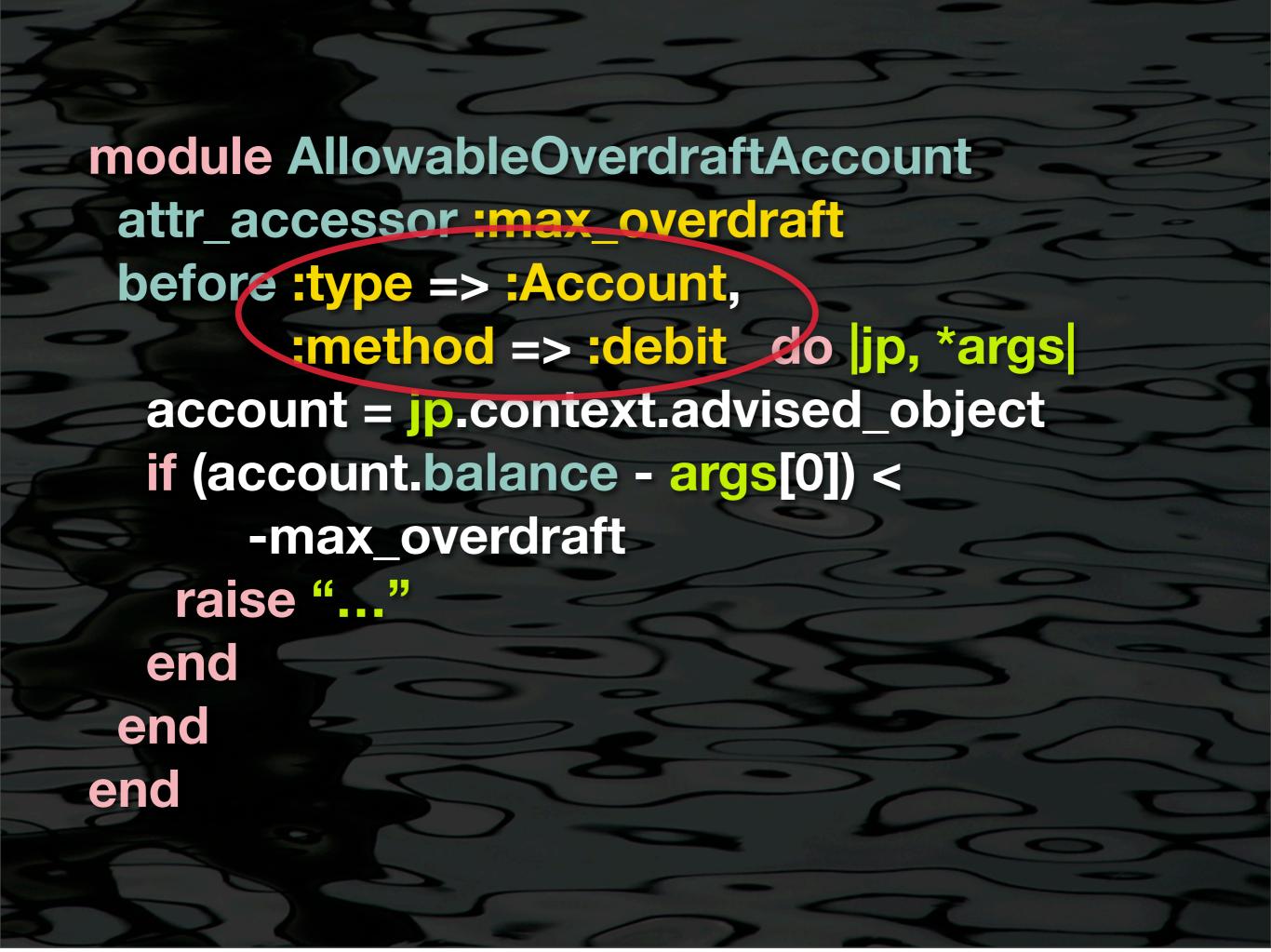


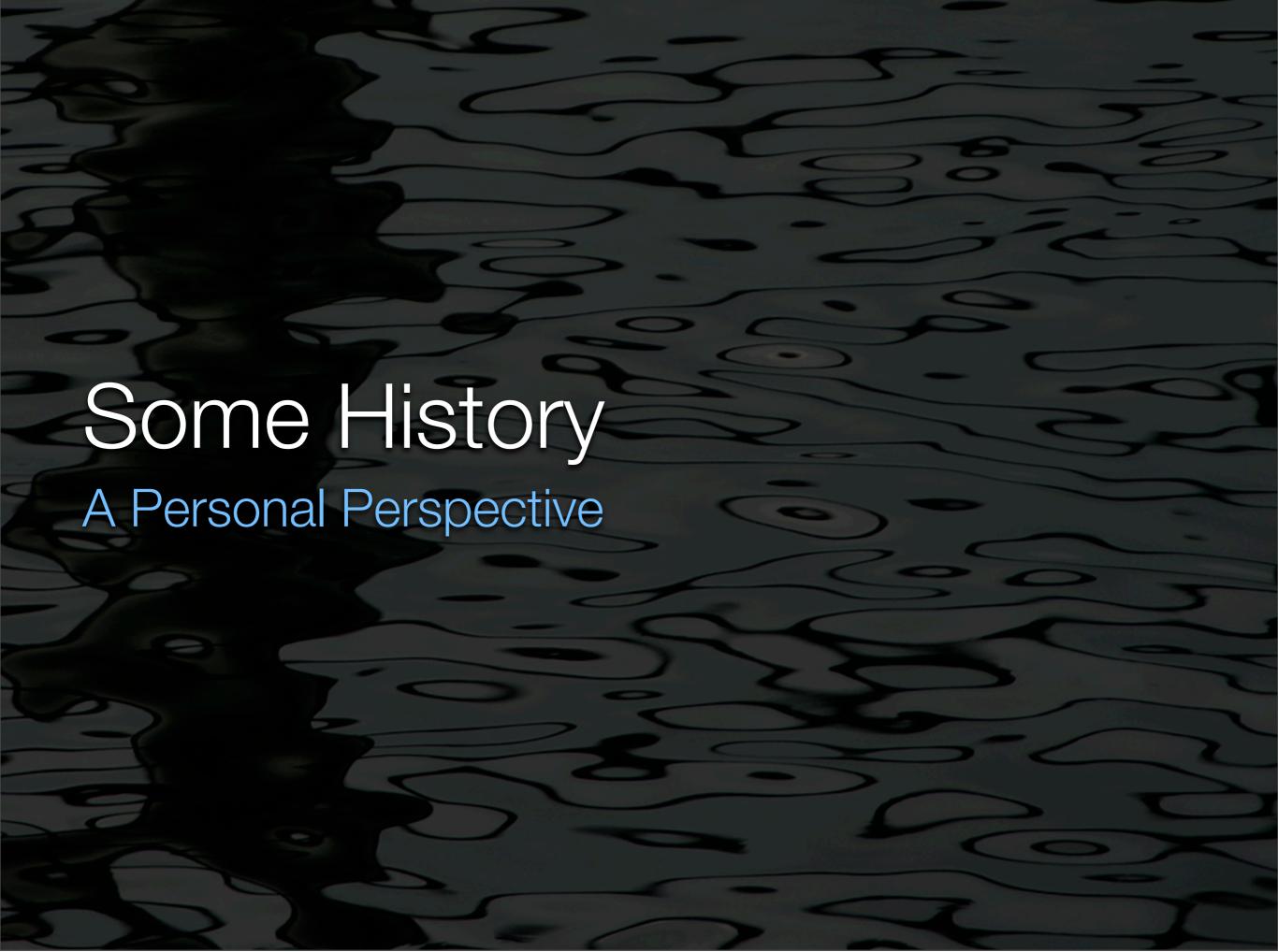




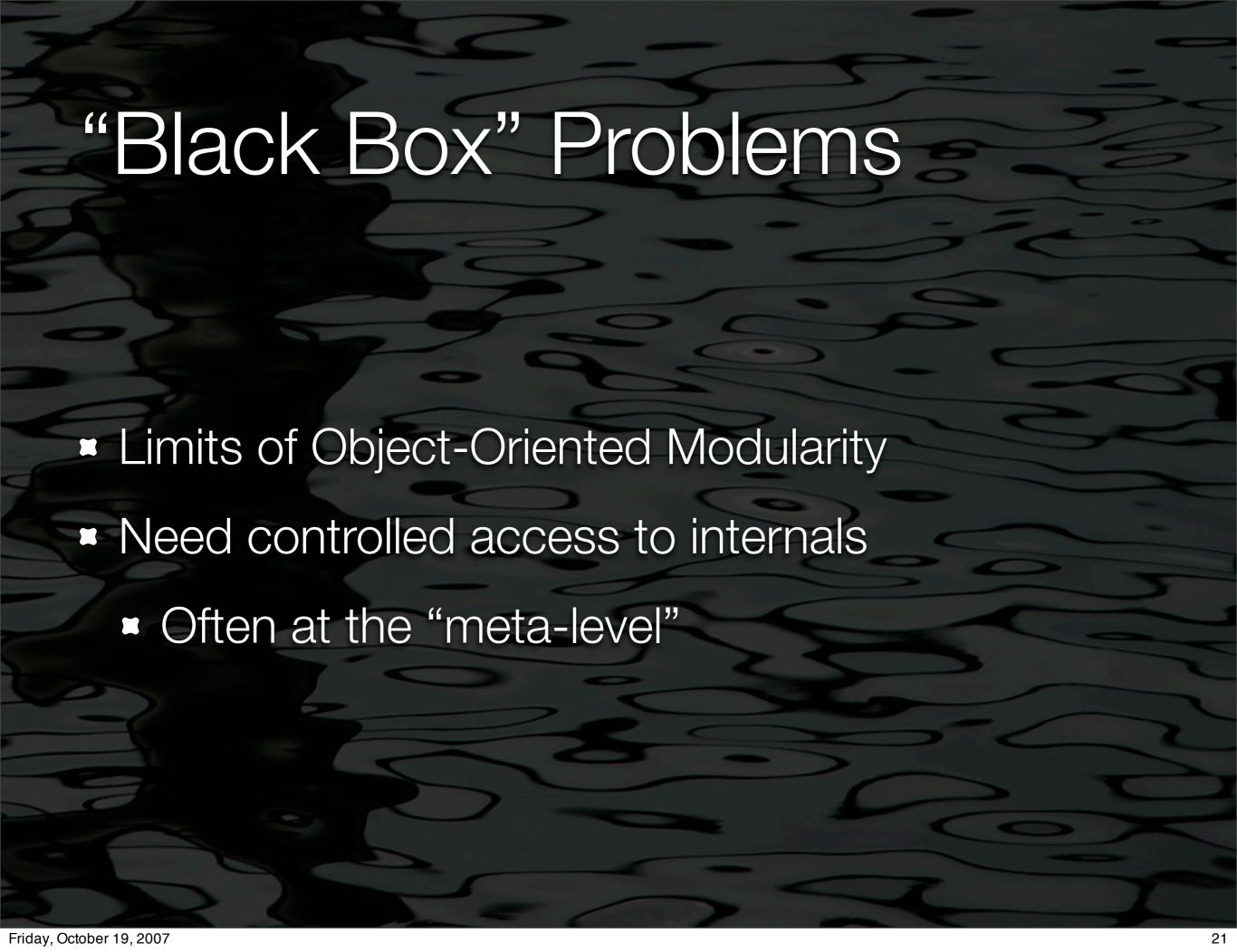






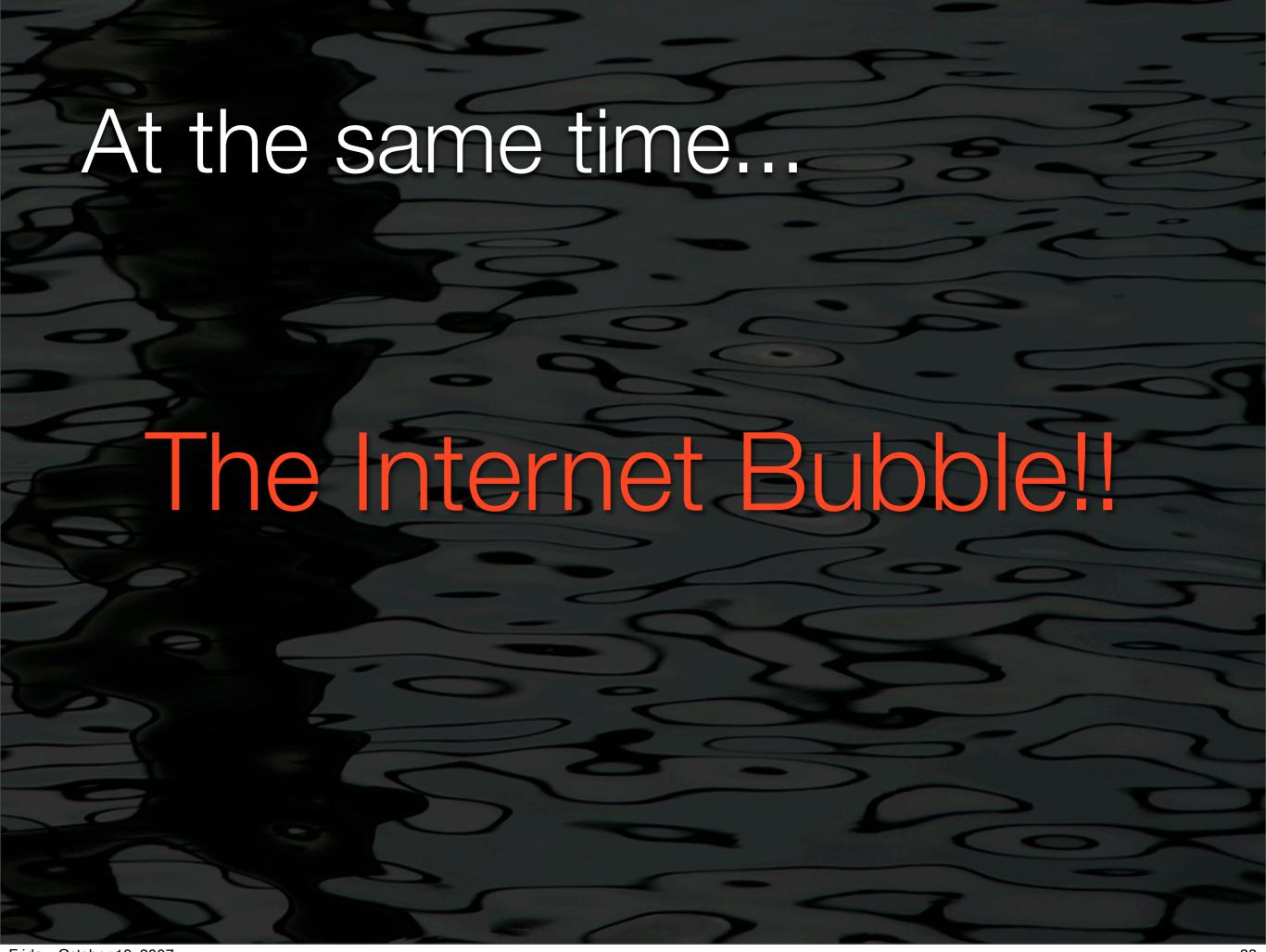


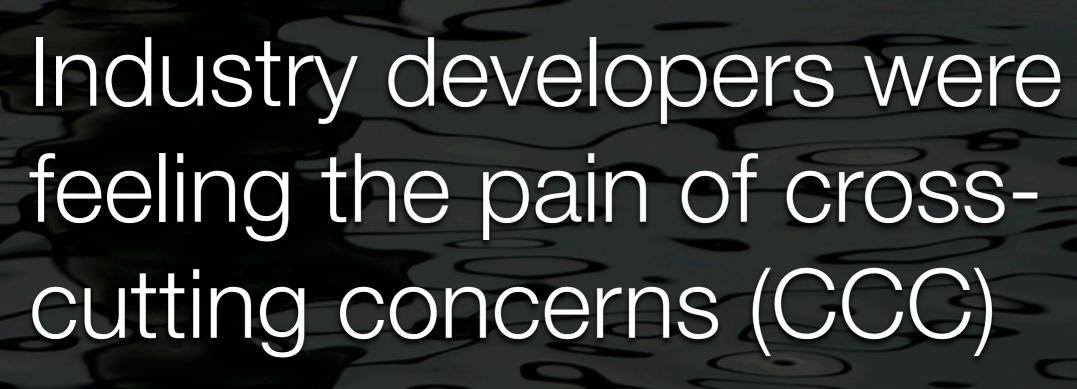






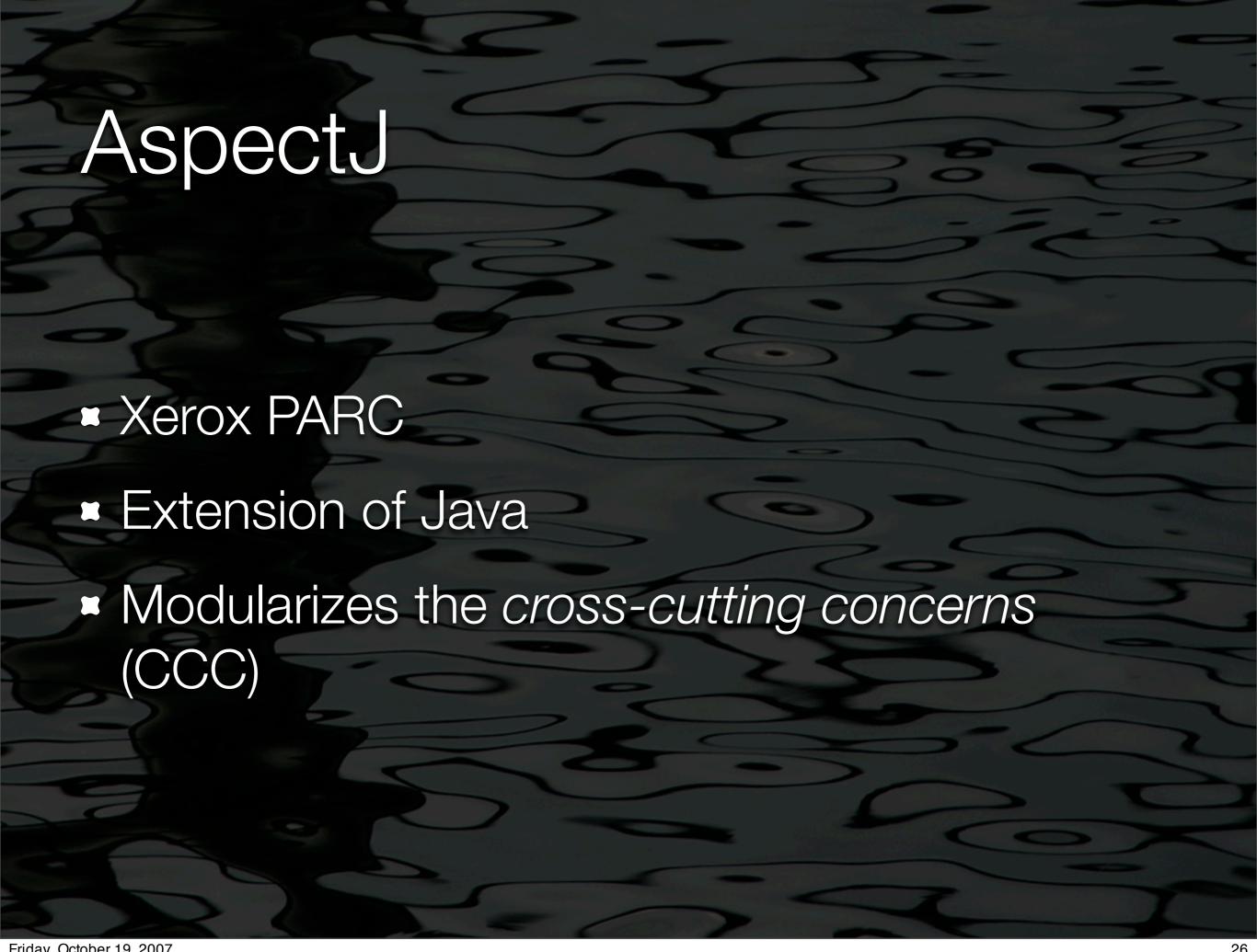
- Metaobject protocols (MOPs) and reflection
- MOPs for
  - File system cache management
  - Virtual memory management tuning
  - Process scheduler tuning

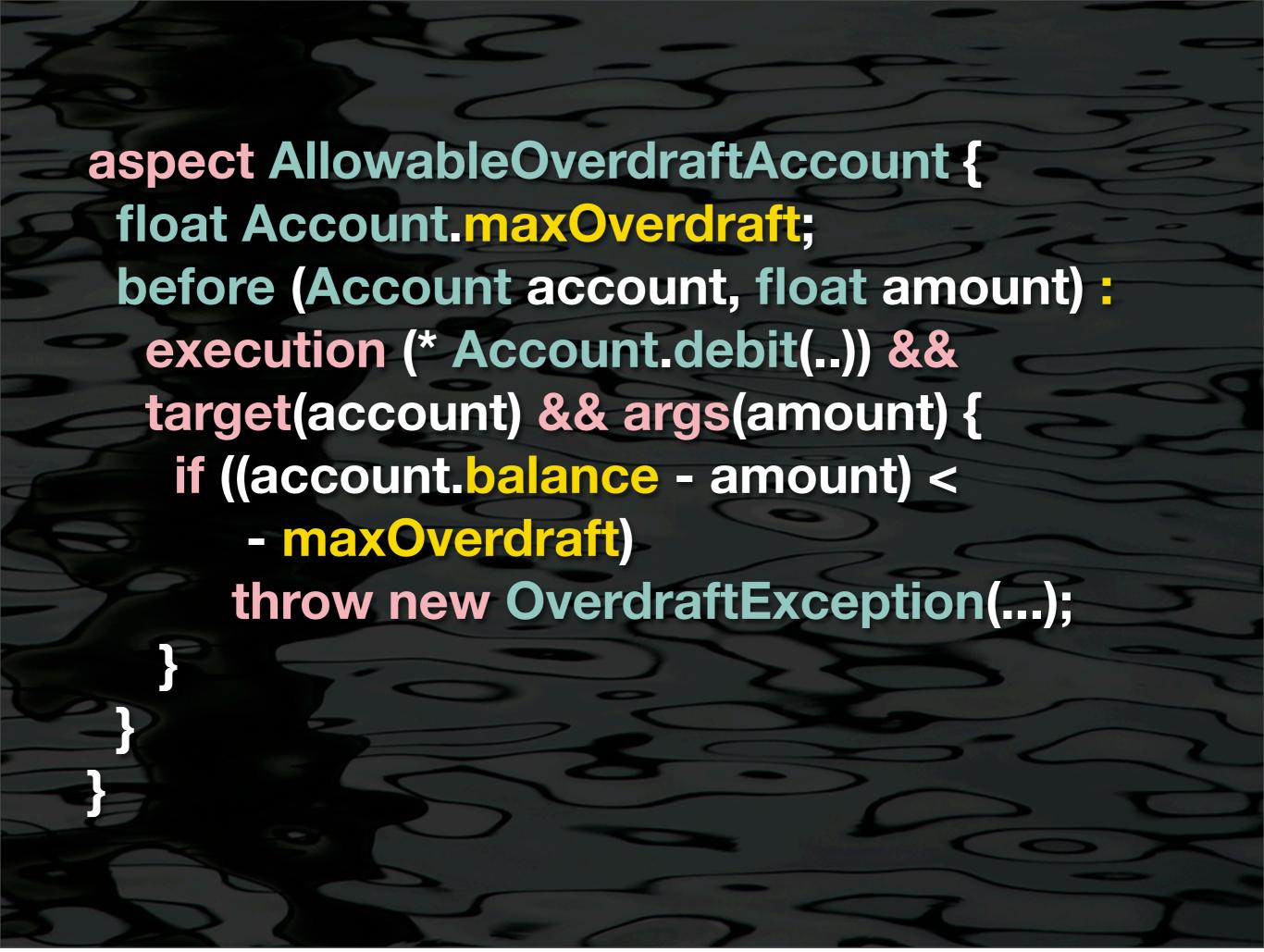




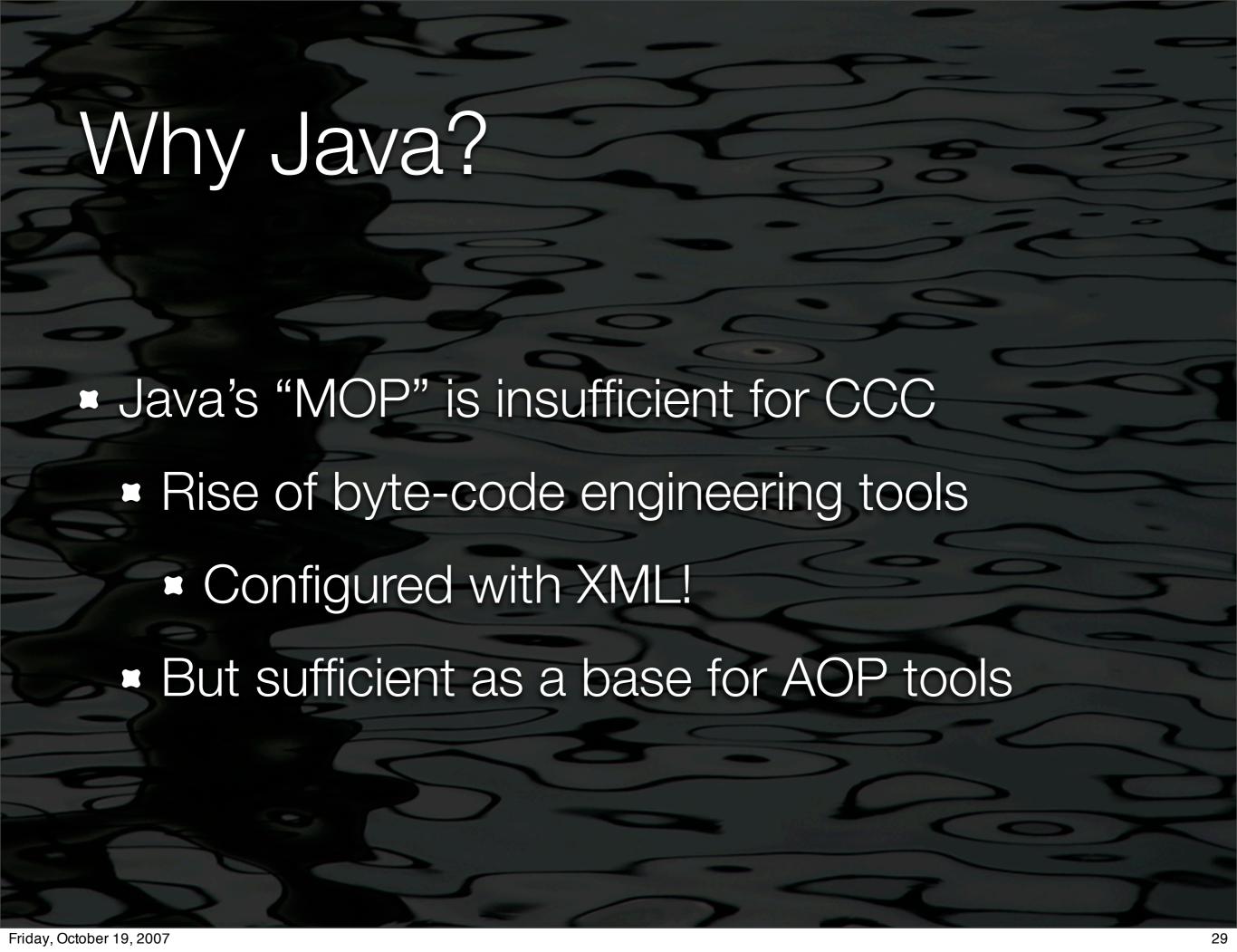
- Persistence
- Transactions
- High availability
- Security

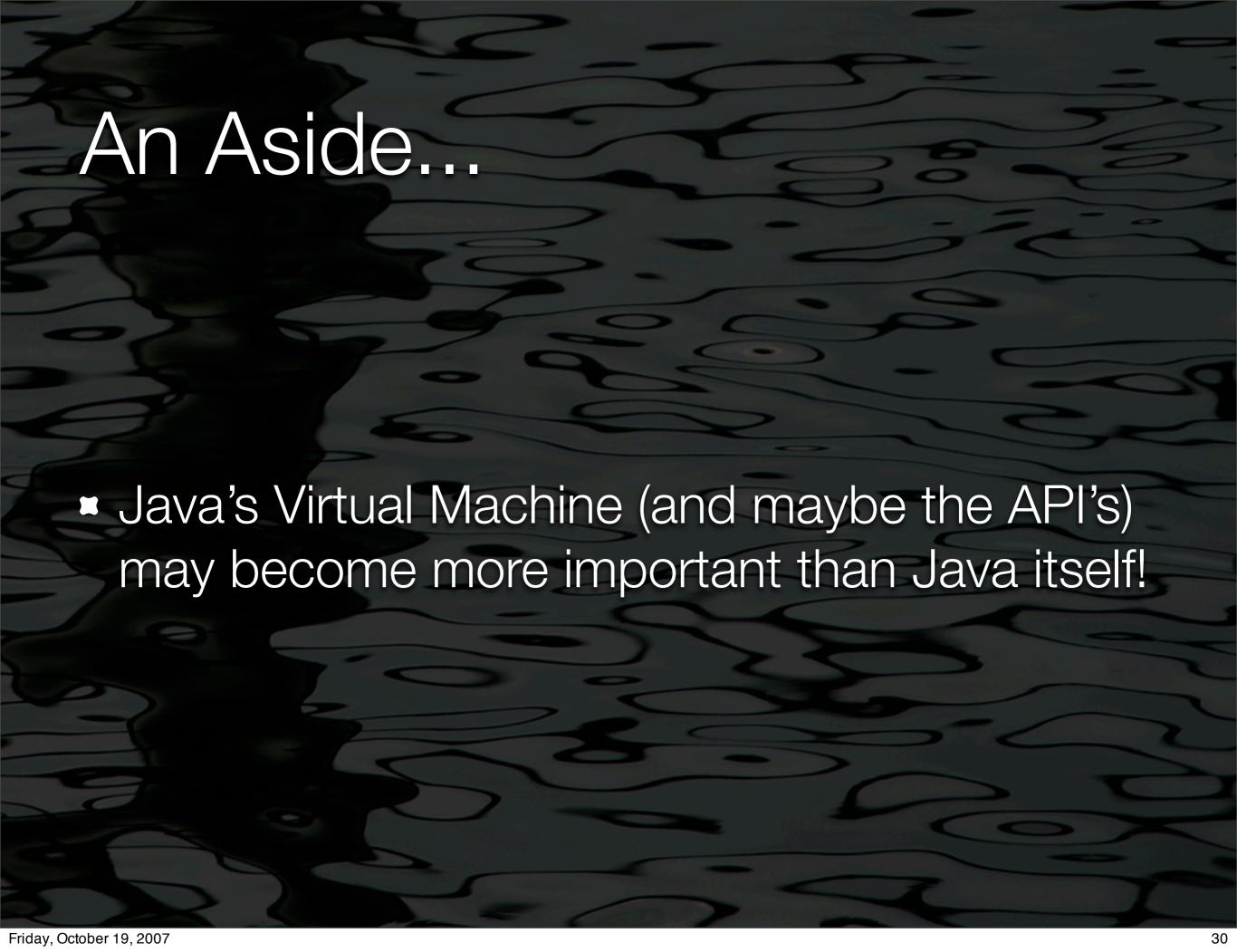


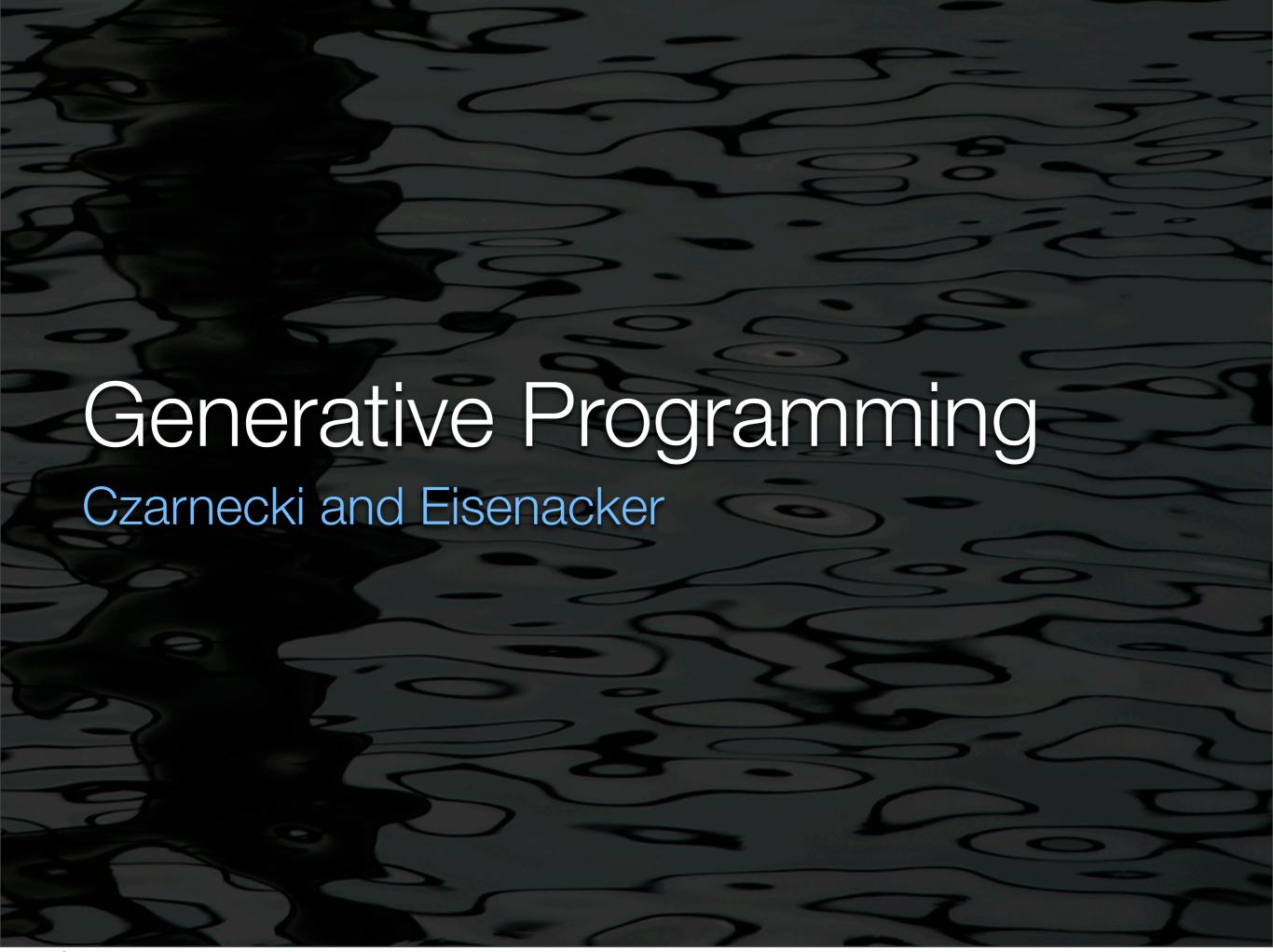


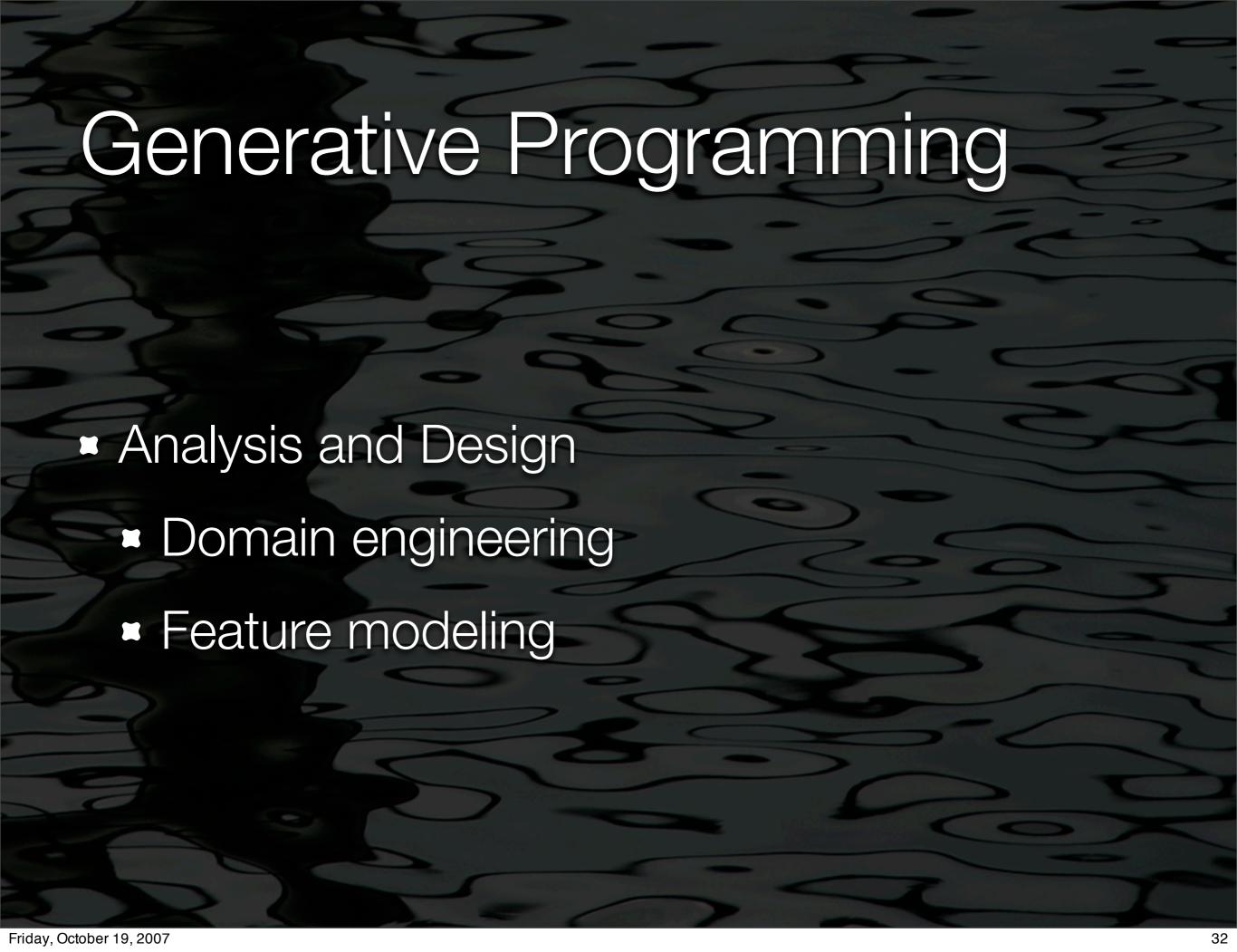














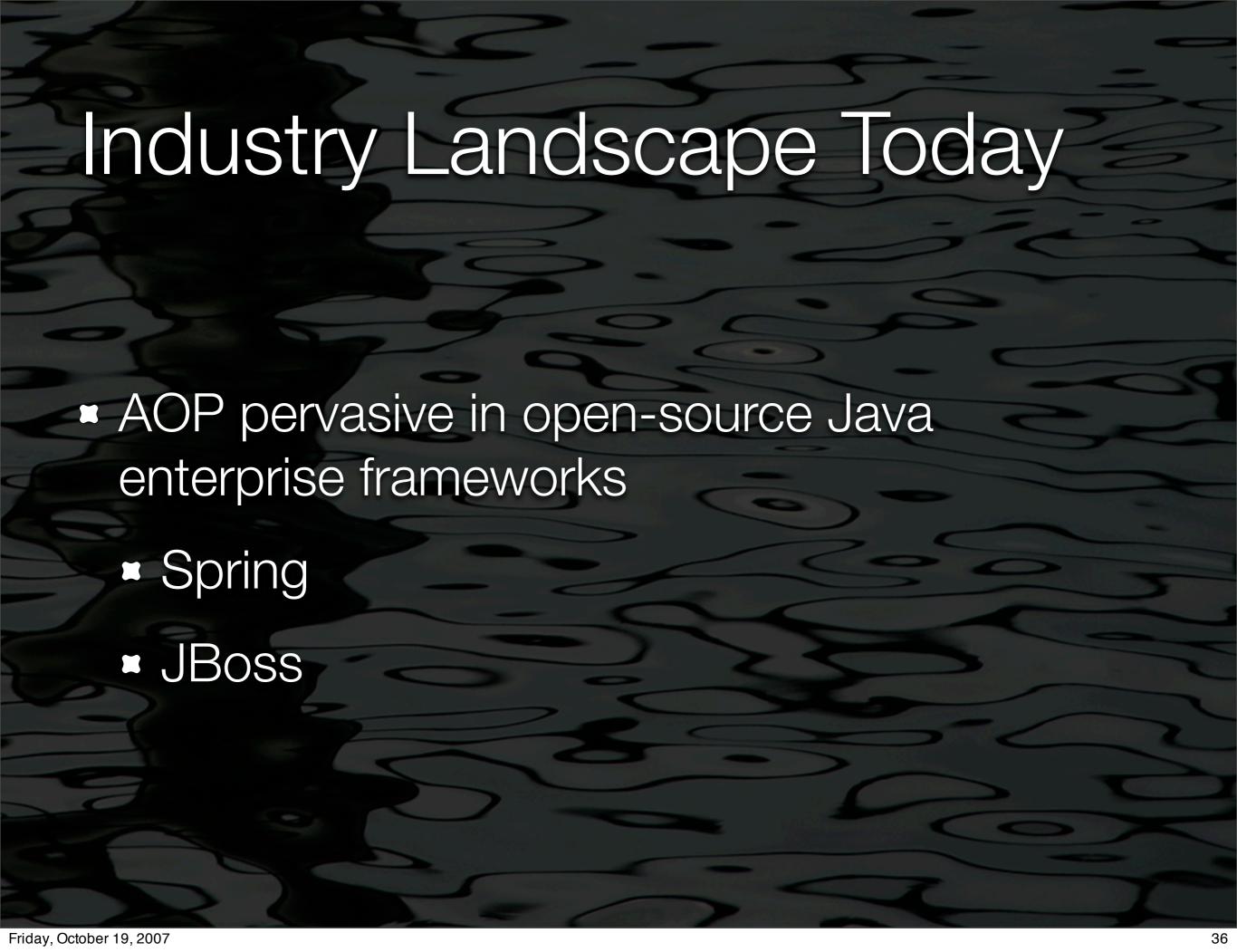
- Implementation Technologies
  - Generic programming
  - C++ template metaprogramming
  - AOP
  - Intentional programming



- IBM Research
- Morphed from "Subject-Oriented Programming"
- Hyper/J
- More ambitious than AspectJ

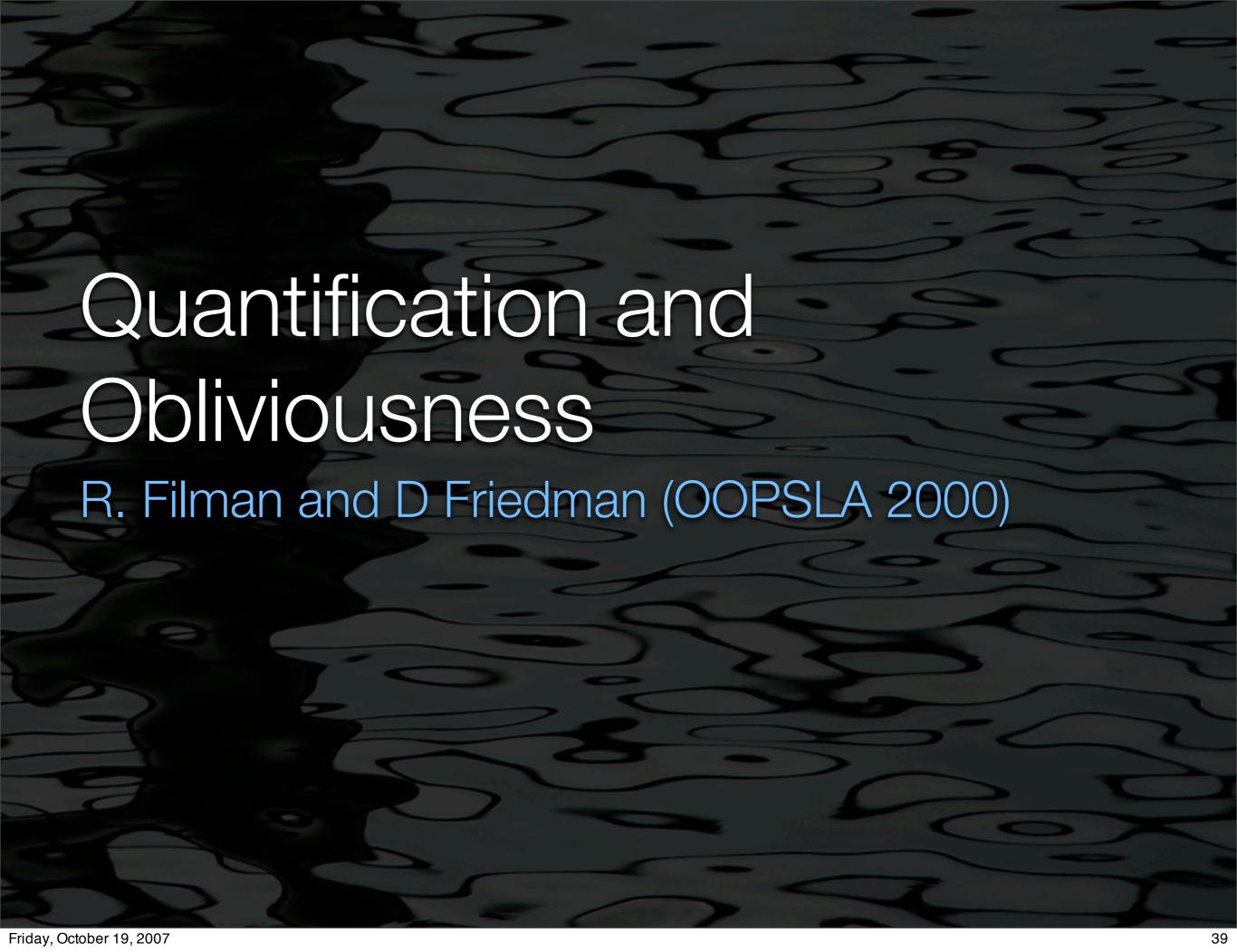
## Multidimensional Separation of Concerns

- Symmetric AOP
  - Aspects as first-class citizens, like classes
- Asymmetric AOP
  - Aspects as "adjuncts"
    - AspectJ's de facto model

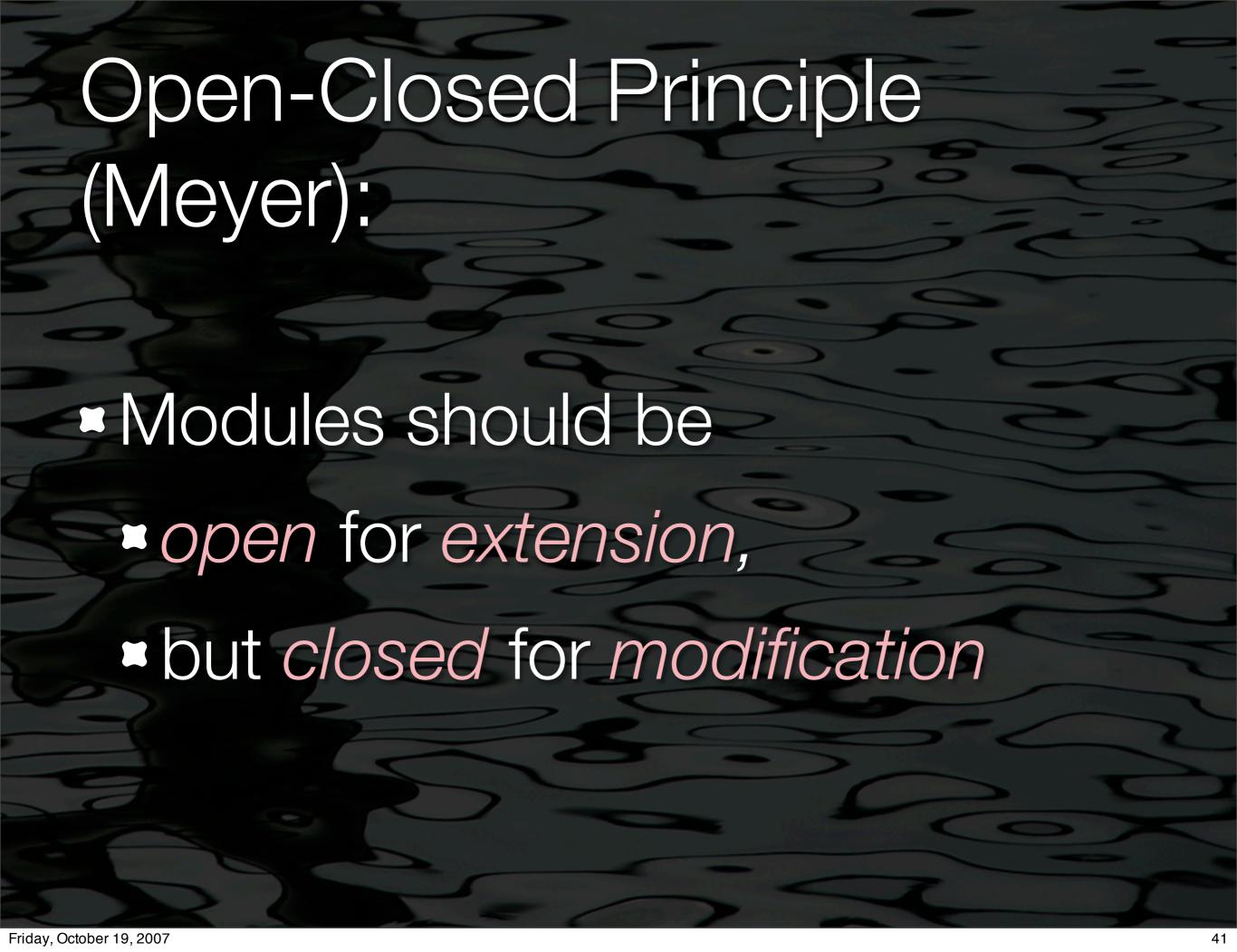


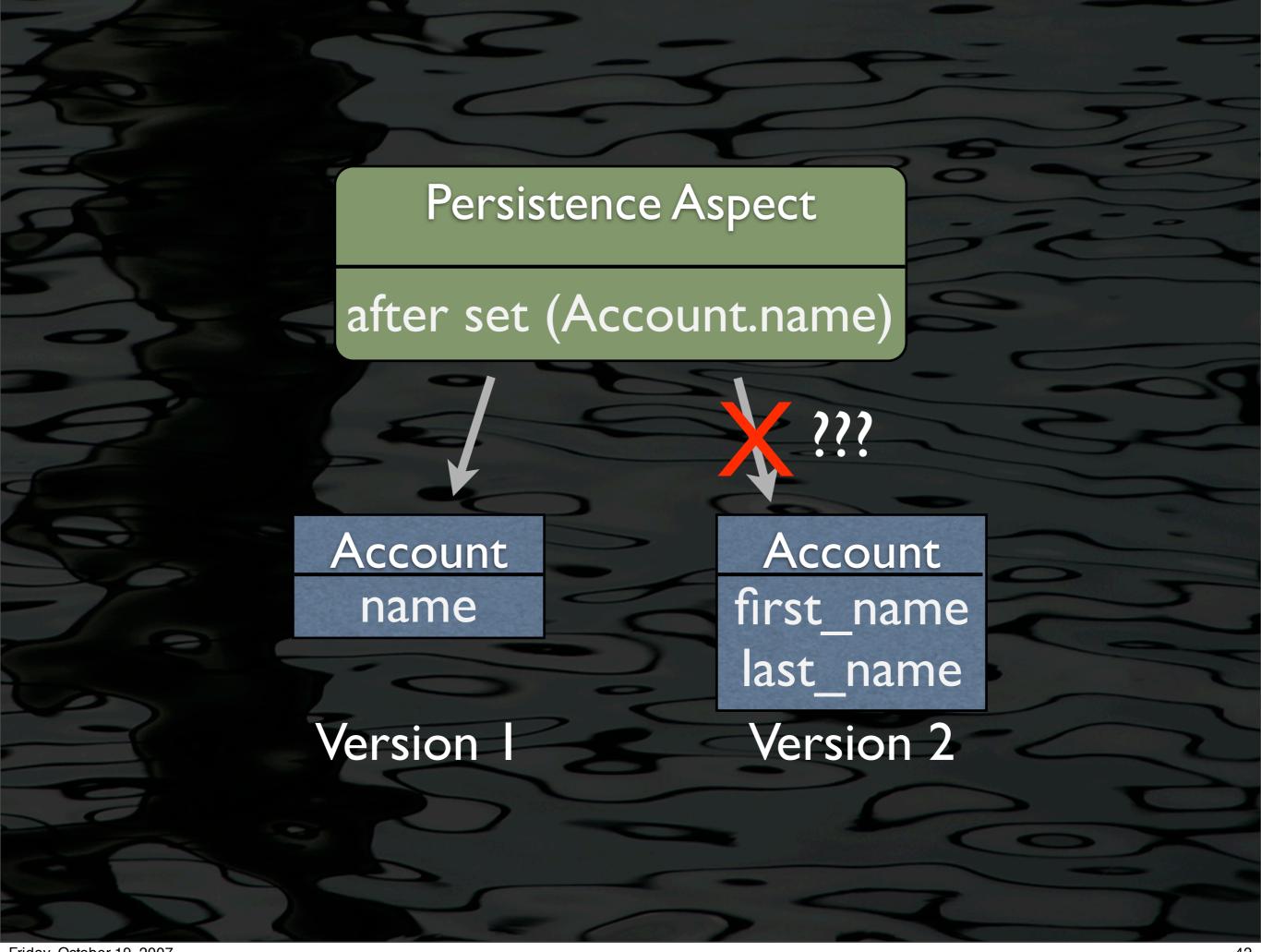


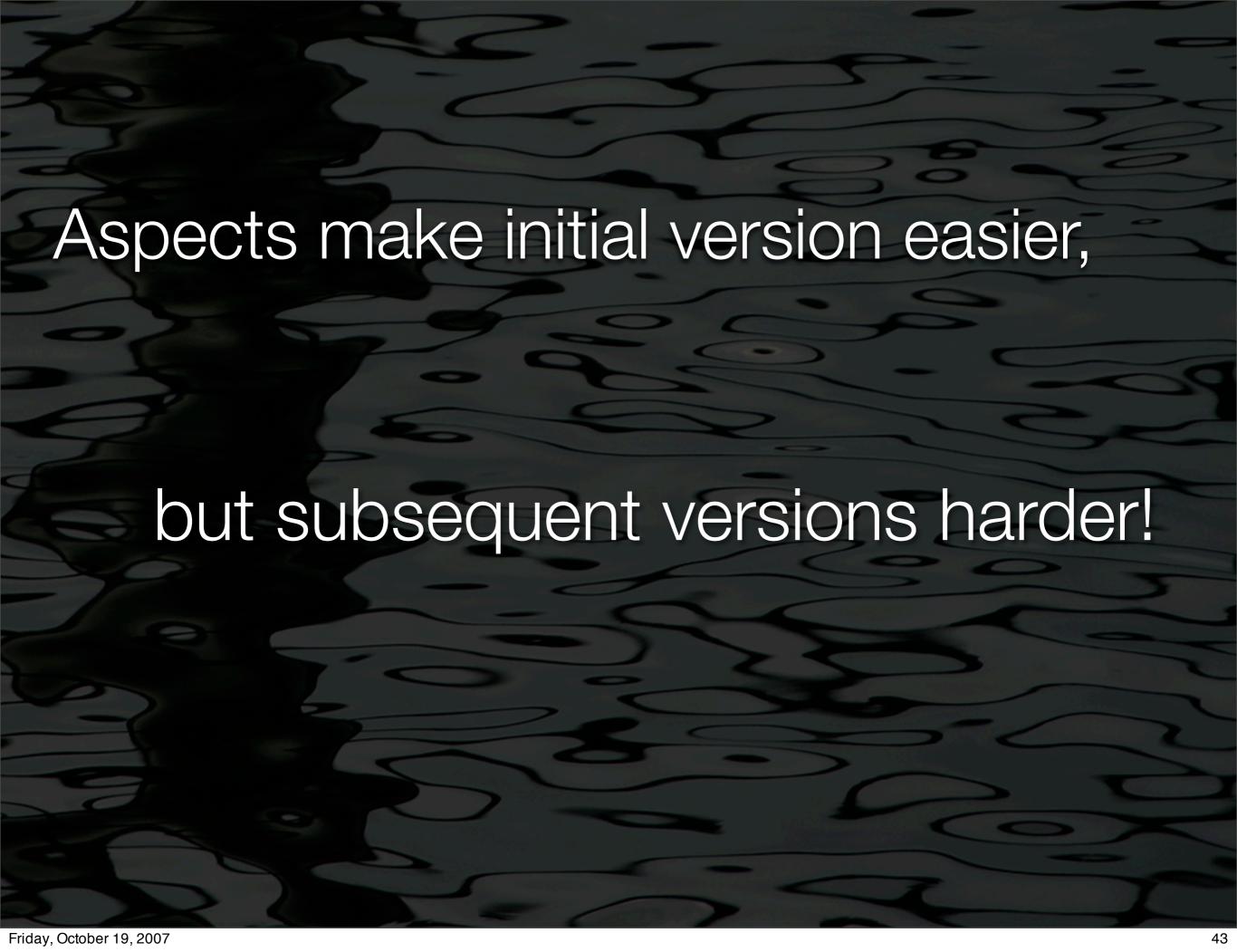


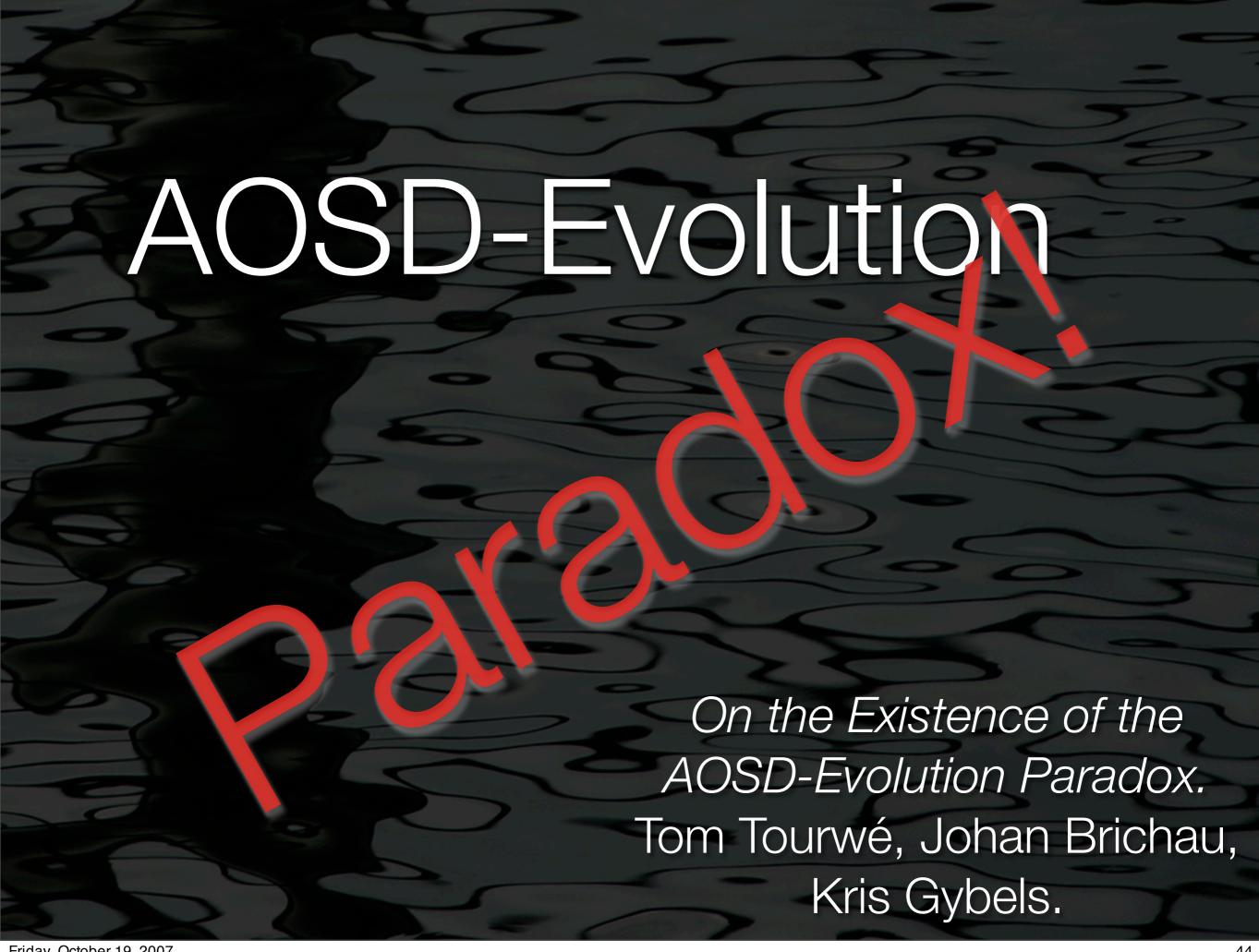


AOP can be understood as the desire to make quantified statements about the behavior of programs, and to have these quantifications hold over programs written by oblivious programmers.

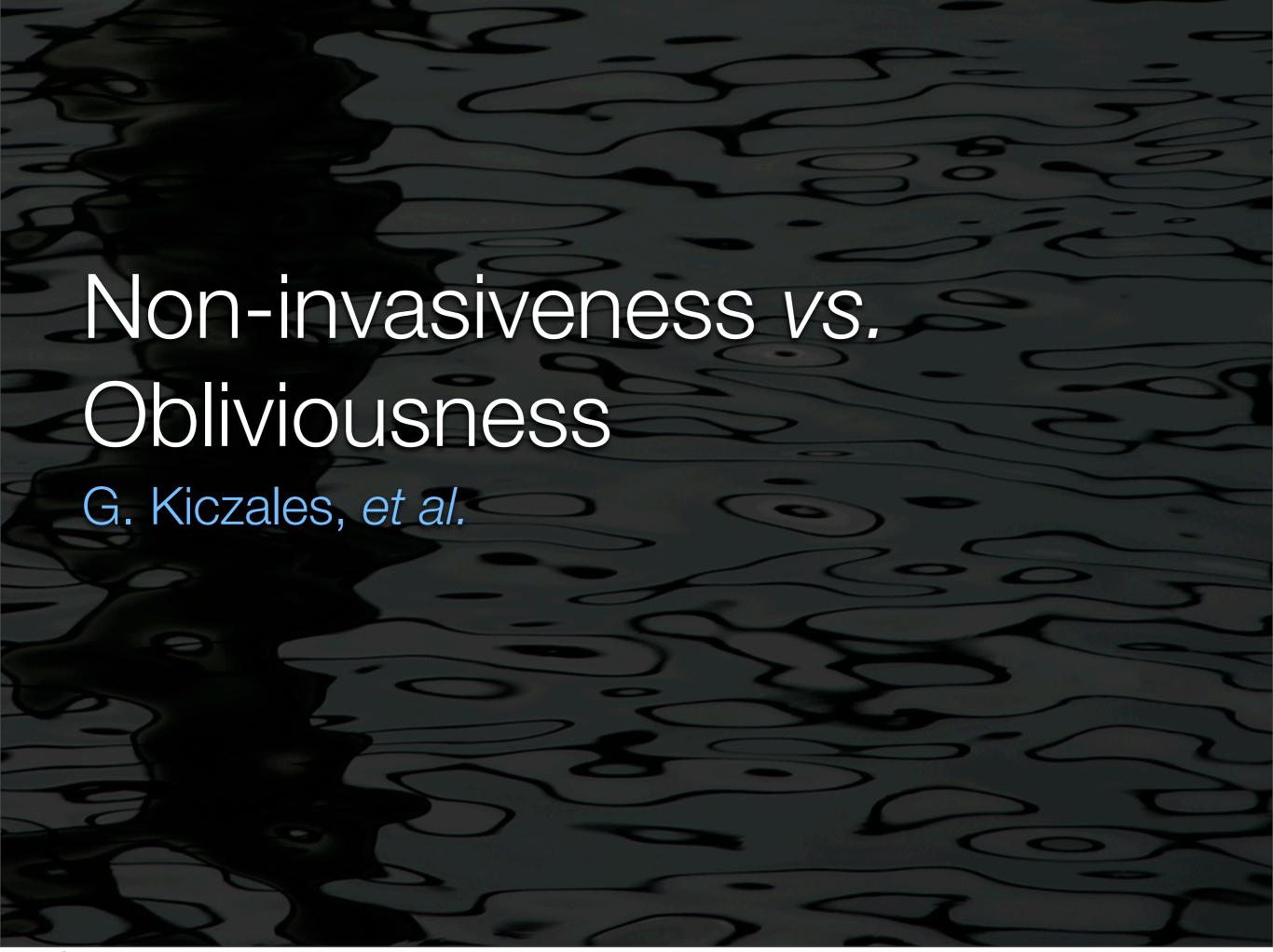




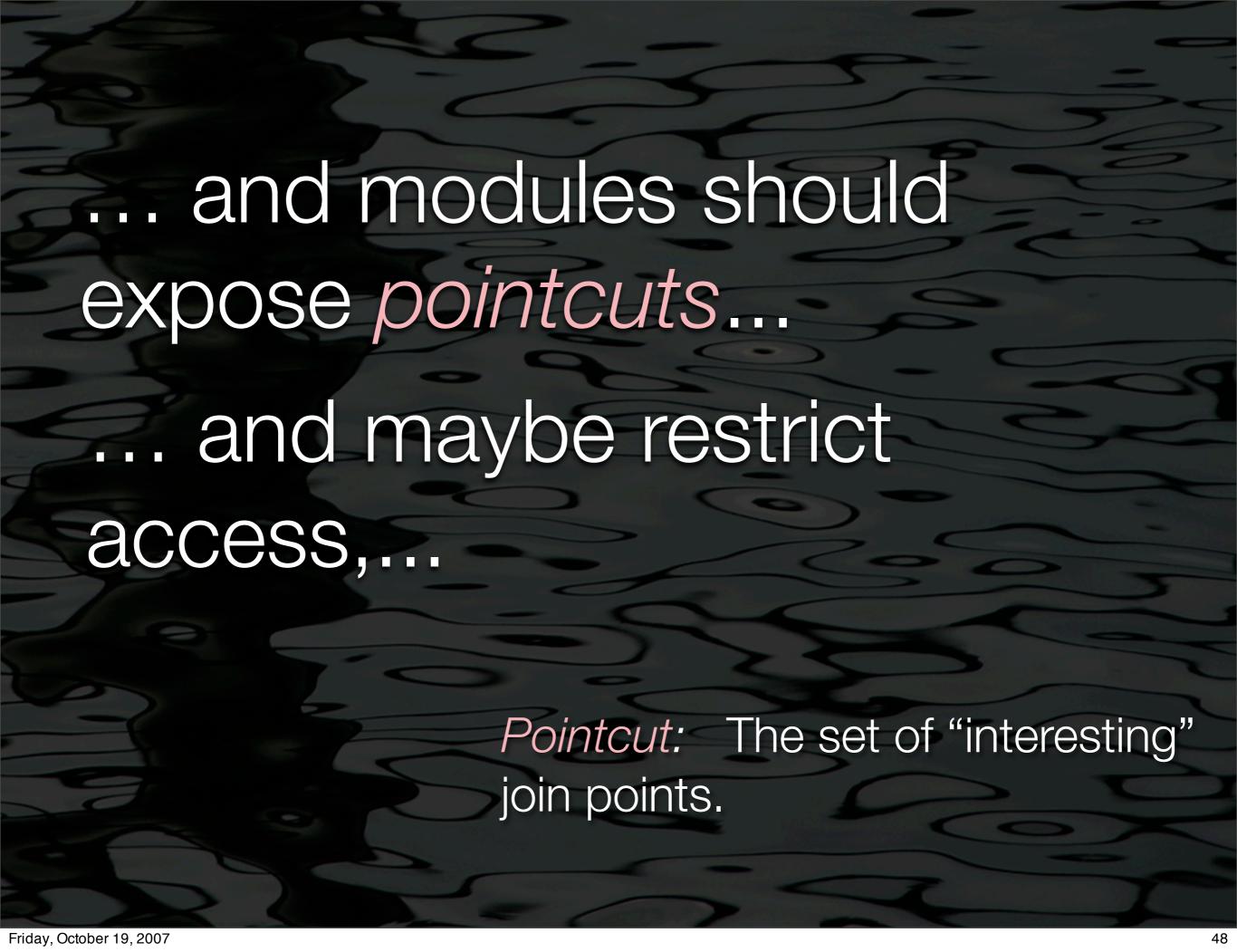


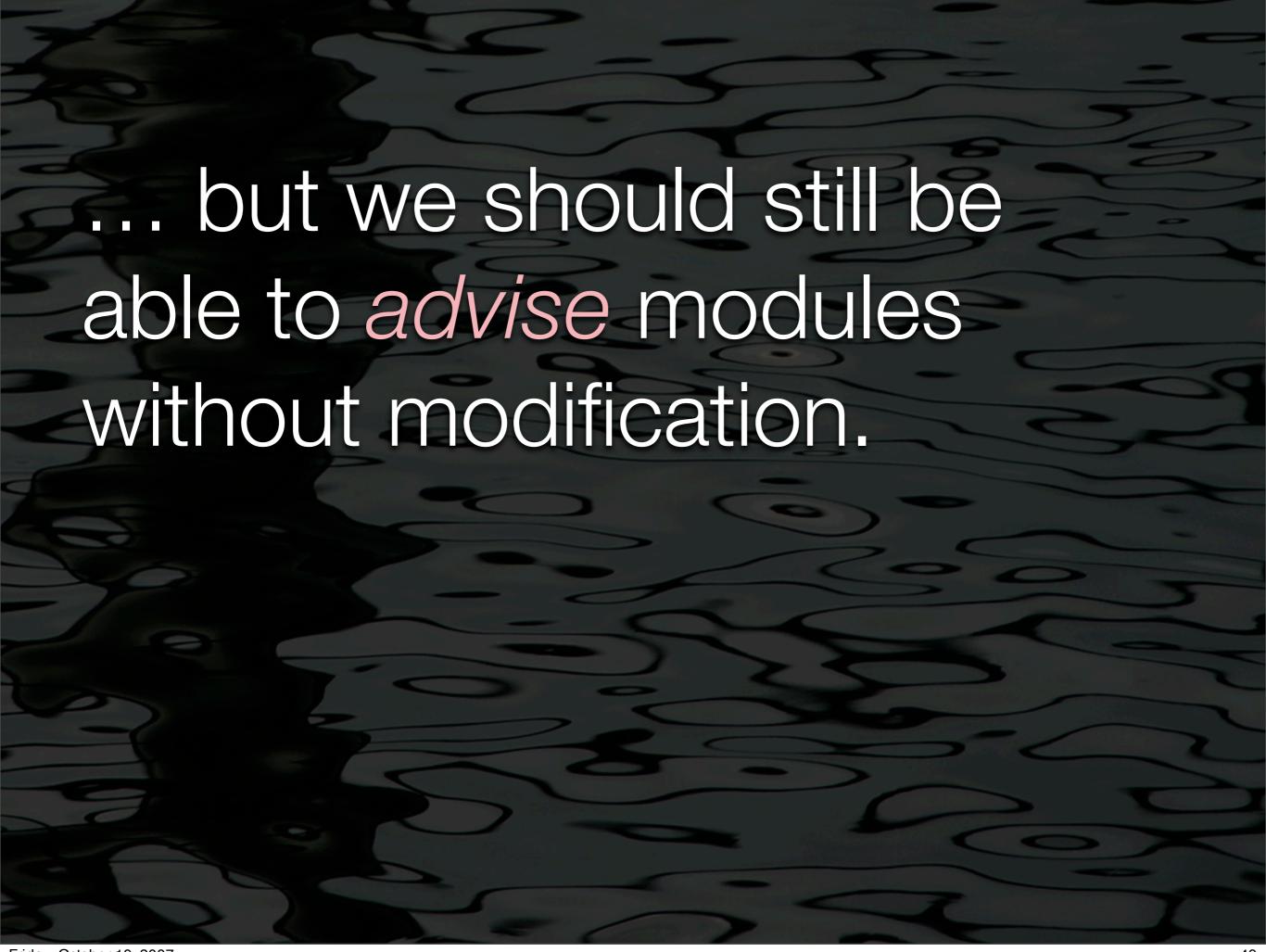


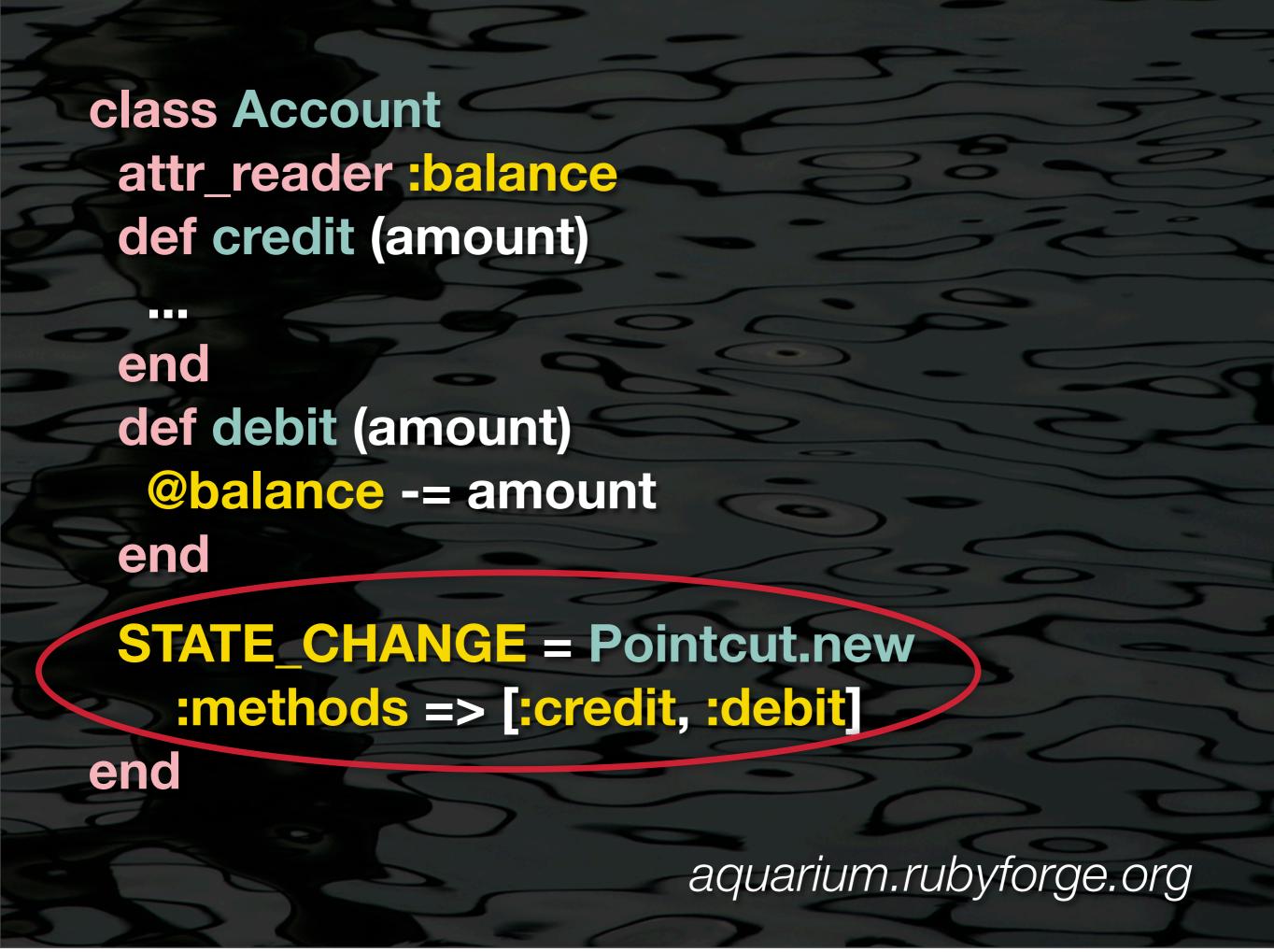


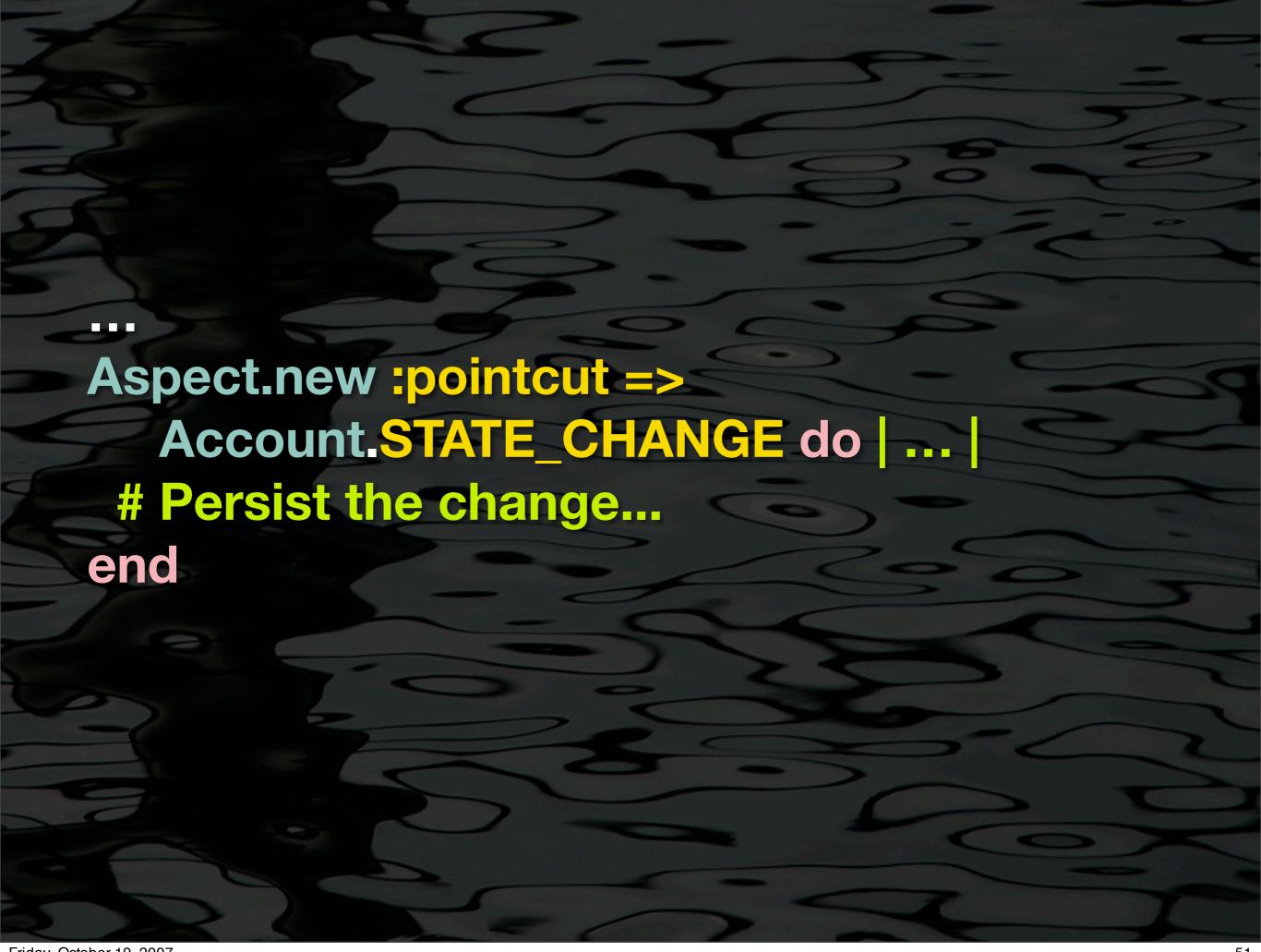


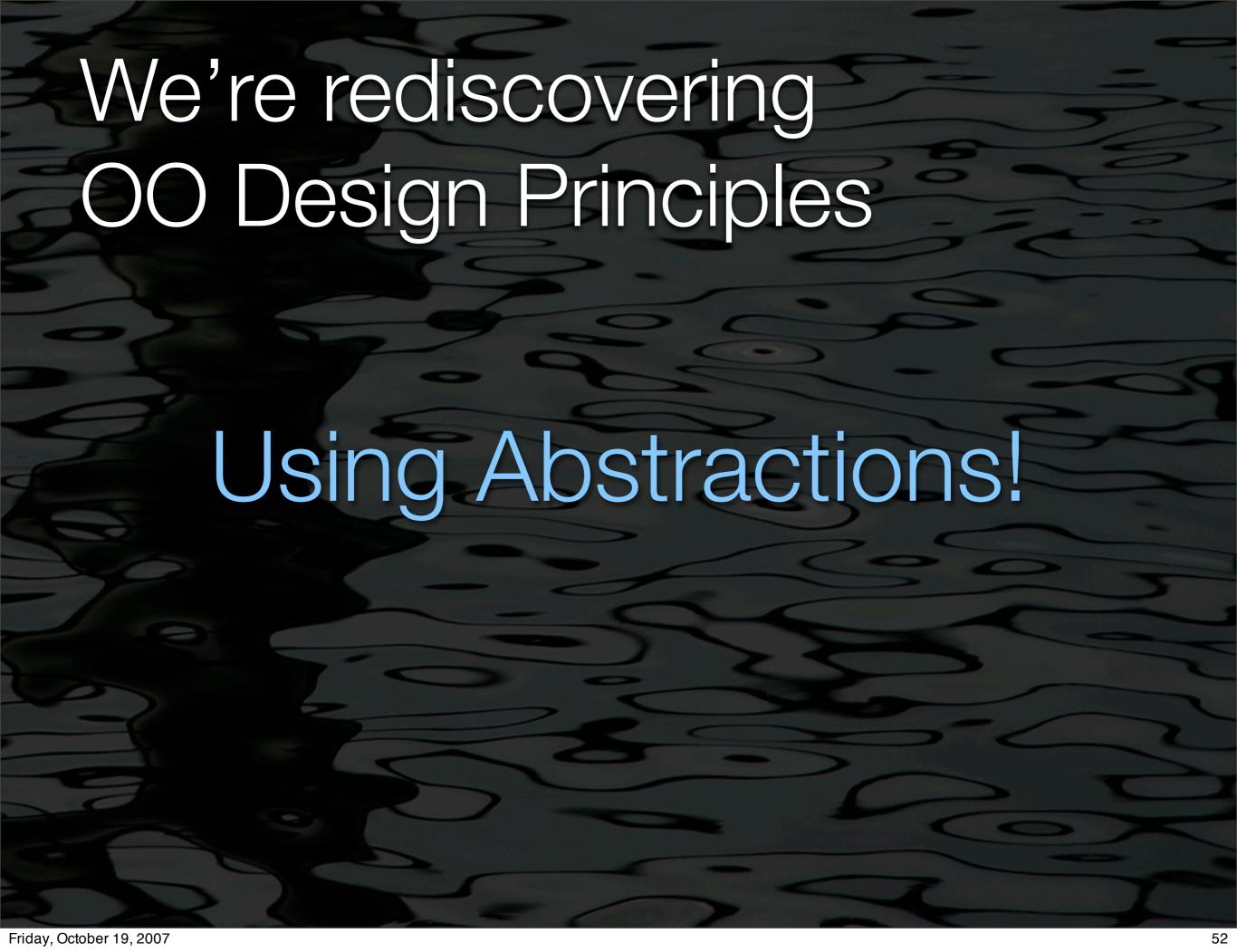












## For Completeness...

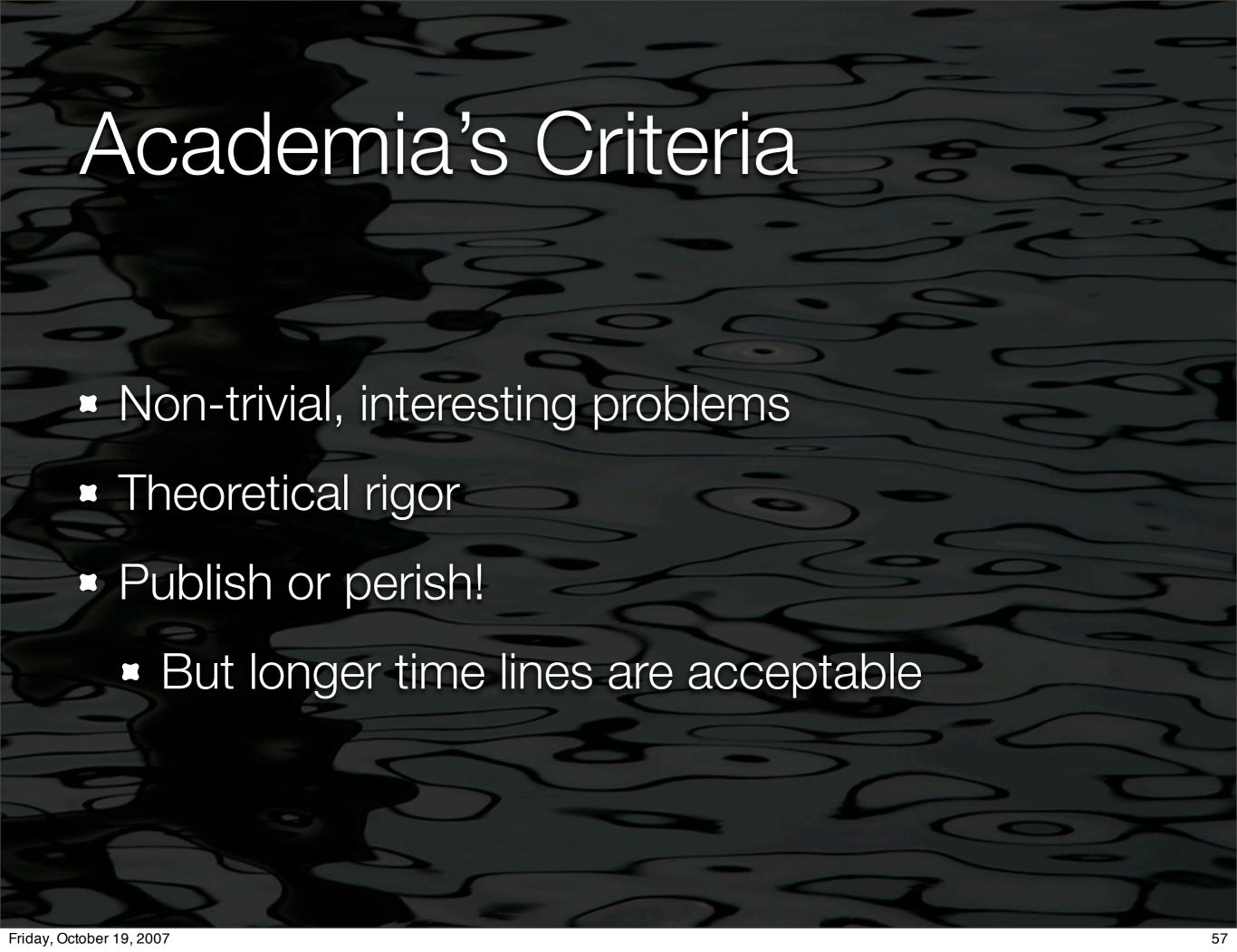
- Open Modules
  - Modular Reasoning About Advice
    - J. Aldrich
- Cross-Cutting Programming Interfaces (XPI)
  - Modular Software Design with Crosscutting Interfaces
    - Griswold, Sullivan, et al.





- Simple (enough) to understand and use
- Strong tool support
- Maintainability of long-lived software
- We must get paid, ASAP!

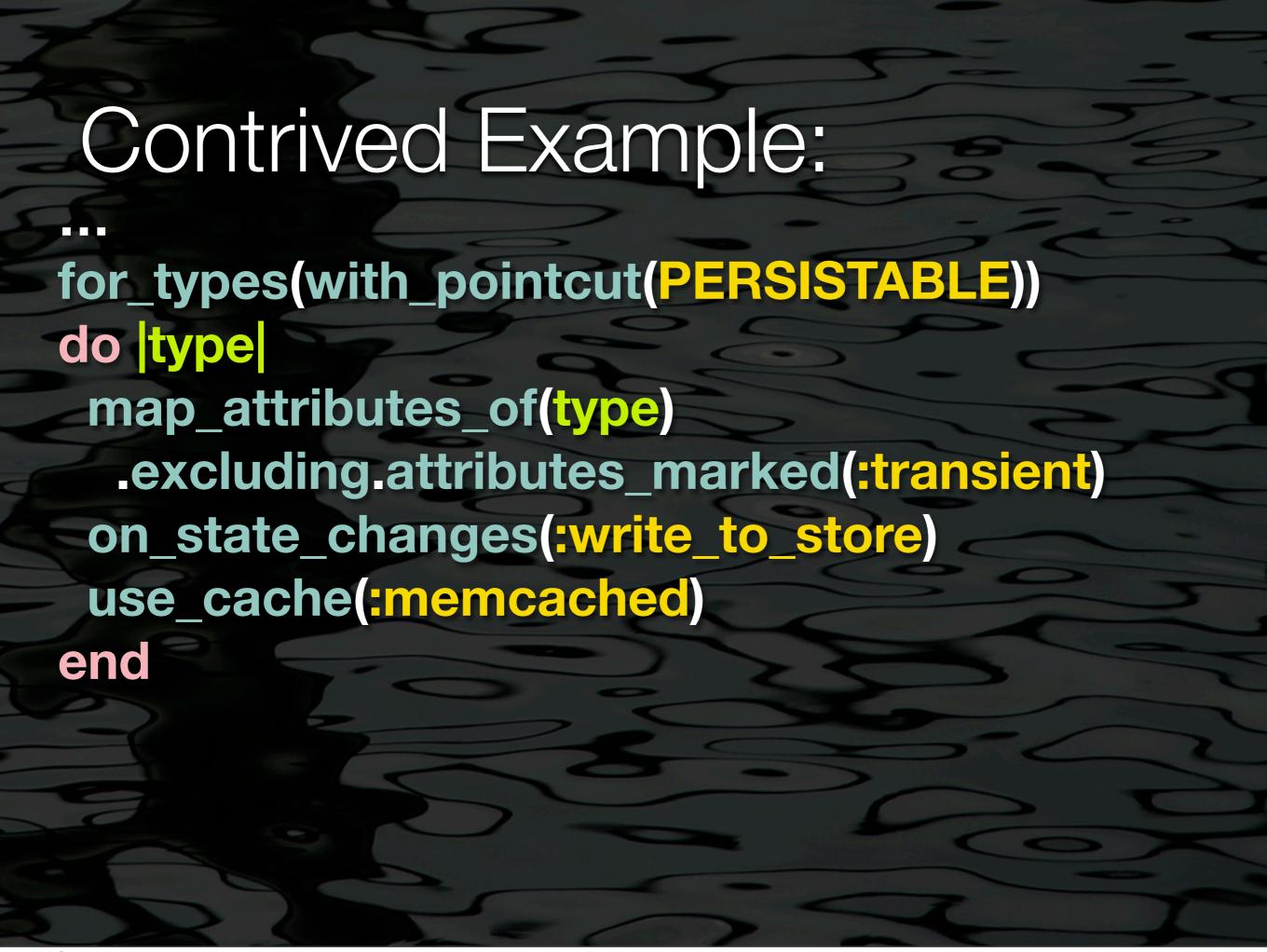








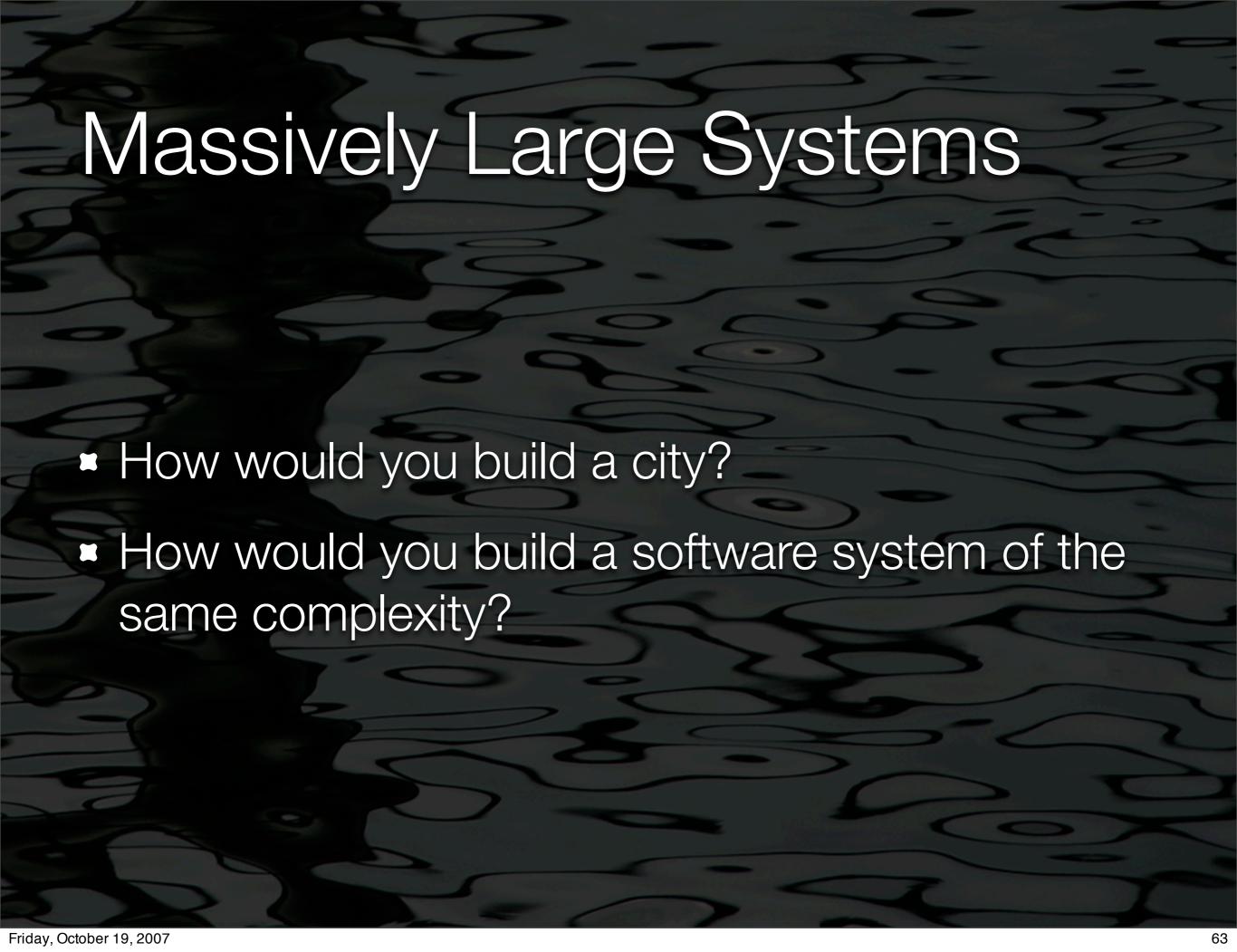
- Raise the abstraction level by constructing Domain Specific Languages (DSLs)
- Could hide the complexity of aspects, objects, metaprogramming, etc.





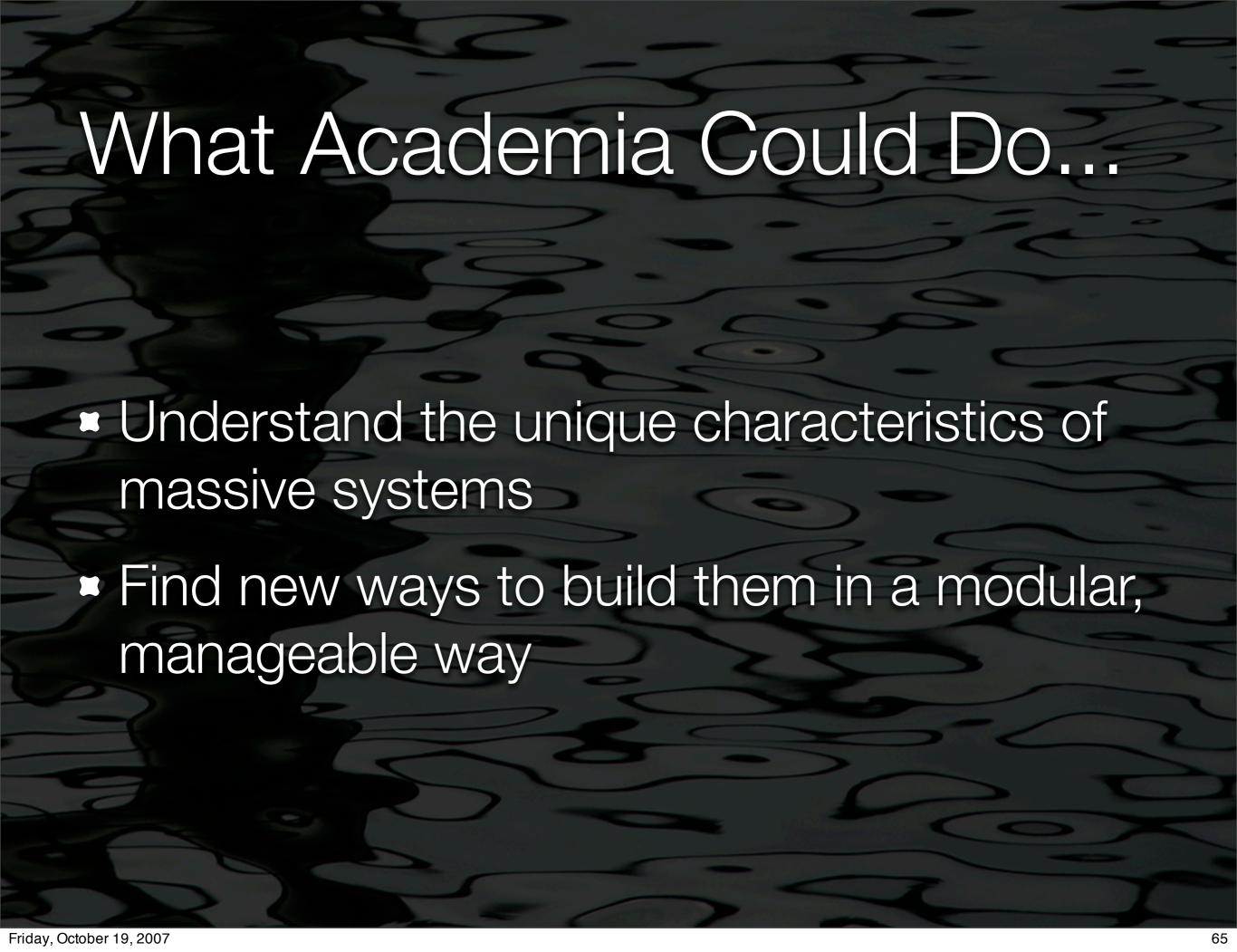


- You understand language design, Al, etc.
- Help industry understand
  - Globally-applicable DSL design principles
  - Mapping DSLs to object, aspect, ... assembly code





- Incremental improvements on what we already know how to do
- Build systems whose complexity exceeds the capabilities of our modularity tools
- Struggle to maintain these systems...





- Don't worry too much about industry relevance!
  - We need people working on longer-term problems
- Instead of incremental improvements...
  - Focus on fundamental problems and innovation!

