Sireum/Kiasan

an extensible symbolic execution framework

Design and Architecture Overview

Funding

NSF AFOSR AIR FORCE OFFICE OF SCIENTIFIC RESEARCH Rockwell Collins

CIS @ K-STATE COMPUTING AND INFORMATION SCIENCES

SANTOS LAB

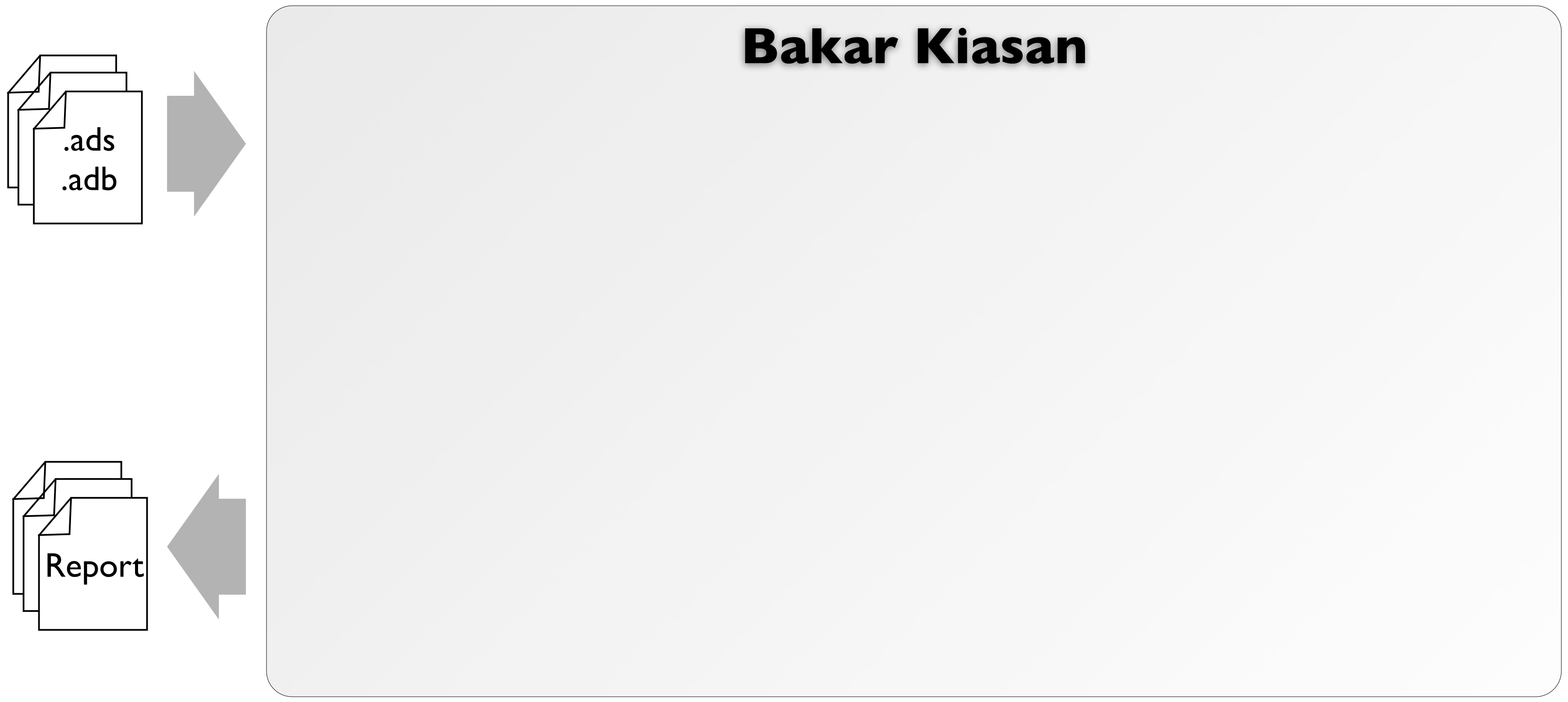
Sireum/Kiasan: Design Goals

- An extensible SymExe framework
  - easy to customize semantics
- ... designed to be highly parallel
  - leverage (massively) multi-core machines
- ... designed to be distributable
  - leverage clusters of machines

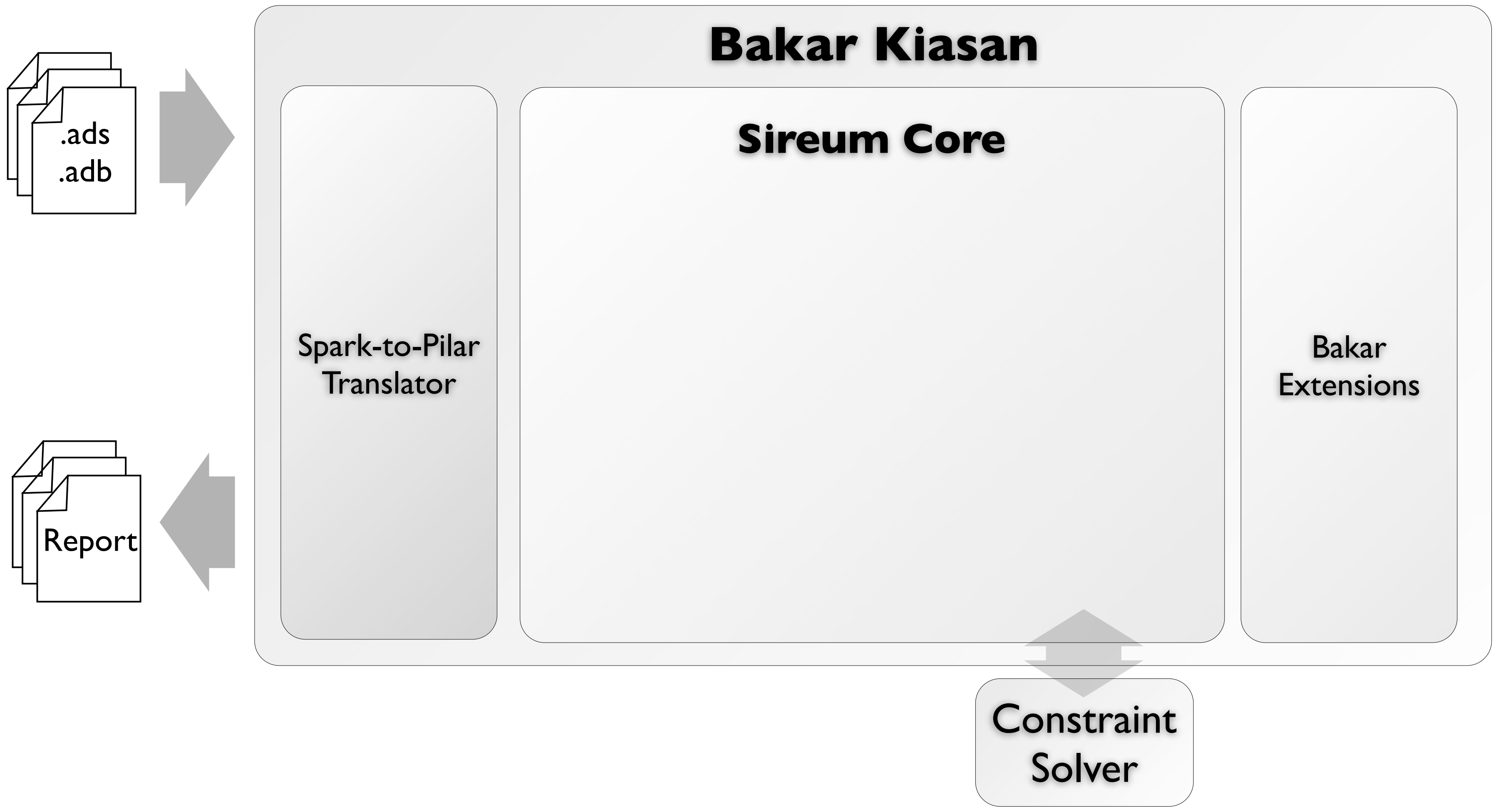
SymExe Components

- Program Representation
- Semantic Domains
  - state and value
- Executions
  - concrete and symbolic
- Constraint Solver

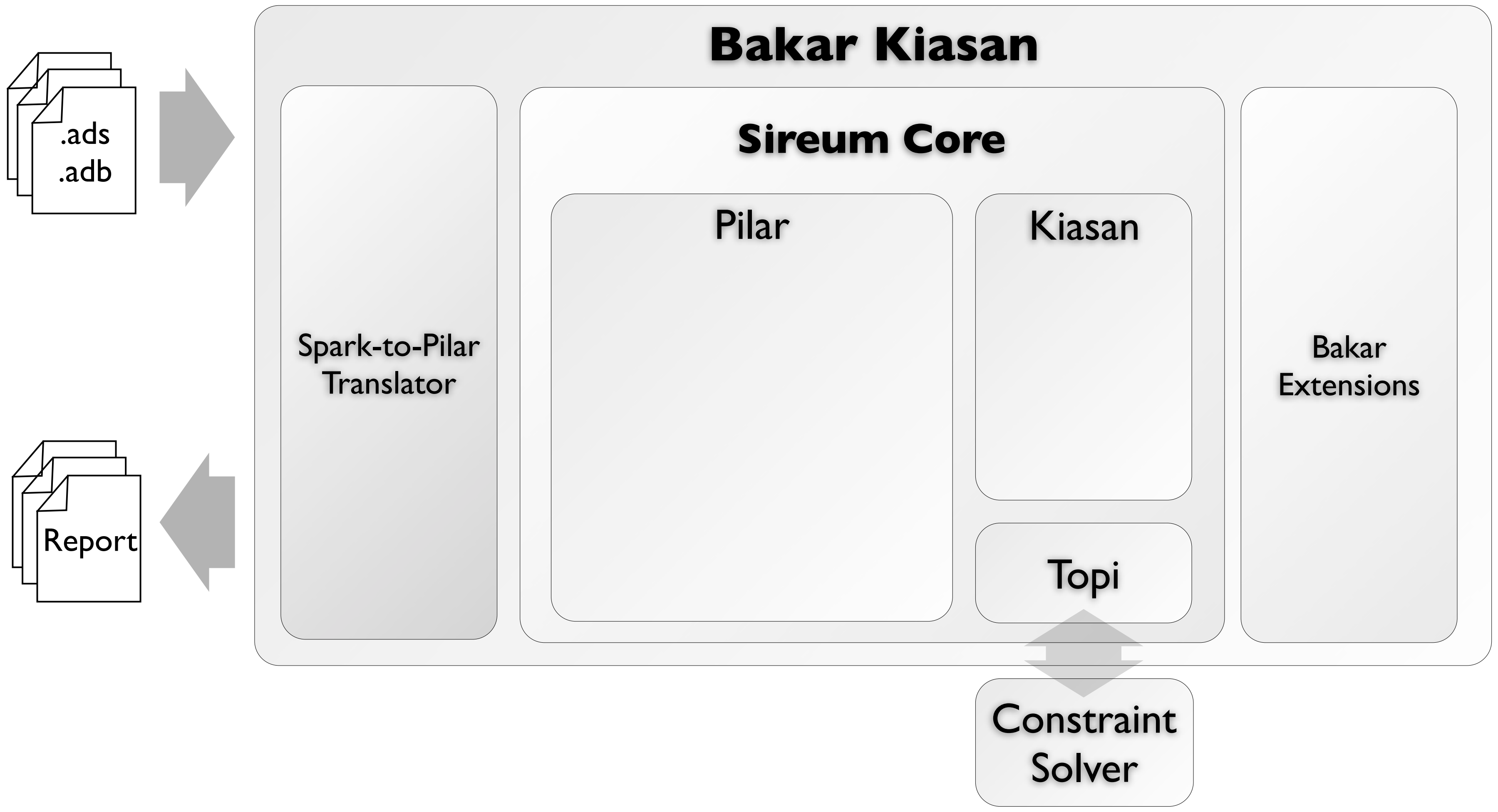
Architecture Overview



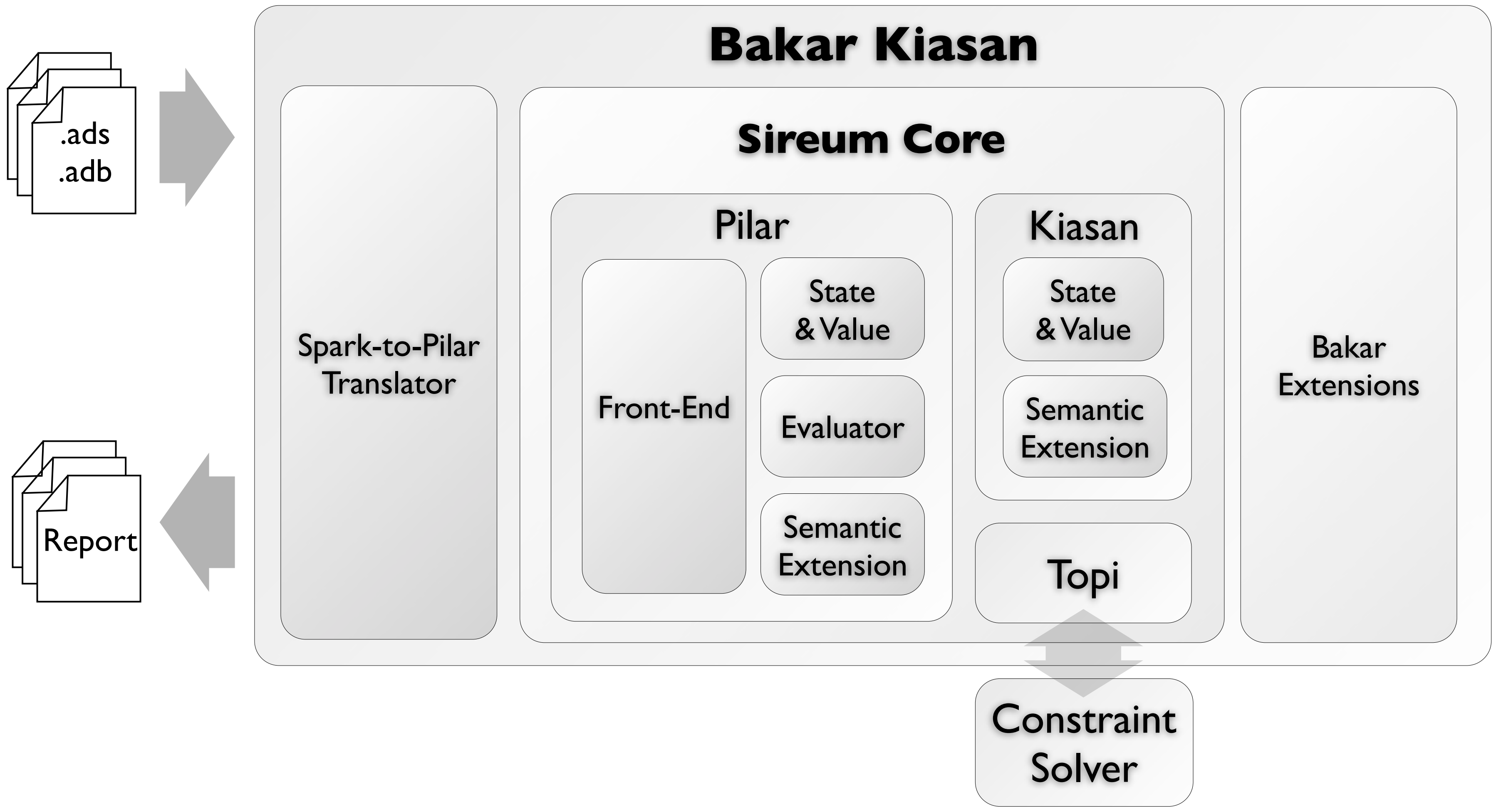
Architecture Overview



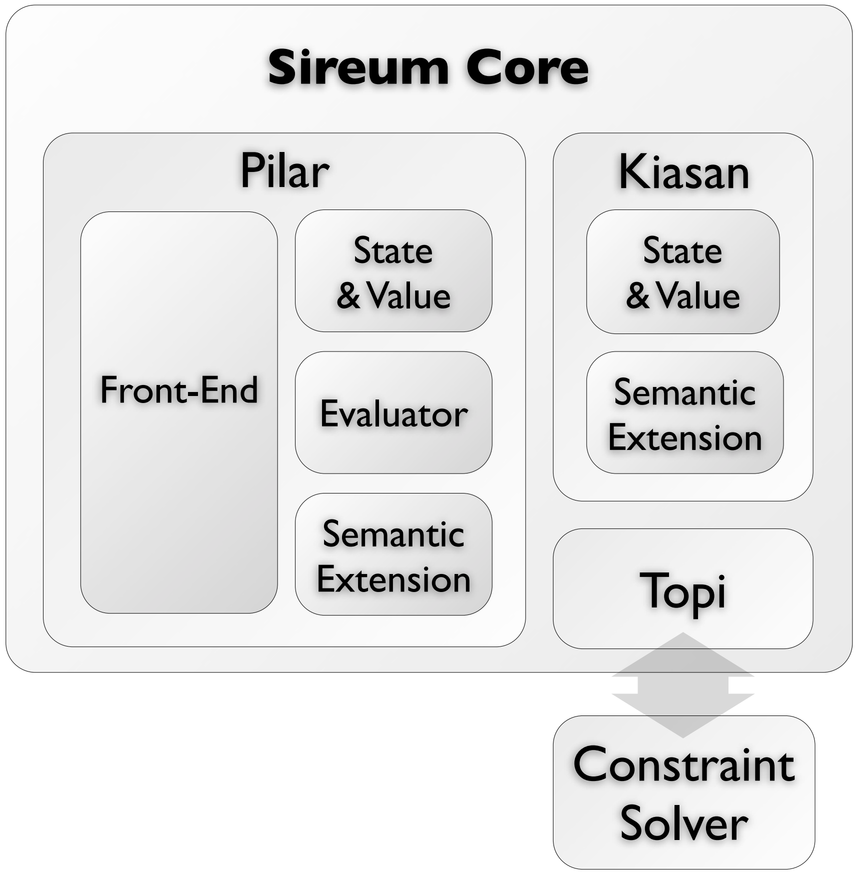
Architecture Overview



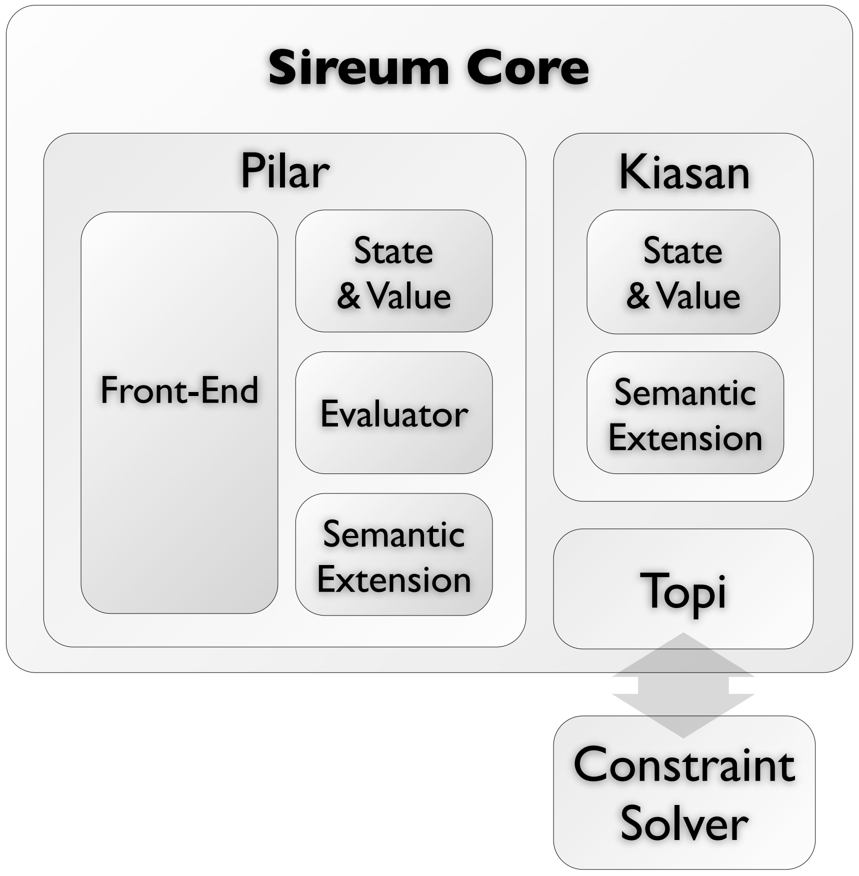
Architecture Overview



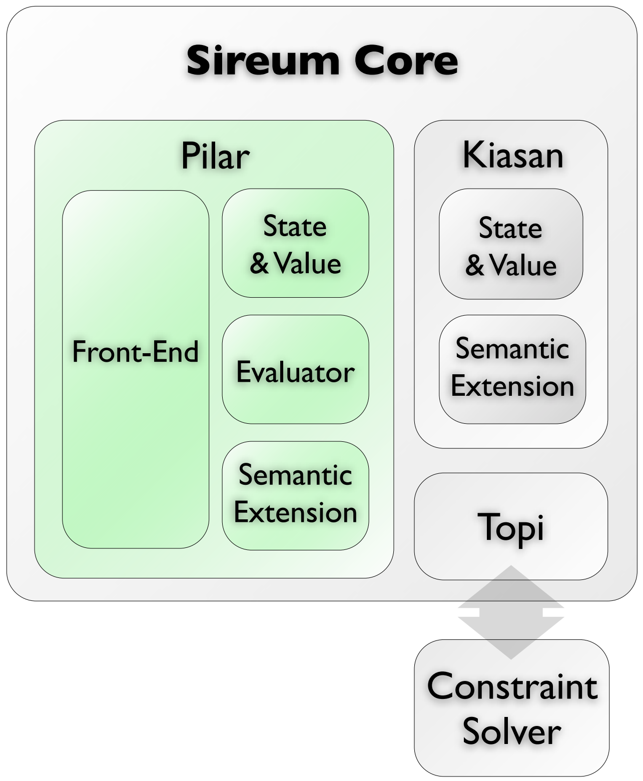
Architecture Overview



Architecture Overview

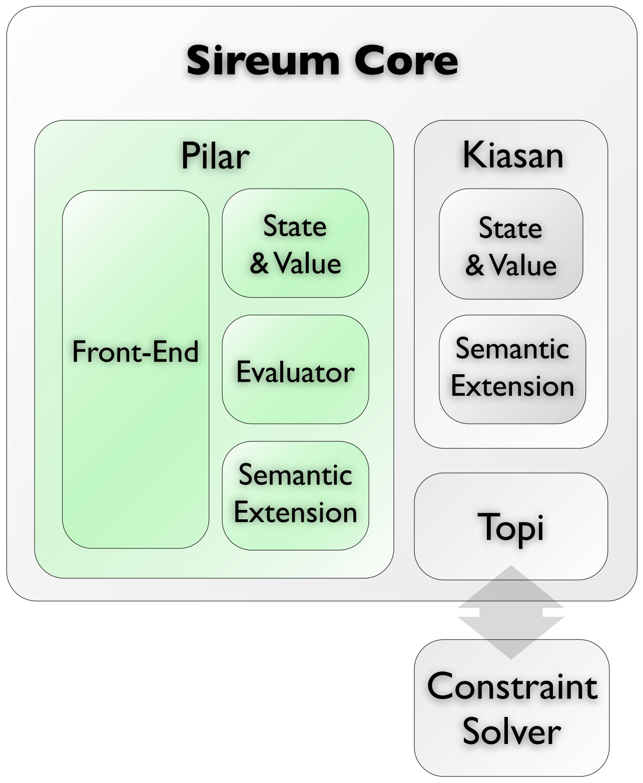


Pilar



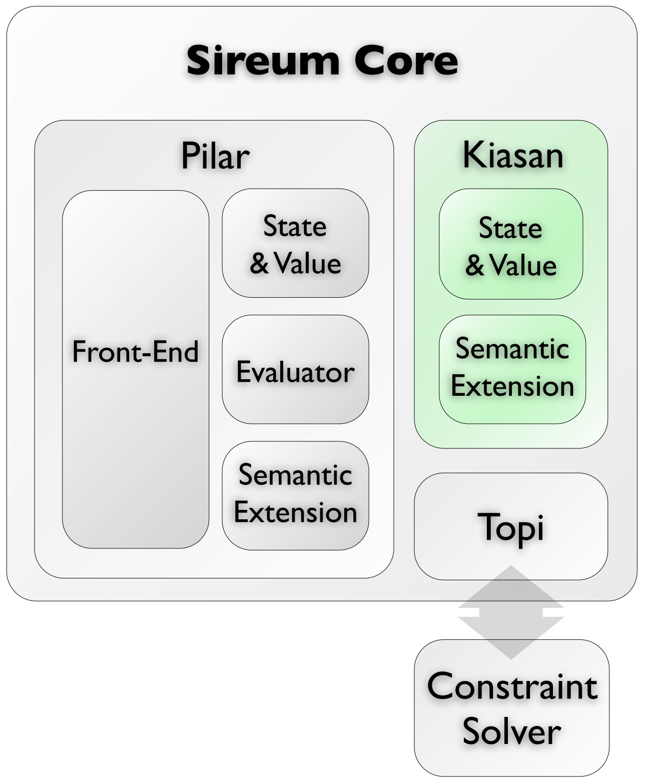
- ... is Sireum's intermediate representation (IR)
- rich syntactic language features
    - objects, exceptions, threads, etc.
- no predefined semantics
    - types, state/value, interpretations
- customize: create a profile!

Pilar



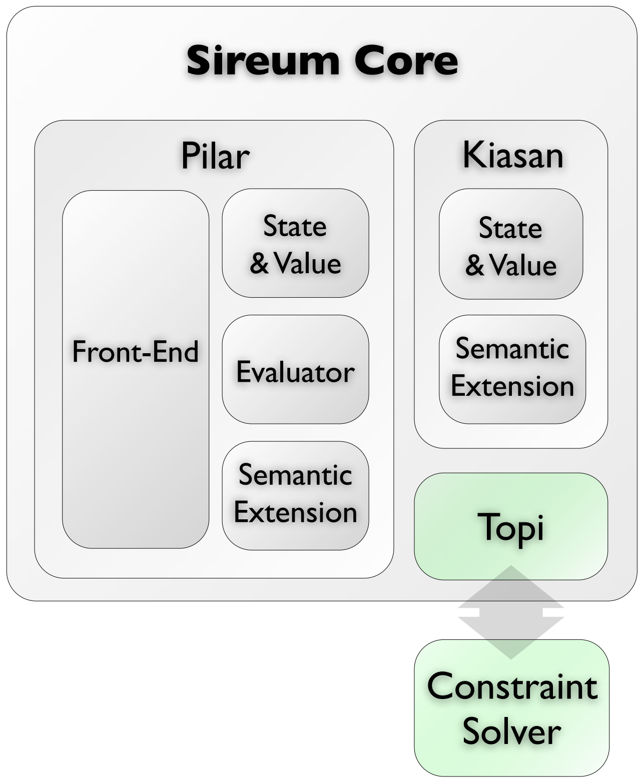
- provides basic building blocks for customization
  - extensible state and value
  - pluggable type system
  - composable evaluators
  - composable semantics via extension mechanism

Kiasan



- provides basic building blocks for building SymExe engine
  - refines Pilar state and value, but still customizable
  - refines Pilar extension mechanism for SymExe

Topi & Constraint Solver



- provides a generic interface to constraint solver
    - SMT: Z3, Yices, etc.
- provides a Lightweight Decision Procedure (LDP) for optimizations
    - linear space and time

Implementation Language

- Sireum (v2): Scala (+ Java)
  - provide more high-level language features
  - static typing with powerful type inference
  - natural to implement operational semantics, analyzer, transformer, etc.
  - leverage Java libraries and JVM
  - concurrency: collection, actor, Akka, etc.
  - IDE support, etc.

Implementation Guidelines

- Scala: a hybrid functional and OOP language
- ... stateless computation components (e.g., eval)
    - pass context/configuration and transform (e.g., SymExe and analyzer state)
    - easy to parallelize and distribute
    - some guarantee by Scala's type system
- ... use imperative features locally or judiciously whenever more convenient

Right After the Break

- Walkthrough on building SymExe engine using Sireum/Kiasan
    - SymExe semantic domains and operational semantics
    - how they are realized as Pilar and Kiasan extensions
- So, get Sireum and workspace ready
    - switch your workspace to Kiasan
    - clean all projects if you have compile errors
    - Run MyInt* JUnit test cases (should be green)