# AJACS : Applying Java to Automotive Control Systems

**Authors :**

**Jérôme Charousset, Antonio Kung. www.trialog.com**

**Thilo Gaul /IPD/U.Karlsruhe. i44www.info.uni-karlsruhe.de**

**Presented by Antonio Kung**


**Embedded Intelligence 2001**
**Nürnberg. February, 15th 2001**

# Content

- Context of Automotive Electronics
- AJACS objectives
- AJACS technical requirements and issues
- J consortium HIPA specification
- Native code approach
- Timetable

# Automotive Market

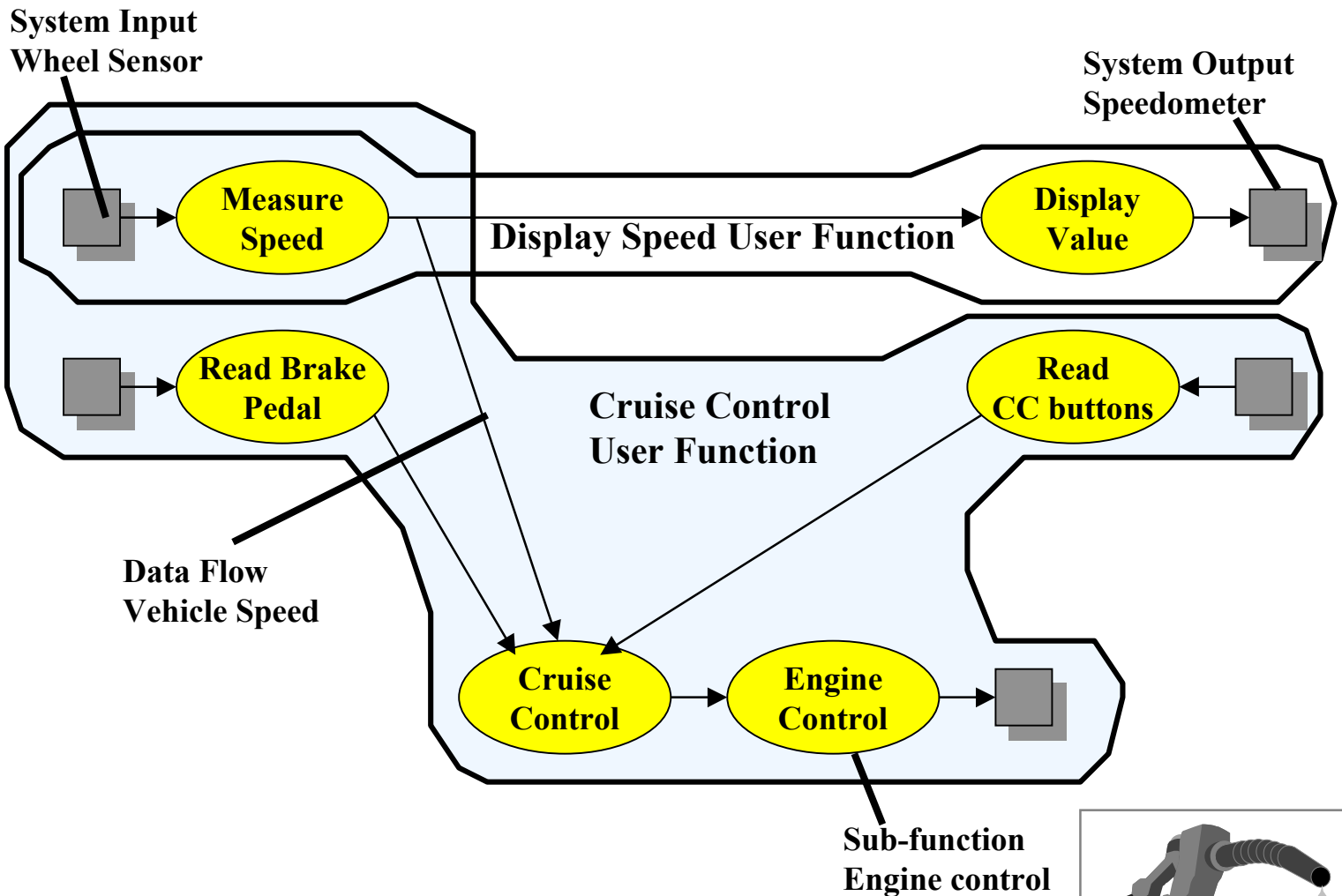
- Increasing number of electronics
  - $240 in a vehicle by 2001
  - $4.9 billion for DSP, microcontrollers, microprocessors
- Fragmented market (4 bit to 128 bits)
  - 8-16-32 bits for control
  - 32 bits+ for infotainment

# Two Worlds

- Infotainment (Navigation, Internet, Telecom).
  - e.g. AMIC initiative (www.ami-c.com) on Java-based technology
- Control bus (powertrain, ABS, engine control ...)
  - e.g. OSEK/VDX initiative (www.osek-vdx.org) on RTOS and multiplexing
  - e.g. LIN Local Interconnect Network announcement
    - Audi, BMW, DaimlerChrysler, Volvo, VW)

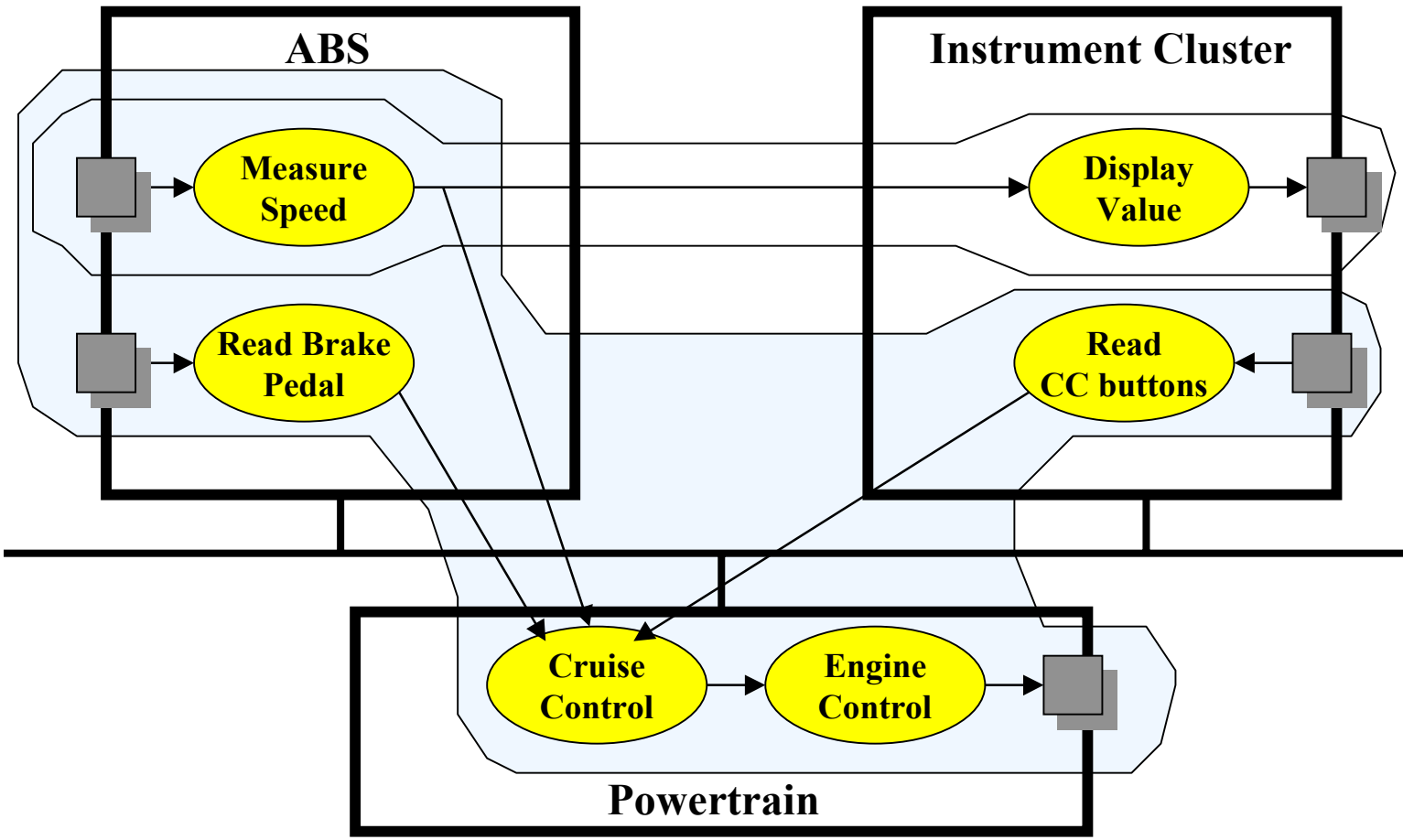
# Automotive Industry

- **More global functions**
  - Multiplexing (CAN)
  - Interconnectivity with vehicles
- **Different Car manufacturer/OEM relationship**
  - Car manufacturer define overall system and retain know-how
    - Car manufacturer provide application
    - OEM provide incomplete Electronic Control Units (ECU)
    - OEM provide software components


System Input
Wheel Sensor
System Output
Speedometer
Measure
Speed
Display Speed User Function
Display
Value
Read Brake
Pedal
Cruise Control
User Function
Read
CC buttons
Data Flow
Vehicle Speed
Cruise
Control
Engine
Control
Sub-function
Engine control



AJACS

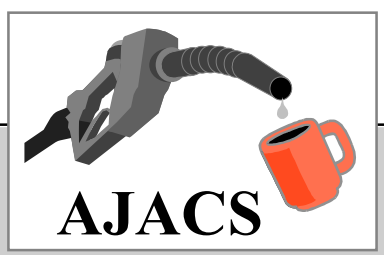

ABS
Instrument Cluster
Measure Speed
Display Value
Read Brake Pedal
Read CC buttons
Cruise Control
Engine Control
Powertrain



AJACS

# Demands on Technology and Tools

- Open systems (e.g. OSEK/VDX, AMIC)
- Advanced methods and tools (e.g. OMT, UML)
- Support for dependability in some cases (e.g. TTP)
- Hardware independence
  - e.g. A provides application, B and C provide ECU hardware
- Need for single chip approach
  - 8-16-32 bits
  - Small memory footprints (128 Kbytes ROM 10Kbyte RAM).

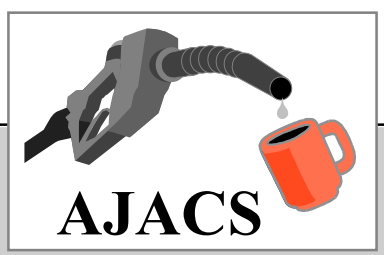


# AJACS

- 2-year Initiative
- Consortium
  - Trialog
  - PSA (Peugeot-Citroën)
  - Centro Riserche Fiat
  - Mecel (technology centre of Delphi)
  - University of Karslruhe

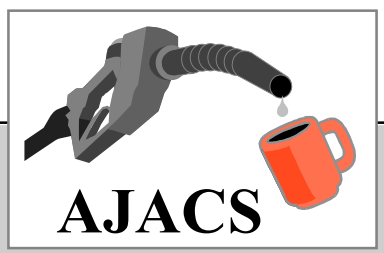
# AJACS Objectives

- **Specification, Development, Demonstration of**
  - an open technology
  - based on Java
  - for deeply embedded automotive control systems
- **Industrial viewpoint**
  - Benefit from object orientation in terms of structuring, reusability, dependability
  - WORA attributes to some extent, robustness attributes
  - Support the same kind of real-time constraints which non Java based ECUs are managing today
  - Single chip approach - Small footprint



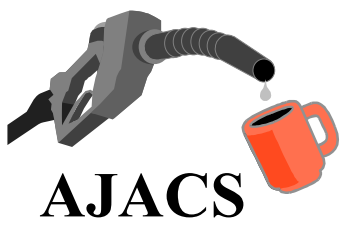
# Technical Requirements

- **Mechanisms and APIs must**
  - support existing standards in the automotive industry (OSEK/VDX)
  - support legacy C code
  - support calibration mechanisms
  - support distribution mechanisms
- **Run-time must have right level of performance. Native code**
- **Issues related to Java**
  - e.g. Memory management, synchronisation, interrupt, ..
  - static versus dynamic



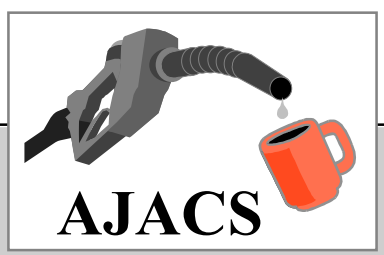
# Static VS Dynamic

- Static systems
  - static predetermined configuration (e.g. task 3)
- … are easier for determinism
- … allows for small footprints
- Example of Threads
  - can only be created at initialization time?
  - Association between Java entity and underlying static entity
- Example of Memory management
  - is GC needed?
  - immortal memory? Scoped memory?



# OSEK/VDX

- Standard architecture for distributed control units in vehicles
- Specifies abstract APIs
  - real-time operating system — OS
  - communication — COM
  - network management — NM
  - system generation — OIL
- Static system



# OSEK/OS supports for

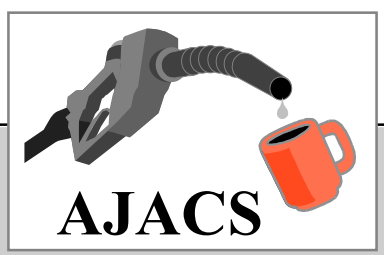
- Tasks
  - Basic tasks - no waiting
  - Extended tasks
- Resource
  - Priority ceiling protocol
  - No waiting
- Events
- Alarms and counters

# OIL (OSEK Implementation Language)

- Entities are described in OIL (e.g. task)
- Run-time entity descriptors (e.g. task descriptor) contains (typically)
  - ROM part
  - RAM part
- OIL builder generates configuration info
  - e.g. constants in ROM
  - e.g. initialization code ...

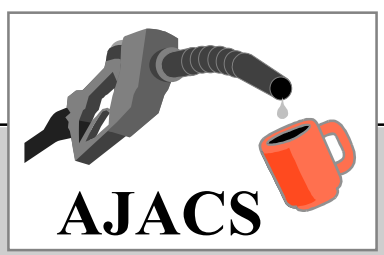
# Issues

- Combine OSEK/VDX execution model with Java execution model
- Combine OIL with Java
  - Entities described in OIL
  - Builder generate structure

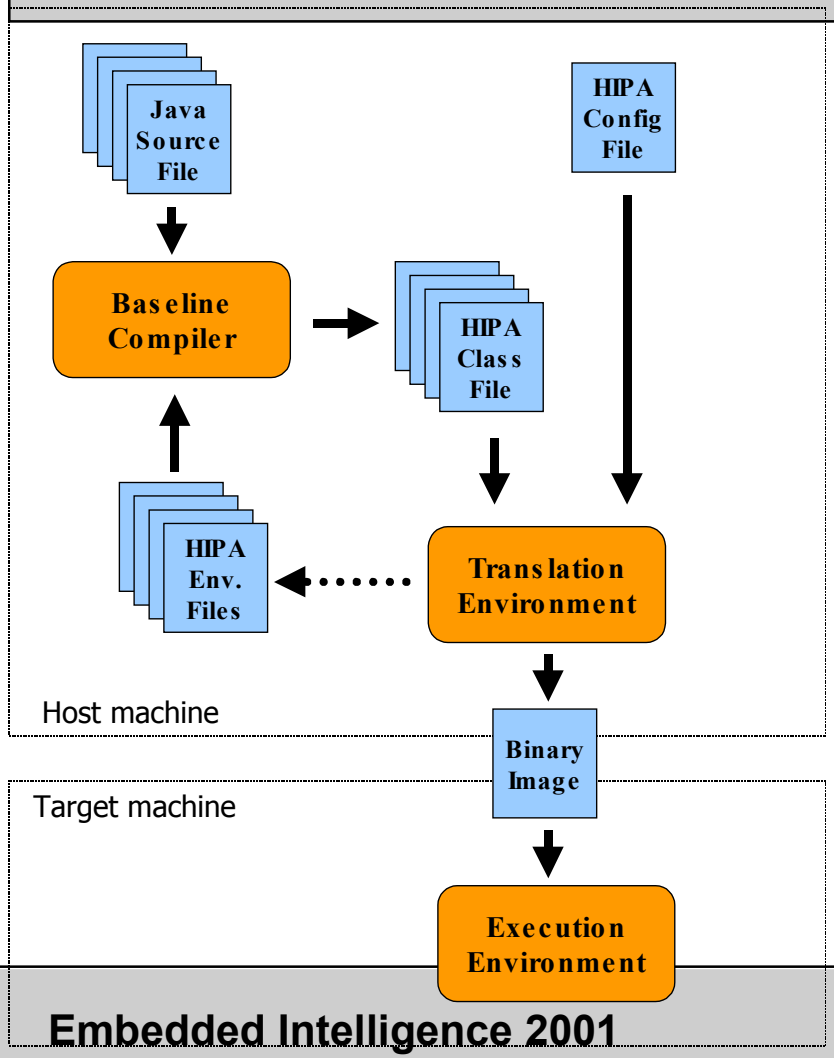


# Standardisation

- Carried out within the J consortium (www.j-consortium.org)
- within the HIP (High Integrity Profile) working group
- defines a specific profile, the HIPA Specification (High Integrity Profile for Automotive applications)



# A HIPA Compliant Implementation




- Application files
- Configuration files
  - counterpart of OIL file
- … create Environment files
- … generate class files
- … generate binary image



AJACS

# A HIPA Compliant Implementation

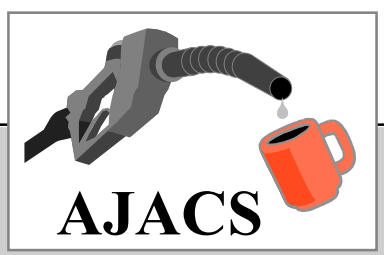
- **API for OSEK**
  - configuration API
  - task management
  - interrupt management
  - event management
  - synchronisation API
- **Conformance**
  - run-time checking as an option
  - OSEK/VDX classes of conformity
    - BCC1, BCC2, ECC1, ECC2

# Native Code Approach vs Interpretation

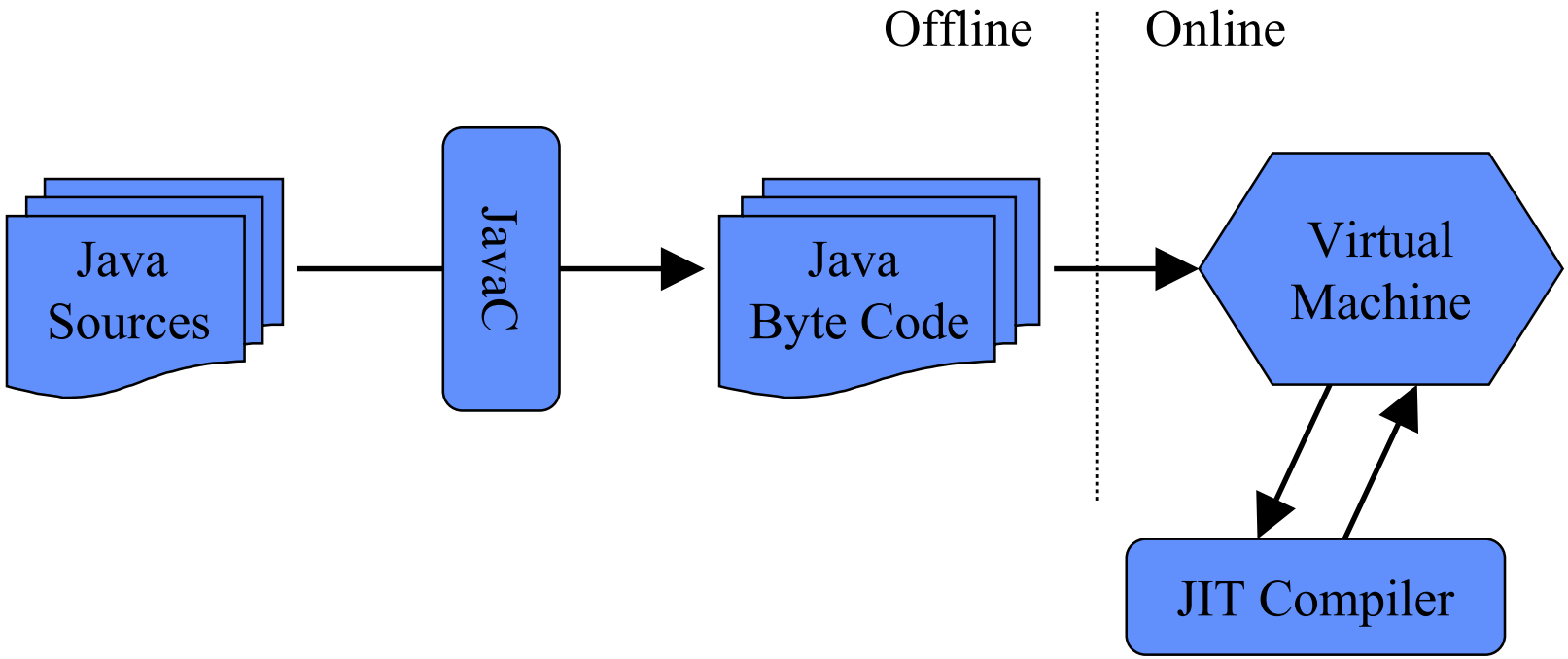
- **Standard approach : Interpretation of Byte-Code**
  - high-level Byte-Code pre-compiled from Java sources
  - virtual machine / interpreter runs the program
  - whole (virtual) state space available to inspection/debugging
  - exchangeable code pieces (dynamic class loading)
- **Partial Compilation: JIT Compiler**
  - Parts (methods/expressions) are compiled to native code
  - Compiler included in Virtual Machine
- **Full Compilation: Offline Compiler**
  - Full native binary

# Native Code Approach vs Interpretation

- Giving up interpretation we lose:
  - Dynamic overloading of classes
  - Compile once, run everywhere
  - Runtime verifier
- We gain:
  - Improved execution speed by orders
  - Better static memory layout, less garbage collection
- We keep:
  - Replacement of software modules
  - Inspection/Debugging interface
  - Write once, compile to many platforms



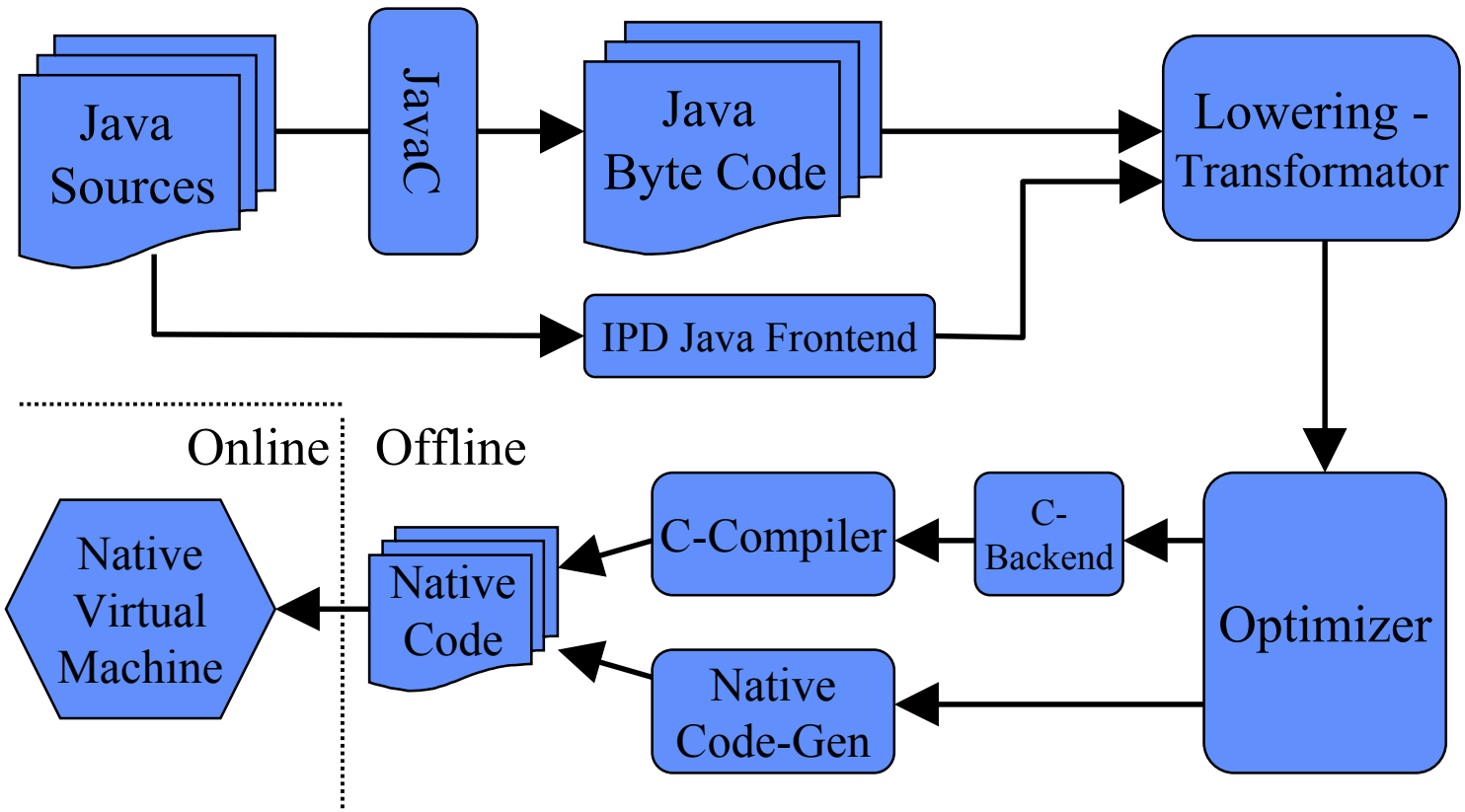
# Native Code VS Interpretation






AJACS

# AJACS Native Code Approach





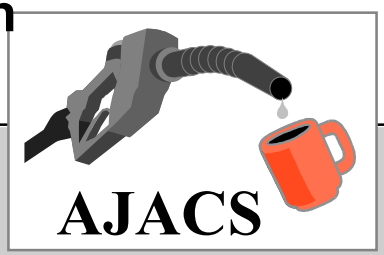

AJACS

# AJACS Native Code Approach

- **Lowering Transformator**
  - high level Java construct transformed into low level intermediate form
  - … called SSA (Single Static Assigment)
- **Optimizer**
  - works on intermediate form

# Optimisation Technology

- **Object orientation**
  - expensive polymorphic calls.
  - many calls to procedures (e.g. 5 times more).
  - frequent accesses to heap variables (e.g. 60% more memory access)
  - lots of heap objects allocation
- **AJACS will use Explicit Dependency Graphs (EDG)**
  - optimization = rewriting of graph
- **and BEG (Back End Generator) tool (U.Karlsruhe)**
  - Bottom-up-rewrite/bottom-up-pattern-match



# AJACS time table

- Draft spec March 2001
- Implementation August 2001