Smart Cards

Towards a modern run-time platform

2. Software & Its Interplay

Thorsten Kramp & Michael Kuyper
IBM Zurich Research Laboratory

"Those parts of the system that you can hit with a hammer (not advised) are called hardware; those program instructions that you can only curse at are called software."

Anonymous

Overview

A. Basic machinery

execution model, byte code vs. native code, language aspects

B. Memory management

basic schemes, memory types: transient vs. persistent, garbage collection

C. Atomicity and transactions

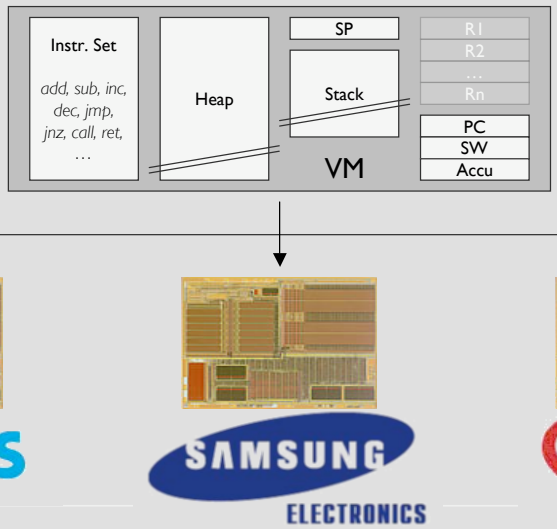
basic schemes, system-level vs. user-level transactions

D. OO programming w/ resource constraints

applet design, RMI, size/performance optimizations

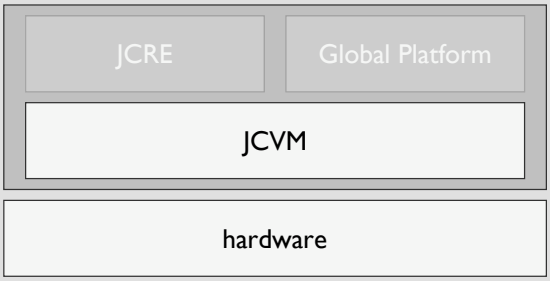
A. Basic Machinery: VM

- Virtual machine
  - *an abstract machine w/ its own instruction set, registers, memory model …*
  - *programs written against an VM instruction set become independent from 'real hardware' (a.k.a. "write once, run everywhere")*



A. Basic Machinery: VM

- Virtual machine
  - *an abstract machine w/ its own instruction set, registers, memory model …*
  - *programs written against an VM instruction set become independent from 'real hardware' (a.k.a. "write once, run everywhere")*
- JavaCard VM
  - *interprets Java 'byte code'*
  - *subset of the Java desktop VM*



A. Basic Machinery: VM

- Virtual machine
  - *an abstract machine w/ its own instruction set, registers, memory model &c.*
  - *programs written against an VM instruction set become independent from 'real hardware' (a.k.a. "write once, run everywhere")*
- JavaCard VM
  - *interprets Java 'byte code'*
  - *subset of the Java desktop VM*

Supported Java features

1. *small primitive data types:* `boolean`, `byte`, `short`
2. *one-dimensional arrays*
3. *packages, classes, interfaces, and exceptions*
4. *object-oriented features: inheritance, virtual methods, overloading and dynamic object creation,*
5. *access scope, and binding rules*
6. *garbage collection (since JC 2.2)*
7. *optional:* `int`

A. Basic Machinery: VM

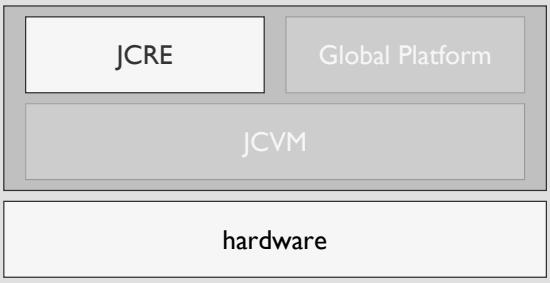
- Virtual machine
  - *an abstract machine w/ its own instruction set, registers, memory model &c.*
  - *programs written against an VM instruction set become independent from 'real hardware' (a.k.a. "write once, run everywhere")*
- JavaCard VM
  - *interprets Java 'byte code'*
  - *subset of the Java desktop VM*

Unsupported Java features

1. *large primitive data types:* `long`, `double`, `float`
2. *characters and strings*
3. *multi-dimensional arrays*
4. *dynamic class loading*
5. *security manager*
6. *finalization (and garbage collection prior to JC 2.2)*
7. *object serialization and cloning*
8. *threads*

A. Basic Machinery: JCRE

- Run-time environment (JCRE)
  - *life time*
    - initialized at card initialization time (only once)
    - after each reset, JCRE enters "receive-process-reply" loop
    - applets and persistent data are preserved over resets
  - *responsible for:*
    - card resource management
    - network communication
    - applet execution
    - system and applet security
  - *defines the JavaCard API*



A. Basic Machinery: JCRE

- Run-time environment (JCRE)
  - *life time*
    - initialized at card initialization time (only once)
    - after each reset, JCRE enters "receive-process-reply" loop
    - applets and persistent data are preserved over resets
  - *responsible for:*
    - card resource management
    - network communication
    - applet execution
    - system and applet security
  - *defines the JavaCard API*

Additional **JavaCard features**

1. *persistent and transient objects; persistent is default* [discussed in 2.B]
2. *atomic operations and transactions* [disucssed in 2.C]
3. *applet firewall and sharing mechanisms* [discussed in 3.A]

A. Basic Machinery: JCRE

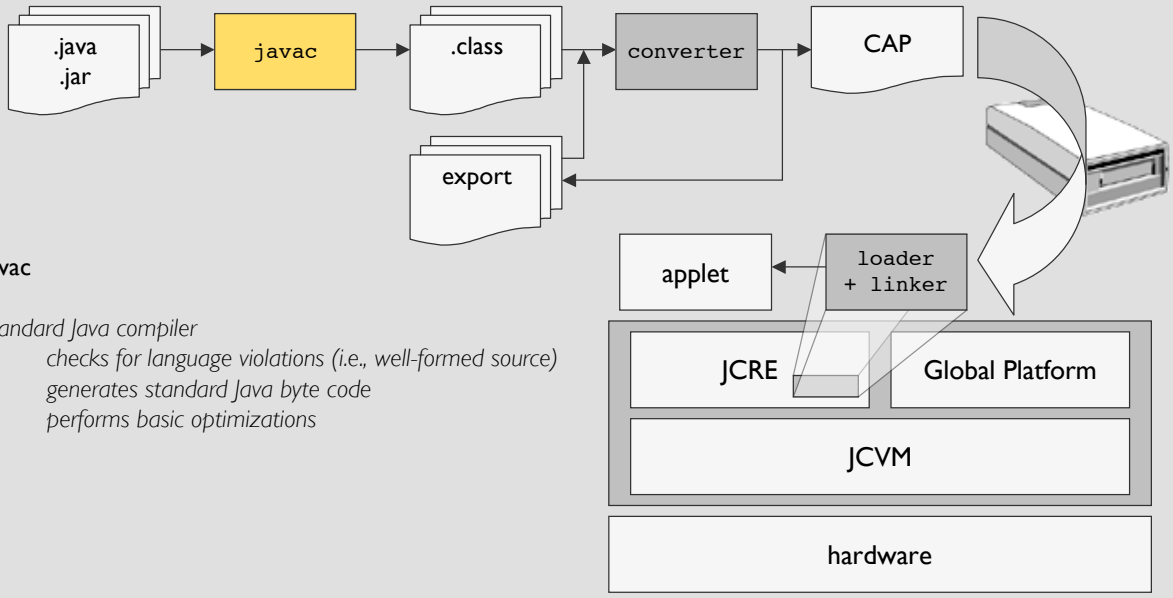
- Run-time environment (JCRE)
  - *life time*
    - initialized at card initialization time (only once)
    - after each reset, JCRE enters "rec
    - applets and persistent data are pr
  - *responsible for:*
    - card resource management
    - network communication
    - applet execution
    - system and applet security
  - *defines the JavaCard API*

API packages overview

`java.lang` *(strict subset of Java* `java.lang`*)*
e.g., `Object`, `Throwable`
`javacard.framework` *(core functionality)*
e.g., `Applet`, `APDU`, `JCSystem`
`javacard.rmi`*(remote method invocation)*
e.g., `Remote` [JC 2.2]
`javacard.framework.service` *(service components)*
e.g., `RMIService`, `SecurityService` [JC 2.2]
`javacard.security` *(crypto functions)*
e.g., `Key`, `Signature`, `MessageDigest`
`javacardx.crypto` *(US export-controlled crypto)*
e.g., `Cipher`

A. Basic Machinery: Off- & On-Card

- From the source onto the card



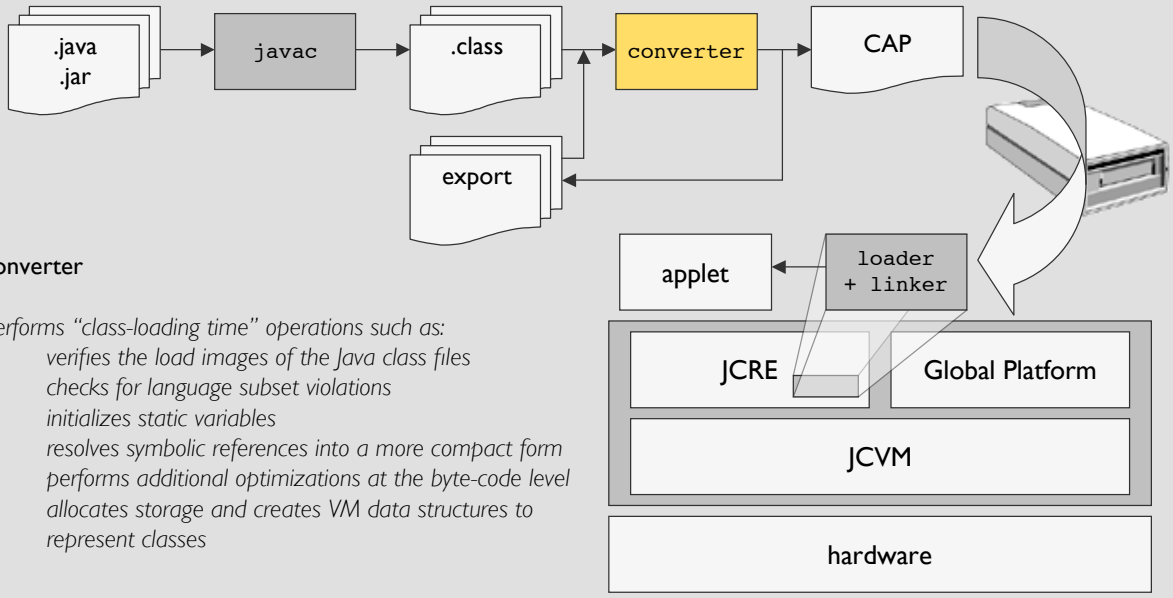
javac

standard Java compiler

1. *checks for language violations (i.e., well-formed source)*
2. *generates standard Java byte code*
3. *performs basic optimizations*

A. Basic Machinery: Off- & On-Card

- From the source onto the card



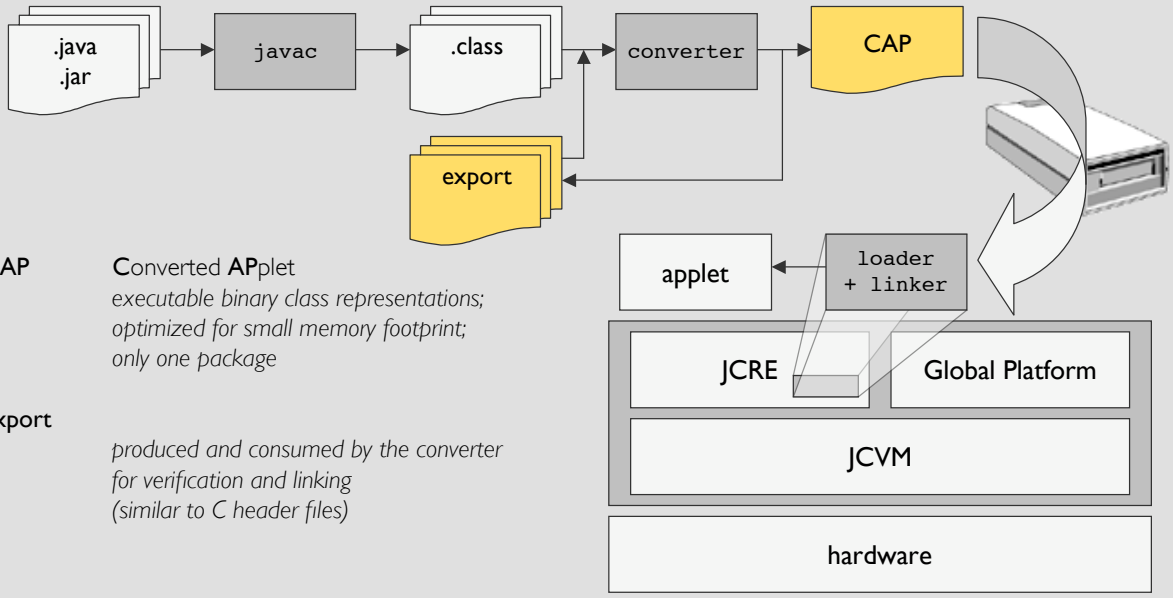
converter

performs "class-loading time" operations such as:

1. *verifies the load images of the Java class files*
2. *checks for language subset violations*
3. *initializes static variables*
4. *resolves symbolic references into a more compact form*
5. *performs additional optimizations at the byte-code level*
6. *allocates storage and creates VM data structures to represent classes*

A. Basic Machinery: Off- & On-Card

- From the source onto the card

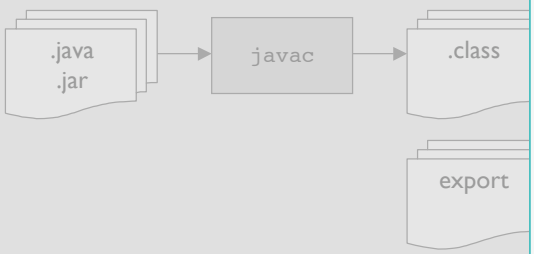


CAP — **C**onverted **AP**plet
executable binary class representations; optimized for small memory footprint; only one package

export
produced and consumed by the converter for verification and linking (similar to C header files)

A. Basic Machinery: Off- & On-Card

- From the source onto the card



CAP — **C**onverted **AP**plet
executable binary class representations; optimized for small memory footprint; only one package

export
produced and consumed by the converter for verification and linking (similar to C header files)

CAP file format

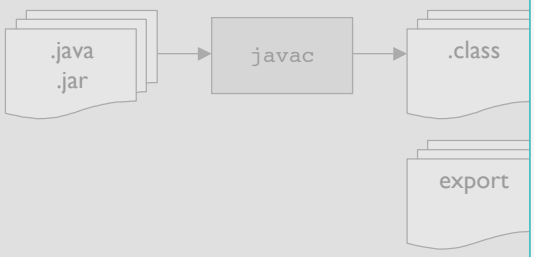
JAR file w/ components:

```
component {
    u1 tag;
    u2 size;
    u1 info[];
}
```

1. **header:** *general information about the CAP file and its package (e.g., CAP format version, package AID,* `int`*).*
2. **directory:** *lists the size of each CAP file component; optional components have size 0.*
3. **applet:** [optional] *one entry for each applet in this package describing, for instance, the applet AID.*
4. **import:** *lists the set of packages imported by classes in this package.*

A. Basic Machinery: Off- & On-Card

- From the source onto the card



CAP Converted **AP**plet
executable binary class representations; optimized for small memory footprint; only one package

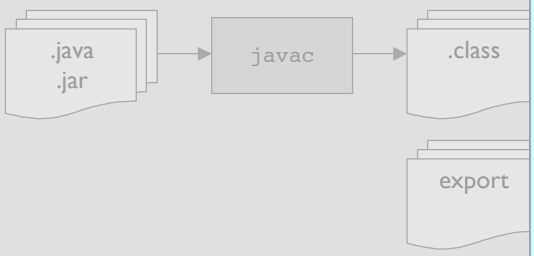
export
produced and consumed by the converter for verification and linking (similar to C header files)

CAP file format (cont'd)

5. **constant pool:** *contains an entry for each of the classes, methods, and fields referenced by elements in the **method** component (i.e., it references elements in the **class**, **method**, and **static field** components).*
6. **class:** *describes each of the classes and interfaces defined in this package; contains access information only as required for execution, not verification.*
7. **method:** *describes each of the methods defined in this package excluding* `clinit` *and interface declarations (abstract methods of classes are included), as well as the exceptions associated with each method.*
8. **static field:** *contains all information to create and initialize the static field image of this package; primitive* `final static` *fields are not included.*

A. Basic Machinery: Off- & On-Card

- From the source onto the card



CAP Converted **AP**plet
executable binary class representations; optimized for small memory footprint; only one package

export
produced and consumed by the converter for verification and linking (similar to C header files)

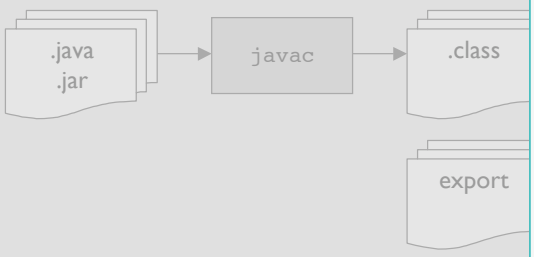
CAP file format (cont'd)

9. **reference location:** *lists of offsets into the **method** component to items containing indices into the **constant pool** component.*
10. **export:** [optional] *lists all static elements in this package that may be imported by classes in other packages; no instance fields or virtual methods.*
11. **descriptor:** *provides information to parse and verify all elements of the CAP file; references elements in the **contanst pool**, **class**, **method**, and **static field** components.*
12. **debug:** [optional] *contains all meta-data for debugging this package.*

[Virtual Machine Specification, JavaCard Platform 2.2.1]

A. Basic Machinery: Off- & On-Card

- From the source onto the card



CAP Converted **AP**plet
executable binary class representations; optimized for small memory footprint; only one package

export
produced and consumed by the converter for verification and linking (similar to C header files)

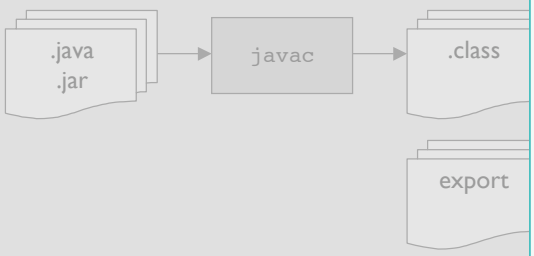
Export file format

```
exportfile {
  u4 magic;
  u1 minor_version;
  u1 major_version;
  u2 constant_pool_count;
  cp_info constant_pool[constant_pool_count];
  u2 this_package;
  u1 export_class_count;
  class_info classes[export_class_count];
}
```

1. **name:** *last portion of the package name* + `.exp` *(e.g.,* `framework.exp`*)*
2. **version:** *different major version number indicates fundamental incompatibility change*

A. Basic Machinery: Off- & On-Card

- From the source onto the card



CAP Converted **AP**plet
executable binary class representations; optimized for small memory footprint; only one package

export
produced and consumed by the converter for verification and linking (similar to C header files)

Export file format

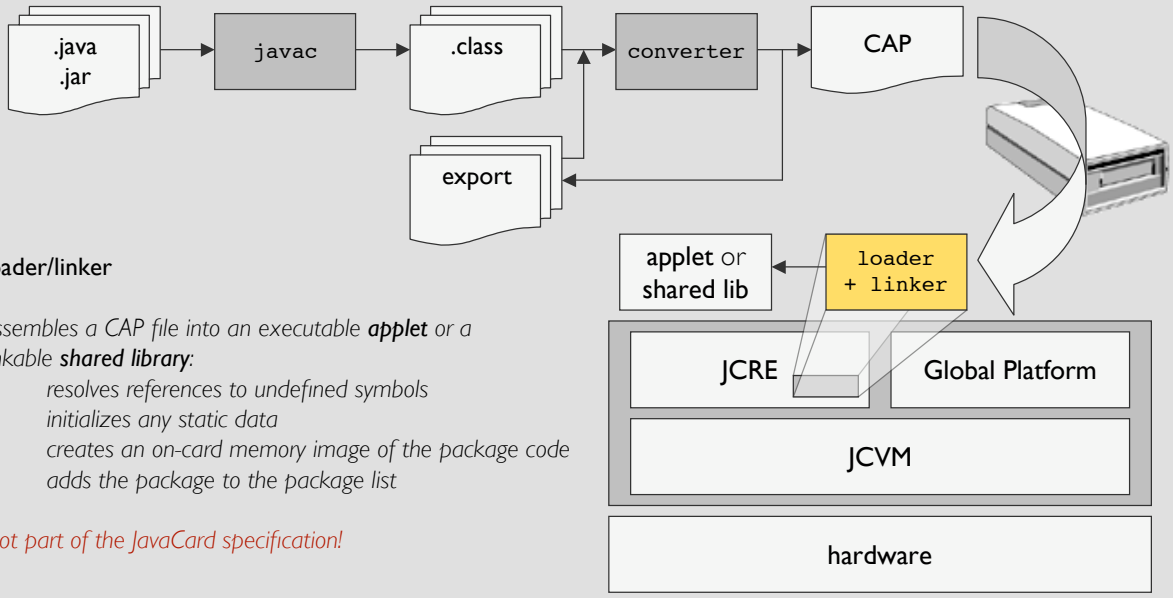
3. **constant pool:** *table of variable-length structures representing string constants, class names, field names, and other constants referred to within the export file.*
4. **this package:** *index to the* ***constant pool*** *element describing the package defined by this export file.*
5. **classes:** *table of variable-length structures describing publicly accessible classes or interfaces declared in the packaged described by this export file.*

Note: *Classes and interfaces represented in an export file include all public elements defined within their respective hierarchies (i.e., all superclasses, superinterfaces, and virtual methods are explicitly listed).*

[Virtual Machine Specification, JavaCard Platform 2.2.1]

A. Basic Machinery: Off- & On-Card

- From the source onto the card



loader/linker

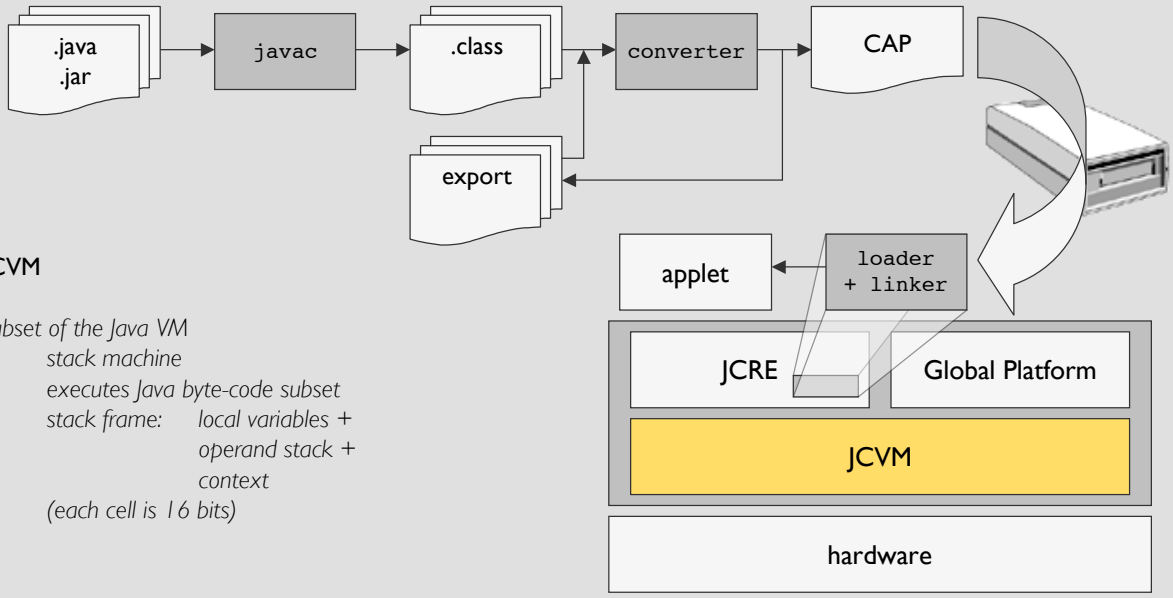
*assembles a CAP file into an executable **applet** or a linkable **shared library**:*

1. *resolves references to undefined symbols*
2. *initializes any static data*
3. *creates an on-card memory image of the package code*
4. *adds the package to the package list*

Not part of the JavaCard specification!

A. Basic Machinery: Off- & On-Card

- From the source onto the card



JCVM

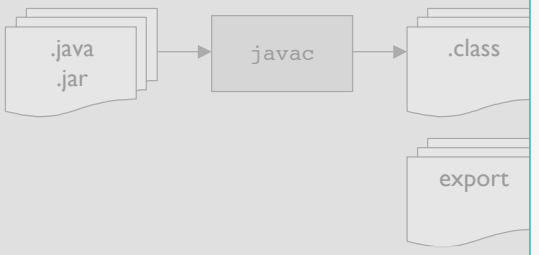
subset of the Java VM

1. *stack machine*
2. *executes Java byte-code subset*
3. *stack frame: local variables + operand stack + context*

(each cell is 16 bits)

A. Basic Machinery: Off- & On-Card

- From the source onto the card



JCVM

subset of the Java VM
1. *stack machine*
2. *executes Java by*
3. *stack frame:*

(each cell is 16 b

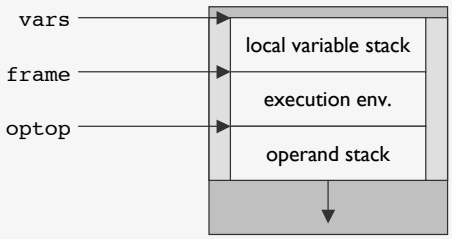
local variable stack

array of words accessed via indices;
method parameters and local variables;
`byte`s, `char`s *converted to* `short`s;
`int`s *use two adjacent slots*

Stack machine

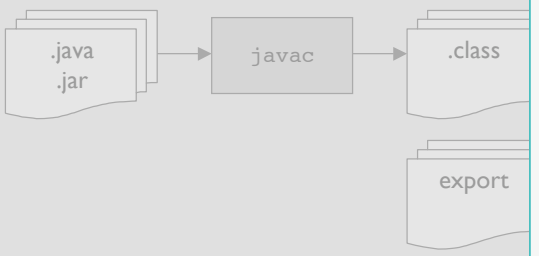
No registers to hold intermediate values but uses the stack for storage.

1. *small instruction set*
2. *allows implementation on architectures with few or irregular general purpose registers*
3. *allows code optimizations at run time (e.g., by just-in-time compilers)*



A. Basic Machinery: Off- & On-Card

- From the source onto the card



JCVM

subset of the Java VM
1. *stack machine*
2. *executes Java by*
3. *stack frame:*

(each cell is 16 b

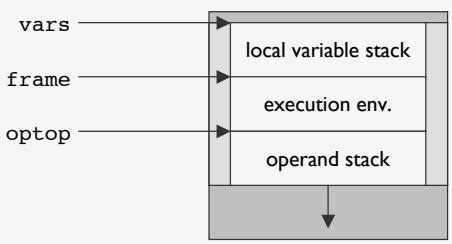
execution environment

constant pool resolution;
normal method return;
exception dispatch;
impl.-dependent information

Stack machine

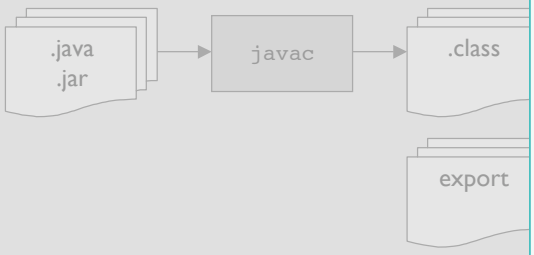
No registers to hold intermediate values but uses the stack for storage.

1. *small instruction set*
2. *allows implementation on architectures with few or irregular general purpose registers*
3. *allows code optimizations at run time (e.g., by just-in-time compilers)*



A. Basic Machinery: Off- & On-Card

- From the source onto the card



JCVM

subset of the Java VM
1. *stack machine*
2. *executes Java by*
3. *stack frame:*

(each cell is 16 b

operand stack

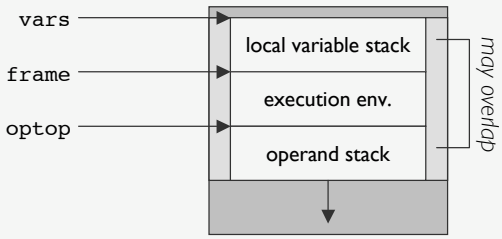
array of words accessed by push/pop work space for operations

example: `sadd`
1. *pop two shorts* `s1`, `s2`
2. *add* `s1` *and* `s2`
3. *push result*

Stack machine

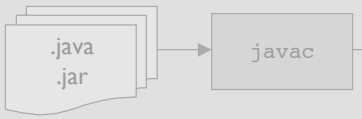
No registers to hold intermediate values but uses the stack for storage.

1. *small instruction set*
2. *allows implementation on architectures with few or irregular general purpose registers*
3. *allows code optimizations at run time (e.g., by just-in-time compilers)*



A. Basic Machinery: Off- & On-Card

- From the source onto the card



JCVM

subset of the Java VM
1. *stack machine*
2. *executes Java byte-code subset*
3. *stack frame: local variables + operand stack + context*

(each cell is 16 bits)

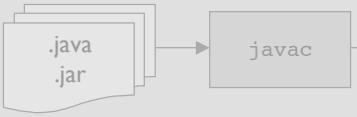
Supported bytes codes:

`nop`	`aaload`	`aastore`	`aconst_null`
`aload`	`aload_<n>`	`anewarray`	`areturn`
`arraylength`	`astore`	`astore_<n>`	`athrow`
`baload`	`bastore`	`bspush`	`dup`
`dup_x`	`dup2`	`goto`	`goto_w`
`new`	`newarray`	`pop`	`pop2`
`jsr`	`ret`	`return`	`sreturn`
`sload`	`sload_<n>`	`sstore`	`sstore<n>`
`saload`	`sastore`	`sconst_<s>`	`sspush`
`smul`	`sdiv`	`sinc`	`sinc_w`
`s2b`	`sneg`	`sor`	`srem`
`sadd`	`ssub`	`sand`	`swap_x`
`sshl`	`sshr`	`sushr`	`sxor`
`getfield_<t>_this`	`getfield_<t>`	`getfield_<t>_w`	`getstatic_<t>`
`putfield_<t>_this`	`putfield_<t>`	`putfield_<t>_w`	`putstatic_<t>`
`slookupswitch`	`stableswitch`		
`invokeinterface`	`invokespecial`	`invokestatic`	`invokevirtual`
`if_acmp<cond>`	`if_acmp<cond>_w`	`if<cond>`	`if<cond>_w`
`if_scmp<cond>`	`if_scmp<cond>_w`	`ifnull`	`ifnull_w`
`ifnonnull`	`ifnonnull_w`	`instanceof`	

[Virtual Machine Specification, JavaCard Platform 2.2.1]

A. Basic Machinery: Off- & On-Card

- From the source onto the card



JCVM

subset of the Java VM
1. *stack machine*
2. *executes Java byte-code subset*
3. *stack frame: local variables + operand stack + context*

(each cell is 16 bits)

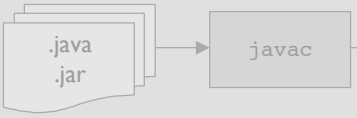
Unsupported bytes codes:

`lconst_<l>`	`fconst_<f>`	`dconst_<d>`	`ldc2_w2`
`lload`	`fload`	`dload`	`lload_<n>`
`fload_<n>`	`dload_<n>`	`laload`	`faload`
`daload`	`caload`	`lstore`	`fstore`
`dstore`	`lstore_<n>`	`fstore_<n>`	`dstore_<n>`
`lastore`	`fastore`	`dastore`	`castore`
`ladd`	`fadd`	`dadd`	`lsub`
`fsub`	`dsub`	`lmul`	`fmul`
`dmul`	`ldiv`	`fdiv`	`ddiv`
`lrem`	`frem`	`drem`	`lneg`
`fneg`	`dneg`	`lshl`	`lshr`
`lushr`	`land`	`lor`	`lxor`
`i2l`	`i2f`	`i2d`	`l2i`
`l2f`	`l2d`	`f2i`	`f2d`
`d2i`	`d2l`	`d2f`	`i2c`
`lcmp`	`fcmpl`	`fcmpg`	`dcmpl`
`dcmpg`	`lreturn`	`freturn`	`dreturn`
`goto_w`	`jsr_w`	`multianewarray`	
`monitorexit`	`monitorenter`		

[Virtual Machine Specification, JavaCard Platform 2.2.1]

A. Basic Machinery: Off- & On-Card

- From the source onto the card



JCVM

subset of the Java VM
1. *stack machine*
2. *executes Java byte-code subset*
3. *stack frame: local variables + operand stack + context*

(each cell is 16 bits)

Optional bytes codes (`int` support)

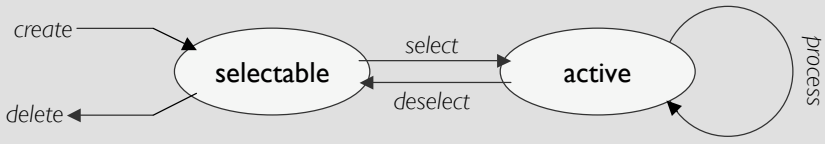
`iadd`	`isub`	`imul`	`idiv`
`irem`	`iinc`	`iinc_w`	`ireturn`
`iand`	`ineg`	`ior`	`ixor`
`icmp`	`ishl`	`ishr`	`iushr`
`i2b`	`i2s`	`s2i`	
`iload`	`iload_<n>`	`istore`	`istore_<n>`
`iaload`	`iastore`	`iconst_<i>`	
`sipush`	`bipush`	`iipush`	
`getfield_i_this`	`getfield_i`	`getfield_i_w`	`getstatic_i`
`putfield_i_this`	`putfield_i`	`putfield_i_w`	`putstatic_i`
`ilookupswitch`	`itableswitch`		

[Virtual Machine Specification, JavaCard Platform 2.2.1]

DEMO

A. Basic Machinery: Applet

- Definition
  - *a smart card application written in Java uniquely identified by an AID*
  - *instance of a class that extends* `javacard.framework.Applet`
  - *any number of applets may be installed [state: selectable]*
  - *only one applet is running at a time [state: active]*
- Applet life cycle
  - *applet's life starts when it is registered with the JCRE [state: selectable]*
  - *must be explicitly selected by the host [state: active]*
  - *purely reactive behaviour*



A. Basic Machinery: Applet Class

```
public abstract class Applet {
   public static void install(byte[] bArray, short bOffset, byte bLength);

   protected final void register();
   protected final void register(byte bArray, short bOffset, byte bLength);

   public boolean select();
   public void deselect();
   protected final boolean selectingApplet();

   public abstract void process(APDU apdu);
   ...
};
```

`public static void install(…)`

- *creates an instance of the applet subclass*
- *should perform any necessary initializations and must call one of the* `register` *methods*
- *installation is successful if the* `register` *methods does not throw an exception*
- *after sucessful installation the applet is selectable*

A. Basic Machinery: Applet Class

```
public abstract class Applet {
   public static void install(byte[] bArray, short bOffset, byte bLength);

   protected final void register();
   protected final void register(byte bArray, short bOffset, byte bLength);

   public boolean sele
   public void deselec
   protected final boo

   public abstract voi
   ...
};
```

```
bArray[bOffset] = length(Li) of instance AID
bArray[bOffset+1..bOffset+Li] = instance AID bytes (5-16 bytes)

bArray[bOffset+Li+1]= length(Lc) of control info
bArray[bOffset+Li+2..bOffset+Li+Lc+1] = control info

bArray[bOffset+Li+Lc+2] = length(La) of applet data
bArray[bOffset+Li+Lc+2..bOffset+Li+Lc+La+1] = applet data
```

`public static void install(…)`

- *creates an instance of the applet subclass*
- *should perform any necessary initializations and must call one of the* `register` *methods*
- *installation is successful if the* `register` *methods does not throw an exception*
- *after sucessful installation the applet is selectable*

A. Basic Machinery: Applet Class

```
public abstract class Applet {
   public static void install(byte[] bArray, short bOffset, byte bLength);

   protected final void register();
   protected final void register(byte bArray, short bOffset, byte bLength);

   public boolean select();
   public void deselect();
   protected final boolean selectingApplet();

   public abstract void process(APDU apdu);
   ...
};
```

`protected final void register(…)`

- *registers the new applet instance with the JCRE*
- *uses the AID specified in the CAP file (only one applet instance possible), or…*
- *…the AID passed in* `bArray` *(multiple instances possible)*

A. Basic Machinery: Applet Class

```
public abstract class Applet {
   public static void install(byte[] bArray, short bOffset, byte bLength);

   protected final void register();
   protected final void register(byte bArray, short bOffset, byte bLength);

   public boolean select();
   public void deselect();
   protected final boolean selectingApplet();

   public abstract void process(APDU apdu);
   ...
};
```

```
public class myApplet extends Applet {
   public static void install(byte[] bArray, short bOffset, byte bLength) {
      (new myApplet()).register(bArray,(short)(bOffset+1),bArray[bOffset]);
   }

   protected myApplet() { // constructor
      ...
   }
};
```

A. Basic Machinery: Applet Class

```
public abstract class Applet {
   public static void install(byte[] bArray, short bOffset, byte bLength);

   protected final void register();
   protected final void register(byte bArray, short bOffset, byte bLength);

   public boolean select();
   public void deselect();
   protected final boolean selectingApplet();

   public abstract void process(APDU apdu);
   ...
};
```

`public boolean select()`

- *called by the JCRE to inform the applet that it has been selected*
- ***default applet** is selected automatically on card reset*

`public boolean deselect()`

- *called by the JCRE to inform the applet that another (or the same) applet will be selected*

A. Basic Machinery: Applet Class

```
public abstract class Applet {
   public static void install(byte[] bArray, short bOffset, byte bLength);

   protected final void register();
   protected final void register(byte bArray, short bOffset, byte bLength);

   public boolean select();
   public void deselect();
   protected final boolean selectingApplet();

   public abstract void process(APDU apdu);
   ...
};
```

`public abstract void process(...)`

- *called by the JCRE to process an incoming APDU command*
- *upon normal return the JCRE sends the ISO 7816 defined success code* `0x9000` *in the APDU response*

`protected native final boolean selectingApplet()`

- *used by the* `process` *method to distinguish between applet selects from other* `SELECT` *APDU commands*

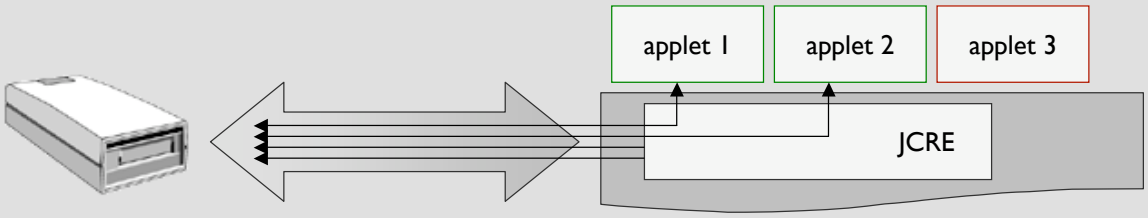
A. Basic Machinery: Applet Class

```
public abstract class Applet {
   public static void install(byte[] bArray, short bOffset, byte bLength);

   protected final void register();
   protected final void register(byte bArray, short bOffset, byte bLength);

   public boolean select();
   public void deselect();
   protected final boolean selectingApplet();

   public abstract void process(APDU apdu);
   ...
};
```

```
public class myApplet extends Applet {
   public void process(APDU apdu) {
      if (selectingApplet()) {
         ... // additional select processing
         return;
      }

      ... // process APDU
   }
};
```


A. Basic Machinery: Logical Channels

- logical channels allow up to four sessions into the smart card
- only logical channel 0 is active on card reset
- specified in ISO 7816-4, introduced in JavaCard 2.2
- one default applet per logical channel
- multi-selectable applets
  - *implement* `javacard.framework.MultiSelectable`
  - *all or none applets within a package shall be multi-selectable*



A. Basic Machinery: Interface Multiselectable

```
public abstract class Applet {
   public static void install(byte[] bArray, short bOffset, byte bLength);

   protected final void register();
   protected final void register(byte bArray, short bOffset, byte bLength);

   public boolean select();
   public void deselect();
   protected final boolean selectingApplet();

   public abstract void process(APDU apdu);
   ...
};
```

```
public interface MultiSelectable {
   public void select(boolean appInstAlreadyActive);
   public void deselect(boolean appInstStillActive);
};
```


A. Basic Machinery: `javacard.framework`

- **`javacard.framework.AID`**
  - *encapsulates the Application IDentifier associated with an applet*
  - *created by the JCRE only*
- **`javacard.framework.APDU`**
  - *encapsulates an Application Protocol Data Unit according to ISO 7816*
  - *singleton object owned by the JCRE*
  - *zeroed out by the JCRE before each new message received*
- **`javacard.framework.Util`**
  - *common static utility functions*
- **`javacard.framework.JCSystem`**
  - *collection of methods to control applet execution, memory management [2.B], atomic transaction management [2.C], inter-applet object sharing [3.A]*

A. Basic Machinery: `javacard.framework`

- `javacard.framework.AID`
  - *encapsulates the Application IDentifier associated with an applet*
  - *created by the JCRE only*
- `javacard.framework.`
  - *encapsulates an Applicati*
  - *singleton object owned by*
  - *zeroed out by the JCRE be*
- `javacard.framework.`
  - *common static utility funct*
- `javacard.framework.`
  - *collection of methods to c*
    atomic transaction manag

```
public final class AID {
    public AID(byte[] bArray, short offset, byte length);

    public byte getBytes(byte[] dest, short offset);
    public byte getPartialBytes(short aidOffset,
          byte[] dest, short oOffset, short oLength);

    boolean equals(byte[] bArray, short offset,
          byte length);
    boolean partialEquals(byte[] bArray, short offset,
          byte length);

    boolean RIDEquals(AID otherAID);

    ...
};
```


A. Basic Machinery: `javacard.framework`

- `javacard.framework.AID`
  - *encapsulates the Application IDentifier associated with an applet*
  - *created by the JCRE only*
- `javacard.framework.`
  - *encapsulates an Applicati*
  - *singleton object owned by*
  - *zeroed out by the JCRE be*
- `javacard.framework.`
  - *common static utility funct*
- `javacard.framework.`
  - *collection of methods to c*
    atomic transaction manag

```
public final class APDU {
    public byte[] getBuffer();
    public static byte getProtocol();

    public short setOutgoing();
    public short setOutgoingNoChaining();
    public short setOutgoingLength(short len);

    public short receiveBytes(short offset);
    public short setIncomingAndReceive();

    public void sendBytes(short offset, short length);
    public void sendBytesLong(byte[] data, short offset,
          short length);
    public void setOutgoingAndSend(short offset,
          short length);

    public static APDU getCurrentAPDU();
    public static byte[] getCurrentAPDUBuffer();

    ...
};
```

A. Basic Machinery: `javacard.framework`

- `javacard.framework.AID`
  - *encapsulates the Application IDentifier associated with an applet*
  - *created by the JCRE only*
- `javacard.framework.`
  - *encapsulates an Applicati*
  - *singleton object owned by*
  - *zeroed out by the JCRE be*
- `javacard.framework.`
  - *common static utility funct*
- `javacard.framework.`
  - *collection of methods to c*
    atomic transaction manag

```
public class Util {
    public static short arrayCopy
          (byte[] src, short srcOfs,
           byte[] dest, short dstOfs, short length);
    public static short arrayFill(byte[] bArray,
          short offset, short length, byte value);
    public static short arrayCompare
          (byte[] src, short srcOfs,
           byte[] dest, short dstOfs, short length);

    public static short makeShort(byte b1, byte b2);
    public static short getShort(byte[] arr, short ofs);
    public static short setShort(byte[] array,
          short offset, short value);

    ...
};
```


A. Basic Machinery: `javacard.framework`

- `javacard.framework.AID`
  - *encapsulates the Application IDentifier associated with an applet*
  - *created by the JCRE only*
- `javacard.framework.`
  - *encapsulates an Applicati*
  - *singleton object owned by*
  - *zeroed out by the JCRE be*
- `javacard.framework.`
  - *common static utility funct*
- `javacard.framework.`
  - *collection of methods to c*
    atomic transaction manag

```
public final class JCSystem {
    public static byte getAssignedChannel();
    public static boolean isAppletActive(AID theApplet);

    ...
};
```