Welcome

Kaffe - one for all..

or

..to have a free Java

Peter C. Mehlitz

TransVirtual Technologies, Inc.
peter@transvirtual.com

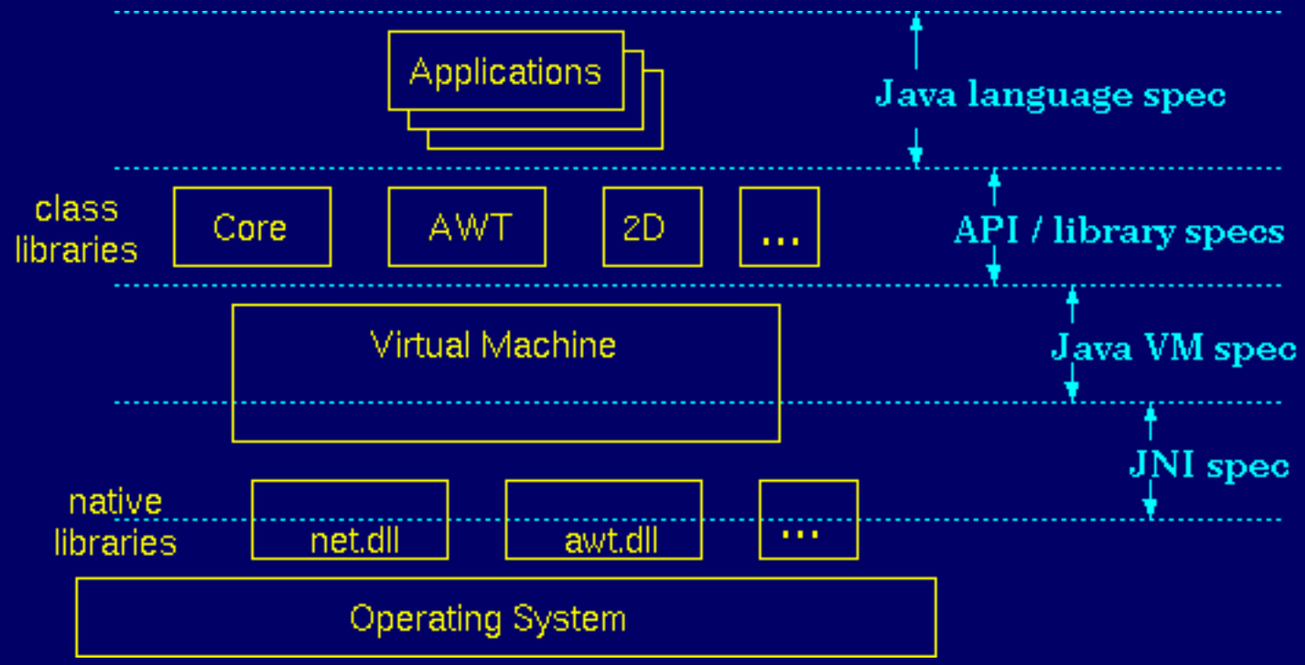
http://www.transvirtual.com/one4all/

Roadmap

- The Basics: "what is.."
  - Java - the tower of specs
  - What is it - a language, an operating system..?
- How About Some Kaffe?
  - From a User Perspective
  - Kaffe for Developers
- Where Are We Heading To
  - down to earth - the Desktop
  - and up to the moon - Embedded Systems

What is Java - the "tower of specs"

- **not** "just another programming language"
- a collection of specs describing a comprehensive, portable programing environment



The Virtual Machine Specification

- type
  - prim. wide/narrow
  - ref wide / narrow
  - ident, String
- context
  - assign, num. promot
  - call, cast
  - String

- on demand ("active" use)
- class / object
- super -> sub
- order
  - fields
  - <init>/<clinit>
- finalize method

- via ConstantPool
  - Fieldref/Methodref -> Class
- load context
  - system ClassLoader
- Errors/Exceptions

- language indep.
- link phase
  - ClassFile format
  - ConstantPool
  - Bytecode/ Methods (instr,stack,locals)
- execution phase
  - type/mthd/field ref

- order of actions
  - use,assign
  - load,store
  - read,write
  - lock,unlock
- volatile vars
- sync/wait/notify

Conversions & Promotions | **Initialization & Finalization** | **Loading & Linking** | **Verification** | **Threading**

Virtual Machine Specification

Exception Handling

Runtime Data Areas | **Types** | **ClassFile Format** | **Bytecode Instructions**

- thread pc registers
- thread stack (frames: locals, operand stack)
- heap (no specific gc)
- method area (classes, code, CP)

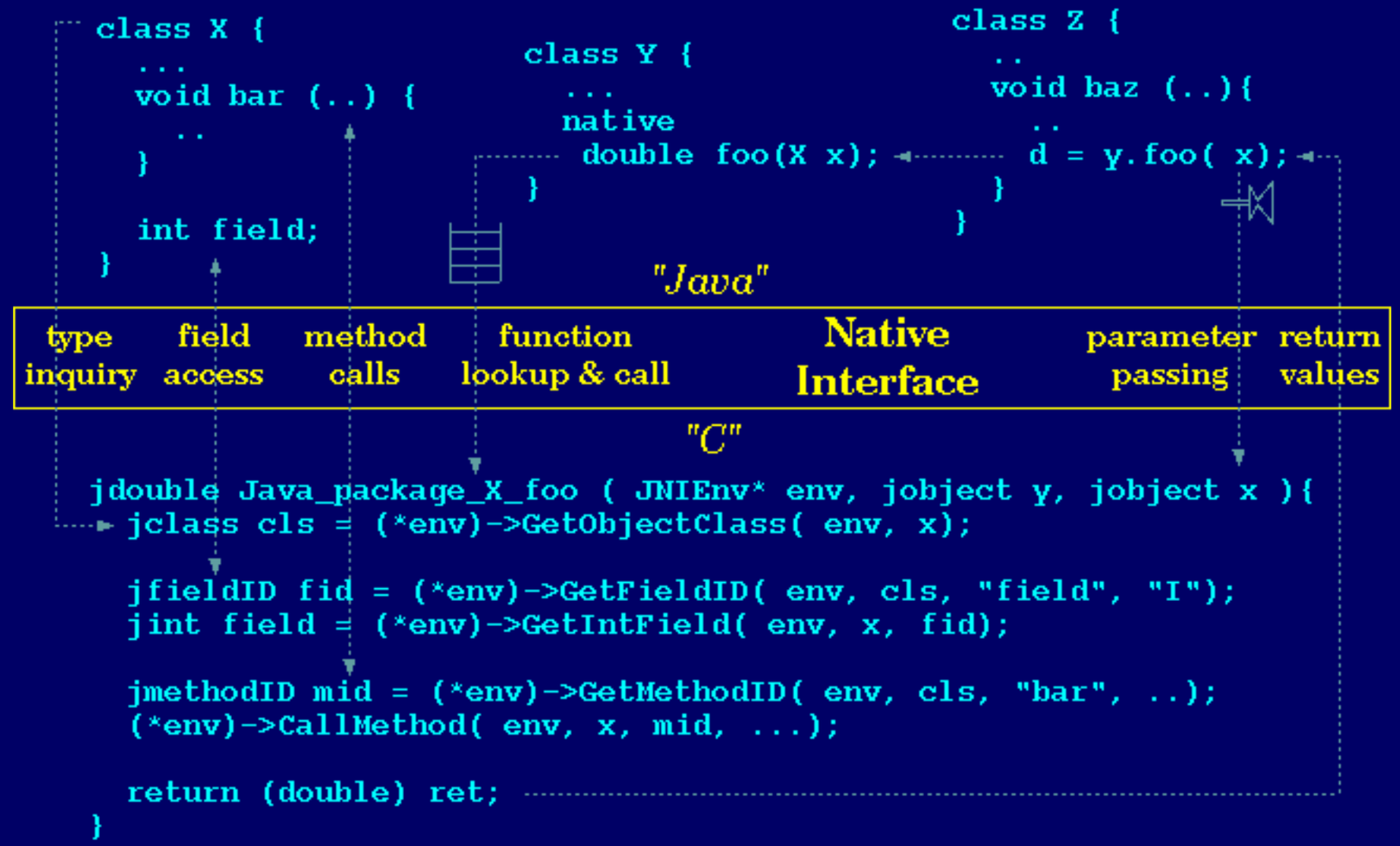
- primitive types
  - byte,char, signed, int. long, float, double
- reference types
  - class
  - interface
  - array

```
struct ClassFile {
...cp_info constant_pool[]
  u2 access_flags
...u2 super_class
...u2 interfaces[..]
...field_info fields[..]
...method_info methods[..]
...attribute_info attributes[.]
}
```

- load & store (iload, dstore,..)
- arithmetic (iadd, fmul, iinc, ..)
- conversion (i2l, f2d, ..)
- creation (new, newarray, ..)
- stack (pop, dup2, swap, ..)
- control (ifeq, tableswitch, ..)
- invocation (invokevirtual,..)
- exception (athrow)

The Java Native Interface (JNI)

- more than just "calling a C function"
- interfacing Java to the outside world is part of Java specs

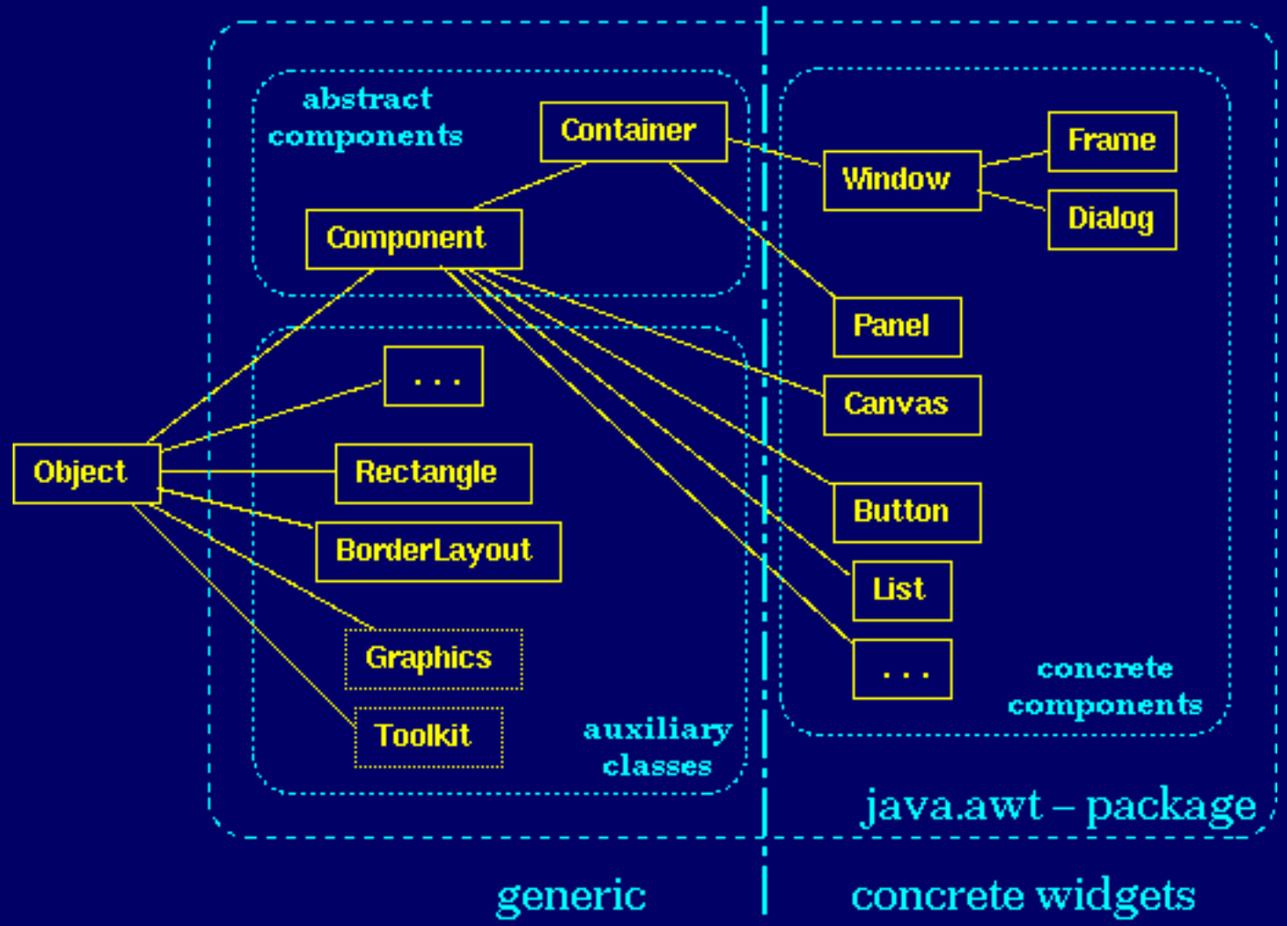


API Specs

- *Application Programming Interfaces* are the fasted moving part of Java development
- many categories (core, AWT, extensions like 2D, 3D, ..)
- whitepapers for some categories (e.g. JDBC), core API also covered (partly) by language and VM spec
- weakest part of spec tower, mostly described by javadoc generated online documentation
- come with sources not requiring a Java source license (but that doesn't do us any good)

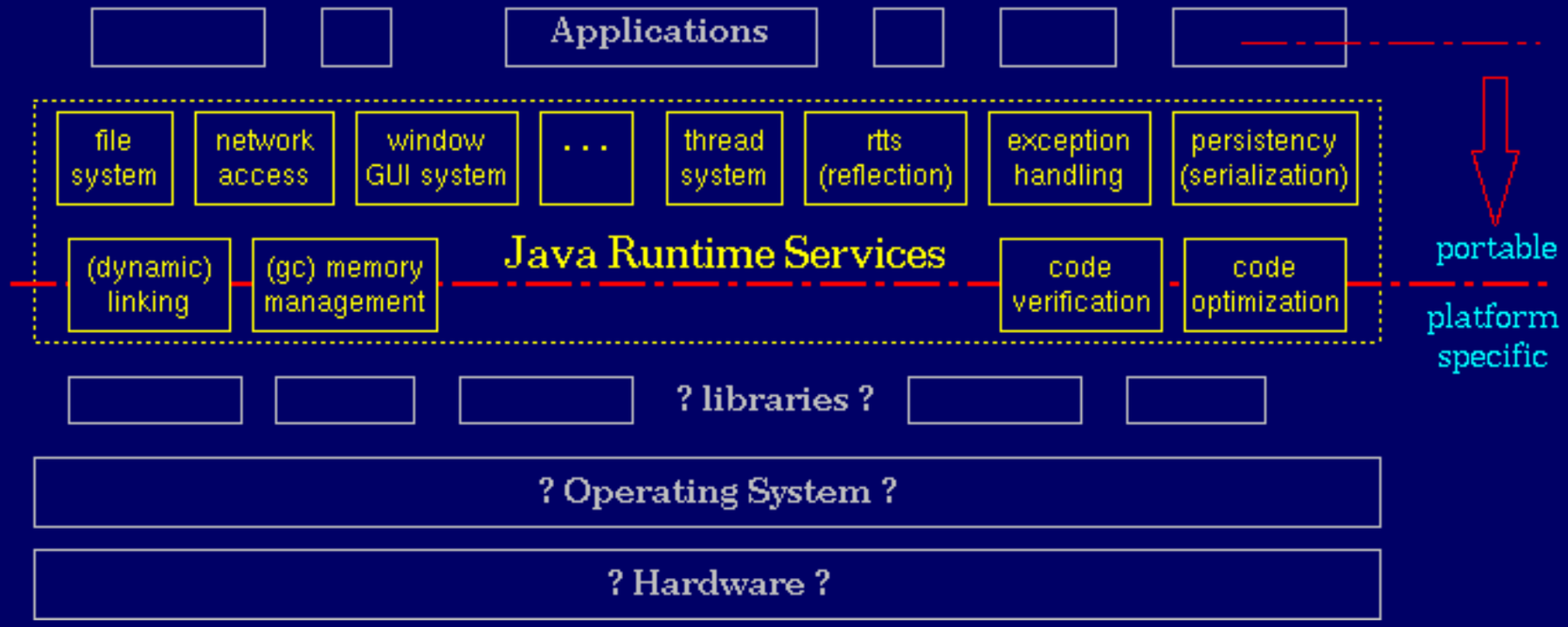
Abstract Window Toolkit Spec

- AWT provides portable abstraction of a GUI system
- includes generic parts to build own widgets as "lightweights"



What is it - language or OS ?

- with respect to runtime services, Java is much more a "Middleware Operating System" than just a language
- includes enough functionality to act as a "standalone" OS (native lib, optional platform OS and HW are completely hidden)
- moves the boundary of platform specific code out of the applications
- reads like a computer science curriculum



Conclusion

too much to remember ?
(you have my sympathy)

- Java is not just a language, it is a "middleware OS"
- Kaffe is not "just" a VM, it is a framework for tailored Java systems
- its desktop future: public sources (GPL), efficiency
- its embedded systems future: portability, scalability

The real future? It's all up to us: don't moarn, fix and improve. Don't grab, contribute.

Thank You!

Kaffe - what is it?

- Where does this strange name come from? How is it pronounced?

  Kaffe ['kafe] is the swedish word for coffe (the work was started in Stockholm)

- How does it compare to the Java standard?

  PersonalJava 1.1 compliant system, consisting of class libraries, VM (incl. JIT) and native libraries

- Is is "just another Java implementation"?

  No, it has a different attitude: providing a framework for using (target system) specialized components (like GC, threads, AWT), all tuned towards: Portability, Scalability, Efficiency

- Is it commercial or free?

  the *Desktop Edition* (complete, non-crippled) is GPL, the *Custom Edition* (subsystems for use in embedded systems) is commercial

- How long has it been around?

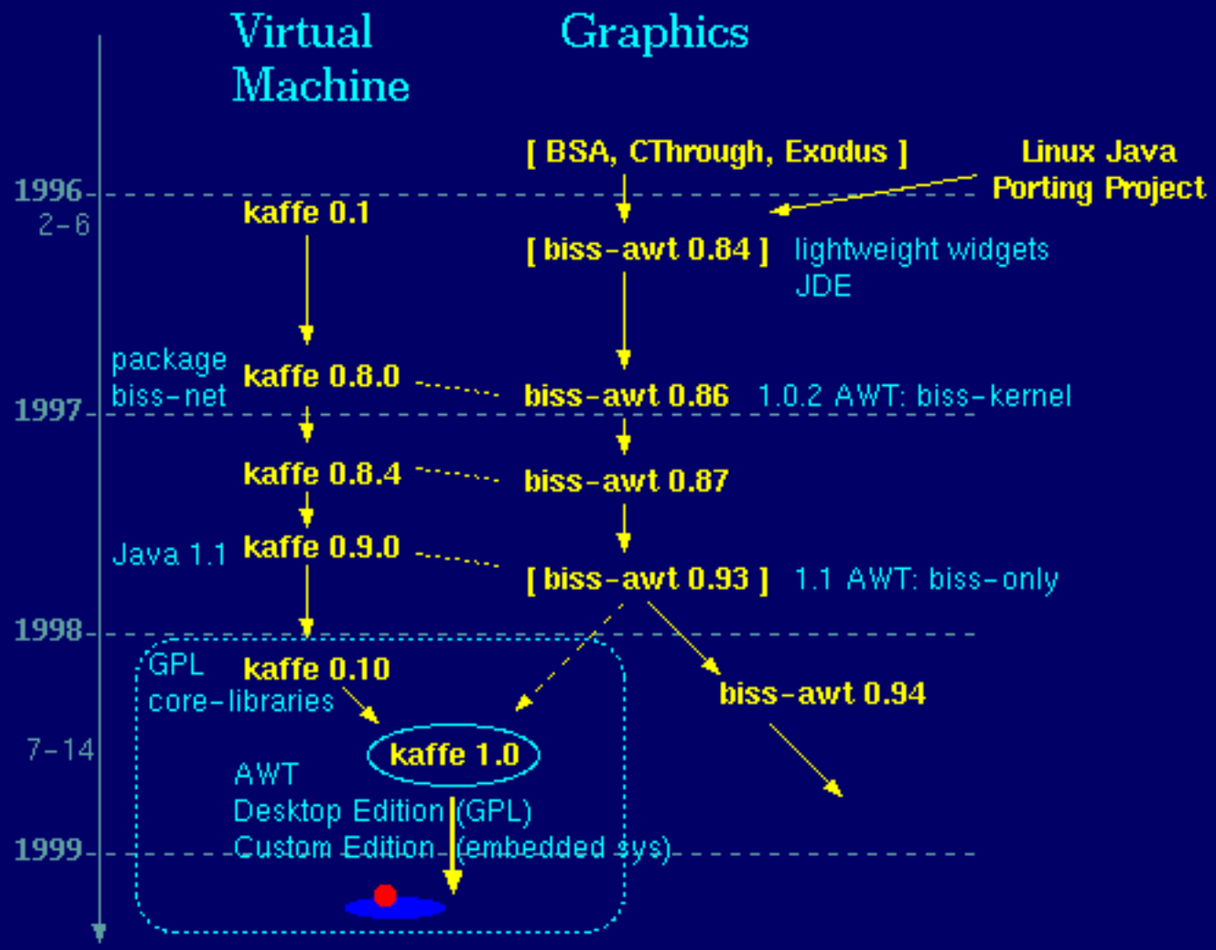
  first release in 2/96, first graphics in 12/96 (biss-awt), first autark GPL release in 6/98. See history

- Who is using it?

  More than 14000 downloads since 6/98 just from the TransVirtual server

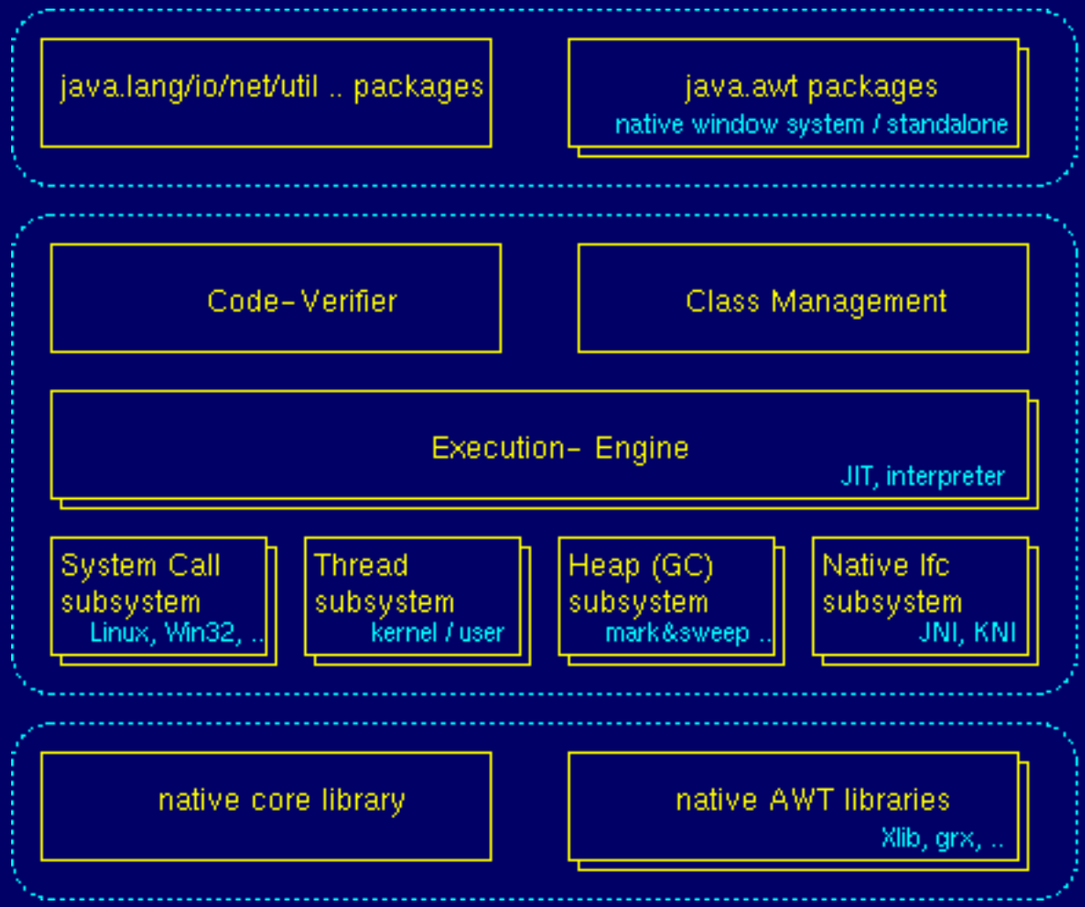
History of Kaffe

- kaffe is a result of combined efforts
- this is not a first try - kaffe has been around for a while



Kaffe from a User Perspective

- complete system (in conjunction with the Pizza compiler, viable alternative for JDK)
- available under GPL, no third party royalties / license restrictions
- small (400K for a statically linked executable, 500K class libs)
- portable (about 30 different operating systems, 8 HW architectures)
- integrated JIT (bridging the gap to C, see performance figures)



PersonalJava 1.1 compliant class – libraries

modular Java 1.1 compliant virtual machine

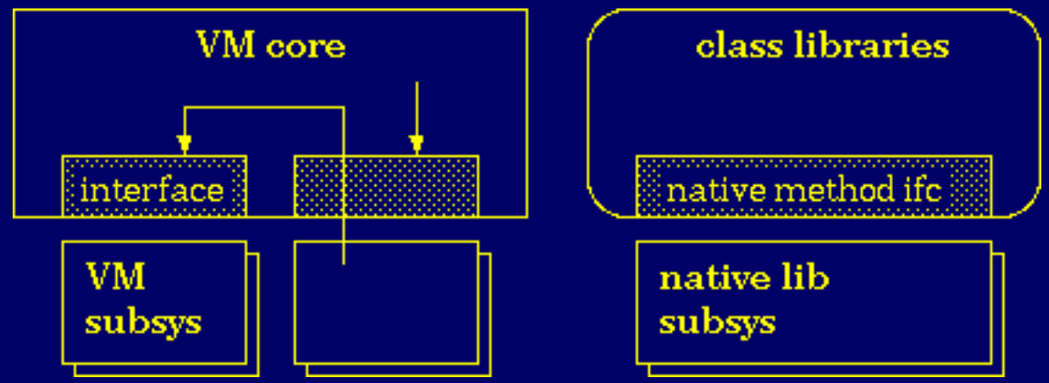
statically / dynamically linked native libraries

Inside Kaffe

- The Main Theme: Modularity
- Core Structures
- Interfaces and Subsystems
  - JIT
  - Interpreter
  - Threading
  - Heap Management
  - Graphics Support (AWT)
- Common InfraStructure
- The Source Tree
- Porting Kaffe

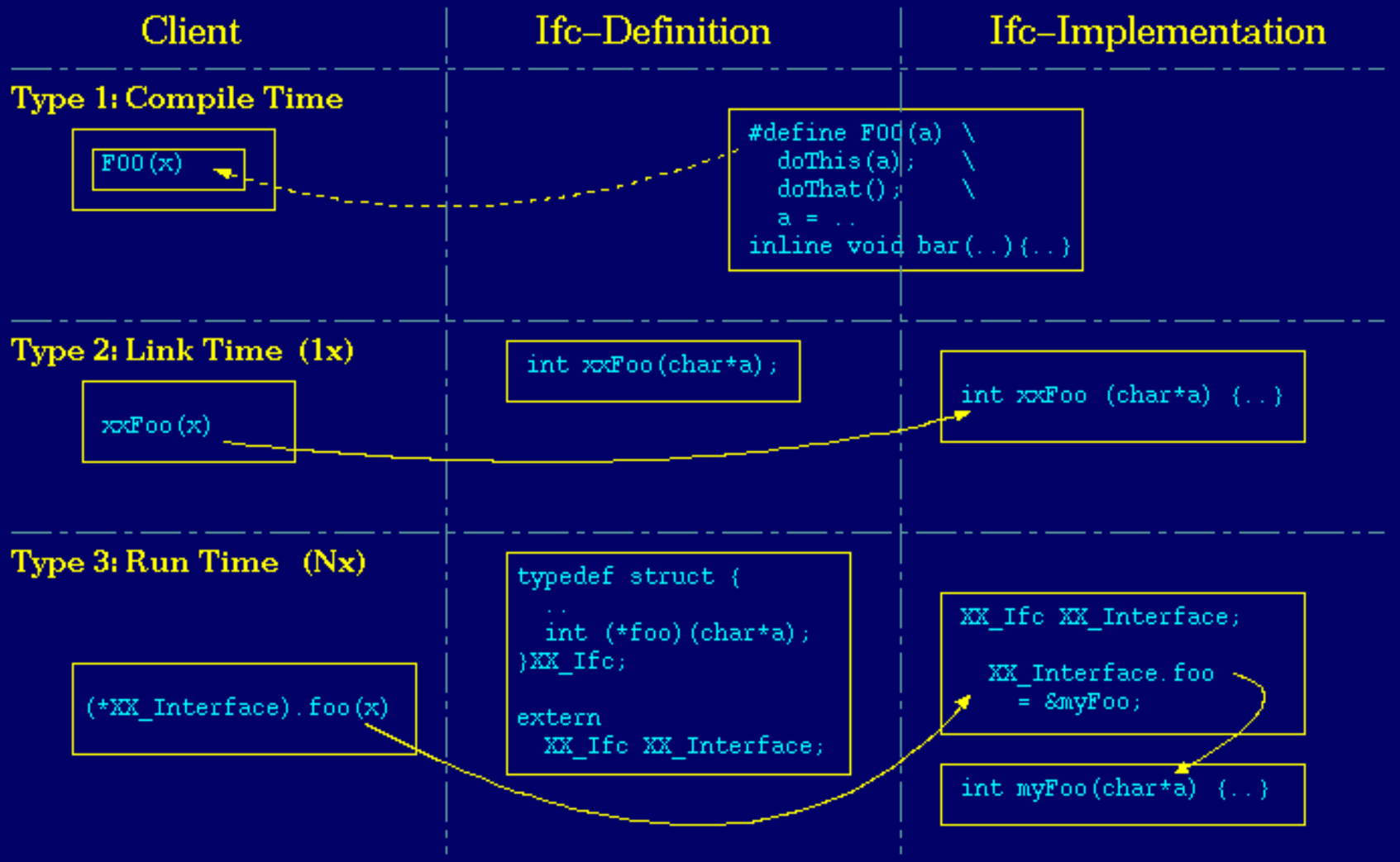
Modularity: Internal Interfaces

- there are several well separated sub-systems (heap management/GC, threading, JIT, graphics support, ..)
- there are numerous different target systems and application requirements (Smartcard to server, graphical / textual app, browser context / stand-alone, ..)
- there is NO silver bullet - to serve them all (optimal) with the same subsystems
- there IS a strong need to make them configurable (pick the right subsystems for the right problem)
- Kaffe uses several types of interfaces



Types of Interfaces

- Kaffe uses mostly (narrow) runtime interfaces for VM subsys, and (wide) link-time interfaces for native lib subsys


Client
Ifc–Definition
Ifc–Implementation
Type 1: Compile Time
FOO(x)
#define FOO(a) \
doThis(a); \
doThat(); \
a = ..
inline void bar(..){..}
Type 2: Link Time (1x)
xxFoo(x)
int xxFoo(char*a);
int xxFoo (char*a) {..}
Type 3: Run Time (Nx)
(*XX_Interface).foo(x)
typedef struct {
..
int (*foo)(char*a);
}XX_Ifc;
extern
XX_Ifc XX_Interface;
XX_Ifc XX_Interface;
XX_Interface.foo
= &myFoo;
int myFoo(char*a) {..}

Basic Structures

include/java_lang_Object.h

Object

dtable
lock

kaffevm/classMethod.h

dispatchTable

class

method[0]
...

kaffevm/classMethod.h

Class

Object head

centry
name
accflags
superclass
constants

methods
nmethods
dtable
msize

fields
bfsize
nfields
nsfields

interfaces
interface_len
total_interface_len

loader
state
flags

kaffevm/classMethod.h

methods

name
sig
accflags
idx

stacksz
localsz

ncode
c
  {start, end} |
  {code, codelen}

class
lines

exception_table
nextActiveMethod

kaffevm/classMethod.h

field

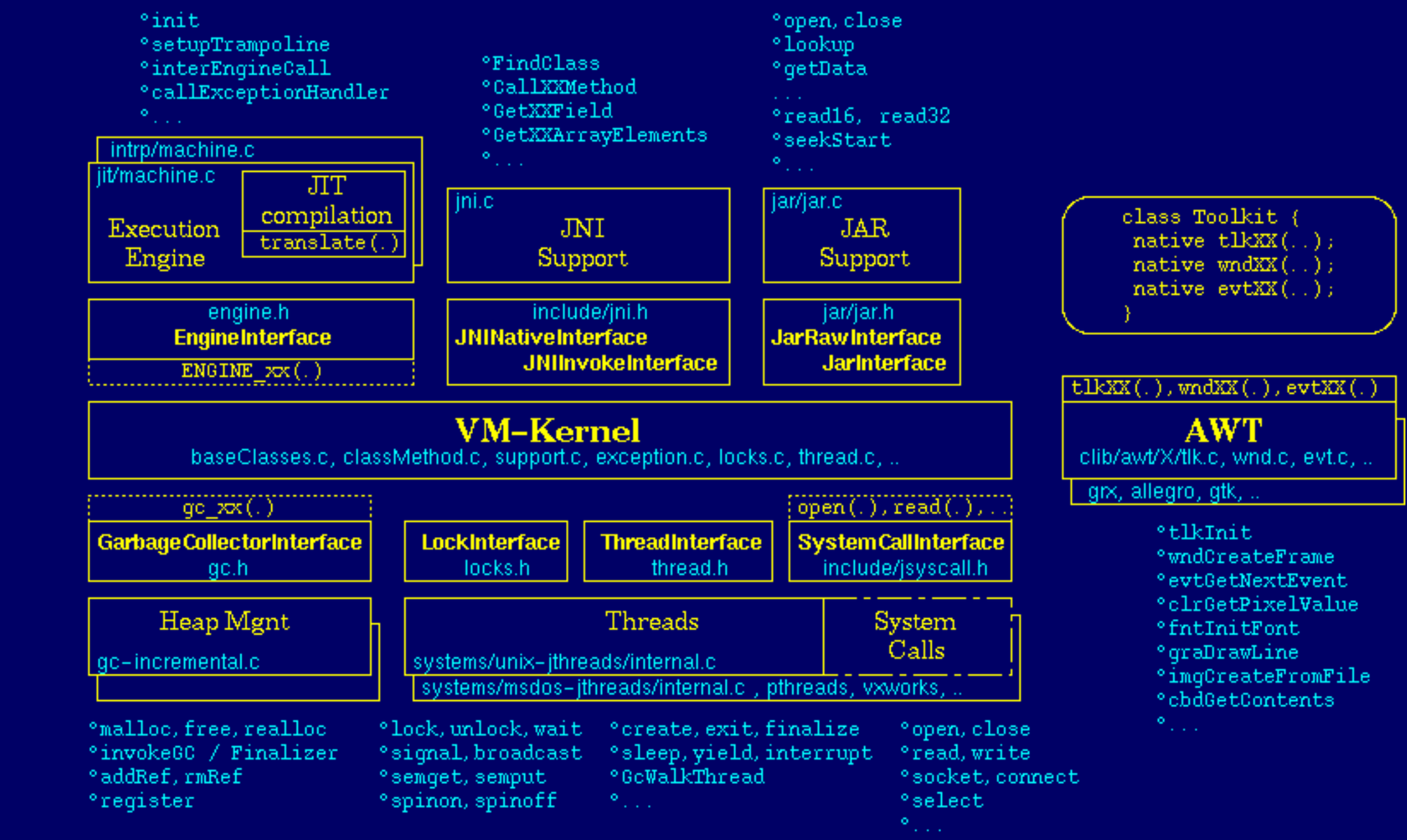
name
type
accflags
...
bsize
info
  boffset |
  addr |
  idx

trampoline

native code

bytecode

Kaffe Interfaces

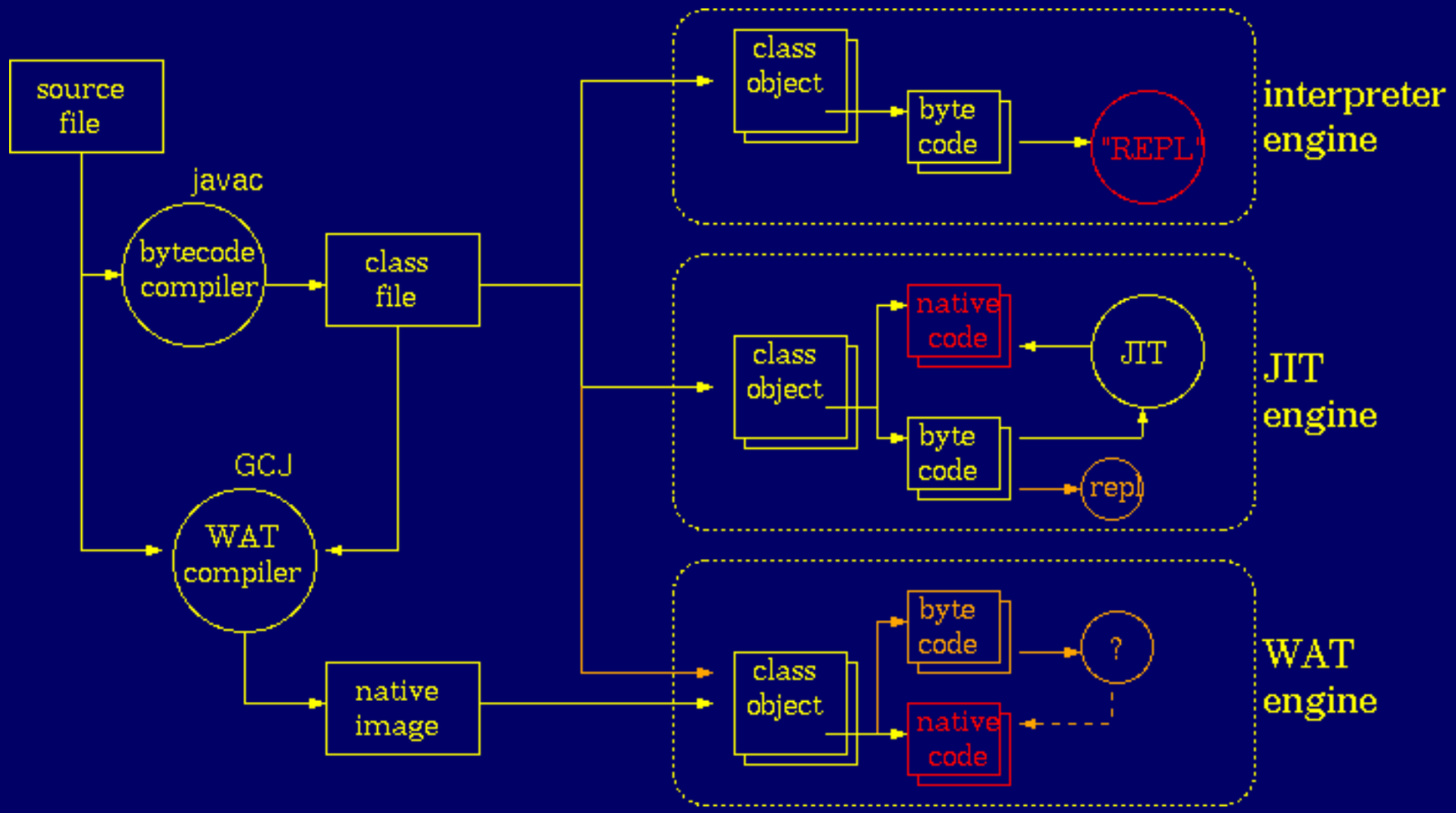


Kaffe Source Tree

```
kaffe
├ config                  configuration                          config.h
│ ├ i386                    architecture: Intel
│ │ ├ linux                   OS: linux
│ │ ├ ..
│ │ └ ..
├ include                 global includes                        jsyscall.h jtypes.h jni.h Object.h
├ kaffe                   VM root                                                                    VM
│ ├ kaffe                   executable
│ └ kaffevm                 VM library                  67 22 465   icode.h jit.def md.c trampolines.c
│   ├ intrp                   interpreter engine         6  1  38
│   ├ jit                     JIT compiler engine       19  8 176
│   ├ jar                     Java archive               4  1  15
│   ├ mem                     heap management & GC       4  2  47
│   └ systems                 thread systems
│     ├ unix-jthreads           kaffe user threads       3  2  51
│     └ ..
├ libraries               libraries root
│ └ clib                    native libraries
│   ├ awt                     java.awt
│   │ ├ X                       Xlib                    13  5 130
│   │ └ ..
│   ├ math                    java.math
│   ├ management              kaffe runtime env. (classpath..)
│   ├ native                  java.lang, java.io        65  6 146
│   ├ net                     java.net
│   └ zip                     java.util.zip
└ javalib                  Java class libraries
  ├ java                    standard packages
  │ └ ..
  └ kaffe                   kaffe specific extensions
```

	files	kloc	kB
*.[ch]	430	70	1690
*.java	550	60	1260
*.def	15	20	350
Σ	995	150	3300

JIT (1) - Execution Engine Types



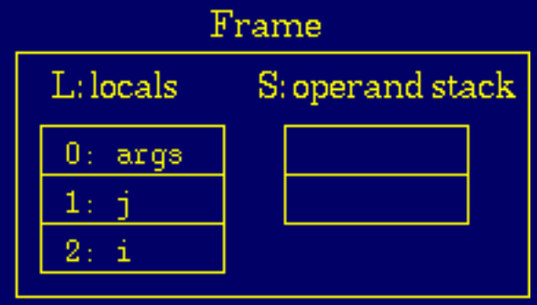
JIT (2) - Java to Bytecode

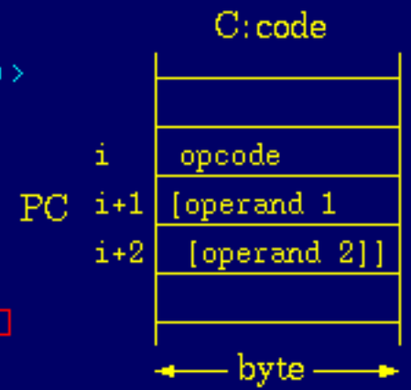
```
static
void main (String[] args)
{
  int j;
  int i = 3;

  if ( i < 5 )
    j = i;
  else
    j = i+1;

  j = foo( i);
}
```

```
void main(java.lang.String[])
   0 iconst_3
   1 istore_2
   2 iload_2
   3 iconst_5
   4 if_icmpge 12
   7 iload_2
   8 istore_1
   9 goto 16
  12 iload_2
  13 iconst_1
  14 iadd
  15 istore_1
  16 iload_2
  17 invokestatic #4 <foo(int)>
  20 istore_1
  21 return
```

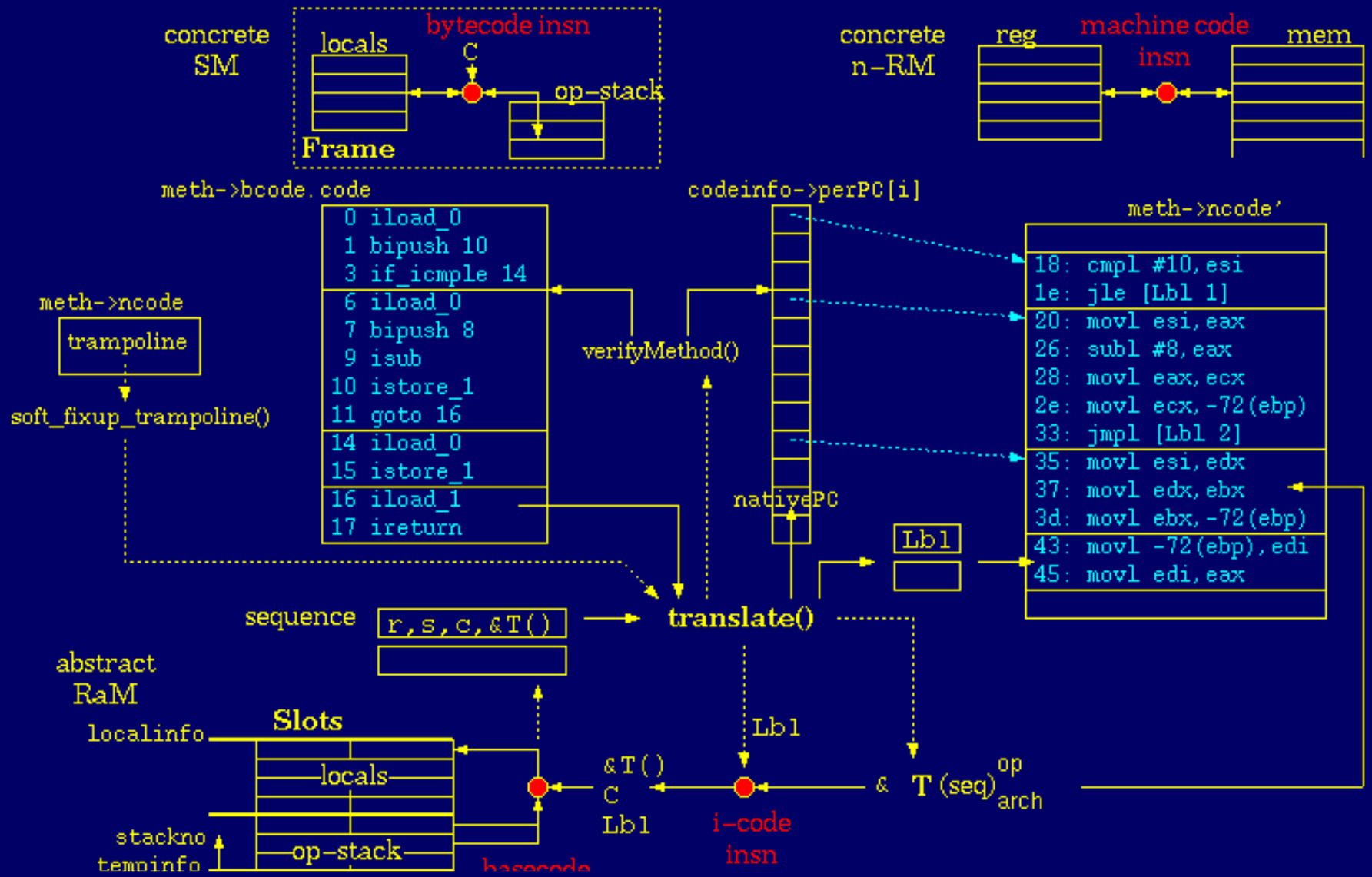


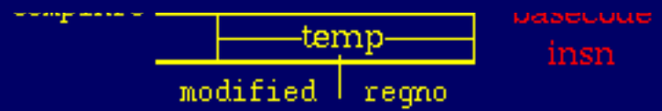


bc_insn: PC, C, L, S —op→ {(cond, PC', L', S')}

PC, C[o1,o2], S[..,v] —**ifge**→ v >= 0: PC+((o1<<8)|o2), S[..]
v < 0: PC+1, S[..]

JIT (3) - Stack Machine to Reg Machine




temp
modified
regno
insn

JIT (4) - Translation Code

machine.c

```
translate (Method* meth){
  verifyMethod(meth)

  for ( pc=0; pc<len; pc = npc ){
    npc=pc + insnLen[bc[pc]];
    switch ( bc[pc] ){
      ..                                  kaffe.def
      case BIPUSH:
        low = (int8)getpc(0);
        push(1);
        move_int_const(stack(0),low);
        break;
      }
      ..
      if ( codeInfo.codePC[pc].flags
            & STARTOFBASICBLOCK )
        // spill stack,locals,temps

      for ( <each seq> )
        (*(seq->func))(seq);
    }
    linkLabels(..);
    installMethodCode(..)
  }

#define push(i)    stackno -= i
#define stack(i)   &localinfo[stackno+i]
```

icode.c

```
move_int_const (SlotInfo* dst, jint val) {
  slot_slot_const( dst, 0, val,
                   HAVE_move_int_const, Tconst); ..
}
```

icode.h <– config/<arch>/jit–icode.h

```
#define HAVE_move_int_const move_RxC
```

funcs.c [jit.def] <– config/<arch>/jit–<arch>.def

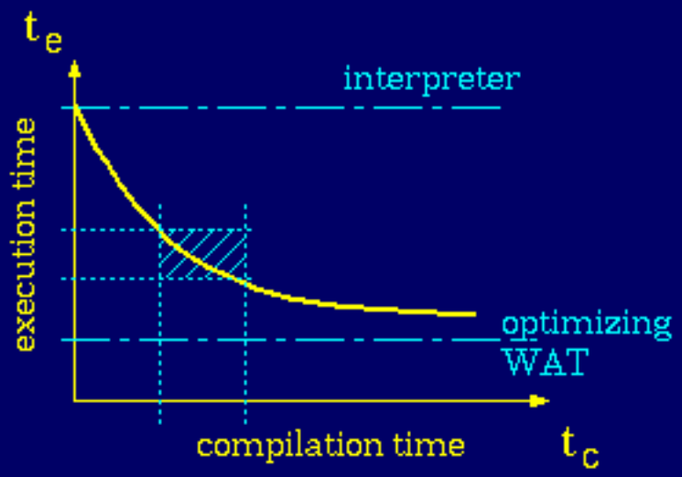
```
move_RxC ( sequence* s ){
  int val = const_int(2);
  int w   = wreg_int(0);
  // movel #val %w
  codeblock[CODEPC++] = 0xB8|w;
  codeblock[(CODEPC+4)-4] = val;
}
```

basecode.c

```
slot_slot_const (SlotInfo* dst, SlotInfo* s1, jword s2,
                 ifunc f, int type) {
#if defined(TWO_OPERAND)
  if (s1 && dst && (s1 !=dst) && (type!= Tload)..){
      move_any(dst, s1);
      s1 = dst; ..
#endif
  sequence *seq = nextSeq();
  ASSIGNSLOT(seq->u[1], s1);
  seq->u[2].iconst = s2;
  ASSIGNSLOT(seq->u[0], dst);
  seq->func = f;
}
```

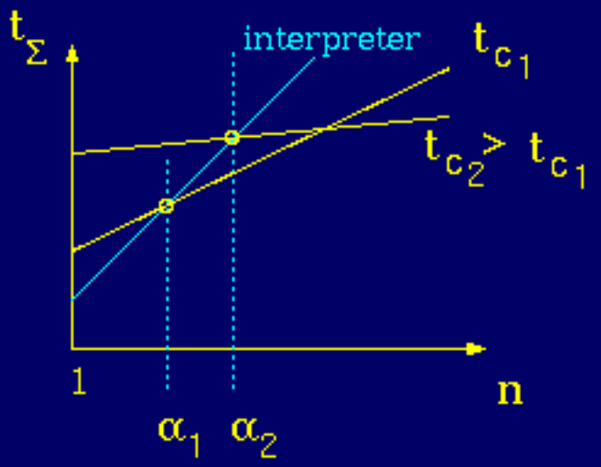
JIT (5) - Optimal JIT?

- simple coherence, difficult conclusion: execution time depends on optimization time
- there is a different "break even" number of calls for each JITed method
- number of calls cannot be predicted in advance
- adaptive JIT compilation (intrp/JIT) adds additional overhead
- adaptive JIT optimization (global reg, peephole) depending on verifier output seems to be better

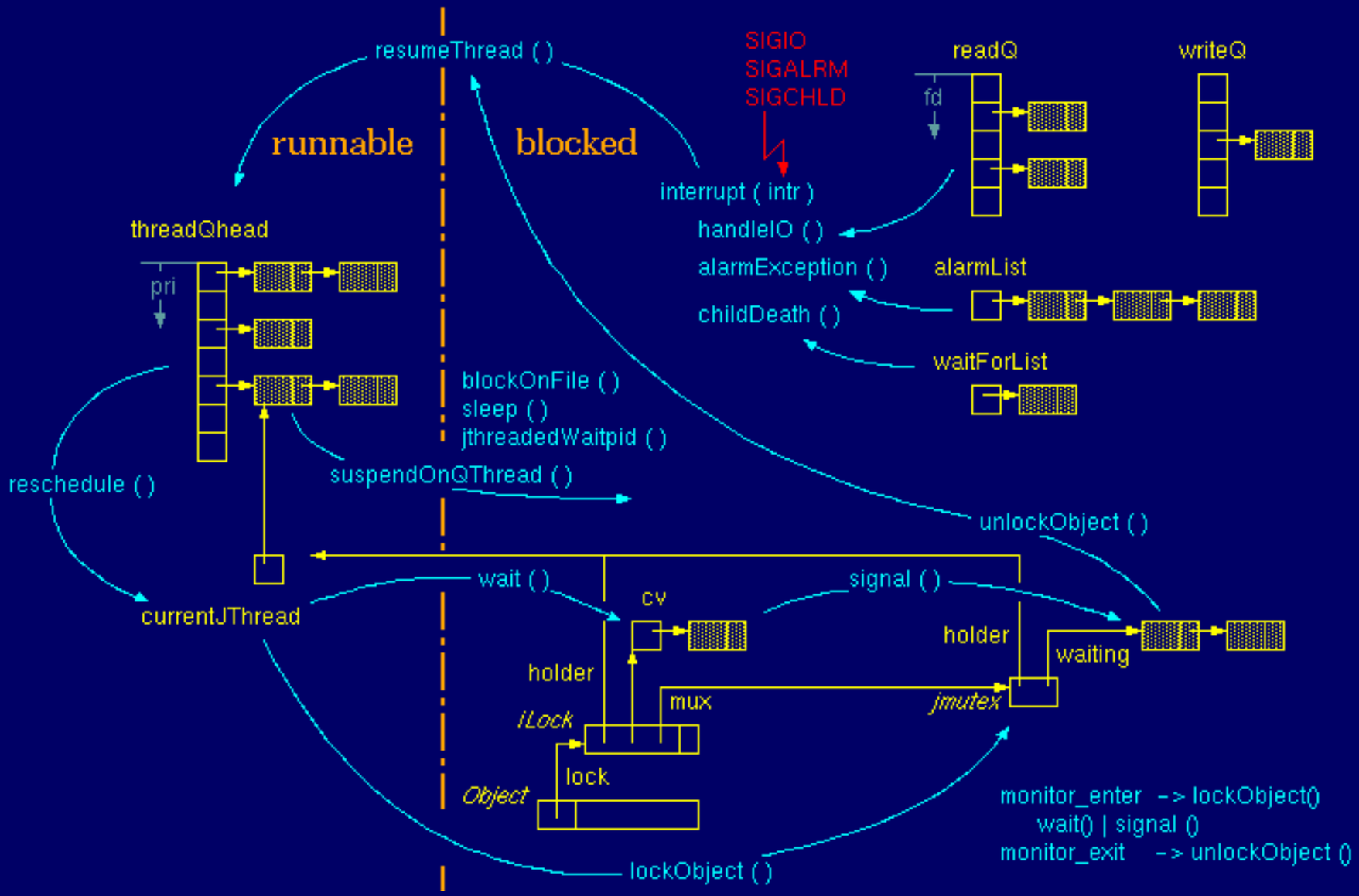


$$t_c = f(bc)$$
$$t_e = f(mc(t_c))$$

$$t_\Sigma(n) = t_c + n * t_e$$



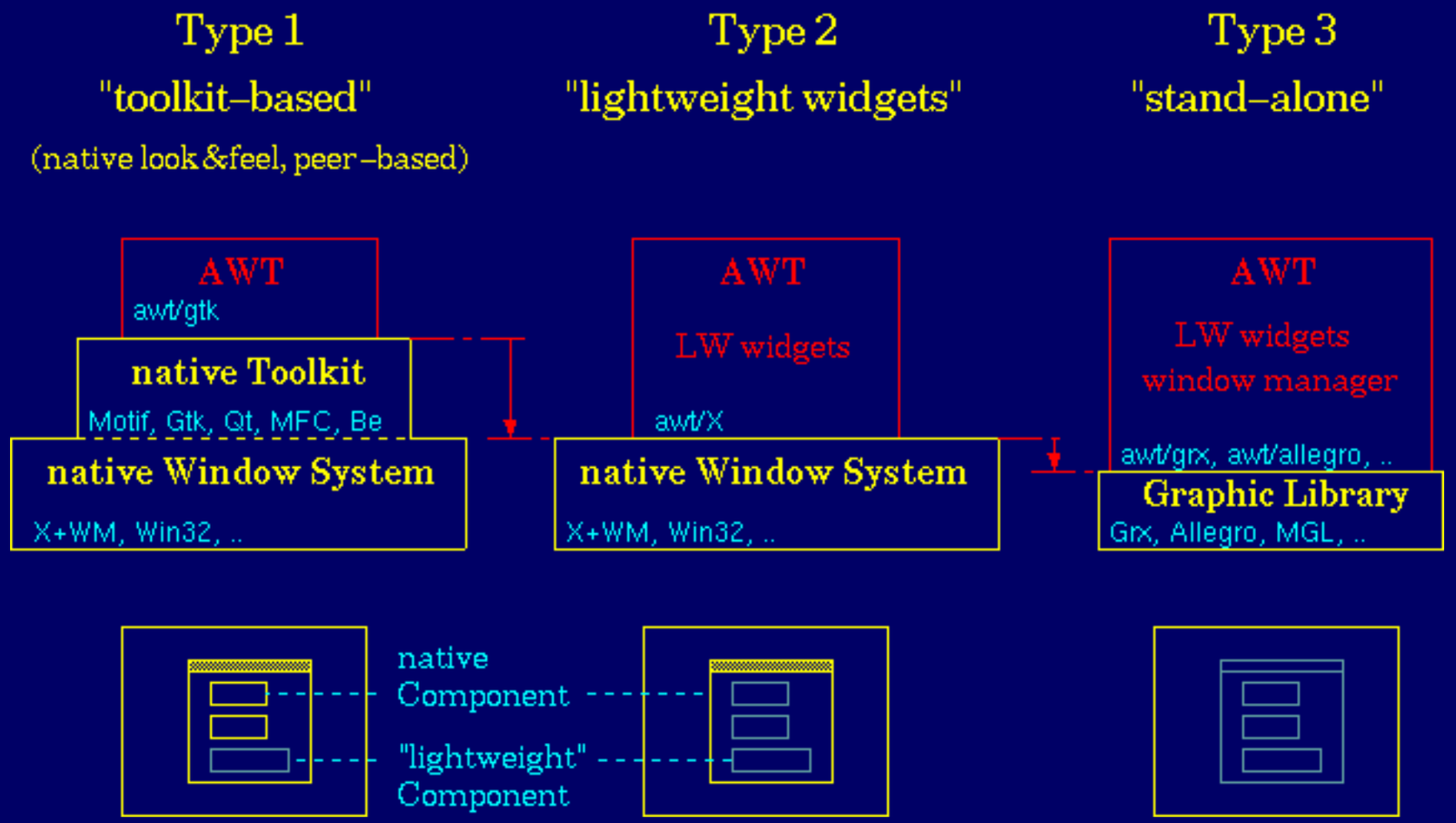
JThreads - Queues Everywhere



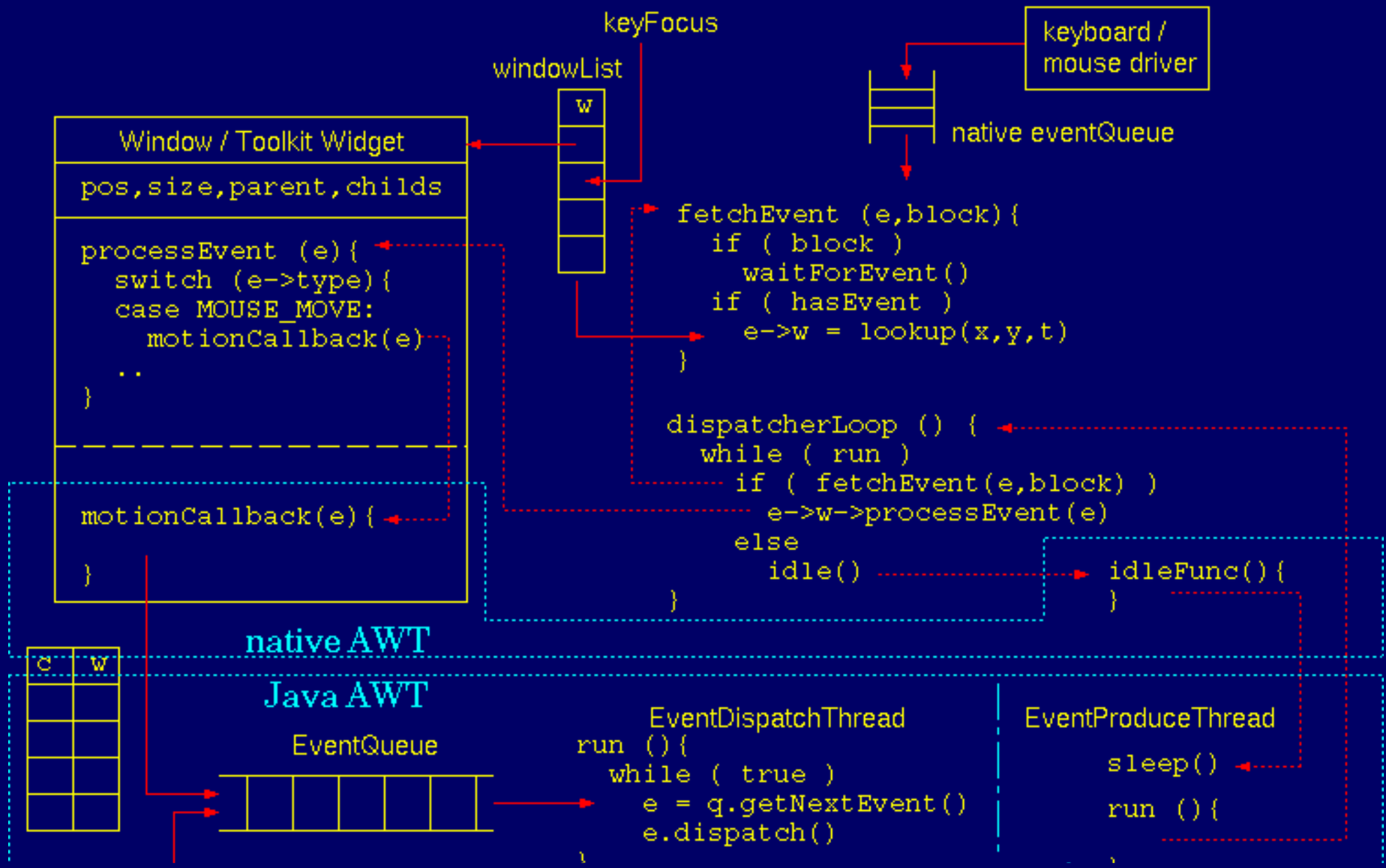
!

AWT (1) - Types

- differences show up especially in event processing

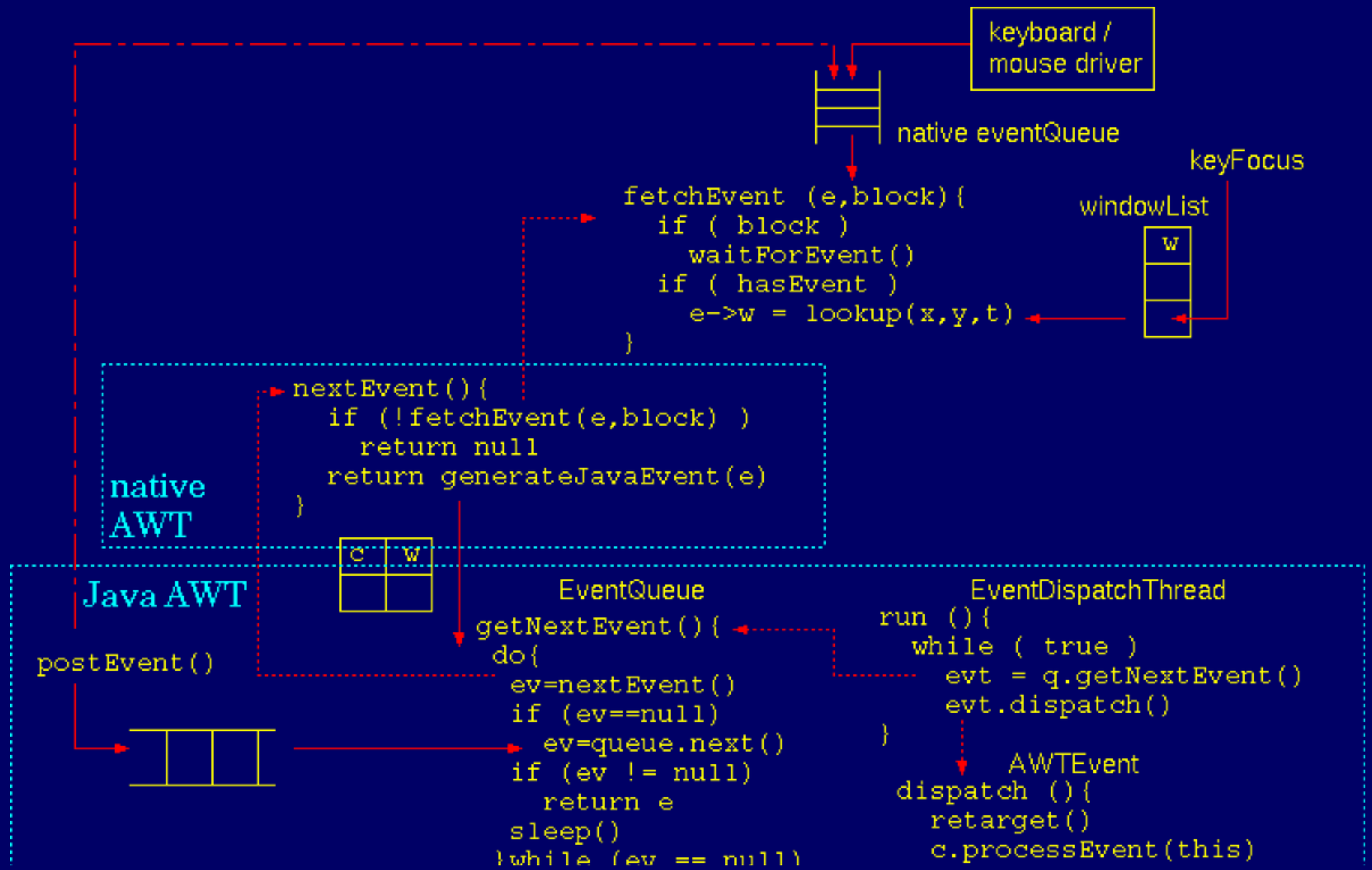


AWT Type-1 Event Processing



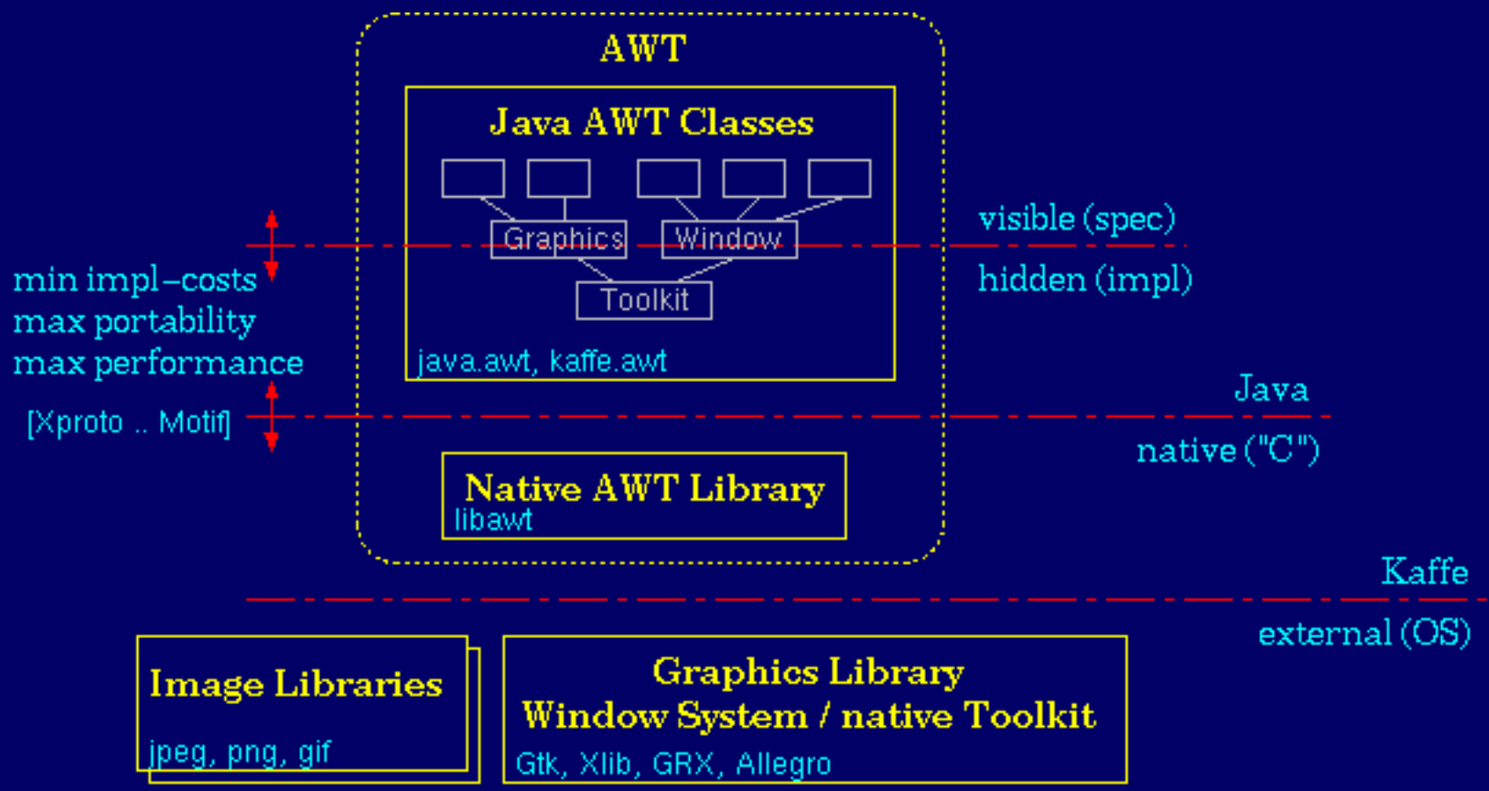

postEvent()
context switch

AWT Type-2 Event Processing

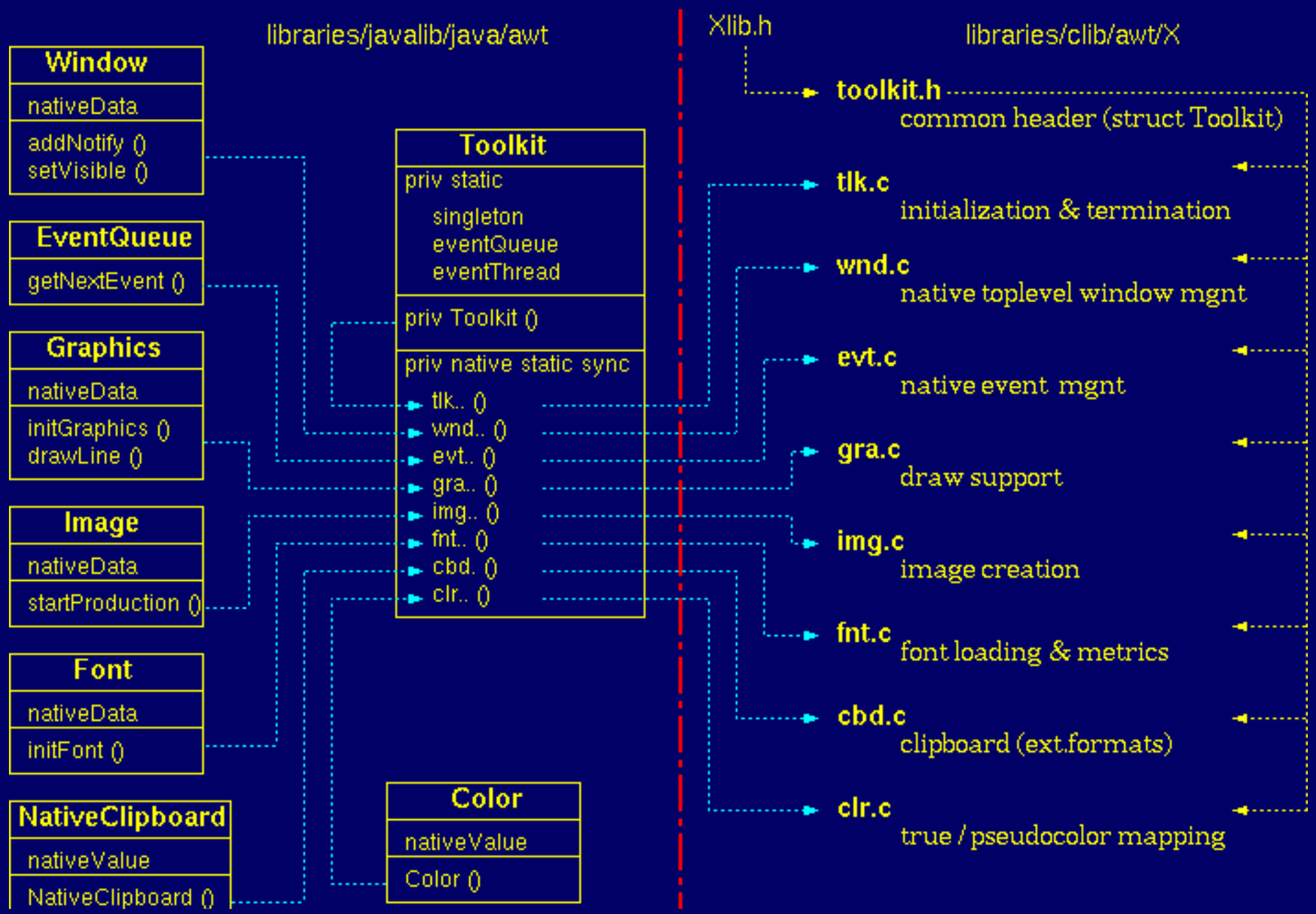


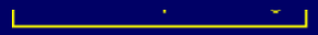
AWT (2) - Layers

- AWT implementation: providing a portable abstraction (public java.awt classes) for a plethora of different graphical environments
- involves three layers of decreasing portability
- requires substantial "external" functionality



AWT (3) - Map


libraries/javalib/java/awt
Xlib.h
libraries/clib/awt/X
Window
nativeData
addNotify ()
setVisible ()
EventQueue
getNextEvent ()
Graphics
nativeData
initGraphics ()
drawLine ()
Image
nativeData
startProduction ()
Font
nativeData
initFont ()
NativeClipboard
nativeValue
NativeClipboard ()
Toolkit
priv static
singleton
eventQueue
eventThread
priv Toolkit ()
priv native static sync
tlk.. ()
wnd.. ()
evt.. ()
gra.. ()
img.. ()
fnt.. ()
cbd. ()
clr.. ()
Color
nativeValue
Color ()
toolkit.h
common header (struct Toolkit)
tlk.c
initialization & termination
wnd.c
native toplevel window mgnt
evt.c
native event mgnt
gra.c
draw support
img.c
image creation
fnt.c
font loading & metrics
cbd.c
clipboard (ext.formats)
clr.c
true / pseudocolor mapping



Java | C

Infrastructure (1) - JDebug

- multithreaded, dynamically generated code -> hard to debug
- standard facility for runtime configurable debug output

jdebug.h (standard header)

```
#if defined(DEBUG)
  #define DBG(topic,msg)..
  #define DBG_ACTION(topic,action)..

  #if defined(DEBUG_MAIN)
    #define DBG_DEFINE(topic) int topic
  #else
    #define DBG_DEFINE(c) extern int..
  #endif

  #include "jdebugdef.h"
#else
  ..
#endif
```

jdebugdef.h (app specific)

```
DBG_DEFINE( awt, "..");
DBG_DEFINE( awt_wnd, "..");

#if defined(DEBUG_MAIN)
void dbg_init() {
  DBG_CHECK( awt);
  DBG_CHECK( awt_wnd);
}
#endif /* DEBUG_MAIN */
```

app.c

```
#define DEBUG_MAIN
#include "jdebug.h"
```

client.c

```
#include "jdebug.h"

DBG_ACTION(awt,XSynchronize(..));
  ...
DBG(awt_wnd,("create: %x\n",wnd));
```

build

gcc .. –DDEBUG ..

run

export DEBUG="awt awt_wnd"

Infrastructure (2) - JMeasure

- standard facility for runtime configurable statistics
- tries to please Heisenberg ($\Delta x * \Delta v_x = \text{const}$) -> no heap, efficient checks

jmeasure.h (standard header)

```
#if defined(MEASURE)
  #define MSR_INC(c)..
  #define MSR_START(c)..
  #define MSR_STOP(c) ..
  #define MSR_LOG(c)..

  #if defined(MEASURE_MAIN)
    #define MSR_DEFINE(c) MsrChannel c
  #else
    #define MSR_DEFINE(c) extern MsrChannel..
  #endif

  #include "jmeasuredef.h"
#else
  ..
#endif
```

jmeasuredef.h (app specific)

```
MSR_DEFINE_LOG(1000)
MSR_DEFINE( jit_time, "..");
MSR_DEFINE( total_time, "..");

#if defined(MEASURE_MAIN)
void msr_init() {
  MSR_CHECK( jit_time);
  MSR_CHECK( total_time);
  MSR_START( total_time);
}

void msr_finish() {
  MSR_STOP( total_time);
  MSR_FINISH();
}
#endif /* MEASURE_MAIN */
```

app.c

```
#define MEASURE_MAIN
#include "jmeasure.h"
```

client.c

```
#include "jmeasure.h"

MSR_START(jit_time);
  ...
MSR_STOP(jit_time);
```

build

gcc .. -DMEASURE ..

run

export MEASURE="jit_time total_time"

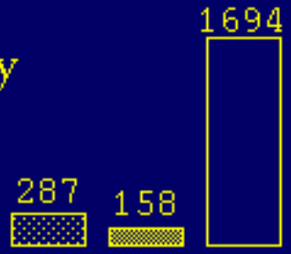
Porting Kaffe

- Virtual Machine
  - start with interpreter
  - JIT is hard part (depending on archirecture): trampolines, call setup, alignment
  - jthreads: setjmp/longjmp, signals, setitimer, select
  - native threads: POSIX-like?
- Graphics
  - type-2 AWT is easiest to port
  - start with polling getNextEvent(), learn about blocking native operations
  - start with TrueColor visual (PseudoColor depends on Colormap access)
  - native image format
  - font access (independent of Graphics target?)

"C" - performance

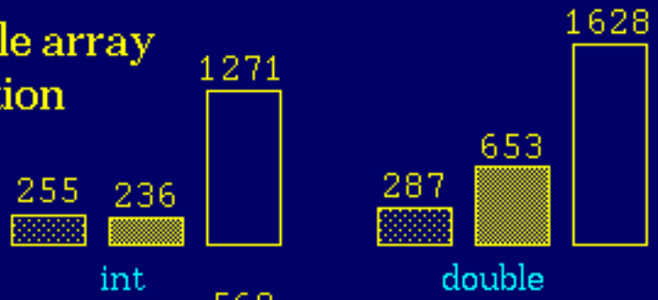
- Java = interpreter = slow ? Not true for JIT !
- consider additional Java functionality (exception handling, array check, GC, ..)
- builtin types, static methods, pre-allocated objects:
  no reason why JITed Java should be slower than compiled C (put aside optimization)

byte array set / test




```
while ( rounds-- != 0 ) { // rounds=30000
  for ( i=0; i<n; i++ ) { // n=100
    a[i] = 1;
    if ( a[i] == 2 ) break;
  }
}
```

int / double array computation



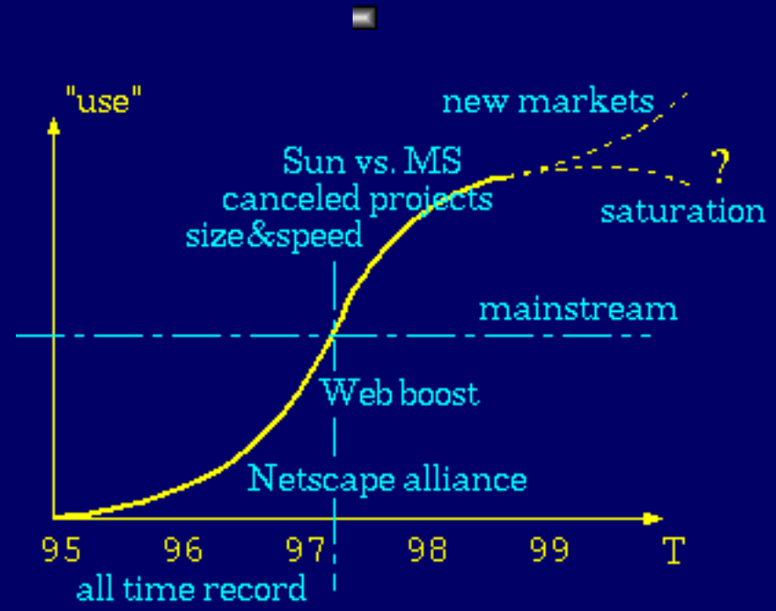

```
while ( rounds-- != 0 ) { // rounds=10000
  for ( i=0; i<n; i++ ){  // n=100
    a[i] = i;  b[i] = i*2;
  }
  for ( i=0, res=0; i<n; i++ )
    res += a[i] * b[i];
}
```

isolated call


```
int foo ( int a, int b ){
  return 1;
}

while ( rounds-- != 0 ) { // rounds=3000000
  foo( 1,2);              // <- test
}
```

Future of Java



- no programming environment ever got mainstreamed so fast
- growth of acceptance now declines somewhat
- **BUT:** Java so far has been validated just in terms of web programming (which was just a "booster")

- Java still gives the right technical answers to real problems:
  - binary compatibility
  - efficiency Provisions
  - comprehensive specs.
  - robustness
  - std. runtime libaries
  - implementation costs
- Java will gain additional momentum if it delivers on reducing application development / distribution costs for:
  - distributed, open applications -> Web (embedded Java)
  - new HW / devices -> **Java in embedded systems**
  - new operating systems / versions / toolkits -> **Java for desktop applications**

Kaffe to the Desktop !

Many applications don't need direct access to OS-specific functionality (email clients, calendars, editors, DB front-ends, Web-browsers ..)

why to program them to a specific OS / GUI toolkit API (version)?

why not to use Java to do it portable ?

- myths
- memory consumption

  improve subsystems (GC, JIT), share resources (Java as a shell, Kaffe server)
- native desktop integration

  native GUI/toolkit compatibility = AWT flexibility (type 1/2 AWTs: Gtk, Xlib, ..)
- missing toolkit functionality

  consider lightweights (+ availability of AWT sources for native widget interfacing)
- environment compatibility

  "glibc" - problem, taming the combinations $\frac{n!}{k! * (n - k)!}$

  - portable VM (+libs), factor out target system dependencies

- GPL sources, automated configure/build process

Kaffe to the Desktop - Myths

- Java is not free (no sources), requires a license

  Kaffe is GPLed (with complete sources). Kaffe has been rebuilt completely from the specs (proof of suitability)

- Java is too slow (interpreter)

  Kaffe has JIT (see performance and WAT (by means of Cygnus GCJ). We (all) can still make it even faster (GC, libs,..).

- Java AWT needs Motif

  Kaffe has Xlib (lightweight widgets) and GTK (native widgets) AWTs

- It's just for fun, why to use a new environment

  Can we really afford this? Ramification can be dangerous (what is the deal to publish something that will be unnoticed).

Kaffe calling.. Embedded Systems

- isolated, stand-alone hosts, workstations, PCs are "out", the current Web is just the advent of inter-networked devices
- HW versatility - just the sky is the limit (smartcards, phones, PDAs, internet terminals, settop boxes, embedded controllers, workstations, cars, aircrafts?, space crafts ??)

  most of these systems have one thing in common: nothing
- HW may have a short life cycle - writing a development system that is used just for a single application is way too expensive! ("gcc vs. MUA", smartcard example)

⇩

strong need for a common development / runtime environment

if it is going to be Java, it has to be:

- Portable

  Kaffe has a reasonably small source tree (<1 MB for a given target system), well separated target-specific components (thread-, system call- interfaces)
- Scalable

  Kaffe can be easily configured to use/omit: JIT/interpreter, JNI, math, zip/jar support, native threads, (Type 3) stand-alone AWTs, ..

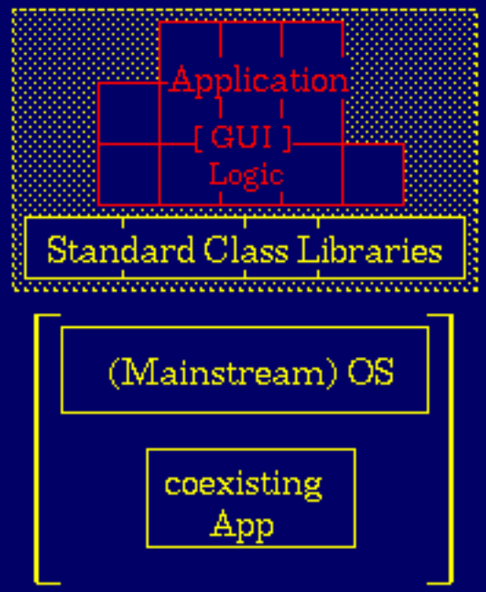
"Between Tcl and Oberon"

- How does Java compare to other (app programming) environments with respect to: ease-of-use, versatility, platform integration ?

Tcl/Tk

- ✚ GUI impl effort
- ✚ coexistence
- ▭ hybrid App model
- ▭ GUI extensibility
- ▭ stand–alone

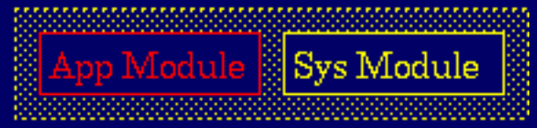
Java



"Middleware OS"

- ✚ binary compatibility
- ✚ Scalability (GUI/text, Script/App)
- ✚ coexistence
- ✚ stand–alone (opt)
- ✚ efficiency (JIT)
- ✚ comprehensive libs
- ▭ resources (multiple VMs)

Oberon



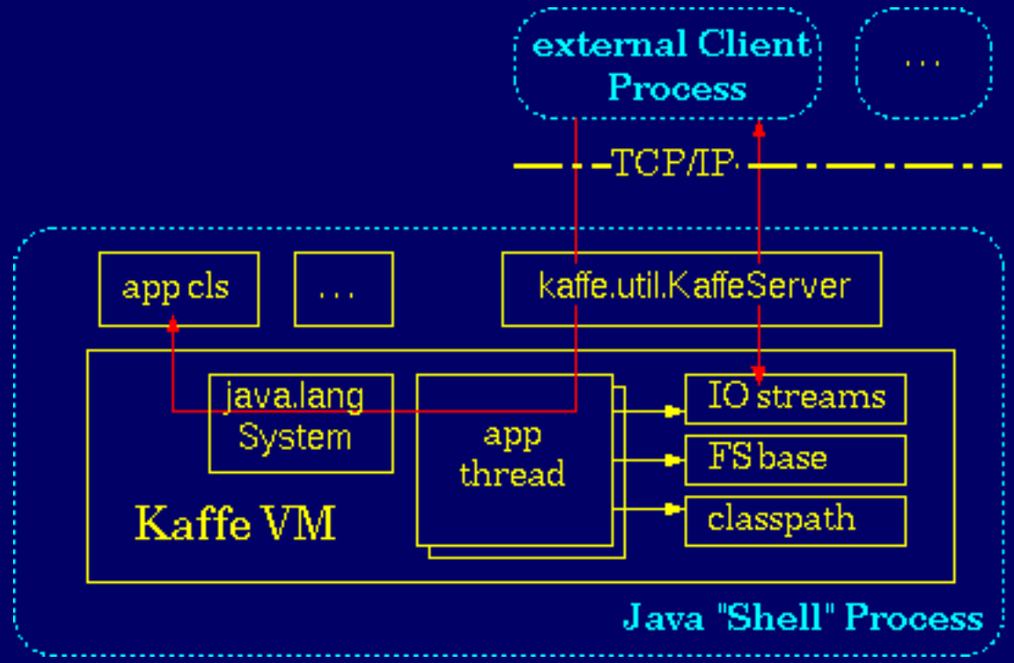
- ✚ efficiency
- ✚ OS "integration"
- ✚ resources
- ▭ coexistence
- ▭ portability

Kaffe Server

- no matter how far we optimize: VM needs a lot of resources
- if you can't shrink it -> share it:

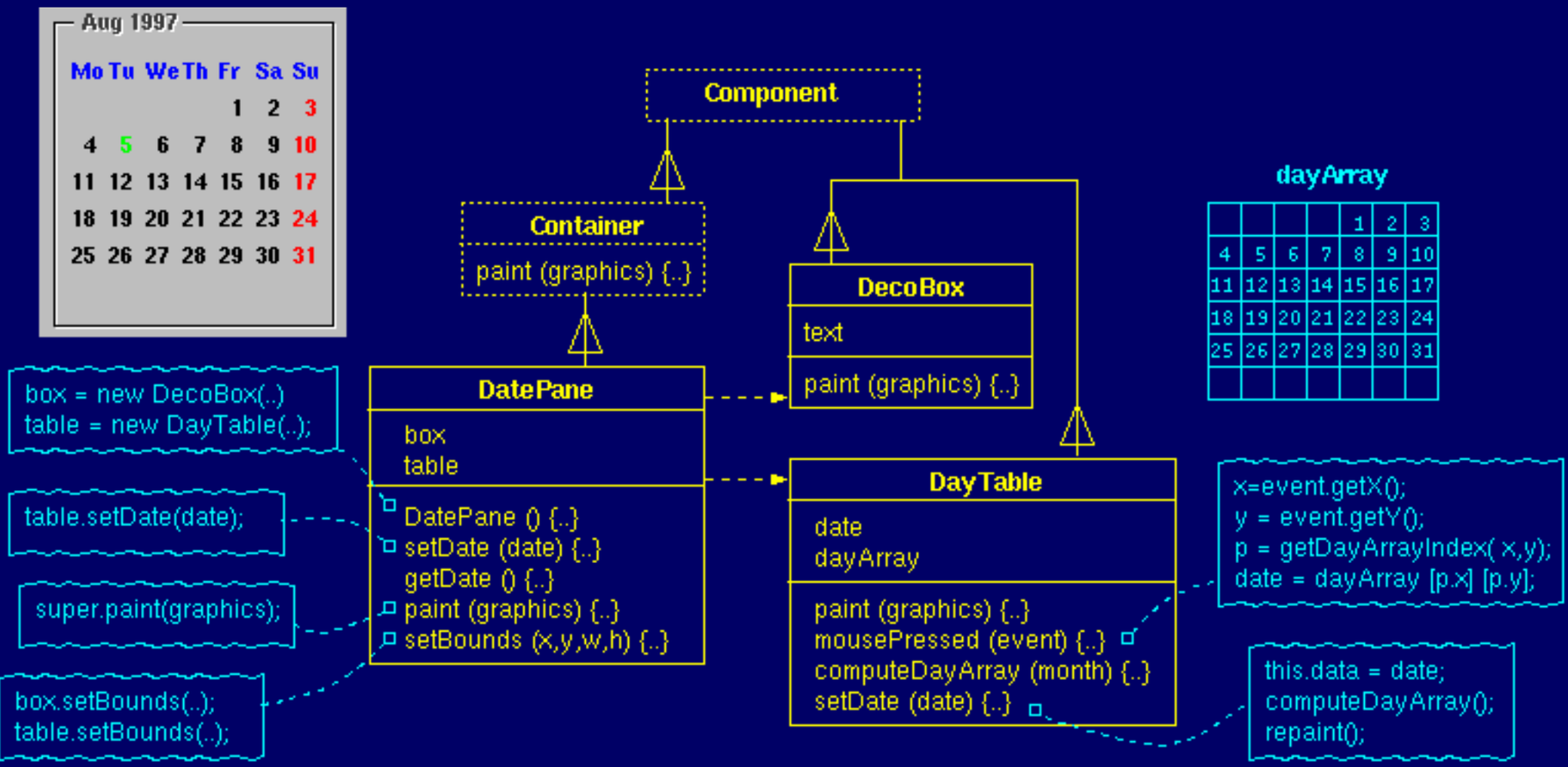
Java as a portable, graphical application shell

- this is NOT just for "scripts", but full fledged applications
- think of the impact on "IPC" (comes for free, incl. synchronization)



Lightweight Widget Example

- **DatePane** example (from biss-awt): mini-button panel to select dates
- contains **DayTable** widget (workhorse), surrounded by **DecoBox**
- acts as Facade, Mediator, Composite
- almost no toolkit pitfalls ("super.paint(..)")


Aug 1997
Mo Tu We Th Fr Sa Su
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
Component
Container
paint (graphics) {..}
DecoBox
text
paint (graphics) {..}
dayArray
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
box = new DecoBox(..)
table = new DayTable(..);
DatePane
box
table
DatePane () {..}
setDate (date) {..}
getDate () {..}
paint (graphics) {..}
setBounds (x,y,w,h) {..}
DayTable
date
dayArray
paint (graphics) {..}
mousePressed (event) {..}
computeDayArray (month) {..}
setDate (date) {..}
table.setDate(date);
super.paint(graphics);
box.setBounds(..);
table.setBounds(..);
x=event.getX();
y = event.getY();
p = getDayArrayIndex(x,y);
date = dayArray [p.x] [p.y];
this.data = date;
computeDayArray();
repaint();

Lightweight Example: DecoBox

```
class DecoBox extends Component
{
  String Text;

  public void setText ( String text ) {
    Text = text;
  }

  public void paint ( Graphics g ) {
    Dimension size = getSize();
    size.width -=2; size.height -= 7;

    g.setColor( Color.white);
    g.drawRect( 1, 6, size.width, size.height);
    g.setColor( Color.black);
    g.drawRect( 0, 5, size.width, size.height);

    if ( Text != null ) {
      FontMetrics fm = getToolkit().getFontMetrics( getFont());
      int w = fm.stringWidth( Text);
      g.setColor( getBackground());
      g.fillRect( 15, 5, w + 6, 3);
      g.setColor( Color.black);
```

```
      g.drawString( Text, 18, 10);
    }
  }

  public boolean contains ( int x, int y ){
    return false; // we are just a decoration
  }
}
```

Lightweight Example: DatePane

```
public class DatePane extends Container
{
  DecoBox         Box = new DecoBox();
  DayTable        Table = new DayTable( this);
  ActionListener  Client;
  static String[] Months = {"Jan","Feb","Mar","Apr","May","Jun",
                            "Jul","Aug","Sep","Oct","Nov","Dec"};

  public DatePane ( Date d ) {
    add( Box);
    add( Table);
    setDate( d);
  }

  public void addActionListener ( ActionListener client ) {
    Client = AWTEventMulticaster.add( Client, client);
  }

  public Date getDate () {
    return Table.getDate();
  }

  public void setDate ( Date dt ) {
```

```
    String mon = Months[ dt.getMonth()] + ' ' +
                 Integer.toString( 1900 + dt.getYear());
    Box.setText( mon);
    Table.setDate( dt);
  }

  public void doLayout () {
    Dimension size = getSize();
    Box.setBounds( 0, 0, size.width, size.height);
    Table.setBounds( 15, 15, size.width-30, size.height-30);
  }

  void notifyClients () {
    if ( Client != null ) {
      ActionEvent e = new ActionEvent( this,
                              ActionEvent.ACTION_PERFORMED,
"select");
      Client.actionPerformed( e);
    }
  }
}
```

Lightweight Example: DayTable

```
class DayTable extends Component implements MouseListener
{
  Date            CurDate;
  int[][]         DayArray;
  FontMetrics     Fm;
  DatePane        Master;
  static int[]    MonthLen = { 31,28,31,30,31,30,31,31,30,31,30,31
};
  static String[] DayShorts = {"Mo","Tu","We","Th","Fr","Sa","Su"
};

  public DayTable ( DatePane master ) {
    Font font = new Font( "Helvetica", Font.BOLD, 12);
    setFont( font);
    Fm = getToolkit().getFontMetrics( font);
    Master = master;
    addMouseListener( this);
  }

  public void mousePressed ( MouseEvent e ) {
    Dimension size = getSize();
    int x = e.getX();
    int y = e.getY();
```

```
    int cw  = 2*Fm.charWidth( '0');
    int cd  = (size.width-7*cw)/8;
    int rh  = Fm.getHeight();
    int rd  = (size.height-7*rh)/8;
    int xIdx = Math.min( x / (cw+cd), 6);
    int yIdx = Math.min( (y-rh-rd) / (rh+rd), 5 );
    int dn;
    Graphics g;

    if ( (dn=DayArray[xIdx][yIdx]) != 0) {
      CurDate.setDate( dn);
      if ( (g = getGraphics()) != null ) {
        redraw( g);
        g.dispose();
      }
    }

    Master.notifyClients();
  }

  public void redraw ( Graphics g) {
    if ( CurDate == null ) return;

    Dimension size = getSize();
    int zw  = Fm.charWidth( '0');
    int cw  = 2*zw;
    int cd  = (size.width-7*cw)/8;
```

```
    int dc  = Fm.getDescent();
    int rh  = Fm.getHeight();
    int rd  = (size.height-7*rh)/8;
    int dn, dof, dim = CurDate.getDate();

    g.setColor( Color.blue);
    g.setFont( getFont() );
    for ( int dy=0; dy<7; dy++)
      g.drawString( DayShorts[dy], cd+dy*(cw+cd), rd+rh-dc);

    for ( int dy=0; dy<7; dy++) {
      for ( int wk=0; wk<6; wk++) {
        if ( (dn=DayArray[dy][wk]) != 0 ){
          if ( dn<10) dof=zw;
          else        dof = 0;
          if (dn==dim) g.setColor( Color.green);
          else if ( dy==6) g.setColor( Color.red);
          else g.setColor( Color.black);
          g.drawString( Integer.toString(dn),
                        dof+cd+dy*(cd+cw), (wk+2)*(rd+rh)-dc );
        }
      }
    }
  }

  public void paint ( Graphics g ) {
    Dimension size = getSize();
```

```
    g.setColor( getBackground());
    g.fillRect( 0, 0, size.width, size.height);

    redraw( g);
  }

  public static int[][] getDayArray ( int mn, int ye) {
    int[][] darr = new int[7][6];
    int sd = (new Date( ye, mn-1, 1)).getDay();
    int maxd = MonthLen[mn-1];

    if ( (mn==2) && (isLeapYear( ye)) ) maxd++;
    if ( sd == 0) sd = 7;

    for( int day=sd-1; day < maxd+sd-1; day++)
      darr[day%7][day/7] = day-sd+2;

    return darr;
  }

  public Date getDate () {
    return CurDate;
  }

  void setDate ( Date dt ) {
    CurDate = dt;
    DayArray = getDayArray( dt.getMonth()+1, dt.getYear() );
```

```
    repaint();
  }

  public void mouseClicked ( MouseEvent e ) {}
  public void mouseReleased ( MouseEvent e ) {}
  public void mouseEntered ( MouseEvent e ) {}
  public void mouseExited ( MouseEvent e ) {}

  public static boolean isLeapYear ( int year) {
    if ( year % 400 == 0 )      return true;
    if ( year % 100 == 0 )      return false;
    return (year % 4) == 0;
  }
}
```