ProGuard

Optimizer and Obfuscator for Android

Eric Lafortune
Saikoa

Eric Lafortune

- 1991 – 1996 K.U.Leuven (Belgium), Phd Eng CompSci
- 1996 – 1999 Cornell University (Ithaca, NY)
- 1999 – 2011 Java GIS
- 2012 Founder Saikoa

Maybe more importantly:

- 1982 TMS-9900 processor
- 1995 ARM2/ARM3 processor
- 2001 Java bytecode
- 2010 Dalvik bytecode

ProGuard

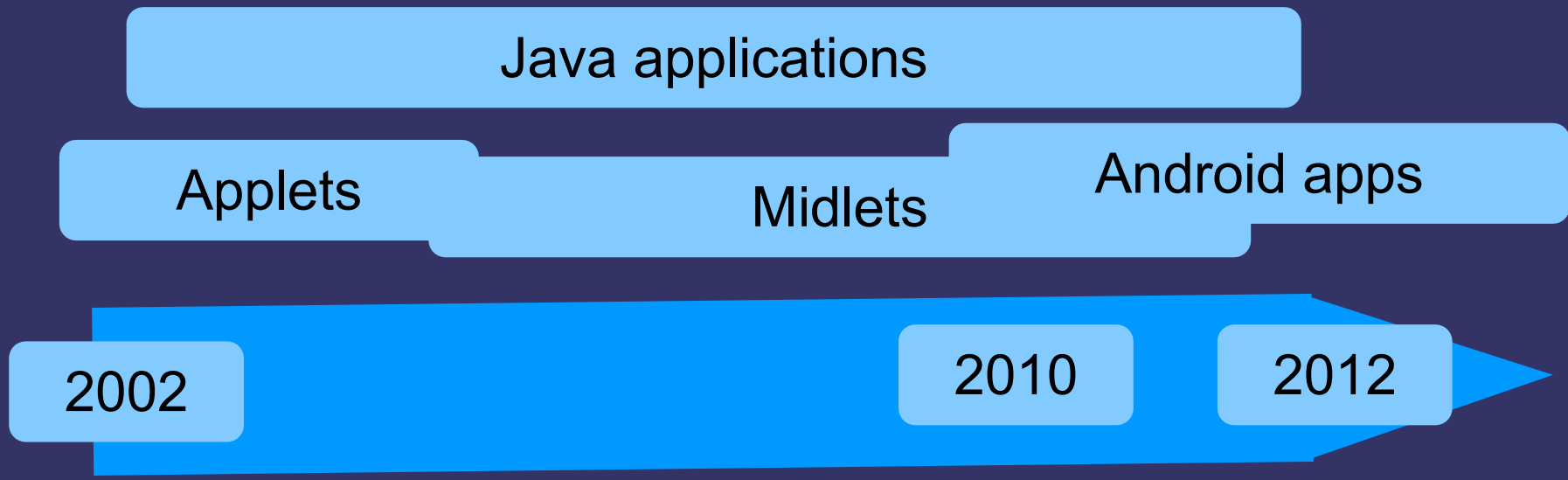
Open source

Generic

Shrinker
Optimizer
Obfuscator

For Java bytecode

ProGuard history



- May 2002 First download!
- Sep 2010 Recommended for protecting LVL
- Dec 2010 Part of Android SDK
- Jan 2012 Startup Saikoa

Why use ProGuard?

- Application size
- Performance
- Remove logging, debugging, testing code
- Battery life
- Protection

Application size

	classes.dex size			.apk size		
	Without ProGuard	With ProGuard	Reduction	Without ProGuard	With ProGuard	Reduction
ApiDemos	716 K	482 K	33%	2.6 M	2.5 M	4%
ApiDemos in Scala*	~6 M	542 K	~90%	~8 M	2.5 M	~70%

* [Stéphane Micheloud, http://lampwww.epfl.ch/~michelou/android/library-code-shrinking.html]

Performance: CaffeineMark

Without ProGuard		With ProGuard	
Sieve score	= 6833	Sieve score	= 6666
Loop score	= 14831	Loop score	= 15473
Logic score	= 19038	Logic score	= 47840
String score	= 7694	String score	= 7717
Float score	= 6425	Float score	= 6488
Method score	= 4850	Method score	= 5229
Overall score	= 8794	Overall score	= 10436

Improvement: 18%

[Acer Iconia Tab A500, nVidia Tegra 2, 1.0 GHz, Android 3.2.1]

Battery life

Extreme example:

"5 x better battery life,
by removing verbose logging code
in a background service"

(but don't count on it)

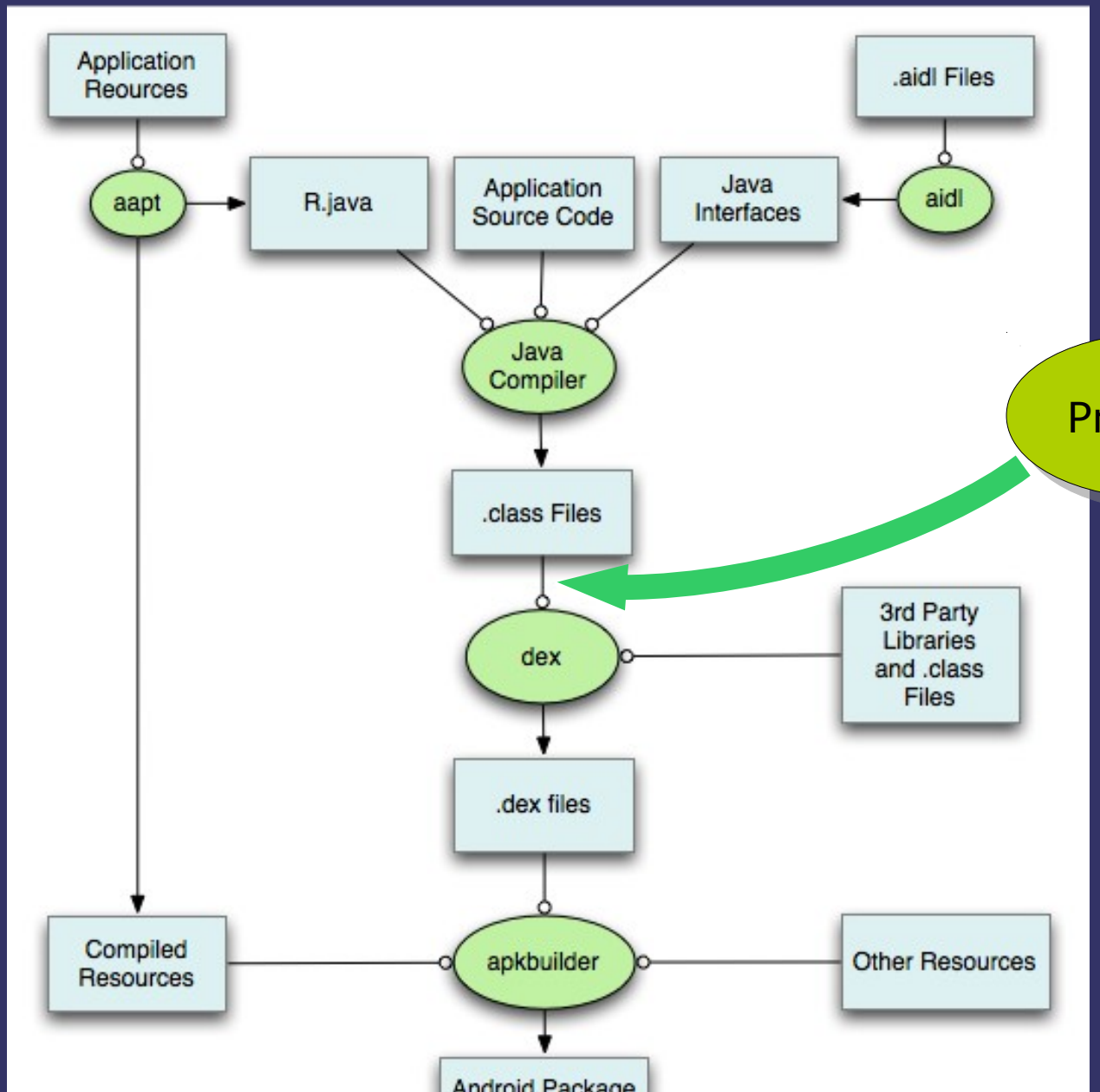
How to enable ProGuard?

project.properties:

```
# To enable ProGuard to shrink and obfuscate
    your code, uncomment this
#proguard.config=
    ${sdk.dir}/tools/proguard/proguard-android.txt:
    proguard-project.txt
```

→ only applied when building release versions

Build process



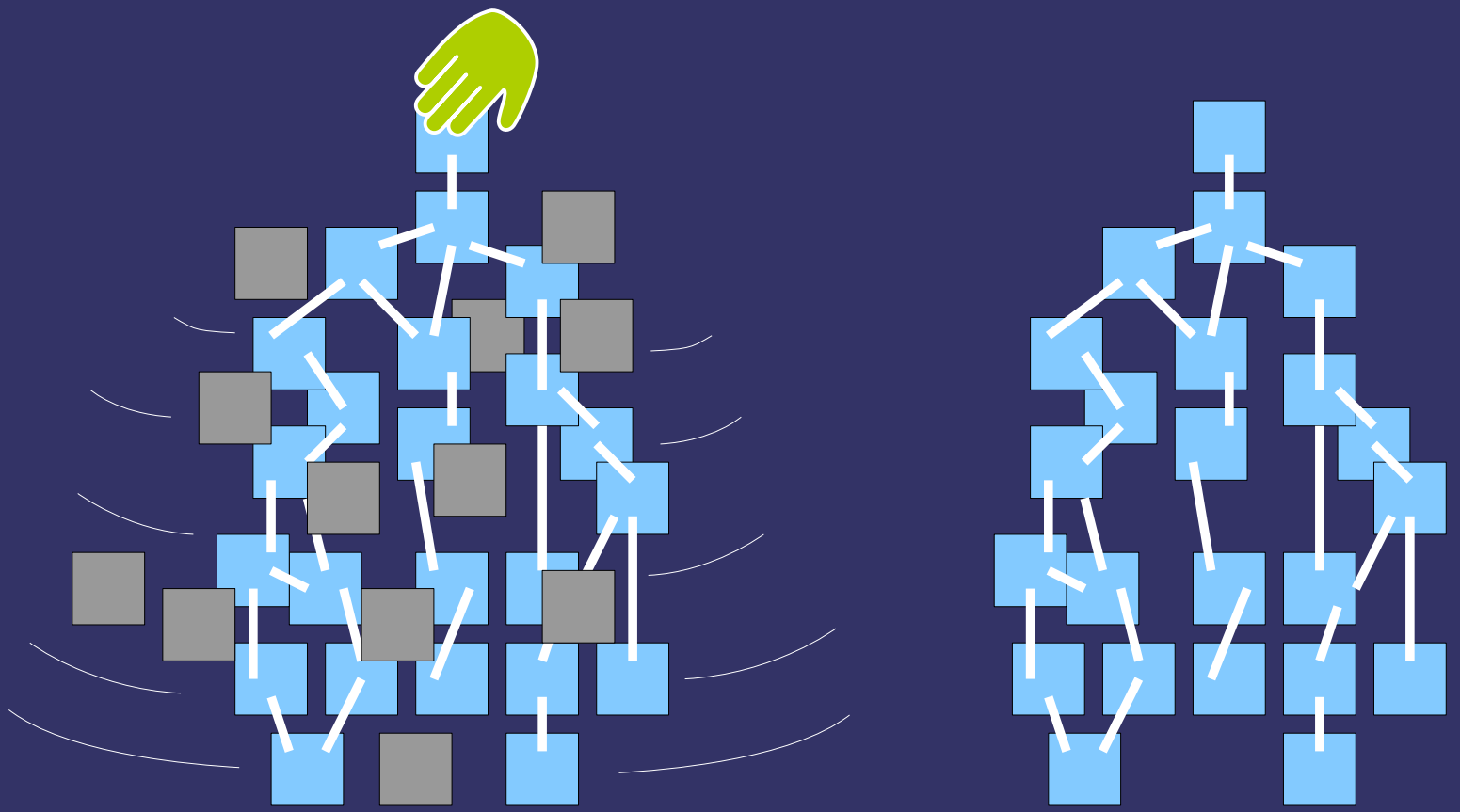
[Android documentation]

Shrinking

Also called treeshaking, minimizing, shrouding

Shrinking

- Classes, fields, methods



Entry points

1) Activities, applications, services, fragments,...

→ provided automatically by Android build process

```
-keep public class * extends android.app.Activity
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
…
```

Entry points

2) Introspection, e.g. Google License Verification Library

→ must be specified in proguard-project.txt

```
-keep public interface
    com.android.vending.licensing.ILicensingService
```

Entry points

More introspection, e.g. Google vending library

→ must be specified in proguard-project.txt

```
-keepclassmembers public class
   com.google.android.vending.expansion.downloader.impl.DownloadsDB$* {
    public static final java.lang.String[][] SCHEMA;
    public static final java.lang.String     TABLE_NAME;
}
```

Entry points

More introspection, e.g. Guice, RoboGuice

→ must be specified in proguard-project.txt:

```
-keepclassmembers class * {
    @javax.inject.**           <fields>;
    @com.google.inject.**      <fields>;
    @roboguice.**              <fields>;
    @roboguice.event.Observes <methods>;
}
```

Notes and warnings

"Closed-world assumption"

```
Warning: twitter4j.internal.logging.Log4JLoggerFactory:
   can't find referenced class org.apache.log4j.Logger
Warning: twitter4j.internal.logging.SLF4JLoggerFactory:
   can't find referenced class org.slf4j.LoggerFactory
...
```

```
Warning: com.dropbox.client2.DropboxAPI:
  can't find referenced class org.json.simple.JSONArray
```

→ if debug build works fine,
then ok to ignore in proguard-project.txt:

```
-dontwarn twitter4j.internal.logging.**
-dontwarn com.dropbox.client2.**
```

Optimization

At the bytecode instruction level:

- Dead code elimination
- Constant propagation
- Method inlining
- Class merging
- Remove logging code
- Peephole optimizations
- Devirtualization
- ...

Dead code elimination

```
boolean debug = false;
…

if (debug) Log.v(“.....”);
…
```

Note: showing equivalent source code instead of bytecode

Dead code elimination

```
boolean debug = false;
…

if (debug) Log.v("....");
…
```

Note: showing equivalent source code instead of bytecode

Constant propagation

Inside methods:

```
int f1 = 6;
…
int f2 = 7;
…
int answer = f1 * f2;
```

Constant propagation

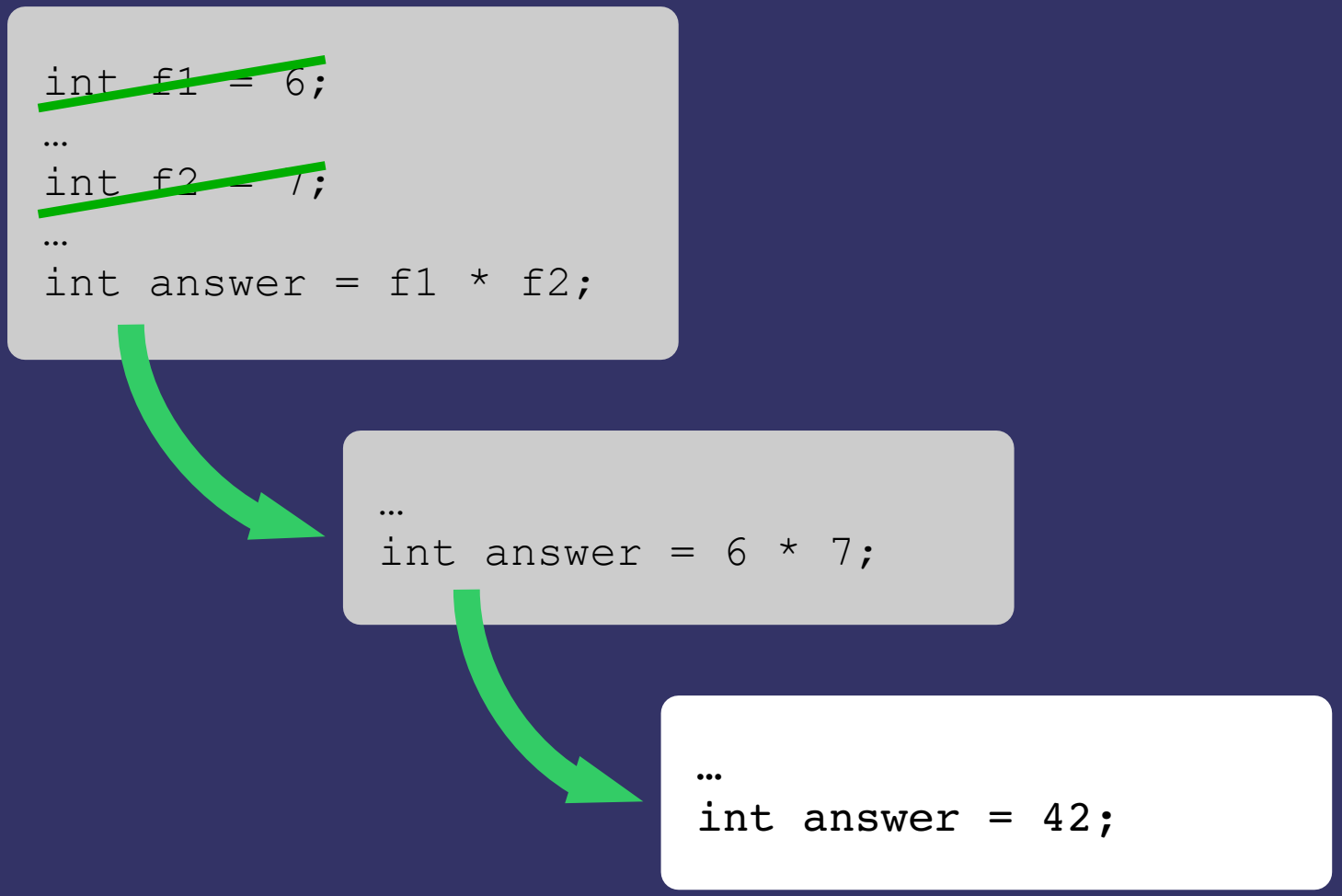
Inside methods:

```
int f1 = 6;
…
int f2 = 7;
…
int answer = f1 * f2;
```

```
…
int answer = 6 * 7;
```

Constant propagation

Inside methods:



Constant propagation

Across methods:

```
int answer = computeAnswer(6, 7);
```

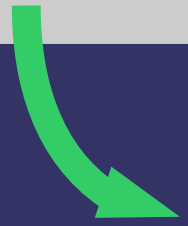
```
int computeAnswer(int f1, int f2) {
    return f1 * f2;
}
```

Constant propagation

Across methods:

```
int answer = computeAnswer(6, 7);
```

```
int computeAnswer(int f1, int f2) {
    return f1 * f2;
}
```



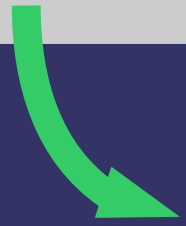
```
int computeAnswer(int f1, int f2) {
    return 6 * 7;
}
```

Constant propagation

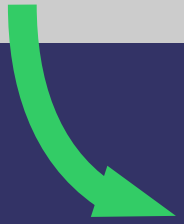
Across methods:

```
int answer = computeAnswer(6, 7);
```

```
int computeAnswer(int f1, int f2) {
    return f1 * f2;
}
```



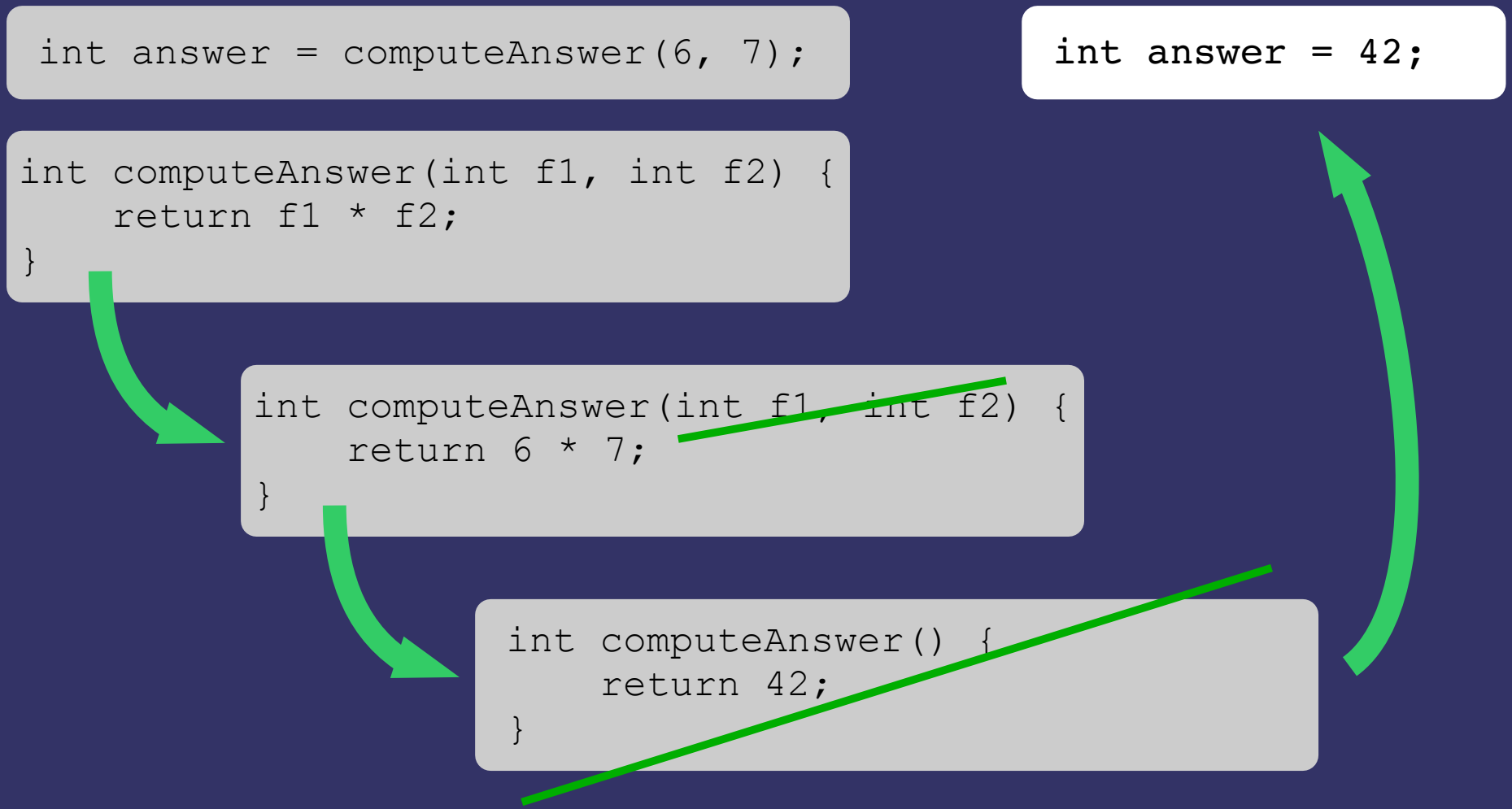
```
int computeAnswer(int f1, int f2) {
    return 6 * 7;
}
```



```
int computeAnswer() {
    return 42;
}
```

Constant propagation

Across methods:



Method inlining

Short methods (or methods that are only invoked once):

```
int answer = image.getPixel(i, j);
```

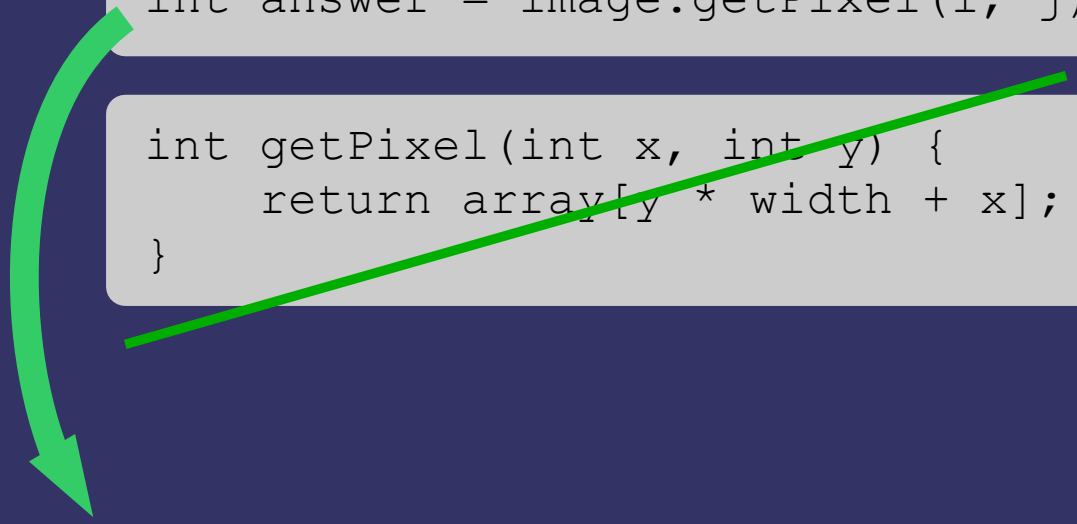
```
int getPixel(int x, int y) {
    return array[y * width + x];
}
```

Method inlining

Short methods (or methods that are only invoked once):

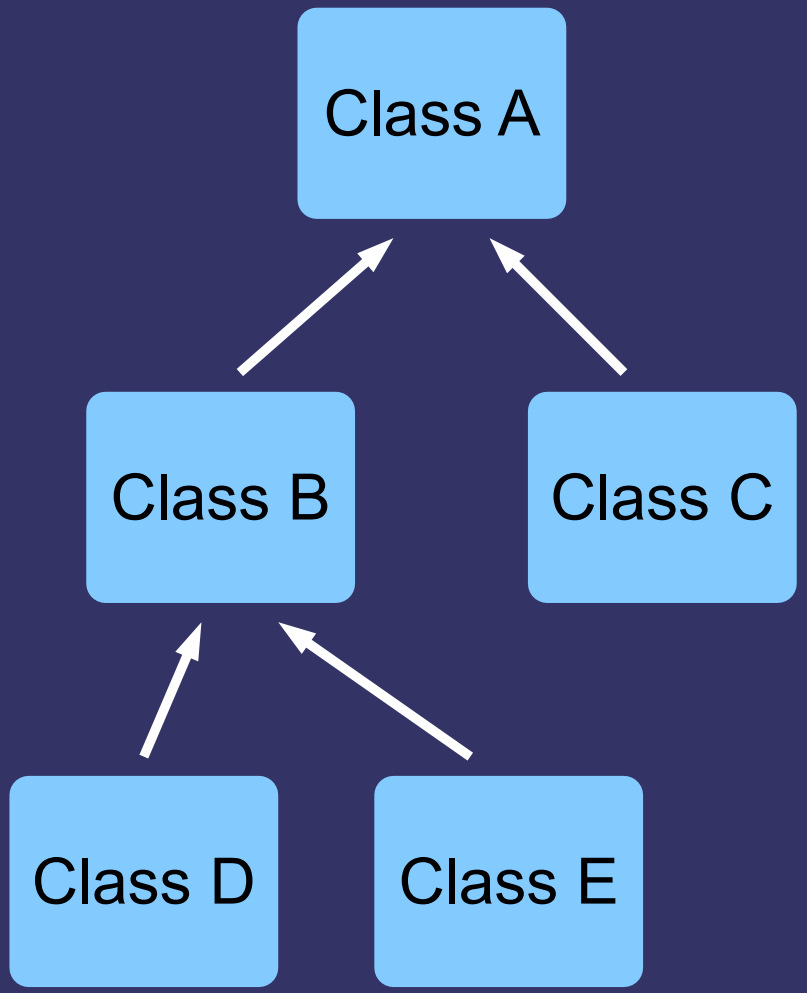
```
int answer = image.getPixel(i, j);
```

```
int getPixel(int x, int y) {
    return array[y * width + x];
}
```

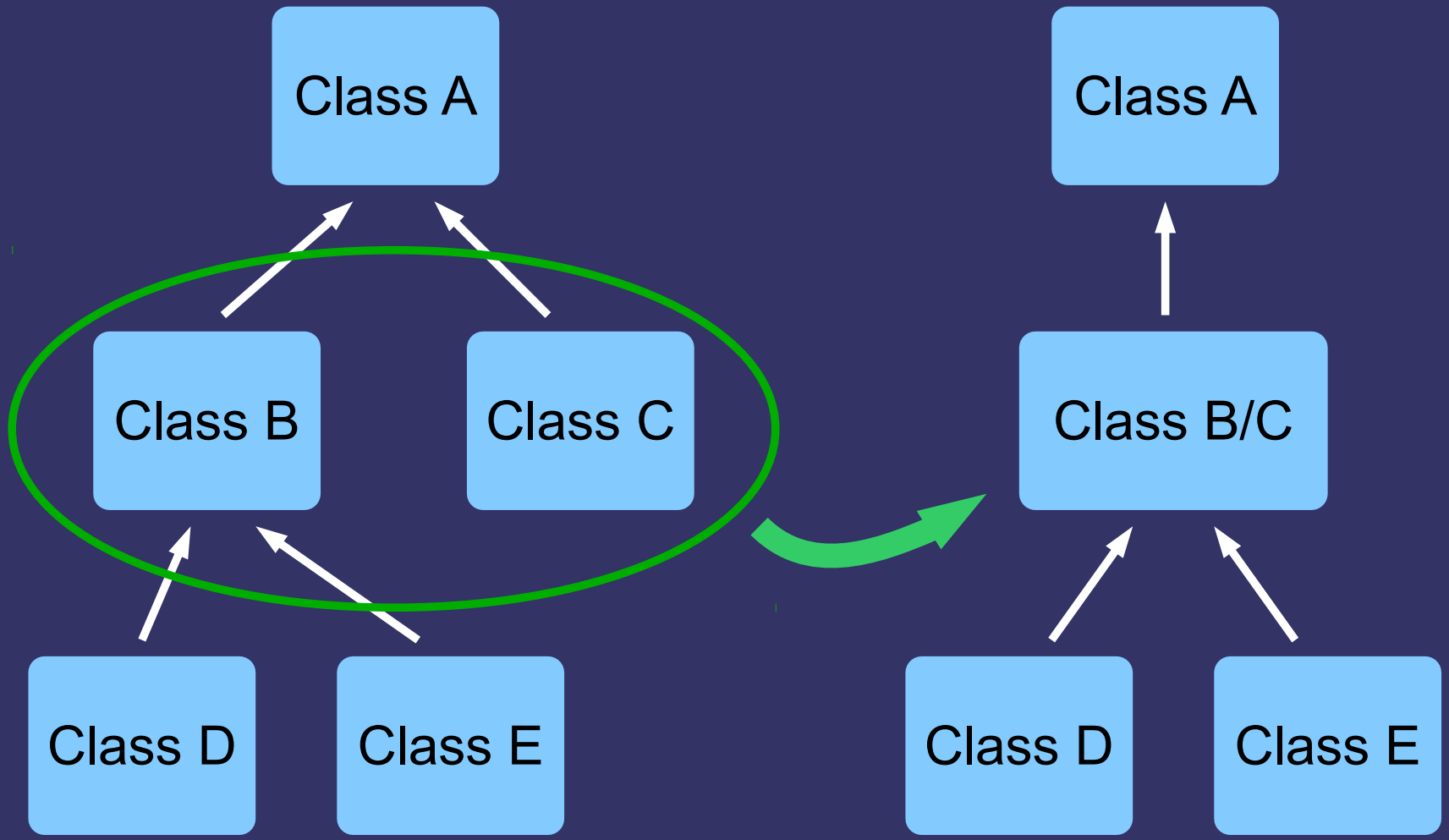


```
int answer = image.array[j * image.width + i];
```

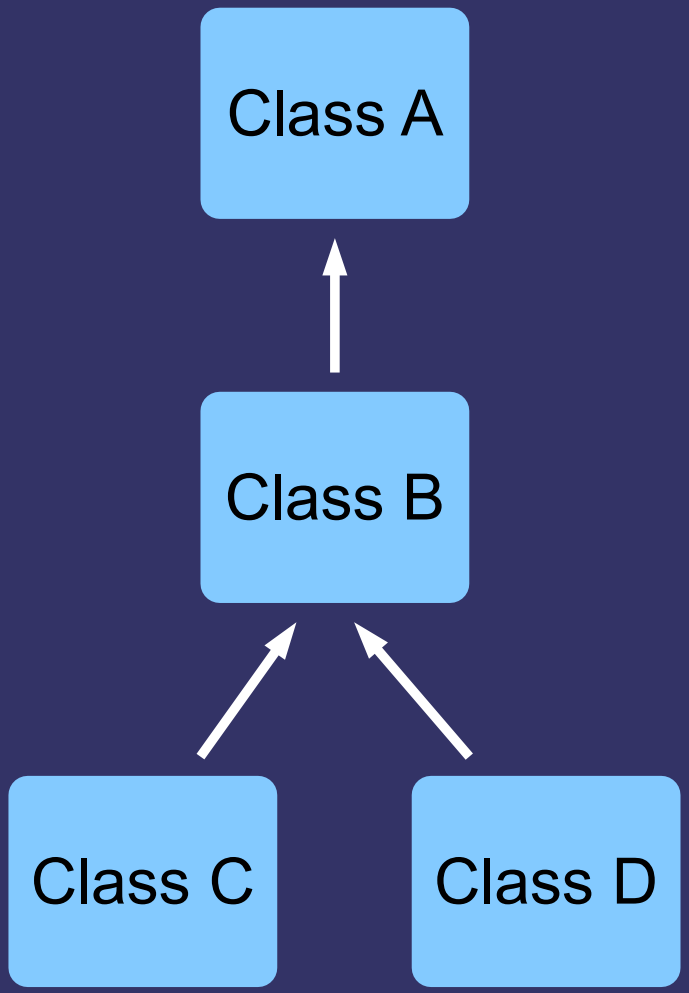
Class merging: horizontally



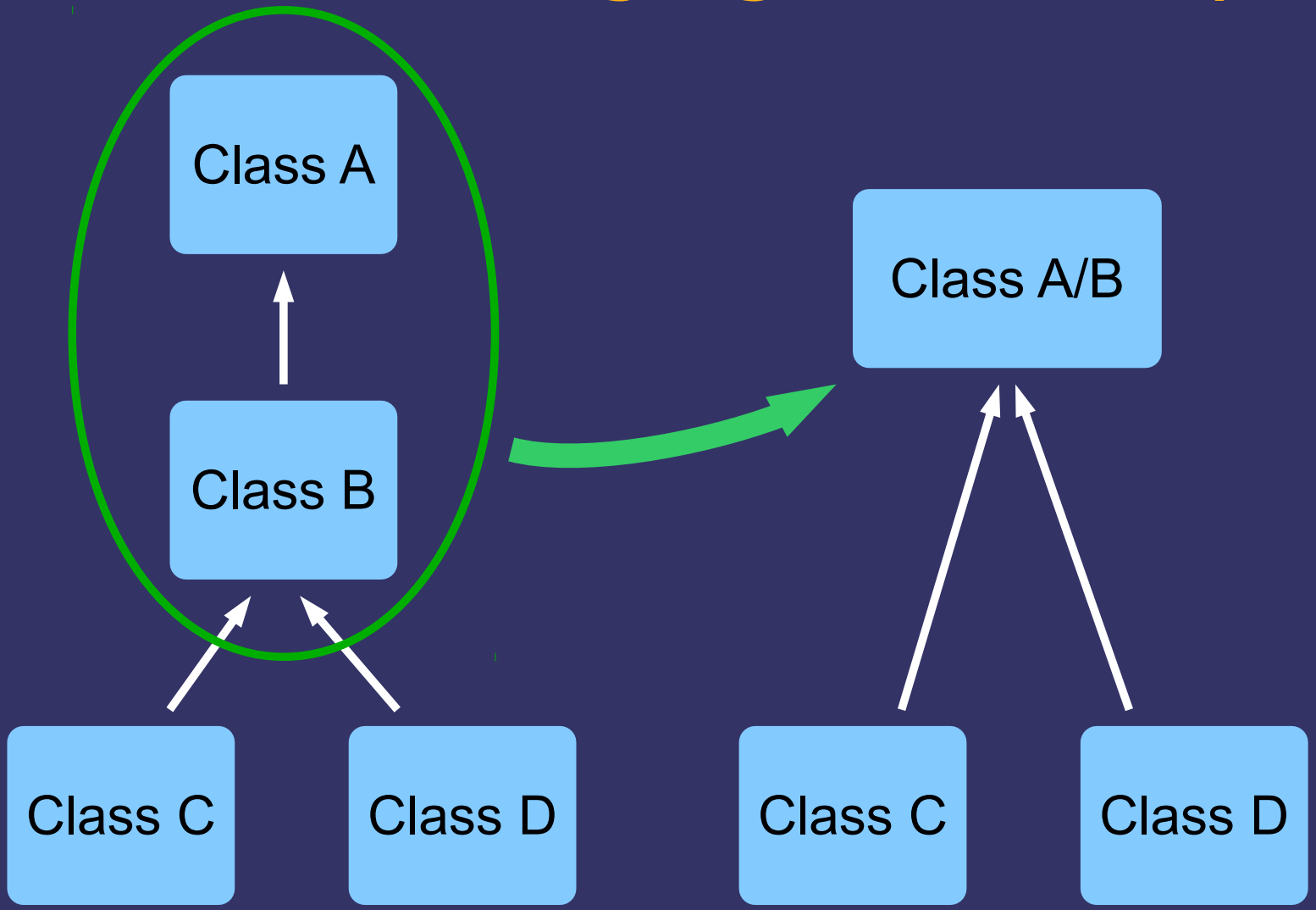
Class merging: horizontally



Class merging: vertically



Class merging: vertically



Tail recursion simplification

```
int answer = computeAnswer(1, 2, 3, 7);
```

```
int computeAnswer(int f1, int f2, int f3, int f4) {
    if (f2 == 1 && f3 == 1 && f4 == 1) {
        return f1;
    } else {
        return computeAnswer(f1 * f2, f3, f4, 1);
    }
}
```

Tail recursion simplification

```
int answer = computeAnswer(1, 2, 3, 7);
```

```
int computeAnswer(int f1, int f2, int f3, int f4) {
    if (f2 == 1 && f3 == 1 && f4 == 1) {
        return f1;
    } else {
        return computeAnswer(f1 * f2, f3, f4, 1);
    }
}
```

```
int computeAnswer(int f1, int f2, int f3, int f4) {
  do {
    if (f2 == 1 && f3 == 1 && f4 == 1) {
        return f1;
    } else {
        f1 = f1 * f2; f2 = f3, f3 = f4, f4 = 1;
    }
  } while (true);
}
```

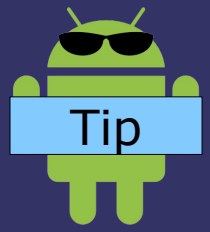
Tail recursion simplification

```
int answer = computeAnswer(1, 2, 3, 7);
```

```
int answer = 42;
```

```
int computeAnswer(int f1, int f2, int f3, int f4) {
    if (f2 == 1 && f3 == 1 && f4 == 1) {
        return f1;
    } else {
        return computeAnswer(f1 * f2, f3, f4, 1);
    }
}
```

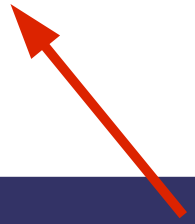
```
int computeAnswer(int f1, int f2, int f3, int f4) {
  do {
    if (f2 == 1 && f3 == 1 && f4 == 1) {
        return f1;
    } else {
        f1 = f1 * f2; f2 = f3, f3 = f4, f4 = 1;
    }
  } while (true);
}
```



How to enable optimization?

project.properties:

```
# To enable ProGuard to shrink and obfuscate
    your code, uncomment this
proguard.config=
    ${sdk.dir}/tools/proguard/proguard-android-optimize.txt:
    proguard-project.txt
```



Remove logging code

Specify assumptions in proguard-project.txt:

```
-assumenosideeffects class android.util.Log {
    public static boolean isLoggable(java.lang.String, int);
    public static int v(...);
    public static int i(...);
    public static int w(...);
    public static int d(...);
    public static int e(...);
    public static java.lang.String getStackTraceString
                                            (java.lang.Throwable);
}
```

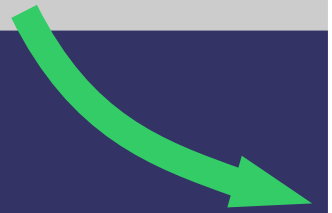
Obfuscation

Traditional name obfuscation:

- Rename identifiers:
  class/field/method names
- Remove debug information:
  line numbers, local variable names,...

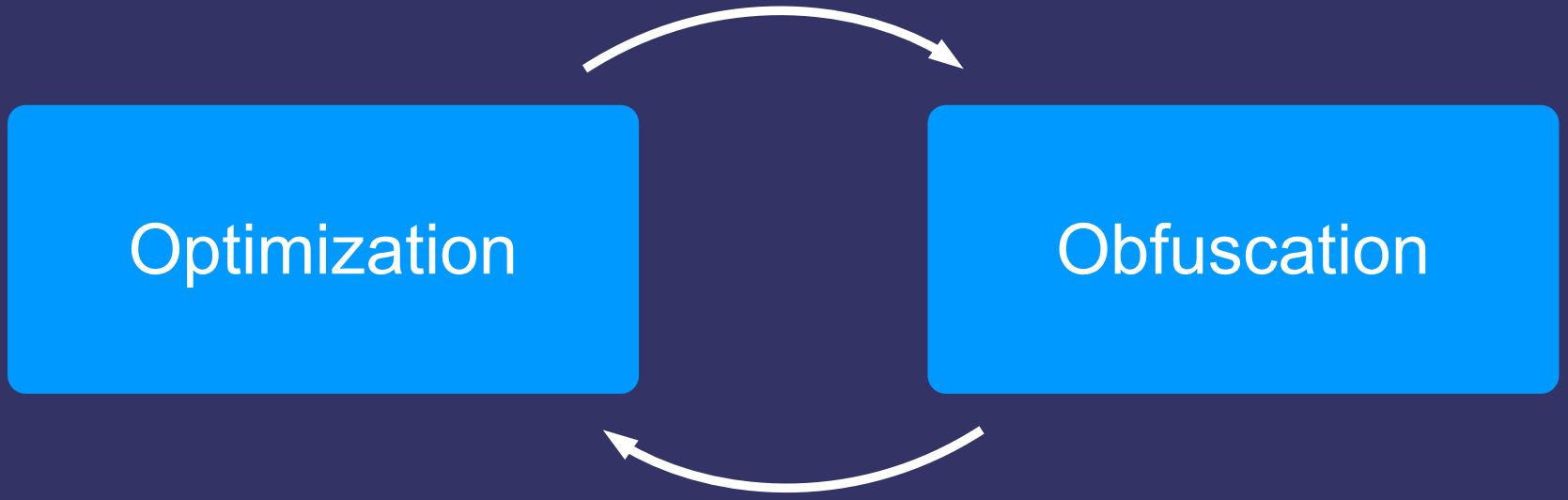
Obfuscation

```
public class MyComputationClass {
    private MySettings  settings;
    private MyAlgorithm algorithm;
    private int         answer;

    public int computeAnswer(int input) {
        …
        return answer;
    }
}
```



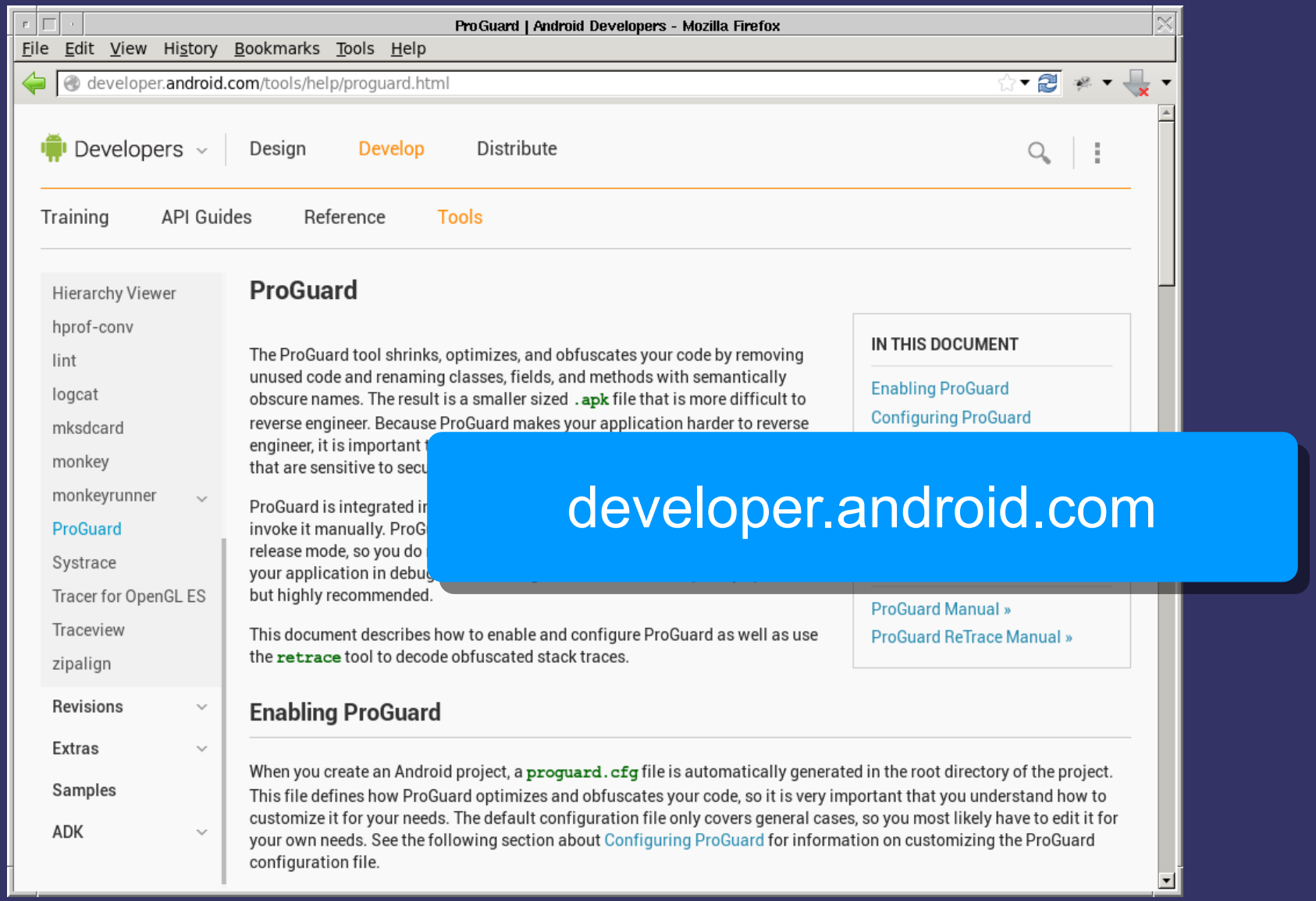
```
public class a {
    private b   a;
    private c   b;
    private int c;

    public int a(int a) {
        …
        return c;
    }
}
```

Complementary steps

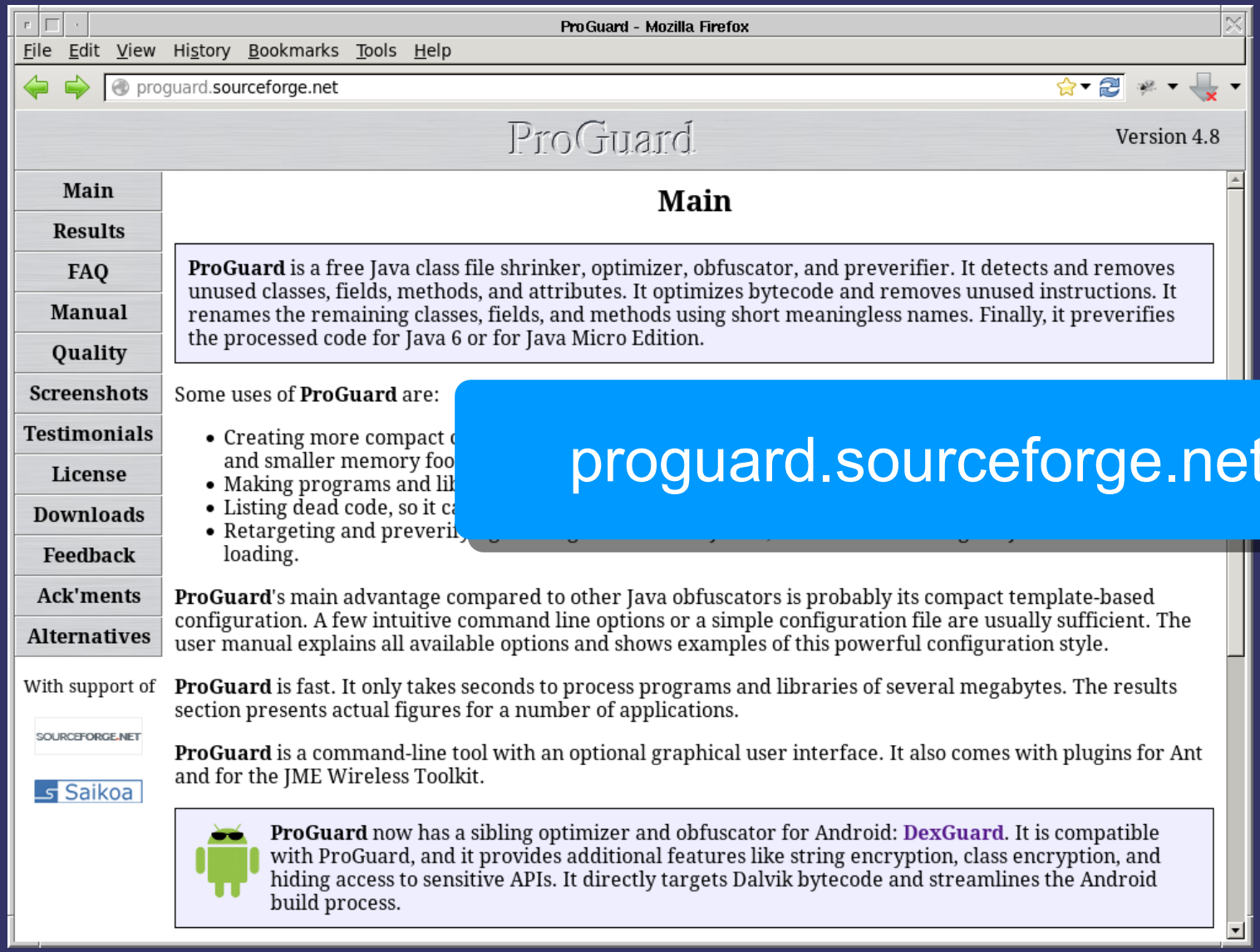


Irreversibly remove information

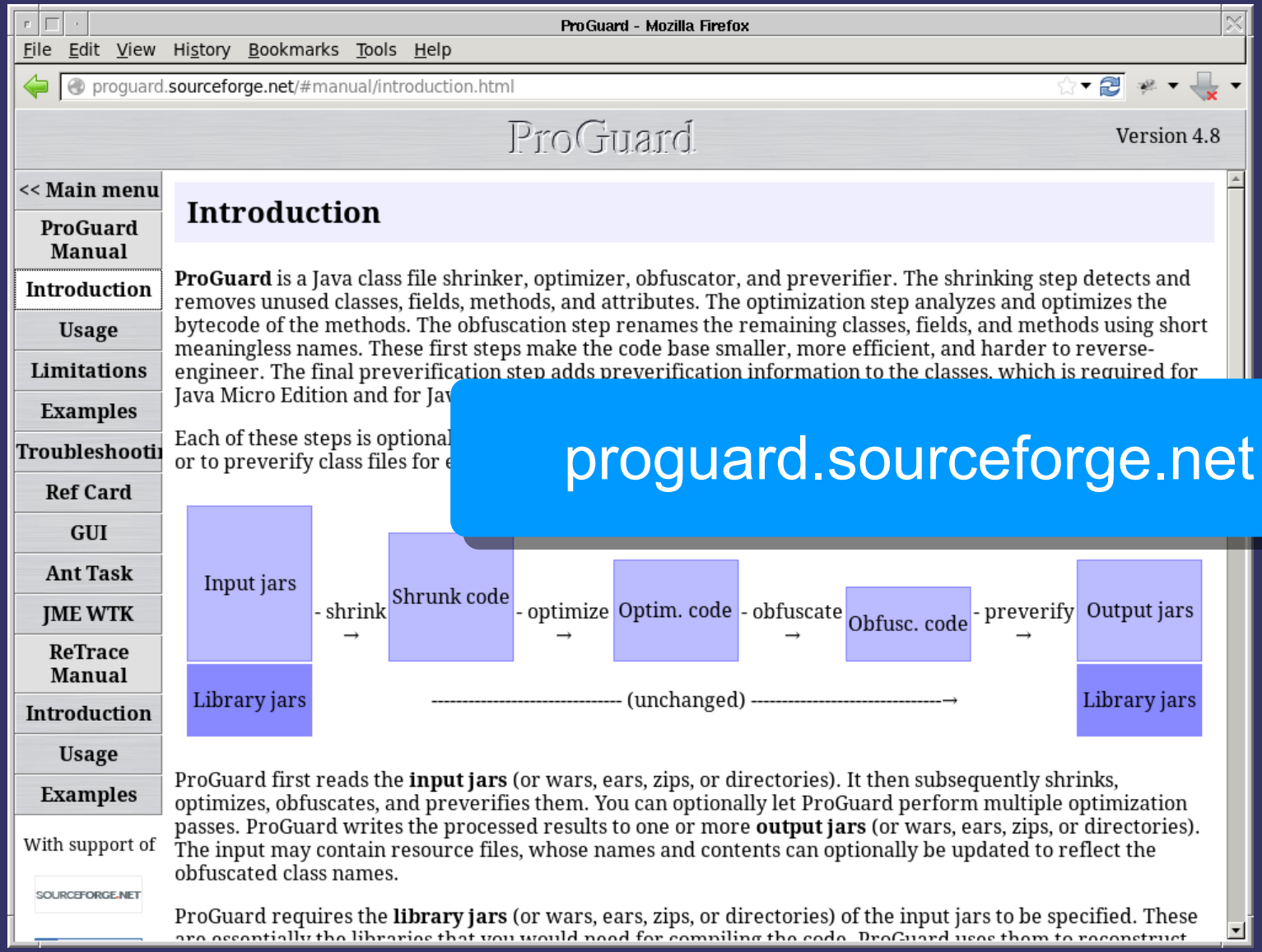
ProGuard guide – Android SDK



ProGuard website



ProGuard manual



Startup: Saikoa

- Open source: ProGuard
- Services: Professional ProGuard support
- Product: DexGuard

ProGuard - DexGuard

ProGuard	DexGuard
Open source	Closed source
Generic	Specialized
Shrinker Optimizer Obfuscator	Shrinker Optimizer Obfuscator Protector
For Java bytecode	For Android

Compatible

Motivations for hacking/cracking

- Anti-malware research
- Reverse-engineering protocols, formats,...
- Fun
- Translation
- Game cheating
- Software piracy
- Remove ads
- Different ads
- Different market
- Extortion
- Extract assets
- Extract API keys
- Insert malware (SMS,...)
- ...

Solutions?

- Ignore it
- Different business model (open source, service)
- Regular updates
- Lock down device
- Server
- Remove motivations
- Obfuscation, application protection

More application protection

Nothing is unbreakable, but you can raise the bar:

- Reflection
- String encryption
- Class encryption
- Tamper detection
- Debug detection
- Emulator detection
- …

→ Automatically applied by DexGuard

Saikoa website

Questions?

ProGuard

Open source

Shrinking
Optimization
Obfuscation

Saikoa

Java bytecode

Dalvik bytecode

DexGuard