

Java Extensions

distribution, transactions, persistence

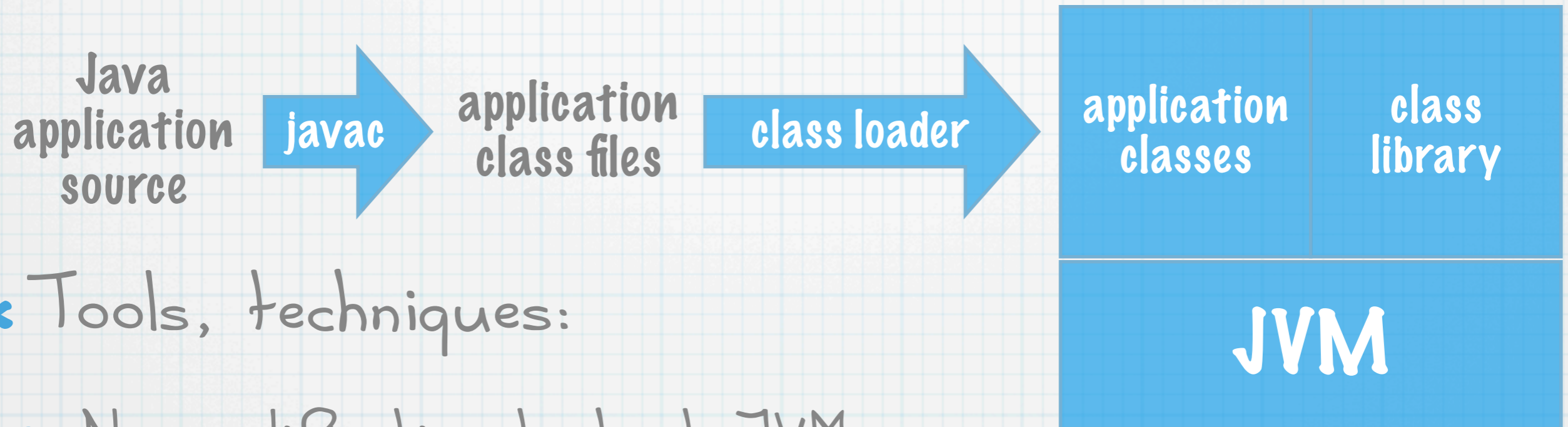
Tony Hosking - Purdue University

Joint work with

Eliot Moss, Trek Palmer, Asjad Khan - UMass

Athul Acharya, Phil McGachey - Purdue

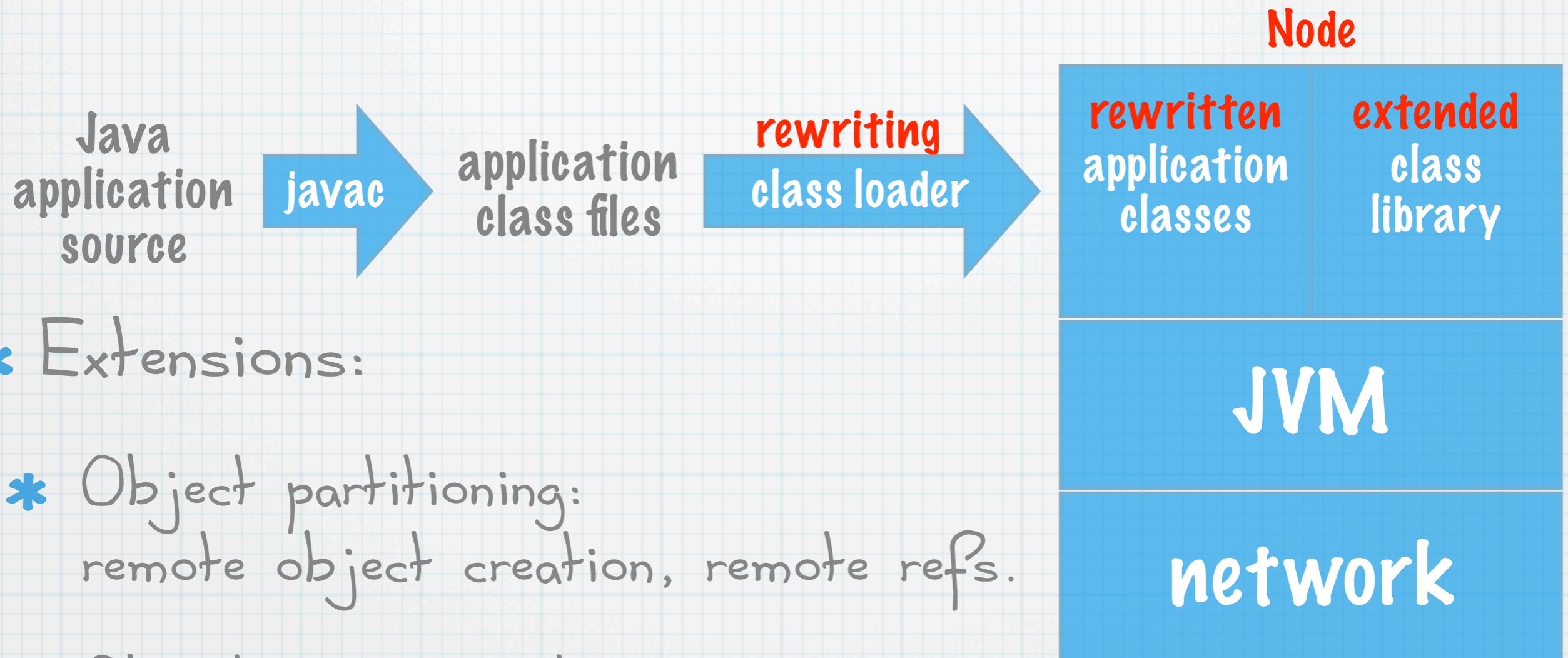
Common Infrastructure



* Tools, techniques:

- * No modification to host JVM
- * Class rewriting: Javassist and ASM
- * User-level class loaders
- * Twin class hierarchies

Transparent distribution



* Extensions:

- * Object partitioning:
remote object creation, remote refs.
- * Object management:
RMI, migration, replication, fault-tolerance
- * Network configuration: topology, listeners

Transparent distribution

- * Rewrites:

- * Class hierarchy

- * Member fields and methods

- * Bytecode substitution

- * Support code

Class rewrites

```
class Foo {  
    int f;  
    int m() {return f;}  
    static int sf;  
    static int sm() {return sf;}  
}
```

Class rewrites

```
class Foo {  
    int f;  
    int m() {return f;}  
    static int sf;  
    static int sm() {return sf;}  
}
```

```
interface Foo
```

```
m(): int
```

```
interface Foo$_$static
```

```
sm(): int
```

Class rewrites

```
class Foo {  
    int f;  
    int m() {return f;}  
    static int sf;  
    static int sm() {return sf;}  
}
```

```
interface Foo
```

```
m(): int
```

```
class Foo$_local
```

```
f: int
```

```
m(): int
```



```
interface Foo$_$static
```

```
sm(): int
```

```
class Foo$_$static_local
```

```
sf: int
```

```
sm(): int
```



Class rewrites

```
class Foo {  
    int f;  
    int m() {return f;}  
    static int sf;  
    static int sm() {return sf;}  
}
```

interface Foo

m(): int

interface Foo\$_\$static

sm(): int

class Foo\$_\$local

f: int

m(): int

class Foo\$_\$static_local

sf: int

sm(): int



Class rewrites

```
class Foo {  
  int f;  
  int m() {return f;}  
  static int sf;  
  static int sm() {return sf;}  
}
```

interface Foo

m(): int

class Foo\$_local

f: int

m(): int

class Foo\$_stub

m(): int

interface Foo\$_static

sm(): int

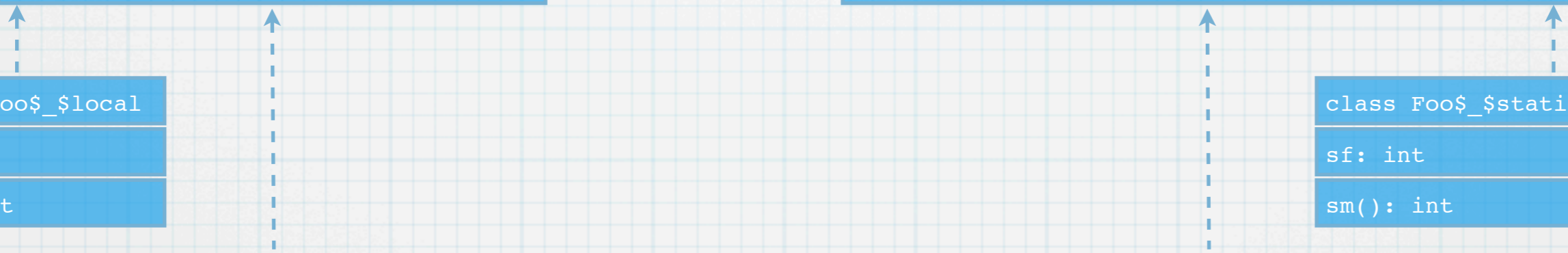
class Foo\$_static_local

sf: int

sm(): int

class Foo\$_static_stub

sm(): int



Class rewrites

```
class Foo {  
    int f;  
    int m() {return f;}  
    static int sf;  
    static int sm() {return sf;}  
}
```

interface Foo

m(): int

class Foo\$_local

f: int

m(): int

class Foo\$_stub

m(): int

interface Foo\$_static

sm(): int

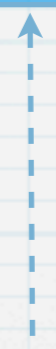
class Foo\$_static_local

sf: int

sm(): int

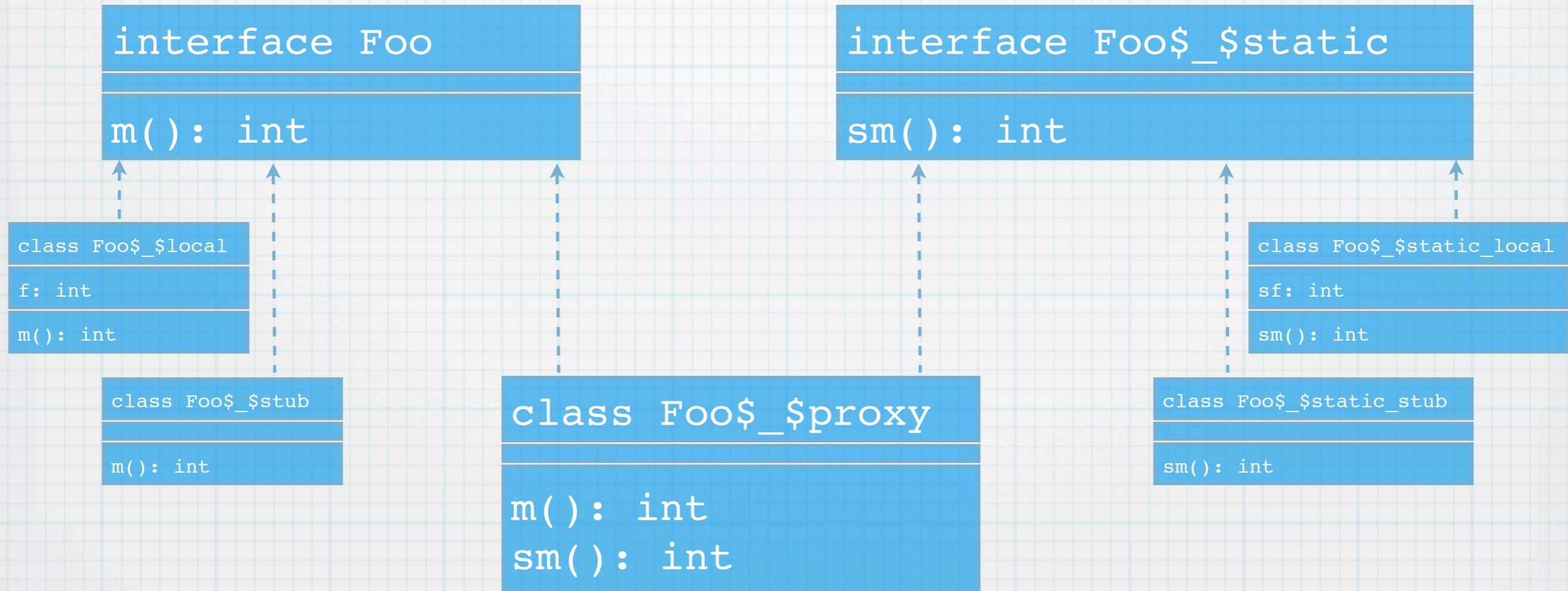
class Foo\$_static_stub

sm(): int



Class rewrites

```
class Foo {  
    int f;  
    int m() {return f;}  
    static int sf;  
    static int sm() {return sf;}  
}
```



Class rewrites

```
class Foo {  
  int f;  
  int m() {return f;}  
  static int sf;  
  static int sm() {return sf;}  
}
```

interface Foo

m(): int

class Foo\$_local

f: int

m(): int

class Foo\$_stub

m(): int

class Foo\$_proxy

m(): int

sm(): int

focusing on instance parts

Class rewrites

```
class Foo {  
    int f;  
    int m() {return f;}  
    static int sf;  
    static int sm() {return sf;}  
}
```

interface Foo

m(): int

class Foo\$_local

f: int

m(): int

class Foo\$_stub

uid: UID

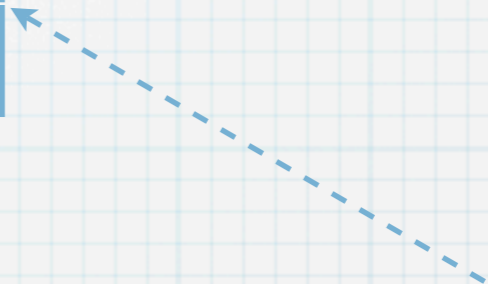
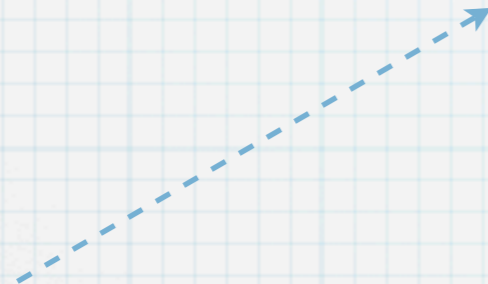
m(): int

class Foo\$_proxy

indirect: Foo

m(): int

sm(): int



Class rewrites

```
class Foo {  
    int f;  
    int m() {return f;}  
    static int sf;  
    static int sm() {return sf;}  
}
```

interface Foo

m(): int
get\$\$f(): int
set\$\$f(int): void
get_uid(): UID

class Foo\$_local

f: int

m(): int
get\$\$f(): int
set\$\$f(int): void
get_uid(): UID

class Foo\$_stub

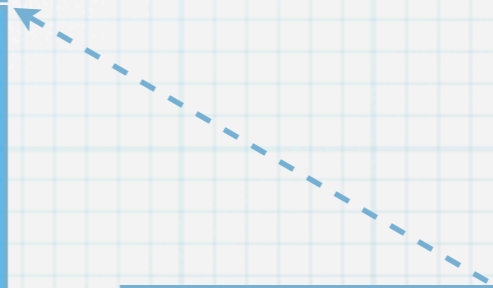
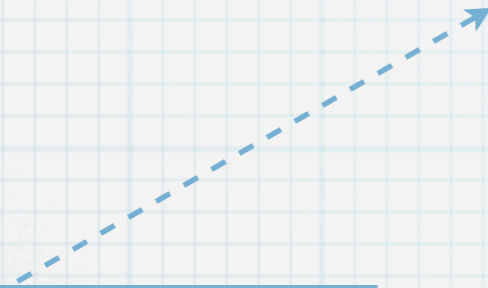
uid: UID

m(): int
get\$\$f: int
set\$\$f(int): void
get_uid(): UID

class Foo\$_proxy

indirect: Foo

m(): int
sm(): int
get\$\$f: int
set\$\$f(int): void
get_uid(): UID
redirect(Foo):void



Statics

- * Statics held in a singleton instance
- * Implements Singleton design pattern
- * Created when class first loaded

Bytecode rewrites

Original: `int m() {return f;}`

Foo\$_local:

```
return get$$f();
```

Foo\$_stub:

```
int node = Remote.find(uid);
```

```
Msg msg = new Msg(uid, "m", ...);
```

```
return msg.invoke(node);
```

Foo\$_proxy:

```
return indirect.m();
```

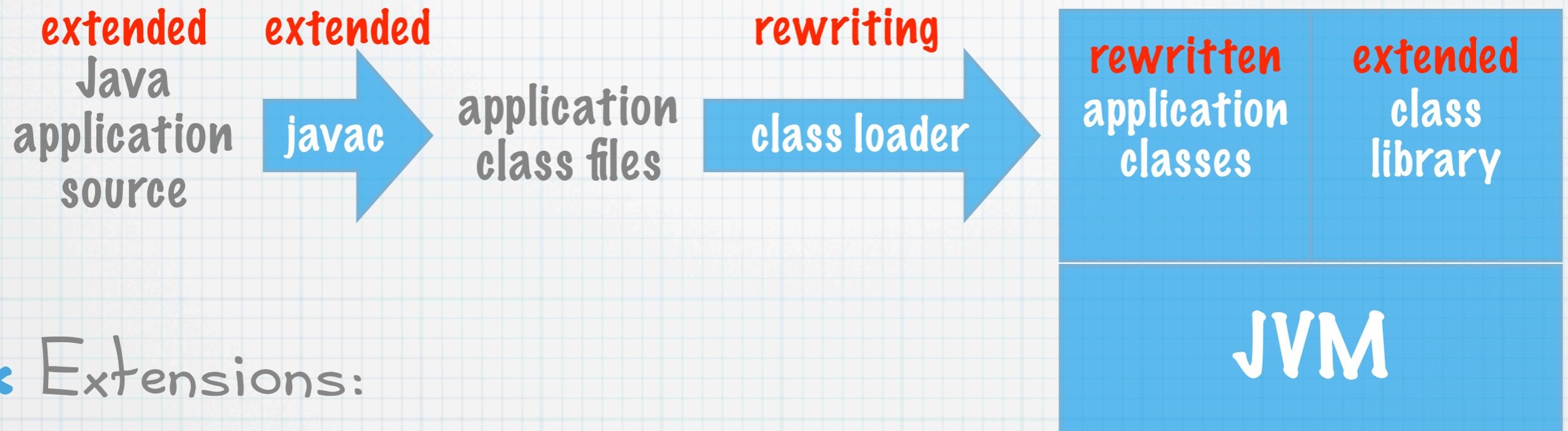
Allocation logic

- * `new Foo` rewritten to `new Foo$_$proxy`
- * `Foo$_$proxy` constructor dynamically determines `new Foo$_$local` or `new Foo$_$stub`
- * `Foo$_$stub` constructor sends remote creation message to obtain UID

Future directions

- * Replication for fault-tolerance
- * Migration for performance
- * Checkpointing transactions for concurrency control and recovery

XJ: Transactional Java



* Extensions:

- * Language supports open/closed nested transactions
- * Class loader rewrites bytecodes: eg, read/write tracking
- * Calls on XJ runtime injected by compiler and rewriter

XJ reference implementation

- * No changes to JVM
- * Transaction support in XJ runtime:
 - * log-based implementation
 - * log reads, writes, handlers for open nested compensations
 - * "abstract" lock manager

OxJava

- * Orthogonally Extensible Java
- * Declarative expression of extensions like persistence, and those for distribution
- * "ProtoClasses"
 - * describe how to extend representation and behavior of all objects ("primordial")
 - * are abstract: cannot be instantiated, simply express extensions

OxJava rewrites

- * Rewrites "generated" from protocolclass specification, applied to application classes
- * Use same underlying machinery as for distribution and XJ

Pervasive issues

- * Some classes cannot be rewritten:
 - * classes loaded by system class loader
 - * native methods
- * Solve using Twin Class Hierarchy approach (Factor et al, oopsla'04)
- * Wrap non-rewritable classes, rewrite other classes to refer only to wrapper classes

Summary

- * A family of transformation infrastructures permitting extension of Java semantics
- * Distribution
- * XJ language extensions
- * OxJava: meta-level extensions