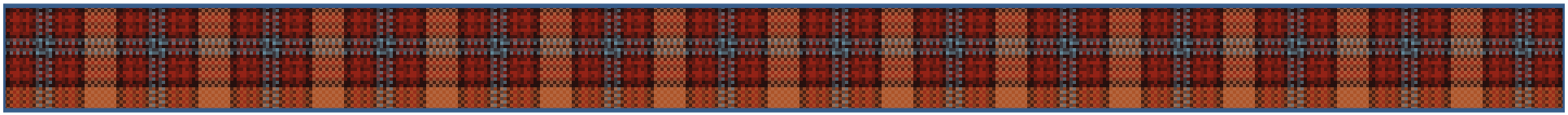
Permission-Based Programming Languages



Jonathan Aldrich Ronald Garcia Mark Hahnenberg

Manuel Mohr Karl Naden Darpan Saini Sven Stork

Joshua Sunshine Éric Tanter Roger Wolff

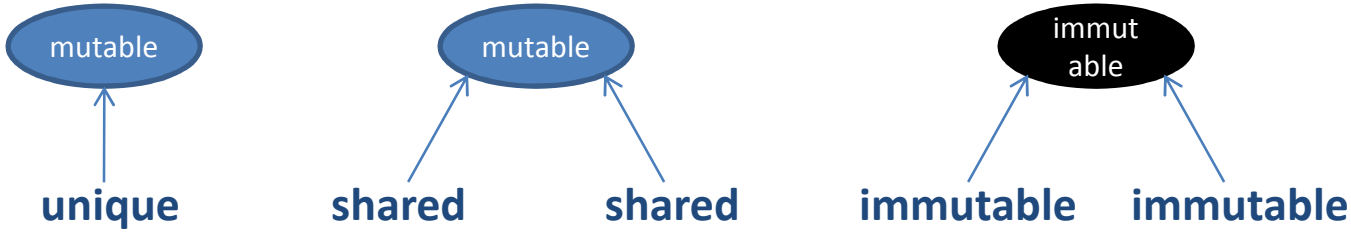
ICSE New Ideas and Emerging Results
May 25, 2011

Background: Permissions

- Permission systems associate every reference with both a type and a permission that restricts aliasing and mutability

```
var unique InputStream stream = new FileInputStream(…);
```

- Some permissions and their intuitive semantics [Boyland][Noble][…]



- Type system checks permission consistency
  - **unique**: no other references to the object
  - **immutable**: no-one can modify the object

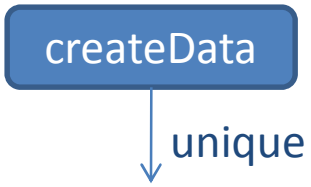
Permission-Based Language

- A language whose type system, object model, and run-time are co-designed with permissions in mind
  - Contrast: prior permission systems layered static permission checking onto existing languages

- Potential benefits
  - Design and encapsulation enforcement
  - Parallel execution
  - Explicit state change in the object model
  - Compile-time and run-time checking

Automatic Parallelization

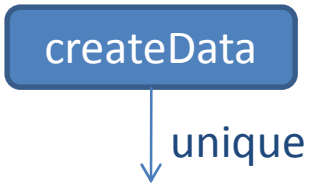
method unique Data createData();




```
val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);
```

Automatic Parallelization

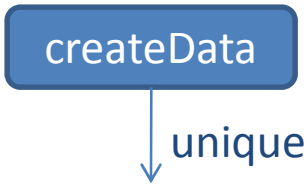
method unique Data createData();



val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);

Automatic Parallelization

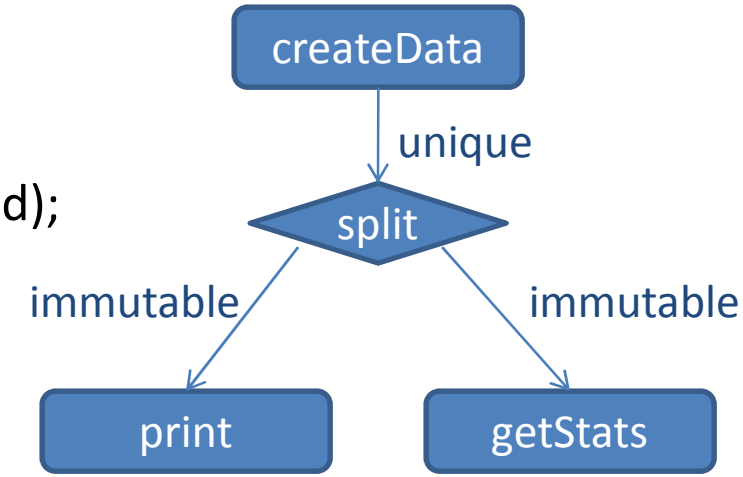
```
method unique Data createData();
method void print(immutable Data d);
method unique Stats getStats(immutable Data d);
```




```
val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);
```

Automatic Parallelization

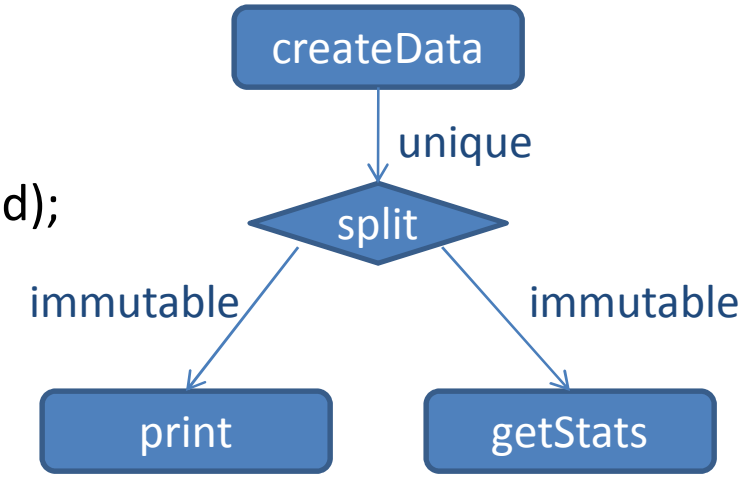
```
method unique Data createData();
method void print(immutable Data d);
method unique Stats getStats(immutable Data d);
```




```
val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);
```

Automatic Parallelization

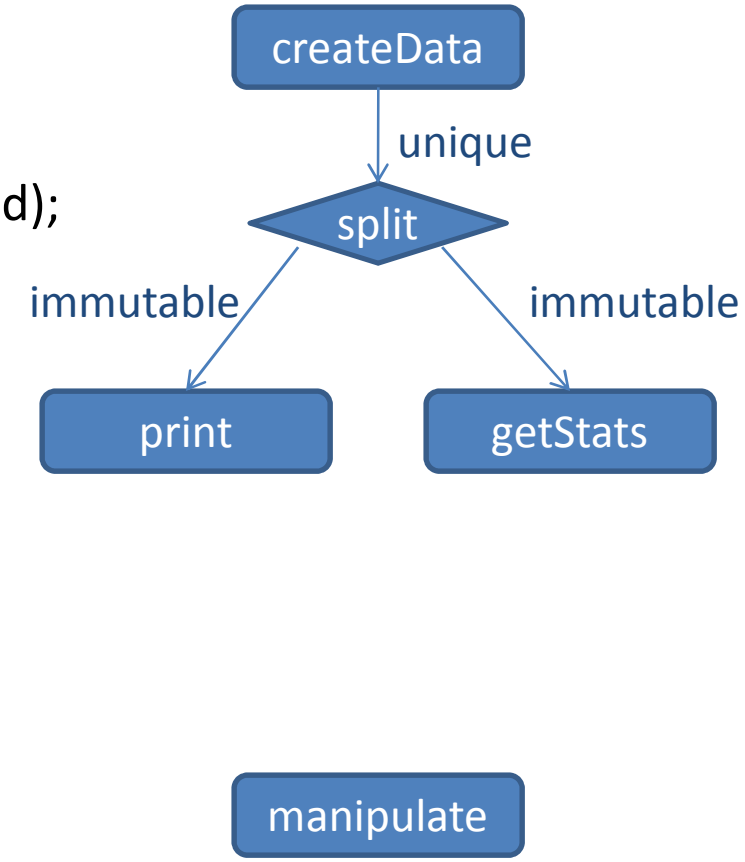
```
method unique Data createData();
method void print(immutable Data d);
method unique Stats getStats(immutable Data d);
```




```
val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);
```

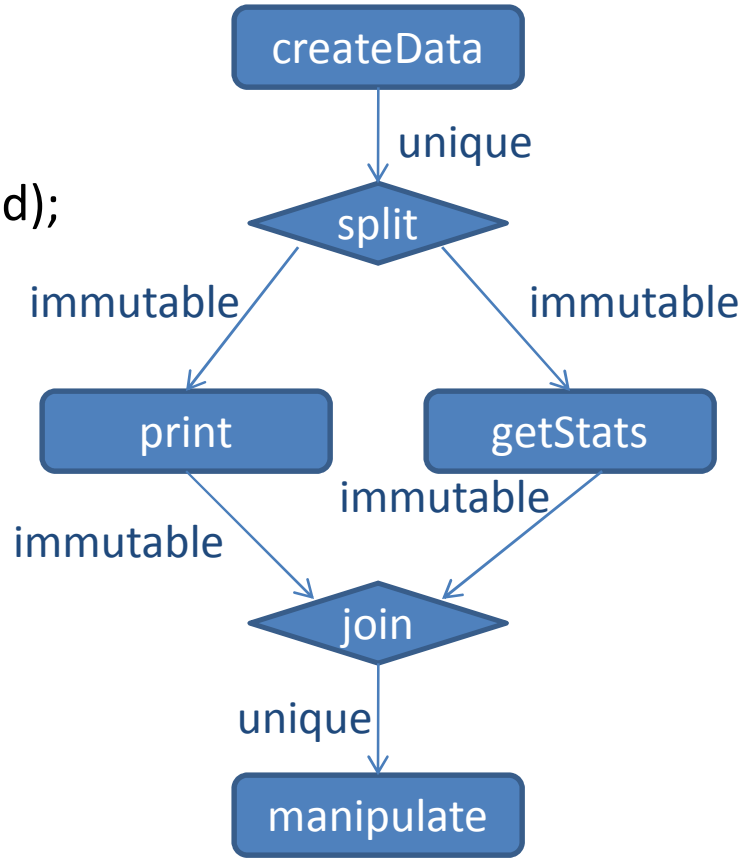
Automatic Parallelization

```
method unique Data createData();
method void print(immutable Data d);
method unique Stats getStats(immutable Data d);
method void manipulate(unique Data d,
                       immutable Stats s);

val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);
```



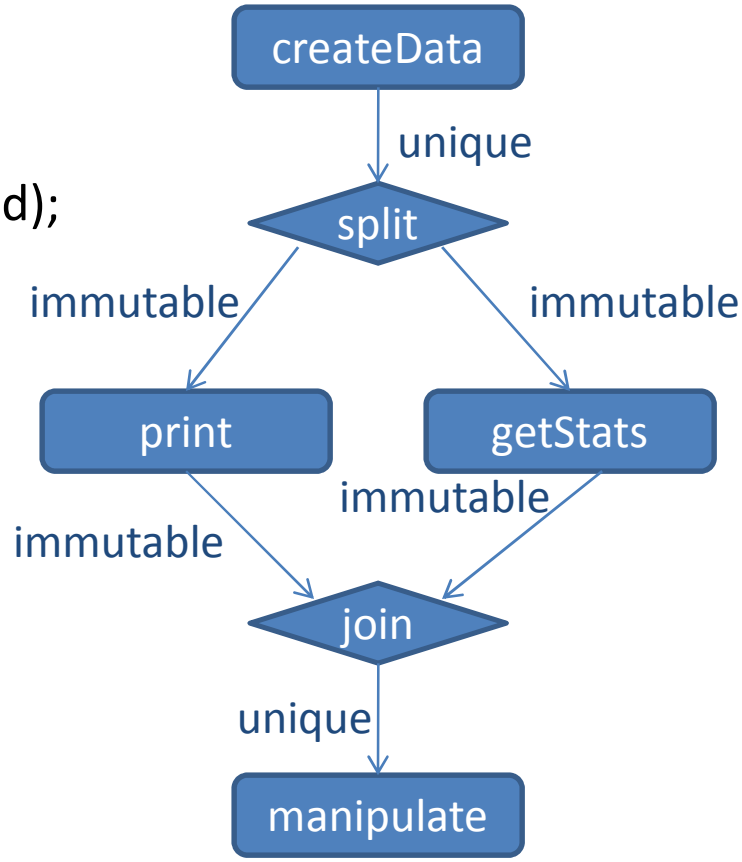
Automatic Parallelization

```
method unique Data createData();
method void print(immutable Data d);
method unique Stats getStats(immutable Data d);
method void manipulate(unique Data d,
                       immutable Stats s);

val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);
```



Automatic Parallelization

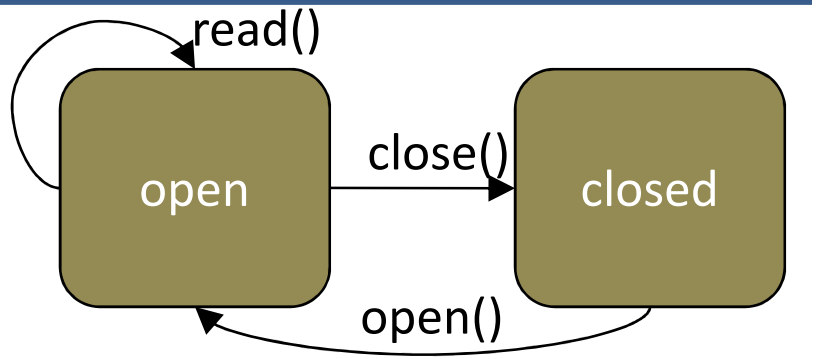
```
method unique Data createData();
method void print(immutable Data d);
method unique Stats getStats(immutable Data d);
method void manipulate(unique Data d,
                       immutable Stats s);

val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);
```



Casts can also be used to recover unique
The runtime checks the cast using reference counts

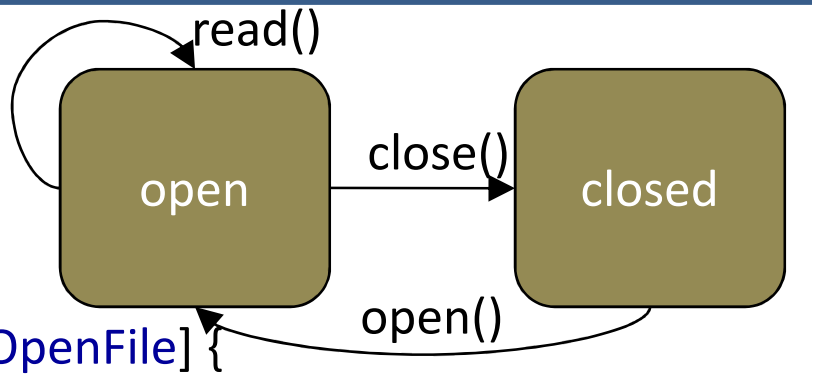
Explicit State Change

```
state File {
    val String filename;
}
```



Explicit State Change

```
state File {
    val String filename;
}
state ClosedFile = File with {
    method void open() [unique ClosedFile>>OpenFile] {



    }
}
```

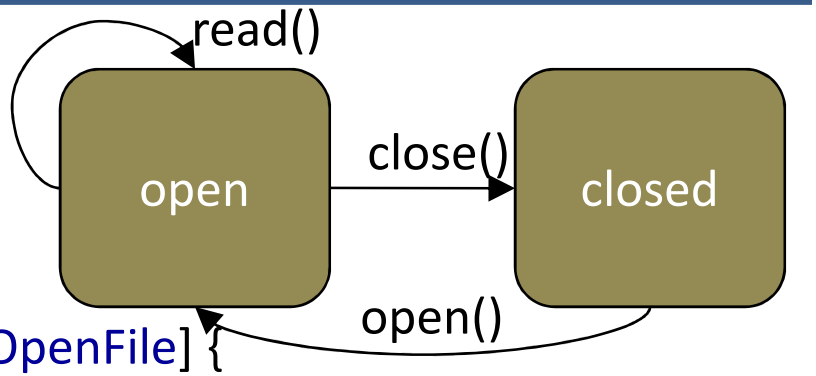


Explicit State Change

```
state File {
    val String filename;
}
state ClosedFile = File with {
    method void open() [unique ClosedFile>>OpenFile] {


    }
}
```

State transition

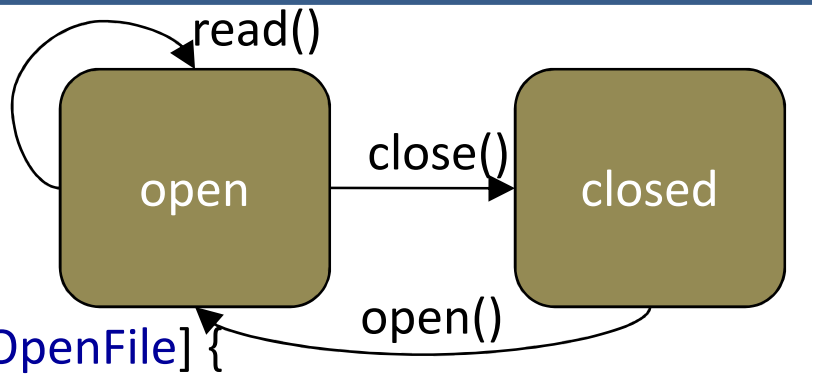
Permission / aliasing info.



Explicit State Change

```
state File {
    val String filename;
}
state ClosedFile = File with {
    method void open() [unique ClosedFile>>OpenFile] {

    }
}
state OpenFile = File with {
    private val CFile fileResource;

    method int read();
    method void close() [OpenFile>>ClosedFile];
}
```

State transition



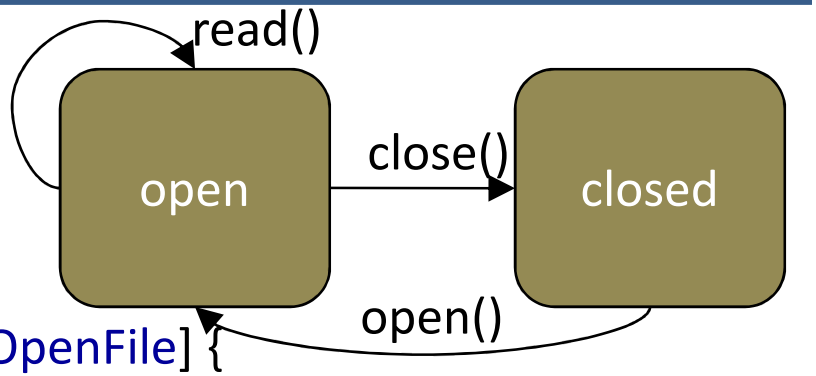
Permission / aliasing info.

New methods, Different representation

Explicit State Change

```
state File {
    val String filename;
}
state ClosedFile = File with {
    method void open() [unique ClosedFile>>OpenFile] {
        this <- OpenFile {
            fileResource = fopen(filename);
        }
    }
}
state OpenFile = File with {
    private val CFile fileResource;

    method int read();
    method void close() [OpenFile>>ClosedFile];
}
```

State transition



Permission / aliasing info.

State change primitive

New methods, Different representation

Plaid: A Permission-Based Language

- Currently exploring these ideas with Plaid
  - First-class abstractions for changing state
  - Naturally safe concurrent execution
  - Practical mix of static & dynamic checking
- Other research directions possible
  - Systems languages: permissions support memory management
  - Security: permissions help control access, information flow
- Status: compiler implemented, typechecker underway
  - Web-based interface available

http://www.plaid-lang.org/