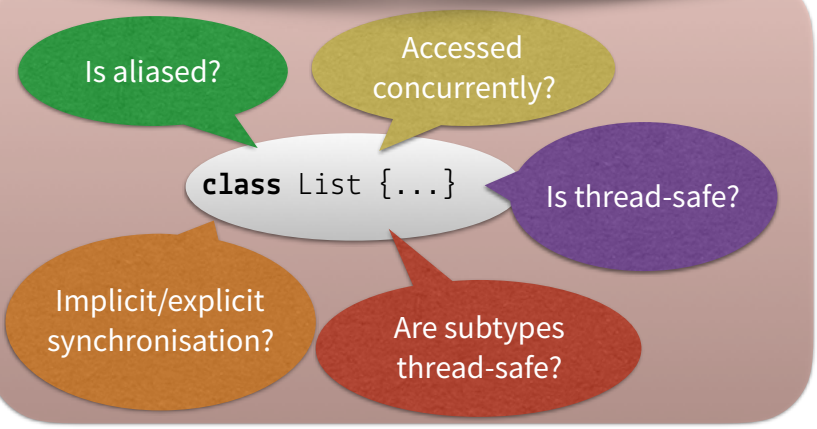
Reference Capabilities for Concurrency Control

UPPSALA UNIVERSITET

UPMARC
Uppsala Programming for Multicore Architectures Research Center

Concurrency imposes many concerns



Same code template safe for different use cases

```
trait Add
  var first : Link
  def add(e : T) : void {...}

class List = Add ⊕ ...
```

mode

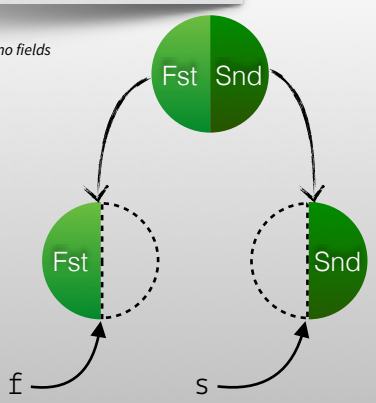
linear	Globally unique
thread	Thread-local
locked	Synchronised
read	Read-only
subordinate	Encapsulated
unsafe	Requires synch.

Composition ⇒ Parallelism

```
class Pair =
  linear Fst ⊗ linear Snd

let f, s = consume p;
finish{
  async{f.set(x)}
  async{s.set(y)}
}
```

Conjunctions share no fields

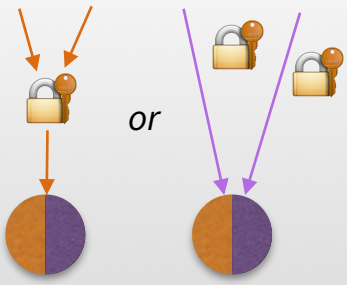


Reuse traits across different concurrency scenarios

```
class ArrayList = unsafe Add ⊕ ...
class Vector = locked Add ⊕ ...
```

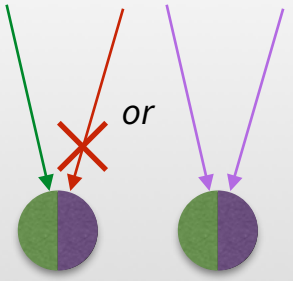
Different combinations of modes express different patterns

readers-writer locks



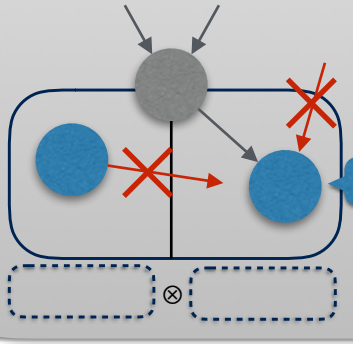
locked ⊕ read

fractional permissions

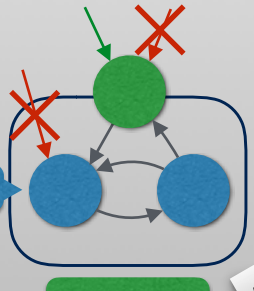


linear ⊕ read

regions and effects



external uniqueness



And more!

✓ No data-races

Traits can assume exclusive access to the underlying object

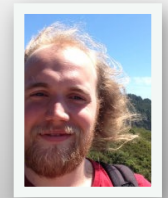
✓ No code duplication

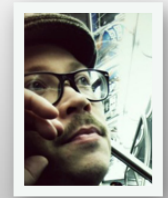
Separate business logic from concurrency concerns

✓ No effect system

References can always be used to the full extent of their types

Elias Castegren Tobias Wrigstad





first.last@it.uu.se